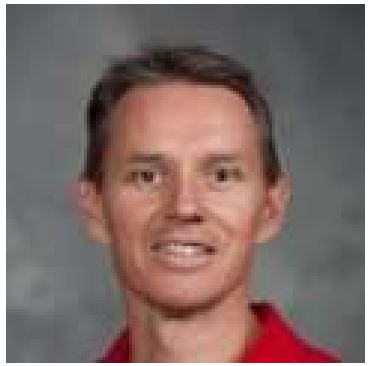
# DB2 for z/OS: Partitioning and Indexing – Query Performance Considerations



par Terry Purcell, IBM

GUIDE Share France
Une Association Indépendante d'Utilisateurs IBM

Réunion du Guide DB2 pour z/OS France
Mardi 18 novembre 2014
Tour Europlaza, Paris-La Défense

# Agenda

- **Introduction**
- **Matching index access**
- **Index screening**
- **Partitioning**
- **Conclusion**



# Introduction



# Indexing and I/O

- Main goal of an index is to improve performance for access to the data
  - By filtering rows in the index
- Number of data I/Os is dependent on
  - Number of rows retrieved
    - To reduce the number of I/Os, request less rows!!!
  - Filtering provided by
    - Matching predicates
    - Screening predicates
    - Partition pruning (page range screening)



# Indexing and I/O (cont.)

- Number of data I/Os is dependent on (cont.)
  - Organization of data
    - How clustering matches data access
  - Random vs Sequential access
    - List prefetch makes random I/Os sequential
      - But List prefetch cannot change location of rows
    - Dynamic/Sequential prefetch can read more pages than necessary
    - FARINDREF will increase random I/Os
      - Variable length (or compressed) rows updated that do not fit in their current page will be relocated.



# Indexing and I/O – Table Joins

- For joins, number of I/Os is dependent on
  - Filtering occurring early in the join sequence
  - Random vs Sequential access
    - Sort of the composite permits sequential access to inner index and possibly table too
    - Hybrid join can improve random I/O to inner table
      - Nested loop join will perform list prefetch on inner once per outer row
      - Hybrid join consolidates RIDs on inner to perform 1 consolidated list prefetch request.
    - Merge scan join performs the join using sequential match/merge



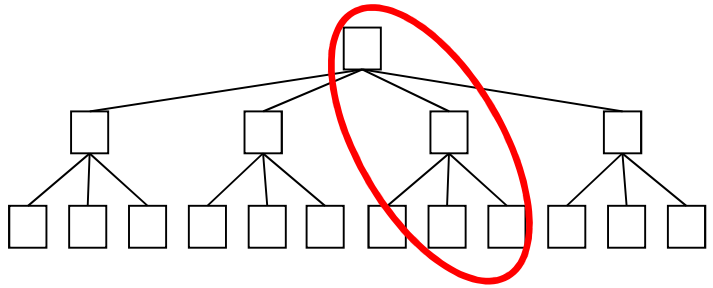
# Matching index access



# Index matching

- Index matching restricts access to a subset of the index
  - Reducing index I/O
    - If 5% of the index rows qualify, then approx 5% of the index pages are read
  - Generally results in reduction in data I/O

```
SELECT *
FROM PHONE_BOOK
WHERE LASTNAME = 'PURCELL'
```

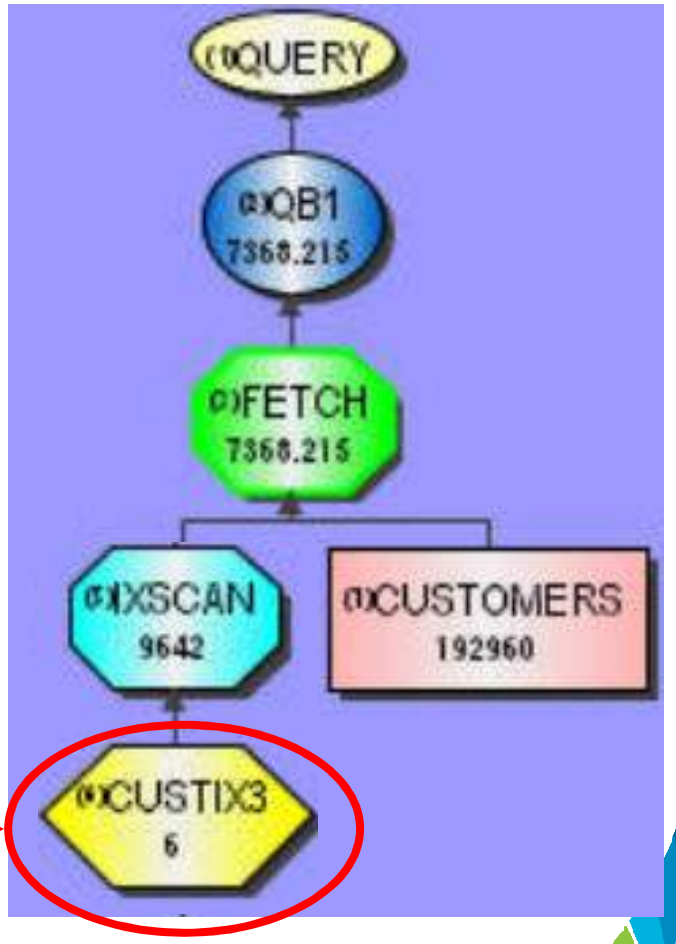


# Index matching to reduce I/O

- Example had Matchcols = 3 given 3 predicates
  - Optimal index choice

```
SELECT *
FROM CUSTOMERS
WHERE COUNTRY = 'USA'
  AND GENDER = 'F'
  AND STATUS = 'N'
OPTIMIZE FOR 1 ROW
```

```
INDEX CUSTIX3 (COUNTRY ASC
              ,STATUS ASC
              ,GENDER ASC)
```



# How many rows qualify?

- Count shows 1,121 rows qualifying
  - Given that there are 192,960 rows in the table
    - That's 0.6% of the table….excellent filtering!!!

```
SELECT COUNT(*) = 1,121
FROM CUSTOMERS
WHERE COUNTRY = 'USA'
  AND GENDER = 'F'
  AND STATUS = 'N'
```



# How many data pages are accessed?

- 0.6% of the table qualifies……..but……
  - It is 0.6% of the rows
  - How many data I/Os are required to retrieve these rows?

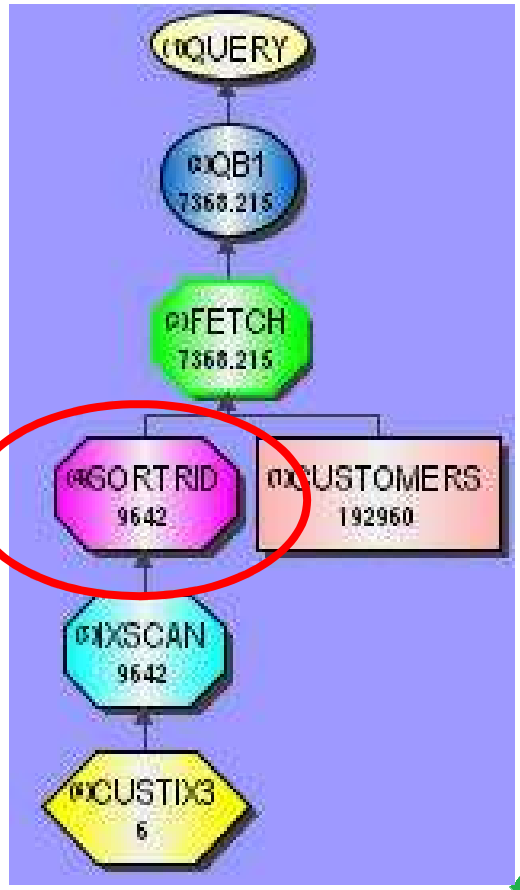
```
BP1     BPOOL ACTIVITY      TOTAL
--------------------------  --------
GETPAGES                      1121
```

1,121 data getpages to retrieve 1,121 rows

# How to reduce the number of I/Os?

- By removing the OPTIMIZE clause
  - List prefetch can be used
  - May reduce the number of I/Os
    - Since duplicate pages are accessed only once

```
SELECT *
FROM CUSTOMERS
WHERE COUNTRY = 'USA'
  AND GENDER = 'F'
  AND STATUS = 'N'
```



# Is list prefetch enough?

- 0.6% of the table qualifies……..but……
  - It is 0.6% of the rows
  - How many data pages do these rows occur on?

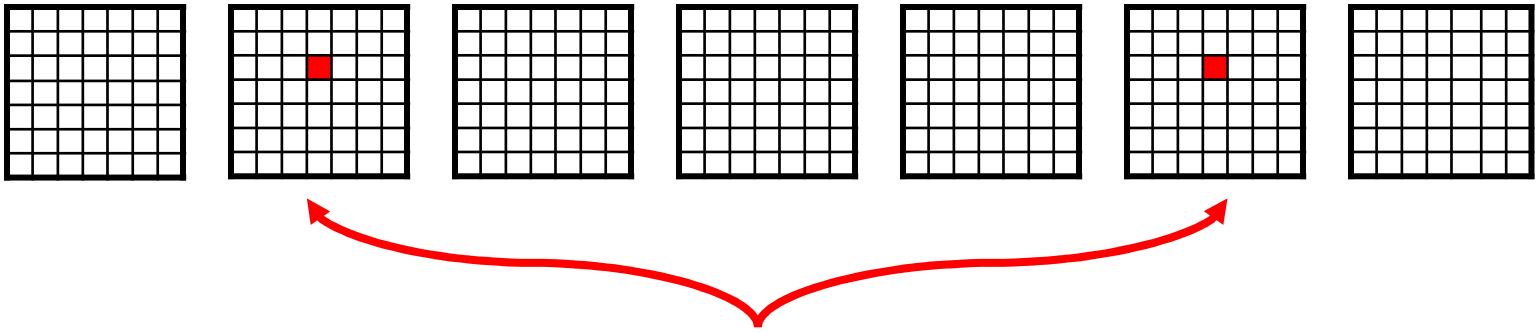
```
BP1     BPOOL ACTIVITY      TOTAL
---------------------   ---------
GETPAGES                      969
```

969 data getpages to retrieve 1,121 rows



# Current clustering effect

- Table is currently clustered by the unique key
  - ACCTNO
- When retrieving a large number of rows unrelated to ACCTNO, qualified rows are found on many pages
  - 969 getpages from 4020 pages total



On average, 1.2 rows found on every 4th page

# Data clustering vs scattering

- 1,121 rows qualified, and 48 rows per page
- Number of data pages accessed via index:
  - Min # pages (clustered) = 1121/48 = 24
  - Max # pages = 1121
    - Pages may or may not be consecutive
    - Actual max getpages is 1121 * 2 = 2242
      - If updated row is relocated because it does not fit on original page (NEARINDREF/FARINDREF)
  - # of getpages
    - = # of qualified pages if list prefetch used (+ # of INDREF rows)
    - <= # of qualified rows (* 2) for random I/O



# After re-clustering

- After clustering by the data access:
  - STATUS, GENDER, ACCTNO
- Number of getpages  25
  - Compared with original 969

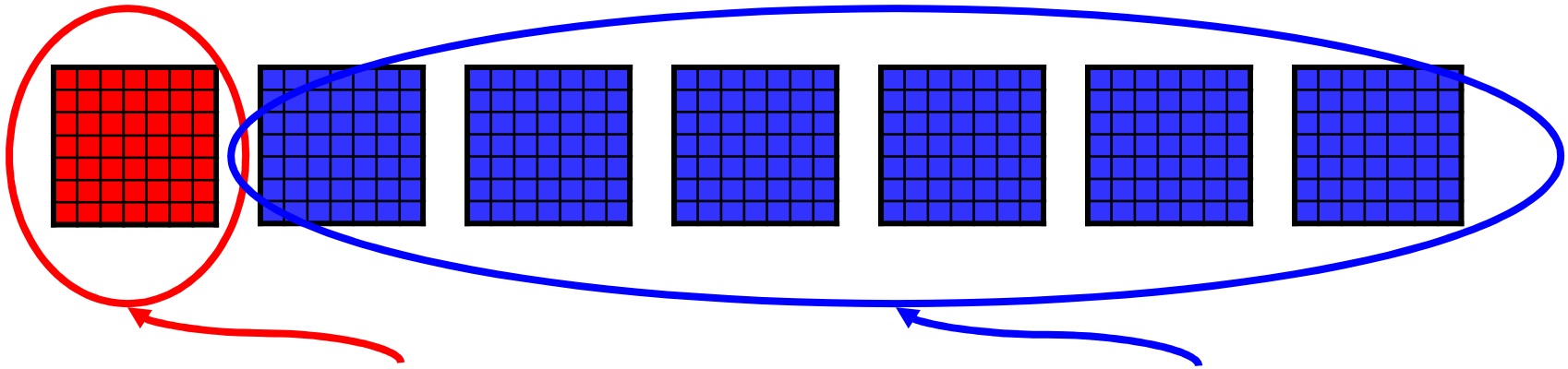
```
BP1     BPOOL ACTIVITY      TOTAL
---------------------   --------
GETPAGES                      25
```

- Note: ACCTNO was added to clustering index so that original clustering is maintained within STATUS/GENDER



# New clustering effect - STATUS

- Clustered by STATUS, GENDER, ACCTNO
  - Status = 95% Y, 5% N
    - The clustering effect is (example not to scale):

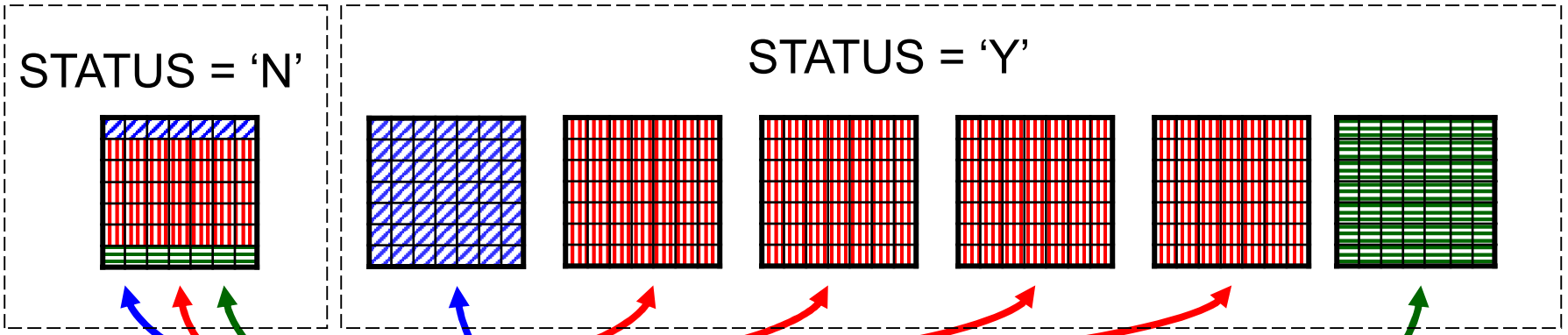


WHERE STATUS = 'N'
accesses only these pages

WHERE STATUS = 'Y'
accesses only these pages

# New clustering effect - GENDER

- Clustered by STATUS, GENDER, ACCTNO
  - Gender = 70% M, 17.5% NULL, 12.5% F
    - GENDER is clustered within each STATUS value
    - The clustering effect is (example not to scale):




GENDER = 'M'
appear on these pages

GENDER = 'F'
appear on these pages

GENDER IS NULL
appear on these pages

# Clustering vs Clusterratio

- Clustering is the density of the data
- Clusterratio implies the sequential nature of the data
  - For the clustering index, clusterratio and clustering are equivalent
  - For the non-clustering index, high clusterratio could mean
    - the data is clustered (dense)
    - OR, the data is sequential but not dense
- DB2 9 RUNSTATS collects data to distinguish these for optimizer
  - Zparm STATCLUS=ENHANCED (default) – Strongly recommended



# Index Screening

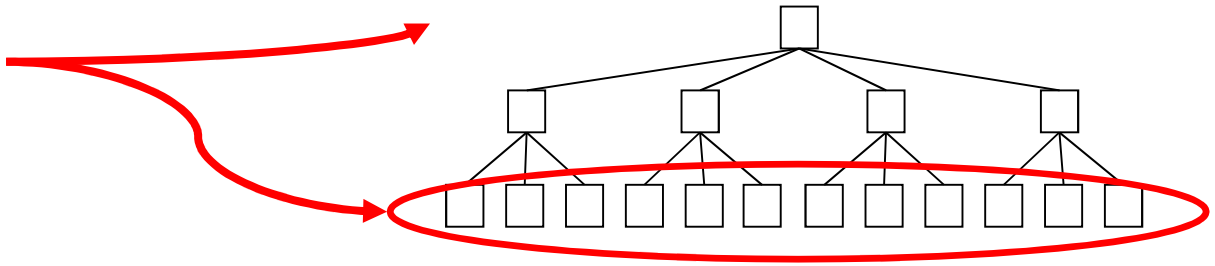


# Index screening

- Index screening does not reduce the number of index rows/pages read
  - Can reduce the number of data rows accessed
  - May or may not result in reduction of data I/O

```
SELECT *
FROM PHONEBOOK
WHERE FIRSTNAME = 'TERRY'
```

No predicate on LASTNAME, so whole index must be read



# Index screening to reduce I/O

- Before reclustering from prior example
  - 1 screening predicate on GENDER
    - Tablespace scan chosen

```
SELECT *
FROM CUSTOMERS
WHERE GENDER = 'F'
```

```
INDEX CUSTIX3 (COUNTRY ASC
              ,STATUS ASC
              ,GENDER ASC)
```

Tablespace scan chosen instead of Non-matching index scan

# How many rows qualify?

- Count shows 24,393 rows qualifying
  - Given that there are 192,960 rows in the table
    - That's 12.64% of the table….OK filtering!!!

```
SELECT COUNT(*) = 24,393
FROM CUSTOMERS
WHERE GENDER = 'F'
```



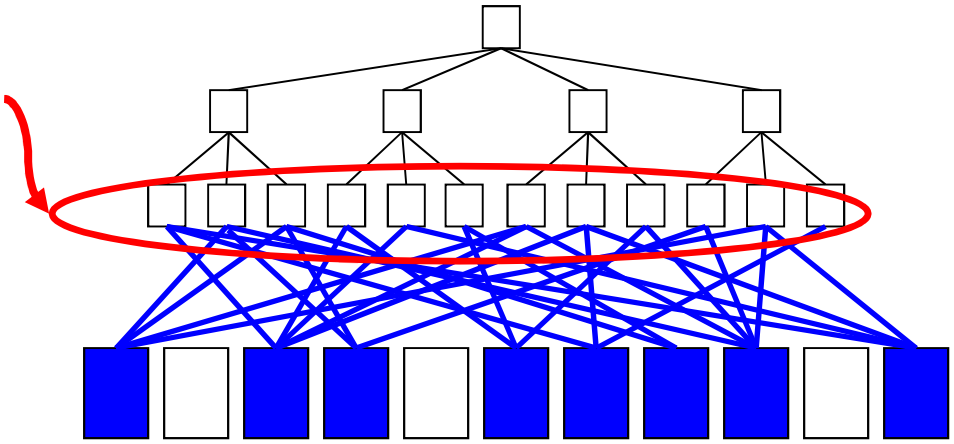
# How many pages do the rows appear on?

- Used OPTHINT to force non-matching index scan

```
BP1     BPOOL ACTIVITY      TOTAL
--------------------------  --------
GETPAGES                    18683
```

  - 18683 getpages to retrieve 24,393 rows
    - Many pages repeatedly accessed
      - Table has 4020 pages

Non-matching index scan



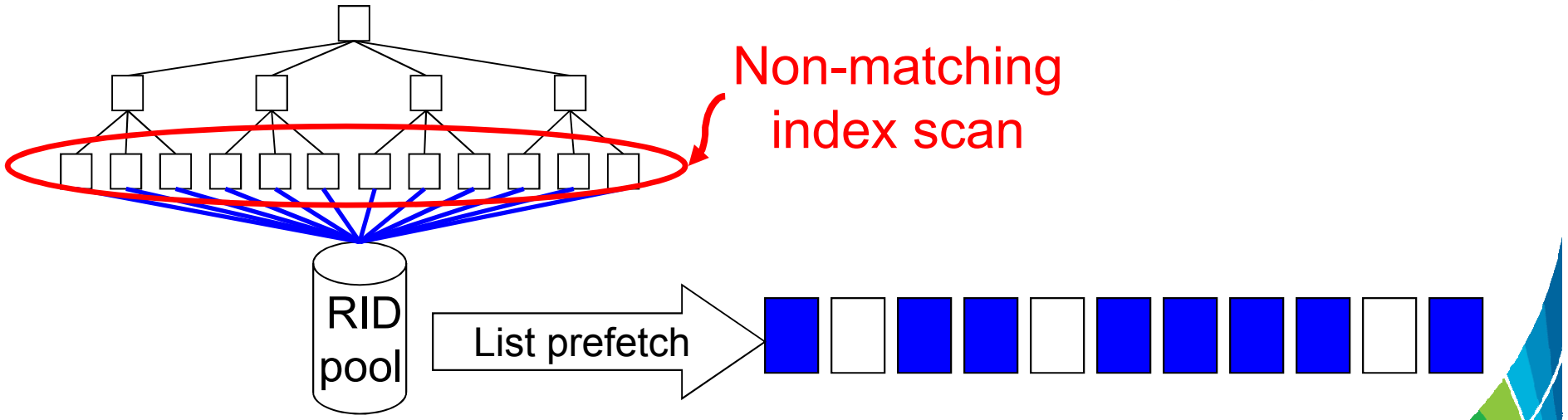


# How many pages do the rows appear on?

- Use OPTHINT to force list prefetch

```
BP1     BPOOL ACTIVITY      TOTAL
---------------------   --------
GETPAGES                    3807
```

  - 3807 getpages to retrieve 24,393 rows
    - GENDER = 'F' appear on most data pages (94.7%)



# After re-clustering

- After clustering by the data access:
  - STATUS, GENDER, ACCTNO
- Number of getpages  509
  - Compared with previous 3807 (with list prefetch)

```
BP1     BPOOL ACTIVITY      TOTAL
---------------------   --------
GETPAGES                      509
```

  - GENDER rows are now clustered within STATUS
    - Rather than scattered throughout all pages



# Matching vs Screening

- The number of qualified rows from the index is dependant on filtering
  - Regardless whether from matching or screening
  - Number of data pages accessed depends on
    - Placement of data (clustering)
    - Ordering of access from index to data
      - High clusterratio implies sequential order
      - List prefetch can “sequentialize” I/O
- Matching can result in a subset of the index accessed
  - Screening does not limit the subset of the index



# Partitioning and indexing (PI, DPSI, and NPI)



# Clustering vs partitioning

- Clustering is a logical grouping
  - Inserts attempt to maintain clustering sequence
- Partitioning is a physical grouping
  - Inserts guarantee that rows can only be inserted into the partition dictated by the partition limit keys
  - Partitioning pruning can be exploited without indexing
  - Can support efficient insert-at-end processing within partitions
  - Freespace search is limited to a partition
- Combine partitioning and clustering (and also UNION ALL in View) for multi-dimensional clustering



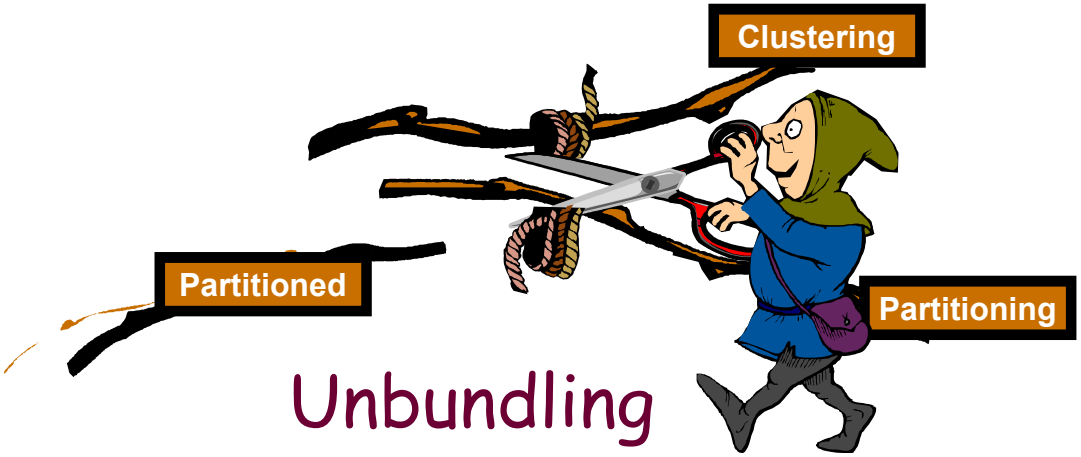
# Table controlled partitioning

- Introduced in DB2 V8
- Unbundling Partitioned Table Attributes
  - Partition without an index
    - Table controlled partitioning
  - Data Partitioned Secondary Index
  - Cluster on any index
- Prior to V8, only index-controlled partitioning supported
  - Many tables may remain as index-controlled

# Table-controlled partitioning

- Partitioning clause moved to CREATE TABLE

```
CREATE  TABLE  CUSTOMER  (
          ACCOUNT_NUM INTEGER,
          CUST_LAST_NM CHAR(30),
          . . .
          LAST_ACTIVITY_DT DATE,
          STATE_CD CHAR(2))
       PARTITION BY ( ACCOUNT_NUM ASC )
       ( PARTITION  1   ENDING AT (199),
         PARTITION  2   ENDING AT (299),
         PARTITION  3   ENDING AT (399),
         PARTITION  4   ENDING AT (499)  ) ;
```



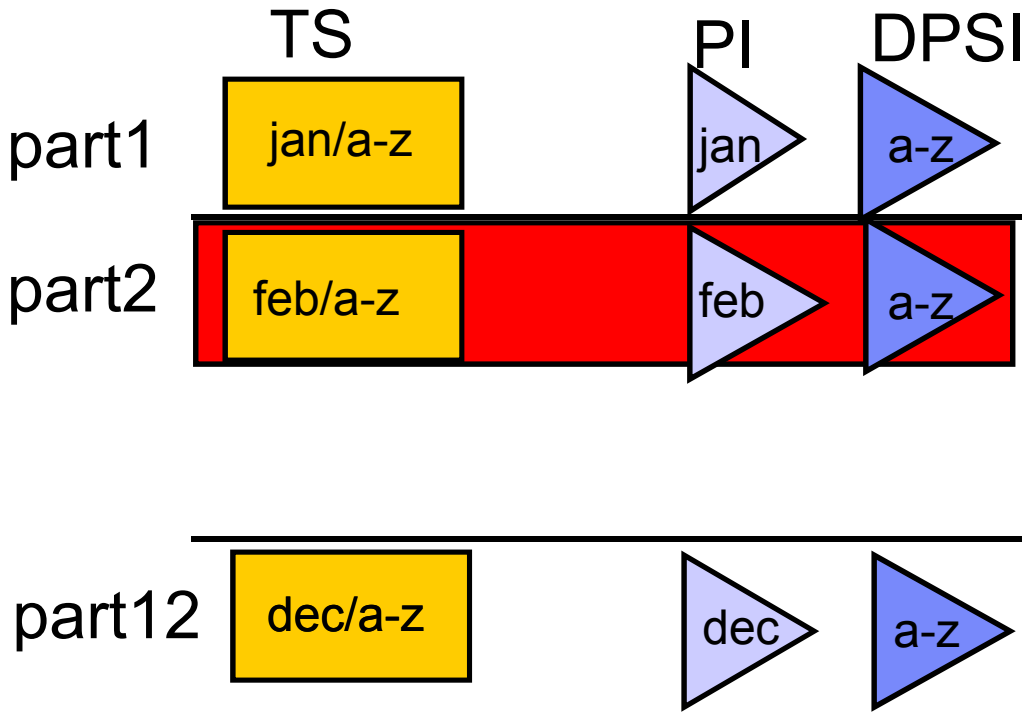

No indexes are required for partitioning!!

TB

| | | | |
|---|---|---|---|
| 101 | 201 | 301 | 401 |
| 102 | 202 | 302 | 402 |
| 103 | 203 | 303 | 403 |
| 104 | 204 | 304 | 404 |
| 105 | 205 | 305 | 405 |
| 106 | 206 | 306 | 406 |

Partitioned table

# Example: Table-based partitioning




- Partition data by month (PI is optional!)
- Clustering by id or name (DPSI clustering)
- No NPI's to deal with, true partition independence

# Version 8+ classification of indexes

- An index may / may not be correlated with the partitioning columns of the table
  - **Partitioning index (PI)**
  - Secondary index
- An index may / may not be physically partitioned
  - Partitioned
  - Non-partitioned
- Clustering index:
  - Any index may be the clustering index
  - The clustering index can be non-unique



# Partitioning indexes

- A partitioning index
  - Has the same leftmost columns as the columns which partition the table
  - These columns have the same collating sequence (ASC / DESC)

```
CREATE TABLE CUSTOMER (
        ACCOUNT_NUM                     INTEGER,
        LAST ACTIVITY_DT        CHAR(3),
        CCODE                                   CHAR(2),
        . . .                                                   )
        PARTITION BY (ACCOUNT_NUM               ASC)
        ...
CREATE . . . INDEX  part_ix_1  ON CUSTOMER (
        ACCOUNT_NUM                     ASC)
```

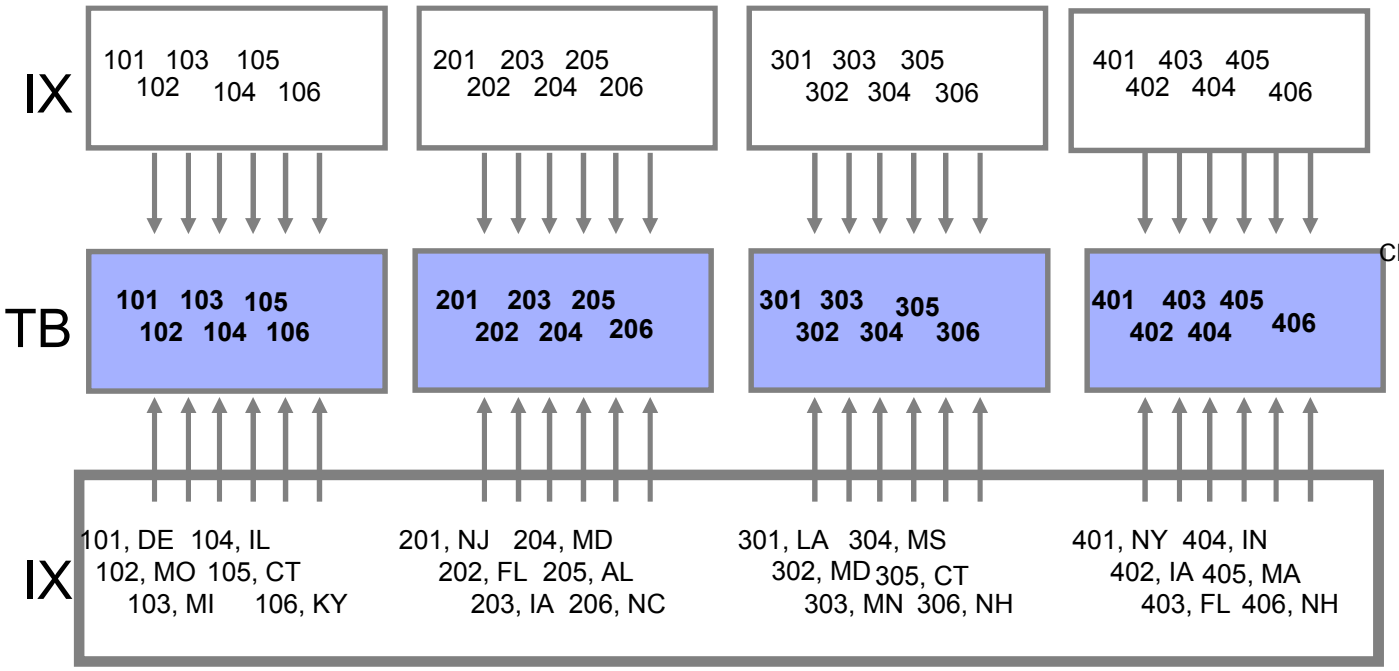
**Partitioning does not equal partitioned!!! But, all partitioning indexes should be partitioned.**

# Partitioning indexes -2

A partitioning index has the same leftmost columns, in the same collating sequence, as the columns which partition the table

- NOTE: IX_1 is partitioned, IX_2 is not (why would you not partition?)

Partitioning index part_IX_1 (ACCOUNT_NUM ASC)



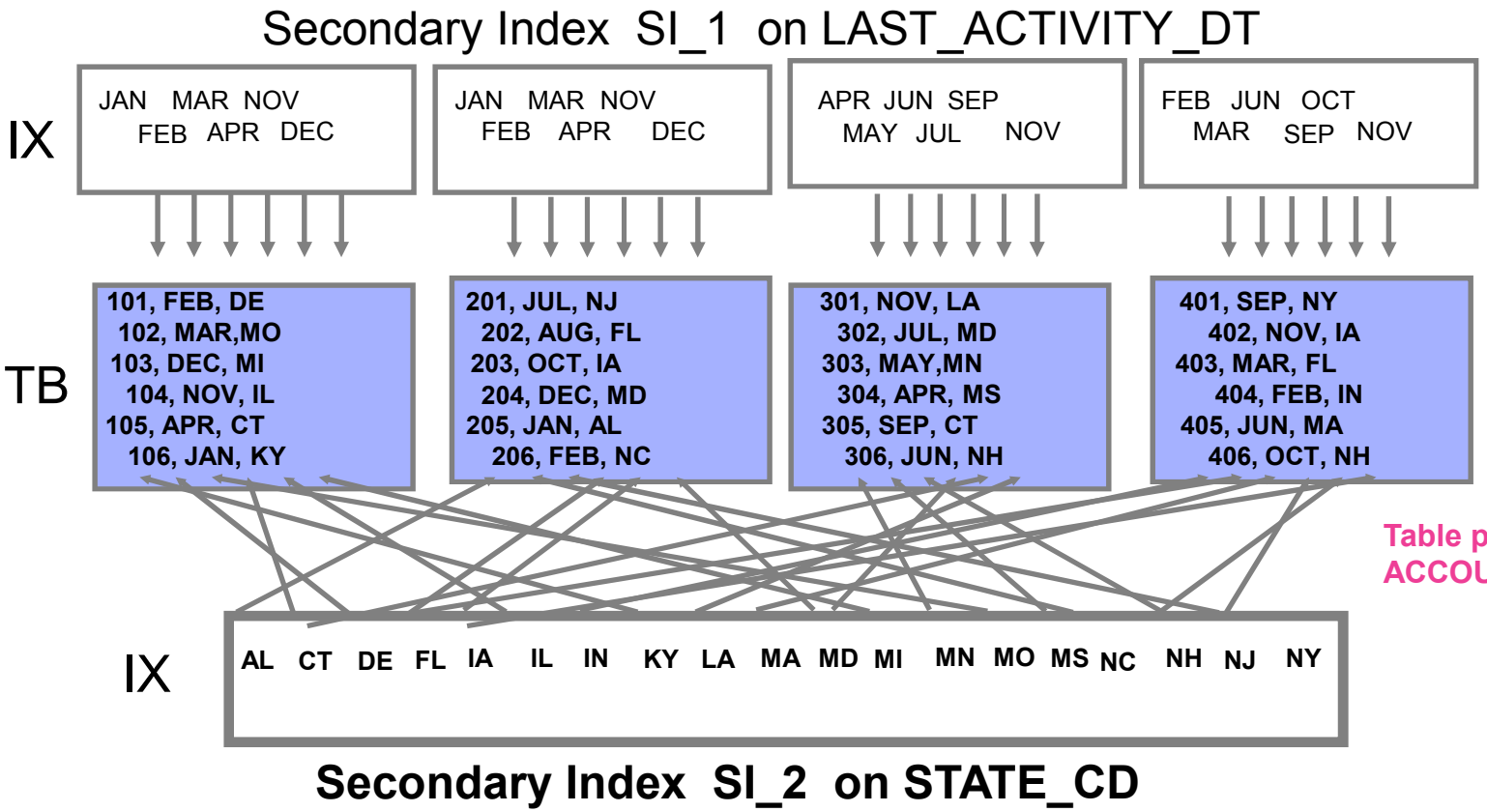

```
CREATE TABLE CUSTOMER (
    ACCOUNT_NUM
            INTEGER,
    CUST_LAST_NM
            CHAR(30),
    . . .
            )
    PARTITION BY (
        ACCOUNT_NUM      ASC)

    . . .
```

Partitioning index part_IX_2 (ACCOUNT_NUM ASC, STATE_CD) [not partitioned]

# Secondary indexes (Partitioned & Non-Partitioned)

A secondary index is any index which is **not** a partitioning index

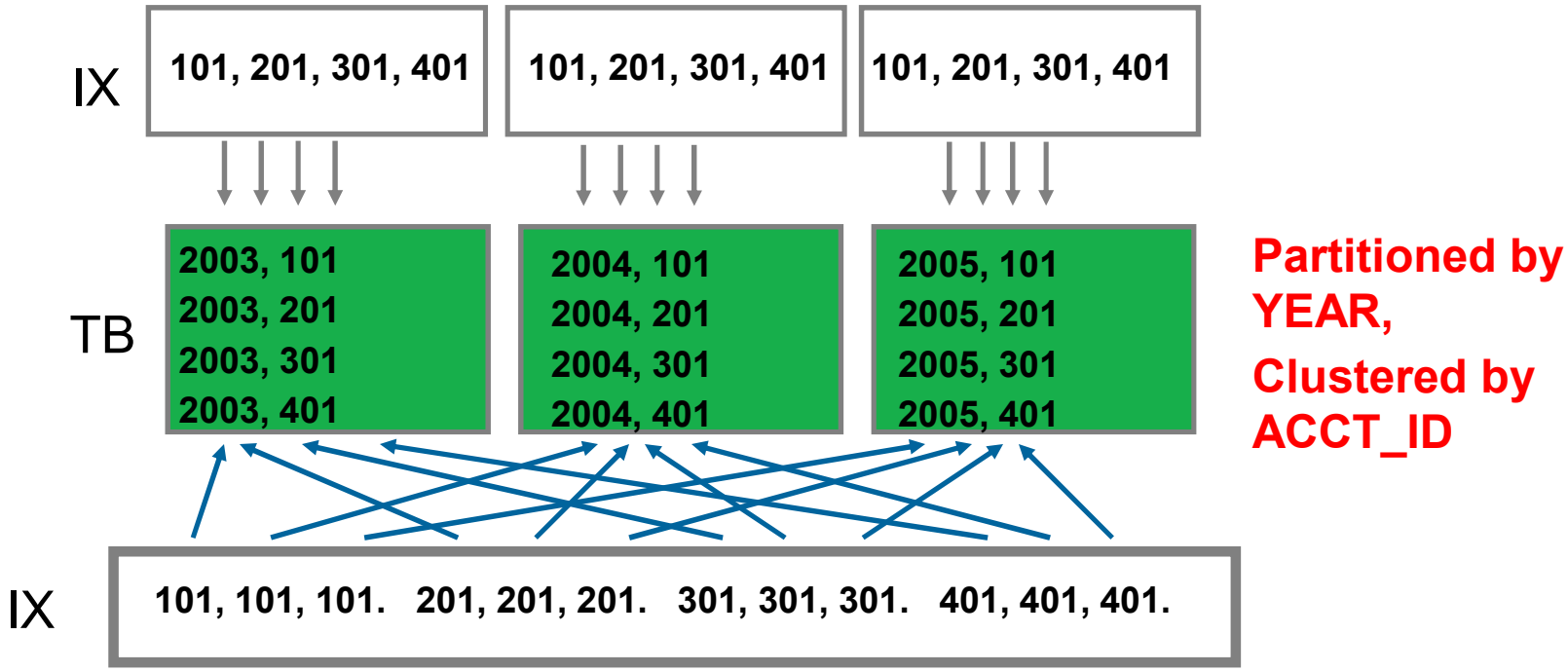
- Can be partitioned – SI_1 known as Data Partitioned Secondary Index (DPSI)
- Or not partitioned – SI_2 – Non-partitioned Secondary Index (NPSI or NPI)



# Partitioning/Clustering Considerations

- Index Clusterratio may differ depending on DPSI or NPI
  - Even if this is the CLUSTERing index

**DPSI on ACCT_ID – Clustering perfectly aligned with partitions**



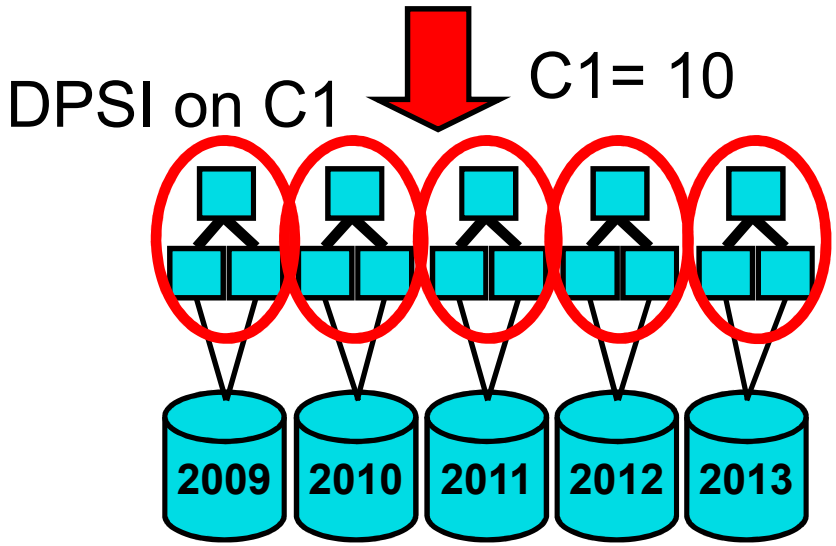

**Partitioned by YEAR,**

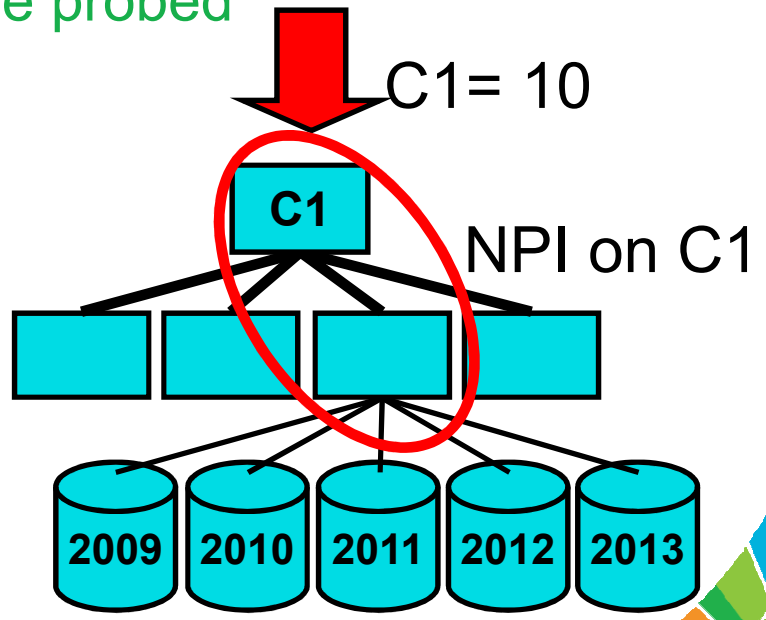
**Clustered by ACCT_ID**

**NPI on ACCT_ID – Clustering not aligned, keys cross partitions**

# Index Access - DPSI vs NPI

- Access to a secondary index without specifying a range de-limiting predicate results in:
  - For DPSI
    - All b-tree structures (up to 4096) must be probed
  - For NPI
    - Only one b-tree structure must be probed

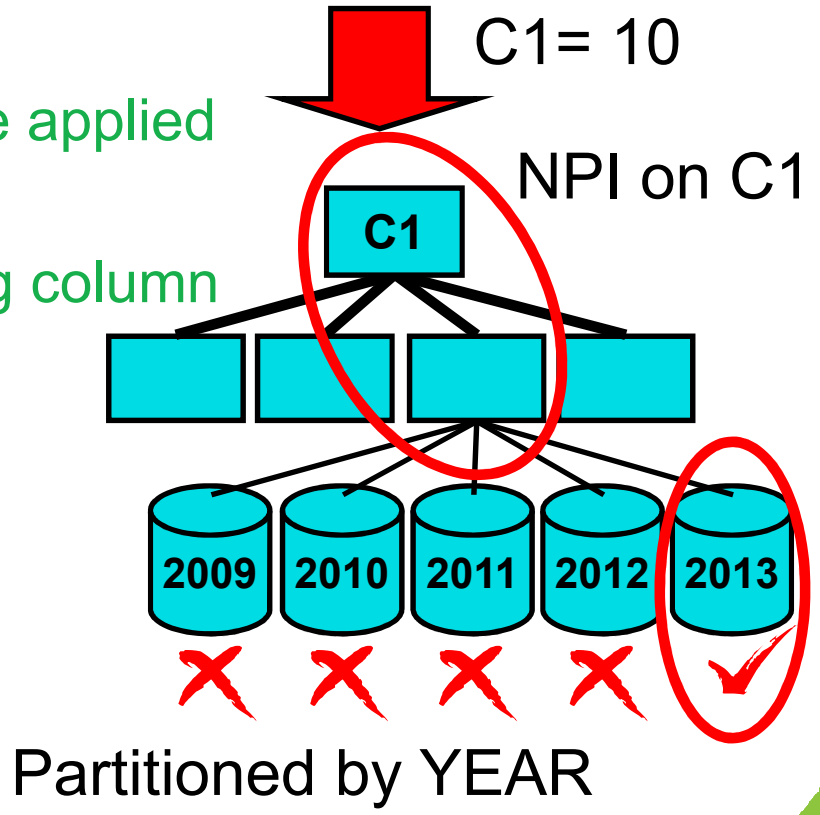


# Page Range Screening - NPI

- Page Range Screening can be applied
  - before data access on an NPI to limit the partitions accessed
    - if a predicate exists that can be applied
  - Similar to index screening
    - Without requiring the screening column to be indexed

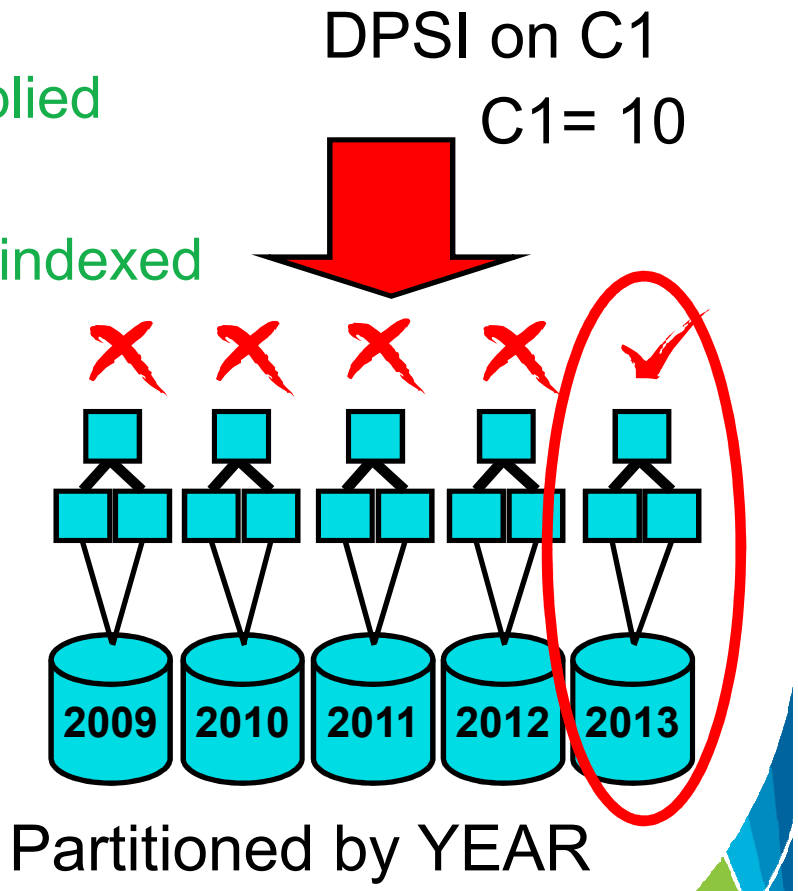
```
SELECT cols
FROM T1
WHERE C1 = 10
AND YEAR = 2013
```



# Page Range Screening - DPSI

- Page Range Screening can be applied
  - When index access occurs
    - if a predicate exists that can be applied
  - Similar to index matching
    - Without requiring the column to be indexed
  - Effect is 2 matching columns
    - 1 from index, 1 from partition
  - Can also function independently

```
WHERE C1 = 10
AND YEAR = 2013
```

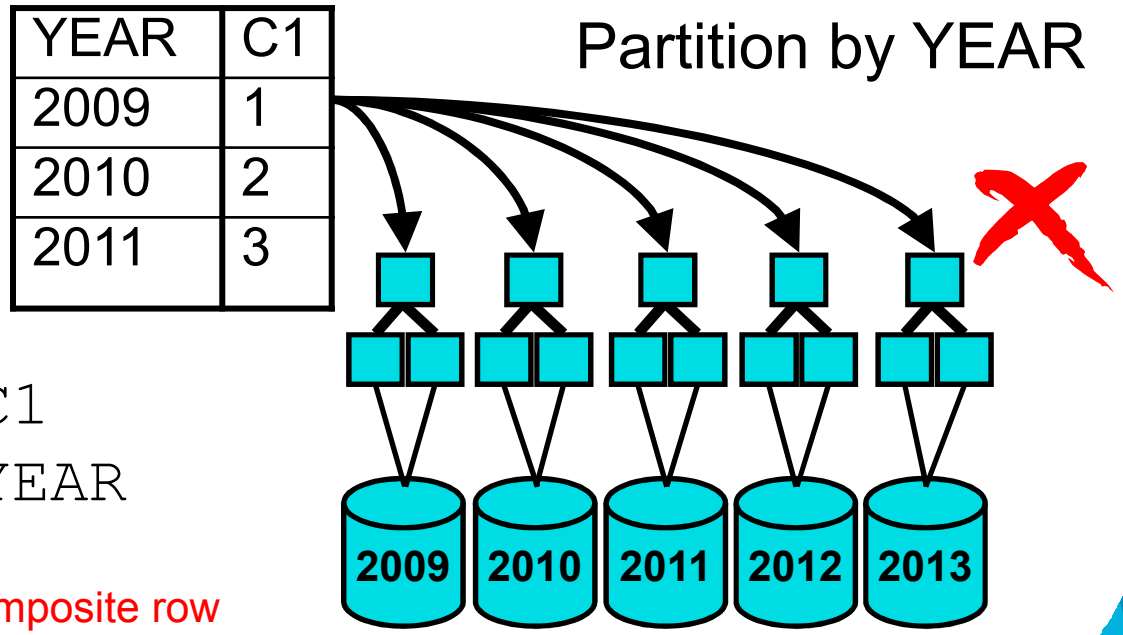


# Pre-V11 DPSI Join Probing (Join on partitioning Col)

- Current challenge
  - 1st composite row probes all parts
  - 2nd composite row probes all parts
  - Etc

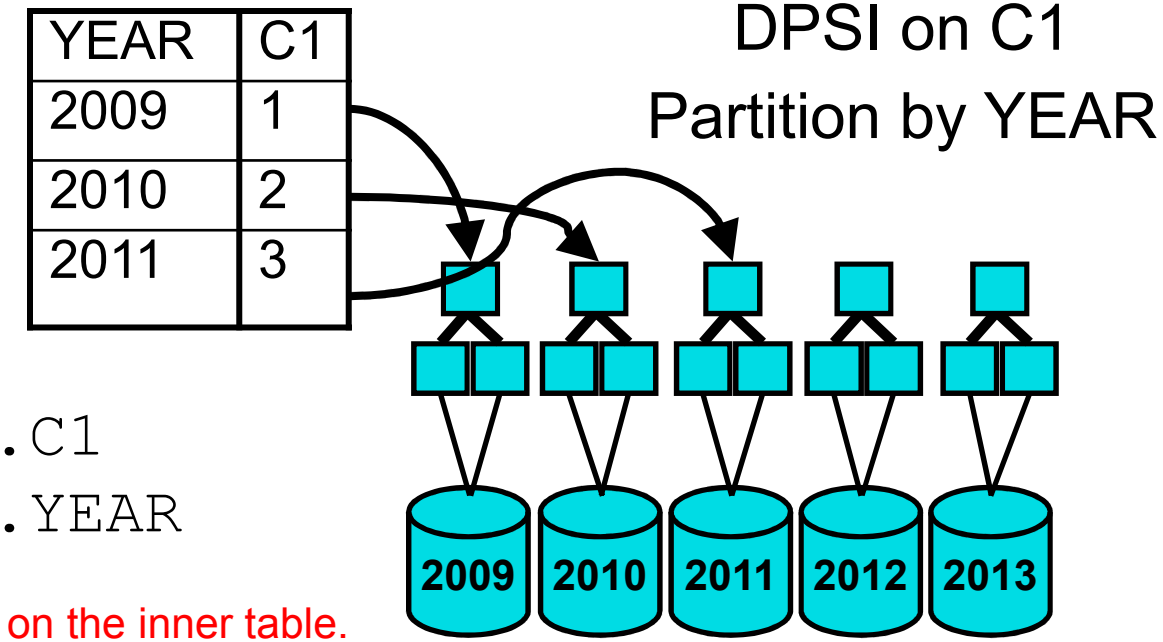
```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.YEAR = T2.YEAR
```

All parts are accessed for each composite row



# V11 DPSI Join Probing (Join on Partitioning Col)

- Join recognizes page range screening
  - 1st composite row probes 1 part.
  - 2nd composite row probes 1 part.
  - And so on.



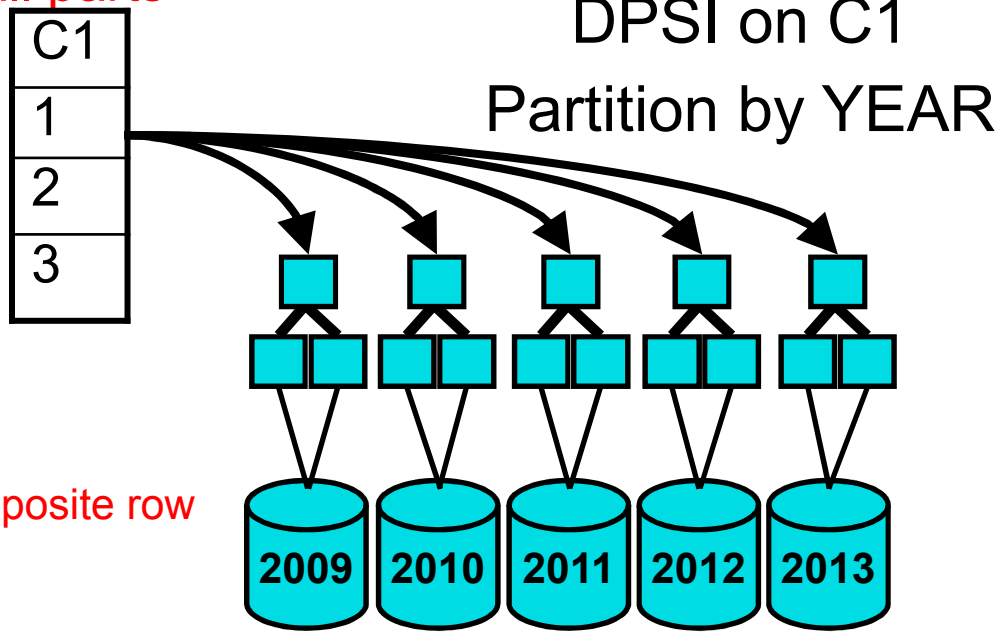

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.YEAR = T2.YEAR
```

Only qualified parts are probed on the inner table.

# Pre V11 DPSI Probing Challenge for Joins

- NOTE: No page range join predicate
- Current challenge for join to a DPSI
  - 1st composite row probes all parts
  - 2nd composite row probes all parts
  - Etc

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
```



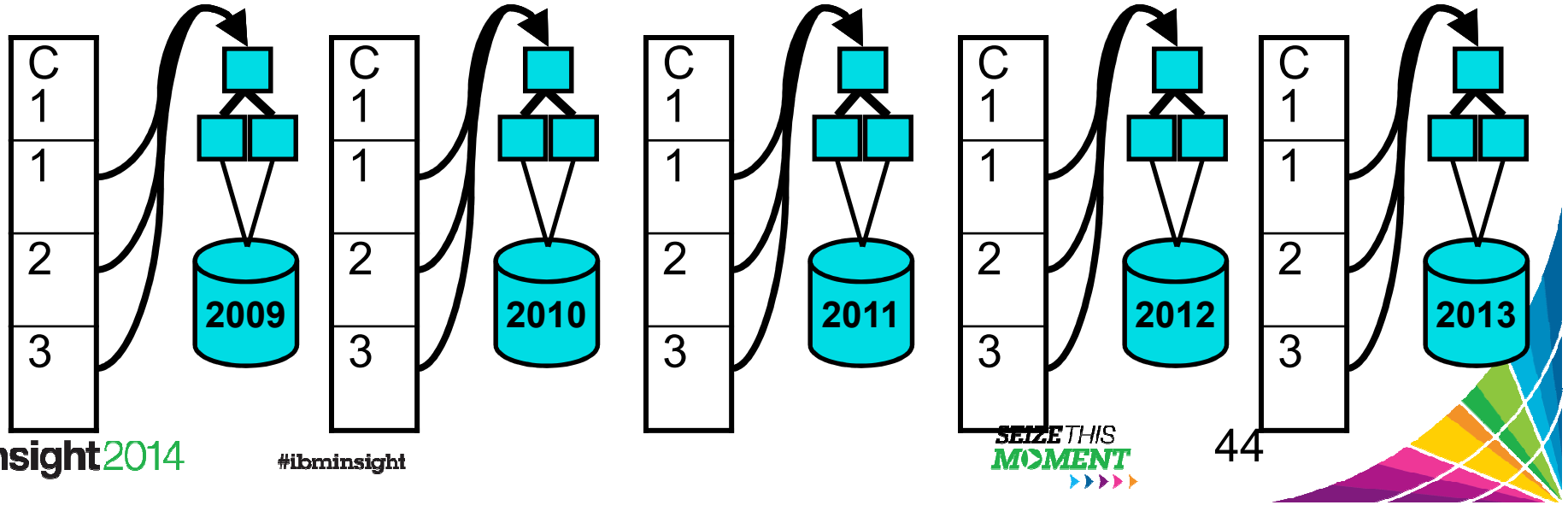

All parts are accessed for each composite row in a round robin fashion

# DPSI Probing – DB2 11 Join Solution

- DB2 11 solution – DPSI part-level Nested Loop Join
  - Repeat composite table for each child task
    - Each child task is a 2 table join
    - Allows each join to T2 to access index sequentially (and data if high CR)

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
```

T2
DPSI on C1



# DPSIs that could be PIs

- Partitioning indexes (PIs) do not have the same query performance challenges as DPSIs
- Remember we said any “partitioned” index that has the same leftmost columns as the partitioning key is a PI (Partitioning Index)???

```
PARTITION BY ("C1" ASC,
              "C2" ASC,
              "C3" ASC,
              "C4" ASC,
              "C5" ASC)
(PARTITION 1
ENDING AT ('19999999')
,PARTITION 2
ENDING AT ('29999999')
.....
,PARTITION 16
ENDING AT (MAXVALUE)
```

```
CREATE  UNIQUE  INDEX "PROD"."IX1"
  ON  "PROD"."T1"  (
   ("C1" ASC,
    "C2" ASC,
    "C3" ASC,
    "C4" ASC,
    "C5" ASC)
 CLUSTER PARTITIONED
```

DPSI or PI?

```
CREATE  INDEX "PROD"."IX2"
  ON  "PROD"."T1"  (
   "C1" ASC,
   "C6" ASC)
 NOT CLUSTER PARTITIONED
```

DPSI or PI?

# DPSIs that could be PIs

- Index keys must align with “PARTITION BY” columns to be a PI.
- What if I specify less columns in the limit key?
  - Doesn’t matter.
- Why would you have more “PARTITION BY columns?
  - Pre-V8, index-controlled partitioning meant uniqueness, clustering and partitioning were all linked
    - Hence the reason for these multi-column indexes
      - Because you needed to create the index for all columns, and not just the partitioning columns.



# DPSIs that could be PIs

- Switch from index controlled to table controlled ignored the fact that only the “limit key” columns were relevant
- ZPARM **IX_TB_PART_CONV_EXCLUDE**
  - Delivered 11/2011 in V9/10 via APAR PM45829 (default NO)
  - APAR PM90893 10/2013 changed default to YES
- What does this zparm do?
  - When switching from index controlled to table controlled partitioning
    - Only “limit key” columns will be used as partitioning columns



# DPSIs that could be PIs

- Recap of original example
  - With **IX_TB_PART_CONV_EXCLUDE=YES**
    - Both indexes would be PIs

```
PARTITION BY ("C1" ASC)
(PARTITION 1
ENDING AT ('19999999')
,PARTITION 2
ENDING AT ('29999999')
.....
,PARTITION 16
ENDING AT (MAXVALUE)
```

```
CREATE  UNIQUE  INDEX "PROD"."IX1"
  ON  "PROD"."T1"  (
   ("C1" ASC,
    "C2" ASC,
    "C3" ASC,
    "C4" ASC,
    "C5" ASC)
 CLUSTER PARTITIONED

CREATE  INDEX "PROD"."IX2"
  ON  "PROD"."T1"  (
   "C1" ASC,
   "C6" ASC)
 NOT CLUSTER PARTITIONED
```



# Conclusion



# I/O Conclusion

- When an index provides 10% filtering
  - Is that 10% of the data pages?
  - Or, 10% of the rows on 100% of the data pages?
- Filtering pages is just as important as filtering rows
  - Indexing filters the rows
  - Clustering/partitioning filters the pages
- Cluster/partition by data access, not by the primary key
  - And remember there is clustering of the index, and clustering of the data



# Acknowledgements and Disclaimers:

**Availability**. References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

# DB2 11 for z/OS: Partitioning and Indexing – Query Performance Considerations

par Terry Purcell, IBM
tpurcel@us.ibm.com