Ethereum Specific Concepts

Chapter – 4

Fall 2025

Middle Tennessee State University

Part - 1

Ethereum-Specific Concepts

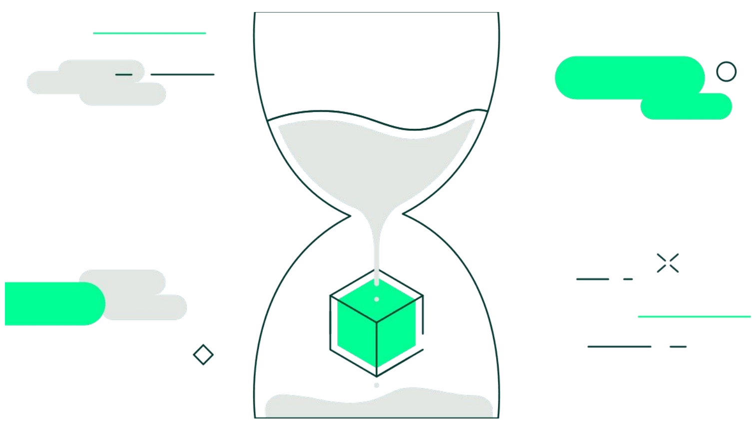
- Block Creation in Ethereum
- Block Time
- Transactions (TXN) in Ethereum
- Gas Cost
- Transaction Cost
- Gas Limit and Gas Used
- Gas Price

Block Creation in Ethereum

- Creation of new block in **Old Ethereum**
  - Proof of Work Consensus
  - Mining Competition

- **Ethereum 2.0** Transition
  - Creation/Proposal of a new block – Proof of Stake (PoS)
  - 15 Sept 2022 – from PoW to PoS
  - Ethereum's energy usage has been reduced by 99%

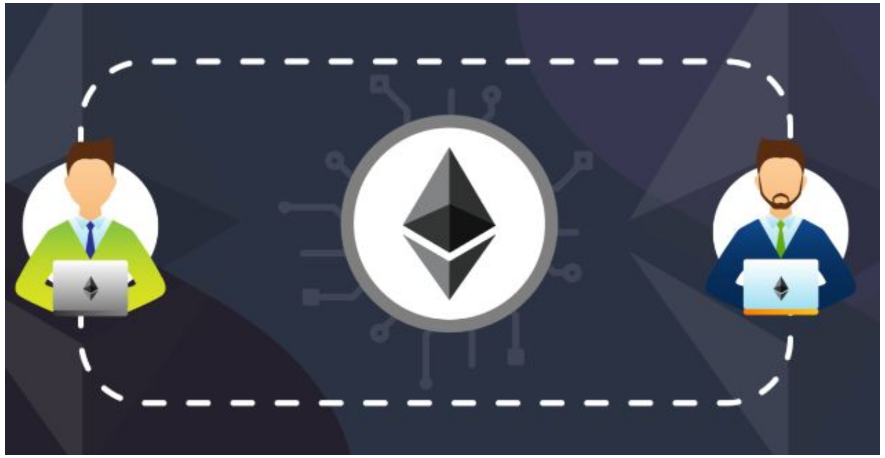
Block Time in Ethereum



- Block time
  - Average time it takes to create a new block in Ethereum
  - Significant role in determining the speed and efficiency of transactions and smart contract executions.
  - Block time -> Around 12 seconds
  - Time is divided up into twelve second units called “slots”
  - Significantly faster than block times in networks like Bitcoin, which have block times of approximately 10 minutes, and Litecoin which has approx. 2.5 minutes

- Why is Block Time important?
  - critical metric for measuring the responsiveness and efficiency of a blockchain network.
  - directly affects the user experience

Transactions (Txn/Tx) in Ethereum



- Foundational element of any Blockchain
- Movement of digital assets (Ether or Tokens)
- Also, execution of smart contracts on the Ethereum network

Transactions (Txn/Tx) in Ethereum – Contd…

Components of a Transaction:

- **Sender**: Address of the account initiating the txn
- **Recipient**: Destination address to which Ether or Tokens are sent
- **Value**: Amount of Ether or tokens being transferred in the transaction
- **Nonce**: A unique number assigned to each transaction, preventing double-spending from the same account
- **Gas Limit**: Maximum amount of gas units that can be used for the transaction
- **Gas Price**: The price in Ether per gas unit that the sender is willing to pay for transaction execution.
- **Data, Signature**

Transactions (Txn) in Ethereum (Examples) – Contd…

E.g.,

1. *Ether Transfer Between Users*

- **Sender**: `0xUserAlice`
- **Recipient**: `0xUserBob`
- **Value**: 1 Ether
- **Gas Limit**: 21,000 units of gas (the default gas limit for simple Ether transfers)
- **Gas Price**: 10 Gwei (0.00000001 Ether per gas unit)
- **Data**: (Empty, as it's a simple Ether transfer)

E.g.,

2. *DApp Interaction*

- **Sender**: `0xUserCharlie`
- **Recipient**: Dapp Smart Contract (`0xSmartContract`)
- **Value**: 0.5 Ether
- **Gas Limit**: 150,000 units of gas
- **Gas Price**: 20 Gwei (0.00000002 Ether per gas unit)
- **Data**: function call to place a bid in a decentralized auction within the Dapp

  Function: **`transfer(address _to, uint256 _value)`**

Transaction Cost

- Also known as Transaction Fee, Network Fee
- Important metric in Blockchain
- **Total expense associated with executing a transaction on the Ethereum blockchain**
- *Transaction Cost (in Ether) = Gas Used * Gas Price*

Transaction Time

- Another crucial metric in blockchain networks
- **Time taken for a transaction to be initiated, confirmed, and recorded on the blockchain**
- Transaction Time Components:
  - Initiation Time
    - Begins when a user initiates a transaction from their wallet
  - Confirmation Time
    - The period from transaction initiation to when it is included in a block
    - Shorter confirmation times are achieved with higher gas price
- Factors influencing Transaction (Txn) Time:
  - Network Congestion
    - During periods of high activity, more transactions compete for limited block space
    - Longer transaction times as miners prioritize transactions with higher gas fees

Gas Cost and Gas Price (Contd…)

What is **Gas** in Ethereum Blockchain?

- a unit of computational work required to perform actions on the Ethereum blockchain
- Example, car running with gas
- Every operation, from sending Ether to executing a smart contract, consumes gas

- Gas – Computational effort required to perform actions
  - Sending Ether
  - Deploying or executing smart contracts on Blockchain
  - Interacting with Dapps

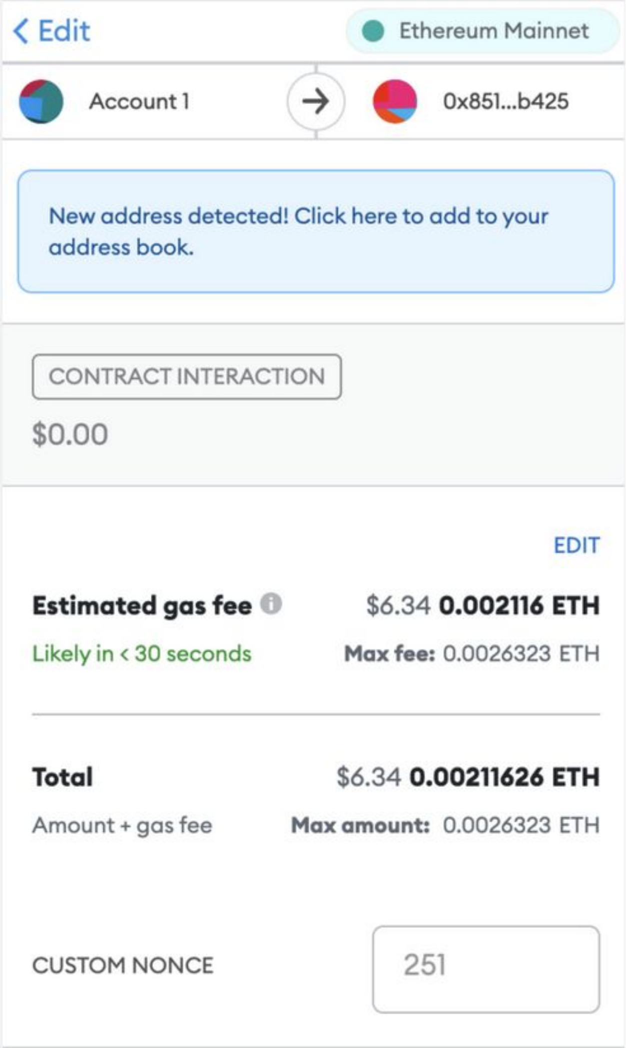
Gas Cost and Gas Price (Contd…)

Gas Cost/Fee:

- Also known as Gas Fee
- **Total amount of gas used** in an Ethereum transaction or smart contract execution
- OR, In other words, the **total cost of computational work** performed during a transaction or smart contract execution

- Measured in units of gas (not Ether), but the total fee is converted and paid in Ether for transaction settlement
- Determined by the complexity of the operation
- Every operation in Ethereum (like addition, storing data, calling a function) has a fixed gas cost defined by the Ethereum Virtual Machine (EVM)
- Essential for users, developers, and anyone interacting with the Ethereum network

- https://etherscan.io/gastracker

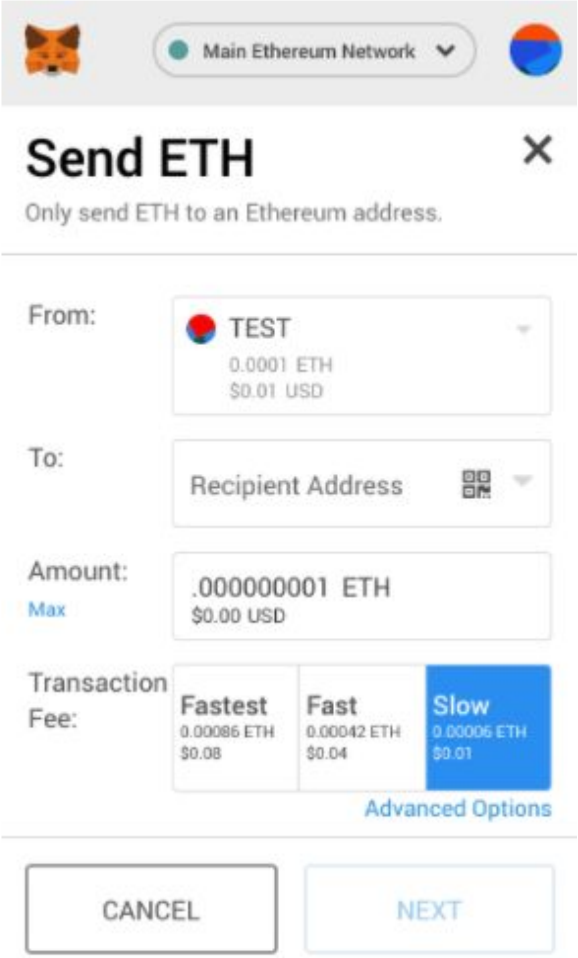


MIDDLE TENNESSEE STATE UNIVERSITY

Gas Cost and Gas Price (Contd…)

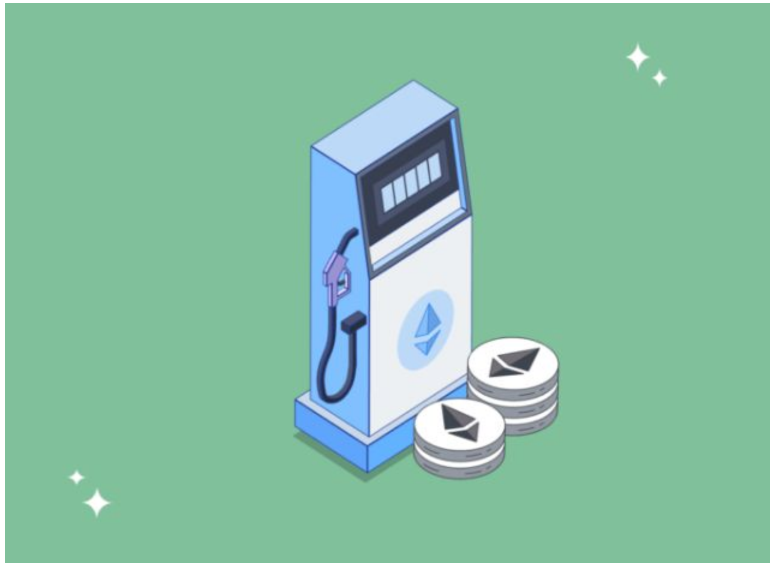
Gas Price:

- **Price in Ether (ETH) that a user is willing to pay for each unit of gas used in a transaction or smart contract execution**
  - setting a bid for transaction speed
  - If you are willing to pay more, miners or validators will likely pick your transaction first
- Gas price is denominated in Gwei, a subunit of Ether
- *1 Gwei equals to 0.000000001 ETH*
- When network demand is high, gas price rises
- When network is quiet, gas price drops



Gas Limit and Gas Used

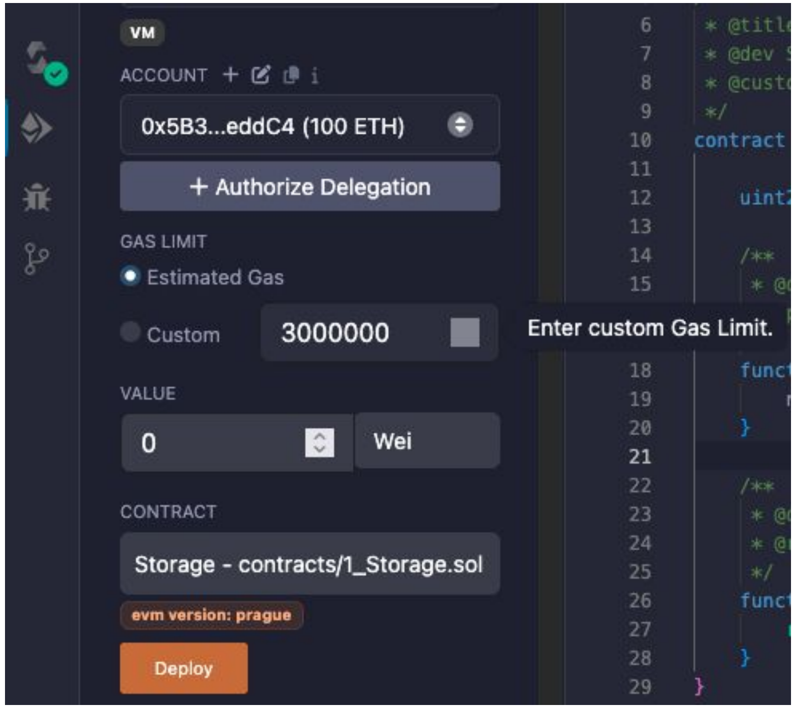
- Gas Limit:
  - maximum amount of gas allocated for a transaction or smart contract execution
- Gas Used:
  - actual amount of gas consumed during a transaction or contract execution
- Transactions with **`Gas Used ≤ Gas Limit`** are successful; but exceeding the limit results in failure
- Users pay for Gas Used, not the full Gas Limit
- E.g.,
  - User initiates a smart contract interaction with a gas limit of 150,000
  - The operation consumes 75,000 gas
  - The user is charged for the 75,000 gas used, not the full 150,000 gas limit



Gas Limit and Gas Used – Contd…

- What kinds of problems might arise if transactions didn't have a predefined gas limit?

  - To prevent infinite computation (safety)
    - A contract with `while(true){}` would run forever, unless gas runs out, forcing it to halt safely
  - To protect against spam and DoS attacks (security)
    - Attackers could flood the blockchain with heavy, complex transactions to overload nodes
  - To give users cost control (financial safety)
    - Prevents unexpected charges when executing complex or buggy contracts
  - To keep blocks manageable (performance & fairness)
    - Each block also has a block gas limit (≈ 30 million gas), preventing blocks from becoming too large to process efficiently



Factors influencing Gas Cost

- **Operation Complexity**
  - Complex operations (e.g., executing complex smart contracts) consume more gas than simple ones (e.g., transferring Ether)
- **Data Storage**
  - Storing data on the blockchain incurs gas costs
- **Gas Price**
  - Gas cost is directly proportional to the gas price set by the user
  - Higher gas prices lead to faster transaction confirmation but increase the total cost

Gas Cost, Gas Price, Txn Cost (Example)

- *If 1 transaction consumes **40,000 units of gas** at a **gas price of 20 Gwei per gas unit**, you can calculate the total cost of the transaction using the following formula:*
  - ***Total Cost (in Ether) = Gas Used * Gas Price***
  - *Gas Used: 40,000 units of gas*
  - *Gas Price: 20 Gwei per gas unit*

  - *<u>Step 1:</u> Convert the gas price from Gwei to Ether*
    - *Gas Price (in Ether) = 20 Gwei / 1,000,000,000 = 0.00000002 Ether*
  - *<u>Step 2:</u> Calculate the total (transaction) cost*
    - *Total (transaction) Cost (in Ether) = 40,000 gas * 0.00000002 Ether/gas = 0.0008 Ether*

Conversions for 1 Ether (ETH) to various denominations – No need to memorize this!

- 1 Ether (ETH) = 1,000,000,000,000,000,000 Wei
- 1 Ether (ETH) = 1,000,000,000,000,000 Kwei (Kilowei or Babbage)
- 1 Ether (ETH) = 1,000,000,000,000 Mwei (Megawei or Lovelace)
- 1 Ether (ETH) = 1,000,000,000 Gwei (Gigawei or Shannon)
- 1 Ether (ETH) = 1,000,000 Szabo (Microether)
- 1 Ether (ETH) = 1,000 Finney (Milliether)
- 1 Ether (ETH) = 1 Ether (Main Unit)

- *Wei* is the smallest indivisible unit of Ether (like cents in a dollar)
- *Gwei* is most commonly used for gas prices: 1 Gwei = 0.000000001 ETH
- Gas prices and fees are measured in Gwei, not ETH

Part - 2

What is a Smart Contract?

- A smart contract is a self-executing piece of code
- Deployed/stored on the blockchain
- It runs automatically once predefined conditions are met
- No middlemen, central authority or human approval required

- Nick Szabo proposed the idea of smart contracts in 1994
  - Over a decade before Ethereum made them real!

- Key Characteristics:
  - Immutable (once deployed, can't be changed)
  - Transparent (everyone can inspect its code)
  - Autonomous (runs by itself when triggered)

What is a Smart Contract? – Contd…

- A smart contract lives at a specific blockchain address and can hold Ether or tokens

- Users send transactions to interact with it
  - These transactions execute functions inside the contract

- For example, a **flight-delay insurance** contract pays customers automatically if the airline's delay API reports a delay over 3 hours
  - No claims officer, no paper form, i.e., pure code enforcement

Ethereum Virtual Machine (EVM)

- The EVM is Ethereum's global runtime environment
  - A sandboxed virtual machine that executes all smart contract bytecode in exactly the same way on every node
- Deterministic Computation
  - Every node receives identical inputs and must produce the same output so that all copies of the blockchain agree on state changes
- Bytecode Execution
  - Solidity code compiles to EVM opcodes (e.g., `ADD, SSTORE, CALL`)
  - The EVM executes these instructions step by step, charging **gas** for each operation
- Example
  - When you call `transfer()` on a token contract, every EVM node runs the same bytecode to deduct tokens from your balance and add them to the receiver's account

Life Cycle of a Smart Contract

1. **Write:** Developer writes Solidity source code defining logic and state variables
2. **Compile:** The Solidity compiler (solc) converts it to EVM bytecode and generates the ABI (Application Binary Interface)
3. **Deploy:** Deployment is a special transaction that stores bytecode on the Ethereum blockchain
4. **Interact:** Users send transactions calling functions using the ABI; each execution changes contract state and costs gas
5. **Terminate (Optional):** `selfdestruct(address)` can delete a contract and send remaining Ether to a target address

Some large DeFi contracts exceed 25 KB of bytecode and cost hundreds of dollars to deploy during peak gas prices!

Programming with Solidity – Basics

- High-level language that resembles C++ and JavaScript
- Solidity is used for writing Ethereum smart contracts

```
pragma solidity ^0.8.0;
contract Hello {
    string public greeting = "Hello Ethereum!";
    function getGreet() public view returns (string memory) {
        return greeting;
    }
}
```

Programming with Solidity – Basics – Contd…

```
pragma solidity ^0.8.0;
contract Hello {
    string public greeting = "Hello Ethereum!";
    function getGreet() public view returns (string memory) {
        return greeting;
    }
}
```

- **Visibility Modifiers:** `public, private, internal, external`
- **Function Types:** `view` (read-only), `pure` (no state access), `payable` (receives Ether)

Programming with Solidity – Basics – Contd…

- **Visibility Modifiers**: Control who can access your functions and variables:
  1. **Public**:
     - Accessible from anywhere: inside the contract, from other contracts, and externally
     - Automatically creates a getter function for state variables
     - For e.g., `string public greeting` (in the last slide) anyone can read this
  2. **Private:**
     - Only accessible within the current contract
     - Not visible to derived contracts or external calls
     - Most restrictive option
  3. **Internal:**
     - Accessible within the current contract and derived contracts (inheritance)
     - Not accessible externally, use when you want a contract and its child contracts to access it
     - Default for state variables
  4. **External:**
     - Only callable from outside the contract, expects data to come from outside
     - More gas-efficient for external calls

- **Note**: Use the most restrictive visibility possible for security!

Programming with Solidity – Basics – Contd…

- Function types:
  1. `view` (read-only)
     - Reads blockchain state but doesn't modify it
     - Free to call when querying
     - For e.g., our `getGreet()` function from previous slide
  2. `pure` (no state access)
     - Does not read or write blockchain state
     - Just performs calculations
     - For e.g.,

```
function add(uint a, uint b) public pure returns (uint) {
    return a + b;  // Only uses parameters, no state variables
}
```

Programming with Solidity – Basics – Contd…

`Function types:`

3. `payable` (receives Ether)

- Can receive Ether or cryptocurrency along with the function call
- Required if you want to accept payments
- Without payable, attempting to send Ether will fail
- For e.g.,

```
function donate() public payable {
    // Can receive Ether
}
```

Ethereum Addresses

- Addresses in Ethereum
  - Every entity on Ethereum has an address
    - Like a bank account number or email address for blockchain
    - Represents either a person or a smart contract on Ethereum network
    - a unique 160-bit (20 bytes) identifier starting with **`0x`** (hexadecimal notation)

1. <u>Externally Owned Accounts (EOAs)</u>
   - Controlled by private keys, i.e., humans or wallets (MetaMask, Coinbase wallet, etc)
   - Can initiate transactions and has a balance of ether
   - No code associated with these kind of accounts
   - E.g, your personal bank account (analogy)
   - E.g.,: `0xAb5801a7D398351b8bE11C439e05C5B3259aec9B` is Vitalik Buterin's (Ethereum's founder) known address

2. <u>Contract Accounts</u>
   - Controlled by code, not a person
   - No private key which means no person controls it. Hence, can only react
   - Code is the law
   - Bug in the code? Funds can be at risk! Audits are crucial

Announcement/Reminder: Project Plan (Phase 1)

- Project Completion (3 Phases) + Final Report + Project Presentation: 15% + 10% + 5% = 30%

- **Project Plan (Phase 1):** Problem Motivation, Literature Survey, and Proposed Solution (**5% of Project Completion**)

- Submission deadline of Phase 1: **Tuesday, October 28, 2025, 11:59 AM**

- **Note:** You can work either with same or different teammate with a group size of 2.
    - You may choose same or different topic from Paper Presentation.

Announcement/Reminder – Contd…

- What do you need to have in your Project Plan Draft?
  1. **Title of your project**
  2. **Problem Definition and Motivation**
     - What is the current issue and problem in your chosen area?
     - Why is it important to resolve this issue you have chosen?
     - Include picture to demonstrate (You can use draw.io to draw diagrams)
  3. **Why Blockchain?**
     - How can Blockchain help for the idea you have chosen?
     - Include picture to demonstrate (You can use draw.io to draw diagrams)
  4. **Literature Survey (Existing works at least 3 recent papers/articles) of the field you have chosen**
     - To see if there is any **novelty** in your project plan in comparison with existing works
- **What to submit in D2L Dropbox?**
  - **2-3 pages draft (word/pdf document) that includes above mentioned 4 sections**

Samples from Class of Fall 2023

1. **Towards Patient-Centric Healthcare: Leveraging Blockchain for Electronic Health Records**
   - **Authors:** Thuan Nhan, Kritagya Upadhyay, Khem Poudel
   - **Year:** 2024
   - **Published in:** SIGMIS-CPR '24 (ACM Conference)

2. **Smart Digital Edition Management: A Blockchain Framework for Papyrology**
   - **Authors:** Matthew I. Swindall, Kritagya Upadhyay, James H. Brusuelas, Graham West, John F. Wallin
   - **Year:** 2024
   - **Published in:** SIGMIS-CPR '24 (ACM Conference)

3. **Decentralized Engagement: Blockchain's Lens on Social Media**
   - **Authors:** Harshit Kumar, Kritagya Upadhyay
   - **Year:** 2024
   - **Published in:** BLOCKCHAIN'24 (6th International Congress on Blockchain and Applications)

4. **QuadraCode AI: Smart Contract Vulnerability Detection with Multimodal Representation**
   - **Authors:** J. Upadhya, K. Upadhyay, A. Sainju, S. Poudel, M.N. Hasan, K. Poudel, J. Ranganathan
   - **Year:** 2024
   - **Published in:** ICCCN 2024 (IEEE International Conference on Computer Communications and Networks)

Samples from Class of Fall 2023 – Contd...

5. VulnFusion: Exploiting Multimodal Representations for Advanced Smart Contract Vulnerability Detection
    - **Authors:** J. Upadhya, A. Sainju, K. Upadhyay, S. Poudel, M.N. Hasan, K. Poudel, J. Ranganathan
    - **Year:** 2024
    - **Published in:** BCCA 2024 (IEEE International Conference on Blockchain Computing and Applications)

6. Chain Your Loot: Implementing Blockchain Into Gaming Loot Box Markets
    - **Authors:** S. Ali, B. Robinson, S. Solomon, S. Poudel, A. Sharma, K. Upadhyay
    - **Year:** 2025
    - **Published in:** CCWC 2025 (IEEE Computing and Communication Workshop and Conference)

7. A Blockchain and IoT-Enabled Framework for Ethical and Secure Coffee Supply Chains
    - **Authors:** J. Byrd, K. Upadhyay, S. Poudel, H. Sharma, Y. Gu
    - **Year:** 2025
    - **Published in:** Future Internet (MDPI Journal)

Ethereum Global Variables

- These built-in variables and namespaces provide context about transactions and blocks

Variable/Keyword	Meaning	Example Usage
`msg.sender`	• Caller address (EOA or contract) • Who is calling?	Restrict access to **owner** functions
`msg.value`	• Ether sent with txn (wei) • How much Ether was sent?	Only available in `payable` function
`block.timestamp`	• Time of current block (UNIX seconds) • What time is it?	Time-locked releases
`block.number`	• Current block height • What block are we in?	Trigger actions every *n* blocks
`address(this)`	• The contract's own address • Address of the current contract	Check internal balance
`address(this).balance`	• Ether balance of this contract • How much Ether (in wei) this contract currently holds?	For payouts or refunds

Fallback and Receive Functions

- Allow contracts to receive Ether or handle calls to undefined functions
- `receive()` function:
  - triggered when Ether is sent to a contract without data
  - `receive() external payable { emit Deposit(msg.sender, msg.value); }`

Action	Ether?	Data?	What gets called?
contract.transfer(5 ether)	Yes	No	receive()
contract.deposit({value: 5 ether})	Yes	Yes (deposit)	deposit() function

Fallback and Receive Functions

- `fallback()` function:
  - Catch-all function in Solidity
  - Runs when someone calls your contract but:
    - The function name doesn't exist, OR,
    - They send some data your contract doesn't understand, OR,
    - They send Ether and you don't have a receive() function

  - triggered when the call data does not match any existing function
  - `fallback() external payable { emit Log("Fallback triggered"); }`

Fallback and Receive Functions – Contd…

- Typical Use Cases
  - Logging unexpected calls, accepting donations, or acting as a router for proxy contracts

- Best Practice
  - If your contract should not receive Ether, include a fallback function that `revert()`s to reject unintended payments
  - Many NFT and token contracts use fallback functions to auto-return funds if someone mistakenly sends Ether to the wrong address

Gas and Optimization in Smart Contracts

- Gas Consumption
  - Every EVM opcode has a specific gas cost (e.g: `SSTORE` ≈ 20,000 gas)
  - Functions that write to storage consume most gas while reading is cheaper

- Optimization Strategies
  - Minimize storage writes (use `memory` or `calldata`)
  - Use `uint256` consistently to avoid padding overhead
  - Avoid loops over dynamic arrays
  - Use events for logging instead of on-chain storage
  - Mark helper functions as `view` or `pure`

- A loop that updates 10 values in storage can cost >200k gas; doing the same in memory costs <5k

- During the 2021 NFT boom, poorly optimized contracts wasted over $100 million worth of gas fees across the network!

Why Solidity has no floating-point numbers?

- Floating-point math can introduce rounding errors and non-deterministic behavior across machines
- To ensure all nodes in the network produce the same results, Solidity only supports integer arithmetic
- Simulate floating-point calculations
  - Use scaled integers to represent decimals
  - For e.g., instead of storing 2.5, store it as 250 with an implied 2 decimal places (scale factor = 100)

Why Solidity has no floating-point numbers? – Contd…

- Addition (e.g., 2.5 + 3.2)

```
pragma solidity ^0.8.0;

contract AddExample {
    uint constant SCALE = 100; // 2 decimal places

    function add() public pure returns (uint) {
        uint a = 250; // represents 2.50
        uint b = 320; // represents 3.20
        uint sum = a + b; // 570
        return sum; // represents 5.70
    }
}
```

Why Solidity has no floating-point numbers? – Contd…

- Division (e.g., 5.0 ÷ 2.0)

```
pragma solidity ^0.8.0;

contract DivideExample {
    uint constant SCALE = 100;

    function divide() public pure returns (uint) {
        uint a = 500; // 5.00
        uint b = 200; // 2.00
        uint result = (a * SCALE) / b; // (500*100)/200 = 250
        return result; // represents 2.50
    }
}
```

Audit and Best Practices

- **Testing Tools:** Truffle, Hardhat, Remix Debugger
- **Static Analysis:** Slither, MythX, SmartCheck find common bugs
- **Formal Verification:** Certora, Scribble prove logic properties
- **Secure Libraries:** Use OpenZeppelin standards (ERC-20, ERC-721)
- **Audits:** Professional security firms (CertiK, ConsenSys Diligence, Trail of Bits) review major projects

End of Chapter-4