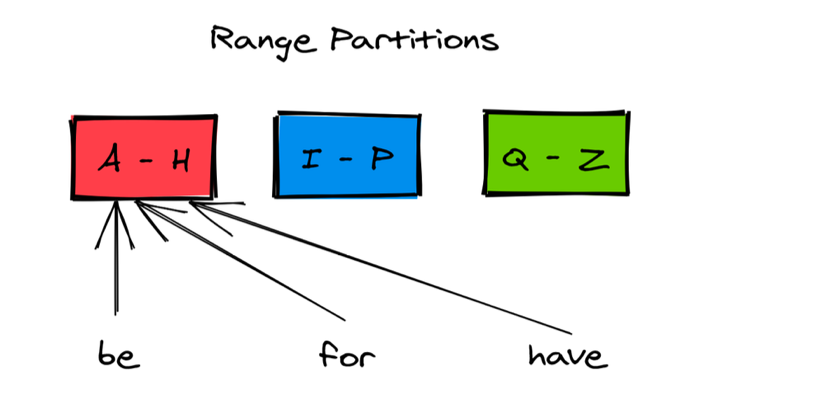
**Partitioning**

1. **How to Partition:** Use a gateway service that can route a request to the node responsible for the partition that the key belongs to. The mapping between keys to partitions can be stored in a strongly consistent configuration store like zookeeper or etcd. How to partition keys:
   1. **Range based partitioning:** Problems: Unbalanced partitions. Some partitions can hold a lot more data than others. Hotspots. Even if the data is uniformly distributed across partitions, some access patterns can hit the same partitions. For example, if A-H belong to partition 1, and the user access a lot of keys starting with A, more requests will hit the same partition. Similarly, f a dataset is range partitioned by date, all writes for the current day end up in the same partition, which degrades the data store’s performance.



* 1. **Hash Based Partitioning:** hash(key) mod N. Problems: If a new node is introduced or deleted, a lot of reshuffling of keys is required.

Shape, polygon

Description automatically generated

1. **Consistent Hashing:** Minimises data movement problems stemming from Hash Based Partitioning.

**Diagram

Description automatically generated**

Now if a new node is introduced, only a small amount of data needs to move

Diagram

Description automatically generated

1. **Rebalancing:** If the dataset increases, new node needs to introduced to keep load on existing nodes under control. If dataset reduces, nodes need to be removed to reduce costs. One simple solution to create a lot of logical partitions and assign them to the existing nodes so that a physical node may contain more than one logical partitions. When a new node joins, a partition can be moved to that node. Drawback: Too many logical partitions create overhead while few partitions limit the data store scalability.