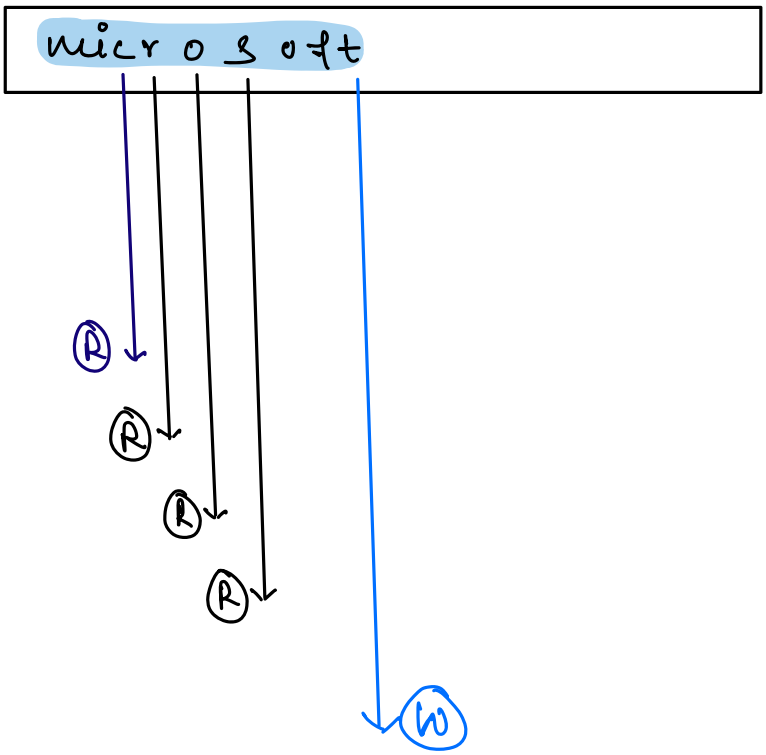
4 Step process for HLD interview :-

1) MVP.
   - Core functionalities.

2) Scale Estimation | Back of the Envelope calculations
   - QPS
     - Write
     - Read
   - Storage requirements.
     - Is Sharding required ?

3) Design Trade Offs.
   - Consistency (vs) Availability.
   - Consistency (vs) Latency.

4) Design Deep dive.
   - API's
   - System diagram + Data flow.



Writes qps = 100K

Read qps = 500K

→ Both Read & Write Heavy.

Consistency (VS) Availability.

=> AP System + Eventual Consistency

=> Super low latency.

APIs

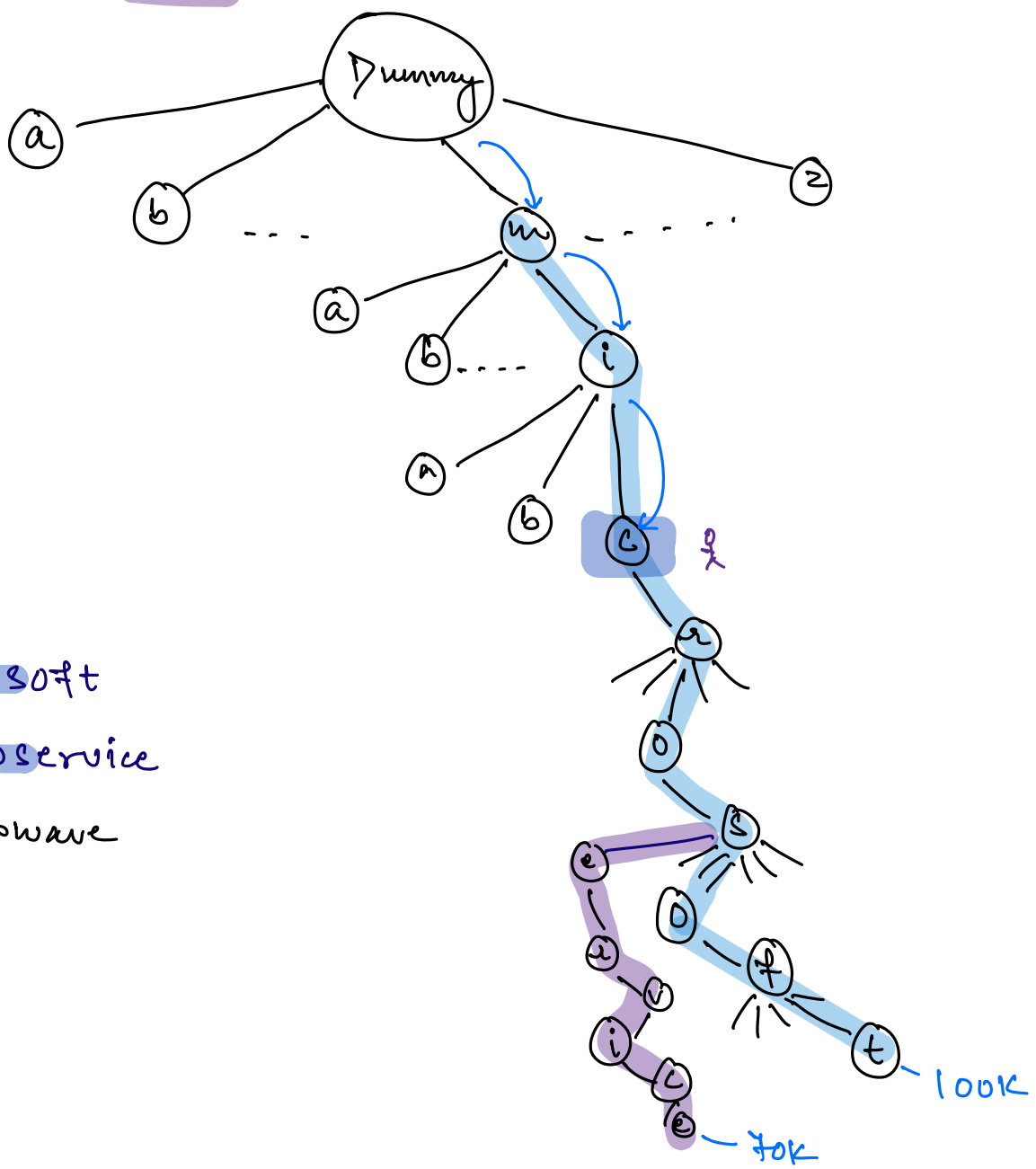
→ getSuggestions(query_prefix, limit = 5)

→ search(search_query)

↳ POST

Prefix Based searching.

↳ Trie.



microsoft

microservice

microwave

⇒ Get the suggestions for the prefix "mic"

Highly unoptimized.

↳ To get the suggestions for a particular prefix, We'll have to recursively find out all the suggestions in subtree for the given prefix.

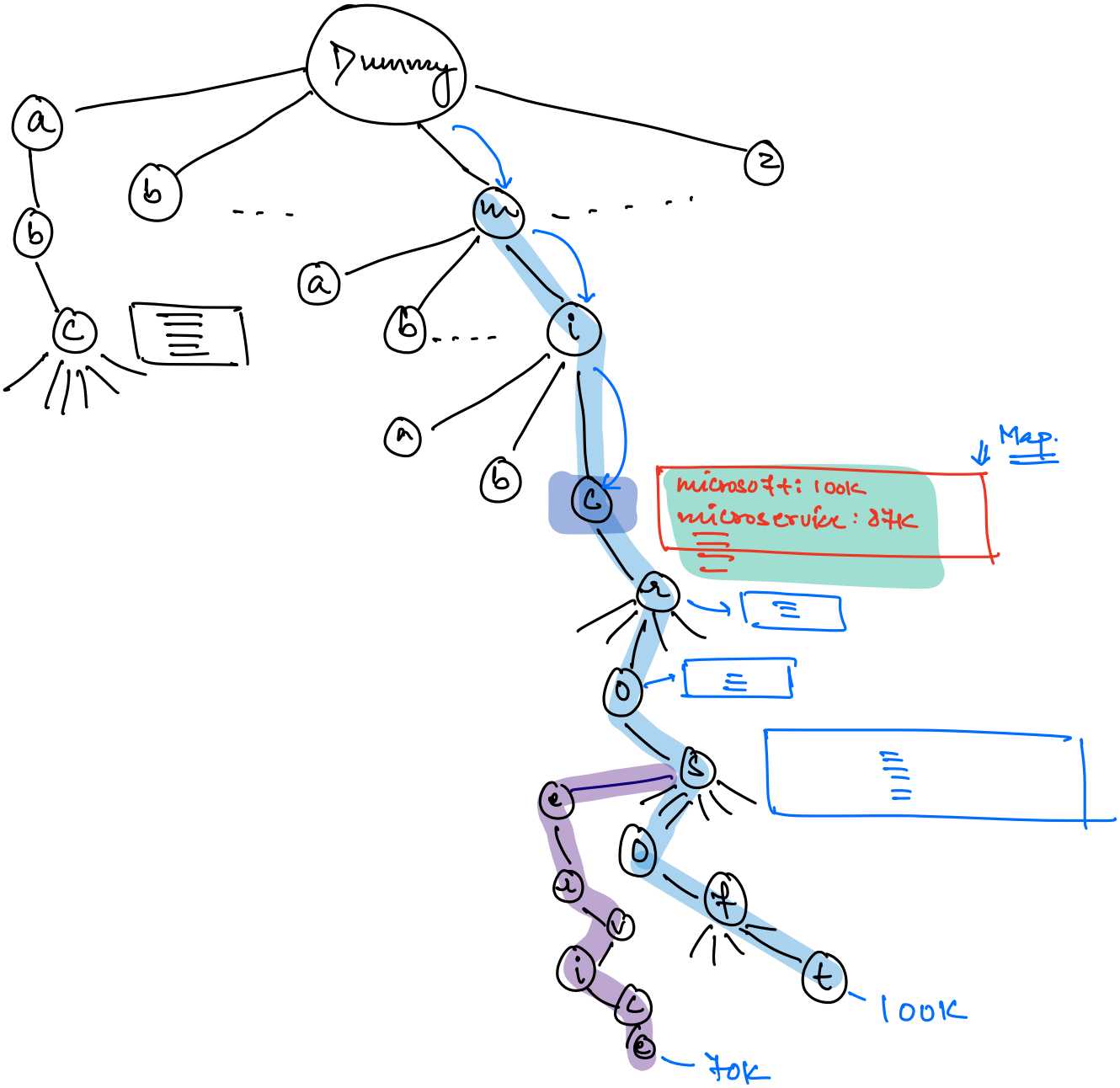
Can this entire Trie be stored in a single DB m/c ?

↳ NO.

↳ We need to SHARD this trie across multiple m/c.

⇒ Sharding a trie is NOT supported by default by any DB, so everything related to sharding, We'll have to do manually.

⇒ So, Simple trie isn't a feasible soln.



⇒ At each node, we can maintain a data structure with the top 5-10 most frequent word with that prefix.

⇒ SHARDING is still an issue.

HashMaps Based Solution.

→ HashMap #1 : frequency map.

query : frequency.

Scaler : 7K

microsoft : 150K ...

→ HashMap #2 : Suggestions map.

HashMap to store top 5 suggestions for a prefix.

"mic" : [microsoft : 100K
microscope : 85K
—— : 78K
—— : 60K
—— : 50K]

⇒ K:V DB.

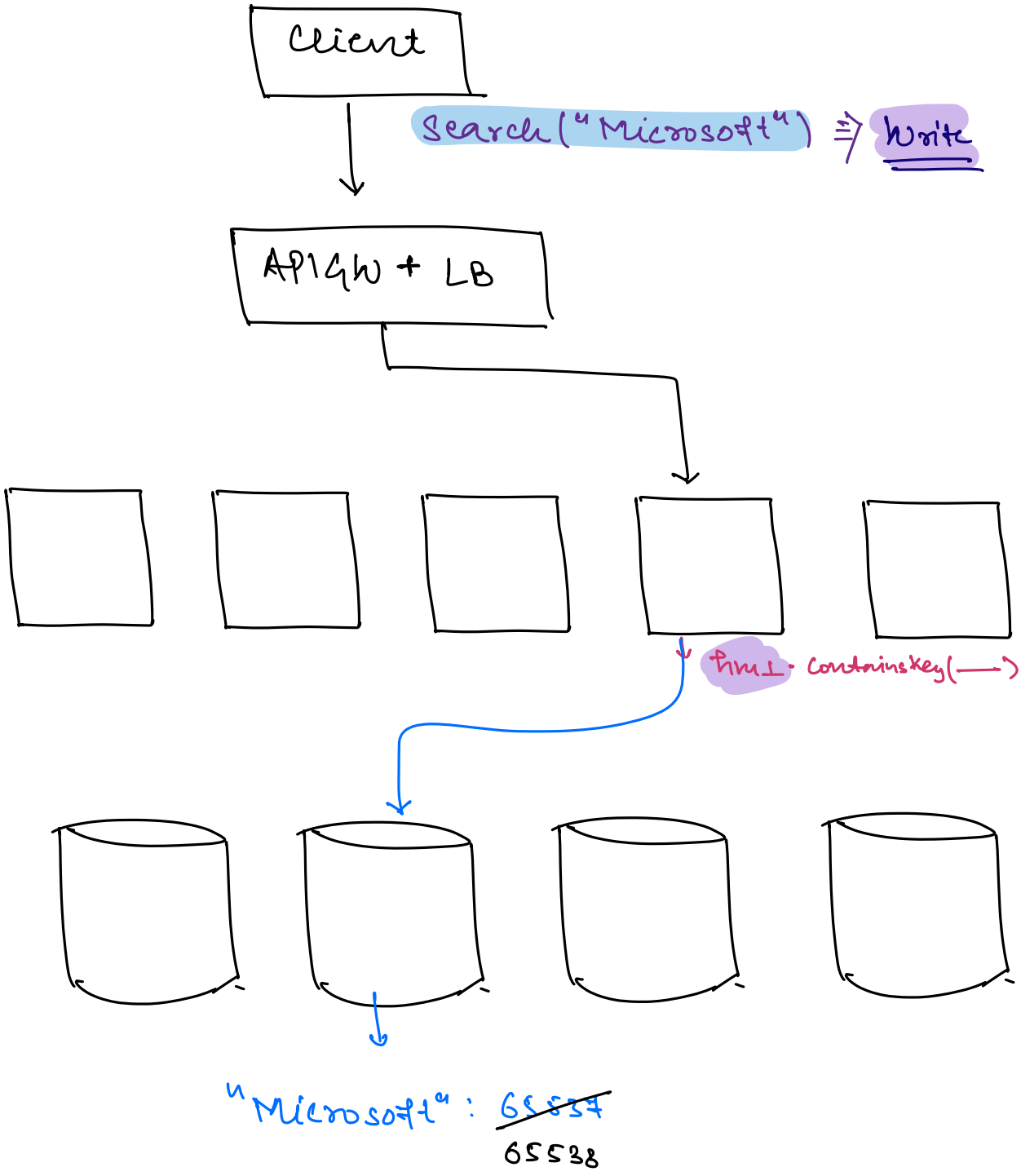
→ NoSQL

→ REDIS.

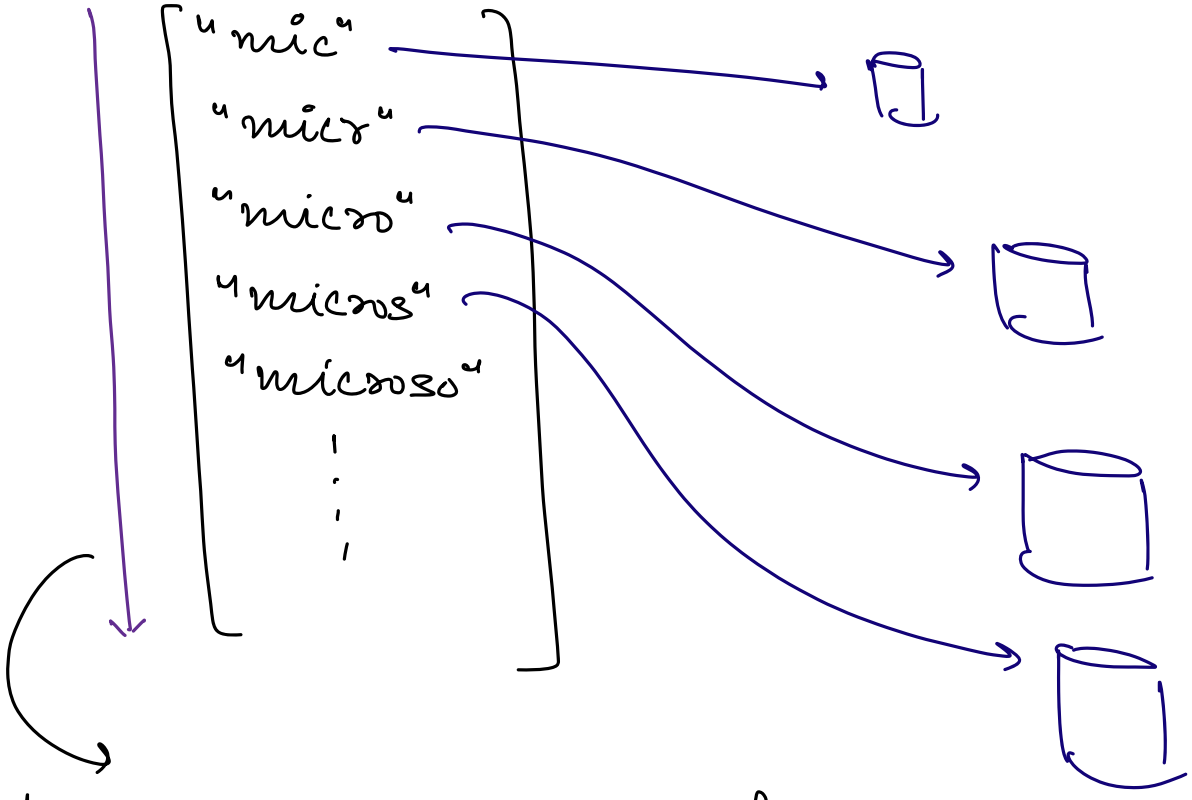
↳ Optional Support for Disk persistence.

→ Supports Sharding automatically based on the key.

→ Distributed K-V DB.



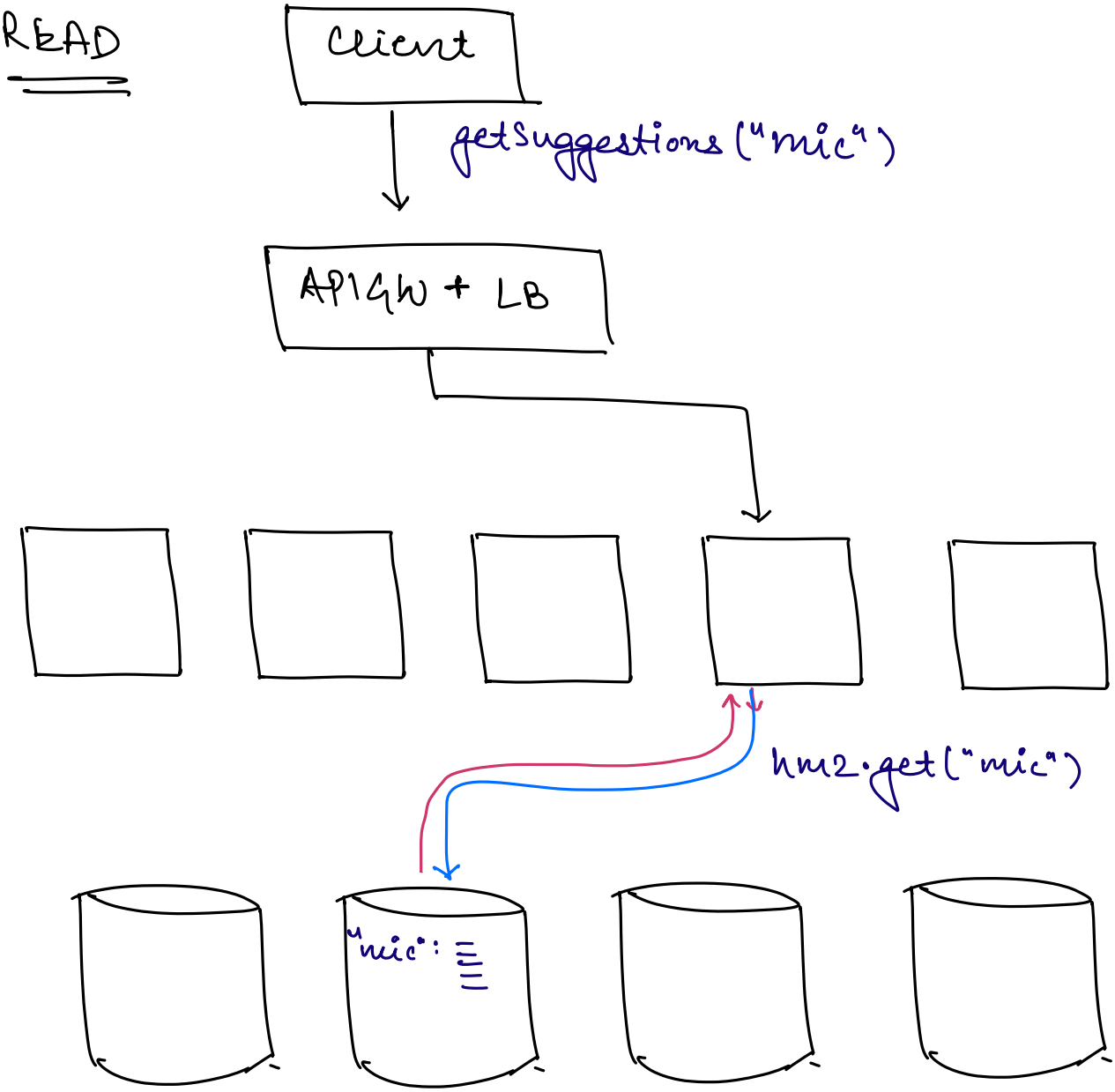
HM1 & HM2 will be distributed across multiple DB m/c.



We need to change the frequency of "microsoft" in all the prefixes (HM 1).

⇒ Inter SHARD Query.

READ



⇒ Intra SHARD Query.

↳ READ queries have low latency.

⇒ To further optimize read queries we can store the most frequently accessed prefixes in the Cache to avoid DB call.

Write Queries / Day ~ 10B

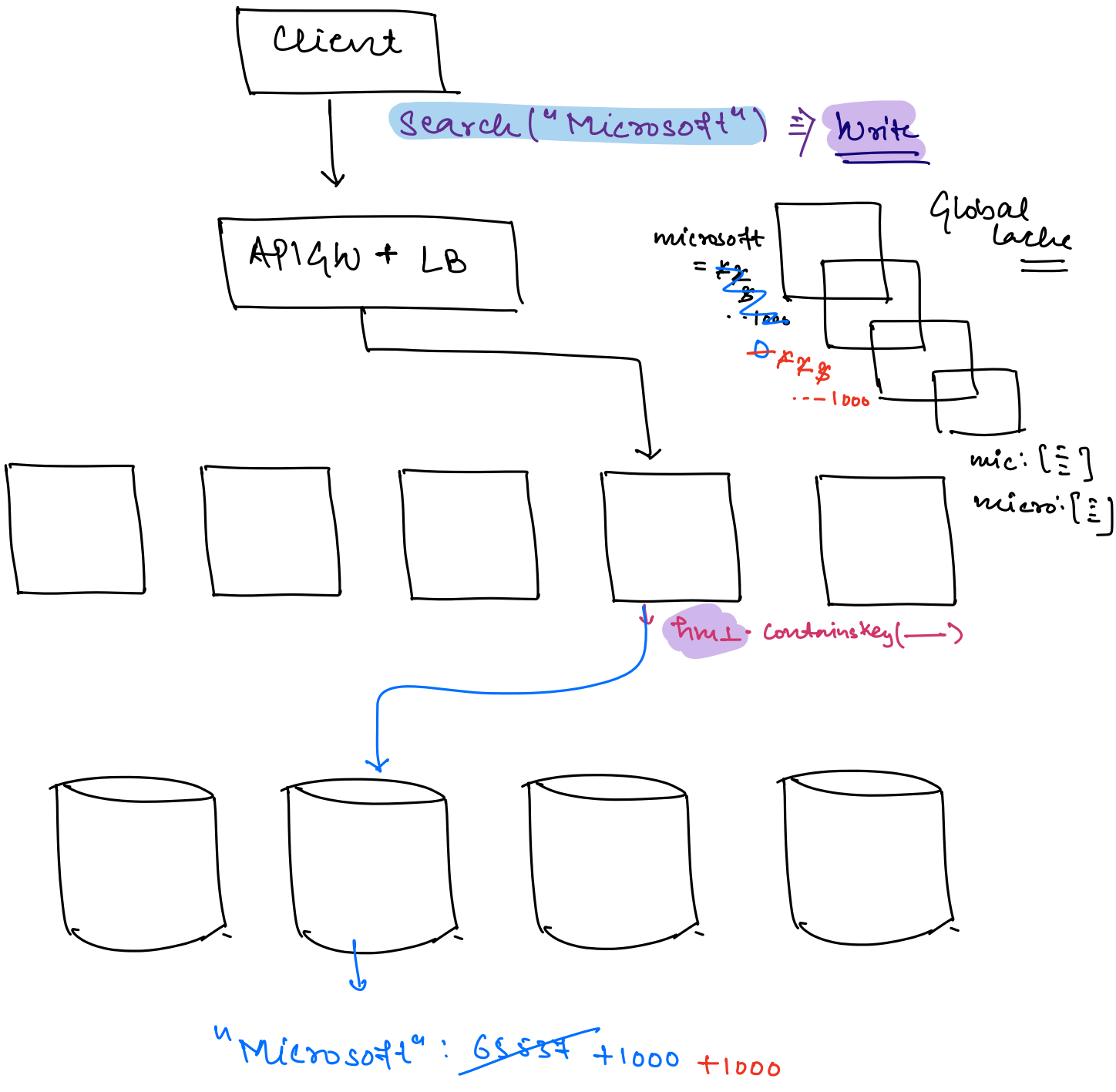
Read Queries / Day ~ 50B

→ Typeahead is a R+W heavy system, we should try to optimize both.

⇒ To optimize the write operation, instead of updating the frequency for the search query for every search, we can increment asynchronously.

↓

Batched Writes.



⇒ Writes | Day = 10 B.

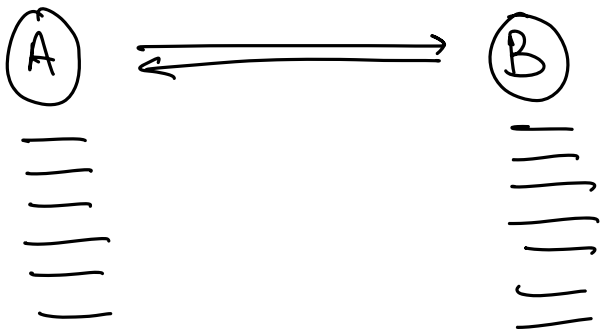
⇒ Batch updates at a threshold of 1000

$$\Rightarrow \frac{10 \times 10^9}{10^3} = 10 \times 10^6$$

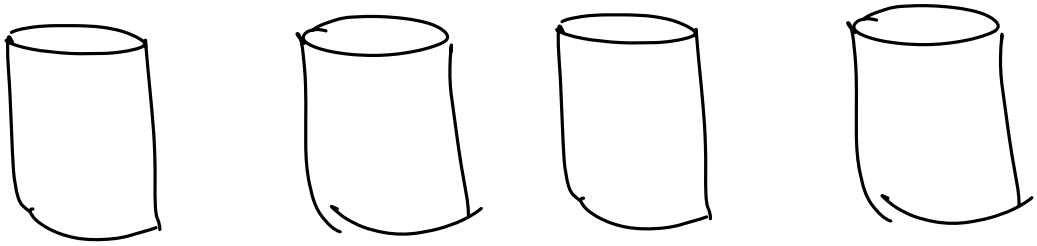
$$= 10M.$$

⇒ FB Messenger App^n

1 : 1 Chats



~~msg_id.~~



user_id.