ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE (3-5)

ETH 2.0 EXPLAINER

ETH 2.0 PHASES

PROOF OF STAKE

Raja Rizwan Saleem
Lead Blockchain Trainer

Class-6

Validator

A validator is a registered participant in the beacon chain. It is very much like a miner of the 'Proof of Work' protocol except they are engaged in the process with Ethereum chain only when they make a security deposit by sending ether (ETH) into the Ethereum 1.0 deposit contract. Validation in Eth2 can not be done without a security deposit. Follow Ethereum 2.0 Basic Terminology for other frequently used terms in Eth2.

Minimum requirement to become a validator

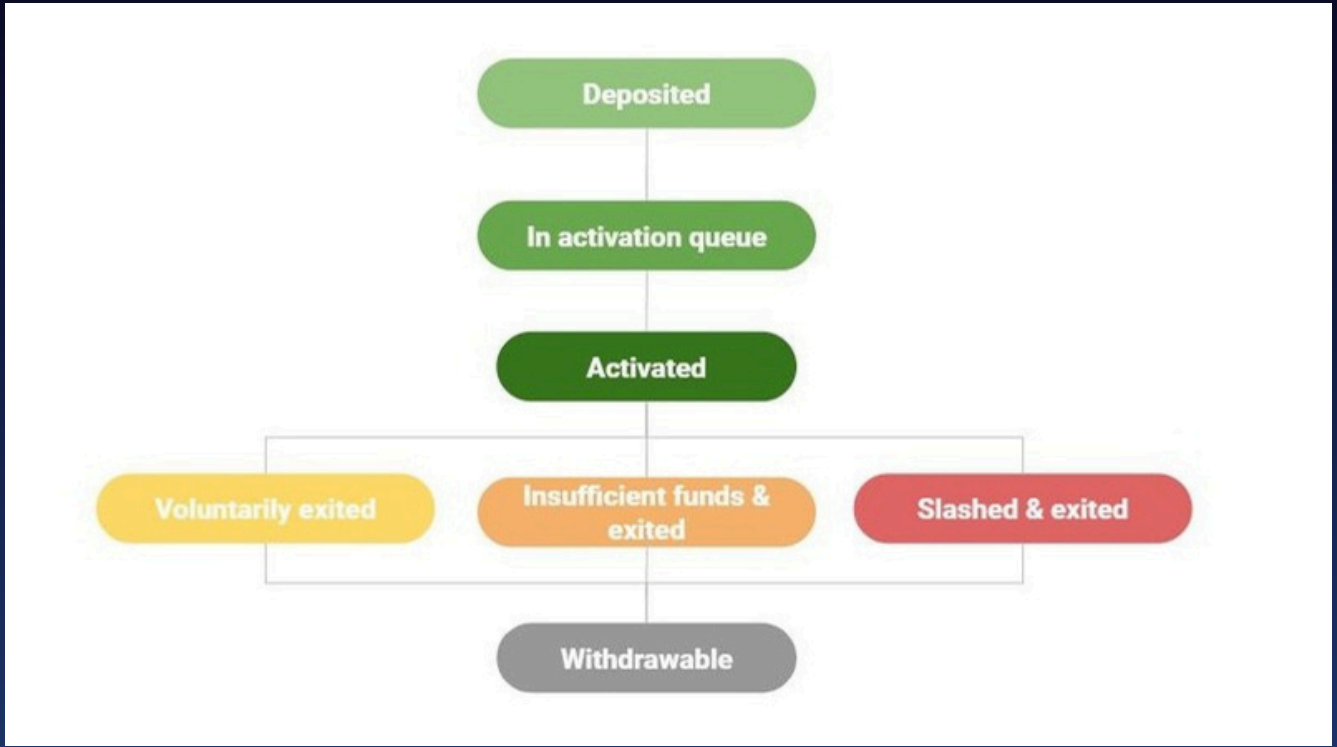
- To become a validator, a user needs
- a computer with minimum hardware requirement
- an internet connection and
- a minimum of 32 ETH to make the deposit.

Minimum requirement to become a validator

A validators is required to

- propose new blocks on shards to which the validator is assigned.
- Participate in committees by signing attestations on blocks proposed by other validators within the committee.
- Aggregrate attestations from other validators on a committee when assigned for broadcasting to the beacon chain.
- Maintain an RPC connection to a trusted beacon node to listen for validator assignment/shuffling.
- Sync assigned shard with beacon chain for each proof of custody period.

Validator Life Cycle



Validator Statuses

The status of a validator is determined by the status epoch fields in BeaconState. The field includes activation eligibility, activation, exit, and withdrawable. At any point of time, a validator may be in one of the following statuses:

Deposited

As obvious as it can be, the first step is to deposit a minimum amout of ETH in the deposit contract and register in BeaconState. It is a swap of funds between the Ethereum accounts and the eth2 validators triggering the corresponding Deposit operation. It determines who is involved, who is validating, how much is involved, and who can withdraw the funds.

In Activation Queue

If the validation of depoit is done then after 1 epoch, a validator is eligible to be activated. In epoch-processing, the validator is eligible for activation if this is checked within churn limit.

Validator Statuses

Activated

Generally, after 4 epochs, the validator is activated. Now that there are more than required validators, there is a waiting list and the selection of validator is decided by the set rules.

Once a validator is checked, the active validator will now be assigned duties at each epoch (certifying, proposing, etc.) and will receive rewards. Usually, validators are expected to remain in activated status for a long time.

Slashed

Slashing is a status imposed on the misbehaving validator in order to maintain the security of the chain. This can be avoided as explained earler. Once slashed, validator is foced to exit.

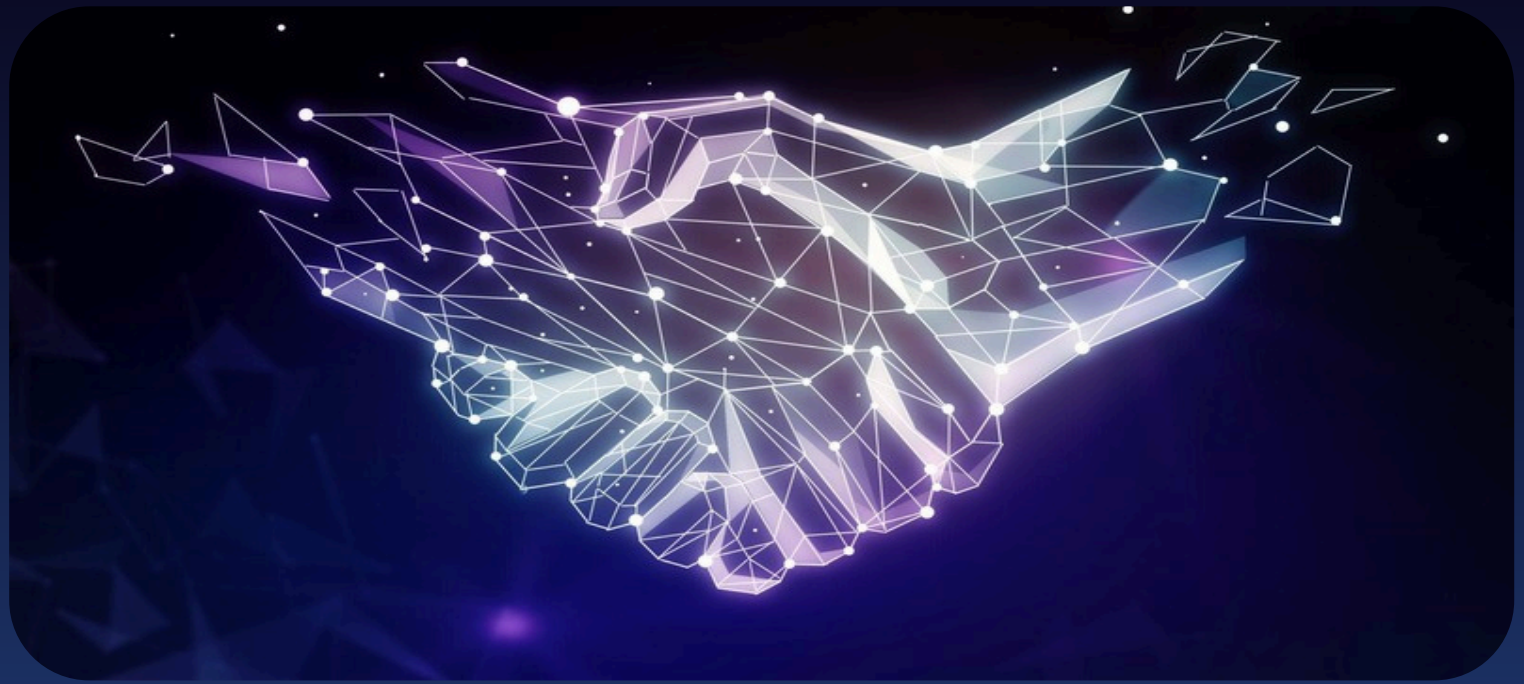
Validator Statuses

Exited

There can be three different conditions in which a validator is exited:

1. Slashing: Generally, malacious acts may lead to slashing of the validator which leads to exit from the chain. But in case a validator is mistaken (i.e. double votes or surround votes), a slashing operation may be created by the other validator and the wrong validator slows down and slashing operations are initiated. The validator will be forced to exit, plus an extra ~36 days delay before it can be withdrawn.
2. **Insufficient funds**: Because of going offline, a validator may get penalities and lose some of the deposit. If at any point deposit drops below half, the validator will be removed from the validator set entirely.
3. Voulantarily: After 2^11 epochs, a voluantry exit of the validator is initiated.

Deposit Contract



Practical Coding Deposit Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @title Ethereum 2.0 Depositor Contract Simulation
/// @notice This contract simulates how users can deposit ETH to become validators in Ethereum 2.0
contract Depositor {

 // Address of the contract deployer (could act as an admin)
 address public owner;

 // Minimum deposit amount (1 ETH for simulation, in real Ethereum 2.0 it is 32 ETH)
 uint256 public constant MIN_DEPOSIT = 1 ether;

 // Mapping to track how much each address has deposited
 mapping(address => uint256) public userDeposits;
```

Practical Coding Deposit Contract

```
// Event that is emitted when a user deposits ETH
 event Deposited(address indexed user, uint256 amount);

 // Event emitted when user withdraws ETH (for simulation; Ethereum 2.0
staking is one-way)
 event Withdrawn(address indexed user, uint256 amount);

 /// @notice Constructor runs once when the contract is deployed
 constructor() {
     owner = msg.sender; // Save the address of the deployer
 }
```

Practical Coding Deposit Contract

```
/// @notice Function for users to deposit ETH
 /// @dev Requires at least the minimum amount (1 ETH) to proceed
 function deposit() external payable {
    require(msg.value >= MIN_DEPOSIT, "Deposit must be at least 1 ETH");

    // Add the deposited amount to the sender's balance
    userDeposits[msg.sender] += msg.value;

    // Emit an event for frontends or monitoring tools
    emit Deposited(msg.sender, msg.value);
 }
```

Practical Coding Deposit Contract

```
/// @notice Returns the contract's total ETH balance
 /// @return The total ETH held by this contract
 function getContractBalance() external view returns (uint256) {
    return address(this).balance;
 }

 /// @notice Returns the amount deposited by a specific user
 /// @param user Address of the user whose balance is being queried
 /// @return The ETH amount deposited by that user
 function getUserDeposit(address user) external view returns (uint256) {
    return userDeposits[user];
 }
```

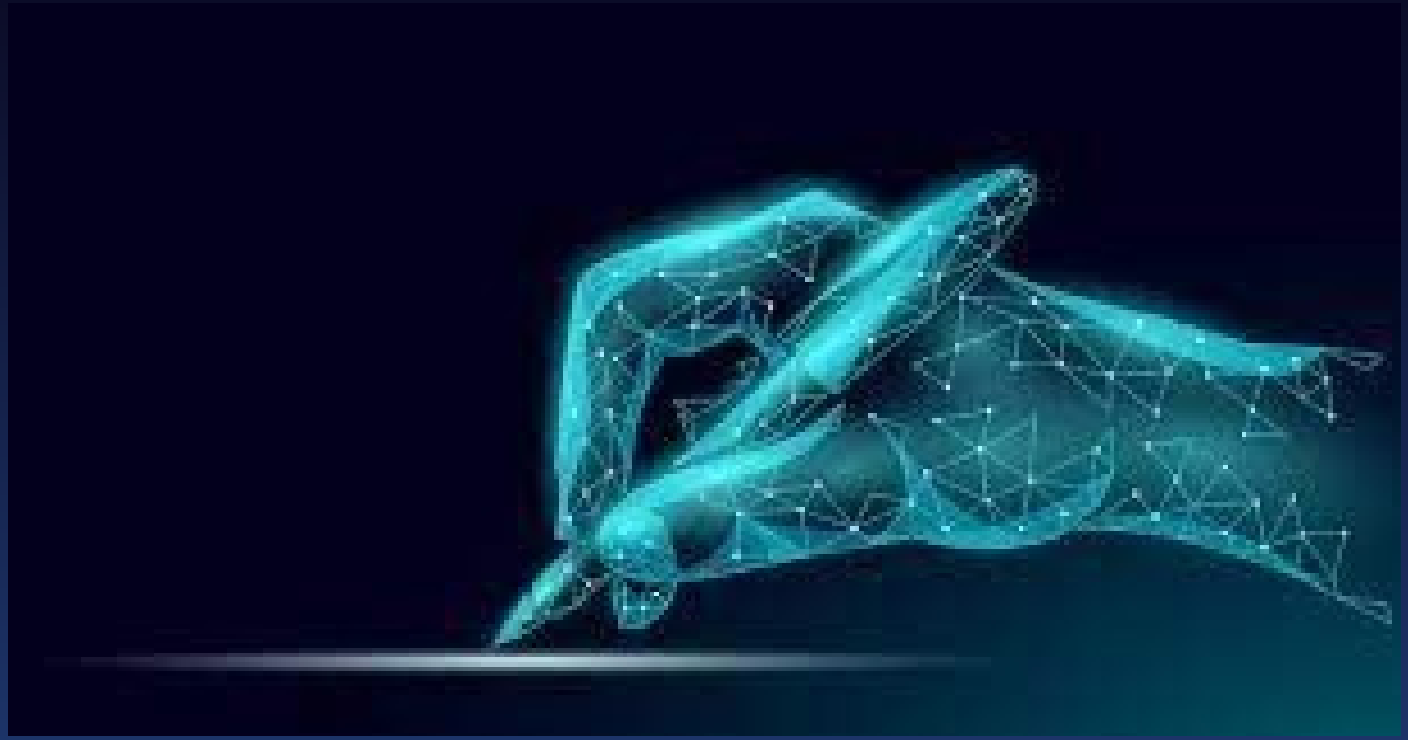
Practical Coding Deposit Contract

```
/// @notice (For simulation only) Allows users to withdraw their ETH
 /// @dev In real ETH2 staking, once staked, ETH is locked until withdrawal credentials are valid on the beacon chain
 function withdraw() external {
     uint256 amount = userDeposits[msg.sender];
     require(amount > 0, "No funds to withdraw");

     // Reset the user's deposit balance
     userDeposits[msg.sender] = 0;

     // Transfer the ETH back to the user
     payable(msg.sender).transfer(amount);

     // Emit a withdrawal event
     emit Withdrawn(msg.sender, amount);
 }
```

Practical Coding Deposit Contract

```
/// @notice (Optional) Admin function to destroy the contract and recover funds
 /// @dev Not used in real ETH2 deposit contracts
 function destroy() external {
     require(msg.sender == owner, "Only owner can destroy");
     selfdestruct(payable(owner));
 }
}
```

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