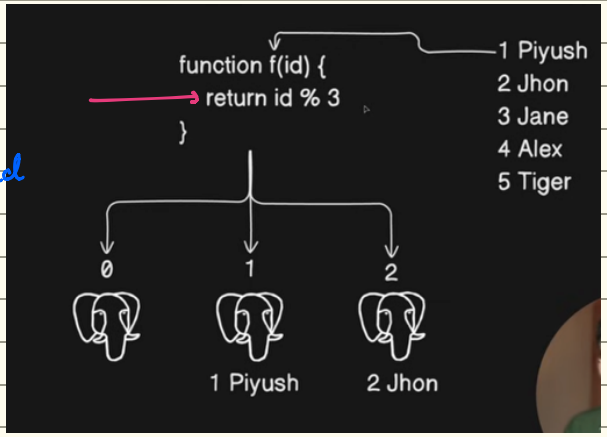
# ADVANCE SYSTEM DESIGN

## ↳ Consistent Hashing.

→ **Hash function:-** A Hash Function in System Design is a function that maps input data (keys) to a fixed-size value (usually an integer), called a hash. It helps store, locate, and retrieve data quickly.

Hash function
↓
Works as a load balancer.
↓
Helps to distribute and assign load.

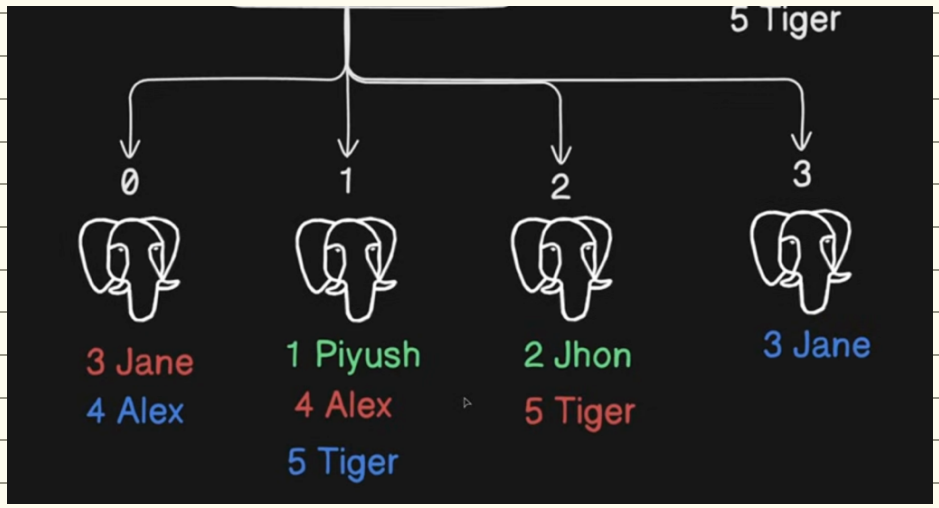



1 % 3 => 1

2 % 3 => 2.

~~id % num_of_servers.~~

Hash function




2 % 4 = 2

1 % 4 = 1

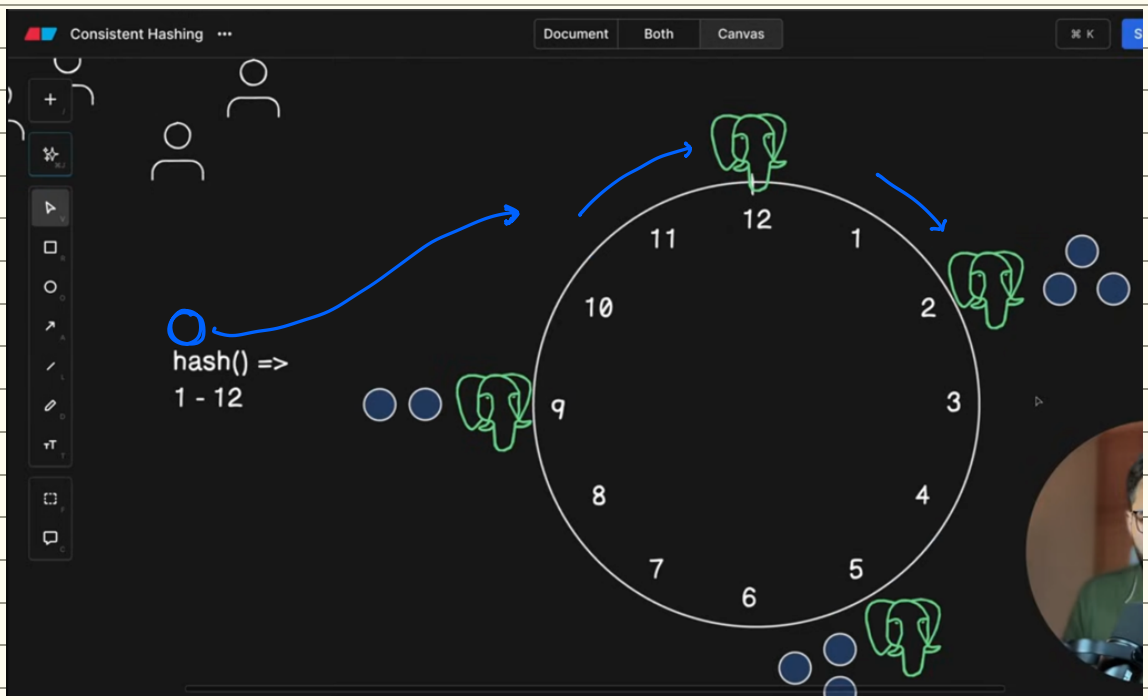
3 % 4 = 3

4 % 4 = 0.

The Hash function is changed in a running application, the keys were different before and now its different, and this causes data inconsistency.

## OBSERVATION:-

- If I scale from 3 servers to 4 servers.
- I need to move 3 keys in the servers for existing data.




The data/request first passes through the hash function, and then it moves in clock-wise direction and the nearest server in clock wise direction is allocated for process

→ Pros:-

- Data movement is minimal. Fraction of keys are moved.
- It is easy to compute which all keys need to be moved.

this mean?

To achieve horizontal scaling, it is important to distribute requests/data efficiently and evenly across servers. Consistent hashing is a commonly used technique to achieve this goal. But first, let us take an in-depth look at the problem.

## Hash space and hash ring

Now we understand the definition of consistent hashing, let us find out how it works. Assume SHA-1 is used as the hash function f, and the output range of the hash function is: *x0, x1, x2, x3, ..., xn*. In cryptography, SHA-1's hash space goes from 0 to 2^160 - 1. That means *x0* corresponds to 0, *xn* corresponds to 2^160 – 1, and all the other hash values in the middle fall between 0 and 2^160 - 1. Figure 3 shows the hash space.


Figure 3

By collecting both ends, we get a hash ring as shown in Figure 4:

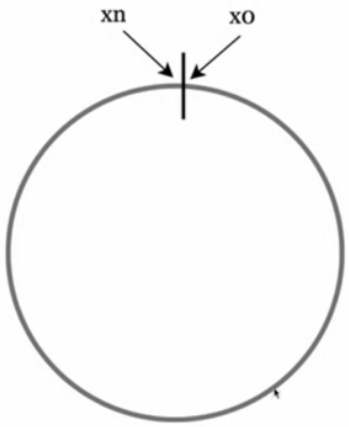
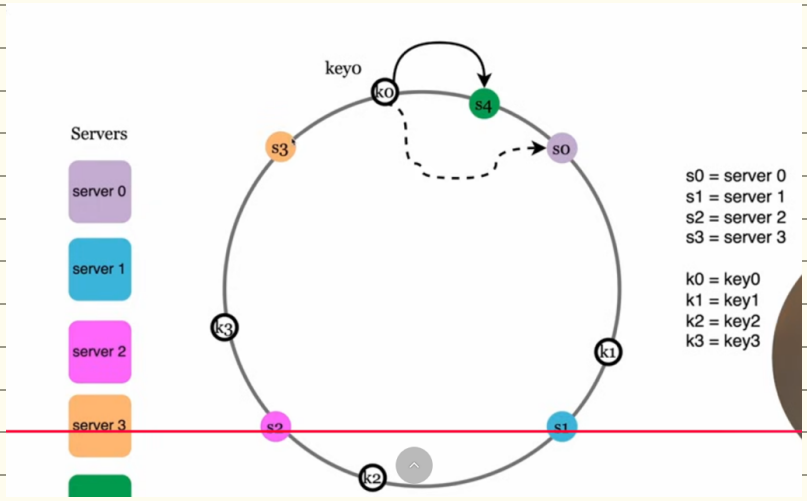


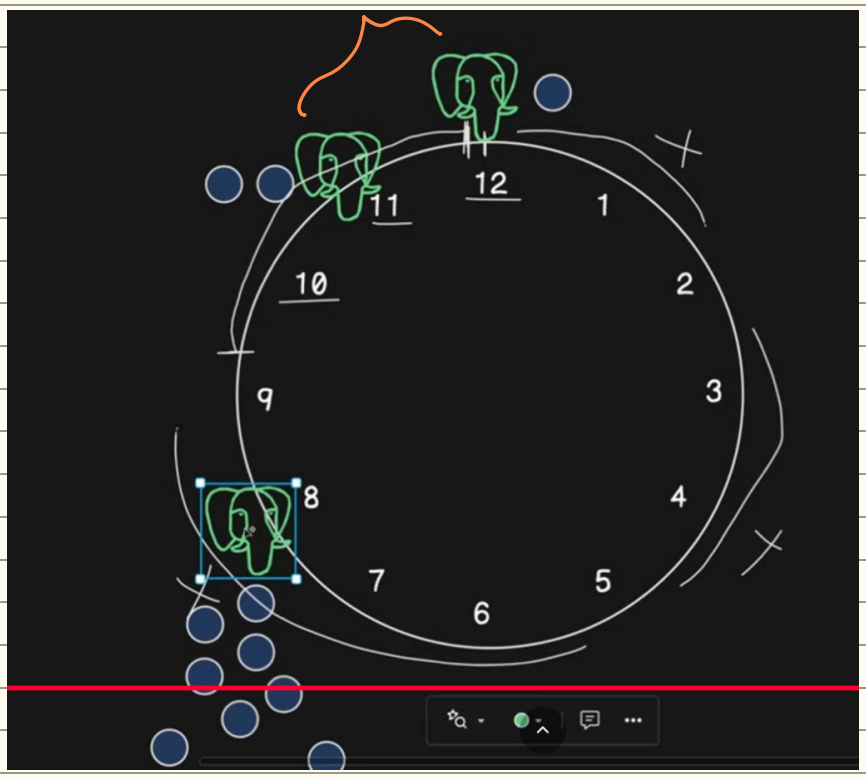

Figure 4

## Server lookup

To determine which server a key is stored on, we go clockwise from the key position on the ring until a server is found. Figure 7 explains this process. Going clockwise, *key0* is stored on *server 0*; *key1* is stored on *server 1*; *key2* is stored on *server 2* and *key3* is stored on *server 3*.

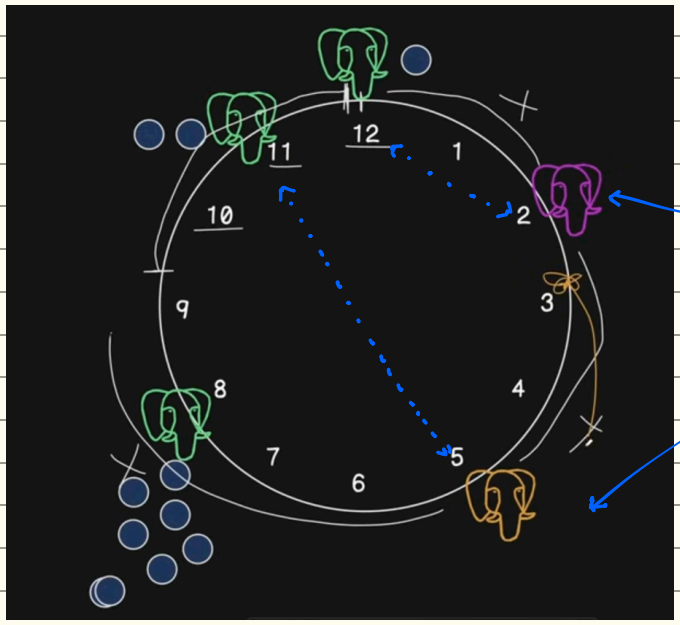



→ Cons




If the hash function places 2 servers very closely, then the request distribution will be disturbed and the distant server takes most of the load.

## → Virtual Nodes :-




Virtual Nodes.

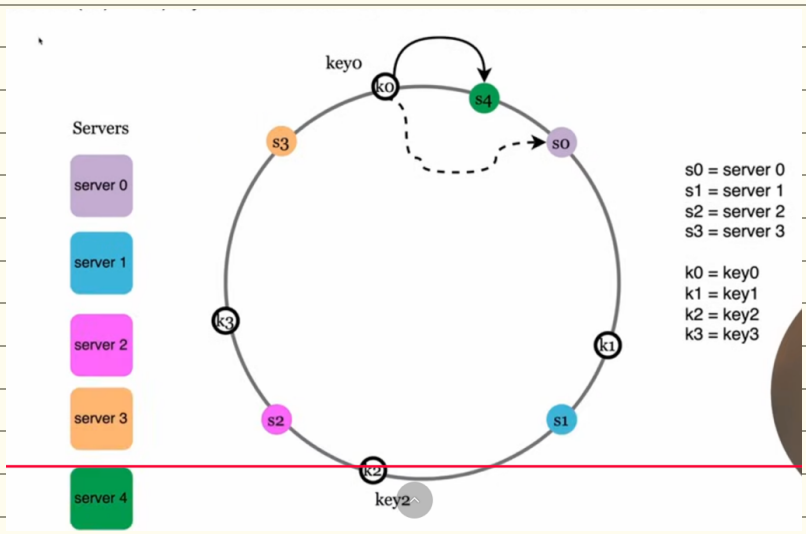
**Virtual Node :-**

A virtual node refers to the real node, and each server is represented by multiple virtual nodes on the ring. In Figure 12, both *server 0* and *server 1* have 3 virtual nodes. The 3 is arbitrarily chosen; and in real-world systems, the number of virtual nodes is much larger. Instead of using *s0*, we have *s0_0*, *s0_1*, and *s0_2* to represent *server 0* on the ring. Similarly, *s1_0*, *s1_1*, and *s1_2* represent server 1 on the ring. With virtual nodes, each server is responsible for multiple partitions. Partitions (edges) with label *s0* are managed by server 0. On the other hand, partitions with label *s1* are managed by *server 1*.

## → Finding Affected Keys :-

When a server is added or removed, a fraction of data needs to be redistributed. How can we find the affected range to redistribute the keys?

In Figure 14, *server 4* is added onto the ring. The affected range starts from *s4* (newly added node) and moves anticlockwise around the ring until a server is found (*s3*). Thus, keys located between *s3* and *s4* need to be redistributed to *s4*.



→ Summary :-

- Minimized keys are redistributed when servers are added or removed.
- It is easy to scale horizontally because data are more evenly distributed.
- Mitigate hotspot key problem. Excessive access to a specific shard could cause server overload. Imagine data for Katy Perry, Justin Bieber, and Lady Gaga all end up on the same shard. Consistent hashing helps to mitigate the problem by distributing the data more evenly.

Consistent hashing is widely used in real-world systems, including some notable ones:

- Partitioning component of Amazon's Dynamo database [3]
- Data partitioning across the cluster in Apache Cassandra [4]
- Discord chat application [5]
- Akamai content delivery network [6]
- Maglev network load balancer [7]