Hashing

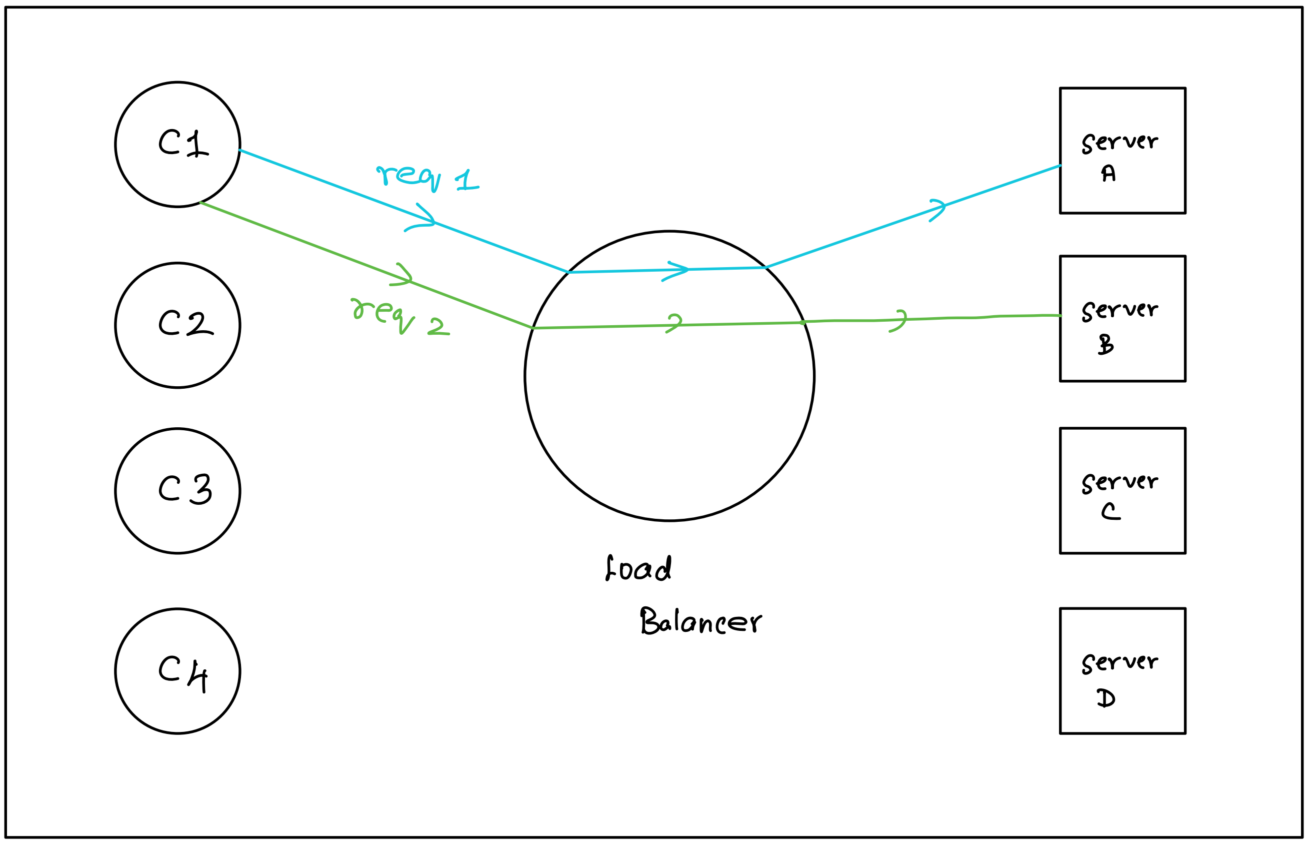
Hashing:

Hashing is the process of converting an arbitrary piece of data into a fixed size value (typically an integer)

In system design we use hashing to hash IP addresses, username, HTTP request etc.

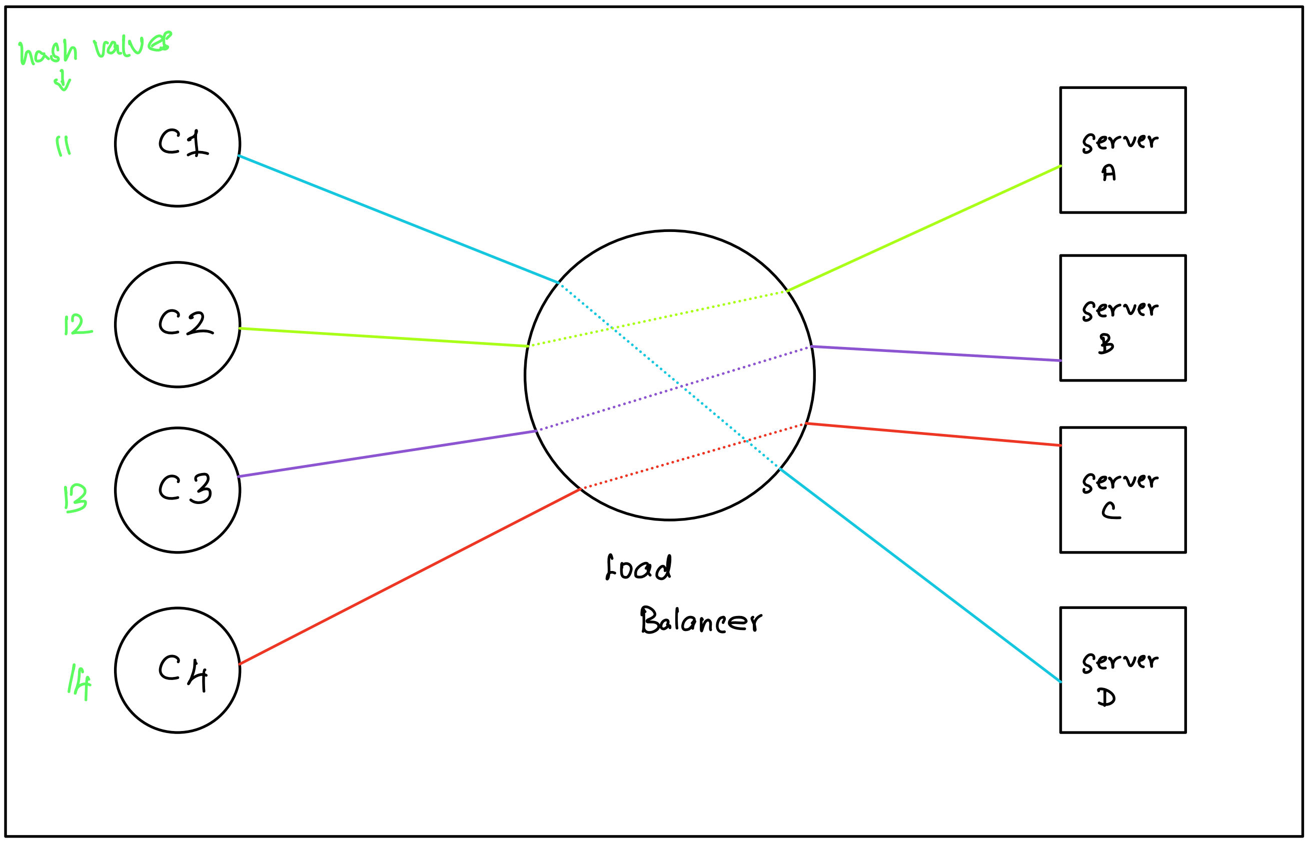
Hashing Use Case

Assume we have four clients C1, C2, C3, C4 and four servers A, B, C, D. The clients are connected to the servers through the load balancer. The four servers are doing computationally long operations specific to the clients. Once the values are calculated for a client it remains constant so they use an cache. Generally in this case, a cache common to all the four servers can be used but due to lack of resources they settle for an in-memory cache (cache in the server)



If the Load Balancer uses Round Robin (say) as the load distribution strategy then the first time the client (C1) sends request it will go to server A but the next time the same client (C1) sends request it will go to server B. But C1's data is already present in server-A (in memory cache).

So in this case it is preferable to hash the clients request and based on the hash value the clients are directed to a particular server



The clients have been hashed to the values 11, 12, 13, 14.

The simplest hasing strategy is the modulo strategy (hash value % no of servers)
of the server

C1 => 11 % 4 => 3 => server-D

C2 => 12 % 4 => 0 => server - A

C3 => 13 % 4 => 1 => server - B

C4 => 14 % 4 => 2 => server - C

Now when C1 sends requests it will always be redirected to server D similarly to other clients

Generally hashing strategies such as the modulo strategy won't be used. For made industry grade hashing functions such as MD5 hashing algorithm, SHA-3 hashing (most popular) algorithm, Bcrypt hash function should be used for proper hashing.

The problem arises when one of our servers die or when we add extra servers then the value which we use in the modulo strategy changes and now the clients will be redirected to different server

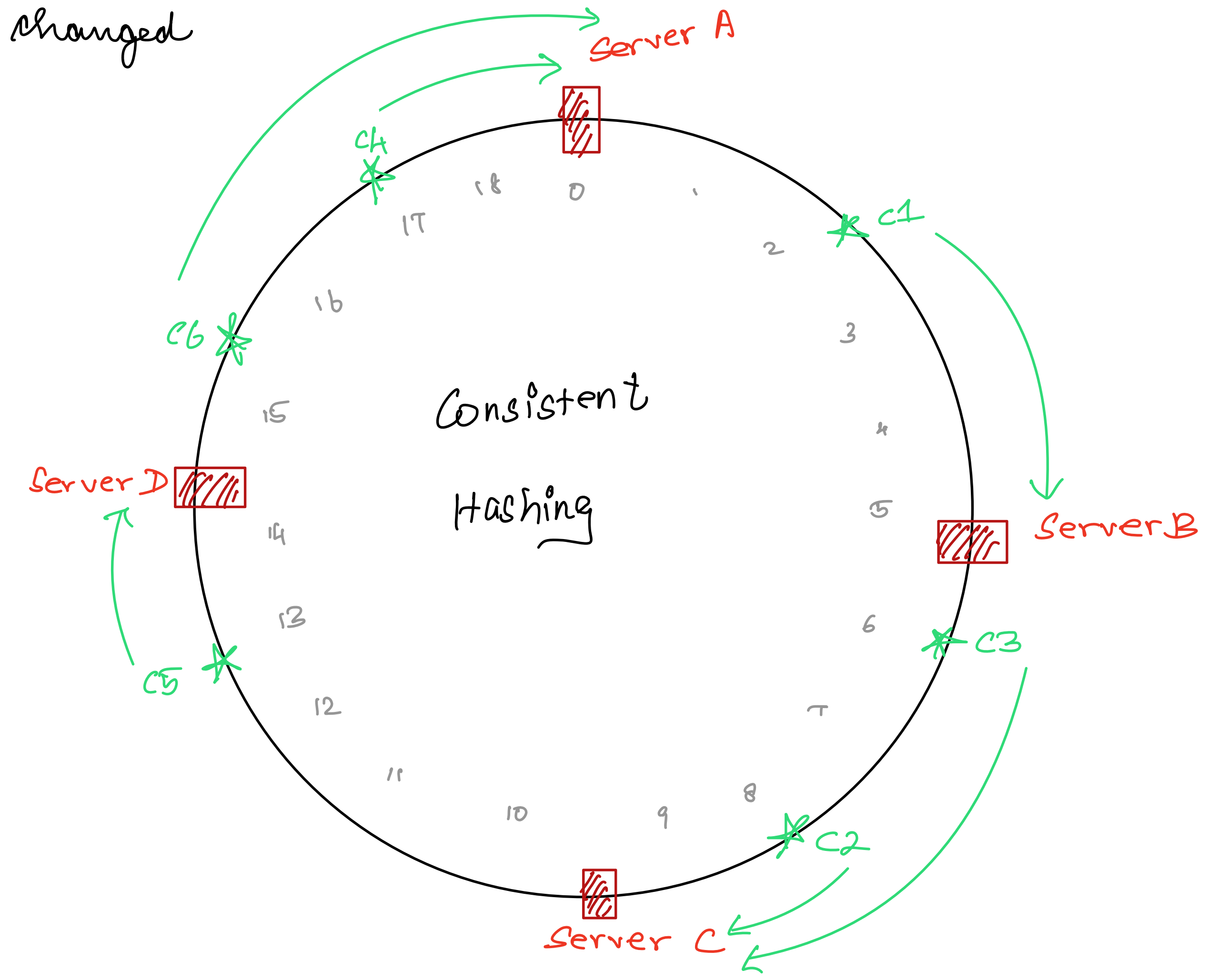
Eg: If 1 new server is added

C1 => 11 % 5 => 1 => Server B (Previously to server D)

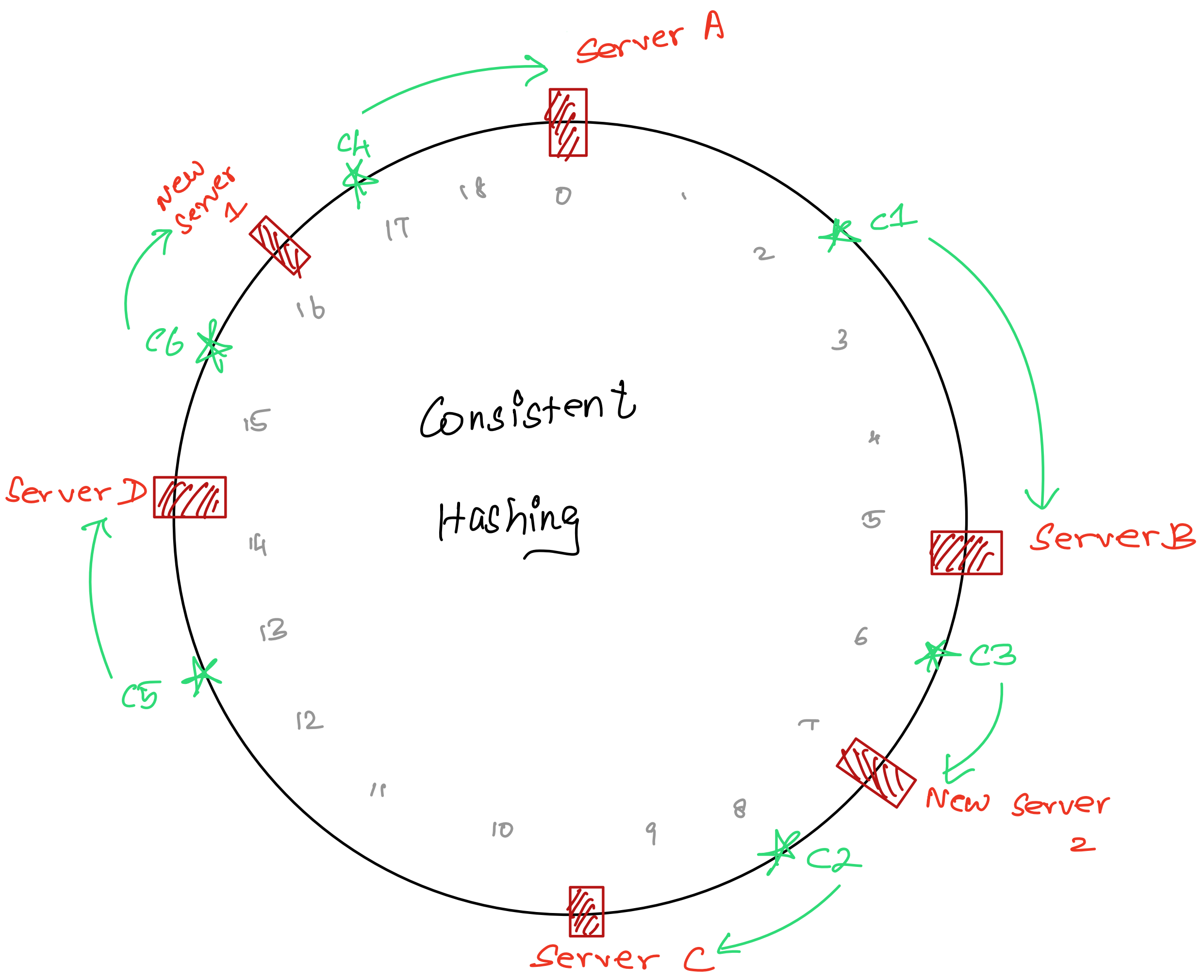
This is where consistent hashing and rendezvous hashing come into play.

Consistent Hashing

A type of hashing that minimizes the number of clients that needs to be remapped when the number of servers gets changed



Assume there are four servers and six clients
Both the servers and the clients are hashed
and depending on the hash value they are
placed on the circle (just for imagination)
All the clients will be directed to the
nearest servers (in clockwise direction (say))
Now when two new servers are added
(New Server 1 & New Server 2) most of the clients
will still be directed to the same servers
Thus there is not much of a shuffle.
as in the previous hashing technique



Rendezvous Hashing

This is also known as Highest Random weight Hashing.

Eg: **server-set**

server 1

server 2

user-set

user 1

user 2

Server 3 user 3

user 4

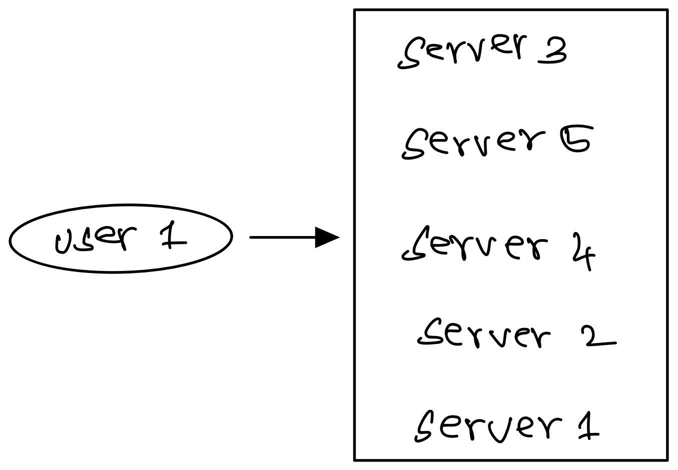
user 5

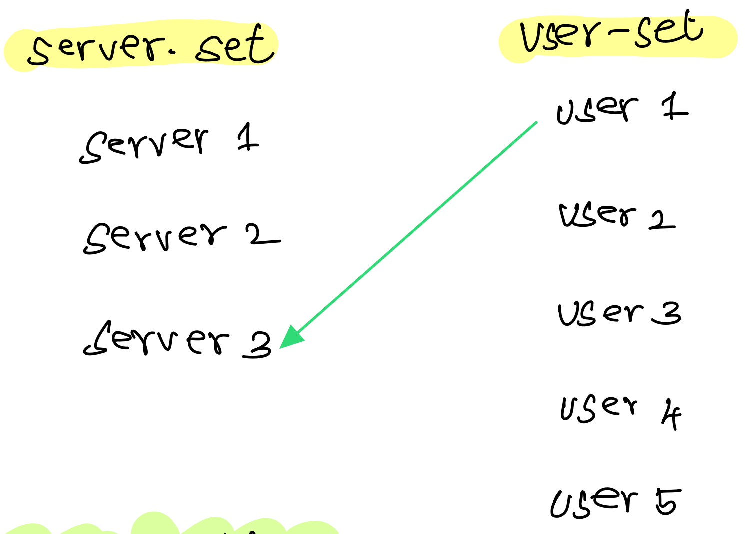
When the user 1 sends a request the key (say username) will be hashed and instead of selecting a single server based on the hash value a list of servers will be selected and the first server from the list will be assigned to the user.

If the first server goes down then the 2nd server in that list will be used

In this way Rendezvous Hashing allows for minimal redistribution of mapping when a server goes down.

User - 1



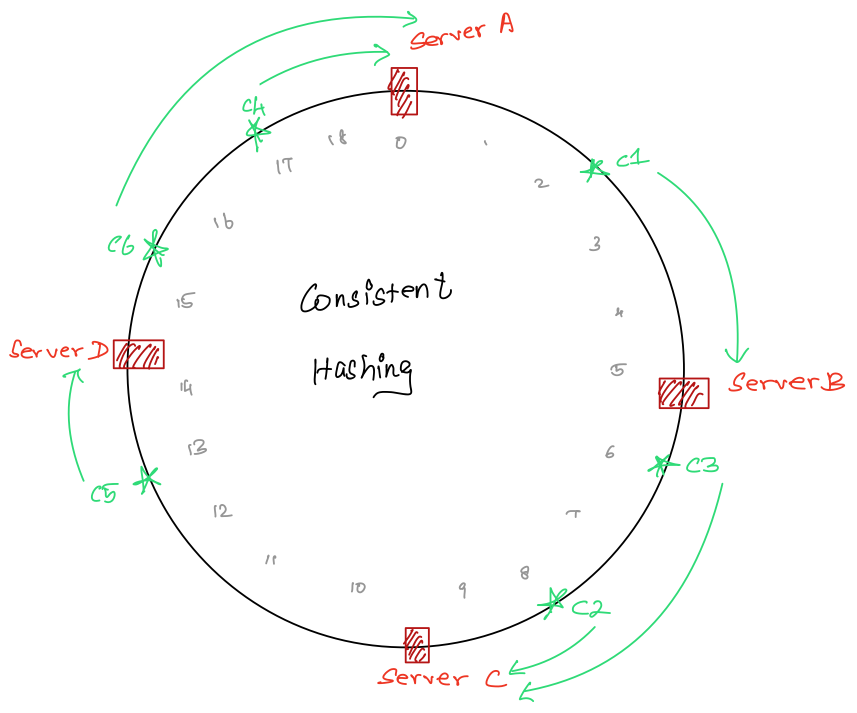


Disadvantages of Rendezvous Hashing:

Rendezvous hashing is suitable for medium size distributed system where O(N) look up cost is not prohibitive

→ For every user all the servers are ranked thus O(N)

Advantages of Rendezvous Hashing :



Assume that server - A goes down then all the load of server - A will be redirected to server - B . Thus consistent hashing does not ensure equal distribution of the load.

The advantage of rendezvous hashing is that even if a server goes down the load of that server will be distributed evenly to all the servers thus having a good load .

balancing performance.