



DB2 for Linux, UNIX, and Windows Optimizer

Guy M. Lohman

lohman@almaden.ibm.com

**IBM Research Division
IBM Almaden Research Center
K55/B1, 650 Harry Road
San Jose, CA 95120**

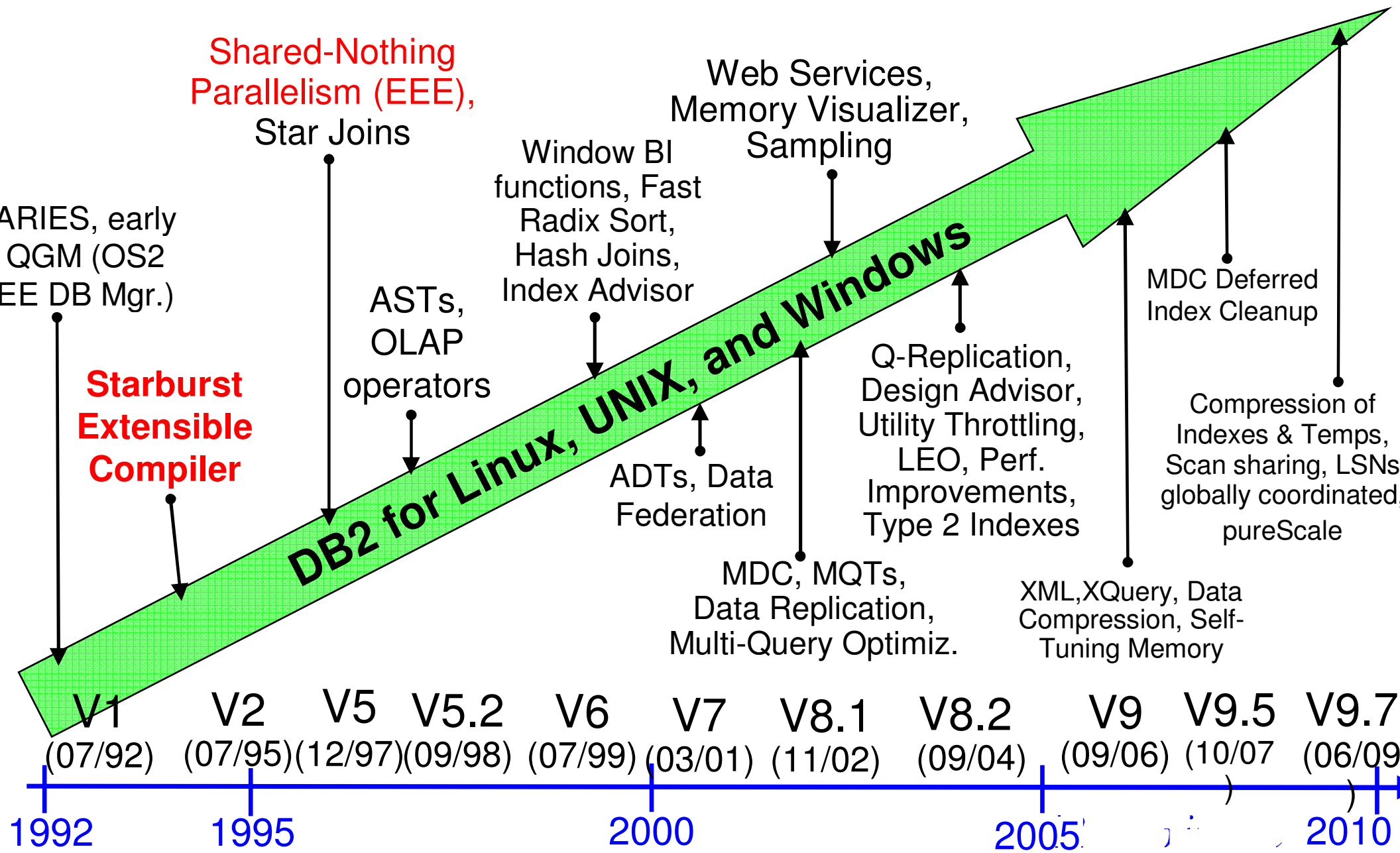
IBM Software

Agenda

- 
- Overview of Query Processing
 - Query ReWrite
 - Plan Selection Optimization basics
 - Elements of Optimization
 - Execution Strategies
 - Cost model & plan properties
 - Search strategy
 - Parallelism
 - Special strategies for OLAP & BI
 - Engineering considerations
 - Conclusions and Future
 - NOTE: Use DB2 for free for academic purposes! See:

<http://www.ibm.com/developerworks/university/data/index.html>

Research Contributions to DB2 for Linux, UNIX, & Windows



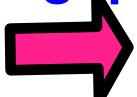
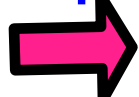
Stretching the Boundaries: Query Processing Challenge

■ Many platforms, but one codebase!

- Software: Unix/Linux (AIX, HP, Sun, Linux), Windows, Sequent, OS/2
- Hardware: Uni, SMP, MPP, Clusters, NUMA
- Query languages: SQL, **SQL/XML**, **XQuery** (New in DB2 V9.1!)

■ Database volume ranges continue to grow: 1GB to > 10 PB

■ Increasing query complexity:

- OLTP  DSS  OLAP / ROLAP
- SQL generated by query generators, naive users

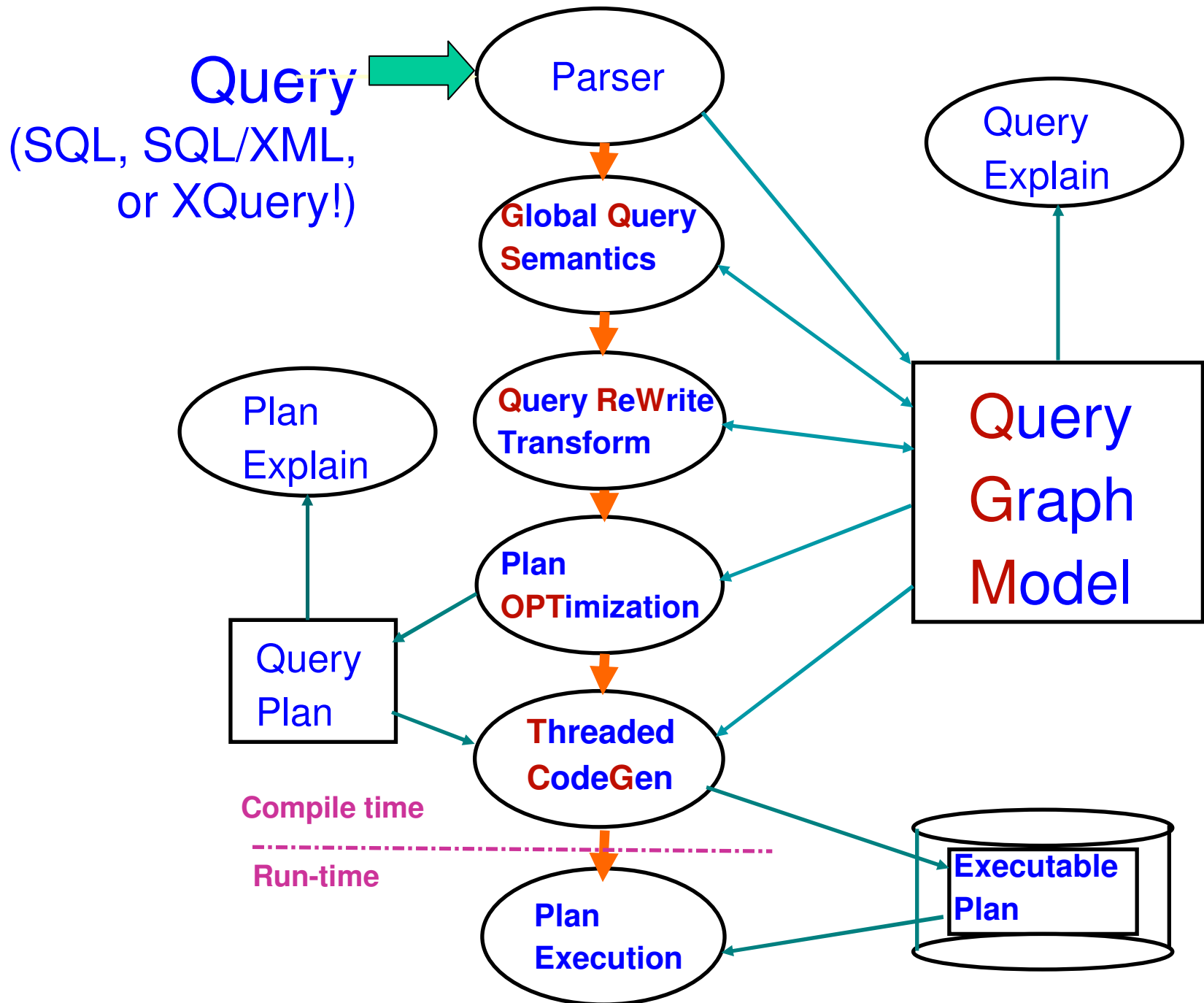
■ Managing complexity

- Fewer skilled administrators available
 - distributed systems
 - database design can be complex
- Too many knobs!

REFN: Laura M. Haas, Walter Chang, Guy M. Lohman, John McPherson, Paul F. Wilms, George Lapis, Bruce G. Lindsay, Hamid Pirahesh, Michael J. Carey, Eugene J. Shekita: Starburst Mid-Flight: As the Dust Clears.

IEEE Trans. Knowl. Data Engr. 2, 1: 143-160 (1990).

Query Compiler Overview



Elements of Query Compilation

■ Parsing

- Analyze "text" of query (can be SQL, SQL/XML, or XQuery!)
- Detect syntax errors
- Create internal query representation

■ Semantic Checking

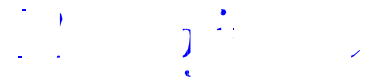
- Validate statement
- View analysis
- Incorporate constraints, triggers, etc.

■ Query Optimization

- Modify query to improve performance (Query Rewrite)
- Choose the most efficient "access plan" (Query Optimization)

■ Code Generation

- Generate code that is
 - executable
 - efficient
 - re-locatable



Query Graph Model (QGM)

- Captures the entire semantics of the query to be compiled
- "Headquarters" for all knowledge about compiling a query
- Represents internally that query's:
 - ✓ Entities (e.g. tables, columns, predicates,...)
 - ✓ Relationships (e.g. "ranges-over", "contains", ...)
- Has its own ("meta"-) schema
 - ☞ Entity-Relationship (ER) model
- Semi-Procedural: Visualized as a high-level Data Flow Model
 - ☞ Boxes (nodes) represent table operations, e.g., Select-Project-Join
 - ☞ Rows flow through the graph
- Implemented as a C++ library
 - ☞ Facilitates construction, use, and destruction of QGM entities
- Designed for flexibility
 - ☞ Easy extension of SQL Language (i.e. SELECT over IUDs)
- ☞ REFN: Hamid Pirahesh, Joseph M. Hellerstein, Waqar Hasan:
"Extensible/Rule Based Query Rewrite Optimization in Starburst",
SIGMOD 1992, pp. 39-48

Query Rewrite - An Overview

■ What is Query Rewrite?

- Rewriting a given query into a semantically equivalent form that
 - may be processed more efficiently
 - gives the Optimizer more latitude

■ Why?

- Same query may have multiple representations (true in SQL, XQuery)
- Complex queries often result in redundancy, especially with views
- Query generators
 - often produce suboptimal queries that don't perform well
 - don't permit "hand optimization"

■ Based on Starburst Query Rewrite

- Rule-based query rewrite engine
- Transforms legal QGM into more efficient QGM
- Some transformations aren't always universally applicable
- Has classes of rules
- Terminates when no rules eligible or budget exceeded

■ REFN: Hamid Pirahesh, T. Y. Cliff Leung, Waqar Hasan, "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS", ICDE 1997, pp. 391-400.

Query Rewrite: Predicate Pushdown Example

■ Original query:

```
CREATE VIEW lineitem_group(suppkey, partkey, total)
AS SELECT I_suppkey, I_partkey, sum(quantity)
   FROM   tpcd.lineitem
   GROUP BY I_suppkey, I_partkey;
```

```
SELECT *
FROM lineitem_group
-WHERE suppkey = 1234567;
```

■ Rewritten query:

```
CREATE VIEW lineitem_group(suppkey, partkey, total)
AS SELECT I_suppkey, I_partkey, sum(quantity)
   FROM   tpcd.lineitem
   WHERE I_suppkey = 1234567
   GROUP BY I_suppkey, I_partkey;
```

```
SELECT *
FROM lineitem_group;
```



Agenda

- Overview of Query Processing
- Query ReWrite
- ➔ ■ Plan Selection Optimization basics
 - Elements of Optimization
 - Execution Strategies
 - Cost model & plan properties
 - Search strategy
- Parallelism
- Special strategies for OLAP & BI
- Engineering considerations
- Conclusions and Future
- NOTE: Use DB2 for free for academic purposes! See:
<http://www.ibm.com/developerworks/university/data/index.html>

What does the Query Optimizer Do?

■ Generates & Evaluates alternative

– Operation order

- joins
- predicate application
- aggregation

– Implementation to use:

- table scan vs. index scan
- nested-loop join vs. sorted-merge join vs. hash join

– Location (in partitioned environments)

- co-located
- re-direct each row of 1 input stream to appropriate node of the other stream
- re-partition both input streams to a third partitioning
- broadcast one input stream to all nodes of the other stream

■ Estimates the execution of that plan

– Number of rows resulting

– CPU, I/O, and memory costs

– Communications costs (in partitioned environments)

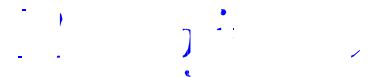
■ Selects the best plan, i.e. with minimal

– Total resource consumption (normally)

– Elapsed time (in parallel environments, OPTIMIZE FOR N ROWS)

Inputs to Optimizer

- **Database characteristics (in system catalogs)**
 - Schema, including constraints
 - Statistics on tables, columns, indexes, etc.
- **Hardware environment, e.g.**
 - **Speed of CPU**
 - determined automatically at database creation time
 - runs a timing program
 - **Storage device characteristics**
 - used to model random and sequential I/O costs
 - set at table-space level
 - overhead (seek & average rotational latency)
 - transfer_rate
 - **Communications bandwidth**
 - to factor communication cost into overall cost, in partitioned environments
 - **Memory resources**
 - Buffer pool(s)
 - Sort heap
- **Concurrency Environment**
 - Average number of users
 - Isolation level / blocking
 - Number of available locks



Major Aspects of Query Optimization

1. Alternative Execution Strategies (methods)

- ★ Rule-based generation of **plan operators**
- ★ Creates alternative
 - ▶ Access paths (e.g. indexes)
 - ▶ Join orders
 - ▶ Join methods

2. Properties and Cost Model

- ★ **Cardinality Model** (Number of rows), based upon
 - ▶ **Statistics** for table
 - ▶ **Selectivity estimate** for predicates
- ★ **Properties & Costs**
 - ▶ Determined per operator type
 - ▶ Tracked per operator instance (cumulative effect)
- ★ Prunes plans that have
 - ▶ Same or subsumed properties
 - ▶ Higher cost

3. Search Strategy

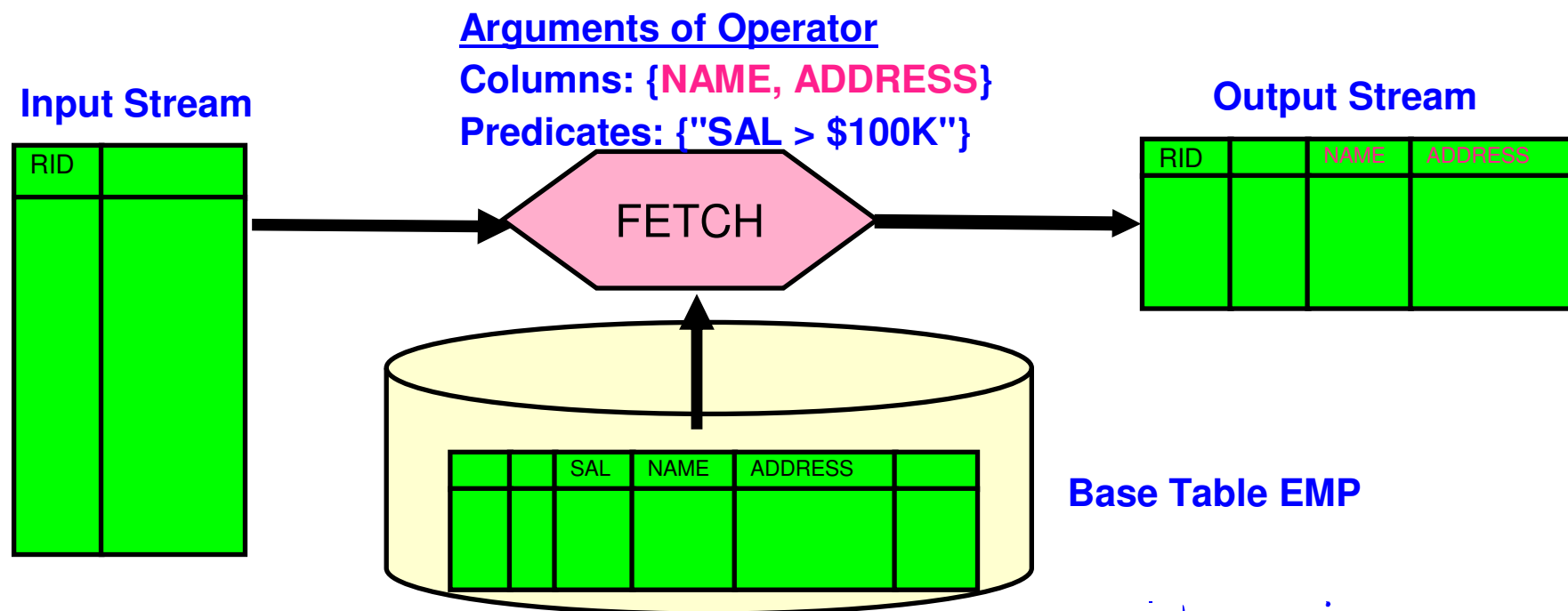
- ★ **Dynamic Programming** vs. **Greedy**
- ★ **Bushy** vs. **Deep**

■REFN: Peter Gassner, Guy M. Lohman, K. Bernhard Schiefer, Yun Wang, "Query Optimization in the IBM DB2 Family", Data Engineering Bulletin 16(4): 4-18 (1993).

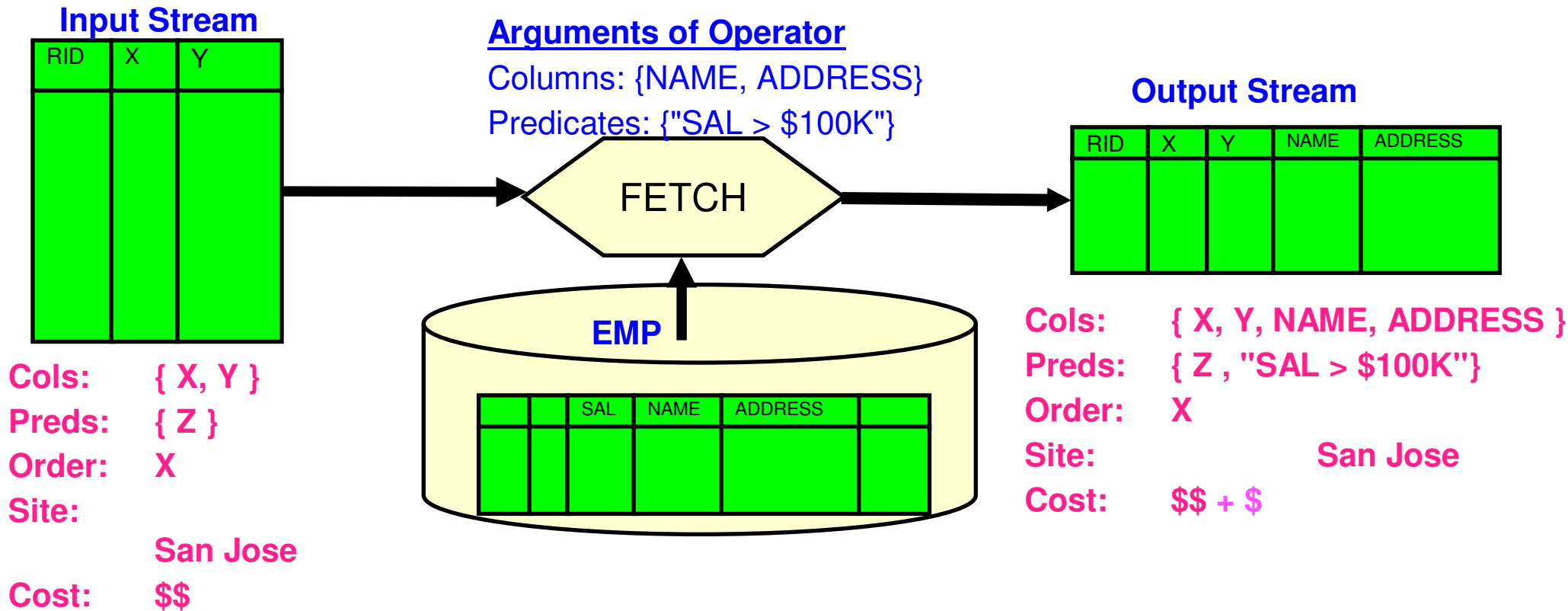
Atomic Object: LOw-LEvel Plan OPerator (LOLEPOP)

- Database operator, interpreted at execution time
- Operates on, and produces, tables
(visualized as in-memory streams of rows)
- Examples:
 - Relational algebra (e.g. JOIN, UNION)
 - Physical operators (e.g. SCAN, SORT, TEMP)
- May be expressed as a function with parameters, e.g.

FETCH(<input stream>, Emp, {Name, Address}, {"SAL > \$100K"})



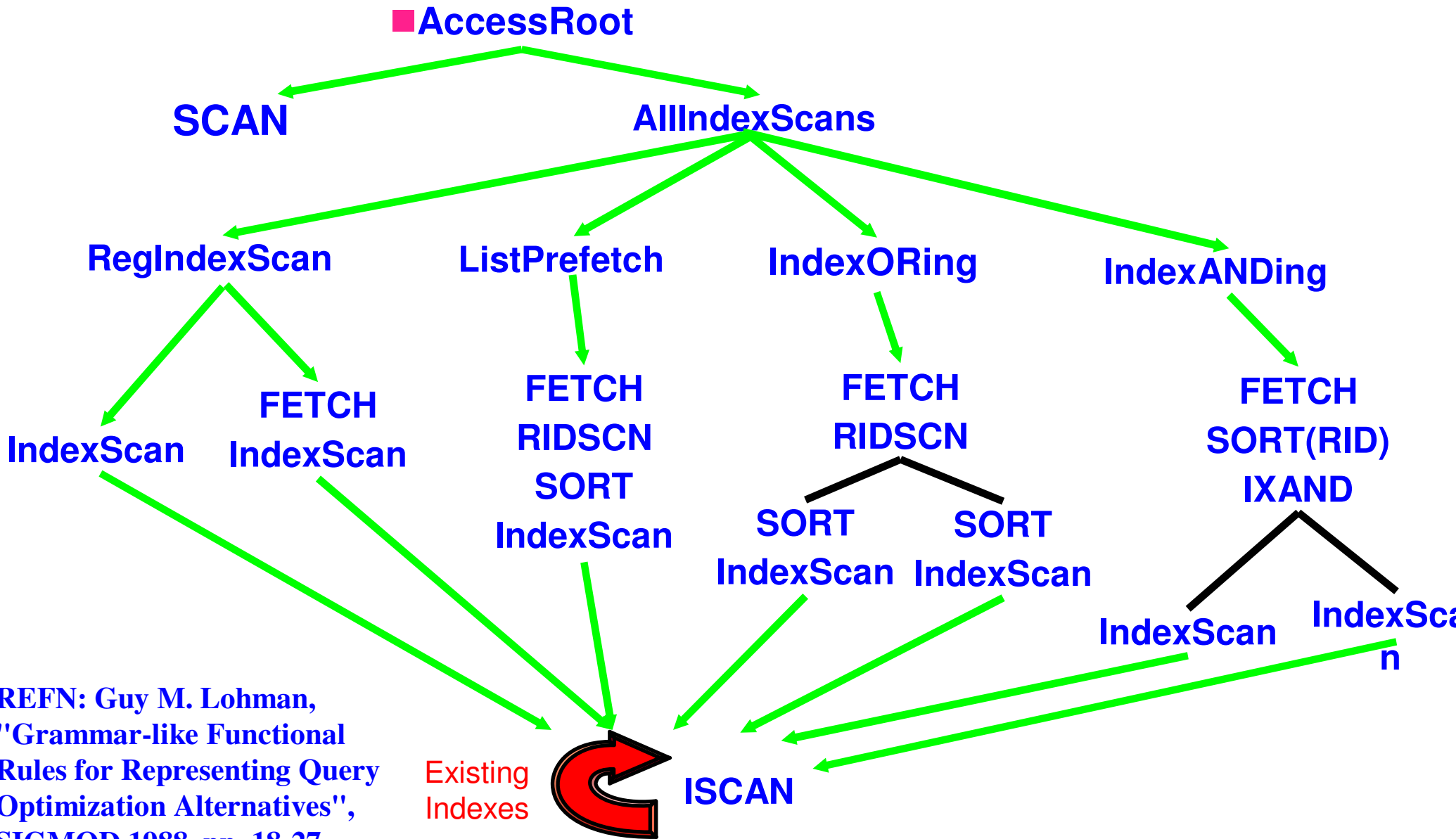
Properties of Plans



- Give cumulative, net result (including cost) of work done
- Initially obtained from statistics in catalogs for stored objects
- Altered by effect of LOLEPOP type (e.g., SORT alters ORDER property)
- Specified in Optimizer by **property** and **cost functions** for each LOLEPOP

REFN: M. K. Lee, J. C. Freytag, G. M. Lohman, "Implementing an Interpreter for Functional Rules in a Query Optimizer", VLDB 1988, 218-229

Generation of Table Access Alternatives



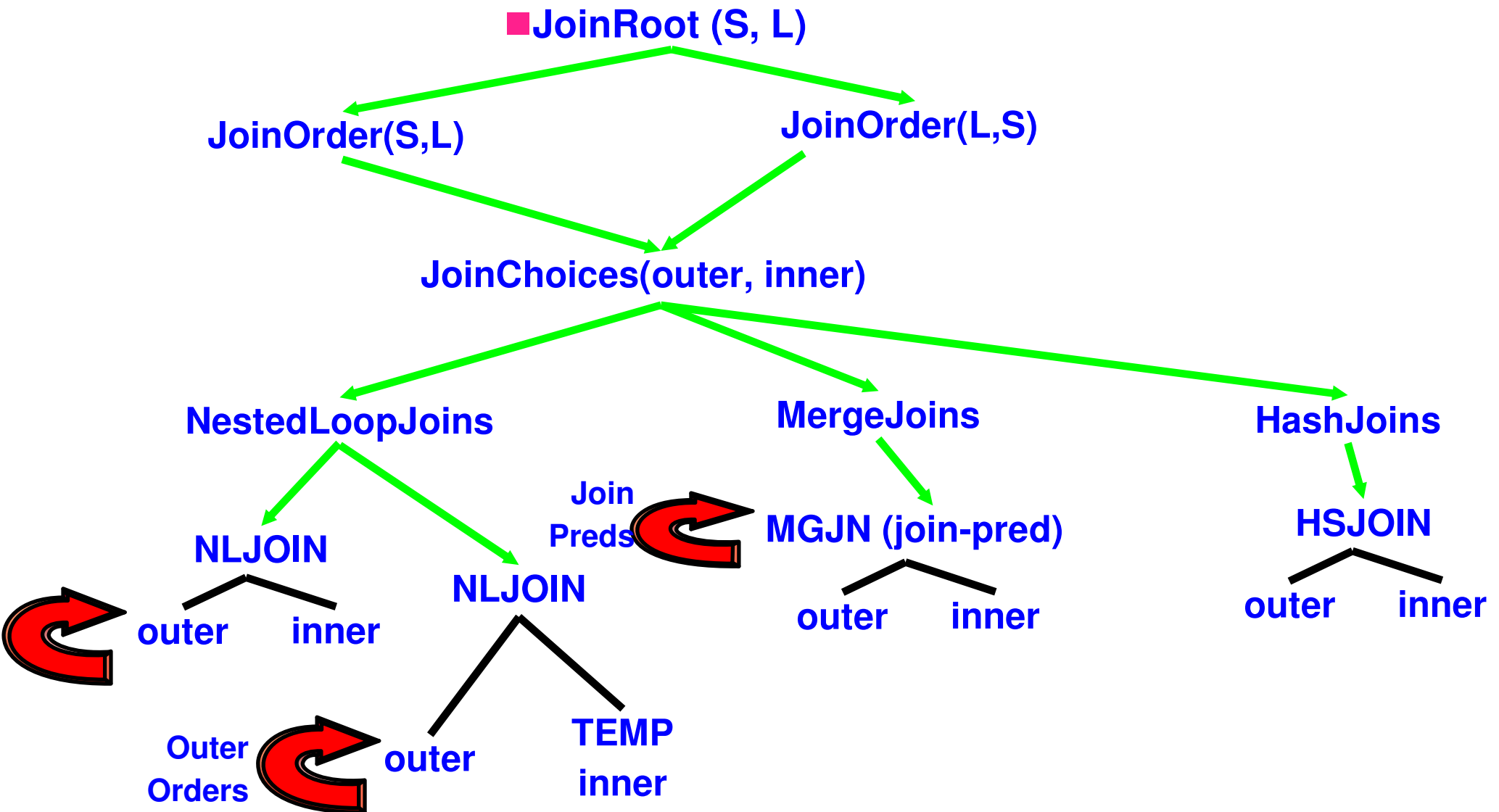
REFN: Guy M. Lohman,
 "Grammar-like Functional
 Rules for Representing Query
 Optimization Alternatives",
 SIGMOD 1988, pp. 18-27.

Existing
 Indexes



ISCAN

Generation of Join Alternatives



REFN: Guy M. Lohman, "Grammar-like Functional Rules for Representing Query Optimization Alternatives", SIGMOD 1988, pp. 18-27.

Optimizer Cost Model

■ Differing objectives: Minimize...

- Elapsed time, in parallel environments, OPTIMIZE FOR N ROWS
- Total resources, otherwise

■ Combines components of estimated

- CPU (# of instructions)
- I/O (random and sequential)
- Communications (# of IP frames)
 - Between nodes, in partitioned environments
 - Between sites, in DataJoiner environments

■ Detailed modeling of

- Buffer needed vs. available, hit ratios
- Rescan costs vs. build costs
- Prefetching and big-block I/O
- Non-uniformity of data
- Operating environment (via configuration parameters)
- First tuple costs (for OPTIMIZE FOR N ROWS)



Catalog Statistics Used by the Optimizer

■ Basic Statistics

- Number of rows/pages in table
- For each column in a table, records
 - # distinct data values, avg. length of data values, data range information
- For each index on a table,
 - # key values, # levels, # leaf pages, etc.

■ Non-uniform distribution statistics ("WITH DISTRIBUTION")

- N most frequent values (default 10) -- Good for equality predicates
- M quantiles (default 20) -- Good for range predicates
- N and M set by DBA as DB configuration parameters
- REFN: **Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, Eugene J. Shekita, "Improved Histograms for Selectivity Estimation of Range Predicates", SIGMOD 1996.**
- N and M can differ per column (New in V8.1!)

■ Index clustering (DETAILED index statistics)

- Empirical model: determines curve of I/O vs. buffer size
- Accounts for benefit of large buffers

■ User-defined function (UDF) statistics

- Can specify I/O & CPU costs
 - per function invocation
 - at function initialization
 - associated with input parameters

■ Column Group Statistics (multiple columns – for correlation)

Modifying Catalog Statistics

■ Statistics values are...

- **Readable** in the system catalogs
 - e.g., HIGH2KEY, LOW2KEY
- **Updateable**, e.g.

```
UPDATE SYSSTAT.TABLES
SET CARD = 1000000
WHERE TABNAME = 'NATION'
```

■ Implications:

- Can simulate a non-existent database
- Can "clone" a production database (in a test environment)

■ Tools

- DB2LOOK captures the table DDL and statistics to replicate an environment

Top-Down vs. Bottom-Up Conundrum

■ Bottom-up (System R, DB2, Oracle, Informix)

- Plans **MUST** be costed bottom-up (need input costs)
- Dynamic programming **REQUIRES** **breadth-first** enumeration to pick best
- Can't pick best plan until it's costed

■ Top-down (Volcano, Cascades, Tandem, SQL Server)

- Operators may **REQUIRE** certain properties (e.g. order or partitioning)
- Limit strategies based upon context of use

■ Solution in DB2:

- Plans built bottom-up, **BUT...**
- Pre-processing amasses candidate future requirements:
 - "Interesting" orders, e.g. for joins, **GROUP BY**, **ORDER BY**
 - "Interesting" partitions, in partitioned environment
 - Used to lump together "un-interesting" properties for pruning
- Operators requiring certain properties:
 - Call "**get-best-plan**" to find a plan with those properties
 - If none found, augment all plans with "**glue**" to get desired properties, e.g. add **SORT** to get desired Order, and pick cheapest
- Hence, could build a top-down (demand-driven) enumerator, using get-best-plan!

Extensible Search Strategy

- **Bottom-up generation of plans**
- **Parameterized search strategy**
 - **Dynamic Programming** (breadth-first, provably optimal, but expensive)
 - Build plans to access base tables
 - For $j = 2$ to # of tables:
 - Build j -way joins from best plans containing $j-1, j-2, \dots, 2, 1$ tables
 - **Greedy** (more efficient for large queries)
- **Generate 2 sets of tables to join, and filter "unjoinable" ones**
- **Parameterized search space**
 - Composite inners or not (actually, maximum # of quantifiers in smaller set)
 - Cartesian products (no join predicate) or not
 - Disable/enable individual rules generating strategies (e.g. hash joins)
- **Interfaces to add/replace entire search strategy**
- **Controlled by "levels of optimization" (1 – 9)**

- **REFN: Kiyoshi Ono, Guy M. Lohman, "Measuring the Complexity of Join Enumeration in Query Optimization", VLDB 1990, pp. 314-325.**

Agenda

- Overview of Query Processing
- Query ReWrite
- Plan Selection Optimization basics
 - Elements of Optimization
 - Execution Strategies
 - Cost model & plan properties
 - Search strategy
-  ■ Parallelism
- Special strategies for OLAP & BI
- Engineering considerations
- Conclusions and Future
- NOTE: Use DB2 for free for academic purposes! See:

<http://www.ibm.com/developerworks/university/data/index.html>

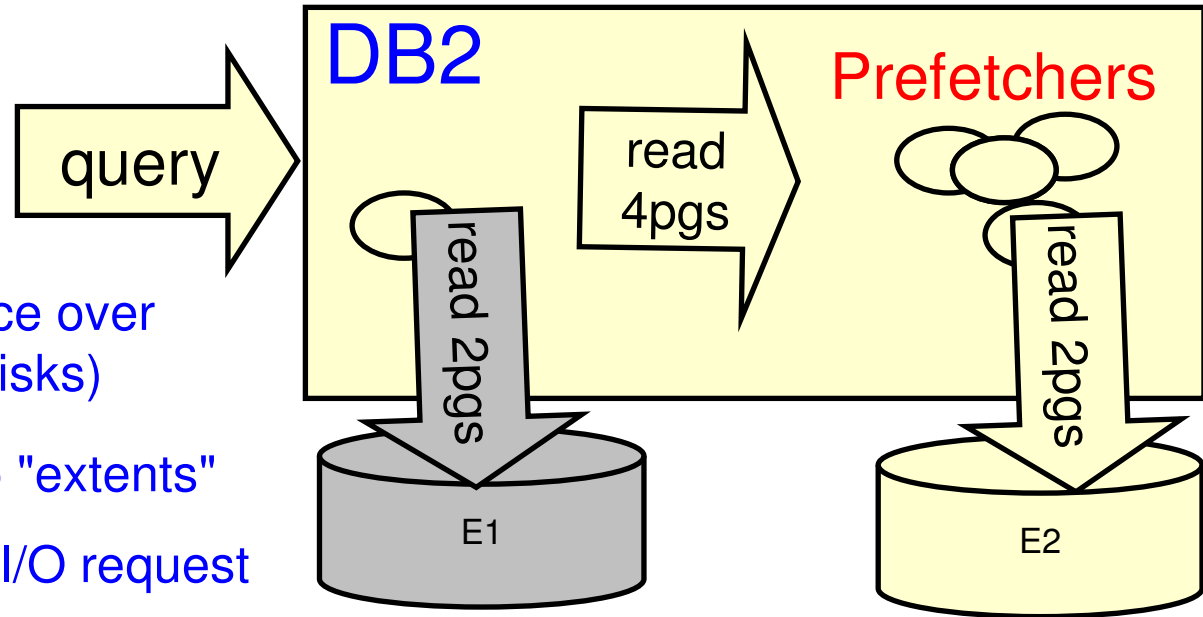
Types of Parallelism in DB2

- **I/O parallelism (data striping)**
- **Inter-partition parallelism (shared-nothing)**
- **Intra-partition parallelism (for SMPs)**



I/O Parallelism (multiple arms)

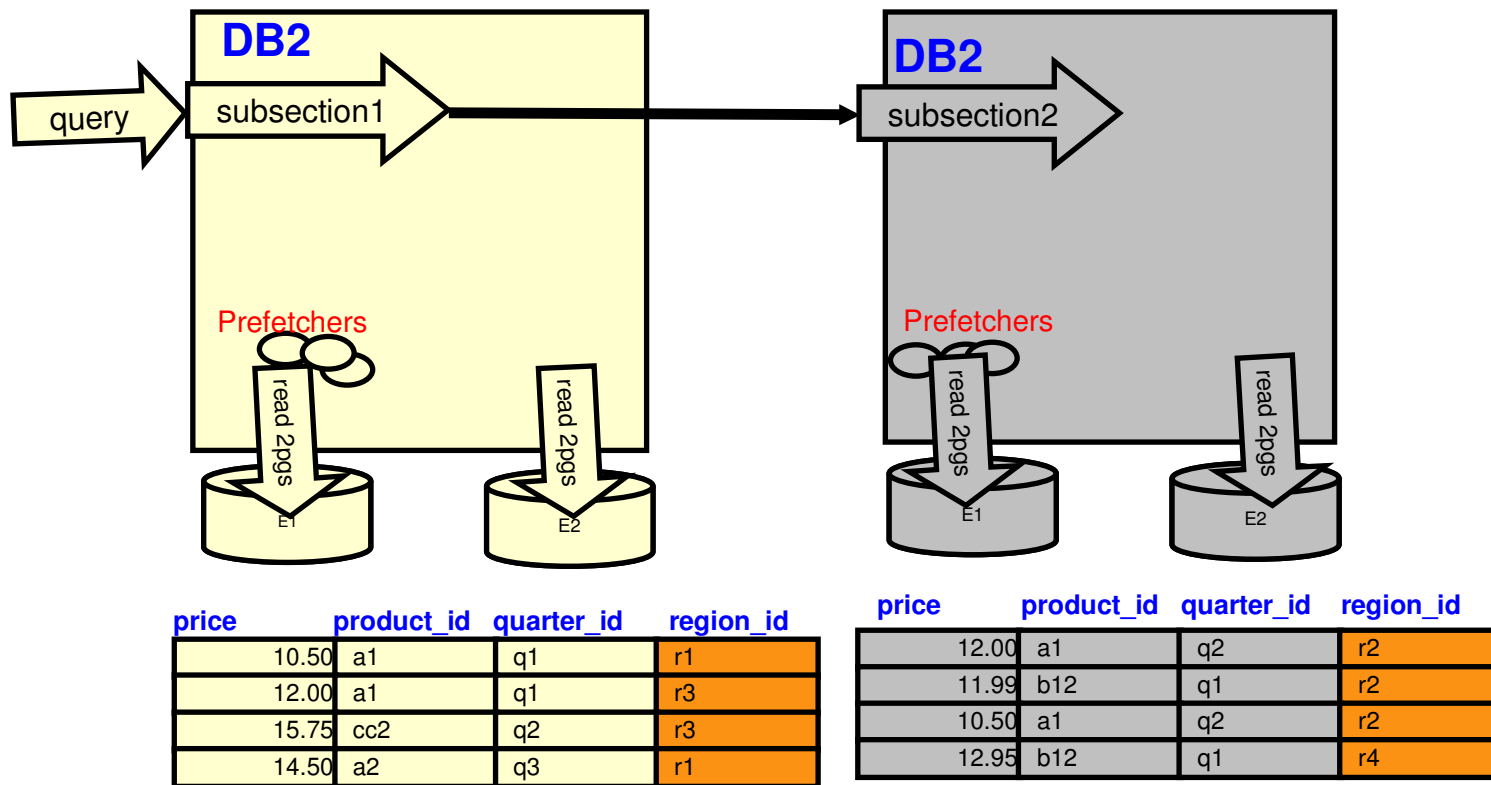
- Parallelism achieved by
 - ➔ User defining tablespace over multiple "containers" (disks)
 - ➔ DB2 breaking table into "extents"
 - ➔ DB2 breaking prefetch I/O request into multiple I/O requests



price	product_id	quarter_id	region_id
10.50	a1	q1	r1
12.00	a1	q1	r3
12.00	a1	q2	r2
11.99	b12	q1	r2
10.50	a1	q2	r2
15.75	cc2	q2	r3
14.50	a2	q3	r1
12.95	b12	q1	r4

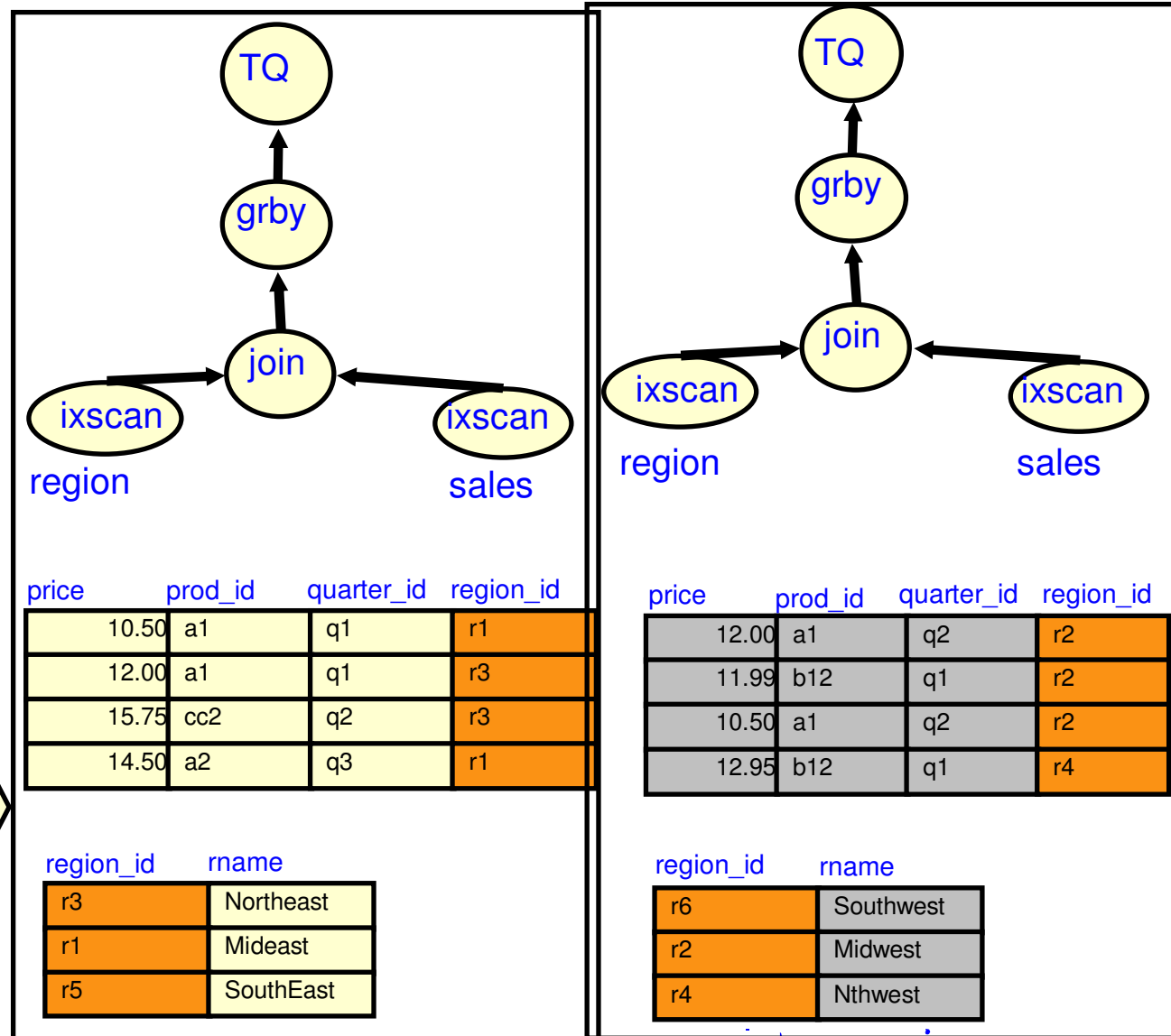
Inter-Partition Parallelism

- System configured with autonomous DB2 instances called "nodes"
 - typically with own CPU, memory, disks
 - connected by high-speed switch
 - can use logical nodes as well
- Tables stored partitioned among nodes via "partitioning key" column(s)



Optimizing Inter-Partition Parallelism

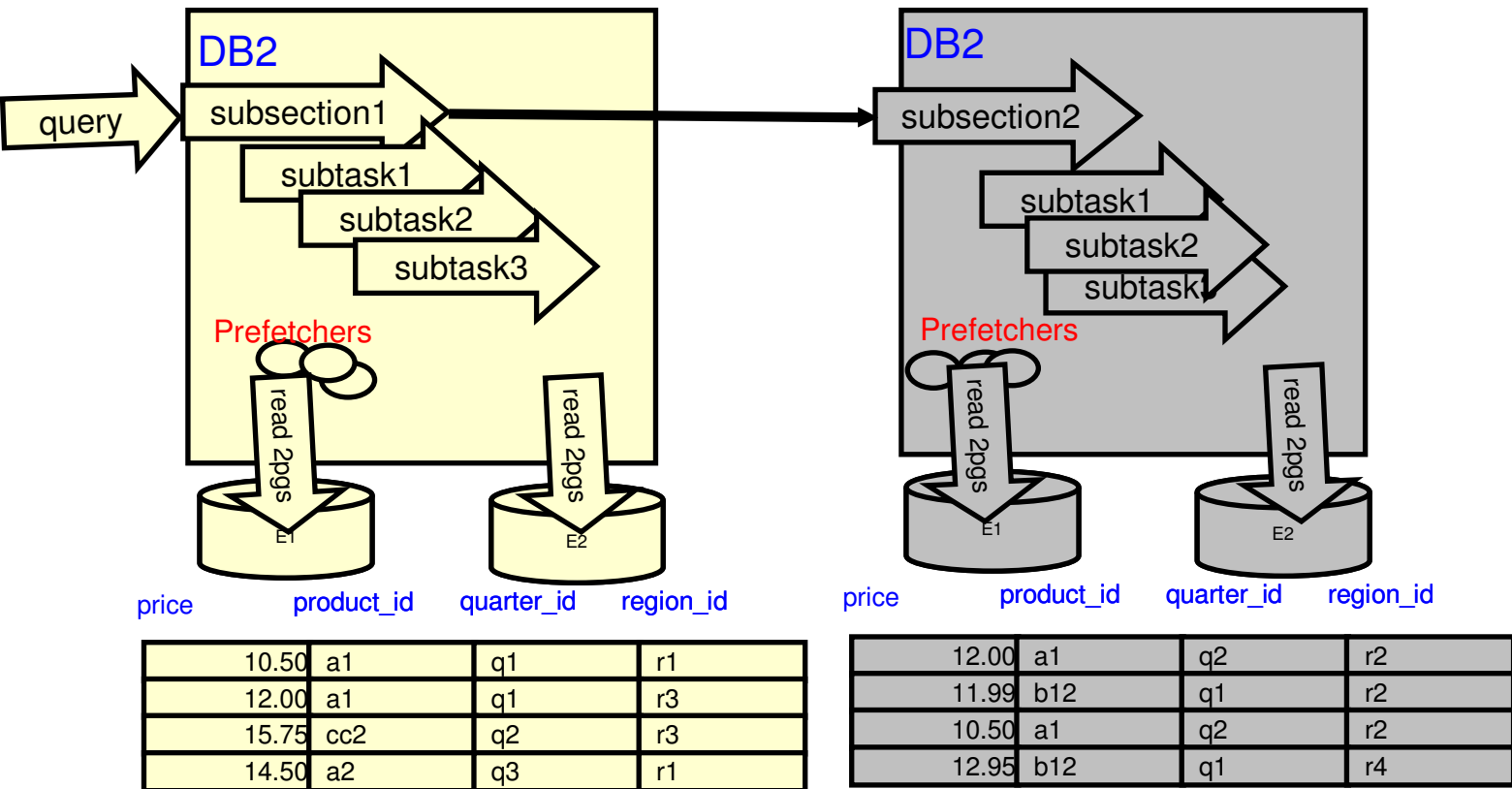
- Query (section) divided into parts (subsections) based upon...
 - ➔ How data is partitioned
 - ➔ Query's semantics
- All nodes assumed equal
- Function is shipped to data
 - ➔ Dynamic repartitioning might be required
- Goal of query optimization:
 - ➔ Minimize elapsed time



**select rname, sum(price),
from sales s, region r
where r.region_id = s.region_id
group by rname, r.region_id**

Intra-Partition Parallelism

- Exploits multiple processors of a symmetric multiprocessor (SMP)
- Multiple agents work on a single plan fragment
- Workload is dynamically balanced at run-time (“Straw Model”)
- Post-optimizer parallelizes best serial/partitioned plan
- Degree of parallelism determined by compiler and run-time, bounded by config. parm.

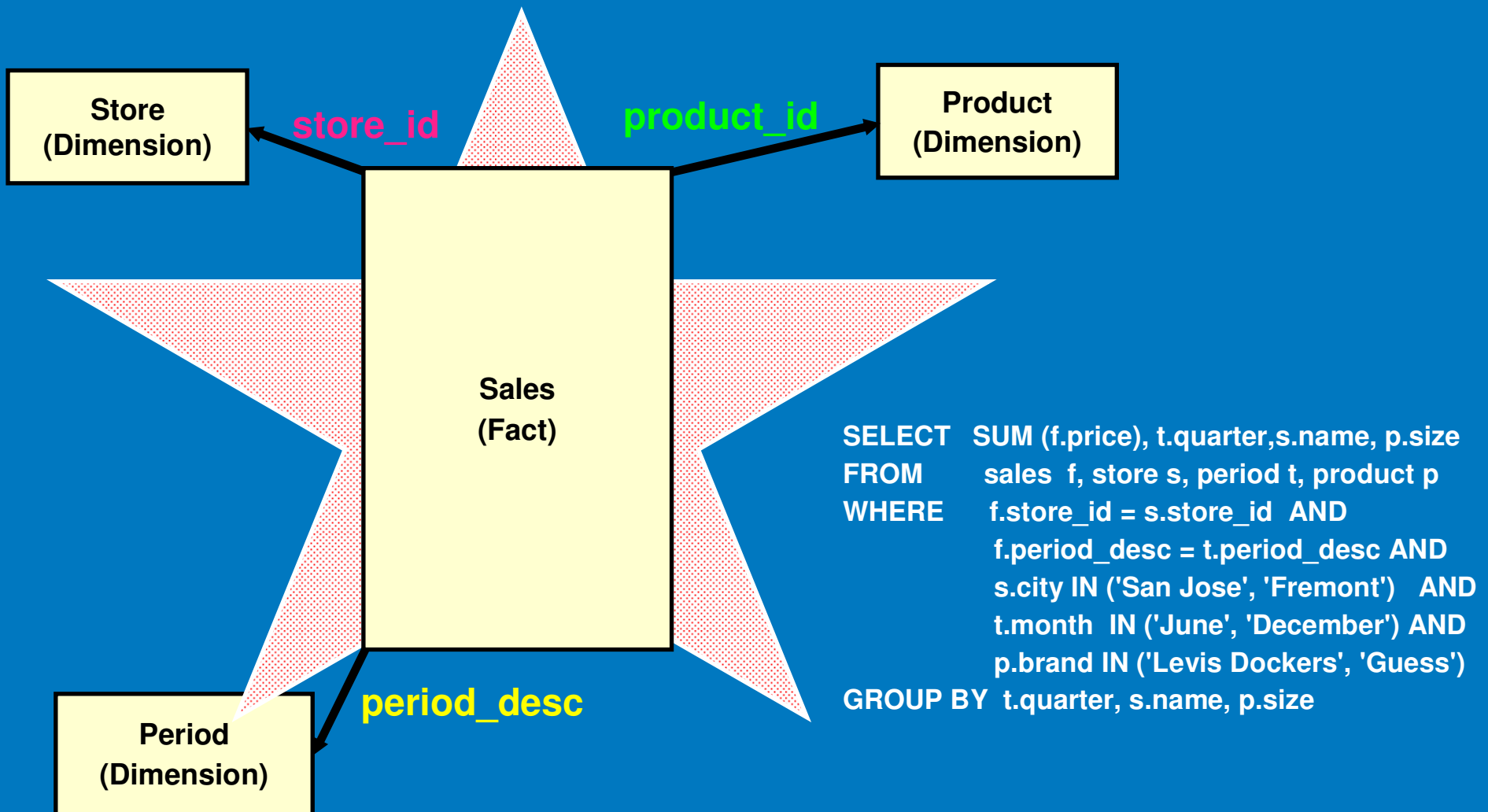


Agenda

- Overview of Query Processing
- Query ReWrite
- Plan Selection Optimization basics
 - Elements of Optimization
 - Execution Strategies
 - Cost model & plan properties
 - Search strategy
- Parallelism
- ➡ ■ Special strategies for OLAP & BI
- Engineering considerations
- Conclusions and Future
- NOTE: Use DB2 for free for academic purposes! See:

<http://www.ibm.com/developerworks/university/data/index.html>

An OLAP Query to a Star Schema:



Why are Special Strategies Needed?

- Optimizer avoids Cartesian joins (since no join predicates)
- Typically there are no join predicates between dimension tables
- So some table must join with Fact table
- Predicates on any one dimension insufficient to limit # of rows
- Large intermediate result (millions to 100s of millions) for next join!
- Therefore, intersection of limits on many dimensions are needed!

Why are Special Strategies Needed?

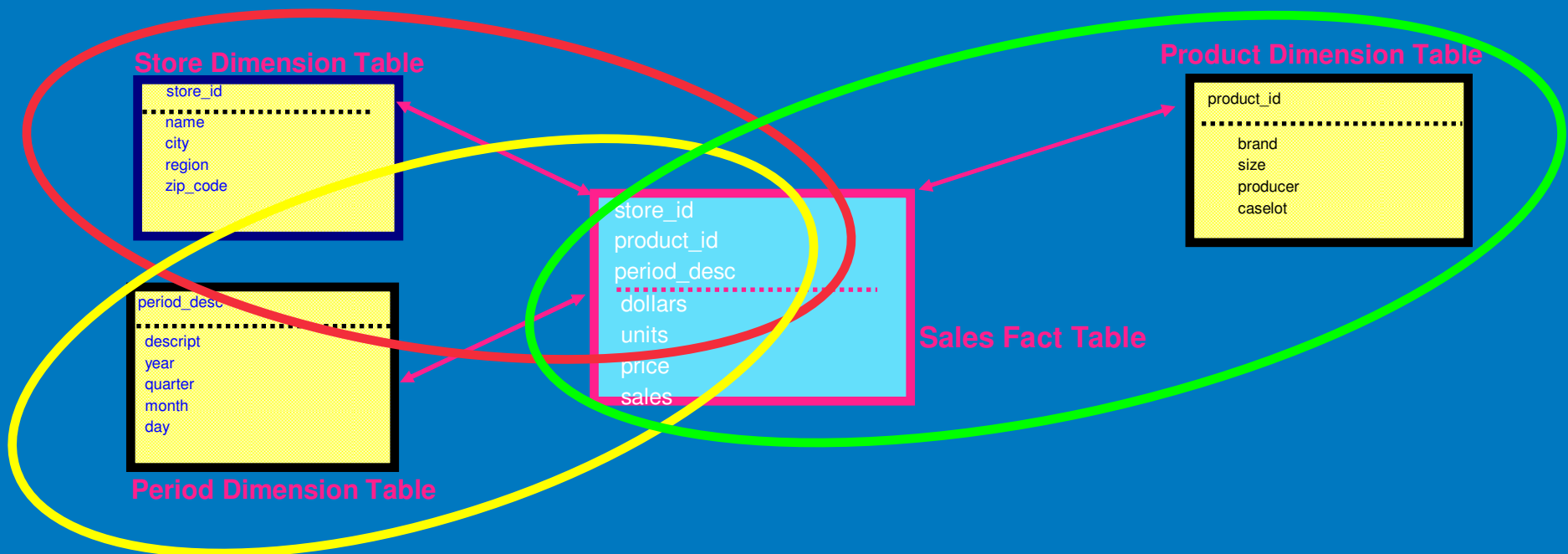
- EXAMPLE:

1 City = 'San Jose': 10s of millions of sales in San Jose stores!

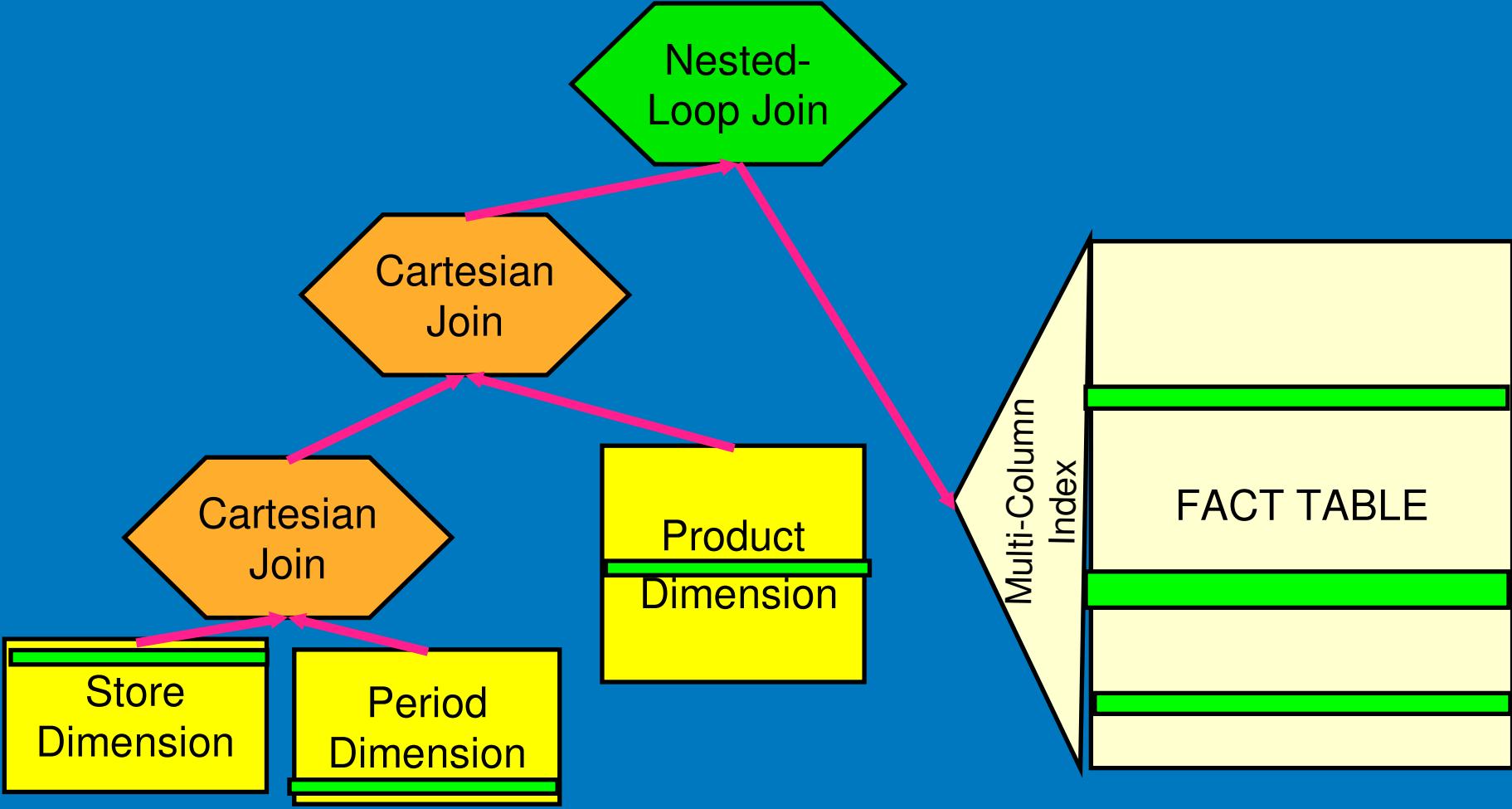
2 Month = 'December': 100s of millions of sales in December!

3 Brand = 'Levi Dockers': millions of Levi's Dockers!

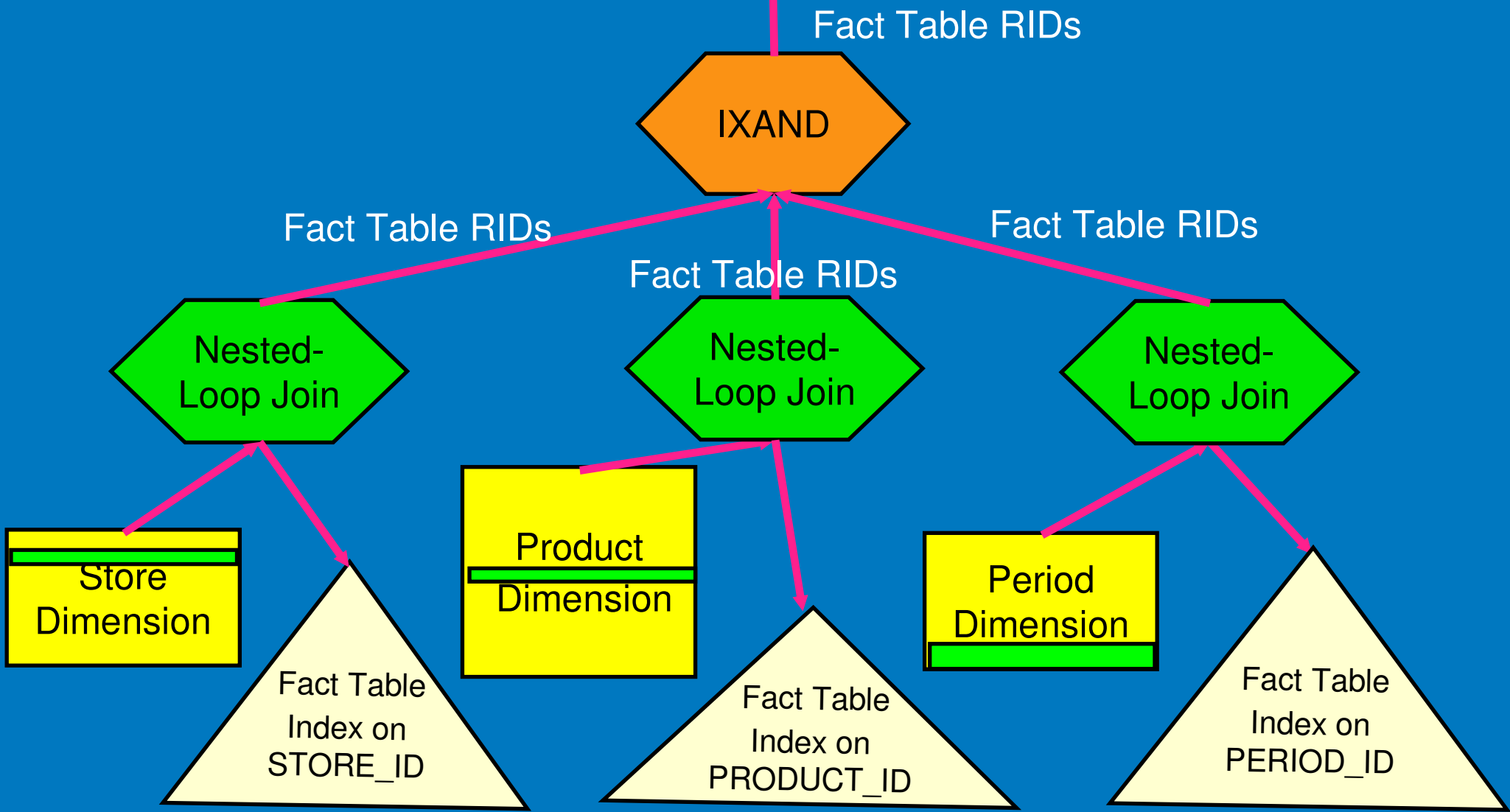
- TOGETHER: only thousands of Levi Dockers sold in San Jose stores in December!!



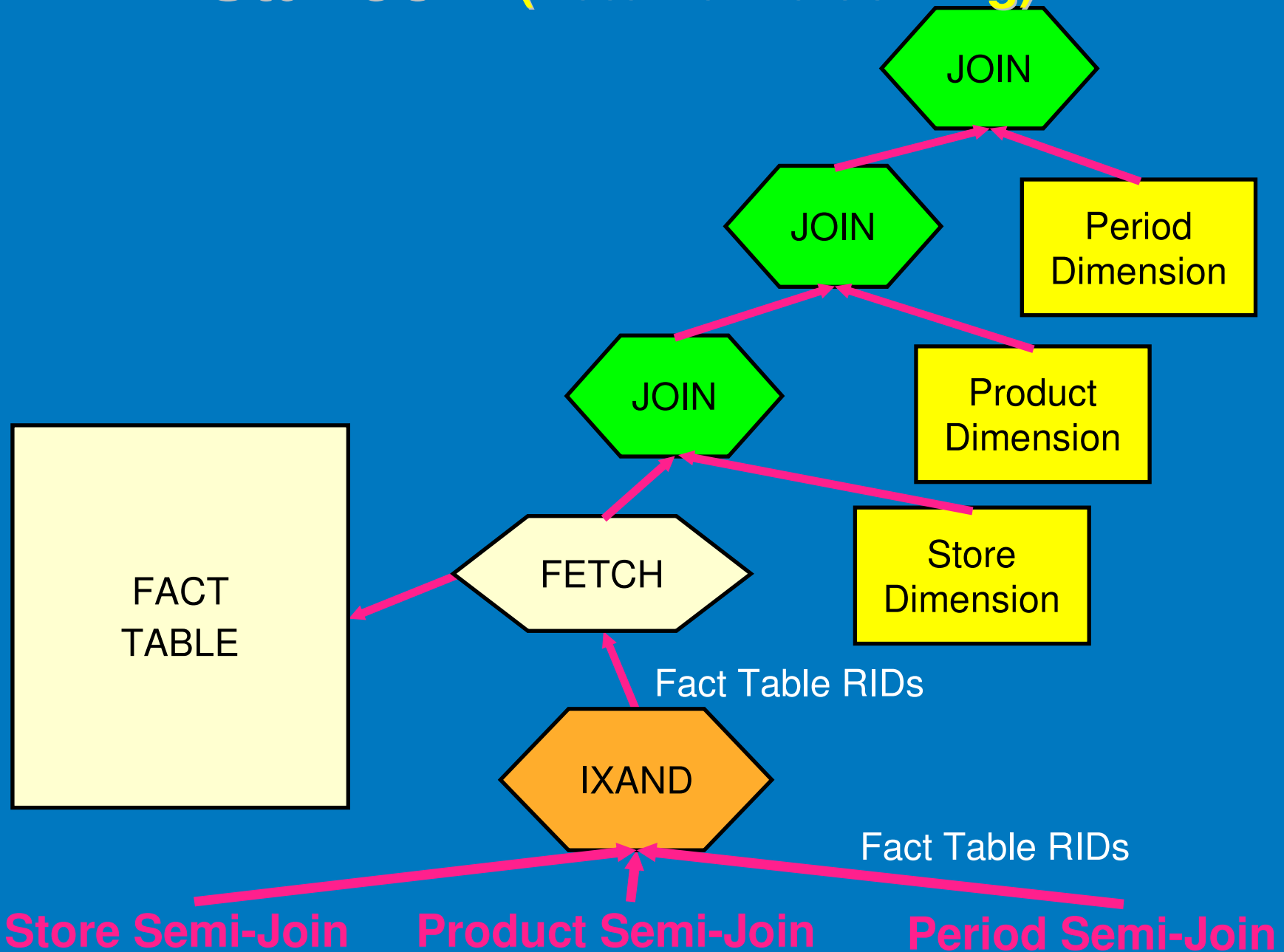
Special Strategy 1: Cartesian-Join of Dimensions



Special Strategy 2: Star Join (semi-join ANDing)



Special Strategy 2: Star Join (Fetch & Re-Joining)



Agenda

- Overview of Query Processing
- Query ReWrite
- Plan Selection Optimization basics
 - Elements of Optimization
 - Execution Strategies
 - Cost model & plan properties
 - Search strategy
- Parallelism
- Special strategies for OLAP & BI
- ➡ ■ Engineering considerations
- Conclusions and Future
- NOTE: Use DB2 for free for academic purposes! See:

<http://www.ibm.com/developerworks/university/data/index.html>

Product-Quality Query Optimizers Must: Support ALL of SQL

- **Subqueries, including expressions of subqueries**
 - **Much more challenging in distributed environment!**
- **Correlation (very complex!)**
- **IN lists**
- **LIKE predicates, with wildcard characters (*,%)**
- **Cursors and WHERE CURRENT OF CURSOR statements**
- **IS NULL and IS NOT NULL**
- **Enforcement of constraints (column, referential integrity)**
- **EXCEPT, INTERSECT, UNION**
 - **ALL**
 - **DISTINCT**
- **Lots more...**



Product-Quality Query Optimizers Must: Address High-Performance Aspects

- **No limits on number of tables, columns, predicates, ...**
- **Efficient utilization of space**
 - representation of sets of objects using bit-vectors
 - location and sharing of sub-plans
 - garbage collection
- **Multi-column indexes, each with start and/or stop key values**
- **Ascending/Descending sort orders (by column)**
- **Implied predicates ($T.a = U.b$ AND $U.b = V.c \implies T.a = V.c$)**
- **Clustering and "density" of rows for page FETCH costing**
- **Optional TEMP_s and SORT_s to improve performance**
- **Non-uniform distribution of values**
- **Sequential & list prefetching of pages**
- **Random vs. sequential I/Os**
- **OPTIMIZE FOR N ROWS, FETCH FIRST N ROWS ONLY**
- **Pipelining and "dams"**

Product-Quality Query Optimizers Must: Deal with Details

- "Halloween problem" on UPDATE/INSERT/DELETE, e.g.

UPDATE Emp SET salary = salary *1.1

WHERE salary > 120K

If an ascending index on salary is used, and no TEMP,

- Everyone gets an infinite raise!
- UPDATE never completes!

- Differing code pages (e.g., Kanji, Arabic, Unicode, ...), esp. in indexes
- Isolation levels
- Lock intents



Things I Didn't Have Time to Discuss

- **Partitioning and clustering**
 - Range Partitioning and Partition Elimination
 - Partitioned indexes
 - Multi-Dimensional Clustering (MDC)
- **Sophisticated querying constructs**
 - OLAP windowing functions (RANK, DENSE, ROLLUP, CUBE)
 - XML Optimization (“pureXML”)
 - Sampling in queries and RUNSTATS
 - Common Sub-Expressions (DAGs) and Recursive SQL
- **Richer Statistics**
 - Column group statistics
 - Statistical views
 - Real-time statistics
- **Performance, performance, performance!**
 - Materialized Query Tables (Materialized Views)
 - Pushing GROUP BY & aggregation through joins
 - Autonomic Features (Design Advisor, LEO, Auto RUNSTATS, ...)
 - Plan stability
 - Compression of tables, indexes, and temp tables!
 - Varying page sizes
- **1000 little improvements... and lots, lots more!**



Summary & Future

- **Industry-Leading Optimization**
- **Extensible – even to XQuery!**
- **Optimizes for Parallel**
 - I/O accesses
 - Within a node (SMP)
 - Between nodes (MPP)
- **Powerful for complex OLAP & BI queries**
- **Industry-Strength Engineering**
- **Portable**
 - Across HW & SW platforms
 - Databases of 1 GB to > 0.5 PB
- **Continuing "technology pump" of improvements from Research, e.g.**
 - **Autonomic features**
 - **pureXML**
 - **REFN: “System RX: One Part Relational, One Part XML”, SIGMOD 2005; “Cost-Based Optimization in DB2/XML”, IBM Systems Journal, May 2006**

धन्यवाद
Hindi

多謝
Traditional Chinese

Ευχαριστώ!
Greek

ขอบคุณ
Thai

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

Thank You
English

Obrigado
Brazilian Portuguese

Grazie
Italian

多谢
Simplified Chinese

Danke
German

Merci
French

நன்றி
Tamil

ありがとうございました
Japanese

감사합니다
Korean

Appendix:



Backup Foils

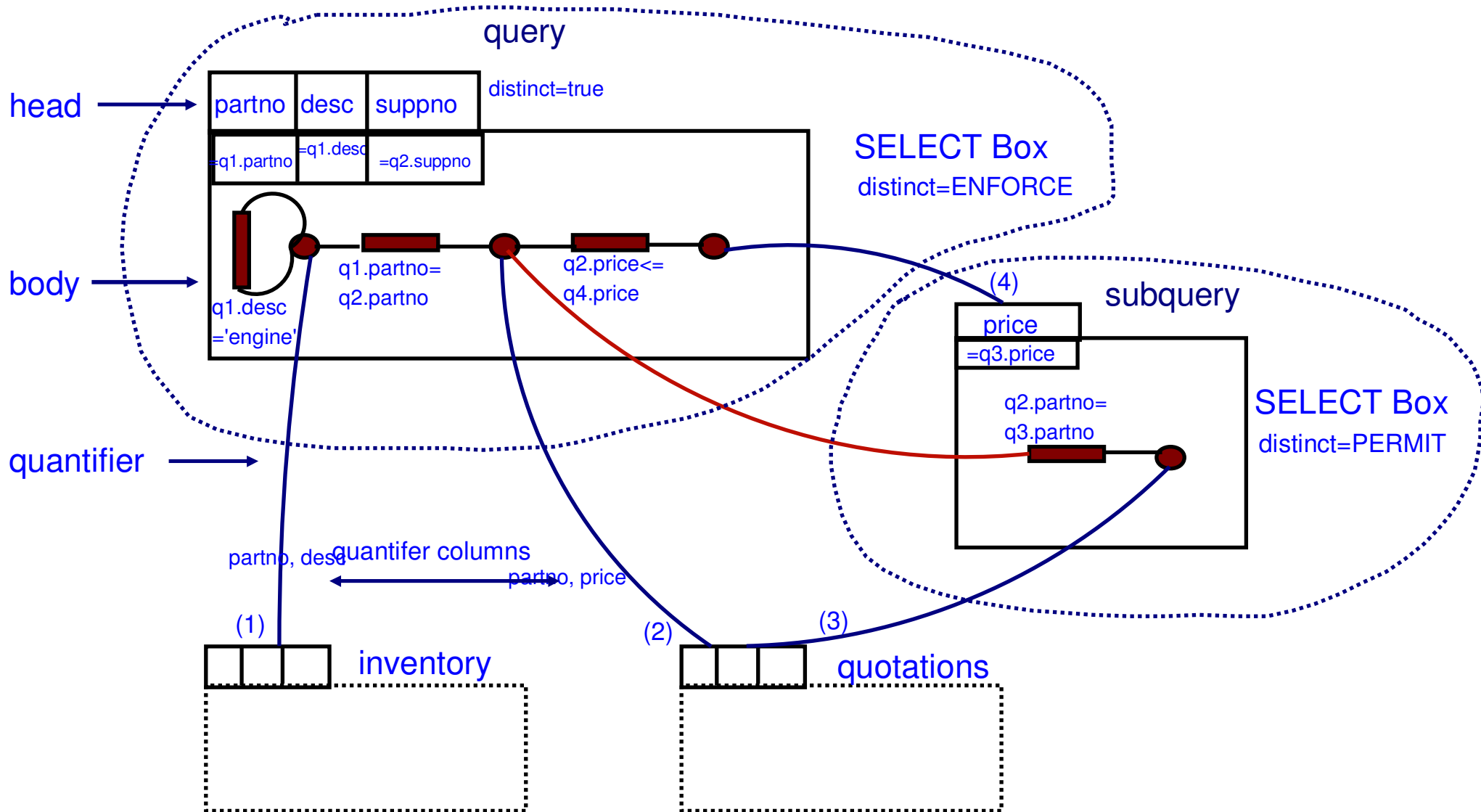
IBM Software

Example QGM for a Query

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
FROM inventory q1, quotations q2
WHERE q1.partno = q2.partno
      AND q1.descr = 'engine'
      AND q2.price <= ALL
          ( SELECT q3.price
            FROM quotations q3
            WHERE q2.partno = q3.partno
          );
```



QGM Graph (after Semantics)



Query Rewrite - A VERY Simple Example

Original Query:

```
SELECT DISTINCT custkey, name FROM tpcd.customer
```

After Query Rewrite:

```
SELECT custkey, name FROM tpcd.customer
```

Rationale:

custkey is unique, DISTINCT is redundant

Query Rewrite - Operation Merge

■ **Goal: give Optimizer maximum latitude in its decisions**

■ **Techniques:**

– **View merge**

- **makes additional join orders possible**
- **can eliminate redundant joins**

– **Subquery-to-join transformation**

- **removes restrictions on join method/order**
- **improves efficiency**

– **Redundant join elimination**

- **satisfies multiple references to the same table with a single scan**



Query Rewrite: Subquery-to-Join Example:

■ Original Query:

```
SELECT ps.*
FROM   tpcd.partsupp ps
WHERE  ps.ps_partkey IN
      (SELECT p_partkey
       FROM tpcd.parts
       WHERE p_name LIKE 'forest%');
```

■ Rewritten Query:

```
SELECT ps.*
FROM   parts, partsupp ps
WHERE  ps.ps_partkey = p_partkey AND
      p_name LIKE `forest%`;
```

**NOTE: Unlike Oracle, DB2 can do this transform,
even if p_partkey is NOT a key!**

Query Rewrite - Operation Movement

■ **Goal: minimum cost / predicate**

■ **Techniques:**

– **Distinct Pushdown**

- **Allow optimizer to eliminate duplicates early, or not**

– **Distinct Pullup**

- **To avoid duplicate elimination**

– **Predicate Pushdown**

- **Apply more selective and cheaper predicates early on;**
- **e.g., push into UNION, GROUP BY**

Query Rewrite - Shared Aggregation Example

■ Original Query:

```
SELECT SUM(O_TOTAL_PRICE) AS OSUM,  
       AVG(O_TOTAL_PRICE) AS OAVG  
FROM ORDERS;
```

■ Rewritten Query:

```
SELECT OSUM, OSUM/OCOUNT AS OAVG  
FROM (SELECT SUM(O_TOTAL_PRICE) AS OSUM,  
           COUNT(O_TOTAL_PRICE) AS OCOUNT  
      FROM ORDERS) AS SHARED_AGG;
```

→ Reduces query from 2 sums and 1 count to 1 sum and 1 count!



Query Rewrite - Predicate Translation

■ GOAL: optimal predicates

■ Examples:

– Distribute NOT

- ... WHERE NOT(COL1 = 10 OR COL2 > 3)

becomes

- ... WHERE COL1 <> 10 AND COL2 <= 3

– Constant expression transformation:

- ... WHERE COL = YEAR(`1994-09-08`)

becomes

- ... WHERE COL = 1994

– Predicate transitive closure, e.g., given predicates:

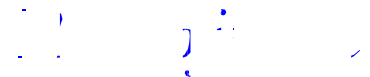
- T1.C1 = T2.C2, T2.C2 = T3.C3, T1.C1 > 5

add these predicates...

- T1.C1 = T3.C3 AND T2.C2 > 5 AND T3.C3 > 5

– IN-to-OR conversion for Index ORing

– and many more...



Query Rewrite - Correlated Subqueries Example

■ Original Query:

```
SELECT PS_SUPPLYCOST FROM PARTSUPP
WHERE PS_PARTKEY <> ALL
      (SELECT L_PARTKEY FROM LINEITEM
       WHERE PS_SUPPKEY = L_SUPPKEY)
```

■ Rewritten Query:

```
SELECT PS_SUPPLYCOST FROM PARTSUPP
WHERE NOT EXISTS
      (SELECT 1 FROM LINEITEM
       WHERE PS_SUPPKEY = L_SUPPKEY
         AND PS_PARTKEY = L_PARTKEY)
```

→ Pushes down predicate to enhance chances of binding partitioning key for each correlation value (here, from PARTSUPP)

Query Rewrite - Decorrelation Example

■ Original Query:

```
SELECT SUM(L_EXTENDEDPRISE)/7.0
FROM LINEITEM, PART P
WHERE P_PARTKEY = L_PARTKEY AND
      P_BRAND = 'Brand#23' AND
      P_CONTAINER = 'MED BOX' AND
      L_QUANTITY < (SELECT 0.2 * AVG(L1.L_QUANTITY)
                    FROM TPCD.LINEITEM L1
                    WHERE L1.L_PARTKEY = P.P_PARTKEY)
```

■ Rewritten Query:

```
WITH GBMAGIC AS (SELECT DISTINCT P_PARTKEY FROM PART P
                 WHERE P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED BOX'),
CTE AS (SELECT 0.2*SUM(L1.L_QUANTITY)/COUNT(L1.L_QUANTITY) AS AVGL_QUANTITY,
        P.PARTKEY FROMLINEITEM L1, GBMAGIC P
        WHERE L1.L_PARTKEY = P.P_PARTKEY GROUP BYP.P_PARTKEY)
SELECT SUM(L_EXTENDEDPRISE)/7.0 AS AVG_YEARLY
FROM LINEITEM, PART P WHERE P_PART_KEY = L_PARTKEY
AND P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED_BOX'
AND L_QUANTITY < (SELECT AVGL_QUANTITY FROM CTE
                  WHERE P_PARTKEY = CTE.P_PARTKEY);
```

→ This SQL computes the avg_quantity per unique part and can then broadcast the result to all nodes containing the lineitem table.

Optimizer -- Key Objectives

■ Extensible (technology from Starburst)

- Clean separation of execution "repertoire", cost eqns., search algorithm
- Cost & properties modularized per operator
- easier to add new operators, strategies
- Adjustable search space
- Object-relational features (user-defined types, methods)

■ Parallel (intra-query)

- CPU and I/O (e.g., prefetching)
- (multi-arm) I/O (i.e., striping)
- Shared-memory (i.e., SMP)
- Shared-nothing (i.e. MPP with pre-partitioned data)

■ Powerful / Sophisticated

- OLAP support
 - Star join
 - ROLLUP
 - CUBE
- Recursive queries
- Statistical functions (rank, linear recursion, etc.)
- and many more...



Example Properties

■ Relational ("What?")

- Tables (quantifiers) accessed
- Columns accessed
- Predicates applied
- Correlation columns referenced
- Keys -- columns on which rows distinct
- Functional dependencies

■ Physical ("How?")

- Columns on which rows ordered
- Columns on which rows partitioned (partitioned environment only)
- Physical site (DataJoiner only)

■ Derived ("How much?")

- Cardinality (estimated number of rows)
- Maximum provable cardinality
- **Estimated cost**, including separated:
 - Total cost
 - CPU (# of instructions)
 - I/O
 - Re-scan costs
 - 1st-row costs (for OPTIMIZE FOR N ROWS)

■ Flags, e.g. Pipelined, Halloween, etc.

■ REFN: M. K. Lee, J. C. Freytag, G. M. Lohman, "Implementing an Interpreter for Functional Rules in a Query Optimizer", VLDB 1988, 218-229.

Explaining Access Plans

■ Visual Explain

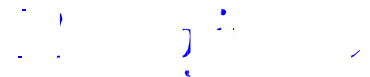
- accessible through DB2 Control Center
- graphical display of query plan
- uses optimization information captured by the optimizer
- invoke with either:
 - SET CURRENT EXPLAIN SNAPSHOT
 - EXPLSNAP bind option
 - EXPLAIN statement with snapshot option

■ Explain tables

- EXPLAIN statement / bind option
- superset of DB2 for MVS/ESA
- SET CURRENT EXPLAIN MODE
- optionally, generate report with DB2EXFMT tool

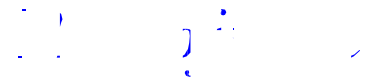
■ EXPLAIN utility (DB2EXPLN)

- explains bound packages into a flat file report
- similar to Version 1 but with many enhancements to usability
- less detailed information than EXPLAIN or Visual Explain



Query Optimization Level

- **Optimization requires**
 - Processing time
 - Memory
- **Users can control resources applied to query optimization**
 - Similar to the -O flag in a C compiler
 - Special register, for dynamic SQL
 - set current query optimization = 1
 - Bind option, for static SQL
 - bind tpcc.bnd queryopt 1
 - Database configuration parameter, for default
 - update db cfg for <db> using dft_queryopt <n>
- **Static & dynamic SQL may use different values**



Query Optimization Level Meaning

■ Use greedy join enumeration

–0 - minimal optimization for OLTP

- use index scan and nested-loop join
- avoid some Query Rewrite

–1 - low optimization

- rough approximation of Version 1 of DB2

–2 - full optimization, limit space/time

- use same query transforms & join strategies as class 7

■ Use dynamic programming join enumeration

–3 - moderate optimization

- rough approximation of DB2 for MVS/ESA

–5 - self-adjusting full optimization (default -- Autonomic!)

- uses all techniques with heuristics

–7 - full optimization

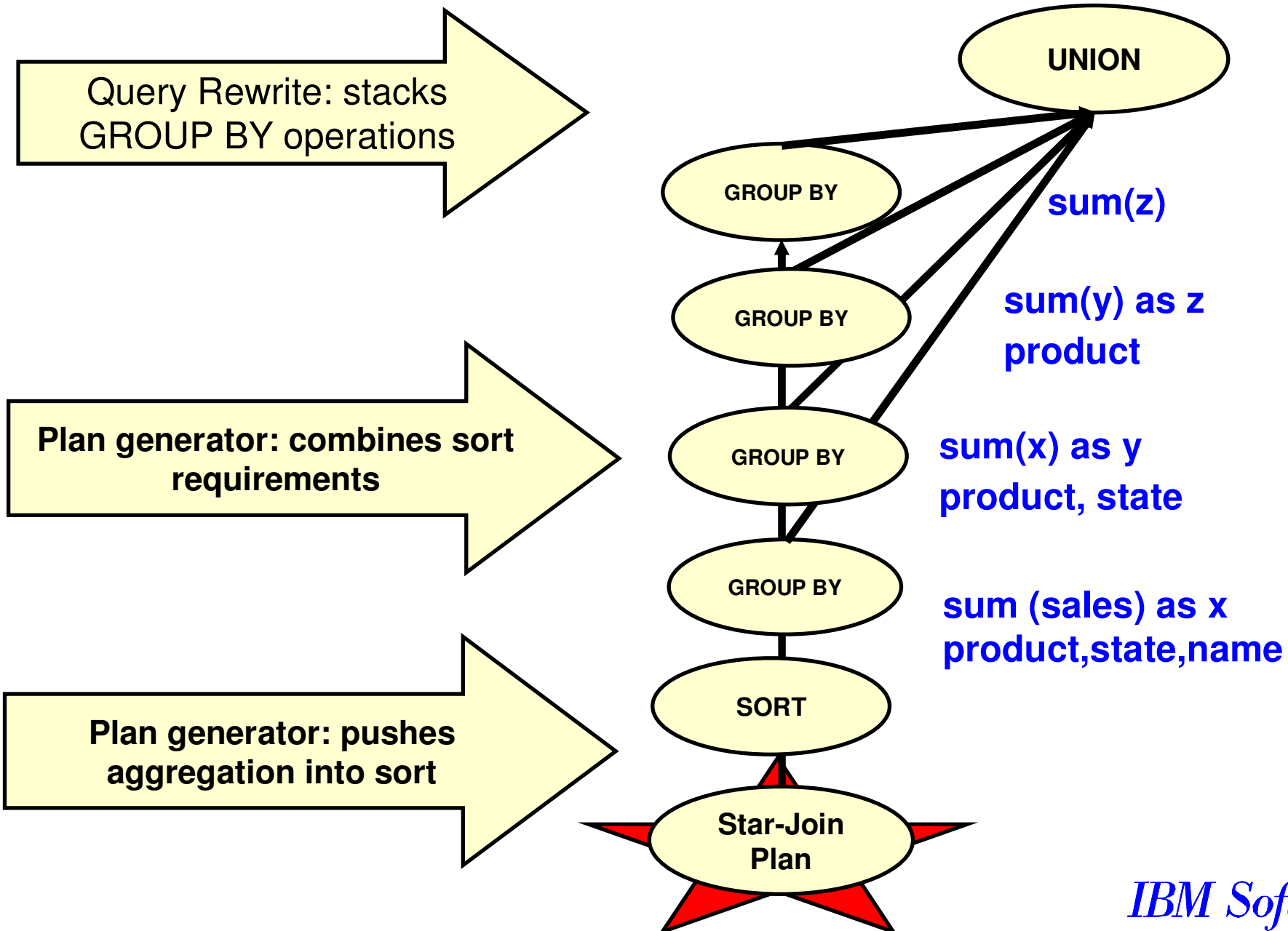
- similar to 5, without heuristics

–9 - maximal optimization

- spare no effort/expense
- considers all possible join orders, including Cartesian products!

■ REFN: Ihab F. Ilyas, Jun Rao, Guy M. Lohman, Dengfeng Gao, Eileen Lin, "Estimating Compilation Time of a Query Optimizer", SIGMOD 2003, pp. 373-384

DB2 UDB ROLAP optimization: ROLLUP



IBM Software

Dynamic Bitmap Index ANDing

- Takes advantage of indexes to apply "AND" predicates

- Selection is cost based, competing with:

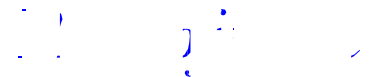
- Table scans
- Index ORing
- List prefetch

- Works by:

- Hashing Row Identifier (RID) values for qualifying rows of each index scan
- Dynamically build bitmap using hashed RIDs
- "AND" together bitmaps in a build-and-probe fashion
- Last index scan probes bitmap and returns qualifying RID
- Fetch qualifying rows

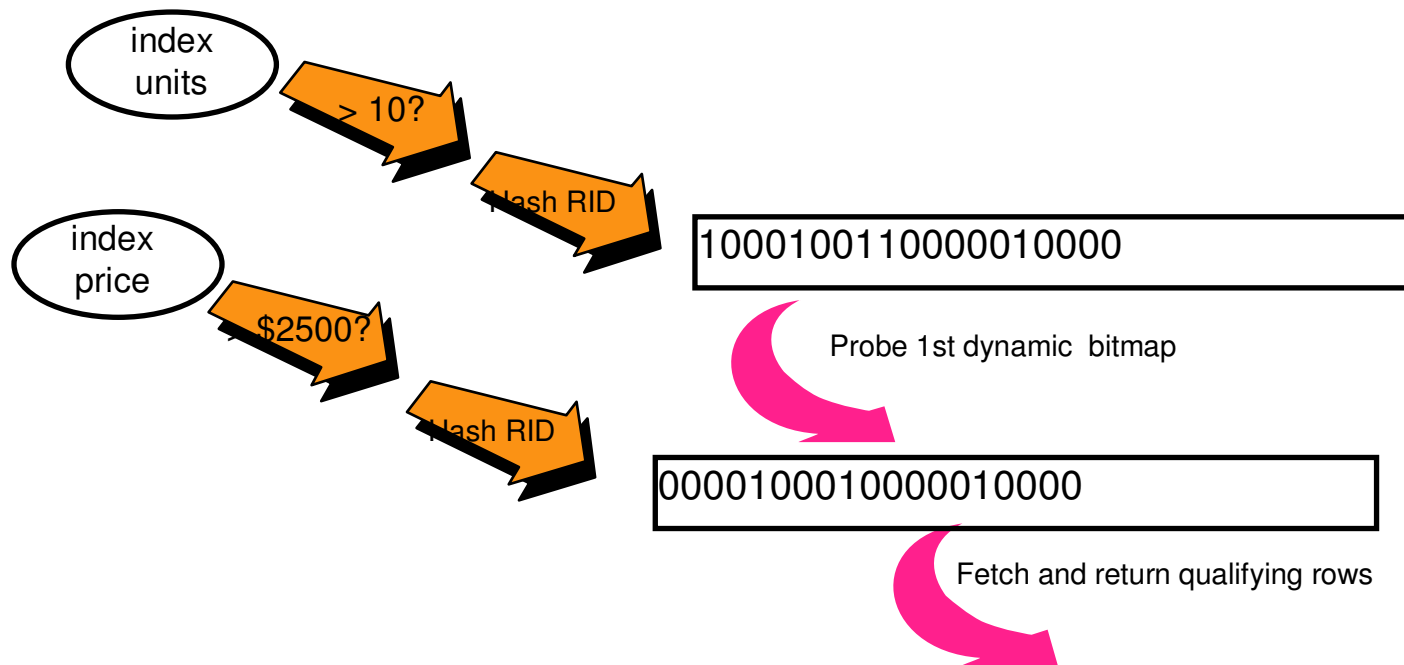
- Advantages:

- Can apply multiple ANDed predicates to different indexes, and get speed of index scanning



Dynamic Bitmap Index ANDing

- Count All products with price > \$2500 and units > 10



SQL/XML Parser

XQuery Parser

DB2 Compiler

Query Model

Run Time

XNav

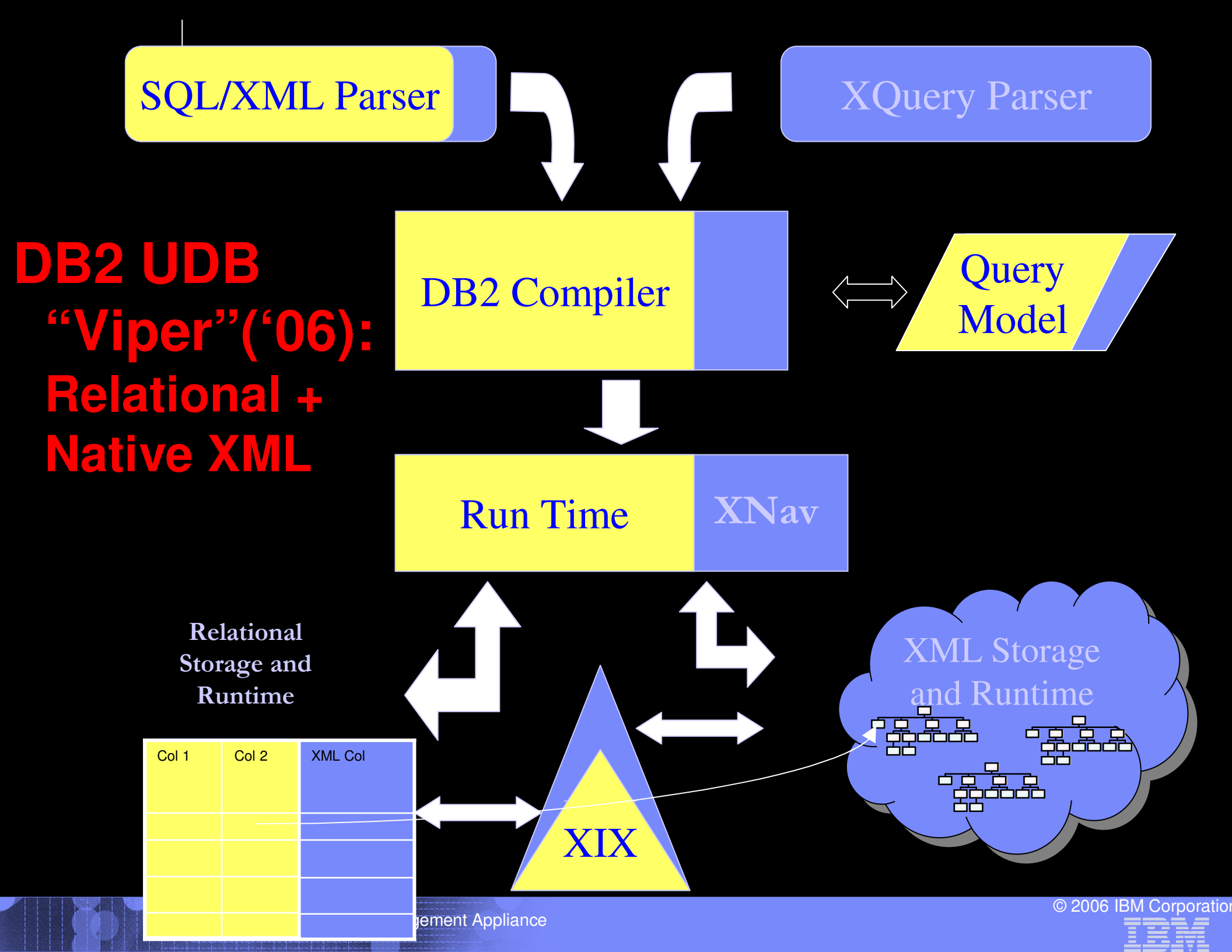
Relational Storage and Runtime

Col 1	Col 2	XML Col

XML Storage and Runtime

XIX

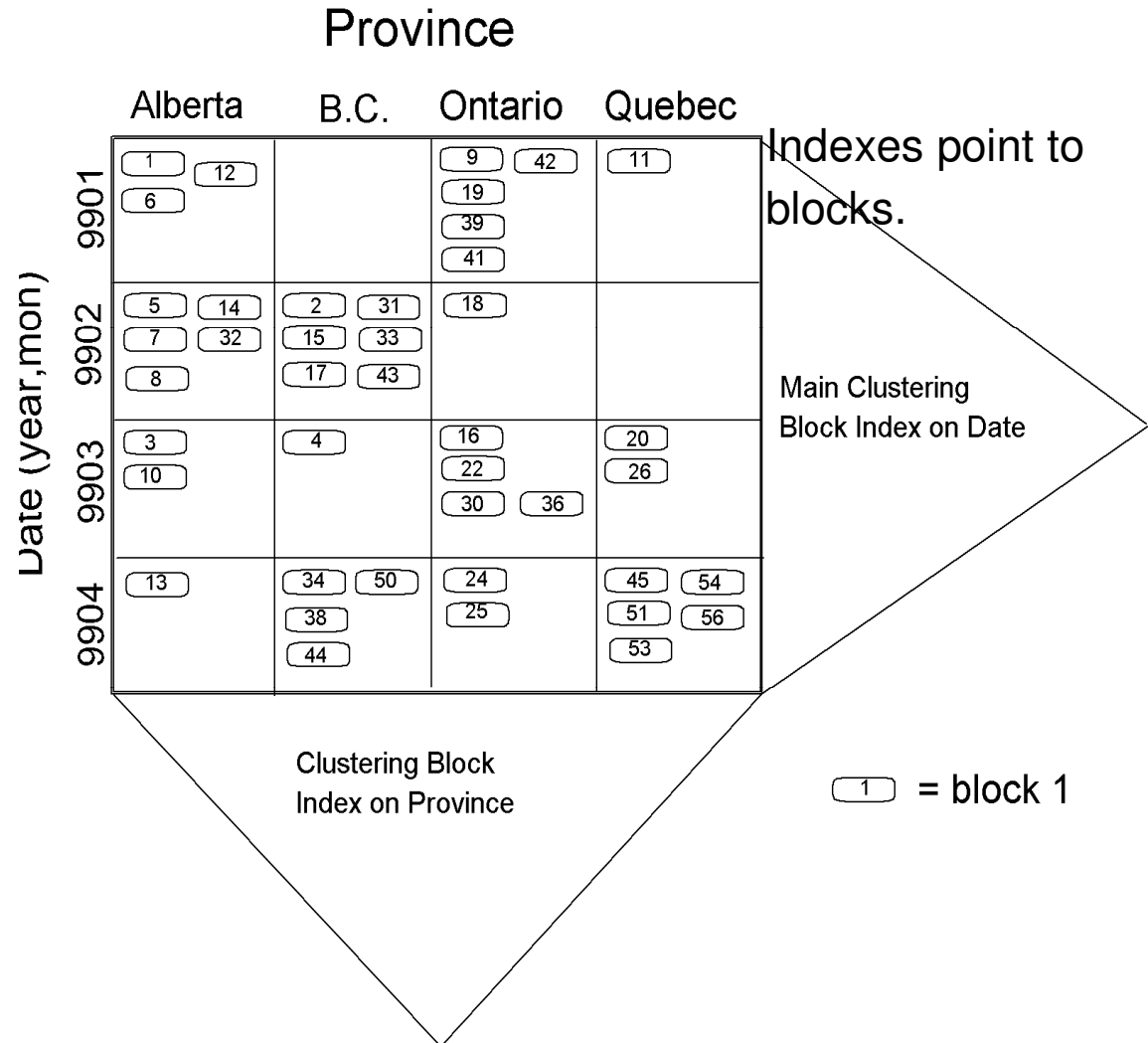
DB2 UDB
“Viper”(‘06):
Relational +
Native XML



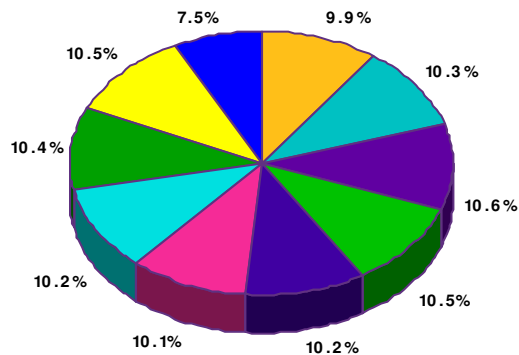
Multi-Dimensional Clustering (MDC)

- Physically clusters data in multiple (n) dimensions
 - Block contains all rows having same value of all dimensions
 - ORGANIZE BY clause in DDL
 - Block indexes: 1 entry/block
- Advantages:
 - Well-suited for data warehouses
 - Block indexes much smaller
 - Partition elimination
 - Improved performance of queries
 - Simpler Roll-in and Roll-out of data
 - Automatic clustering maintenance
 - REORG not necessary
 - Less overhead for logging and maintenance

Cell: unique dimension combination
Block: a contiguous set of data pages



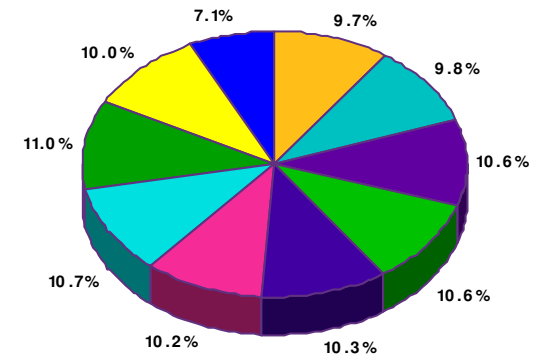
Sampling in SQL



```
SELECT year,
SUM(revenue) / total_revenue
FROM Sales
GROUP BY year
```

- 1999
- 1998
- 1997
- 1996
- 1995
- 1994
- 1993
- 1992
- 1991
- 1990

1% sampling (2 secs.)



```
SELECT year,
SUM(revenue) / total_revenue
FROM Sales
TABLESAMPLE SYSTEM (1)
GROUP BY year
```

