13

# The Merge and Beyond

In this chapter, we will cover "The Merge," the name given to the Ethereum upgrade that replaces **Proof-of-Work** (**PoW**) with **Proof-of-Stake** (**PoS**) consensus in the Ethereum blockchain. We will also cover what the future holds for Ethereum and how, through many novel technical innovations and upgrades, Ethereum is achieving its eventual goal of becoming a scalable world computer.

We'll cover the following topics in this chapter.

- Introduction
- Ethereum after the Merge
- The Merge
- Sharding
- The future of Ethereum

## Introduction

The goal of Ethereum is to ultimately transition into a scalable, performant, and secure version of Ethereum that will serve as the **world computer**, which is Ethereum's original vision. The concept of a world computer was introduced with Ethereum, in which a global, decentralized network of interconnected nodes runs P2P contracts without the limitation of shutting down or being censored or hacked. This vision started in 2015 with the Ethereum PoW chain and gained a lot of traction; however, challenges such as scalability, privacy, and performance somewhat hindered mass adoption. So-called Ethereum 2.0 was expected to address these issues in pursuit of becoming a world computer, however, there is no Ethereum 2.0 now; but the vision remains and the first major milestone of adopting PoS instead of PoW has been achieved. There is a rich and innovative roadmap ahead, which will ensure further efficiency and improvement in order to realize the eventual goal of a scalable world computer. This work is being done in several upgrades, which we'll cover in this chapter.

## Ethereum after The Merge

Note that Ethereum 1.0 and Ethereum 2.0 terminology is no longer used. Instead, the terminologies execution layer and consensus layer are now used to describe Ethereum 1 and Ethereum 2 respectively.

Remember that now the Ethereum vision looks like this:

Ethereum 1 → execution layer

Ethereum 2 → consensus layer

Execution layer + consensus layer = Ethereum

Eth 1 and Eth 2 terminologies are no longer used, and there is only one terminology now, i.e., Ethereum, which is the current and future version of Ethereum. Individual improvements and features like the Beacon Chain, The Merge, and sharding are now called upgrades. The Ethereum network will be upgraded through several upgrades over a period of a few years to achieve its eventual goal of scalable Ethereum becoming a World Computer. We can see this as a logical division in the protocol where execution deals with the processing of the transactions inside a block and maintains the world state of the chain by creating and executing smart contracts and transfer transactions. Consensus, on the other hand, chooses which blocks to include in the canonical chain.

Execution clients are new versions of Ethereum clients already in use like Geth, Nethermind, and Besu. These clients no longer have the PoW component (ETHASH) and now solely rely on consensus clients to achieve consensus.

Consensus clients are new client software that is developed specifically for PoS Ethereum. Some examples include Lodestar, Lighthouse, Prysm, and Teku. These clients are responsible for the validation of blocks and consensus and ensuring that valid blocks make it to the canonical chain.

> A full Ethereum node after The Merge is now a combination of an execution and consensus client.

The major goals behind the upgrades of Ethereum can be divided into the following different dimensions:

- **Staking**: Ethereum, from its early days, aimed for a transition to PoS. This was achieved in September 2022.
- **Sharding**: One of the main aims of Ethereum is sharding, which will improve scalability, efficiency, and performance.
- **Energy efficiency**: Ethereum became 99.95% efficient due to the move to PoS.
- **Increased security**: With the advent of quantum cryptography and to mitigate the threats faced by cryptography in the post-quantum world, it is envisaged that in Ethereum, quantum-resistant cryptography (or at least easy pluggability of quantum-resistant elements) will be introduced, so that the blockchain remains secure even in a post-quantum world. Also due to the proof-of-stake mechanism, compromising the network would be difficult.
- **Increased participation**: It is also expected that greater participation of validators will help with improved security and overall participation.
- **Increased performance**: Instead of requiring specialist hardware or high-end CPUs and GPUs for performing validation activities, in Ethereum, it is expected that validation activities will be performed by using typical consumer computers with the usual resources. With the introduction of **PoS**, a faster block time is expected. This also results in increased performance.

The following list of Ethereum's main releases shows how Ethereum has evolved over time, along with its vision of eventually achieving a world computer—a scalable, decentralized, and secure blockchain network:

- July 2015: Frontier
- March 2016: Homestead
- October 2017: Byzantium
- February 2019: Constantinople
- December 2019: Istanbul
- Late 2020: Berlin
- December 2020: Beacon Chain
- August 2021: London upgrade
- September 2022: The Merge
- 2023 and beyond: Sharding
- April 2023: Shanghai upgrade/Shapella update

As indicated here, the development of Ethereum is divided into upgrades. The most recent relevant upgrades related to the new vision of Ethereum are:

- **Beacon Chain**: Introduced staking to Ethereum and provides a foundation for future upgrades.
- **The Merge**: The switch from PoW to PoS
- **Sharding**: Consists of multiple upgrades to improve the scalability and capacity of Ethereum.

Let's first discuss the Beacon Chain in more detail.

## The Beacon Chain

The introduction of the Beacon Chain marked the integration of the PoS mechanism into the Ethereum ecosystem. The Beacon Chain introduced a consensus logic and block gossip protocol, which serves to secure the Ethereum network.

The **Beacon Chain** is the core system chain and manages the **Gasper PoS** consensus mechanism (explored later in the chapter). It is also used to store and maintain the registry of validators. Validators are nodes that participate in the PoS consensus mechanism. The Beacon Chain has a number of features, listed as follows:

- Production of good-quality randomness, which will be used for selecting block proposers and attestation committees without bias. Currently, a RANDAO-based scheme is prominent; however, other options such as BLS-based and STARK-based schemes have been considered in the past. RANDAO is a pseudorandom process that selects proposers for each slot in every epoch and rearranges the validators in the committees.
- Provision of attestation, which essentially means availability votes for a block in a shard chain. An adequate number of attestations for a shard block will result in the creation of a **cross link**, which provides confirmation for shard blocks in the Beacon Chain.
- Validator and stake management.
- Provision of validator sets (committees) for voting on proposed blocks.
- Enforcement of the consensus mechanism and the reward and penalty mechanisms.
- Serving as a central system chain where shards can write their states so that cross-shard transactions can be enabled.

In order to participate in the Beacon Chain, a Beacon Chain client (or node) is required.

## Beacon nodes

The **beacon node** is the primary link in the Beacon Chain that forms the central core of the Ethereum blockchain. The beacon node's responsibilities include the synchronization of the Beacon Chain with other peers. It performs the attestation of blocks and runs an RPC server to provide critical information to the validators regarding assignments and block attestation. In addition, it also handles state transition and acts as a source of randomness for the validator assignment process.

> Note that we use node and client interchangeably. However, there is a subtle difference between the terms client and node: when we say client, it actually means the software client that performs the functions of consensus or execution, whereas a node can be seen as a combination of client software and the hardware (computer) that it is running on. Generally, however, client and node are used interchangeably.

The beacon node also serves as a listener for deposit events in the Ethereum chain, as well as creating and managing validator sets. A beacon node keeps a synchronized clock with other nodes on the Beacon Chain to ensure the application of penalty rules (slashing). In a beacon node, various services based on the responsibilities mentioned previously are implemented. These include the blockchain service, synchronization service, operations service, Ethereum service, public RPC server, and the P2P networking service.

To participate in the Beacon Chain, a Beacon Chain client, or consensus client, is required. Based on the Ethereum specification, this client is developed by a number of different teams. A non-exhaustive list is presented here: `https://ethereum.org/en/developers/docs/nodes-and-clients/#consensus-clients`.

## Consensus client

The primary responsibility of the consensus client is to maintain synchronization with the Ethereum network by implementing the necessary logic. This process involves receiving blocks from other nodes and using the fork choice algorithm to ensure that the node always follows the chain with the most attestations. In addition, the consensus client also participates in its P2P network for block and attestation sharing.

However, the consensus client does not propose or attest blocks. Instead, this task is reserved for validators, an optional component added to the consensus client if the user stakes at least 32 ETH.

Without a validator, the consensus client only tracks the head of the Beacon Chain, ensuring that the node remains synchronized.

The consensus client mainly performs blocks and attestations gossiping over its P2P network, tracking the chain, running the fork choice algorithm, and beacon state management.

## Execution client

The execution client acts as an entry point for Ethereum users into the Ethereum blockchain network. The execution client handles transactions, manages the state, and contains the **Ethereum Virtual Machine** (**EVM**), state, and transaction pool. The execution client generates execution payloads, which comprise transactions, updated state tries, and similar data. These execution payloads are included in every block by consensus clients. Additionally, the execution client, using the EVM, re-executes transactions from the blocks it receives to ensure transaction validity. Moreover, the execution client offers users an interface to interact with Ethereum through RPC calls, enabling them to query the Ethereum blockchain, submit transactions, and deploy smart contracts. RPC calls are typically managed by a library like web3.js or via a cryptocurrency wallet. Execution clients allow interaction with Ethereum via the JSON RPC API and gossip transactions over the P2P network, execute transactions, verify state changes, manage state tries and receipts tries, and create the execution payload and send it to the consensus client.

In the next section, we introduce validator nodes, which serve as block proposers and attesters on the network as part of the PoS mechanism in Ethereum.

## Validator client

A **validator node**, also referred to as a validator client, actively participates in the consensus mechanism to propose blocks and provide attestations. A user can stake a minimum of 32 ETH to get the ability to verify and attest to the validity of blocks. As validators participate in the consensus mechanism to secure the protocol, they are incentivized financially for their effort. The validators earn **ether** as a staker by creating and validating new blocks on the chain.

> If you want to become a staker, follow the instructions at the link here: `https://launchpad.ethereum.org/en/`.

To add a validator to their consensus client, node operators must deposit 32 ETH in to the deposit contract. The validator client is included with the consensus client and can be added to the node whenever desired. The validator is responsible for attesting and proposing blocks, and it allows the node to receive rewards, penalties, or slashing. Additionally, running the validator software makes the node a candidate to be chosen as the block proposer for a slot.

Validators do block proposals, receive rewards and penalties, and block attestations.

**How to stake on Ethereum**

There are four main ways to stake on Ethereum: solo staking, staking as a service, pooled staking, and staking on centralized exchanges:

- Solo staking is the most impactful, providing complete control and rewards while being trustless. However, it requires technical knowledge and a dedicated computer connected to the internet.

- Staking as a service allows you to delegate the difficult part to a service provider while you earn rewards, but you need to entrust the node operation to a provider.
- Pooled staking is a simple option that allows users to stake any amount, even if it's below 32 ETH, and earn rewards proportionally. Several solutions exist, including "liquid staking," which involves an ERC-20 liquidity token representing the staked ETH. However, pooled staking is not native to the Ethereum network and carries risks.
- Finally, centralized exchanges provide staking services but require the highest level of trust assumption and can create a single centralized target for hacking and a single point of failure, making the network more vulnerable and less secure.

A validator has to perform a number of configurations before they can deposit ether as a stake to become a validator. Once they are accepted, they become part of the validator registry that is stored and maintained on the Beacon Chain.

---

Instructions on how to set up an Ethereum node including various consensus layer and execution layer clients are available here: `https://ethereum.org/en/developers/docs/nodes-and-clients/run-a-node/`.

---

The key functions that a validator client performs are listed as follows:

- New block proposals.
- Attestation provision for blocks proposed by other validators.
- Attestation aggregation.
- Connection with a trusted beacon node to listen for new validator assignments and shuffling.

A validator is assigned a status based on its current state. It can have one of six statuses: **deposited**, **pending**, **activated**, **slashed**, **exited**, and **withdrawable**. The deposited status means that a validator has made a valid deposit and is registered in the Beacon Chain state, pending means that the validator is eligible for activation, activated means that the activator is active, slashed means the validator has lost a portion of their stake, exited means the validator has exited from the validator set, and withdrawable indicates that the validator is able to withdraw funds.

A withdrawable state occurs after a validator has exited and roughly 27 hours have passed. A validator can be in multiple states simultaneously—for example, a validator can be in both the activated and slashed states at the same time.

Honest validator behavior is incentivized as part of the PoS mechanism, and dishonest behavior results in what is referred to as **slashing**. Slashing results in the removal of a validator from the validator committee (the active validator set) and the burning of a portion of its deposited funds. Slashing serves two purposes. First, it makes it prohibitively expensive to attack Ethereum and, secondly, it encourages (enforces) validators to actively perform their duties (functions). For example, a validator going offline when it is supposed to be actively participating in the consensus is penalized for its inactivity.

There are three scenarios when slashing can occur. The first is proposer slashing, which is when a validator signs two different blocks on the Beacon Chain in the same epoch. The second case where slashing can occur is when a validator signs two conflicting attestations. Thirdly, slashing can also occur when a validator, when attesting, signs an attestation that surrounds another attestation. In other words, it means that this scenario occurs when a validator first attests to one version of the chain and then later attests to another version of the chain, resulting in confusion as to which chain the validator actually supports. When any of the three preceding scenarios occur, the offending node is reported by a **whistleblowing validator**. The whistleblowing node creates a message containing evidence of the offense, sends it to a proposer to include it in a block, and propagates it on the network. In Phase 0, the total slashing reward is taken by the proposer and the whistleblowing validator does not get any reward. This has the potential to change, where both actors might be rewarded.

Also, remember that penalization and slashing are two related but different concepts. Penalization results in a decrease in the balance of a validator as a result of, for example, being inactive or going offline. On the other hand, slashing results in a forceful exit from the Beacon Chain along with the responsible validator's deposit being penalized in every epoch for as long as it has to wait for its turn in the exit queue to leave the chain.

Slashing and penalty calculations are based on several factors with various variables such as the length of validator inactivity and the type of offense that triggered the slashing. Moreover, a penalty is applied at various points in the slashing process. For example, a minimum penalty of *effective balance of slashed validator / 32* is applied when a validator proposer includes the message reporting the offense from the whistleblower in a block. After that, at the beginning of each epoch, a penalty calculated as *3 * base reward* is applied. Another penalty is applied between the time of inclusion of the whistleblowing message in a block, and the time when the slashed validator is able to withdraw.

Earning rewards by staking ether in the deposit contract also depends on several factors. A simple example is that if someone stakes 32 ETH with a current price of, for example, USD 240 per ETH, with a validator uptime of 80%, the annual interest earned will be around 8%. The base reward is calculated as per the following formula from the Phase 0 specification:

```
base_reward = effective_balance * (base_reward_factor / (base_rewards_per_epoch * sqrt(sum(a
```

Here, `base_reward_factor` is set at the default value of 64, the `base_rewards_per_epoch` value is 4, and `sum(active balance)` is the total staked ether across all active validators.

The reward received by a validator is determined by two factors: their effective balance and the number of validators in the network. As the number of validators increases, the overall reward issued by the network also increases, but the base reward given to each validator decreases proportionally. These factors significantly influence the APR for a staking node. The total reward is calculated by combining five distinct components, each with a specific weight that determines the extent to which it contributes to the overall reward.

These five components are source vote, target vote, head vote, sync committee reward, and proposer reward. Votes are related to the timely voting of the consensus mechanism for the source, target checkpoints, and the head vote is related to the timely voting of the correct head block. The sync committee reward is a reward for participating in a sync committee, and the proposer reward is for proposing a block in the correct slot. All these components have assigned weights that sum up to 64.

The total reward received by a validator is determined by calculating the sum of the weightings of the five reward components and dividing it by 64. A validator who has made timely source, target, and head votes, proposed a block, and participated in a sync committee can receive `64/64 * base_reward`.

The effective balance is used to calculate the proportion of rewards and penalties applied to a validator. It is based on the validator's balance and the previous effective balance. The maximum effective balance can always be only up to 32 ETH. Even if the actual balance is 1,000 ETH, the effective balance will still be 32 ETH.

Validators can earn rewards by performing actions shown in the table below.

| **Action** | **Reward frequency** |
|---|---|
| `New block proposal` | Each epoch |
| `Block attestation` | Each epoch |
| `Sync committee participation` | Every 27.3 hours (256 epochs). However, for example with 300,000 validators, the likelihood of a validator being chosen is very low and can happen roughly every 22 months on average as the number of validators grow the likelihood of selection will decrease accordingly. |

Overall, there are three main components: the Beacon Chain, beacon nodes, and validator nodes. There are some key differences between a beacon node and a validator node, which are listed in the following table.

In the Ethereum Beacon Chain (consensus layer), there are two distinct types of nodes: Beacon Chain nodes and validator nodes. The key differences between these node types are presented in the following table.

| **Feature** | **Beacon nodes** | **Validator nodes** |
|---|---|---|
| `Networking` | Connected via P2P to other beacon nodes | Dedicated connection with a single beacon |

| | | |
|---|---|---|
| | | node |
| `Staking` | No staking requirements to participate in the network | Ether staking required to participate in the network |
| `Block creation` | Attest validations and propagate blocks across the beacon chain | Propose and sign blocks |
| `Operation` | Read | Write |

Being a PoS-based blockchain, Ethereum must have the ability to take deposits from users as a stake. This requirement is addressed by a deposit contract.

Let’s now explain PoS in a bit more detail.

## Proof-of-stake

In the PoS mechanism, validators stake their funds in a smart contract on the Ethereum chain. This stake serves as collateral that can be lost if the validator behaves dishonestly or negligently. Validators are responsible for checking the validity of new blocks and occasionally creating and propagating new blocks themselves.

PoS has several advantages over the PoW mechanism. It is more energy-efficient, as it does not require significant energy consumption for computational purposes. It also has lower barriers to entry, as high-end, expensive specialized hardware like ASICs is not required to participate in the consensus process. PoS also reduces the risk of centralization by encouraging more nodes to secure the network. Furthermore, a low amount of ether issuance is needed to incentivize participation because it requires low energy. In the event of a 51% attack, PoS imposes economic penalties for misconduct, making it much more costly for attackers than PoW. The community can also use social recovery mechanisms to recover an honest chain if a 51% attack occurs and overwhelms the crypto-economic defenses.

To become a validator in Ethereum’s PoS system, a user must deposit 32 ETH into the deposit contract and use three software components: an execution client, a consensus client, and a validator. Upon deposit, the user enters an activation queue, which also serves as a mechanism to limit the number of new validators joining the network. Once activated, validators receive new blocks from their peers on the Ethereum network. The role of a validator here is to check the validity of the transactions in the received block and signature before voting in favor of the block and announcing the attestation to the network.

Unlike PoW, where the timing of blocks is determined by mining difficulty, PoS has a fixed block production frequency. Time is divided into slots of 12 seconds, and 32 slots make up an epoch of 6.4 minutes.

A period of 2,048 epochs, roughly 9.1 days, is called an eek or "Ethereum week." It can be used to measure processes that take a long time.

We can visualize this in *Figure 13.1* below:

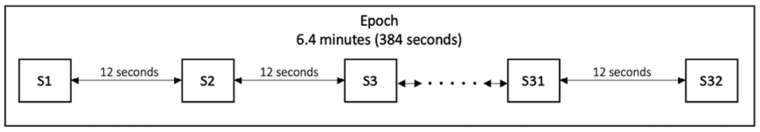

*Figure 13.1: An epoch consists of 32 slots (S1 to S32)*

In each slot, a single validator is chosen randomly through RANDAO to propose a new block and broadcast it to other nodes. Additionally, a committee of validators is randomly selected in each slot, and their votes are used to determine the validity and eventual acceptance of the proposed block.

Note that not every validator participates in voting for every block. Instead, they are assigned to a "committee" for each epoch, and each committee is randomly assigned a slot to attest to the proposed block's validity. These committees are distributed among the 32 slots in an epoch, with a maximum of 2,048 validators per committee. This means that each validator attests to only one block per epoch, specifically the one assigned to their committee's slot. Consequently, each slot or block is attested by 1/32 of the entire validator set.

We can visualize this in *Figure 13.2* below:

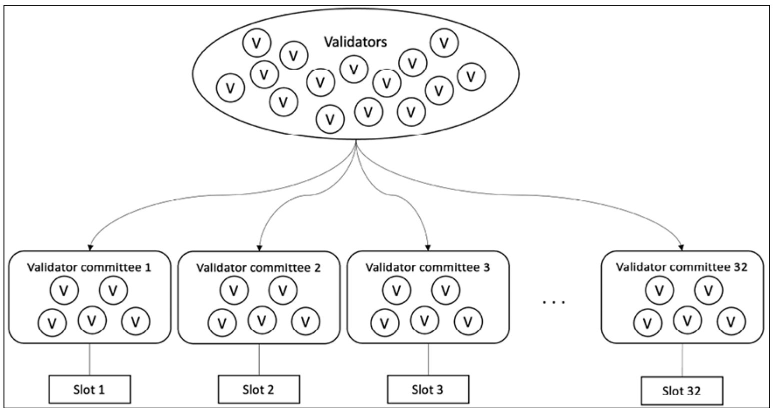

*Figure 13.2: One committee assigned to one slot*

In addition to attesting to the current head block, validators also attest to two other blocks referred to as "checkpoint" blocks. Each epoch features a single checkpoint block that designates the most recent block at the epoch's start.

During every epoch, validators validate two checkpoints: the "source" and "target" blocks.

Other than head, source, and target blocks, validators also include some other information in the attestation message, described below:

- `aggregation_bits` : This is a bit vector of validators where the bit position maps to the validator index in their committee. The value `0` or `1` indicates whether the validator is active and in agreement with the block proposer. In other words, whether the validator has signed (attested) the message or not.
- Data: This field contains several elements including the slot, index, beacon block root, source, and target:
  - Slot: Attested slot number
  - Index: The validator's committee number in a given slot
  - Beacon block root: Root hash of the recent block (at the head of the blockchain) observed by the validator
  - Source: Validator's view of the most recent justified block
  - Target: Validator's view of the first block in the current epoch
  - Signature: A **Boneh-Lynn-Shacham** (**BLS**) aggregate signature of the signatures of the validators.

After creating the data, the validator can change the bit in the `aggregation_bits` that corresponds to their own validator index from `0` to `1` to indicate their participation in the vote. To complete the process, the validator uses a private key to sign the attestation and sends it out to the network.

Not every validator listens to every other validator on the network. Passing attestation data by every validator to every other validator in the network would be too noisy and could cause congestion on the network. To address this issue, attestations are aggregated within the so-called "subnets" before broadcasting. These subnets can be seen as smaller networks within the larger network created by dividing the larger network. These subnets make communication efficient by reducing the distance that data needs to travel. Each committee is divided into 64 subnets to allow aggregation.

At every epoch, an aggregator is chosen from each subnet, whose role is to gather all the attestations that contain equivalent data to their own. The `aggregation_bits` field records the sender of each matching attestation. This means that all the validators who agree with the aggregator's data aggregate their signatures into a single signature. Finally, the aggregator broadcasts the aggregated attestations to the wider network. Once a validator is chosen as a block proposer, they assemble the aggregated attestations from the subnets up to the most recent slot and include them in the new block.

We can visualize the concept of subnets and an aggregator in *Figure 13.3* below:

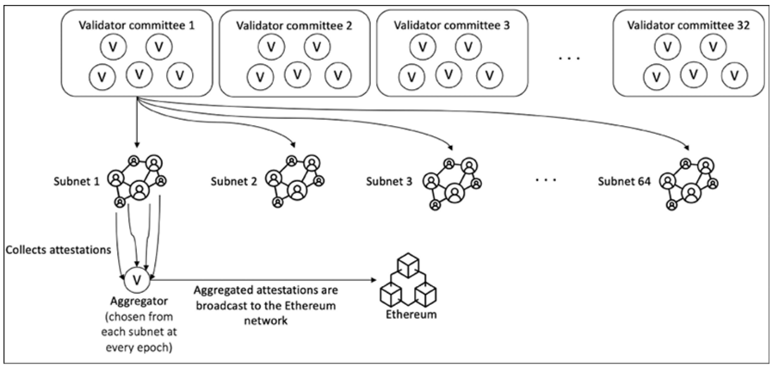

*Figure 13.3: Subnets and aggregator*

Now let’s see how the transaction flow works in the PoS Ethereum, i.e., Ethereum after The Merge.

The PoS transaction execution flow can be broken down into several steps, as described below:

1. A transaction is composed and signed by a user’s private key. Typically, this is done using a library like ether.js or web3.js. The user specifies the gas amount they will pay to incentivize validators to include the transaction in a block.
2. This transaction is submitted to an Ethereum execution client, which verifies that the sender’s ETH balance is enough and the correctness of their signatures. If the transaction is valid, it’s added to the client’s mempool.
3. This transaction is broadcast to other nodes via the execution layer’s P2P gossip network.
4. A random node on the network is chosen using RANDAO as the block proposer for the current slot. This node’s role is to construct and broadcast the subsequent block to be appended to the blockchain and update the global state. Note that this node is not chosen as part of the transaction execution process; it’s chosen randomly by the underlying protocol for each slot.
5. The execution client gathers transactions from the mempool and batches them into an “execution payload,” which is then executed locally on the EVM to produce the new state.
6. The execution client sends this execution payload to the consensus client.
7. The consensus client (sometimes called the beacon client) encapsulates it within a “beacon block.” Beacon blocks also contain information regarding rewards, penalties, attestations, and other similar information.
8. Other consensus nodes receive the beacon block via the consensus layer’s P2P gossip network.
9. These clients send the payload to their respective execution clients, where the transactions are executed again locally on the EVM to validate the proposed state change.
10. The validator client determines and verifies that the block is valid and that it is the correct logical next block based on the heaviest weight of attestation defined in the fork choice rule, i.e., **Latest Message Driven Greediest Heaviest Observed Subtree** (**LMD GHOST**).
11. The block is then added to the local database of each node that has verified it.
12. A transaction is deemed “finalized” once it becomes a part of a chain with a supermajority link between two checkpoints.
13. Once an adequate number of validators has attested a block, it is appended to the head of the chain and finalized.

We can visualize the transaction flow in *Figure 13.4* below:

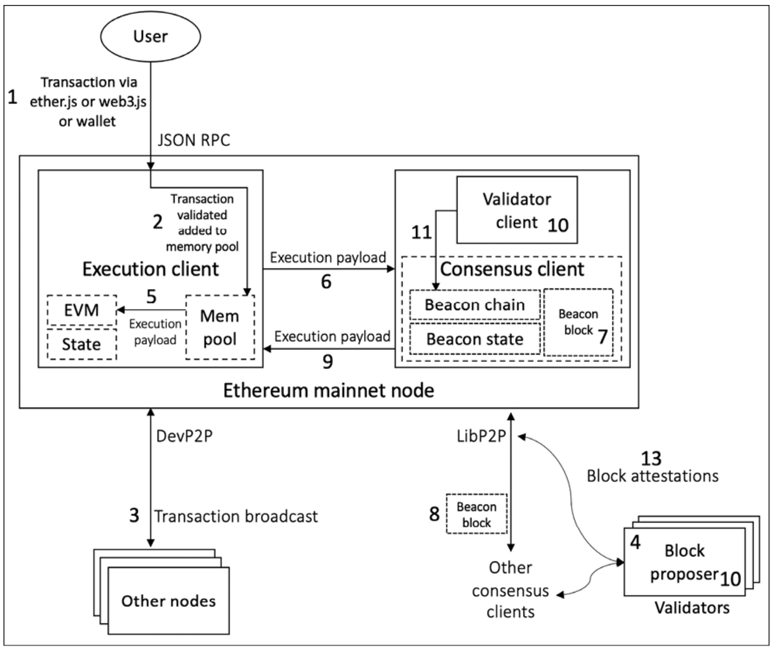

*Figure 13.4: Ethereum transaction flow after The Merge*

Read *Figure 13.4* using the numbers shown, which match the steps described in the transaction flow above.

Note that checkpoints are established at the start of each epoch and require a 66% attestation from the network's total staked ETH to qualify as a supermajority link.

A transaction is considered "final" when it is included in a block that cannot be changed without a significant amount of ETH being spent. Ethereum's PoS protocol achieves this through the so-called "checkpoint" blocks. The first block in each epoch serves as a checkpoint, and validators vote for pairs of checkpoints they deem valid. For example, suppose a pair of checkpoints receive votes representing at least two-thirds of the total staked ETH. In that case, the checkpoints are upgraded, with the more recent block becoming "justified" and the earlier block already justified from the previous epoch becoming "finalized." To reverse a finalized block, an attacker must commit to losing at least one-third of the total staked ETH, which is prohibitively expensive.

As finality requires a two-thirds majority, an attacker could prevent the network from reaching finality by voting with only one-third of the total stake. As a defense against this threat, an "inactivity leak" mechanism activates when the chain fails to finalize for more than four epochs. The inactivity leak gradually drains away the staked ETH from validators as a penalty for voting against the majority, enabling the majority of validators to regain a two-thirds majority and finalize the chain.

Finality, can be defined as the assurance that a block, once finalized, will not revert back. To achieve this, a mechanism called **Casper the Friendly Finality Gadget** (**Casper-FFG**) is implemented in the consensus layer. A paper on the topic is available at the following URL: `https://ethresear.ch/uploads/default/original/1X/1493a5e9434627fcf6d8ae62783b1f687c88c45c.pdf`.

In Gasper, finality is achieved using the following logic:

- If at least two thirds of the total staked ether cast their votes in favor of a block, it becomes “justified.” While justified blocks are typically stable, they can still be reverted under certain conditions.
- If another block is justified on top of a justified block, it is then “finalized.” Once a block is finalized, it is committed to the canonical chain and cannot be reverted unless an attacker manages to destroy a substantial amount of ether.

It is worth noting that not every slot leads to a “justified” or “finalized” state for blocks. These states are only reached by the first block of each epoch, which is referred to as the “checkpoint” block. When a checkpoint block is upgraded to justified, it must be linked to the previous checkpoint. This means that at least two thirds of the total staked ether must vote that checkpoint B is the rightful descendant of checkpoint A. As a result, the previous checkpoint block is then finalized, and the more recent block is justified.

Being a validator on the network is a serious commitment that requires maintaining proper hardware and internet connectivity to participate in the block validation and proposal process. In exchange for their services, validators are rewarded with ether, which increases their staked balance.

However, opening up participation publicly on the network as a validator also comes with risks, as it can expose the network to attacks by malicious users. As a prevention mechanism against hacking attacks, validators are penalized by slashing their stake if they fail to participate in the network as and when expected. In addition, they risk losing their entire stake if they engage in dishonest behavior. The two primary forms of dishonest behavior are:

- Equivocating, which means proposing more than one block in a single slot.
- Contradictory attestations, where a validator submits conflicting attestations.

The amount of slashed ETH depends on the number of validators being penalized simultaneously, known as the “correlation penalty.” This slashing can range from a minor penalty of around 1% of the stake to losing 100% of their stake in a mass slashing event. The penalty is imposed halfway through a forced exit period. There’s an immediate penalty of up to 0.5 ether on the first day. On day 18, the correlation penalty is enforced, and expulsion from the network occurs in roughly 5 weeks (36 days). In addition, validators who are present on the network but do not submit votes receive minor attestation penalties daily. As a result, a coordinated attack on the network would be prohibitively expensive for the attacker.

The fork choice rule in Ethereum is called **LMD GHOST**. Remember we discussed the **Greediest Heaviest Observed SubTree** (**GHOST**) in the context of Ethereum mining in *Chapter 9*, *Ethereum Architecture*. LMD GHOST is a variant of GHOST that was implemented in Ethereum with some modifications. More information can also be found at **https://ethereum.org/en/whitepaper/#modified-ghost-implementation**.

LMD GHOST governs a fork-handling mechanism to ensure that in the case of a fork, the correct honest chain is automatically chosen. As a general rule, the honest chain is the one that has the most attestations and stake (weight). In the event of a fork, the clients will use this fork choice rule to select the correct honest chain. Forks may occur due to the actions of colluding participants; however, the Beacon Chain's random selection of validators mitigates that to some extent, because validators do not know in advance when they will be selected. The paper on the topic is available at the following URL: `https://arxiv.org/pdf/2003.03052`.

In the next section, we introduce another important element that deals with all the networking requirements of Ethereum: the P2P interface.

## P2P interface (networking)

This element deals with the networking interfaces and protocols for the Ethereum blockchain. There are three main dimensions addressed by the development of the Ethereum P2P/networking elements:

- The gossip domain
- The discovery domain
- The request/response domain

Currently, libP2P is used in various clients for this purpose. More details on this topic are available at `https://libp2p.io`.

The networking specification also covers the essentials of a test network where multiple clients can run simultaneously, that is, the interoperability of the test network and mainnet launch. In order to achieve interoperability, all Ethereum client implementations must support the TCP libp2p transport. This must also be enabled for both incoming and outgoing connections. Incoming connections are also called inbound connections or listening connections. Outgoing connections are also called outbound connections or dialing connections. Implementors may choose not to implement an IPv4 addressing scheme for the mainnet. Ethereum may implement an inbound connectivity feature only for IPv6, but the clients must support both inbound and outbound connections for both IPv4 and IPv6 addresses.

**Simple Serialize**, or **SSZ** for short, is the algorithm standard for providing serialization abilities for all data structures in Ethereum client software. SSZ has support for many data types, such as Boolean ( `bool` ), unsigned integers ( `uint8` , `uint16` , `uint32` , and `uint64` ), slices, arrays, structs, and pointers.

**BLS cryptography** (BLS12-381) is used extensively in Ethereum to provide security and efficient verification of digital signatures. **BLS** signatures, allows the aggregation of cryptographic signatures to contribute to the scalability of the network. BLS is used by validator clients to sign messages, which are then aggregated and eventually verified efficiently in the distributed network, thus increasing the overall efficiency of the network. A Go implementation of the BLS12-381 pairing is available at `https://github.com/phoreproject/bls`.

This completes our introduction to the Beacon Chain. In the next section, we explore features of The Merge.

## The Merge

The Beacon Chain – the consensus layer, has been functioning independently since December 2020 and implements PoS by coordinating a network of validators using data from the Ethereum network. The Merge combined these networks, allowing Ethereum to transition to PoS as both the execution and consensus clients work together to verify the state of the network. In *Figure 13.5* below, we can visualize what Ethereum looked like before The Merge and what it looks like after The Merge:

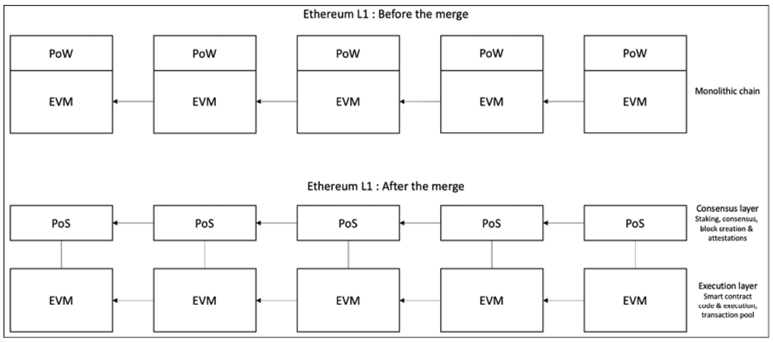
Ethereum L1 : Before the merge
PoW
PoW
PoW
PoW
PoW
EVM
EVM
EVM
EVM
EVM
Monolithic chain
Ethereum L1 : After the merge
PoS
PoS
PoS
PoS
PoS
Consensus layer
Staking, consensus, block creation & attestations
EVM
EVM
EVM
EVM
EVM
Execution layer
Smart contract code & execution, transaction pool

*Figure 13.5: Ethereum before The Merge vs. after The Merge*

Note that it is no longer possible to run an execution client alone. Following The Merge, both execution and consensus clients must be run together to allow access to the Ethereum network as shown below in *Figure 13.6*:

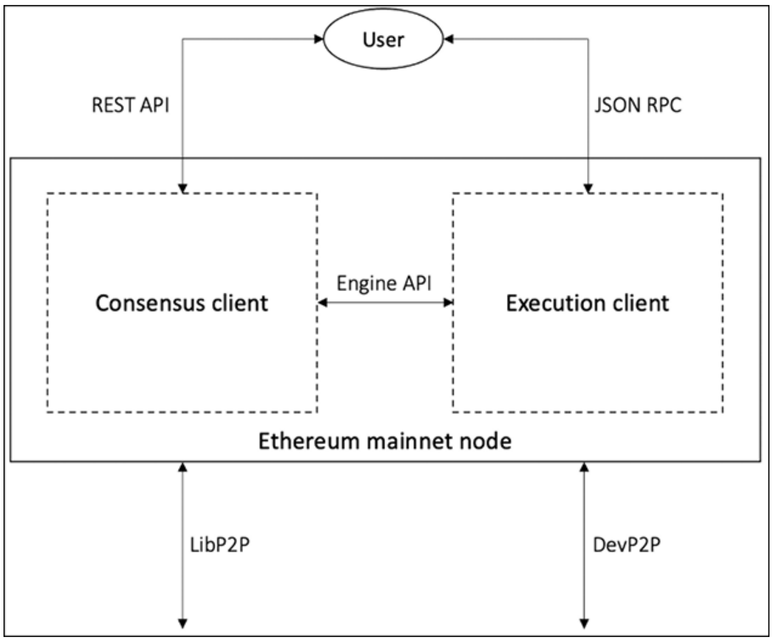

*Figure 13.6: Ethereum full-node architecture after The Merge*

As shown in the preceding diagram, after The Merge, two separate clients run in the full node:

- **Execution client**: The original PoW Ethereum clients that will continue to handle execution and process blocks, maintain memory pools, and synchronize blocks. The PoW component is removed.
- **Consensus client**: This is the Beacon Chain client responsible for performing PoS consensus, ensuring canonical chain emergence, block

gossiping, block attestation, and receiving validator rewards.

These clients communicate with each other via the Engine API.

The diagram below shows all three node types and their relationships:

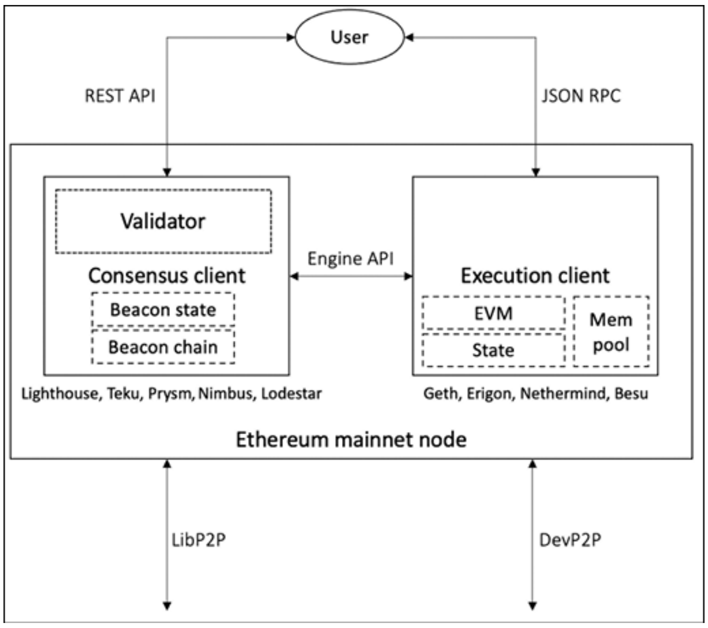

*Figure 13.7: Different node types in Ethereum and their relationship*

There is also a change in the block architecture after The Merge, i.e., the merge of the Beacon Chain and the old PoW chain, which is shown below:

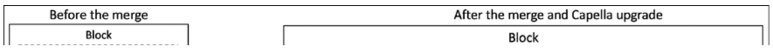

*Figure 13.8: Ethereum block architecture before and after The Merge (showing main fields)*

As shown in the preceding diagram after The Merge, the network no longer has PoW blocks. Instead, the contents of the old PoW blocks are integrated into blocks produced on the Beacon Chain. This change makes the Beacon Chain the new primary consensus layer for Ethereum, taking over from the previous PoW consensus layer.

The block production frequency is now 12 seconds, instead of the average 13 seconds in PoW.

The consensus layer Beacon Chain blocks contain `ExecutionPayloads`, which serve as the post-Merge equivalent of blocks on the old PoW chain. For users, interactions with Ethereum occur through these `ExecutionPayloads`.

Transactions on this layer are still processed by execution layer clients.

---

All changes in the block structure are described here in EIP-3675: **https://eips.ethereum.org/EIPS/eip-3675#block-structure**.

---

Some fields that were present in PoW block headers are no longer relevant to PoS and are therefore unused. To avoid causing too much disruption to existing tools and infrastructure, these fields are not removed from the data structure entirely but are instead set to 0 or their equivalent in the data structure. These fields include ommers, ommer hash, difficulty, and nonce.

The `mixHash` field now contains the RANDAO value.

The BLOCKHASH opcode remains available after The Merge, but its pseudo randomness is significantly weaker since it is no longer generated through the PoW hashing process. The DIFFICULTY opcode (0x44) has been updated and renamed PREVRANDAO. It returns the output of the randomness beacon provided by the Beacon Chain, making it a stronger source of randomness than BLOCKHASH, although it can still be biased. The value exposed by PREVRANDAO is stored in the `ExecutionPayload` where `mixHash`, a value associated with PoW computation, used to be stored. The `mixHash` field of the payload is renamed PREVRANDAO.

The target size of each block is set at 15 million gas units. However, the actual size of the blocks can vary based on the network demands as long as they stay within the block limit of 30 million gas units. To maintain a suitable block size, the total amount of gas used by all transactions in a block must be less than the block gas limit. This restriction is essential because it prevents blocks from becoming excessively large. If blocks could grow without limits, low-power nodes would be unable to keep up with the network due to their hardware limitations.

The larger the block, the more computing power is required to process it. Lower power nodes would miss out as a block must be processed within the next designated time slot. These requirements for more computing power would ultimately centralize the network. Therefore, the block size restriction helps prevent centralization, which could occur when only those users with high-end hardware could process the blocks, resulting in the network being used only by a select few. Limiting the block size helps to decentralize the network.

An Ethereum blockchain block contains information about the consensus, identification, execution, and other metadata necessary for the execution of the protocol. The block contains the following fields.

| Field | Description |
|---|---|

| | |
|---|---|
| `slot` | The number of the slot the block is in |
| `proposer_index` | Proposing validators, ID |
| `parent_root` | Hash of the previous block |
| `state_root` | Root hash of the state tree |
| `body` | Contains several fields, described next |

The body contains several fields as shown below.

| **Field** | **Description** |
|---|---|
| `randao_reveal` | A verifiable random value provided by the proposer validator to be mixed with the RANDAO. Used for the next validator selection |
| `eth1_data` | Deposit contract information including deposit root, the deposit count, and blockhash |
| `graffiti` | Arbitrary data field – can be used for writing any information |
| `proposer_slashings` | The list of validators to be slashed |
| `attester_slashings` | The list of validators to be slashed |
| `attestations` | The list of attestations for the current block |
| `deposits` | The list of new deposits to the deposit contract |
| `voluntary_exits` | The list of validators exiting the network |
| `sync_aggregate` | A signature and a bit vector representing the sync committee (subset of validators) |
| `execution_payload` | Transactions sent from the execution client |
| `bls_to_execution_changes` | Contains a message to modify old BLS credentials to the new withdrawal credentials in execution address format to facilitate withdrawing balances |

The Beacon Chain relies on a mechanism to generate in-protocol randomness called **RANDAO**. RANDAO serves as an accumulator that continuously accumulates randomness contributions from various sources. The proposer includes a random contribution to the current RANDAO value with each new block. The RANDAO value maintained by the Beacon Chain is updated with each new block added to the chain. This update occurs by mixing in the `randao_reveal` value provided by the block proposing validator. Thus, the RANDAO value gradually accumulates randomness from all block proposers over time.

The attestations field in a block consists of all the attestation associated with the block. The next validator is selected every epoch based on the RANDAO randomness.

The attestation data structure contains several fields as shown below.

| Field | Description |
|---|---|
| `aggregation_bits` | A list of contributing attesting validators |
| `data` | A field consisting of various fields (described next) |
| `signature` | The aggregate signature of all attesting validators |

The data field present in the attestation consists of the elements shown below.

| Field | Description |
|---|---|
| `slot` | The slot number that the attestation is associated with |
| `index` | The indices for attesting validators |
| `beacon_block_root` | The root hash of the beacon block containing the data field |
| `source` | The last justified checkpoint |
| `target` | The latest epoch target block |

The execution payload consists of a header and payload, both of which contain several fields. The payload contains transactions and withdrawals to be processed. When transactions in the execution payload are executed, they result in updating the global state.

All execution clients execute the transactions using the EVM as soon as they receive it from the consensus client to ensure the validity of the transactions. Practically, this means ensuring that the new state resulting from the transaction execution matches the one in the new block's state root field. This verification enables clients to verify that the new block is valid. If it is valid, it's added to the blockchain.

The execution payload header contains several fields, which we describe next.

| Field | Description |
|---|---|
| `parent_hash` | The hash of the parent block |
| `fee_recipient` | The account address of the beneficiary for receiving transaction fees |
| `state_root` | The root hash of the global state after applying changes to this block |
| `receipts_root` | The hash of the transaction receipts trie |
| `logs_bloom` | The data structure containing event logs |
| `prev_randao` | A value used for random validator selection |
| `block_number` | The number of the current block |
| `gas_limit` | The maximum gas allowed in this block |
| `gas_used` | The amount of gas used in this block |
| `timestamp` | The block creation time |
| `extra_data` | Arbitrary additional data as raw bytes |
| `base_fee_per_gas` | The base fee value |
| `block_hash` | Hash of execution block |
| `transactions_root` | Root hash of the transactions in the payload |
| `withdrawals_root` | Root hash of the withdrawals in the payload |

The execution payload contains the same fields as the header including `parent_hash`, `fee_recipient`, `state_root`, `receipts_root`, `logs_bloom`, `prev_randao`, `block_number`, `gas_limit`, `gas_used`, `timestamp`, `extra_data`, `base_fee_per_gas`, and `block_hash` are same as the header except that instead of the root hashes of transactions and withdrawals, it contains the actual transactions and withdrawals to be executed.

| Field | Description |
|---|---|
| `transactions` | List of transactions to be executed |
| `withdrawals` | List of withdrawals to be executed |

There is no shared code between consensus clients and execution clients. They are two separately developed artifacts. However, there are some common tasks that they perform that differ only in terms of the technology stack used. For example, networking and blockchain-related tasks are the same. Under **networking functions**, such as peer discovery, peer management, gossiping, Sybil resistance, and DoS prevention are common in both clients. Under blockchain functionality are functions such as blockchain integrity, e.g., chain reorganization handling, block synchronization, memory pools, state transition handling and relevant functions, and blockchain management.

While the functions they perform are somewhat similar and in some cases the same, the technology that has been used to achieve these is significantly different in many ways. The differences mean that, currently at least, users must run both clients. These differences are shown in the table below.

| Attribute | Execution client | Consensus client |
|---|---|---|
| `Network layer protocol` | `DEVP2P` | `LIBP2P` |
| `Discovery protocol` | `Discovery V4` | `Discovery V5` |
| `Serialization format` | `RLP` | `SSZ` |
| `Hash functions` | `SHA256` | `KECCAK` |
| `Signature scheme` | `ECDSA signature scheme` | `BLS signature scheme` |

This means that in the future, there is a possibility that the two could be merged, and only a single client will emerge that deals with both consensus and execution.

Some possible solutions that have been proposed are:

- A light consensus client built into the execution clients. This approach would make it easier for end users to run a basic node and reduce the system requirements. However, this approach could result in more architectural complexity and security vulnerabilities.
- Combine consensus and execution clients. This approach would make it simple for end users to run a node as it would be just a single executable. However, it could increase the architecture complexity and code complexity, reduce security, lead to no separation of concerns, and cause development complexity.

We may eventually see a single client; however, for now, it is two executables users need to run. Whether with combined or separate clients, the Ethereum ecosystem is set on growth and the future roadmap looks quite promising.

Post-Merge, Ethereum does not use PoW anymore; it uses Gasper – a combination of Casper-FFG and LMD GHOST. Gasper is an integration of **Casper the Friendly Finality Gadget** (**Casper-FFG**) and the LMD- GHOST fork choice algorithm, serving as the consensus mechanism that safeguards PoS Ethereum. The function of Casper-FFG is to promote blocks to a “finalized” state, providing assurance to new participants in the network that they are synchronizing with the authoritative (canonical) blockchain. The fork choice algorithm, on the other hand, leverages accumulated voting to facilitate the selection of the correct blockchain in case of forks. Gasper operates in a PoS environment where nodes (stakers) stake ether as collateral, which can be forfeited if they act sluggishly or dishonestly when proposing or verifying blocks. Gasper defines the rules for rewarding and penalizing validators, determining which blocks to accept or reject, and selecting which branch of the blockchain to continue building more blocks on.

Despite the many positive outcomes of The Merge, there are some issues with Ethereum’s PoS, which we introduce next:

- First, there is a growing concern about centralization. This problem arose because Ethereum PoS required at least 32 ETH to run a validator, which means that users holding less than 32 ETH cannot become validators. They have to resort to using staking services, which leads to centralization due to more and more users relying on these services. There are some top service providers who make up more than 50% of the total stake. This means that, potentially, the control is in the hands of the top few stake service providers, which results in centralization. Also, a single central authority in a worldwide financial network isn’t ideal, and giving someone control over your private keys who could be prone to making mistakes, vulnerable to hacking, penalized for unacceptable behavior by the network protocol, or even censored could be quite detrimental for the investor and the overall Ethereum ecosystem.
- Solo validators could be susceptible to losing keys if not managed appropriately, especially if users are inexperienced. Private key custody is a critical issue, as a solo validator might be vulnerable to malware attacks or lax user behavior. Private keys could be kept with a staking service provider, but that means giving up control of the keys, and stake service providers are also not immune to malware attacks and hacking.

In order to address the post-Merge centralization issue, **Distributed Validator Technology** (**DVT**) has been proposed. It is a technique to distribute the responsibilities of a validator between a network of multiple nodes, which will result in significantly enhancing the overall resilience of the system. In comparison to the traditional method of relying solely on a single machine to run the validator client, the utilization of DVTs offers improved safety and liveness, which not only makes the system less vulnerable to potential problems but also ensures a higher degree of stability and robustness overall.

To understand DVT better, let’s first understand that there are two key risks that validator operators face on Ethereum: key theft and node failure. In the validation process, there are two private keys involved, the signing key and the withdrawal key, which can be stolen if not secured properly. Additionally, node failures can occur due to software bugs, viruses, or internet connectivity issues, resulting in the slashing of the stake. These risks can discourage some users from running a validator node, especially given the high cost of staking, i.e., 32 ETH. As a solution to this financial barrier, several centralized exchanges started to offer custodial staking solutions that allow users with less than 32 ETH to participate as validators.

However, this convenience comes at the cost of trusting the exchange and giving them full control over the ETH deposited. This approach is not in line with the decentralized ethos of blockchain, resulting in centralization. Moreover, validators are single points of failure.

Ethereum validators participate in the **PoS** protocol by signing blocks or attestations, using their private keys. These keys can only be accessed by the validator client software, which is responsible for scheduling the creation and signing of messages in accordance with the duties assigned to the validator. This traditional setup involves several risks, including:

- The centralization of the staking private key in a single location, which increases the likelihood of a potential adversary gaining access to the key and causing conflicts through the creation of conflicting messages, resulting in the slashing of the validator's deposit.
- The necessity for stakers who do not operate their own validator to entrust their staking private key to a third-party stake service provider, creating a dependency on the operator to ensure the security of the key.
- If the network experiences software crashes, loss of network connection, hardware faults, and other similar problems, the validator's stake is reduced due to "sluggish behavior." The failure of the validator client software to create timely messages results in an inactivity leak, which reduces the stake as a penalty.
- There is also a risk to a validator of following a minority fork. This can happen due to a fault in the beacon node to which the validator client is connected, which would make the validator appear offline to the rest of the PoS network.

This is where DVT technology helps as it allows an Ethereum validator to run across multiple nodes instead of only one, resulting in better security, no single point of failure, transparency, and accountability due to the **Istanbul Byzantine Fault Tolerant** (**IBFT**) consensus mechanism running between nodes and improving decentralization. The **Distributed Validator Protocol** provides a way to alleviate the risks and concerns mentioned above and other issues present in traditional validator client setups.

DVT makes use of four key components:

- Distributed key generation to split the private validator key into multiple parts so that it's split across multiple participants to take control of the key away from a single party.
- Shamir's secret sharing scheme to reconstruct the key from multiple parts.
- Multi-party computation, which allows participants to sign messages without rebuilding the entire private key on a single node, thus im-

proving decentralization and the risk of key theft.

- The IBFT consensus algorithm is used to agree on a block by using a beacon node as a proposer and if the majority approves, then the block is added to the chain.

The diagram below shows the high-level architecture of DVT:

*Figure 13.9: Distributed validator technology architecture*

As shown in the diagram in *Figure 13.9*, DVT can be seen as a **Multi-Signature** (**M of N**) scheme to run a validator node.

An **M of N** threshold signature scheme allows a validator's staking key to be divided into **N** parts and each of the co-validators keeps a share. When **M** of the **N** co-validators reach an agreement via the consensus mechanism, the message is signed and, to the beacon nodes, it would simply be a standard and valid signed message by a validator. However, here, the signature is applied after co-validators have agreed via consensus protocol to sign the message.

The IBFT consensus mechanism allows splitting the responsibilities of a single validator into multiple co-validators who coordinate together to achieve an agreement before signing any message. The IBFT consensus protocol also allows the meeting of the safety and liveness requirements of the DV network. These requirements include safety against key theft, safety against slashing, and protection against deadlock, ensuring eventual production of a new attestation or block under a partially synchronous network with the possibility of Byzantine faults.

This combination of consensus and a threshold signature scheme ensures that consensus is secured through cryptography and that decisions are made only when at least the required threshold of co-validators agree.

After this introduction to The Merge and other innovations, let's now have a look at sharding.

## Sharding

Sharding is the main scalability feature in Ethereum. It is a multi-phase enhancement to the Ethereum network to improve its scalability and capacity. This upgrade involves making the Ethereum chain amenable to rollups, makes rollups even more economical, and simplifies the operation of nodes. Sharding enables layer 2 solutions to offer low transaction fees while retaining the security of Ethereum.

The original plan was to introduce 64 separate EVM data shard chains connecting to the Beacon Chain, but this idea is no longer being pursued due to its design complexity, bad user experience, and security challenges.

The new approach is called danksharding. This is the latest scaling solution proposed by Dankrad Feist. The idea here is to shard data instead of many separate EVM shard chains. There is an Ethereum improvement proposal, EIP-4844, that proposes this, however, it is not the complete dankssharding that's proposed in this EIP; instead, it proposes proto-danksharding, which allows for achieving a more feasible scaling solution in the short term before all technical challenges are addressed, before progressing with full danksharding. Proto-danksharding implements some elements of full danksharding and paves the path for full danksharding.

Rollups are the only viable trustless scaling solution for Ethereum in the near and medium-term future, and possibly even longer. High transaction fees on the Ethereum main chain, i.e., layer 1, pose a significant barrier to the adoption of new users and applications. The implementation of EIP-4844 will play a crucial role in fostering a shift toward rollups across the entire Ethereum ecosystem. This is also in line with Ethereum's rollup-centric future. Once EIP-4844 is implemented, it will make layer 2 rollup solutions more amenable to the Ethereum chain and will achieve huge scalability.

Another relevant EIP is EIP-4488, which aims to significantly reduce the gas cost of transaction calldata and also limits the total amount of transaction calldata in a block. This means that the transaction cost of Ethereum layer 2s such as Optimism or zk-rollups like zkSync will also reduce. This is, however, a short-term solution as the eventual solution to this issue will come through sharding. Sharding has not been implemented in the April 2023 Shanghai upgrade.

In later Ethereum upgrades, EIP-4844 implements proto-danksharding, which introduces blob-carrying transactions, which marks the first step toward implementing full sharding on Ethereum. This EIP introduces a new transaction format for "blob-carrying transactions," which can contain a large amount of data that is not executed by the EVM but whose commitment can be accessed. This format is designed to be fully compatible with the format that will be used in full sharding.

Rollups are a promising scaling solution for Ethereum and have already significantly reduced fees for many Ethereum users. However, they are still too expensive for some users. The ultimate solution to the limitations of rollups is data sharding, which would add 16 MB of dedicated data space to the chain per block for rollups to use. However, full data sharding is likely to take a long time to implement completely. This EIP provides a solution in the interim by implementing the transaction format that will eventually be used in full danksharding, but without actually implementing the sharding of transactions. Instead, the data from these transactions will be part of the Beacon Chain and will be fully downloadable by all consensus nodes but can be deleted after a relatively short delay.

Rollups involve posting a large amount of data, which could overburden nodes if they were required to download all of it. This would increase resource requirements and negatively impact decentralization. To address this, **Data Availability Sampling** (**DAS**) enables nodes, including light clients, to easily and securely confirm that all data has been made available without the need to download it all:

*Figure 13.10: Data availability sampling*

Danksharding solves the data availability problem by providing anyone with the ability to download the full data should the need arise. It uses a technique called *DAS* where a client randomly samples an erasure-coded (using Reed-Solomon codes) blob of data to ascertain the availability of data. In other words, the data is erasure-coded and extended using the Reed-Solomon codes, which allows for representing the data as a polynomial and evaluating it at various points. This technique means that now only 50% of the erasure code data is enough to ascertain the full data availability, as the other 50% can be reconstructed.

For an attacker to successfully deceive the DAS mechanism to make nodes think that the full data is available, they would have to conceal more than half of the block. With many random samples taken, the probability of less than half of the data being available becomes negligible. This technique works but the challenge here is how we ensure that the erasure coding is done correctly. What if the block producer extended the block with some useless data and the sampling came back fine, but the actual data was just junk, which cannot be used to reconstruct the data? This is where the KZG commitment scheme helps.

The KZG commitment scheme is a polynomial commitment scheme where a prover can commit to a polynomial. This means that the prover can reveal the value of the polynomial at any given point and prove that it is equal to a claimed value. Once the prover has committed to a value, the prover cannot change the polynomial later. The prover can only provide valid proofs for this specific polynomial and won't be able to produce a proof altogether or, if somehow they do, it won't be accepted by the verifier.

Sharding is also known as a phase named "The Surge," after The Merge.

The upgrades are divided into various phases, as listed below:

- **The Merge**: Already done on 15th September 2022, which switched from PoW to PoS.
- **The Surge**: Achieve scalability by making Ethereum rollup-friendly.
- **The Scourge**: Solve centralization and **Maximal Extractable Value (MEV)**, and achieve neutral transaction inclusion.
- **The Verge**: Make block verification quick and easy by utilizing SNARKs.
- **The Purge**: Remove excess historical data, simplify the protocol, eliminate technical debt, and reduce the cost of participation.
- **The Splurge**: Solve any other remaining issues.

> Note that some of these phases and terminologies could change as the roadmap is evolving; however, any new phases or efforts will be geared toward achieving the eventual goal of scalable Ethereum and making working with the layer 2 protocol easy.

Currently, in layer 2 solutions, the batch of transactions is submitted to layer 1 via *calldata*. However, with the introduction of EIP-4844, a new type of transaction called a "blob-carrying transaction" has been added, which carries “blob” data and is responsible for paying the transaction fee. The “blob-carrying transaction” includes a commitment to prove the existence of data in the blob. It is important to note that the additional content, which is the blob data, is separate from the “blob transaction” and can be considered a *sidecar*.

Danksharding is a simpler sharding design as compared to previous approaches such as multiple shard chains. The key idea here is that instead of increasing the space for transactions, sharding provides more space for data blobs, which are not interpreted by the Ethereum protocol at all. Blob verification simply involves checking that the blob is available and downloadable, which is achieved by DAS. These blobs will be used by layer 2 rollups to enable faster transaction processing and thus scalability.

The lifecycle of a blob transaction is described below:

1. The layer 2 transaction is submitted by the layer 2 users.
2. A rollup provider then creates a transaction batch and submits that to layer 1.
3. This transaction resides in the layer 1 transaction pool as a blob transaction containing blob data.
4. The blob transaction is encapsulated in the execution payload consensus client (Beacon Chain), which processes this and separates out the blob transaction in the execution payload for layer 1 execution and blob data.
5. The execution payload is executed on layer 1 and gets stored there.
6. The blob data is stored on beacon nodes for a month and layer 2 persists this data for future use.

We can visualize this flow at a network level in *Figure 13.11* below:

*Figure 13.11: Blob transaction network flow*

*Figure 13.11* shows several steps, which are explained below:

1. The layer 2 sequencer submits regular transactions and blob-carrying transactions to layer 1, which ends up in the layer 1 transaction pool.
2. The layer 1 execution engine picks up the transaction and reads the execution payload and blobs.
3. The block proposer (consensus client) picks this up and creates a beacon block containing the execution payload. It also separates the blob sidecar from the blob transaction.
4. The consensus peer receives it and the execution payload is sent to the layer 1 execution engine. The execution payload contains transactions for execution.
5. The blob sidecar goes to the layer 2 verifier, which downloads these sidecars and synchronizes the layer 2 nodes.

Note that there is a distinction between the lifetimes of calldata and blob data. Calldata is present in the “execution payload” of a standard layer 1 transaction, whereas blob data is kept in the consensus layer. This means the “blob” is stored in a consensus layer client like Lighthouse or a similar node rather than in the execution layer's Geth or a similar client. The consensus layer nodes discard the blob data after a month.

The primary innovation introduced by danksharding is the **merged fee market**, where a single proposer selects all transactions and data for a specific slot, rather than multiple shards with different block proposers. However, this could lead to high system requirements for validators. To address this issue, a separate set of actors called *block builders* bid for the right to choose the contents of the slot. The *block proposer* only selects the valid header with the highest bid, and the *block builder* processes the entire block. This approach allows other validators and users to efficiently verify the blocks through DAS while reducing the computational burden on validators.

This separation of builders and proposers is called **Proposer-Builder Separation** (**PBS**). It is a promising solution to address the issue of censorship and MEV attacks in blockchain technology. With PBS, the responsibilities of constructing blocks and proposing blocks are separated, with each being assigned to distinct entities/roles within the network. Currently, in Ethereum, the block proposer has the discretion to select which transactions to include in the candidate block by evaluating the transaction in the memory pool and selecting those that have paid the highest fees.

This freedom of the block proposer allows it i.e., the block proposer to employ different sophisticated approaches for unfairly prioritizing transactions or even to include their own, to exploit opportunities such as DEX arbitrage and liquidations, to undeservedly maximize their profits. This is called MEV. Usually, due to the complexities and high cost of executing these strategies, individual validators do not perform these activities; instead, centralized pools see this as an advantage and opportunistically perform this on behalf of their pool participants. To address this issue, PBS has been proposed, which disassociates the role of block construction from that of block proposal. In this scheme, there is a new category of entities referred to as "builders" who:

1. Create units of executable block bodies, which consist of a sequence of transactions.
2. The builders then place bids, and the role of the proposer is simply to select the highest bid and accept the corresponding block body.

It's crucial to note here that the proposer and all other parties remain uninformed about the details of the block body until after the header has been selected. This pre-confirmation confidentiality is the key to preventing MEV exploits.

We can visualize the proto-danksharding EIP 4844 big-picture view in *Figure 13.12* below:

*Figure 13.12: Proto-danksharding overview*

As rollups post large amounts of data, it's not desirable to force nodes to download the entire data to ensure data availability. This enforcement will increase the load on the nodes, resulting in high-end hardware requirements and thus centralization risk as high-end hardware could be within reach of only a few users.

To resolve this, **DAS** has been proposed. A straightforward approach to DAS is randomly checking some parts of the block, and if they are valid, then the decision is made to accept the block. However, what if only that single chunk that contained rogue information is not checked, e.g., an illegal transfer of funds to another party? This would be detrimental to blockchain integrity.

The solution is to use error correcting codes, specifically Reed-Solomon erasure codes, which expand the data and allow data recovery by error correction. This is not a new technique; Reed-Solomon codes are used for error correction in many systems, including communication and storage technologies such as CDs, DVDs, barcodes, satellite communication, and DSL technology.

Using Reed-Solomon erasure codes, the entire block can be reconstructed as long as 50% of the erasure-coded data is available for a block. As we can see, this immediately reduces the storage requirements. From a data availability point of view, we can conclude with a very high probability, after several random samplings, that the complete block is available without needing to download the entire block. So far, so good; however, think about a scenario where the block producer is malicious and produces a rogue block with junk data. This means that while the sampling is okay, the actual data within the block is junk and not erasure-coded correctly.

To ensure that the proposer extended the block correctly with the Reed-Solomon codes, the KZG commitment scheme, a polynomial commitment scheme, is used. Polynomial commitments are discussed in *Chapter 17*, *Scalability*, in more detail.

Mathematically, we want to ensure that the original and extended data are on the same polynomial. The KZG polynomial commitment scheme allows us to commit to the original block data and its extension (from Reed-Solomon codes) and prove that they are on the same low-degree polynomial. Simply, it means the data using Reed-Solomon codes is extended correctly. So, in summary, DAS ensures the availability of erasure-coded data, and the KZG commitment proves that the original data was correctly extended.

As the first step in proto-danksharding, Ethereum validators must download shard blobs fully; however, when full danksharding is finally implemented, the validator only has to do data availability sampling.

Currently, in Ethereum, validators do two tasks: build and propose blocks, which leads to MEV. **PBS** splits these two tasks and assigns block building to builders, and proposers select the blocks. The idea here is that builders can be specialized high-power nodes or even a single powerful node that provides censorship resistance and liveness. Still, proposers are general-purpose and lightweight, meaning even users with commodity hardware can participate, thus improving decentralization. Builders in PBS have more power, which could lead to transaction censorship. The crList or censorship resistance list solution is proposed to address this issue. The idea is that proposers publish transactions from the memory pool, and builders are algorithmically mandated to include those in a block.

The proposed scheme is a hybrid PBS and comprises several steps described below:

1. The proposer publishes a censorship resistance list (crList) and its summary with all qualified transactions from the memory pool.
2. The builder proposes a block body and submits a bid that includes the hash of the crList summary, which proves that they have observed it.
3. The builder wins the bid and submits the block body. Note that the builder has yet to see the block body; it is not aware of any transactions in the block body.
4. The builder posts their block with proof that they have included all the transactions from the crList.
5. If the proof is valid, the LMD GHOST fork choice rule accepts the block, and attesters validate the published block body.

In summary, PBS allows for a scenario where, effectively, even if block production becomes centralized due to high-end hardware requirements, block verification (validation) is still decentralized and trustless while preventing transaction censorship. This scheme's overall effect still decentralizes the network, even though builders are centralized. The key idea to remember here is that block verification decentralization is more important than block production decentralization. Ethereum is eventually expected to evolve into such a mechanism by adopting danksharding and PBS. With danksharding, Ethereum will become a unified data availability and settlement layer.

Danksharding creates a tighter integration of the Beacon Chain execution block and shards. A single builder creates the entire block, and a proposer and a committee vote on it at a time. PBS is required for danksharding, as regular validators cannot handle a block full of blobs of data. With this tighter integration of execution blocks and shards, data availability can be ensured in a single go. With the introduction of large data blobs, effectively, an identical effect has been achieved, as if block size has been increased; however, the EVM does not touch the data blob at all.

We introduced LMD GHOST earlier. It is part of the Gasper consensus protocol. Gasper is the new consensus mechanism that secures the Ethereum PoS blockchain after The Merge. It is the combination of Casper-FFG, and the fork choice algorithm called LMD GHOST. The Casper-FFG mechanism finalizes blocks, ensuring new network joiners synchronize to the correct chain.

On the other hand, the fork choice algorithm uses accumulated votes to assist nodes in selecting the correct chain from forks. Gasper determines the incentives and penalties for validators, chooses which blocks to approve or decline, and picks the correct blockchain fork to use and build on. As post-Merge Ethereum is PoS-b based, the validators provide ether as a security deposit, which can be forfeited if inactive or dishonest in their block validation or proposal duties.

It's worth noting that there is no need for a fork choice rule in normal conditions where there's a single block proposer for each slot, and honest validators confirm it. However, the fork choice algorithm becomes critical in cases of significant network asynchrony or when a block proposer behaves dishonestly by making inconsistent claims. In those cases, the fork choice algorithm is essential in securing the correct blockchain.

As the first step to achieving danksharding, proto-danksharding is introduced in EIP-4844. It implements the logic, transaction formats, and verification rules that are in the full danksharding specification but does not implement the actual sharding yet. This means users still must directly validate the complete data availability. A key feature introduced by this EIP is a new transaction type called a “blob-carrying transaction”. This transaction is similar to a regular Ethereum transaction, but it also has an extra piece of data called a “blob,” which is roughly 125 kilobytes in size.

The key advantage here is that instead of using *calldata* to write data from rollups, these blobs can contain data from rollups, which can be a much cheaper option compared to *calldata*. Note that the EVM doesn’t have access to this blob; it can only view a commitment to the blob. This approach results in scalability improvement because blobs themselves do not consume any gas and rollups can use them for data.

So, in summary, just a few key points to remember:

- Danksharding focuses primarily on data availability instead of code execution.
- The eventual aim of danksharding is to reduce the cost of rollups significantly for layer 2s and establish Ethereum’s data settlement and availability.
- Proto-danksharding is the first step toward achieving danksharding, implementing many foundational functions.
- EIP-4488 implements a reduction in calldata cost from 16 gas units to 3 gas units per byte and puts a limit on the maximum calldata per block.
- EIP-4844 implements proto-danksharding, which introduces a new type of transaction called a blob-carrying transaction, which serves a similar purpose as calldata, i.e., submitting rollup batches to layer 1 for data availability, but by using blobs of data instead of using calldata, almost like a large-sized transaction, but without impacting the execution layer, as blobs are stored in beacon nodes.

In future, full danksharding will be implemented, which will make Ethereum a decentralization-and security-focused settlement and data availability layer.

We can visualize this in *Figure 13.13* below:

*Figure 13.13: Danksharding – what Ethereum looks like after full danksharding*

As shown in the preceding diagram, eventually sharding will be achieved at layer 1, with layer 2 EVMs writing blobs of data to layer 1, while Ethereum layer 1 becomes the base settlement and data availability layer, consisting of the layer 1 consensus and execution layer.

Let’s now focus on what is the future road map of Ethereum and what to expect next.

## The future roadmap of Ethereum

The Shanghai hard fork upgrade (also called Shapella due to combining the Capella upgrade on the Beacon Chain and the Shanghai upgrade on the execution layer) is scheduled for April 2023 and is the most important upgrade since The Merge. This includes several updates to reduce gas costs on Ethereum.

As we know, The Merge occurred in September 2022 and the Ethereum network transitioned to a PoS consensus mechanism from PoW. With the switch to PoS, users participate in validation by staking 32 ETH instead of using specialized mining hardware. However, after The Merge with the PoS Beacon Chain, users did not have any option to withdraw their staked funds. This changes with the Shanghai update (EIP-4895 – `https://eips.ethereum.org/EIPS/eip-4895`), which adds the much-needed withdrawal functionality.

It is important to acknowledge that eventually some degree of centralization may be required to attain high scalability. However, the Ethereum community is trying to avoid it as much as possible. Even if decentralization is sacrificed a bit, it is crucial to maintain the property of trustless validation and censorship resistance in the blockchain. The implementation of concepts such as PBS and weak statelessness facilitates a division between the building and validation processes, thereby facilitating scalability while preserving security and decentralization.

Current problems that Ethereum face are network congestion and excessive disk space usage. Both these problems mean that there is a sheer need to increase the speed of the network and reduce disk space usage. A simple approach to addressing these issues would be to increase centralization; however, decentralization is critical as it provides impartiality, resistance to censorship, openness, data ownership, and robust security. The aim is to be more scalable and secure but also maintain decentralization, which is not easy to achieve and has been called the “scalability trilemma.” However, Ethereum appears to be on the right path, and its upgrades aim to resolve the scalability trilemma by making Ethereum more scalable, secure, and decentralized.

Scalability is achieved by sharding, which increases transaction throughput and will also reduce the validator node power requirements. Moreover, staking lowers the barriers to participation, which will result in creating a more decentralized network. We will discuss the scalability trilemma in detail in *Chapter 17*, *Scalability*.

For rollups to function optimally, cost-effective storage at the layer 1 level is required. Post-Merge upgrades, especially danksharding, will provide this and create ample room for storage, which will allow Ethereum to grow by maximizing the efficiency of rollups, enabling exponential speedups. More on rollups will be covered in *Chapter 17*, *Scalability*.

## Summary

In this chapter, we looked at Ethereum after The Merge and its roadmap. We explored various aspects of its architecture and the overall vision behind the Ethereum upgrades to solve the scalability trilemma. We covered the Beacon Chain, The Merge, and sharding. We also touched upon some relevant concepts such as danksharding, PBS, and the distributed validator protocol, all leading to a more scalable and secure future for Ethereum. In the next chapter, we'll introduce Hyperledger, which is a very active and popular blockchain project upon which multiple blockchain projects are being built.

## Join us on Discord!

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:

https://packt.link/ips2H