# Parallel Databases

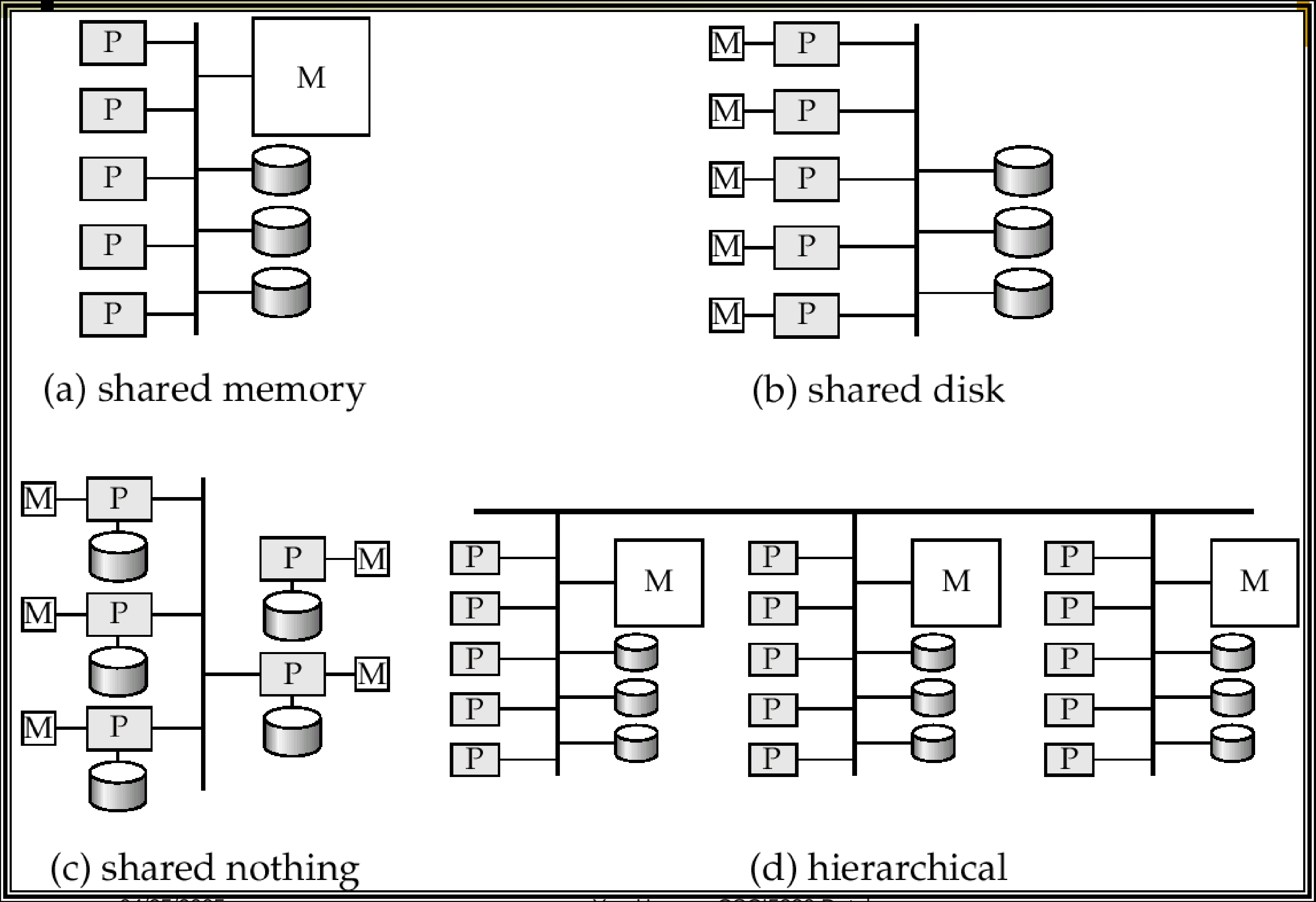
## Introduction

- 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
- Large-scale parallel database systems increasingly used for:
  - storing large volumes of data
  - processing time-consuming decision-support queries
  - providing high throughput for transaction processing

## Parallelism in Databases

- Data can be partitioned across multiple disks for parallel I/O.
- Individual relational operations (e.g., sort, join, aggregation) can be executed in parallel
  - data can be partitioned and each processor can work independently on its own partition.
- Queries are expressed in high level language (SQL, translated to relational algebra)
  - makes parallelization easier.
- Different queries can be run in parallel with each other. Concurrency control takes care of conflicts.
- Thus, databases naturally lend themselves to parallelism.

# Parallel Database Architectures



## Parallel Database Architectures

- **Shared memory** -- processors share a common memory
- **Shared disk** -- processors share a common disk
- **Shared nothing** -- processors share neither a common memory nor common disk
- **Hierarchical** -- hybrid of the above architectures

## Shared Memory

- Extremely efficient communication between processors
- Downside –is not scalable beyond 32 or 64 processors Widely used for lower degrees of parallelism (4 to 8).

## Shared Disk

- Examples: IBM Sysplex and DEC clusters (now part of Compaq/HP) running Rdb (now Oracle Rdb) were early commercial users
- Downside: bottleneck at interconnection to the disk subsystem.
- Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.

## Shared Nothing

- Examples: Teradata, Tandem, Oracle-n CUBE
- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing.
- Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.
- Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.

## Hierarchical

- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.

## Apple Supercomputer

- “Soon after the announcement, Varadarajan took delivery of his very first PowerBook laptop running Mac OS X. Within days, he placed an order for the 1100 dual processor, 2.0 GHz Power Mac G5 computers that now drive Virginia Tech’s new supercomputer. Smart choice: In November of 2003 the giant system — named System X — became the third fastest supercomputer in the world.

  System X is radically different from traditional, high-performance supercomputers. Unlike most, it is based on a “supercluster” of Power Mac G5 computers, each of which has 4GB of main memory, and 160GB of serial ATA storage. Not only is System X the world’s fastest, most powerful “home-built” supercomputer, it quite possibly has the cheapest price/performance of any supercomputer on the TOP500 list.”

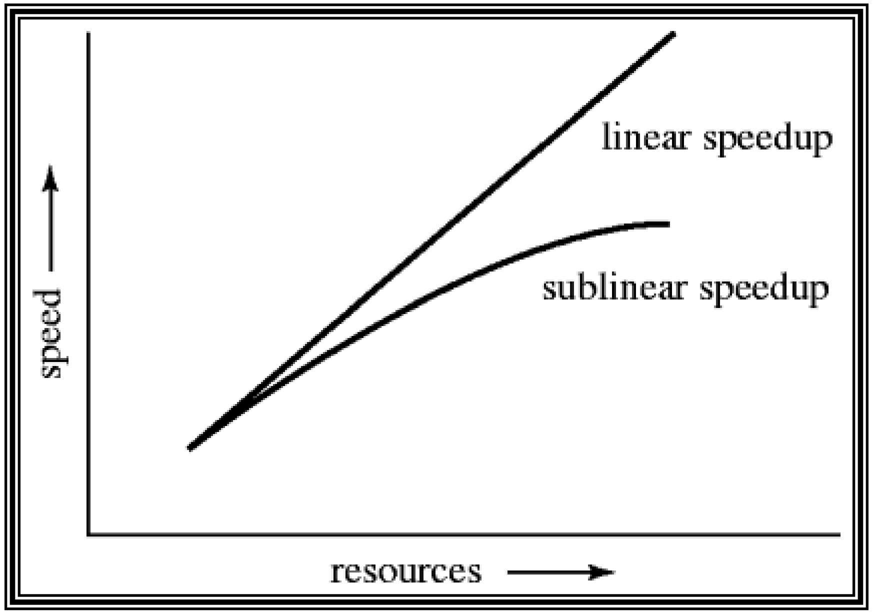
  --- From Apple Website

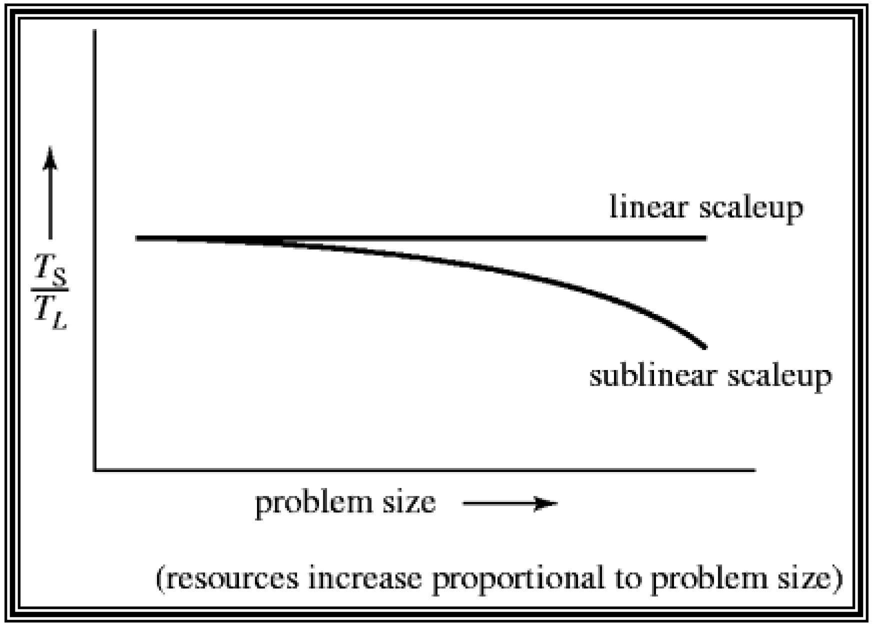
## Parallel Level

- A **coarse-grain parallel** machine consists of a small number of powerful processors
- A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.

# Parallel System Performance Measure

- **Speedup**: = $\dfrac{\textit{small system elapsed time}}{\textit{large system elapsed time}}$
- **Scaleup**: = $\dfrac{\textit{small system small problem elapsed time}}{\textit{big system big problem elapsed time}}$





## Database Performance Measures

- **throughput** --- the number of tasks that can be completed in a given time interval
- **response time** --- the amount of time it takes to complete a single task from the time it is submitted

## Batch and Transaction Scaleup

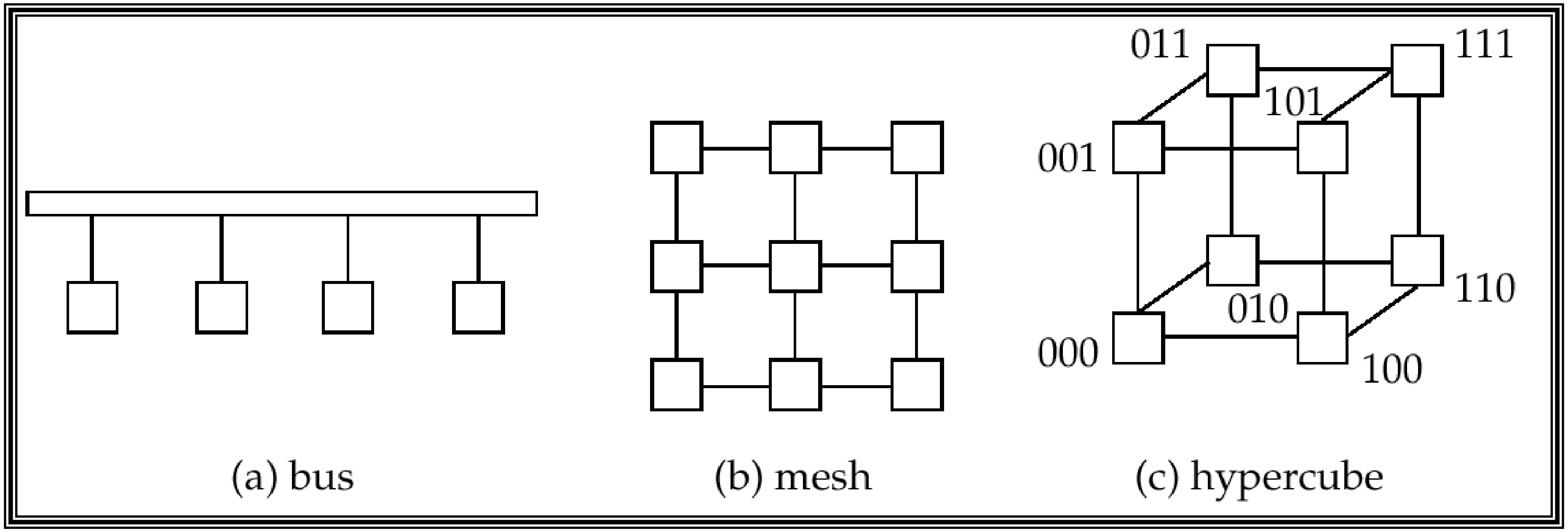
- **Batch scaleup:**
  - A single large job; typical of most database queries and scientific simulation.
  - Use an *N*-times larger computer on *N*-times larger problem.
- **Transaction scaleup**:
  - Numerous small queries submitted by independent users to a shared database; typical transaction processing and timesharing systems.
  - *N*-times as many users submitting requests (hence, *N*-times as many requests) to an *N*-times larger database, on an *N*-times larger computer.
  - Well-suited to parallel execution.

## Factors Limiting Speedup and Scaleup

Speedup and scaleup are often sublinear due to:

- **Startup costs**: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.
- **Interference**: Processes accessing shared resources (e.g.,system bus, disks, or locks) compete with each other
- **Skew**: Overall execution time determined by **slowest** of parallely executing tasks.

## Interconnection Architectures



(a) bus (b) mesh (c) hypercube

## Parallel Database Issues

- Data Partitioning
- Parallel Query Processing

## I/O Parallelism

- Horizontal partitioning – tuples of a relation are divided among many disks such that each tuple resides on one disk.
- Partitioning techniques (number of disks = *n*):
  - Round-robin
  - Hash partitioning
  - Range partitioning

## Skew

- The distribution of tuples to disks may be **skewed**
  - **Attribute-value skew.**
    - Some values appear in the partitioning attributes of many tuples
  - **Partition skew**.
    - Too many tuples to some partitions and too few to others
- Round robin handles skew well
- Hashing and ranging may result in skew

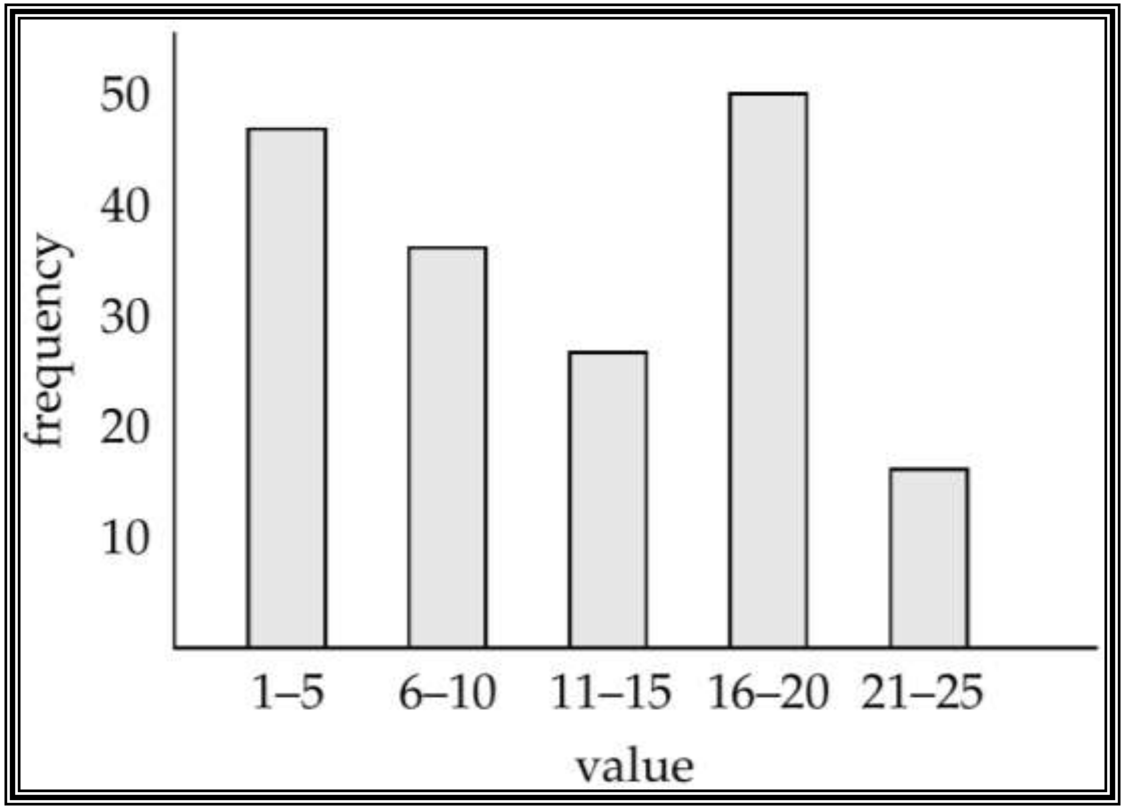
## Typical Database Query Types

- Sequential scan
- Point query
- Range query

## Comparison of Partitioning Techniques

| | Round Robin | Hashing | Range |
|---|---|---|---|
| Sequential Scan | Best/good parallelism | Good | Good |
| Point Query | Difficult | Good for hash key | Good for range vector |
| Range Query | Difficult | Difficult | Good for range vector |

## Handling Skew using Histograms



## Interquery Parallelism

- Queries/transactions execute in parallel with one another.
- Increase throughput

## Intraquery Parallelism

- Execution of a single query in parallel on multiple processors/disks
- Speed up long-running queries.
- Two complementary forms of intraquery parallelism :
  - **Intraoperation Parallelism** – parallelize the execution of each individual operation in the query.
  - **Interoperation Parallelism** – execute the different operations in a query expression in parallel.

## Parallel Processing of Relational Operations

- Our discussion of parallel algorithms assumes:
  - *read-only* queries
  - shared-nothing architecture
  - *n* processors, $P_0$, ..., $P_{n-1}$, and *n* disks $D_0$, ..., $D_{n-1}$, where disk $D_i$ is associated with processor $P_i$.

## Parallel Sort

**Range-Partitioning Sort**

- Choose processors $P_0, \ldots, P_m$, where $m \leq n - 1$ to do sorting.
- Create range-partition vector with m entries, on the sorting attributes
- Redistribute the relation using range partitioning
  - all tuples that lie in the ith range are sent to processor $P_i$
  - $P_i$ stores the tuples it received temporarily on disk $D_i$.
  - This step requires I/O and communication overhead.
- Each processor $P_i$ sorts its partition of the relation locally.
- Each processors executes same operation (sort) in parallel with other processors, without any interaction with the others (**data parallelism)**.
- Final merge operation is trivial

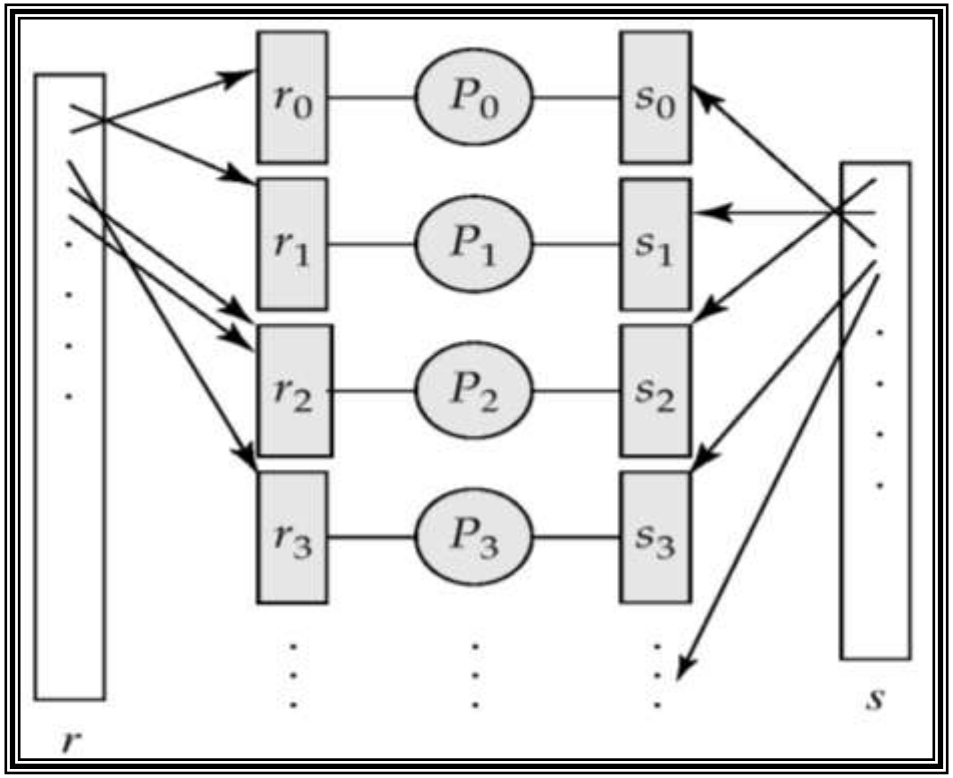
## Parallel Sort (Cont.)

**Parallel External Sort-Merge**

- Assume the relation has already been partitioned among disks $D_0, \ldots, D_{n-1}$ (in whatever manner).
- Each processor $P_i$ locally sorts the data on disk $D_i$.
- The sorted runs on each processor are then merged to get the final sorted output.
- Parallelize the merging of sorted runs as follows:
  - The sorted partitions at each processor $P_i$ are range-partitioned across the processors $P_0, \ldots, P_{m-1}$.
  - Each processor $P_i$ performs a merge on the streams as they are received, to get a single sorted run.
  - The sorted runs on processors $P_0, \ldots, P_{m-1}$ are concatenated to get the final result.

## Partitioned Join

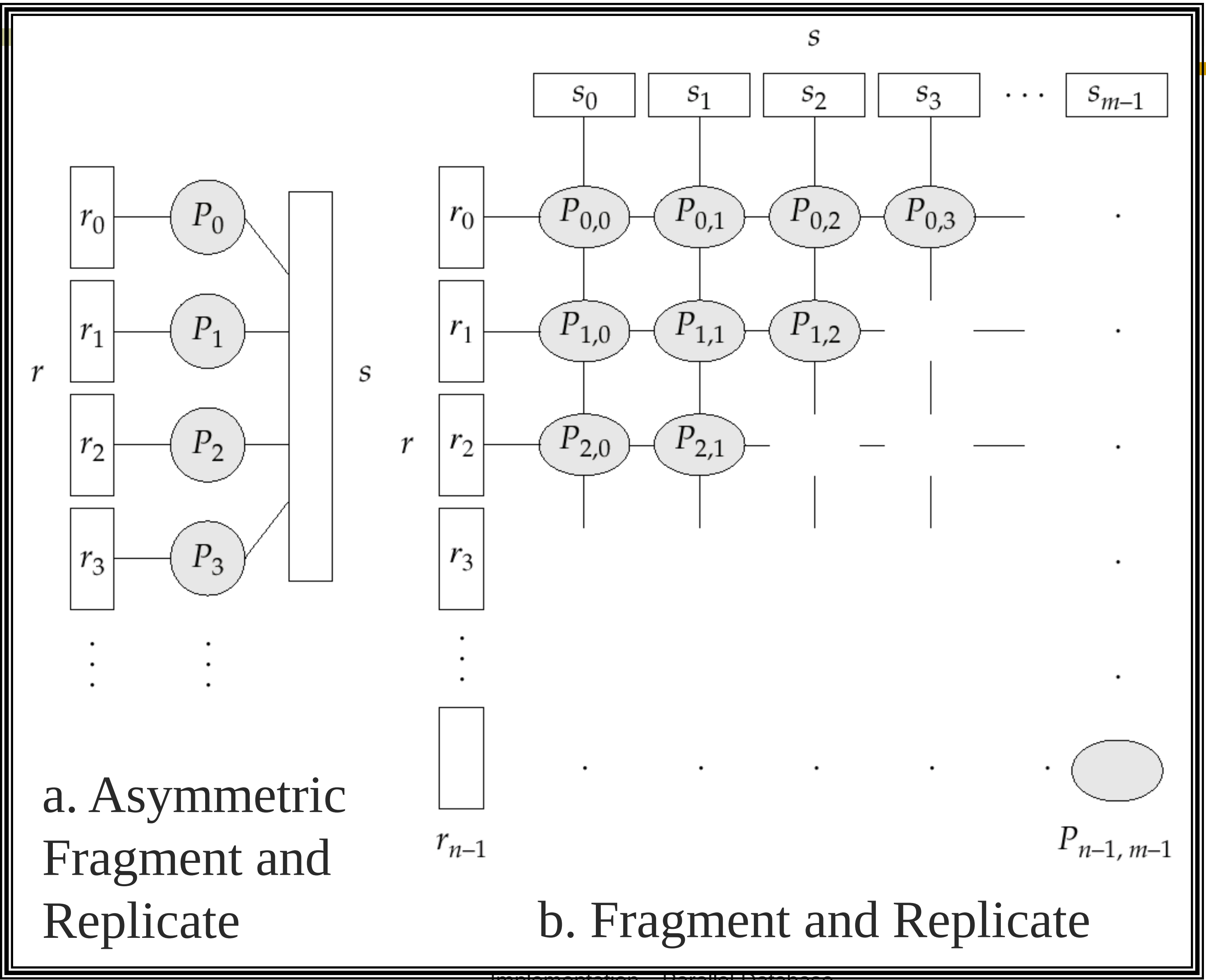
- For equi-joins and natural joins, it is possible to *partition* the two input relations across the processors, and compute the join locally at each processor.



## Fragment-and-Replicate Join

- Partitioning not possible for some join conditions
  - e.g., non-equijoin conditions, such as r.A > s.B.
- For joins were partitioning is not applicable, parallelization  can be accomplished by **fragment and replicate** technique

## Depiction of Fragment-and-Replicate Joins



## Interoperator Parallelism

- **Pipelined parallelism**
  - Consider a join of four relations
    - $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$
  - Set up a pipeline that computes the three joins in parallel
    - Let P1 be assigned the computation of
      temp1 = $r_1 \bowtie r_2$
    - And P2 be assigned the computation of temp2 = temp1 $\bowtie r_3$
    - And P3 be assigned the computation of temp2 $\bowtie r_4$
  - Each of these operations can execute in parallel, sending result tuples it computes to the next operation even as it is computing further results
    - Provided a pipelineable join evaluation algorithm (e.g. indexed nested loops join) is used

## Independent Parallelism

- **Independent parallelism**
  - Consider a join of four relations

    $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$
    - Let P1 be assigned the computation of

      temp1 = $r_1 \bowtie r_2$
    - And P2 be assigned the computation of temp2 = $r_3 \bowtie r_4$
    - And P3 be assigned the computation of temp1 $\bowtie$ temp2
    - P1 and P2 can work **independently in parallel**
    - P3 has to wait for input from P1 and P2
      - Can pipeline output of P1 and P2 to P3, combining independent parallelism and pipelined parallelism
  - Does not provide a high degree of parallelism
    - useful with a lower degree of parallelism.
    - less useful in a highly parallel system,