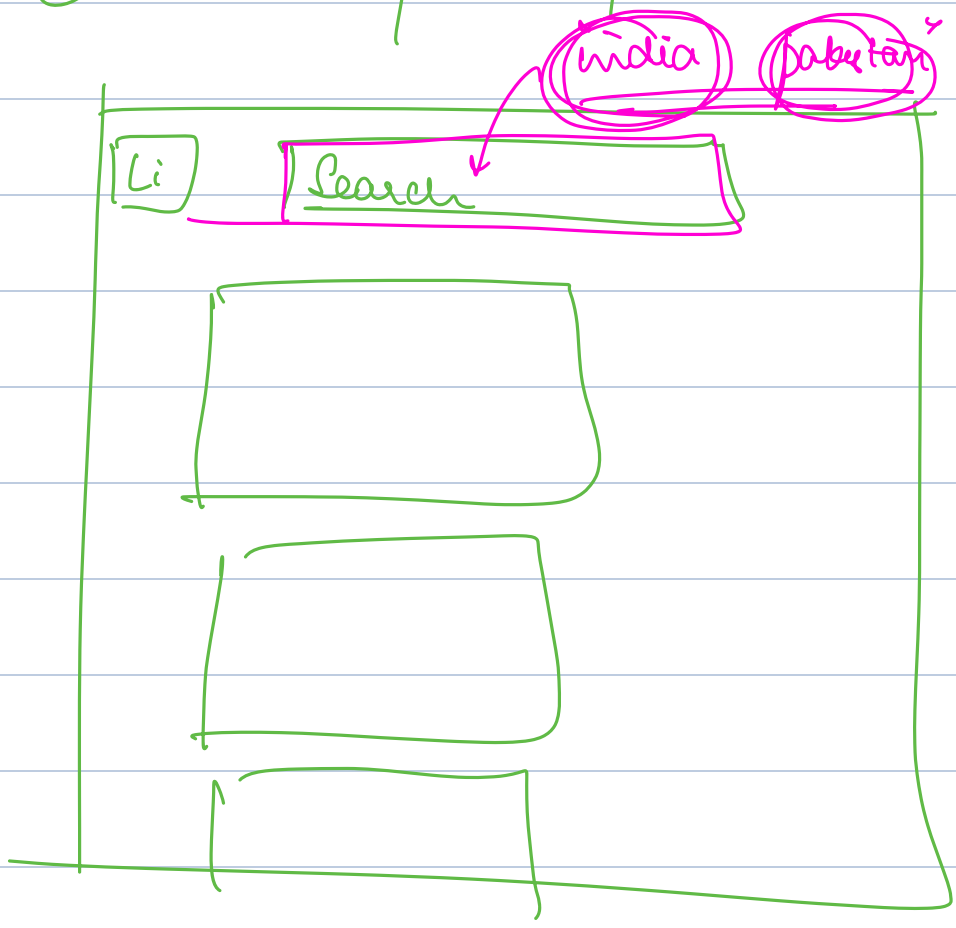
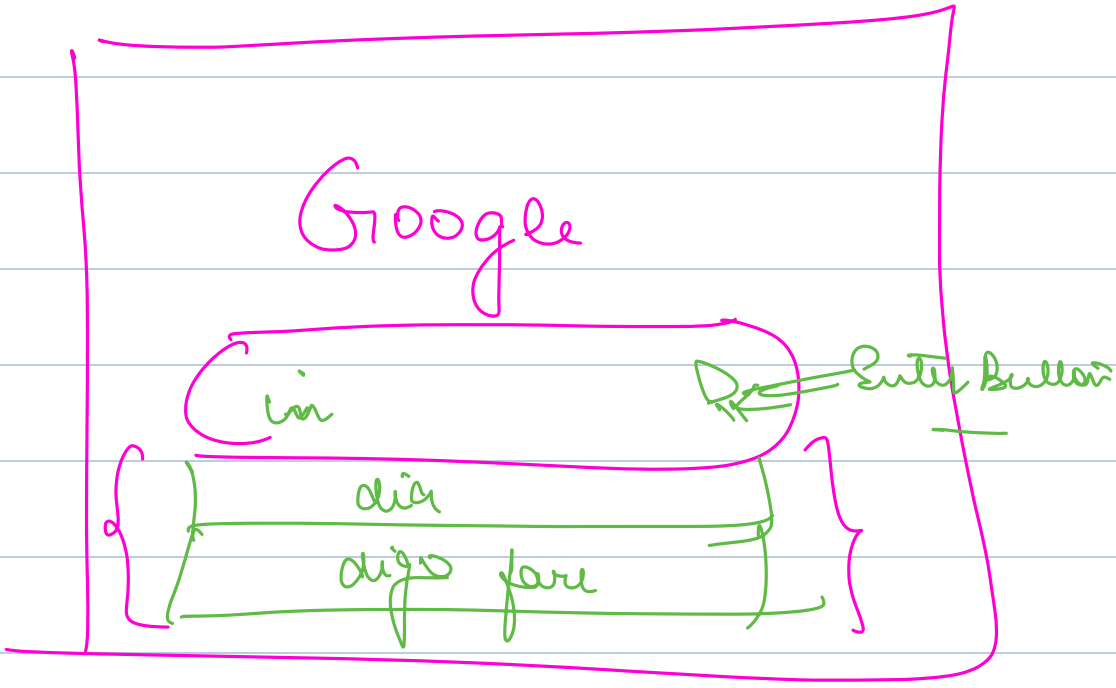
Elastic Search

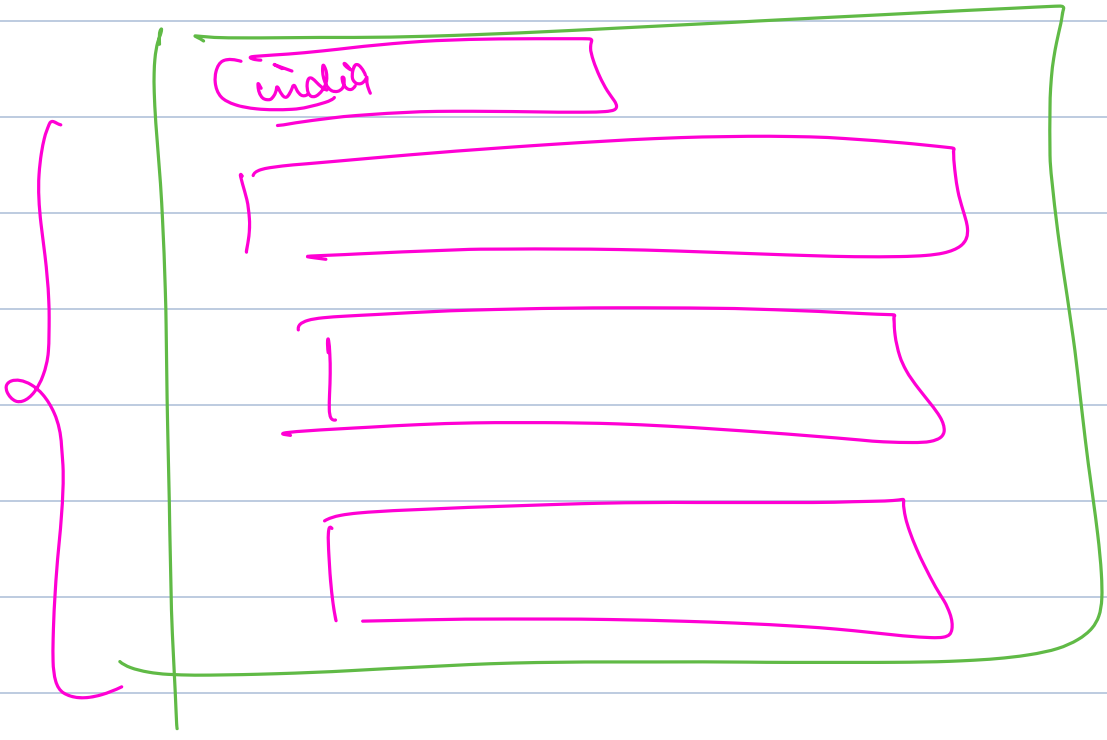
"Design Elastic Search"

SWE @ linkedin / Twitter / etc



Build a system that allows you to given a string query get relevant posts for that query.





'Implement a Search Service'

Search Service

Typahead Suggⁿ Service

posts (SQL DB)

id	createdAt	user-id	content
			- - - - - - - -

Search → India pakistan →

Ans)

```
select
    from posts
    where
        content like '%India pakistan%'
```

Wont work:

1.) Slow

indexes wont help either

2.)

relevant ✓ return ✗

recently there was a match b/w india and pakistan

Ans 2

select *

from posts

where

content like '% india %. pakistan %'

The pak team had recently come to play with India

relevant ✓

return ✗

Slow !! !

Ans 3

Select *

from posts

where

content like '%. india %'

and content like '%. pakistan %'

relevant ✓ ⟸ Correct !!!

But v.v.v. slow

O(N) ⟨ go through every row of post

for each post:

find string

KMP Algo O(N)

$10^9 \Rightarrow 1s$ ⇓ $O(N^2)$

$10^9 s$

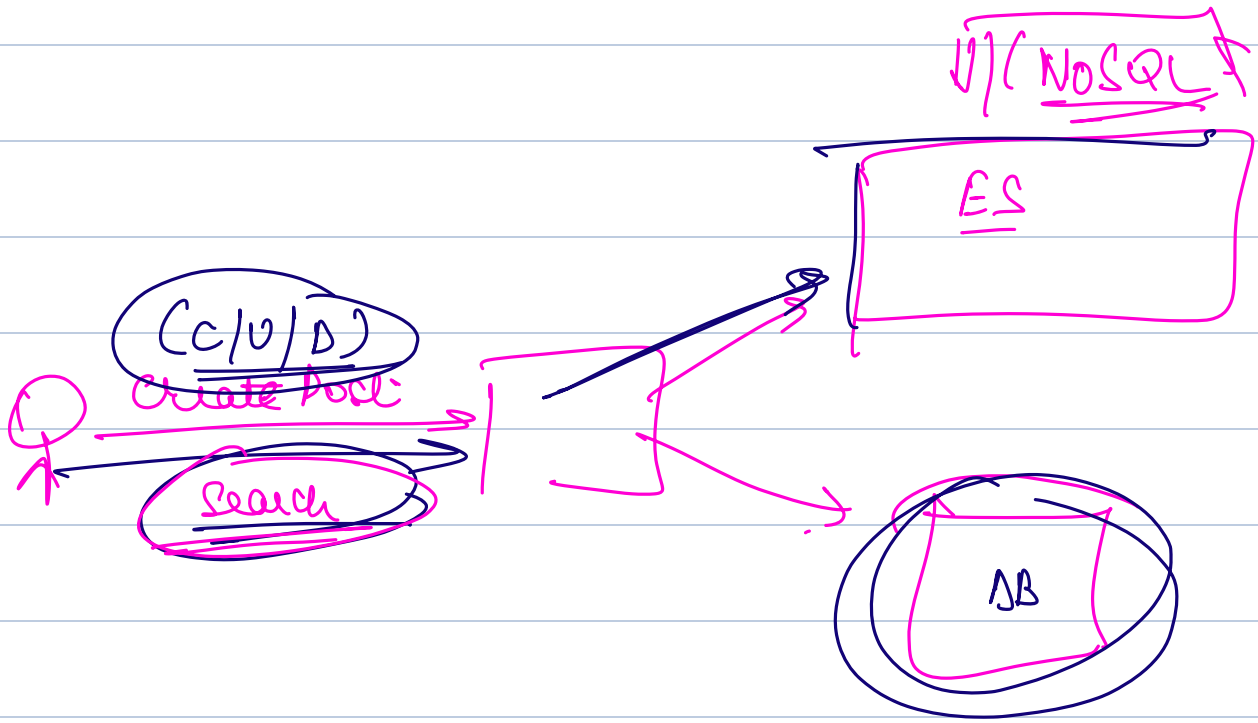
$N = 1B$

$N^2 \Rightarrow 10^{18}$ ⇒ Might take a day to compute !!

Elastic Search

↳ allow to handle search queries

- → filters
- → sorting
- → pagination



Design ES

Relevancy

→ presence of words in query

[india pakistan]

$O(N^2)$ 1) - Posts that contain India

2) Posts that contain Pakistan

$O(N^2)$

common between them (intersection)

What if:

(i) I precompute and store all posts for each word.

assume this thing exists

word	list<posts>
India	[1, 7, 11, 14, 19] M
Pakistan	[7, 14, 23, 29] N
Punjab	[13, 18, 19, 28]

Map

how would you have searched

India Pakistan

$l1 =$ map.get(india)
$l2 =$ map.get(pakistan)

$O(M+N)$

2 pts approach

This is actually what is used in all search systems, from Google to Elastic Search

20 GB

20 GB

CS: "Information Retrieval"

PageRank

Map $\Rightarrow$ Inverted Index

B.F. = went through posts to find words.

Map $\Rightarrow$ went through words to find common posts.

Thing to be Learned
(Post, Hashtag, Project, Topopening)

Search System

"The match with Pakistan today was a great relief for India"

Break sentence into words

Tokenization

Remove Stop words.

Convert words into root word

Lemmatization
Stemming

Inverted Index

map<word, list<Posts>>

India → Ind

1990s

IR → NLP → ML → ... → LLM

NLP via ML

NIR

ChatGPT
2020s

Lemmatization, Stem, Lem

visa policy of usa

V/S

visa policy usa

india vs pakistan

india pakistan

running

doc contains:

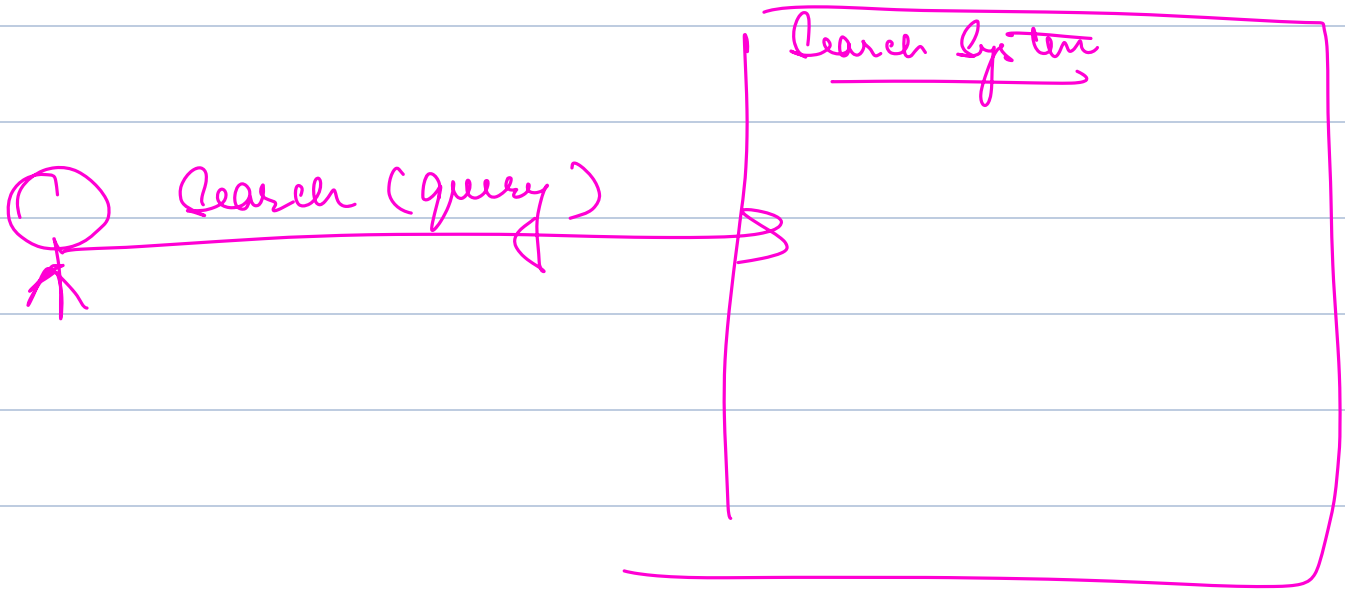
ran.

ran

index = Map

string	list (Posts)

```
ingest (post) {
    words = [post.split(" ")]
    for each word :
        if stop_words.contain(word):
            return;
        root = getRoot(word)
        index.get(root).append(post.id)
}
```




```
Search (query) {
    words = [query.split()]
    ans = null

    for each word:
        if stop(word): continue
        root = getRoot(word)
        matchingPosts = map.get(root)
        if ans == null
            ans = matchingPosts
        else:
            ans = intersection(ans, matchingPosts)
```

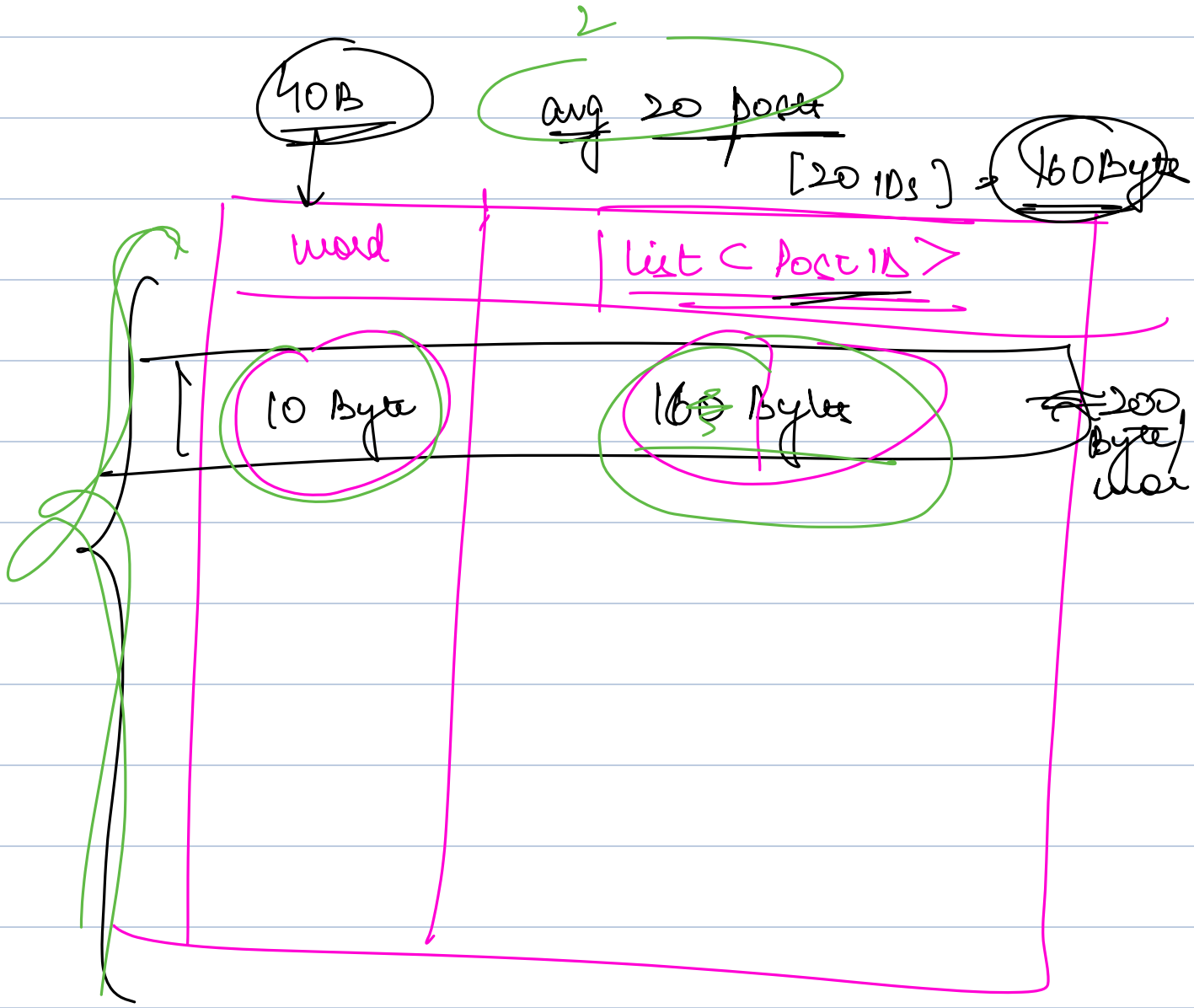
go to next word

posts that contain word

"InDiA". lower() ⇒ india

||

"INDIA". lower() ⇒ india



LinkedIn Post

400 M DAU ⇐ daily active users

1% post in a day

⇒ 4M posts in a day

⇒ 100 words / post

⇒ 400 M words

~~10% unique~~ ⇒ 40M words

⇒ 5 years ⇒ $40 \times 10^6 \times 5 \times 400$ words

$\approx 80 \times 10^9$ words

⇒ 80B words

50% unique ⇒ 40B words

avg len of word = 10 char ⇒ 10 Bytes

⇒ 40×200 Bytes $\times 10^9$

$\approx 8 \times 10^{12}$ Byte

⇒ 8TB

⇒ Sharding

How will we shard data

⇒ what will be the sharding key

default access key

Data

K – V

Word	List<Post ID>
	M1
	M2
	M1
	M2
	M2
	M3
	M1

Sharding Key ⇒ word

0th → word

0th 2 ⇒ post ID

get Post (word)

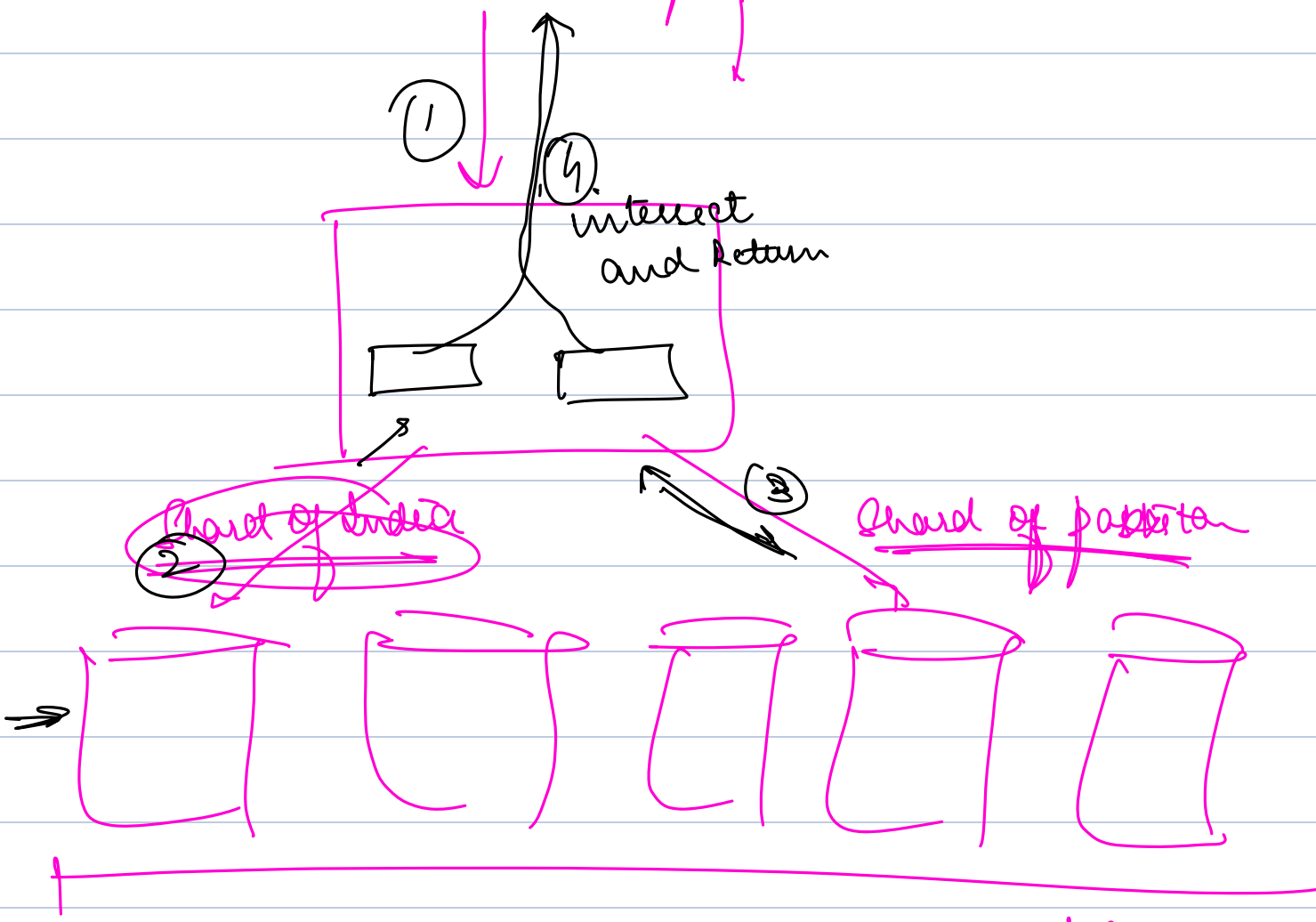
get Posts (word)

get Post (u, id)

Sharding Key

OPTION 1 : WORD

Search (india pakistan)



How many shards

Less # of Cra

Search | min (#words , # DB Shards)

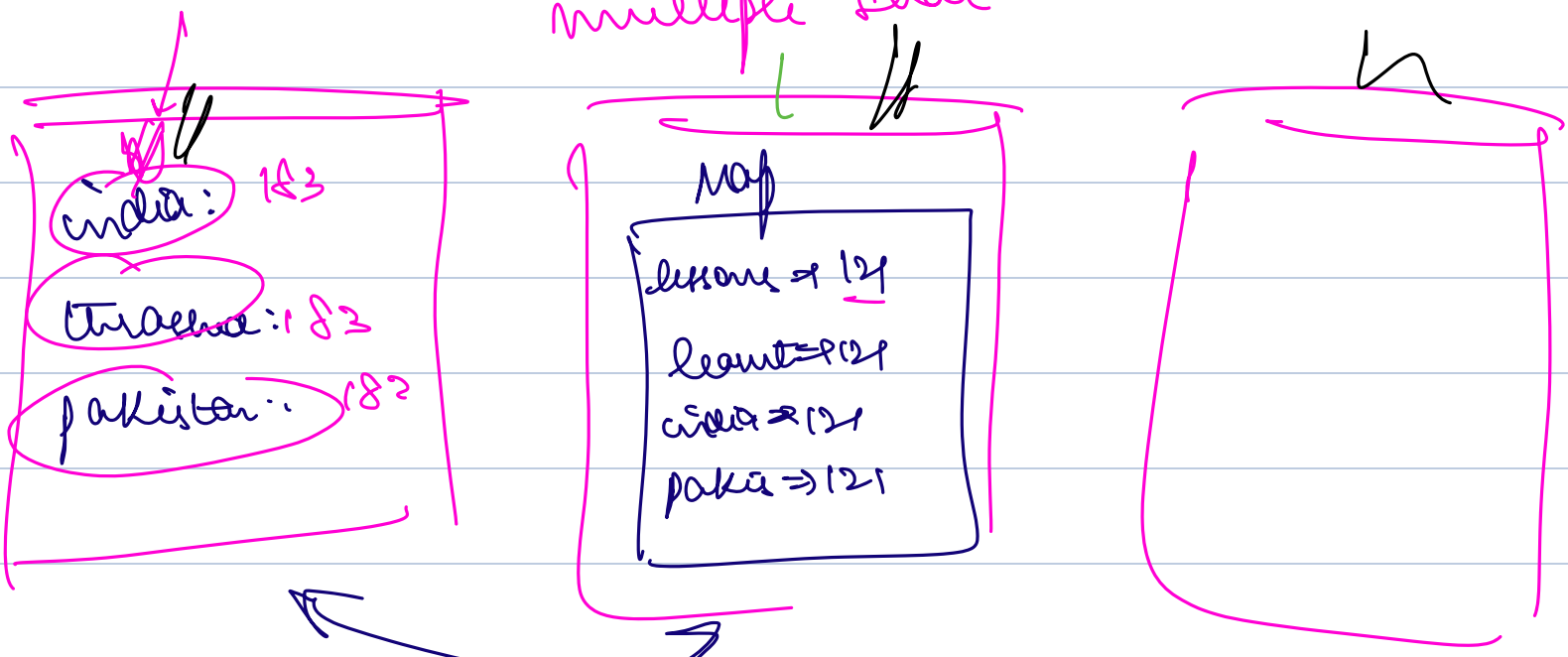
10 3

Ingest | min (#words , # DB Shards)

OPTION 2 : POST ID

Ans

Same word can be in multiple shards



Shards

121

These are the 5 lessons I learnt from last matches of India and Pak

183

India Chinese Pak

INGEST ⇒ 1

SEARCHING ⇒ ALL SHARDS

"india pak"

(will have to)

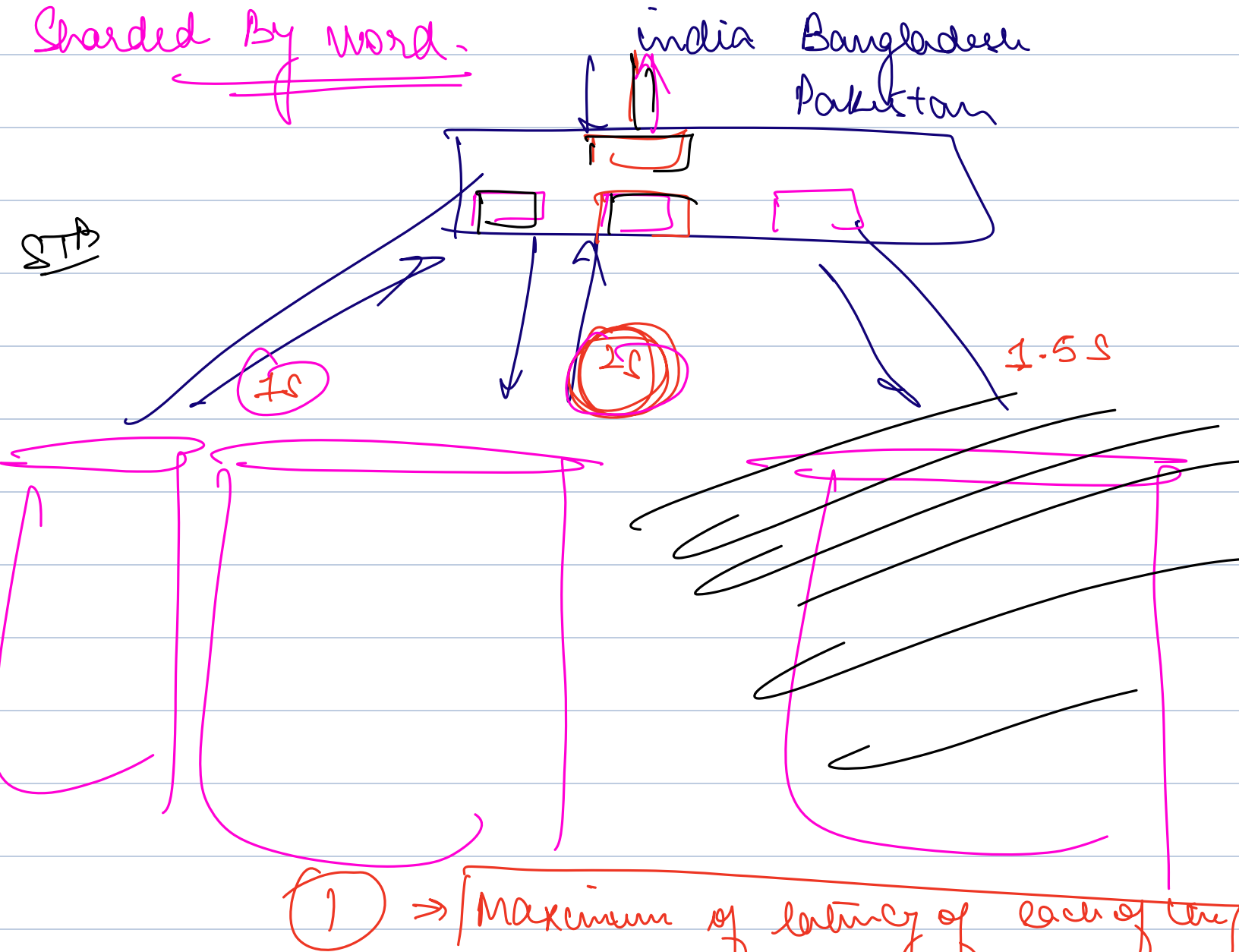
Why ES uses post-id as sharding key

→ return all matching results

CP/S A → Availability

CP/S low-latency → Low latency

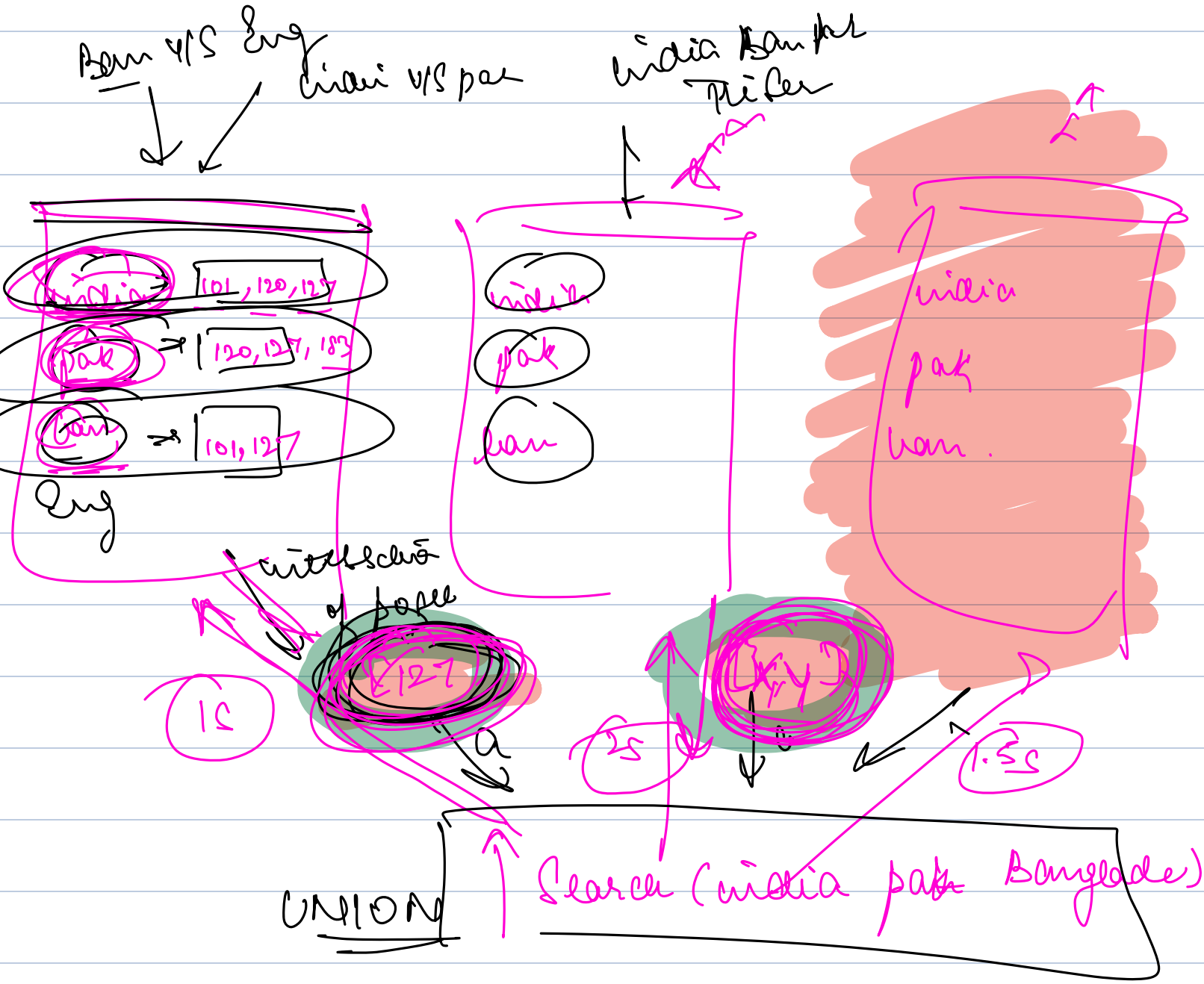
Sharded By Word.



1 → Maximum of latency of each of the

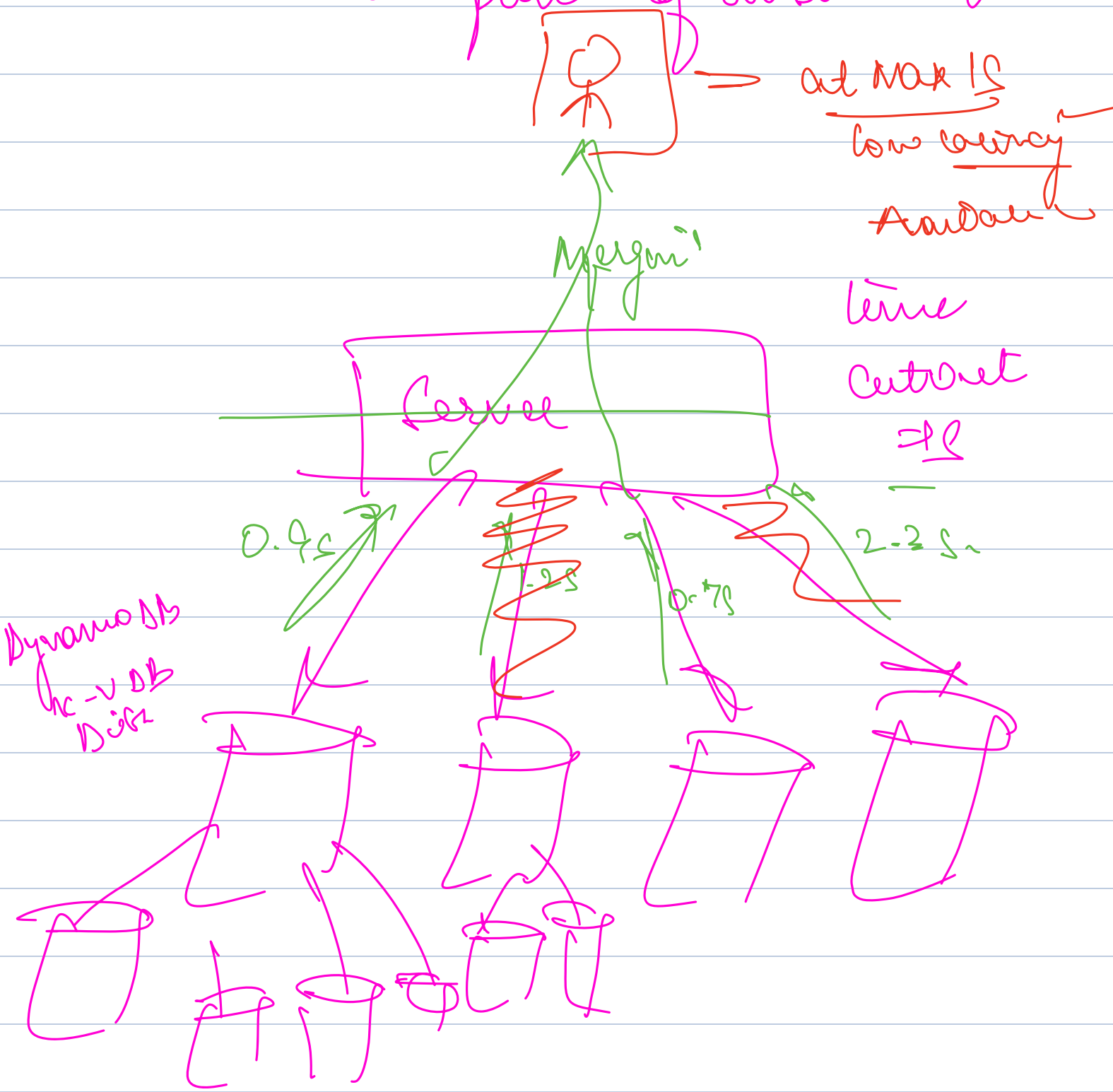
(2) ⇒ Shard of a word is down
⇒ will we be
able to retrieve
resp.

Shard By Post



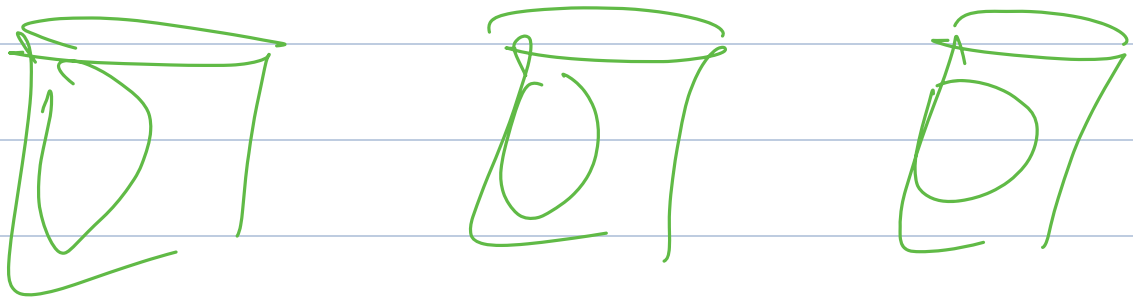
a. append (b). append (c)

→ each shard is returning a part of answer



1B posts

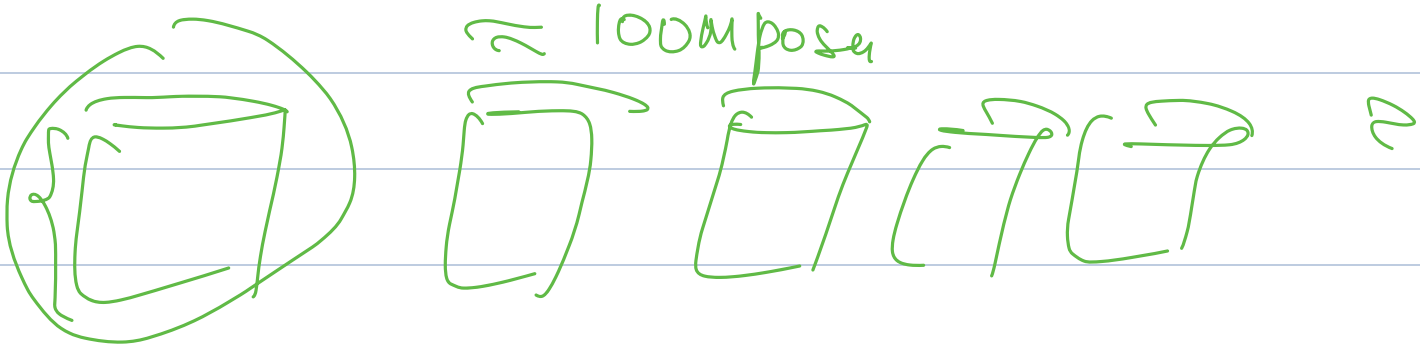
~300M ~300M posts ~300M



⇒ 210 Crore

50M posts

≈ 100M posts



≈ 150

≈ 50 M posts

× 100 more

$50 \times 10^6 \times 100 \Rightarrow 5 \times 10^9 \Rightarrow 5.0 \times 10^9$ B

150 GB

- Simply
- Use OG
- Twitter Bane

GET /files/folder_name/file_name.txt

404