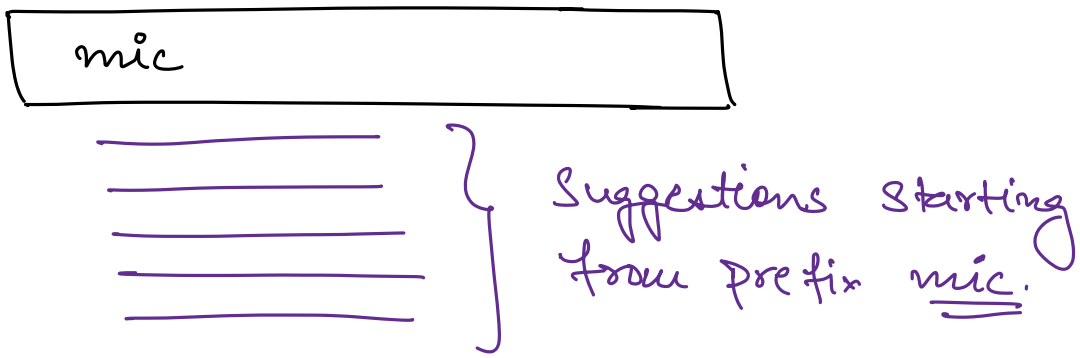
Agenda.

⇒ Case Study: Design Typeahead.



⇒ 4 steps.

1) MVP.
   ↳ Minimum Viable Product.

2) Scale Estimation
   (back of the Envelope)
   - # of users
   - QPS (query per sec)
   - Storage requirements. ⇒ Is Sharding required?
   - # of read queries
     # of write queries

   Read heavy | Write heavy | Both read & write heavy

# of reads >>>> # of writes ⇒ Read Heavy System.

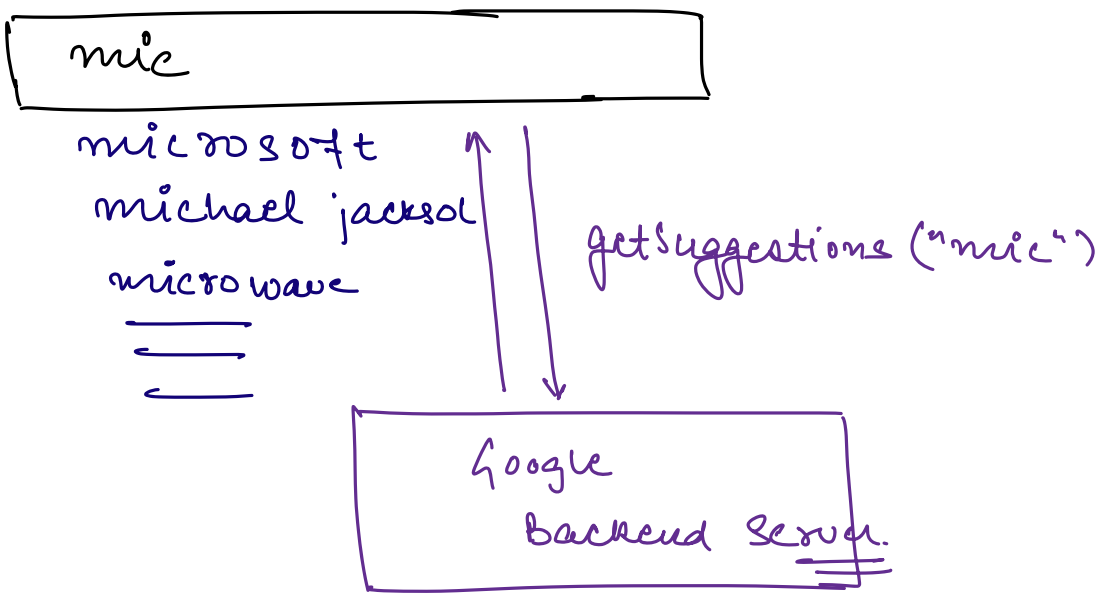
3) Design Trade-Offs / Goals.

- High Consistency (vs) High Availability.
- Latency.

4) Design Deep Dive

- API's. (3-4 main APIs)
- HLD. (Architecture Diagram)
- Data flow.

Google Typeahead.



MVP

1) SearchPrefix / getSuggestions

2) Max 10 Suggestions.

3) Minimum 3 characters required to get the Suggestions.

4) Relevant Suggestions.

↳ based on the recent searches.

Scale Estimation / Back of the Envelope

⇓

Assumptions.

# of users = 3B.

DAU = 1B

↑

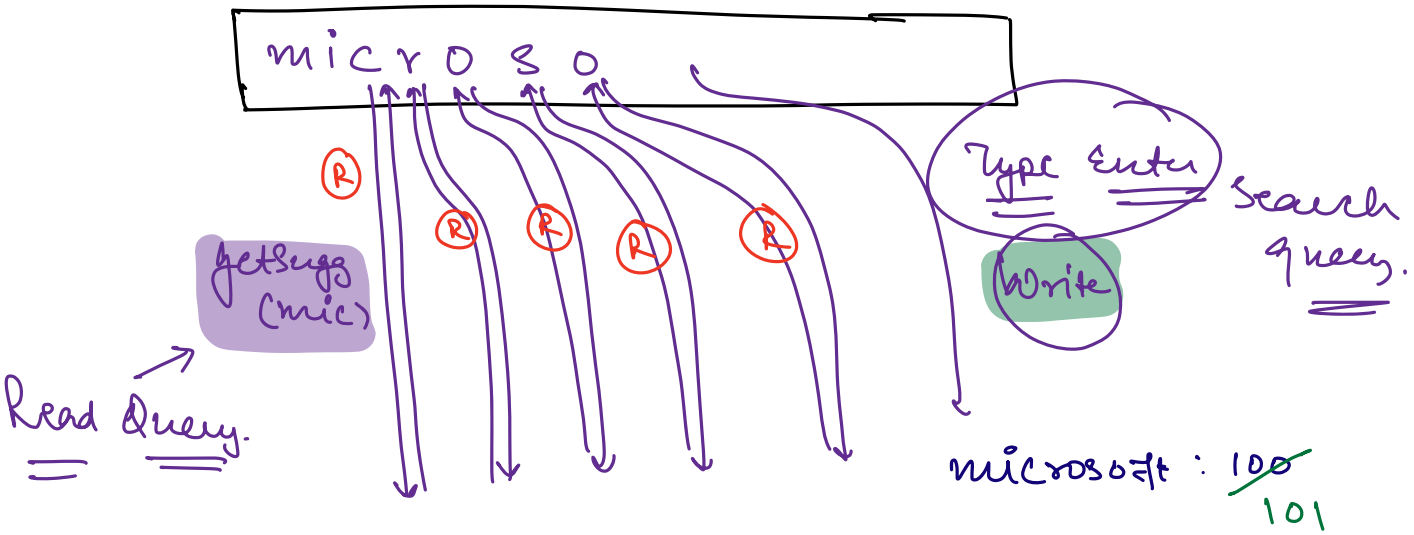
Daily Active Users

Avg # of searches per user per day = 20

# of Search queries / day = 20 × 1B.

= 20B.

# of write queries.



# of write queries / Day = 20B.

$$\text{Write qps} = \frac{20B.}{24 \times 60 \times 60} \rightarrow 86400$$

$$= \frac{20 \times 10^{9}}{10^{5}}$$

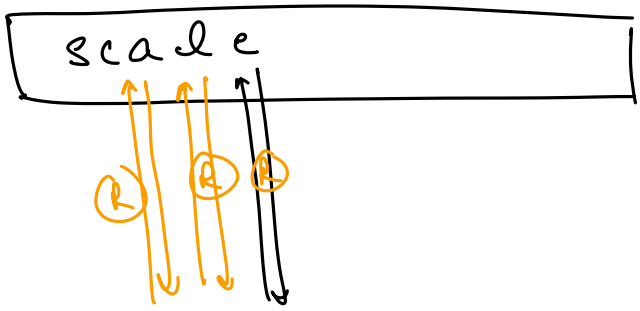
$$= 2 \times 10^{5}$$

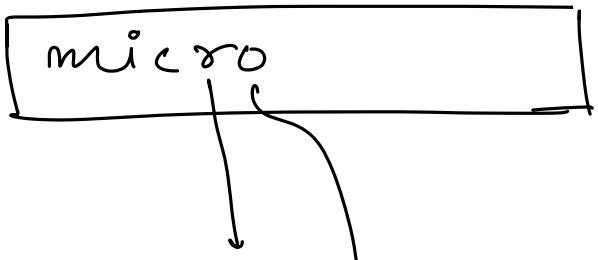
$$= 200K \text{ qps.}$$

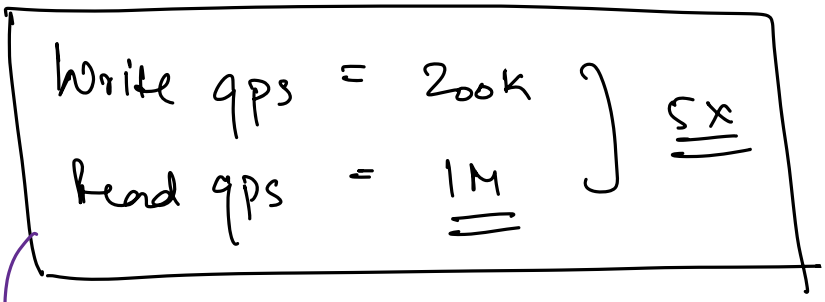
Avg # of getSuggestions call per search query = 5

Read qps = 5x 200K

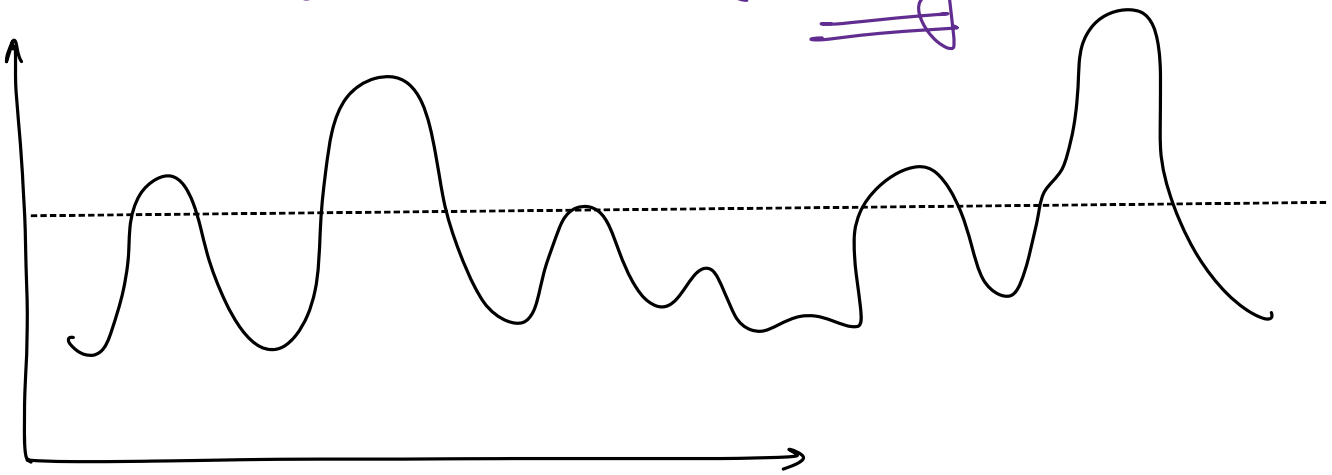
getSuggestions. = 1M.







→ Both read & Write heavy.

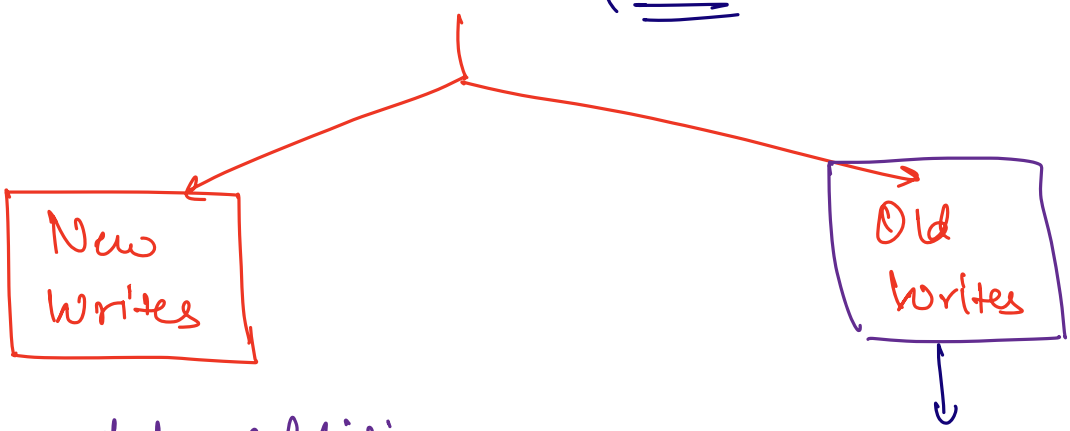


Peak qps ⇒ 2 × Avg. qps

Storage Requirements.

Write qps	= 200K
Read qps	= 1M

20B writes / day.



⇒ New data addition.

Scaler : 1

SST : 1

{String, Long}

Search query + Long

70-80 B. + 8B. } ⇒ 100B.

Only increment in frequency is required.

⇒ No new data addition

New queries / Day = 10% × 20B

= 2B

2B × 100 Bytes per Day.

200 GB.

1 Yr = 200 GB × 365

400

= 8 × 10 × 10^3 GB

= 80 TB.

10 Yr ⇒ 800 TB. ⇒ 0.8 PB.

1 PB

⇒ Sharding

Design Trade Offs.

I) Highly Availability >> High Consistency.

II) Ultra low latency.

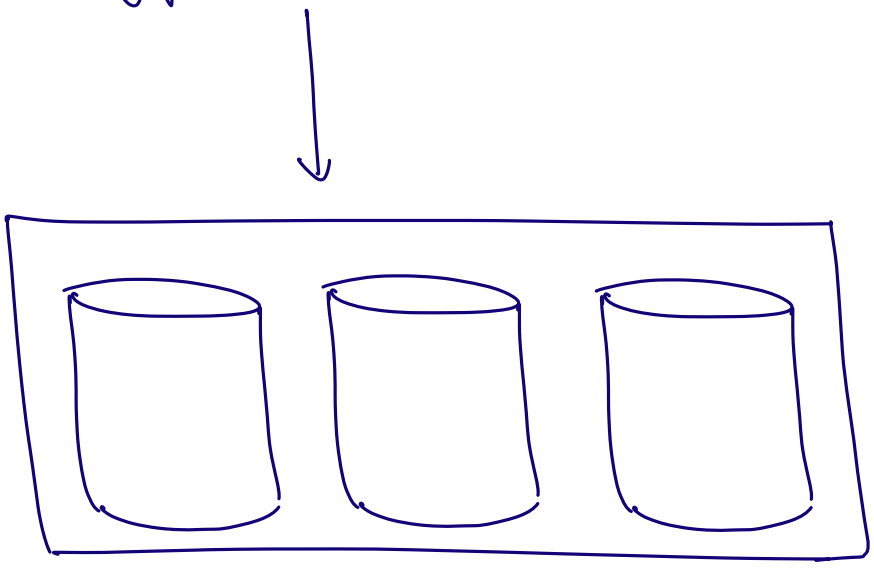
APIs

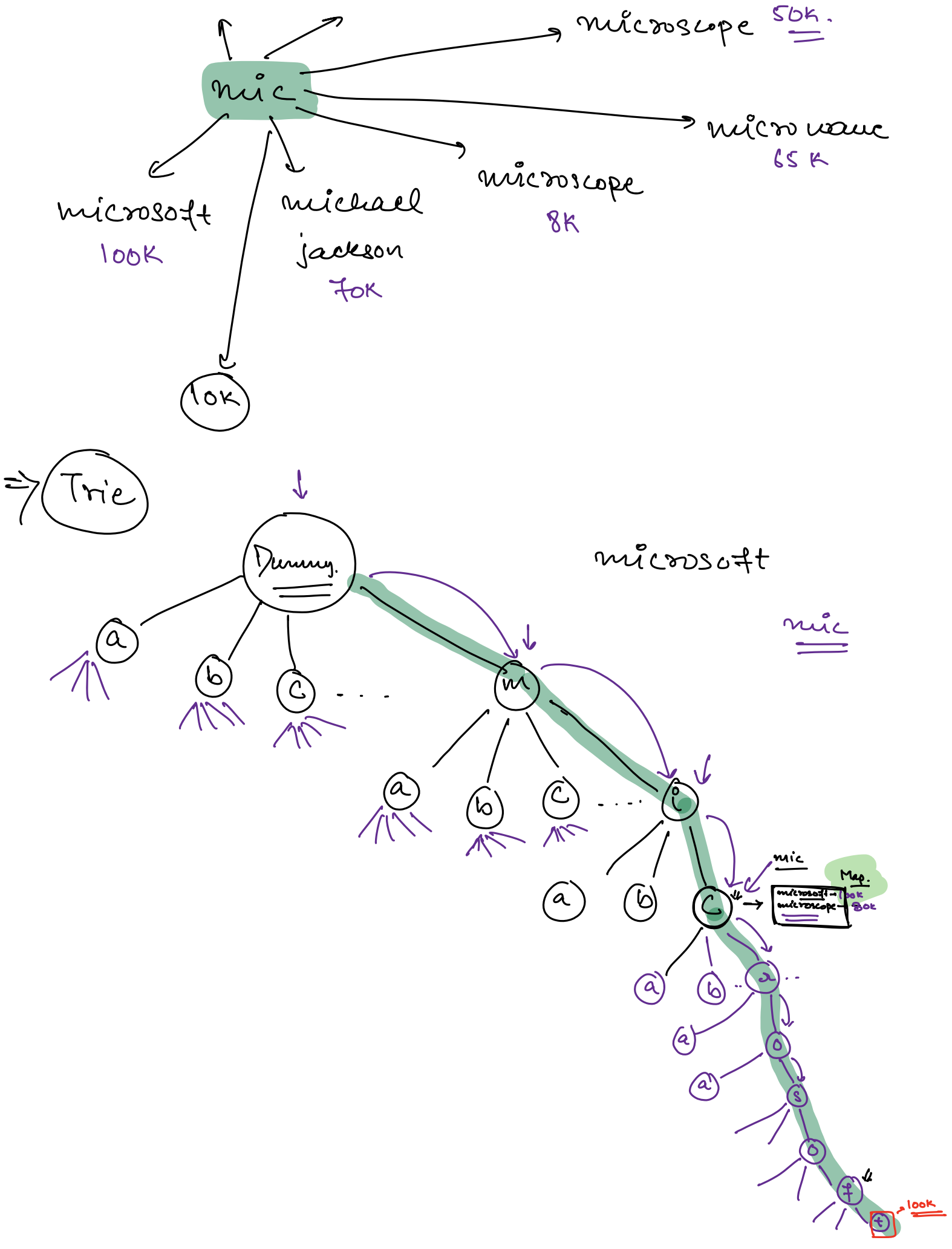
1) getSuggestions (prefix , limit)

2) updatefrequency (query)

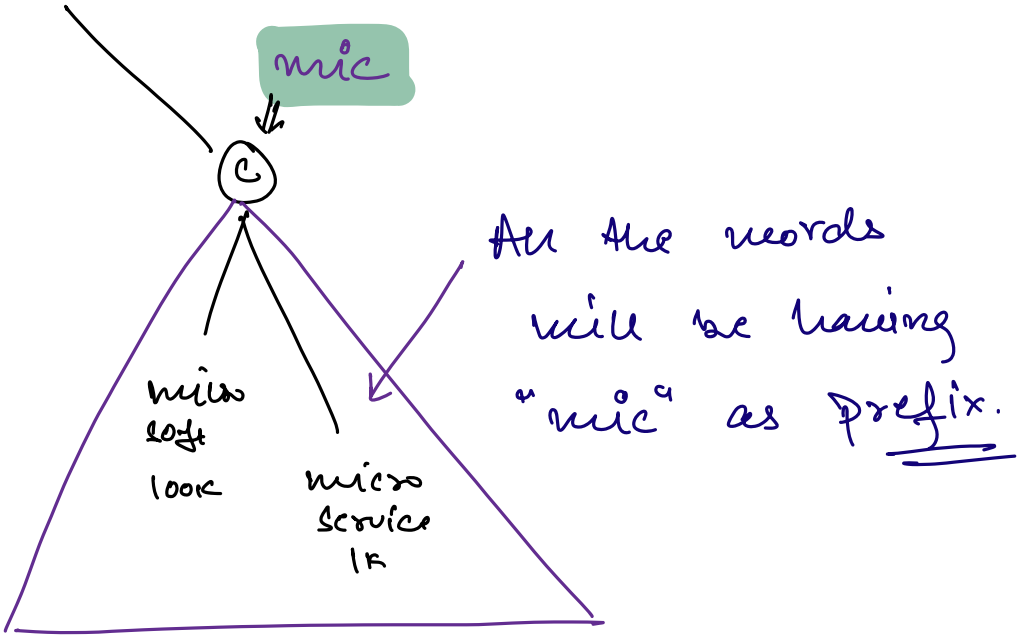
Design Dive.

getSuggestions ("mic", 5)

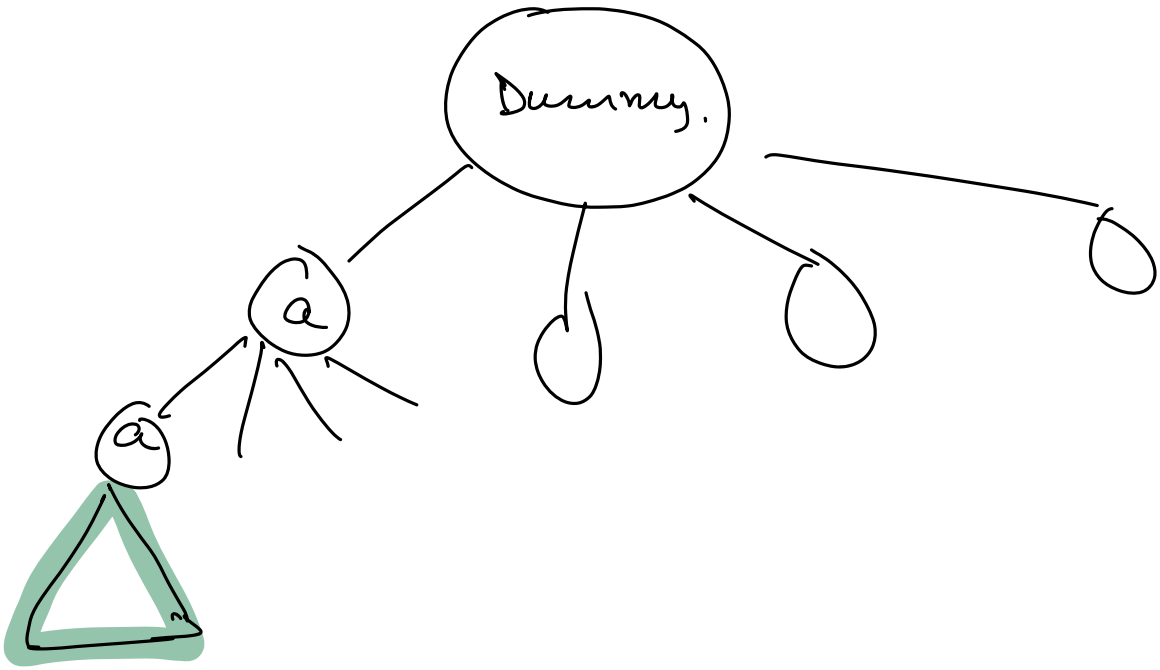



mic
microscope
50k.
microwave
65 K
microscope
8K
microsoft
100K
michael
jackson
70K
10K
Trie
Dummy.
microsoft
mic
m
i
c
Map.
mic
microsoft → 100K
microscope → 80K
look

Prefix. : "mic".



⇒ Backtracking Sol^n



=> for every Node : We can maintain a map with the most frequently accessed word starting with the prefix.

microsoft : —

microservice : —

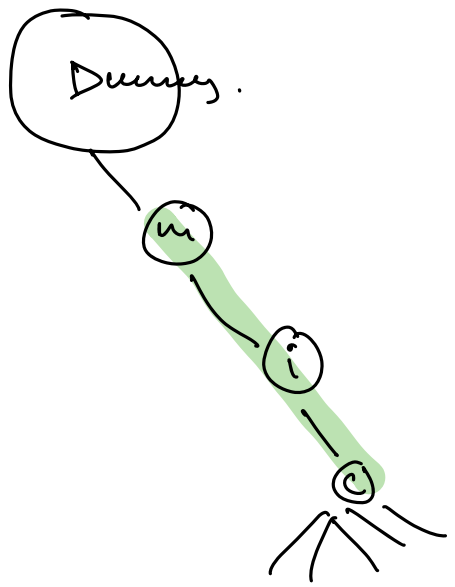
microscope : —

microwave

—

—

—



=> We'll continue Typeahead in the next class