QUERY PROCESSING & OPTIMIZATION

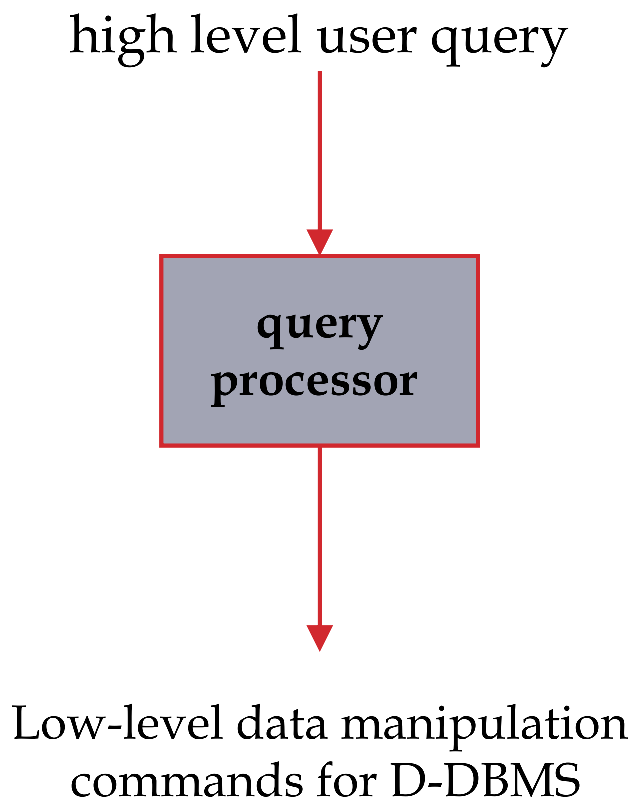
CHAPTER 19 (6/E)

CHAPTER 15 (5/E)

LECTURE OUTLINE

- Query Processing Methodology
- Basic Operations and Their Costs
- Generation of Execution Plans

QUERY PROCESSING IN A DDBMS



SELECTING ALTERNATIVES

```
SELECT    ENAME
FROM      EMP,ASG
WHERE     EMP.ENO = ASG.ENO
AND       ASG.RESP = "Manager"
```

Strategy 1

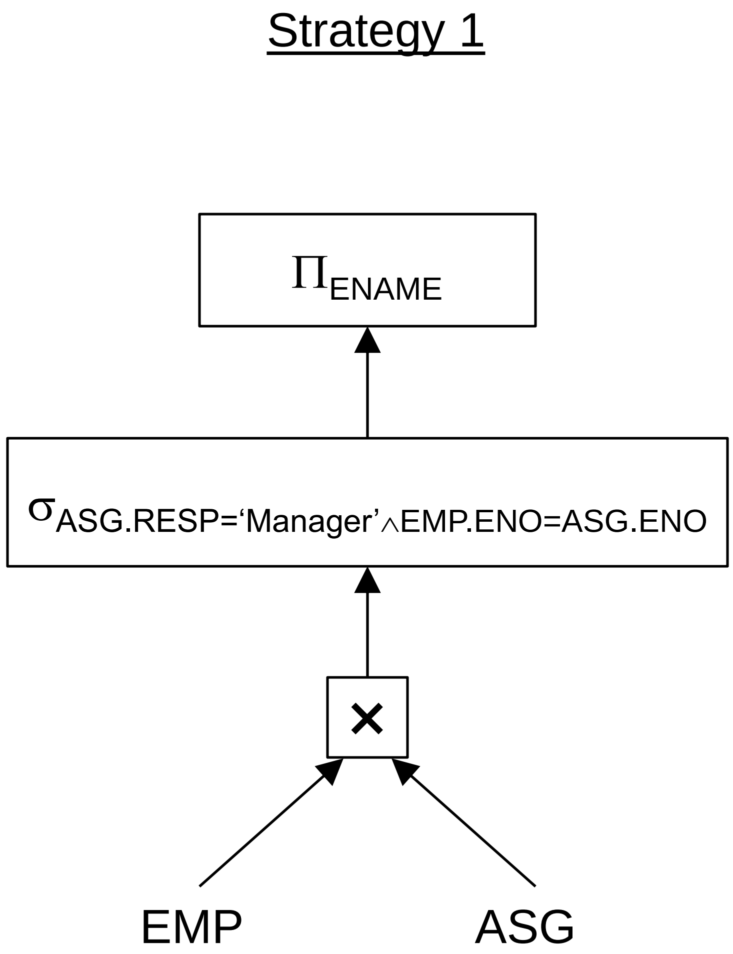
$\Pi_{ENAME}(\sigma_{RESP=\text{“Manager”} \wedge EMP.ENO=ASG.ENO}(EMP \times ASG))$

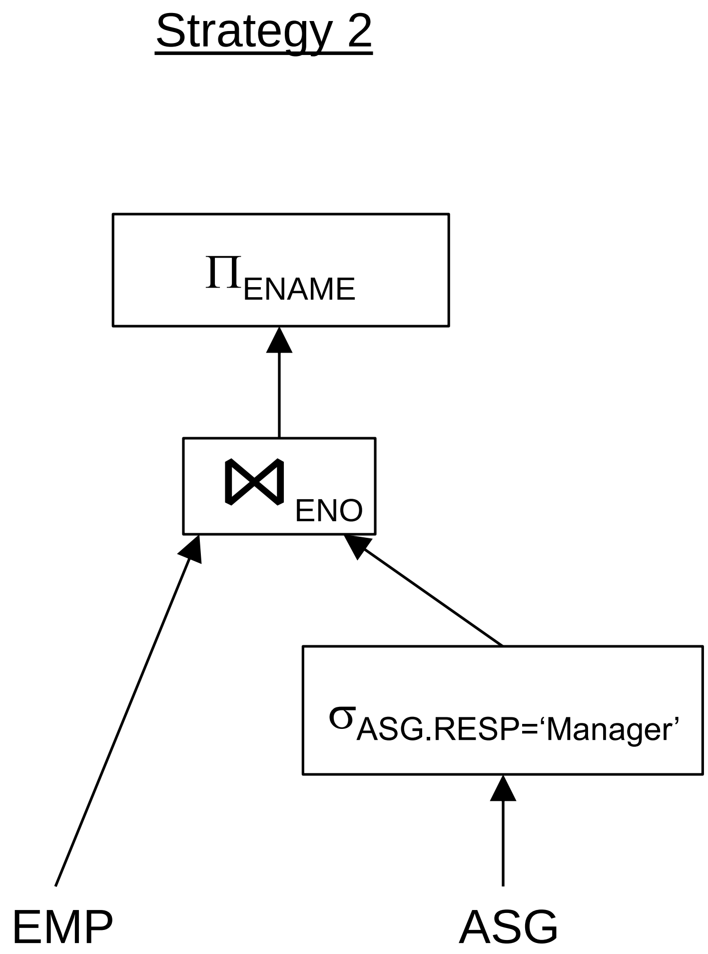
Strategy 2

$\Pi_{ENAME}(EMP \bowtie_{ENO} (\sigma_{RESP=\text{“Manager”}}(ASG))$

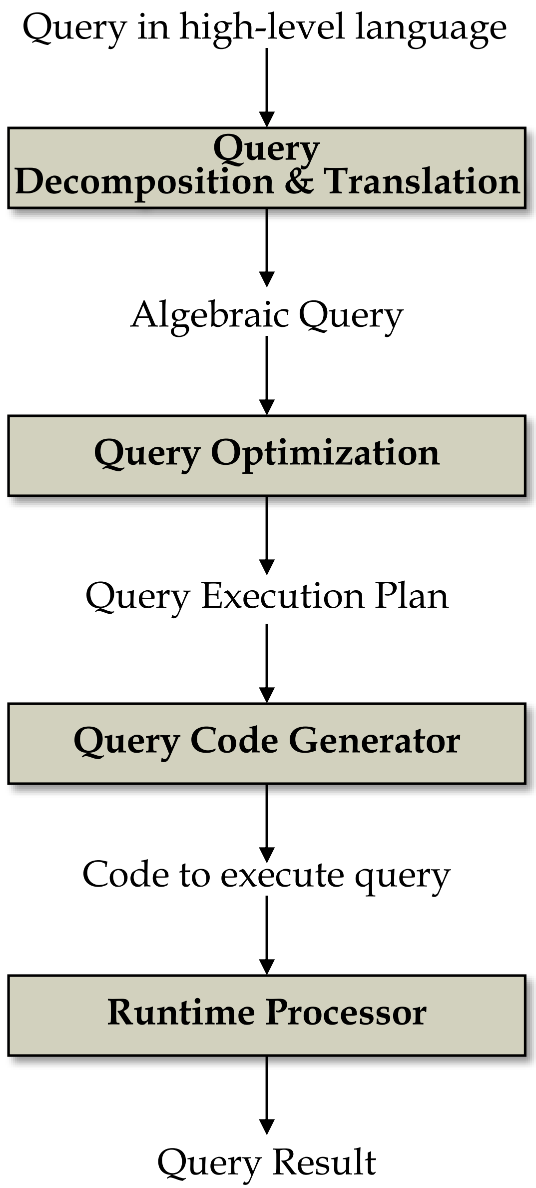
Strategy 2 avoids Cartesian product, so may be “better”

PICTORIALLY





QUERY PROCESSING METHODOLOGY



SQL

- check SQL syntax
- check existence of relations and attributes
- replace views by their definitions
- transform query into an internal form

- generate alternative access plans, i.e., procedure, for processing the query
- select an efficient access plan

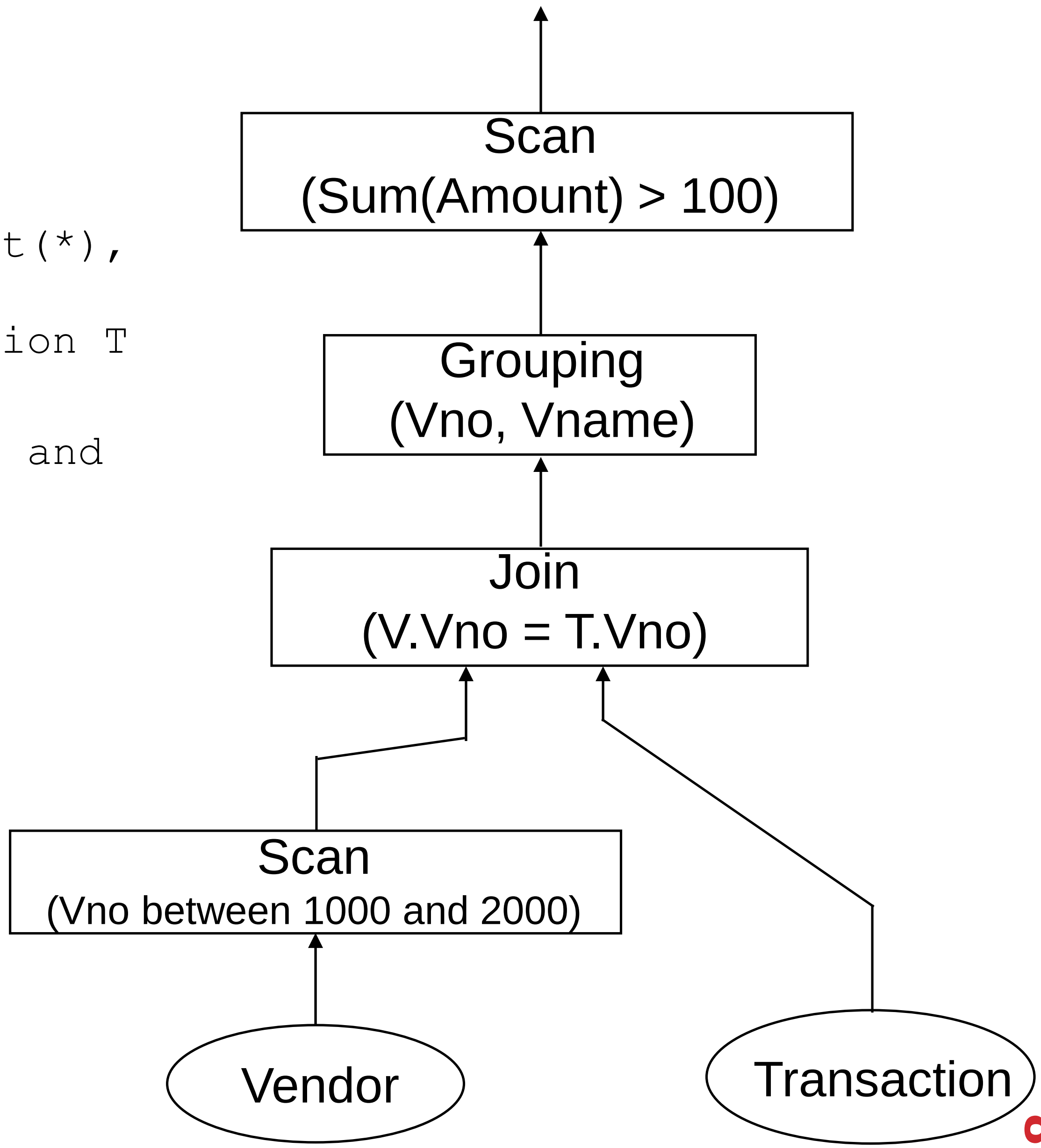
EXAMPLE

```
SELECT  V.Vno, Vname,
count(*), sum(Amount)
FROM    Vendor V,
Transaction T
WHERE   V.Vno = T.Vno
AND     V.Vno between 1000
and 2000
GROUP BY V.Vno, Vname
HAVING  sum(Amount) > 100
```

- Scan the Vendor table, select all tuples where Vno = [1000, 2000], eliminate attributes other than Vno and Vname, and place the result in a temporary relation R_1
- Join the tables R_1 and Transaction, eliminate attributes other than Vno, Vname, and Amount, and place the result in a temporary relation R_2. This may involve:
  - sorting R_1 on Vno
  - sorting Transaction on Vno
  - merging the two sorted relations to produce R_2
- Perform grouping on R_2, and place the result in a temporary relation R_3. This may involve:
  - sorting R_2 on Vno and Vname
  - grouping tuples with identical values of Vno and Vname
  - counting the number of tuples in each group, and adding their Amounts
- Scan R_3, select all tuples with sum(Amount) > 100 to produce the result.

EXAMPLE

```
SELECT   V.Vno, Vname, count(*),
sum(Amount)
FROM     Vendor V, Transaction T
WHERE    V.Vno = T.Vno
AND      V.Vno between 1000 and
2000
GROUP BY V.Vno, Vname
HAVING   sum(Amount) > 100
```



QUERY OPTIMIZATION ISSUES

- Determining the “shape” of the execution plan
  - Order of execution
- Determining which how each “node” in the plan should be executed
  - Operator implementations
- These are interdependent and an optimizer would do both in generating the execution plan

"SHAPE" OF THE EXECUTION PLAN

- Finding query trees that are "equivalent"
  - Produce the same result – provably
- These are based on the transformation (equivalence) rules
- Commutativity of selection
  - $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}R) \Leftrightarrow \sigma_{p_2(A_2)}(\sigma_{p_1(A_1)}R)$
- Commutativity of binary operations
  - $R \times S \Leftrightarrow S \times R$
  - $R \bowtie S \Leftrightarrow S \bowtie R$
  - $R \cup S \Leftrightarrow S \cup R$
  - $R \cap S \Leftrightarrow S \cap R$
- Associativity of binary operations
  - $(R \times S) \times T \Leftrightarrow R \times (S \times T)$
  - $(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$
  - $(R \cup S) \cup T \Leftrightarrow (S \cup R) \cup T$
- Cascading of unary operations
  - $\Pi_{A''}(\Pi_{A'}(R)) \Leftrightarrow \Pi_{A'}(R)$ where $R[A]$ and $A' \subseteq A$, $A'' \subseteq A$ and $A' \subseteq A''$
  - $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \Leftrightarrow \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$

OTHER TRANSFORMATION RULES

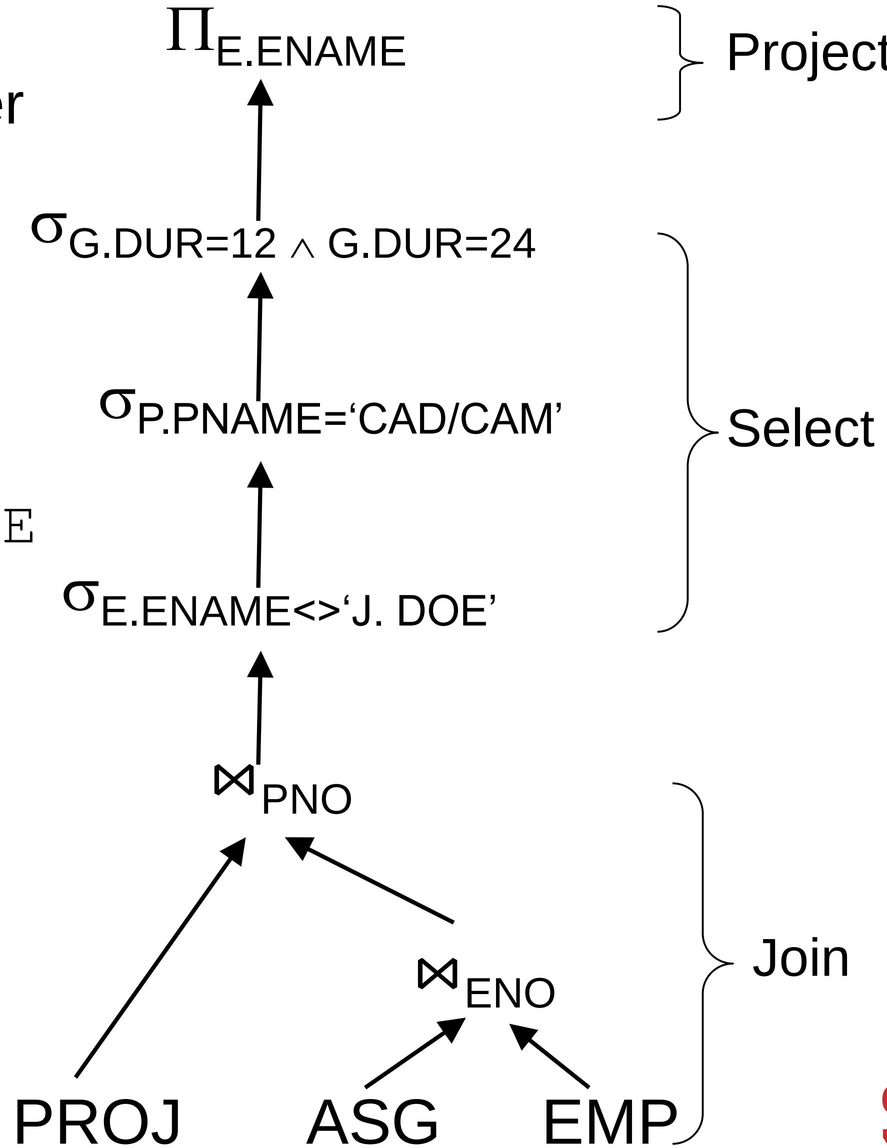
- Commuting selection with projection
  - $\Pi_B (\sigma_{p(A)} R) \Leftrightarrow \sigma_{p(A)}(\Pi_B R)$ (where $B \subseteq A$)
- Commuting selection with binary operations
  - $\sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)} (R)) \times S$ (where A belongs to R only)
  - $\sigma_{p(A_i)}(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow (\sigma_{p(A_i)} (R)) \bowtie_{(A_j,B_k)} S$ (where A_i belongs to R *only*)
  - $\sigma_{p(A_i)}(R \cup S) \Leftrightarrow \sigma_{p(A_i)} (R) \cup \sigma_{p(A_i)} (S)$ (where A_i belongs to R and S)
  - $\sigma_{p(A_i)}(R \cap S) \Leftrightarrow \sigma_{p(A_i)} (R) \cap \sigma_{p(A_i)} (s)$ (where A_i belongs to R and S)
- Commuting projection with binary operations
  - $\Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S)$
  - $\Pi_C(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{(A_j,B_k)} \Pi_{B'}(S)$
  - $\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$
  - $\Pi_C(R \cap S) \Leftrightarrow \Pi_C(R) \cap \Pi_C(S)$

  where $R[A]$ and $S[B]$; $C = A' \cup B'$ where $A' \subseteq A$, $B' \subseteq B$

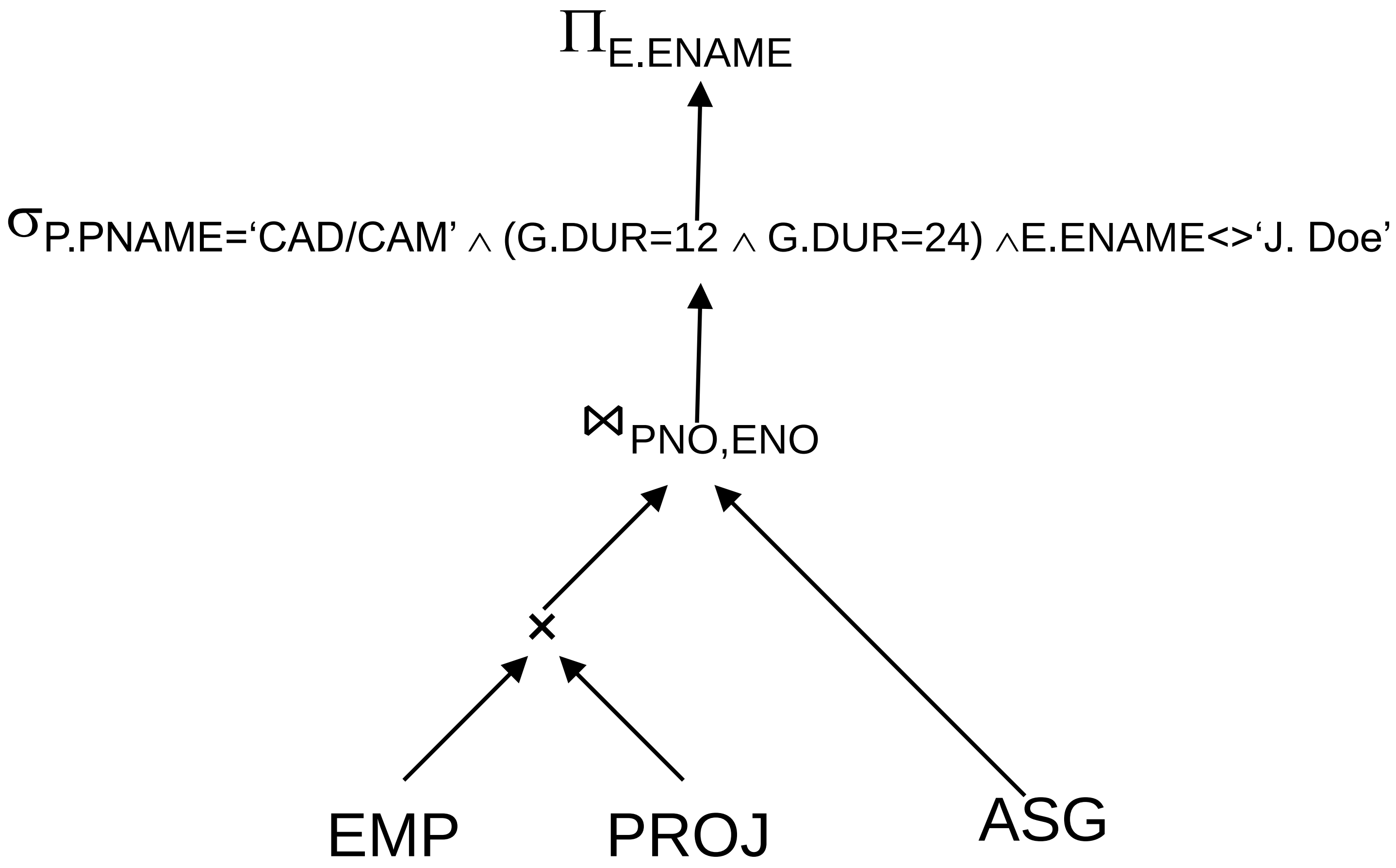
EXAMPLE TRANSFORMATION

Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years.

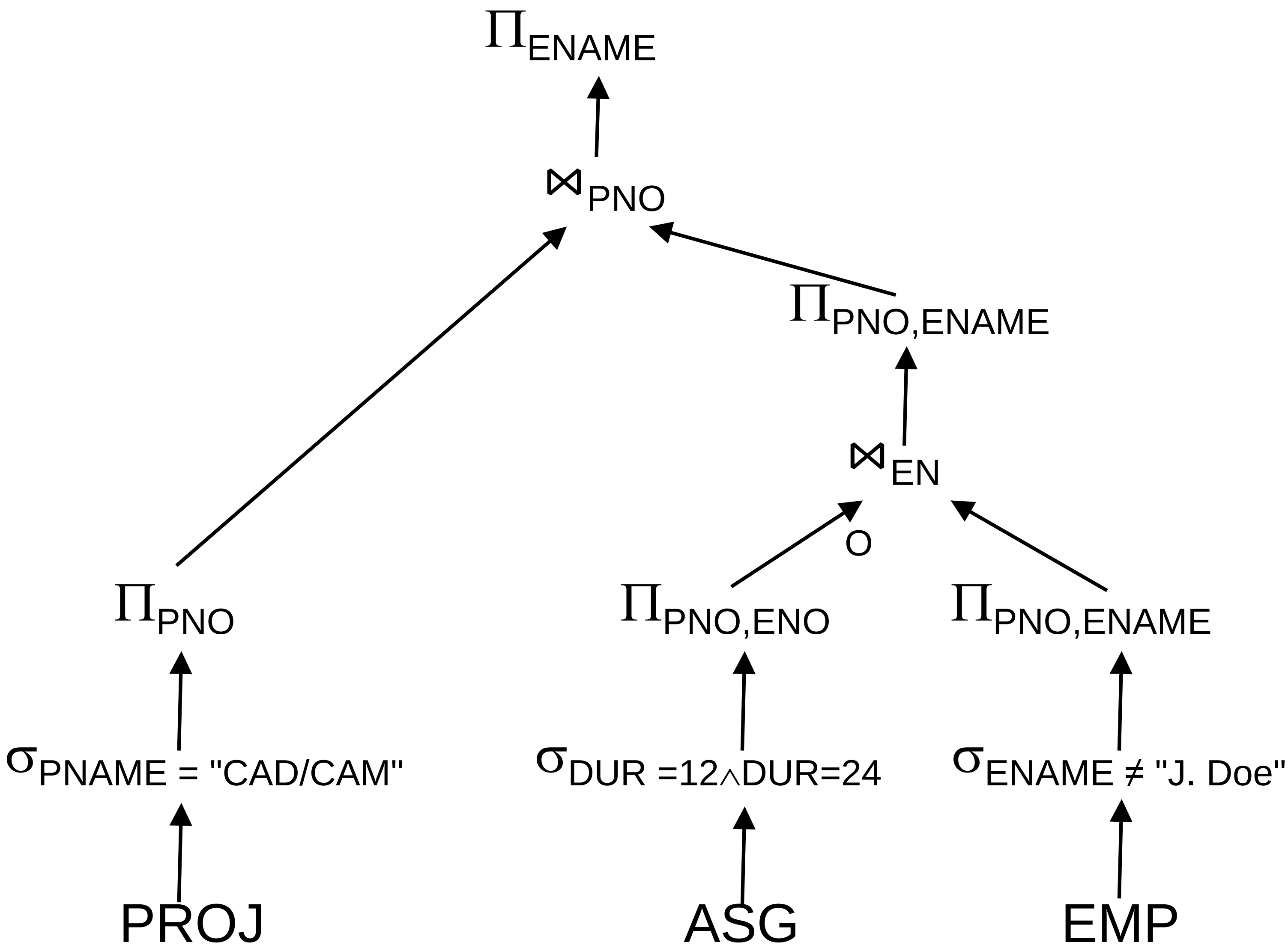
```
SELECT ENAME
FROM   PROJ P, ASG G, EMP E
WHERE  G.ENO=E.ENO
AND    G.PNO=P.PNO
AND    E.ENAME <> 'J. Doe'
AND    P.PNAME='CAD/CAM'
AND    (G.DUR=12 OR
G.DUR=24)
```



EQUIVALENT QUERY

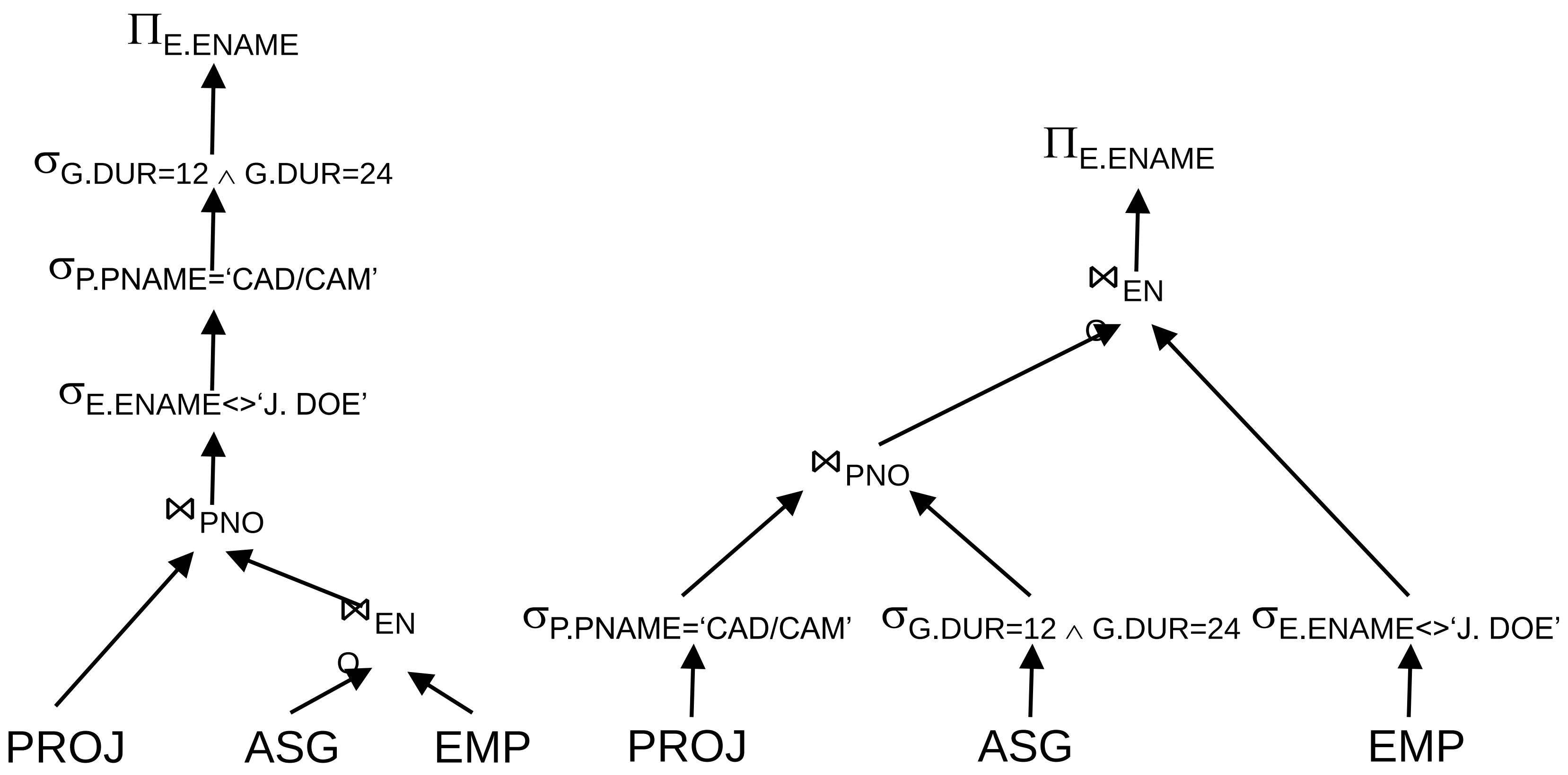


ANOTHER EQUIVALENT QUERY



CLICKER QUESTION #36

- Is the right query plan equivalent to the left query plan?



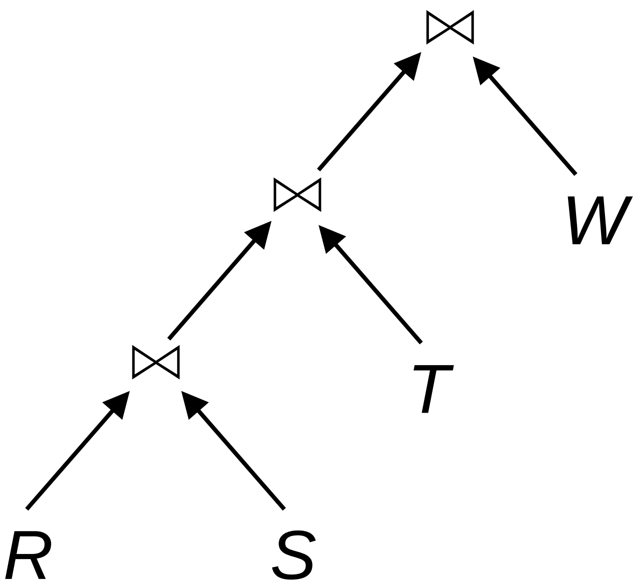
(a) Yes
(b) No

IMPORTANT PROBLEM – JOIN ORDER

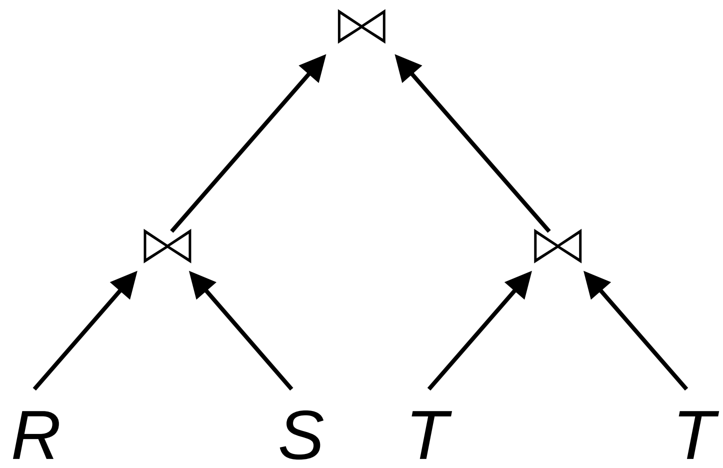
- Assume you have

$$R \bowtie S \bowtie T \bowtie W$$

Linear Join Tree



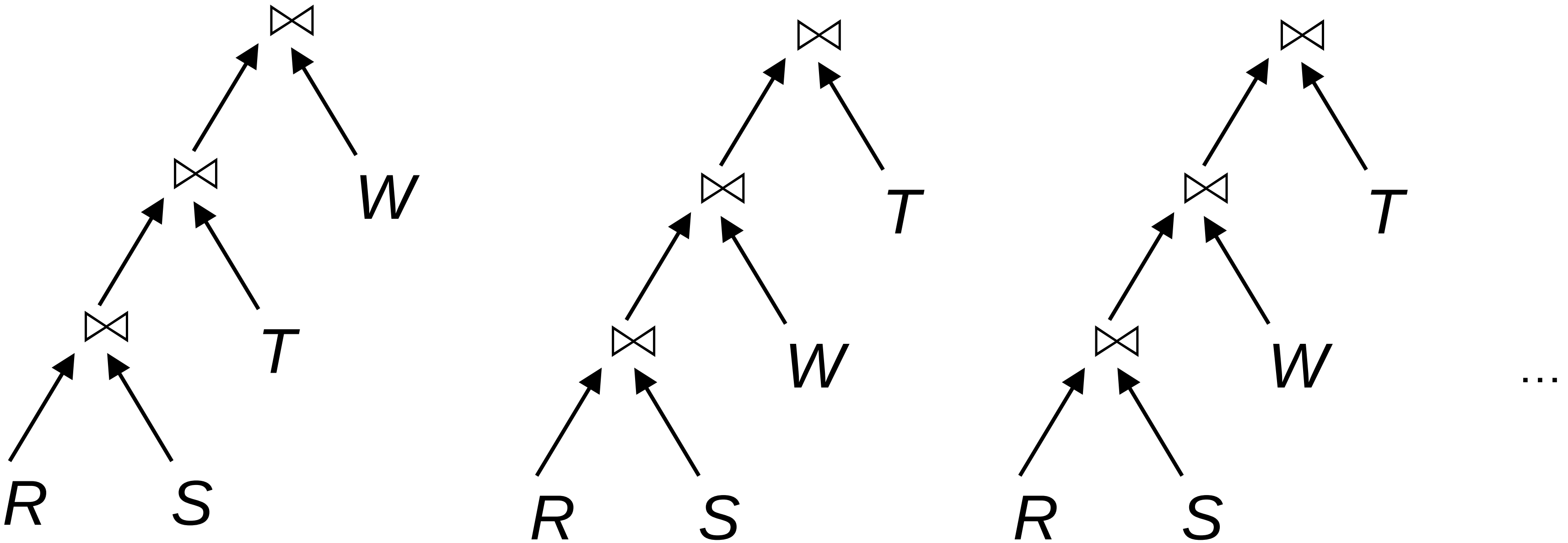
Bushy Join Tree



- Most systems implement linear join trees
  - Left-linear

JOIN ORDERING

- Even with left-linear, how do you know which order?
  - Assume natural join over common attributes



SOME OPERATOR IMPLEMENTATIONS

- Tuple Selection
  - without an index
  - with a clustered index
  - with an unclustered index
  - with multiple indices
- Projection
- Joining
  - nested loop join
  - sort-merge join
  - and others...
- Grouping and Duplicate Elimination
  - by sorting
  - by hashing
- Sorting

EXAMPLE – JOIN ALGORITHMS

```
SELECT C.Cnum, A.Balance
FROM   Customer C, Accounts A
WHERE  C.Cnum = A.Cnum
```

- Nested loop join:

```
for each tuple c in Customer do
    for each tuple a in Accounts do
            if c.Cnum = a.Cnum then
                output c.Cnum,a.Balance
    end
end
```

EXAMPLE – JOIN ALGORITHMS (2)

```
SELECT  C.Cnum, A.Balance
FROM    Customer C, Accounts A
WHERE   C.Cnum = A.Cnum
```

- Index join:

  for each tuple c in Customer do
  use the index to find Accounts tuples *a*
  where *a*.Cnum matches c.Cnum
  if there are any such tuples *a* then
  output c.Cnum, *a*.Balance
  end
  end

- Sort-merge join:

  sort Customer and Accounts on Cnum
  merge the resulting sorted relations

COMPLEXITY OF OPERATORS

- Assume
  - Relations of cardinality *n*
  - Sequential scan

Operation	Complexity
Select Project (without duplicate elimination)	$O(n)$
Project (with duplicate elimination) Group	$O(n * \log n)$
Join Semi-join Division Set Operators	$O(n * \log n)$
Cartesian Product	$O(n^2)$

COST OF PLANS

- Alternative access plans may be compared according to cost.
- The cost of an access plan is the sum of the costs of its component operations.
- There are many possible cost metrics. However, most metrics reflect the amounts of system resources consumed by the access plan. System resources may include:
  - disk block I/O's
  - processing time
  - network bandwidth

LECTURE SUMMARY

- Query processing methodology
- Basic query operations and their costs
- Generation of execution plans