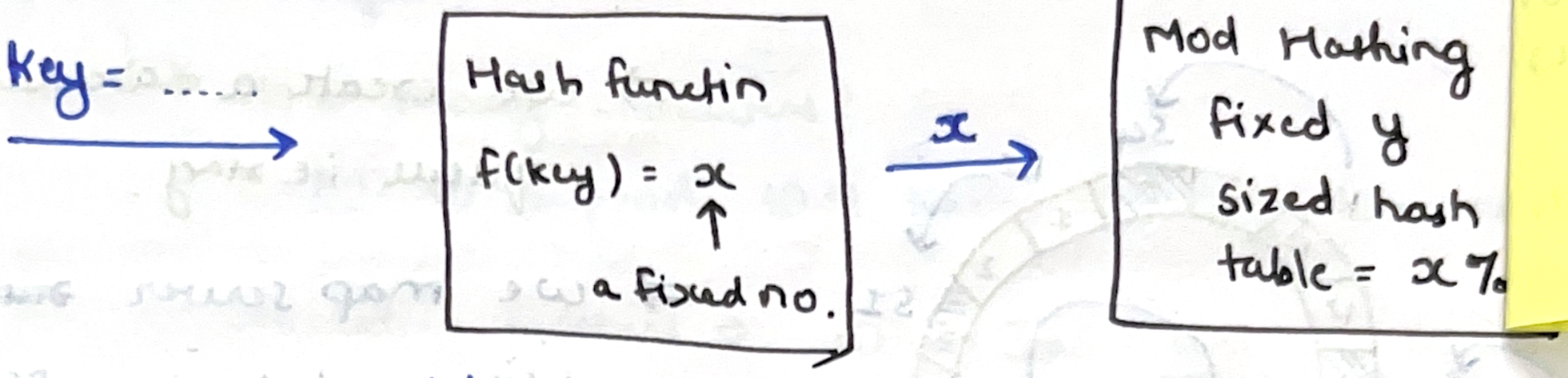
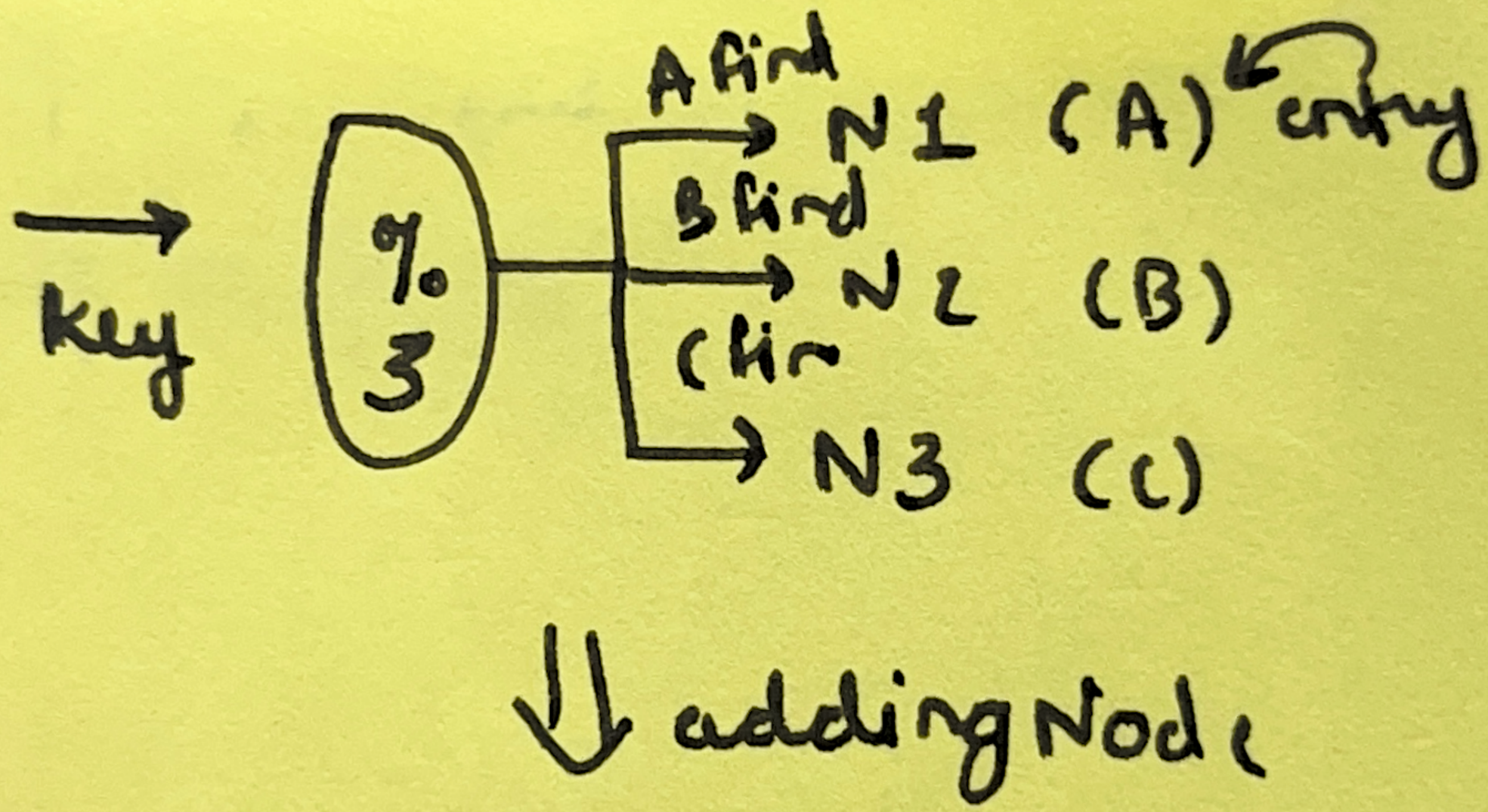
Consistent Hashing



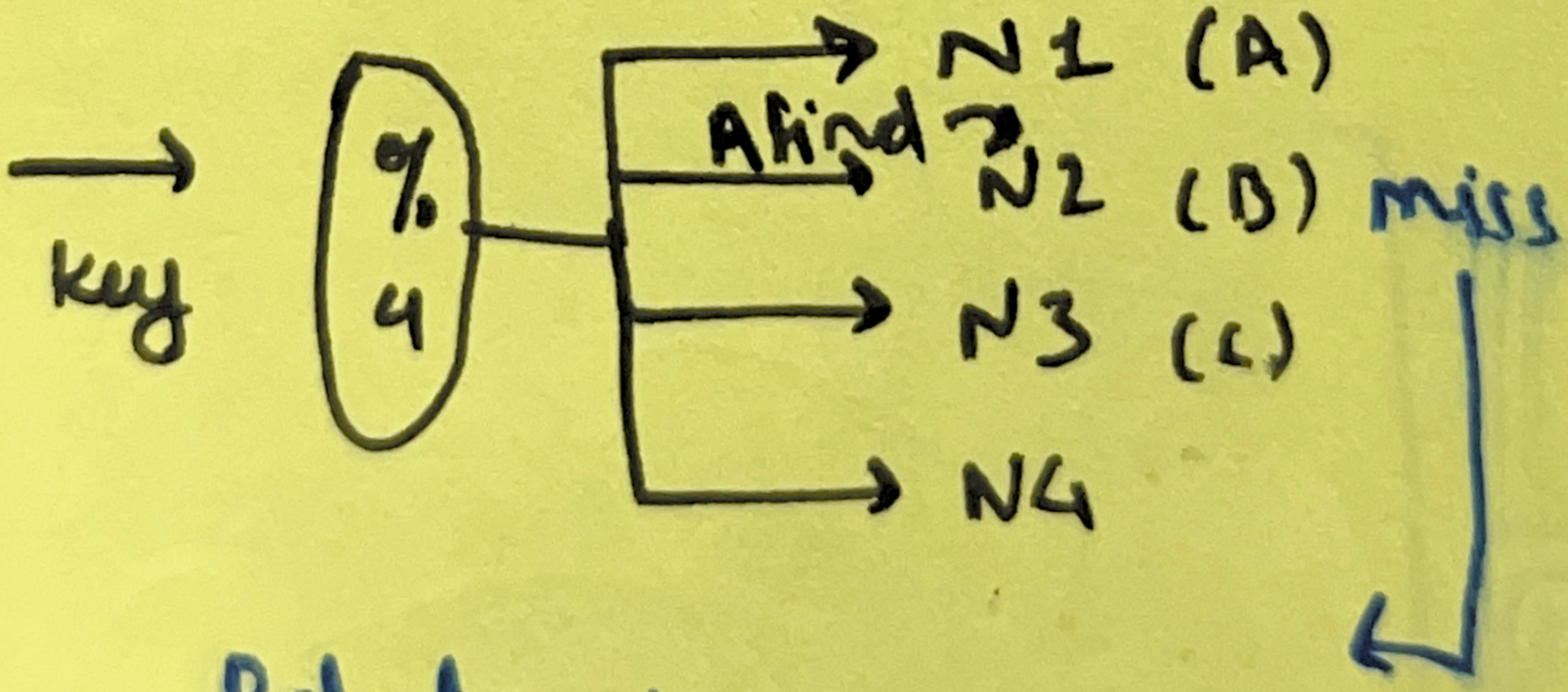
above hashing works perfectly fine when we have fixed size of hash table, But in real world no. of nodes i.e. output of hashing would never be constant.

For Ex: Load Balancing
DB sharding
} nodes are never constant, so if we keep hash table of same length then newly spawned up servers will always be idel.

* when new DB server is added our hasing function would have mod (y+1): this leads to data being searched in wrong node, thus we would need costly rebalancing if new nodes added.



⇓ adding Node



Rebalancing req.

what does consistency Hashing tend to solve?

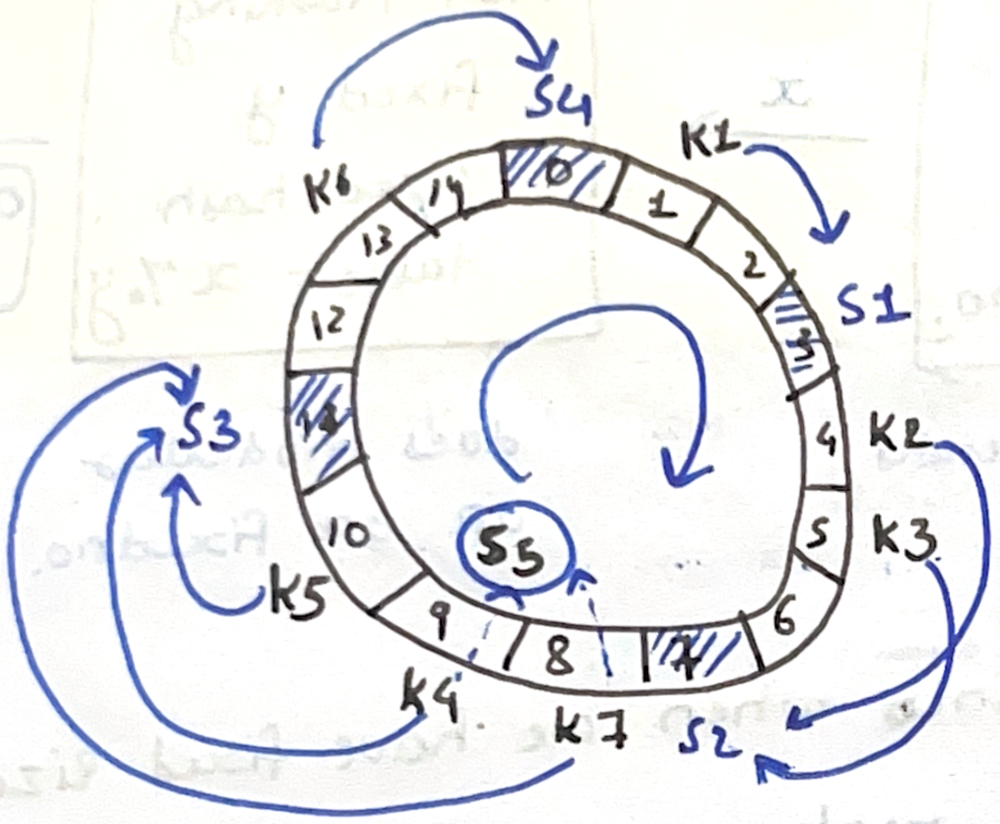
⇒ incase of adding/removal of a node (server)

as ~~too~~ High as $\boxed{\frac{1}{n}\%}$ of rebalancing is required

↓ total nodes

and Not more than that.

i.e. Reducing Rebalancing among servers.



k_i = Requests

S_i = server

⇒ we create a circular queue i.e ring.

⇒ we map server ~~randomly~~ using mod hashing f(S1) % 15

↑ Hash fn ↑ ring size

⇒ incoming request are also mapped to ring using same mod hashing

⇒ to fulfill the request, we choose nearest server in clockwise direction.

⇒ adding new server can lead to rebalancing of very few req. ↑ added through same mod hashing

Tip: Use replicas of same servers over the ring to avoid request getting congested in one server.

↑ Virtual objects: Avoid disadvantage due to clotting of servers nearby due to Hashing.