Distributed DB architectures: An Introduction

Dario Della Monica

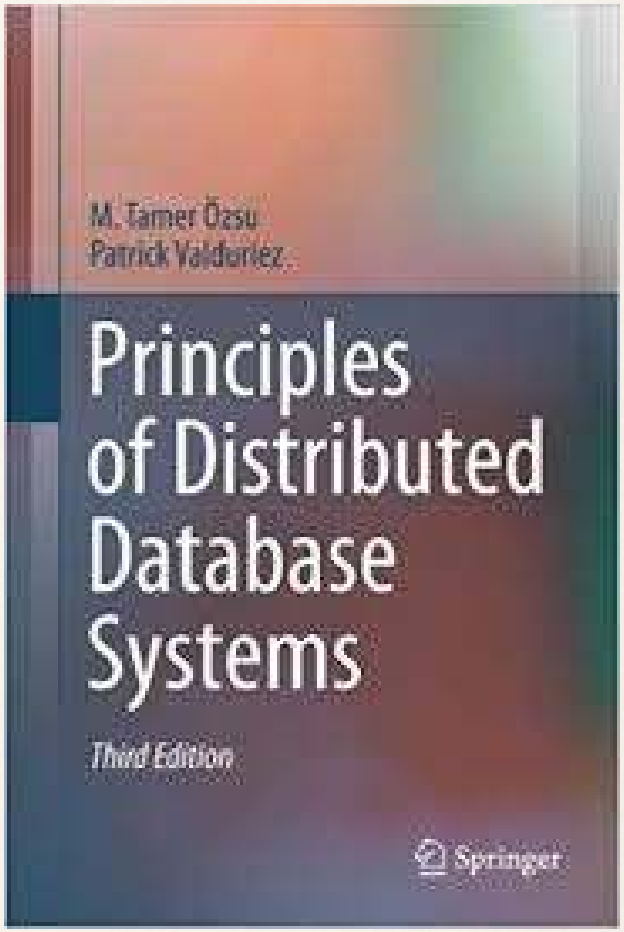
These slides are a modified version of the slides provided with the book

Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

The original version of the slides is available at: extras.springer.com

Outline (distributed DB)

- Introduction (Ch. 1) *
  - ➡ What is a distributed DBMS
  - ➡ Distributed DBMS Architecture
- Distributed Database Design (Ch. 3) *
- Distributed Query Processing (Ch. 6-8) *
- Distributed Transaction Management (Ch. 10-12) *



* **Özsu and Valduriez,**
Principles of Distributed Database Systems,
3rd edition, 2011

Outline (today)

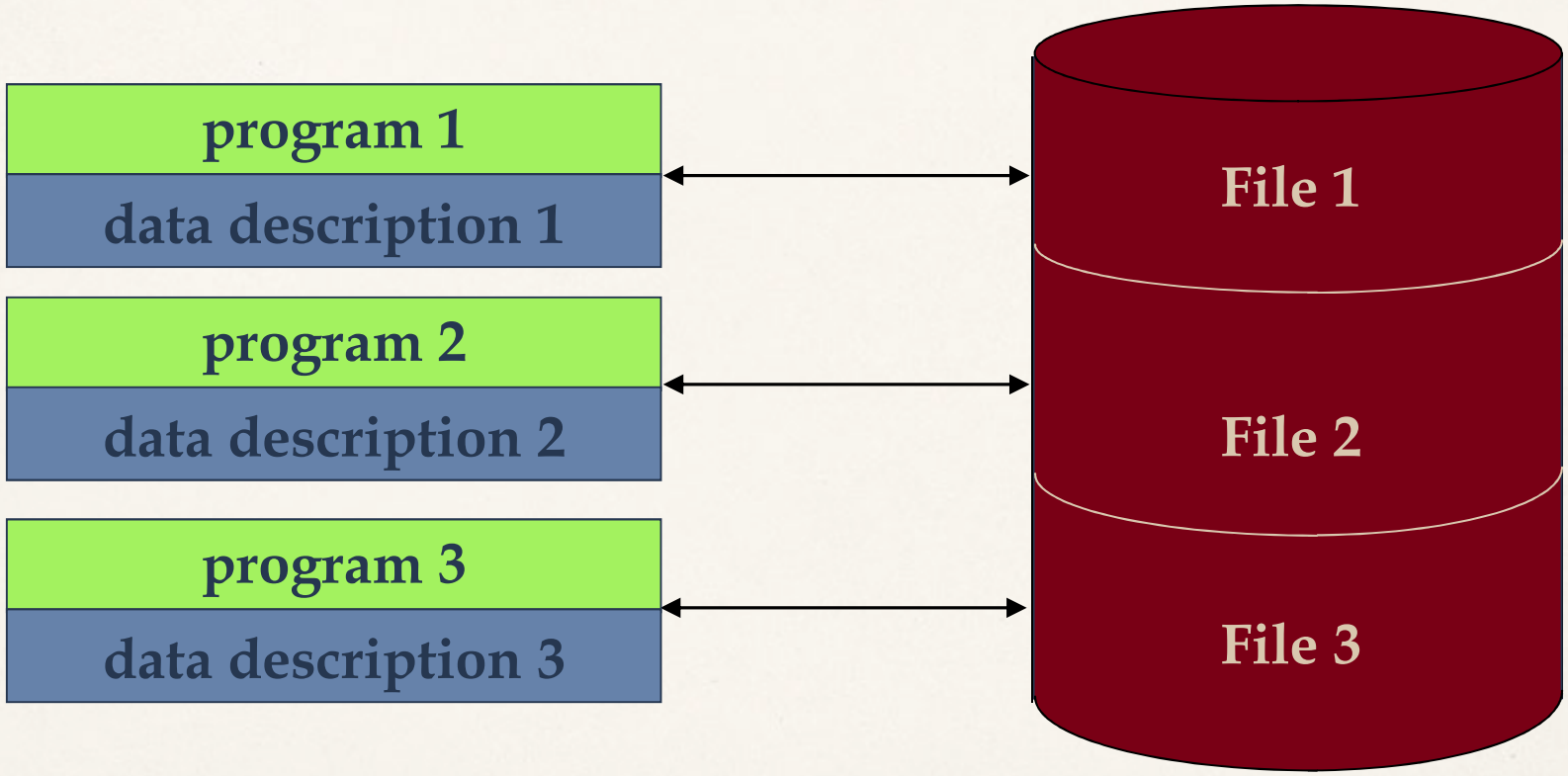
- Introduction (Ch. 1) *
  - Introduction to distributed processing
  - What is (not) a Distributed Database System (DDBS)
  - Data delivery alternatives
  - Promises of DDBS
    - Transparency
    - Reliability (introduction to the problem of distributed transactions)
    - Improved performances
    - Easier system expansion
  - Design issues
  - Classification of distributed data management systems (**information systems**)
    - Dimensions of the classification (autonomy, distribution, heterogeneity)
    - Different distributed data management architectures

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

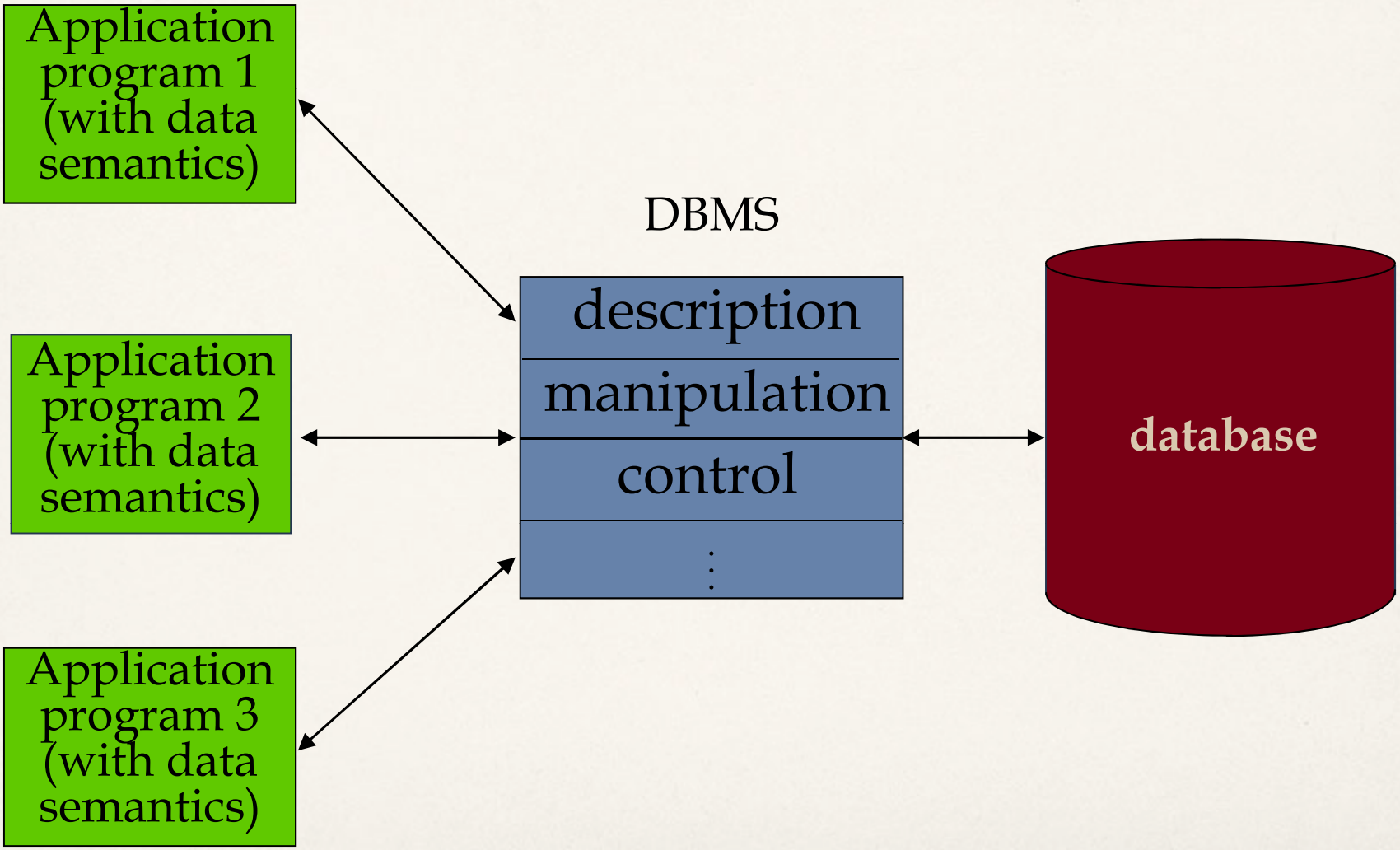
Centralization, distribution, integration

- Database philosophy
  - Separation between application logic and data
  - Centralization (integration) of data
  - Transparency, data independence, access control
- Computer network
  - Distributed applications
  - Distribution of data (big data)
  - Concurrency, redundancy (backup), localization/proximity
- Distributed databases
  - Centralization data is logical
  - Distribution of data is physical
  - Integration

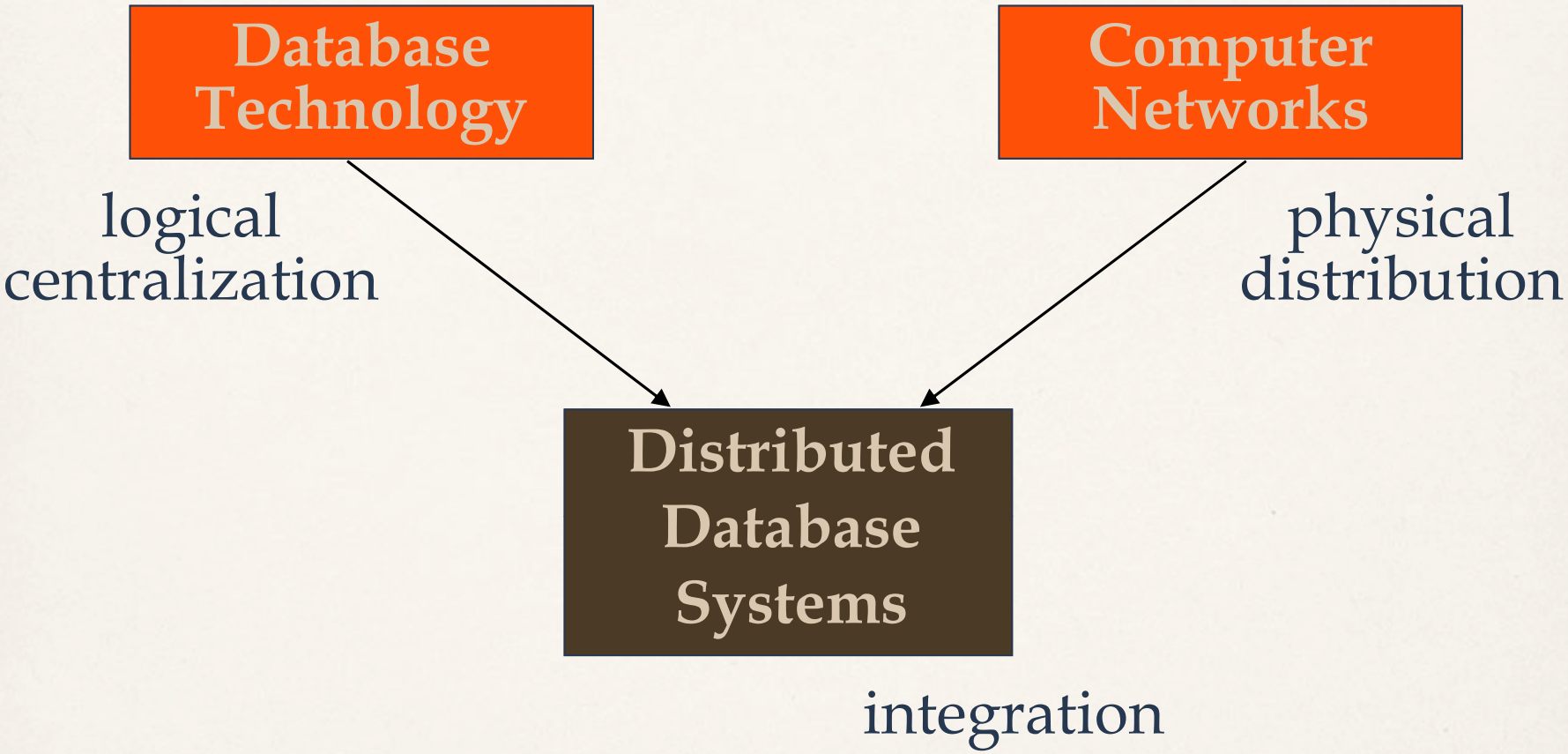
File Systems



Database Management



Motivation



integration ≠ centralization

Distributed Computing

- Some forms of distributed processing are everywhere (CPU vs. I/O, parallel computation, multi-processor, multi-core)
- **Definition (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

Distribution: what and why?

- What is being distributed?
  - Processing logic (processing elements, processing devices)
  - Function (tasks that are specific to a piece of hardware or software)
  - Data
  - Control (supervision, management of tasks)
- Why to distribute?
  - Widely distributed (physically) enterprises
  - Reliability
  - More responsive systems
  - More importantly: nowadays applications are intrinsically distributed and so is the data (web-based, e-commerce, social)
  - In one word (actually two): big data → divide-et-impera (divide-and-conquer)

distr. DB systems : distr. processing = DB systems : centr. processing

What is a Distributed Database System?

A distributed database (DDB) is a collection of multiple, *logically interrelated* databases *distributed over a computer network*.

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

Distributed database system (DDBS) = DDB + D–DBMS

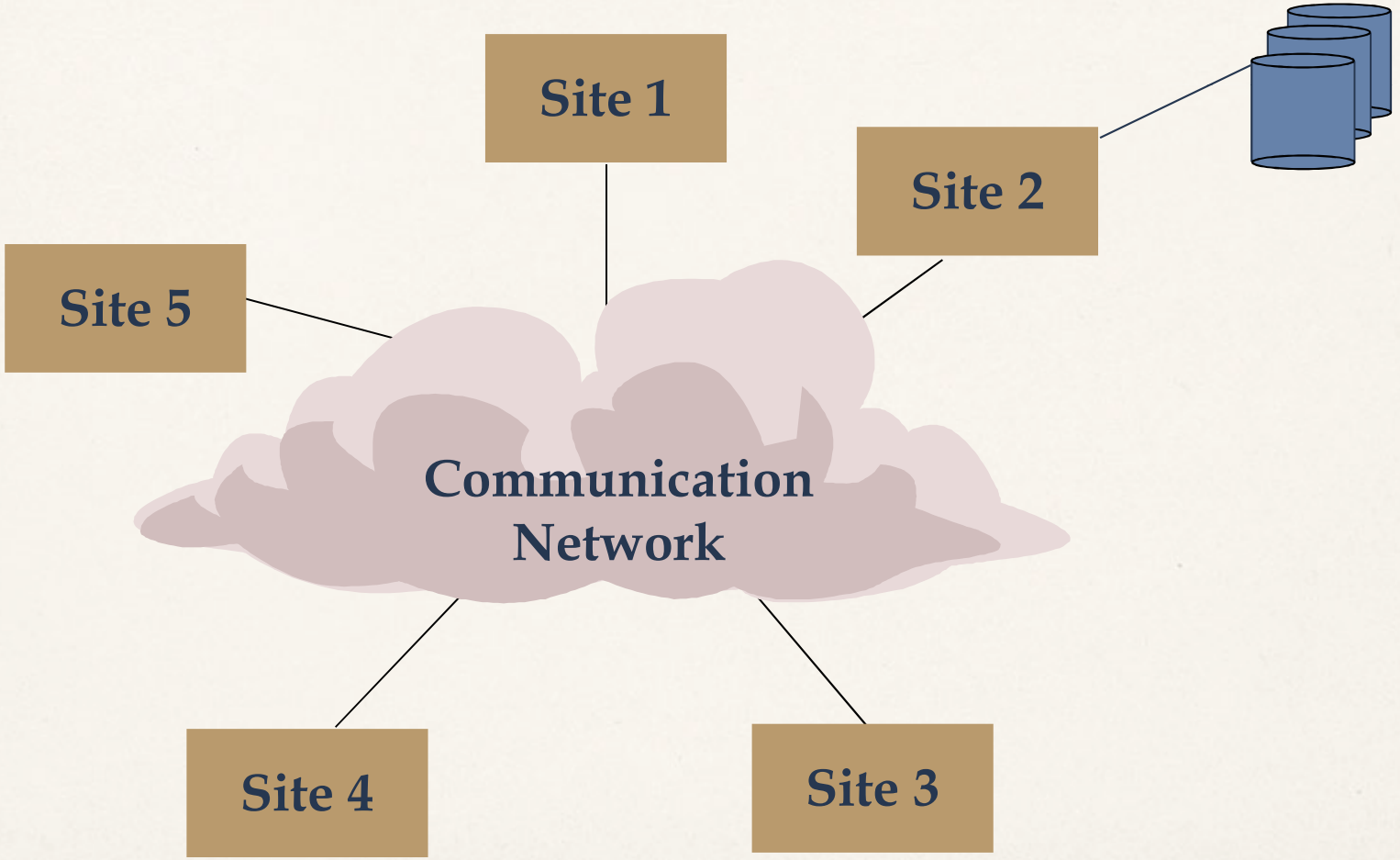
What is not a DDBS?

- Files distributed over a network (missing structure and common logical access interface) → logical interrelation is missing
- A number of related DB that reside on the same system → physical distribution is missing (not necessarily over a wide area, network communication)
- A loosely coupled multiprocessor system → even though communication issues are similar to the one over network (no disk or memory shared), they are not enough heterogeneous (identical processors and OS's)

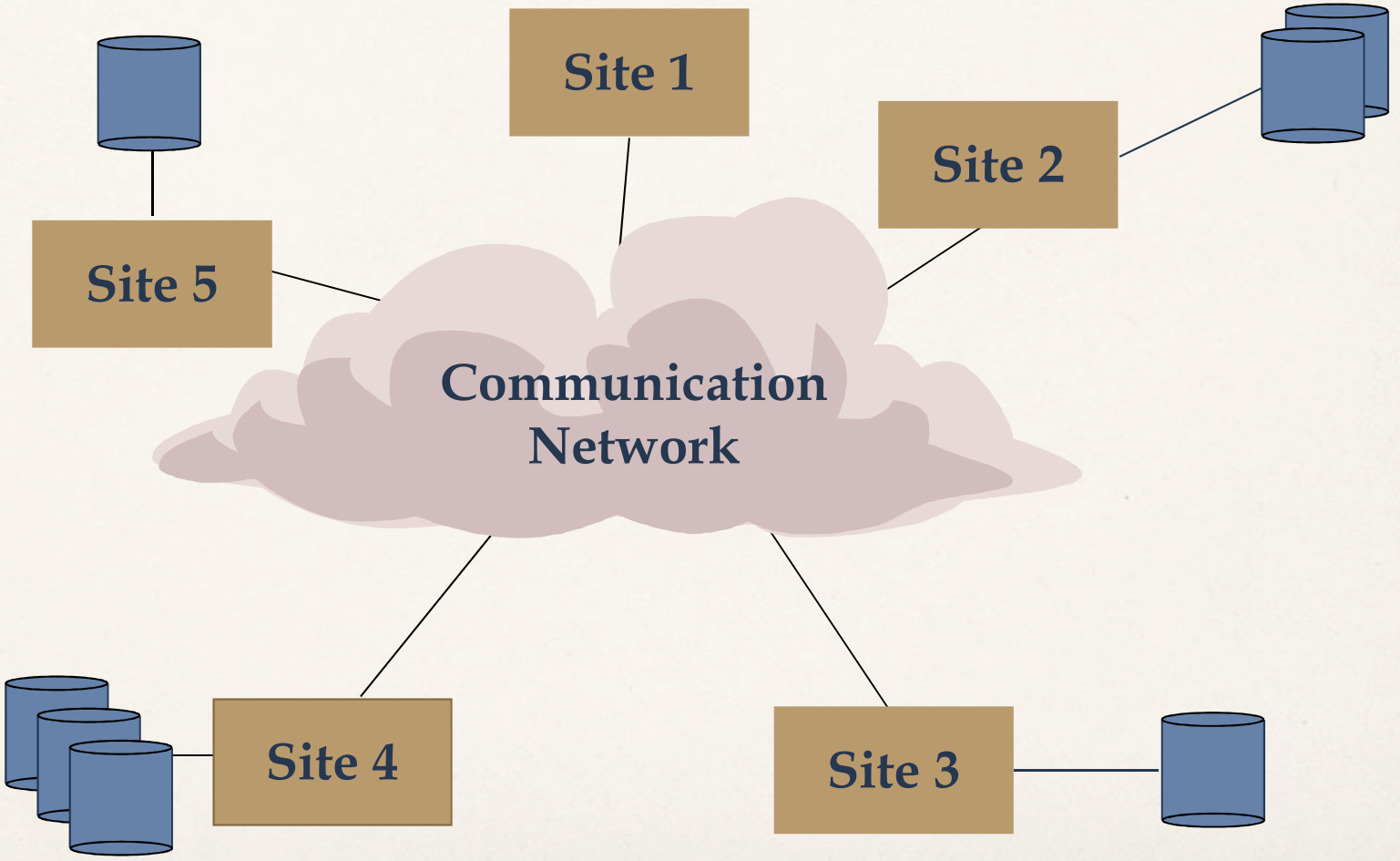
  ➡ these are rather **parallel DB systems**
- A database system which resides at one of the nodes of a network of computers – this is a centralized database on a network node → multiple DB

  ➡ these are rather **client/server DB systems**

Centralized DBMS on a Network



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
- D-DBMS is a full-fledged DBMS
  - ➡ Not remote file system, not a TP system

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
- We focus on pull-only, ad-hoc, unicast communication

Distributed DBMS Promises

1. Transparent management of distributed, fragmented, and replicated data
2. Improved reliability/availability through distributed transactions (distributed transactions implement protocols for, e.g., distributed concurrent access and tolerance to network communication failure)
3. Improved performance
4. 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
- 4 types of transparency
  - Data independence (only form of transparency present also in centralized DBMS)
    - logical VS. physical data independence
  - Network transparency (or distribution transparency)
    - location transparency
    - naming transparency
  - Replication transparency
  - Fragmentation transparency
    - horizontal fragmentation: selection
    - vertical fragmentation: projection
    - hybrid
- Full transparency: tradeoff between ease of use for user and implementation difficulties/costs (poor manageability, poor modularity, poor message performance)

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

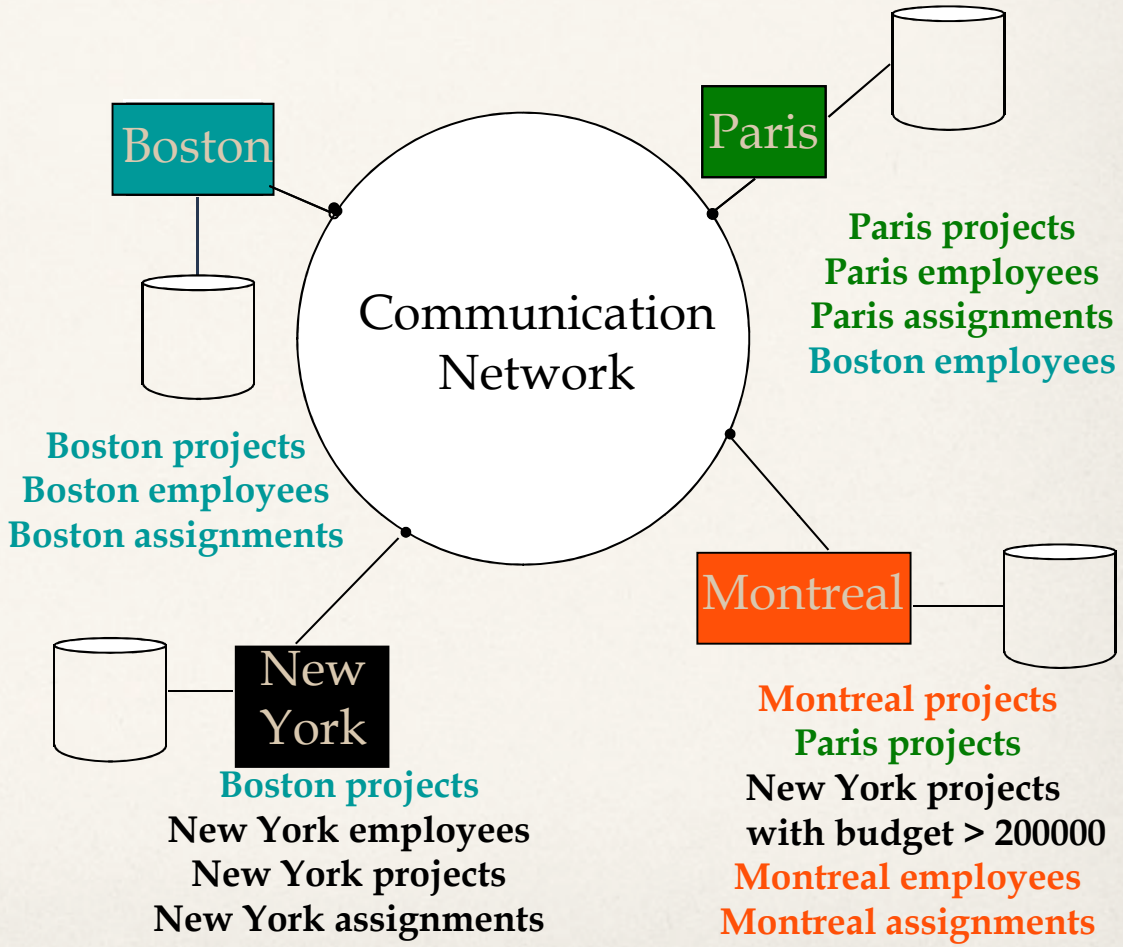
PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

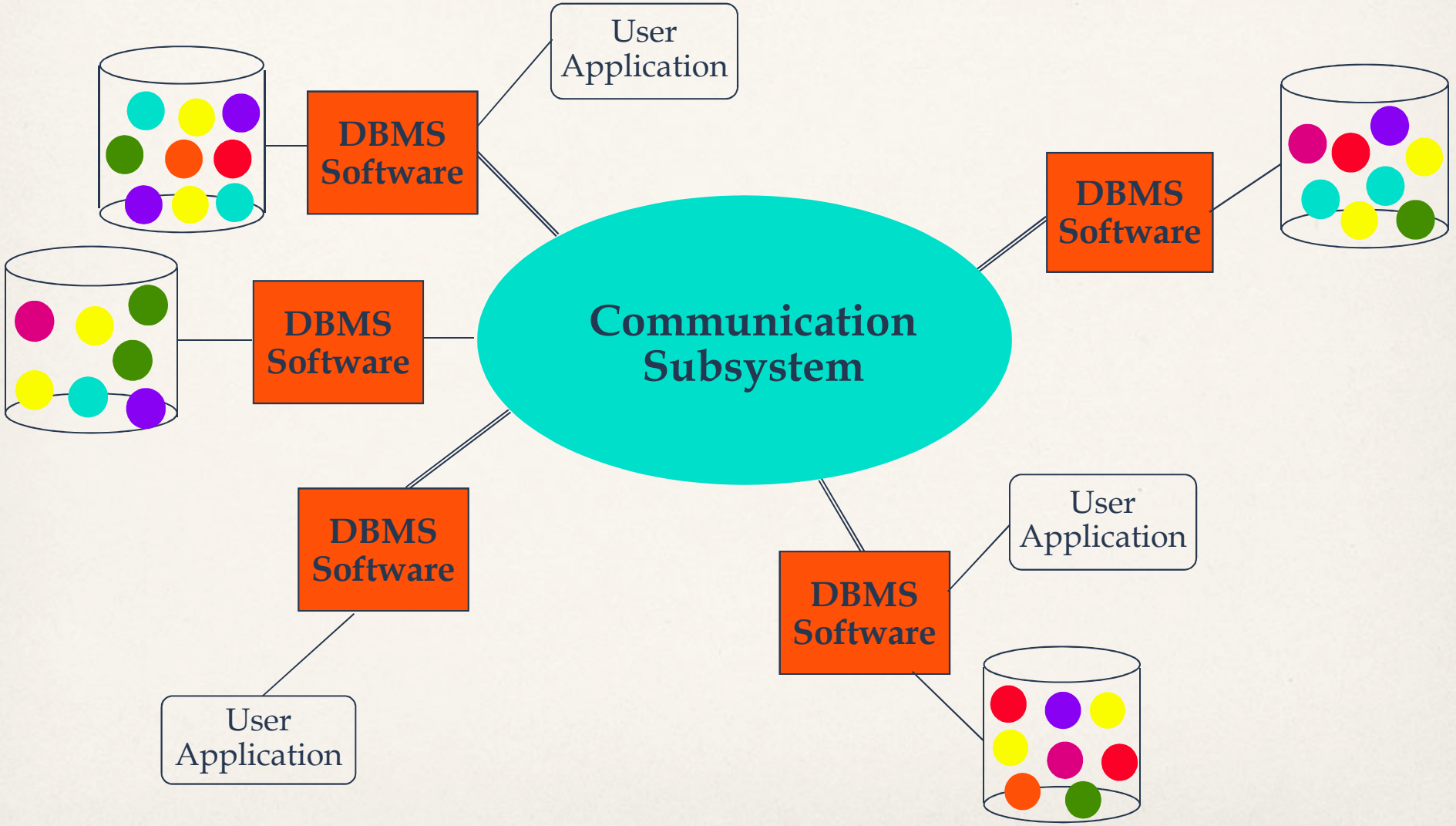
TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Transparent Access

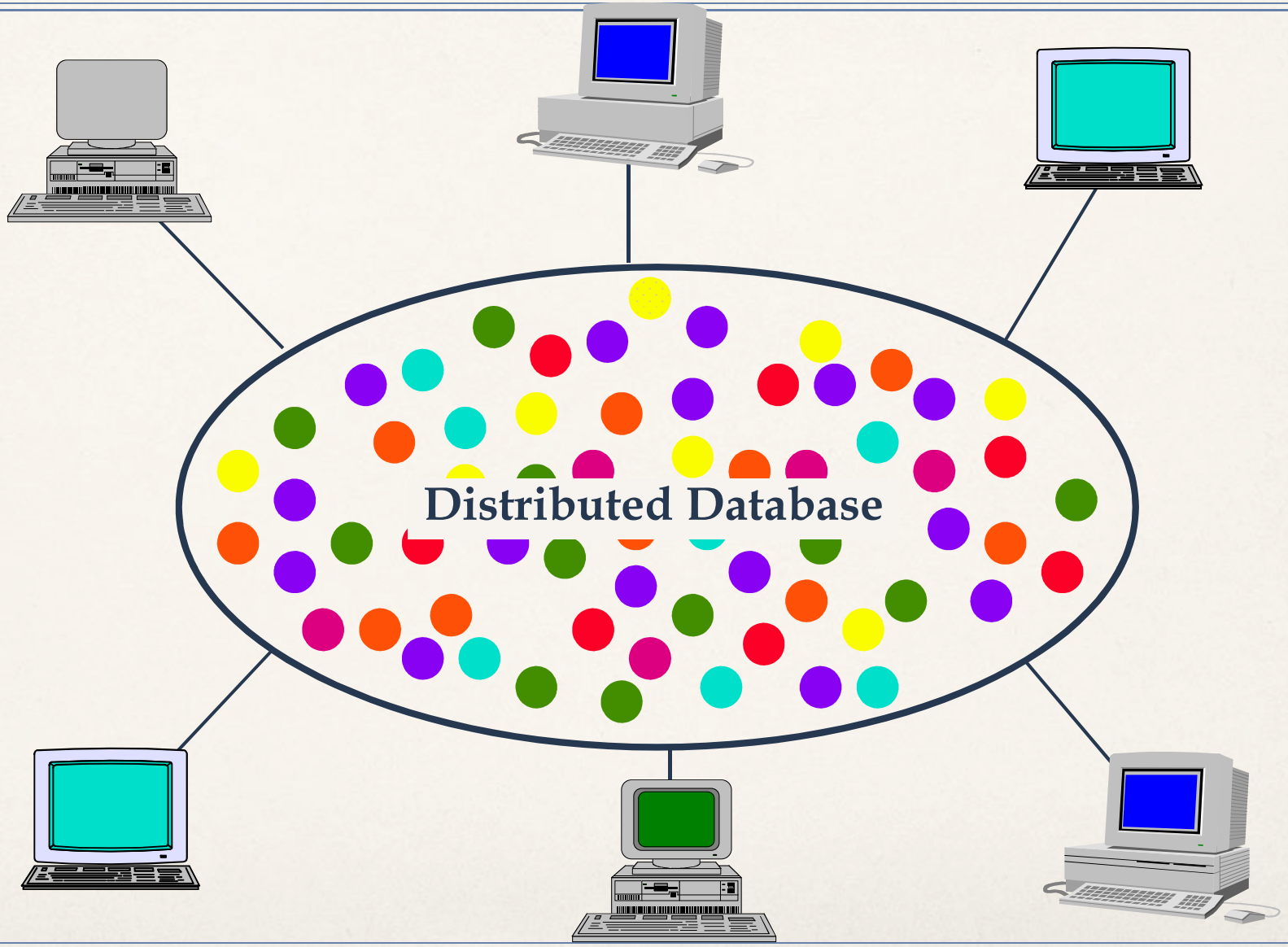
```
SELECT ENAME,SAL
FROM   EMP,ASG,PAY
WHERE  DUR > 12
AND    EMP.ENO = ASG.ENO
AND    PAY.TITLE = EMP.TITLE
```



Distributed DBMS – Reality



Distributed DBMS – User View



Reliability Through Transactions

- Reliability: DB behaves as expected (*consistency*, *availability*, *partition tolerance*)
- Replicated components and data should make distributed DBMS **more reliable** (*no single points of failure*)
- (Distributed) transactions provide *reliability*
  - ➡ bring the DB from a consistent state to another consistent one even in presence of concurrent accesses (*concurrency transparency*) or failures (*failure atomicity*)
- Distributed transaction support requires implementation of
  - ➡ Distributed concurrency control protocols
  - ➡ Distributed reliability protocols (commit and recovery protocols)
- Data replication (we will not deal with it)
  - ➡ Great for read-intensive workloads, problematic for updates
  - ➡ Replication protocols

Potentially Improved Performance

- Proximity of data to its points of use (*data localization*) – thanks to replication and fragmentation
  - ➡ Less contention for CPU and I/O services
  - ➡ Less delay related to network communication
- Distributed systems improve parallelism (inter- and intra-query parallelism) – thanks to replication and fragmentation
- Requires some support for fragmentation and replication (distribution of computation)

Easier System Expansion

- Issue is database scaling
- Expansion: as easy as adding a new node in the network (with its own DBMS)
  - ➡ Easy thanks to transparency
- Expanding a centralized DBMS is more difficult than adding a new one

Complications introduced by Distribution

Distributed systems are intrinsically problematic

- Additional complexity of problems present in the centralized setting
- Additional problems due to distribution
  - ➡ Tolerance to failures of network nodes or links
  - ➡ Synchronization of transactions executed on multiple sites (concurrent access)
  - ➡ Replication (we will not deal with it)
    - ✦ Choosing one of the stored copies
    - ✦ Keep information consistent in all copies
- **CAP Theorem** states that it is impossible for a distributed DB (where data is fragmented and replicated) to provide simultaneously:
  - ➡ Consistency (always the updated data is read)
  - ➡ Availability (every request receives a response)
  - ➡ Partition tolerance (tolerance to network communication failures)

Design Issues

1. **Distributed Database Design (Ch. 3)** *
   - How to distribute the database (data and applications) across the sites
   - (Fully/partially) Replicated & non-replicated database distribution
   - Design issues: *fragmentation* and *allocation* (aka, *data distribution*)
   - Theoretical research: mathematical investigation to solve optimization problems (e.g., minimize cost of storing, processing, and communication)
   - NP-hard: thus, no exact solutions, rather solutions based on heuristics
2. **Distributed Directory Management (Ch. 3)** *
   - Directory contains information (e.g., location) about data
   - Same issues as in **Distributed Database Design**
     - Directory can be global (whole DB) or local (to each site)
     - Centralized (at one site) or distributed (over several sites)
     - Single copy or multiple copies

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Design Issues (2)

3. **Distributed Query Processing (Ch. 6-8)** *
   - ➡ Convert user transactions to data manipulation instructions
   - ➡ Finding the most efficient way to execute a query over the network
     - ✦ Split a global query (over global relations) into partial queries (over fragments)
   - ➡ Optimization problem
     - ✦ min{cost = data transmission + local processing}
   - ➡ NP-hard: thus, no exact solutions, rather solutions based on heuristics

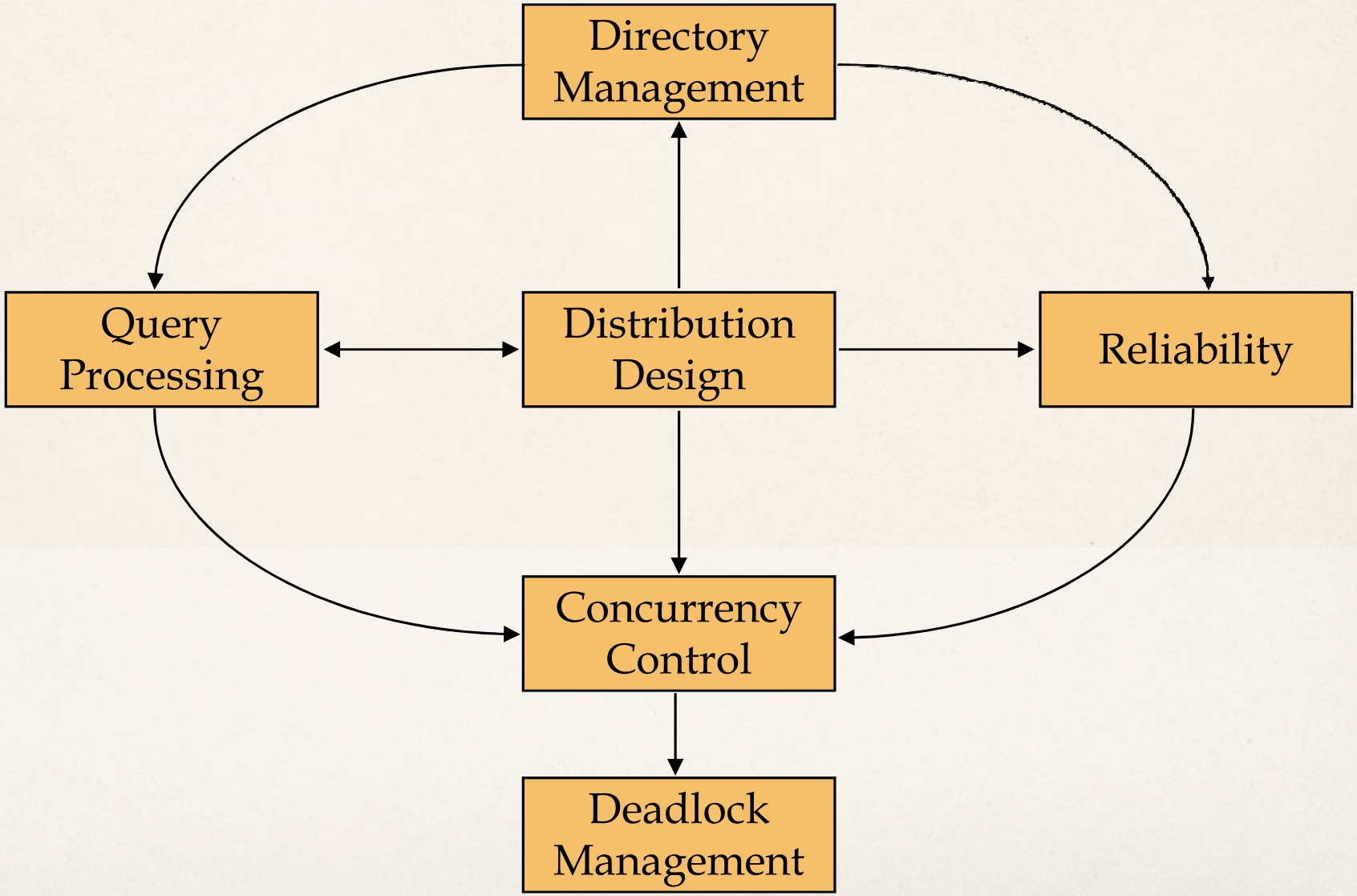
* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Design Issues (3)

4. **Distributed Concurrency Control (Ch. 11)** *
   - ➡ Synchronization of concurrent accesses
   - ➡ Consistency and isolation
   - ➡ Locking and timestamp-based techniques
5. **Distributed Deadlock Management (Ch. 11)** *
   - ➡ Concurrency control mechanisms based on locking may cause deadlocks (as in OS's)
   - ➡ Prevention, avoidance, detection/recovery techniques
6. **Reliability of Distributed DBMS (Ch. 12)** *
   - ➡ How to make the system resilient to failures
   - ➡ Atomicity and durability
7. **Replication (Ch. 13)** *
   - ➡ Consistency of the information (eager and lazy protocols)

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Relationship Among Issues

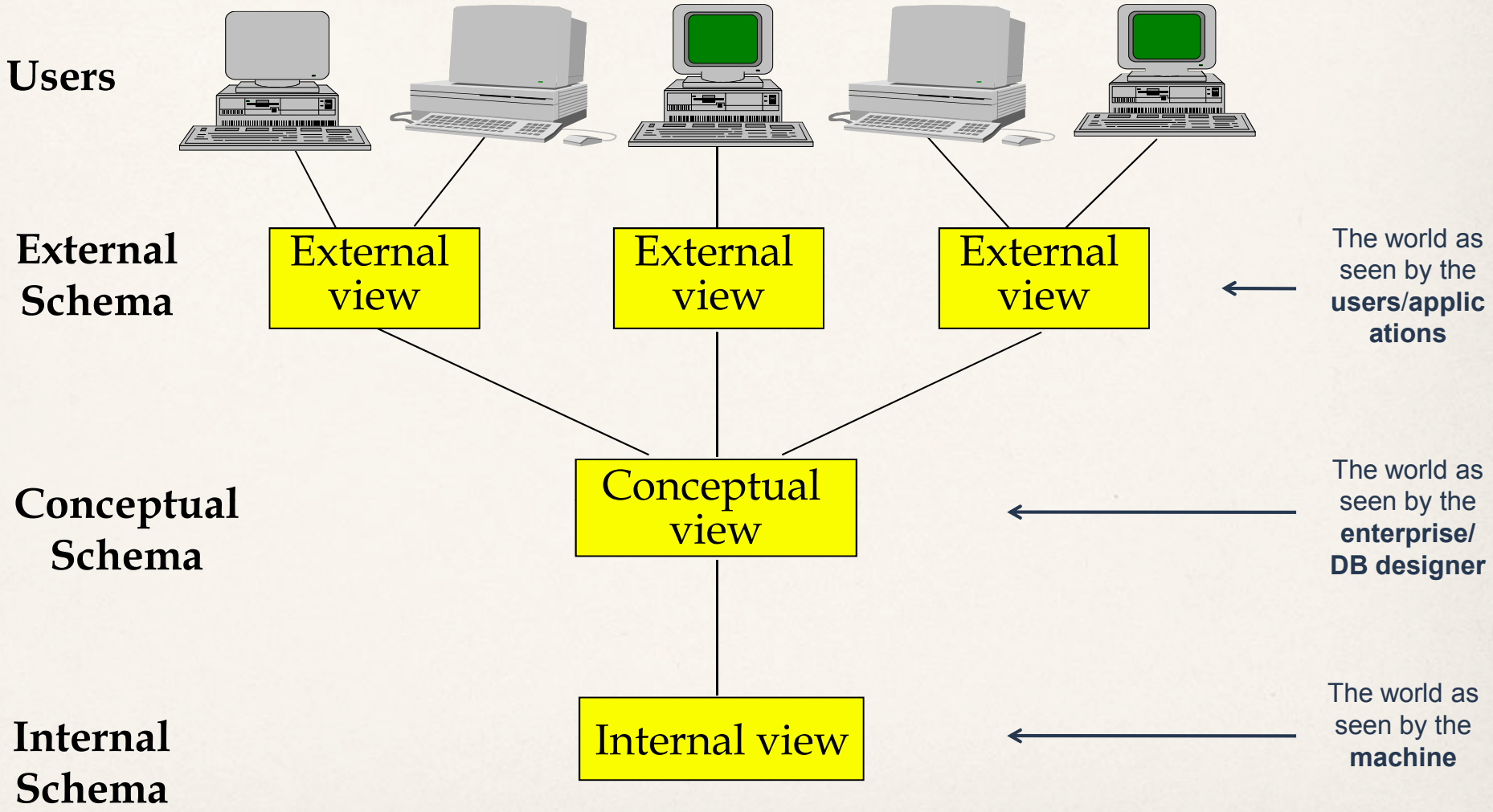


© M. T. Özsu & P. Valduriez

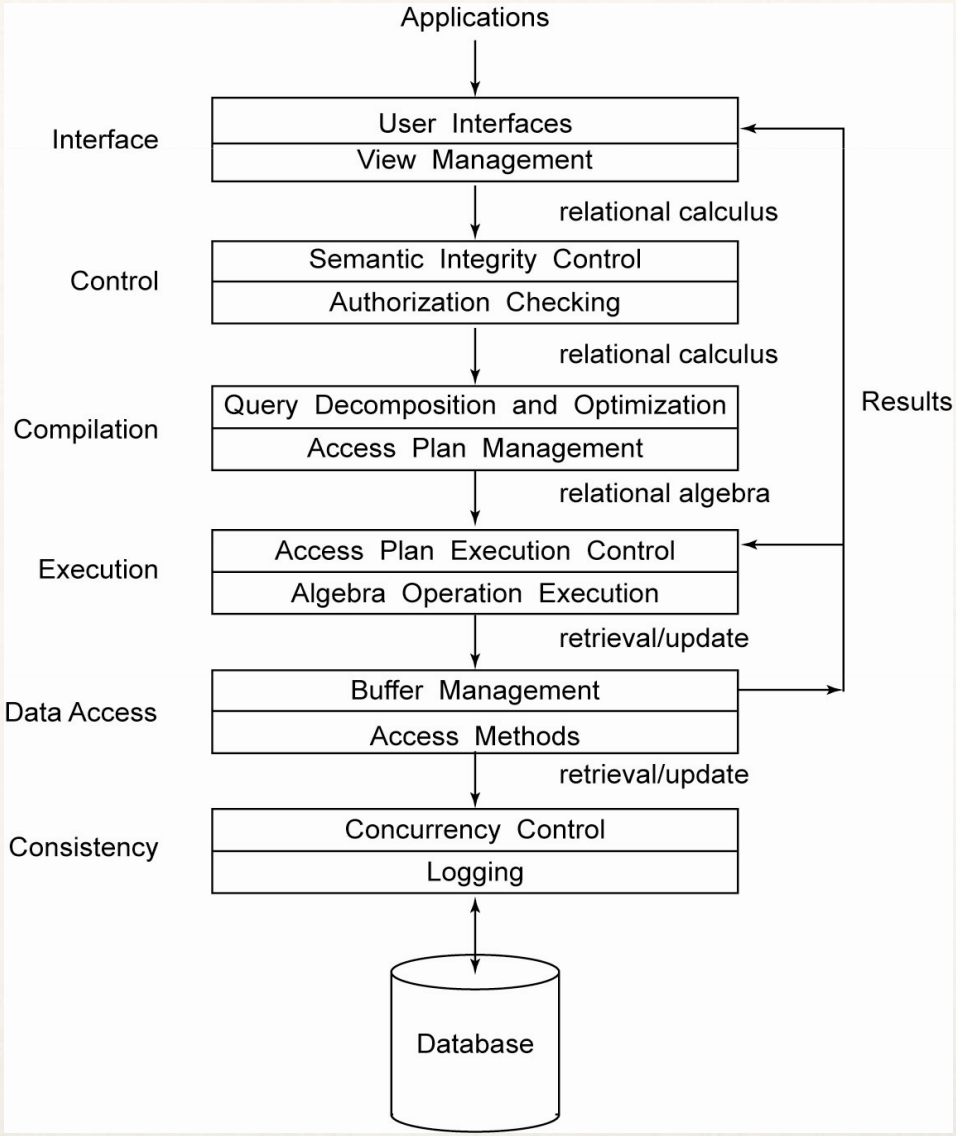
Architecture

- Defines the structure of the system
  - components
  - functions of each component
  - interrelationships and interactions between components
- First standard: ANSI/SPARC Architecture (70s)
- Reference architectures for distributed data management
  - client/server
  - peer-to-peer (classic and modern) – in the terminology used in the textbook classic peer-to-peer systems correspond to Distributed DBMS we deal with in this course
  - multidatabase systems (MDBS)

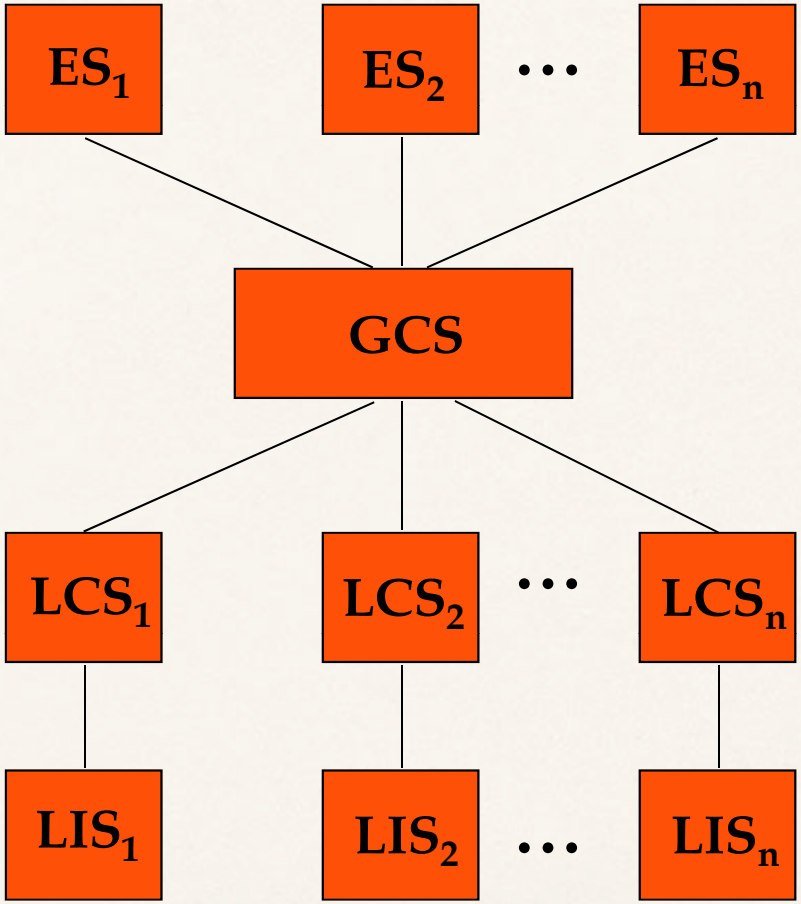
ANSI/SPARC Standard



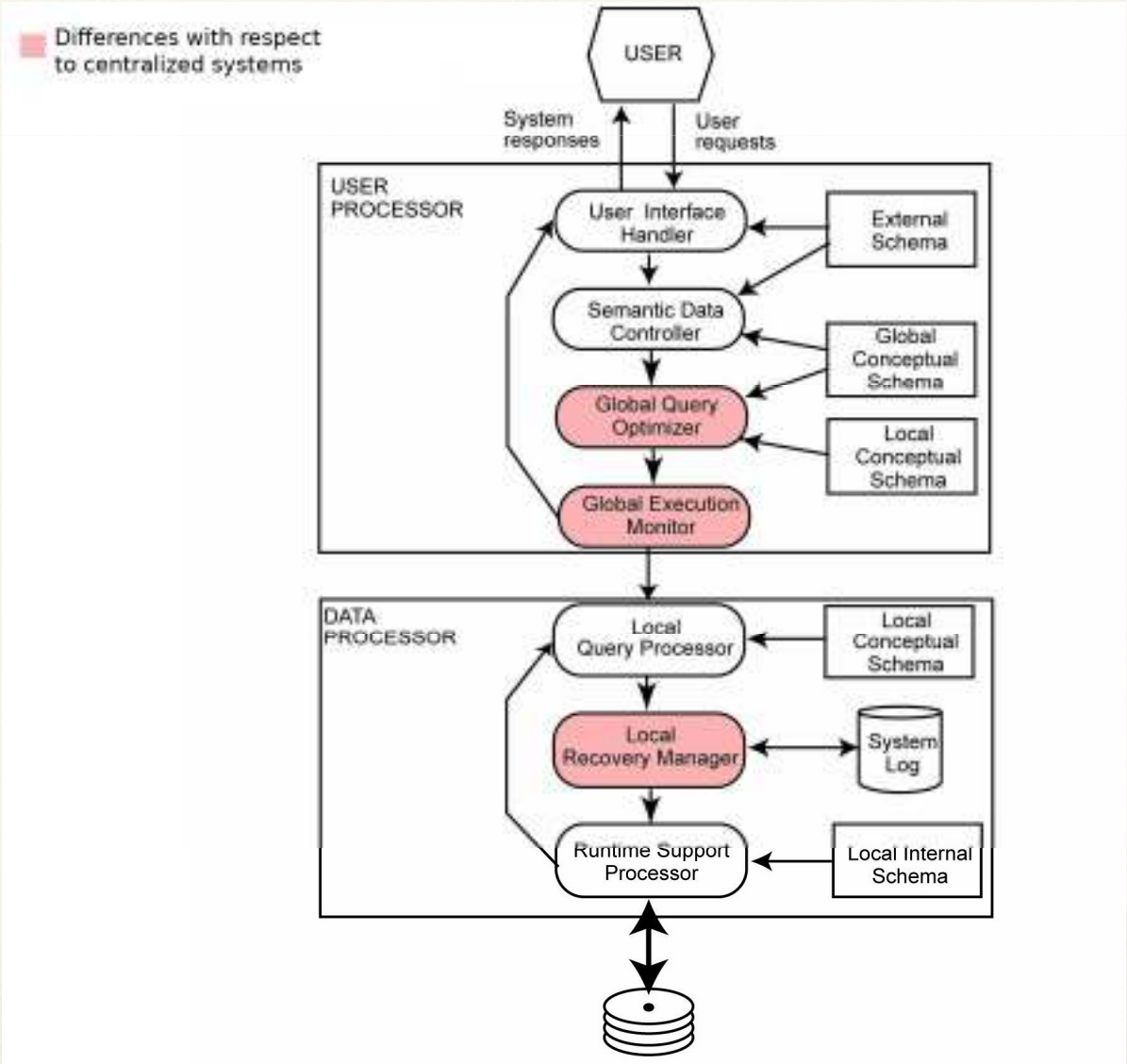
Generic DBMS Architecture



DDBS Data Model



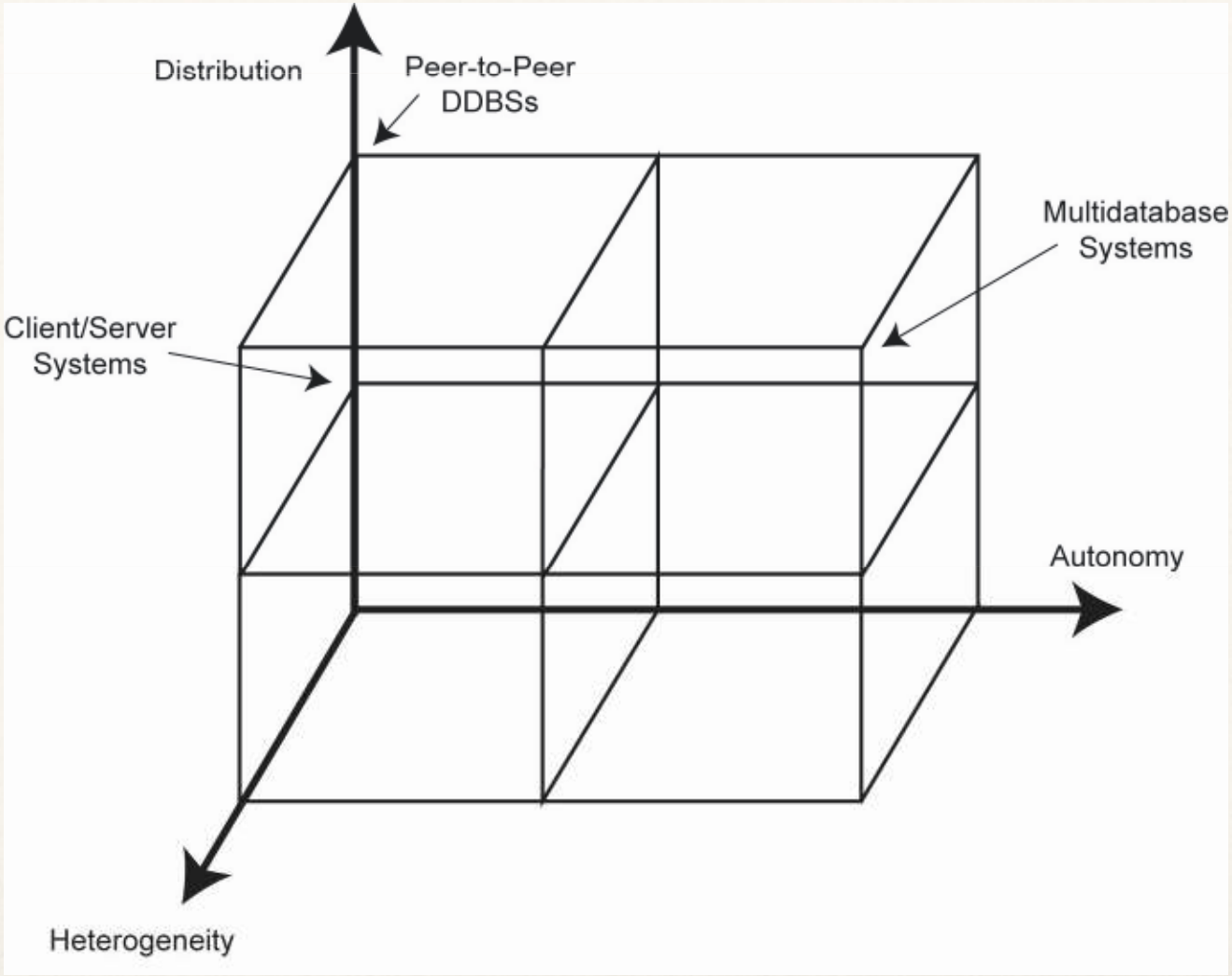
Distributed DBMS Architecture



A classification of data management systems

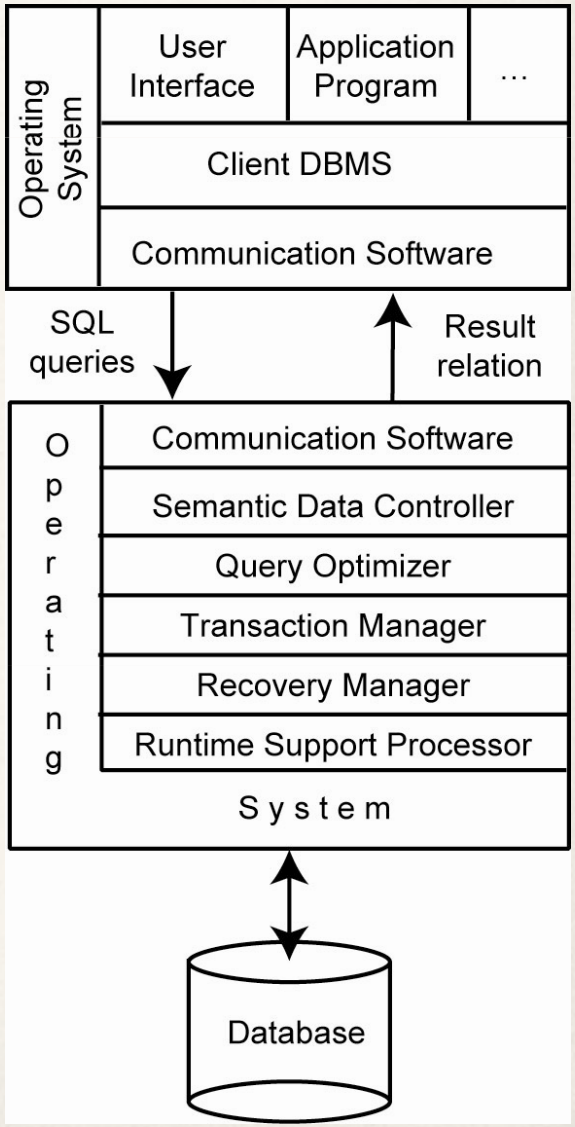
- Distribution (of data and control) – *non-distributed* **VS.** *client-server* **VS.** ***fully distributed***
  - Degree of distribution of data over multiple machines (sites)
- Heterogeneity – ***homogeneous*** **VS.** *heterogeneous*
  - **data model:** different expressive power
  - **query language:** different languages, SQL comes in different implementations
  - **transaction management:** different protocols and policies
- Autonomy (degree of synchronization) – ***tight integration*** **VS.** *semiautonomous* **VS.** *total isolation*
  - Degree to which single DBMS can operate independently
    - Design autonomy: ability to decide on design issues (data models, transaction management, …)
    - Communication autonomy: ability to decide what information to share and how to communicate
    - Execution autonomy: ability to execute transactions in any manner
  - Levels of autonomy
    - Tight integration (one unique logical view of the DB, data manager (DM) do not operate independently, they collaborate to execute user request)
    - Semiautonomous (DM act independently, selective data sharing, common communication protocols)
    - Total isolation (*multidatabase systems* – totally independent, unaware of each other and of communication systems, no global control, no unified DB view, each DB decides what to share with who)

DBMS Implementation Alternatives



Client/Server Architecture

- Nodes of the network are not identical (divided into two groups: clients and servers)
- DBMS is on server(s) only
- Most of the work is done by server(s)
- On clients: only **user interface** and a **DBMS client** for *sending* queries and *receiving/displaying/caching* results (possibly caching transaction locks)
- Not fully distributed
  - ➡ *multiple client/single server*: like centralized DBMS (no distribution-related issues)
  - ➡ *multiple client/multiple servers*: data and control are distributed among few servers (mild form of distribution)
- Our discussion mainly refers to fully distributed DBMS, but often applies to client/server architectures

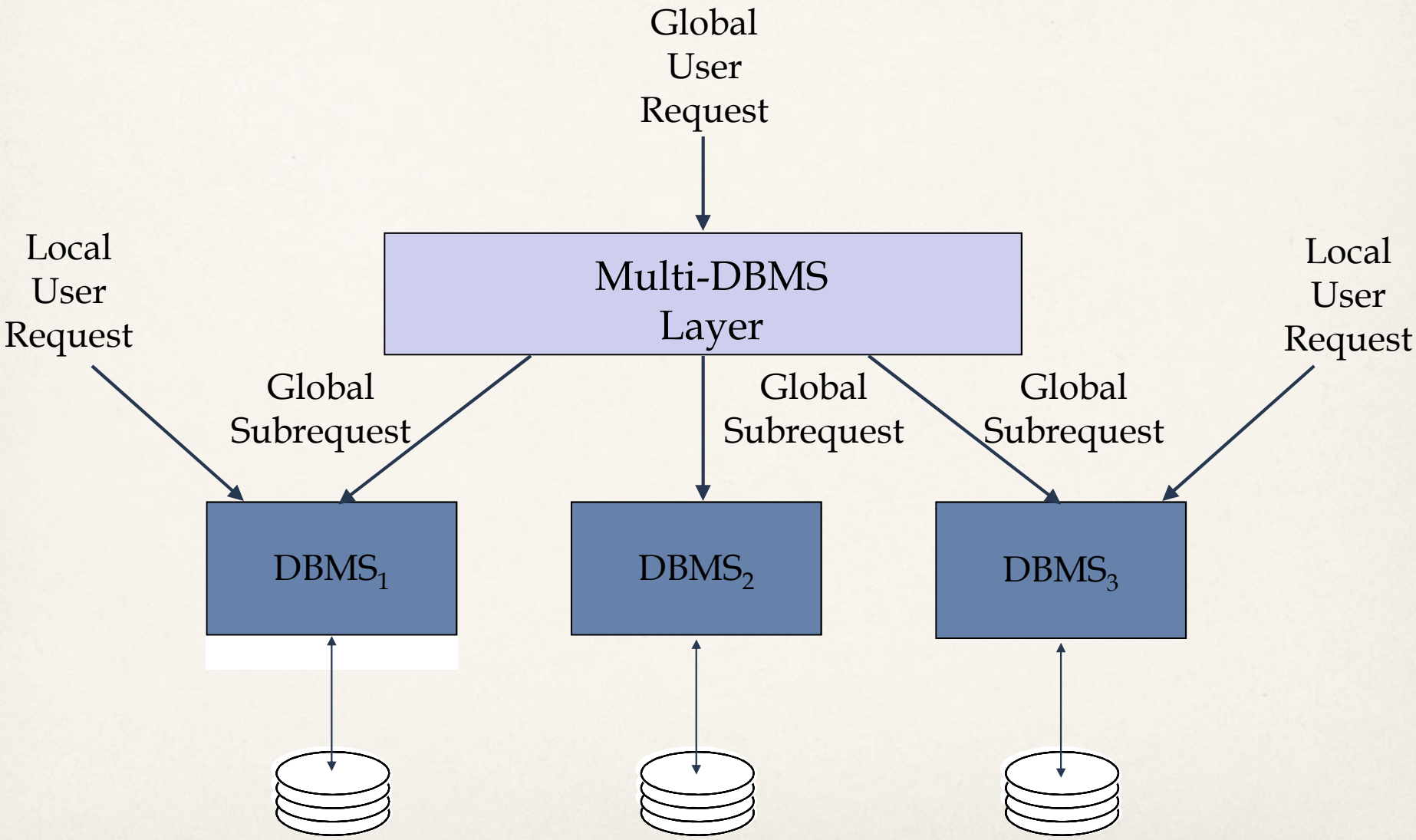


Multidatabase system architecture

- Multidatabase system (MDBS) are beyond the scope of this course
- In a nutshell
  - Characterized by presence of several fully autonomous, independent, and heterogeneous DBMS that are located apart from each other
  - Unaware of each other existence
  - Systems do not know how to communicate with other ones
  - No (explicit) interrelation between DBMS
  - Each DBMS provides an **export schema** to make (some of) its data available to (some of) the other systems
  - Control and coordination is implemented by a **multidatabase layer** which uses such export schema and provide interface (transparency) to the users
  - In DDBS: unique conceptual view of the entire DB (GCS, providing transparency) results from the "union" local conceptual schemas (LCS) of parts of DB stored locally at each node of the network
    In MDBS: not even a clear notion of "entire DB"
  - Architecture design is built bottom-up, from LCS to a GCS (in DDBS it is built top-down, opposite direction)
- If interested, see the discussions at Ch. 1.10, 4, 9 ⋆

⋆ Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

MDBS Components & Execution



© M. T. Özsu & P. Valduriez

Recent Developments

- So far, and in the rest of the course: focus on "traditional" DDBS
- Recent directions
  - ➡ **Multidatabase systems**, aka federated databases, aka data integration systems – *not in this course (Ch. 1.7.10, 4, 9)* *
    - ✦ Heterogeneity of data source
  - ➡ **Peer-to-peer** "reloaded" **and web data management** – *not in this course (Ch. 16, 17)* *
    - ✦ Rethinking of an old architecture, oriented to needs of data sharing over the web
  - ➡ **Parallel databases** – *not in this course (Ch. 14)* *
    - ✦ Specific issues (not really distributed)

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Terminology disambiguation

- In our discussion, by distributed DBMS and DDBS we mostly refer to fully distributed, **peer-to-peer** data management systems
- In particular, the term "peer-to-peer" (like many terms, e.g., client/server, …) is overloaded in the book and in the literature. It is used to refer to:
  - ➡ "traditional" peer-to-peer DBMS (also referred to as fully distributed DBMS) → our focus is on these systems
  - ➡ "modern" peer-to-peer data management systems *[we refer to this meaning in the slide titled "Recent developments"]*
    - ✦ These refer to an evolution of the traditional peer-to-peer architecture, to cope with the need of data sharing, e.g., over the web
    - ✦ Ongoing research area, related issues still being investigated
    - ✦ Out of the scope of this course (see Ch. 16) *

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011