Agenda.

⇒ NoSQL Internals.

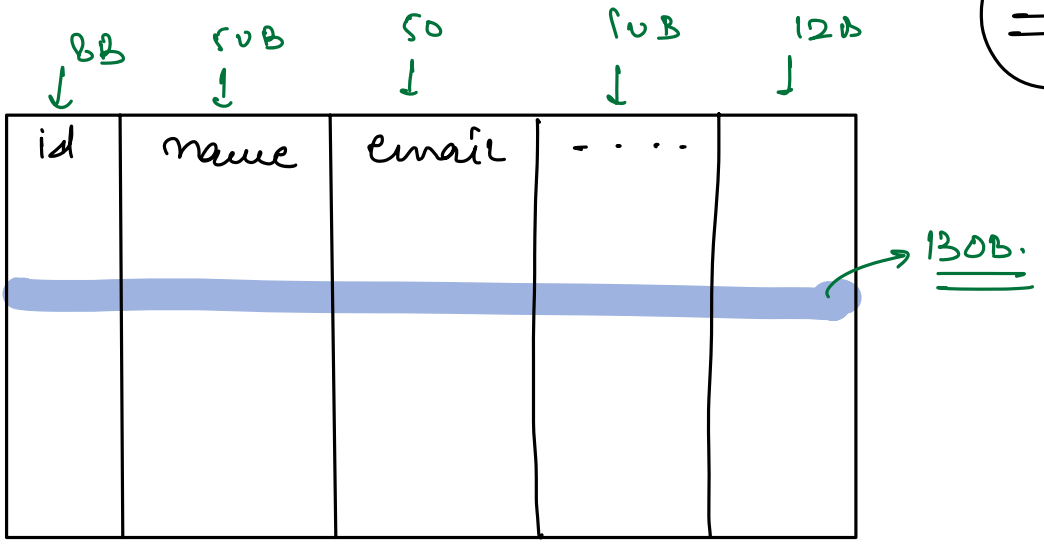
⇒ SQL.

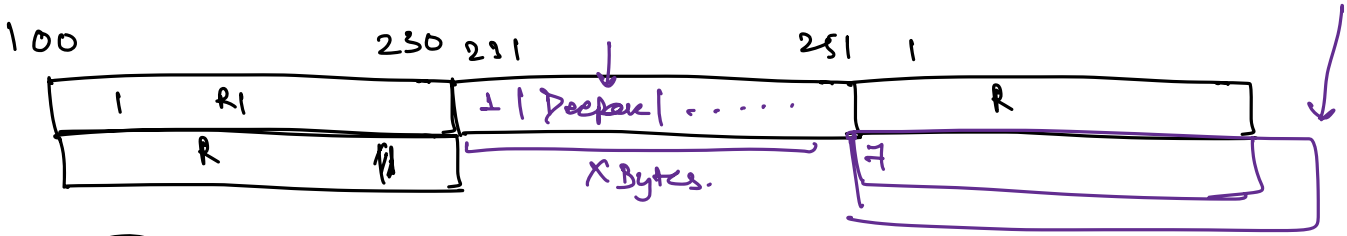
- Structured data
- fixed schema

⇒ NoSQL.

- Unstructured | Semi structured data
- No fixed schema.

HDD-

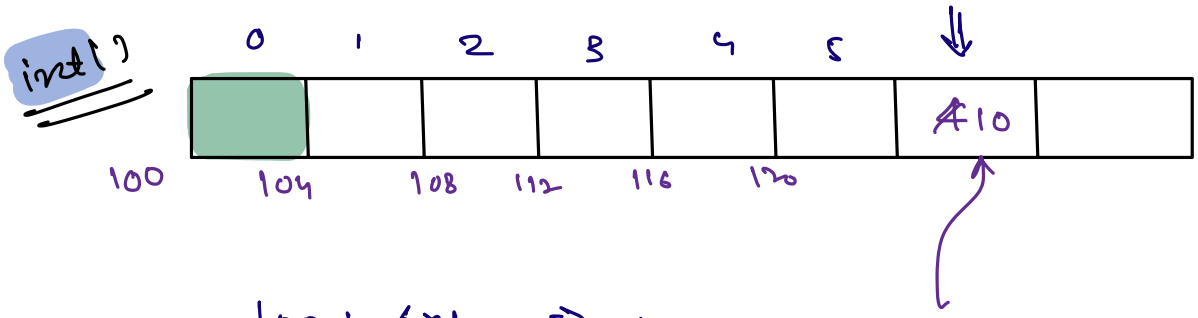




⇒ Blocks.

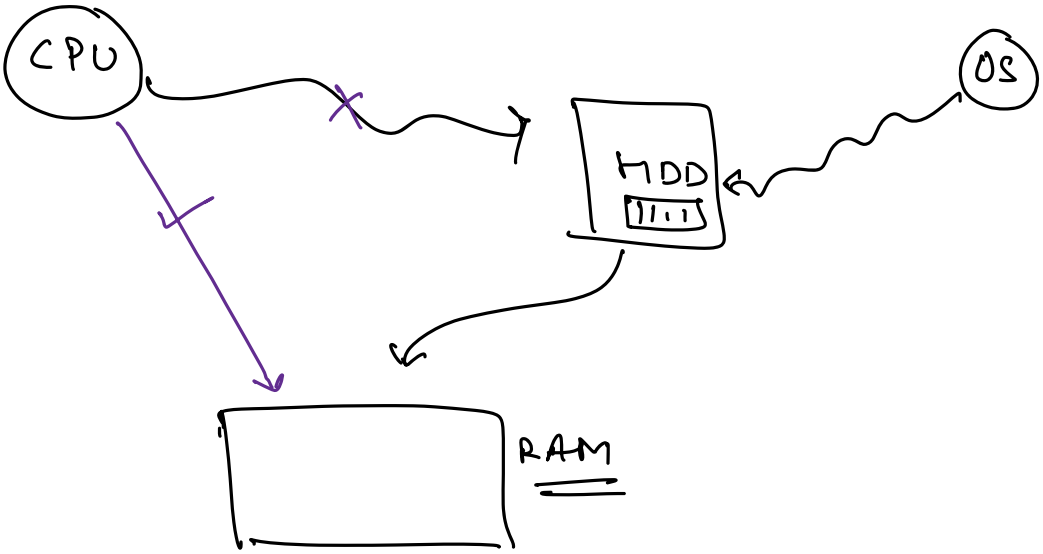
↳ 64B.

Indexing : id = 4 | B7

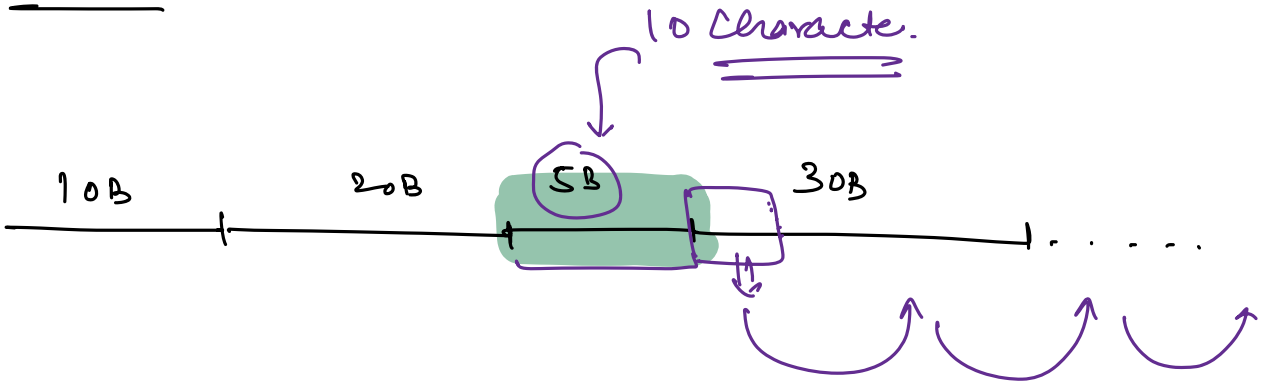


100 + 6×4 ⇒ 124

↳ 0111



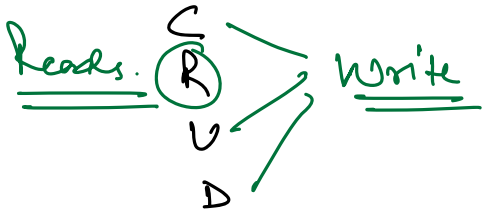
⇒ NoSQL



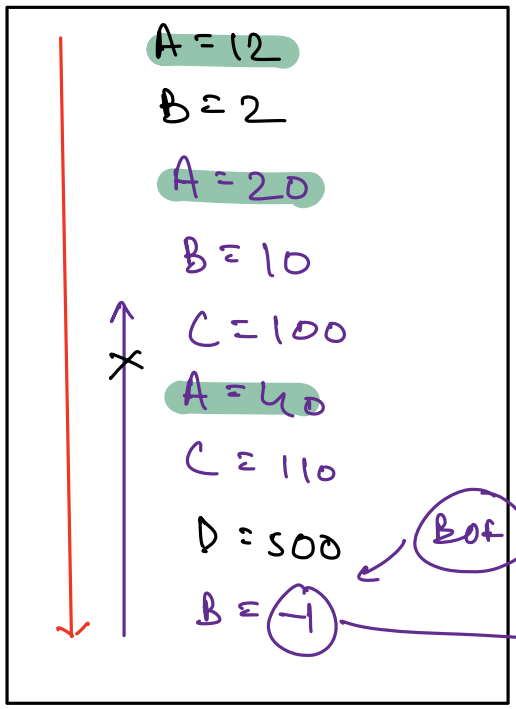
WAL.

→ Write Ahead Log

→ Append Only.



#



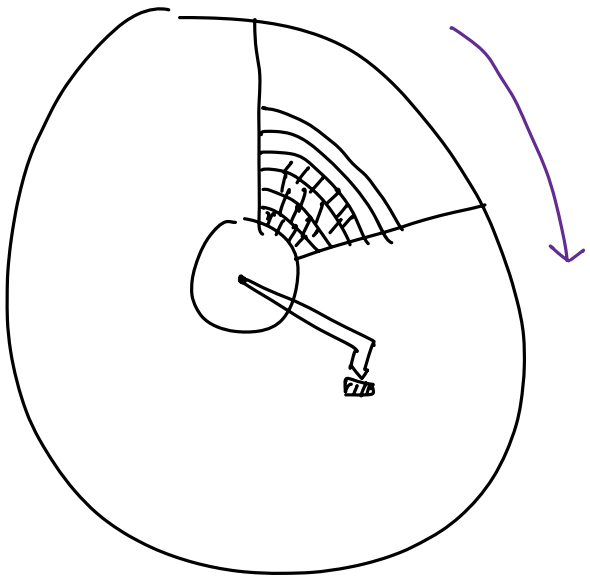
↙ WAL ⇒ Disk.

A = 12
40

→ Any undefined value we can set of Delete.

⇒ Write : O(1)

⇒ Read : O(N) → # of write operations.



⇒ Whenever Write happens, Append the write operation in WAL file.

⇒ Read : Iterate the WAL file & get the latest value.

SQL DB.

- Read : O(logN)
- Write : O(logN)

NoSQL DB.

Approach # 1

- Only use WAL.

TC :

Read : O(N)

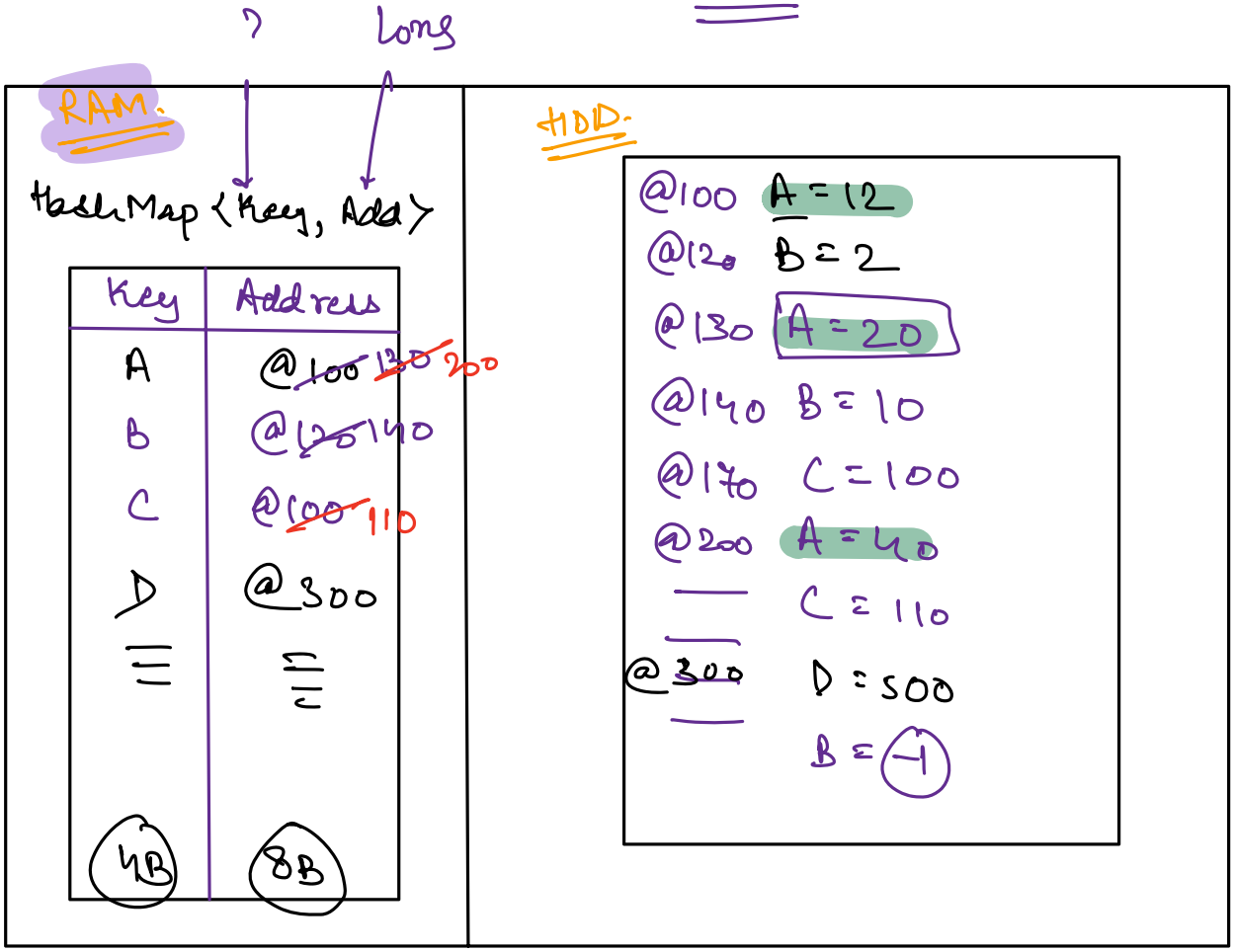
Write : O(1)

WAL ⇒ Disk.

@100	A = 12
@120	B = 2
@130	A = 20
@140	B = 10
@170	C = 100
@200	A = 40
—	C = 110
—	D = 500
—	B = -1

Approach #2 WAL + HashMap.

RAM.



Write

- Append WAL ⇒ O(1)
- Update HashMap ⇒ O(1)

⇒ O(1)

Read

- Get address from HashMap ⇒ O(1)
- Read from the address in WAL. ⇒ O(1)

Write, Read } O(1)

⇒ Size of (WAL) file = 1TB.

(WAL) → K, V (Int, Int)

Size of 1 entry = 8B

No. of entries = $\frac{1TB}{8B}$

= $\frac{10^{12}}{10}$ Bytes

= 10^{11} entries. = 100 Billion

Size of 1 entry in HashMap = 12B.

Size of entire HashMap = 12B × 100Billion

$= 12 \times 100 \times 10^9$ B.

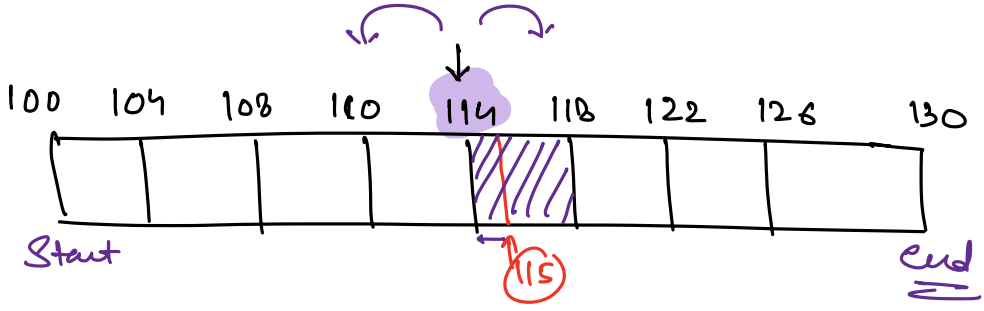
$= 1.2$ TB.

⇒ We might not be able to store this much of huge data inside the RAM.

⇒ We'll have to re-initialize the HashMap in case our m/c gets restarted.

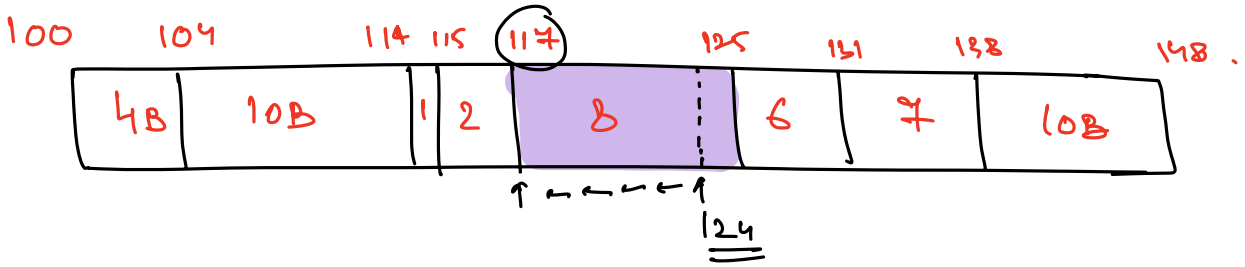
Binary Search

↳ Sorted + Equal sized Data



(X)

$\frac{100+130}{2} = 115$



$$mid = \frac{100 + 148}{2} = 124$$

Approach # 3. HAL + TreeMap.

TreeMap - <K, V>

Key	Address
A	@~~100~~ ~~130~~ 200
B	@~~120~~ 140
C	@~~100~~ 110
D	@300
=	=
4B	8B

HDD:

@100 A = 12
@120 B = 2
@130 A = 20
@140 B = 10
@170 C = 100
@200 A = 40
— C = 110
@~~300~~ D = 500
— B = -1

⇒ TreeMap : Allows us to store keys in the sorted order.

Write

- Append WAL ⇒
- Update TreeMap ⇒

] O(logN)

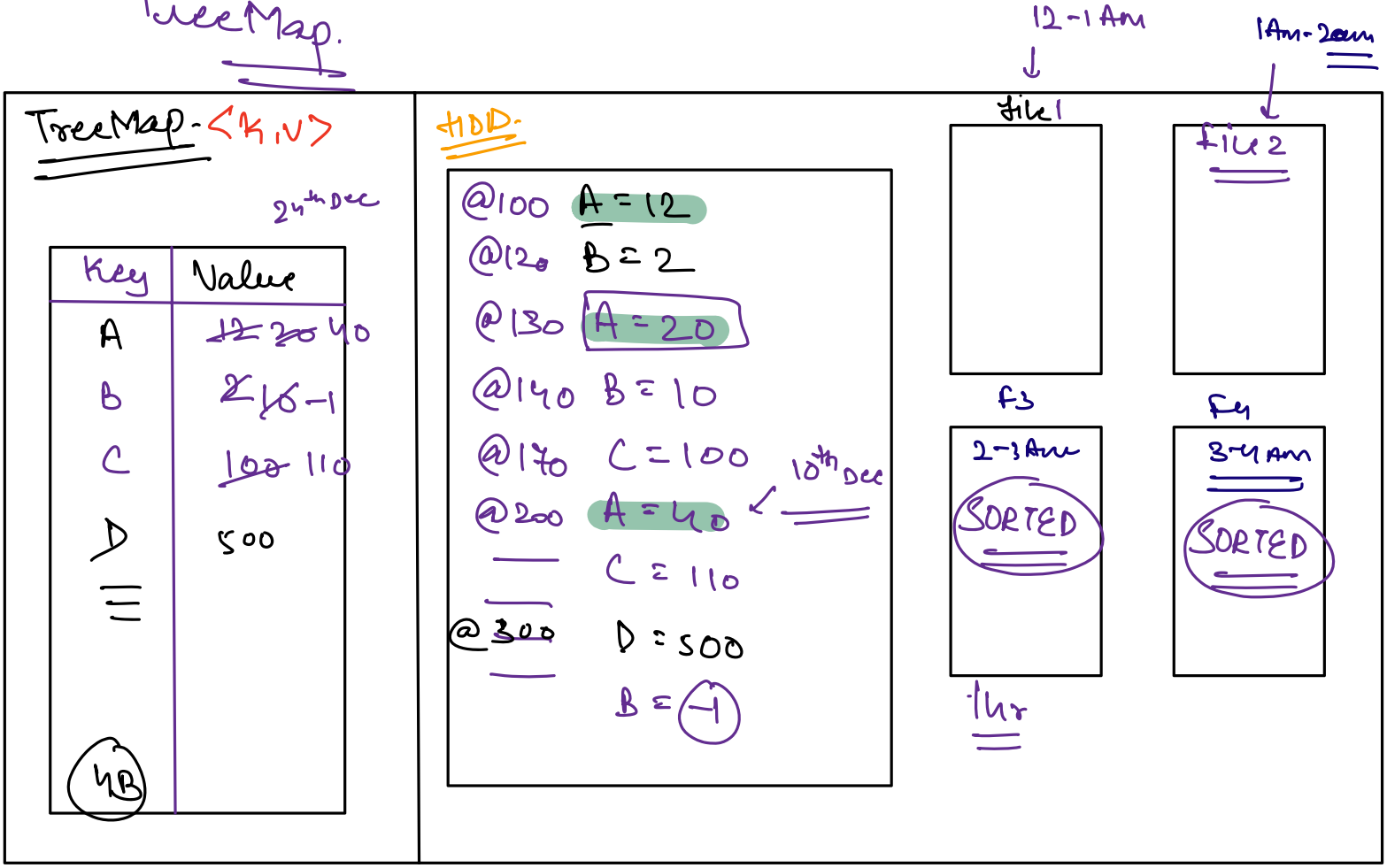
Read

- Read TreeMap
- Read WAL

] O(logN)

Background Script

⇒ After every 1 hour, take the complete data from TreeMap, create a file & reset the TreeMap.



11:59 Pm

Write.

- Write in WAL file → O(1)
- Add in treeMap → O(logN)

⇒ O(logN)

Read

- Read from treeMap.
  - if we get the key
  - if not

→ If we don't get the data from treeMap.

24 files ⇒ Start reading data from the latest file.

TC: O(logN) + (# of files) × N. + N
(N: entries of files; N: WAL)

: 25 * N.

Summary.

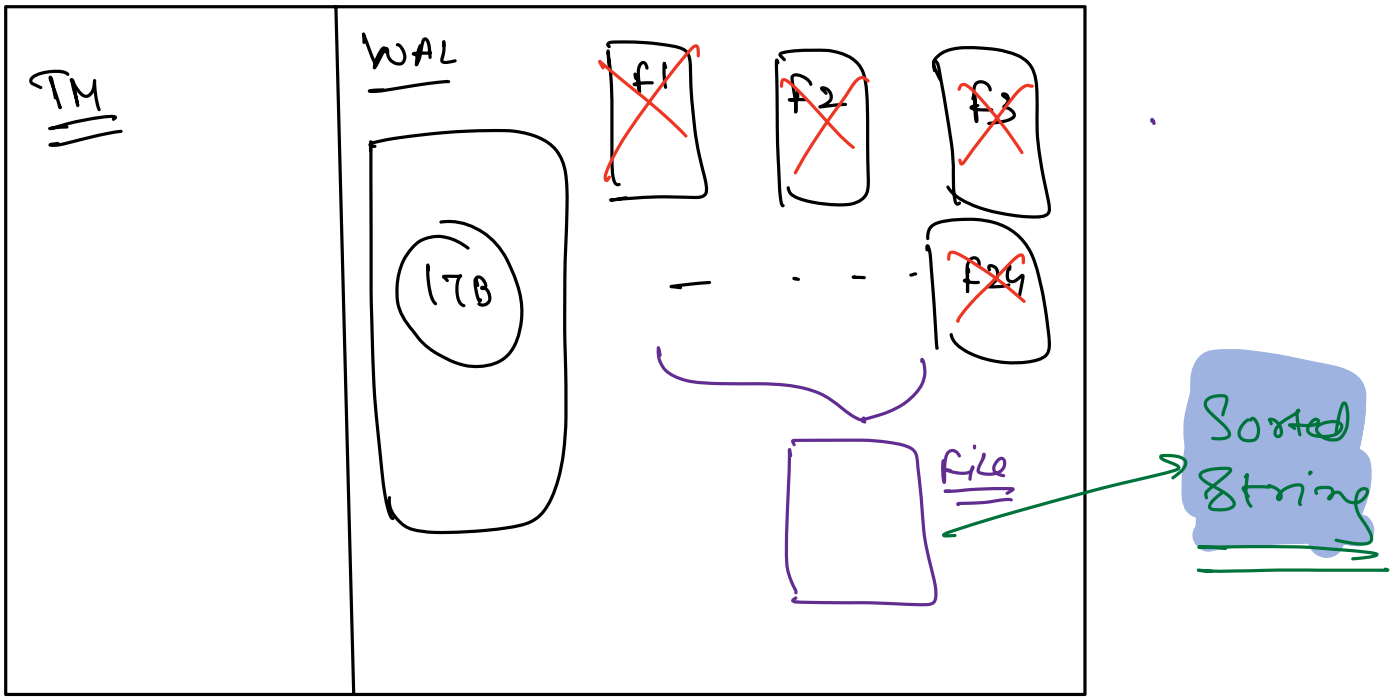
	Write TC	Read TC
Only WAL	O(1)	O(N)
WAL + HM	O(1)	O(1)
WAL + Tree Map	O(logN)	O(logN)
WAL + Tree + files every 1 hr Map	O(logN)	(# of files) * N

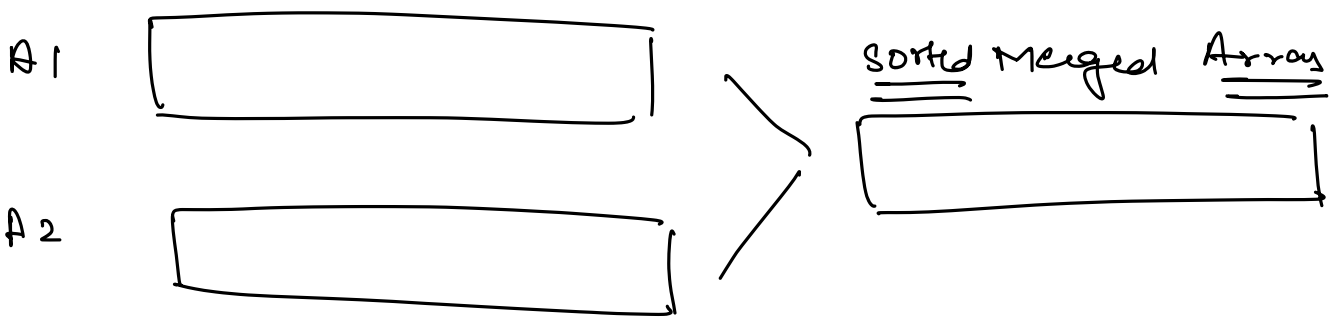
Approach #4

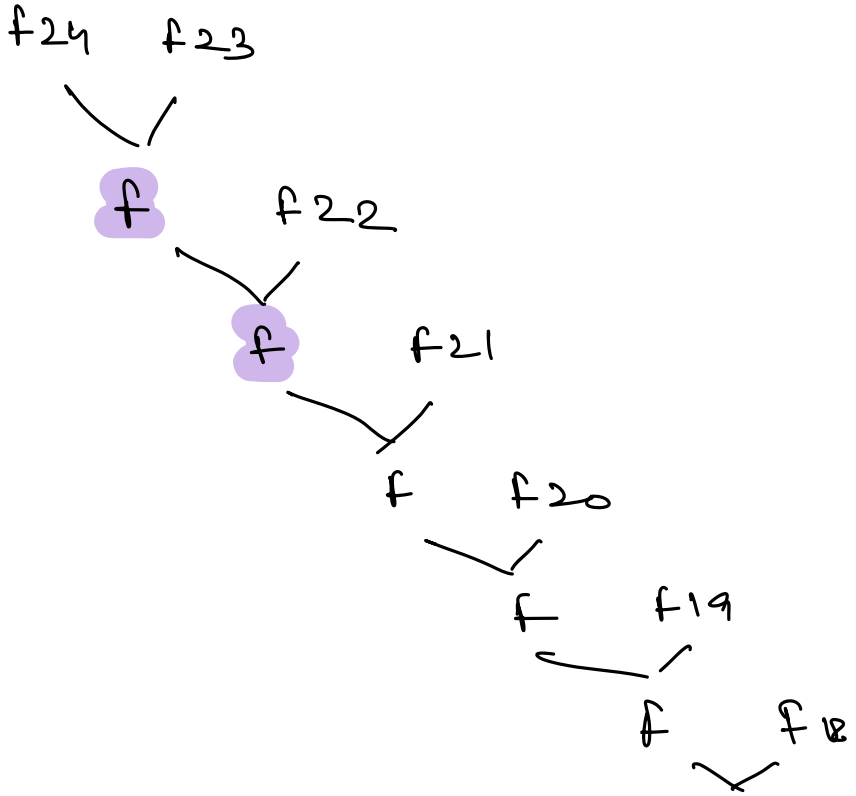
↳ WAL + TreeMap + Merge files every 1 Day.

Background Script

⇒ Merge all the 24 files of a Day into 1 file


TM
WAL
f1
f2
f3
f24
178
file
Sorted String


A1
A2
Sorted Merged Array


f24
f23
f
f22
f
f21
f
f20
f
f19
f
f18

Write

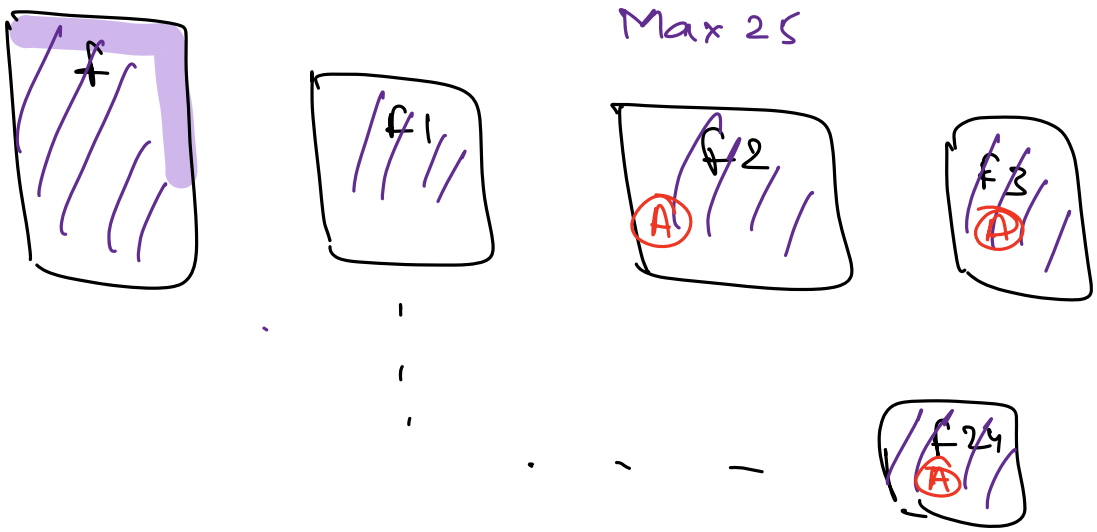
- Write in WAL
- Write in TM

} O(logN)

Read

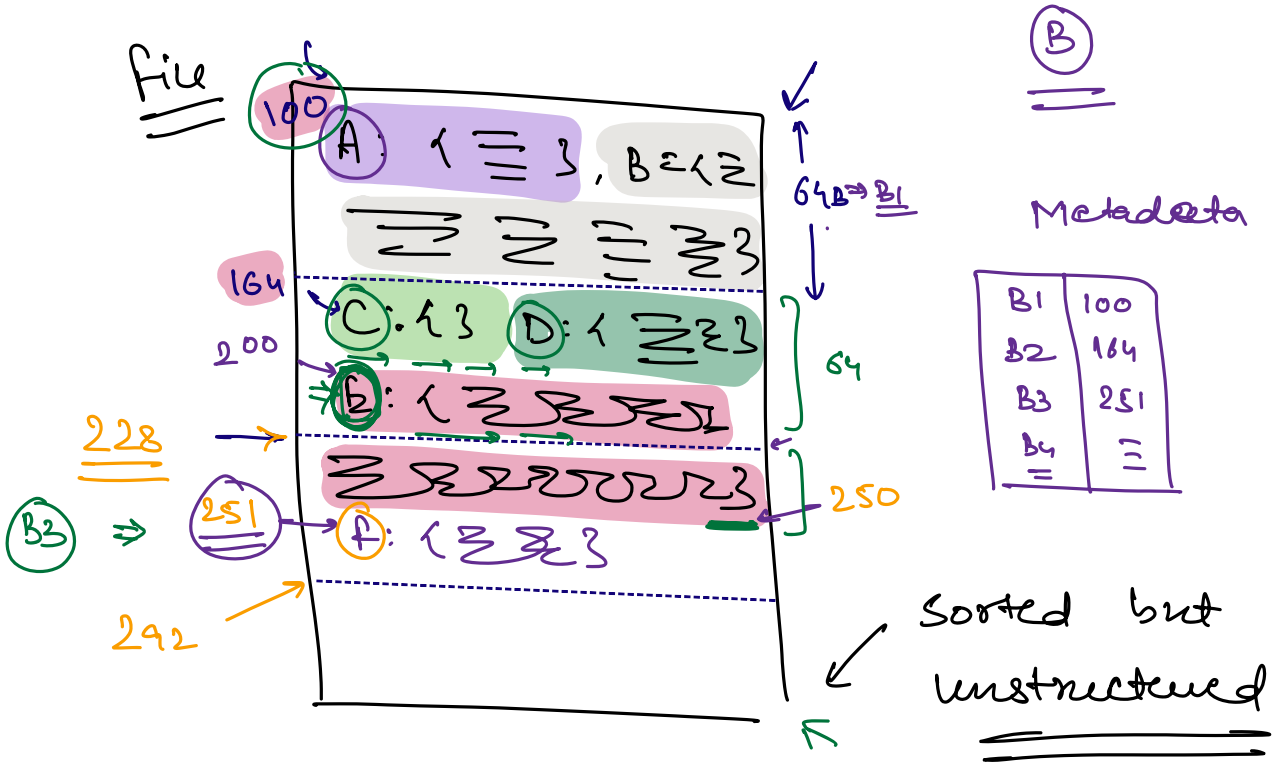
- Read TreeMap → O(logN)
- Iterate through all the files in WC → (# of files) × N.

Max 25



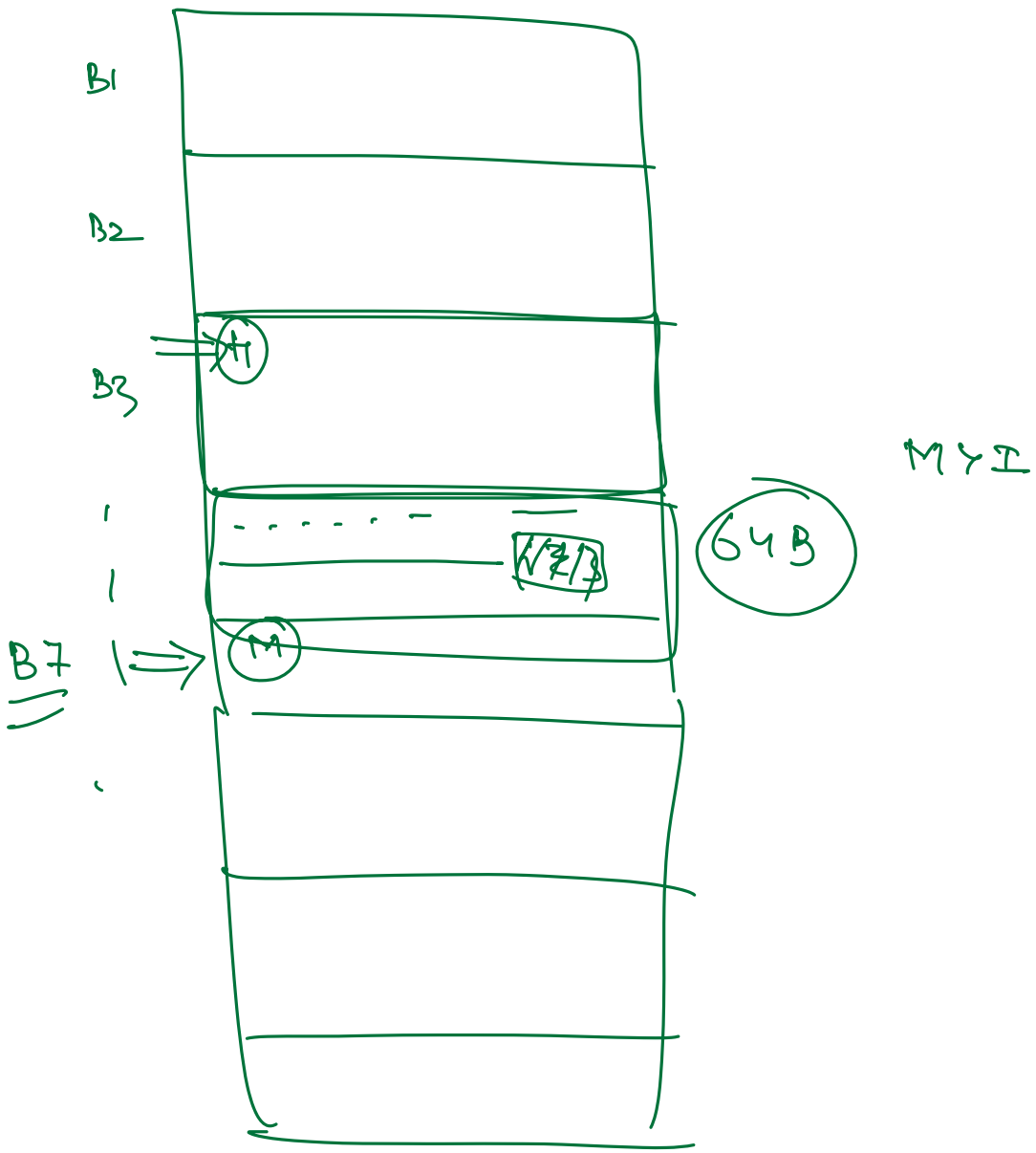
TC: 25×N.

Note: Some how if we are able to search in O(logN) TC within every file.



⇒ When we are creating files, we will divide every file into logical blocks of equal size, let's say 64B.

⇒ Along with each file we also maintain a metadata which contains the address of first element in each block.



WAL + TreeMap + Merged files.

Write

- Write in WAL
- Write in TM

O(logN)

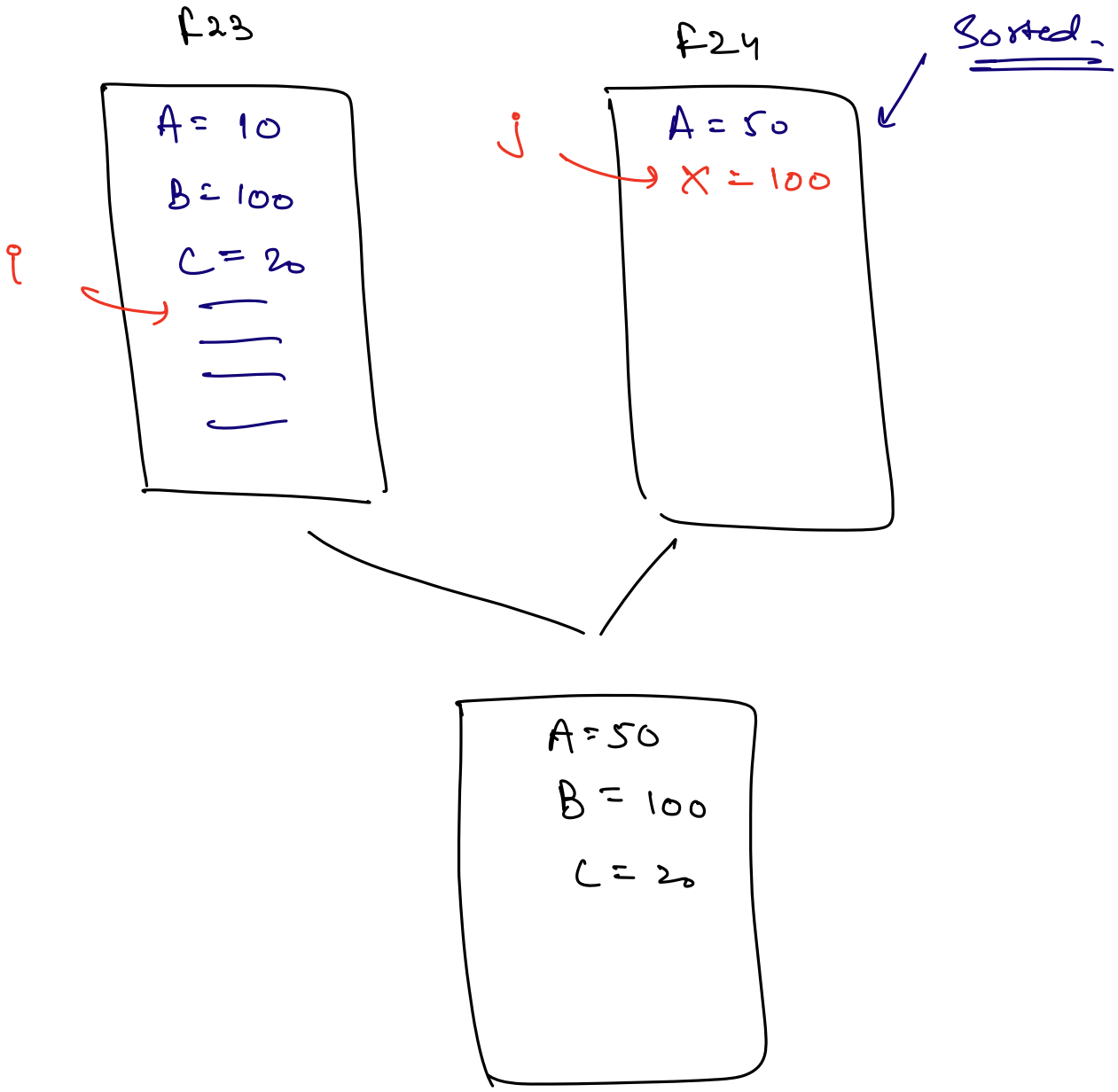
Read TC

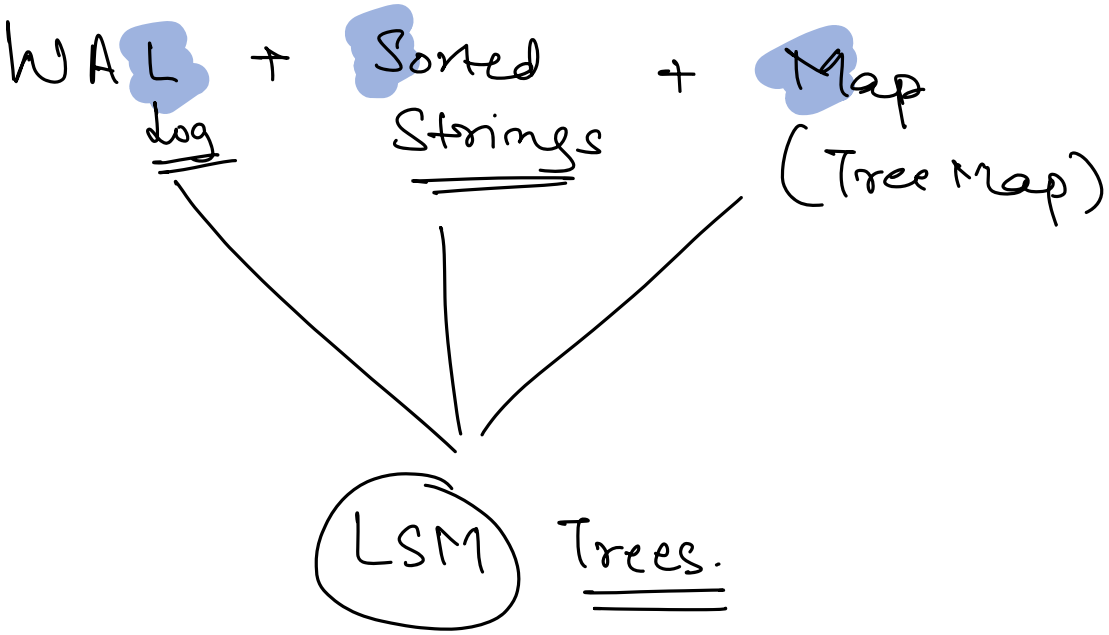
- Tree Map ⇒ O(log N)
- Iterate each file using Binary Search

⇓

(# of files) * O(log N)

TC ⇒ O(log N) ✓




WAL
log
+
Sorted
Strings
+
Map
(Tree Map)
LSM Trees.