3/11/23

1. Database System Architectures

- Types of DBMS

a) centralized and client-server system

b) Server system architectures.

c) & Parallel Systems

d) Distributed systems

e) Network Types

a) centralized database system:- → one sys, one db, one appl" every thing on a single machine.

- centralized database system consists of single data server and multiple users connected to the server
- Database is kept at single physical location.
- Different users can acess this database from local machines or from remote machine.
- Management of the system and data are controlled centrally. Eg:- MS-office.

Advantages :- • operations such as insert, update, delete, backup, recovery are easier to Perform.

• security interms of authorization., Reliable (security)

Disadvantages:- • when central computer or database system goes down, no one can access the database.

• All terminals must have connection with the central computer and this increase cost in networking.

types of DBMS

→ on the basis of single user → single user
                              → multi user

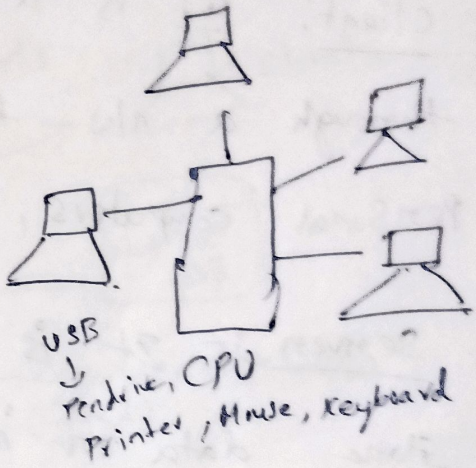
→ on the basis of location of servers/site
- centralized
- Distributed
- Parallel
- client-server

centralized :- single resources, processor, storage devices.

→ single location (I/O devices)

→ support multiple users.

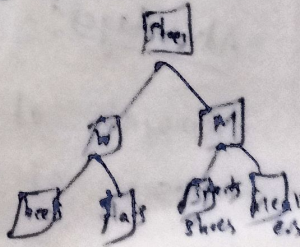
→ one Administrator.



→ What is an advanced database?

A) Advanced database systems try to meet the requirements of present-day database appl's by offering advanced functionality in terms of data modeling, multimedia datatype support, data integration capabilities, query languages, system features, and interfaces to other worlds.

Data model :- Data models describe how a db's logical structure is represented.



Most advanced databases are

→ MySQL
→ PostgreSQL
→ Oracle
→ Microsoft SQL Server.

Client - Server Systems :- (open)

1. Client 2. Server

Client: It is a device which request online for data through a n/w from a server. This may be Personal computers, laptops, smartphone. (Front end)

Server :- It is a large workstation which holds or store data on info and sends to the client on request through a n/w. Servers are also called backend.

In client server db system, the DBMS appl's and tools are available on all clients, but the actual database management system and physical database is available on servers.

Ex: Artificial intelligence appln, online transaction processing (Govt jobs)

Advantages:- 1. This system helps to manage a large amount of data to find out all possible successful uses of existing data.

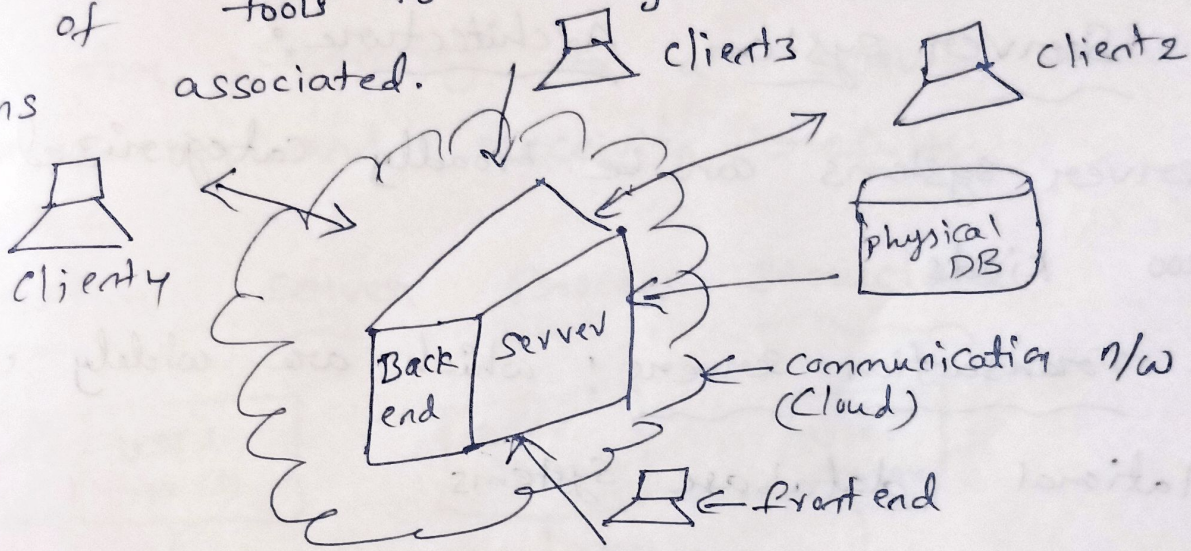
2. It is more comfortable and lenient (tolevent than expected)

3. Even a single user can entertain multiple client at once.

4. Reduce the cost of db.

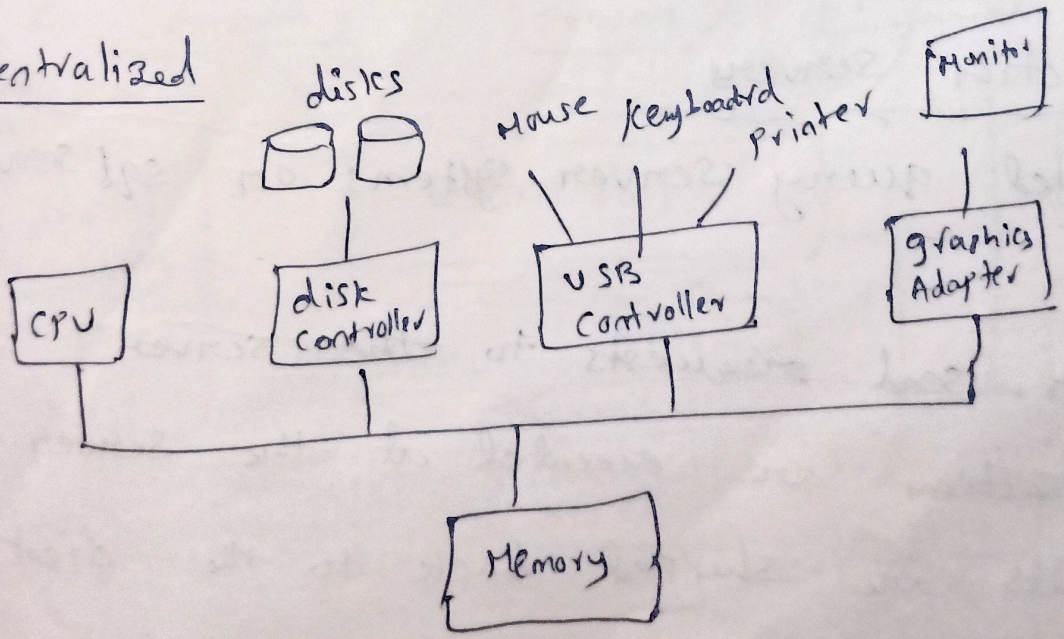
disadvantages: 1. Less security and reliability

2. The architecture can not provide complete range of tools to manage the n/w and systems associated.

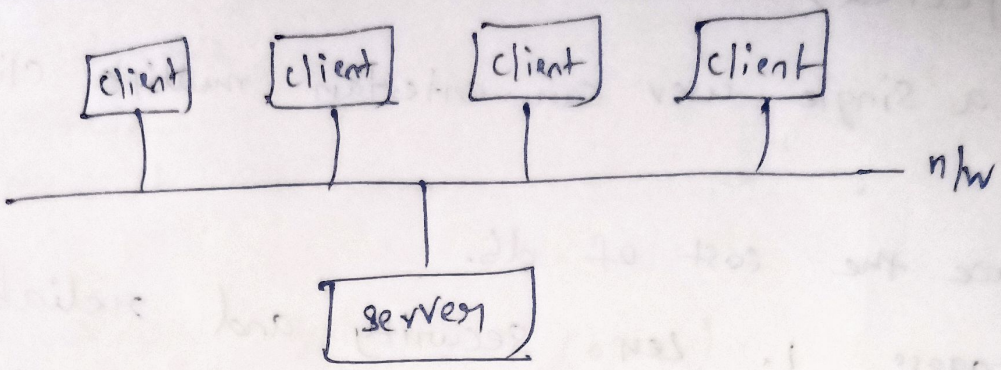


online form ← Ex

centralized



client - server :-



→ 1/2/23

Server system Architecture :

Server systems can be broadly categorized into two kinds

1. Transaction servers : which are widely used in relational database systems

2. data servers : used in object-oriented db sys

Transaction servers

Also called query server systems or sql server systems

→ clients send requests to the server

→ Transactions are executed at the server

→ Results are shipped back to the client.

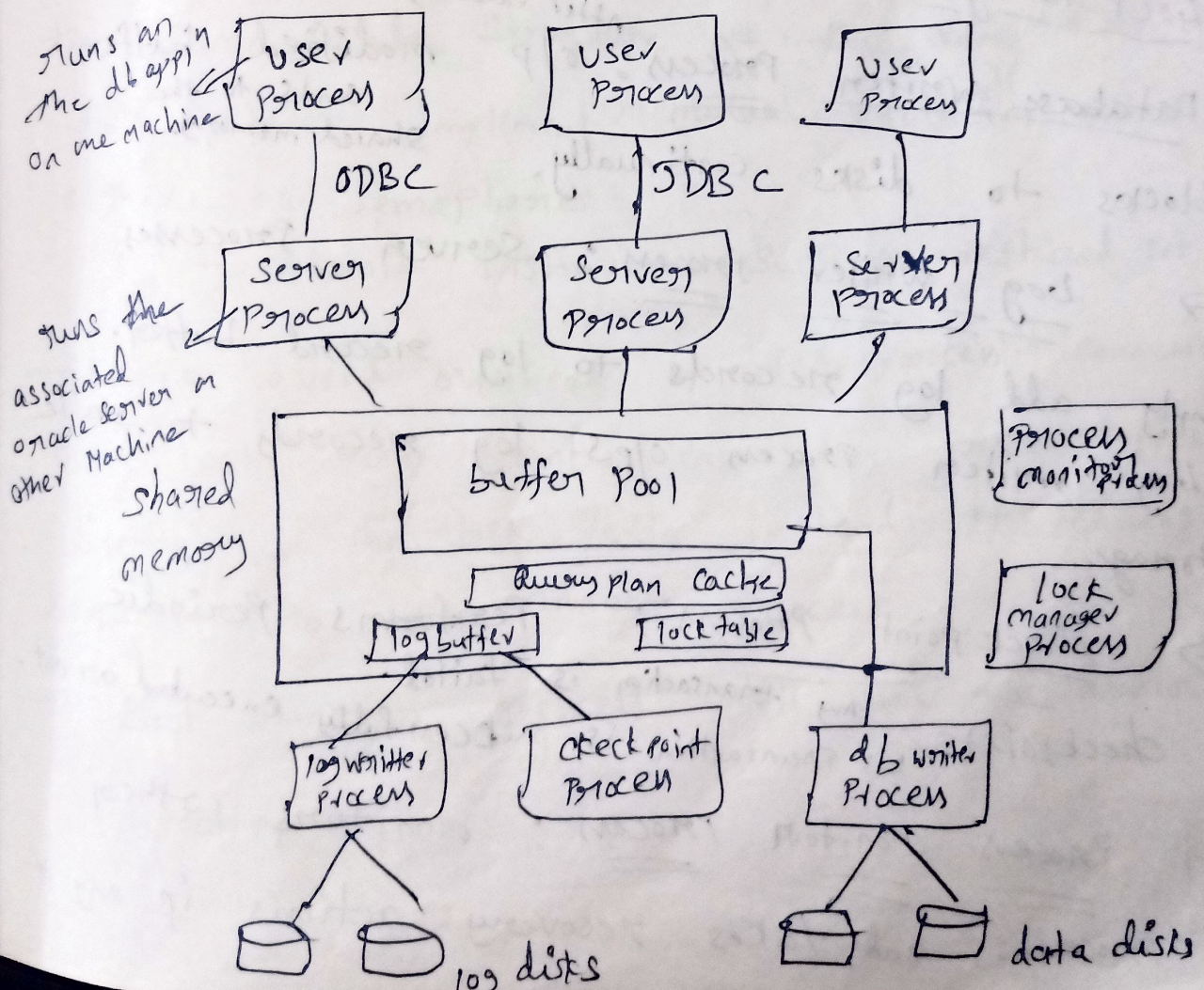
→ Requests are specified in SQL and communicated to the server through a remote procedure call (RPC) mechanism.

→ Transactional RPC allows RPC calls to form a transaction.

→ Open database connectivity (ODBC) is a C language appln program interface standard from Microsoft for connecting to a server, sending SQL requests and receiving results.

→ JDBC

Transaction Server Process Structure



→ A typical transaction server consists of multiple processes accessing data in shared memory.

→ Server Processes: They receive user queries (transactions), execute them and send results back.

→ Processes may be multithreaded, allowing single process to execute several user queries concurrently.

→ Typically multiple multithreaded server processes.

Lock manager: Lock manager process

Database writer process: After executing the query o/p modified buffer blocks to disks continually. sends to the shared memory.

→ Log writer process: Server processes simply add log records to log record buffer. Log writer process o/p's log records to stable storage.

→ checkpoint process: Performs periodic checkpoints. Any transaction is failed. If transaction is successfully executed or not

→ Process monitor process: monitors other processes, and takes recovery actions if any of

other processes fail.

Eg: aborting any transactions being executed by a server process and restarting it.

Shared memory: contains shared data

→ Buffer Pool –

→ Lock table – it is a table who are granting and releasing all the logical which are available into the lock table

→ Log buffer

→ cached query Plans (reused if same query submitted again)
retrieve forms cache itself
same data

All db processes can access shared memory

→ To ensure that no two processes are accessing the same data structure at the same time, db's systems implement mutual exclusion using either → OS semaphores
→ Atomic instructions such as test and set

→ To avoid overhead of interprocess communication for lock req / grant, each db process operates directly on the lock table. instead of sending requests to lock manager process.

→ Lock manager process still used for deadlock detection.

What is Database Architecture

A database Architecture is a representation of DBMS design. It helps to design, develop, implement, and maintain the db management System.

→ Lock manger Process is responsible for granting and releasing the locks. for avoiding the Concurrency control.

Data Servers :- used in object-oriented db System. because now-a-days instead of using the relational db system.

→ used in high-speed LANS, in cases where the clients are comparable in processing power to the server → the tasks to be executed are compute intensive.

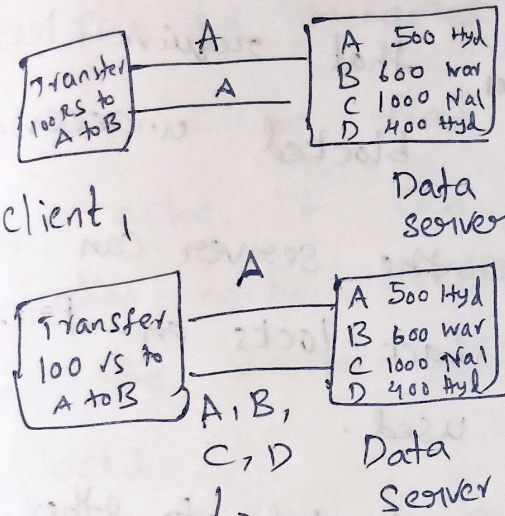
→ In most of the time server will be busy so we can execute the query (transaction) in client side also (capable)

→ client can ask for data at server side

- The tasks to be executed are compute intensive.
- This architecture requires full back-end functionality at the clients.

→ issues:-

1) - Page-shipping versus Item-shipping (how much data single tuple (Record) shifted to the server side) — entire page shifted to client side.



shipping ↳ entire page to the client side

- Fetching items even before they are requested is called prefetching.
  ↳ Page shipping can be thought of as a form of prefetching.

- Unit of communication for data can be of
  - Coarse-greater (Page) granularity : Page
  - Fine granularity : Tuple (referred as item) ↙ smaller (1 tuple or 1 record)

- Item shipping
  - overhead of message passing is high compared to data transmit.

→ Instead, when a particular item is requested, other items are also sent which may be used in future.

Adaptive lock Granularity :-

→ Locks are usually granted by the server for the data items that it ships to the client M/c's.

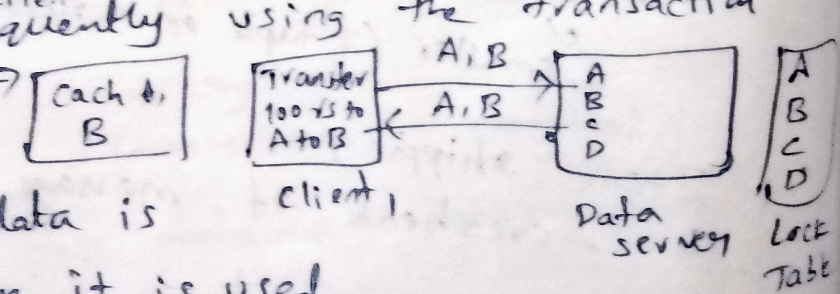
Disadvantage of Page Shipping

- client machines may be granted lock on a page.
- Locks all items contained in the page, even if the client is not accessing some items in the page.
- other client machines that require locks on those items may be blocked unnecessarily.
- Lock de-escalation: the server can request its clients to transfer back locks on prefetched items that are not used.
- Those locks can be allocated to other clients.

3) Data Caching :-

→ Data that are shipped to the client can be cached at client even after completing the transaction. (frequently using the transaction)

→ Cache coherency:



check that the cached data is up-to-date before it is used

→ Ensure that the data is not updated by diff client after caching.

→ Message must still be exchanged with the server to check its validity and to acquire lock on the data again.

4) Lock caching

→ If the use of data is partitioned among the clients, with clients rarely requesting data that are also requested by other clients.

→ Locks can be retained by client machine.

→ If ~~the~~ client find data and lock permission in the cache, it can access the data without communicating with server.

→ But the server must keep track of all cached locks.

→ If other ~~&~~ client requests a lock on data item, the server calls back locks from clients when it receives conflicting lock request.

→ client returns lock once no local transaction is using it.

→ Similar to descalation, but across transactions.

Parallel Systems: Parallel db systems consists multiple processors and multiple disks connected by a fast interconnection n/w.

→ A Coarse-grain Parallel machine consists of a small of Powerful Processors.

→ A massively Parallel or fine grain Parallel Machine utilizes thousands of smaller Processors.

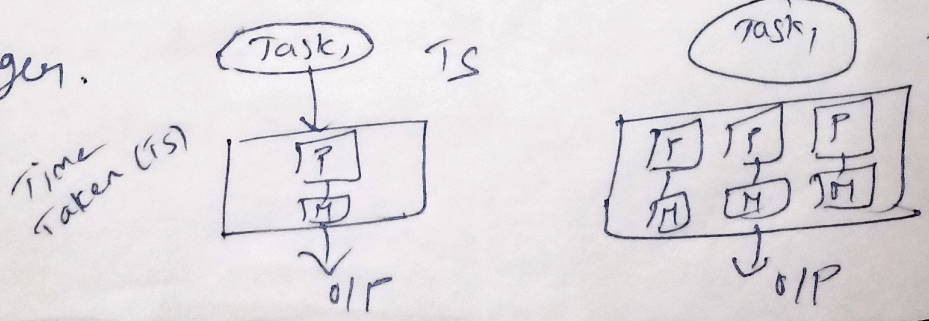
⇒ Two main Performance measures

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 is submitted.

Speed-up and Scale-up

Speed up: a fixed sized Problem executing on a small system is given to a system which is N-times larger.



Measured by

$$\text{Speedup} = \frac{\text{Small system elapsed time}}{\text{large system elapsed time}} = \frac{T_S}{T_L}$$

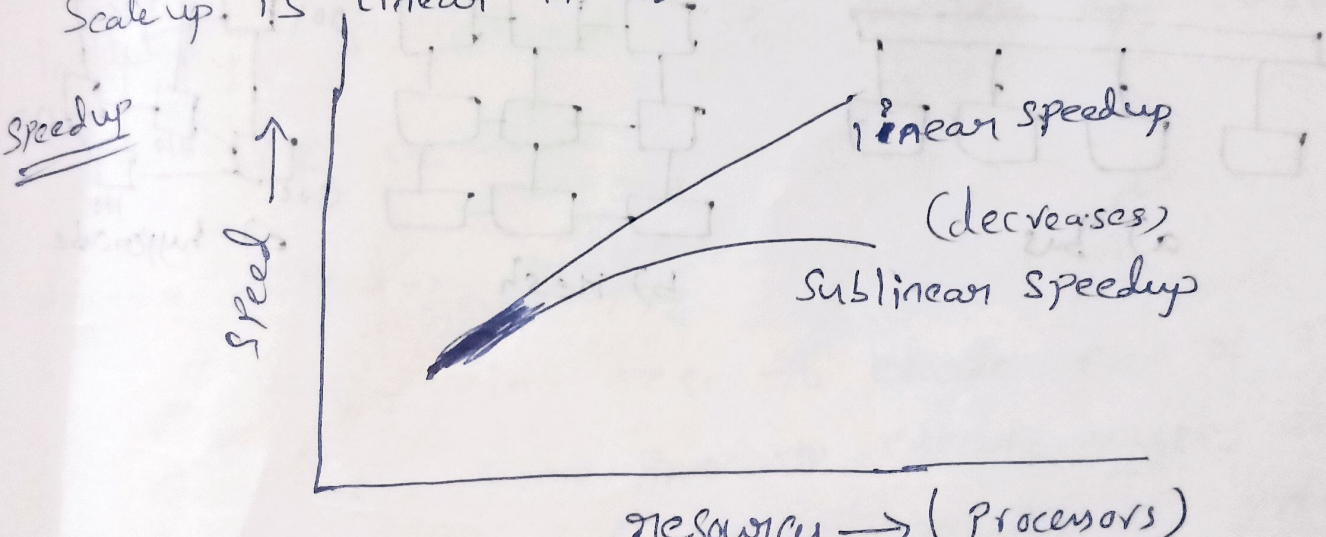
speed up is linear if equation equals N.

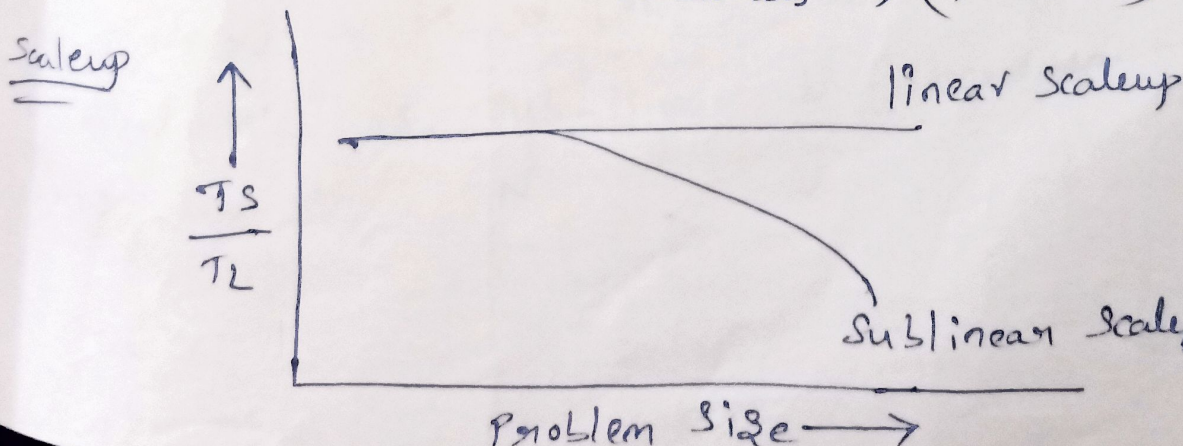
Scale up increase the size of both the problem and the system N-times large system used to perform N-time larger job

Measured by:

$$\text{Scaleup} = \frac{\text{Small system small problem elapsed time}}{\text{big system big problem elasped time}}$$

Scale up is linear in equation equals 1





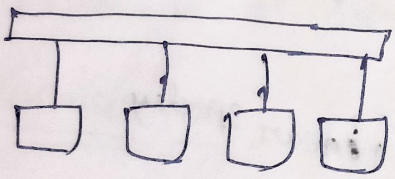
Interconnection N/W Architecture

Bus: System components send data on and receive data from a single communication bus;

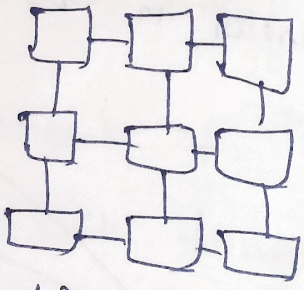
Does not scale well with increasing parallelism.

Mesh: Components are arranged as nodes in grid, and each component is connected to all adjacent components.

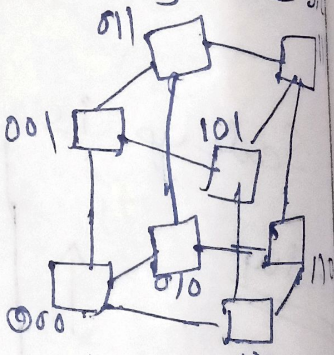
Hypercube: Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit.



a) bus



b) Mesh



c) hypercube

Parameters for Parallel Database

→ To judge the performance of parallel database we are using the parameters

1) Response time: It is the time taken to complete a single task for given time.

2) Speed up in Parallel db

→ speed up is the process of increasing degree of (resource) parallelism to complete a running task in less time.

→ The time required for running task is inversely proportional to number of resources.

$$T(R) \propto \frac{1}{\text{no. of res}}$$

formula Speed up = T_S / T_L

T_S = Time req to execute task of size Q

T_L = Time req to execute task of size N+Q

linear speed up is N (no. of resources)

speed-up is sub-linear if speed up is less than N.

(3) Scaleup in Parallel database

Scale-up is the ability to keep performance constant, when no. of Process and resources increase Proportionally

Formula :

Let Q be the task & Q_N the task where N is greater than Q.

T_S = Execution time of task Q on smaller M/c M_S

T_L = Execution time of task Q on smaller M/c M_2

Scaleup = T_S / T_L

Scale up = No of Large M/c / Vol of smaller M/c

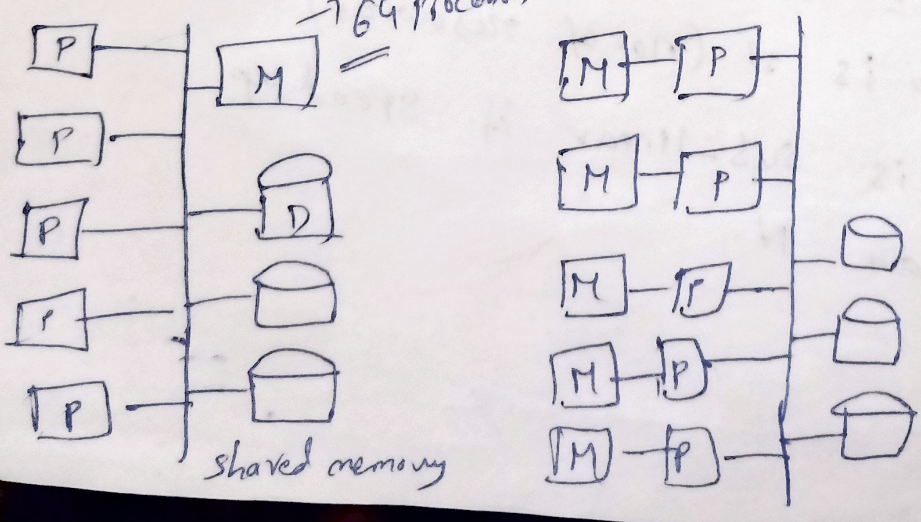
Parallel Database Architectures

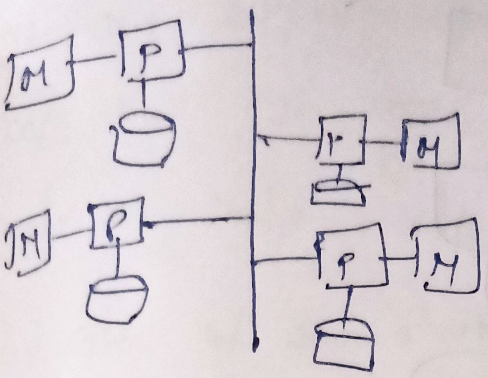
1) Shared memory - processors shares a common Mem

2) Shard disk - Processors share a common disk

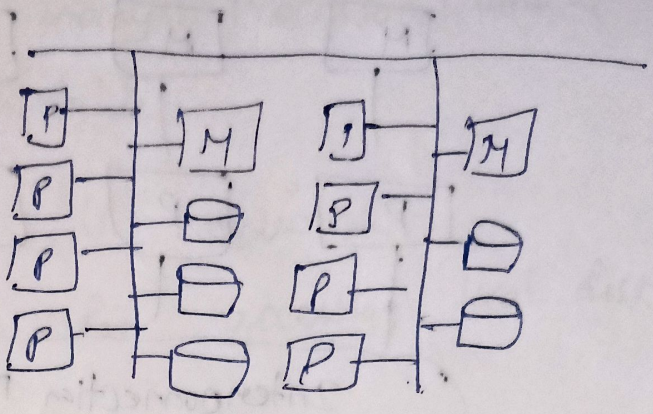
3) shared nothing - Processors share neither a common memory nor common disk.

4) Hierarchial - hybrid of the above architectures





shared nothing



Hieravchial

Shared Advantages

→ Data is easily accessible to any Processors

→ one Processor can send msg to other efficiently.

Disadv

→ Waiting time of Processors is increased due to more no of Processors

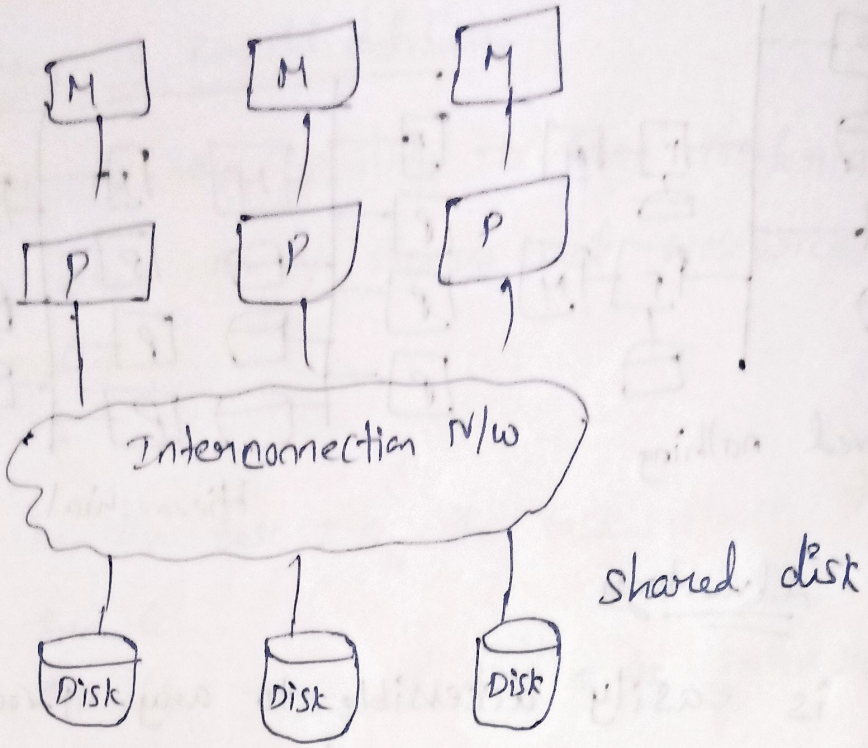
→ Bandwidth Problem.

Shared disk:

→ shared disk system uses multiple Processors which are accessible to multiple disk via intercommunication channel and every Processor has local memory.

→ Each Processor has its own memory so that data sharing is efficient.

→ The system built around this system are called as clusters.



Adv

1) Fault tolerance is achieved using shared disk system.

Fault tolerance: If a Processor or its memory fails the other Processor can complete the task. This is called as fault tolerance.

disadv:

→ shared disk sys has limited scalability as large amount of data travels through the interconnection channel.

→ If more Processors are added the existing Processors are slowed down.

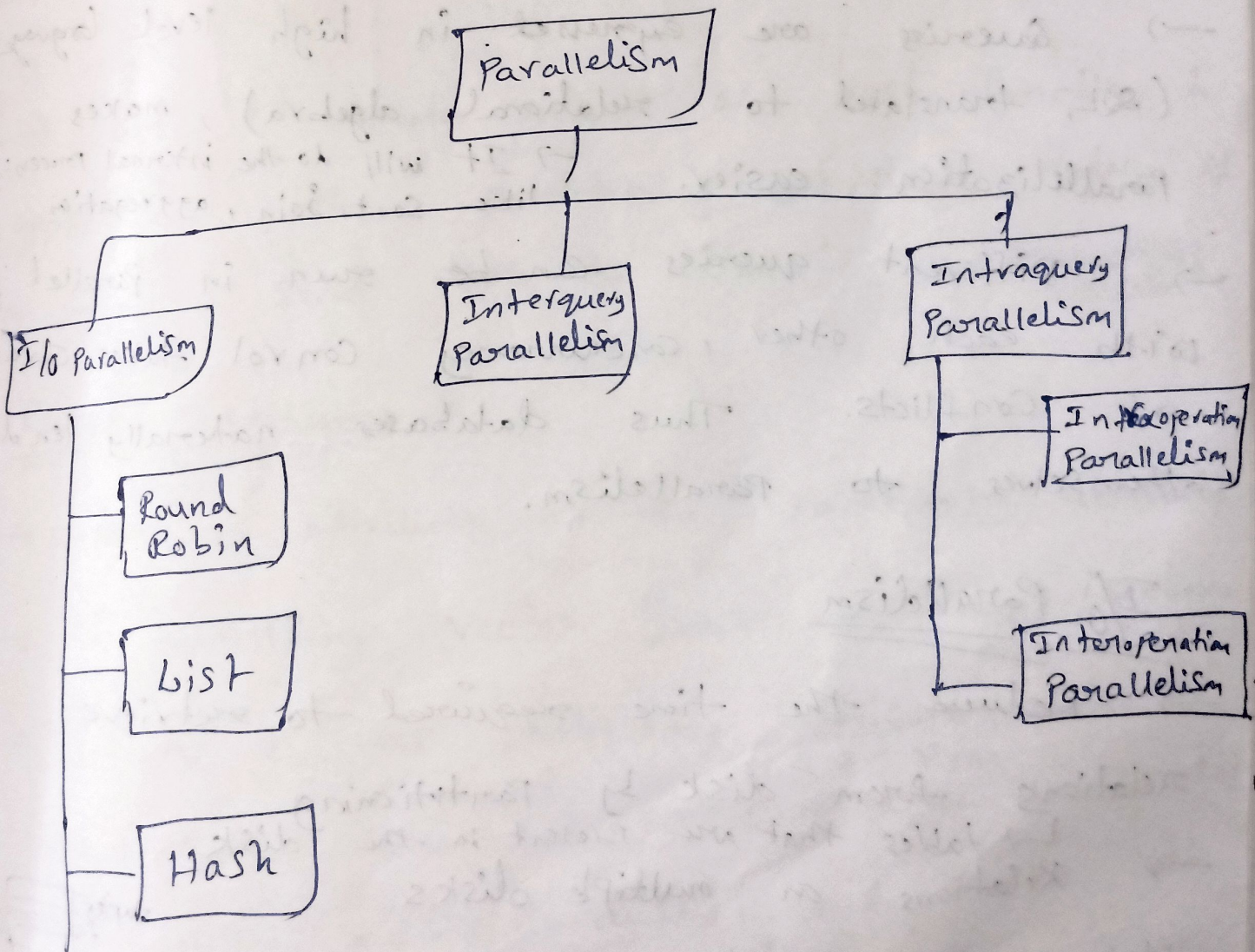
Shared nothing:- Adv 1) No. of Processors and disk can be connected as per the requirement in shared nothing disk sys.

→ It can support for many Processors, which makes the sys more scalable.

Disadv 1) Data Partioning is required

2) cost of communication for accessing local disk is much higher.

→ How Parallelism is achieved



Parallelism in Databases

→ Data can be Partitioned across multiple disks for Parallel I/O. → More no. of disks we can store more data

→ Data can be partitioned and each processor can work independently on its own partition.

→ Individual relational operations (e.g. sort, join, aggregation) executed in Parallel.

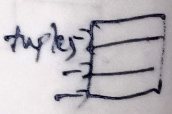
→ Queries are expressed in high level language (SQL, translated to relational algebra), makes Parallelization easier.
↳ It will do the internal process like sort, join, aggregation

→ Different queries can be run in Parallel with each other, Concurrency control takes care of conflicts. Thus databases naturally lend themselves to Parallelism.

I/O Parallelism

→ Reduce the time required to retrieve relations from disk by Partitioning
↳ tables that are present in the disk.

→ Relations on multiple disks



Horizontal Partitioning – tuples of a relation are divided among many disks such that each tuple resides on one disk. (number of disks =

Partitioning techniques (number of disks = n):

Round-Robin:

Send the I^{th} tuple inserted in the relation to disk $i \bmod n$.

Hash Partitioning

→ Choose one or more attributes as the Partitioning attributes.

→ choose the hash function h with range $0 \ldots n-1$.

→ Let 'i' denote result of hash function h applied to the Partitioning attribute value of a tuple. Send tuple to disk i.

Range Partitioning

→ Choose an attribute as the Partitioning attribute

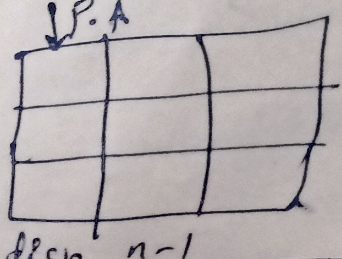
→ A Partitioning vector $[v_0, v_1, \ldots, v_{n-2}]$ is chosen ($n=3$)

→ Let v be the Partitioning attribute value of tuple. 1) Tuples such that $v_i \le v \le v_{i+1}$ go to disk i+1.

2) Tuples with $v < v_0$ to disk 0 and tuples with $v \ge v_{n-2}$ go to disk n-1.

Vector $[v_0, v_1]$ Tuple with $v < v_i$ go to disk 0

Tuple with $v > v_{n-2}$ go to disk n-1



Eg: with a Partitioning vector [5, 11], a with Partitioning attribute value of 2 will disk 0, a tuple with value 8 will go to while a tuple with value 20 will go to

Round Robin

i - record no
n - no. of disks

i mod n is used for splitting records

Record	id	Name	Branch
1	1	Sam	Hyd
2	2	Ram	War
3	3	Kiran	Nal
4	4	Kriti	Hyd
5	5	Athar	Nal

no. of disks = 3

1 mod 3 = 1 disk1
2 mod 3 = 2 (disk2)

Disk 0 Disk1 Disk2
2 Ram War

i mod 3 = disk

1 mod 3 = 1
2 mod 3 = 2
3 mod 3 = 0
4 mod 3 = 1
5 mod 3 = 2
6 mod 3 = 0

Round Robin

Disadv

→ only suitable for full table scans
→ Not suitable for Point queries or range queries.

L) select * from employee where name = 'Sam';

Range

select * from employee where id between 3 and 5

Range:- Range Partitioning strategy partitions the data based on the partitioning attribute values.

→ We need to find set of range vectors on which we are about to partition.

Eg:- the records with a salary range 100 to 5000 will be in disk1, 5001 to 10000 in disk2

id	Name	Branch	Salary

Salary Key

Sal < 10000 P0

Sal 10000 < 30000 - P1

Sal > 30000 - P2

List Partitioning

chennai, vellore, Hyd, tamilnadu, mumbai

↳ based on Branch.

Tamilnadu Partition

chennai

vellore

Maharashtra

Mumbai

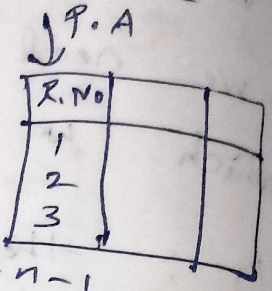
Pune

→ List Partitioning enables you to explicit control how rows map to partitions by specifying a list of discrete values for the Partitioning key in the description for each Partition.

→ for a table with a Branch column as the Partitioning key, the tamilnadu Partition

might contain Values chennai and Vellore. the Maharashtra Partition might contain Mumbai

Hash :- We need to choose one or more attributes from the table.



we need to apply hash function h with range 0 ... n-1
↳ no. of disks

0 - 2
→ 0, 1, 2

if zero is the result of hash function the tuple will move to disk 0.

→ Comparison of Partitioning Techniques

1) Scanning the entire relation - Scan queries
↳ table

2) Locating a tuple associatively - Point queries
↳ A particular tuple E.g A = 25

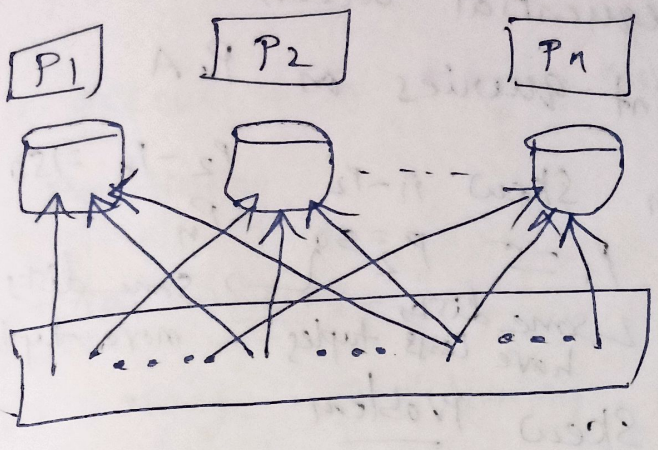
3) Range queries 10 and 25
Range
Sal b/w 10000 & 20,000

Round robin Partitioning

→ Best suited for sequential Scan of entire relation on each query.
- All disks have almost an equal no. of tuples
- Retrieval work for entire relation is
they well balanced b/w disks. (equal distribution

→ Point queries and range queries are difficult to Process.

- No clustering — tuples are scattered across all disks.
  ↳ no grouping



Given data.

Hash Partitioning

→ Good for sequential access

• Assuming hash function is good and Partitioning attributes from a key tuples will be equally distributed b/w disks. (Primary key)

• Retrieval work for entire relation is then well balanced b/w disks.

→ Good for Point queries on P.A is a key

• Can lookup single disk, leaving others available for answering other queries.

• Index on P.A can be local to disk, making lookup and update more efficient.

→ No clustering, so difficult to answer range queries.

Range: another group [5 ∞] another group

→ Provides data clustering by partitioning attribute value.

→ Good for sequential access

→ Good for point queries on P.A

Problem Execution skew $P_1 - P_2$ $P_2 - P_4 = 150$ P_4

(imbalance b/w disks) → $P_1 = 5$ $P_2 = 50$ → some disks have more tuples

→ some disks have less tuples

Handling of Skew Problem

→ Partitioning a Relation accross disks

1) If a relations only a few tuples which will fit into a single disk block, then assign the relation to a single disk.

2) Large relations are preferably partitioned across all the available disks.

3) If a relation consists of m disk blocks and there are n disks available in the system then the relation should be allocated min(m, n) disks.

→ The distribution of tuples to disks may be skewed → some disks have many tuples while others may have fewer tuples

→ Types of Skew:

1) Attribute-value Skew:

a) All the tuples with the same value for the partitioning attribute end up in the same partition.

b) Can occcur with range-partitioning and hash-partitioning.

2) Partition Skew:-

a) with range-partitioning, badly chosen partition vector may assign too many tuples to some partitions and too few to others.

b) Less likely with hash-partitioning if a good hash-function is choosen.

→ Handling Skew in Range-partitioning

→ To create a balanced partitioning vector (assuming attribute forms a key of the relation)

a) Sort the relation on the partitioning attribute.

b) Construct a partition vector by scanning the relation in sorted order as follows

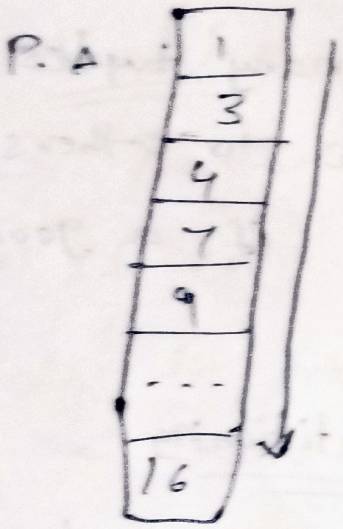
After every $\frac{1}{n\text{-th}}$ of the relation has been read, the value of the partitioning attribute

of the next tuple is added to the Partitio vector.

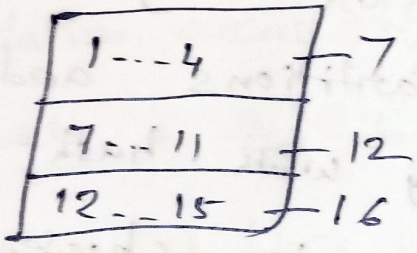
b) n denotes the the number of Partitions to constructed

c) Duplicate entries or imbalances can result i duplicates are present in P.A.

Perform Partitioning using the Balanced Partitio vector.



Balanced Partitioning Vector



[7, 12, 16]

<7
=7 to 12
=12 to 16
>16

→ Handling Skew using Histograms

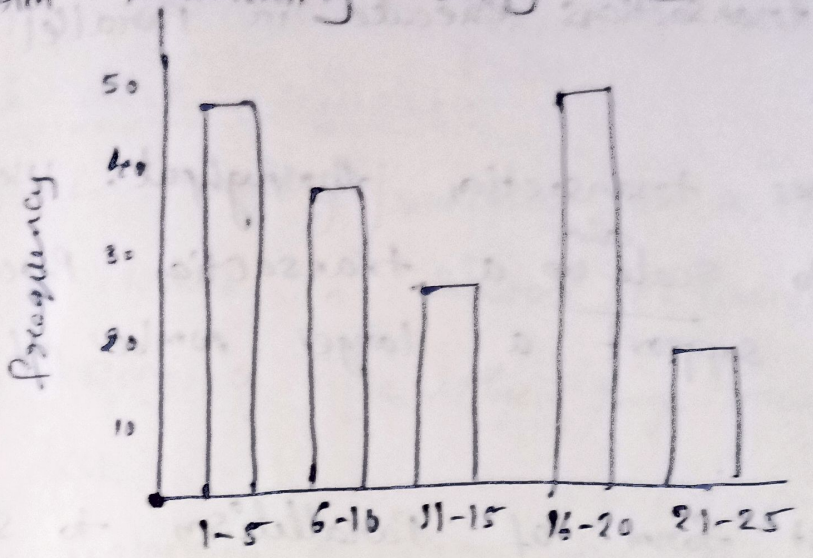
—> Balanced Partitioning Vector can be constructed from histogram in a relatively straightforward fashion.

—) Assume uniform distribution within each range of the histogram.

—> Histogram can be constructed by Scanning relation, or Sampling (blocks containing tuples of relation)

→ Histograms can be stored in the Sys Catalog

Perform Partitioning using the balanced P.V



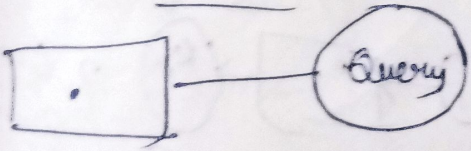
→ Handling Skew using virtual Processor Partitioning

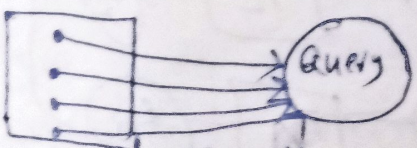
Skew in range Partitioning can be handled elegantly using virtual Processor Partitioning

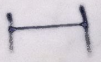
→ Create a large no. of Partitions (Say 10 to 20 time the no. of processors)

→ Assign virtual Processors either in round-robin fashion or based on estimated cost of Processing each virtual Partition.

Query Parallelism







Speedup = 4min/1m
= 4

→ In parallel DL Sys to improve the performance of the Sys, Parallelism is used.
→ It is achieved through query parallelism.

Interquery Parallelism

→ Different queries / transactions execute in parallel with one another.

→ Increases transaction throughput! Used primarily to scale up (aim) a transaction processing system to support a larger number of trans per second.

→ Easiest form of "Parallelism" to support particularly in a shared memory parallel db, because sequential db systems support concurrent processing.

→ More complicated to implement to on shared disk or shared-nothing architectures.

1) Locking and logging must be coordinated by passing msgs b/w processors.

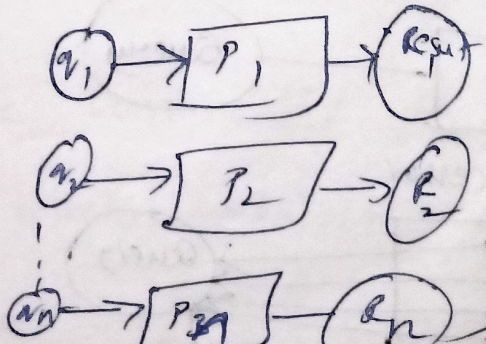
2) Data in a local buffer may have been updated at another processor.

cache-coherency → check that the cached data is up-to-date or not has to be maintained - reads and writes of data in buffer must find latest version of data.

→ Select * from emp;

→ Select * from Dept where mgrname = 'Steve';

→ select Furnisher_Name, cost from Furniture;



Cache Coherency Protocol

Example of Cache Coherency protocol for Shared disk Systems:

→ Before reading / writing to a Page, the Page must be locked in shared / exclusive mode.

→ on locking a Page, the Page must be read from disk.

→ Before unlocking a Page, the Page must be written to disk if it was modified.

→ More Complex Protocols with fewer disks read / writes exist.

→ Most common nothing

→ Cache Coherency Protocols for Shared Systems are similar. Each db Page is assigned a home Processor. Requests to fetch the Page or write it to disk are sent to the home Processor.

→ Intraquery Parallelism

The same query is being Parallelized — Means Part of same query is given to one Proccesor and another part of same query is given to other processor / disk.

→ Select * from Emp Dept where Emp.dno = Dept.dno;

→ Execution of single query in Parallel on multiple Processors / disks. → Purpose / aim

→ Important to speeding up long-running queries

Two complementary forms of intra-query Parallelism.

1) Intra-operation Parallelism – Parallelize the execution of each individual operation in the query. (Same operation is being is Parallelized).

Ex Sort – 1 lakh

50,000 – 50,000
1 P 2 P.

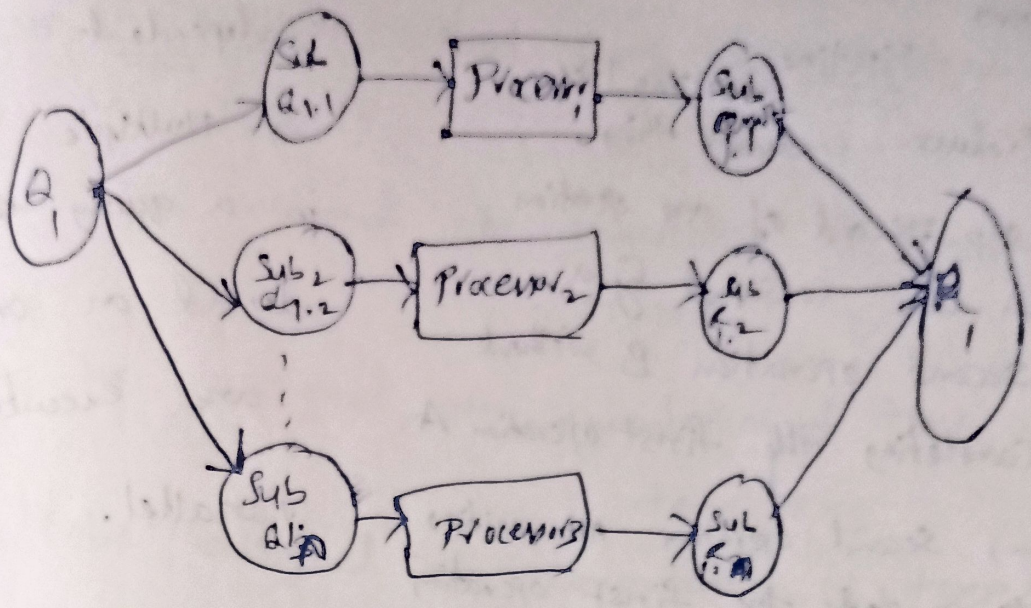
Eg: Parallel sort, Parallel search

→ This form scales better with increasing Parallelism because the number of tuples processed by each operation is typically more than the no. of operations in a query.

Inter-operation Parallelism – Execute the different operations in a query expression in Parallel.

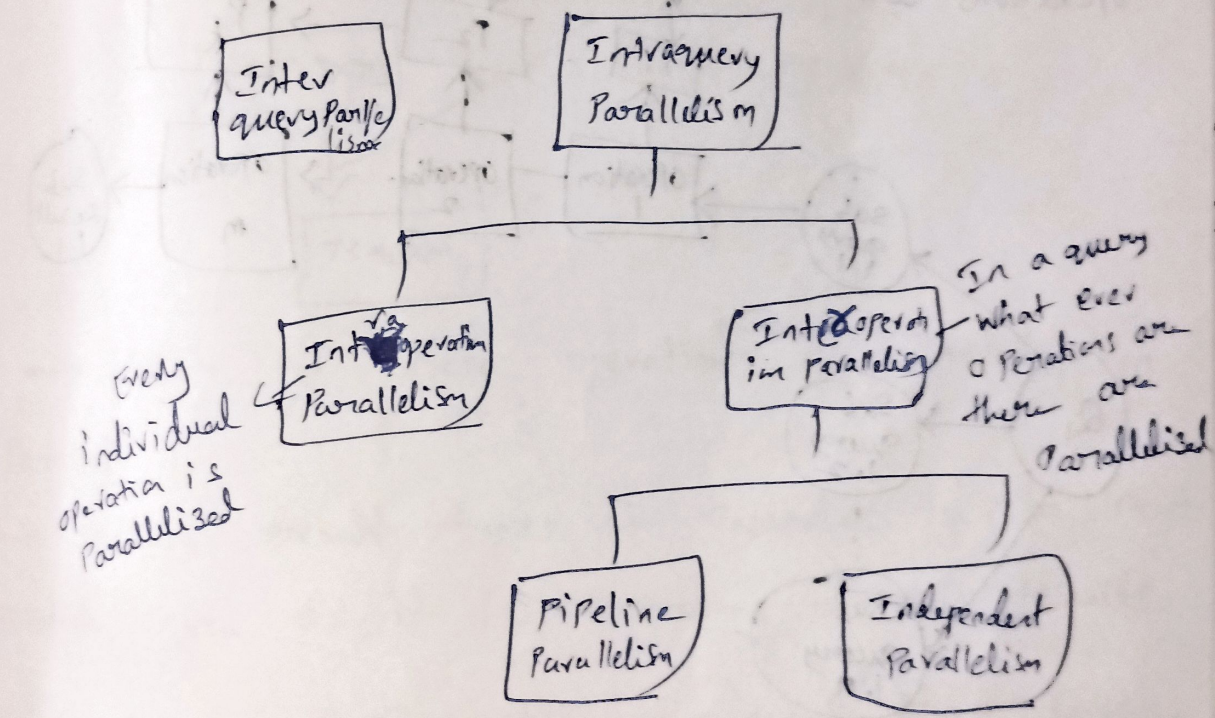
Example: simultaneous sorting or searching

→ The operation Sorting and Searching are Parallelized.



Intra query Parallelism

→ Forms of query Parallelism



→ Interoperation Parallelism : within a same query or transaction different operations are concurrently executing.

Pipeline working like Producer-consumer thing.

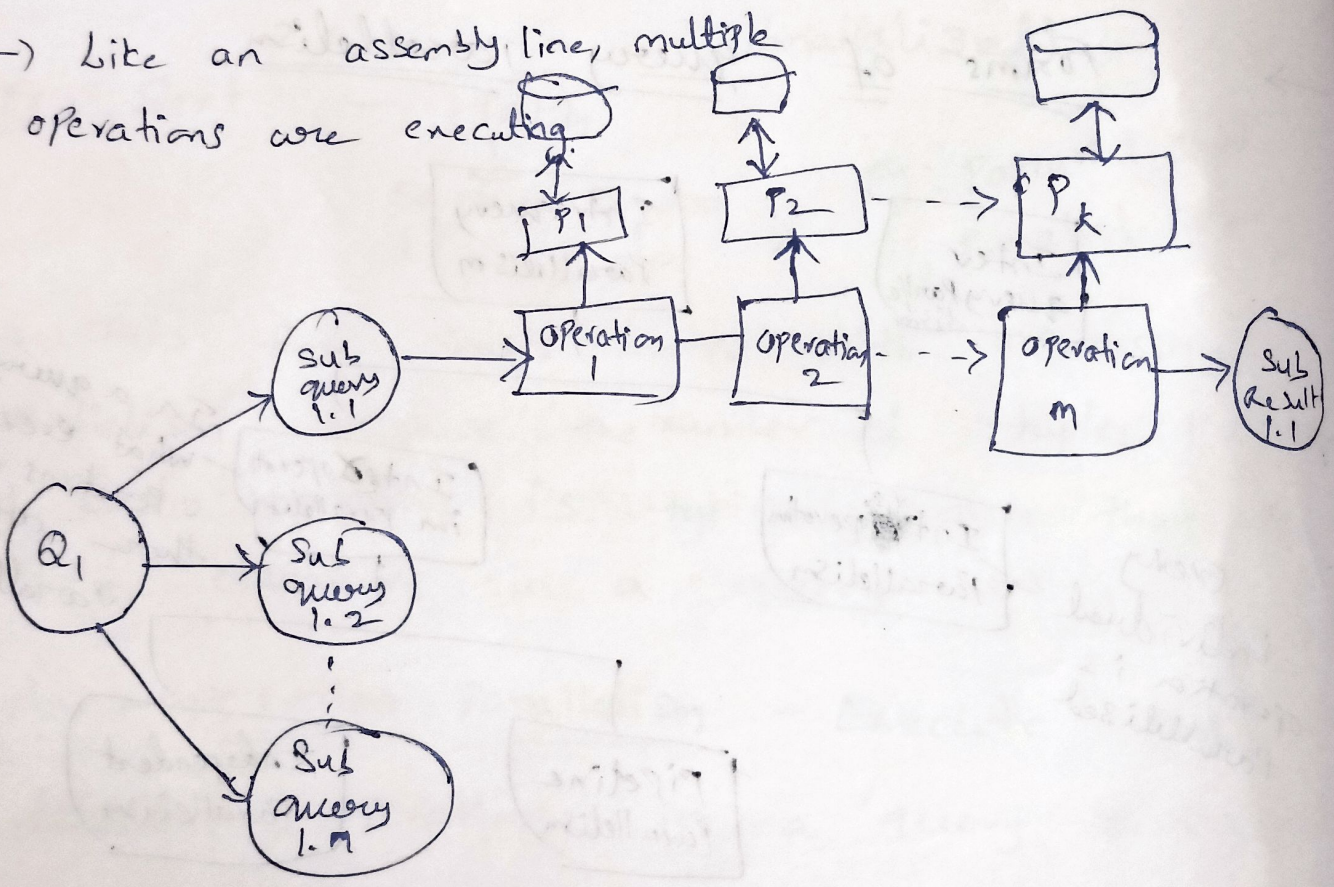
o/p record of one operation A are consumed by a second operation B without completing the first operation A.

→ Second operation not waiting to complete the first operation Result of operation A is i/p of second operation B.

→ Like an assembly line, multiple operations are executing

Independent-

→ Multiple operations in a query that do not depend on one another are executed in parallel.



Pipeline Parallelism

Consider a join of four relations

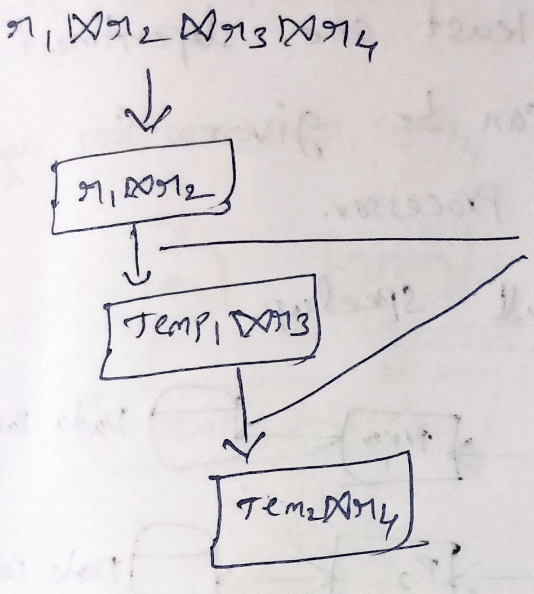
$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$

Set up a pipeline that computes the three joins in parallel

Let P_1 be assigned to computation of $temp_1 = r_1 \bowtie r_2$

and P_2 be assigned to computation of $temp_2 = temp_1 \bowtie r_3$

P_3 be assigned to computation of $temp_2 \bowtie r_4$



→ Each of these operations can execute in parallel.

→ sending result tuples it computes to the next operation even as it is consuming further results.

→ Provided a pipelineable join evaluation algorithm (e.g. indexed nested loop join) is used.

→ Factors Limiting utility of Pipeline Parallelism

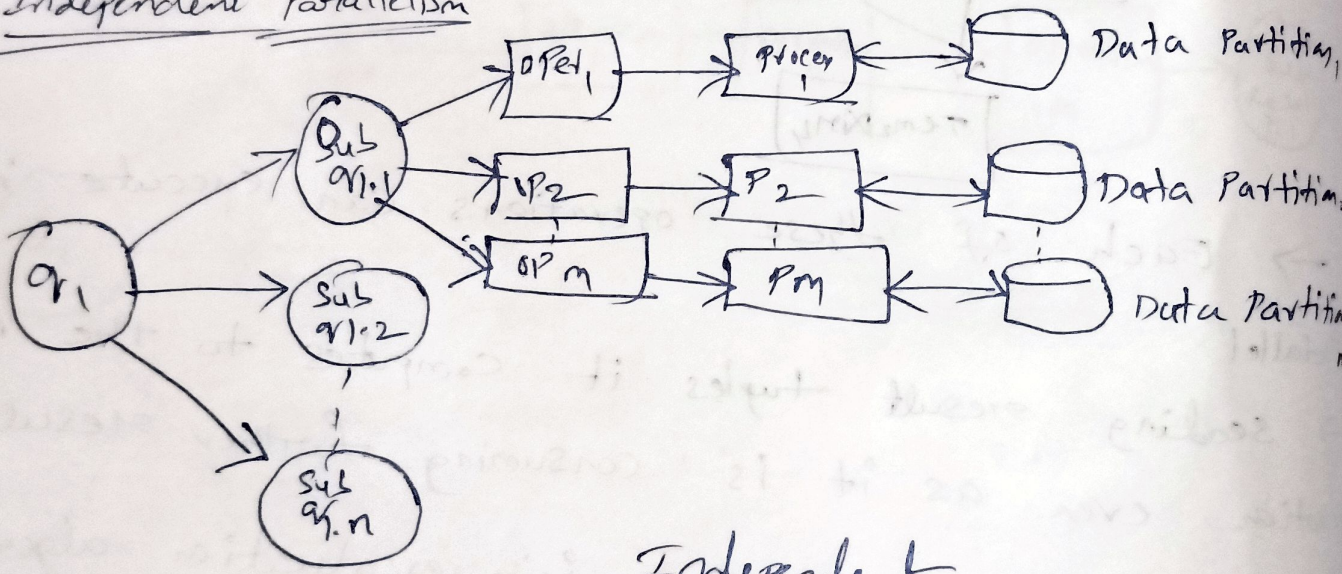
1. Pipelined Parallelism is not the good choice if degree of Parallelism is high.
2. Useful with small number of Processors.
3. Not all operation can be Pipelined.

→ Example: Select AVG(salary) FROM Employee GROUPBY Dept_Id;

→ need to group at least one department emp
→ Then only the o/p can be given for aggregate operation on the next Processor.

4. Cannot expect a full speedup.

→ Independent Parallelism



Independent

Consider a join four relations

$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$

Let P_1 be the assigned the computation of temp

$= r_1 \bowtie r_2$

Let P_2 be assigned the computation of $temp_2 = r_3 \bowtie r_4$

Let P_3 be assigned the computation of $temp_1 \bowtie temp_2$

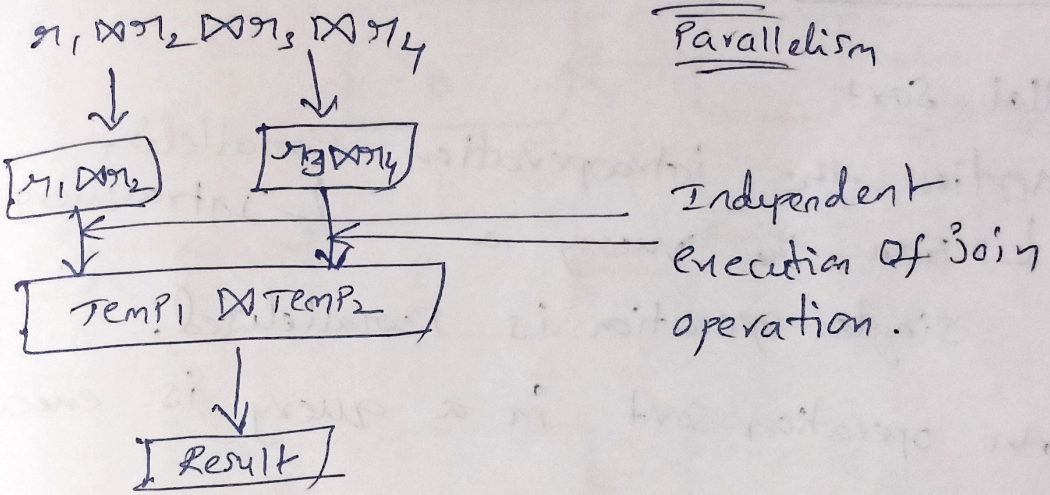
P_1 and P_2 can work independently in parallel

P_3 has to wait for i/p from P_1 & P_2

→ Can pipeline o/p of P_1 and P_2 to P_3. Combining independent parallelism and pipelined parallelism

→ Does not provide a high degree of parallelism.

Pipelined Parallelism

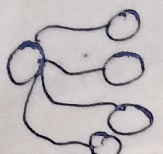


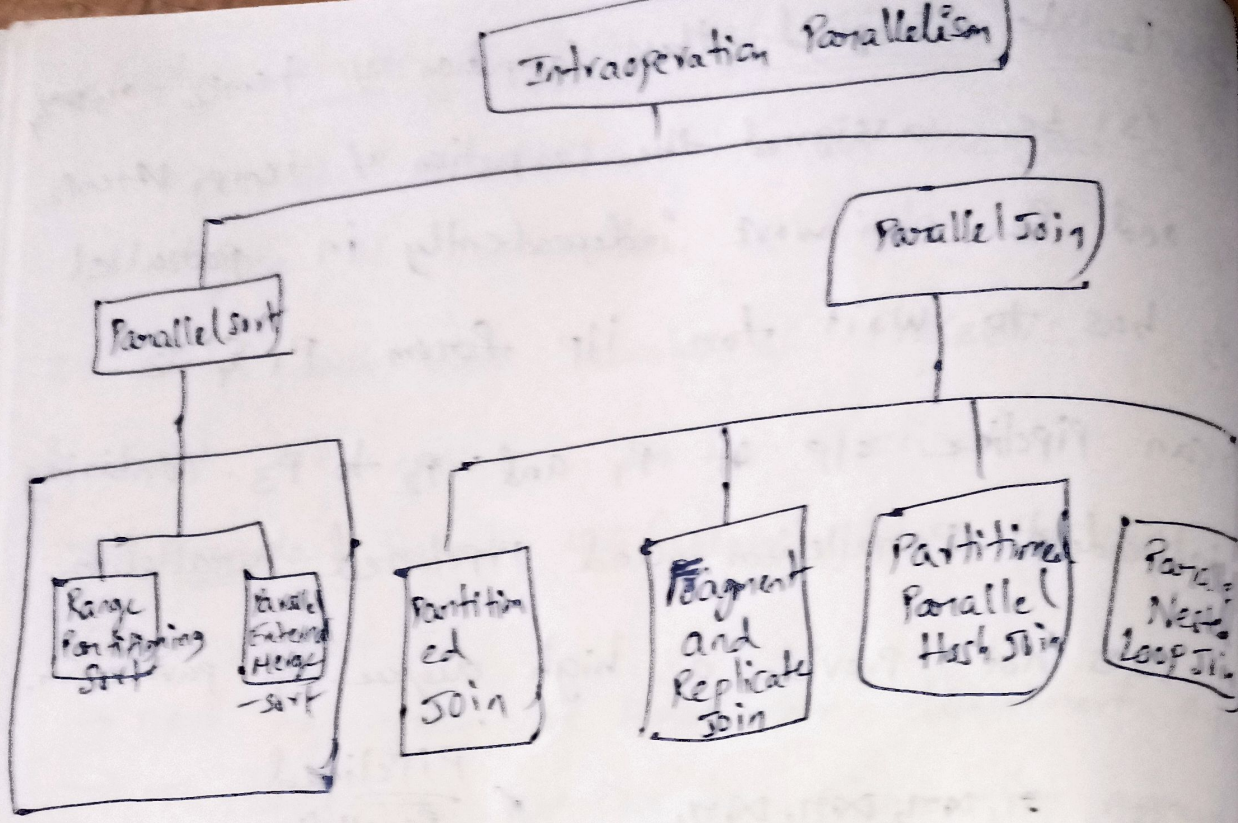
→ Intraoperation Parallelism

→ Each individual operation in a query is Parallelized.

→ Example: Parallel sort, Parallel search, Parallel Join.

→ A single operation Joining is Parallelized.





Parallel Sort

→ Applies the intraoperation Parallelism
└ Intraquery Parallelism
└ From the query a single operation is Parallelized.

→ An operation Sort in a query is executed in Parallel.

Parallel Sorting Techniques

1. Range Partitioning Sort
2. External Sort-Merge.

Assumptions:

→ Assume n Processors $P_0, P_1, P_2 \cdots P_{n-1}$ and disks n disks $D_0, D_1, D_2 \cdots D_{n-1}$

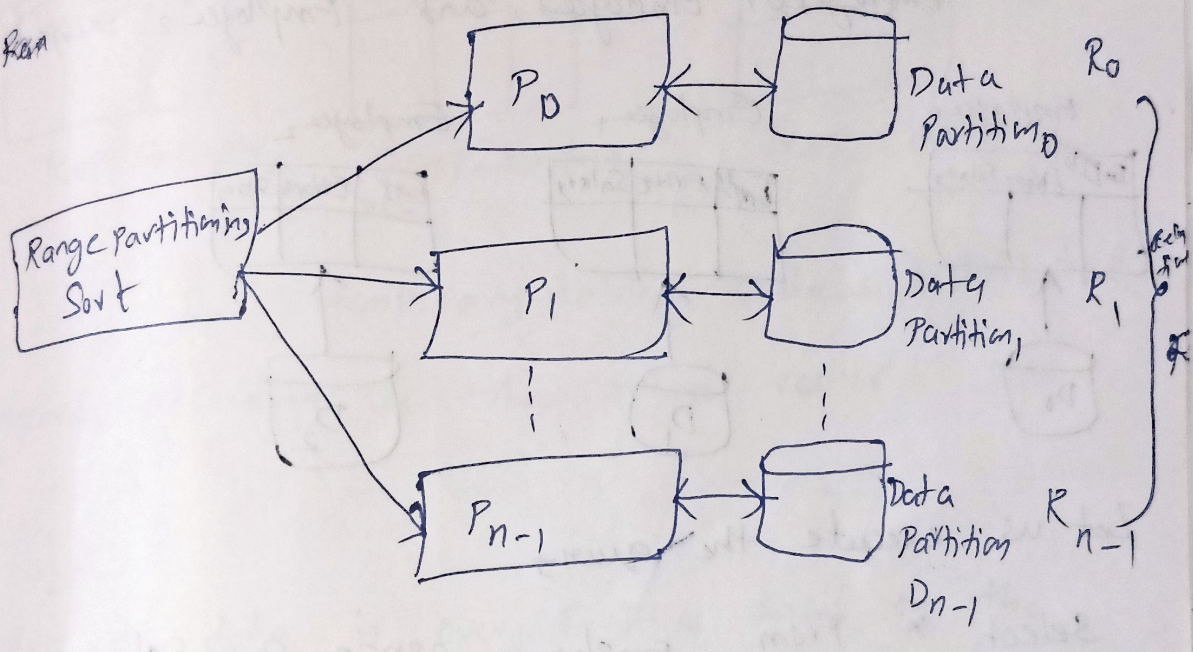
→ Disk D_i is associated with Processor P_i.

→ Relation R is Partitioned into $R_0, R_1, R_2 \cdots R_{n-1}$ using Round-Robin technique or Hash-Partitioning technique or Range Partitioning technique.

Objective

our objective is to sort a relation (table) R that resides on n disks on an attribute A in Parallel

Ex: Sort on a table Student records by roll no (A) sort



STEPS:

STEP 1: Partition the Relation R on the sorting attribute A at every Processor using a range vector V.

Send the Partitioned records which fall in the ith range to Processor P_i. Where they are temporarily Sorted in D_i.

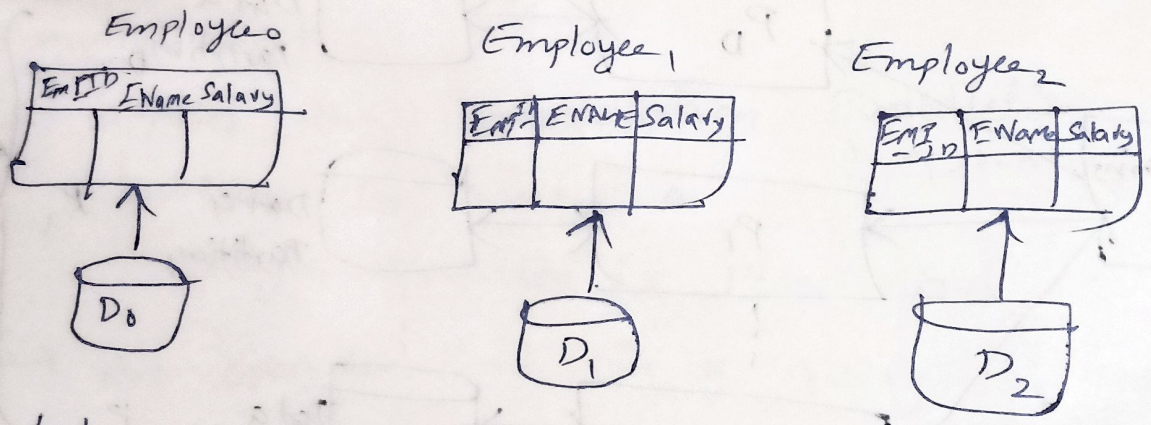
Step 2: Sort each Partition locally at each Processor P_i in Parallel Merge the Sorted results.

Ex

Employee (Emp_ID, EName, Salary)

→ Assume that relation Employee is Partitioned using round robin technique into 3 disks D_0, D_1, D_2 which are associated with Processors P_0, P_1, P_2.

→ At Processors P_0, P_1, P_2, the relations are named Employee0, Employee1, and Employee2 respectively.



Let us execute the query

Select * From Employee ORDER By Salary

→ Here the table is not Partitioned according to the salary attribute.

→ It is Partitioned by Emp_ID.

Step 1 (Range Partitioning) (Redistribution)

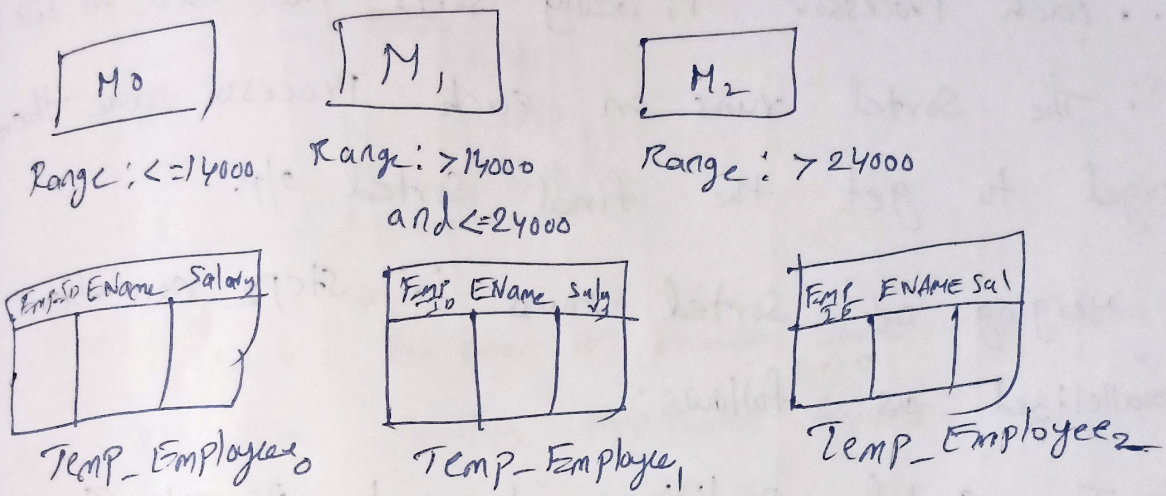
→ Identifying a range vector v on the Salary attribute as $v[v_0, v_1, \dots v_{n-2}]$

For our Example, the tuples are distributed in 3 ranges are

Employee 0 in $disk_0$ with Range 0 (14000 and less)

Employee 1 in $disk_1$ with Range 1 (14001 to 24000)

Employee 2 in $disk_2$ with Range 2 (24001 to more)



These temporary tables contains distributed records according to the range vector

Step 2:

→ Sort the data in every Partition based on the Salary attribute.

→ Every processor sorts its disk in Parallel on different data sets. It is called Data Parallelism.

Final Result:

→ After the processors completed the sorting, we can simply collect the data from different Processors & merge them.

Parallel Sort - Parallel External Sort - Merge

→ Assume the relation has already been Partitioned among disks $D_0, D_1 \cdots D_{n-1}$ (using any technique)

→ In Parallel External Sort-merge

- 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 o/p.

→ Merging of Sorted runs in step2 can be Parallelized as follows:

- The Sorted Partitions at each Processor P_i are range Partitioned accross the Processor $P_0 \cdots P_{m-1}$
- Each Processor P_i Performs a merge on the Stream as they are received, to get a single Sorted Partition.
- The Sorted runs on Processors $P_0 \cdots P_{m-1}$ are concatenated to get the final result.

Example:

→ Employe relation is Partitioned using round robin

→ Select * from Employe order by Salary;

→ Tables are not Partitioned on Sorting attribute Salary.

D_0

ID	Name	Dept name	Salary
6			10000
3			5000
9			6000
12			15000

D_1

ID	Name	Dept name	Salary
1			15500
4			35800
7			10500

1 mark
2

D_2

ID	Name	Dept name	Salary
2			25800
5			5000
8			7500

Step 1 Each Processor P_i locally sorts the data on disk D_i on sorting attribute salary

D_0

ID	Name	Dept name	Salary
3			5000
9			6000
6			10000
12			15000

D_1

ID	Name	Dept na	Salary
7			10500
1			15500
4			35800

P_0 P_1

ID	Name	Dept name	Salary
5			5000
8			7500
2			25800

steps:

→ Partition Vector on Salary attribute {14000, 25000}

→ Redistribute every Partition using range Vector on three disks.

P_0 Processor P0 merges incoming data at Partition D0

ID	Name	Dept name	Salary
3			5000
5			5000
9			6000
8			7500
6			10000
7			10500

P_1 >=14000 <25000

ID	Name	Dept name	Salary
12			15000
11			15500

Processor P_1 merges incoming data at Partition D_1

P_2

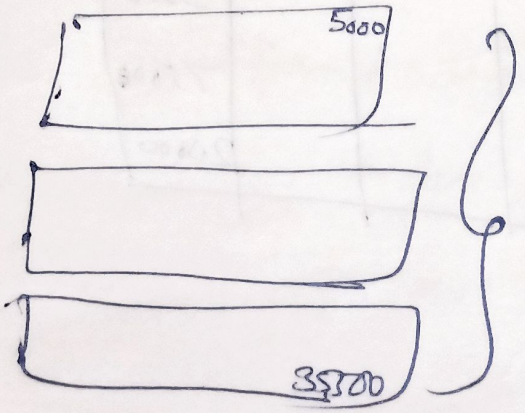
ID	Name	Dept name	Salary
2			25000
4			35000

Processor P_2 merges incoming data at Partition D_2

Final merge

concatenate of sorted data from all the disks

D_0 D_1 D_2



The data is sorted.

Parallel Join

→ The join operation requires pairs of tuples to be tested to see if they satisfy the join condition, and if they do, the pair is added to the join o/p.

→ Parallel join algorithms attempt to split the pairs to be tested over several processors. Each processor then computes part of the join locally.

→ In a final step, the result from each processor can be collected together to produce the final result.

→ Equi Join

Let r and s be the i/p relations and we want to compute $r \bowtie_{r.A = s.B} s$ and r and s each are partitioned into n partitions denoted $r_0, r_1, \ldots r_{n-1}$ and $s_0, s_1, \ldots s_{n-1}$.

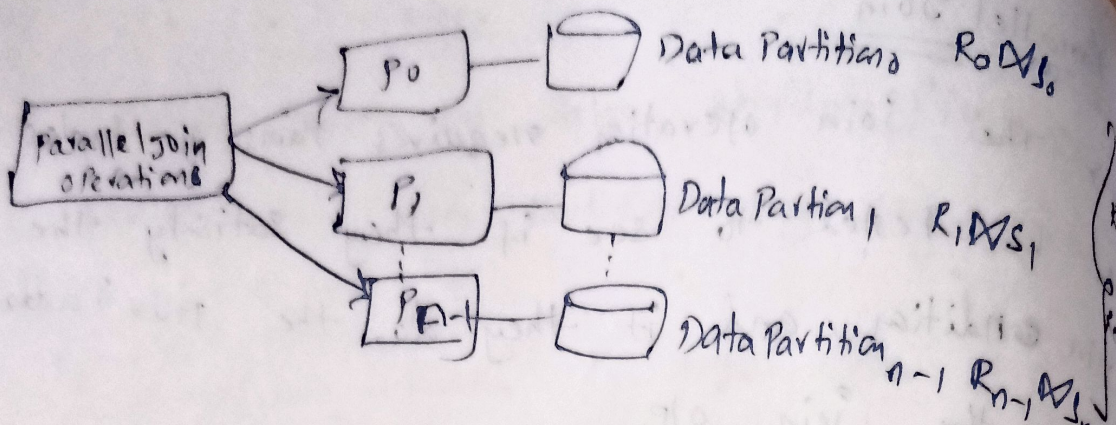
• Equi join is a special type of join in which we use only equality operator.

```
SELECT * FROM Emp INNER JOIN Dept
ON Emp.DEPTID = Dept.DEPTID;
```

• Natural Join

Natural join is a type of equi join which occurs implicitly by comparing all the names columns in both tables.

→ SELECT * FROM Emp NATURAL JOIN Dept



Example :- Partitioned Join

Student

RollNo	Name	Gender	Phone
1	Aman	M	- - -
3	Anand	M	- - -
4	Rupa	F	- - -
2	Ram	M	- - -

Course

RollNo	Course Name
4	AD
2	AI
3	DM
1	AI

Common Attribute
CoA : RollNo

range (or) Hash Partitioning

Relations: Student and Course

Joining attribute : Roll No

No. of disks = 2 hence No. of Partitions = 2

Partitioning technique : Hash Partitioning

P. A : RollNo

Hash Function : $h(\text{RollNo}) = (\text{RollNo} \bmod n) = (\text{RollNo} \bmod 2)$

Partition 0

RollNo	Name	Gen	Phone
2			
4			

Student 0

Partition 1

RollNo	Name	Gen	Phone
1			
3			

Student 1

1 mod 2 = 1
2 mod 2 = 0
3 mod 2 = 1
4 mod 2 = 0

~~Student~~ Course.

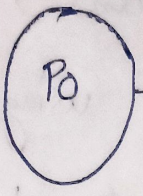
RollNo	Course Name
4	
2	

Course

RollNo	Course
3	
1	

Student

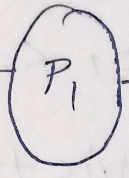
Roll	Name	Gen	Phone
4			
2			



Course

RollNo	Course Name
4	
2	

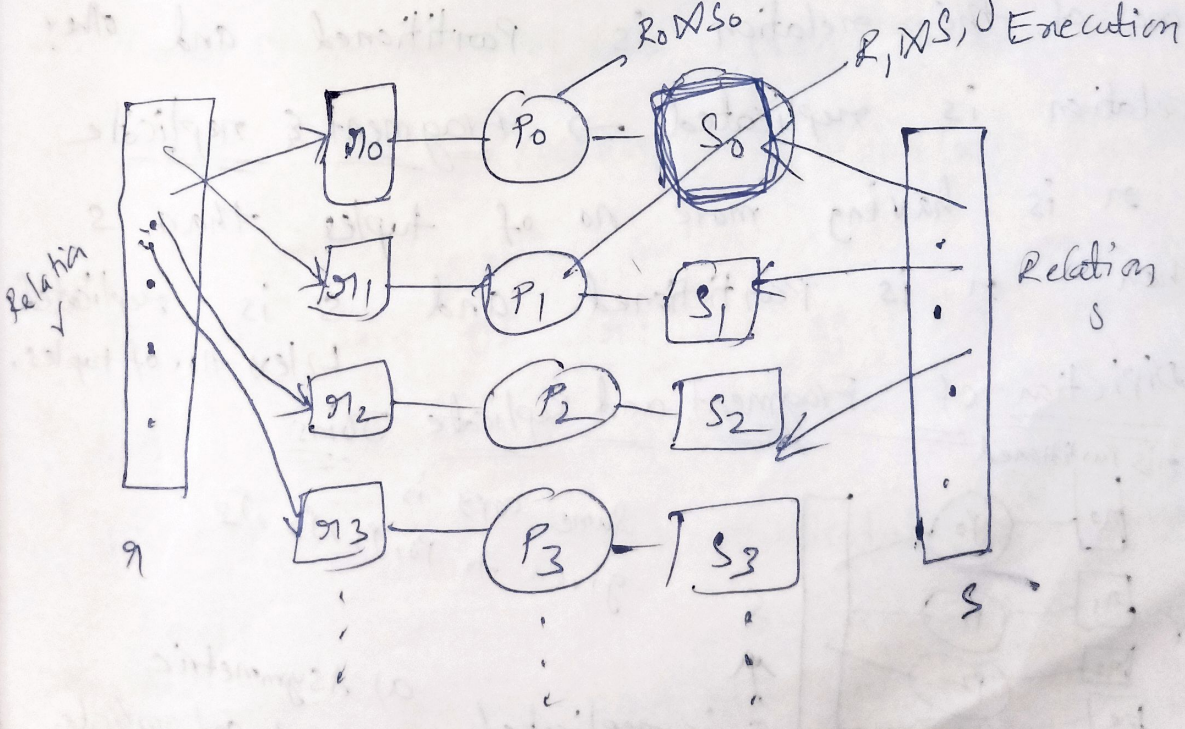
Roll
1
3



3
1

Result of Parallel Join operation: Student ⋈ Course

RollNo	Name	Gen	Phone	Course
4	Rupa	F	---	AD
2	Ram	M	----	AI
1	Aman	M	----	AI
3	Anand	M	----	DM



Fragment and Replicate Join

→ Partitioning not possible for some join conditions

eg: non-equijoin conditions, such as $r.A > s.B$

- condition is established using all comparison operators except the equal (=) operator.

like $!=, >=, <=, <, >$.

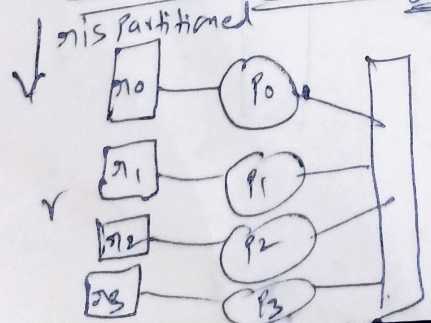
- Special case - Asymmetric fragment and replicate
  - one of the relations, say r, is partitioned
  - any partitioning technique can be used.
  - The other relation, s, is replicated across all the processors.
  - Processor P_i, then locally computes the join of r_i with all of s using any join technique.

→ one of the relation is partitioned and other relation is replicated → Fragment & replicate

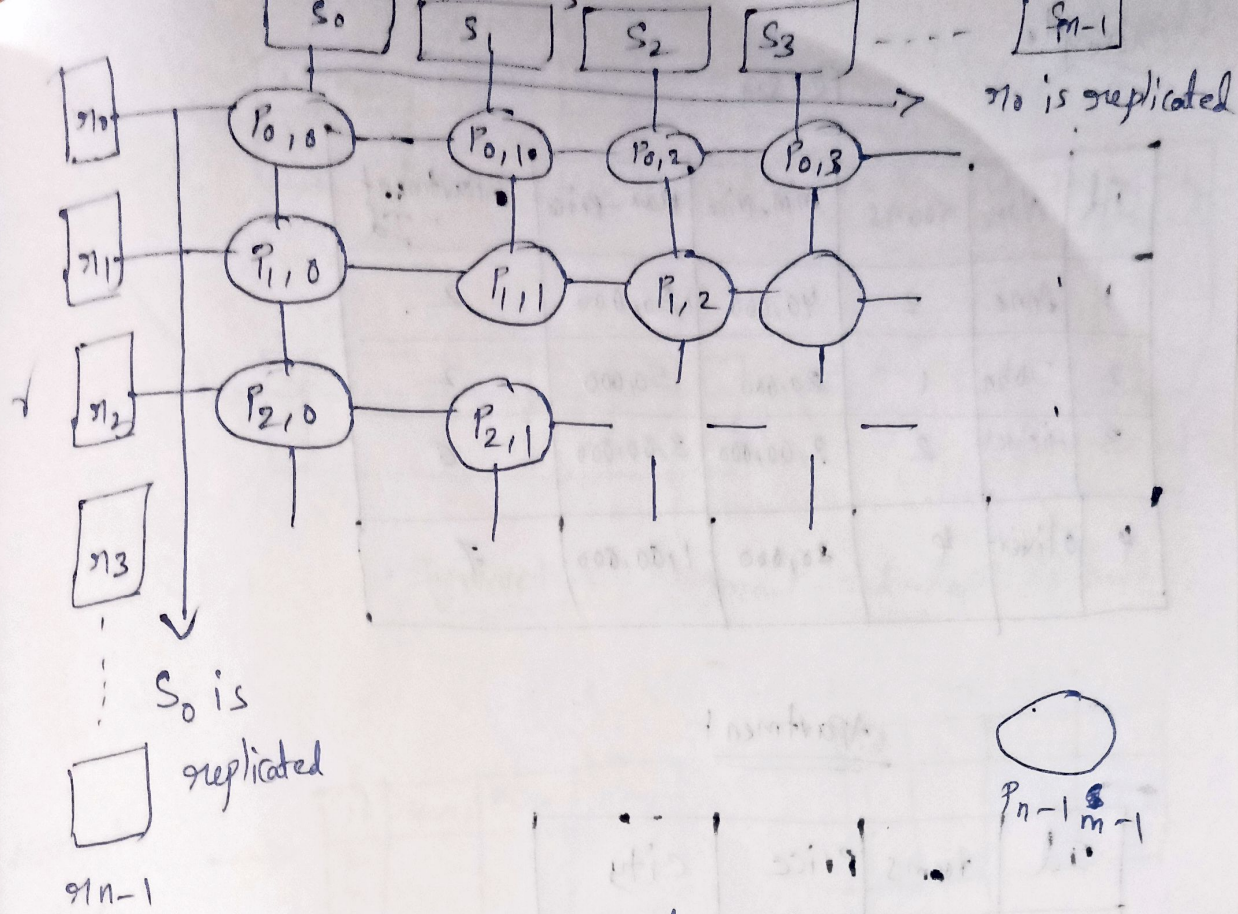
r is having more no of tuples than s then r is partitioned and s is replicated

Less no. of tuples

Diptiction of Fragment and Replicate Joins



a) Asymmetric fragment and replicate



b) Fragment and replicate.

→ r is Partitioned into n Partitions $r_0, r_1 \cdots r_{n-1}$

→ s is Partitioned into m Partitions $s_1, s_2 \cdots s_{m-1}$

→ Any Partitioning technique may be used.

→ There must be at least $m*n$ Processors

→ Label the Processors as $P_{00}, P_{01} \cdots P_{0m-1}, P_{10}, P_{11} \cdots P_{n-1,m-1}$

→ P_{ij} the join of r_i with s_j. In order to do so, r_i is replicated to $P_{i0}, P_{i1} \cdots P_{im-1}$, while s_j is replicated to $P_{0j}, P_{1j} \cdots P_{n-1,j}$

→ Any join technique can be used at each Processor P_{ij}.

Example

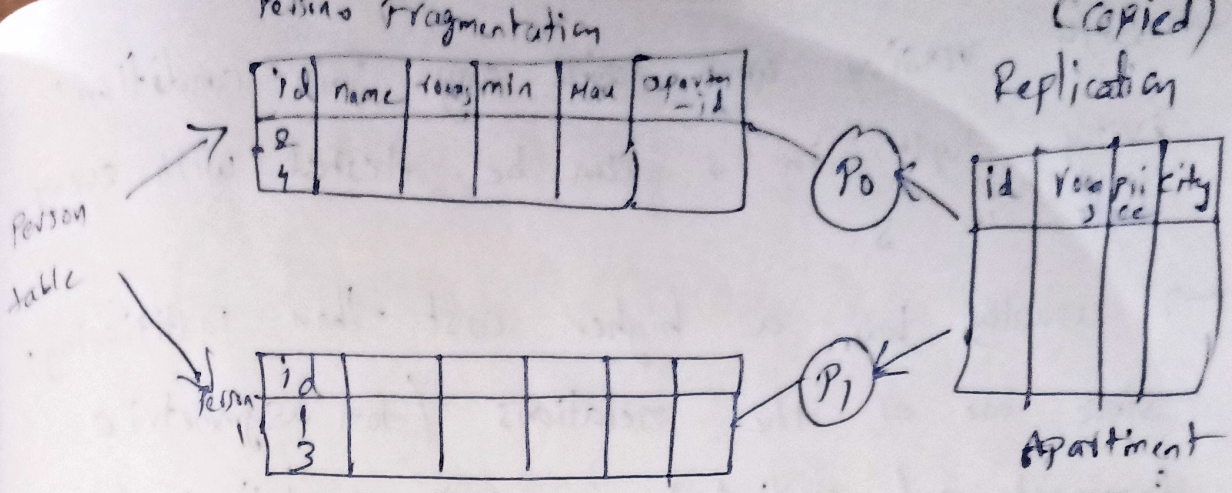
Person

Id	Name	rooms	min_Price	Max_Price	Apartment_Id
1	Anne	2	40,000	150,000	2
2	John	1	20,000	50,000	2
3	Michael	2	2,00,000	3,00,000	6
4	Oliver	4	85,000	1,00,000	7

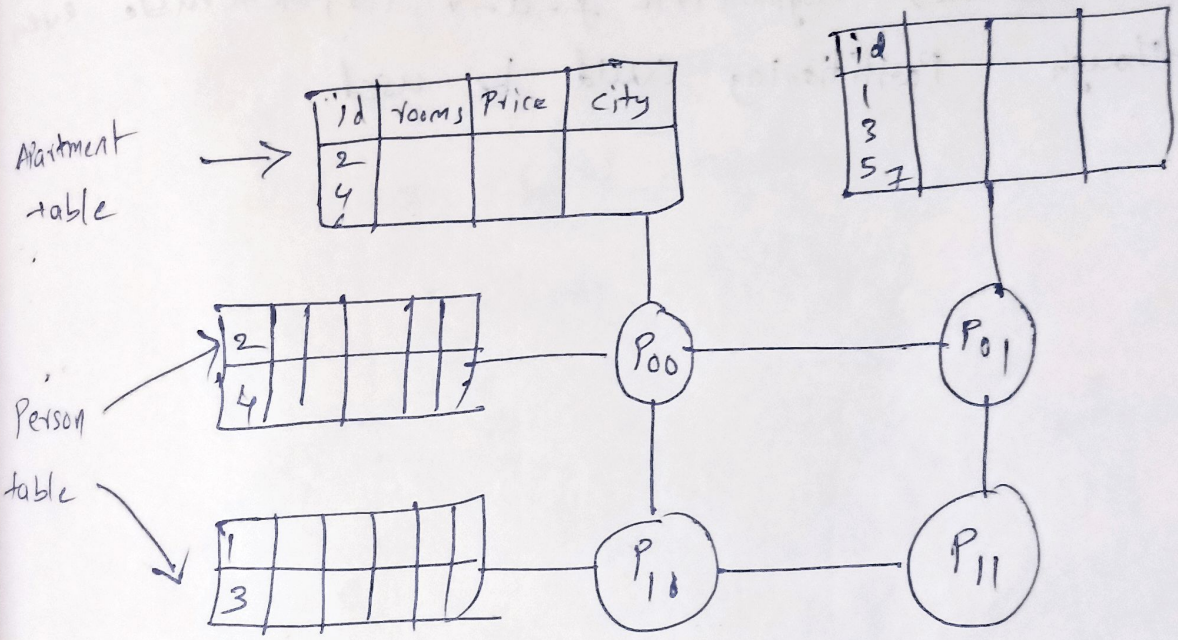
Apartment

Id	rooms	Price	city
1	2	30,000	Houston
2	2	45,000	Dallas
3	3	1,25,000	Chicago
4	5	2,45,000	Los Angleles
5	4	3,40,000	San Jose
6	4	2,20,000	San Diego
7	1	36,000	Cleveland

```
SELECT Name, min_price, max_price, price, city
FROM person JOIN apatment ON apartment.id != person.
                                              apatment_
AND price BETWEEN min_price AND max_price;
```



Symmetric Fragment-and-Replicate Join



Fragment and replicate join

Name	Min-Price	Max-Price	Price	City
John	20,000	50,000	30,000	Houston
John	20,000	50,000	36,000	cleveland
Anne	40,000	1,50,000	1,25,000	chicago
Michael	2,00,000	3,00,000	2,45,000	Los Angles
oliver	30,000	1,00,000	45,000	Dallal
oliver	30,000	1,00,000	30,000	Howston

→ Both version work with any join condition: every tuple in r can be tested with every tuple in s.

→ usually has a higher cost than Partitioning: since one of the relations (for asymmetric fragment-and-replicate) or both relations (for general fragment-and-replicate) have to be replicated

→ sometimes asymmetric f-a-r is preferable even though Partitioning could be used.

Parallel Join : Partitioned Parallel Hash Join

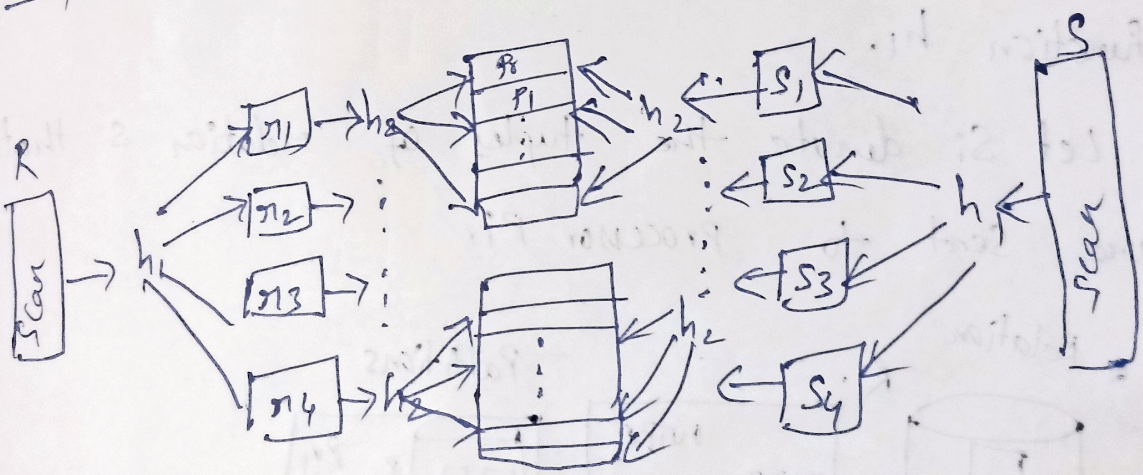
→ Assume that we have

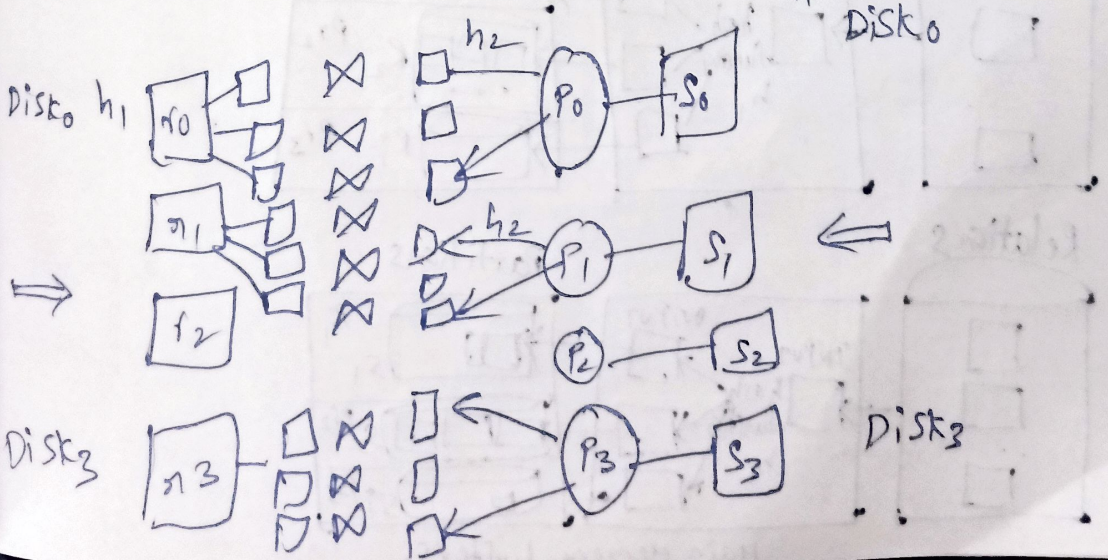
→ n processors $P_0, P_1 - - - P_{n-1}$

→ Two tables r and s for join operation (r and s are already partitioned into disks of n processors)

→ Assume s is smaller than r and therefore s is chosen as the build relation.

→ Hash functions : h_1 & h_2.





→ h_1 is used for first level Partitioning

→ h_2 is used for Second level Partitioning

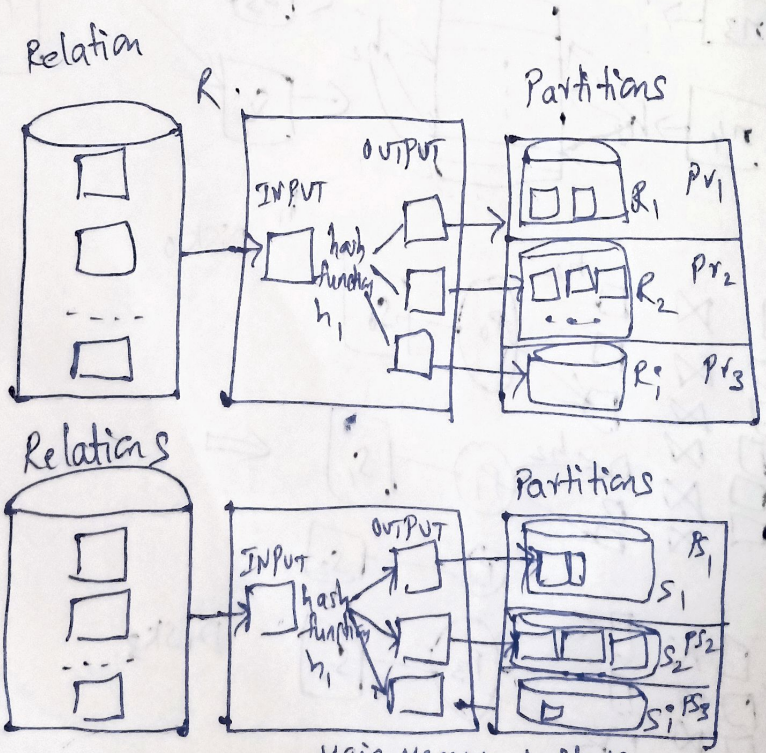
→ Later join operation takes place in P

Hash function h_1

- takes the join attribute value of each tup in s and maps the tuple to one of the n Proc

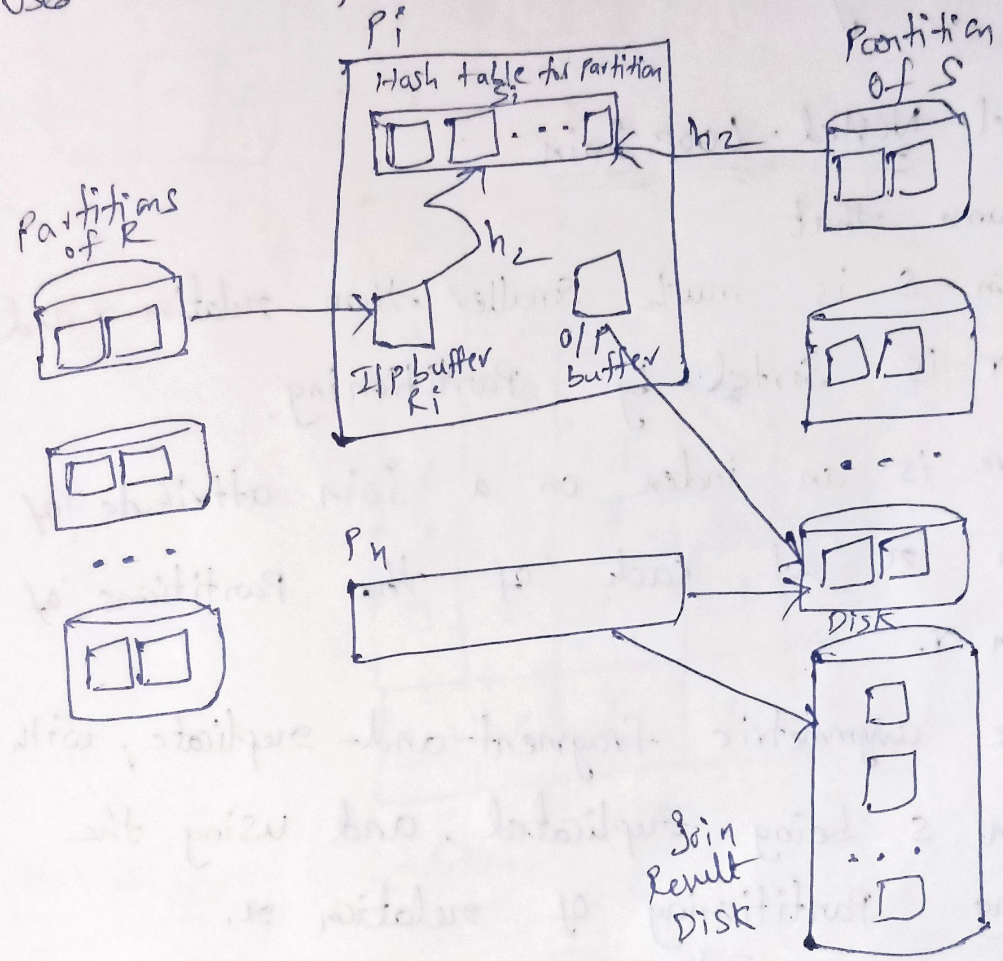
- Each Processor P_i reads the tuples of s t are on its disk D_i, and sends each tup to the appropriate Processor based on ha function h_1.

- Let S_i denote the tuples of relation s th are sent to processor P_i.



→ Hash function h_2.

— As tuples of relation s are received at the destination processors, they are partitioned further using another hash function h_2, which is used to compute the hash-join locally.



→ Each processor P_i executes the build and probe phases of the hash-join algorithm on the local partitions r_i and s_i of r and s to produce a partition of the final result of the hash-join.

→ Note: Hash-join optimizations can be applied to the parallel case.

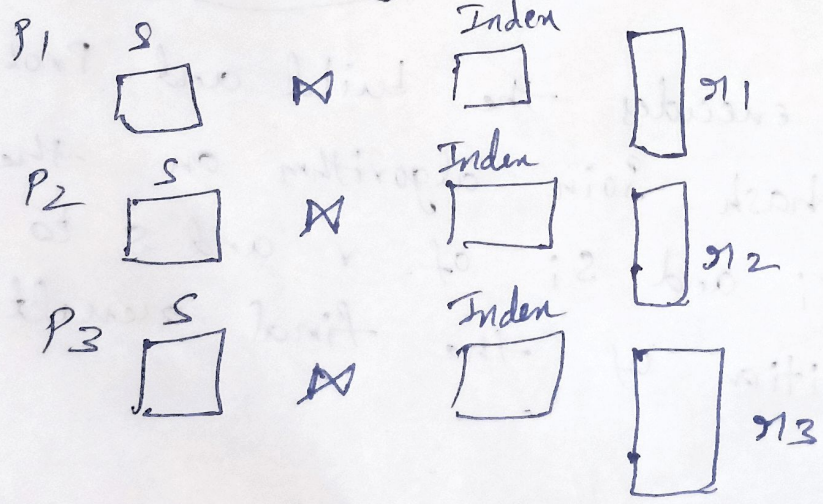
→ e.g. - the hybrid hash-join algorithm can be used to cache some of the incoming tuples in memory and avoid the cost of writing and reading them back in.

Parallel Nested-Loop Join

→ Assume that
- Relation s is much smaller than relation r and that r is stored by partitioning.

→ There is an index on a join attribute of relation r at each of the partitions of relation r.

→ use asymmetric fragment-and-replicate, with relation s being replicated, and using the existing partitioning of relation r.



Indexed nested loop join of s with the ith partition of relation r.

It is using Indexing

Joins:-

Cartesian product

P

A	B	C
1	a	b
2	c	d
3	e	f

Q

D	E
1	c
2	d

P × Q

A	B	C	D	E
1	a	b	1	c
1	a	b	2	d
2	c	d	1	c
2	c	d	2	d
3	e	f	1	c
3	e	f	2	d

Natural join : condition.

P

A	B	C
1	a	b
2	c	d

Q

A	D
1	d
3	e

Same name and datatype

P Natural Join Q

Attribute

A	B	C	D
1	a	b	d

Record of tuple

P

A	B
1	a
2	c

Q

C	D
1	b
3	d

P Left outer join Q
A=c

A	B	C	D
1	a	1	b
2	c	NULL	NULL

P Right outer join Q
A=c

A	B	C	D
1	a	1	b
NULL	NULL	3	d

Full outer join

A	B	C	D
1	a	1	b
2	c	NULL	NULL
NULL	NULL	3	d

INNER P INNER Join Q A=C :

Based on condition

A	B	C	D
1	a	1	b

SELECT * From P JOIN Q ON P.A=Q.C

→ Inter operation :-

SELECT * FROM Emp WHERE Salary > 5000
ORDER BY NAME;

SCAN Emp table for Salary > 5000

Sort all the Emp records on Name attribute

Two processors one to scan table, and the other to perform sorting.

Interoperation Parallelism

It is about executing different operations of a query in Parallel. A single query may involve multiple operations at once. We may exploit Parallelism to achieve better performance of such queries. Consider the example query given below

```
SELECT AVG(salary) FROM Employee GROUP BY Dept_Id;
```

(Dept_Id: attribute)

It involves two operations

First one is an aggregation and second is grouping.

For executing this query we need to group all the emp. records based on the attribute Dept_Id first.

Then, for every group we can apply the AVG aggregate function to get the final result.

We can use Interoperation Parallelism concept to Parallelize these two operations

1. Pipelined 2) Independent

In Pipelined Parallelism, the idea is to consume the result produced by one operation by the next operation in the Pipeline.

Example - Hash Partitioning

→ Assume that there are $n = 3$ disks, D_0, D_1, D_2 among which the relation is to be distributed and Partitioning attribute is ID

→ The hash function chosen is $H(x) = x \bmod n$ where x is value of a P.A ID

if $x = 1$, the tuple is sent to $H(1) = 1 \bmod 3 = 1$, that is D_1

if $x = 2$, the tuple is sent to $H(2) = 2 \bmod 3 = 2$ that is D_2

if $x = 3$, $H(3) = 3 \bmod 3 = 0$, that is D_0

if $x = 10$, $H(10) = 10 \bmod 3 = 1$, that is D_1

$x = 11$, $H(11) = 11 \bmod 3 = 2$, that is D_2

$x = 12$, $H(12) = 12 \bmod 3 = 0$, that is D_0

ID	Name	Dept_name	Salary
1	Srinivas	CSE	40000
2	Katz	ENTC	3500
3	Crick	ENTC	49000
⋮			
10	Singh	CSE	75000
11	Brooks	MECH	60000
12	Tanaka	ENTC	65000

	ID
D_0	3
	12
D_1	1
	10
D_2	2
	11

Range: Assume that there are $n = 5$ disks D_0, D_1, D_2, D_3

D_4 P.A = ID

Partition vector is chosen as [20, 40, 60, 80]

ID	Name	Dept_name	Salary
1			
2			
3			
⋮			
98			
99			
100			

	ID
D_0	1
	19
D_1	20
	39
D_2	40
	59

It chooses a P.A A

It chooses a P.V $[v_0, v_1 \cdots v_{n-2}]$ such that if $i < j$ then $v_i < v_j$

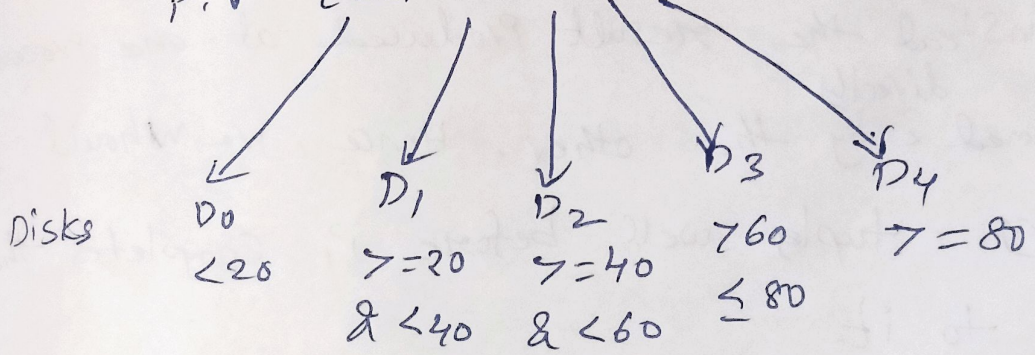
consider a tuple t such that $t[A] = x$

if $x < v_0$, then t goes to disk 0

if $x \geq v_{n-2}$, then t goes to disk $n-1$

if $v_i \leq x < v_{i+1}$, then goes to on disk D_{i+1}

P.V [20, 40, 60, 80]



For Example, consider the following operation

$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$

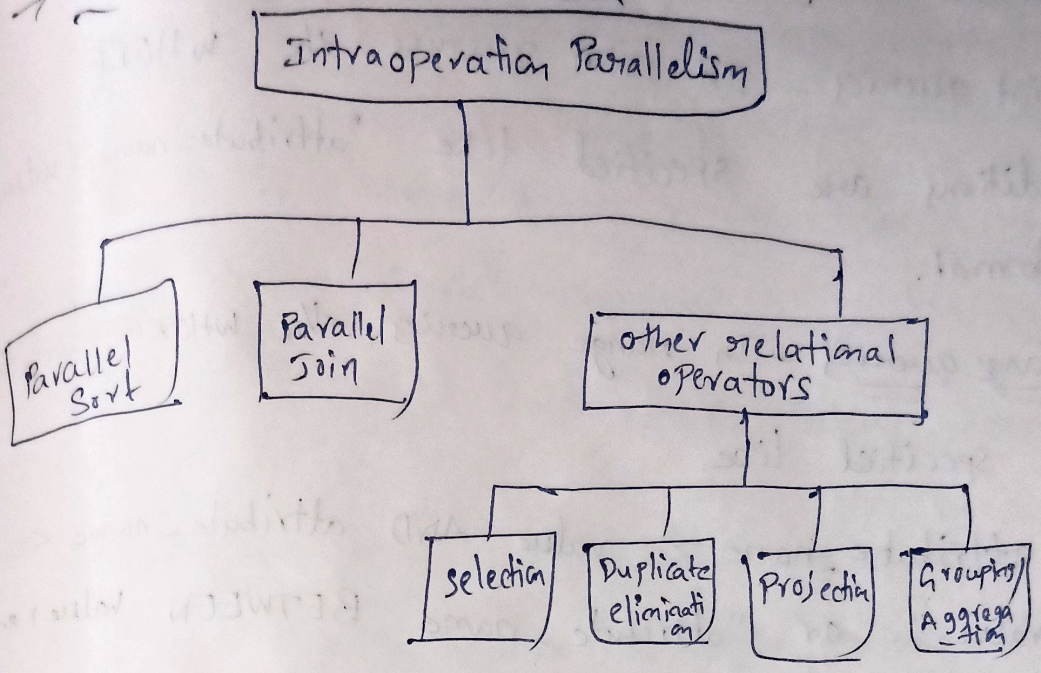
The above expression shows natural join operation. This actually joins four tables. This operation can Pipelined as follows

Perform $temp_1 \leftarrow r_1 \bowtie r_2$ at processor P_1 and send result $temp_1$ to processor P_2 to perform $temp_2 \leftarrow temp_1 \bowtie r_3$ and send the result $temp_2$ to Processor P_3 to perform result $\leftarrow temp_2 \bowtie r_4$.

The advantage is, we do not need to store the intermediate results, and instead the result produced at one processor can be consumed directly by the other. Hence, we should start receiving tuples well before P_1 completes the join assigned to it.

Join: Join statement is mainly used to combine two tables based on a specified common field b/w them.

→ other relational operators



→ Relation : Employee

→ Assume that table is Partitioned and stored in disk $D_0, D_1 \cdots D_{n-1}$ with Processors $P_0, P_1, \cdots P_{n-1}$.

→ Employee is Partitioned using Range Partitioning technique.

EmpID	EName	Salary
E101		27000
E102		22500
E103		80000
E106		270000
E113		20000
E111		31500
E109		26500
E110		22500
E212		40500

D_0

EmpID	EName	Salary
E112		22500
E113		20000
E110		22500

D_1

EmpID	EName	Salary
E101		27000
E106		270000
E109		26500

D_2

EmpID	EName	Salary
E103		80000
E111		31500
E112		40500

Recall the 3 types of queries based on data access

- Point queries - In point queries, the WHERE conditions are specified like "attribute_name = Value" format.
- Range queries - In range queries, the WHERE conditions are specified like "attribute_name >= value AND attribute_name <= value". or "attribute_name BETWEEN Value1 and Value2."
- Scanning the entire relation - In queries where the WHERE condition involves non-key attributes, then the system has to look for all the data in the specified table. This is called the relation scan.

Parallelizing the selection operation

Consider the following general syntax of any SQL Select query,

SELECT list_of_attributes FROM table_name WHERE condition O;

This query can be parallelized based on the condition O given in the WHERE clause.

Selection $\sigma_\theta(r)$

→ case 1: If θ is of the form $a_i = v$, where a_i is an attribute and v is a value.

→ If r is Partitioned on a_i the Selection is Performed at a single Processor.

SELECT * FROM emp WHERE Salary = 20000

→ case 2: If θ is of the form $l <= a_i <= u$ (i.e, is a range selection) and the relation has been range-Partitioned on a_i

- Selection is Performed at each Processor whose Partition overlaps with the specified range of values.

SELECT * FROM emp WHERE Salary BETWEEN 25000 and 30000.

→ case 3: other cases: the selection is Performed in Parallel at all the Processors.

SELECT * FROM emp WHERE Ename = 'Sawyer"

→ Duplicate elimination

→ Perform by using either of the Parallel sort techniques

- Eliminate duplicates as soon as they are found during sorting.

→ can also Partition the tuples (using either range or hash Partitioning) and Perform duplicate elimination locally at each Processor.

Projection:-

→ Projection without duplicate elimination can be Performed as tuples are read in from disk in Parallel.

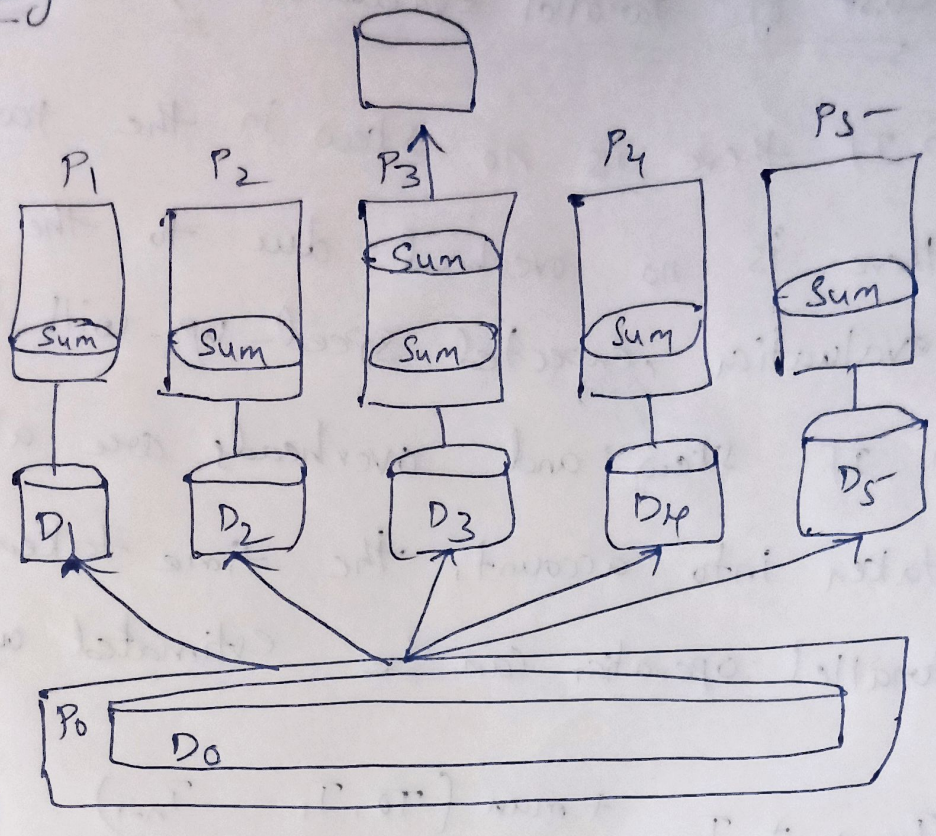
→ If duplicate elimination is required, any of the above duplicate elimination techniques can be used.

→ Grouping / Aggregation

· Partition the relation on the grouping attributes and then compute the aggregate values locally at each processor.

→ can reduce cost of transferring tuples during Partitioning by Partly computing aggregate values before Partitioning.

Partly computing



Partly computing

Grouping partitioning

→ Consider the sum aggregation operation:

→ Perform aggregation operation at each Processor P_i on those tuples stored on disk D_i.

→ results in tuples with partial sums at each Processor.

→ Results of the local aggregation is partitioned on the grouping attributes.

→ Aggregation performed at each Processor P_i to get the final result.

→ Fewer tuples need to be sent to other Processors during partitioning.

Cost of Parallel evaluation of operations

→ If there is no skew in the partitioning and there is no overhead due to the parallel evaluation, expected speed-up will be 1/n.

→ If skew and overheads are also to be taken into account, the time taken by a parallel operation can be estimated as

$$T_{part} + T_{asm} + \max(T_0, T_1, \ldots T_{n-1})$$

- T_{part} is the time for partitioning the relations
- T_{asm} is the time for assembling the results
- T_i is the time taken for the operation at processor P_i
  → this needs to be estimated taking into account the skew and the time wasted on contentions.