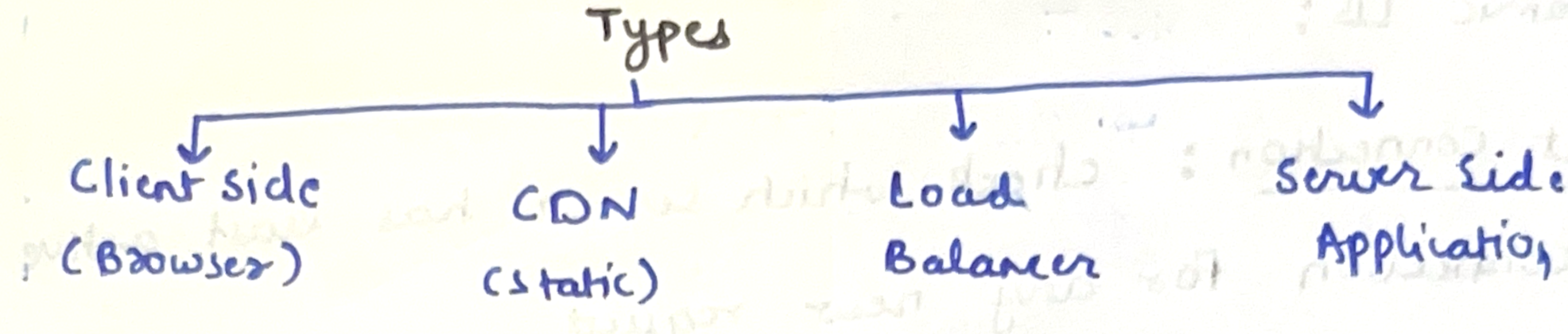
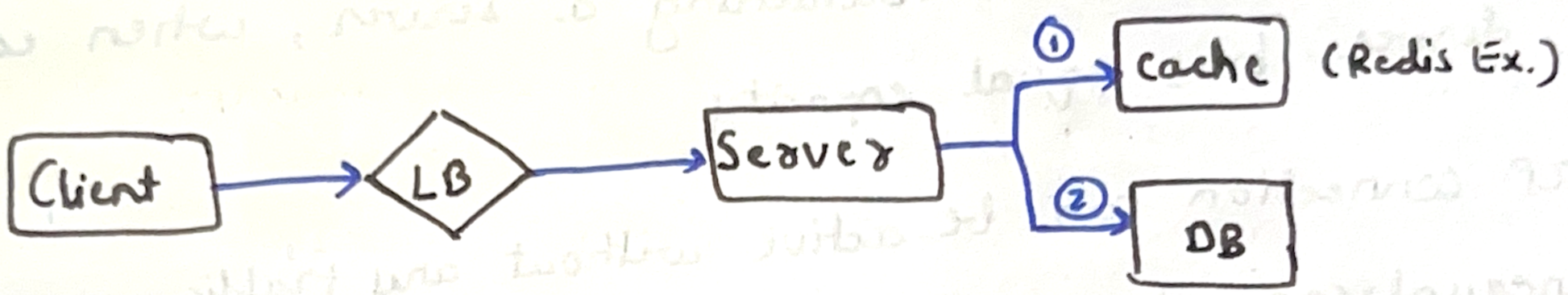
caching :-

⇒ Technique to store frequently stored data in a fast access memory.





in case of distributed System we have ~~Redis~~ cache client and n number of cache servers, there would be consistent hashing implimentation to manage these servers.

↳ this would make our cache scalable and helps single Point of failure

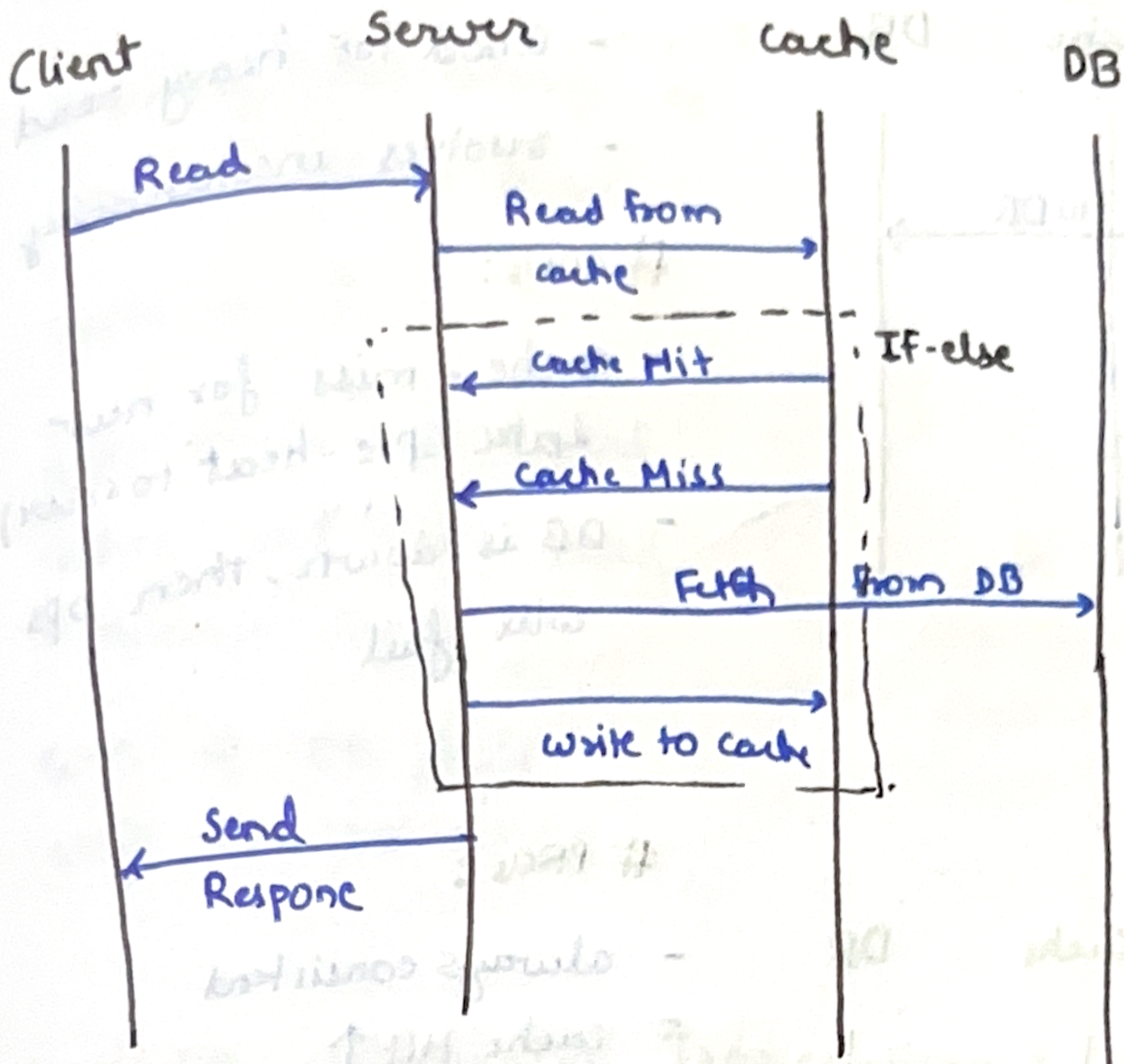
Eviction policy

- LRU
- FIFO
- LFU, etc.

⇒ for each entry there is

TTL (time to leave) ↳ after this entry gets evicted/ invalidated.

Cache Aside :



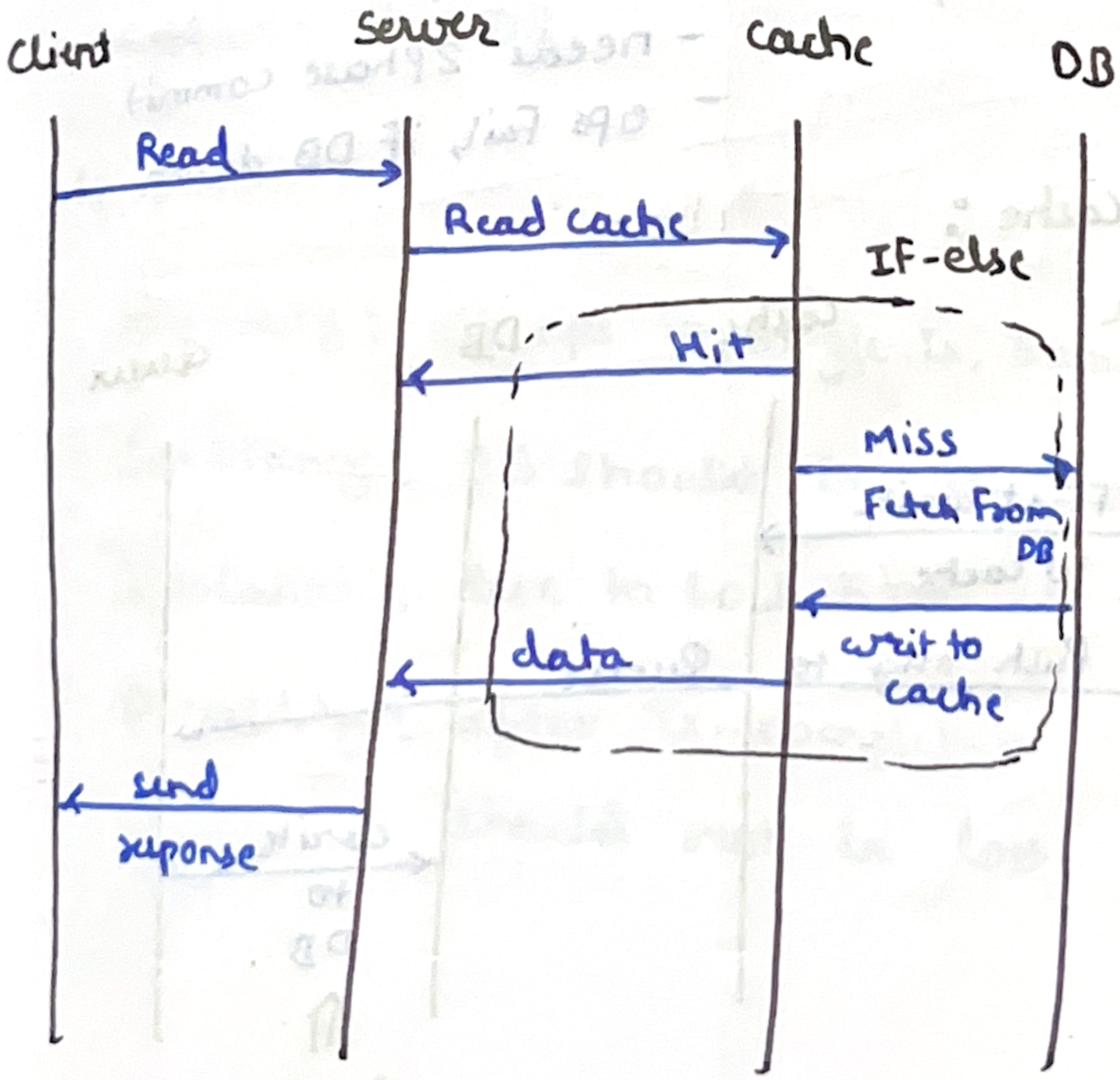
Adv :

- good for Heavy read
- request won't fail even if cache is down
- cache doc. can be diff from DB

Cons :

- For new data, always cache miss (Pre-Head to solve)
- Inconsistency in Put.

Read through cache :



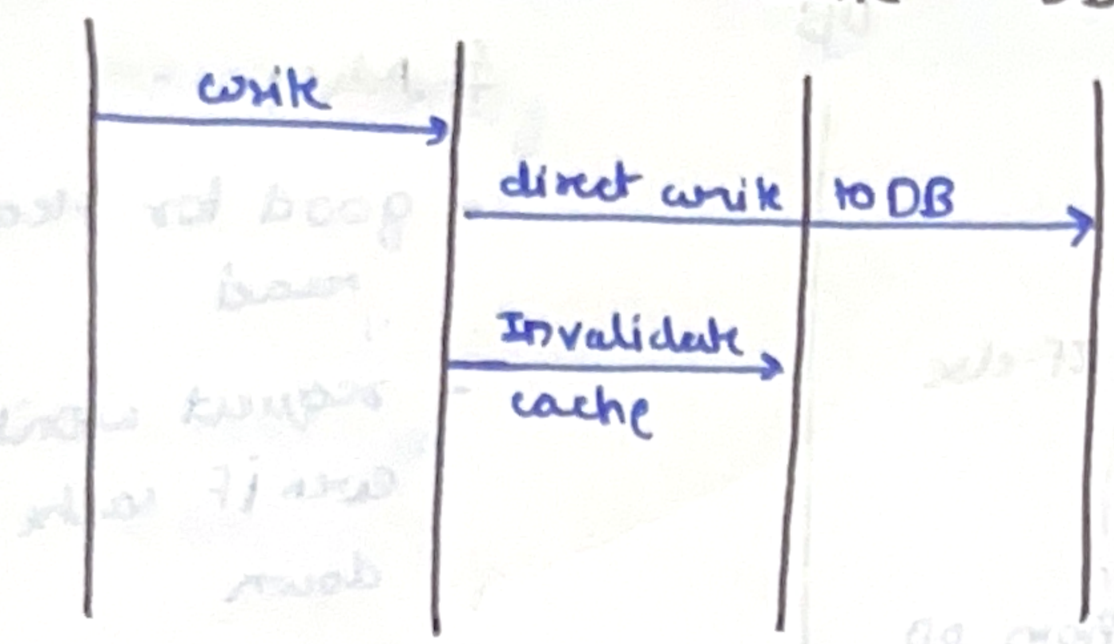
PROs :

- good for heavy read
- Logic of updating cache is seperated from APP.

CONs :

- Cache miss for new data
- inconsistency in Put.
- Cache document should be same as DB.

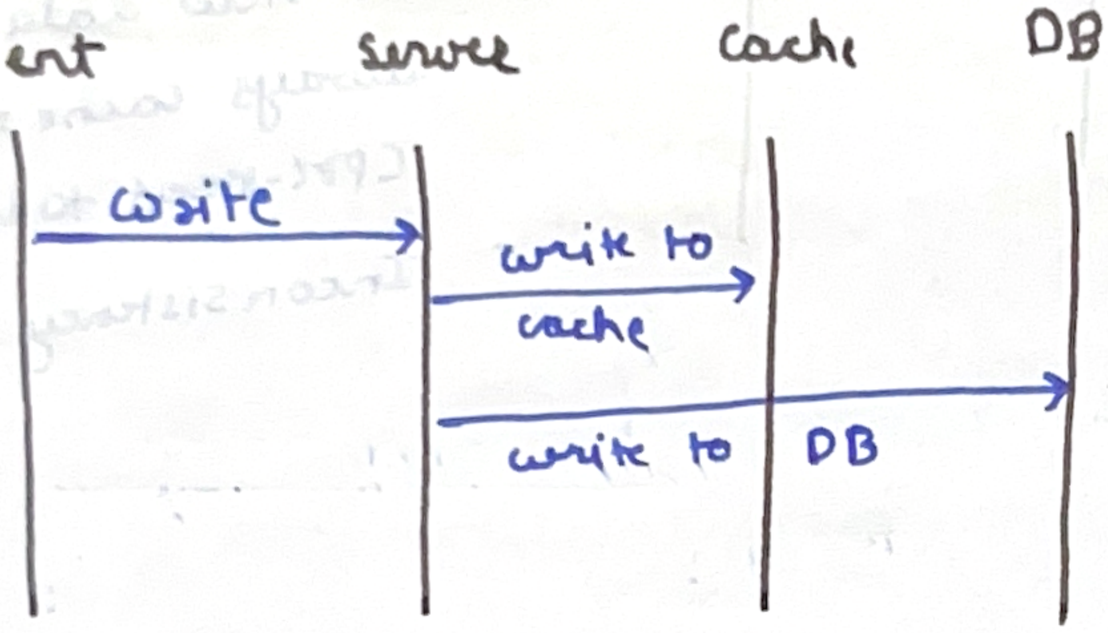
write Around Cache:



PROs:
- Good for heavy read
- resolves inconsistency

CONs:
- cache - miss for new data (pre-heat to solve)
- DB is down, then ops will fail

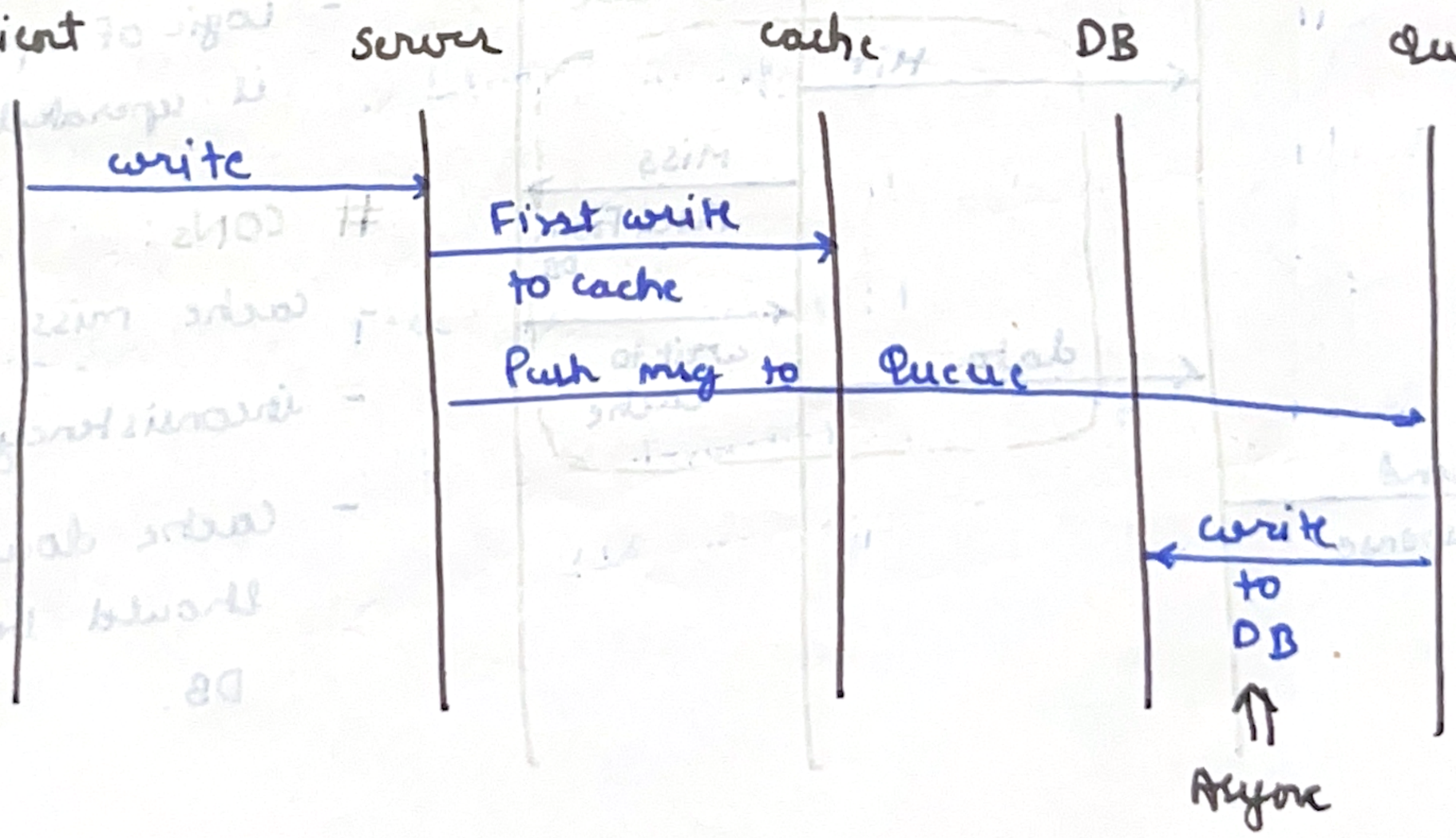
write through cache:



PROs:
- always consistent
- cache Hit ↑
- when used with Read Algos like Read Th or Cache aside), Latency ↑

CONs:
- needs 2Phase commit
- OPs fail, if DB down

write Back (or Behind) cache:



PROs:

- Good for write Heavy Apps
- Improves write latency, as write to DB is Async
- cache Hit ↑
- gives much better perf. when used with read through cache
- even if DB fails write OPs still works

CONs:

- If data is removed from cache and DB is not updated yet then there is chance to temporary missing data. (solve by using high TTL)