IBM Information Management software

DB2 Partitioning

Data Management Emerging Partnerships and Technologies
IBM Toronto Lab

IBM Software Group

IBM Information Management software

Data Management – DB2 9.5

Winter/Spring 2008

Data Management Emerging Partnerships and Technologies
IBM Toronto Lab

Overview

- Partitioning Before DB2 9
- Partitioning in DB2 9
- How to Create Table Partitioning
- Storing Table Objects
- Accessing Partitioned Data
- Roll-Out Scenario
- Roll-In Scenario
- Putting it all together

Partitioning Before DB2 9

DB2 Partitioning : Shared Nothing?

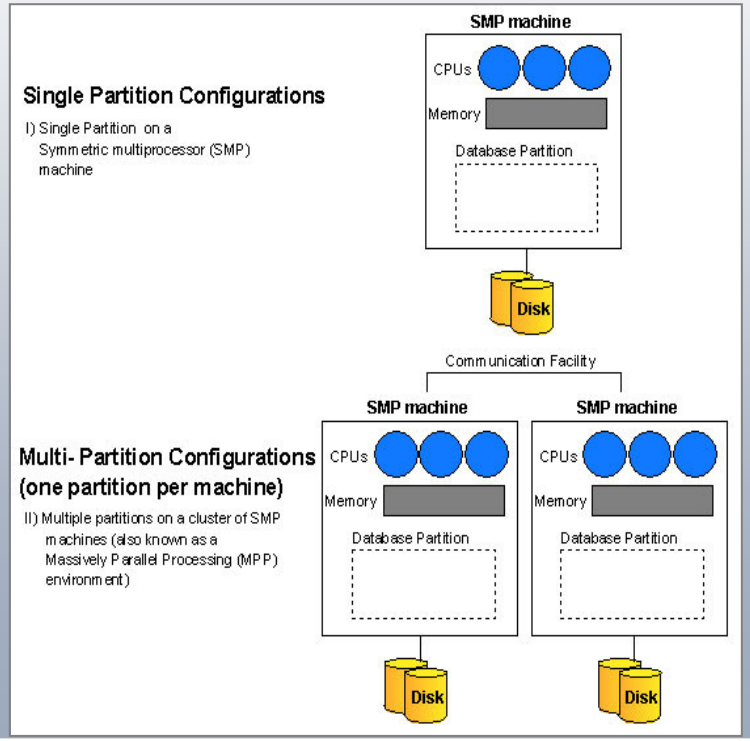
- **Allows you to partition your database across multiple servers or within a large SMP server**
- **Users can connect to the database and issue commands as usual without the need to know the database is spread across among several partitions**
- **Allows for scalability cause you can add machines and spread your database across them**
- **Means more CPU's, more memory, and more disks**
- **Ideal for larger databases from data warehousing, data mining, online analytical processing or working with online transaction processing workloads**

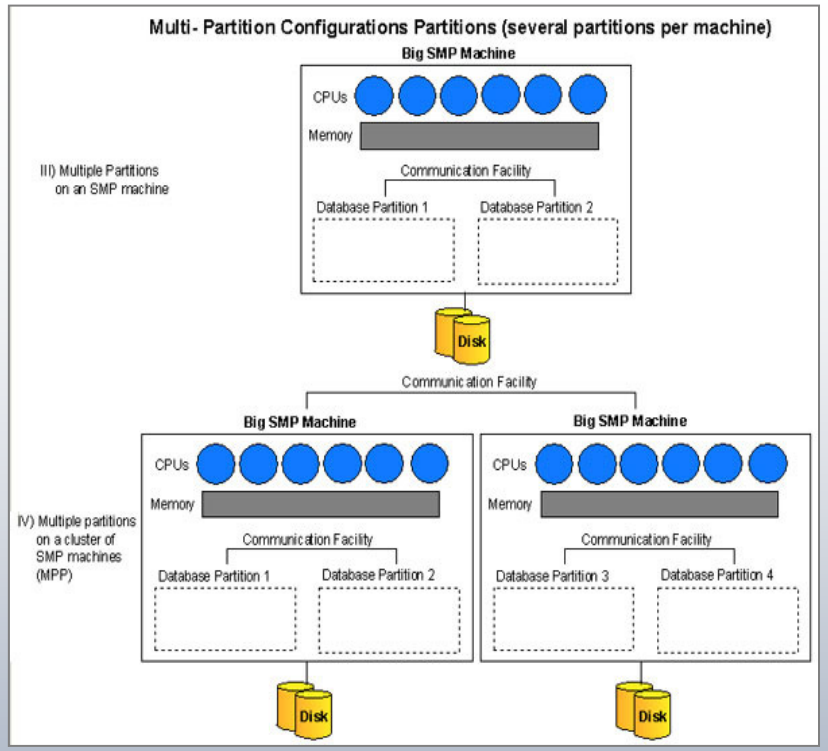
DB2 with DPF

- **Why** Partition **– Scale Out , Performance …**
- **DB2 core** Scale-Out **architecture based on** Parallelism **aka (Shared Nothing)**
  - Inter and intraNode/partition Parallelism… Inter Query Parallelism.. Intra Query…
  - performance, divide and rule, limitless scale out
  - Cost-based Optimizer with Query Rewrite
  - Full Parallelism for SQL and Utilities
  - Dynamic throttling based on load
  - Asynchronous I/O Parallel I/O

Database Partitioning Feature

- **Allows you to partition your database across one or more multiple servers**





- **Allows you to have multiple partitions across one or more multiple servers**

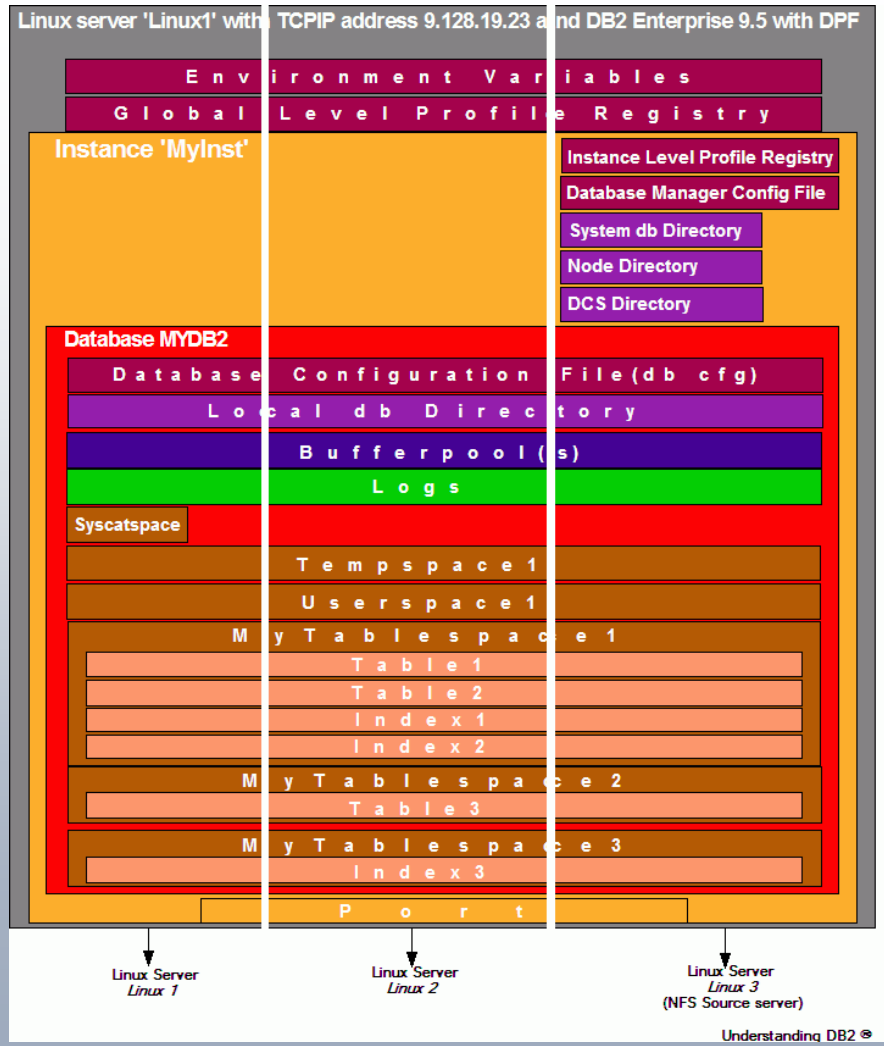
Database Partitioning Feature (DPF)

- Database Level Partitioning
  - DPF
  - Database on shared disk or dedicated disk
  - Active-Active/Active-Passive/Cascade scenarios
    - By setting DB2NODE Environment Variables to active (1) or passive (0)
  - SQL Functions shipped between partitions
  - Adapts to all hardware configurations (small SMP, Large SMP, NUMA, clusters)
  - To protect against failure use HACMP, HADR, etc.

Partition	Server Name	Logical Port
0	serverA	0
1	serverA	1
2	serverB	0
3	serverB	1
4	serverC	0
5	serverC	1
6	serverD	0
7	ServerD	1

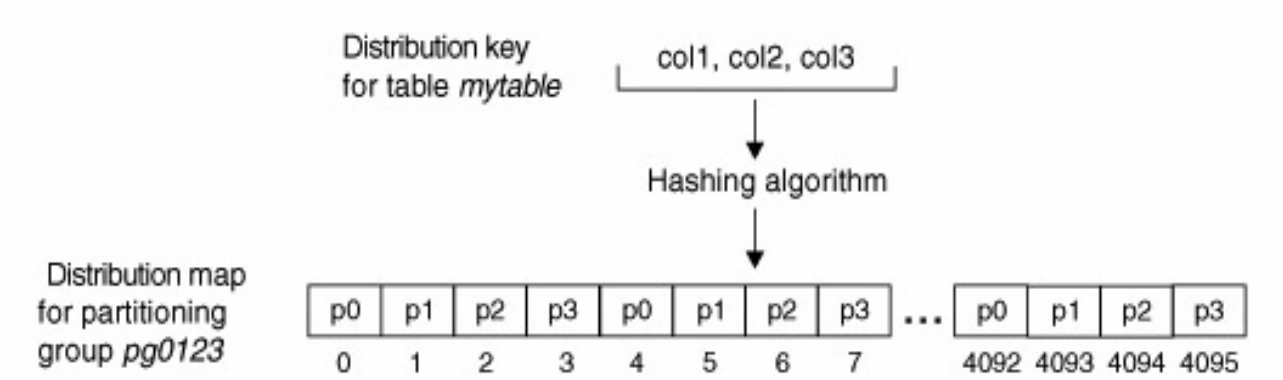
DPF: How it's Distributed

- **To visualize how a db2 environment is split in a DPF system**
- **All partitions share**
  - Instance level profile registry
  - Database manager config file (dbm.cfg)
  - System db dictionary
  - Node dictionary
  - DCS directory
- **Each server can have its own**
  - Environment variables
  - Global-level profile registry variable
  - Database configuration file
  - Local database directory
  - Log files
  - Lock managers
  - Can have different FixPak levels



DPF: How its Distributed

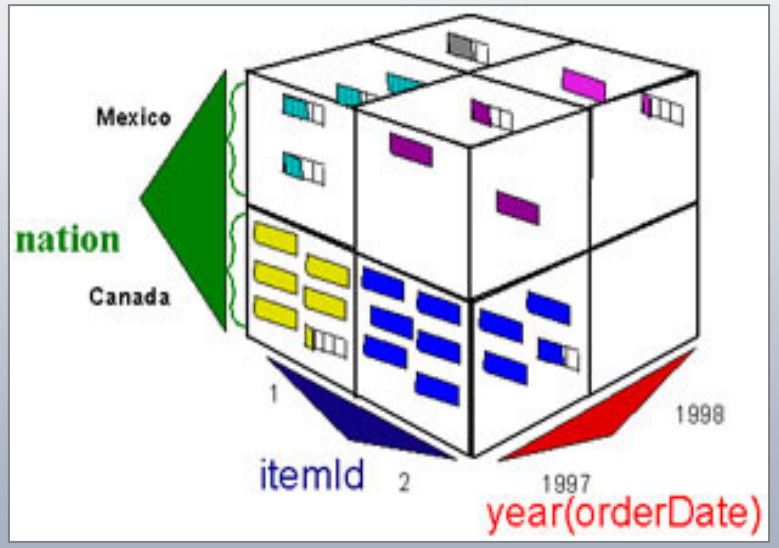
- **Partition Group** is a set of one or more database partitions
- **Distribution Map**[1] is an internally generated array used for decided where data will be stored within the partitions
  - Partition numbers are specified in a round-robin fashion in the array
- **Distribution key**[1] is a column(s) that determines the partition on which a particular row of data is physically stored
  - Can define key using CREATE TABLE statement with the DISTRIBUTE BY[2] clause
- A combination of distribution map, distribution key and a hashing algorithm is used to determine which partition to store a given row



- Note [1]: Prior to DB2 9, the DISTRIBUTION MAP and DISTRIBUTION KEY terms were known as PARTITIONING MAP and PARTITIONING KEY respectively.
- Note [2]: Prior to DB2 9, the distribution key can be defined using create table statement with the PARTITIONING BY clause, the older syntax is retained for backward compatibility.

Multidimensional Clustering

- **Allows for clustering of the physical data pages in multiple dimensions**
- **Guarantees clustering over time even if there are frequent INSERT operations performed**
- **Blocks**
  - DB2 places records that have the same column values in physical locations that are close together
- **Block Indexes**
  - indexes that point to an entire block of pages
- **Cells**
  - Blocks that have the same dimension values are group together



Partitioning in DB2 9

DB2 Table Level Partitioning

- Allows a single logical table to be broken up into multiple separate physical storage objects (up to 32K range partitions)
  - Each corresponds to a 'partition' of the table
  - Partition boundaries correspond to specified value ranges in a specified partition key
- Main Benefits
  - Increase table capacity limit
  - Partition elimination during SQL processing improve performance
  - Optimized roll-in / roll-out processing (e.g. minimized logging)
  - "Divide and conquer" management of huge tables
  - Family compatibility with DB2 on z/OS and IDS
  - Improved HSM integration

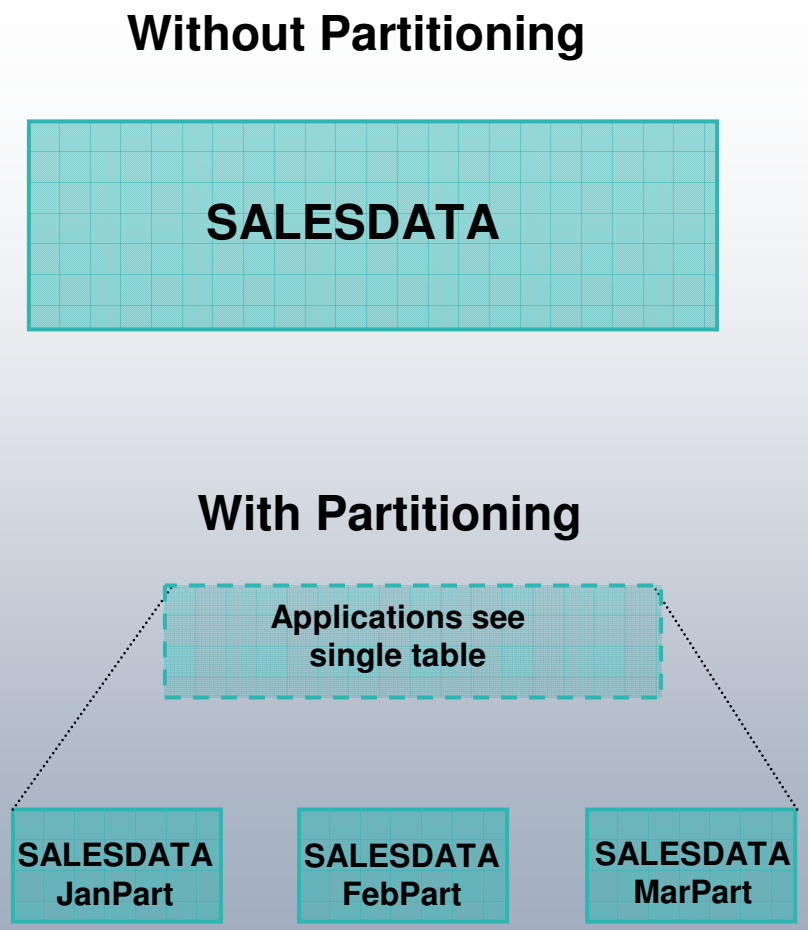
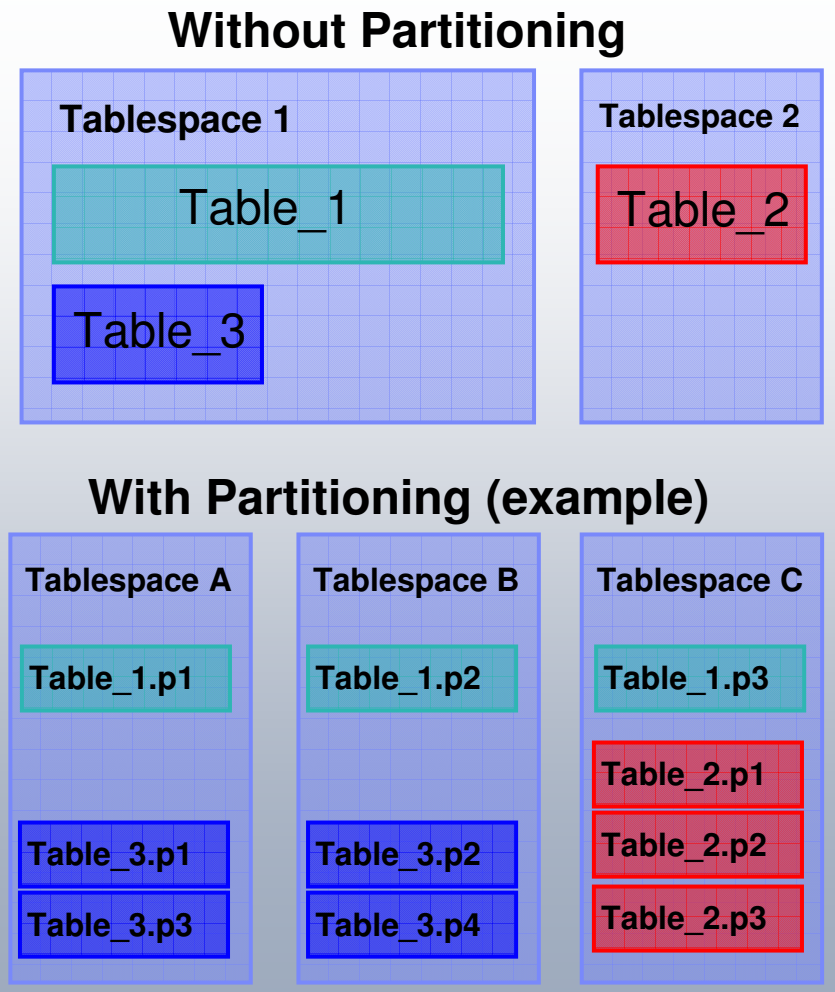


Table Partitioning : More Benefits

- More Benefits
  - Partition a single table across multiple tablespaces
  - SET INTEGRITY processing is now online
  - Indices of single partitioned tables can be placed in separate tablespaces and bufferpools



Terminology (Don't Let It Confuse You)

- DATABASE PARTITIONING
  - Distributing data by key hash across logical nodes of the database (aka DPF)
- DATABASE PARTITION
  - An individual "node" of a database that is using database partitioning (aka a DPF node)
- TABLE PARTITIONING *(new)*
  - **Splitting data by key range over multiple physical objects within a logical database partition**
- RANGE or DATA PARTITION *(new)*
  - **An individual "range" of a table partitioned using table partitioning**
  - **Represented by an object on disk**
- MULTI DIMENSIONAL CLUSTERING
  - Organizing data in table (or range of a table) by multiple key values (aka MDC)

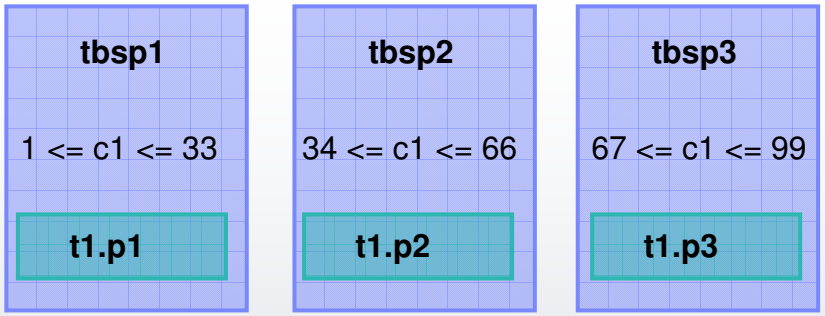
Database Partitioning, Table Partitioning and Multi Dimensional Clustering can be used simultaneously on the same table (more on this later)

How to Create Table Partitioning

Creating a Range Partitioned Table : Overview

- Short and long forms
- Partitioning column(s)
  - Must be base types (eg. No LOBS, LONG VARCHARS)
  - Can specify multiple columns
  - Can specify generated columns
  - Can specify tablespace using IN clause
- Notes
  - SQL0327N The row cannot be inserted because it is outside the bounds
  - Special values, MINVALUE, MAXVALUE can be used to specify open ended ranges, eg:

    CREATE TABLE t1 …
    (STARTING(MINVALUE)
    ENDING(MAXVALUE) …



Short Form

```
CREATE TABLE t1(c1 INT) IN tbsp1, tbsp2, tbsp3
   PARTITION BY RANGE (c1)
   (STARTING (1) ENDING (99) EVERY (33))
```

– or –

Long Form

```
CREATE TABLE t1(c1 INT)
   PARTITION BY RANGE (c1)
   (PARTITION p1 STARTING(1) ENDING(33) IN tbsp1,
       PART p2 ENDING(66)     IN tbsp2,
       PART p3 ENDING(99)    IN tbsp3)
```

Create Table : Inclusive and Exclusive Bounds

- **Use the EXCLUSIVE keyword to indicate range boundary is exclusive**
  - By default, bounds are **inclusive**
  - This examples avoid 'holes' by making each ending bound the same as the next starting bound, and using EXCLUSIVE for the ending bound

```
CREATE TABLE sales(sale_date DATE, customer INT)
PARTITION BY RANGE(sale_date)
       (STARTING MINVALUE ENDING '1/1/2000' EXCLUSIVE,
        STARTING '1/1/2000' ENDING '2/1/2000' EXCLUSIVE,
        STARTING '2/1/2000' ENDING '3/1/2000' EXCLUSIVE,
          …
        STARTING '11/1/2000' ENDING '12/1/2000' EXCLUSIVE,
        STARTING '12/1/2000' ENDING '12/31/2004');
```

Avoid Changing from 2/28 to 2/29 for leap year 2000

Create Table : NULL Handling

- **Use the NULLS FIRST/LAST keywords to specify what partition NULLs are placed in:**
  - Default is NULLS LAST: rows with NULL in the partitioning key column are placed in the range ending at MAXVALUE
  - Use NULLS FIRST to put them in the range starting with MINVALUE

```
…
PARTITION BY RANGE(sale_date NULLS FIRST)
…
```

Partitioning on Multiple Columns

- **Multiple columns can be specified in the PARTITION BY clause**
  - Analogous to multiple columns in an index key

```
CREATE TABLE sales(year INT, month INT)
PARTITION BY RANGE(year, month)
    (
    STARTING (2000, 1) ENDING (2000, 3),
    STARTING (2000, 4) ENDING (2000, 6),
    STARTING (2000, 7) ENDING (2000, 9),
    STARTING (2000, 10) ENDING (2000, 12),
    STARTING (2001,1) ENDING (2001,3)
    );
```

Multiple Columns Are NOT Multiple Dimensions

- **Ranges cannot overlap**
  - This example fails because the second range overlaps with the first
  - Use Multidimensional Clustering (MDC) if you need to define grids or cubes

```
CREATE TABLE neighborhoods(street INT, avenue INT)
PARTITION BY RANGE (street, avenue)
    (STARTING (1,1)  ENDING (10,10),
     STARTING (1,11) ENDING (10,20));
```

(1,1)
(1,5)
(1,10)
(1,11)
(5,1)
(5,10)
(10,1)
(10,10)
(10,20)
(15,1)
(15,10)

Storing Table Objects

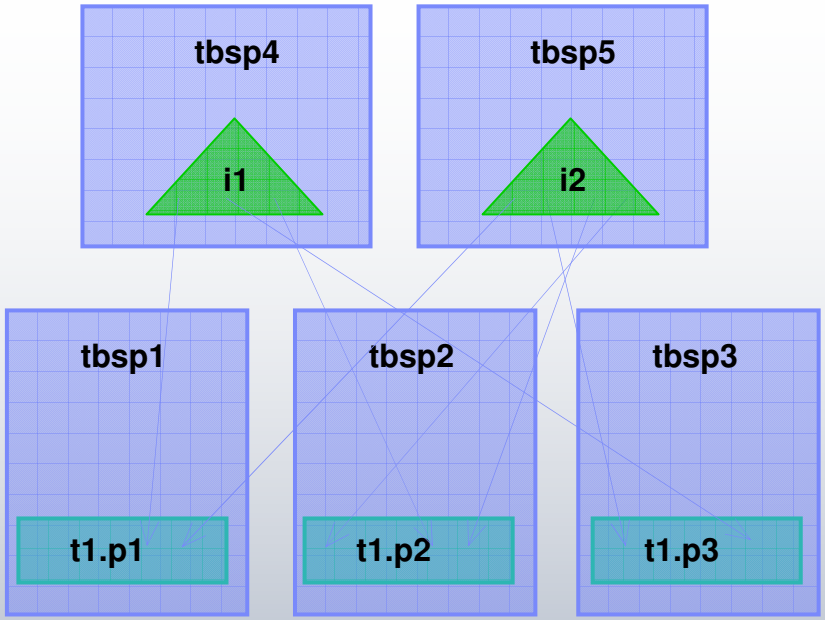
Storage Mapping : Indexes are ***Global*** in DB2 9

Indexes are ***global*** (in DB2 9)

- RIDs in index pages contain 2-byte partition ID
- Each index is in a separate storage object
  - By default, in the same tablespace as the first data partition
  - Can be created in different tablespaces, via
    - INDEX IN clause on CREATE TABLE (default is tablespace of first partition)
      - Note: INDEX IN clause works for MDC indexes ('block' indexes)
    - New IN clause on CREATE INDEX

Recommendation

- Place indexes in LARGE tablespaces



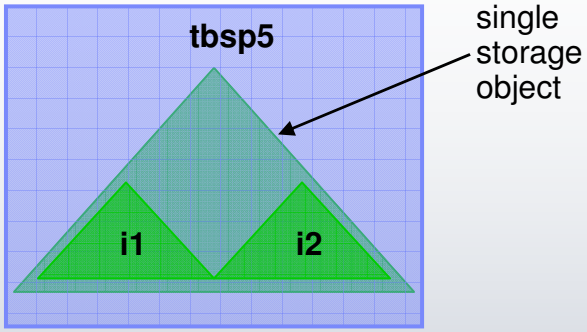

```
CREATE TABLE t1(c1 INT, c2 INT)
   IN tbsp1, tbsp2, tbsp3
   INDEX IN tbsp4
   PARTITION BY RANGE(c1)
   (STARTING FROM (1) ENDING (100)EVERY (33))

CREATE INDEX i1(c1)
CREATE INDEX i2(c2) IN tbsp5
```

Storage Mapping : Indexes

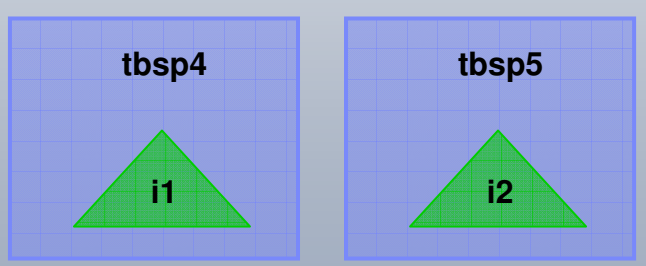
- Without table partitioning, all indexes for a particular table are stored in the *same* storage object*
  - SMS: If the table is in an SMS tablespace, this single index object is always in the same tablespace as the table
  - If the table is in a DMS or Auto Storage, this single index object can be in a different DMS (specified via INDEX IN table create key word)

Without Partitioning



- With table partitioning, each index is placed in it's own storage object*
  - Including MDC (aka block) indexes
  - Each index object can be placed in a different tablespace (regardless of tablespace type) via INDEX IN index create key word)

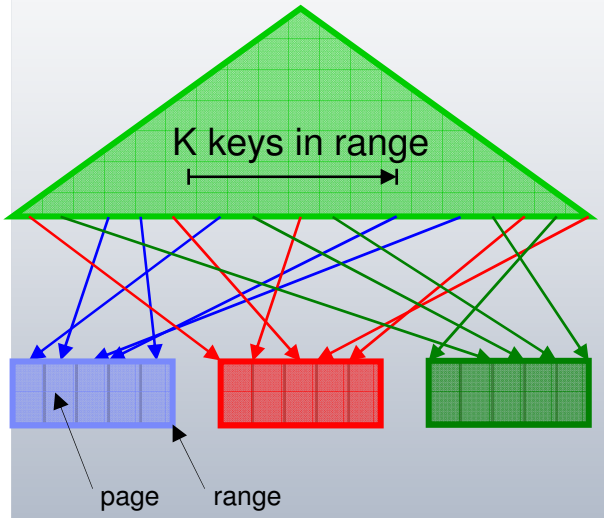
Possible with Partitioning



* Of course, in DPF there is a storage object in each database partition in the table's database partition group

Index Clustering

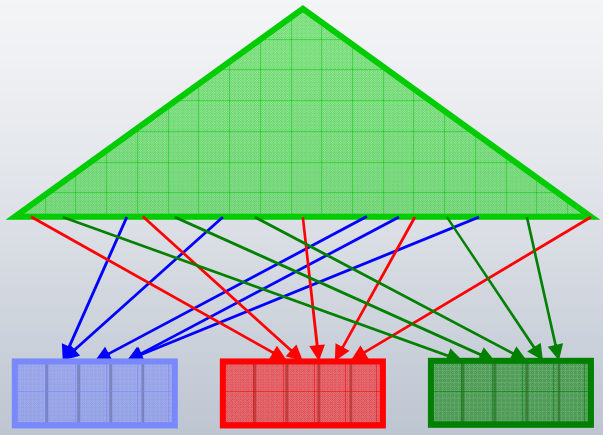
Not Clustered



Range scans may require a working set of K pages

K = # keys in range predicate

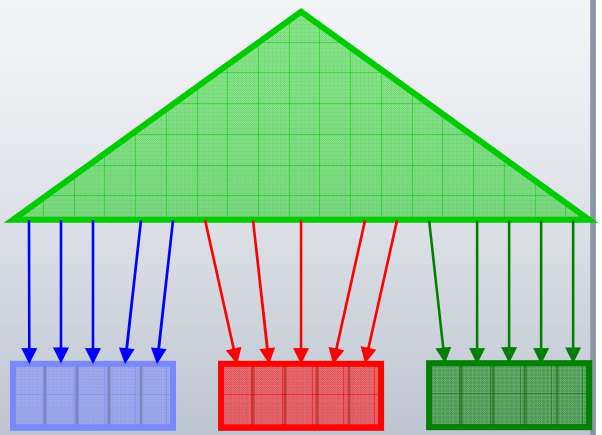
Clustering Doesn't Match Partitioning



Range scans may require a working set of 3 pages

3 = # range partitions

Clustering with Partition Key as Prefix



Range scans may require a working set of 1 page

Index Clustering

- **Cluster-*ed* Index : Index happens to be clustered (typically because you loaded the rows in key order)**
- **Cluster-*ing* Index : DB2 tries to preserve clustering order during inserts**
  - DB2 probes the clustering index to find what page an existing row with the same or similar key value exists on
  - If the keys found by the probe are on different partitions than the inserted key, DB2 can't use this information to preserve clustering
- **Recommendations**
  - Consider a form of clustering (having a clustering or defining a clustered index) if you have a lot of range scans over a particular key
  - Cluster key: Consider prefixing this key with the partitioning columns to maximize scan performance benefit and DB2's ability to key the index clustered eg:

    PARTITION BY RANGE (Month, Region)

    CREATE INDEX … (Month, Region, Department) CLUSTER

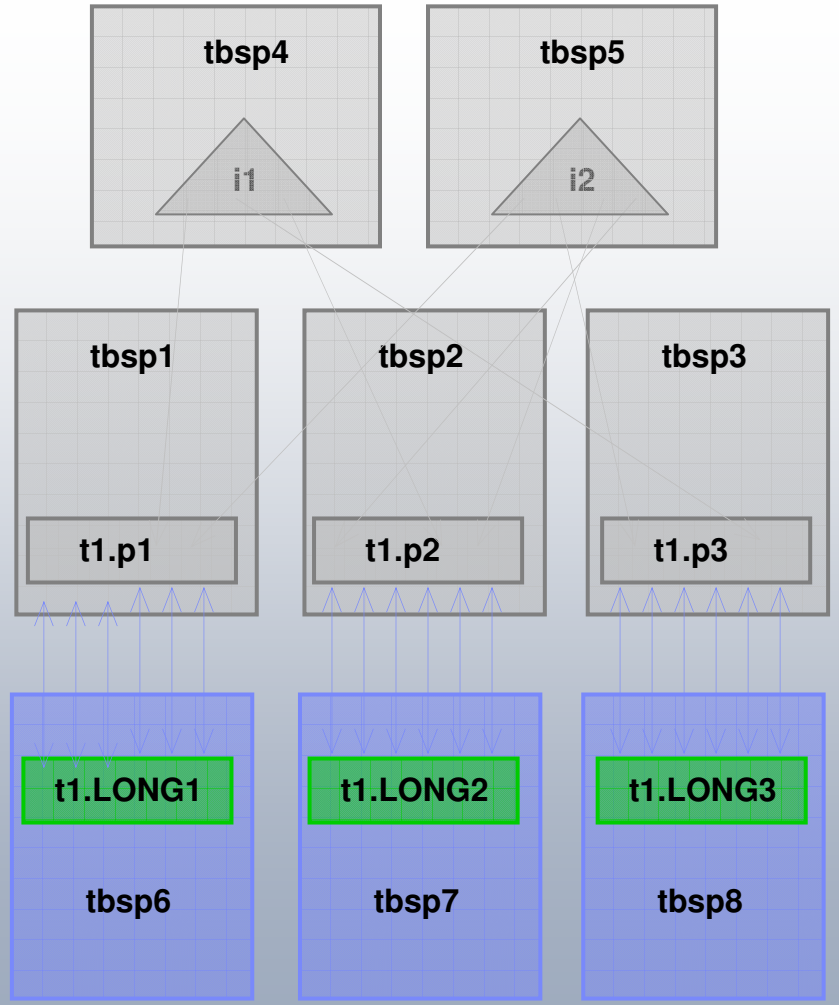
Tip

Storage Mapping : Large Objects are ***Local***

- Large objects (LOBs, etc) are ***local***
  - Separate storage object for each partition
  - By default in same partition as corresponding data partition
  - Can be specified per partition via LONG IN clause on CREATE TABLE

```
CREATE TABLE t1(c1 INT, c2 INT, c3 BLOB)
   IN tbsp1, tbsp2, tbsp3
   INDEX IN tbsp4
   LONG IN tbsp6, tbsp7, tbsp8
   PARTITION BY RANGE(c1)
   (STARTING FROM (1) ENDING (100)
EVERY (33))

CREATE INDEX i1(c1)
CREATE INDEX i2(c2) IN tbsp5
```

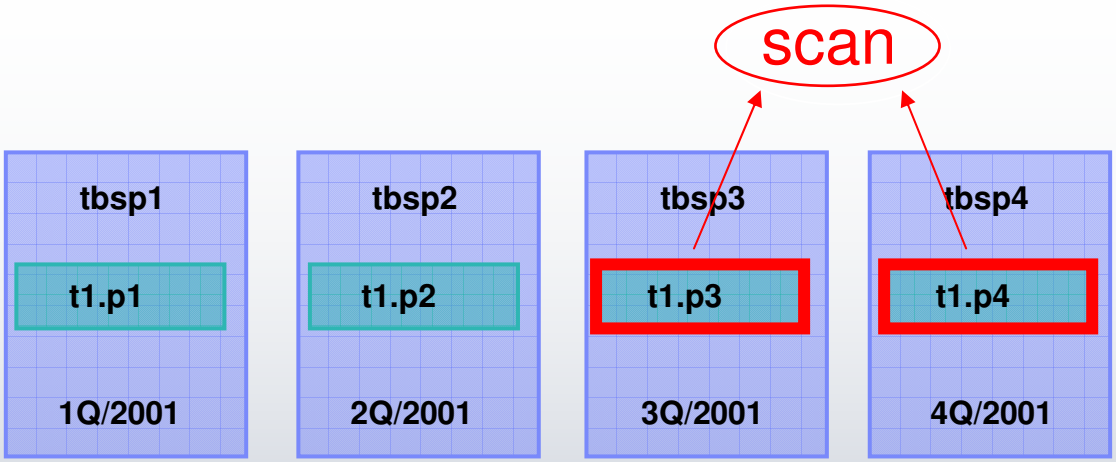


Accessing Partitioned Data

Partition Elimination : Table Scans

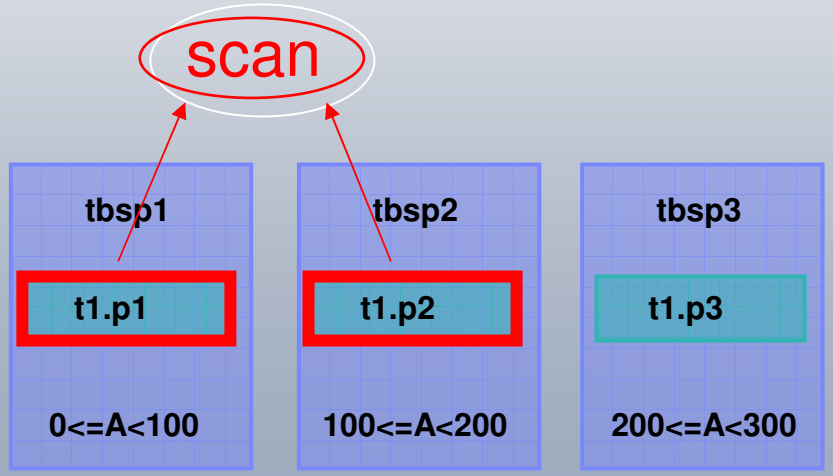
```
SELECT * FROM t1
  WHERE
  year = 2001 AND month > 7
```

- Will only access data in tablespace tbsp3 and tbsp4




```
SELECT * FROM t1
   WHERE
   A>50 AND A<150
```

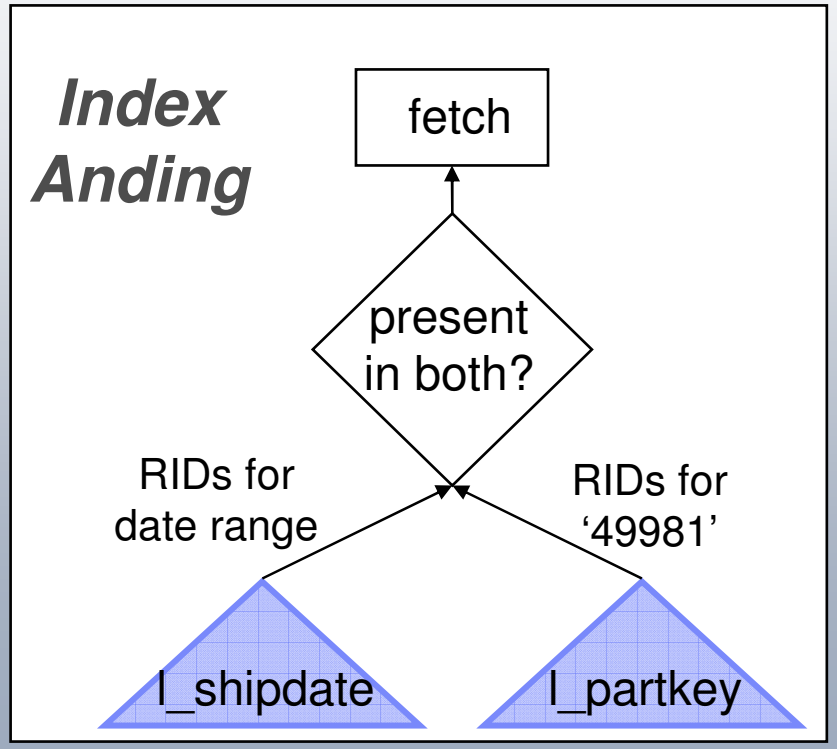
- Will only access data in tbsp1 and tbsp2

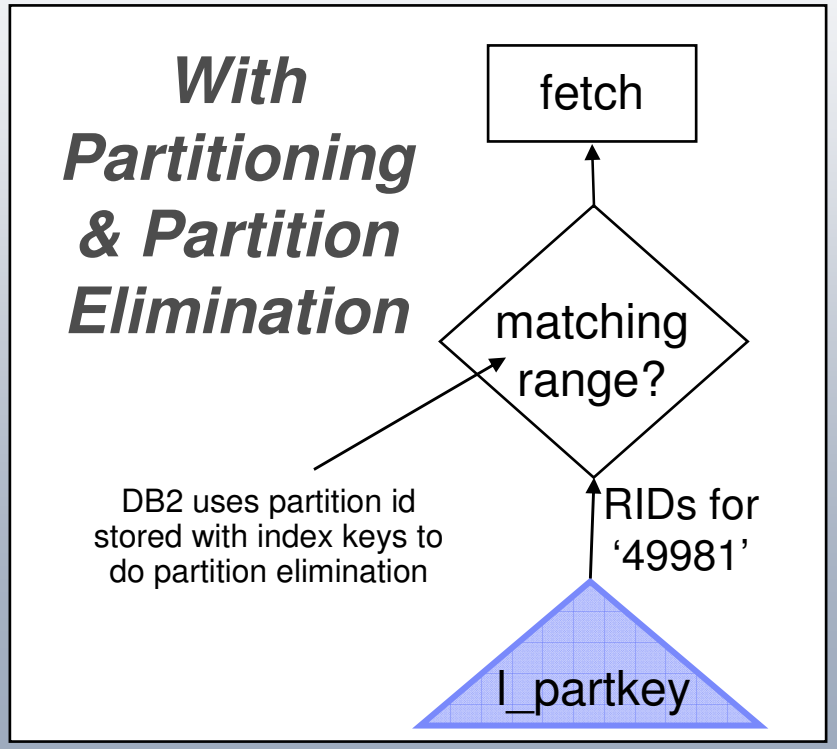


Partition Elimination : Index Scan

- **Scan involving two indexes:**

```
SELECT l_shipdate, l_partkey, l_returnflag
    FROM lineitem
    WHERE l_shipdate BETWEEN `01/01/1993' and '03/31/1993`
    AND l_partkey=49981
```





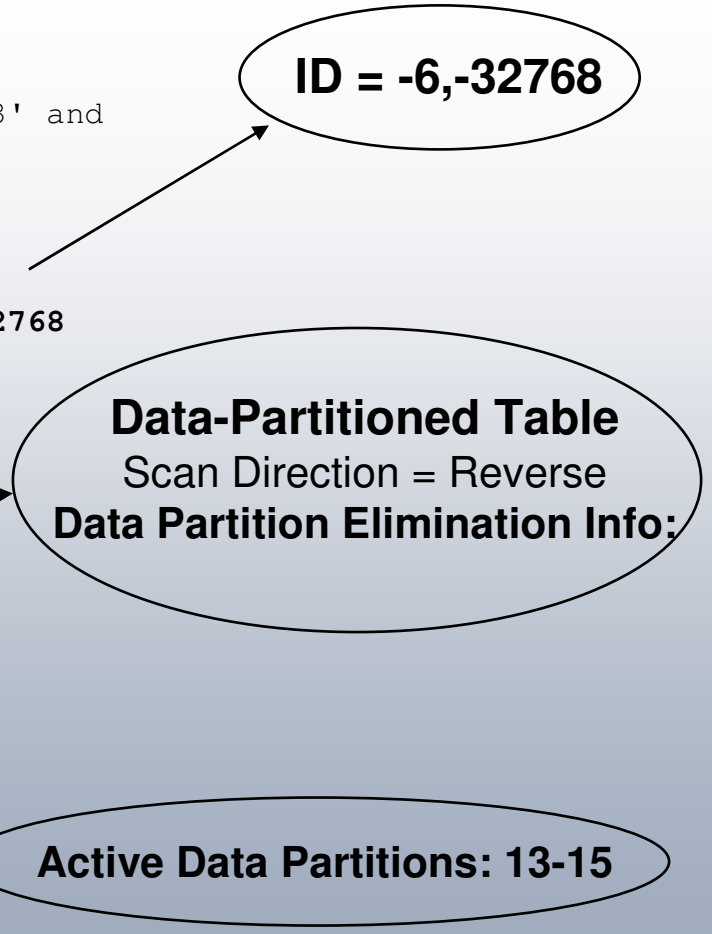
Partition Elimination Shown in DB2 Explain (1 of 2)

```
Statement:

  select l_shipdate, l_partkey, l_returnflag
  from lineitem
  where l_shipdate between '01/01/1993' and '03/31/1993' and
          l_partkey=49981

Estimated Cost = 50.485638
Estimated Cardinality = 2.126619

(    2) Access Table Name = KBECK.LINEITEM  ID = -6,-32768
        |  Index Scan:  Name = KBECK.LI_PK  ID = 2
        |  |  Regular Index (Not Clustered)
        |  |  Index Columns:
        |  |  |  1: L_PARTKEY (Ascending)
        |  #Columns = 2
        |  Data-Partitioned Table
        |  Scan Direction = Reverse
        |  Data Partition Elimination Info:
        |  |  Range 1:
        |  |  |  #Key Columns = 1
        |  |  |  |  Start Key: Inclusive Value
        |  |  |  |  |  1: 1993-01-01
        |  |  |  |  Stop Key: Inclusive Value
        |  |  |  |  |  1: 1993-03-31
        |  Active Data Partitions: 13-15
        |  #Key Columns = 1
        |  |  Start Key: Inclusive Value
        |  |  |  |  1: 49981
```



Partition Elimination Shown in DB2 Explain (2 of 2)

```
        |  |  Regular Index (Not Clustered)
        |  |  Index Columns:
        |  |  |  1: L_PARTKEY (Ascending)
        |  #Columns = 2
        |  Data-Partitioned Table
        |  Scan Direction = Reverse
        |  Data Partition Elimination Info:
        |  |  Range 1:
        |  |  |  #Key Columns = 1
        |  |  |  |  Start Key: Inclusive Value
        |  |  |  |  |  1: 1993-01-01
        |  |  |  |  Stop Key: Inclusive Value
        |  |  |  |  |  1: 1993-03-31
        |  Active Data Partitions: 13-15
        |  #Key Columns = 1
        |  |  Start Key: Inclusive Value
        |  |  |  |  1: 49981
        |  |  Stop Key: Inclusive Value
        |  |  |  |  1: 49981
        |  Data Prefetch: None
        |  Index Prefetch: None
        |  Lock Intents
        |  |  Table: Intent Share
        |  |  Row  : Next Key Share
        |  Sargable Predicate(s)
        |  |  #Predicates = 2
(    1) |  |  Return Data to Application
        |  |  |  #Columns = 3
(    1) Return Data Completion
End of section
```


```
Data Partition Elimination Info:
  |  Range 1:
  |  |  #Key Columns = 1
  |  |  |  Start Key: Inclusive Value
  |  |  |  |  1: 1993-01-01
  |  |  |  Stop Key: Inclusive Value
  |  |  |  |  1: 1993-03-31
```

New Operations for Roll-Out and Roll-In

- **ALTER TABLE … DETACH**
  - An existing range is split off as a stand alone table
  - Data instantly becomes invisible
  - Minimal interruption to other queries accessing table
- **ALTER TABLE … ATTACH**
  - Incorporates an existing table as a new range
  - Follow with SET INTEGRITY to validate data and maintain indexes
  - Data becomes visible all at once after COMMIT
  - Minimal interruption to other queries accessing table
- **Key points**
  - No data movement
  - Nearly instantaneous
  - SET INTEGRITY is now online

Roll-Out Scenario

Typical Roll-Out Scenario (today)

```
CREATE TABLE sales_old …
INSERT INTO sales_old (SELECT * FROM sales WHERE …);
DELETE FROM sales WHERE ….
```

- Slow, error prone
- What will queries show while DELETE is in progress?
  - Different sets of results, possibility of deadlocks
- Also possible to use UNION ALL views

Roll-Out Scenario (with Table Partitioning)

`ALTER TABLE Big_Table DETACH PARTITION p3 INTO TABLE OldMonthSales`

- Queries are drained and table locked
- Very fast operation
- No data movement required
- Index maintenance done later (asynchronously in background)
- Dependent MQT's go offline

`COMMIT`

- Detached data now invisible
- Detached partition ignored in index scans
- Rest of Big_Table available
- Index maintenance is kicked off

`SET INTEGRITY FOR Mqt1,Mqt2 FULL ACCESS`

- (Optional) maintains MQTs on Big_Table

`EXPORT OldMonthSales; DROP OldMonthSales`

- (Optional) this becomes a standalone table that you can do whatever you want with

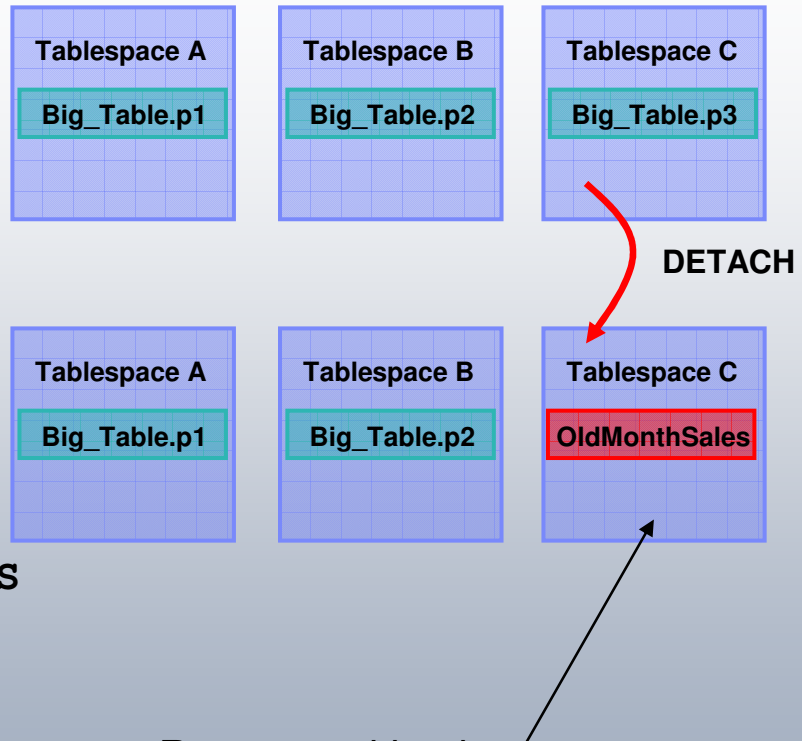
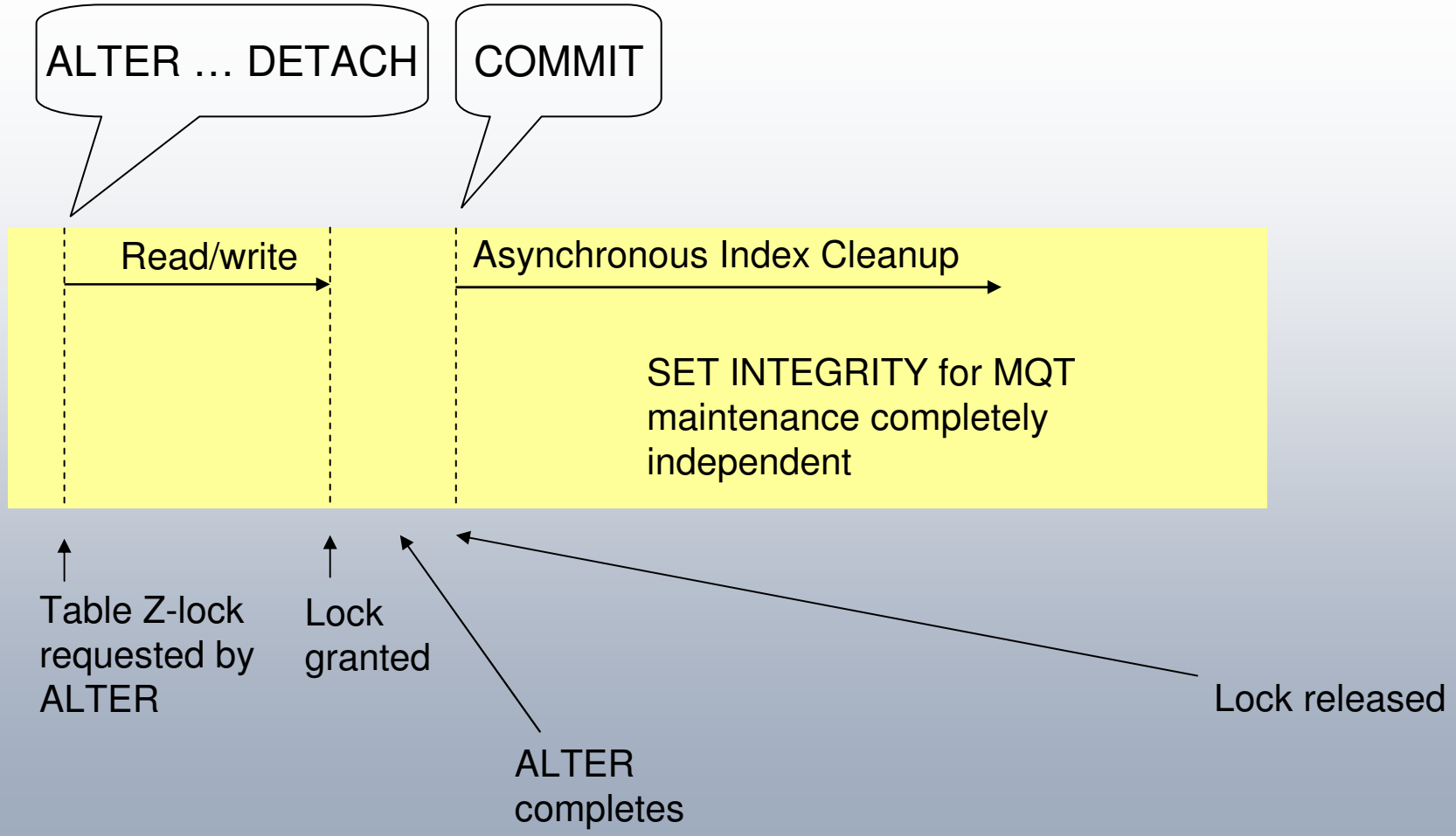


Table Availability During Roll-Out



Roll-In Scenario

Typical Roll-In Scenario (today)

- **Data in a single table**
  - Extract data from operational data store
  - Do data cleansing/transformation
  - Load into table
  - Use SET INTEGRITY to check RI constraints, maintain MQTs
- **Using UNION ALL view**
  - Extract/transform/load into a new table
  - Drop and recreate the view to incorporate new data
  - SET INTEGRITY for constraints, MQTs

Roll-In Summary

CREATE TABLE NewMonth

- Create empty staging table

LOAD / Insert into NewMonthSales

(…perform ETL on NewMonthSales)

ALTER TABLE Big_Table ATTACH PARTITION
STARTING '03/01/2005'ENDING '03/31/2005'
FROM TABLE NewMonthSales

- Very fast operation
- No data movement required
- Index maintenance done later

COMMIT

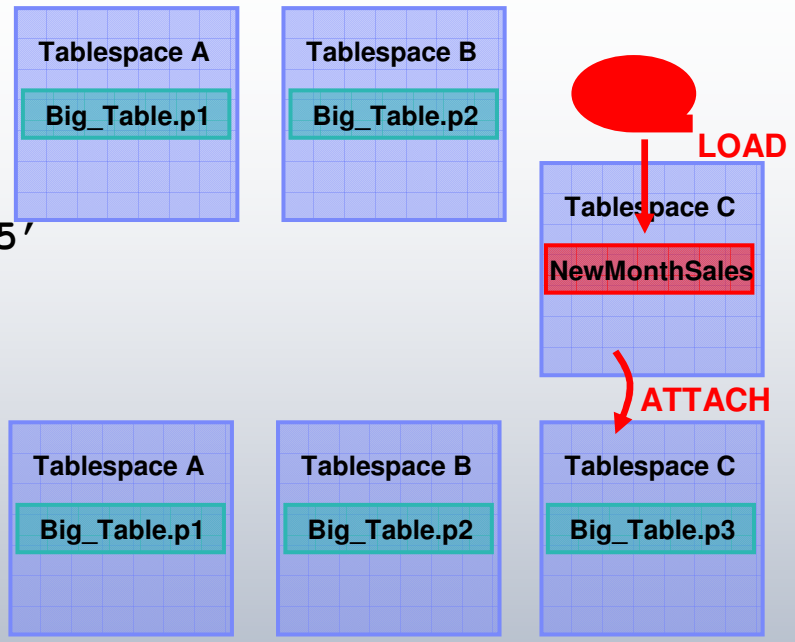
- New data still not visible

SET INTEGRITY FOR Big_Table ……

- Potentially long running operation
  - Validates data
  - Maintains global indexes, MQTs
- Existing data available while it runs

COMMIT

- New data visible



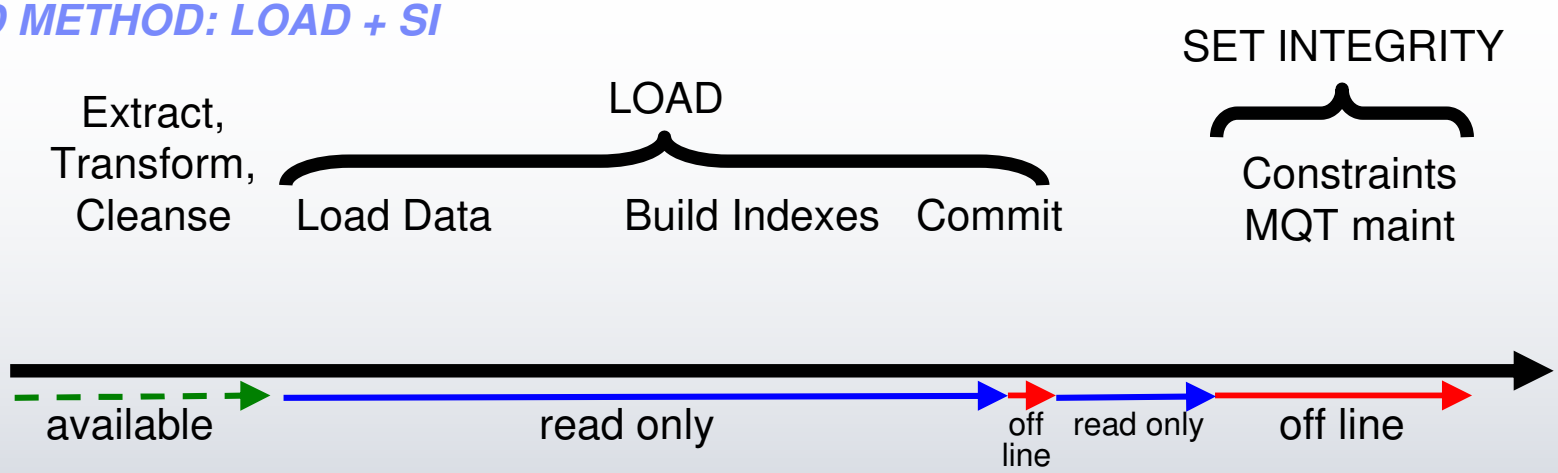
Use SET INTEGRITY to Complete the Roll-in

```
SET INTEGRITY FOR sales ALLOW WRITE ACCESS,
       sales_by_region ALLOW WRITE ACCESS
           IMMEDIATE CHECKED INCREMENTAL
           FOR EXCEPTION IN sales USE sales_ex;
```

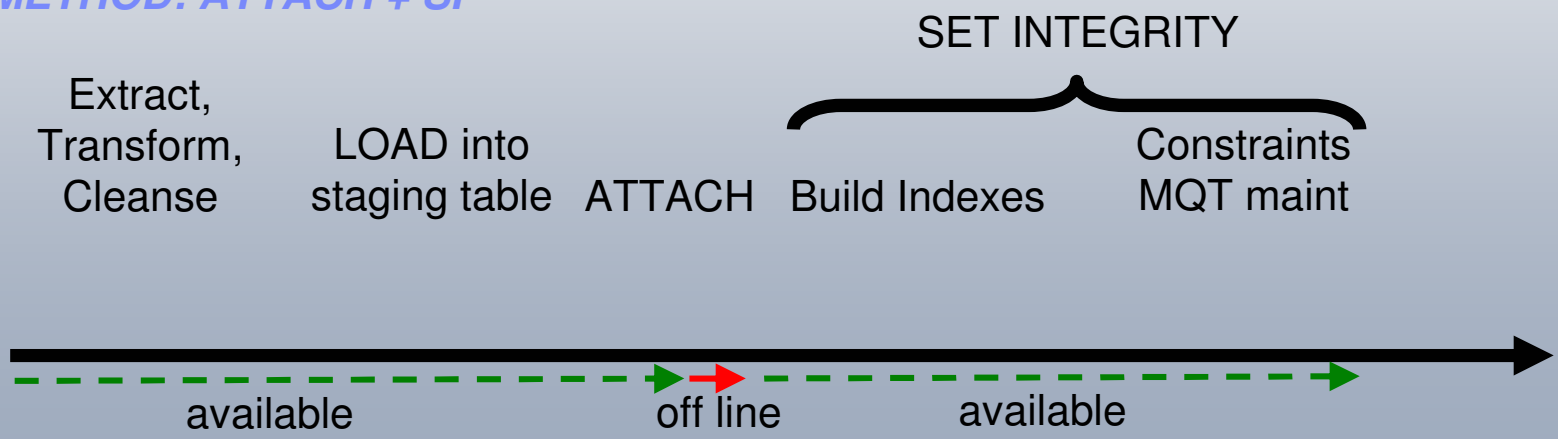
- **SET INTEGRITY does**
  - Index maintenance
  - Checking of range and other constraints
  - MQT maintenance
  - Generated column maintenance
- **Table is online through out process except for ATTACH**
- **New data becomes visible at end of SET INTEGRITY**
- **Specify ALLOW WRITE ACCESS the default is old behaviour**
  - Also available: ALLOW READ ACCESS
- **Use an exception table, any violation will fail the entire operation**

Comparison of Table Availability

OLD METHOD: LOAD + SI



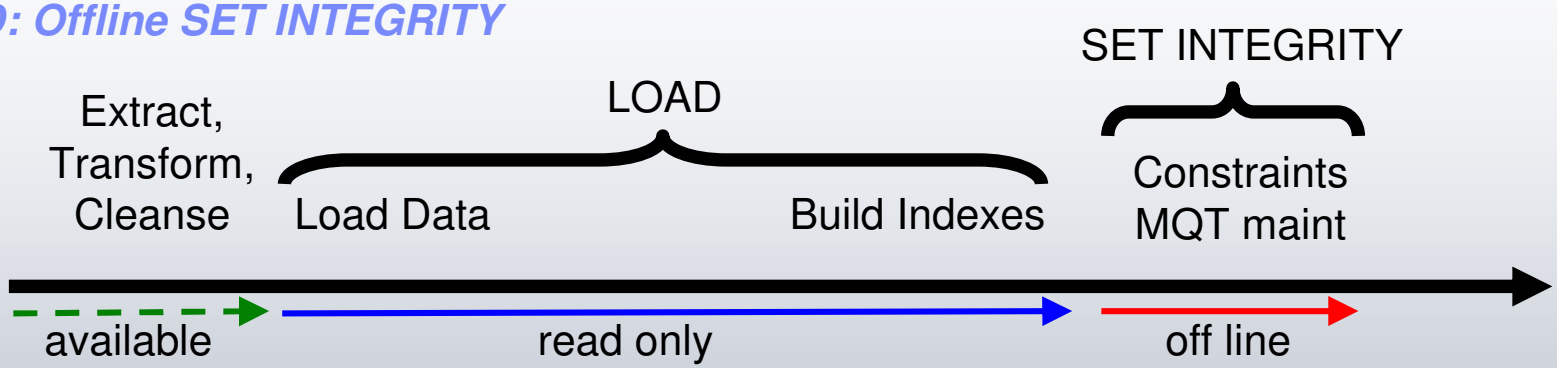
NEW METHOD: ATTACH + SI



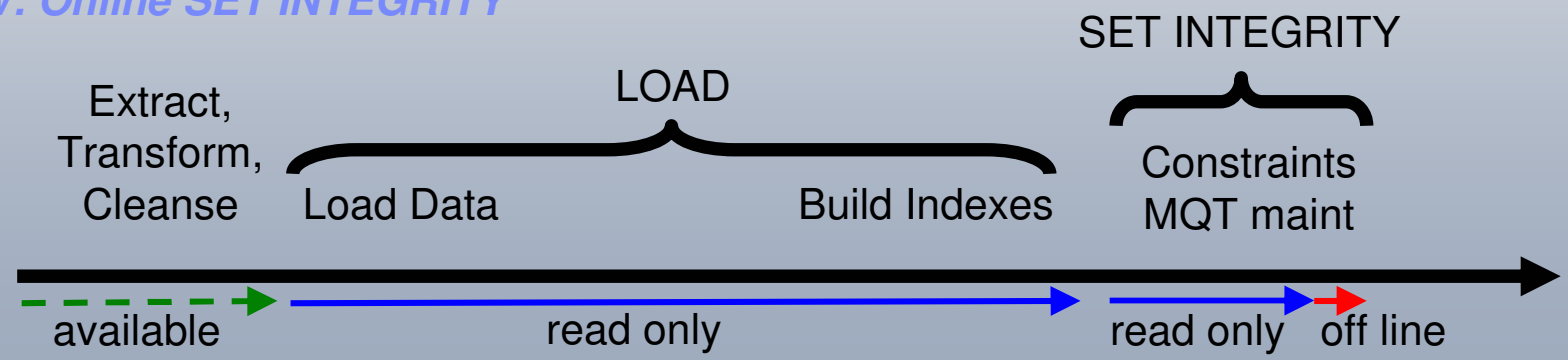
Online SET INTEGRITY Without Table Partitioning

- **The online SET INTEGRITY also works with non-partitioned tables**
- **SET INTEGRITY after LOAD can now be read access until the end**

OLD: Offline SET INTEGRITY



NEW: Online SET INTEGRITY



Putting it all together

Multidimensional Clustering with Table Partitioning and DPF

```
CREATE TABLE my_hybrid
     (A INT, B INT, C Date, D INT …)
     IN Tablespace A, Tablespace B, Tablespace C
     INDEX IN Tablespace B
     DISTRIBUTE BY HASH (A)
     PARTITION BY RANGE (B) (STARTING FROM (100) ENDING (300) EVERY (100))
     ORGANIZE BY DIMENSIONS (A,B,C)
```

Quiz: What isn't shown correctly in this picture?

Data blocks without MDC

A	B	C	D
101	21	04/02	1
101	21	04/02	1
101	7	04/02	2

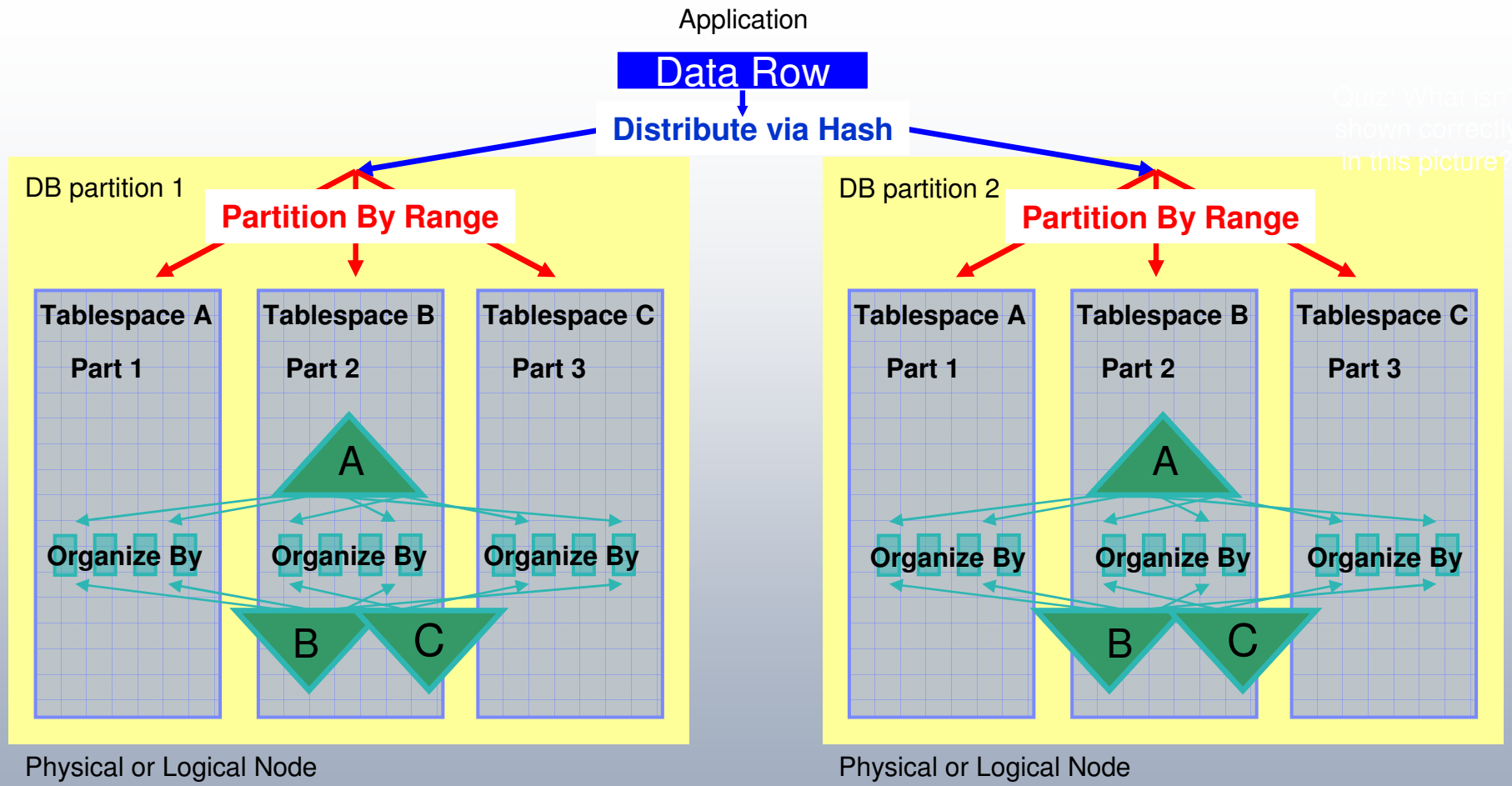
A	B	C	D
101	7	04/02	1
101	21	04/02	3

Data blocks with MDC

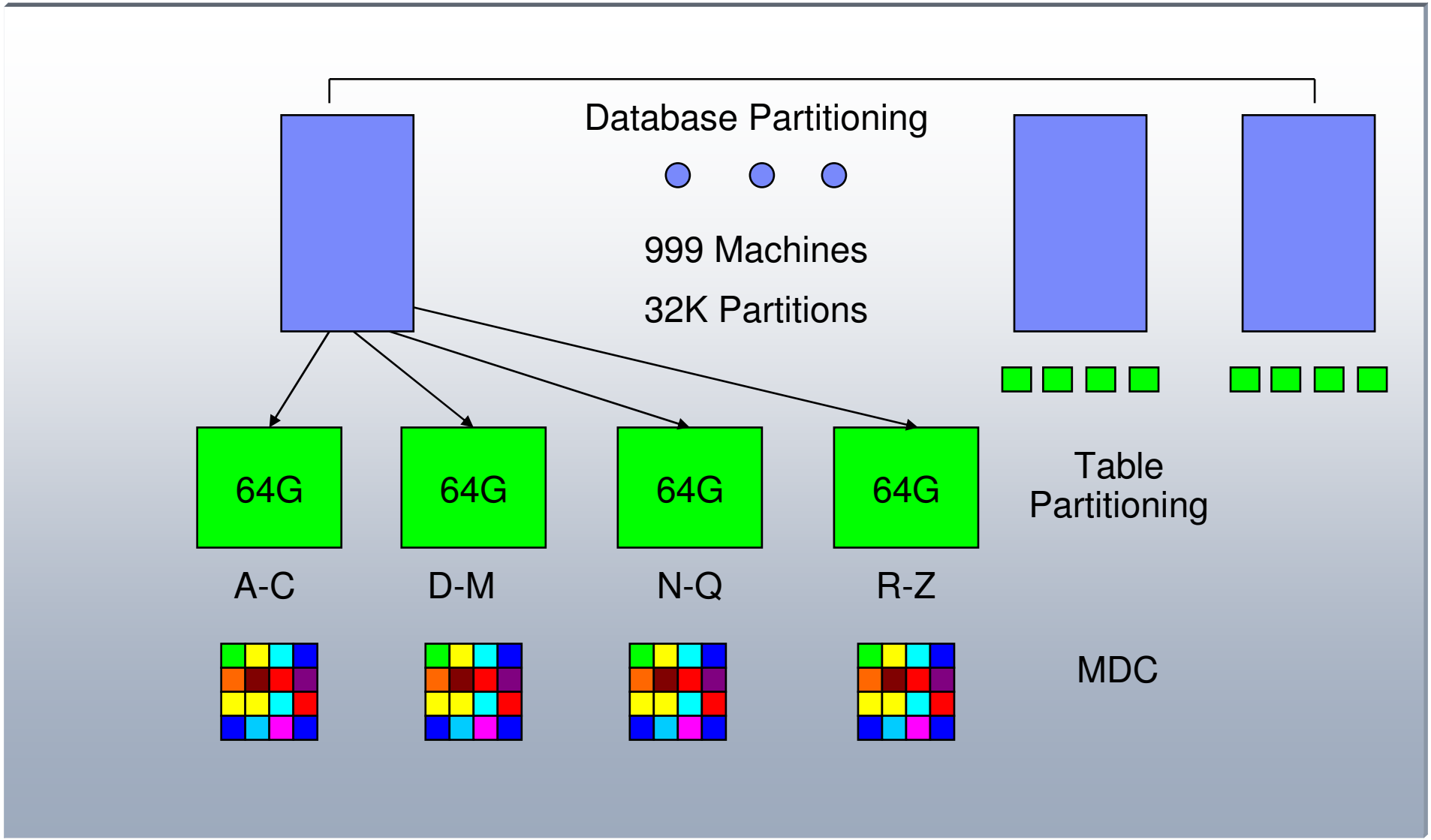
A	B	C	D
101	21	04/02	1
101	21	04/02	1
101	21	04/02	3

A	B	C	D
101	7	04/02	1
101	7	04/02	2

Table Partitioning with DPF and MDC



Hybrid Partitioning



IBM Software Group

IBM Information Management software

Data Management – DB2 9.5

Winter/Spring 2008

Data Management Emerging Partnerships and Technologies
IBM Toronto Lab

IBM Information Management software

DB2 Partitioning

Data Management Emerging Partnerships and Technologies
IBM Toronto Lab