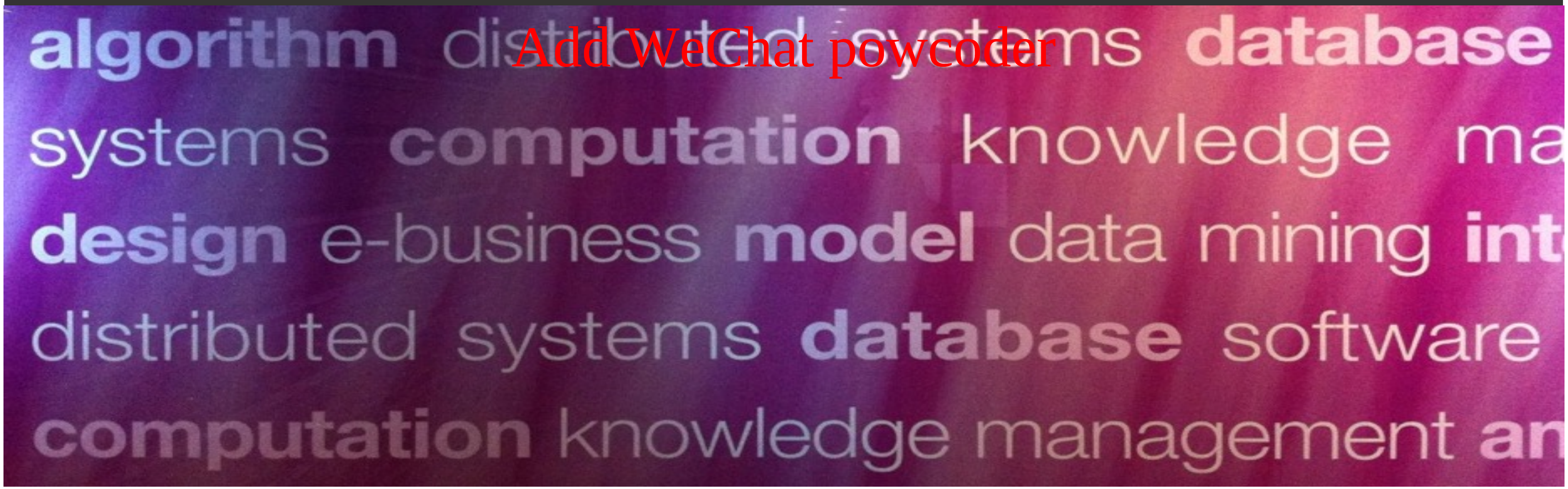
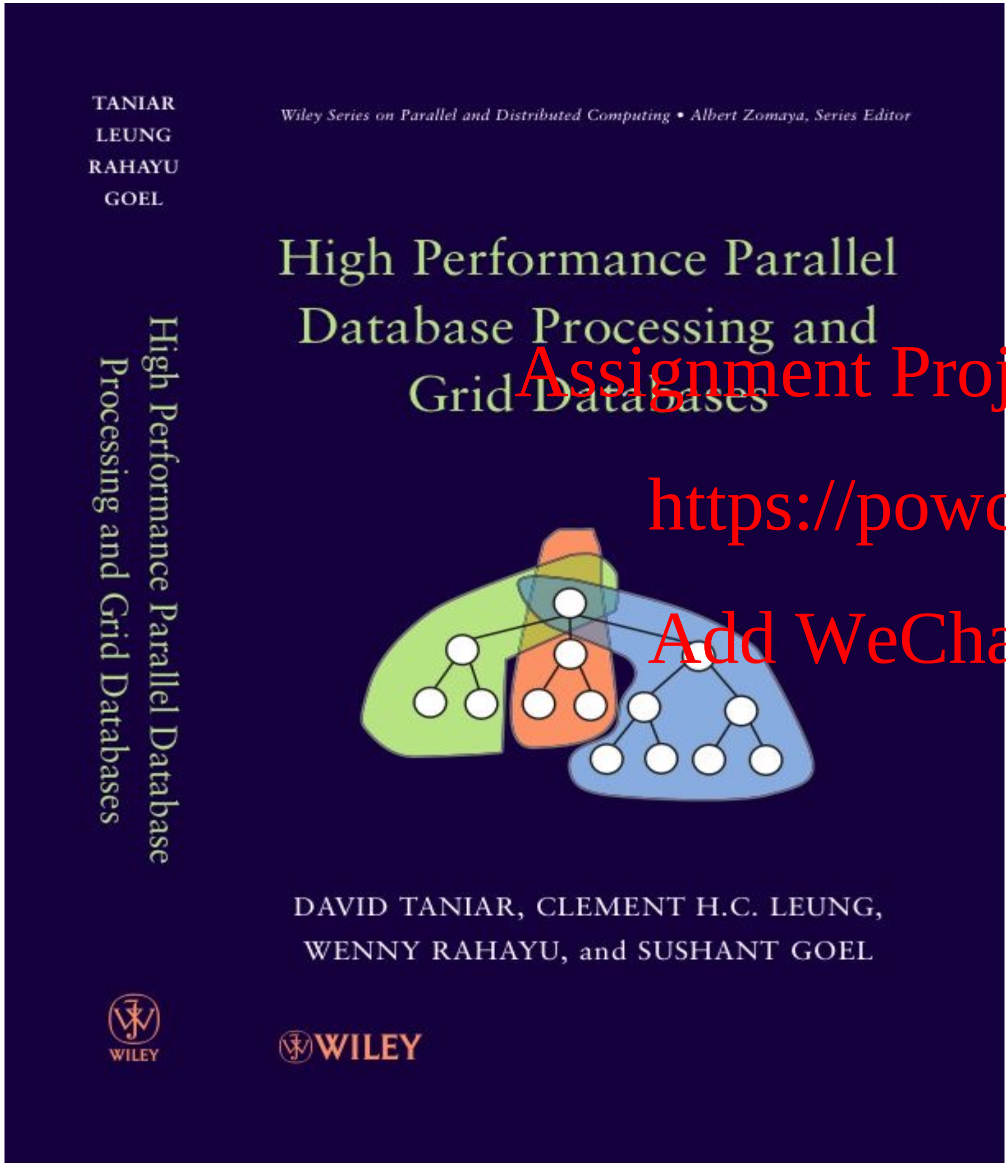
MONASH University

Information Technology

FIT5202 (Volume I - Introduction)

Week 2a – Introduction to Parallel Databases





Chapter 1 Introduction

1.1 A Brief Overview - Parallel Databases and Grid Databases
1.2 Parallel Query Processing: Motivations
1.3 Parallel Query Processing: Objectives
1.4 Forms of Parallelism
1.5 Parallel Database Architectures
1.6 Grid Database Architecture
1.7 Structure of this Book
1.8 Summary
1.9 Bibliographical Notes
1.10 Exercises

Revision

- **Exercise 1 (FLUX Quiz)**
  - Using the freeway analogy, number of cars that can pass through the freeway (M1: Monash Freeway) during the morning peak hour from 7 to 9am is called:

  - A. Throughput
  - B. Response Time
  - C. None of the above
  - D. A and B

Revision

- **Exercise 2 (FLUX Quiz)**
  - Using the freeway analogy, the duration I take to drive my car to go to work on a freeway (say M1 Monash Freeway) from the Burke Road entrance to the Blackburn Road exit is called:

  - A. Throughput
  - B. Response Time
  - C. None of the above
  - D. A and B

1.3. Objectives (cont'd)

- **Parallel Obstacles**
  - Start-up and Consolidation costs,
  - Interference and Communication, and
  - Skew

1.3. Objectives (cont'd)

- **Start-up and Consolidation**
  - Start up: initiation of multiple processes
  - Consolidation: the cost for collecting results obtained from each processor by a host processor

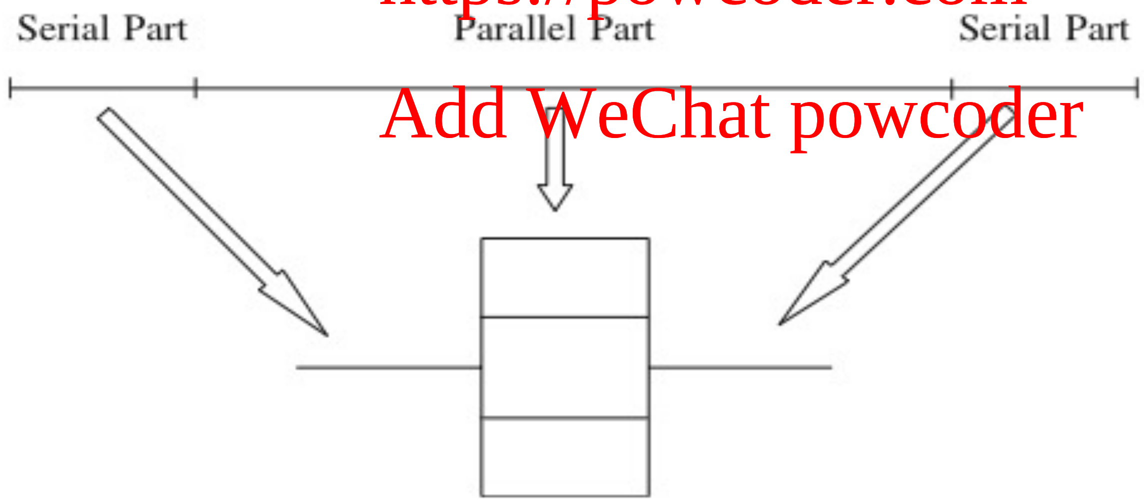


Figure 1.3 Serial part vs. parallel part

1.3. Objectives (cont'd)

- **Interference and Communication**
  - Interference: competing to access shared resources
  - Communication: one process communicating with other processes, and often one has to wait for others to be ready for communication (i.e. waiting time).

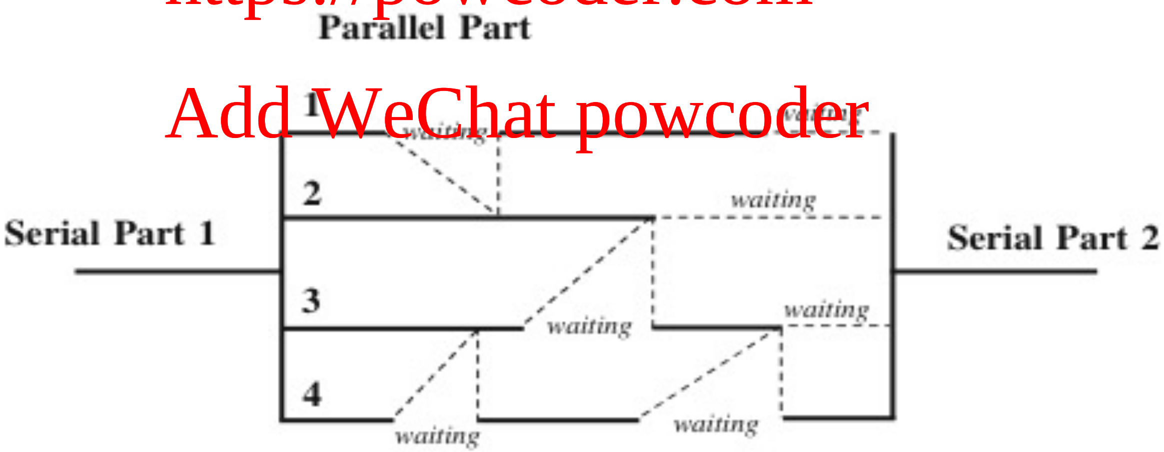


Figure 1.4 Waiting period

- **Exercise 3 (Flux Quiz)**
  - There is a job that will take 1 hour to complete, if this is done by 1 processor.
  - The serial part of this job is 10%
  - There are 4 processors to use in this job, but each processor will have an overhead of 20% due to waiting time, communication time, etc.
  - What type of **speed up** do we get?

1.3. Objectives (cont'd)

- **Skew**
  - Unevenness of workload
  - Load balancing is one of the critical factors to achieve linear speed up

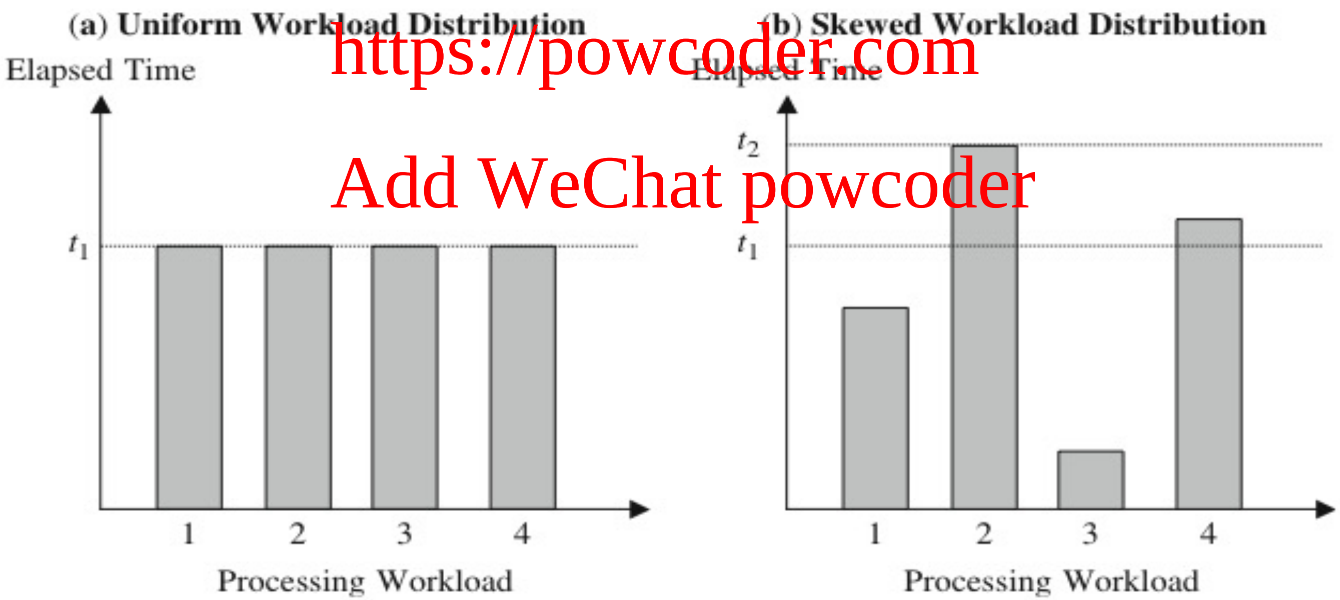


Figure 1.5 Balanced workload vs. unbalanced workload (skewed)

- **Exercise 4 (FLUX Quiz)**
  - Pick a number (between 1 and 4)
  - A. 1
  - B. 2
  - C. 3
  - D. 4

- **Exercise 5 (FLUX Quiz)**
  - Pick a number again (between 1 and 4)
  - A. 1
  - B. 2
  - C. 3
  - D. 4

- **Exercise 6 (FLUX Quiz)**
  - Pick a number again (between 1 and 4)
  - A. 1
  - B. 2
  - C. 3
  - D. 4

1.3. Objectives (cont'd)

- **Skew**
  - *Zipf* distribution model to model skew. Measured in terms of different sizes of fragments allocated to the processors

$$|R_i| = \frac{|R|}{i^{\theta} \times \sum_{j=1}^{N} \frac{1}{j^{\theta}}} \qquad \text{where } 0 \leq \theta \leq 1 \tag{2.1}$$

  - The symbol θ denotes the degree of skewness, where $\theta = 0$ indicates no skew, and **θ = 1 indicates highly skewed**

  - **|*R*|** is number of records in the table, **|*Ri*| is number of records in processor *i***, and ***N*** is number of processor (*j* is a loop counter, starting from 1 to *N*)

  - Example: |*R*|=100,000 records, *N*=8 processors

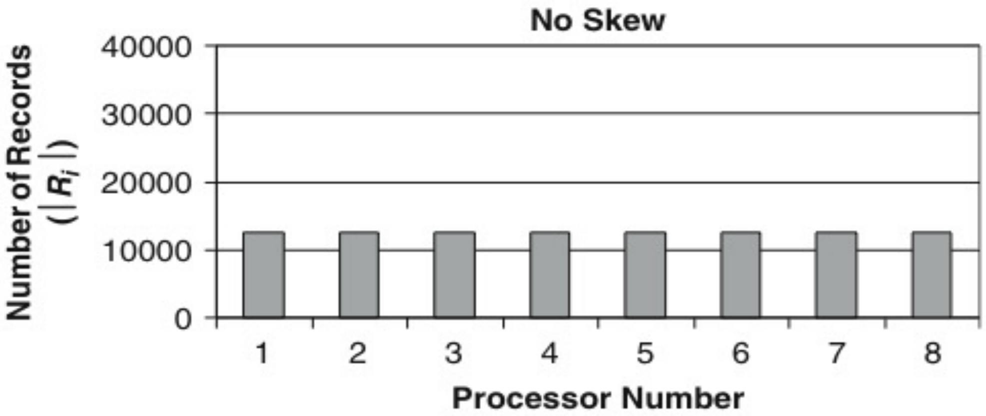


Figure 2.1 Uniform distribution (no skew)

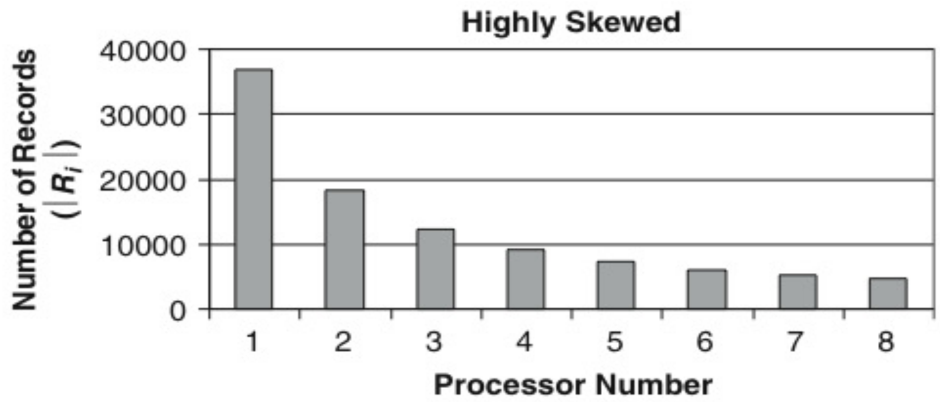


Figure 2.2 Highly skewed distribution

1.3. Objectives (cont'd)

- **No skew vs. highly skewed**

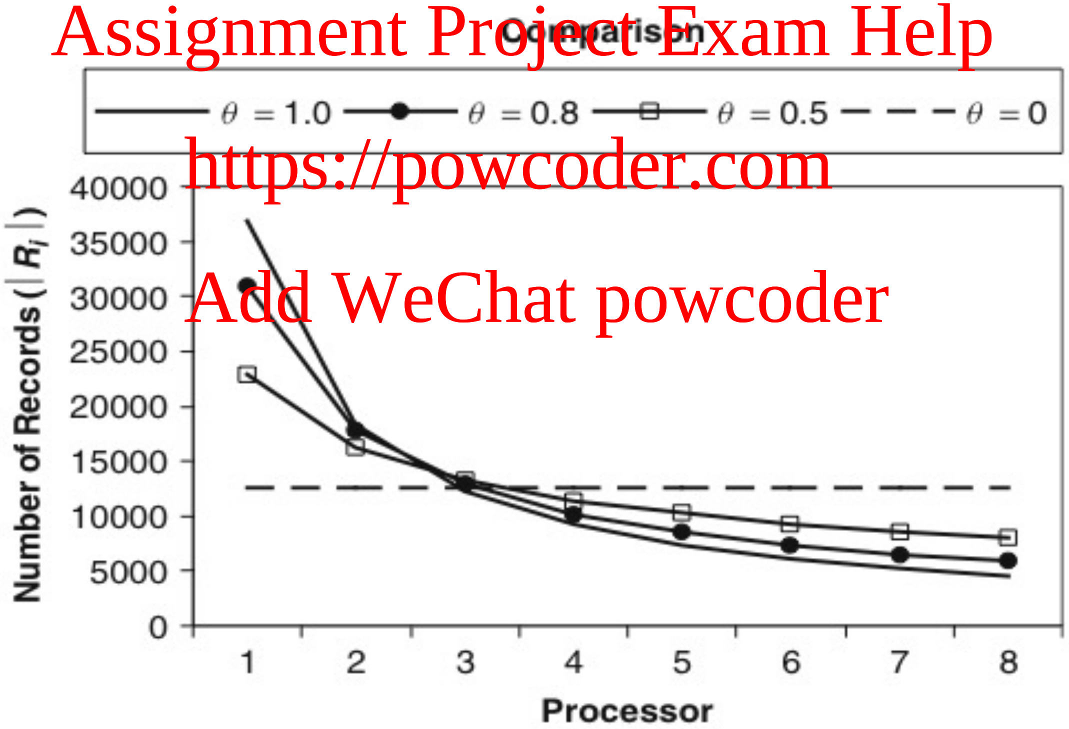
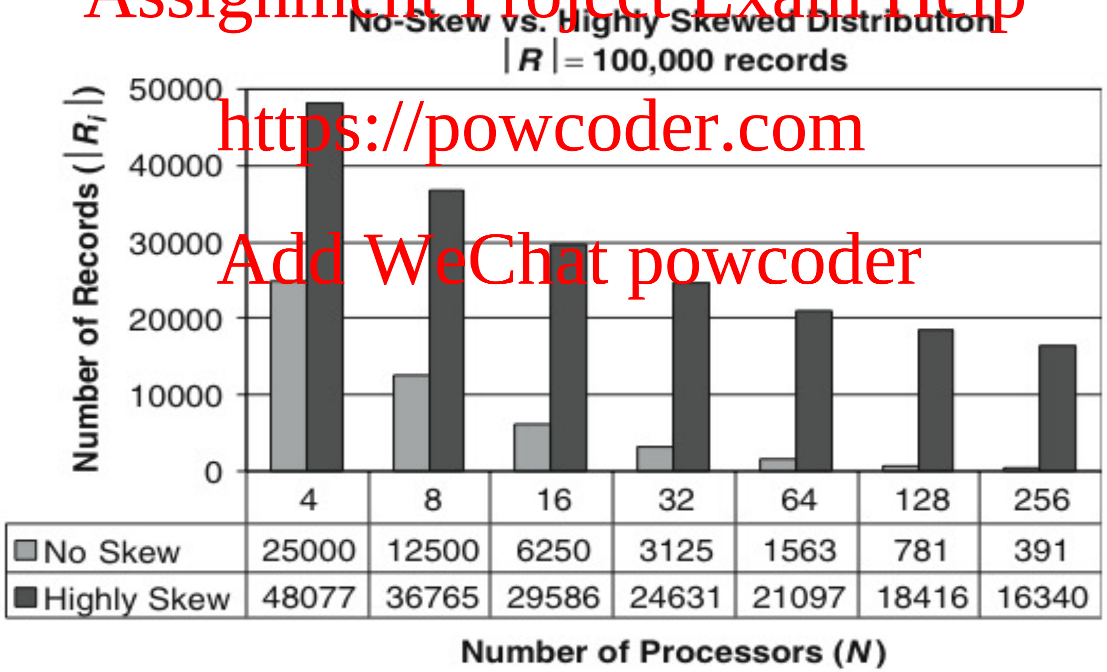


Figure 2.3 Comparison between highly skewed, less skewed, and no-skew distributions

1.3. Objectives (cont'd)

- **No skew vs. highly skewed**



	4	8	16	32	64	128	256
No Skew	25000	12500	6250	3125	1563	781	391
Highly Skew	48077	36765	29586	24631	21097	18416	16340

Figure 2.4 Comparison between the heaviest loaded processors using no-skew and highly skewed distributions

1.3. Objectives (cont'd)

- **No skew vs. highly skewed**

Table 2.2 Divisors (with vs. without skew)

N	4	8	16	32	64	128	256
Divisor without skew	4	8	16	32	64	128	256
Divisor with skew	2.08	2.72	3.38	4.06	4.74	5.43	6.12

- **Exercise 7 (FLUX Quiz)**
  - There 100,000 records in the table to be distributed to 32 processors. Assuming that the skewness degree is high (θ = 1), what is the estimated number of records in the heaviest processor?
  - A. 48,000 records
  - B. 29,000 records
  - C. 24,000 records
  - D. It is not possible to predict

1.4. Forms of Parallelism

- Forms of parallelism for database processing:
  - Interquery parallelism
  - Intraquery parallelism
  - Interoperation parallelism
  - Intraoperation parallelism
  - Mixed parallelism

1.4. Forms of Parallelism (cont'd)

- **Interquery Parallelism**
  - "Parallelism among queries"
  - Different queries or transactions are executed in parallel with one another
  - Main aim: scaling up transaction processing systems

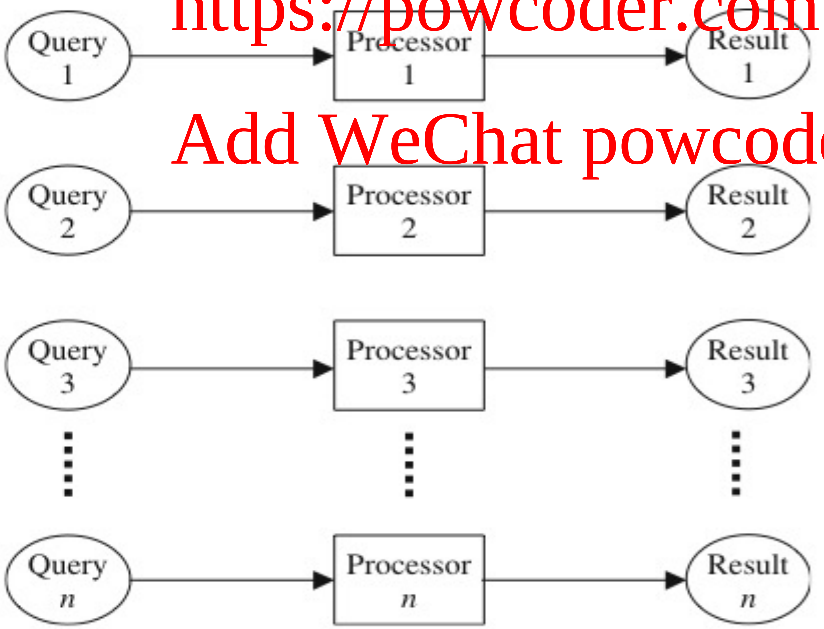


Figure 1.6 Interquery parallelism

1.4. Forms of Parallelism (cont'd)

- **Intraquery Parallelism**
  - "Parallelism within a query"
  - Execution of a single query in parallel on multiple processors and disks
  - Main aim: speeding up long-running queries

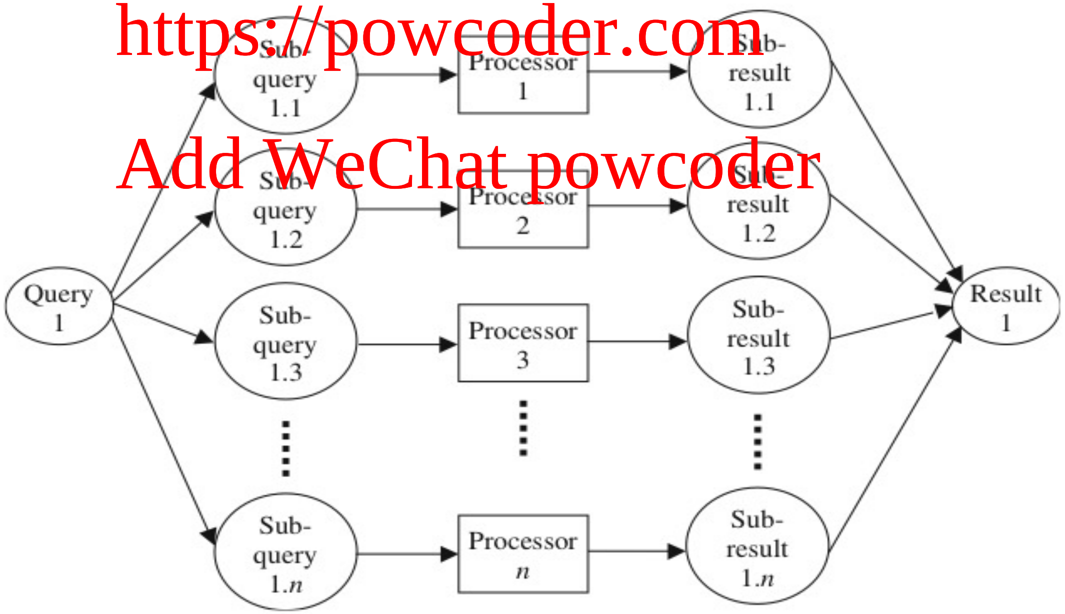


Figure 1.7 Intraquery parallelism

1.4. Forms of Parallelism (cont'd)

- Execution of a single query can be parallelized in two ways:
  - **Intraoperation parallelism**: Speeding up the processing of a query by parallelizing the execution of each individual operation (e.g. parallel sort, parallel search, etc)
  - **Interoperation parallelism**: Speeding up the processing of a query by executing in parallel different operations in a query expression (e.g. simultaneous sorting or searching)

1.4. Forms of Parallelism (cont'd)

- **Intraoperation Parallelism**
  - "Partitioned parallelism"
  - Parallelism due to the data being partitioned
  - Since the number of records in a table can be large, the degree of parallelism is potentially enourmous

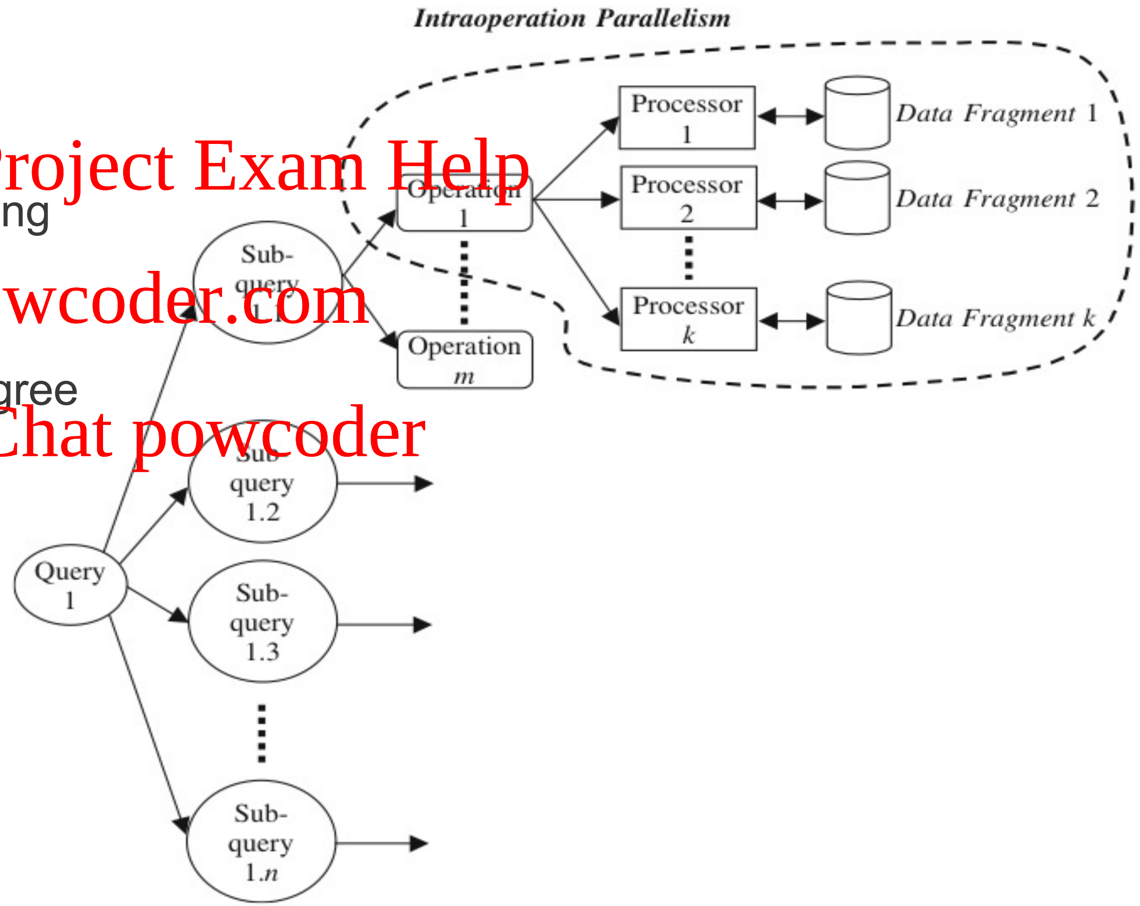


Figure 1.8 Intraoperation parallelism

1.4. Forms of Parallelism (cont'd)

- **Interoperation parallelism**: Parallelism created by concurrently executing different operations within the same query or transaction
  - Pipeline parallelism
  - Independent parallelism

1.4. Forms of Parallelism (cont'd)

- **Pipeline Parallelism**
  - Output record of one operation *A* are consumed by a second operation *B*, even before the first operation has produced the entire set of records in its output
  - Multiple operations form some sort of assembly line to manufacture the query results
  - Useful with a small number of processors, but does not scale up well

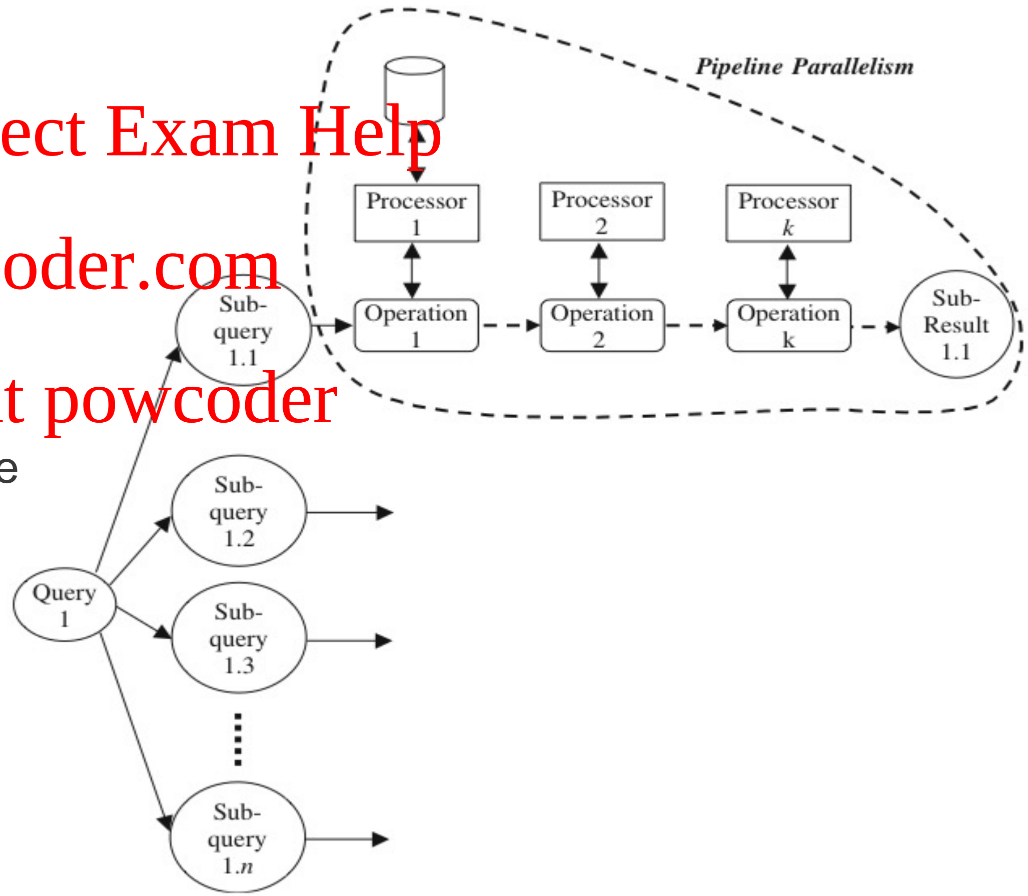


Figure 1.9 Pipeline parallelism

1.4. Forms of Parallelism (cont'd)

- **Independent Parallelism**
  - Operations in a query that do not depend on one another are executed in parallel
  - Does not provide a high degree of parallelism

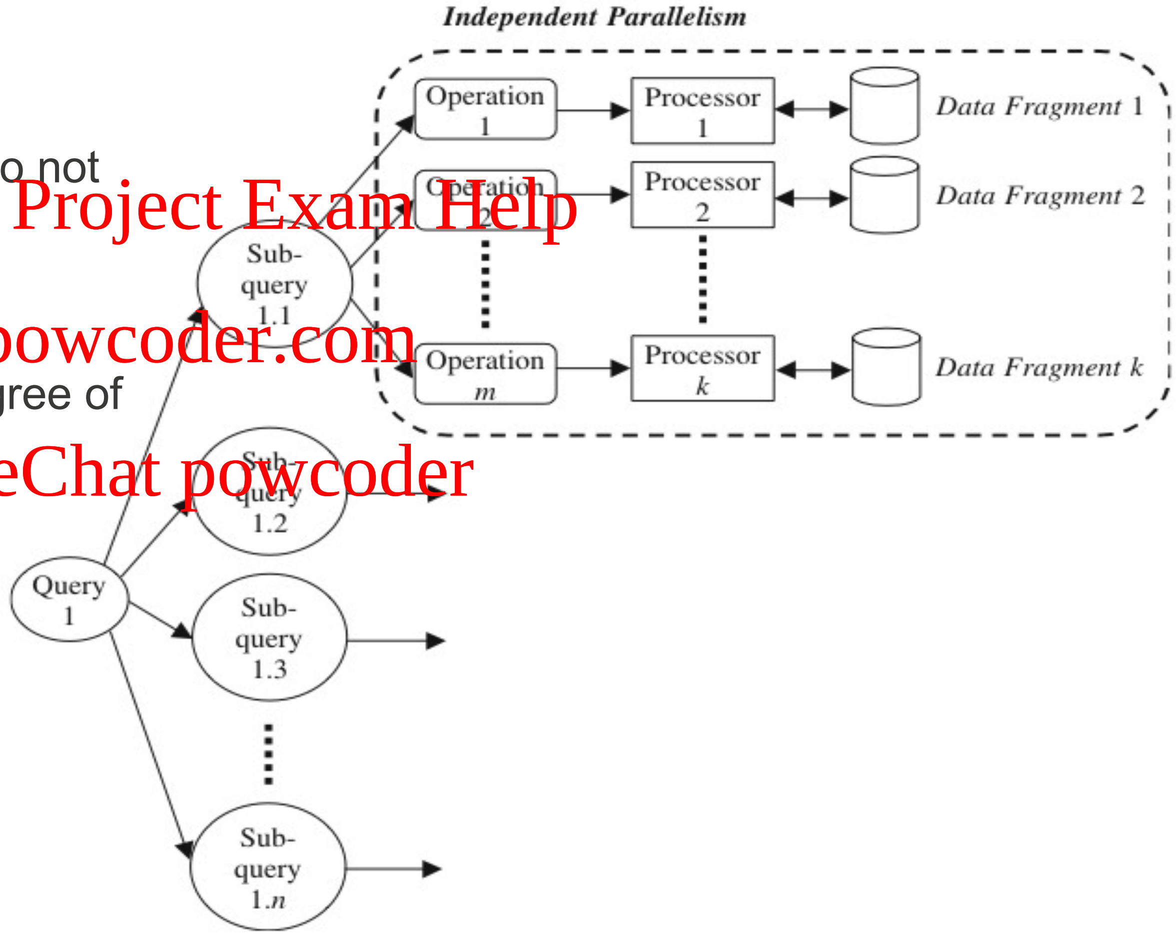


Figure 1.10 Independent parallelism

1.4. Forms of Parallelism (cont’d)

- **Mixed Parallelism**
  - In practice, a mixture of all available parallelism forms is used.

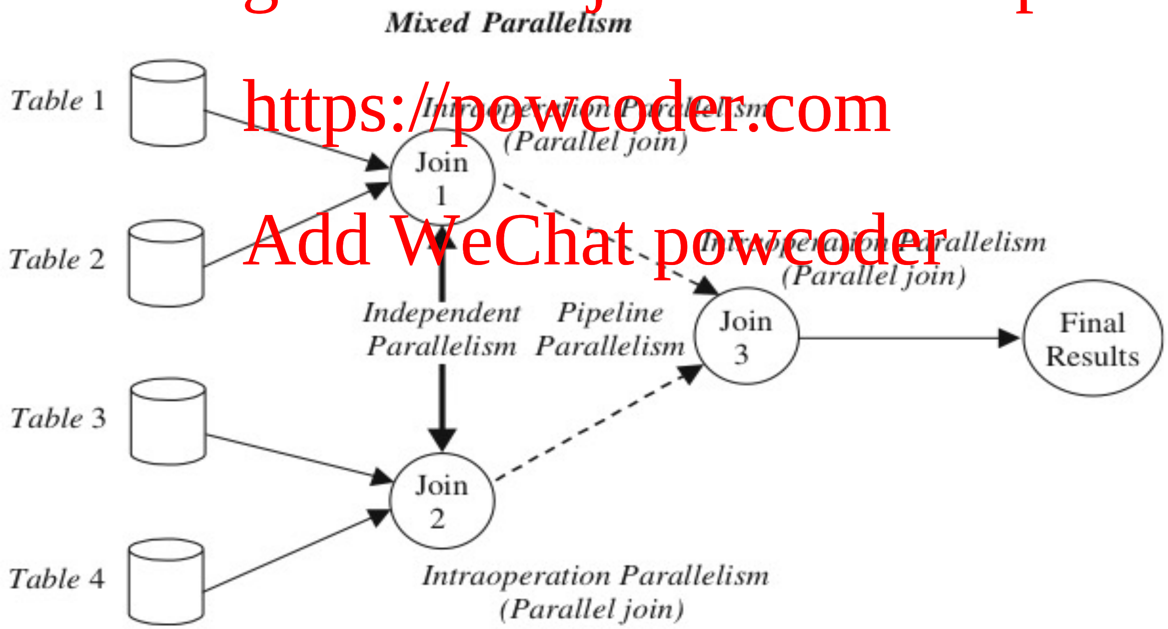


Figure 1.11 Mixed parallelism

1.5. Parallel Database Architectures

- Parallel computers are no longer a monopoly of supercomputers
- Parallel computers are available in many forms:
  - Shared-memory architecture
  - Shared-disk architecture
  - Shared-nothing architecture
  - Shared-something architecture

1.5. Parallel Database Architectures (cont'd)

- **Shared-Memory** and **Shared-Disk Architectures**
  - Shared-Memory: all processors share a common main memory and secondary memory
  - Load balancing is relatively easy to achieve, but suffer from memory and bus contention
  - Shared-Disk: all processors, each of which has its own local main memory, share the disks

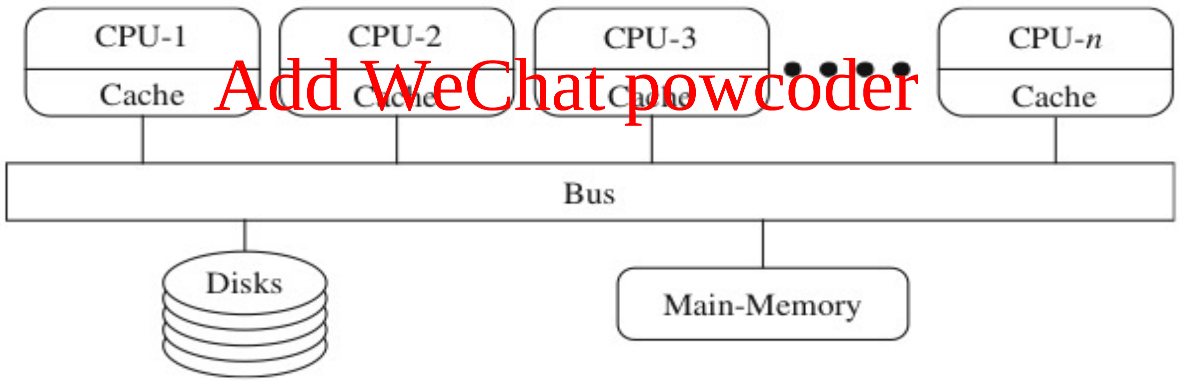


Figure 1.12 An SMP architecture

1.5. Parallel Database Architectures (cont'd)

- **Shared-Nothing Architecture**
  - Each processor has its own local main memory and disks
  - Load balancing becomes difficult

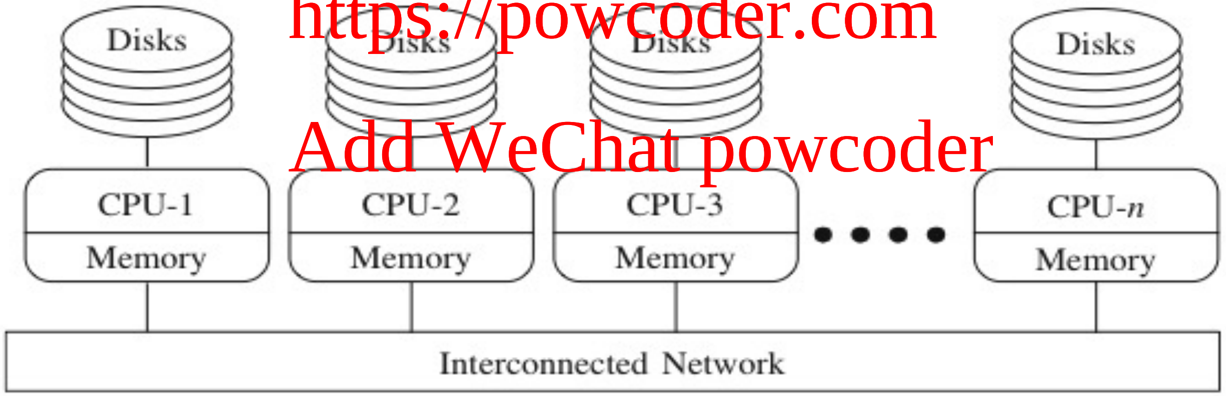


Figure 1.13 A shared-nothing architecture

1.5. Parallel Database Architectures (cont'd)

- **Shared-Something Architecture**
  - A mixture of shared-memory and shared-nothing architectures
  - Each node is a shared-memory architecture connected to an interconnection network aka shared-nothing architecture

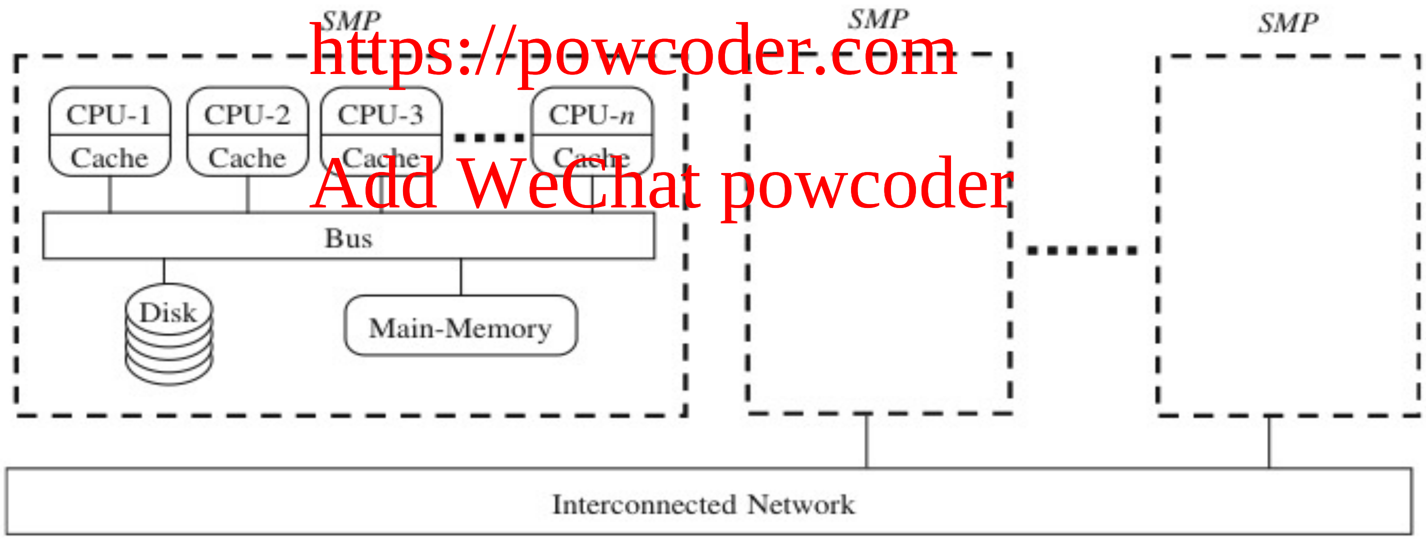


Figure 1.14 Cluster of SMP architectures

1.5. Parallel Database Architectures (cont'd)

- **Interconnection Networks**
  - Bus, Mesh, Hypercube

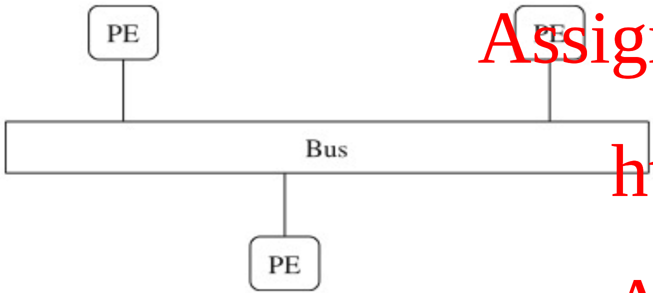


Figure 1.15 Bus interconnection network

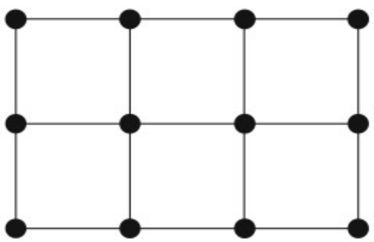


Figure 1.16 Mesh interconnection network

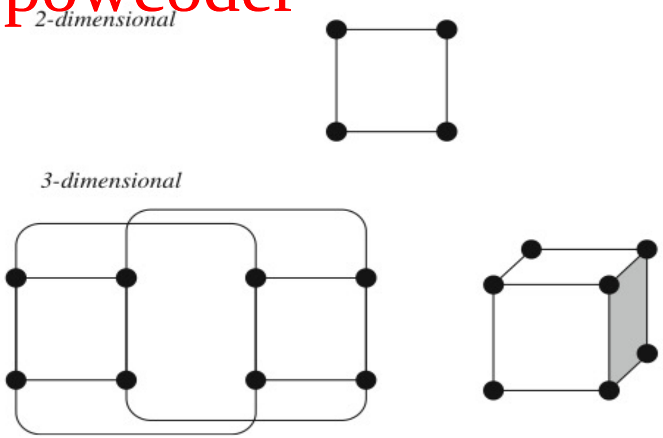


Figure 1.17 Hypercube interconnection network

1.7. Exercises (from the textbook)

- **Q1.3:** Highlight the differences between **speed up** and **scale up**.

- **Q1.7:** Skewed workload distribution is generally undesirable. Under what conditions that parallelism (i.e. the workload is divided among all processors) is not desirable?

1.8. Summary

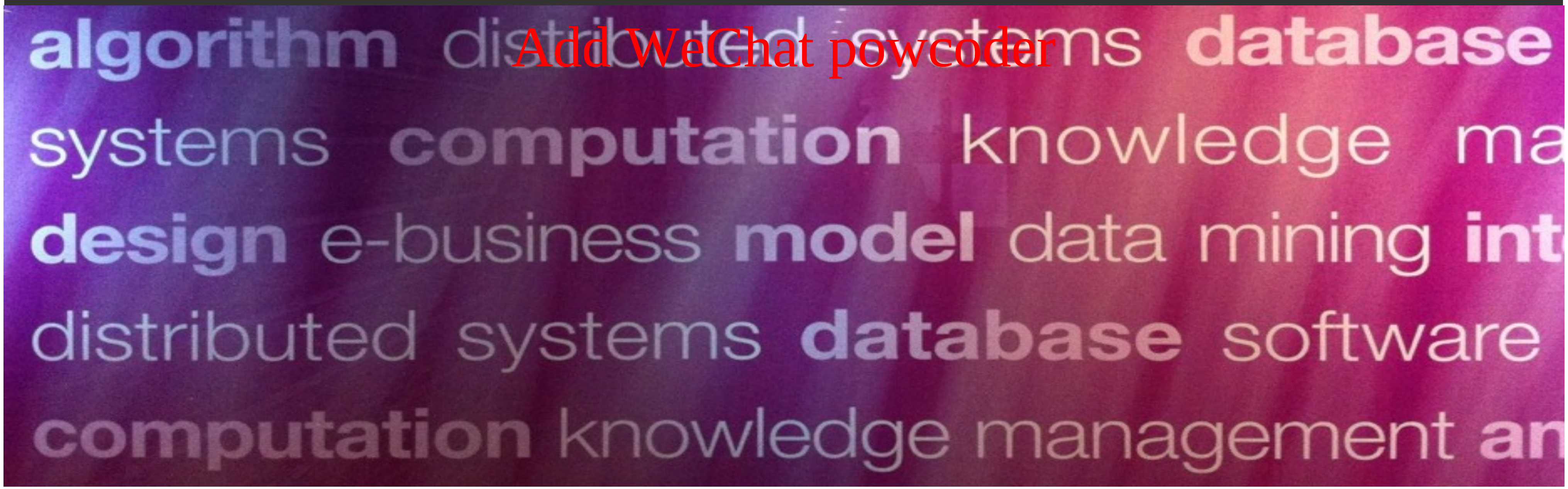
- **Why**, **What**, and **How** of parallel query processing:
  - Why is parallelism necessary in database processing?
  - What can be achieved by parallelism in database processing?
  - How parallelism performed in database processing?
  - What facilities of parallel computing can be used?

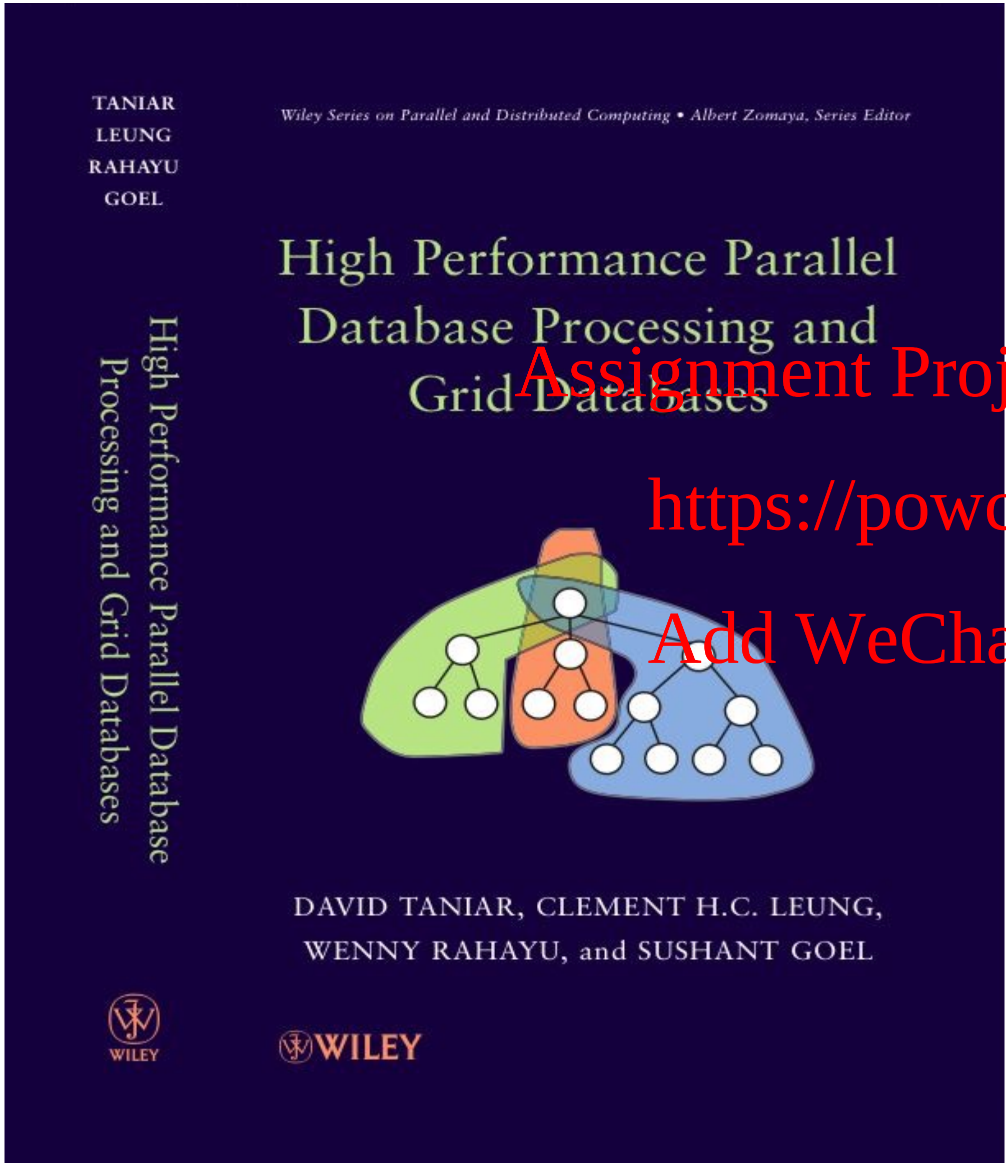
MONASH University

Information Technology

FIT5148 (Volume II - Search)

Week 2b – Parallel Search





Chapter 3
Parallel Search

3.1. Search Queries

- Search is ***selection*** operation in database queries
- Selects specified records based on a given criteria
- The result is a horizontal subset (records) of the operand

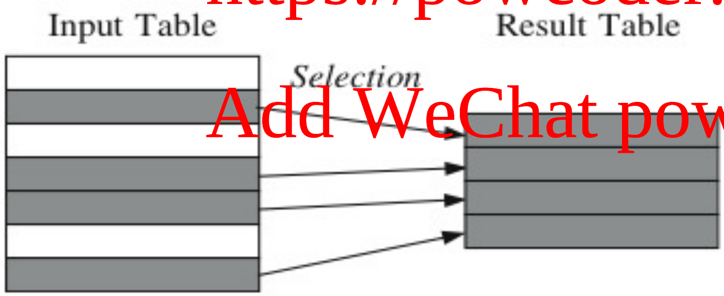


Figure 3.1 Selection operation

- Three kinds of search queries:
  - Exact-match search
  - Range search
  - Multi attribute search

3.1. Search Queries (cont'd)

- **Exact-Match Search**
  - Selection predicate on an attribute to check for an exact match between a search attribute and a given value
  - Expressed by the WHERE clause in SQL
  - Query 3.1 will produce a unique record (if the record is found), whereas Query 3.2 will likely produce multiple records


```
Query 3.1:
    Select *
    From STUDENT
    Where Sid = 23;
```

```
Query 3.2:
    Select *
    From STUDENT
    Where Slname = 'Robinson';
```

3.1. Search Queries (cont'd)

- **Range Search Query**
  - The search covers a certain range
  - **Continuous range search query**

```
Query 3.3:
    Select *
    From STUDENT
    Where Sgpa > 3.50;
```

  - **Discrete range search query**

```
Query 3.4:
    Select *
    From STUDENT
    Where Sdegree IN ('BCS', 'BInfSys');
```

3.1. Search Queries (cont'd)

- **Multiattribute Search Query**
  - More than attribute is involved in the search
  - Conjunctive (AND) or Disjunctive (OR)
  - If both are used, it must be in a form of *conjunctive prenex normal form* (CPNF)

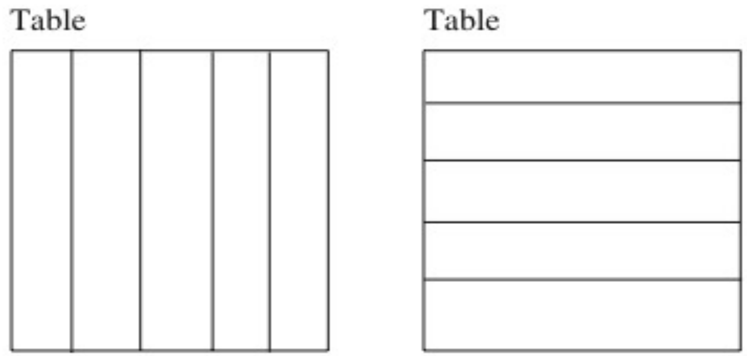

```
Query 3.6:
    Select *
    From STUDENT
    Where Slname = 'Robinson'
    And Sdegree IN ('BCS', 'BInfSys');
```

3.2. Data Partitioning

- Distributes data over a number of processing elements
- Each processing element is then executed simultaneously with other processing elements, thereby creating parallelism
- Can be physical or logical data partitioning
- In a shared-nothing architecture, data is placed permanently over several disks
- In a shared-everything (shared-memory and shared-disk) architecture, data is assigned logically to each processor
- Two kinds of data partitioning:
  - Basic data partitioning
  - Complex data partitioning

3.2. Data Partitioning (cont'd)

- **Basic Data Partitioning**
  - Vertical vs. Horizontal data partitioning
  - Vertical partitioning partitions the data vertically across all processors. Each processor has a full number of records of a particular table. This model is more common in distributed database systems
  - Horizontal partitioning is a model in which each processor holds a partial number of complete records of a particular table. It is more common in parallel relational database systems



(a) Vertical Partitioning *(b) Horizontal Partitioning*

Figure 3.2 Vertical and horizontal data partitioning

3.2. Data Partitioning (cont'd)

- **Basic Data Partitioning**
  - Round-robin data partitioning
  - Hash data partitioning
  - Range data partitioning
  - Random-unequal data partitioning

3.2. Data Partitioning (cont'd)

- **Round-robin data partitioning**
  - Each record in turn is allocated to a processing element in a clockwise manner
  - "Equal partitioning" or "Random-equal partitioning"
  - Data evenly distributed, hence supports load balance
  - But data is not grouped semantically

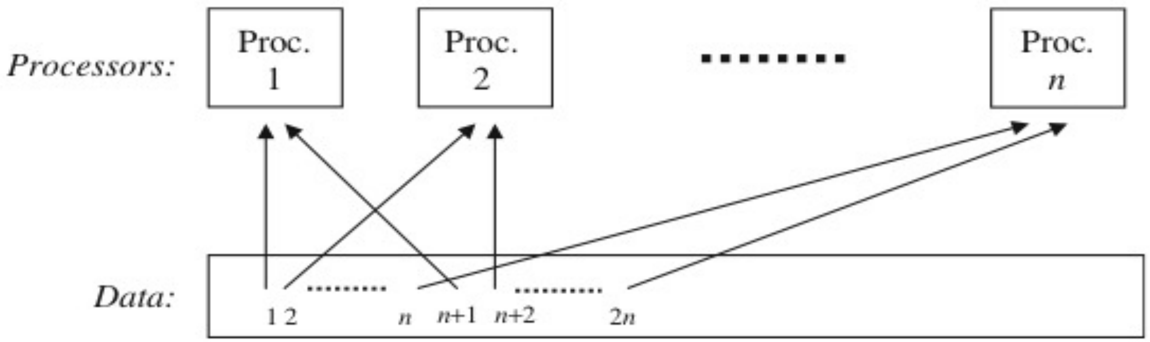
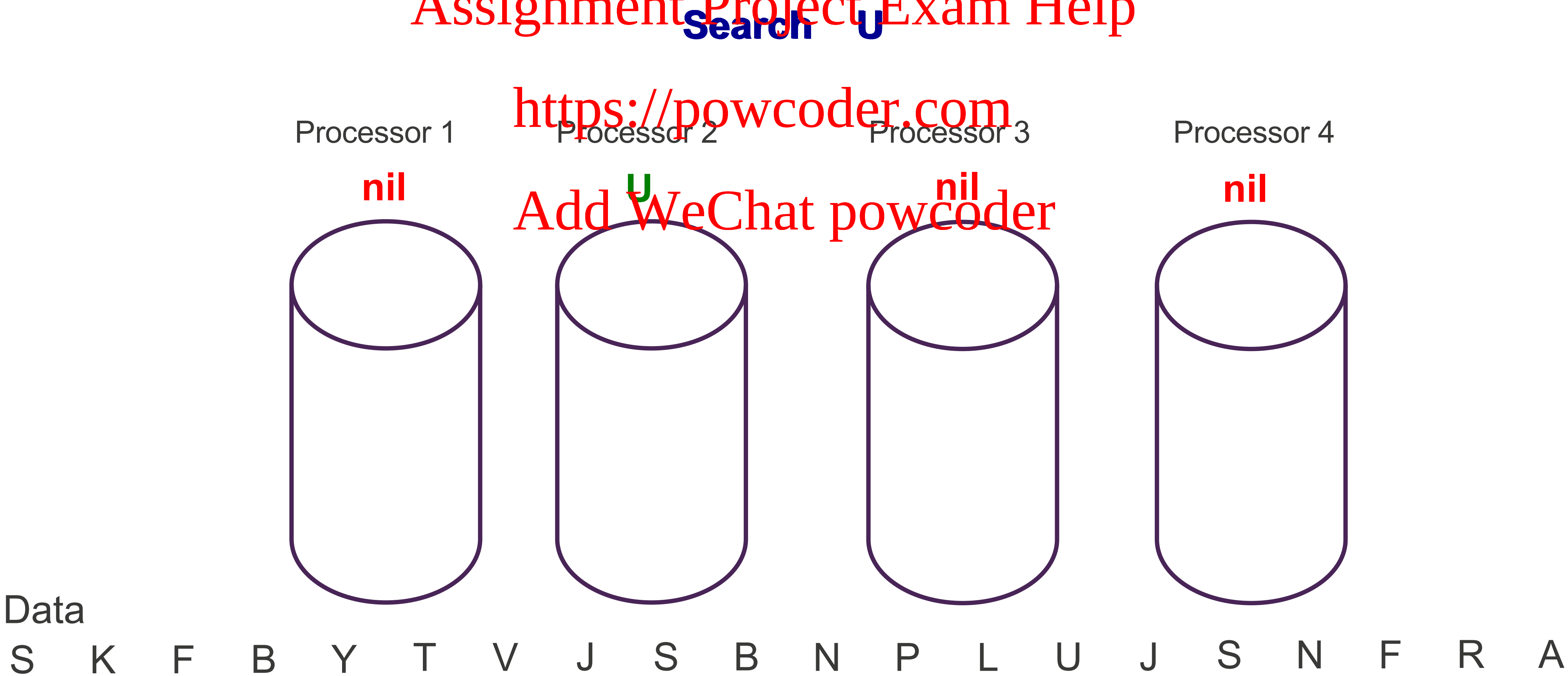


Figure 3.3 *Round-robin* data partitioning

3.2. Data Partitioning (cont'd)

- **Round-robin data partitioning**



3.2. Data Partitioning (cont'd)

- **Hash data partitioning**
  - A hash function is used to partition the data
  - Hence, data is grouped semantically, that is data on the same group shared the same hash value
  - Selected processors may be identified when processing a search operation (exact-match search), but for range search (especially continuous range), all processors must be used
  - Initial data allocation is not balanced either

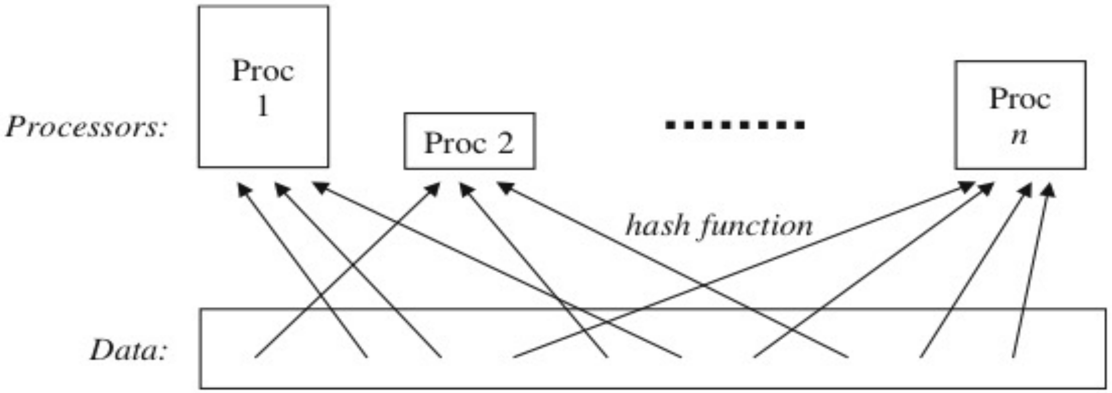


Figure 3.4 *Hash* data partitioning

3.2. Data Partitioning (cont'd)

- **Range data partitioning**
  - Spreads the records based on a given range of the partitioning attribute
  - Processing records on a specific range can be directed to certain processors only
  - Initial data allocation is skewed too

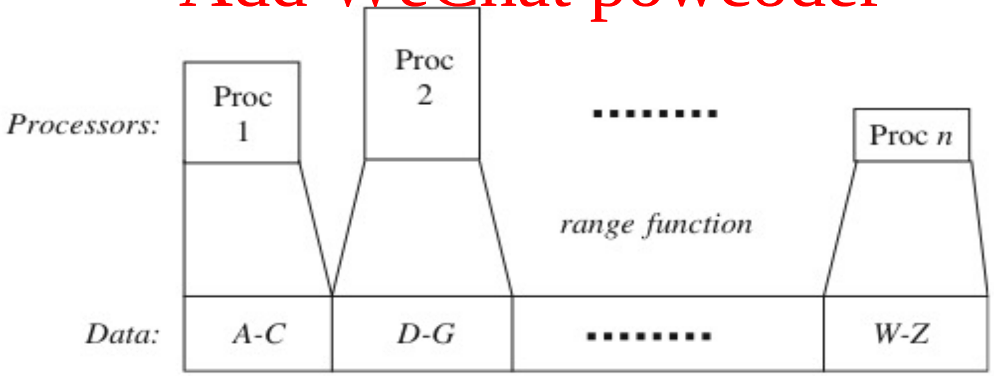
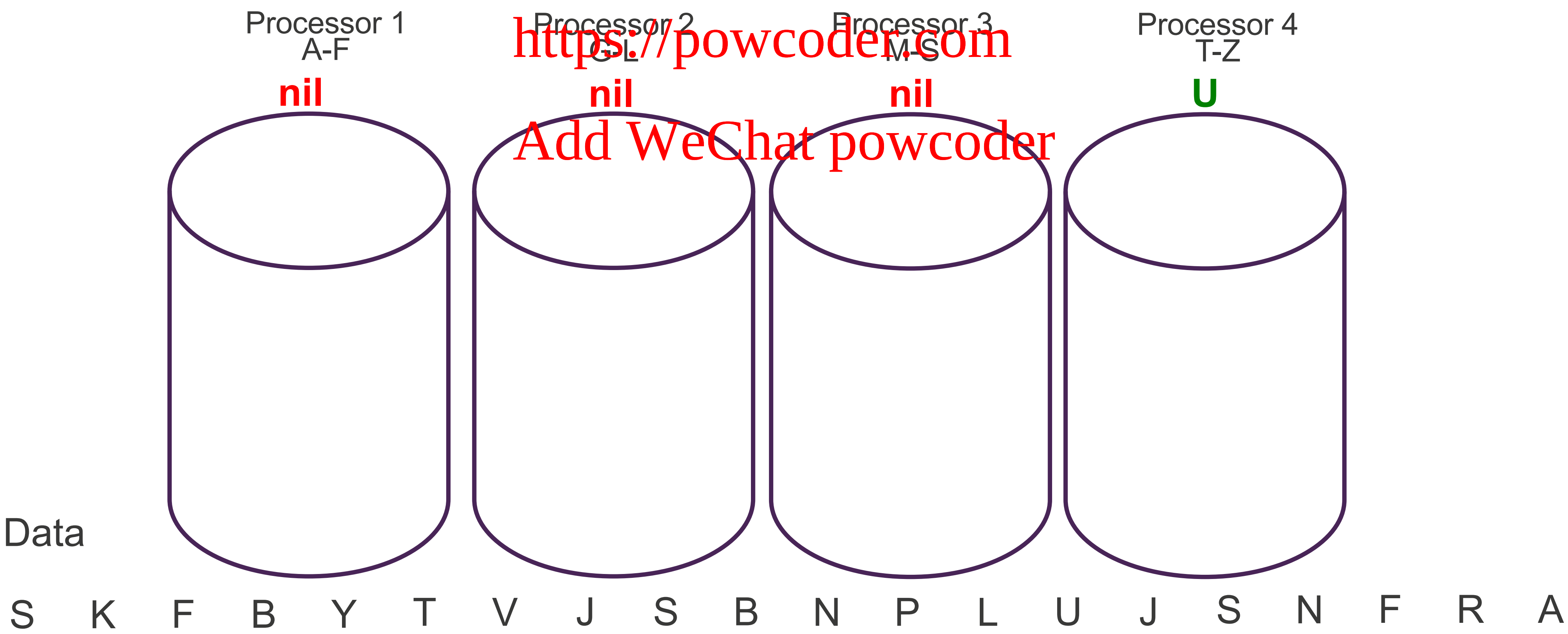


Figure 3.5 *Range* data partitioning

3.2. Data Partitioning (cont'd)

- **Range data partitioning**

Search: U



3.2. Data Partitioning (cont'd)

- **Random-unequal data partitioning**
  - Partitioning is not based on the same attribute as the retrieval processing is based on a non-retrieval processing attribute, or the partitioning method is unknown
  - The size of each partitioning is likely to be unequal
  - Records within each partition are not grouped semantically
  - This is common especially when the operation is actually an operation based on temporary results obtained from the previous operations

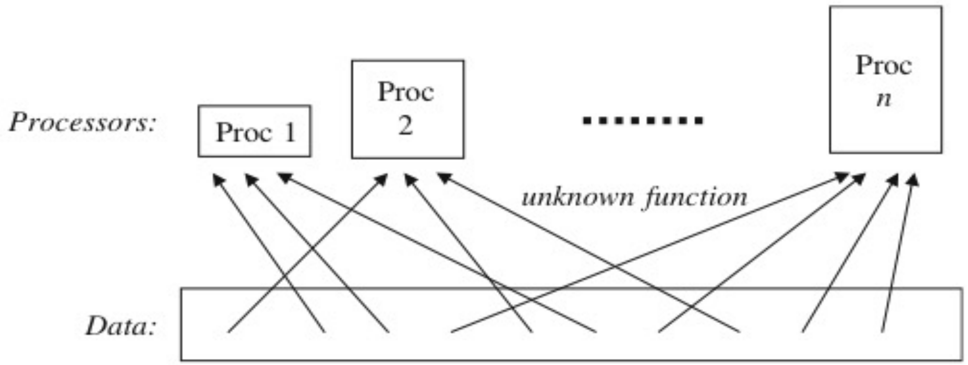
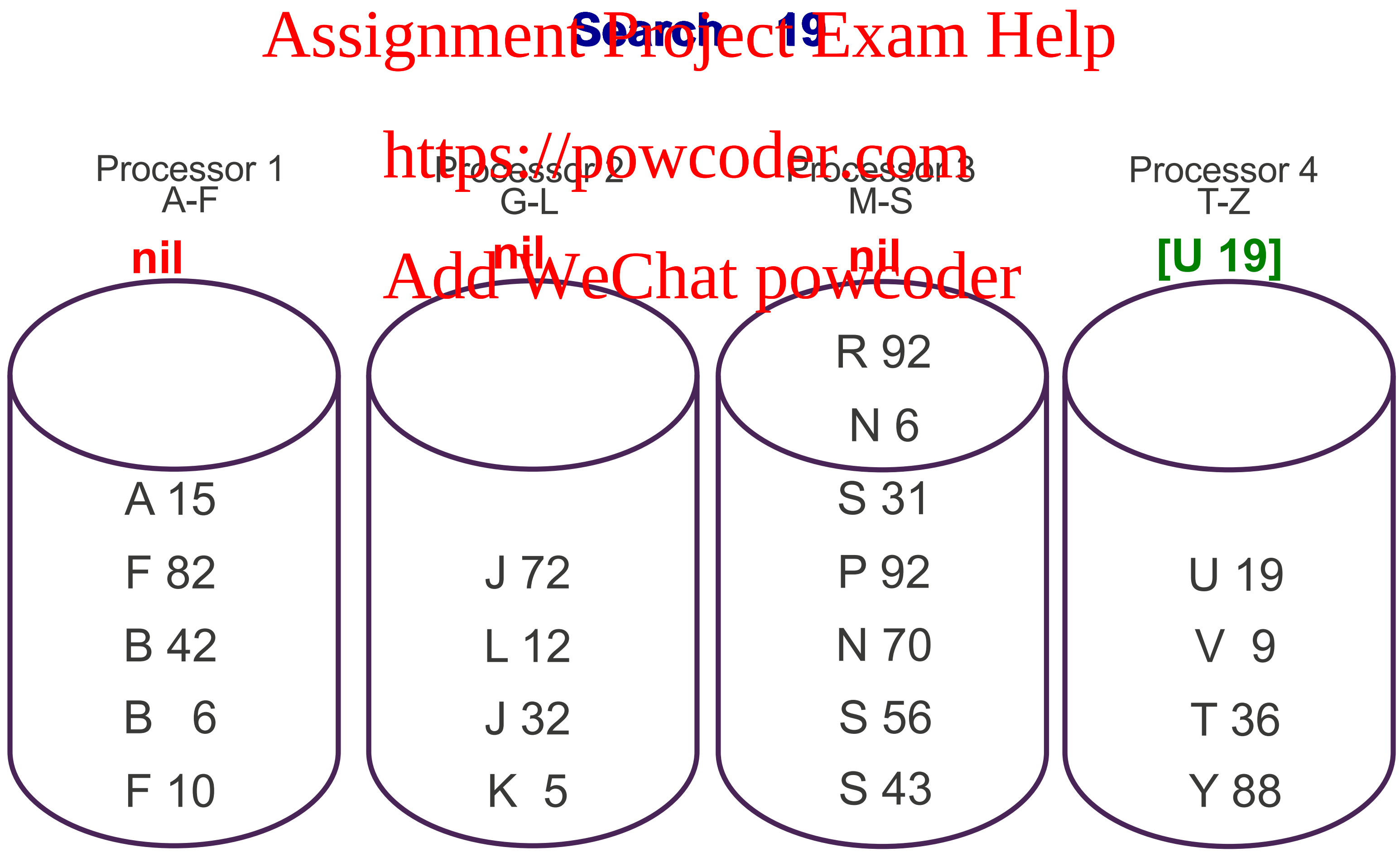


Figure 3.6 *Random-unequal* data partitioning

3.2. Data Partitioning (cont'd)

- **Random-unequal data partitioning**



3.2. Data Partitioning (cont'd)

- **Basic Data Partitioning**
  - Attribute-based data partitioning
  - Non-attribute-based data partitioning

Table 3.1 *Attribute-based* versus *non-attribute-based* data partitioning

Attribute-Based Partitioning	Non-Attribute-Based Partitioning
Based on a particular attribute	Not based on any attribute
Has grouping semantics	No grouping semantics
Skew	Balanced

3.2. Data Partitioning (cont'd)

- **Exercise 1 (Textbook Q3.6)**
  - Given a data set D = {55; 30; 68; 39; 1; 4; 49; 90; 34; 76; 82; 56; 31; 25; 78; 56; 38; 32; 88; 9; 44; 98; 11; 70; 66; 89; 99; 22; 23; 26}, three processors, and a **random-equal data partitioning**, illustrate how the parallel searching of **data item 78** is carried out.

3.2. Data Partitioning (cont'd)

- **Exercise 2 (Textbook Q3.7)**
  - Given a data set *D* = {55; 30; 68; 39; 1; 4; 49; 90; 34; 76; 82; 56; 31; 25; 78; 56; 38; 32; 88; 9; 44; 98; 11; 70; 66; 89; 99; 22; 23; 26}, three processors, and a **range data partitioning**, illustrate how the parallel searching of data items **between 70 and 79** can be carried out.

3.2. Data Partitioning (cont'd)

- **Exercise 3 (Textbook Q3.8)**
  - Given a data set *D* = {55; 30; 68; 39; 1; 4; 49; 90; 34; 76; 82; 56; 31; 25; 78; 56; 38; 32; 88; 9; 44; 98; 11; 70; 66; 89; 99; 22; 23; 26}, three processors, and a **hash data partitioning**, illustrate how the parallel searching of data items **10, 20, 30, …, 90** can be carried out.

3.2. Data Partitioning (cont'd)

- **Complex Data Partitioning**
  - Basic data partitioning is based on a single attribute (or no attribute)
  - Complex data partitioning is based on multiple attributes or is based on a single attribute but with multiple partitioning methods

  - Hybrid-Range Partitioning Strategy (HRPS)
  - Multiattribute Grid Declustering (MAGIC)
  - Bubba's Extended Range Declustering (BERB)

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - Partitions the table into many fragments using range, and the fragments are distributed to all processors using round-robin
  - Each fragment contains approx *FC* records

$$FC = \frac{RecordsPerQ_{Ave}}{M} \qquad (3.1)$$

Where $RecordsPerQ_{Ave}$ is the average number of records retrieved and processed by each query, and *M* is the number of processors that should participate in the execution of an average query

  - Each fragment contains a unique range of values of the partitioning attribute
  - The table must be sorted on the partitioning attribute, then it is partitioned that each fragment contains *FC* records, and the fragments are distributed in round-robin ensuring that *M* adjacent fragements assigned to different processors

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - Example: 10000 student records, and the partitioning attribute is StudentID (PK) that ranges from 1 to 10000. Assume the average query retrieves a range of 500 records (*RecordsPerQ*=500). Queries access students per year enrolment wth average results of 500 records. Assume the optimal performance is achieved when 5 processors are used (*M*=5)

$$FC = \frac{RecordsPerQ_{Ave}}{M} = 100$$

  - The table will be partitioned into 100 fragments
  - Three cases: *M* = *N*, *M* > *N*, or *M* < *N* (where *N* is the number of processors in the configuration, and *M* is the number of processors participating in the query execution

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - **Case 1: *M* = *N***
  - Because the query will overlap with 5-6 fragments, all processors will be used (high degree of parallelism)
  - Compared with hash partitioning: Hash will also use *N* processors, since it cannot localize the execution of a range query
  - Compared with range partitioning: Range will only use 1-2 processors, and hence the degree of parallelism is small

HRPS	1-100 9501-9600	101-200 9601-9700	201-300 9701-9800	301-400 9801-9900	401-500 9901-10000
Range	1-2000	2001-4000	4001-6000	6001-8000	8001-10000

Figure 3.7 Case 1 ($M = N$) and a comparison with the range partitioning method

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - **Case 2: *M* > *N*** (e.g. *M*=5, and *N*=2)
  - HRPS will still use all *N* processors, because it enforces the constraint that the *M* adjacent fragments be assigned to different processors whenever possible
  - Compared with range partitioning: an increased probability that a query will use only one processor (in this example)

HRPS	1-100 201-300 9801-9900	101-200 301-400 9901-10000
Range	1-5000	5001-10000

Figure 3.8 Case 2 ($M > N$) and a comparison with the range partitioning method

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - **Case 3: *M* < *N*** (e.g. *M*=5, and *N*=10)
  - HRPS distributes 100 fragments to all *N* processors. Since the query will overlap with only 5-6 fragments, each individual query is localized to almost the optimal number of processors
  - Compared with hash partitioning: Hash will use all *N* processors, and hence less efficient due to start up, communication, and termination overheads
  - Compared with range partitioning: The query will use 1-2 processors only, and hence less optimal

HRPS	1-100 9001-9100	101-200 9101-9200	201-300 9201-9300	301-400 9301-9400	401-500 9401-9500	501-600 9501-9600	601-700 9601-9700	701-800 9701-9800	801-900 9801-9900	901-1000 9901-10000
Range	1-1000	1001-2000	2001-3000	3001-4000	4001-5000	5001-6000	6001-7000	7001-8000	8001-9000	9001-10000

Figure 3.9 Case 3 ($M < N$) and a comparison with the range partitioning method

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - **Support for Small Tables**

    If the number of fragments of a table is less than the number of processors, then the table will automatically be partitioned across a subset of the processors

  - **Support for Tables with Nonuniform Distributions of the Partitioning Attribute Values**

    Because the cardinality of each fragment is not based on the value of the partitioning attribute value, once the HRPS determines the cardinality of each fragment, it will partition a table based on that value

3.2. Data Partitioning (cont'd)

- **Multiattribute Grid Declustering (MAGIC)**
  - Based on multiple attributes – to support search queries based on either of data partitioning attributes
  - Support range and exact match search on each of the partitioning attributes
  - Example: Query 1 (one-half of the accesses) Slname='Roberts', and Query 2 (the other half) SID between 98555 and 98600. Assume both queries produce only a few records
  - Create a two-dim grid with the two partitioning attributes (Slname and SID). The number of cells in the grid equal the number of processing elements
  - Determine the range value for each column and row, and allocate a processor in each cell in the grid

3.2. Data Partitioning (cont'd)

- **Multiattribute Grid Declustering (MAGIC)**
  - Query 1 (exact match on Slname): Hash partitioning can localize the query processing on one processor. MAGIC will use 6 processors
  - Query 2 (range on SID): if the hash partitioning uses Slname, whereas the query is on SID, the query must use all 36 processors. MAGIC on the other hand, will only use 6 processors.
  - Compared with range partitioning, suppose the partitioning is based on SID, then Q1 will use 36 processors whilst Q2 will use 1 processor

Table 3.2 MAGIC data partitioning

		Slname					
		A–D	**E–H**	**I–L**	**M–P**	**Q–T**	**U–Z**
Sid	**98000–98100**	1	2	3	4	5	6
	98101–98200	7	8	9	10	11	12
	98201–98300	13	14	15	16	17	18
	98301–98400	19	20	21	22	23	24
	98401–98500	25	26	27	28	29	30
	98501–98600	31	32	33	34	35	36

3.2. Data Partitioning (cont’d)

- **Bubba’s Extended Range Declustering (BERB)**
  - Another multiattribute partitioning method - used in the Bubba Database Machine
  - Two levels of data partitioning: ***primary*** and ***secondary*** data partitioning
  - Step 1: Partition the table based on the primary partitioning attribute and uses a range partitioning method

Table 3.3 Primary partitioning in BERD

Sid	*Slname*	*Sid*	*Slname*	*Sid*	*Slname*
98001	Robertson	98105	Black	98250	Chan
98050	Williamson	98113	White	98270	Tan
98001–98100		98101–98200		98201–98300	

3.2. Data Partitioning (cont'd)

- **Bubba's Extended Range Declustering (BERB)**
  - Step 2: Each fragment is scanned and an 'aux' table is created from the attribute value of the secondary partitioning attribute and a list of processors containing the original records
  - Table 3.4 shows the 'aux' table (called Table *IndexB*)

Table 3.4 Auxiliary table in the secondary partitioning

Slname	***Processor***
Robertson	1
Black	2
Chan	3
Williamson	1
White	2
Tan	3

3.2. Data Partitioning (cont’d)

- **Bubba’s Extended Range Declustering (BERB)**
  - Step 3: The ‘aux’ table is range partitioned on the secondary partitioning attribute (e.g. Surname)
  - Step 4: Place the fragments from steps 1 and 3 into multiple processors

Table 3.5 BERD partitioning combining the primary partitions and the secondary partitions

IndexB

Black	2
Chan	3

Student

98001	Robertson
98050	Williamson

IndexB

Robertson	1
Tan	3

Student

98005	Black
98113	White

IndexB

Williamson	1
White	2

Student

98250	Chan
98270	Tan

3.3. Search Algorithms

- **Serial** search algorithms:
  - Linear search
  - Binary search

- **Parallel** search algorithms:
  - Processor activation or involvement
  - Local searching method
  - Key comparison

3.3. Search Algorithms (cont'd)

- **Linear Search**
  - Exhaustive search - search each record one by one until it is found or end of table is reached

- **Binary Search**
  - Must be pre-sorted
  - The complexity is $O(\log_2(n))$

3.3. Search Algorithms (cont'd)

- **Parallel** search algorithms:
  - Processor activation or involvement
  - Local searching method
  - Key comparison

3.3. Search Algorithms (cont'd)

- **Processor activation or involvement**
  - The number of processors to be used by the algorithm
  - If we know where the data to be sought are stored, then there is no point in activating all other processors in the searching process
  - Depends on the data partitioning method used
  - Also depends on what type of selection query is performed

Table 3.6 Processor activation or involvement of parallel search algorithms

		Data Partitioning Methods			
		Random-Equal	**Hash**	**Range**	**Random-Unequal**
Exact Match		All	1	1	All
Range Selection	**Continuous**	All	All	Selected	All
	Discrete	All	Selected	Selected	All

3.3. Search Algorithms (cont'd)

- **Local searching method**
  - The searching method applied to the processor(s) involved in the searching process
  - Depends on the data ordering, regarding the type of the search (exact match of range)

Table 3.7 Local searching method of parallel search algorithms

		Records Ordering	
		Ordered	**Unordered**
Exact Match		Binary Search	Linear Search
Range Selection	**Continuous**	Binary Search	Linear Search
	Discrete	Binary Search	Linear Search

3.3. Search Algorithms (cont'd)

- **Key comparison**
  - Compares the data from the table with the condition specified by the query
  - When a match is found: continue to find other matches, or terminate
  - Depends on whether the data in the table is unique or not

Table 3.8 Key comparison of parallel search algorithms

		Search Attribute Values	
		Unique	**Duplicate**
Exact Match		Stop	Continue
Range Selection	**Continuous**	Continue	Continue
	Discrete	Continue	Continue

3.4. Summary

- Search queries in SQL using the WHERE clause
- Search predicates indicates the type of search operation
  - Exact-match, range (continuous or discrete), or multiattribute search
- Data partitioning is a basic mechanism of parallel search
  - Single attribute-based, no attribute-based, or multiattribute-based partitioning
- Parallel search algorithms have three main components
  - Processor involvement, local searching method, and key comparison

Homework: Read Chapter 5 for next week