Fast Track to Optimal DB2 Performance

The future runs on System z

Agenda

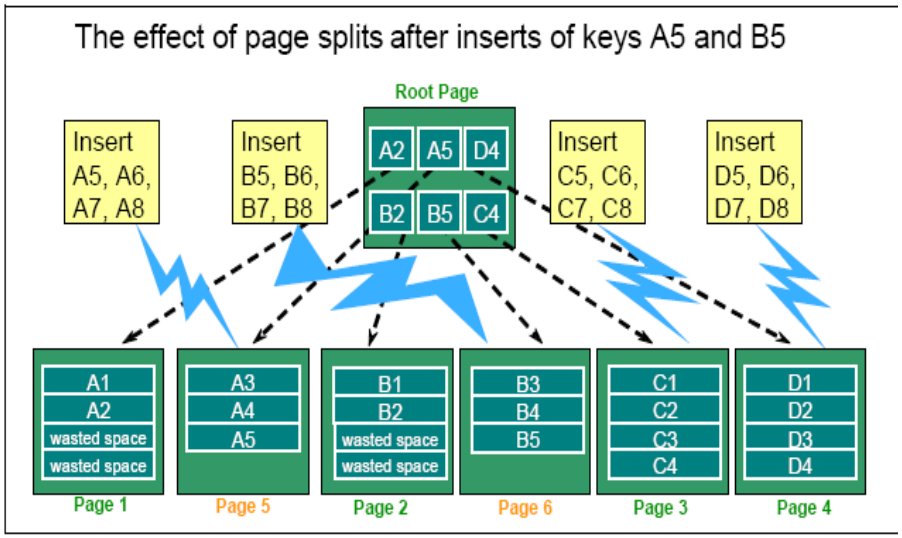
- Insert/Update/Delete Performance
- Virtual Storage Constraint Relief
- Query Performance Enhancements
  - Plan Stability
  - REOPT(AUTO)
  - Improved Optimizer Statistics
  - Subquery Optimization
  - Generalized Sparse Index and In-Memory Workfile

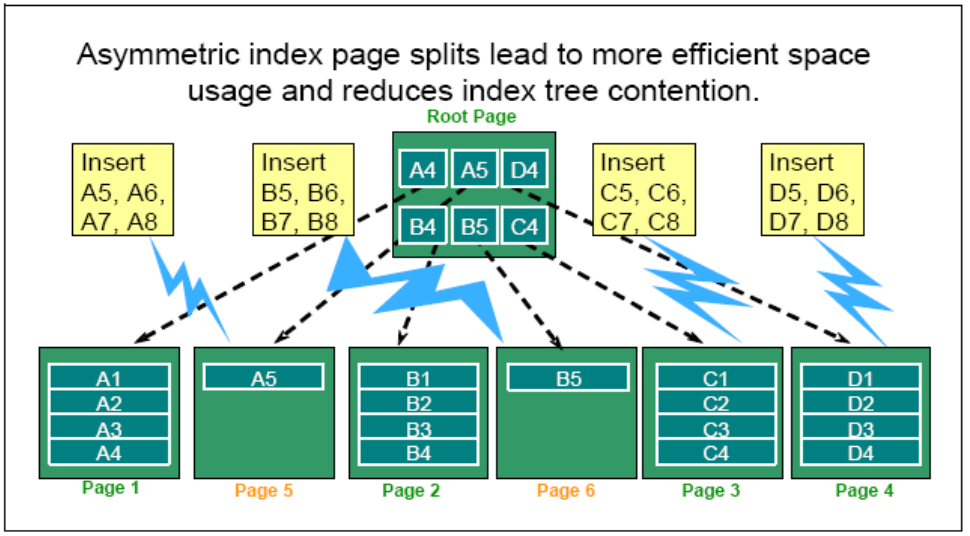
Insert/Update/Delete Performance

DB2 9 offers improved Scalability and Performance
- Especially in data sharing environments

- Reduced Latch contention
  - Reduced Log Latch (Latch 19) Contention
  - Reduced LRSN Spin
- Asymmetric leaf page split
- Randomized index key
- Larger index page sizes
- Increased index look-aside
- Table space APPEND option (can ALTER on and off)
- Not logged tablespaces

Asymmetric Index Page Split (NFM)

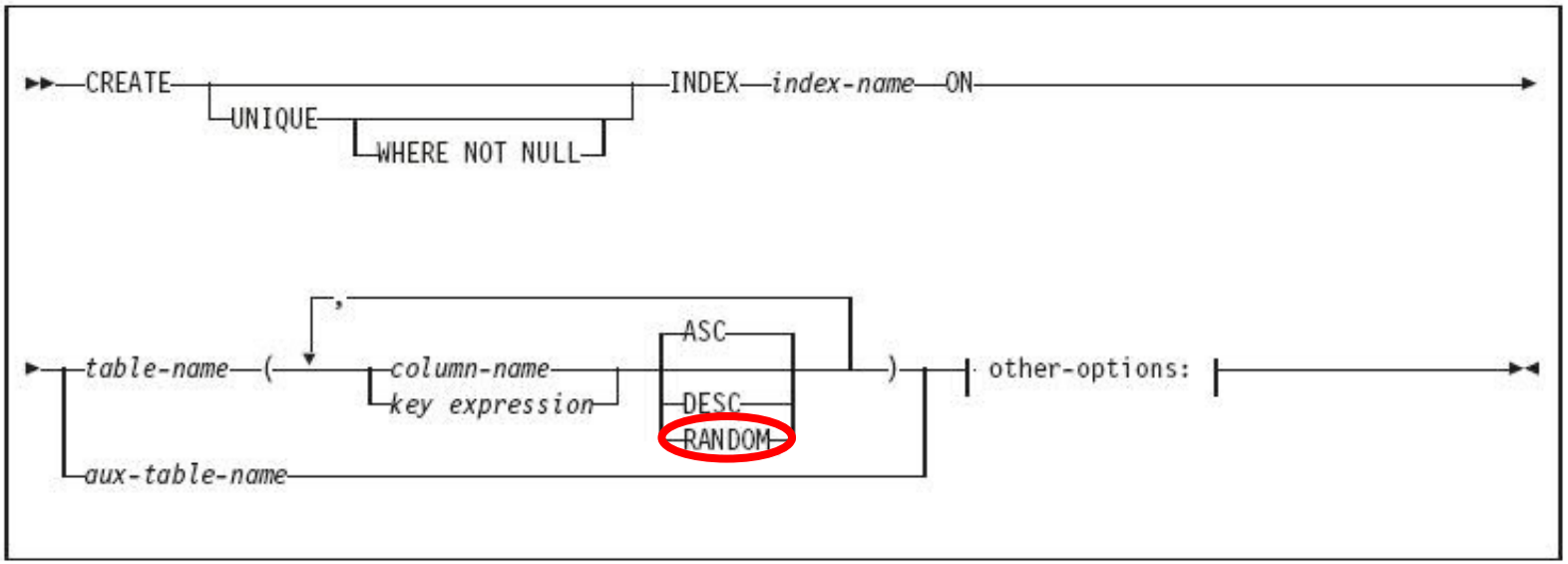




- Index split roughly 50/50 (prior to DB2 9)
- Sequential inserts → ~50% free space

- New algorithm dynamically accommodates a varying pattern of inserts
- Up to 90/10 split
- Effective across multiple inserting threads (due to tracking at the page level).
- Improve space utilization and reduce contention.

Randomized Index Key (NFM)



- Lock contention relief

Vs.

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

- Cannot support order
- Can provide dramatic improvement or degradation!
- Recommend making randomized indexes bufferpool resident

Larger Index page Sizes (NFM)

- 8K, 16K, or 32K page
  - Up to 8 times less index split
- 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 shown in IFCID57
- Lower NLEAF & NLEVELS
- Exploitation of larger page sizes (> 8K) more likely without index compression

Index Compression (NFM)

Difference between data and index compression

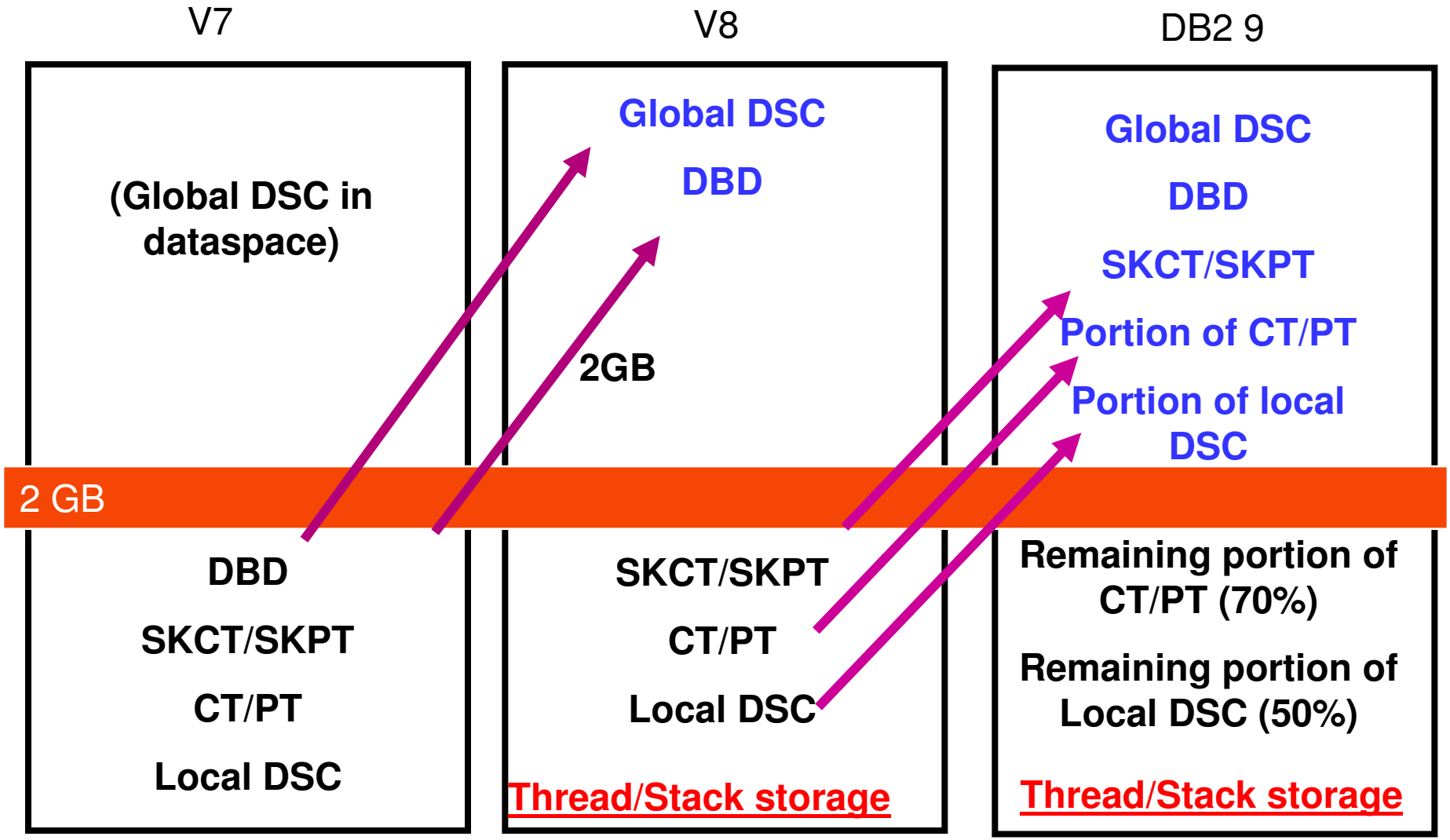
	Data	Index
Level of compression	Row	Page (1)
CPU overhead	In Acctg	In Acctg and/or DBM1 SRB
Comp in DASD	Yes	Yes
Comp in BP and Log	Yes	No
Comp Dictionary	Yes	No (2)
'Typical' Comp Ratio CR	10 - 90%	25 - 75% (3)

Use DSN1COMP utility to predict index compression ratio.

Index *Look-aside* (CM)

- In V8
  - Insert – clustering index only
  - Delete – no index lookaside
- In V9,
  - Insert & Delete – now possible for additional indexes where CLUSTERRATIO >= 80%
- Potential for big reduction in the number of index getpages with substantial reduction in CPU time
  - Benchmark Example - Heavy insert
    - Large table, 3 indexes, all in ascending index key sequence,
    - 0+6+6=12 index Getpages per average insert in V8
    - 0+1+1=2 in V9

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 have a larger portion above the bar than
  - DSC statement text moved above the bar

- 40-60% reduction in below the bar EDMPOOL size observed for lab workloads
- Almost 300 MB reduction in below-bar storage for SAP tests

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)

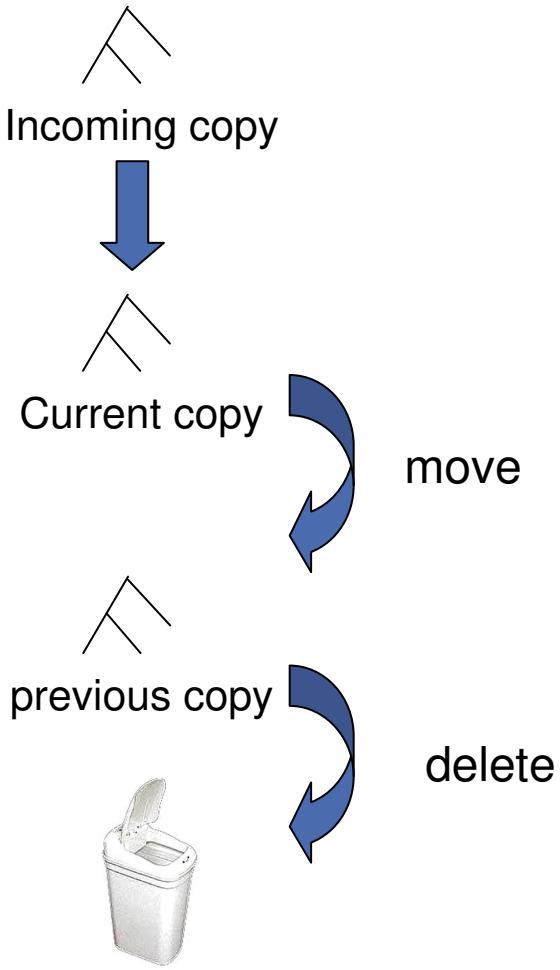
  3 copies: Current, Previous, Original
- REBIND PACKAGE …
  - SWITCH(PREVIOUS)

  Switch between current & previous
  - SWITCH(ORIGINAL)

  Switch between current & original
- Most bind options can be changed at REBIND
  - *But a few must be the same …*
- 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
- Invalidation and Auto Bind
  - Each copy invalidated separately
- PK80375 – SPT01 Compression

Query Plan Stability - BASIC support

REBIND … PLANMGMT(BASIC)



REBIND … SWITCH(PREVIOUS)

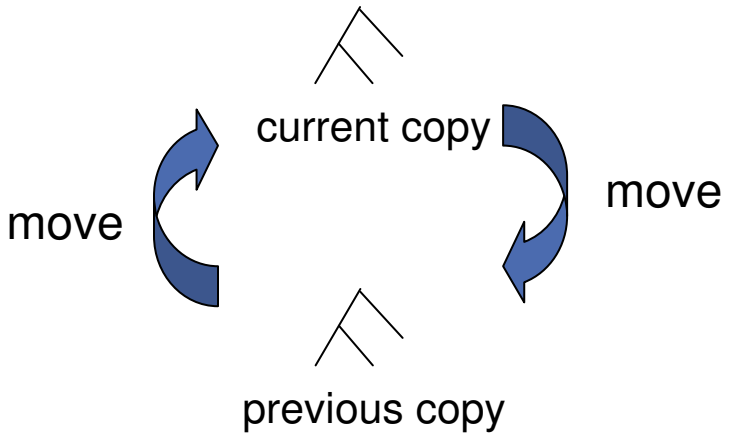
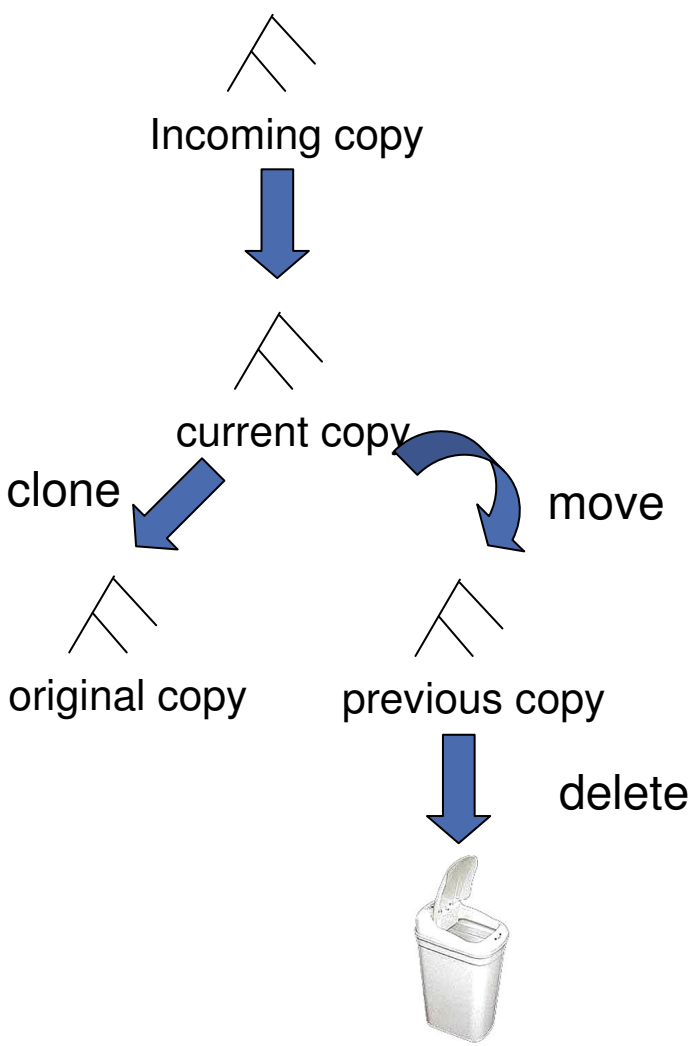


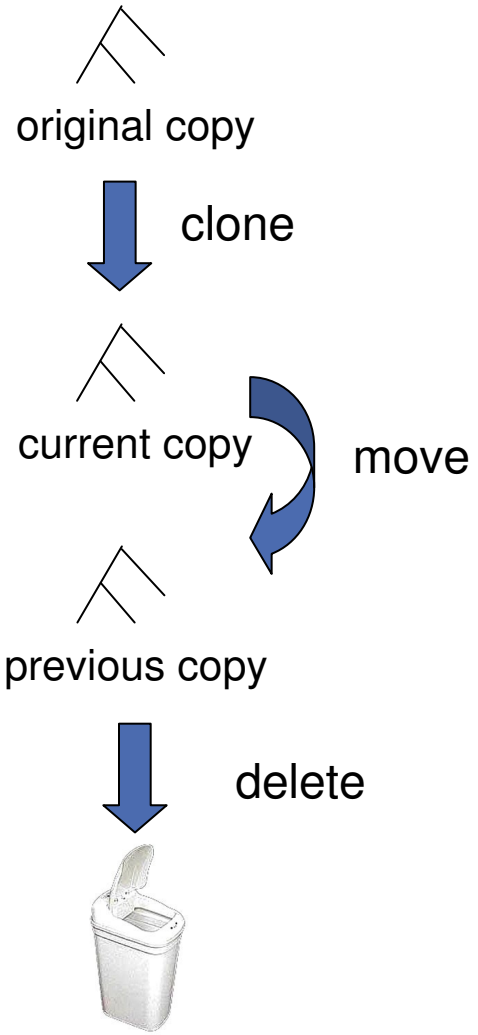
Chart is to be read from bottom to top

Query Plan Stability - EXTENDED support

REBIND … PLANMGMT(EXTENDED)



REBIND … SWITCH(ORIGINAL)



REOPT(AUTO) for Dynamic SQL (CM)

- 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

Improved Optimizer Statistics - Histograms

- RUNSTATS will produce equal-depth histogram
  - Each quantile (range) will have approx same number of rows
  - Address data skew across ranges of data values

- 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

RUNSTATS Histogram Statistics Notes

- RUNSTATS
  - Maximum 100 quantiles for a column
  - Same value columns WILL be in the same quantile
  - Quantiles will be similar size but:
    - Will try to avoid big gaps inside quantiles
    - Null WILL have a separate quantile
- Supports column groups as well as single columns
- Think “frequencies” for high cardinality columns

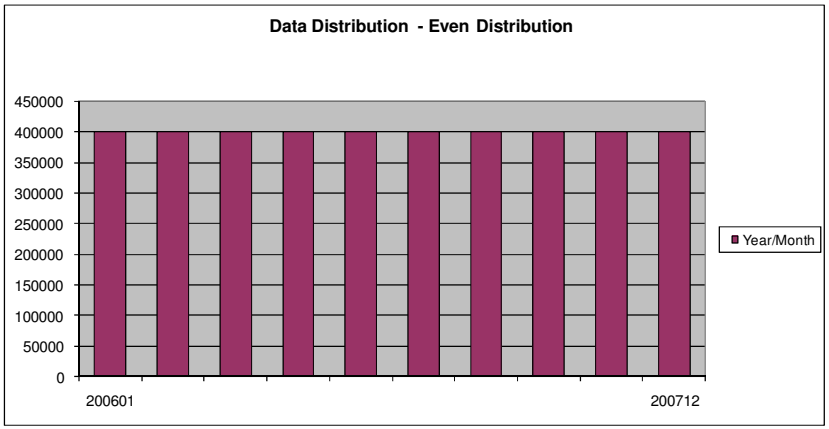
Histogram Statistics – An Example

- SAP uses INTEGER (or VARCHAR) for YEAR-MONTH

WHERE YEARMONTH BETWEEN 200601 AND 200612

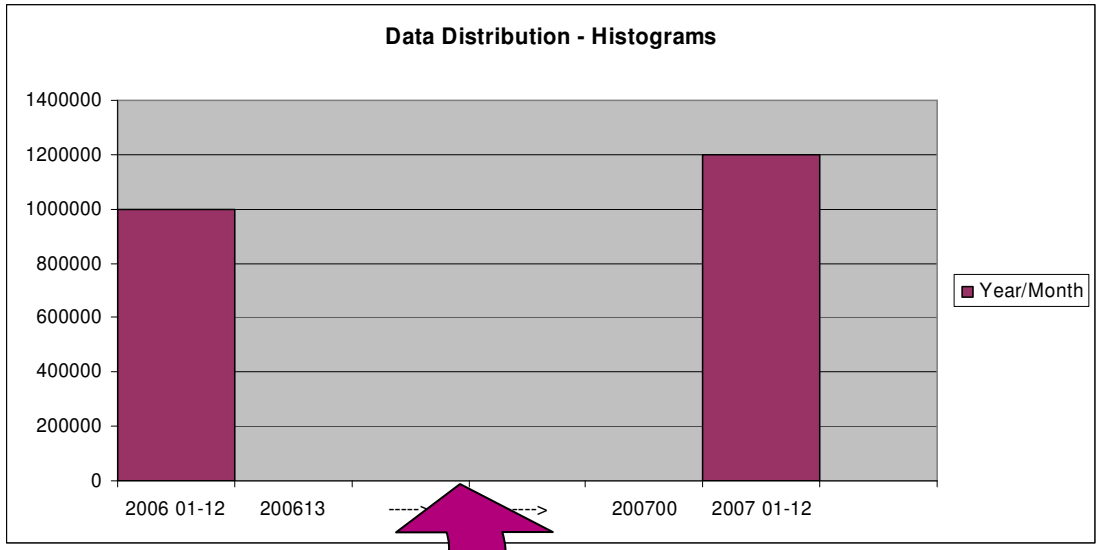
- Assuming data for 2006 & 2007
  - FF = (high-value – low-value) / (high2key – low2key)
  - FF = (200612 – 200601) / (200711 – 200602)
  - **10% of rows estimated to return**

Data assumed as evenly distributed between low and high range



Histogram Statistics – An Example

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



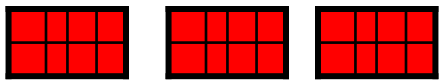
No data between
200613 & 200700

WHERE YEARMONTH BETWEEN 200601 AND 200612

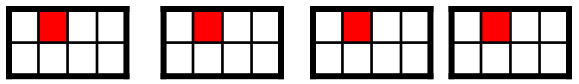
Improved Optimizer Statistics

CLUSTERRATIO and DRF

- New ZPARM, STATCLUS, in DB2 9
  - Default STATCLUS = ENHANCED
  - New CLUSTERRATIO formula
    - Better awareness of prefetch range
    - 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

Dynamic Prefetch Enhancements

Sequential Prefetch	Dynamic Prefetch
Chosen at bind/prepare time	Detected at runtime
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

Improved Subquery Optimization (CM)

```
SELECT *
FROM TABLEA A
WHERE A.C1 IN
      (SELECT B.C1
       FROM TABLEB B)
```

Non-Correlated

```
SELECT *
FROM TABLE A
WHERE EXISTS
          (SELECT 1
           FROM TABLEB B
           WHERE B.C1 = A.C1)
```

Correlated

- Semantically equivalent queries
  - Performance could vary drastically!
- DB2 9 optimizer considers both contexts

Improved Subquery Optimization

Scenario 1: Non-correlated → Correlated

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

```
SELECT * FROM SMALL_TABLE A
WHERE A.C1 IN
        (SELECT B.C1 FROM BIG_TABLE B)
```

  - "BIG_TABLE" is scanned and put into workfile
  - "SMALL_TABLE" 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_TABLE was avoided
  - Allows matching index access on BIG_TABLE

```
SELECT * FROM SMALL_TABLE A
WHERE EXISTS
  (SELECT 1 FROM BIG_TABLE 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_TABLE A
WHERE EXISTS
  (SELECT 1 FROM SMALL_TABLE B WHERE A.C1 = B.C1)
```

  - "BIG_TABLE" is scanned to obtain A.C1 value
  - "SMALL_TABLE" gets matching index access

* Assumes subquery is not transformed to join

- DB2 9 may rewrite correlated subquery to non-correlated

```
SELECT * FROM BIG_TABLE A
WHERE A.C1 IN
  (SELECT B.C1 FROM SMALL_TABLE B)
```

  - "SMALL_TABLE" scanned and put in workfile
  - Allows more efficient matching index access on BIG_TABLE

Improved Subquery Optimization

Representing Subqueries as Virtual Tables

- A new way to internally represent subqueries
  - Represented as a Virtual table
    - Allows subquery to be considered in different join sequences
    - May or may not represent a workfile
    - Apply only to subqueries that cannot be transformed to joins

Correlated or non-correlated?.....I shouldn't have to care!

Improved Subquery Optimization

Subqueries in EXPLAIN Output

- Additional row for materialized “Virtual Table”
  - Table type is "W" for "Workfile".
    - Name includes an indicator of the subquery QB number
      - Example ➔ “DSNWF(02)”
  - Non-materialized Virtual tables will not be shown in EXPLAIN output.
- Additional column PARENT_PLANNO
  - Used with PARENT_QBLOCKNO to connect child QB to parent
  - V8 only contains PARENT_QBNO
    - Not possible to distinguish which plan step the child tasks belong to.

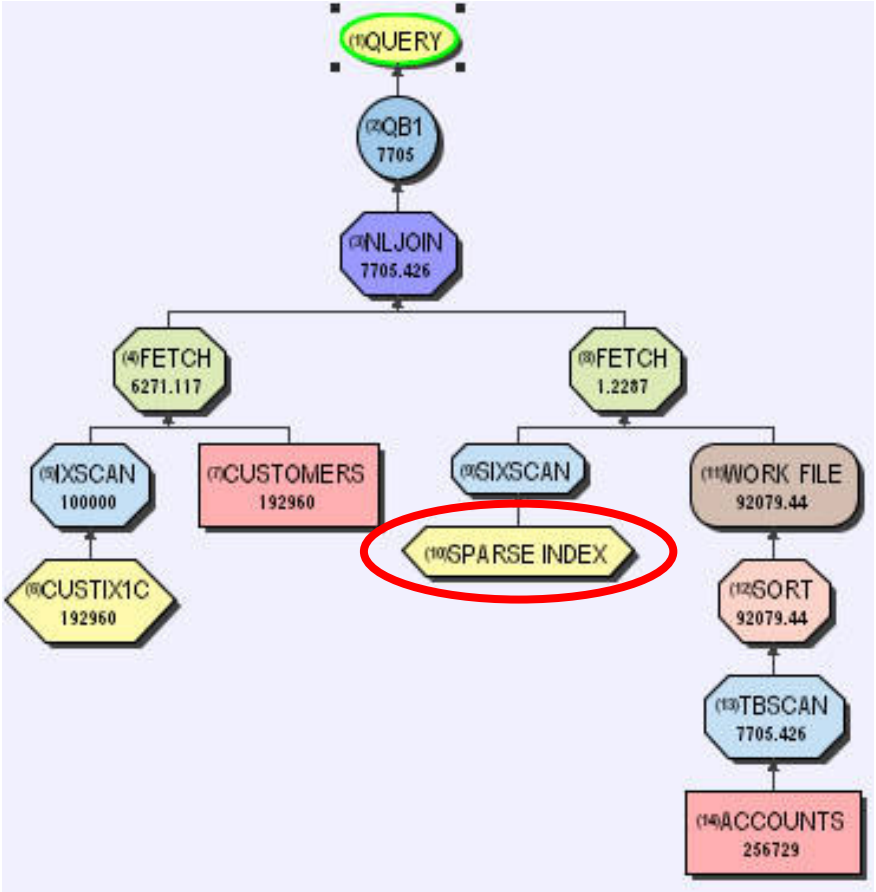
Sparse Indexes
A timeline

- V4 introduced sparse index
  - for non-correlated subquery workfiles
- V7 extended sparse index
  - for the materialized work files within star join
- V8 replaced sparse index
  - with in-memory data caching for star join
    - Runtime fallback to sparse index when memory is insufficient

Compensating for missing indexes

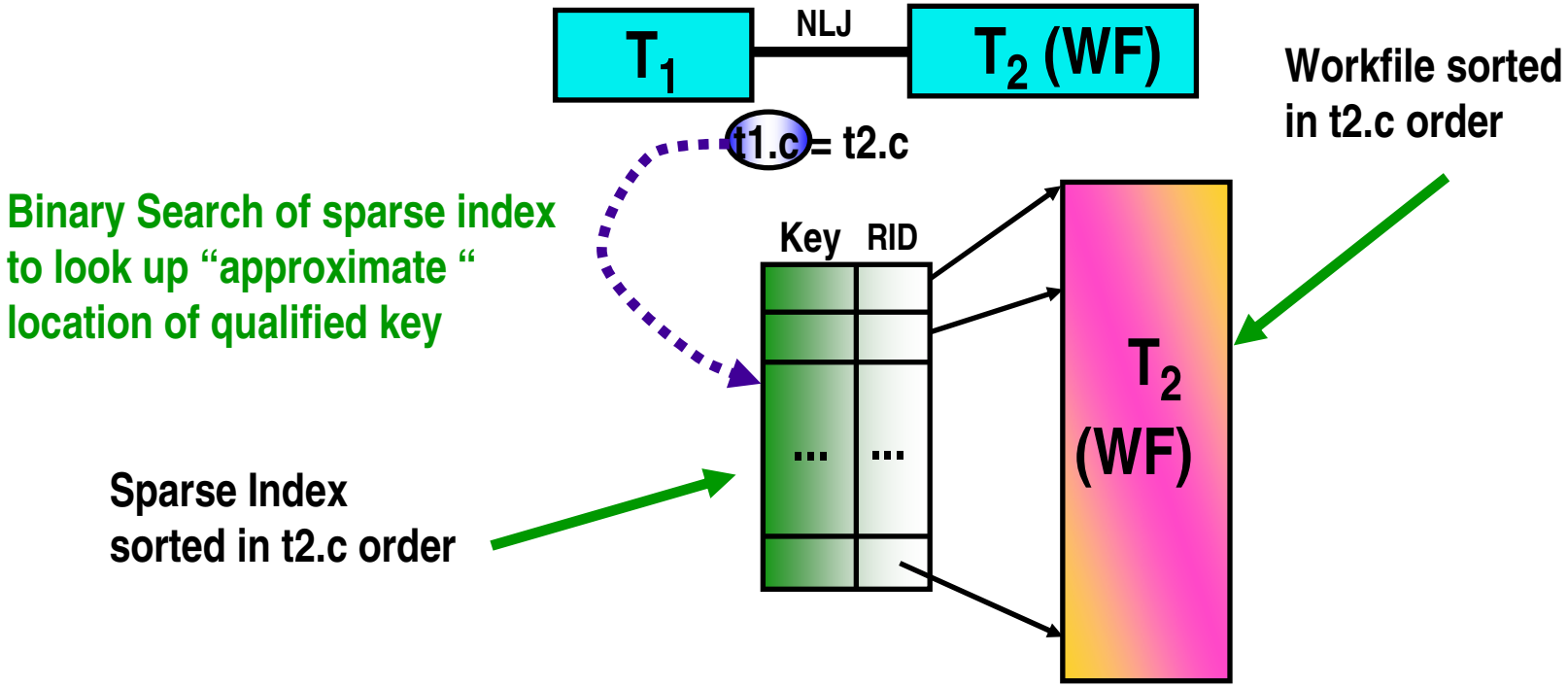
Generalized sparse indexes and IMWF

- If DB2 query optimizer doesn't find efficient indexes to support a join, it can create:
  - An "in-memory work file" (IMWF) to cache the entire inner table, OR
  - A "sparse index" atop a materialized workfile.
- Previously only available for Star Join



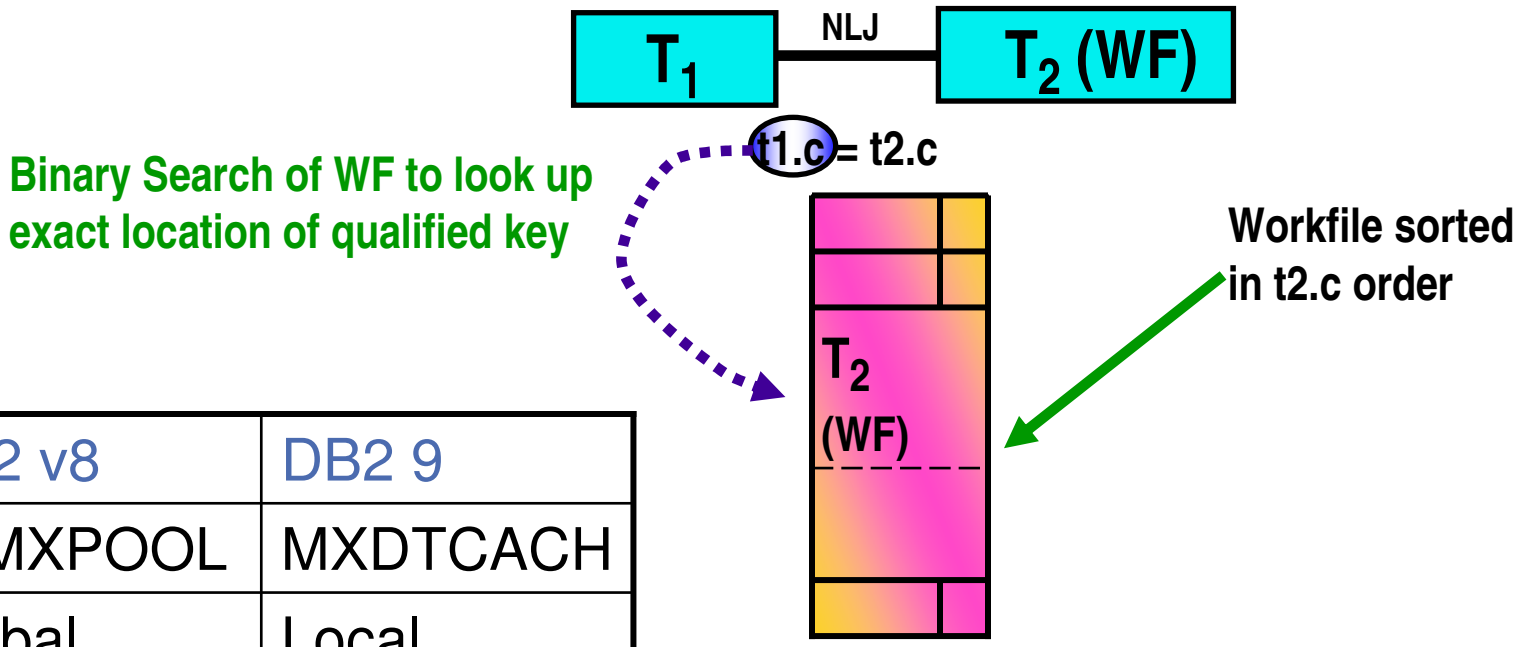
How does Sparse Index work?

- Sparse index may be a subset of workfile (WF)
  - Example, WF may have 10,000 entries
    - Sparse index may have enough space (240K) for 1,000 entries
    - Sparse index is “binary searched” to find target location of search key
    - At most 10 WF entries are scanned



How does In-Memory WF work?

- Whereas sparse index may be a subset of WF
  - IMWF contains the full result (not sparse)
  - Example, WF may have 10,000 entries
    - IMWF is “binary searched” to find target location of search key

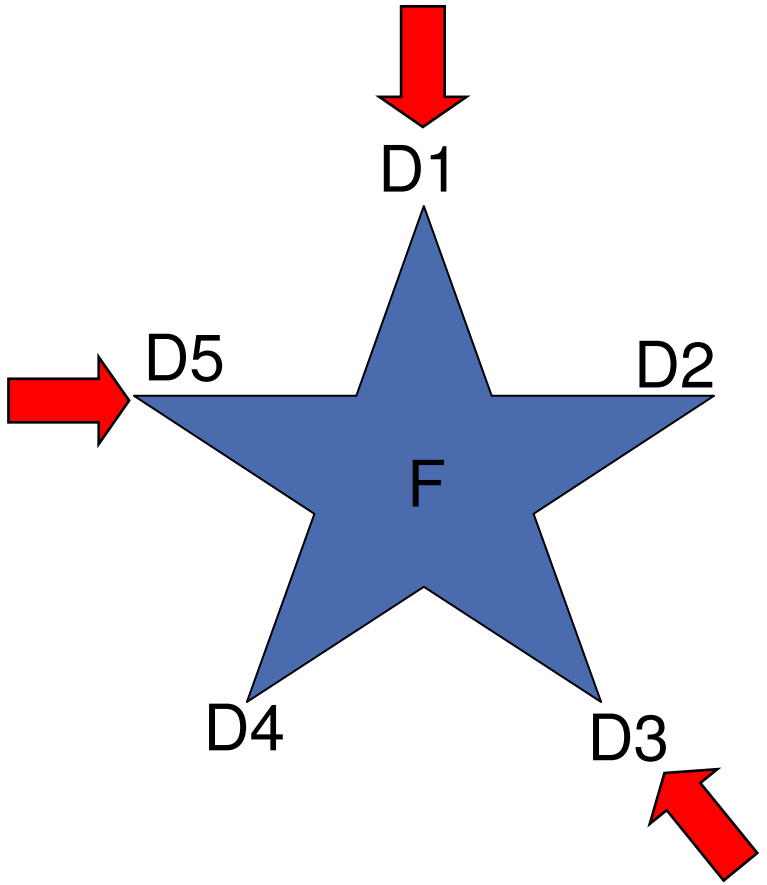


DB2 v8	DB2 9
SJMXPOOL	MXDTCACH
Global	Local

Star Join Enhancements

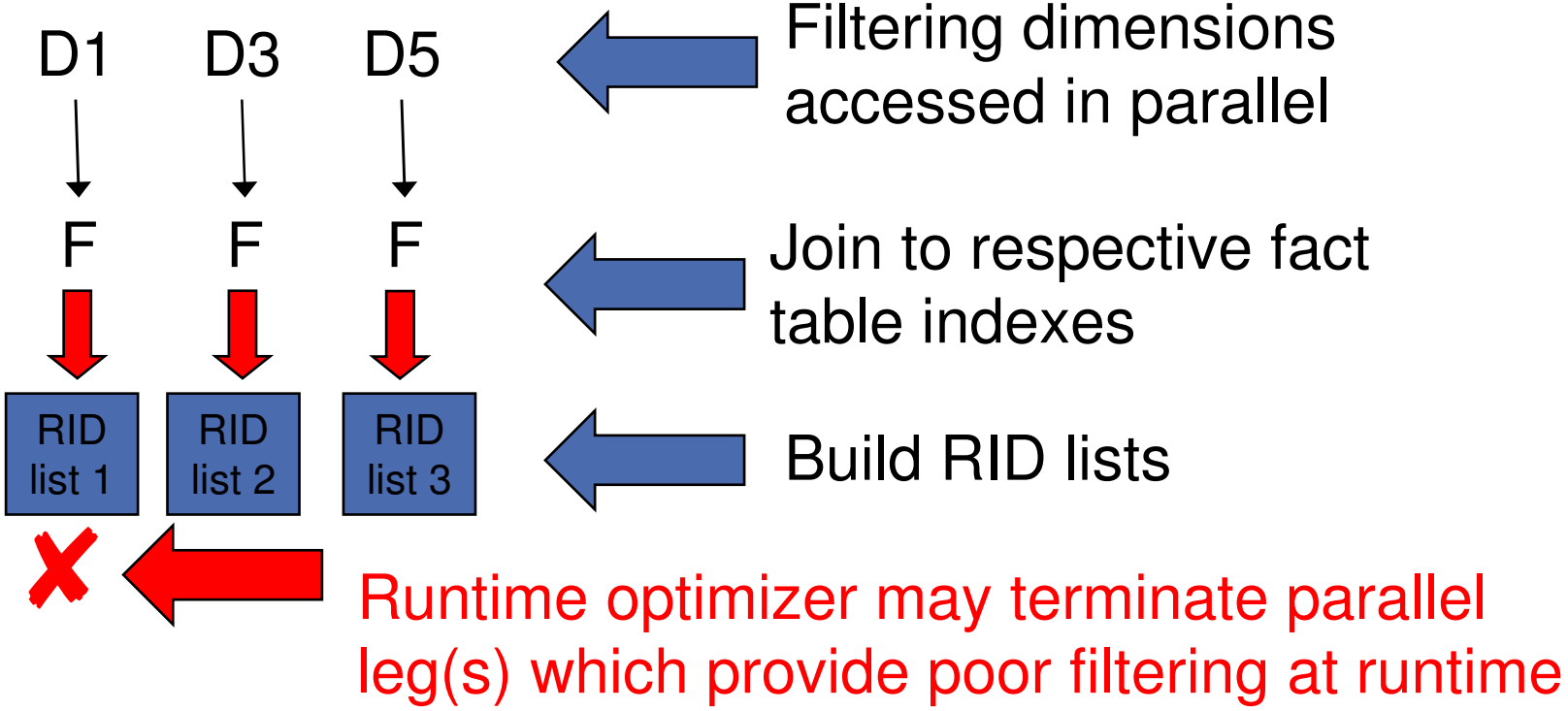
The Index ANDing Challenge

- Filtering may come from multiple dimensions
- In DB2 8, star join processing relies on the presence of multi-column indexes.
- However, creating multi-column indexes to support all useful combinations is often impractical.



Index ANDing – Pre-Fact

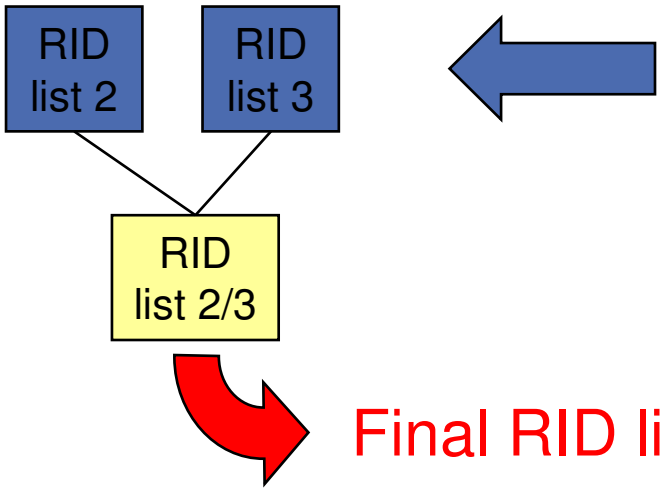
- Pre-fact table access
  - –Filtering may not be (truly) known until runtime



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



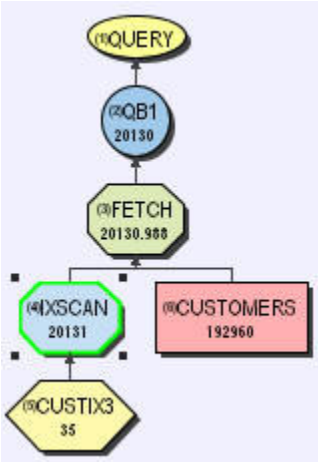
Remaining RID lists are "ANDed" (intersected)

Final RID list used for parallel fact table access

Index on Expression

- 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

Tracking Index Usage

- Additional indexes require overhead for
  - Utilities (REORG, RUNSTATS, LOAD etc)
  - Data maintenance (INSERT, UPDATE, DELETE)
  - Disk storage
  - Optimization time (Increases optimizer's choices)

- RTS records the index last used date
  - SYSINDEXSPACESTATS.LASTUSED
    - Updated once in a 24 hour period, but only if index used
  - "Used", as defined by DB2 as:
    - As an access path for query or fetch.
    - For searched UPDATE / DELETE SQL statement.
    - As a primary index for referential integrity.
    - To support foreign key access

Sort Avoidance

Improved DISTINCT and GROUP BY

- Improved Sort avoidance for DISTINCT
  - From V9, DISTINCT can avoid sort using duplicate index
    - APAR PK71121 – Avoid WF creation for zero rows
- Sort avoidance for GROUP BY
  - Order of GROUP BY columns re-arranged to match index
    - Data may be returned in a different order
      - Relational theory states that order is NOT guaranteed without ORDER BY

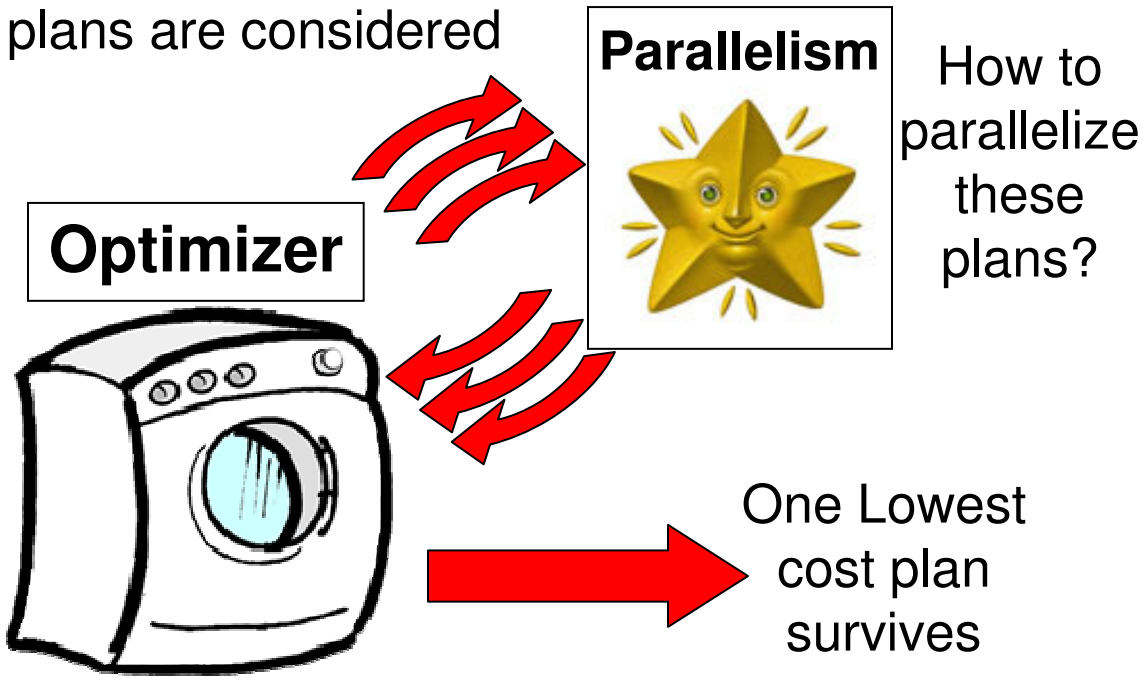
```
GROUP BY C2, C1 <=  GROUP BY in C2, C1 sequence

Index 1 (C1, C2)<=  Index in C1, C2 sequence
```

Query Parallelism Enhancements

- In V8
  - Lowest cost is BEFORE parallelism
- In DB2 9
  - Lowest cost is AFTER parallelism
    - Only a subset of plans are considered for parallelism

The end result?

More opportunities for parallel access paths!

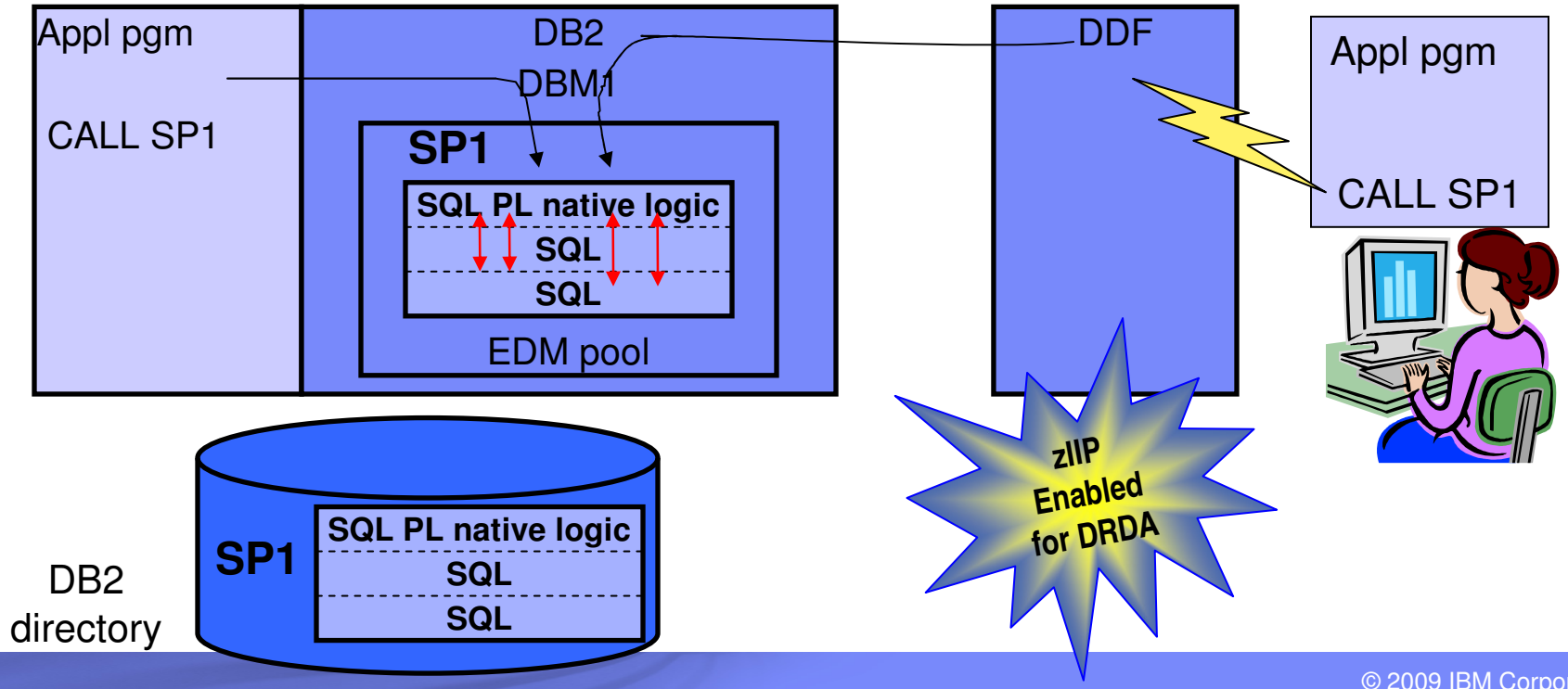


Query Parallelism Enhancements (contd.)

- In V8
  - –Degree cut on leading table (exception star join)
- In DB2 9
  - –Degree can cut on non-leading table
    - Benefit for leading workfile, 1-row table etc.
  - –Histogram statistics exploited for more even distribution
    - For index access with NPI
  - –CPU bound query degree <= # of CPUs * 4
    - <= # of CPUs in V8
- Increased parallelism results in greater zIIP offload

Native SQL Procedures

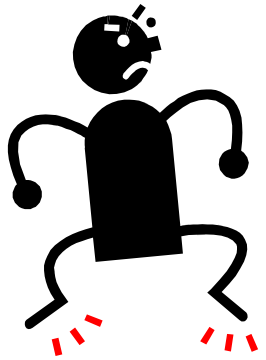
- Eliminates generated C code and compilation
- Fully integrated into the DB2 engine
- Extensive support for versioning
- Allow nested compound statements within a procedure



Favoring Index-only Access

- Ever created an index to support index-only?
  - Only to have optimizer choose index + data?

```
SELECT C2
FROM T1
WHERE C1 = ?

Index 1 (C1)          ←Index + data
Index 2 (C1, C2)      ←Index-only
```



- ZPARM OPTIXOPREF
  - Prioritize index-only over index + data with same index prefix
    - V8 APAR PK51734
    - V9 APAR PK77426 changes default to ENABLE

Reference

- Redbooks at www.redbooks.ibm.com
  - DB2 9 for z/OS Technical Overview SG24-7330
  - DB2 9 for z/OS Performance Topics SG24-7473
- DB2 for z/OS home page at www.ibm.com/software/db2zos
  - E-support (presentations and papers) at www.ibm.com/software/db2zos/support.html