Database Applications (15-415)

DBMS Internals- Part VII
Lecture 15, March 17, 2014

Mohammad Hammoud

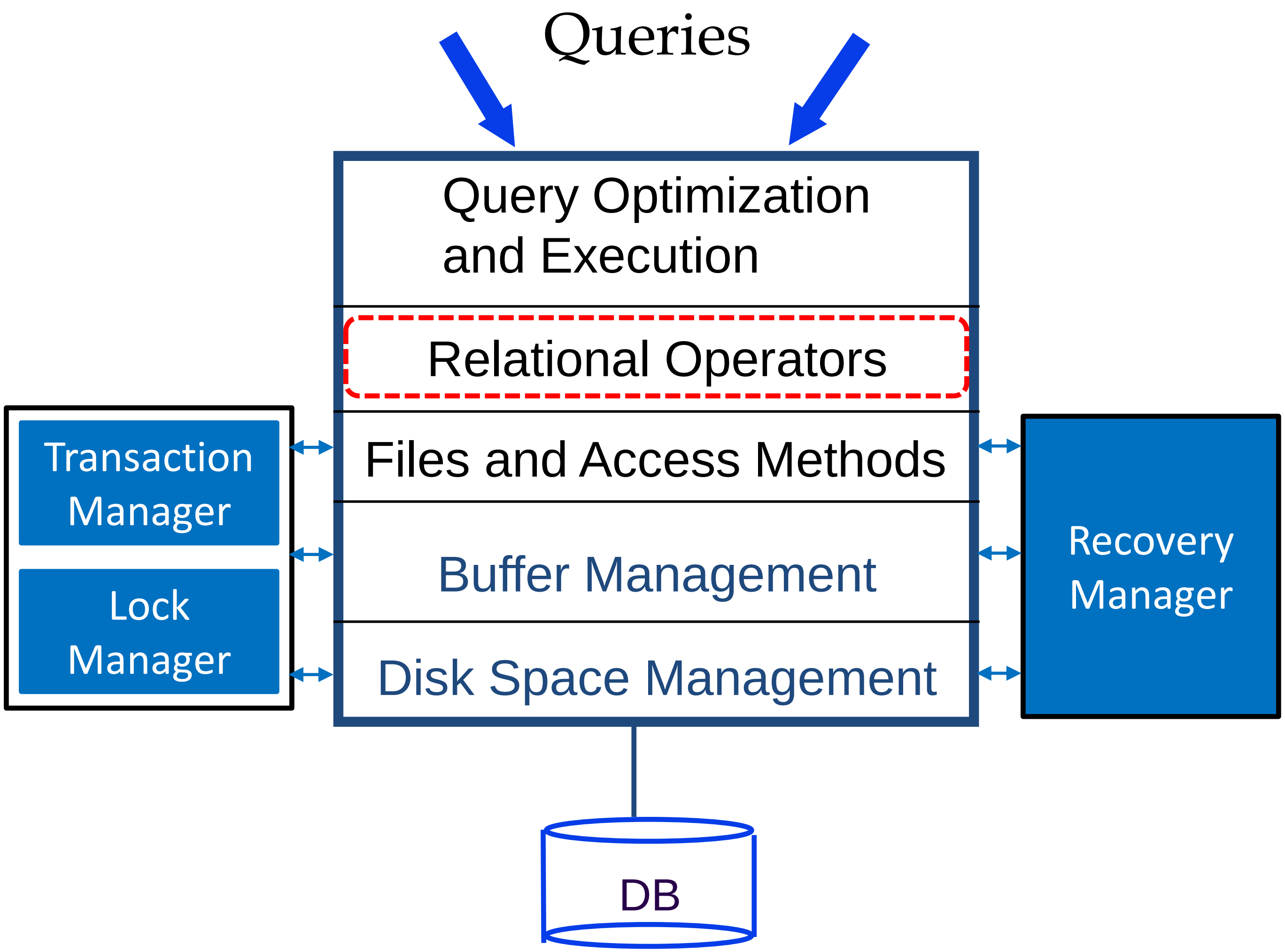
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Today…

- Last Session:
  - DBMS Internals- Part VI
    - Algorithms for Relational Operations

- Today's Session:
  - DBMS Internals- Part VII
    - Algorithms for Relational Operations (*Cont'd*)

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

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 ✓

Last Class

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

The Join Operation

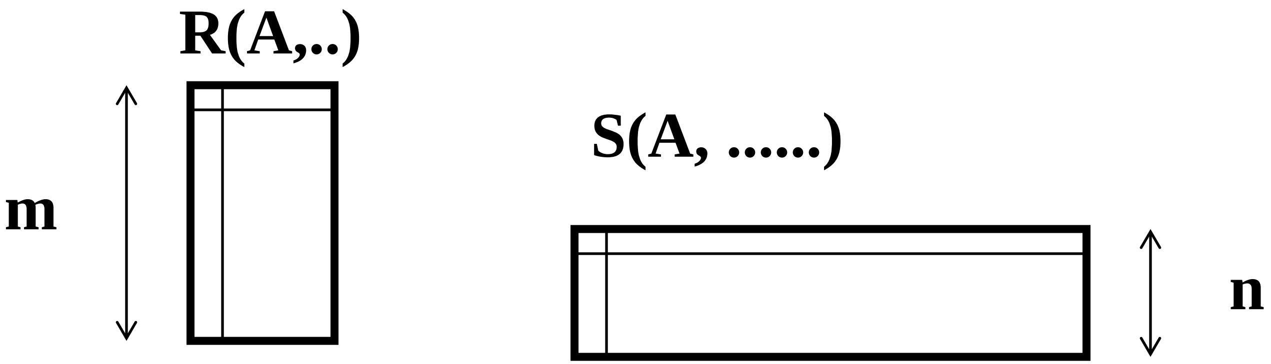
- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

- 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

Simple Nested Loops Join

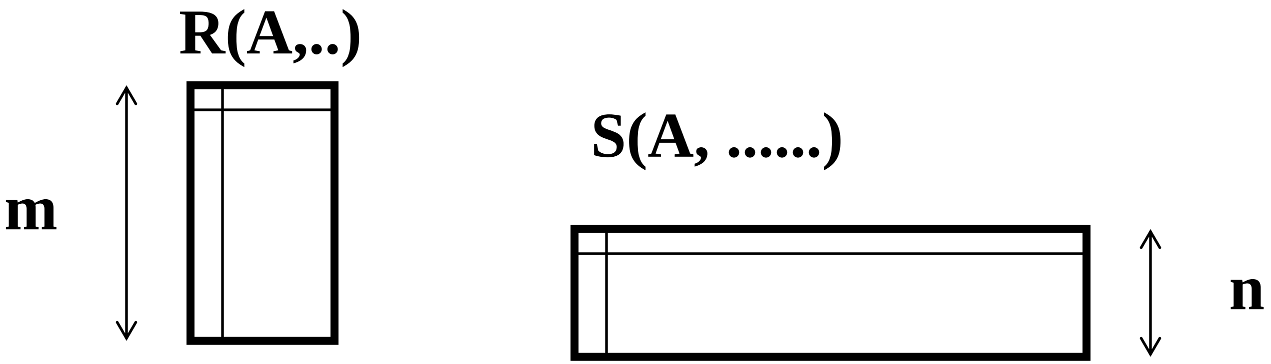
- Algorithm #0: (naive) nested loop (**SLOW**!)



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



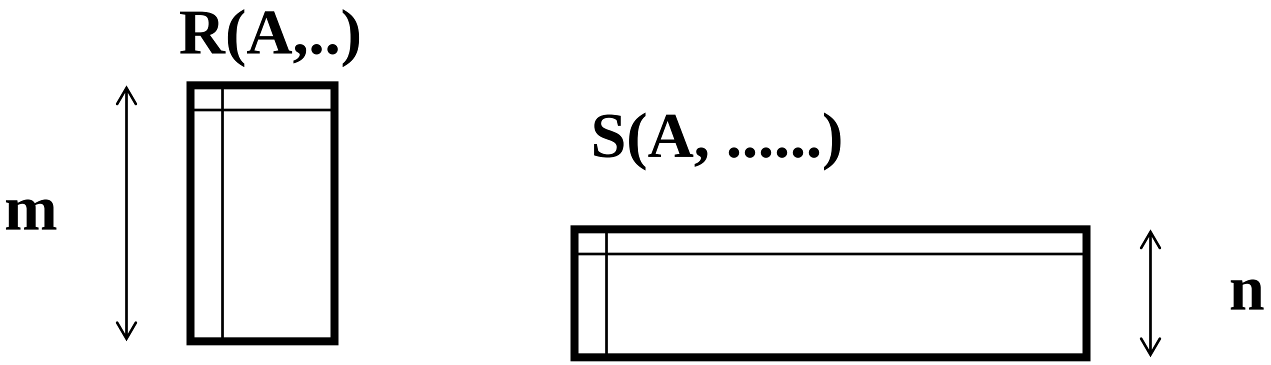
Simple Nested Loops Join

- 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

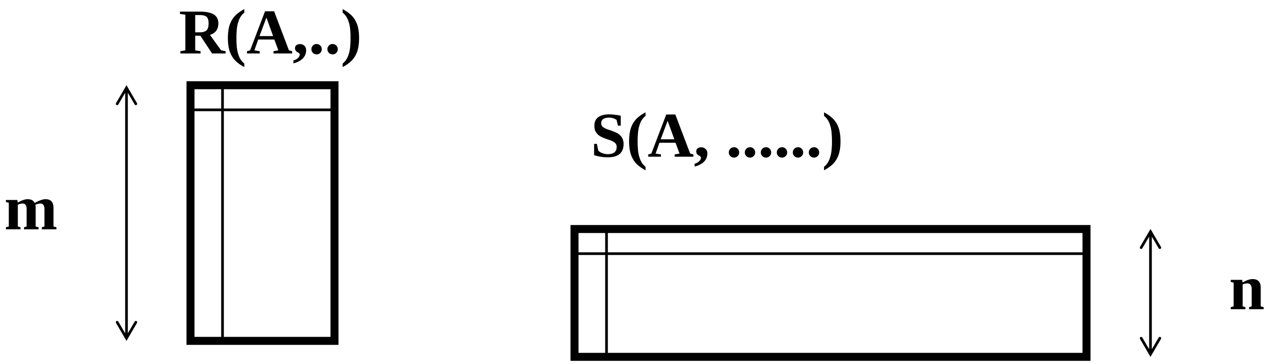
S ← Inner Relation



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’)?

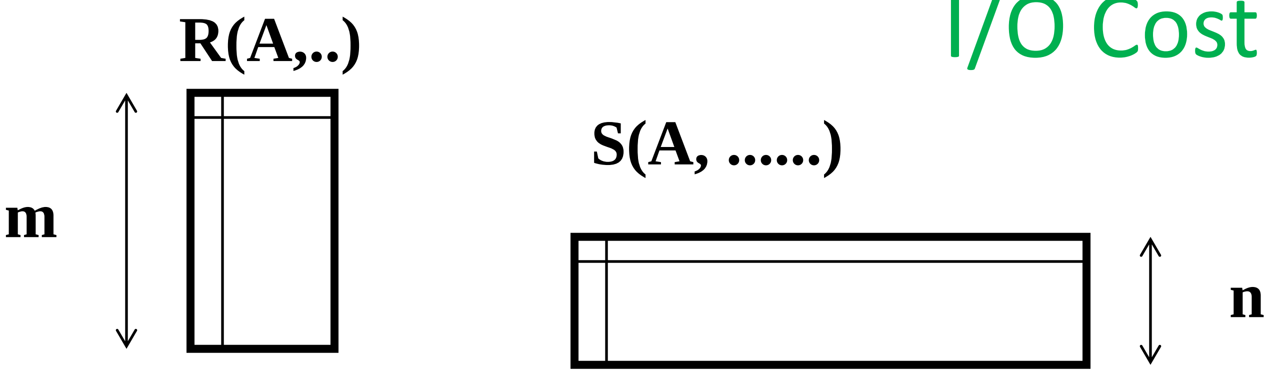


Simple Nested Loops Join

- 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

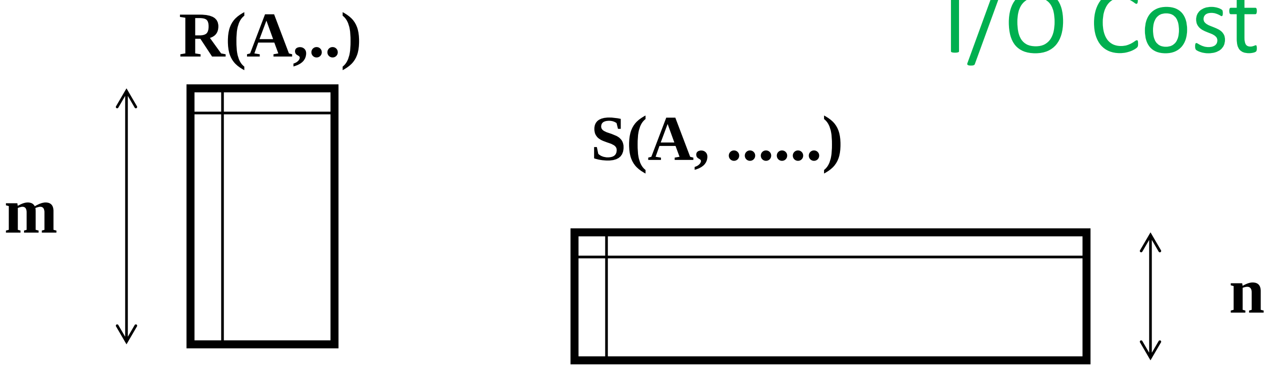


Simple Nested Loops Join

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

- Cost = $(p_R * M) * N + M$ = 100*1000*500+1000 I/Os
- At 10ms/IO, total = ~6days (!)

I/O Cost = M+m*N

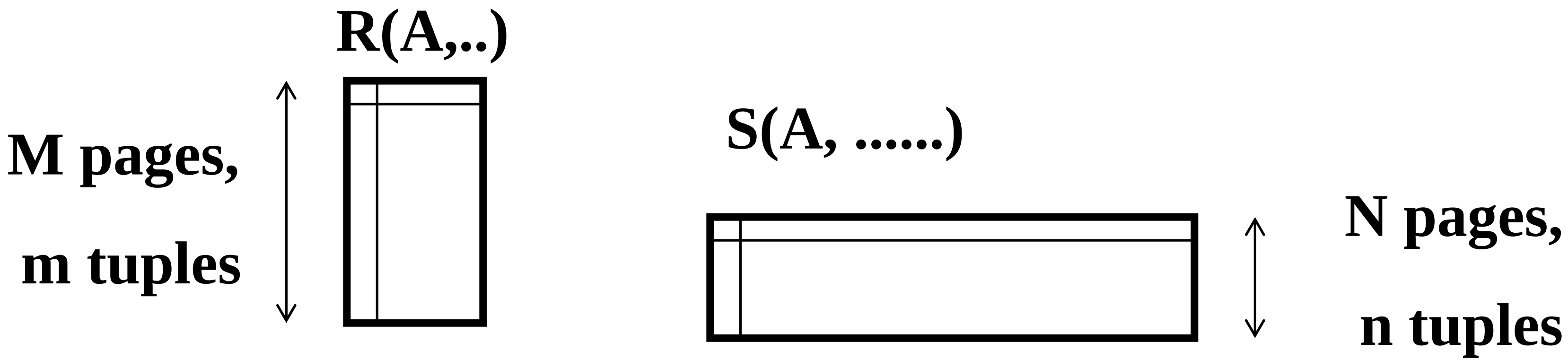


Can we do better?

Nested Loops Join: A Simple Refinement

- Algorithm:
  - Read in a *page* of R
    - Read in a *page* of S
      - Print matching tuples

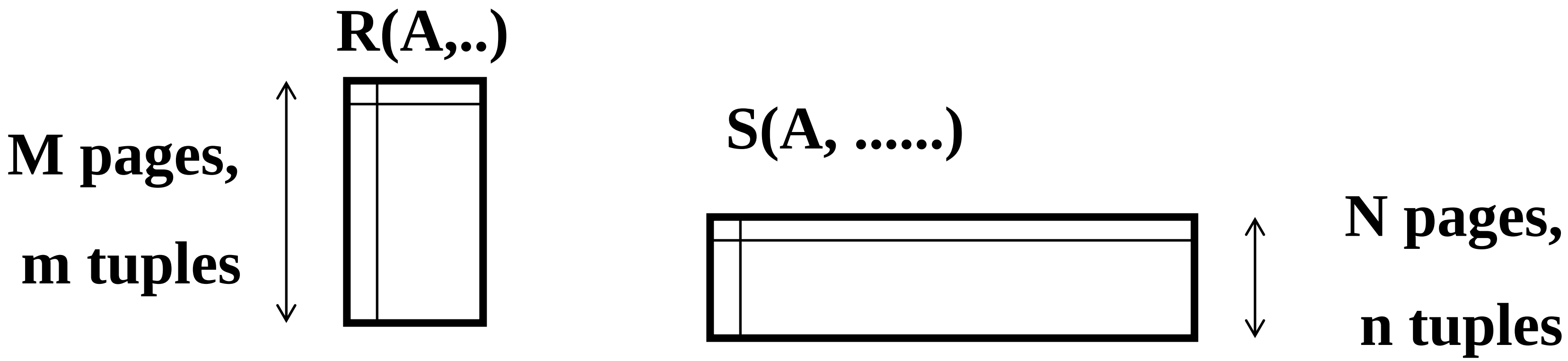
COST= ?



Nested Loops Join: A Simple Refinement

- Algorithm:
  - Read in a *page* of R
    - Read in a *page* of S
      - Print matching tuples

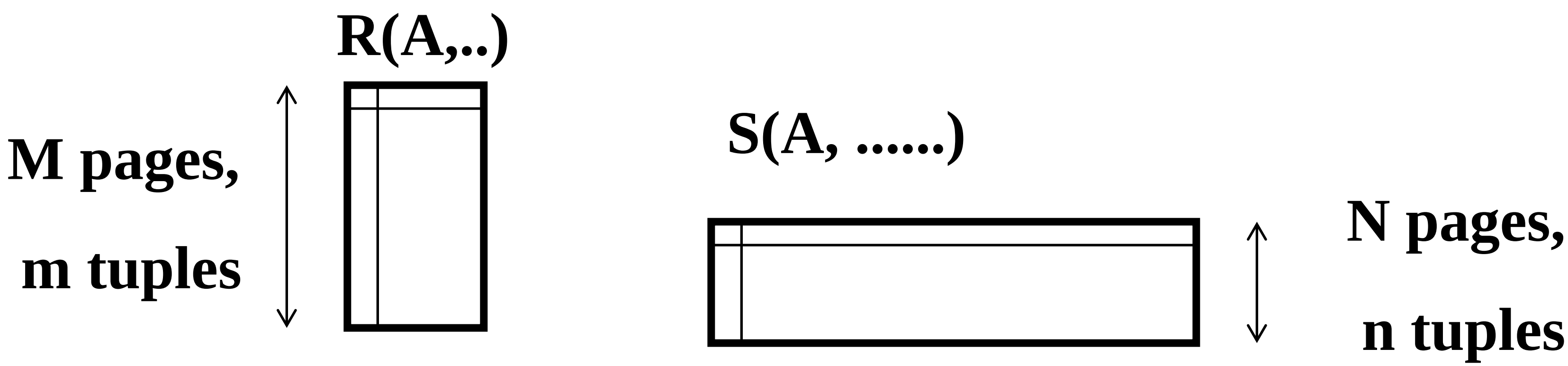
COST= M+M*N



Nested Loops Join

- Which relation should be the *outer*?

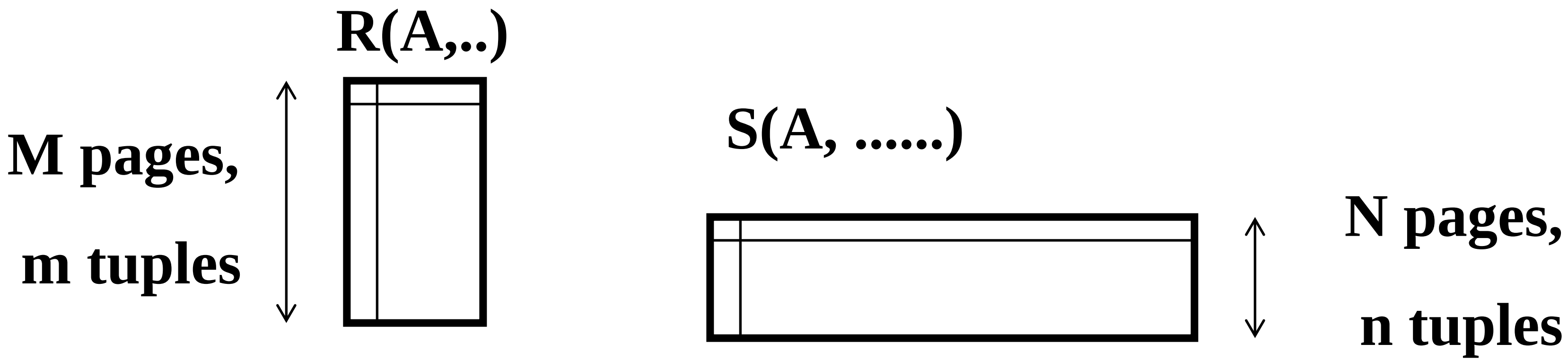
COST= M+M*N



Nested Loops Join

- Which relation should be the *outer*?
- A: The smaller (page-wise)

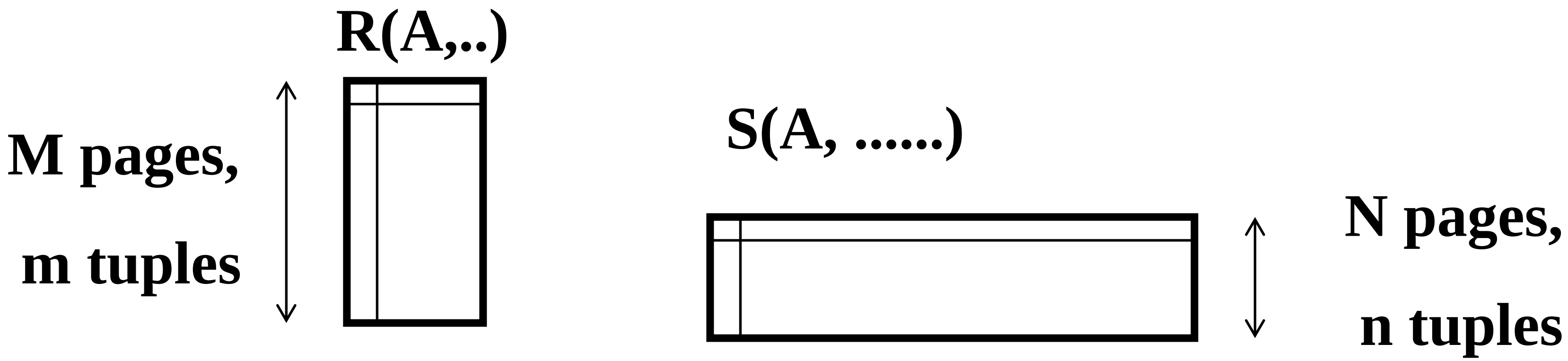
COST= M+M*N



Nested Loops Join

- M=1000, N=500 - *if larger is the outer*:
- Cost = 1000 + 1000*500 = 501,000
- = 5010 sec ~ 1.4h

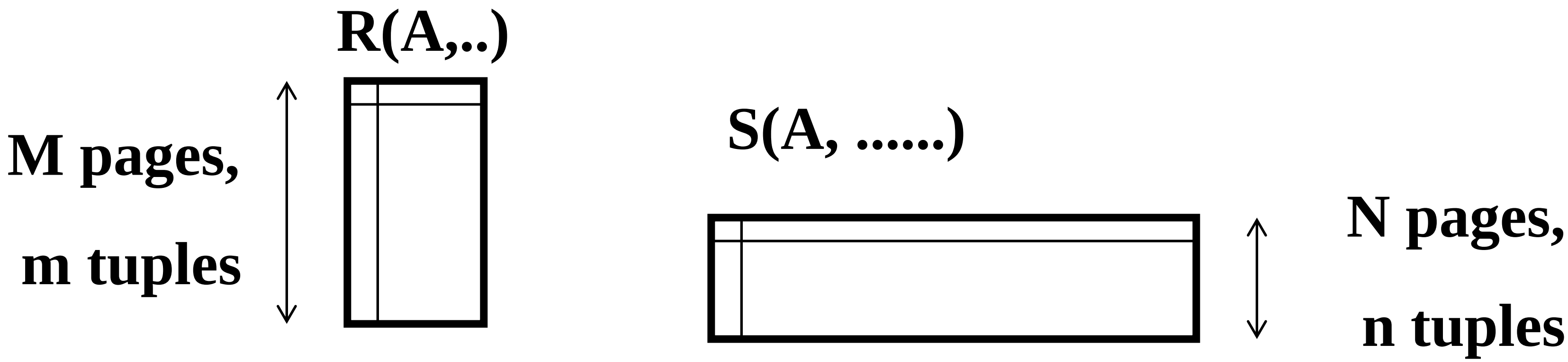
COST= M+M*N



Nested Loops Join

- M=1000, N=500 - *if smaller is the outer*:
- Cost = 500 + 1000*500 = 500,500
- = 5005 sec ~ 1.4h

COST= N+M*N



Simple Nested Loops Join

- What if we do not apply the page-oriented refinement?
  - Cost = (p_R * M) * N + M = 100*1000*500+1000 I/Os
  - At 10ms/IO, total = ~6days (!)
- What if we apply the page-oriented refinement?
  - Cost = M * N + M = 1000*500+1000 I/Os
  - At 10ms/IO, total = 1.4 hours (!)
- What if the *smaller* relation is the outer?
  - Slightly better

The Join Operation

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

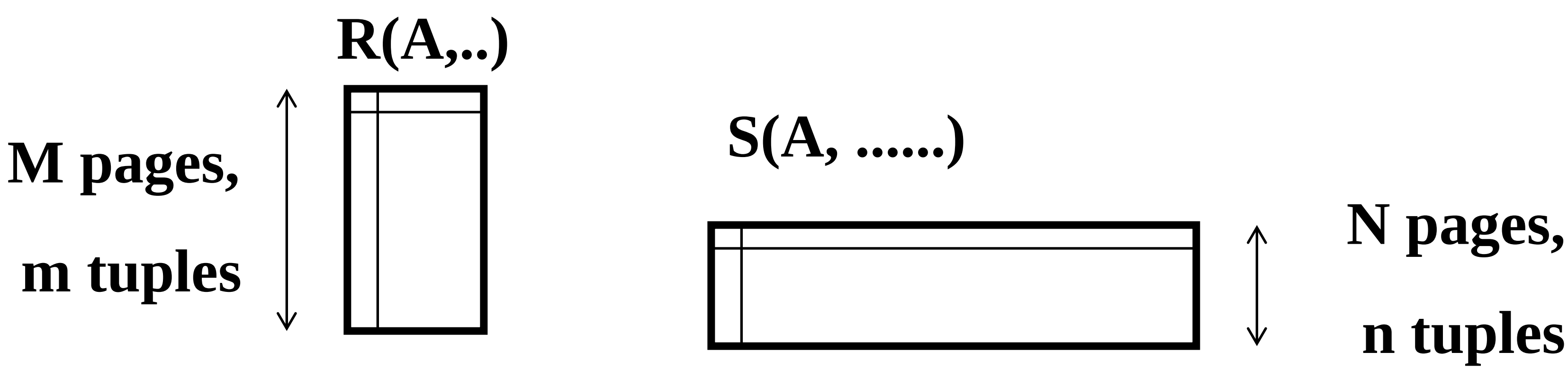
- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

  ✓

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

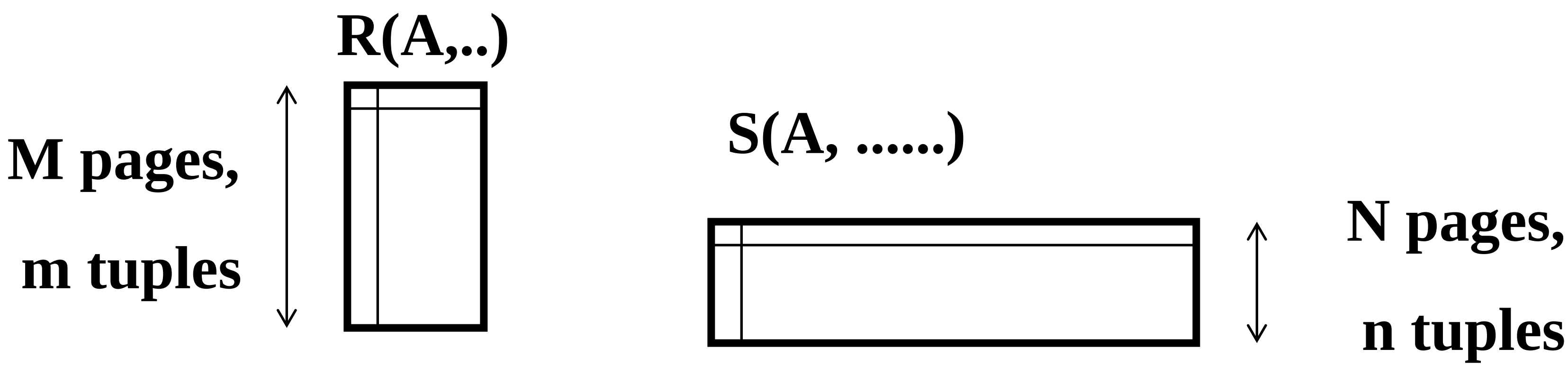
Block Nested Loops

- What if we have *B* buffer pages available?



Block Nested Loops

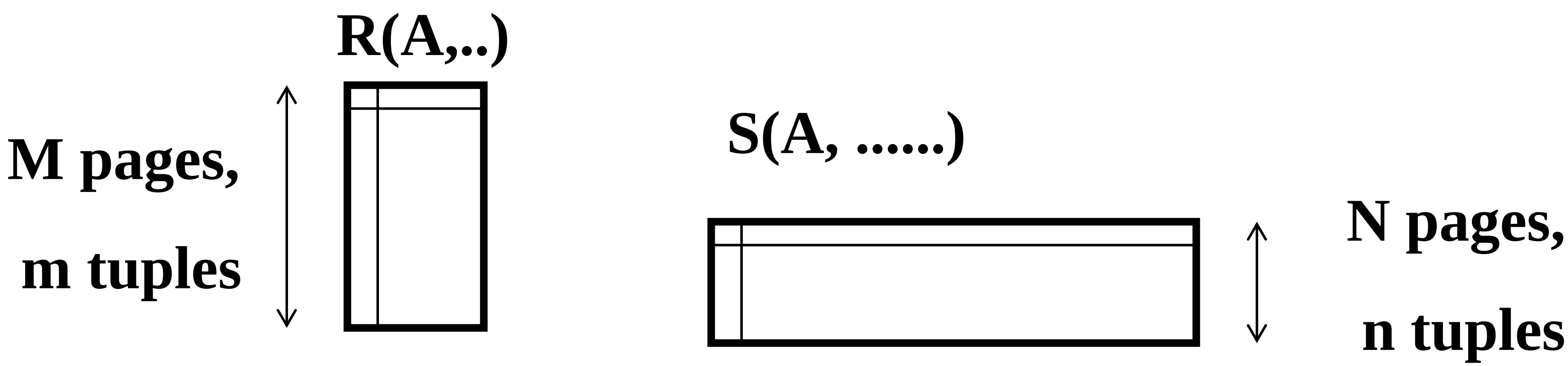
- What if we have *B* buffer pages available?
- A: give *B-2* buffer pages to outer, 1 to inner, 1 for output



Block Nested Loops

- Algorithm:
  - Read in *B-2* pages of R
    - Read in a page of S
      - Print matching tuples

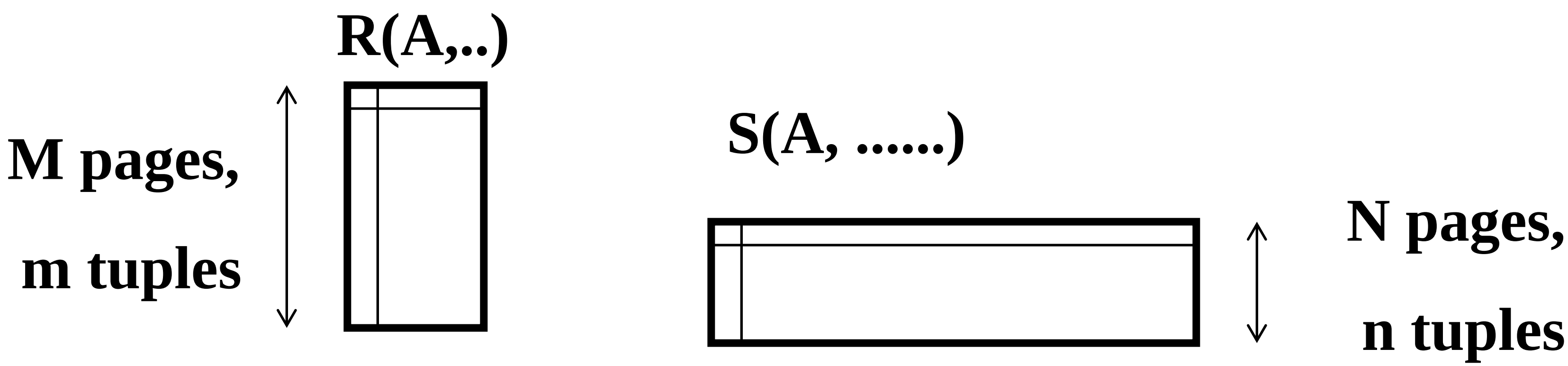
COST= ?



Block Nested Loops

- Algorithm:
  - Read in *B-2* pages of R
    - Read in a page of S
      - Print matching tuples

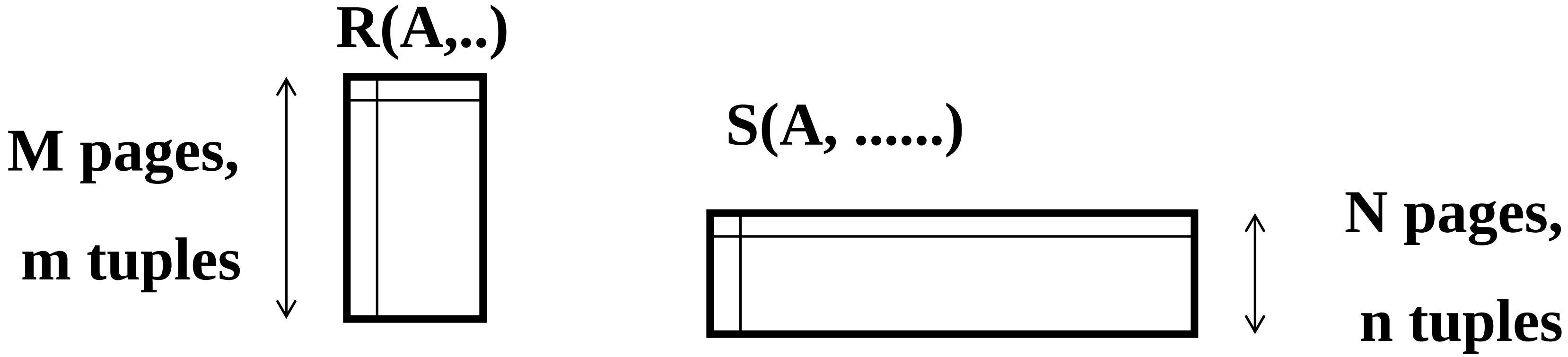
COST= $M+M/(B-2)*N$



Block Nested Loops

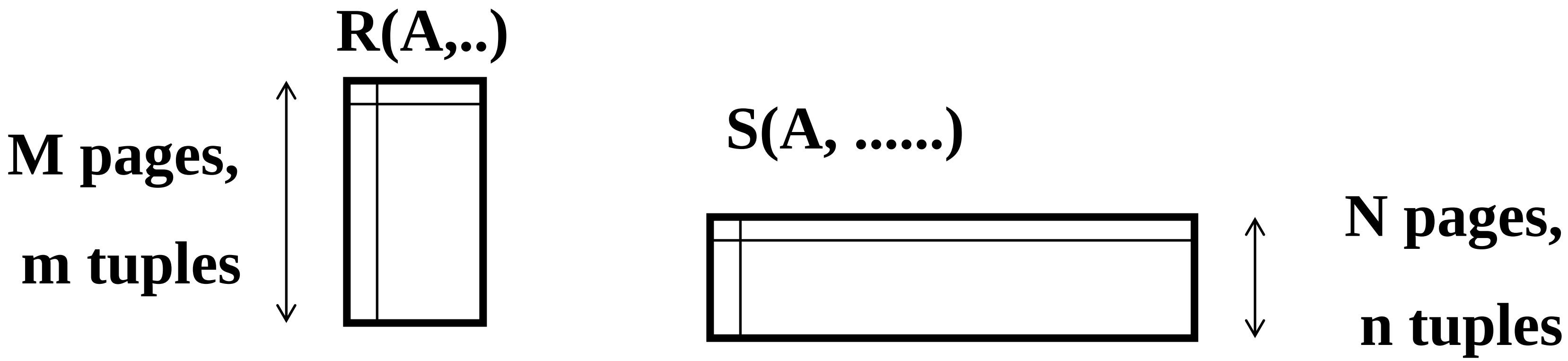
- And, actually:
- Cost = M + ceiling(M/(B-2)) * N

COST= M+M/(B-2)*N



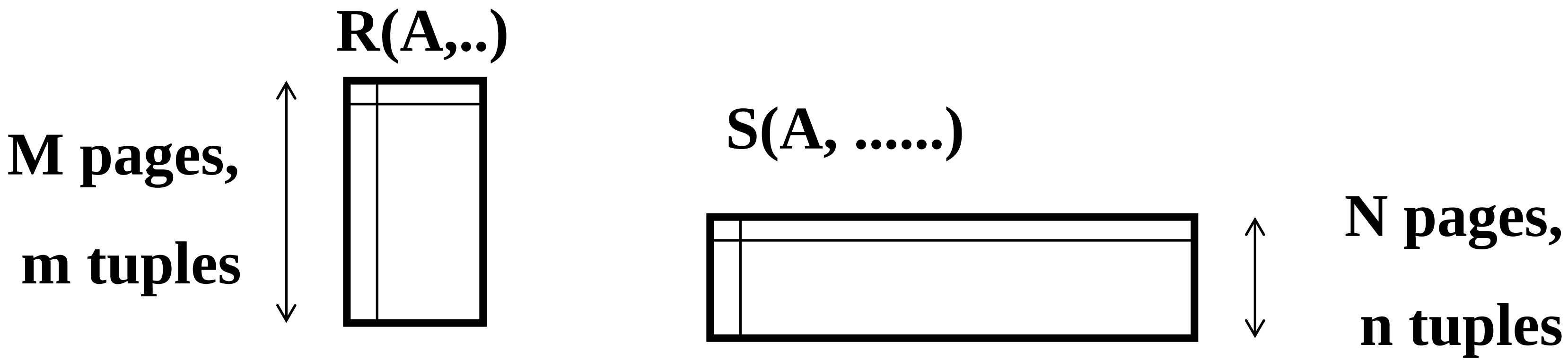
Block Nested Loops

- If the smallest (outer) relation fits in memory?
- That is, B= N+2
- Cost =?



Block Nested Loops

- If the smallest (outer) relation fits in memory?
- That is, B= N+2
- Cost =N+M (minimum!)



Nested Loops - Guidelines

- Pick as outer the smallest table
  (= fewest pages)

- Fit as much of it in memory as possible

- Loop over the inner

The Join Operation

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

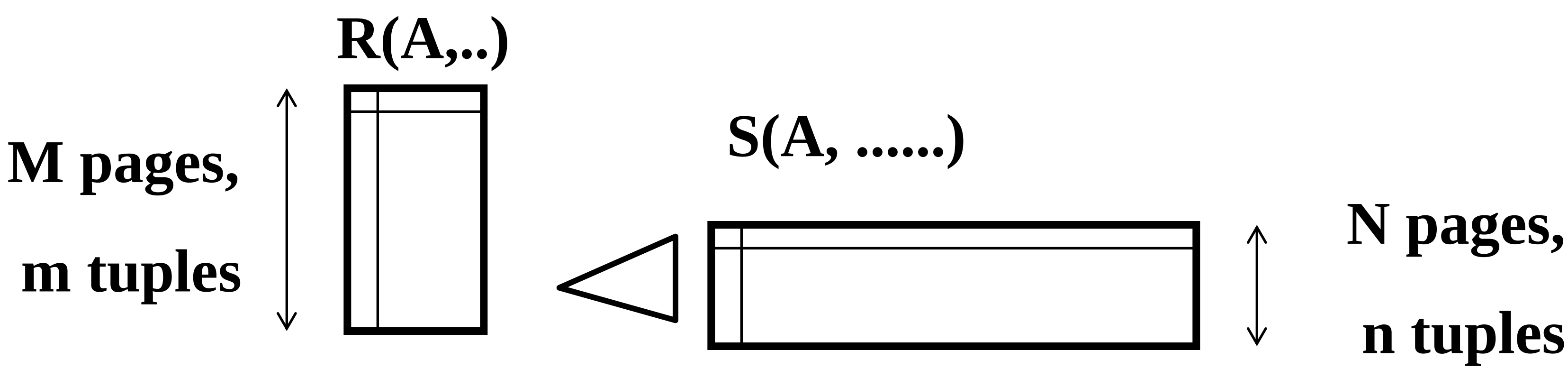
- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

  ✓

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

Index Nested Loops Join

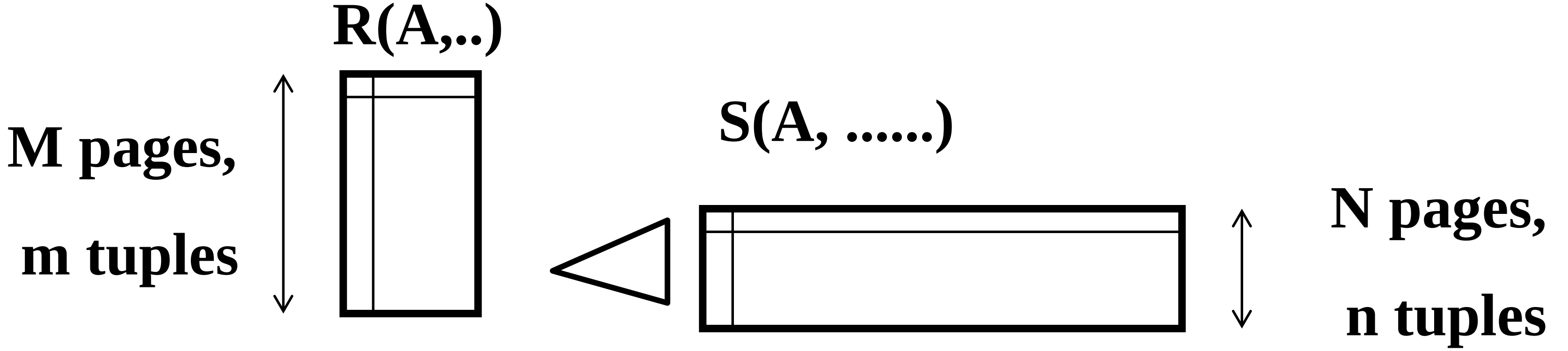
- What if there is an index on one of the relations on the join attribute(s)?
- A: Leverage the index by making the indexed relation *inner*



Index Nested Loops Join

- Assuming an index on S:

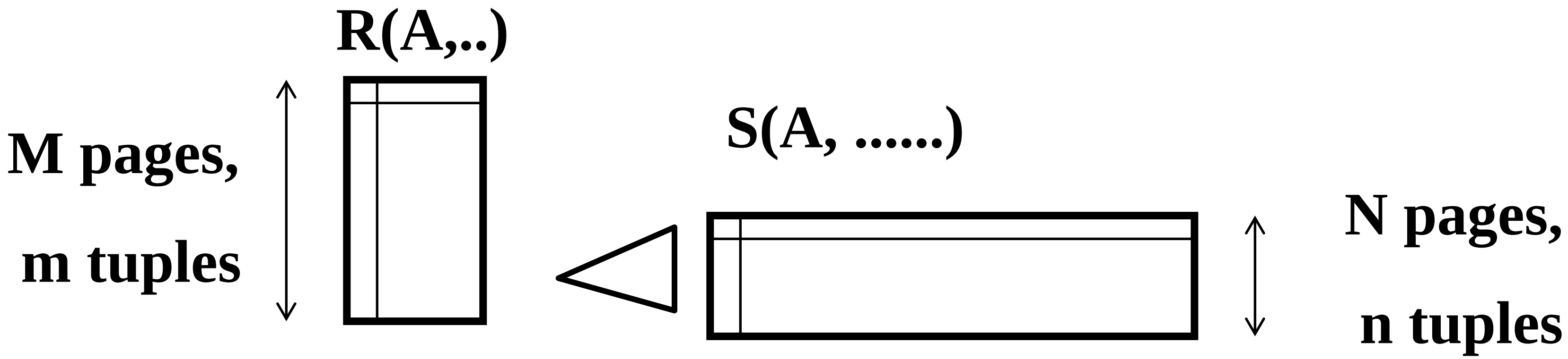
```
for each tuple r of R
    for each tuple s of S where r_i == s_j
        Add (r, s) to result
```



Index Nested Loops Join

- What will be the cost?
- Cost: M + m * c   (c: look-up cost)

'c' depends on the type of index, the adopted alternative and whether the index is clustered or un-clustered!



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

  ✓

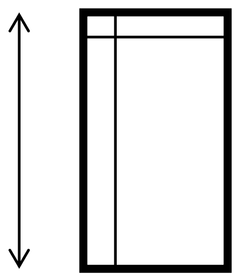
Sort-Merge Join

- Sort both relations on join attribute(s)
- Scan each relation and merge
- This works only for equality join conditions!

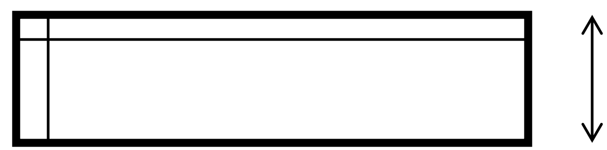
R(A,..)

M pages,

m tuples



S(A, ...)



N pages,

n tuples

Sort-Merge Join: An Example

= ?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

= NO

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

= ?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

= YES

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Output the two tuples

Sort-Merge Join: An Example

= ?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

= **YES**

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

= **YES**

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Output the two tuples

Sort-Merge Join: An Example

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

= ?

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

= NO

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

= ?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

= YES

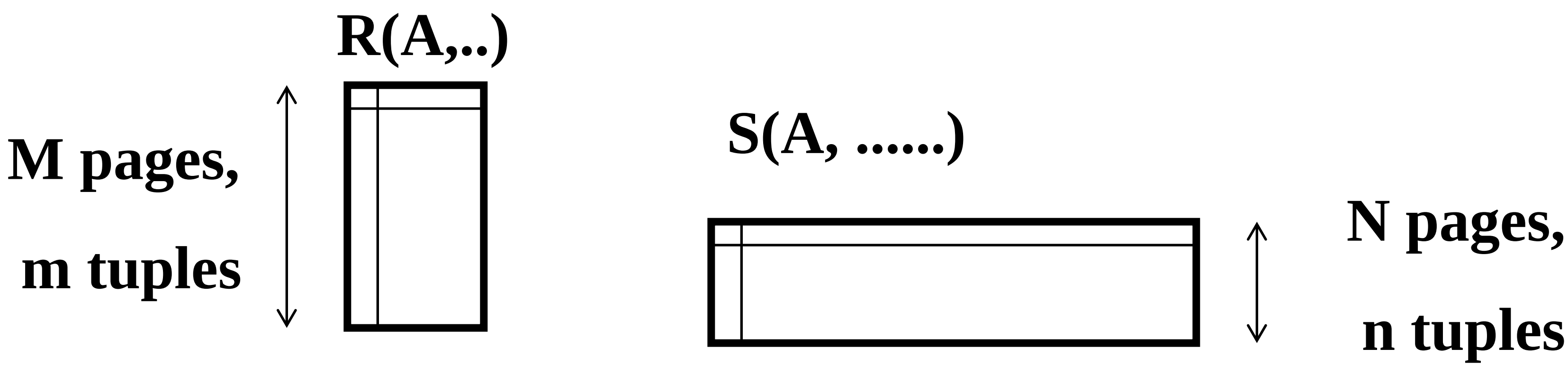
sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Output the two tuples

Continue the same way!

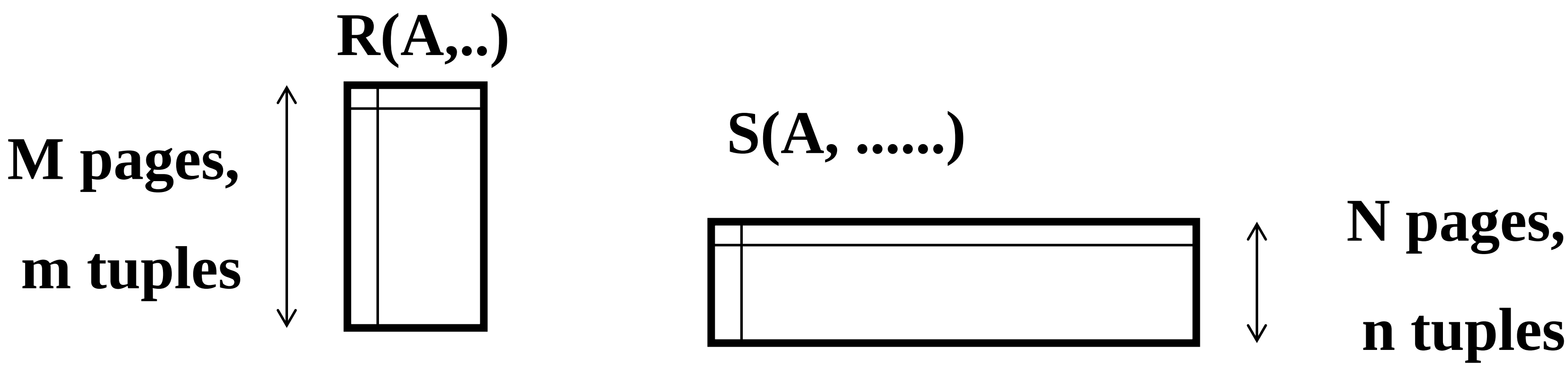
Sort-Merge Join

- What is the cost?
- $\sim 2*M*\log M/\log B + 2*N*\log N/\log B + M + N$



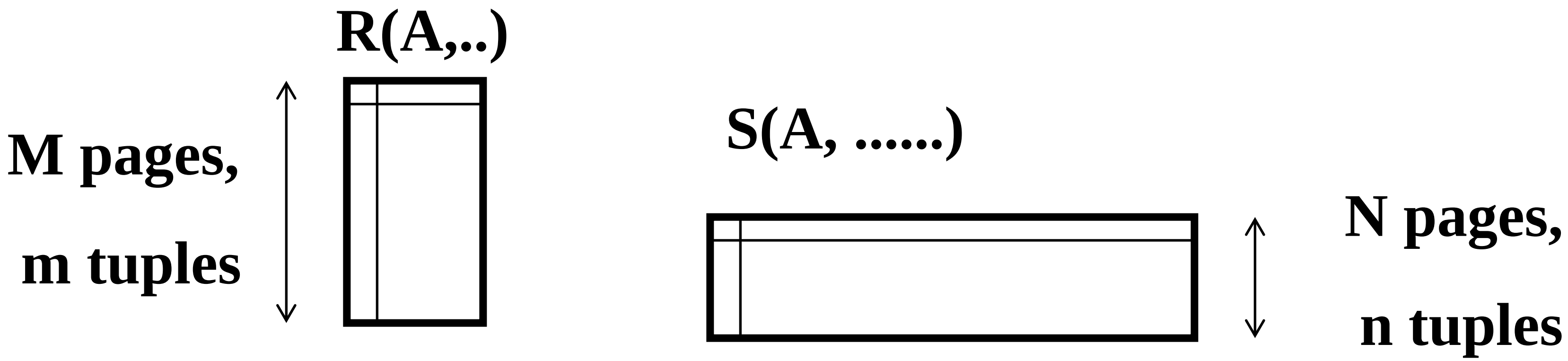
Sort-Merge Join

- Assuming 100 buffer pages, Reserves and Sailors can be sorted in 2 passes
- Total cost = 7500 I/Os
- Cost of Block Nested Loops Join = 7500 I/Os



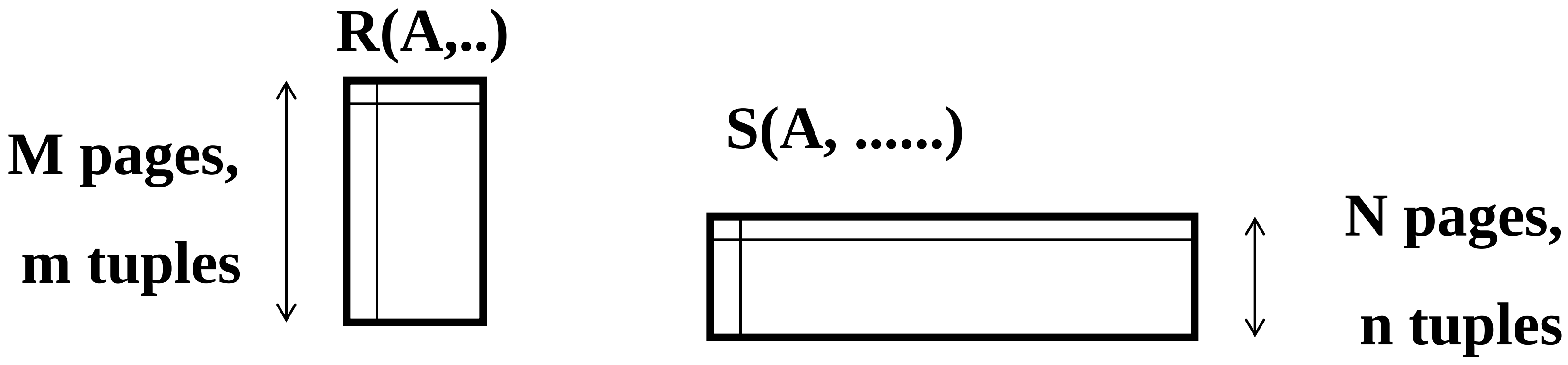
Sort-Merge Join

- Assuming 35 buffer pages, Reserves and Sailors can be sorted in 2 passes
- Total cost = 7500 I/Os
- Cost of Block Nested Loops Join = 15000 I/Os



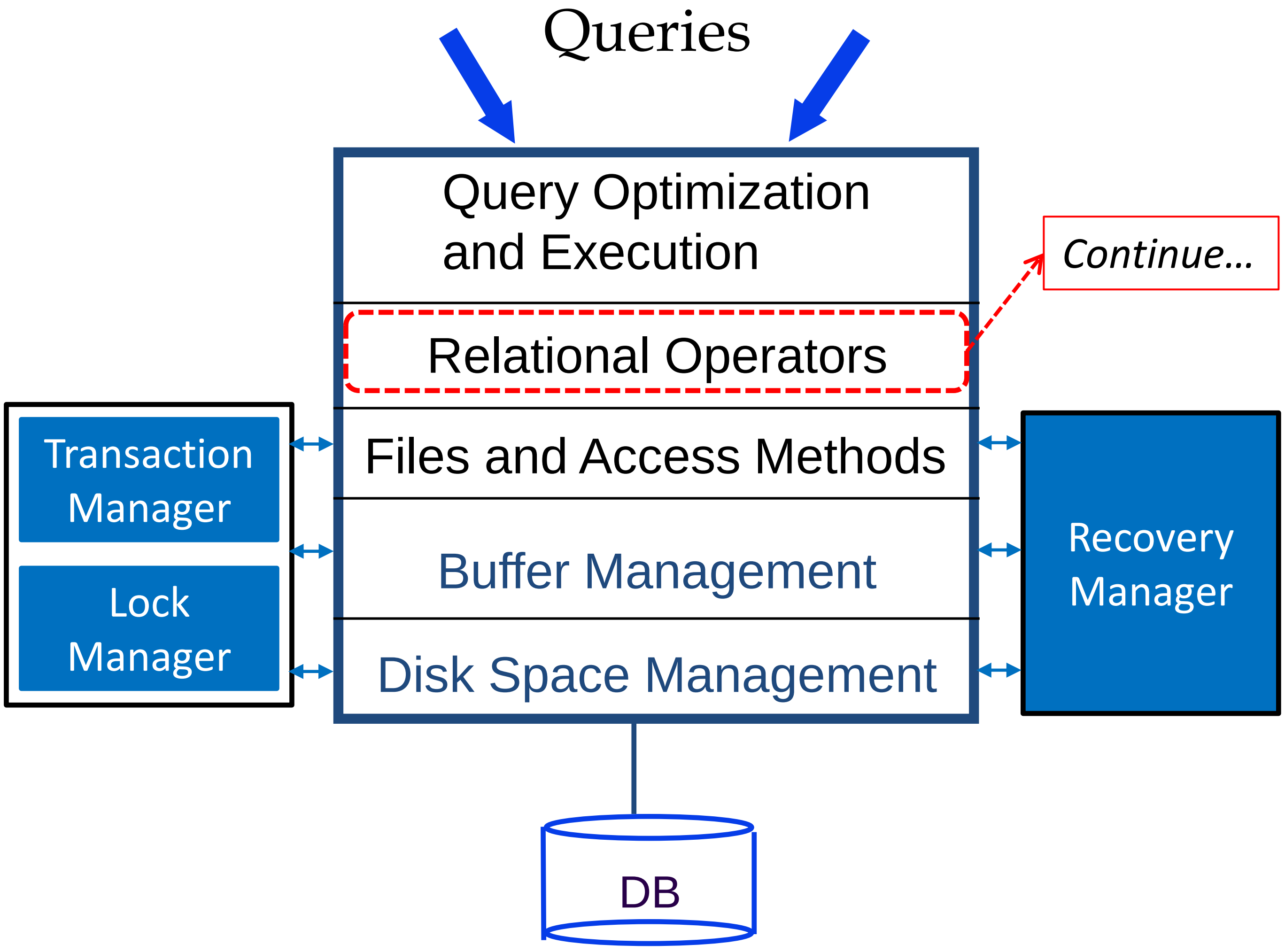
Sort-Merge Join

- Assuming 300 buffer pages, Reserves and Sailors can be sorted in 2 passes
- Total cost = 7500 I/Os
- Cost of Block Nested Loops Join = 2500 I/Os



It is possible to improve the Sort-Merge Join algorithm by combining the merging phase of sorting with the merging phase of the join!

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