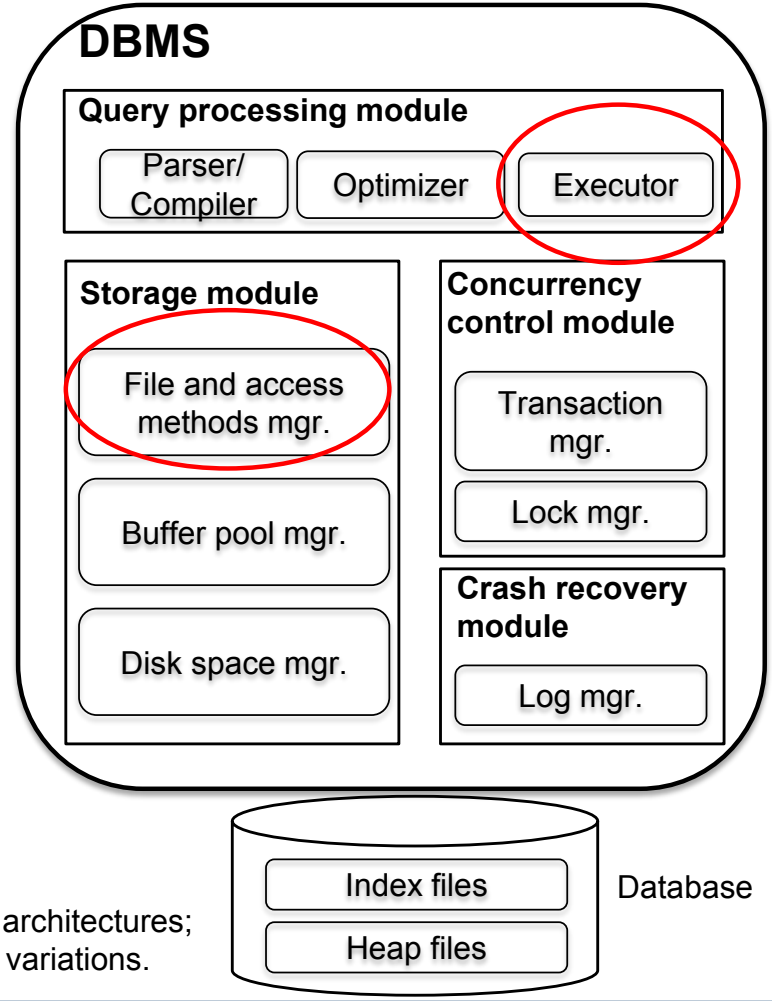
INFO20003 Database Systems

Dr Renata Borovica-Gajic

Lecture 12
Query Processing Part II

Week 6

Remember this? Components of a DBMS



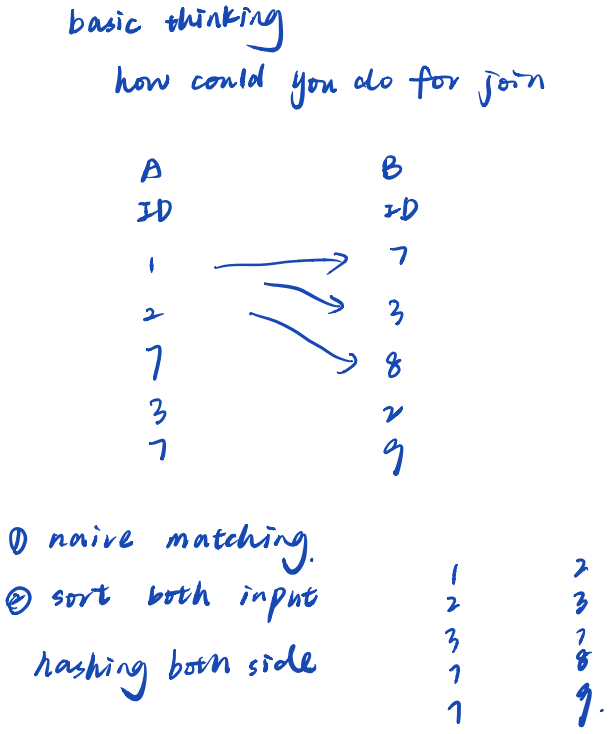
TODAY
Joins

Will briefly touch upon …

This is one of several possible architectures; each system has its own slight variations.

Coverage

- Nested loops join
- Sort-merge join
- Hash join
- General joins



Readings: Chapter 14, Ramakrishnan & Gehrke, Database Systems

Joins

- Are very common and can be **very** expensive (cross product in the worst case)
- There are many implementation techniques for join operations
- Join techniques we will cover:
  1. Nested-loops join
  2. Sort-merge join
  3. Hash join

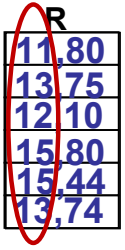
Equality Joins With One Join Column

Example:

```
SELECT  *
   FROM Reserves R1, Sailors S1
WHERE  R1.sid=S1.sid
```

- In algebra: R ⋈ S. They are very common and need to be carefully optimized.
- R X S is large; so, R X S followed by a selection is inefficient.

Left / Outer



Right / Inner

- Join is associative and commutative:
  - AxB == BxA
  - Ax(BxC)==(AxB)xC

- **Cost metric** : Number of pages; Number of I/O

Schema for Examples

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

- **Sailors (S)**:
  - 80 tuples per page, **500 pages**
  - NPages(S) = 500, NTuplesPerPage(S) = 80
  - NTuples(S) = 500*80 = 40000

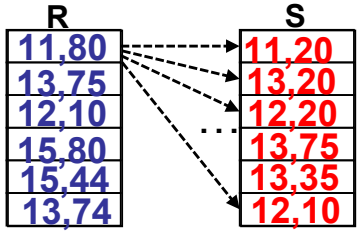
- **Reserves (R)**:
  - 100 tuples per page, **1000 pages**
  - NPages(R) = 1000, NTuplesPerPage(R) =100
  - NTuples(R) = 100000

Simple Nested Loops Join

- For each tuple in the *outer* relation R, we scan the entire *inner* relation S

Pseudo code:

foreach tuple r in **R** do
    foreach tuple s in **S** do
        if r_i == s_j then add <r, s> to result



- Cost:

left

Cost (SNJL) = NPages(Outer) +
NTuples(Outer) * NPages(Inner)

for each record in the left, traverse inner once

- **Our example:**

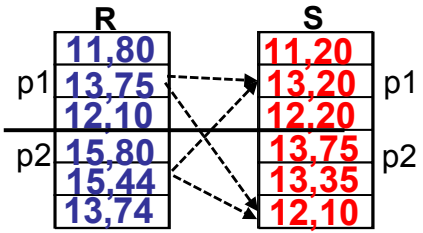
Cost (SNLJ)= 1000+ 100*1000*500
= 50001000 (I/O)

Page-Oriented Nested Loops Join

- For each *page* of R
  - –get each *page* of S
  - –write out matching pairs of tuples <r, s>, where r is in R-page and S is in S-page

Pseudo code:

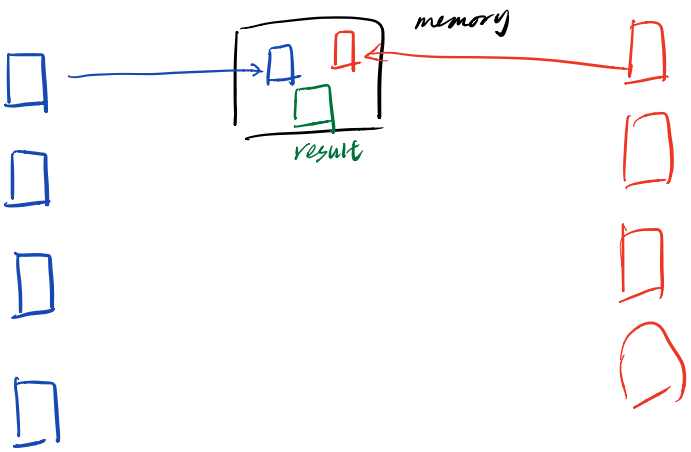
foreach page b_R in R do
  foreach page b_S in S do
    foreach tuple r in b_R do
      foreach tuple s in $b_S$do
        if r_i == s_j then add <r, s> to result



Cost (PNJL) = NPages(Outer) + NPages(Outer) * NPages(Inner)

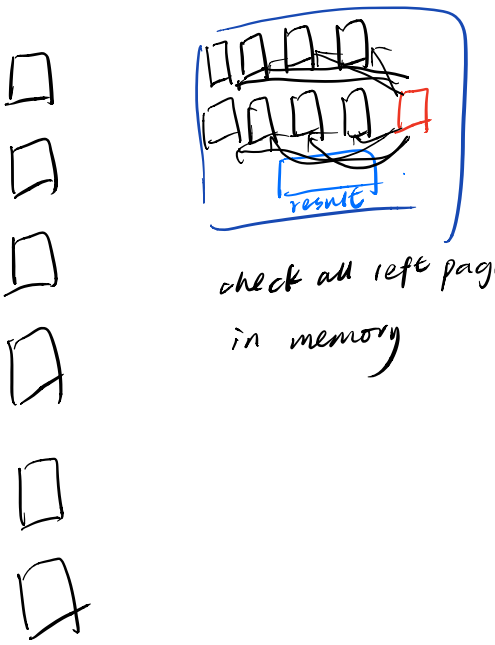
- **Our example:**

Cost (PNLJ)= 1000+1000*500 = 501000 (I/O)



⇒ in reality, have more space to store one page from left and one page from right in memory

update

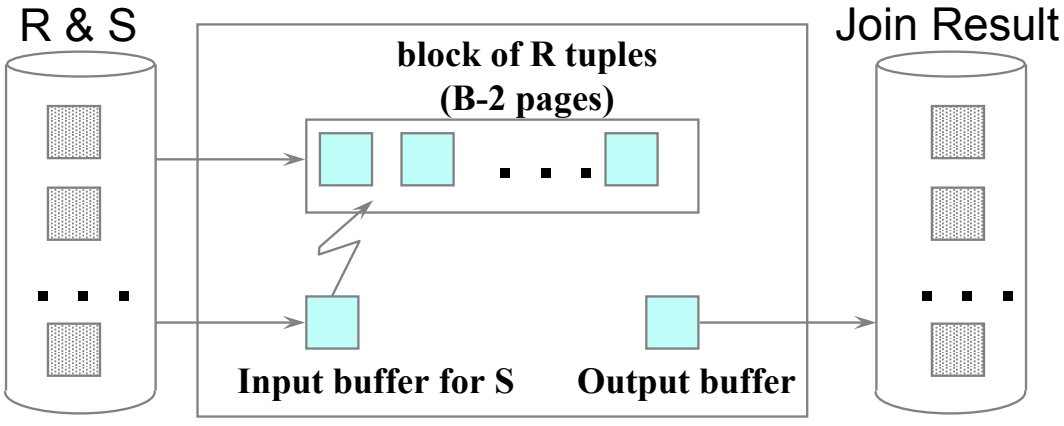


check all left page in memory

then reduce to just 2 passes

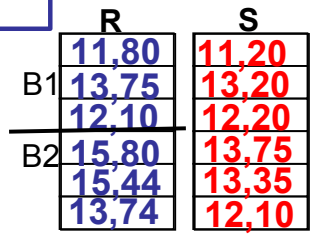
Block Nested Loops Join

- Page-oriented NL doesn't exploit extra memory buffers
- Alternative approach:
  - Use one page as an input buffer for scanning the inner S, one page as the output buffer, and use all remaining pages to hold 'block' of outer R
- For each matching tuple r in R-block, s in S-page, add <r, s> to result. Then read next R-block, scan S, etc



Block Nested Loops Join Cost

Cost (BNJL) = NPages(Outer) +
NBlocks(Outer) * NPages(Inner)



- **NBlocks(Outer) =** $\left\lceil \frac{NPages(Outer)}{B-2} \right\rceil$

Npasses

one input of right
one output

- **Our example**:

Let's say we have 102 pages of space in memory, and consider Reserves (R) as the outer and Sailors (S) as the inner table.

NBlocks(R) = 1000/(102-2) = 10
Cost(BNLJ) = 1000 + 10* 500 = 6000 I/O

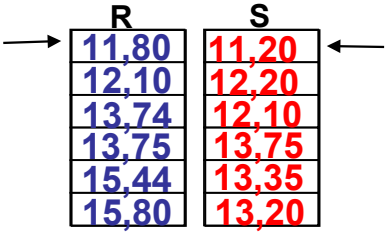
Query Processing: Joins

- Nested loops join
- Sort-merge join
- Hash join
- General joins

Readings: Chapter 14, Ramakrishnan & Gehrke, Database Systems

Sort-Merge Join ($R \bowtie_{i=j} S$)

- **Sort** R and S on the join column, then scan them to do a **merge** (on join column), and output result tuples



once data is sorted,
just do one pass over both inputs

cost for sorting is expensive

- Sorted R is scanned once;
- Each S group of the same key values is scanned once per matching R tuple (typically means Sorted S is scanned once too).

- Useful when:
  - one or both inputs are already sorted on join attribute(s)
  - output is required to be sorted on join attributes(s)

Sort-Merge Join Cost

Cost (SMJ) = Sort(Outer) + Sort(Inner)
+ NPages(Outer) + NPages(Inner)

Sort inputs
Merge inputs

Sort(R) = External Sort Cost **= 2*NumPasses*NPages(R)**

read & write

Our example:
Let's say that both Reserves and Sailors can be sorted in 2 passes, then:

Cost(SMJ) = Sort R + Sort S + NPages(R) + NPages(S)
= 2*2*NPages(R)+ 2*2*NPages(S)
+ NPages(R) + NPages (S)
= 5*1000 + 5* 500 = 7500 I/O

Query Processing: Joins

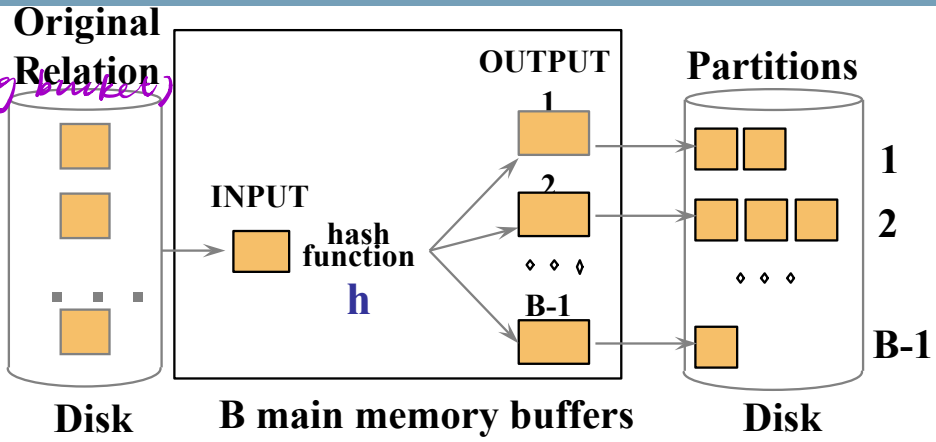
- Nested loops join
- Sort-merge join
- Hash join
- General joins

Readings: Chapter 14, Ramakrishnan & Gehrke, Database Systems

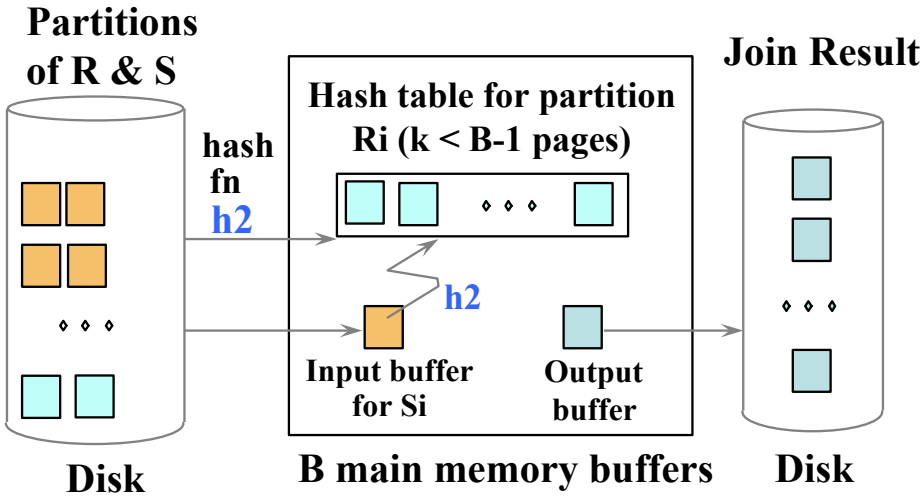
Hash-Join

(match will only in the corresponding bucket)

- Partition both relations using hash function h: R tuples in partition I will only match S tuples in partition I



- Read in a partition of R, hash it using h2 (<> h!). Scan matching partition of S, probe hash table for matches



Hash-Join Cost

1. In partitioning phase, we read+write both relations
2. In matching phase, we read both relations

read+write read+write

Cost (HJ) = 2 * NPages(Outer) + 2* NPages(Inner) Create partitions
+ NPages(Outer) + NPages(Inner) Match partitions

- **Our example:**

Cost(HJ) = 2*NPages(R) + 2*NPages(S) + NPages(R) + NPages(S)
= 3 * 1000 + 3* 500 = 4500 I/Os

Watch this video if you are confused

https://www.youtube.com/watch?v=o1dMJ6-CKzU

From 0:58

Query Processing: Joins

- Nested loops join
- Sort-merge join
- Hash join
- General joins

Readings: Chapter 14, Ramakrishnan & Gehrke, Database Systems

General Join Conditions

- Equalities over several attributes (e.g., *R.sid=S.sid* AND *R.rname=S.sname*):
  - For Sort-Merge and Hash Join, sort/partition on combination of the two join columns

- Inequality conditions (e.g., *R.rname < S.sname*):
  - Hash Join, Sort Merge Join not applicable
  - Block NL quite likely to be the best join method here

↑ nested loop

Summary

- A virtue of relational DBMSs:
  - Queries are composed of a few basic operators
  - Implementation of operators can be carefully tuned
  - Important to do this
- Many alternative implementations for each operator
  - No universally superior technique for most operators
- Must consider alternatives for each operation in a query and choose best one based on system statistics…
  - Part of the broader task of optimizing a query composed of several operations

What's examinable

- Understand alternatives for join operator implementations
  - Be able to calculate the cost of alternatives
- Important for Assignment 3 as well

Next Lecture

- Query optimization
  - How does a DBMS pick a good query plan?