## Distributed Query processing

Material from:
Principles of Distributed Database Systems
**Özsu**, M. Tamer, **Valduriez**, Patrick, 3rd ed. 2011
+ slides C. Roncancio



# Query Processing in a DDBMS

high level user query

↓

**query processor**

↓

Low-level data manipulation commands for D-DBMS

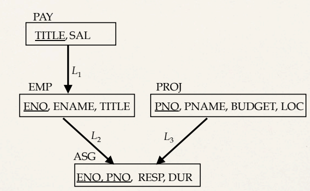


# Query Processing Components

- Query language that is used
  - SQL: “intergalactic dataspeak”
- Query execution methodology
  - 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



# Example






# Selecting Alternatives

```
SELECT   ENAME
FROM     EMP,ASG
WHERE    EMP.ENO = ASG.ENO
AND      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”



# Cost of Alternatives

- Assume
  - *size*(EMP) = 400
  - *size*(ASG) = 1000
  - 20 managers / uniform distribution
  - tuple access cost = 1 unit;
  - tuple transfer cost = 10 units
  - Index on `EMP.ENO`
  - Index on `ASG.RESP`

# Cost of Alternatives

- 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
  - Final union and projection 20 * tuple access cost 20

  Total Cost 460



# Cost of Alternatives

- 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
    - high protocol overhead
- Local area networks
  - communication cost not that dominant
  - total cost function should be considered
- Can also maximize throughput



# Complexity of Relational Operations

- Assume
  - relations of cardinality $n$
  - sequential scan

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



# Query Optimization Issues – 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 semijoins
  - Reorder operations to reduce intermediate relation size
  - Optimize individual operations



# Query Optimization Issues – 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

# Query Optimization Issues – 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
  - R*
- Dynamic
  - Run time optimization
  - Exact information on the intermediate relation sizes
  - Have to reoptimize for multiple executions
  - Distributed INGRES
- Hybrid
  - Compile using a static algorithm
  - If the error in estimate sizes > threshold, reoptimize at run time
  - Mermaid



# Query Optimization Issues – 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
- Common assumptions
  - Independence between different attribute values
  - Uniform distribution of attribute values within their domain

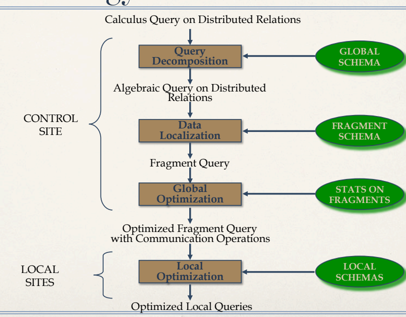


# Query Optimization Issues – Decision Sites

- 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



# Distributed Query Processing Methodology






# Outline

- Distributed Query Processing
  - Overview
  - Query decomposition and localization
  - Distributed query optimization



# Step 1 – Query Decomposition

Input : Calculus query on global relations

- Normalization
  - manipulate query quantifiers and qualification
- Analysis
  - detect and reject "incorrect" queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate redundant predicates
- Restructuring
  - calculus query ➔ algebraic query
  - more than one translation is possible
  - use transformation rules

# Normalization

- Lexical and syntactic analysis
  - check validity (similar to compilers)
  - check for attributes and relations
  - type checking on the qualification
- Put into normal form
  - Conjunctive normal form

    $$(p_{11} \vee p_{12} \vee \ldots \vee p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee \ldots \vee p_{mn})$$
  - Disjunctive normal form

    $$(p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n}) \vee \ldots \vee (p_{m1} \wedge p_{m2} \wedge \ldots \wedge p_{mn})$$
  - OR's mapped into union
  - AND's mapped into join or selection



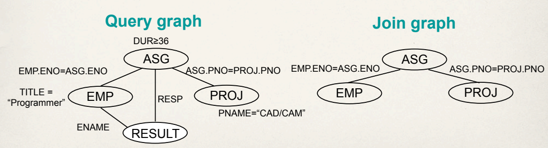
# Analysis

- Refute incorrect queries
- Type incorrect
  - If any of its attribute or relation names are not defined in the global schema
  - If operations are applied to attributes of the wrong type
- Semantically incorrect
  - Components do not contribute in any way to the generation of the result
  - Only a subset of relational calculus queries can be tested for correctness
  - Those that do not contain disjunction and negation
  - To detect
    - connection graph (query graph)
    - join graph



# Analysis – Example

```
SELECT  ENAME,RESP
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO = ASG.ENO
AND     ASG.PNO = PROJ.PNO
AND     PNAME = "CAD/CAM"
AND     DUR ≥ 36
AND     TITLE = "Programmer"
```

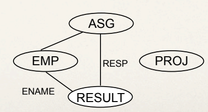





# Analysis

If the query graph is not connected, the query may be wrong or use Cartesian product

```
SELECT  ENAME,RESP
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO = ASG.ENO
AND     PNAME = "CAD/CAM"
AND     DUR > 36
AND     TITLE = "Programmer"
```






# Simplification

- Why simplify?
  - Remember the example
- How? Use transformation rules
  - Elimination of redundancy
    - idempotency rules

      $p_1 \wedge \neg(p_1) \Leftrightarrow \text{false}$

      $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$

      $p_1 \wedge \text{false} \Leftrightarrow p_1$

      …
  - Application of transitivity
  - Use of integrity rules



# Simplification – Example

```
SELECT  TITLE
FROM    EMP
WHERE   EMP.ENAME = "J. Doe"
OR      (NOT(EMP.TITLE = "Programmer")
AND     (EMP.TITLE = "Programmer"
OR      EMP.TITLE = "Elect. Eng.")
AND     NOT(EMP.TITLE = "Elect. Eng."))
```

## Simplification – Example

```
SELECT   TITLE
FROM     EMP
WHERE    EMP.ENAME = "J. Doe"
OR       (NOT(EMP.TITLE = "Programmer")
AND      (EMP.TITLE = "Programmer"
OR       EMP.TITLE = "Elect. Eng.")
AND      NOT(EMP.TITLE = "Elect. Eng."))
```

↓

```
SELECT   TITLE
FROM     EMP
WHERE    EMP.ENAME = "J. Doe"
```

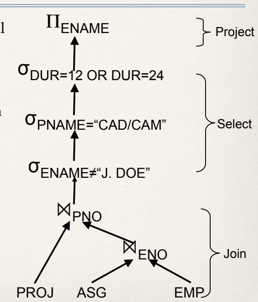


## Restructuring

- Convert relational calculus to relational algebra
- Make use of query trees
- Example

  Find the names of employees other than J. Doe who worked on the CAD/CAM project for either 1 or 2 years.

```
SELECT  ENAME
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO = ASG.ENO
AND     ASG.PNO = PROJ.PNO
AND     ENAME≠ "J. Doe"
AND     PNAME = "CAD/CAM"
AND     (DUR = 12 OR DUR = 24)
```






## Restructuring –Transformation Rules

- Commutativity of binary operations
  - ➡ $R \times S \Leftrightarrow S \times R$
  - ➡ $R \bowtie S \Leftrightarrow S \bowtie R$
  - ➡ $R \cup S \Leftrightarrow S \cup R$
- Associativity of binary operations
  - ➡ $(R \times S) \times T \Leftrightarrow R \times (S \times T)$
  - ➡ $(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$
- Idempotence of unary operations
  - ➡ $\Pi_{A'}(\Pi_{A'}(R)) \Leftrightarrow \Pi_{A'}(R)$
  - ➡ $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \Leftrightarrow \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$

  where $R[A]$ and $A' \subseteq A$, $A'' \subseteq A$ and $A' \subseteq A''$
- Commuting selection with projection



## Restructuring – Transformation Rules

- Commuting selection with binary operations
  - ➡ $\sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)}(R)) \times S$
  - ➡ $\sigma_{p(A_i)}(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow (\sigma_{p(A_i)}(R)) \bowtie_{(A_j,B_k)} S$
  - ➡ $\sigma_{p(A_i)}(R \cup T) \Leftrightarrow \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(T)$

  where $A_i$ belongs to $R$ and $T$
- Commuting projection with binary operations
  - ➡ $\Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S)$
  - ➡ $\Pi_C(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{(A_j,B_k)} \Pi_{B'}(S)$
  - ➡ $\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$

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



## Example

Recall the previous example:

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, ASG, EMP
WHERE  ASG.ENO=EMP.ENO
AND    ASG.PNO=PROJ.PNO
AND    ENAME ≠ "J. Doe"
AND    PROJ.PNAME="CAD/CAM"
AND    (DUR=12 OR DUR=24)
```

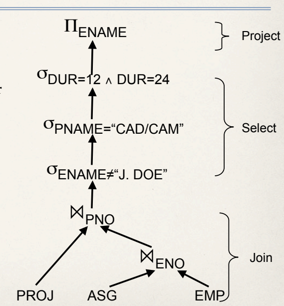


## Example

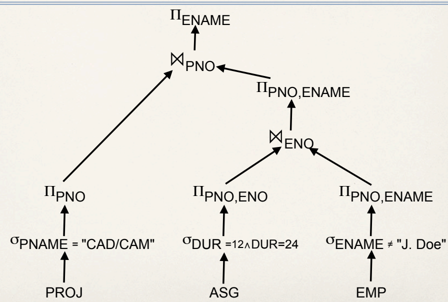
Recall the previous example:

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, ASG, EMP
WHERE  ASG.ENO=EMP.ENO
AND    ASG.PNO=PROJ.PNO
AND    ENAME ≠ "J. Doe"
AND    PROJ.PNAME="CAD/CAM"
AND    (DUR=12 OR DUR=24)
```



# Restructuring






# 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 into $EMP_1$, $EMP_2$, $EMP_3$ 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 into $ASG_1$ and $ASG_2$ as follows:
  - $ASG_1 = \sigma_{ENO \leq \text{"E3"}}(ASG)$
  - $ASG_2 = \sigma_{ENO > \text{"E3"}}(ASG)$

Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$ and ASG by $(ASG_1 \cup ASG_2)$ in any query

$ASG_2$

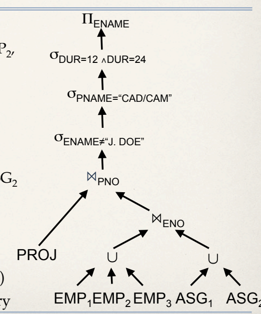


# Example

Assume

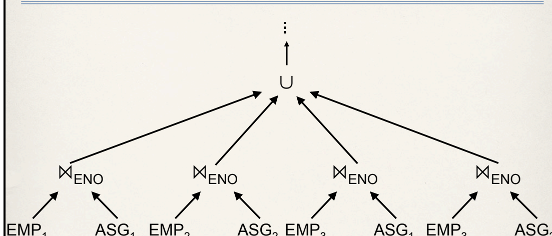
- EMP is fragmented into $EMP_1$, $EMP_2$, $EMP_3$ as follows:
  - $EMP_1 = \sigma_{ENO \leq \text{"E3"}}(EMP)$
  - $EMP_2 = \sigma_{\text{"E3"} < ENO \leq \text{"E6"}}(EMP)$
  - $EMP_3 = \sigma_{ENO \geq \text{"E6"}}(EMP)$
- ASG fragmented into $ASG_1$ and $ASG_2$ as follows:
  - $ASG_1 = \sigma_{ENO \leq \text{"E3"}}(ASG)$
  - $ASG_2 = \sigma_{ENO > \text{"E3"}}(ASG)$

Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$ and ASG by $(ASG_1 \cup ASG_2)$ in any query



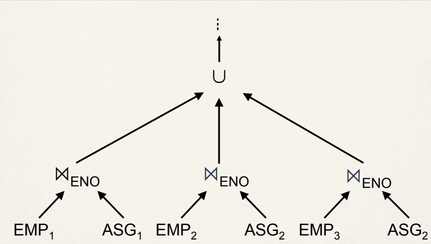



# Provides Parallellism






# Eliminates Unnecessary Work

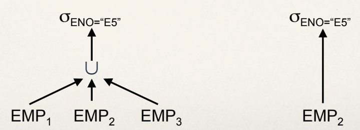


# Reduction for PHF

- 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 $R$: $\neg(p_i(x) \wedge p_j(x))$
  - Example

```
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 $R_j = \sigma_{p_j}(R)$

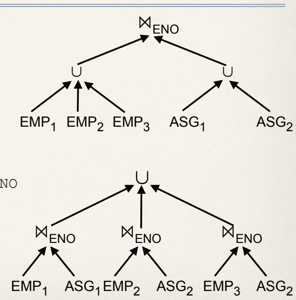


# Reduction for PHF

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

```
SELECT   *
FROM     EMP,ASG
WHERE    EMP.ENO=ASG.ENO
```

- Distribute join over unions
- Apply the reduction rule






# Reduction for VF

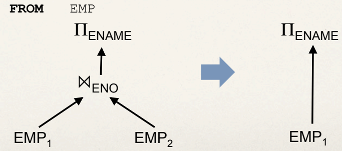
- Find useless (not empty) intermediate relations

  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

  $ASG_1$: $ASG \ltimes_{ENO} EMP_1$

  $ASG_2$: $ASG \ltimes_{ENO} EMP_2$

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

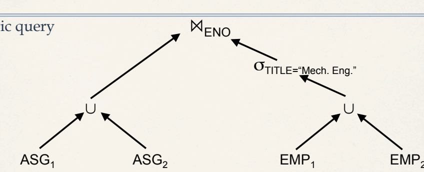
  $EMP_2$: $\sigma_{TITLE <> \text{"Programmer"}}(EMP)$
- Query

```
SELECT   *
FROM     EMP, ASG
WHERE    ASG.ENO = EMP.ENO
AND      EMP.TITLE = "Mech. Eng."
```

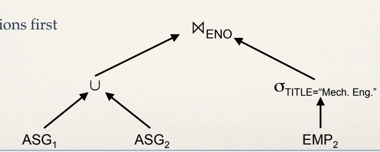


# Reduction for DHF

Generic query

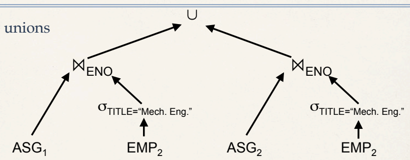



Selections first

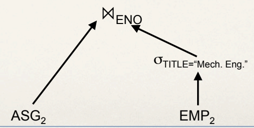


## Reduction for DHF

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:

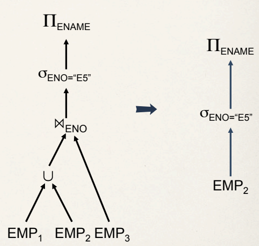
$EMP_1 = \sigma_{ENO \leq "E4"} (\Pi_{ENO,ENAME} (EMP))$

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

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

and the query

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





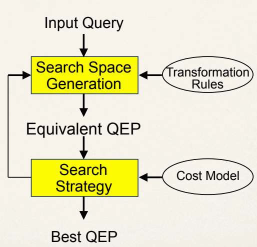

## Step 3 – Global Query Optimization

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 semijoins
    - Semijoin saves on communication at the expense of more local processing.
  - Join methods
    - nested loop vs ordered joins (merge join or hash join)



## Query Optimization Process

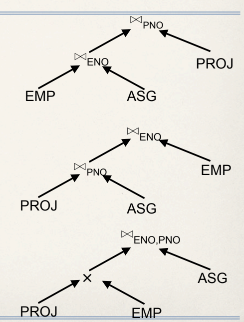





## Search Space

- Search space characterized by alternative execution
- Focus on join trees
- For $N$ relations, there are $O(N!)$ equivalent join trees that can be obtained by applying commutativity and associativity rules

```
SELECT ENAME,RESP
FROM   EMP, ASG,PROJ
WHERE  EMP.ENO=ASG.ENO
AND    ASG.PNO=PROJ.PNO
```



## Information about data

- Relations & fragments
- Size and cardinality
- Per attribut :
  - cardinality,
  - max and min values ,
  - distribution,
  - size,
  - effectif distinct values



## Estimation of the size of intermediate results

- Selectivity factor, SF … similar to centralised systems
  - card($\sigma_c$ R) = card(R) * SF( c)
  - card(R * S) = card(R) * card(S) * SF of join condition
- Statistiques issued from previous executions



## Joins!

- The evaluation cost is « dominated » by data transfers
- Where do we evaluate joins ?
- Exemple : E@S1 *G@S2 * J@S3
  - E ---> S2; join ; Res ---> S3
  - G ---> S1; join; Res ---> S3
  - G ---> S3; join; Res ---> S1
  - J ---> S2; join; Res ---> S1
  - E, J ---> S2; joins



## Semi-join / join

- Important technique
- Given R(X), S( Y), condition c over attributes Z
- R αc S = Πx ( R *c S)
- R *c S = (R αc S) *c S = (R αc S) *c (S αc R)
- Idea:
  - use semi-joins to evaluate joins
  - Transfer less data



## Technique using Semi-joins

- Example : R@S1 * S@S2
  - Πz (S) ---> S1
  - At S1: R αc S; Res ---> S2
  - Finish at S2 : Res *c S
- Transferred data:
  - size(Πz (S)) + size(R αc S)
- Transferred data without semi-joins:
  - size(R)



## Discussion…

- size(Πz (S)) + size (R αc S) vs size(R)
- Depends on the selectivity factor
- More treatment:
  - One relation is used twice
  - No index on intermediate results
- In some cases …less transferts

## Semi-joins & bit vectors

- R@S1 αc S@S2 , condition over attributes Z
- Use a hash function
  - F : dom(Z) --> [1,N]
- Evaluate @S2: F(Πz (S))
  - This is a sub-set of [1,N]
- Represent as a bit vector and send it to S1
- Evaluate @S1: R1 and send back to S2
  - R1 = { r de R / F(r.z) ε F(Πz (S)) }
  - R α S is a sub-set of R1 which is a sub-set of R
- Evaluate @ S2 final result…

Slides, C. L. Roncancio 57

## Semi-joins & bit vectors:Discussion

- Pros: less transfers
- Cons:
  - More I/O:2 access to S.
  - Hash function: collisions?
- Variants:
  - Compresion of the vector
  - Use several vectors
  - Send vectors in the «two-ways »
  - …

Slides, C. L. Roncancio 59

## Conclusion & perspectives

- Extension of centralized techniques
- Many algorithms!!!
- Complexity is increased by
  - Autonomy of the participants
  - Increased number of participants
  - Large scale
- What about energy consumption issues?

Slides, C. L. Roncancio 60