

1/30 8:26:42 * * *



Distributed DBs

what is distributed?

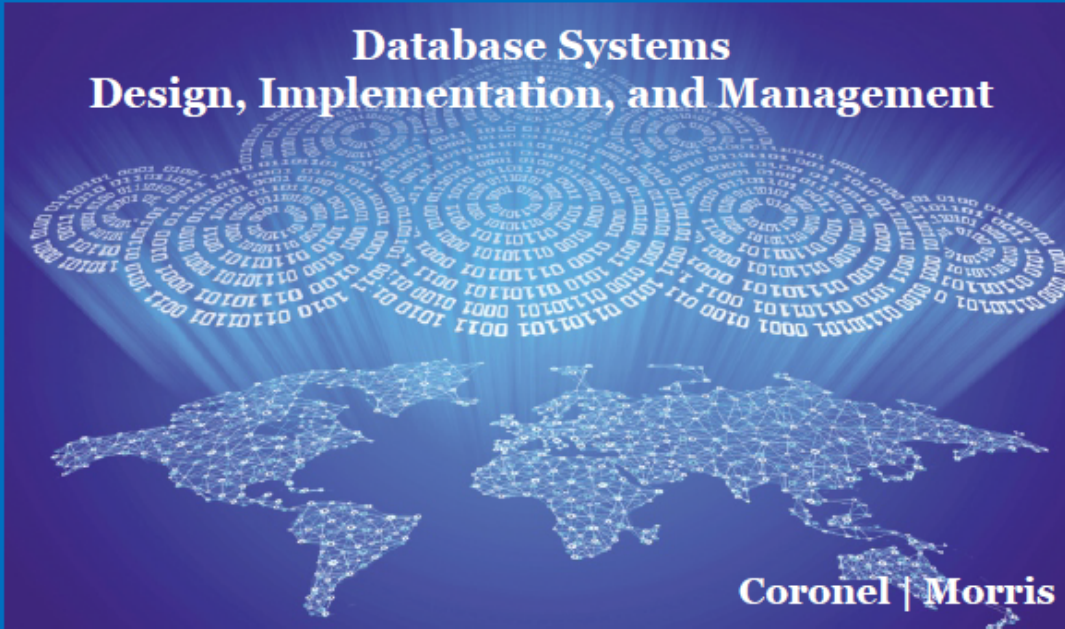
how are transactions managed?

what is the architecture?

Ch.12

11e

Database Systems Design, Implementation, and Management

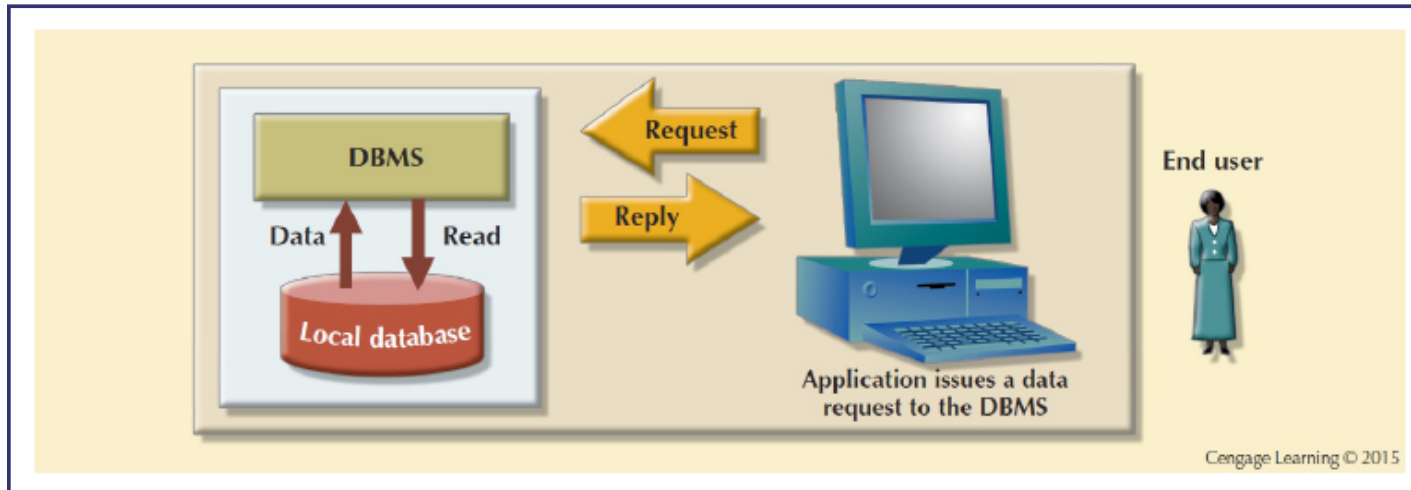


Chapter 12

Distributed Database Management Systems

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

'Centralized' DBs - no longer popular/useful



Factors Affecting the Centralized Database Systems

- Globalization of business operation
- Advancement of web-based services
- Rapid growth of social and network technologies
- Digitization resulting in multiple types of data
- Innovative business intelligence through analysis of data

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

6

Two factors in particular, necessitated change:

- rapid, ad-hoc access to data was needed ['Internet speed' decision-making]
- distributed data access was needed, to serve dispersed business units [globalization]

So now we have **DDBMSs** - **distributed** DBMSs.

Desirability of Distributed DBMS Over Centralized DBMS

Performance
degradation

High costs

Reliability
problems

Scalability
problems

Organizational
rigidity

Distributed DBs - almost ALL (web-based)!

Evolution Database Management Systems

- **Distributed database management system (DDBMS):** Governs storage and processing of logically related data over interconnected computer systems
 - Data and processing functions are distributed among several sites
- **Centralized database management system**
 - Required that corporate data be stored in a single central site
 - Data access provided through dumb terminals

Factors That Aided DDBMS to Cope With Technological Advancement

Acceptance of Internet as a platform for business

Mobile wireless revolution

Usage of application as a service

Focus on mobile business intelligence

Advantages and Disadvantages of DDBMS

Advantages

- Data are located near greatest demand site
- Faster data access and processing
- Growth facilitation
- Improved communications
- Reduced operating costs
- User-friendly interface
- Less danger of a single-point failure
- Processor independence

Disadvantages

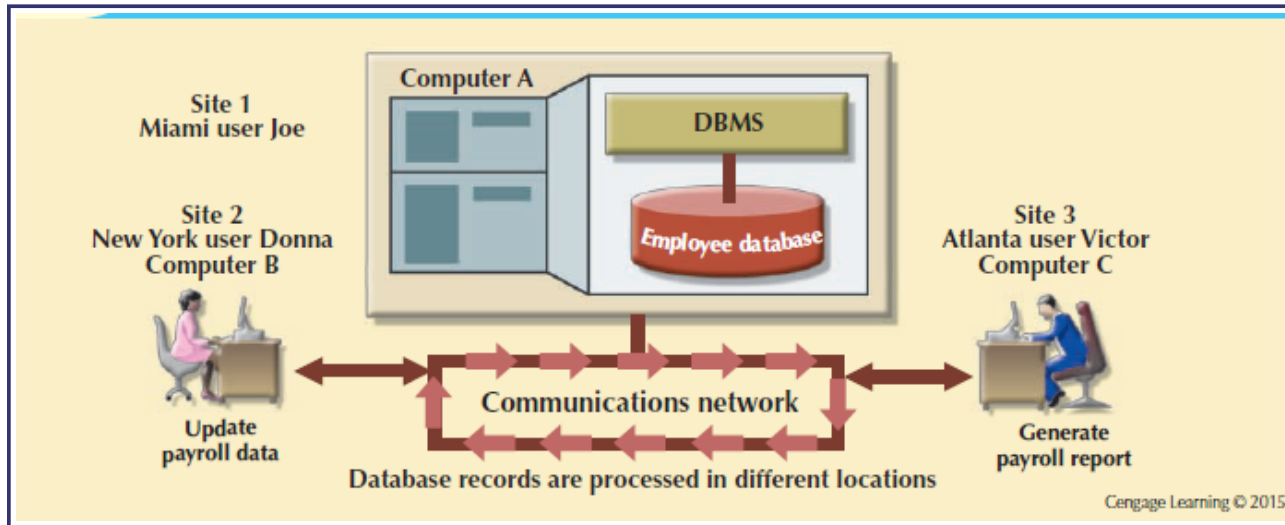
- Complexity of management and control
- Technological difficulty
- Security
- Lack of standards
- Increased storage and infrastructure requirements
- Increased training cost
- Costs incurred due to the requirement of duplicated infrastructure

Distributed *processing* vs distributed *databases*

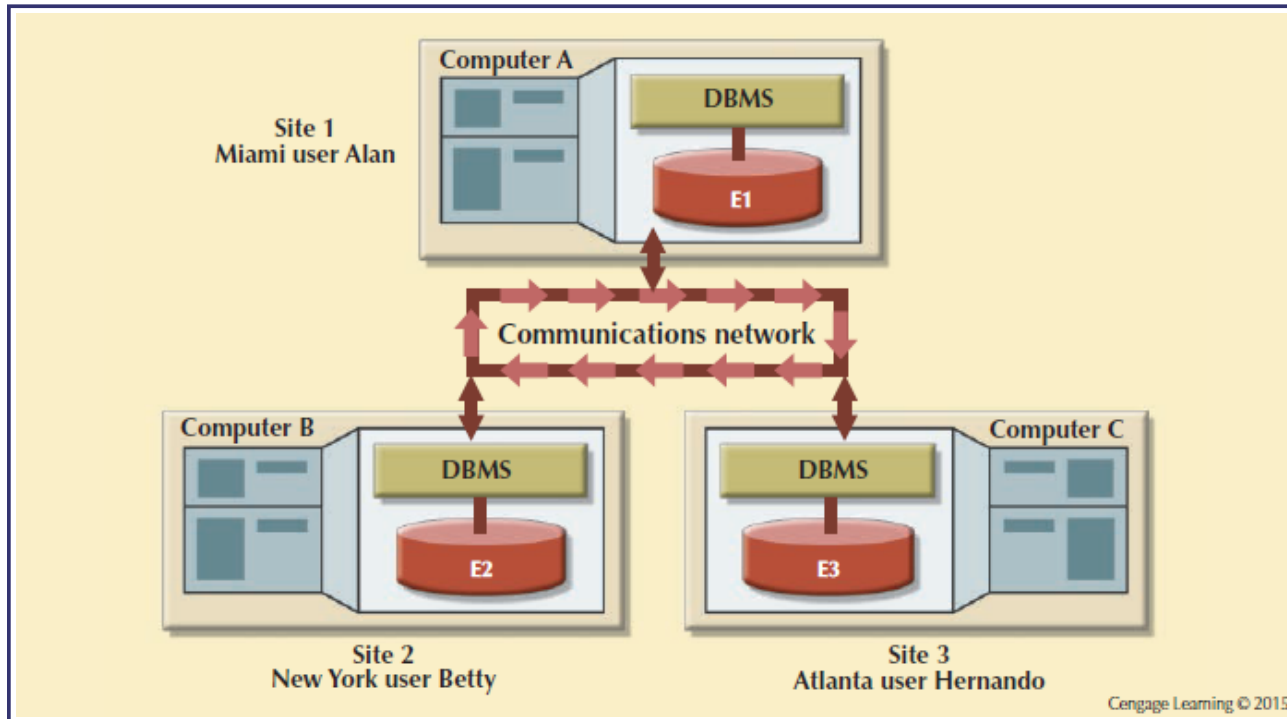
Distributed Processing and Distributed Databases

- **Distributed processing:** Database's logical processing is shared among two or more physically independent sites via network
- **Distributed database:** Stores logically related database over two or more physically independent sites via computer network
- **Database fragments:** Database composed of many parts in distributed database system

Distributed *processing*



Distributed *databases*

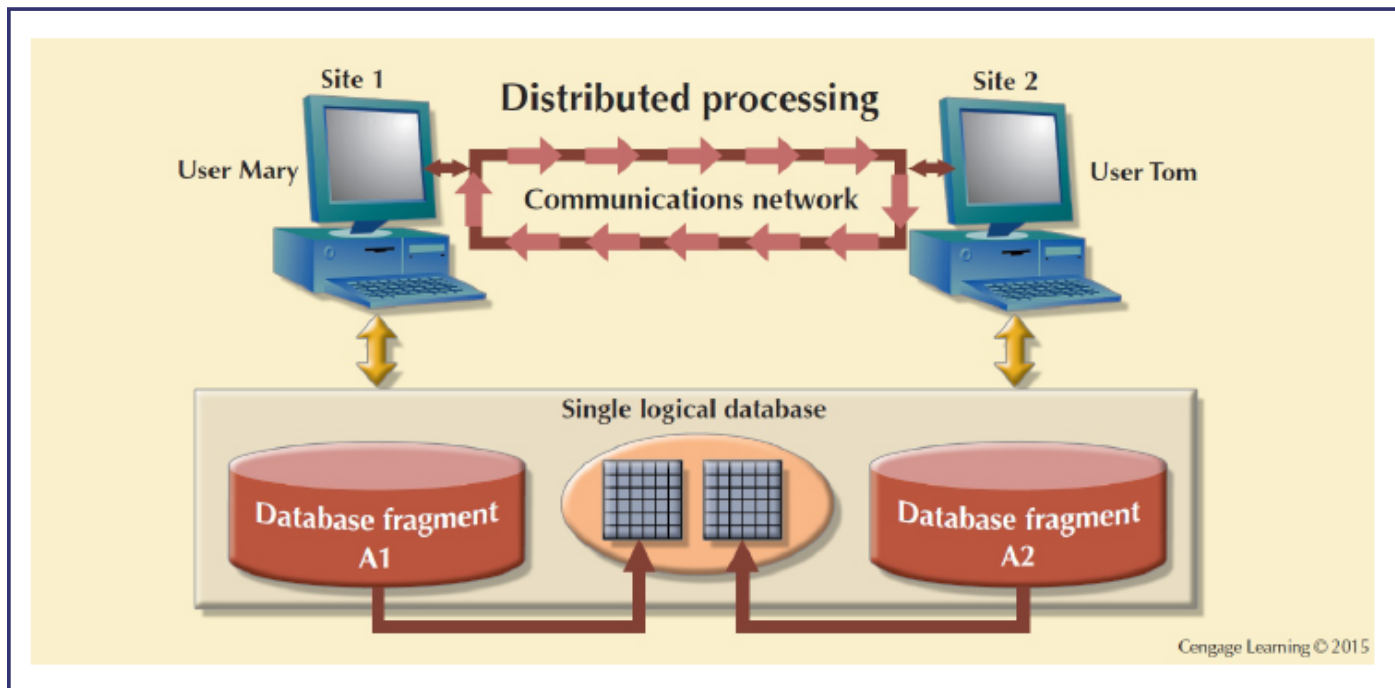


Fully distributed DBMSs: functions

Distributed processing AND data storage -> 'fully' distributed.

Functions of Distributed DBMS

- Receives the request of an application
- **Validates analyzes**, and decomposes the request
- Maps the request
- Decomposes request into several I/O operations
- Searches and validates data
- Ensures consistency, security, and integrity
- Validates data for specific conditions
- Presents data in required format

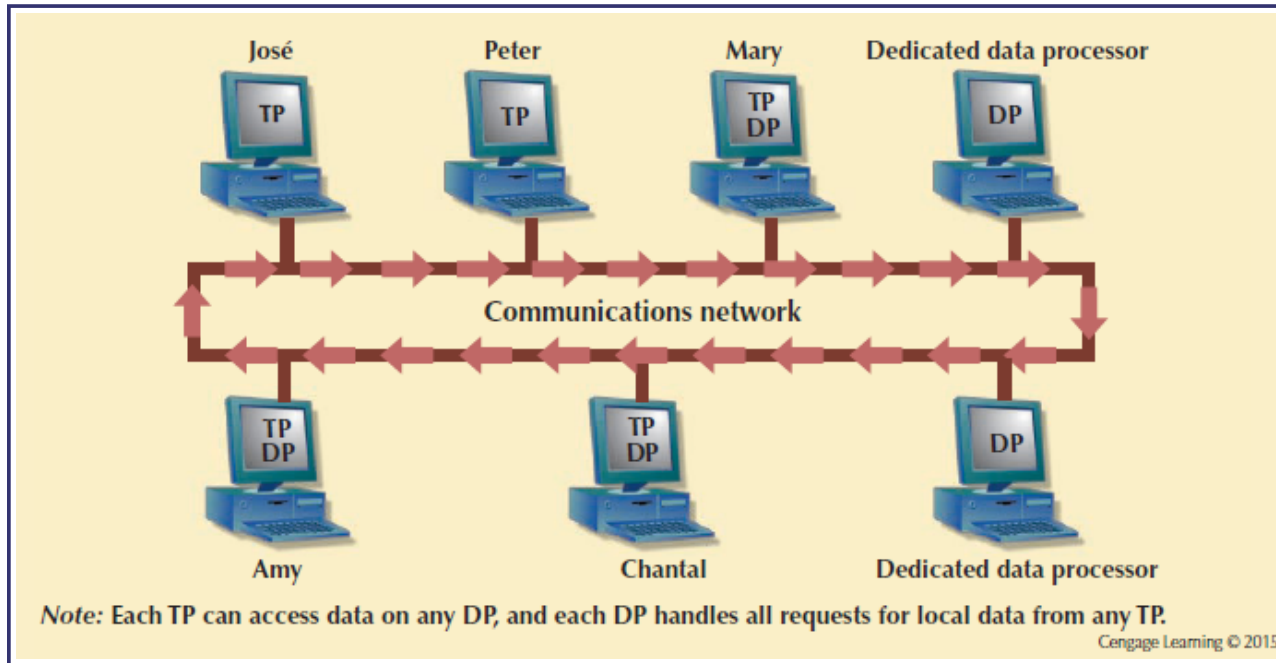


Two fragments, two sites - but each user thinks they have a single (their own) local version (transparent access).

TP(TM) vs DP(DM)

DDBMS Components

- Computer workstations or remote devices
- Network hardware and software components
- Communications media
- **Transaction processor (TP):** Software component of a system that requests data
- Known as **transaction manager (TM)** or **application processor (AP)**
- **Data processor (DP)** or **data manager (DM)**
- **Software** component on a system that stores and retrieves data from its location



A set of protocols is used by the DDBMS, to enable the TPs and DPs to communicate with each other.

TP: Transaction Processor - receives data requests (from an application), and requests data (from a DP); also known as AP (Application Processor) or TM (Transaction Manager). A TP is what fulfills data requests on behalf of a transaction.

DP: Data Processor - receives data requests from a TP (in general, multiple TPs can request data from a single DP), retrieves and returns the requested data; also known as DM (Data Manager).

Data and processing distribution: 3 variations

12.6 LEVELS OF DATA AND PROCESS DISTRIBUTION

Current database systems can be classified on the basis of how process distribution and data distribution are supported. For example, a DBMS may store data in a single site (using a centralized DB) or in multiple sites (using a distributed DB), and may support data processing at one or more sites. Table 12.2 uses a simple matrix to classify database systems according to data and process distribution. These types of processes are discussed in the sections that follow.

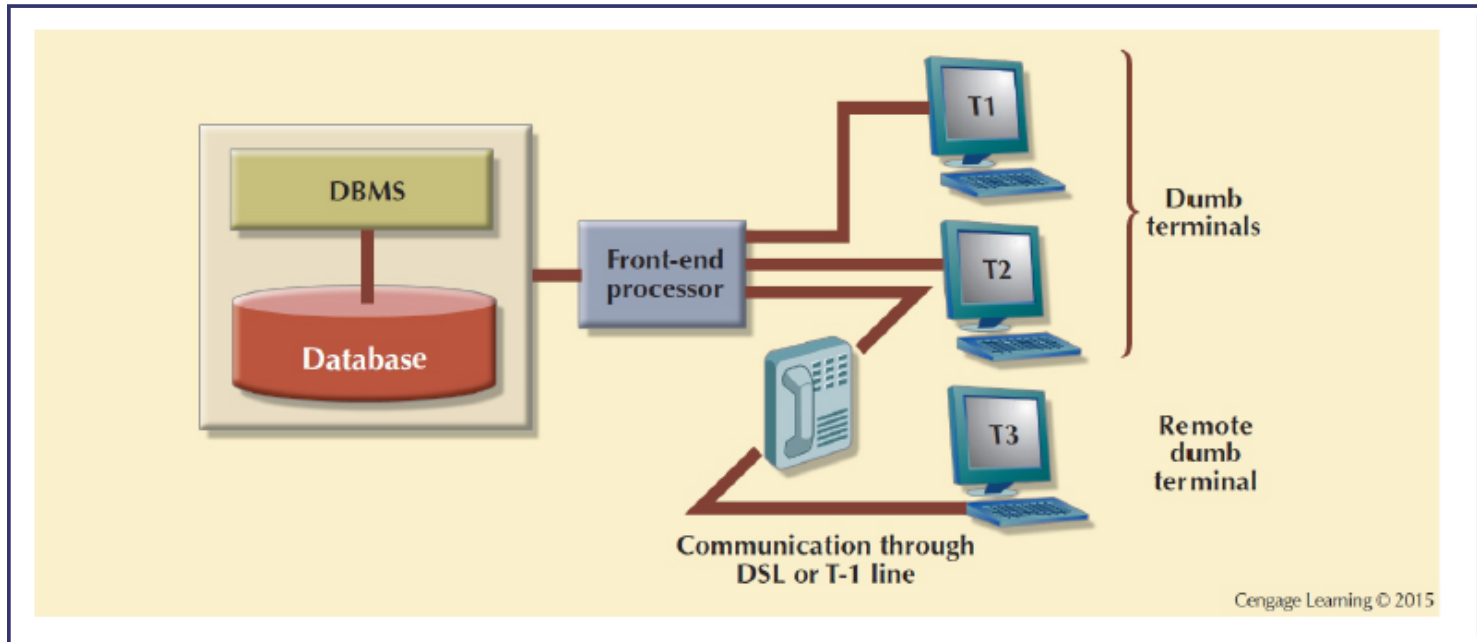
TABLE 12.2 Database Systems: Levels of Data and Process Distribution

	SINGLE-SITE DATA	MULTIPLE-SITE DATA
Single-site process	Host DBMS	Not applicable (Requires multiple processes)
Multiple-site process	File server Client/server DBMS (LAN DBMS)	Fully distributed Client/server DDBMS

SP/SD

Single-Site Processing, Single-Site Data (SPSD)

- Processing is done on a single host computer
- Data stored on host computer's local disk
- Processing restricted on end user's side
- DBMS is accessed by dumb terminals



MP/SD [note: no SP/MD!]

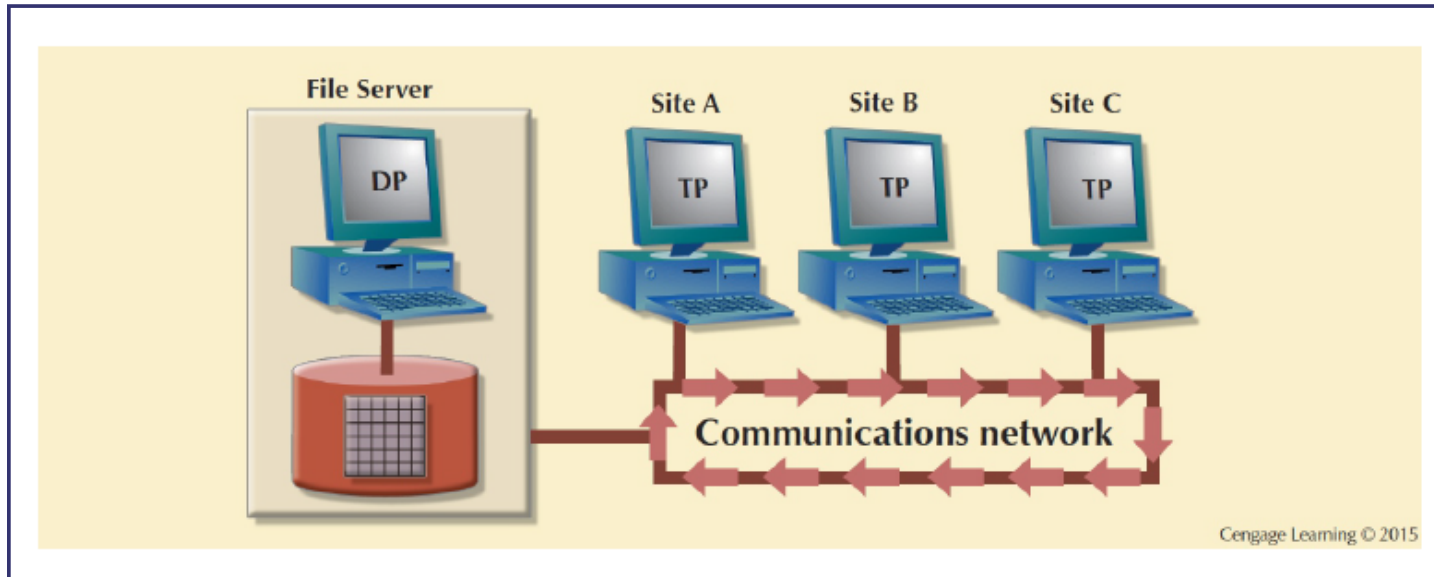
Multiple-Site Processing, Single-Site Data (MPSD)

- Multiple processes run on different computers sharing a single data repository
- Require network file server running conventional applications
 - Accessed through LAN
- **Client/server architecture**
 - Reduces network traffic
 - Processing is distributed
 - Supports data at multiple sites

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

17

Not truly 'distributed' (data I/O is centralized).



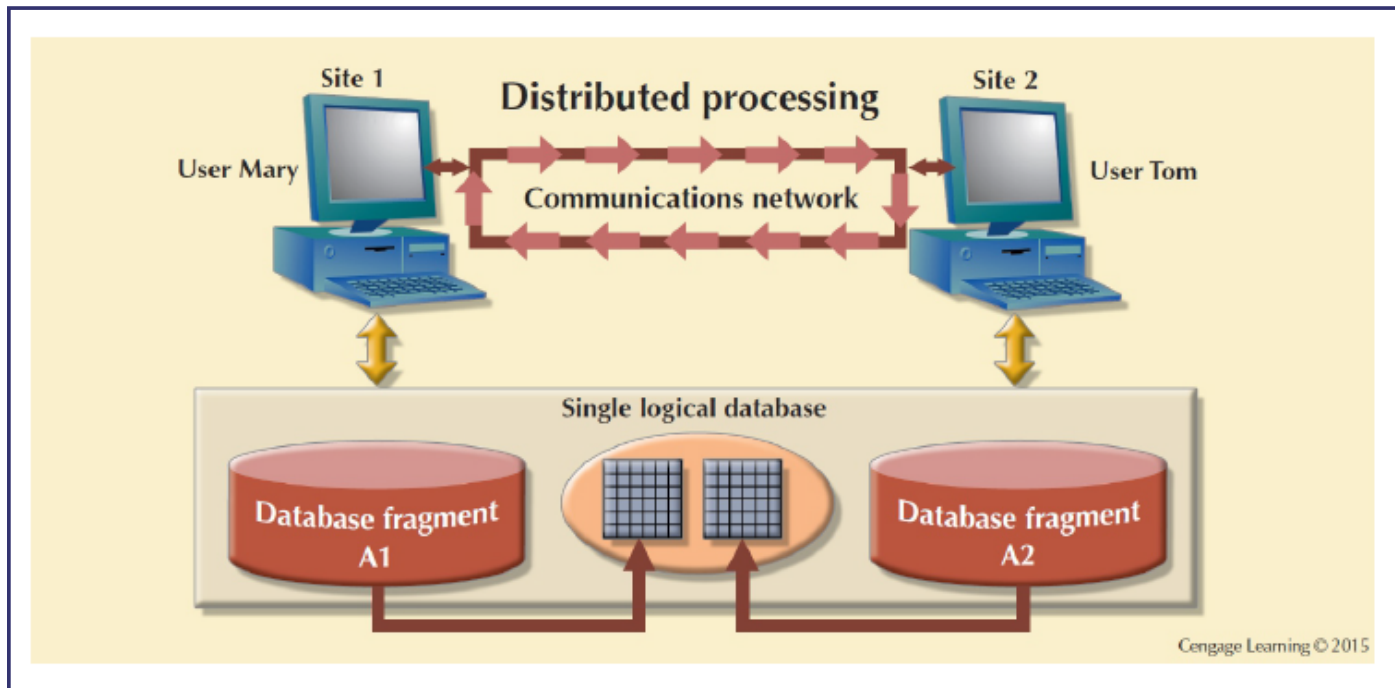
MP/MD ['fully distributed']

- Fully distributed database management system
 - Support multiple data processors and transaction processors at multiple sites
- Classification of DDBMS depending on the level of support for various types of databases
 - **Homogeneous**: Integrate multiple instances of same DBMS over a network
 - **Heterogeneous**: Integrate different types of DBMSs
 - **Fully heterogeneous**: Support different DBMSs, each supporting different data model

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

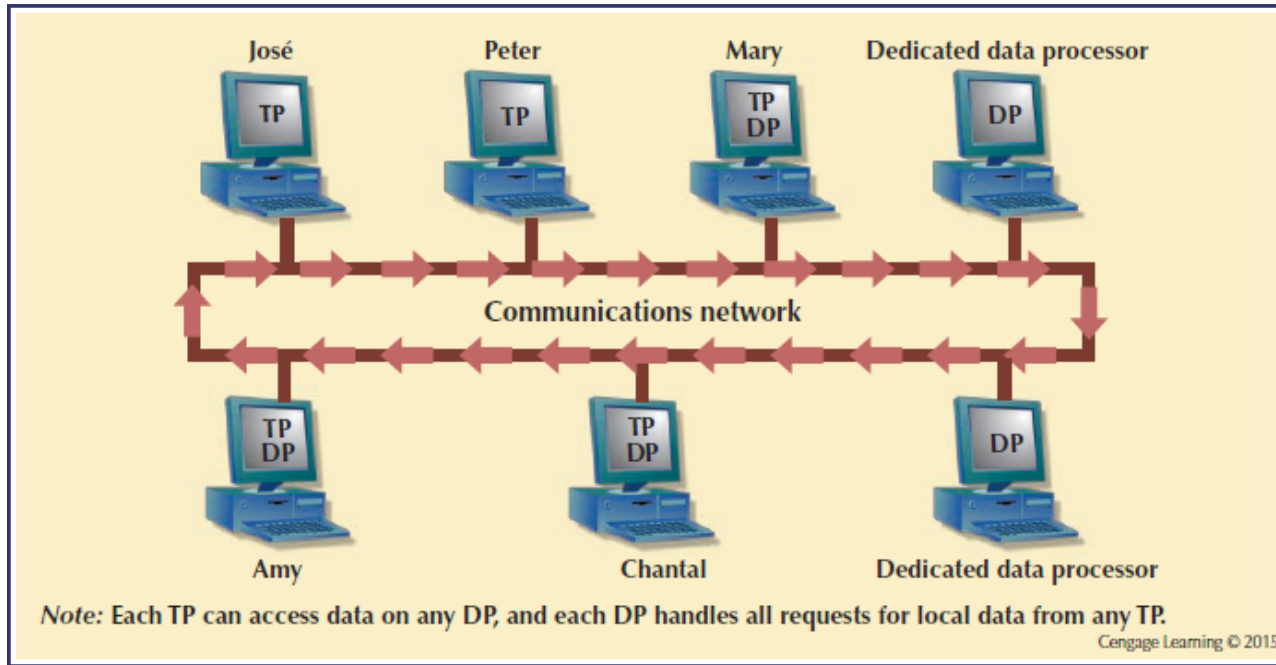
19

"The whole enchilada!".. Comes in three varieties.



The above schematic is same as the one you saw earlier..

In a fully distributed database, the TPs (that request data) are distributed, as are the DPs (that serve data), like so [you saw this in an earlier slide]:



The DDBMS uses protocols to transport across its network, commands as well as data, from distributed DPs and TPs: it receives data requests from multiple TPs, collects/synchronizes data (needed by multiple TPs) from multiple DPs, and intelligently routes to multiple TPs, the data (collected from multiple DPs) they requested. In other words, since transactions need data, the DDBMS acts as a switch, shuttling and routing data, from multiple DPs, to multiple TPs.

DDBMS restrictions

Restrictions of DDBMS

- Remote access is provided on a read-only basis
- Restrictions on the number of remote tables that may be accessed in a single transaction
- Restrictions on the number of distinct databases that may be accessed
- Restrictions on the database model that may be accessed

DDBMs span the spectrum from homegenous to fully heterogenous, and tend to come with restrictions.

Local/distributed requests/transactions

Remote/localized requests and transactions:

Remote request

- Single SQL statement accesses data processed by a single remote database processor

Remote transaction

- Accesses data at single remote site composed of several requests

Distributed (not localized) requests and transactions:

Distributed request

- Single SQL statement references data at several DP sites

Distributed transaction

- Requests data from several different remote sites on network

Distributed transactions are what need to be carefully executed so as to maintain distribution transparency - the transactions have to execute 'as if' they all ran on the same machine/location. This is achieved using '2PC' [explained in upcoming slides].

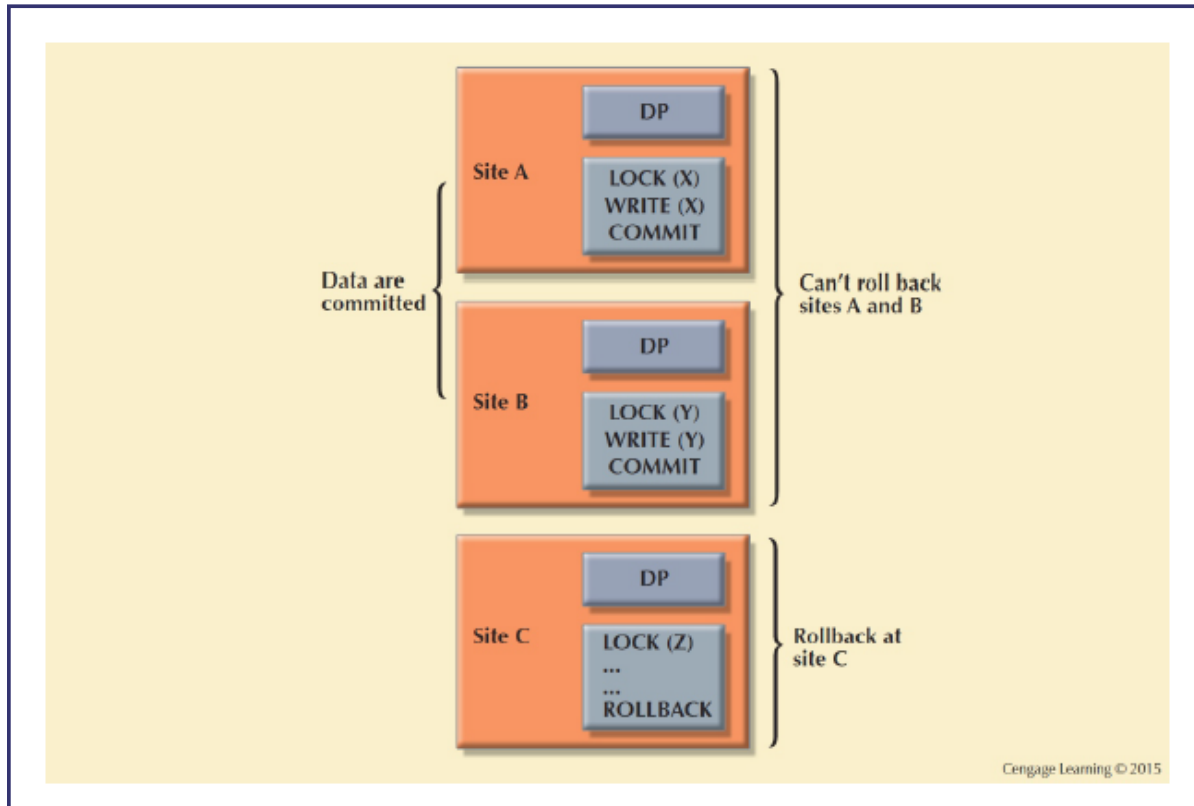
Distributed concurrency control

Distributed Concurrency Control

- Concurrency control is important in distributed databases environment
 - Due to multi-site multiple-process operations that create inconsistencies and deadlocked transactions

TP must ensure that all parts of a transaction (across sites) are completed, before doing a COMMIT.

Problem!



Can't COMMIT the entire transaction, can't ROLLBACK it either! Solution: 2PC.

Two-phase commit ("2PC") protocol

Two-Phase Commit Protocol (2PC)

- Guarantees if a portion of a transaction operation cannot be committed, all changes made at the other sites will be undone
 - To maintain a consistent database state
- Requires that each DP's transaction log entry be written before database fragment is updated
- **DO-UNDO-REDO protocol:** Roll transactions back and forward with the help of the system's transaction log entries

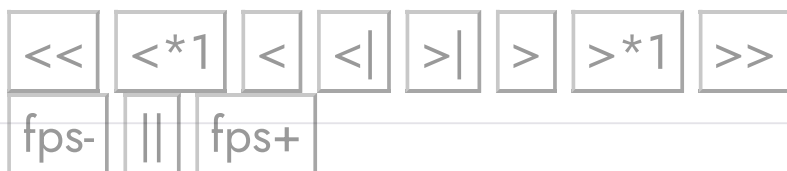
Two-Phase Commit Protocol (2PC)

- **Write-ahead protocol:** Forces the log entry to be written to permanent storage before actual operation takes place
- Defines operations between **coordinator** and **subordinates**
- Phases of implementation
 - Preparation
 - The final COMMIT

2PC: steps

The following is the sequence of steps (phase 1, phase2) that are used to effect a complete distributed transaction (keep clicking on the >| button):

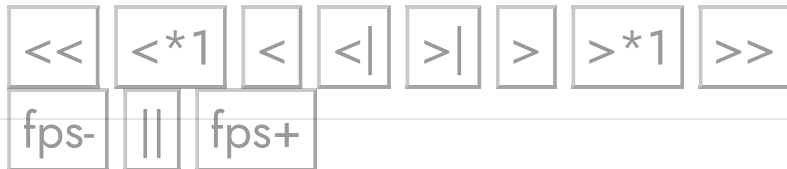
Click anywhere to show/hide controls:
 * <<, >>: first/last frame
 * <*1, >*1: play once from curr, rev/fwd
 * <, >: play (loop) rev/fwd
 * <|, >|: prev/next frame
 * fps -/+ : decr/incr fps
 * ?? : HUD toggle
 * || : halt
 Note - fps steps: 0.125,0.25,0.5,1,5,7,10,12,15,18,20,24,28,30



And here's how a transaction is aborted, when one of the participating nodes is unable to commit a sub-transaction:

Click anywhere to show/hide controls:

- * <<, >>: first/last frame
 - * <*1, >*1: play once from curr, rev/fwd
 - * <, >: play (loop) rev/fwd
 - * <|, >|: prev/next frame
 - * fps -/+ : decr/incr fps
 - * ?? : HUD toggle
 - * ||: halt
- Note - fps steps: 0.125,0.25,0.5,1,5,7,10,12,15,18,20,24,28,30



2PC: another description

In the two phase commit protocol, as the name implies, there are two phases:

- phase 1: commit request phase, aka 'voting' phase: coordinator sends a 'commit or abort?' (aka 'query to commit' or 'prepare to commit') message to each participating transaction node; each node responds (votes), with a 'can commit' (aka 'ready') or 'need to abort' (aka 'abort') message
- phase 2: commit phase, aka 'completion' phase:
 - if in phase 1, all nodes responded with 'can commit', the coordinator sends a 'commit' phase to each node; each node commits, and sends an acknowledgment to the coordinator
 - or, if in phase 1, any node responded with 'need to abort', the coordinator sends an 'abort' message to each node; each node aborts, and sends an acknowledgment to the coordinator

The devil's in the details - all sorts of variations/refinements exist in implementations, but the above is the overall (simple, robust) idea.

It's an all-or-nothing scheme - either all nodes of a distributed transaction locally commit (in which case the overall transaction is successful), or all of them locally abort so that the DB is not left in an inconsistent state unlike in the 'Problem!' slide (in which case the overall transaction fails and needs to be redone).

FYI, this is yet another description :)

"What distribution? We have NO distribution!"

Distributed Database Transparency Features

Distribution
transparency

Transaction
transparency

Failure
transparency

Performance
transparency

Heterogeneity
transparency

DDBMSs are designed to HIDE distribution specifics - this feature is called 'transparency', and can be classified into different kinds.

Distribution transparency

Distribution Transparency

- Allows management of physically dispersed database as if centralized
- Levels
 - **Fragmentation transparency**
 - **Location transparency**
 - **Local mapping transparency**

Fragmentation transparency: end user does not know that the data is fragmented.

Location transparency: end user does not know where fragments are located.

Location mapping transparency: end user does not know how fragments are mapped.

Distrib. transparency is supported via a distributed data dictionary (DDD) [aka DDC], which contains the distributed global schema which local TPs use to translate user requests for processing by DPs.

Transaction transparency

Transaction Transparency

- Ensures database transactions will maintain distributed database's integrity and consistency
- Ensures transaction completed only when all database sites involved complete their part
- Distributed database systems require complex mechanisms to manage transactions

Performance transparency, failure transparency

Performance and Failure Transparency

- **Performance transparency:** Allows a DDBMS to perform as if it were a centralized database
- **Failure transparency:** Ensures the system will operate in case of network failure
- Considerations for resolving requests in a distributed data environment
 - Data distribution
 - Data replication
 - **Replica transparency:** DDBMS's ability to hide multiple copies of data from the user

Performance and Failure Transparency

- Network and node availability
 - **Network latency:** delay imposed by the amount of time required for a data packet to make a round trip
 - **Network partitioning:** delay imposed when nodes become suddenly unavailable due to a network failure

We also need to take into account, network delay and network failure ("partitioning") when planning for or evaluating transparency.

Distributed DB design

Distributed Database Design

Data fragmentation

- How to partition database into fragments

Data replication

- Which fragments to replicate

Data allocation

- Where to locate those fragments and replicas

Partition (divide), replicate (copy)? Where to store the partitions/copies?

Fragmentation (partitioning)

Data Fragmentation

- Breaks single object into many segments
 - Information is stored in distributed data catalog (DDC)
- Strategies
 - **Horizontal fragmentation:** Division of a relation into subsets (fragments) of tuples (rows)
 - **Vertical fragmentation:** Division of a relation into attribute (column) subsets
 - **Mixed fragmentation:** Combination of horizontal and vertical strategies

Replication

Data Replication

- Data copies stored at multiple sites served by a computer network
- **Mutual consistency rule:** Replicated data fragments should be identical
- Styles of replication
 - Push replication
 - Pull replication
- Helps restore lost data

Data Replication Scenarios

Fully replicated database

- Stores multiple copies of each database fragment at multiple sites

Partially replicated database

- Stores multiple copies of some database fragments at multiple sites

Unreplicated database

- Stores each database fragment at a single site

You can read more about replication, [here](https://bytes.usc.edu/cs585/f23-Da-taaa/lectures/DistributedDBs/slides.html).

Storage of fragments/replicas

Data Allocation Strategies

Centralized data allocation

- Entire database stored at one site

Partitioned data allocation

- Database is divided into two or more disjointed fragments and stored at two or more sites

Replicated data allocation

- Copies of one or more database fragments are stored at several sites

The CAP 'theorem' [2 out of 3; 1 out of 2]

The CAP Theorem

- CAP stands for:
 - Consistency
 - Availability
 - Partition tolerance
- **Basically available, soft state, eventually consistent (BASE)**
 - Data changes are not immediate but propagate slowly through the system until all replicas are consistent

Consistency: always correct data.

Availability: requests are always filled.

Partition ("outage") tolerance: continue to operate even if (some/most) nodes fail.

The CAP theorem 'used to say' that in a networked (distributed) DB system, at most 2 out of 3 of the above are achievable [PC, CA, or PA]. But CA means low P, that means that we can't even operate, which makes CA a moot point! In other words, **CA doesn't exist**, ie. a low P is not an option! So now we think of it differently: in the event of a network partition (P has occurred), only one of availability or consistency is achievable. You can read more [here](#).

In the 'ACID' (Atomicity, Consistency, Isolation, Durability) world of older relational DBs, the CAP Theorem was a reminder that it is usually difficult to achieve C,A,P all at once, and that consistency is more important than availability.

In today's 'BASE' (Basically Available, Soft_state, Eventually_consistent) model of non-relational (eg. NoSQL) DBs, we prefer to sacrifice consistency in favor of availability.

DBMS TYPE	CONSISTENCY	AVAILABILITY	PARTITION TOLERANCE	TRANSACTION MODEL	TRADE-OFF
Centralized DBMS	High	High	N/A	ACID	No distributed data processing
Relational DDBMS (2PC)	High Sacrifices availability to ensure consistency and isolation	Relaxed	High	ACID	
NoSQL DDBMS	Relaxed	High	High	BASE	Sacrifices consistency to ensure availability

Cengage Learning © 2015

Date's '12 commandments' for distributed DBMSs

RULE NUMBER	RULE NAME	RULE EXPLANATION
1	<i>Local-site independence</i>	Each local site can act as an independent, autonomous, centralized DBMS. Each site is responsible for security, concurrency control, backup, and recovery.
2	<i>Central-site independence</i>	No site in the network relies on a central site or any other site. All sites have the same capabilities.
3	<i>Failure independence</i>	The system is not affected by node failures. The system is in continuous operation even in the case of a node failure or an expansion of the network.
4	<i>Location transparency</i>	The user does not need to know the location of data to retrieve those data.
5	<i>Fragmentation transparency</i>	Data fragmentation is transparent to the user, who sees only one logical database. The user does not need to know the name of the database fragments to retrieve them.
6	<i>Replication transparency</i>	The user sees only one logical database. The DDBMS transparently selects the database fragment to access. To the user, the DDBMS manages all fragments transparently.
7	<i>Distributed query processing</i>	A distributed query may be executed at several different DP sites. Query optimization is performed transparently by the DDBMS.
8	<i>Distributed transaction processing</i>	A transaction may update data at several different sites, and the transaction is executed transparently.
9	<i>Hardware independence</i>	The system must run on any hardware platform.
10	<i>Operating system independence</i>	The system must run on any operating system platform.
11	<i>Network independence</i>	The system must run on any network platform.
12	<i>Database independence</i>	The system must support any vendor's database product.

Think of these more as checklist items, rather than commandments.