Advanced Database Systems

COMP7/8116 (Fall 2024)

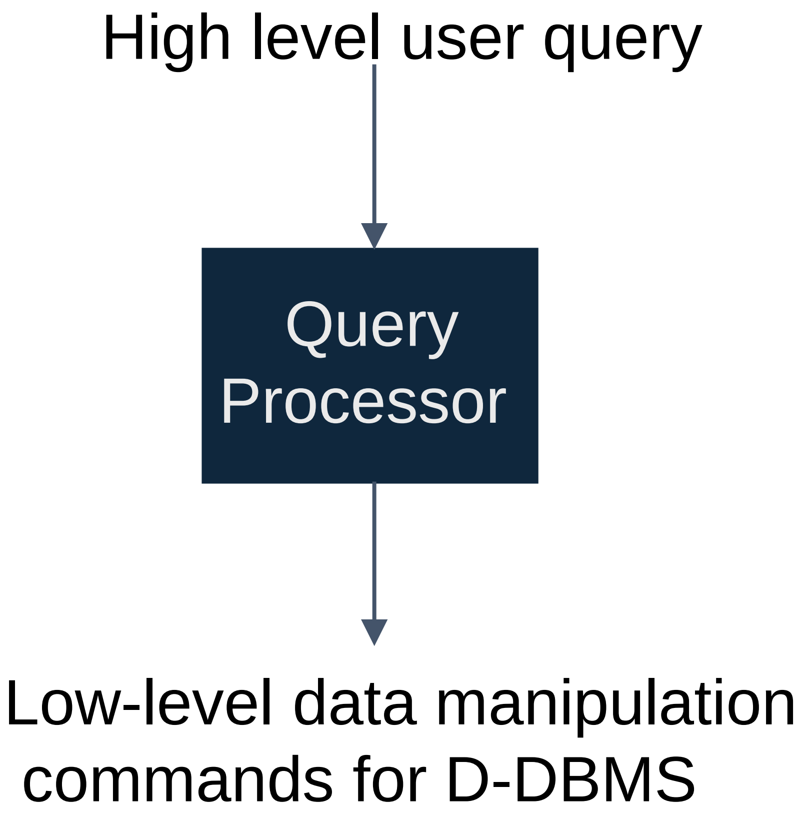
Distributed Query Processing

Xiaofei Zhang

Outline

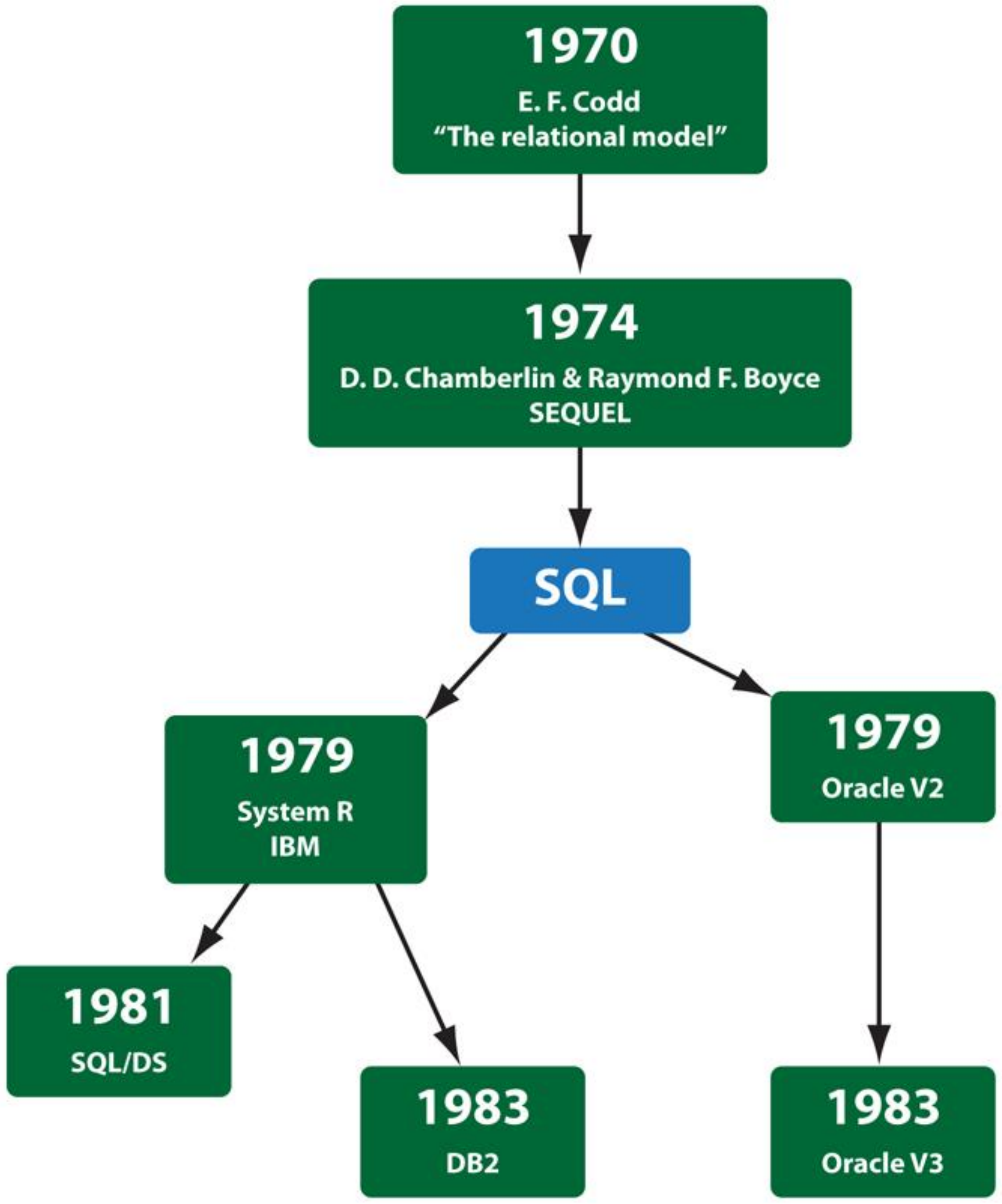
- Distributed Query Processing
  - Query Decomposition and Localization
  - Join Ordering
  - Distributed Query Optimization
  - Adaptive Query Processing

Query Processing in a DDBMS

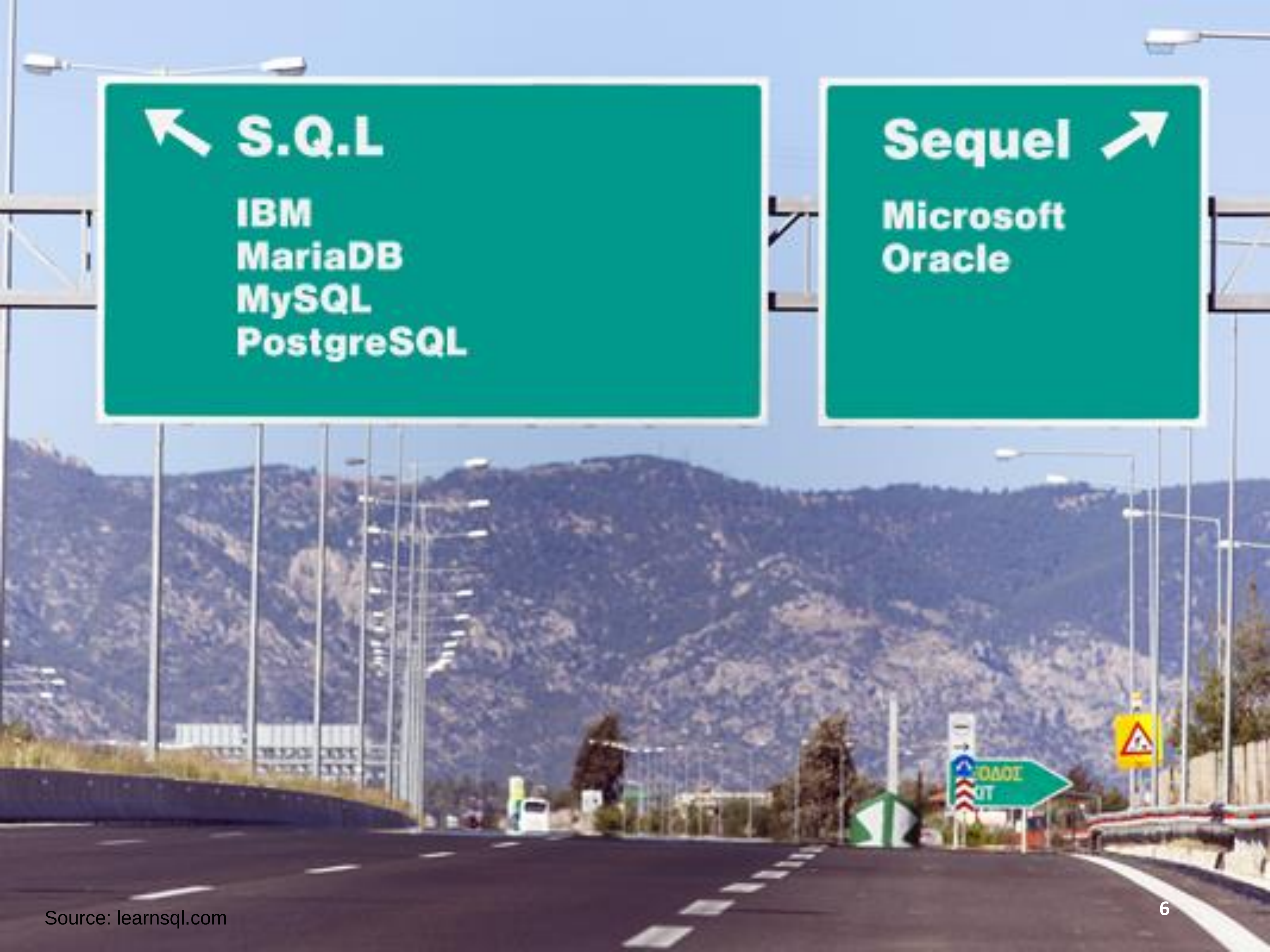


Query Processing Components

- Query language
  - SQL: “intergalactic dataspeak”
- Query execution
  - The steps that one goes through in executing high-level (declarative) user queries.
- Query optimization
  - How do we determine the “best” execution plan?
- We assume a homogeneous D-DBMS



Source: learnsql.com



Source: learnsql.com

Selecting Alternatives

```
SELECT   ENAME
FROM     EMP NATURAL JOIN ASG
WHERE    RESP = "Manager"
```

Strategy 1

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

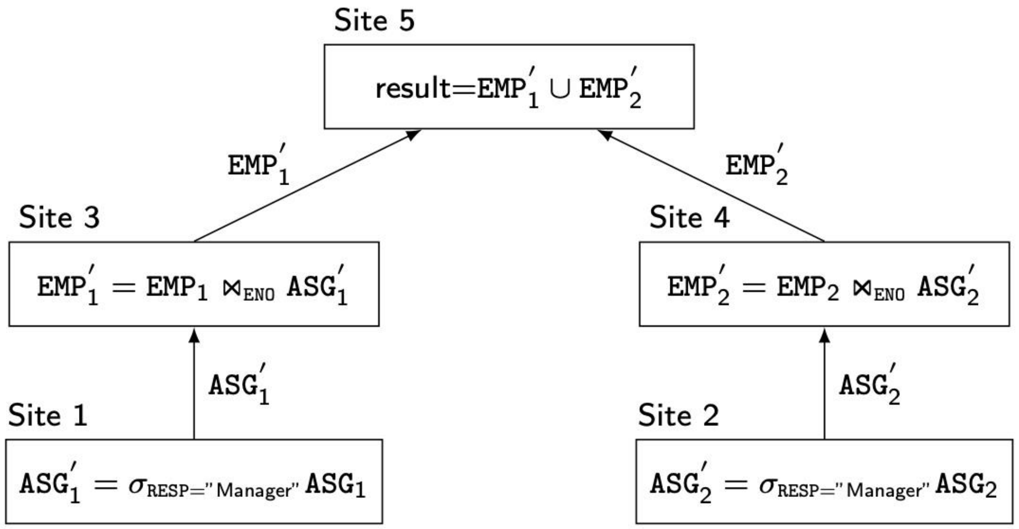
Strategy 2

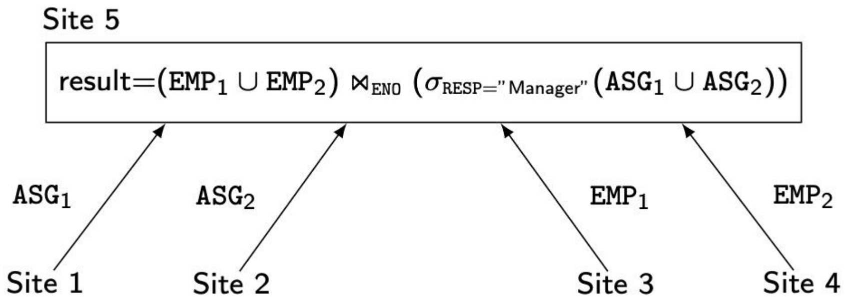
$\Pi_{ENAME}(EMP \bowtie_{ENO} (\sigma_{RESP=\text{“Manager”}}(ASG))$

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

What is the Problem?

Site 1	Site 2	Site 3	Site 4	Site 5
$ASG_1 = \sigma_{ENO \leq \text{“E3”}}(ASG)$	$ASG_2 = \sigma_{ENO > \text{“E3”}}(ASG)$	$EMP_1 = \sigma_{ENO \leq \text{“E3”}}(EMP)$	$EMP_2 = \sigma_{ENO > \text{“E3”}}(EMP)$	Result





Cost of Alternatives

- Assume
  - *size*(EMP) = 400, *size*(ASG) = 1000
  - tuple access cost = 1 unit; tuple transfer cost = 10 units
  - There are 20 managers (evenly distributed in ASG)
- Strategy 1

• produce ASG': (10+10) ∗ tuple access cost	20
• transfer ASG' to the sites of EMP: (10+10) ∗ tuple transfer cost	200
• produce EMP': (10+10) ∗ tuple access cost ∗ 2	40
• transfer EMP' to result site: (10+10) ∗ tuple transfer cost	200
Total Cost	460

- Strategy 2

• transfer EMP to site 5: 400 ∗ tuple transfer cost	4,000
• transfer ASG to site 5: 1000 ∗ tuple transfer cost	10,000
• produce ASG': 1000 ∗ tuple access cost	1,000
• join EMP and ASG': 400 ∗ 20 ∗ tuple access cost	8,000
Total Cost	23,000

Query Optimization Objectives

- Minimize a cost function
  - I/O cost + CPU cost + communication cost
  - These might have different weights in different distributed environments
- Wide area networks
  - Communication cost may dominate or vary much
    - Bandwidth
    - Speed
    - Protocol overhead
- Local area networks
  - Communication cost not that dominant, so total cost function should be considered
- Can also maximize throughput

Complexity of Relational Operations

- 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)

Types Of Optimizers

- Exhaustive search
  - Cost-based
  - Optimal
  - Combinatorial complexity in the number of relations
- Heuristics
  - Not optimal
  - Regroup common sub-expressions
  - Perform selection, projection first
  - Replace a join by a series of semi-joins
  - Reorder operations to reduce intermediate relation size
  - Optimize individual operations

Optimization Granularity

- Single query at a time
    - Cannot use common intermediate results
- Multiple queries at a time
    - Efficient if many similar queries
    - Decision space is much larger

Optimization Timing

- Static
  - Compilation ➔ optimize prior to the execution
  - Difficult to estimate the size of the intermediate results®error propagation
  - Can amortize over many executions
- Dynamic
  - Run time optimization
  - Exact information on the intermediate relation sizes
  - Have to re-optimize for multiple executions
- Hybrid
  - Compile using a static algorithm
  - If the error in estimate sizes > threshold, re-optimize at run time

Statistics

- Relation
  - Cardinality
  - Size of a tuple
  - Fraction of tuples participating in a join with another relation
- Attribute
  - Cardinality of domain
  - Actual number of distinct values
- Simplifying assumptions
  - Independence between different attribute values
  - Uniform distribution of attribute values within their domain

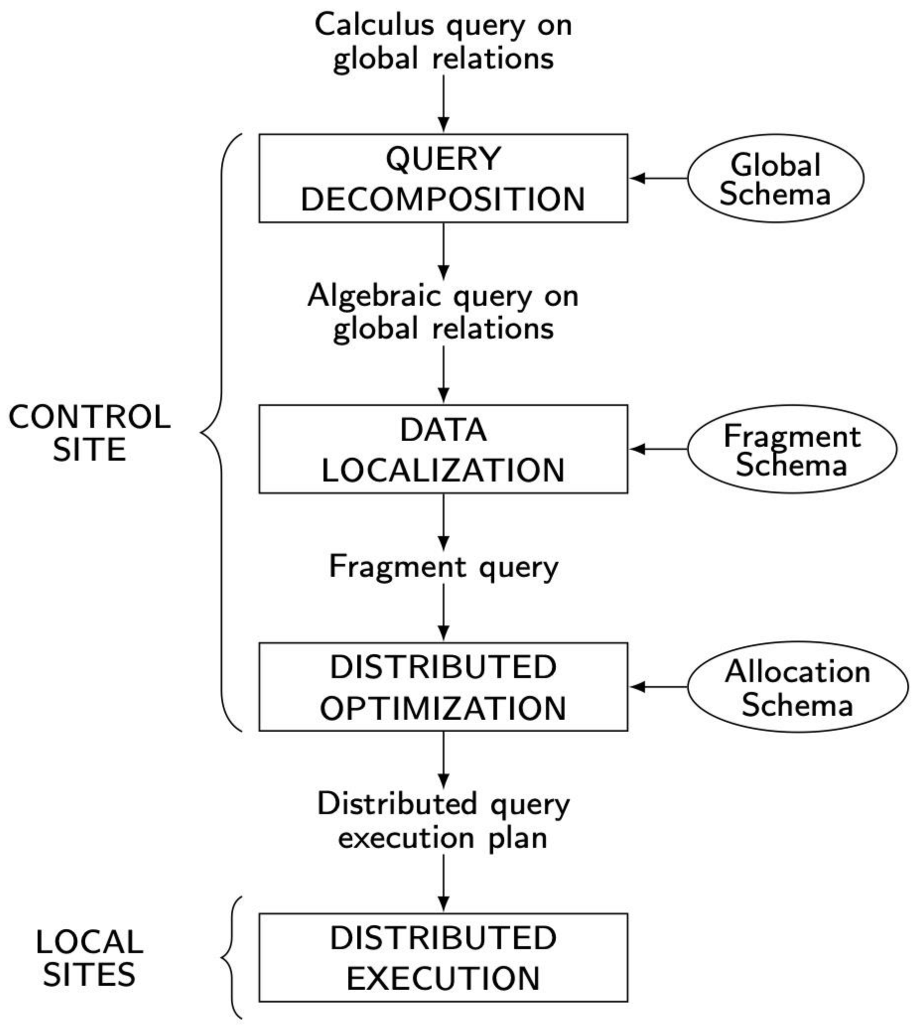
Optimization Decision Sites

- Centralized
  - Single site determines the "best" schedule
  - Simple
  - Need knowledge about the entire distributed database
- Distributed
  - Cooperation among sites to determine the schedule
  - Need only local information
  - Cost of cooperation
- Hybrid
  - One site determines the global schedule
  - Each site optimizes the local subqueries

Network Topology

- Wide area networks (WAN) – point-to-point
  - Characteristics
    - Relatively low bandwidth (compared to local CPU/IO)
    - High protocol overhead
  - Communication cost may dominate; ignore all other cost factors
  - Global schedule to minimize communication cost
  - Local schedules according to centralized query optimization
- Local area networks (LAN)
  - Communication cost not that dominant
  - Total cost function should be considered
  - Broadcasting can be exploited (joins)
  - Special algorithms exist for star networks

Distributed Query Processing Methodology



Outline

- Distributed Query Processing
  - Query Decomposition and Localization
  - Distributed Query Optimization
  - Join Ordering
  - Adaptive Query Processing

Step 1 – Query Decomposition

Same as centralized query processing

Input : Calculus query on global relations

- Normalization
  - Manipulate query quantifiers and qualification
- Analysis
  - Detect and reject “incorrect” queries
- Simplification
  - Eliminate redundant predicates
- Restructuring
  - Calculus query ➔ algebraic query
  - Use transformation rules

Step 2 – Data Localization

Input: Algebraic query on distributed relations

- Determine which fragments are involved
- Localization program
  - Substitute for each global query its materialization program
  - Optimize

Example

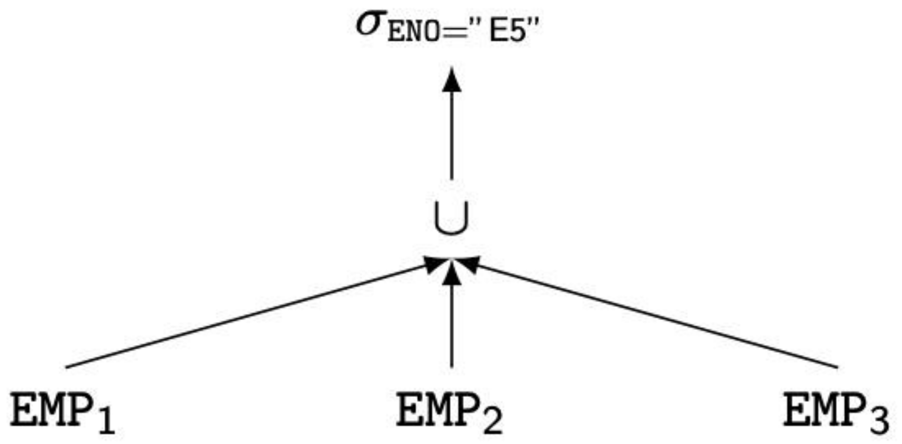
- Assume
  - EMP is fragmented as follows:
    - $EMP_1 = \sigma_{ENO \leq \text{“E3”}}(EMP)$
    - $EMP_2 = \sigma_{\text{“E3”} < ENO \leq \text{“E6”}}(EMP)$
    - $EMP_3 = \sigma_{ENO \geq \text{“E6”}}(EMP)$
  - ASG fragmented as follows:
    - $ASG_1 = \sigma_{ENO \leq \text{“E3”}}(ASG)$
    - $ASG_2 = \sigma_{ENO > \text{“E3”}}(ASG)$
- In any query
  - Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$
  - Replace ASG by $(ASG_1 \cup ASG_2)$

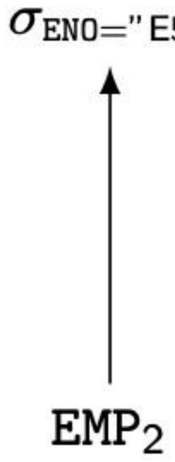
Reduction for PHF

- Reduction with selection
  - Relation R and $F_R=\{R_1, R_2, \ldots, R_w\}$ where $R_j=\sigma_{p_j}(R)$

    $\sigma_{p_i}(R_j)=\varnothing$ if $\forall x$ in R: $\neg(p_i(x) \wedge p_j(x))$

```
SELECT *
FROM   EMP
WHERE  ENO="E5"
```





Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over union

$$(R_1 \cup R_2) \bowtie S \Leftrightarrow (R_1 \bowtie S) \cup (R_2 \bowtie S)$$

  - Given $R_i = \sigma_{p_i}(R)$ and $R_j = \sigma_{p_j}(R)$

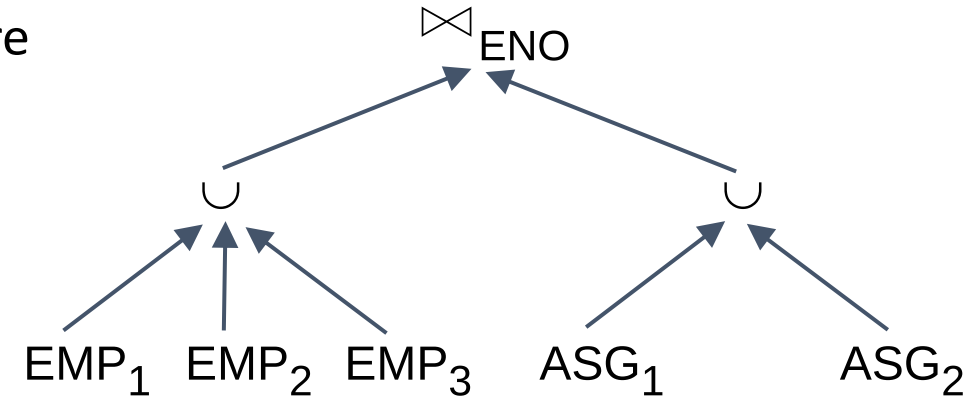
$$R_i \bowtie R_j = \emptyset \text{ if } \forall x \text{ in } R_i, \forall y \text{ in } R_j: \neg(p_i(x) \wedge p_j(y))$$

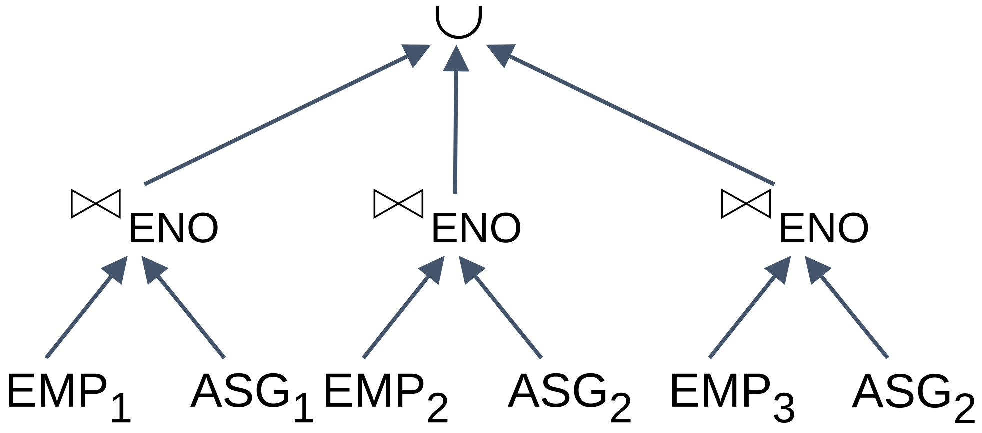
Reduction for PHF

- Assume EMP is fragmented as before and
  - ASG_1: $\sigma_{ENO \leq "E3"}(ASG)$
  - ASG_2: $\sigma_{ENO > "E3"}(ASG)$
- Consider the query

```
SELECT  *
FROM    EMP
NATURAL JOIN ASG
```

- Distribute join over unions
- Apply the reduction rule





Reduction for VF

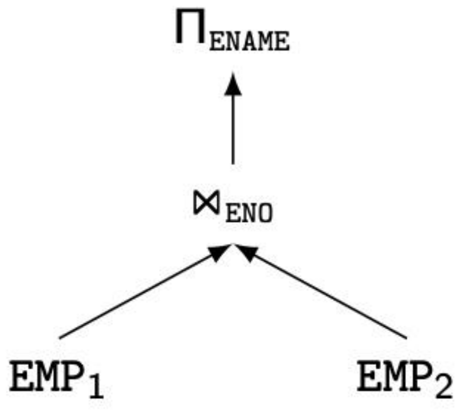
- Find useless (not empty) intermediate relations

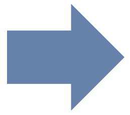
Relation R defined over attributes $A = \{A_1, ..., A_n\}$ vertically fragmented as $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$:

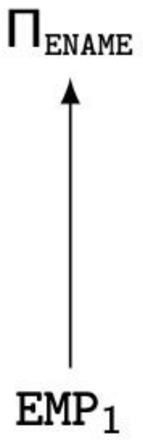
$\Pi_{D,K}(R_i)$ is useless if the set of projection attributes D is not in A'

Example: $EMP_1 = \Pi_{ENO,ENAME}(EMP)$; $EMP_2 = \Pi_{ENO,TITLE}(EMP)$

```
SELECT  ENAME
FROM    EMP
```







Reduction for DHF

- Rule :
  - Distribute joins over unions
  - Apply the join reduction for horizontal fragmentation
- Example

  ASG_1: ASG $\ltimes_{ENO}$ EMP_1

  ASG_2: ASG $\ltimes_{ENO}$ EMP_2

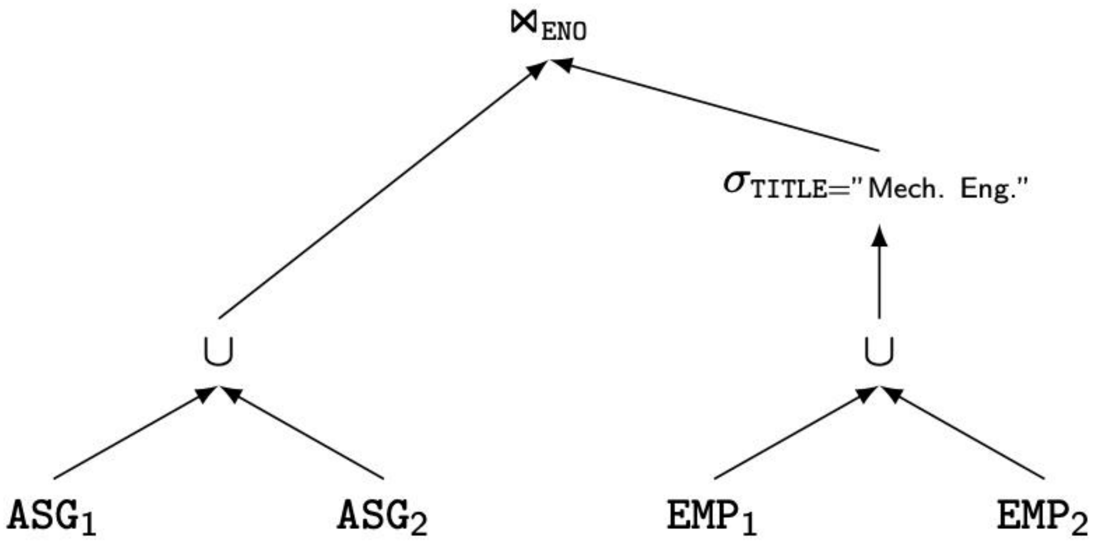
  EMP_1: $\sigma_{TITLE=\text{“Programmer”}}$ (EMP)

  EMP_2: $\sigma_{TITLE!=\text{“Programmer”}}$ (EMP)
- Query

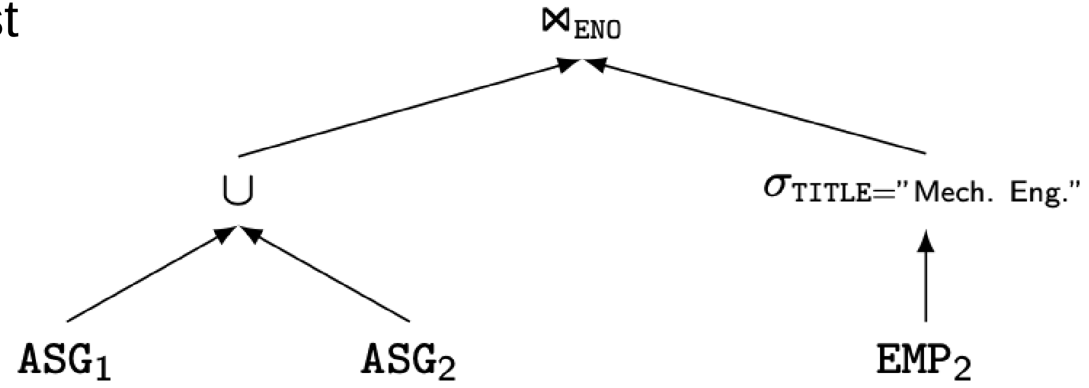
```
SELECT    *
FROM  EMP NATURAL JOIN ASG
WHERE EMP.TITLE = "Mech. Eng."
```

Reduction for DHF

Generic query

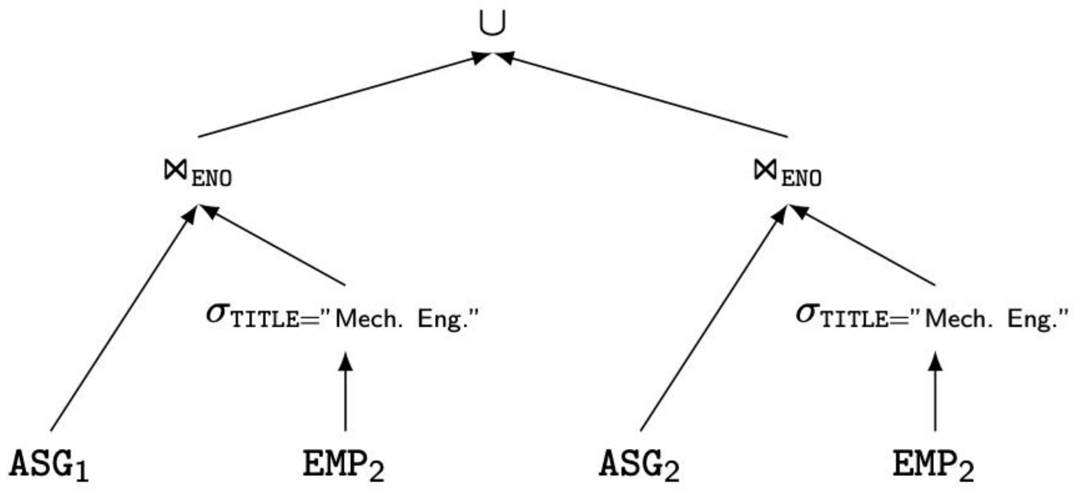


Selections first

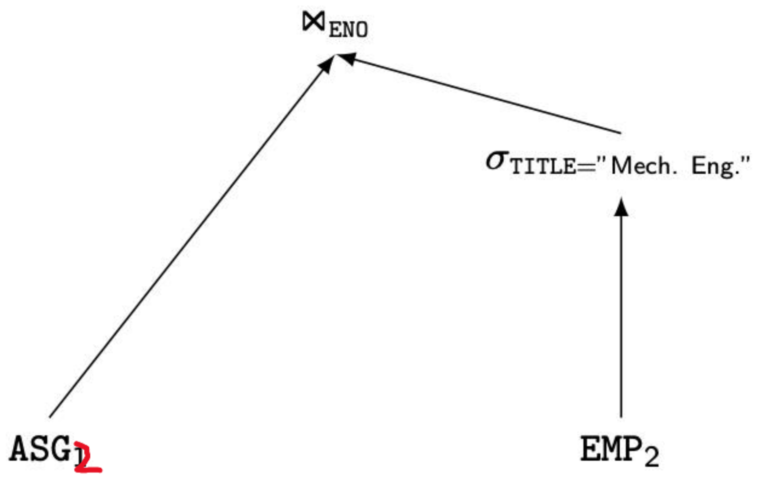


Reduction for DHF

Joins over unions



Elimination of the empty intermediate relations (left sub-tree)



Reduction for Hybrid Fragmentation

- Combine the rules already specified:
  - Remove empty relations generated by contradicting selections on horizontal fragments;
  - Remove useless relations generated by projections on vertical fragments;
  - Distribute joins over unions in order to isolate and remove useless joins.

Reduction for HF

Example

Consider the following hybrid fragmentation:

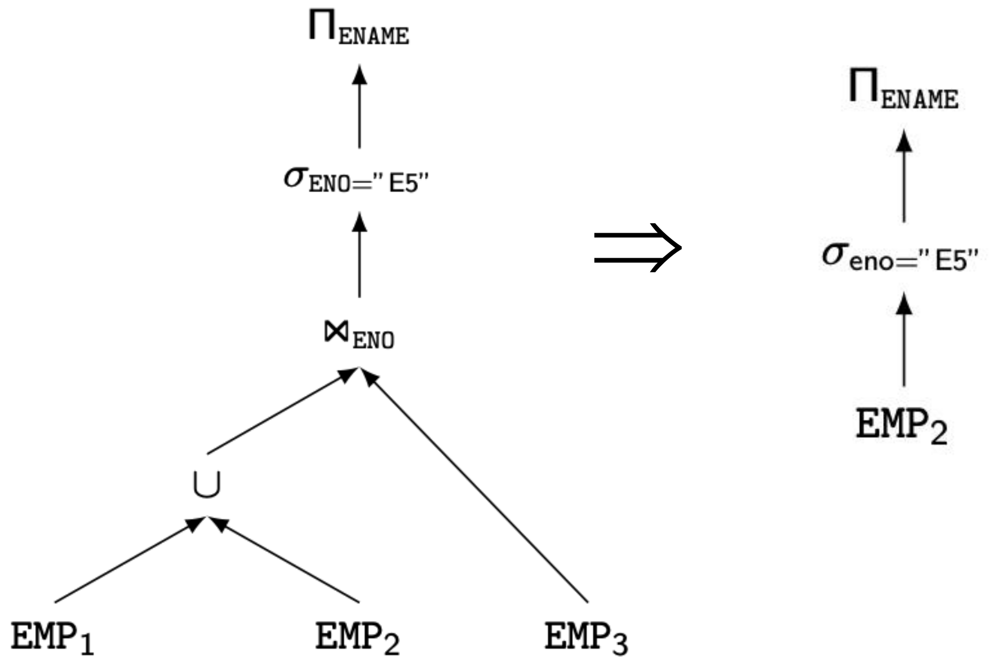
$EMP_1 = \sigma_{ENO \leq "E4"} (\Pi_{ENO,ENAME} (EMP))$

$EMP_2 = \sigma_{ENO > "E4"} (\Pi_{ENO,ENAME} (EMP))$

$EMP_3 = \sigma_{ENO,TITLE} (EMP)$

and the query

```
SELECT   ENAME
FROM     EMP
WHERE    ENO="E5"
```



Outline

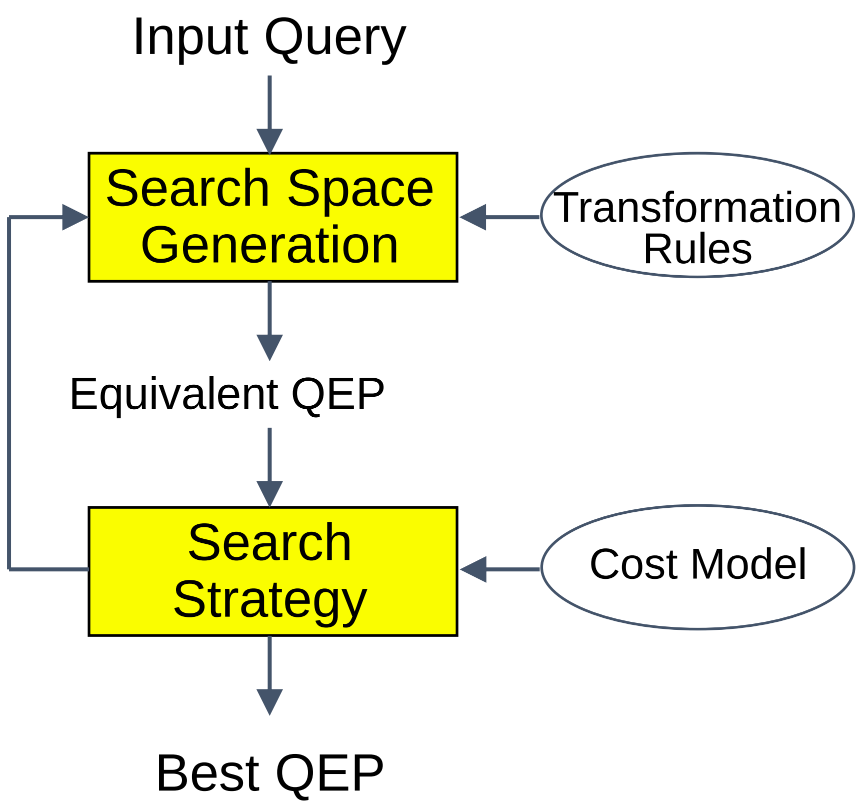
- Distributed Query Processing
    - Query Decomposition and Localization
    - Distributed Query Optimization
    - Join Ordering
    - Adaptive Query Processing

Step 3 – Global Query Optimization

Input: Fragment query

- Find the *best* (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
    - Bushy vs. linear trees
    - Which relation to ship where?
    - Ship-whole vs. ship-as-needed
  - Decide on the use of semi-joins
    - Semi-join saves on communication at the expense of more local processing
  - Join methods
    - Nested loop, merge join or hash join

Query Optimization Process



Components

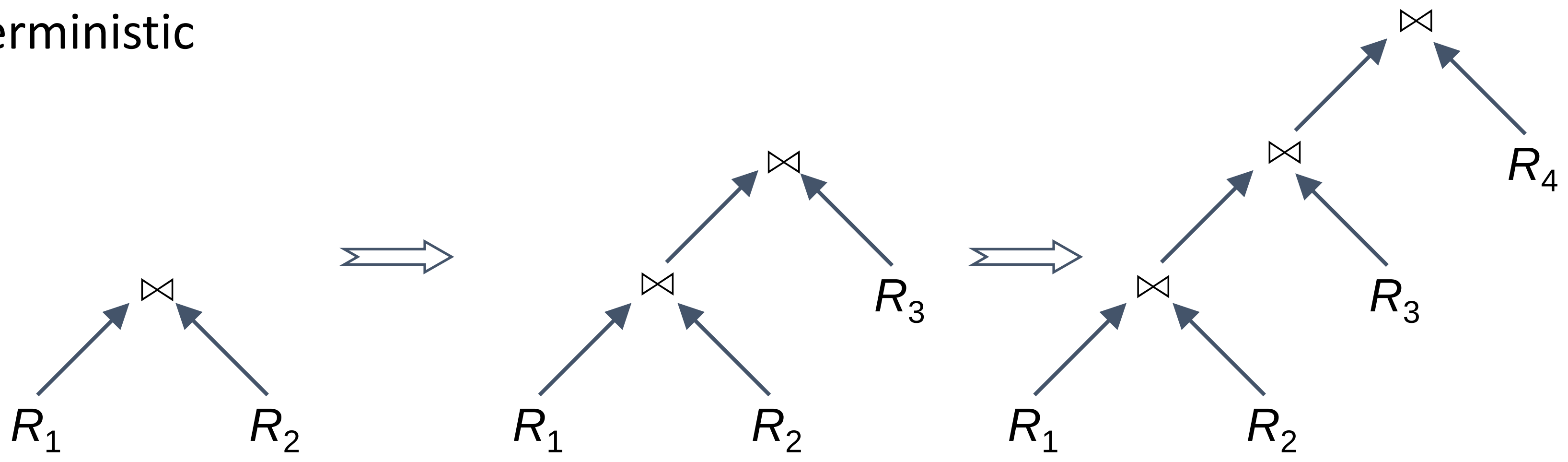
- Search space
    - The set of equivalent algebra expressions (query trees)
- Cost model
    - I/O cost + CPU cost + communication cost
    - These might have different weights in different distributed environments (LAN vs WAN)
    - Can also maximize throughput
- Search algorithm
    - How do we move inside the solution space?
    - Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic,…)

Search Strategy

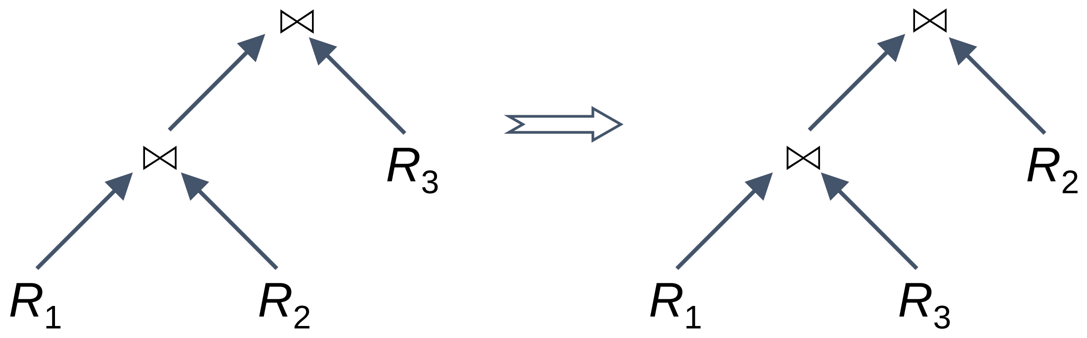
- How to "move" in the search space
- Deterministic
  - ➨Start from base relations and build plans by adding one relation at each step
  - ➨Dynamic programming: breadth-first
  - ➨Greedy: depth-first
- Randomized
  - ➨Search for optimalities around a particular starting point
  - ➨Trade optimization time for execution time
  - ➨Better when > 10 relations
  - ➨Simulated annealing
  - ➨Iterative improvement

Search Strategies

- Deterministic



- **Randomized**



⋈
⋈
R_1
R_2
R_3
⋈
⋈
R_1
R_3
R_2