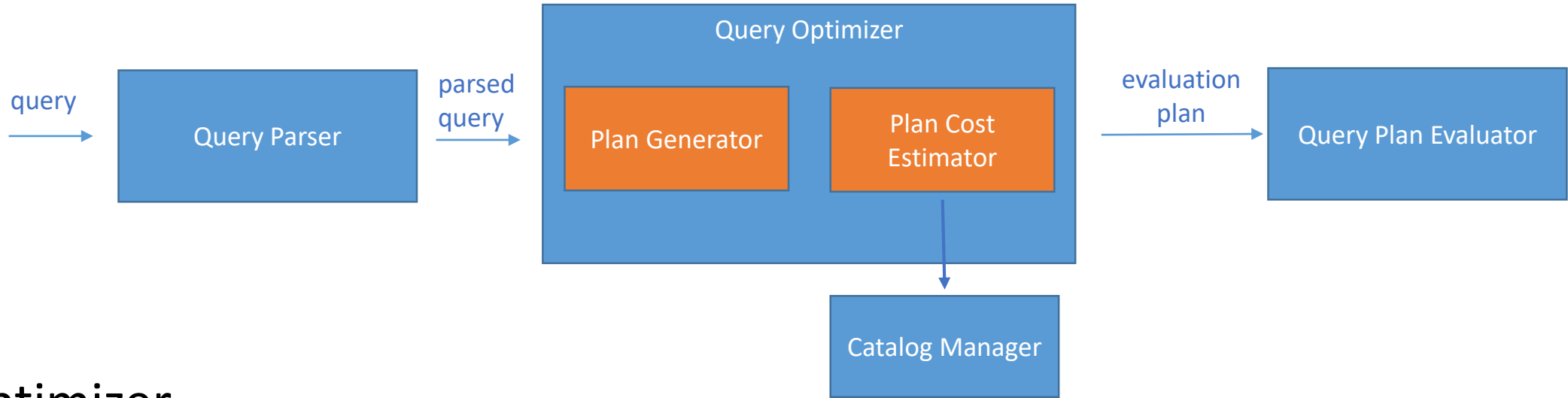# Database Management Systems

Lecture 9

Evaluating Relational Operators

Query Optimization (IV)

Query Optimization

- optimizer
  - objective
    - given a query Q, find a good evaluation plan for a Q
  - generates alternative plans for Q, estimates their costs, and chooses the one with the least estimated cost
  - uses information from the system catalogs

- 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, EDate: date</u>, Grade: integer)

- Students
  - every record has 50 bytes
  - there are 80 records / page
  - 500 pages
- Courses
  - every record has 40 bytes
  - there are 100 records / page
  - 1 page

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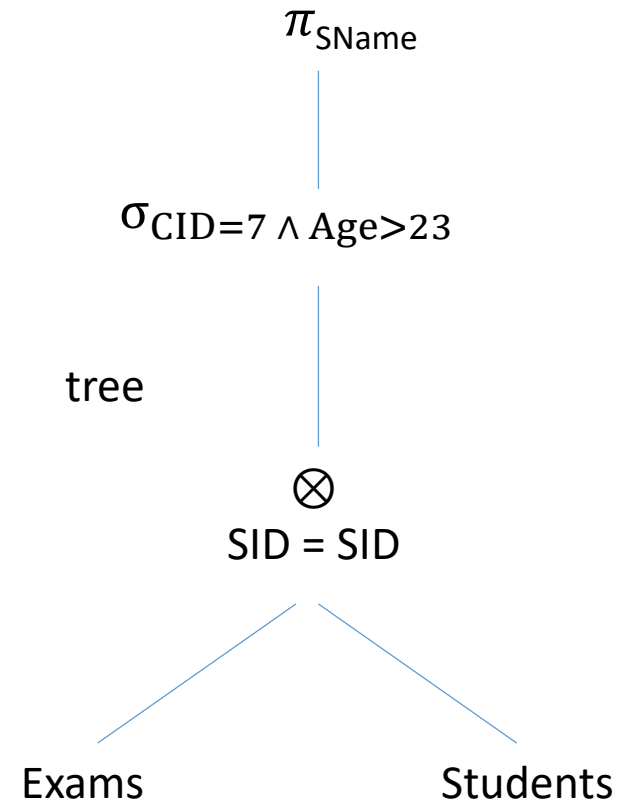 - there are 100 records / page
  - 1000 pages

Query Evaluation Plans

```
SELECT S.SName
FROM Exams E, Students S          query
WHERE E.SID = S.SID AND E.CID = 7
      AND S.Age > 23
```

$$\pi_{SName}(\sigma_{CID=7 \wedge Age>23}(Exams \otimes_{SID=SID} Students))$$

relational algebra expression

π_{SName}

$\sigma_{CID=7 \wedge Age>23}$

tree

$\otimes$

SID = SID
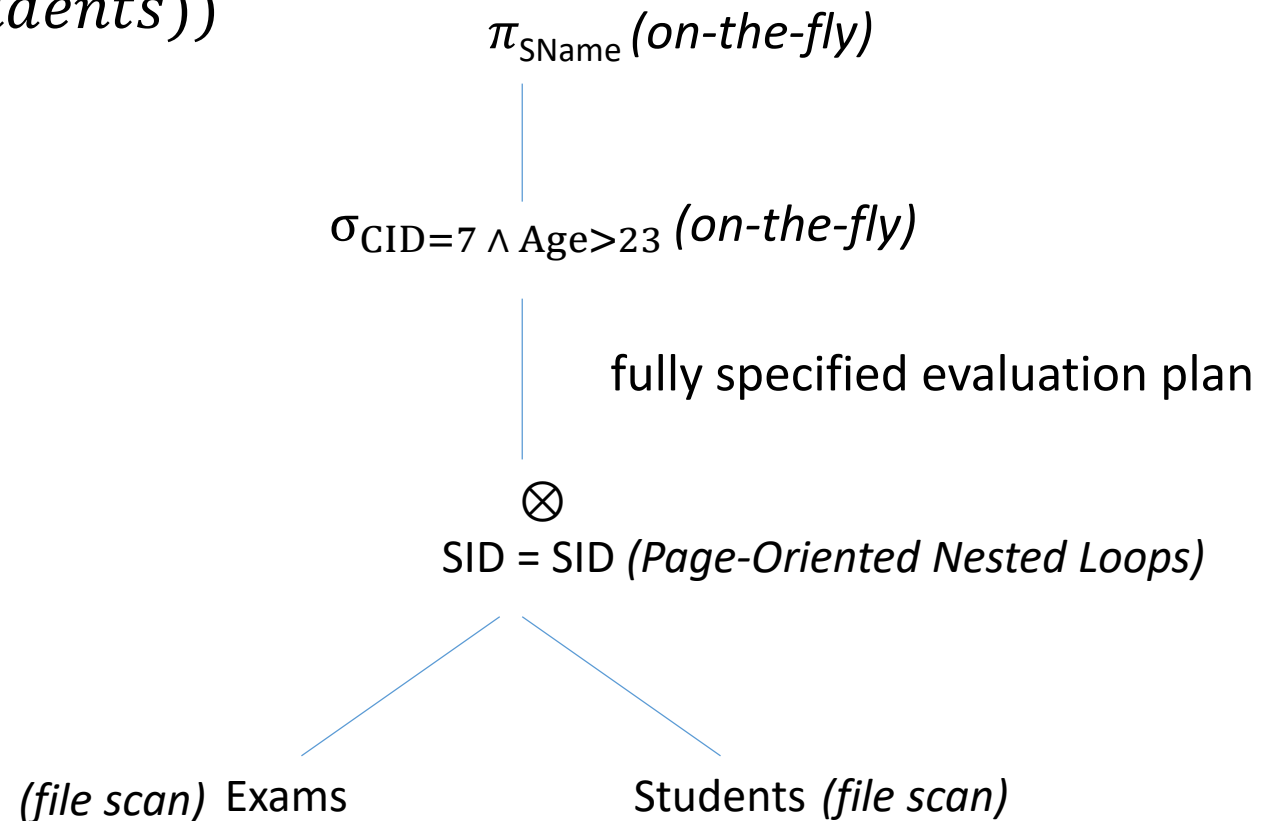
Exams   Students

Query Evaluation Plans

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
    AND S.Age > 23
```

$$\pi_{SName}(\sigma_{CID=7 \wedge Age>23}(Exams \otimes_{SID=SID} Students))$$

- query evaluation plan
  - extended relational algebra tree
  - node – annotations
    - relation
      - access method
    - relational operator
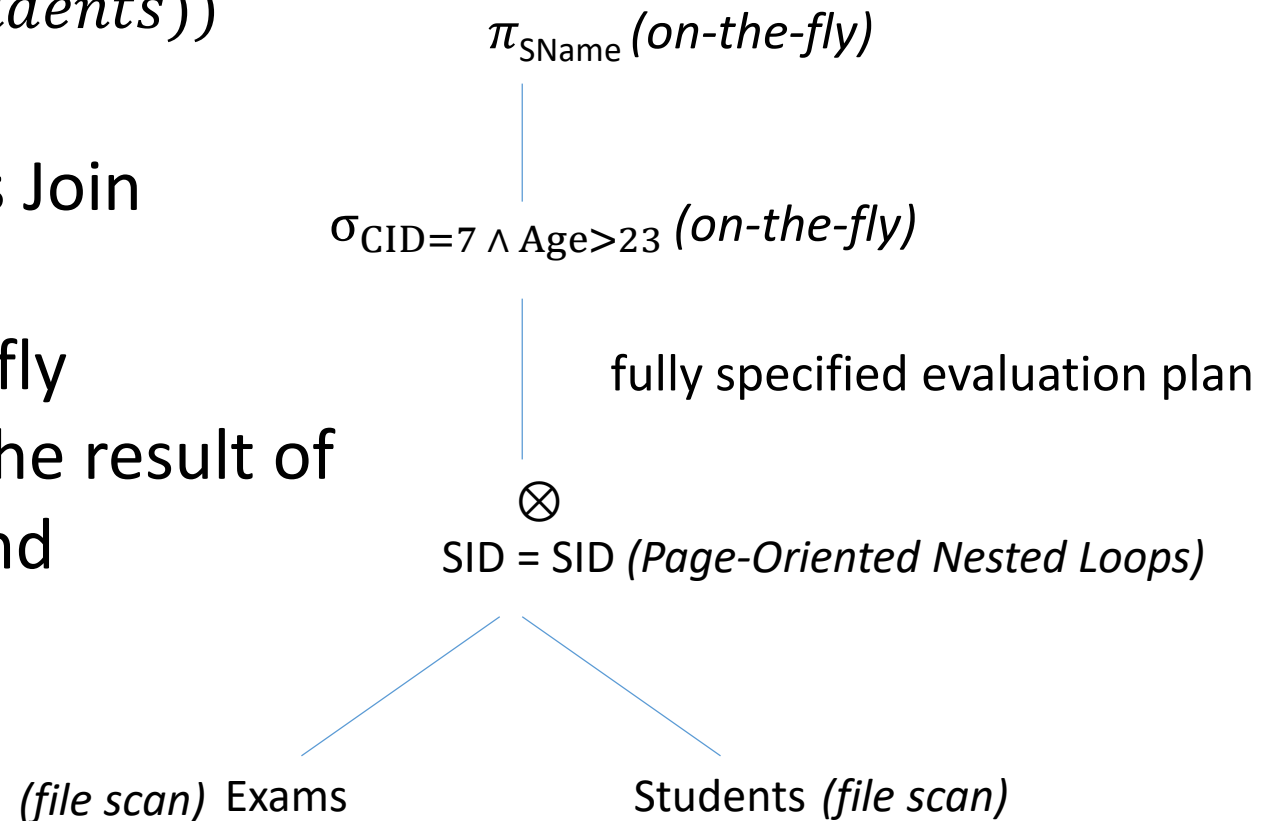   - implementation method

π_{SName} *(on-the-fly)*

$\sigma_{CID=7 \wedge Age>23}$ *(on-the-fly)*

fully specified evaluation plan

$\otimes$
SID = SID *(Page-Oriented Nested Loops)*

(file scan) Exams   Students *(file scan)*

Query Evaluation Plans

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
     AND S.Age > 23
```

$$\pi_{SName}(\sigma_{CID=7 \land Age>23}(Exams\otimes_{SID=SID} Students))$$

- page-oriented Simpled Nested Loops Join
  - Exams – outer relation
- selection, projection applied on-the-fly to each tuple in the join result, i.e., the result of the join (before applying selection and projection) is not stored

π_{SName} *(on-the-fly)*

$\sigma_{CID=7 \land Age>23}$ *(on-the-fly)*

fully specified evaluation plan

$\otimes$

SID = SID *(Page-Oriented Nested Loops)*

(file scan) Exams   Students *(file scan)*

Pipelined Evaluation

```
SELECT *
FROM Exams
WHERE EDate > '1-1-2017' AND Grade > 8
           T1                    T2
```

$$\sigma_{Grade>8}(\sigma_{EDate>'1-1-2017'}(Exams))$$

- index *I* matches *T1*
- v1 - *materialization*
  - evaluate *T1*
  - write out result tuples to temporary relation *R,* i.e., tuples are *materialized*
  - apply the 2nd selection to *R*
  - cost: read and write *R*

Pipelined Evaluation

```
SELECT  *
FROM  Exams
WHERE  EDate > '1-1-2017'  AND  Grade > 8
```
T1 *T2*

- v2 – *pipelined evaluation*
  - apply the 2nd selection to each tuple in the result of the 1st selection as it is produced
  - i.e., 2nd selection operator is applied *on-the-fly*
  - saves the cost of writing out / reading in the temporary relation *R*

Query Blocks – Units of Optimization
- optimize SQL query Q
  - parse Q => collection of query *blocks*
  - optimizer:
    - optimize one block at a time
- query *block - SQL* query:
  - without nesting
  - with exactly: one SELECT clause, one FROM clause
  - with at most: one WHERE clause, one GROUP BY clause, one HAVING clause
    - WHERE condition - CNF

Query Blocks – Units of Optimization

- query Q:

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                    S.Age = (SELECT MAX(S2.Age)
                            FROM Students S2)
                                            nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

- decompose query into a collection of blocks without nesting

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                S.Age = Reference to nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

Query Blocks – Units of Optimization

* block optimization

- express query block as a relational algebra expression

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                S.Age = Reference to nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

$\pi_{S.SID, MIN(E.EDate)}($
$HAVING_{COUNT(*) > 2}($
$GROUP\ BY_{S.SID}($
$\sigma_{S.SID = E.SID \wedge E.CID = C.CID \wedge C.Description = 'Elective' \wedge S.Age = value\_from\_nested\_block}($
$\quad Students \times Exams \times Courses))))$

- GROUP BY, HAVING – operators in the extended algebra used for plans
- argument list of projection can include aggregate operations

Query Blocks – Units of Optimization

- query Q treated as a $\sigma\,\pi\,\times$ algebra expression
- the remaining operations in Q are performed on the result of the $\sigma\,\pi\,\times$ expression

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                S.Age = Reference to nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

$\pi_{S.SID,\ E.EDate}($
$\sigma_{S.SID\ =\ E.SID\ \wedge\ E.CID\ =\ C.CID\ \wedge\ C.Description\ =\ 'Elective'\ \wedge\ S.Age\ =\ value\_from\_nested\_block}($
$\qquad Students \times Exams \times Courses\ ))$

- attributes in GROUP BY, HAVING are added to the argument list of projection
- aggregate expressions in the argument list of projection are replaced with their argument attributes

Query Blocks – Units of Optimization
* block optimization
- find best plan P for the $\sigma \, \pi \, \times$ expression
- evaluate P => result set RS
- sort/hash RS => groups
- apply HAVING to eliminate some groups
- compute aggregate expressions in SELECT for each remaining group

$\pi_{S.SID, MIN(E.EDate)}($
$HAVING_{COUNT(*) > 2}($
$GROUP \; BY_{S.SID}($
$\pi_{S.SID, E.EDate}($
$\sigma_{S.SID = E.SID \land E.CID = C.CID \land C.Description = 'Elective' \land S.Age = value\_from\_nested\_block}($
$Students \times Exams \times Courses \,)))))$

IBM's System R Optimizer
- tremendous influence on subsequent relational optimizers
- design choices:
  - use statistics to estimate the costs of query evaluation plans
  - consider only plans with binary joins in which the inner relation is a base relation
  - focus optimization on SQL queries without nesting
  - don't eliminate duplicates when performing projections (unless DISTINCT is used)
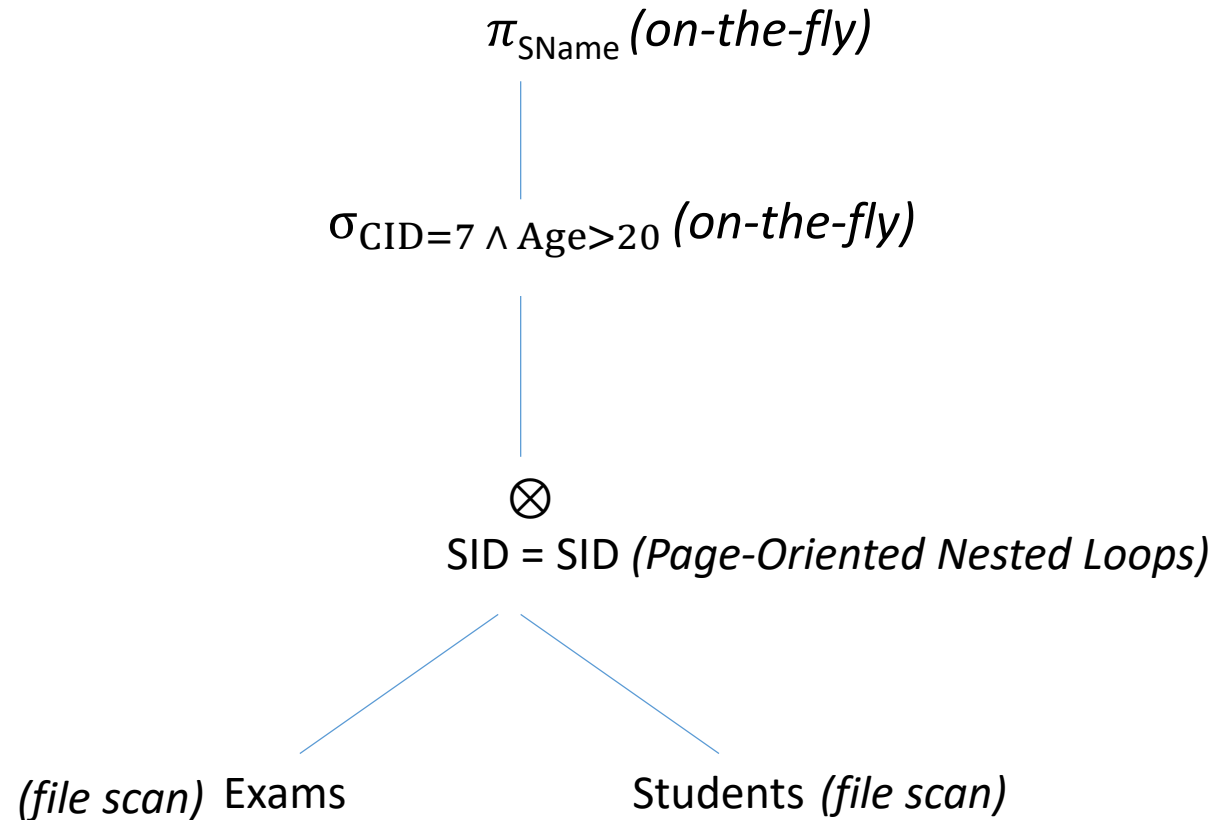
Motivating Example

* E - 1000 pages *

* S - 500 pages *

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
      AND S.Age > 20
```
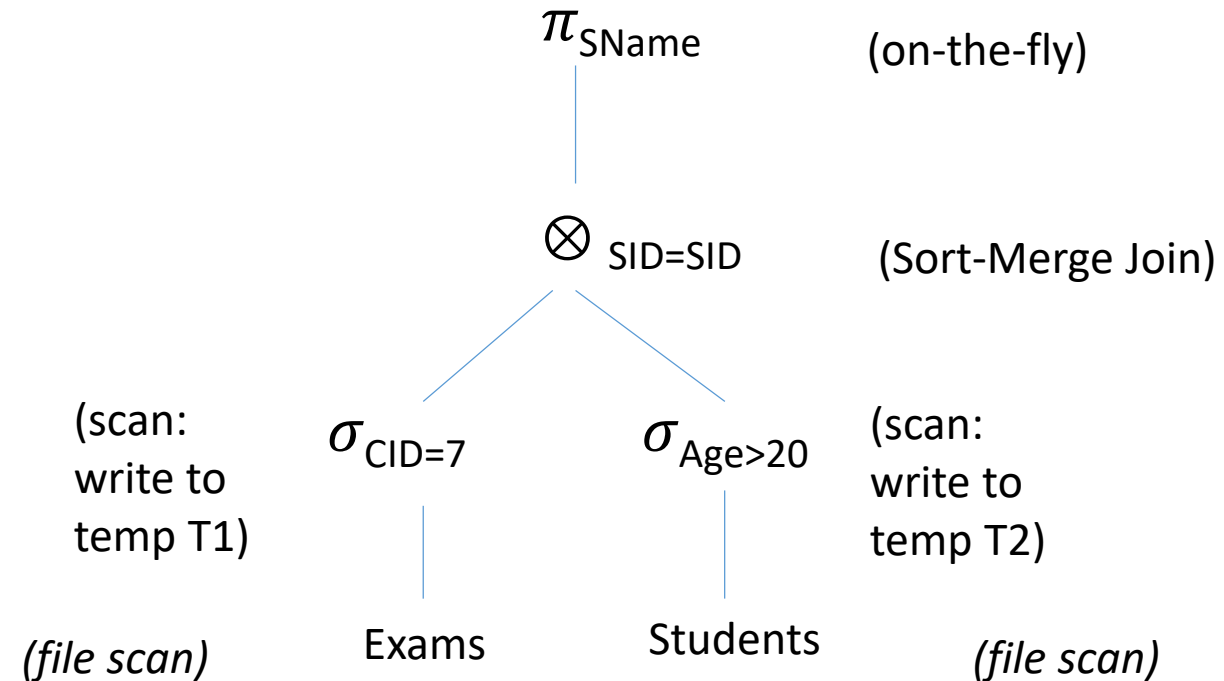
π_{SName} *(on-the-fly)*

$\sigma_{\text{CID}=7 \wedge \text{Age}>20}$ *(on-the-fly)*

$\otimes$
SID = SID *(Page-Oriented Nested Loops)*

(file scan) Exams   Students *(file scan)*

- σ, π – on-the-fly
- cost of plan – very high:
  - 1000 + 1000 * 500 = 501,000 I/Os
- less efficient plan
  - join: cross-product followed by a selection
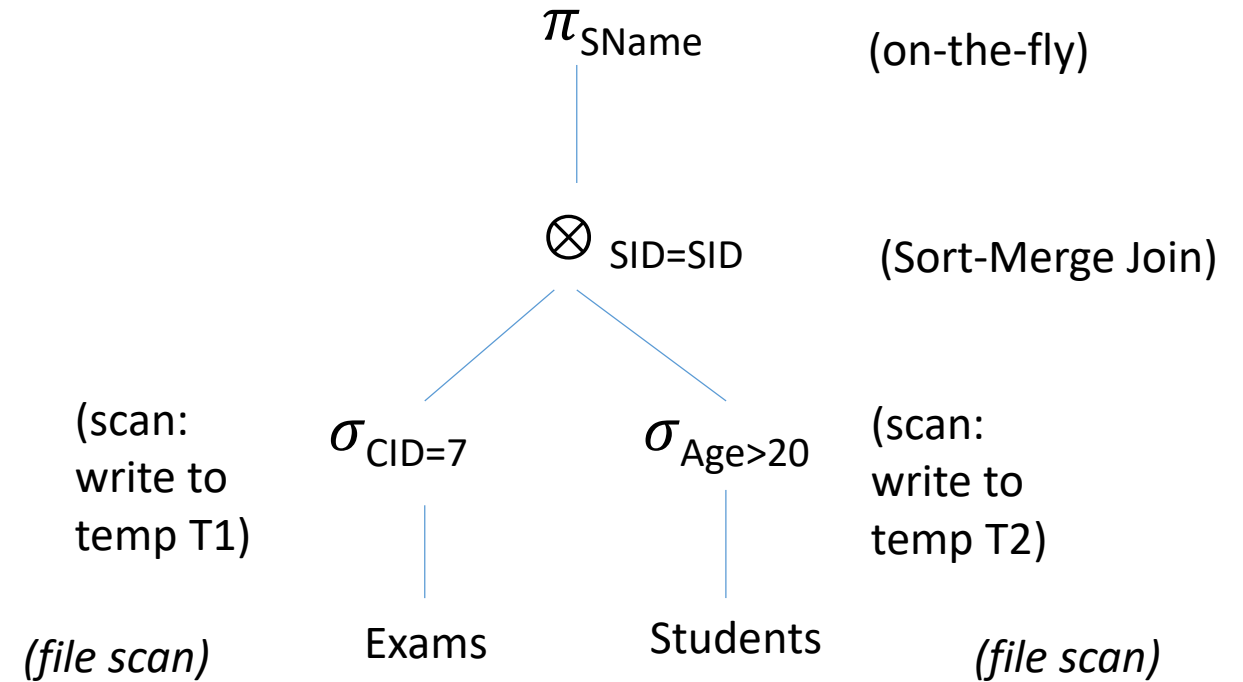
Motivating Example

* optimizations

- reduce sizes of the relations to be joined

  - push selections, projections ahead of the join

- alternative plans

  - push selections ahead of joins

- selection

  - file scan

  - write the result to a temporary relation on disk

- join the temporary relations using Sort-Merge Join

π_{SName} (on-the-fly)

$\otimes$ SID=SID   (Sort-Merge Join)

(scan: write to temp T1)

$\sigma_{CID=7}$ $\sigma_{Age>20}$ (scan: write to temp T2)

(file scan)   Exams   Students   *(file scan)*
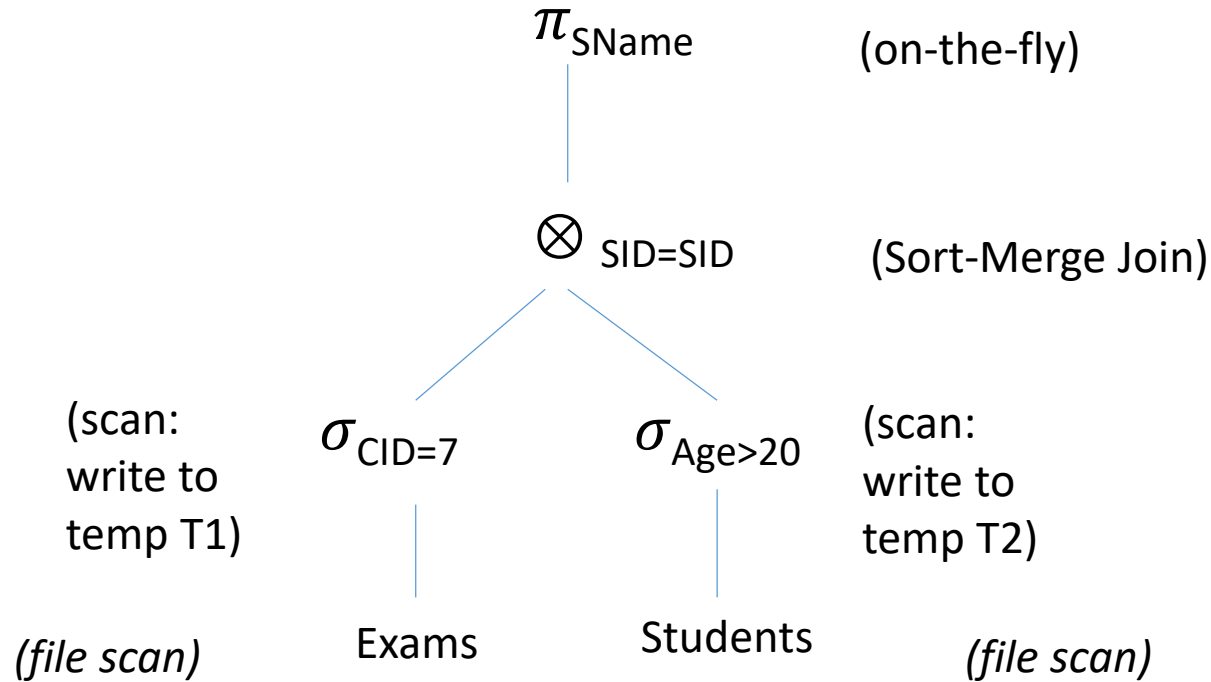
Motivating Example

- 5 available buffer pages

- cost

  - $\sigma_{CID=7}$

    - scan Exams: 1000 I/Os

    - write T1

      - assume exams are uniformly distributed across all courses, i.e., T1 has 10 pages

  - $\sigma_{Age>20}$

    - scan Students: 500 I/Os

    - write T2

      - assume ages are uniformly distributed over the range 19 to 22, i.e., T2 has 250 pages

π_{SName} (on-the-fly)

$\otimes$ SID=SID   (Sort-Merge Join)

(scan: write to temp T1)

$\sigma_{CID=7}$ $\sigma_{Age>20}$ (scan: write to temp T2)

(file scan)   Exams   Students   *(file scan)*

Motivating Example

- 5 available buffer pages

- cost

  - Sort-Merge Join

    - T1 - 10 pages

      - sort T1: 2 * 2 * 10 = 40 I/Os

    - T2 - 250 pages

      - sort T2: 2 * 4 * 250 = 2000 I/Os

    - merge sorted T1 and T2
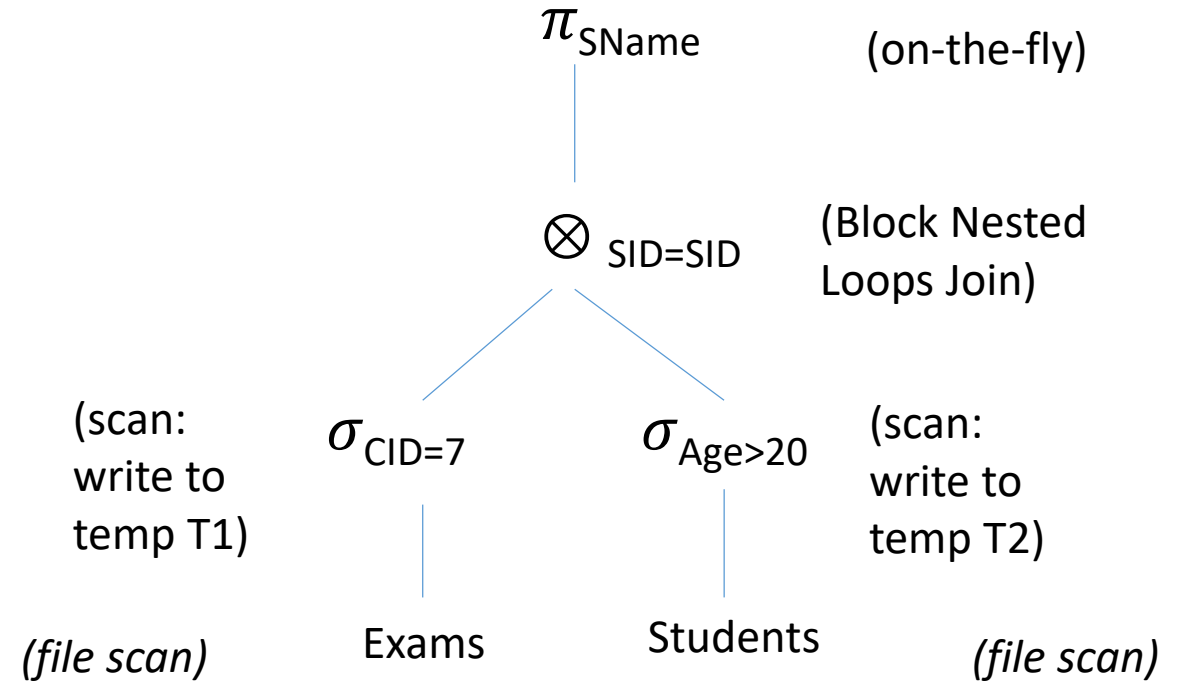
      - 10 + 250 = 260 I/Os

  - π - on the fly

=> **total cost**: $\underline{1000 + 10 + 500 + 250}$ + $\underline{40 + 2000 + 260}$ = **4060 I/Os**
$\qquad\qquad\qquad$ selection $\qquad\qquad\qquad\qquad$ join

π_{SName} $\qquad$ (on-the-fly)

$\otimes$ SID=SID $\qquad$ (Sort-Merge Join)

(scan:
write to
temp T1)

$\sigma_{CID=7}$ $\qquad$ $\sigma_{Age>20}$ $\qquad$ (scan:
write to
temp T2)

(file scan) $\qquad$ Exams $\qquad$ Students $\qquad$ *(file scan)*

Motivating Example

- 5 available buffer pages

- cost

  - Block Nested Loops Join

    - T1 - 10 pages, T2 - 250 pages

    - T1 - outer relation

      => scan T1: 10 I/Os

    - $\lceil 10/3 \rceil = 4$ T1 blocks

      => T2 scanned 4 times: 4 * 250 = 1000 I/Os

    - BNLJ cost: 10 + 1000 = 1010 I/Os

  - π - on the fly

=> **total cost**: 1000 + 10 + 500 + 250 + 10 + 1000 = **2770 I/Os**

selection   join

π_{SName} (on-the-fly)

$\otimes_{SID=SID}$ (Block Nested Loops Join)

(scan: write to temp T1) $\sigma_{CID=7}$ $\sigma_{Age>20}$ (scan: write to temp T2)

(file scan)   Exams   Students   *(file scan)*

Motivating Example

- push projections ahead of joins
  - drop unwanted columns while scanning Exams and Students to evaluate selections => T1[SID], T2[SID, SName]
- T1 fits within 3 buffer pages
  => T2 scanned only once
  => **total cost**: about **2000 I/Os**

Motivating Example

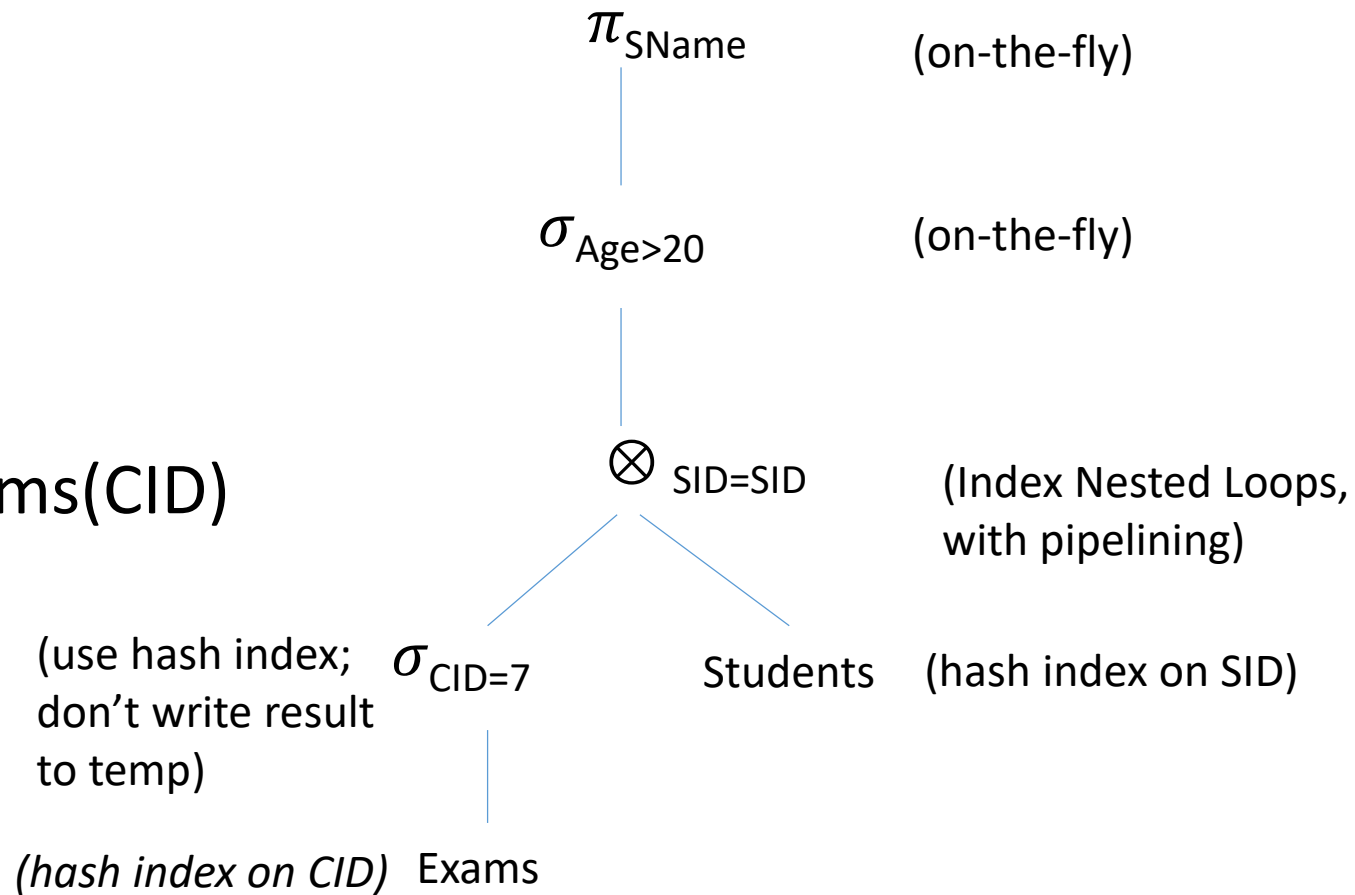* optimizations

- investigate the use of indexes

- clustered static hash index on Exams(CID)

- hash index on Students(SID)
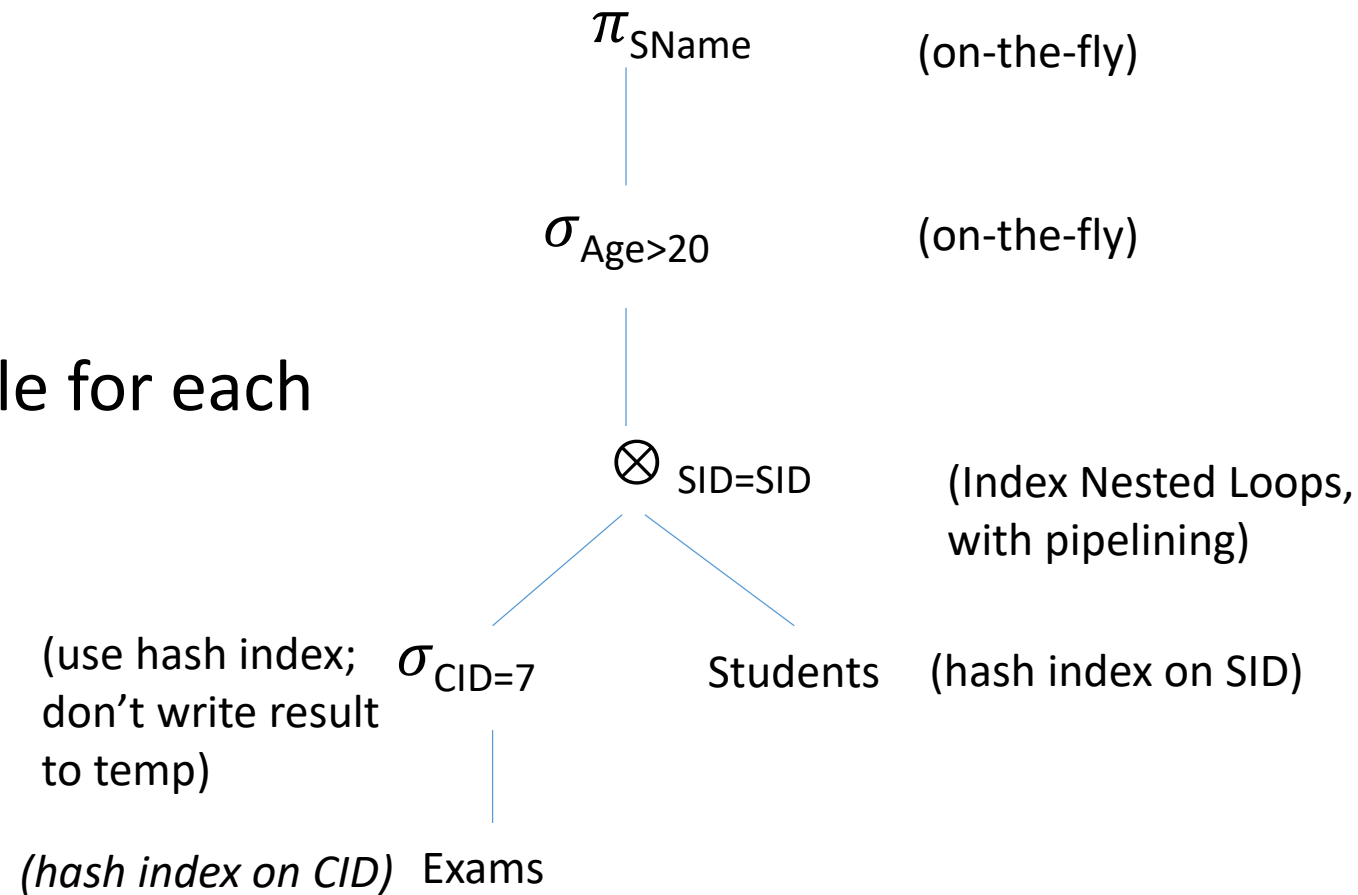
- cost

  - $\sigma_{CID=7}$

    - assume exams are uniformly distributed across all courses => 100,000 exams / 100 courses => 1,000 exams / course

    - clustered index on CID => 1,000 tuples for course with CID=7 appear consecutively within the same bucket => cost: 10 I/Os

    - the result of the selection is not materialized, the join is pipelined

π_{SName} (on-the-fly)

$\sigma_{Age>20}$ (on-the-fly)

$\otimes$ SID=SID   (Index Nested Loops, with pipelining)

(use hash index; don't write result to temp) $\sigma_{CID=7}$ Students   (hash index on SID)

(hash index on CID)   Exams

Motivating Example

- cost

  - Index Nested Loops

    - find matching Students tuple for each selected exam

    - use hash index on SID

      - assume the index uses a1 => cost of 1.2 I/Os (on avg.) per exam

π_{SName} (on-the-fly)

$\sigma_{Age>20}$ (on-the-fly)

$\otimes$ SID=SID   (Index Nested Loops, with pipelining)

(use hash index; don't write result to temp) $\sigma_{CID=7}$ Students   (hash index on SID)

(hash index on CID)   Exams

- σ, π – performed on-the-fly on each tuple in the result of the join

  => **total cost** = $\underline{10}$ + $\underline{1000}$ * $\underline{1.2}$ =   **1210** I/Os

         σ on Exams   num. of Exams tuples   find matching Students tuple (on avg.)

* can we push the selection *Age>20* ahead of the join?

Estimating the Cost of a Plan

* estimating the cost of an evaluation plan for a query block

- for each node N in the tree:
    - estimate the cost of the corresponding operation (pipelining versus temporary relations)
    - estimate the size of N's result and whether it is sorted
        - N's result is the input of N's parent node
        - these estimates affect the estimation of cost, size, and sort order for N's parent

Estimating the Cost of a Plan

- estimating costs
  - use data about the input relations (such statistics are stored in the DBMS's system catalogs)
    - number of pages, existing indexes, etc.
- obtained estimates are at best approximations to actual sizes and costs

=> one shouldn't expect the optimizer to find the best possible plan

- optimizer - goals:
  - avoid the worst plans
  - find a good plan

Statistics Maintained by the DBMS
- updated periodically, not every time the data is changed
  - relation R
    - cardinality - *NTuples(R)*
      - the number of tuples in R
    - size - *NPages(R)*
      - the number of pages in R
  - index I
    - cardinality - *NKeys(I)*
      - the number of distinct key values for I
    - size - *INPages(I)*
      - the number of pages for I
      - B+ tree index
        - number of leaf pages
    - height - *IHeight(I)*
      - maintained for tree indexes
      - the number of nonleaf levels in I
    - range - *ILow(I), IHigh(I)*
      - the minimum / maximum key value in I

Estimating Result Sizes

- query Q

  ```
  SELECT attribute list
  FROM relation list
  WHERE term₁ ∧ ... ∧ termₖ
  ```

- the maximum number of tuples in Q's result:
  - $\prod |R_i|, R_i \in$ `relation list`
- every $term_j$ in the WHERE clause eliminates some candidate tuples
  - associate a reduction factor RF_j with each term $term_j$
  - RF_j models the impact $term_j$ has on the result size
- estimate the actual size of the result: $\prod |R_i| * \prod RF_j$
  - i.e., the maximum result size times the product of the reduction factors for the terms in the WHERE clause
- assumption: the conditions tested by the terms in the WHERE clause are statistically independent

Estimating Result Sizes

- compute reduction factors for terms in the WHERE clause
- assumptions:
  - uniform distribution of values
  - independent distribution of values in different columns
  ```
  SELECT attribute list
  FROM relation list
  WHERE term₁ AND ... AND termₖ
  ```
  - *column = value*
    - index I on *column* => RF approximated by 1/NKeys(I)
    - no index on *column* => RF: 1/10
      - maintain statistics on *column* (e.g., number of distinct values in *column*) to obtain a better value
  - *column1 = column2*
    - indexes *I1* on *column1*, *I2* on *column2*
      => RF: 1/MAX(NKeys(I1), NKeys(I2))

Estimating Result Sizes

- only one index I (on one of the 2 columns) => RF: $1/NKeys(I)$
- no indexes => RF: $1/10$
- *column > value*
  - index I on *column* => RF: $(IHigh(I) - value) / (IHigh(I) - ILow(I))$
  - no index on *column* or *column* not of an arithmetic type
    => a value less than 0.5 is arbitrarily chosen
  - similar formulas can be obtained for other range selections
- *column IN (list of values)*
  => RF: (RF for *column = value*) * number of items in list (but at most 0.5)
- *NOT condition*
  => RF: 1 - RF for *condition*
- obtain better estimates
  - use more detailed statistics (e.g., histograms of the values in a column)

Relational Algebra Equivalences
- central role in generating alternative plans
- different join orders can be considered
- selections, projections can be pushed ahead of joins
- cross-products can be converted to joins

- selections
  - *cascading selections*
    - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R)))$
  - *commutativity*
    - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
- projections
  - *cascading projections*
    - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$
    - a_i – set of attributes in R
    - $a_i \subseteq a_{i+1}$, for i = 1..n-1

Relational Algebra Equivalences
- joins and cross-products
  - assumption
    - fields are identified by their name, not by their position
  - *associativity*
    - R × (S × T) ≡ (R × S) × T
    - R * (S * T) ≡ (R * S) * T
  - *commutativity*
    - R × S ≡ S × R
    - R * S ≡ S * R
    - can choose the inner / outer relation in a join
- e.g., check that R * (S * T) ≡ (T * R) * S
  - commutativity
    - R * (S * T) ≡ R * (T * S)
  - associativity
    - R * (T * S) ≡ (R * T) * S
  - commutativity
    - (R * T) * S ≡ (T * R) * S

Relational Algebra Equivalences
- can commute σ with π if σ uses only attributes retained by π
  - $\pi_a(\sigma_c(R)) \equiv \sigma_c(\pi_a(R))$
- can combine σ with $\times$ to form a join
  - $R \otimes_c S \equiv \sigma_c(R \times S)$
- can commute σ with $\times$ or a join when the selection condition includes only fields of one of the arguments (to the cross-product or join)
  - for instance:
    - $\sigma_c(R * S) \equiv \sigma_c(R) * S$
    - $\sigma_c(R \times S) \equiv \sigma_c(R) \times S$
      - condition c must include only fields from R
- in general: $\sigma_c(R \times S) \equiv \sigma_{c1}(\sigma_{c2}(R) \times \sigma_{c3}(S))$
  - c1 – attributes of both R and S
  - c2 – only attributes of R
  - c3 – only attributes of S

Relational Algebra Equivalences
- can commute π with $\times$
  - $\pi_a(R \times S) \equiv \pi_{a1}(R) \times \pi_{a2}(S)$
  - a1 – attributes in a that appear in R
  - a2 – attributes in a that appear in S
- can commute π with join
  - $\pi_a(R \otimes_c S) \equiv \pi_{a1}(R) \otimes_c \pi_{a2}(S)$
    - every attribute in c must appear in a
    - a1 – attributes in a that appear in R
    - a2 – attributes in a that appear in S
  - a doesn't contain all the attributes in c – generalization
    - eliminate unwanted fields, compute join, eliminate fields not in a
      - $\pi_a(R \otimes_c S) \equiv \pi_a(\pi_{a1}(R) \otimes_c \pi_{a2}(S))$
      - a1 – attributes of R that appear in either a or c
      - a2 – attributes of S that appear in either a or c

Enumeration of Alternative Plans

- query Q
  - consider a certain set of plans
  - choose the plan with the least estimated cost
    - algebraic equivalences
    - implementations techniques for Q's operators
- not all algebraically equivalent plans are enumerated (optimization costs would be too high)
- two main cases:
  - queries with one relation in the FROM clause
  - queries with two or more relations in the FROM clause

Enumeration of Alternative Plans

- queries with one relation in the FROM clause (i.e., no joins; only σ, π, grouping, aggregate operations):
  - if there is only one σ or π or aggregate operation: consider implementation techniques and cost estimates discussed in previous lectures
  - if there is a combination of several σ, π, aggregate operations:
    - plans with / without indexes
  - example query:

```
SELECT S.RoundedGPA, COUNT(*)
FROM Students S
WHERE S.RoundedGPA > 5 AND S.Age = 20
GROUP BY S.RoundedGPA
HAVING COUNT(DISTINCT S.SName) > 5
```

$\pi_{S.RoundedGPA, COUNT(*)} ($
$HAVING_{COUNT\ DISTINCT\ (S.SName) > 5} ($
$GROUP\ BY_{S.RoundedGPA} ($
$\pi_{S.RoundedGPA, S.SName} ($
$\sigma_{S.RoundedGPA > 5 \land S.Age = 20} ($
$Students)))))$

Enumeration of Alternative Plans

* plans without indexes:
- apply σ, π while scanning Students
  - file scan: NPages(Students):   500 I/Os
  - write out tuples to a temporary relation T:
    - NPages(Students) * RF(RoundedGPA > 5) * RF(Age = 20) * (size of a *pair <RoundedGPA, SName>* / size of a *Students tuple*)
    - RF for *RoundedGPA > 5*:   0.5
    - RF for *Age = 20*:   0.1
    - size of <RoundedGPA, SName>:   about 0.8 * size of a Students tuple
    => 500 * 0.5 * 0.1 * 0.8 = 20 I/Os (temporary relation T)
- GROUP BY:
  - sort T in 2 passes: 4 * 20 = 80 I/Os
- HAVING, aggregations: no additional I/O
- **total cost:** 500 + 20 + 80 = **600 I/Os**

Enumeration of Alternative Plans

* plans that use an index:

- available indexes on Students - a2
  - hash index on <Age>
  - B+ tree index on <RoundedGPA>
  - B+ tree index on <RoundedGPA, SName, Age>
- single-index access path:
  - choose the index that provides the most selective access path
  - apply π, nonprimary selection terms
  - compute grouping and aggregation operations
  - example:
    - use the hash index on Age to retrieve Students with Age = 20
      - cost: retrieve index entries and corresponding Students tuples
    - apply condition *RoundedGPA > 5* to each retrieved tuple
    - retain RoundedGPA and SName

Enumeration of Alternative Plans
* plans that use an index:
- single-index access path – example:
  - write out tuples to a temporary relation
  - sort the temporary relation by RoundedGPA to identify groups
  - apply the HAVING condition (to eliminate some groups)
- multiple-index access path:
  - several indexes using a2 / a3 match the selection condition, e.g., I1, I2
  - retrieve $Rids_{I1}$, $Rids_{I2}$ using I1, I2
  - get tuples with rids in $Rids_{I1}$ ∩ $Rids_{I2}$ (tuples satisfying the primary selection terms of I1 and I2)
  - apply π, nonprimary selection terms
  - compute grouping and aggregation operations
  - example:
    - use the index on Age => rids of tuples with Age = 20 (R1)

Enumeration of Alternative Plans

* plans that use an index:

- <u>multiple-index access path</u> – example:
    - index on RoundedGPA => rids of tuples with RoundedGPA > 5 (R2)
    - retrieve tuples with rids in R1 ∩ R2
    - keep only RoundedGPA and SName
    - write out tuples to a temporary relation
    - sort the temporary relation by RoundedGPA to identify groups
    - apply the HAVING condition (to eliminate some groups)
- <u>sorted index access path:</u>
    - works well when the index is clustered
    - B+ tree index I with search key K
    - GROUP BY attributes – prefix of K
    - use the index to retrieve tuples in the order required by the GROUP BY clause

Enumeration of Alternative Plans
* plans that use an index:
- apply σ, π
- compute aggregation operations
- <u>sorted index access path</u> – example:
  - use the B+ tree index on RoundedGPA to retrieve Students tuples with RoundedGPA > 5, ordered by RoundedGPA
  - aggregations in HAVING, SELECT - computed on-the-fly
- <u>index-only access path:</u>
  - dense index I with search key K
  - all the attributes in the query are included in K
  => index-only scan, don't need to retrieve tuples from the relation
  - data entries: apply σ, perform π, sort the result (to identify groups), compute aggregate operations
    * obs. index I doesn't need to match the selections in the WHERE clause

Enumeration of Alternative Plans
* plans that use an index:
- <u>index-only access path</u>
  * obs. I – tree index, GROUP BY attributes – prefix of K
  => can avoid sorting
  - example:
    - use the B+ tree index on <RoundedGPA, SName, Age> to retrieve entries with RoundedGPA > 5, ordered by RoundedGPA
    - select entries with Age = 20
   - aggregation operations in the HAVING and SELECT clauses - computed on-the-fly

Enumeration of Alternative Plans
- queries with several relations in the FROM clause:
  - joins, cross-products => queries can be quite expensive
  - different join orders => intermediate relations of widely varying sizes => plans with very different costs

- class of plans considered by the optimizer
- plan enumeration

Enumeration of Alternative Plans
- queries with several relations in the FROM clause

A * B * C * D

left-deep tree

outer relation / A   B / inner relation

linear trees
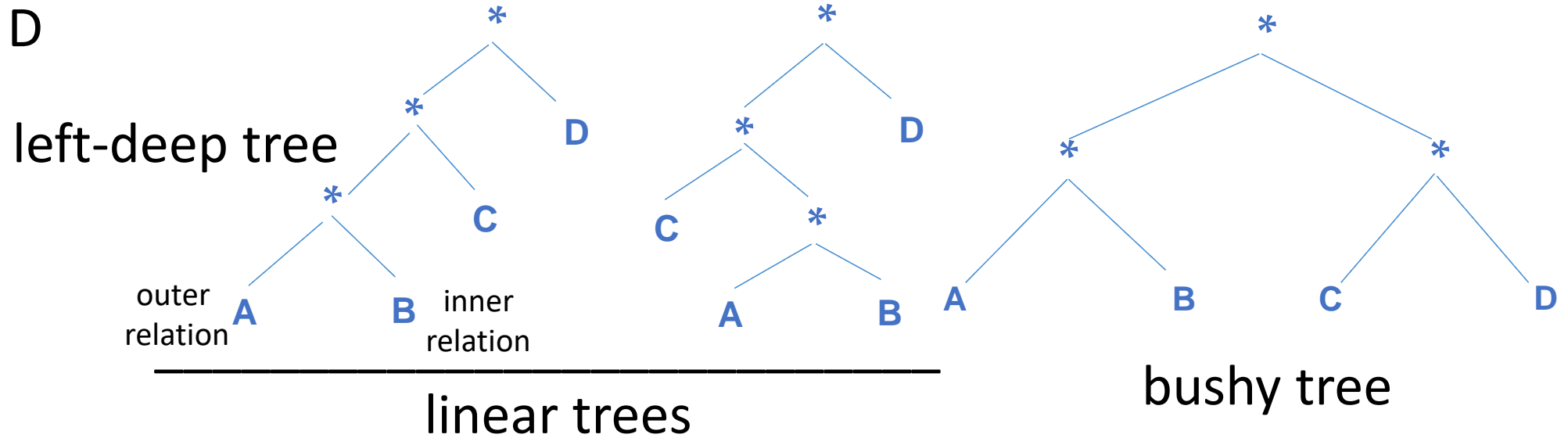
bushy tree

- linear trees:
  - at least one child of a join node is a base relation
- left-deep trees:
  - the right child of each join node is a base relation
- bushy tree: not linear

Enumeration of Alternative Plans
- queries with several relations in the FROM clause
- fundamental decision in System R:
    - only *left-deep trees* are considered
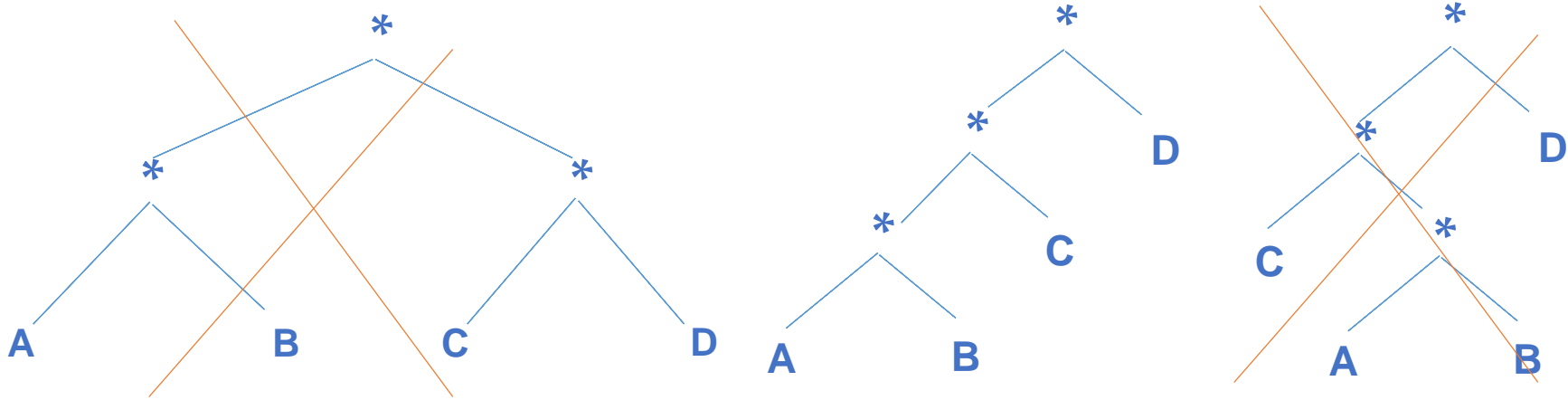
- motivation:
    - number of joins increases => number of alternative plans increases quickly => must prune the search space
    - left-deep trees generate all fully pipelined plans (all joins are evaluated using pipelining)

Enumeration of Alternative Plans

- queries with several relations in the FROM clause

  * obs: not all plans based on left-deep trees are fully pipelined

  - e.g., sort-merge join: outer tuples may have to be retrieved in a particular order, so the outer relation should be materialized

- multiple passes:

  - pass 1:
    - find best 1-relation plan for each relation

  - pass 2:
    - find best way to join the result of each 1-relation plan (as the outer argument) with another relation (all 2-relation plans)

    ...

  - repeat until the obtained plans contain all the relations in the query

- for each subset of relations, retain: the cheapest overall plan & the cheapest plan for each interesting ordering of tuples

Enumeration of Alternative Plans

- queries with several relations in the FROM clause
    - GROUP BY, aggregates are handled as a final step:
        - use a plan with an interesting ordering of tuples
        - use an additional sorting operator
- obs. avoid cross-products if possible

Example

- unclustered indexes using a2
- Students: B+ tree index on Age, hash index on SID
- Exams: B+ tree index on CID
- pass 1:
  - Students – 3 possible access paths:
    B+ tree index, hash index, file scan
    - *Age > 20* matches the B+ tree index on Age
    - hash index / file scan => probably much higher cost
    - keep the plan using the B+ tree index => retrieved tuples are ordered by Age
  - Exams – 2 possible access paths
    - *CID = 7* matches the B+ tree index on CID
    - better than file scan => keep the plan using the B+ tree index
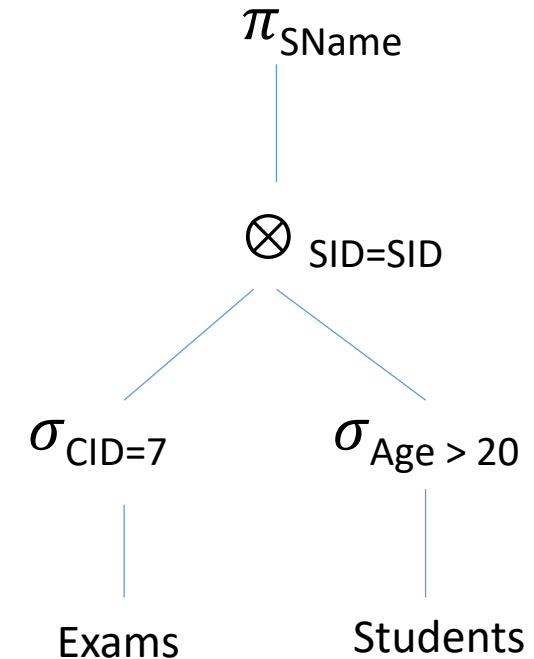
π_{SName}

$\otimes$ SID=SID

$\sigma_{CID=7}$ $\sigma_{Age > 20}$

Exams   Students

<u>Example</u>

- Students: B+ tree index on Age, hash index on SID
- Exams: B+ tree index on CID
- <u>pass 2</u>:
  - consider (the result of) each plan from pass 1 as the outer argument and analyze how to join it with the other relation
  - e.g., Exams – outer argument
    - examine alternative access methods / join methods
    - access methods:
      - need Students tuples s.t. SID = *value* from outer tuple and Age > 20
      - hash index => Students tuples s.t. SID = *value* from outer tuple
      - B+ tree index => Students tuples s.t. Age > 20
    - join methods:
      - consider all available methods
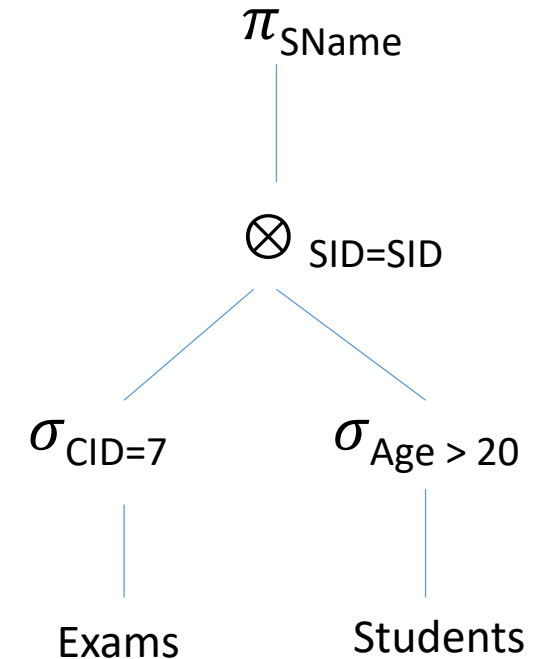
π_{SName}

$\otimes$ SID=SID

$\sigma_{CID=7}$ $\sigma_{Age > 20}$

Exams   Students

Example

- Students: B+ tree index on Age, hash index on SID
- Exams: B+ tree index on CID
- pass 2:
  - consider (the result of) each plan from pass 1 as the outer argument and analyze how to join it with the other relation
  - e.g., Students – outer argument
    - access / join methods
    - access methods:
      - need Exams tuples s.t. SID = *value* from outer tuple and CID = 7
    - join methods:
      - consider all available methods
  - retain cheapest plan overall!

Nested Queries - optional
- usually handled using some form of nested loops evaluation
- correlated query - typical evaluation strategy:
  - the inner block is evaluated for each tuple of the outer block
- some strategies are not considered
  - e.g., index on SID in Students
  - best plan could be INLJ with Exams as the outer argument, and Students as the inner one; such a plan is never considered by the optimizer
- the unnested version of the query is typically optimized better

```
SELECT  S.SName
FROM  Students S
WHERE EXISTS
   (SELECT  *
    FROM  Exams E
    WHERE  E.CID=7
     AND E.SID=S.SID)
```

Equivalent unnested query:
```
SELECT  S.SName
FROM Students S, Exams E
WHERE E.SID=S.SID AND E.CID = 7
```