What's New from the Optimizer in DB2 10 & 11 for z/OS?

Speaker Name and Title

Acknowledgements and Disclaimers:

Agenda

- DB2 10
  - Predicate application
  - Safe Optimization
  - Parallelism Enhancement
  - Other Enhancements
- DB2 10 & 11
  - Plan Management
- DB2 11
  - Predicate indexability
  - Sparse index and in-memory workfile
  - Duplicate removal
  - DPSI performance
  - Misc. Enhancements

Predicate Application

Improvements to predicate application

- **Major enhancements to OR and IN predicates**
  - Improved performance for AND/OR combinations and long IN-lists
    - General performance improvement to stage 1 predicate processing
  - IN-list matching
    - Matching on multiple IN-lists
    - Transitive closure support for IN-list predicates
    - List prefetch support
    - Trim IN-lists from matching when preceding equals are highly filtering
  - Range-list Access for SQL pagination
    - Single index matching for complex OR conditions

- **Many stage 2 expressions to be executed at stage 1**
  - Stage 2 expressions eligible for index screening

IN-list Table - Table Type 'I' and Access Type 'IN'

- **The IN-list predicate will be represented as an in-memory table if:**
  - **List prefetch is chosen, OR**
  - **More than one IN-list is chosen as matching.**

  - **The EXPLAIN output associated with the in-memory table will have:**
    - **New Table Type: TBTYPE – ‘I’**
    - **New Access Type: ACTYPE – ‘IN’**

```
SELECT *
    FROM T1
    WHERE T1.C1 IN (?, ?, ?);
```

```
QBNO   PLANNO   METHOD TNAME          ACTYPE  MC  ACNAME     QBTYPE    TBTYPE PREFETCH

1      1        0      DSNIN001(01)   IN      0              SELECT    I
1      2        1      T1             I       1   T1_IX_C1   SELECT    T      L
```

IN-list Predicate Transitive Closure (PTC)

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
  AND T1.C1 IN (?, ?, ?)

  AND T2.C1 IN (?, ?, ?) ←    Optimizer can generate
                              this predicate via PTC
```

- Without IN-list PTC (DB2 9)
  - Optimizer will be unlikely to consider T2 is the first table accessed
- With IN-list PTC (DB2 10)
  - Optimizer can choose to access T2 or T1 first.

Range-list Access for SQL Pagination

- Scroll forward to obtain the next 20 rows
  - Assumes index is available on (LASTNAME, FIRSTNAME)
  - WHERE clause may appear as:

```
WHERE (LASTNAME='JONES' AND FIRSTNAME>'WENDY')
   OR (LASTNAME>'JONES')
ORDER BY LASTNAME, FIRSTNAME;
```

  - DB2 10 supports
    - Single matching index access with sort avoided
  - DB2 9 requires
    - Multi-index access, list prefetch and sort
    - OR, extra predicate (AND LASTNAME >= ‘JONES’) for matching single index access and sort avoidance

- NOTE: APAR PM56355 to encourage range-list access with OFnR and extra predicate

Stage 2 predicates “pushed down” to IM/DM

- Most Stage 2 (residual) predicates can execute as index screening (indexable) or as stage 1 (sargable)
  - CPU time improvement
  - Reduced data getpages if stage 2 predicate becomes index screening
  - Applies to
    - Arithmetic/datetime expressions, scalar built-in functions, CASE, CAST, (essentially all expressions without subqueries)
    - OR’d predicates cannot span different predicate stages
- Externalized in DSN_FILTER_TABLE column PUSHDOWN
- Not enabled for List Prefetch access type

Stage 2 predicates “pushed down” to IM/DM

- Examples
  - Suppose there exists index on (C1,C3)
  - … WHERE SUBSTR(C1,1,1) = ? ==> index screening
  - … WHERE SUBSTR(C1,1,1) = ? OR C3 = ? ==> index screening
  - … WHERE SUBSTR(C1,1,1) = ? OR C4 = ? ==> stage 1
  - … WHERE SUBSTR(C1,1,1) = ? AND C4 = ? ==> index screening and stage 1
  - … WHERE SUBSTR(C1,1,1) = ? OR C3 = (SELECT...) ==> stage 2
  - … WHERE SUBSTR(C1,1,1) = ? AND C3 = (SELECT...) ==> index scr. and stage 2

Safe Optimization

Minimizing Optimizer Challenges

- Potential causes of sub-optimal plans
  - Insufficient statistics
  - Unknown literal values used for host variables or parameter markers

- Optimizer will evaluate the risk for each predicate
  - For example: WHERE BIRTHDATE < ?
    - Could qualify 0-100% of data depending on literal value used
- As part of access path selection
  - Compare access paths with close cost and choose lowest risk plan

Minimizing impact of RID failure

- RID overflow can occur for
  - Concurrent queries each consuming shared RID pool
  - Single query requesting > 25% of table or hitting RID pool limit

- DB2 9 will fallback to tablespace scan*

- DB2 10 will continue by writing new RIDs to workfile
  - Work-file usage may increase
    - Mitigate by increasing RID pool size (default increased in DB2 10).
    - MAXTEMPS_RID zparm for maximum WF usage for each RID listd

* Hybrid join can incrementally process. Dynamic Index ANDing will use WF for failover.

Parallelism Enhancement

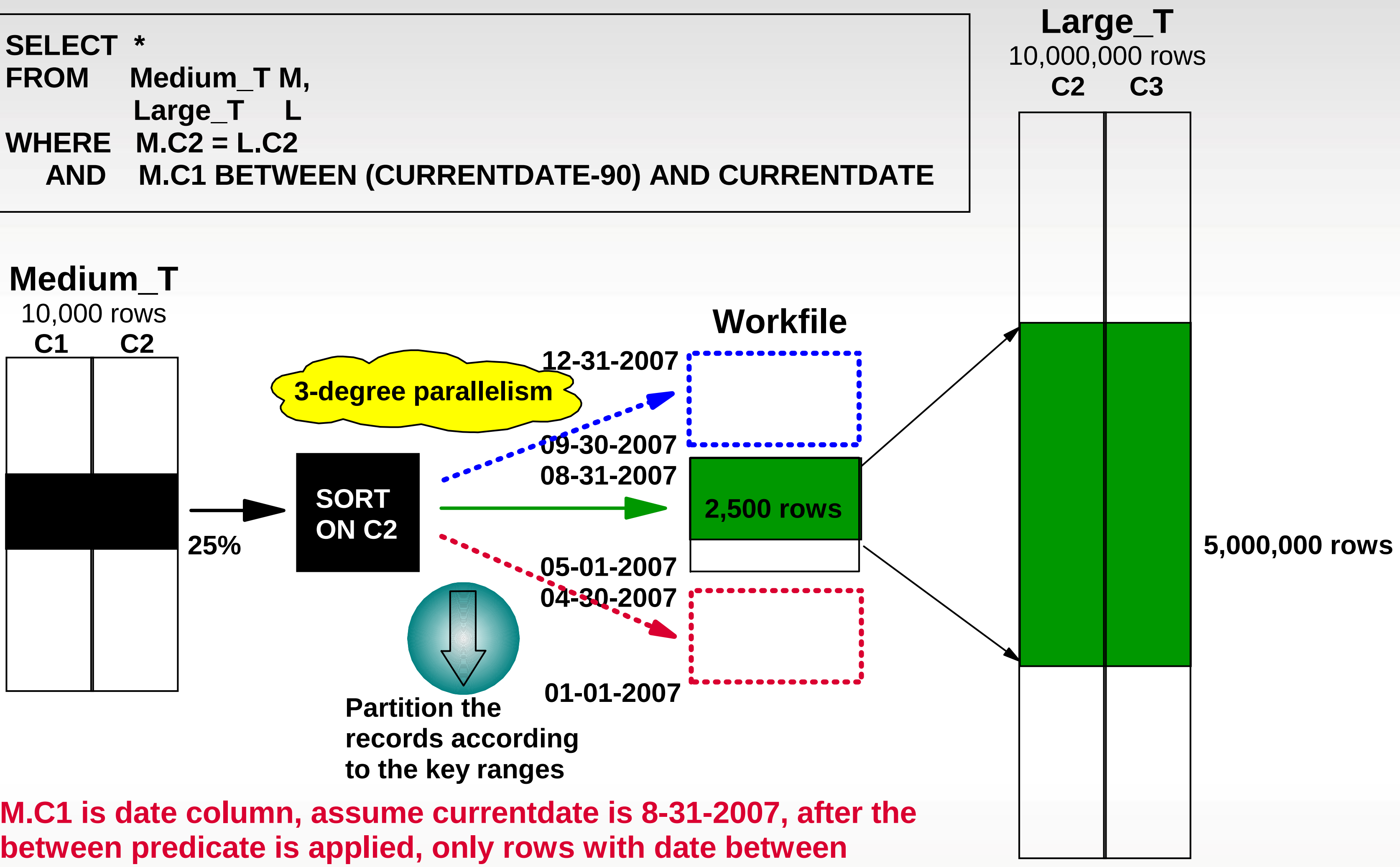
Parallelism Enhancements - Effectiveness

- Previous Releases of DB2 use Key Range Partitioning
  - Key Ranges Decided at Bind Time
  - Based on Statistics (low2key, high2key, column cardinality)
    - Complaint is often that data is not evenly distributed across child tasks.

- DB2 10 solutions available to the optimizer
  - Dynamic Record Range partitioning
    - Introduce a sort to redistribute the data evenly at execution time
  - Straw Model Parallelism
    - Create more work elements than there are concurrent tasks
      - As one child task completes, it takes the next off the queue

Key range partition - Today

```
SELECT  *
FROM     Medium_T M,
         Large_T     L
WHERE   M.C2 = L.C2
    AND    M.C1 BETWEEN (CURRENTDATE-90) AND CURRENTDATE
```

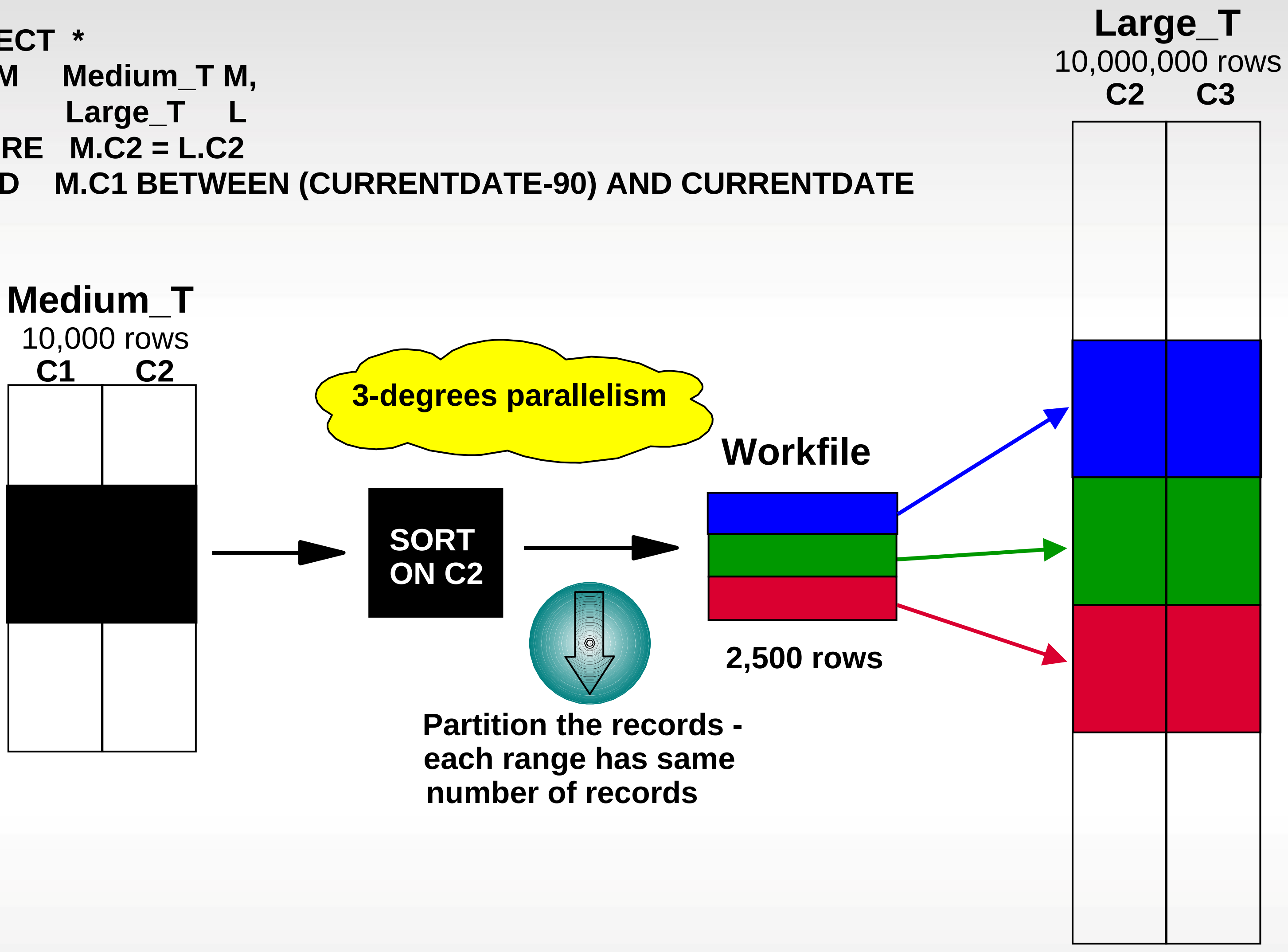


Partition the records according to the key ranges

M.C1 is date column, assume currentdate is 8-31-2007, after the between predicate is applied, only rows with date between 06-03-2007 and 8-31-2007 survived, but optimizer chops up the key ranges within the whole year after the records are sorted :-(

Dynamic record range partition

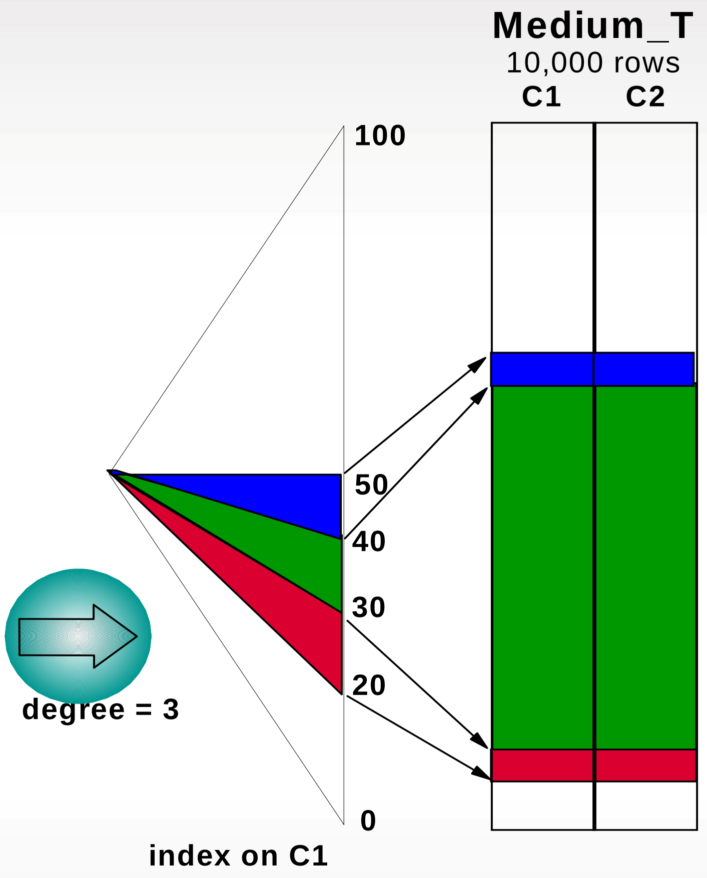
```
SELECT  *
FROM     Medium_T M,
         Large_T    L
WHERE   M.C2 = L.C2
  AND    M.C1 BETWEEN (CURRENTDATE-90) AND CURRENTDATE
```

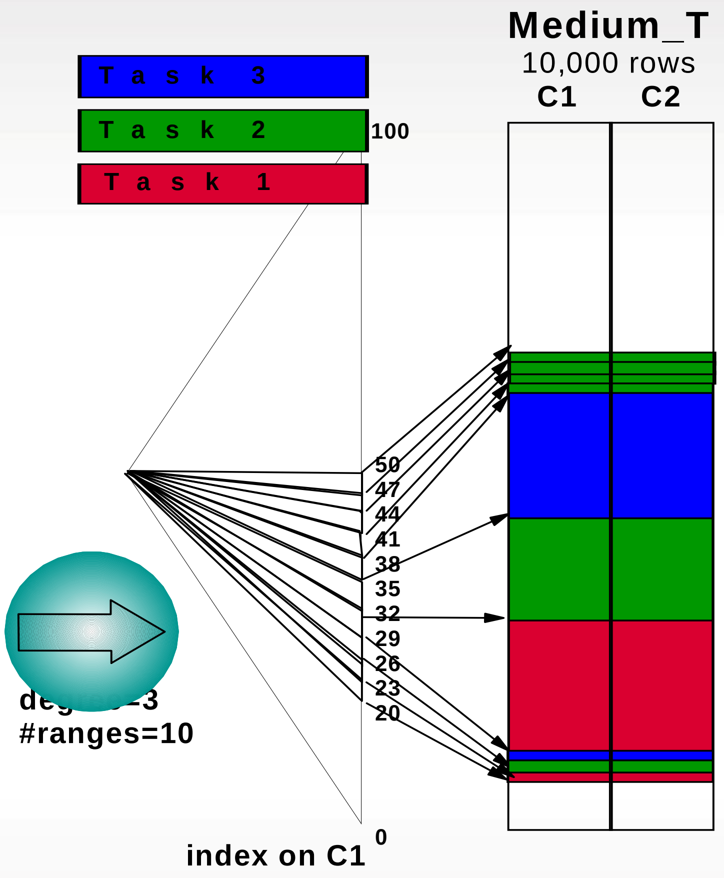


Partition the records - each range has same number of records

STRAW Model

```
SELECT  *
FROM    Medium_T M
WHERE   M.C1 BETWEEN 20 AND 50
```





Divided in key ranges before DB2 10

Divided in key ranges with Straw Model

Other DB2 10 Enhancements

Extending VOLATILE TABLE usage

- VOLATILE TABLE support added in DB2 V8
  - Targeted to SAP Cluster Tables
    - Use Index access whenever possible
    - **Avoids list prefetch**
      - Can be a problem for OR predicates or UPDATEs at risk of loop

- DB2 10 provides VOLATILE to general cases
  - Tables matching SAP cluster tables will maintain original limitations
    - Table with 1 unique index
  - Tables with > 1 index will follow NPGTHRSH rules
    - Use Index access whenever possible
    - **No limitation on list prefetch**
    - Less chance of getting r-scan when list-prefetch plan is only alternative

Misc Performance enhancements

- Index INCLUDE columns
  - Create an Index as UNIQUE, and add additional columns
  - Ability to consolidate redundant indexes

```
INDEX1 UNIQUE (C1)   }    Consolidate to
INDEX2 (C1,C2)       }    INDEX1 UNIQUE (C1) INCLUDE (C2)
```

Plan Management Enhancements V10 & V11

Static Plan Management

- DB2 10 delivered APREUSE(ERROR)
  - Allowed potential for reuse of prior plan to generate new runtime structure
  - Failure of reuse failed the entire package
  - APREUSE(ERROR) EXPLAIN(ONLY) failure may not represent a valid plan in DB2 10
    - Failed access path is written to PLAN_TABLE
- DB2 11 delivers APREUSE(WARN)
  - Upon failure of reuse, Optimizer will generate a new access path choice
    - Thus failure of 1 SQL will not fail the entire package
  - PLAN_TABLE output will represent a valid plan
    - For both ERROR or WARN

Access Path Stability with statement level hints in V10

- Current limitations in hint matching
  - QUERYNO is used to link queries to their hints – a bit fragile
- For dynamic SQL, require a change to apps – can be impractical
- New mechanisms:
  - Associate query text with its corresponding hint … more robust
  - Hints can be enforced for the entire DB2 subsystem
    - irrespective of static vs. dynamic, etc.
  - Hints integrated into the access path repository
- PLAN_TABLE isn't going away
  - Only the "hint lookup" mechanism is being improved.

Statement level hints (cont.)

- Steps to use new hints mechanism
  - Enable OPTHINTS zparm
  - Populate a user table DSN_USERQUERY_TABLE with query text
    - Insert from SYSPACKSTMT (static) or DSN_STATEMENT_CACHE_TABLE (dynamic)
  - Populate PLAN_TABLE with the corresponding hints
    - QUERYNO must match between PLAN_TABLE & DSN_USERQUERY_TABLE
  - Run new command BIND QUERY
    - To integrate the hint into the repository.
    - Next package bind or dynamic prepare can pickup hint.

  - FREE QUERY can be used to remove the query.

Statement-level BIND options

- Statement-level granularity may be required rather than:
  - Subsystem level ZPARMs (STARJOIN, SJTABLES, MAX_PAR_DEGREE)
  - Package level BIND options (REOPT, DEF_CURR_DEGREE)
- For example
  - Only one statement in the package needs REOPT(ALWAYS)
- New mechanism for statement-level bind options:
  - Similar to mechanism used for hints
    - Enable OPTHINTS zparm
    - Populate DSN_USERQUERY_TABLE with query text and desired option
      - Use a QUERYNO that is NOT in PLAN_TABLE
    - Issue BIND QUERY
    - Next package bind/rebind or dynamic prepare can pickup statement option
    - FREE QUERY can be used to remove the query

DB2 11 optimizer enhancements

- Predicate indexability improvements
- Sparse index with in-memory workfile
- Duplicate removal
- DPSI performance
- Misc. enhancements

Predicate Indexability Improvements

Rewrite Common Stage 2 predicates to indexable

- Targeted Stage 2 predicates
  - YEAR(DATE_COL)
  - DATE(TIMESTAMP_COL)
  - value BETWEEN C1 AND C2
  - SUBSTR(C1,1,10) ← SUBSTR from position 1 only

- Stage 2 predicates are ONLY rewritten if there is no candidate Index On Expression to support the predicate
  - Regardless of whether the optimizer chooses that IOE
- Applies to literals or host variables/parameter markers
  - Requires REBIND for static

Indexability for OR/IN and OR COL IS NULL combinations

- Improved single matching index access for OR C1 IS NULL
  - Examples

```
WHERE C1 = ? OR C1 IS NULL

WHERE C1 IN (1, 2) OR C1 IS NULL

WHERE C1 > ? OR C1 IS NULL
```

- IN/OR combination to allow multi-index access……

```
WHERE C1 = ? OR C2 IN (1,2)
Becomes
WHERE C1 = ? OR C2 = 1 OR C2 = 2
```

Prune always true predicates

- Example WHERE 1=1
  - So what's the problem with this harmless predicate?
    - DB2 will execute the WHERE 1=1 predicate for every qualified row

    -
      ```
      SELECT *
      FROM TABLE
      WHERE 1=1
      AND CUSTNO = ?
      ```
  - Prune always true predicate to become

    -
      ```
      SELECT *
      FROM TABLE
      WHERE CUSTNO = ?
      ```

Prune always false predicates

- DB2 10 already prunes “always false” equal/IN under OR
  - WHERE C1 = ? OR ‘A’ = ‘B’
- DB2 11 extends to “always false” underneath parent “AND”

  -
    ```
    SELECT *
    FROM TABLE1 T1, TABLE2 T2
    WHERE (1=1 AND T1.C1 = T2.C1)
          OR (1=2 AND T1.C2 = T2.C2)
    ```
  - Prune always true/false predicates to become

    -
      ```
      SELECT *
      FROM TABLE1 T1, TABLE2 T2
      WHERE T1.C1 = T2.C1
      ```
- NOTE: “OR 0=1” is NOT pruned

Indexability for CASE predicates

- Case can now be indexable (formerly stage 2)
  - For local predicate

```
– SELECT *   FROM T1
  WHERE COL =     CASE (CAST(? AS INT))
                  WHEN 1 THEN 'CA'
                  WHEN 2 THEN 'NY'
                  ELSE 'AL' END;
```

  - For JOIN predicate
    - CASE expression must be evaluated before the join.
    - In example below, join predicate is indexable if T1 accessed before T2.

```
– SELECT * FROM T1, T2
  WHERE T2.COL = CASE WHEN T1.COL = ‘Y’
                         THEN T1.COL2
                 ELSE T1.COL3
                 END;
```

Predicate Pushdown

- DB2 11 supports pushdown into materialized views/TEs of
  - Non-boolean term (OR) predicate

```
SELECT EMPNO, SALARY, DEPTCOUNT
FROM   EMP A ,
(SELECT WORKDEPT, COUNT(*)
 FROM   EMP
 GROUP  BY WORKDEPT) AS B(WORKDEPT, DEPTCOUNT)
WHERE A.WORKDEPT = B.WORKDEPT
AND  (B.WORKDEPT LIKE 'C%' OR B.WORKDEPT LIKE 'A%‘);
```

  - Stage 2 predicates (expressions)

```
SELECT EMPNO, SALARY, DEPTCOUNT
FROM   EMP A ,
(SELECT WORKDEPT, COUNT(*)
 FROM   EMP
 GROUP  BY WORKDEPT) AS B(WORKDEPT, DEPTCOUNT)
WHERE A.WORKDEPT = B.WORKDEPT
AND UPPER( B.WORKDEPT) = 'C01'
```

Predicate Pushdowns (cont.)

- DB2 11 supports pushdown into materialized views/TEs of
  - Predicate in the ON clause of an outer join

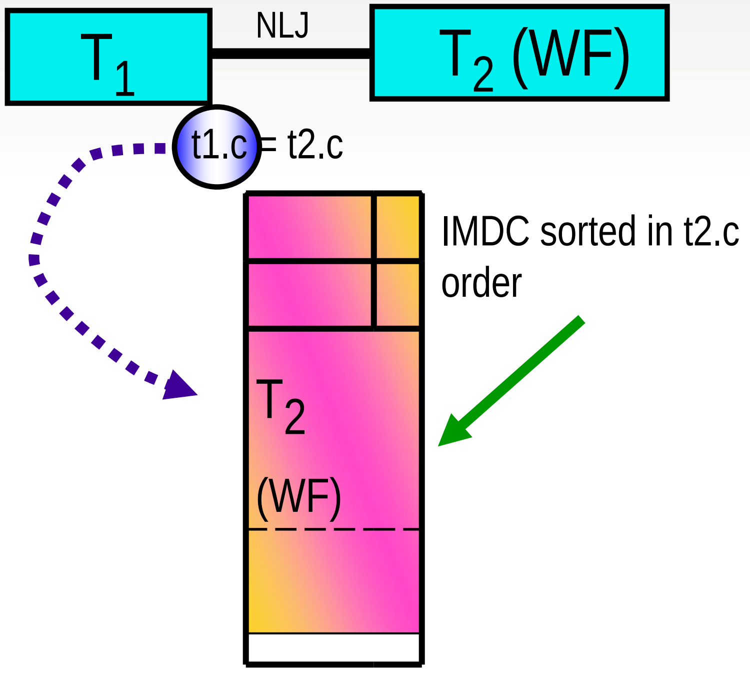
```
SELECT EMPNO, SALARY, DEPTCOUNT
FROM   EMP A
LEFT OUTER JOIN
(SELECT WORKDEPT, COUNT(*)
 FROM   EMP
 GROUP  BY WORKDEPT) AS B(WORKDEPT, DEPTCOUNT)
ON A.WORKDEPT = B.WORKDEPT
AND B.WORKDEPT = 'C01';
```

  - Also – when the view/TE contains a scalar function in the SELECT list
  - Some restrictions still remain, such as:
    - If all 3 examples had predicates against table A – predicate not pushed in
    - Expression pushdown may not qualify for index on expression

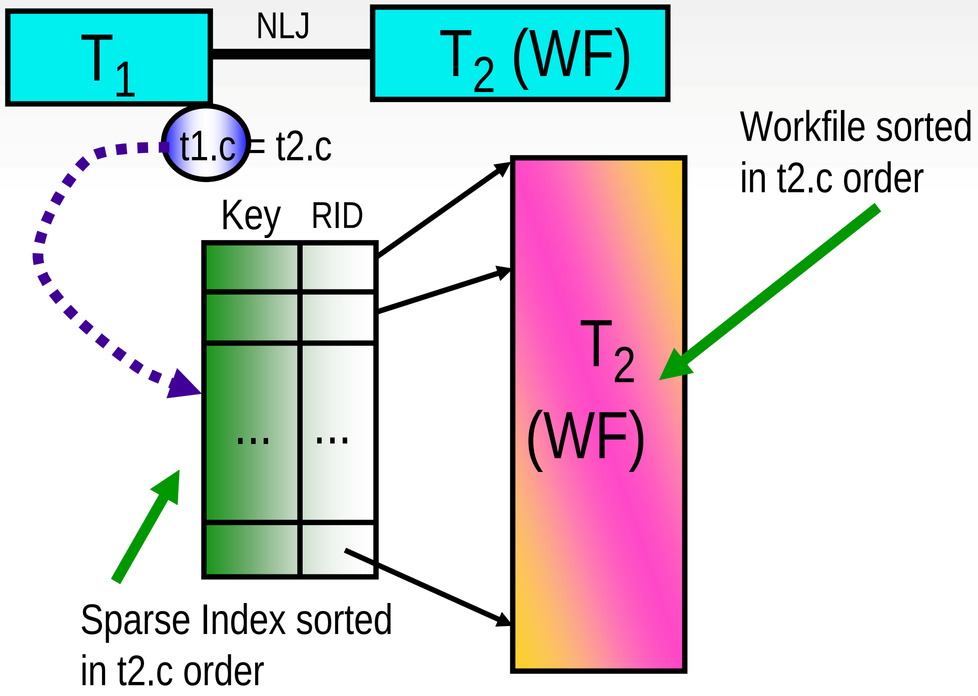
In-memory data cache/Sparse Indexing (or hash join)

In Memory Data Cache (Workfile) vs. Sparse Index

- IMDC
- Sparse Index
  - When insufficient memory for IMDC



Binary Search of WF to look up exact location of qualified key (Hash used if sufficient memory)



T_1
NLJ
T_2 (WF)
t1.c = t2.c
Key
RID
...
...
T_2
(WF)
Workfile sorted in t2.c order
Sparse Index sorted in t2.c order

Binary Search of sparse index to look up “approximate “ location of qualified key

IMDC/Sparse Index DB2 11 Enhancements

- Improved memory management by optimizer and runtime
  - Controlled by zparm MXDTCACH (default 20MB)

  - Each sparse index/IMDC is given a % of MXDTCACH
    - From optimizer cost perspective
    - At runtime (based upon cost estimation)

  - Runtime will choose appropriate implementation based upon available storage
    - Hash, binary search, or spill over to workfile

- Non-correlated subqueries will also share MXDTCACH

- Improved optimizer cost model
  - Allowing this to be opened up in more join scenarios

- Improvements to IFCID 27 for detail, 2 & 3 for summary

IMDC/Sparse index – Tuning basics

- DB2 11 provides simple accounting/statistics data for sparse index
  - Sparse IX disabled
    - indicates main memory was insufficient for the MXDTCACH memory request
    - Suggest reducing MXDTCACH or allocating more memory to the system
  - Sparse IX built WF
    - MXDTCACH was insufficient to contain sparse index
      - Increase MXDTCACH
    - Look at sort BP sync I/O
      - If high, also reduce VPSEQT in sort BP (do not use VPSEQT=100)

```
MISCELLANEOUS                AVERAGE      TOTAL
--------------------       --------   --------
SPARSE IX DISABLED             0.00          0
SPARSE IX BUILT WF             0.36          8
```

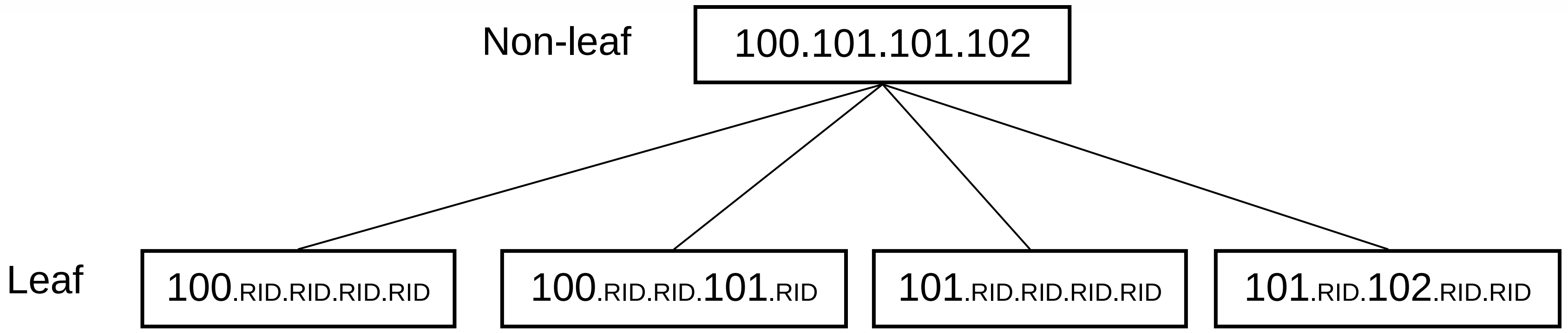
Duplicate Removal

Index skipping and Early-out – DB2 11 Enhancements

- Improvements to queries involving GROUP BY, DISTINCT or non-correlated subq
  - Where an index can be used for sort avoidance
    - By skipping over duplicates (see next few slides)
- Improvement to join queries using GROUP BY, DISTINCT
  - By NOT accessing duplicates from inner table of a join if DISTINCT/GROUP BY will remove those duplicates
- Improvement to correlated subqueries
  - Early-out for ordered access to MAX/MIN correlated subqueries
    - When I1-fetch is not available
  - Optimize usage of the "result cache" for access to subquery with duplicate keys from the outer query
    - 100 element result cache dates back to DB2 V2 as a runtime optimization
    - DB2 11 adds optimizer recognition of benefit

Pre-DB2 11 Duplicate Removal using an index (no sort)

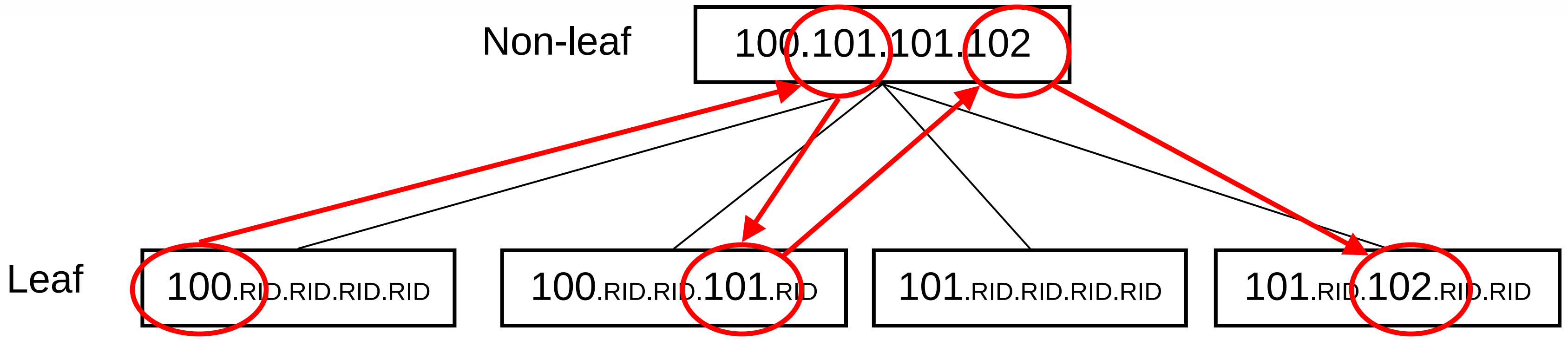
```
SELECT C1
FROM T
GROUP BY C1
```



Scan qualified leaf pages (and all rids) with runtime discarding duplicates

DB2 11 - Duplicate Removal with Index Skipping

```
SELECT C1
 FROM T
GROUP BY C1
```



Index Skipping (over-simplified)

Use index lookaside (current leaf high key and non-leaf) to get the next key greater than current key

Early-out join

- DB2 11 supports early-out for join tables where duplicates are not necessary
  - Previously only available for correlated EXISTS subquery transformed to join.
  - For below example: Duplicates from T2 are removed by DISTINCT
    - In DB2 11, each inner table probe will stop after 1st match is found
      - NOTE: For LEFT OUTER JOIN V10 will prune T2

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

    - Also applies to Non-Boolean Term join conditions with “early-out” table

```
SELECT DISTINCT T1.*
FROM T1, T2
WHERE T1.C1 = 1
   OR T1.C1 = T2.C1
```

Optimize usage of subquery result cache

- DB2 V2 introduced a result cache for saving the 100 most recent correlated subquery execution results
  - Each subquery execution would 1st scan the cache to find the result
    - If found, cache value is used
    - If not found, subquery is executed, and result saved in cache
- DB2 11 adds optimizer recognition of the cache
  - And the benefit to accessing the subquery in order
    - Ordered access will reduce the cache size from 100
      - For example below, the optimizer recognizes that accessing the outer in CUSTNO order (via CUSTNO index or tablespace scan if CUSTNO is clustering) would result in cache hits for repeat CUSTNO values

```
SELECT *
FROM POLICY P1
WHERE P1.POLICY_DATE =
(SELECT MAX(P2.POLICY_DATE)
 FROM POLICY P2
 WHERE P2.CUSTNO = P1.CUSTNO)
```

DPSI Performance

DPSI Performance – DB2 11 Enhancements

- DPSI and page range performance Improvements
  - Page Range Screening on Join Predicates
    - Access only qualified partitions
      - Equal predicates, same datatype/length only
  - Improved DPSI Join Performance
    - Partition-level join with sequential access to join inner table

- NOTE: Above 2 performance enhancements are dependent on partitioning scheme
  - To benefit from page range screening for join predicates
    - Requires partition by columns used as join predicates
  - To benefit from part-level join
    - Requires partition by non-join columns, but DPSI on join columns

- Enhancements to sort avoidance for DPSIs (Also known as DPSI return in order)
  - Use of Index On Expression (IOE)
    - Ability to avoid sorting with DPSI IOE (already available for DPSI non-IOE)
  - Index lookaside when DPSI used for sort avoidance

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

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

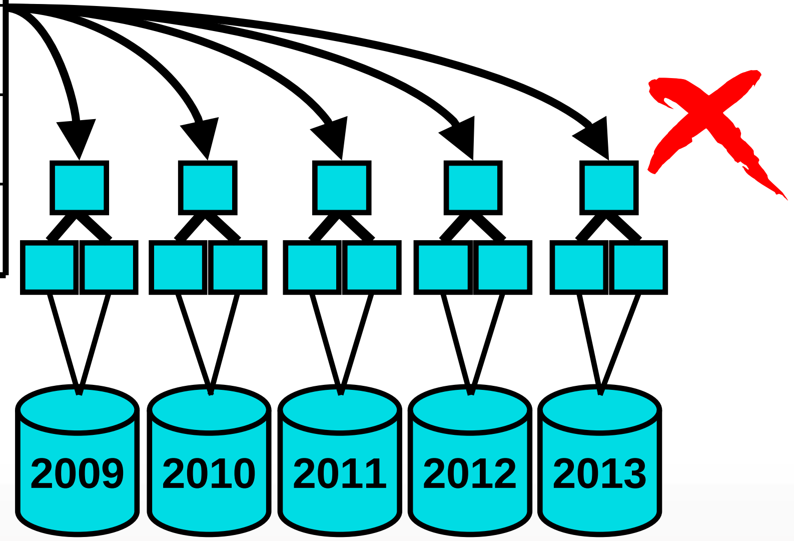
```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.YEAR = T2.YEAR
```

YEAR	C1
2009	1
2010	2
2011	3

T2

DPSI on C1

Partition by YEAR



All parts are accessed for each composite row

V11 DPSI Join Probing (Join on Partitioning Col)

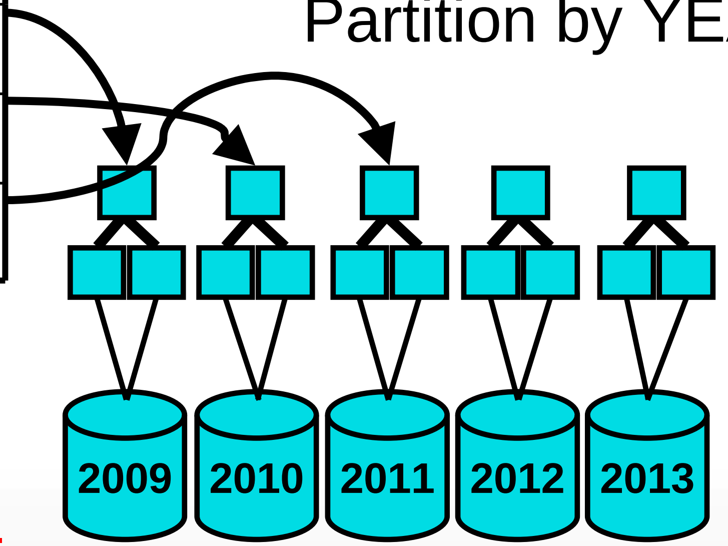
- Join recognizes page range screening
  - 1st composite row probes 1 part.
  - 2nd composite row probes 1 part.
  - And so on.

YEAR	C1
2009	1
2010	2
2011	3

T2

DPSI on C1

Partition by YEAR



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

Only qualified parts are probed on the inner table.

DPSI Parallelism Enhancements

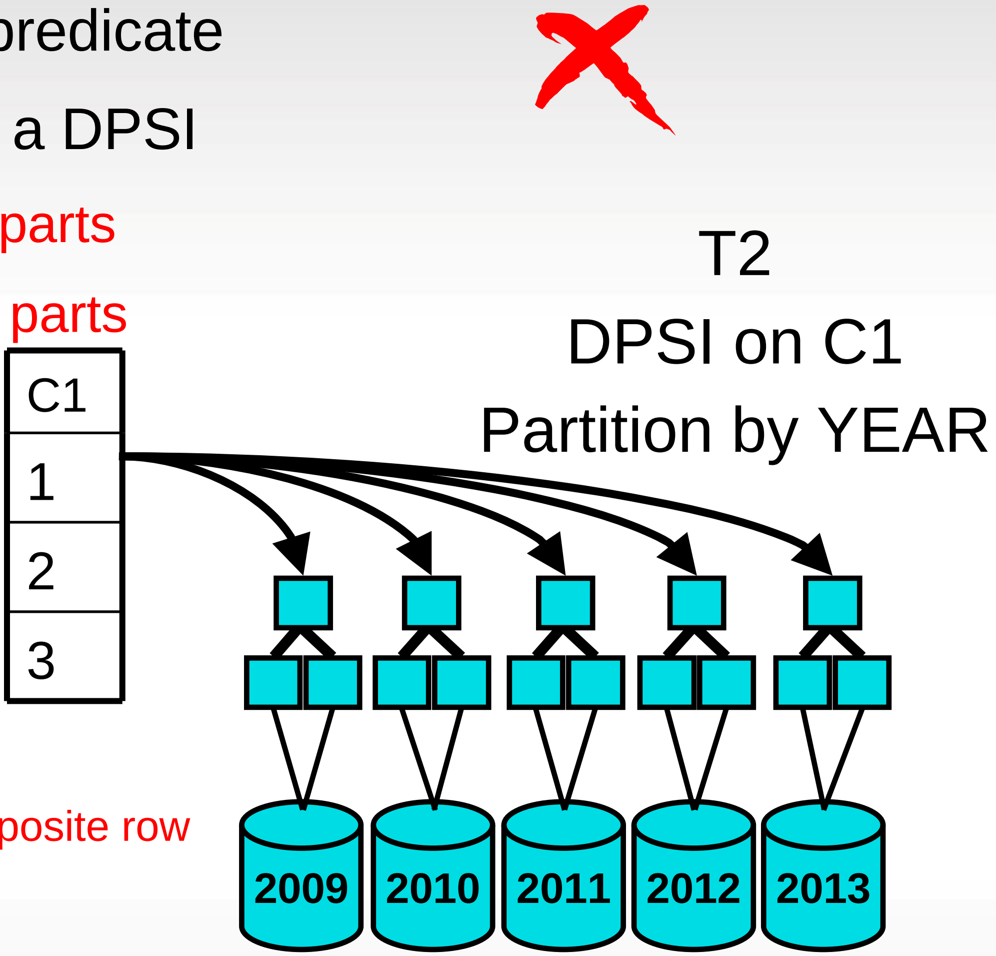
- DPSI Part-level (Nested Loop) Join Operation
  - Parallelism allows each child task to process 1 DPSI part.
  - Controlled by zparm PARAMDEG_DPSI

- Straw-model parallelism support for DPSI
  - Straw-model (delivered in V10) implies that DB2 creates more work elements than there are degrees on parallelism.

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
```



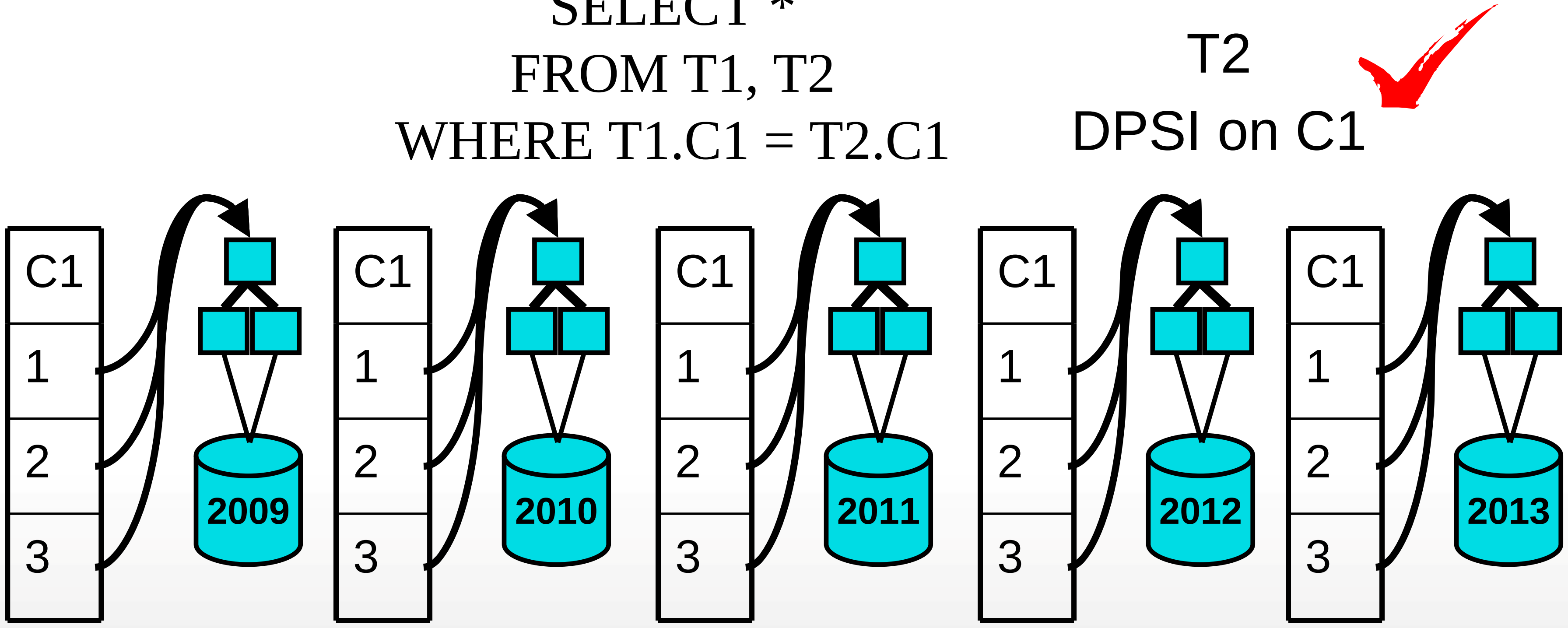
All parts are accessed for each composite row

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



Misc Performance Items

Sort / Workfile Performance Improvement

- In memory workfile support in DB2 9 and 10
  - DB2 9 RDS simple sort up to 32K
  - DB2 10 RDS simple sort up to 1MB (no user control)
  - DB2 10 intermediate workfile usage up to 32K for selective path
- More in memory operation in DB2 11
  - RDS simple sort up to 128MB
    - Controlled by a new zparm MAXSORT_IN_MEMORY (default 1MB)
  - Wider range of usage for in memory
    - Materialized view, table expression, outer Join, EXISTS,   etc.
    - In memory up to 32K then continue with physical workfiles
- Avoid workfile usages for final merge on top level sort
  - Reduces physical workfile usage for large top level sort

RID processing enhancments

- Pre-DB2 11
  - DB2 10 added RID failover to WF
    - Did not apply to queries involving column function
  - A single Hybrid Join query could consume 100% of the RID pool
    - Causing other concurrent queries to hit RID limit if > 1 RID block needed

- DB2 11
  - RID failover to WF extended to all scenarios when RID limit is hit
  - Hybrid join limited to 80% of the RID pool

Other interesting performance items

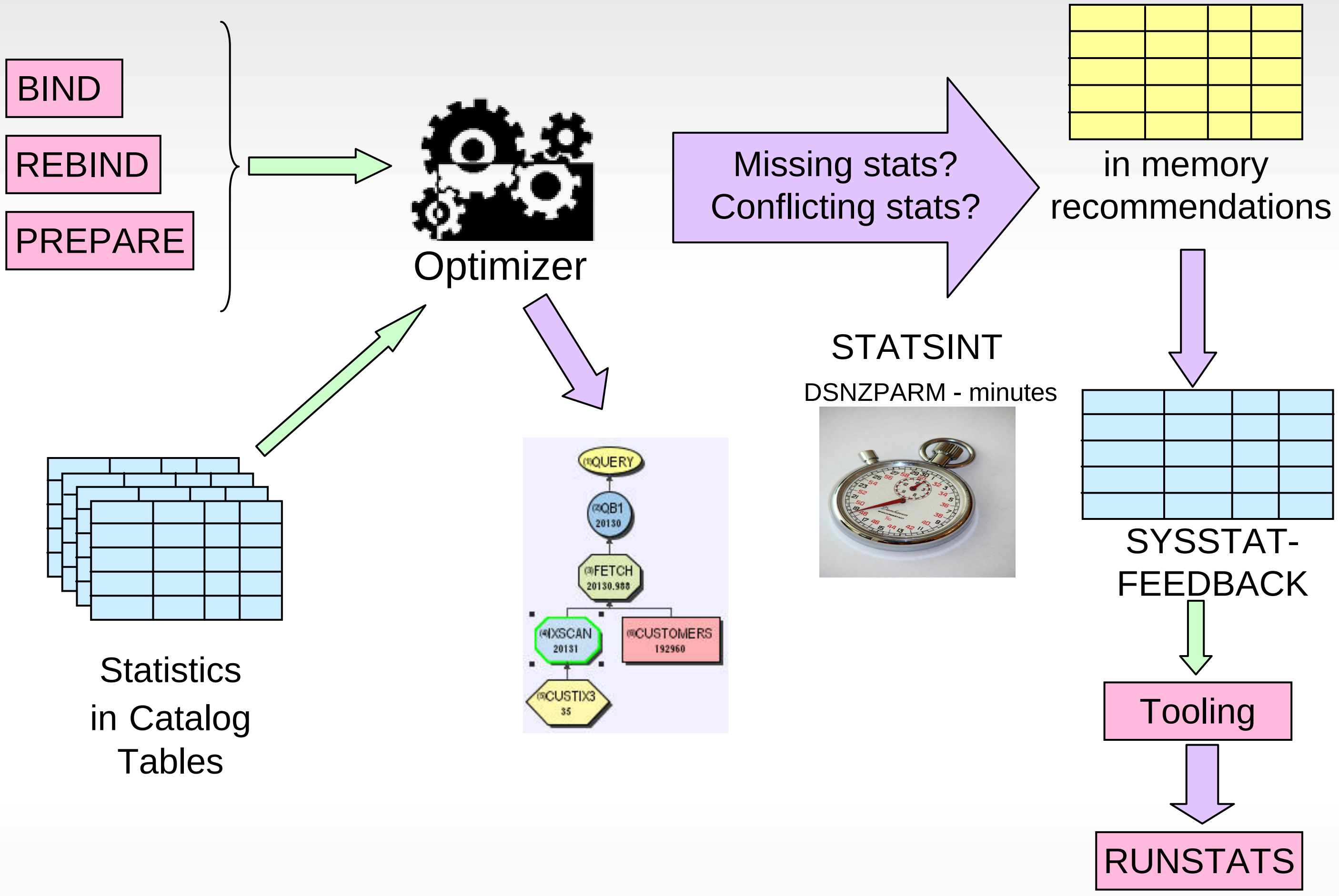
- DGTT NOT LOGGED support
- EXCLUDE NULL indexes
- Pseudo-deleted index entry cleanup
- Reduction of indirect references
- Decompression performance improvements
- DECFLOAT performance improvements (used extensively in XML)

Optimizer externalization of missing stats and Overriding FF estimates

DB2 Optimizer and Statistics - Challenge

- DB2 cost-based optimizer relies on statistics about tables and indexes
- Customers often gather only standard or default statistics
  - E.g. RUNSTATS TABLE(ALL) INDEX(ALL) KEYCARD
- Queries would often perform better if DB2 optimizer could exploit more complete statistics
- Customers have difficulty knowing which statistics are needed

DB2 11 – Optimizer externalization of missing statistics



DB2 11 Solution: Optimizer Externalization

- During access path calculation, optimizer will identify missing or conflicting statistics
  - On every BIND, REBIND or PREPARE
    - Asynchronously writes recommendations to SYSIBM.SYSSTATFEEDBACK
  - DB2 also provides statistics recommendations on EXPLAIN
    - Populates DSN_STAT_FEEDBACK synchronously

- Contents of SYSSTATFEEDBACK or DSN_STAT_FEEDBACK can be used to generate input to RUNSTATS
  - Contents not directly consumable by RUNSTATS
  - Requires DBA or tooling to convert to RUNSTATS input

Optimizer selectivity - The Filter Factor Problem

- Query optimization challenges
  - Cost based query optimization
    - Estimate cost of available choices to identify choice with cheapest cost
  - The optimizer needs to know how many rows are filtered at every step
    - How much does it cost to scan index ? Matching, screen filtering
    - Which table should be outer?

- Sometimes, the optimizer is unable to accurately estimate selectivity
  - Lack of statistics
  - Join skew, join correlation
  - Complex predicates
  - Predicate combinations
  - Unknowns (host variables, parameter markers, special registers)

DB2 11 Selectivity Overrides (FF hints)

- Process of supplying more robust selectivity (Filter Factor) input
  - Rather than a whole OPTHINT – just FF hints

- Selectivity profile allows User to
  - Provide optimizer with a more accurate view of selectivities used in query execution
    - For one, some or all predicates in a query
    - For one or more representative “executions” of a query
      - Weighted by frequency of occurrence

- Similar to the SELECTIVITY clause on SQL statement, but ...
    - Doesn’t require changing applications
    - Handle variations in execution

- Also has OQWT tooling support

Virtual Index Improvements

Virtual Index Enhancements – Table Changes

- DSN_VIRTUAL_INDEXES enhanced
  - Columns added to complete index modelling capabilities
    - UNIQUE_COUNT
      To support INCLUDE index columns
    - SPARSE
      To support NULL Supressed indexes
    - DATAREPEATFACTORF
      To support enhanced statistics gathering
    - KEY_TARGET_COUNT & IX_EXTENSION
      To support Index on Expression and XML Index
- DSN_VIRTUAL_KEYTARGETS
  - New EXPLAIN table used for Index Advisor support for IOE and XML indexes

Thank You

Your feedback is important!