Database Management Systems

Lecture 7

Evaluating Relational Operators

Query Optimization

- queries
  - relational operators - building blocks
    - selection (σ)
      - selects a subset of records from a relation
    - projection (π)
      - eliminates certain columns from a relation
    - join ($\otimes$)
      - combines two relations
    - cross-product ($\times$)
      - returns every record in R1 concatenated with every record in R2
    - set-difference ($-$)
      - returns records that belong to R1 and don’t belong to R2
    - union ($\cup$)
      - returns all records in relations R1 and R2

- queries
  - relational operators - building blocks
    - intersection (∩)
      - returns records that belong to both R1 and R2
    - grouping and aggregate operators

- every operation returns a relation

=> operations can be composed

- optimizer
  - presented with a query written in SQL
  - takes into account the manner in which data is stored
    - information available in the system catalog
  - produces an efficient execution plan for evaluating the query
    - algorithm selection for each operator
      - factors:
        - sizes of:
          - relations
          - available buffer pool
        - the existence of:
          - indexes
          - sort orders
        - buffer replacement policy
    - application order for operators

- techniques to develop algorithms for operators
  - iteration
    - examine iteratively:
      - all tuples in input relations
      - data entries in an index (provided they contain all the necessary fields)
        - smaller than data records
  - indexing
    - selection condition, join condition
    - use an index to examine only the tuples that satisfy the condition
  - partitioning
    - partition the tuples
    - decompose operation into collection of less expensive operations on partitions

- techniques to develop algorithms for operators
  - partitioning
    - partitioning techniques
      - sorting
      - hashing

- access path
  - way of retrieving tuples from a relation
    - file scan
    - index *I* + matching selection condition *C*
  - *C* matches *I* if *I* can be used to retrieve just the tuples satisfying *C*
  - example:
    - condition *C: attr op value*, *op* ϵ *{<, <=, =, <>, >=, >}*
    - *C* matches *I* if:
    - the search key of *I* is *attr* and:
      - *I* is a tree index or
      - *I* is a hash index and *op* is *=*

  - relation *R* has an index *I* that matches selection *C*

=> at least 2 access paths

- access path
    - selectivity of an access path
        - number of retrieved pages when using the access path to obtain desired tuples
        - both data and index pages are counted
    - most selective access path
        - retrieves the fewest pages
        - minimizes data retrieval costs

- general selection conditions
  - boolean combination (i.e., expression using ∧, ∨) of terms of the form:
    - *attr op constant*
    - *attr1 op attr2*

```
SELECT *
FROM Exams
WHERE SID = 7 AND EDate = '04-01-2019'
```

$$\sigma_{SID=7 \wedge EDate='04-01-2019'}(Exams)$$

- general selection conditions
  - CNF – conjunctive normal form
    - standard form for general selection conditions
    - condition in CNF:
      - collection of conjuncts connected with the ∧ operator
    - a *conjunct* has one or more terms connected with the ∨ operator
    - *term*:
        - *attr op constant*
        - *attr1 op attr2*

  - example:

(EDate < '4-1-2019' ∧ Grade = 10) ∨ CID = 5 ∨ SID = 3

rewritten as

(EDate < '4-1-2019' ∨ CID = 5 ∨ SID = 3) ∧ (Grade = 10 ∨ CID = 5 ∨ SID = 3)

- general selection conditions matching an index
  - index I with search key <a, b, c>
  - examples

Condition	B+ tree index	Hash index
a = 10 AND b = 5 AND c = 2	Yes	Yes
a = 10 AND b = 5	Yes	No
b = 5	No	No
b = 5 AND c = 2	No	No
d = 2	No	No
a = 20 AND b = 10 AND c = 5 AND d > 11	Yes (partly)	Yes (partly)

- general selection conditions matching an index
  - index I1 with search key <a, b>
  - B+ tree index I2 with search key <c>

Condition	Indexes
c < 100 AND a = 3 AND b = 5	- use I1 or I2 to retrieve tuples - check terms in the selection condition that do not match the index for each retrieved tuple

- general selection conditions matching an index
  - index *I*, general selection condition *C* (CNF)

  - *I*
    - hash index
  - *C*
    - $\bigwedge_{i=1}^{n} T_i$
    - term T_i: $attr = value$
  - *I* matches *C* if *C* contains exactly one term for each attribute in the search key of *I*

- general selection conditions matching an index
  - index *I*, general selection condition *C* (CNF)

  - *I*
    - tree index
  - *C*
    - $\wedge_{i=1}^{n} T_i$
    - term T_i: $attr\ op\ value$
  - *I* matches *C* if *C* contains exactly one term for each attribute in a prefix of the search key of *I*
  - examples of prefixes for search key <a, b, c>: <a>, <a, b>

- running example - schema
  - Students (<u>SID: integer</u>, SName: string, Age: integer)
  - Courses (<u>CID: integer</u>, CName: string, Description: string)
  - Exams (<u>SID: integer, CID: integer</u>, EDate: date, Grade: integer, FacultyMember: string)

  - Students
    - every record has 50 bytes
    - there are 80 records / page
    - 500 pages
  - Courses
    - every record has 50 bytes
    - there are 80 records / page
    - 100 pages

- running example - schema
  - Students (SID: integer, SName: string, Age: integer)
  - Courses (CID: integer, CName: string, Description: string)
  - Exams (SID: integer, CID: integer, EDate: date, Grade: integer, FacultyMember: string)

  - Exams
    - every record has 40 bytes
    - there are 100 records / page
    - 1000 pages

- joins

```
SELECT *
FROM Exams E, Students S
WHERE E.SID = S.SID
```

- algebra: E $\otimes$ S
  - to be carefully optimized
  - size of E × S is large, so computing E × S followed by selection is inefficient
- E
  - M pages
  - p_E records / page
- S
  - N pages
  - p_S records / page
- evaluation: number of I/O operations

- joins – implementation techniques
  - iteration
    - Simple/Page-Oriented Nested Loops Join
    - Block Nested Loops Join
  - indexing
    - Index Nested Loops Join
  - partitioning
    - Sort-Merge Join
    - Hash Join

- equality join, one join column
  - join condition: $E_i = S_j$

Simple Nested Loops Join

```
foreach tuple e ∈ E do
    foreach tuple s ∈ S do
        if e_i == s_j then add <e, s> to the result
```

- for each record in the outer relation E, scan the entire inner relation S
- cost
  - $\mathbf{M + p_E * M * N}$ = 1000 + 100*1000*500 I/Os = 1000 + ($5 * 10^7$) I/Os

* E - M pages, p_E records / page * * 1000 pages * * 100 records / page*

* S - N pages, p_S records / page * * 500 pages * * 80 records / page *

Page-Oriented Nested Loops Join

```
foreach page pe ∈ E do
    foreach page ps ∈ S do
        if e_i == s_j then add <e, s> to the result
```

- for each page in E read each page in S
- pairs of records <e, s> that meet the join condition are added to the result (where record *e* is on page *pe*, and record *s* – on page *ps*)
- refinement of Simple Nested Loops Join

Page-Oriented Nested Loops Join

```
foreach page pe ∈ E do
    foreach page ps ∈ S do
        if e_i == s_j then add <e, s> to the result
```

- cost
  - **M + M*N** = 1000 + 1000*500 I/Os = 501.000 I/Os
    - significantly lower than the cost of Simple Nested Loops Join
    - improvement - factor of p_E
  - choose smaller table (*S*) as outer table

  => cost = 500 + 500 * 1000 I/Os = 500.500 I/Os

* E - M pages, p_E records / page *	* 1000 pages *	* 100 records / page*
* S - N pages, p_S records / page *	* 500 pages *	* 80 records / page *

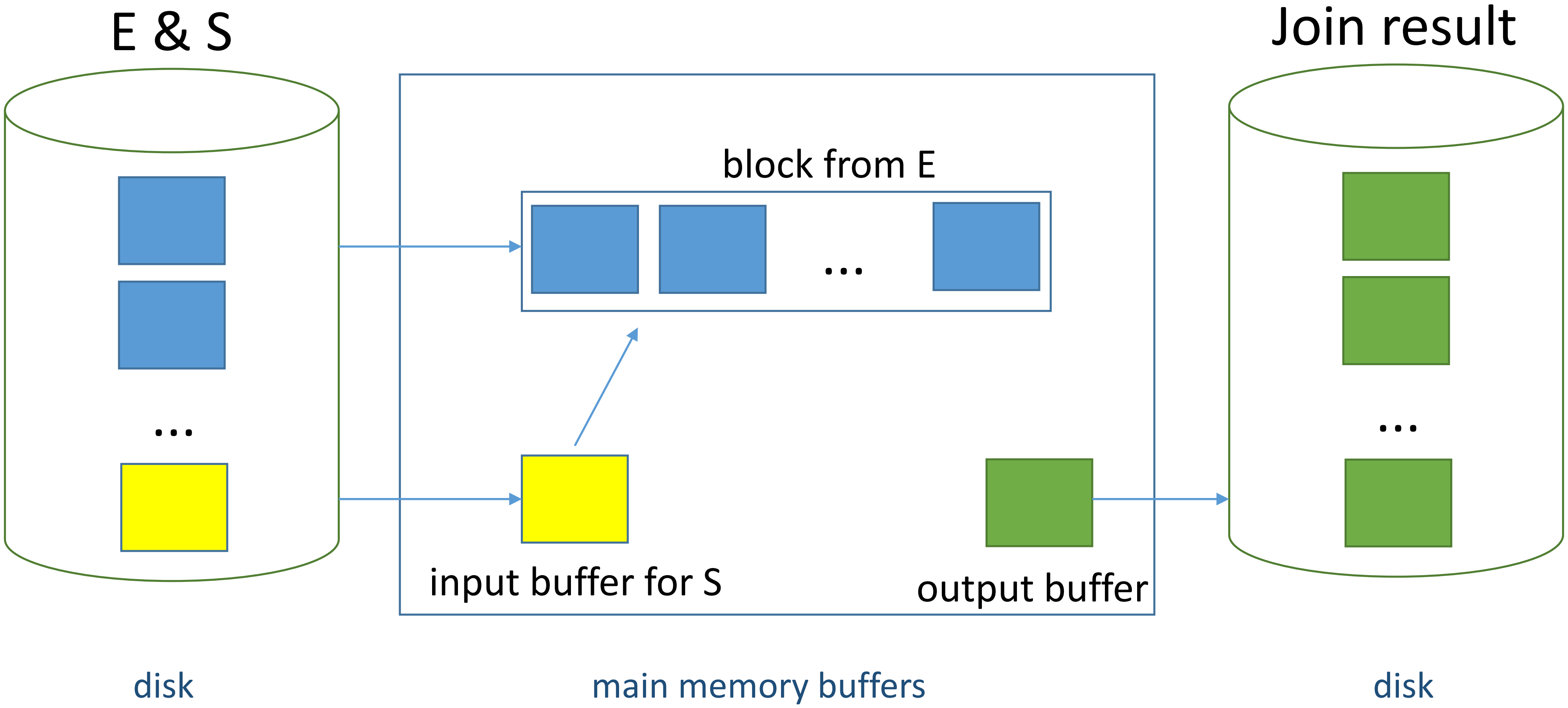
Block Nested Loops Join

- Simple Nested Loops Join - buffer pages are not used effectively
- improvement
  - store smaller relation R1 in memory
  - keep at least 2 extra buffer pages B1 and B2
  - use B1 to read larger relation R2
  - use B2 as output buffer
  - for each tuple in R2, search R1 for matching tuples

  => optimal cost: num. pages in R1 + num. pages in R2
  - refinement
    - build in-memory hash table for the smaller relation
    - I/O cost unchanged, but CPU cost is usually much lower

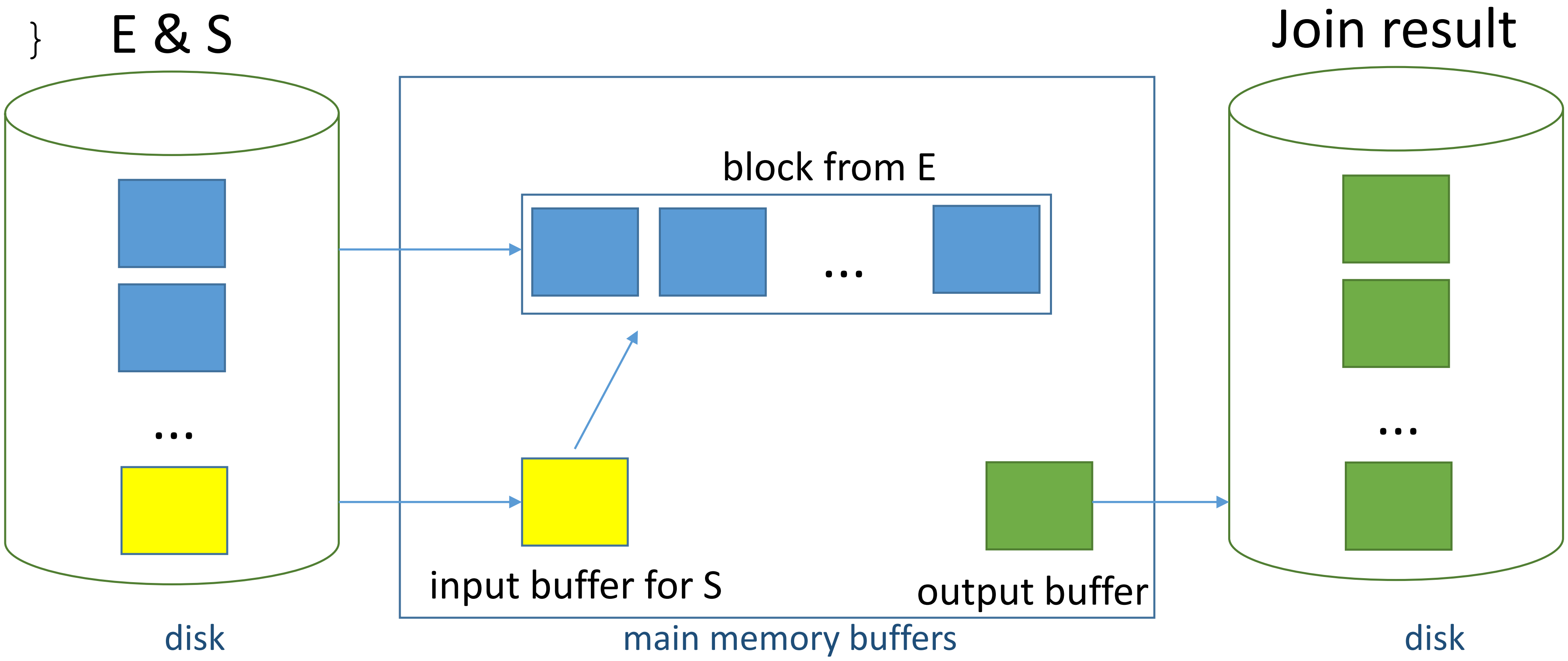
Block Nested Loops Join

- use one buffer page to scan inner table S
- use one page for the result
- use all remaining pages to read a *block* from outer table E



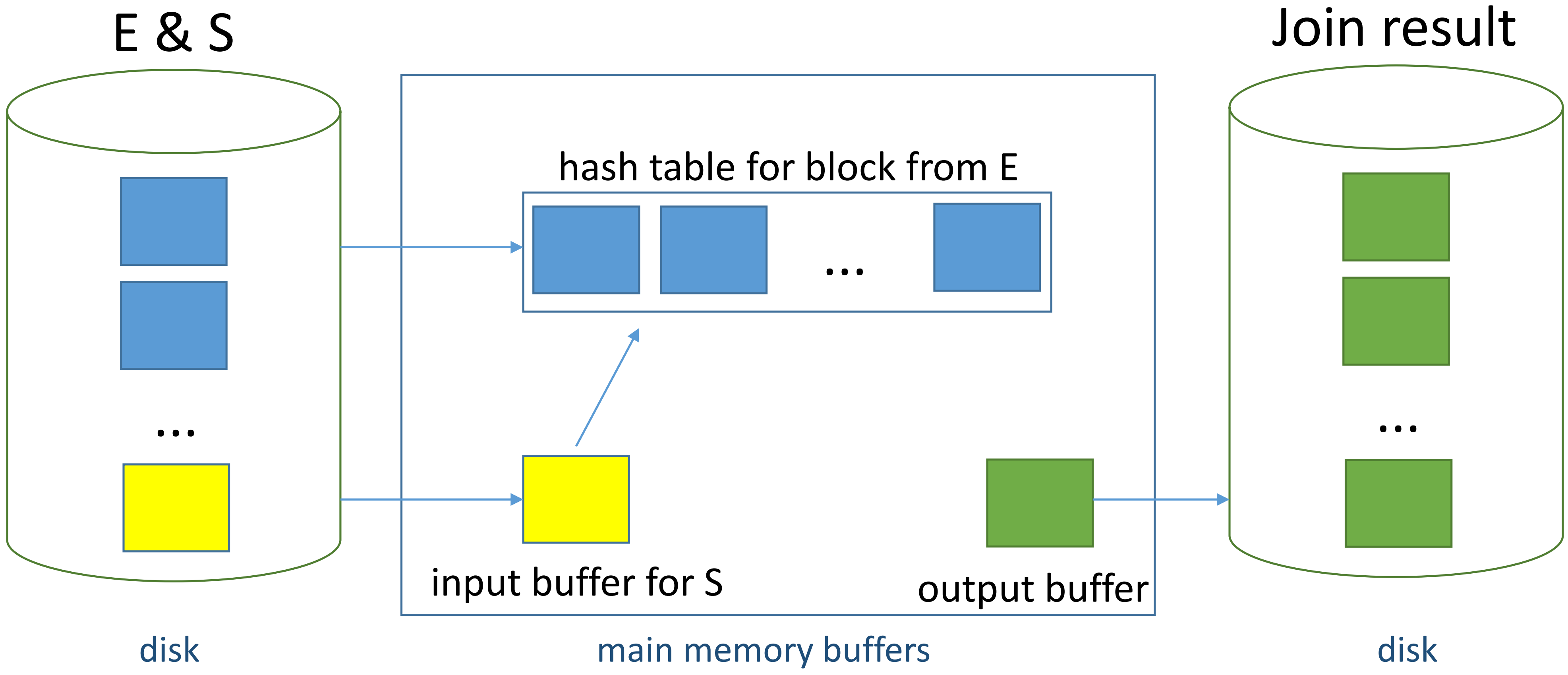
Block Nested Loops Join

```
foreach block be ∈ E do
   foreach page ps ∈ S do
     {
        for all pairs of tuples <e, s> that meet the join
          condition, where e ∈ be and s ∈ ps,
               add <e, s> to the result
     }
```



Block Nested Loops Join

- refinement to efficiently find matching tuples
  - main-memory hash table for the block of E
  - trade-off: reduce size of E block



Block Nested Loops Join

- cost
  - scan of outer table + # outer blocks * scan of inner table
  - # outer blocks = $\left\lceil \frac{\text{number of pages in outer table}}{\text{size of block}} \right\rceil$
  - outer table: Exams (E), block of 100 pages
    - scan cost for E: 1000 I/Os
    - number of blocks: $\left\lceil \frac{1000}{100} \right\rceil$ = 10
    - foreach block in E, scan Students (S): 10*500 I/Os

  => total cost = 1000 + 10 * 500 = **6000 I/Os**

* E - M pages, p_E records / page * * 1000 pages * * 100 records / page*

* S - N pages, p_S records / page * * 500 pages * * 80 records / page *

Block Nested Loops Join

- cost
  - scan of outer table + # outer blocks * scan of inner table
  - # outer blocks = $\left\lceil \frac{\text{number of pages in outer table}}{\text{size of block}} \right\rceil$
  - outer table: Exams (E)
    - buffer has 90 pages available for E, i.e., block of 90 pages

    => number of blocks: $\left\lceil \frac{1000}{90} \right\rceil$ = 12

    => S is scanned 12 times
    - scan cost for E: 1000 I/Os
    - foreach block in E, scan Students (S): 12*500 I/Os

  => total cost = 1000 + 12 * 500 = **7000 I/Os**

* E - M pages, p_E records / page * * 1000 pages * * 100 records / page*

* S - N pages, p_S records / page * * 500 pages * * 80 records / page *

Block Nested Loops Join

- cost
  - scan of outer table + # outer blocks * scan of inner table
  - # outer blocks = $\left\lceil \frac{\text{number of pages in outer table}}{\text{size of block}} \right\rceil$
  - outer table: Students (S), block of 100 pages
    - scan cost for S: 500 I/Os
    - number of blocks: $\left\lceil \frac{500}{100} \right\rceil$ = 5
    - for each block in S, scan E: 5 * 1000 I/Os

=> total cost = 500 + 5 * 1000 = **5050 I/Os**

* E - M pages, p_E records / page *   * 1000 pages *   * 100 records / page*

* S - N pages, p_S records / page *   * 500 pages *   * 80 records / page *

Index Nested Loops Join

```
foreach tuple e in E do
    foreach tuple s in S where e_i == s_j
        add <e, s> to the result
```

- if there is an index on the join column of S, S can be considered as inner table and the index can be used
- cost
  - M + ((M*p_E) * cost of finding corresponding records in S)

* E - M pages, p_E records / page *	* 1000 pages *	* 100 records / page*
* S - N pages, p_S records / page *	* 500 pages *	* 80 records / page *

Index Nested Loops Join

- for a record e in E:
  - cost of examining the index on S:
    - approx. 1.2 for a hash index (typical cost for hash indexes)
    - 2-4 for a B+-tree index

  - cost of reading corresponding records in S:
    - clustered index
      - plus one I/O for each outer tuple in E (typically)
    - nonclustered index
      - up to one I/O for each corresponding record in S
        (worst case – *n* matching records in S located on *n* different pages!)

Index Nested Loops Join

- hash index on SID in Students
- scan Exams
    - 1000 I/Os, 100*1000 records
- for each record in Exams
    - (on average) 1.2 I/Os to obtain the page in the index
      and
    - 1 I/O to retrieve the page in Students that contains the matching tuple (exactly one!)

=> cost to retrieve matching Students tuples: 1000 * 100 * (1.2 + 1) = 220.000

- total cost: 1000 + 220.000 = 221.000 I/Os

* E - M pages, p_E records / page *   * 1000 pages *   * 100 records / page*

* S - N pages, p_S records / page *   * 500 pages *   * 80 records / page *

References

- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010, http://codex.cs.yale.edu/avi/db-book/
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html