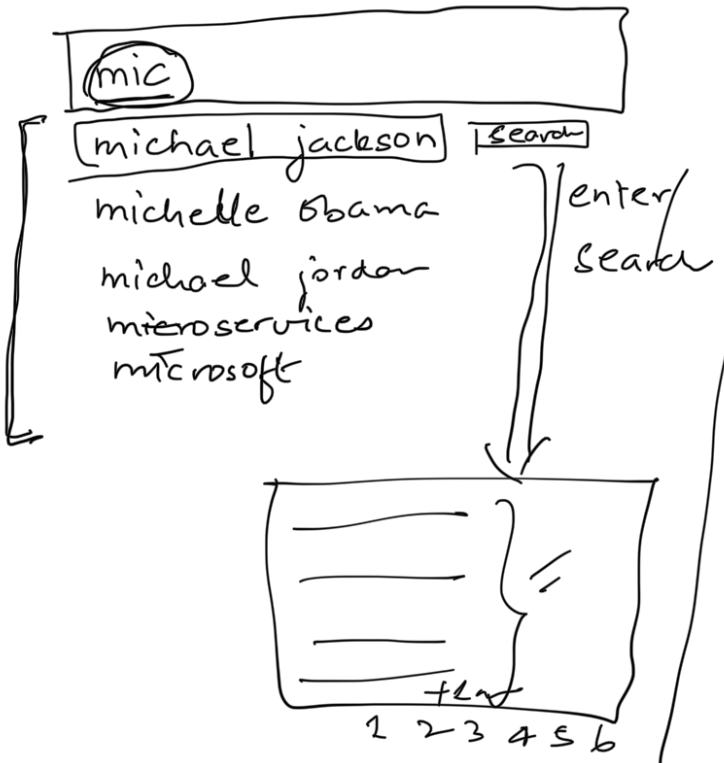
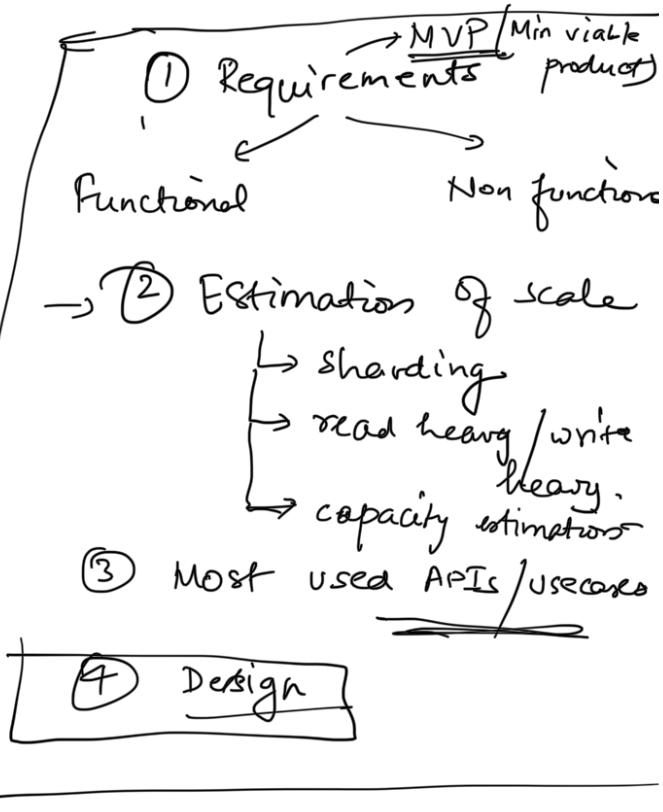
# DESIGN GOOGLE TYPEAHEAD




① MVP

① how many suggestions (max) → 5 suggestions.

② ordering / choosing → most popular 5 (v1)
recency
# of times it is searched

③ personalise (location, system, person based)

⇒ ③-5 suggestions ~~should have~~ typed text as strict prefix → NOT NEEDED IN V0

④ 3 letters before showing suggestions.

mic · —

⑤ spelling mistakes → NOT NEEDED FOR V0 — mci

⑥ case sensitivity → case insensitive.

---

Non-functional requirement — Consistency →

① Highly available — mic ji

② Latency → very low latency

↳ 5800
↳ 5700

## Estimation of scale

size of data ⇐ ① sharding → YES

Both ⇐ ② write vs read

③ capacity → QPS bottleneck

Search term → frequency

# searches → 10% → 8 billion / day.

8 Billion / 10 new search terms / day.

0.8 B * 365 * 10 terms * 100 bytes

365 · 10 years

= 86 * 365 * 100

= ~300 TB

8 Billion queries per day

8 Billion / 10 new queries / day

8 Billion / 10 * 365 * 10 new search terms.

⇒ bytes ⇒ 8 B * 365 * 100 bytes

= 8 GB * 365 * 100 ≃ 300 TB

Read

every autosuggest query

1 read, 1 read, 1 read, 1 read

cache

Write

every search

Much

read

Sachin tendulkar [Enter]

read & write heavy

michael

write

---

① Traffic → queries per second.

9:30 – 11:30

8 Billion searches / day

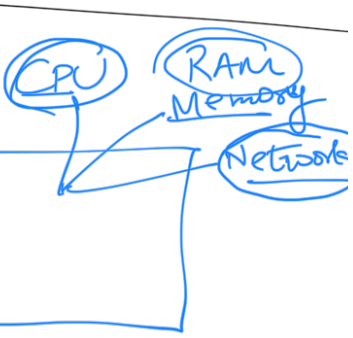
+ → writes.

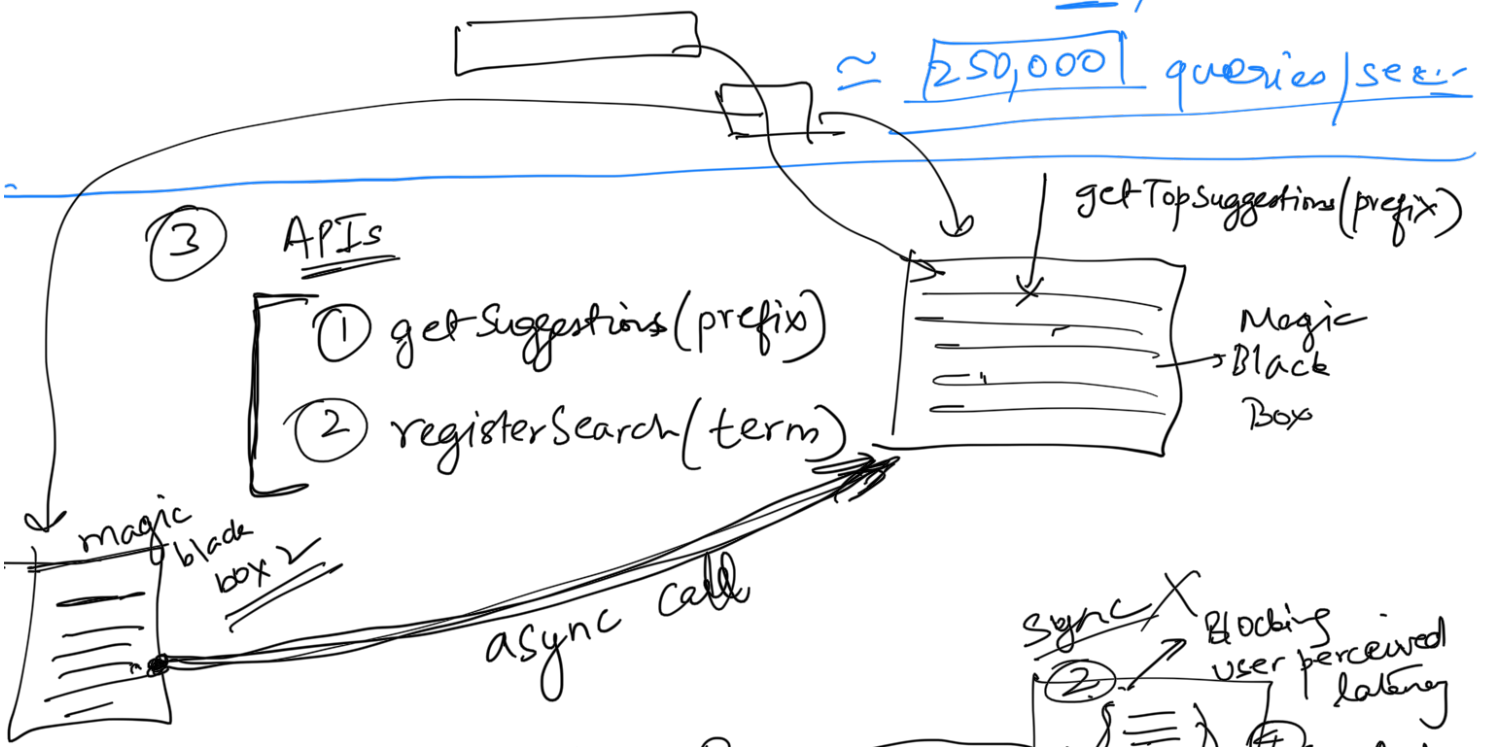
16 Billion reads / day

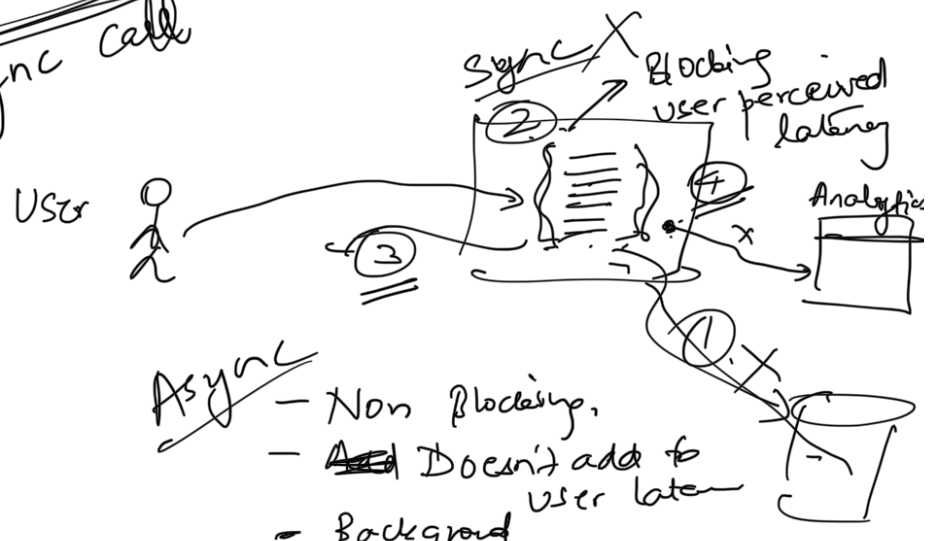
24 Billion = 24 Billion / 24 * 60 * 60 QPS

= 1000 * 1000 * 100 / 36 QPS

≃ 250,000 queries / sec

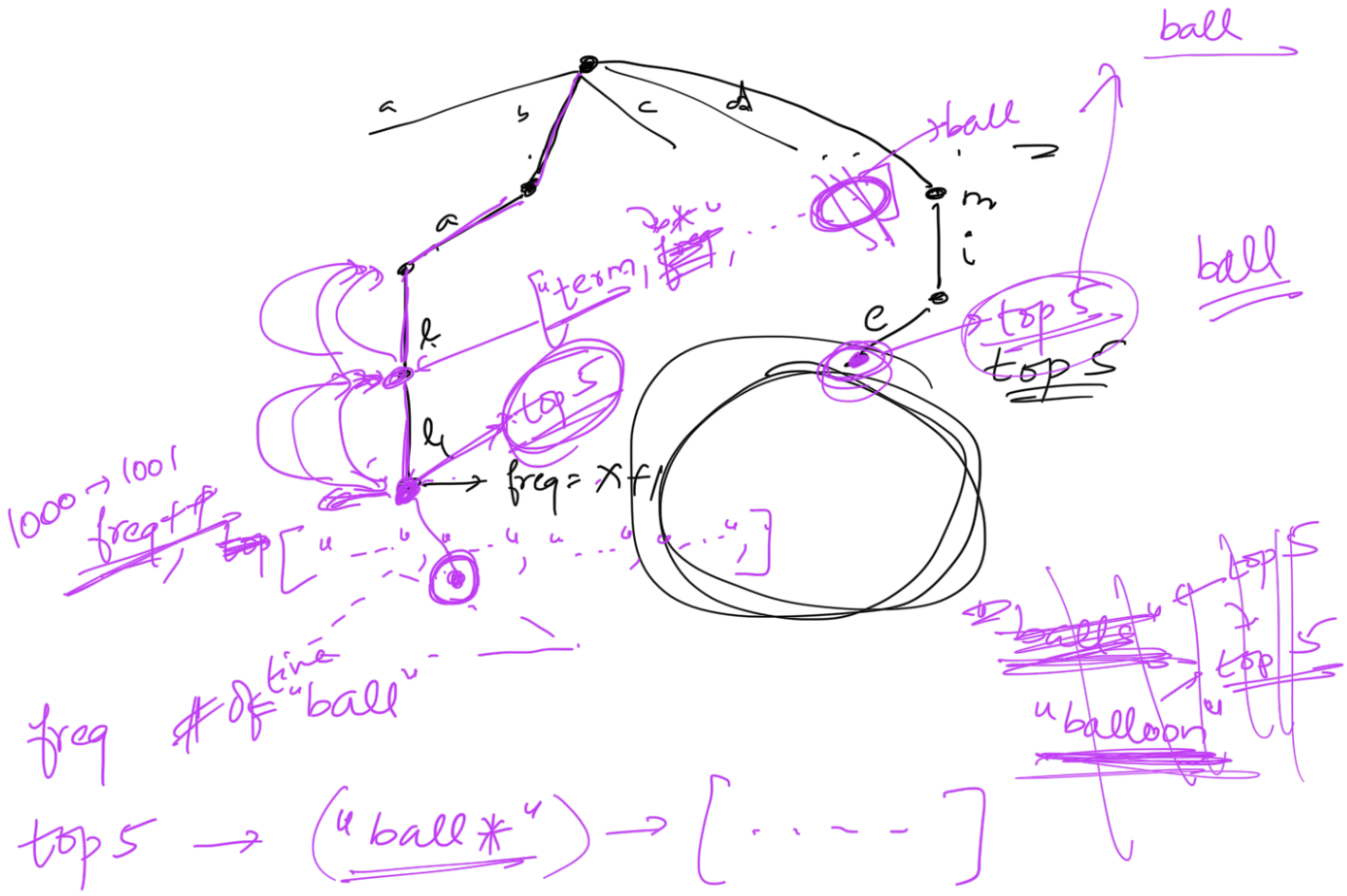


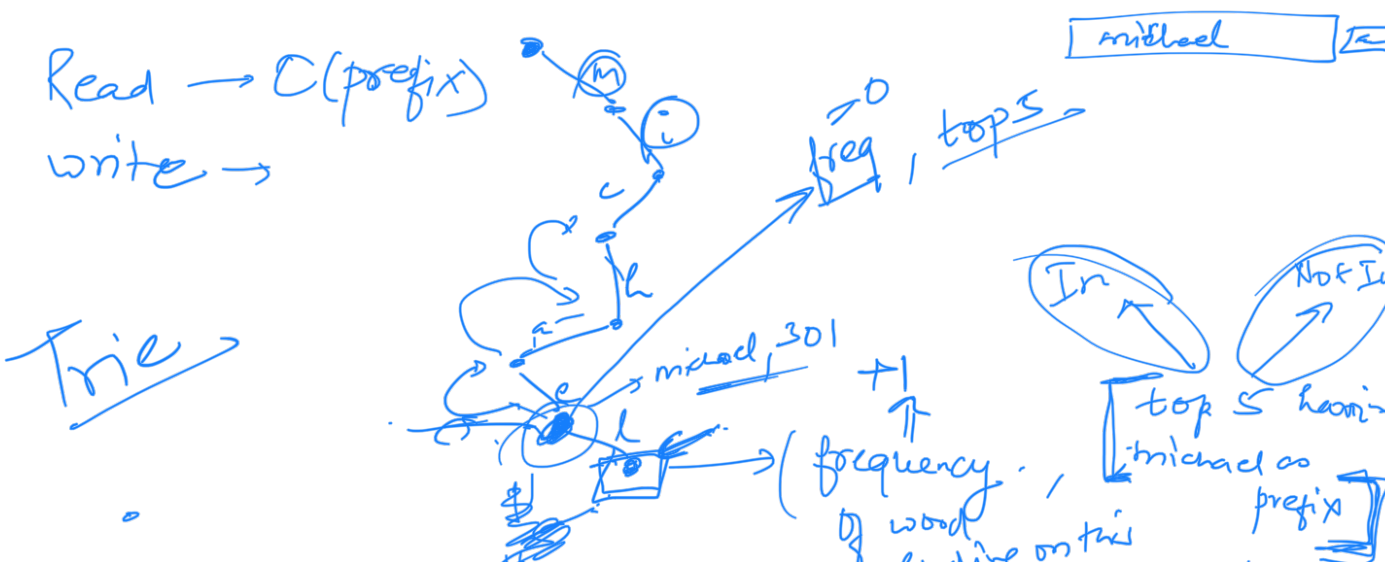



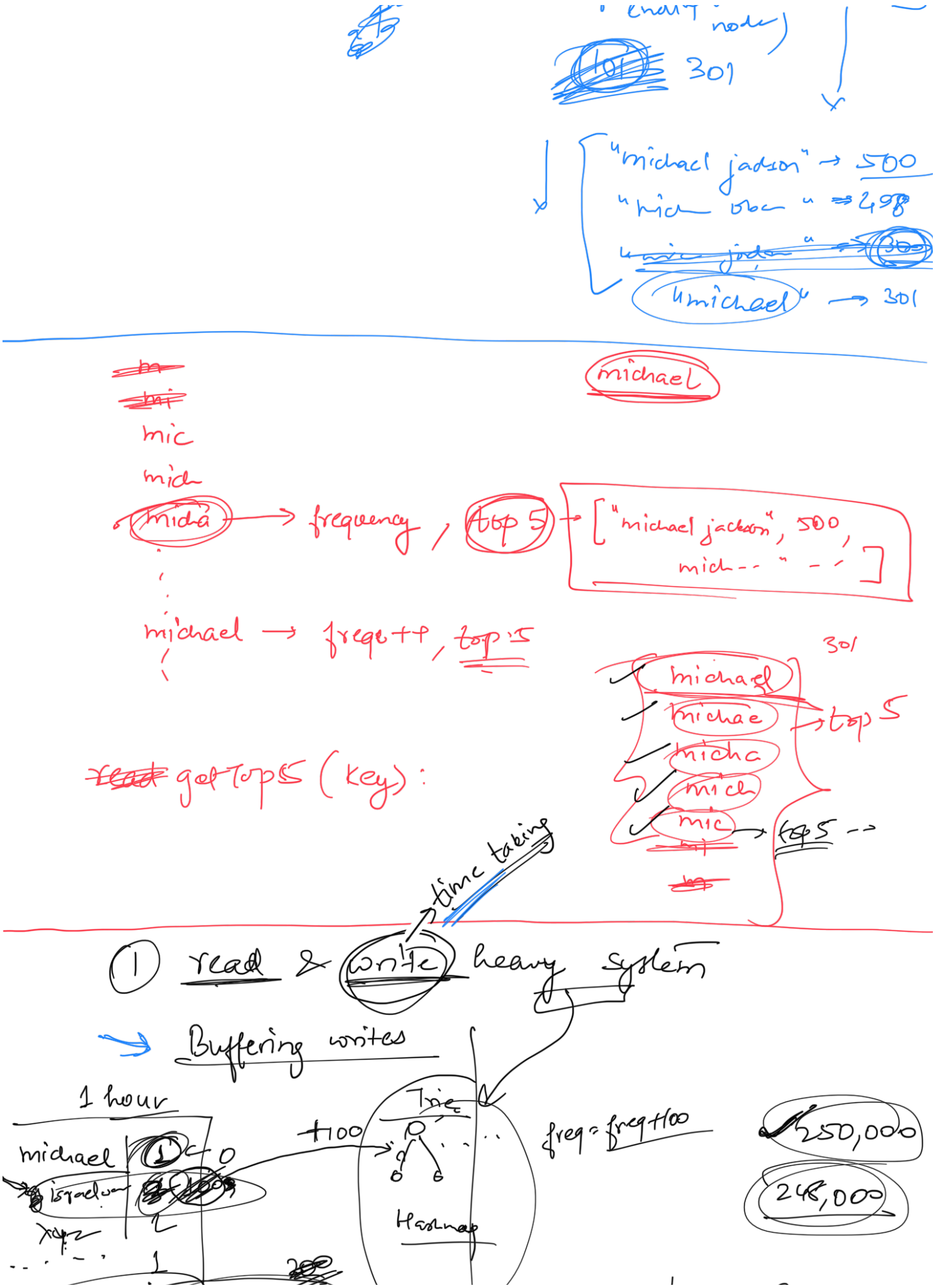



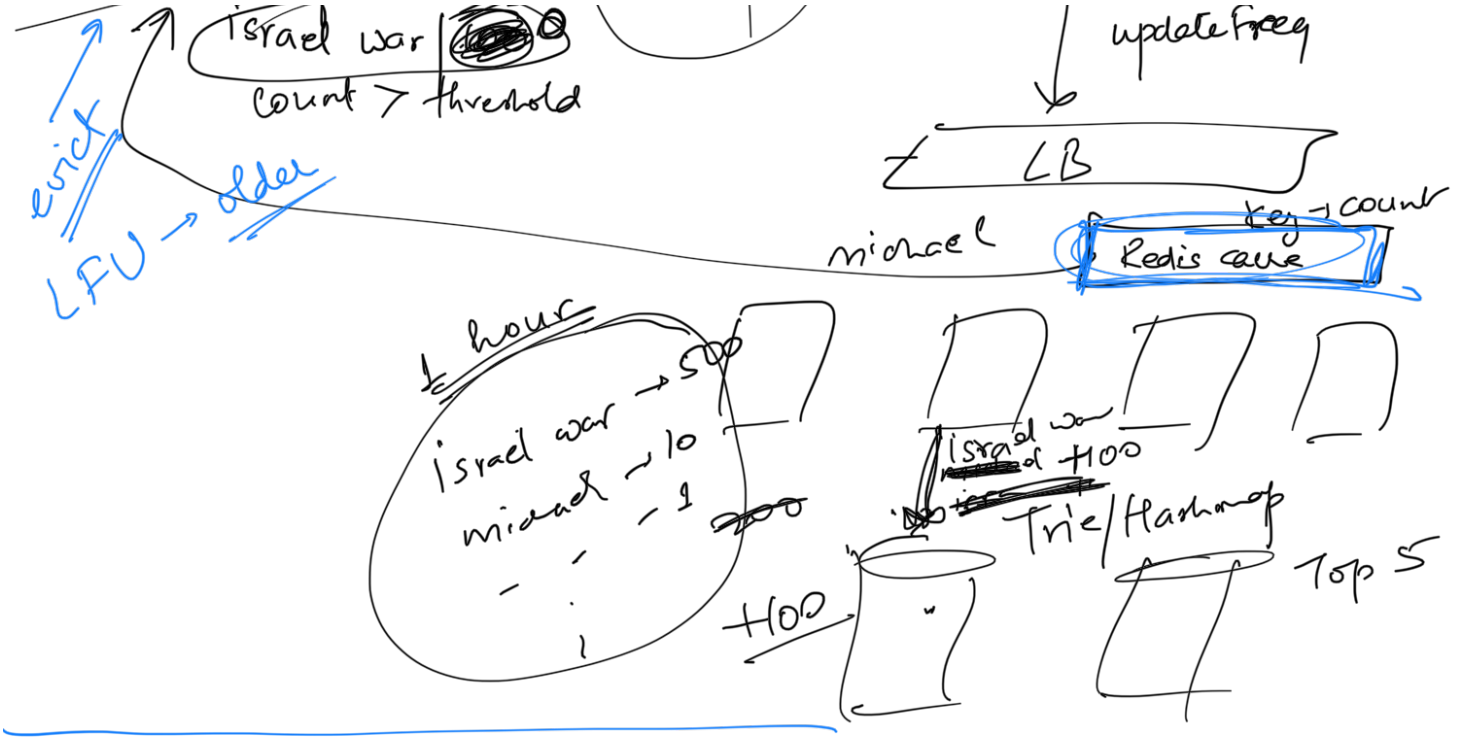
very low latency →

- getTop5suggestion (prefix)
- updateFreq(term)




"michael jackson" → 500
"mich obc " → 498
"michael" → 301
301
michael
mic
mich
micha → frequency, top 5 → ["michael jackson", 500, mich-- " - -]
michael → freq++, top 5
getTop5 (key):
michael
michae
micha
mich
mic
top 5
time taking
1 read & write heavy system
Buffering writes
1 hour
michael
Trie
+100
freq = freq+100
Hashmap
250,000
248,000




Sampling → 10 lakh people

80 million → 10,000 → random sample

Trend

Sampling

```
updateFreq(search_term):
    int r = rand() % 100        → 0-99
    if (r != 0) return;         ← 1/100th
```

1% sample ← trie/hashmap

~~100 billion~~ 150 crore → 1.5 crore (random)

~~100~~ crore
70% → darker
30% → fair

~ 70% dark → 1 crore
~ 30% fair → 50 lakh

50 Cr

① Read/write ratio

② Trie/Hashmap → read fast

③ Shard

④ Recency

Hashmap

mic

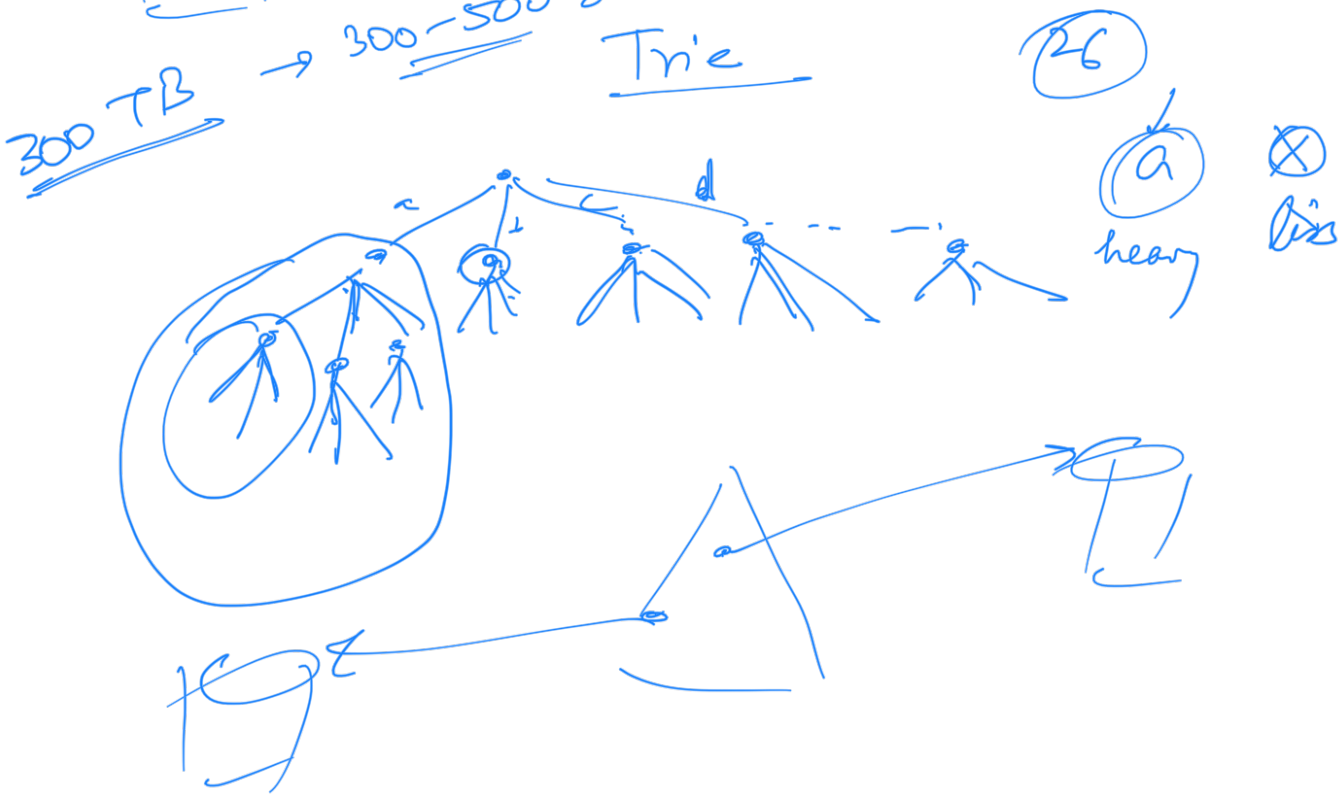
read

Key → freq, top 5

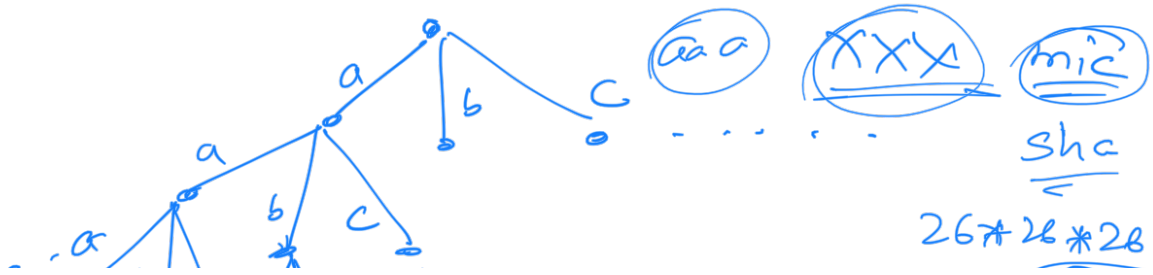


300 TB → 300-500 shards

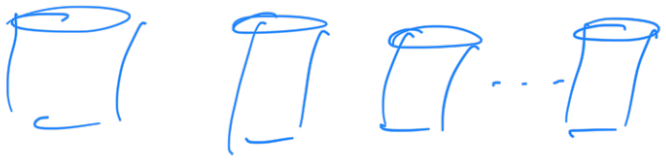
Trie




mic



Consistent hash



~18K

"aaa" "aab" "aac"

"zad" ....

israel war → 100 ← recency

ishani world → 500

freq + recency

Time decay

day 1

michael → 100

ishani word → 10

day 2

day 100

israel → 100

ishani → 2

tdf = 1.1

tdf = 1.01

day 1

michael → 10,000

10000 / 1.1^10

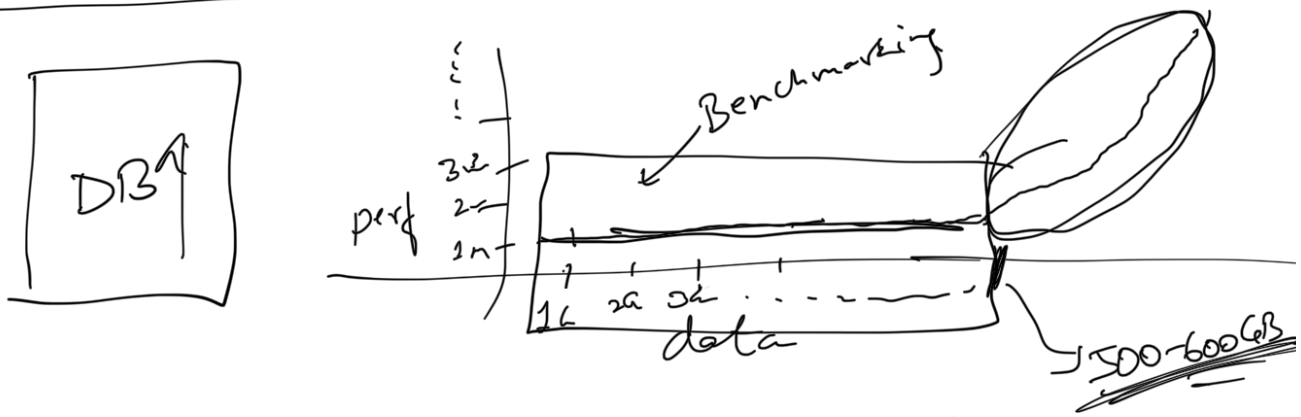
day 100 → 4000

michael → 9000

10 times

day → $C_{cur\_day} + C_{prev\_day}/tdf$ = 10,000 / (10)^1 = 10,000 / ~100 = 1

$+ C_{prev\_prev\_day}/tdf^2 + \frac{C_{d-3}}{tdf^3} + \frac{C_{d-4}}{tdf^4} + \dots$

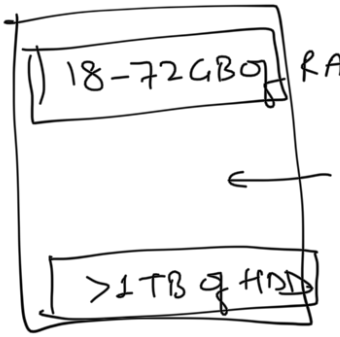



1 TB

Redis

Kafka → microservices

BigTable | DynamoDB → Consistent Hashing

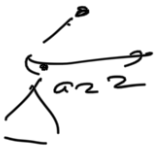


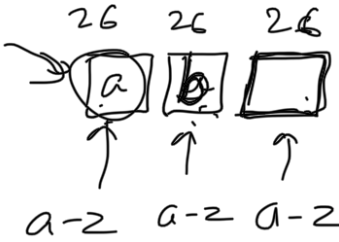

AWS → EC2 machine

| | RAM | CPU | HD |
|---|---|---|---|
| t2.micro | | | |
| t3.large | | | |

Relational DBs → Tries  Oracle DB

aaa, aab, aac, aad, .....




26 + 26 + 26 ....

26 * 26

+ 26 * 26

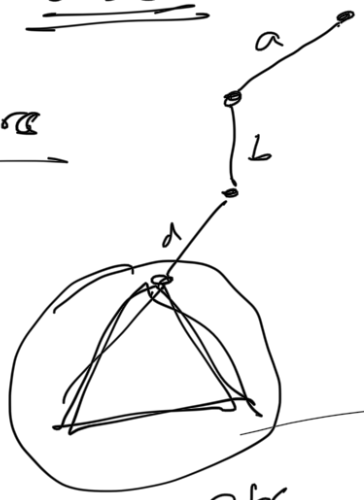
26 + ...

= 26 * 26 * 26

= ~18000

abd

18000 users




CH

100 shards


abc

aba

>1 subtree

180 subtrees.

18000 / 100 = 180