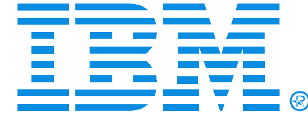


Z26: DB2 for z/OS

Major Index Changes in DB2 Version 8

James Teng (jteng@us.ibm.com)

IBM DB2 Information Management
Technical Conference

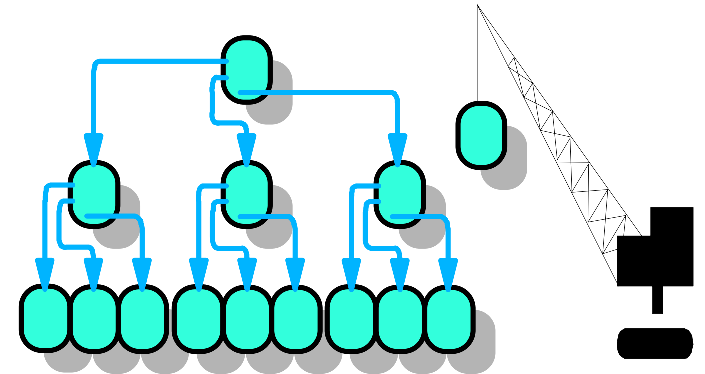
Sept. 20-24, 2004

Las Vegas, NV

© IBM Corporation 2004

Agenda

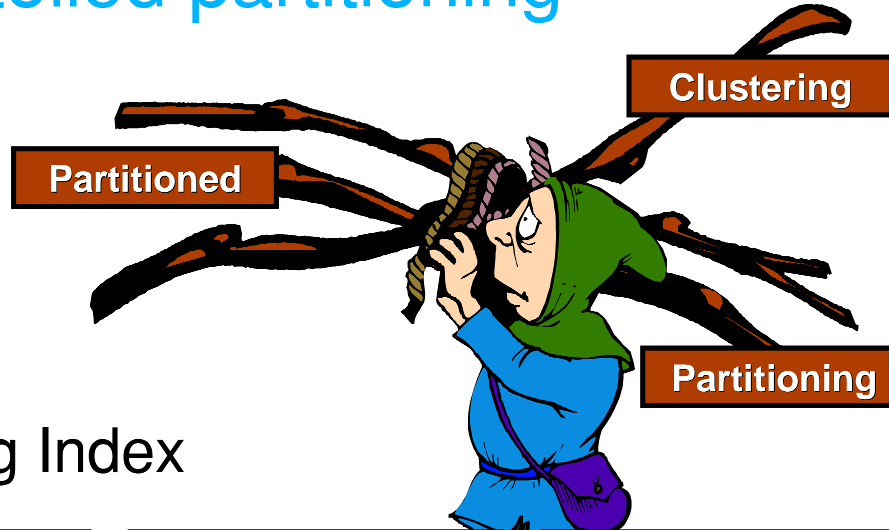
- Index Partitioning enhancements



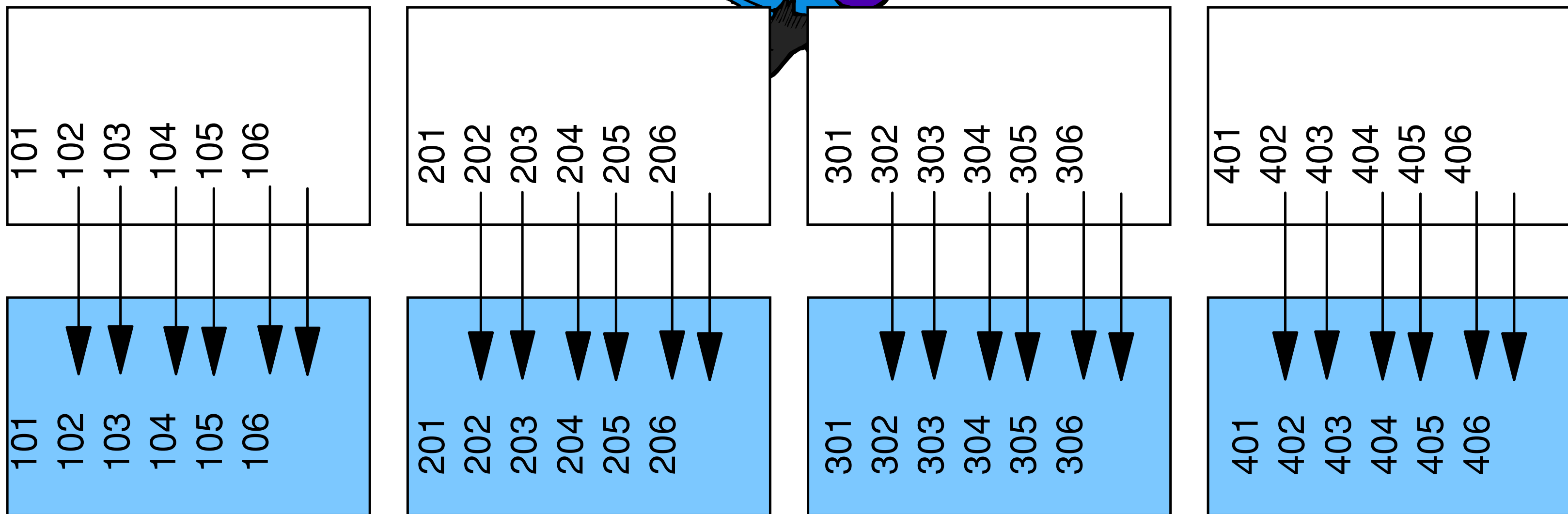
- Variable length index keys
- Long index keys & backward index scan
- Schema changes with an emphasis on altering indexes
- Predicates indexable for unlike types
- Summary

© 2004 IBM Corporation

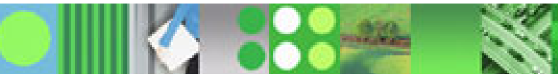
V7 - Index-contolled partitioning



Partitioning Index



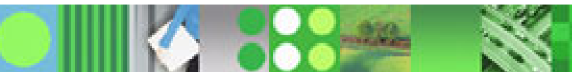
Partitioned Table



© 2004 IBM Corporation

Index Types Naming Convention

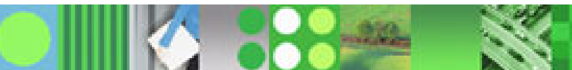
- Partitioned Index
  - Index with multiple physical partitions
  - Can only be created on partitioned tables
  - #parts (index) = #parts(table)
  - In V8, a single table can have many partitioned indexes
  - Each partition has its own index B+ tree
  - Allows index REORG at PART level
- Non-partitioned Index
  - Index does not have partitions
  - Applies to both partitioned and non-partitoined tables
  - A single B+ tree for the entire index
  - Can consist of multiple data sets (controlled by PIECESIZE)



© 2004 IBM Corporation

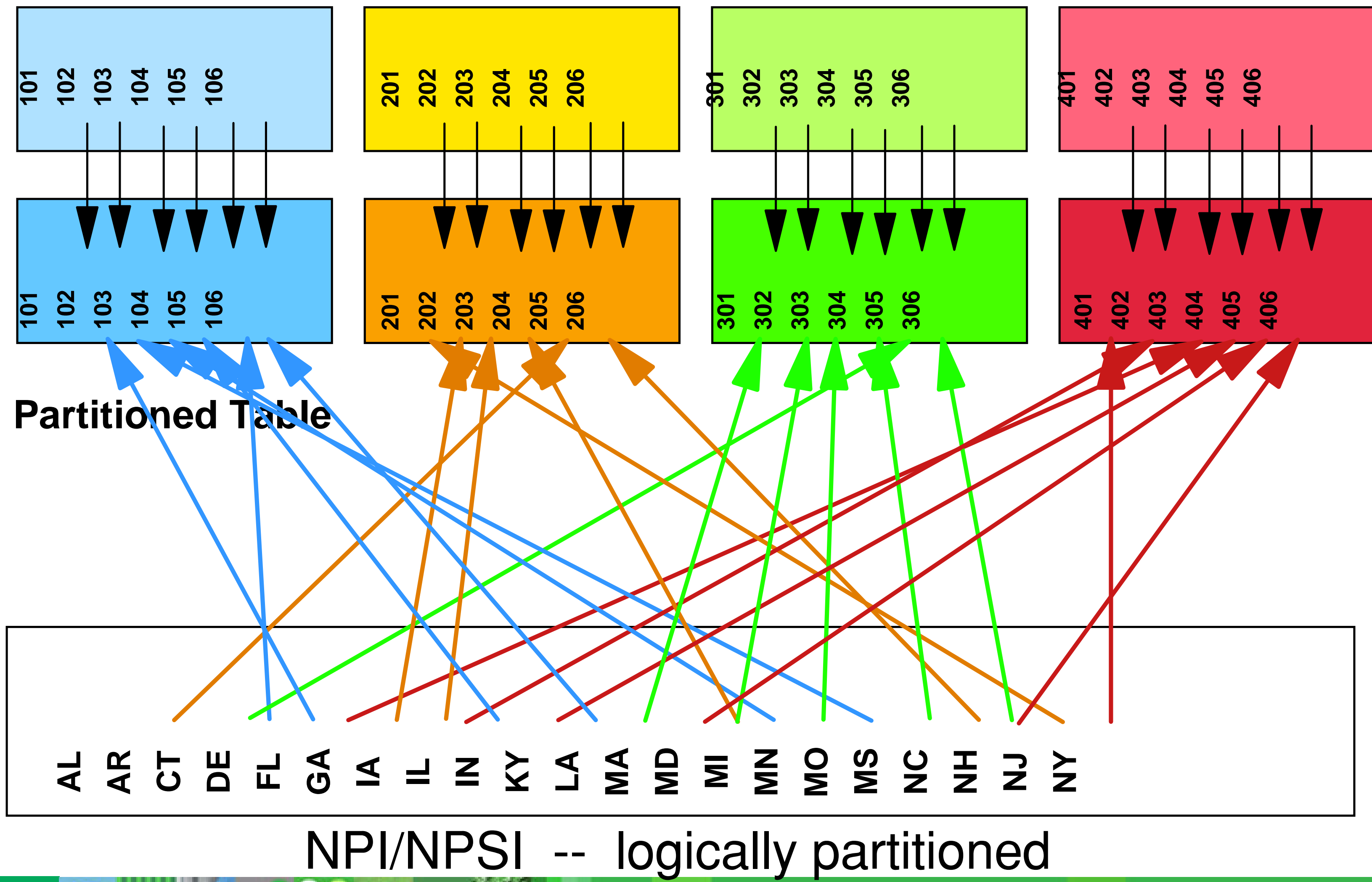
Index Types Naming Convention ...

- Partitioning Index
  - Index key columns match the partitioning key for the table
  - Same columns, same order, same collating sequence
  - Index can be a superset of partition key columns
  - A partitioned table can have many partitioning indexes in V8
  - Not necessary a partitioned index
  - In V7, partitioning index = partitioned index
- Secondary Index
  - Index key is not prefixed with the key used for table-controlled partitioning
  - Partitioned - Data Partitioned Secondary Index (DPSI)
  - Non-partitioned - Non-Partitioned Secondary Index (NPSI/NPI)



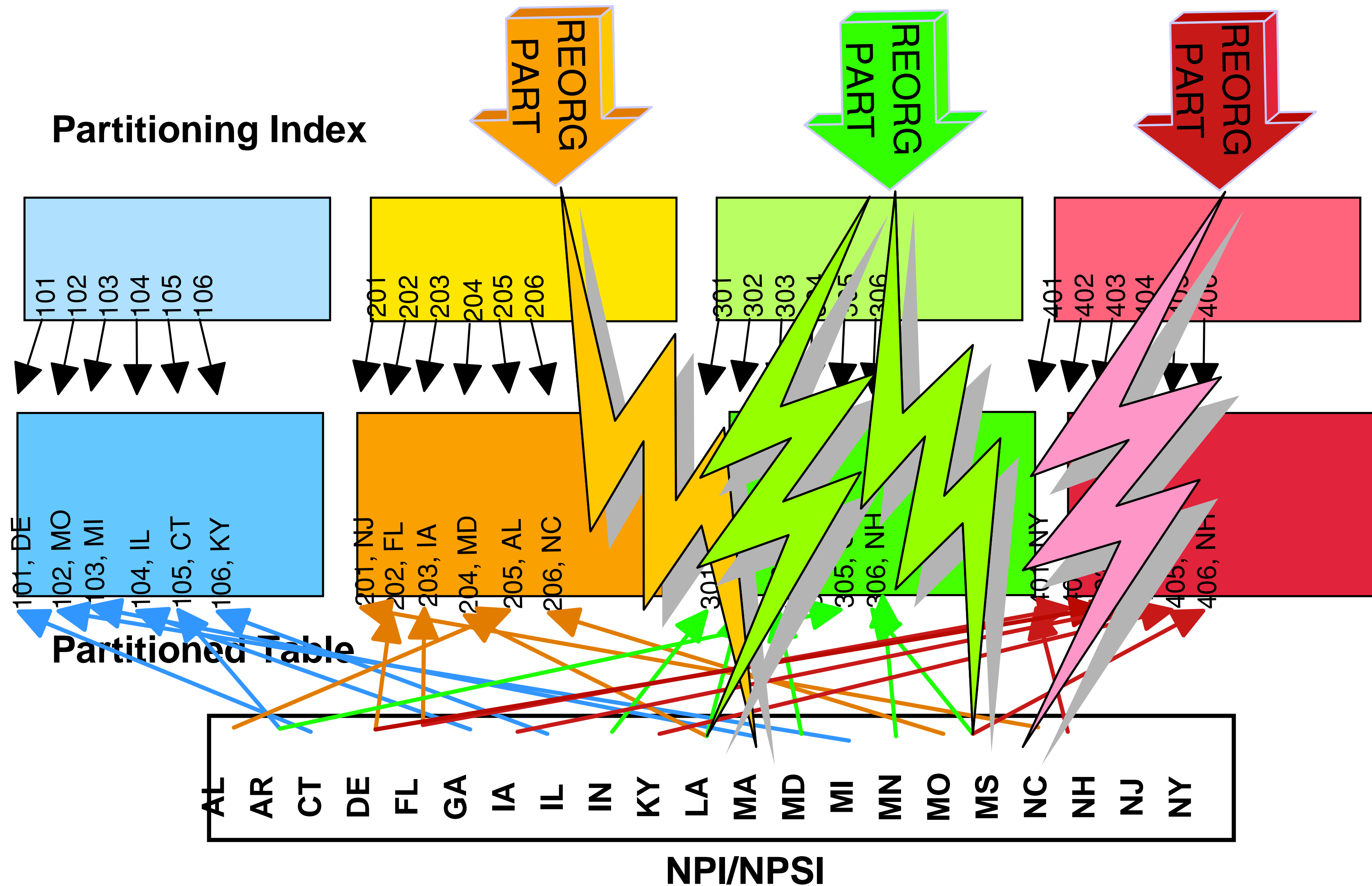
© 2004 IBM Corporation

V7 PI and NPI/NPSI



© 2004 IBM Corporation

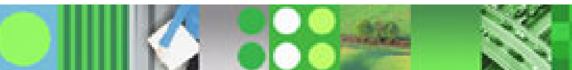
V7 Utility interactions - contention on NPI/NPSI



© 2004 IBM Corporation

NPSI/NPI challenges

- Recovery must be done at the entire index level
- Unable to support data affinity in data sharing
- Partition-level operations (add, rotate, load replace), are less clean with secondary indexes
- Difficulties with utility operations
  - BUILD2 phase of REORG PART blocks queries from operating
  - LOAD PART jobs that target different partitions of the table space have locking contention between keys in the NPI
  - Processing of keys in non-partitioned indexes requires insert logic, and is unable to use more efficient append-mode logic
  - NPIs over large partitioned tables makes their management as a single object difficult



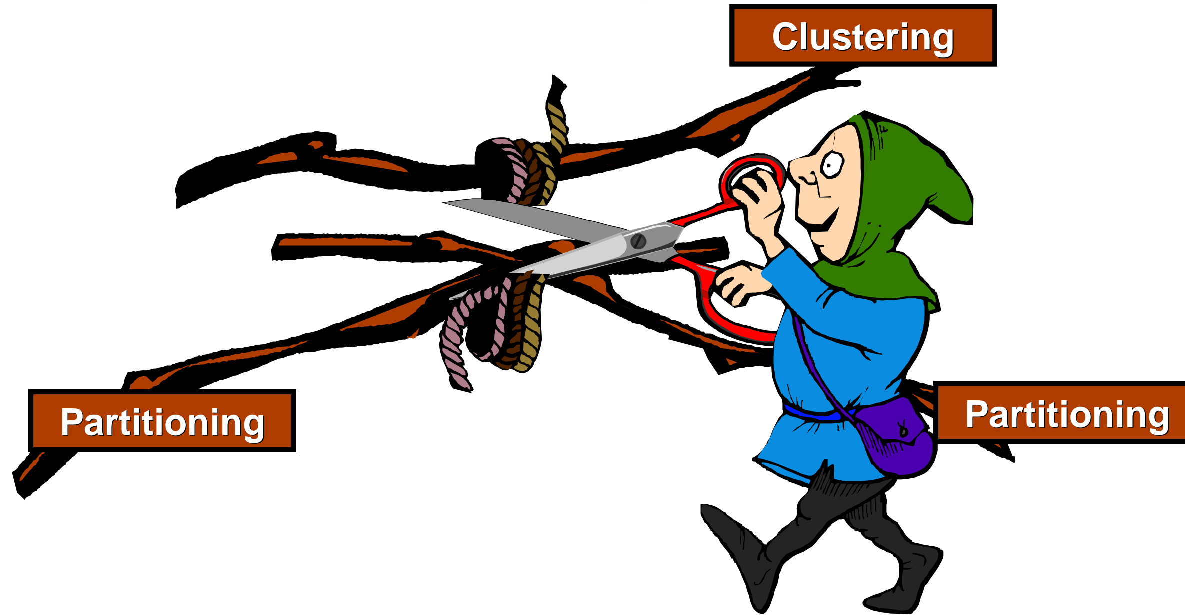
© 2004 IBM Corporation

V8 design points

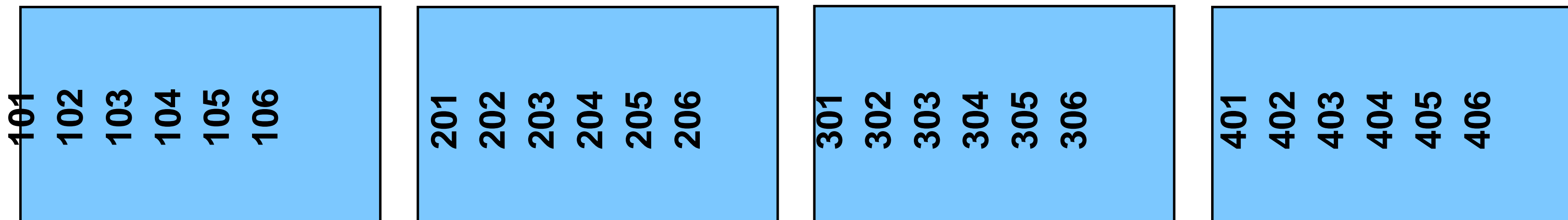
- Improve recovery characteristics of secondary indexes
- Reduce data sharing overhead for secondary indexes
- Facilitate partition-level operations (add, rotate PART)
- Improve utility operations
  - Eliminate REORG BUILD2 phase
  - Eliminate contention between LOAD PART jobs that target different partitions of the table space
  - Enable more efficient append-mode insertion of data

© 2004 IBM Corporation

V8 - table-controlled partitioning



No indexes are required for partitioning!!

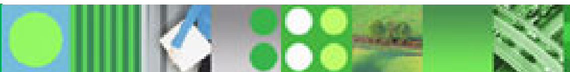


Partitioned table

© 2004 IBM Corporation

Converting to table controlled partitioning

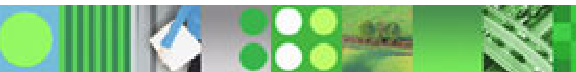
- Not necessary to drop and recreate the table
- Start with an existing table with a single PI
- Use any new function and a conversion is triggered from index-controlled to table-controlled partitioning:
  - Drop the partitioning index (partitioning by table)
  - ALTER partitioning index NOT CLUSTER
  - Create index PARTITIONED (DPSI or PI)
  - Add a partition
  - Rotate partitions
  - Create INDEX VALUES but no CLUSTER keyword



© 2004 IBM Corporation

Data Partitioned Secondary Indexes (DPSIs)

- A new V8 index type
  - Physical partitions like table
  - DPSI = physically partitioned secondary index
  - #parts(DPSI) = #parts(table)
  - Keys in part 'n' of DPSI refer only to rows in part 'n' of table
- 3 kinds of indexes now:
  - Partitioning Index (PI).
    - As today, except optional in V8 and may or may not be partitioned
  - New Data Partitioned Secondary Index (DPSI).
  - Non Partitioned Secondary Index (NPSI) As today's NPI



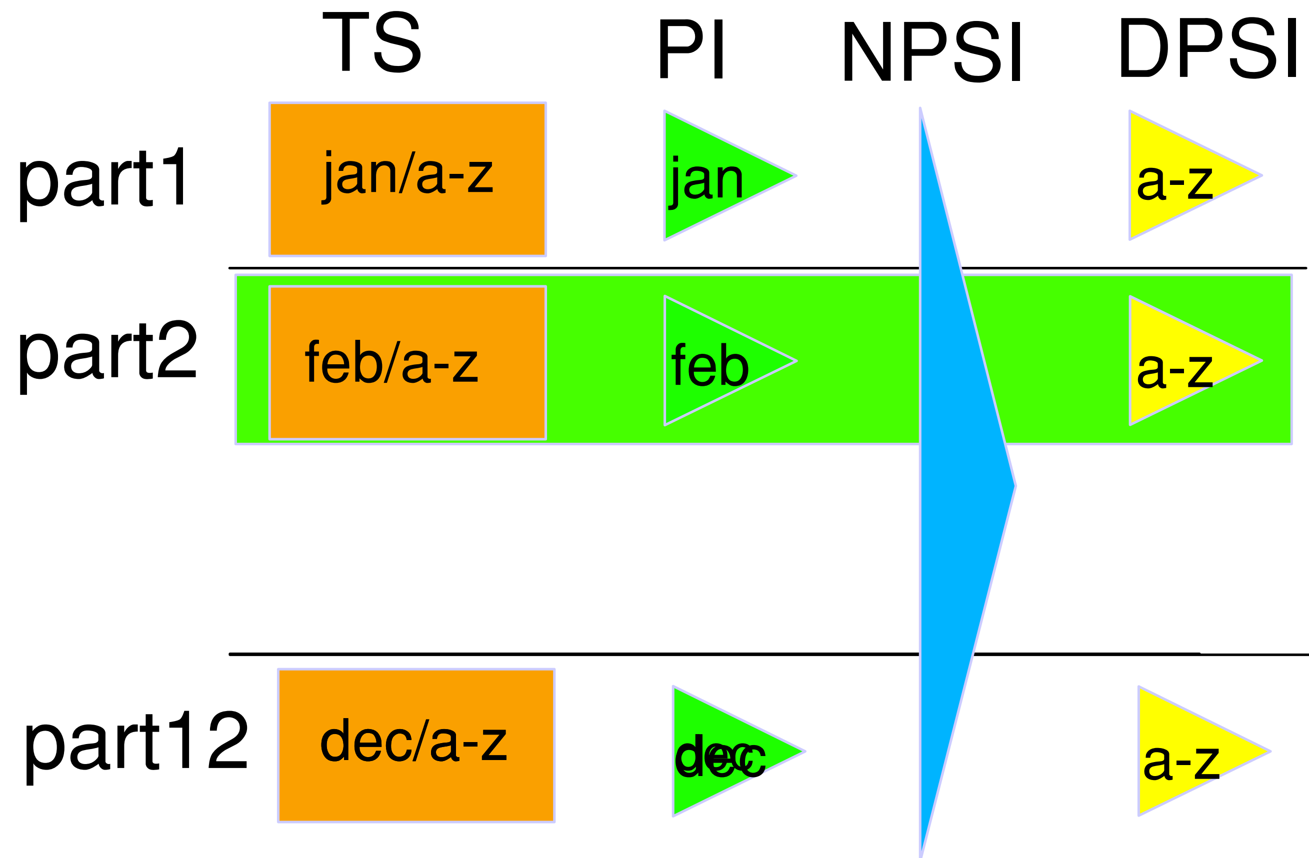
© 2004 IBM Corporation

Data Partitioned Secondary Indexes (DPSIs)

- Benefits:
  - Full Partition Independence
  - Eliminate REORG BUILD2 phase
  - Eliminate LOAD PART contention
  - Parallel Utilities (REORG, LOAD, RECOVE)
  - Partition scope operations (add, rotate, reset)
  - Data affinity in data sharing
  - Potentially more efficient partitions pruning
- Potential impact to query performance
  - Many partitions to search if partition key is not specified
  - Not allowed for unique index

© 2004 IBM Corporation

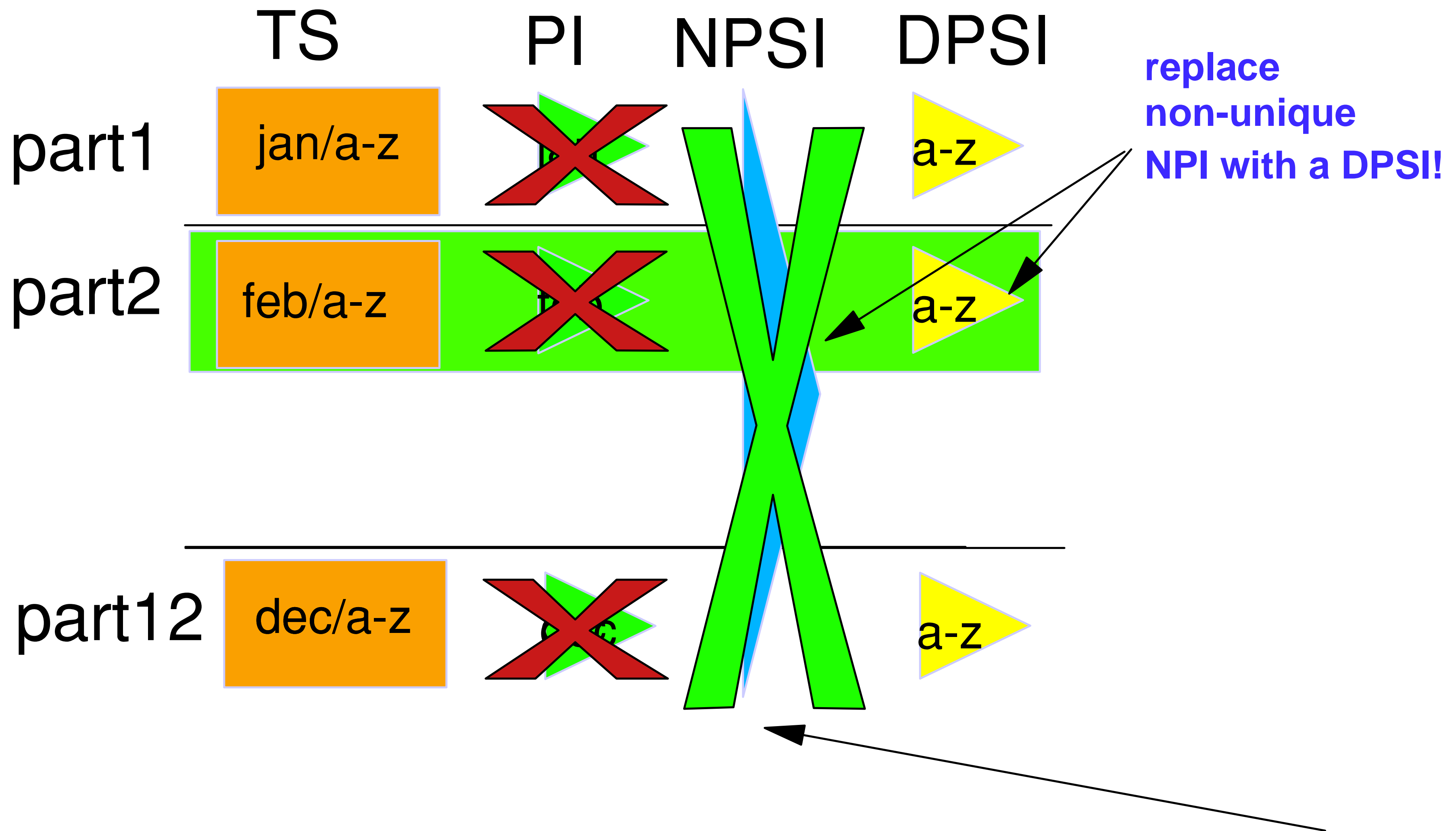
Sample New Style Table



- Partition data by month (PI is optional!)
- Clustering by id or name (DPSI clustering)
- Ideal for Online Reorg with fast switch

© 2004 IBM Corporation

Drop Partitioning Index

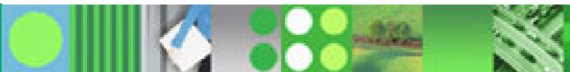


- PI created only for partitioning, not for access...Drop It!

© 2004 IBM Corporation

Utility operations

- CHECK DATA: When running on entire table space, sort must be done for DPSI keys. In basically all other cases, sort is avoided
- CHECK INDEX: can be run on partition of DPSI, or logical partition of NPSI
- RUNSTATS: may be run against single partitions, including DPSIs. Partition-level statistics are used to update aggregate statistics for the entire table.
- Partition parallelism: DPSIs allow for totally concurrent operations with PART keyword, as do PIs
  - LOAD, REORG, REBUILD INDEX, CHECK INDEX
- Work data sets may require more space if there is a mixture of DPSIs and NPSIs



© 2004 IBM Corporation

Altering column data types

- CHAR(10) to CHAR(20)
- CHAR(10) to VARCHAR(40)
- DEC(5,0) to DEC(10,0)
- INT to DEC(10,0)

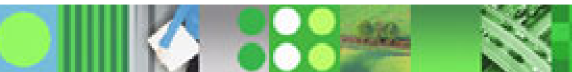
- CHAR(20) to CHAR(10)
- SMALLINT to DEC(3,0)

Note: Column must be large enough to hold maximum value possible for original column

© 2004 IBM Corporation

What happens to any dependent indexes?

- New version created for indexes that references altered column
  - Up to 16 versions per index
- Immediate access for character data type extensions
  - Index placed in AREO*
- Delayed access for numeric data types
  - Index placed in rebuild pending (RBDP)
  - Static SQL disallowed as plans/packages accessing the index are invalidated
  - Invalidate dynamic cache statements
  - Dynamic queries will avoid RBDP indexes
  - Dynamic deletes allowed
  - Updates and inserts allowed for non-unique indexes



© 2004 IBM Corporation

Altering index attributes

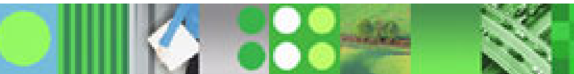
ALTER INDEX index-name

ADD COLUMN (column-name [ASC | DESC])

NOT CLUSTER
CLUSTER

NOT PADDED
PADDED

other options



© 2004 IBM Corporation

Alter index add column

- Ability to add a column to the end of an index
  - Creates a new version
- When column preexists in the table index is placed in RBDP
- If it's a **new column in the table**, add it to the table and index in the **same UOW**
  E.g., ALTER TABLE CUST
  ADD COLUMN NEW_COL;
  ALTER INDEX CUST_IDX
  ADD COLUMN NEW_COL ASC;
  COMMIT;

© 2004 IBM Corporation

Restrictions

- Cannot exceed 64 columns in an index
- Length maximum
  - 2000-n for padded, where n is #nullable columns
  - 2000-n-2m, where, where m is #varying columns
- Disallowed for
  - System defined indexes
  - Partitioning indexes
  - Indexes enforcing a primary or unique constraint

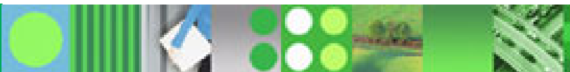
© 2004 IBM Corporation

Alter Not Padded/Padded

- Creates a new index version

- ALTER INDEX PADDED sets index to RBDP
  - Reset by REORG, LOAD REPLACE, or REBUILD
  - For NPI, it is set to PSRBD
  - Index must be rebuilt from data

- ALTER INDEX NOT PADDED sets Index to RBDP
  - Reset by Reorg TS, Load replace or Rebuild
  - For NPI, it is set to PSRBD
  - Index must be rebuilt from data
  - Optimizer can then choose index for index only access



© 2004 IBM Corporation

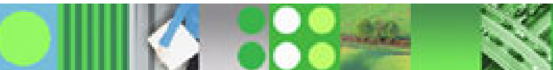
Alter clustering attribute of indexes

- Clustering has been unbundled from partitioning
  - A partitioning index does not have to be the explicit clustering index

- Change Clustering Index with two steps

  - ALTER INDEX index1 NOT CLUSTER
    - Will continue to be used until new clustering index is defined

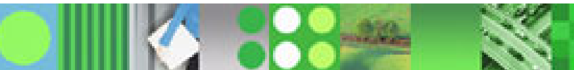
  - ALTER INDEX index2 CLUSTER
    - Immediate effect - inserts follow new clustering but needs reorg!



© 2004 IBM Corporation

Index avoidance

- Indexes bypassed for all DELETE processing
- Non-unique indexes bypassed by DM for updates and inserts
- Optimizer will avoid indexes as follows:
  - Static BIND
    - Indexes in RBDP get resource unavailable
  - Dynamic PREPARE
    - Indexes in RBDP avoided
  - Cached
    - If cached, PREPARE is bypassed
    - Invalidation occurs when index set in RBDP or reset from RBDP
    - RUNSTATS UPDATE NONE REPORT NO flushes cache too
  - Reoptimization
    - Acts the same as initial BIND or PREPARE



© 2004 IBM Corporation

Backward index scan enabled

- DB2 will now select an ascending index and use a backward scan to avoid the sort for the descending order

- DB2 will use the descending index to avoid the sort and scan the descending index backwards to provide the ascending order

- To be able to use an index for backward scan,
  - **Index must be defined on the same columns as ORDER BY and**
  - **Ordering must be exactly opposite of what is requested in ORDER BY.**
  - **i.e., If index defined as DATE DESC, TIME ASC, can do:**
    - **Forward scan for ORDER BY DATE DESC, TIME ASC**
    - **Backward scan for ORDER BY DATE ASC, TIME DESC**
  - **But must sort for**
    - **ORDER BY DATE ASC, TIME ASC or ORDER BY DATE DESC, TIME DESC**

© 2004 IBM Corporation

Avoid Sort by using Backward Index Scan with ORDER BY

```
SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE DESC, STATUS_TIME DESC;

SELECTSTATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE ASC, STATUS_TIME ASC;
```

Index on ACCT_STAT is
ACCT_NUM, STATUS_DATE, STATUS_TIME

Same Index is used.

Backward index scan

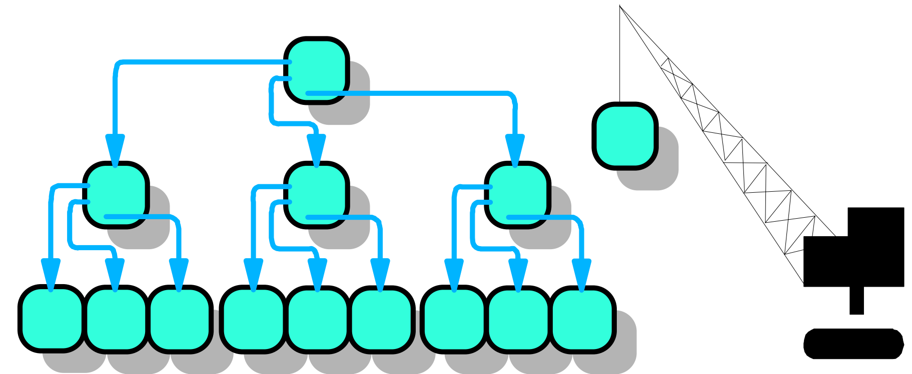
Forward index scan

For scrollable and non scrollable cursors

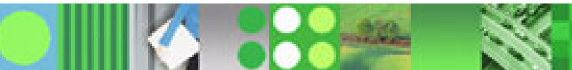
DB2 optimizer will select an ascending index to provide a descending sort order by traversing the index backwards rather than do a sort

© 2004 IBM Corporation

Indexable Predicates: DB2 for z/OS



- Predicates indexable for unlike types
  - Column is decimal; Host variable is float
  - Column char(3); Literal or host variable char(4)
  - Can be used with transitive closure
  - Some restrictions still for stage 1, indexable



© 2004 IBM Corporation

Stage 1 Indexable Unlike-types

- DB2 enhanced to allow index access when host variable & target column are not the same data type
- Deals with programming languages that don't support the full range of SQL data types
  - C/C++ has no DECIMAL data type
  - Java has no fixed length CHAR data type
  - etc.
- Significant performance improvement for many applications
- Simplifies application programmer & DBA tasks

© 2004 IBM Corporation

Summary

- **V8 Index enhancements provide greater efficiencies with regard to:**
  - **Index only access on varying length data**
  - **Decreased storage requirements (in most cases) for varying-length keys**
- **Increased key size, now up to 2000 bytes**
- **Ability to ALTER index columns rather than DROP and CREATE**
- **Clustering is no longer tied to partitioning**

© 2004 IBM Corporation