DB2 9 for z/OS Performance Highlights

The future runs on System z

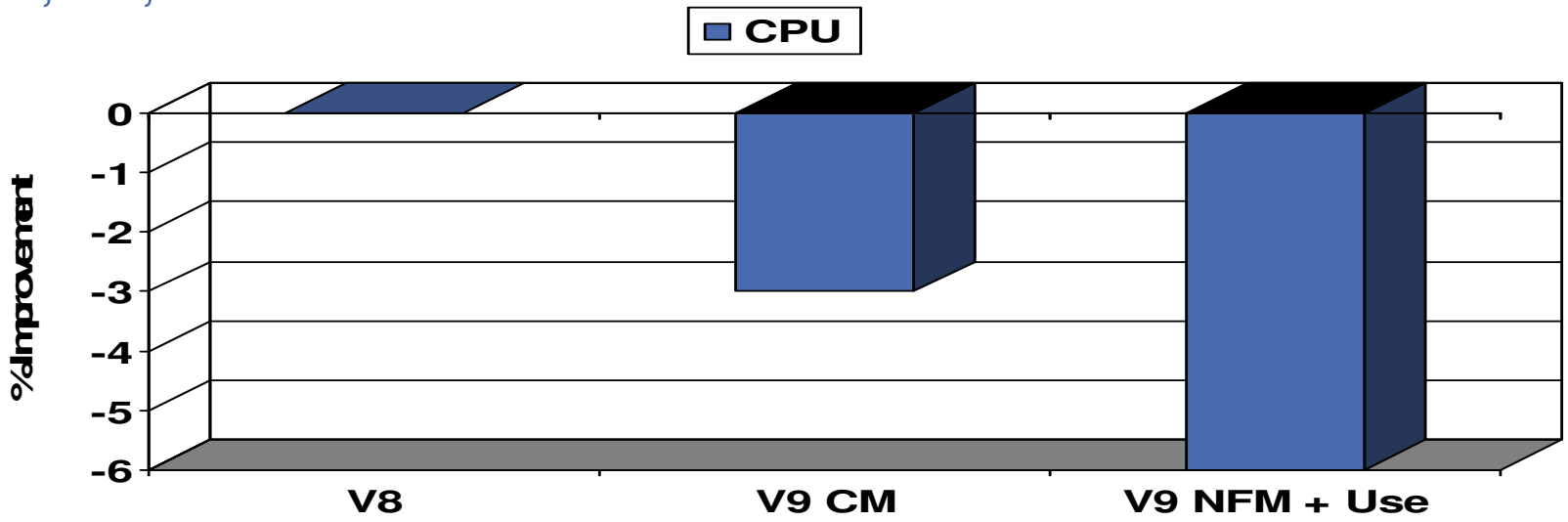
Outline

- General Enhancements
  - Synergy with hardware and O/S
  - Modest overall CPU improvements
  - Significant CPU time reduction in utilities
  - Improved virtual storage usage
  - High-performance INSERT / UPDATE / DELETE
  - Index “Look-aside”
- SQL Performance Enhancements
  - Query Plan Stability
  - DISTINCT and GROUP BY
  - Dynamic Prefetch
  - Improved Subquery Optimization
  - Generalized sparse indexes and data-caching
  - Star Join improvements
  - Query Re-optimization (REOPT)
  - Indexes on Expressions
  - Improved statistics (RUNSTATS)
  - Enhanced query parallelism
- Other Enhancements
  - Native SQL procedures
- And more enhancements ...
  - WLM assisted buffer pool management
  - Reordered row format
  - Buffer manager enhancements
  - Enhanced preformatting
  - Universal table space
  - Index Compression
  - Clone table
  - Object-level recovery
  - Log I/O enhancements
  - WORKFILE database enhancements
  - Optimistic locking
  - MERGE and SELECT FROM MERGE
  - SELECT FROM UPDATE or DELETE
  - FETCH FIRST and ORDER BY in subselect and fullselect
  - TRUNCATE SQL statement
  - INTERSECT and EXCEPT
  - INSTEAD OF triggers
  - BIGINT, VARBINARY, BINARY, and DECFLOAT
  - LOB performance enhancements

+++

DB2 9 CPU Usage: A typical scenario

z10, z9, z890 and z990



- Moving to V9 CM
  - Many performance enhancements available in CM
  - Instant CPU savings
- Moving to V9 NFM and using new function
  - Make database changes (such as use of large index pages)
  - Make application changes (such as use of native SQL procedures)

➢ Perform REORGs, collect statistics, adjust DSNZPARMs

➢ REBIND plans/packages

DB2 9 Utilities Performance Improvements

Broad-based CPU reductions in utilities

20 – 30% overall reduction reported in the field

- 10 to 20% in Image Copy (even with forced CHECKPAGE YES)
- 5 to 30% in Load, REORG, REORG Partition, Rebuild Index
- 20 to 40% in Load
- 20 to 60% in Check Index
- 35% in Load Partition
- 30 to 40% in RUNSTATS Index
- 40 to 50% in REORG Index
- +++

- Improved index processing a significant contributor
  - Reduced path length in index manager
  - Improved index key generation
  - Avoid row movement between batch and DBM1 address spaces

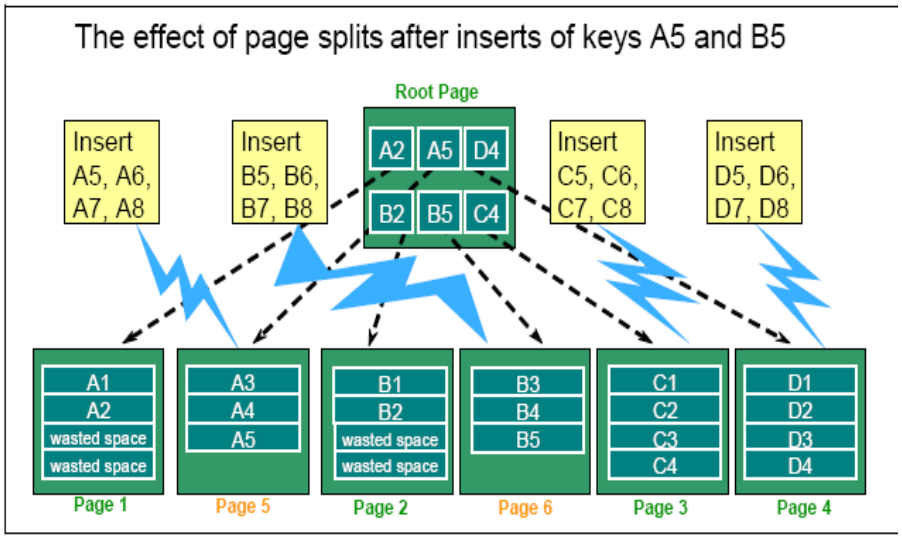
Insert/Update/Delete Performance

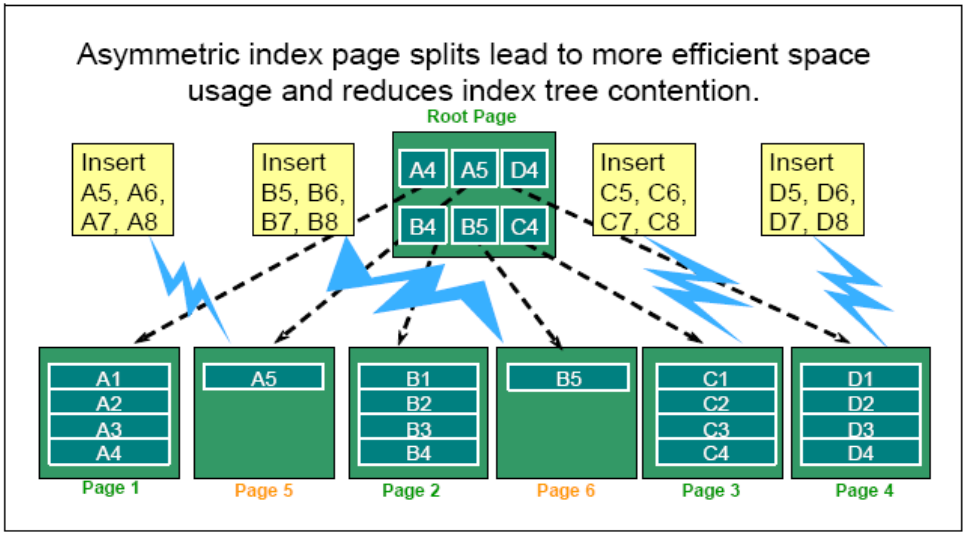
DB2 9 addresses several traditional problem areas for high bandwidth INSERT/UPDATE/DELETE workloads.

- Log latch (LC19) contention relief
- Asymmetric leaf page split
- Larger index page sizes
- Randomized index key
- Increased index look-aside
- Table space APPEND option (can ALTER on and off)
- Not logged tablespaces

- 10x reduction in LC19 waits
- Up to 2x increased logging rate
- May need to increase LOGBUFF

Asymmetric Index Page Split





DB2 V8

- Index split roughly 50/50 (prior to DB2 9)
- Sequential inserts → ~50% free space

Internal lab benchmarks indicate:
- Up to 50% reduction in IX page splits
- Up to 20% reduction in DB2 CPU
- Up to 30% reduction in DB2 ET

DB2 9

- New algorithm dynamically accommodates a varying pattern of inserts
- Up to 90/10 or 10/90 split
- Effective across multiple inserting threads (due to tracking at the page level).
- Improved space utilization (i.e. less REORGs) and reduced contention

Larger Index Page Sizes

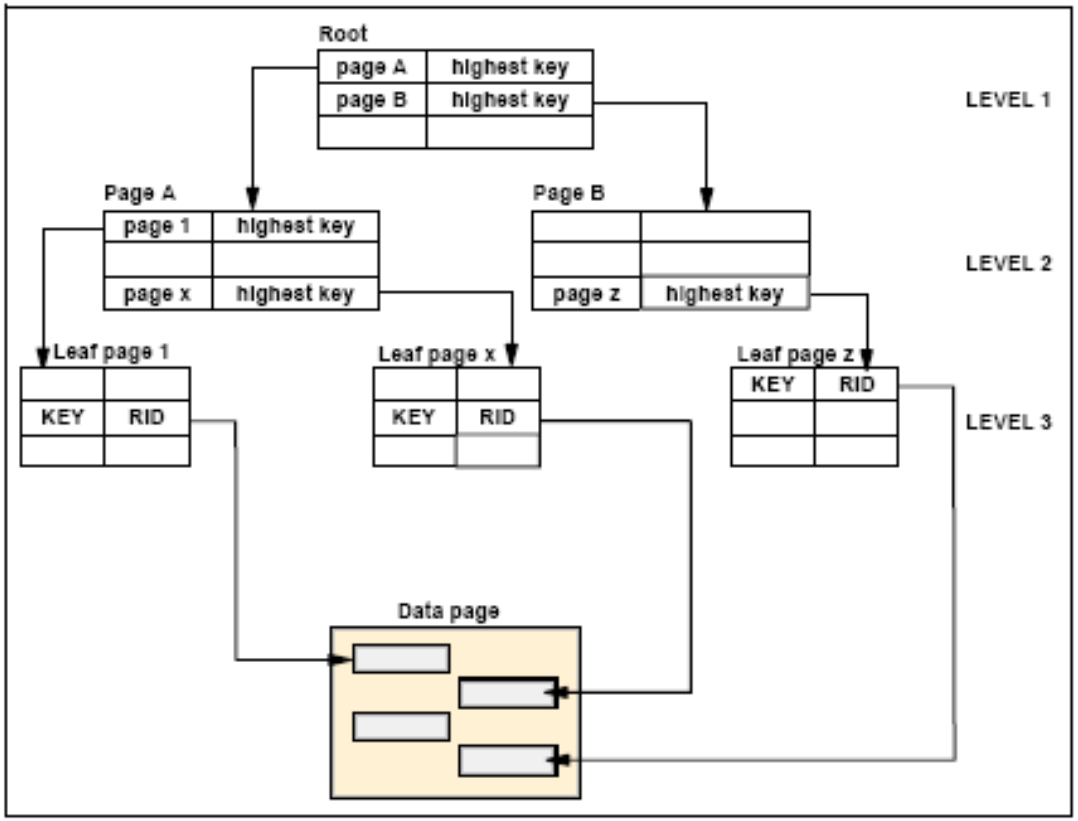
- V8 only support 4K index pages

- DB2 9 supports 8K, 16K, or 32K page
  - Specified via CREATE or ALTER INDEX statement
  - Lower NLEAF & NLEVELS (more keys per page)
  - Up to 8 times less index split (~16x with asymmetric page splits)
- Good for heavy inserts to reduce index splits
  - Especially recommended if high LC6 contention in data sharing
    - 2 forced log writes per split in data sharing
  - Or high LC254 contention in non data-sharing
  - In data sharing, a tradeoff exists between the page splits and potential higher index page physical lock (P-lock) contention.
- Better IX *look-aside* and getpage avoidance

• Up to 50% CPU & 40% ET reduction in DS
• Up to 20% CPU & 30% ET reduction in non DS

Index *Look-aside* (CM)

- DB2 optimizes the lookup of keys on the same page / nearby pages
- Minimizes the number of index getpages with substantial reduction in CPU time
- Big winner for sequential insert, update or delete patterns



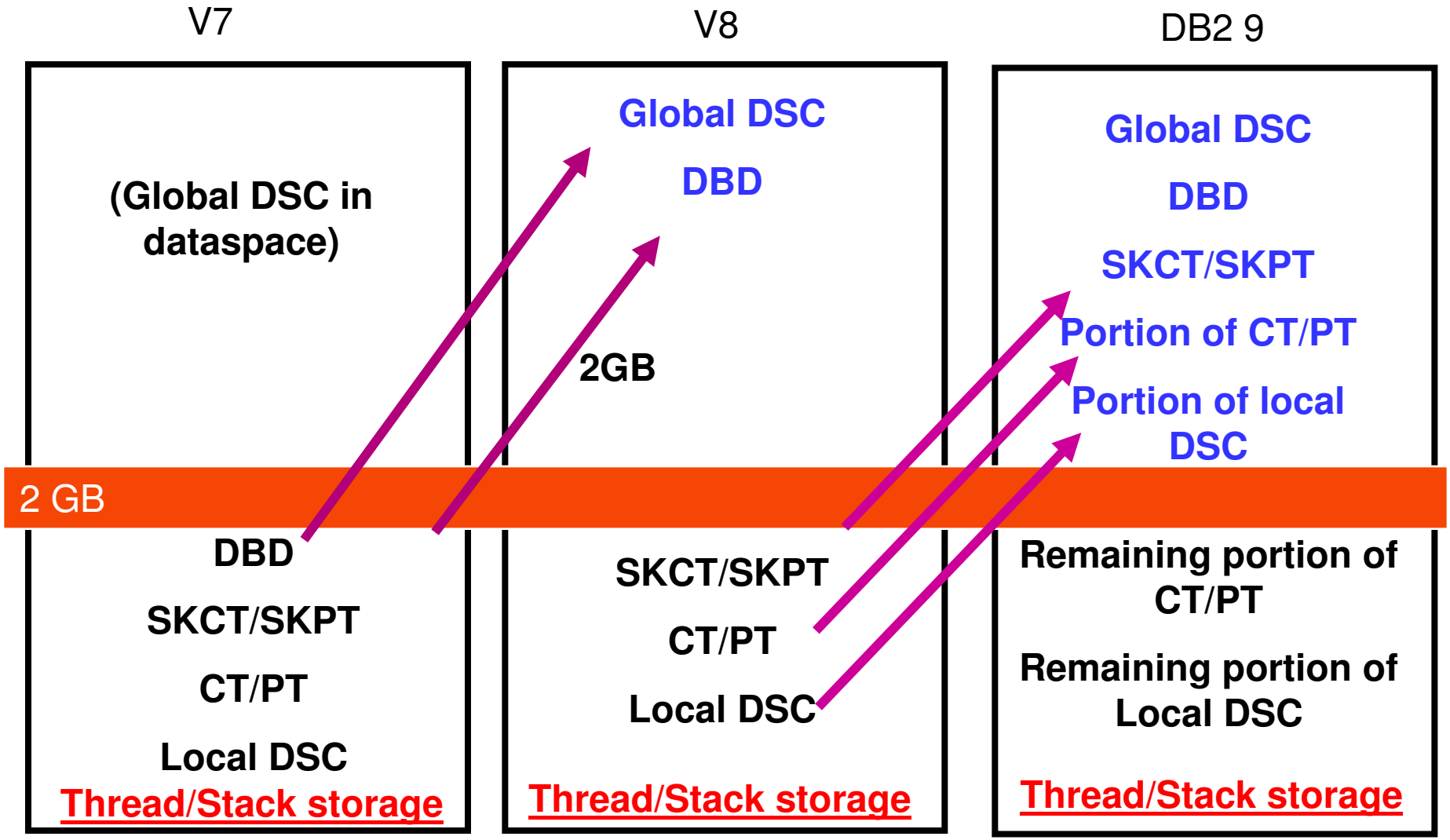
In V8

INSERT – for clustering index only
UPDATE & DELETE – no index *look-aside*

In V9

INSERT, UPDATE & DELETE – index *look-aside* possible for additional indexes where CLUSTERRATIO >= 80%

Virtual Storage Constraint Relief



Virtual Storage Constraint Relief (contd.)

- Each statement bound on V9 now has a below-the-bar portion and an above-the-bar portion.
  - Actual above the bar portion varies by statement, can be 5-90%
  - For static statements, must REBIND plans and packages to get this benefit
- Dynamic statements now have a larger portion above the bar
  - DSC statement text moved above the bar
- Short-term optimizer working memory also above-the-bar
  - Includes parse tree, scratch plans, etc.

- The EDMPOOL reduction will vary based on workload mix
- 40-60% reduction in below-the-bar EDMPOOL size observed for lab workloads

DB2 9 for z/OS Performance Overview

- DB2 Subsystem Enhancements
  - Synergy with hardware and O/S
  - Modest CPU improvements in OLTP workloads
  - Significant CPU time reduction in utilities
  - High-performance INSERT / UPDATE / DELETE
  - Index “Look-aside”
  - Improved virtual storage usage

- SQL Performance Enhancements
  - Query Plan Stability
  - DISTINCT and GROUP BY
  - Dynamic Prefetch
  - Improved Subquery Optimization
  - Generalized sparse indexes and data-caching
  - Star Join improvements
  - Query Re-optimization (REOPT)
  - Indexes on Expressions
  - Improved statistics (RUNSTATS)
  - Enhanced query parallelism

- Other Enhancements
  - Native SQL procedures

Query Plan Stability (CM)

Safeguard against regressions

Preserve old static SQL access paths, Restore when needed

- REBIND PACKAGE …
  - PLANMGMT (BASIC)

  2 copies: Current and Previous
  - PLANMGMT (EXTENDED)

  3 copies: Current, Previous, Original
- REBIND PACKAGE …
  - SWITCH(PREVIOUS)

  Switch between current & previous
  - SWITCH(ORIGINAL)

  Switch between current & original
- FREE PACKAGE …
  - PLANMGMTSCOPE(ALL) – Free package completely
  - PLANMGMTSCOPE(INACTIVE) – Free old copies
- Catalog support
  - SYSPACKAGE reflects active copy
  - SYSPACKDEP reflects dependencies of all copies
  - Other catalogs (SYSPKSYSTEM, …) reflect metadata for all copies

Query Plan Stability - Recommended Usage

- For DASD-constrained environments
  - Use PLANMGMT(BASIC) for the first REBIND after migration to DB2 9
  - For subsequently REBINDs, use PLANMGMT(OFF) <= IMPORTANT!
    - Using PLANMGMT(BASIC) multiple times will destroy older V8 packages
  - This mechanism will preserve DB2 V8 packages but not any older DB2 9 packages
  - Use SWITCH(PREVIOUS) to restore DB2 V8 packages

- For environments without DASD constraints
  - Use PLANMGMT(EXTENDED)
  - Use SWITCH(ORIGINAL) to restore V8 packages

- Reclaiming DASD space
  - Once your packages have reached “steady state” …
  - Use FREE PACKAGE … PLANMGMTSCOPE(INACTIVE) to delete old copies

- APAR PK80375
  - Supports compression of SPT01
  - Field tests show 60-70% reduction in SPT01 usage
  - Enabled via an online changeable ZPARM (‘COMPRESS_SPT01)
  - APAR due to close in Sep 09, usermod available

Improved DISTINCT and GROUP BY (CM)

- Sort avoidance for DISTINCT
  - A DISTINCT query can avoid sorting when using a non-unique index

```
/* Non-unique index exists on P_PARTNAME */
SELECT DISTINCT(P_PARTNAME) FROM PART
ORDER BY P_PARTNAME
FETCH FIRST 10 ROWS ONLY;
```

- Group collapsing for GROUP BY
  - In V8, grouping is done after sort input processing.
  - In V9, DB2 applies the group collapsing optimization for the GROUP BY query without a column function. The result is fewer workfile getpages and less CPU.

```
SELECT COVGTYPE FROM COVERAGE
GROUP BY COVGTYPE
```

- Sort avoidance for GROUP BY
  - Order of GROUP BY columns re-arranged to match index (data may be returned in a different order)
    - GROUP BY C2, C1 <= GROUP BY in C2, C1 sequence
    - Index 1 (C1, C2) <= Index in C1, C2 sequence
  - Relational theory states that order is NOT guaranteed without ORDER BY

Dynamic Prefetch Enhancements (CM)

Sequential Prefetch	Dynamic Prefetch
Chosen at bind/prepare time	Detected at runtime
“Static” – Doesn’t change	“Adaptive” – Can turn on/off
Requires hit to a triggering page	Tracks sequential access pattern
Only prefetch in one direction	Prefetch forward or backward
DB2 9 uses for tablespace scan	DB2 9 uses for other access paths

- Aligned with new cluster ratio formula
- Seq. Pref. cannot fall back to Dyn. Pref. at run time
- Plan table may still show ‘S’ for IX + Data access

- ET reductions between 5-50% measured at the lab
- 10-75% reduction in synchronous I/O’s

Improved Subquery Optimization (CM)

```
SELECT *
FROM OUTER A
WHERE A.C1 IN
      (SELECT B.C1
       FROM INNER B)
```

Non-Correlated

```
SELECT *
FROM OUTER A
WHERE EXISTS
          (SELECT 1
           FROM INNER B
           WHERE B.C1 = A.C1)
```

Correlated

- Semantically equivalent queries
  - Performance could vary drastically!
- DB2 9 optimizer considers both variants
  - Irrespective of how the query is written

Improved Subquery Optimization

Scenario 1: Non-correlated → Correlated

- DB2 V8, Large Non-correlated subquery is materialized*

```
SELECT * FROM SMALL_RELATION A
WHERE A.C1 IN
        (SELECT B.C1 FROM BIG_RELATION B)
```

  - "BIG_RELATION" is scanned and put into workfile
  - "SMALL_RELATION" is joined with the workfile

* Assumes subquery is not transformed to join

- DB2 9 may rewrite non-correlated subquery to correlated
  - Much more efficient if scan / materialisation of BIG_RELATION was avoided
  - Allows matching index access on BIG_RELATION

```
SELECT * FROM SMALL_RELATION A
WHERE EXISTS
  (SELECT 1 FROM BIG_RELATION B
                WHERE B.C1 = A.C1)
```

Improved Subquery Optimization

Scenario 2: Correlated → Non-Correlated

- DB2 V8, Large outer table scanned rather than using matching index access*

```
SELECT * FROM BIG_RELATION A
WHERE EXISTS
  (SELECT 1 FROM SMALL_RELATION B WHERE A.C1 = B.C1)
```

  - "BIG_RELATION" is scanned to obtain A.C1 value
  - "SMALL_RELATION" gets matching index access

* Assumes subquery is not transformed to join

- DB2 9 may rewrite correlated subquery to non-correlated

```
SELECT * FROM BIG_RELATION A
WHERE A.C1 IN
  (SELECT B.C1 FROM SMALL_RELATION B)
```

  - "SMALL_RELATION" scanned and put in workfile
  - Allows more efficient matching index access on BIG_RELATION

Compensating for missing indexes

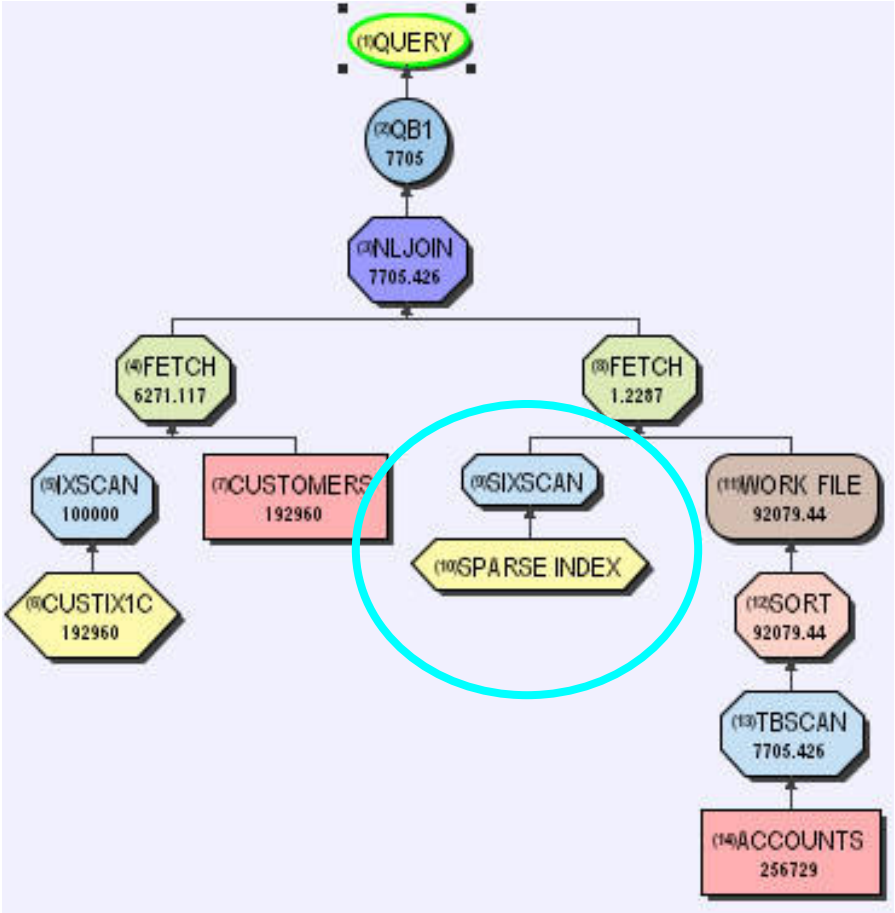
Generalized sparse indexes and IMWF (CM)

- What if DB2 doesn't find efficient indexes to support a join?

- Query optimizer considers building-
  - An "in-memory work file" (IMWF) to cache the entire inner table, OR
  - A "sparse index" atop a materialized workfile.
- Sparse index / IMWF can be built on
  - Base tables
  - Temporary tables
  - Table expressions
  - Materialized views
- Avoids the disaster "NLJ with R-scan of inner table" access path
- Previously only available for star joins

Compensating for missing indexes

Generalized sparse indexes and IMWF



In-memory work file

- IMWF caches the full result (not sparse)
- Is “binary searched” to find target location of search key
- New ZPARM MXDTCACH controls memory use

Sparse index

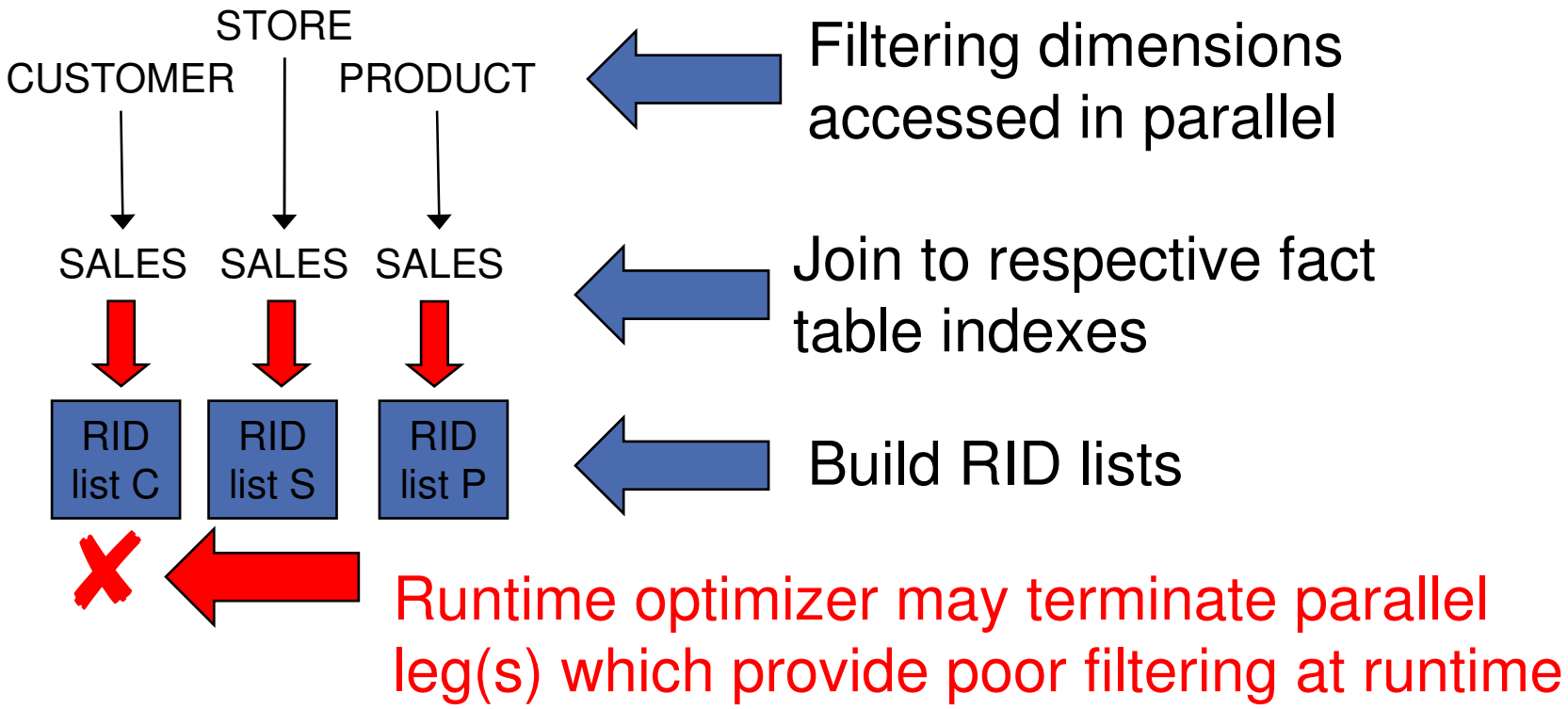
- For insufficient memory (> 240K)
- First, sparse index is “binary searched” …
- … then, a second lookup in sorted workfile
- DB2 9 supports multi-column sparse indexes

Improved Star Query Processing

- In star queries, filtering may come from multiple dimensions
- DB2 8 relies on the presence of multi-column indexes.
  - Requires designing ahead so not appropriate for adhoc queries
  - Too many combinations lead to too many indexes
- DB2 9 introduces new approach
  - Dynamic Index ANDing
  - Only requires one fact index per dimension join column
  - Adaptive algorithms avoids runtime disasters.

Index ANDing – Pre-Fact

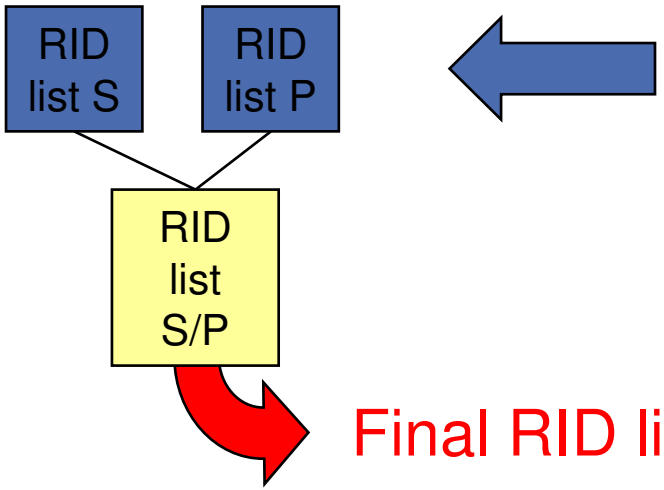
- Pre-fact table access
  - –Filtering may not be (truly) known until runtime



Index ANDing – Fact and Post-Fact

- Fact table access
  - –Intersect filtering RID lists
  - –Access fact table
    - From RID list
- Post fact table
  - –Join back to dimension tables

Using parallelism



Remaining RID lists are "ANDed" (intersected)

Final RID list used for parallel fact table access

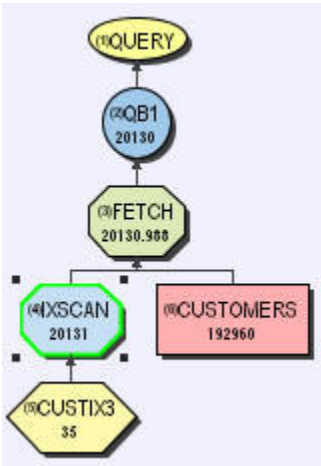
Index on Expression

- Improved filtering at the IX
- Less CPU to evaluate
- Dramatic perf. improvement for matching SQL
- Trade off for updates and IX maint.

- DB2 9 supports “index on expression”
  - Can turn a stage 2 predicate into indexable

```
SELECT *
FROM CUSTOMERS
WHERE YEAR(BIRTHDATE) = 1971

CREATE INDEX ADMF001.CUSTIX3
    ON ADMF001.CUSTOMERS
(YEAR(BIRTHDATE) ASC)
```



Previous FF = 1/25
Now, RUNSTATS collects frequencies. Improved FF accuracy

Name	Value
Input RIDs	192960
Index Leaf Pages	241
Matching Predicates	Filter Factor
ADMF001.CUSTOMERS.= CAST(1971 AS INTEGER)	0.1043
Scanned Leaf Pages	26
Output RIDs	20131
Total Filter Factor	0.1043
Matching Columns	1

Improved Optimizer Statistics

Histograms

- RUNSTATS can now collect histogram statistics
  - Think of histogram as a mechanism to support "*range frequencies*"
- Histograms address skews across ranges of data values
- Summarizes data distribution on an interval scale
- DB2 uses equal-depth histograms
  - Each quantile has about the same number of rows
  - Example - 1, 3, 3, 4, 4, 6, 7, 8, 9, 10, 12, 15 is cut into 3 quantiles

Seq No	Low Value	High Value	Cardinality	Frequency
1	1	4	5	5/12
2	6	9	4	4/12
3	10	15	3	3/12

Improved Optimizer Statistics

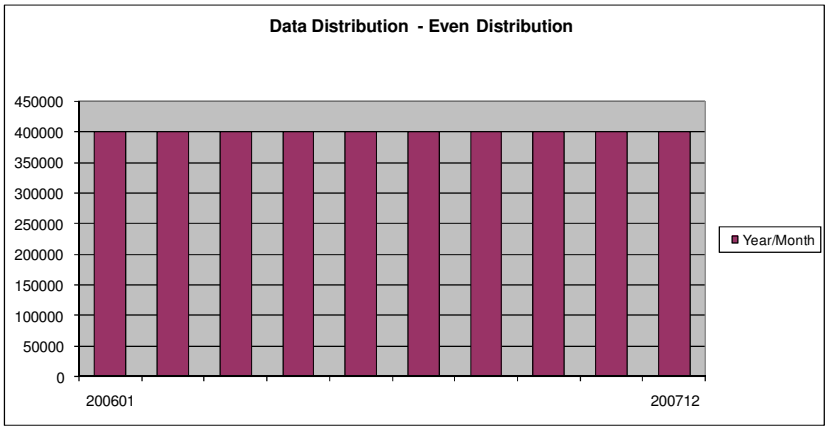
Histograms – An Example

SAP uses INTEGER (or VARCHAR) for YEAR-MONTH

WHERE YEARMONTH BETWEEN 200601 AND 200612

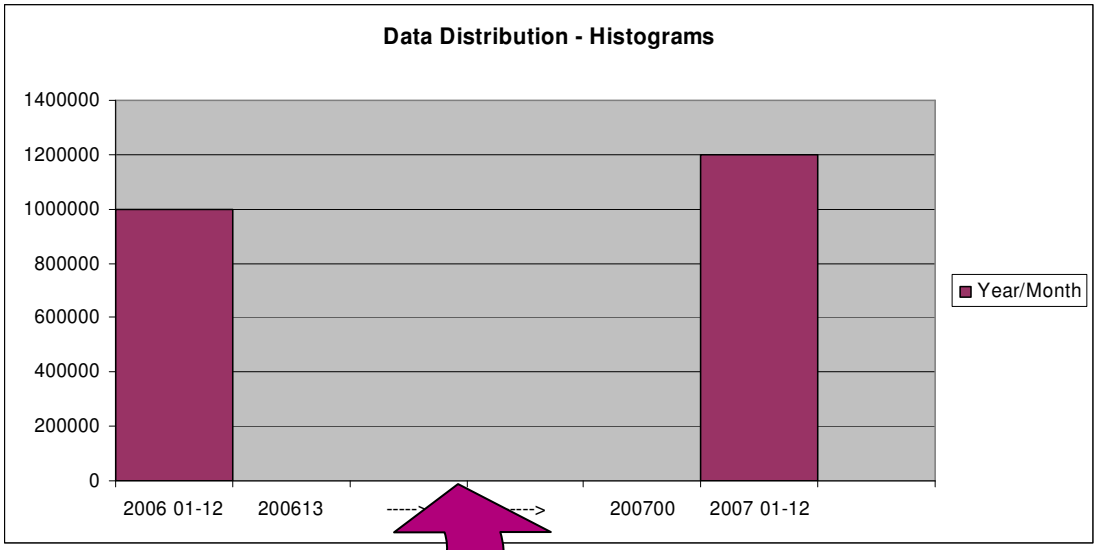
- Assuming data for 2006 & 2007
  - FF = (high-value – low-value) / (high2key – low2key)
  - FF = (200612 – 200601) / (200711 – 200602)
  - FF = 11 / 109
  - **10% of rows estimated to return**

Data assumed as evenly distributed between low and high range



Histograms – An Example (contd.)

- Data only exists in ranges 200601-12 & 200701-12
  - Collect via histograms
    - 45% of rows estimated to return



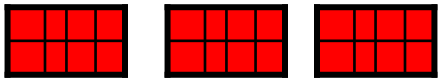
No data between
200613 & 200700

WHERE YEARMONTH BETWEEN 200601 AND 200612

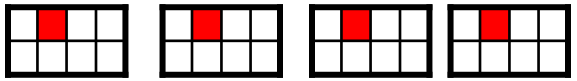
Improved Optimizer Statistics

CLUSTERRATIO and DRF (CM)

- New ZPARM, STATCLUS, in DB2 9
  - Default STATCLUS = ENHANCED
  - New CLUSTERRATIO formula
    - Better awareness of prefetch range
    - Considers forward- and backward- clustering
    - More accurate value for lower cardinality indexes
  - New statistic, DATAREPEATFACTOR
    - Differentiates density and sequential



Dense (and sequential)



Sequential (not dense)

- Recommend RUNSTATS before mass REBIND in DB2 9

REOPT(AUTO) for Dynamic SQL (CM)

- Question: How many rows are returned by this query?

```
SELECT *
FROM PEOPLE_IN_THIS_ROOM
WHERE AGE BETWEEN ? and ?
```

- Host variable and parameter markers affect query optimization!
  - Many “good” access paths may exist for the same query
    - AGE BETWEEN 21 and 100 — Tablespace scan
    - AGE BETWEEN 0 and 21 — Index scan
  - When data is skewed, knowing values at BIND/PREPARE becomes important
  - Without the values, the optimizer takes a guess
  - Query reoptimization (REOPT) comes to the rescue

- V8 REOPT options
  - Dynamic SQL
    - REOPT(NONE, ONCE, ALWAYS)
  - Static SQL
    - REOPT(NONE, ALWAYS)

- DB2 9 Addition for Dynamic SQL
  - Bind option REOPT(AUTO)

REOPT(AUTO) for Dynamic SQL (contd.)

- For dynamic SQL with parameter markers
  - DB2 will automatically re-optimize the SQL when
    - Filtering of one or more of the predicates changes dramatically
      - Such that table join sequence or index selection may change
    - Some statistics cached to improve performance of runtime check
  - Newly generated access path will replace the global statement cache copy.

- First optimization is the same as REOPT(ONCE)
  - Followed by analysis of the values supplied at each execution of the statement

Query Parallelism Enhancements

```
SELECT … FROM PART, SUPPLIER WHERE …
```

- V8 parallelizes the “cheapest” serial plan
  - E.g., PART nested-loop-join SUPPLIER
  - This cheapest serial plan may result in a very poor parallel plan
- DB2 parallelizes many “good” serial plans
  - E.g.,
    - PART nested-loop-join SUPPLIER,
    - SUPPLIER nested-loop-join PART, …
  - Lowest cost parallel plan is chosen
- The end result?
  - More opportunities for parallel access paths
  - Increased zIIP offload

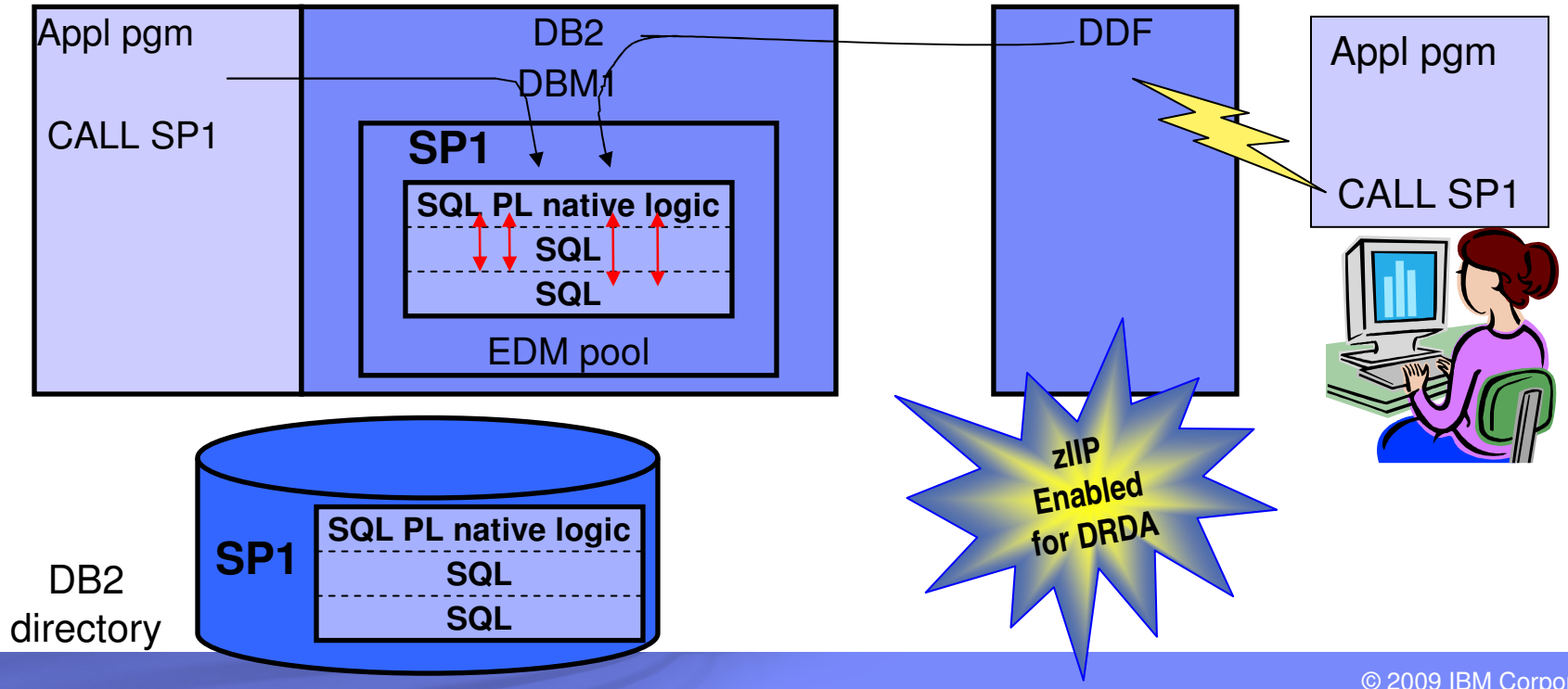
Query Parallelism Enhancements (contd.)

- Query parallelism is all about *divide and conquer*
- DEGREE determines the extent of division (or “cutting”)

- In V8
  - Division happens on the leading table
  - Star queries an an exception (fact table is divided)
- In DB2 9
  - Division can happen on non-leading table
    - A leading workfile or 1-row table cannot be divided
    - In such cases, inner table could provide a better division of work
  - Histogram statistics exploited for more even division
  - CPU bound query DEGREE can be <= # of CPUs * 4
- Again, increased parallelism results in greater zIIP offload

Native SQL Procedures

- Eliminates generated C code and compilation
- Fully integrated into the DB2 engine
- Extensive support for versioning
- Allow nested compound statements within a procedure



Summary

- DB2 Subsystem Enhancements
  - Synergy with hardware and O/S
  - Modest CPU improvements in OLTP workloads
  - Significant CPU time reduction in utilities
  - High-performance INSERT / UPDATE / DELETE
  - Index “Look-aside”
  - Improved virtual storage usage
- SQL Performance Enhancements
  - Query Plan Stability
  - DISTINCT and GROUP BY
  - Dynamic Prefetch
  - Improved Subquery Optimization
  - Generalized sparse indexes and data-caching
  - Star Join improvements
  - Query Re-optimization (REOPT)
  - Indexes on Expressions
  - Improved statistics (RUNSTATS)
  - Enhanced query parallelism
- Other Enhancements
  - Native SQL procedures

Reference

- Main DB2 for z/OS page
  - http://www.ibm.com/software/data/db2/zos/index.html
- DB2 for z/OS 9
  - http://www.ibm.com/software/data/db2/zos/db2zosv91.html
  - DB2 9 Performance Monitoring and Tuning Guide
- Redbooks at www.redbooks.ibm.com
  - DB2 9 for z/OS Technical Overview SG24-7330
  - DB2 9 for z/OS Performance Topics SG24-7473
- Other useful links
  - E-support (presentations and papers) at www.ibm.com/software/db2zos/support.html
  - Presentations ftp://ftp.software.ibm.com/software/data/db2/zos/presentations/

Backup Slides

Index Compression (NFM)

- Always stored as 4k page on disk
- Best with high BP hit ratio

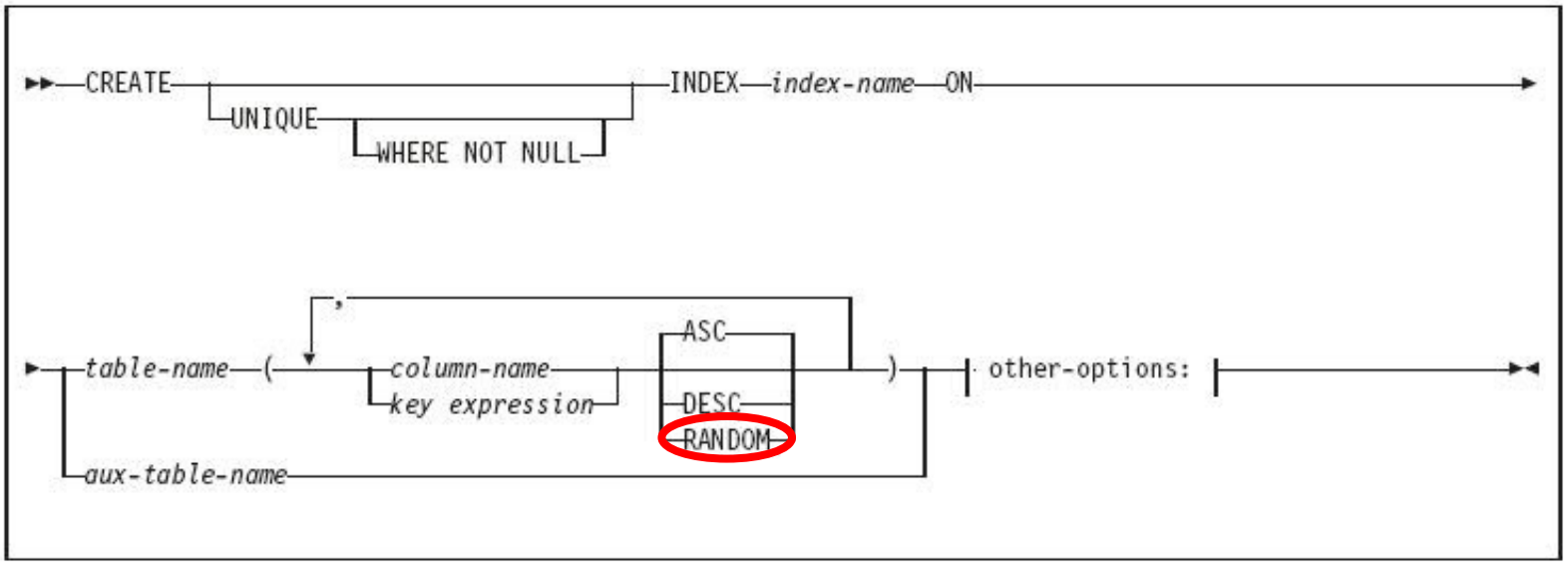
Recommended when indexes take lot more space than compressed data.

Difference between data and index compression

	Data	Index
Level of compression	Row	Page
Comp in DASD	Yes	Yes
Comp in BP and Log	Yes	No
Comp Dictionary	Yes	No
'Typical' Comp Ratio CR	10 - 90%	25 - 75%

Use DSN1COMP utility to predict index compression ratio.

Randomized Index Key (NFM)



- Lock contention relief
- LC 6 relief

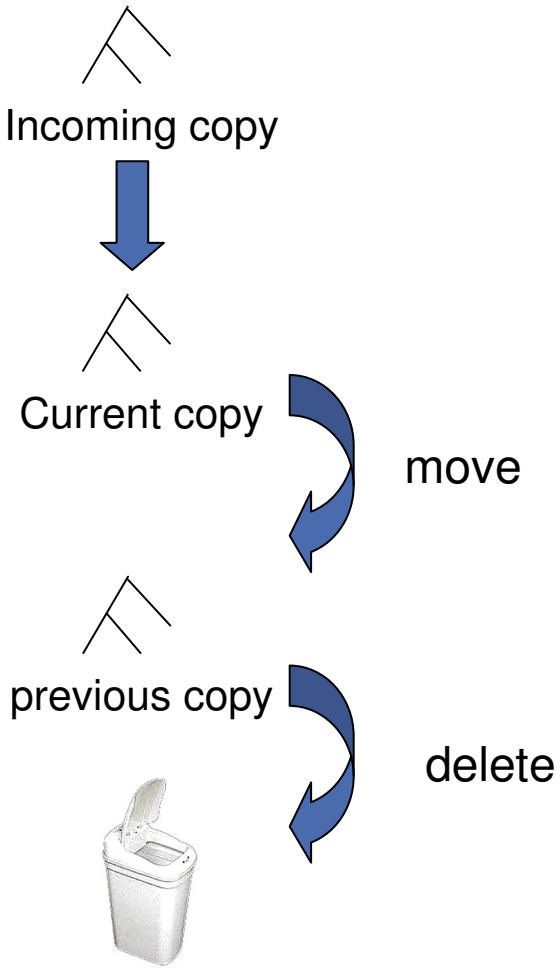
Vs.

- Additional getpages
- Additional read/write I/Os
- Increased lock requests

- Can provide dramatic improvement or degradation!
- Recommend making randomized indexes bufferpool resident
- Can be any one or more columns of an IX key
- Cannot support order

Query Plan Stability - BASIC support

REBIND … PLANMGMT(BASIC)



REBIND … SWITCH(PREVIOUS)

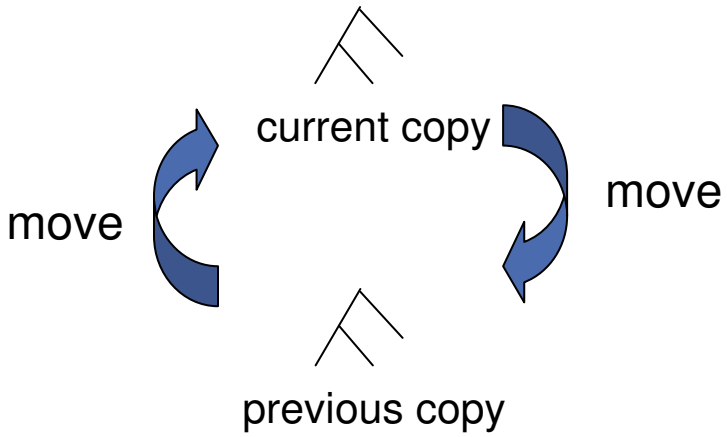
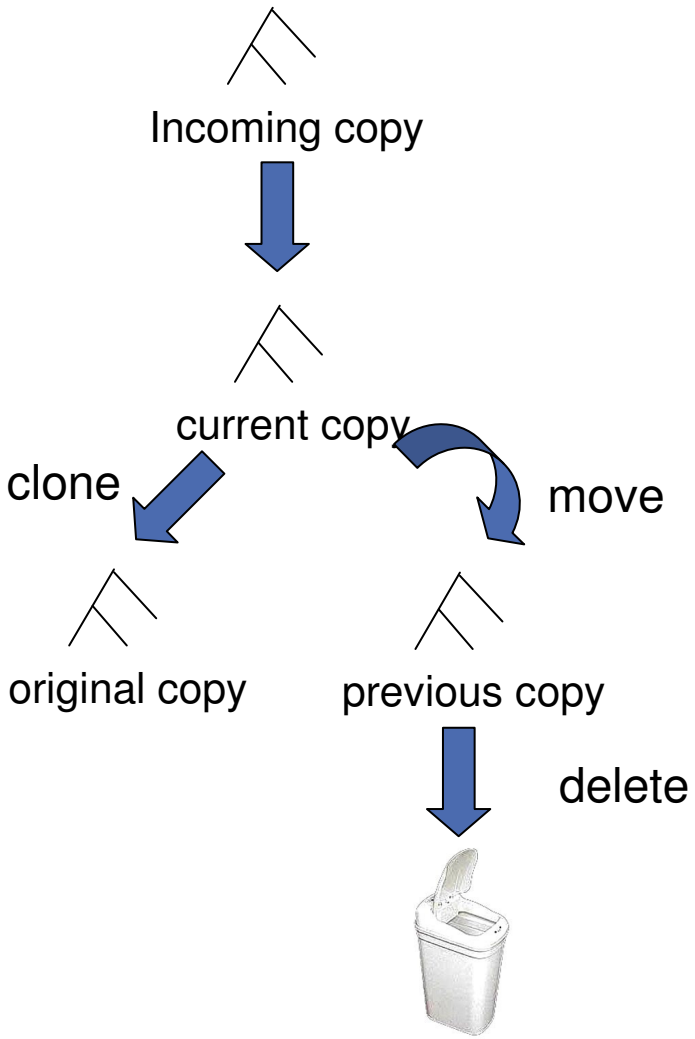


Chart is to be read from bottom to top

Query Plan Stability - EXTENDED support

REBIND … PLANMGMT(EXTENDED)



REBIND … SWITCH(ORIGINAL)