Database Applications (15-415)

DBMS Internals- Part VI
Lecture 14, March 12, 2014

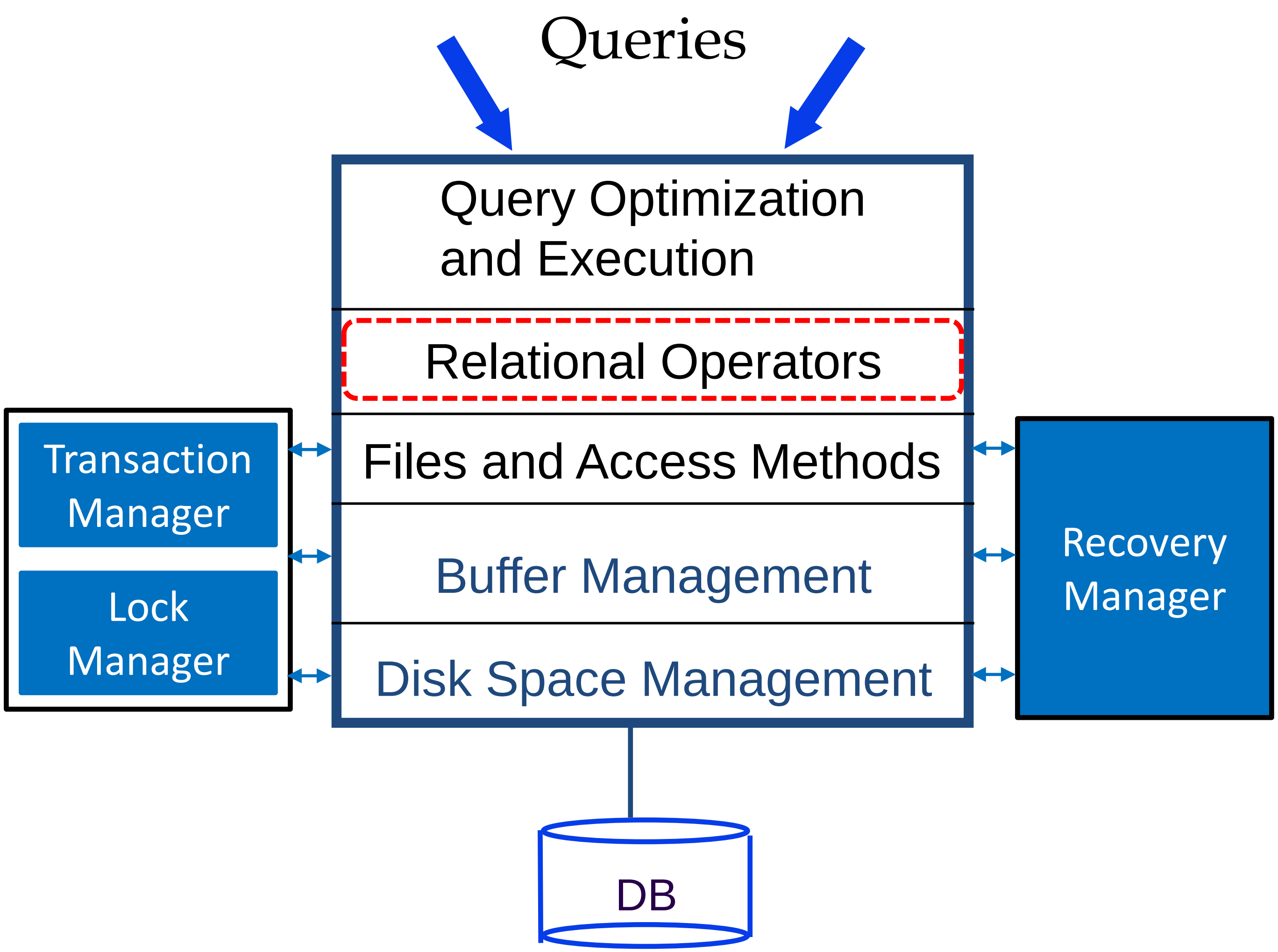
Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Today…

- Last Session:
  - DBMS Internals- Part V
    - Hash-based indexes (Cont'd) and External Sorting
- Today's Session:
  - DBMS Internals- Part VI
    - Algorithms for Relational Operations
- Announcements:
  - Project 2 is due on March 15 (NOT 13) by midnight
  - We will solve the midterm exam tomorrow at the recitation. Please bring any question about project 2 as well.

DBMS Layers


Queries
Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
Transaction Manager
Lock Manager
Recovery Manager
DB

Outline

- Introduction ✓
- The Selection Operation
- The Projection Operation
- The Join Operation

Relational Operations

- We will consider how to implement:
  - *Selection* (σ)
  - *Projection* (π)
  - *Join* ($\bowtie$)
  - *Set-difference* ($-$)
  - *Union* ($\cup$)
  - *Aggregation* (SUM, MIN, etc.) and GROUP BY

- Since each operation returns a relation, ops can be *composed*!

- After we cover how to implement operations, we will discuss how to *optimize* queries (formed by composing operators)

Assumptions

- We assume the following two relations:

Sailors (<u>*sid*: integer</u>, *sname*: string, *rating*: integer, *age*: real)

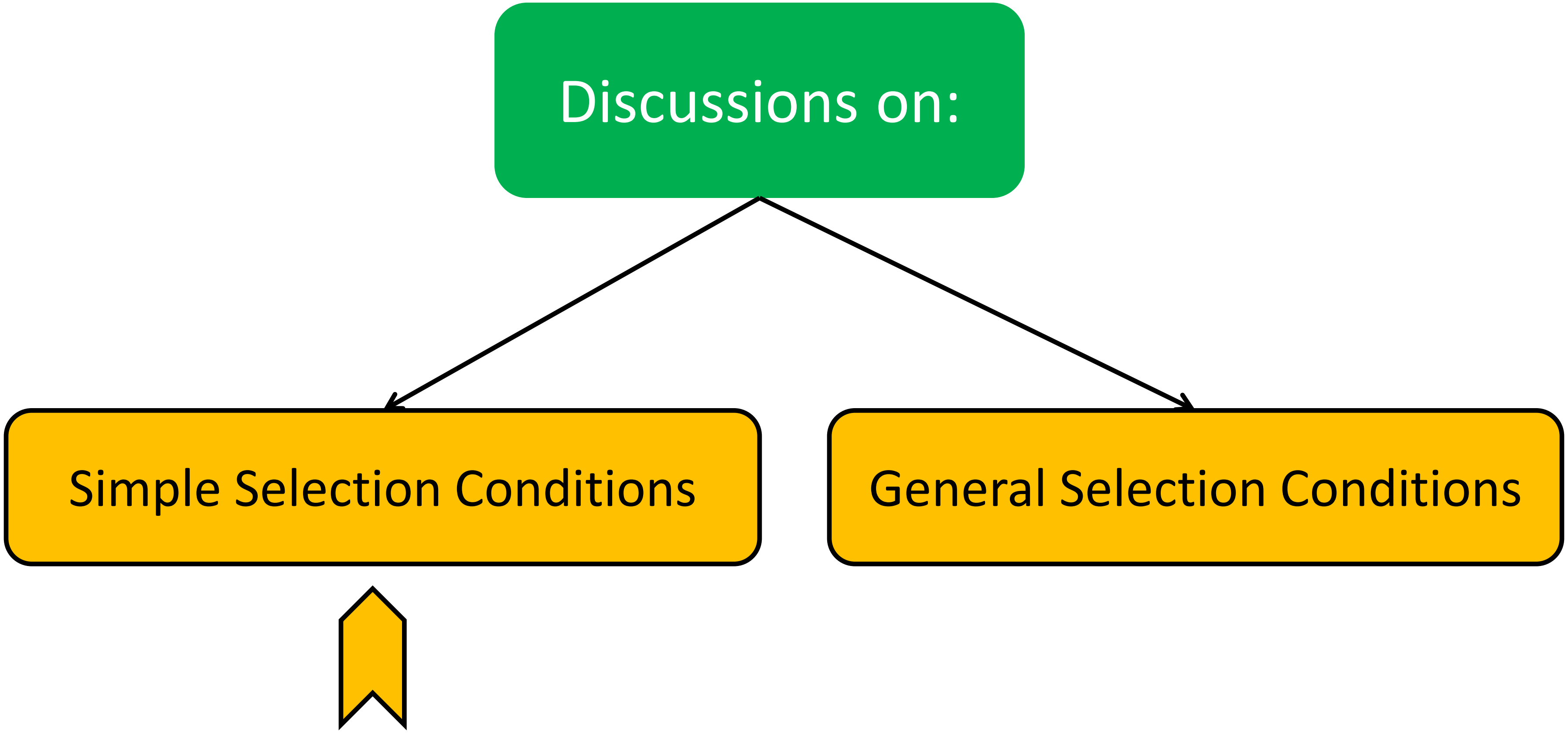
Reserves (<u>*sid*: integer, *bid*: integer, *day*: dates</u>, *rname*: string)

- For Reserves, we assume:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages
- For Sailors, we assume:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages
- Our cost metric is the number of I/Os
- We ignore the computational and output costs

Outline

- Introduction
- The Selection Operation ✓
- The Projection Operation
- The Join Operation

The Selection Operation



The Selection Operation: Basic Approach

- Consider the following selection query, Q:

```
SELECT *
FROM Reserves R
WHERE R.rname = 'Joe'
```

- How can we evaluate Q?
  - Scan Reserves entirely
  - Check the condition on each tuple
  - Add the tuple to the result if the condition is satisfied

- What is the I/O cost?
  - 1000 I/Os (since Reserves contains 1000 pages)!

Can we do better?

How to Improve Upon the Basic Approach for Selections?

- We can utilize the information in the selection condition and use an index (if a *suitable* index is available)

- For instance, a B+ tree index on *rname* can be used to answer Q considerably faster
  - But, an index on bid (for example) would not be useful!

- Different data organizations dictate different evaluations for the selection operation:
  - No Index, Unsorted Data
  - No Index, Sorted Data
  - B+ Tree Index
  - Hash Index

No Index, Unsorted Data

- Assume a selection operation of the form:

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

- If there is no index on ***R.attr*** and ***R*** is not sorted, we have to scan ***R*** entirely

- Therefore, the *most selective access path* is a file scan

- During the file scan, for each tuple, we test the condition ***R.attr op value*** and add the tuple to the result if the condition is satisfied (*this is the basic approach!*)

No Index, Sorted Data

- Assume a selection operation of the form:

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

- What can be done if there is no index on ***R.attr*** but ***R*** is sorted?
  - Do a binary search to locate the first tuple
  - Start at the located tuple and scan ***R*** until the selection condition is no more satisfied

- Therefore, the most selective access path is a sorted-file scan

- I/O cost = O($\log_2$ ***M***) + scan cost (which can vary from 0 to ***M***)

B+ Tree Index

- Assume a selection operation of the form:

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

- What can be done if there is a B+ tree index on ***R.attr***?
  - Search the tree to locate the first index entry that points to a qualifying tuple of ***R*** (STEP 1)
  - Scan the leaf pages to retrieve all entries in which the key value satisfies the selection condition (STEP 2)

- What would be the I/O cost?
  - STEP 1: 2 or 3 I/Os
  - STEP 2: Depends on the number of qualifying tuples, the employed alternative and whether the index is clustered

B+ Tree Index (Cont'd)

- What if the index uses *Alternative (1)*?
  - The leaf pages contain the actual tuples and no additional cost is incurred
- What if the index is *clustered* and uses *Alternative (2)* or *(3)*?
  - Best case: 1 I/O
  - Worst case: # of leaf <u>pages</u> scanned
- What if the index is *un-clustered* and uses *Alternative (2)* or *(3)*?
  - Each index entry can point to a qualifying tuple on a different page
  - Cost = 1 I/O <u>per a qualifying tuple</u>!
  - Can we do better?

B+ Tree Index (Cont'd)

- Important refinement for un-clustered indexes:
  - Find qualifying index entries
  - Sort the rids by their page-id component
  - Read tuples in order

- This ensures that each data page is fetched just once

- I/O Cost = 1 I/O per a data page (vs. 1 I/O per a qualifying tuple)!

Hash Index

- Assume an "equality" selection operation ***S*** of the form:

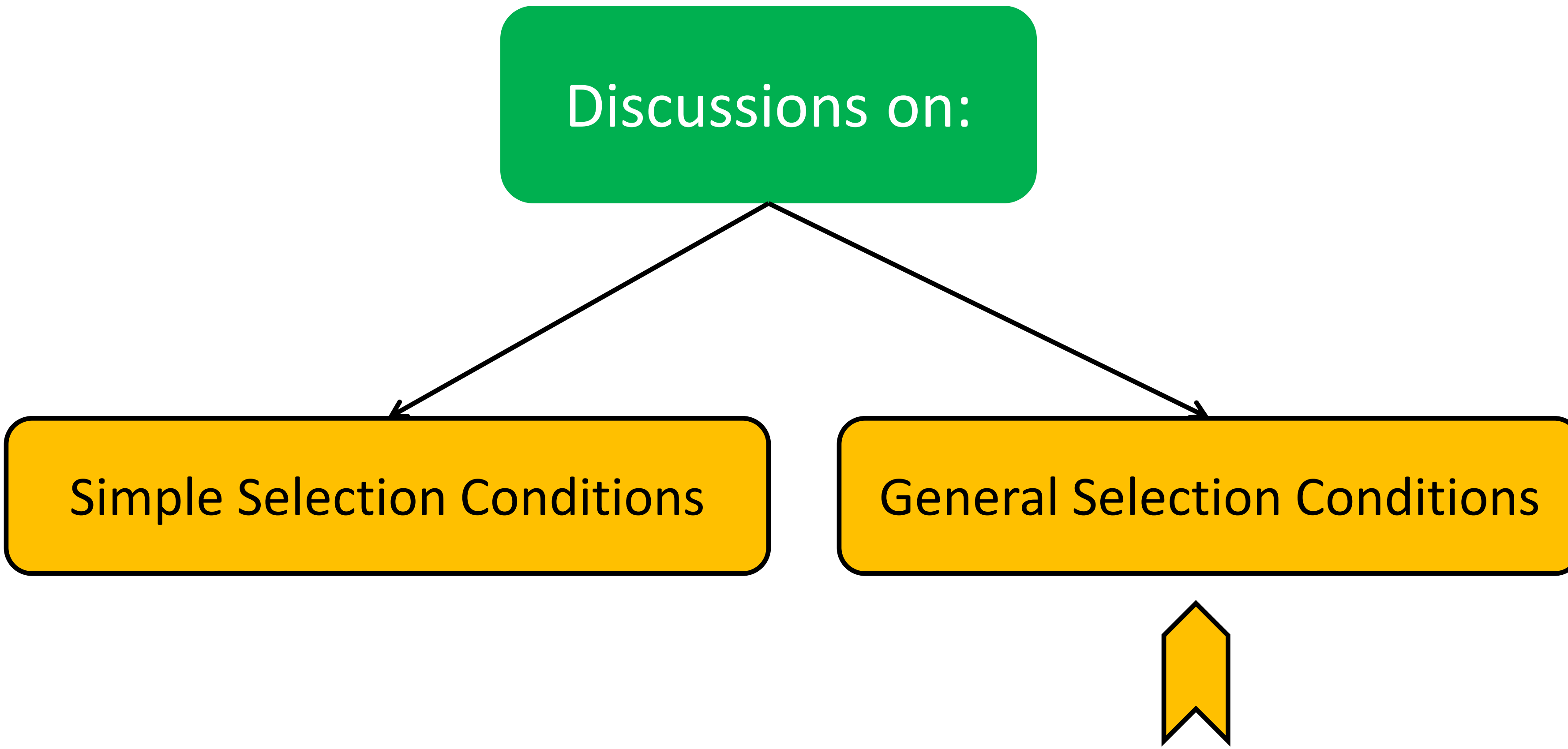
$$\sigma_{R.attr\ =\ value}\ (R)$$

- The best way to implement ***S*** is to use a *hash index* (if available on ***R.attr***)

- Cost = 1 or 2 I/Os (to retrieve the appropriate bucket page) + # of I/Os to retrieve qualifying tuples (could be 1 or *many*)

- The cost of retrieving qualifying tuples depends on:
  - The number of such tuples
  - Whether the index is clustered or un-clustered!

The Selection Operation



General Selection Conditions

- Thus far, we have considered only simple selection conditions of the form *R.attr op value*

- In general, a selection condition is an expression with logical connectives (i.e., ^ and ˅) of *terms*
  - E.g., *R.rname = 'Joe' ^ R.bid=r (R)*

- A selection with *conjunctions* of conditions is said to be in *Conjunctive Normal Form* (*CNF*) and each condition is called a *conjunct*

- A conjunct can contain disjunctions and is said to be *disjunctive*

General Selection Conditions (Cont'd)

- Selection conditions that contain disjunctive conjuncts can be rewritten in CNF
  - E.g., (day < 8/9/02 ^ rname = 'Joe') ˅ bid=5 ˅ sid=3 is equivalent to (day < 8/9/02 ˅ bid=5 ˅ sid=3) ^ (rname = 'Joe' ˅ bid=5 ˅ sid=3)

- A tree index *matches* a CNF selection if conjuncts involve attributes in only a *prefix* of the search key
  - E.g., Tree index on *<a, b, c>* *matches* the selection condition *a=5 AND b=3*, and *a=5 AND b>6*, but not *b=3*

- A hash index *matches* a CNF selection if there is a conjunct for every attribute in the index's search key
  - E.g., Hash index on *<a, b, c>* *matches* *a=5 AND b=3 AND c=5*; but it does not match *b=3, or a=5 AND b=3, or a>5 AND b=3 AND c=5*

Two General Cases

- We will discuss general selections:
  - Without Disjunctions
  - With Disjunctions

Two General Cases

- We will discuss general selections:
  - Without Disjunctions
  - With Disjunctions

Evaluating Selections *without* Disjunctions

- There are two approaches to general selections without disjunctions:
  - Approach 1- The Single-Index Approach:
    - Find the *most selective access path*, MSAP
    - Retrieve tuples using MSAP
    - Check for each retrieved tuple any remaining terms which do not match the index

The Single-Index Approach: Examples

- Consider day<8/9/94 AND bid=5 AND sid=3:
  - Example 1:
    - A B+ tree index on *day* is used
    - Then, bid=5 and sid=3 must be checked for each retrieved tuple
  - Example 2:
    - A hash index on *<bid, sid>* is used
    - Then, day<8/9/94 must be checked for each retrieved tuple

Evaluating Selections *without* Disjunctions

- There are two approaches to general selections without disjunctions:
  - Approach 2- The Multiple-Indices Approach:
    - Get sets of rids (assuming Alternative (2) or (3)) using each matching index
    - *Intersect* these sets of rids
    - Retrieve the tuples
    - Check for each retrieved tuple any remaining terms which do not match indices

The Multiple-Indices Approach: An Example

- Consider day<8/9/94 AND bid=5 AND sid=3:
  - If we have a B+ tree index on *day* ($\boldsymbol{I_d}$) and an index on sid ($\boldsymbol{I_s}$), we can:
    - Retrieve *rids* satisfying day<8/9/94 using $\boldsymbol{I_d}$
    - Retrieve *rids* satisfying sid=3 using $\boldsymbol{I_s}$
    - Intersect results
    - Retrieve tuples and check bid=5

Two General Cases

- We will discuss general selections:
  - Without Disjunctions
  - With Disjunctions

Evaluating Selections *with* Disjunctions

- There are <u>three cases</u> to general selections with disjunctions:
  - CASE 1: If a conjunct, ***C***, is a disjunction of terms, and one term requires a file scan, testing ***C*** would require a file scan
  - CASE 2: If the selection condition is CNF and contains a conjunct with disjunctions, we can take advantage of other conjuncts
  - CASE 3: If every term in a disjunction has a matching index, we can retrieve candidate tuples using the indices and *union* them all

Evaluating Selections *with* Disjunctions

- CASE 1: If a conjunct, ***C***, is a disjunction of terms and one term requires a file scan, testing ***C*** would require a file scan

- E.g., Consider *day<8/9/94 OR rname='Joe'* and suppose hash indices on *rname* (i.e., I_1) and *sid* (i.e., I_2), are available
  - We can retrieve tuples satisfying *rname = 'Joe'* using I_1
  - However, *day<8/9/94* requires a file scan
  - Hence, as the file scan is to be done, we can check the condition *rname='Joe'* and preclude using I_1 at first place
  - Therefore, the most selective access path is a file scan *only*

Evaluating Selections *with* Disjunctions

- CASE 2: If the selection condition is CNF and contains a conjunct with a disjunction, we can take advantage of other conjuncts

- E.g., Consider *(day<8/9/94 OR rname='Joe') AND sid=3*. Suppose also the existence of a hash index on *sid* (I_s)
  - We can use I_s to find qualifying tuples on *sid* and check for each retrieved tuple *day<8/9/94 OR rname='Joe'*
  - Therefore, the most selective access path is the index on *sid*

Evaluating Selections *with* Disjunctions

- CASE 3: If every term in a disjunction has a matching index, we can retrieve candidate tuples using the indices and *union* them all

- E.g., Consider *day<8/9/94 OR rname='Joe'* and suppose B+ indices on *day* (i.e., I_1) and *rname* (i.e., I_2), are available
  - We can retrieve tuples satisfying *day<8/9/94* using I_1
  - In addition, we can retrieve tuples satisfying *rname = 'Joe'* using I_2
  - We can subsequently union their results

Q: What if all matching indices use Alternative (2) or (3)?

A: Apply the refinement for un-clustered indices! (see Slide 15)

Outline

- Introduction
- The Selection Operation
- The Projection Operation ✓
- The Join Operation

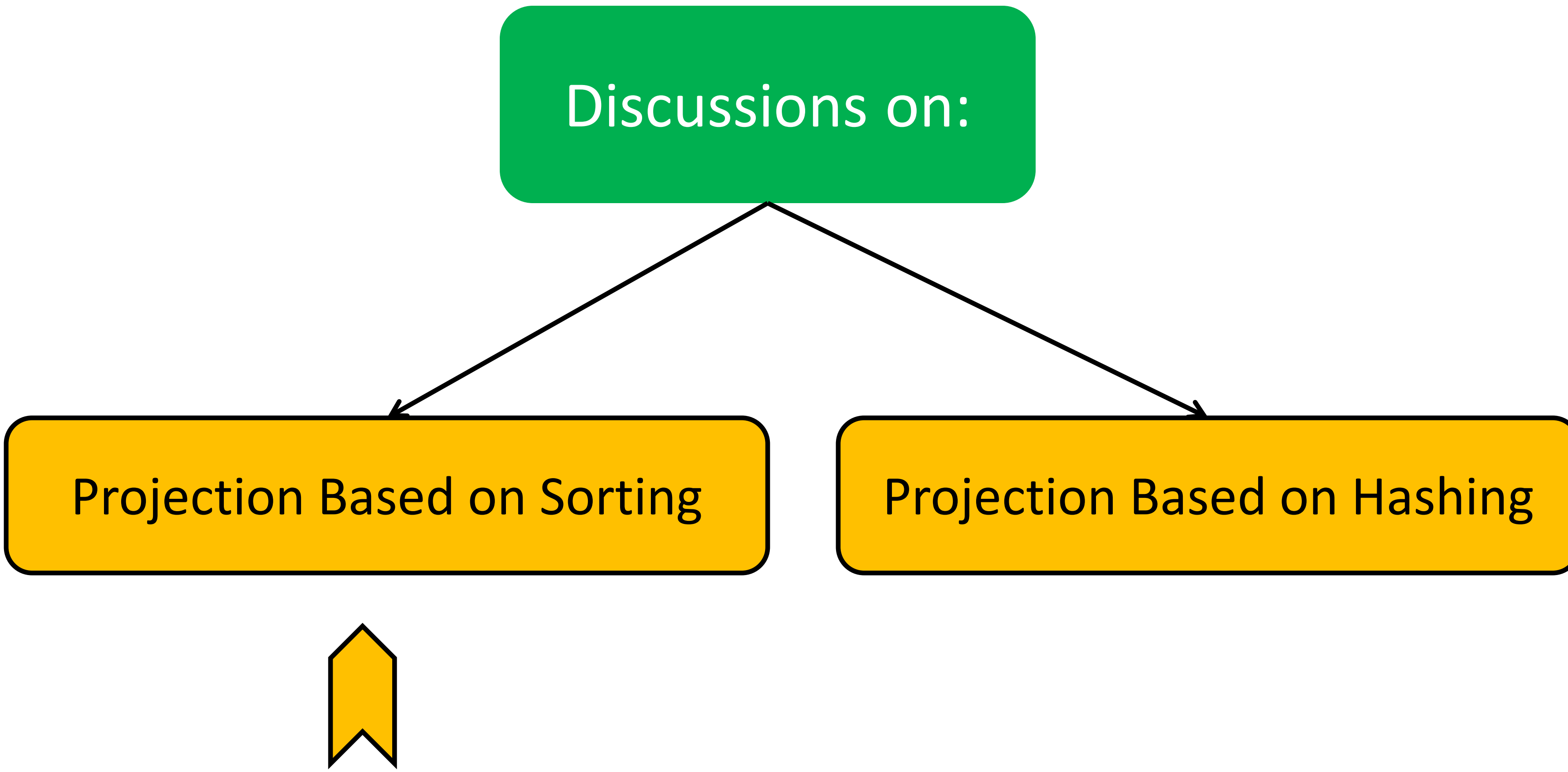
The Projection Operation

- Consider the following query, Q, which implies a projection:

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

- How can we evaluate Q?
  - Scan R and remove unwanted attributes (STEP 1)
  - Eliminate any duplicate tuples (STEP 2)
- STEP2 is difficult and can be pursued using *two* basic algorithms:
  - Projection Based on Sorting
  - Projection Based on Hashing

The Projection Operation



Projection Based on Sorting

- The algorithm based on sorting has the following steps:
  - Step 1: Scan ***R*** and produce a set of tuples, ***S***, which contains only the wanted attributes
  - Step 2: Sort ***S*** using *external sorting*
  - Step 3: Scan the sorted result, compare adjacent tuples, and discard duplicates

- What is the I/O cost (assuming we use *temporary* relations)?
  - Step 1: ***M*** + ***T*** I/Os, where ***M*** is the number of pages of ***R*** and ***T*** is the number of pages of the temporary relation
  - Step 2: 2***T*** × # of passes I/Os
  - Step 3: ***T*** I/Os

The Projection Operation: An Example

- Consider Q again:

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

- How many I/Os would evaluating Q incur?
  - Step 1: ***M*** + ***T*** = 1000 I/Os + 250 I/Os, assuming each tuple written in the temporary relation is 10 bytes long
  - Step 2: if ***B*** (say) is 20, we can sort the temporary relation in 2 passes at a cost of 2×250×2 = 1000 I/Os
  - Step 3: add another 250 I/Os for the scan
  - Total = 2500 I/Os

Can we do better?

Projection Based on *Modified* External Sorting

- Projection based on sorting can be simply done by *modifying* the external sorting algorithm

- How can this be achieved?
  - Pass 0: Project out unwanted attributes
  - Passes 2, 3, etc.: Eliminate duplicates during merging

- What is the I/O cost?
  - Pass 0: ***M*** + ***T*** I/Os
  - Passes 2, 3, etc.: Cost of merging

Projection Based on *Modified* External Sorting: An Example

- Consider Q again:

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

- How many I/Os would evaluating Q incur?
  - Pass 0: ***M*** + ***T*** = 1000 + 250 I/Os
  - Pass 1: read the runs (total of 250 pages) and merge them
  - Grand Total = 1500 I/Os (as opposed to 2500 I/Os using the *unmodified* version!)

The Projection Operation

Discussions on:

- Projection Based on Sorting
- Projection Based on Hashing

Projection Based on Hashing

- The algorithm based on hashing has two phases:
  - Partitioning Phase
  - Duplicate Elimination Phase

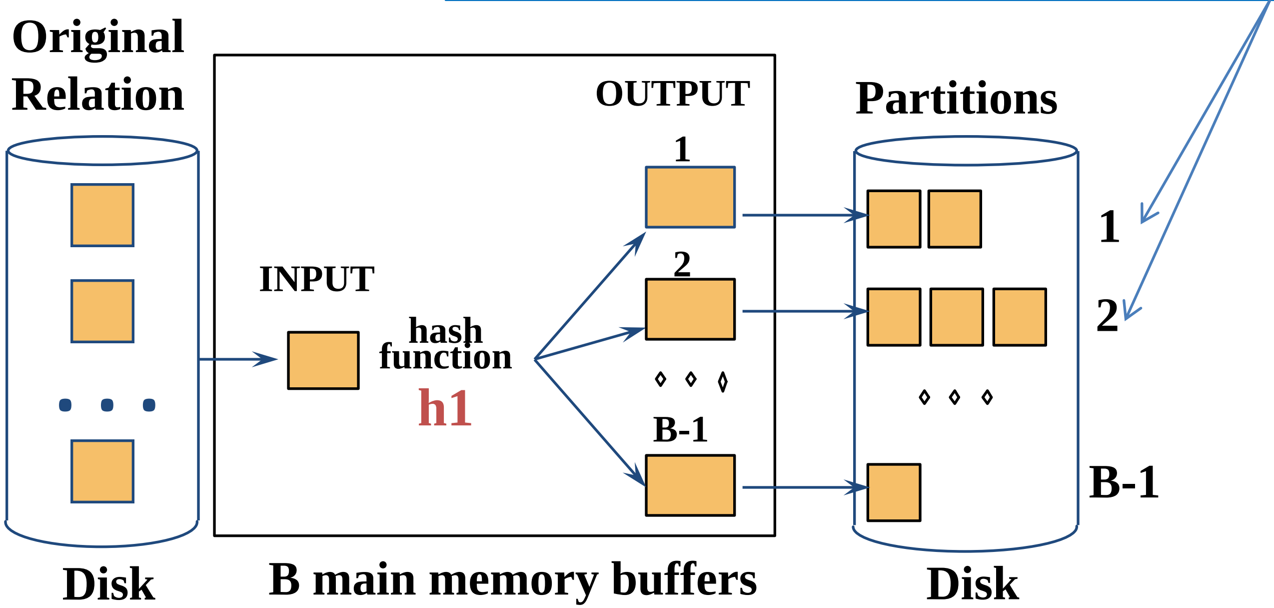
- Partitioning Phase (*assuming B buffers*):
  - Read ***R*** using 1 input buffer, *one* page at a time
  - For each tuple in the input page
    - Discard unwanted fields
    - Apply hash function ***h1*** to choose one of ***B***-1 output buffers

Projection Based on Hashing

- The algorithm based on hashing has two phases:
  - Partitioning Phase
  - Duplicate Elimination Phase

- Partitioning Phase:

Two tuples that belong to different partitions are guaranteed not to be duplicates



Projection Based on Hashing

- The algorithm based on hashing has two phases:
  - Partitioning Phase
  - Duplicate Elimination Phase

- Duplicate Elimination Phase:
  - Read each partition and build a corresponding *in-memory* hash table, using hash function ***h2*** (<> ***h1***) on all fields, while discarding duplicates
  - If a partition ***P*** does not fit in memory, apply hash-based projection algorithm *recursively* on ***P***

Projection Based on Hashing

- The algorithm based on hashing has two phases:
  - Partitioning Phase
  - Duplicate Elimination Phase

- What is the I/O cost of hash-based projection?
  - Partitioning phase = ***M*** (to read ***R***) + ***T*** (to write out the projected tuples) I/Os
  - Duplicate Elimination phase = ***T*** (to read in every partition) (CPU and final writing costs are ignored)
  - Total Cost = ***M*** + 2***T***

Projection Based on Hashing: An Example

- Consider Q again:

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

- How many I/Os would evaluating Q incur?
  - Partitioning phase: ***M*** + ***T*** = 1000 + 250 I/Os
  - Duplicate Elimination phase: **T** = 250 I/Os
  - Total = 1500 I/Os (as opposed to 2500 I/Os and 1500 I/Os using *projection based on sorting* and *projection based on modified external sorting*, respectively)

Which one is better, *projection based on modified external sorting* or *projection based on hashing*?

Sorting vs. Hashing

- The sorting-based approach is superior *if:*
  - The duplicate frequency is high
  - Or the distribution of (hash) values is very skewed
- With the sorting-based approach the result is sorted!
- Most DBMSs incorporate a *sorting utility*, which can be used to implement projection relatively easy
- Hence, sorting is the standard approach for projection!

Index-Only Scan

- Can an index be used for projections?
  - Useful if the key includes *all* wanted attributes
  - As such, key values can be simply retrieved from the index without ever accessing the actual relation!
  - This technique is referred to as index-only scan

- If an ordered (i.e., tree) index contains all wanted attributes as *prefix* of search key, we can:
  - Retrieve index entries in order (index-only scan)
  - Discard unwanted fields and compare adjacent tuples to eliminate duplicates

Outline

- Introduction
- The Selection Operation
- The Projection Operation
- The Join Operation ✓

The Join Operation

- Consider the following query, Q, which implies a join:

```
SELECT *
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
```

- How can we evaluate Q?
  - Compute R × S
  - Select (and project) as required

- But, the result of a cross-product is typically much larger than the result of a join

- Hence, it is very important to implement joins *without* materializing the underlying cross-product

The Join Operation

- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

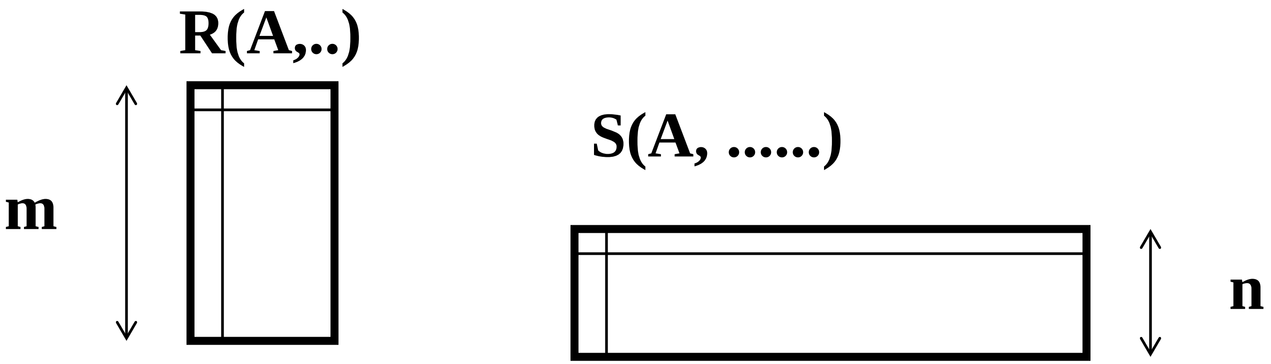
- Join algorithms which do not enumerate the cross-product:
  - Index Nested Loops Join
  - Sort-Merge Join
  - Hash Join

Assumptions

- We assume *equality* joins with:
  - ***R*** represents Reserves and ***S*** represents Sailors
  - ***M*** pages in ***R***, p_R tuples per page, ***m*** tuples total
  - ***N*** pages in ***S***, p_S tuples per page, ***n*** tuples total
- We will consider more complex join conditions later
- Our cost metric is the number of I/Os
- We ignore output and computational costs

Simple Nested Loops Join

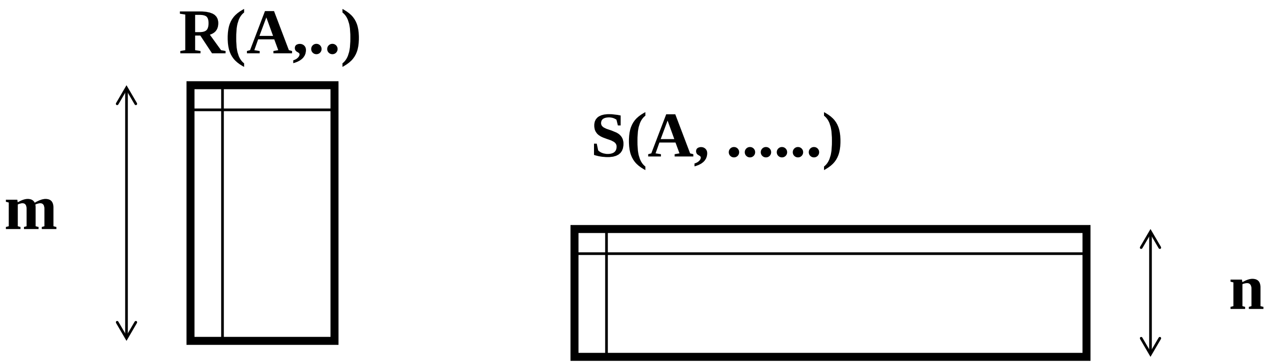
- Algorithm #0: (naive) nested loop (**SLOW**!)



Simple Nested Loops Join

- Algorithm #0: (naive) nested loop (**SLOW**!)

```
for each tuple r of R
    for each tuple s of S
        print, if they match
```



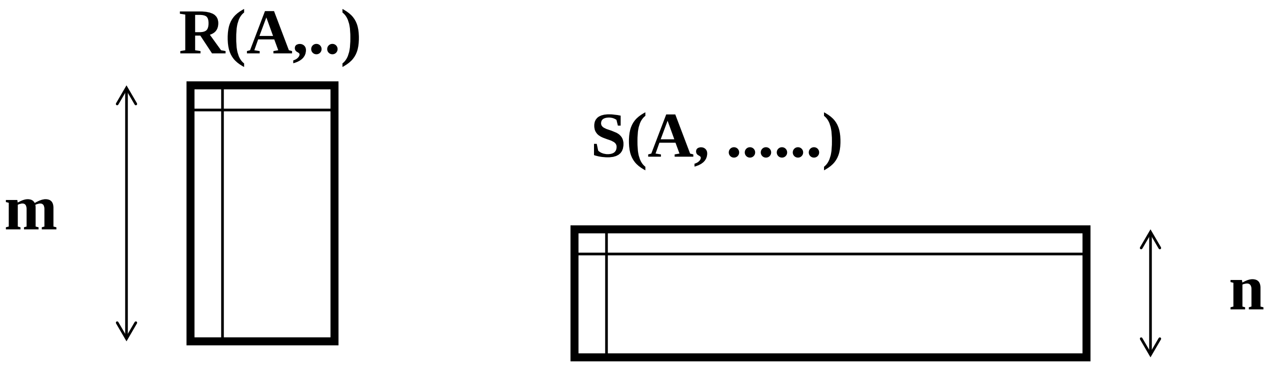
Simple Nested Loops Join

- Algorithm #0: (naive) nested loop (**SLOW**!)

```
for each tuple r of R
    for each tuple s of S
        print, if they match
```

R: Outer Relation

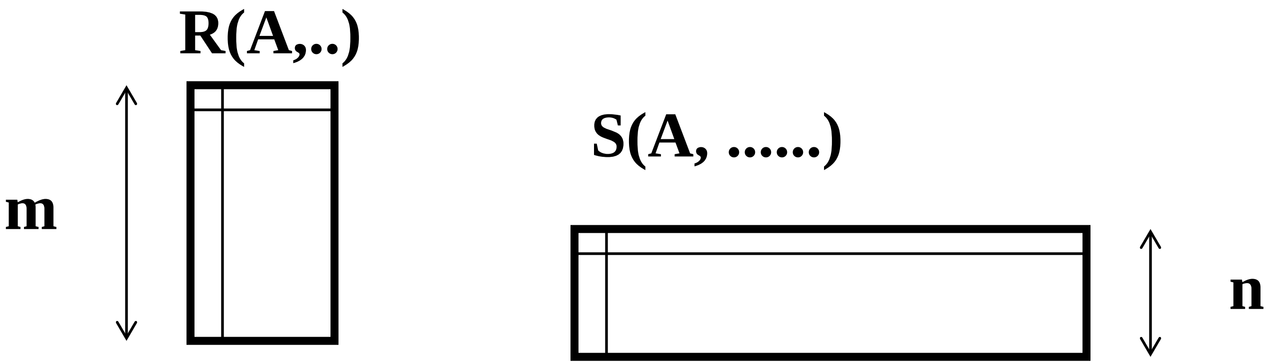
S: Inner Relation



Simple Nested Loops Join

- Algorithm #0: (naive) nested loop (**SLOW**!)

How many disk accesses ('M' and 'N' are the numbers of pages for 'R' and 'S')?

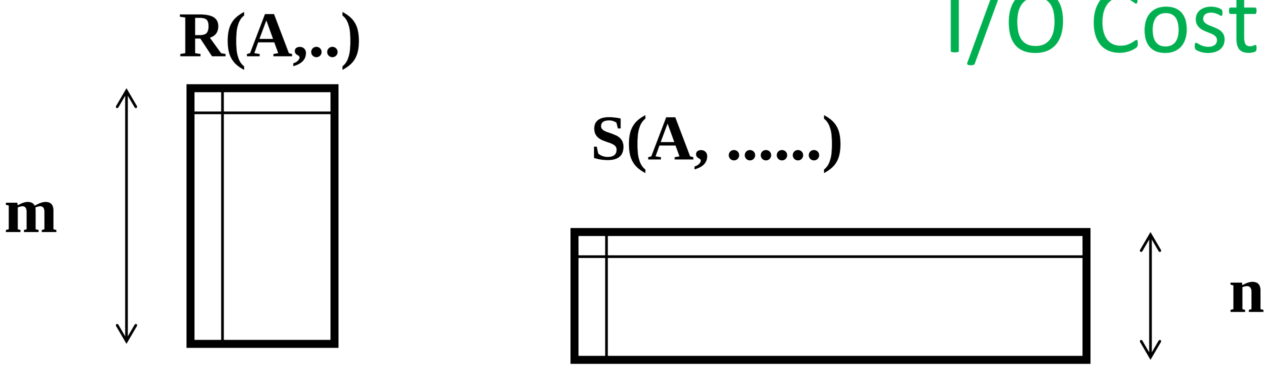


Simple Nested Loops Join

- Algorithm #0: (naive) nested loop (**SLOW**!)

How many disk accesses ('M' and 'N' are the numbers of pages for 'R' and 'S')?

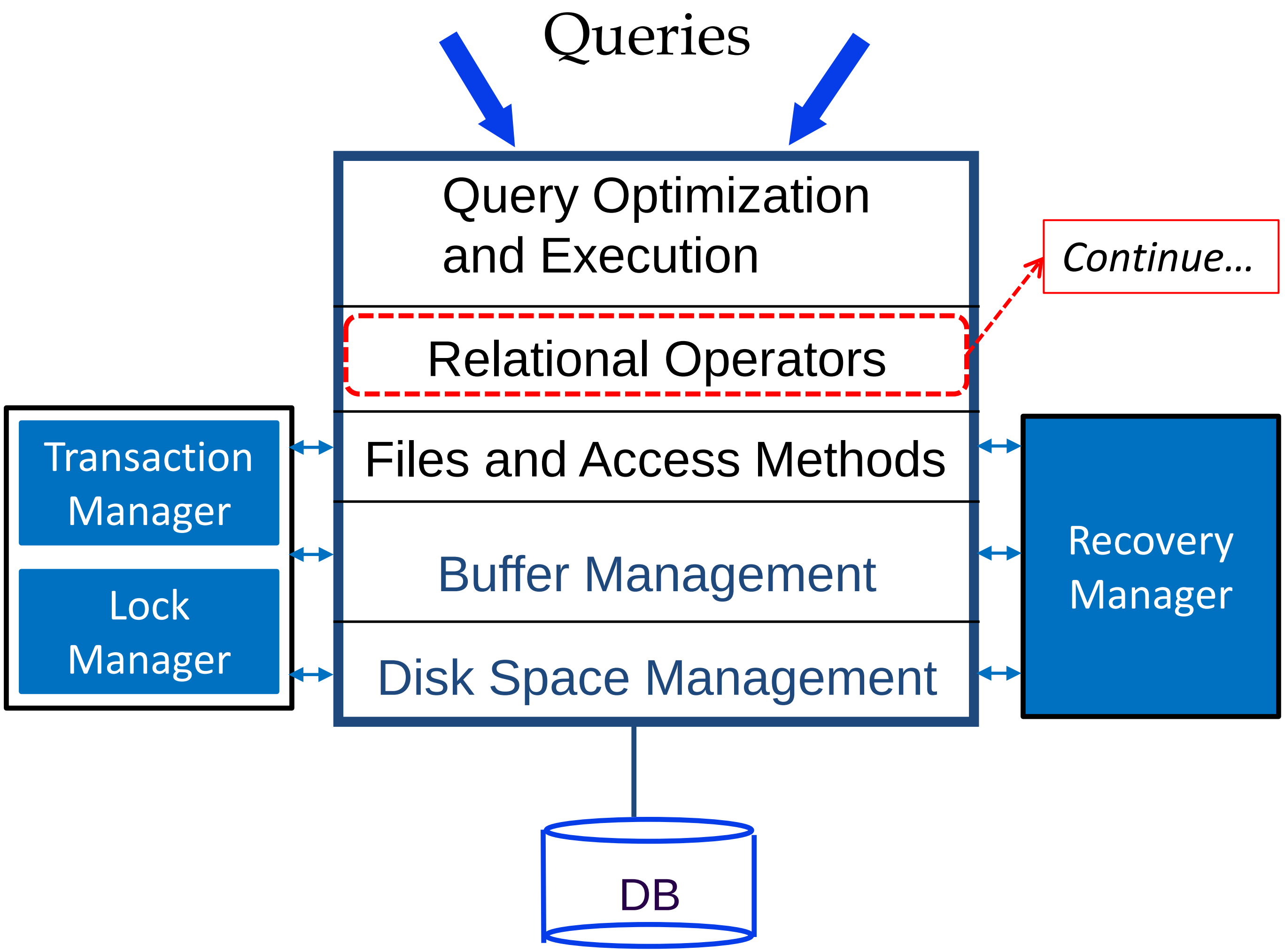
I/O Cost = M+m*N



Simple Nested Loops Join

- Let us check with actual numbers:
  - Cost = $(p_R * M) * N + M$ = 100*1000*500+1000 I/Os
  - At 10ms/IO, total = ~6days (!)
- What if we do the join *one-page-at-a-time*?
  - Cost = $M * N + M$ = 1000*500+1000 I/Os
  - At 10ms/IO, total = 1.4 hours (!)
- What if smaller relation (S) was outer?
  - (1000*500+1000) vs. (1000*500+500)
  - Slightly better

Next Class


Queries
Query Optimization and Execution
Relational Operators
Continue…
Files and Access Methods
Buffer Management
Disk Space Management
Transaction Manager
Lock Manager
Recovery Manager
DB