

Distributed databases

Acknowledgements to A. Kemper and Eickler



Distributed Databases

- Motivation:
 - Former times:
 - Bank with subsidiaries
 - Nowadays:
 - Virtual organization

- + Subsidiaries should work on data of local customers
- + Central site should be able to access all data

Federation:

- ◆ Commercial database systems (DB2...)
- ◆ Networked Graphs in RDF/SPARQL

Distribution in the Cloud

- ◆ Large scale data processing
- ◆ RDF (current diploma thesis)

Distributed Database

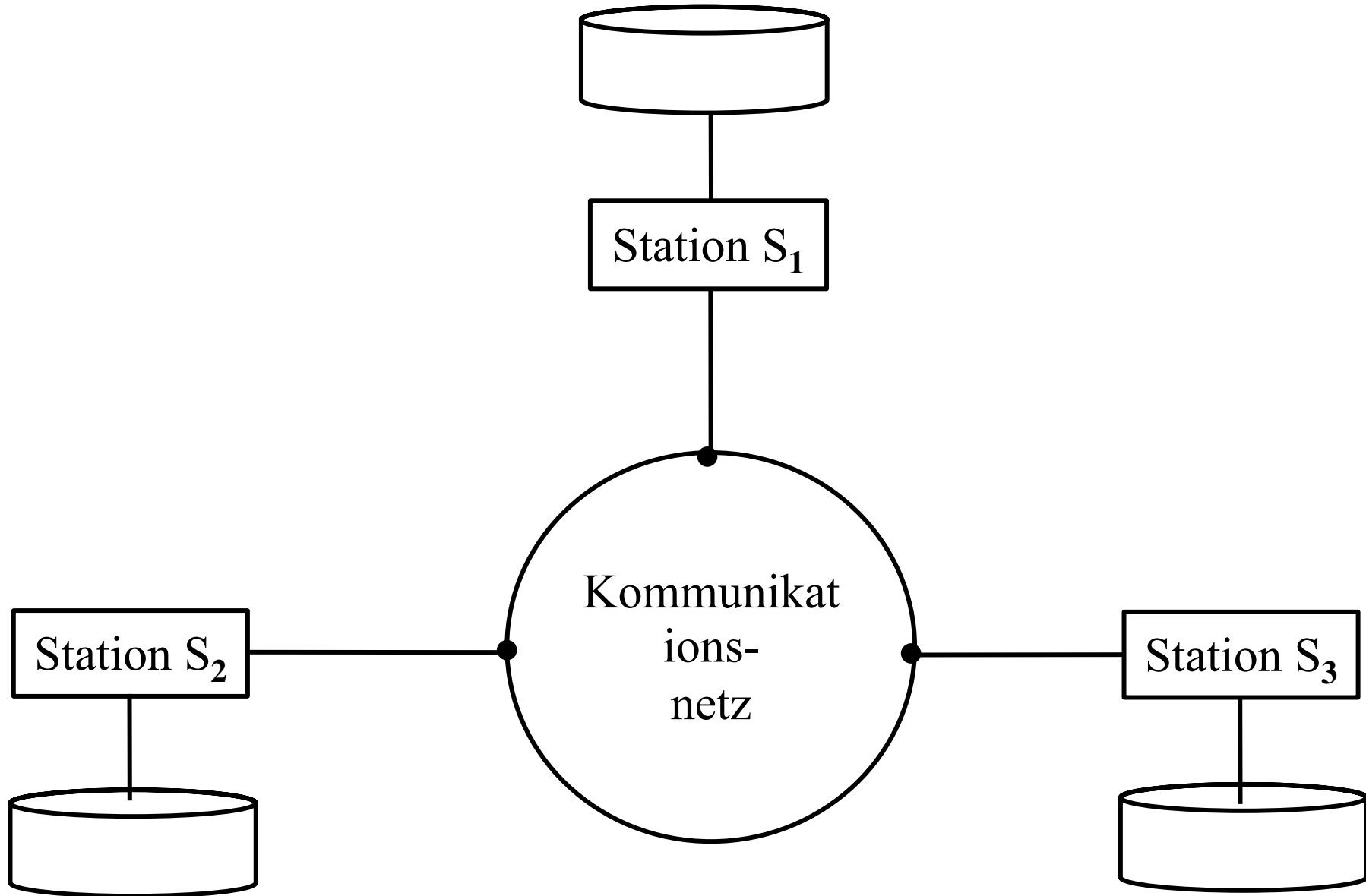
- ◆ Collection of information units, distributed on multiple computers connected with communication net

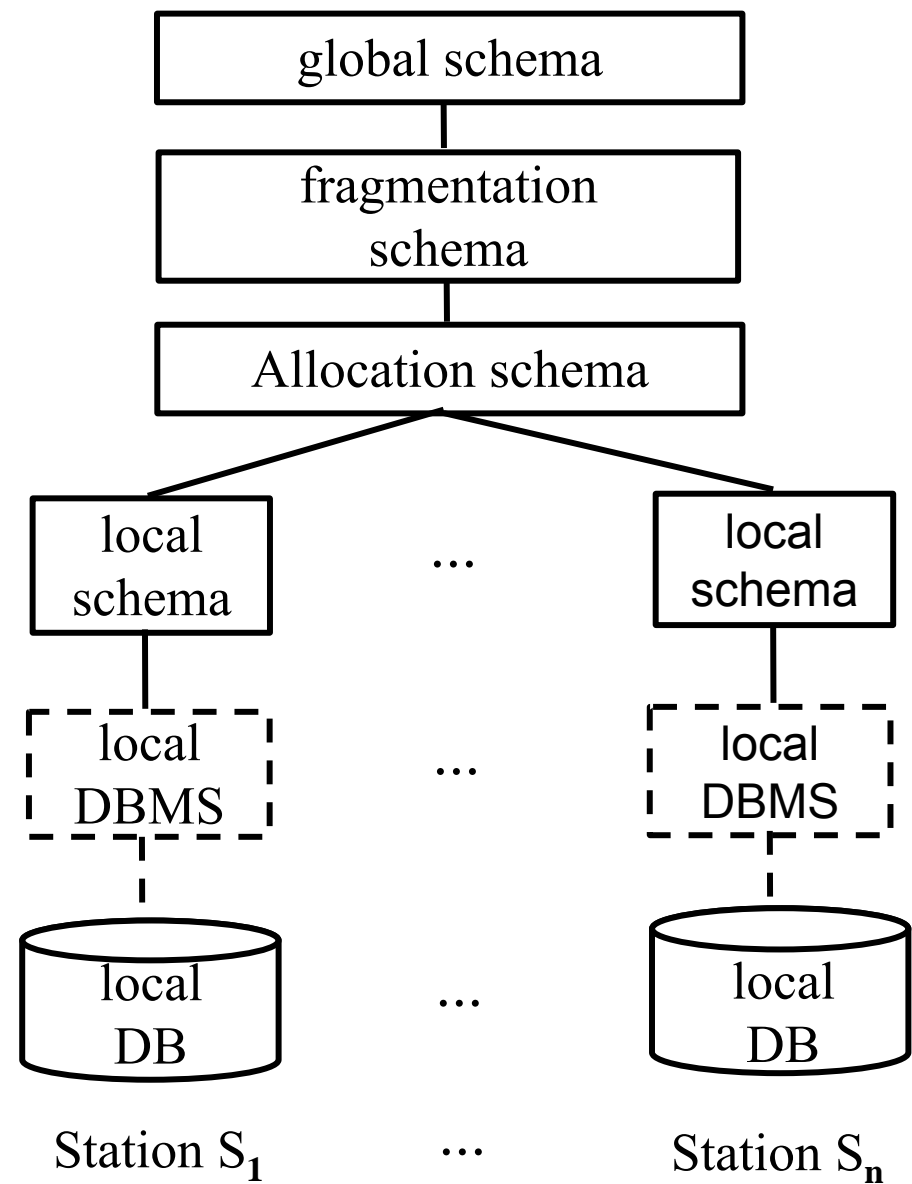
→ *Ceri & Pelagatti (1984)*

Cooperation between autonomously working stations for performing global tasks

Loosely integrated databases / Multi-database systems

- ◆ Autonomous working
- ◆ Autonomous structuring/schema
- ◆ No joint administration

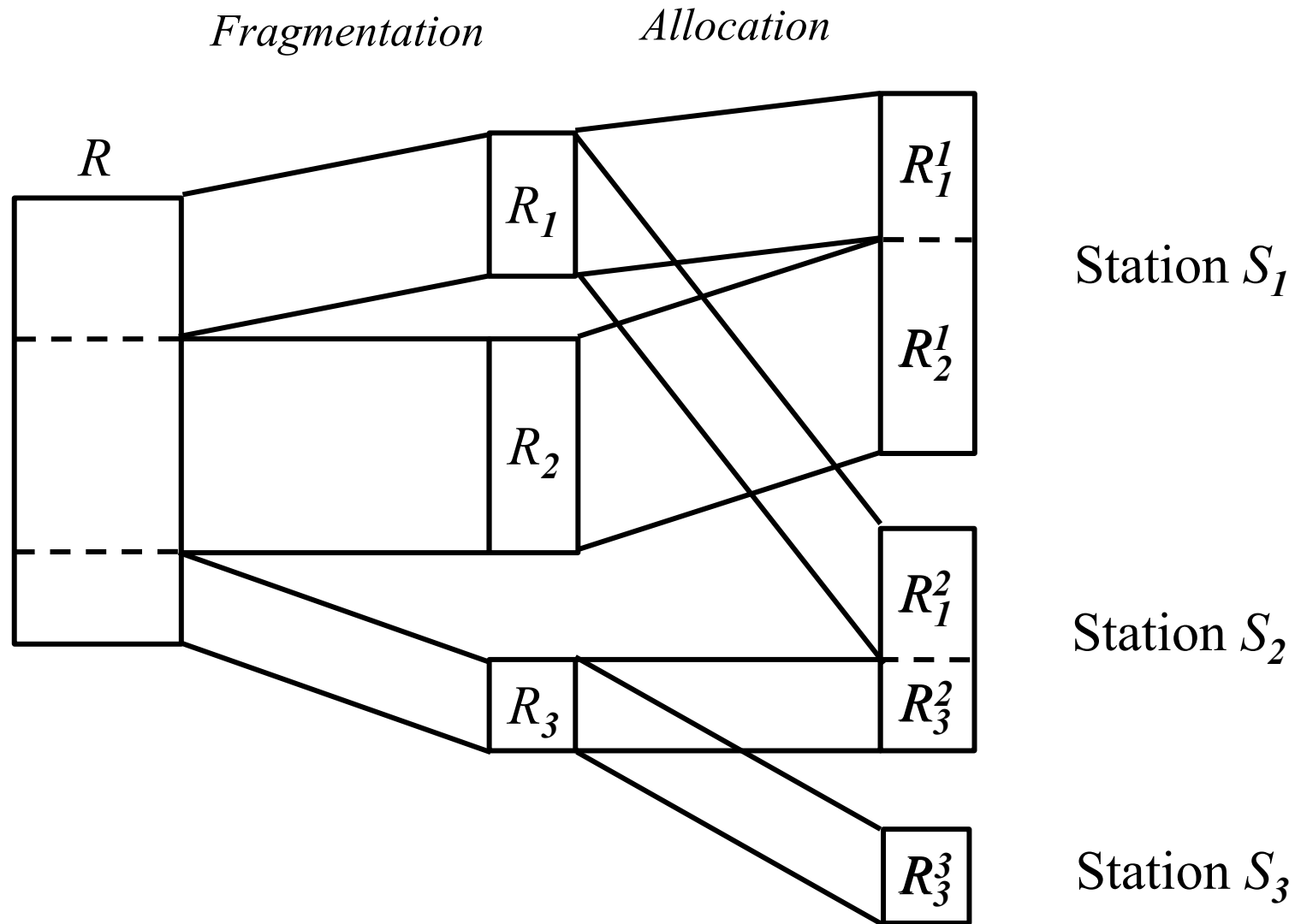




Fragmentation: Fragments contain data with likewise access patterns

Allocation: Fragments are assigned to the stations

- with replication
- without replication



horizontal fragmentation:

Partitioning the relation in disjoint tuple sets

vertical fragmentation:

Summarization of attributes with likewise access patterns

Combined fragmentation: Application of horizontal and vertical fragmentation on one relation

Reconstructable

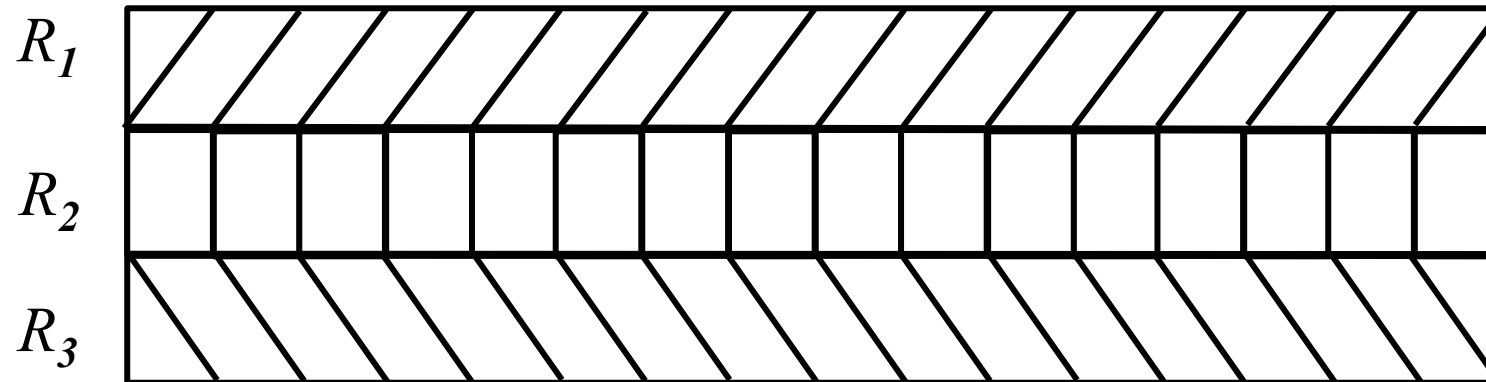
Complete

Disjoint (to some extent)

| Professors | | | | | | |
|------------|------------|------|------|-------------|--------|--------------|
| PersNr | Name | Rang | Raum | Fakultät | Gehalt | Steuerklasse |
| 2125 | Sokrates | C4 | 226 | Philosophie | 85000 | 1 |
| 2126 | Russel | C4 | 232 | Philosophie | 80000 | 3 |
| 2127 | Kopernikus | C3 | 310 | Physik | 65000 | 5 |
| 2133 | Popper | C3 | 52 | Philosophie | 68000 | 1 |
| 2134 | Augustinus | C3 | 309 | Theologie | 55000 | 5 |
| 2136 | Curie | C4 | 36 | Physik | 95000 | 3 |
| 2137 | Kant | C4 | 7 | Philosophie | 98000 | 1 |

Abstract :

R



For 2 predicates p_1 and p_2 there are 4 fragments:

$$R1 := \sigma_{p_1 \wedge p_2}(R)$$

$$R2 := \sigma_{p_1 \wedge \neg p_2}(R)$$

$$R3 := \sigma_{\neg p_1 \wedge p_2}(R)$$

$$R4 := \sigma_{\neg p_1 \wedge \neg p_2}(R)$$



n partitioning predicates p_1, \dots, p_n yield 2^n fragments

→ 3 fragmentation predicates:

$p_1 \equiv \text{Fakultät} = \text{'Theologie'}$

$p_2 \equiv \text{Fakultät} = \text{'Physik'}$

$p_3 \equiv \text{Fakultät} = \text{'Philosophie'}$

$\text{TheolProfs}' := \sigma_{p_1 \wedge \neg p_2 \wedge \neg p_3}(\text{Professoren}) = \sigma_{p_1}(\text{Professoren})$

$\text{PhysikProfs}' := \sigma_{\neg p_1 \wedge p_2 \wedge \neg p_3}(\text{Professoren}) = \sigma_{p_2}(\text{Professoren})$

$\text{PhiloProfs}' := \sigma_{\neg p_1 \wedge \neg p_2 \wedge p_3}(\text{Professoren}) = \sigma_{p_3}(\text{Professoren})$

$\text{AndereProfs}' := \sigma_{\neg p_1 \wedge \neg p_2 \wedge \neg p_3}(\text{Professoren})$

| Professoren | | | |
|-------------|------------|------|------|
| PersNr | Name | Rang | Raum |
| 2125 | Sokrates | C4 | 226 |
| 2126 | Russel | C4 | 232 |
| 2127 | Kopernikus | C3 | 310 |
| 2133 | Popper | C3 | 52 |
| 2134 | Augustinus | C3 | 309 |
| 2136 | Curie | C4 | 36 |
| 2137 | Kant | C4 | 7 |

| Vorlesungen | | | |
|-------------|----------------------|-----|-------------|
| VorlNr | Titel | SWS | Gelesen Von |
| 5001 | Grundzüge | 4 | 2137 |
| 5041 | Ethik | 4 | 2125 |
| 5043 | Erkenntnistheorie | 3 | 2126 |
| 5049 | Mäeutik | 2 | 2125 |
| 4052 | Logik | 4 | 2125 |
| 5052 | Wissenschaftstheorie | 3 | 2126 |
| 5216 | Bioethik | 2 | 2126 |
| 5259 | Der Wiener Kreis | 2 | 2133 |
| 5022 | Glaube und Wissen | 2 | 2134 |
| 4630 | Die 3 Kritiken | 4 | 2137 |

Example *Vorlesungen* from the university schema:
Fragmentation into groups with likewise SWS-Zahl

2SWSVorls := $\sigma_{\text{SWS}=2}$ (Vorlesungen)

3SWSVorls := $\sigma_{\text{SWS}=3}$ (Vorlesungen)

4SWSVorls := $\sigma_{\text{SWS}=4}$ (Vorlesungen)

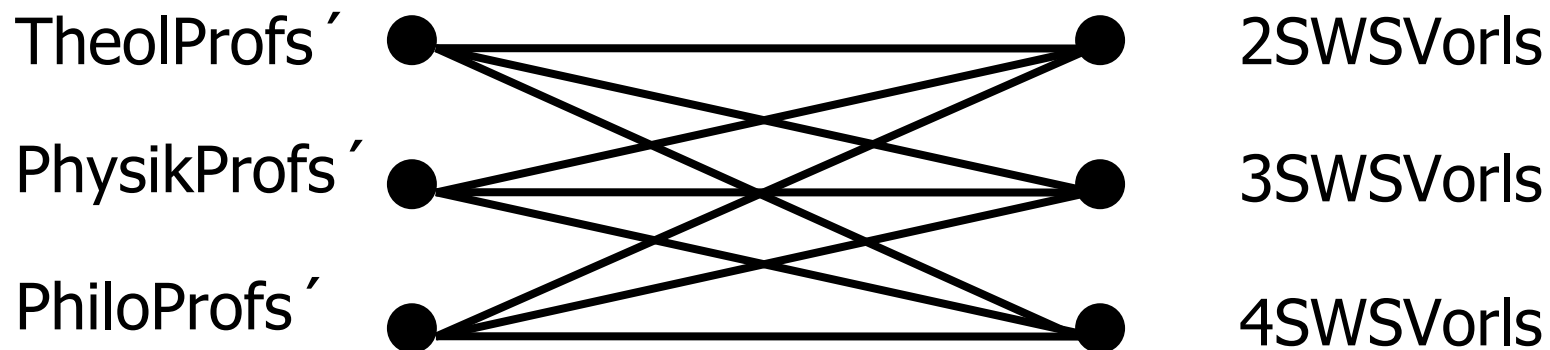
➔ Unsuitable fragmentation for querying

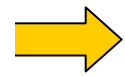
```
select Titel, Name  
from Vorlesungen, Professoren  
where gelesenVon = PersNr;
```

resultsIn:

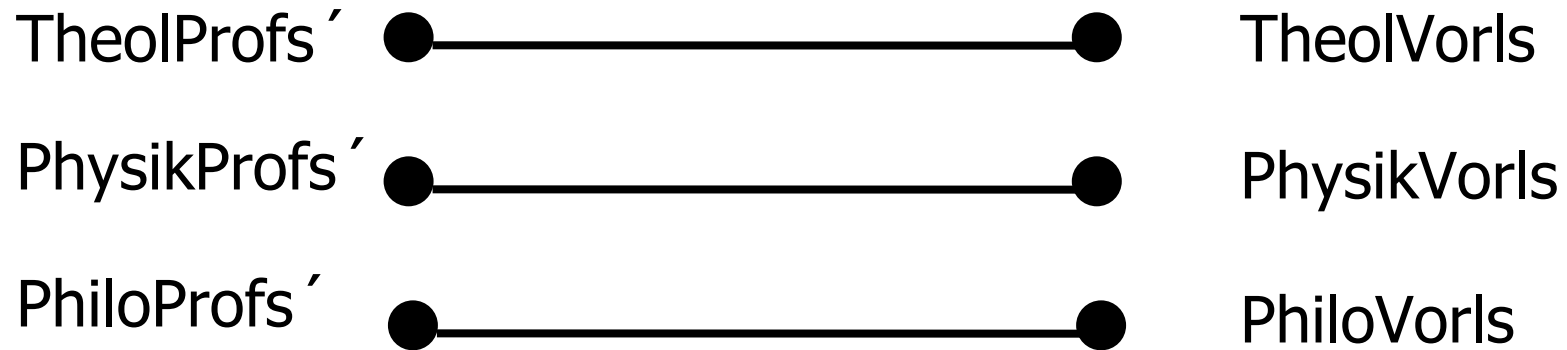
$$\Pi_{\text{Titel, Name}}((\text{TheolProfs}' \bowtie_{\text{q}} \text{2SWSVorls}) \cup$$
$$(\text{TheolProfs}' \bowtie \text{3SWSVorls}) \cup$$
$$\dots \cup (\text{PhiloProfs}' \bowtie \text{4SWSVorls}))$$

Join-Graph for this problem:





Solution: derived fragmentation [isweb](http://isweb.org)



TheolVorls := Vorlesungen $\bowtie_{\text{gelesenVon=PersNr}}$ TheolProfs'

PhysikVorls := Vorlesungen $\bowtie_{\text{gelesenVon=PersNr}}$ PhysikProfs'

PhiloVorls := Vorlesungen $\bowtie_{\text{gelesenVon=PersNr}}$ PhiloProfs'

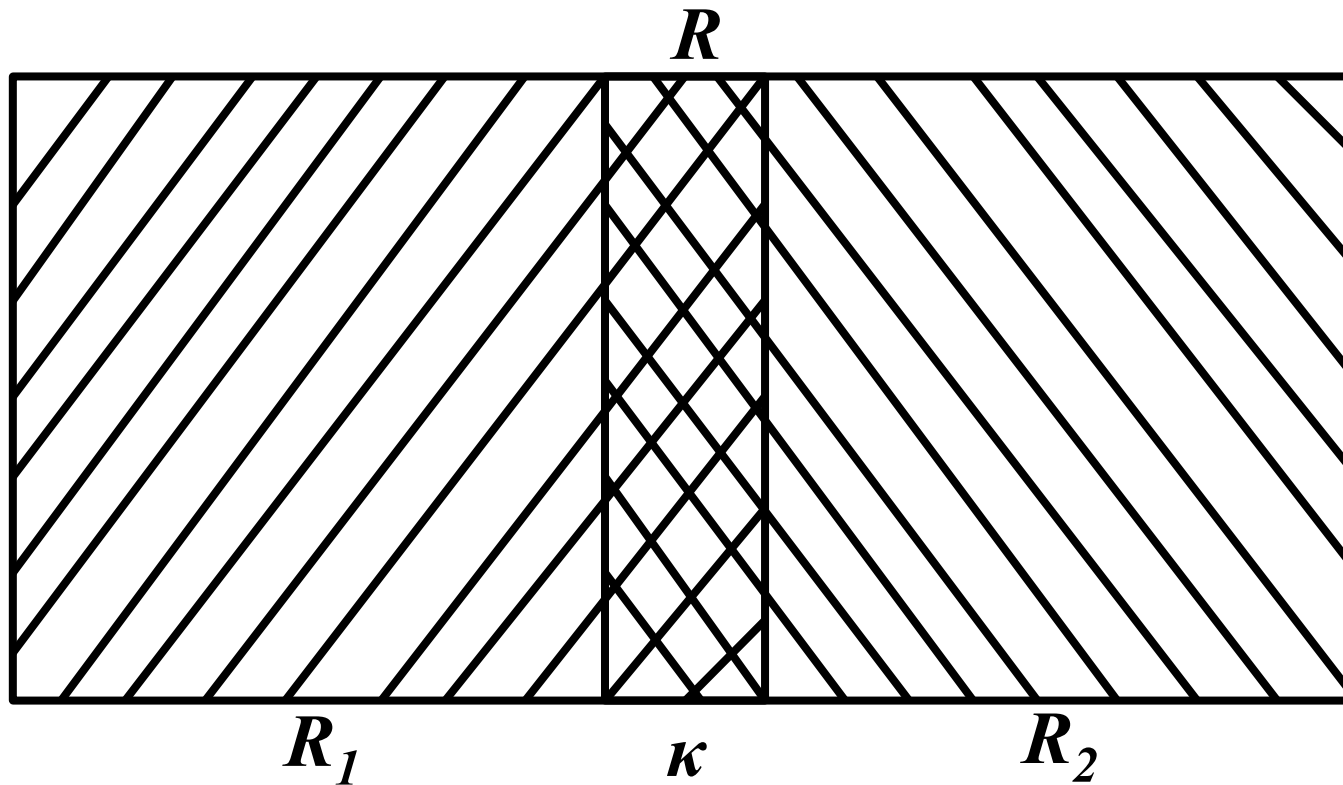
$$\Pi_{\text{Titel, Name}}((\text{TheolProfs}' \bowtie_p \text{TheolVorls}) \cup$$

$$(\text{PhysikProfs}' \bowtie_p \text{PhysikVorls}) \cup$$

$$(\text{PhiloProfs}' \bowtie_p \text{PhiloVorls}))$$

with $p \equiv (\text{PersNr} = \text{gelesenVon})$

Abstract representation:



Arbitrary vertical fragmentation does *not guarantee reconstructability*

2 possible approaches to ensure reconstructability:

Each fragment contains the primary key of the original relation – destroy *disjointness*

Each tuple of the original relation is assigned a unique *surrogate* (= artificially generated object identifier), which is part of each vertical fragment of a tuple

The university administration is interested in:
PersNr, Name, Gehalt (salary) and Steuerklasse (taxation category)

ProfVerw := $\Pi_{\text{PersNr, Name, Gehalt, Steuerklasse}}$ (Professoren)

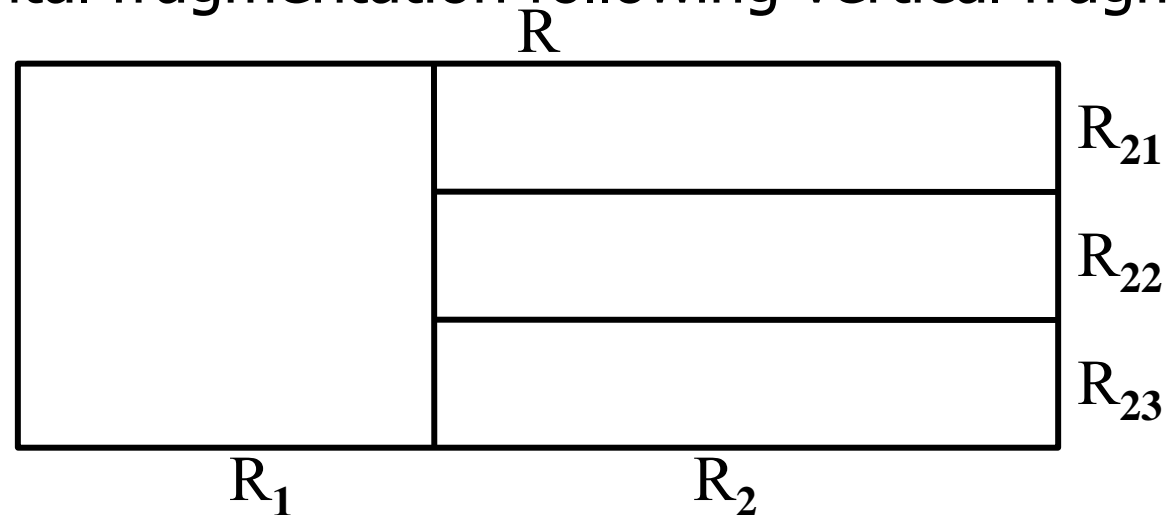
Teaching and research relates only to
PersNr, Name, Rang (status), Raum (room) and Fakultät (faculty):

Profs := $\Pi_{\text{PersNr, Name, Rang, Raum, Fakultät}}$ (Professoren)

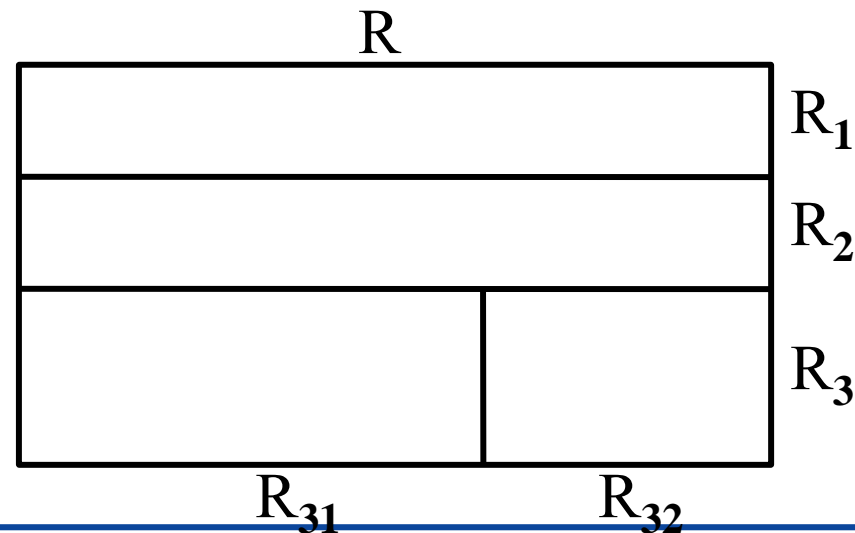
Reconstituting the original relation *Professoren*:

Professoren = ProfVerw $\bowtie_{\text{ProfVerw.PersNr} = \text{Profs.PersNr}}$ Profs

a) horizontal fragmentation following vertical fragmentation



b) vertical fragmentation following horizontal fragmentation



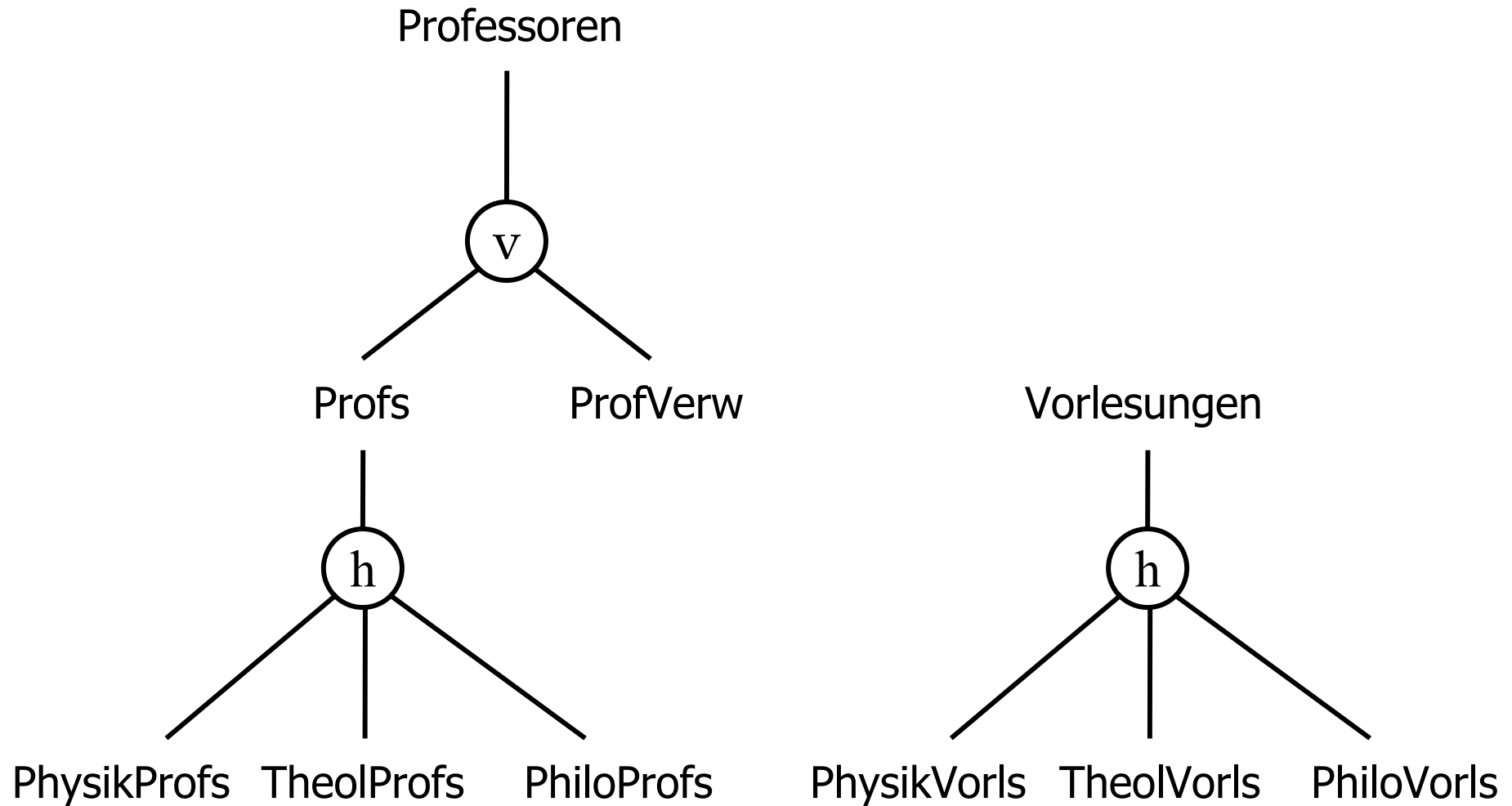
Case a)

$$R = R_1 \cup D_p (R_{21} \cup R_{22} \cup R_{23})$$

Case b)

$$R = R_1 \cup R_2 \cup (R_{31} \cup D_{R_{31} \cdot \kappa = R_{32} \cdot \kappa} R_{32})$$

Tree representations of fragmentations (example)



Individual fragments can be allocated to multiple nodes

Allocation for our example without replication

⇒ **redundancy-free** allocation

| Node | Remark | Allocated fragments |
|---------------------|--------------------------|--------------------------------|
| S_{Verw} | Administration | $\{ProfVerw\}$ |
| S_{Physik} | Dean's office Physik | $\{PhysikVorls, PhysikProfs\}$ |
| S_{Philo} | Dean's office Philosophy | $\{PhiloVorls, PhiloProfs\}$ |
| S_{Theol} | Dean's office Theology | $\{TheolVorls, TheolProfs\}$ |

Degree of transparency that a distributed database management system gives to the user for accessing distributed data

Different degrees of transparency:

- ◆ Fragmentation transparency
- ◆ Allocation transparency
- ◆ Local schema transparency

Example query requiring fragmentation transparency:

```
select Titel, Name  
from Vorlesungen, Professoren  
where gelesenVon = PersNr;
```

Example for change operation requiring fragmentation transparency:

```
update Professoren  
    set Fakultät = ‚Theologie‘  
    where Name = ‚Sokrates‘;
```

Changing the attribute value of *Fakultät*

Transferring the Sokrates-Tuple from fragment *PhiloProfs* into fragment *TheolProfs* (= deleting from *PhiloProfs*, inserting into *TheolProfs*)

Updating derived fragmentations of *Vorlesungen* (= Inserting lectures given by Sokrates into *TheolVorls*, deleting his lectures from *PhiloVorls*)

User must know fragmentation, but not their “location”

Example query:

```
select Gehalt  
from ProfVerw  
where Name = ‚Sokrates‘;
```

Original relation must remain reconstructable

Example:

Administration wants to know how much C4 professors earn in theology

Due to lack of fragmentation transparency the query must be reformulated:

```
select sum (Gehalt)  
from ProfVerw, TheolProfs  
where ProfVerw.PersNr = TheolProfs.PersNr and  
      Rang = ,C4`;
```

The user must know the computing node, which is the location of the fragment.

Example query:

```
select Name  
from TheolProfs at  $S_{Theol}$   
where Rang = ‚C3‘;
```

Local schema transparency requires that all nodes use the same data model and query language.

⇒ previous query may also be executed on the analogous computing node S_{Philo}

This is not possible if different DBMS are linked together

Use of different data models at local DBMS are called „Multi-Database-Systems“

Premise: Fragmentation transparency

Task of the query translator: generation of a query execution plan on fragments

Task of the query planner: generation of an efficient query execution plan → depending on the allocation of fragments to different computing nodes

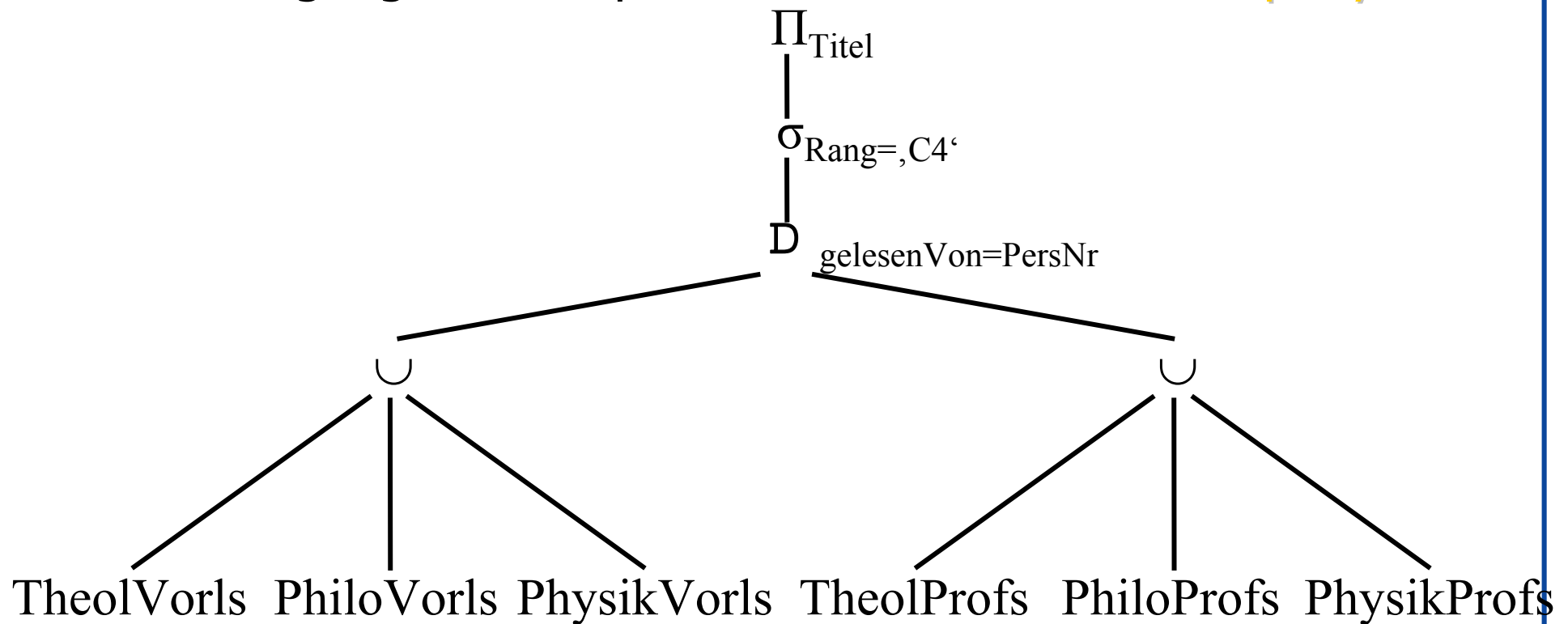
Translation of SQL query to global schema into an equivalent query on fragments requires two steps:

1. Reconstruction of all global relations needed in the query from their fragments. Result is an source algebraic expression
2. Applying the query algebraic expression on the source algebraic expression.

Example

```
select Titel
from Vorlesungen, Profs
where gelesenVon = PersNr and
      Rang = ,C4';
```

The resulting algebraic expression is called **canonical query form**:



For efficient query execution the optimizer uses the following property:

$$(R_1 \cup R_2) \bowtie_p (S_1 \cup S_2) = (R_1 \bowtie_p S_1) \cup (R_1 \bowtie_p S_2) \cup (R_2 \bowtie_p S_1) \cup (R_2 \bowtie_p S_2)$$

The generalization to n horizontal fragments R_1, \dots, R_n of R and m fragments S_1, \dots, S_m of S results in:

$$(R_1 \cup \dots \cup R_n) \bowtie_p (S_1 \cup \dots \cup S_m) = \bigcup_{1 \leq i \leq n} \bigcup_{1 \leq j \leq m} (R_i \bowtie_p S_j)$$

If: $S_i = S \bowtie_p R_i$ mit $S = S_1 \cup \dots \cup S_n$

Then:

$$R_i \bowtie_p S_j = \emptyset \text{ für } i \neq j$$

For a derived horizontal fragmentation of S:

$$(R_1 \cup \dots \cup R_n) \bowtie_p (S_1 \cup \dots \cup S_m) = \\ (R_1 \bowtie_p S_1) \cup (R_2 \bowtie_p S_2) \cup \dots \cup (R_n \bowtie_p S_n)$$

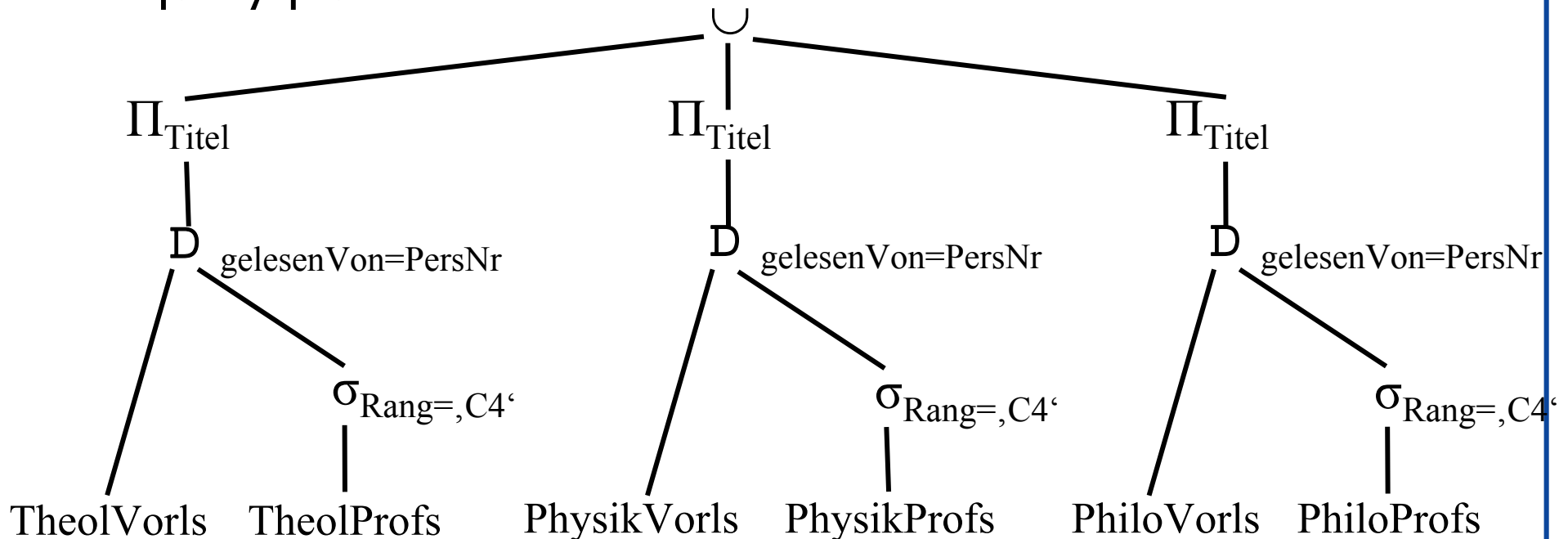
For our example:

$$(\text{TheolVorls} \cup \text{PhysikVorls} \cup \text{PhiloVorls}) \bowtie_{\dots} \\ (\text{TheolProfes} \cup \text{PhysikProfes} \cup \text{PhiloProfes})$$

To push down selections and projections in the query execution tree the following rules apply:

$$\sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2) \\ \Pi_L(R_1 \cup R_2) = \Pi_L(R_1) \cup \Pi_L(R_2)$$

Applying these algebraic rules generates the following query plan:

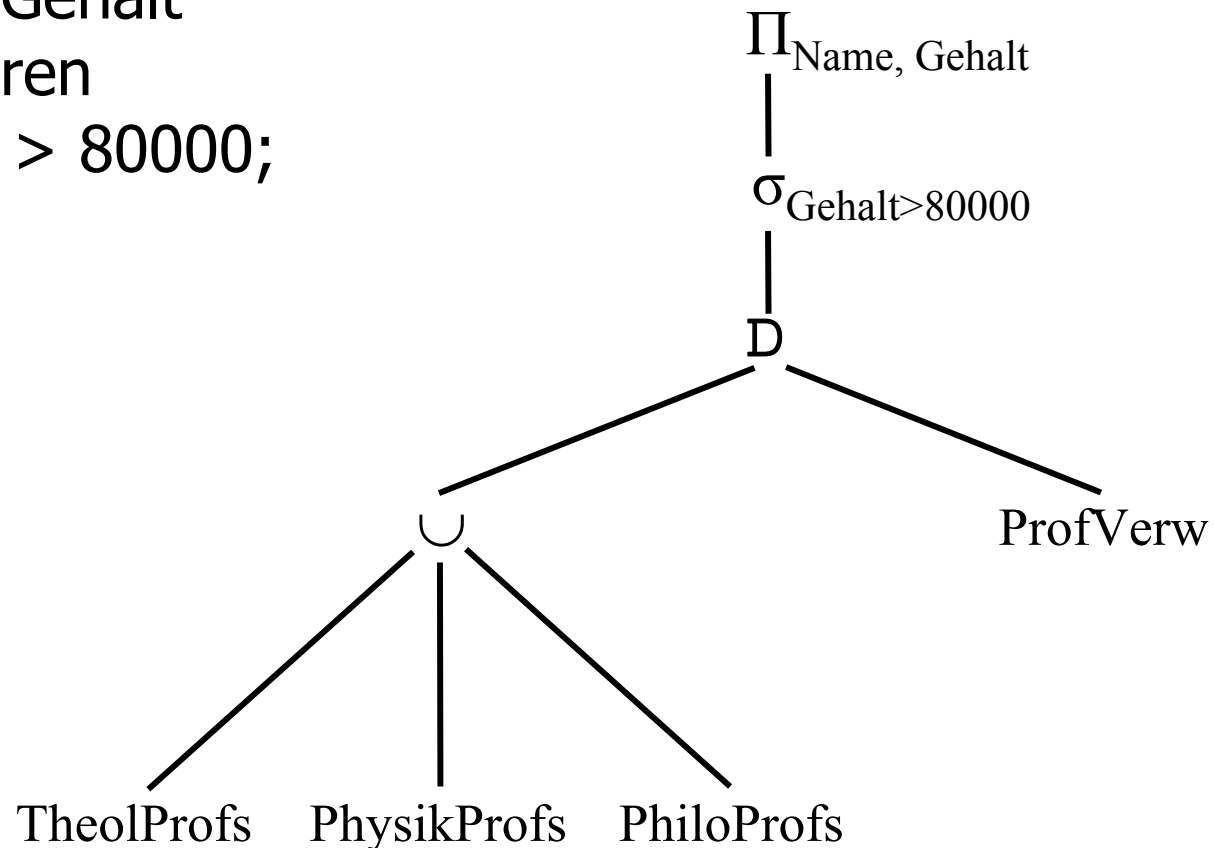


Query executions can be performed locally on nodes S_{Theol} , S_{Physik} and $S_{Philo} \Rightarrow$ Nodes may work in parallel and transmit local results independently of each other to the node that computes the union

Example:

select Name, Gehalt
from Professoren
where Gehalt > 80000;

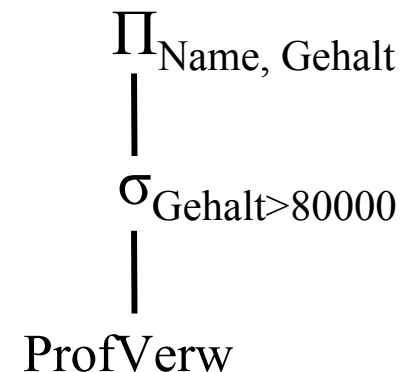
Canonical query plan:



For our example:

All required data are contained in *ProfVerw* \Rightarrow discard join and union.

Resulting query execution plan:



Example for query that is hard to optimize:
(Attribute *Rang* is lacking in *ProfVerw*)

```
select Name, Gehalt, Rang
from Professoren
where Gehalt > 80000;
```

| R | | |
|----------------|----------------|----------------|
| A | B | C |
| a ₁ | b ₁ | c ₁ |
| a ₂ | b ₂ | c ₂ |
| a ₃ | b ₃ | c ₁ |
| a ₄ | b ₄ | c ₂ |
| a ₅ | b ₅ | c ₃ |
| a ₆ | b ₆ | c ₂ |
| a ₇ | b ₇ | c ₆ |

D

| S | | |
|----------------|----------------|----------------|
| C | D | E |
| c ₁ | d ₁ | e ₁ |
| c ₃ | d ₂ | e ₂ |
| c ₄ | d ₃ | e ₃ |
| c ₅ | d ₄ | e ₄ |
| c ₇ | d ₅ | e ₅ |
| c ₈ | d ₆ | e ₆ |
| c ₅ | d ₇ | e ₇ |

=

| R D S | | | | |
|----------------|----------------|----------------|----------------|----------------|
| A | B | C | D | E |
| a ₁ | b ₁ | c ₁ | d ₁ | e ₁ |
| a ₃ | b ₃ | c ₁ | d ₁ | e ₁ |
| a ₅ | b ₅ | c ₃ | d ₂ | e ₂ |

Even more critical than in centralized data bases

Problem: Join arguments may be distributed to two different nodes of the distributed DBMS

2 possibilities: Join-evaluation with and without filter

Most general case:

Outer argument relation R is allocated to computing node St_R

Inner argument relation S is allocated to computing node St_S

Result of join evaluation is required on third node St_{Result}

Nested loops

Transfer of one argument relation

Transfer of both argument relations

Iteration over outer relation R using iteration variable r and requesting compatible tuples $s \in S$ with $r.C = s.C$
(using communication net at St_S)

This approach requires for each tuple from R one request and a compatible tuple set from S (which may be empty)

$\Rightarrow 2 * |R|$ messages must be send

1. Complete transfer of argument relation (e.g. R) to node of other argument relation
2. Exploitation of Index on $S.C$ if available

1. Transfer of both argument relations to node St_{Result}
2. Computation of result on node St_{Result} using
 - a) Merge join (if sorted)
 - or
 - b) Hash join (if unsorted)
 - Loss of existing index for join evaluation
 - No loss of sorting of argument relation(s)

Filtering using semi joins

Key idea:

transfer only tuples with compatible join partners

Exploiting algebraic equivalences:

$$R \bowtie D \bowtie S = R \bowtie D \bowtie (R \bowtie I \bowtie S)$$

$$R \bowtie I \bowtie S = \Pi_C(R) \bowtie I \bowtie S$$

(example, filtering relation S)

1. Transfer of different C -values from $R (= \Pi_C(R))$ to St_S
2. Evaluation of semi join $R \bowtie S = \Pi_C(R) \bowtie S$ on St_S and transfer to St_R
3. Evaluation of join on St_R , which only needs the transferred result tuples of semi joins

Transfer costs are reduced iff:

$$\| \Pi_C(R) \| + \| R \bowtie S \| < \| S \|$$

with $\| P \| = \text{Size (in Byte) of a relation}$

15 attribute values ...

St_{Result}

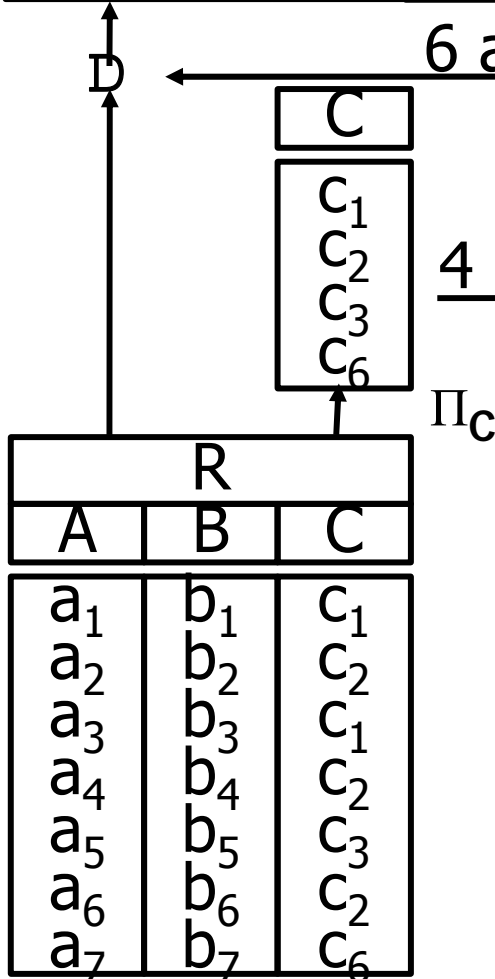
| R D ($\Pi_C(R)$ I S) | | | | |
|-----------------------|-------|-------|-------|-------|
| A | B | C | D | E |
| a_1 | b_1 | c_1 | d_1 | e_1 |
| a_3 | b_3 | c_1 | d_1 | e_1 |
| a_5 | b_5 | c_3 | d_2 | e_2 |

6 attribute values

| $\Pi_C(R)$ I S | | |
|----------------|-------|-------|
| C | D | E |
| c_1 | d_1 | e_1 |
| c_3 | d_2 | e_2 |

4 attribute values

| S | | |
|-------|-------|-------|
| C | D | E |
| c_1 | d_1 | e_1 |
| c_3 | d_2 | e_2 |
| c_4 | d_3 | e_1 |
| c_5 | d_4 | e_2 |
| c_7 | d_5 | e_3 |
| c_8 | d_6 | e_2 |
| c_5 | d_7 | e_6 |

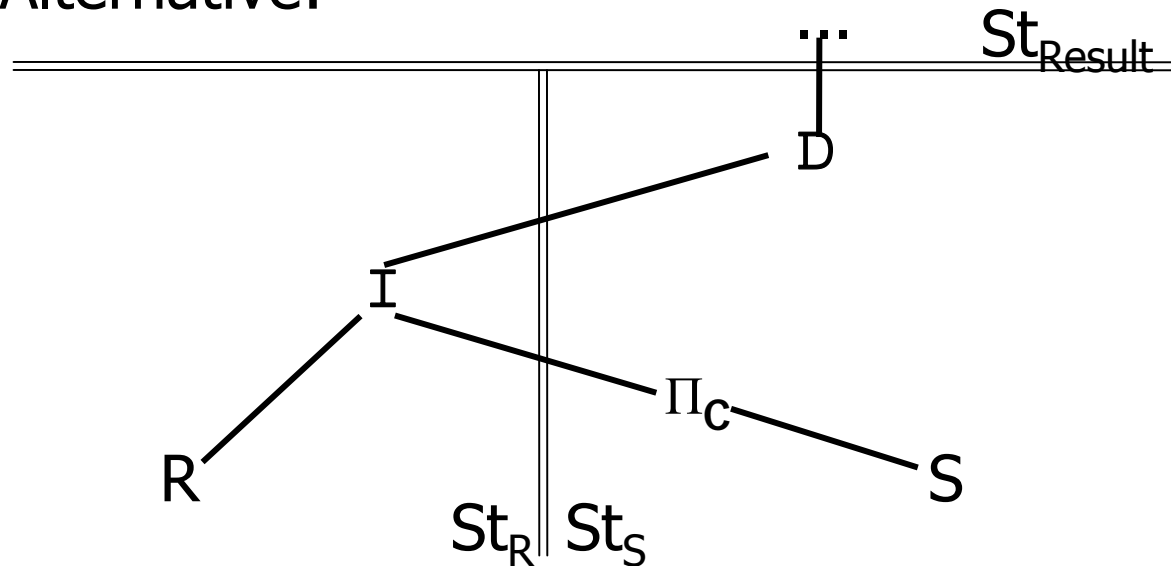


St_R

St_S

Evaluation of the join $R D S$
with semi join filtering on S

1. Alternative:



2. Alternative:

$(R \bowtie \Pi_C(S)) \bowtie (\Pi_C(R) \bowtie S)$

Problem:

For data item A there exist several copies A_1, A_2, \dots, A_n , which may reside on different nodes.

Read actions require only one copy, updates must change all existing copies.

⇒ Problems with high efficiency and availability

Trade-off between efficiency of read and update transactions

- + shifting some overhead from update to read transaction.
Approach: assign copies A_i of a replicated data item A individual weights

Read quorum $Q_r(A)$

Write quorum $Q_w(A)$

The following conditions must be met:

1. $Q_w(A) + Q_w(A) > W(A)$
2. $Q_r(A) + Q_w(A) > W(A)$

| <i>Station (S_i)</i> | <i>Kopie (A_i)</i> | <i>Gewicht (w_i)</i> |
|-----------------------------------|---------------------------------|-----------------------------------|
| S_1 | A_1 | 3 |
| S_2 | A_2 | 1 |
| S_3 | A_3 | 2 |
| S_4 | A_4 | 2 |

$$W(A) = \sum_{i=1}^4 w_i(A) = 8$$

$$Q_r(A) = 4$$

$$Q_w(A) = 5$$

a) Before writing

| Station | Kopie | Gewicht | Wert | Versions# |
|---------|-------|---------|------|-----------|
| S_1 | A_1 | 3 | 1000 | 1 |
| S_2 | A_2 | 1 | 1000 | 1 |
| S_3 | A_3 | 2 | 1000 | 1 |
| S_4 | A_4 | 2 | 1000 | 1 |

b) After writing using write quorum 5

| Station | Kopie | Gewicht | Wert | Versions# |
|---------|-------|---------|------|-----------|
| S_1 | A_1 | 3 | 1100 | 2 |
| S_2 | A_2 | 1 | 1000 | 1 |
| S_3 | A_3 | 2 | 1100 | 2 |
| S_4 | A_4 | 2 | 1000 | 1 |