CS222/CS122C: Principles of Data Management

UCI, Fall 2019
Notes #10

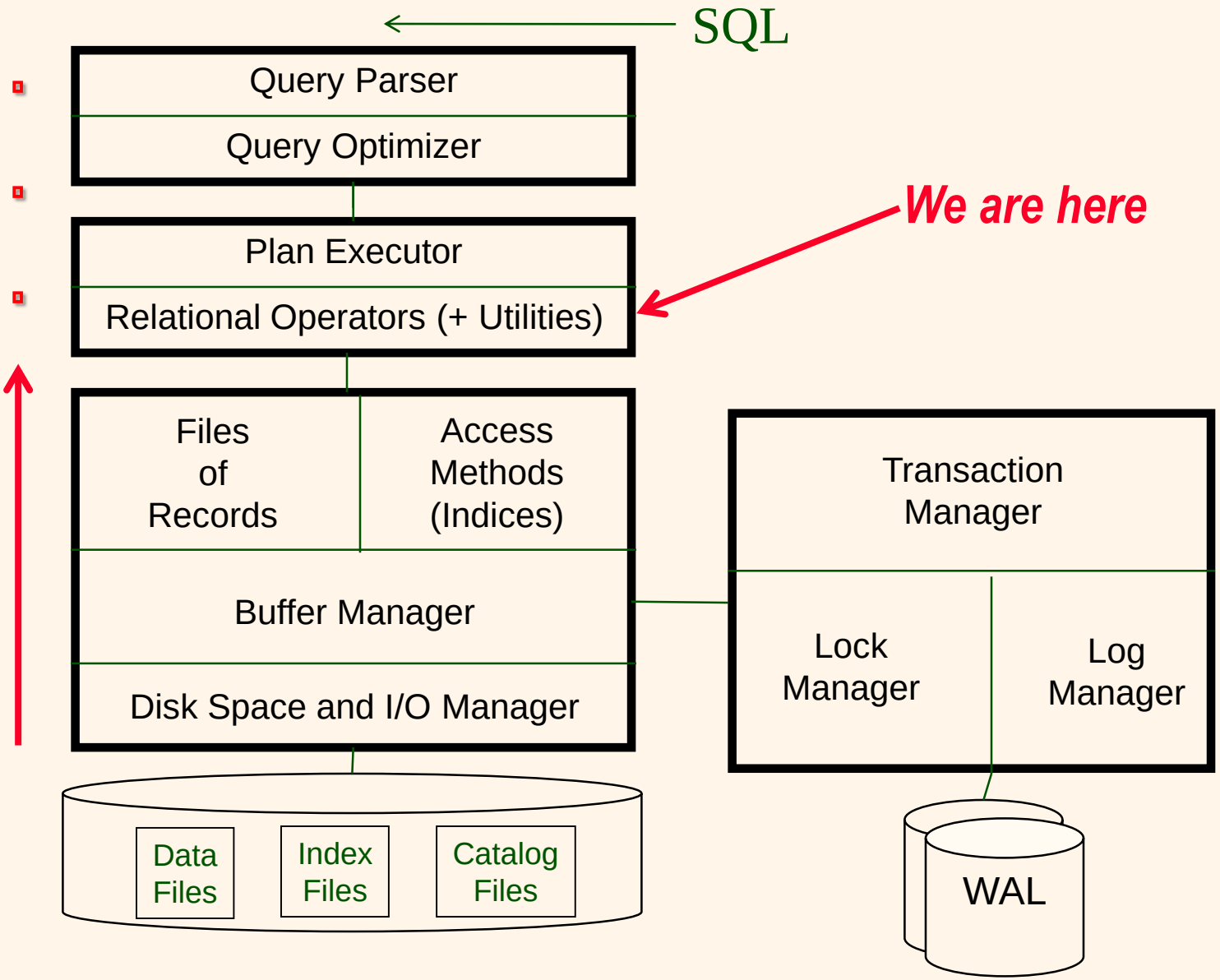
Selection, Projection (DINSTINCT)

Instructor: Chen Li

Memory requirement for 2-pass external sort

- What's the minimum memory requirement?
- N/B <= B – 1
- So roughly B >= sqrt(|N|)

DBMS Structure (from Lecture 1)



Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
    (Total cardinality is thus 100,000 reservations)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
    (Total cardinality is thus 40,000 sailing club members)

Using an Index for Selections

- Cost depends on #qualifying tuples and clustering.
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering).

    Ex: SELECT * FROM WHERE R.rname < 'C%'

  - Assuming a uniform distribution of names, about 10% of tuples qualify (100 pages, 10,000 tuples). With a clustered index, cost is 100+ I/Os; if unclustered, cost is 10,000+ I/Os!
- *Important refinement for unclustered indexes*:
  1. Use index (rname) to find qualifying data entries (→ rids).
  2. *Sort* rid's of the data records to be retrieved.
  3. Now fetch w/rids *in order*. Each data page accessed once (though # of data pages higher than w/clustering).

A Note on Complex Selections

(day<8/9/94 AND rname= 'Paul') OR bid=5 OR sid=3

- Selection conditions are first converted to be in *conjunctive normal form* (CNF):

  (day<8/9/94 OR bid=5 OR sid=3) ***AND***
  (rname= 'Paul' OR bid=5 OR sid=3)

OR conditions (Disjunctions)

- *(day<8/9/94 OR bid=5 OR sid=3)*
  - File scan
  - Retrieve 3 rid lists using indexes, and take the union
- *(day<8/9/94 OR bid=5) AND sid=3*
  - Use index to get records of "sid = 3"
  - Verify the OR conditions

Approaches to Conjunctive Selections

- First approach: Find the *most selective access path*, retrieve tuples using it, and apply any remaining ("residual") terms that don't match the index
  - *Most selective access path:* An index or file scan that we estimate will require the *fewest page I/Os*.
  - Terms that match this index reduce the number of tuples *retrieved*; other terms then used to discard some retrieved tuples, but don't lower the # of tuples/pages fetched.
  - Consider *day<8/9/94 AND bid=5 AND sid=3*. A B+ tree index on *day* can be used; then *bid=5* and *sid=3* must be checked for each retrieved tuple. Or, a hash index on <*bid, sid*> can be used; *day<8/9/94* must then be checked.

equality

Intersection of Ridlists

- Second approach: (If we have 2 or more matching indexes that use ridlists – or keylists – in their leaves)
  - Get sets of rids of data records using *each* matching index.
  - Then *intersect* these sets of rids.
  - Retrieve those records and apply any remaining terms.
  - Consider *day<8/9/94 AND bid=5 AND sid=3*. If we have a B+ tree index on *day* and an index on *sid*, we can get the rids of records satisfying *day<8/9/94* using the first one, rids of records satisfying *sid=3* using the second one , intersect them, retrieve the records, and check for *bid=5*.

The Projection Operation

("Hard" Part: ***Duplicate Elimination****!)*

```
SELECT   DISTINCT
         R.sid, R.bid
FROM     Reserves R
```

- An approach based on **Sorting**:
  - Modify Pass 0 of external sort to eliminate unwanted fields. Runs of pages are till produced, but tuples in runs are smaller than the input tuples. (Reduction depends on the # and sizes of fields that are dropped.)
  - Modify merging passes to eliminate duplicates (!). Thus, number of sorted result tuples is smaller than the input. (Difference depends on the # of duplicates.)
  - Cost: In Pass 0, read original relation (size M), write out same number of smaller tuples. In merging passes, fewer tuples written out in each pass. Using Reserves example, 1000 input pages become 250 after Pass 0 if 0.25 size ratio.

*Projection Based on **Hashing***

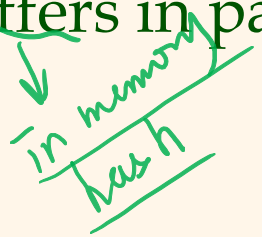
- *Partitioning phase*: Read R using one input buffer. For each tuple, discard unwanted fields and do hash *h1(whole tuple)* to pick one of B-1 output buffers.
  - Result is B-1 partitions (of tuples w/o unwanted fields). 2 tuples in different partitions are sure to be distinct (!).
- *Duplicate elimination phase*: For each partition, read it and build an in-memory hash table using hash *h2* (<> *h1*) on *all fields*, discarding duplicates (!).
  - If partition does not fit in memory, apply hash-based projection algorithm recursively to this partition.
- Cost: For partitioning, read R, write out each tuple, but with fewer fields. Less data read in next phase.

Discussion of Projection

- Sort-based approach is the standard; less impacted by skew and result is sorted (a potential bonus).
- If (any) index on the relation contains all wanted attributes in its search key, can do *index-only* scan.
  - Apply projection techniques to data entries (much smaller!)
- If an ordered (e.g., B+ tree) index contains all wanted attributes as *prefix* of search key, that's even better:
  - Retrieve data in order (index-only scan), discard unwanted fields, compare adjacent tuples to check for duplicates.

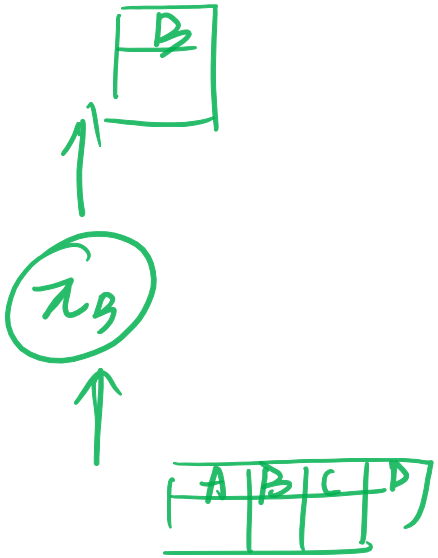
Memory requirement (hashing)

- Minimum pages needed for *2-pass* operation: sqrt(|N|)
  - Let *B* be the # of partitions that we divide the problem into.
  - We'll need *B-1* output buffers to do the dividing work in pass 1
  - Resulting size of each of pass 2's input partitions ≈ |N|/*(B-1)*, and this much data (one partition) must fit in memory in pass 2.
  - If we have *B* buffers in pass 2, need (|N|/*B-1*) ≤ *B*, so |N| <= B^2.



Projection:

Duplicate elimination

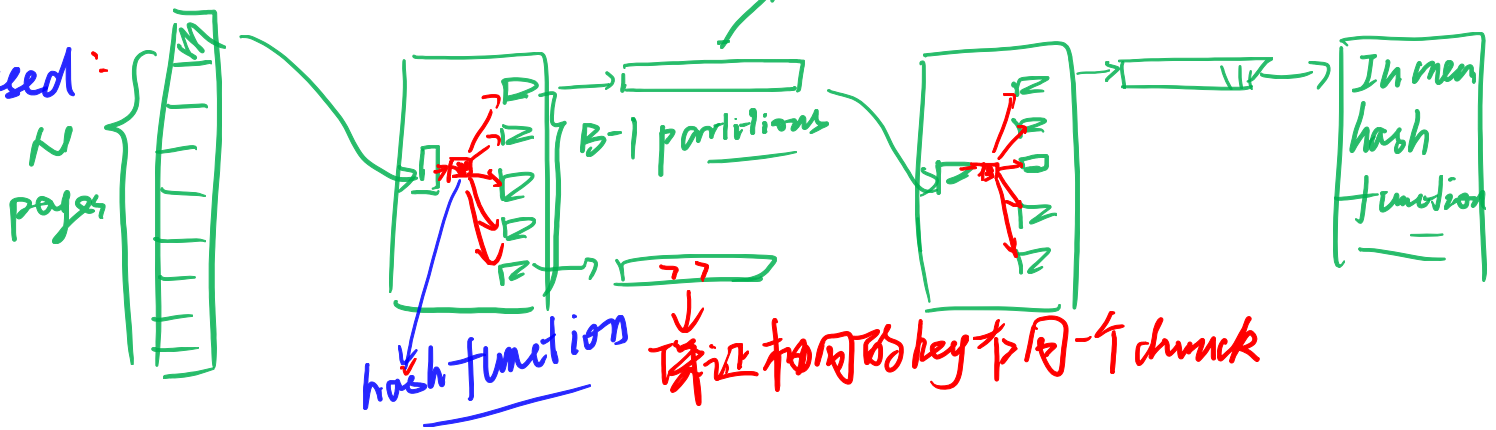


Select distinct (B)
from table

If we can't feed all B into memory

{ Sort-based = external sort

Hash-based:



⇒ hash function should could eliminate the duplicated key inside page

Selection σ_C

Complicated Selection Conditions = (Boolean exp)

(price < 3500 & brand = 'apple') OR (price < 1000 & brand = 'Samsung')

conjunct

conjunction

disjunct

disjunction

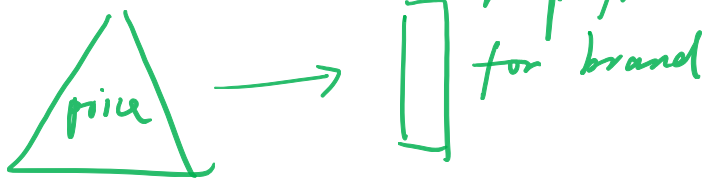
union

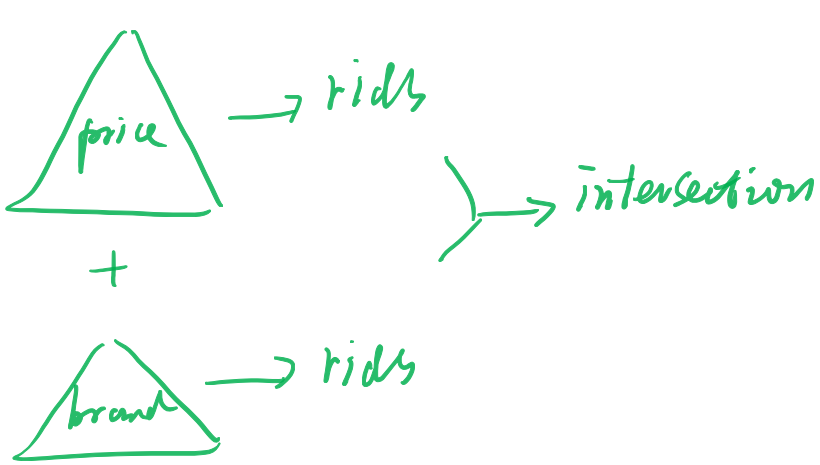
Normalization → Disjunctive Normal Form. → Deal with each conjunction and combine them.

Conjunction Normal Form: (A or B) & (C or D)

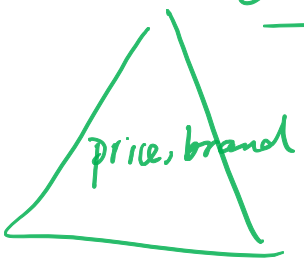
price < 1500 & brand = 'apple' : 4 ways

① scan

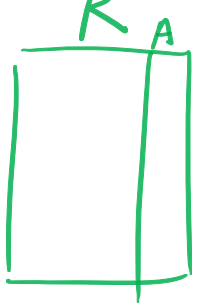
② 

③ 

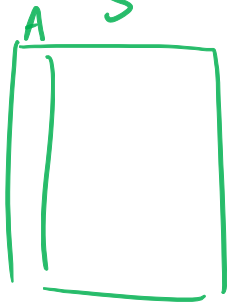
④ Composite index



join:



∞ $R.A = S.A$



no index

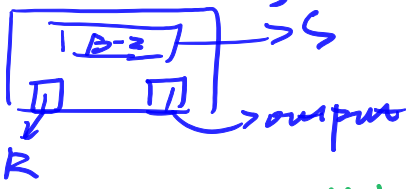
1) Simple Nested Loop join.

(for each record in Sailor, scan Reserves table)

Disk IO: $M + M \cdot 80 \cdot N$

2) Block Nested Loop join.

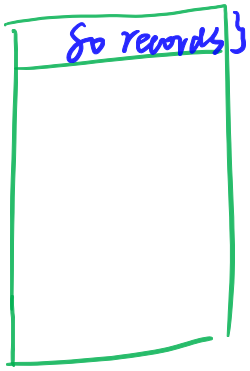
(hold (B-2) pages into memory, then scan Reserve-table)



Disk IO: $M + \frac{M}{B-2} \cdot N$

Each page of R, do join with B-2 page of S in memory

Sailors



M
500 pages

Reserves

N
1000 pages

3) Index-based Nested Loop join.

Reserves table is index-based

$M + M \times 80 \times (1.2 + \lceil \frac{100 N}{80 M} \rceil)$

average # of reserve per seller

1.2 ↓

B+ tree search IO

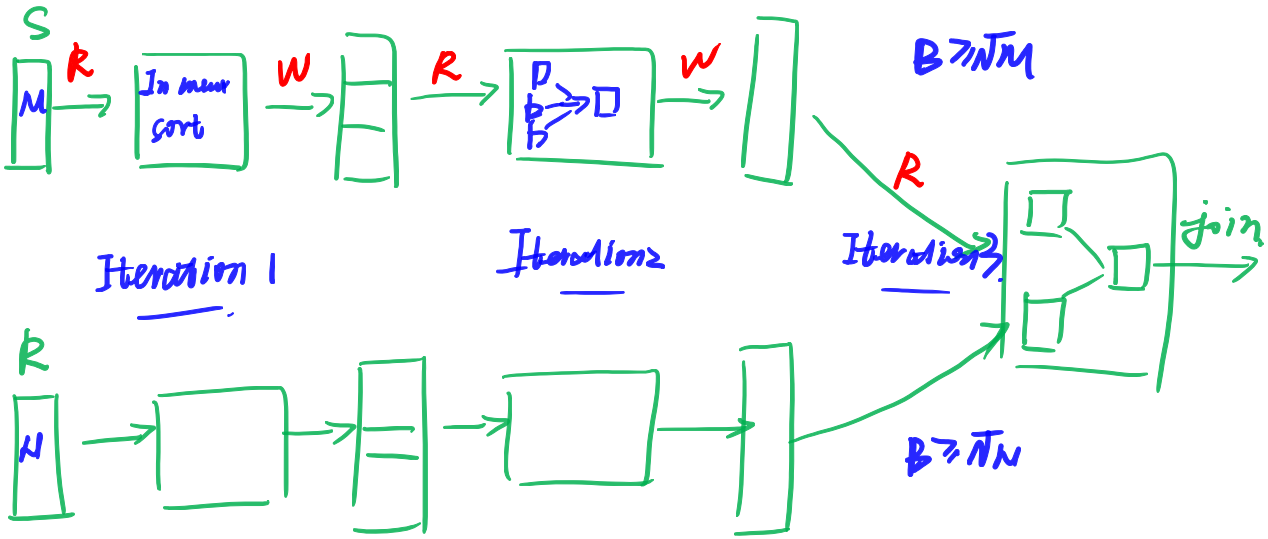
Each sailor could have multiple reserves

Each reserve can only have on sailor.

- Joins
  - Simple nested Loop join — $M + M \cdot So \cdot N$
  - Blocked NL join — $M + \frac{M}{B-2} \cdot N$
  - Index NL join — $M + M * So * (1.2 + T\lceil \frac{100N}{So*M} \rceil)$ → this is better when one of table is pretty small, otherwise blocked NL join

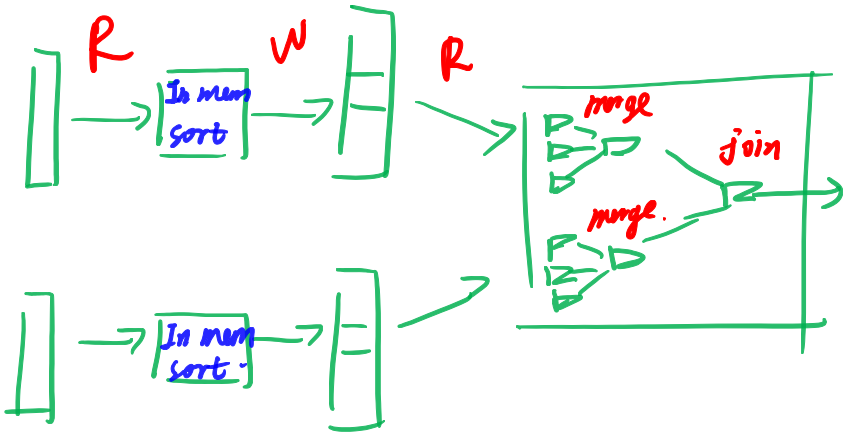
  → Use index file
  - Sort-merge join
  - Grace Hash join
  - "Simple" Hash join
  - Hybrid Hash join

Sort-merge Join: disk IO: $3(M+N)$



If merge iteration 2 & 3 together.

$3(M+N)$ < Disk IO



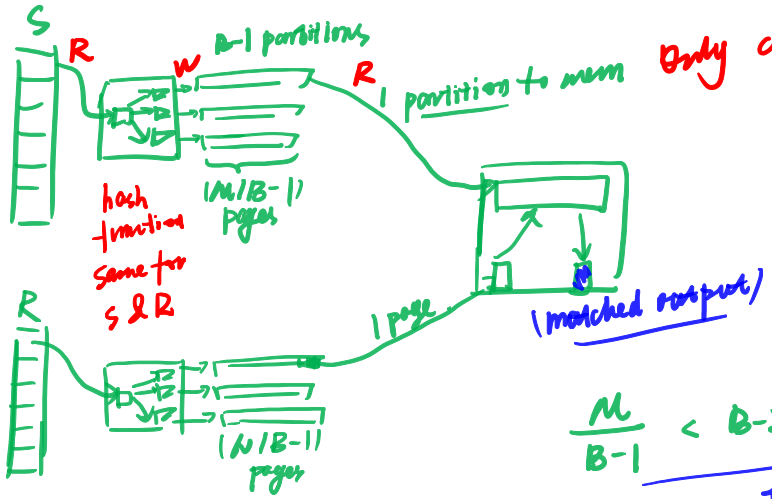
$\frac{M}{B} + \frac{N}{B} + 1 \le B$

$\Rightarrow B \ge \sqrt{M+N}$

Sorted-method only only gives join result but also give us an sorted result, → need more mem space..

Hash-based join.

$3(M+N)$ < Disk IO



"Grace Hash join."

lower mem requirement than sort merge

$B \ge \sqrt{\max(M,N)}$

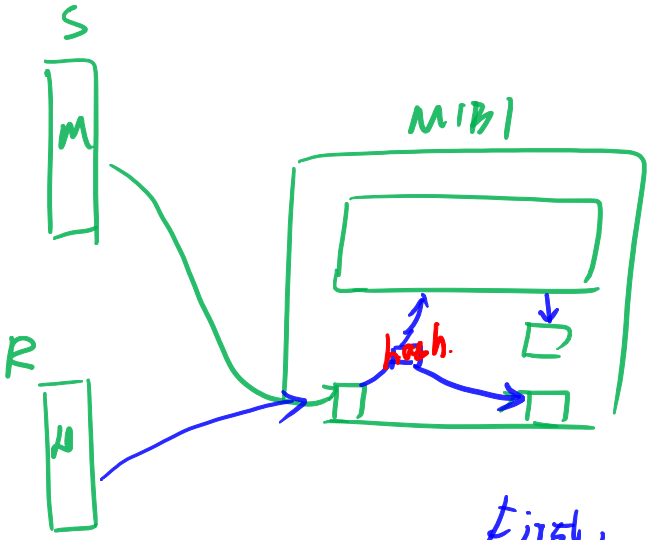
$\frac{M}{B-1} < B-2 \rightarrow B \ge \sqrt{M}$

If load partition of R to mem

$\frac{N}{B-1} < B-2$ $B \ge \sqrt{N}$

Simple Hash join:

situation: $M > B \gg \sqrt{M}$ Assume, Leaf of Records could fit in mem

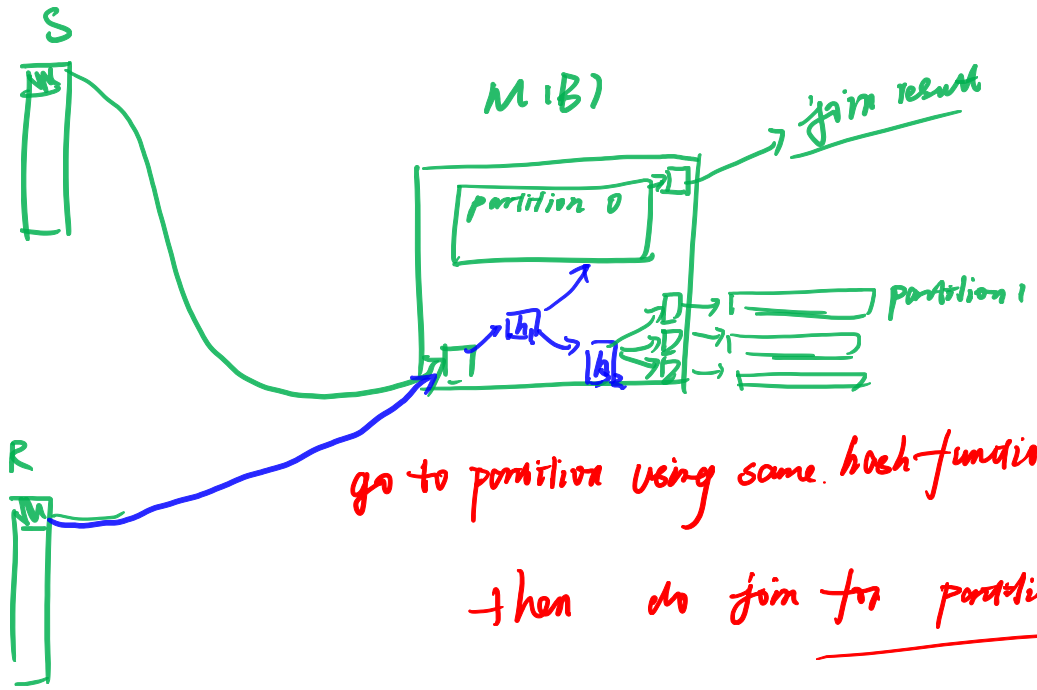


Disk IO $(\frac{M}{2} + \frac{N}{2})$ → read

$+ (\frac{M}{2} + \frac{N}{2}) \cdot 3$ → read write read.

First, hash all pages of S, and last pages stay in mem, then input each page of R, and use the same hash function, do join or write to disk. After finish the R, then hash the remaining last of S back into mem using hash function, the input the left page of R.

Hybrid Hash Join : If mem can't fit all pages



h_1 : wether stay in mem

h_2 : different hash function

go to partition using same hash functions. → to memory or disk (partition 1, 2, ...)

→ then do join for partition 1 & partition 1'