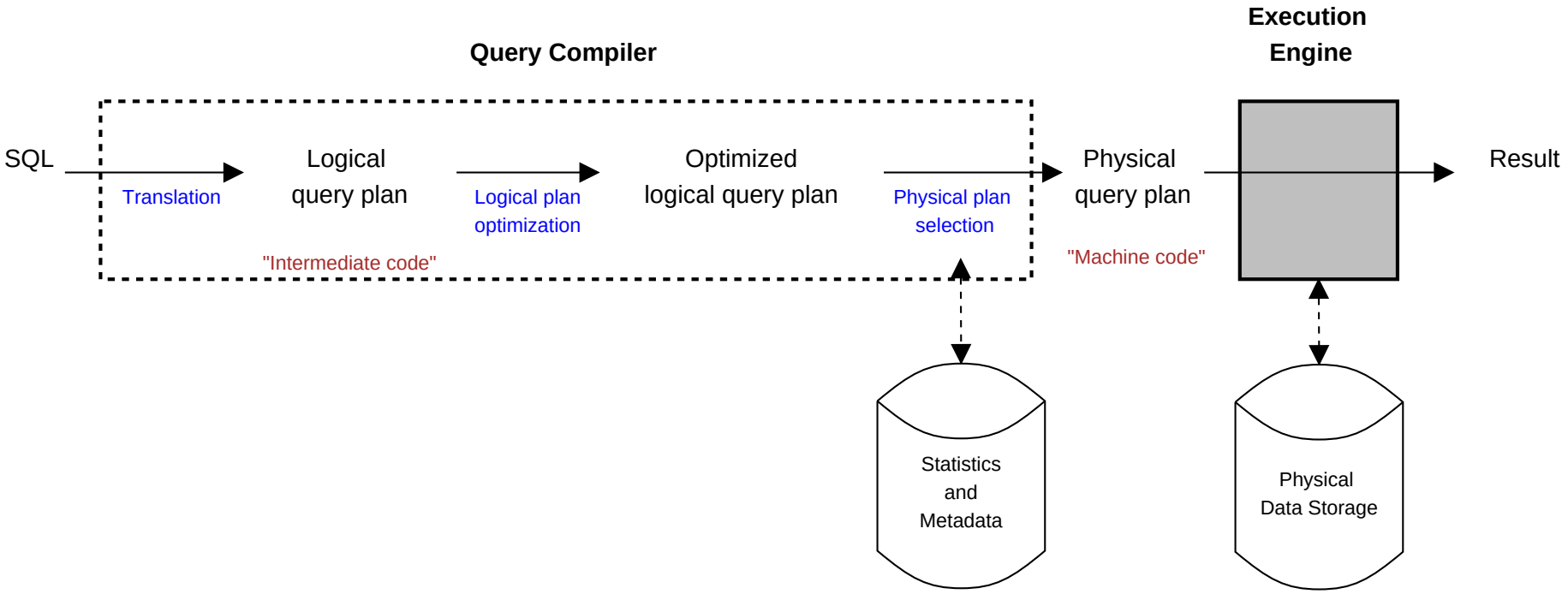
Cost-Based Plan Selection

Enumerate, Estimate, Select

Cost-Based Plan Selection

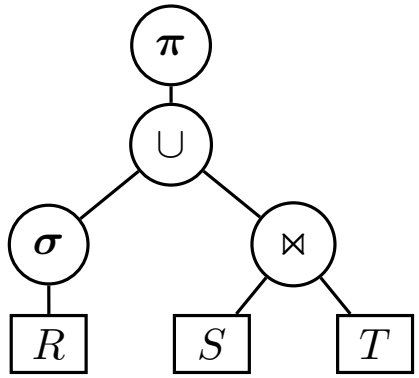


Components of the query compiler that we already know:

- SQL $\rightarrow$ relational algebra (i.e., a logical query plan)
- Logical query plan $\rightarrow$ optimized logical query plan

Cost-Based Plan Selection

The next step: logical query plan → physical query plan

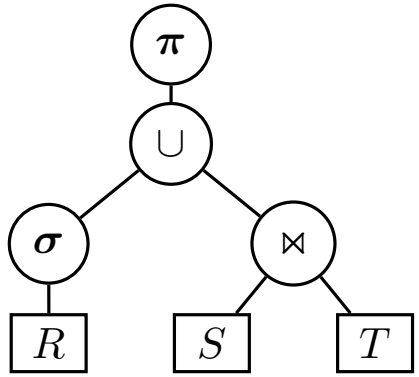


- To obtain a physical query plan we need to assign to each node in the logical query plan a physical operator.
- We want to obtain the physical plan with the smallest total execution cost.
- Hence, we need to compare, for every node and every applicable physical operator, its cost.

- In order to estimate this cost we need (among others) the parameters $B(R)$, $T(R)$, and $V(R, A_1, \ldots, A_n)$
- These belong to the statistics that a DBMS typically stores in its system catalog
- But these statistics only exist for the relations stored in the database, not for subresults computed during query evaluation!

Cost-Based Plan Selection

Result size estimation



- For every internal node n we hence need to estimate the parameters $B(n)$, $T(n)$, and $V(n, A_1, \ldots, A_k)$
- Note that we can compute $B(n)$ given (1) $T(n)$; (2) the size of the tuples output by n; and (3) the size of a block
- Also note that $T(n)$ and $V(n, A_1, \ldots, A_k)$ only depend on the logical query plan, not on the physical plan that we are computing!

- For every logical operator we hence want to estimate, given the parameters of its input relations, the number of tuples produced by the operator as well as the number of distinct $A_1, \ldots, A_n$ values in the output.

  $\rightarrow$ see section 16.4 in the book
- A DBMS often also collects more detailed statistics

  $\rightarrow$ see sections 16.5.1 and 16.5.2 in the book

Cost-Based Plan Selection

Join ordering

During the optimization of the logical query plan we:

- remove redundant joins;
- push selections and projections; recognize joins.

The order in which the joins are to be executed is not yet fixed, however!

Cost-Based Plan Selection

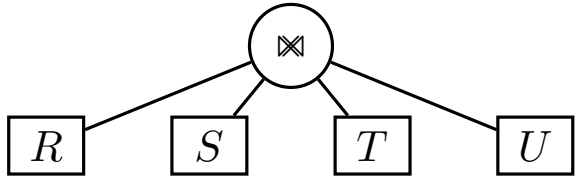
Join ordering

Example: relations $R(A,B), S(B,C), T(C,D), U(D,A)$ and the query

SQL:
```
SELECT * FROM R,S,T,U
WHERE R.B = S.B AND S.C = T.C AND T.D = U.D
```

Algebra: $R \bowtie S \bowtie T \bowtie U$

- So far, we have always considered the join as a polyadic operator:



  After all, the join order is irrelevant for logical query plans.
- However, physical join operators are binary!
- When devising a physical query plan, the join order therefore becomes very important, as we illustrate next.

Cost-Based Plan Selection

Join ordering

Example: relations $R(A,B), S(B,C), T(C,D), U(D,A)$ and the query

SQL:
```
SELECT * FROM R,S,T,U
WHERE R.B = S.B AND S.C = T.C AND T.D = U.D
```

Algebra: $R \bowtie S \bowtie T \bowtie U$

We can interpret this as:

$$((R \bowtie S) \bowtie T) \bowtie U \quad \text{or} \quad (R \bowtie S) \bowtie (T \bowtie U) \quad \text{or} \quad \ldots$$

But also as:

$$((R \bowtie T) \bowtie U) \bowtie S \quad \text{or} \quad ((R \bowtie S) \bowtie U) \bowtie T \quad \text{of} \quad \ldots$$

Cost-Based Plan Selection

Join ordering

The chosen order can influence the total cost of the physical query plan.

Consider, for example, $R(A,B), S(B,C), T(A,E)$. Assume

$$B(R) = 50 \qquad B(S) = 50 \qquad B(T) = 50$$
$$B(R \bowtie S) = 150 \qquad B(S \bowtie T) = 2500 \qquad B(R \bowtie T) = 200$$

Further assume that we execute all joins by means of the one-pass algorithm. What is the best order to compute $R \bowtie S \bowtie T$?

1. **Cost of** $R \bowtie (S \bowtie T)$**:**

$$B(R) + B(S \bowtie T) + B(S) + B(T) = 2650$$

2. **Cost of** $S \bowtie (R \bowtie T)$**:**

$$B(S) + B(R \bowtie T) + B(R) + B(T) = 350$$

3. **Cost of** $T \bowtie (R \bowtie S)$**:**

$$B(T) + B(R \bowtie S) + B(R) + B(S) = 300$$

Cost-Based Plan Selection

Join ordering

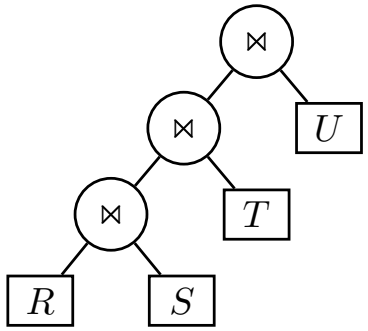
- To obtain the physical plan with the least cost we would hence have to enumerate and compare every possible join ordering.
- The number of possible ordering to join n relations: $n! \times T(n)$ with

$$T(1) = 1 \qquad\qquad T(n) = \sum_{i=1}^{n-1} T(i) \times T(n-i)$$

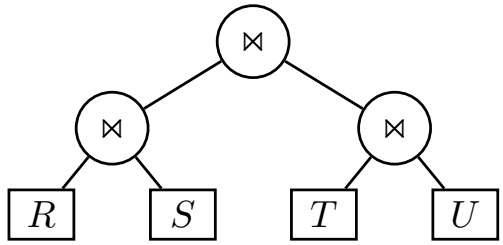
$\rightarrow$ Query optimization should in no case take more time than the actual execution of the query. We will therefore not consider all possible orders, but only a limited subclass.

Cost-Based Plan Selection

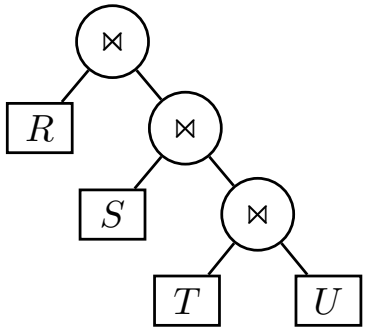
Kinds of join orderings



left-deep



bushy



right-deep

In practice a query compiler usually only considers left-deep join orderings:

- There are still $n!$ possible orderings of this form, but that is already a lot less.
- Left-deep orderings use, in general, less memory. Furthermore, in general they require fewer subresults to be stored.

→ See section 16.6.3 in the book

Cost-Based Plan Selection

Plan selection

To compute the best physical plan for a given logical query plan we should, in principle:

1. Calculate all possible (left-deep) join orderings of the logical plan
2. For each such plan calculate all possible assignments of physical operators to the nodes
3. From this enormous pile of candidate physical query plans choose the one with the least estimated cost.

There are exponentially many candidate physical query plans

- Query compilation should in no case take longer than the actual execution of the query!
- In general it is hence impossible to inspect all candidate physical plans.

  Heuristics: Branch-and-Bound Plan Enumeration; Hill Climbing; Dynamic Programming; Selinger-Style; Greedy

  → See section 16.5.4, 16.6.4 and 16.6.5 in the book

Cost-Based Plan Selection

Greedy plan selection

In the exercises we will use the following greedy algorithm.

- Start with a logical query plan without join ordering.
- We work bottom-up: first we assign physical operators to the leaves, then to the parents of the leaves, then to their parents, and so on. At each point we choose the phyiscal operator with the least cost.
- When we reach a join operator (e.g., $R \bowtie S \bowtie T \bowtie U$) and need to determine an ordering of its various members then:
  1. We start by joining the two relations for which the best physical join algorithm yields the smallest cost
     $\rightarrow$ e.g., execute $R \bowtie T$ through a hash-join
  2. Add, from the remaining relations (S or U), those relations to the join for which the best physical join-algorithm yields the smallest cost.
     $\rightarrow$ e.g., $(R \bowtie T) \bowtie U$ through a one-pass join
  3. Repeat the previous step until we have a complete join ordering.

Cost-Based Plan Selection

Greedy plan selection

- This is a generalization of the greedy algorithm to compute a join ordering described in section 16.6.6 from the book. However, we use I/O operations as our cost metric instead of the size of the intermediate results as done in the book.
- Often, the leaves of the logical query plan are selections. We have seen two physical operators for selections: table-scan and index-scan. The book describes in section 16.7.1 how we can choose the best selection method when the selection condition is complex.

Cost-Based Plan Selection

Greedy plan selection need not return the optimal plan

- It may return a more expensive join ordering. For example:

$$R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(A,D)$$

Assume: the greedy algorithm computes $((R \bowtie S) \bowtie T) \bowtie U)$ with

$$B(R \bowtie S) = 100 \qquad\qquad B((R \bowtie S) \bowtie T) = 2000$$

Assume: the alternative ordering $((R \bowtie U) \bowtie T) \bowtie S$ yields

$$B(R \bowtie U) = 200 \qquad\qquad B((R \bowtie U) \bowtie T) = 1000$$

When we hence execute the joins using the one-pass algorithm we get the following costs, respectively:

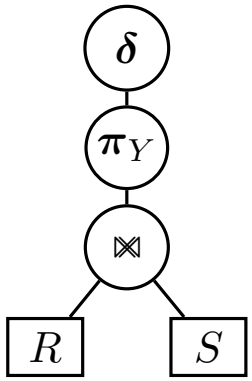
1. $B(R) + B(S) + B(R \bowtie S) + B(T) + B((R \bowtie S) \bowtie T) + B(U)$
2. $B(R) + B(U) + B(R \bowtie U) + B(T) + B((R \bowtie U) \bowtie T) + B(S)$

The second ordering yields a saving of 900 I/Os.

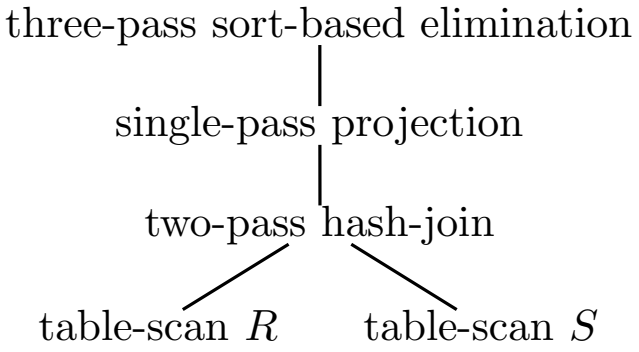
Cost-Based Plan Selection

Greedy plan selection need not return the optimal plan

- It does not take into account the properties of the output of an operator. For example (R and S share only the Y attribute):



logical plan



physical plan

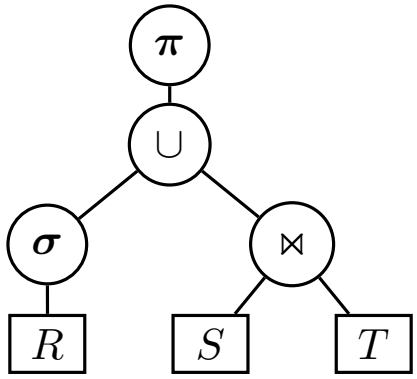
Consider the setting where there is limited memory available. The optimized sort-merge join is not applicable; only the non-optimized version. In this case the two-pass hash-join is cheaper, and is hence selected by the greedy algorithm. Because the output of $R \bowtie S$ is large, we will eventually have to remove duplicates by means of a three-pass algorithm.

If, however, we had executed the join by means of a two-pass sort-merge join, then its result would have been sorted on Y and we would have been able to compute the duplicate removal by means of the one-pass algorithm instead of the three-pass one. In that case, the total costs would have been smaller (check this!)

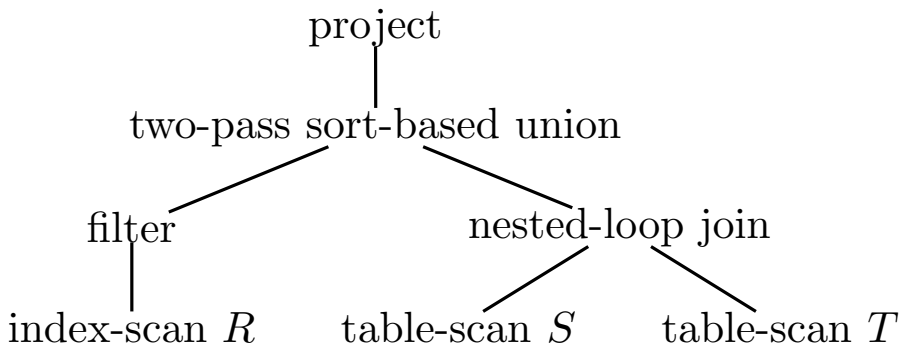
Cost-Based Plan Selection

Finally

The result of the greedy algorithm is an execution tree in which every node is a physical operator.



logical plan



physical plan

We remain to decide, for every internal node, whether we will materialize or pipeline the subresults.

→ See sections 16.7.3, 16.7.4, and 16.7.5