8 | Dec | 2023

# Search Type Ahead

① Features / Minimum Viable Product (MVP)

② Estimation of Scale

→ Sharding is a necessity OR NOT

→ Read heavy OR Write heavy
OR Read + Write heavy

③ Design TradeOffs + Design Goals

① CAP

② Horizontal Scaling vs Vertical

③ App Servers — stateful OR stateless

④ Latency
→ High
→ low
→ Super low

⑤ Caching

⑥ SQL vs NoSQL

# ④ Design Deep dive

① APIs

② Components → LB, AppServer, Global Cache, Database layer, Blob, Archival Storage

③ Data Flow

## Search Typeahead Suggestions

Google, Bing, Amazon, Hotstar

### ① Minimum Viable Product (MVP)

ⓐ Prefix of a query

← suggestions #5

ⓑ at every key stroke, after first 3 chars.

ⓒ Suggestions

— should be based on popularity

## ② Estimation of Scale

google scale

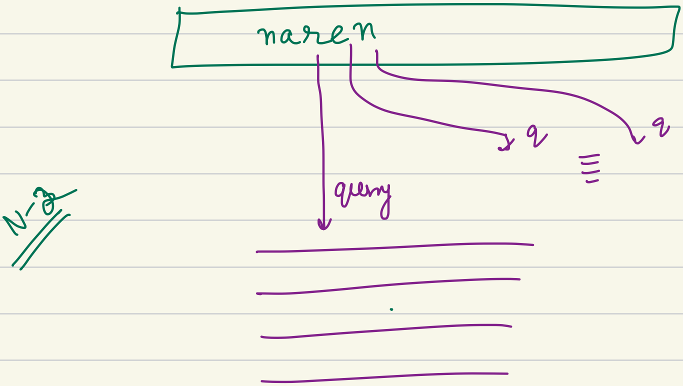
# Users — 4 B

# DAU — 1 B

10 queries / day

# Queries / day = 10 Billion ☺

# search typeahead queries / search query = 5

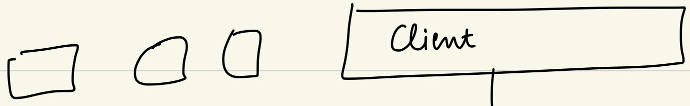


# typeahead completion queries = 10 B X 5

= 50B / day

☺

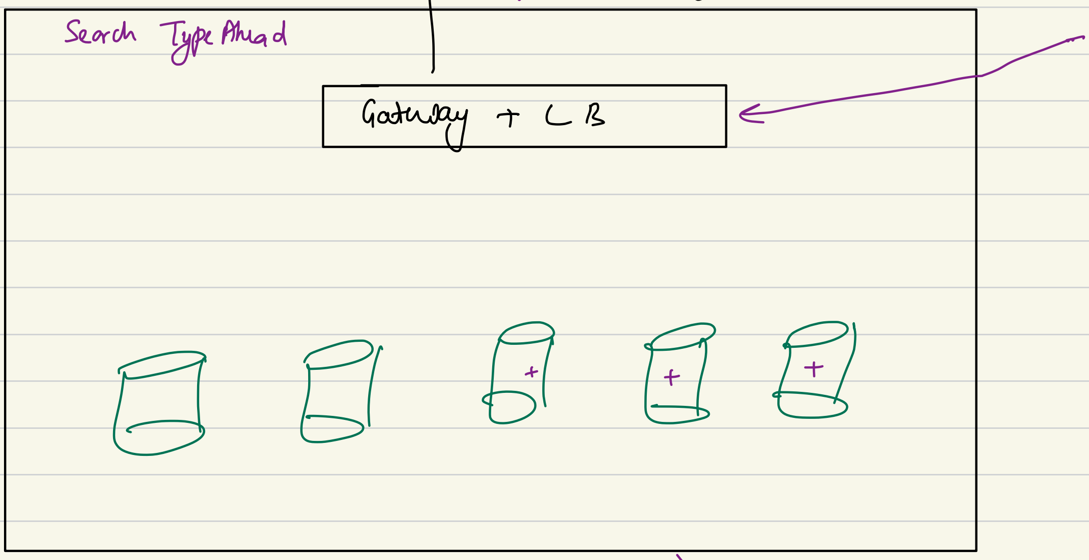
1 Billion



10 search queries ≡ 10 B searches

5 times on avg we need search typeahead

50 Billion /day



✓ Read queries/day = 50 Billion

✓ Write query / day = 10 Billion

Ex 100B R
10B W
☆ R+W

100 B R
100M W
☆ Read

☆ Read + Write heavy system

# Concurrency

QPS ≡ queries per second

# Write Queries/ day = 10 B

$$\frac{10\ B}{24 \times 60 \times 60} \rightarrow 86400$$

$$\frac{\boxed{10B}}{86400} \quad \text{QPS}$$

$$= \frac{1\cancel{0} \times 10^{8}}{86\ 4\cancel{00}}$$

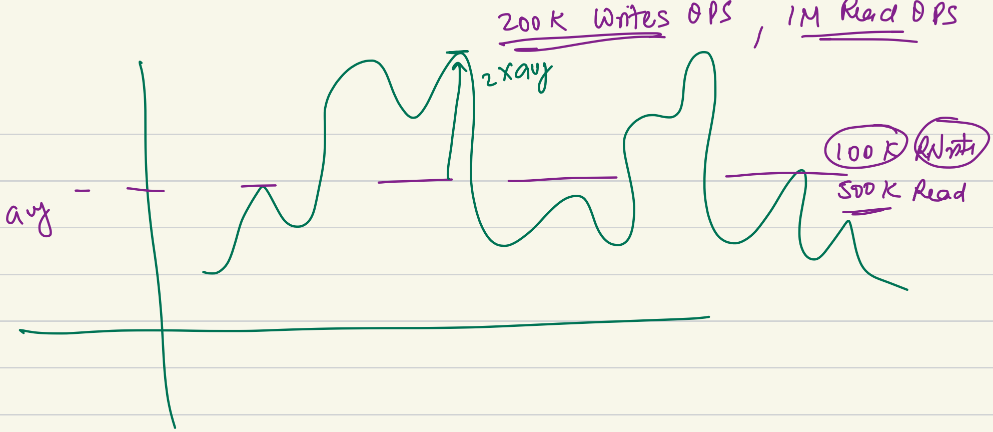
$$= \frac{\overset{1.1}{\cancel{1000}} \times 10^{5}}{\cancel{864}}$$

$$= 1.1 \times 10^{5}\ \text{QPS}$$

Avg

100K Write QPS 😮 😮 😮

500K Read QPS 😮 😮 😮



How much storage will be required?

10 Billion Write Requests / day

10% queries ≡ new

1 Billion new searches / day ☺

Search query 30 chars = 30 Bytes

Count = 10 Bytes

Metadata 60 Bytes

100 Bytes / new search

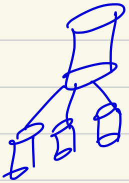
1 B × 100 bytes = 100 GB/day

10 years

100GB × 365 × 10

= ~~1000GB~~ TB × 365

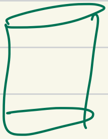
= 365 TB data ☺



1 Master 3 slaves Replication

365 TB × 4 = 1500TB

4TB–10TB



☆ sharding is a necessity ☺

# Design TradeOffs

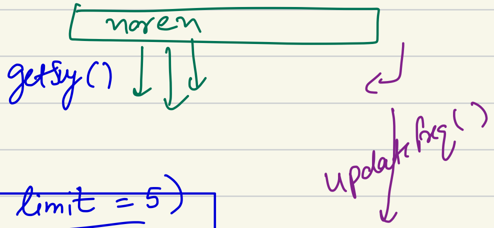
① Availability >> Consistency

AP ~~CP~~

② [ Latency — super low ]

( This system competes with the typing speed of users and hence the latency should be super low ]

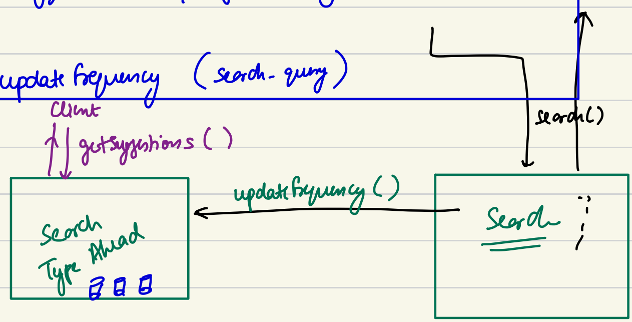
## ④ Design deep dive

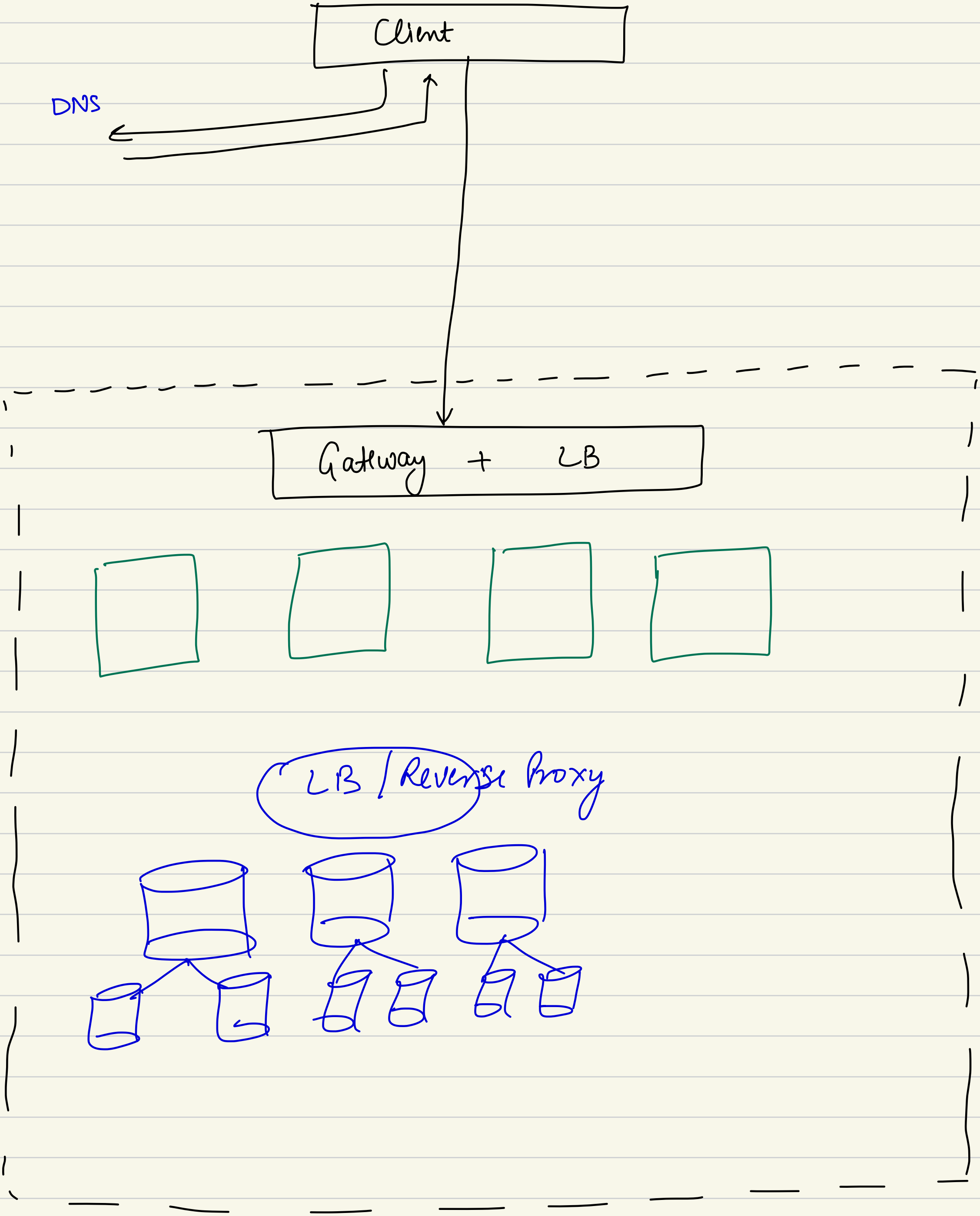


① APIs

ⓐ getSuggestions ( prefix_string , limit = 5)

ⓑ updateFrequency (search_query)

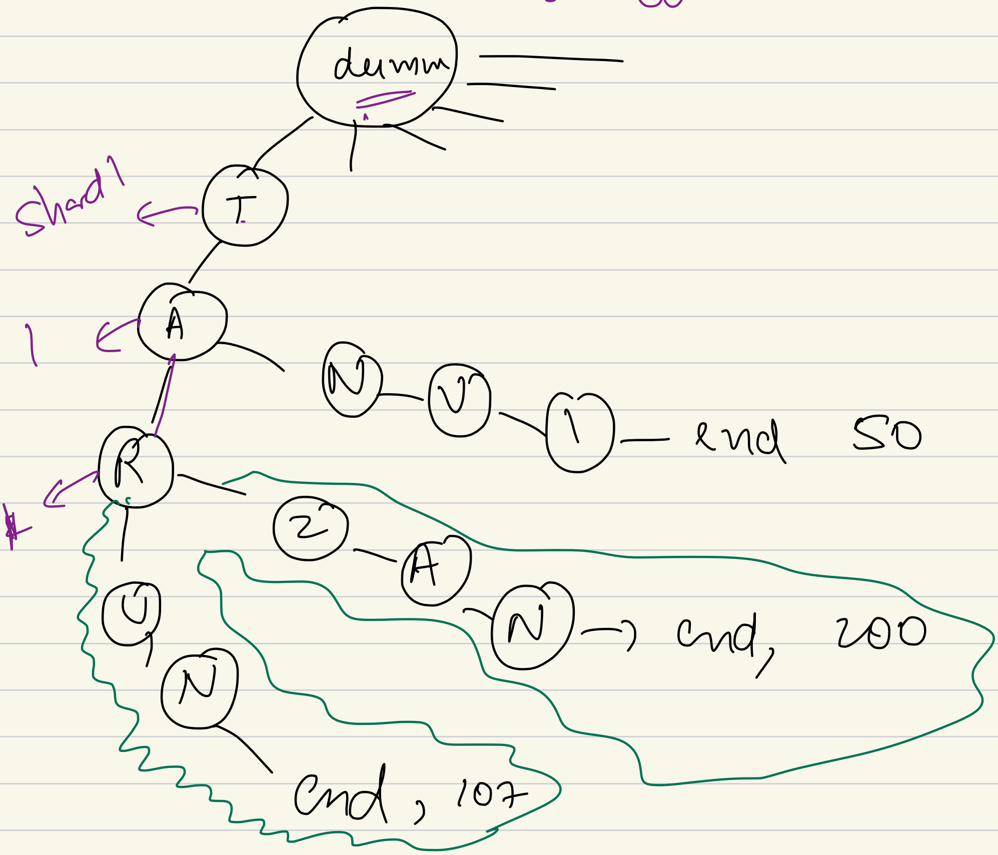



Client
DNS
Gateway + LB
LB / Reverse Proxy

# Trie

# Hash Map

getSuggestions ("TAR")



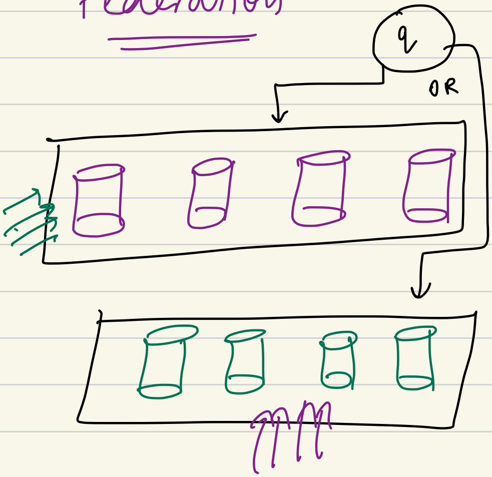
① sharding is a challenge

② query on Trie might lead to multiple shard query which is also bad

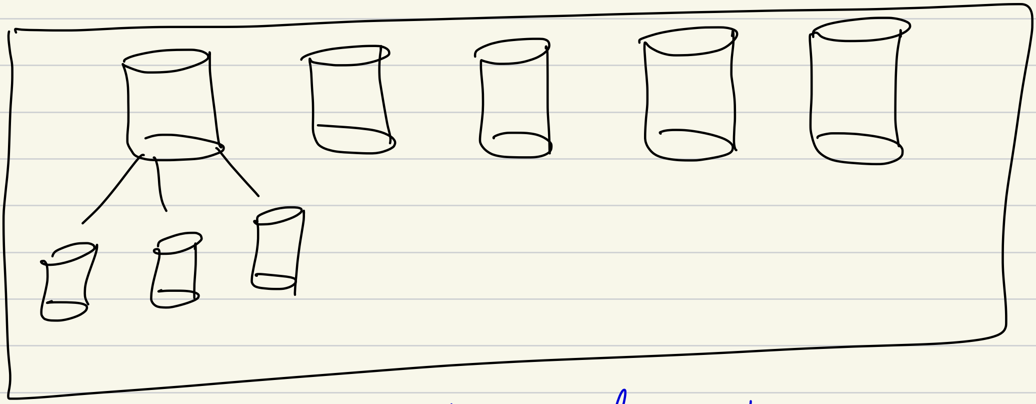
③ Backtracking ☹ ☹ → Backtracking across shards ☹ ☹

# Hashmap Approach

## Federation



✓ DB 1 ≡ cluster | distributed databases ☺



Key-value store
ex. Redis

| Search Query | Frequency |
|---|---|
| Tarun Malhotra | 700 |
| Michelle Obama | 900 |
| Michael Clark | 400 |
| Michael Jackson | 600 |

DB2 → distributed / cluster

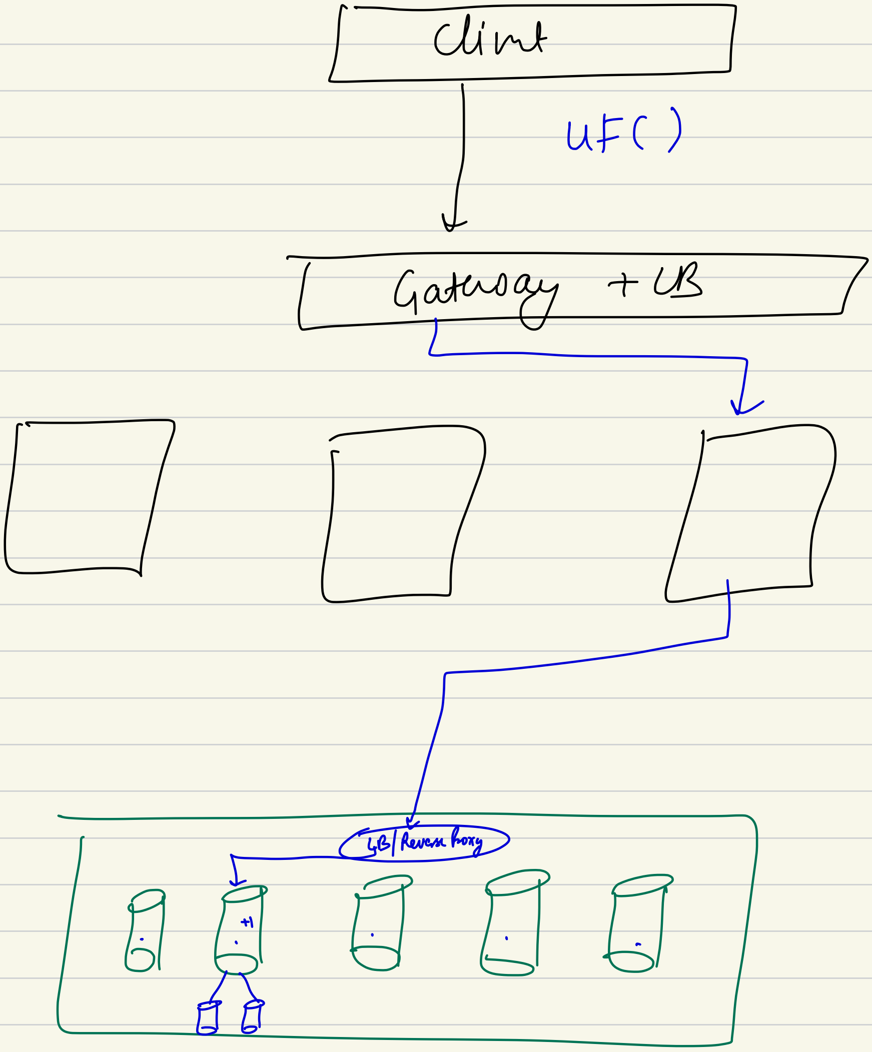
(key-value store)

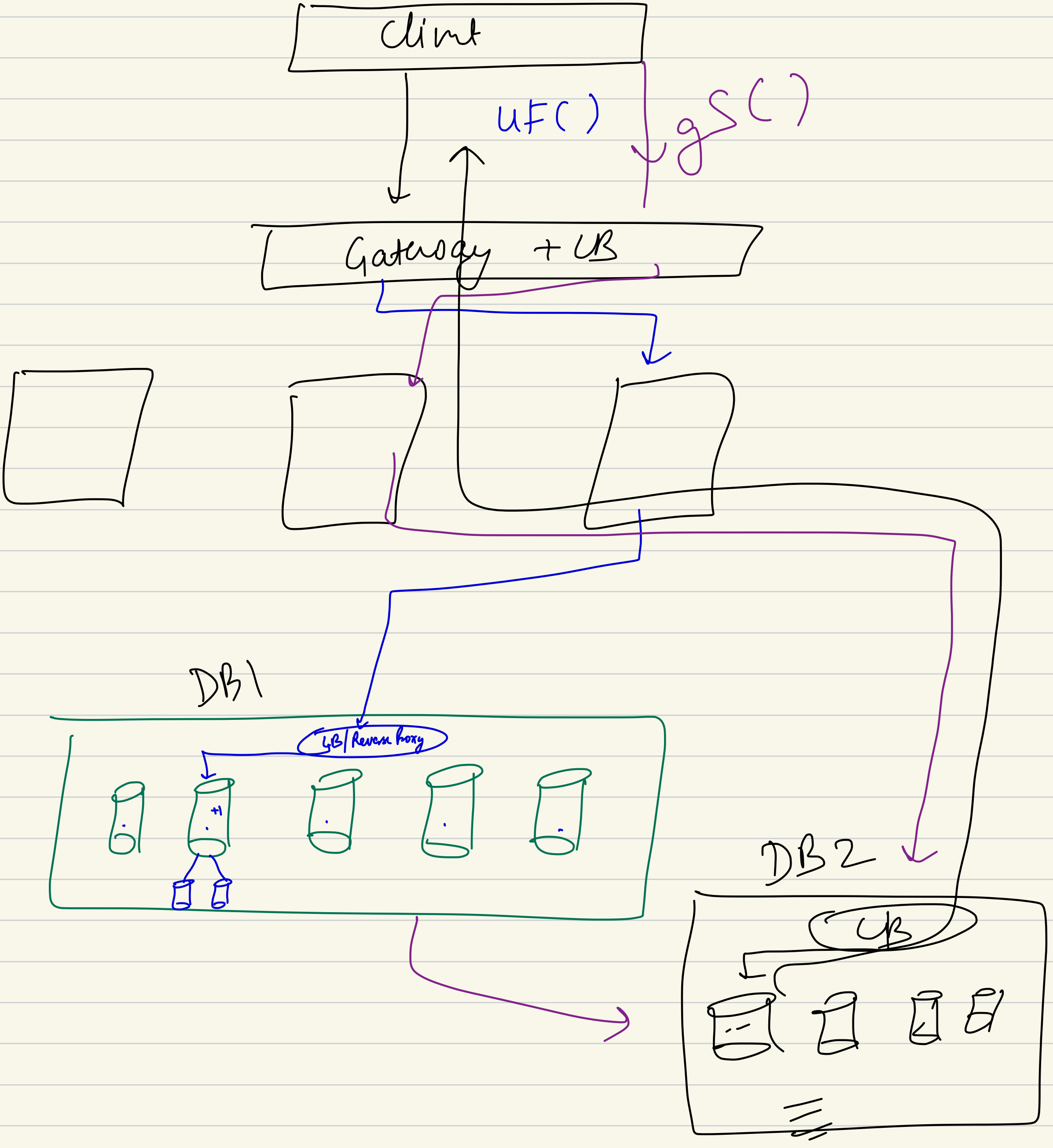
shardingkey = Prefix

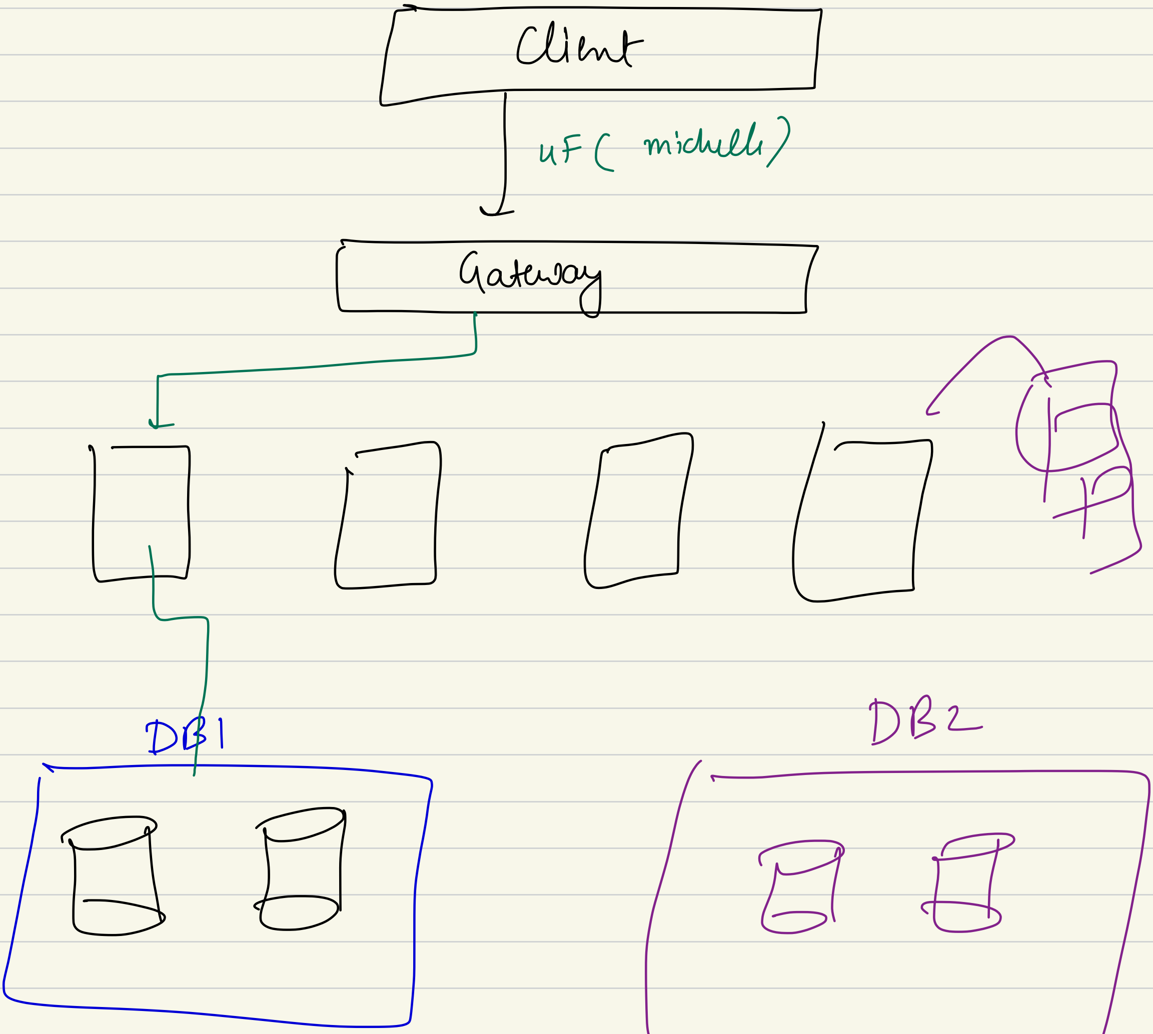
| Prefix | Top 5 Suggestions |
|---|---|
| tar | [ Tarun , Tarun Malhotra , - - - - - - ]<br>F1 F2 |
| Tarru | [ ] |
| mich | [ ] |
| micha | [ ] |
| mic | [ ] |

shard key = search Term

update frequency ( search_query )




Clint
uF( )
gS( )
Gateway + LB
DB1
LB/Reverse Proxy
+1
DB2
LB



| Search Query | Frequency |
|---|---|
| Michelle | 700 +1 |
| Michael Jackson | 900 |
| Microsoft | 1100 |

Problem

| Prefix | Top 5 Suggestions |
|---|---|
| Mic | [Michelle, Microsoft, Mickey mouse] |
| Tar | |
| Mich | [ ] |
| Micha | |

Miche [ ]

☹ ☹ ☹ ☹

# Read + Write heavy system

We somehow need to absorb high writes and then the above system will become managable.

① Messaging Queue → 😛 😛 😛 😛

② Threshold Approach ☺ ☺

DB1 freq

+ 10,000 increase ~

OR

+1,00,000 increase ✓

OR

+ 10L increase ✓

DB1 $\xrightarrow{\text{+update}}$ DB2

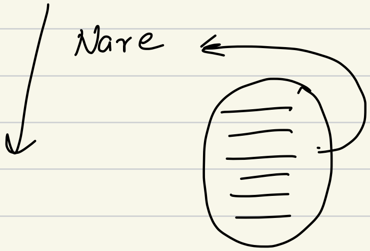
☺ ☺

# Recency Factor :) ☺

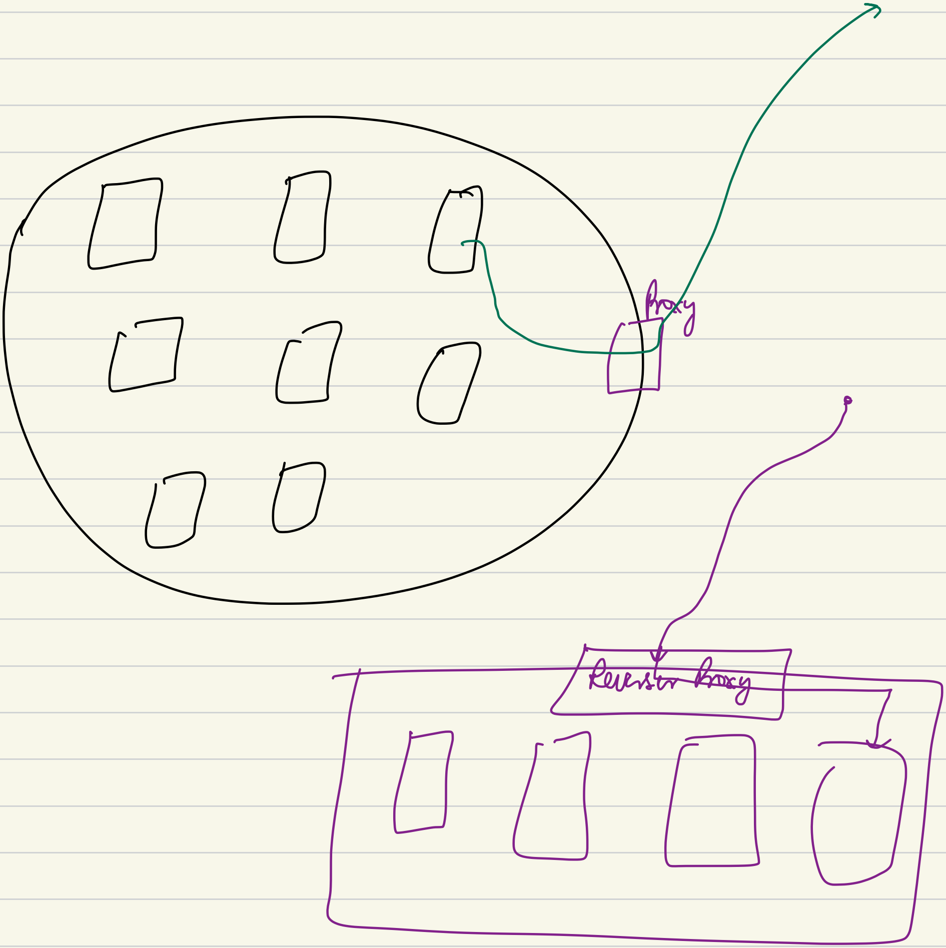
Case Study

1. MVP →
2. Estimation of Scale
3. Design TradeOffs →
4. Design deep dive
    - API
    - Components ✓ [Architectural diagram]
    - Request flow ✓

# Question : Tech-stack



Client Side Caching



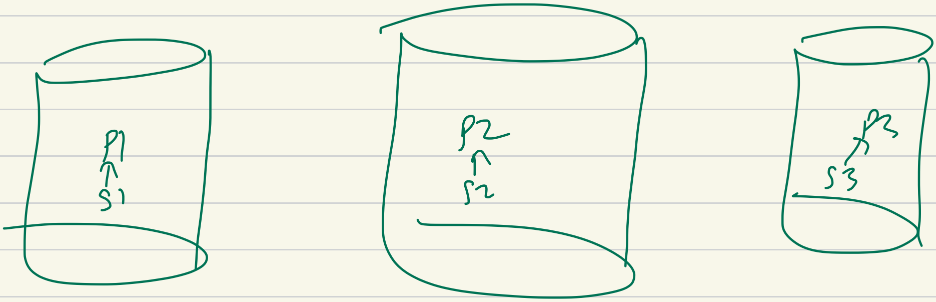
① MVP → 4–5 mins → 7 mins

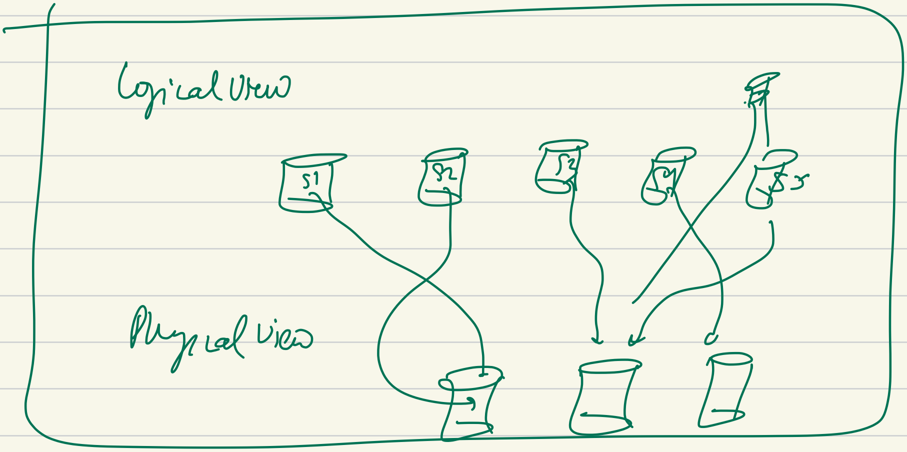
② Estimation of scale → 3 mins

③ Design TradeOffs → 3–4 mins

Try to save even most time

④ Design Deepdive → 30–45 mins ☺


P1
S1
P2
S2
P3
S3


Logical View
S1
S2
S3
S4
S5
Physical View