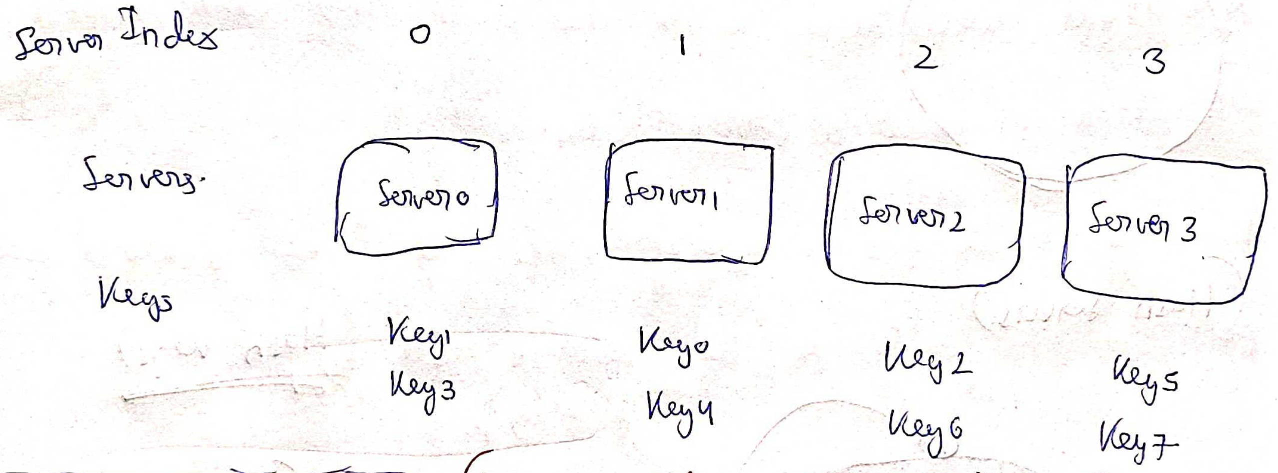
Chapter 5: Design Consistent Hashing

To achieve horizontal scaling it is important to distribute requests/data efficiently and evenly across all servers.

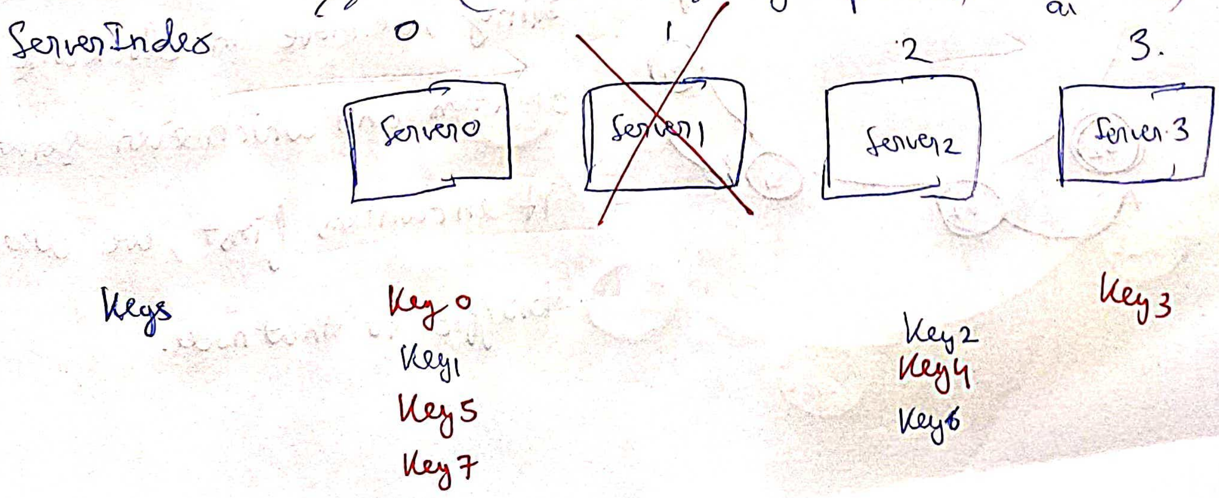
Most Basic approach that comes to mind is that use the hash function. We have N servers and we use the

serverIndex = hash(Key) % N.



$\boxed{\text{Lets say Server1 Crashes}}$ → (Huge amount of key distribution)

(Keys are getting remapped and lot of Keys are getting replaced, We want to a)



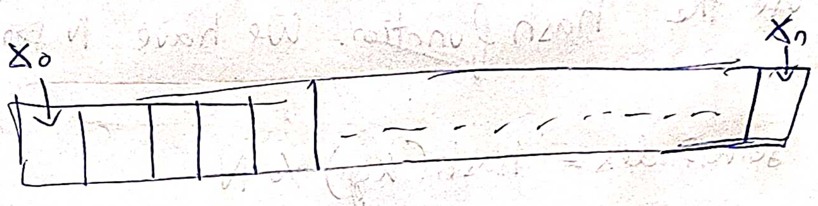
Consistent Hashing

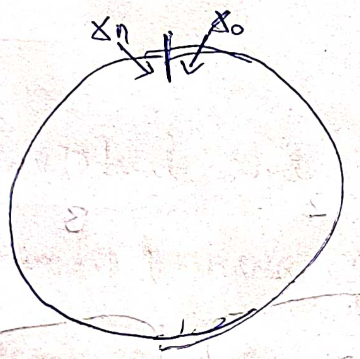
Consistent Hashing is a special kind of Hashing where only K/n keys need to be remapped on average

$K \rightarrow$ no of keys

$n \rightarrow$ no of slots

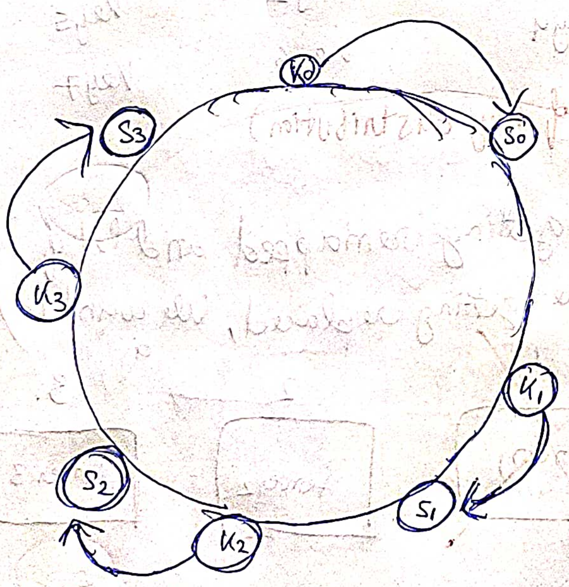
(Hash ~~Key~~ Ring)





(Hash Servers)

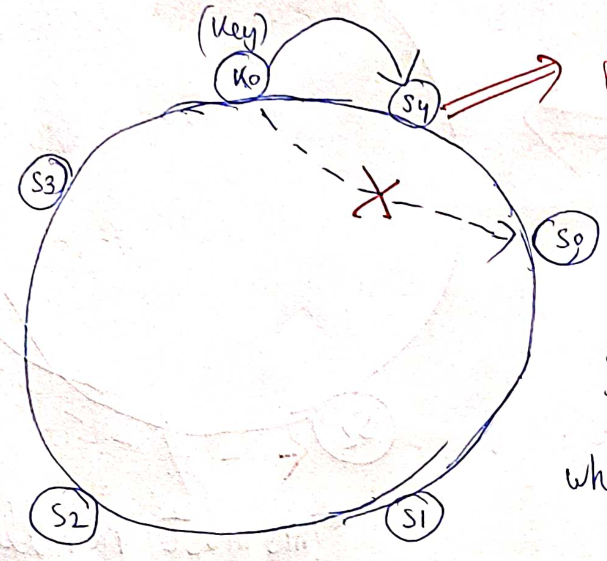
(Hash Keys)



Keys (K_0, K_1, K_2, K_3)

the key which ends up on the ring, we move in clockwise direction and which ever server node it encounters first, we direct traffic to that node

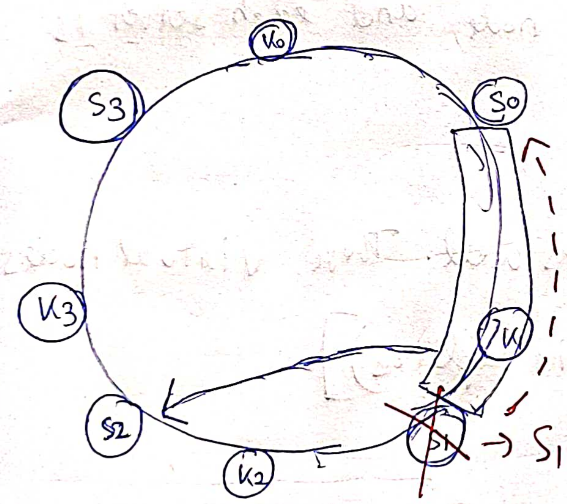
Adding a Server and Removing a Server.



→ this server got newly added.

S4 got added,

So we travel from where S4 got added to S3 and we travel anticlockwise, and whatever keys are there will be mapped to S4.



→ The server which fails move anticlockwise till we find the previous server.

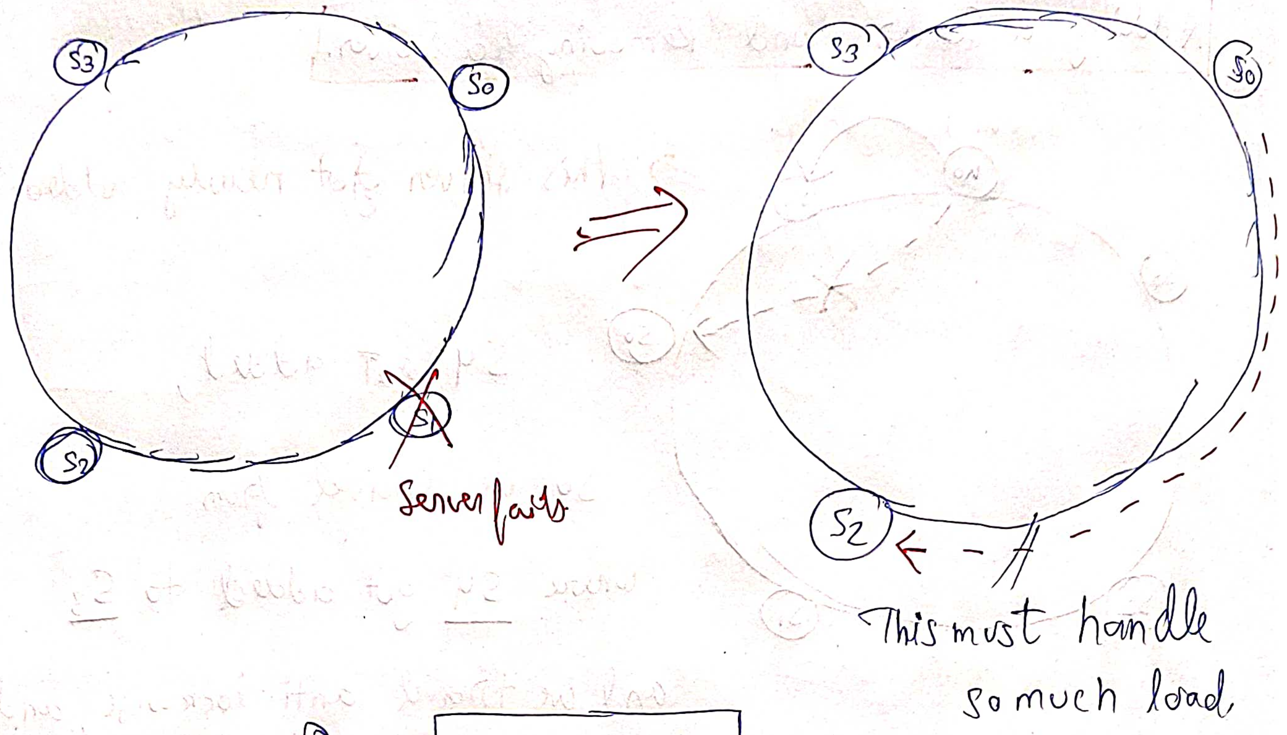
→ S1 fails.

Get all the Keys and map it ~~to~~ S2 → server

But this approach, might lead to uneven distribution.

We cannot make a gurantee on the partition size as we have addition and removal of server.

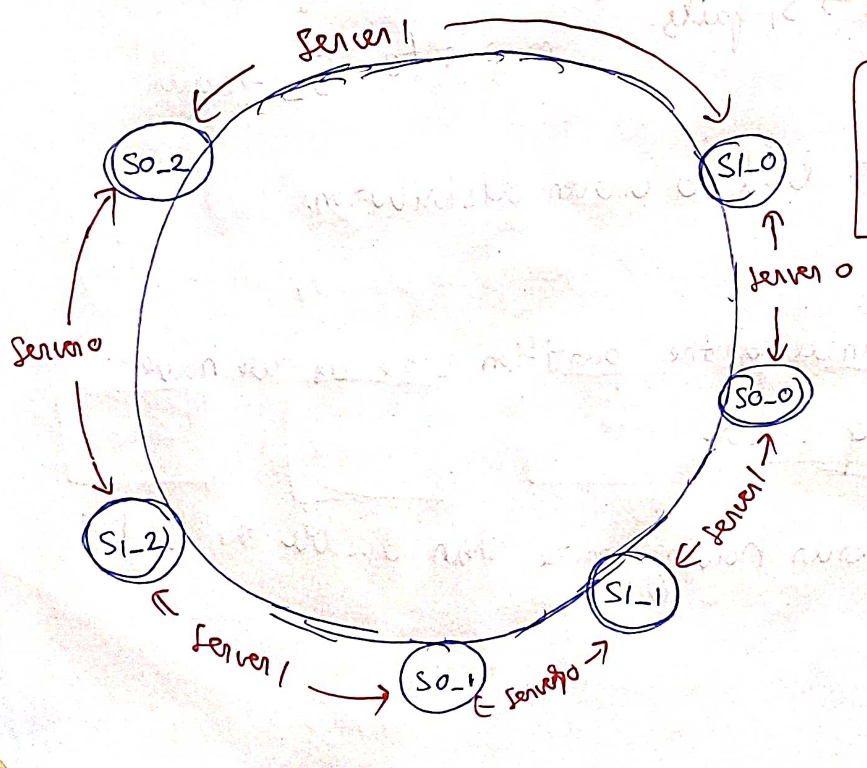
This can lead to a server having more than double the load of out of the total



(How to resolve this) → $\boxed{\text{Virtual Node}}$

A virtual node refers to the real node, and each server is represented by multiple virtual nodes.

Each real node has its replications as well. These virtual nodes point to the [real node (along with its replicas)].



[S0 → server 0
S1 → server 1]