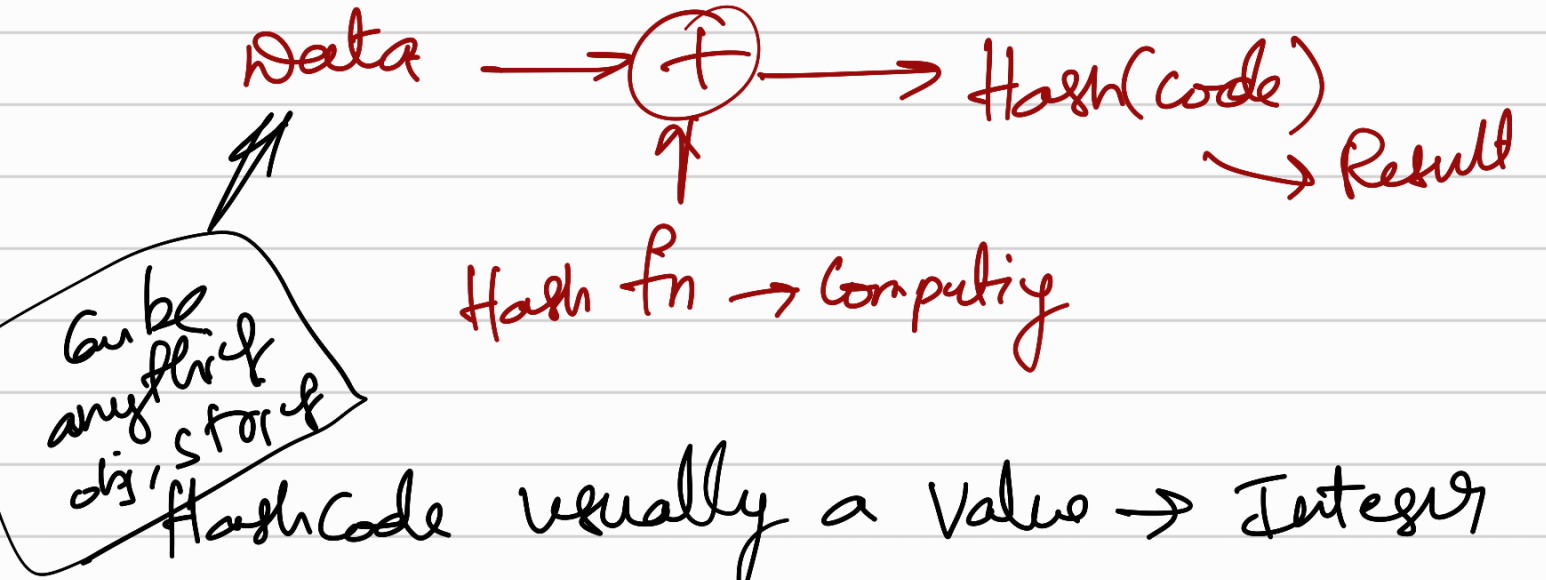
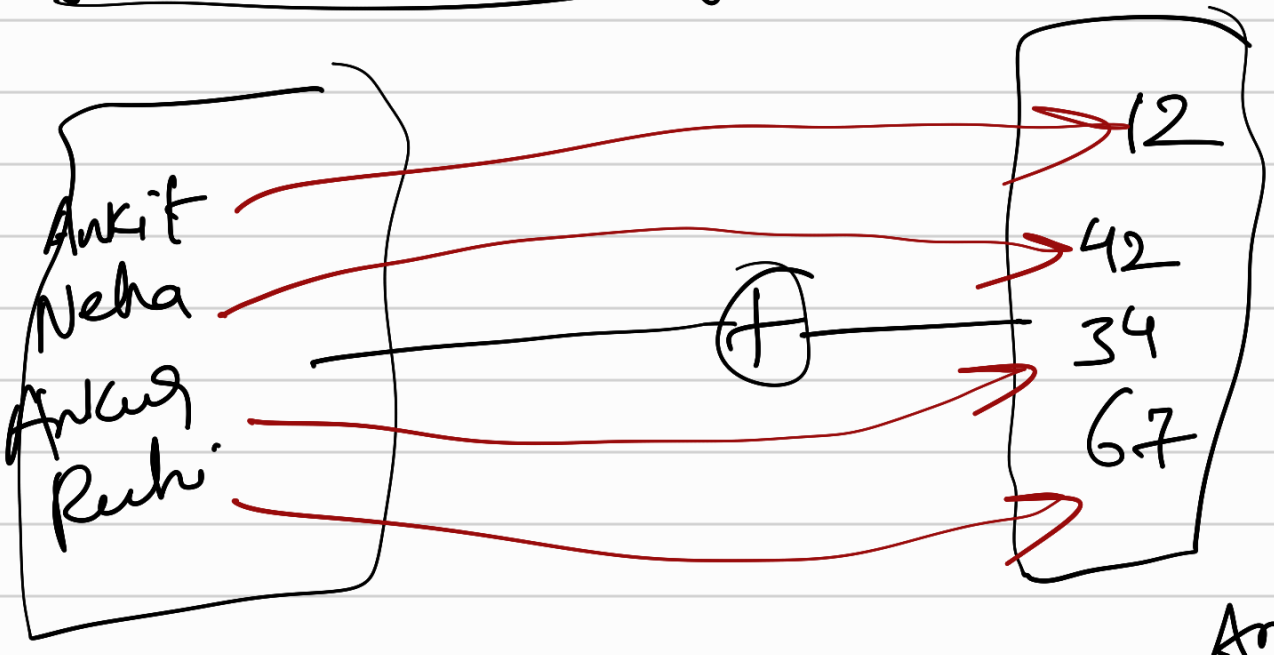
Hashing ?

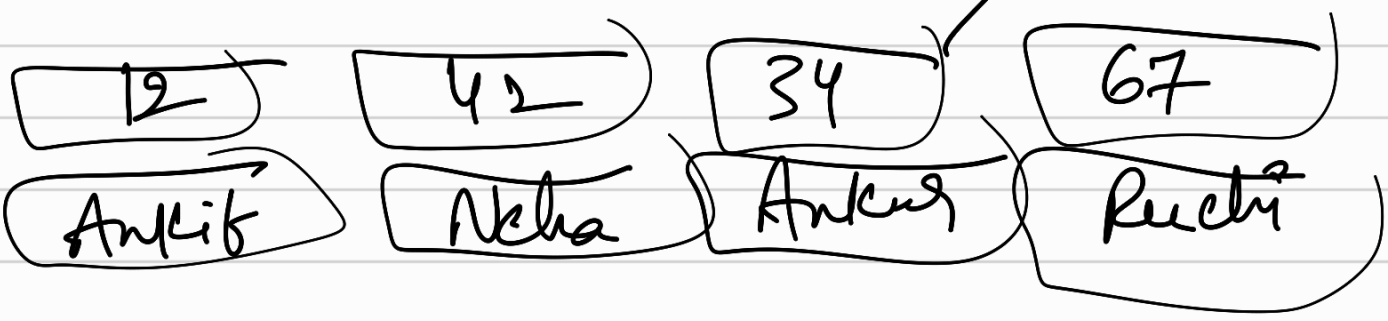


purpose of Hashing:



Searching is easy

Array index



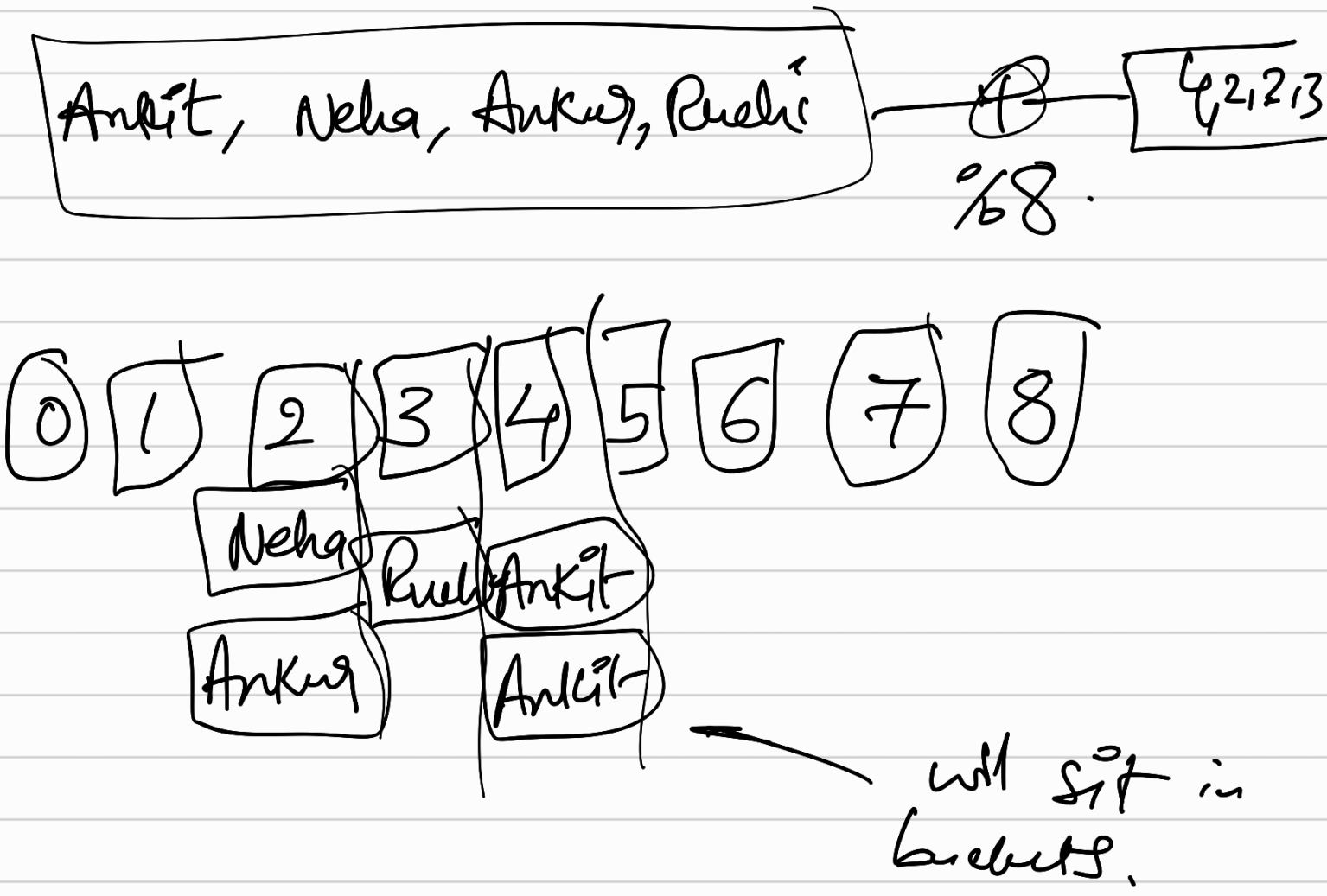
✓ Accessing time becomes constant
bcz Hash fn o/p constant

⊗ if binary search used Complexity becomes $\log n$
worst case n.
more time to search.

✓ Hence if we have array of
Hash o/p's accessing will be
easier.

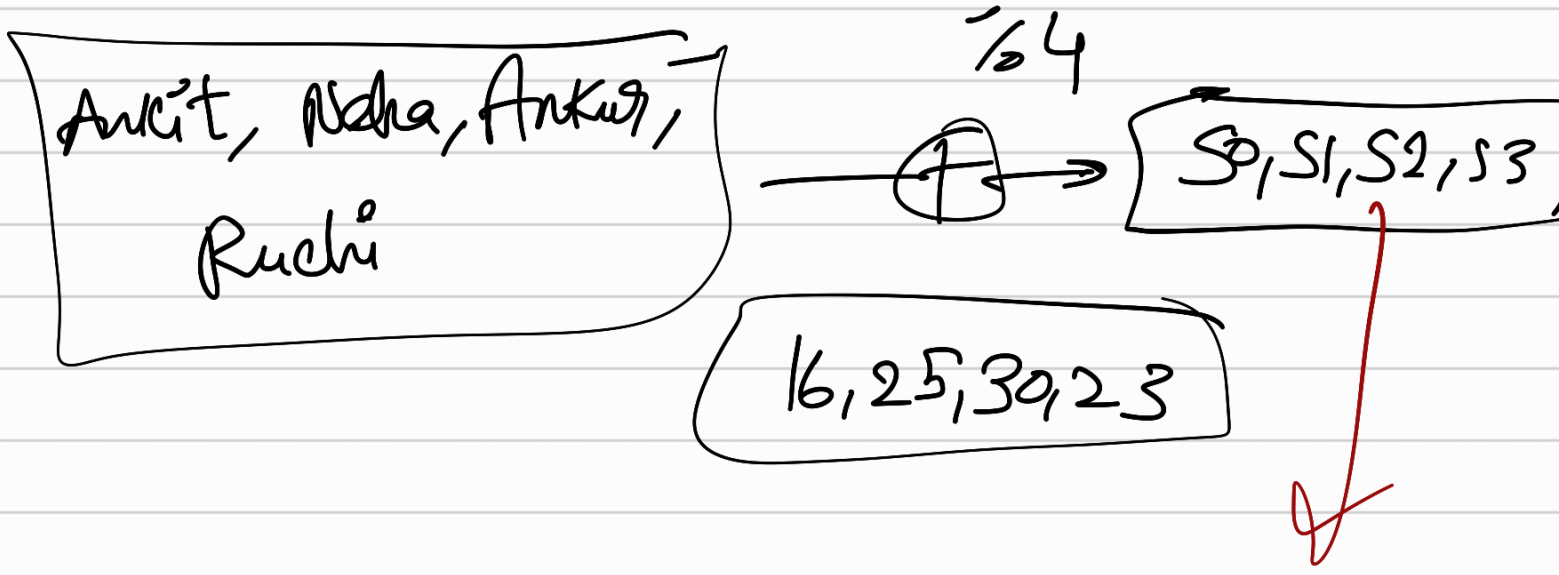
Range of Hash values $0 - 2^{32}$

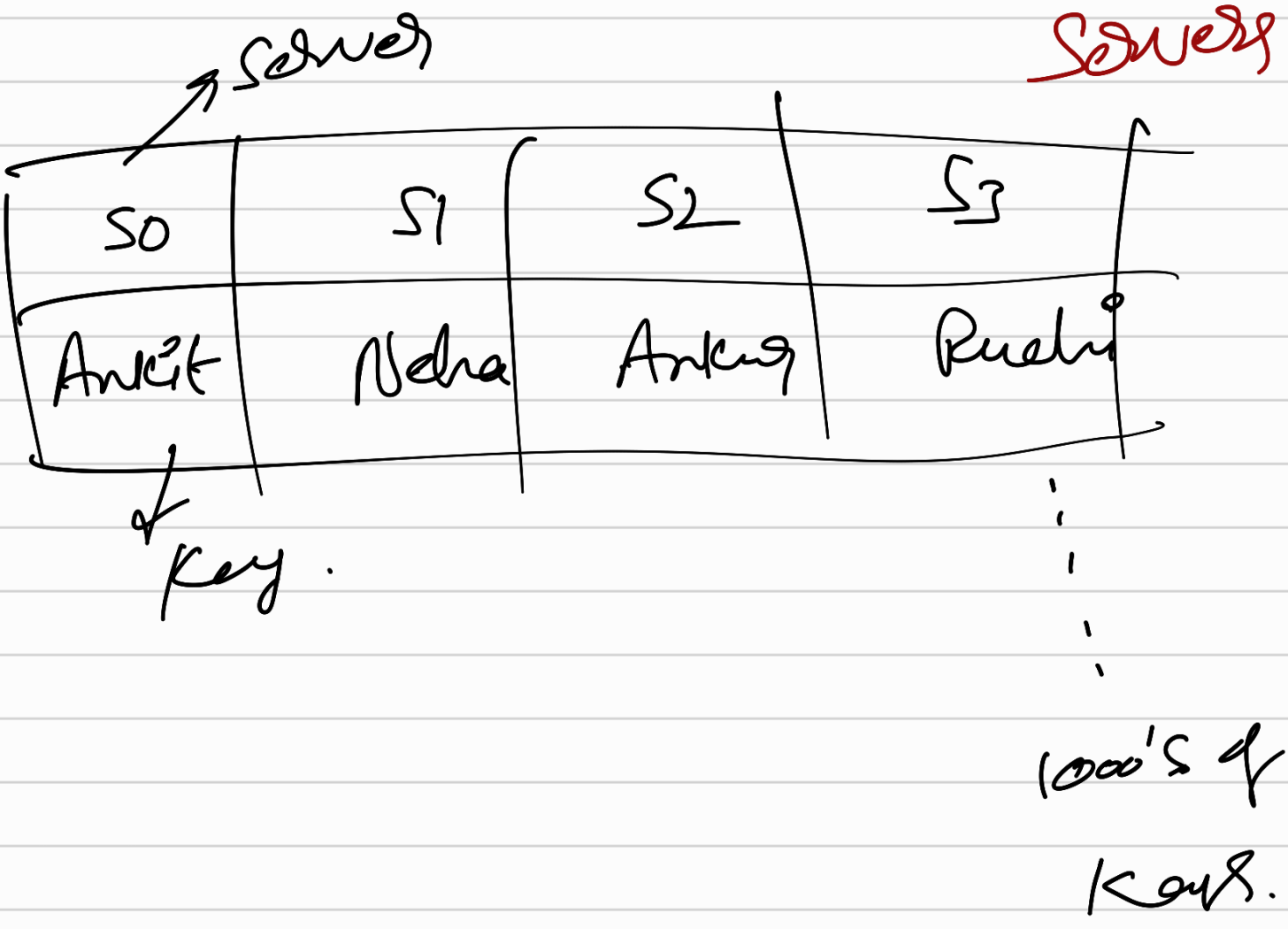
Many values hence
we will take modulus.



So Hashing helps us in storing key value pairs & accessing key value pairs very very efficiently

Hashing Use Cases:

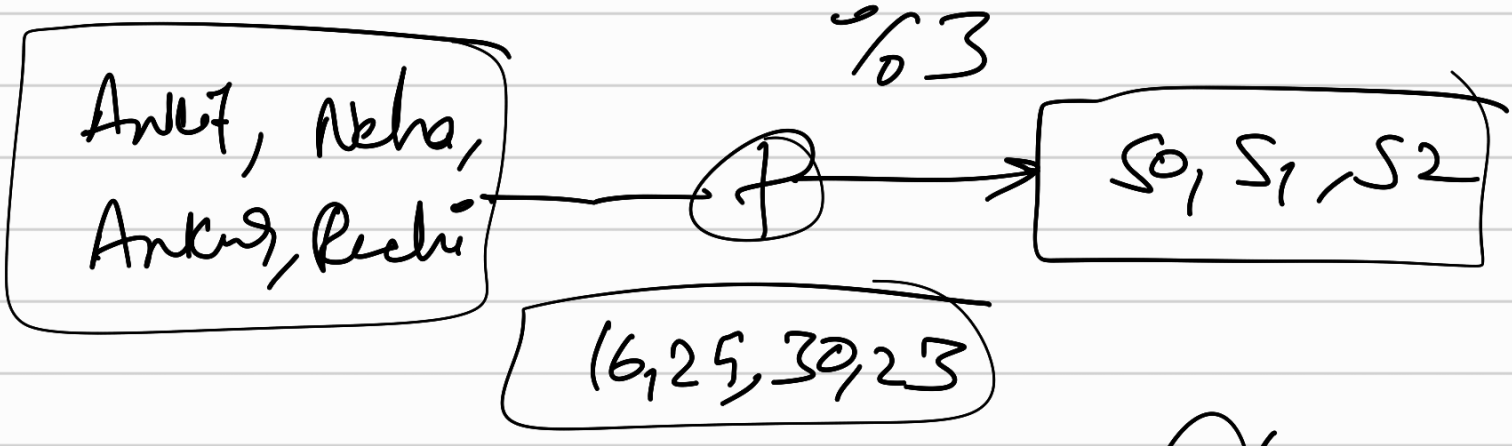




even increased key distribution would be in similar manner.

problem with above approach:

if servers increased / Decreased



Ruchi has to be assigned

but S3 removed.

16 % 3 = 1

25 % 3 = 1

30 % 3 = 0

23 % 3 = 2

Here all keys are re-assigned
Computation becomes more
if servers added / removed.

this is the problem with
normal Hashing

↓

scale in
scale out } → Computation becomes more

to minimize

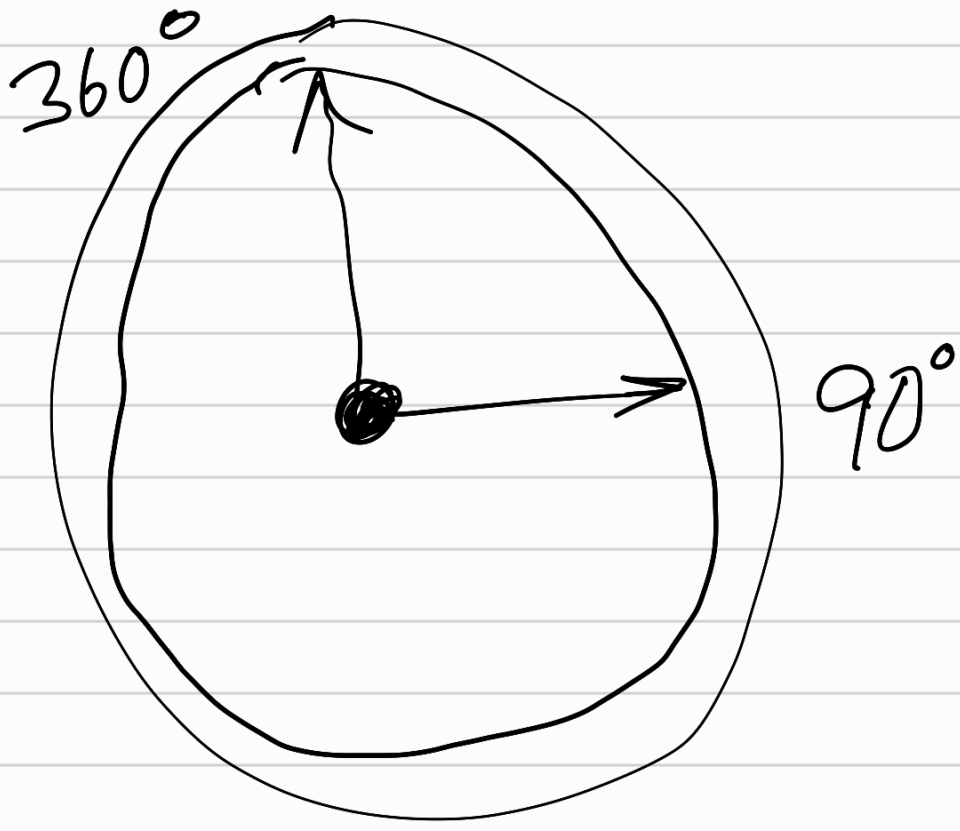
↓

Consistent Hashing.

Remapping of the existing keys on increase(↑) decrease(↓) of servers

?

Consistent Hashing



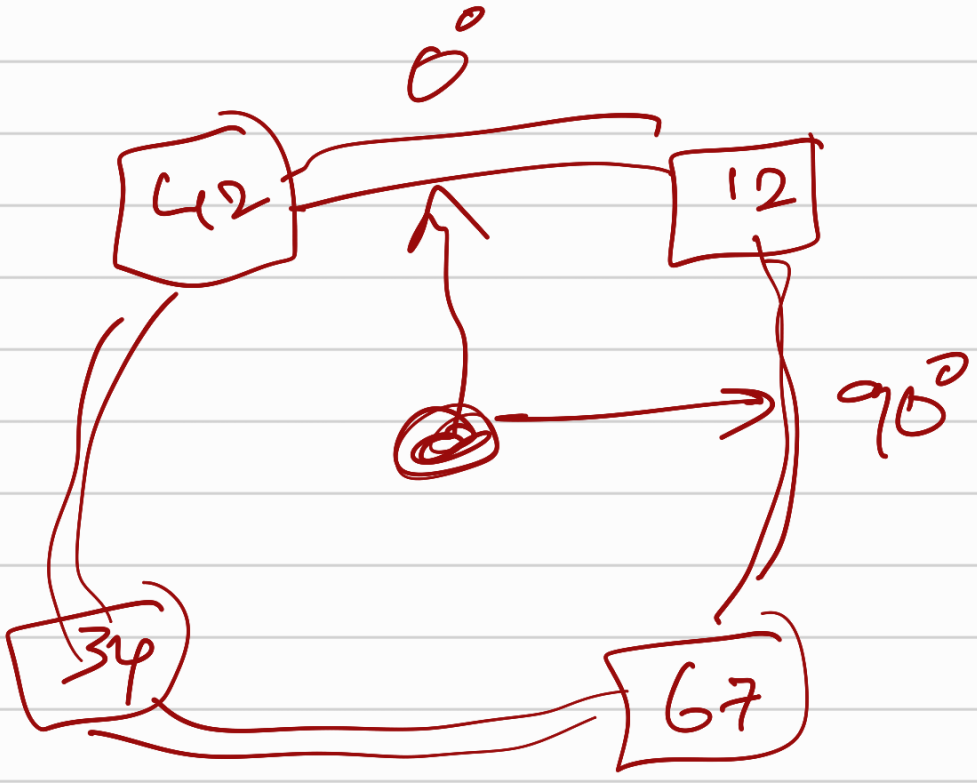
Eg:

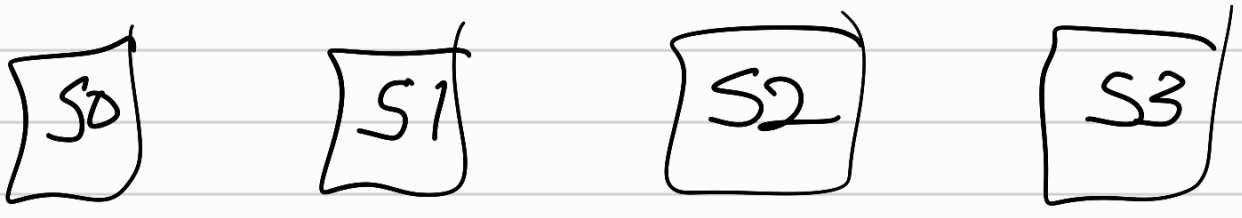
Hash o/p range [0-100]

Angles on Circle [0-360]

Ankit, Neha, Ankur, Ruchi — f — 12, 42, 34, 67

Keys.

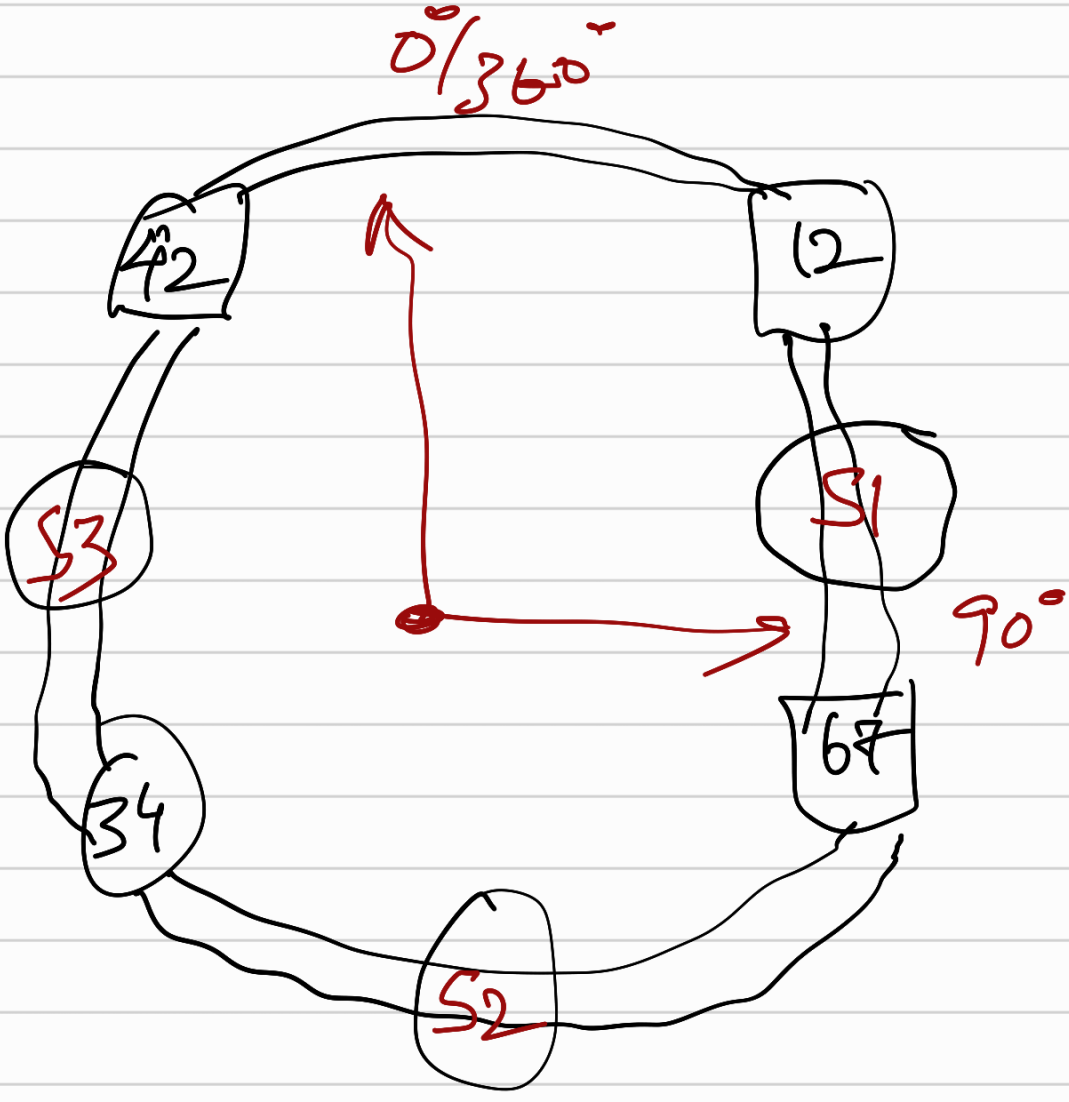




Ankit Neha Ankur Ruchi

similary hash for each Server

Eg:- based on IP add of machine



Hash values of servers will also going the sit on circle.

S1, S2, S3, S4 ← + — 25, 67, 49

Here the servers and ip value

range share the same ip range.

on circle.

Servers Hash
Data Hash
} share same ip range

Mapping of values to servers:

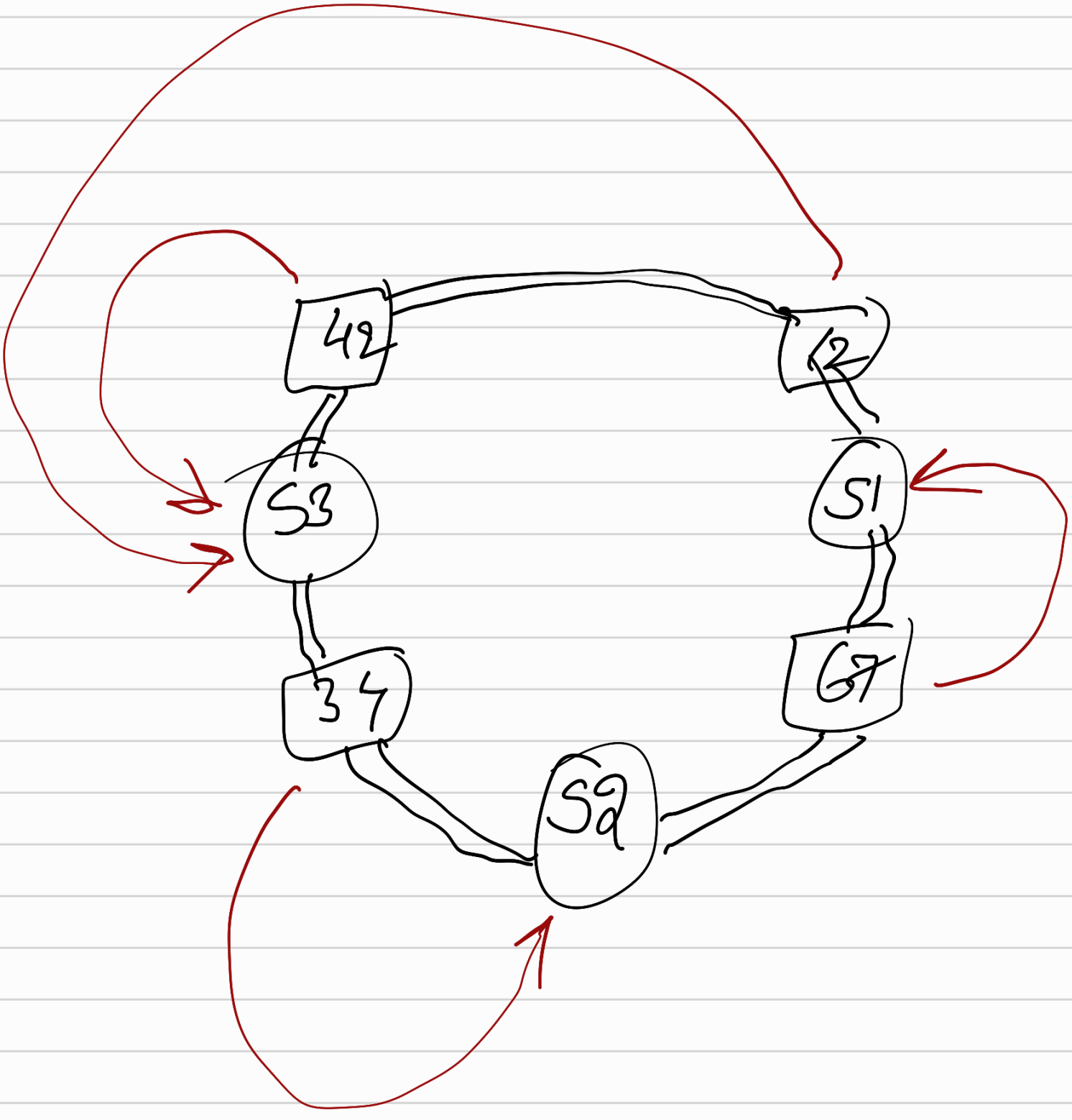
clock

anti clock.

Every value will sit on next server

in clock wise direction

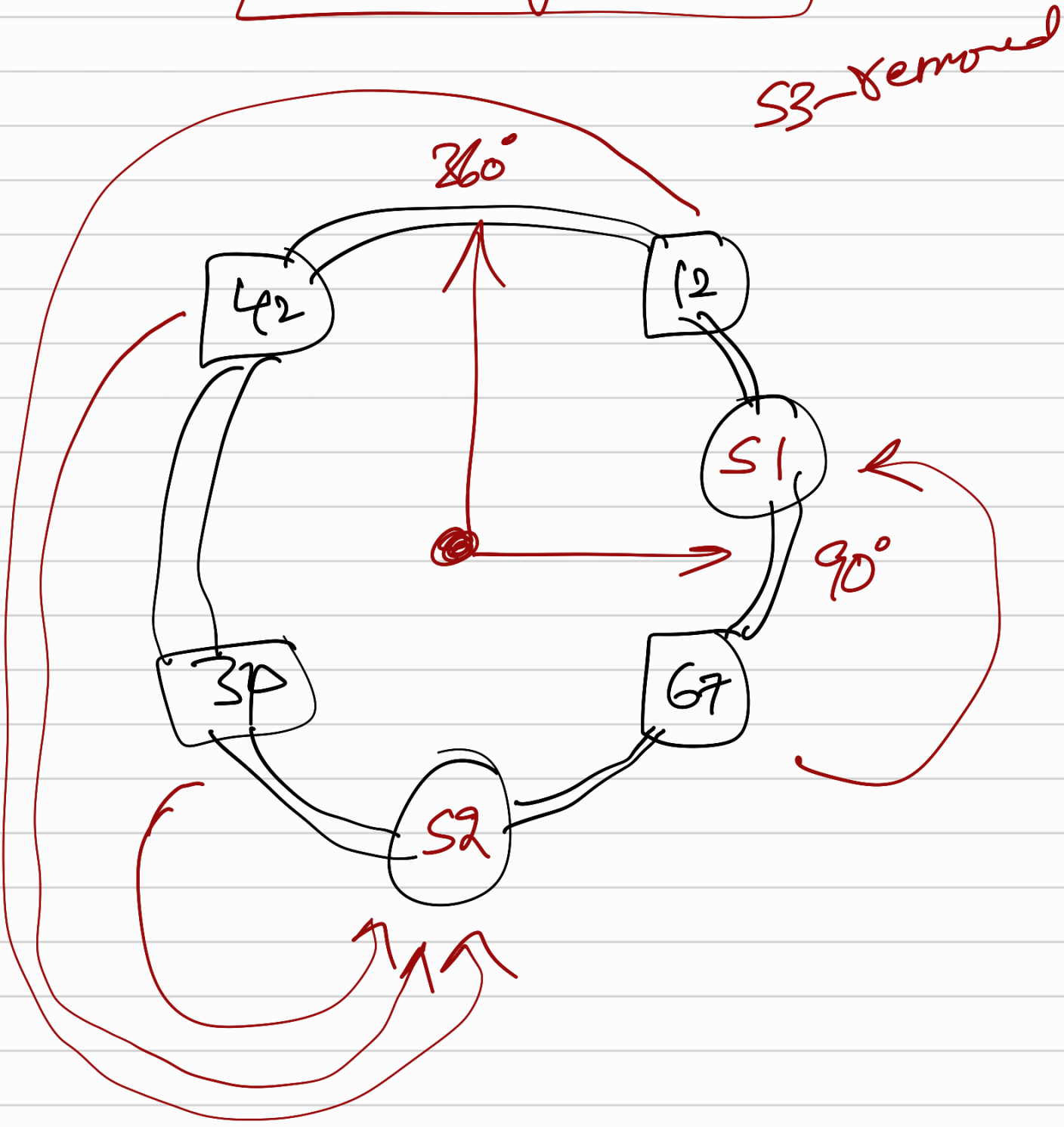
can be anti clock wise direction.



for every value we are trying to find out the closest value in clock/Anticlock direction.

Hash → map key/Nodes to diff servers.

Removing Servers



67 ✓

34 ✓

12 ✗

42 ✗

50% of keys re-mapped but in previous 100% of keys re-mapped

✓ less no of keys re-mapped

✓ less Computation.

Once S3 removed → suddenly 12, 42, 34 mapped to S2

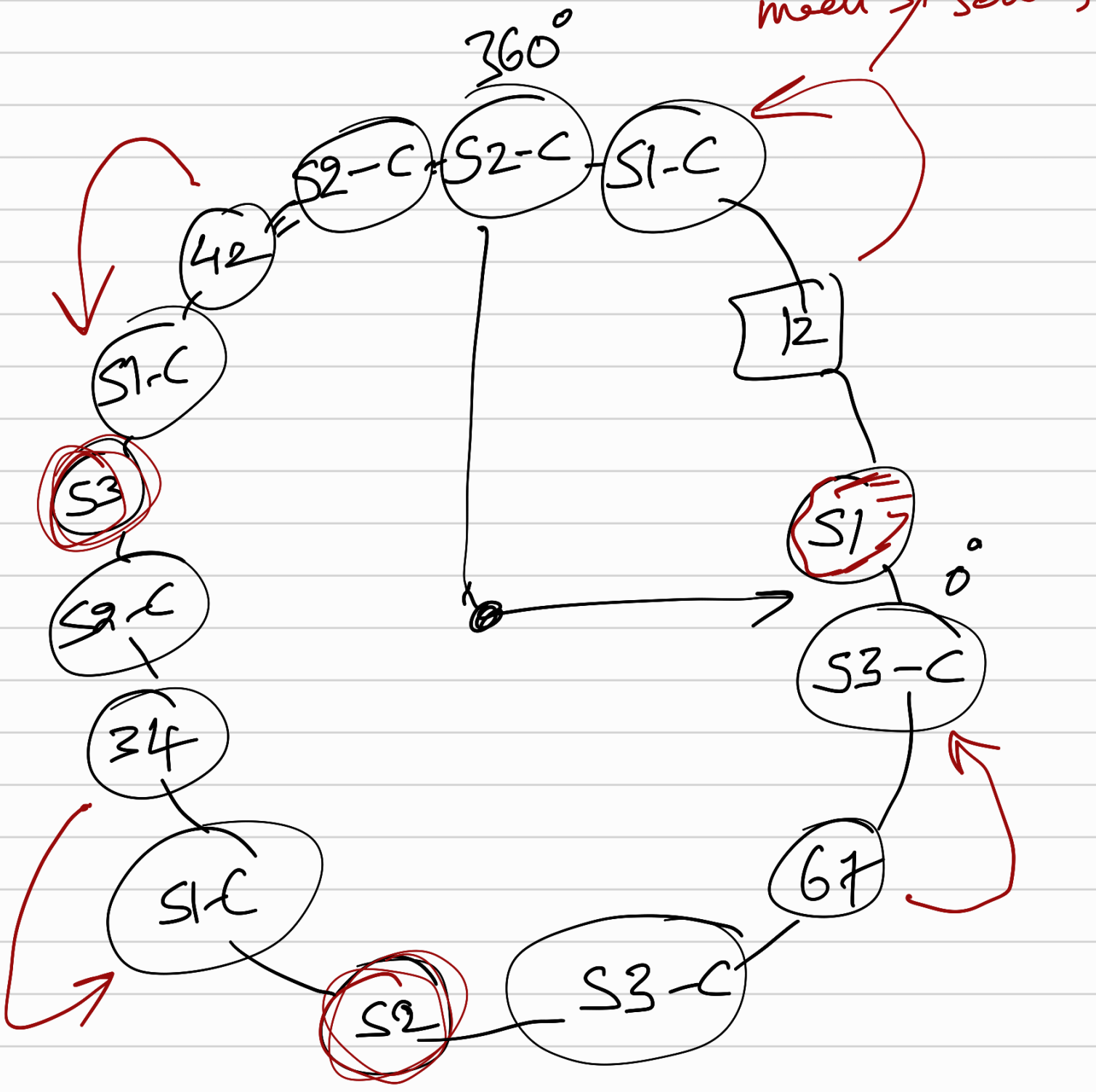
S2 → overloaded with req's.

S1 — has less keys

S2 - has more more keys

To overcome → Replica of nodes/servers

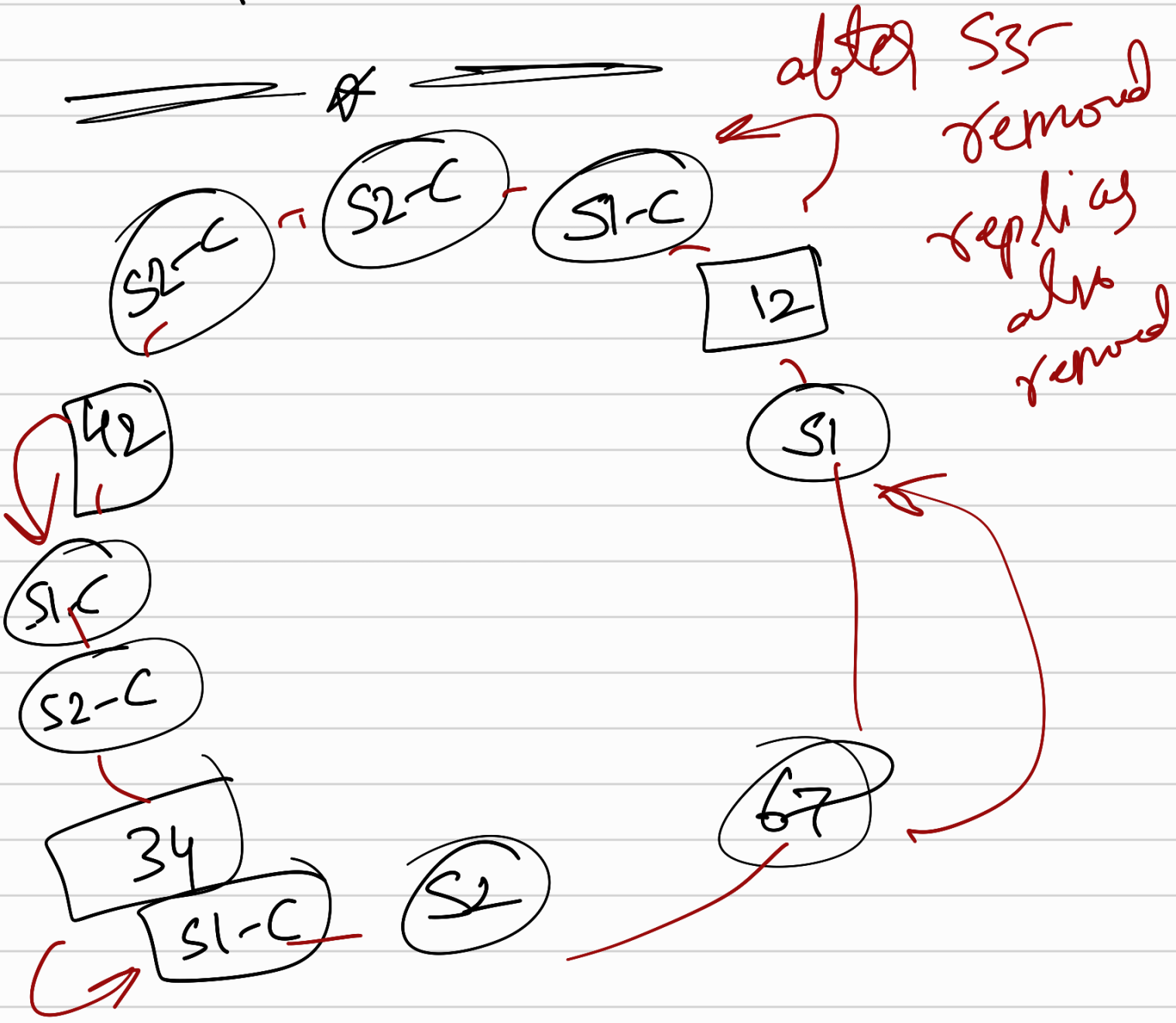
create



S1 replicas - S1-C

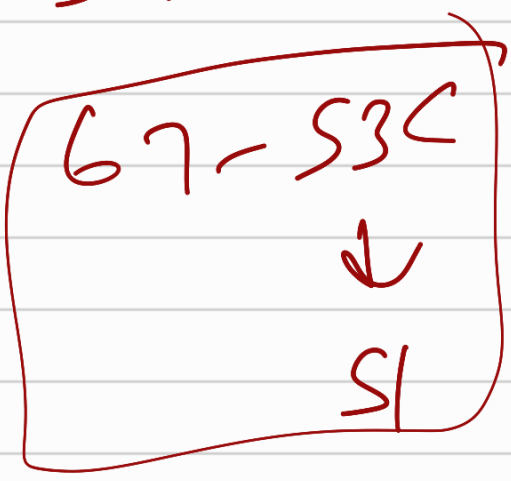
S2 replicas - S2-C

S3 replicas - S3-C



12 - S1C ✓

42 - S1C ✓



only 1-Key have to be re-mapped.

⟹

w/o replicas — 50% Keys remapped

with replicas — 25% Keys remapped.