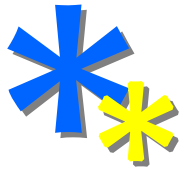
The Best Reliable Partner for High Availability



2008 상반기 효과적인 시스템 관리를 위한 기술 세미나 - COEX, Seoul, Korea

DB2 UDB DPF Overview and Performance

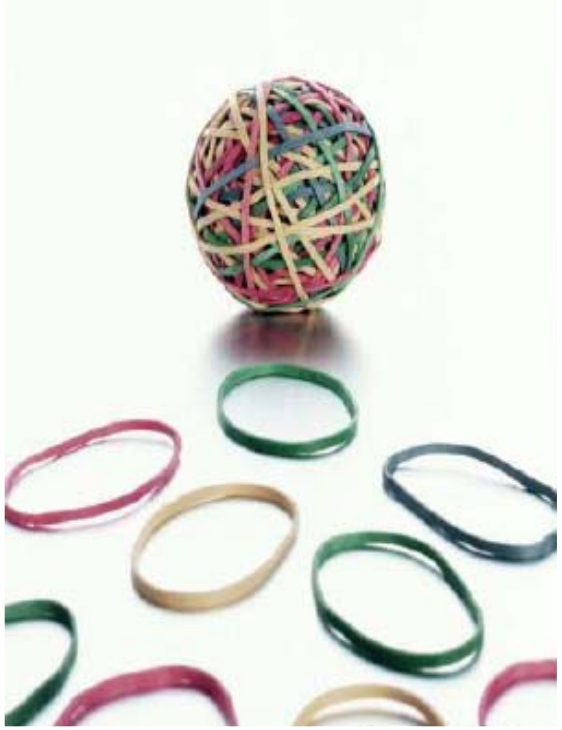
이상근 차장(sgelee@kr.ibm.com)

Senior IT Specialist

MTS, IBM Global Technology Services Korea

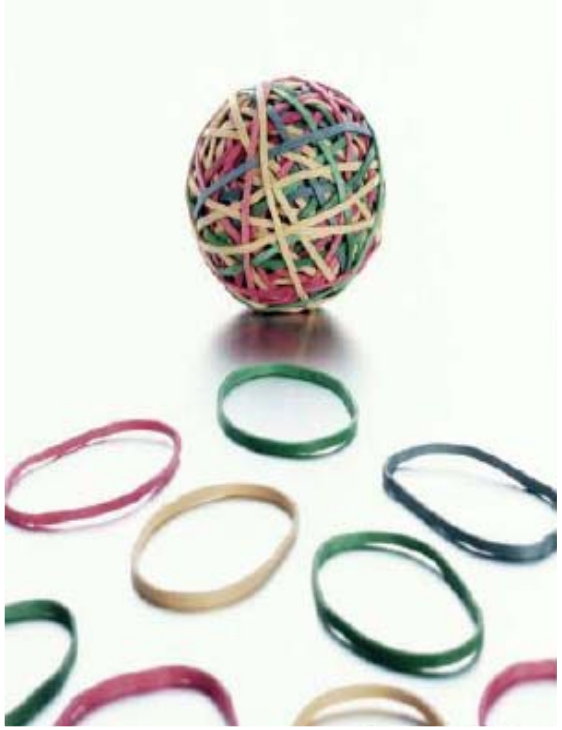
April 8, 2008

Agenda



1. Architecture and Concepts
2. Installation and Create Objects
3. Join Strategies and Explain
4. Performance Considerations

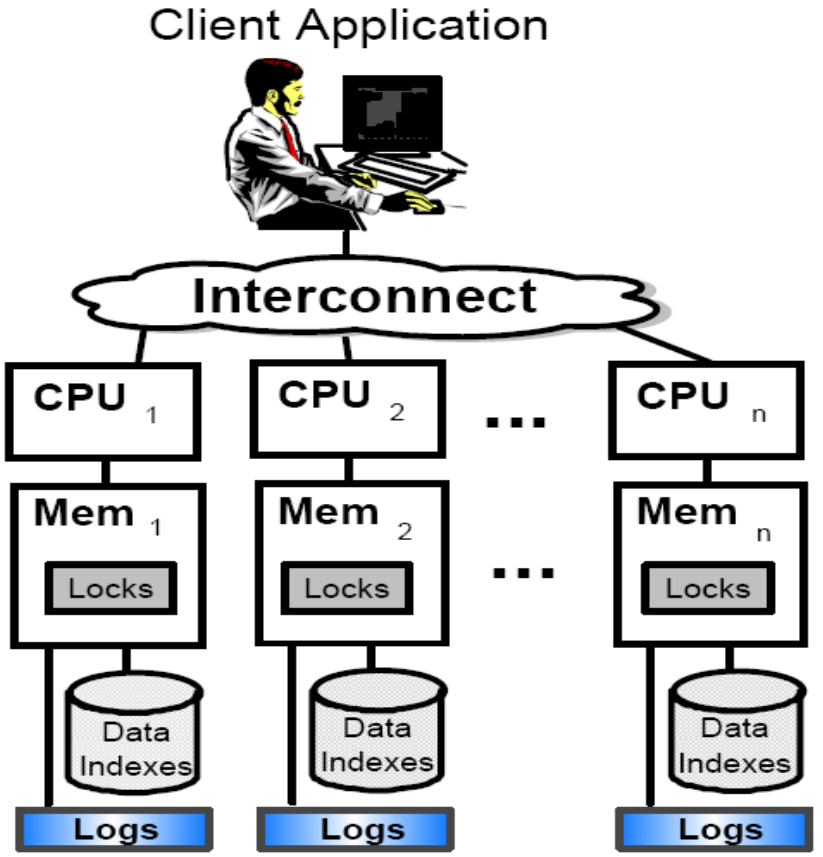
Agenda



1. Architecture and Concepts
2. Installation and Create Objects
3. Join Strategies and Explain
4. Performance Considerations

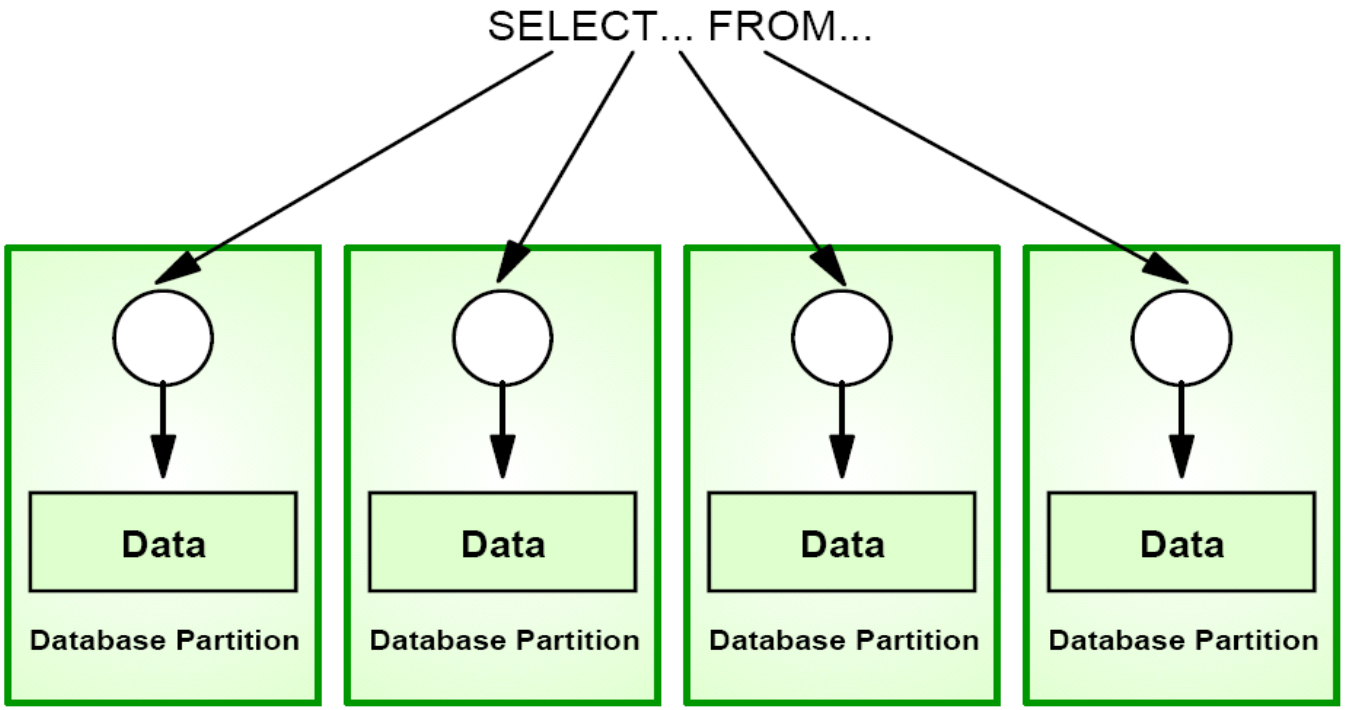
DB2 UDB DPF – Shared Nothing Architecture

- Partitioned Database Model
  - Divided into multiple partitions
  - Single system image to user and application
- Independent Partitions
  - CPU, Memory
  - Data, Index, Log
  - Locking
- Parallel Features
  - Parallel Optimization
  - Parallel Processing on all partitions
- Flexible Configurations
  - Add new partitions
  - Changes Conf. parameters independently



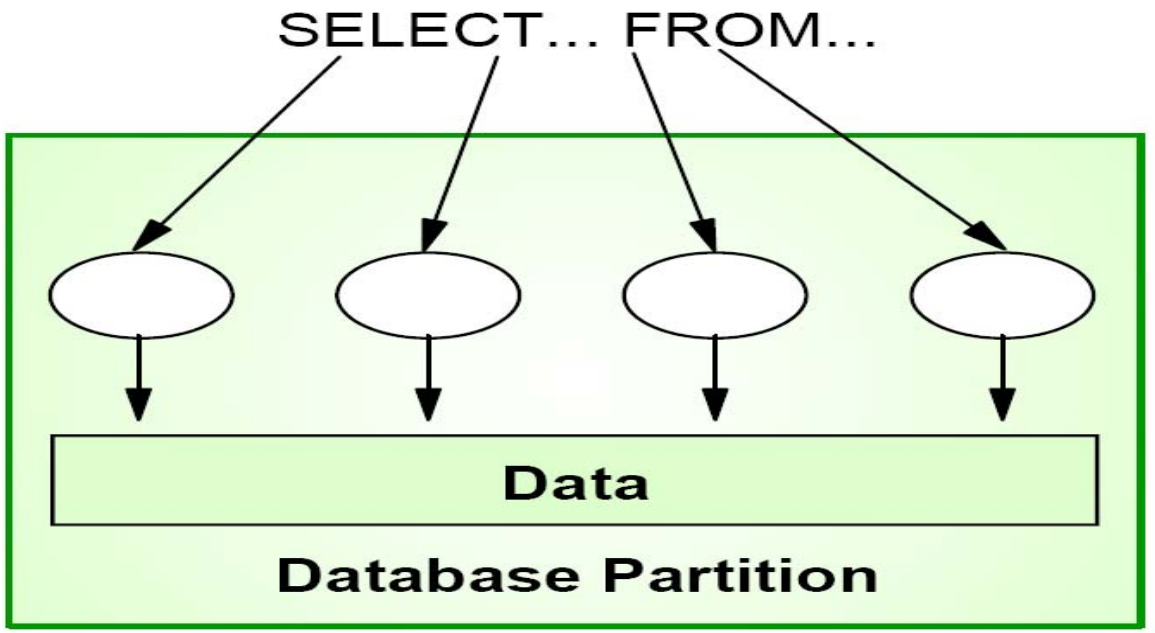
Inter-Partition Parallelism

- Break up a query into multiple parts across multiple partitions
- Send the query to each database partition used by the table
- A single query is performed in parallel
- The benefit is speed-up of processor time



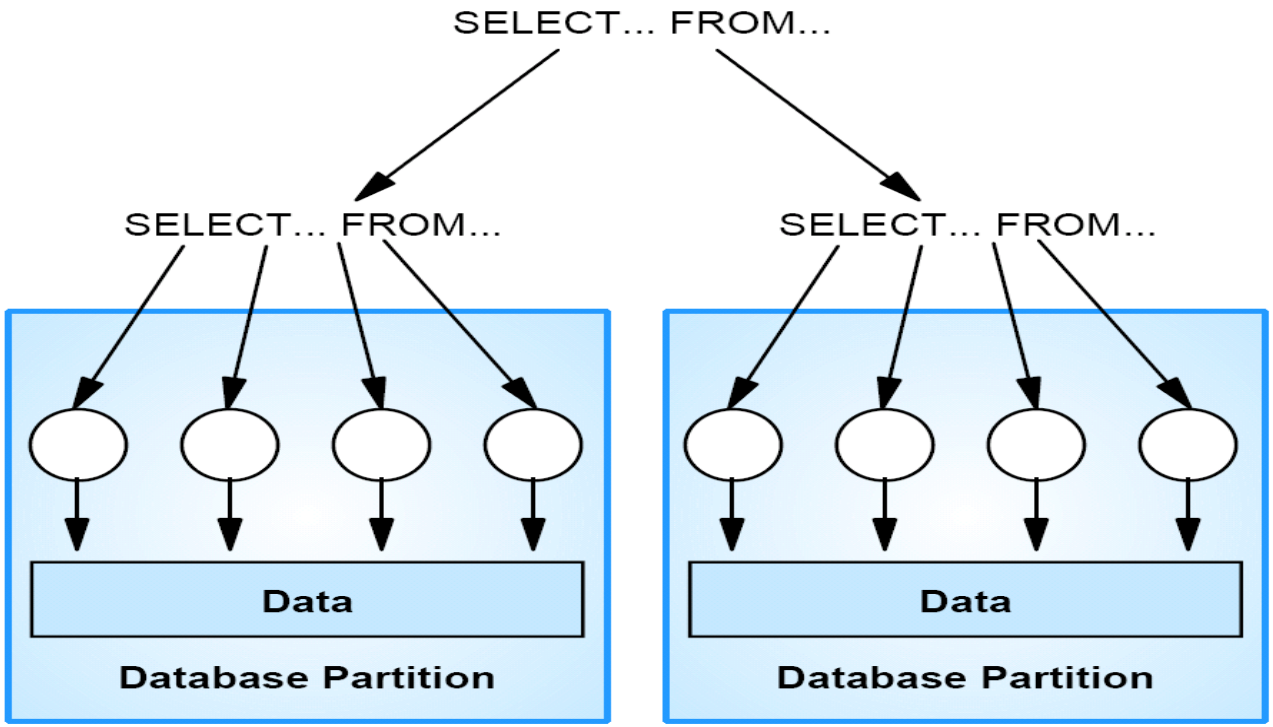
Intra-Partition Parallelism

- Break up a query into multiple parts in a partition
- Broken into multiple pieces that can be executed in parallel
- Define DBM CFG - INTRA_PARALLEL=YES, DB CFG - DFT_DEGREE >= 1



Inter-Partition and Intra-Partition Parallelism

- Can use intra-partition parallelism and inter-partition parallelism at the same time.
- This can result in an even more dramatic increase in the speed at which queries are processed.



Intra-Partition Parallelism Recommendations

- Enabled by DBM CFG - INTRA_PARALLEL, DB CFG – DFT_DEGREE
- Enable on DSS and Disable on OLTP
- For mixed OLTP/DSS

  - set INTRA_PARALLEL=YES, but set DB CFG Parameter DFT_DEGREE=1
  - For DSS Query, SET CURRENT DEGREE can be used to set degree of parallelism greater than one
  - If number of DSS users is greater than two times number of CPUs, leave INTRA_PARALLEL=NO
- Parallelism load

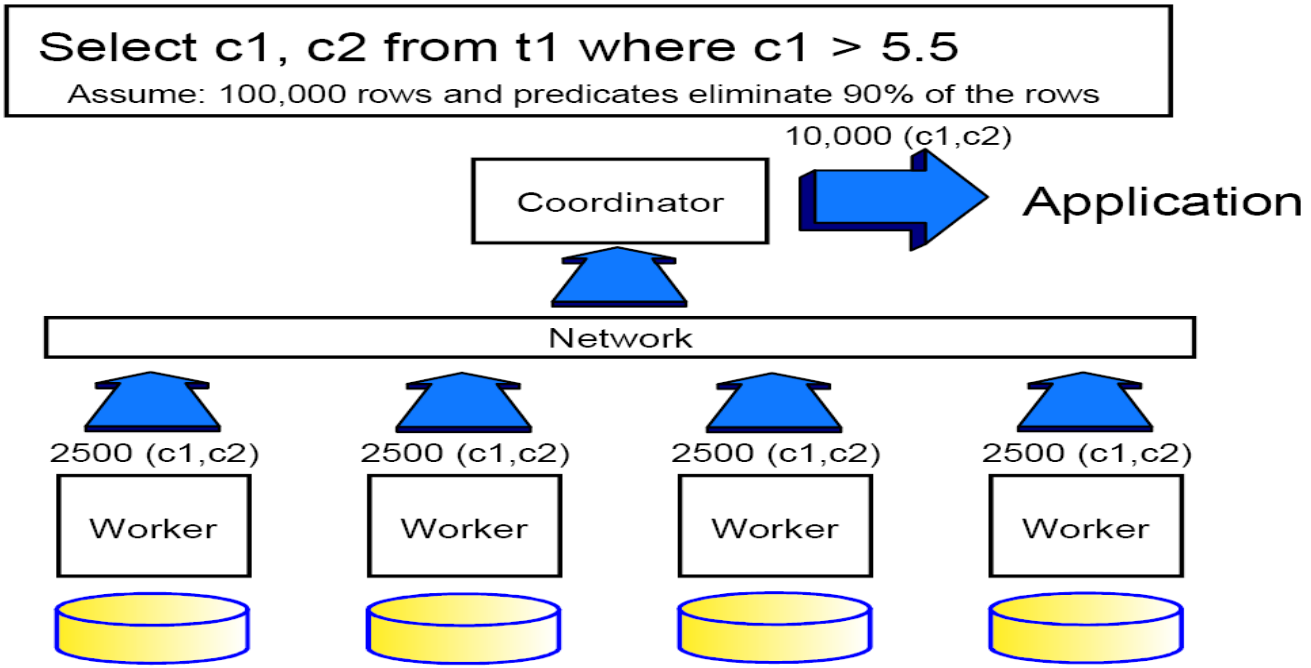
  - The number of active users multiplied by the degree of parallelism

    Ex) 50 active users * 4 parallelism = 200 parallelism load

    (200 processes in the run queue)
  - A parallelism load of between 1.5 and 2.0 times the number of available CPUs

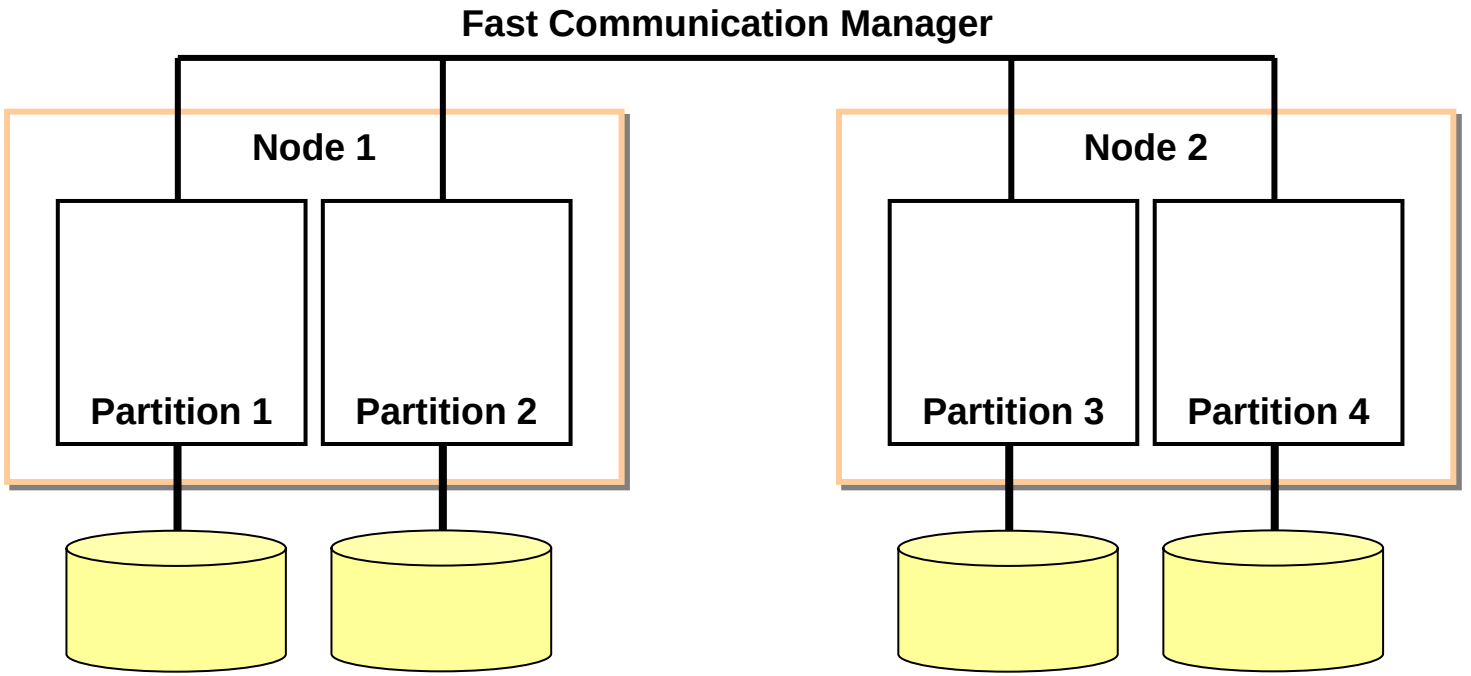
Maximize Parallelism – Function Shipping

- The select statement is shipped to each of the worker processes.
- The predicates are applied to reduce the number of rows.
- The number of columns are also reduced.
- Performance gains by reduction of network traffic.



DB2 UDB DPF on SMP Environments

- Four partition database is running on an SMP containing any number of processors.
- The example shows four logical database partitions on two nodes.
- Each database partition has its dedicated resources.
- The Shared Nothing architecture still being used in SMP.

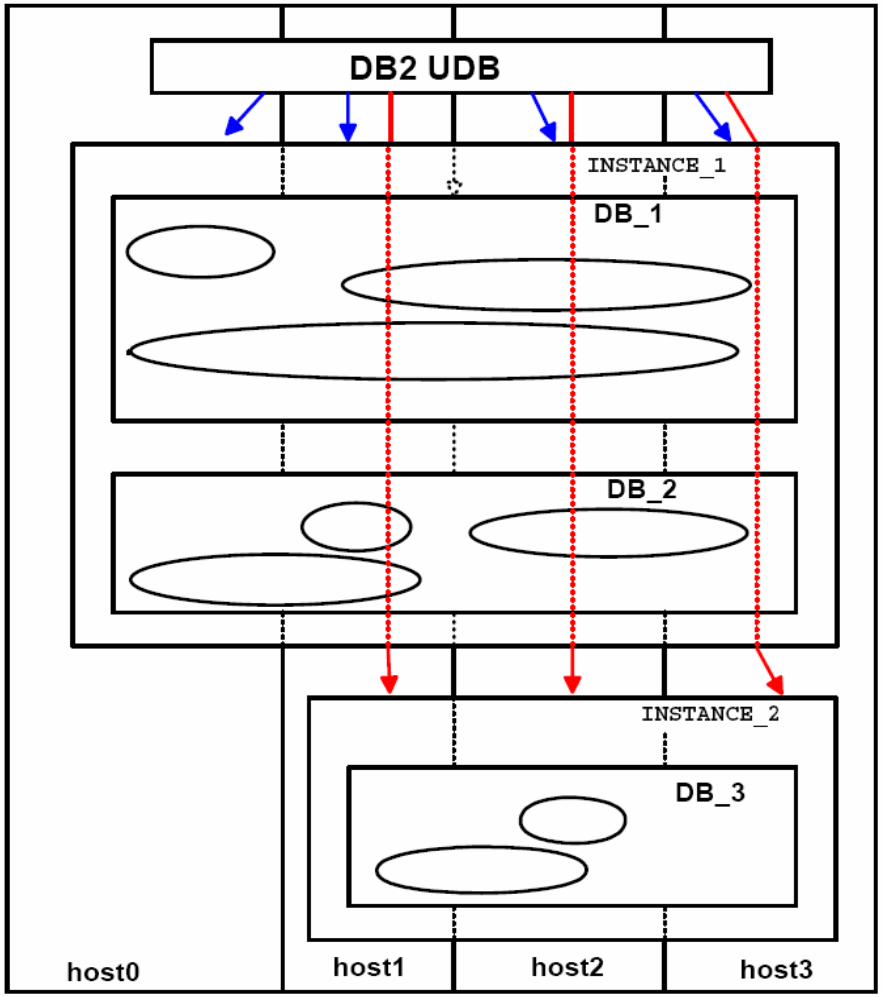


DB2 UDB DPF Instance & Database

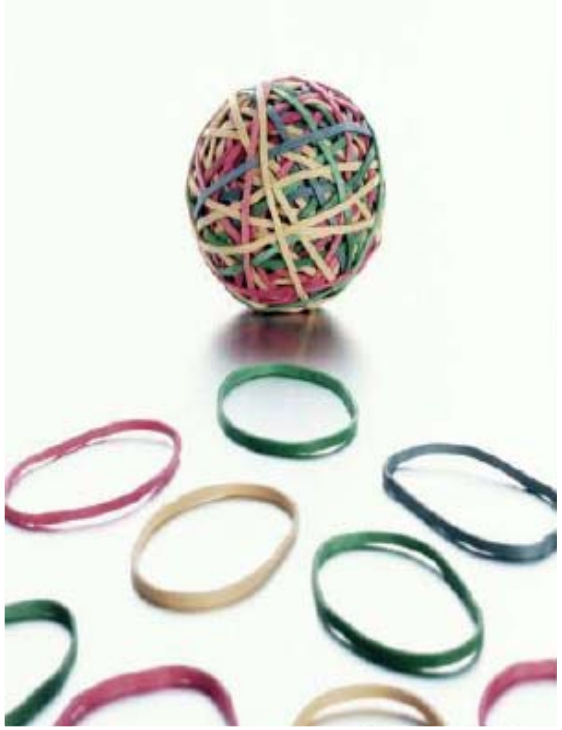
- It is possible to have multiple DB2 UDB DPF instance on same group of parallel nodes.
- There are several reasons

  - To maintain distinct test and production environments

  - To use different software releases
- Each instance can manage multiple databases.



Agenda



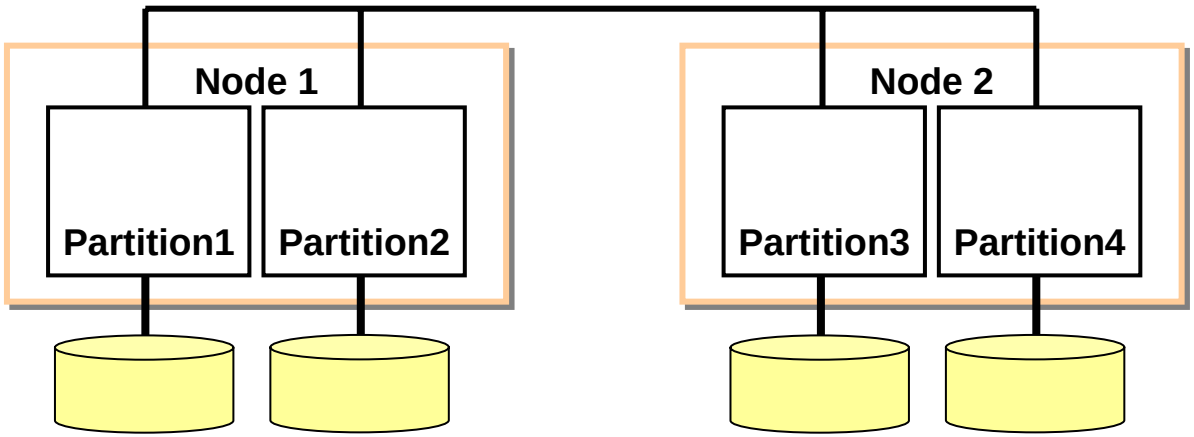
1. Architecture and Concepts
2. Installation and Create Objects
3. Join Strategies and Explain
4. Performance Considerations

DB2 UDB DPF Setup Checklist

- Install DB2 DPF Feature Code
- Export home directory of instance owner userid and mount it on each node
- Create groups (Instance owner, fenced, DAS)
- Create users (Instance owner, fenced, DAS) and set passwords
- Create Instance, DAS
- Update Node Configuration File (db2nodes.cfg)
- Reserve inter-communication ports to enable FCM (/etc/services)
- Modify Configuration for DB2 UDB DPF
- Enable execution of remote commands ($HOME/.rhosts)
- DB2 Start

Node Configuration File and /etc/services File

- This example shows the same four database partitions running on two physical machines
- /etc/services
  - Must be greater than or equal to the greatest number of database partitions on any physical partition in the instance
  - The name of the service of the first port must be ‘DB2_instance-name’
  - The name of the service of the last port must be ‘DB2_instance-name_END’
- $HOME/sqllib/db2nodes.cfg
  - partition : Must be unique and in ascending sequence
  - Hostname : The hostname of the ip address used by db2start and db2stop
  - Port : The logical port number for the database partition



DB2_inst1	**60000/tcp**
DB2_inst1_END	**60001/tcp**

/etc/services

partition	hostname	Port
1	**node1**	**0**
2	**node1**	**1**
3	**node2**	**0**
4	**node2**	**1**

$HOME/sqllib/db2nodes.cfg

.rhosts File

- DB2 DPF uses the rsh command to execute some commands such as db2start
- This can be done by updating the .rhosts file in the instances home directory
- The hostname must be defined in the /etc/hosts file

hostname	Userid
node1	**db2inst1**
node2	**db2inst1**
node3	**db2inst1**
node4	**db2inst1**

$HOME/.rhosts

------ OR -----

hostname	Userid
+	**db2inst1**

$HOME/.rhosts

Database, Partition Group, Table Spaces and Tables, PK

- Databases are defined across database partitions specified at the instance level
  - $DBPATH/NODE0000/instname/SQL0001,
    $DBPATH/NODE0001/instname/SQL0001,…..
- Partition groups are defined across one or more database partitions
- Create buffer pool with partition group
- Table spaces are created in partition groups
- Table spaces are a logical layer created within partition groups and provide a logical layer between database objects and physical storage
- Tables are created with Partitioning Key within table spaces

Create Options (1)

- Create Database

  - db2 create database sample on /database

    - create a database named sample on the path /database across all partitions in db2nodes.cfg

    - /database/instance-name/NODE0001/SQL00001

      /database/instance-name/NODE0002/SQL00001

      …….

    - catalog partition is the partition from which CREATE DATABASE was issued

- Create Partition Group

  - create database partition group PG1 on all dbpartitionnums ;
  - create database partition group PG2 on partitionnums (1,3,4) ;
  - create database partition group PG3 on partitionnums (2) ;

- Create Bufferpool

  - create bufferpool buff_1 database partition group PG1 size 10000 ;
  - create bufferpool buff_1 size 10000;

    alter bufferpool buff_1 add database partition group PG1 ;

Create Options (2)

- Create DMS Regular Tablespace

```
- CREATE REGULAR TABLESPACE TS1 IN DATABASE PARTITION GROUP PG1
  PAGESIZE 32K MANAGED BY DATABASE
              USING (DEVICE '/dev/container1' 917504,
                     DEVICE '/dev/container2' 917504) ON DBPARTITIONNUM(1)
              USING (DEVICE '/dev/container3' 917504) ON DBPARTITIONNUM(3)
              USING (DEVICE '/dev/container4' 917504) ON DBPARTITIONNUM(4)
  bufferpool buff_1 ;
```

- Create SMS Regular Tablespace

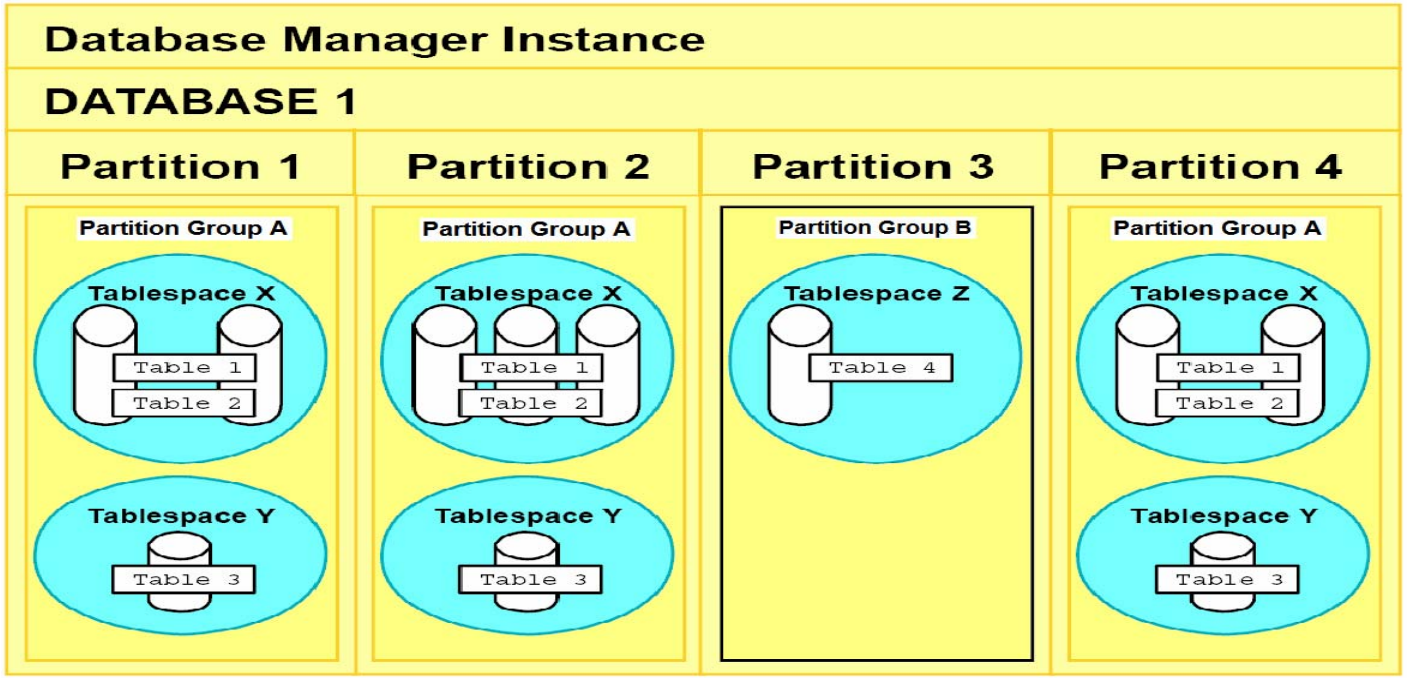
```
- CREATE REGULAR TABLESPACE TS2 IN DATABASE PARTITION GROUP PG2
  PAGESIZE 32K MANAGED BY SYSTEM
              USING ('/database/NODE0001/sms/cont1,
                     /database/NODE0001/sms/cont2') ON DBPARTITIONNUM(1)
              USING ('/database/NODE0002/sms/cont1,
                      /database/NODE0002/sms/cont2') ON DBPARTITIONNUM(2)
              USING ('/database/NODE0004/sms/cont1') ON DBPARTITIONNUM(4)
  bufferpool buff_2 ;
```

- Create Table

```
- CREATE TABLE T1  (C1 INTEGER, C2 CHAR(10), C3 VARCHAR(100))
  PARTITIONING KEY (C1) USING HASHING
  IN TBS1 INDEX IN TBS2 ;
```

Database Objects

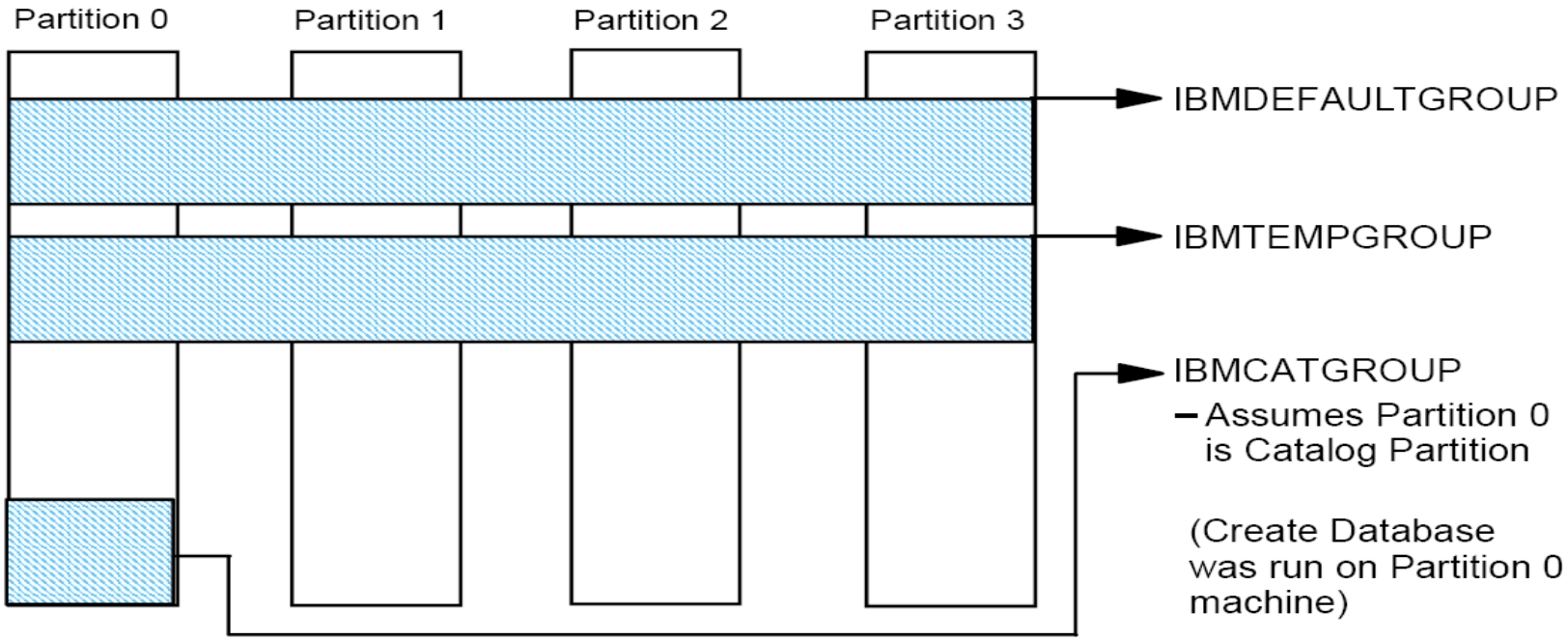
- Partition group A is defined across Partition1,2,4
- Partition group B is a single partition group on partition 3
- Tablespace X, Y is created in Partition group A, Tablespace Z is created in Partition group B
- Table 1 and Table 2 are created in Tablespace X and Table 3 is created in Tablespace Y
- Table 4 is created in Tablespace Z



Three Partition groups Created by Database Creation

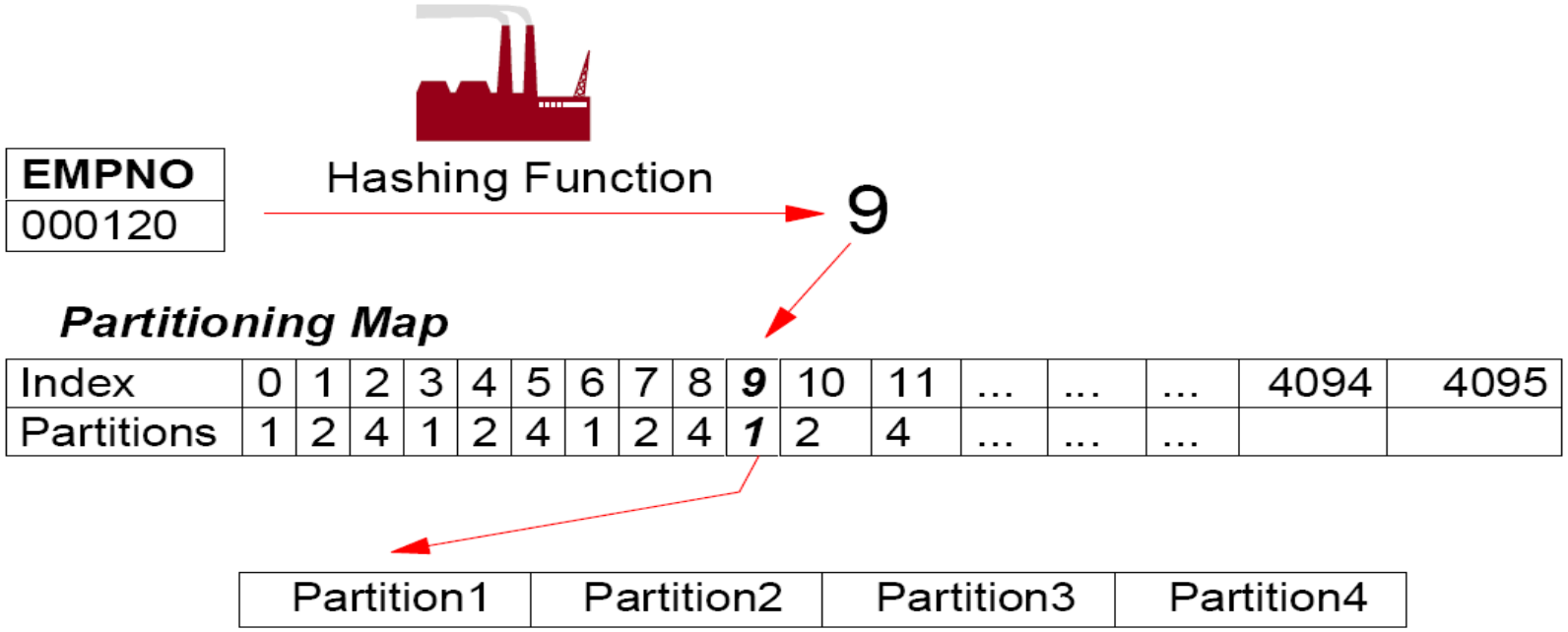
- Instance is defined across Partition 0, Partition 1, Partition 2, Partition 3
- Default partitioning map is created for partition group when partition group is created

  - IBMDEFAULTGROUP, IBMTEMPGROUP, IBMCATGROUP



Partitioning Map

- Created when partition group is created or during data redistribution
- Usually only one partitioning map for a partition group
- Partition Map is a vector of 4096 parallel database partition numbers
- The hashing algorithm uses the partition key as an input to generate a partition number



Partitioning Key (PK) Considerations

- Include frequently used join columns
- Should spread data evenly across partitions
- Broad range domain
- Formed with minimum number of columns
- Integer is more efficient than character which is more efficient than decimal
- No long fields allowed
- Unique index or primary key must be superset of PK
- If DB2_UPDATE_PART_KEY=ON, update of Partitioning Key allowed
- No alterations of Partitioning Key allowed

Activate Database

- The ACTIVATE DATABASE command starts up the specified databases

  - db2 activate database sample

- Allocates DB Global Memory
- Starts Database Processes
- Allocates Log files
- Must be shut down with the DEACTIVATE DATABASE command or with the db2stop command

- In a partitioned database, this command activate the selected partitioned database on all database partitions
- When connect without activate db, only coordinator and catalog partitions are activated

db2_all Command

- The db2_all command sends the command to all database partitions listed in the db2nodes.cfg file

- To send to all database partitions :

  - db2_all ‘command’

  - db2_all ‘command; command’

- Send command to only database partition 1 :

  - db2_all ‘<<+1< command’

- Send command to all database partitions except 1 :

  - db2_all ‘<<-1< command’

- To execute command simultaneously across all database partitions :

  - db2_all ‘|| command’

SQL Functions Unique to DB2 UDB DPF

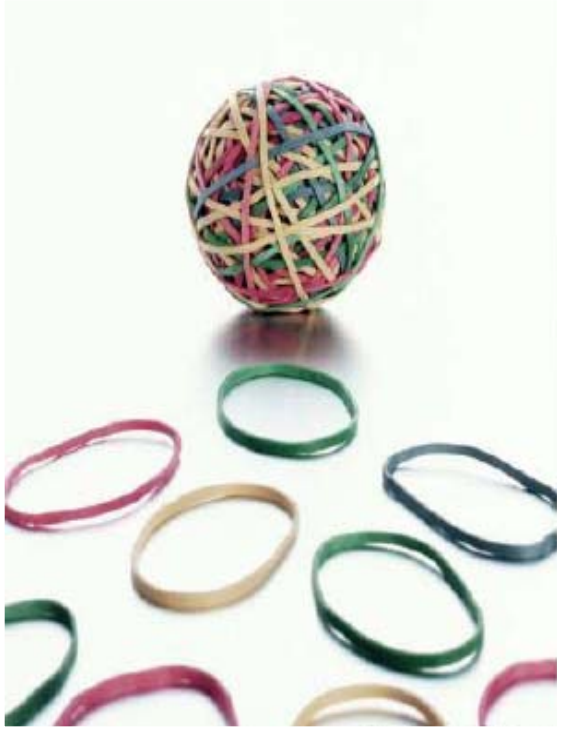
- DBPARTITIONNUM (COLUMN)
  - Returns the partition number of the rows
  - select count(*) from employee where dbpartitionnum(empno) = 3 ;
    - Returns # of rows of EMPLOYEE table on partition 3
- PARTITION (COLUMN)
  - Returns the hash bucket number of the rows
  - select count(*) from employee where partition(empno) = 4093 ;
    - Returns # of rows which hash to bucket 4093
- CURRENT NODE special register
  - Current node is set to coordinator partition
  - select empno from employee where partitionnum(empno) = CURRENT NODE ;
- Change working partition number
  - db2 terminate
  - export DB2NODE=2

Skew Check

- Data skew should NOT happened in any case
- Skew check example :

```
- select dbpartitionnum(col1), count(1)
  from table_name
  group by dbpartitionnum(col1)
  order by 1 ;
```

Agenda



1. Architecture and Concepts
2. Installation and Create Objects
3. Join Strategies and Explain
4. Performance Considerations

DB2 UDB DPF Join Concepts

- The data that is being joined must be physically located on the same partition
- If Not on the same partition, DB2 will need to relocate rows from one partition to another to physically accomplish the join
- DB2 has several methods to relocate the rows

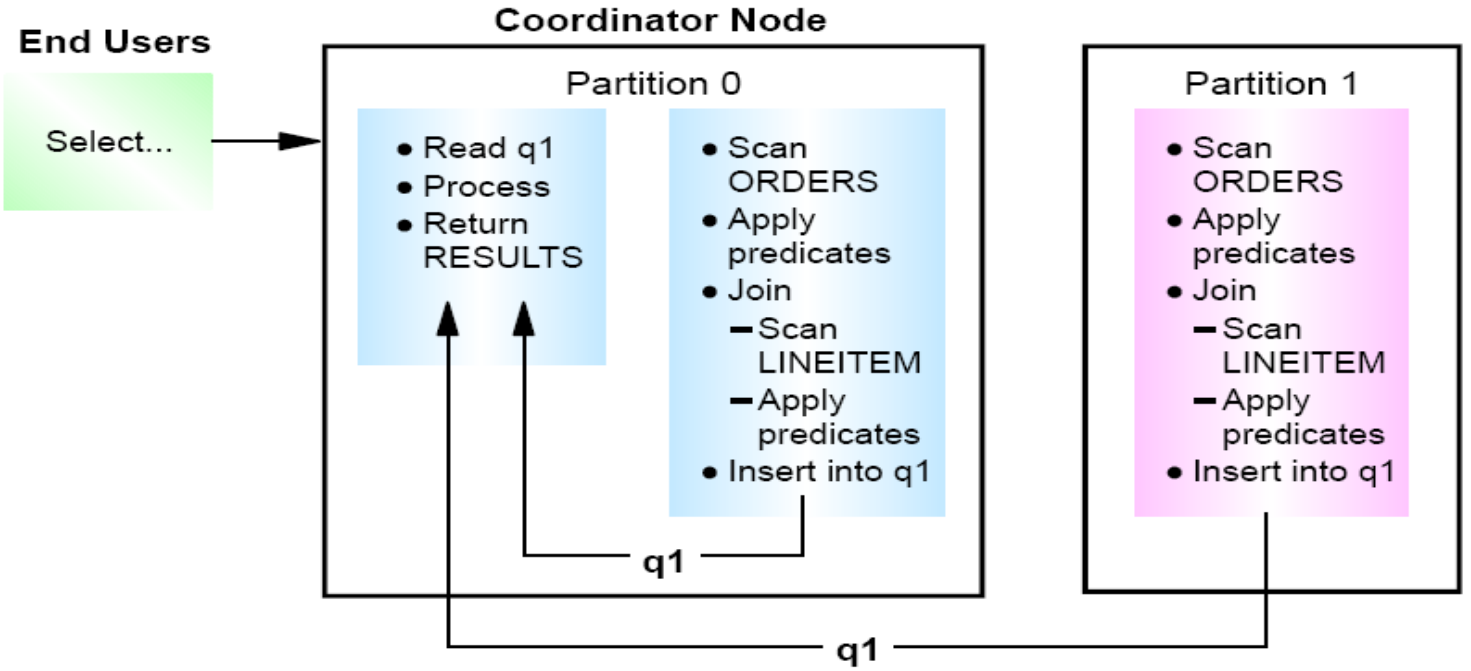
DB2 UDB DPF Join Strategies

- **Collocated**
  - Join locally on each partition
- **Directed Outer**
  - Hash rows from outer table based on join columns
  - Direct rows to partitions of inner table using join columns as the partition key
- **Directed Inner**
  - Hash rows from inner table based on join columns
  - Direct rows to partitions of outer table using join columns as the partition key
- **Directed Inner and Outer**
  - Hash rows on both tables to new partitions by hashing join columns
- **Broadcast Outer**
  - Broadcast outer table to all partitions containing the inner table
- **Broadcast Inner**
  - Broadcast inner table to all partitions containing the outer table

Collocated Join

- The join predicate is assumed to be :

  ORDERS.ORDERKEY = LINEITEM.ORDERKEY
- Both the LINEITEM and ORDERS tables are partitioned on the ORDERDEY column.
- The ORDERKEY column of two tables is assumed CHAR column
- The join is done locally at each database partition.

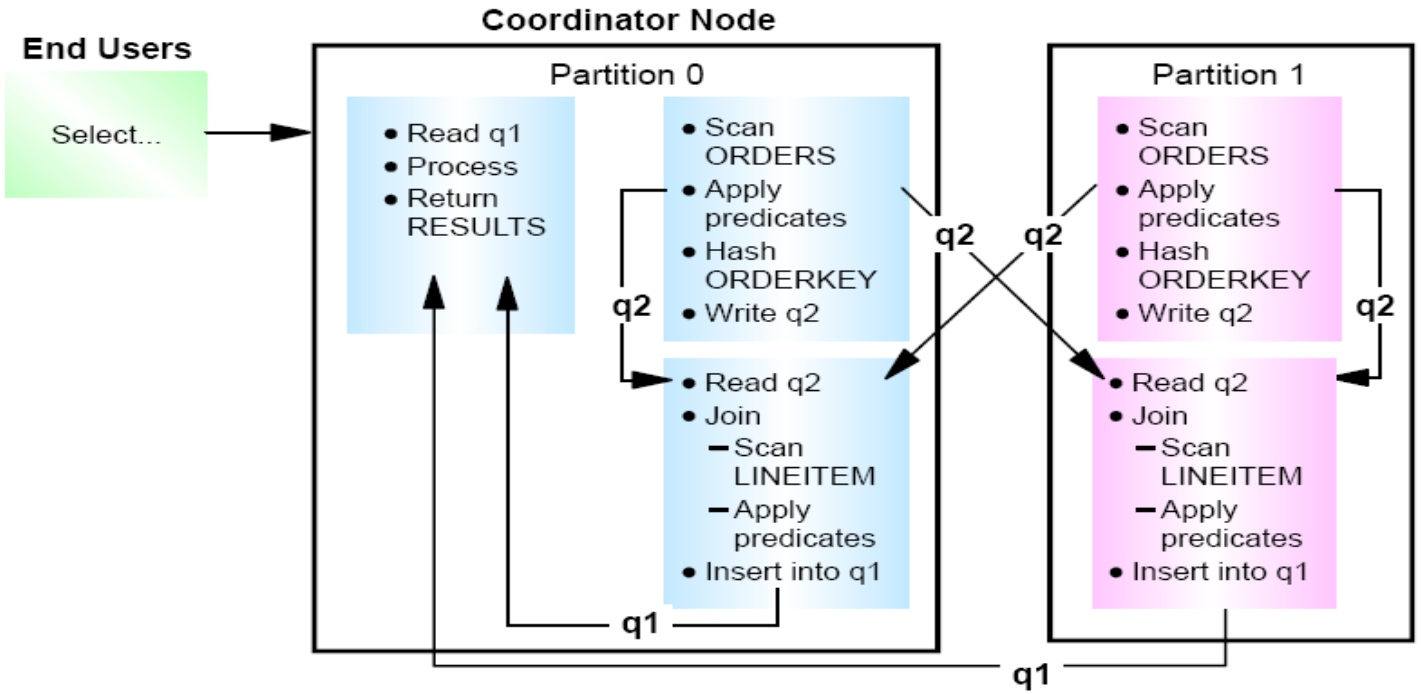


Directed Outer Table Join

- The join predicate is assumed to be :

  ORDERS.ORDERKEY = LINEITEM.ORDERKEY

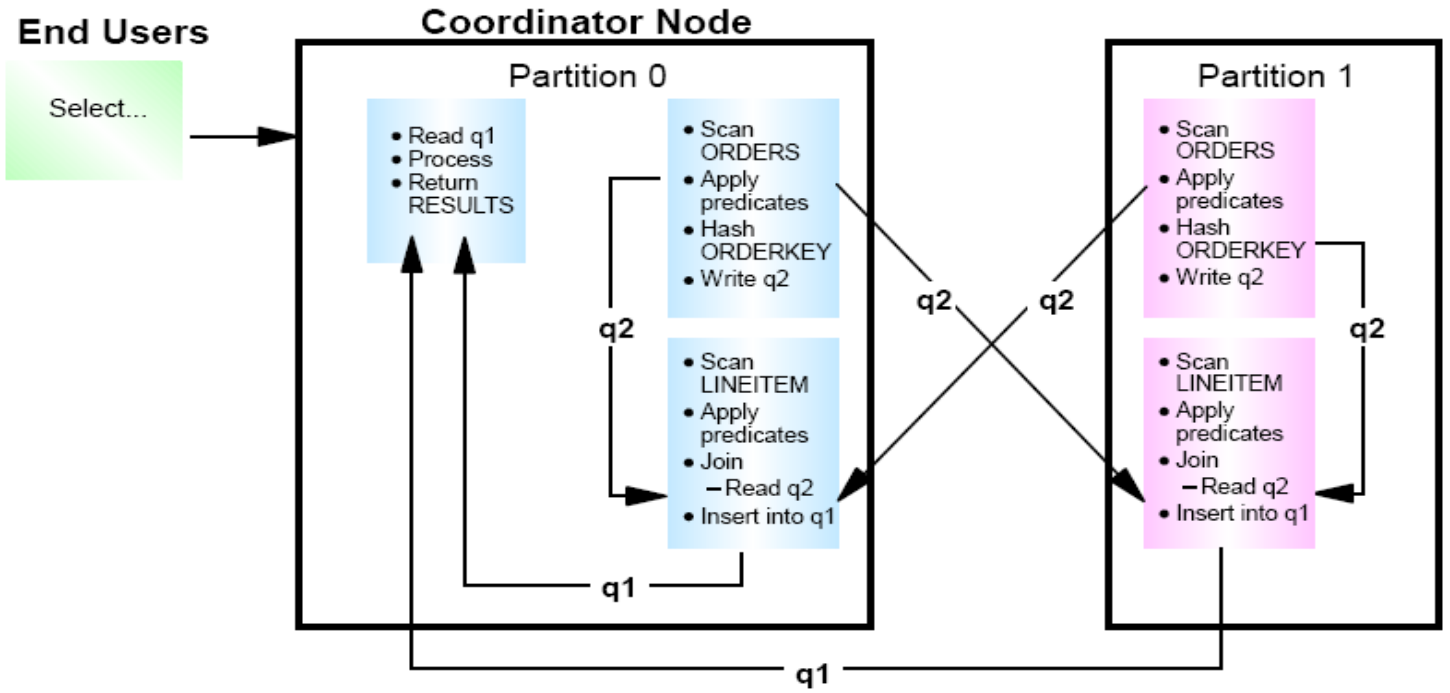
- The LINEITEM table is partitioned on the ORDERKEY column
- The ORDERS table is partitioned on a different column
- The ORDER table is hashed and sent to the correct LINEITEM table database partition



Directed Inner Table Join

- The join predicate is assumed to be :

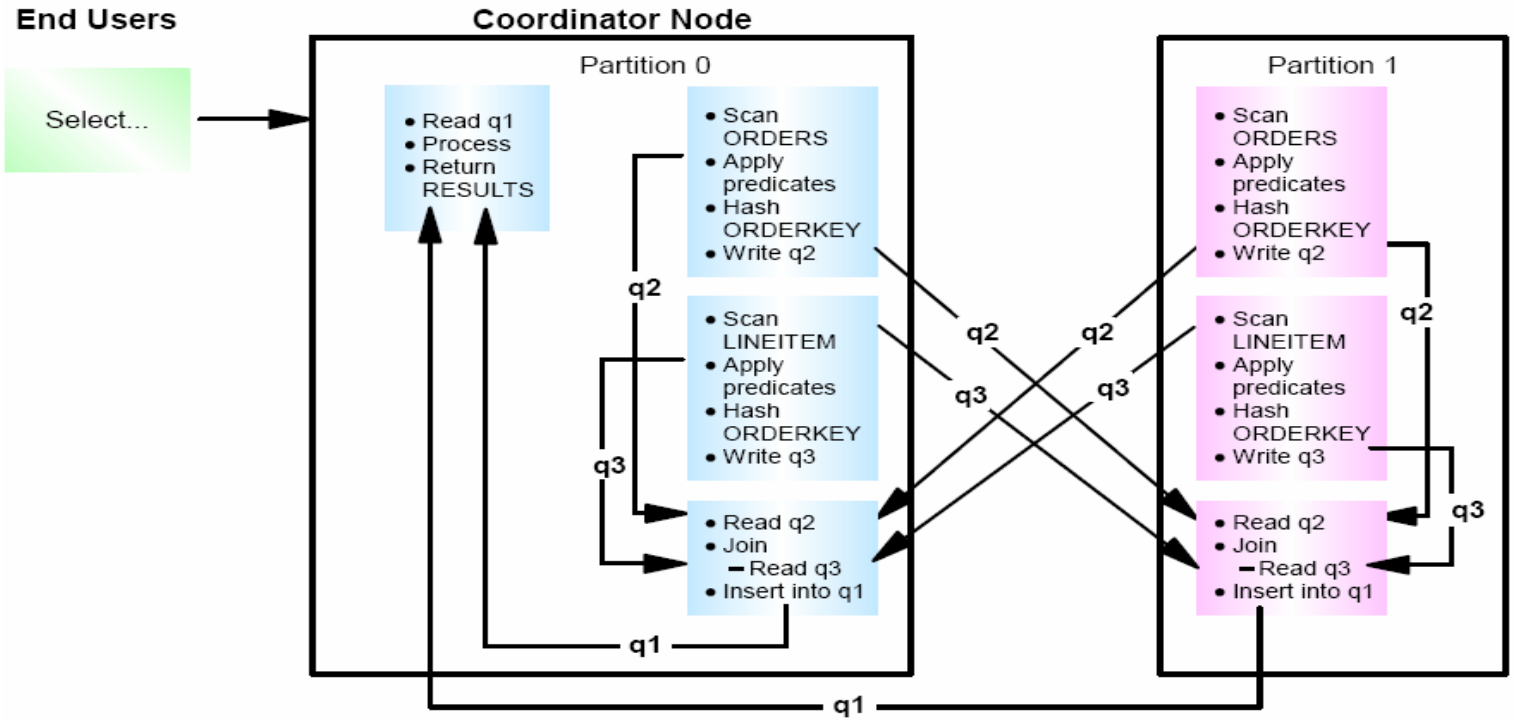
  ORDERS.ORDERKEY = LINEITEM.ORDERKEY
- The LINEITEM table is partitioned on the ORDERKEY column
- The ORDERS table is partitioned on a different column
- The ORDERS table is hashed and sent to the correct LINEITEM table database partition



Directed Inner and Outer Table Join

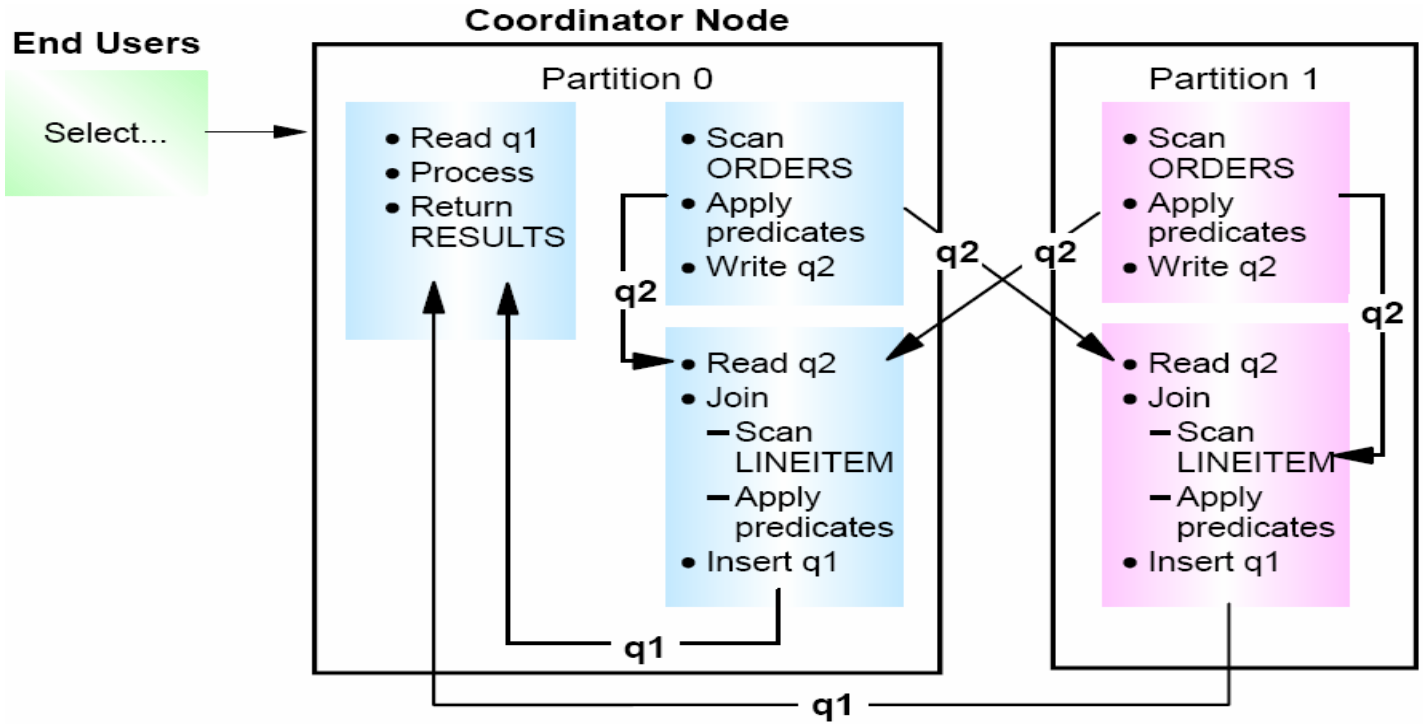
- The join predicate is assumed to be :

  ORDERS.ORDERKEY = LINEITEM.ORDERKEY
- Neither table is partitioned on the ORDERKEY column
- Both table are hashed and sent to the database partitions where they are joined
- Both table queues q2 and q3 are directed



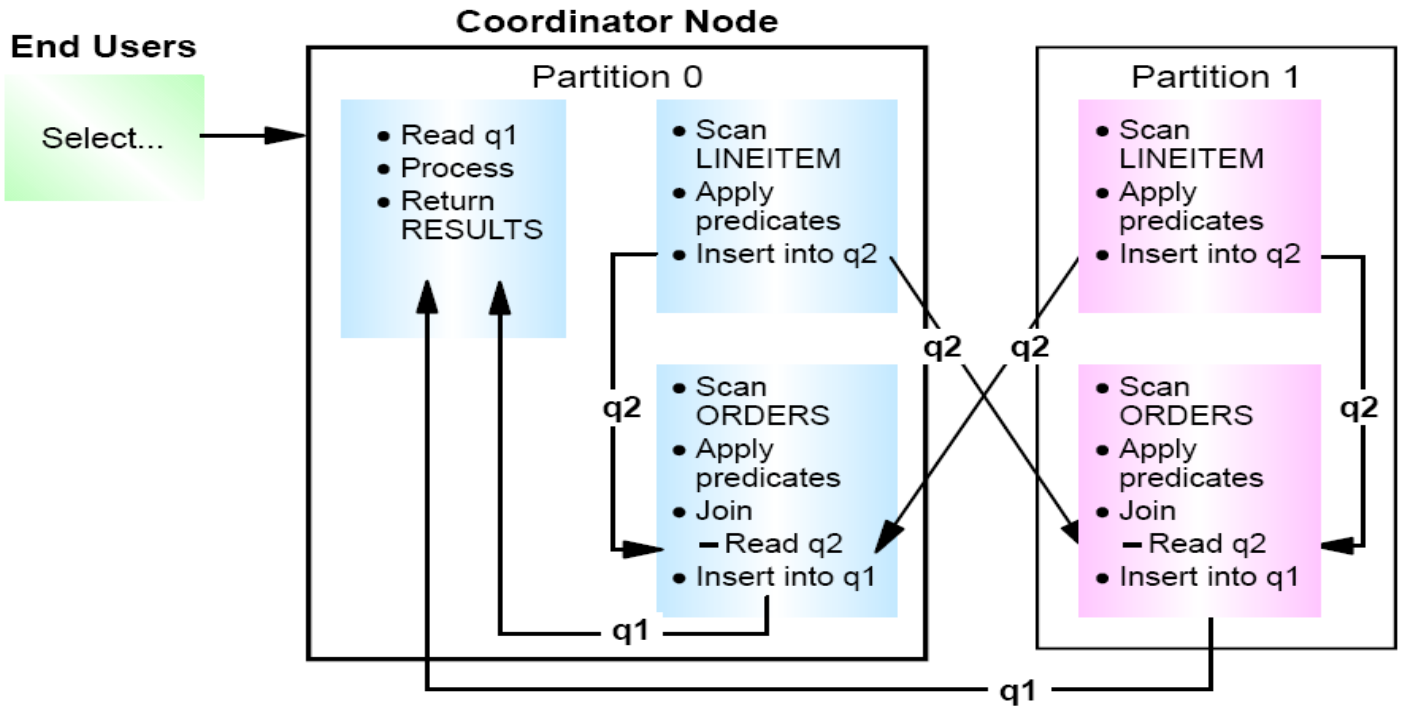
Broadcast Outer Table Join

- Neither table is partitioned on the join predicate columns
- LINEITEM table is very large table and ORDERS table is very small table
- ORDERS table is broadcast to all partitions of the LINEITEM table
- Rather than partition both tables, it may be cheaper to broadcast the smaller table to the larger table



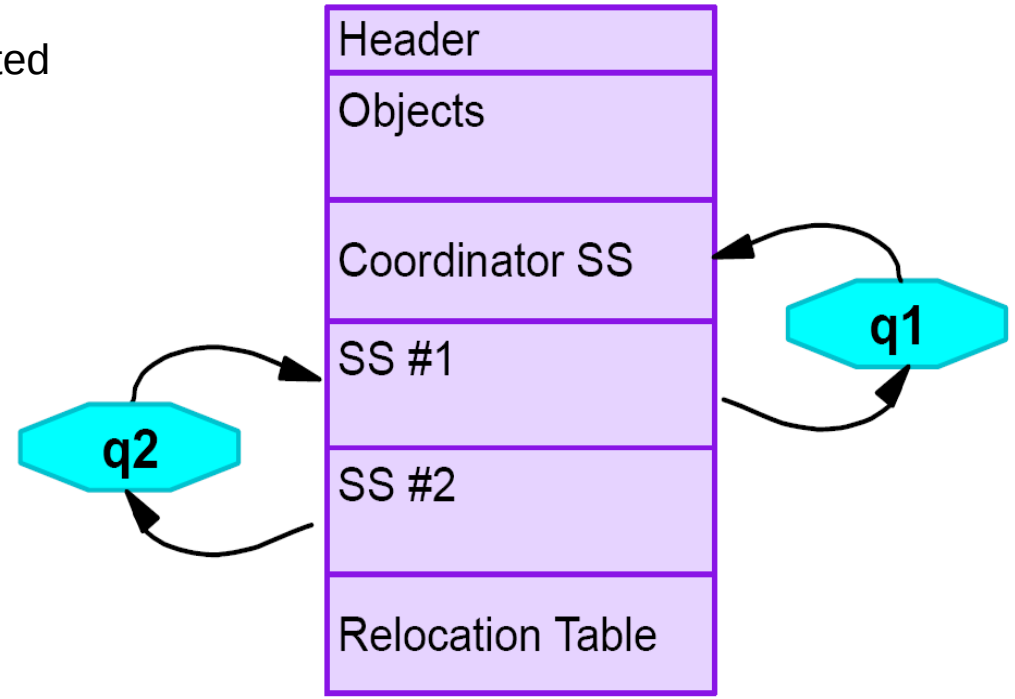
Broadcast Inner Table Join

- Neither table is partitioned on the join predicate columns
- ORDERS table is very large table and LINEITEM table is very small table
- LINEITEM table is broadcast to all partitions of the ORDERS table
- Rather than partition both tables, it may be cheaper to broadcast the smaller table to the larger table



Package Organization in Multipartitioned Database

- Each SQL statement has one section
- A section has one or more subsections in Multipartitioned environment
- Subsections are performed by processes
- The coordinator subsection is the master process
- Table queues connect subsections
- In complex query, many subsections is created



Subsection

- Subsections are used in inter-partition parallelism
- The optimizer may divide a query into subsection
- A subsection is the smallest unit of a SQL request which may be shipped between database partitions
- In intra-partition parallelism, subsections can be further parallelized via subsection pieces
- A subsection piece is a sequence of one or more database operators belonging to the same SQL query

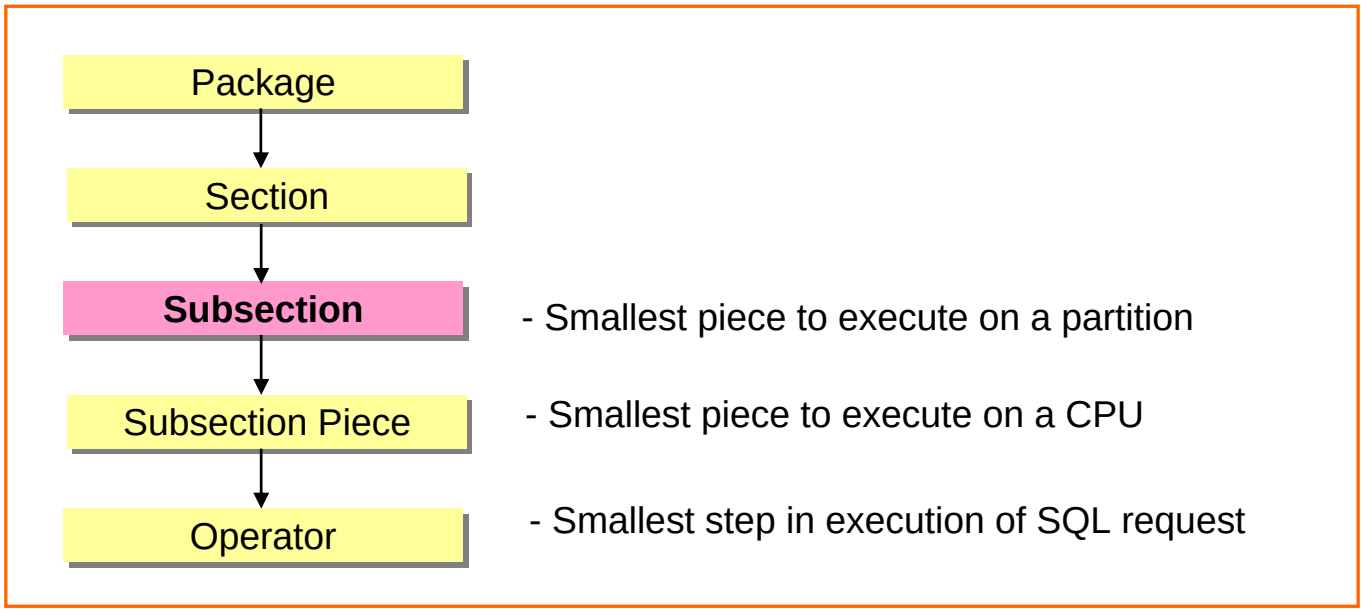
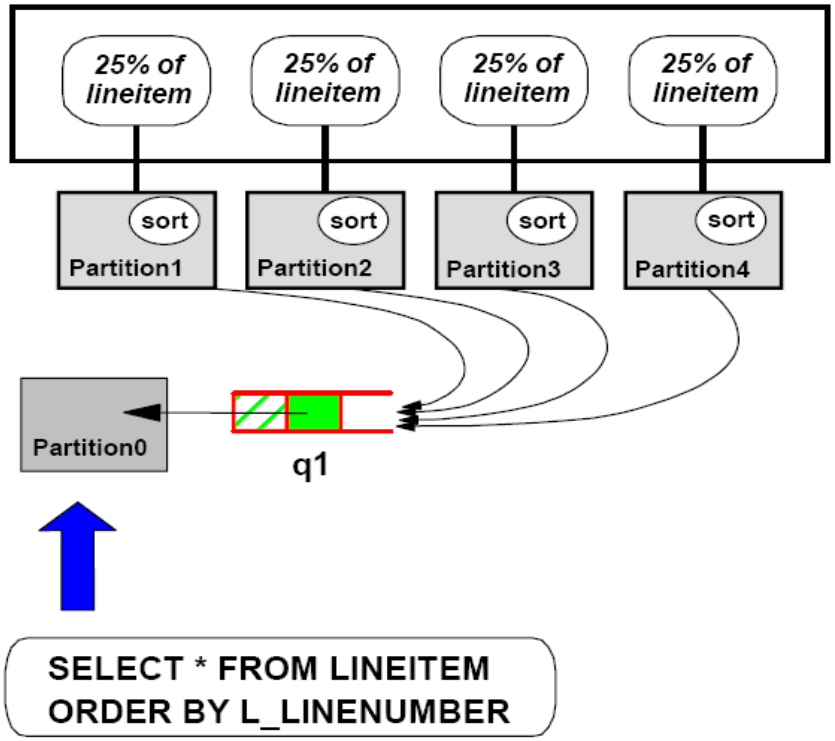


Table Queues

- A table queue can be thought of as a temporary table
- Never Materialized to disk – In memory only
- Used to communicate via table I/O
  - Read from table queue
  - Write to table queue
- A table queue physically consists of the 4KB Communication buffers



Subsections – Simple Select

SQL Statement:

```
select *
from inst31.lineitem
order by L_LINENUMBER
```

Coordinator Subsection - Main Processing:

```
Distribute Subsection #1
|  Broadcast to Node List
|  |  Nodes = 1, 2, 3, 4
Access Table Queue  ID = q1  #Columns = 16
|  Output Sorted
|  |  #Key Columns = 1
|  |  |  Key 1: (Ascending)
Return Data to Application
|  #Columns = 16
```

```
Subsection #1:
  Access Table Name = INST31.LINEITEM  ID = 8,4
  |  #Columns = 16
  |  Relation Scan
  |  |  Prefetch: Eligible
  |  Lock Intents
  |  |  Table: Intent Share
  |  |  Row  : Next Key Share
  |  Sargable Predicate(s)
  |  |  Insert Into Sorted Temp Table  ID = t1
  |  |  |  #Columns = 16
  |  |  |  #Sort Key Columns = 1
  |  |  |  |  Key 1: L_LINENUMBER (Ascending)
  |  |  |  Sortheap Allocation Parameters:
  |  |  |  |  #Rows     = 14878
  |  |  |  |  Row Width = 120
  |  |  |  Piped
  Sorted Temp Table Completion  ID = t1
  Access Temp Table  ID = t1
  |  #Columns = 16
  |  Relation Scan
  |  |  Prefetch: Eligible
  |  Sargable Predicate(s)
  |  |  Insert Into Asynchronous Table Queue  ID = q1
  |  |  |  Broadcast to Coordinator Node
  |  |  |  Rows Can Overflow to Temporary Table
  Insert Into Asynchronous Table Queue Completion  ID = q1

End of section
```

Optimizer Plan:

```
RETURN
(  1)
  |
MBTQ
(  2)
  |
TBSCAN
(  3)
  |
 SORT
(  4)
  |
TBSCAN
(  5)
  |
Table:
INST31
LINEITEM
```

Merge Broadcast Table Queue

Subsections – Collocated Join (1)

SQL Statement:

```
select count(*)
from inst31.lineitem, inst31.orders
where l_orderkey =o_orderkey
```

Coordinator Subsection - Main Processing:

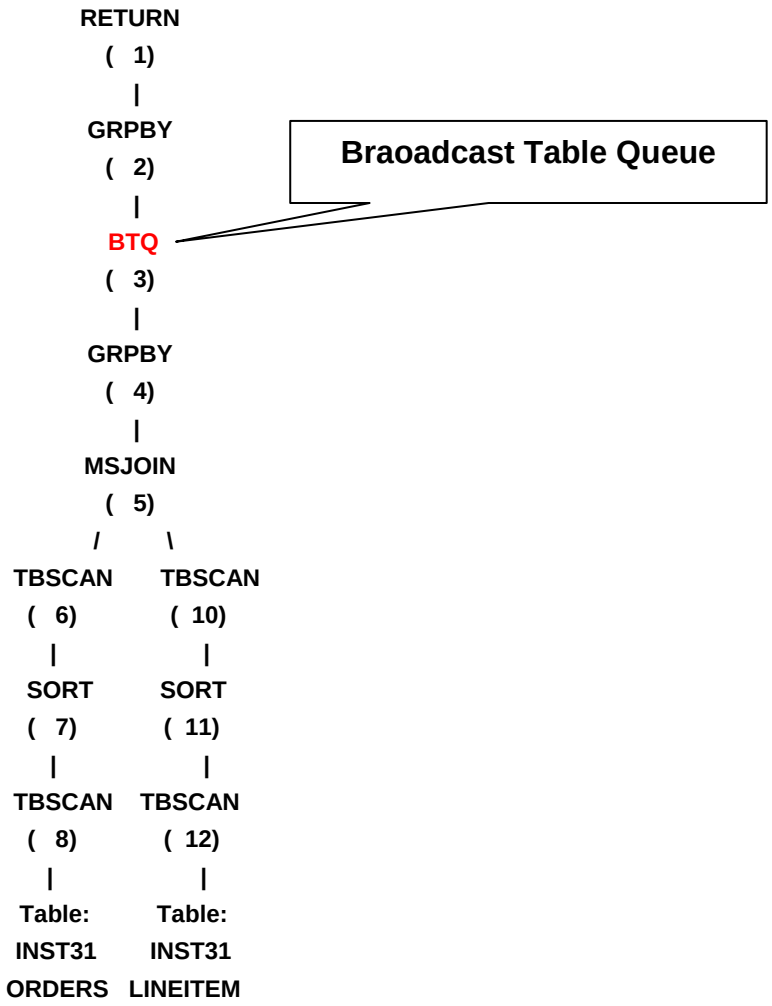
```
Distribute Subsection #1
|  Broadcast to Node List
|  |  Nodes = 1, 2, 3, 4
Access Table Queue  ID = q1  #Columns = 1
Final Aggregation
|  Column Function(s)
Return Data to Application
|  #Columns = 1
```

Subsection #1:

```
Access Table Name = INST31.ORDERS  ID = 6,4
|  #Columns = 1
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Sargable Predicate(s)
|  |  Insert Into Sorted Temp Table  ID = t1
|  |  |  #Columns = 1
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: O_ORDERKEY (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 3742
|  |  |  |  Row Width = 8
|  |  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 1
|  Relation Scan
|  |  Prefetch: Eligible
Merge Join
|  Access Table Name = INST31.LINEITEM  ID = 8,4
|  |  #Columns = 1
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Sargable Predicate(s)
|  |  |  Insert Into Sorted Temp Table  ID = t2
|  |  |  |  #Columns = 1
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  Key 1: L_ORDERKEY (Ascending)
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 14878
|  |  |  |  |  Row Width = 8
|  |  |  |  Piped
|  Sorted Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 1
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Partial Aggregation
|  Column Function(s)
Insert Into Asynchronous Table Queue  ID = q1
|  Broadcast to Coordinator Node
|  Rows Can Overflow to Temporary Table
```

Subsections – Collocated Join (2)

Optimizer Plan:



db2exfmt Tool

- Format the contents of the explain tables
- Externalize the information provided by Visual Explain in a text-based output
- Must create EXPLAIN tables

  - execute ‘$HOME/sqllib/misc/EXPLAIN.DDL’

- Populate the EXPLAIN tables

  - db2 connect to dss
  - db2 set current explain mode explain
  - < Execute Query >
  - db2 set current explain mode no

- Invoke db2exfmt

  - db2exfmt –d <dbname> -w -1 -n % -s % -# 0 -o exfmt.out

db2exfmt – Output Summary

- Original/Optimized Statement
- Access Plan Graph
- Plan Operators
- Plan Operator's Detail Description

db2exfmt Output – Original/Optimized

Original Statement:

```
select count(*)
from inst31.lineitem, inst31.orders
where l_orderkey = o_orderkey
```

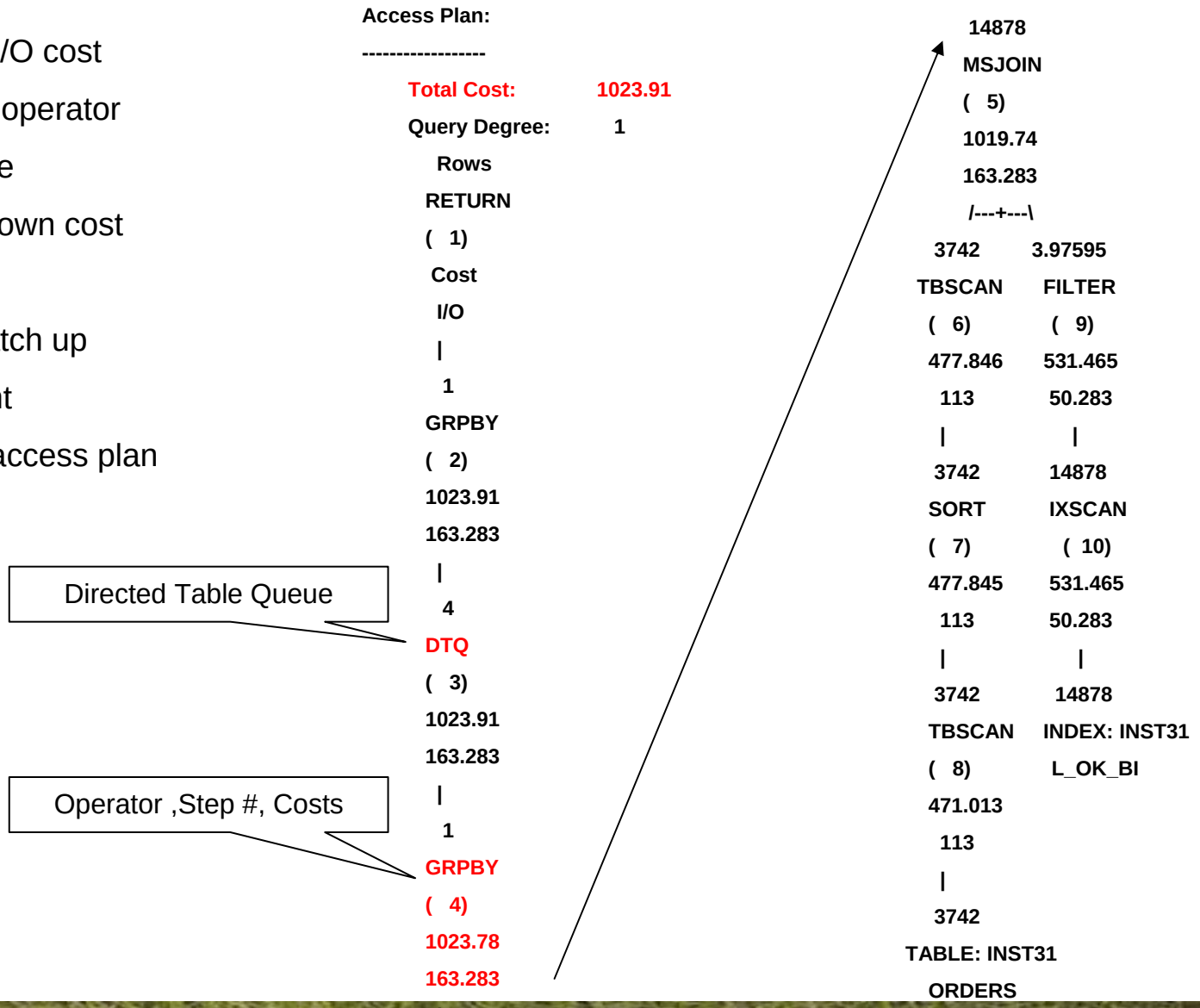
- **Reverse-translated from the internal compiler representation of the query**
- **To allow an understanding of the SQL context from which the SQL optimizer chose the access plan**

Optimized Statement:

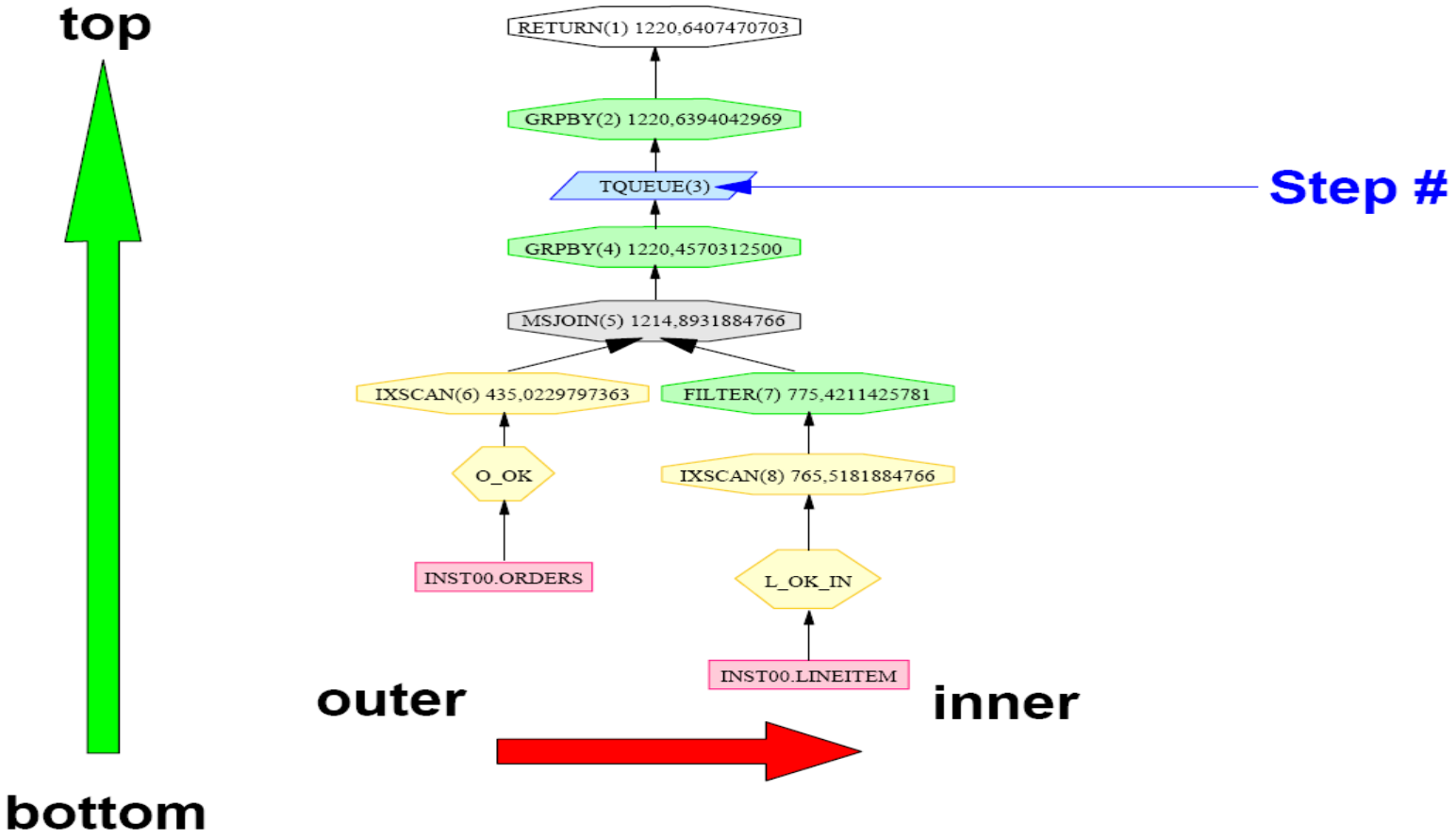
```
SELECT Q4.$C0
FROM
  (SELECT COUNT(1)
  FROM
    (SELECT $RID$
    FROM INST31.ORDERS AS Q1, INST31.LINEITEM AS Q2
    WHERE (Q2.L_ORDERKEY = Q1.O_ORDERKEY)) AS Q3) AS Q4
```

db2exfmt Output – Access Plan

- A graph with total cost, I/O cost and cardinality for each operator
- The costs are cumulative
- Every operator adds its own cost to the plan
- Operator IDs can be match up the steps in the different representations of the access plan



Access Plan – Visual Explain



db2exfmt Output – Plan Operators

- Base access methods
  - TBSCAN, IXSCAN, FETCH,..
- Joins
  - NLJOIN : Nested loop join
  - MSJOIN : Merge scan join
  - HSJOIN : Hash Join
- Aggregation
  - GRPBY, SUM, MIN/MAX,….
- Temping (TEMP) and sorting (SORT)
- Specialized operations
  - Index ANDing (IXA), dynamic bitmap indexing
  - Index Oring and list prefetch (RIDSCAN)
  - Table queues (TQ)
    - Broadcast (BTQ)
    - Directed (DTQ)
    - Merging option (MDTQ, MBTQ)
    - Local table queue for SMP intra-partition parallelism (LTQ)

db2exfmt Output – Plan Operator Details

- Every operator has its own description like:

  - Costs
    - Total CPU and I/O resources consumed

  - Arguments
    - Details on how operators executes
    - Details are saved in the EXPLAIN_ARGUMENT table

  - Predicates
    - A predicate is an element of a search condition that expresses or implies a comparison operation
    - Predicates are included in clauses beginning with WHERE or HAVING

  - Input Stream/Output Stream
    - Represents the input and output data streams between individual operators and data objects.
    - The data objects themselves are represented in the EXPLAIN_OBJECT table

Plan Operator – Costs Example

3) TQ : (Table Queue)

Cumulative Total Cost:	1023.91	Cumulative costs
Cumulative CPU Cost:	5.29762e+07	
Cumulative I/O Cost:	163.283	
Cumulative Re-Total Cost:	1023.78	Cost to re-execute subplan - fetching the next row
Cumulative Re-CPU Cost:	5.28555e+07	
Cumulative Re-I/O Cost:	163.283	
Cumulative First Row Cost:	1023.86	Total cost to return first row
Cumulative Comm Cost:	4.03857	Communication cost
Cumulative First Comm Cost:	0	
Estimated Bufferpool Buffers:	66	Bufferpool pages required by this operator

Plan Operator – Arguments Example

- Provide details on how operators execute
- Description provided in EXPLAIN_ARGUMENT Table

```
5) MSJOIN: (Merge Scan Join)
        ..........
Arguments:
        ---------
        EARLYOUT: (Early Out flag)
           FALSE
        INNERCOL: (Inner Order Columns)
           1: L_ORDERKEY(A)
        OUTERCOL: (Outer Order columns)
           1: O_ORDERKEY(A)
        TEMPSIZE: (Temporary Table Page Size)
           4096


6) IXSCAN: (Index Scan)
Arguments:
        ---------
        JN INPUT: (Join input leg)
           OUTER
```

Get next outer after finding first match on inner.
Guaranteed one match on inner.

Inner Column in Ascending order

Outer Column in Ascending order

Indicates if operator is the operator feeding the inner or outer of a join.

Plan Operator – Predicates Example

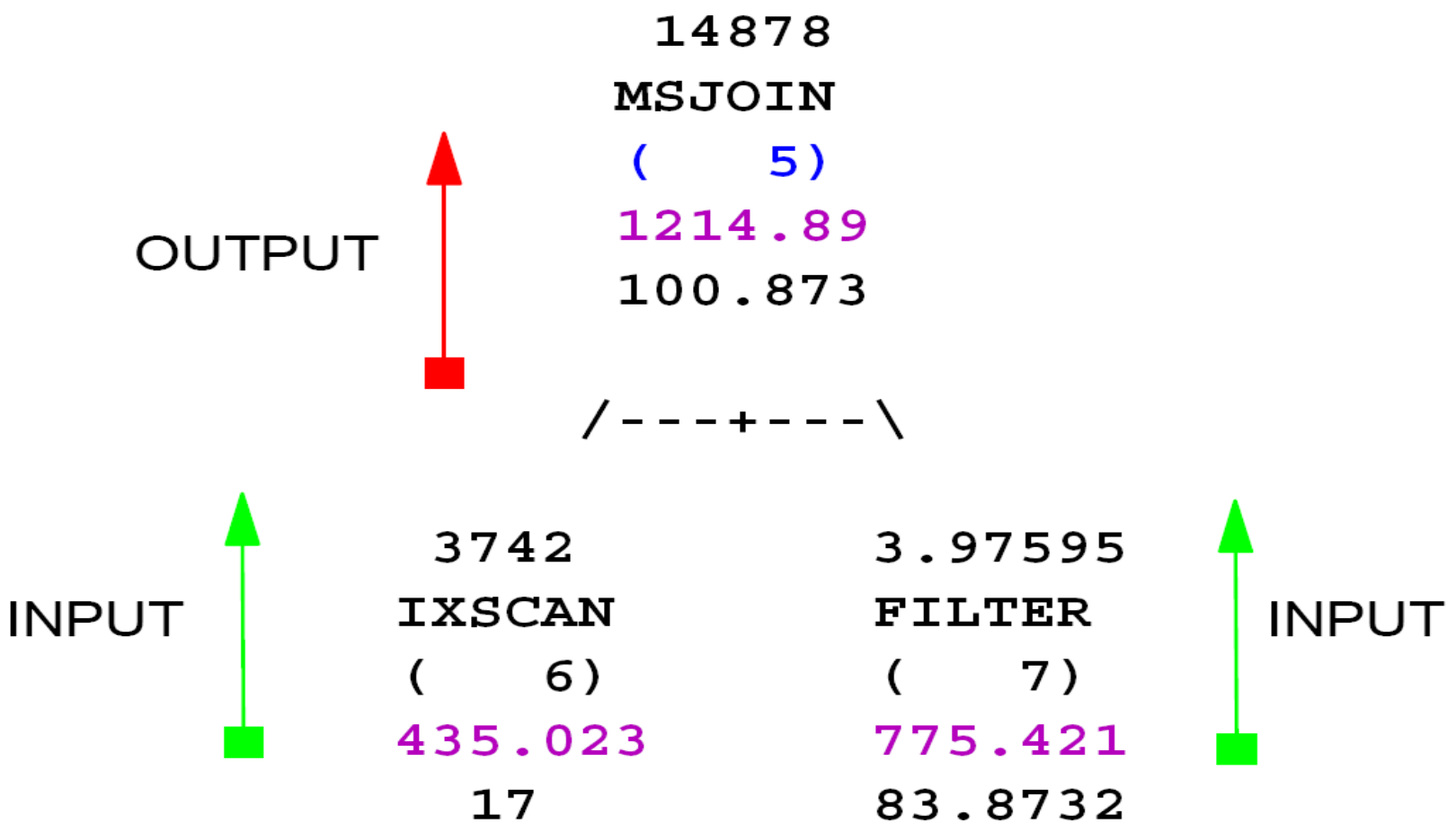
- Provide details on which predicates (if any) are applied
- Description provided in EXPLAIN_PREDICATE Table

```
8) IXSCAN: (Index Scan)
        .                                        7.65423
        .                                        IXSCAN
        .                                        (   8)
    Predicates:                                  983.995
    ----------                                   83.8732
    2) Sargable Predicate                          |
        Relational Operator:        Equal (=)     15186
        Subquery Input Required:    No          INDEX: INST64
        Filter Factor:          0.000504032     L_OK_PK

        Predicate Text:
        --------------
        (Q1.L_PARTKEY = 1924)
```

Filter Factor * #Rows = Cardinality

0.000504032*15186 = 7.65423

Plan Operator – Input/Output Streams

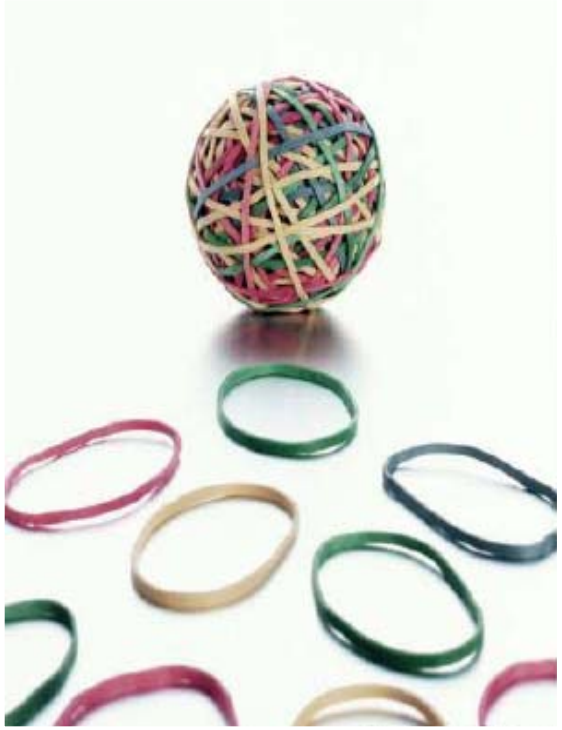


Plan Operator – Input/Output Streams Example

```
Input Streams:
--------------------
  2) From Operator #6
        Estimated number of rows:  3742
        Partition Map ID:          3
        Partitioning:              (MULT )
                                Multiple Partitions
        Number of columns:         2
        Subquery predicate ID:     Not Applicable

        Column Names:
        ------------
        +O_ORDERKEY(A)+$RID$

        Partition Column Names:
        ----------------------
        +1: O_ORDERKEY

  5) From Operator #7
        Estimated number of rows:  3.97595
        Partition Map ID:          3
        Partitioning:              (MULT )
                                Multiple Partitions
        Number of columns:         2
        Subquery predicate ID:     Not Applicable

        Column Names:
        ------------
        +L_ORDERKEY(A)+$RID$

        Partition Column Names:
        ----------------------
        +1: L_ORDERKEY
```

```
Output Streams:
--------------
  6) To Operator #4

        Estimated number of rows:  14878
        Partition Map ID:          3
        Partitioning:              (MULT )
                                Multiple Partitions
        Number of columns:         0
        Subquery predicate ID:     Not Applicable

        Partition Column Names:
        ----------------------
        +NONE
```

Agenda



1. Architecture and Concepts
2. Installation and Create Objects
3. Join Strategies and Explain
4. Performance Considerations

System Memory Considerations

- Consider the memory required by the operating system
- At the database manager level, the primary memory usage parameters deal with the requirements of FCM
- At the database level, the parameter that needs to be considered most highly is the configuration of your buffer pools

  - OLTP environment

    To allocate up to 75% of the machine's remaining memory to the buffer pools

  - OLAP environment

    The recommendation would tend toward 50% of the machine's remaining memory to the buffer pool
- Sort heap threshold (sheapthres)

  - A starting point is the memory remaining after the buffer pool space
- Sort heap (sortheap)

  - sheapthres/(number of concurrent sorts)

  - In an OLAP environment, all of the executing agents will be doing sorts
- Application memory

  - For local applications – those running on the database server

  - They can have a significant impact on the amount of memory available on the machine

Fast Communication Manager

- Fast Communication Manager transfers data between partitions
- fcm_num_buffers
  - DBM CFG Parameter
  - Number of FCM buffers
  - Specifies number of 4KB buffers used for internal communications among and within db partitions
  - FCM daemons on the same node communicate through UNIX sockets
- FCM Monitor Data Elements
  - db2 get snapshot for dbm
  - db2 get snapshot for fcm for all nodes

FCM Snapshot

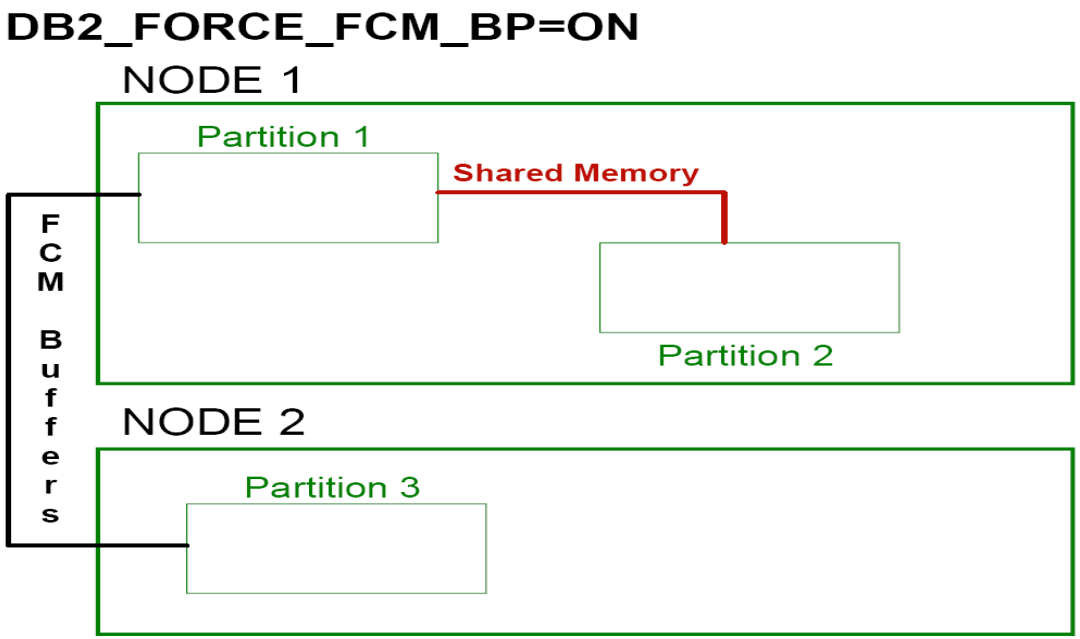
Node FCM information corresponds to	= 1
Free FCM buffers	= 4093
Free FCM buffers low water mark	= 4071
Free FCM message anchors	= 1534
Free FCM message anchors low water mark	= 1530
Free FCM connection entries	= 1536
Free FCM connection entries low water mark	= 1523
Free FCM request blocks	= 2022
Free FCM request blocks low water mark	= 2011
Snapshot timestamp	= 02/18/2008 23:42:48.482434
Number of FCM nodes	= 4

Node Number	Total Buffers Sent	Total Buffers Received	Connection Status
1	0	0	Active
2	95	135	Active
3	85	115	Active
4	86	117	Active

DB2_FORCE_FCM_BP – Registry Variable

- Applicable to DB2 UDB ESE for AIX when using multiple logical partitions
- Enabled by default on all non-AIX platforms (OFF), Not Applicable
- When this registry variable is ON :

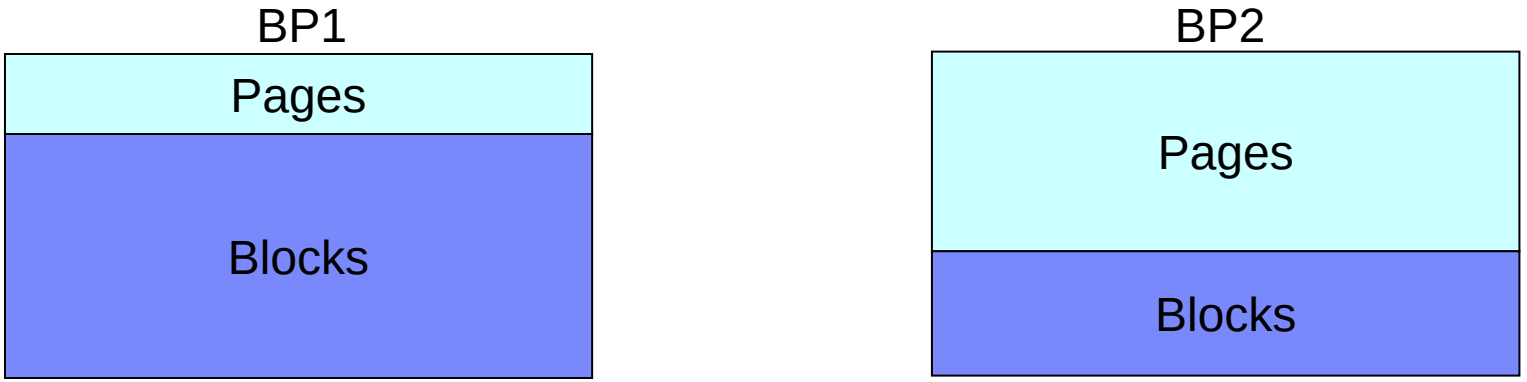
  - The FCM buffers are always created in a separate memory segment
  - The communication between FCM daemons occurs through shared memory



Block-Based Buffer Pools

- By default, the buffer pools are page-based
  - Prefetching pages from disk is expensive because of I/O overhead
  - Most platforms provide High-performance primitives that read contiguous pages from disk into non-contiguous portions of memory

- Block-based buffer pools for improved sequential prefetching
  - Sequential prefetching can be enhanced if contiguous pages can be read into contiguous pages within a buffer pool
  - A Block-based Buffer Pool has both 'Page Area' and a 'Block Area'
  - For optimal performance, BLOCKSIZE should be less than table space extent size or equal
  - If applications don't use sequential prefetching, then block area of the buffer pool is wasted



- Create bufferpool bufferpool-name size-of-pages numblockpages 60 blocksize 32 ;

Monitoring Prefetch using Block I/O and Vectored I/O

- db2 get snapshot for bufferpools on sample

```
Vectored IOs                    = 8362028
Pages from vectored IOs         = 28771169
Block IOs                       = 1094466149
Pages from block IOs            = 17417600434
```

- Vectored IOs
  - The number of vectored I/O requests
- Pages from vectored IOs
  - The total number of pages read by vectored I/O into the page area of the buffer pool
- Block IOs
  - The number of block I/O requests
- Pages from block IOs
  - The total number of pages read by block I/O into the block area of the buffer pool

Database Monitoring – Snapshot

- The snapshot monitor switches can be obtained using the command
  - db2 get monitor switches [at dbpartitionnum n | global]

- Default values for these switches can be defined within DBM CFG File
  - db2 update dbm cfg using DFT_MON_BUPOOL ON
  - db2 update dbm cfg using DFT_MON_LOCK ON

- The monitor switches can be turn on or off using the command
  - db2 update monitor switches using switch-name on [at dbpartitionnum n | global]

- Taking snapshots using GET SNAPSHOT command
  - db2 get snapshot for database on sample [at dbpartitionnum n | global]
  - db2 get snapshot for bufferpools on sample [at dbpartitionnum n | global]

- Taking snapshots using SQL Table functions

  To capture a snapshot for the current connected partition :

  ```
  - select rows_written, rows_read, table_name
    from (snapshot_table('TP1', -1)) as snap;
  ```

  To capture a global snapshot :

  ```
  - select rows_written, rows_read, table_name
    from (snapshot_table('TP1', -2)) as snap;
  ```

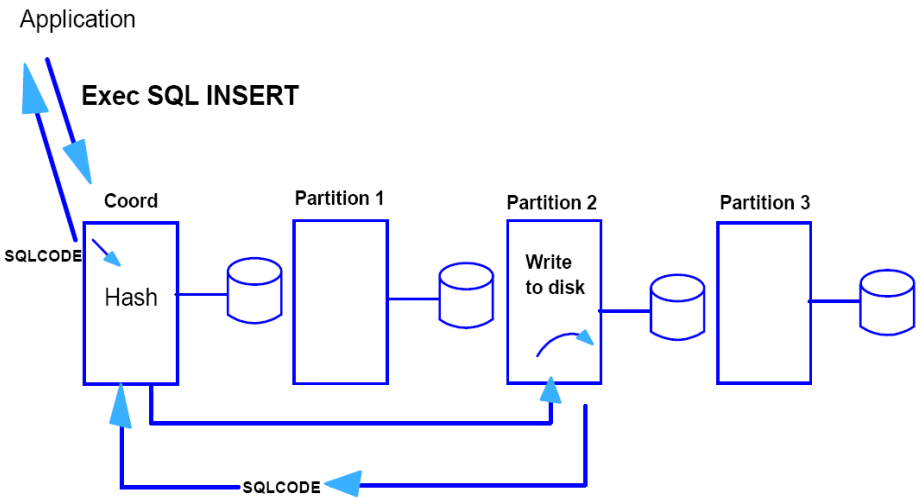
db2advis – Partitioning Recommendations

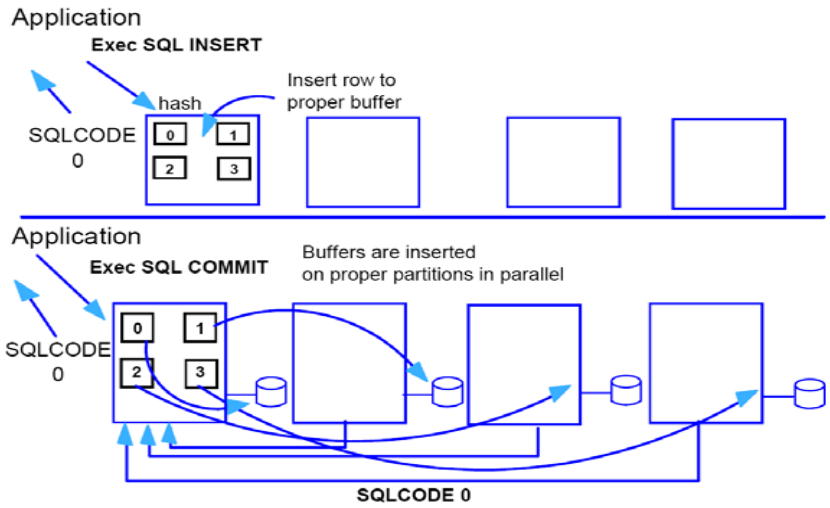
- To recommend new indexes of the tables in a query workload
- To recommend changes in the partitioning keys in a query workload
- The option '-m IP' requests the design advisor to evaluate both indexes and partitioning keys
- To help you migrate from a single-partition to a multi-partition database
- Can only recommend partitioning on DB2 ESE

```
# db2advis –d dss –i join1.sql –m IP –o partadvise.ddl
```

Non-Buffered Insert & Buffered Insert

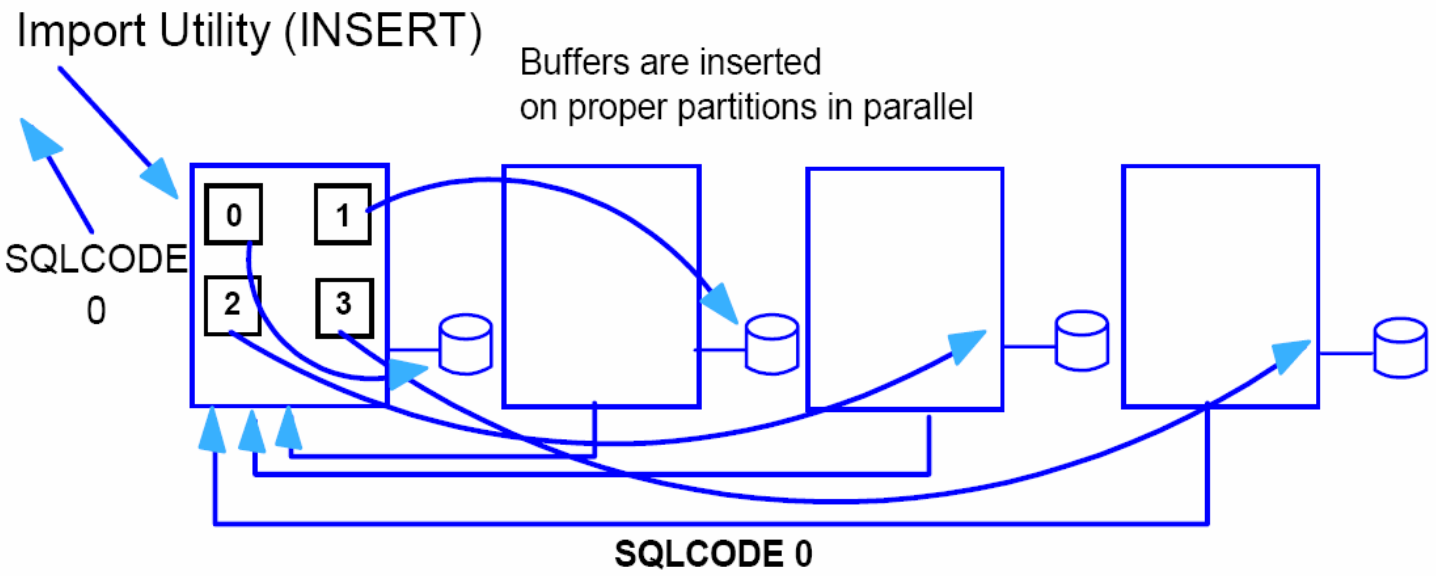
- Non-Buffered Insert
  - The Default insert strategy
  - INSERT DEF precompile or bind option
  - Each Row is individually hashed and loaded to the appropriate partition
  - Occurs serially across all partitions
- Buffered Insert
  - INSERT BUF precompile or bind option
  - Each Row is hashed into a 4KB insert buffer at the coordinator partition
  - When the buffer becomes full, the buffer is sent to its target partition
  - Occurs in parallel across all partitions





Import with Buffered Inserts

- Use the DB2 bind utility to request buffered inserts
  - db2 bind $HOME/sqllib/bnd/db2uimpm.bnd grant public blocking all insert buf
- But details about a failed buffered insert are not returned – Error reporting



Application Design – Partition Connection

- Application aware of data distribution can connect to correct partition
- High performance OLTP : Local Bypass
- Specifies the database partition to which a connect is to be made
  - db2 set client connect_dbpartitionnum n
  - db2 connect to dss user inst01 using inst01
- Specifies the database partition to which an attach is to be made
  - db2 set client attach_dbpartitionnum n
  - db2 attach to inst01 user inst01 using inst01
- Permits the client to connect to the catalog database partition
  - db2 set client connect_dbpartitionnum catalog_dbpartitionnum

DB2_PARALLEL_IO – Registry Variable

- If this registry variable is not set, the degree of parallelism of any table space will be the number of containers of the table space

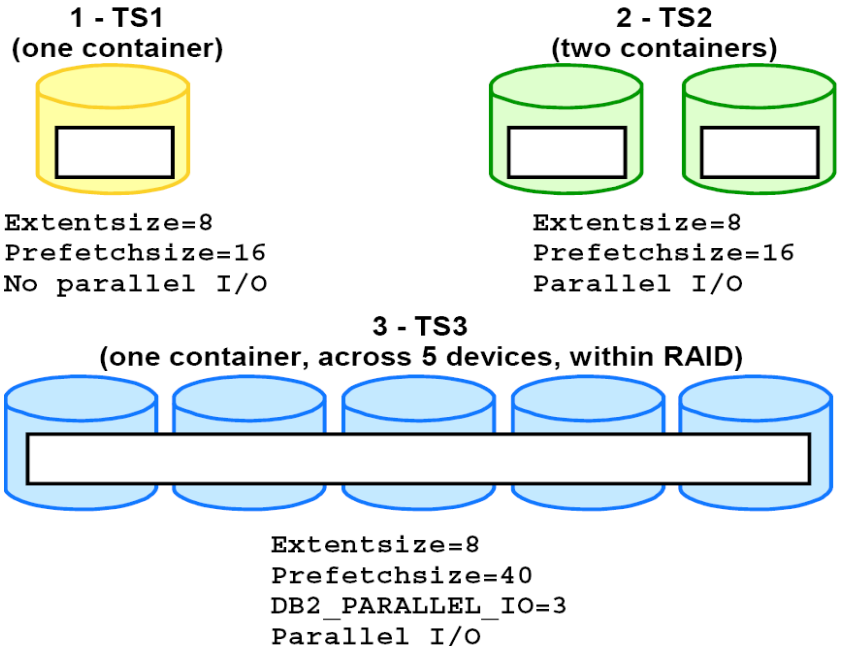
- If this registry variable is set, then the degree of parallelism of the table space will be the ratio between the prefetch size and the extent size of this table space

- The prefetch size should be calculated based on the following equation:

  pretetch size = (number of containers) * (number of disks per container) * extent size

- db2set DB2_PARALLEL_IO=*

  db2set DB2_PARALLEL_IO=*,1:3



Automatic Prefetch Calculation

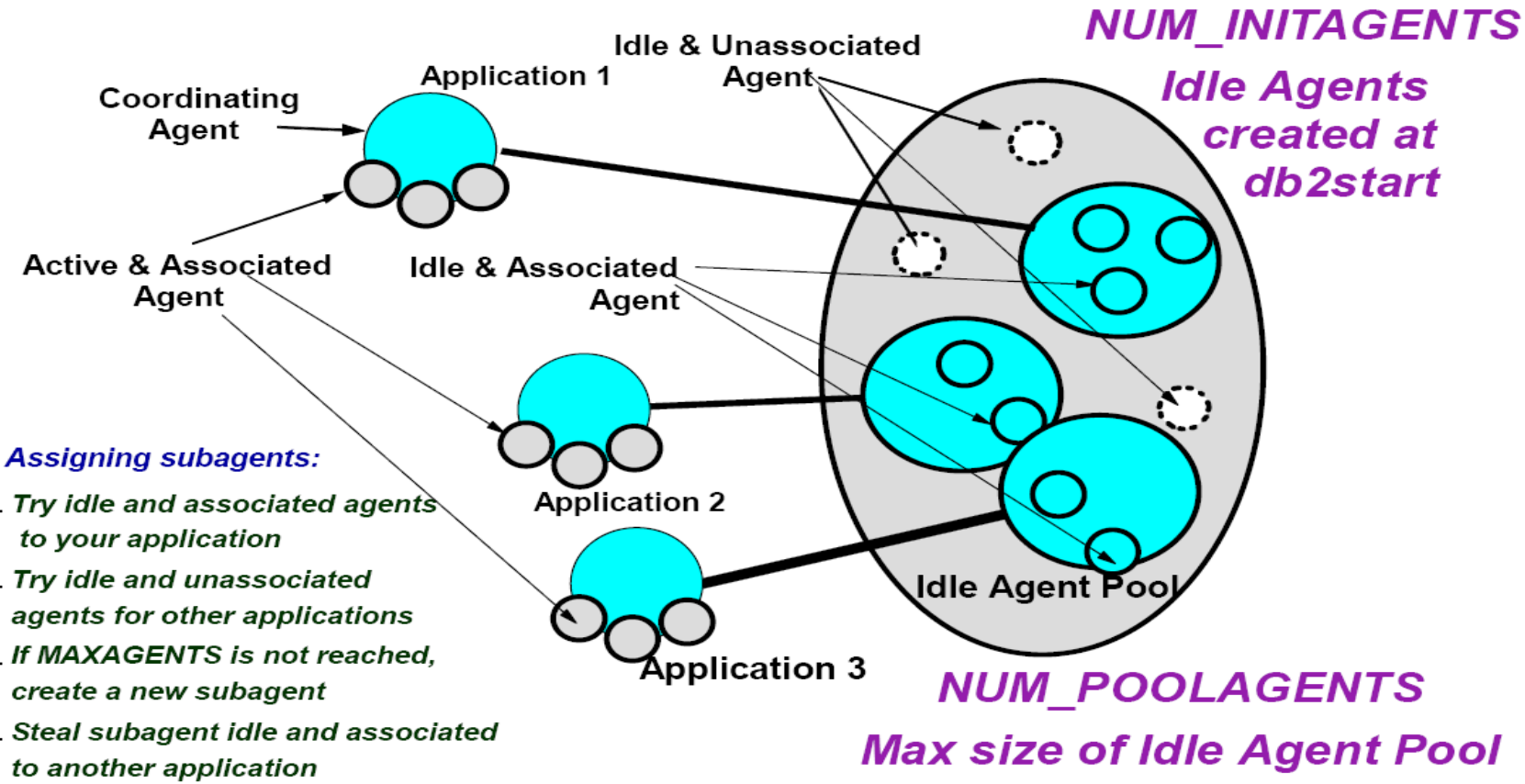
- If a table space is created with a PREFETCHSIZE of AUTOMATIC
- Or If the database is configured with DFT_PREFETCH_SZ of AUTOMATIC and no PREFETCHSIZE is specified for the table space
- DB2 will automatically calculate and update the prefetch size of table space using the following equation:
  - prefetch size = (# containers) * (# phisical spindles) * extent size
  - Physical spindles can be specified through the DB2_PARALLEL_IO
    - db2set DB2_PARALLEL_IO=1:3
    - A default of 6 at DB2_PARALLEL_IO (the value for a RAID-5 device)
- This calculation is performed:
  - At database start-up time
  - When a table space is first created with AUTOMATIC prefetch size
  - When the number of containers for a table space changes through execution of an ALTER TABLESPACE statement
  - When the prefetch size for a table space is updated to be AUTOMATIC through execution of an ALTER TABLESPACE statement

Processes, Connections and Agents

- In a non-partitioned, but intra-patition parallel environment
  - maxcagents
    The maximum number of db2agents concurrently executing a database manager transaction
  - max_coordagents
    Limits the maximum number that can be allocated
  - maxappls
    The maximum number of concurrent applications that can connect to a database

- In a partitioned environment
  - maxcagents = max_coordagents
    - The maximum number of db2agents concurrently executing a database manager transaction.
    - Total of all coordinator and subagents.
      - db2agent : coordinator agent
      - db2agntp : Active subagent
      - db2agnta : Idle subagent
  - maxappls
    The maximum number of applications that can be active a database partition

Agent States – Assigning Subagents



Q&A