**Lecture Notes:**  
  
**CAP Theorem:**



**Consistency:** All nodes see the same data at the same time. This means users can read or write from/to any node in the system and will receive the same data. It is equivalent to having a single up-to-date copy of the data.

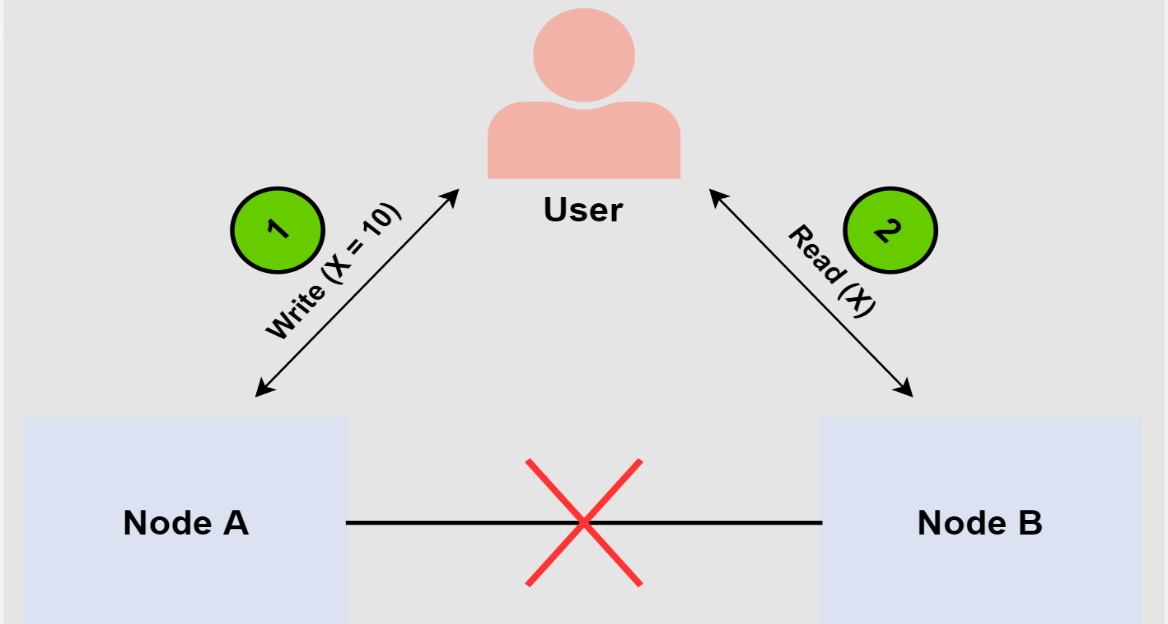
**Availability:** Availability means every request received by a non-failing node in the system must result in a response. Even when severe network failures occur, every request must terminate. In simple terms, availability refers to a system’s ability to remain accessible even if one or more nodes in the system go down.

**Partition tolerance:** A partition is a communication break (or a network failure) between any two nodes in the system, i.e., both nodes are up but cannot communicate with each other. A partition-tolerant system continues to operate even if there are partitions in the system. Such a system can sustain any network failure that does not result in the failure of the entire network. Data is sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages.

According to the CAP theorem, any distributed system needs to pick two out of the three properties. The three options are CA, CP, and AP. However, CA is not really a coherent option, as a system that is not partition-tolerant will be forced to give up either Consistency or Availability in the case of a network partition. Therefore, the theorem can really be stated as: In the presence of a network partition, a distributed system must choose either Consistency or Availability.

We cannot build a general data store that is continually available, sequentially consistent, and tolerant to any partition failures. We can only build a system that has any two of these three properties. Because, to be consistent, all nodes should see the same set of updates in the same order. But if the network loses a partition, updates in one partition might not make it to the other partitions before a client reads from the out-of-date partition after having read from the up-to-date one. The only thing that can be done to cope with this possibility is to stop serving requests from the out-of-date partition, but then the service is no longer 100% available.

**CAP Theorem proof:**  
Let’s look at a simple proof of the CAP theorem. Imagine a distributed system consisting of two nodes:



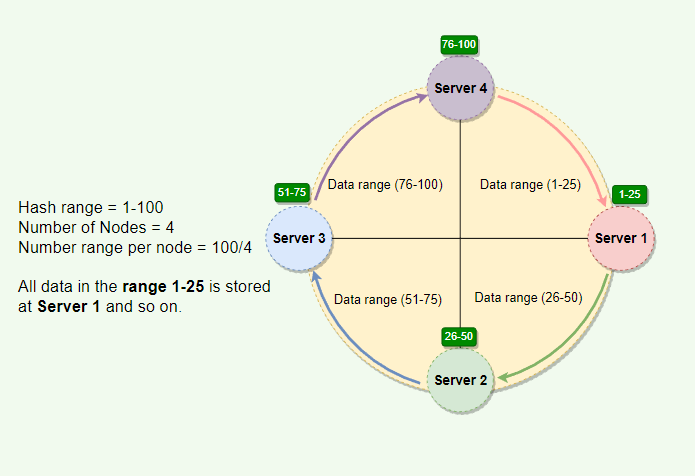
The distributed system acts as a plain register with the value of variable X. There’s a network failure that results in a network partition between the two nodes in the system. An end-user performs a write request, and then a read request. Let’s examine a case where a different node of the system processes each request. In this case, our system has two options:

* It can fail at one of the requests, breaking the system’s availability.
* It can execute both requests, returning a stale value from the read request and breaking the system’s consistency.

The system can’t process both requests successfully while also ensuring that the read returns the latest value written by the write. This is because the results of the write operation can’t be propagated from node A to node B because of the network partition.

**Consistent Hashing:**

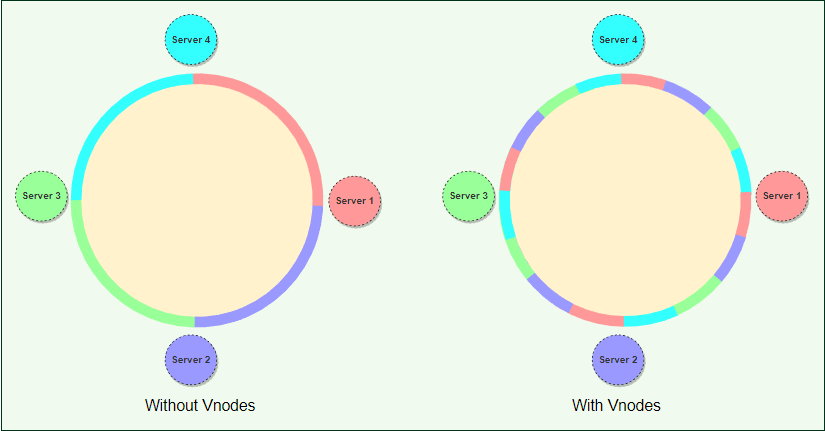
Use the Consistent Hashing algorithm to distribute data across nodes. Consistent Hashing maps data to physical nodes and ensures that only a small set of keys move when servers are added or removed.

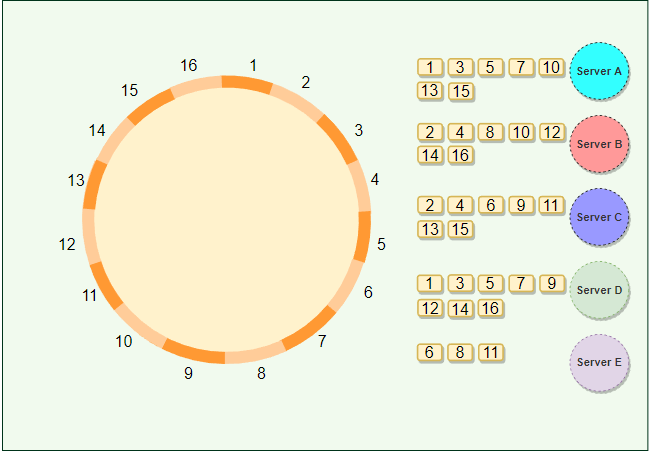
  
  
Issues with basic consistent Hashing:

Adding or removing nodes: Adding or removing nodes will result in recomputing the tokens causing a significant administrative overhead for a large cluster.

Hotspots: Since each node is assigned one large range, if the data is not evenly distributed, some nodes can become hotspots.

Node rebuilding: Since each node’s data might be replicated (for fault-tolerance) on a fixed number of other nodes, when we need to rebuild a node, only its replica nodes can provide the data. This puts a lot of pressure on the replica nodes and can lead to service degradation.





**Advantages of Vnodes**

Vnodes gives the following advantages:

* As Vnodes help spread the load more evenly across the physical nodes on the cluster by dividing the hash ranges into smaller subranges, this speeds up the rebalancing process after adding or removing nodes. When a new node is added, it receives many Vnodes from the existing nodes to maintain a balanced cluster. Similarly, when a node needs to be rebuilt, instead of getting data from a fixed number of replicas, many nodes participate in the rebuild process.
* Vnodes make it easier to maintain a cluster containing heterogeneous machines. This means, with Vnodes, we can assign a high number of sub-ranges to a powerful server and a lower number of sub-ranges to a less powerful server.

* In contrast to one big range, since Vnodes help assign smaller ranges to each physical node, this decreases the probability of hotspots.