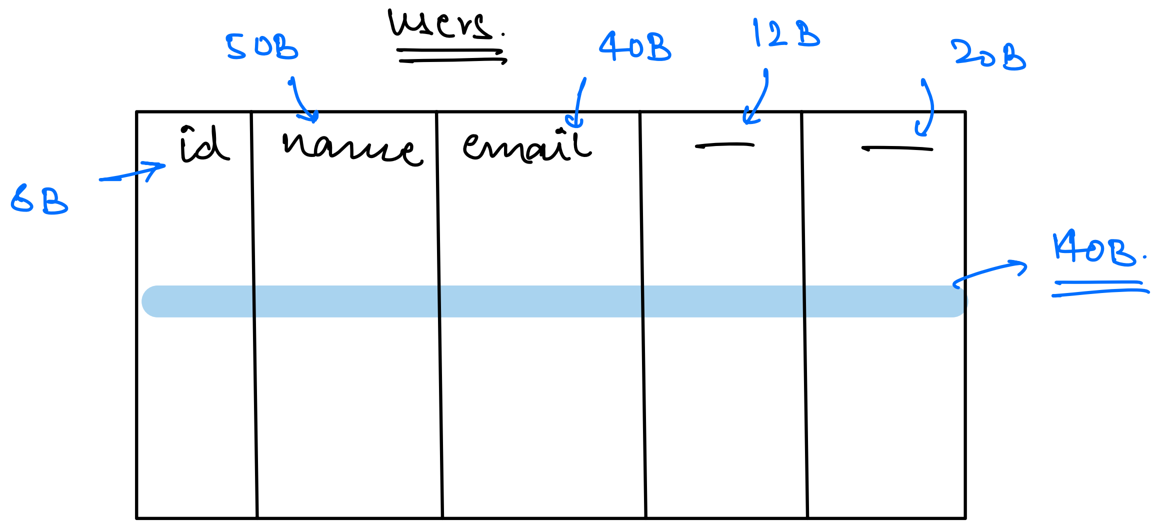
Agenda.

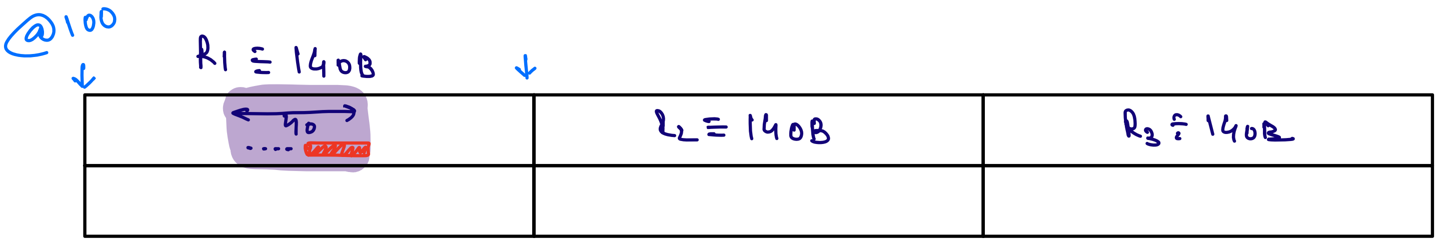
⇒ NoSQL Internals.

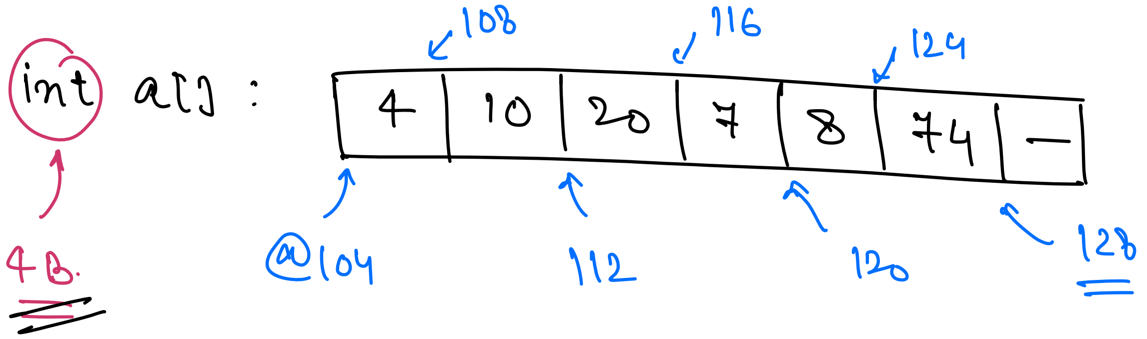
How NoSQL DBs stores data internally.

SQL DB.

- Structured Data
- fixed Schema.



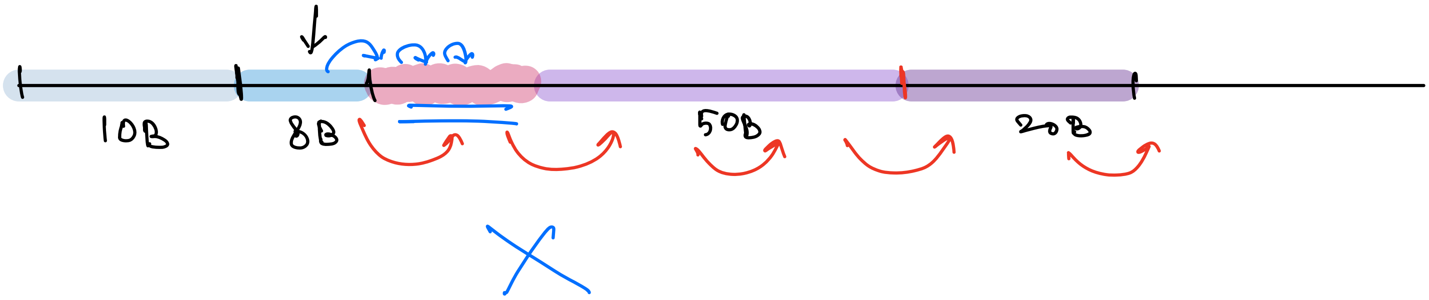




$$a[i] \leftarrow \underbrace{a[0] + i \times 4}_{O(1)}$$

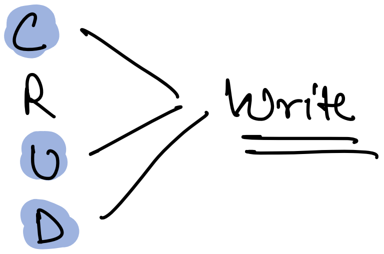
NoSQL DB.

- → Unstructured Data
- → No fixed Schema.



⇒ WAL

- → Write Ahead Log.
- → Append Only file.



WAL

A= 12
B= 40
C = 20
A = 24
D= 100
B = 41
E = 500
C = -1
A= 30
EOF →

K, V

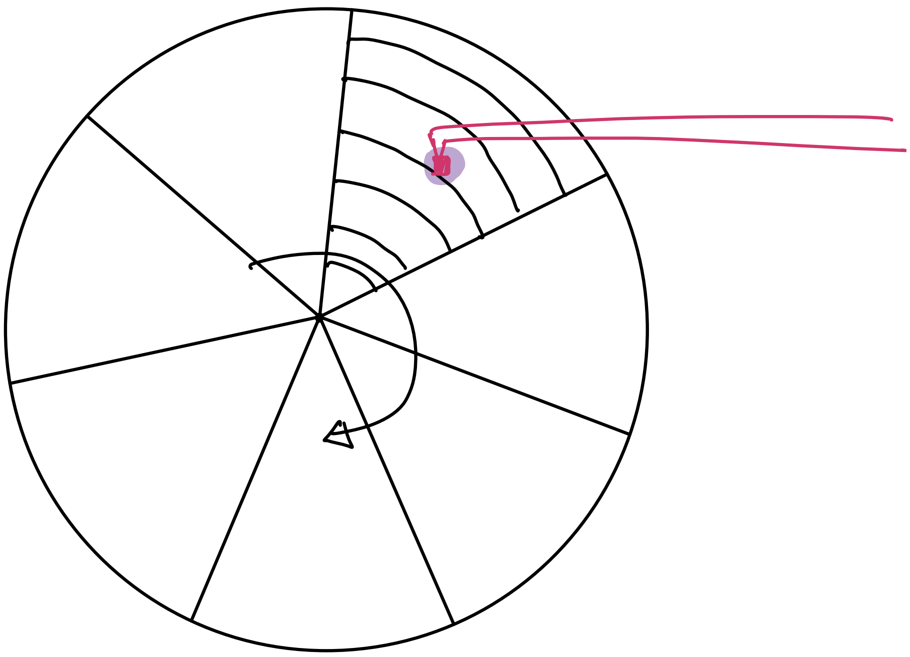
WAL file

- → Maintains the complete history of operations.
- → Reconstruct our DB from WAL.
- → Used to sync Replicas

Write opⁿ TC in WAL ⇒ O(1)

Read opⁿ TC in WAL ⇒ O(N) → # of write opⁿ

→ Start reading the file from starting till the end and get the latest value of the key.



SQL DB

→ Read TC : O(logN)

→ Write TC : O(logN)

Approach#1

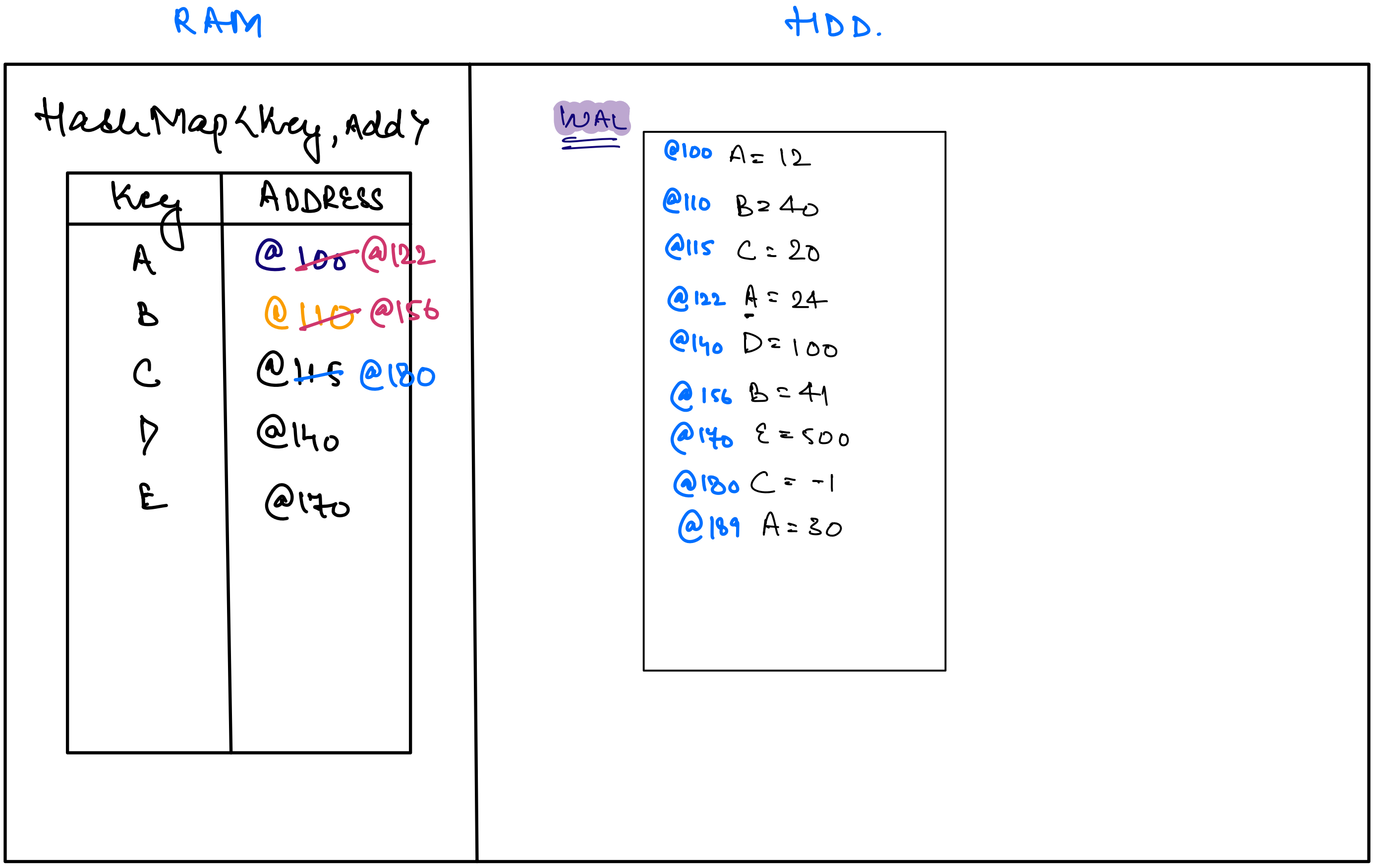
→ Only WAL.

Read TC : O(N)

Write TC : O(1)

Approach#2

→ WAL + HashMap.



HashMap<Key, Add>

Key	ADDRESS
A	@ ~~100~~ @122
B	@ ~~110~~ @156
C	@ ~~115~~ @180
D	@140
E	@170

@100 A= 12
@110 B= 40
@115 C = 20
@122 A = 24
@140 D= 100
@156 B = 41
@170 E = 500
@180 C = -1
@189 A = 30

Write Op^n

↳ Append in WAL ⇒ O(1)
↳ Update HM ⇒ O(1)

TC of write ⇒ O(1)

Read Op^n ⇒ O(1)

→ Get the address of the key from HM and read the value from this address in WAL

Cons.

↳ HM is present in the RAM (which is volatile storage), in case our m/c restarts, we'll have to rebuild the complete HM from scratch.

Size of WAL = 10 TB

Size of 1 entry = 10 B.

No of entries $= \frac{10\,TB}{10\,B} = \frac{\cancel{10}\times 10^{12}\,\cancel{B}}{1\cancel{0}\,\cancel{B}}$

$= 10^{12}$ entries.

Assumptions : Unique keys $= \frac{10^{12}}{10} = 10^{11}$

Size of 1 K,V pair in HM ≃ 16 B.

Size of the HM $= 10^{11} \times 16$ B.

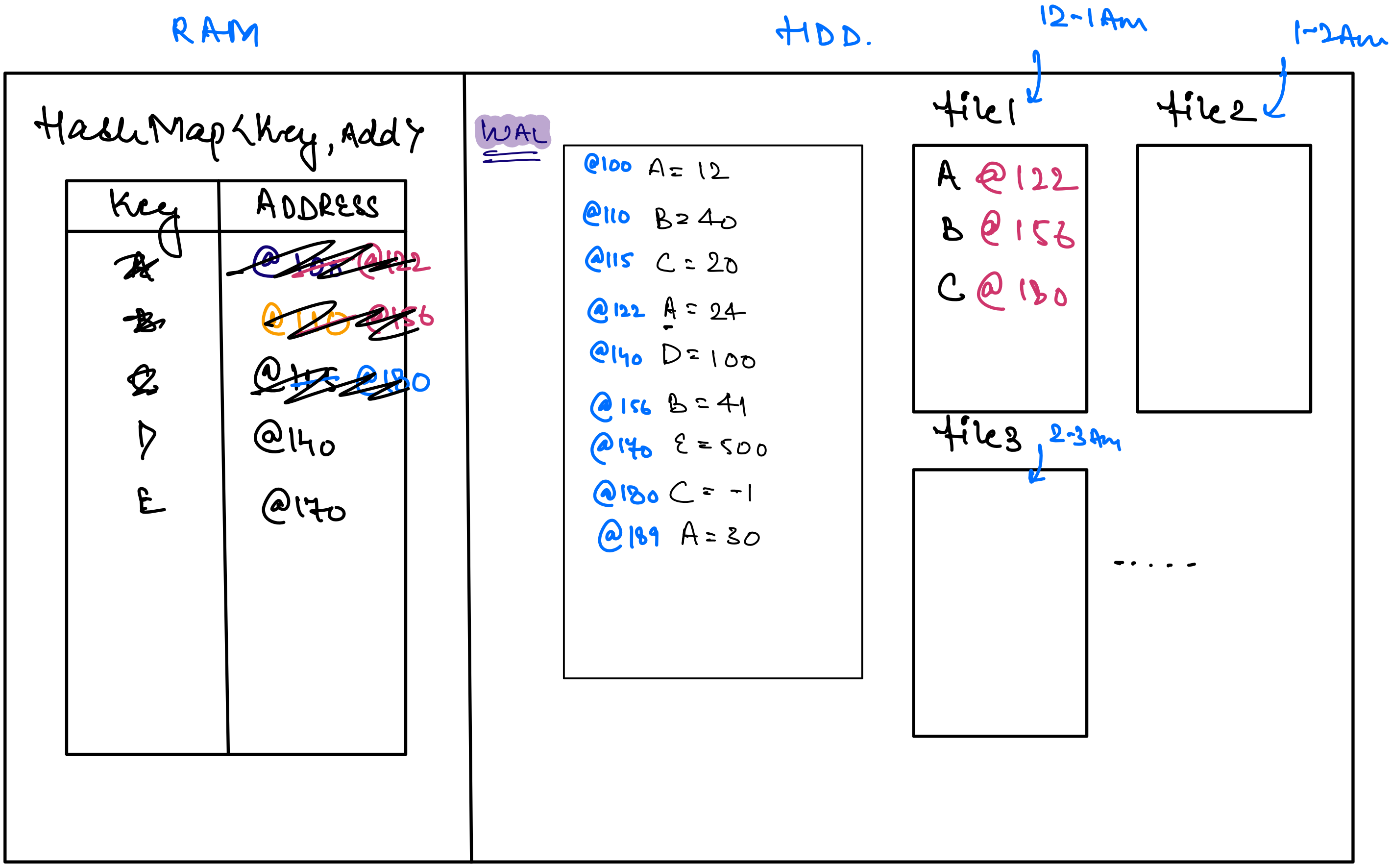
$= 1.6 \times 10^{12}$ B.

$= 1.6$ TB.

⇒ Size of the HM can be huge, it will cost huge amount of money.

Background Script.

⇒ Every 1 hour, take the Data from HM and put it in the file in HDD & reset the HM.



Key	ADDRESS
~~A~~	~~@100~~ ~~@122~~
~~B~~	~~@110~~ ~~@156~~
~~C~~	~~@115~~ ~~@180~~
D	@140
E	@170

Write

- → Write in WAL ⇒ O(1)
- → Write in HM ⇒ O(1)

Read.

→ find the key in the Map

if found

Get the Address from HM & read from WAL.

if NOT found

Read all the files from latest to oldest (Max 24 files)

$TC: \underset{\substack{\downarrow \\ HM}}{O(1)} + (\# \text{ of files}) \times \boxed{\underline{\underline{N}}}$ → Iterate every hourly file <u>linearly</u>.

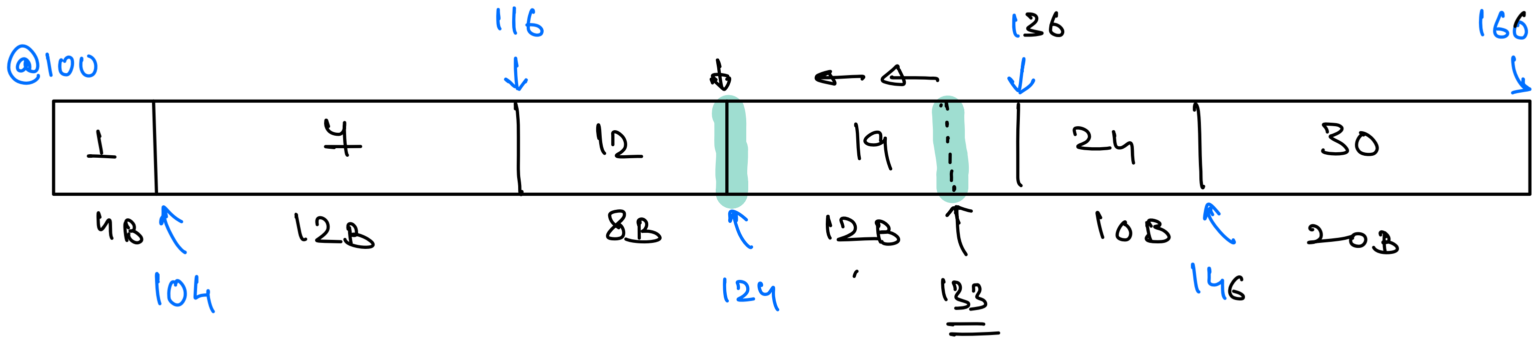
$: (\# \text{ of files}) \times \underline{\underline{N}}.$

Summary.

	Read TC	Write TC
Only WAL	O(N)	O(1)
WAL + HM	O(1)	O(1)
WAL + HM + Hourly files	$(\# \text{ of files}) \times \underline{\underline{N}}.$	O(1)

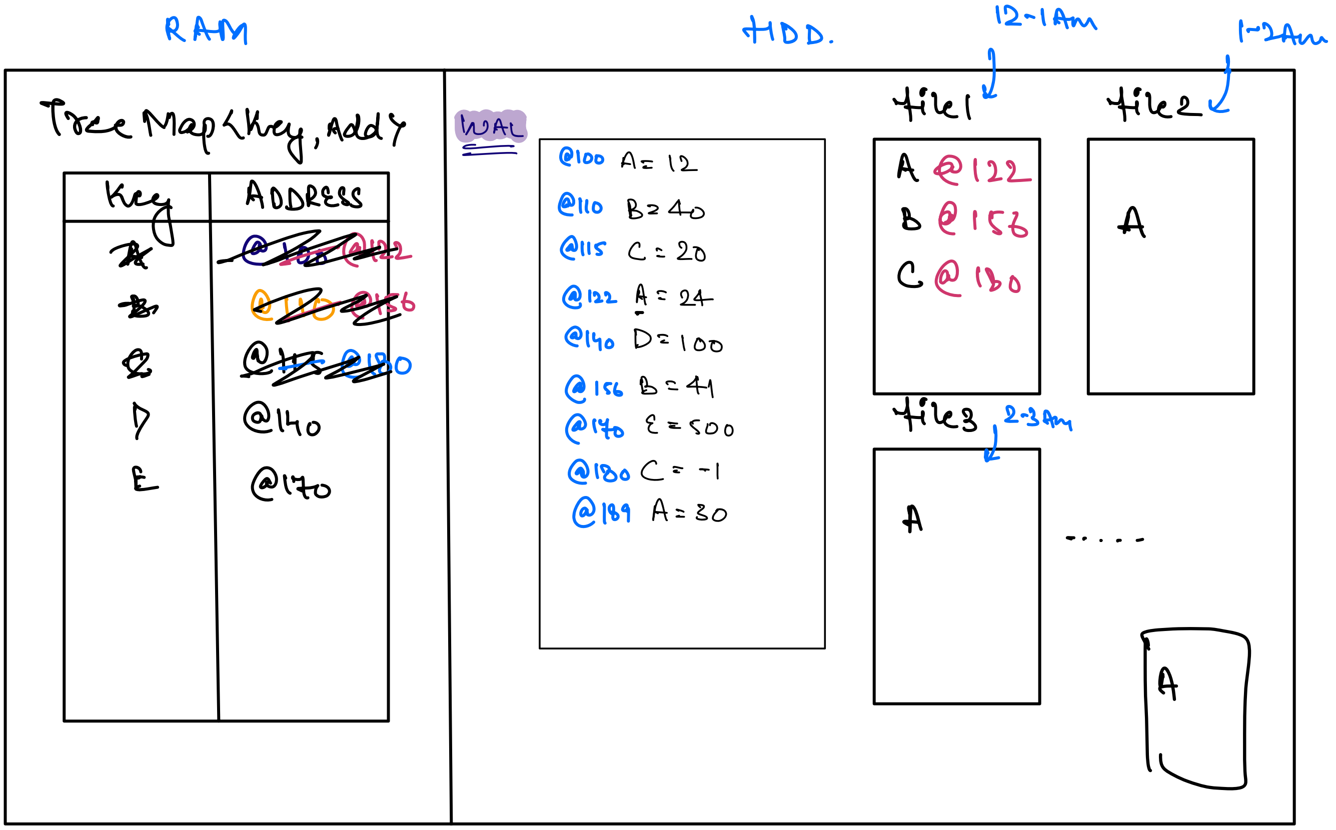
Binary Search

↳ Sorted Dataset + Equal Size Data.



$$\frac{100+166}{2} = 133$$

⇒ Instead of HashMap, use Tree Map.



TreeMap<Key, Add>

Key	ADDRESS
~~A~~	~~@100~~ ~~@122~~
~~B~~	~~@110~~ ~~@156~~
~~C~~	~~@115~~ ~~@180~~
D	@140
E	@170

@100 A= 12
@110 B= 40
@115 C = 20
@122 A = 24
@140 D= 100
@156 B = 41
@170 E = 500
@180 C = -1
@189 A = 30

file1

A @122
B @156
C @180

Write

- → Write in WAL ⇒ $O(1)$
- → Write in Tree Map ⇒ $O(\log N)$

$\}$ $O(\log N)$

Read.

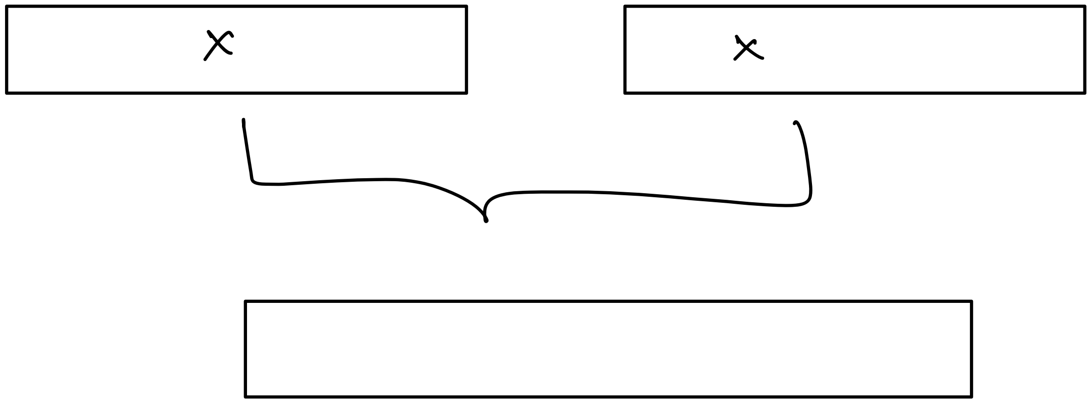
- → find the key in the TreeMap
  - ↙ if found
  - ↘ if NOT found

Read TC : $O(\log N) + (\# \text{ of files}) \times \boxed{\underline{N}}$

Summary.

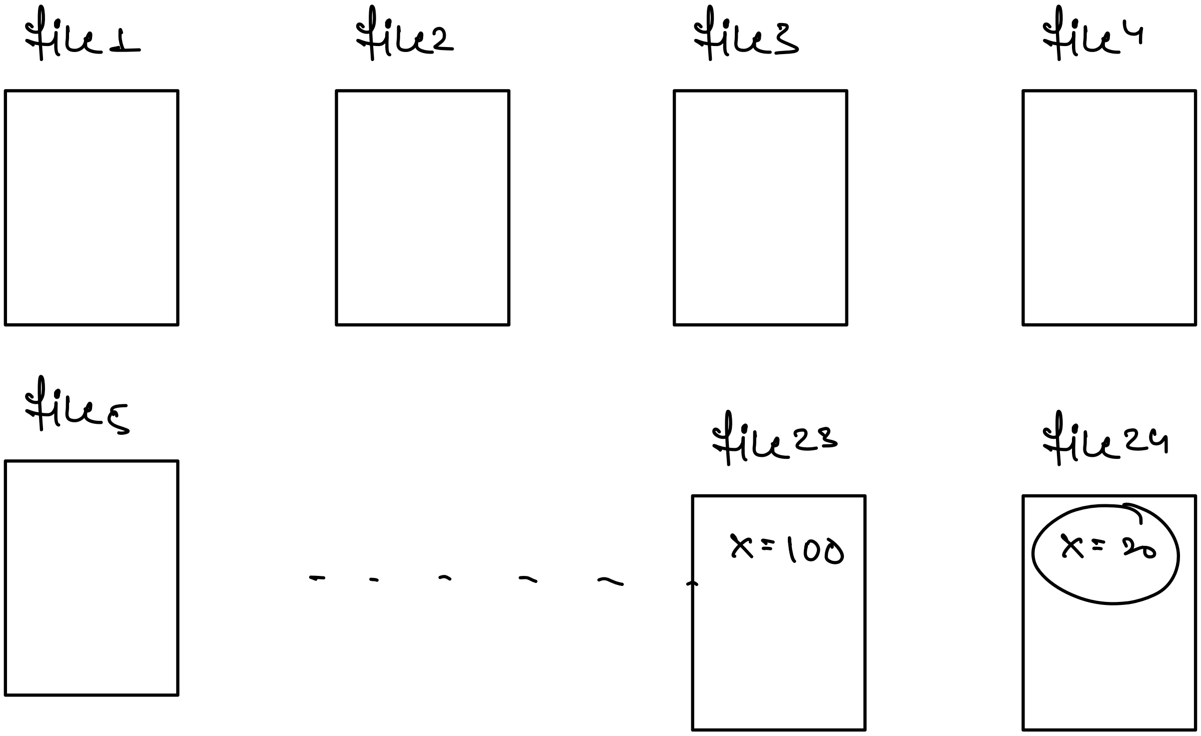
	Read TC	Write TC
Only WAL	$O(N)$	$O(1)$
WAL + HM	$O(1)$	$O(1)$
WAL + HM + Hourly files	$(\# \text{ of files}) \times \underline{N}.$	$O(1)$
WAL + TM + Hourly files	$(\# \text{ of files}) \times \underline{N}$	$O(\log N)$

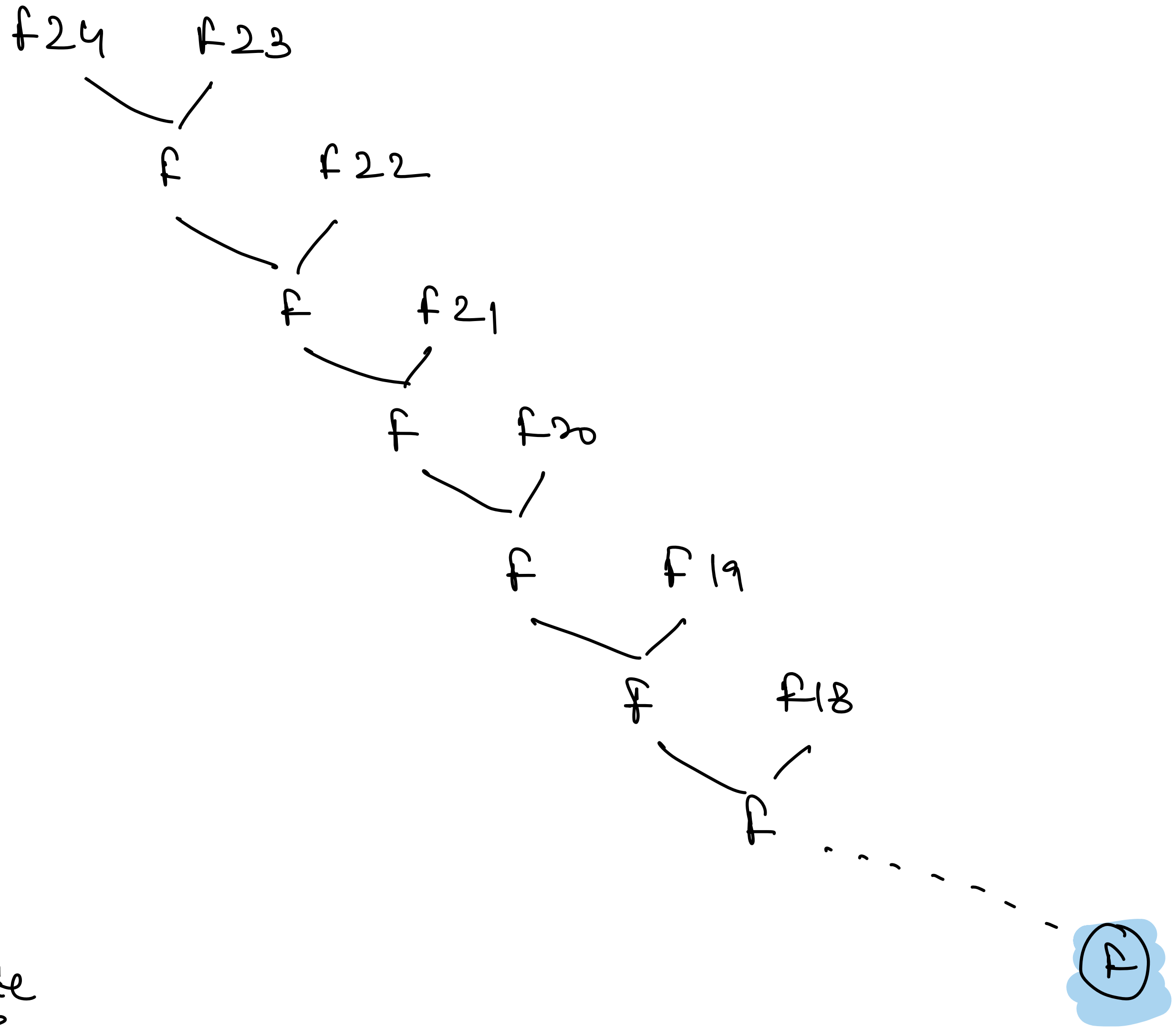
2 Sorted Array.



Background Script.

⇒ Every 24 hours, merge all the files (from file 1 to file 24) into 1 sorted file.





Write

- Write in WAL ⇒ O(1)
- Write in Tree Map ⇒ O(logN)

} O(logN)

Read.

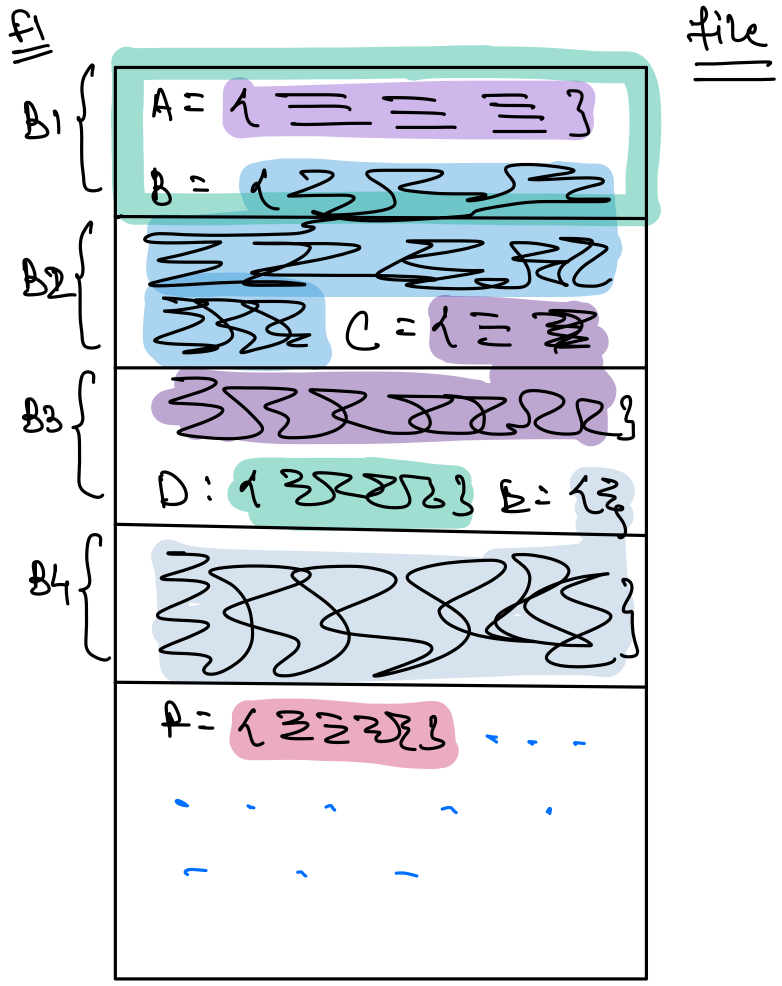
- Read from TreeMap.

if found
- Get from WAL

if NOT found
- Iterate through all the files from latest to oldest.

Read TC : O(logN) + (# of files) × N.

Sol^n : Some how if we can apply Binary Search on the files, then we are done.



64 B

(Map)

Metadata for f1

B1 :	A
B2 :	C
B3 :	D
B 5 :	f

Final Sol^n

Write

- → Write in WAL ⇒ O(1)
- → Write in Tree Map ⇒ O(logN)

} O(logN)

Read.

↳ Read from TreeMap.

- if found
  - Get from WAL
- if NOT found
  - ⇒ Apply Binary search on the files from latest to oldest.

$$\begin{tikzcd}
f24 \arrow[r] \arrow[d] & f23 \arrow[r] \arrow[d] & f22 \arrow[d] & \cdots\cdots & f1 \arrow[r] \arrow[d] & F \arrow[d] \\
\log N & \log N & \log N & & \log N & \log N
\end{tikzcd}$$

Read TC ⇒ logN + (# of files) × logN.

= logN.

Log file + Sorted + Merge + Trees
(WAL) Strings (TreeMaps)

≡ LSM
Trees