Merkle-Patricia Tries:

It is comprised of two components - Merkle trees and Patricia tries. The state of Ethereum (the totality of all accounts, balances, and smart contracts), is encoded into these data structures (although the actual data is stored on some database). In Merkle trees, each node is represented by its hash values and the the root node acts as a fingerprint for the entire dataset. The Patrcia-trie component helps in efficient data retrieval of items that comprise the Ethereum state. The only way to generate a state root is by computing it from each individual piece of the state, and two states that are identical can be easily proven so by comparing the root hash and the hashes that led to it (*a Merkle proof*). This data structure allows inserts, deletes, updates in O(log(n)) time and it takes O(n) time to build the entire data structure.

The structure of a Merkle Trie makes witness sizes very large - too large to safely broadcast between peers within a 12 second slot. This is because the witness is a path connecting the data, which is held in leaves, to the root hash.

Verkle Trees:

It is a portmanteau of two words – vector commitment and Merkle trees. This data structure can be used to upgrade Ethereum nodes so that they can stop storing large amounts of state data without losing the ability to validate blocks. Verkle trees are (key, value) pairs where the keys are 32-byte elements composed of a 31-byte *stem* and a single byte *suffix*. The main difference between the Verkle tree and the Merkle tree structure is that the Verkle tree is much flatter, meaning there are fewer intermediate nodes linking a leaf to the root, and therefore less data required to generate a proof. A Verkle Tree with branching factor k achieves O(kn) construction time and $O(\log_k n)$ membership proof-size. This means that the branching factor, k, offers a tradeoff between computational power and bandwidth