Consistent Hashing

To achieve horizontal scaling, it is important to distribute requests/data efficiently and evenly
across servers. Consistent hashing is a commonly used technique to achieve this goal.

The Problem with Traditional Hashing

If you have n cache servers, a common way to balance the load is to use the following hash
method:

> 💡 **serverIndex = hash(key) % N**
>
> where N is the size of the server pool.

key	hash	hash % 4
key0	18358617	1
key1	26143584	0
key2	18131146	2
key3	35863496	0
key4	34085809	1
key5	27581703	3
key6	38164978	2
key7	22530351	3

This approach works well when the size of the server pool is fixed, and the data distribution

is even. However, problems arise when new servers are added, or existing servers are
removed.

For example, if server 1 goes offline, the size of the server pool becomes 3. Using the same hash function, we get the same hash value for a key. But applying modular
operation gives us different server indexes because the number of servers is reduced by 1. We
get the results as shown in Table 5-2 by applying hash % 3:

key	Hash	hash % 3
key0	18358617	0
key1	26143584	0
key2	18131146	1
key3	35863496	2
key4	34085809	1
key5	27581703	0
key6	38164978	1
key7	22530351	0

As shown, most keys are redistributed, not just the ones originally stored in the offline server (server 1). This means that when server 1 goes offline, most cache clients will
connect to the wrong servers to fetch data. This causes a storm of cache misses. Consistent
hashing is an effective technique to mitigate this problem.

Consistent hashing

Consistent hashing is a special kind of hashing such that when a hash table is re-sized and consistent hashing is used, only k/n keys need to be remapped on average, where k is the number of keys, and n is the number of slots.

How Consistent hashing Works

Consistent hashing is a **distributed hashing technique** used to efficiently distribute data across multiple nodes (servers, caches, etc.).

It uses a **circular hash space** (hash ring) with a large and constant hash space.

Both nodes (servers, caches, or databases) and keys (data items) are mapped to positions on this hash ring using a **hash function**.

Constructing the Hash Ring

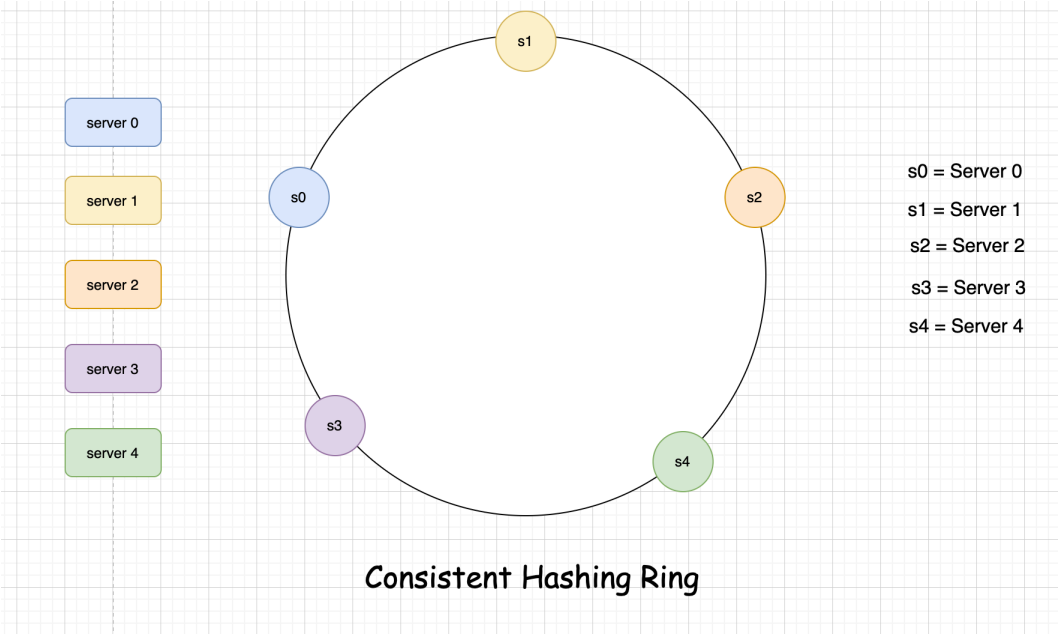
Instead of distributing keys based on `Hash(key) mod N`, consistent hashing places both servers and keys on a circular hash ring.

Defining the Hash Space

- We use a large, fixed hash space ranging from `0` to `2^32 - 1` (assuming a 32-bit hash function).

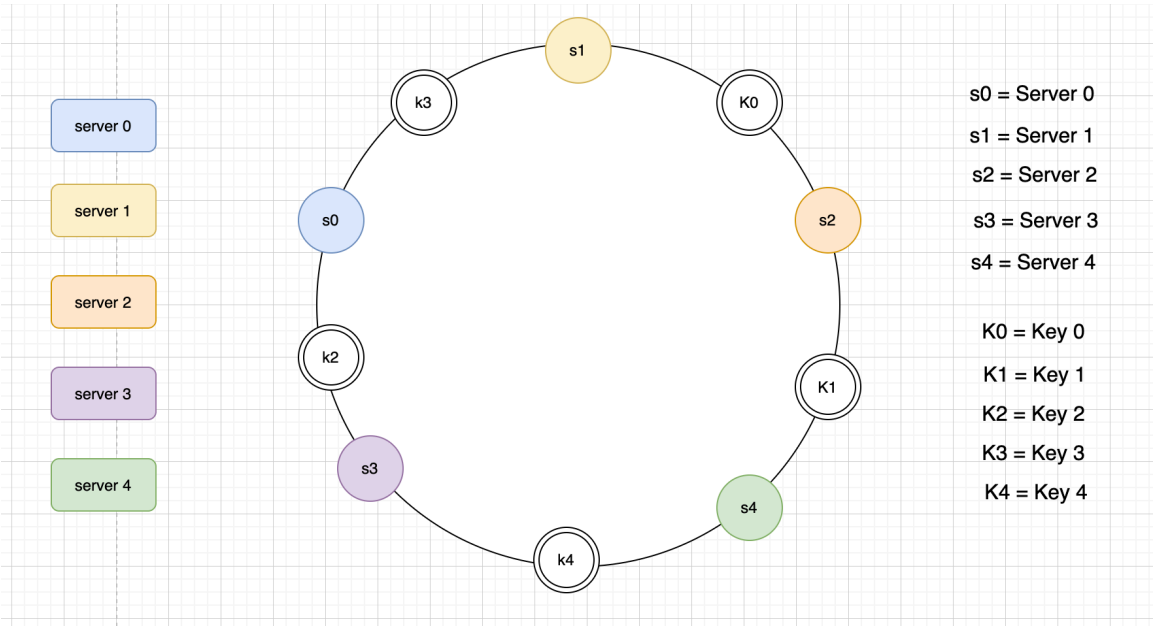
Hash Servers

Using the same hash function f, we map servers based on server IP or name onto the ring.



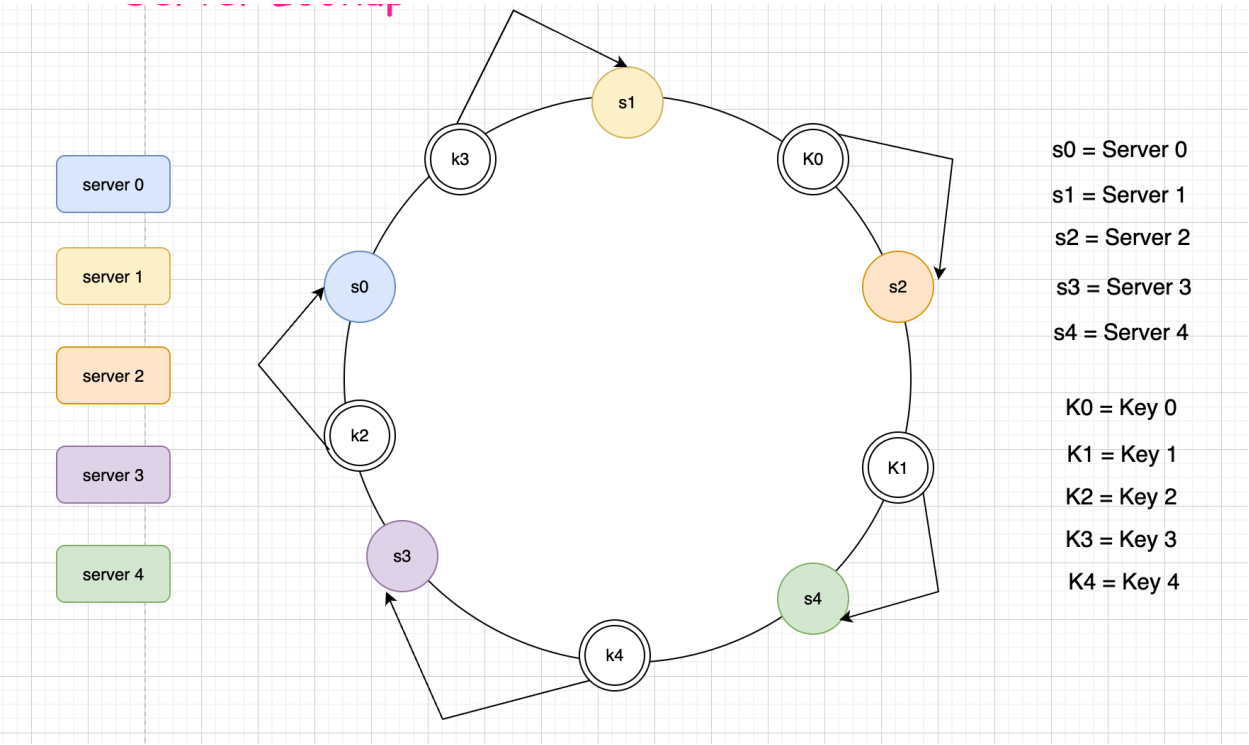
Hash Keys

One thing worth mentioning is that hash function used here is different from the one in “the
rehashing problem,and there is no modular operation. As shown in Figure, 4 cache keys
(key0, key1, key2, and key3) are hashed onto the hash ring



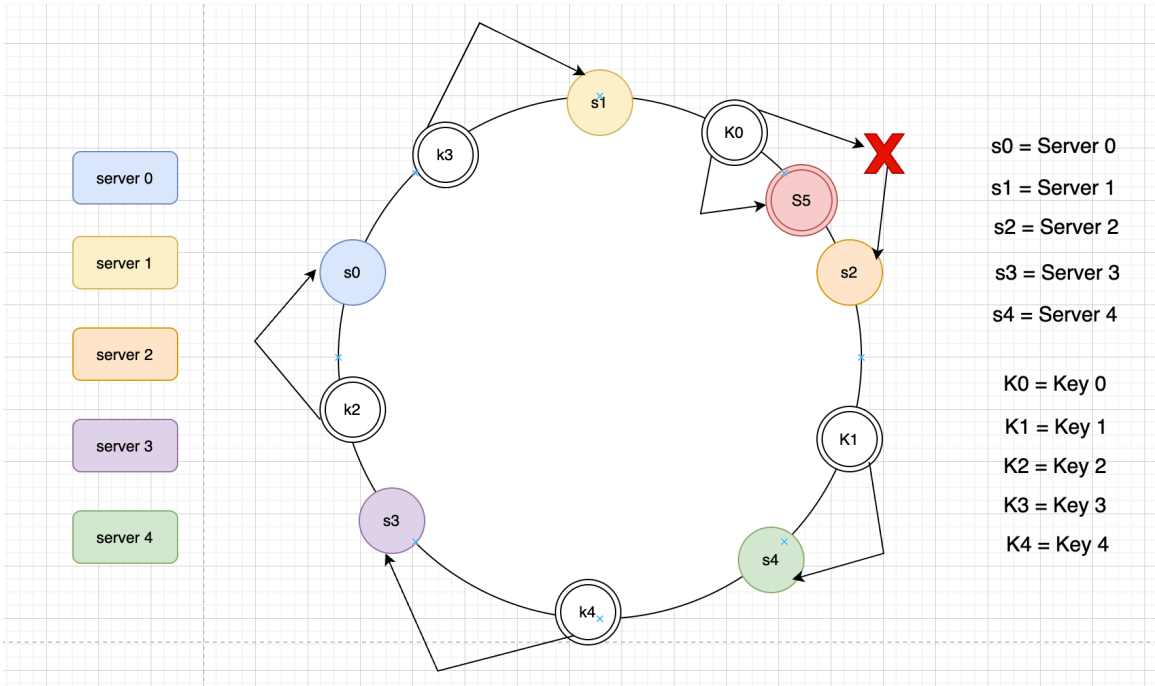
Server Lookup

To determine which server a key is stored on, we go clockwise from the key position on the
ring until a server is found.



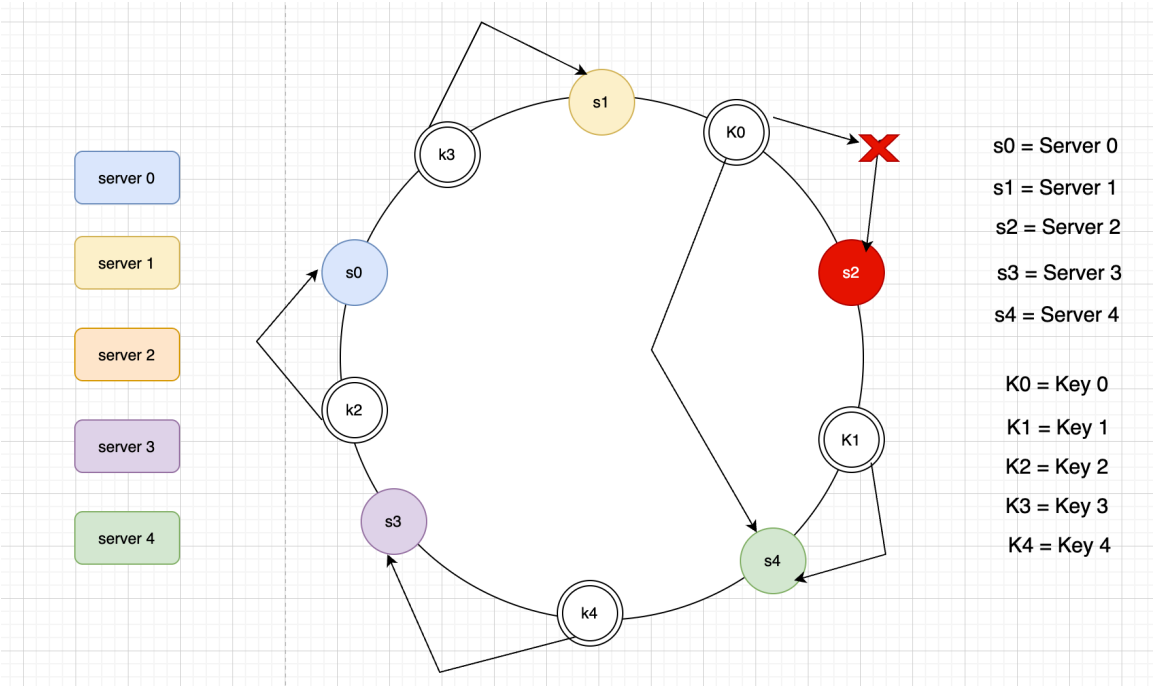
Adding Server

Using the logic described above, adding a new server will only require redistribution of a
fraction of keys.



Removing Server

Using the logic described above, removing a server will only require redistribution of a
fraction of keys.



Virtual Nodes

In **basic consistent hashing**, each server is assigned **a single position** on the hash ring. However, this can lead to **uneven data distribution**, especially when:

- The number of servers is small.
- Some servers accidentally get clustered together, creating **hot spots**.
- A server is **removed**, causing a sudden load shift to its immediate neighbor.

Virtual nodes (VNodes) are a technique used in consistent hashing to improve **load balancing** and **fault tolerance** by distributing data more evenly across servers.

How Virtual Nodes Work

Instead of assigning one position per server, each physical server is assigned **multiple positions** (virtual nodes) on the hash ring.

- Each server is hashed multiple times to different locations on the ring.
- When a request (key) is hashed, it is assigned to the next virtual node in a clockwise direction.
- The request is then routed to the actual server associated with the virtual node.

Virtual Nodes