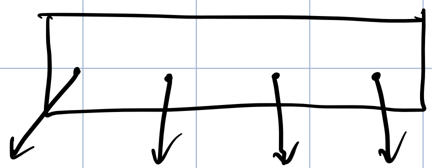
25 Feb 2025 – DBMS-II – Week 08

Final Submission of Group Project → 25 - 26 March

→ For this course, you don't need exact algorithms for B^+-trees.



max pointers = N

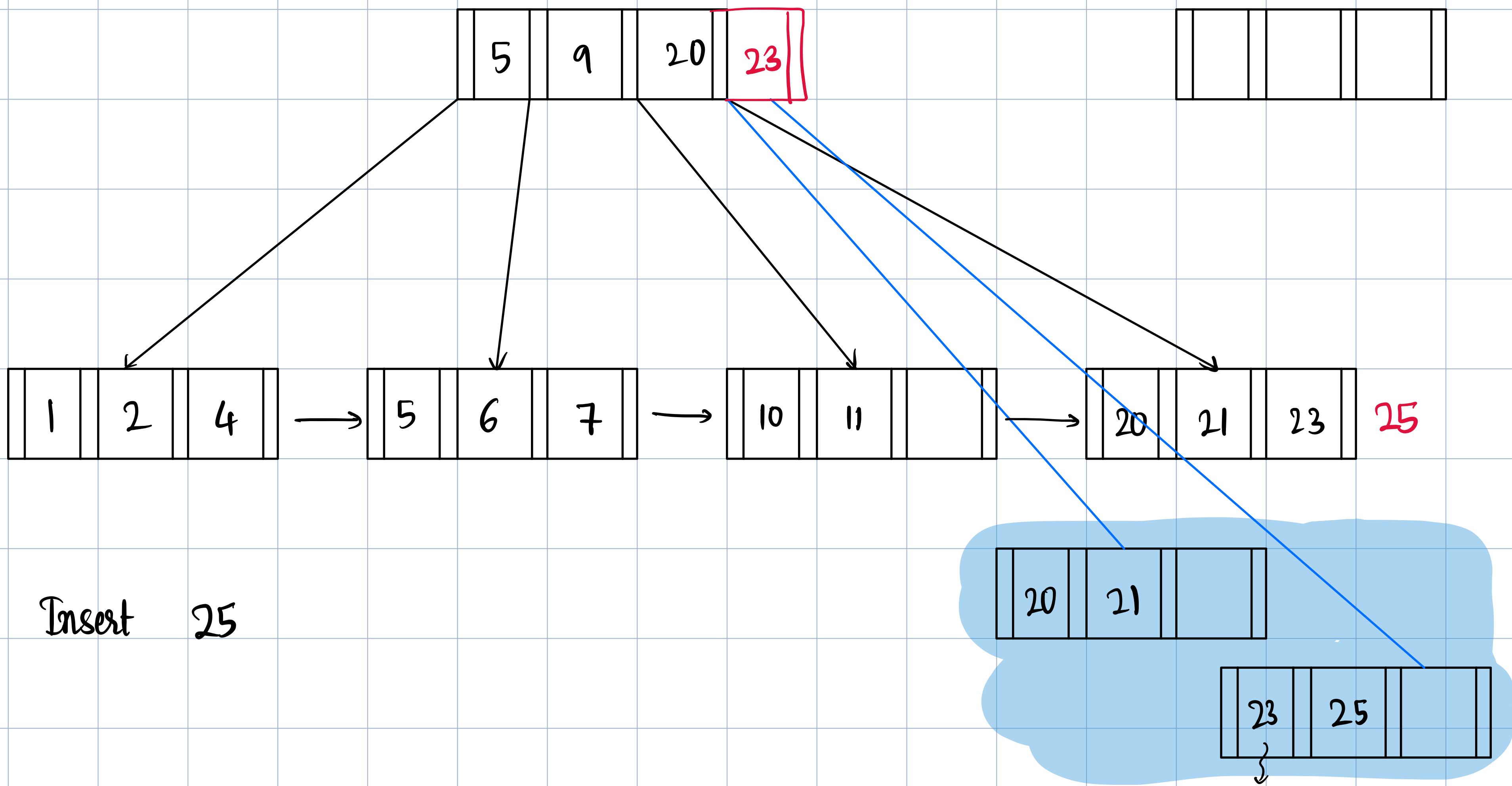
min pointers = $\lceil N/2 \rceil$

→ each non-root and non-leaf node has max N children

min $\lceil N/2 \rceil$ children

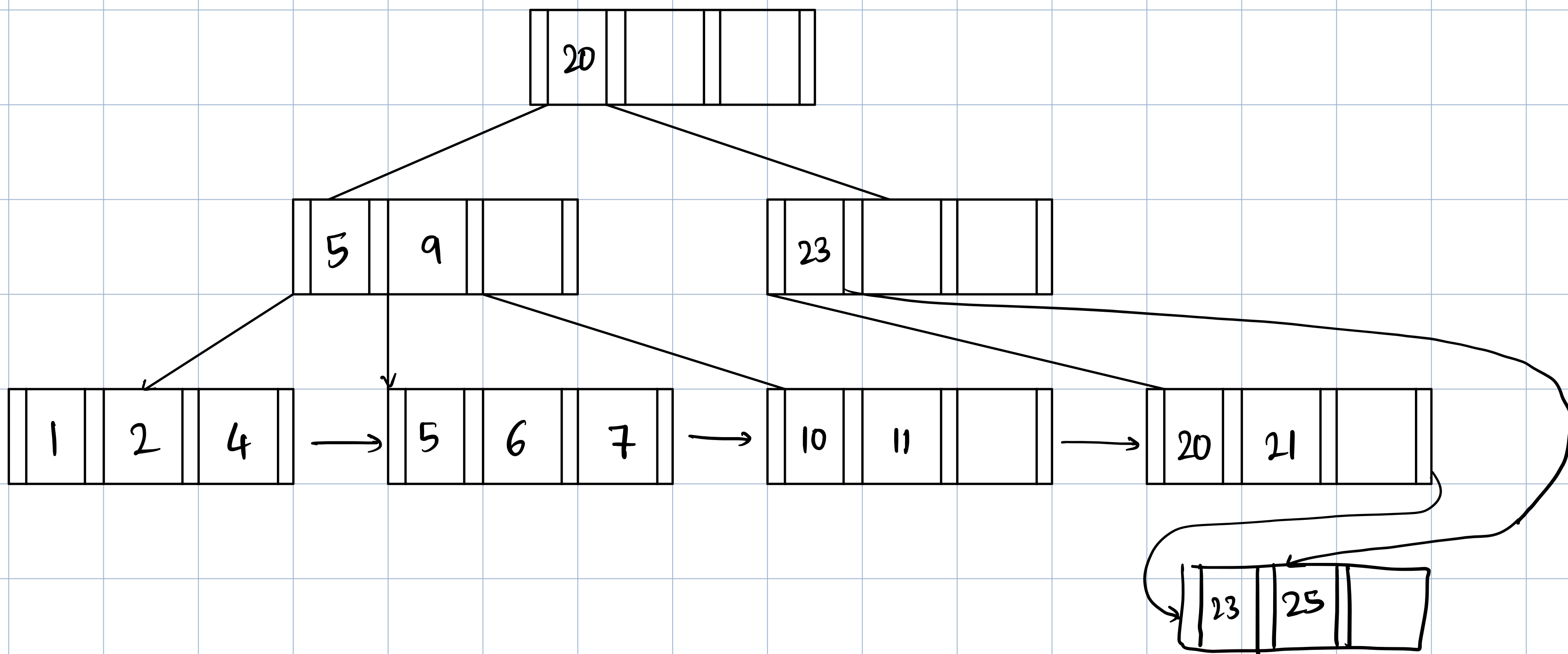
→ each leaf node has max $(N-1)$ values

min $\lceil \frac{N-1}{2} \rceil$ values.



Insert 25

after adding on top,
23 can be
removed in
B-tree not
B+ tree



Benefit : All leaves are at the same height

* Example of B^+ - Tree Index file.
  → no need for shifting on insertion.
  See also : B^+ - Tree file (not an index, leaf nodes are records)

How to move minimum stuff from main memory to cache

→ work of a computer architect
not discussed here.

How to move minimum no. of pages from secondary memory to main memory → B^+ - trees.
one option

* Deletion in B^+ - trees.

→ If on deletion min no. of nodes < $\lceil \frac{N-1}{2} \rceil$
try to shift from sibling
→ if sibling had exactly $\lceil \frac{N-1}{2} \rceil$ children,
merge nodes → update parent (recursively)

→ Deletion may decrease the height.

→ Insertion may increase the height

Understand examples first, then code.

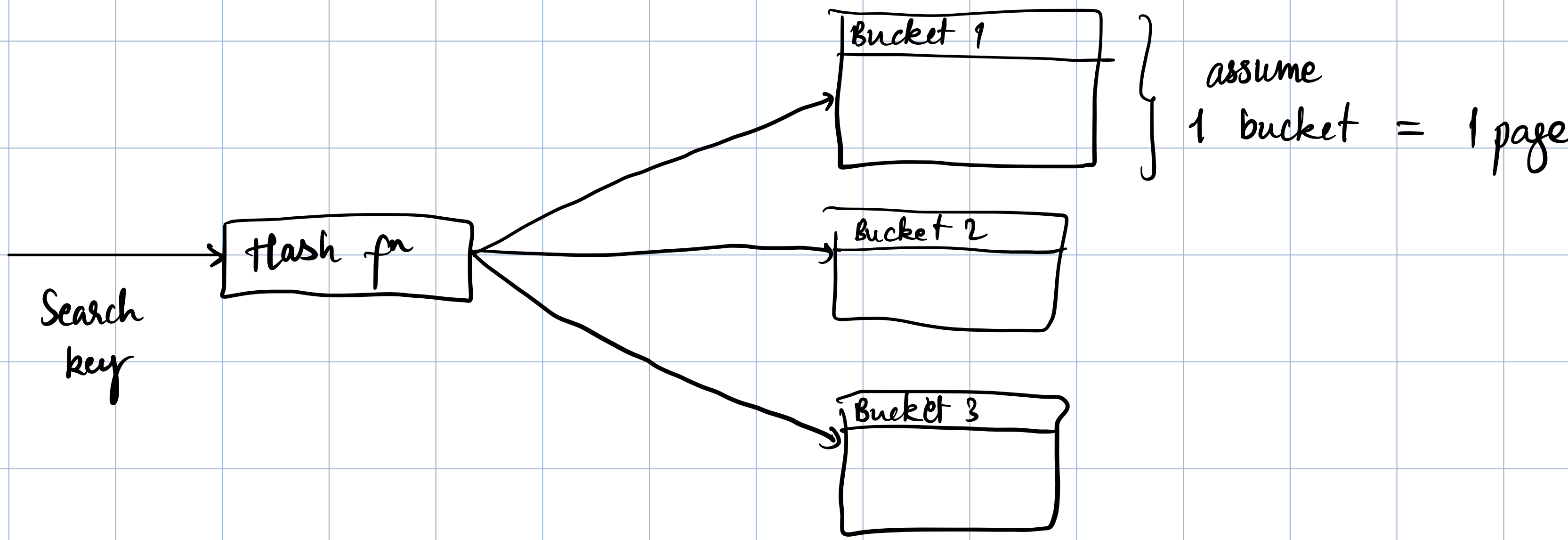
Hashing

→ Indices are not sorted.

→ Even in B^+ - trees, you need to search 3-4 pages

→ Heap file ~ no ordering

→ Hash file ⇝ buckets



→ Faster even than B^+- trees

→ Not good for range - based queries.

Static Hashing

- Bucket
- Hash file organization
- Hash function

Example of Hash file organisation

8 buckets

hash function :

- see slides

→ Ideal hash function is uniform and random

- See slides

Handling Bucket Overflows

1. Insufficient buckets
2. Skew in distribution of records → bucket based on salary, what if many records have same salary?
   - Hash function inefficient

Overflow chaining

1. Open hashing
2. Closed hashing

Hash indices

- Hash can be used not only for file-organization, but also for index-structure creation.

25 Feb 2025 – Extra lecture

Static hashing → predecided buckets

If all buckets are full, you will need to chain, create more buckets.

- See slides

(RM)
Raghuramakrishnan book for some parts from now
→ short and to the point

RM **Extendible Hashing**

Create one more bucket (not chained) and modify the hash functions in a way that old buckets are pointed in the same way

2 bits to 3 bits

- See slides

Cost Model

B data pages with R records per page

fetch sec mem to main mem

Avg. time to read or write a disk page is D

Avg. time to process a record is C

CPU processing, main mem → Cache

In hashed file organization → time required to apply hash function = h

Comparision of 3 file organisations ↗ on 5 operations

Heap file

Scan : $B(D + RC)$

- D → bring page to main mem
- RC → processing time in CPU for all records in a page
- $B(D + RC)$: $(D + RC)$ = total time required to process 1 page

Search with equality selection : $0.5B(D + RC)$ → on average
(if search key is candidate key)
if not $B(D + RC)$

Search with range selection : $B(D + RC)$ → not sorted, scan everything

Insert : $2D + C$ → $2D$ ∵ main mem ⇆ sec mem.
∵ Free list is maintained

Delete : cost of searching + $\underbrace{C}$ + D

↓ small amount compared to D can be ignored.

Sorted file

Scan : $B(D + RC)$

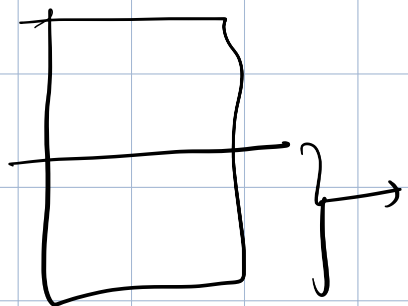
Search : binary search

$D \log_2 B + C \log_2 R$ ⟶ (minimum) only if qualifying record's key is candidate key

If not $D\log_2 B + C \log_2 R$ + cost of reading all qualifying records in seq. order

Search with range selection : → same as search with equality selection

Insert



search and move half on avg.

Search + $2 \times (0.5 B (D + RC))$

Delete : same as insert :

search cost + $B(D + RC)$

} very costly

Hashed files

Scan : $1.25 \times B(D + RC)$

pages kept at about 80 percent occupancy

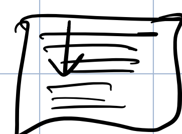
Buckets may overflow → consider extra space

Search with equality selection :

H (which bucket to retrieve)

$+ D$ (retrieve bucket

$+ 0.5 RC$ (if bucket is not sorted)



$+ (\log_2 R) C$ if sorted.

Search with range selection : bad : 1.25 B (D + C)

Insert : cost of search (→ minus K; page in main memory) + C (process) + D (write to sec mem.)

* You need extra storage in hash.

Summary table

In a real DBMS, a file is almost never kept fully sorted.

→ Periodic reorganisation.

Visit Ladakh before finishing BTech

everything uncertain after

Filmora

250 km/day

K2K ⊗

from North-East

External Sorting

??

B^+ trees

leaf node = data entries or sequence set

other nodes = index entries

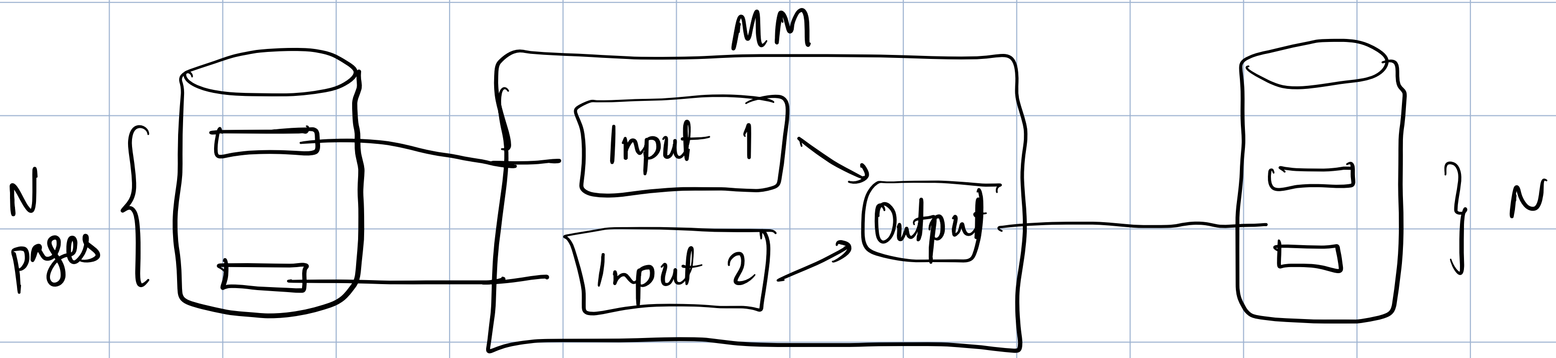
Select distinct requires sorting
other methods exist, sorting is most efficient

Sorting is the most important task, mostly done in background

Merge sort and Quick sort
↓ More space required — Quick sort: in place sorting

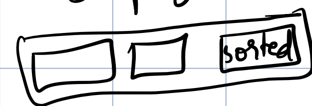
→ Usually records don't fit in main memory

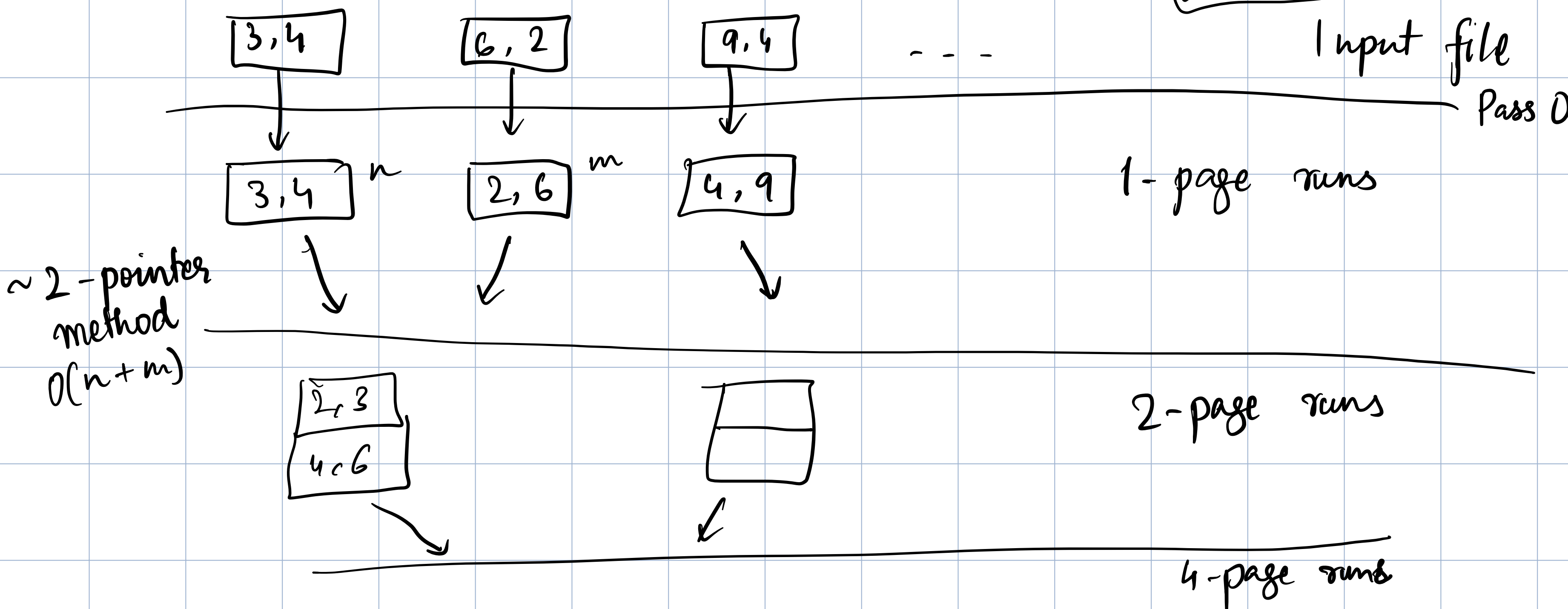
→ Not good for DBMS



Simplified 2-way merge sort

Assume
Main-memory can contain 3 pages





No. of passes = $\lceil \log_2 N \rceil + 1$ N = number of pages in the file

cost of all read-writes = $2N \lceil \log_2 N \rceil$

If N = 8 } 56

Improvement

→ Pass 0 ⇝ instead of one page at a time, read in B pages at a time and sort internally to produce $\lceil N/B \rceil$ runs of B pages.

→ Use quick sort (does not need extra pages)

- Read slides (RM) Lost.

Because B is quite large, savings are exponential.

$$2N * \left(\lceil \log_{B-1} N \rceil + 1 \right)$$

Table → nice

$N = 1000000\ 000$ } → only 4 passes
$B = 257$

More modification : Double buffering

Using B^+ trees for external sorting

① clustered index → Book says beneficial

Sir says <u>no point</u>
↓
you can scan directly

→ useful when you want a small part sorted.

② Unclustered index
↓
Not much benefit

You will have to read one page multiple times

→ not always good (if query = age < 10 and results are only 10% of records

Evaluation on Relational Operators

→ Interesting

SS: useless

RM: Hash indices can be clustered or unclustered

Access paths

select * from ... where age == ~

(1) file scan (can be done in all cases

(2) an index plus matching selection conditions

Selectivity of an access path → no. of pages retrieved

Most selective → least no. of pages

<u>Example</u>

Sailors (   )

Reserves (   )

→ Values

Selection operation

Select *

From Reserves R

where R. rname = 'Joe'

No Index, unsorted data → fetch into main mem
scan row by row

No Index sorted data → binary search

B^+ tree index → clustered or not?

clustered ↓ retrieve 2-3 pages

not ↓ next page

Hash Index, Equality Selection

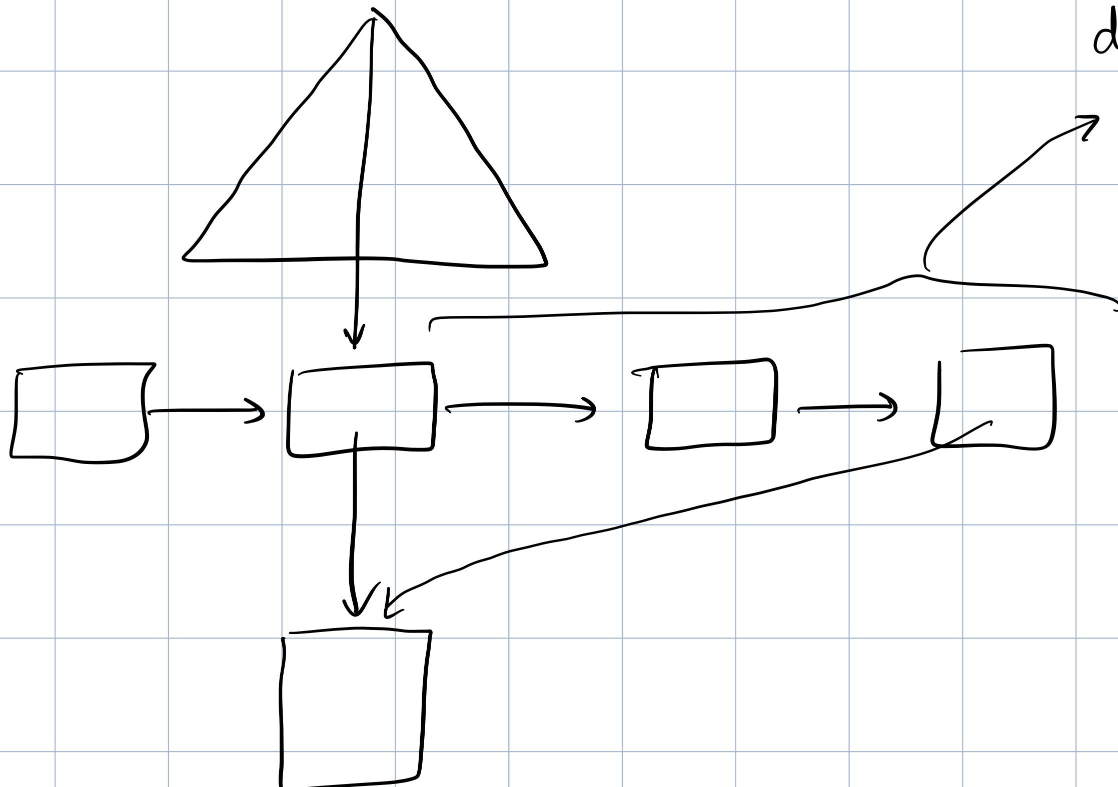
not good for range

$\sigma_{R.attr\ (op)\ value}(R)$

op → (=)

select * from R
where condition

B+ tree unclustered → how to reduce no. of pages accesses?

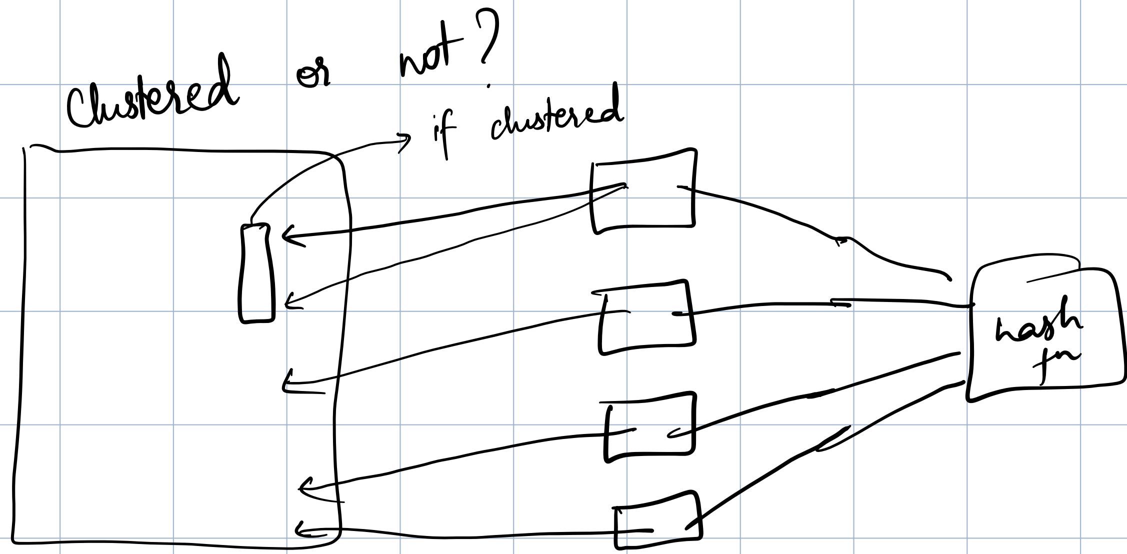


don't fetch file first sort the pointers, then fetch pages

Hash index → clustered or unclustered

Hash index is different from hash file organisation

Alternative (2) in RM



General Selection Conditions

Index on ⟨age, name⟩ → B^+ tree

- Multivalued Hash index

<u>26 Feb 2025</u>

Option ① Scanning

Option ② Use index

Select *

from Reserves R

where R.rname = 'Joe' and R.bid = r

$\sigma_{R.rname = Joe \wedge R.bid = r}(R)$

Conjunctive normal form

$$(A \vee B \vee C) \wedge (D \vee E \vee F)$$

$(bid = 5) \wedge (roll = 7) \wedge (age > 5)$ → CNF with no disjunction

---

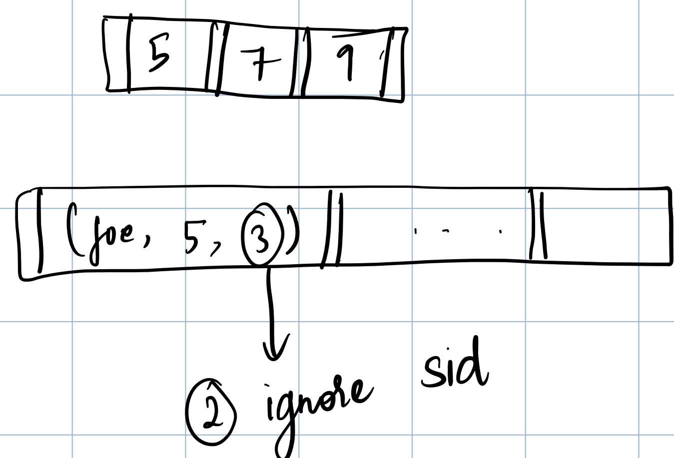
CNF and Index matching

If we have hash index on search key <rname, bid, sid>

① rname = Joe ∧ bid = 5 ∧ sid = 3 ✓

② rname = Joe ∧ bid = 5 ⊗ you have to go for scanning (for hash index)

B^+ - tree ⟶ ① and ② both can be searched



name = 'Joe' ∧ bid = 5 ∧ sid = 3

Even if we have a search key on ⟨bid, sid⟩ we'll still get benefits

B+ tree on ⟨a, b, c⟩

- ✓ a = 5 ∧ b = 5 ∧ c > 5
- ✓ a = 5 ∧ b = 5
- ✗ a = 5 ∧ c = 5
- ✗ b = 5 ∧ c = 5

Using both B+ tree index on day and hash index on ⟨bid, sid⟩

day < ~ ∧ bid = 5 ∧ sid = 3

- day < ~ → fetch pointers to pages from B+ tree
- bid = 5 ∧ sid = 3 → fetch pointers to pages from Hash index

} Take intersection

Conjunction :

day < 8/9/94 ∨ rname = 'Joe' → hash index (on rname = 'Joe')

day < 8/9/94 will require a scan

The Projection Operation

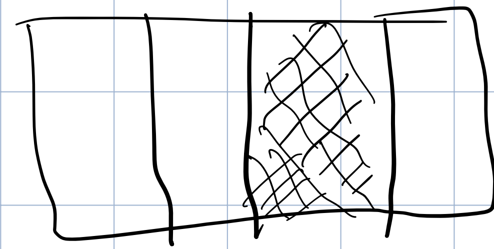
Select Distinct R. sid , R. bid
from Reserves

|||

$\pi_{sid,bid}$ Reserves → project operation does not output duplicates

To implement projection :

- Remove unwanted attributes
- Eliminate any duplicates
  - ① Sorting
  - ② Hashing



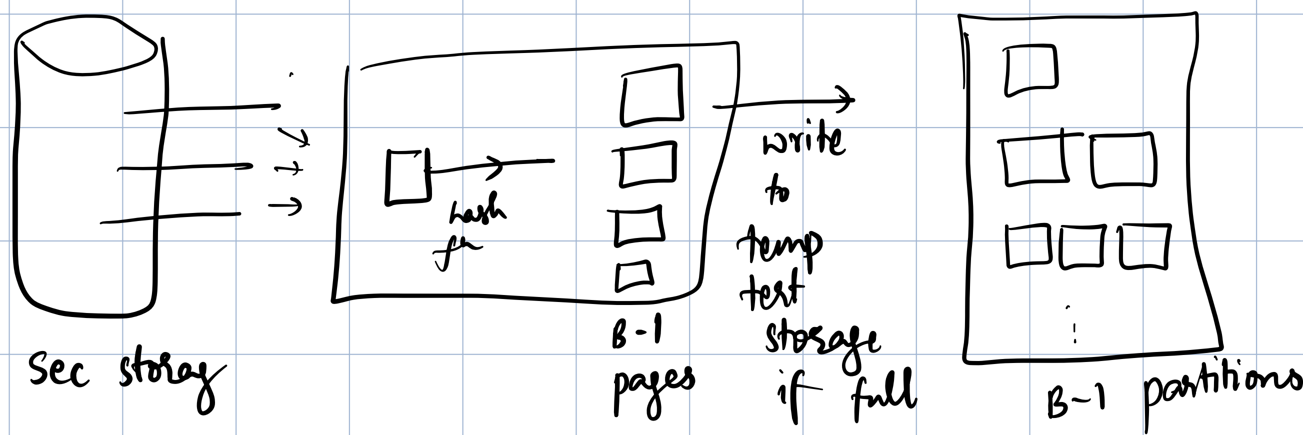
Projection based on sorting

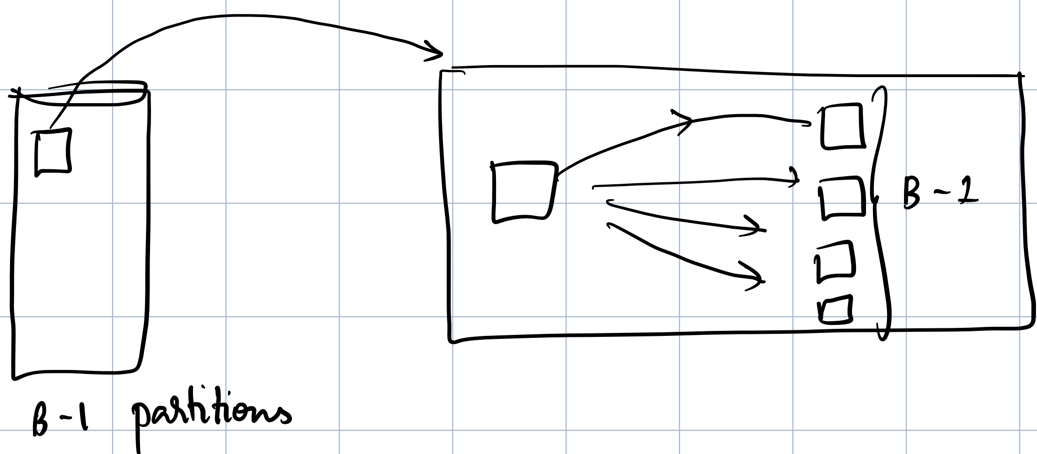
① M pages scan + T pages write

② Sort T pages

③ Scan sorted result

28 Feb 2025


Sec storag
hash fn
B-1 pages
write to temp test storage if full
B-1 partitions


B-2
B-1 partitions

Sorting versus Hashing

- Read from book

Join operation

Select *

from Reserves R, Sailors S

where R.sid = S.sid

General method

- → Start with only a file and think of how you will write a C program
- → Try modifying your solution