Design 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 Rehashing Problem
  - If you have n cache servers, a common way to balance the load is to use following hash method:
    serverIndex = hash (key) % N, N - Size of server pool

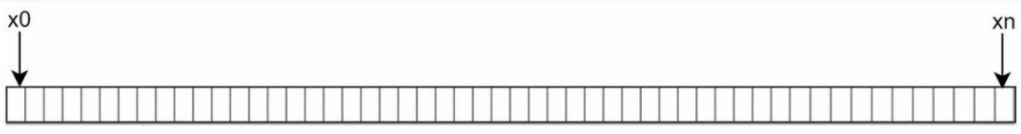
key	hash	hash % 4
key0	18358617	1
key1	26143584	0
key2	18131146	2
key3	35863496	0
key4	34085809	1
key5	27581703	3
key6	38164978	2
key7	22530351	3

  - This approach works well when the size of server pool is fixed, and data distribution is even. However, problem arises when servers are added or removed.

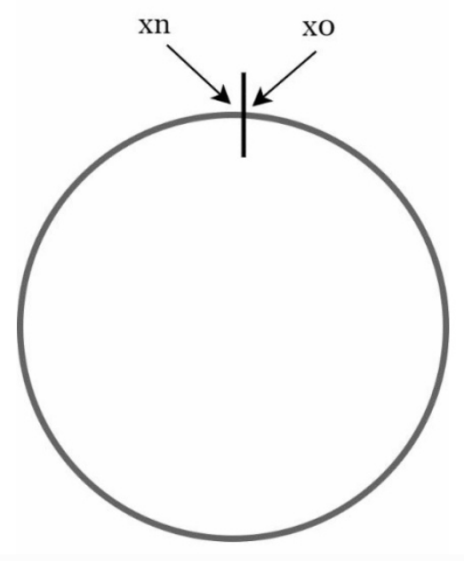
- Consistent Hashing
  - Consistent hashing is a special kind of hashing such that when a hash table is re-sized and consisted hashing is used, only k/n keys need to be remapped on average, where k is number of keys and n is the number of slots. In

contrast, most traditional hash tables, a change in number of array slots causes nearly all keys to be remapped.

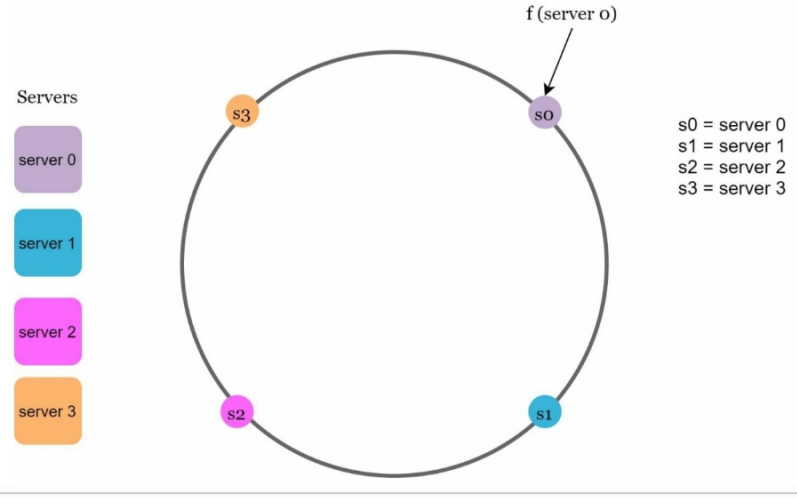
- Hash Space and Hash Ring
  - Assume SHA-1 is used as the hash function f, and the output range of the hash function is: $x_0, x_1 \ldots x_n$. SHA-1 hash space goes from 0 to 2^160 - 1.



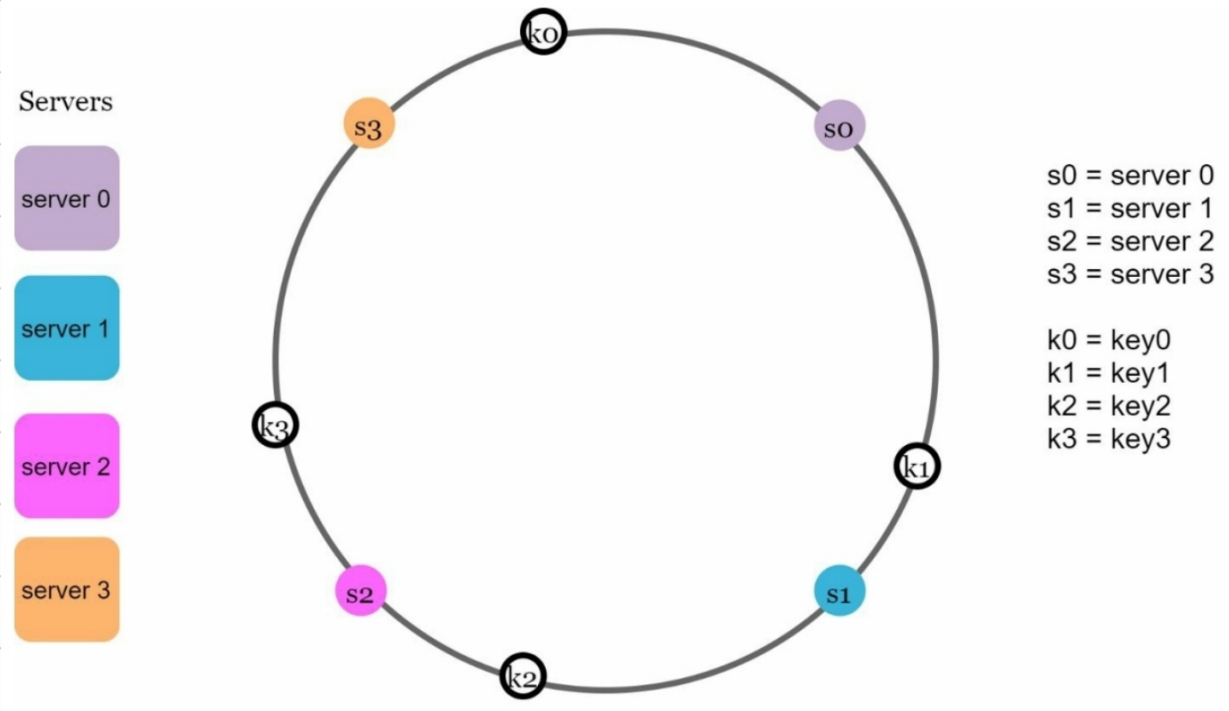
  - By connecting both ends, we get a hash ring.



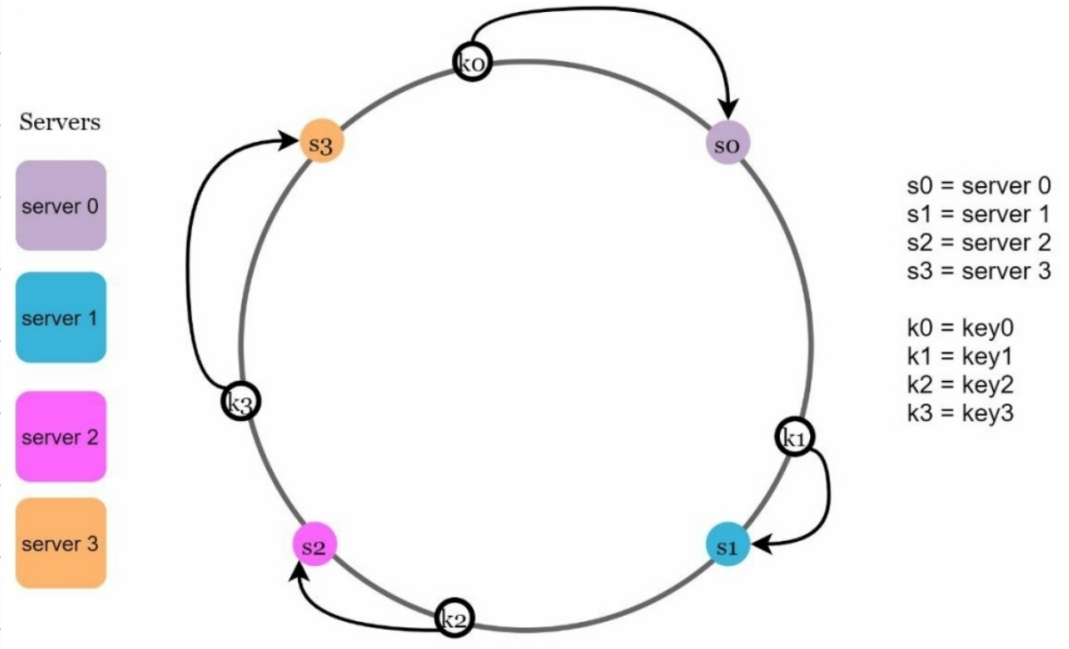
- Hash Servers
  - Using the same hash function f, servers are mapped based on server IP or name onto the ring.



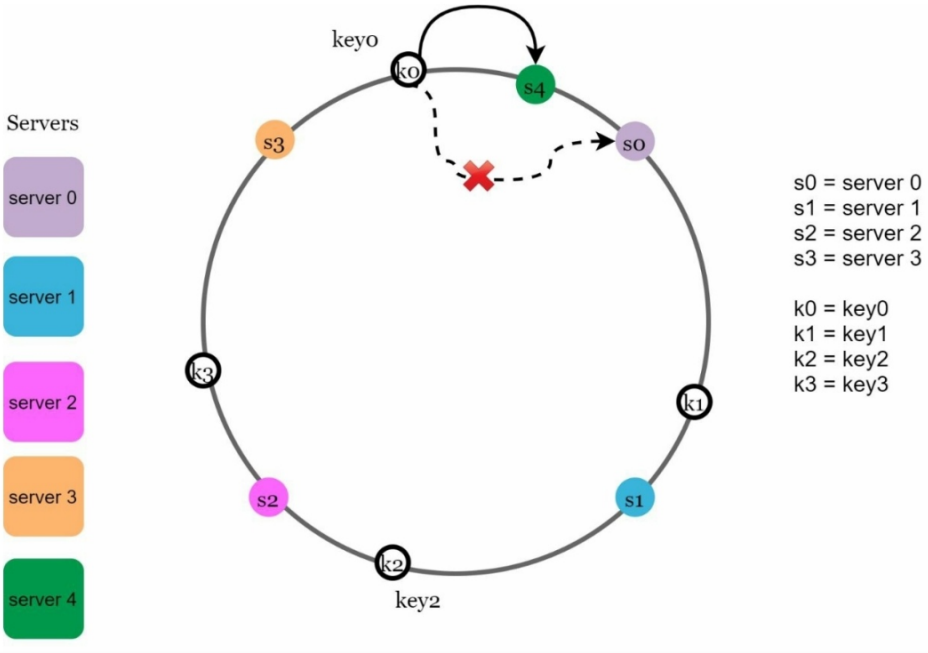
- Hash Keys
  - Hash functions used here is different from the one in "the rehashing problem" and there is no modular operation.
  - Cache keys are hashed onto the hash ring.



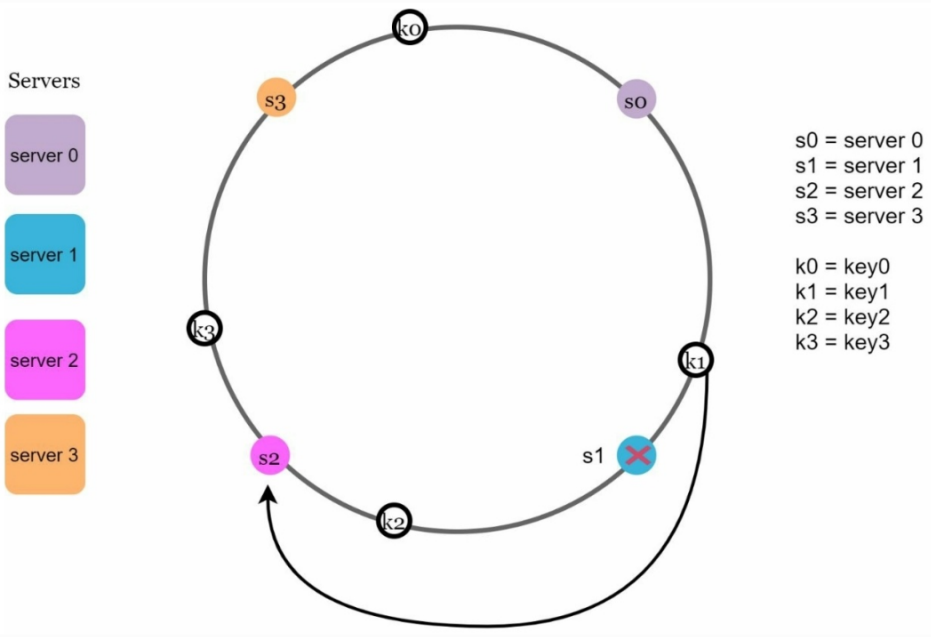
- Server Lookup
  - To determine which server key is stored on, we go clockwise from the key position on the ring until a server is found.



- Add a Server
  - Adding a new server will only require redistribution of a fraction of keys.
  - After a new server 4 is added, only key 0 needs to be redistributed. k1, k2, k3 remain on the same servers. Before server 4 is added, key 0 is stored on server 0. Now, key0 will be stored on server 4 because server 4 is the first server it encounters by going clockwise from key0's position on the ring.
  - The other keys are not redistributed based on consistent hashing algorithms.

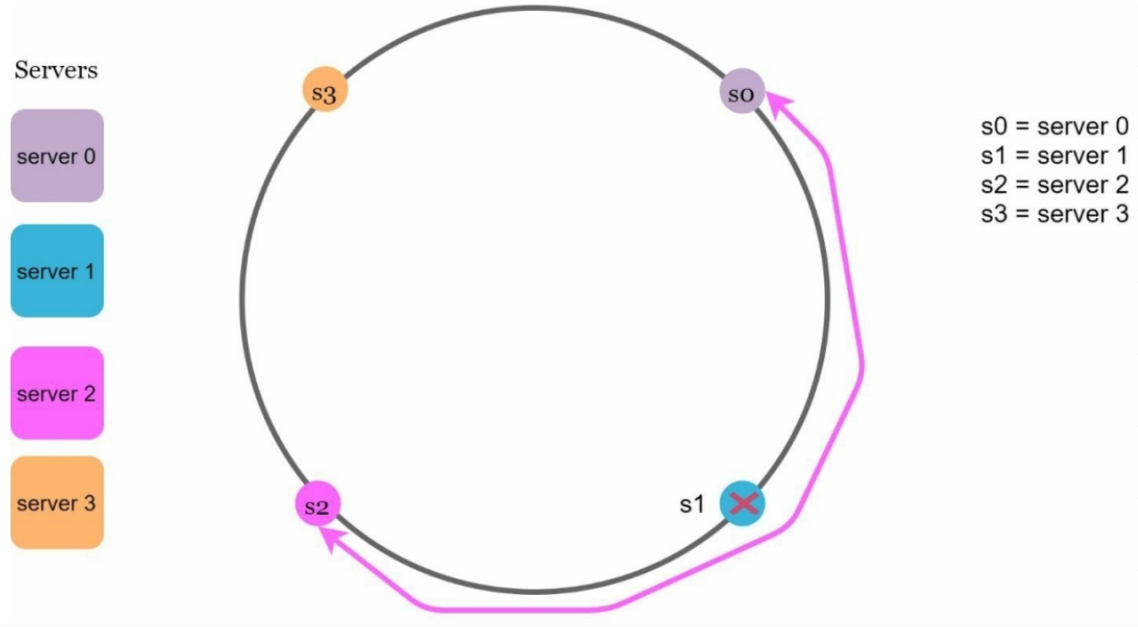


- Remove a Server
  - When a server is removed, only a fraction of keys require redistribution with consistent hashing.
  - When server 1 is removed, only key1 must be remapped to server 2.
  - The rest of the keys are unaffected.

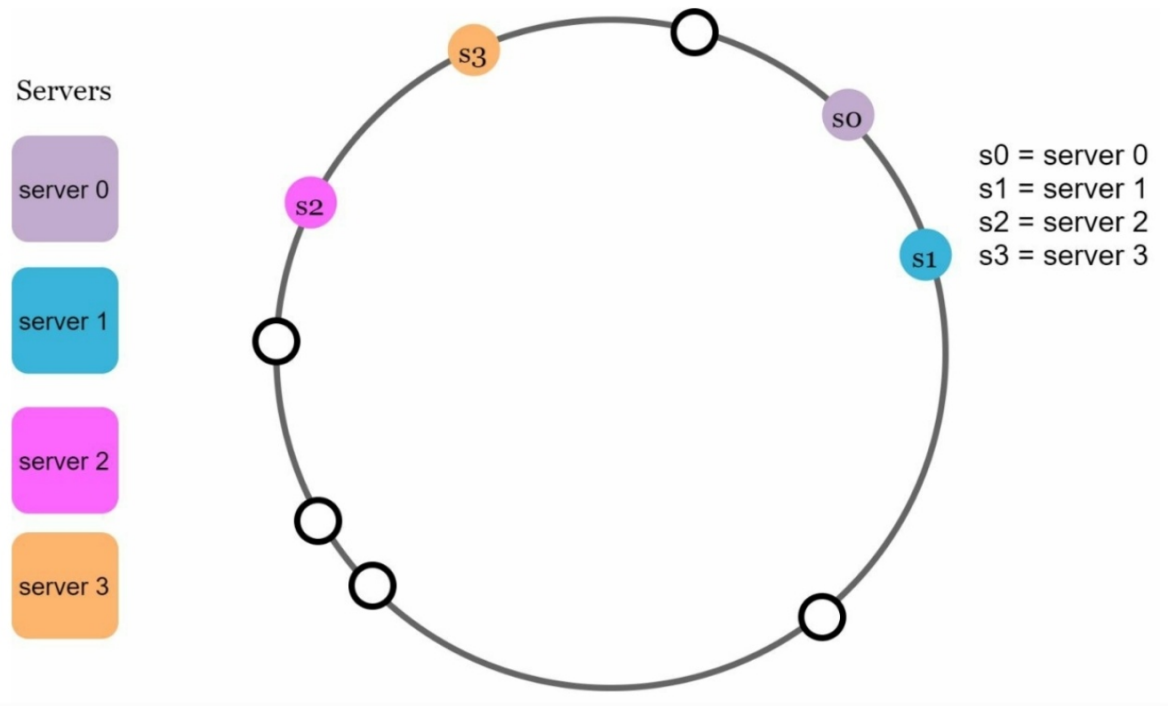


- Two issues in the basic approach
  - Basic steps for consistent hashing algorithms are:
    - Map servers and keys onto the ring using a uniformly distributed hash function.
    - To find out which server a key is mapped to, go clockwise from the key position until the first server on the ring is found.

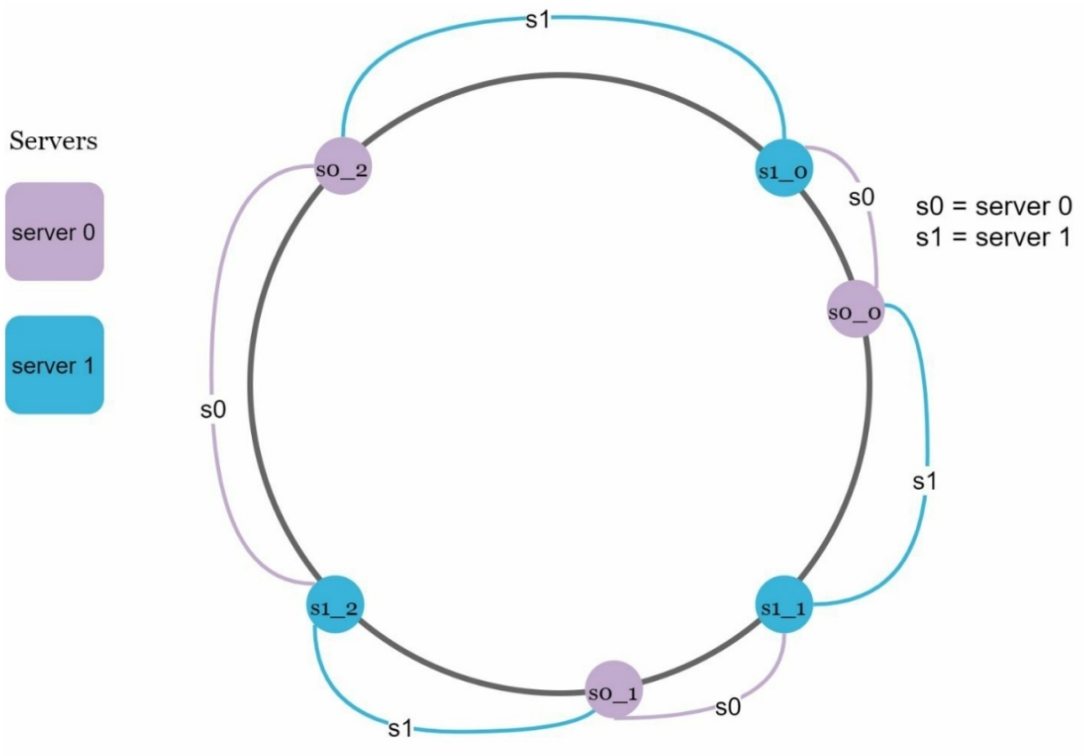
  - First, it is impossible to keep the same size of partitions on the ring for all servers considering a server can be added or removed.



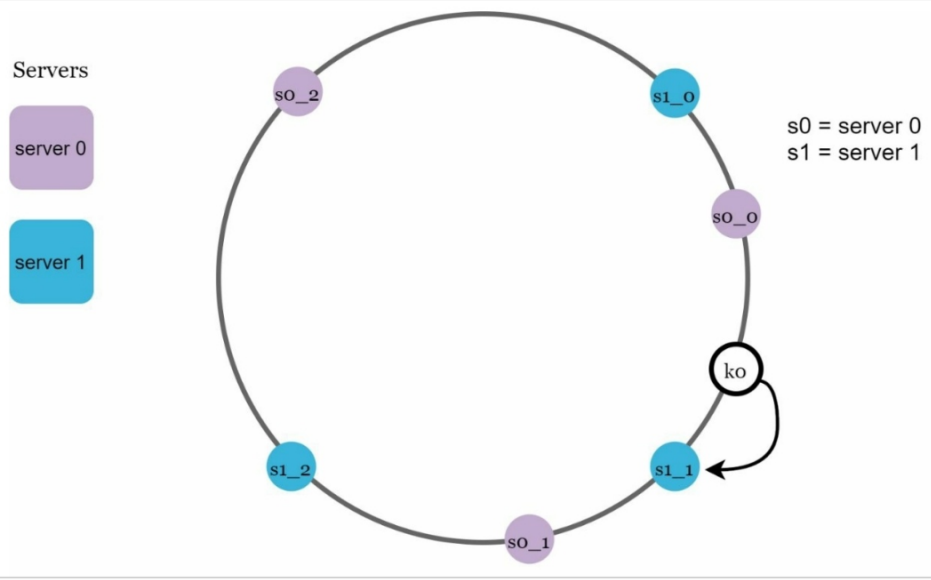
- Secondly, it is possible to have a non-uniform key distribution on the ring.



- Virtual Nodes
  - A virtual node refers to the real node, and each server is represented by multiple virtual nodes on the ring.
  - With virtual nodes, each server is responsible for multiple partitions.

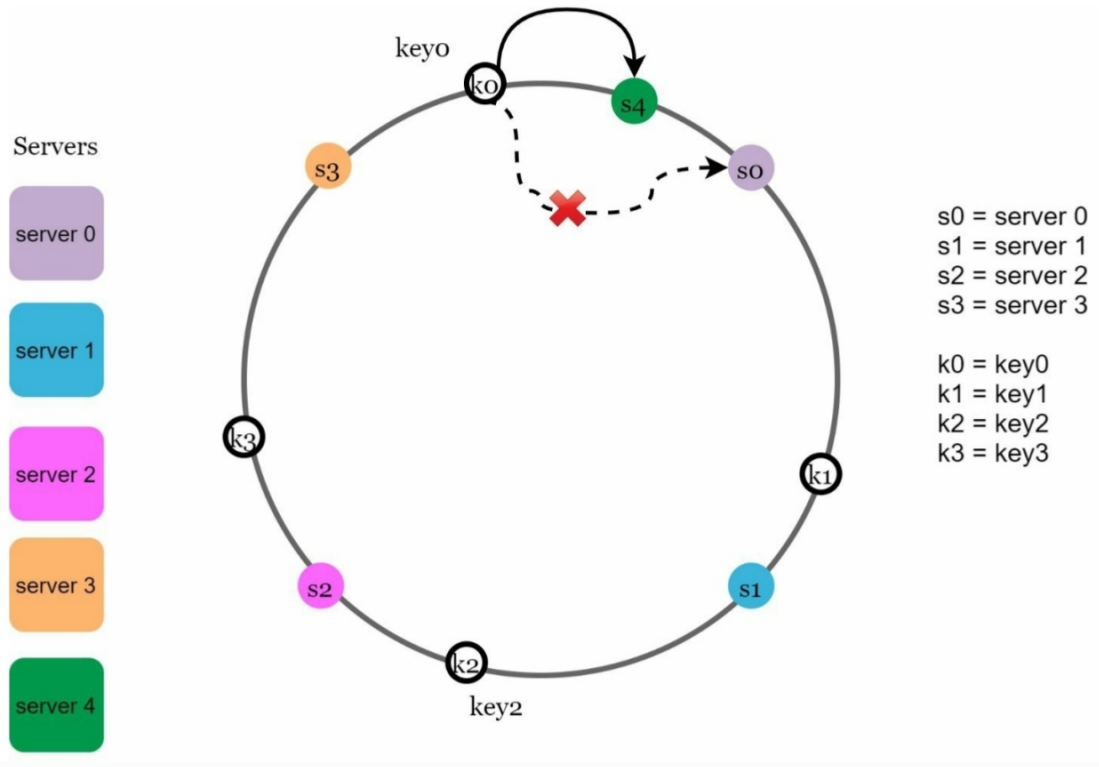


- To find which server a key is stored on, we go clockwise from the key's location and find the first virtual node encountered on the ring.

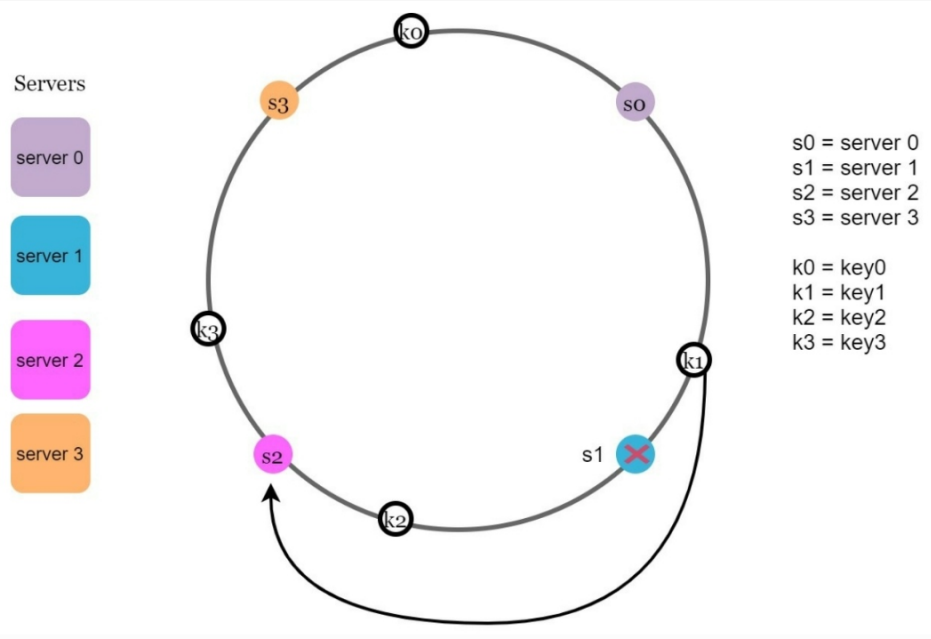


- As the number of virtual nodes increases, the distribution of keys becomes more balanced.

- Find Affected Keys
  - When a server is added or removed, a fraction of data needs to be redistributed.



- Server 4 is added onto the ring. The affected range starts from s4 and moves anticlockwise around the ring until a server is found. Thus, keys located between s3 and s4 need to be redistributed to s4.



- When a server is removed, the affected range starts from s1 and moves anticlockwise around the ring until a server is found. Thus keys located between s_0 and s_1 needs to redistributed to s_2.

- Wrap Up
  - Benefits of consistent hashing
    - Minimized keys are redistributed when servers are added or removed.
    - It is easy to scale horizontally because data are more evenly distributed.
    - Mitigate hotspot key problem. Consistent hashing helps to mitigate the problem by distributing the data more evenly.