COP 4710: Database Systems
Fall 2012

Query Processing (Chapter 12)

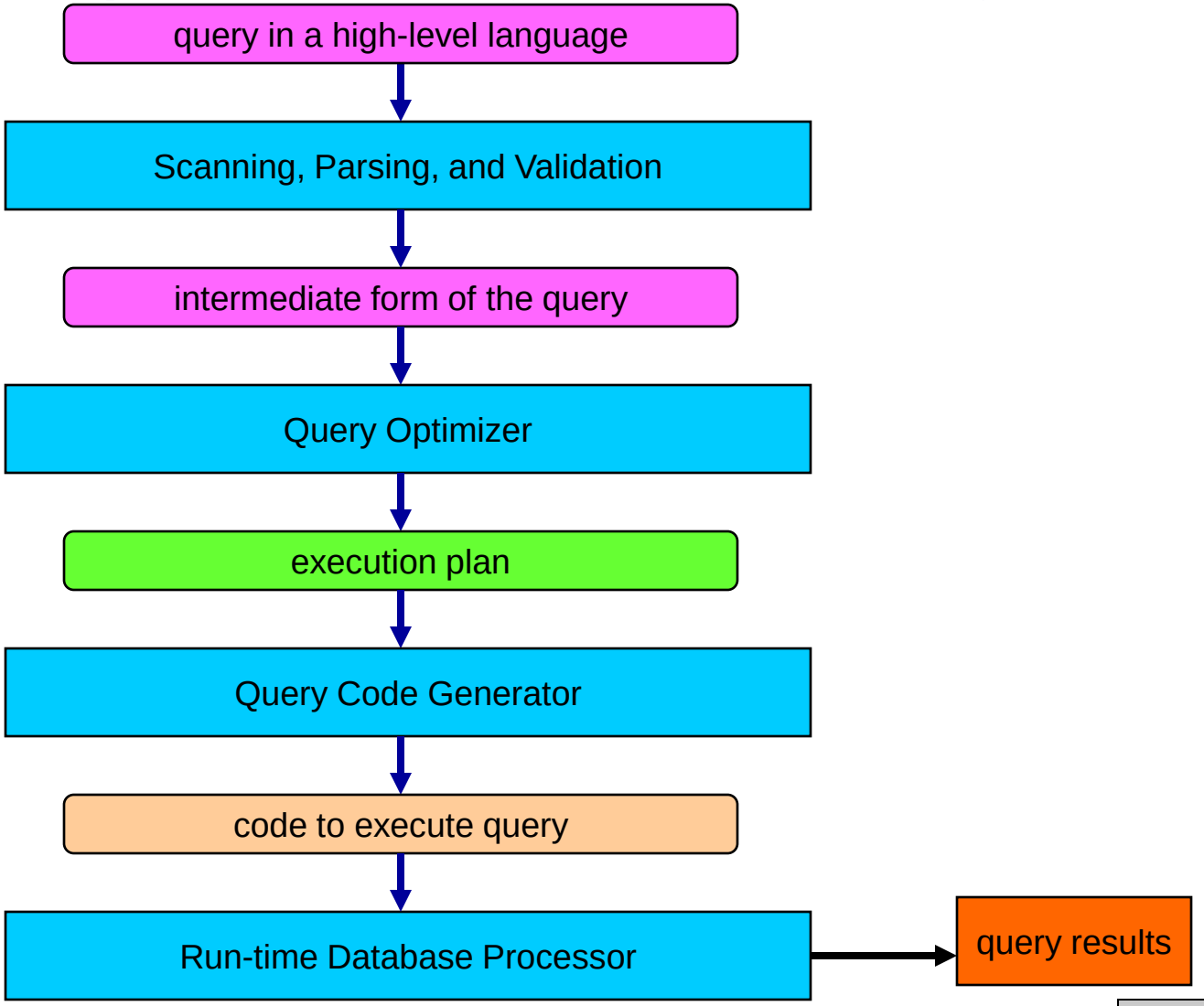
Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4710/fall2012

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Query Processing and Optimization

- A query expressed in a high-level language like SQL must first be scanned, parsed, and validated.
- Once the above steps are completed, an internal representation of the query is created. Typically this is either a tree or graph structure, called a query tree or query graph.
- Using the query tree or query graph the RDBMS must devise an execution strategy for retrieving the results from the internal files.
- For all but the most simple queries, several different execution strategies are possible. The process of choosing a suitable execution strategy is called query optimization.

The Steps in Query Processing



Query Optimization

- The term query optimization may be somewhat misleading. Typically, no attempt is made to achieve an optimal query execution strategy overall – merely a *reasonably efficient strategy*.
- Finding an optimal strategy is usually too time consuming except for very simple queries and for these it usually doesn't matter.
- Queries may be "hand-tuned" for optimal performance, but this is rare.
- Each RDBMS will typically maintain a number of general database access algorithms that implement basic relational operations such as select and join. Hybrid combinations of relational operations also typically exist.

Query Optimization (cont.)

- Only execution strategies that can be implemented by the DBMS access algorithms and which apply to the particular database in question can be considered by the query optimizer.

- There are two basic techniques that can be applied to query optimization:

  1. Heuristic rules: these are rules that will typically reorder the operations in the query tree for a particular execution strategy.

  2. Systematical estimation: the cost of various execution strategies are systematically estimated and the plan with the least "cost" is chosen. What constitutes cost can also vary. It could be a monetary cost, or it could be a cost in terms of time or other factors.

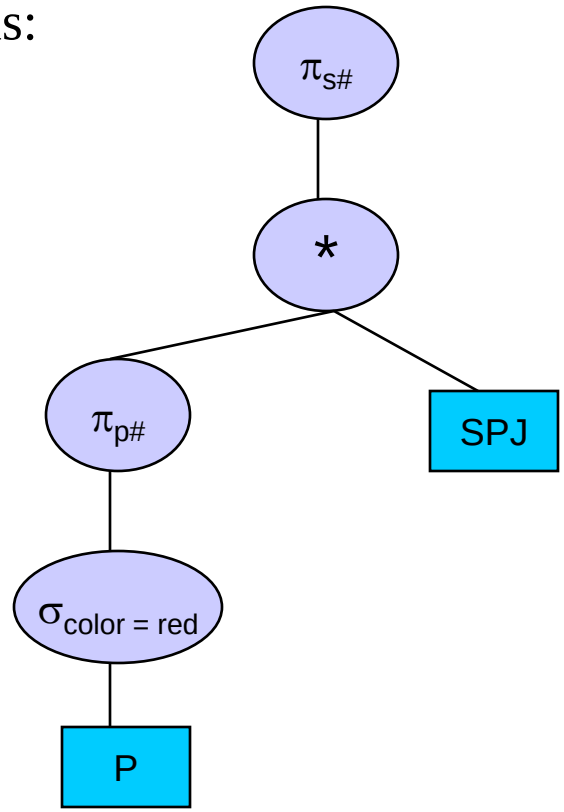
- Most query optimizers use a combination of both techniques.

Query Trees

- A query tree is a tree representation of a relational algebra expression which represents the operand relations as leaf nodes and the relational algebra operators as internal nodes.
- Execution of the query tree consists of executing and internal node operation whenever its operands are available and then replacing that internal node by the virtual relation which results from the execution of the operation.
- Execution terminates when the root node is executed and the resulting relation is produced.
- This technique is similar to what many compilers do for 3GLs like C.

Query Tree Example

- Consider the query: "list the supplier numbers for suppliers who supply a red part." (this one should be really familiar by now!!)
- In relational algebra we have: $\pi_{S\#}(spj * (\pi_{p\#}(\sigma_{color='red'}(P))))$
- The corresponding query tree is:



Query Trees

- There are usually several different ways to generate a relational algebra expression for a query. This should be quite obvious by now after doing the homework for the course.
- Since several different relational algebra expressions are possible for a given query, so too are there multiple query trees possible for the same query.
- The next page shows several different relational algebra expressions for a given query and the following couple of pages illustrate the possible query trees.

Query Expressions

- Query: list the names of those suppliers who ship both part numbers P1 and P2.

SQL Version #1:

```
Select name
From Suppliers
Where s# In (Select s#
                From Shipments
                Where p# = "P1")
  And  s# In (Select s#
                From Shipments
                 Where p# = "P2")
```

SQL Version #2:

```
Select name
From Suppliers
Where Exists (Select s#
                From Shipments
                Where p# = "P1" and
              Suppliers.s# = Shipments.s#)
  And  Exists (Select s#
                From Shipments
                 Where p# = "P2" and
              Suppliers.s# = Shipments.s#)
```


Query Expressions

- Query: list the names of those suppliers who ship both part numbers P1 and P2.

SQL Version #3:

```
Select name
From Suppliers
Where s# In (Select Shipments.s#
                From Shipments Cross Join Shipments As SPJ
                Where Shipments.s# = SPJ.s#
                        And Shipments.p# = "P1"
                        And SPJ.p# = "P2")
```


Query Expressions

- Query: list the names of those suppliers who ship both part numbers P1 and P2.

exp #1: $(\pi_{name}(s * (\pi_{s\#}(\sigma_{p\#=P1}(spj))))) \cap (\pi_{name}(s * (\pi_{s\#}(\sigma_{p\#=P2}(spj)))))$

exp #2: $\pi_{name}(s * ((\pi_{s\#}(\sigma_{p\#=P1}(spj))) \cap (\pi_{s\#}(\sigma_{p\#=P2}(spj)))))$

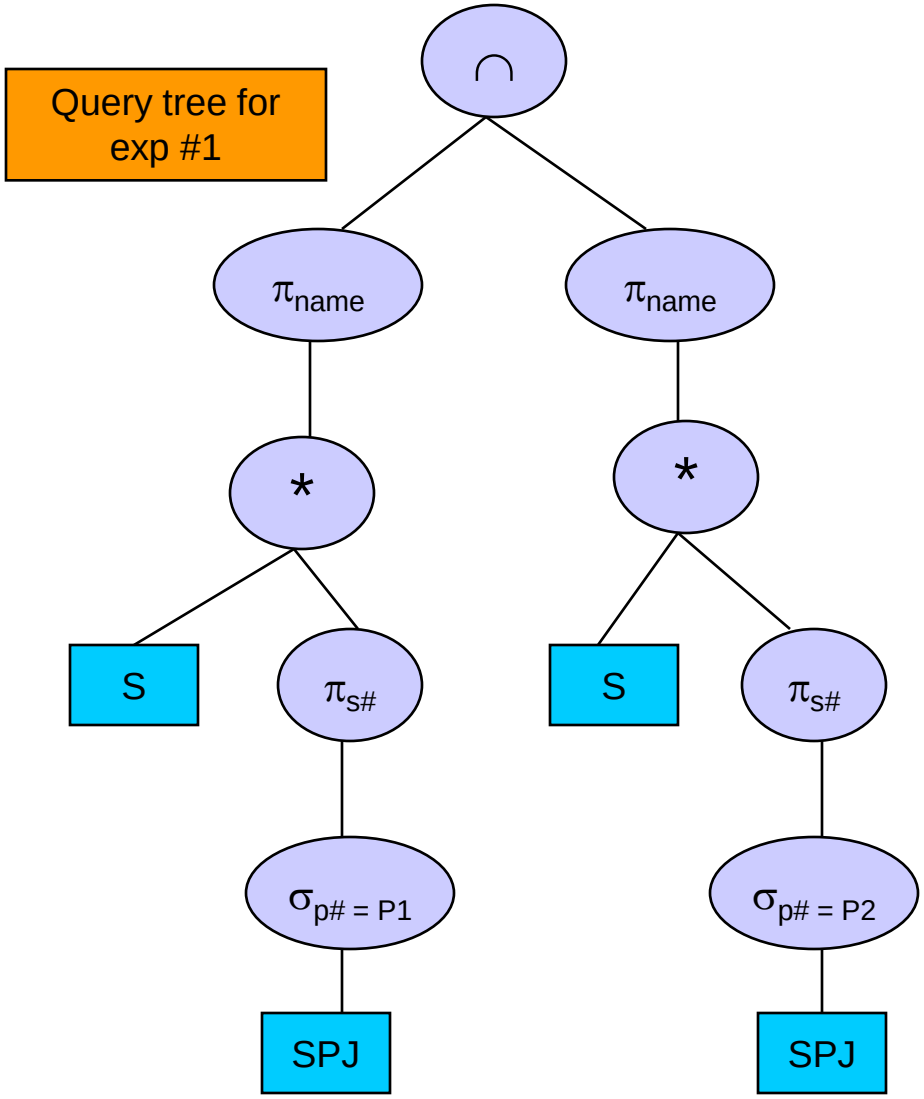
Similar to SQL Versions #1 and #2

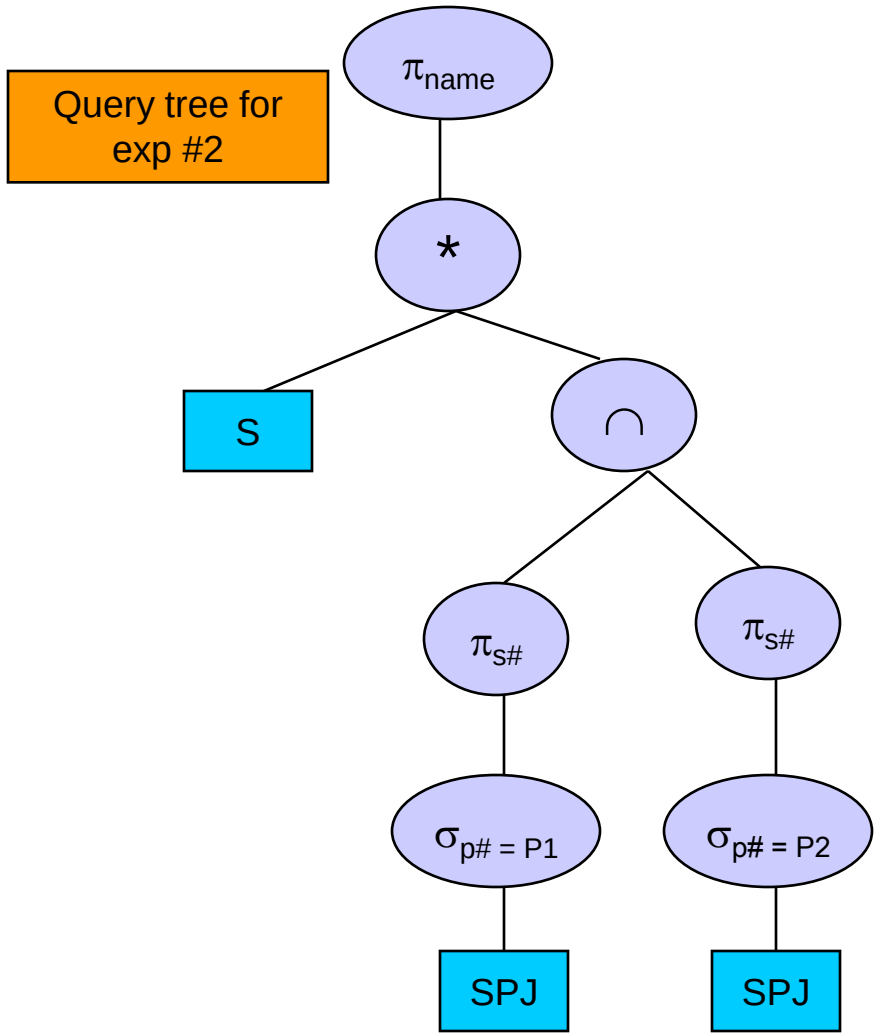
exp #3: $\pi_{name}(s * (\pi_{s\#}(\sigma_{spj.p\#=P1}(spj)(\sigma_{spj1.p\#=P2}(spj1)(spj \times spj1)))))$

Similar to SQL Version #3

exp #4: $\pi_{name}(s * (\sigma_{spj.p\#=P1}(\sigma_{spj1.p\#=P2}(\sigma_{spj.s\#=spj1.s\#}(\pi_{spj.s\#,spl1.s\#,spj.p\#,spj1.p\#}(spj \times spj1))))))$

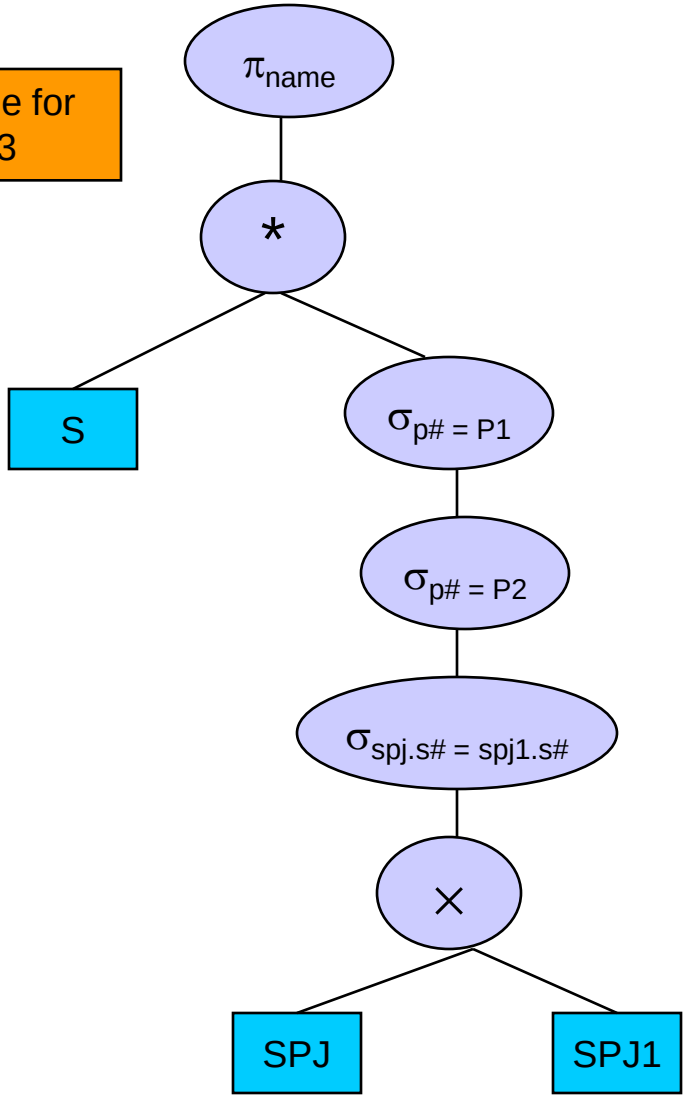
Corresponding Query Trees



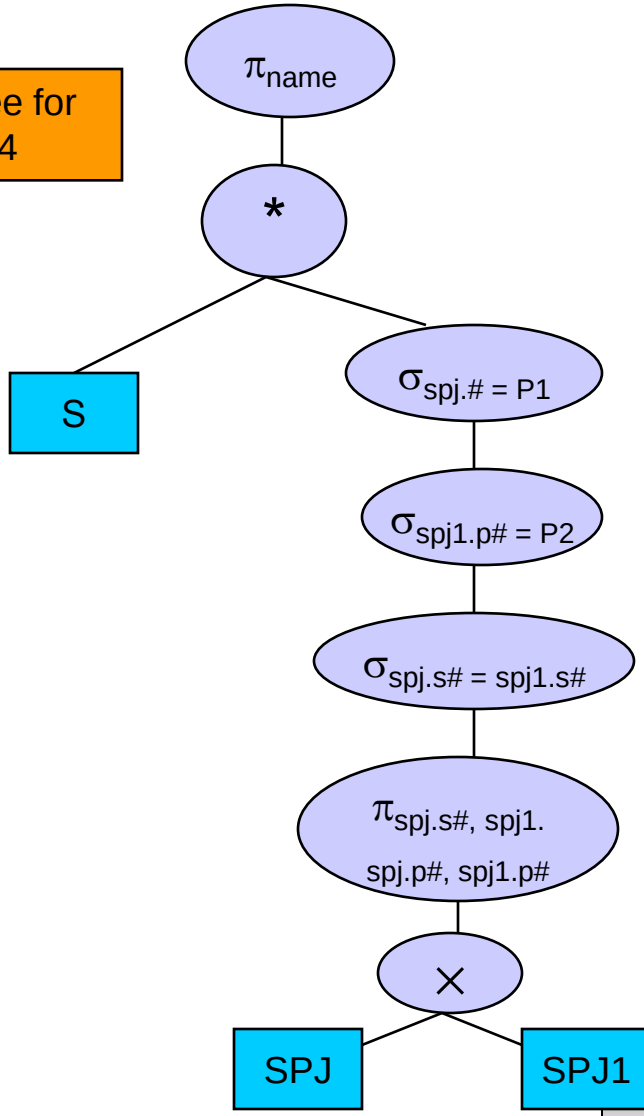


Corresponding Query Trees

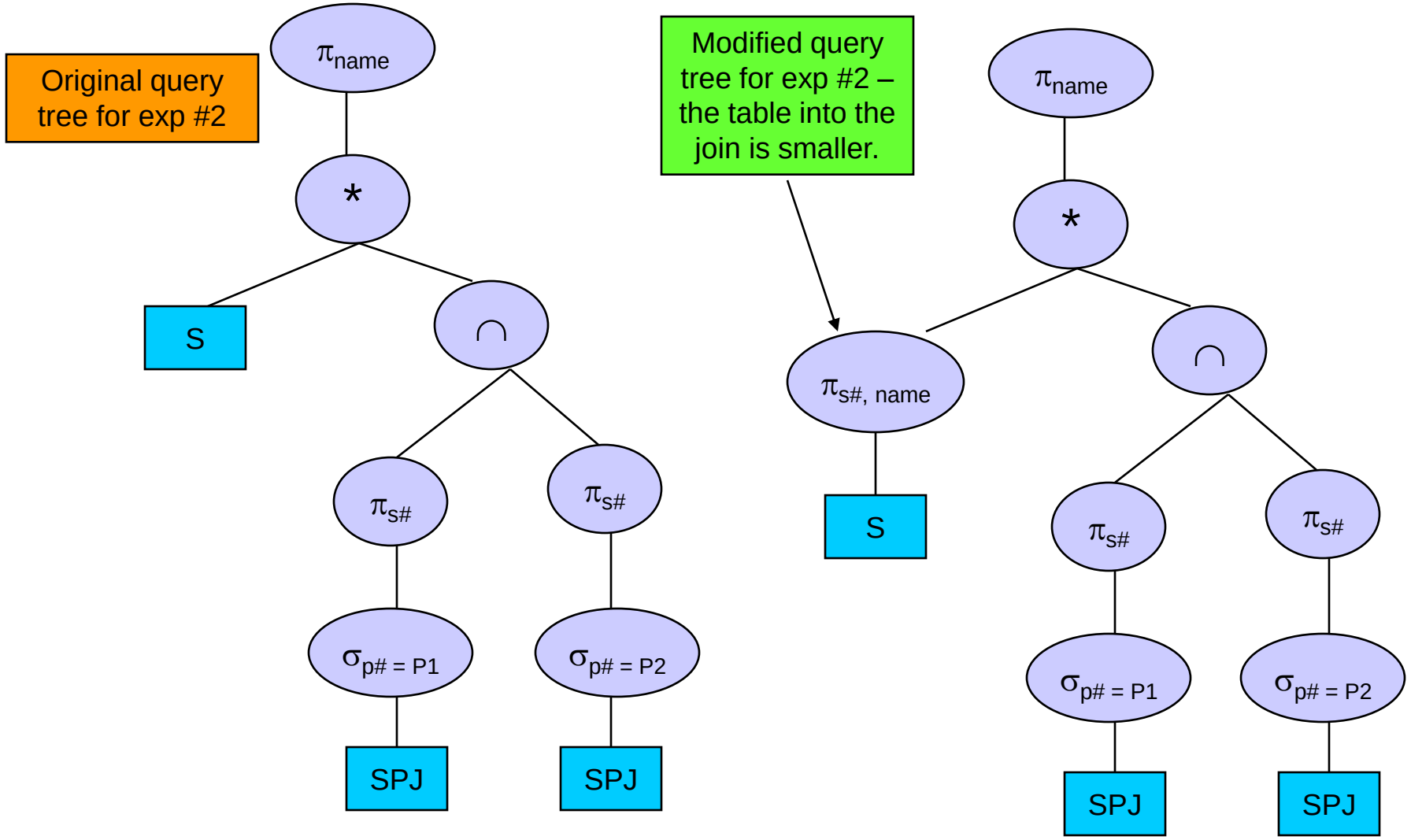
Query tree for exp #3



Query tree for exp #4



Corresponding Query Trees



Basic Query Execution Algorithms

- For each operation (relational algebra operation, plus others) as well as combinations of operations, the DBMS will maintain one or more algorithms to execute the operation.
- Certain algorithms will apply to particular storage structures and access paths and thus can only be utilized if the underlying files involved in the operation include these access paths.
- Typically, the access paths will involve indices and/or hash tables, although other hybrid access paths are also possible.
- In the next few pages will examine some of these query execution strategies for the basic relational algebra operations.

Algorithms for Selection Operations

- There are many different options for Select operations based on the availability of access paths, indices, etc.
- Search algorithms for Select operations are one of two types:
  - index scans: search is directed from an index structure.
  - file scans: records are selected directly from the file structure.
- (FS1-linear search): Heap files typically are searched with a linear search algorithm.
- (FS2-fast search): Sequential files are typically searched with a binary or jump type of search algorithm.
- (IS3-primary index or hash key to extract single record): In these cases the selection condition involves an equality comparison on a key attribute for which a primary index has been created (or a hash key can be used.)

Algorithms for Selection Operations (cont.)

- (IS4-primary index or hash key to extract multiple records): In these cases the selection condition involves a non-equality based comparison (<, <=, >, >=) on a key attribute for which a primary index has been created. The primary index is used to find the record which satisfies the equality condition and then based upon this record, all other preceding (< or <=) or subsequent (> or >=) records are retrieved from the ordered file.

- (IS5-clustering index to extract multiple records): In these cases the selection condition involves an equality comparison on a non-key attribute which has a clustering index (a secondary index). The clustering index is used to retrieve all records which satisfy the selection condition.

- (IS6 – secondary index, B^+ tree): A selection condition with an equality comparison, a secondary index can be used to retrieve a single record if the indexing field is a key or to retrieve multiple records if the indexing field is not a key. Secondary indices can also be used for any of the comparison operators, not just equality.

Algorithms for Conjunctive Selections

- Conjunctive selections are selection conditions in which several conditions are logically AND'ed together.
- For simple (non-conjunctive) selection conditions, optimization basically means that you check for the existence of an access path on the attribute involved in the condition and use it if available, otherwise a linear search is performed.
- Query optimization for selection is most useful for conjunctive conditions whenever more than one of the participating attributes has an access path.
- The optimizer should choose the access path that retrieves the fewest records in the most efficient manner.

Algorithms for Conjunctive Selections (cont.)

- The overriding concern when choosing between multiple simple conditions in a conjunctive select condition is the selectivity of each condition.
- Selectivity is defined as:

$$Selectivity = \frac{\#\ of\ \ records\ which\ satisfy\ the\ condition}{\#\ of\ \ records\ in\ the\ relation}$$

- The smaller the selectivity the fewer the tuples the condition selects.
- Thus the optimizer should schedule the conjunctive selection comparisons so that the smallest selectivity conditions are applied first followed by the higher and higher selectivity values so that the last condition applied has the highest selectivity value.

Algorithms for Conjunctive Selections (cont.)

- Usually, exact selectivity values for all conditions are not available. However, the DBMS will maintain estimates for most if not all types of conditions and these estimates will be used by the optimizer.
- For example:
  - The selectivity of an equality condition on a key attribute of a relation r(R) is:

$$\frac{1}{|r(R)|}$$

  - The selectivity of an equality condition on an attribute with n distinct values can be estimated by:

$$\frac{\left(\frac{|r(R)|}{n}\right)}{|r(R)|} = \frac{1}{n}$$

Assuming that the records are evenly distributed across the n distinct values, a total of |r(R)|/n records would satisfy an equality condition on this attribute.

Algorithms for Conjunctive Selections (cont.)

- (IS7-conjunctive selection): If an attribute is involved in any single simple condition in the conjunctive selection has an access path that permits the use of any of FS2 through IS6, use that condition to retrieve the records, then check if each retrieved record satisfies the remaining simple conditions in the conjunctive condition.
- (IS8-conjunctive selection using a composite index): If two or more attributes are involved in an equality condition and a composite index (or hash structure) exists for the combined fields – use the composite index directly.
- (IS9-conjunctive selection by intersection of record pointers): If secondary indices are available on any or all of the attributes involved in an equality comparison (assuming that the indices use record pointer and not block pointers), then each index is used to retrieve the record pointers that satisfy the individual simple conditions. The intersection of these record pointers is the set of tuples that satisfy the conjunction.

Algorithms for Join Operations

- The join operation and its variants are the most time consuming operations in query processing.
- Most joins are either natural joins or equi-joins.
- Joins which involve two relations are called two-way joins while joins involving more that two relations are called multiway joins.
- While there are several different strategies that can be employed to process two-way joins, the number of potential strategies grows very rapidly for multiway joins.

Two-way Join Strategies

- We'll assume that the relations to be joined are named R and S, where R contains an attribute named A and S contains an attribute named B which are join compatible.
- For the time-being, we'll consider only natural or equijoin strategies involving these two attributes.
- Note that for a natural join to occur on attributes A and B, a renaming operation on one or both of the attributes must occur prior to the natural join operation.
  - Note too, that if attributes A and B are the only join compatible attributes in R and S, that the equi-join operation R $*_{A=B}$ S has the same effect as a natural join operation.

Algorithms for Two-way Join Operations

- (J1-nested loop): A brute force technique where for each record $t \in R$ (outer loop) retrieve every record $s \in S$ (inner loop) and test if the two records satisfy the join condition, namely does t.A = s.B?
- (J2-single loop w/access structure): If an index or hash key exists for one of the two join attribute, for example, $B \in S$, retrieve each record $t \in R$ one at a time and then use the access structure to retrieve directly all matching records $s \in S$ that satisfy t.A = s.B.
- (J3-sort-merge join): If the records of both R and S are physically sorted (ordered) by the values of the join attributes A and B, then the join can be processed using the most efficient strategy. Both relations are scanned in the order of the join attributes; matching the records that have the same A and B values. In this fashion, each relation is scanned only once.
- (J4-hash-join): In this technique, the records of both relations R and S are hashed using the same hashing function (on the join attributes) to the same hash file. A single pass through the smaller relation will hash its records to the hash file. A single pass through the other relation will hash its records to the same bucket as the first pass combining all similar records.

Pipelining Operations

- Query optimization can also be effected by reducing the number of intermediate relations that are produced as a result of executing a query stream.
- This reduction in the number of intermediate relations is accomplished by combining several relational operations into a single pipeline of operations. This method is also sometimes referred to as stream-based processing.
- While the combining of operations in a pipeline eliminates some of the cost of reading and writing intermediate relations, it does not eliminate all reading and writing costs associated with the operations nor does it eliminate any processing.
- As an example, consider the natural join of two relations R and S, followed by the projection of a set of attributes from the join result.

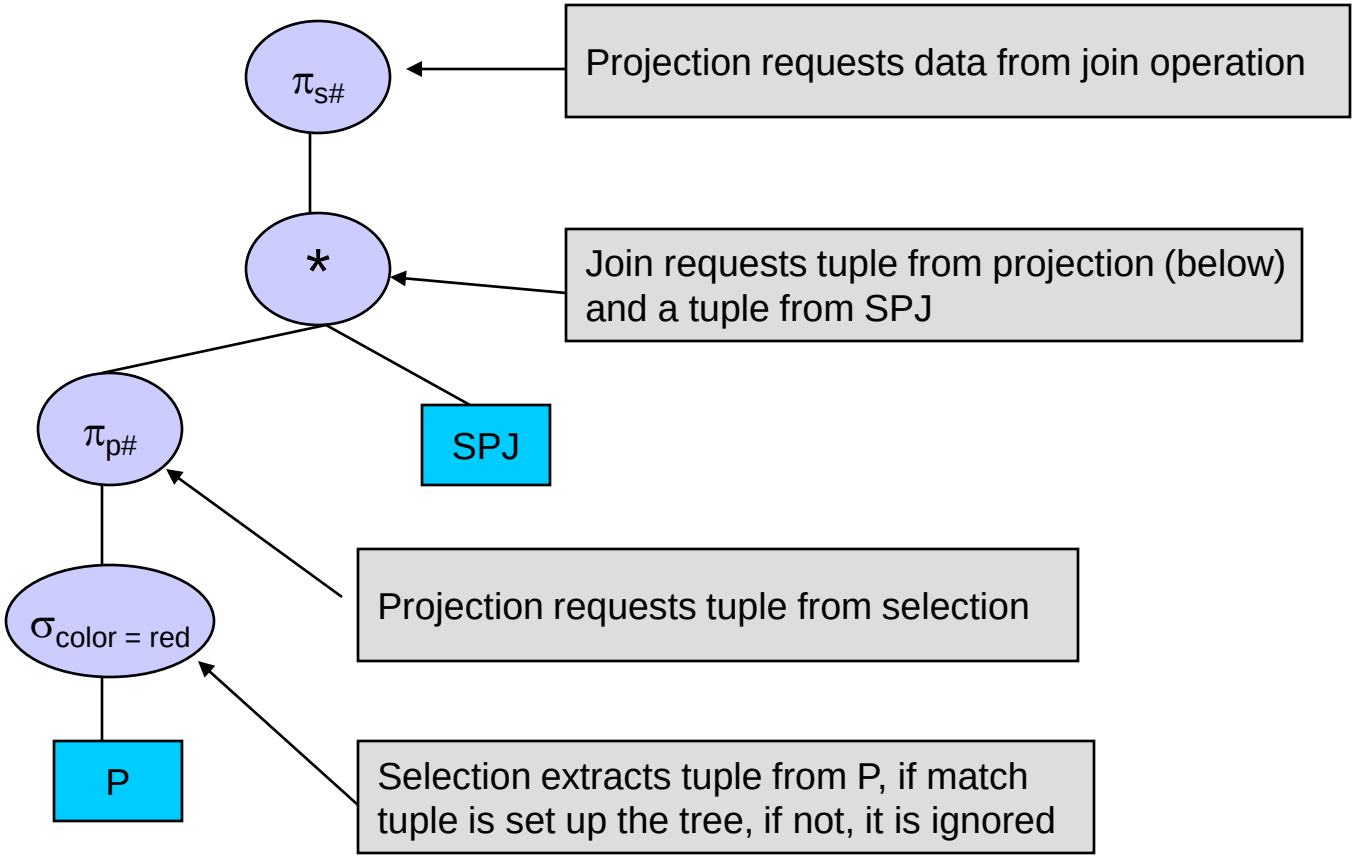
Pipelining Operations (cont.)

- In relational algebra this query looks like: $\pi_{(a, b, c)}(R * S)$
- This set of two operations could be executed as:
  - construct the join of R and S, save as intermediate table T1. [T1 = R * S]
  - project the desired set of attributed from table T1. [result = $\pi_{(a, b, c)}$(T1)]
- In the pipelined execution of this query, no intermediate relation T1 is produced. Instead, as soon as a tuple in the join of R and S is produced it is immediately passed to the projection operation to processing. The final result is created directly.
- In the pipelined version, results are being produced even before the entire join has been processed.

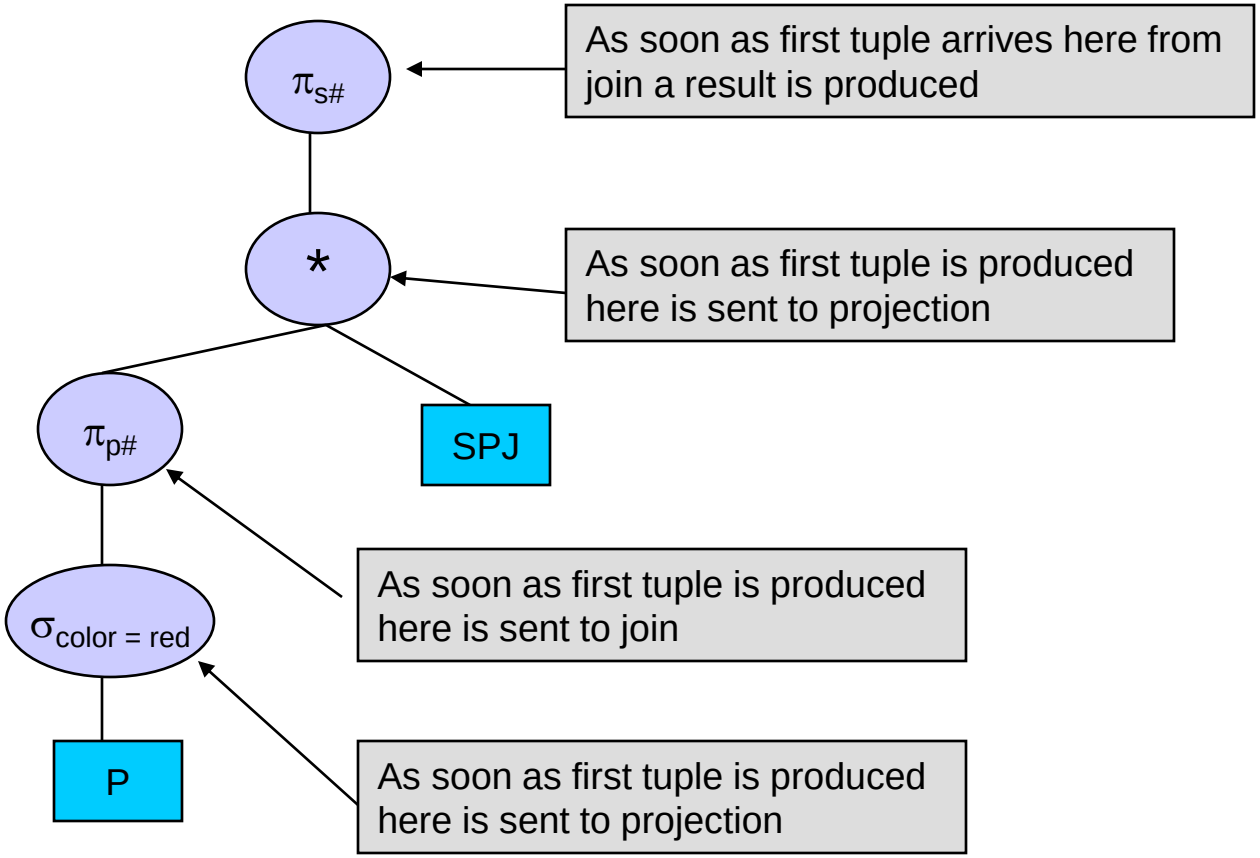
Pipelining Operations (cont.)

- There are two basic strategies that can be used to pipeline operations.
- Demand-driven pipelining: In effect, data is "pulled-up" the query tree as operations request data to operate upon.
- Producer-driven pipelining: In effect, data is "pushed-up" the query tree as lower level operations produce data which is sent to operations higher in the query tree.

Demand-Driven Pipelining Example



Producer-Driven Pipelining Example



Using Heuristics in Query Optimization

- The parser of the high-level query language generates the internal representation of the query which is optimized according to heuristic rules.
- The access routines which execute groups of operations together are based upon the access paths available for the relations involved are chosen by the query optimizer.
- One of the main heuristic rules is to apply projections and selections as early as possible. This is useful because the size of the relations involved in subsequent join operations (or other binary operations) are as small as possible.
- Basically, the query optimizer generates several different query expressions and selects the best choice.

Using Heuristics in Query Optimization (cont.)

- When an equivalent query expression is generated, you must be certain that it is in fact an equivalent expression.
- To this end, the query optimizer must follow certain transformation rules that will ensure equivalency amongst the various query expressions.
- The level of information available to the optimizer will affect the effectiveness of the equivalence generation scheme.
  - At the lowest level – only relation names are known:
    - $R \cap R \equiv R$
    - $\pi_X (R \cup S) \equiv \pi_X (R) \cup \pi_X (S)$
    - $\sigma_{A=B \text{ AND } B=C \text{ AND } A=C} (R) \equiv \sigma_{A=B \text{ AND } B=C} (R)$

Using Heuristics in Query Optimization (cont.)

- If schema information is available:
  - Given R(A, B), S(B,C) with r(R) and s(S) then,
  - $\sigma_{A=a}(R * S) \equiv \sigma_{A=a}(R) * S$
- If constraint information is known, they provide even more information and modification possibilities:
  - If you know that R(A,B,C,D) with r(R) and you also know that r satisfies $B \rightarrow C$, then $\pi_{A,B}(r) * \pi_{B,C}(r) \equiv \pi_{A,B,C}(r)$

- In general, there are many different equivalences that will hold and the optimizer can utilize as many as possible.
  - For example: $r \cup r \equiv r$, $r \cap r \equiv r$, $r - r \equiv \emptyset$

Using Heuristics in Query Optimization (cont.)

- Commutivity rules can also be applied to optimize query execution.
- For example what is the difference between R * S and S * R?
  - Suppose that R contains 3 tuples and S contains 5 tuples. Further suppose that each tuple in R is 10 bytes long and each tuple in S is 100 bytes long.
  - R * S: 1 pass through R generates 3 × 10 bytes = 30 bytes. Three passes through S (one for each tuple generated from R) generates 15 tuples × 100 bytes = 1500 bytes. Total = 1530 bytes.
  - S * R: 1 pass through S generates 5 × 100 bytes = 500 bytes. Five passes through R (one for each tuple generated from S) generates 15 tuples × 10 bytes = 150 bytes. Total = 650 bytes.
  - Clearly, S*R is a better strategy than is R*S.

Using Cost Estimation in Query Optimization

- Cost estimation is typically only used for "canned" query execution code, i.e., compiled queries that will be executed repeatedly.
- The time and effort required for this type of analysis is not justified for simple one-time query execution.
- The cost estimation technique considers the cost of executing a query from four different perspectives:

1. Access costs to secondary storage: this involves all the costs of searching, reading, and writing secondary storage.
2. Storage costs: this involves the cost of storing the intermediate files generated by the chosen execution strategy.
3. Computation costs: Sorting, merging, computation in attributes (selection and join conditions).
4. Communication costs: In a distributed environment, this includes the cost of shipping the query and/or its results to the originating site.

Semantic Query Optimization

- This technique uses the semantics of the database and the various constraints that apply to semantically modify queries into queries which are more efficient.
- For example, suppose a user issues the following query:
  - $\pi_{s\#}(\sigma_{qty > 100}(SPJ))$ {list supplier numbers for suppliers who ship at least one part in a quantity greater than 100.}
  - If a constraint exists that states: all quantities <= 75, then the optimizer could inform the system that the query did not need to be executed at all and the result is simply the empty set.