

Exchange : Rules that decide, msg will go in which Queue

Fan-out : Broadcast

Direct Exchange : match MK to RK.

Topic Exchange : wildcards are allowed. (\*)

Keys :

Message key : key that comes with the msg

Routing key : exchange key

Topic Exchange :

MK : 123qty

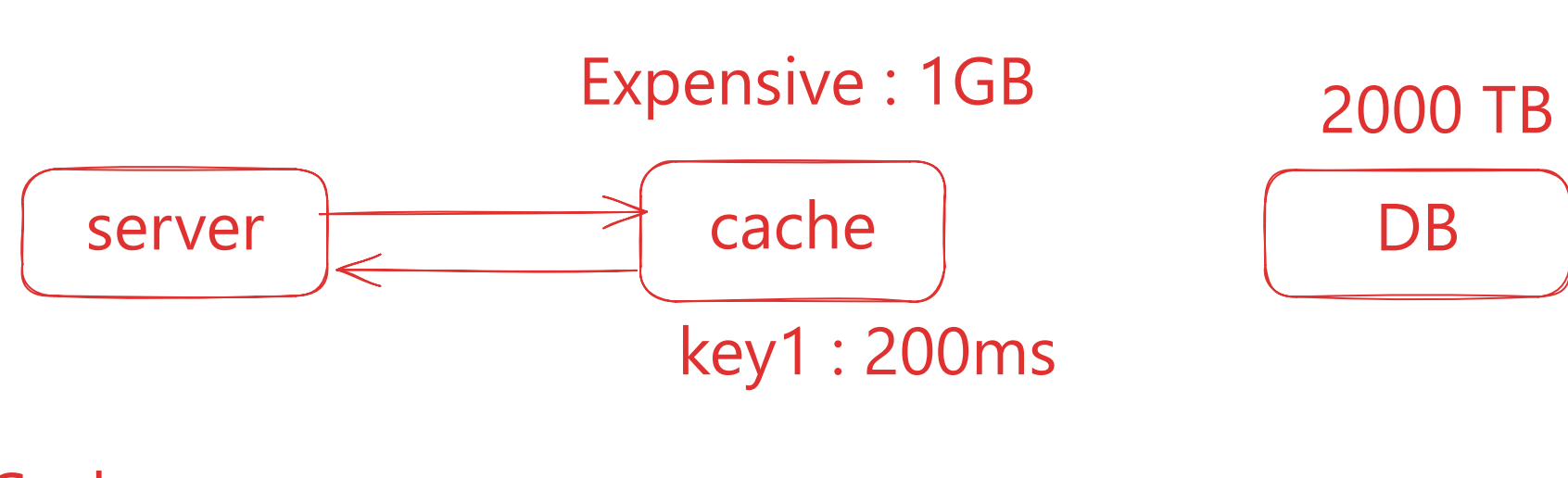
RK : 123q\*

Re-Queue mechanism

kafka : Pull based technique

RabbitMq : Push based technique

Caching



Cache :  
Temporary storage  
TTL (Time to Live)

DB : 200 ms  
Cache : 20 ms

Types of Cache:

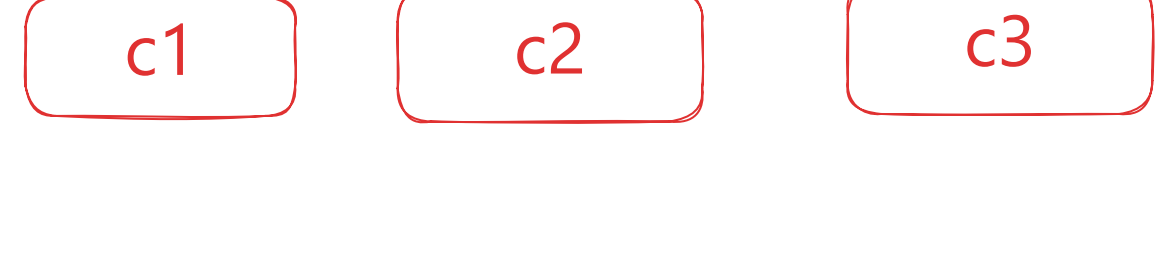
CDN --> caching use (Static resource)

Load Balancer (web pages)

Server side caching (Redis)

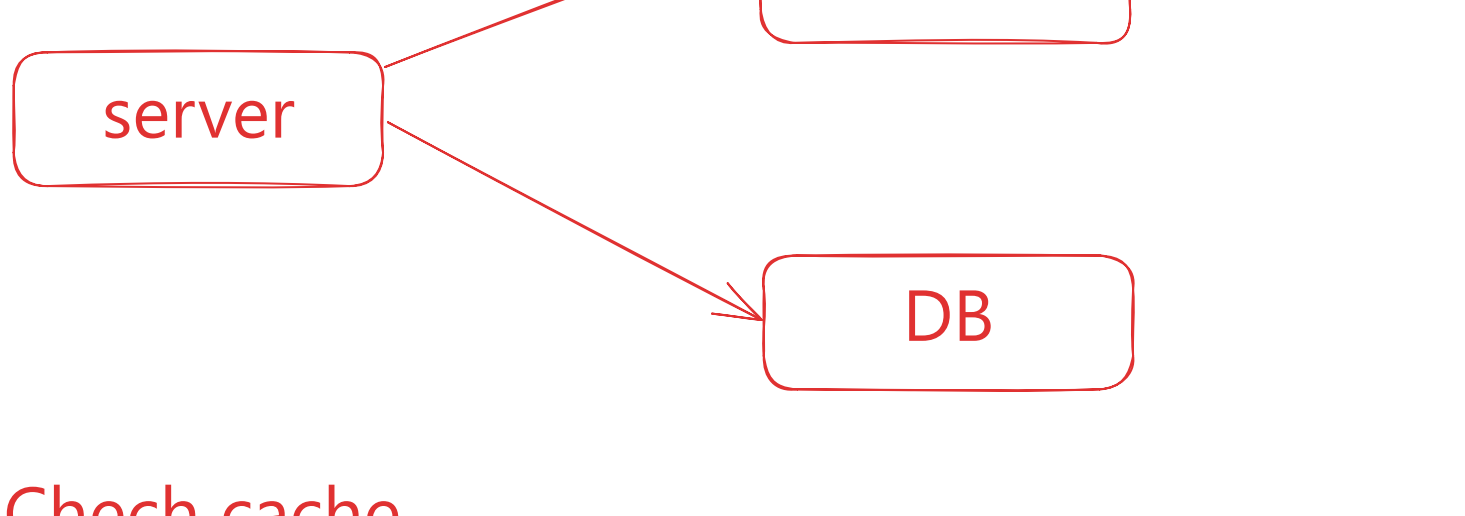
Proxies also use caching.

Distributed caching --> Consistent hashing



Cache Data POST/GET

1. Cache Aside technique (Retrieve)



1. Check cache
2. if found (cache hit) return the data
3. if not, (cache miss) fetch the data from DB, put it into the cache and then return thr data

Pros :

Simple to implement

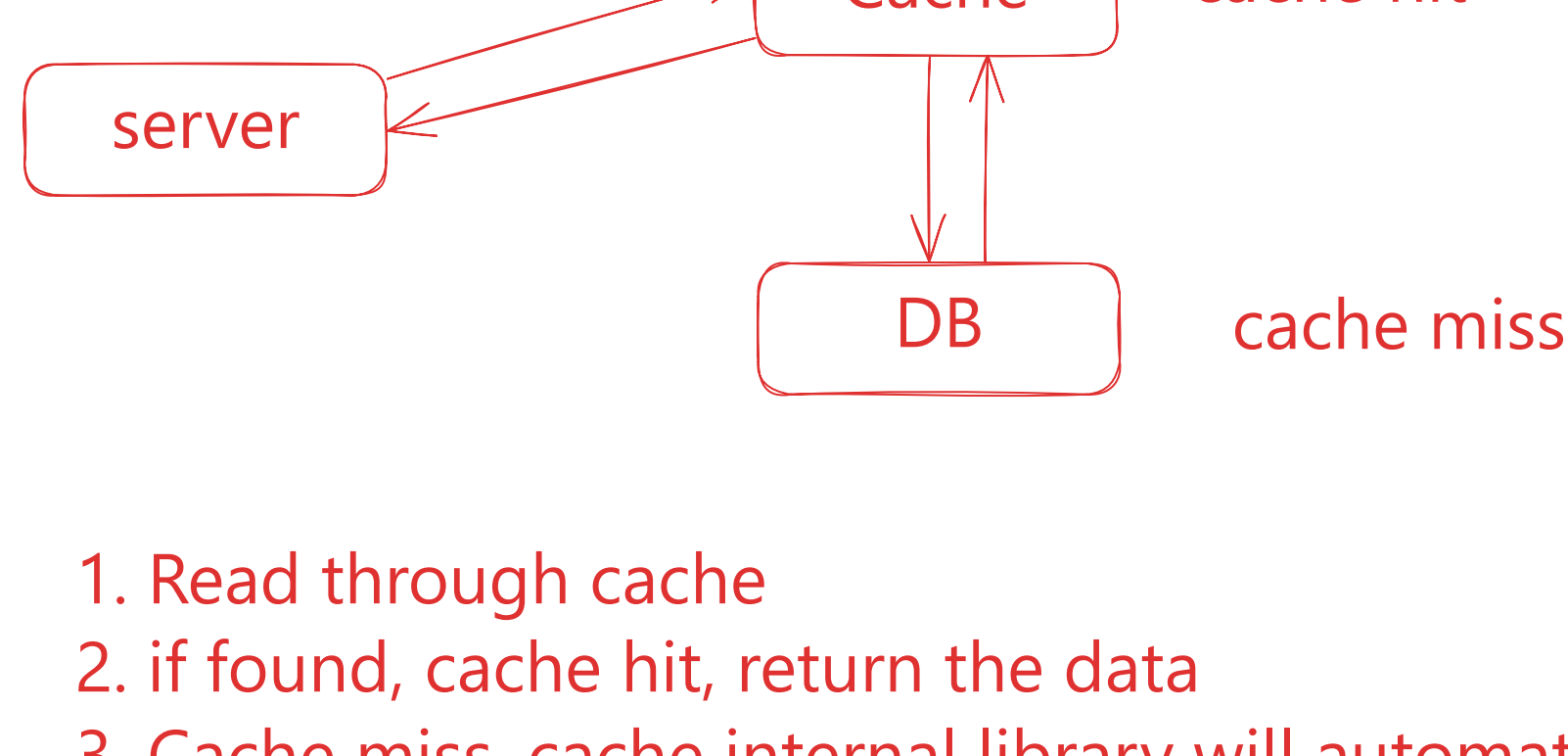
DB doc structure and cache can be different

Cons:

For every new data there will be always be cache miss.

Pre-heating the cache

Read through cache: (Data retrieval)



1. Read through cache
2. if found, cache hit, return the data
3. Cache miss, cache internal library will automatically fetch the data from DB, and put it in cache and return to the server

pros :

Client don't need to bother about DB logic.

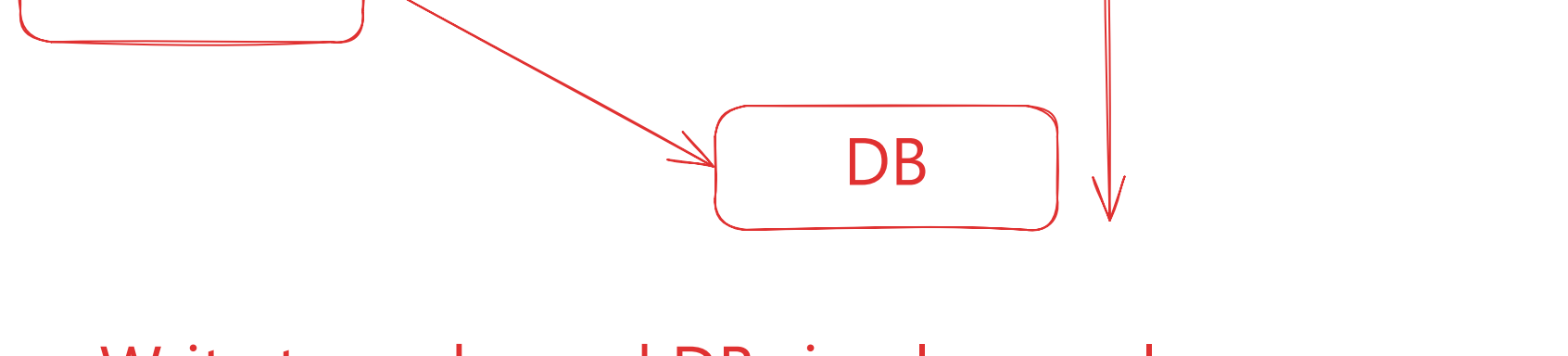
Cons :

Cache doc structure should be same as DB.

Starting --> cache miss.

Pre-heat the cache.

Write through cache : (To POST data)



Write to cache and DB simultaneously.

pros : Consistency will be high.

COns:

Slow.

uses 2 phase commit (overhead)

Write back/behind cache:



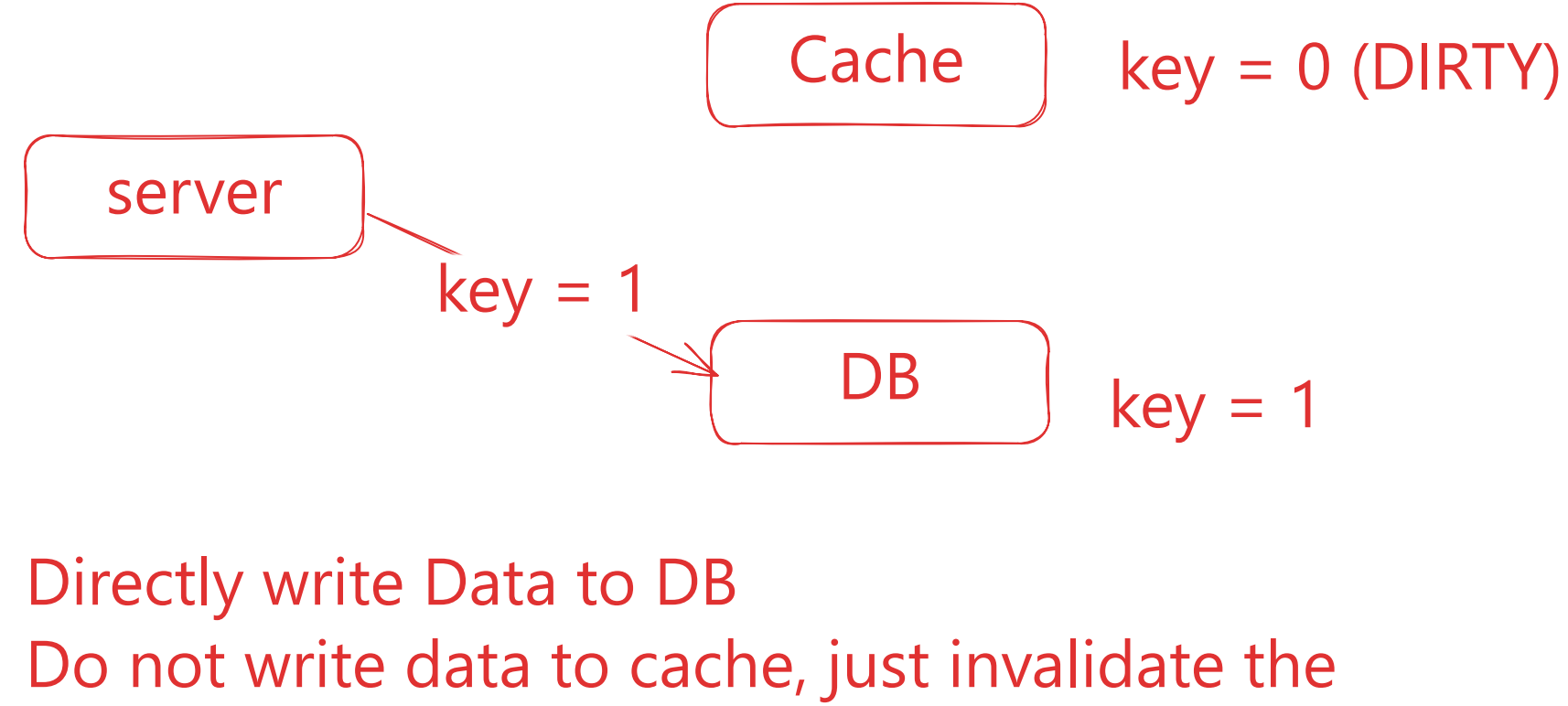
Write into cache and async write to DB

Pros: Fast

Disadvantage :

Inconsistency can come.

Write Around technique



Directly write Data to DB

Do not write data to cache, just invalidate the data in cache. (mark it as dirty)

Advantage :

without using any 2PL, it can maintain consistency.

Cache Eviction Policies



TTL

How to and what to remove data from cache.

LRU Cache (Least Recently used)

MRU (most recently used)

LFU (Least Frequently used)

FIFO( First in first out)

Random

LRU ( Least Recently used )

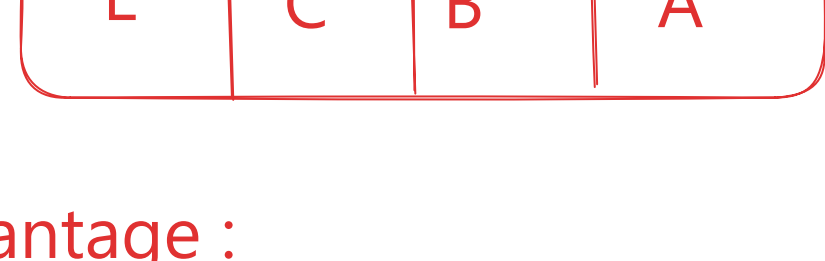
when the cache is full, LRU evict the item that hasnt been accessed in the longest time.



Pros : simplicity

cons : Assume past record patterns will repeat in future.

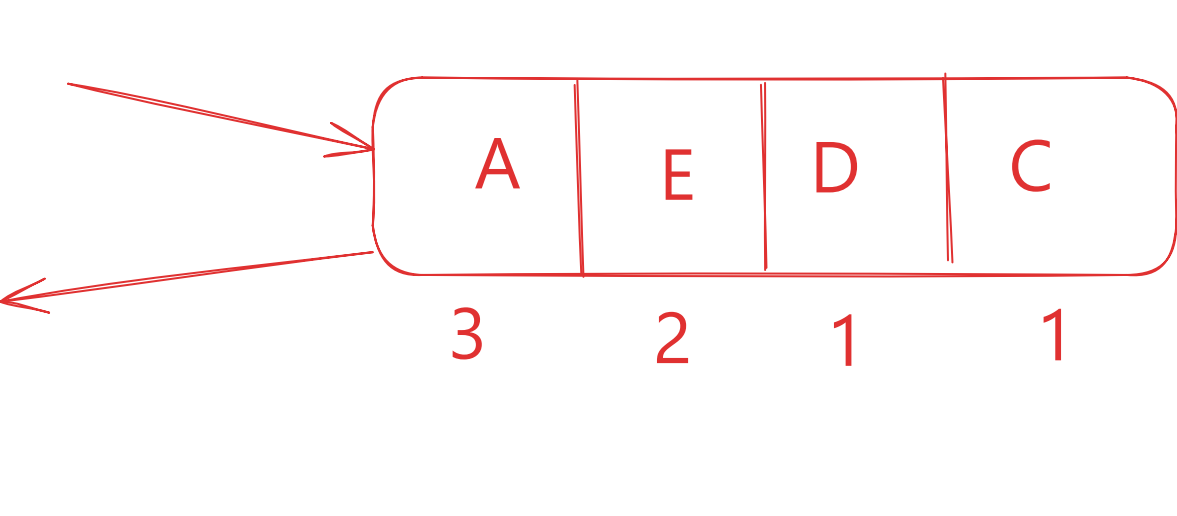
Most recently used (MRU)



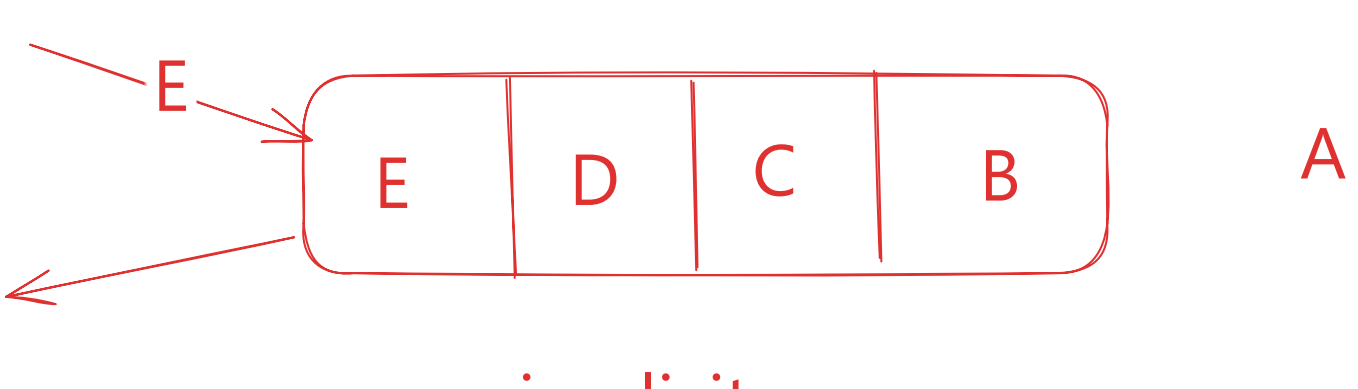
Disadvantage :

Do not bother about past records.

Least Frequently used (LFU)



FIFO (First in- first out)



pros : simplicity

Do not need to maintain any timestamp or frequency

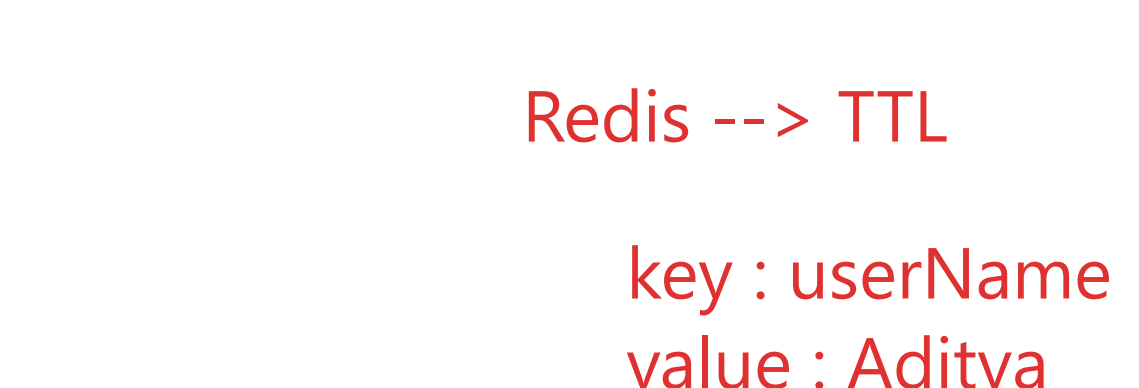
Cons :

Cache miss will be higher.

practically : FIFO

Assumptions --> Algorithms

Random cache eviction



Redis --> TTL

key : userName  
value : Aditya  
ttl : cache specific --> Redis

auth token  
user id  
session id

time bounded

Load balancers

CDN --> dynamic caching also supported.

API gateway

Proxies

Server side caching

client side caching --> cookies

Authenticaton & auhtorization HLD