Distributed Databases

Distributed Databases

Terminology

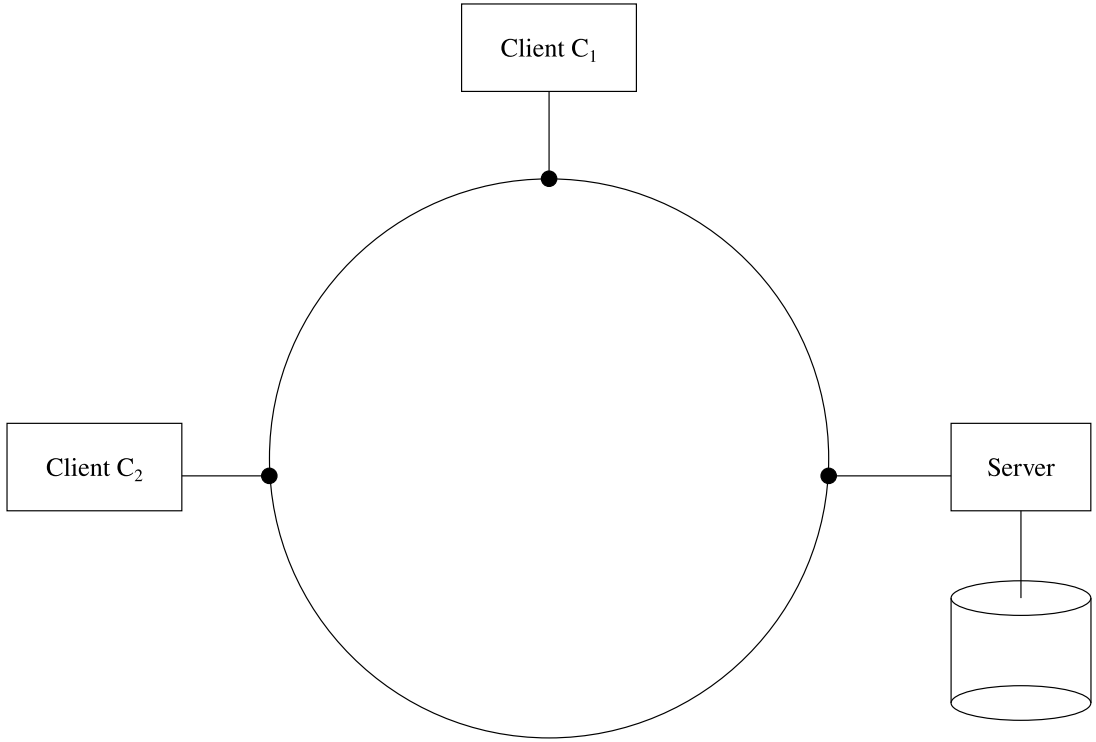
- A *distributed database* (DDBMS) is a collection of information units that is distributed across different computers, which are connected by communication networks
- Each site can
  - work autonomously with local data
  - cooperate globally with other computers in the network

Communication Network

- The communication network can be
  - LAN: local area network (Ethernet, Token Ring, FDDI network)
  - WAN: wide area network (Internet)
  - Telephone connections: ISDN, modem
- The communication network is transparent to the database application

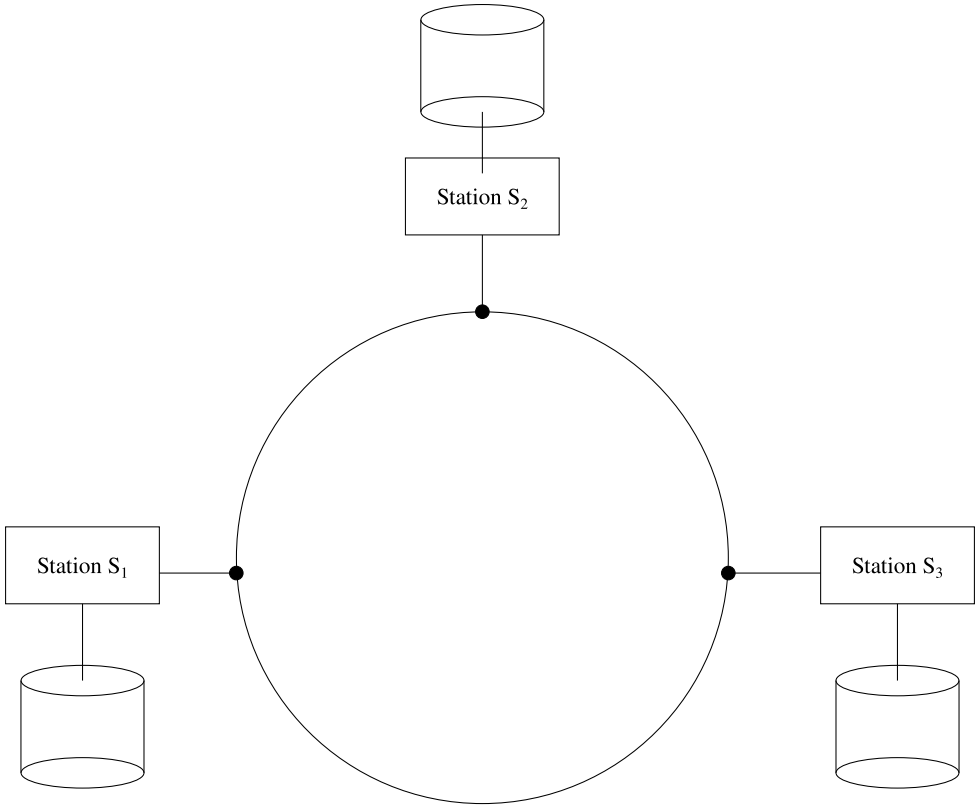
Delimitation

- A DDBMS is not a client-server architecture

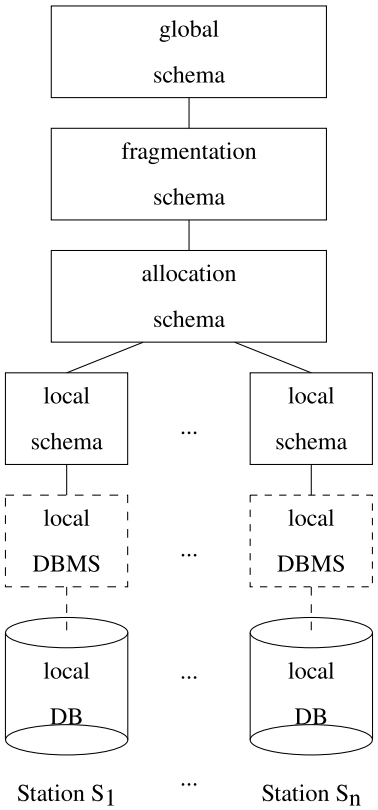


Delimitation(2)

- Each site holds its own data



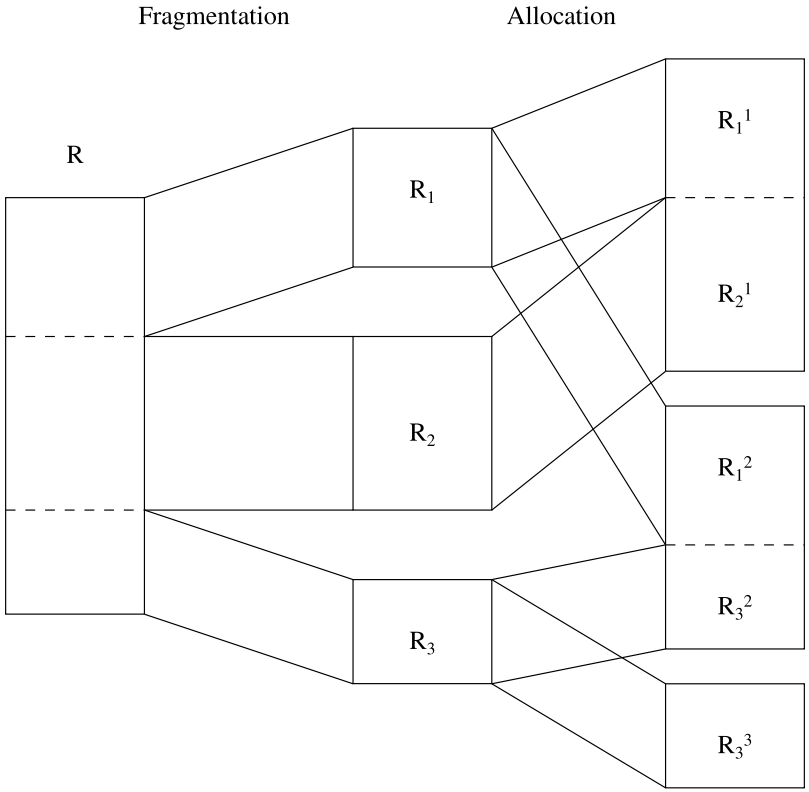
Structure of a DDBMS



Fragmentation/Allocation

- Fragmentation
  - Fragments contain data with the same access behavior
- Allocation
  - Fragments are assigned to sites
    - With replication
    - Without replication

Fragmentation/Allocation(2)



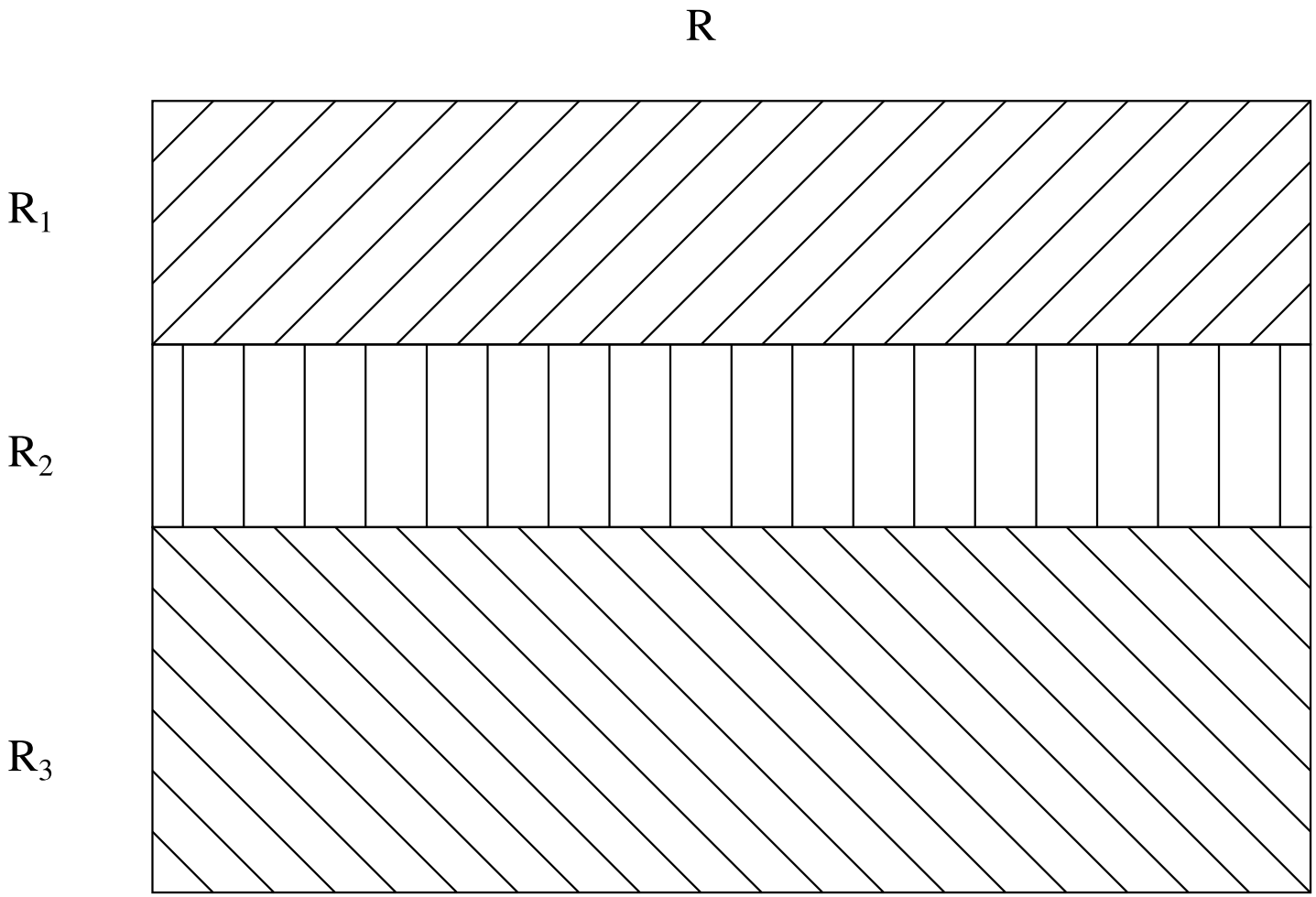
Fragmentation

- There are different methods of fragmentation:
  - Horizontal: decomposition of a relation into disjoint sets of tuples, decomposition by selections
  - Vertical: grouping attributes with the same access patterns, decomposition by projections
  - Combined: horizontal and vertical fragmentation on the same relation

Correctness

- There are three basic correctness requirements for fragmentations:
  - Reconstructability: the original relation can be restored from the fragments
  - Completeness: each data item is assigned to a fragment
  - Disjointness: fragments do not overlap, i.e., a data item is not assigned to multiple fragments

Horizontal Fragmentation



Horizontal Fragmentation(2)

- With n decomposition predicates there are 2^n possible fragments in total
- One predicate p_1:

$$R_1 := \sigma_{p_1}(R)$$
$$R_2 := \sigma_{\neg p_1}(R)$$

- Two predicates p_1, p_2:

$$R_1 := \sigma_{p_1 \wedge p_2}(R)$$
$$R_2 := \sigma_{p_1 \wedge \neg p_2}(R)$$
$$R_3 := \sigma_{\neg p_1 \wedge p_2}(R)$$
$$R_4 := \sigma_{\neg p_1 \wedge \neg p_2}(R)$$

Example

Professors						
PersNr	Name	Rank	Room	Faculty	Salary	TaxClass
2125	Sokrates	C4	226	Philosophy	85000	1
2126	Russel	C4	232	Philosophy	80000	3
2127	Kopernikus	C3	310	Physics	65000	5
2133	Popper	C3	52	Philosophy	68000	1
2134	Augustinus	C3	309	Theology	55000	5
2136	Curie	C4	36	Physics	95000	3
2137	Kant	C4	7	Philosophy	98000	1

Example(2)

$$p_1 \equiv \text{Faculty} = \text{'Theology'}$$
$$p_2 \equiv \text{Faculty} = \text{'Physics'}$$
$$p_3 \equiv \text{Faculty} = \text{'Philosophy'}$$

$$\text{TheolProfs}' := \sigma_{p_1 \wedge \neg p_2 \wedge \neg p_3}(\text{Professors}) = \sigma_{p_1}(\text{Professors})$$
$$\text{PhysicsProfs}' := \sigma_{\neg p_1 \wedge p_2 \wedge \neg p_3}(\text{Professors}) = \sigma_{p_2}(\text{Professors})$$
$$\text{PhiloProfs}' := \sigma_{\neg p_1 \wedge \neg p_2 \wedge p_3}(\text{Professors}) = \sigma_{p_3}(\text{Professors})$$
$$\text{OtherProfs}' := \sigma_{\neg p_1 \wedge \neg p_2 \wedge \neg p_3}(\text{Professors})$$

Derived Horiz. Fragmentation

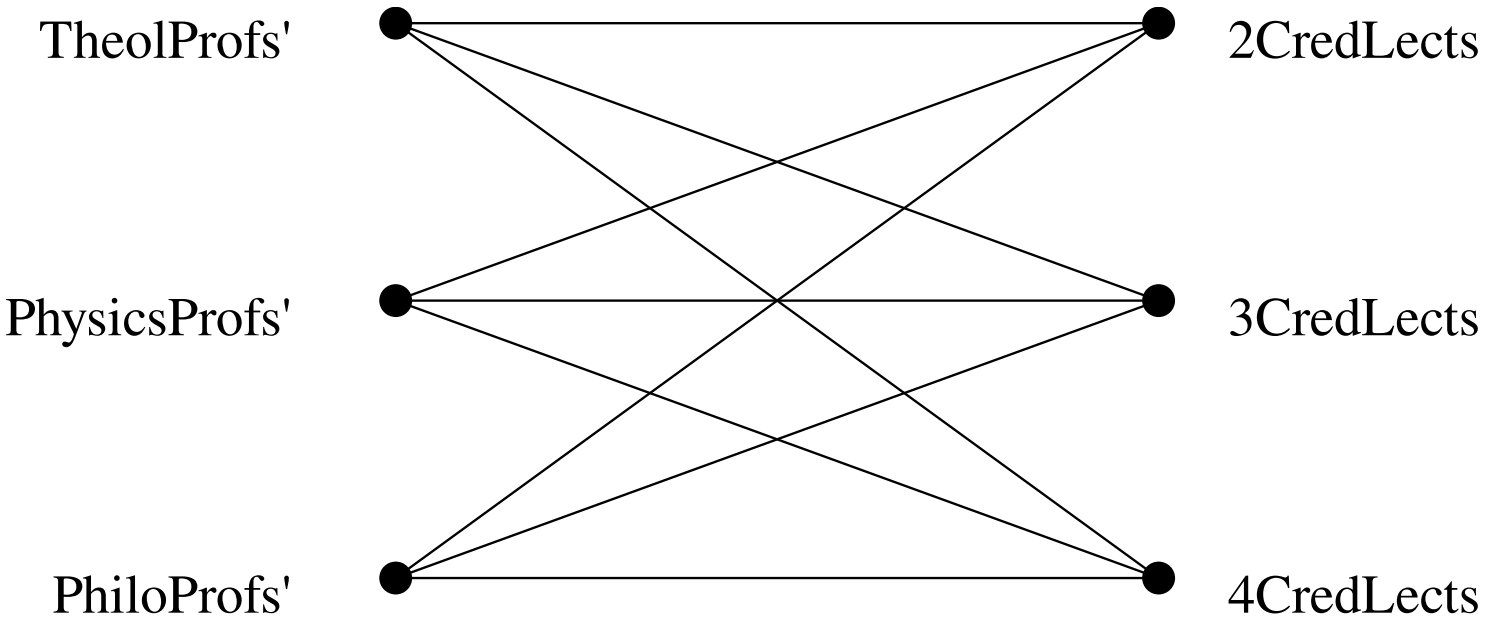
- Sometimes it makes sense to decompose a relation depending on another horizontal fragmentation
- Example: completely independent decomposition of Lectures by Credits:

$$\begin{aligned} \text{2CredLects} &:= \sigma_{\text{Credits}=2}(\text{Lectures}) \\ \text{3CredLects} &:= \sigma_{\text{Credits}=3}(\text{Lectures}) \\ \text{4CredLects} &:= \sigma_{\text{Credits}=4}(\text{Lectures}) \end{aligned}$$

Derived Fragmentation(2)

- To answer the following query, 9 joins of fragments have to be performed:

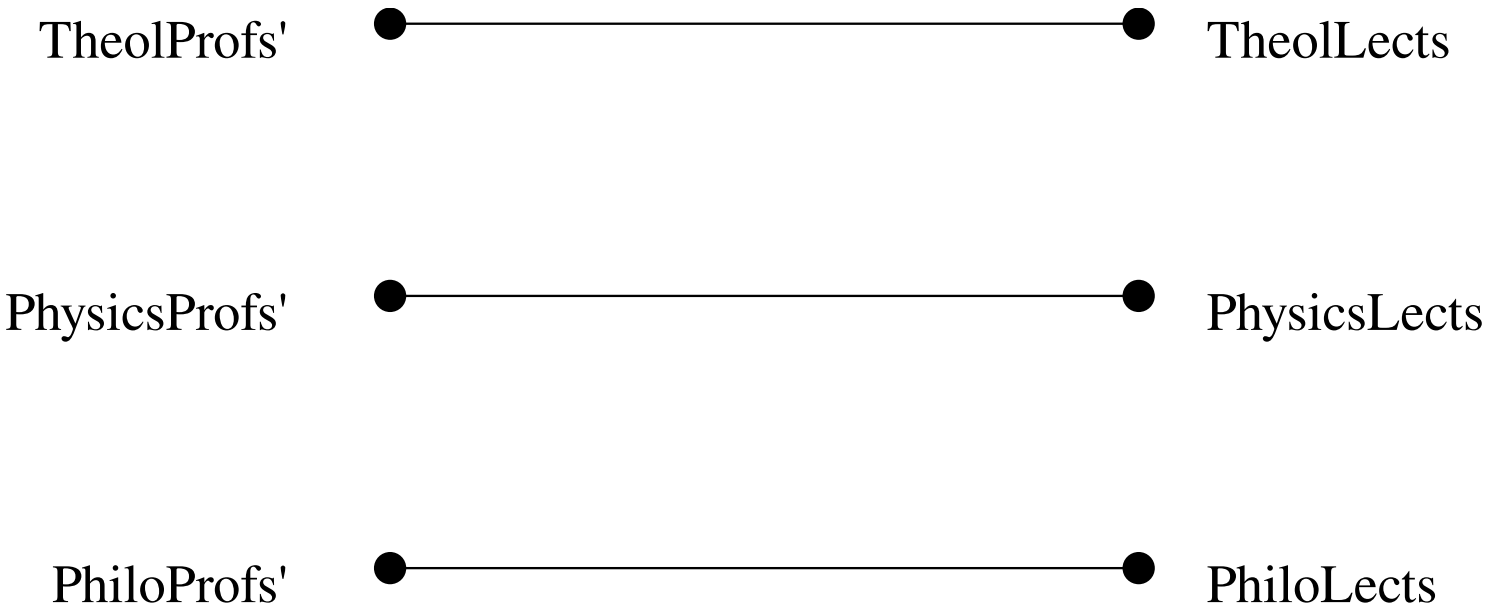
```
select Title, Name
from   Lectures, Professors
where  GivenBy = PersNr
```



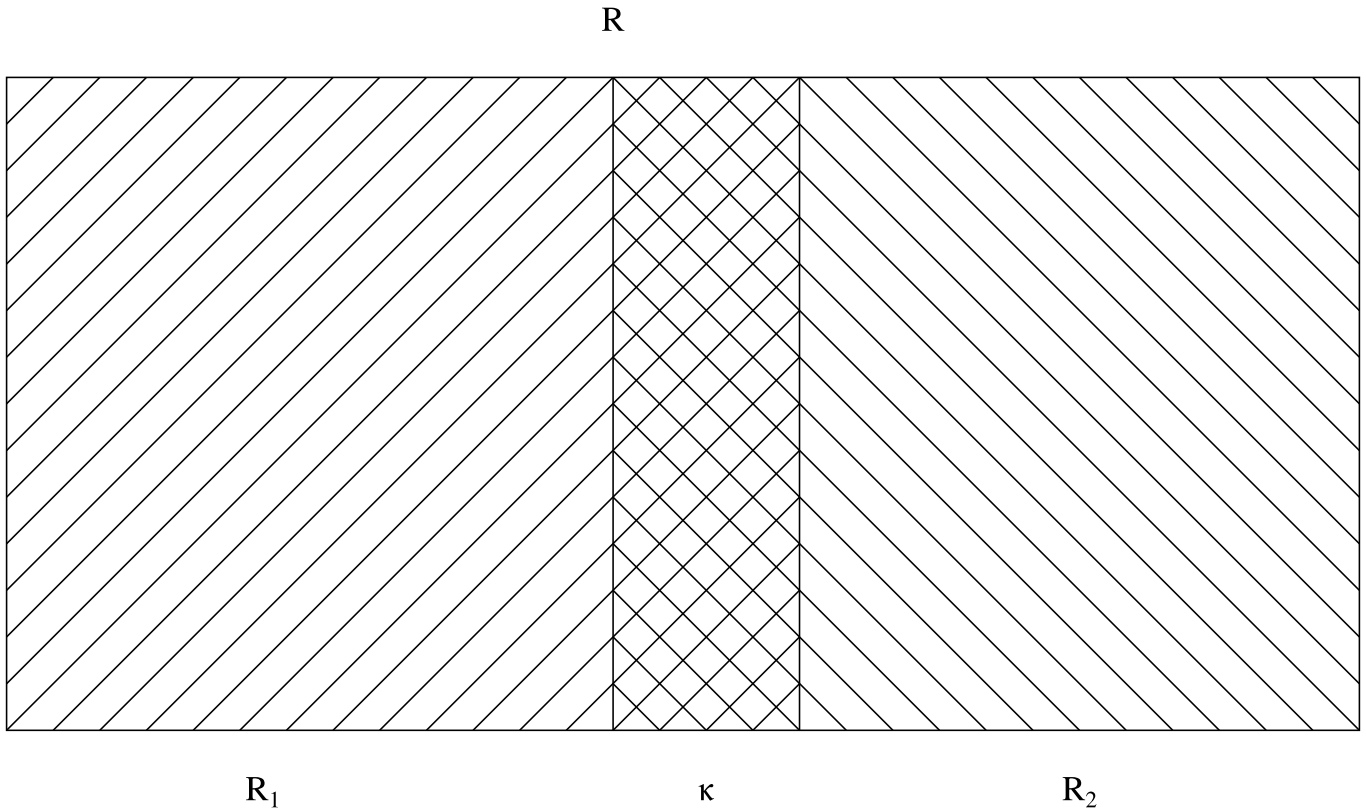
Derived Fragmentation(3)

- The following (derived) fragmentation is more sensible:

$$\begin{aligned} \text{TheolLects} &:= \text{Lectures} \ltimes_{\text{GivenBy=PersNr}} \text{TheolProfs}' \\ \text{PhysicsLects} &:= \text{Lectures} \ltimes_{\text{GivenBy=PersNr}} \text{PhysicsProfs}' \\ \text{PhiloLects} &:= \text{Lectures} \ltimes_{\text{GivenBy=PersNr}} \text{PhiloProfs}' \end{aligned}$$



Vertical Fragmentation



Vertical Fragmentation(2)

- With a decomposition without overlap, vertical fragmentation has a problem: violation of reconstructability
- One allows a “slight” violation of disjointness:
  - Each fragment contains the primary key
  - Each tuple of the original relation is assigned an artificial surrogate key, which is included in the fragment

Example

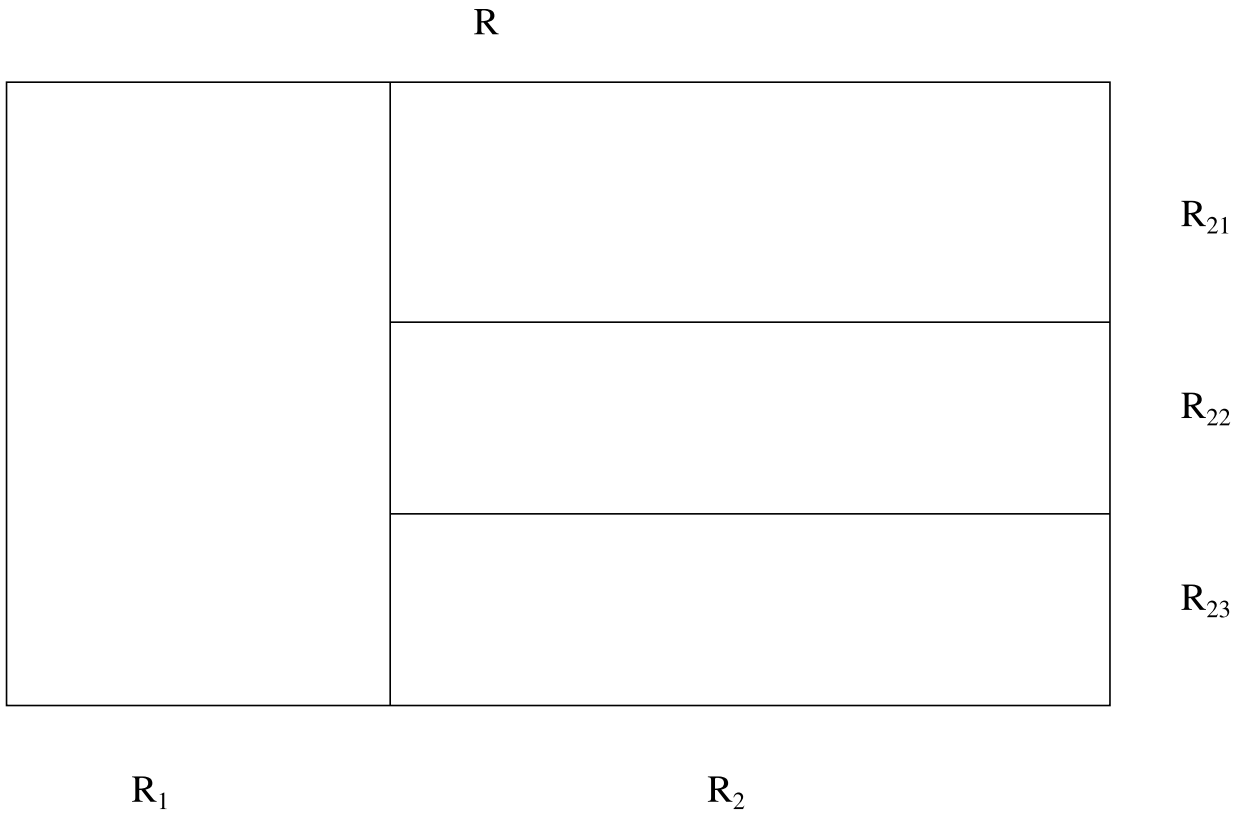
- One fragment for the university administration: ProfAdmin
- One fragment for teaching and research: Profs

$$\begin{aligned} \text{ProfAdmin} &:= \Pi_{\text{PersNr, Name, Salary, TaxClass}}(\text{Professors}) \\ \text{Profs} &:= \Pi_{\text{PersNr, Name, Rank, Room, Faculty}}(\text{Professors}) \end{aligned}$$

$$\text{Professors} = \text{ProfAdmin} \bowtie_{\text{ProfAdmin.PersNr=Profs.PersNr}} (\text{Profs})$$

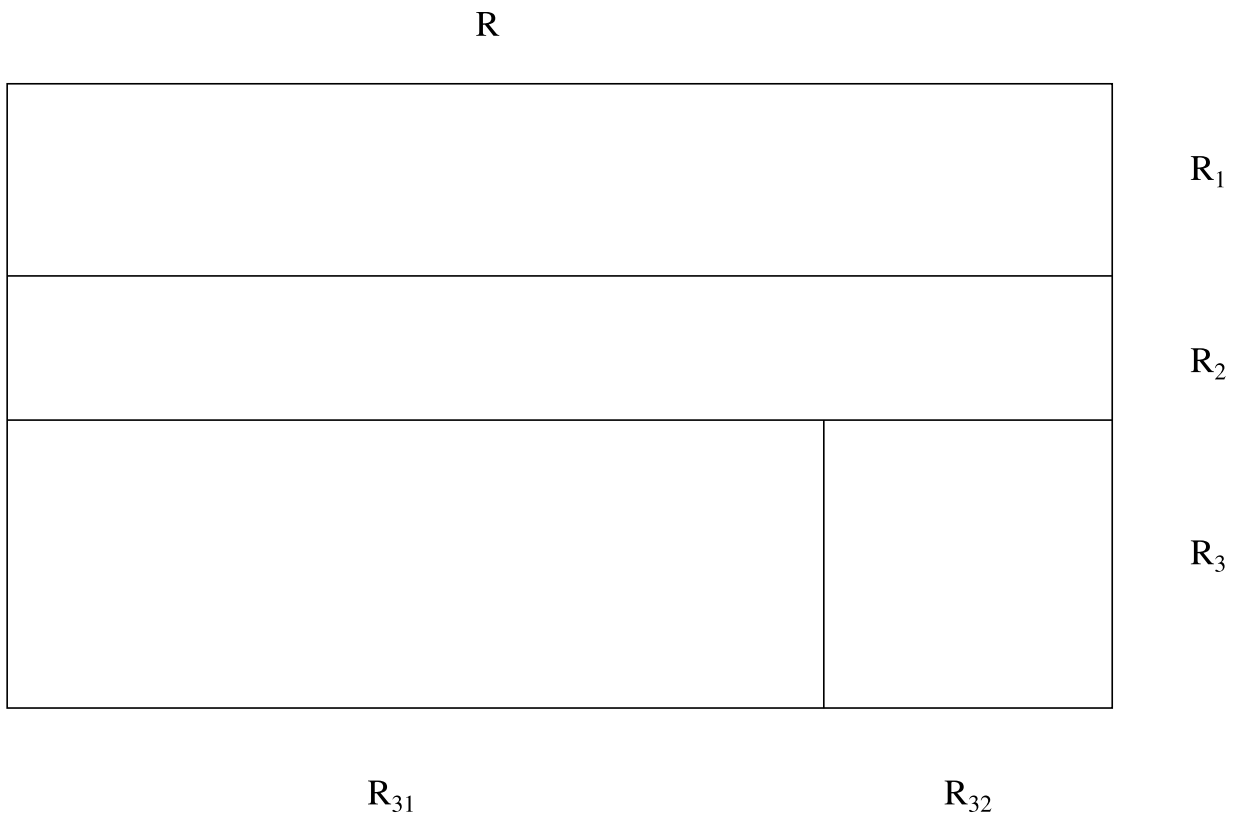
Combined Fragmentation

- First vertical, then horizontal:

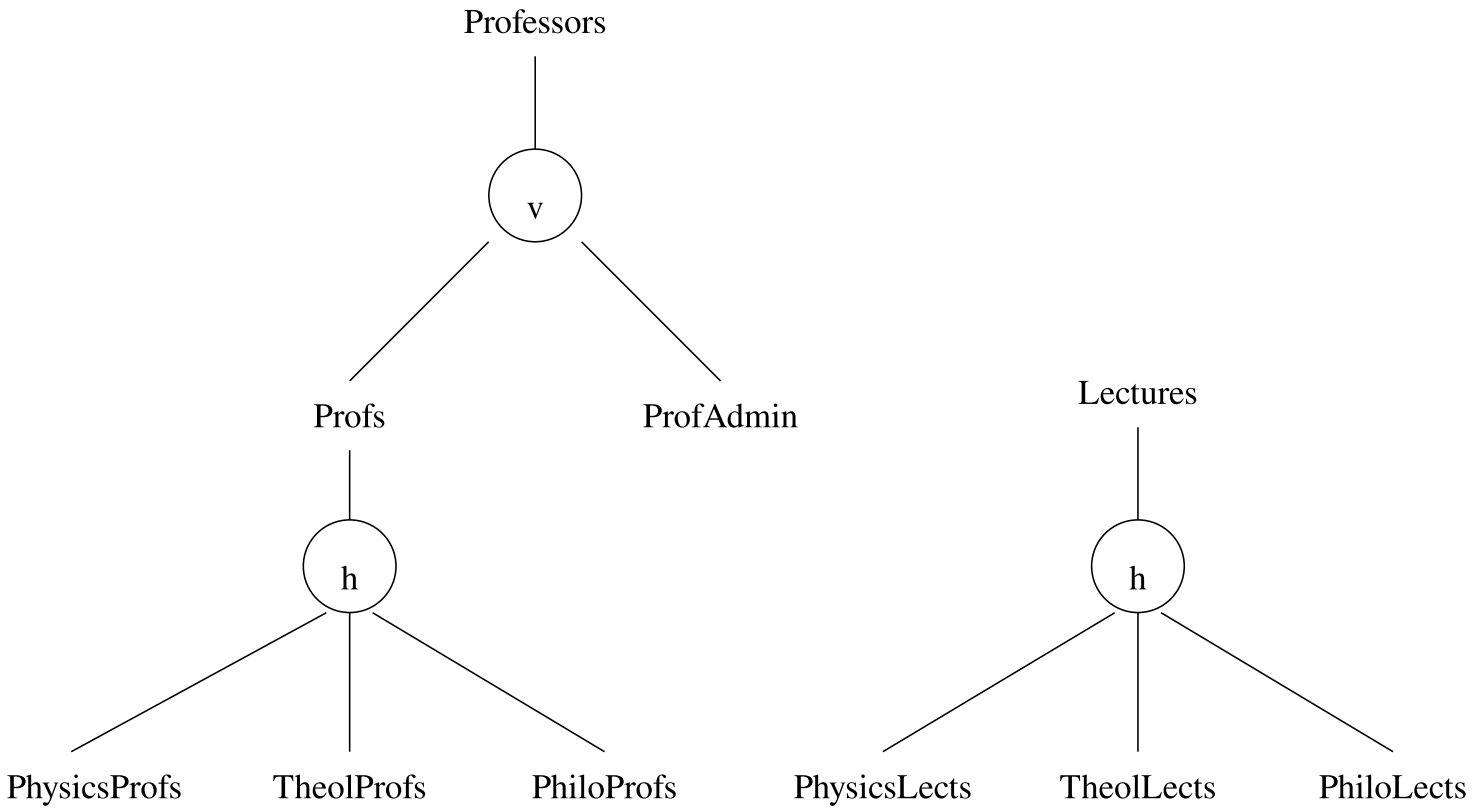


Combined Fragmentation(2)

- First horizontal, then vertical:



Example Application



Example Application(2)

- During allocation, the fragments are now assigned to sites (here without replication)

Site	Remark	Assigned Fragments
S_{Admin}	Administration computer	{*ProfAdmin*}
$S_{Physics}$	Physics dean's office	$\{PhysicsLects, PhysicsProfs\}$
S_{Philo}	Philosophy dean's office	$\{PhiloLects, PhiloProfs\}$
S_{Theol}	Theology dean's office	$\{TheolLects, TheolProfs\}$

Transparency

- *Transparency* is the degree of independence that a DDBMS provides to the user
- Different levels are distinguished:
  - Fragmentation transparency
  - Allocation transparency
  - Local schema transparency

Fragmentation Transparency

- Highest level of transparency (ideal state)
- User works on the global schema and the DDBMS translates queries into operations on fragments
- Example:

```
select Title, Name
from   Lectures, Professors
where  GivenBy = PersNr
```

Allocation Transparency

- Next lower level
- User has to know the fragments, but not their location
- Example:

```
select Salary
from   ProfAdmin
where  Name = 'Sokrates'
```

Local Schema Transparency

- At this level the user has to know both the fragment and its location
- The question arises to what extent there is any transparency left at all (all computers use the same data model)
- Example:

```
select Name
from   TheolProfs at S_Theol
where  Rank = 'C3'
```

Aspects of Distributed DBS

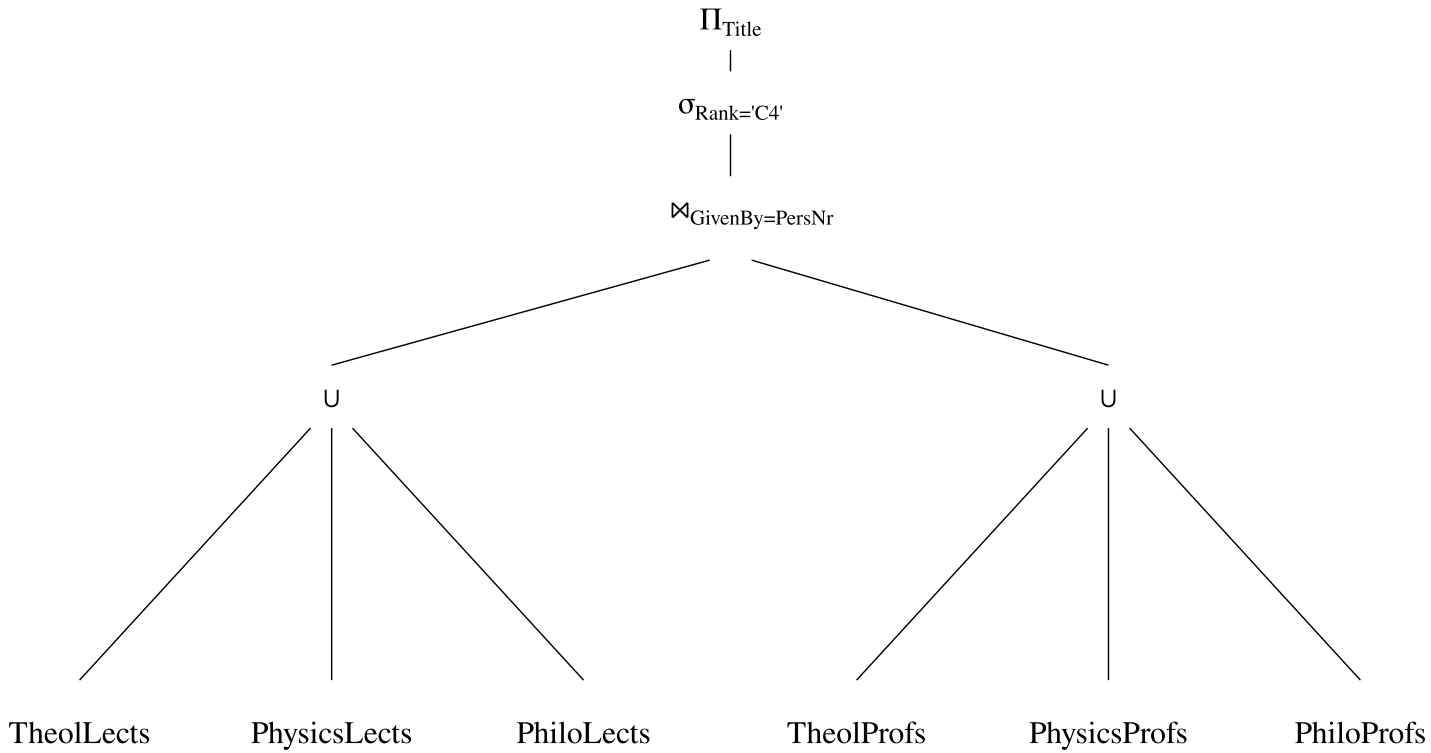
- Due to the distribution of the data, the following areas have to be adapted:
  - Query processing/optimization
  - Transaction control
  - Concurrency control

- We distinguish between
  - Horizontal fragmentation
  - Vertical fragmentation

```
select Title
from   Lectures, Profs
where  GivenBy = PersNr and
       Rank = 'C4'
```

- Reconstruct all global relations occurring in the query from the fragments
- Combine the reconstruction expression with the expression from the translation of the SQL query

Canonical Form



- The canonical form is correct, but inefficient
- A central property of relational algebra is:

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

- But this does not gain much (combining $R_1, ..., R_n$ and $S_1, ..., S_m$ requires $n \cdot m$ join operations)

- But if each S_i is a derived horizontal fragmentation, i.e.,

$$S_i = S \ltimes_p R_i \quad \text{with} \quad S = S_1 \cup \cdots \cup S_n$$

- then

$$R_i \bowtie_p S_j = \varnothing \quad \text{for} \quad i \neq j$$

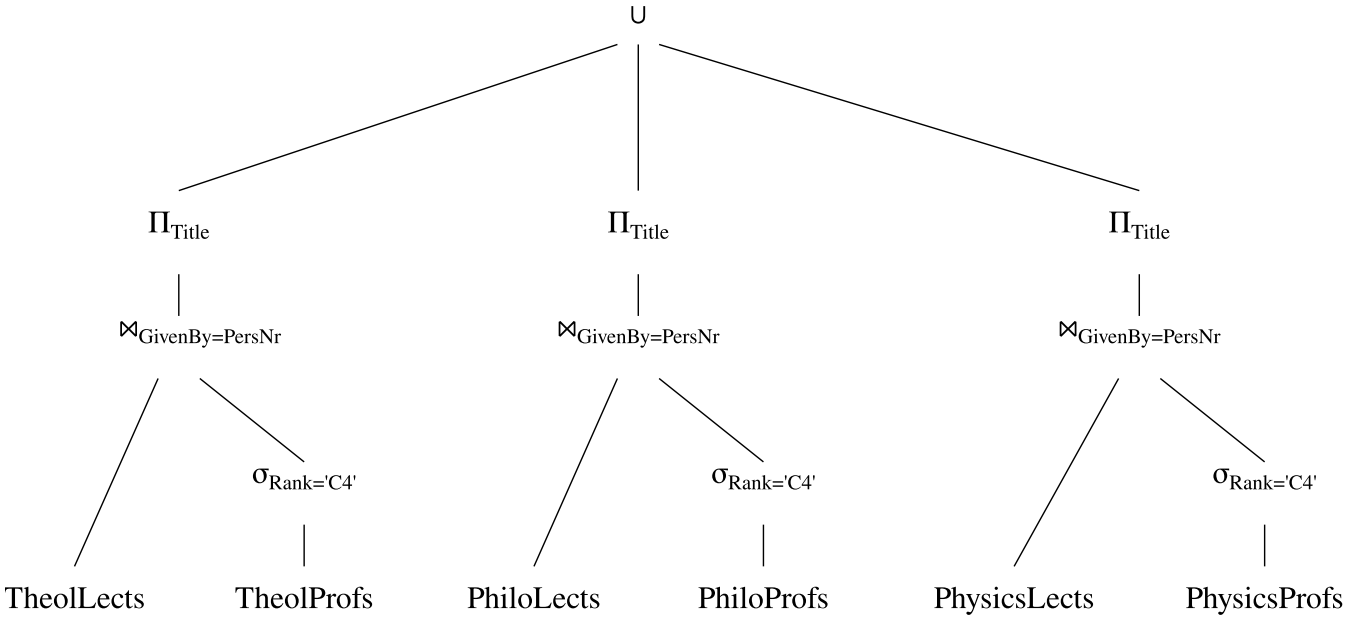
- and thus

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

- Thus the joins of our example can be executed locally
- In addition, there are rules for pushing selections and projections down:

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

Optimized Plan

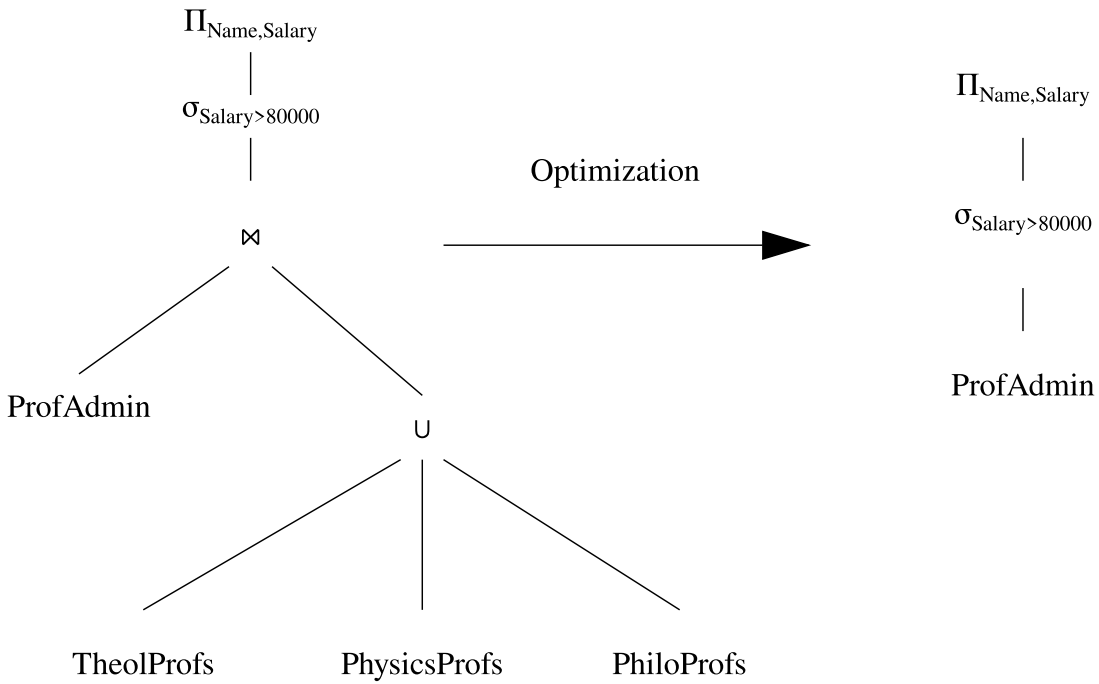


Vertical Fragmentation

```
select Name, Salary
from   Professors
where  Salary > 80000
```

- Naive approach: reconstruct the global relation, then evaluate the query
- More sensible: fetch only relevant fragments

Optimization



- Problem: relations to be joined may reside at different sites
- Consider the most general case:
  - Outer relation R is at site St_R
  - Inner relation S is at site St_S
  - Result is needed at site St_{Result}

Evaluation without Filtering

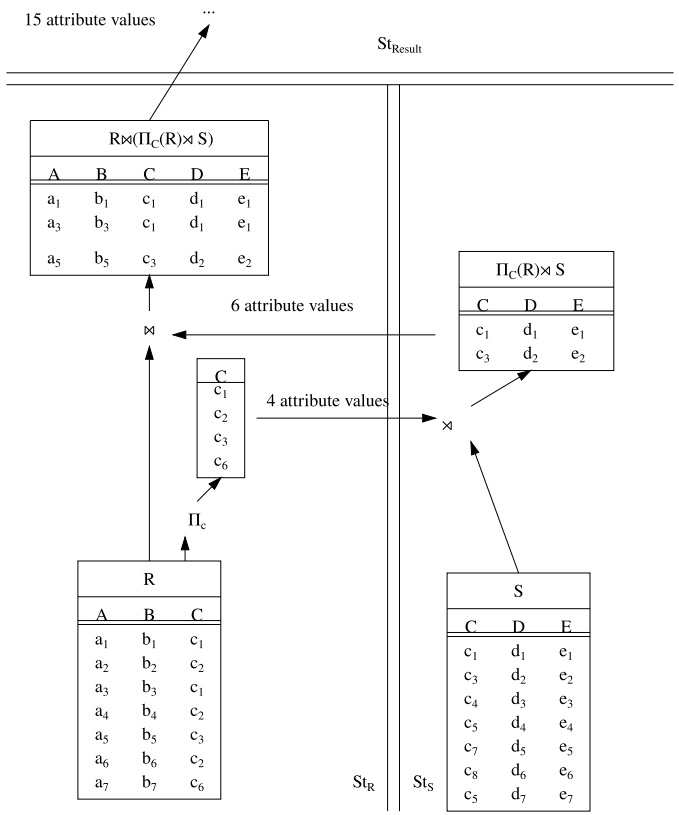
- Nested loop: iterate through the tuples of R, send each tuple to St_S, look up matching tuples, join and send the result to St_{Result}
- Transfer of one relation: send the complete relation to the other node and perform the join there, send the result to St_{Result}
- Transfer of both relations: send both relations to St_{Result} and perform the join there

Evaluation with Filtering

- Without filtering, large amounts of data have to be sent over the network, although the result may be very small
- Idea: send only tuples that actually find a join partner
- The following properties are used
  (C is the join attribute):

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

Example


15 attribute values
...
St_{Result}
$R \bowtie (\Pi_C(R) \bowtie S)$
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
$\Pi_C(R) \bowtie S$
C D E
c_1 d_1 e_1
c_3 d_2 e_2
$\bowtie$
C
c_1
c_2
c_3
c_6
4 attribute values
$\bowtie$
Π_c
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
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
St_R
St_S

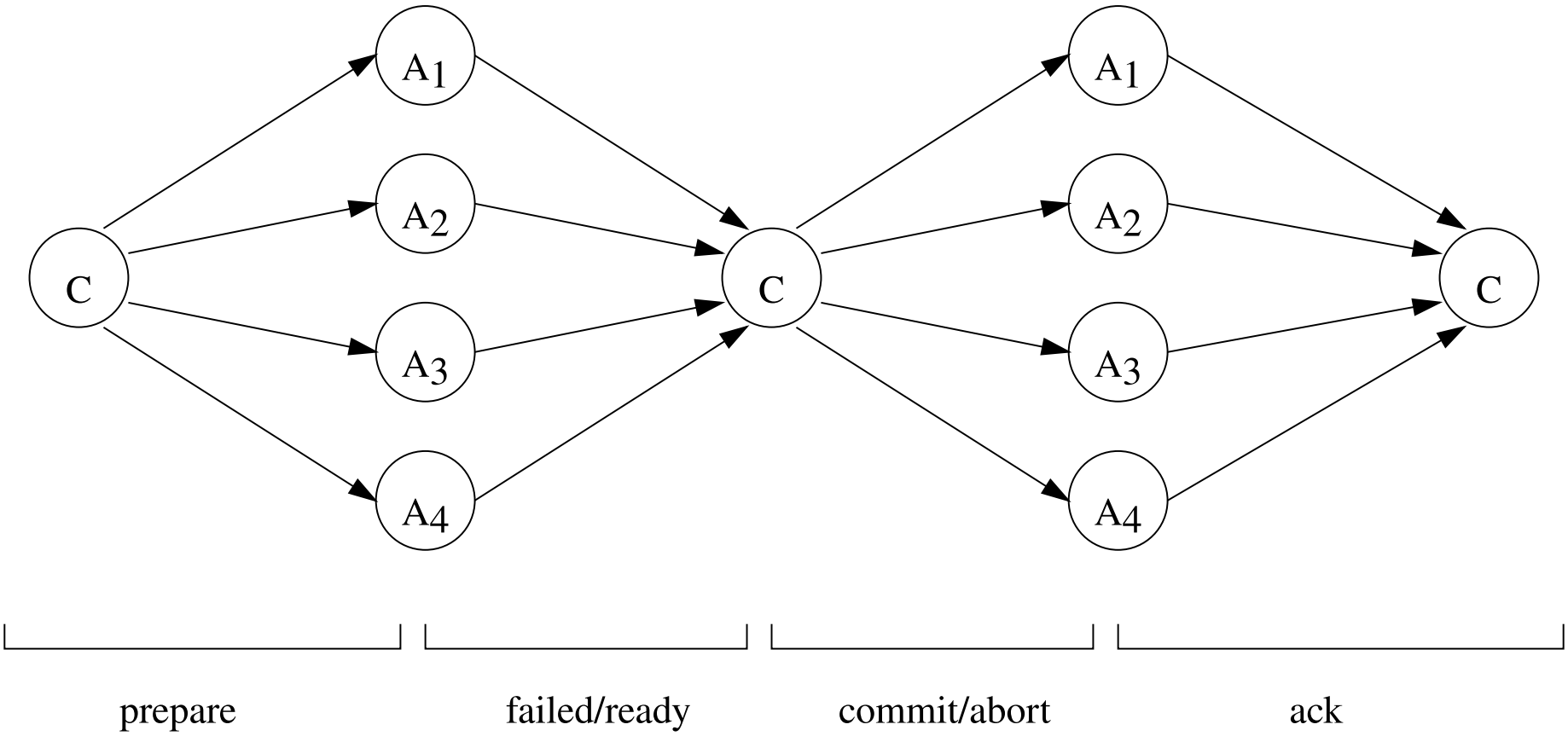
Transaction Control

- Transactions can span multiple computer nodes
- All sites write local log records about executed operations
- Needed during restart to reconstruct the data of a crashed site

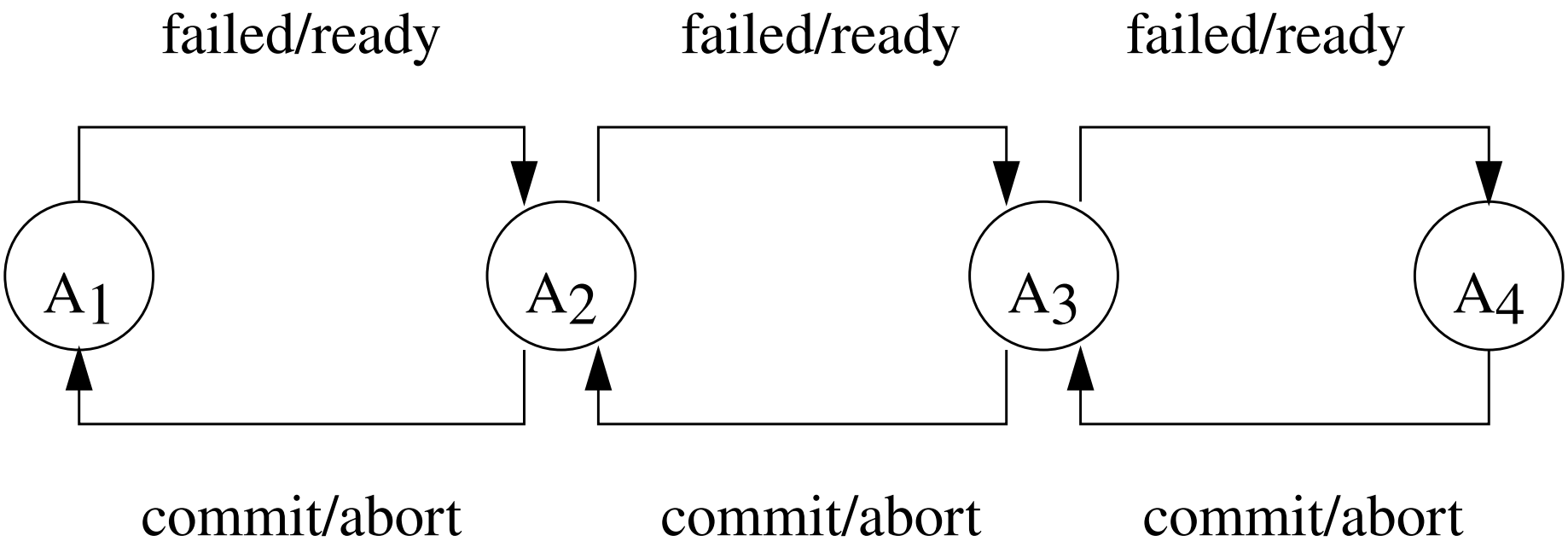
Transaction Control(2)

- Abort: when a global transaction is aborted, all local parts have to be rolled back
- Commit: fundamental difficulty when terminating:
  - Atomic termination of the transaction must be guaranteed

Two-Phase Commit (2PC)



C=coordinator, A=agent


failed/ready
failed/ready
failed/ready
A1
A2
A3
A4
commit/abort
commit/abort
commit/abort

Error Handling

- During the distributed commit, the following failure cases can occur:
  - Crash of the coordinator
  - Crash of an agent
  - Lost message
- In the worst case, nodes block

- Local serializability does not guarantee global serializability
- Example:

S_1		
Step	T_1	T_2
1.	$r(A)$	
2.		$w(A)$

S_2		
Step	T_1	T_2
3.		$w(B)$
4.	$r(B)$	

- Regular 2PL is not sufficient in the distributed case
- Only strict 2PL guarantees serializability
- Lock management:
  - central
  - local

Central Management

- All transactions request locks at a dedicated site
- This site can easily become a “bottleneck”
- In addition, this approach violates the local autonomy of the sites
- Therefore this approach is not used

Local Management

- Global transactions (transactions that run on more than one site) have to obtain the lock from the local lock manager before modifying a data item
- Local transactions only have to communicate with their own manager
- However, detecting deadlocks is more difficult than with central management

Deadlocks

- Local deadlock detection is not sufficient:

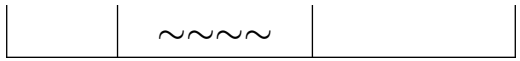
S_1

Step	T_1	T_2
0.	**BOT**	
1.	**lockS**(A)	
2.	$r(A)$	
6.		**lockX**(A)
		$\sim\sim\sim\sim$

S_2

Step	T_1	T_2
3.		**BOT**
4.		**lockX**(B)
5.		$w(B)$
7.	**lockS**(B)	

Deadlocks



Deadlock Detection

- Timeouts: after a time interval has elapsed, the transaction is rolled back (choice of the interval is critical)
- Centralized deadlock detection: one node builds a central wait-for graph (high overhead, phantom deadlocks)
- Decentralized deadlock detection: local wait-for graphs + special node *External*

Decentralized Detection

- Each transaction is assigned a home node (usually where the transaction was started)
- A transaction can start external subtransactions at other sites
- In the deadlock example, S_1 is the home of T_1 and S_2 is the home of T_2

Decentralized Detection(2)

- For an external subtransaction T_i the following edge is introduced:

$$External \rightarrow T_i$$

- At another site, someone is waiting for T_i to complete (namely the transaction that initiated the external subtransaction)
- For a transaction T_j that initiates a subtransaction, the edge

$$T_j \rightarrow External$$

- T_j waits for the completion of the subtransaction started at another site

Decentralized Detection(3)

- For our example this means

$$S_1 : \boxed{External \rightarrow T_2 \rightarrow T_1 \rightarrow External}$$

$$S_2 : \boxed{External \rightarrow T_1 \rightarrow T_2 \rightarrow External}$$

- A cycle containing *External* is not necessarily a deadlock
- To determine a deadlock, sites have to exchange information

Decentralized Detection(4)

- A site with local wait-for graph

$$External \rightarrow T_1' \rightarrow T_2' \rightarrow ... \rightarrow T_n' \rightarrow External$$

  sends its local graph to the site where T_n' started a subtransaction
- For our example:

$$S_2: \boxed{External \leftrightarrow T_1 \leftrightarrow T_2 \leftrightarrow External}$$

$$T_1 \rightarrow T_2 \rightarrow T_1$$

$$T_2 \rightarrow T_1 \rightarrow T_2$$

Decentralized Detection(5)

- To avoid redundant messages (in the example above, S_1 sends information to S_2 and vice versa), the graph is not always sent
- For a (local) wait-for graph

$$External \rightarrow T'_1 \rightarrow T'_2 \rightarrow ... \rightarrow T'_n \rightarrow External$$

the information is only sent if the transaction identifier of T'_n is greater than the transaction identifier of T'_1

Deadlock Avoidance

- There are approaches that use timestamps to avoid deadlocks
  - Timestamp-based concurrency control
  - Deadlock avoidance for lock-based protocols: wound/wait, wait/die
- Requires that globally unique timestamps can be generated

Timestamp Generation

- Most common method:



- The site ID must be in the least significant bits
- Otherwise transactions of certain sites would always be preferred
- In addition, clocks should not deviate too far from each other

Synchronization with Replication

- What if there are multiple copies of a data item?
- If data is only ever read, this is unproblematic
- It suffices to read any copy
- Updates are where it becomes problematic

Write All/Read Any

- An update operation has to modify all copies
- Favors read operations, which only have to read one copy
- If a copy fails, update operations can no longer be executed or are delayed

- Idea: copies get weights (depending on robustness and performance of the site)
- It suffices to collect copies with a certain total weight

Site (S_i)	Copy (A_i)	Weight (w_i)
S_1	A_1	3
S_2	A_2	1
S_3	A_3	2
S_4	A_4	2

Quorum-Consensus(2)

$W(A) = \sum \texttt{limits}_{i=1}^{4} w_i(A) = 8.$

Read quorum $Q_r(A)$

Write quorum $Q_w(A)$

- $Q_w(A) + Q_w(A) > W(A)$ and
- $Q_r(A) + Q_w(A) > W(A)$.

Example:

- $Q_r(A) = 4$
- $Q_w(A) = 5$

Update Operation

- Before writing:

Site	Copy	Weight	Value	Version#
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

- After writing:

Site	Copy	Weight	Value	Version#
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

Summary

- In distributed database systems, the data is distributed across spatially (widely) separated computers
- Due to the distribution of the data, several of the mechanisms commonly used in DBMS become considerably more complicated