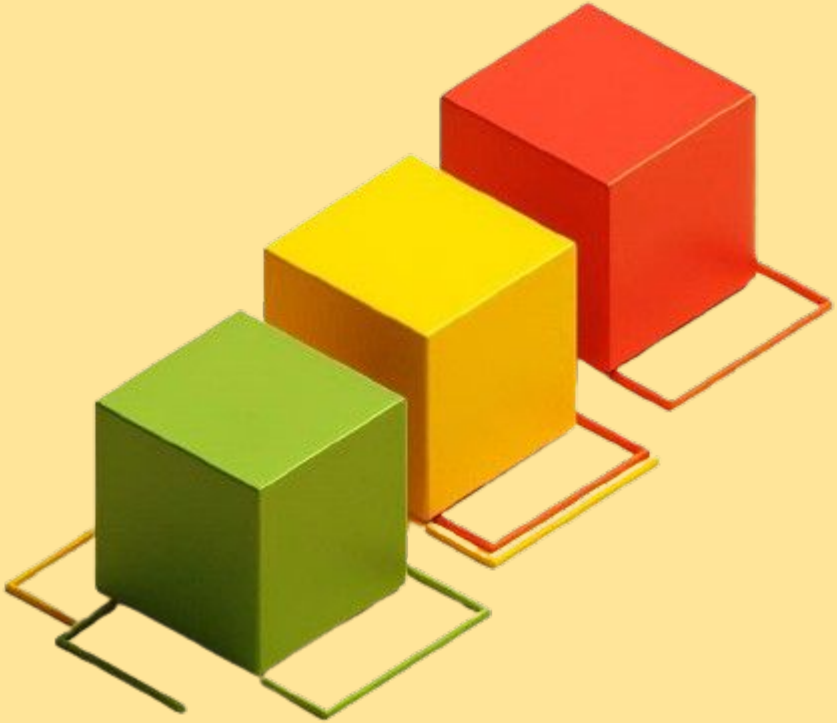
Ethereum Development

by Talat Fakhri



What is a Blockchain?

What is Blockchain and How Does It Work?

- Digital ledger of transactions
- Immutable and tamper-resistant
- Distributed across many nodes
- Uses cryptography for security
- Blocks linked by cryptographic hashes

Centralized vs. Decentralized vs. Distributed Systems

- Centralized: Single point of control
- Decentralized: Multiple authorities, partial control
- Distributed: Multiple nodes with equal authority
- Impact on security, trust, and fault tolerance

Blockchain vs. Traditional Databases

- Blockchain is append-only and immutable
- Traditional DBs allow updates and deletions
- Blockchain uses consensus for data validation
- Databases controlled by administrators
- Use cases differ significantly

Bitcoin vs. Ethereum

- Bitcoin: Digital gold, peer-to-peer cash system
- Ethereum: Programmable blockchain, supports smart contracts
- Bitcoin uses UTXO model; Ethereum uses account-based
- Ethereum supports decentralized applications (dApps)
- Different consensus mechanisms (PoW, moving to PoS for Ethereum)

What Are Smart Contracts?

- Self-executing code on blockchain
- Automate agreements and processes
- Immutable and transparent
- Trigger actions based on predefined conditions
- Enable decentralized applications (dApps)

How Are Smart Contracts Used?

- Decentralized Finance (DeFi) protocols
- Token creation and management (ERC-20, ERC-721)
- Voting and governance systems
- Supply chain transparency
- Automated escrow and legal agreements

Smart Contract Programming Basics

Advantages and Drawbacks of Smart Contracts

- Self-executing with predefined rules
- Trustless: no intermediaries needed
- Immutable once deployed
- Transparent on public blockchains
- Limitations in error handling and upgrades

Layer 1 vs. Layer 2

- Layer 1: Base blockchain (e.g., Ethereum Mainnet)
- Layer 2: Built atop Layer 1 for scalability (e.g., Optimism, Arbitrum)
- Layer 2 reduces gas fees and increases throughput
- Smart contracts can interact across layers

High-Level vs. Low-Level Languages

- High-level: Easier to read/write (Solidity, Vyper)
- Low-level: More granular control (Yul, EVM bytecode)
- Choose based on use case: development speed vs. optimization

Comparing Smart Contract Languages

- Solidity: Most widely used, supported by major tools (Remix, Truffle)
- Vyper: Python-like syntax, more secure but less mature
- Others: Huff, Bamboo (experimental, niche)

Smart Contracts with Solidity

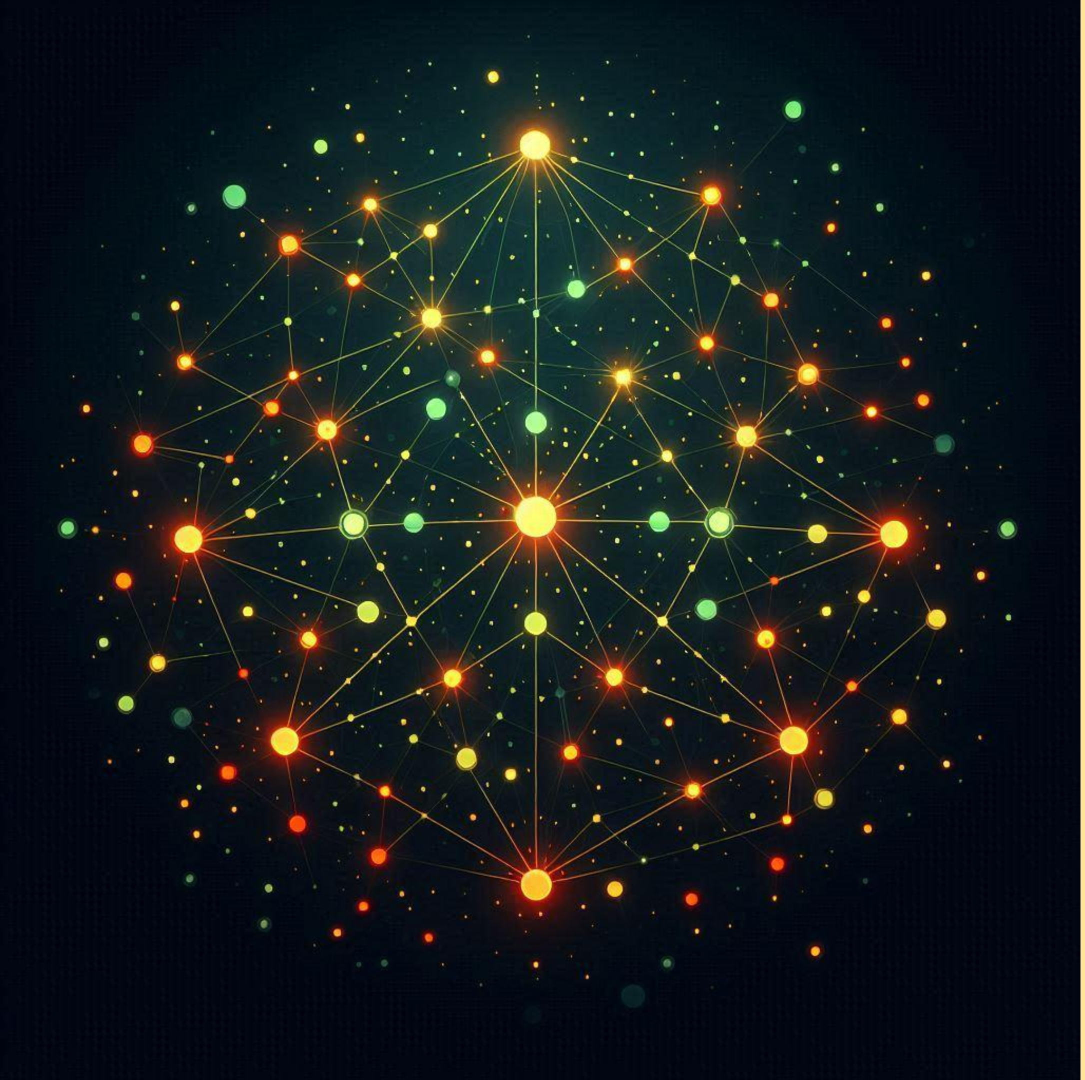
- Contract = Object-oriented class
- Logic, data, access control encapsulated
- Deployed on-chain and interacts via ABI

Anatomy of a Solidity File

- pragma: Compiler version
- import: External dependencies
- contract: Definition
- State variables, constructors, functions
- Modifiers, events, visibility controls

Lab 1: Solidity Hands-On Tasks

- Declare variables: `uint`, `string`, `address`, `bool`
- Use `public`, `private`, and `internal` visibility
- Define constructors to initialize state
- Create basic setter and getter functions



Decentralized Information and Web3

Blockchain Access Structures and Architectures

- Remote Blockchain Nodes vs. Local Blockchain Nodes
- Different ways to connect and interact with blockchains
- Trade-offs in control, latency, and resource use

Blockchain Access vs. Centralized RESTful API

- Blockchain access is decentralized and permissionless
- RESTful APIs are centralized and controlled by single entities
- Blockchain APIs provide data authenticity but with different trust models

Understanding Web3.js API

- Web3.js is the primary JavaScript library to interact with Ethereum
- Enables communication with smart contracts and nodes
- Facilitates transaction creation, signing, and querying blockchain data

Understanding Transactions and Consensus

- Transactions are the fundamental operations that change blockchain state
- Consensus algorithms ensure all nodes agree on the blockchain state
- Proof of Work, Proof of Stake, and other consensus mechanisms

Private Keys, Public Keys, and Signatures

- Private keys authorize transaction signing and access to funds
- Public keys identify accounts on the blockchain
- Digital signatures verify transaction authenticity without exposing private keys

Understanding Privacy on Public Blockchains

- Blockchain data is transparent and visible to all participants
- Pseudonymity vs. true anonymity
- Techniques like zero-knowledge proofs and mixers to enhance privacy

Understanding the Architecture of KeyStores: MetaMask or MIST

- KeyStores manage private keys locally, often encrypted with a password
- MetaMask and MIST provide user-friendly wallets and interfaces
- Integration with browsers and dApps for seamless user experience

Lab Tasks Overview (Lab 2 – Ropsten Test-Ether and MetaMask)

- Installing and configuring MetaMask wallet
- Obtaining test Ether on the Ropsten testnet
- Using block explorers to trace transactions
- Understanding Infura’s role as a remote node provider

Basics of Ethereum and EVM

Ethereum Denominations

- Wei is the smallest denomination of Ether (1 ETH = 10^{18} Wei)
- Common units: Wei, Gwei (10^{9} Wei), Ether
- Gas prices are often denominated in Gwei
- Understanding denominations is essential for gas management and transaction cost estimation

Understanding EVM and the ABI Interface

- EVM = Ethereum Virtual Machine, a sandboxed environment
- It executes smart contracts in a deterministic way
- ABI (Application Binary Interface) connects off-chain code with smart contracts
- ABI encoding and decoding governs how data is passed in function calls

Calls vs. Transactions

- Calls are read-only operations; free and don’t alter blockchain state
- Transactions are write operations; they consume gas and change state
- Only transactions are mined and confirmed by the network
- Choose calls for queries; transactions for state changes

Concurrency and Events

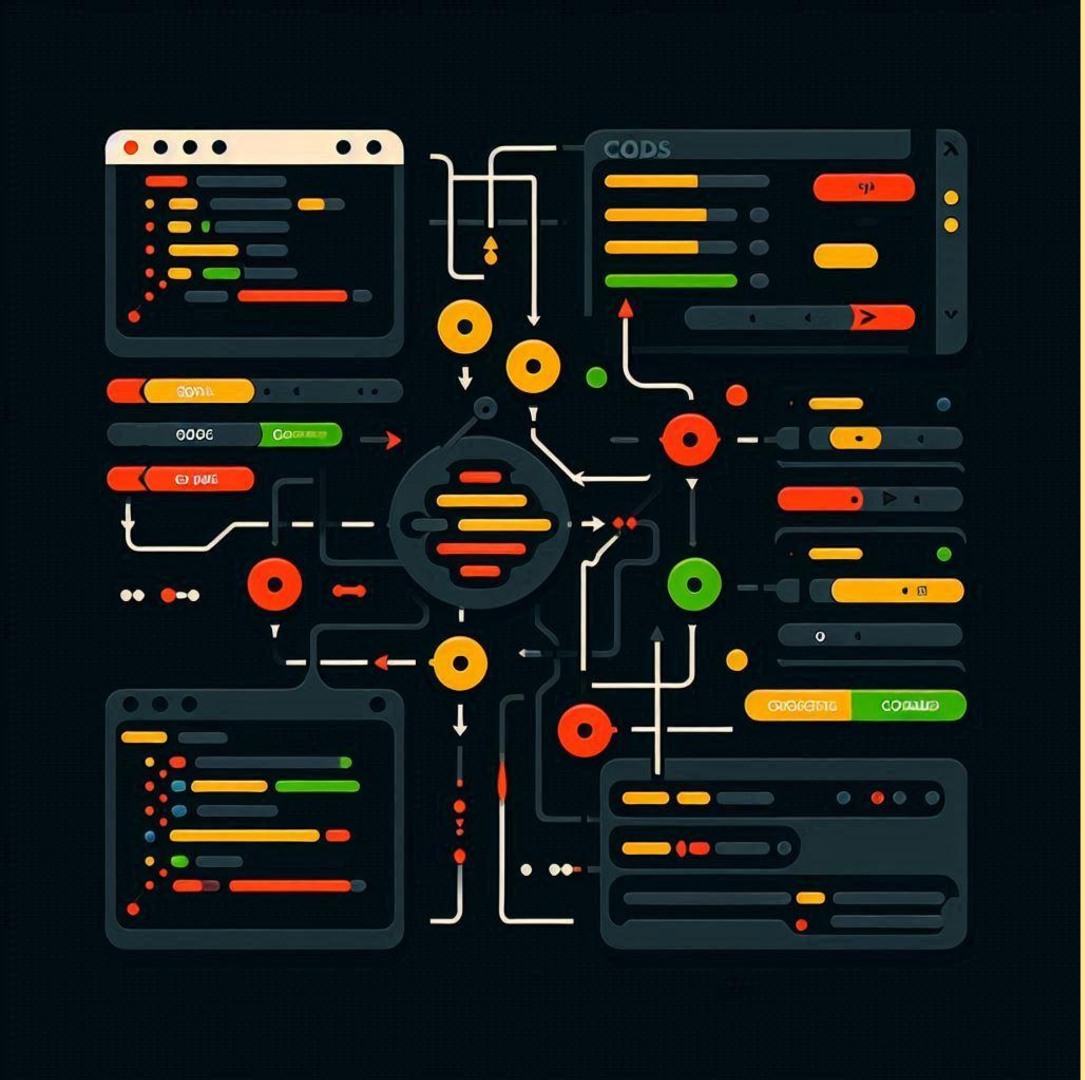
- Ethereum doesn’t support traditional multi-threaded concurrency
- Instead, it uses a single-threaded transaction model
- Events are used for asynchronous logging and external notifications
- Events don’t affect state but help index and track changes

Use Cases of Events

- Logging key actions (e.g., funds transferred, roles assigned)
- Enabling real-time frontend updates
- Auditing and analytics of blockchain activity
- Integration with off-chain services like The Graph

LAB TASKS — Web3JS Operations & Events

- Install and use Ganache for local Ethereum simulation
- Connect frontend to Ethereum via Web3.js
- Use Infura for remote node access
- Define events in Solidity
- Listen and respond to events using JavaScript



Solidity Advanced

Understanding Functions, Mappings, and Structs

- Functions are the core logic blocks of smart contracts.
- Mappings allow for key-value pair storage—ideal for tracking balances or permissions.
- Structs group multiple variables—useful for modeling entities like users or assets.

When to Use Modifiers

- Modifiers help enforce access control and preconditions.
- Improve readability by abstracting repetitive validation logic.
- Can be stacked for complex business rules.

Libraries vs. Inheritance

- Libraries are for reusable logic with no state.
- Inheritance is for extending behavior and shared state.
- Choose based on statelessness and security requirements.
  .

LAB TASKS — Modifiers, Mappings, Structs, and Inheritance

- Lab 5: Implement custom modifiers like onlyOwner and onlyWhitelisted.
- Lab 6: Use mappings to store user data; define and deploy structs.
- Lab 7: Create a base contract and inherit it in child contracts to demonstrate reusability.

  .

Deployment and Costs

Understanding Deployment and Costs

- Deployment is the bridge between development and blockchain execution
- Smart contracts are immutable post-deployment (unless upgradeable patterns are used)
- Poor deployment practices lead to high costs or security flaws

Development and Deployment Cycles

- Develop → Test → Audit → Compile → Deploy
- Use testnets for validation (Sepolia, Goerli)
- Separate staging and production environments

Solidity Compilation and Deployment

- Compiled contracts generate bytecode and ABI
- Remix, Hardhat, Foundry, Truffle support compilation
- Deployment = sending bytecode via a transaction to the blockchain

Gas and Gas Costs

- Gas = execution unit in Ethereum
- Every operation (SSTORE, CALL, CREATE) has a cost
- Optimizing code reduces deployment & runtime gas costs
- Use tools like Remix’s Gas Analyzer, Tenderly

Upgradeability and Data Migration Techniques

- Smart contracts are immutable by default
- Use proxy patterns (Transparent, UUPS) for upgrades
- Migrate data between versions via migration scripts
- Use OpenZeppelin’s upgradeable contracts

Understanding the Moving Parts

- Compiler (Solc): Translates Solidity into bytecode
- Blockchain (EVM): Executes deployed code
- API/ABI: Interface for external apps (Web3.js, Ethers.js)
- Keystore: Secures private keys for deployment

Mining, Proof of Work vs. Proof of Authority

What is Mining in PoW?

- Mining = Block creation + consensus mechanism
- Miners solve computational puzzles (hashing)
- First to solve adds the block to the chain
- Incentivized through block rewards and transaction fees

How Blocks Are Generated

- Transactions collected in a mempool
- Miners select, order, and hash transactions
- A valid block must meet the target hash (difficulty)
- Block added, chain extended, and propagated

PoW vs. PoA vs. PoS

- PoW: Secured by computational effort (Bitcoin, Ethereum pre-2.0)
- PoA: Authority-based, fast but centralized (used in private chains)
- PoS: Secured by staked capital, energy-efficient (Ethereum 2.0, Cardano)

Go-Ethereum (Geth) and Ganache/TestRPC for Local Development

- Geth: Full-featured Ethereum client (run full node or private networks)
- Ganache: Lightweight, fast blockchain emulator
- Ideal for smart contract development and testing
- Enables account creation, mining simulation, and state resets

Private Blockchains vs. Public Blockchains

- Public: Open to all, decentralized, trustless (e.g., Ethereum, Bitcoin)
- Private: Permissioned, centralized control (e.g., Hyperledger, Quorum)
- Trade-offs: Transparency vs. speed, Trust vs. control, Openness vs. privacy
- Hybrid chains also emerging (e.g., Polygon Supernets, Avalanche Subnets)

Current Problems, Solutions, Outlook, Serenity

Ethereum Now and Ethereum Future

- Ethereum has evolved through several major phases: Frontier, Homestead, Metropolis, and now Beacon Chain
- Present Ethereum (as of 2025) operates fully under Proof of Stake (PoS)
- Ethereum's vision: the world computer for decentralized applications
- Future: Scalability, sustainability, and decentralization via Serenity (Ethereum 2.0)

Where We Are At with Ethereum

- Ethereum has transitioned from PoW to PoS (The Merge)
- The Homestead phase solidified Ethereum’s foundation
- PoW era faced issues: high energy costs, centralization of mining

Where Ethereum Is Heading – Serenity

- Serenity (Ethereum 2.0) is a multi-phase upgrade
- Final goal: scalable, secure, and sustainable blockchain
- Includes upgrades like sharding and state execution changes

Where We Are At with Ethereum

- Ethereum has transitioned from PoW to PoS (The Merge)
- The Homestead phase solidified Ethereum’s foundation
- PoW era faced issues: high energy costs, centralization of mining

PoW to PoS – The Energy Revolution

- PoW required mining; PoS requires staking
- PoS uses economic penalties and rewards to maintain security
- PoS is 99.95% more energy efficient

Sharding – Unlocking Scalability

- Sharding divides the Ethereum network into smaller chains
- Each shard processes its own transactions and smart contracts
- Enables parallel processing: thousands of TPS

Recommended Newsletters and Community Groups

- Week in Ethereum News – curated updates from the ecosystem
- Bankless – deep dives into DeFi, DAOs, and the future of money
- EthHub – knowledge base and weekly newsletter
- r/ethereum & Ethereum Magicians – active communities
- Discord: ETH Staker & Devcon channels – validator and dev discussions

Working in Teams, Testing and Versioning

Understanding What Truffle Is

- Truffle is a development framework for Ethereum.
- It includes a compiler, deployment tool, and testing suite.
- Offers project scaffolding and smart contract abstraction.
- Integrates with Ganache for local testing

Comparison to Embark

- Embark also supports smart contract development and deployment.
- Embark emphasizes decentralized storage and front-end integration.
- Truffle is more mature, with broader community adoption.
- Truffle has stronger testing and migration support.

How to Manage Code for Teams

- Use Git with clear branching strategy (e.g., GitFlow).
- Share Truffle project structure and enforce linting.
- Store contract ABIs and artifacts in a shared build/ directory.
- Document migrations and network configurations clearly.

Understanding Migrations

- Migrations are scripted deployments of smart contracts.
- They track deployed contracts using Migrations.sol.
- Ensure deterministic, repeatable deployments across environments.
- Written in JavaScript within the migrations/ folder.

Understanding Unit-Testing with Truffle

- Truffle supports tests in JavaScript and Solidity.
- Built on Mocha (test runner) and Chai (assertion library).
- Supports contract deployment, method calls, and error testing.
- Enables test coverage and CI integration.

IPFS and distributed File-Storage

What is IPFS (InterPlanetary File System)

- Peer-to-peer hypermedia protocol
- Content-addressable: files retrieved by hash
- Decentralized storage and sharing
- Replaces HTTP with distributed delivery.

IPFS, Filecoin, Swarm, Sia, Storj

- IPFS: Data distribution, no built-in incentive
- Filecoin: IPFS-compatible, incentivized with FIL token
- Swarm: Ethereum-native, incentivized via BZZ token
- Sia: Renter-host model with Siacoin payments
- Storj: Decentralized cloud for enterprise, with STORJ token

QUESTIONS