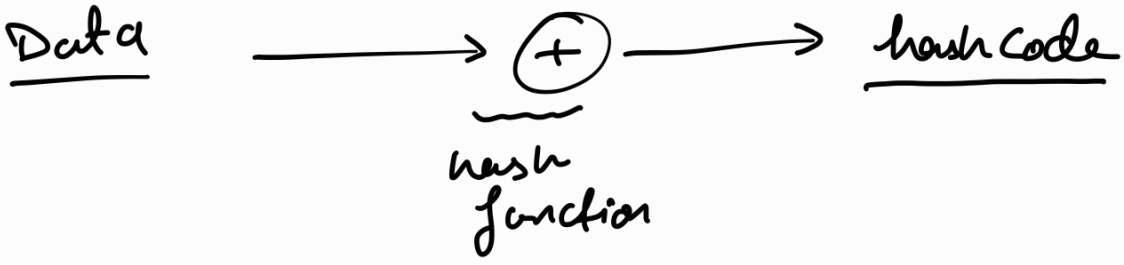
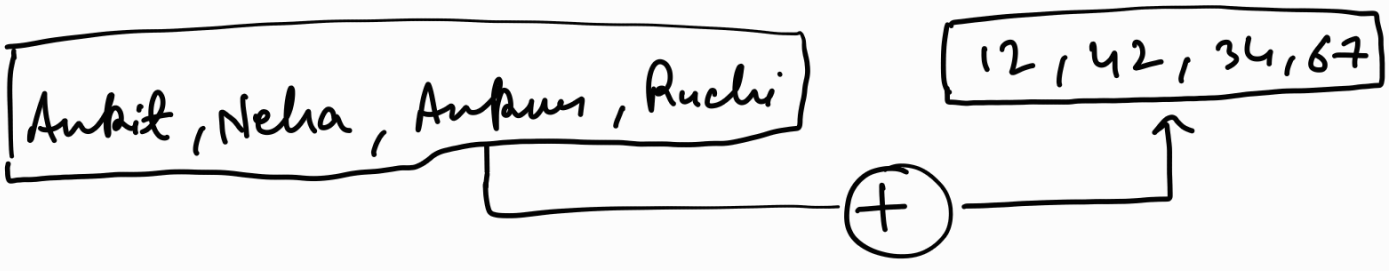
Hashing | Consistent Hashing

Hashing :



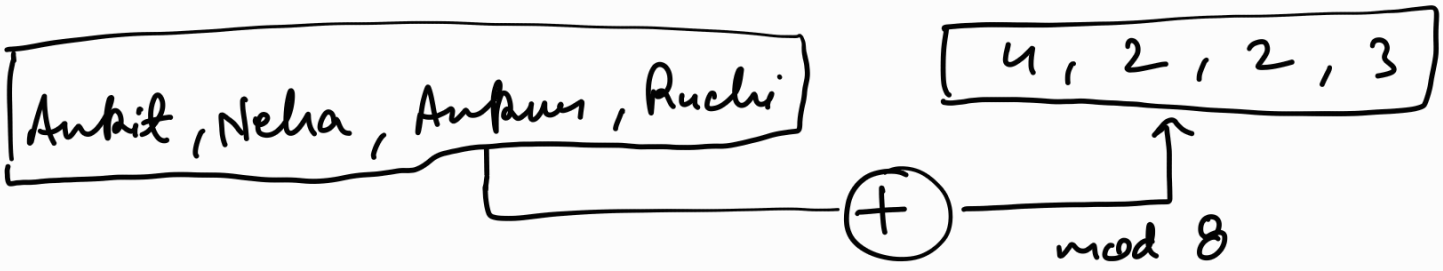
Ex :



array indices

12	42	34	67
Ankit	Neha	Anjum	Ruchi

int range = $0 - (2^{32}-1)$



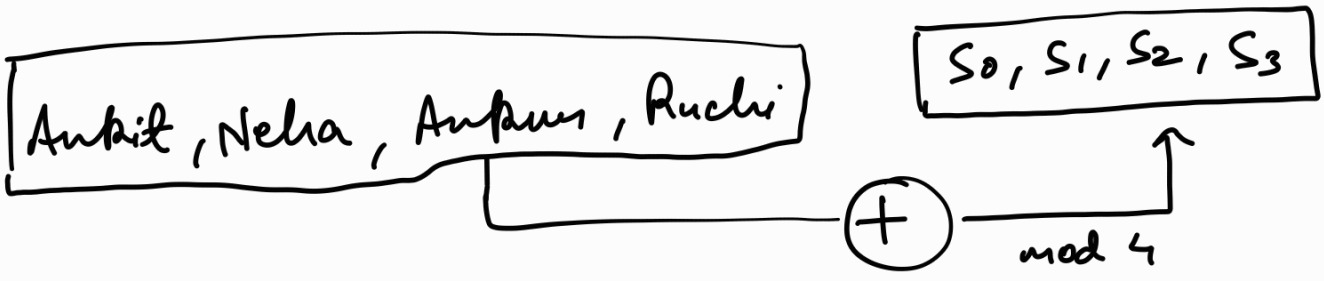
0	1	2	3	4	5	6	7	8
		Neha	Ruchi	Ankit				
		Anpum						

→ Collision (study about it seperately)

use of hashing is immense in key-value stores such as Memcached & Redis

If we have an immensely large key-value store then we will have to store the values across multiple servers in distributed manner.

Ex: let's say we have 4 servers (S0, S1, S2, S3)



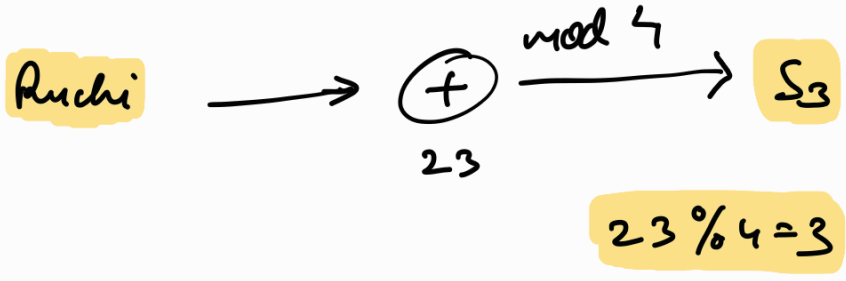
(16, 25, 30, 23)

16 % 4 = 0	25 % 4 == 1	30 % 4 = 2	23 % 4 = 3
S0	S1	S2	S3
Ankit	Neha	Anpum	Ruchi

Q. How to access keys from these servers ?

Ex:

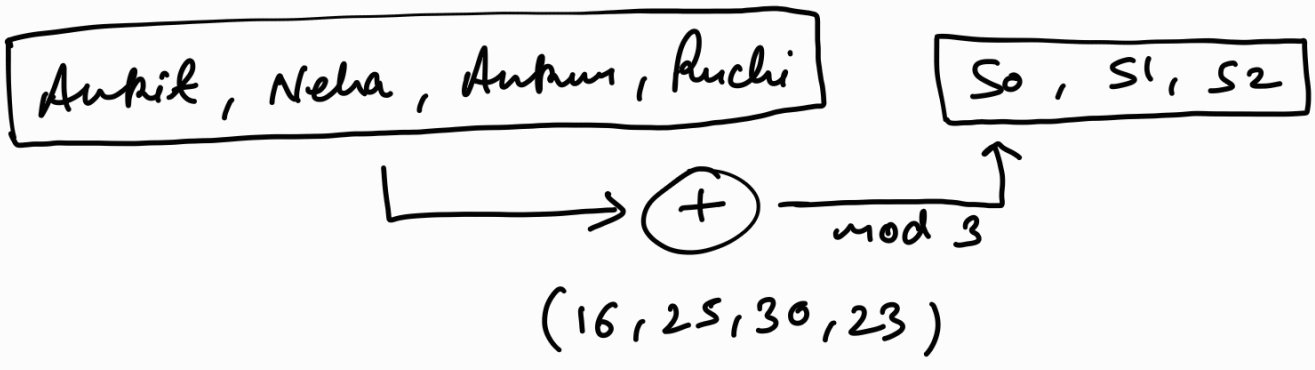
Key : Ruchi .



Q1 what happens when the load increases or decreases and we might have to increase or decrease the number of servers.

let's Consider we have a decreased load, and Consequently we remove the server S3

Now the situation is :



we will have to redistribute load that was handled by S3 to the remaining (S0, S1, S2)

S0 S1 S2

Anmol

16 % 3 = 1

Ankit

Neha

25 % 3 = 1

30 % 3 = 0

Ruchi

23 % 3 = 2

We can see that not only the keys mapped to S_3 got reassigned but other keys also got reassigned

Reassigned keys :

1. Ruchi from S_3 to S_2
2. Ankit from S_0 to S_1
3. Anmol from S_2 to S_0

4 out of 3 keys were reassigned and remapped

75% keys

Question :

let's say

S_i (initial servers) = x

S_f (final servers) = y

here $x < y$ or $x > y$

n = no. of keys that got reassigned.

find the relation b/w x, y and N

solution :

let's Consider :

$x < y$ → servers got increased.

Total keys = N :

There are 2 challenges when we try to distribute data:

1: how to know that on which node a particular piece of data will be stored?

2: when we add or remove nodes/servers, how to know that the data will be moved from existing nodes to the new nodes?
Additionally, how can we minimize data movement when nodes join or leave?

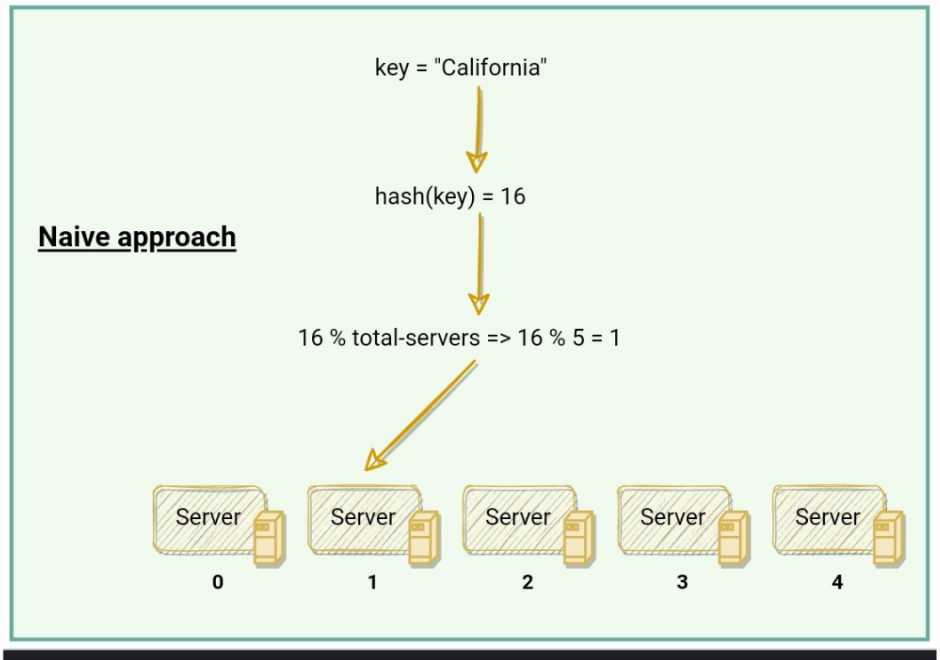
Naive Approach:

a) Use a suitable hash function to map data key to a number

b) Find the server by applying modulo on this number.

From the above discussed Example:

we can see that standard Hashing is not a very flexible solution



Data partitioning using simple hashing

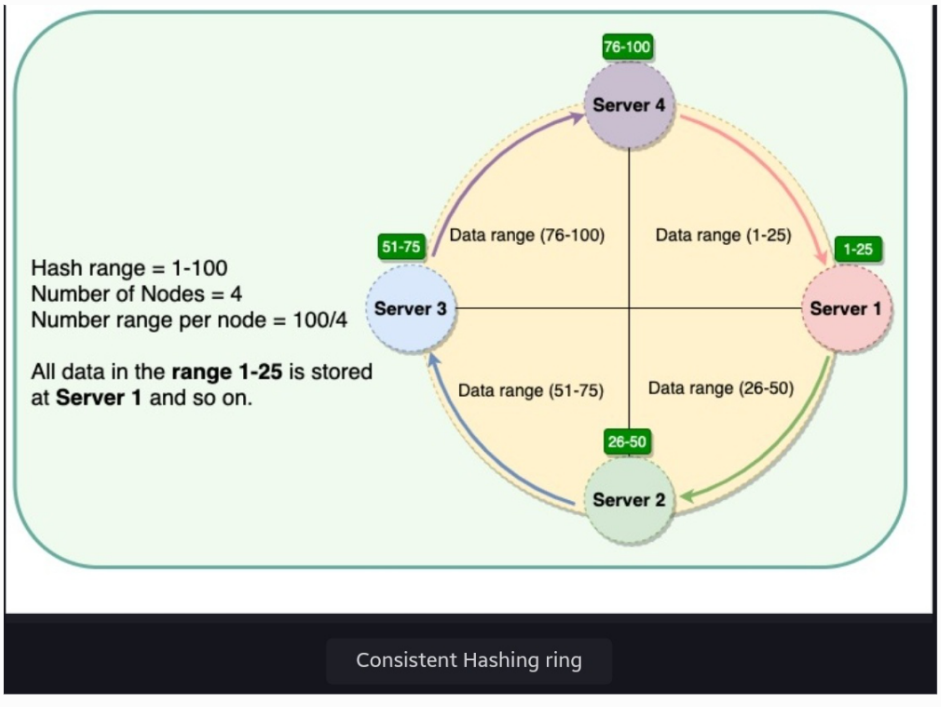
when we add or remove a server, all our existing mappings will be broken

we have to **remap all the keys** and move our data based on the new server count, which will be a **complete mess!**

Consistent Hashing to the rescue#

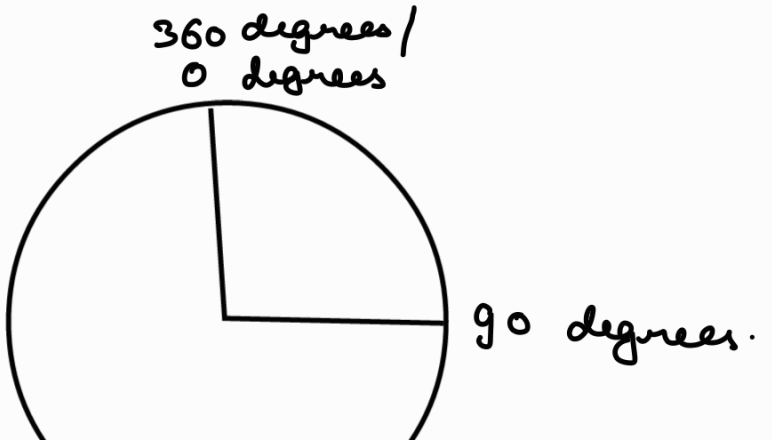
Consistent Hashing maps data to physical nodes and ensures that **only a small set of keys move when servers are added or removed.**

Consistent Hashing stores the data managed by a distributed system in a ring. Each node in the ring is assigned a range of data. Here is an example of the consistent hash ring:



Consistent Hashing ring

We want to reduce the number of keys that get reassigned.



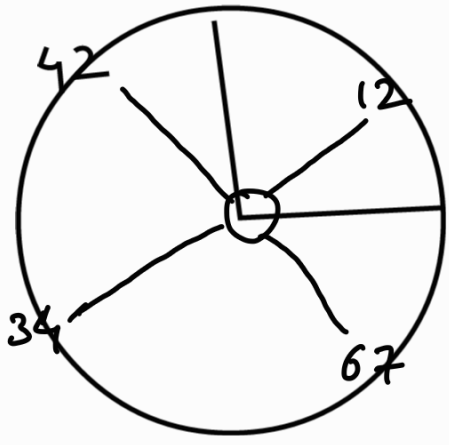
let's say:

hash output range : [0, 100]

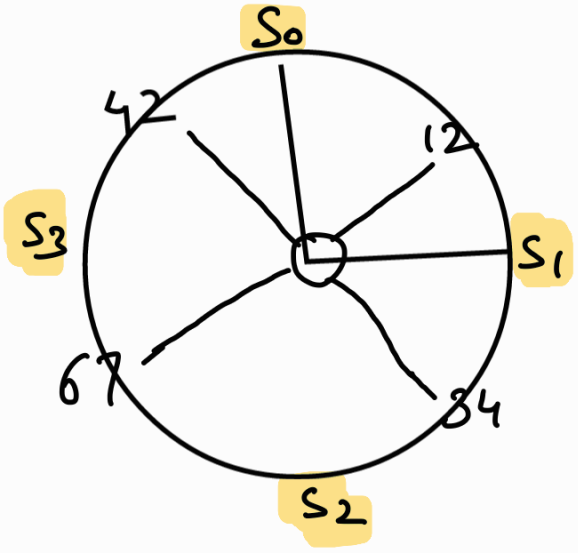
Angles on Circle : [0 - 360]

[Ankit, Neha, Anjum Ruchi]

+ [12, 42, 34, 67]



S0	S1	S2	S3
Neha	Ankit	Ruchi	Anjum
42	12	67	34



S0 - 0
S1 - 25
S2 - 50
S3 - 75

server hash and data-value hash, on the same circle.

(1) the basic idea behind Consistent hashing is that we represent the hash values for both data (Ankit, Ruchi etc..)

and the hash value of the servers (S_0, S_1 ...) on the same circle.

This means that for every key-value pair, there will be a hash generated that will lie on the circle

and for every server too, there will be a hash generated that will lie on the circle within the hash value range (0 – 100)
→ for our example

→ hash for servers could be generated using their IP, name etc.

Note: important =

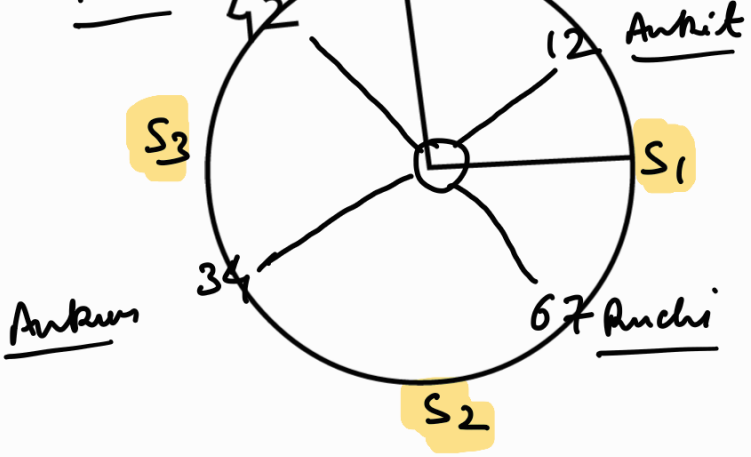
Server's hash value and the data's hash value lie in the same range and both are mapped on the same circle

(2) Map the values to the servers:

We can follow either of the conventions to map the data to the server

(1) clockwise mapping: Every value will sit on the next server it finds on the circle in clockwise direction.

Ex: In this case:



Ankit - 12 will be saved on S1

Ruchi - 67 will be saved on S2

⋮

etc.

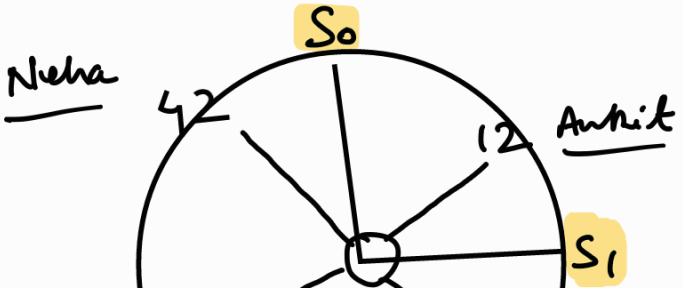
S0	S1	S2	S3
Neha	Ankit	Ruchi	Ankum
42	12	67	34

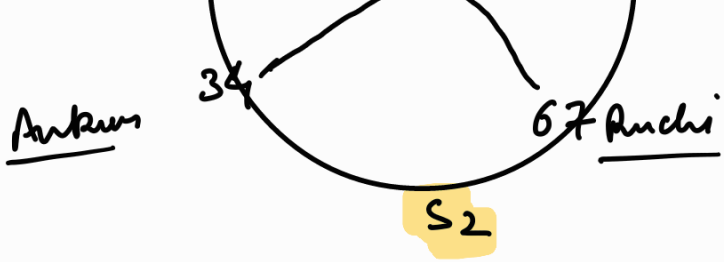
(2) Anti-clockwise : vice versa of clockwise convention.

(3) Adding / Removing servers

Let's Consider removing a server:

say we have removed S3





S0	S1	S2	~~S3~~ S0
Neha	Ankit	Ruchi	Ankur
42	12	67	34

↳ only one of the keys had to be remapped

less no. of keys have to be remapped.