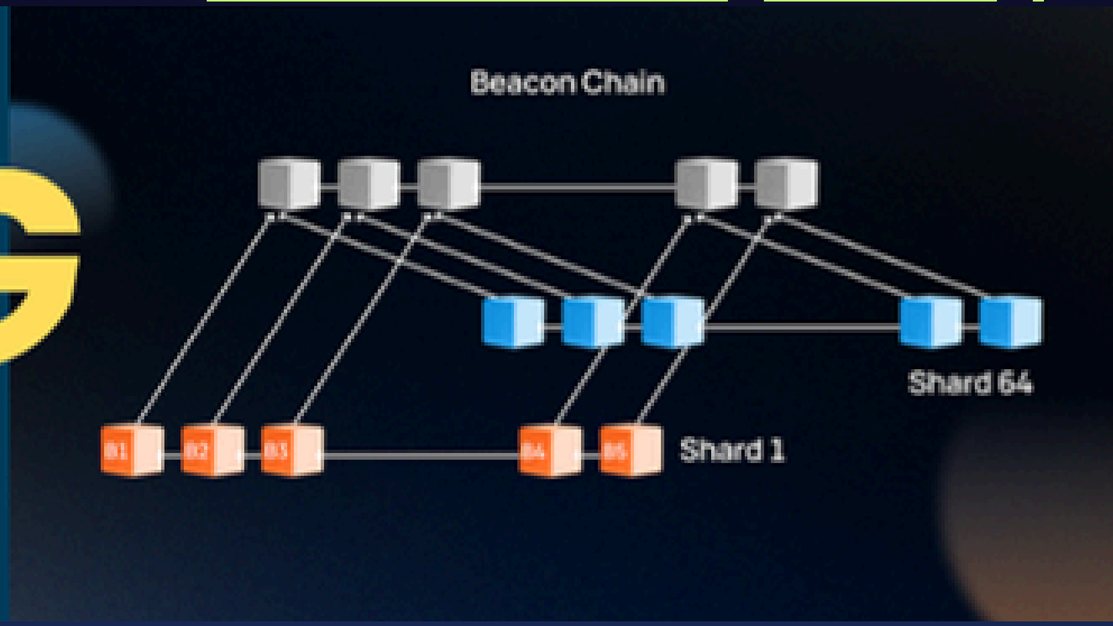
ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE (6-10)

SHARDING
& ETHDO TOOLS



Raja Rizwan Saleem
Lead Blockchain Trainer

Class-1

www.edversity.com.pk

OVERVIEW OF CASPER THE FRIENDLY FINALITY GADGET

Overview of Casper the Friendly Finality Gadget

Introduction to Casper FFG

- Casper FFG is a **hybrid consensus protocol** that combines elements of PoW and PoS during the Ethereum transition.
- It is referred to as a "finality gadget" because it introduces the concept of **finality** into the consensus mechanism, ensuring that once a block is finalized, it cannot be reverted.

Overview of Casper the Friendly Finality Gadget

Purpose and Motivation

- The primary goal of Casper FFG is to **gradually shift Ethereum** from PoW to a more scalable and energy-efficient PoS system.
- It aims to improve **security and scalability** by introducing finality, which protects the network from long reorganization attacks.

Overview of Casper the Friendly Finality Gadget

Key Components of Casper FFG

- **Hybrid PoW/PoS Model**: Initially, Ethereum continues to rely on PoW for block proposal, but validators in Casper FFG vote on the finality of blocks.
- **Validators and Checkpoints**: Validators in Casper FFG stake ETH and vote on "checkpoints" to confirm the finality of a chain.
- **Finality Votes**: Once 2/3 of validators agree on a checkpoint, that block and all previous blocks are considered finalized, ensuring they cannot be changed.

Overview of Casper the Friendly Finality Gadget

How Casper FFG Works

- **Block Proposal (PoW)**: New blocks are still proposed through PoW mechanisms.
- **Validator Voting (PoS)**: Validators who have staked ETH participate in voting to finalize blocks.
- **Finality**: Validators submit votes for checkpoint blocks, and if 2/3 consensus is achieved, the block is marked as "finalized."
- **Incentive Structure**: Validators are rewarded for correct votes but can be penalized for misbehavior (such as double voting).

Overview of Casper the Friendly Finality Gadget

Benefits of Casper FFG

- **Security**: With finality, the network becomes more resistant to certain types of attacks (e.g., the 51% attack).
- **Energy Efficiency**: It gradually shifts consensus away from energy-intensive PoW toward PoS.
- **Scalability**: Casper FFG is a stepping stone toward Ethereum 2.0’s fully PoS system, which is designed for scalability through mechanisms like sharding.

Overview of Casper the Friendly Finality Gadget

Penalties for Misbehavior

- **Slashing Conditions**: Validators who attempt malicious actions (e.g., signing multiple conflicting votes) can lose their staked ETH, which incentivizes good behavior.

  Role in Ethereum 2.0
- **Casper FFG as a Transition Mechanism**: It plays a transitional role before Ethereum fully adopts PoS through the Beacon Chain and Eth 2.0.
- **Finality in PoS**: Once Ethereum 2.0 is live, Casper will provide finality entirely based on PoS without PoW.

Overview of Casper the Friendly Finality Gadget

Challenges and Future Development

- **Validator Collusion**: Validators colluding could still cause network instability, though penalties like slashing mitigate this risk.
- **Full PoS Migration**: Casper FFG’s hybrid design is temporary, and the full shift to PoS will eliminate PoW entirely.

Summarizing Casper the Friendly Finality Gadget

Casper FFG introduces finality into Ethereum, allowing blocks to be finalized through validator consensus. It represents a key step toward Ethereum 2.0’s Proof of Stake mechanism, offering improved security, scalability, and efficiency during the network's transition away from Proof of Work.

DETAILS OF SHARDING CONCEPTS, SHARDING ALGORITHMS AND DATA DISTRIBUTION

What is Sharding

Sharding is a key concept in blockchain technology, especially with Ethereum 2.0, as it significantly improves scalability and throughput by splitting the network into smaller, more manageable sections known as "shards."

1. **Sharding Concepts**

Sharding is a process that divides the blockchain into smaller partitions called shards, each capable of processing transactions and smart contracts independently. Sharding allows parallel processing, meaning different shards can handle separate portions of network traffic, thus improving performance.

1. **Sharding Concepts**

Shard Chains: Shard chains are smaller chains that operate alongside the main Ethereum chain (beacon chain). Each shard has its own state and transaction history.

Beacon Chain: This is the central chain that coordinates all the shards and ensures network consensus. It manages validator information and assigns tasks to shards.

1. **Sharding Concepts**

Parallel Processing: By splitting the blockchain into shards, transactions can be processed simultaneously, leading to improved throughput. Shards can handle their own sets of smart contracts, transactions, and data.

Validator Assignments: Validators are randomly assigned to shards to validate transactions, which ensures security and decentralization, preventing any single shard from being taken over by bad actors.

2. **Sharding Algorithms**

Sharding algorithms manage how data and transactions are assigned across multiple shards in a secure and efficient way. Key algorithms include:

- **Random Validator Assignment (Proof-of-Stake)**: Validators (who propose and attest to blocks) are randomly assigned to different shards by the beacon chain. This randomness ensures that no shard becomes compromised or dominated by malicious actors.

2. **Sharding Algorithms**

- **Cross-Linking**: This is the process by which shard chains communicate with the beacon chain. Cross-links are checkpoints for shard chains, added to the beacon chain periodically to verify and update the state of each shard.
- **Data Availability Sampling**: This is a technique used to ensure that shard data is available to the network. Instead of requiring all validators to download all data, it allows a small sample of validators to check the availability of data in a shard, improving efficiency.

2. **Sharding Algorithms**

- **Committee Selection**: Validators in sharding are often organized into committees, groups assigned to specific shards to perform tasks. Committees are rotated regularly to prevent attackers from gaining control of any shard.
- **RANDAO**: This is a pseudo-random number generation algorithm used to ensure the randomness of validator assignments. It prevents any pre-planned strategies by attackers to dominate a shard.

3. **Data Distribution**

In a sharded network, data is distributed across different shards instead of being replicated across the entire network, as in the traditional blockchain. This distribution is a key reason for the performance improvement.

3. **Data Distribution**

State Storage: Each shard maintains its own state (e.g., account balances, contract data) rather than a single global state for the entire blockchain. This means data can be managed locally within each shard, reducing the load on individual validators.

Inter-Shard Communication: For shards to interact (e.g., if one shard needs data from another), a mechanism called **cross-shard communication** is employed. This allows different shards to send messages to each other. Ethereum 2.0 achieves this by relaying data through the beacon chain.

3. **Data Distribution**

Data Partitioning: Data in a sharded blockchain is partitioned such that each shard is responsible for a subset of the network's overall data. This prevents each node from needing to process every single transaction and state update in the network.

Security in Distribution: Even though data is split across shards, Ethereum 2.0 uses cryptographic methods like **zk-SNARKs** (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge) to ensure the integrity and correctness of transactions, making sure that even though data is distributed, the security remains intact.

Benefits of Sharding

Scalability: By parallelizing transaction processing across multiple shards, the blockchain can handle more users, applications, and data without degrading performance.

Efficiency: Each node in the network only needs to store and process data relevant to its assigned shard, reducing the workload compared to storing the entire chain's data.

Decentralization: Sharding keeps the network decentralized by distributing data and processing load across many smaller shards rather than relying on a single large blockchain.

Different Methods of Shard Management

Shard management involves techniques to ensure that shards are properly maintained, secure, and coordinated within a blockchain system. Managing shards effectively is essential for maintaining performance, consistency, and security in a sharded architecture like Ethereum 2.0.

Different Methods of Shard Management

1. Static Sharding

In static sharding, the number of shards and the assignment of validators to shards is predetermined and fixed. The key idea is that once the network is divided into a set number of shards, these shards remain constant.

Features:

- **Predefined Number of Shards**: The total number of shards is decided in advance, and it does not change dynamically based on network conditions.
- **Fixed Validator Assignment**: Validators are assigned to a specific shard and remain there for extended periods.

Different Methods of Shard Management

Advantages:

- **Simplicity**: Static sharding is relatively simple to implement since the shard structure and validator assignments do not change frequently.
- **Predictability**: Since the number of shards is fixed, there is no uncertainty in the network configuration.

Different Methods of Shard Management

Disadvantages:

- **Lack of Flexibility**: Static sharding does not adapt well to changes in network load or demand. Some shards may become overwhelmed, while others remain underutilized.
- **Vulnerability**: Fixed validator assignments can make it easier for malicious actors to target and attack specific shards.

Different Methods of Shard Management

2. Dynamic Sharding

Dynamic sharding is a more flexible approach where the number of shards and validator assignments can change over time based on network conditions, transaction volume, or other criteria.

Features:

- **Elastic Shards**: The number of shards can increase or decrease depending on network traffic and needs. This allows the network to adjust its resources in real-time.
- **Dynamic Validator Assignment**: Validators are periodically reassigned to different shards to prevent long-term dominance over a single shard and ensure decentralization.

Different Methods of Shard Management

Advantages:

- **Scalability**: Dynamic sharding allows the network to scale up or down efficiently by adding or removing shards as needed.
- **Security**: Regularly rotating validators among shards reduces the risk of a coordinated attack on any specific shard.

  Disadvantages:
- **Complexity**: Managing dynamic shards requires sophisticated algorithms to handle shard reassignments, cross-shard communication, and maintaining network consistency.

Validator Contract

Validator Contract

🔍 Key Features

Feature	Purpose
`registerValidator()`	Stake 32 ETH to become validator
`deregisterValidator()`	Unstake and exit the validator pool
`slashValidator()`	Simulate a validator penalty (no fund return)
`getAllValidators()`	View all registered validators
`isValidator()`	Check if an address is an active validator
`getValidatorCount()`	Get total validators

Validator Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/// @title Basic Validator Contract
/// @notice Simulates a simplified version of Ethereum 2.0 validator registration
contract BasicValidator {
 // Minimum amount required to become a validator (32 ETH)
 uint256 public constant MINIMUM_STAKE = 32 ether;

 // Structure to represent a validator
 struct Validator {
     bool isActive;     // Whether the validator is currently active
     uint256 stakedAmount; // Amount of ETH staked (should be 32 ETH)
 }
```

Validator Contract

```
// Mapping from address to Validator information
 mapping(address => Validator) public validators;

 // Event emitted when a validator is registered
 event ValidatorRegistered(address indexed validator);

 // Event emitted when a validator is removed
 event ValidatorRemoved(address indexed validator);

 /// @notice Register a validator by sending exactly 32 ETH
 function register() external payable {
     // Check if the sender has already registered
     require(!validators[msg.sender].isActive, "Already a validator");
```

Validator Contract

```
// Require exact stake of 32 ETH
    require(msg.value == MINIMUM_STAKE, "Must stake exactly 32 ETH");

    // Register the validator
    validators[msg.sender] = Validator({
        isActive: true,
        stakedAmount: msg.value
    });

    emit ValidatorRegistered(msg.sender); // Emit event
}

/// @notice Deregister the validator and withdraw the staked ETH
function deregister() external {
    // Ensure the caller is an active validator
    require(validators[msg.sender].isActive, "Not an active validator");
```

Validator Contract

```
// Save the amount to transfer
    uint256 amount = validators[msg.sender].stakedAmount;

    // Remove validator status
    validators[msg.sender].isActive = false;
    validators[msg.sender].stakedAmount = 0;

    // Send back the staked ETH
    payable(msg.sender).transfer(amount);

    emit ValidatorRemoved(msg.sender); // Emit event
 }
```

Validator Contract

```
/// @notice Check if an address is a currently active validator
 /// @param _validator The address to check
 /// @return Returns true if active
 function isValidator(address _validator) external view returns (bool) {
     return validators[_validator].isActive;
 }
}
```

THANK-YOU