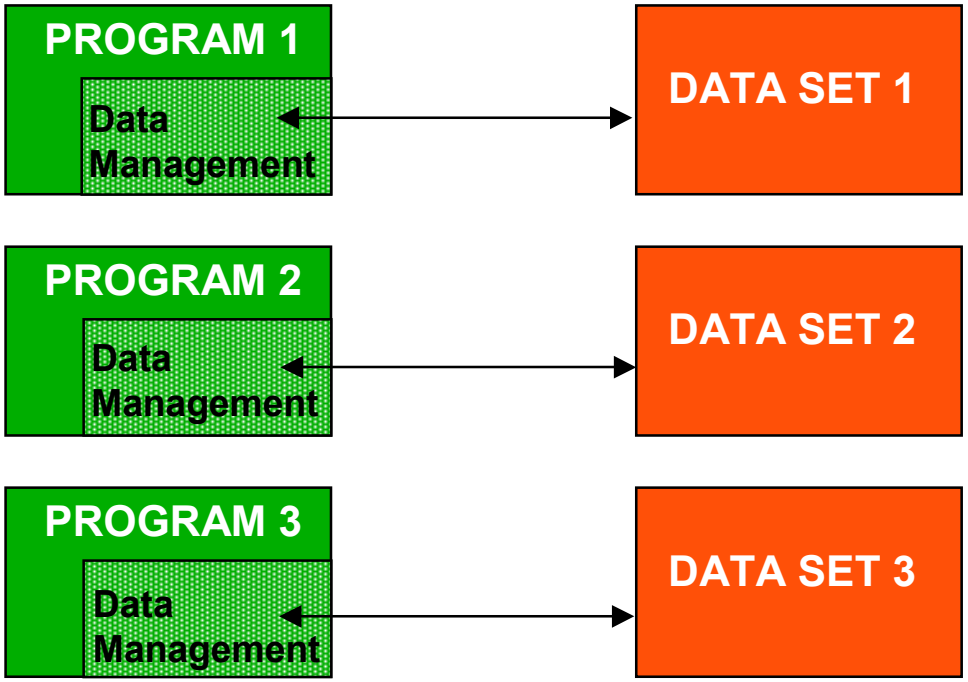
What is "Data"?

- ANSI definition:
  - Data
    1. A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or by automatic means.
    2. Any representation such as characters or analog quantities to which meaning is or might be assigned. Generally, we perform operations on data or data items to supply some information about an entity.
- Volatile vs. persistent data
  - Our concern is primarily with persistent data

Early Data Management - Ancient History

- Data are not stored on disk
- Programmer defines both logical data structure and physical structure (storage structure, access methods, I/O modes, etc)
- One data set per program. High data redundancy.

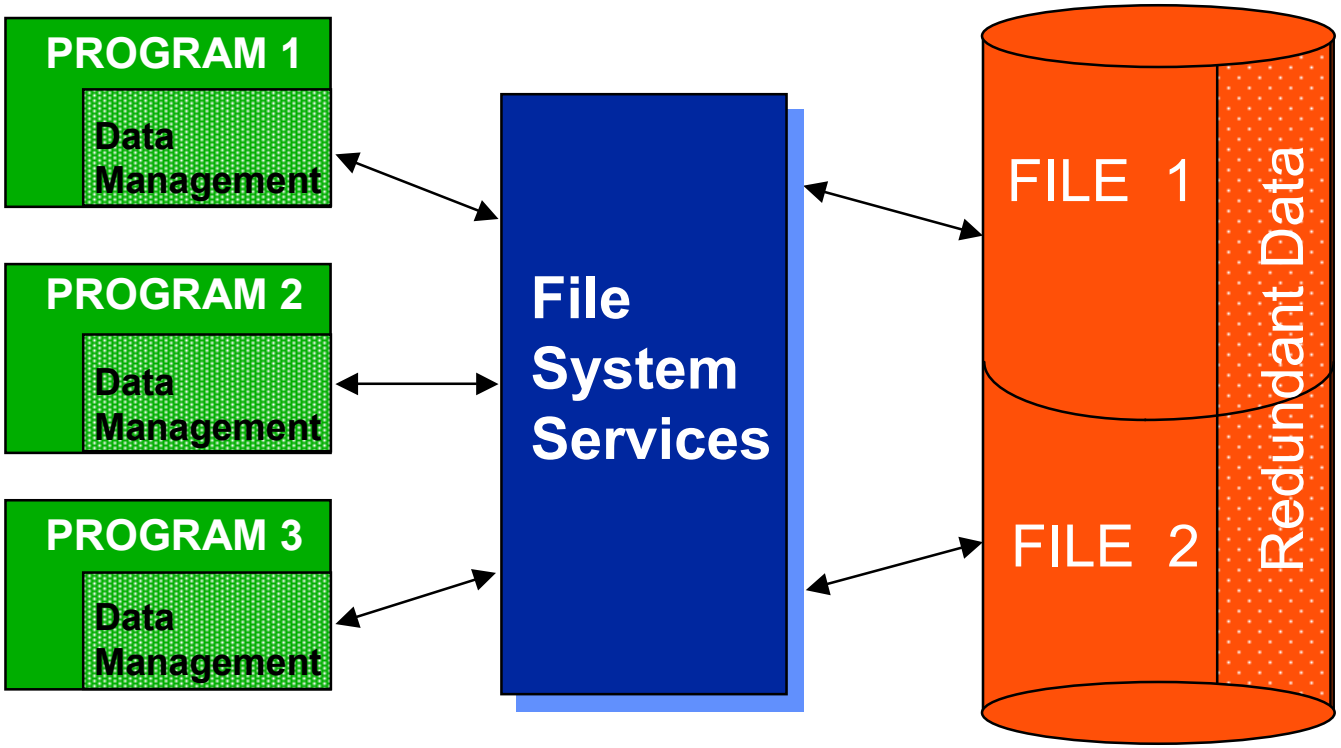


Problems

- There is no persistence.
  - All data is transient and disappears when the program terminates.
- Random access memory (RAM) is expensive and limited
  - All data may not fit available memory
- Programmer productivity low
  - The programmer has to do a lot of tedious work.

File Processing - Recent (and Current) History

- Data are stored in files with interface between programs and files.
- Various access methods exist (e.g., sequential, indexed, random)
- One file corresponds to one or several programs.



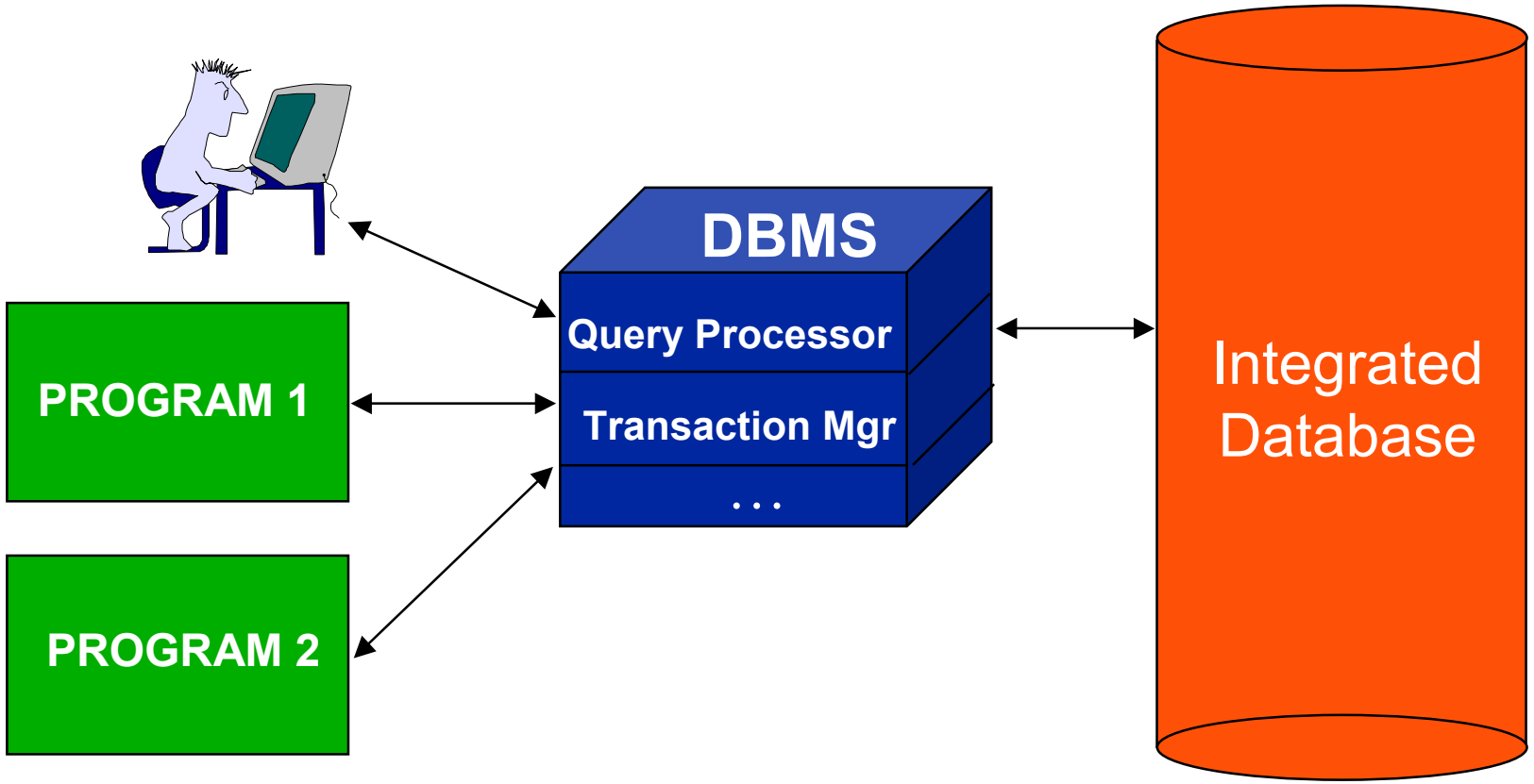
File System Functions

- Mapping between logical files and physical files
  - Logical files: a file viewed by users and programs.
    - Data may be viewed as a collection of bytes or as a collection of records (collection of bytes with a particular structure)
    - Programs manipulate logical files
  - Physical files: a file as it actually exists on a storage device.
    - Data usually viewed as a collection of bytes located at a physical address on the device
    - Operating systems manipulate physical files.
- A set of services and an interface (usually called application independent interface – API)

Problems With File Systems

- Data are highly redundant
  - sharing limited and at the file level
- Data is unstructured
  - “flat” files
- High maintenance costs
  - data dependence; accessing data is difficult
  - ensuring data consistency and controlling access to data
- Sharing granularity is very coarse
- Difficulties in developing new applications

Database Approach



What is a Database?

- A database is an integrated and structured collection of stored operational data used (shared) by application systems of an enterprise

Manufacturing	Product data
University	Student data, courses
Hospital	Patient data, facilities
Bank	Account data

What is a Database?

- A database (DB) is a structured collection of data about the entities that exist in the environment that is being modeled.
- The structure of the database is determined by the abstract data model that is used.
- A database management system (DBMS) is the generalized tool that facilitates the management of and access to the database.

Data Model

- Formalism that defines what the structure of the data are
  - within a file
  - between files
- File systems can at best specify data organization within one file
- Alternatives for business data
  - Hierarchical; network
  - Relational
  - Object-oriented
- This structure is usually called the schema
  - A schema can have many instances

Example Relation Instances

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

WORKS

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ

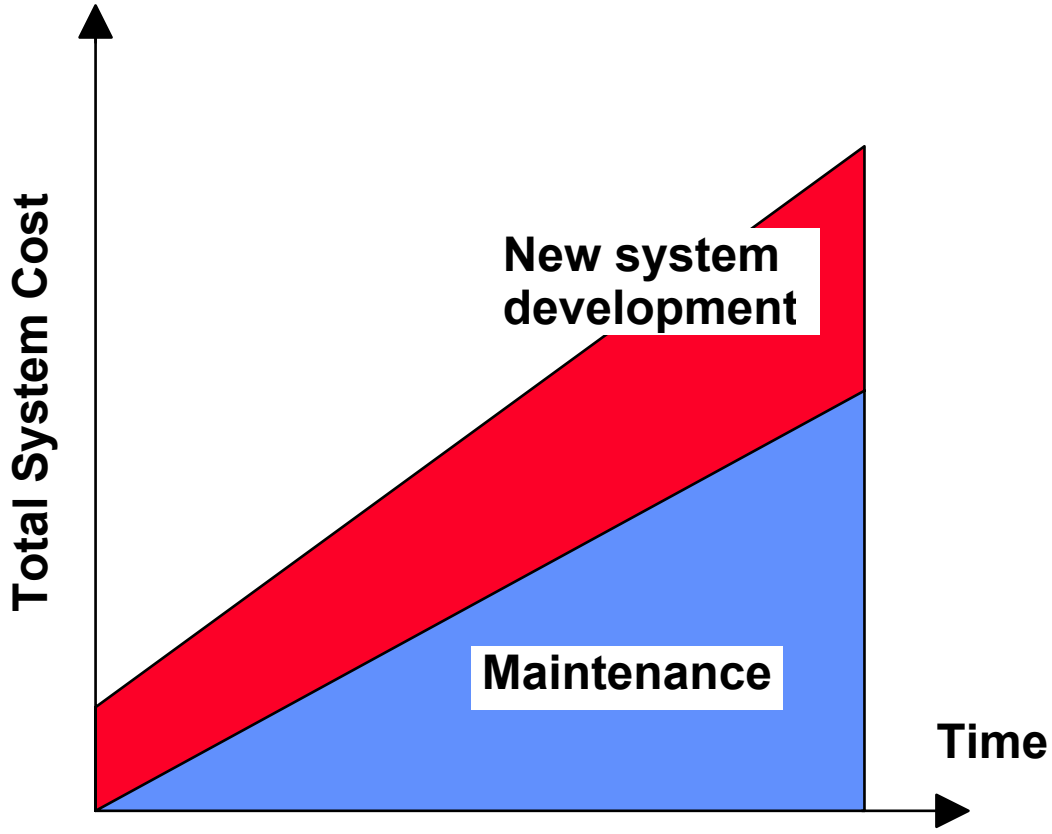
PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Why Database Technology

- Data constitute an organizational asset ⇨ Integrated control
  - Reduction of redundancy
  - Avoidance of inconsistency
  - Sharability
  - Standards
  - Improved security
  - Data integrity
- Programmer productivity ⇨ Data Independence

Why Database Technology

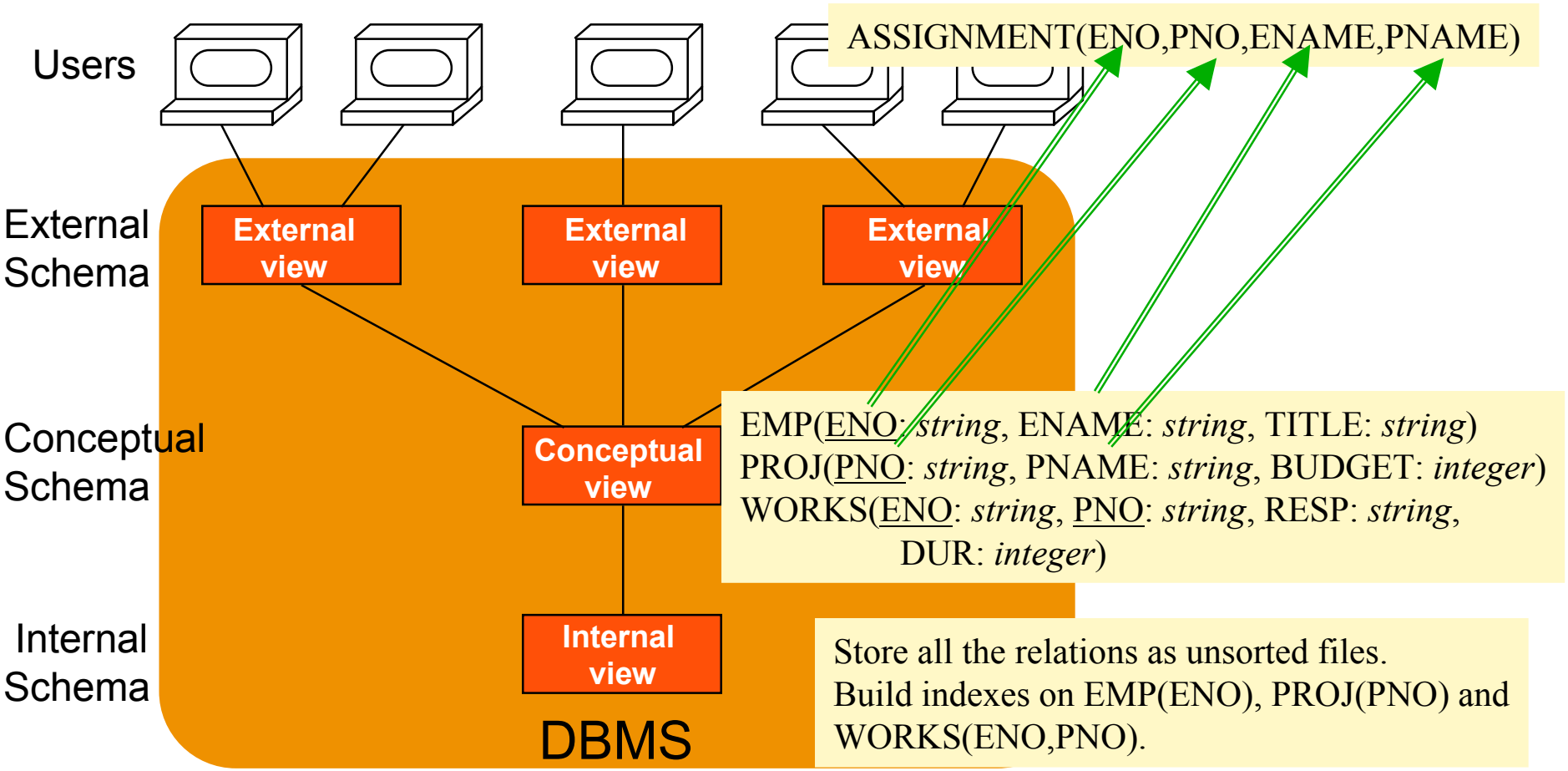
- Programmer productivity
  - High data independence



Data Independence

- Invisibility (transparency) of the details of conceptual organization, storage structure and access strategy to the users
  - Logical
    - transparency of the conceptual organization
    - transparency of logical access strategy
  - Physical
    - transparency of the physical storage organization
    - transparency of physical access paths

ANSI/SPARC Architecture



Database Functionality

- Integrated schema
  - Users have uniform view of data
  - They see things only as relations (tables) in the relational model
- Declarative integrity and consistency enforcement
  - $24000 \leq \text{Salary} \leq 250000$
  - No employee can have a salary greater than his/her manager.
  - User specifies and system enforces.
- Individualized views
  - Restrictions to certain relations
  - Reorganization of relations for certain classes of users

Database Functionality (cont'd)

- Declarative access
  - Query language - SQL
    - Find the names of all electrical engineers.

```
SELECT   ENAME
FROM     EMP
WHERE    TITLE = "Elect. Eng."
```

    - Find the names of all employees who have worked on a project as a manager for more than 12 months.

```
SELECT   EMP.ENAME
FROM     EMP,ASG
WHERE    RESP = "Manager"
AND      DUR > 12
AND      EMP.ENO = ASG.ENO
```

- System determined execution
  - Query processor & optimizer

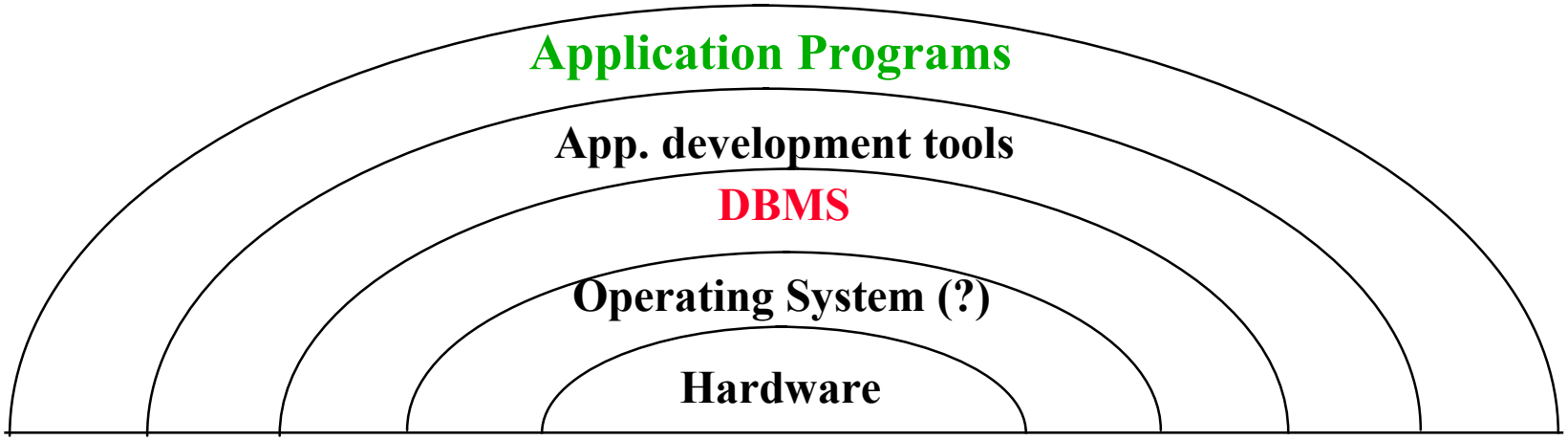
Database Functionality (cont'd)

- Transactions
  - Execute user requests as atomic units
  - May contain one query or multiple queries
  - Provide
    - Concurrency transparency
      - Multiple users may access the database, but they each see the database as their own personal data
      - Concurrency control
    - Failure transparency
      - Even when system failure occurs, database consistency is not violated
      - Logging and recovery

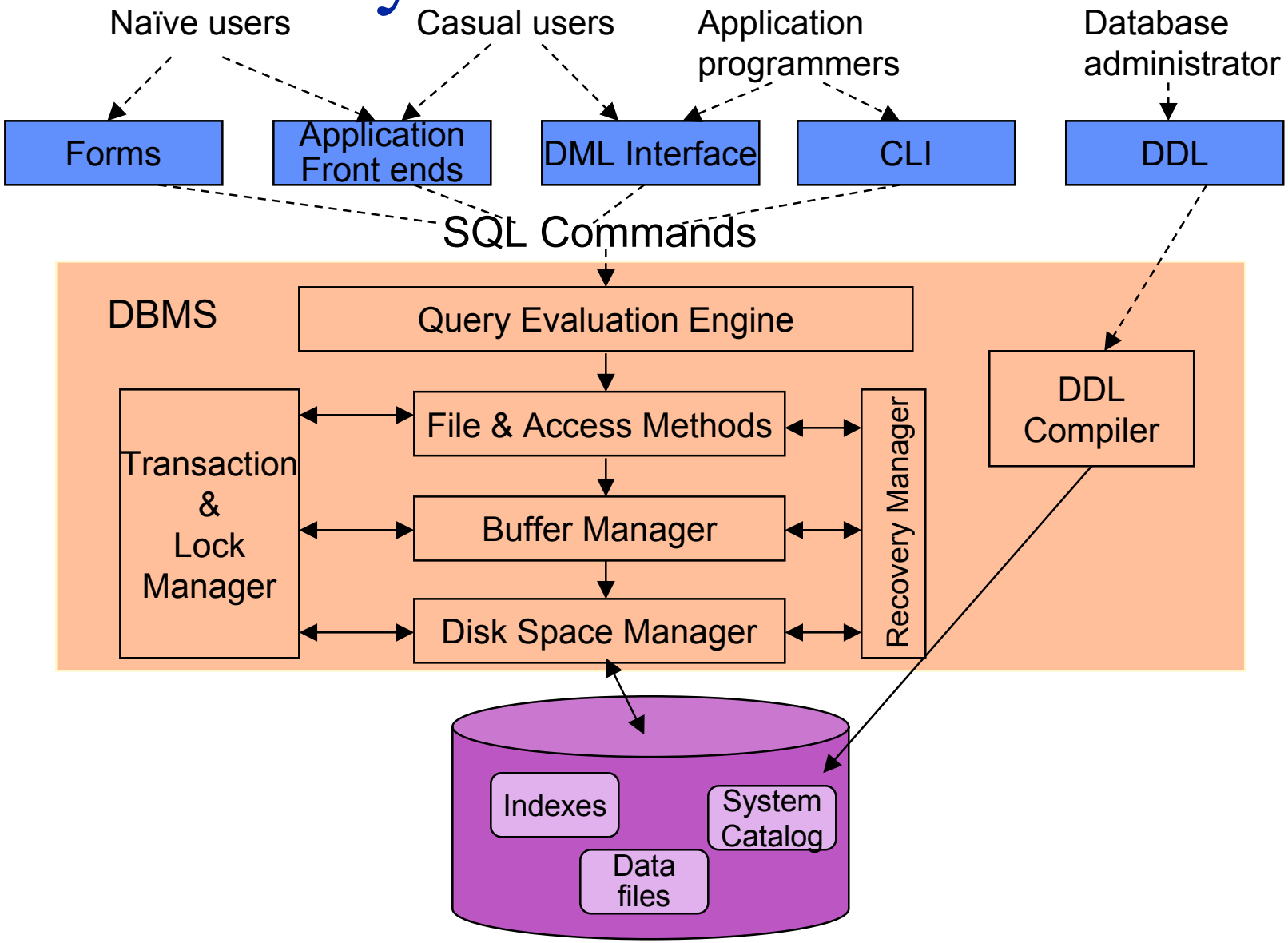
Database Functionality (cont'd)

- Transaction Properties
  - Atomicity
    - All-or-nothing property
  - Consistency
    - Each transaction is correct and does not violate database consistency
  - Isolation
    - Concurrent transactions do not interfere with each other
  - Durability
    - Once the transaction completes its work (commits), its effects are guaranteed to be reflected in the database regardless of what may occur

Place in a Computer System



System Structure



Database Users

- End user
  - Naïve or casual user
  - Accesses database either through forms or through application front-ends
  - More sophisticated ones generate ad hoc queries using DML
- Application programmer/developer
  - Designs and implements applications that access the database (some may be used by end users)
- Database administrator (DBA)
  - Defines and manages the conceptual schema
  - Defines application and user views
  - Monitors and tunes DBMS performance (defines/modifies internal schema)
  - Loads and reformats the database
  - Responsible for security and reliability

DBMS Languages

- Data Definition Language (DDL)
  - Defines conceptual schema, external schema, and internal schema, as well as mappings between them
  - Language used for each level may be different
  - The definitions and the generated information is stored in system catalog
- Data Manipulation Language (DML)
  - Can be
    - embedded query language in a host language
    - "stand-alone" query language
  - Can be
    - Procedural: specify where and how (navigational)
    - Declarative: specify what

Brief History of Databases

- 1960s:
  - Early 1960s: Charles Bachmann developed first DBMS at Honeywell (IDS)
    - Network model where data relationships are represented as a graph.
  - Late 1960s: First commercially successful DBMS developed at IBM (IMS)
    - Hierarchical model where data relationships are represented as a tree
    - Still in use today (SABRE reservations; Travelocity)
  - Late 1960s: Conference On DAta Systems Languages (CODASYL) model defined. This is the network model, but more standardized.

Brief History of Databases

- 1970s:
  - 1970: Ted Codd defined the relational data model at IBM San Jose Laboratory (now IBM Almaden)
  - Two major projects start (both were operational in late 1970s)
    - INGRES at University of California, Berkeley
      - Became commercial INGRES, followed-up by POSTGRES which was incorporated into Informix
    - System R at IBM San Jose Laboratory
      - Became DB2
  - 1976: Peter Chen defined the Entity-Relationship (ER) model

Brief History of Databases

- 1980s
  - Maturation of relational database technology
  - SQL standardization (mid-to-late 1980s) through ISO
  - The real growth period
- 1990s
  - Continued expansion of relational technology and improvement of performance
  - Distribution becomes a reality
  - New data models: object-oriented, deductive
  - Late 1990s: incorporation of object-orientation in relational DBMSs $\rightarrow$ Object-Relational DBMSs
  - New application areas: Data warehousing and OLAP, Web and Internet, interest in text and multimedia

Relational Databases

- Basic concepts
  - Data model: organize data as tables
  - A relational database is a set of tables
- Advantages
  - Simple concepts
  - Solid mathematical foundation
    - set theory
  - Powerful query languages
  - Efficient query optimization strategies
  - Design theory
- Industry standard
  - Relational model
  - SQL language

Relational Model

- Relation
  - A relation R with attributes $A=\{A_1, A_2, \ldots, A_n\}$ defined over n domains $D=\{D_1, D_2, ..., D_n\}$ (not necessarily distinct) with values $\{Dom_1, Dom_2, ..., Dom_n\}$ is a finite, time varying set of n-tuples $<d_1, d_2, ..., d_n>$ such that $d_1 \in Dom_1$, $d_2 \in Dom_2$, ..., $d_n \in Dom_n$ and $A_1 \in D_1$, $A_2 \in D_2$, ..., $A_n \in D_n$.
  - Notation: $R(A_1, A_2, \ldots, A_n)$ or $R(A_1: D_1, A_2: D_2, \ldots, A_n: D_n)$
  - Alternatively, given R as defined above, an instance of it at a given time is a set of n-tuples:

    $\{< A_1: d_1, A_2: d_2, \ldots, A_n: d_n> \mid d_1 \in Dom_1, \ d_2 \in Dom_2, ..., d_n \in Dom_n\}$
- Tabular structure of data where
  - R is the table heading
  - attributes are table columns
  - each tuple is a row

Relation Schemes and Instances

- Relational scheme
  - A relation scheme is the definition; i.e., a set of attributes
  - A relational database scheme is a set of relation schemes:
    - i.e., a set of sets of attributes
- Relation instance (simply *relation*)
  - An relation is an instance of a relation scheme
  - a relation **r** over a relation scheme $R = \{A_1, ..., A_n\}$ is a subset of the Cartesian product of the domains of all attributes, i.e.,

$$\mathbf{r} \subseteq Dom_1 \times Dom_2 \times \ldots \times Dom_n$$

Domains

- A domain is a *type* in the programming language sense
  - Name: String
  - Salary: Real
- Domain values is a set of acceptable values for a variable of a given type.
  - Name: CdnNames = {…},
  - Salary: ProfSalary = {45,000 - 150,000}
  - Simple/Composite domains
    - Address = Street name+street number+city+province+ postal code
- Domain compatibility
  - Binary operations (e.g., comparison to one another, addition, etc) can be performed on them.
- Full support for domains is not provided in many current relational DBMSs

Relation Schemes

EMP(ENO, ENAME, TITLE)

PROJ (PNO, PNAME, BUDGET)

WORKS(ENO,PNO, RESP, DUR)

PAY(TITLE, SALARY)

- Underlined attributes are relation keys (tuple identifiers).
- Tabular form

EMP

ENO	ENAME	TITLE

PROJ

PNO	PNAME	BUDGET

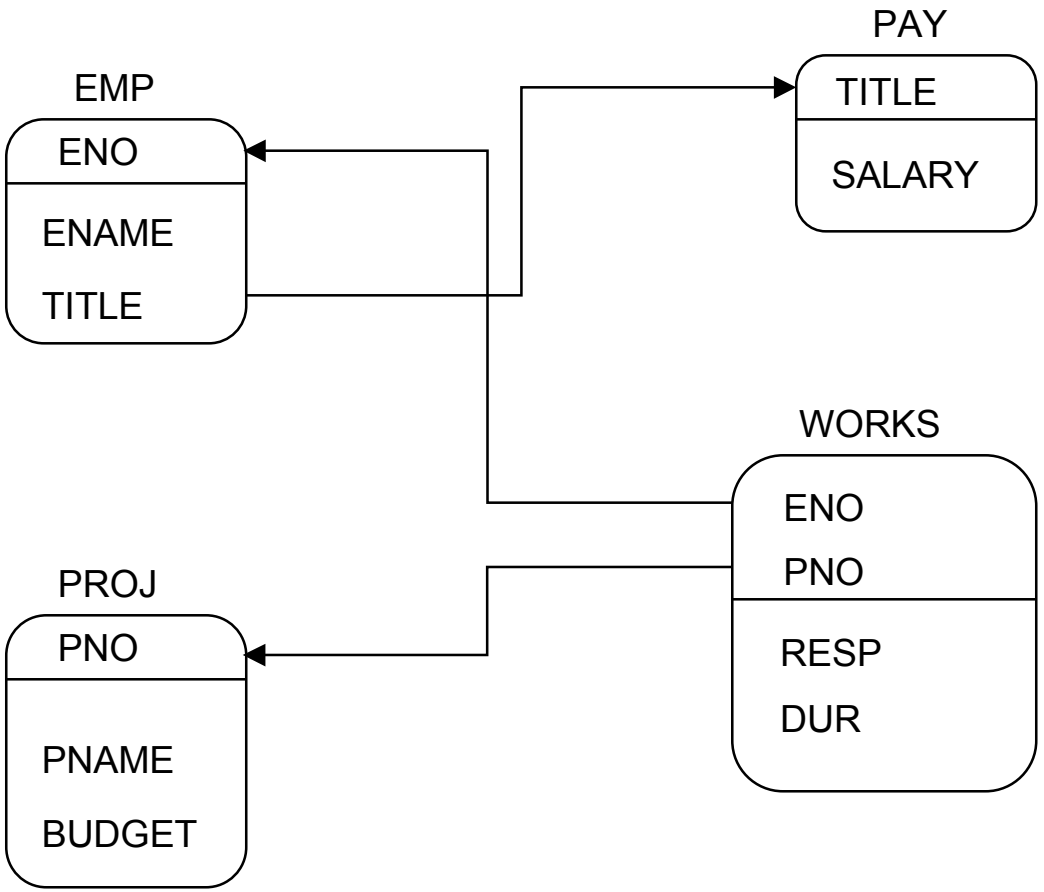
WORKS

ENO	PNO	RESP	DUR

PAY

TITLE	SALARY

Different Representation



Example Relation Instances

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

WORKS

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

Properties

- Based on finite set theory
  - No ordering among attributes
    - Sometimes we prefer to refer to them by their relative order
  - No ordering among tuples
    - Query results may be ordered, but two differently ordered relation instances are equivalent
  - No duplicate tuples allowed
    - Commercial systems allow duplicates (so bag semantics)
  - Value-oriented: tuples are identified by the attributes values
- All attribute values are atomic
  - no tuples, or sets, or other structures
- Degree or arity
  - number of attributes
- Cardinality
  - number of tuples

Integrity Constraints

- Key Constraints
  - Key: a set of attributes that uniquely identifies tuples
  - Candidate key: a minimum set of attributes that form a key
  - Superkey: A set of one or more attributes, which, taken collectively, allow us to identify uniquely a tuple in a relation.
  - Primary key: a designated candidate key
- Data Constraints
  - Functional dependency, multivalued dependency, …
  - Check constraints
- Others
  - Null constraints
  - Referential constraints

Views

- Views can be defined
  - on single relations PROJECT(PNO, PNAME)
  - on multiple relations SAL(ENO,TITLE,SALARY)
- Relations from which they are derived are called base relations
- View relations can be
  - virtual; never physically created
    - updates to views is a problem
  - materialized: physical relations exist
    - propagation of base table updates to materialized view tables

View Updates

- Views that are derived from multiple tables may cause problems

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

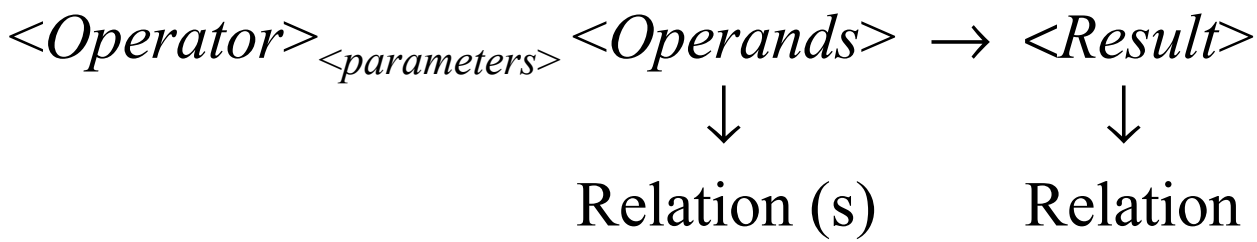
SAL

ENO	TITLE	SALARY
E1	Elect. Eng.	55000
E2	Syst. Anal.	70000
E3	Mech. Eng.	45000
E4	Programmer	60000
E5	Syst. Anal.	70000
E6	Elect. Eng.	55000
E7	Mech. Eng.	45000
E8	Syst. Anal.	70000

How do you delete a tuple from SAL?

Relational Algebra

Form



Relational Algebra Operators

- Fundamental
  - union
  - set difference
  - selection
  - projection
  - Cartesian product
- Additional
  - rename
  - intersection
  - join
  - quotient (division)
- Union compatibility
  - same degree
  - corresponding attributes defined over the same domain

Union

- Similar to set union
- General form

$$R \cup S = \{t \mid t \in R \text{ or } t \in S\}$$

where R, S are relations, t is a tuple variable

  - Result contains tuples that are in R or in S, but not both (duplicates removed)
  - R, S should be union-compatible

Set Difference

- General Form

$$R - S = \{t \mid t \in R \text{ and } t \notin S\}$$

where R and S are relations, t is a tuple variable

  - Result contains all tuples that are in R, but not in S.
  - $R - S \neq S - R$
  - R, S union-compatible

Selection

- Produces a horizontal subset of the operand relation
- General form

$$\sigma_F(R)=\{t \mid t \in R \text{ and } F(t) \text{ is true}\}$$

where

  - R is a relation, t is a tuple variable
  - F is a formula consisting of
    - operands that are constants or attributes
    - arithmetic comparison operators

      $<, >, =, \neq, \leq, \geq$

    - logical operators

      $\wedge, \vee, \neg$

Selection Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

$\sigma_{\text{TITLE='Elect. Eng.'}}(\text{EMP})$

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E6	L. Chu	Elect. Eng.

Projection

- Produces a vertical slice of a relation
- General form

$$\Pi_{A_1,\ldots,A_n}(R)=\{t[A_1,\ldots,A_n] \mid t \in R\}$$

where

  - R is a relation, t is a tuple variable
  - $\{A_1,\ldots,A_n\}$ is a subset of the attributes of R over which the projection will be performed

- Note: projection can generate duplicate tuples. Commercial systems (and SQL) allow this and provide
  - Projection with duplicate elimination
  - Projection without duplicate elimination

Projection Example

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

$\Pi_{PNO,BUDGET}(PROJ)$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

Cartesian (Cross) Product

- Given relations
  - R of degree k_1, cardinality n_1
  - S of degree k_2, cardinality n_2
- Cartesian (cross) product:

$$R \times S = \{t\,[A_1,\ldots,A_{k_1}, A_{k_1+1},\ldots,A_{k_1+k_2}] \mid t[A_1,\ldots,A_{k_1}] \in R \text{ and } t[A_{k_1+1},\ldots,A_{k_1+k_2}] \in S\}$$

The result of $R \times S$ is a relation of degree $(k_1 + k_2)$ and consists of all $(n_1 * n_2)$-tuples where each tuple is a concatenation of one tuple of R with one tuple of S.

Cartesian Product Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

EMP × PAY

ENO	ENAME	EMP.TITLE	PAY.TITLE	SALARY
E1	J. Doe	Elect. Eng.	Elect. Eng.	55000
E1	J. Doe	Elect. Eng.	Syst. Anal.	70000
E1	J. Doe	Elect. Eng.	Mech. Eng.	45000
E1	J. Doe	Elect. Eng.	Programmer	60000
E2	M. Smith	Syst. Anal.	Elect. Eng.	55000
E2	M. Smith	Syst. Anal.	Syst. Anal.	70000
E2	M. Smith	Syst. Anal.	Mech. Eng.	45000
E2	M. Smith	Syst. Anal.	Programmer	60000
E3	A. Lee	Mech. Eng.	Elect. Eng.	55000
E3	A. Lee	Mech. Eng.	Syst. Anal.	70000
E3	A. Lee	Mech. Eng.	Mech. Eng.	45000
E3	A. Lee	Mech. Eng.	Programmer	60000
E8	J. Jones	Syst. Anal.	Elect. Eng.	55000
E8	J. Jones	Syst. Anal.	Syst. Anal.	70000
E8	J. Jones	Syst. Anal.	Mech. Eng.	45000
E8	J. Jones	Syst. Anal.	Programmer	60000

Intersection

- Typical set intersection

$$R \cap S = \{t \mid t \in R \text{ and } t \in S\}$$

$$= R - (R - S)$$

- R, S union-compatible

Join

- General form

$$R \bowtie_{F(R.A_i,S.B_j)} S = \{t[A_1,\ldots,A_n,B_1,\ldots,B_m] \mid t[A_1,\ldots,A_n] \in R \text{ and } t[B_1,\ldots,B_m] \in S \text{ and } F(R.A_i, S.B_j) \text{ is true}\}$$

  where

  - R, S are relations, t is a tuple variable
  - $F(R.A_i, S.B_j)$is a formula defined as that of selection.

- A derivative of Cartesian product
  - $R \bowtie_F S = \sigma_F(R \times S)$

Join Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

WORKS

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

$EMP \bowtie_{EMP.ENO>WORKS.ENO} WORKS$

EMP. ENO.	ENAME	TITLE	WORKS. ENO	PNO	RESP	DUR
E2	M. Smith	Elect. Eng.	E1	P1	Manager	12
E3	A. Lee	Syst. Anal.	E1	P1	Manager	12
E3	A. Lee	Syst. Anal.	E2	P1	Analyst	24
E3	A. Lee	Syst. Anal.	E2	P2	Analyst	6
E4	J. Miller	Programmer	E1	P1	Manager	12
E4	J. Miller	Programmer	E2	P1	Analyst	24
E4	J. Miller	Programmer	E2	P2	Analyst	6
E4	J. Miller	Programmer	E3	P3	Consultant	10
E4	J. Miller	Programmer	E3	P4	Engineer	48
E5	B. Casey	Syst. Anal.	E1	P1	Manager	12
E5	B. Casey	Syst. Anal.	E2	P1	Analyst	24
E5	B. Casey	Syst. Anal.	E2	P2	Analyst	6
E5	B. Casey	Syst. Anal.	E3	P3	Consultant	10
E5	B. Casey	Syst. Anal.	E3	P4	Engineer	48
E5	B. Casey	Syst. Anal.	E4	P2	Programmer	18
E6	L. Chu	Elect. Eng.	E1	P1	Manager	12
E6	L. Chu	Elect. Eng.	E2	P1	Analyst	24
E6	L. Chu	Elect. Eng.	E2	P2	Analyst	6
E6	L. Chu	Elect. Eng.	E3	P3	Consultant	10
E6	L. Chu	Elect. Eng.	E3	P4	Engineer	48
E6	L. Chu	Elect. Eng.	E4	P2	Programmer	18
E6	L. Chu	Elect. Eng.	E5	P2	Manager	24
…	…	…	…	…	…	…

Types of Join

- θ-join
  - The formula F uses operator θ
- Equi-join
  - The formula F only contains equality
  - $R \bowtie_{R.A=S.B} S$
- Natural join
  - Equi-join of two relations R and S over an attribute (or attributes) common to both R and S and projecting out one copy of those attributes
  - $R \bowtie S = \Pi_{R \cup S} \sigma_F (R \times S)$

Natural Join Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

EMP⋈PAY

ENO	ENAME	TITLE	SALARY
E1	J. Doe	Elect. Eng.	55000
E2	M. Smith	Analyst	70000
E3	A. Lee	Mech. Eng.	45000
E4	J. Miller	Programmer	60000
E5	B. Casey	Syst. Anal.	70000
E6	L. Chu	Elect. Eng.	55000
E7	R. Davis	Mech. Eng.	45000
E8	J. Jones	Syst. Anal.	70000

Join is over the common attribute TITLE

Types of Join

- Outer-Join
  - Ensures that tuples from one or both relations that do not satisfy the join condition still appear in the final result with other relation's attribute values set to NULL
  - Left outer join ⟕
  - Right outer join ⟖
  - Full outer join ⟗

Division (Quotient)

Given relations

- R of degree k_1 ($R = \{A_1,\ldots,A_{k_1}\}$)
- S of degree k_2 ($S = \{B_1,\ldots,B_{k_2}\}$)

Let $A = \{A_1,\ldots,A_{k_1}\}$ [i.e., $R(A)$]and $B = \{B_1,\ldots,B_{k_2}\}$ [i.e., $S(B)$] and $B \subseteq A$.

Then, $T = R \div S$ gives T of degree k_1-k_2 [i.e., $T(Y)$ where $Y = A$-B] such that for a tuple t to appear in T, the values in t must appear in R in combination with *every tuple* in S.

Division (cont'd)

R

X	Y
x1	y1
x2	y1
x3	y1
x4	y1
x1	y2
x3	y2
x2	y3
x3	y3
x4	y3
x1	y4
x2	y4
x3	y4

S

X
x1
x2
x3

T

Y
y1
y4

$$T_1 \leftarrow \Pi_Y(R)$$
$$T_2 \leftarrow \Pi_Y((S \times T_1) - R)$$
$$T \leftarrow T_1 - T_2$$

Division - Formally

Given relations

- R of degree k_1 ($R = \{A_1,\ldots,A_{k_1}\}$)
- S of degree k_2 ($S = \{B_1,\ldots,B_{k_2}\}$)

Division of R by S (given , $\{B_1,\ldots,B_{k_2}\} \subseteq \{A_1,\ldots,A_{k_1}\}$)

$$R \div S = \{t[\{A_1,\ldots,A_{k_1}\} - \{B_1,\ldots,B_{k_2}\}] \mid \forall u \in S \exists v \in R(v[S] = u \wedge v[R-S] = t)\} =$$

$$\Pi_{R\text{-}S}(R) - \Pi_{R\text{-}S}((\Pi_{R\text{-}S}(R) \times S) - R)$$

$R \div S$ results in a relation of degree $(k_1 - k_2)$ and consists of all $(k_1 - k_2)$-tuples t such that for all k_1-tuples u in S, the tuple tu is in R.

Division Example

EMP

ENO	PNO	PNAME	BUDGET
E1	P1	Instrumentation	150000
E2	P1	Instrumentation	150000
E2	P2	Database Develop.	135000
E3	P1	Instrumentation	150000
E3	P4	Maintenance	310000
E4	P2	Instrumentation	150000
E5	P2	Instrumentation	150000
E6	P4	Maintenance	310000
E7	P3	CAD/CAM	250000
E8	P3	CAD/CAM	250000
E3	P2	Database Develop.	135000
E3	P3	CAD/CAM	250000

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

EMP÷PROJ

ENO
E3

Example Queries

Emp (Eno, Ename, Title, City) (note we added City)
Project(Pno, Pname, Budget, City) (note we added City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- List names of all employees.
  - $\Pi_{Ename}(Emp)$
- List names of all projects together with their budgets.
  - $\Pi_{Pname,Budget}(Project)$

Example Queries

Emp (Eno, Ename, Title, City) (note we added City)
Project(Pno, Pname, Budget, City) (note we added City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all job titles to which at least one employee has been hired.
  - $\Pi_{Title}(Emp)$
- Find the records of all employees who work in Toronto.
  - $\sigma_{City='Toronto'}(Emp)$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all cities where either an employee works or a project exists.
  - $\Pi_{City}(Emp) \cup \Pi_{City}(Project)$
- Find all cities that has a project but no employees who work there.
  - $\Pi_{City}(Project) - \Pi_{City}(Emp)$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find the names of all projects with budgets greater than $225,000.
  - $\Pi_{Pname}(\sigma_{Budget>225000}(Project))$
- List the names and budgets of projects on which employee E1 works.
  - $\Pi_{Pname, Budget}(Project \bowtie (\sigma_{Eno='E1'}(Works)))$
  - $\Pi_{Pname, Budget}(\sigma_{Emp.Eno=Works.Eno}(Project \times \sigma_{Eno='E1'}(Works)))$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find the name of all the employees who work in a city where no project is located.
  - $\Pi_{Ename}(Emp \bowtie (\Pi_{City}(Emp) - \Pi_{City}(Project))$
- Find all the cities that have both employees and projects.
  - $\Pi_{City}(Emp) \cap \Pi_{City}(Project)$
- Find all the employees who work on every project.
  - $\Pi_{Eno,\ Pno}(Works) \div \Pi_{Pno}(Project)$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find the names and budgets of all projects who employ programmers.
  - $\Pi_{Pname,Budget}(Project \bowtie Works \bowtie \sigma_{Title='Programmer'}(Emp))$
- List the names of employees and projects that are co-located.
  - $\Pi_{Ename,\ Pname}(Emp \bowtie Project)$

Relational Calculus

- Instead of specifying how to obtain the result, specify what the result is, i.e., the relationships that is supposed to hold in the result.
- Based on first-order predicate logic.
  - symbol alphabet
    - logic symbols (e.g., $\Rightarrow$, $\neg$)
    - a set of constants
    - a set of variables
    - a set of n-ary predicates
    - a set of n-ary functions
    - parentheses
  - expressions (called well formed formulae (wff)) built from this symbol alphabet.

Types of Relational Calculus

- According to the primitive variable used in specifying the queries.
  - tuple relational calculus
  - domain relational calculus

Tuple Relational Calculus

- The primitive variable is a tuple variable which specifies a tuple of a relation. In other words, it ranges over the tuples of a relation.
- In tuple relational calculus queries are specified as

$$\{t \mid F(t)\}$$

where t is a tuple variable and F is a formula consisting of the atoms and operators. F evaluates to True or False.

t can be qualified for only some attributes: $t[A]$

Tuple Relational Calculus

- The atoms are the following:
  1. Tuple variables
     - If the relation over which the variable ranges is known, the variable may be qualified by the name of the relation as $R.t$ or $R(t)$.
  2. Conditions
     - $s[A]\ \theta\ t[B]$, where s and t are tuple variables and A and B are components of s and t, respectively;

       $\theta \in \{<, >, =, \neq, \leq, \geq\}$.

       Specifies that component A of s stands in relation θ to the B component of t (e.g., s[SALARY] > t[SALARY]).
     - $s[A]\ \theta\ c$, where s, A and θ are as defined above and c is a constant. For example, s[NAME] = "Smith".

Tuple Relational Calculus

- A formula F is composed of
  - atoms
  - Boolean operators $\wedge$, $\vee$, $\neg$
  - existential quantifier $\exists$
  - universal quantifier $\forall$
- Formation rules:
  - Each atom is a formula.
  - If F and G are formulae, so are $F \wedge G$, $F \vee G$, $\neg F$, and $\neg G$.
  - If F is a formula, so is (F).
  - If F is a formula and t is a free variable in F, then $\exists t(F)$ and $\forall t(F)$ are also formulae. These can also be written as $\exists t F(t)$ and $\forall t F(t)$
  - Nothing else is a formula.

Safety of Calculus Expressions

- Problem:
  - the size of $\{t \mid F(t)\}$ must be finite.
  - $\{t \mid \neg t \in R\}$ is not finite
- Safety:
  - A query is safe if, for all databases conforming to the schema, the query result can be computed using only constants appearing in the database or the query itself.
  - Since database is finite, the set of constants appearing in it is finite as well as the constants in the query; therefore, the query result will be finite

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- List names of all employees.

  $\{t[\text{Ename}] \mid t \in \text{Emp}\}$

- List names of all projects together with their budgets.

  $\{<t[\text{Pname}], t[\text{Budget}]> \mid t \in \text{Project}]\}$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all job titles to which at least one employee has been hired.

  $\{t[\text{Title}] \mid t \in \text{Emp})\}$

- Find the records of all employees who work in Toronto.

  $\{t \mid t \in \text{customer} \wedge t[\text{City}] = \text{‘Toronto’}\}$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all cities where either an employee works or a project exists.

$$\{t[\text{City}] \mid t \in \text{Emp} \vee \exists s(s \in \text{Project} \wedge t[\text{City}] = s[\text{City}])\}$$
$$\{t[\text{City}] \mid t \in \text{Project} \vee \exists s(s \in \text{Emp} \wedge t[\text{City}] = s[\text{City}])\}$$

- Find all cities that has a project but no employees who work there.

$$\{t[\text{City}] \mid t \in \text{Project} \wedge \neg\exists s(s \in \text{Emp} \wedge t[\text{City}] = s[\text{City}])\}$$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find the names of all projects with budgets greater than $225,000.

$$\{t[\text{Pname}] \mid t \in \text{Project} \wedge t[\text{Budget}] > 225000\}$$

- List the names and budgets of projects in which employee E1 works.

$$\{<t[\text{Pname}], t[\text{Budget}]> \mid t \in \text{Project} \wedge \exists s(s \in \text{Works} \wedge t[\text{Pname}]=s[\text{Pname}] \wedge s[\text{Eno}]=\text{'E1'})\}$$

Tuple Calculus and Relational Algebra

- THEOREM: Safe relational calculus and algebra are equivalent in terms of their expressive power.
  - This is called relational completeness.
  - This does not mean all useful computations can be performed
    - Aggregation, counting, transitive closure not specified
- Basic Correspondence

Algebra Operation	Calculus Operator
Π	$\exists$
σ	$x\ \theta$ *constant*
$\cup$	$\vee$
$\bowtie$	$\wedge$
-	$\neg$
$\div$	$\forall$

Tuple Calculus and Relational Algebra

Π is like $\exists$ “there exists” ...

$\div$ is like $\forall$ “for all” ...

Expressing $\div$ using basic operators

$$R \div S = \Pi_A(R) - \Pi_A(\Pi_A(R) \bowtie S - R)$$

Similar to

$$\forall x\ F(x) = \neg \quad (\exists x \quad \neg\ F(x)\ )$$

Domain Relational Calculus

- The primitive variable is a domain variable which specifies a component of a tuple.

  ⇨ the range of a domain variable consists of the domains over which the relation is defined.
- Other differences from tuple relational calculus:
  - The atoms are the following :
    - Each domain is an atom.
    - Conditions which can be defined as follows are atoms :
      - $x\ \theta\ y$, where x and y are domain variables or constants;
      - $<x_1, x_2, ..., x_n> \in R$ where R is a relation of degree n and each x_i is a domain variable or constant.
  - Formulae are defined in exactly the same way as in tuple relational calculus, with the exception of using domain variables instead of tuple variables.

Domain Relational Calculus

The queries are specified in the following form :

$$\{<x_1, x_2, ..., x_n> \mid F(x_1, x_2, ..., x_n)\}$$

where F is a formula in which x_1,..., x_n are the free variables.

Domain Relational Calculus

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

List the names and budgets of projects in which employee E1 works.

$$\{<b,c> \mid \exists a,d(<a,b,c,d> \in \text{Project} \wedge \exists e,f,g(<e,a,f,g> \in \text{Works} \wedge e = \text{'E1'}))\}$$

QBE (Query-by-Example) Queries

- Find the names of all employees

Emp	Eno	Ename	Title	City
		P.		

Pay	Title	Salary

Project	Pno	Pname	Budget	City

Works	Eno	Pno	Resp	Dur

QBE Queries

- Find the names of projects with budgets greater than $350,000.

Emp	Eno	Ename	Title	City

Pay	Title	Salary

Project	Pno	Pname	Budget	City
		P.	>350000	

Works	Eno	Pno	Resp	Dur

QBE Queries

- Find the name and cities of all employees who work on a project for more than 20 months.

Emp	Eno	Ename	Title	City
	_x	P.		P.

Pay	Title	Salary

Project	Pno	Pname	Budget	City

Works	Eno	Pno	Resp	Dur
	_x			>20

Introduction to Distributed Database Systems

Outline

- Introduction
- Distributed and Parallel Database Design
- Distributed Data Control
- Distributed Query Processing
- Distributed Transaction Processing
- Data Replication
- Database Integration – Multidatabase Systems
- Parallel Database Systems
- Peer-to-Peer Data Management
- Big Data Processing
- NoSQL, NewSQL and Polystores
- Web Data Management

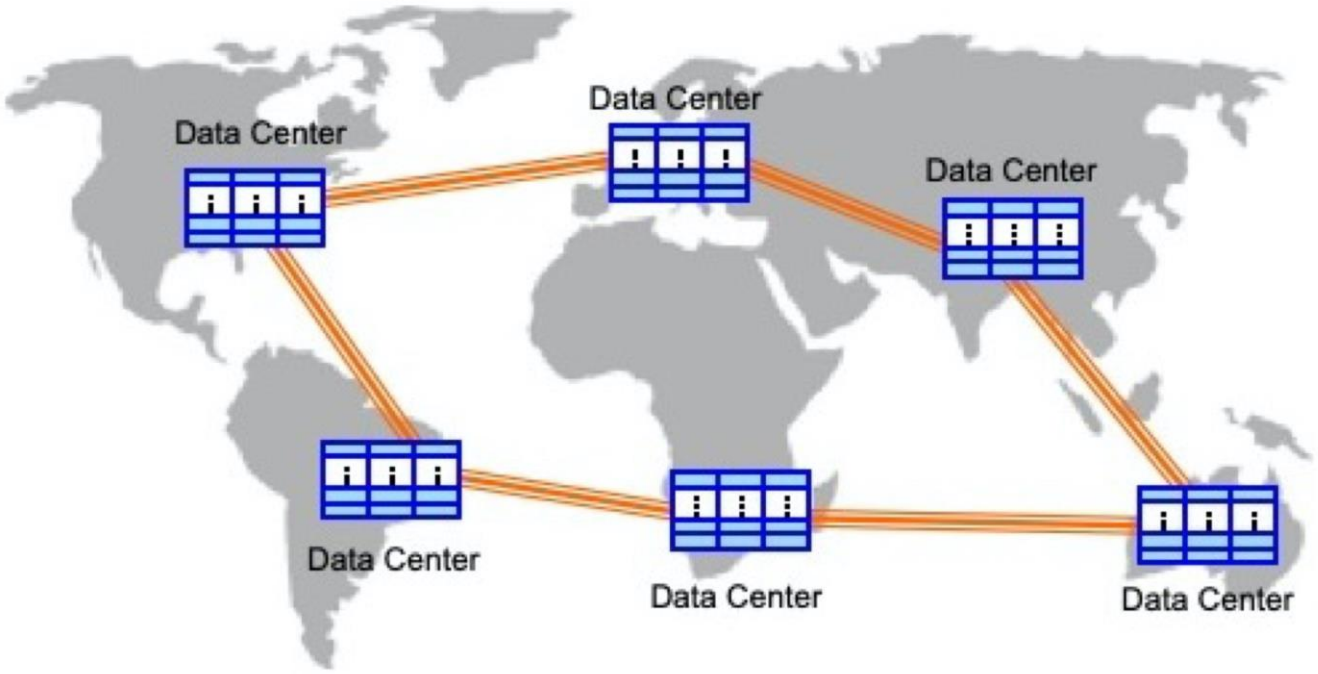
Outline

- Introduction
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture

Distributed Computing

- A number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.
- What is being distributed?
  - Processing logic
  - Function
  - Data
  - Control

Current Distribution – Geographically Distributed Data Centers



What is a Distributed Database System?

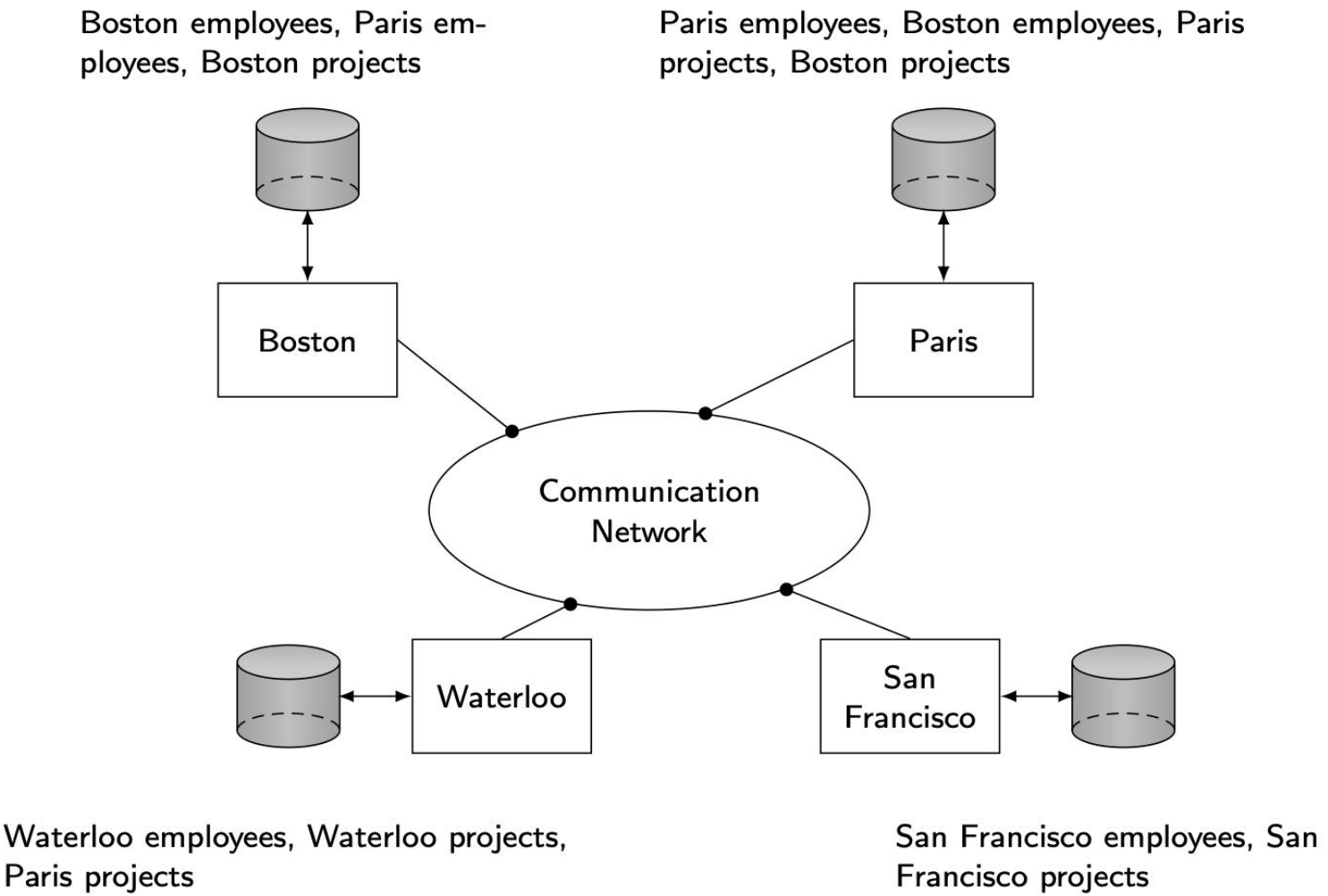
A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network

A distributed database management system (Distributed DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users

What is not a DDBS?

- A timesharing computer system
- A loosely or tightly coupled multiprocessor system
- A database system which resides at one of the nodes of a network of computers - this is a centralized database on a network node

Distributed DBMS Environment



Implicit Assumptions

- Data stored at a number of sites → each site *logically* consists of a single processor
- Processors at different sites are interconnected by a computer network → not a multiprocessor system
  - Parallel database systems
- Distributed database is a database, not a collection of files → data logically related as exhibited in the users' access patterns
  - Relational data model
- Distributed DBMS is a full-fledged DBMS
  - Not remote file system, not a TP system

Important Point

Logically integrated

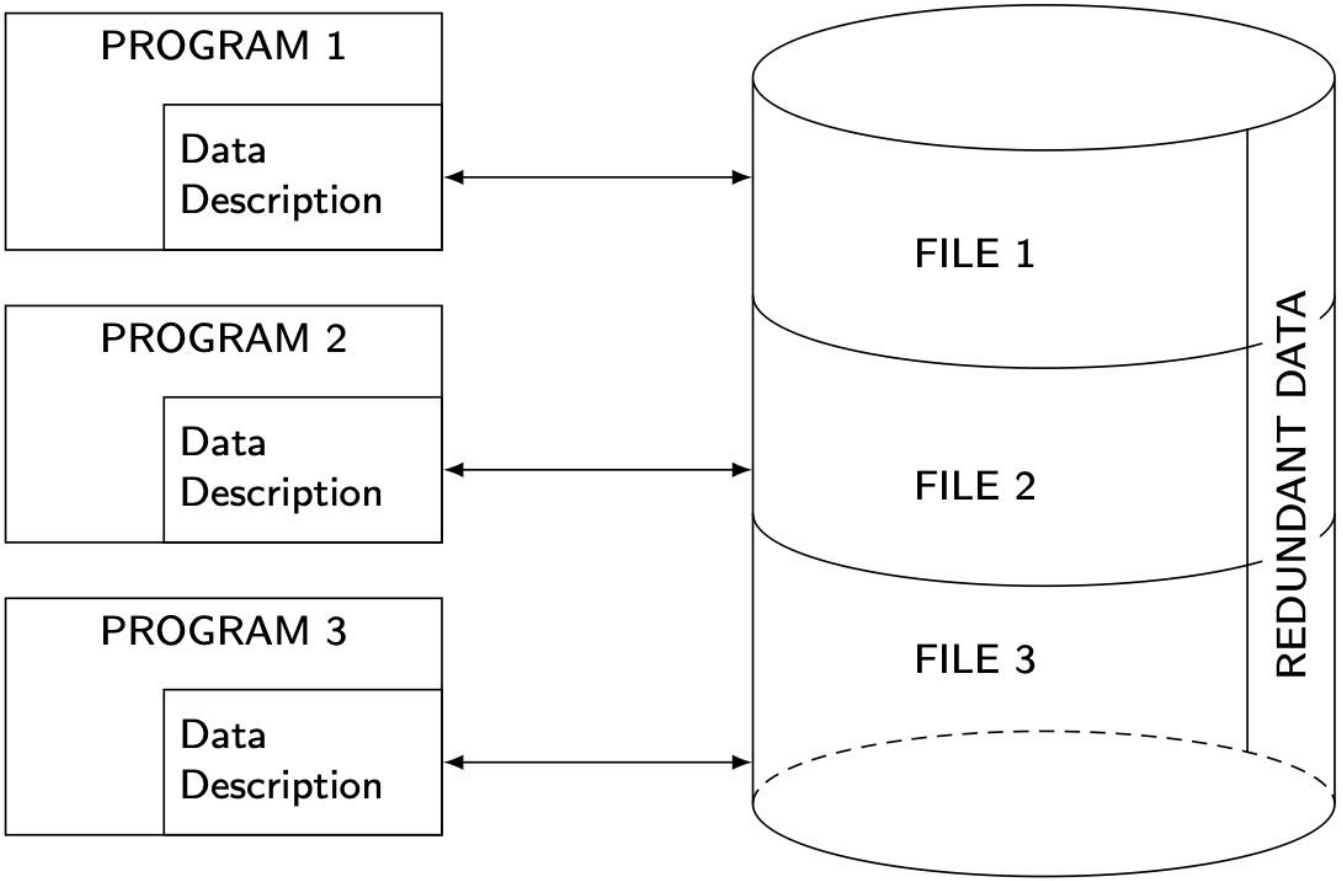
but

Physically distributed

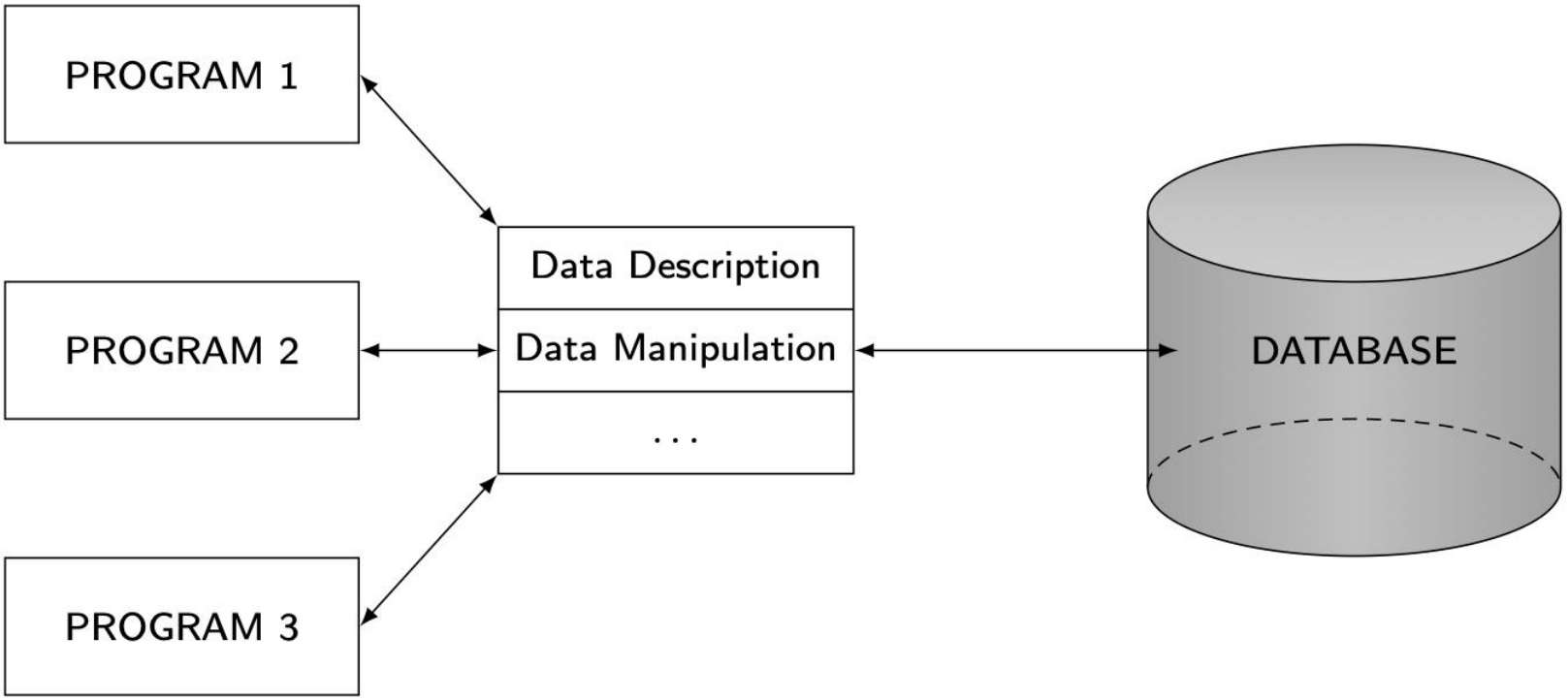
Outline

- Introduction
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture

History – File Systems

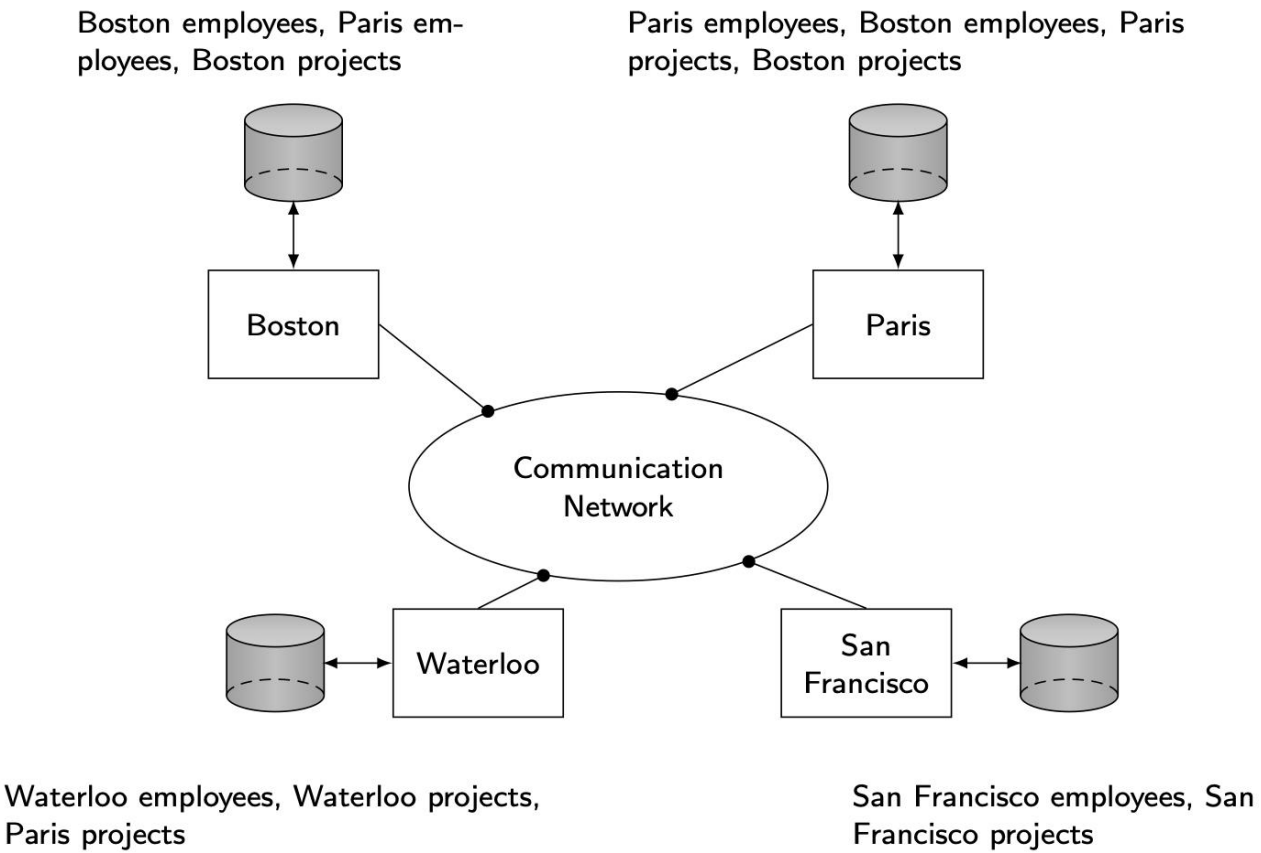


History – Database Management

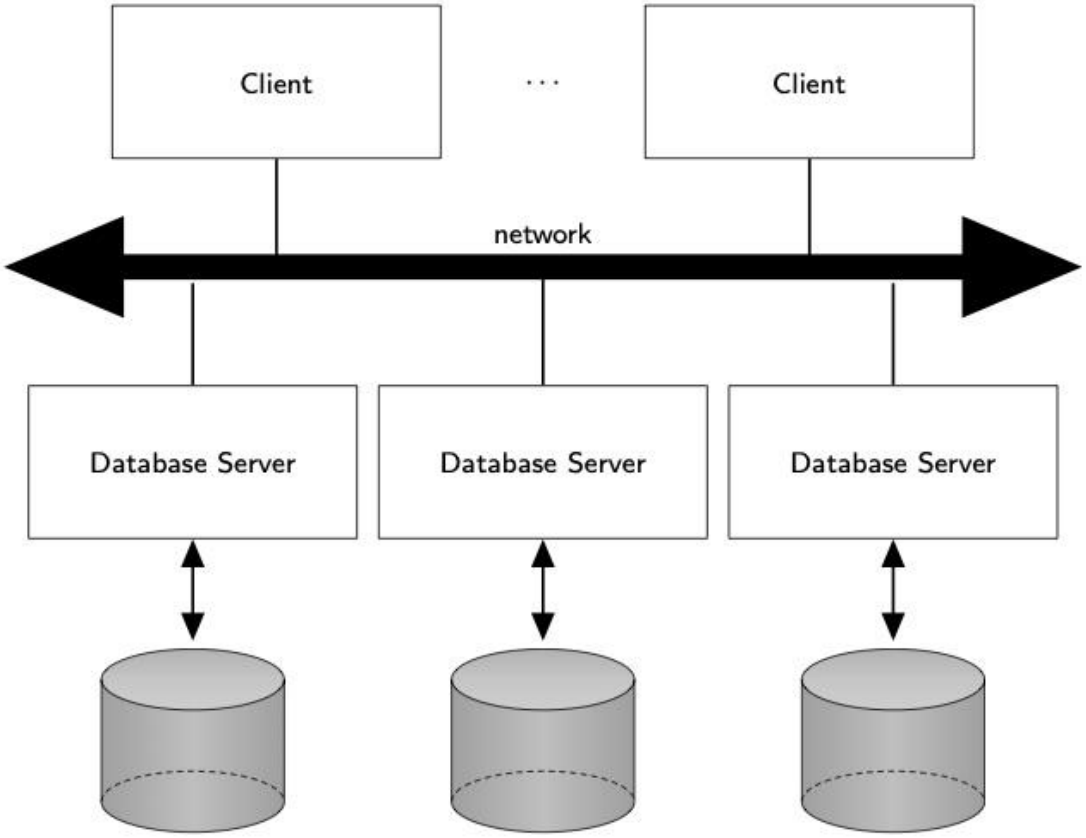


History – Early Distribution

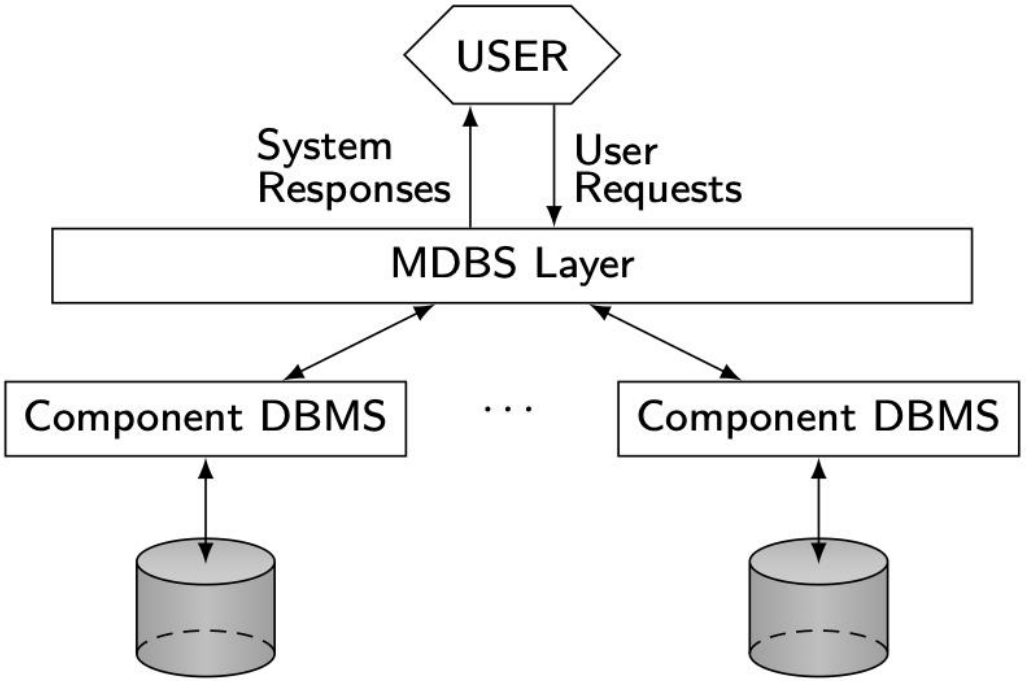
Peer-to-Peer (P2P)



History – Client/Server



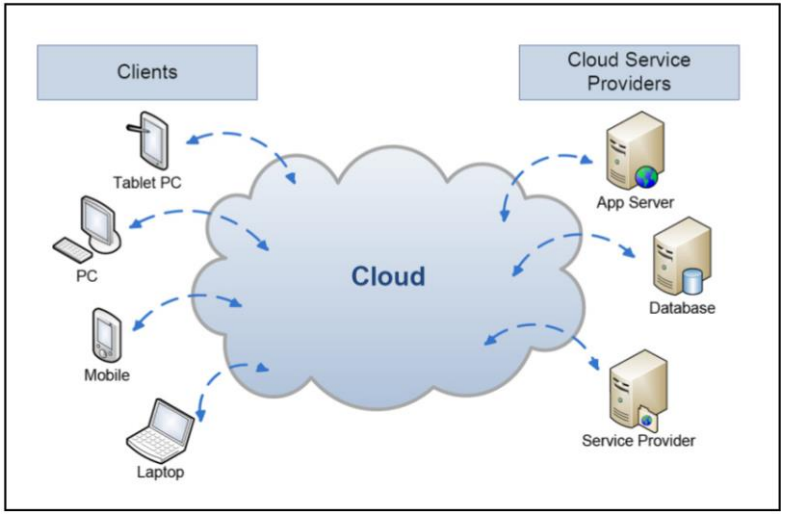
History – Data Integration



History – Cloud Computing

On-demand, reliable services provided over the Internet in a cost-efficient manner

- Cost savings: no need to maintain dedicated compute power
- Elasticity: better adaptivity to changing workload



Data Delivery Alternatives

- Delivery modes
  - Pull-only
  - Push-only
  - Hybrid
- Frequency
  - Periodic
  - Conditional
  - Ad-hoc or irregular
- Communication Methods
  - Unicast
  - One-to-many
- Note: not all combinations make sense

Outline

- Introduction
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture

Distributed DBMS Promises

❶ Transparent management of distributed, fragmented, and replicated data

❷ Improved reliability/availability through distributed transactions

❸ Improved performance

❹ Easier and more economical system expansion

Transparency

- Transparency is the separation of the higher-level semantics of a system from the lower level implementation issues.
- Fundamental issue is to provide

  data independence

  in the distributed environment
  - Network (distribution) transparency
  - Replication transparency
  - Fragmentation transparency
    - horizontal fragmentation: selection
    - vertical fragmentation: projection
    - hybrid

Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

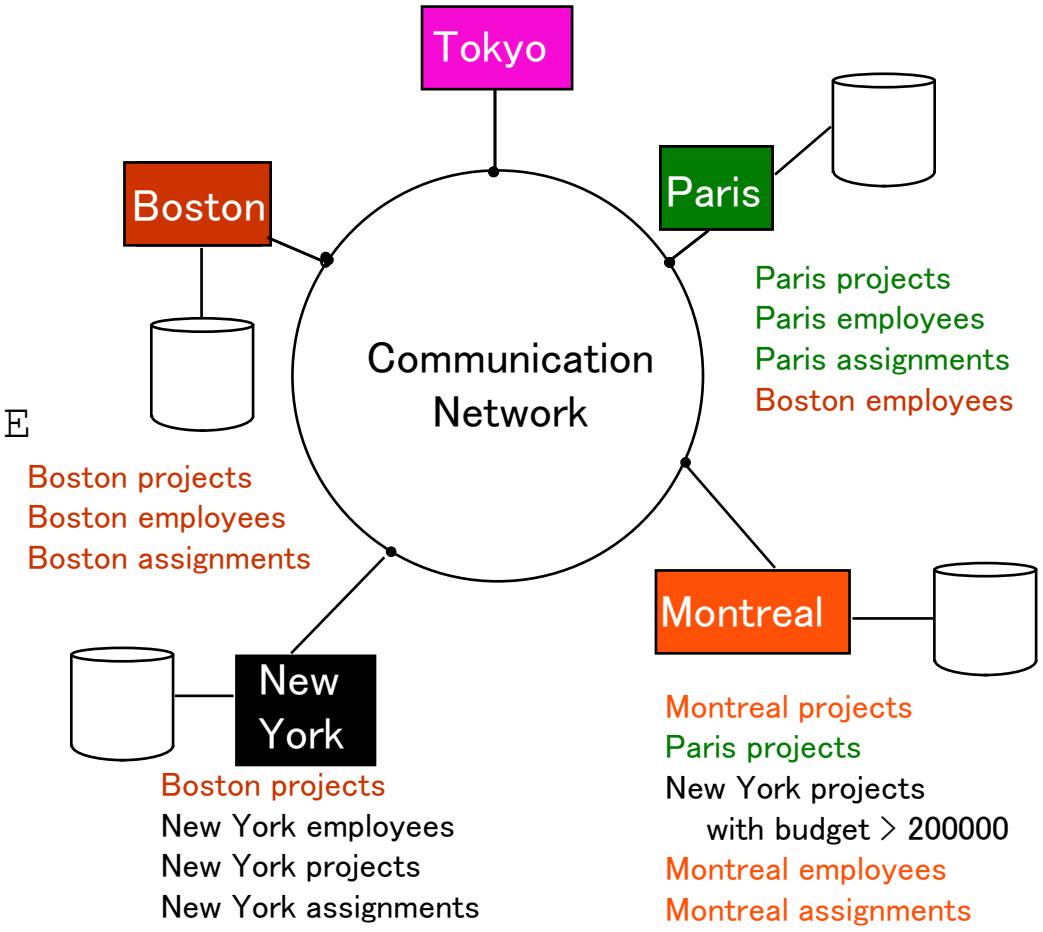
PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

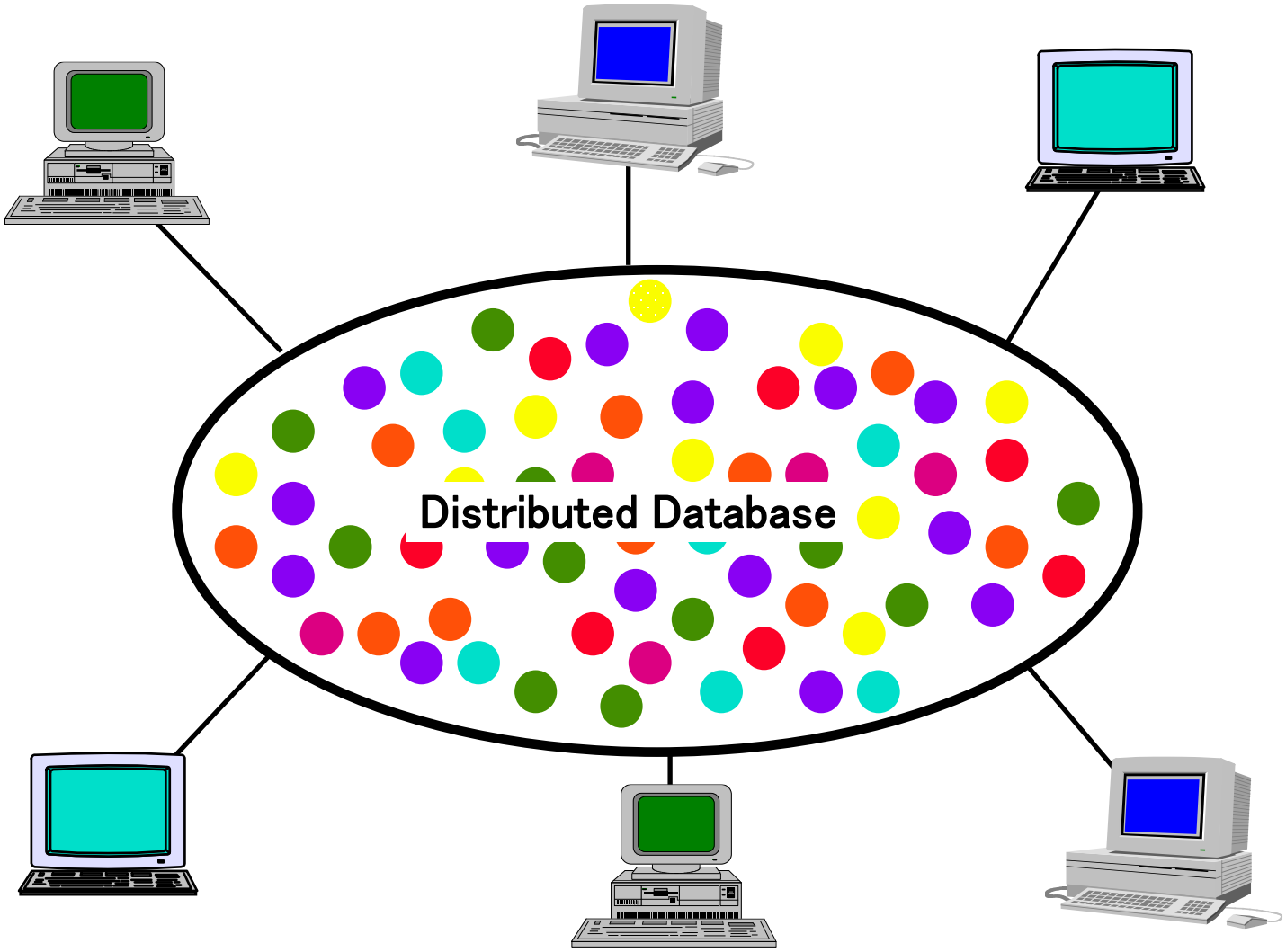
TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Transparent Access

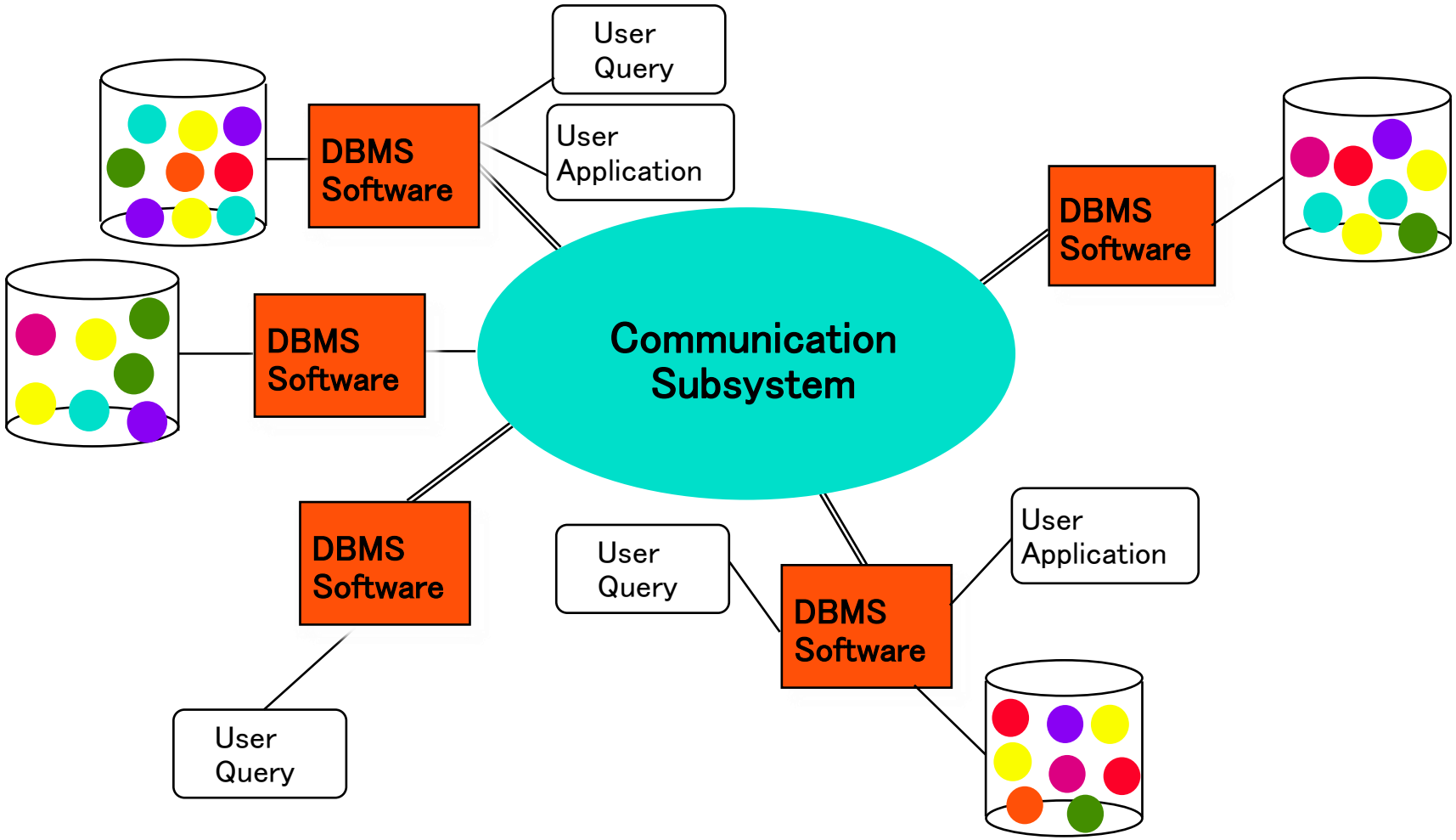
```
SELECT ENAME,SAL
FROM   EMP,ASG,PAY
WHERE  DUR > 12
AND    EMP.ENO = ASG.ENO
AND    PAY.TITLE = EMP.TITLE
```



Distributed Database - User View



Distributed DBMS - Reality



Types of Transparency

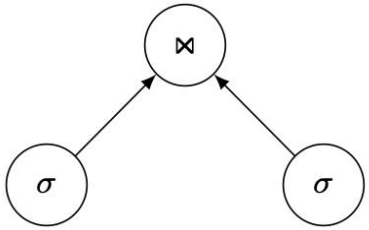
- Data independence
- Network transparency (or distribution transparency)
  - Location transparency
  - Naming transparency
- Fragmentation transparency
- Replication transparency

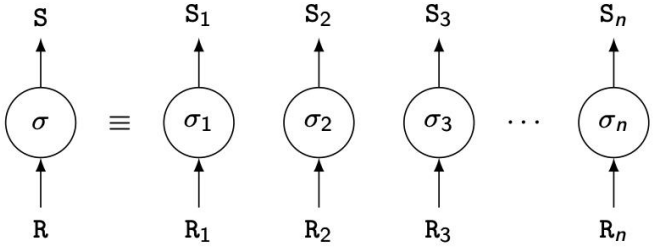
Reliability Through Transactions

- Replicated components and data should make distributed DBMS more reliable.
- Distributed transactions provide
  - Concurrency transparency
  - Failure transparency
- Distributed transaction support requires implementation of
  - Distributed concurrency control protocols
  - Commit protocols
- Data replication
  - Great for read-intensive workloads, problematic for updates
  - Replication protocols

Potentially Improved Performance

- Proximity of data to its points of use
  - Requires some support for fragmentation and replication
- Parallelism in execution
  - Inter-query parallelism
  - Intra-query parallelism





Scalability

- Issue is database scaling and workload scaling
- Adding processing and storage power
- Scale-out: add more servers
  - Scale-up: increase the capacity of one server → has limits

Outline

- Introduction
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture

Distributed DBMS Issues

- Distributed database design
  - How to distribute the database
  - Replicated & non-replicated database distribution
  - A related problem in directory management
- Distributed query processing
  - Convert user transactions to data manipulation instructions
  - Optimization problem
    - min{cost = data transmission + local processing}
  - General formulation is NP-hard

Distributed DBMS Issues

- Distributed concurrency control
  - Synchronization of concurrent accesses
  - Consistency and isolation of transactions' effects
  - Deadlock management
- Reliability
  - How to make the system resilient to failures
  - Atomicity and durability

Distributed DBMS Issues

- Replication
  - Mutual consistency
  - Freshness of copies
  - Eager vs lazy
  - Centralized vs distributed
- Parallel DBMS
  - Objectives: high scalability and performance
  - Not geo-distributed
  - Cluster computing

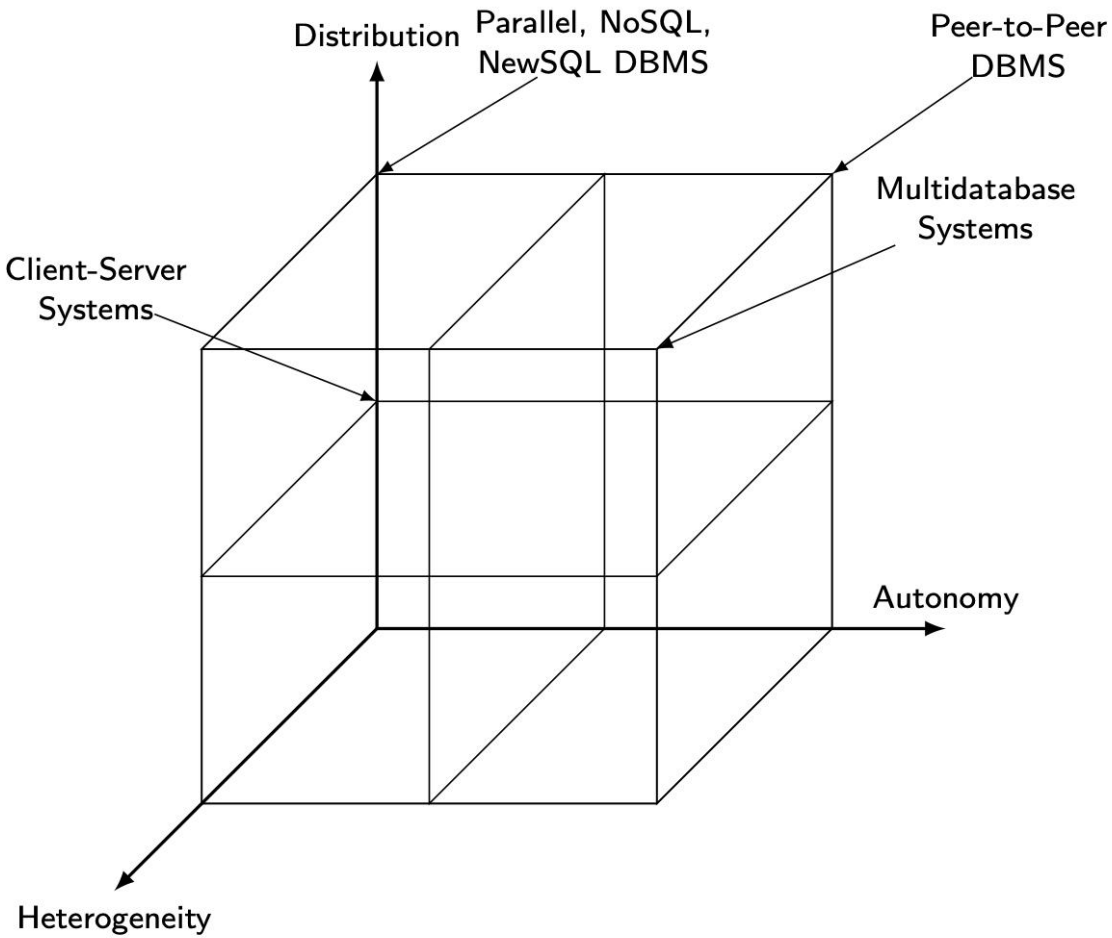
Related Issues

- Alternative distribution approaches
  - Modern P2P
  - World Wide Web (WWW or Web)
- Big data processing
  - 4V: volume, variety, velocity, veracity
  - MapReduce & Spark
  - Stream data
  - Graph analytics
  - NoSQL
  - NewSQL
  - Polystores

Outline

- Introduction
  - What is a distributed DBMS
  - History
  - Distributed DBMS promises
  - Design issues
  - Distributed DBMS architecture

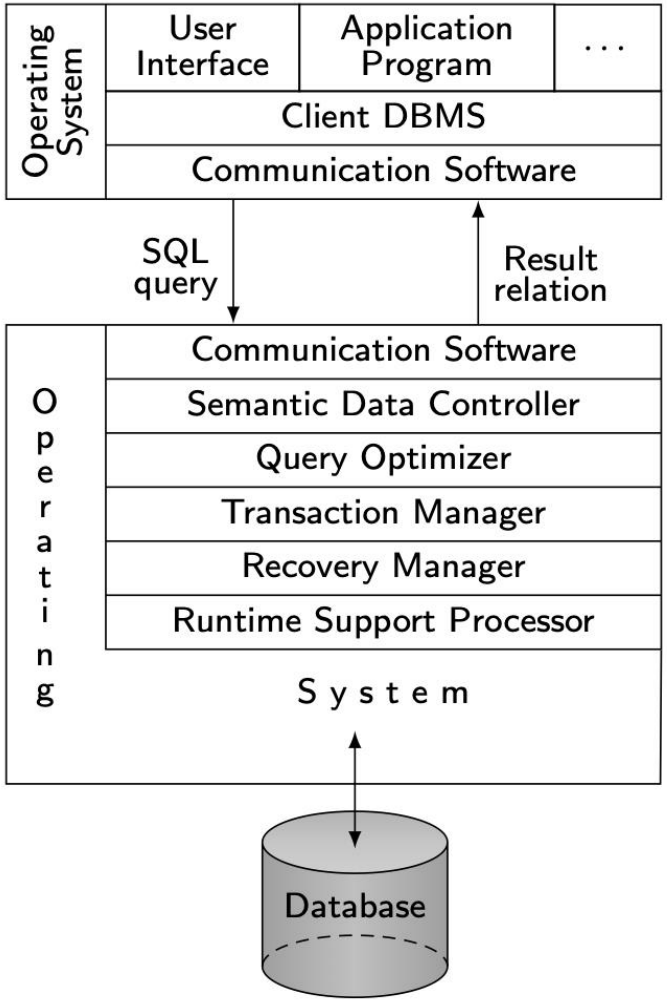
DBMS Implementation Alternatives



Dimensions of the Problem

- Distribution
  - Whether the components of the system are located on the same machine or not
- Heterogeneity
  - Various levels (hardware, communications, operating system)
  - DBMS important one
    - data model, query language,transaction management algorithms
- Autonomy
  - Not well understood and most troublesome
  - Various versions
    - Design autonomy: Ability of a component DBMS to decide on issues related to its own design.
    - Communication autonomy: Ability of a component DBMS to decide whether and how to communicate with other DBMSs.
    - Execution autonomy: Ability of a component DBMS to execute local operations in any manner it wants to.

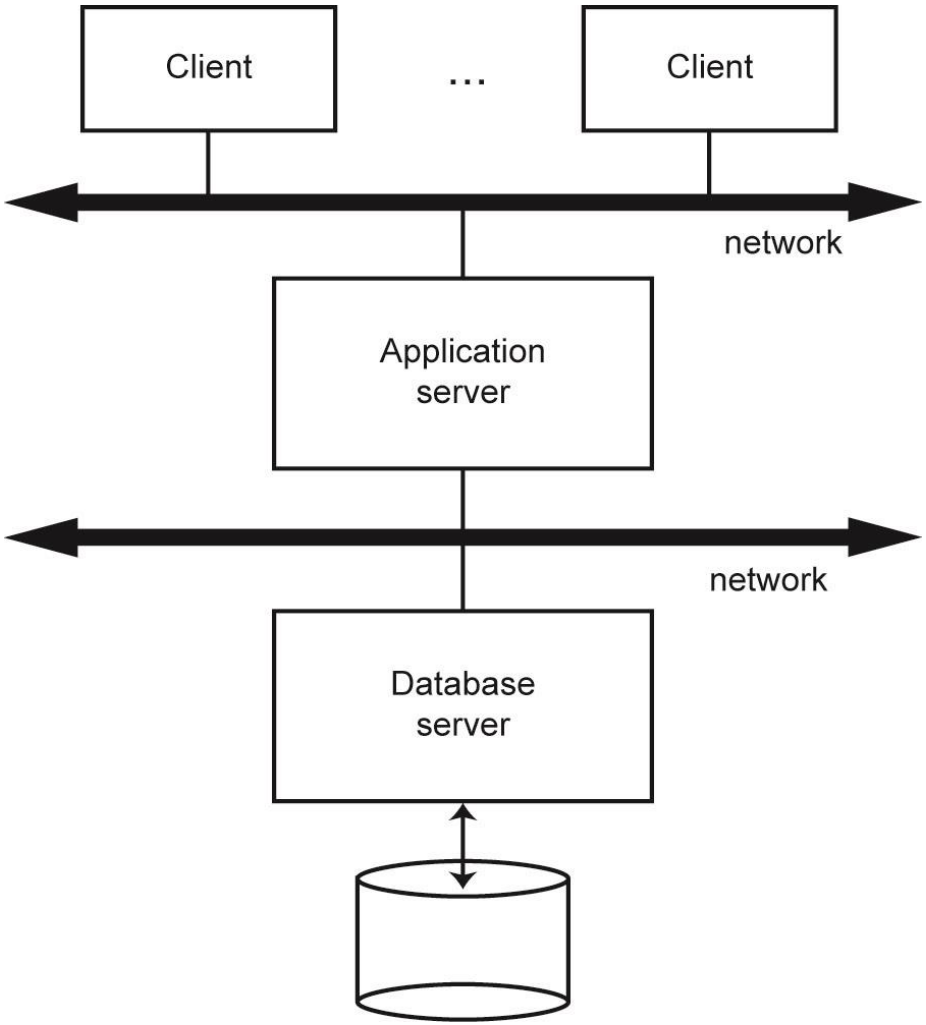
Client/Server Architecture



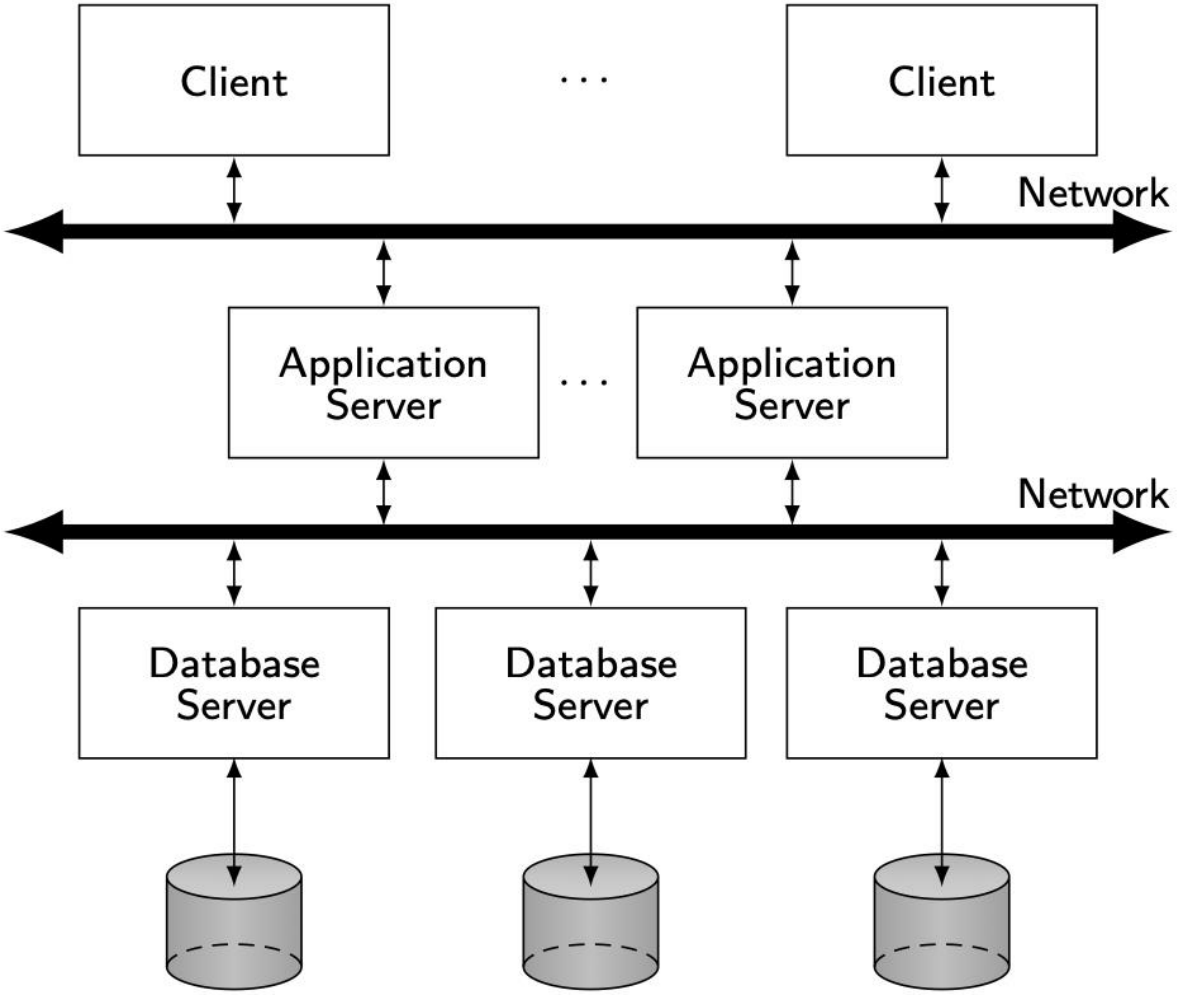
Advantages of Client-Server Architectures

- More efficient division of labor
- Horizontal and vertical scaling of resources
- Better price/performance on client machines
- Ability to use familiar tools on client machines
- Client access to remote data (via standards)
- Full DBMS functionality provided to client workstations
- Overall better system price/performance

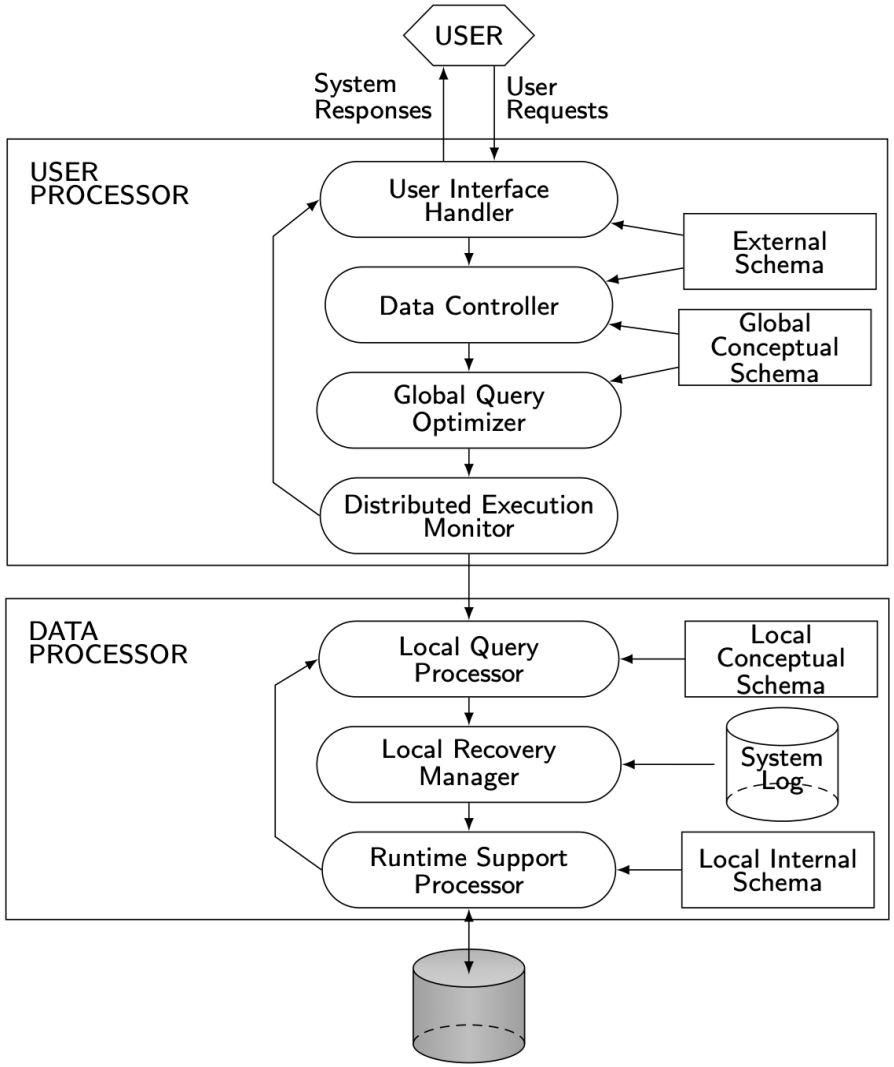
Database Server



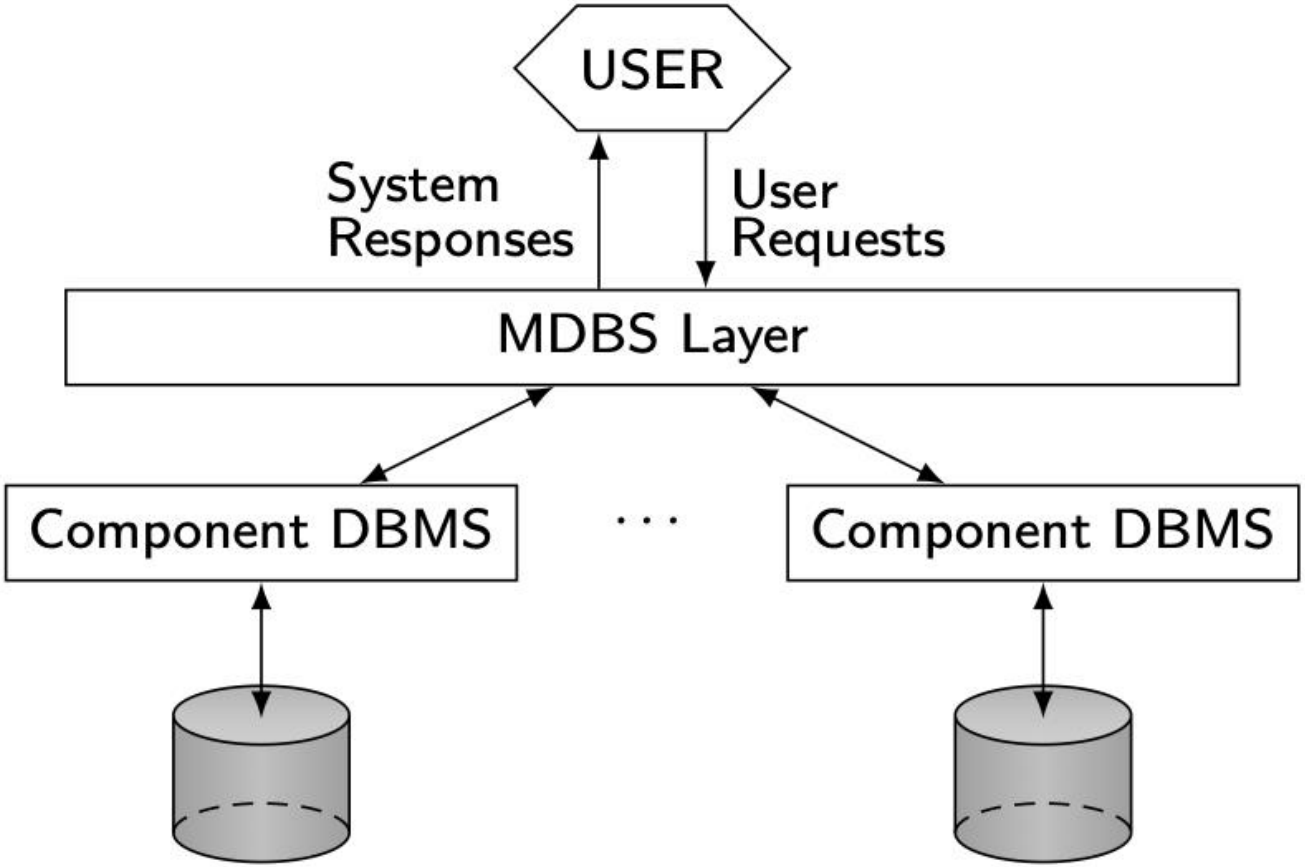
Distributed Database Servers



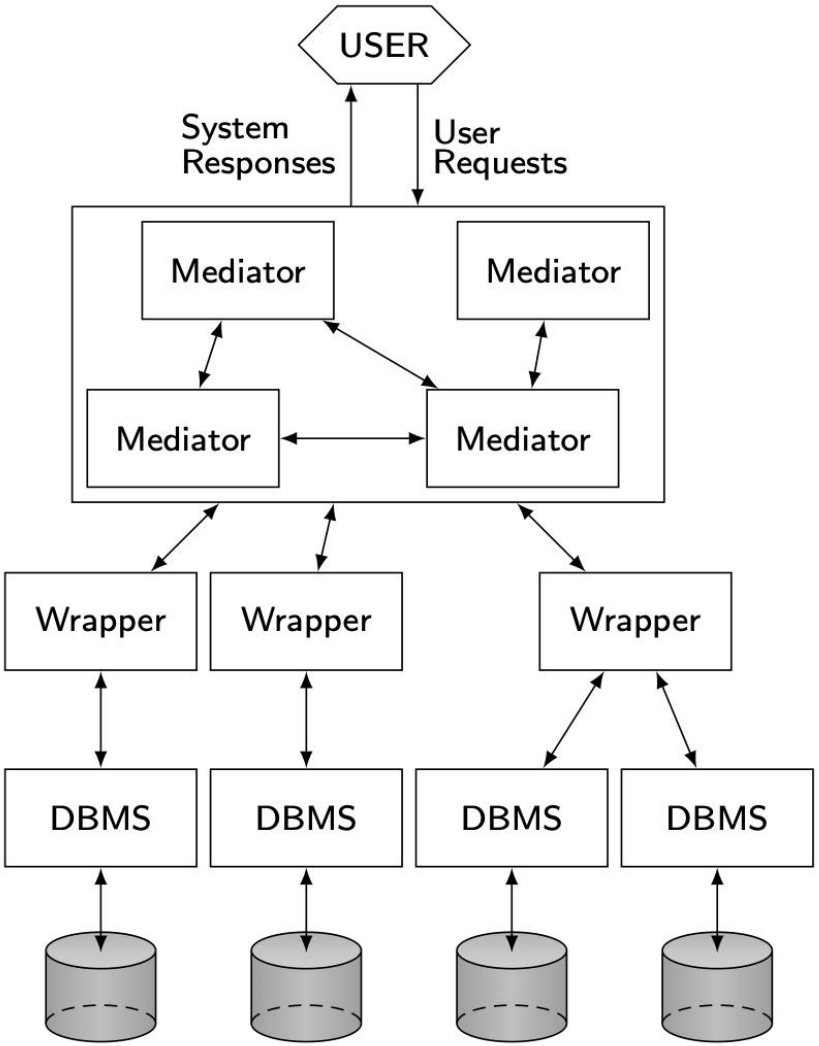
Peer-to-Peer Component Architecture



MDBS Components & Execution



Mediator/Wrapper Architecture

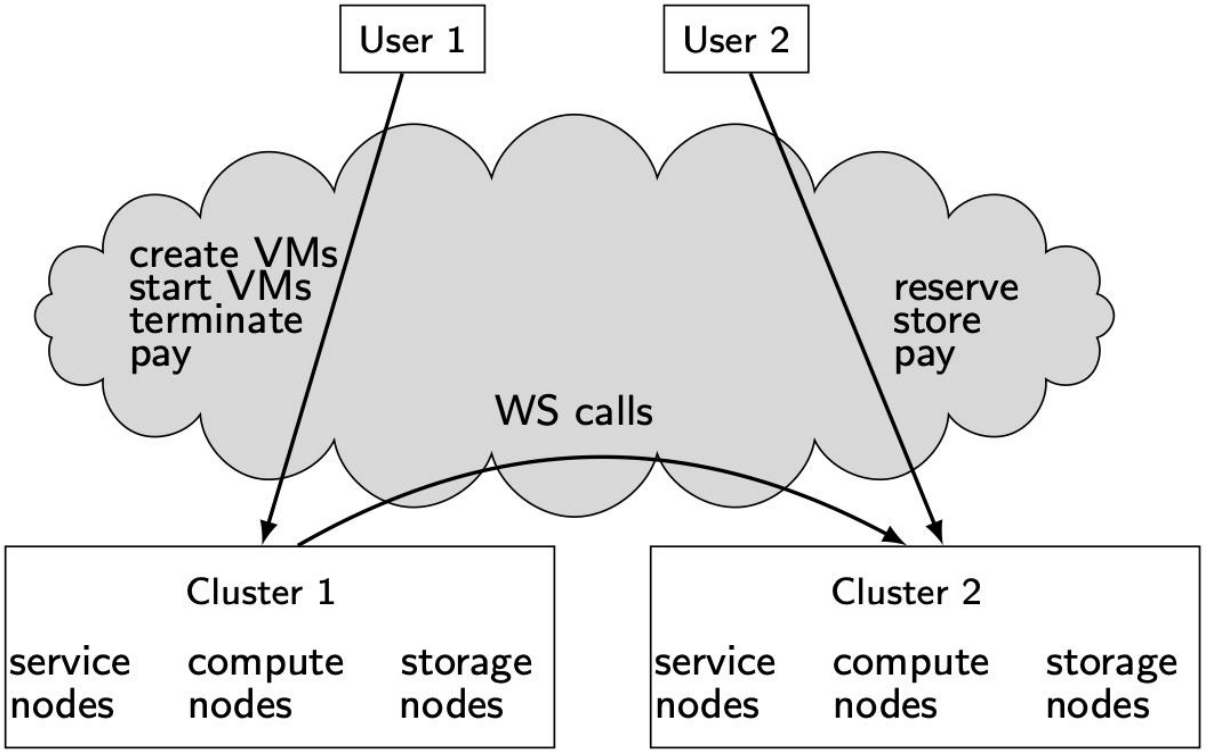


Cloud Computing

On-demand, reliable services provided over the Internet in a cost-efficient manner

- IaaS – Infrastructure-as-a-Service
- PaaS – Platform-as-a-Service
- SaaS – Software-as-a-Service
- DaaS – Database-as-a-Service

Simplified Cloud Architecture



Principles of Distributed Database Systems

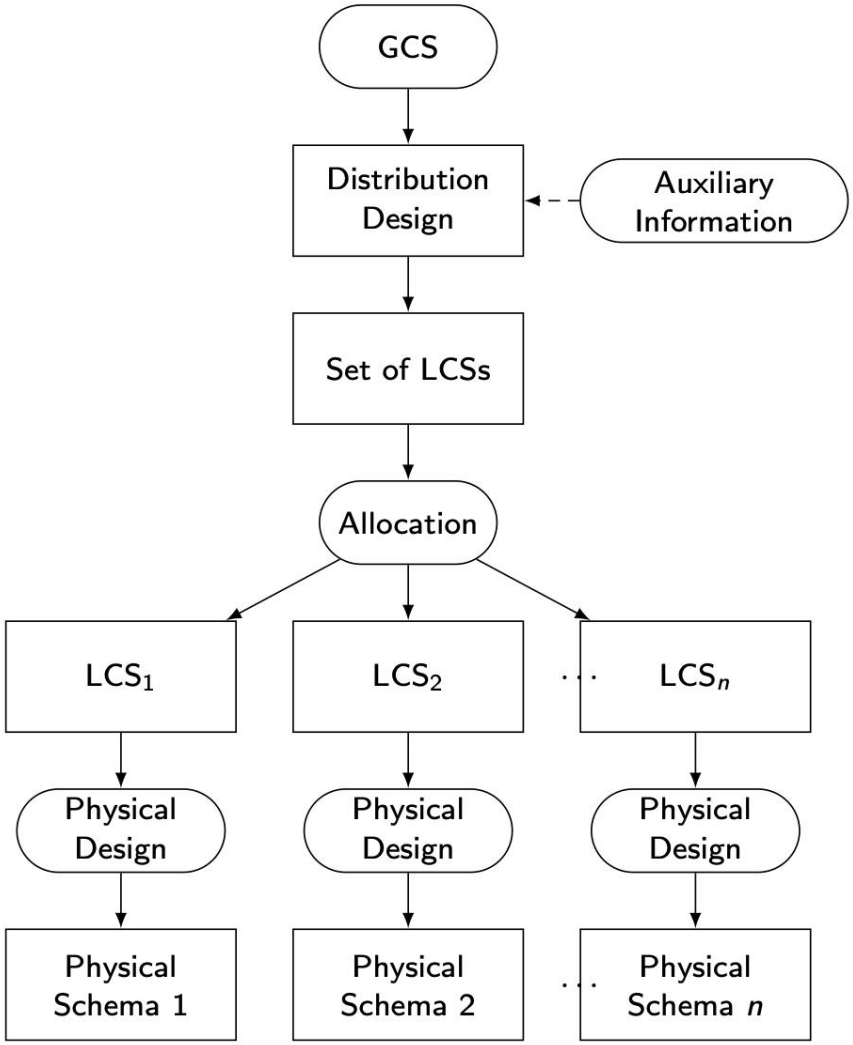
Outline

- Introduction
- Distributed and Parallel Database Design
- Distributed Data Control
- Distributed Query Processing
- Distributed Transaction Processing
- Data Replication
- Database Integration – Multidatabase Systems
- Parallel Database Systems
- Peer-to-Peer Data Management
- Big Data Processing
- NoSQL, NewSQL and Polystores
- Web Data Management

Outline

- Distributed and Parallel Database Design
  - Fragmentation
  - Data distribution
  - Combined approaches

Distribution Design



Outline

- Distributed and Parallel Database Design
  - Fragmentation
  - Data distribution
  - Combined approaches

Fragmentation

- Can't we just distribute relations?
- What is a reasonable unit of distribution?
  - relation
    - views are subsets of relations ➔ locality
    - extra communication
  - fragments of relations (sub-relations)
    - concurrent execution of a number of transactions that access different portions of a relation
    - views that cannot be defined on a single fragment will require extra processing
    - semantic data control (especially integrity enforcement) more difficult

Example Database

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Fragmentation Alternatives – Horizontal

$PROJ_1$: projects with budgets less than $200,000

$PROJ_2$: projects with budgets greater than or equal to $200,000

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	255000	New York
P4	Maintenance	310000	Paris

Fragmentation Alternatives – Vertical

$PROJ_1$: information about project budgets

$PROJ_2$: information about project names and locations

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris

Correctness of Fragmentation

- Completeness
  - Decomposition of relation R into fragments R_1, R_2, ..., R_n is complete if and only if each data item in R can also be found in some R_i
- Reconstruction
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, then there should exist some relational operator ∇ such that

$$R = \nabla_{1 \leq i \leq n} R_i$$

- Disjointness
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, and data item d_i is in R_j, then d_i should not be in any other fragment R_k ($k \neq j$).

Allocation Alternatives

- Non-replicated
  - partitioned : each fragment resides at only one site
- Replicated
  - fully replicated : each fragment at each site
  - partially replicated : each fragment at some of the sites
- Rule of thumb:

$$\text{If } \frac{\text{read-only queries}}{\text{update queries}} << 1, \text{ replication is advantageous,}$$
$$\text{otherwise replication may cause problems}$$

Comparison of Replication Alternatives

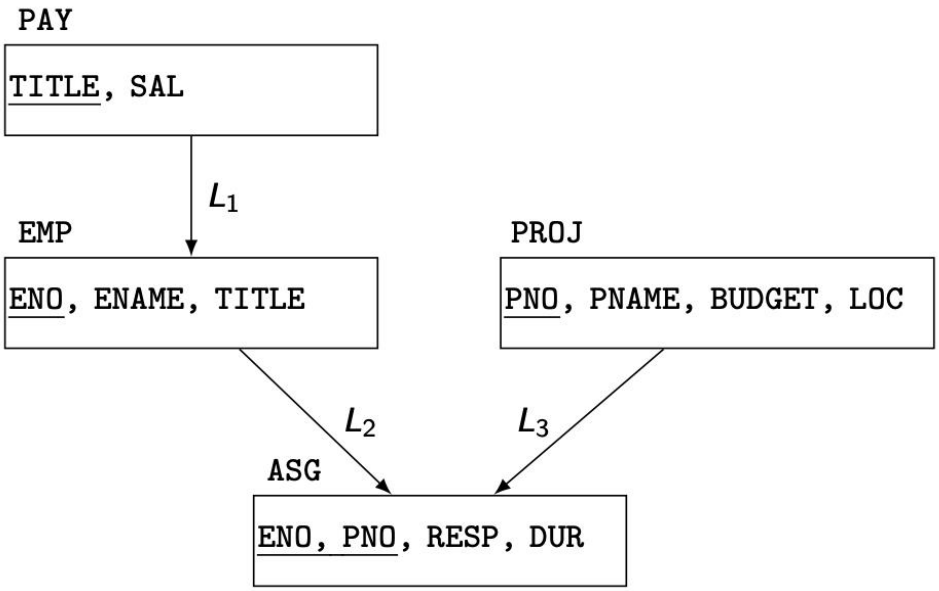
	Full replication	Partial replication	Partitioning
QUERY PROCESSING	Easy	Same difficulty	
DIRECTORY MANAGEMENT	Easy or nonexistent	Same difficulty	
CONCURRENCY CONTROL	Moderate	Difficult	Easy
RELIABILITY	Very high	High	Low
REALITY	Possible application	Realistic	Possible application

Fragmentation

- Horizontal Fragmentation (HF)
  - Primary Horizontal Fragmentation (PHF)
  - Derived Horizontal Fragmentation (DHF)
- Vertical Fragmentation (VF)
- Hybrid Fragmentation (HF)

PHF – Information Requirements

- Database Information
  - relationship



  - cardinality of each relation: *card*(*R*)

PHF - Information Requirements

- Application Information
  - **simple predicates** : Given $R[A_1, A_2, \ldots, A_n]$, a simple predicate p_j is

    $$p_j : A_i \, \theta Value$$

    where $\theta \in \{=,<,\leq,>,\geq,\neq\}$, $Value \in D_i$ and D_i is the domain of A_i.

    For relation R we define $Pr = \{p_1, p_2, \ldots, p_m\}$

    Example :

    PNAME = "Maintenance"

    BUDGET ≤ 200000
  - **minterm predicates** : Given R and $Pr = \{p_1, p_2, \ldots, p_m\}$ define $M = \{m_1, m_2, \ldots, m_r\}$ as

    $$M = \{ m_i \mid m_i = \bigwedge_{p_j \in Pr} p_j^* \}, 1 \leq j \leq m, 1 \leq i \leq z$$

    where $p_j^* = p_j$ or $p_j^* = \neg(p_j)$.

PHF – Information Requirements

Example

m_1: PNAME="Maintenance" $\wedge$ BUDGET≤200000

m_2: **NOT**(PNAME="Maintenance") $\wedge$ BUDGET≤200000

m_3: PNAME= "Maintenance" $\wedge$ **NOT**(BUDGET≤200000)

m_4: **NOT**(PNAME="Maintenance") $\wedge$ **NOT**(BUDGET≤200000)

PHF – Information Requirements

- Application Information
  - **minterm selectivitie**s: *sel*(m_i)
    - The number of tuples of the relation that would be accessed by a user query which is specified according to a given minterm predicate m_i.
  - **access frequencies**: *acc*(q_i)
    - The frequency with which a user application *qi* accesses data.
    - Access frequency for a minterm predicate can also be defined.

Primary Horizontal Fragmentation

Definition :

$$R_j = \sigma_{F_j}(R), \quad 1 \leq j \leq w$$

where F_j is a selection formula, which is (preferably) a minterm predicate.

Therefore,

A horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i.

Given a set of minterm predicates *M,* there are as many horizontal fragments of relation *R* as there are minterm predicates.

Set of horizontal fragments also referred to as minterm fragments.

PHF – Algorithm

Given: A relation R, the set of simple predicates Pr

Output: The set of fragments of $R = \{R_1, R_2, \ldots, R_w\}$ which obey the fragmentation rules.

Preliminaries :

- Pr should be *complete*
- Pr should be *minimal*

Completeness of Simple Predicates

- A set of simple predicates *Pr* is said to be *complete* if and only if the accesses to the tuples of the minterm fragments defined on *Pr* requires that two tuples of the same minterm fragment have the same probability of being accessed by any application.

- Example :
  - Assume PROJ[PNO,PNAME,BUDGET,LOC] has two applications defined on it.
  - Find the budgets of projects at each location. (1)
  - Find projects with budgets less than $200000. (2)

Completeness of Simple Predicates

According to (1),

Pr={LOC="Montreal",LOC="New York",LOC="Paris"}

which is not complete with respect to (2).

Modify

Pr ={LOC="Montreal",LOC="New York",LOC="Paris",
BUDGET≤200000,BUDGET>200000}

which is complete.

Minimality of Simple Predicates

- If a predicate influences how fragmentation is performed, (i.e., causes a fragment *f* to be further fragmented into, say, f_i and f_j) then there should be at least one application that accesses f_i and f_j differently.
- In other words, the simple predicate should be *relevant* in determining a fragmentation.
- If all the predicates of a set *Pr* are relevant, then *Pr* is *minimal*.

$$\frac{acc(m_i)}{card(f_i)} = \frac{acc(m_j)}{card(f_j)}$$

Minimality of Simple Predicates

Example :

Pr ={LOC="Montreal",LOC="New York", LOC="Paris",
BUDGET≤200000,BUDGET>200000}

is minimal (in addition to being complete). However, if we add

PNAME = "Instrumentation"

then *Pr* is not minimal.

COM_MIN Algorithm

Given: a relation *R* and a set of simple predicates *Pr*

Output: a *complete* and *minimal* set of simple predicates *Pr'* for *Pr*

Rule 1: a relation or fragment is partitioned into at least two parts which are accessed differently by at least one application.

COM_MIN Algorithm

1. Initialization :
   - find a $p_i \in Pr$ such that p_i partitions R according to *Rule 1*
   - set $Pr' = p_i$; $Pr \leftarrow Pr - \{p_i\}$; $F \leftarrow \{f_i\}$
2. Iteratively add predicates to Pr' until it is complete
   - find a $p_j \in Pr$ such that p_j partitions some f_k defined according to minterm predicate over Pr' according to *Rule 1*
   - set $Pr' = Pr' \cup \{p_i\}$; $Pr \leftarrow Pr - \{p_i\}$; $F \leftarrow F \cup \{f_i\}$
   - if $\exists p_k \in Pr'$ which is nonrelevant then

     $Pr' \leftarrow Pr - \{p_i\}$

     $F \leftarrow F - \{f_i\}$

PHORIZONTAL Algorithm

Makes use of COM_MIN to perform fragmentation.

Input: a relation R and a set of simple predicates Pr

Output: a set of minterm predicates M according to which relation R is to be fragmented

❶ $Pr' \leftarrow$ COM_MIN (R,Pr)

❷ determine the set M of minterm predicates

❸ determine the set I of implications among $p_i \in Pr$

❹ eliminate the contradictory minterms from M

PHF – Example

- Two candidate relations : PAY and PROJ.
- Fragmentation of relation PAY
  - Application: Check the salary info and determine raise.
  - Employee records kept at two sites $\Rightarrow$ application run at two sites
  - Simple predicates

    p_1 : SAL $\leq$ 30000

    p_2 : SAL > 30000

    $Pr = \{p_1, p_2\}$ which is complete and minimal $Pr' = Pr$
  - Minterm predicates

    m_1 : (SAL $\leq$ 30000)

    m_2 : **NOT**(SAL $\leq$ 30000) = (SAL > 30000)

PHF – Example

PAY_1

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

PAY_2

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000

PHF – Example

- Fragmentation of relation PROJ
  - Applications:
    - Find the name and budget of projects given their no.
      - Issued at three sites
    - Access project information according to budget
      - one site accesses ≤200000 other accesses >200000
  - Simple predicates
  - For application (1)

    p_1 : LOC = “Montreal”

    p_2 : LOC = “New York”

    p_3 : LOC = “Paris”
  - For application (2)

    p_4 : BUDGET ≤ 200000

    p_5 : BUDGET > 200000
  - $Pr = Pr' = \{p_1, p_2, p_3, p_4, p_5\}$

PHF – Example

- Fragmentation of relation PROJ continued
  - Minterm fragments left after elimination

m_1 : (LOC = “Montreal”) $\wedge$ (BUDGET $\leq$ 200000)

m_2 : (LOC = “Montreal”) $\wedge$ (BUDGET > 200000)

m_3 : (LOC = “New York”) $\wedge$ (BUDGET $\leq$ 200000)

m_4 : (LOC = “New York”) $\wedge$ (BUDGET > 200000)

m_5 : (LOC = “Paris”) $\wedge$ (BUDGET $\leq$ 200000)

m_6 : (LOC = “Paris”) $\wedge$ (BUDGET > 200000)

PHF – Example

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_3$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_4$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	255000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

PHF – Correctness

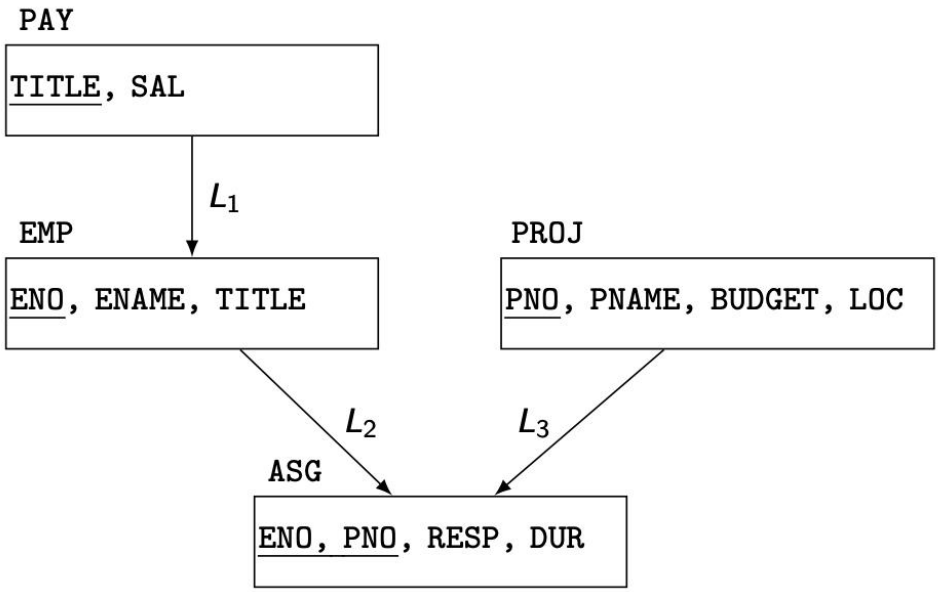
- Completeness
  - Since *Pr*' is complete and minimal, the selection predicates are complete
- Reconstruction
  - If relation R is fragmented into $F_R = \{R_1, R_2, \ldots, R_r\}$

$$R = \cup_{\forall R_i \in FR} R_i$$

- Disjointness
  - Minterm predicates that form the basis of fragmentation should be mutually exclusive.

Derived Horizontal Fragmentation

- Defined on a member relation of a link according to a selection operation specified on its owner.
  - Each link is an equijoin.
  - Equijoin can be implemented by means of semijoins.



DHF – Definition

Given a link L where $owner(L)=S$ and $member(L)=R$, the derived horizontal fragments of R are defined as

$$R_i = R \ltimes_F S_i,\ 1 \leq i \leq w$$

where w is the maximum number of fragments that will be defined on R and

$$S_i = \sigma_{F_i}(S)$$

where F_i is the formula according to which the primary horizontal fragment S_i is defined.

DHF – Example

Given link L_1 where owner(L_1)=SKILL and member(L_1)=EMP

$EMP_1 = EMP \ltimes SKILL_1$

$EMP_2 = EMP \ltimes SKILL_2$

where

$SKILL_1 = \sigma_{SAL \leq 30000}(SKILL)$

$SKILL_2 = \sigma_{SAL > 30000}(SKILL)$

ASG_1

ENO	PNO	RESP	DUR
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E7	P3	Engineer	36

ASG_2

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E5	P2	Manager	24
E6	P4	Manager	48
E8	P3	Manager	40

DHF – Correctness

- Completeness
  - Referential integrity
  - Let R be the member relation of a link whose owner is relation S which is fragmented as $F_S = \{S_1, S_2, ..., S_n\}$. Furthermore, let A be the join attribute between R and S. Then, for each tuple t of R, there should be a tuple t' of S such that

    $$t[A] = t'[A]$$
- Reconstruction
  - Same as primary horizontal fragmentation.
- Disjointness
  - Simple join graphs between the owner and the member fragments.

Vertical Fragmentation

- Has been studied within the centralized context
  - design methodology
  - physical clustering
- More difficult than horizontal, because more alternatives exist.

  Two approaches :
  - grouping
    - attributes to fragments
  - splitting
    - relation to fragments

Vertical Fragmentation

- Overlapping fragments
  - grouping
- Non-overlapping fragments
  - splitting

We do not consider the replicated key attributes to be overlapping.

Advantage:

Easier to enforce functional dependencies

(for integrity checking etc.)

VF – Information Requirements

- Application Information
  - Attribute affinities
    - a measure that indicates how closely related the attributes are
    - This is obtained from more primitive usage data
  - Attribute usage values
    - Given a set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ that will run on the relation $R[A_1, A_2, \ldots, A_n]$,

$$use(q_i, A_j) = \begin{cases} 1 \text{ if attribute } A_j \text{ is referenced by query } q_i \\ 0 \text{ otherwise} \end{cases}$$

$use(q_i, \bullet)$ can be defined accordingly

VF – Definition of $use(q_i, A_j)$

Consider the following 4 queries for relation PROJ

q_1: **SELECT** BUDGET
FROM PROJ
WHERE PNO=Value

q_2: **SELECT** PNAME,BUDGET
FROM PROJ

q_3: **SELECT** PNAME
FROM PROJ
WHERE LOC=Value

q_4: **SELECT** **SUM**(BUDGET)
FROM PROJ
WHERE LOC=Value

	PNO	PNAME	BUDGET	LOC
q_1	0	1	1	0
q_2	1	1	1	0
q_3	1	0	0	1
q_4	0	0	1	0

VF – Affinity Measure $aff(A_i,A_j)$

The attribute affinity measure between two attributes A_i and A_j of a relation $R[A_1, A_2, \ldots, A_n]$ with respect to the set of applications $Q = (q_1, q_2, \ldots, q_q)$ is defined as follows :

$$aff(A_i, A_j) = \sum_{\text{all queries that access } A_i \text{ and } A_j} \text{(query access)}$$

$$\text{query access} = \sum_{\text{all sites}} \text{access frequency of a query} * \frac{\text{access}}{\text{execution}}$$

VF – Calculation of $aff(A_i, A_j)$

Assume each query in the previous example accesses the attributes once during each execution.

Also assume the access frequencies

	S_1	S_2	S_3
q_1	15	20	10
q_2	5	0	0
q_3	25	25	25
q_4	3	0	0

Then

$aff(A_1, A_3)$ = 15*1 + 20*1+10*1
= 45

and the attribute affinity matrix AA is
(Let A_1=PNO, A_2=PNAME, A_3=BUDGET, A_4=LOC)

	PNO	PNAME	BUDGET	LOC
PNO	–	0	45	0
PNAME	0	–	5	75
BUDGET	45	5	–	3
LOC	0	75	3	–

VF – Clustering Algorithm

- Take the attribute affinity matrix *AA* and reorganize the attribute orders to form clusters where the attributes in each cluster demonstrate high affinity to one another.
- Bond Energy Algorithm (BEA) has been used for clustering of entities. BEA finds an ordering of entities (in our case attributes) such that the global affinity measure is maximized.

$$AM = \sum_i \sum_j \text{(affinity of } A_i \text{ and } A_j \text{ with their neighbors)}$$

Bond Energy Algorithm

Input: The *AA* matrix

Output: The clustered affinity matrix *CA* which is a perturbation of *AA*

❶ *Initialization*: Place and fix one of the columns of *AA* in *CA*.

❷ *Iteration*: Place the remaining *n-i* columns in the remaining *i*+1 positions in the *CA* matrix. For each column, choose the placement that makes the most contribution to the global affinity measure.

❸ *Row order*: Order the rows according to the column ordering.

Bond Energy Algorithm

"Best" placement? Define contribution of a placement:

$$cont(A_i, A_k, A_j) = 2bond(A_i, A_k)+2bond(A_k, A_l) -2bond(A_i, A_j)$$

where

$$bond(A_x,A_y) = \sum_{z=1}^{n} aff(A_z,A_x)aff(A_z,A_y)$$

BEA – Example

Consider the following *AA* matrix and the corresponding *CA* matrix where PNO and PNAME have been placed. Place BUDGET:

	PNO	PNAME	BUDGET	LOC
PNO	–	0	45	0
PNAME	0	–	5	75
BUDGET	45	5	–	3
LOC	0	75	3	–

	PNO	PNAME
PNO	45	0
PNAME	0	80
BUDGET	45	5
LOC	0	75

Ordering (0-3-1) :

$cont(A_0, \text{BUDGET}, \text{PNO}) = 2bond(A_0, \text{BUDGET}) + 2bond(\text{BUDGET}, \text{PNO}) - 2bond(A_0, \text{PNO})$

$= 8820$

Ordering (1-3-2) :

$cont(\text{PNO}, \text{BUDGET}, \text{PNAME}) = 10150$

Ordering (2-3-4) :

$cont(\text{PNAME}, \text{BUDGET}, \text{LOC}) = 1780$

BEA – Example

- Therefore, the CA matrix has the form

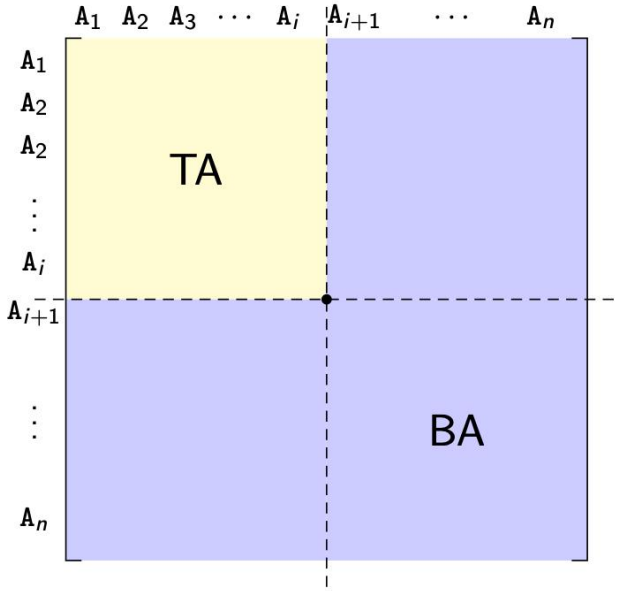
	PNO	BUDGET	PNAME
PNO	45	45	0
PNAME	0	5	80
BUDGET	45	53	5
LOC	0	3	75

- When LOC is placed, the final form of the *CA* matrix (after row organization) is

	PNO	BUDGET	PNAME	LOC
PNO	45	45	0	0
BUDGET	45	53	5	3
PNAME	0	5	80	75
LOC	0	3	75	78

VF – Algorithm

How can you divide a set of clustered attributes $\{A_1, A_2, \ldots, A_n\}$ into two (or more) sets $\{A_1, A_2, \ldots, A_i\}$ and $\{A_i, \ldots, A_n\}$ such that there are no (or minimal) applications that access both (or more than one) of the sets.



VF – ALgorithm

Define

TQ = set of applications that access only *TA*

BQ = set of applications that access only *BA*

OQ = set of applications that access both *TA* and *BA*

and

CTQ = total number of accesses to attributes by applications that access only *TA*

CBQ = total number of accesses to attributes by applications that access only *BA*

COQ = total number of accesses to attributes by applications that access both *TA* and *BA*

Then find the point along the diagonal that maximizes

$$CTQ * CBQ - COQ^2$$

VF – Algorithm

Two problems :

- Cluster forming in the middle of the *CA* matrix
  - Shift a row up and a column left and apply the algorithm to find the “best” partitioning point
  - Do this for all possible shifts
  - Cost $O(m^2)$
- More than two clusters
  - *m*-way partitioning
  - try 1, 2, …, *m*–1 split points along diagonal and try to find the best point for each of these
  - Cost $O(2^m)$

VF – Correctness

A relation R, defined over attribute set A and key K, generates the vertical partitioning $F_R = \{R_1, R_2, \ldots, R_r\}$.

- Completeness
  - The following should be true for A:

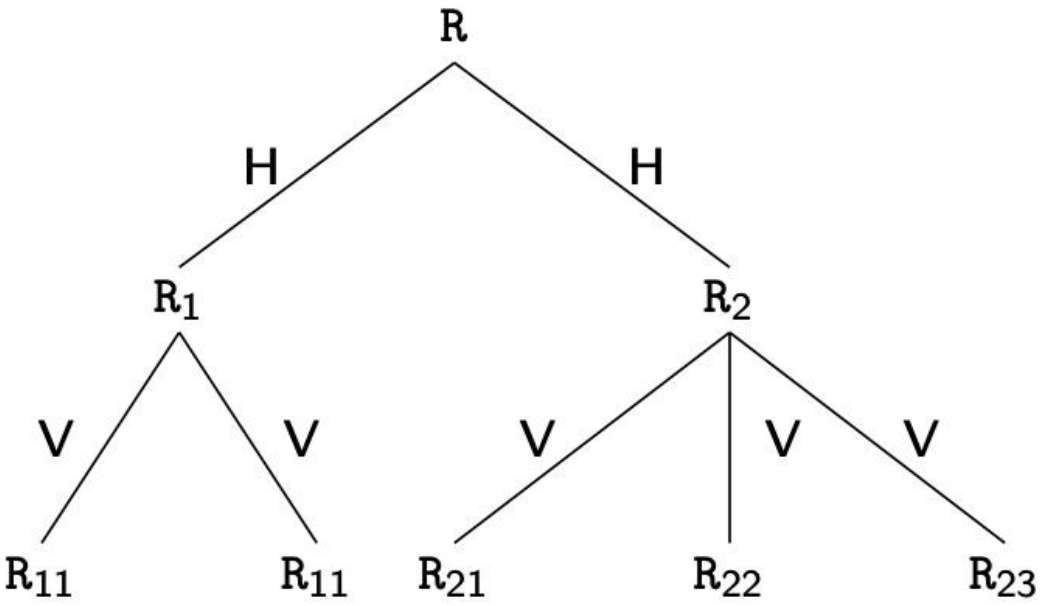
$$A = \cup A_{R_i}$$

- Reconstruction
  - Reconstruction can be achieved by

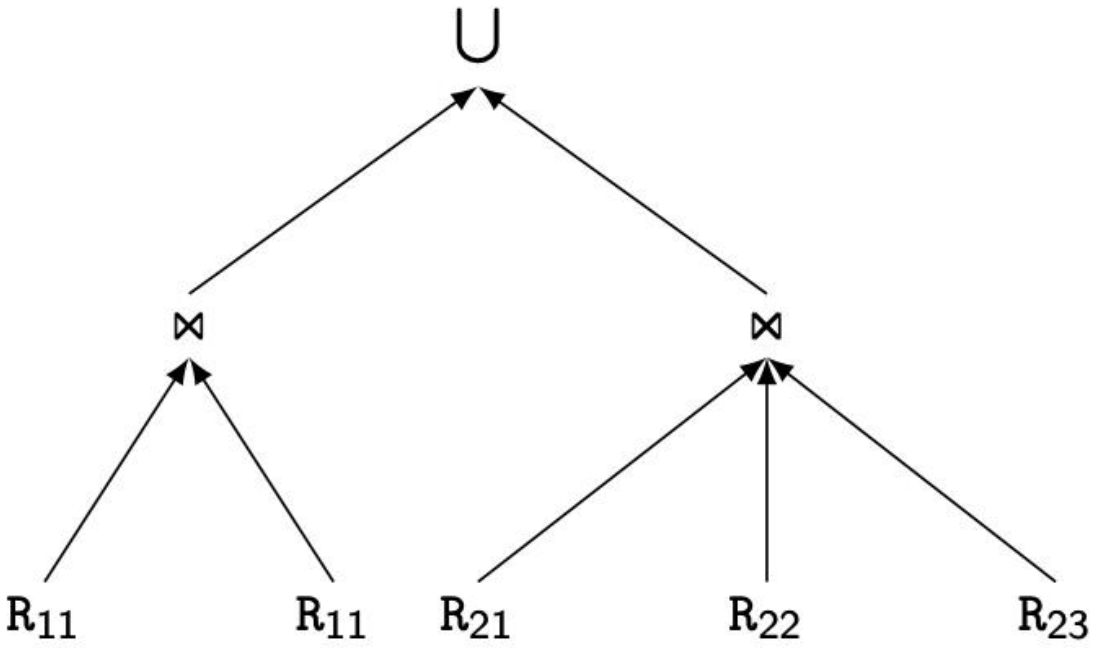
$$R = \bowtie_K R_i, \forall R_i \in F_R$$

- Disjointness
  - TID's are not considered to be overlapping since they are maintained by the system
  - Duplicated keys are not considered to be overlapping

Hybrid Fragmentation



Reconstruction of HF



Outline

- Distributed and Parallel Database Design
  - Fragmentation
  - Data distribution
  - Combined approaches

Fragment Allocation

- Problem Statement

  Given

  $F = \{F_1, F_2, \ldots, F_n\}$ fragments

  $S = \{S_1, S_2, \ldots, S_m\}$ network sites

  $Q = \{q_1, q_2, \ldots, q_q\}$ applications

  Find the "optimal" distribution of F to S.

- Optimality
  - Minimal cost
    - Communication + storage + processing (read & update)
    - Cost in terms of time (usually)
  - Performance

    Response time and/or throughput
  - Constraints
    - Per site constraints (storage & processing)

Information Requirements

- Database information
  - selectivity of fragments
  - size of a fragment
- Application information
  - access types and numbers
  - access localities
- Communication network information
  - unit cost of storing data at a site
  - unit cost of processing at a site
- Computer system information
  - bandwidth
  - latency
  - communication overhead

Allocation

File Allocation (FAP) vs Database Allocation (DAP):

- Fragments are not individual files
  - relationships have to be maintained
- Access to databases is more complicated
  - remote file access model not applicable
  - relationship between allocation and query processing
- Cost of integrity enforcement should be considered
- Cost of concurrency control should be considered

Allocation Model

General Form

min(Total Cost)

subject to

response time constraint

storage constraint

processing constraint

Decision Variable

$$x_{ij} = \begin{cases} 1 \text{ if fragment } F_i \text{ is stored at site } S_j \\ 0 \text{ otherwise} \end{cases}$$

Allocation Model

- Total Cost

$$\sum_{\text{all queries}} \text{query processing cost} +$$

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} \text{cost of storing a fragment at a site}$$

- Storage Cost (of fragment F_j at S_k)

(unit storage cost at S_k) $*$ (size of F_j) $*$ x_{jk}

- Query Processing Cost (for one query)

processing component + transmission component

Allocation Model

- Query Processing Cost

  Processing component

  access cost + integrity enforcement cost + concurrency control cost

  - Access cost

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} (\text{no. of update accesses} + \text{no. of read accesses}) * x_{ij} * \text{local processing cost at a site}$$

  - Integrity enforcement and concurrency control costs
    - Can be similarly calculated

Allocation Model

- Query Processing Cost
  - Transmission component

    cost of processing updates + cost of processing retrievals
  - Cost of updates

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} \text{update message cost} +$$
$$\sum_{\text{all sites}} \sum_{\text{all fragments}} \text{acknowledgment cost}$$

  - Retrieval Cost

$$\sum_{\text{all fragments}} \min_{\text{all sites}} (\text{cost of retrieval command} +$$
$$\text{cost of sending back the result})$$

Allocation Model

- Constraints
  - Response Time

    execution time of query ≤ max. allowable response time for that query

  - Storage Constraint (for a site)

    $$\sum_{\text{all fragments}} \text{storage requirement of a fragment at that site} \leq \text{storage capacity at that site}$$

  - Processing constraint (for a site)

    $$\sum_{\text{all queries}} \text{processing load of a query at that site} \leq \text{processing capacity of that site}$$

Allocation Model

- Solution Methods
  - FAP is NP-complete
  - DAP also NP-complete
- Heuristics based on
  - single commodity warehouse location (for FAP)
  - knapsack problem
  - branch and bound techniques
  - network flow

Allocation Model

- Attempts to reduce the solution space
  - assume all candidate partitionings known; select the “best” partitioning
  - ignore replication at first
  - sliding window on fragments

Outline

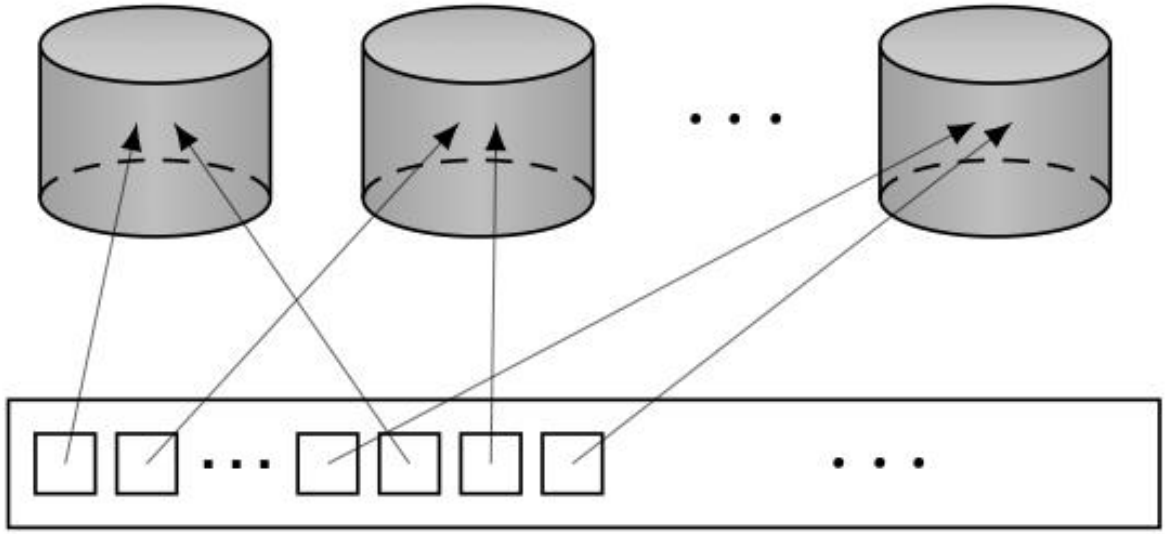
- Distributed and Parallel Database Design
  - Fragmentation
  - Data distribution
  - Combined approaches

Combining Fragmentation & Allocation

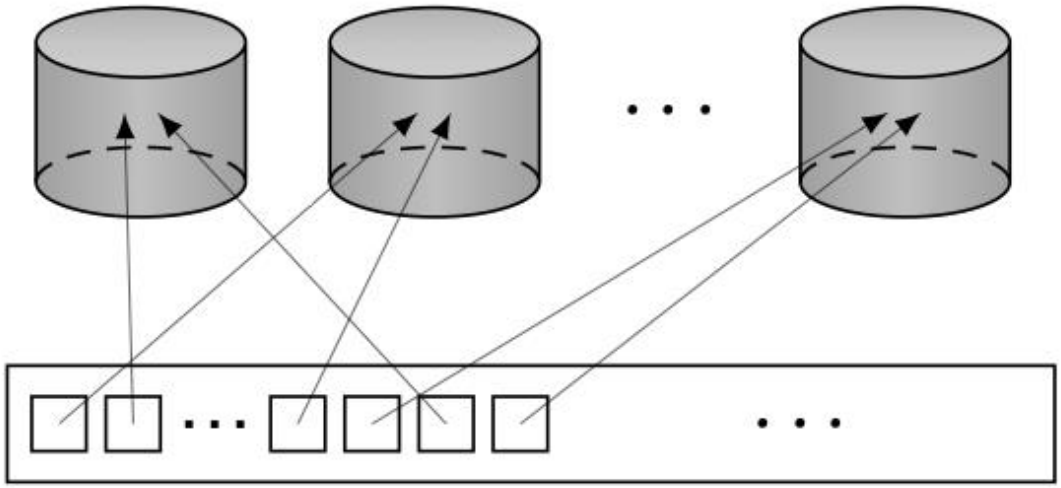
Partition the data to dictate where it is located

- Workload-agnostic techniques
  - Round-robin partitioning
  - Hash partitioning
  - Range partitioning
- Workload-aware techniques
  - Graph-based approach

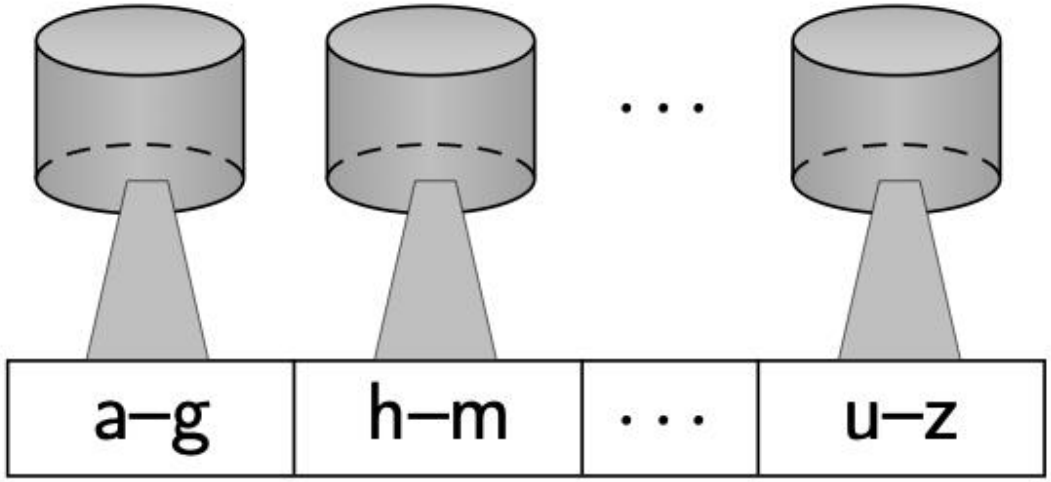
Round-robin Partitioning



Hash Partitioning

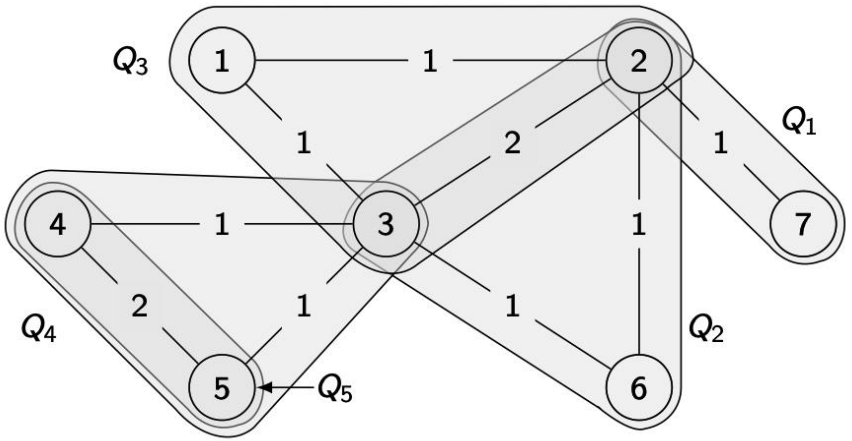


Range Partitioning



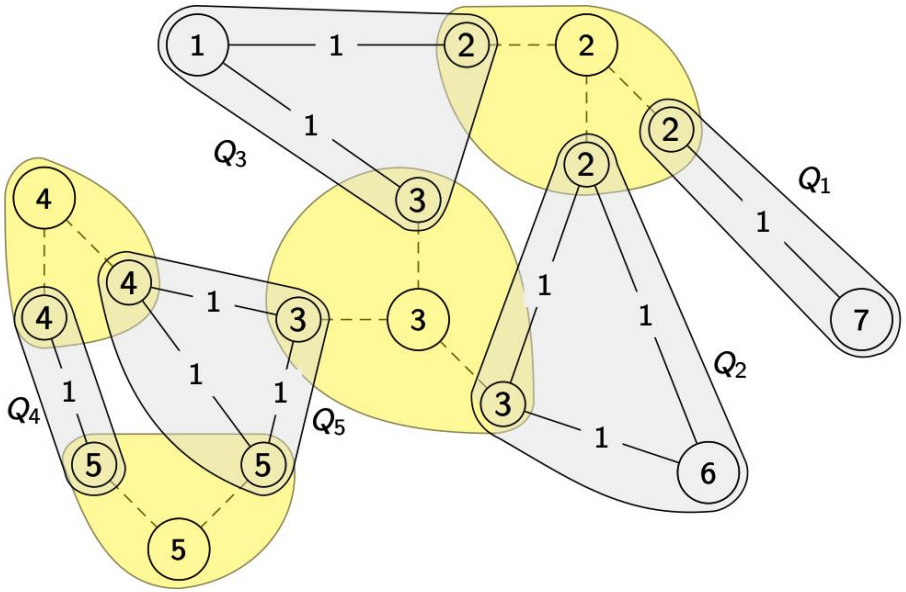
Workload-Aware Partitioning

- Examplar: Schism
  - Graph $G=(V,E)$ where
    - vertex $v_i \in V$ represents a tuple in database,
    - edge $e=(v_i,v_j) \in E$ represents a query that accesses both tuples v_i and v_j;
    - each edge has weight counting the no. of queries that access both tuples
  - Perform vertex disjoint graph partitioning
    - Each vertex is assigned to a separate partition



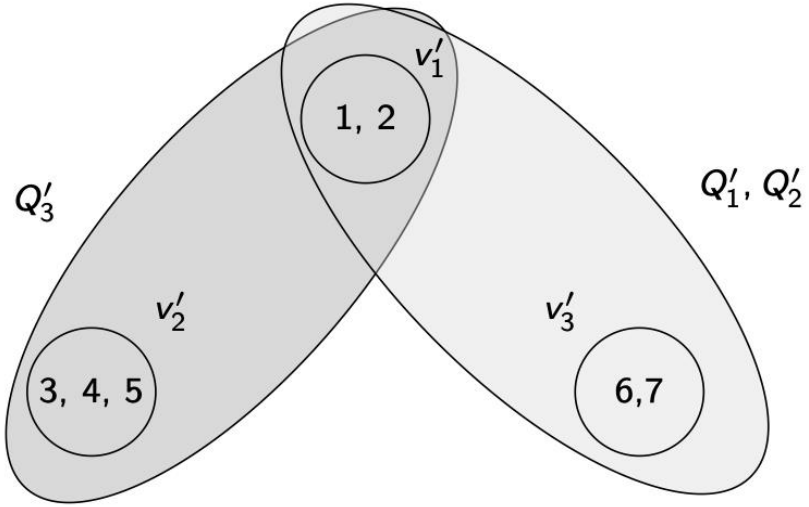
Incorporating Replication

- Replicate each vertex based on the no. of transactions accessing that tuple ➔ each transaction accesses a separate copy



Dealing with graph size

- Each tuple a vertex ➔ graph too big ➔ directory too big
- SWORD
  - Use hypergraph model
  - Compress the directory



Adaptive approaches

- Redesign as physical (network characteristics, available storage) and logical (workload) changes occur.
- Most focus on logical
- Most follow combined approach
- Three issues:
  1. How to detect workload changes?
  2. How to determine impacted data items?
  3. How to perform changes efficiently?

Detecting workload changes

- Not much work
- Periodically analyze system logs
- Continuously monitor workload within DBMS
  - SWORD: no. of distributed queries
  - E-Store: monitor system-level metrics (e.g., CPU utilization) and tuple-level access

Detecting affected data items

- Depends on the workload change detection method
- If monitoring queries ➔ queries will identify data items
  - Apollo: generalize from "similar" queries

```
SELECT PNAME FROM PROJ WHERE BUDGET>20000 AND
LOC='LONDON'
```

```
SELECT PNAME FROM PROJ WHERE BUDGET>? AND LOC='?'
```

- If monitoring tuple-level access (E-Store), this will tell you

Performing changes

- Periodically compute redistribution
  - Not efficient
- Incremental computation and migration
  - Graph representation ➔ look at changes in graph
    - SWORD and AdaptCache: Incremental graph partitioning initiates data migration for reconfiguration
  - E-Store: determine hot tuples for which a migration plan is prepared determine; cold tuple reallocation as well
    - Optimization problem; real-time heuristic solutions
  - Database cracking: continuously reorganize data to match query workload
    - Incoming queries are used as advice
    - When a node needs data for a local query, this is hint that data may need to be moved