CS3223 Lecture 6
Query Evaluation & Optimization

Set Operations

- Set operations
  - Cross-product: $R \times S$
  - Intersection: $R \cap S$
  - Union: $R \cup S$
  - Difference: $R - S$
- Intersection & Cross-product: special cases of join
  - $R(A,B) \cap S(A,B) = \pi_{R.A,R.B}(R \bowtie_p S)$
    - ⋆ $p = (R.A = S.A) \wedge (R.B = S.B)$
  - $R \times S = R \bowtie_{true} S$
- Union & Difference
  - Sorting-based approach
  - Hashing-based approach

Set Operations (cont.)

- Sorting approach for $R \cup S$:
  - Sort R using all attributes
  - Sort S using all attributes
  - Merge the sorted operands to combine them and discard duplicates
- Hashing approach for $R \cup S$:
  - Partition R into $\{R_1, \cdots, R_k\}$ using hash function h on all attributes
  - Partition S into $\{S_1, \cdots, S_k\}$ using hash function h on all attributes
  - For i = 1 to k
    - ★ Build an in-memory hash table T_i (using hash function h') for S_i
    - ★ Scan R_i: for each tuple $t \in R_i$, probe T_i. Discard t if t is in T_i; otherwise, insert t into T_i
    - ★ Write T_i to disk
- Algorithms for $R - S$ are similar to those for $R \cup S$

Aggregation Operation: Simple Aggregation

- Example:

  select **count**(*)
  from Movies

- Algorithm: Maintain some running information while scanning table

Aggregate Operator	Running Information
SUM	total of retrieved values
COUNT	count of retrieved values
AVG	(total, count) of retrieved values
MIN	smallest retrieved value
MAX	largest retrieved value

Aggregation Operation: Group-by Aggregation

- Example:

  select year, **count**(*)
  from Movies
  group by year

- Sorting Approach:
  - Sort relation on grouping attribute(s)
  - Scan sorted relation to compute aggregate for each group
- Hashing Approach:
  - Scan relation to build a hash table on grouping attribute(s)
  - For each group, maintain (grouping-value, running-information)

Aggregation Operation: Using Index

- Avoid table scan: If there's a covering index for the query, the aggregation operation can be computed from the index's data entries instead of data records
- **Example**: Suppose we have a B^+-tree index I on $(director, year)$. The following query can be evaluated with a sorting/hashing approach using an index scan of I

```
select    year, count(*)
from      Movies
group by  year
```

Aggregation Operation: Using Index (cont.)

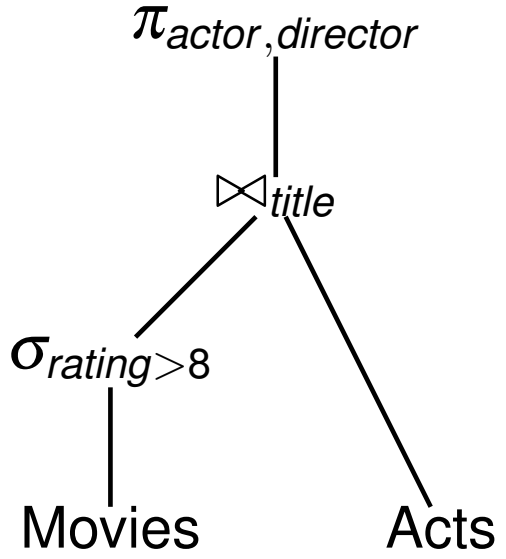
- Ordered scan using index: For group-by aggregation, if set of the group-by attributes forms a prefix of a B^+-tree index's search key, the data entries (and data records if necessary) for each group can be retrieved without an explicit sorting
- **Example**: Suppose we have a B^+-tree index I on $(year, director)$. The following query can be evaluated with an index scan of I

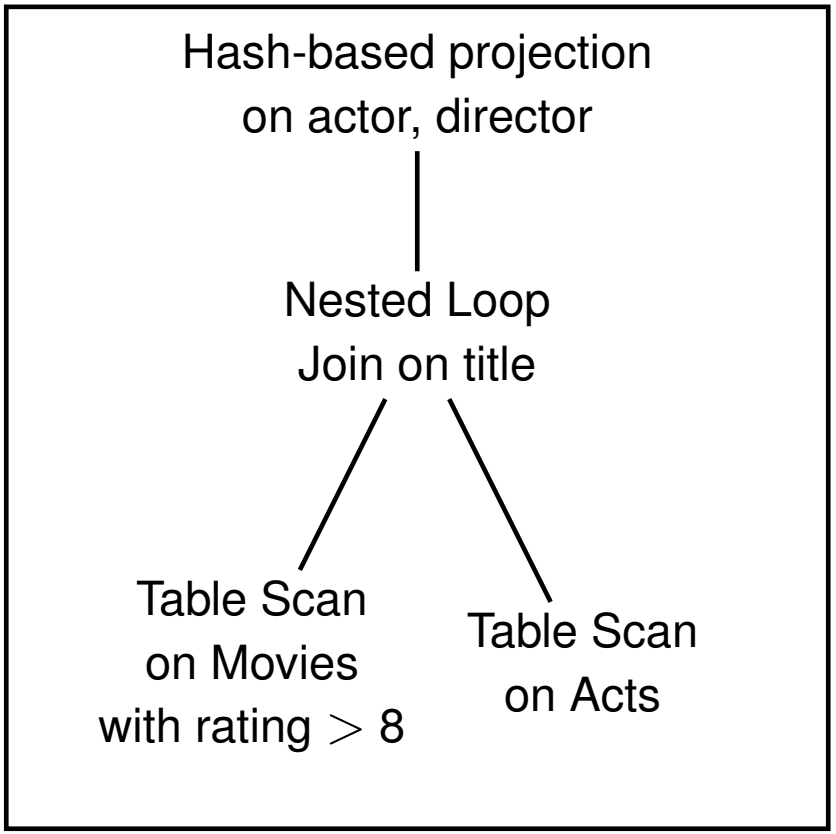
```
select    year, count(*)
from      Movies
group by  year
```

Query Evaluation

Movies (title, director, year, rating)
Acts (title, actor, role)

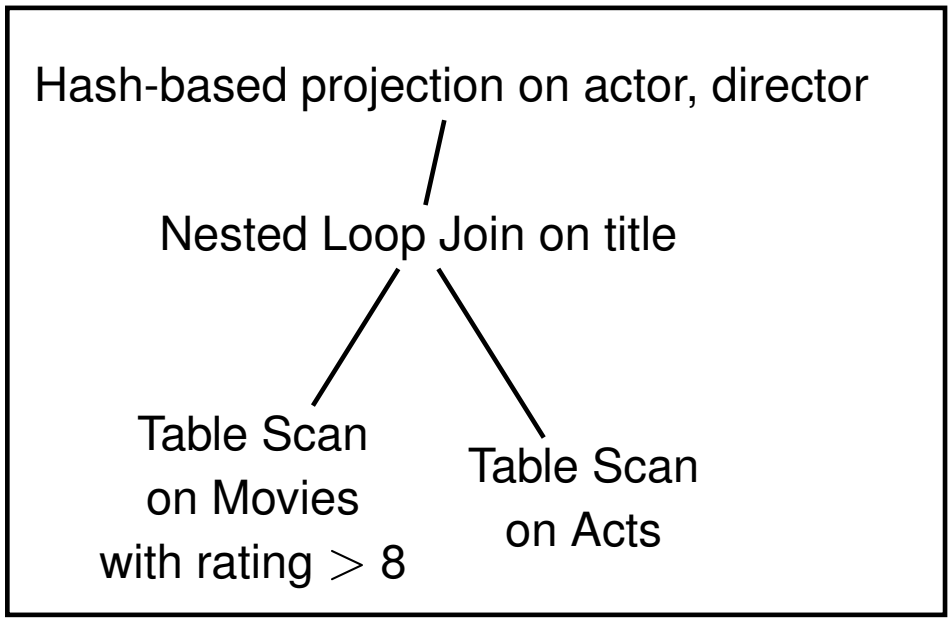
```
SELECT  actor, director
FROM    Movies, Acts
WHERE   Movies.title = Acts.title
AND     Movies.rating > 8
```



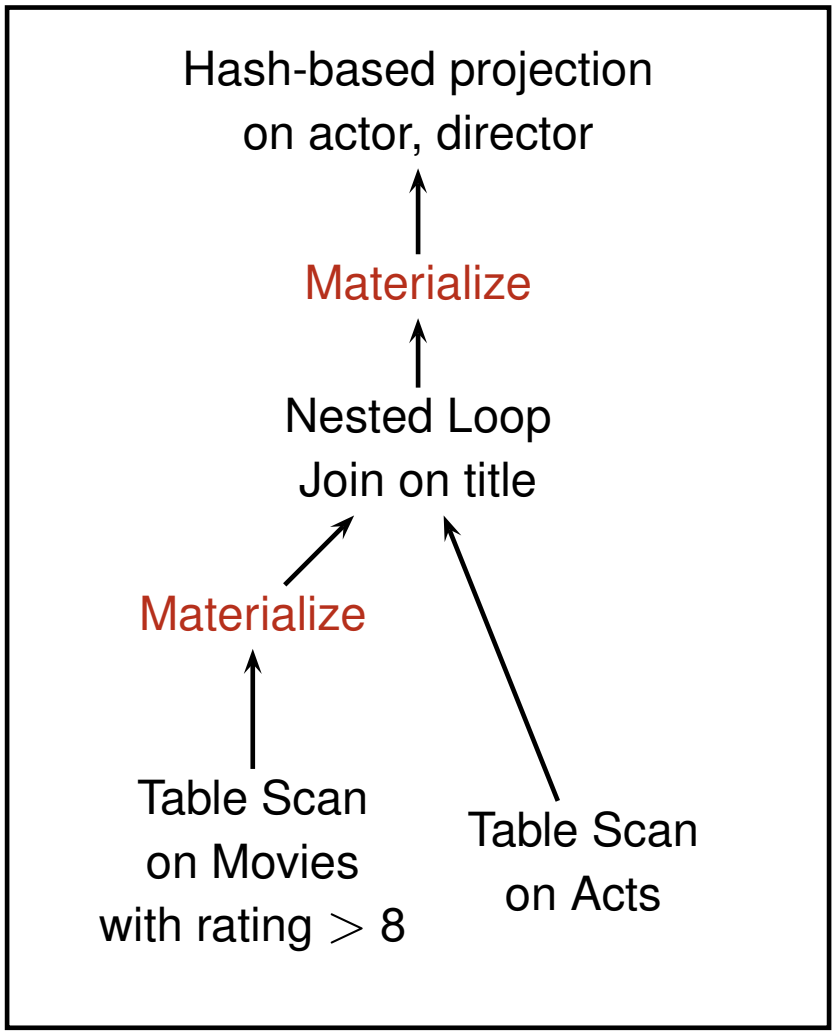


Query Evaluation Approaches

- **Materialized evaluation**
  - An operator is evaluated only when each of its operands has been completely evaluated or materialized.
  - Intermediate results are materialized to disk
- **Pipelined evaluation**
  - The output produced by a operator is passed directly to its parent operator
  - Execution of operators is interleaved



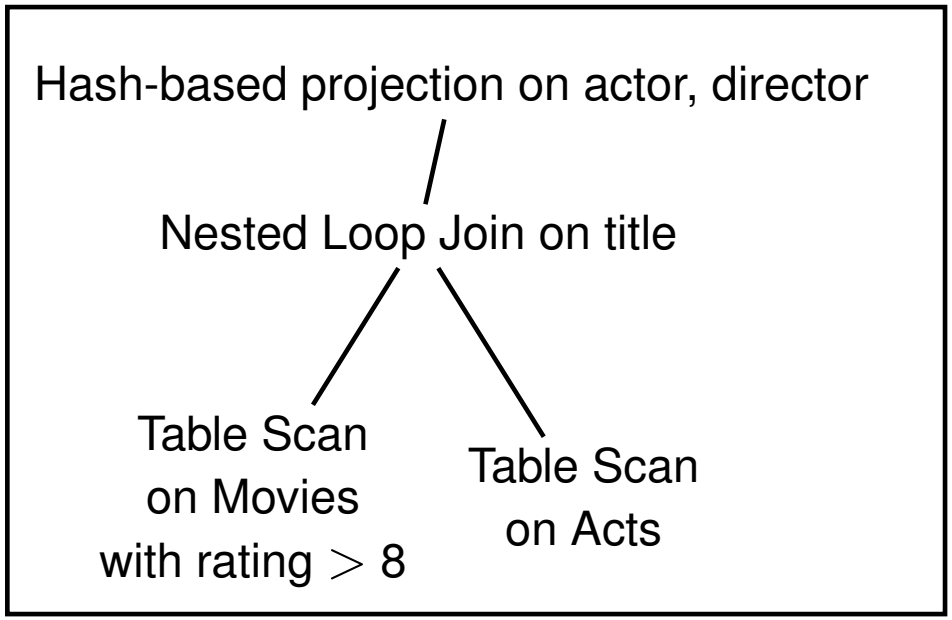
Materialized Evaluation



1. temp1 = Table scan on Movies with rating $>$ 8
2. temp2 = Nested loop join of temp1 and Acts on title
3. result = Hash-based projection of temp2 on actor, director

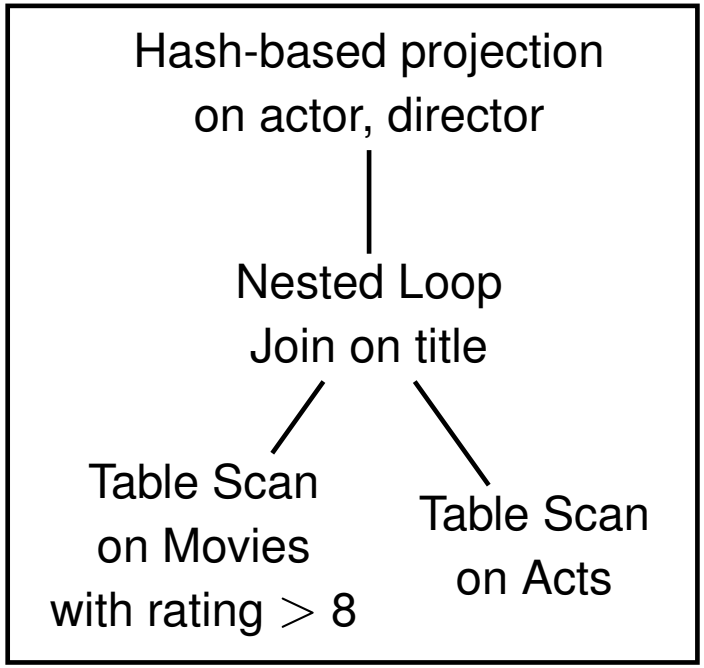
Pipelined Evaluation

- The output produced by a operator is passed directly to its parent operator
- Execution of operators is interleaved
- An operator O is a blocking operator if O may not be able to produce any output until it has received all the input tuples from its child operator(s)
  - Examples of blocking operator: external merge sort, sort-merge join, Grace hash join



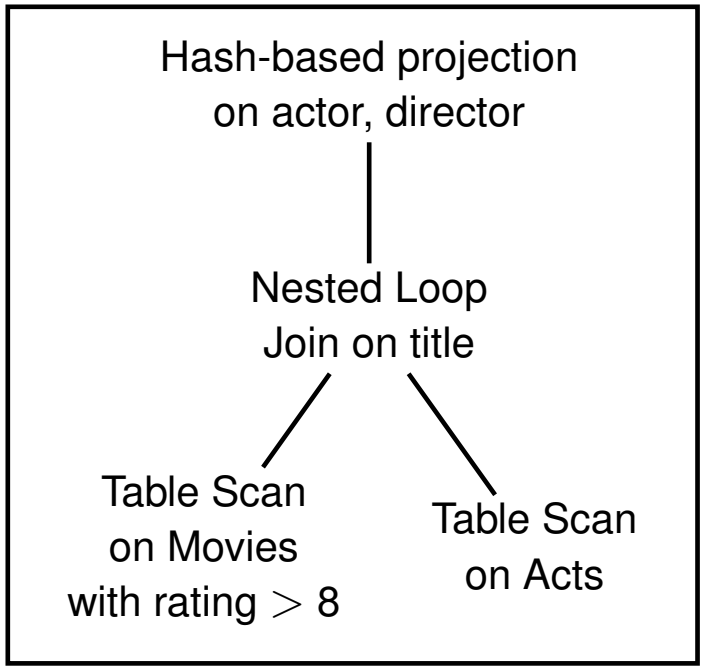
Pipelined Evaluation: Iterator Interface

- Top-down, demand-driven approach
- Iterator interface for operator evaluation
  1. open
     - ⋆ initializes state of iterator: allocates resources for operation, initializes operator's arguments (e.g., selection conditions)
  2. getNext: generates next output tuple
     - ⋆ Returns null if all output tuples have been generated
  3. close: deallocates state information



Example 1: Hash-based Projection (with operand R)

open()
Allocate memory for operation
Initialize hash table HT
R.open(); r $\leftarrow$ R.getNext()
while (r $\neq$ null)
    Hash on r & write r to appropriate output page
    r $\leftarrow$ R.getNext()
Let R be partitioned into $R_1, \cdots, R_n$
Initialize i = 1



getNext()
while ($i \leq n$)
    Initialize hash table HT for partition R_i
    Hash each tuple r from R_i
    Insert r into HT if it's not a duplicate
    while (HT is not empty)
        return next tuple from HT
    i = i + 1
return null

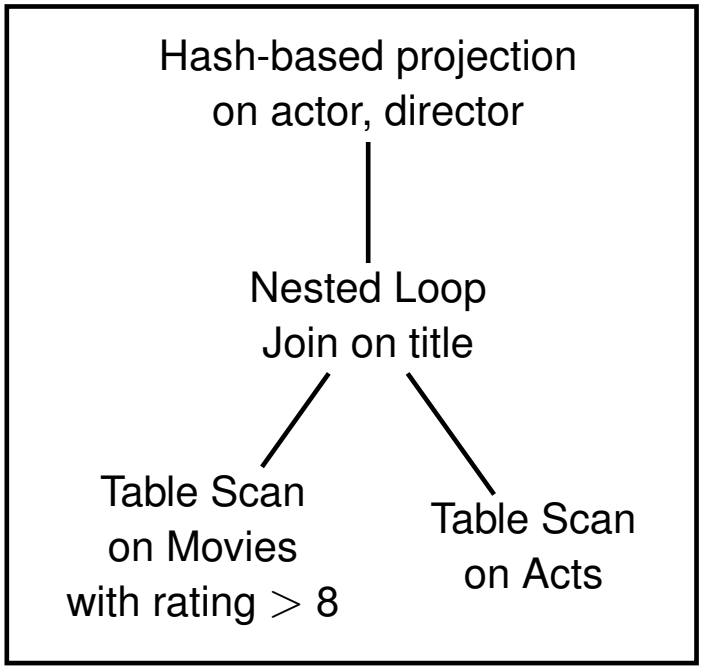
close()
R.close()
Deallocate memory

Example 2: Nested Loop Join (with operands L & R)

open()
Allocate memory for operation
L.open()
R.open()

close()
L.close()
R.close()
Deallocate memory

getNext()
$\ell \leftarrow$ L.getNext()
while ($\ell \neq$ null)
    r $\leftarrow$ R.getNext()
    while (r $\neq$ null)
        if (ℓ & r satisfy p) then return (ℓ, r)
        r $\leftarrow$ R.getNext()
    R.rewind()
    $\ell \leftarrow$ L.getNext()
return null

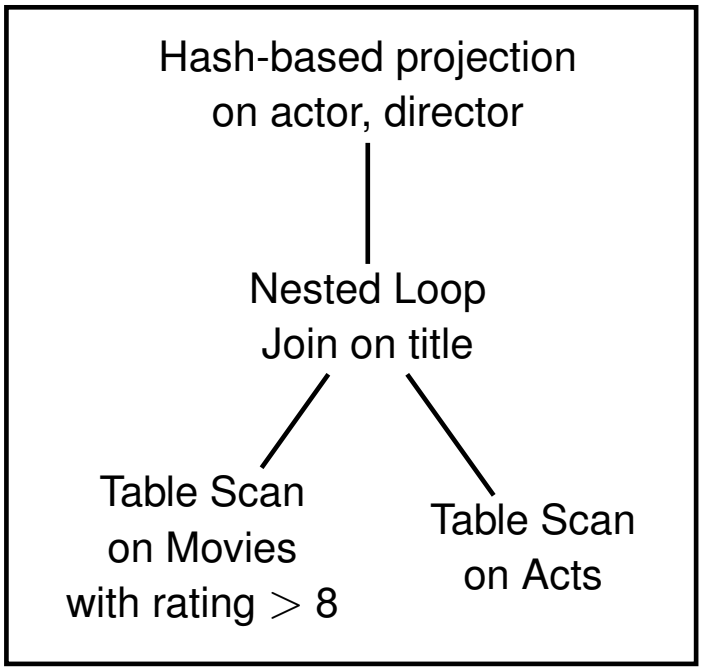


Example 3: Table Scan (with operand R & predicate p)

open()
R.open()

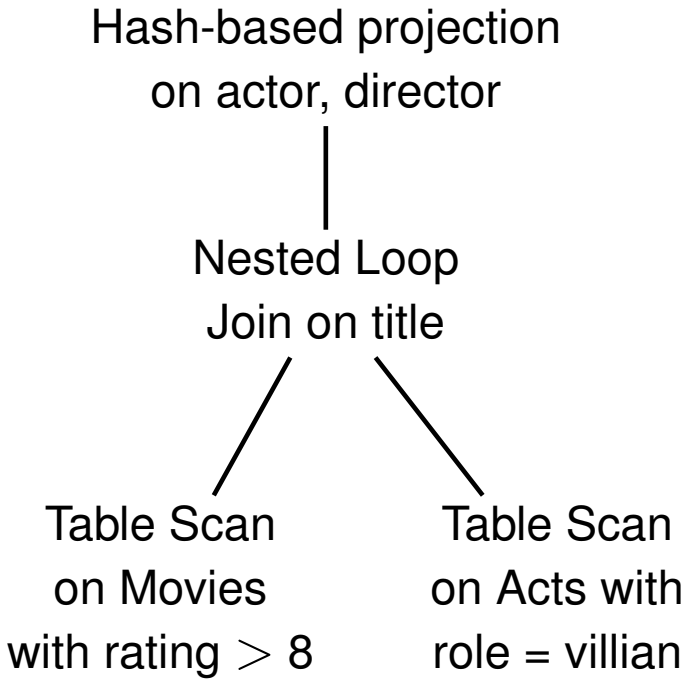
getNext()
r $\leftarrow$ R.getNext()
while (r $\neq$ null)
    if (r satisfies p) then
        return r
    r $\leftarrow$ R.getNext()
return null

close()
R.close()

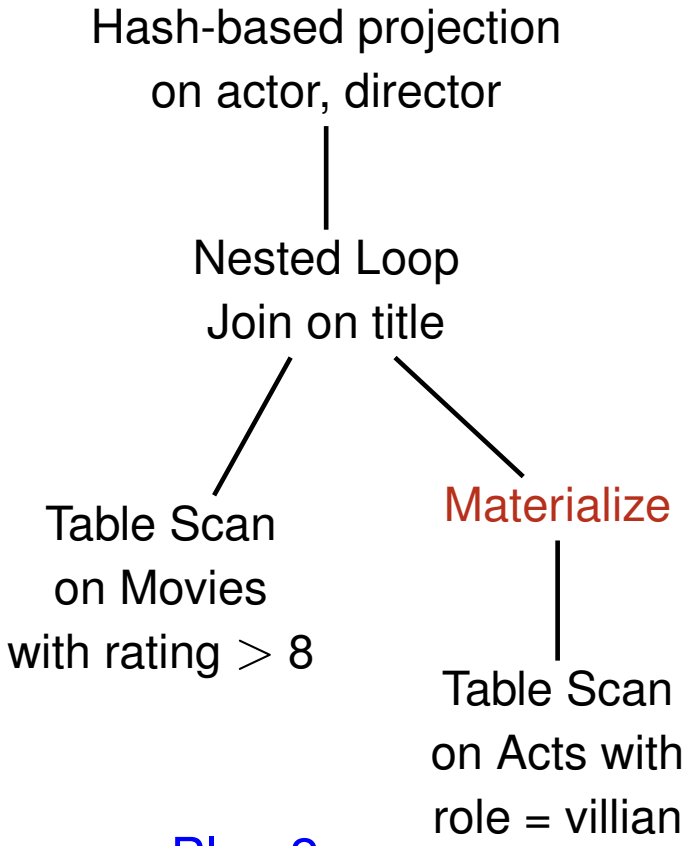


Pipelined Evaluation with Partial Materialization

```
SELECT actor, director FROM Movies, Acts
WHERE  Movies.title = Acts.title
AND    (rating > 8) AND (role = villian)
```

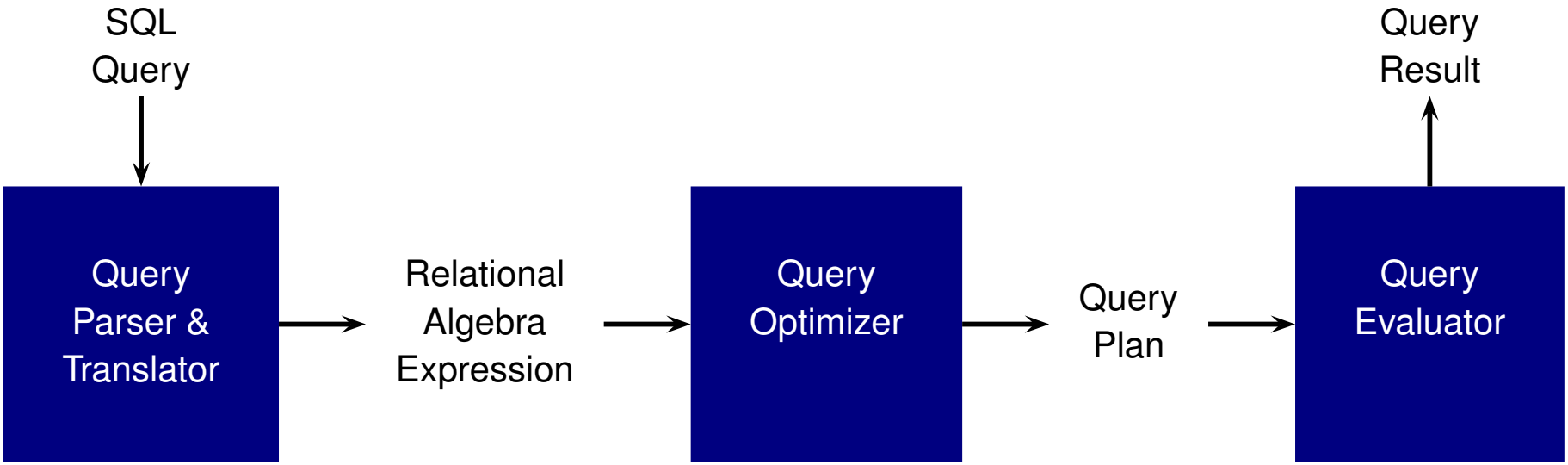


Plan 1



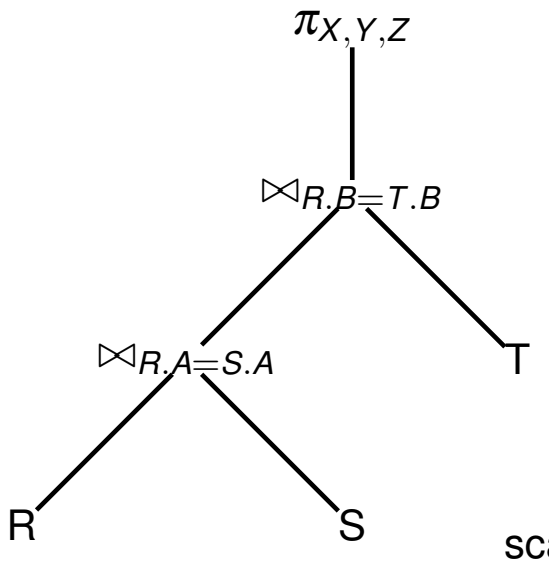
Plan 2

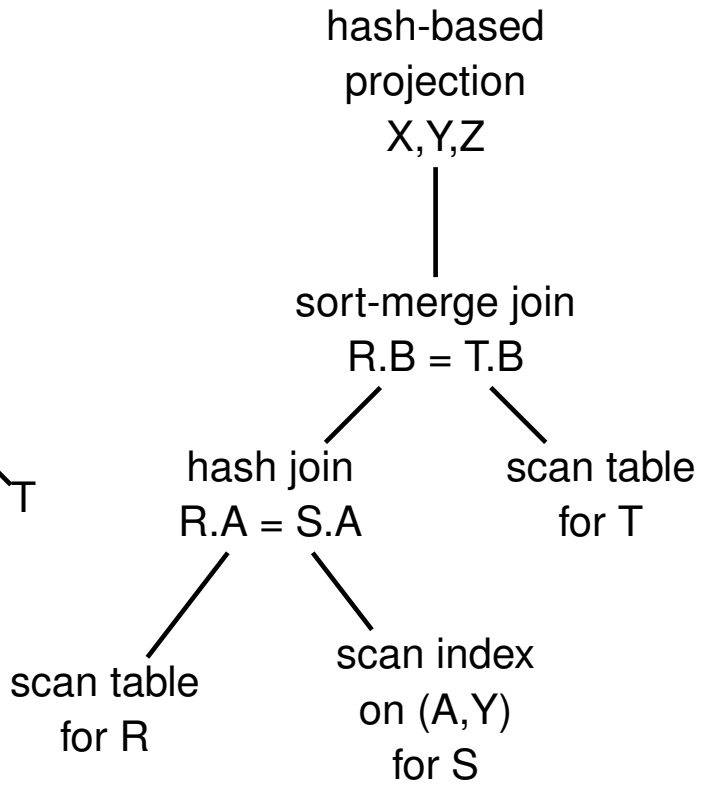
Query Processing



Query Plans

```
select R.X, S.Y, T.Z
from   R, S, T
where  R.A = S.A
and    R.B = T.B
```



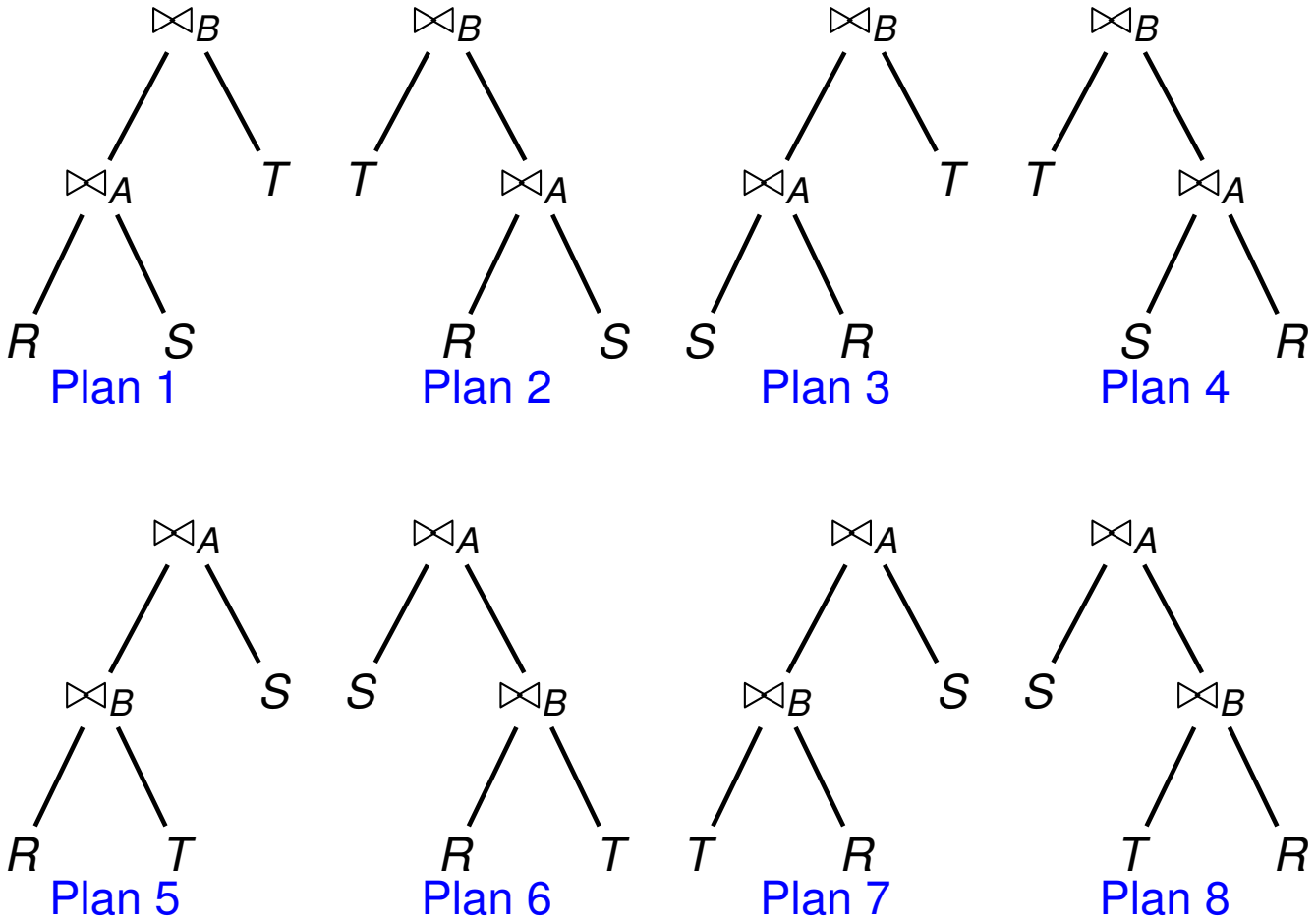


SQL query *Q* Logical plan for *Q* Physical plan for *Q*

Logical Query Plans

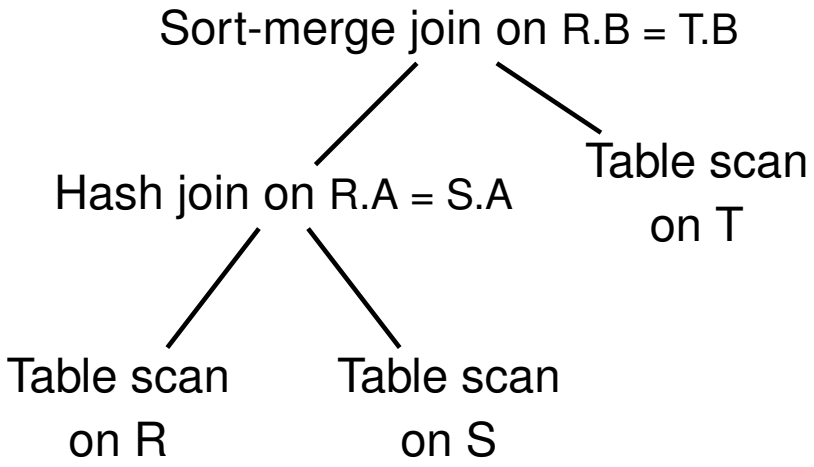
A query generally has many equivalent logical query plans

```
SELECT  *
FROM    R, S, T
WHERE   R.A = S.A
AND     R.B = T.B
```

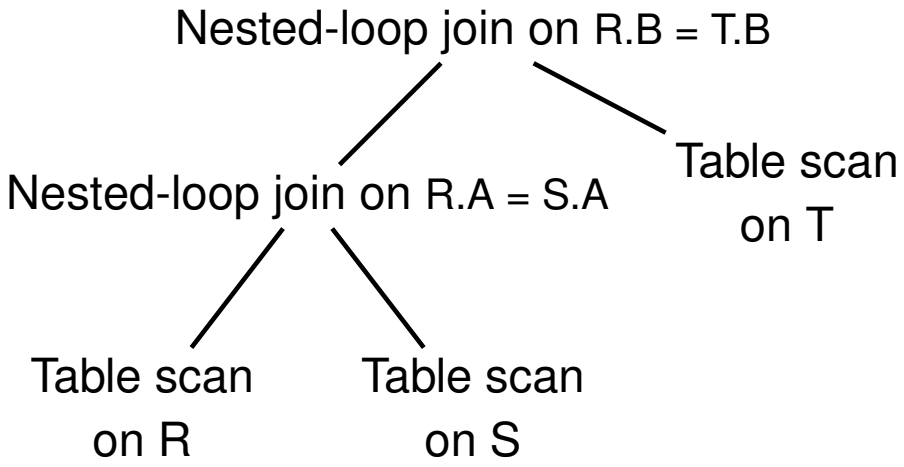


Physical Query Plans

- Each logical plan can be implemented by many physical query plans
- **Example**: Two possible physical query plans for $(R \bowtie_A S) \bowtie_B T$

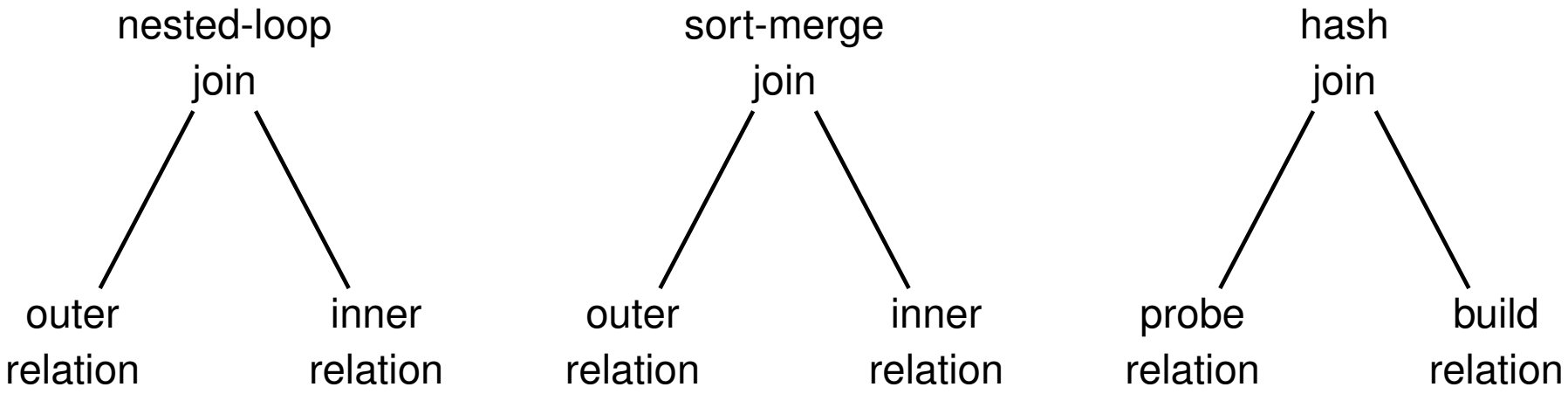


(a) Physical Plan 1



(b) Physical Plan 2

Join Plan Notation



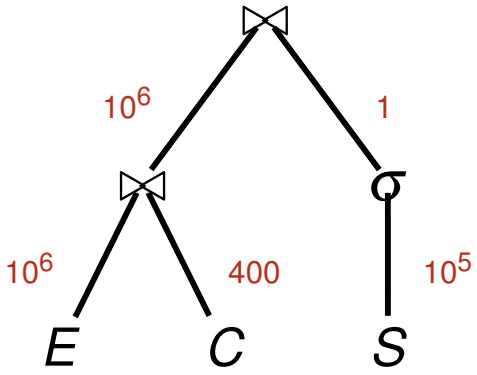
Why Optimize?

Student (sid, sname, major)
Course (cid, cname, area)
Enrol (sid,cid, grade)

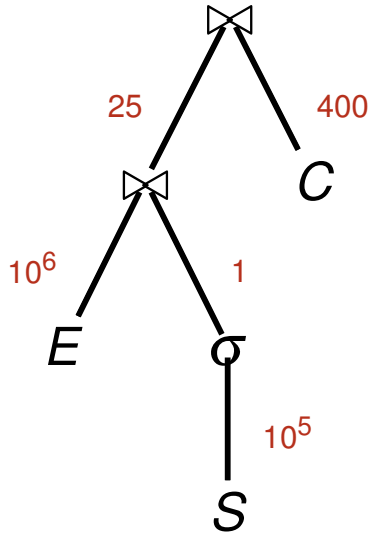
```
SELECT   *
FROM     Student S, Course C, Enrol E
WHERE    E.sid = S.sid
AND      E.cid = C.cid
AND      S.sid = 123
```

Example Query Plans:

$||C|| = 400$
$||E|| = 10^6$
$||S|| = 10^5$
$||\sigma_{sid=123}(S)|| = 1$
$||C \bowtie_{cid} E|| = 10^6$
$||E \bowtie_{sid} \sigma_{sid=123}(S)|| = 25$



Plan 1



Plan 2

Relational Algebra Equivalence Rules

attributes(R) = Set of attributes in schema of relation R
attributes(p) = Set of attributes in predicate p

1. **Commutativity of binary operators**
   - 1.1 $R \times S \equiv S \times R$
   - 1.2 $R \bowtie S \equiv S \bowtie R$
2. **Associativity of binary operators**
   - 2.1 $(R \times S) \times T \equiv R \times (S \times T)$
   - 2.2 $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
3. **Idempotence of unary operators**
   - 3.1 $\pi_{L'}(\pi_L(R)) \equiv \pi_{L'}(R)$
     if $L' \subseteq L \subseteq attributes(R)$
   - 3.2 $\sigma_{p_1}(\sigma_{p_2}(R)) \equiv \sigma_{p_1 \wedge p_2}(R)$

Relational Algebra Equivalence Rules (cont.)

4. **Commutating selection with projection**
   - 4.1 $\pi_L(\sigma_p(R)) \equiv \pi_L(\sigma_p(\pi_{L \cup attributes(p)}(R)))$
5. **Commutating selection with binary operators**
   - 5.1 $\sigma_p(R \times S) \equiv \sigma_p(R) \times S$
     if $attributes(p) \subseteq attributes(R)$
   - 5.2 $\sigma_p(R \bowtie_{p'} S) \equiv \sigma_p(R) \bowtie_{p'} S$
     if $attributes(p) \subseteq attributes(R)$
   - 5.3 $\sigma_p(R \cup S) \equiv \sigma_p(R) \cup \sigma_p(S)$

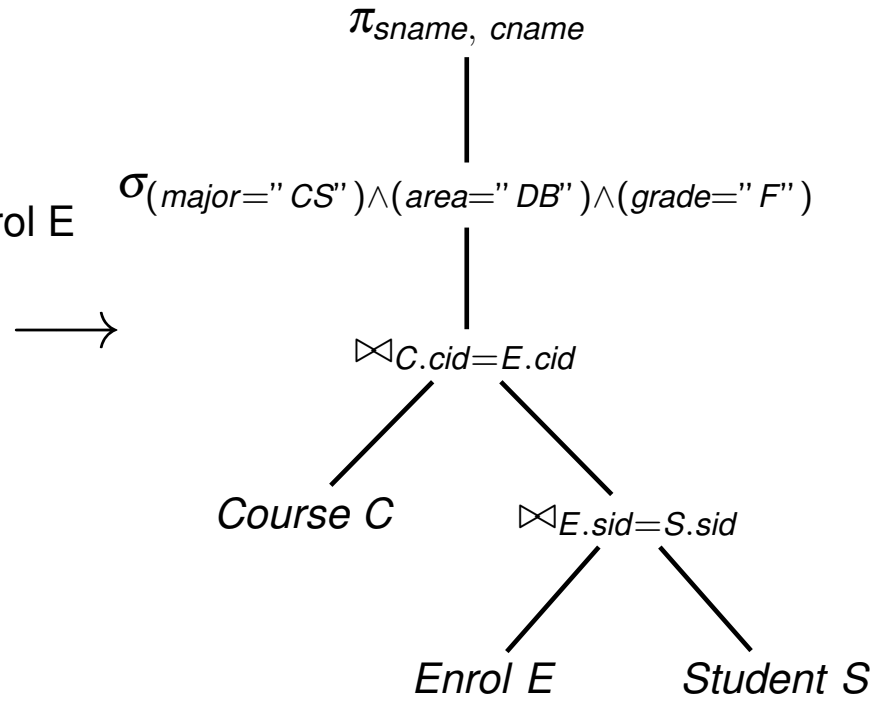
Relational Algebra Equivalence Rules (cont.)

6. **Commutating projection with binary operators**
   Let $L = L_R \cup L_S$, where $L_R \subseteq \mathit{attributes}(R)$ and $L_S \subseteq \mathit{attributes}(S)$
   - 6.1 $\pi_L(R \times S) \equiv \pi_{L_R}(R) \times \pi_{L_S}(S)$
   - 6.2 $\pi_L(R \bowtie_p S) \equiv \pi_{L_R}(R) \bowtie_p \pi_{L_S}(S)$
     if $\mathit{attributes}(p) \cap \mathit{attributes}(R) \subseteq L_R$ and $\mathit{attributes}(p) \cap \mathit{attributes}(S) \subseteq L_S$
   - 6.3 $\pi_L(R \cup S) \equiv \pi_L(R) \cup \pi_L(S)$

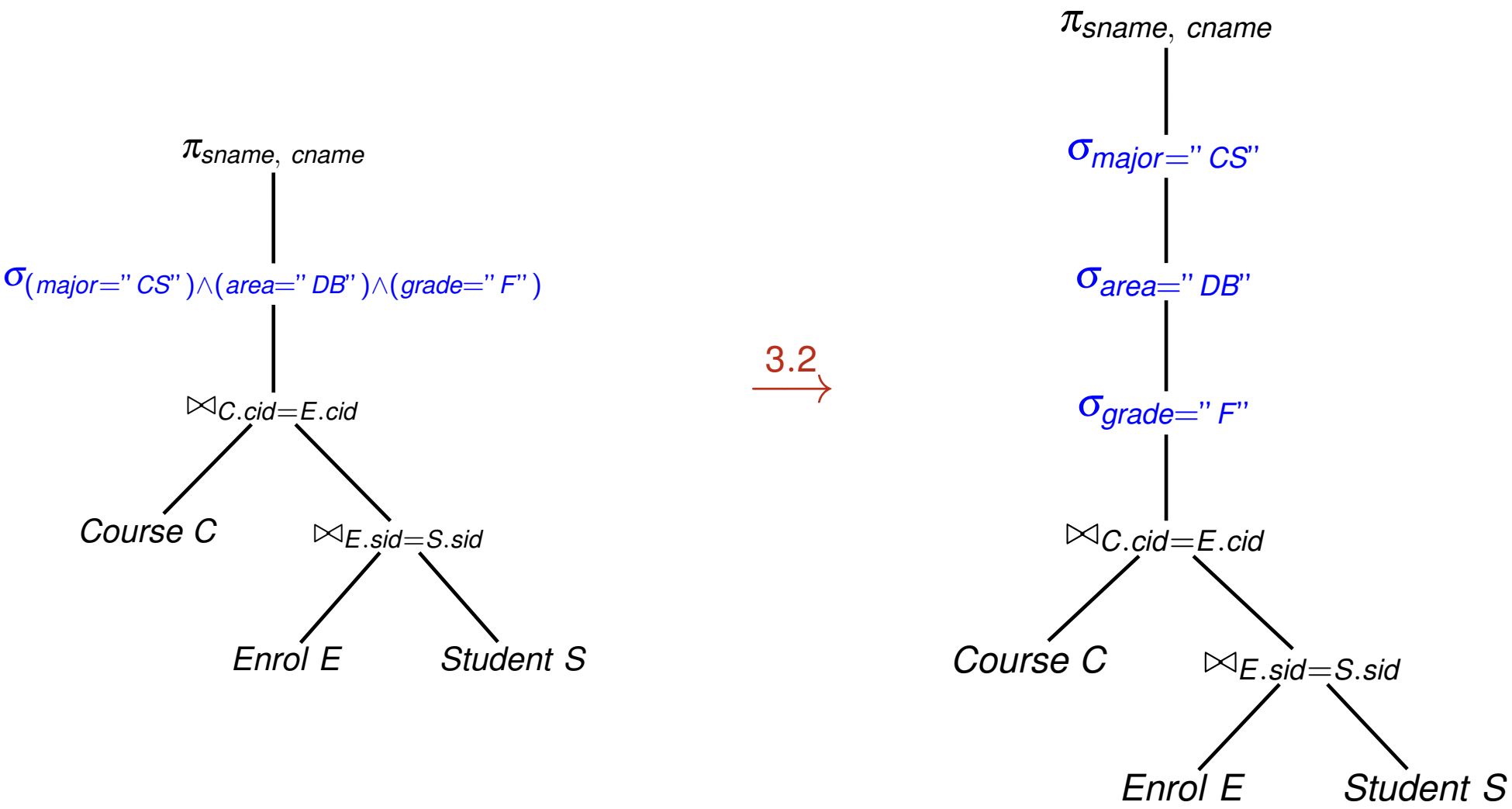
Example

`Student` (sid, sname, major)
`Course` (cid, cname, area)
`Enrol` (sid,cid, grade)

```
SELECT   sname, cname
FROM     Student S, Course C, Enrol E
WHERE    E.sid = S.sid
AND      E.cid = C.cid
AND      major = "CS"
AND      area = "DB"
AND      grade = "F"
```

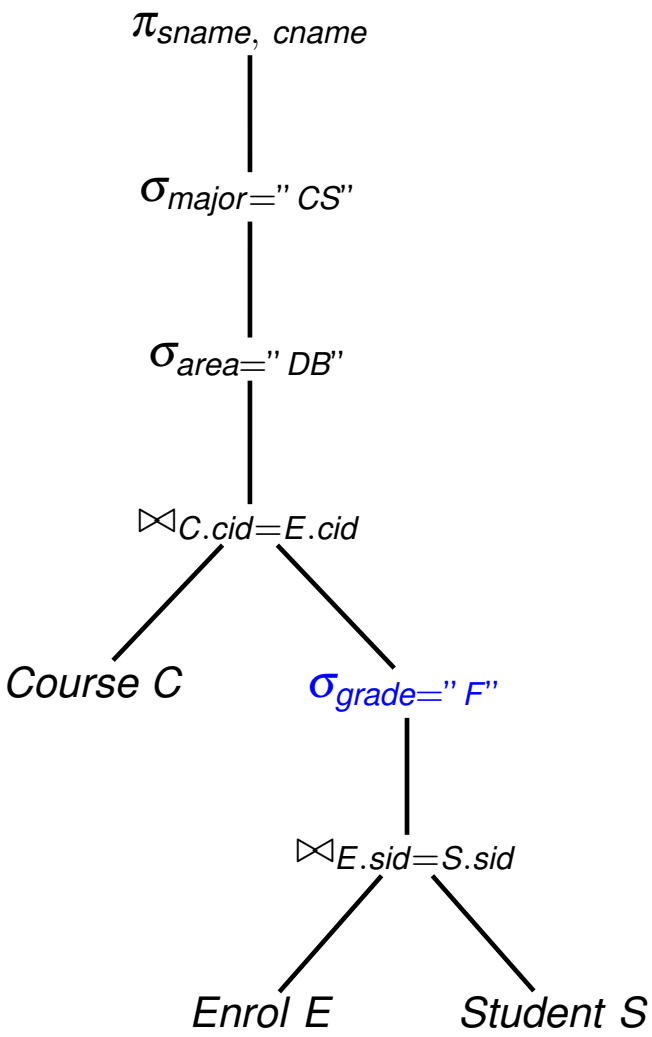


Example (cont.)

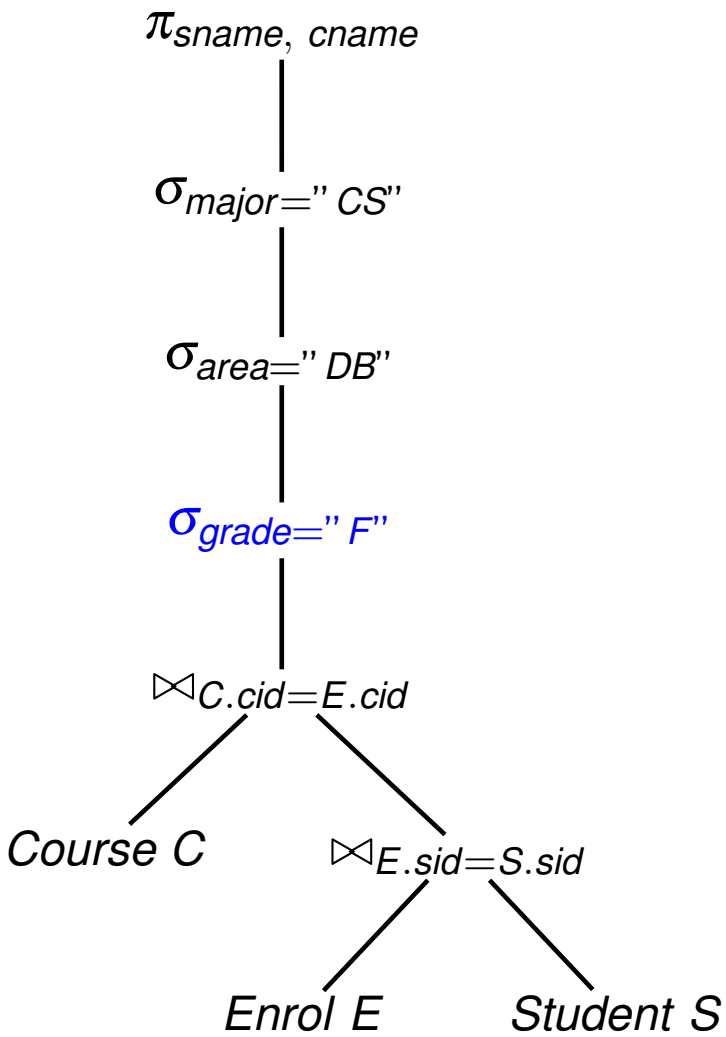


$$\sigma_{p_1}(\sigma_{p_2}(R)) \equiv \sigma_{p_1 \wedge p_2}(R)$$

Example (cont.)

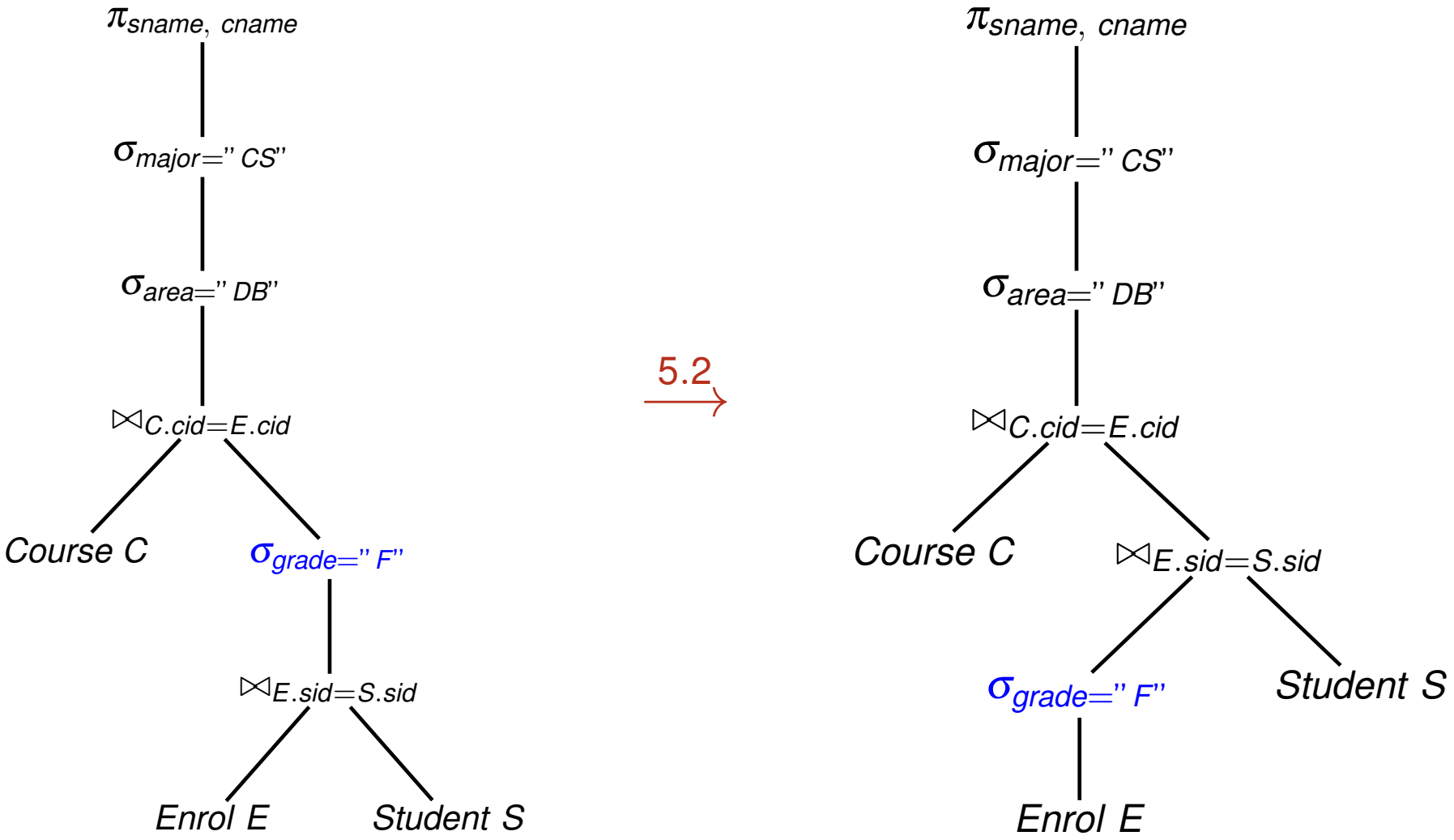


5.2 ←



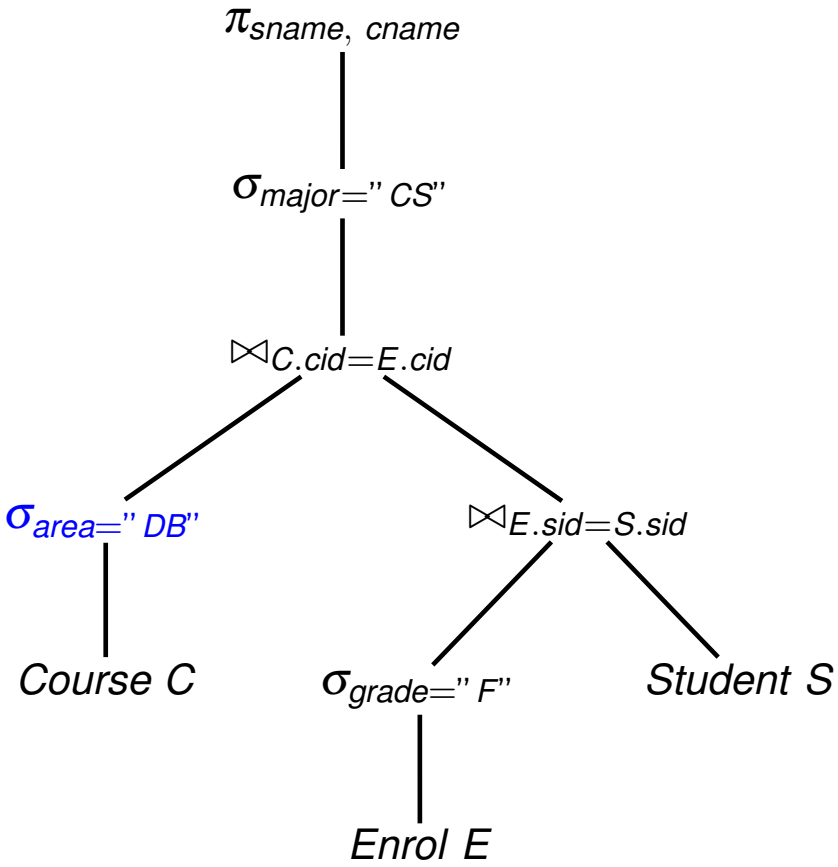
$$\sigma_p(R \bowtie_{p'} S) \equiv \sigma_p(R) \bowtie_{p'} S \text{ if } attributes(p) \subseteq attributes(R)$$

Example (cont.)

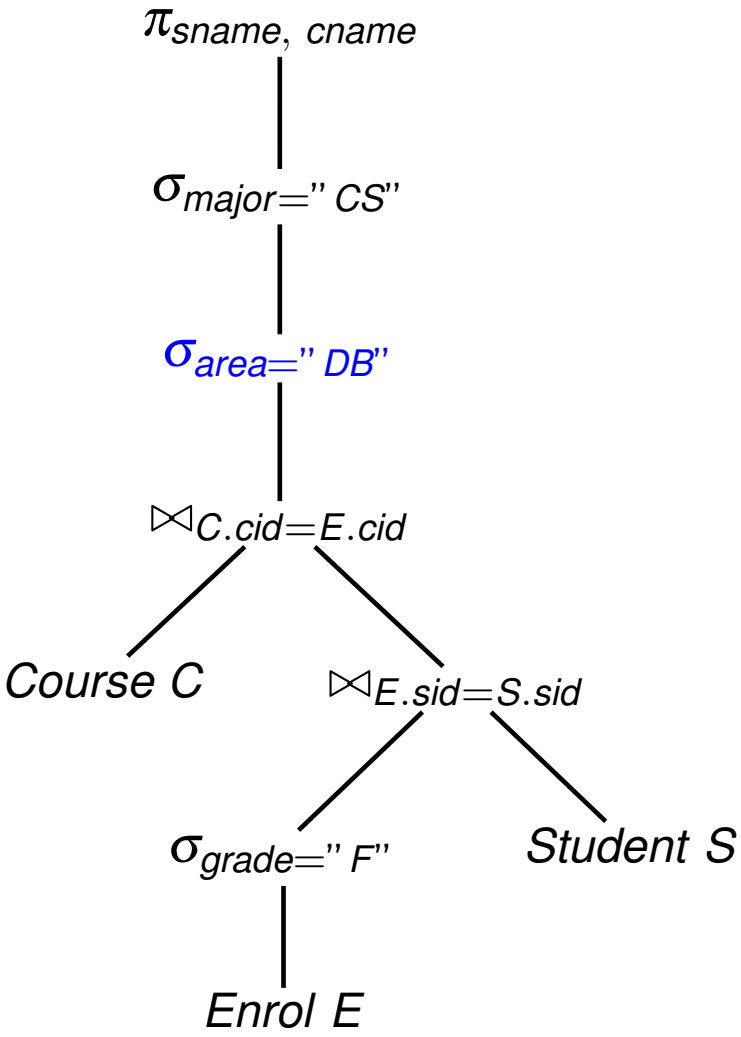


$$\sigma_p(R \bowtie_{p'} S) \equiv \sigma_p(R) \bowtie_{p'} S \text{ if } attributes(p) \subseteq attributes(R)$$

Example (cont.)

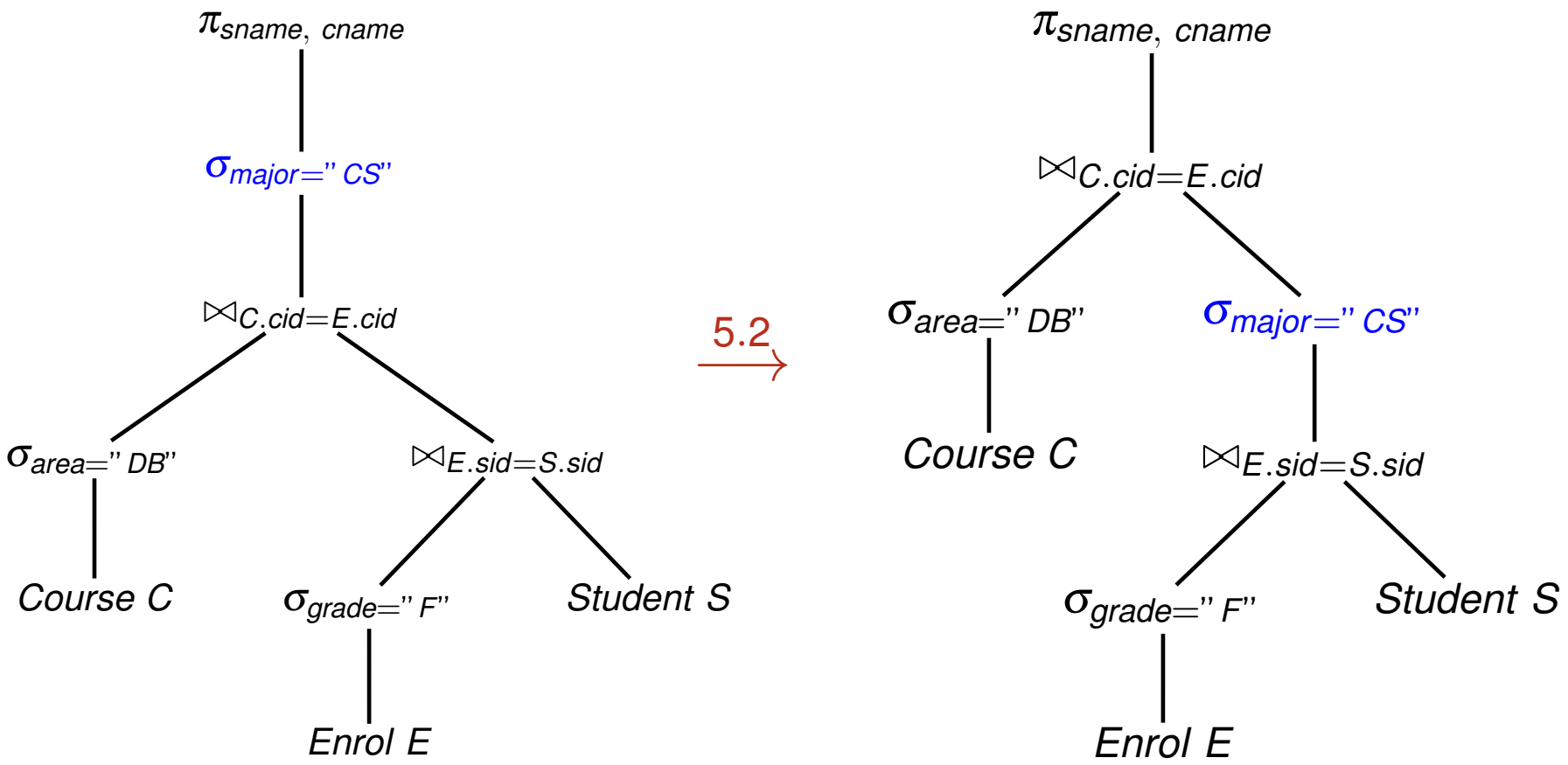


5.2 ←



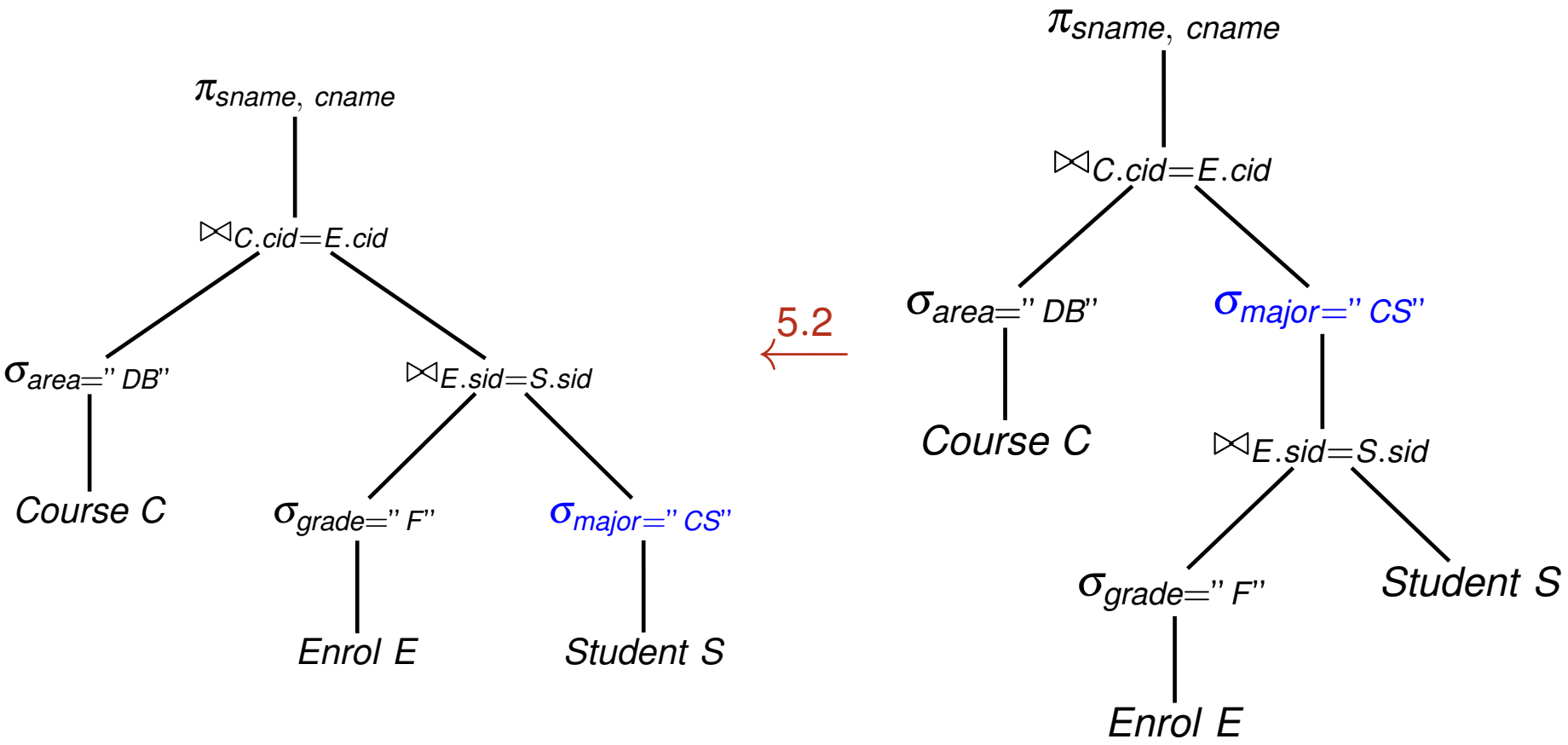
$$\sigma_p(R \bowtie_{p'} S) \equiv \sigma_p(R) \bowtie_{p'} S \text{ if } attributes(p) \subseteq attributes(R)$$

Example (cont.)



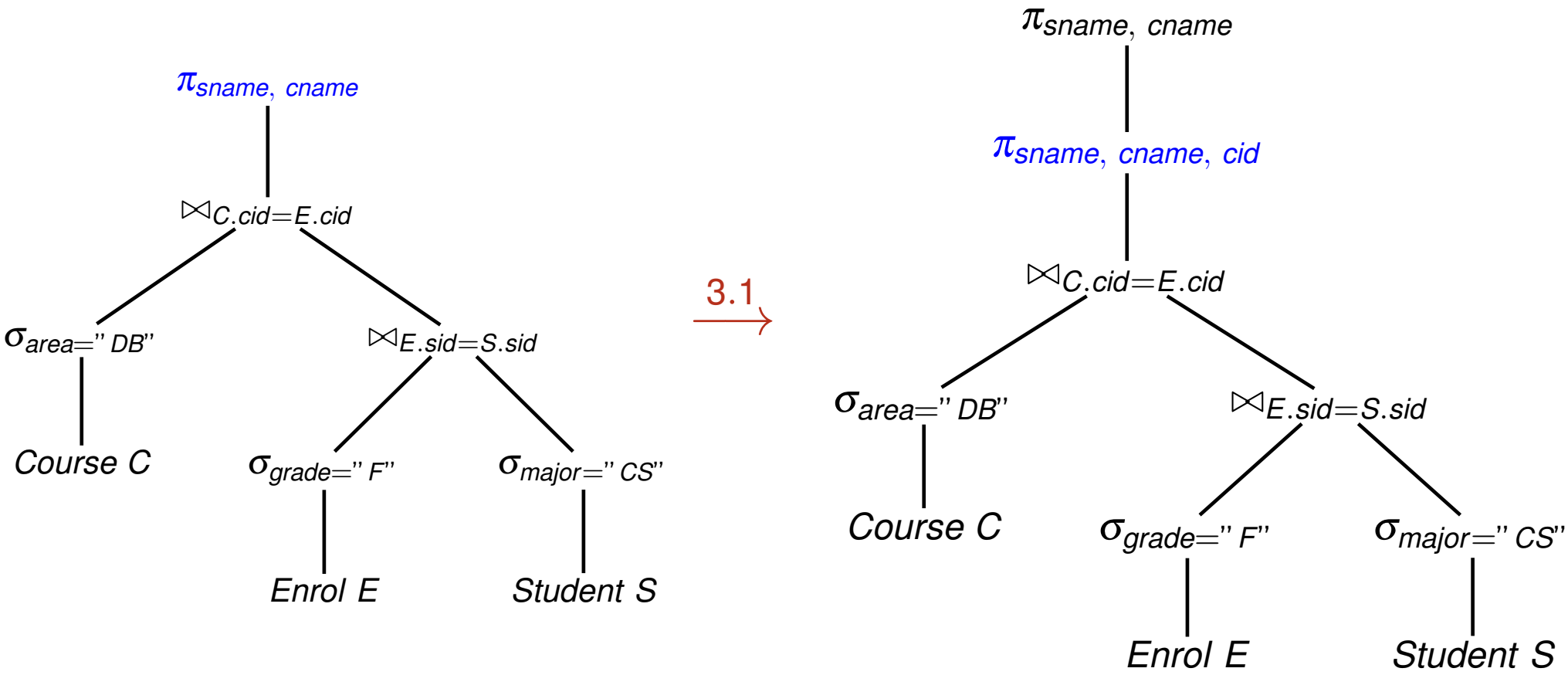
$$\sigma_p(R \bowtie_{p'} S) \equiv \sigma_p(R) \bowtie_{p'} S \text{ if } attributes(p) \subseteq attributes(R)$$

Example (cont.)



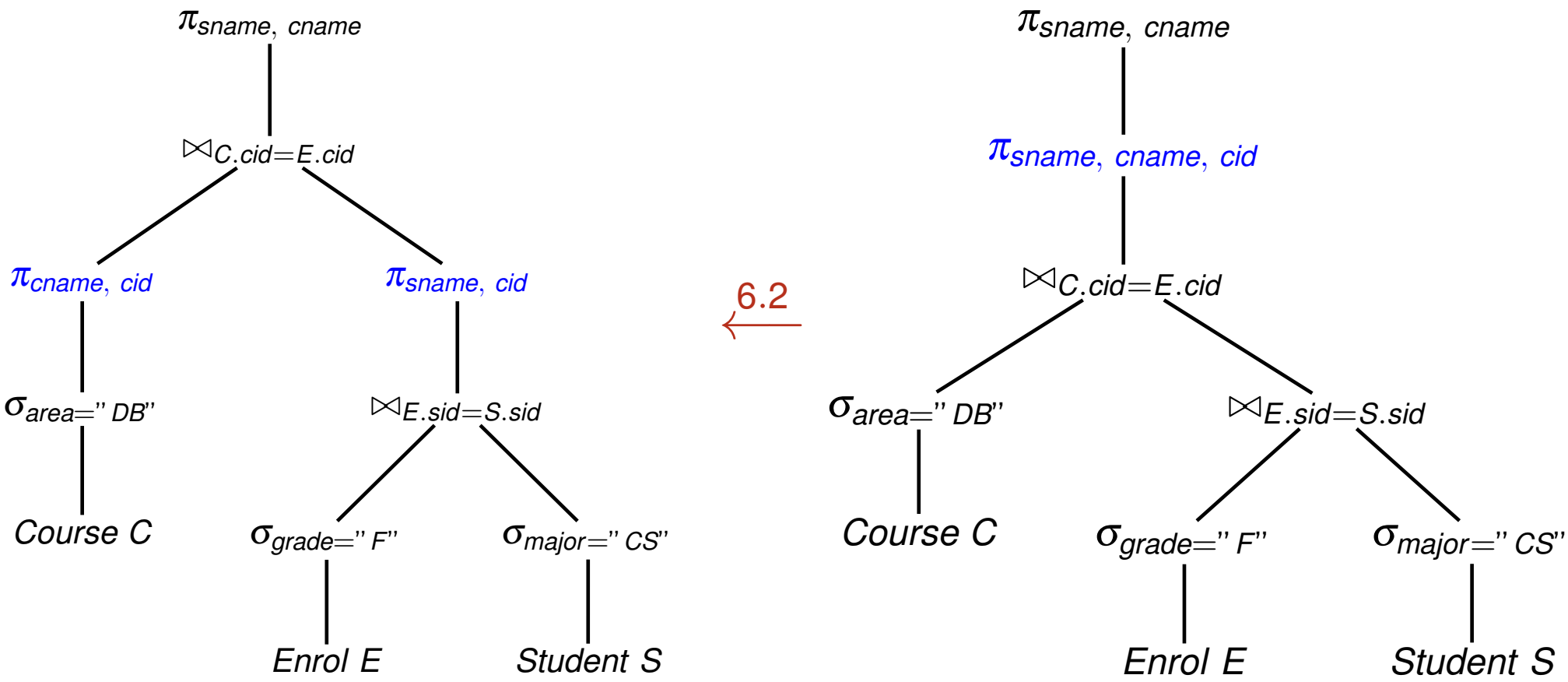
$$\sigma_p(R \bowtie_{p'} S) \equiv \sigma_p(R) \bowtie_{p'} S \text{ if } attributes(p) \subseteq attributes(R)$$

Example (cont.)



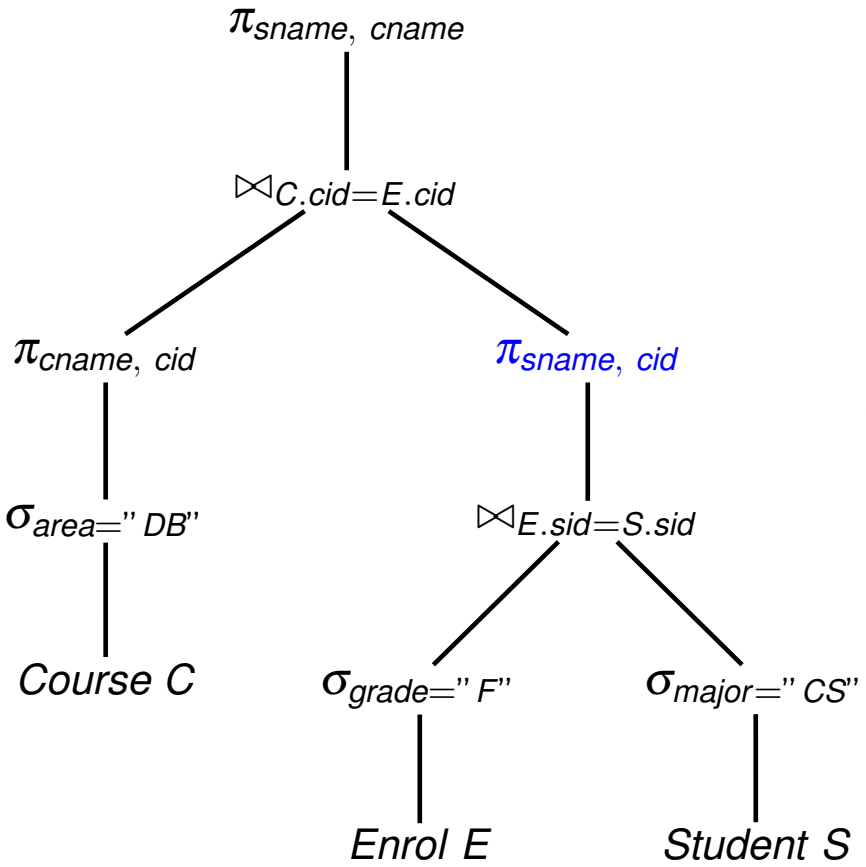
$$\pi_{L'}(\pi_L(R)) \equiv \pi_{L'}(R) \text{ if } L' \subseteq L \subseteq attributes(R)$$

Example (cont.)

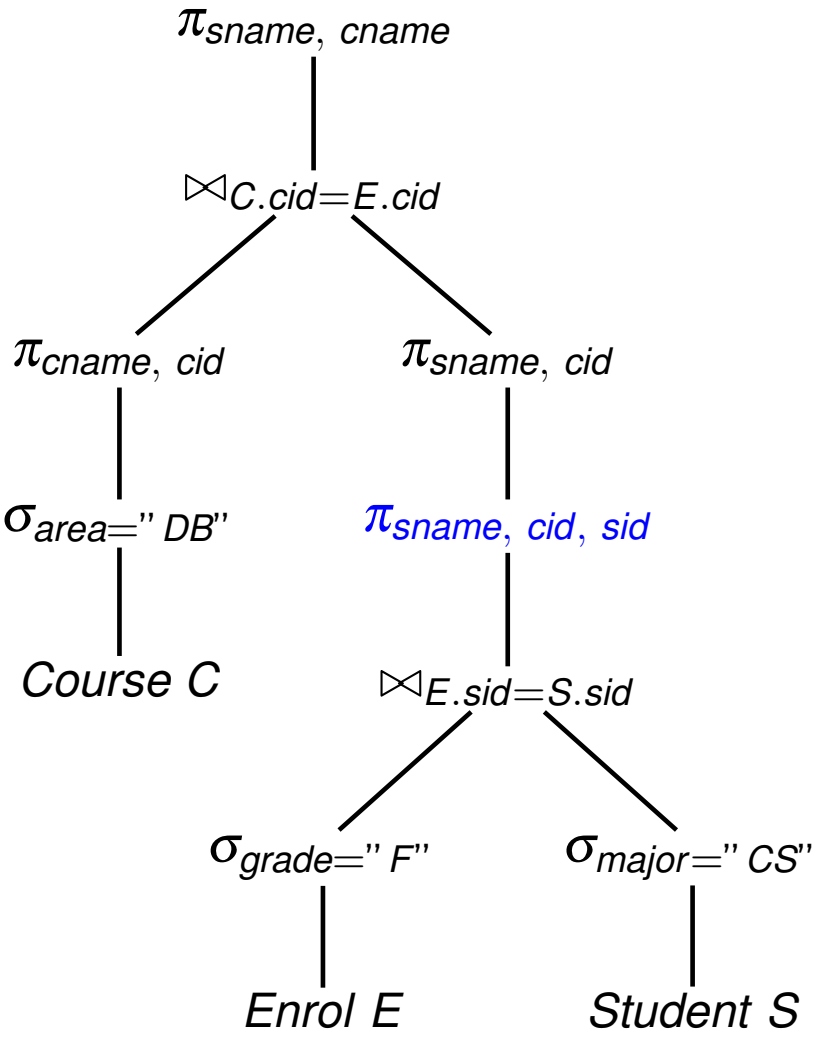


$$\pi_L(R \bowtie_p S) \equiv \pi_{L_R}(R) \bowtie_p \pi_{L_S}(S)$$

Example (cont.)

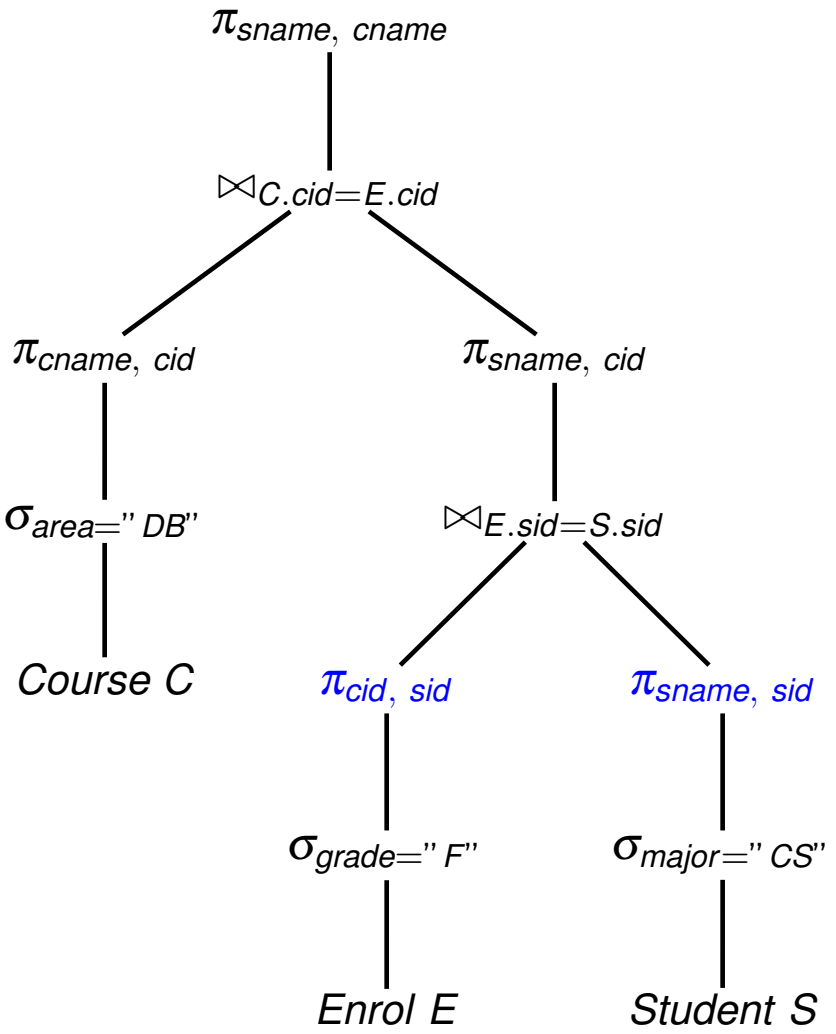


3.1 →

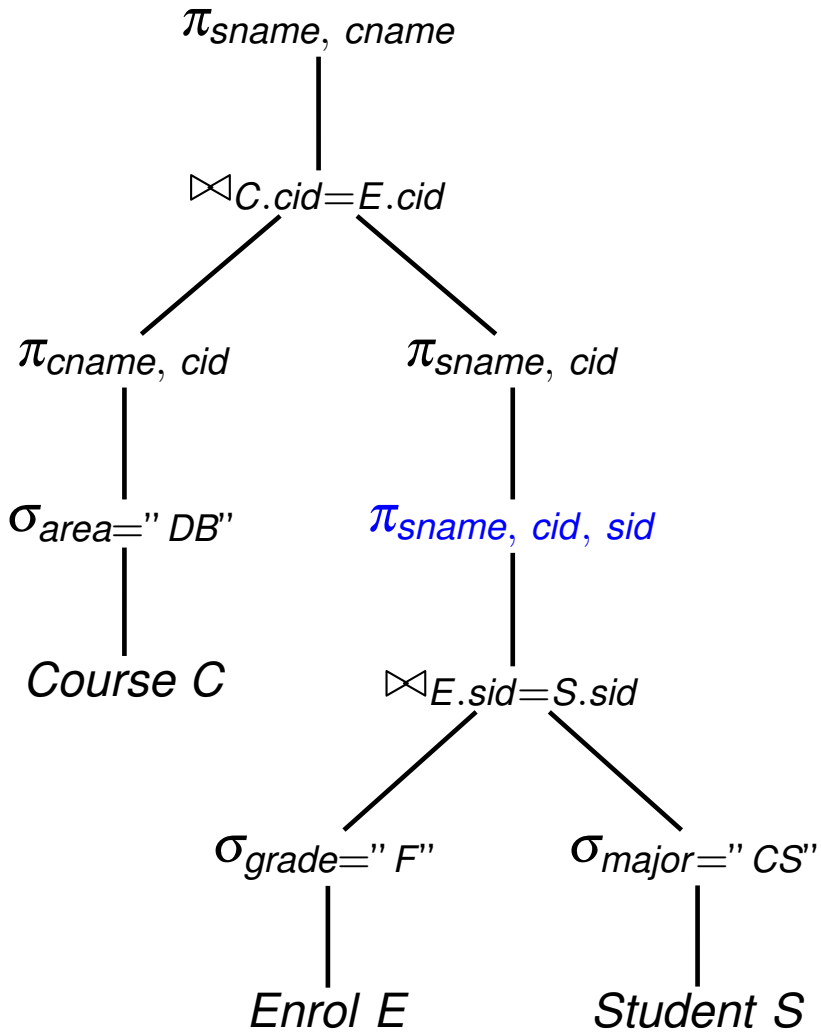


$$\pi_{L'}(\pi_L(R)) \equiv \pi_{L'}(R) \text{ if } L' \subseteq L \subseteq attributes(R)$$

Example (cont.)

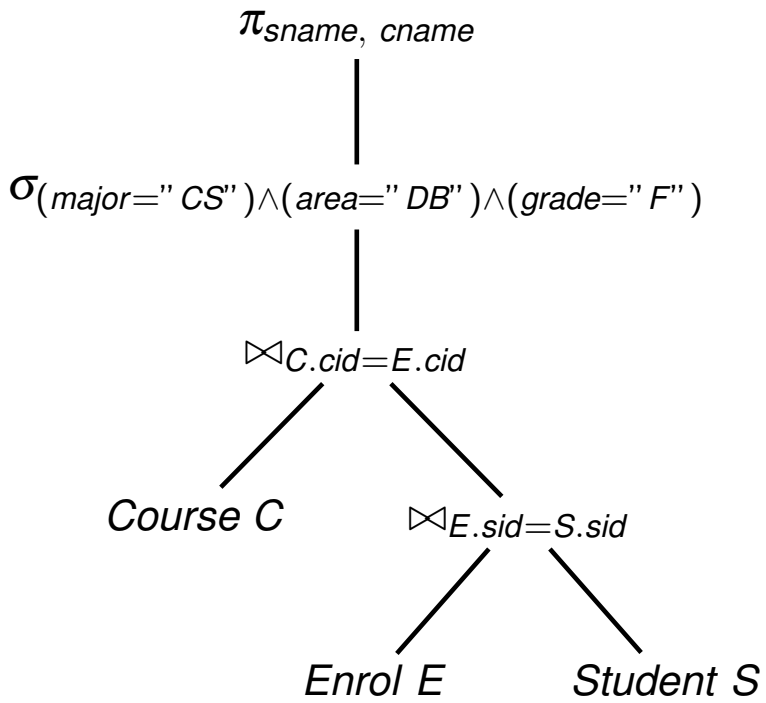


6.2

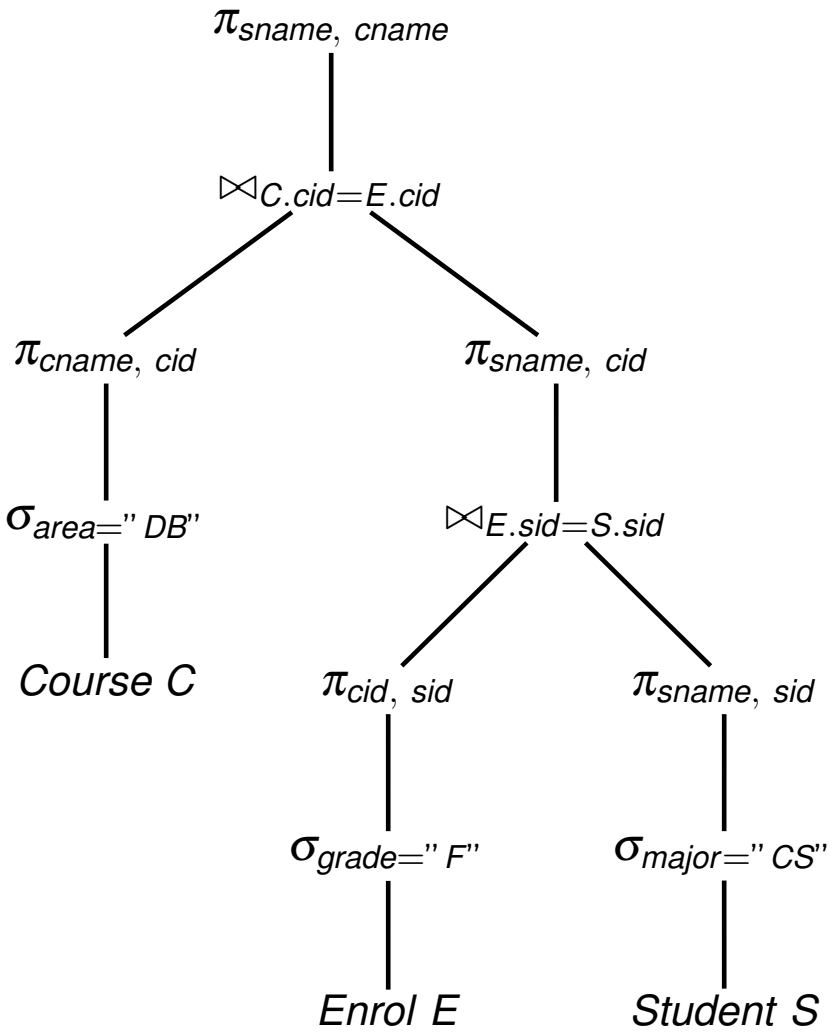


$$\pi_L(R \bowtie_p S) \equiv \pi_{L_R}(R) \bowtie_p \pi_{L_S}(S)$$

Example (cont.)



Original Query



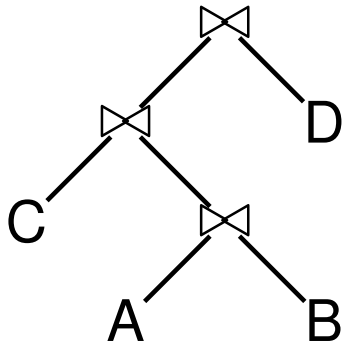
Final Query

Query Optimization

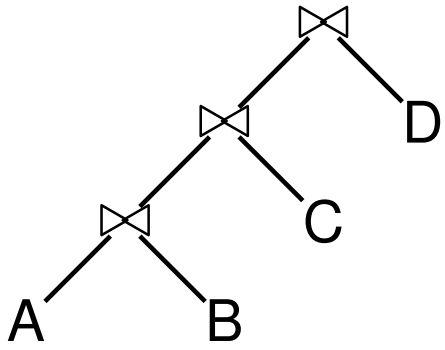
- Finds an optimal query evaluation plan for query
- Three key components:
  - Search space - what is the space of query plans being considered?
  - Plan enumeration - how to enumerate the space of query plans?
  - Cost model - how to estimate the cost of a query plan?

Types of Query Plan Trees

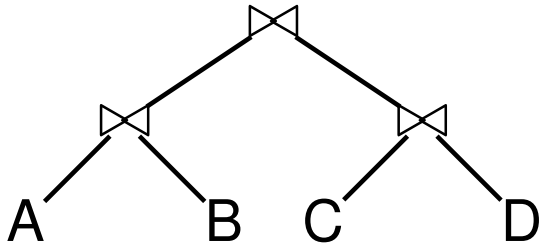
- A query plan is linear if at least one operand for each join operation is a base relation; otherwise, the plan is bushy
- A linear query plan is left-deep if every right join operand is a base relation
- A linear query plan is right-deep if every left join operand is a base relation
- **Example**: Consider the query $A \bowtie B \bowtie C \bowtie D$



Linear tree

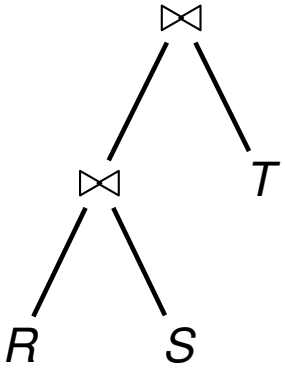


Left-deep tree

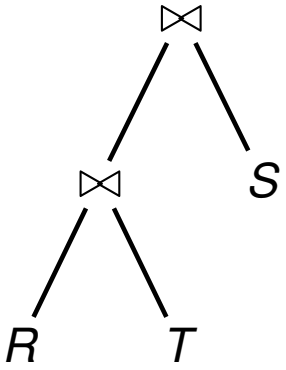


Bushy tree

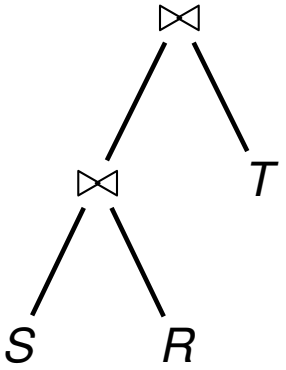
Query Plan Search Space for $R \bowtie S \bowtie T$



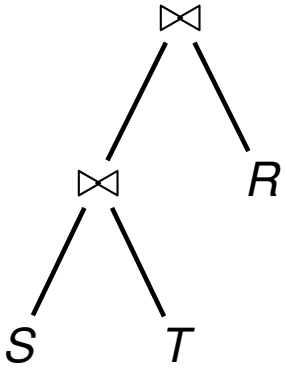
Plan 1



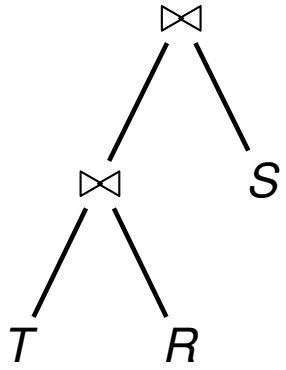
Plan 2



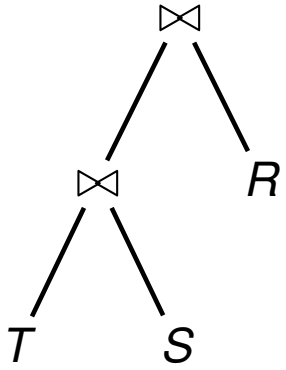
Plan 3



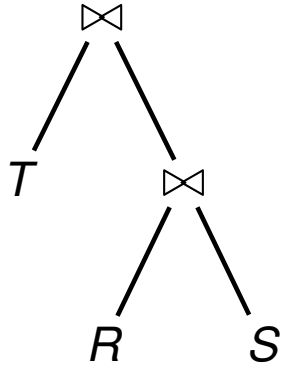
Plan 4



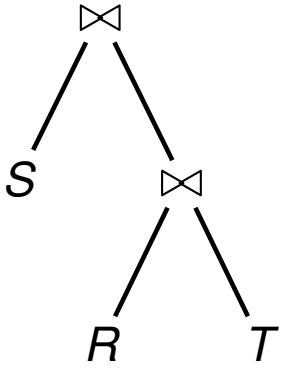
Plan 5



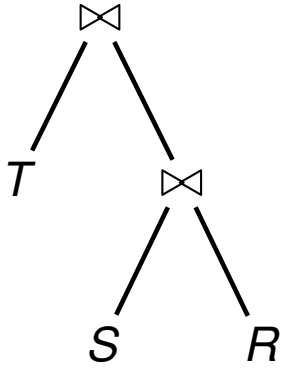
Plan 6



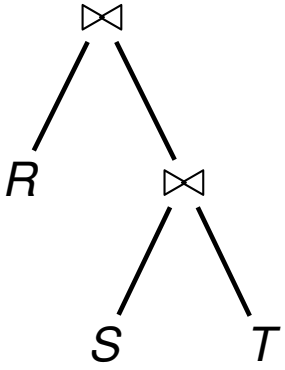
Plan 7



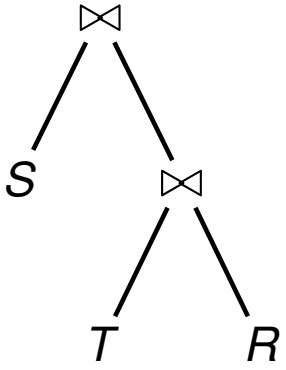
Plan 8



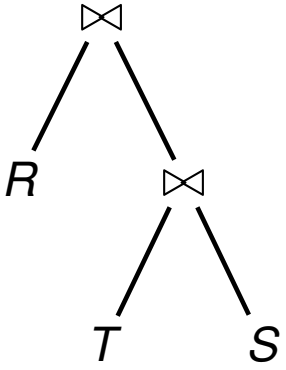
Plan 9



Plan 10



Plan 11



Plan 12

Query Plan Enumeration

Dynamic programming formulation

Input: A SPJ query q on relations $R_1, R_2, \cdots, R_n$
Output: An optimal query plan for q

```
01.  for i = 1 to n do
02.      optPlan({R_i}) = best access plan for R_i
03.  for i = 2 to n do
04.      for each S ⊆ {R_1, ··· , R_n}, |S| = i do
05.          bestPlan = dummy plan with cost(bestPlan) = ∞
06.          for each S_j, S_k, |S_j| ∈ [1, i), S = S_j ∪ S_k do
07.              p = best way to join optPlan(S_j) and optPlan(S_k)
08.              if (cost(p) ≤ cost(bestplan)) then
09.                bestPlan = p
10.          optPlan(S) = bestPlan
11.  return optPlan({R_1, ··· , R_n})
```

Query Plan Enumeration: Example

- Schema: **R**(A,B,C,D), **S**(X,Y), **T**(E,F,G)
- Query:

```
select *
from   R join S on R.A = S.X join T on R.D = T.F
where  R.B > 10
and    R.C = 20
and    T.E < 100
```

- Available B^+-tree indexes: I_B, I_C, I_E
- Assumptions on database system
  - Supports only one join algorithm: hash join
  - Avoids cartesian products

Example: Enumeration of Single-relation Plans

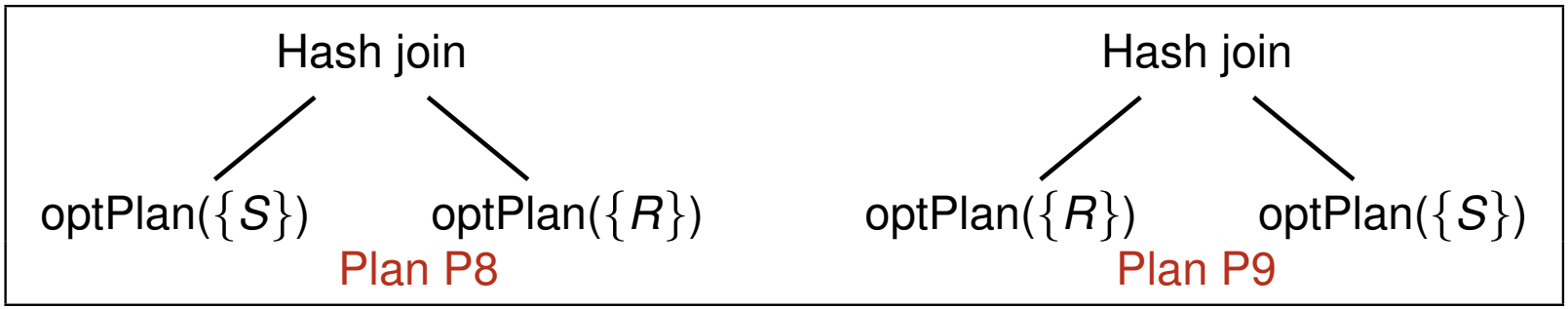
$$\sigma_p(R \bowtie_{R.A=S.X} S \bowtie_{R.D=T.F} T),\ p = (R.B > 10) \wedge (R.C = 20) \wedge (T.E < 100)$$

- **Plans for $\{\mathbf{R}\}$**
  - Plan P1: Table scan with "$(B > 10) \wedge (C = 20)$"
  - Plan P2: Index seek with I_B & RID-lookups with "$C = 20$"
  - Plan P3: Index seek with I_C & RID-lookups with "$B > 10$"
  - Plan P4: Index intersection with I_B & I_C, and RID-lookups
  - Assume $cost(P3) < cost(P4) < cost(P2) < cost(P1)$
  - optPlan($\{R\}$) = P3
- **Plans for $\{\mathbf{S}\}$**
  - Plan P5: Table scan of S
  - optPlan($\{S\}$) = P5
- **Plans for $\{\mathbf{T}\}$**
  - Plan P6: Table scan of T with "$(E < 100)$"
  - Plan P7: Index seek with I_E & RID-lookups
  - Assume $cost(P7) < cost(P6)$
  - optPlan($\{T\}$) = P7

Example: Enumeration of Two-relation Plans

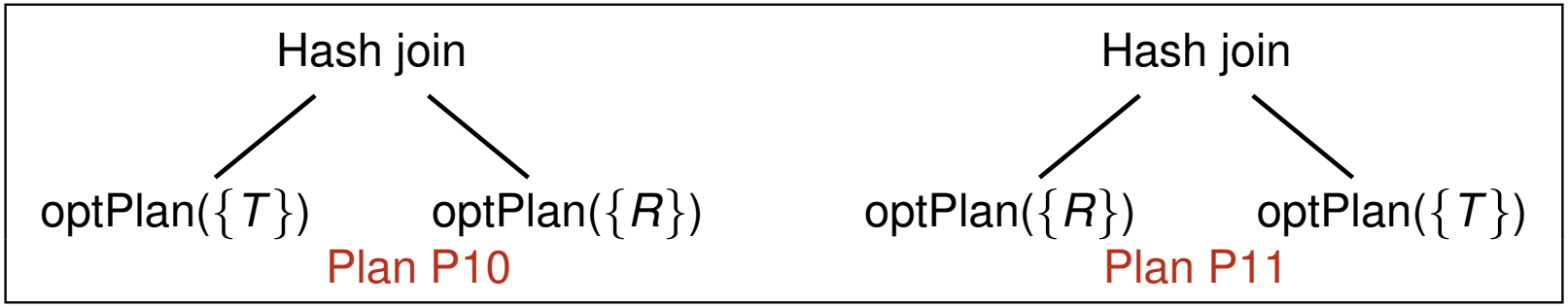
$$\sigma_p(R \bowtie_{R.A=S.X} S \bowtie_{R.D=T.F} T),\ p = (R.B > 10) \wedge (R.C = 20) \wedge (T.E < 100)$$

- **Plans for $\{\mathbf{R}, \mathbf{S}\}$**



  - Assume $cost(P8) < cost(P9)$
  - optPlan($\{R, S\}$) = P8

- **Plans for $\{\mathbf{R}, \mathbf{T}\}$**

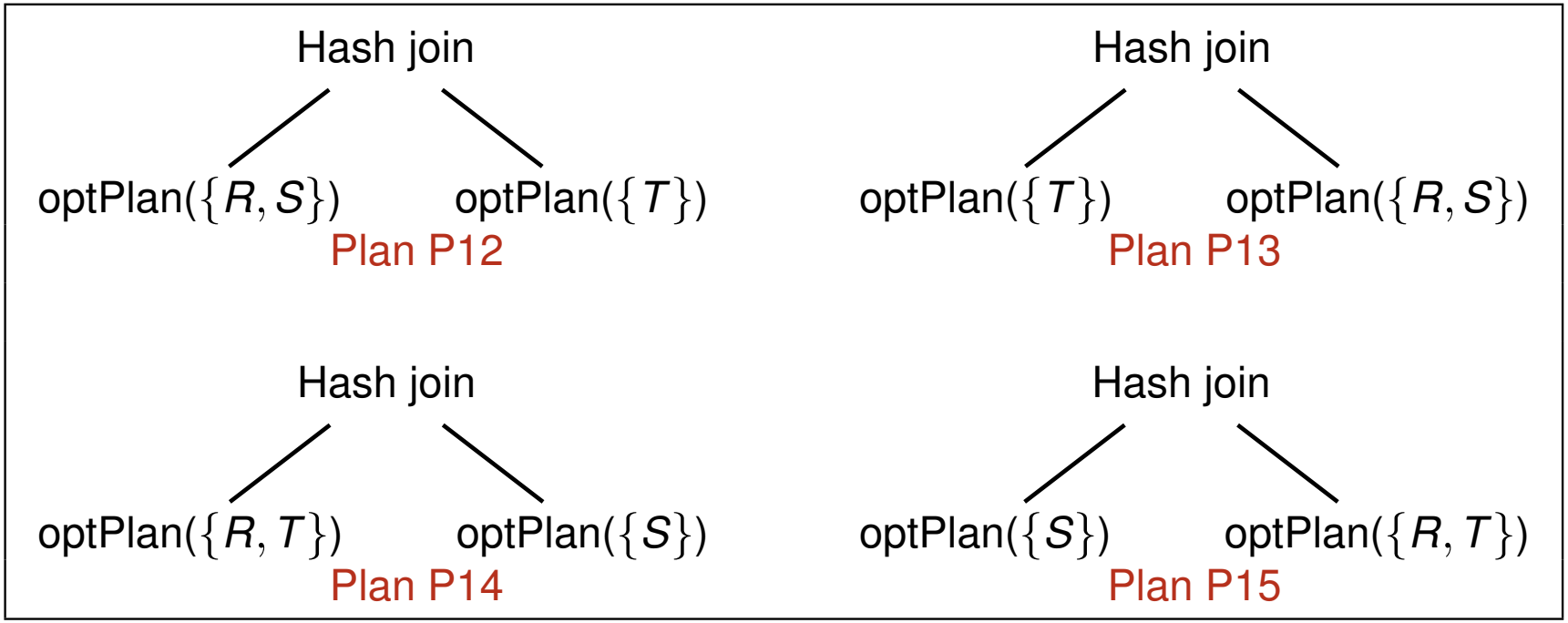


  - Assume $cost(P11) < cost(P10)$
  - optPlan($\{R, T\}$) = P11

Example: Enumeration of Three-relation Plans

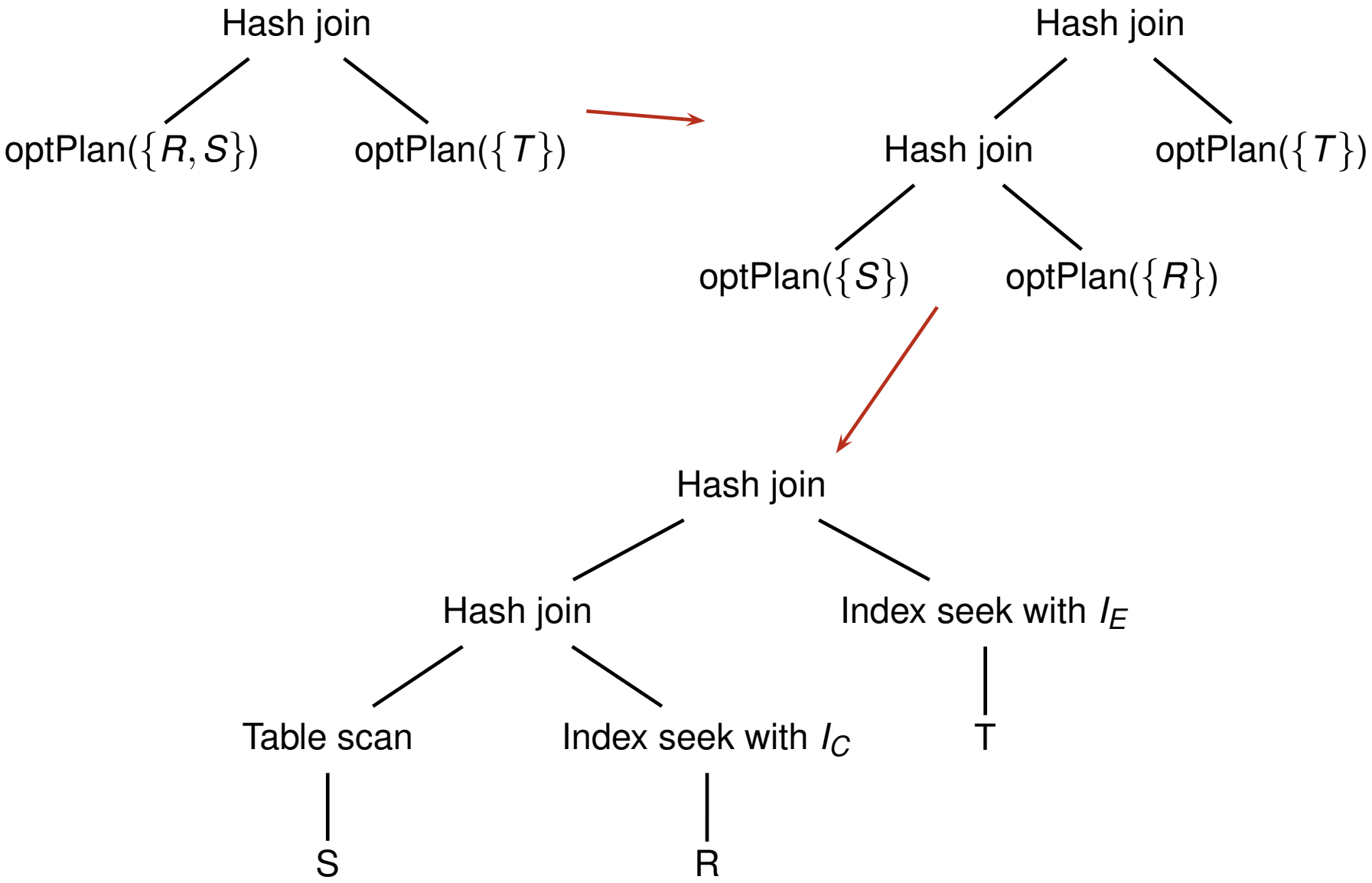
$$\sigma_p(R \bowtie_{R.A=S.X} S \bowtie_{R.D=T.F} T),\ p = (R.B > 10) \wedge (R.C = 20) \wedge (T.E < 100)$$

- **Plans for $\{\mathbf{R}, \mathbf{S}, \mathbf{T}\}$**



  - Assume $cost(P12) < cost(P14) < cost(P13) < cost(P15)$
  - optPlan($\{R, S, T\}$) = P12

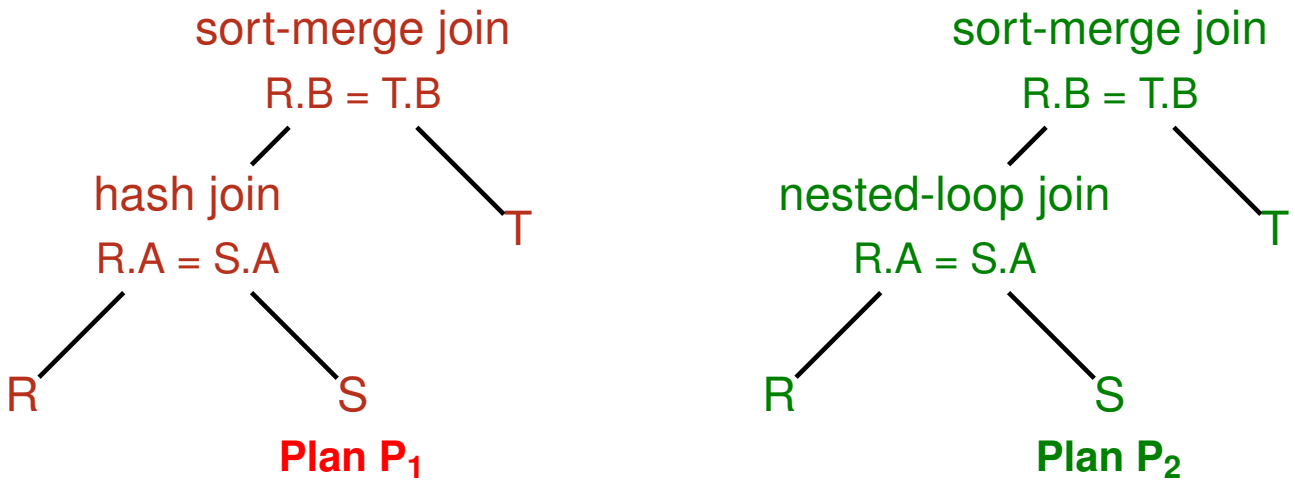
Example: Optimal Plan



System R Optimizer

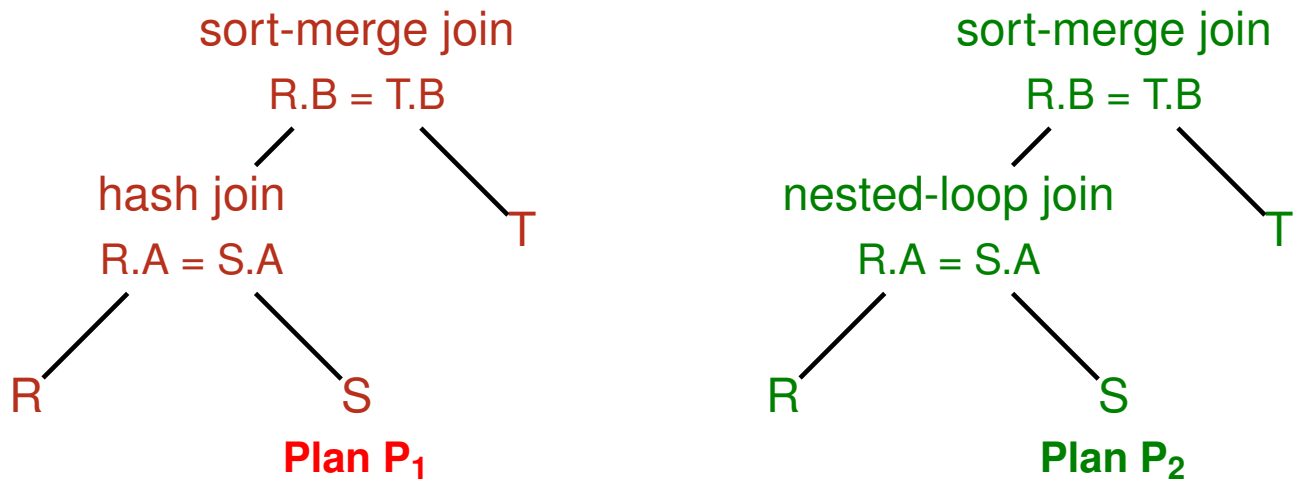
- Uses heuristics to prune search space:
  - Enumerates only left-deep query plans
  - Avoids cross-product query plans
  - Considers early selections & projections
- Uses **enhanced dynamic programming approach** that considers sort order of query plan's output
  - Maintains optPlan(S_i, o_i) instead of optPlan(S_i)
  - o_i captures the sort order of output produced by query plan wrt S_i
  - o_i = either *null* if output is unordered or a sequence of attributes
  - optPlan(S_i, o_i) = cheapest query plan for relations S_i with output ordered by o_i if $o_i \neq null$

System R Optimizer: Example



- Suppose that Cost(hash join of R & S) $<$ Cost(nested loop join of R & S)
- In **basic dynamic programming approach**
  - optPlan($\{R, S\}$) = hash join of R and S
  - Therefore, plan P_2 will not be considered
- However, it is possible for Cost(P_2) $<$ Cost(P_1) if the output of the nested-loop join is sorted on $R.B$

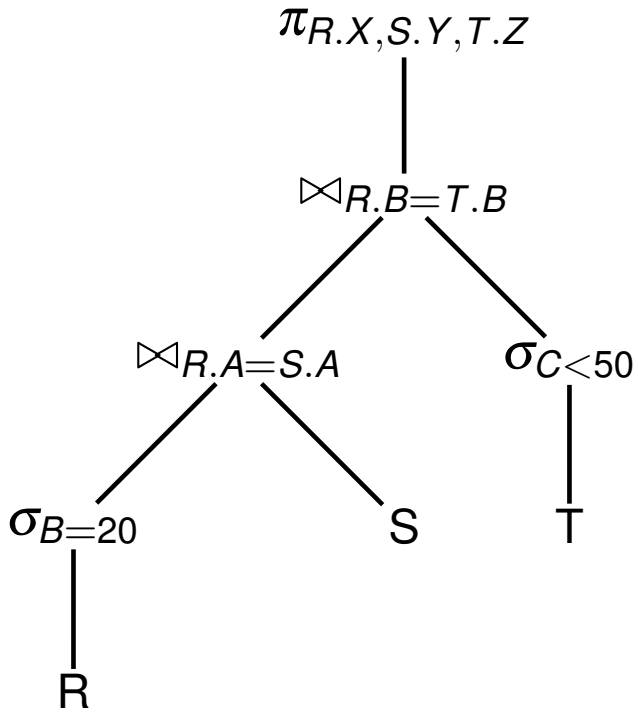
System R Optimizer: Example (cont.)



- Assume that the output of nested-loop join is sorted on $R.B$
- In **enhanced dynamic programming approach**
  - optPlan($\{R, S\}$, null) = hash join of R and S
  - optPlan($\{R, S\}$, ($R.B$)) = nested-loop join of R and S
  - Therefore, plan P_2 will be considered during plan enumeration for $\{R, S, T\}$

Cost Estimation of Query Plans

- Cost estimation involves the following:
  1. What is the evaluation cost of each operation?
     - ★ Cost model depends on: size of input operands, available buffer pages, available indexes, etc.
  2. What is the output size of each operation?



How to estimate?

- Cost model for each operator's algorithms
- Estimation assumptions
  - **Uniformity assumption**: uniform distribution of attribute values
  - **Independence assumption**: independent distribution of values in different attributes
  - **Inclusion assumption**: For $R \bowtie_{R.A=S.B} S$, if $||\pi_A(R)|| \leq ||\pi_B(S)||$, then $\pi_A(R) \subseteq \pi_B(S)$
- Database statistics
  - relation cardinality (i.e. number of tuples)
  - number of distinct values in each column
  - the highest & lowest values in each column
  - frequent values of some columns
  - column group statistics
  - histograms
  - etc.

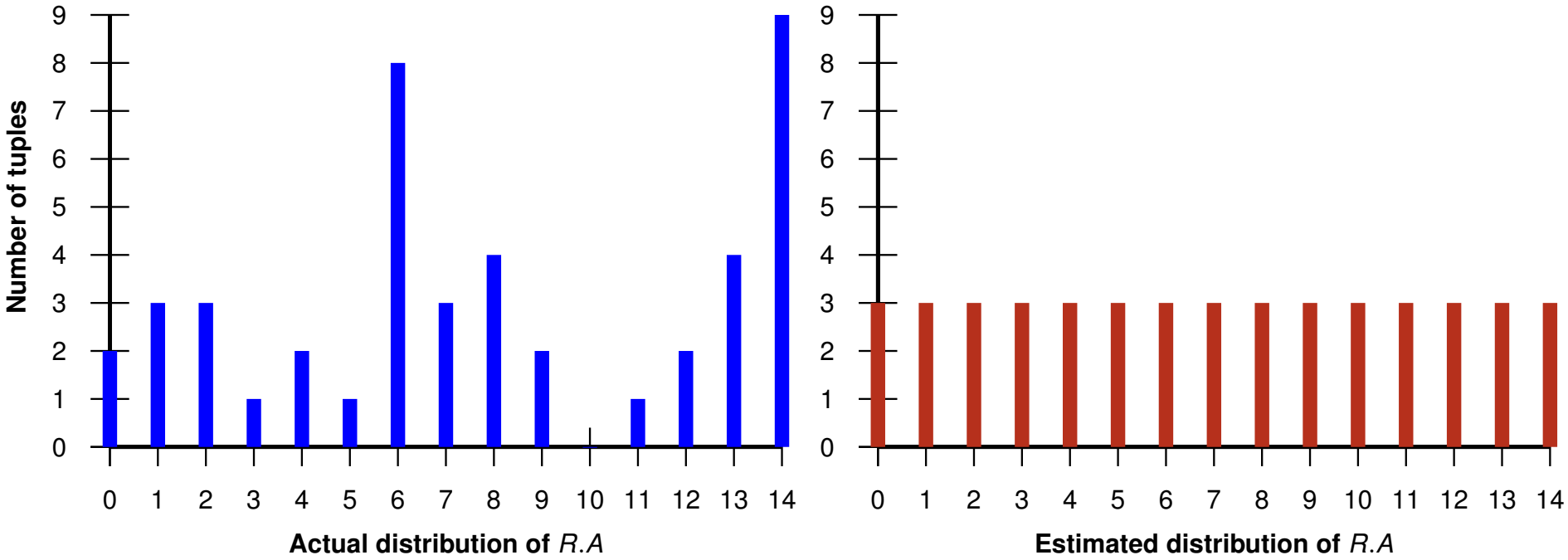
Size Estimation

- Consider a query $q = \sigma_p(e)$
  - $p = t_1 \wedge t_2 \wedge \cdots \wedge t_n$
  - $e = R_1 \times R_2 \times \cdots \times R_m$
- How to estimate $||q||$?
- We have $||e|| = \prod_{i=1}^{m} ||R_i|| = ||R_1|| \times ||R_2|| \times \cdots \times ||R_m||$
- Each term t_i potentially filters out some tuples in e
- Reduction factor of a term t_i (denoted by $rf(t_i)$) is the fraction of tuples in e that satisfy t_i; i.e., $rf(t_i) = \frac{||\sigma_{t_i}(e)||}{||e||}$
- Reduction factor is also known as selectivity factor
- Assuming the terms in p are statistically independent,

$$||q|| \approx ||e|| \times \prod_{i=1}^{n} rf(t_i)$$

Estimation of Selectivity Factor

- Consider a relation $R(A, \cdots)$ with $||R|| = 45$ and $||\pi_A(R)|| = 15$
- Using **uniformity assumption**, $rf(A = c) \approx \frac{1}{||\pi_A(R)||}$



Size Estimation: Example

- Consider a relation $R(\underline{A}, B, C)$
  - $||R|| = 1000$
  - $||\pi_B(R)|| = 10$
  - $||\pi_C(R)|| = 50$
- What is the output size of query Q: $\sigma_{B=10 \wedge C=23}(R)$?
- By uniformity assumption,
  - $rf(B = 10) \approx \frac{1}{10}$
  - $rf(C = 23) \approx \frac{1}{50}$
- By independence assumption,
  - $rf((B = 10) \wedge (C = 23)) \approx \frac{1}{10} \times \frac{1}{50}$
- $||Q|| \approx 1000 \times \frac{1}{10} \times \frac{1}{50} = 2$

Join Selectivity

- Join selectivity factor = selectivity factor for join predicates
- Consider query Q: SELECT * FROM R JOIN S ON R.A = S.B
  - $rf(R.A = S.B) = \frac{||R \bowtie_{R.A=S.B} S||}{||R|| \times ||S||}$
- Inclusion assumption: Consider $R \bowtie_{R.A=S.B} S$

  If $||\pi_A(R)|| \leq ||\pi_B(S)||$, then $\pi_A(R) \subseteq \pi_B(S)$
- Join selectivity estimation:
  - Assume $||\pi_A(R)|| \leq ||\pi_B(S)||$
  - By inclusion assumption, every R-tuple joins with some S-tuple
  - By uniformity assumption, there are $\frac{||S||}{||\pi_B(S)||}$ S-tuples corresponding to each S.B value
  - Therefore, each R-tuple joins with $\frac{||S||}{||\pi_B(S)||}$ S-tuples
  - Thus, $||Q|| \approx ||R|| \times \frac{||S||}{||\pi_B(S)||}$

$$\boxed{rf(R.A = S.B) \approx \frac{1}{\max\{||\pi_A(R)||, ||\pi_B(S)||\}}}$$

Join Selectivity: Example

- Inclusion assumption: Consider $R \bowtie_{R.A=S.B} S$

  If $||\pi_A(R)|| \leq ||\pi_B(S)||$, then $\pi_A(R) \subseteq \pi_B(S)$

- Join selectivity estimation:

$$rf(R.A = S.B) \approx \frac{1}{\max\{||\pi_A(R)||, ||\pi_B(S)||\}}$$

- **Example**: Consider query Q: $R \bowtie_{dept} S$

R

name	dept
Alice	CS
Bob	CS
Carol	CS
Dave	CS
Eve	CS
Fred	CS
George	CS
Henry	EE
Ivy	EE
Jane	EE

S

dept	course
CS	CS101
CS	CS111
CS	CS302
Maths	MA105
Maths	MA203
Music	MU108
Physics	PH113
Physics	PH203

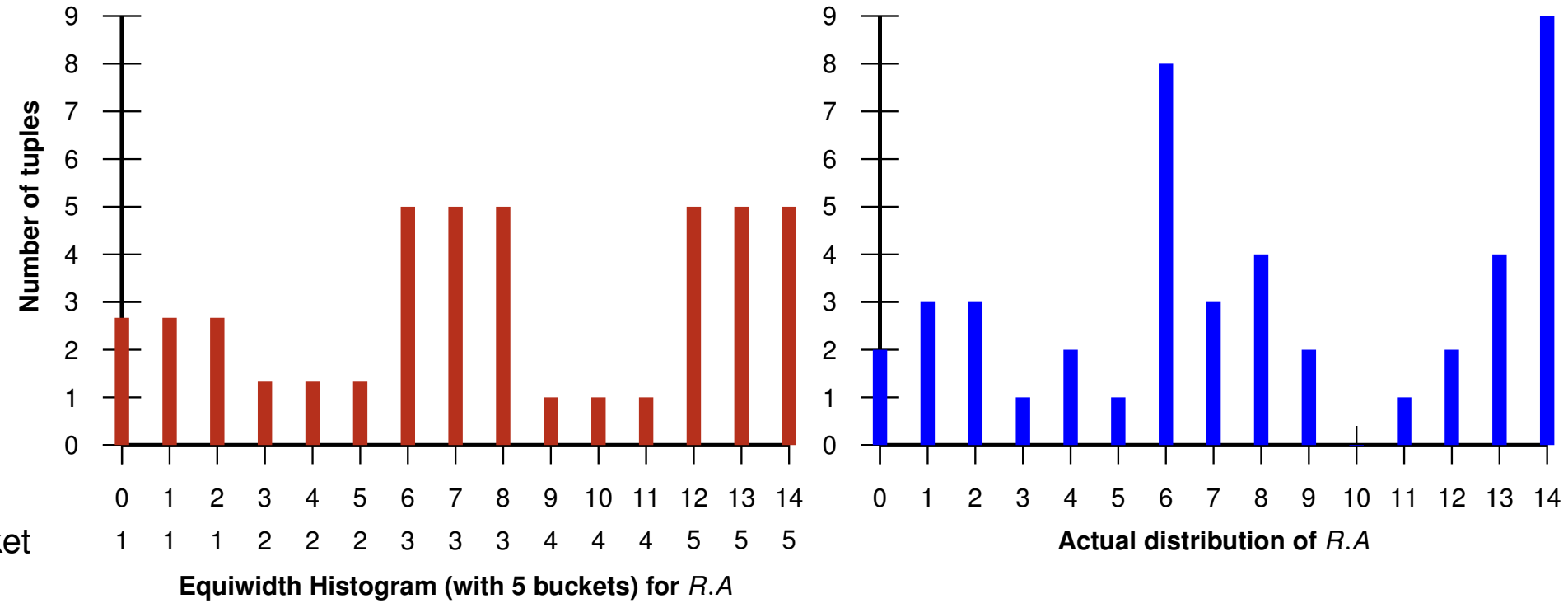
$$||R|| = 10, \quad ||\pi_{dept}(R)|| = 2$$
$$||S|| = 8, \quad ||\pi_{dept}(S)|| = 4$$
$$rf(R.dept = S.dept) \approx \frac{1}{||\pi_{dept}(S)||} = \frac{1}{4}$$
$$||Q|| \approx ||R|| \times \frac{||S||}{||\pi_{dept}(S)||} = 10 \times \frac{8}{4} = 20$$

Estimation using Histograms

- histogram = statistical information maintained by DBMS to estimate data distribution
- Main idea:
  - Partition attribute's domain into sub-ranges called buckets
  - Assume value distribution within each bucket is uniform
- Types of histograms:
  - Equiwidth histograms
    - ★ Each bucket has (almost) equal number of values
  - Equidepth histograms
    - ★ Each bucket has (almost) equal number of tuples
    - ★ Sub-ranges of adjacent buckets might overlap

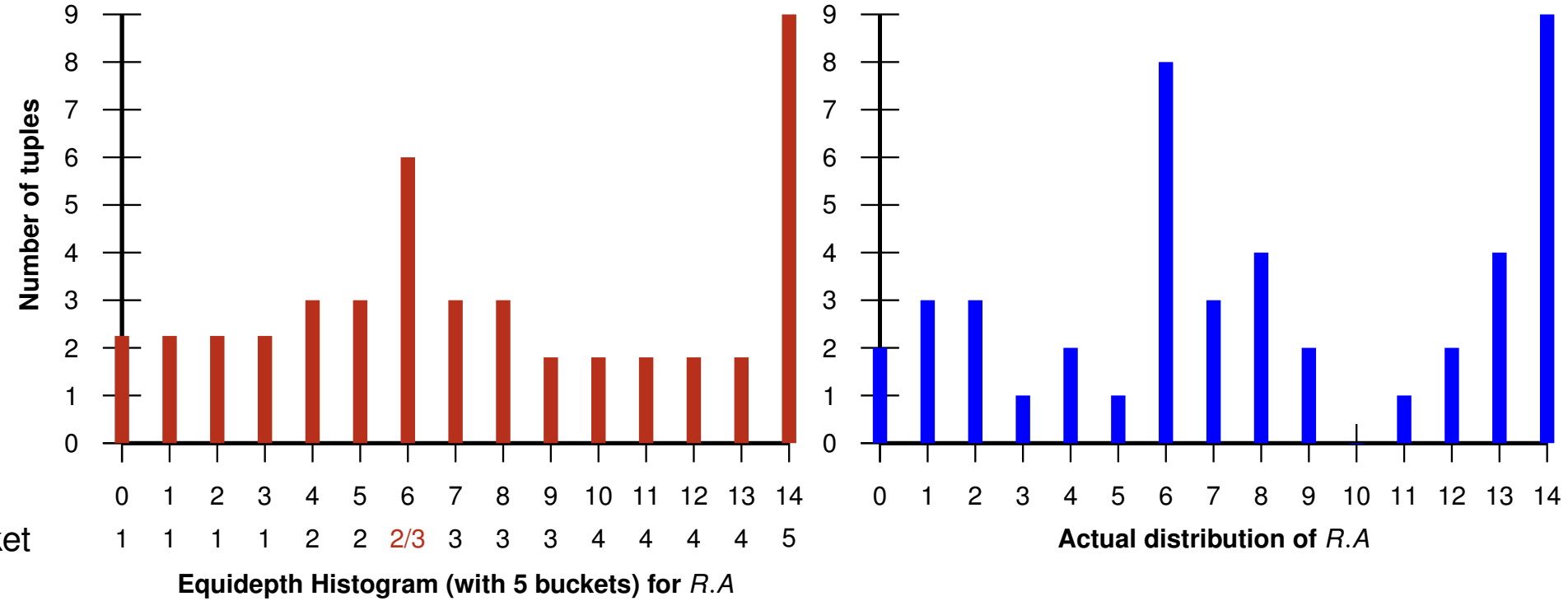
Equiwidth Histograms (with 5 buckets)

Bucket No	Value Range	No. of Tuples	Estimated Number of Tuples per Bucket Value
1	$[0,2]$	2+3+3=8	$8/3 = 2.67$
2	$[3,5]$	1+2+1=4	$4/3 = 1.33$
3	$[6,8]$	8+3+4=15	$15/3 = 5$
4	$[9,11]$	2+0+1=3	$3/3 = 1$
5	$[12,14]$	2+4+9=15	$15/3 = 5$



Equidepth Histograms (with 5 buckets)

Bucket No	Value Range	No. of Tuples
1	[0, 3]	2+3+3+1=9
2	[4, 6]	2+1+6=9
3	[6, 8]	2+3+4=9
4	[9, 13]	2+0+1+2+4=9
5	[14, 14]	9



Estimation with Histograms: Example

Equiwidth Histogram

Bucket No	Value Range	No. of Tuples
1	[0,2]	8
2	[3,5]	4
3	[6,8]	15
4	[9,11]	3
5	[12,14]	15

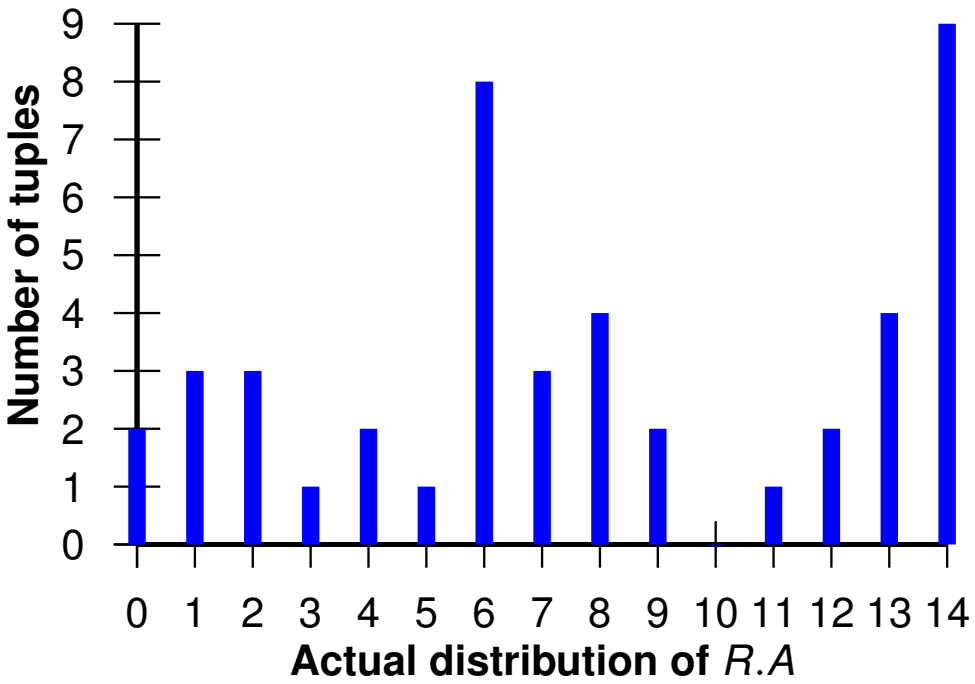
Equidepth Histogram

Bucket No	Value Range	No. of Tuples
1	[0,3]	9
2	[4,6]	9
3	[6,8]	9
4	[9,13]	9
5	[14,14]	9

- Query $Q_1 : \sigma_{A=6}(R)$, $||Q_1|| = 8$
  - Without histogram: $||Q_1|| \approx 45/15 = 3$
  - Equiwidth histogram: $||Q_1|| \approx 15/3 = 5$
  - Equidepth histogram: $||Q_1|| \approx (1/3 \times 9) + (1/3 \times 9) = 6$
- Query $Q_2 : \sigma_{A\in[7,12]}(R)$, $||Q_2|| = 12$
  - Without histogram: $||Q_2|| \approx 45/15 \times 6 = 18$
  - Equiwidth histogram: $||Q_2|| \approx (2/3 \times 15) + 3 + (1/3 \times 15) = 18$
  - Equidepth histogram: $||Q_2|| \approx (2/3 \times 9) + (4/5 \times 9) = 13.2$

Improved Histogram Estimation with MCV

- **MCV** = Most Common Values
- Idea: Separately keep track of the frequencies of the top-k most common values and exclude MCV from histogram's buckets



MCV (k=2)

Value	No. of Tuples
6	8
14	9

Equidepth Histogram (with 3 buckets)

Bucket No	Value Range	No. of Tuples
1	[0, 3]	9
2	[4, 8]	10
3	[9, 14]	9

Improved Histogram Estimation with MCV

MCV (k=2)

Value	No. of Tuples
6	8
14	9

Equidepth Histogram (with 3 buckets)

Bucket No	Value Range	No. of Tuples
1	$[0,3]$	9
2	$[4,8]$	10
3	$[9,14]$	9

- Query $Q_1 : \sigma_{A=6}(R)$, $||Q_1|| = 8$
  - Equidepth histogram: $||Q_1|| \approx (1/3 \times 9) + (1/3 \times 9) = 6$
  - Equidepth histogram with MCV: $||Q_1|| \approx 8$
- Query $Q_2 : \sigma_{A \in [7,12]}(R)$, $||Q_2|| = 12$
  - Equidepth histogram: $||Q_2|| \approx (2/3 \times 9) + (4/5 \times 9) = 13.2$
  - Equidepth histogram with MCV:
    $||Q_2|| \approx (2/(5-1) \times 10) + (4/(6-1) \times 9) = 12.2$