# Distributed DBMSs - Concepts and Design Transparencies

Dr. Xu Yang

## Objectives

- Concepts.
- Advantages and disadvantages of distributed databases.
- Functions and architecture for a DDBMS.
- Distributed database design.
- Levels of transparency.

# DBMS

## Centralized DBMS

- It allows users to access only a single logical database located at one site under its control.

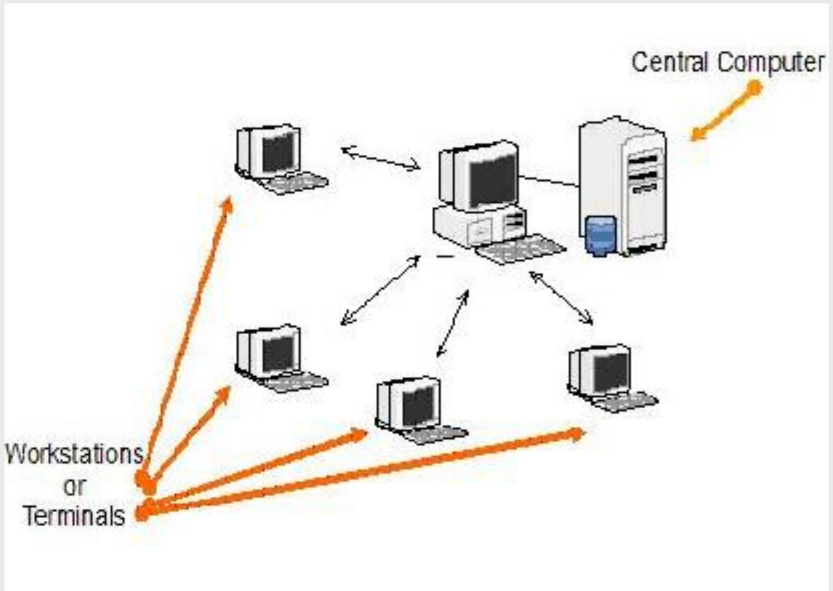


Fig. Centralised DBMS

## Distributed DBMS

- It allows users to access not only the data at their own site but also data stored at remote sites.

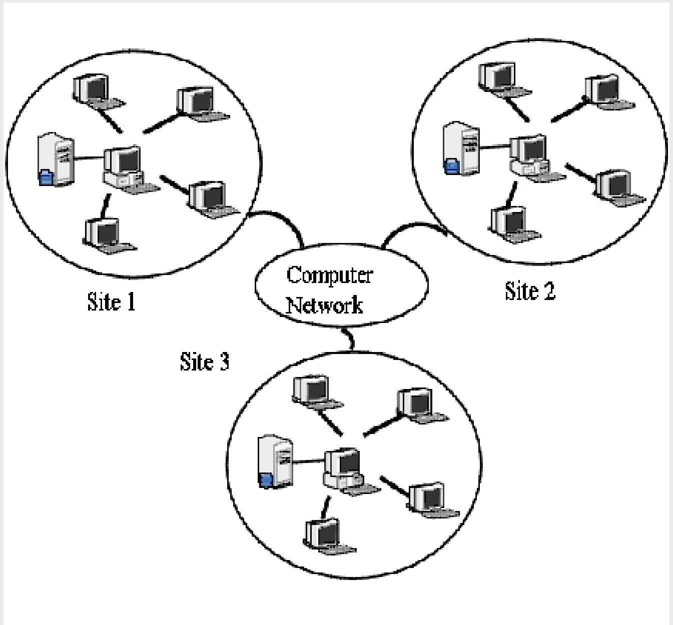
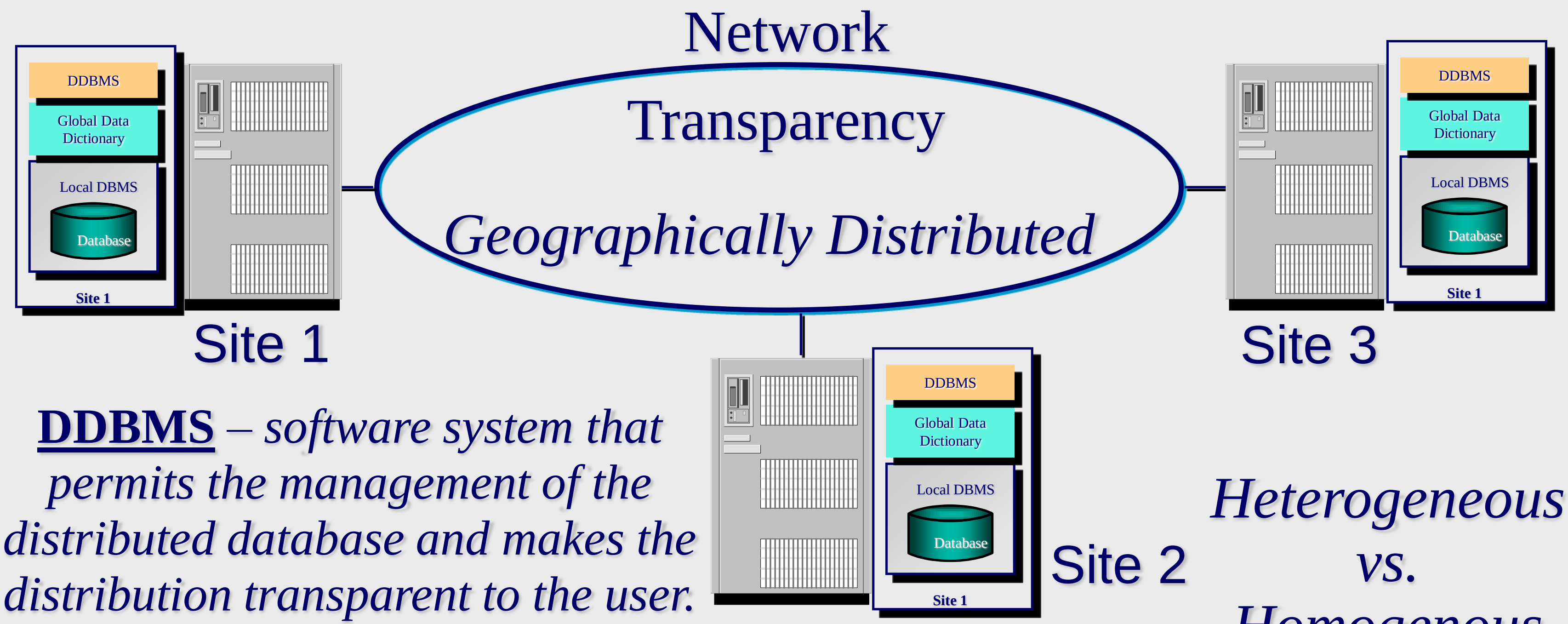


Fig. Distributed database

# Distributed Database

<u>**Distributed database:**</u>

A logically interrelated collection of shared data, physically distributed over a computer network



<u>**DDBMS**</u> – *software system that permits the management of the distributed database and makes the distribution transparent to the user.*

*Heterogeneous vs. Homogenous*

## Transparent & Location Independence

- In a Transparent manner
  - each user within the system may access all of the data within all of the databases as if they were a single database
- There should be ‘location independence’
  - as the user is unaware of where the data is located it is possible to move the data from one physical location to another without affecting the user.

## Global and Local Applications

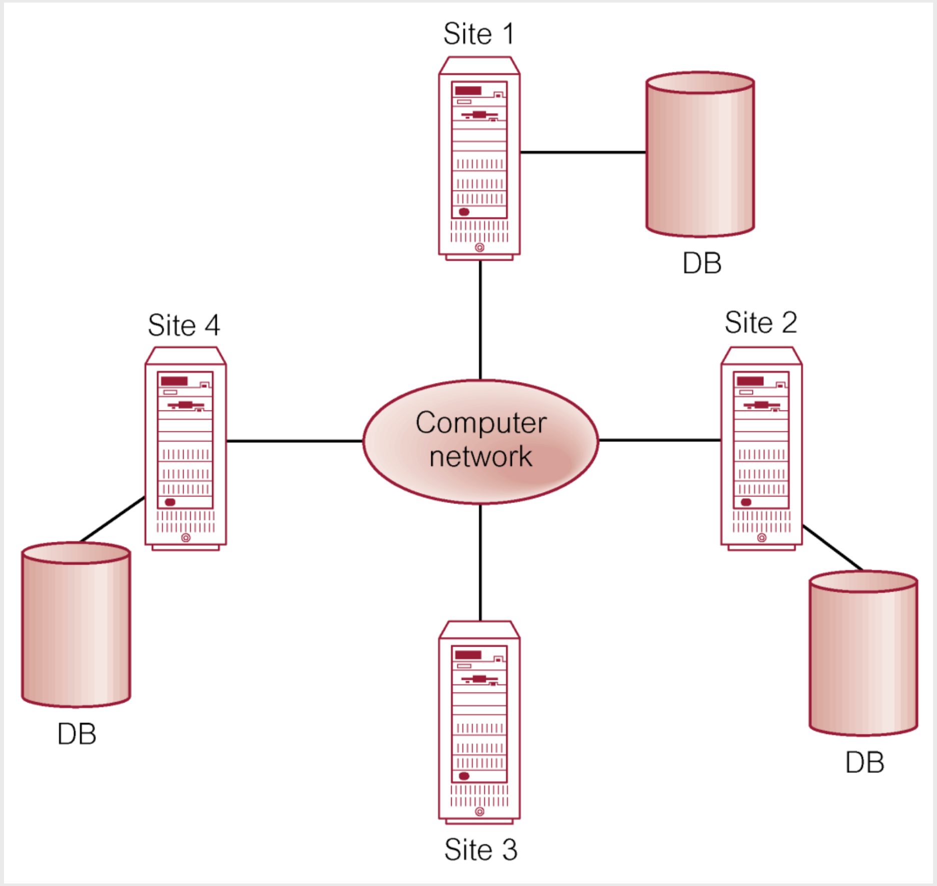
- Users access the distributed database via applications
  - Local applications: those applications that do not require data from other sites
  - Global applications: those applications that require data from other sites.
- A DDBMS have at least one global application.

## A DDBMS has the following characteristics

- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communications network.
- Data at each site is under control of a DBMS.
- DBMSs handle local applications autonomously.
- Each DBMS participates in at least one global application.

## Distributed DBMS Example

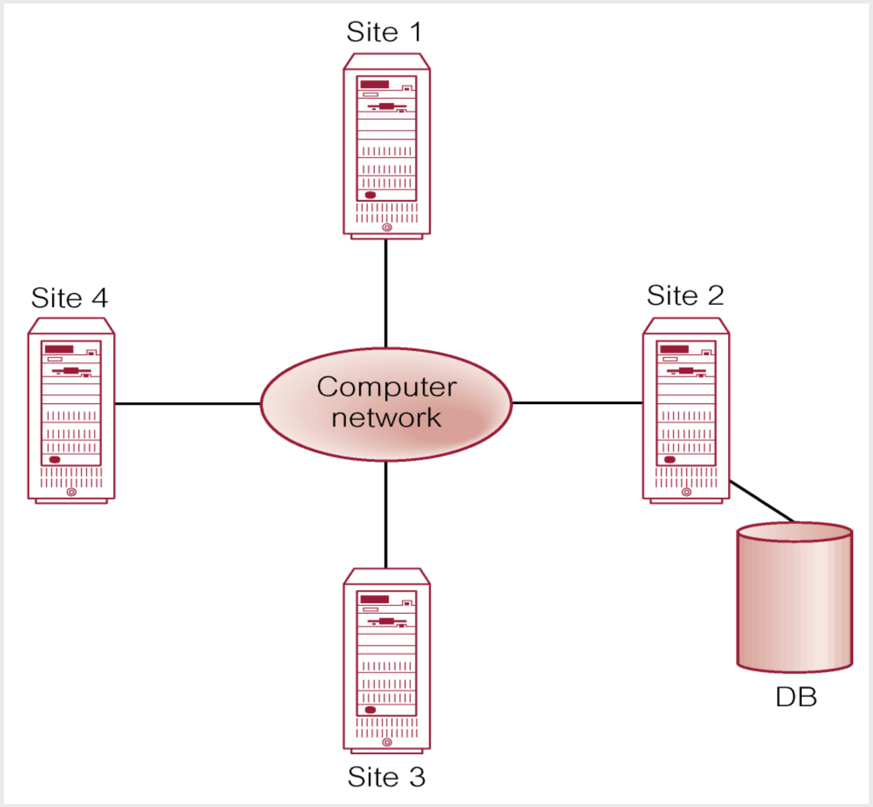
- It is not necessary for every site in the system to have its own local database.



## Distributed Processing

**Distributed Processing**

**A centralized database that can be accessed over a computer network.**



## Distributed DBMS vs. Distributed Processing

**Distributed DBMS**

- System consists of data that is physically distributed across a number of sites in the network.

**Distributed processing**

- Data is centralized, even though other users may be accessing the data over the network.

## Advantages of DDBMSs

- **Reflects organizational structure**
  - Many organizations are naturally distributed over several locations.
- **Improved shareability and local autonomy**
  - The database is brought nearer to its users. This can effect a cultural change as it allows potentially greater control over local data .
- **Improved availability**
  - In Distributed DBMSs  if a single node fails, the system may be able to reroute the failed node’s requests to another site.

## Advantages of DDBMSs

- **Improved reliability**
  - As data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible.
- **Improved performance**
  - the data is located near the site of 'greatest demand', speed of database access may be better than that achievable from a remote centralized database.
- **Economics**
  - It costs much less to create a system of smaller computers with the equivalent power of a single large computer.

## Advantages of DDBMSs

- **Modular growth**
  - In a distributed environment, it is much easier to handle expansion.
- **Integration**
  - The integration of legacy systems and software components of different vendors.
- **Remaining competitive**
  - A number of relatively recent developments that rely heavily on distributed database technology such as e-Business, computer-supported collaborative work.

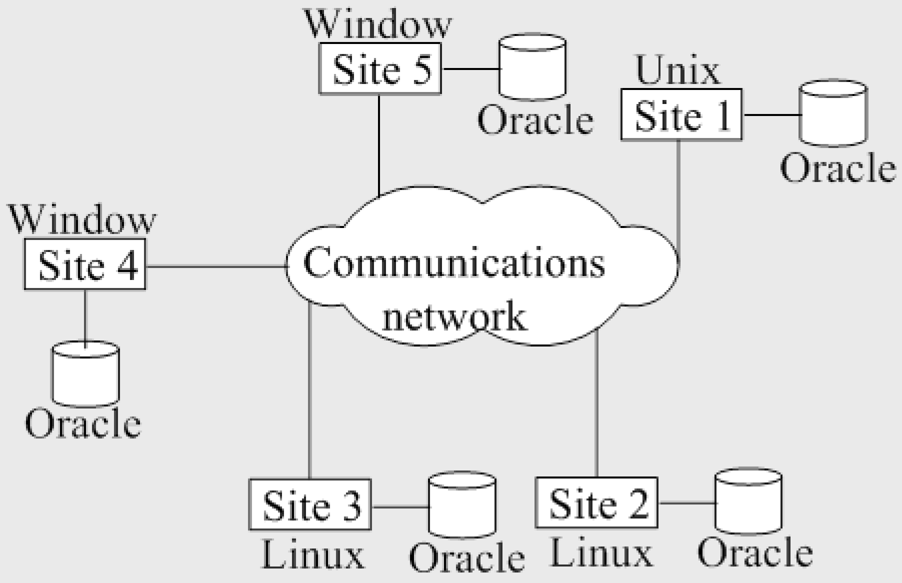
## Disadvantages of DDBMSs

- **Complexity**
  - A distributed DBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability, and availability is inherently more complex than a centralized DBMS.
- **Cost**
  - High maintenances costs for a DDBMS. A distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred.
- **Security**
  - In a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but the network itself has to be made secure.

## Disadvantages of DDBMSs

- **Integrity control more difficult**
  - In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be prohibitive.
- **Lack of standards**
  - we are only now starting to see the appearance of standard communication and data access protocols in DDBMS.
- **Lack of experience**
  - General-purpose distributed DBMSs have not been widely accepted
- **Database design more complex**

## Two Types of DDBMS (Homogeneous)

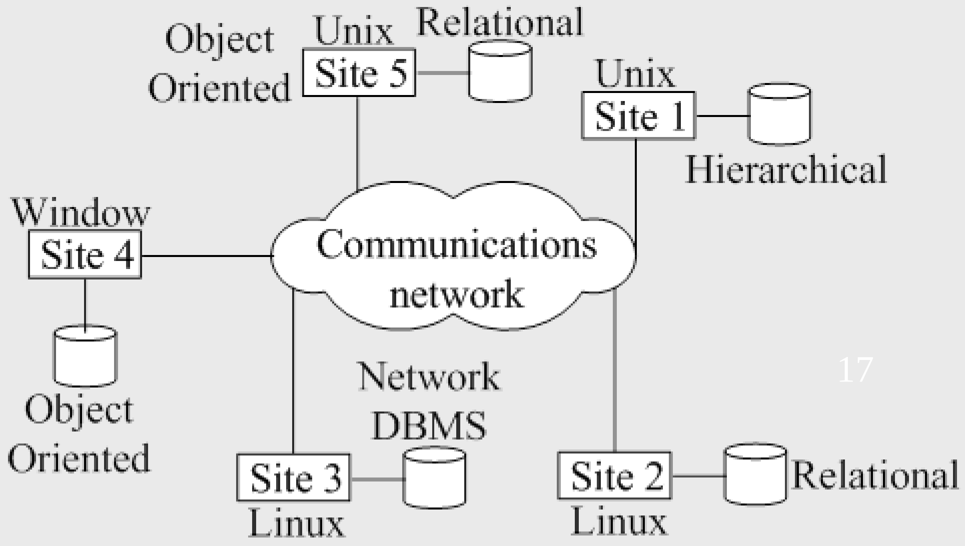


- **Homogeneous**
  - All sites of the database system have identical setup, i.e., same database system software.
  - The underlying operating system may be different.
    - For example, all sites run Oracle or DB2, or Sybase or some other database system.

- **Much easier to design and manage.**

## Two Types of DDBMS (Heterogenous)

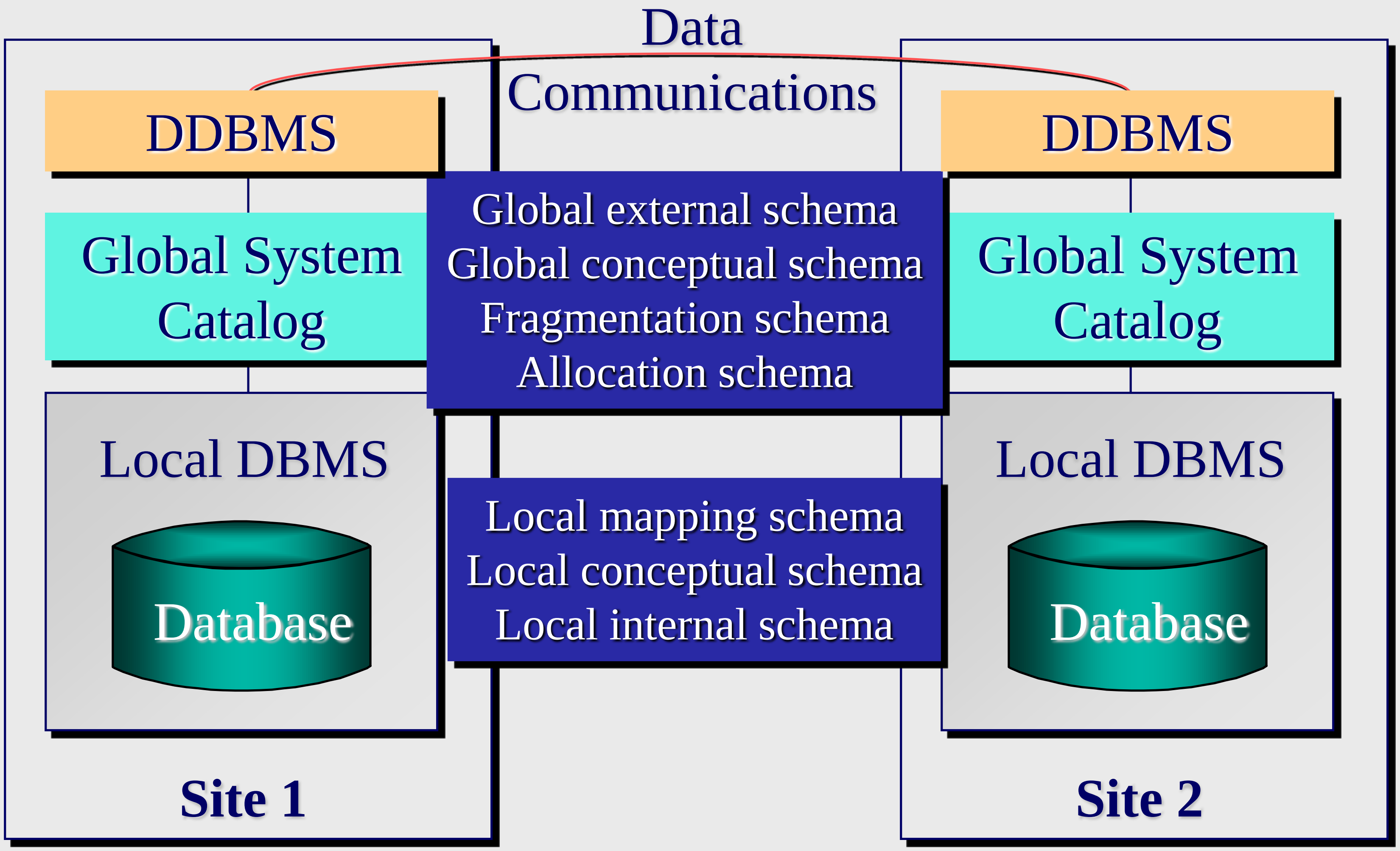
- **Heterogeneous DDBMS**
  - Sites may run different DBMS products, with possibly different underlying data models.
  - Occurs when sites have implemented their own databases and integration is considered later.
  - Translations required to allow communications between different DBMS products.
  - Typical solution is to use *gateways*.



## Reference Architecture for DDBMS

- Due to diversity of DDBMS system, no accepted architecture equivalent to ANSI/SPARC 3-level architecture.
- A *reference architecture* that only address *data distribution*, consists of:
  - Set of global external schemas.
  - Global conceptual schema (GCS).
  - Fragmentation schema and allocation schema.
  - Set of schemas for each local DBMS conforming to 3-level ANSI/SPARC.
- Some levels may be missing, depending on levels of transparency supported.

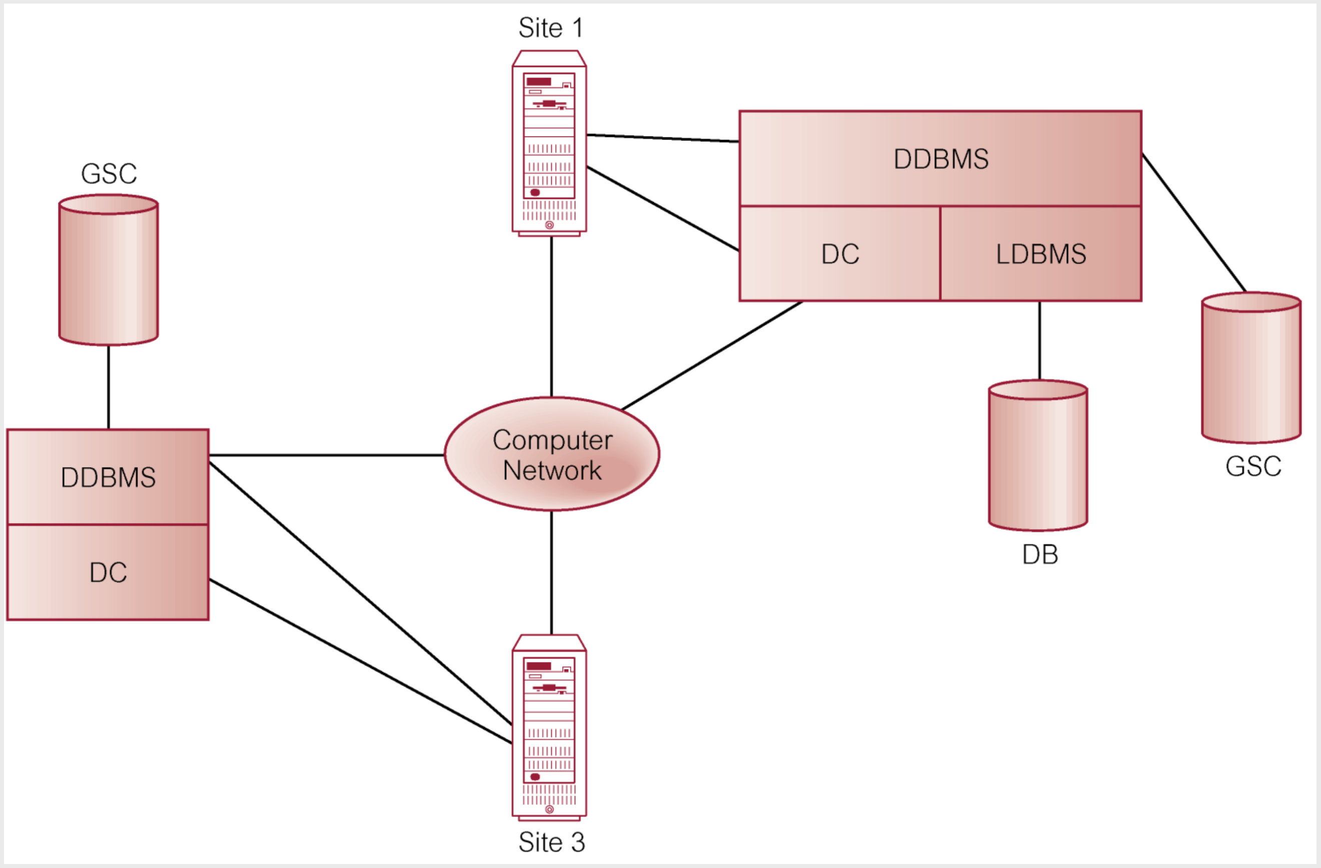
## DDBMS Architecture



## DDBMS Architecture

- Global Conceptual Schema
  - The global conceptual schema is a logical description of the whole database, as if it were not distributed.
- Fragmentation schema
  - The fragmentation schema is a description of how the data is to be logically partitioned.
- Allocation schema
  - The allocation schema is a description of where the data is to be located, taking account of any replication.
- Local schemas
  - Each local DBMS has its own set of schemas. It is DBMS independent and is the basis for supporting heterogeneous DBMSs.

## Components of a DDBMS



## Components of a DDBMS

- Local DBMS component
  - is a standard DBMS, responsible for controlling the local data at each site that has a database. It has its own local system catalog that stores information about the data held at that site.
- Data communications component
  - is the software that enables all sites to communicate with each other.
- Global system catalog
  - has the same functionality as the system catalog of a centralized system.
- Distributed DBMS component
  - is the controlling unit of the entire system.

## Functions of a DDBMS

- Expect DDBMS to have at least the functionality of a DBMS.
- Also to have following functionality:
  - Extended communication services
  - Extended system catalog.
  - Distributed query processing.
  - Extended concurrency control.
  - Extended recovery services.

## Distributed Database Design

- **Three Key Issues of distributed database design**

  **Fragmentation**

  Relation may be divided into a number of sub-relations, which are then distributed.

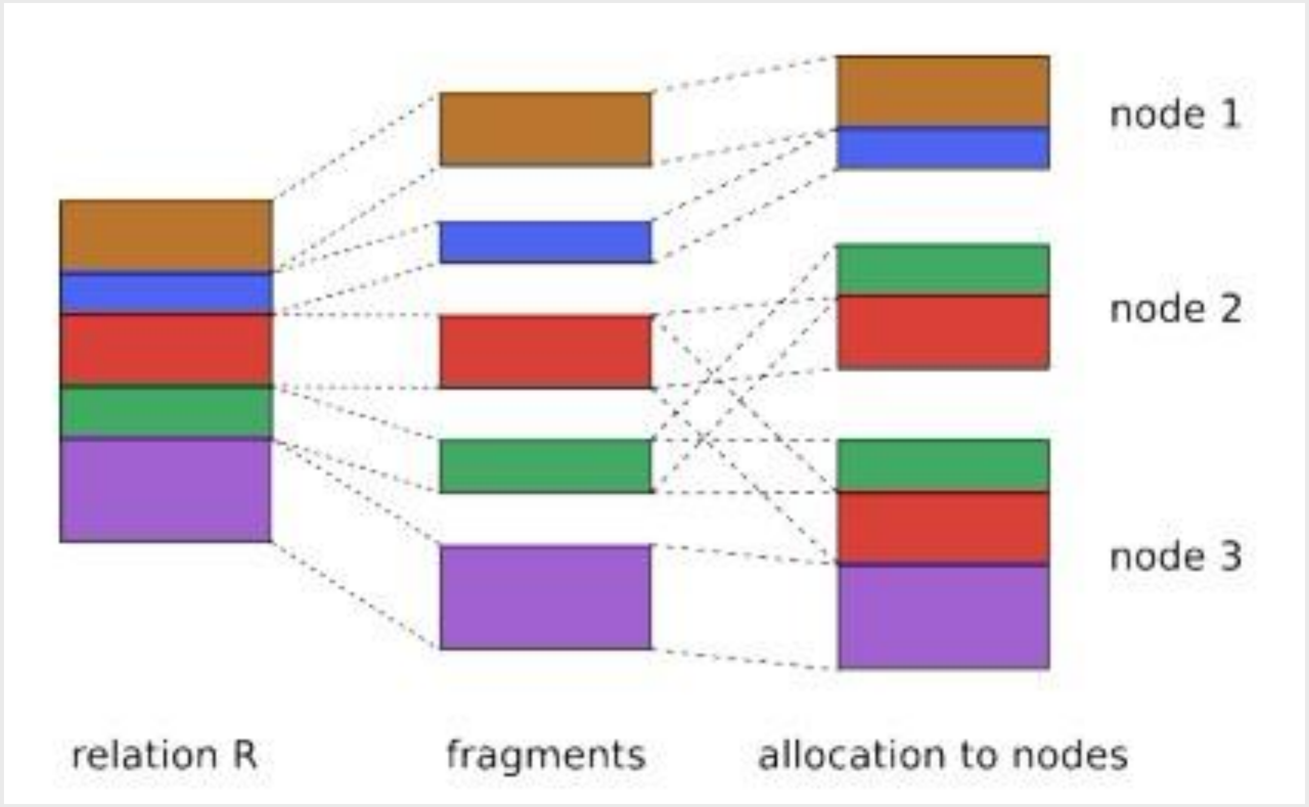
  **Replication**

  Copy of fragment may be maintained at several sites.

  **Allocation**

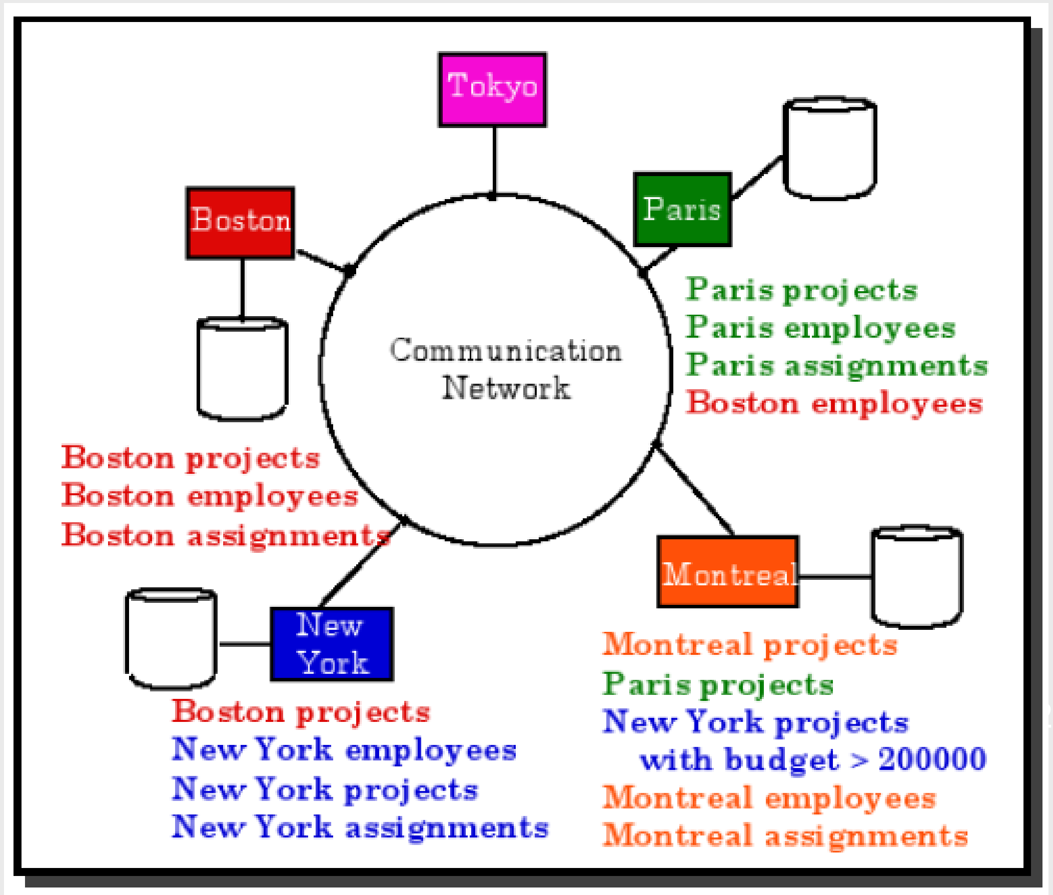
  Each fragment is stored at site with “optimal” distribution.

## Fragmentation and Allocation



## Example of Fragmentation and Replication

- Database consists of 3 relations ***employees, projects***, and ***assignment*** which are partitioned and stored at different sites (fragmentation).



## Replication ?

- **Relations** as unit of distribution:
  - If the relation is not replicated, we get a high volume of remote data accesses.
  - If the relation is replicated, we get unnecessary replications, which causes problems in executing updates and waste disk space.

## Why Fragmentation

- ***Usage***—application work with views rather than relations; Therefore, for data distribution, it seems appropriate to work with subsets of relations as the unit of distribution.
- ***Efficiency***– data is stored close to where it is most frequently used. In addition, data that is not needed by local applications is not stored.
- ***Parallelism***—increase the degree of concurrency. With fragments as the unit of distribution, a transaction can be divided into several subqueries that operate on fragments.
- ***Security***– data not required by local application is not stored and consequently not available

## Disadvantages of fragmentation

- ***Performance***– the performance of global applications that require data from several fragments located at different sites may be slower.
- ***Integrity***– Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.

## How to do Fragmentation

- Involves analyzing most important applications, based on quantitative/qualitative information.
  - Quantitative information may include:
    - » frequency of queries, sites;
    - » Where query is run;
    - » Selectivity of the queries, etc.
  - Qualitative information may include transactions that are executed by application, such as
    - » the relations, attributes, and tuples accessed;
    - » type of access (read or write)
    - » predicates of read operations.

## Data Replication

- **Replication**: Which fragments shall be stored as multiple copies?
  - Complete Replication: Complete copy of the database is maintained in each site
  - Selective Replication:  Selected fragments are replicated in some sites. This strategy is a combination of fragmentation, replication, and centralization.

## Data Allocation

- **Allocation**: On which sites to store the various fragments?
  - **Golden Rule:** place data as close as possible to where it will be used
  - Centralized: consists of a single DB and DBMS stored at one site with users distributed across the network (distributed processing).
  - Partitioned (fragmented): Database is partitioned into disjoint fragments, each fragment assigned to one site
    - » Vertical (by columns)
    - » Horizontal (by rows)
    - » Mixed (by columns and rows)

## Three Correctness Rules of Fragmentation

**There Rules that MUST be followed during fragmentation:**

**1. Completeness:** *Fragments contain all data.*

If relation *R* is decomposed into fragments $R_1$, $R_2$, ... $R_n$, each data item that can be found in *R* must appear in at least one fragment.

**2. Reconstruction:** *Fragments preserves the data and properties of the original relation.*

- Must be possible to define a relational operation that will reconstruct *R* from its fragments.
- Reconstruction for horizontal fragmentation is Union operation and Join for vertical .

## Correctness of Fragmentation

**3. Disjointness:** *Fragments do not overlap*

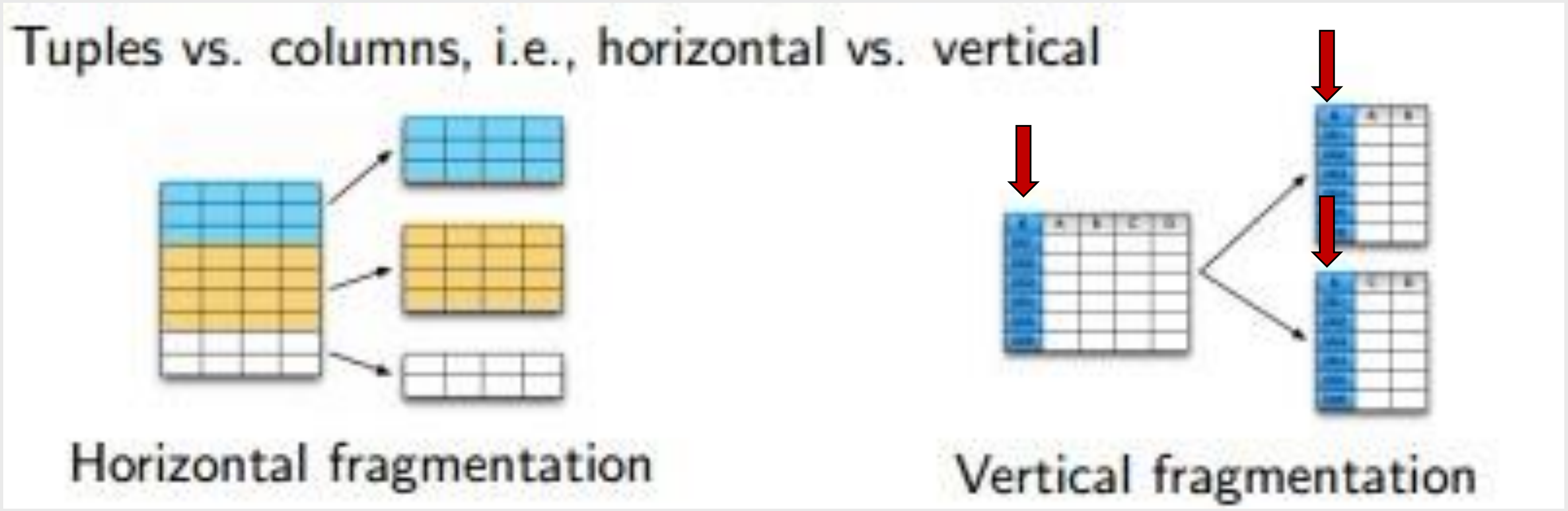
- If data item $d_i$ appears in fragment $R_i$, then it should not appear in any other fragment.
- Exception: vertical fragmentation, where primary key attributes must be repeated to allow reconstruction.
- For horizontal fragmentation, data item is a tuple.
- For vertical fragmentation, data item is an attribute.

## Types of Fragmentation

- **Four types of fragmentation:**
  - **Horizontal**-consists of a subset of the tuples of a relation.
  - **Vertical**-consists of a subset of the attributes of a relation.
  - **Mixed** - Consists of a horizontal fragment that is subsequently vertically fragmented, or a vertical fragment that is then horizontally fragmented.
  - **Derived** - A horizontal fragment that is based on the horizontal fragmentation of a parent relation.
- **Other possibility is no fragmentation:**
  - If relation is small and not updated frequently, may be better not to fragment relation.

# Horizontal and Vertical Fragmentation

- Note: **every vertical fragment contain Primary key.**



## Horizontal Fragmentation

- **Intuition** behind horizontal fragmentation
  - Every site should hold all information that is used to query at the site
  - The information at the site should be fragmented so the queries of the site run faster
- **Consists of a subset of the tuples of a relation.**
  - Tuples are distributed among fragments
  - Defined using *Selection* operation of relational algebra:

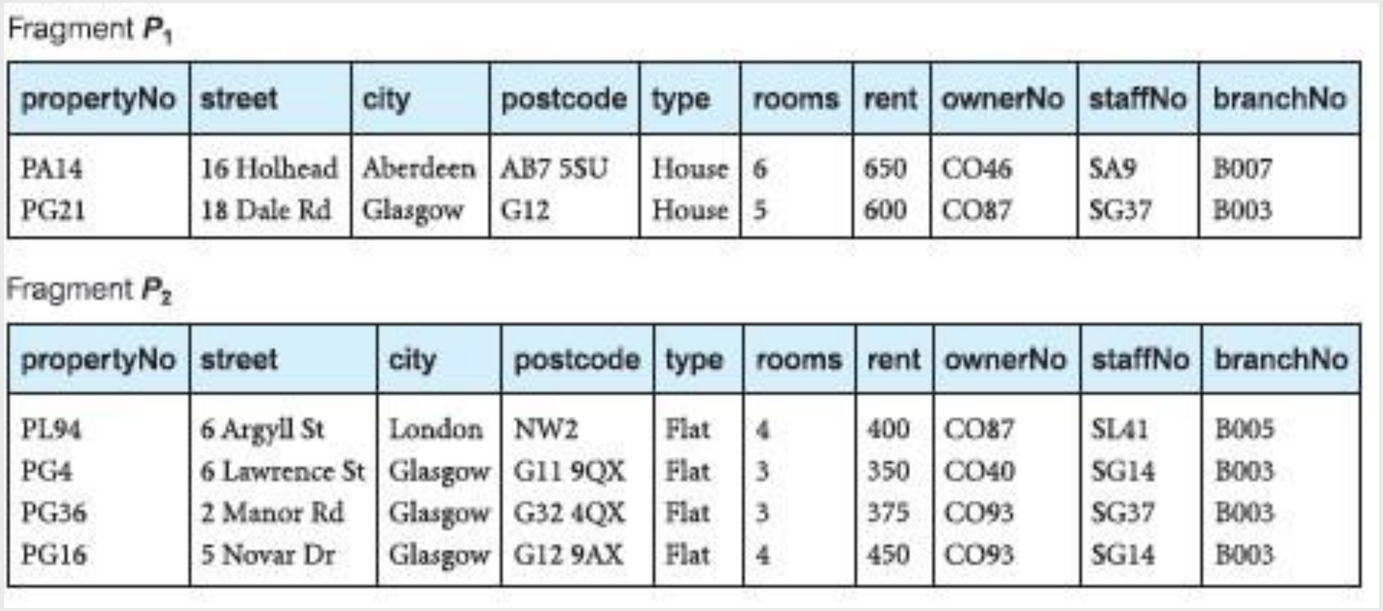
$$\boldsymbol{\sigma_p(R)}$$

where p is referred to as fragmentation predicate

## Example of Horizontal Fragmentation

- For example: the value of the type attribute is “house” or “flat”.

$$P_1 = \sigma_{type='House'}(PropertyForRent)$$
$$P_2 = \sigma_{type='Flat'}(PropertyForRent)$$



Fragment $P_1$

| propertyNo | street | city | postcode | type | rooms | rent | ownerNo | staffNo | branchNo |
|---|---|---|---|---|---|---|---|---|---|
| PA14 | 16 Holhead | Aberdeen | AB7 5SU | House | 6 | 650 | CO46 | SA9 | B007 |
| PG21 | 18 Dale Rd | Glasgow | G12 | House | 5 | 600 | CO87 | SG37 | B003 |

Fragment $P_2$

| propertyNo | street | city | postcode | type | rooms | rent | ownerNo | staffNo | branchNo |
|---|---|---|---|---|---|---|---|---|---|
| PL94 | 6 Argyll St | London | NW2 | Flat | 4 | 400 | CO87 | SL41 | B005 |
| PG4 | 6 Lawrence St | Glasgow | G11 9QX | Flat | 3 | 350 | CO40 | SG14 | B003 |
| PG36 | 2 Manor Rd | Glasgow | G32 4QX | Flat | 3 | 375 | CO93 | SG37 | B003 |
| PG16 | 5 Novar Dr | Glasgow | G12 9AX | Flat | 4 | 450 | CO93 | SG14 | B003 |

## Example of Horizontal Fragmentation

- ***Completeness***
  - Each tuple in the relation appears in either fragment P1 or P2.
- ***Reconstruction***
  - The PropertyForRent relation can be reconstructed from the fragments using the Union operation, thus:

$$P1 \cup P2 = \text{PropertyForRent}$$

- ***Disjointness***
  - The fragments are disjoint; there can be no property type that is both ‘House’ and ‘Flat’.

## Horizontal Fragmentation Strategy

- Horizontal fragmentation strategy is determined by looking at
  - predicates used by transactions.
    - Simple– single attributes
    - Complex– multiple attributes
  - Or queries in the applications.

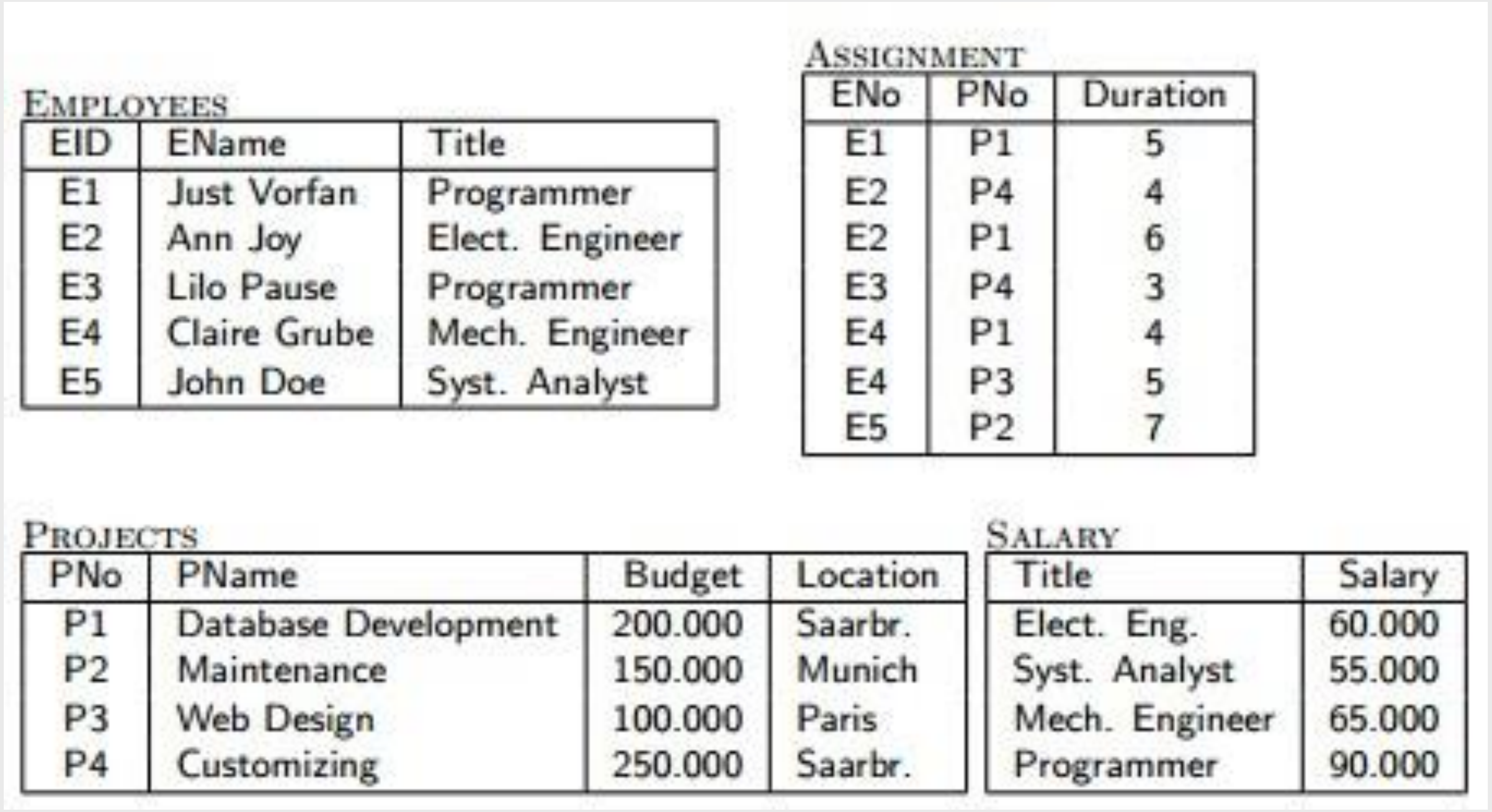
## Vertical Fragmentation

- **Objective** of vertical fragmentation is to partition a relation into a set of smaller relations so that many of the applications will run on only one fragment.
- **Consists of a subset of attributes of a relation.**
  - The relation is split vertically, i.e., attributes are distributed among fragments
  - Defined using ***Projection*** operation of relational algebra:

$$\Pi_{\mathbf{a1, \ldots, an}}(\mathbf{R})$$

  - Primary key attributes are replicated in each fragmentation.
- Vertical fragmentation has also been studied for (centralized) DBMS
  - Smaller relations, and hence less page accesses

## Database Example



EMPLOYEES

| EID | EName | Title |
|---|---|---|
| E1 | Just Vorfan | Programmer |
| E2 | Ann Joy | Elect. Engineer |
| E3 | Lilo Pause | Programmer |
| E4 | Claire Grube | Mech. Engineer |
| E5 | John Doe | Syst. Analyst |

ASSIGNMENT

| ENo | PNo | Duration |
|---|---|---|
| E1 | P1 | 5 |
| E2 | P4 | 4 |
| E2 | P1 | 6 |
| E3 | P4 | 3 |
| E4 | P1 | 4 |
| E4 | P3 | 5 |
| E5 | P2 | 7 |

PROJECTS

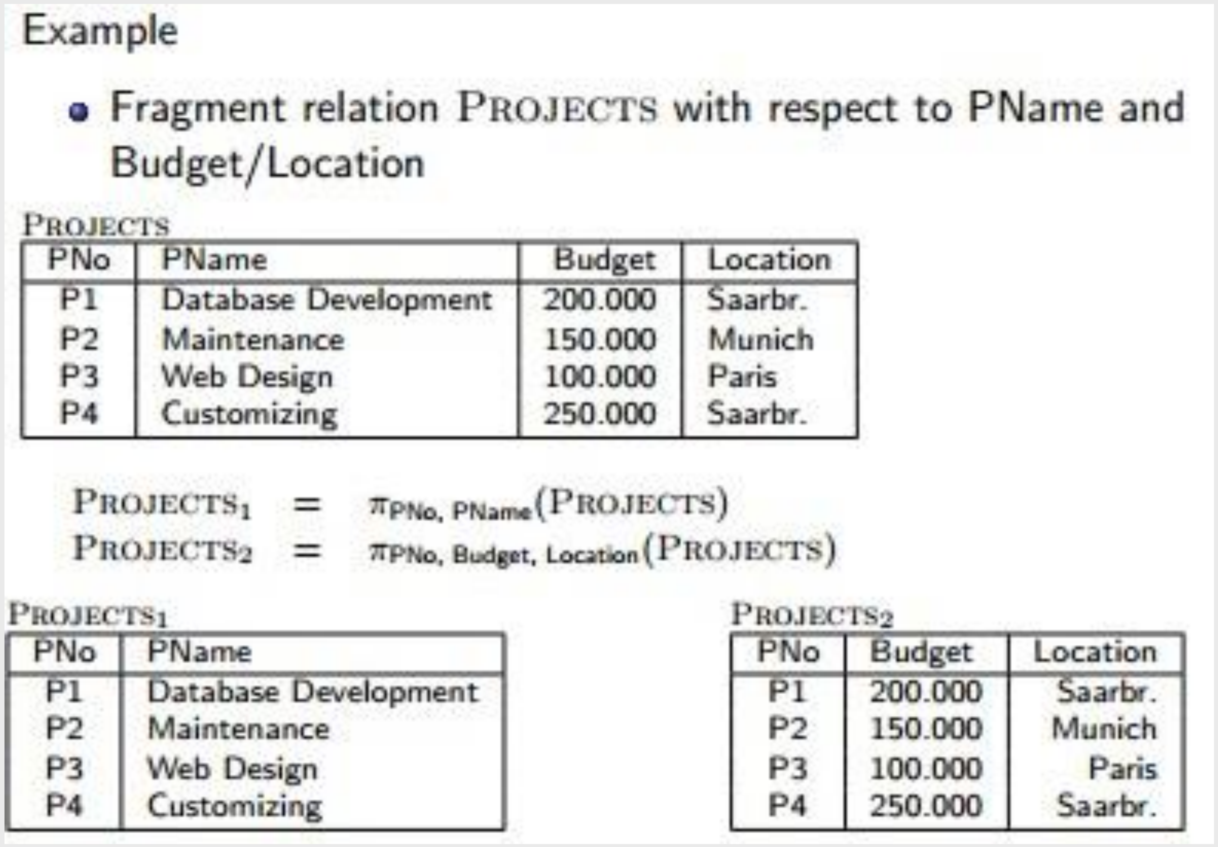
| PNo | PName | Budget | Location |
|---|---|---|---|
| P1 | Database Development | 200.000 | Saarbr. |
| P2 | Maintenance | 150.000 | Munich |
| P3 | Web Design | 100.000 | Paris |
| P4 | Customizing | 250.000 | Saarbr. |

SALARY

| Title | Salary |
|---|---|
| Elect. Eng. | 60.000 |
| Syst. Analyst | 55.000 |
| Mech. Engineer | 65.000 |
| Programmer | 90.000 |

## Example of Vertical Fragmentation

- **Note: both fragments contain the primary key , PNo, to enable the original relation to be reconstructed**



Example

- Fragment relation PROJECTS with respect to PName and Budget/Location

PROJECTS

| PNo | PName | Budget | Location |
|---|---|---|---|
| P1 | Database Development | 200.000 | Saarbr. |
| P2 | Maintenance | 150.000 | Munich |
| P3 | Web Design | 100.000 | Paris |
| P4 | Customizing | 250.000 | Saarbr. |

$$\text{PROJECTS}_1 = \pi_{\text{PNo, PName}}(\text{PROJECTS})$$
$$\text{PROJECTS}_2 = \pi_{\text{PNo, Budget, Location}}(\text{PROJECTS})$$

PROJECTS1

| PNo | PName |
|---|---|
| P1 | Database Development |
| P2 | Maintenance |
| P3 | Web Design |
| P4 | Customizing |

PROJECTS2

| PNo | Budget | Location |
|---|---|---|
| P1 | 200.000 | Saarbr. |
| P2 | 150.000 | Munich |
| P3 | 100.000 | Paris |
| P4 | 250.000 | Saarbr. |

## Correctness Rules

- This fragmentation schema satisfies the correctness rules
  - *Completeness:* Each attribute in the PROJECTS relation appears in either fragment $PROJECTS_1$ or $PROJECTS_2$.
  - *Reconstruction:* The PROJECTS relation can be reconstructed from the fragments using the Natural join operation, thus:

$$PROJECTS_1 \bowtie PROJECTS_2 = PROJECTS$$

  - *Disjointness:* The fragments are disjoint except for the primary key, which is necessary for reconstruction.

## Mixed Fragmentation

- In most cases simple horizontal or vertical fragmentation of a DB schema will not be sufficient to satisfy the requirements of the applications.
- **Mixed fragmentation (hybrid fragmentation):** Consists of a horizontal fragment that is vertically fragmented, or a vertical fragment that is horizontally fragmented.
- **Defined using *Selection* and *Projection* operations of relational algebra:**

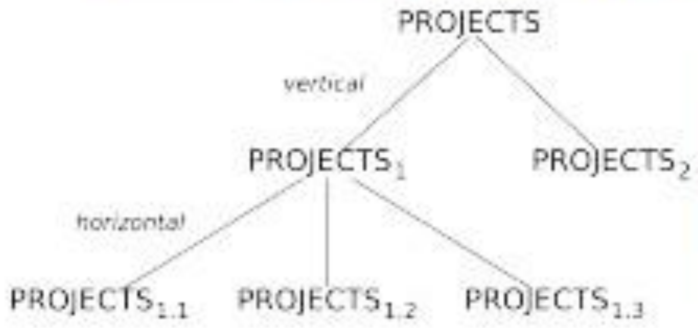
$$\sigma_{p}(\Pi_{a1, \ldots, an}(R))$$

**or**

$$\Pi_{a1, \ldots, an}(\sigma_{p}(R))$$

## Example - Mixed Fragmentation

- A fragment of a relation is a relation itself
- Fragments can be further fragmented
- Combination of horizontal and vertical fragmentation possible



$$\begin{aligned}
\text{PROJECTS}_1 &= \pi_{\text{PNo, PName, Location}}(\text{PROJECTS}) \\
\text{PROJECTS}_2 &= \pi_{\text{PNo, Budget}}(\text{PROJECTS}) \\
\text{PROJECTS}_{1.1} &= \sigma_{\text{Location='Saarbr.'}}(\text{PROJECTS}_1) \\
\text{PROJECTS}_{2.1} &= \sigma_{\text{Location='Munich'}}(\text{PROJECTS}_1) \\
\text{PROJECTS}_{3.1} &= \sigma_{\text{Location='Paris'}}(\text{PROJECTS}_1)
\end{aligned}$$

$$\text{PROJECTS} = (\text{PROJECTS}_{1.1} \cup \text{PROJECTS}_{1.2} \cup \text{PROJECTS}_{1.3}) \bowtie \text{PROJECTS}_2$$

## Correctness Rules

- ***Completeness***
  - Each attribute in the PROJECT relation appears in either fragments PROJECT1 or PROJECT2; each (part) tuple appears in fragment PROJECT2 and either fragment PROJECT11, PROJECT21, or PROJECT31.
- ***Reconstruction***
  - The PROJECT relation can be reconstructed from the fragments using the Union and Natural join operations.
- ***Disjointness***
  - The fragments are disjoint; there can be no staff member who works in more than one branch and PROJECT1 and PROJECT2 are disjoint except for the necessary duplication of primary key.

## Review: Semijoin

- $R \rhd_F S$
  - Defines a relation that contains the tuples of R that participate in the join of R with S.

$$R \rhd_F S = \Pi_A(R \bowtie_F S)$$

## Derived Horizontal Fragmentation

- **Intuition:** Some applications may involve a join of two or more relations. If the relations are stored at different locations, there may be a significant overhead in processing the join.
- In such cases, it may be more appropriate to ensure that the relations, or fragments of relations, are at the same location.

- **A horizontal fragment that is based on horizontal fragmentation of a parent relation.**
  - Splitting up a relation in dependence on another relation

# Derived Horizontal Fragmentation

- ***Child*** refer to the relation that contains the foreign key and ***parent*** to the relation containing the targeted primary key.

- Given a child relation R and parent S, the derived fragmentation of ***R*** is defined as:

$$\mathbf{R_i = \boldsymbol{R} \rhd_{F} \boldsymbol{S_i},} \qquad \mathbf{1 \leq i \leq w}$$

where *w* is the number of horizontal fragments defined on *S* and F is the join attribute.

## Derived Horizontal Fragmentation

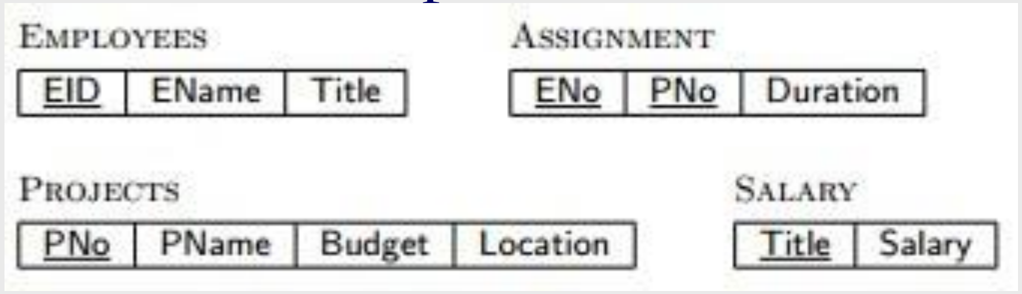
- **Goal:**
  - Horizontal fragmentation of relation S based on the fragmentation of another relation R.
- **Given**
  - Relation R and S
  - R.A is foreign key in S
  - R is already fragmented into $R_1$ , $R_2$,… $R_n$
- **Result from derived horizontal fragmentation**
  - Relation S is partitioned with respect to the fragments of R.

## Derived Horizontal Fragmentation

- Fragmentation of S based on the fragments of R ($R_1$, $R_2$,..., $R_n$)
  - Using semijoin operator
  - $S_i = S \rhd R_i = S \rhd \sigma_{P_i}(R)$
  - Fragmentation expression $\sigma_{P_i}$ only refers to R
  - If relation contains more than one foreign key, need to select one as parent.
  - Choice can be based on fragmentation used most frequently or fragmentation with better join characteristics.

## Example

Consider multiple relations:


EMPLOYEES

| <u>EID</u> | EName | Title |
|---|---|---|

ASSIGNMENT

| <u>ENo</u> | <u>PNo</u> | Duration |
|---|---|---|

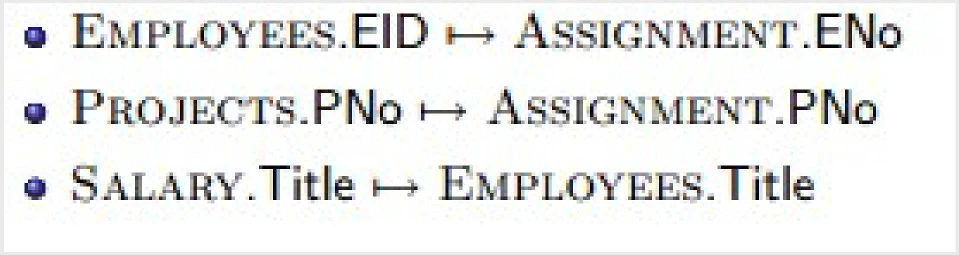
PROJECTS

| <u>PNo</u> | PName | Budget | Location |
|---|---|---|---|

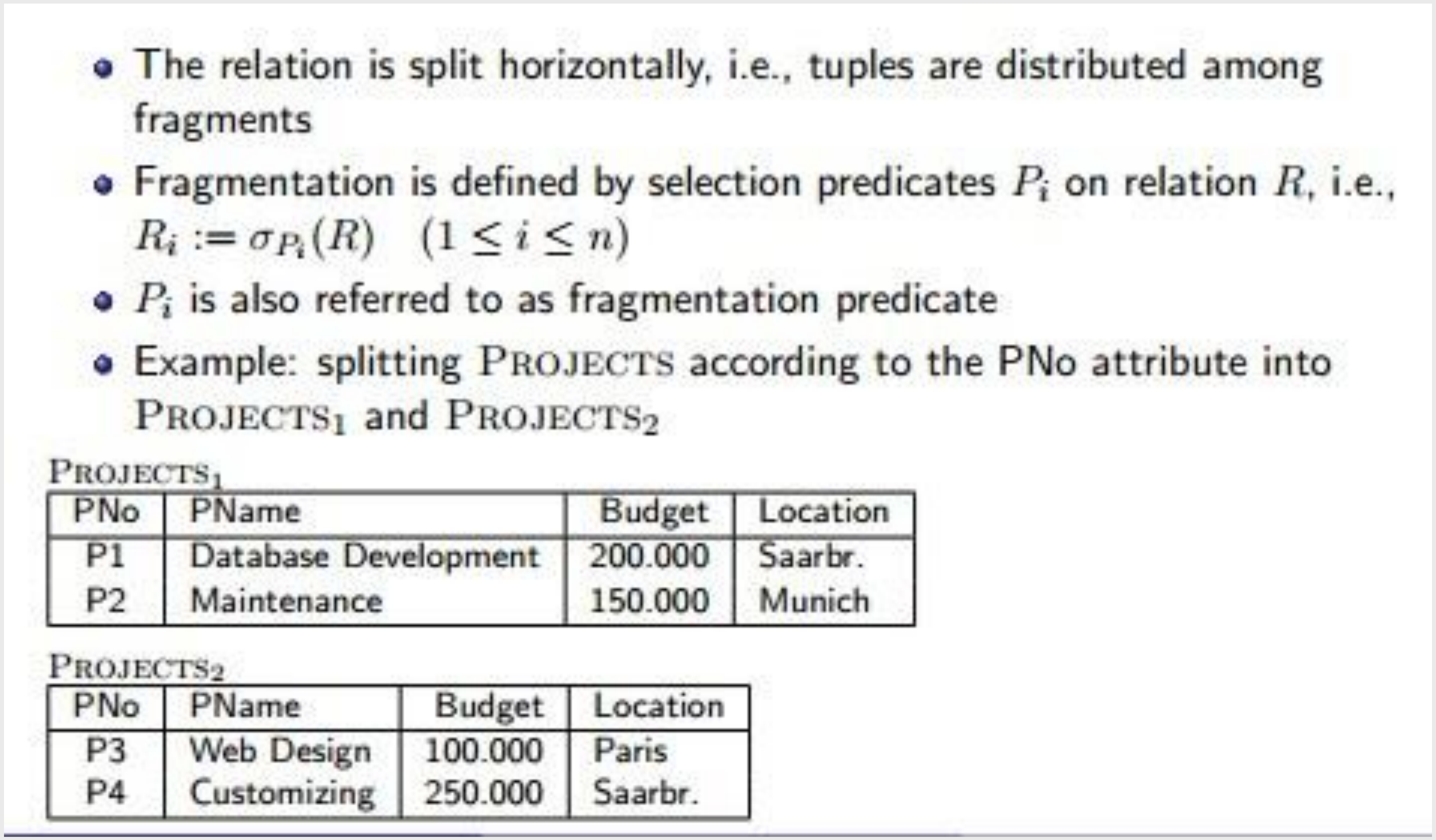
SALARY

| <u>Title</u> | Salary |
|---|---|

Foreign key relationship:


- EMPLOYEES.EID $\mapsto$ ASSIGNMENT.ENo
- PROJECTS.PNo $\mapsto$ ASSIGNMENT.PNo
- SALARY.Title $\mapsto$ EMPLOYEES.Title

## Example of Horizontal Fragmentation



- The relation is split horizontally, i.e., tuples are distributed among fragments
- Fragmentation is defined by selection predicates $P_i$ on relation $R$, i.e., $R_i := \sigma_{P_i}(R) \quad (1 \leq i \leq n)$
- $P_i$ is also referred to as fragmentation predicate
- Example: splitting PROJECTS according to the PNo attribute into $\text{PROJECTS}_1$ and $\text{PROJECTS}_2$

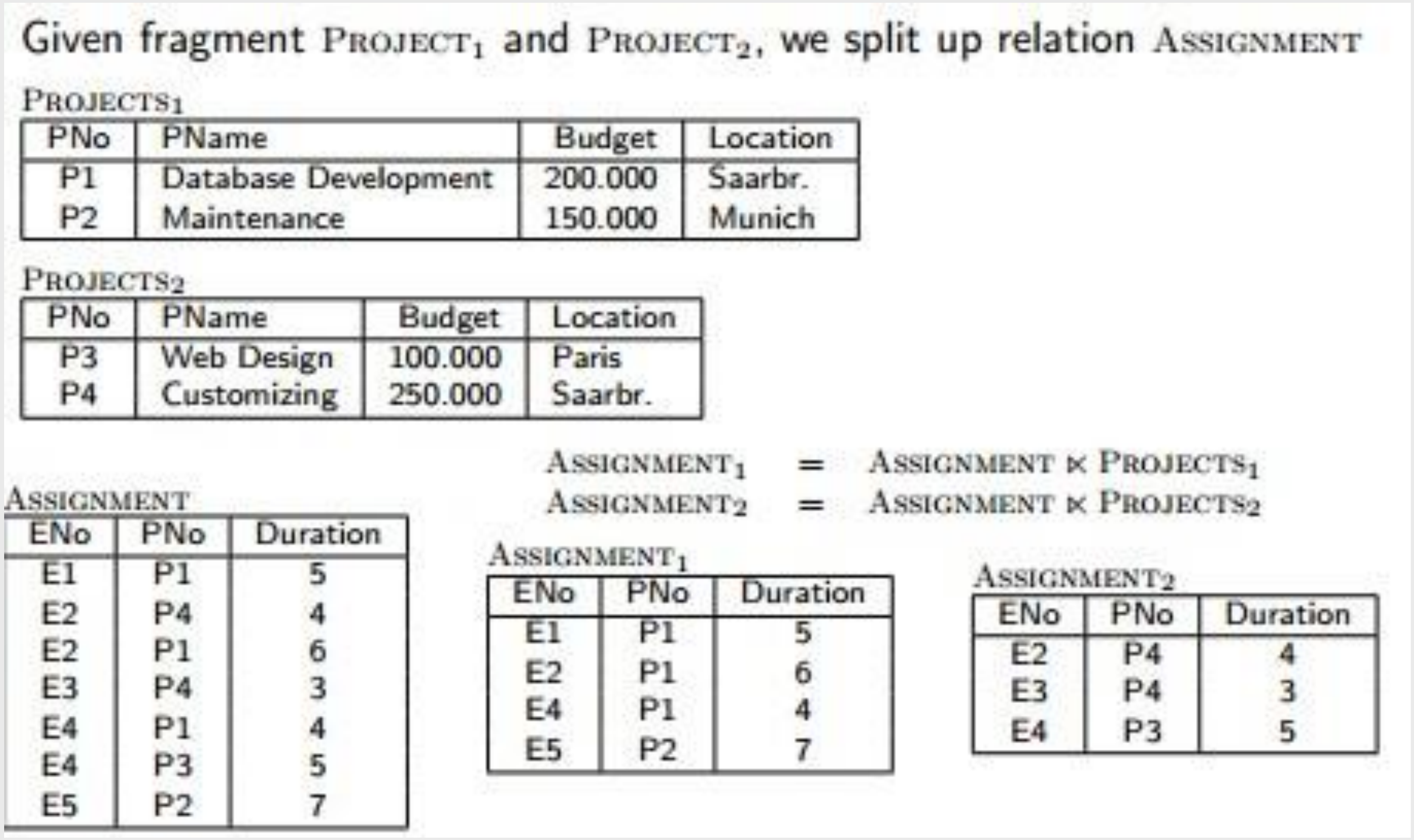
$\text{PROJECTS}_1$

| PNo | PName | Budget | Location |
|---|---|---|---|
| P1 | Database Development | 200.000 | Saarbr. |
| P2 | Maintenance | 150.000 | Munich |

$\text{PROJECTS}_2$

| PNo | PName | Budget | Location |
|---|---|---|---|
| P3 | Web Design | 100.000 | Paris |
| P4 | Customizing | 250.000 | Saarbr. |

# Example Derived Horizontal Fragmentation



Given fragment PROJECT$_1$ and PROJECT$_2$, we split up relation ASSIGNMENT

PROJECTS$_1$

| PNo | PName | Budget | Location |
|---|---|---|---|
| P1 | Database Development | 200.000 | Saarbr. |
| P2 | Maintenance | 150.000 | Munich |

PROJECTS$_2$

| PNo | PName | Budget | Location |
|---|---|---|---|
| P3 | Web Design | 100.000 | Paris |
| P4 | Customizing | 250.000 | Saarbr. |

$$\text{ASSIGNMENT}_1 = \text{ASSIGNMENT} \ltimes \text{PROJECTS}_1$$
$$\text{ASSIGNMENT}_2 = \text{ASSIGNMENT} \ltimes \text{PROJECTS}_2$$

ASSIGNMENT

| ENo | PNo | Duration |
|---|---|---|
| E1 | P1 | 5 |
| E2 | P4 | 4 |
| E2 | P1 | 6 |
| E3 | P4 | 3 |
| E4 | P1 | 4 |
| E4 | P3 | 5 |
| E5 | P2 | 7 |

ASSIGNMENT$_1$

| ENo | PNo | Duration |
|---|---|---|
| E1 | P1 | 5 |
| E2 | P1 | 6 |
| E4 | P1 | 4 |
| E5 | P2 | 7 |

ASSIGNMENT$_2$

| ENo | PNo | Duration |
|---|---|---|
| E2 | P4 | 4 |
| E3 | P4 | 3 |
| E4 | P3 | 5 |

## No Fragmentation

- A final strategy is not to fragment a relation.
- For example, the Branch relation contains only a small number of tuples and is not updated very frequently. Rather than trying to horizontally fragment the relation on, for example, branch number, it would be more sensible to leave the relation whole and simply replicate the Branch relation at each site.

## Summary of Fragmentation Variations

- Horizontal fragmentation
  - Defined by predicates referring to relation R
- Vertical fragmentation
  - Partitioning attributes of relation R
- Derived horizontal fragmentation
  - Horizontal fragmentation relation R is defined by predicates referring to another relation S
- Mixed fragmentation
  - Combinations of different types of fragmentation

## Transparencies in a DDBMS

- **DDBMS system should make the distribution transparent to the user.**
  - Distribution Transparency
    - The users should not be aware of where the data are allocated.
  - Transaction Transparency
    - All distributed transactions maintain the distributed database's integrity and consistency. (concurrency transparency and failure transparency)
  - Performance Transparency
    - DDBMS must perform as if it were a centralized DBMS
  - DBMS transparency
    - Hides the knowledge that the local DBMS may be different (applicable to heterogeneous DDBMS)

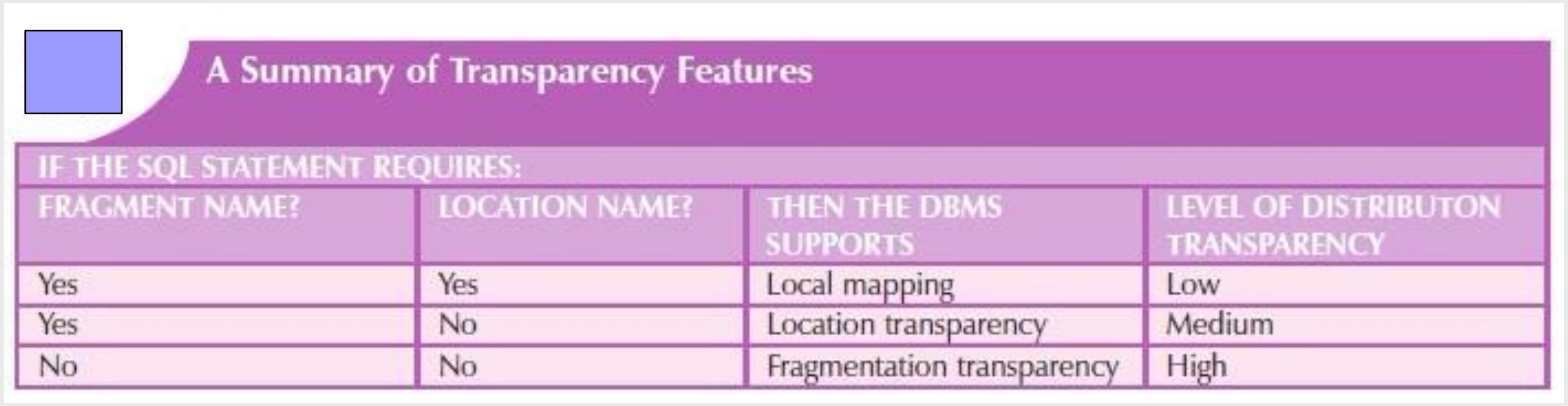
**Overall Objective: to make the use of the distributed database equivalent to that of a centralized database.**

## Distribution Transparency

- **Distribution transparency allows user to perceive database as single, logical entity.**
- If DDBMS exhibits distribution transparency, user does not need to know:
  - data is fragmented (fragmentation transparency),
  - location of data items (location transparency),
  - If the user need to know that the data is fragmented and the location of fragments, we call this local mapping transparency.
- With replication transparency, user is unaware of replication of fragments .

## Summary of Distribution Transparency Features

- Local Mapping Transparency is the lowest level of distribution transparency. user needs to specify both fragment names and the location of data items, taking into consideration any replication that may exist.

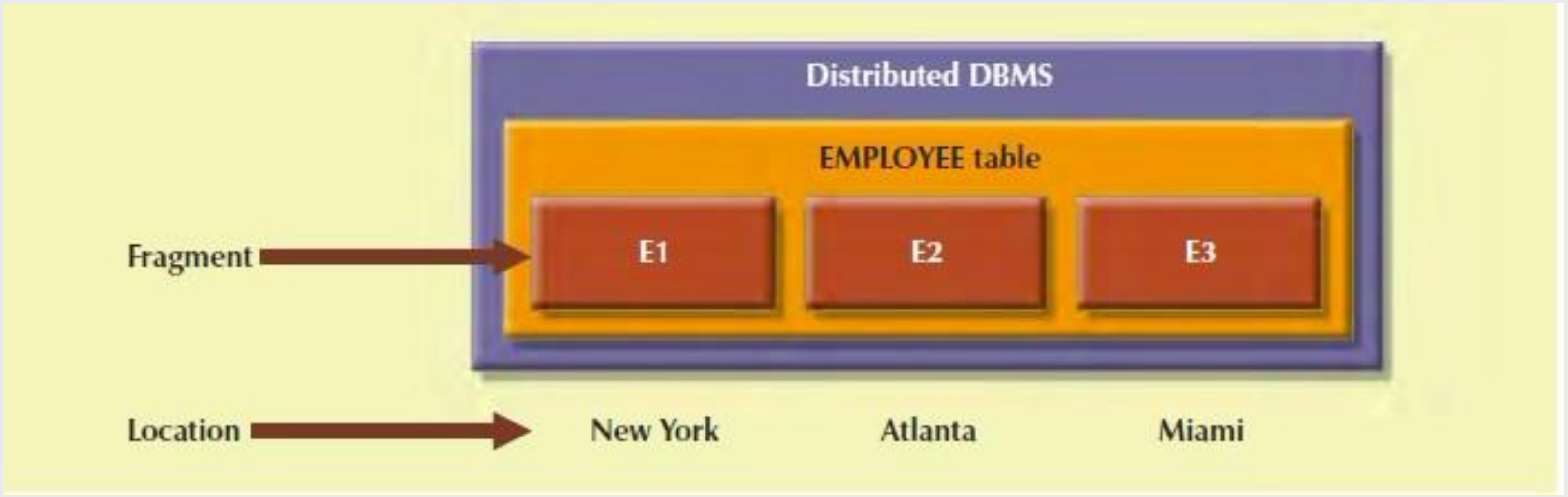


A Summary of Transparency Features

| IF THE SQL STATEMENT REQUIRES: | | | |
|---|---|---|---|
| FRAGMENT NAME? | LOCATION NAME? | THEN THE DBMS SUPPORTS | LEVEL OF DISTRIBUTON TRANSPARENCY |
| Yes | Yes | Local mapping | Low |
| Yes | No | Location transparency | Medium |
| No | No | Fragmentation transparency | High |

## Example

- Fragment Location



## Case 1: The Database Supports Fragmentation Transparency

- The query conforms to a non-distributed database query format; that is, it does NOT specify fragment names or locations.

**The query reads:**

```
SELECT *
FROM EMPLOYEE
WHERE EMP_DOB < '01-JAN-196';
```

## Case 2: The Database Supports Location Transparency

- Fragment names must be specified in the query, but the fragment's location is not specified.

  **The query reads:**

```
SELECT *
FROM E1
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E2
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E3
WHERE EMP_DOB < '01-JAN-1960';
```

## Case 3: The Database Supports Local Mapping Transparency

- Both the fragment name and its location must be specified in the query.

**Using pseudo-SQL:**

```
SELECT *
FROM El NODE NY
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E2 NODE ATL
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E3 NODE MIA
WHERE EMP_DOB < '01-JAN-1960';
```

## Naming Transparency

- As in a centralized database, each item in a distributed database must have a unique name. Therefore, the DDBMS must ensure that no two sites create a database object with the same name.
- **One solution:** a central name server.
  - Loss of some local autonomy
  - Performance problems, if the central site becomes a bottleneck;
  - Low availability; if the central site fails, the remaining sites can not create any new database objects.

## Alternative solution

- **Prefix object with identifier of site that created it.**

For example

- Branch created at site $S_1$ might be named S1.BRANCH.
- Also need to identify each fragment and its copies.
- Thus, copy 2 of fragment 3 of Branch created at site $S_1$ might be referred to as S1.BRANCH.F3.C2.
- However, this results in loss of distribution transparency.

## Another Approach

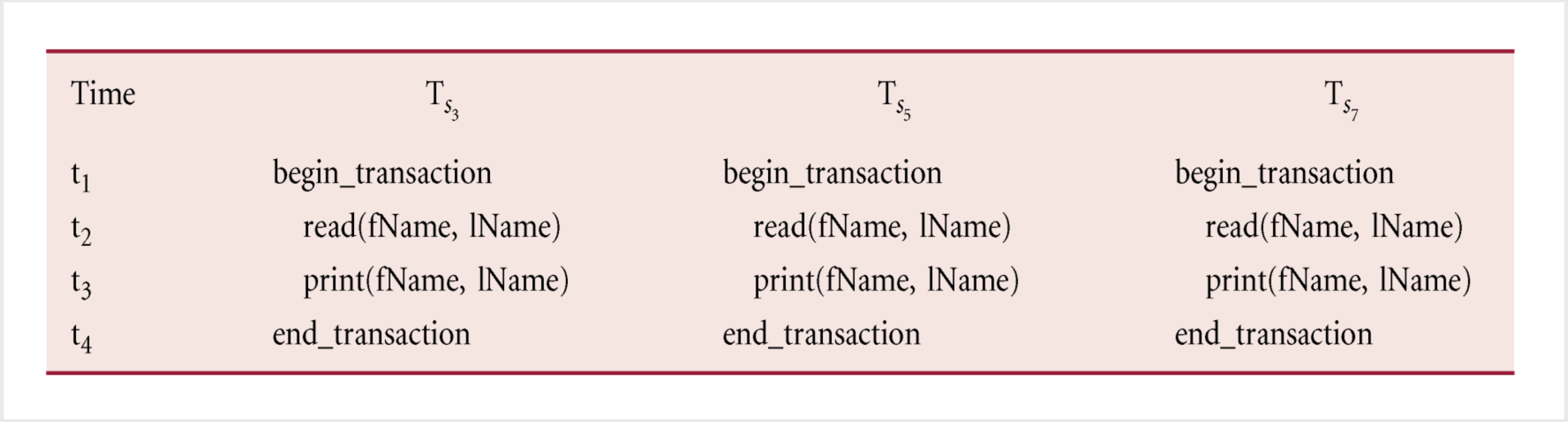
- An approach that resolves these problems uses aliases for each database object.
  - Thus, S1.BRANCH.F3.C2 might be known as LocalBranch by user at site $S_1$.
  - DDBMS has task of mapping an alias to appropriate database object.

## Transaction Transparency

- Ensures that all distributed transactions maintain distributed database's integrity and consistency.
  - Distributed transaction accesses data stored at more than one location.
  - Each transaction is divided into number of subtransactions, one for each site that has to be accessed.
  - Subtransaction is represented by an **agent**
  - DDBMS must ensure the indivisibility of both the global transaction and each of the subtransactions.
- Two aspects of transaction transparency: concurrency transparency and failure transparency.

## Example - Distributed Transaction

- T prints out names of all staff, staff relation are divided into fragments as $S_1$, $S_2$, $S_{21}$, $S_{22}$, and $S_{23}$. Staff names information are stored in $S_{21}$, $S_{22}$, and $S_{23.}$ at sites 3,5, and 7. Define three subtransactions $T_{S3}$, $T_{S5}$, and $T_{S7}$ to represent agents at sites 3, 5, and 7.



| Time | $T_{s_3}$ | $T_{s_5}$ | $T_{s_7}$ |
|---|---|---|---|
| $t_1$ | begin_transaction | begin_transaction | begin_transaction |
| $t_2$ | read(fName, lName) | read(fName, lName) | read(fName, lName) |
| $t_3$ | print(fName, lName) | print(fName, lName) | print(fName, lName) |
| $t_4$ | end_transaction | end_transaction | end_transaction |

## Concurrency Transparency

- **All transactions must execute independently and be logically consistent with results obtained if transactions executed one at a time, in some arbitrary serial order.**
  - Same fundamental principles as for centralized DBMS.
  - DDBMS must ensure both global and local transactions do not interfere with each other.
  - Similarly, DDBMS must ensure consistency of all subtransactions of global transaction.

## Concurrency Transparency

- Replication makes concurrency more complex.
  - If a copy of a replicated data item is updated, update must be propagated to all copies.
  - Could propagate changes as part of original transaction, making it an atomic operation.
  - However, if one site holding copy is not reachable, then transaction is delayed until site is reachable.

## Failure Transparency

- DDBMS must ensure atomicity and durability of global transaction.
  - Means ensuring that subtransactions of global transaction either all commit or all abort.
  - Thus, DDBMS must synchronize global transaction to ensure that all subtransactions have completed successfully before recording a final COMMIT for global transaction.
  - Must do this in presence of site and network failures.

## Performance Transparency

- DDBMS must perform as if it were a centralized DBMS.
  - DDBMS should not suffer any performance degradation due to distributed architecture.
  - DDBMS should determine most cost-effective strategy to execute a request.

## Distributed Query Processor (DQP)

- Distributed Query Processor (DQP) evaluate every data request and find an optimal execution strategy, maps data request into ordered sequence of operations on local databases.
- Must consider fragmentation, replication, and allocation schemas. DQP has to decide:
  - which fragment to access;
  - which copy of a fragment to use;
  - which location to use.

## Distributed Query Processor (DQP)

- DQP produces execution strategy optimized with respect to some cost function.
- Typically, costs associated with a distributed request include:
  - I/O cost;
  - CPU cost;
  - communication cost.

## DBMS Transparency

- DBMS transparency hides the knowledge that the local DBMSs may be different, and is therefore only applicable to heterogeneous DDBMSs. It is one of the most difficult transparencies to provide as a generalization.

## Date’s Twelve Rules for A DDBMS

A distributed system looks exactly like a non-distributed system to the user!

1. Local autonomy
2. No reliance on a central site
3. Continuous operation
4. Location independence
5. Fragmentation independence
6. Replication independence
7. Distributed query independence
8. Distributed transaction processing
9. Hardware independence
10. Operating system independence
11. Network independence
12. Database independence

$\left.\begin{array}{l}9\\10\\11\\12\end{array}\right\}$ Last four are ideals

## Relative Advantage of Centralized and Distributed Database

- A Centralized Database
  - Easier to design and implement.
  - Cheaper to operate
- A Distributed Database
  - allows a user convenient and transparent access to data which is not stored at the site, while allowing each site control over its own local data.
  - more reliable than a centralized system because if one site fails, the database can continue functioning, but if the centralized system fails, the database can no longer continue with its normal operation.
  - allows parallel execution of queries and possibly splitting one query into many parts to increase throughput.

## End