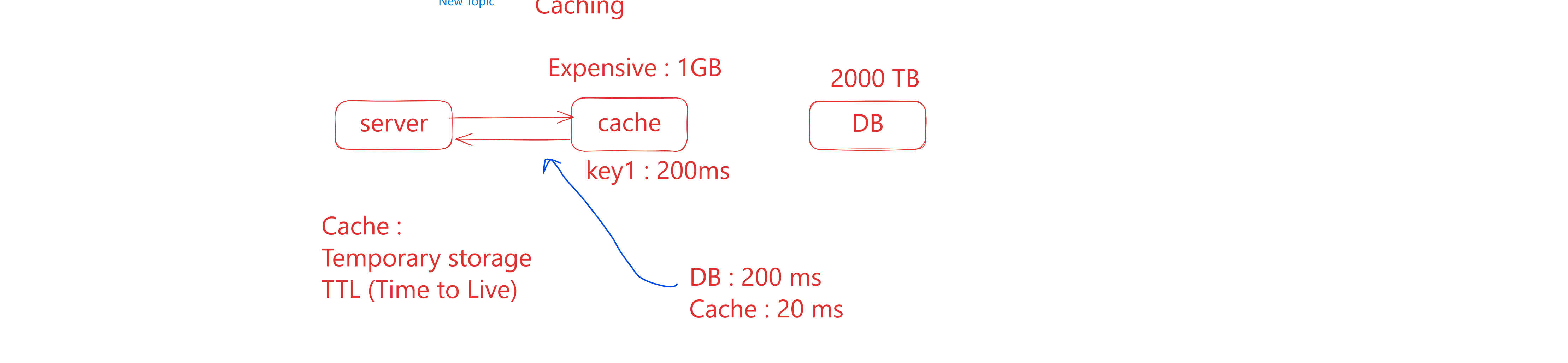


**Re-Queue mechanism** if requeue mechanism fails then it goes to dead letter queue

**difference**

kafka : Pull based technique

RabbitMq : Push based technique



**Types of Cache:**

CDN --> caching use (Static resource) and dynamic but difficult

Load Balancer (web pages)

Server side caching (Redis)

Proxies also use caching.

**Distributed caching --> Consistent hashing**

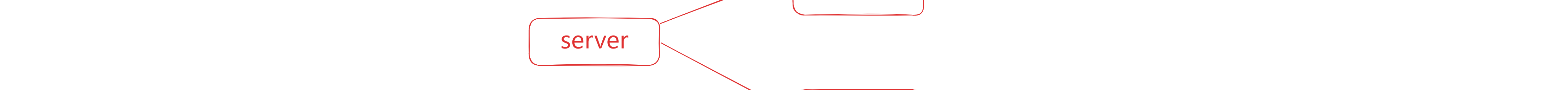
uses

since servers are distributed



**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 the data

**Pros :**

Simple to implement

DB doc structure and cache can be different

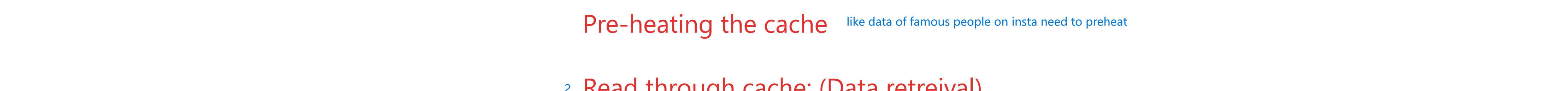
i.e. user id data type may be int i.e. user id data type may be string

**Cons:**

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

**Pre-heating the cache** like data of famous people on insta need to preheat

**2. 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 :**

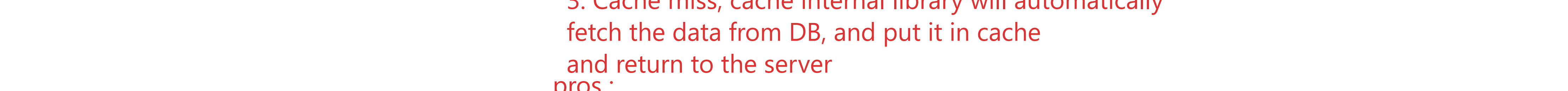
Cache doc structure should be same as DB.

Starting --> cache miss.

Pre-heat the cache.

solution

**Write through cache : (To POST data)**



**1. Write to cache and DB simultaneously.**

**pros :** Consistency will be high.

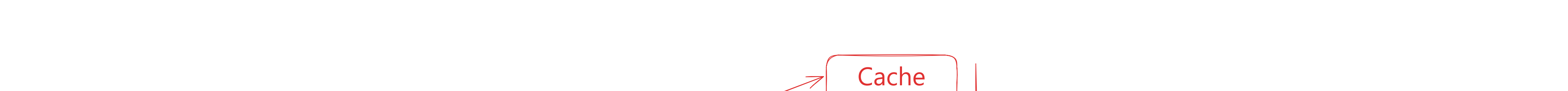
**CONs:**

Slow.

uses 2 phase commit (overhead) slow

locking mechanism to lock distributed servers

**2. Write back/behind cache:**



**Write into cache and async write to DB**

**Pros:** Fast

**Disadvantage :**

Inconsistency can come.

**3. 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.

2 phase

**Cache Eviction Policies**



**TTL**

How to and what to remove data from cache.

LRU (Least Recently used)

MRU (most recently used)

LFU (Least Frequently used)

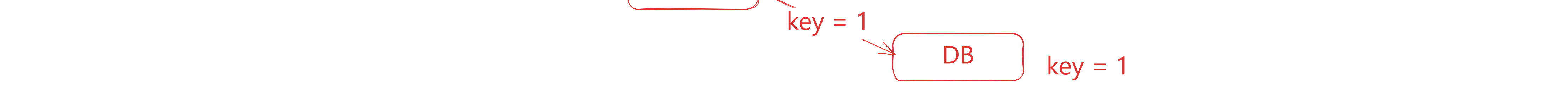
FIFO (First in first out)

Random

**LRU ( Least Recently used )**

when the cache is full,

LRU evict the item that hasnt been accessed in the longest time.



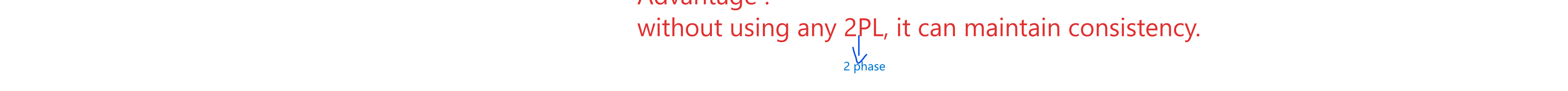
**Pros :** simplicity

**cons :** Assume past record patterns will repeat in future.

evicts B assuming B will not be used in future

**Most recently used (MRU)**

removes most recently used



**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.

cache is almost same as LRU or LFU but FIFO is better

**practically :** FIFO

**Assumptions --> Algorithms** in HLD

**Random cache eviction**



**Redis --> TTL**

key : userName

value : Aditya

ttd : 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

**Authentication & authorization HLD**

caching is used everywhere