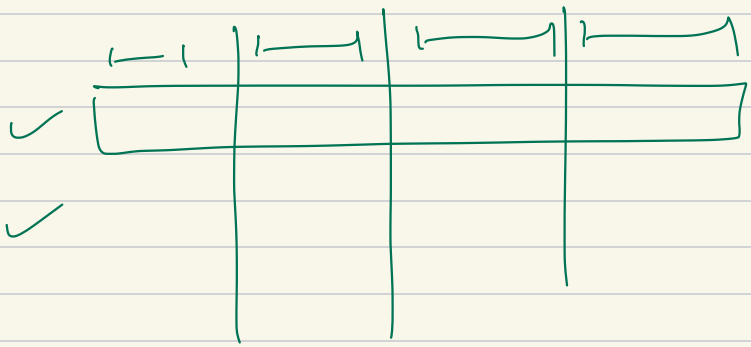
11/Dec/2023

NoSQL Internals



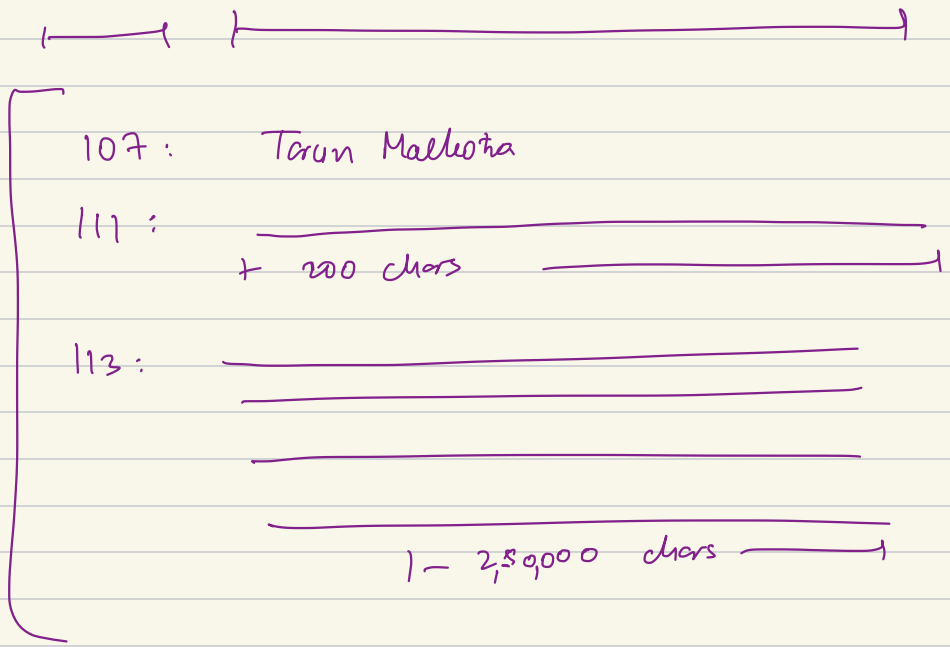
10th

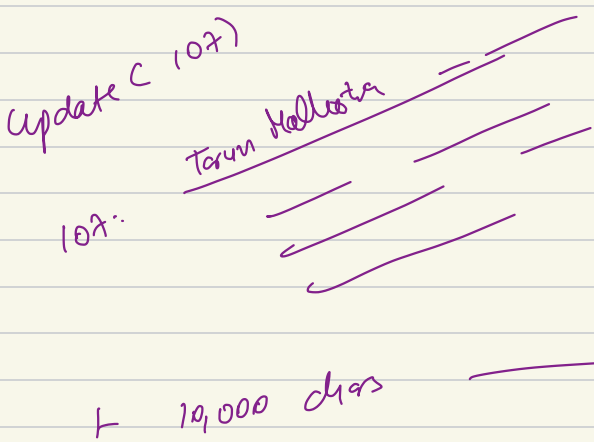
B+ Trees

SQL → optimized for Reads + Writes

Because of fixed sized contiguous rows ☺

Key value


107: Tarun Malhotra
111:
+ 200 chars
113:
1 – 2,50,000 chars

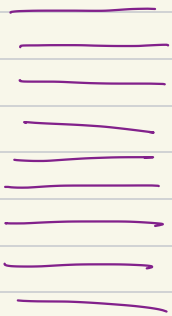

update C 107)
Tarun Malhotra
107:
+ 10,000 chars

detour — Redis

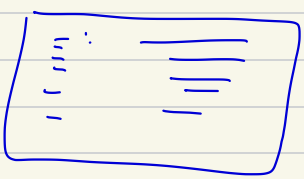
① ~~WAL~~ (Write Ahead Log)

AOF

WAL ≡ AOF
(Append only File)



② Snapshots of data

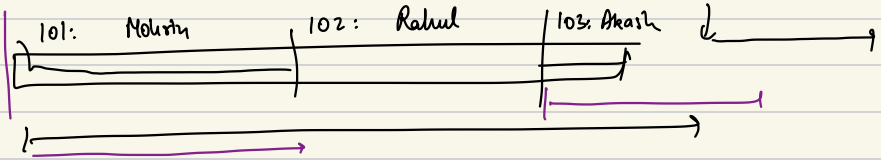


snapshot is saved in the disk

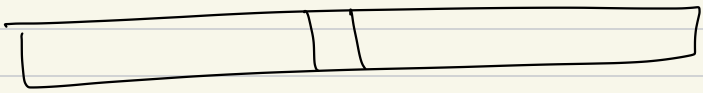
Iteration 1 :

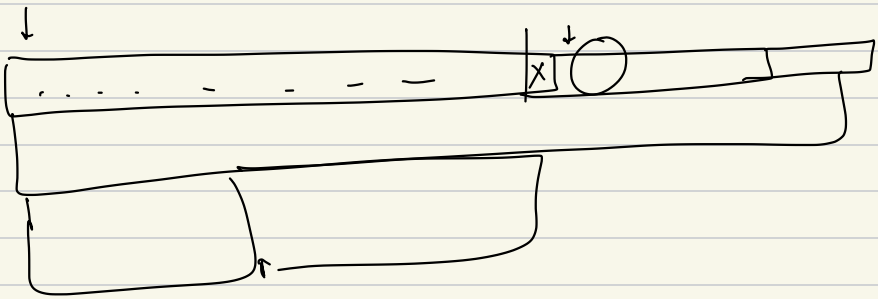
101 : Mohsin
102 : Rahul
103 : Akash

Search : O(n)
Read : O(n)
Write : O(1) + O(k)



Update : O(N) + O(N) = O(N)





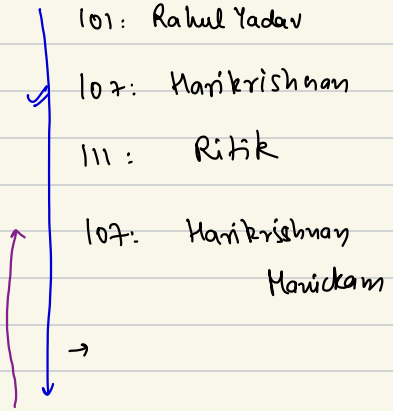
Iteration 2:

T1 : 101 → Rahul Yadav

T2 : 107 → Harikrishnan

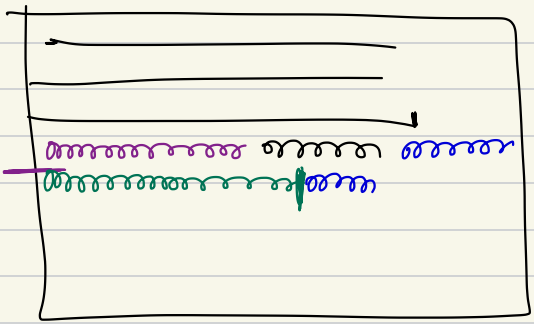
T3 : 111 → Ritik

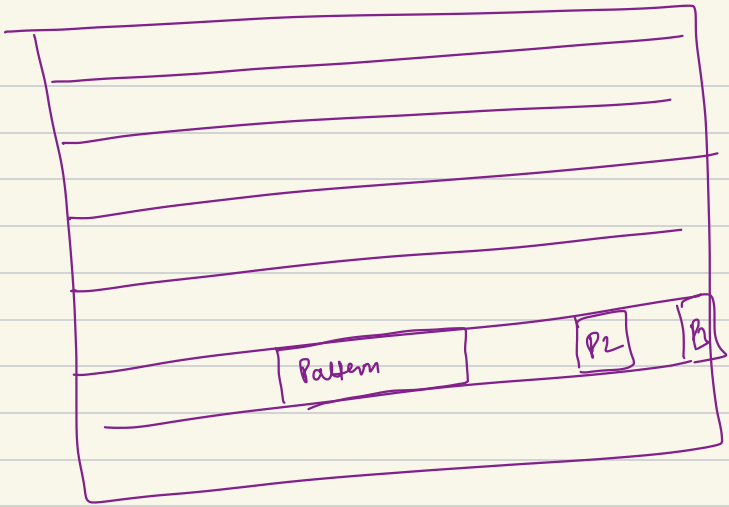
T4 : update 107 → Harikrishnan Manickam



WRITE | UPDATE → O(1) ☺

T5 : Read (107)



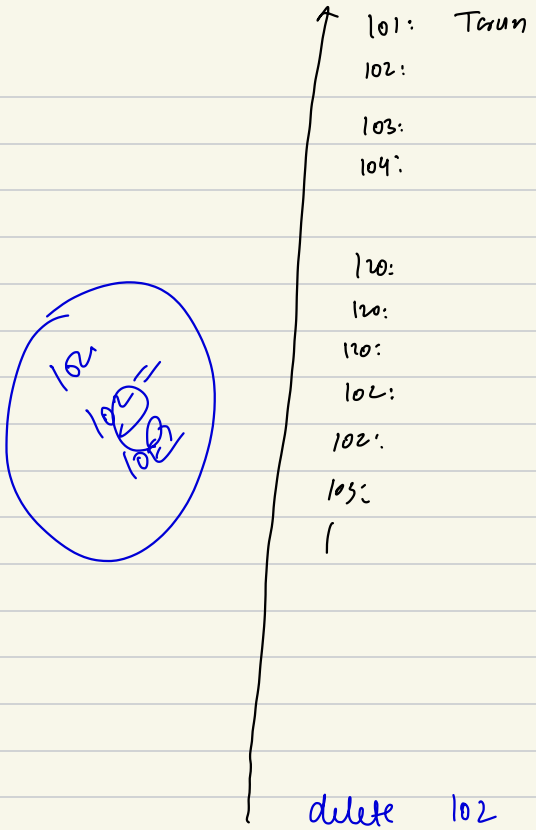


N → no of unique keys

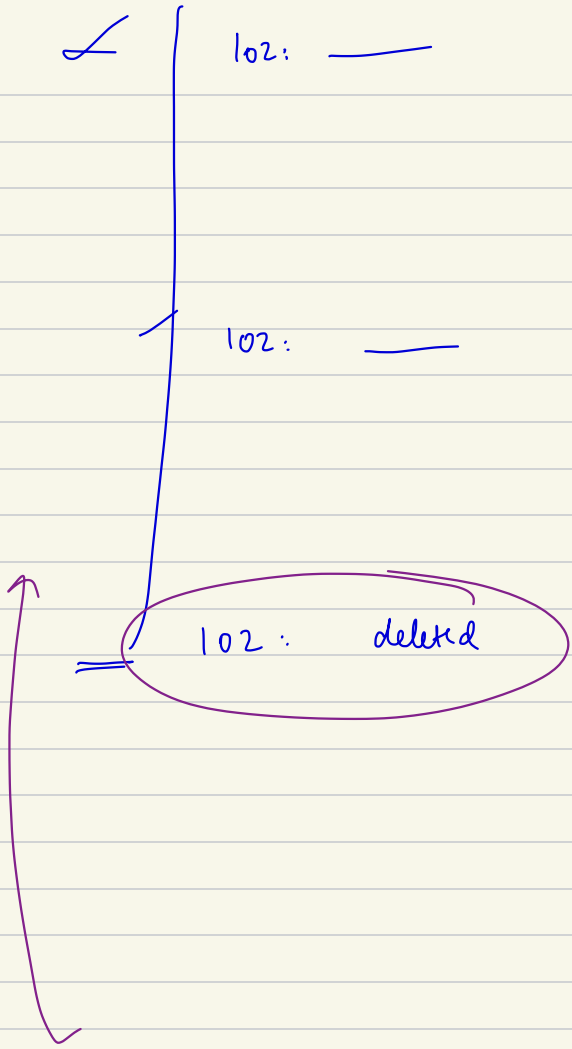
O(1) Write → at the EOF

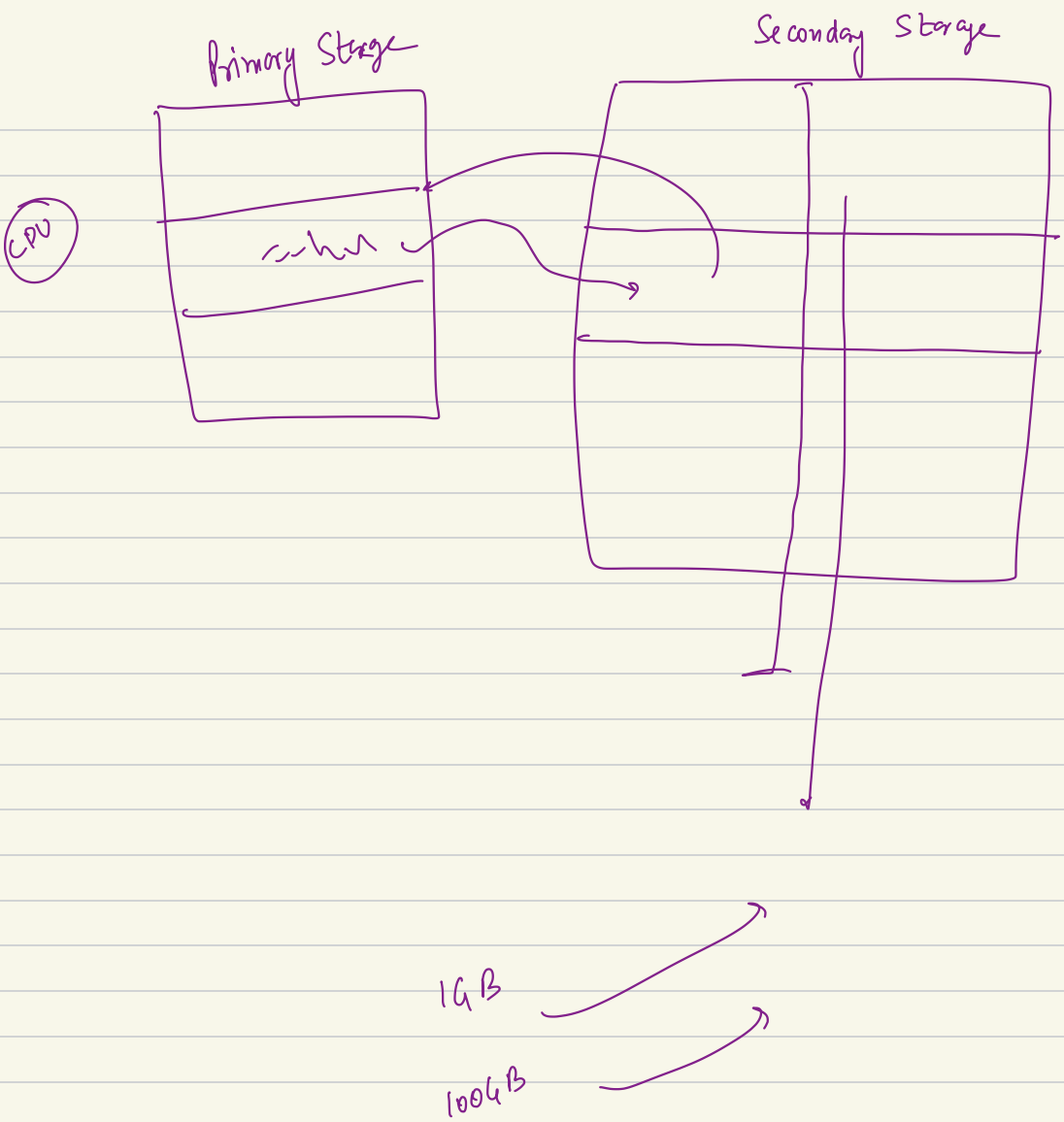
O(1) Update → No search; simply write at EOF

~~O(1)~~ Read → start reading from end and stop at the first key match



Writes	are	great	O(1)
Updates	are	great	O(1)
Reads	are	super bad	O(N x avg duplicates)
Storage space	is	super bad	O(N x avg duplicates)

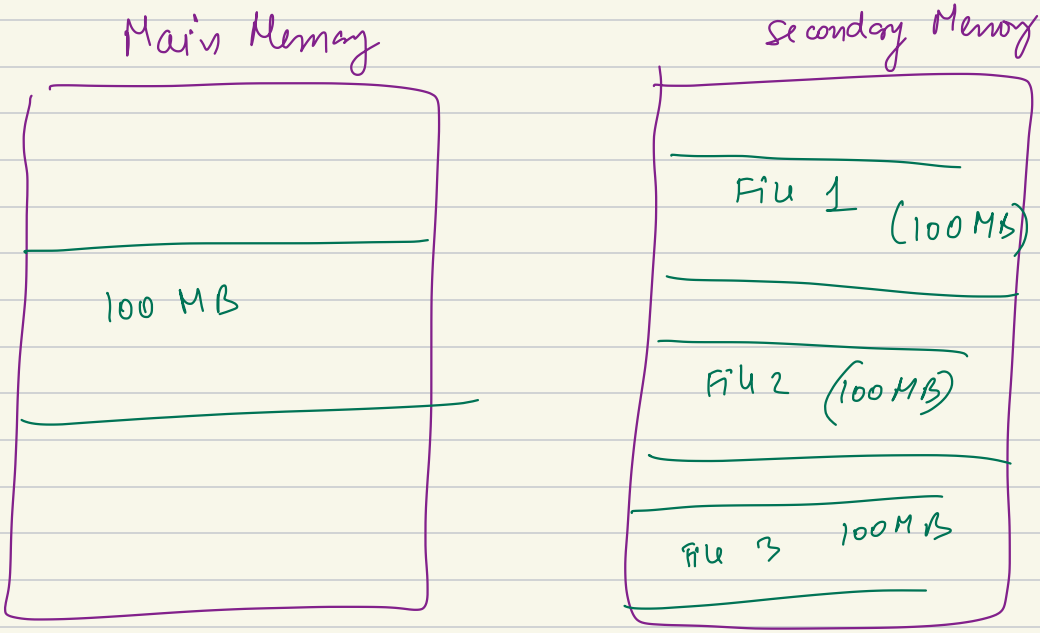

102:
102:
102: deleted

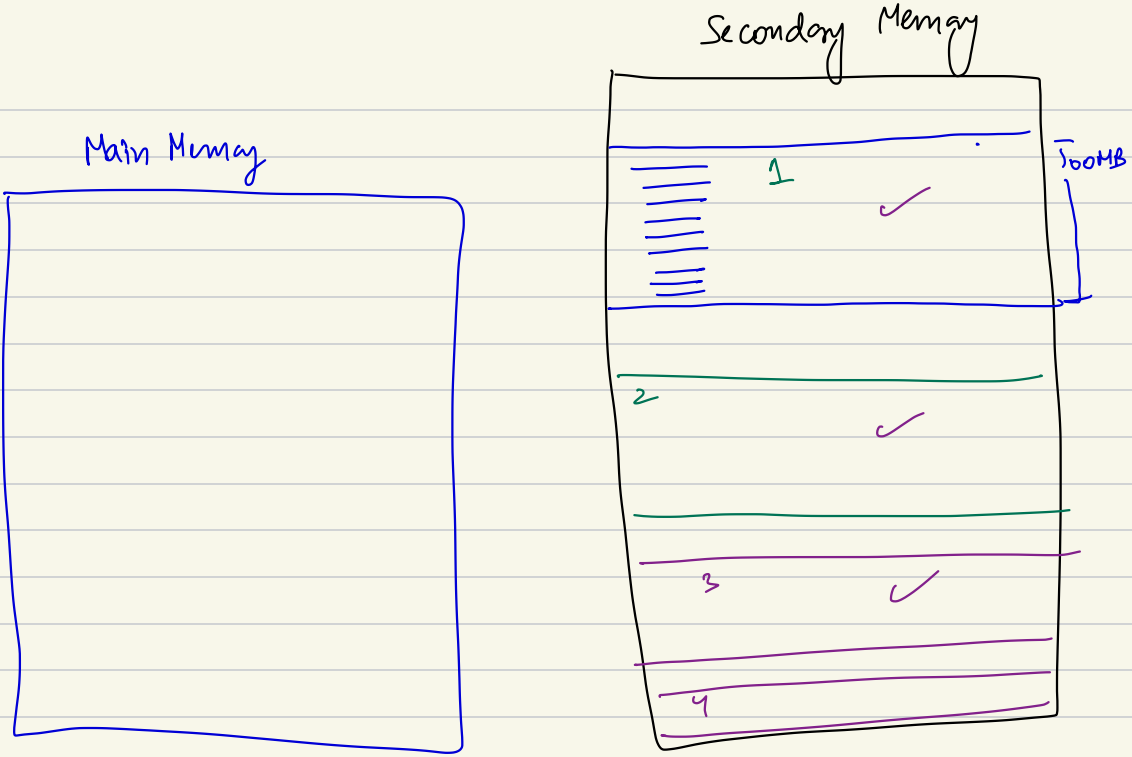


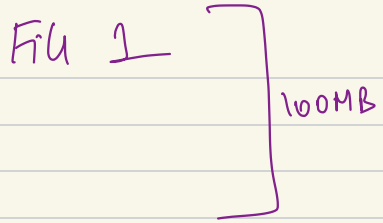
1GB

100GB

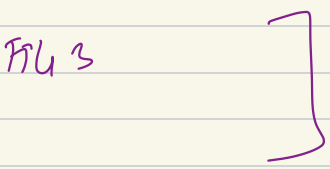
Given that your file which stores your database will keep on becoming bigger and bigger within no time it will become big enough that it can't be loaded to main memory in 1 chunk ☺


Main Memory
100 MB
Secondary Memory
File 1 (100 MB)
File 2 (100 MB)
File 3 100 MB

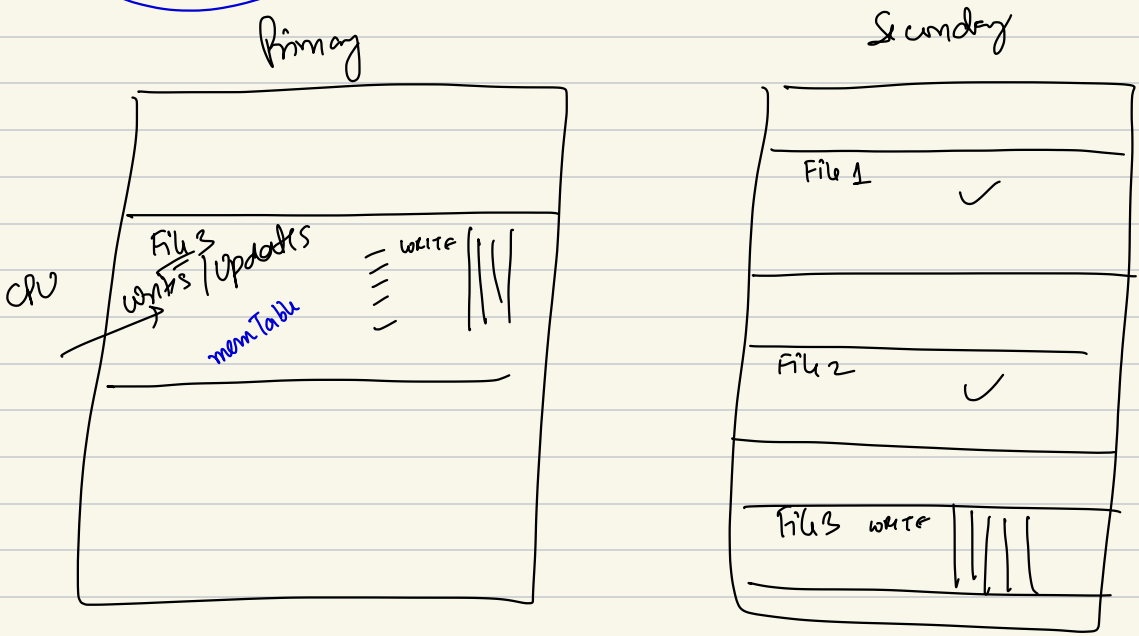

Secondary Memory
Main Memory
100MB
1
2
3
4

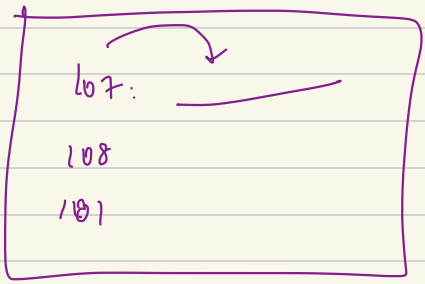

File 1
100MB


File 2
100M


File 3

ITERATION 3


Primary
Secondary
CPU
File 3
writes
updates
WRITE
memTable
File 1
File 2
File 3
WRITE



107: _____

108: _____

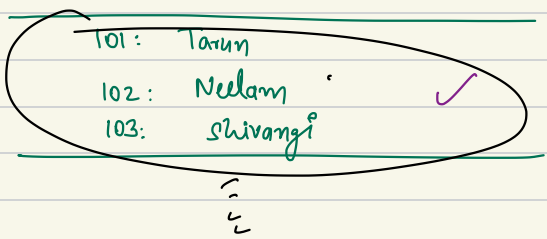
101: _____

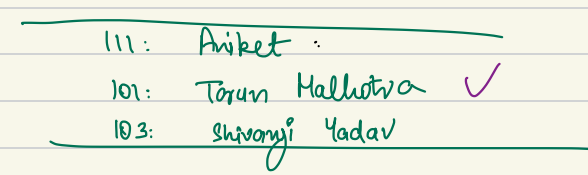
107: _____

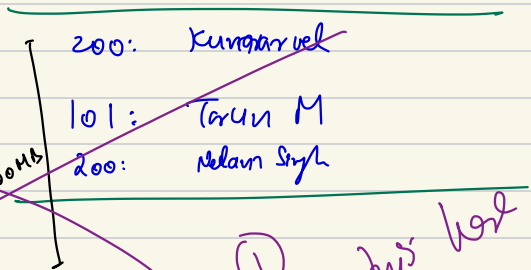
Iteration 4

Primary Memory

Secondary Memory







② safely delete

① dump his load

Primary

Secondary

(1)

memTable

Why NOT have this as a sorted map

(3) delete backup file

File 1 Backup

101 : Bhagwan

111 : Narendra

101 : Bhagwan Singh

Serialise this memTable and dump it (2)

File 1

101 : Bhagwan Singh

111 : Narendra

File 2 Bkp

Sorted String Table

SSTable

Iteration 4:

All the chunk files stored on the secondary memory are

1. without duplicates inside the same file
2. They are sorted on keys 🙂
3. All of them are ≃ 100 MB in size 🙂

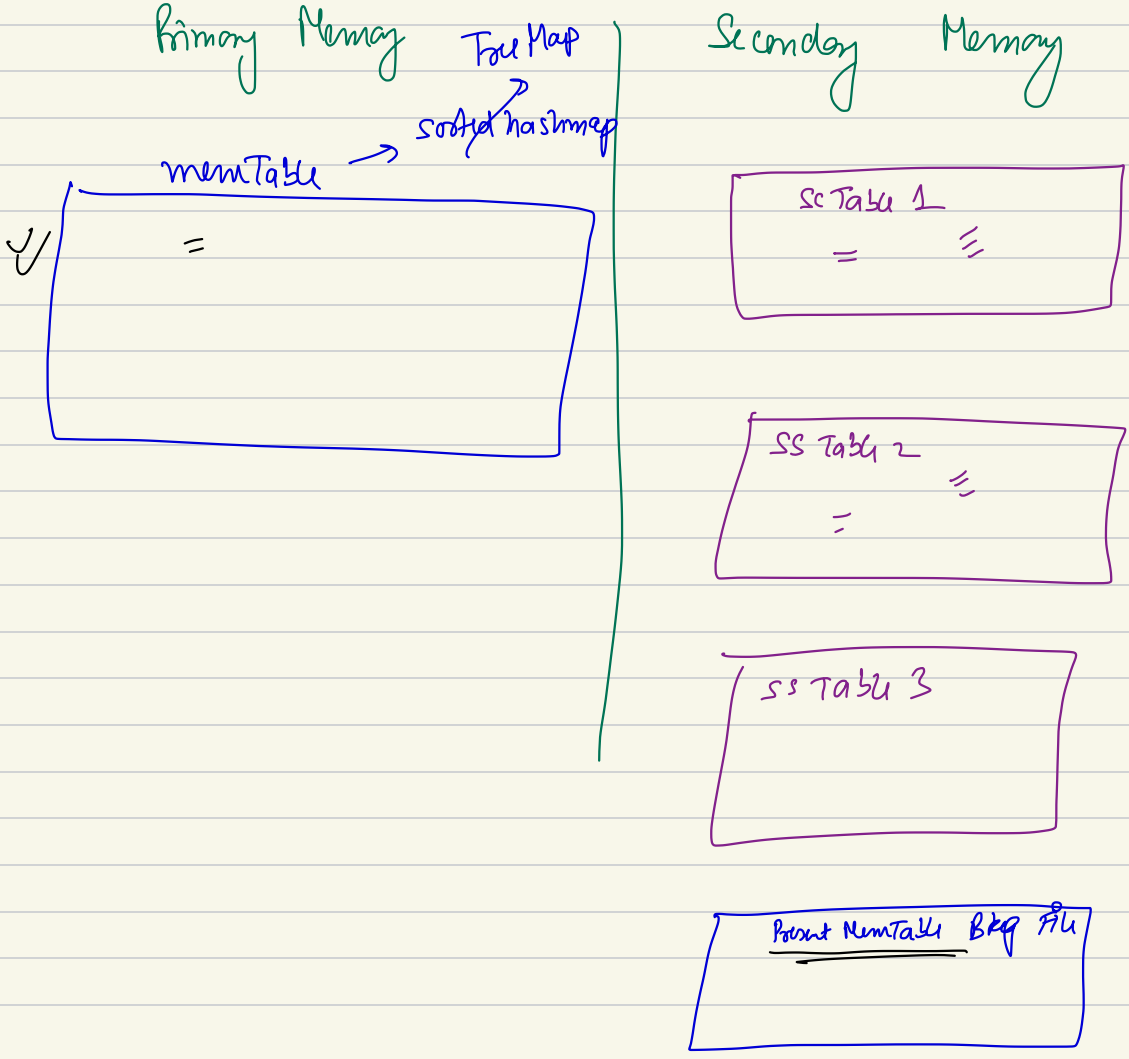
1. Duplicates can still exist between SSTables 😐 😐
2. Only within a SSTable keys are sorted, not b/w 2 different SSTables 😐 😐

Break. 10:40 PM – 10:48 PM 🙂

MemTable SSTables

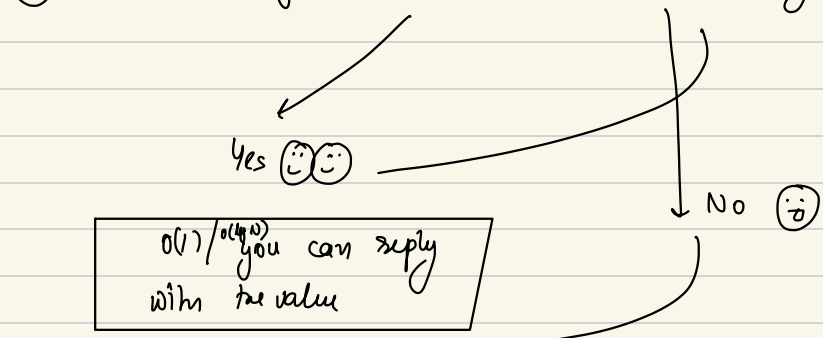
How Reads will happen?

Read (107) ⇒

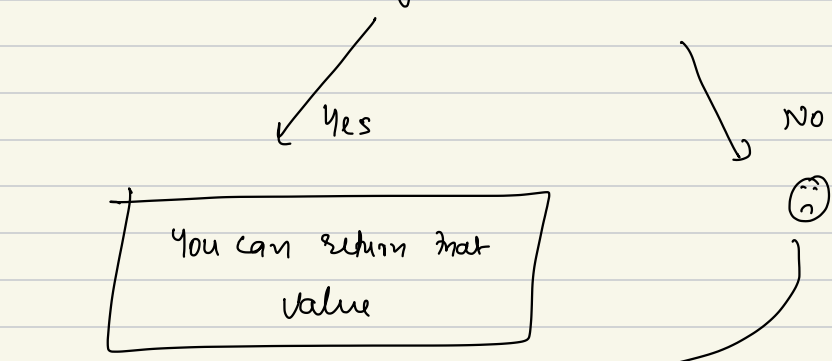


Read (key)

1. Check if memTable contains the key



2. Look into the latest SSTable [ex SSTable 3] and see if key is present there.



3. Penultimate SSTable (SSTable 2)

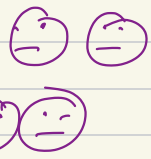
STEP 2 might have to be performed recursively.

① Updates are great

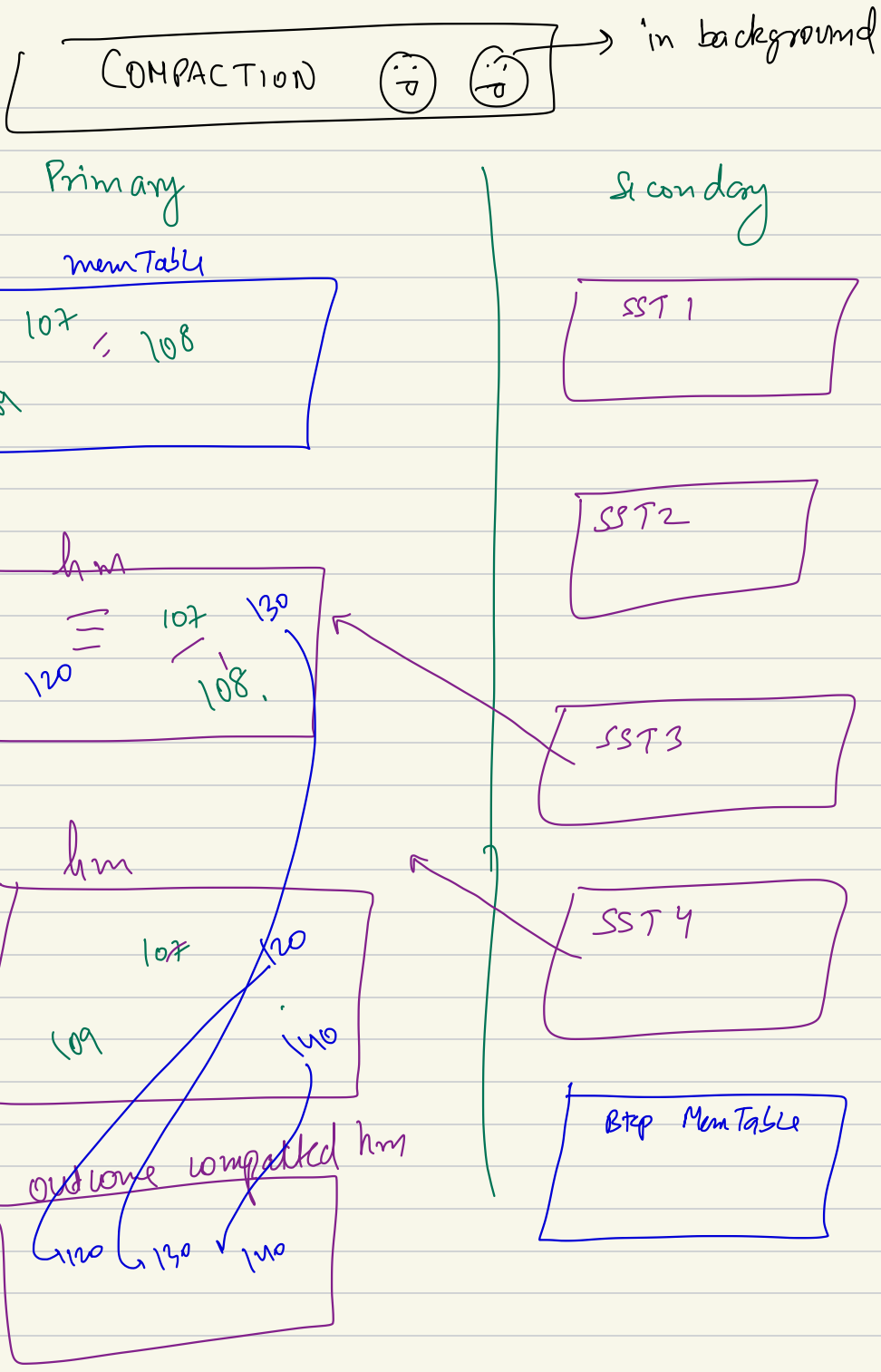
② Writes are great

③ Reads - Best case scenario is great.

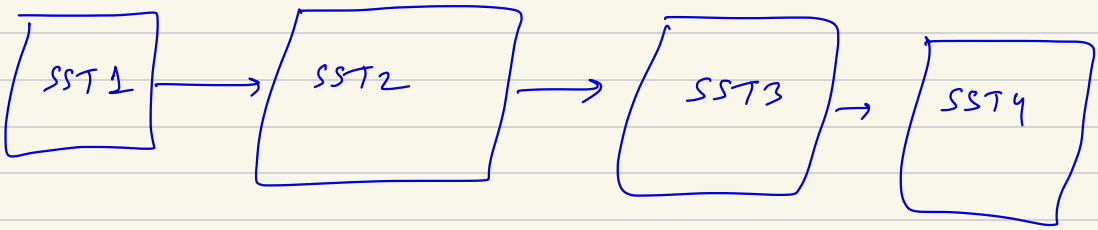
- Avg case scenario is GOOD

- Worst-Case scenario is Very Bad 

④ Space utilization is also Bad 

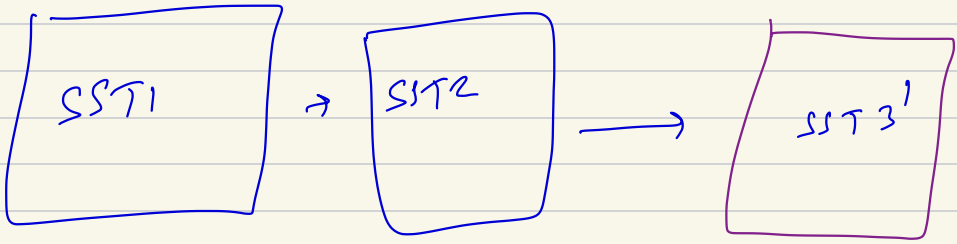

COMPACTION
in background
Primary
Secondary
memTable
107
108
109
SST 1
SST 2
SST 3
SST 4
hm
120
130
140
Btp MemTable
old one compacted hm

Imagine SSTables as a LL



Pick 3 + 4

Compact

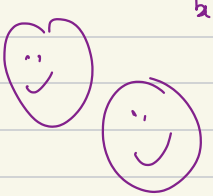


ITERATIONS: includes compaction

1. Updates are great ✓
2. Writes are great ✓
3. Reads - Best case scenario is great. ✓
   - Avg case scenario is GOOD ✓
   - Worst-Case scenario is ~~Very Bad~~ slightly less bad
4. Space utilization is also Bad

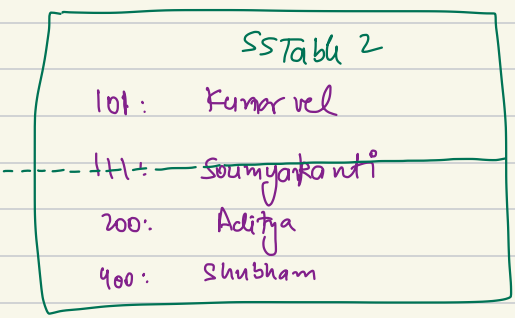
improved this a lot

because now we are removing duplicates all throughout using the background process of compaction.

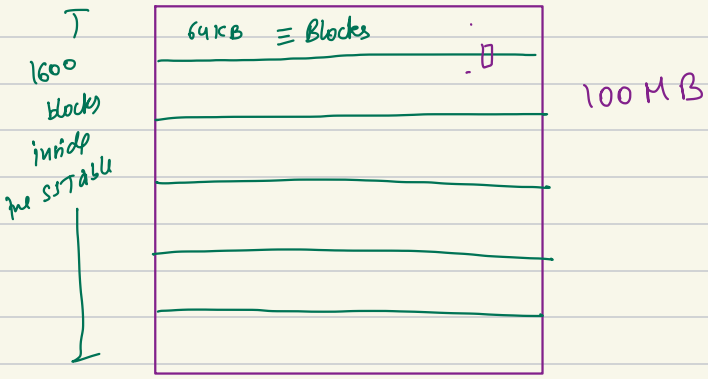


Indexing - Using Binary search to make Read way better....

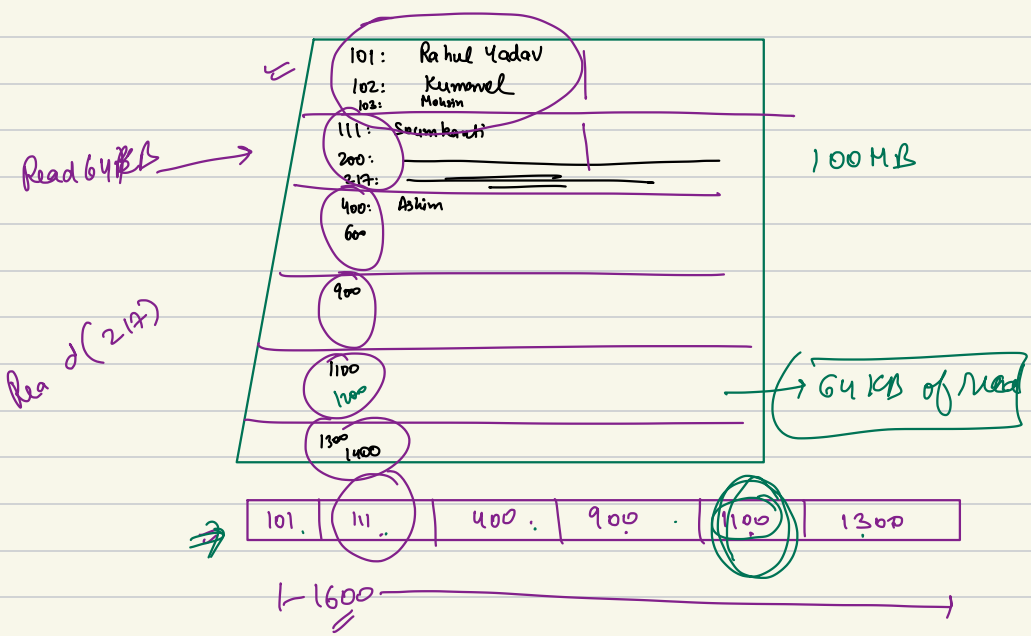
Why binary search can't be applied directly?



☆ we can't apply B.S directly as elements are NOT of equal size



$$\#\text{ Blocks} = \frac{100\text{MB}}{64\text{KB}} = \frac{1.6 \times 1000}{} = 1600 \text{ Blocks}$$



Read Algorithm

Read (700)

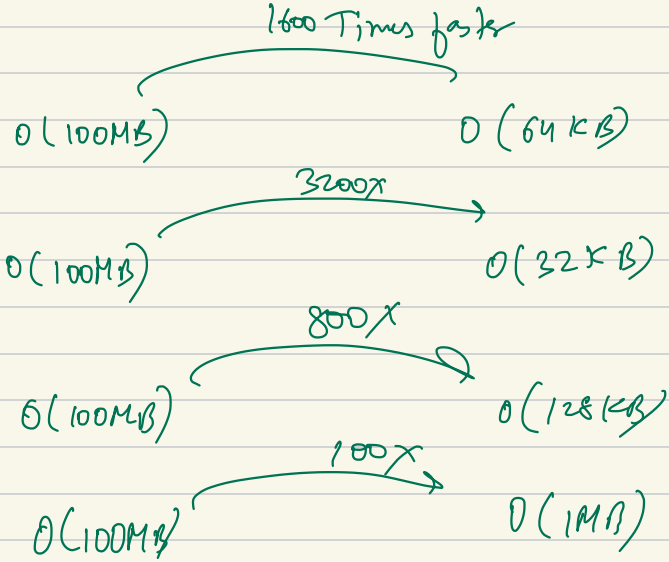
1. Mem Table — O(log N)
2. SS Table last — Repeated X

O(100 MB) → O(64 KB)

1600 Times fast

Block Size

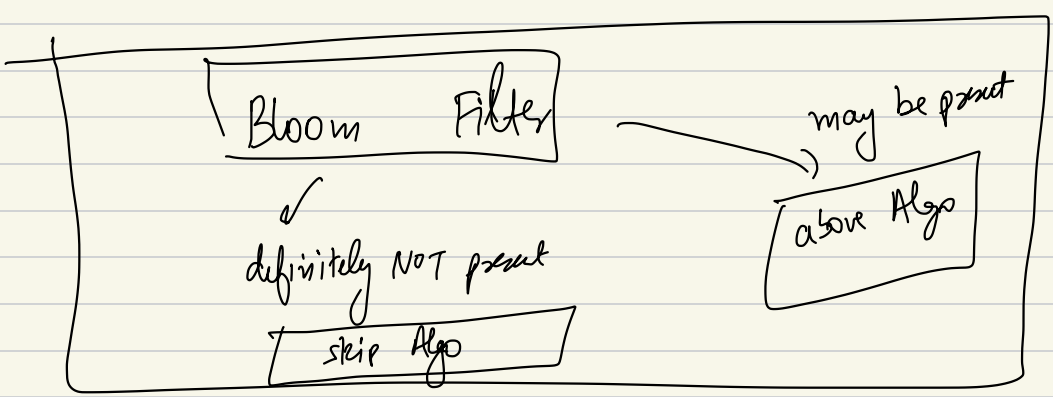
- ✓ 64 KB
- ✓ 32 KB
- ✓ 128 KB
- ✓ 1 MB



Finally

1. Updates are great
2. Writes are great
3. Reads - Best case scenario is great.
   - Avg case scenario is GOOD
   - Worst-case scenario is ~~Very Bad~~ Good
4. Space utilization is also ~~Bad~~

Removed all duplicates due to compaction.



LSM Trees

→ by Stratchmid Hayt Trees