Database Applications (15-415)

DBMS Internals- Part VII
Lecture 16, October 25, 2016

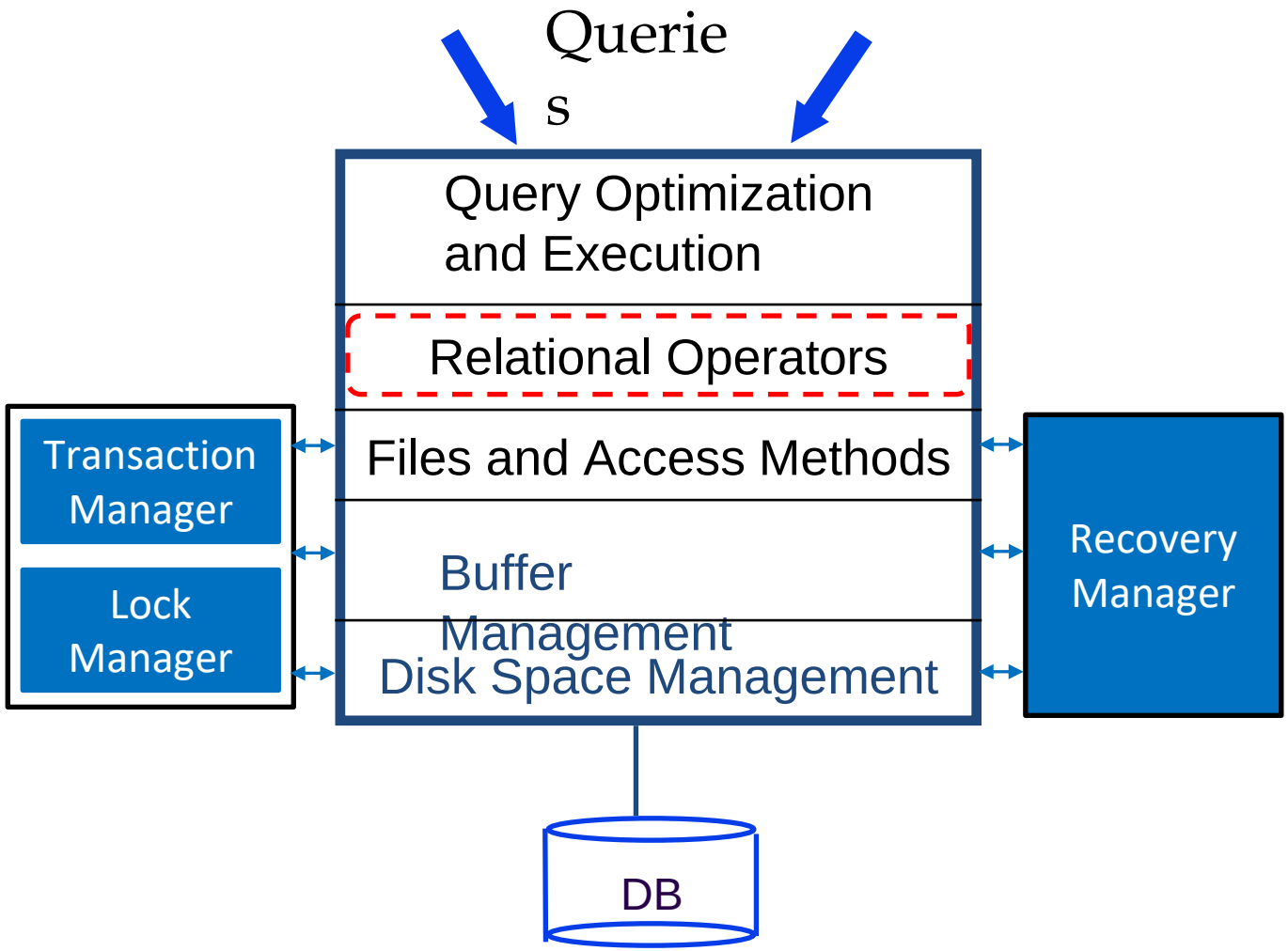
Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

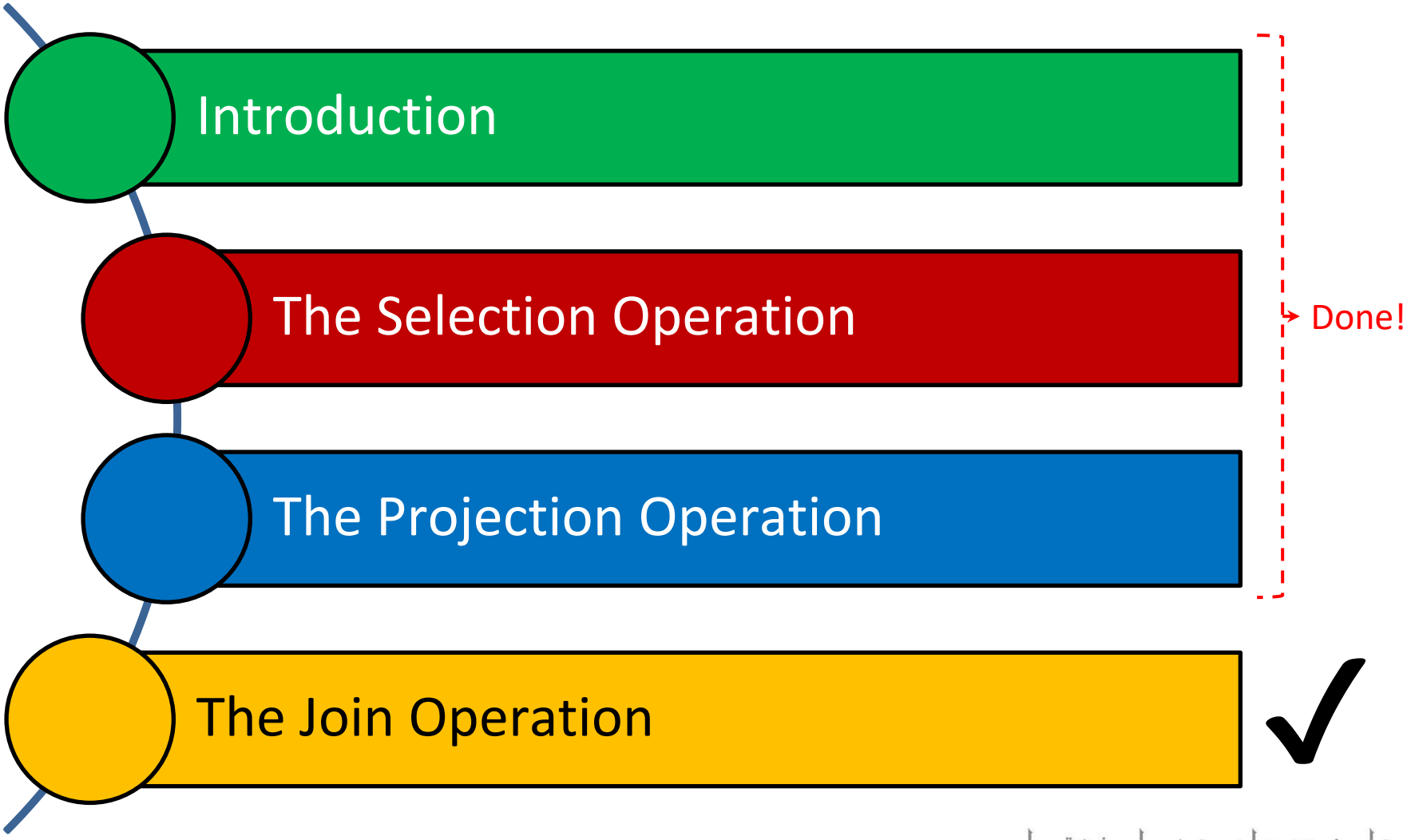
Today…

- Last Session:
  - DBMS Internals- Part VI
    - Algorithms for Relational Operations
- Today's Session:
  - DBMS Internals- Part VII
    - Algorithms for Relational Operations (*Cont'd*)
- Announcements:
  - P2 is due on Oct 27
  - PS3 is due on Nov 1

DBMS Layers



Outline



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*** representing Reserves and ***S*** representing 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 ignore the output and computational costs

The Join Operation

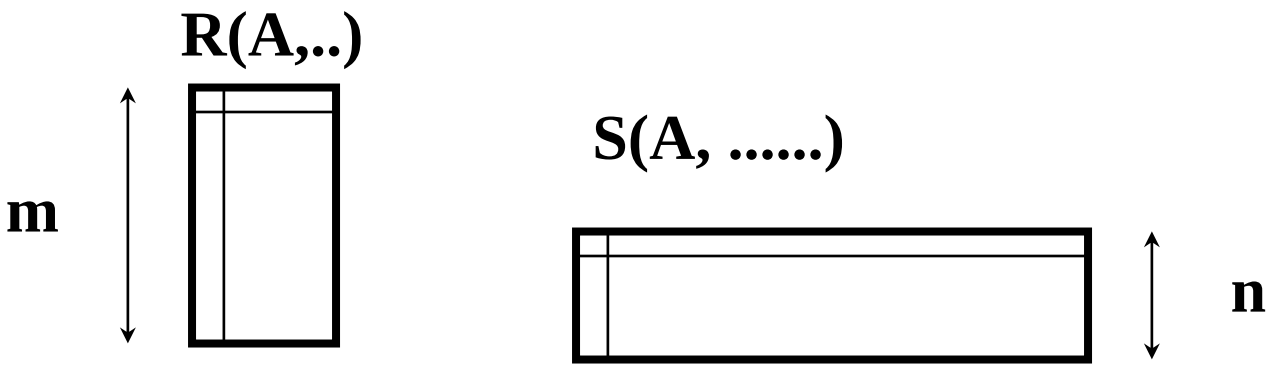
- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

- 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

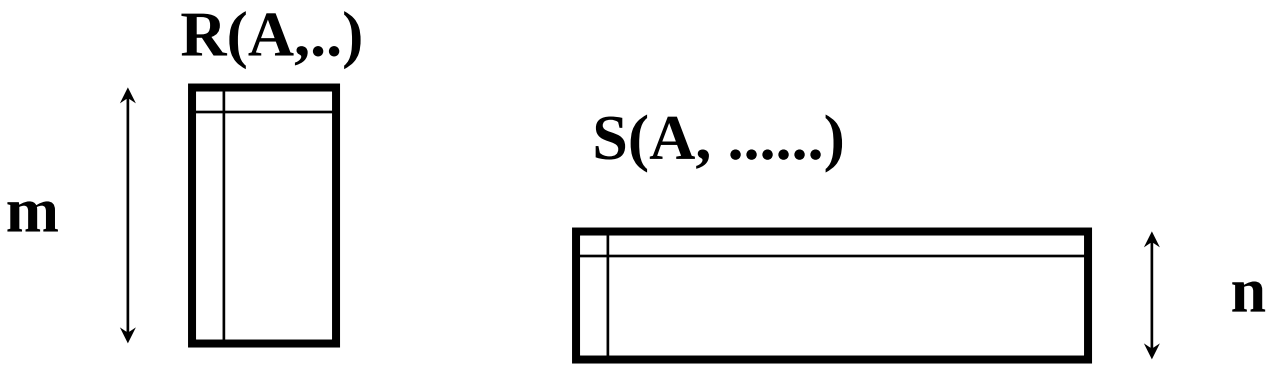
- Algorithm #0: (*naive*) nested loop (**SLOW**!)



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
```



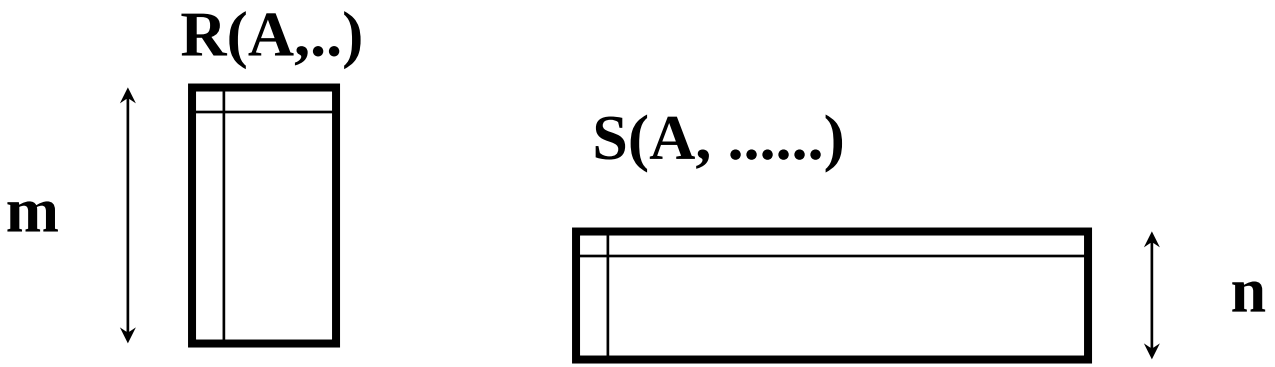
Simple Nested Loops Join

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

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

Outer Relation

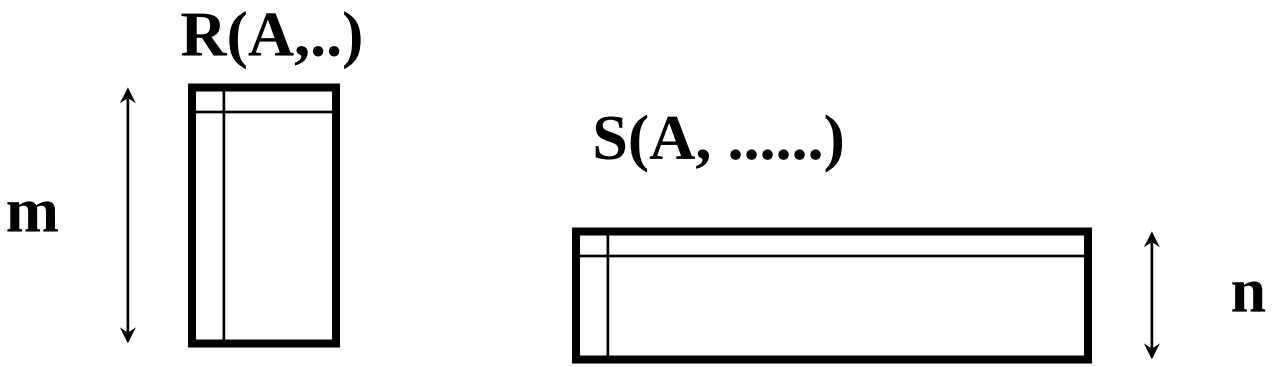
Inner Relation



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')?

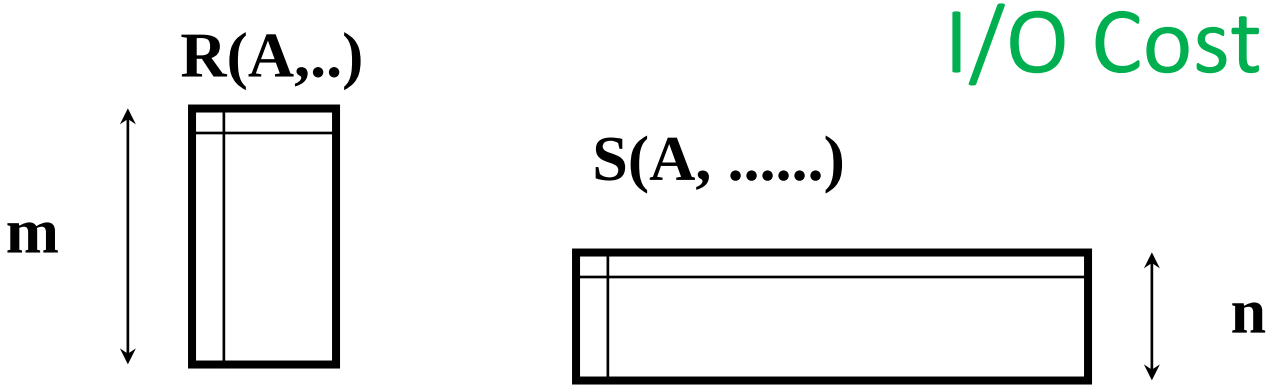


Simple Nested Loops Join

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

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

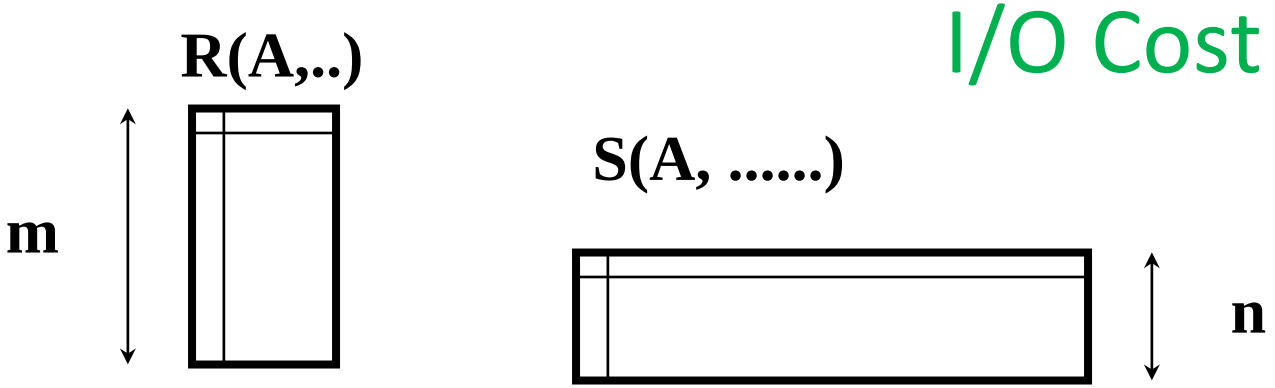
I/O Cost = M+m*N



Simple Nested Loops Join

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

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



I/O Cost = M+m*N

Can we do better?

Nested Loops Join: A Simple Refinement

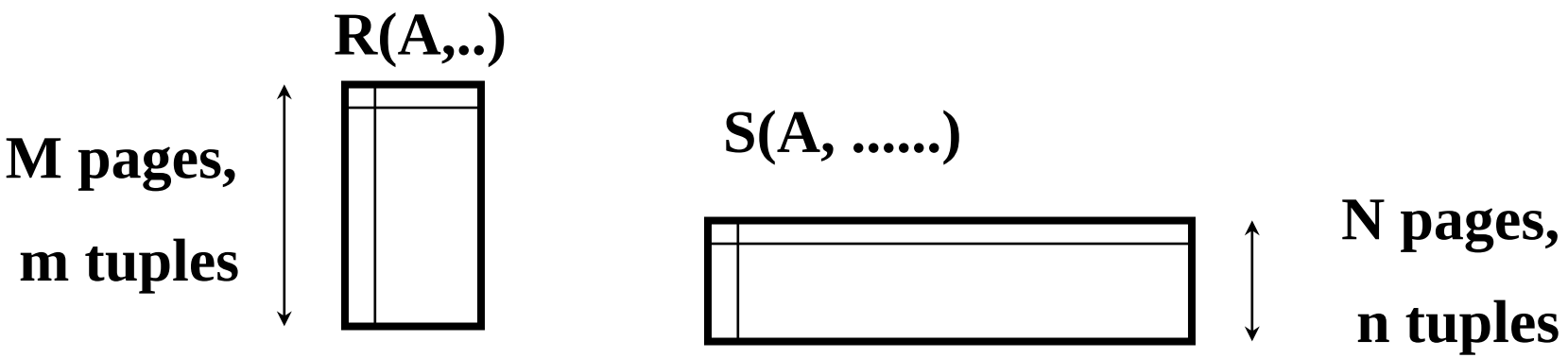
- Algorithm:

Read in a *page* of R

Read in a *page* of S

Print matching tuples

COST= ?



Nested Loops Join: A Simple Refinement

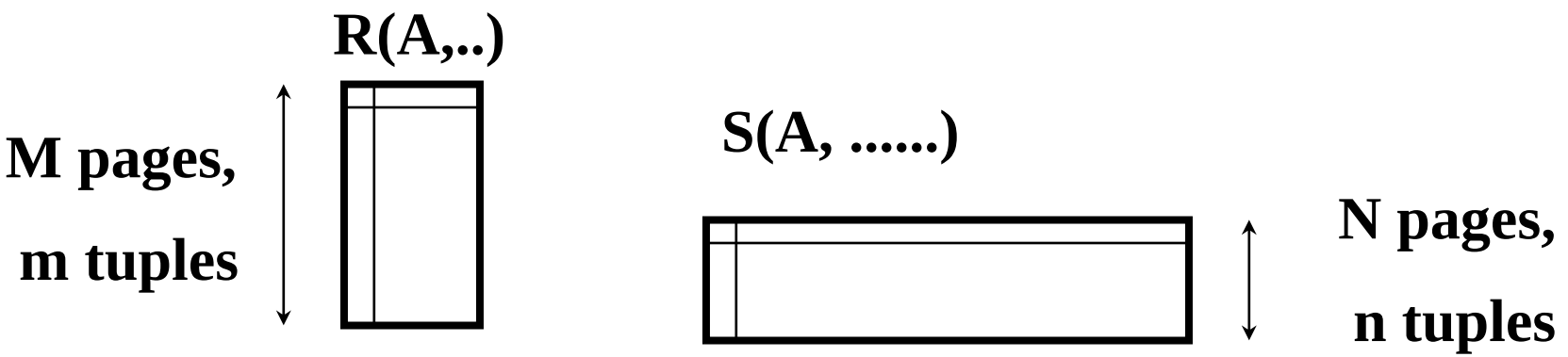
- Algorithm:

Read in a *page* of R

Read in a *page* of S

Print matching tuples

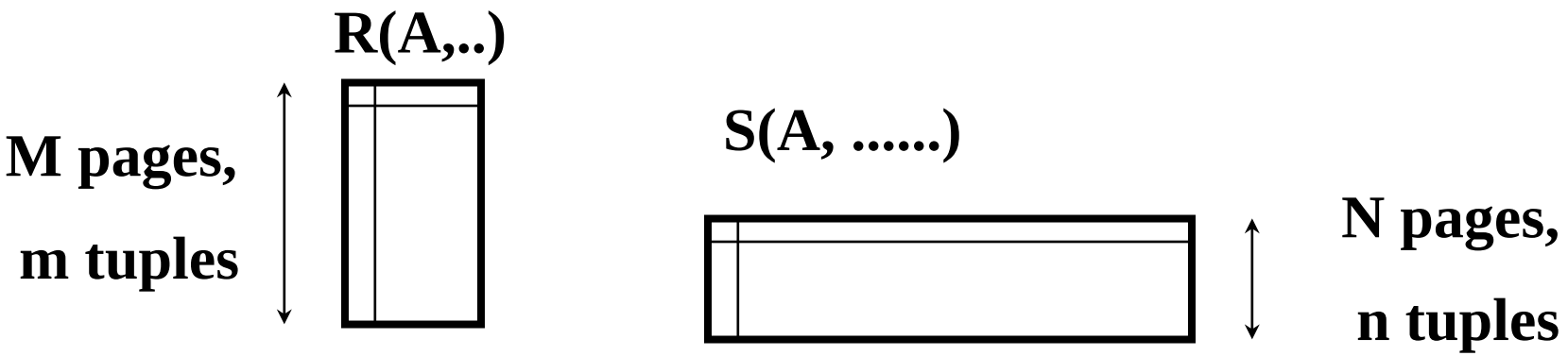
COST= M+M*N



Nested Loops Join

- Which relation should be the *outer*?

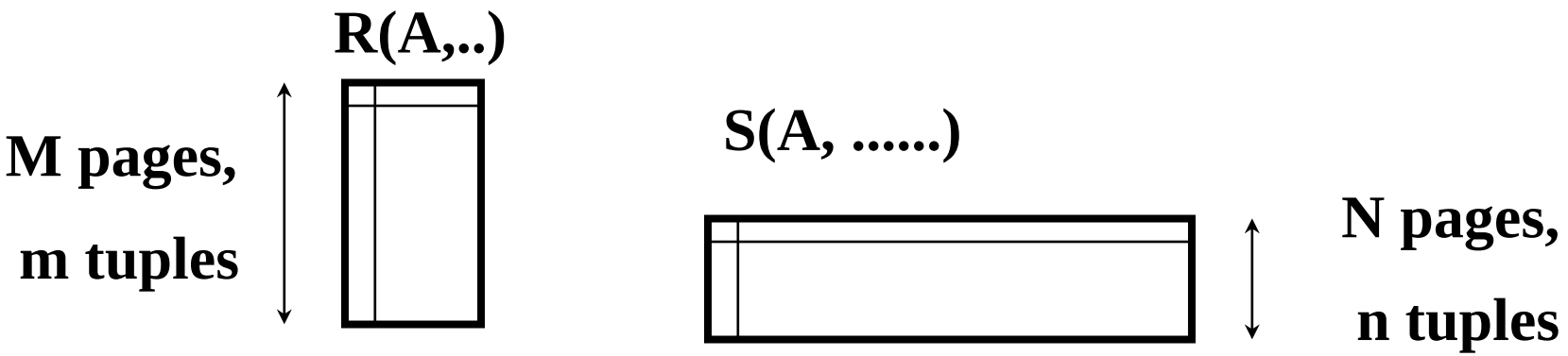
COST= M+M*N



Nested Loops Join

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

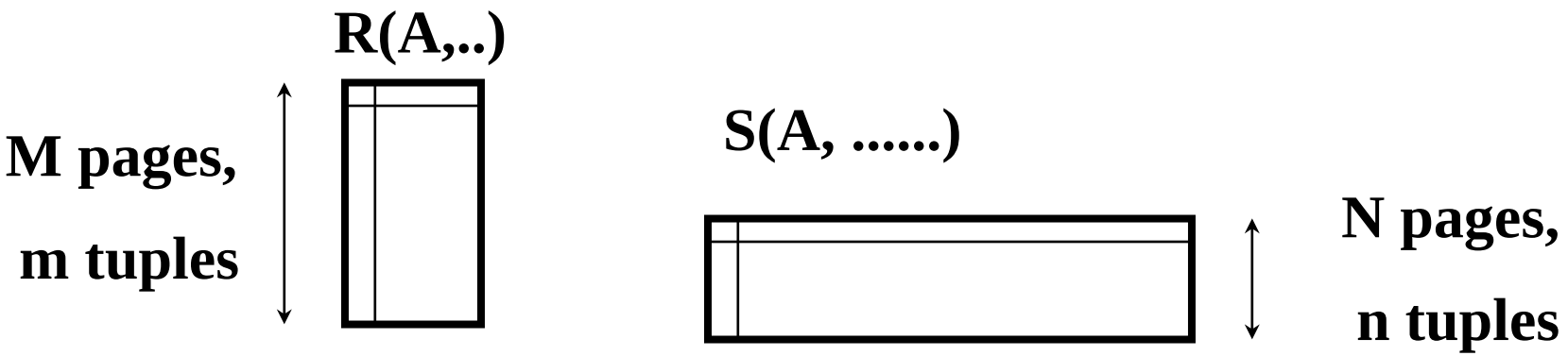
COST= M+M*N



Nested Loops Join

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

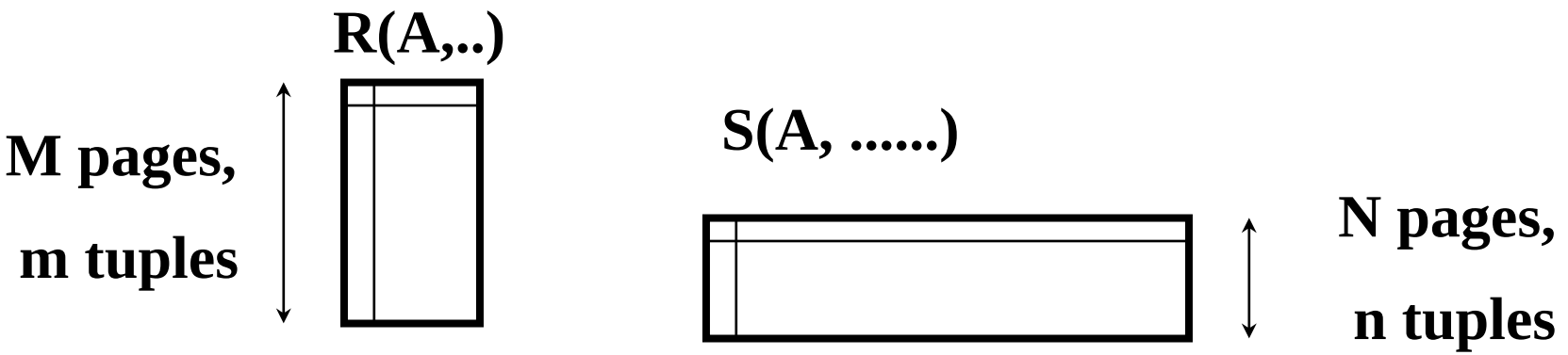
COST= M+M*N



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



Summary: Simple Nested Loops Join

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

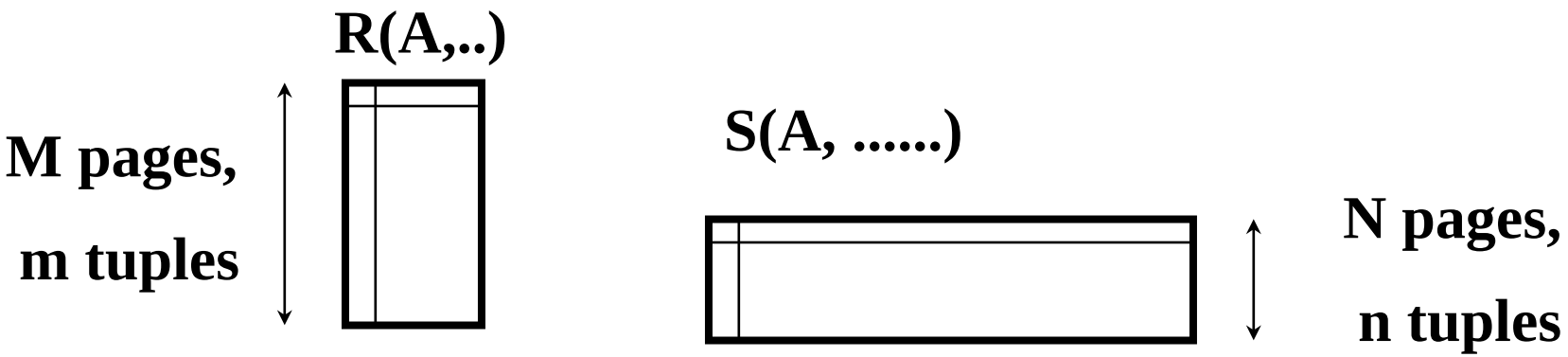
- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

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

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

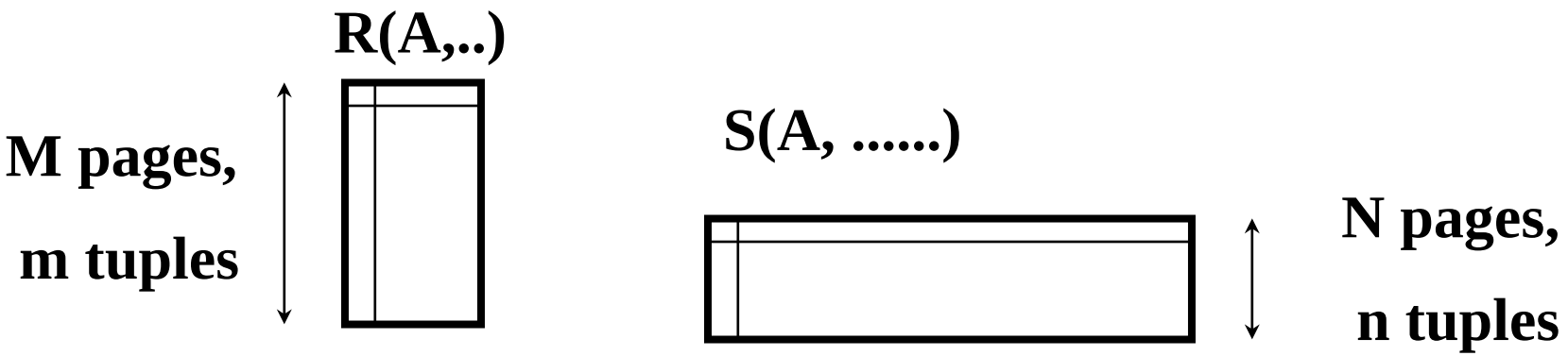
Block Nested Loops

- 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

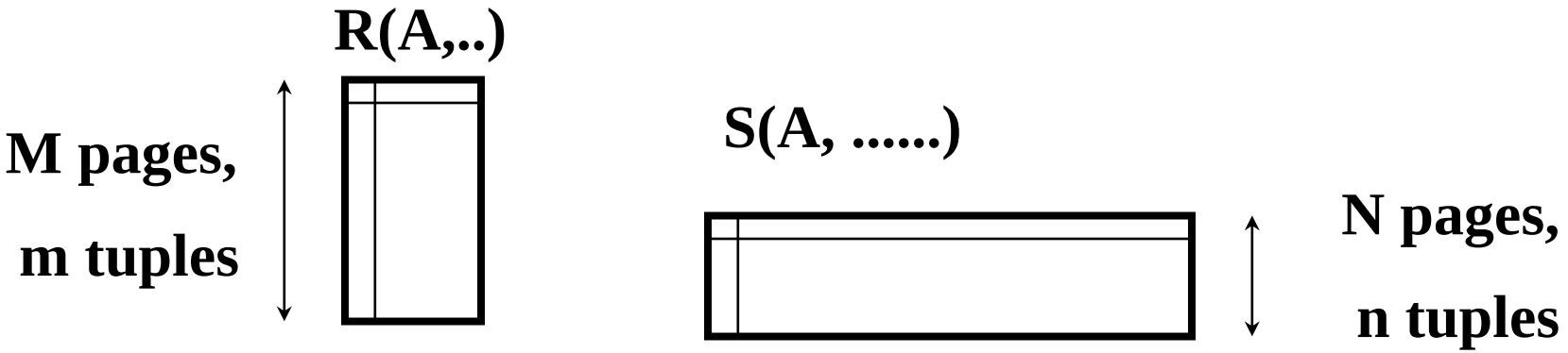
- Algorithm:

Read in *B-2* pages of R

Read in a page of S

Print matching tuples

COST= ?



Block Nested Loops

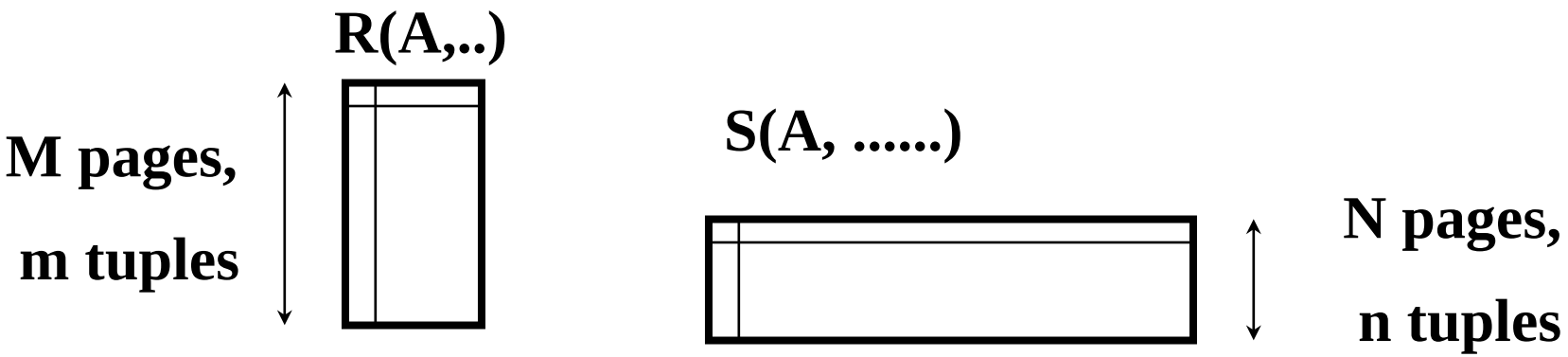
- Algorithm:

Read in *B-2* pages of R

Read in a page of S

Print matching tuples

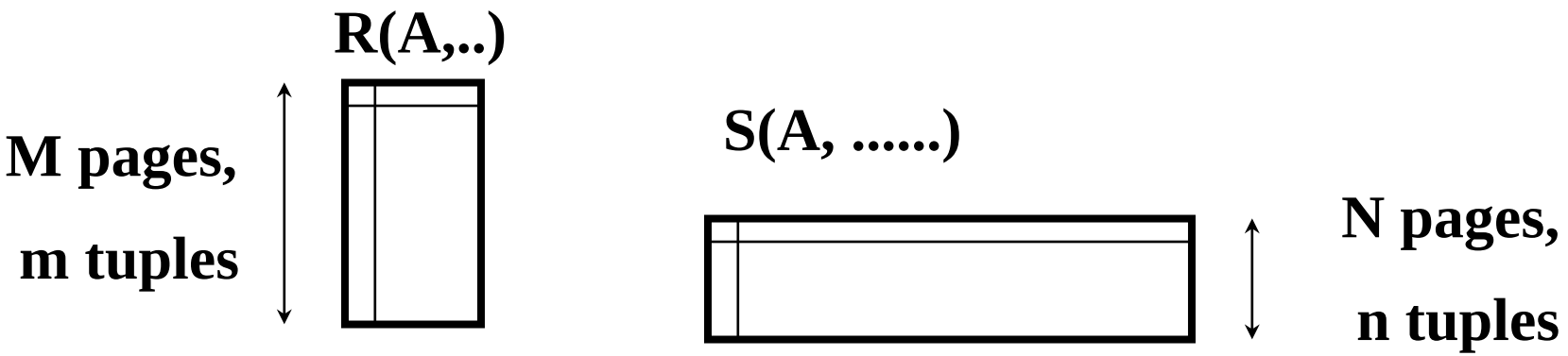
COST= M+M/(B-2)*N



Block Nested Loops

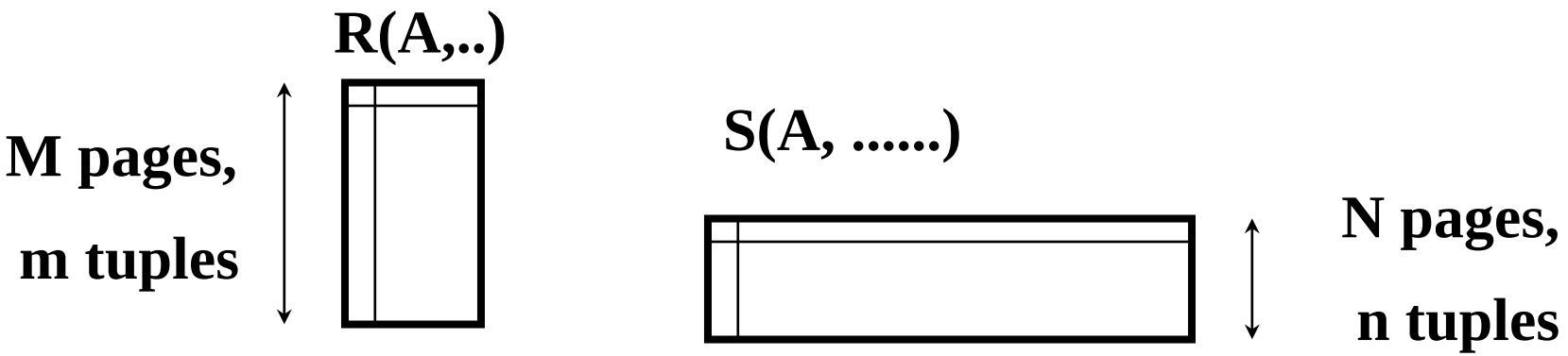
- And, actually:
- Cost = M + ceiling(M/(B-2)) * N

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



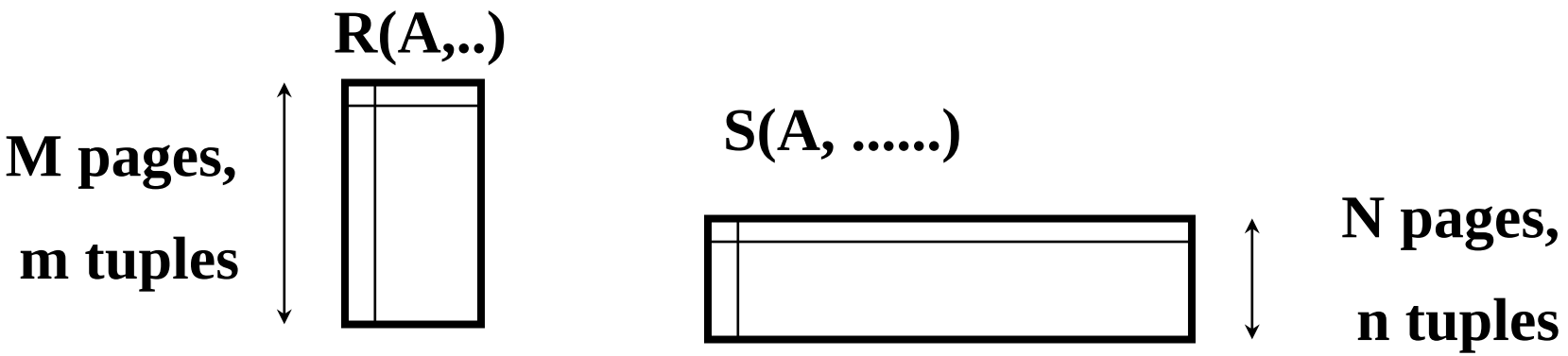
Block Nested Loops

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



Block Nested Loops

- If the smallest (outer) relation fits in memory?
- That is, ***B*** = M+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

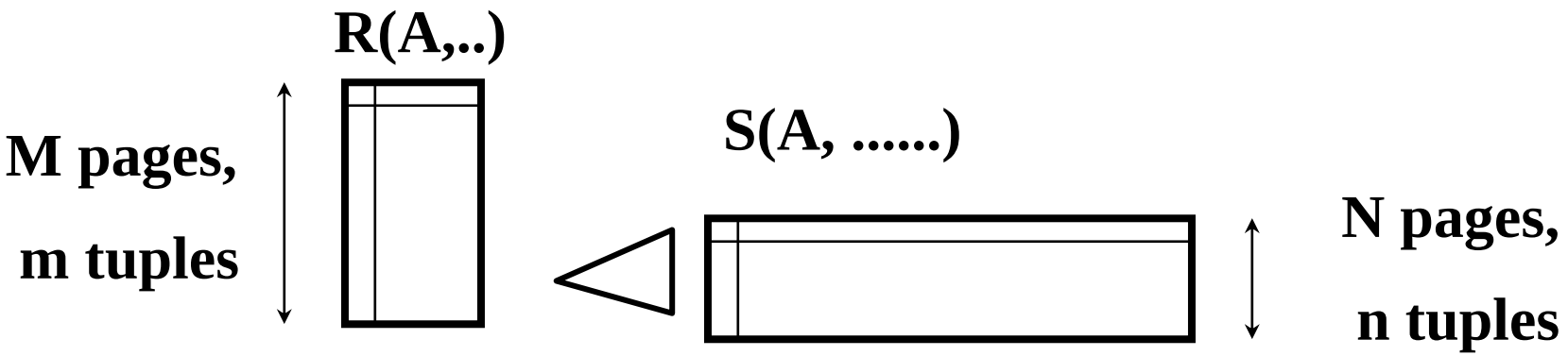
- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

- 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

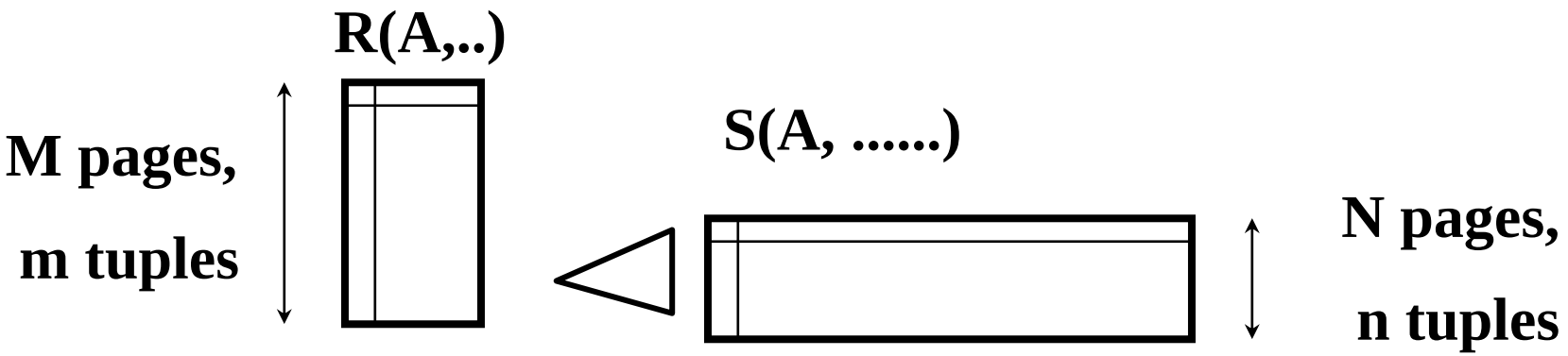
- What if there is an index on one of the relations on the join attribute(s)?
- A: Leverage the index by making the indexed relation *inner*



Index Nested Loops Join

- Assuming an index on S:

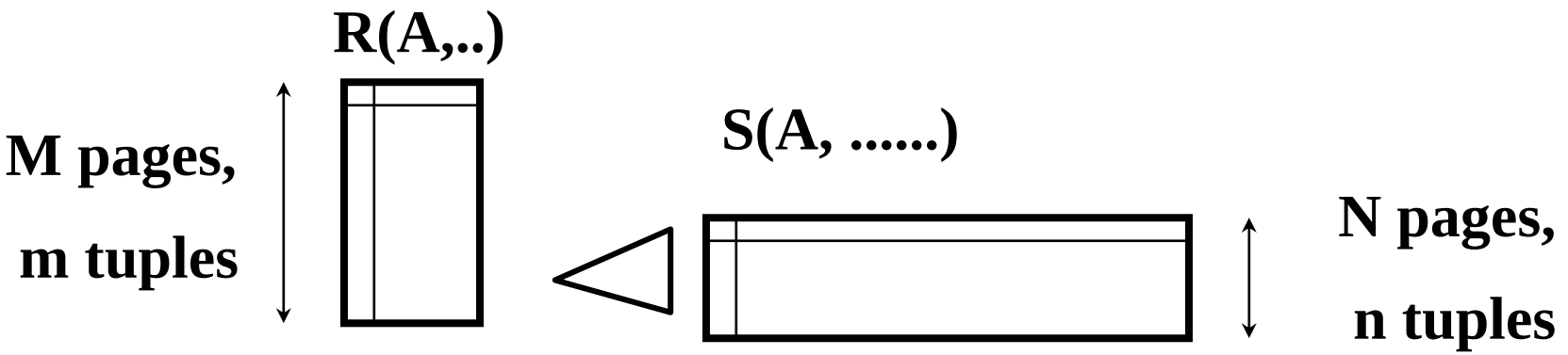
```
for each tuple r of R
    for each tuple s of S where r_i == s_j
        Add (r, s) to result
```



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

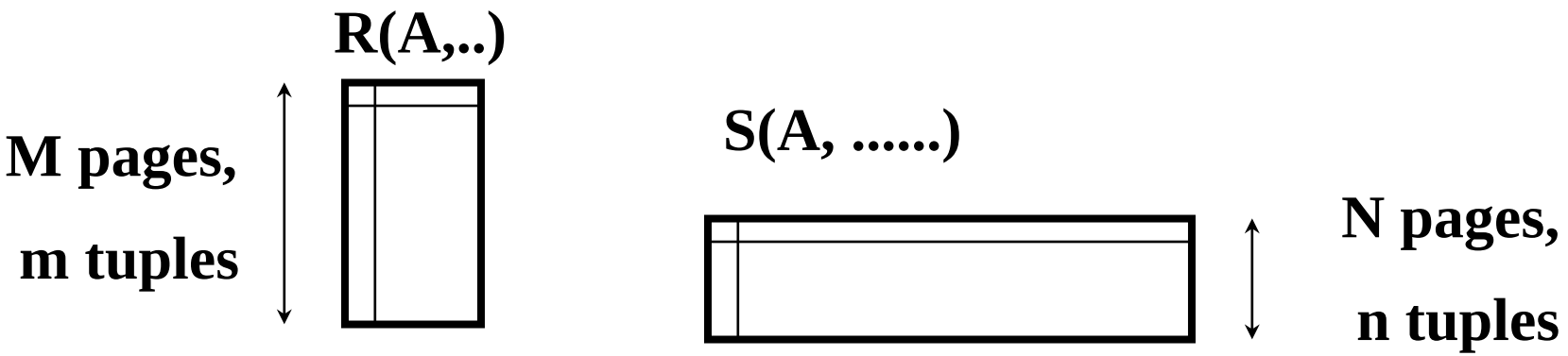
- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

- 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!



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

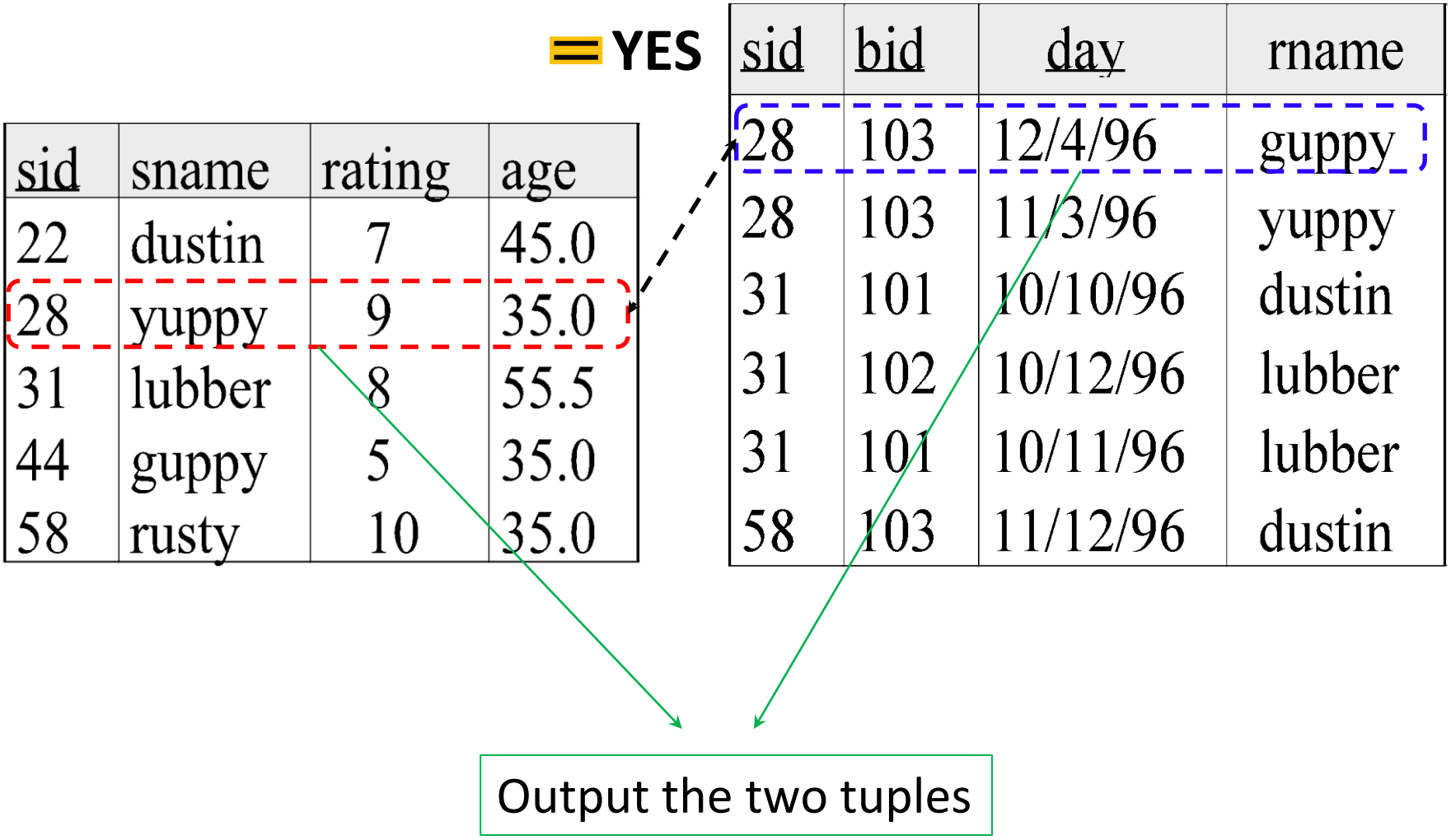
Sort-Merge Join: An Example

= ?

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



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

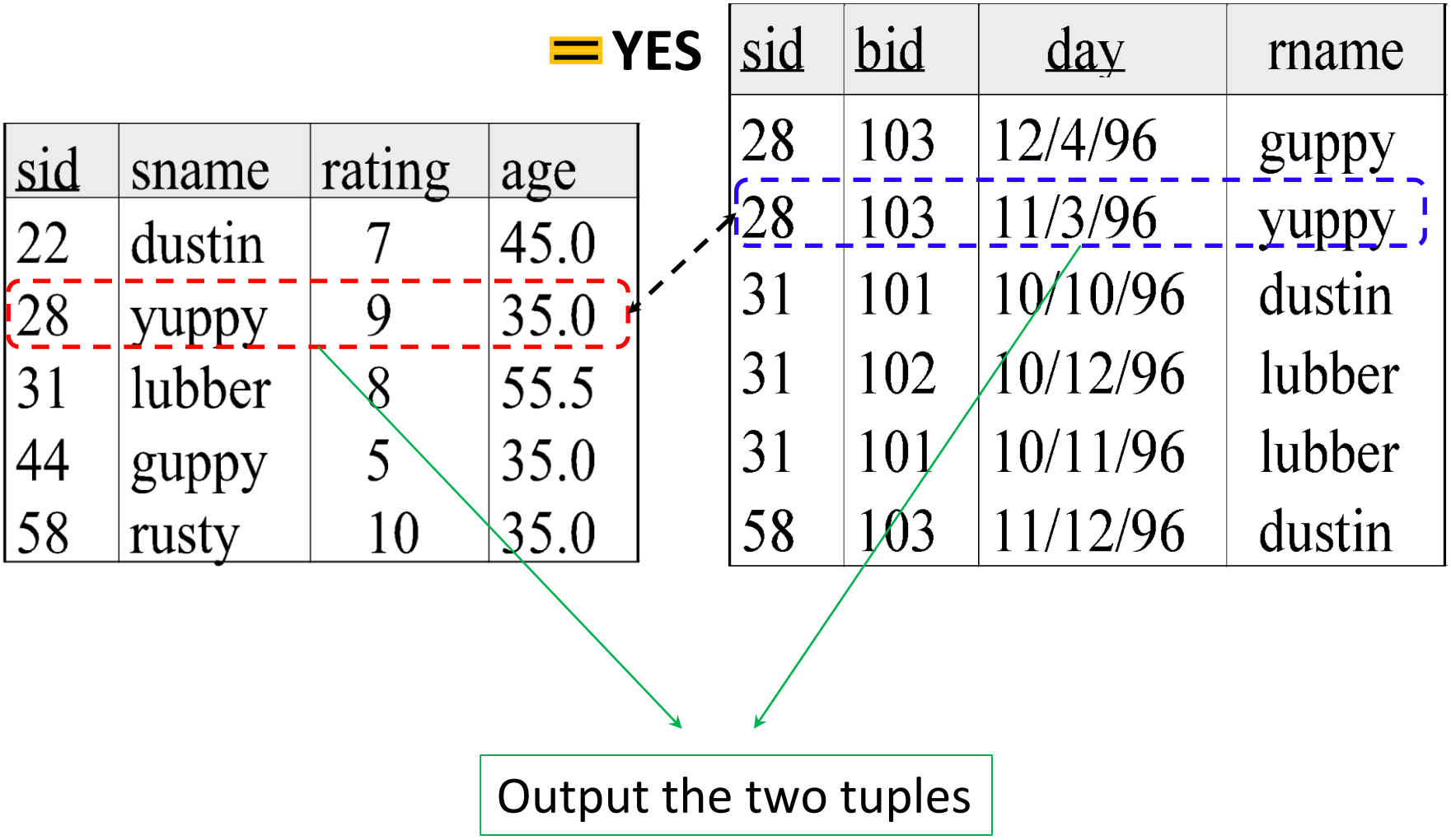
sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

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

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



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

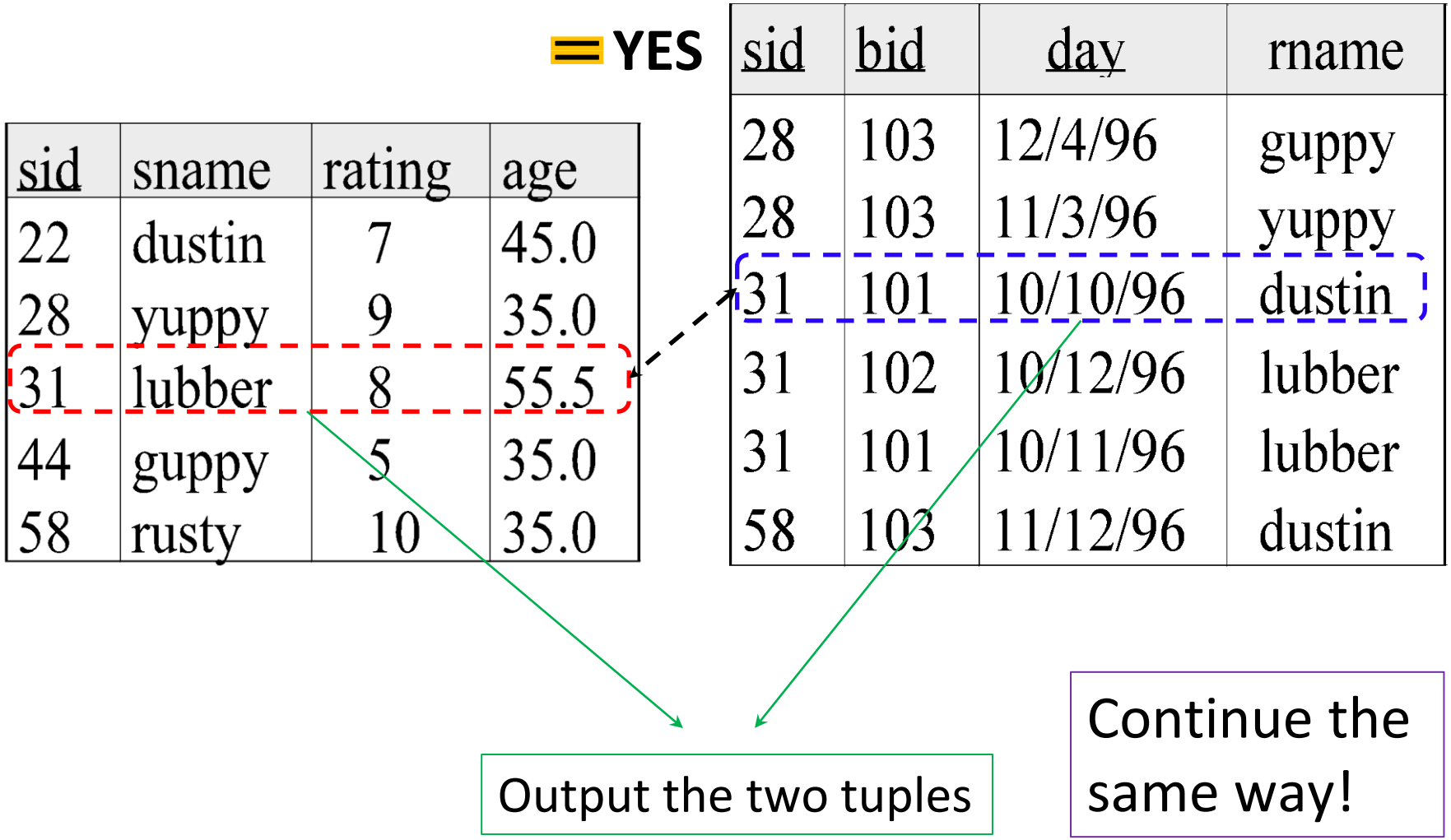
Sort-Merge Join: An Example

=?

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

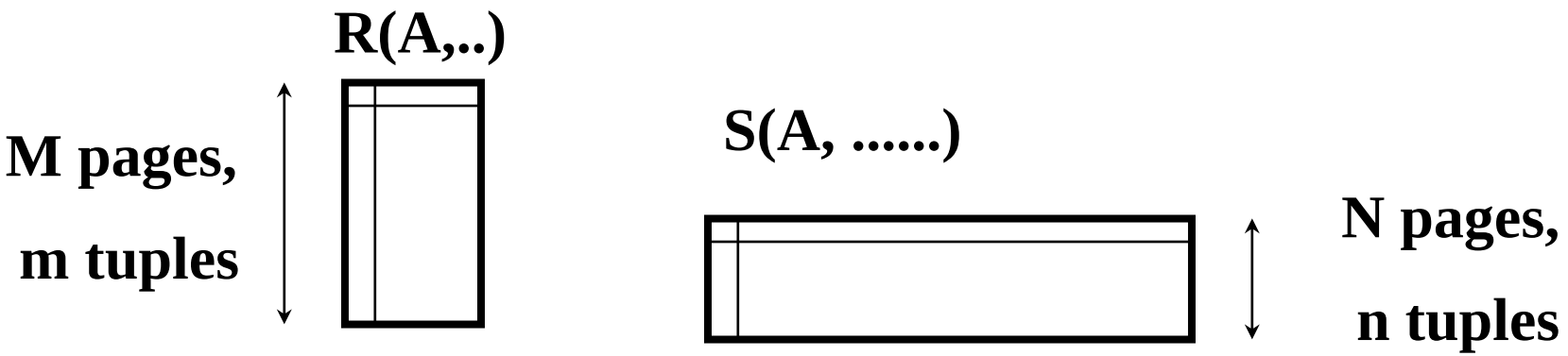
sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: An Example



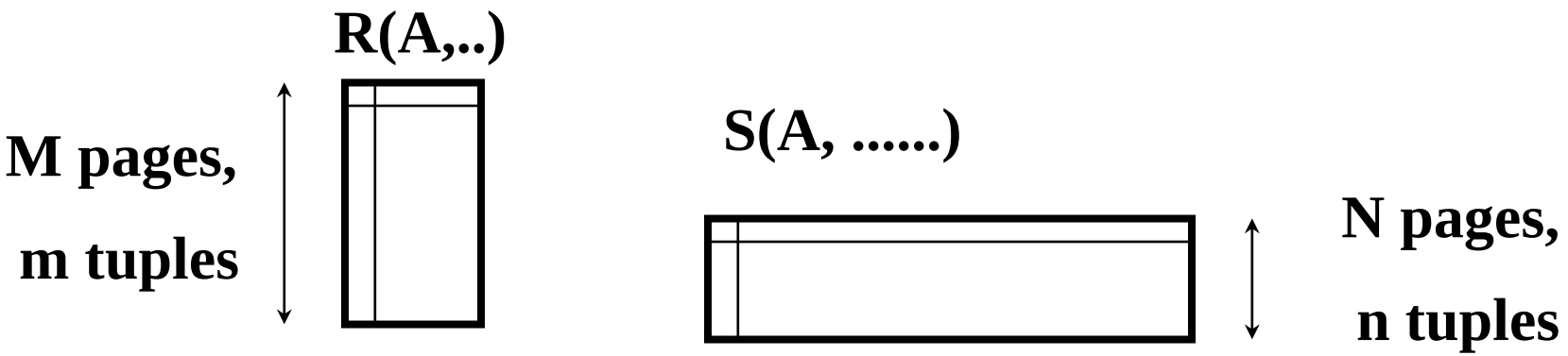
Sort-Merge Join

- What is the cost?
- ~ 2*M*logM/logB + 2*N* logN/logB + M + N



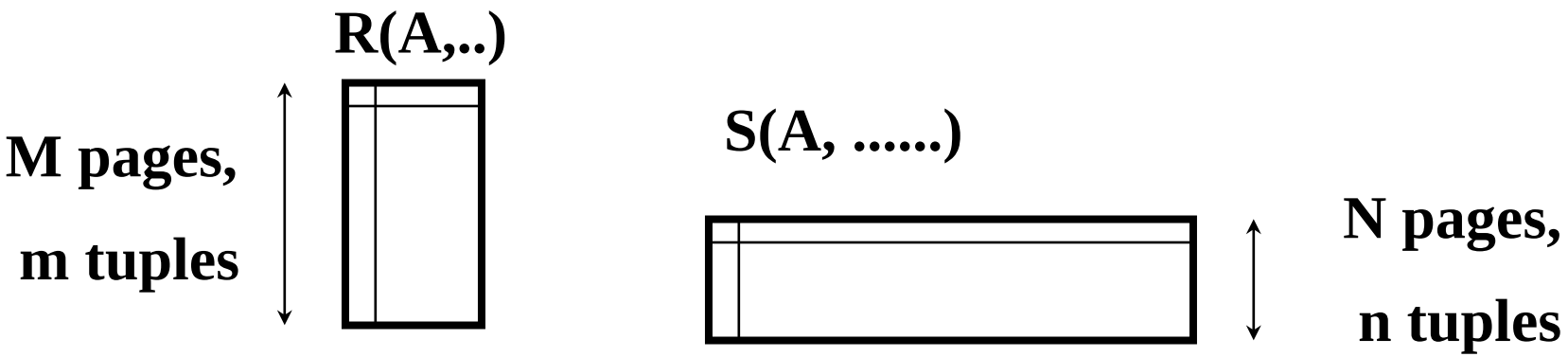
Sort-Merge Join

- 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



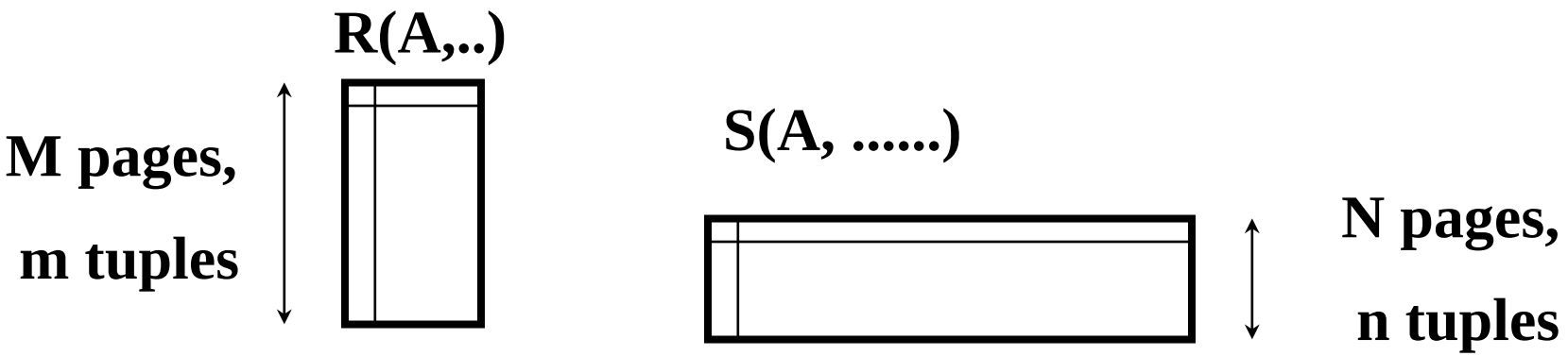
Sort-Merge Join

- 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



The Block Nested Loops Join is more sensitive to the buffer size!

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 ✓

Hash Join

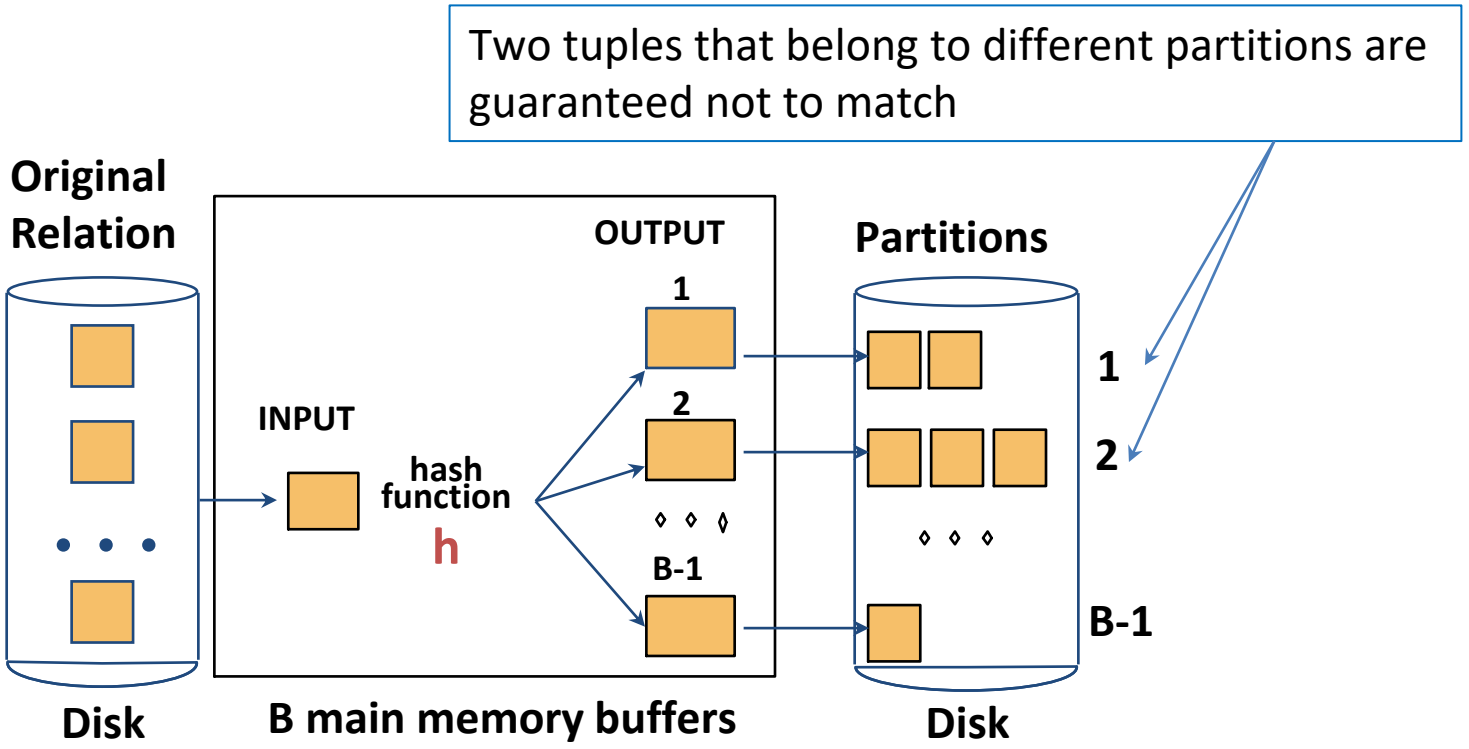
- The join algorithm based on hashing has two phases:
  - Partitioning (also called *Building*) Phase
  - Probing (also called *Matching*) Phase

- Idea: Hash both relations on the join attribute into ***k*** partitions, using the same hash function ***h***

- Premise: R tuples in partition ***i*** can join only with S tuples in the same partition ***i***

Hash Join: Partitioning Phase

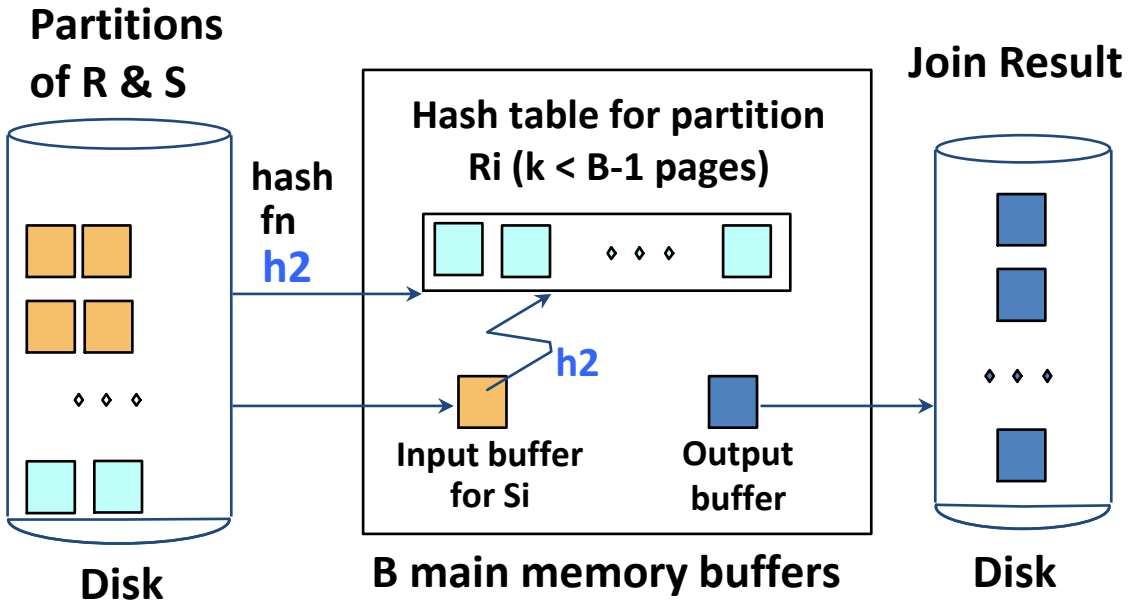
- Partition both relations using hash function ***h***



Hash Join: Probing Phase

- Read in a partition of R, hash it using ***h2*** **(<> *h*)**

- Scan the corresponding partition of S and search for matches



Hash Join: Cost

- What is the cost of the partitioning phase?
  - We need to scan R and S, and write them out once
  - Hence, cost is 2(M+N) I/Os

- What is the cost of the probing phase?
  - We need to scan each partition once (*assuming no partition overflows*) of R and S
  - Hence, cost is M + N I/Os

- Total Cost = 3 (M + N)

Hash Join: Cost (*Cont'd*)

- Total Cost = 3 (M + N)
- Joining Reserves and Sailors would cost 3 (500 + 1000) = 4500 I/Os
- Assuming 10ms per I/O, hash join takes less than 1 minute!
- This underscores the importance of using a good join algorithm (e.g., *Simple NL Join takes ~140 hours!*)

But, so far we have been assuming that partitions fit in memory!

Memory Requirements and Overflow Handling

- How can we increase the chances for a given partition in the probing phase to fit in memory?
  - Maximize the number of partitions in the building phase
- If we partition R (or S) into ***k*** partitions, what would be the size of each partition (in terms of B)?
  - At least ***k*** output buffer pages and 1 input buffer page
  - Given B buffer pages, ***k*** = B – 1
  - Hence, the size of an R (or S) partition = M/B-1
- What is the number of pages in the (in-memory) hash table built during the probing phase per a partition?
  - ***f***.M/B-1, where ***f*** is a *fudge factor*

Memory Requirements and Overflow Handling

- What do we need else in the probing phase?
  - A buffer page for scanning the S partition
  - An output buffer page

- What is a good value of B as such?
  - B > f.M/B-1 + 2
  - Therefore, we need ~ $B > \sqrt{f.M}$

- What if a partition overflows?
  - Apply the hash join technique *recursively* (as is the case with the projection operation)

Hash Join vs. Sort-Merge Join

- If $B > \sqrt{M}$ (M is the # of pages in the *smaller* relation) and we assume uniform partitioning, the cost of hash join is 3(M+N) I/Os

- If $B > \sqrt{N}$ (N is the # of pages in the *larger* relation), the cost of sort-merge join is 3(M+N) I/Os

Which algorithm to use, hash join or sort-merge join?

Hash Join vs. Sort-Merge Join

- If the available number of buffer pages falls between $\sqrt{M}$ and $\sqrt{N}$, hash join is preferred (why?)
- Hash Join shown to be highly parallelizable (*beyond the scope of the class*)
- Hash join is sensitive to data skew while sort-merge join is not
- Results are sorted after applying sort-merge join (may help "upstream" operators)
- Sort-merge join goes fast if one of the input relations is already sorted

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

Next Class