Query Optimization Overview

- Generally, there are many possible access plans for processing a given query.
- The costs of these plans may differ substantially.
- The optimizer must try to choose a reasonable plan quickly.

Pushing Selections Down

- Find the last names of the employees responsible for projects in departments managed by employee number '00020'
- The expression

$$\pi_{\mathbf{LastName}}(\sigma_{\mathbf{MgrNo}='00020'}(E \bowtie_{RespEmp=EmpNo} (D \bowtie P)))$$

  is equivalent to

$$\pi_{\mathbf{LastName}}(E \bowtie_{RespEmp=EmpNo} ((\sigma_{\mathbf{MgrNo}='00020'}(D)) \bowtie P))$$

Selection Pushdown

$$\sigma_{R1.A=x}(R1 \bowtie R2) \equiv (\sigma_{R1.A=x}(R1) \bowtie R2)$$

Conjunctive Selection Conditions

- If a selection condition is not simple, the previous transformation may not apply directly:

```
Select *
From Emp_Act A, Project P
Where A.ProjNo = P.ProjNo
And A.ActNo = 10
And P.DeptNo = 'D01'
```

becomes

$$\sigma_{\mathbf{A.ActNo}=10 \wedge \mathbf{P.DeptNo}='\mathbf{D01}'}(A \bowtie P)$$

- This can be transformed to

$$\sigma_{\mathbf{A.ActNo}=10}(\sigma_{\mathbf{P.DeptNo}='\mathbf{D01}'}(A \bowtie P))$$

Conjunctive Selection Conditions (cont'd)

- The selection pushdown may now be performed:

$$\sigma_{\mathbf{A.ActNo}=10}(A \bowtie \sigma_{\mathbf{DeptNo}='\mathbf{D01}'}(P))$$

and again to produce

$$(\sigma_{\mathbf{ActNo}=10}(A)) \bowtie (\sigma_{\mathbf{DeptNo}='\mathbf{D01}'}(P))$$

Early Projection

- Projection will not reduce the number of tuples in a relation, but it can reduce the size of each tuple by eliminating unnecessary attributes.
- However, this:

$$\pi_{\mathbf{LastName}}(\sigma_{\mathbf{MgrNo}='00020'}(E \bowtie_{RespEmp=EmpNo} (D \bowtie P)))$$

  cannot simply be transformed into this:

$$\sigma_{\mathbf{MgrNo}='00020'}((\pi_{\mathbf{LastName}}(E)) \bowtie_{RespEmp=EmpNo} (D \bowtie P))$$

- In general, projection may be used to eliminate any attributes that will not be used in the result, and that will not be used in subsequent joins or selections.

Reordering Joins

- Joins (and cross products) are associative

$$(R_1 \bowtie R_2) \bowtie R_3 \equiv R_1 \bowtie (R_2 \bowtie R_3)$$

- and commutative

$$R_1 \bowtie R_2 \equiv R_2 \bowtie R_1$$

Join Order

- The join order may have a significant impact on the cost of a plan. Consider the modified plan:

$$\pi_{\mathbf{LastName}}(E \bowtie_{RespEmp=EmpNo} ((\sigma_{\mathbf{MgrNo}='00020'}(D)) \bowtie P))$$

- The joins can be computed, pair-wise, like this:

$$E \bowtie_{RespEmp=EmpNo} (D \bowtie P)$$

- or like this:

$$(E \times D) \bowtie_{RespEmp=EmpNo} P$$

- or like this:

$$(E \bowtie_{RespEmp=EmpNo} P) \bowtie D$$

Cost Models

- An optimizer estimates costs for plans so that it can choose the least expensive plan from a set of alternatives.
- Inputs to the cost model include:
  - the query
  - database statistics
  - resource availability
  - system configuation parameters
- There are many possible cost models. General models attempt to estimate the expected resource consumption of a plan. System resources may include:
  - disk bandwidth
  - processing time
  - network bandwidth

A Simple Cost Model

- Suppose that the cost of an operation is the number of disk block transfers it is expected to require:
- The number of disk block retrievals depends on the number of input tuples and on the tuple blocking factor (tuples per block).
- For a relation R:
  - $|R|$ will represent the number of tuples in R
  - $b(R)$ will represent the blocking factor for R
  - $|R|/b(R)$ is the number of blocks used to store R

B-Tree Cost Example

- Suppose that a clustered, dense B-tree index is defined on attribute A of relation R, with the following properties:
  - there are 4K (4096) bytes per block
  - each tuple of R occupies 256 bytes
  - there are 1,000,000 tuples in R
  - leaf blocks contain tuples, and are 65% full, on average
  - each internal index block holds up to 100 pointers
  - internal index blocks are 100% full
- If a index scan operator in a query needs to retrieve from R all of the tuples for which $A = c$ (for some constant c), how many disk block retrievals will it require?

B-Tree Cost Example (cont'd)

- Each leaf block holds at most $(4096/256) * 0.65 \approx 10$ tuples.
- A total of $1000000/10 = 100000$ leaf blocks are needed to store the tuples of *R*.
- Since each internal index block holds 100 pointers, there will be $100000/100 = 1000$ internal index blocks in the lowest internal level of the B-tree.
- There will be $1000/100 = 10$ internal index blocks in the next level of the B-tree.
- Counting the root, there are 3 levels of internal index nodes in the b-tree.
- Retrieving the matching tuples will require three block retrievals (one for each level of the index), plus as many retrievals as there are leaf blocks containing tuples with `A=c`.

Costs of Alternative Plans

```
Select ProjName, PrStaff
From Project
Where DeptNo = 'D01' And RespEmp = '00100'
And PrStaff > 4
```

- Indices:
  - clustered B-tree index DeptInd on DeptNo
  - non-clustered B-tree index EmpInd on RespEmp
- Assume:
  - $|\mathbf{Project}| = 10000$
  - $b(\mathbf{Project}) = 50$
  - 500 different Employees
  - 100 different Departments
  - index leaves contain tuple IDs (100 each), index depth is 2

Selection Strategy: Use DeptInd

- Assuming *uniform distribution* of tuples over the departments, there will be about $|\mathbf{Project}|/100 = 100$ tuples with DeptNo = 'D01'
- A traversal of DeptInd costs 2. An additional index leaf may need to be retrieved to find all of the matching tuple IDs. Retrieval of the 100 matching tuples adds a cost of $\approx 100/b(\mathbf{Project}) = 2.$ leaf blocks (since the relation is clustered on DeptNo), for a total cost of about 4 or 5 block retrievals.

Selection Strategy: Use EmpInd

- Assuming *uniform distribution* of tuples over employees, there will be about $|\mathbf{Project}|/500 = 20$ tuples for each employee.
- A traversal of EmpInd has a cost of 2. Since there are only 20 matching tuples, no additional index leaf blocks are likely to be required. Since this is not a clustered index, we will make the pessimistic assumption that each matching record is in a separate data block, i.e., 20 blocks will need to be read. The total cost is approximately 22 block retrievals.

Selection Strategy: Scan the Relation

- The relation occupies 10,000/50 = 200 blocks, so 200 blocks will be retrieved.

Database Statistics

- Assume that R is a relation and that $R.a$ is an attribute of R. Commonly maintained statistics include:
  - $|R|$: the cardinality of R, i.e., the number of tuples in R
  - $min(R.a)$: the minimum value for $R.a$
  - $max(R.a)$: the maximum value for $R.a$
  - $distinct(R.a)$: the number of distinct values of $R.a$
- As the database is modified, statistics need to be updated.
- Updates may be incremental, on each update, or may be periodic and controlled by the DBA.

Database Statistics in DB2

```
localhost[114] db2 "runstats on table kmsalem.employee
                    with distribution"
DB20000I  The RUNSTATS command completed successfully.

localhost[116] db2 "select card from sysstat.tables
           where tabname = 'EMPLOYEE'"

CARD
--------------------
                  32

  1 record(s) selected.
```

Database Statistics in DB2 (cont'd)

```
db2 "select colname,colcard,high2key,low2key
   from sysstat.columns where tabname = 'EMPLOYEE'"

BIRTHDATE  30 '1955-04-12' '1926-05-17'
BONUS       8 +0000900.00    +0000400.00
COMM       32 +0003720.00    +0001272.00
EDLEVEL     8 19              14
EMPNO      32 '000330'        '000020'
FIRSTNME   30 'WILLIAM'       'CHRISTINE'
HIREDATE   31 '1980-06-19'    '1949-08-17'
JOB         8 'PRES    '      'CLERK   '
LASTNAME   31 'WALKER'        'BROWN'
MIDINIT    20 'W'             'A'
PHONENO    32 '9001'          '0942'
SALARY     32 +0046500.00    +0015900.00
SEX         2 'M'             'F'
WORKDEPT    8 'E11'           'B01'
```

Database Statistics in DB2 (cont'd)

```
db2 "select seqno,colvalue,valcount from
     sysstat.coldist where tabname = 'EMPLOYEE'
     and colname = 'SALARY'"

SEQNO  COLVALUE       VALCOUNT
1      +0015340.00    1
2      +0015900.00    2
3      +0017750.00    4
4      +0018270.00    5
5      +0019950.00    7
6      +0021340.00    9
7      +0022180.00   10
...
16     +0036170.00   26
17     +0038250.00   27
18     +0040175.00   29
19     +0046500.00   31
20     +0052750.00   32
```

Selectivity and Join Size Estimation

- Because
  - the cost of an operation depends on the size of its input, and
  - the output of one operation often becomes the input of another, and

  database systems need some way to estimate the size of the result of an operation, such as a selection or a join.
- These estimates are often made using (potentially unrealistic) assumptions and/or about attribute value statistics. In particular:

  Uniformity : all possible values of attribute A are equally likely to occur in a relation

  Independence : the likelihood that a tuple will have $A_1 = a_1$ does not depend on what value it has for attribute A_2.

Selectivity Estimation

- The selectivity of a selection $\sigma_{\text{condition}}(R)$ is defined as:

$$\text{sel}(\sigma_{\text{condition}}(R)) = \frac{|\sigma_{\text{condition}}(R)|}{|R|}$$

- If the DBMS has no other information, it may use selectivity estimators based on the available statistics. For example:
  - $\text{sel}(\sigma_{a=c}(R)) \approx \frac{1}{distinct(R.a)}$
  - $\text{sel}(\sigma_{a\leq c}(R)) \approx \frac{c-min(R.a)}{max(R.a)-min(R.a)}$
  - $\text{sel}(\sigma_{a\geq c}(R)) \approx \frac{max(R.a)-c}{max(R.a)-min(R.a)}$
  - $\text{sel}(\sigma_{\text{cond1}\wedge\text{cond2}}(R)) \approx \text{sel}(\sigma_{\text{cond1}}(R))\text{sel}(\sigma_{\text{cond2}}(R))$

Join Size Estimation

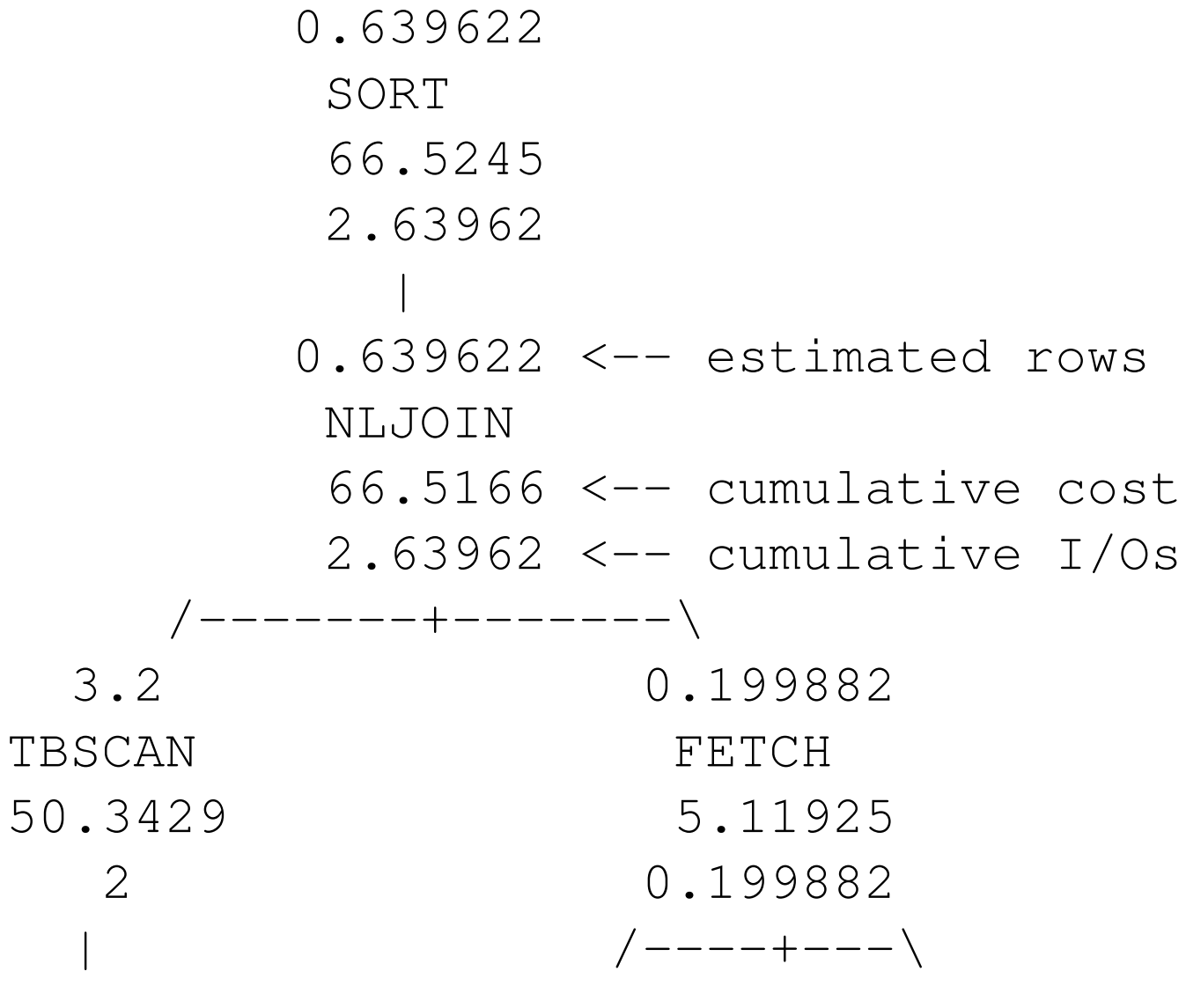
- The number of tuples in the result of a join depends on the number of values that match in the join attributes of the two tables. Some examples:
  - Consider $P \bowtie_{(RespEmp=EmpNo)} E$). Since EmpNo is the key of E, the join size may be estimated as $|P|$. Many joins are foreign key joins, like this one.
  - Consider $R_1 \bowtie_{R_1.a=R_2.b} R_2$ that is not a foreign key join. Assuming that $distinct(R_1.a) \leq distinct(R_2.b)$, that each distinct value of $R_1.a$ matches a value in $R_2.b$, and that tuples in R_2 are uniformly distributed over the distinct values of $R_2.b$, the join size might be estimated as

$$|R_1 \bowtie_{R_1.a=R_2.b} R_2| \approx |R_1| \frac{|R_2|}{distinct(R_2.b)}$$

Costing Info in DB2

```
Parallelism:  None
CPU Speed:  2.664809e-06
Comm Speed:  0
Buffer Pool size:  1000
Sort Heap size:  256
Database Heap size:  1200
Average Applications:  1
```

Costing Info in DB2 (cont'd)



Cost-Based Optimization

- find a plan with low cost
- number of possible plans grows quickly with the number of relations involved in the query
  - $n!$ left-deep join orders over for n relation query
- dynamic programming approach (from System R):
  - bottom-up determination of join order
  - prune high-cost alternatives
  - retain alternatives with "interesting" features

Dynamic Programming Optimization Example

```
Select LastName, EmpTime, Projname
From Employee E, Emp_Act A, Project P
Where E.Empno = A.Empno And A.ProjNo = P.ProjNo
And A.EmStDate like '82%' And A.EmpTime >= 0.5
```

Available Access Methods:

EI1:	clustered Btree on E.Empno	relevant
EI2:	table scan of E	relevant
PI1:	clustered Btree on P.Projno	relevant
PI2:	table scan of P	relevant
AI1:	clustered Btree on A.(Actno,Projno)	not relevant
AI2:	unclustered Btree on A.EmStDate	relevant
AI3:	unclustered Btree on A.Empno	relevant
AI4:	table scan of A	relevant

Dynamic Programming Optimization Example (cont'd)

- first iteration: choose the best plan(s) for generating the required tuples from each single relation
  - $\sigma_{EmStDate \text{ like } '82\%' \wedge EmpTime \geq 0.5}(A)$
  - E
  - P
- to choose plans for generating tuples from a relation, consider the available access methods for that relation

Dynamic Programming Optimization Example (cont'd)

Choose plan(s) for $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A)$

1. Generate possible plans:
   - **A1**: table scan (AI4), then $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}$
   - **A2**: index scan (AI2) tuples with EmStDate like '82%', then $\sigma_{EmpTime \geq 0.5}$
   - **A3**: index scan (AI3) all tuples, then $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}$
2. Estimate costs of possible plans:
   - suppose that $cost(A2) < cost(A1) < cost(A3)$.
3. Prune plans:
   - **A1**: PRUNE!
   - **A2**: keep (lowest cost)
   - **A3**: keep (more costly than A2, but generates tuples in an *interesting* order - Empno order)

Dynamic Programming Optimization Example (cont'd)

Choose plan(s) for E

1. Generate possible plans:
   - **E1**: table scan (EI2)
   - **E2**: index scan (EI1)
2. Estimate cost of possible plans:
   - suppose that $cost(E1) < cost(E2)$.
3. Prune plans:
   - **E1**: keep (lowest cost)
   - **E2**: keep (more costly than E1, but generates tuples in Empno order)

Dynamic Programming Optimization Example (cont'd)

Choose plan(s) for P

1. Generate possible plans:
   - **P1**: table scan (PI2)
   - **P2**: index scan (PI1)
2. Estimate cost of possible plans:
   - suppose that $cost(P1) < cost(P2)$.
3. Prune plans:
   - **P1**: keep (lowest cost)
   - **P2**: keep (more costly than P1, but generates tuples in Projno order)

Dynamic Programming Optimization Example (cont'd)

- second iteration: choose the best plan(s) for generating the required tuples from each pair of relations
  - $\sigma_{EmStDate \text{ like } '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie E$
  - $\sigma_{EmStDate \text{ like } '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie P$
  - $E \bowtie P$
- to build plans for generating tuples from n relations:
  - choose a join type
  - choose an unpruned plan for $n - 1$ relations as the outer input to the join
  - choose an access method for the remaining relation as the inner input to the join

Dynamic Programming Optimization Example (cont'd)

Choose plan(s) for $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie E$

1. Generate possible plans (not all shown)
   - **AE1**: nested loop join A2 and EI2
   - **AE2**: index nested loop join A2 and EI1
   - **AE3**: merge join sort(A2) and EI1 (Empno order)
   - **AE4**: nested loop join A3 and EI1 (Empno order)
   - **AE5**: merge join A3 and EI1 (Empno order)
   - **EA1**: nested loop join E1 and AI2
   - **EA2**: merge join E2 and sort(AI4) (Empno order)
   - **EA3**: index nested loop join E2 and AI3 (Empno order)
2. Estimate costs of possible plans and prune:
   - suppose that *cost*(*EA*2) is lowest among plans generating Empno order and *cost*(*AE*1) is the cheapest overall. Prune all but EA2 and AE1.

Dynamic Programming Optimization Example (cont'd)

Choose plan(s) for $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie P$

1. Generate possible plans (not all shown)
   - **AP1**: nested loop join A3 and sort(PI2) (Empno order)
   - **AP2**: merge join sort(A2) and PI1 (Projno order)
   - **AP3**: index nested loop join A3 and PI1 (Empno order)
   - **AP4**: index nested loop join A2 and PI1
   - **PA1**: index nested loop join P2 and AI3 (Projno order)
2. Estimate costs of possible plans and prune:
   - suppose that AP2 is cheapest overall, and that $cost(AP3) < cost(AP1)$
   - keep only AP2 (cheapest) and AP3 (more expensive, but Empno order is interesting)

Dynamic Programming Optimization Example (cont'd)

Choose plan(s) for $P \bowtie E$

1. Generate possible plans (not all shown)
   - **PE1**: nested loop join of P2 and EI2 (Projno order)
   - **EP1**: nested loop join of E2 and PI2 (Empno order)
2. Estimate costs of possible plans and prune:
   - suppose that PE2 is cheapest overall, and EP1 is the cheapest plan producing Empno order
   - keep PE2 (cheapest) and PE1 (interesting order)

Dynamic Programming Optimization Example (cont'd)

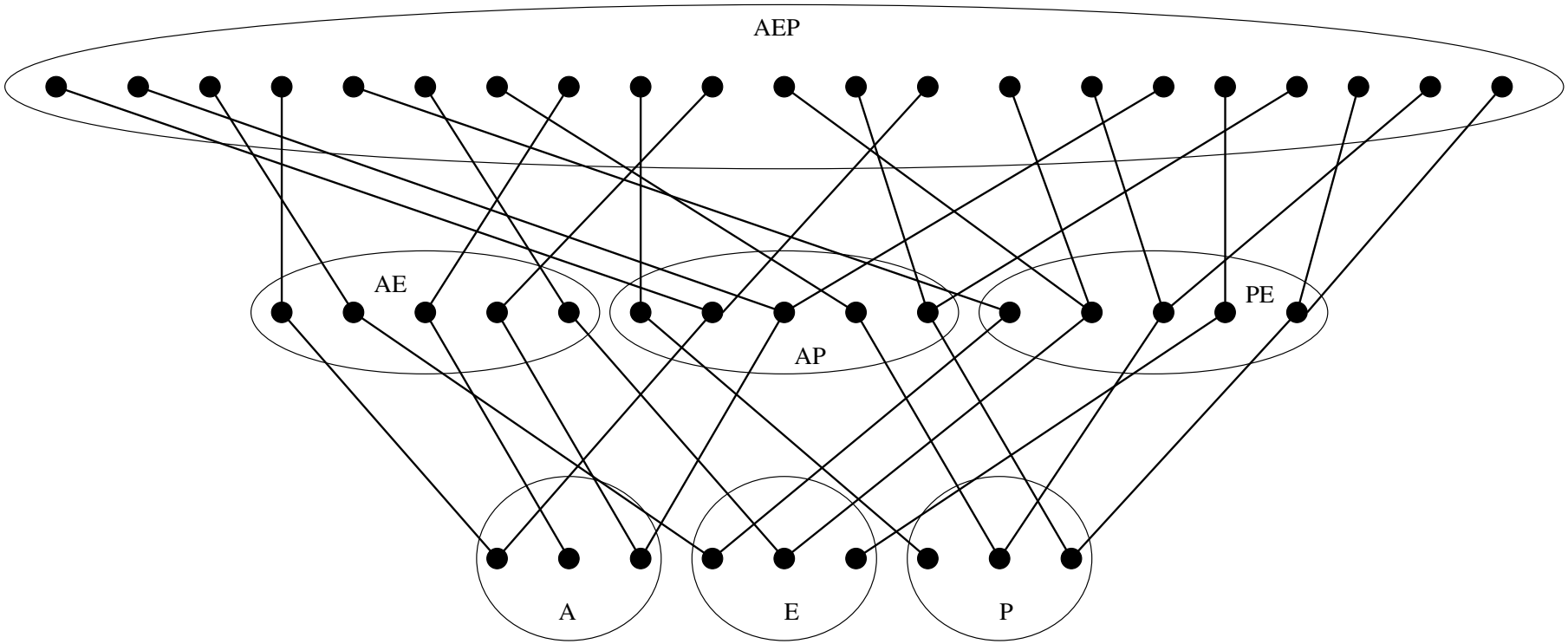
- third iteration: choose the best plan(s) for generating the required tuples from all three relations
  - $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie E \bowtie P$
- consider
  - the best plans for $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie E$ combined with an access method for P
  - the best plans for $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie P$ combined with an access method for E
  - the best plans for $E \bowtie P$, combined with an access method for A

Dynamic Programming Optimization Example (cont'd)

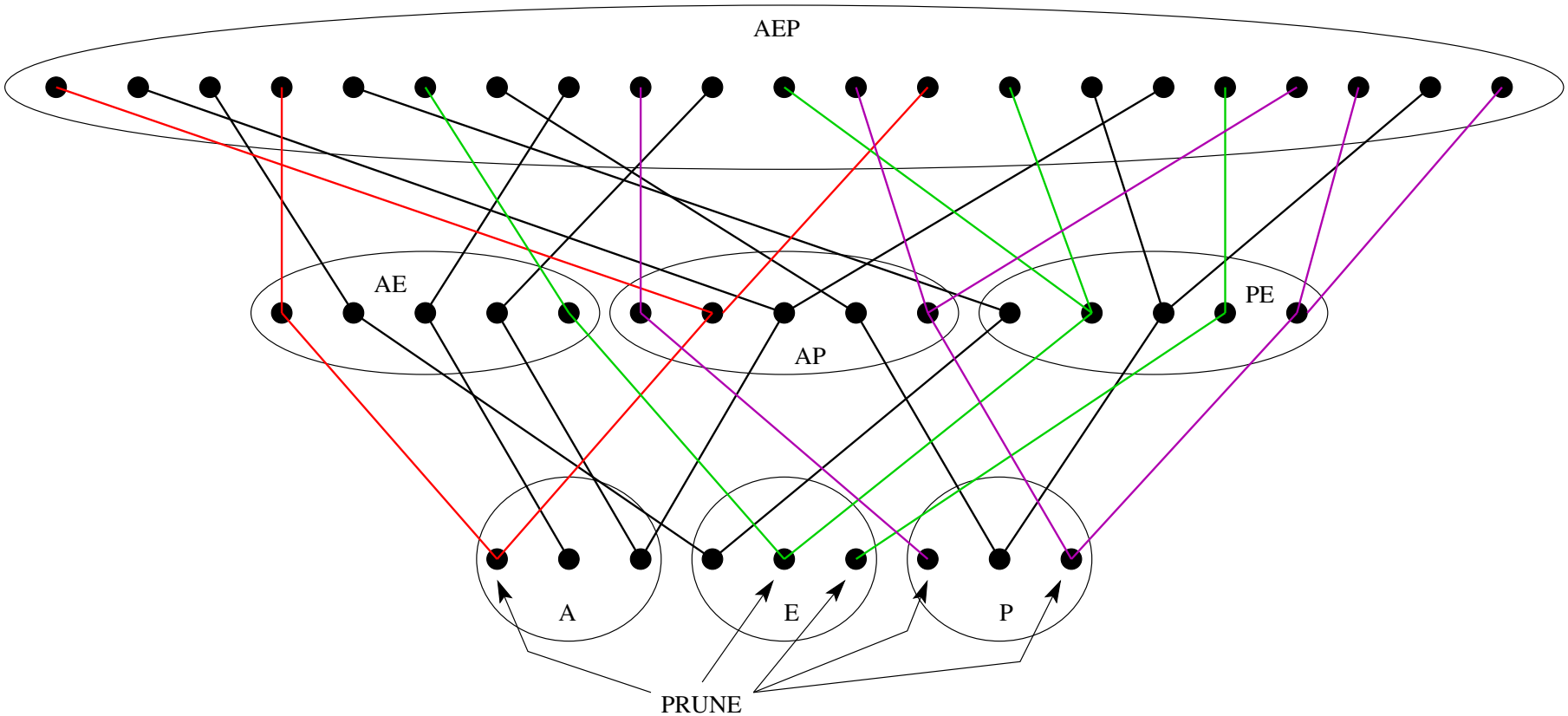
Choose plan(s) for $\sigma_{EmStDate\ \mathbf{like}\ '82\%' \wedge EmpTime \geq 0.5}(A) \bowtie E \bowtie P$

1. Generate possible plans (not all shown)
   - **AEP1**: index nested loop join AE1 and PI1
   - **AEP2**: nested loop join AE1 and PI2
   - **APE1**: index nested loop join AP2 and EI1
   - **APE2**: merge join AP3 and EI1 (Empno order)
   - **PEA1**: index nested loop join PE1 and AI3 (Projno order)
   - **PEA2**: merge join PE2 and AI3 (Empno order)
2. Estimate costs of possible plans and prune:
   - suppose that AEP1 is cheapest
   - since there are no more relations to be joined and there are no GROUP BY or ORDER BY clauses in the query, there is no need to further preserve interesting orders.
   - prune all plans except the winner: AEP1

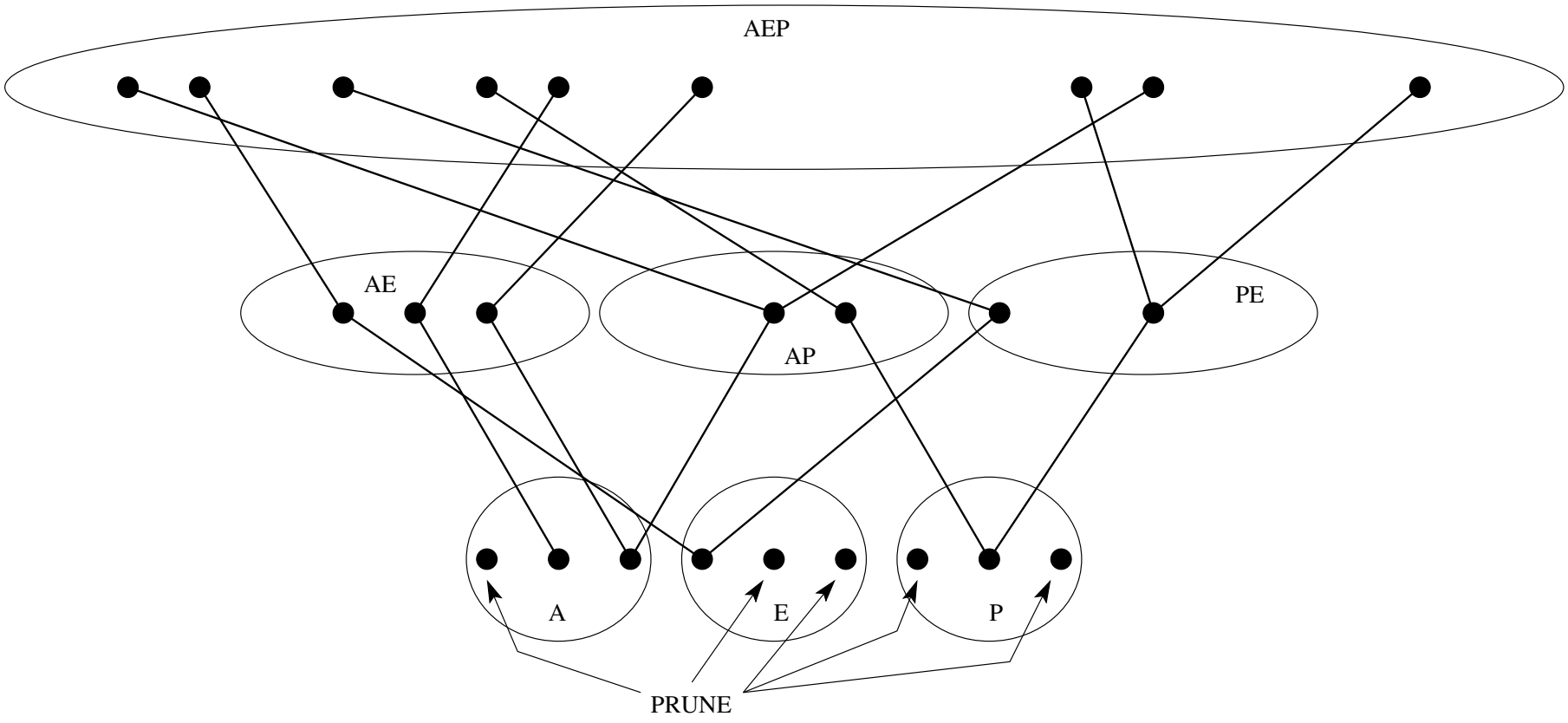
Effects of Pruning



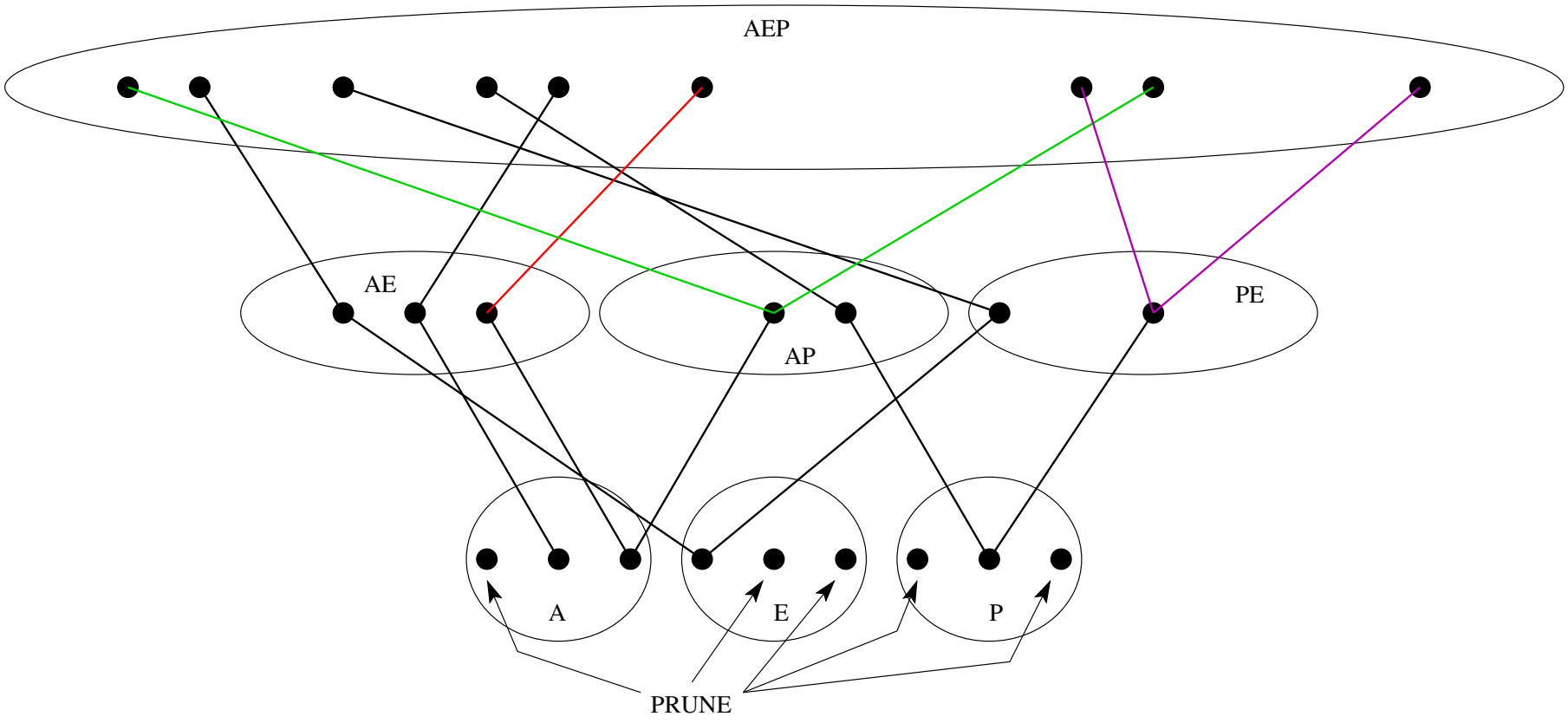
Effects of Pruning (cont'd)



Effects of Pruning (cont'd)



Effects of Pruning (cont'd)



Effects of Pruning (cont'd)