1.Parallel and Distributed Databases

Parallel database

1. -Introduction
2. -Architecture for Parallel databases.
3. - Parallel query Evaluation
4. - Parallelizing Individual operations.

1. Parallel DB /D.S.Jagli
2

Introduction

- What is a Centralized Database ?

-all the data is maintained at a single site and assumed that the processing of individual transaction is essentially sequential.

1. Parallel DB /D.S.Jagli
3

PARALLEL DBMSs

WHY DO WE NEED THEM?

- More and More Data!

  We have databases that hold a high amount of data, in the order of 10^{12} bytes:

  10,000,000,000,000 bytes!

- Faster and Faster Access!

  We have data applications that need to process data at very high speeds:

  10,000s transactions per second!

SINGLE-PROCESSOR DBMS AREN'T UP TO THE JOB!

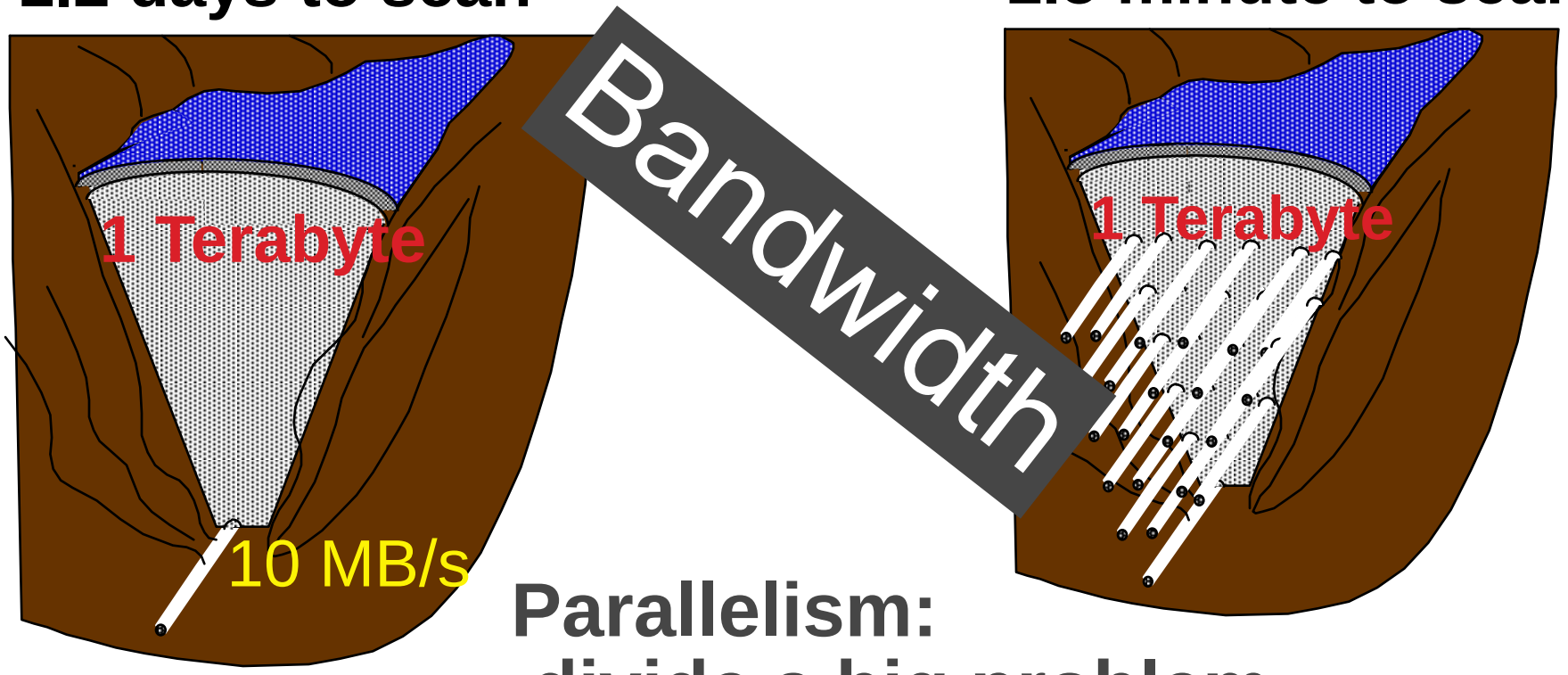
1. Parallel DB /D.S.Jagli

4

Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan

1,000 x parallel
1.5 minute to scan.



Parallelism:
divide a big problem
into many smaller ones
to be solved in parallel.

Parallel DB

- Parallel database system seeks to improve performance through parallelization of various operations such as loading data ,building indexes, and evaluating queries by using multiple CPUs and Disks in Parallel.

- Motivation for Parallel DB
- Parallel machines are becoming quite common and affordable
  - Prices of microprocessors, memory and disks have dropped sharply
- Databases are growing increasingly large
  - large volumes of transaction data are collected and stored for later analysis.
  - multimedia objects like images are increasingly stored in databases

PARALLEL DBMSs

BENEFITS OF A PARALLEL DBMS

☺ Improves Response Time.

INTERQUERY PARALLELISM

It is possible to process a number of transactions in parallel with each other.

☺ Improves Throughput.

INTRAQUERY PARALLELISM

It is possible to process 'sub-tasks' of a transaction in parallel with each other.

1. Parallel DB /D.S.Jagli

7

PARALLEL DBMSs

HOW TO MEASURE THE BENEFITS

- Speed-Up
  - Adding more resources results in proportionally less running time for a fixed amount of data.

10 seconds to scan a DB of 10,000 records using 1 CPU
1 second to scan a DB of 10,000 records using 10 CPUs

- Scale-Up
- If resources are increased in proportion to an increase in data/problem size, the overall time should remain constant

  - 1 second to scan a DB of 1,000 records using 1 CPU

1 second to scan a DB of 10,000 records using 10 CPUs

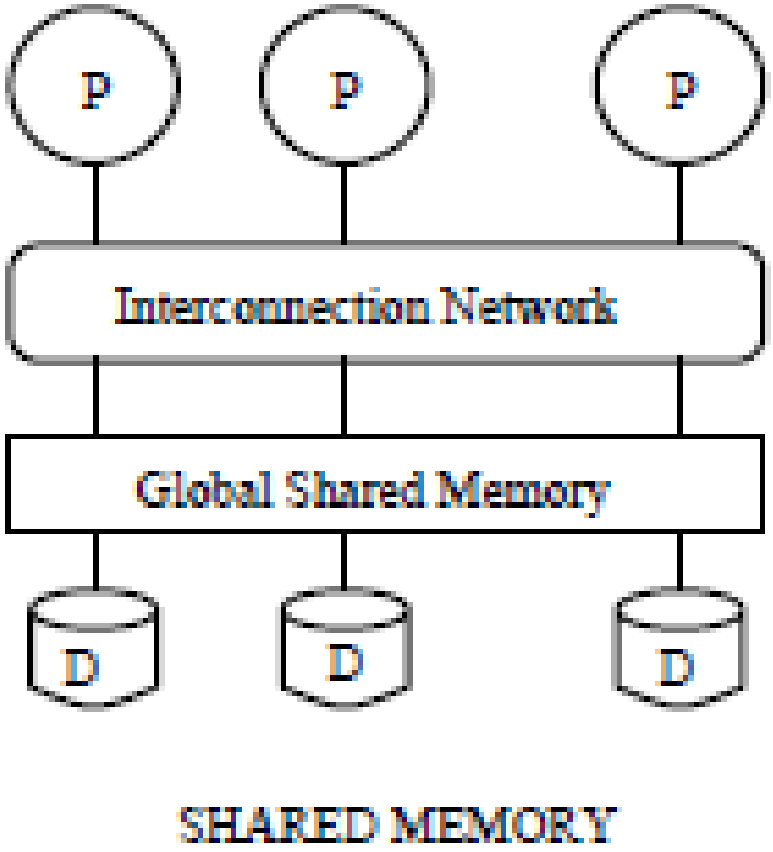
1. Parallel DB /D.S.Jagli
8

Architectures for Parallel Databases

- The basic idea behind Parallel DB is to carry out evaluation steps in parallel whenever is possible.
- There are many opportunities for parallelism in RDBMS.
- 3 main architectures have been proposed for building parallel DBMSs.
  1. **Shared Memory**
  2. **Shared Disk**
  3. **Shared Nothing**

1. Parallel DB /D.S.Jagli
9

Shared Memory



SHARED MEMORY

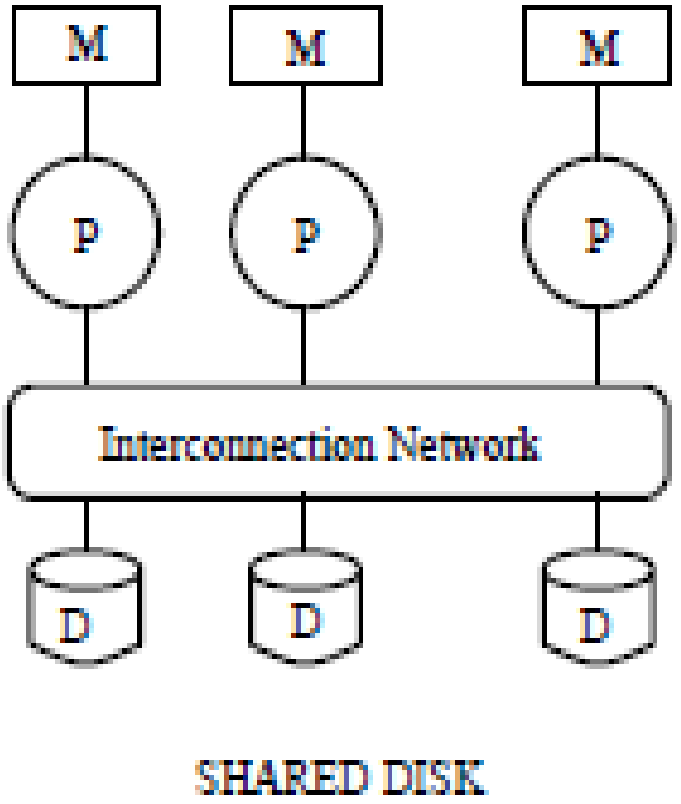
- **Advantages:**

1. It is closer to conventional machine , Easy to program
2. overhead is low.
3. OS services are leveraged to utilize the additional CPUs.

- **Disadvantage:**

1. It leads to bottleneck problem
2. Expensive to build
3. It is less sensitive to partitioning

Shared Disk



SHARED DISK

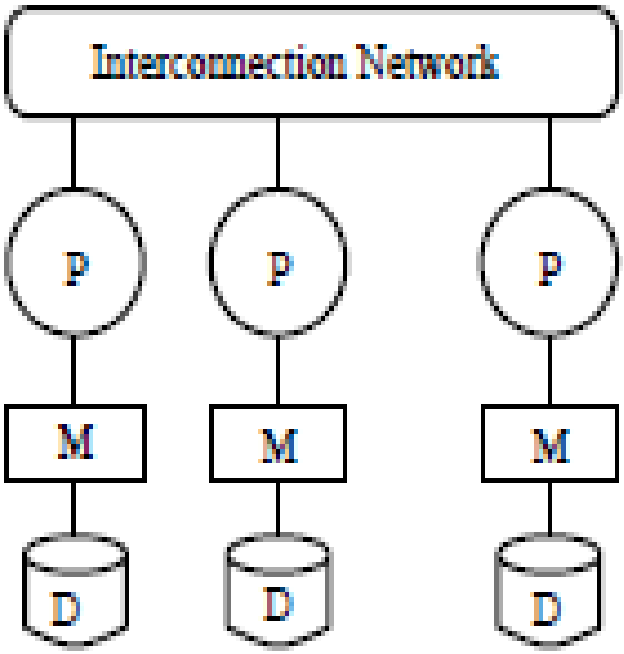
- **Advantages:**

1. Almost same

- **Disadvantages:**

1. More interference
2. Increases N/W band width
3. Shared disk less sensitive to partitioning

Shared Nothing



SHARED NOTHING

- **Advantages:**

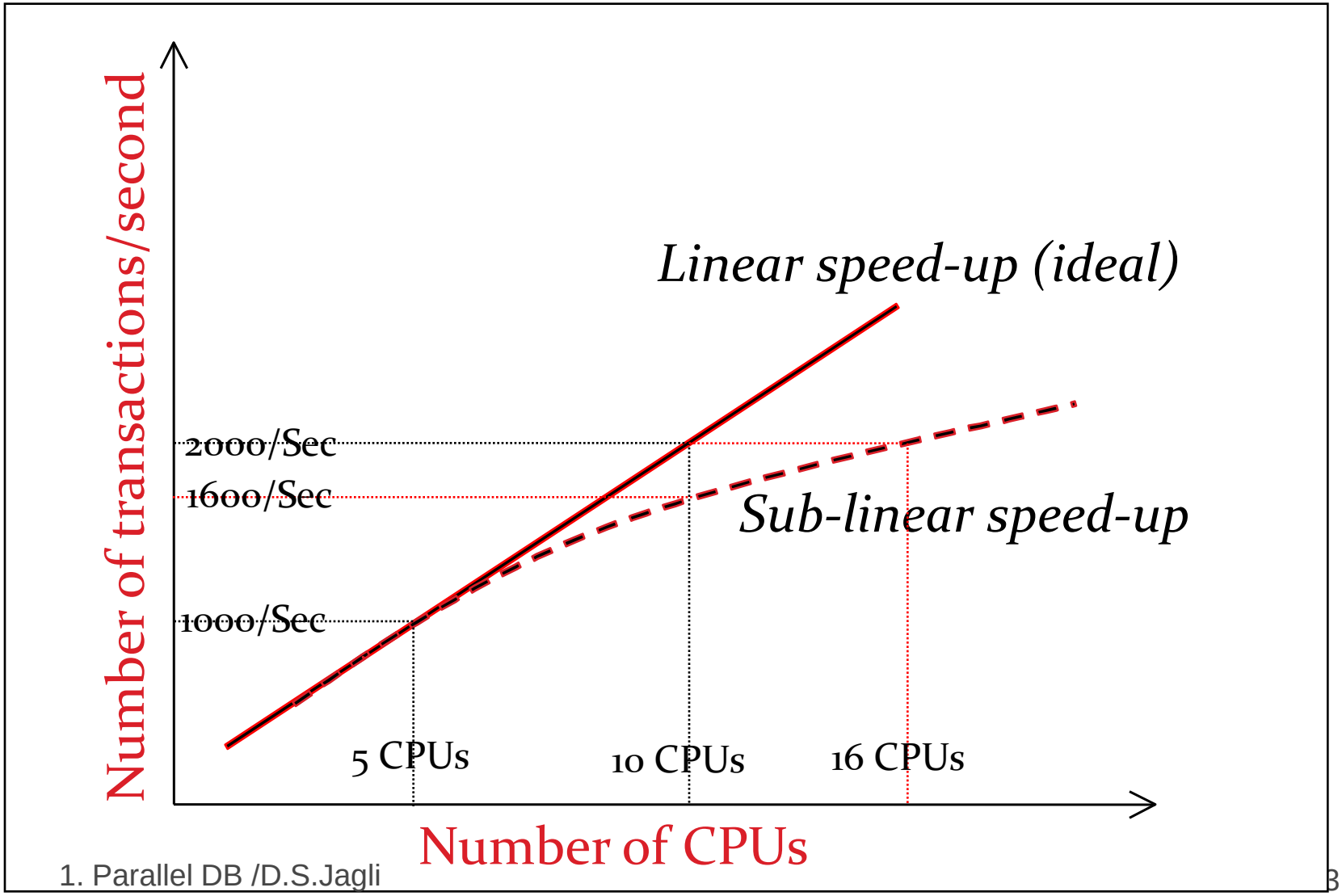
1. It provides linear scale up &linear speed up
2. Shared nothing benefits from "good" partitioning
3. Cheap to build

- **Disadvantage**

1. Hard to program
2. Addition of new nodes requires reorganizing

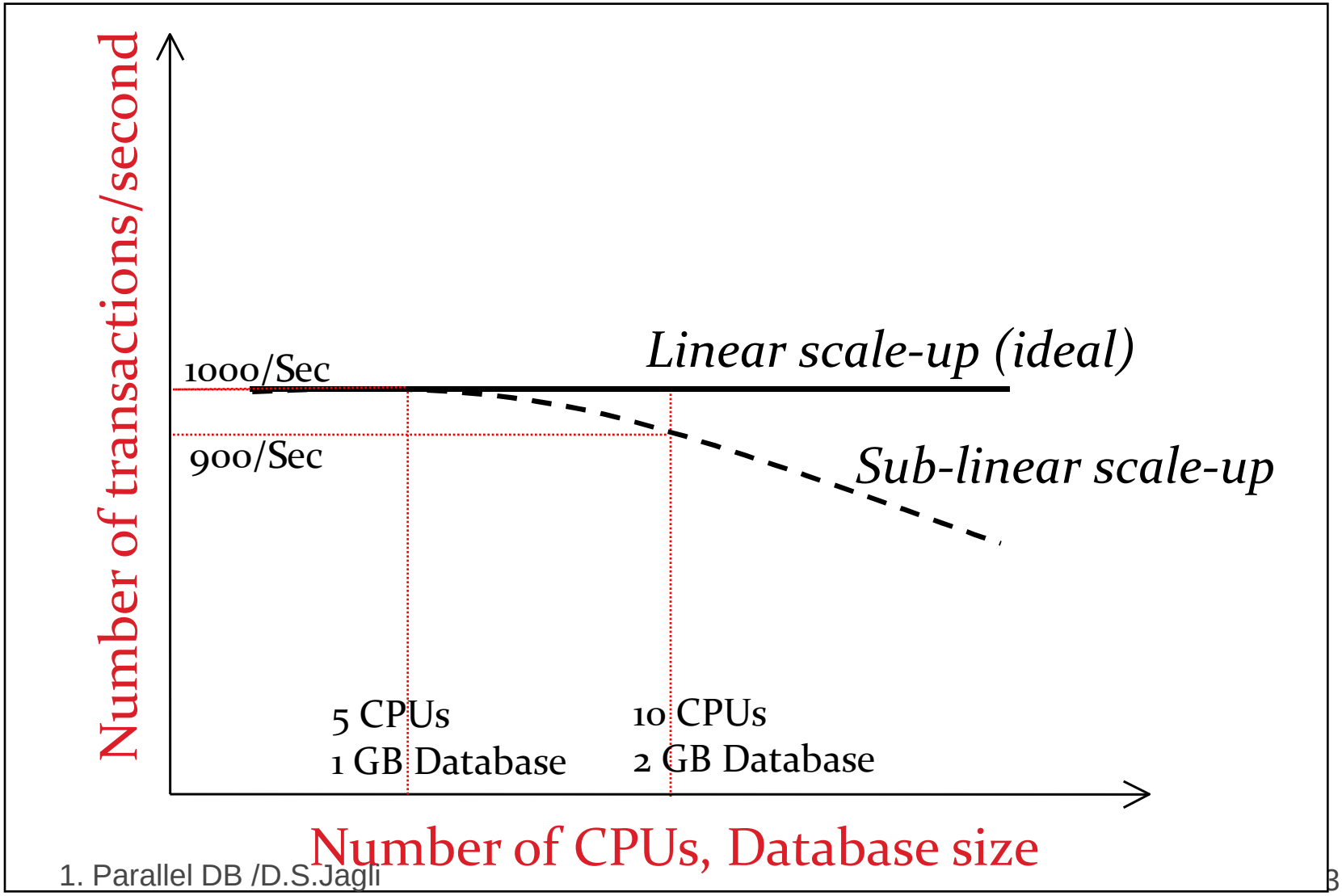
PARALLEL DBMSs

SPEED-UP



PARALLEL DBMSs

SCALE-UP



PARALLEL QUERY EVALUATION

A relational query execution plan is graph/tree of relational algebra operators (based on this operators can execute in parallel)

1. 1. Parallel DB /D.S.Jagli

15

Different Types of DBMS ||-ism

- Parallel evaluation of a relational query in DBMS With shared –nothing architecture

1. **Inter-query parallelism**
   - Multiple queries run on different sites
2. **Intra-query parallelism**
   - Parallel execution of single query run on different sites.

   a) **Intra-operator parallelism**

      a) get all machines working together to compute a given operation (scan, sort, join).

   b) **Inter-operator parallelism**
   - each operator may run concurrently on a different site (exploits pipelining).
   - In order to evaluate different operators in parallel, we need to evaluate each operator in query plan in Parallel.

Data Partitioning

- **Types of Partitioning**

1. **Horizontal Partitioning**: tuple of a relation are divided among many disks such that each tuple resides on one disk.
   - *It enables to exploit the I/O band width of disks by reading & writing them in parallel.*
   - *Reduce the time required to retrieve relations from disk by partitioning the relations on multiple disks.*
     1. **Range Partitioning**
     2. **Hash Partitioning**
     3. **Round Robin Partitioning**

2. **Vertical Partitioning**

1.Range Partitioning

- Tuples are sorted (conceptually), and n ranges are chosen for the sort key values so that each range contains roughly the same number of tuples;
- tuples in range i are assigned to processor i.
- Eg:
  - sailor _id 1-10 assigned to disk 1
  - sailor _id 10-20 assigned to disk 2
  - sailor _id 20-30 assigned to disk 3

- *range partitioning can lead to **data skew;** that is, partitions with widely varying number of tuples across*

2.Hash Partitioning

- **A hash function** is applied to selected fields of a tuple to determine its processor.

- Hash partitioning has the additional virtue that it keeps data evenly distributed even if the data grows and shrinks over time.

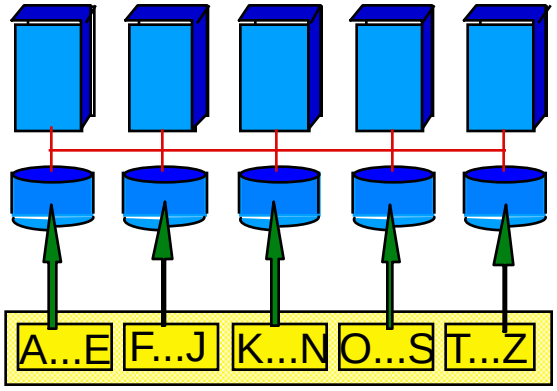
3.Round Robin Partitioning

- If there are *n processors, the i th tuple is assigned to processor* ***i mod n*** in **round-robin partitioning.**

- Round-robin partitioning is suitable for efficiently evaluating queries that access the entire relation.

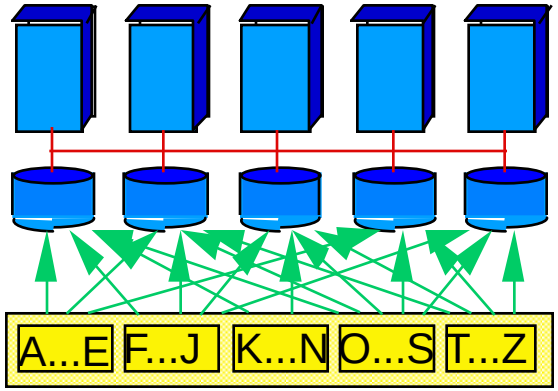
  - *If only a subset of the tuples (e.g., those that satisfy the selection condition age = 20) is required, hash partitioning and range partitioning are better than round-robin partitioning*

Range



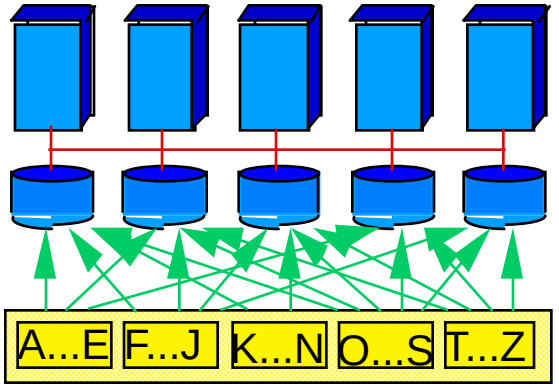
Good for equijoins, exact-match queries, and range queries

Hash



Good for equijoins, exact match queries

Round Robin



Good to spread load

Parallelizing Sequential Operator Evaluation Code

1. An elegant software architecture for parallel DBMSs enables us to readily parallelize existing code for sequentially evaluating a relational operator.

2. The basic idea is to use parallel data streams.

3. Streams are merged as needed to provide the inputs for a relational operator.

4. The output of an operator is split as needed to parallelize subsequent processing.

5. A parallel evaluation plan consists of a dataflow network of ***relational, merge, and split operators.***

PARALLELIZING INDIVIDUAL OPERATIONS

- How various operations can be implemented in parallel in a shared-nothing architecture?

- Techniques
  1. Bulk loading& scanning
  2. Sorting
  3. Joins

1.Bulk Loading and scanning

- ***scanning a relation:*** *Pages* can be read in parallel while scanning a relation, and the retrieved tuples can then be merged, if the relation is partitioned across several disks.

- ***bulk loading*****:** if a relation has associated indexes, any sorting of data entries required for building the indexes during bulk loading can also be done in parallel.

2.Parallel Sorting :

- **Parallel sorting steps:**
  1. First redistribute all tuples in the relation using range partitioning.
  2. Each processor then sorts the tuples assigned to it
  3. The entire sorted relation can be retrieved by visiting the processors in an order corresponding to the ranges assigned to them.

- Problem: ***Data skew***

- Solution: “sample” the data at the outset to determine good range partition points.

A particularly important application of parallel sorting is sorting the data entries in tree-structured indexes.

1. Parallel DB /D.S.Jagli
25

3.Parallel Join

1. The basic idea for joining A and B in parallel is to decompose the join into a collection of k smaller joins by using partition.

2. By using the same partitioning function for both A and B, we ensure that the union of the k smaller joins computes the join of A and B.

- **Hash-Join**
- **Sort-merge-join**

1. Parallel DB /D.S.Jagli

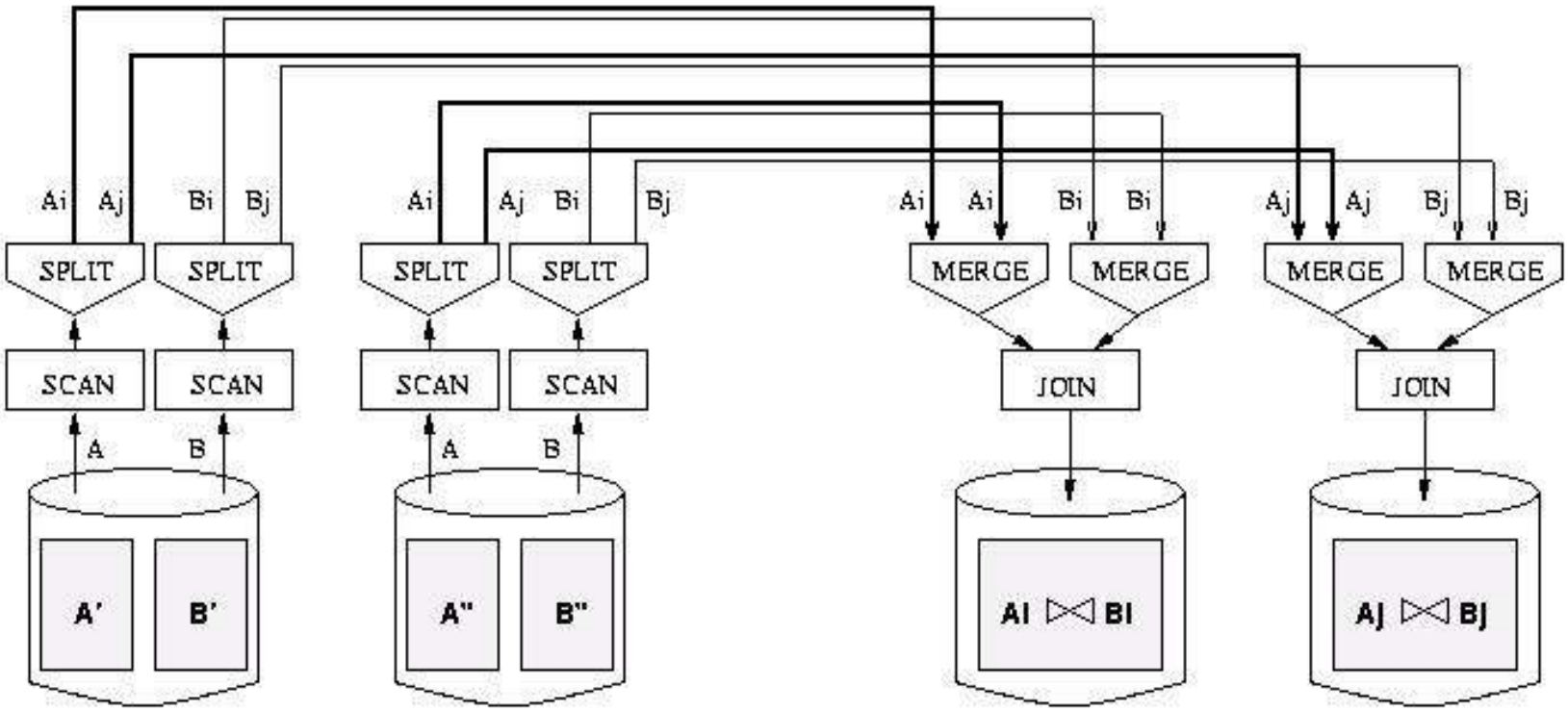
26

Sort-merge-join

- partition A and B by dividing the range of the join attribute into *k* disjoint subranges and placing A and B tuples into partitions according to the subrange to which their values belong.
- Each processor carry out a local join.
- In this case the number of partitions *k* is chosen to be equal to the number of processors *n* .
- The result of the join of A and B, the output of the join process may be split into several data streams.

The advantage that the output is available in sorted order

Dataflow Network of Operators for Parallel Join



Good use of split/merge makes it easier to build parallel versions of sequential join code

DDBMS

I. Introduction to DDBMS
II. Architecture of DDBs
III. Storing data in DDBs
IV. Distributed catalog management
V. Distributed query processing
VI. Transaction Processing
VII. Distributed concurrency control and recovery

1. Parallel DB /D.S.Jagli
29

I.Introduction to DDBMS

- Data in a distributed database system is stored across several sites.
- Each site is typically managed by a DBMS that can run independently of the other sites that co-operate in a transparent manner.
  - ***Transparent*** implies that each user within the system may access all of the data within all of the databases as if they were a single database
- There should be ‘location independence’ i.e.- as the user is unaware of where the data is located it is possible to move the data from one physical location to another without affecting the user.

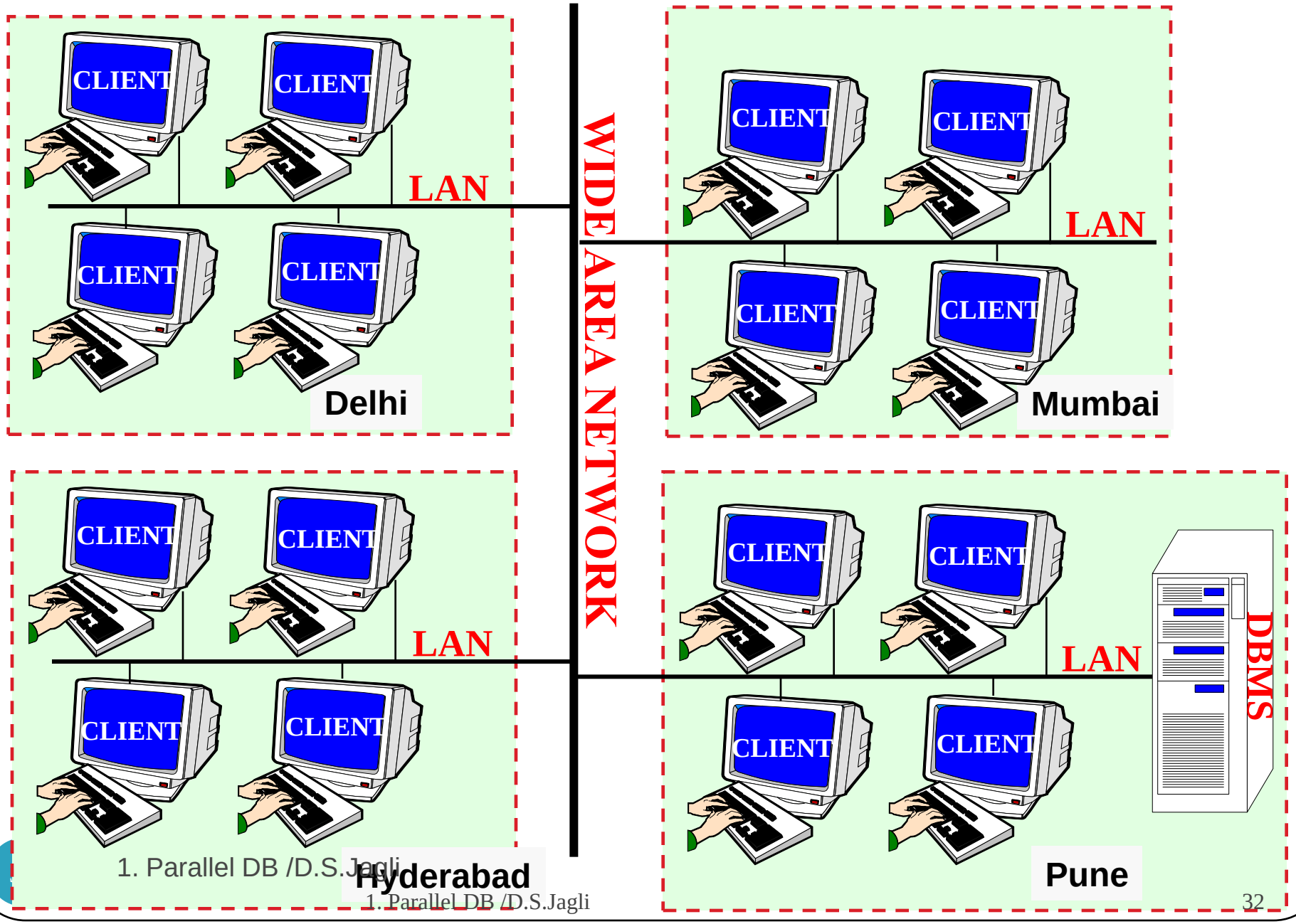
1. Parallel DB /D.S.Jagli
30

DDBMS properties

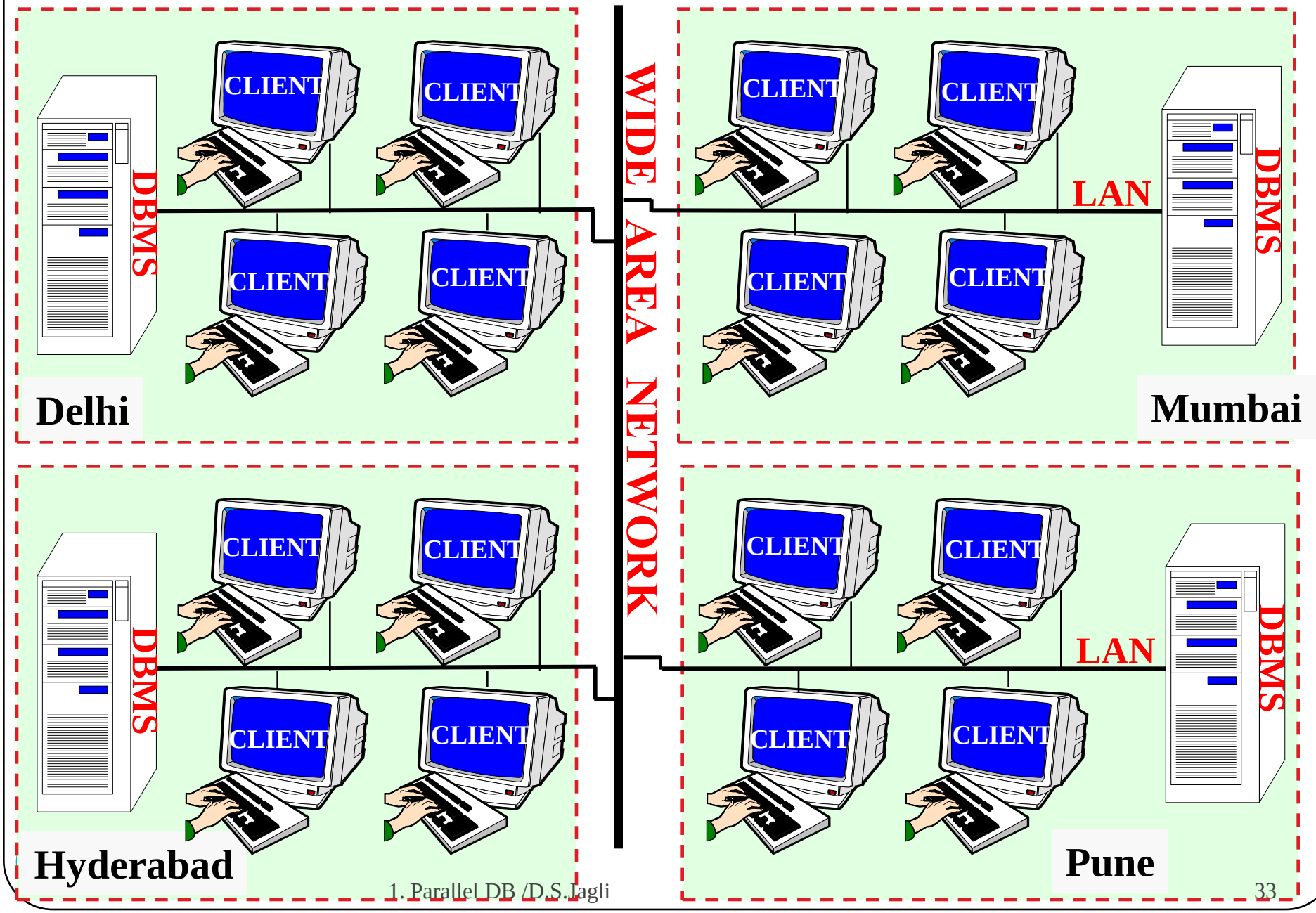
- **Distributed data independence**: Users should be able to ask queries without specifying where the referenced relations, or copies or fragments of the relations, are located.

- **Distributed transaction atomicity:** Users should be able to write transactions that access and update data at several sites just as they would write transactions over purely local data.

DISTRIBUTED PROCESSING ARCHITECTURE



DISTRIBUTED DATABASE ARCHITECTURE

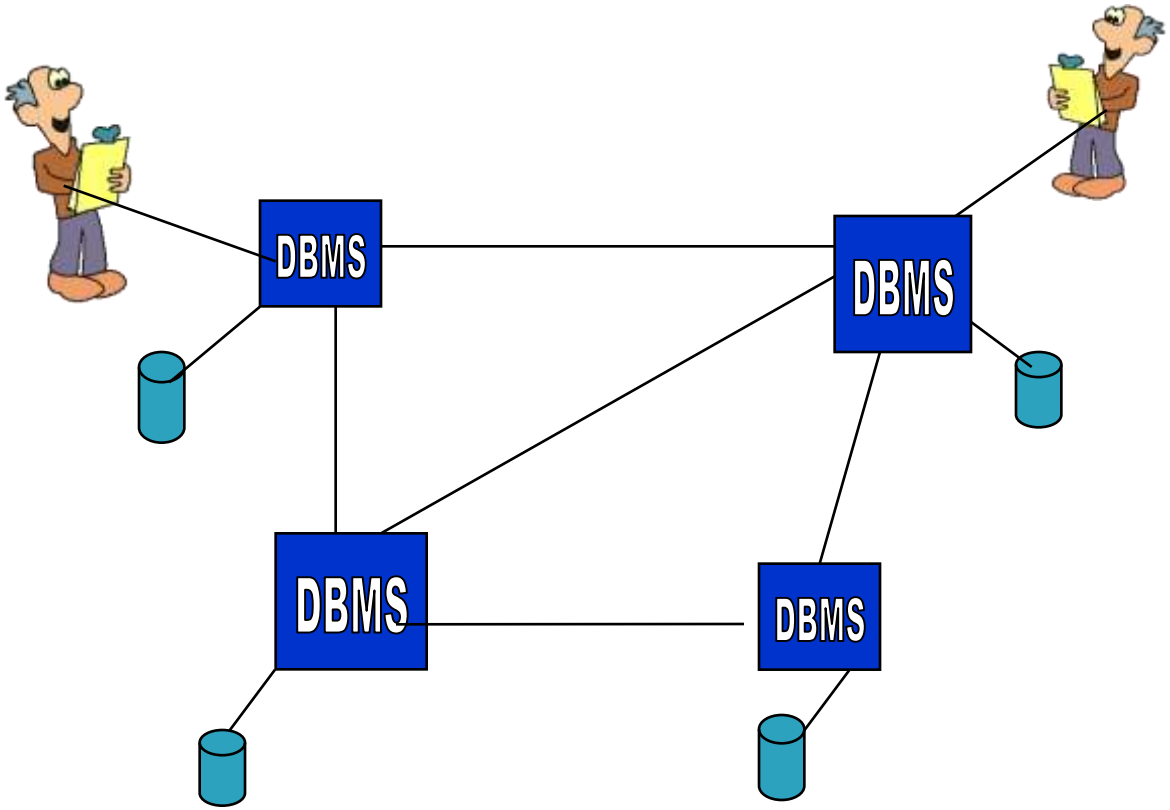


Distributed database

- Communication Network- DBMS and Data at each node

•Users are unaware of the distribution of the data

= Location transparency



Types of Distributed Databases

- **Homogeneous distributed database system** :
  - If data is distributed but all servers run the same DBMS software.

- **Heterogeneous distributed database** :
  - If different sites run under the control of different DBMSs, essentially autonomously, are connected to enable access to data from multiple sites.

- The key to building heterogeneous systems is to have well-accepted standards for gateway protocols.
- **A gateway protocol** is an API that exposes DBMS functionality to external applications.

DDBMS

I. Introduction to DDBMS
II. Architecture of DDBs
III. Storing data in DDBs
IV. Distributed catalog management
V. Distributed query processing
VI. Transaction Processing
VII. Distributed concurrency control and recovery

1. Parallel DB /D.S.Jagli
36

2.DISTRIBUTED DBMS ARCHITECTURES

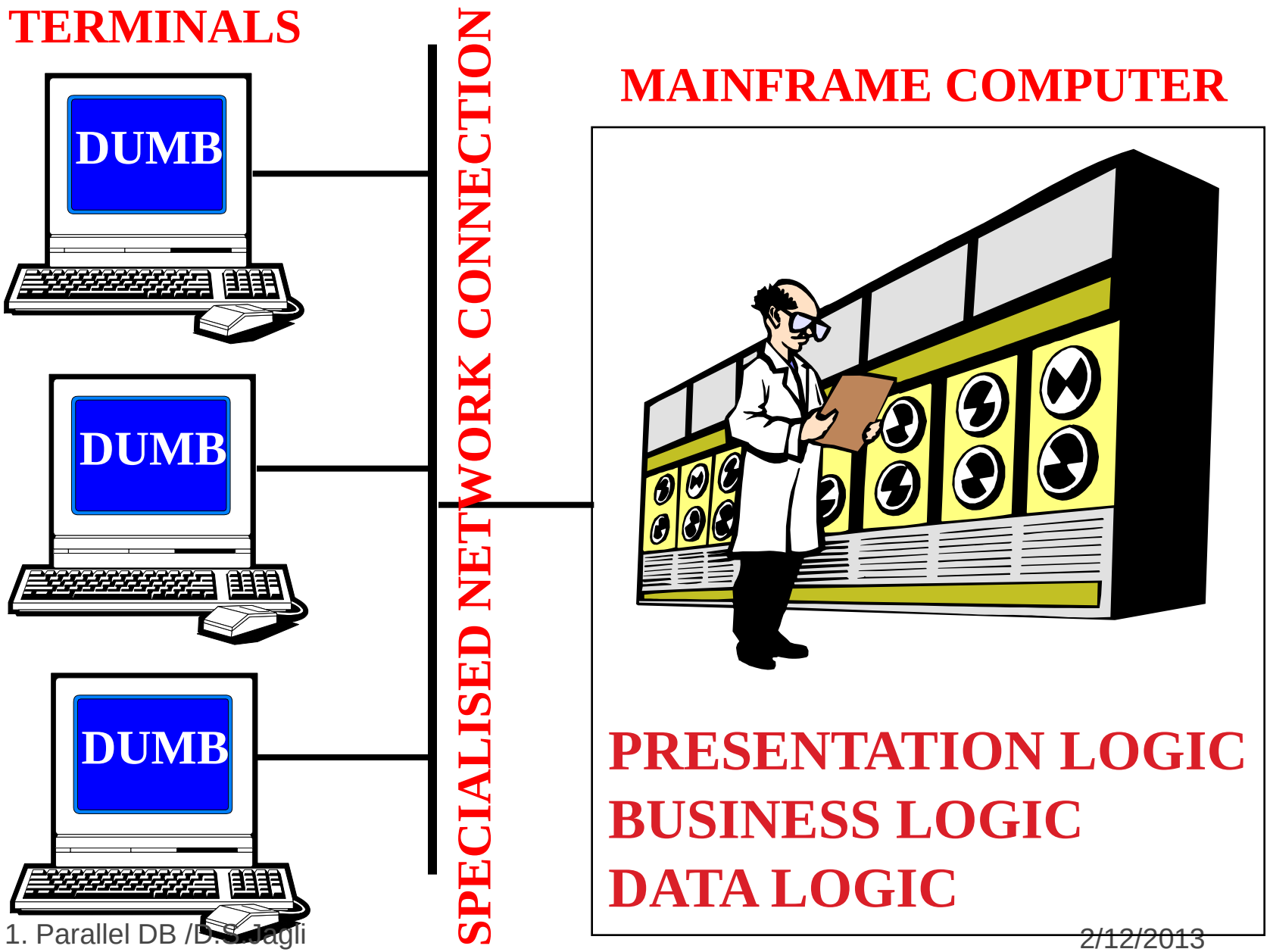
1. Client-Server Systems:
2. Collaborating Server Systems
3. Middleware Systems

1. Parallel DB /D.S.Jagli
37

1.Client-Server Systems:

1. A Client-Server system has one or more client processes and one or more server processes,
2. A client process can send a query to any one server process.
3. Clients are responsible for user-interface issues,
4. Servers manage data and execute transactions.
5. A client process could run on a personal computer and send queries to a server running on a mainframe.
6. The Client-Server architecture does not allow a single query to span multiple servers

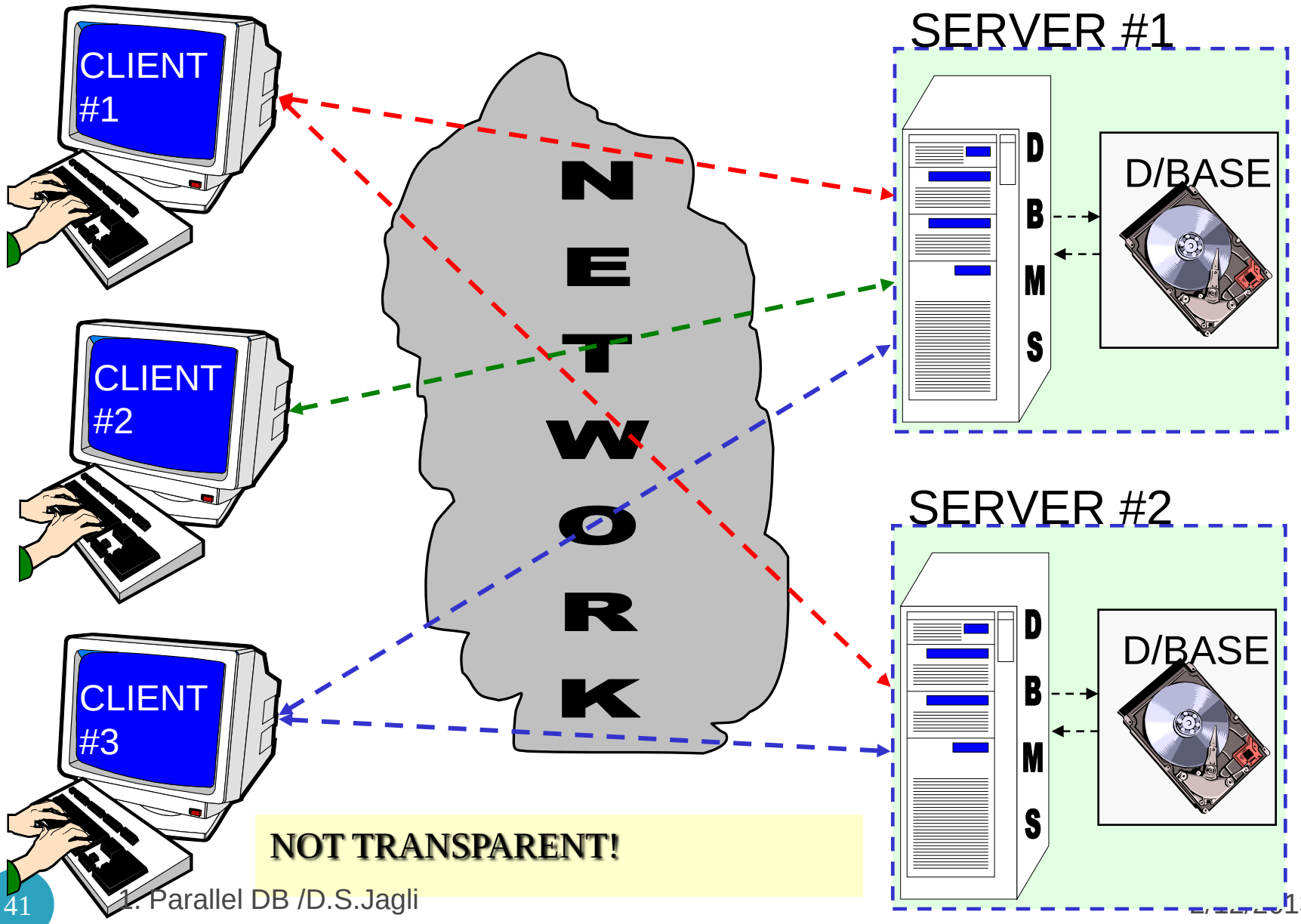
1. Parallel DB /D.S.Jagli
38


TERMINALS
DUMB
DUMB
DUMB
SPECIALISED NETWORK CONNECTION
MAINFRAME COMPUTER
PRESENTATION LOGIC
BUSINESS LOGIC
DATA LOGIC

2.Collaborating Server Systems

1. The client process would have to be capable of breaking such a query into appropriate subqueries.
2. **A Collaborating Server** system can have a collection of database servers, each capable of running transactions against local data, which cooperatively execute transactions spanning multiple servers.
3. When a server receives a query that requires access to data at other servers, it generates appropriate subqueries to be executed by other servers .
4. puts the results together to compute answers to the original query.

M:N CLIENT/SERVER DBMS ARCHITECTURE



NOT TRANSPARENT!

3.Middleware Systems:

- The Middleware architecture is designed to allow a single query to span multiple servers, without requiring all database servers to be capable of managing such multisite execution strategies.

- It is especially attractive when trying to integrate several legacy systems, whose basic capabilities cannot be extended.

DDBMS

I. Introduction to DDBMS
II. Architecture of DDBs
III. Storing data in DDBs
IV. Distributed catalog management
V. Distributed query processing
VI. Transaction Processing
VII. Distributed concurrency control and recovery

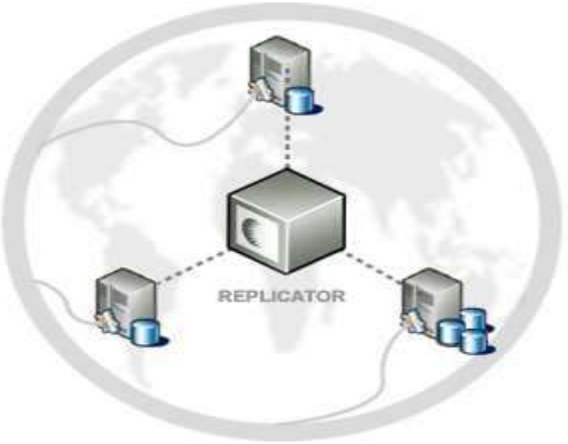
1. Parallel DB /D.S.Jagli
43

3.Storing Data in DDBs

- ✓ In a distributed DBMS, relations are stored across several sites.
- ✓ Accessing a relation that is stored at a remote site includes message-passing costs.
- ✓ A single relation may be *partitioned* or *fragmented* across several sites.

1. Parallel DB /D.S.Jagli

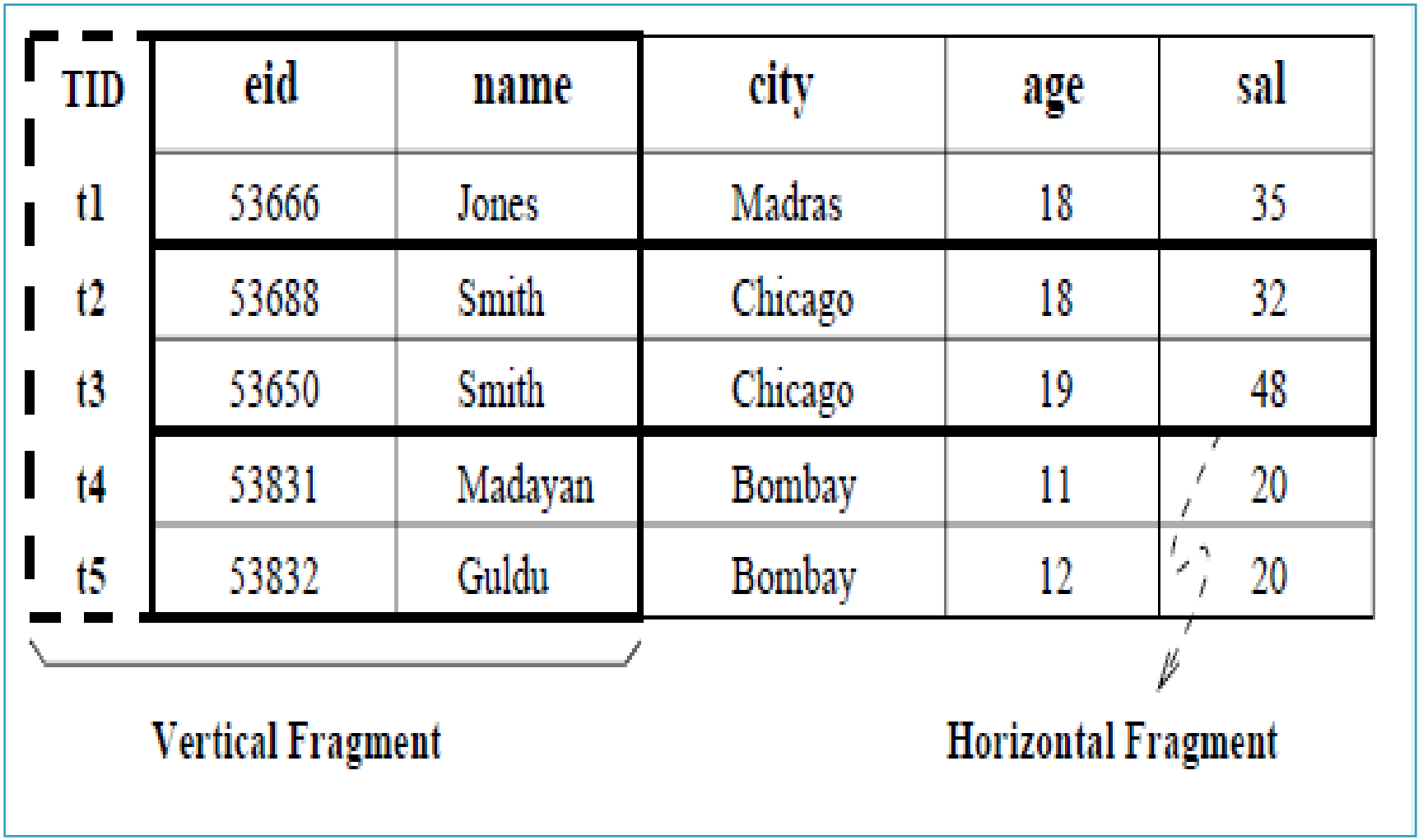
44



Types of Fragmentation:

Horizontal fragmentation: The union of the horizontal fragments must be equal to the original relation. Fragments are usually also required to be disjoint.

Vertical fragmentation: The collection of vertical fragments should be a lossless-join decomposition.


TID
eid
name
city
age
sal
t1
53666
Jones
Madras
18
35
t2
53688
Smith
Chicago
18
32
t3
53650
Smith
Chicago
19
48
t4
53831
Madayan
Bombay
11
20
t5
53832
Guldu
Bombay
12
20
Vertical Fragment
Horizontal Fragment

Replication

- **Replication** means that we store several copies of a relation or relation fragment.

- The motivation for replication is twofold:
  1. **Increased availability of data:**
  2. **Faster query evaluation:**

- Two kinds of replications
  1. synchronous replication
  2. asynchronous replication

DDBMS

I. Introduction to DDBMS
II. Architecture of DDBs
III. Storing data in DDBs
IV. Distributed Catalog Management
V. Distributed Query Processing
VI. Transaction Processing
VII. Distributed concurrency control and recovery

1. Parallel DB /D.S.Jagli
48

4.Distributed Catalog Management

1. Naming Objects

- If a relation is fragmented and replicated, we must be able to uniquely identify each replica of each fragment.
  1. **A *local name field***
  2. **A *birth site field***

2. Catalog Structure

- A centralized system catalog can be used It is vulnerable to failure of the site containing the catalog).
- An alternative is to maintain a copy of a global system catalog. compromises site autonomy,)

4.Distributed Catalog Management

- **A better approach:**
- Each site maintains a local catalog that describes all copies of data stored at that site.

- In addition, the catalog at the birth site for a relation is responsible for keeping track of where replicas of the relation are stored.

DDBMS

I. Introduction to DDBMS
II. Architecture of DDBs
III. Storing data in DDBs
IV. Distributed Catalog management
V. Distributed Query Processing
VI. Transaction Processing
VII. Distributed concurrency control and recovery

1. Parallel DB /D.S.Jagli
51

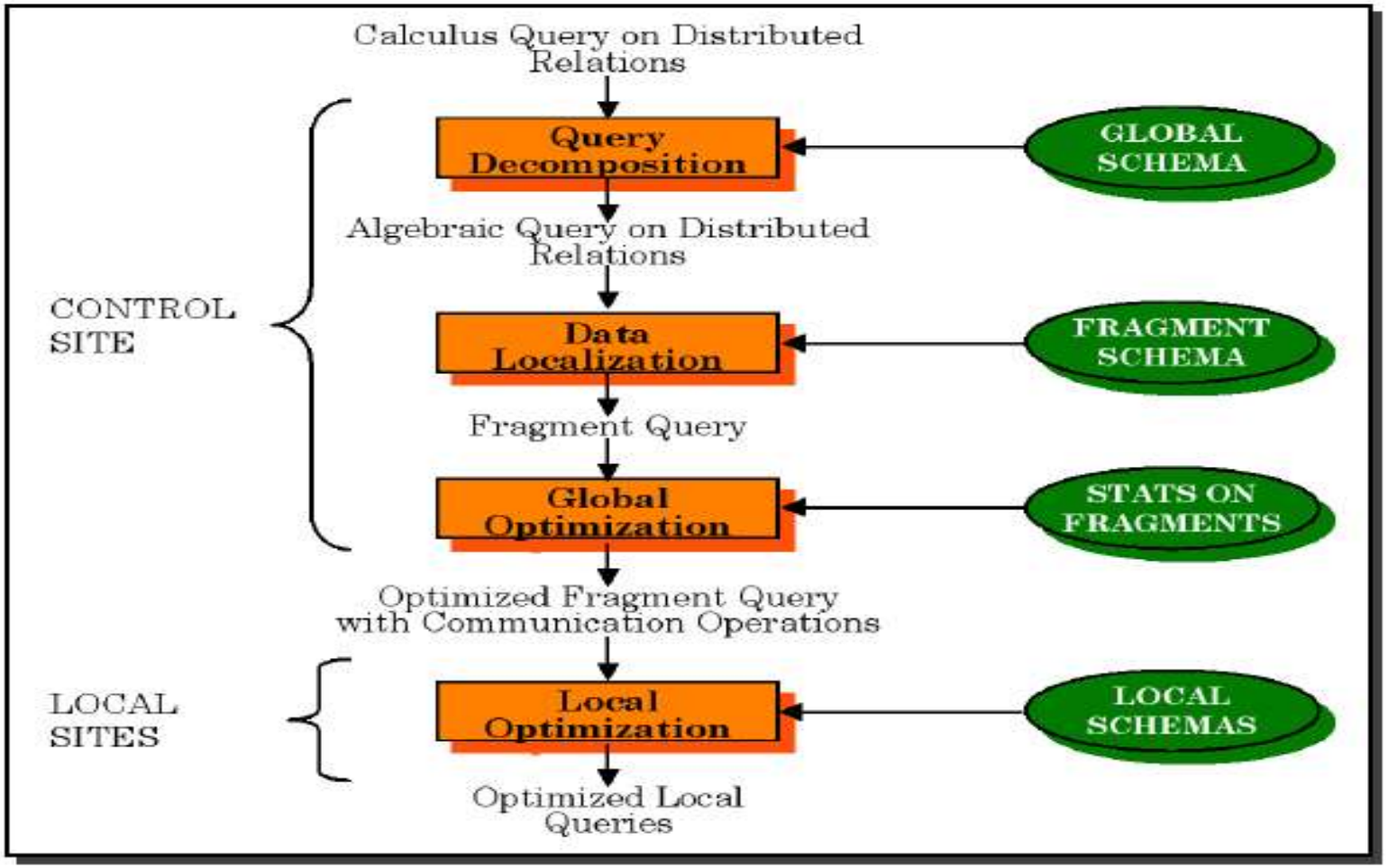
5.Distributed Query Processing

- **Distributed query processing:** Transform a high-level query (of relational calculus/SQL) on a distributed database (i.e., a set of global relations) into an equivalent and efficient lower-level query (of relational algebra) on relation fragments.

- Distributed query processing is more complex

1. **– Fragmentation/replication of relations**
2. **– Additional communication costs**
3. **– Parallel execution**

Distributed Query Processing Steps



5.Distributed Query Processing

- **Sailors(*sid: integer, sname: string, rating: integer, age: real)***
- **Reserves(*sid: integer, bid: integer, day: date, rname: string)***

- Assume Reserves and Sailors relations
  - each tuple of Reserves is 40 bytes long
  - a page can hold 100 Reserves tuples
  - 1,000 pages of such tuples.
  - each tuple of Sailors is 50 bytes long
  - a page can hold 80 Sailors Tuples
  - 500 pages of such tuples
- ***How to estimate the cost?***

5.Distributed Query Processing

- Criteria for measuring the cost of a query evaluation strategy
  - For centralized DBMSs number of disk accesses (# blocks read / written)
  - For distributed databases, additionally
    - The cost of data transmission over the network
    - Potential gain in performance from having several sites processing parts of the query in parallel

5.Distributed query processing

- To estimate the cost of an evaluation strategy, in addition to counting the number of page I/Os.
- we must count the number of pages that are shipped is a communication costs.
- Communication costs is a significant component of overall cost in a distributed database.

1. **Nonjoin Queries in a Distributed DBMS**
2. **Joins in a Distributed DBMS**
3. **Cost-Based Query Optimization**

5.Distributed Query Processing

1.Nonjoin Queries in a Distributed DBMS

- Even simple operations such as scanning a relation, selection, and projection are affected by fragmentation and replication.

```
SELECT S.age
FROM Sailors S
WHERE S.rating > 3 AND S.rating < 7
```

- Suppose that the Sailors relation is horizontally fragmented, with all tuples having a rating less than 5 at **Mumbai** and all tuples having a rating greater than 5 at **Delhi.**

The DBMS must answer this query by evaluating it at both sites and taking the union of the answers.

5.Distributed Query Processing

Eg 1:

```
SELECT avg(age)
FROM Sailors S
WHERE S.rating > 3 AND S.rating < 7
```

- ***taking the union of the answers is not enough***

Eg 2:

```
SELECT S.age
FROM Sailors S
WHERE S.rating > 6
```

- ***taking the union of the answers is not enough***

- **Eg 3:** suppose that the Sailors relation is vertically fragmented, with the *sid* and *rating* fields at MUMBAI and the *sname* and *age* fields at DELHI
- This vertical fragmentation would be a lossy decomposition

5.Distributed Query Processing

Eg 4:the entire Sailors relation is stored at both MUMBAI and DELHI sites.

- ***Where should the query be executed?***
- **2 Joins in a Distributed DBMS**
- **Eg:** the Sailors relation is stored at MUMBAI, and that the Reserves relation is stored at DELHI.
- Joins of relations at different sites can be very expensive.
- JOIN STRATEGY
  1. **Fetch as needed**
  2. **Ship whole**
  3. **Semijoins**
  4. **Bloomjoins**

Which strategy is better for me?

5.Distributed Query Processing

- **1.Fetch As Needed**
- **Page-oriented Nested Loops join:** For each *page* of R, get each *page* of S, and write out matching pairs of tuples <r, s>, where r is in R-page and s is in S-page.

- We could do a page-oriented nested loops join in **MUMBAI** with Sailors as the outer, and for each Sailors page, fetch all Reserves pages from **DELHI**.

- If we cache the fetched Reserves pages in MUMBAI until the join is complete, pages are fetched only once

Fetch As Needed: Transferring the relation piecewise

R

A	B
3	7
1	1
4	6
7	7
4	5
6	2
5	7

S

B	C	D
9	8	8
1	5	1
9	4	⋈
4	3	3
4	2	6
5	7	8

R ⋈ S

A	B	C	D
1	1	5	1
4	5	7	8

QUERY: The query asks for R ⋈ S

5.Distributed Query Processing

- Assume Reserves and Sailors relations
  - **each tuple of Reserves is 40 bytes long**
  - **a page can hold 100 Reserves tuples**
  - **1,000 pages of such tuples.**
  - each tuple of Sailors is 50 bytes long
  - a page can hold 80 Sailors Tuples
  - 500 pages of such tuples
- *td* is cost to read/write page; ts is cost to ship page.
- The cost is = 500*td* to scan Sailors
- for each Sailors page, the cost of scanning shipping all of Reserves, which is =1000(*td* + *ts*).
- The **total cost** is = 500*td* + 500, 000(*td* + *ts*).

5.Distributed Query Processing

- This cost depends on the size of the result.
- The cost of shipping the result is greater than the cost of shipping both Sailors and Reserves to the query site.

- **2.Ship to One Site (sort-merge-join)**
- The cost of scanning and shipping Sailors, saving it at DELHI, and then doing the join at DELHI is

=500(2*td* + *ts*) + (Result)*td*,

- The cost of shipping Reserves and doing the join at MUMBAI is

=1000(2*td*+*ts*)+(result)*td*.

Ship Whole: Transferring the complete relation

R

A	B
3	7
1	1
4	6
7	7
4	5
6	2
5	7

S

B	C	D
9	8	8
1	5	1
9	4	2
4	3	3
4	2	6
5	7	8

R ⋈ S

A	B	C	D
1	1	5	1
4	5	7	8

QUERY: The query asks for R ⋈ S

5.Distributed Query Processing

- Ship Whole vs Fetch as needed:
- Fetch as needed results in a high number of messages
- Ship whole results in high amounts of transferred data
- *Note: Some tuples in Reserves do not join with any tuple in Sailors, we could avoid shipping them.*

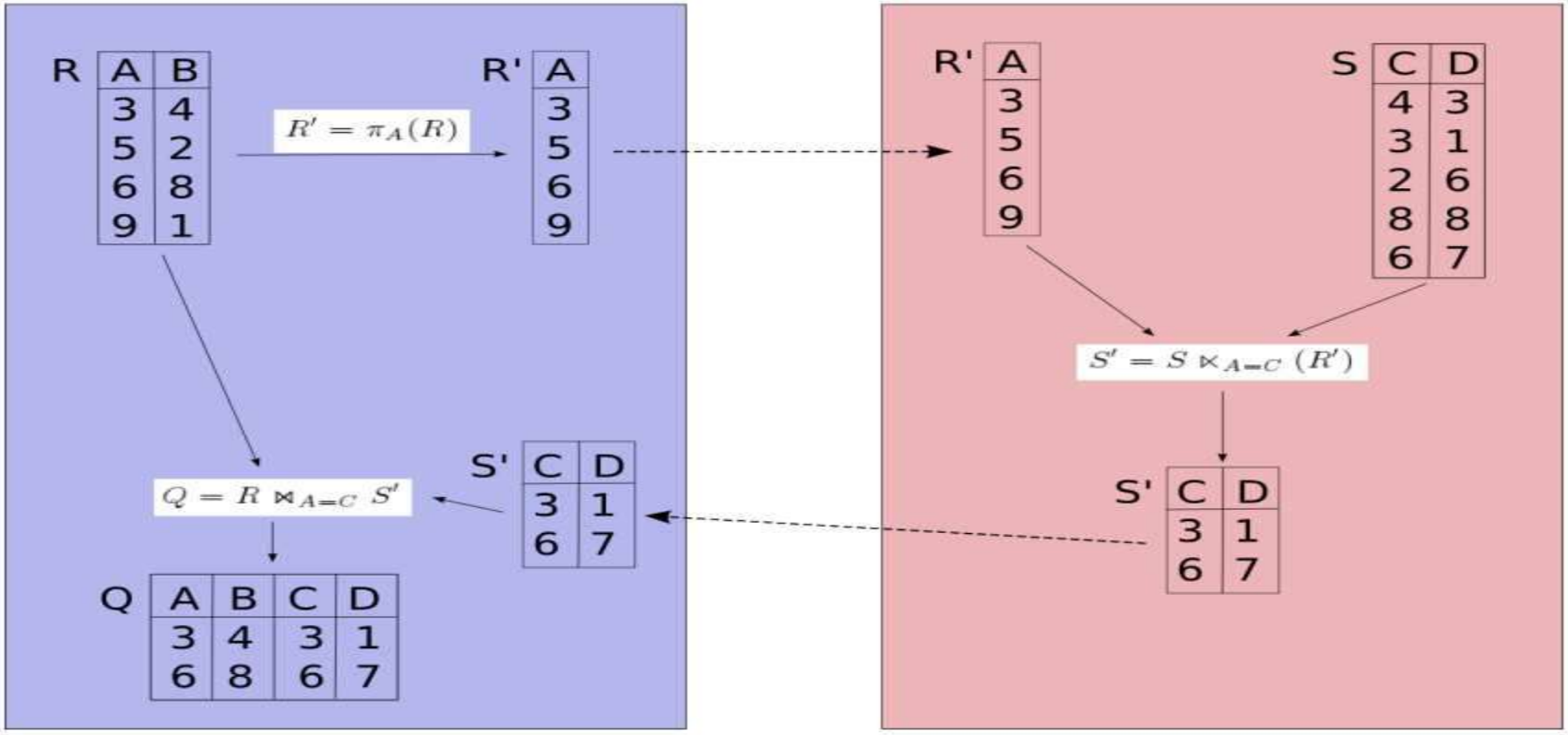
3.Semijoins and Bloomjoins

Semijoins: 3 steps:

1. At MUMBAI, compute the **projection** of Sailors onto the join columns i.e *sid* and ship this projection to DELHI.
2. 2. At DELHI, compute the **natural join** of the projection received from the first site with the Reserves relation.
3. The result of this join is called the **reduction** of Reserves with respect to Sailors. ship the reduction of Reserves to MUMBAI.
4. 3. At MUMBAI, compute the join of the reduction of Reserves with Sailors.

Semijoin:

- Semijoin: Requesting all join partners in just one step.



5.Distributed query processing

- **Bloomjoins:** 3 steps:

1. At MUMBAI, A bit-vector of (some chosen) size k is computed by hashing each tuple of Sailors into the range 0 to $k-1$ and setting bit i to 1. if some tuple hashes to i, and 0 otherwise then ship this to DELHI

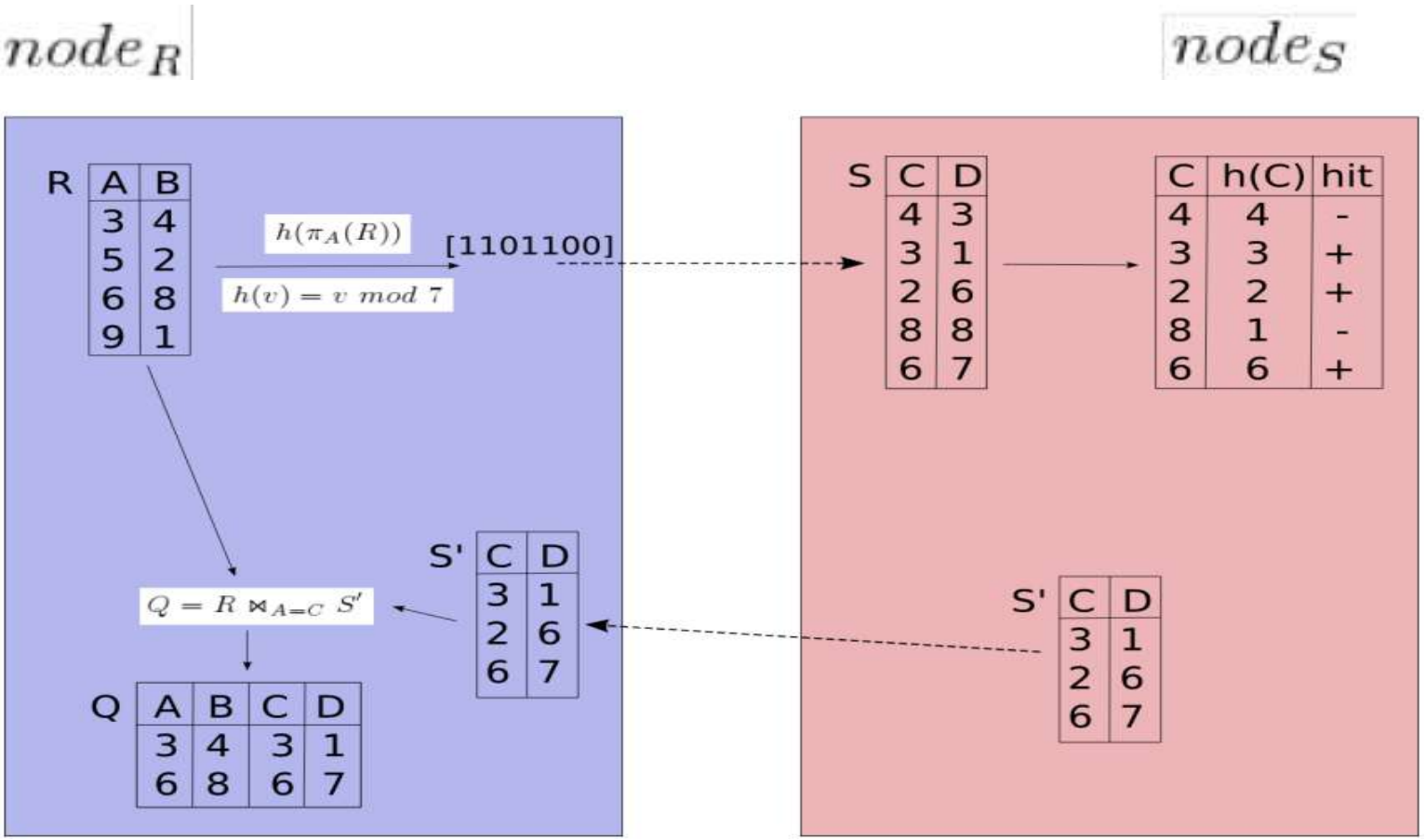
2. At DELHI, the reduction of Reserves is computed by hashing each tuple of Reserves (using the *sid* field) into the range 0 to $k-1$, using the same hash function used to construct the bit-vector, and discarding tuples whose hash value i corresponds to a 0 bit.ship the reduction of Reserves to MUMBAI.

3. At MUMBAI, compute the join of the reduction of Reserves with Sailors.

Bloom join:

- Bloom join:
- Also known as bit-vector join
- Avoiding to transfer all join attribute values to the other node
- Instead transferring a bitvector B[1 : : : n]
- Transformation
  - Choose an appropriate hash function h
  - Apply h to transform attribute values to the range [1 : : : n]
  - Set the corresponding bits in the bitvector B[1 : : : n] to

Bloom join:


$node_R$
$node_S$
R
A B
3 4
5 2
6 8
9 1
$h(\pi_A(R))$
$h(v) = v \bmod 7$
[1101100]
S
C D
4 3
3 1
2 6
8 8
6 7
C h(C) hit
4 4 -
3 3 +
2 2 +
8 1 -
6 6 +
S'
C D
3 1
2 6
6 7
$Q = R \bowtie_{A=C} S'$
Q
A B C D
3 4 3 1
6 8 6 7
S'
C D
3 1
2 6
6 7

Cost-Based Query Optimization

- optimizing queries in a distributed database poses the following additional challenges:

  - Communication costs must be considered. If we have several copies of a relation, we must also decide which copy to use.

  - If individual sites are run under the control of different DBMSs, the autonomy of each site must be respected while doing global query planning.

Cost-Based Query Optimization

- Cost-based approach; consider all plans, pick cheapest; similar to centralized optimization.
  - ***Difference 1: Communication costs must be considered.***
  - ***Difference 2: Local site autonomy must be respected.***
  - ***Difference 3: New distributed join methods.***
- Query site constructs global plan, with suggested local plans describing processing at each site.
- ***If a site can improve suggested local plan, free to do so.***

6.DISTRIBUTED TRANSACTIONS PROCESSING

- A given transaction is submitted at some one site, but it can access data at other sites.
- When a transaction is submitted at some site, the transaction manager at that site breaks it up into a collection of one or more subtransactions that execute at different sites.
-

7.Distributed Concurrency Control

- **Lock management** can be distributed across sites in many ways:

1. **Centralized:** A single site is incharge of handling lock and unlock requests for all objects.
2. **Primary copy:** One copy of each object is designated as the primary copy.
   - All requests to lock or unlock a copy of this object are handled by the lock manager at the site where the primary copy is stored.

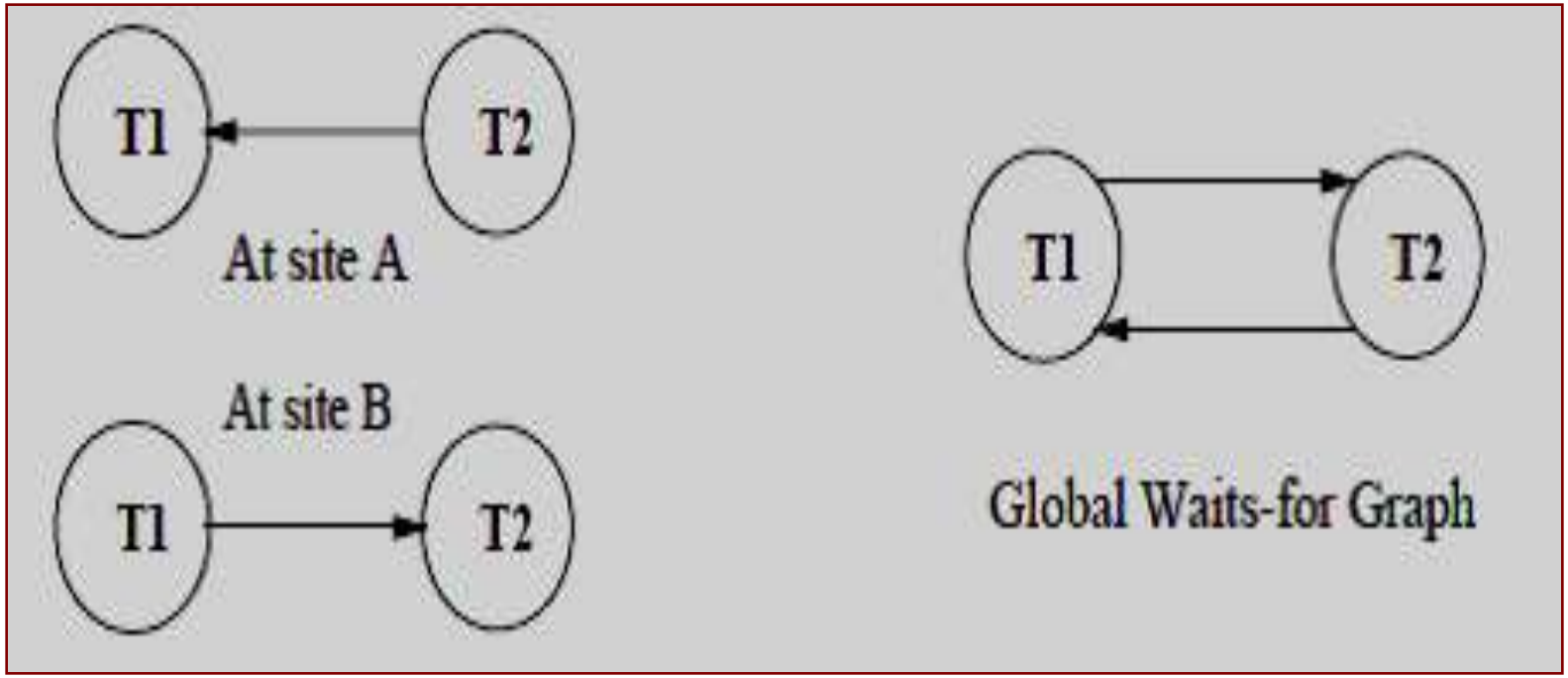
3. **Fully distributed:** Requests to lock or unlock a copy of an object stored at a site are handled by the lock manager at the site where the copy is stored.

7.Distributed Concurrency Control

- **Distributed Deadlock Detection:**
- Each site maintains a local waits-for graph.
- A global deadlock might exist even if the local graphs contain no cycles:

- **Three solutions:**
- **Centralized:** send all local graphs to one site ;
- **Hierarchical:** organize sites into a hierarchy and send local graphs to parent in the hierarchy;
- **Timeout** : abort Xact if it waits too long.

- **Phantom Deadlocks:** delays in propagating local information might cause the deadlock detection algorithm to identify `**deadlocks'** that do not really exist. Such situations are called **phantom deadlocks**



7.Distributed Recovery

- Recovery in a distributed DBMS is more complicated than in a centralized DBMS for the following reasons:

1. New kinds of failure can arise:
   - failure of communication links.
   - failure of a remote site at which a subtransaction is executing.
2. Either all subtransactions of a given transaction must commit, or none must commit.
3. This property must be guaranteed in spite of any combination of site and link failures.

7.Distributed Recovery

- **Two-Phase Commit (2PC):**

- Site at which Xact originates is **coordinator**;
- other sites at which it executes subXact are **subordinates**.
- When the user decides to commit a transaction:
  - The commit command is sent to the coordinator for the transaction.
  - This initiates the 2PC protocol:

7.Distributed Recovery(2pc)

1. Coordinator sends prepare msg to each subordinate.

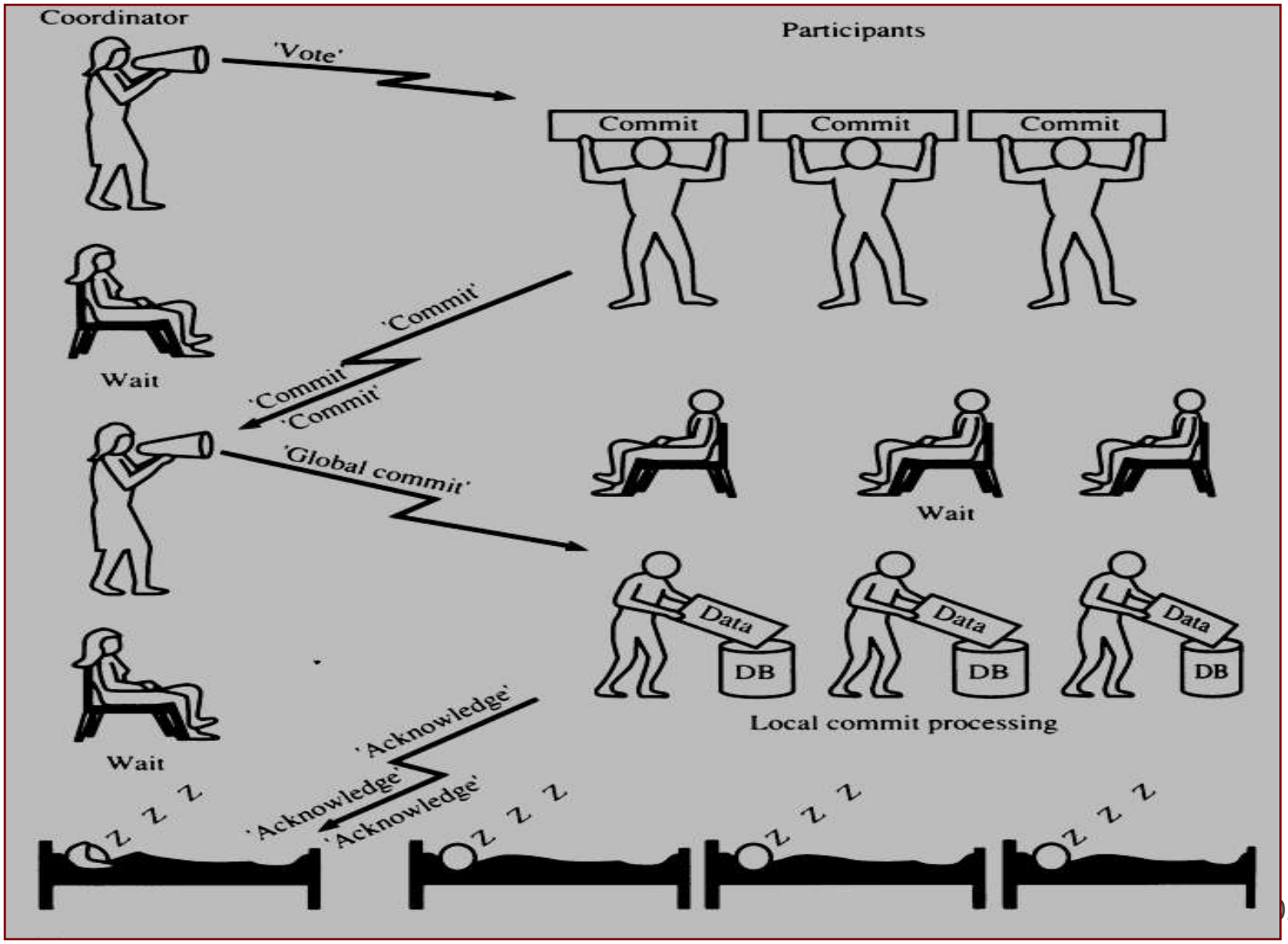
2. Subordinate force-writes an abort or prepare log record and then sends a no or yes msg to coordinator.

3. If coordinator gets all yes votes, force-writes a commit log record and sends commit msg to all subs. Else, force-writes abort log rec, and sends abort msg.

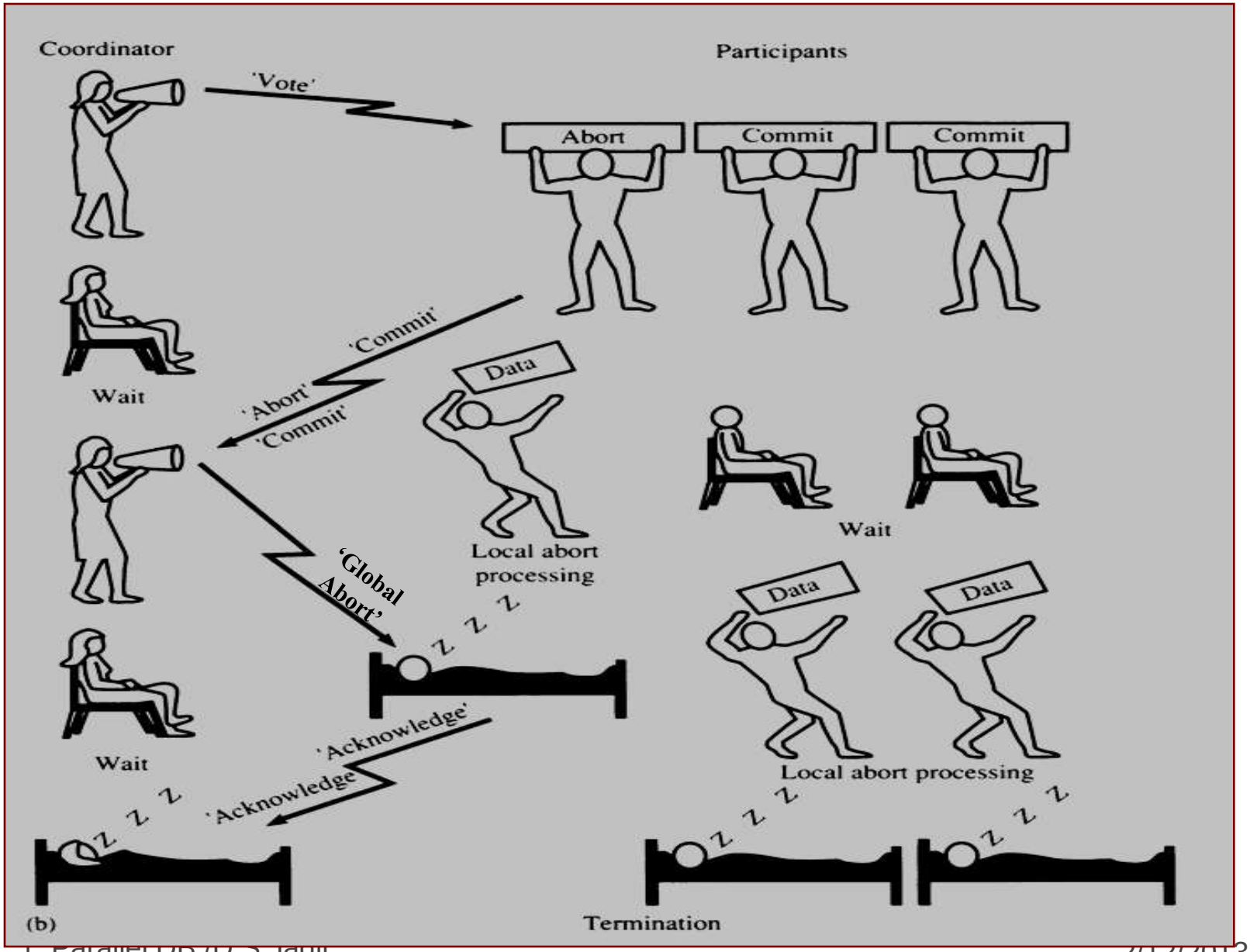
4. Subordinates force-write abort/commit log rec based on msg they get, then send ack msg to coordinator.

5. Coordinator writes end log rec after getting ack msg from all subs

TWO-PHASE COMMIT (2PC) - commit



TWO-PHASE COMMIT (2PC) - ABORT



7.Distributed Recovery(3pc)

- **Three-Phase Commit**

1. A commit protocol called **Three-Phase Commit (3PC)** can avoid blocking even if the coordinator site fails during recovery.

2. The basic idea is that when the coordinator sends out *prepare messages* and receives *yes* votes from all subordinates.

3. It sends all sites a *precommit* message, rather than a *commit* message.
4. When a sufficient number more than the maximum number of failures that must be handled of *acks* have been received.
5. The coordinator force-writes a *commit* log record and sends a *commit message* to all subordinates.

Distributed Database

- ***Advantages:***
  - Reliability
  - Performance
  - Growth (incremental)
  - Local control
  - Transparency
- ***Disadvantages:***
  - Complexity of Query opt.
  - Concurrency control
  - Recovery
  - Catalog management

Thank you

Queries?