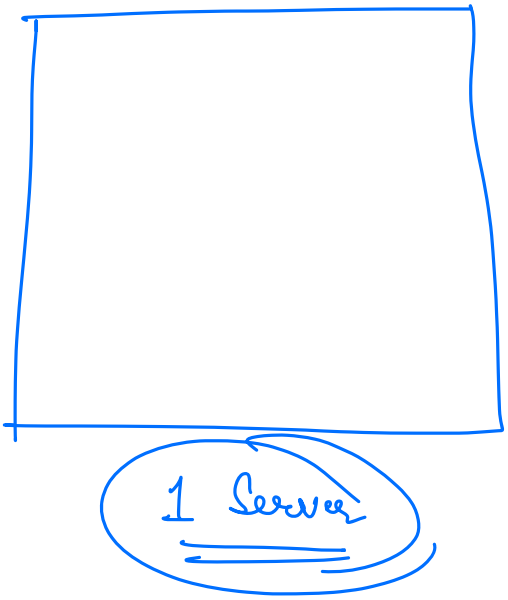
Complete | Design Google Typeahead

↳ (continuation of prev class)

192 TB of data

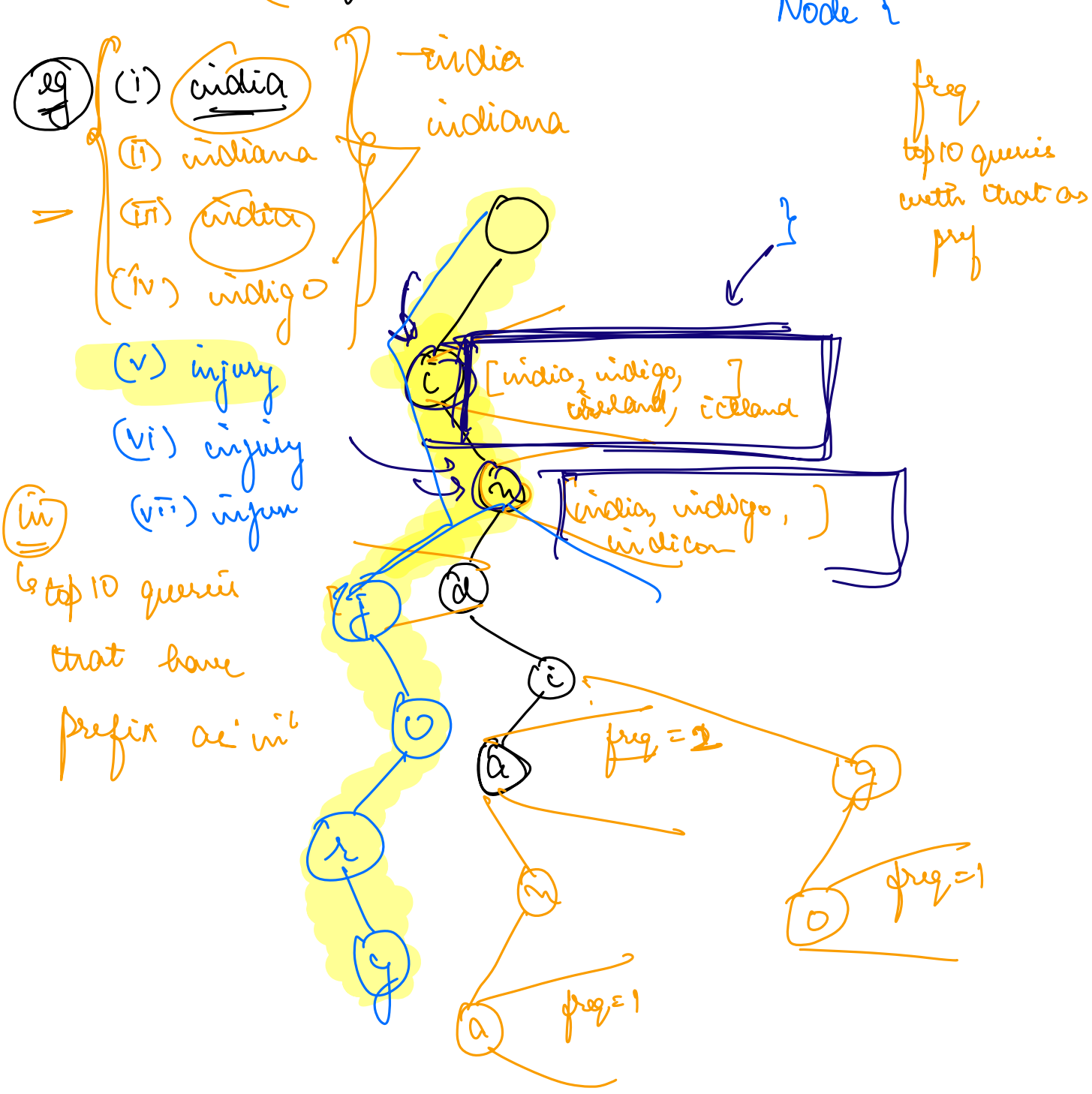


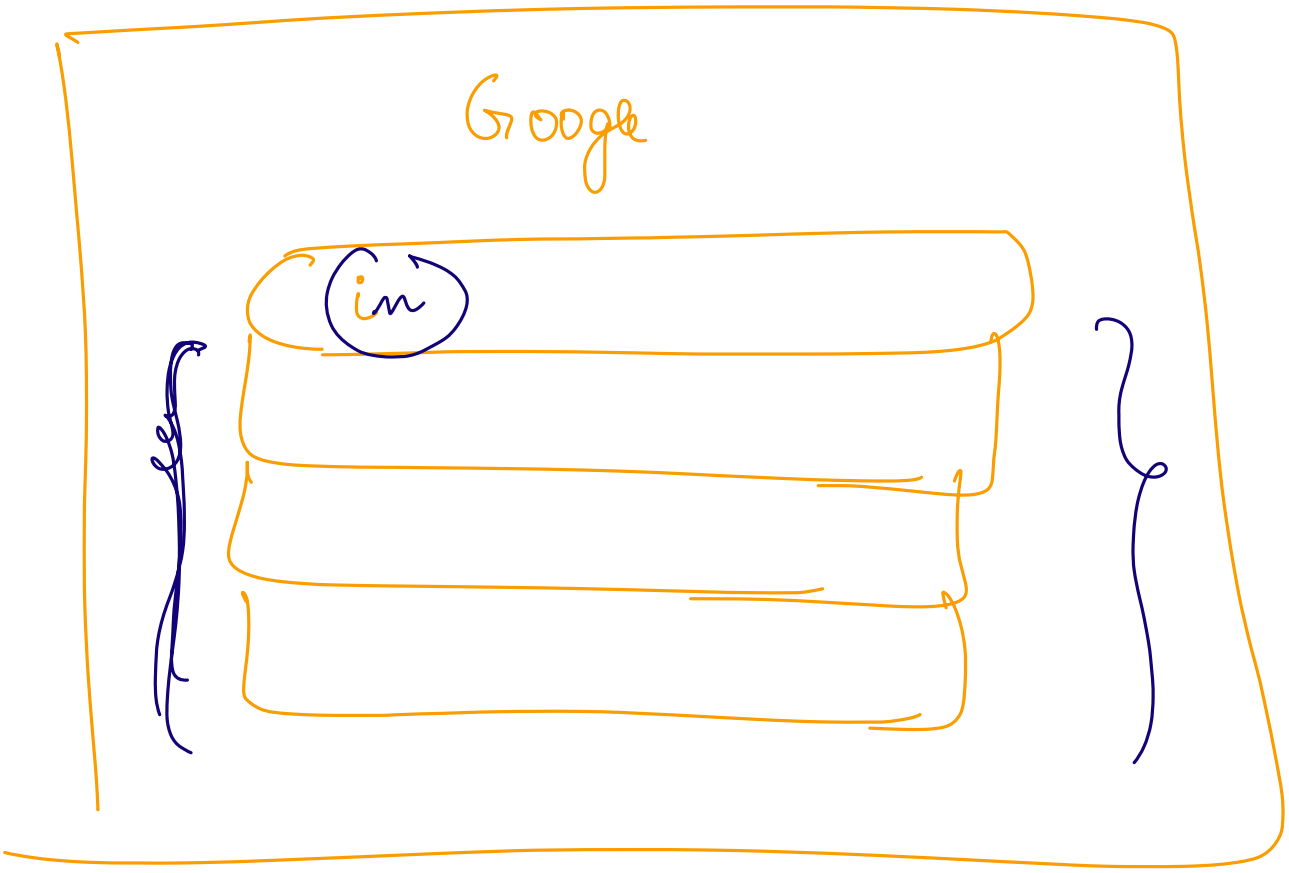
Size of data is less

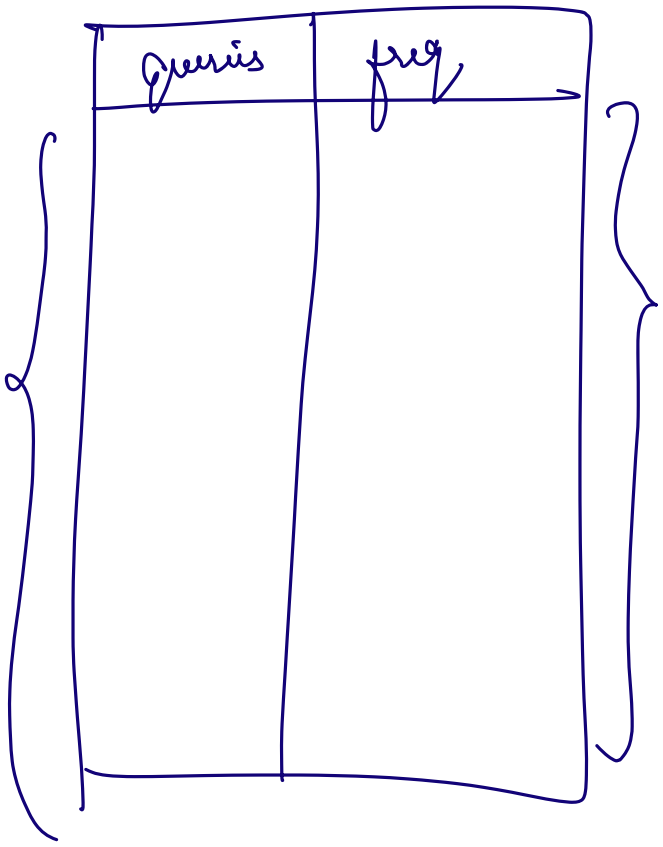
1. getRecommendations(prefix)
2. updateFreq(SearchTerm)

TRIES ⇒ do diff kind of op^{n} on prefixes for strings

⇒ let's create a trie and in that put every search query that ever happens at google alongwith its freq.




Google
in


queries
freq

showSuggestions(prefix)

(i) go to the node with the prefix

(ii) return list

updateFreq(searchTerm)

(i) go to the node of the term

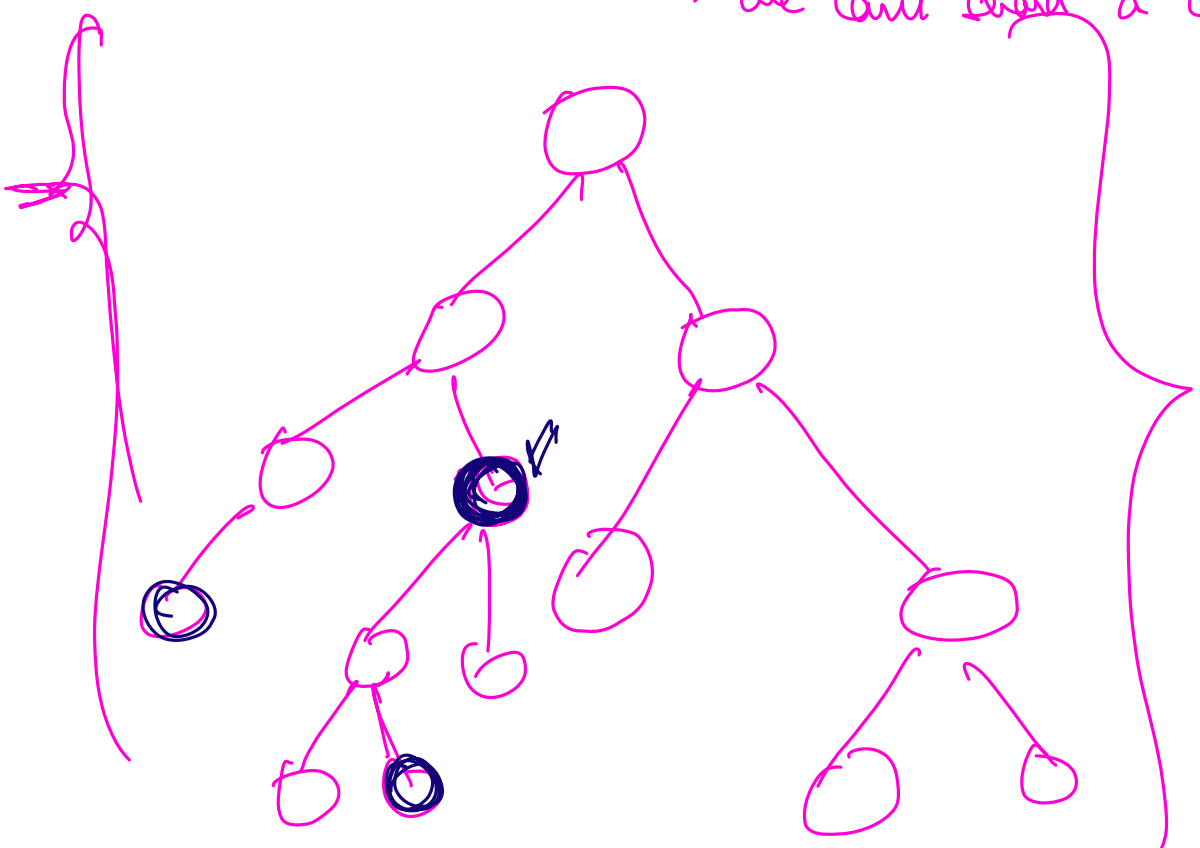
(ii) update freq at that node

(iii) for all node in path till search term:

update top5 if needed.

→ can I scale trie?

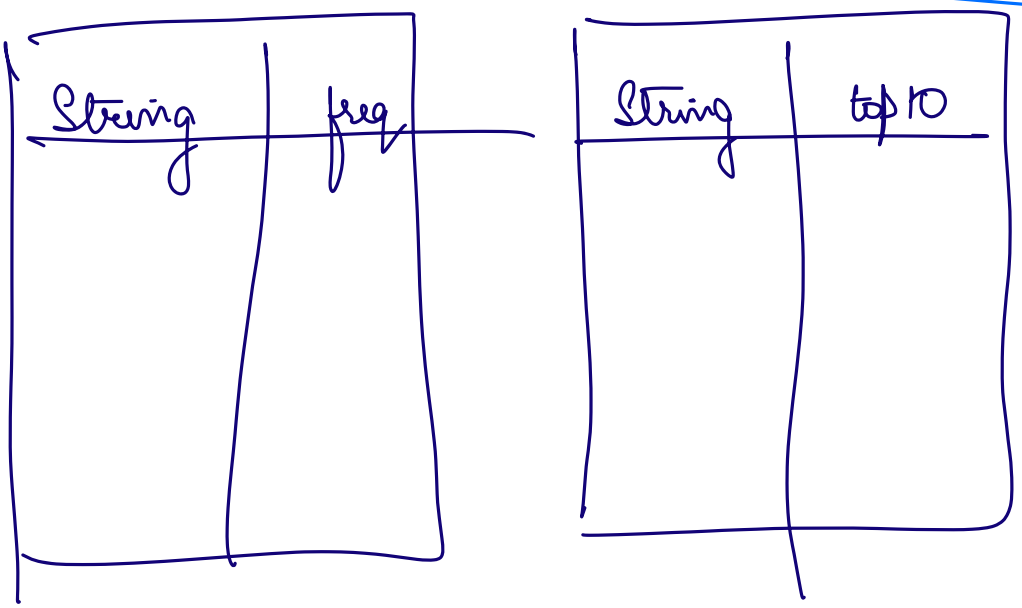
→ we can't load a trie



⇒ All benefits of Trie existed only in 1 machine

⇒ Every node in trie is storing some info about a String :

⇒
- (i) freq of that String as search q
- (ii) top 10 search queries having that String as prefix



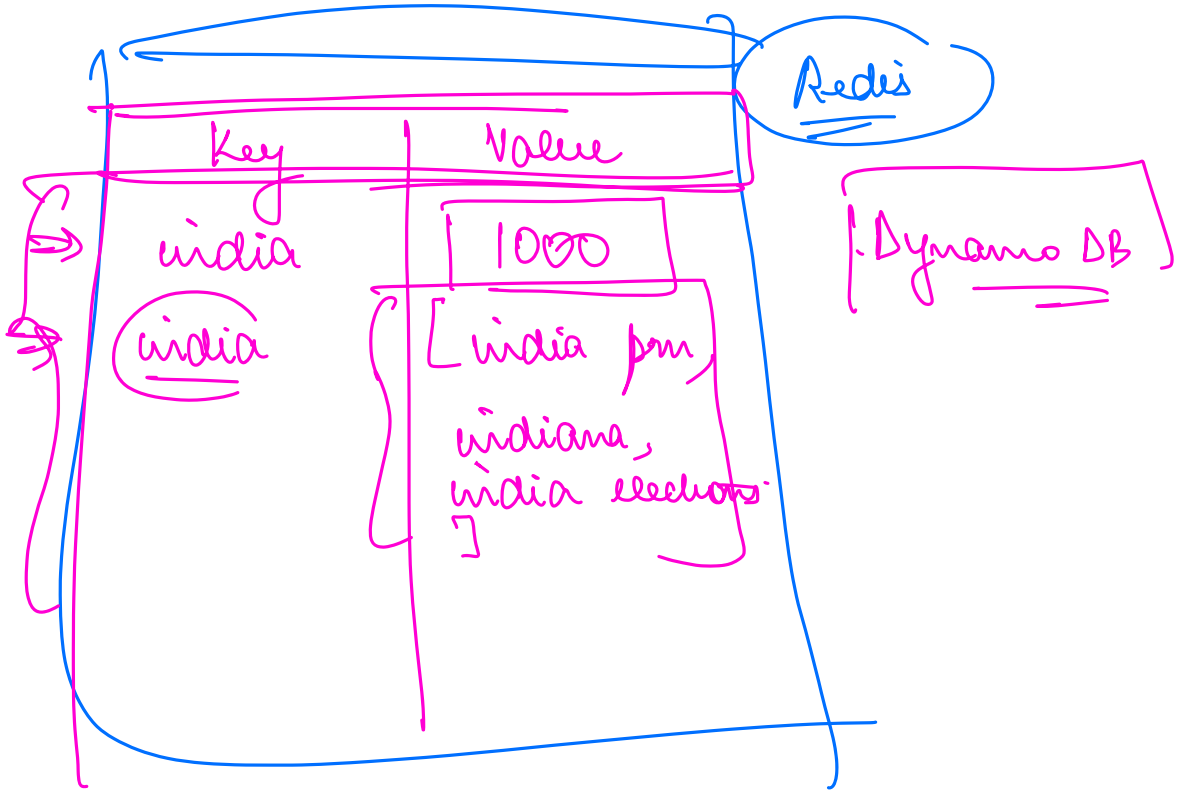
HashMap ⇒ Key, value

Key → String

2 HashMaps

1. Map<String, Integer> freqMap;
2. Map<String, List<String>> topQueries

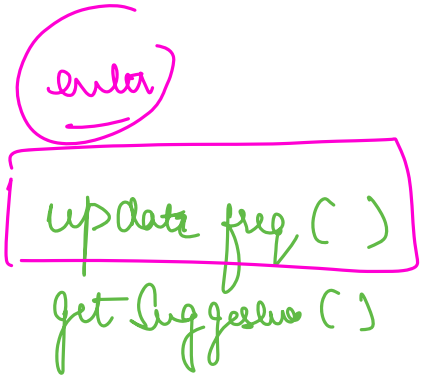
Key Value DB wide work



(1) A single machine won't be able to store all k-v pairs

↳ but here we can shard.

(2)



R : W

10 : 1

↓

10 : 0.1

→ 10 : 0.01

⇒ I need to somehow reduce writes

How ?

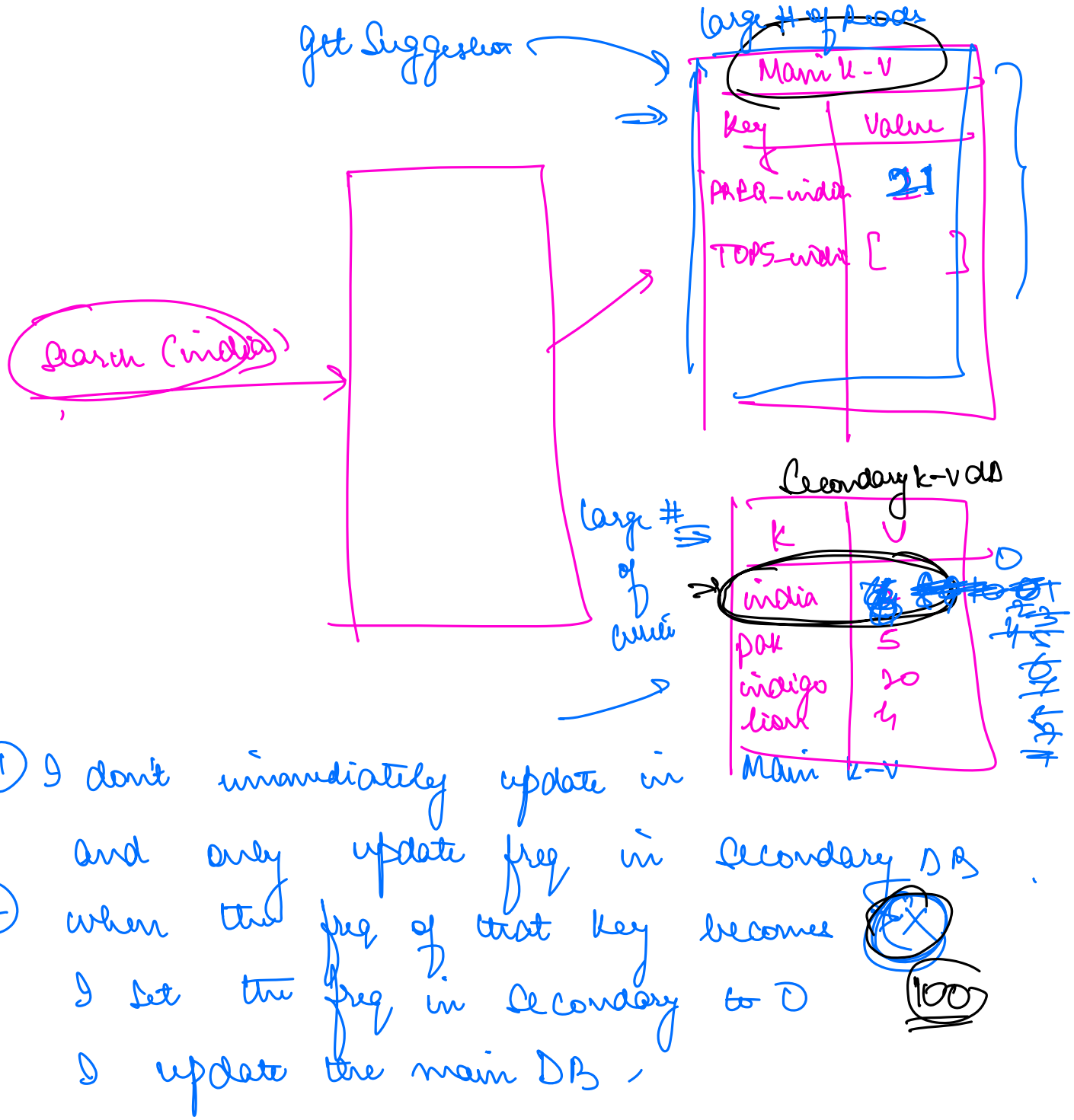
⇒ every new search will not really change the top 5 much

⇒ let me not update the map (K-V DB) on every search

⇒ I create another K-V DB

- indig

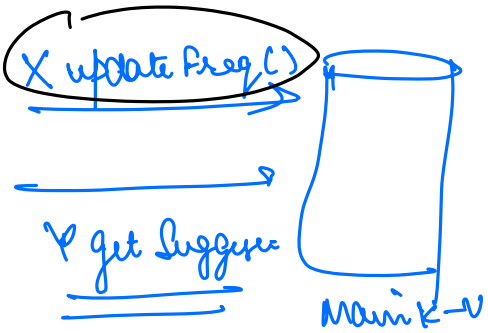
iiran



(1) I don't immediately update in and only update freq in Secondary DB.

(2) when the freq of that key becomes 100 I set the freq in Secondary to 0 & update the main DB.

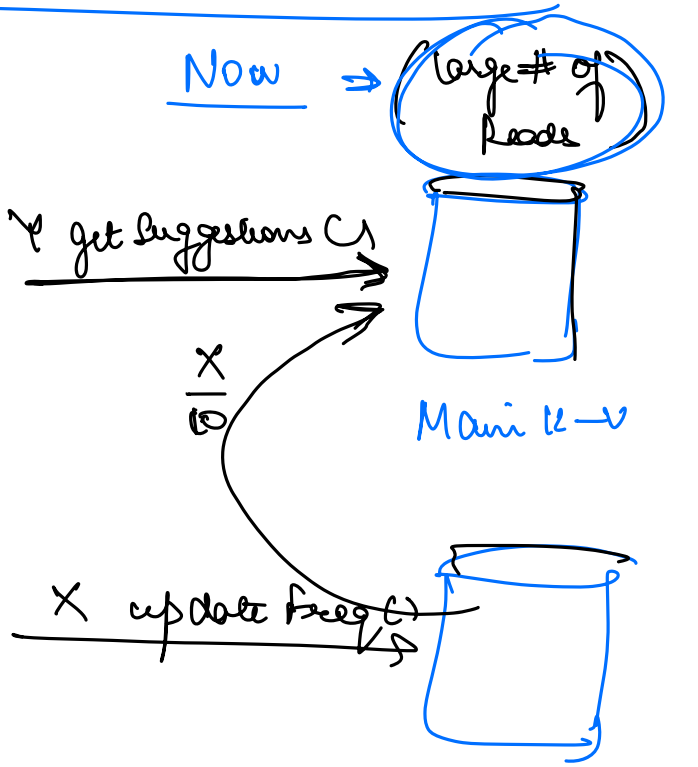
Earlier



X : Y

1 : 10

Now



Secondary K-V

$\frac{X}{10}$: Y

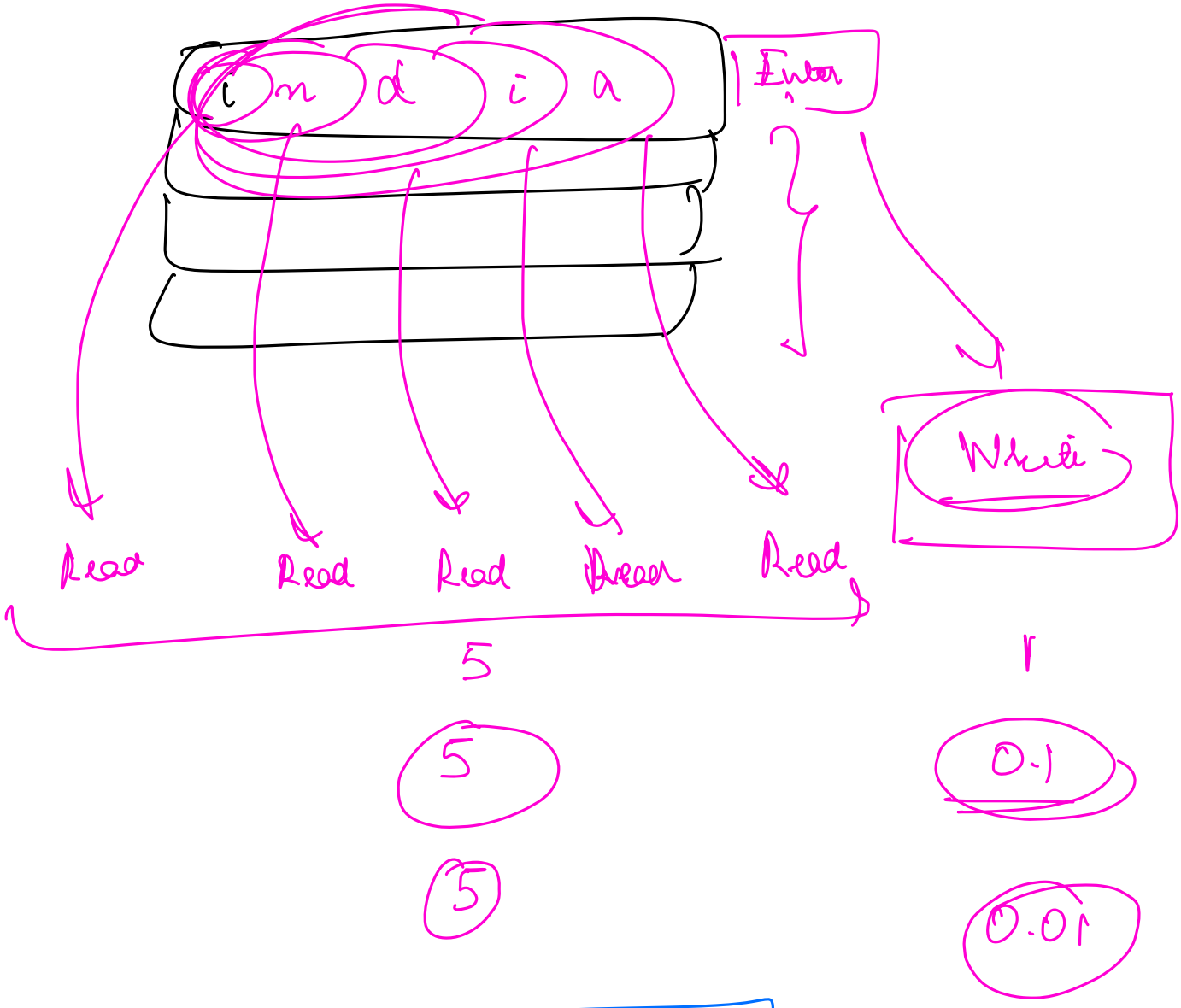
0.1 : 10 = (large # of Writes)

0.01 : 10

0.1 : 100

1 : 1000

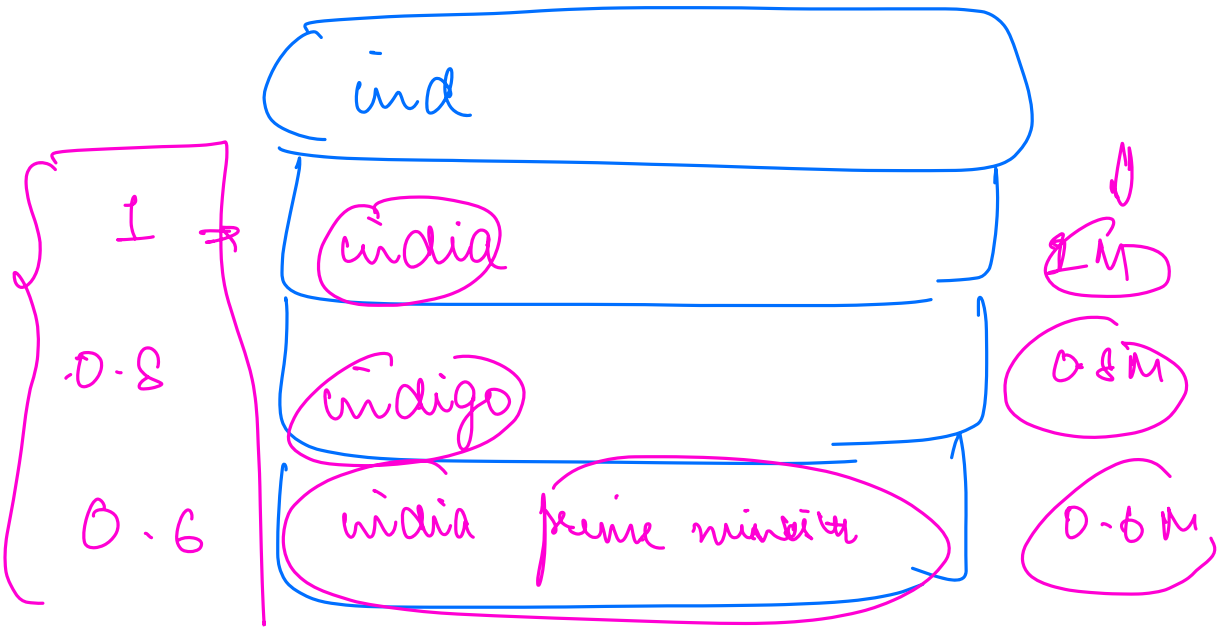
R W



Problem : More infra overhead

↳ manage more machines

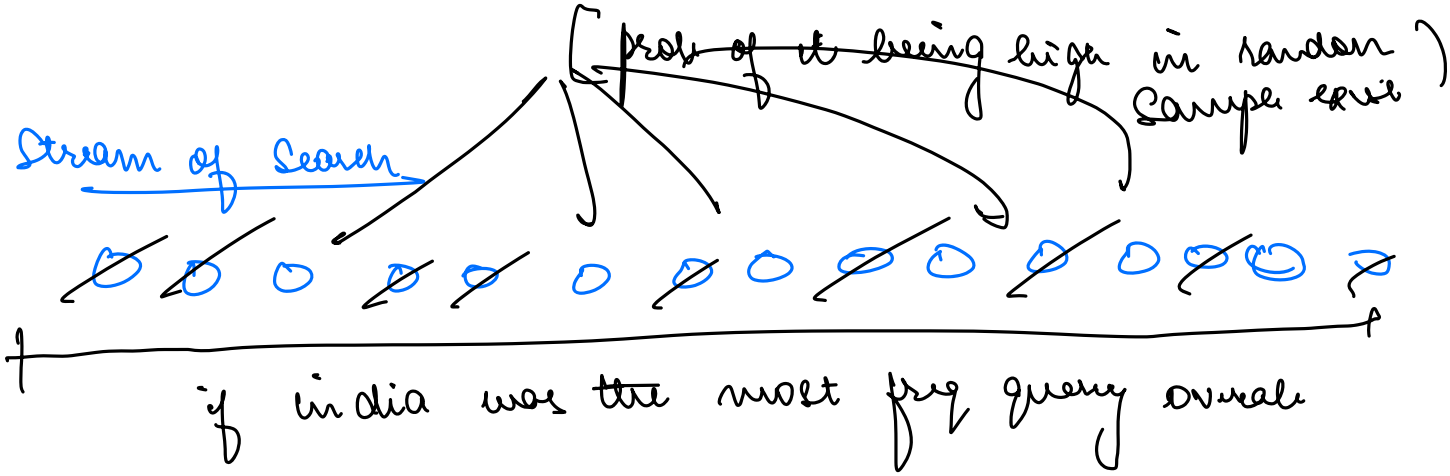
↳ 2 diff types of DB.



→ I don't care about exact value of each search term.

→ I just care about relative freq.

india > indigo > india prime min



if india was the most freq query ever

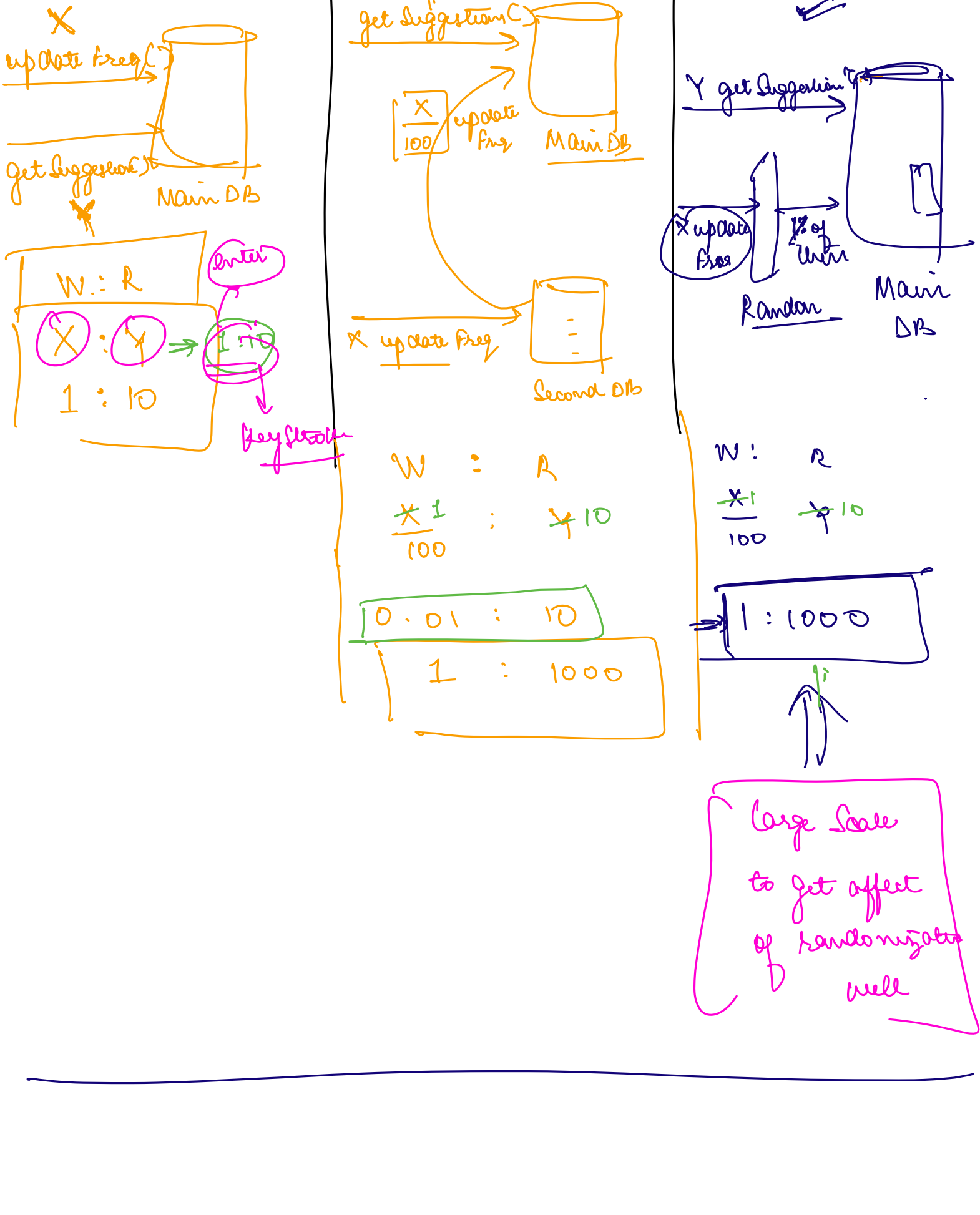
O ⇒ Search query

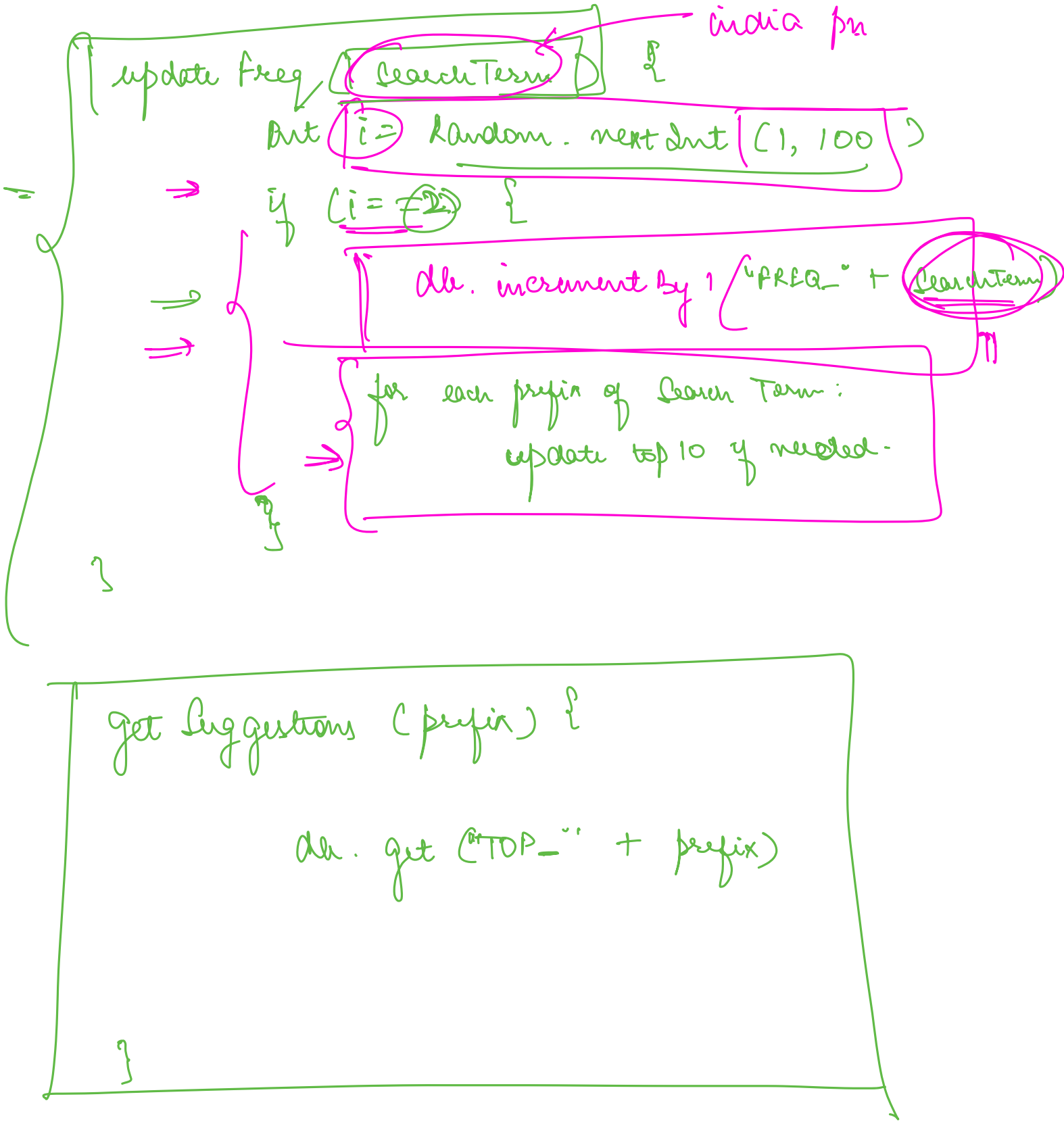
⇒ randomly ignore 50% of them

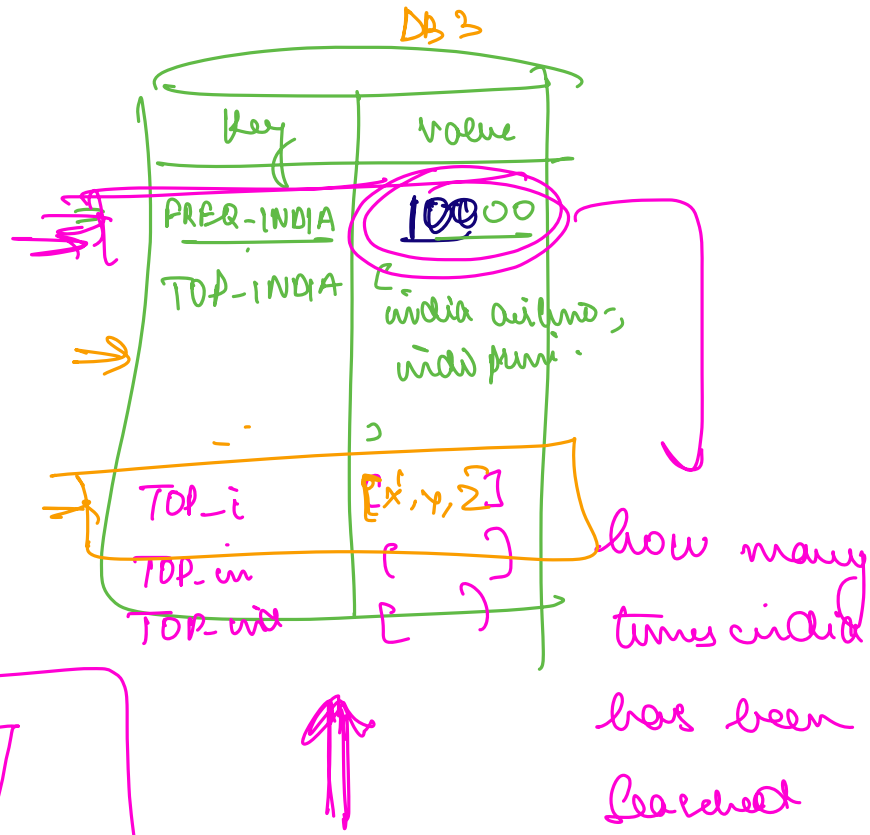
↳ don't update freq at all.

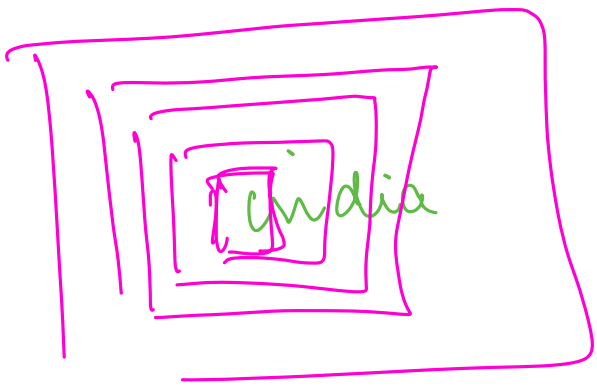
Random Sampling

V0 | V1 | V2


Case helper DB
Sampling
X
update freq()
get Suggestion()
Main DB
W : R
Enter
X : Y
1 : 10
1 : 10
Key Stroke
get Suggestion()
X
100
update Freq
Main DB
X update Freq
Second DB
W : R
X 1 / 100 : Y 10
0.01 : 10
1 : 1000
Y get Suggestion
X update Freq
1% of Users
Random
Main DB
W : R
X 1 / 100 Y 10
1 : 1000
Large Scale
to get effect
of randomization
well


india pn
updateFreq(searchTerm) {
int i = random.nextInt(1, 100)
if (i == 2) {
db.increment by 1("FREQ_" + searchTerm)
for each prefix of searchTerm:
update top 10 if needed.
}
}
getSuggestions(prefix) {
db.get("TOP_" + prefix)
}





⇒ How 1 DB machine will be able to handles reads and write

⇓

Sampling. ⇐ R:W ratio solved

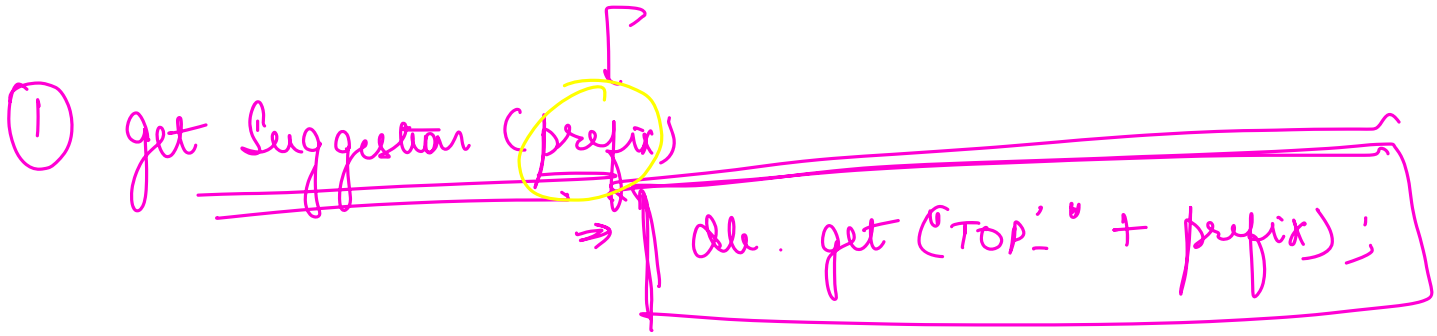
But how to handle 192 TB of data.

⇒ Sharding

But shard by what?

what sharding key?

1. Uniform dist / load.
2. Query to execute on as less DB as needed.



2 updateFreq (SearchTerm) {

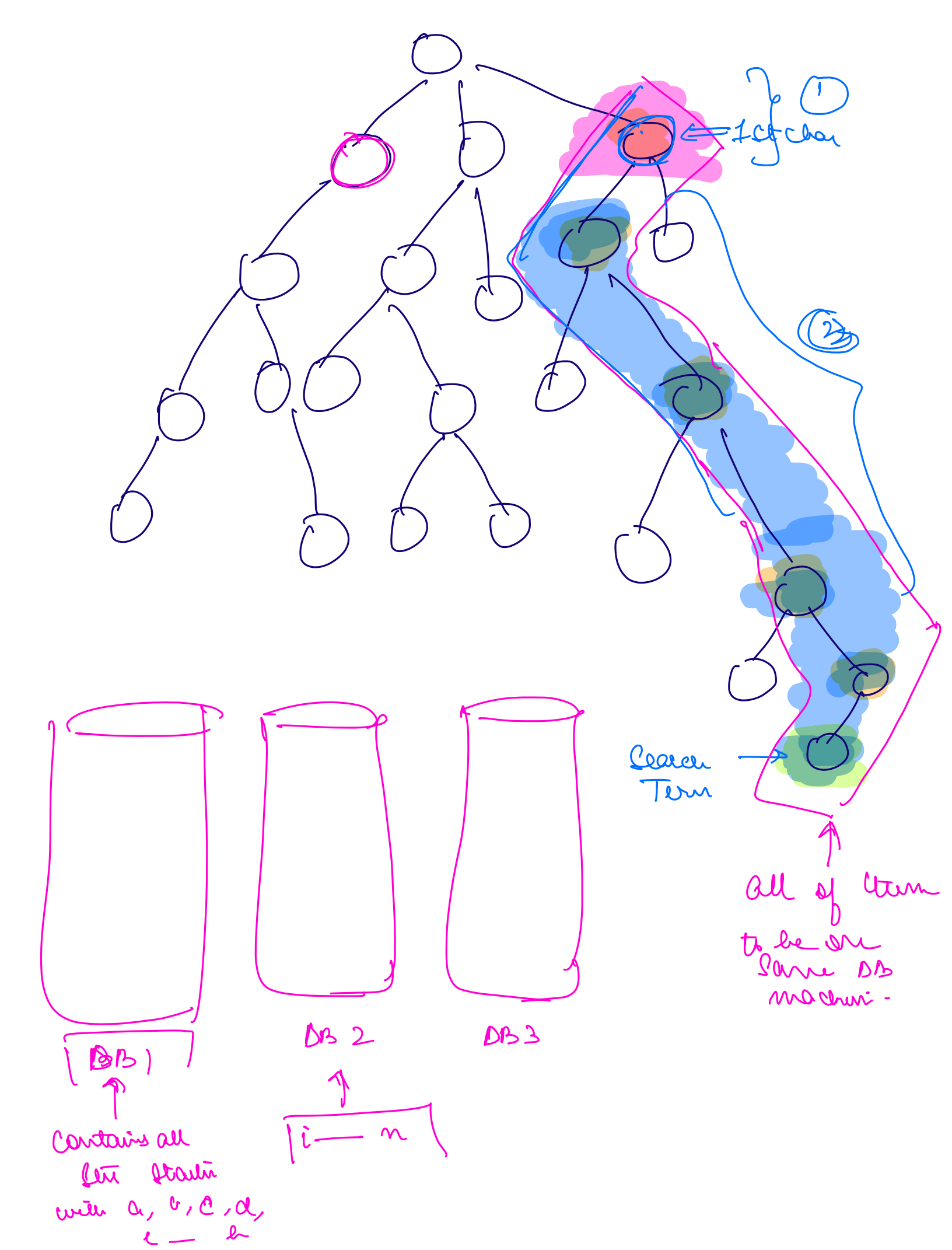
db.update("FREQ_" + SearchTerm")

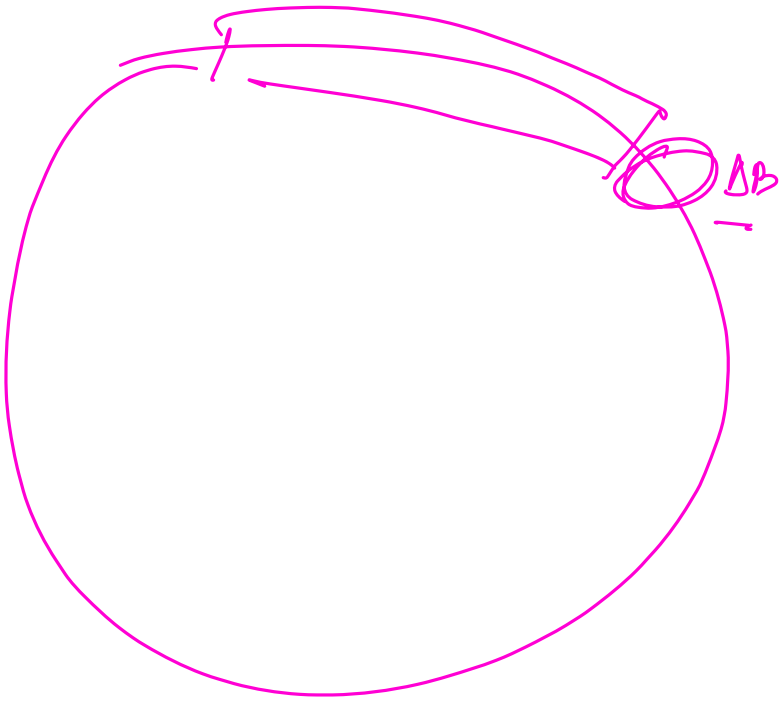
for all pref of SearchTerm:

db.updateTop10("TOP_" + prefix);

}

⇒ Ideal Sharding Key: SearchTerm as well as all prefixes on same machine.


1
1st char
2
Search Term
All of them to be on same DB machine.
DB 1
DB 2
DB 3
Contains all the words with a, b, c, d, e — h
i — n



(1) Shard by first char

⇒ only 26 shards possible at max

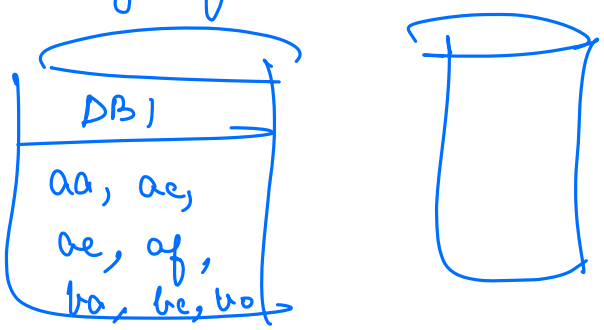
⇒ some shards may overflow

= 192 TB

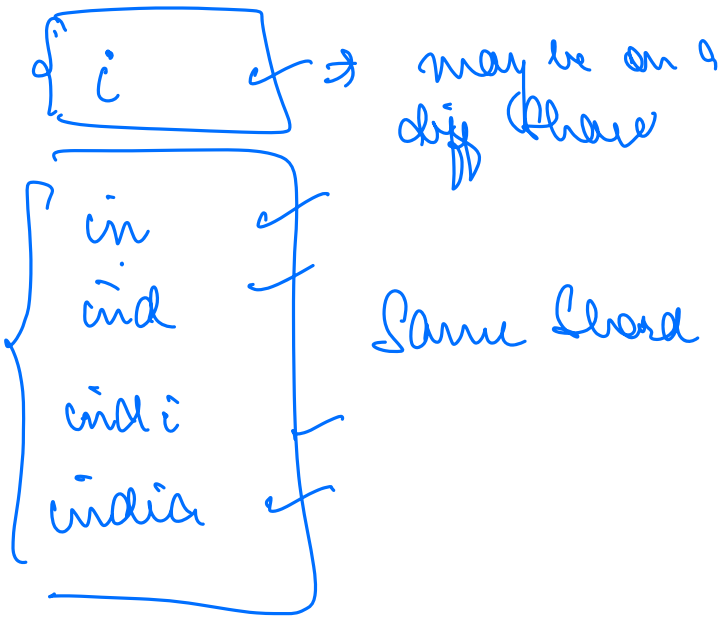
26

1 TB data/ shard

(2) Shard by first 2 char



26^26



192 TB

26^26

≈ ~~200~~ 8

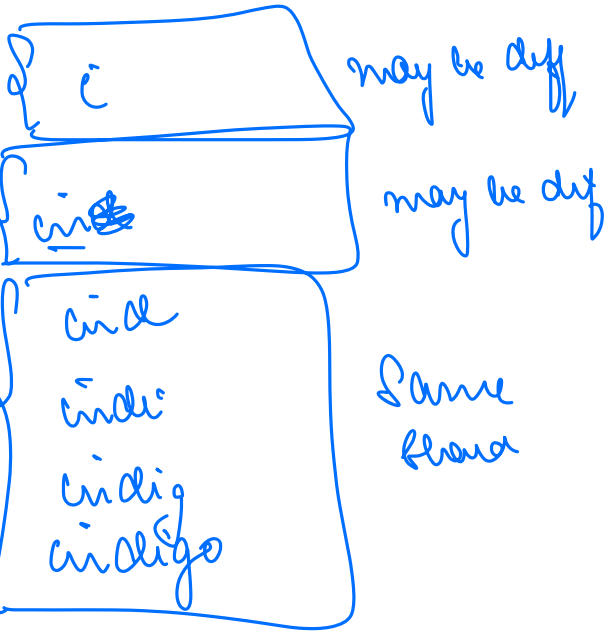
~~625~~ 25

≈ $\frac{10}{25} \simeq \frac{2}{5}$ TB

⇒ 0.4 TB

= 400 GB

③ Shard on first 3 chars

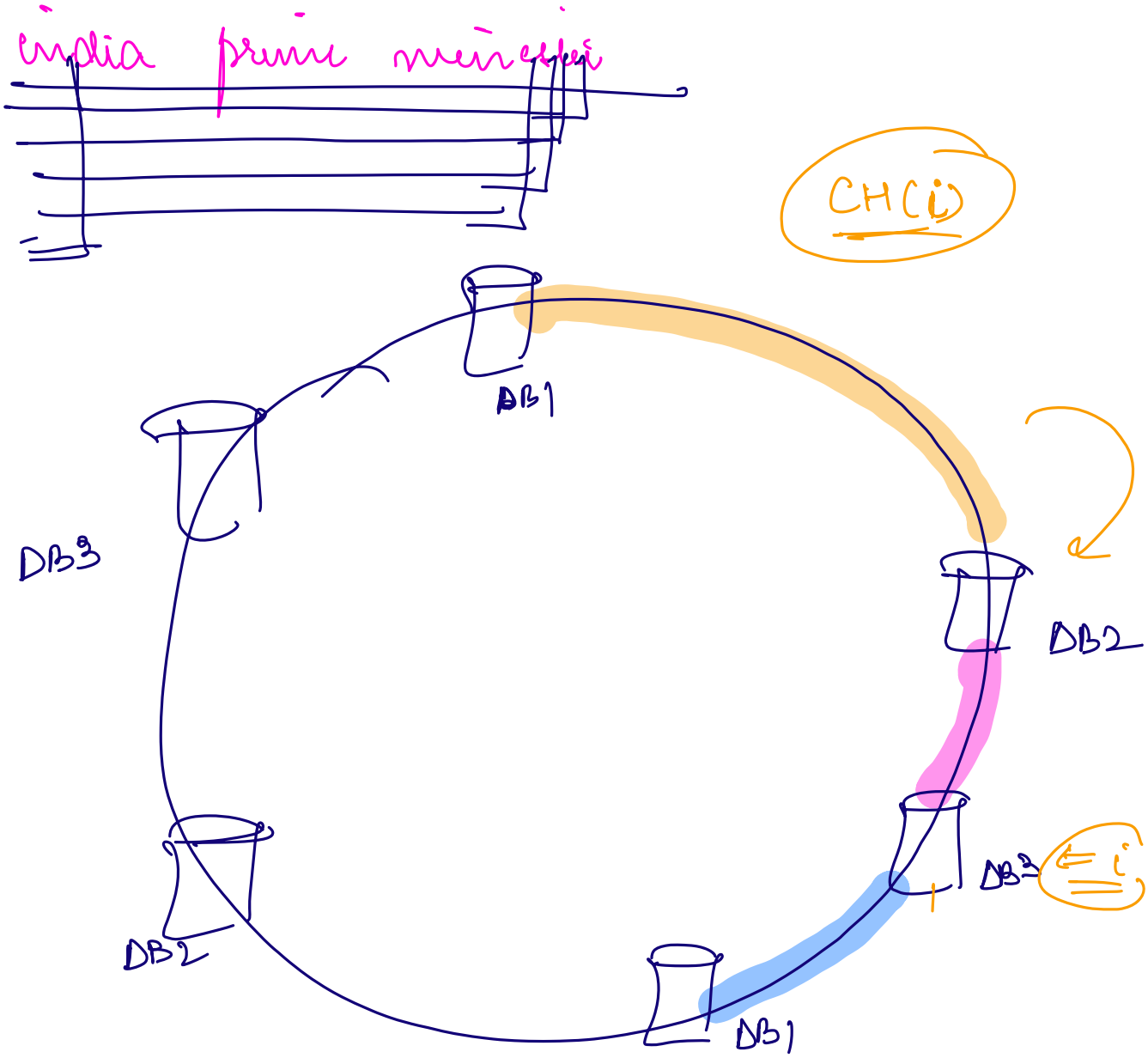


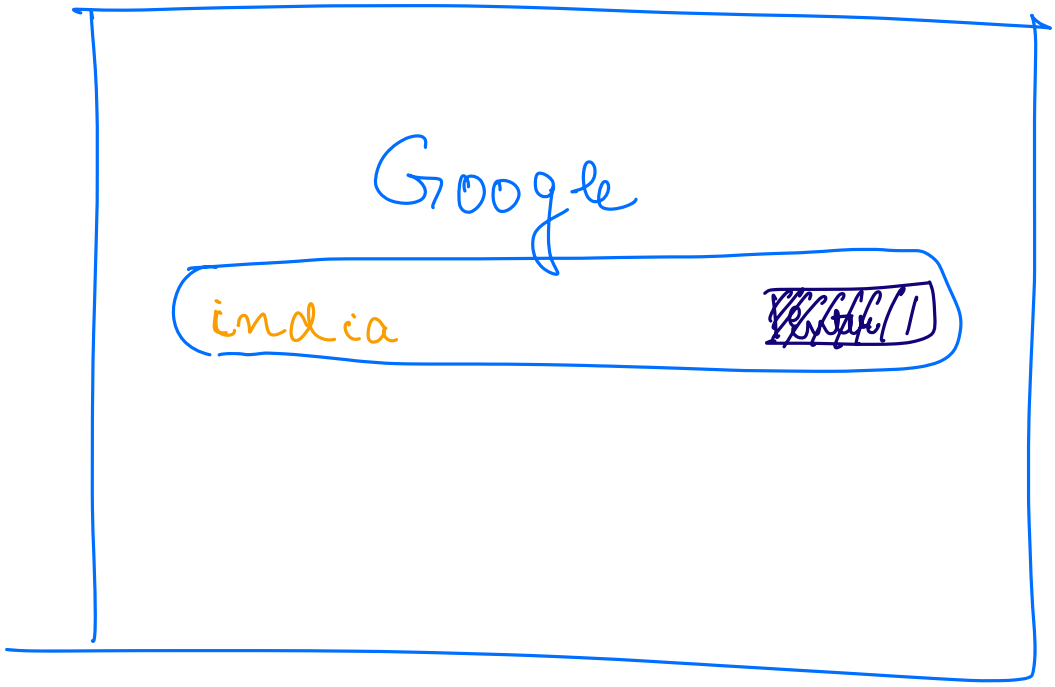
⇒ query across 3 machines

$\approx \frac{192\ TB}{26 \hat{} 26 \hat{} 26}$

$\approx 0.01\ TB$

$\approx$ 10 GB ↵





1. getSuggestions(i)

1 Machine

go to the Head that has value of i

dh.get("TOP-" + i)

2. getSuggestions(ia)

⇒ 1 Machine

3. getSuggestions(ind)
4. getSuggestions(indi)
5. getSuggestions(india)

6. updateFreq(india)

⇒ 99% reject req.

⇒ machine that has India ⇒ x

X.update(FREQ_ + india, 1)

X.update(indi , TOP10)
ind , TOP10)

dh.update(india)

dh.update(city)

→ 3 Machines

100 chars

1 char

H

H 2ch

3 ch

CH(i) → 1

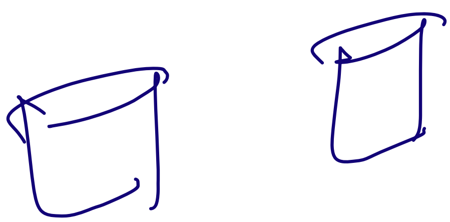
CH(in) → 3

CH(ind) → 2

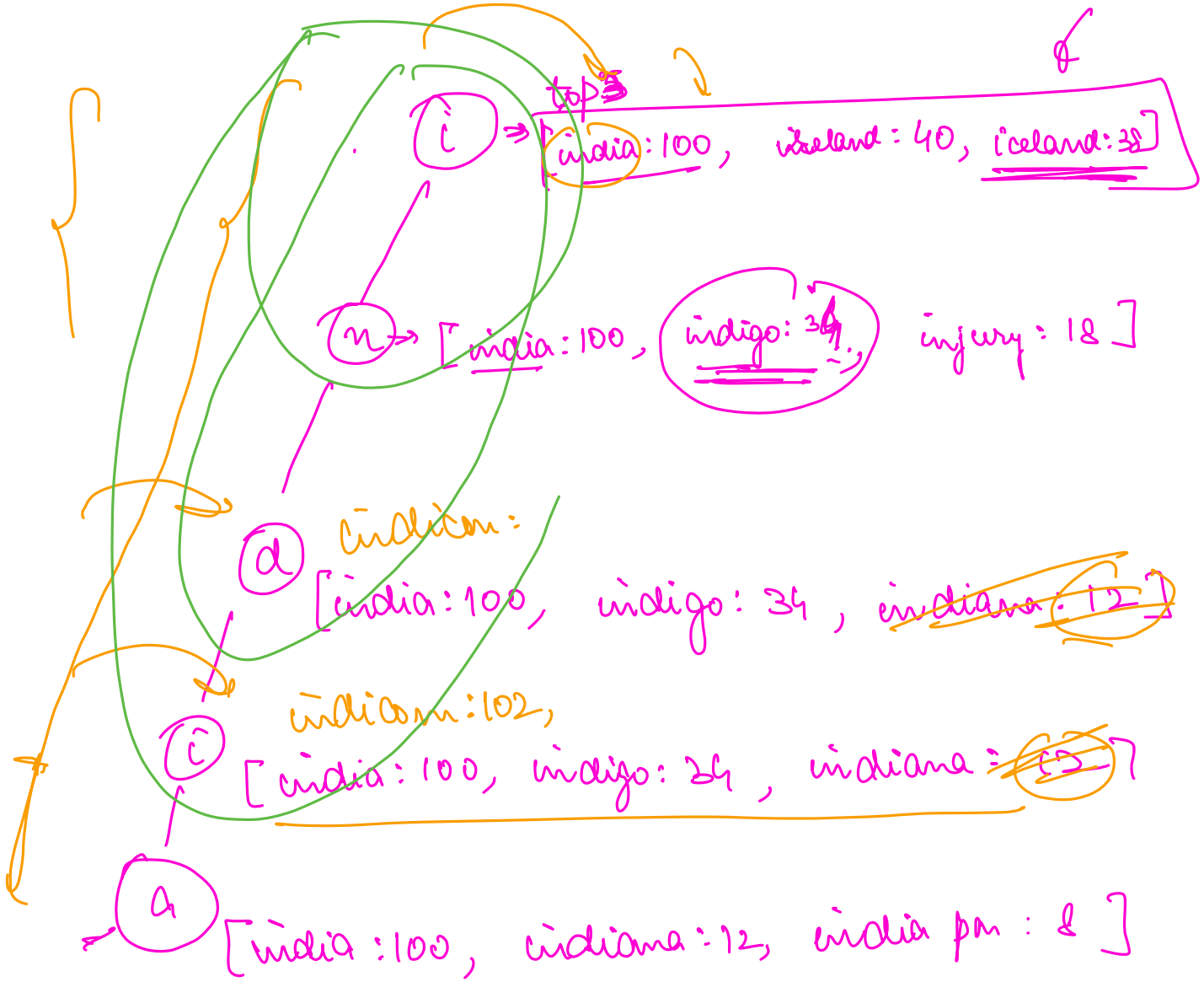
indi → 2

india → 2

26 x 26 x 26



26 x 26



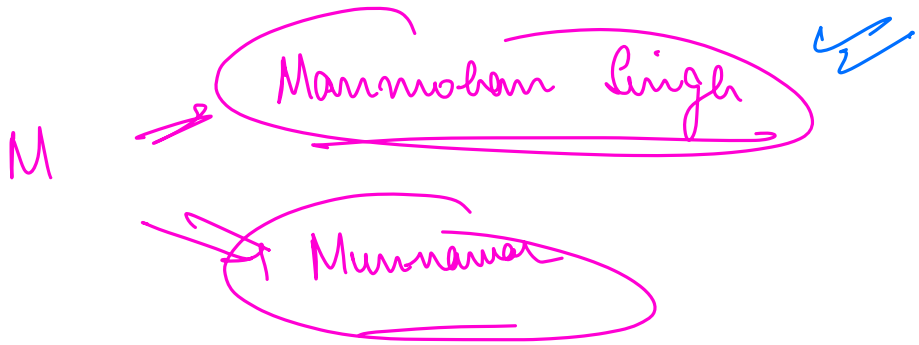
Someone searched indiacom

→ ~~india~~ 101

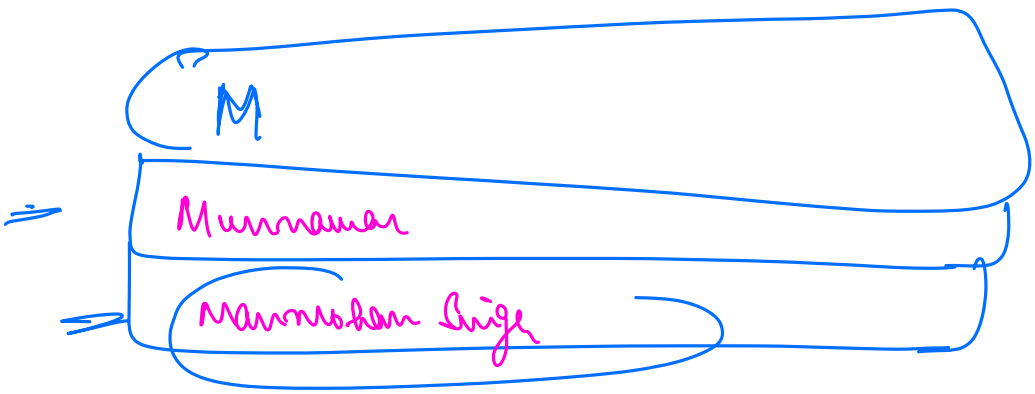
indiacom * 102

"a_ _"

Trending Words



⇒ across all time of BS Manmohan Singh will be searched more.



⇒ But because munnawar is now popular today, google wont want to suggest that.

⇒ Trending

every day at 12 midnight:
half the freq of everything

= Manmohan

T0: 10000 + 500

T1: 5250 + 500

T2: 3000 + 500

T3: 1750 + 500

T4 → 1125

Munnavar.

T0: 500 + 2000

T1: 1250 + 2000

T2: 1600 + 2000

T3: 1800 + 0

T4: 900

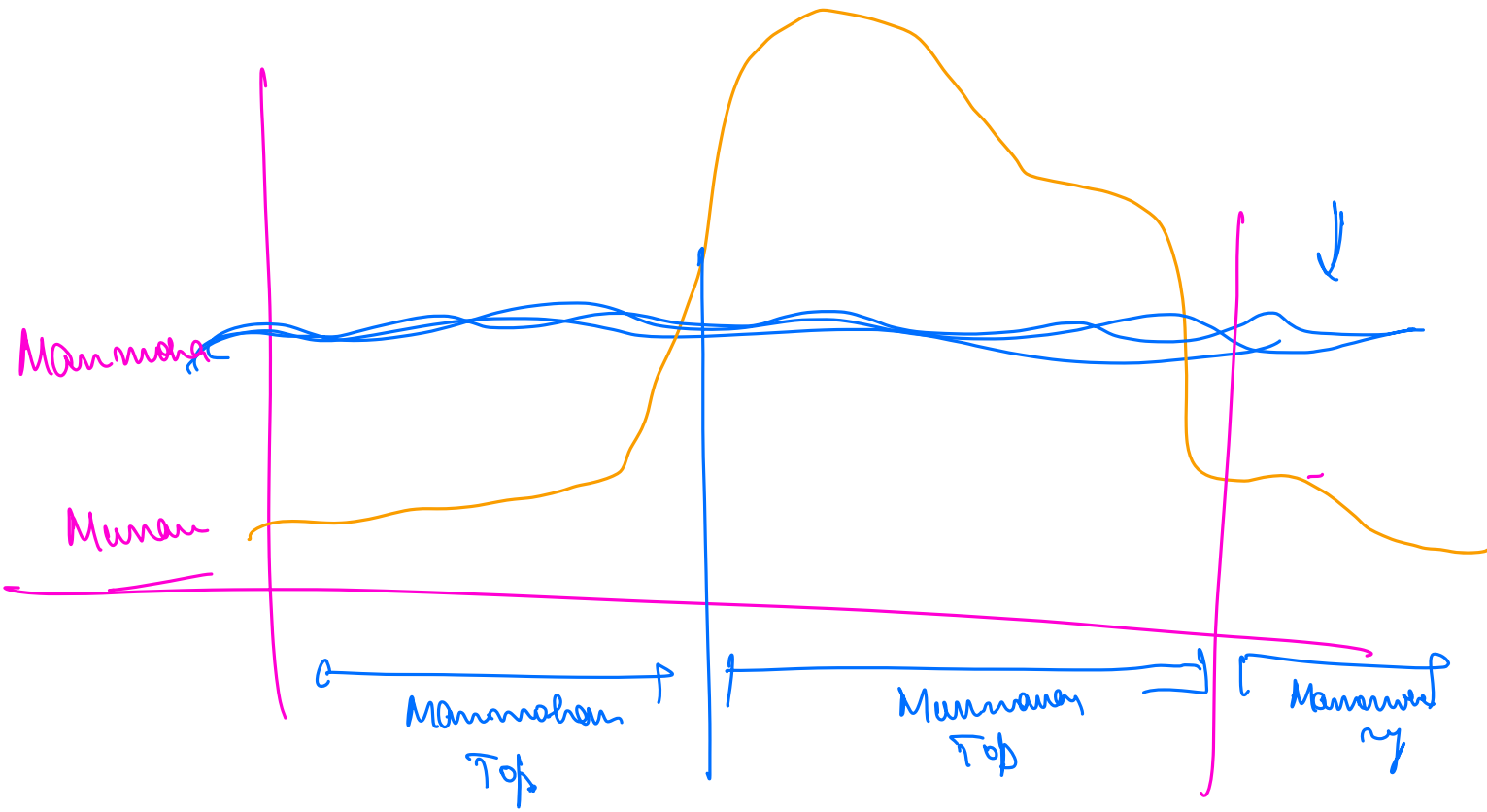
⇒ I am giving more weight to newer readers

T0: X X

T1: X/2 X/2 Y Y Y

T2: X/4 X/4 Y/2 Y/2 Y/2 Z ZZZ

T3: X/8 X/8 Y/4 Y/4 Y/4 Z/2 Z/2 Z/2 Z/2 AAA



Mongo

X Sharding Keys 26^26^26

Y Machines 3 Machines

Y <= X

1. Read the typed notes
2. Revision today and prev class
3. Write the Trial Code of Typeahead.
4. On a sheet paper put the complete flow of typeahead