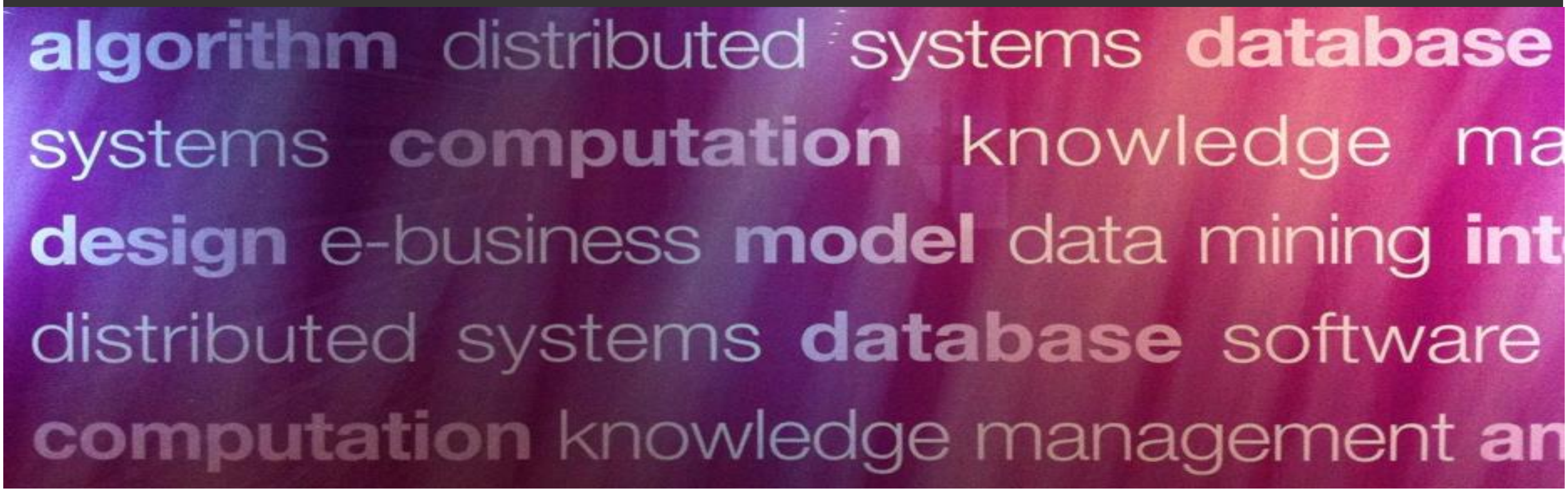
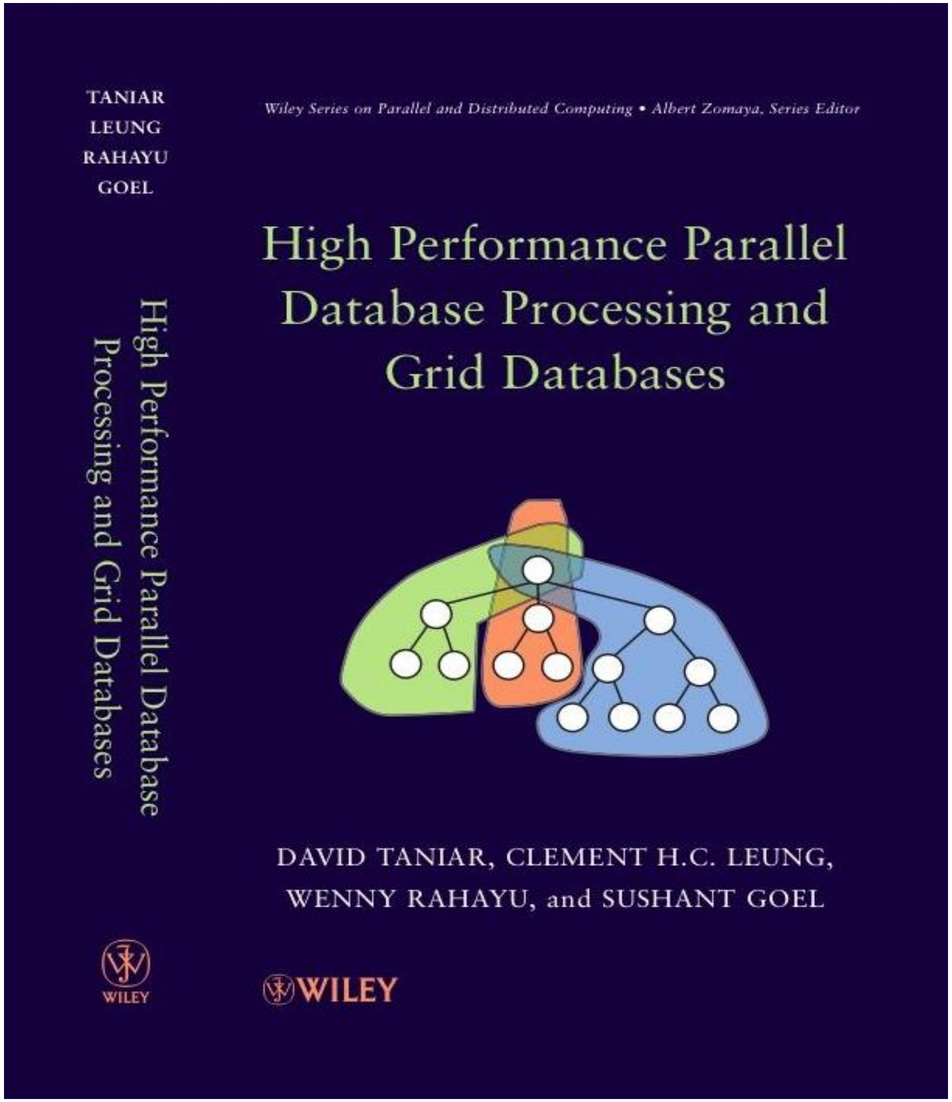
MONASH University

Information Technology

FIT5202 (Volume II - Search)

Week 2b – Parallel Search





Chapter 3
Parallel Search

Step 1: Data partitioning
Step 2: Perform parallel search in partitioned data

3.1. Search Queries

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

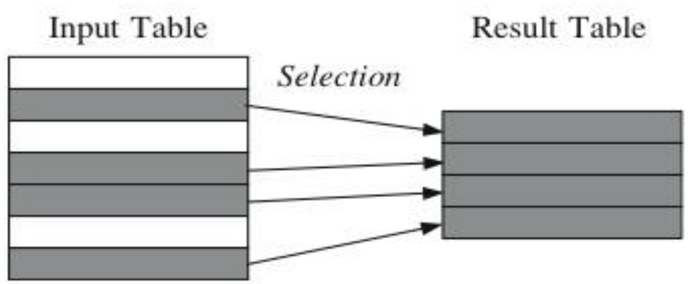


Figure 3.1 Selection operation

- 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 analytical 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 transactional relational database systems

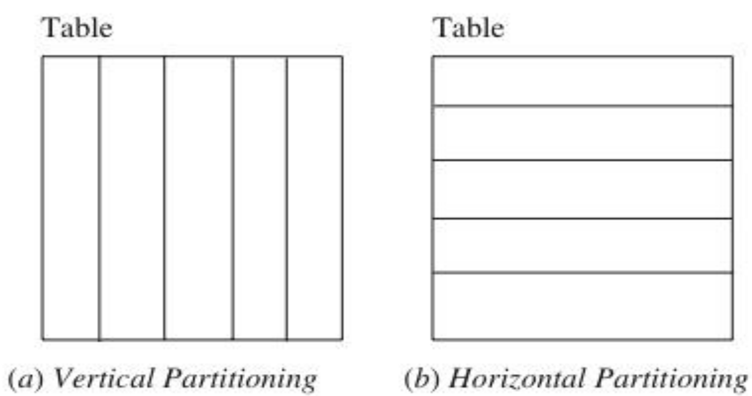


Figure 3.2 Vertical and horizontal data partitioning

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

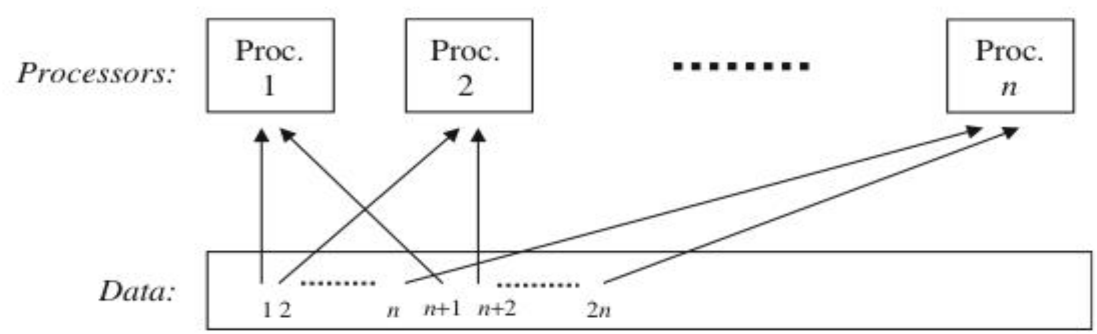


Figure 3.3 *Round-robin* data partitioning

3.2. Data Partitioning (cont'd)

- **Round-robin data partitioning**

Search U

Processor 1 | Processor 2 | Processor 3 | Processor 4

nil | U | nil | nil

Data

S K F B Y T V J S B N P L U J S N F R A

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

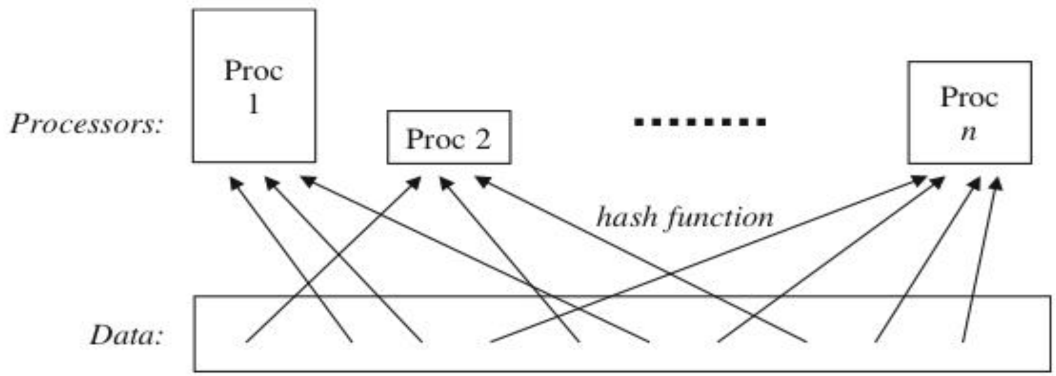


Figure 3.4 *Hash* data partitioning

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

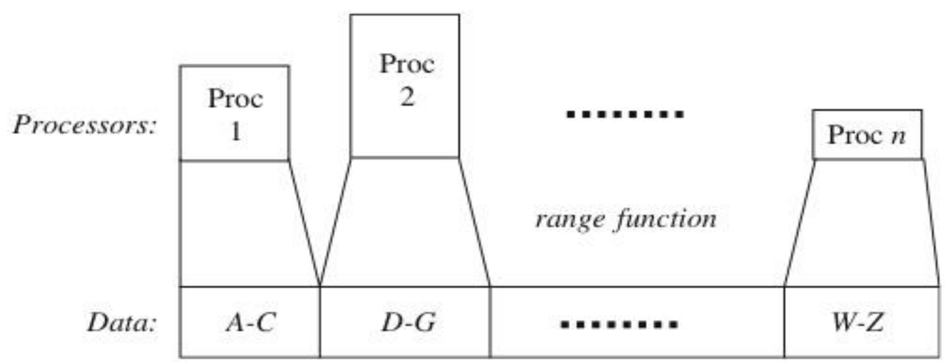
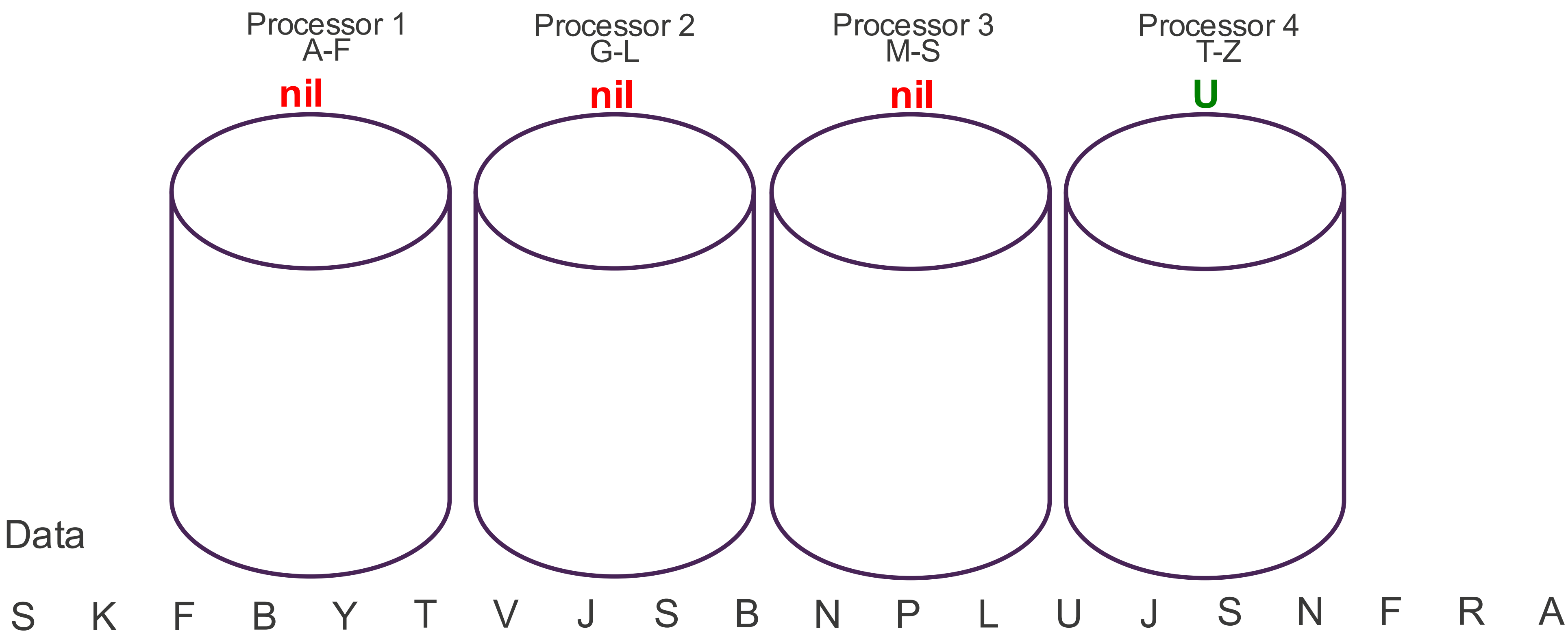


Figure 3.5 *Range* data partitioning

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 (e.g., partition attribute ≠ search 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

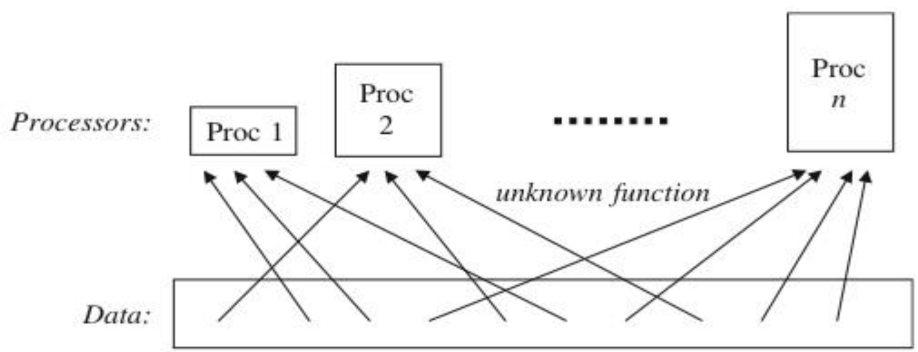
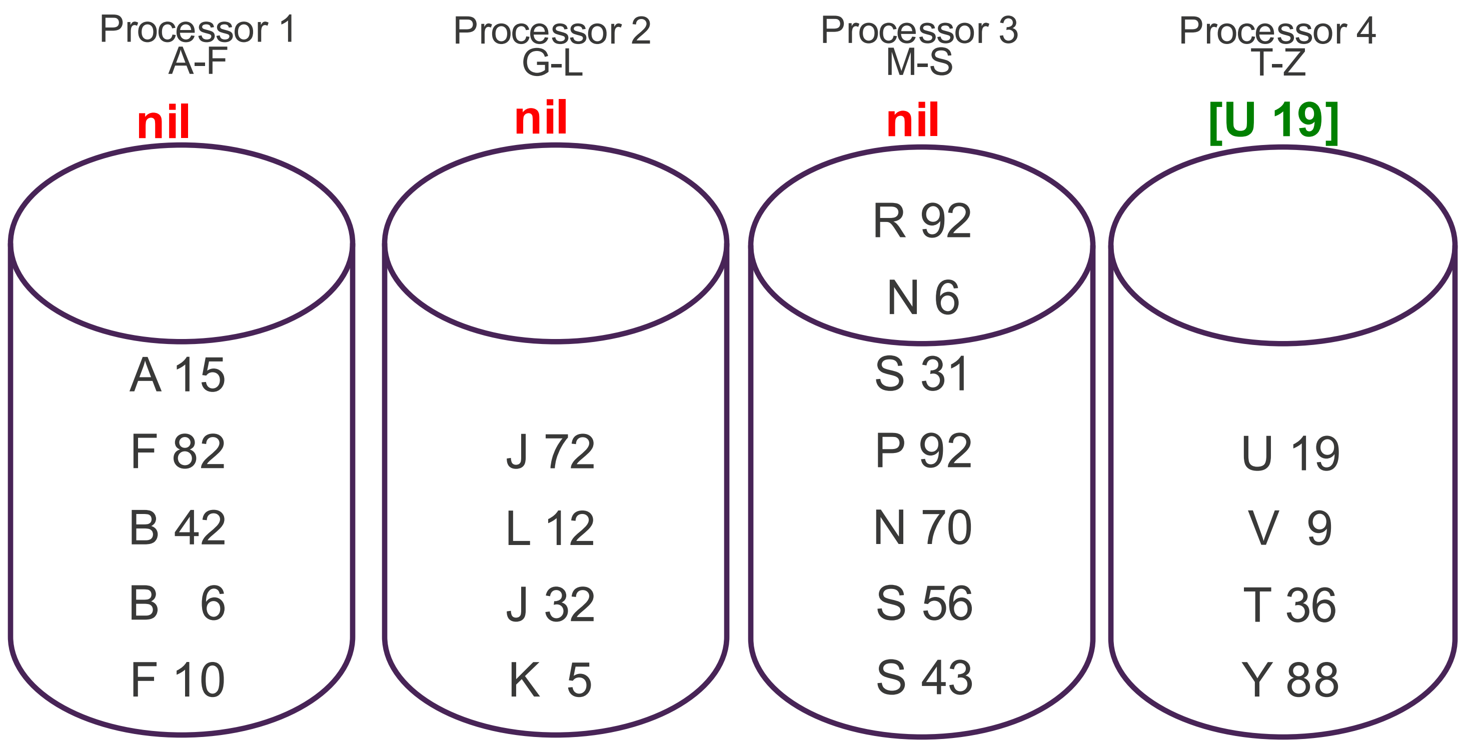


Figure 3.6 *Random-unequal* data partitioning

3.2. Data Partitioning (cont'd)

- **Random-unequal data partitioning**

Search 19



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)

- **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

  - **Single attribute with multiple partitioning methods:**
    - Hybrid-Range Partitioning Strategy (HRPS)
      - Step 1: Range partitioning
      - Step 2: Round-robin partitioning

  - **Multiple attributes:**
    - Multiattribute Grid Declustering (MAGIC)
    - Bubba's Extended Range Declustering (BERD)

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 fragments 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

HRPS example

Step 1: Generate fragments of data using range partitioning

Size of fragments (FC)

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

= 500 / 5 = 100

M = number of participating processors

Step 2: Distribute fragments of data to processors using round-robin partitioning

1-100 501-600	101-200 601-700	201-300 701-800	301-400	401-500
P1	P2	P3	P4	P5

3.2. Data Partitioning (cont'd)

- **Hybrid-Range Partitioning Strategy (HRPS)**
  - **Case 1: *M = N (# participating processors = # available processors)***
  - 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: In 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) SIname='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 (SIname 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

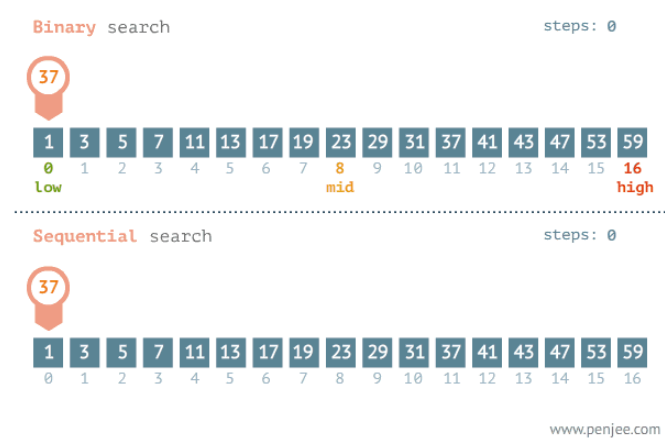
		Slname					
		A–D	E–H	I–L	M–P	Q–T	U–Z
	98000–98100	1	2	3	4	5	6
	98101–98200	7	8	9	10	11	12
Sid	98201–98300	13	14	15	16	17	18
	98301–98400	19	20	21	22	23	24
	98401–98500	25	26	27	28	29	30
	98501–98600	31	32	33	34	35	36

3.3. Search Algorithms

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

Local search methods to be applied on each data partition

- **Parallel** search algorithms:
  - Processor activation or involvement
  - Local searching method (linear or binary)
  - 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