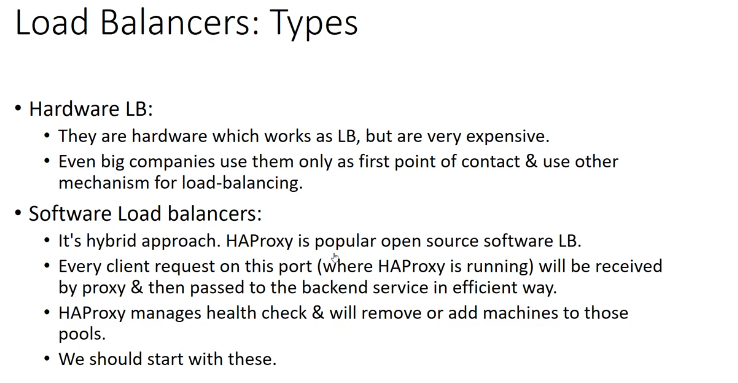
Load Balancers:

* It helps to distribute load across multiple nodes
* It also keep track of status of all the resources while distributing requests. If a server is not available, it stops sending traffic.
* Where it can be added? User- Web server, web server – Internal server, Internal server – Database.



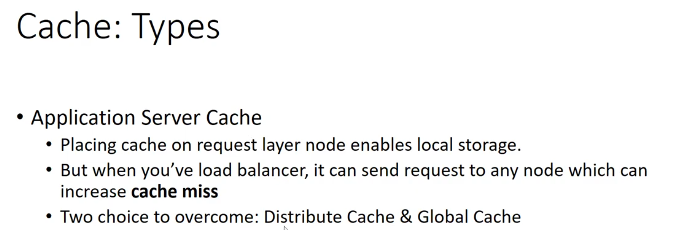


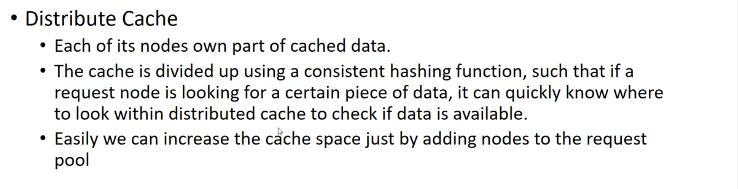
**System Design - Cache | Caching | Cache Invalidation | Cache Eviction**

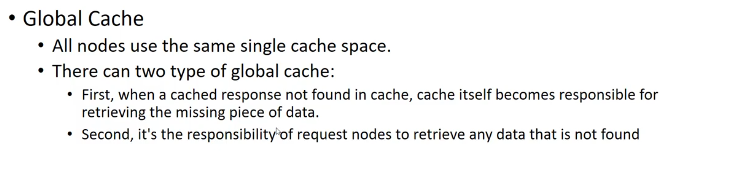
* Caching works on locality of reference principle: recently requested data is likely to be requested again.
* Its like short-term memory which has limited space but is faster & contains most recently accessed items.
* Cache can be used in almost every layer: hardware, OS, web browsers, webapplication, but are often found nearest to the front end.

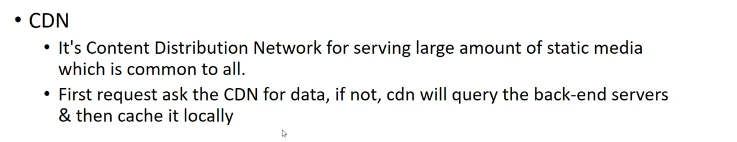
Types of Cache:

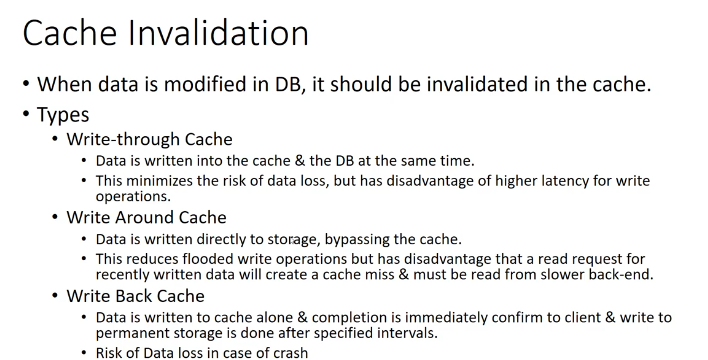
1. Application Server Cache
2. Distributed Cache
3. Global Cache
4. CDN





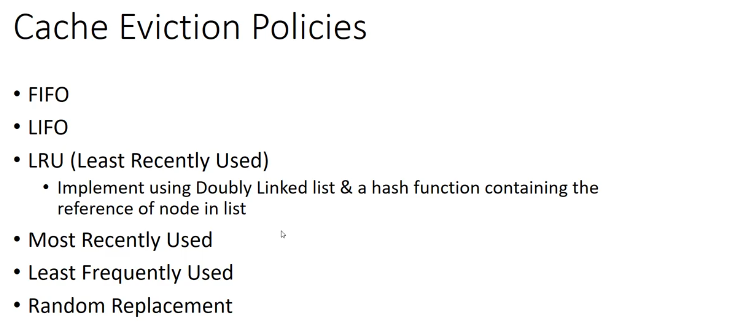




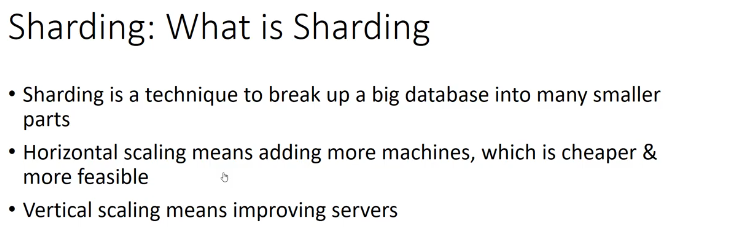


If the cache is full then how to delete entire in cache. Using below cache Eviction policies.

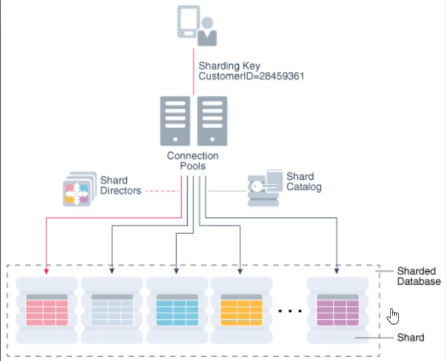
LRU is best policies.

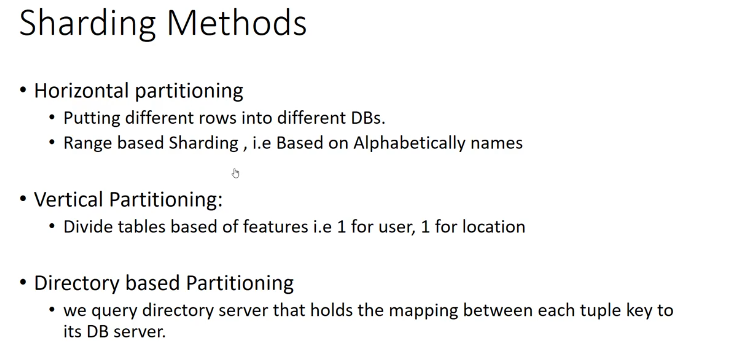


# System Design – Sharding

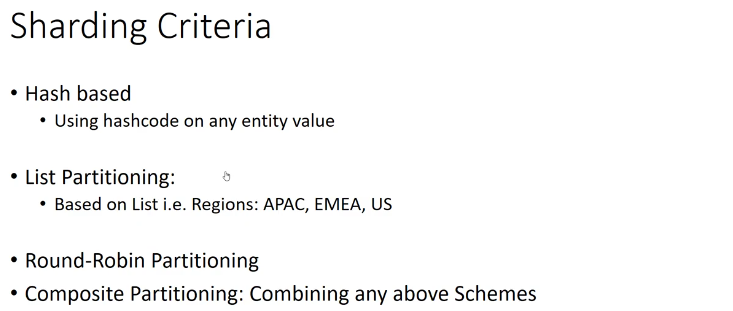


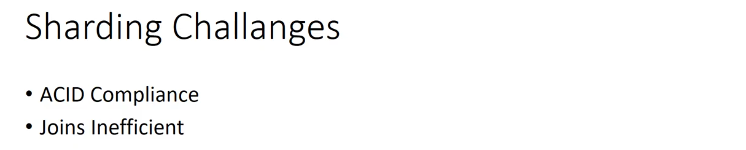
Sharding is done mainly in DB ( data layer side)





* Each entry has DB id.





ACID

To have optimal transaction control, a database system must be ACID compliant, which stands for the following properties: Atomicity, Consistency, Isolation, Durability. The definitions of the ACID properties are:

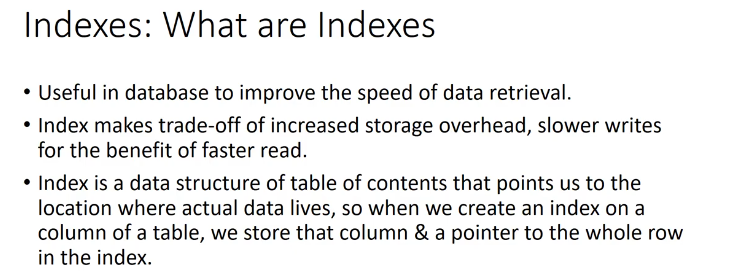
**Atomicity**: A transaction must be completed in its entirety or not at all. If a transaction aborts in the middle, all operations up to that point must be completely nullified.

**Consistency**: A transaction must transform a database from one consistent state to another consistent state. In other words, all data in a database must work as a state machine. The database must ensure all data is consistent at all times with all rules.

**Isolation**: Each transaction must occur independently of other transactions occurring at the same time. In other words, queries and transactions always run at a point in time. You can query data while many other users are changing data and you will not see their changes, and they will not see each other’s changes.

**Durability**: Committed transactions must be fully recoverable in all but the most extreme circumstances. Write-ahead logs provide absolute data durability until data is eventually written into permanent data and index files.

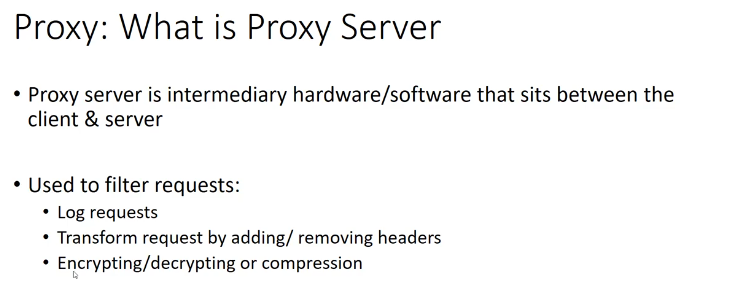
# System Design - Indexes | Indexing | Index benefits

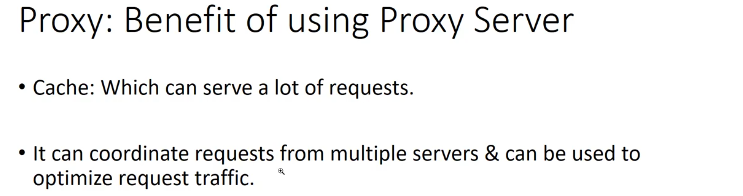




# System Design - Proxy Server

Blocking of the sits are done using proxy server.

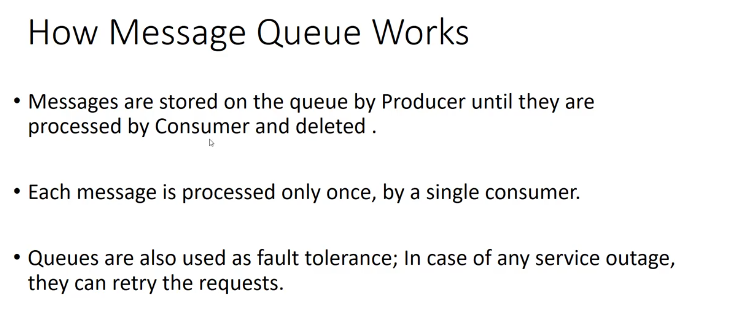


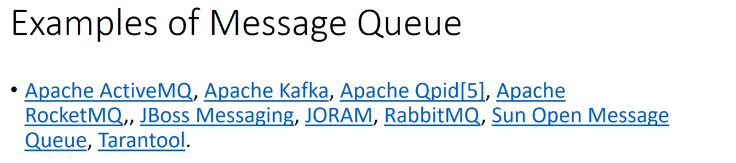




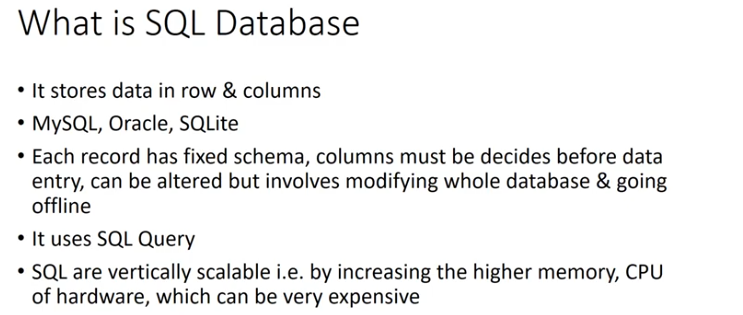
# System Design - Messaging Queue | Message Queue | Kafka

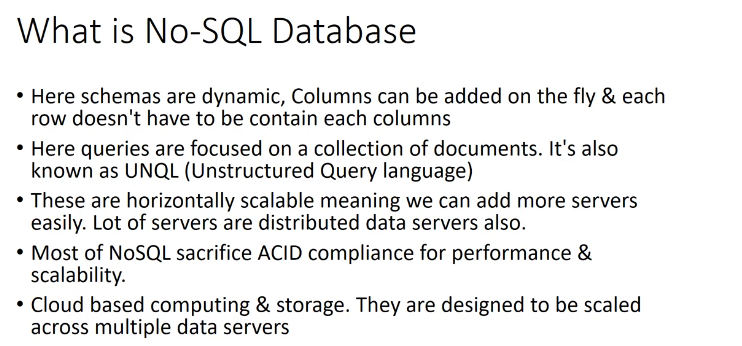
It’s asynchronous services-to-services communication used in serverless and microservices architecture.

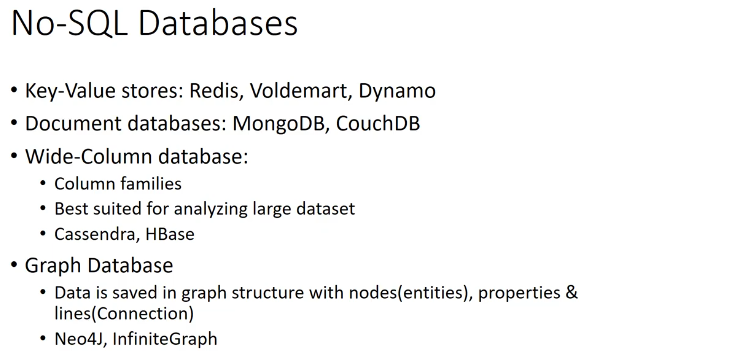


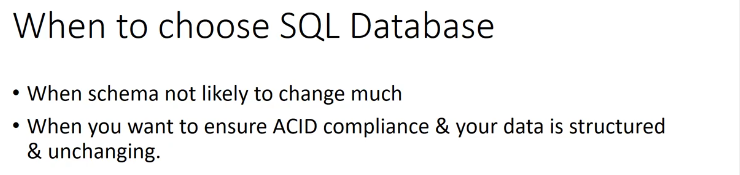


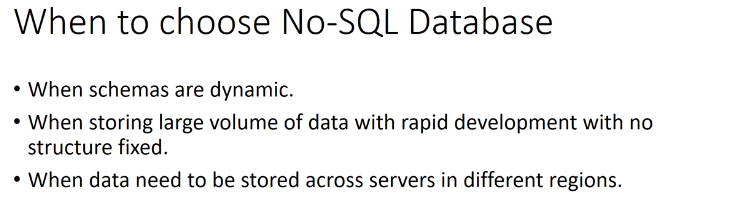
# System Design: Choosing Database b/w SQL vs NoSQL



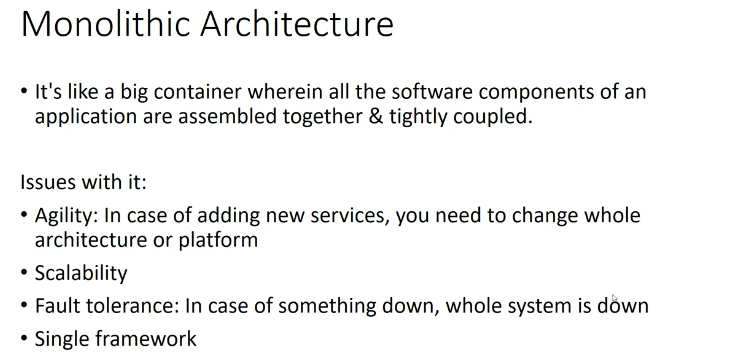


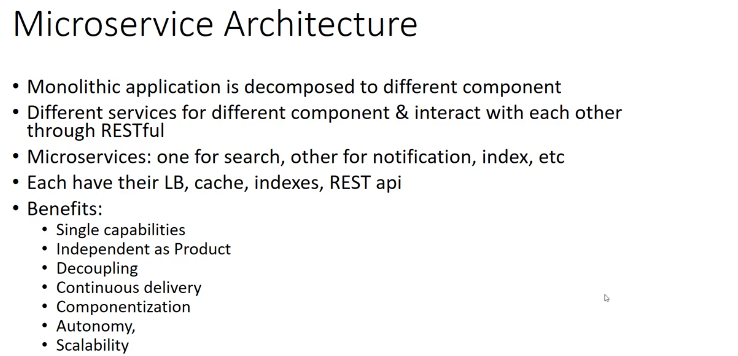






# System Design - Monolithic vs MicroServices Architecture







# REST APIs - System Design | What is REST APIs

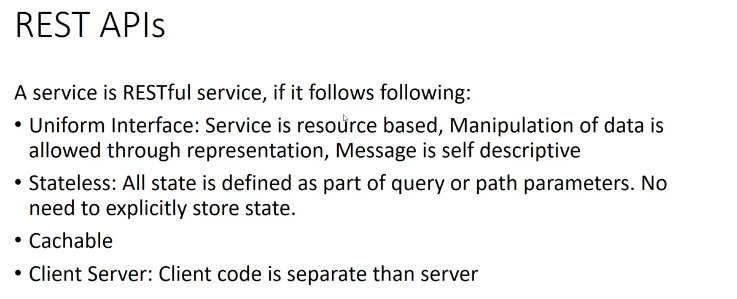
At the core of REST are resources, and resources are identified using **Uniform Resource Identifiers** (**URIs**). Conceptually, resources are separate from their representation.

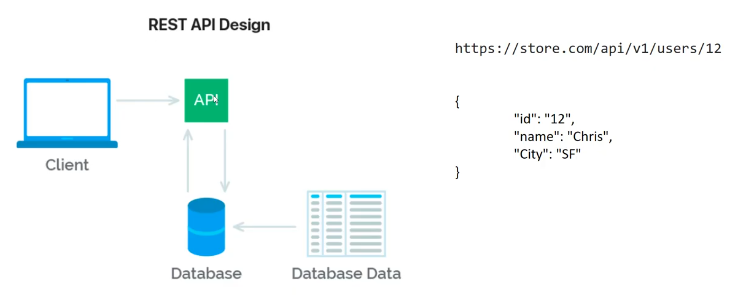
### Client-Server

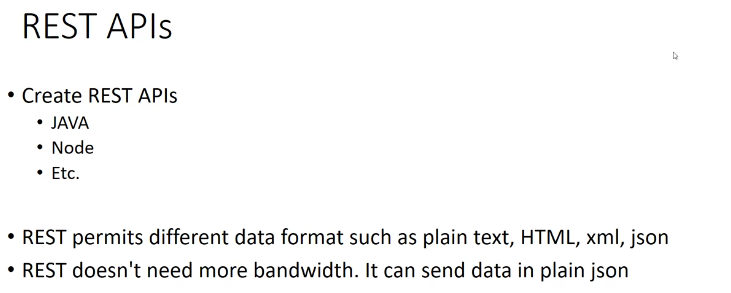
The client-server model that REST embraces enables the separation of client concerns, such as user interaction or user state management, from that of server concerns such as data storage and scalability.

### Layered system

Given the style of communication between clients and servers, clients are not aware of which specific server they are interacting with. This property allows the introduction of intermediate servers that can, for example, handle security or offer load-balancing capabilities.

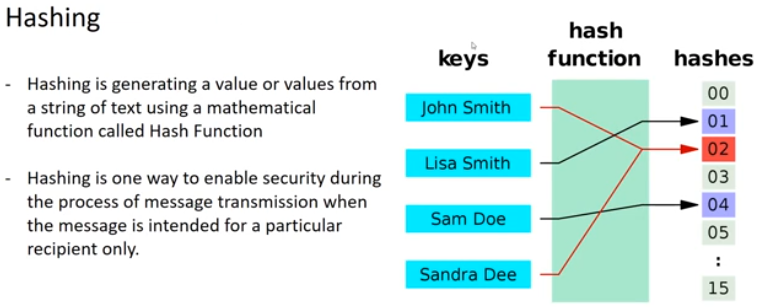


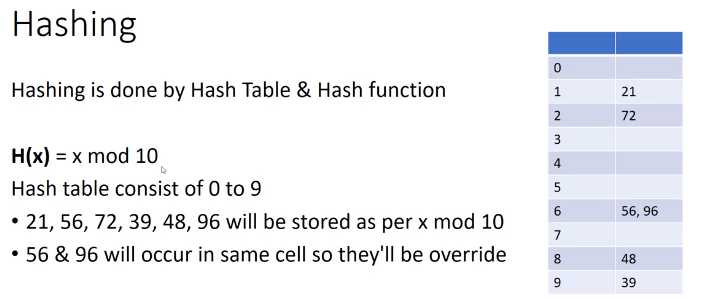


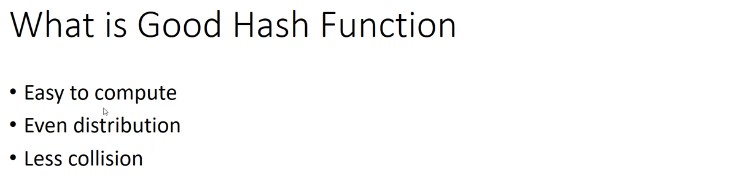


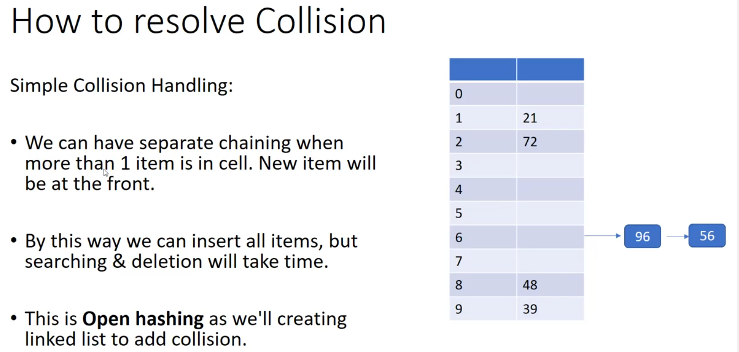
# Hashing - System Design | How Hashing Works

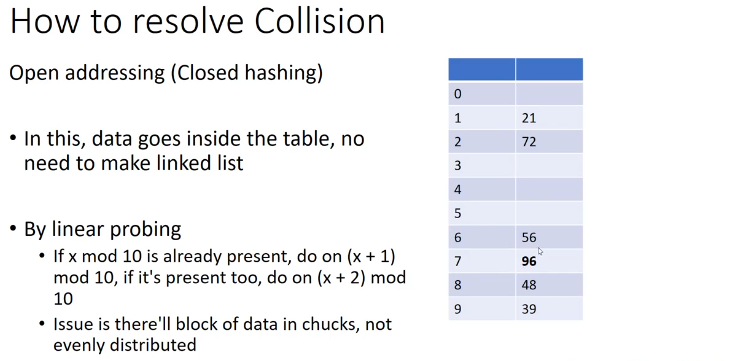


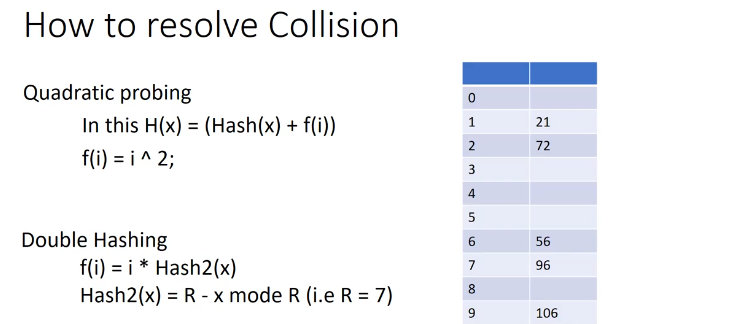




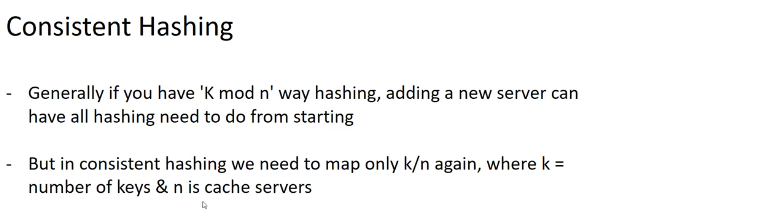


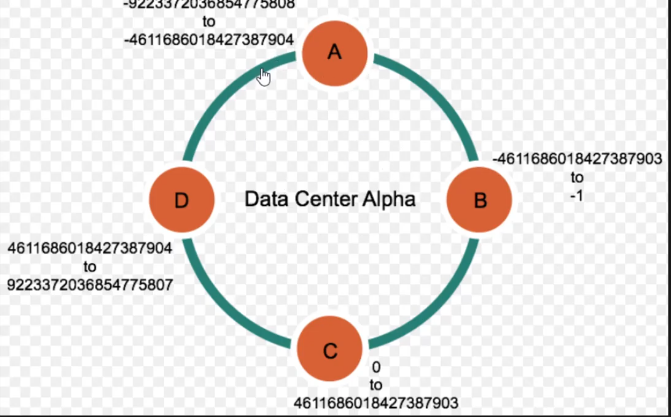


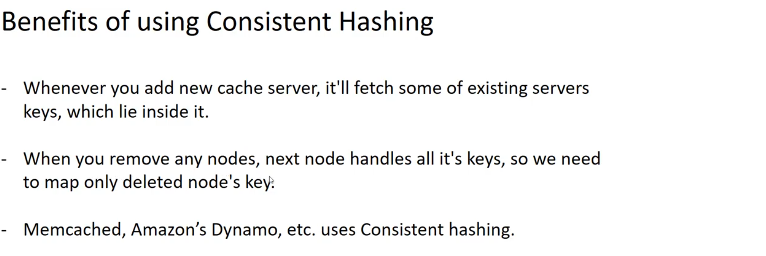




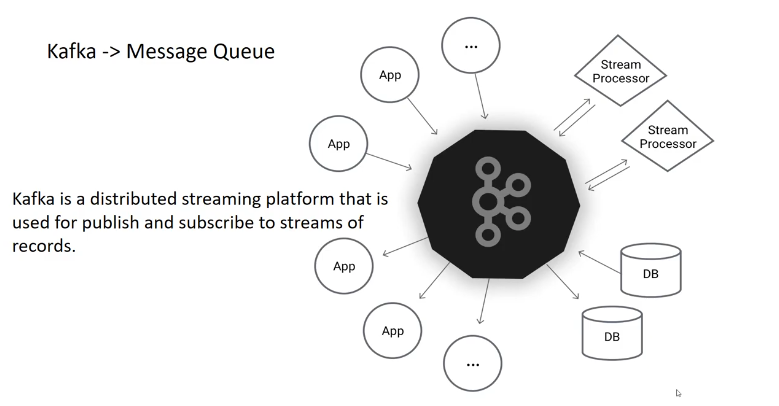
# Consistent Hashing - System Design | What is Consistent Hashing

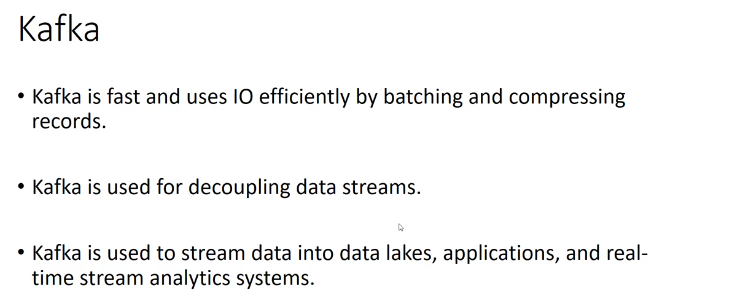


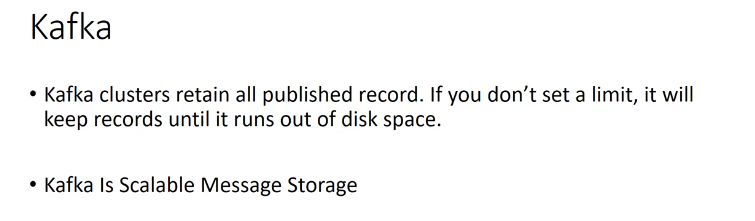


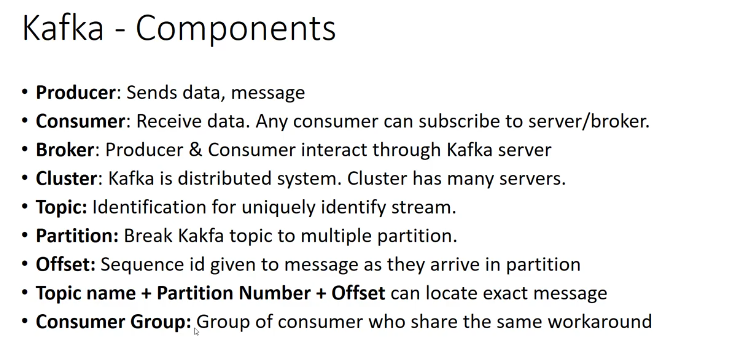


# Kafka - System Design | What is Kafka | Kafka Components

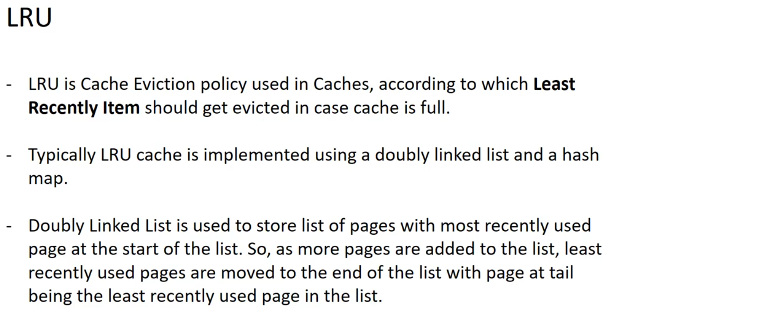


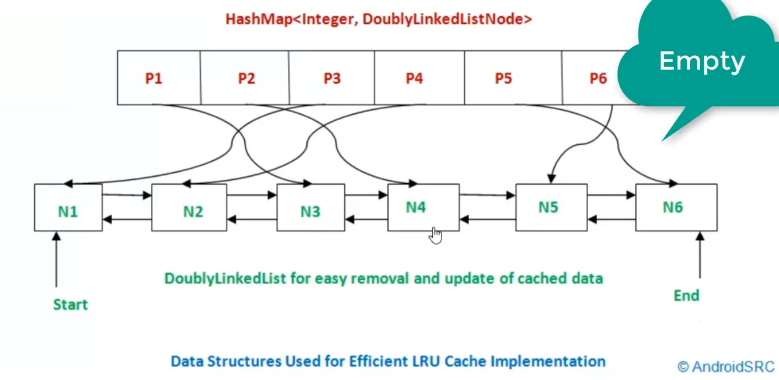


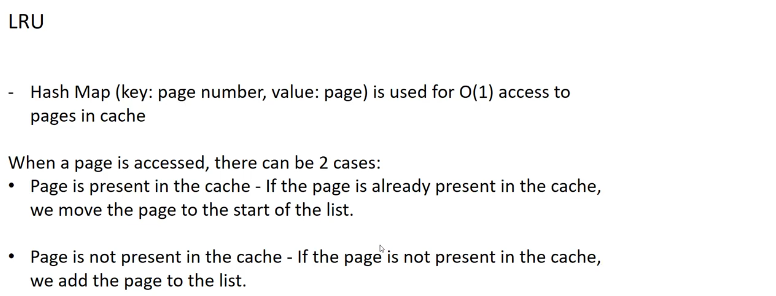


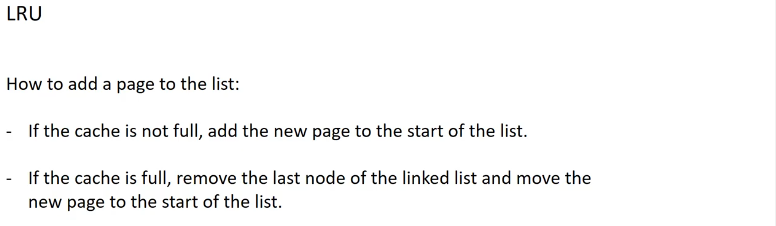


# LRU Cache - System Design | How LRU works









# Zookeeper - System Design | What is Zookeeper

