IBM Smart Analytics Optimizer – Not Your Father's Database!

The future runs on System z

Guy Lohman
Manager, Disruptive Information Management Architectures
IBM Almaden Research Center
lohman@almaden.ibm.com

IBM Smart Analytics Optimizer (ISAO) – Agenda

- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **Behind the Curtain – The Query Engine Technology**
- **ISAO vs. Column Stores**
- **Conclusions**

Why Optimize Smart Analytics?

- Today, performance of Business Intelligence (BI) queries is too <u>unpredictable</u>
  - When an analyst submits a query, s/he doesn't know whether to:
    - Wait for the response
    - Go out for coffee
    - Go out for dinner
    - Go home for the night!
  - Response time depends upon "performance layer" of indexes & materializations
  - Depends critically on predicting the workload
  - But BI is inherently ***ad hoc!***
- Goal of IBM Smart Analytics Optimizer:

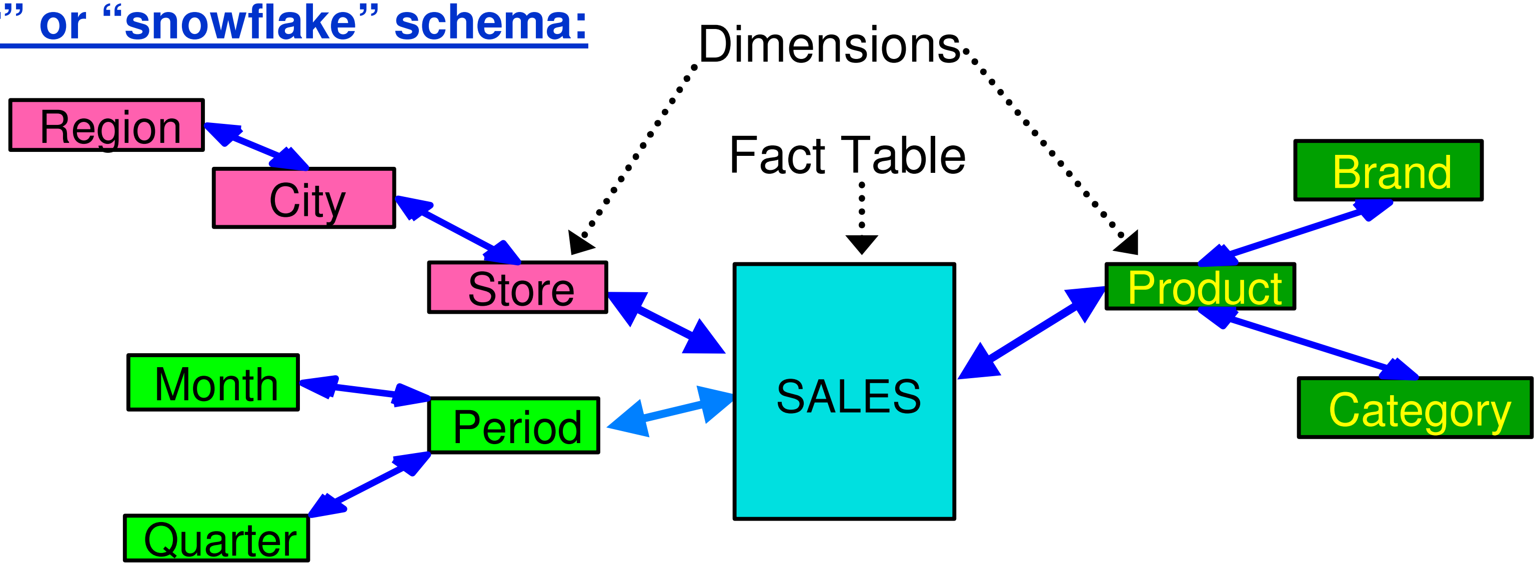
  Predictably Fast (i.e., <u>Interactive</u>) Ad Hoc Querying
  - **Any** query should run in about the same time
  - Permit an Analyst to interact with the data

What Is the IBM Smart Analytics Optimizer?

- IBM Smart Analytics Optimizer for z/OS (ISAO) **is:**
  - Network-attached accelerator to DB2 on z/OS
    - (Future: also DB2 for Linux, UNIX, and Windows and Informix IDS)
  - Exploits:
    - Large main memories
    - Commodity multi-core processors
    - Extreme compression
  - Speeds up typical Data Warehouse / Business Intelligence SQL queries by **10x to 100x**
  - Without requiring tuning of indexes, materialized views, etc.

Target Market: Business Intelligence (BI)

- **Characterized by:**
  - **"Star" or "snowflake" schema:**



  - **Complex, ad hoc queries that typically**
    - Look for trends, exceptions to make actionable business decisions
    - Touch large subset of the database (unlike OLTP)
    - Involve aggregation functions (e.g., COUNT, SUM, AVG,…)
    - **The "Sweet Spot" for IBM Smart Analytics Optimizer!**

What IBM Smart Analytics Optimizer is Designed For

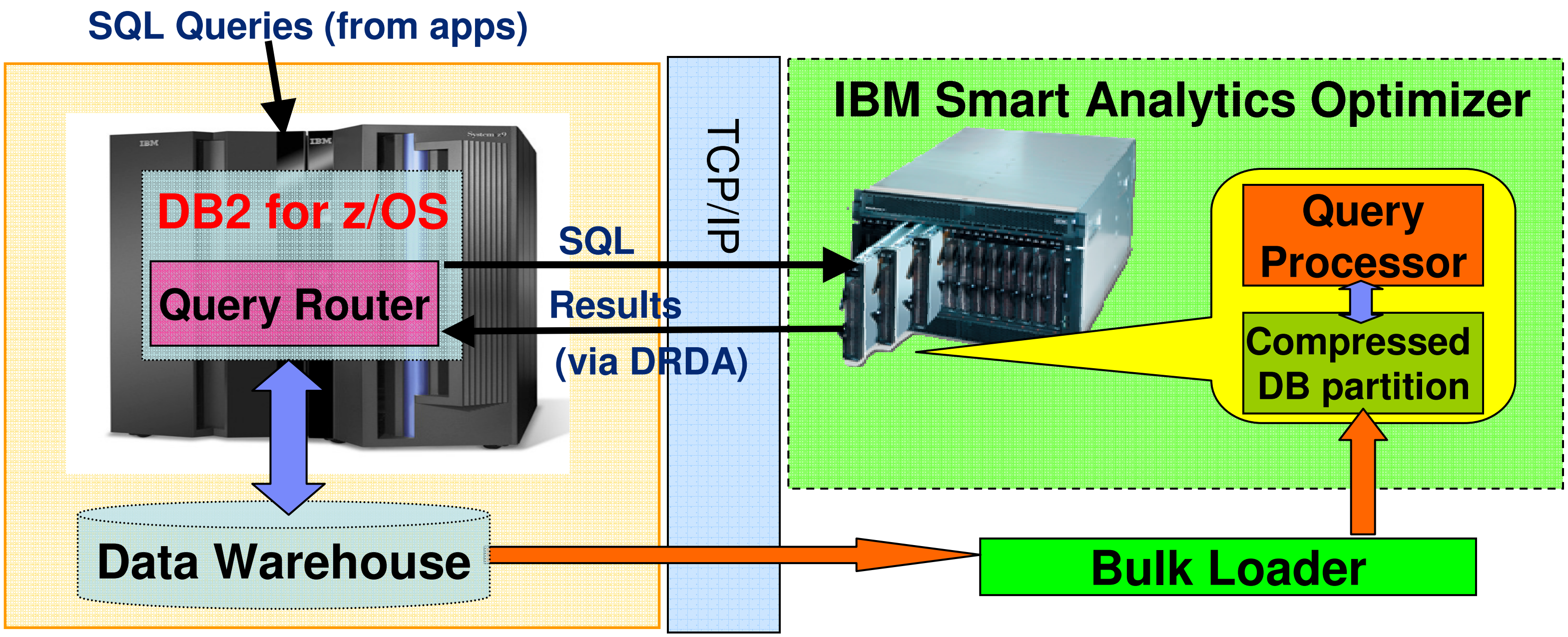
- OLAP-style SQL queries:
  - Relational star schema (large **fact table** joined to multiple **dimensions**)
  - Large subset of data warehouse accessed, reduced significantly by...
  - **Aggregations** (SUM, AVG, COUNT) and optional **grouping** (GROUP BY)
  - **Looking for trends or exceptions**
- EXAMPLE SQL:

```
SELECT PRODUCT_DEPARTMENT, REGION, SUM(REVENUE)
FROM FACT_SALES F
      INNER JOIN DIM_PRODUCT P ON F.FKP = P.PK
      INNER JOIN DIM_REGION R ON F.FKR = R.PK
      LEFT OUTER JOIN DIM_TIME T ON F.FKT = T.PK
WHERE T.YEAR = 2007
GROUP BY PRODUCT_DEPARTMENT, REGION
```

IBM Smart Analytics Optimizer (ISAO) – Agenda

- **Why and What <u>is</u> the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **Behind the Curtain – The Query Engine Technology**
- **ISAO vs. Column Stores**
- **Conclusions**

IBM Smart Analytics Optimizer Configuration



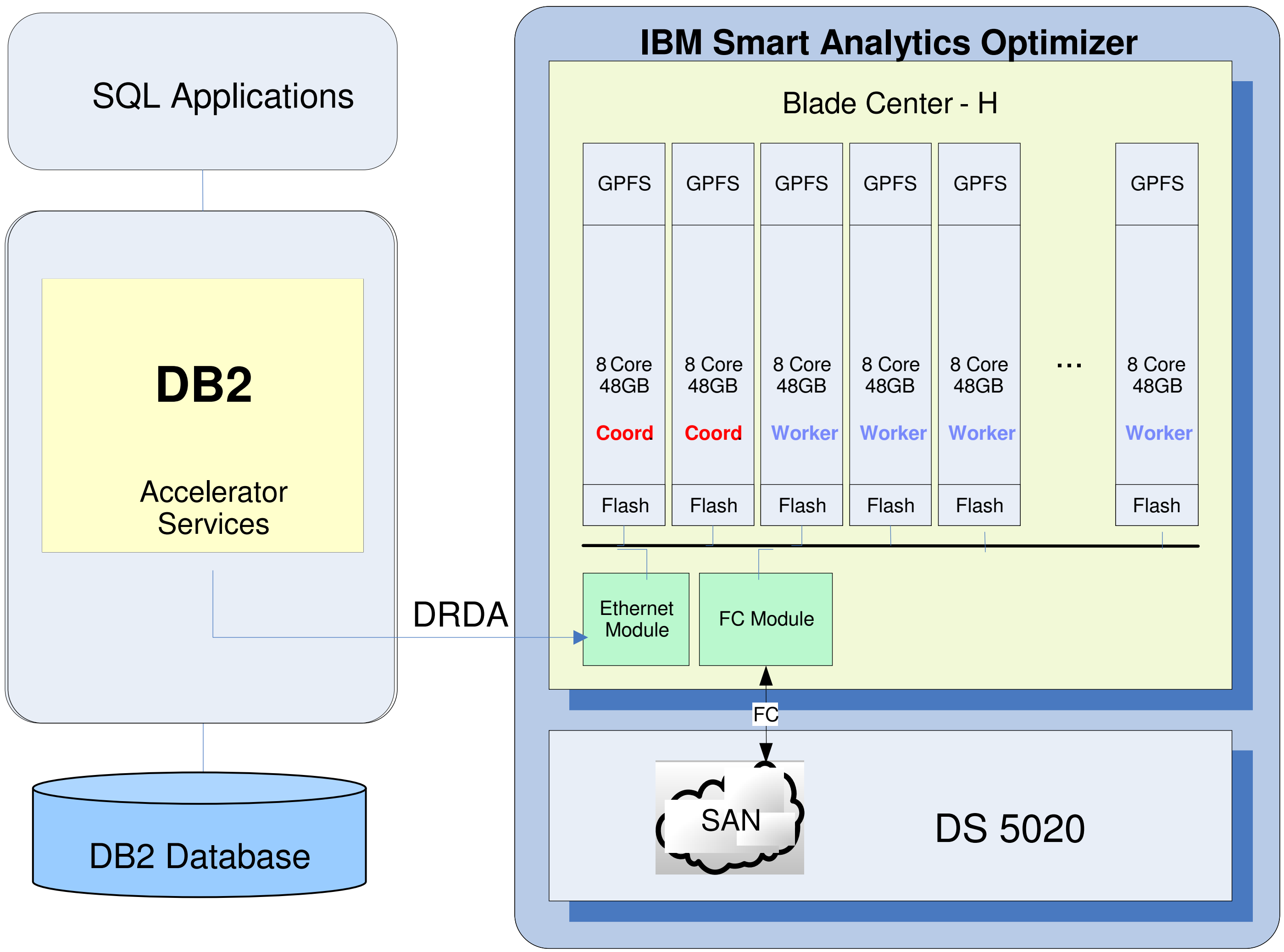
DB2 for z/OS:

- Routes SQL queries to accelerator
- User need not change SQL or apps.
- Can always run query in DB2, e.g., if
  - **too complex SQL, or**
  - **too short an est. execution time**

IBM Smart Analytics Optimizer:

- Multiple blades in blade center
- Connects to DB2 via TCP/IP & DRDA
- Analyzes, compresses, and loads
  - Copy of (portion of) warehouse
  - Partitioned among nodes
- Processes routed SQL query and returns answer to DB2

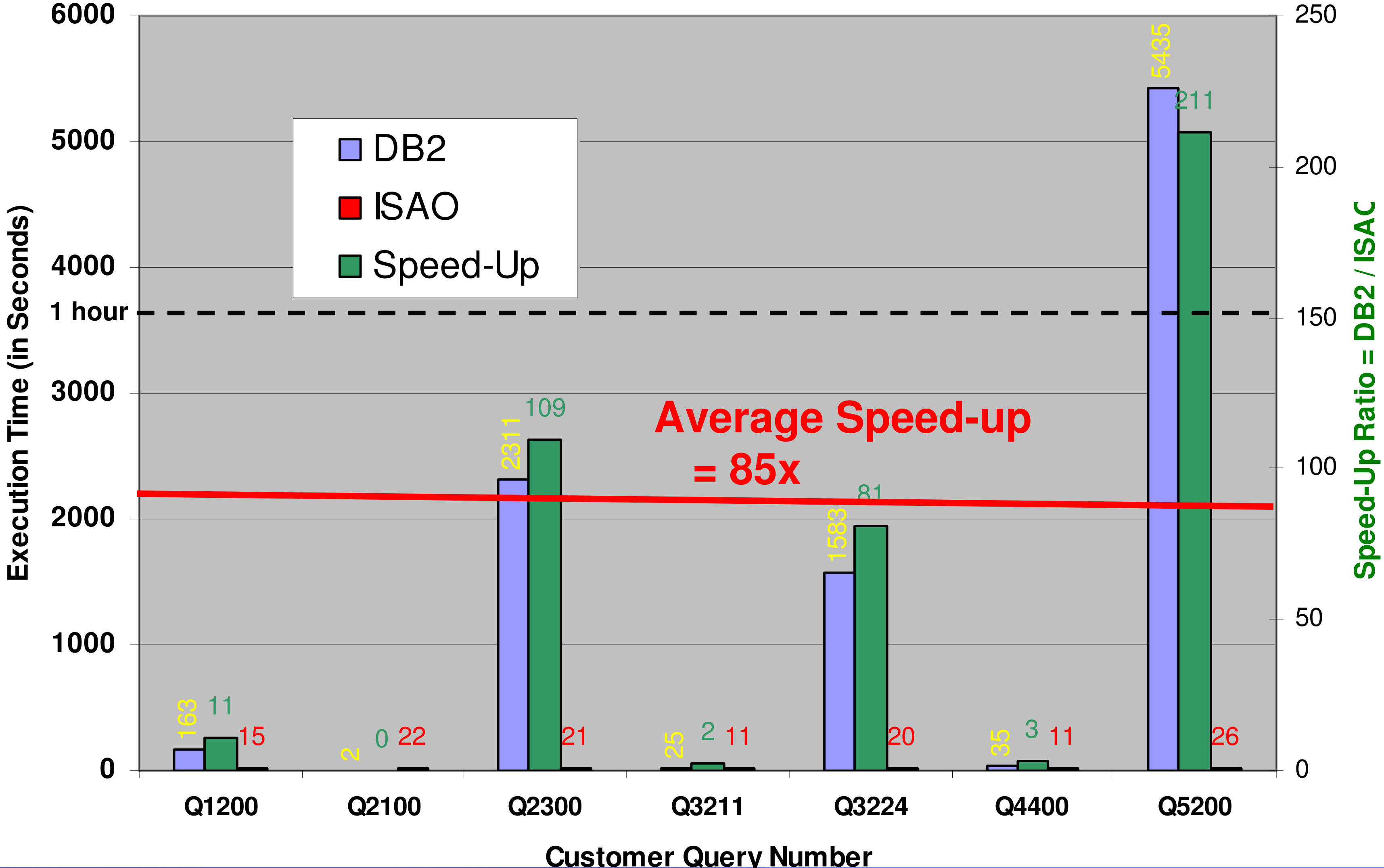
IBM Smart Analytics Optimizer Architecture



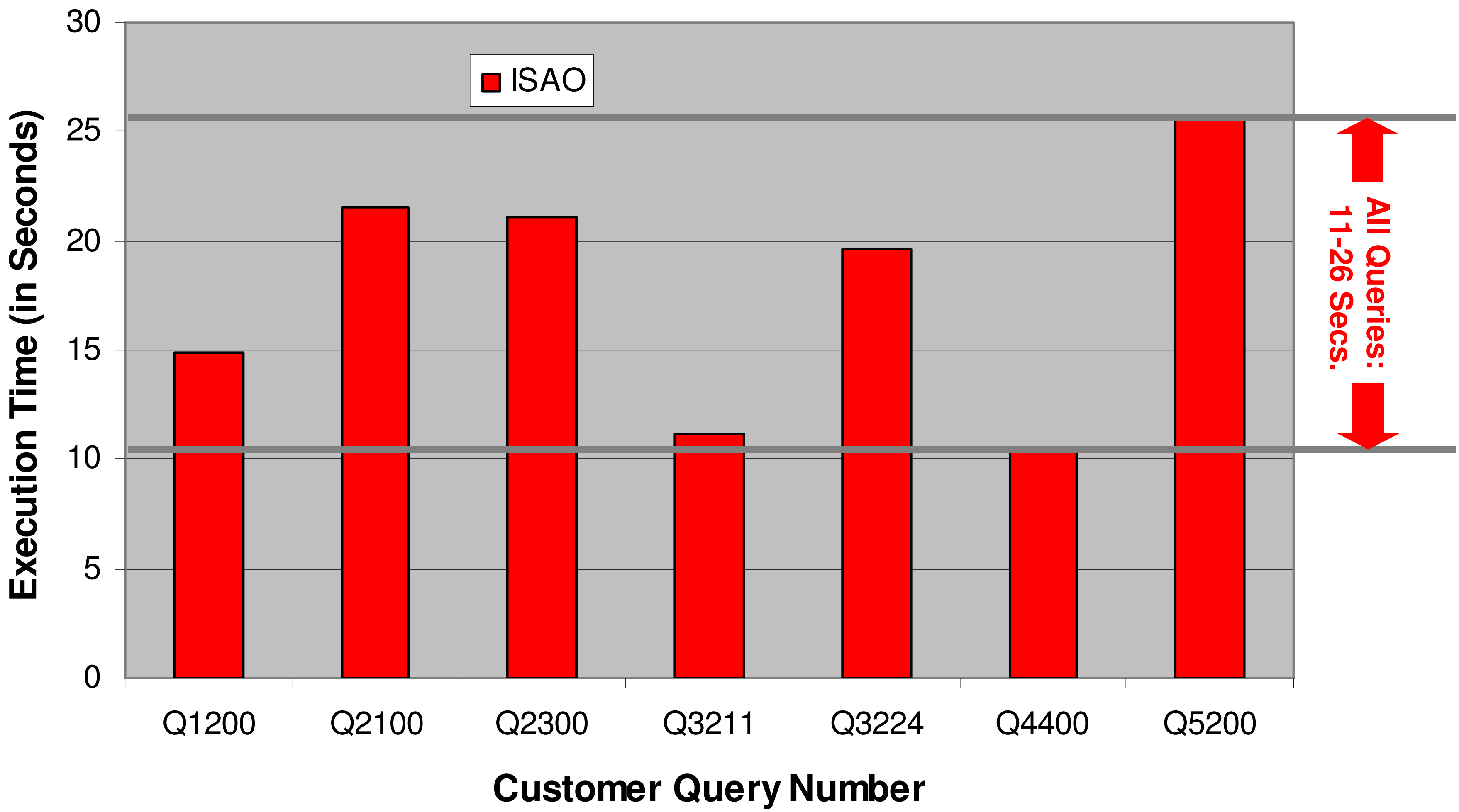
IBM Smart Analytics Optimizer (ISAO) – Agenda

- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **Behind the Curtain – The Query Engine Technology**
- **ISAO vs. Column Stores**
- **Conclusions**

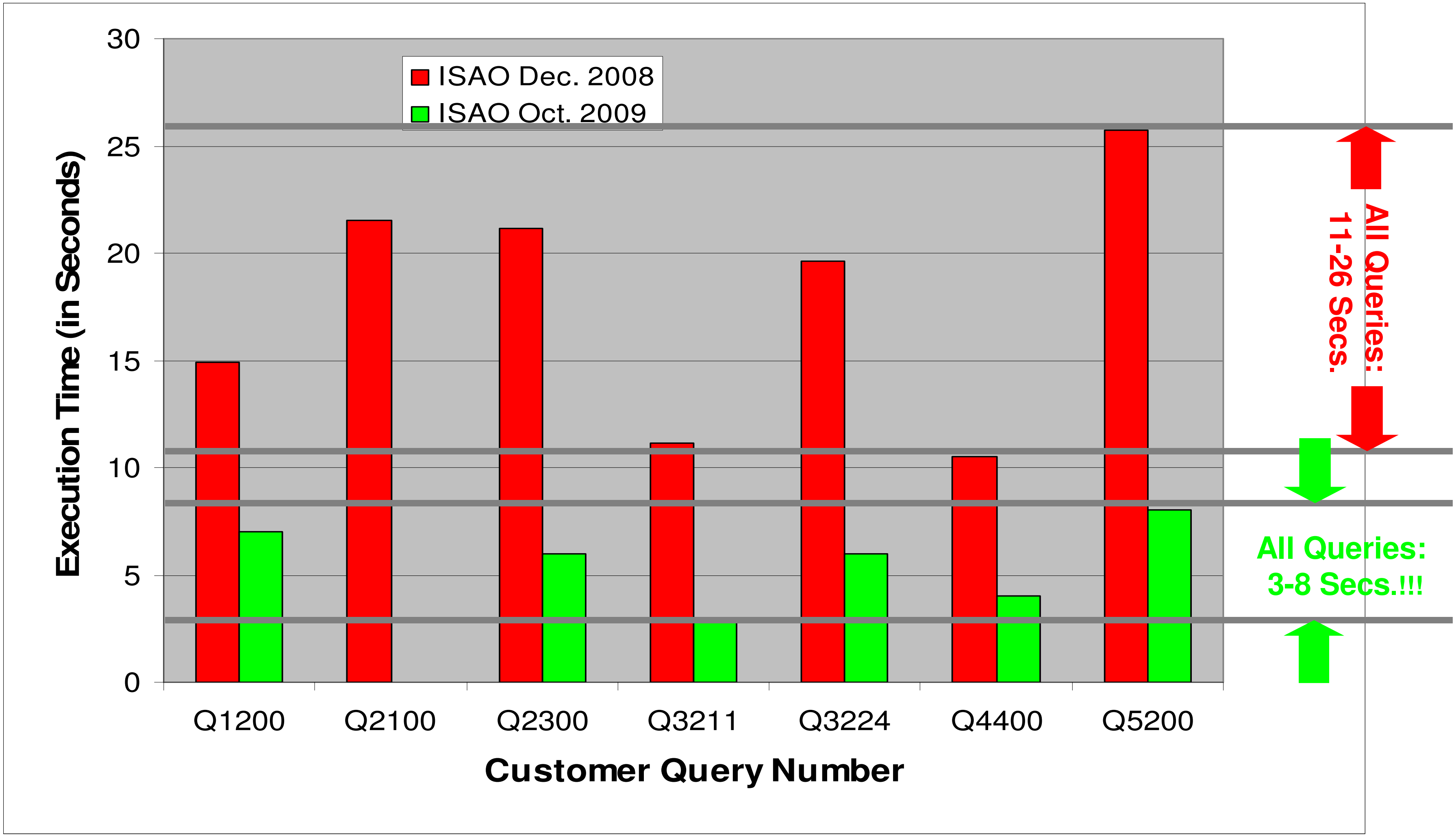
ISAO Accelerates Most the Longest-Running DB2 Queries



ISAO Query Execution times (magnified)



ISAO Query Execution times (magnified)



IBM Smart Analytics Optimizer (ISAO) – Agenda

- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **Behind the Curtain – The Query Engine Technology**
- **ISAO vs. Column Stores**
- **Conclusions**

Getting Started: Loading Data into ISAO

A. DBA Defines Mart (Data to Accelerate)

- A ***mart*** is a logical collection of tables which are related to each other.
  - For example, all tables of a single star schema would belong to the same ***mart***.
- The administrator uses a rich client interface in Data Studio to define the tables that belong to a ***mart***, together with information about their relationships.

B. DB2 Automatically Copies *Mart*

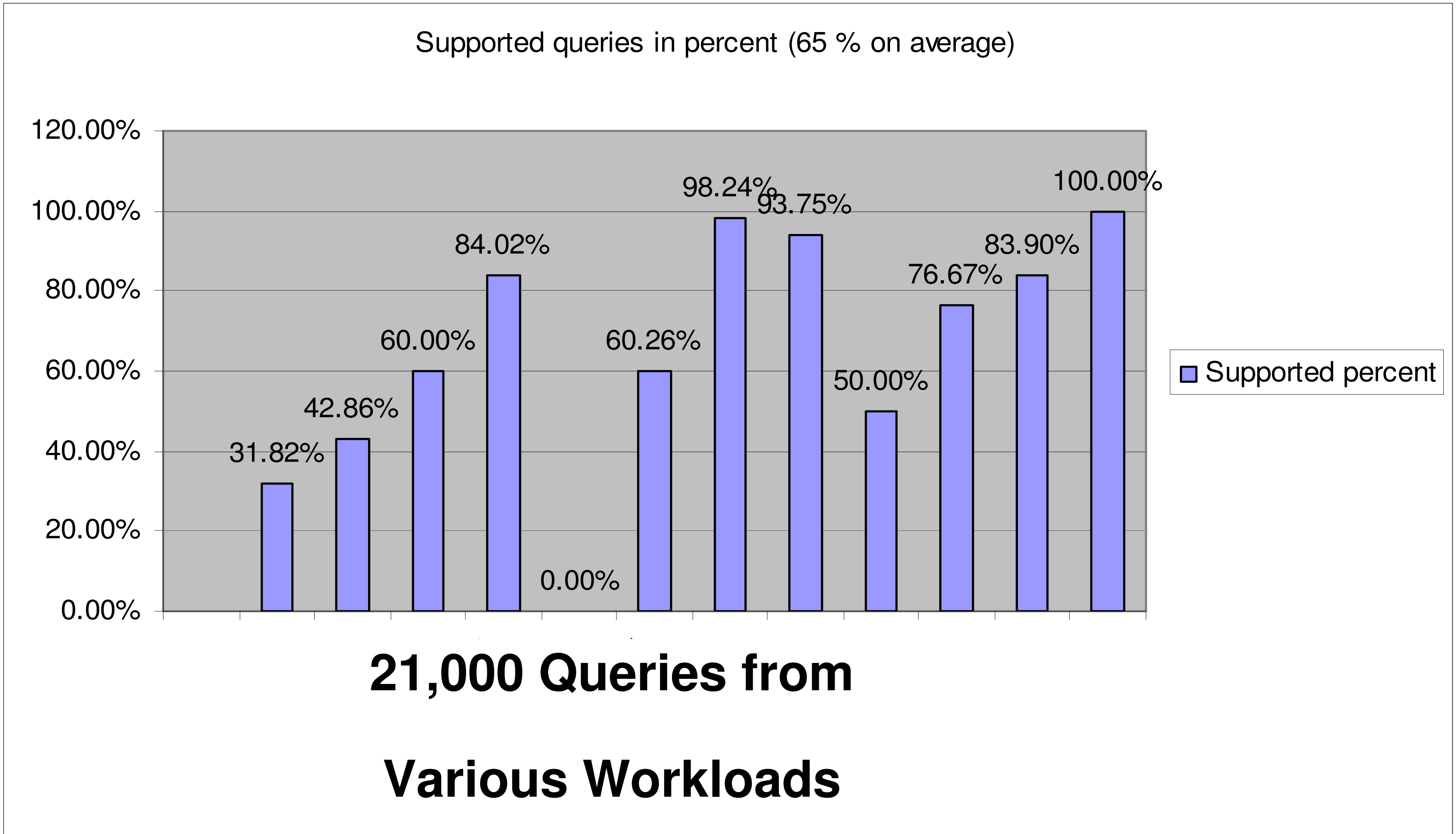
- DB2 creates definitions for these ***marts*** in its own catalog.
- The ***mart'***s data is read from the DB2 tables and transferred to ISAO.

C. ISAO Automatically Transforms & Loads *Mart* Data **into a highly compressed, scan-optimized format that is kept locally (in memory) on ISAO**

Supported Query Types

- IBM Smart Analytics Optimizer can process queries that contain:
  - Most built-in SQL functions
  - Most SQL data types
  - Only one ***query block*** (SELECT... FROM... WHERE...) at a time
    - DB2 routes to ISAO each query block of a query
    - Queries including subquery predicates cannot be routed
  - Only equi-joins (ON FACT.FK = DIM.PK)
    - Referencing columns with compatible types
    - SQL allows conversion without explicit cast
    - Expressions such as A.YEAR = YEAR(B.TIMESTAMP) not supported
  - Only
    - Inner joins (explicit INNER JOIN or implicit join syntax), or
    - <Fact table> LEFT OUTER JOIN <Dimension table>
- IBM Smart Analytics Optimizer canNOT process queries that contain:
  - Large objects (LOBs), ROWID, binary, or XML data types
  - > 225 tables
  - > 750 columns in an Accelerator Query Table (AQT)
  - Certain functions not supported in V1 of ISAO:
    - Mathematical functions such as SIN, COS, TAN, EXP, and CORRELATION
    - User-Defined Functions
    - Advanced string functions such as LOCATE, LEFT, OVERLAY, and POSITION
    - Advanced OLAP functions Isuch as RANK, DENSE, ROW NUMBER, ROLLUP, and CUBE
  - Self-joins or cycles in the join graph

Supported Queries in Percent per Workload



IBM Smart Analytics Optimizer (ISAO) – Agenda

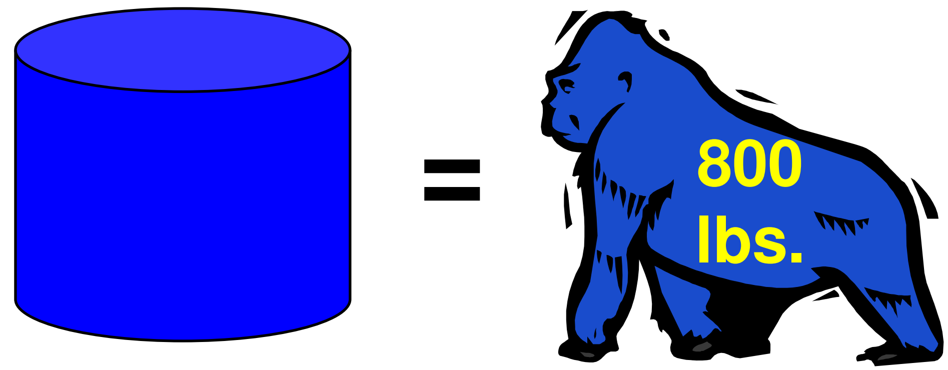
- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **The Query Engine Technology**
- **Behind the Curtain – The Query Engine Technology**
- **ISAO vs. Column Stores**
- **Conclusions**

What's the Big Deal? What's so Disruptive?

- ISAO rides the wave of hardware technology trends:
  - Multi-core processors
  - Large main memories
  - Fast interconnects
  - Increasing latency gap between DRAM and disk
- ISAO disrupts at least 3 major tenets

  that have been held sacrosanct for over 4 decades!

Disruption 1 of 3



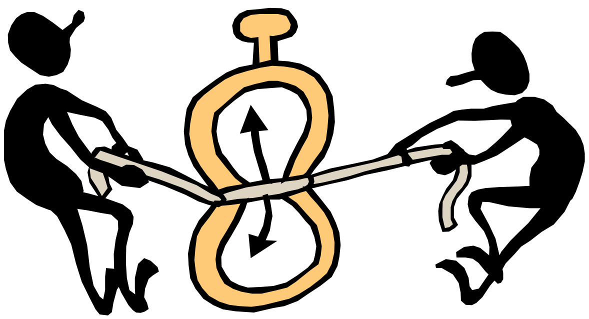
- Tenet #1: **Data warehouses are too big for memory**
- Consequence of Tenet #1: **Disk I/O concerns dominate DBMS…**
  - Costs
  - Performance
  - Administration efforts
- Disruption #1: **Huge, cheap main memories (RAM) and flash memories**
- Consequences of Disruption #1:
  - Portions of warehouse can fit, if partitioned among multiple machines
  - Compression helps!
  - New bottleneck is memory bandwidth (RAM ←→ L2 cache) and CPU
  - No preferred access path

Disruption 2 of 3

- Tenet #2: **Need many Indexes & MQTs for scalable OLAP performance**
- Consequences of Tenet #2:
  - Need an optimizer to choose among access paths
  - Need a ★★★★ wizard to design "performance layer" (expensive!)
  - Must anticipate queries
  - Large time to update performance layer when new data added
- Disruption #2: **Massive parallelism achieves DB scan in seconds!**
  - Arbitrarily partition database among nodes (32–64 GB RAM / node)
  - Exploit multi-core architectures within nodes (1 user or DB cell / core)
- Consequences of Disruption #2:
  - Only need to define 1 AQT in DB2 to satisfy many queries on the accelerator
  - Always scan tables!!
  - Accelerator automatically does equivalent of partition elimination
    - If literal is not in dictionary of that partition
  - Accelerator itself doesn't need
    - Performance layer (indexes or materialized views)!
    - Optimizer!
  - **Simpler! (no need for 4-star wizard)**
  - **Lower TCO!**
  - Consistent response times

Disruption 3 of 3

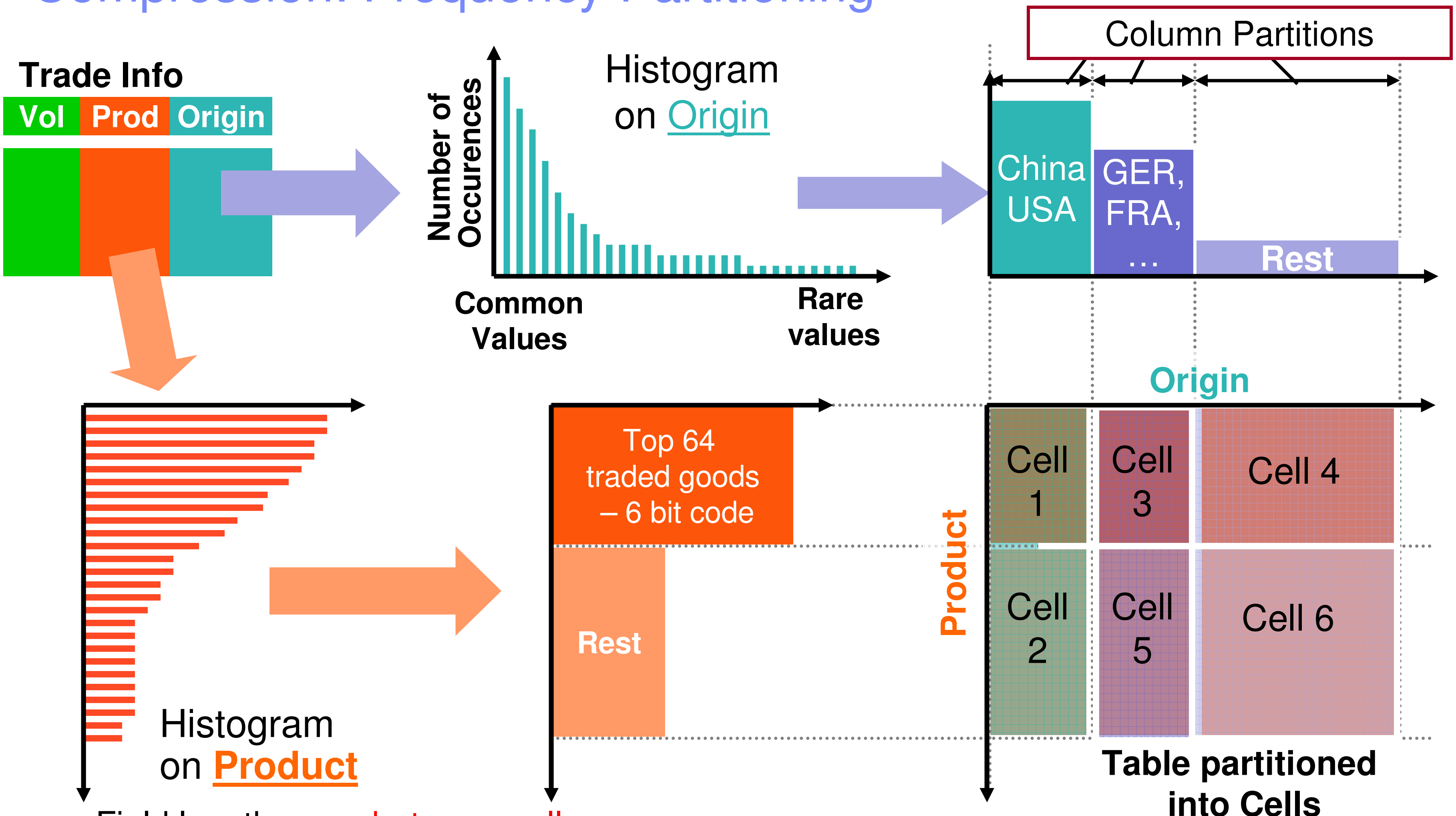
- Tenet #3: **Main-memory DBMSs are the same as a big buffer pool**
- Disruption #3: **Clever engineering can save lots more!**
- Examples of Disruption #3:
  - Specialized order-preserving & fixed-length compression within partitions permits:
    - Faster access
    - Performing most operations **on encoded values (saves decoding cost)**
    - **Simultaneous application of predicate conjuncts (1 compare!)**
  - Cache-conscious algorithms make max. use of L2 cache and large registers
  - Exploit multi-core processors
  - Hash-based grouping avoids sorting



IBM Smart Analytics Optimizer (ISAO) – Agenda

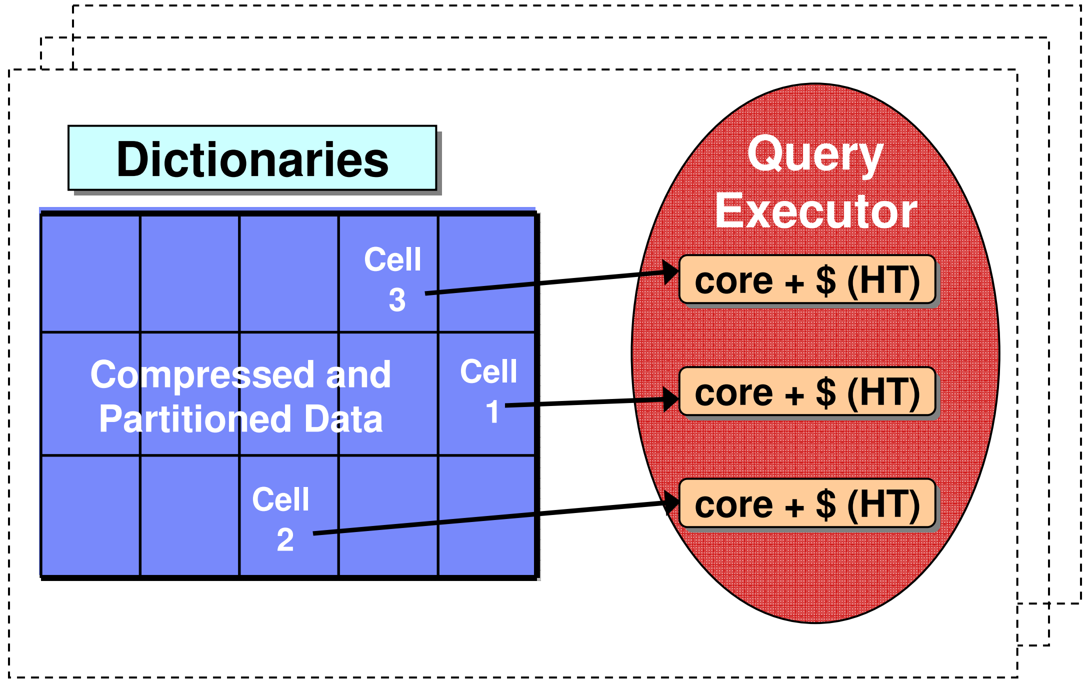
- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **Behind the Curtain – The Query Engine Technology**
- **ISAO vs. Column Stores**
- **Conclusions**

Compression: Frequency Partitioning



- Field lengths vary between cells
  - Higher Frequencies ➔ Shorter Codes (Approximate Huffman)
- Field lengths fixed within cells

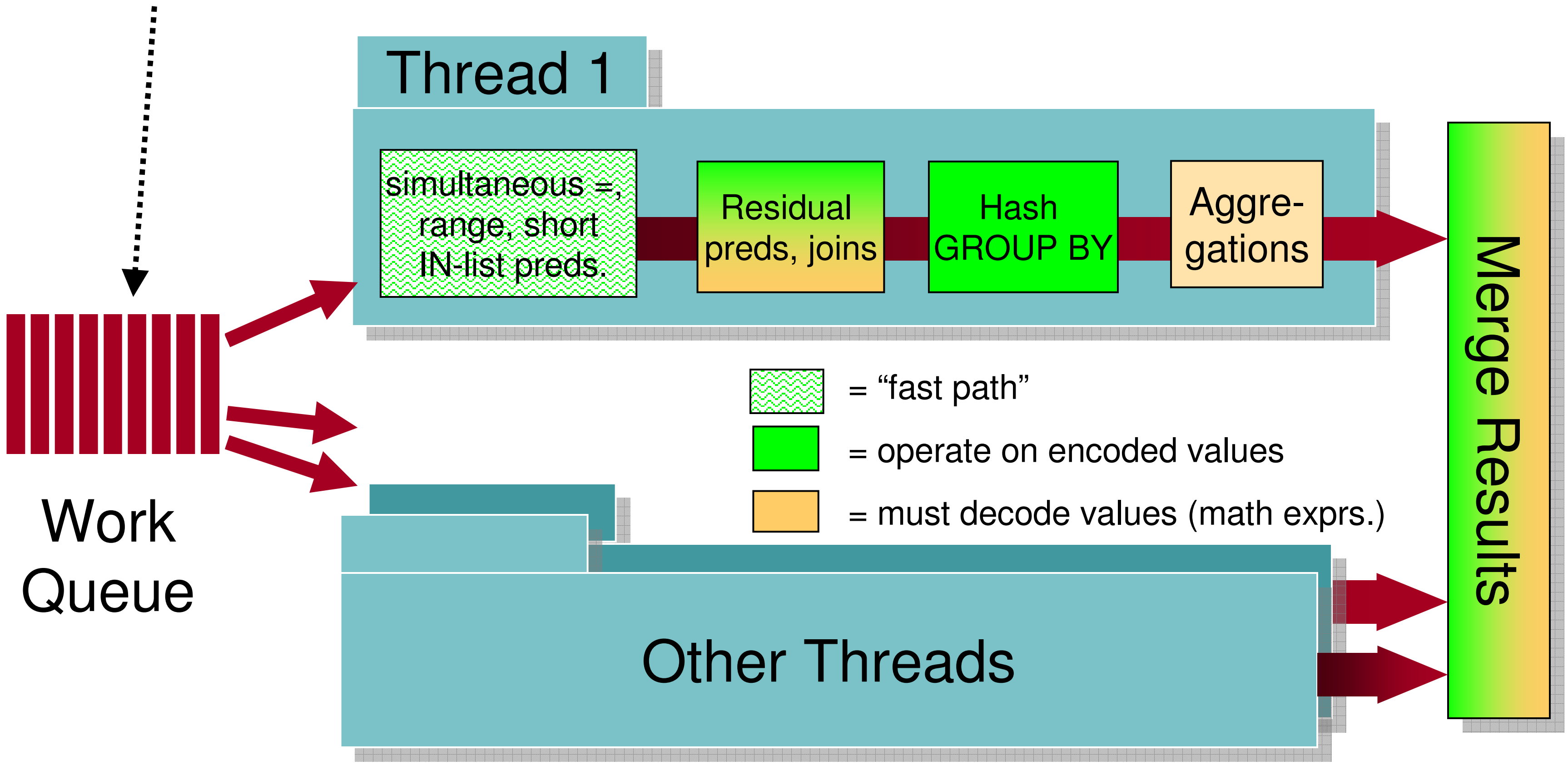
Query Processing



- Cell is also the unit of processing, each cell…
  - Assigned to one core
  - Has its own hash table in cache (so no shared object that needs latching!)
- Main operator: SCAN over compressed, main-memory table
  - Do selections, GROUP BY, and aggregation as part of this SCAN
  - Only need de-compress for aggregation
- Response time ∝ (database size) / (# cores x # nodes)
  - Embarrassing Parallelism – little data exchange across nodes

Fine-Grained Data Parallelism

Work Unit = Block of Frequency-Partitioned Cell



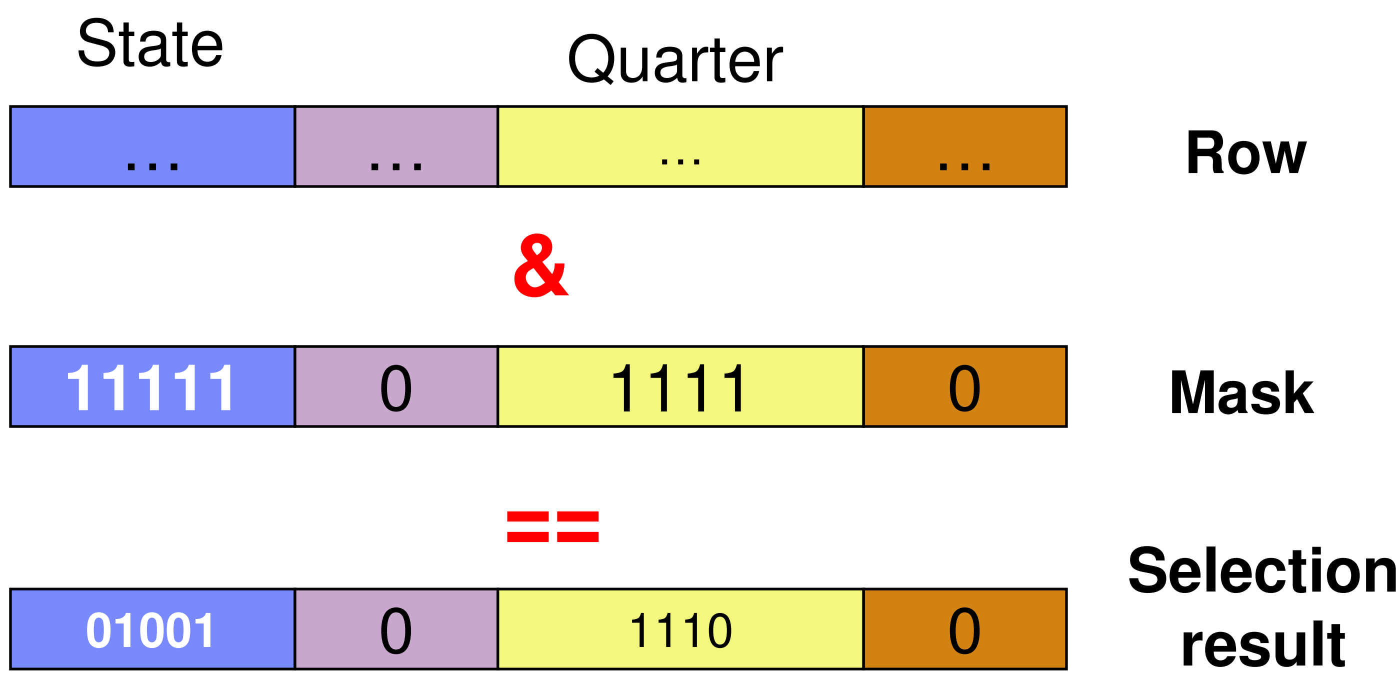
Simultaneous Evaluation of Equality Predicates

- CPU operates on 128-bit units
  - Lots of fields fit in 128 bits
- These fields are at fixed offsets
- Apply predicates to all columns simultaneously!

State==‘CA’ && Quarter == ‘Q4’

Translate value query to Code query

State==01001 && Quarter==1110



Fast Hash-based Grouping

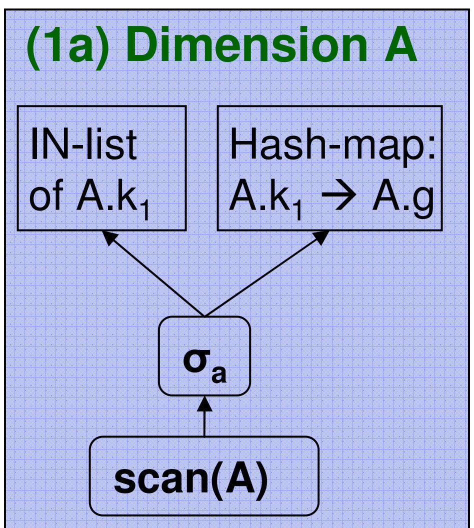
- **Encoding makes grouping simple!**
  - Coded values assigned densely (by construction)
  - Hence, **in principle**, grouping is simple: aggTable[group] += aggValue
- **Challenges:**
  - Fitting hash table in L2 cache
  - Avoiding all branches in hash table lookup
- **IBM Smart Analytics Optimizer adaptively uses one of 2 techniques, depending on # of distinct groups**
  1. Use dictionary code as a perfect hash (i.e. collision-free), OR
     - aggTable[groupCode] += aggValue
     - No branches, no hash function computation
     - Works great if groupCode is dense
       – i.e., single column, or multiple column with little correlation
  2. Use usual linear probing
     - Involves branches, random access, …

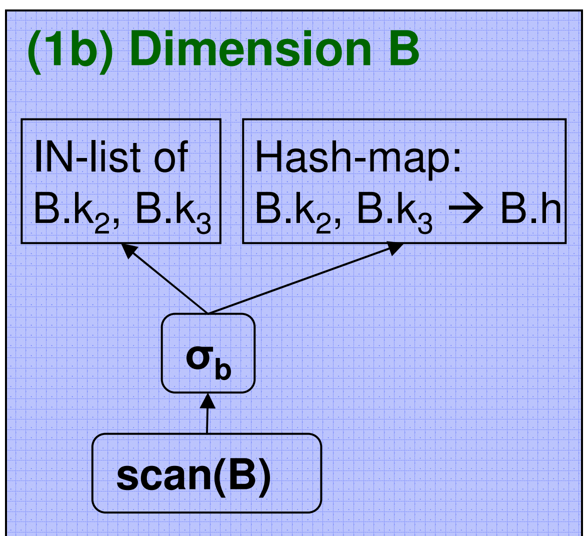
Joins

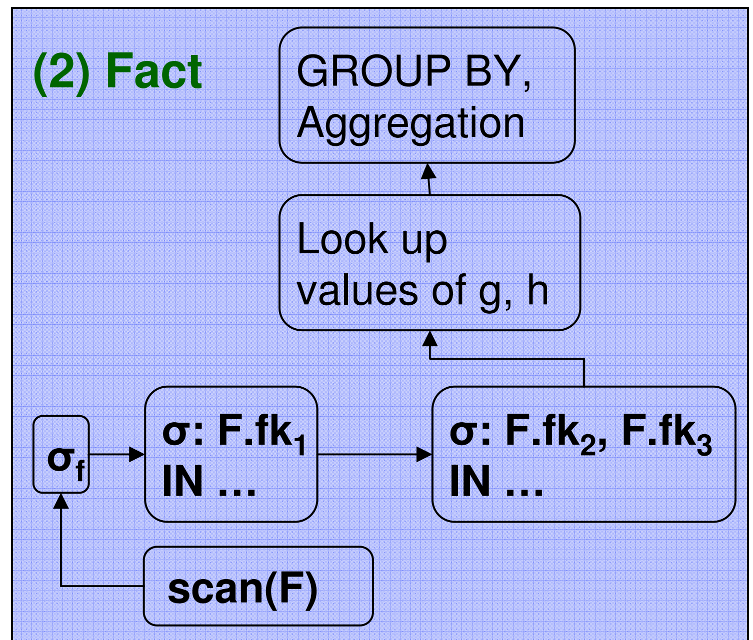
- **Basic idea: Re-write Join as <u>multiple scans</u>:**

1. Over each **dimension**, to form:
   - A list of qualifying ***primary keys (PKs),*** decoded
   - A ***hash-map*** of primary key ➔ ***auxiliary columns*** (those used later in query for GROUP BY, etc.)
2. Over **fact** table**:**
   - First convert PKs to ***foreign keys (FKs)*** in fact table column
   - Apply as (very big) IN-list predicates (a semi-join), one per dimension
   - Look up into hash-maps to pick up other columns
   - Complete Grouping and Aggregation

- **Snowflakes: apply repeatedly, outside in**







What About Updates?

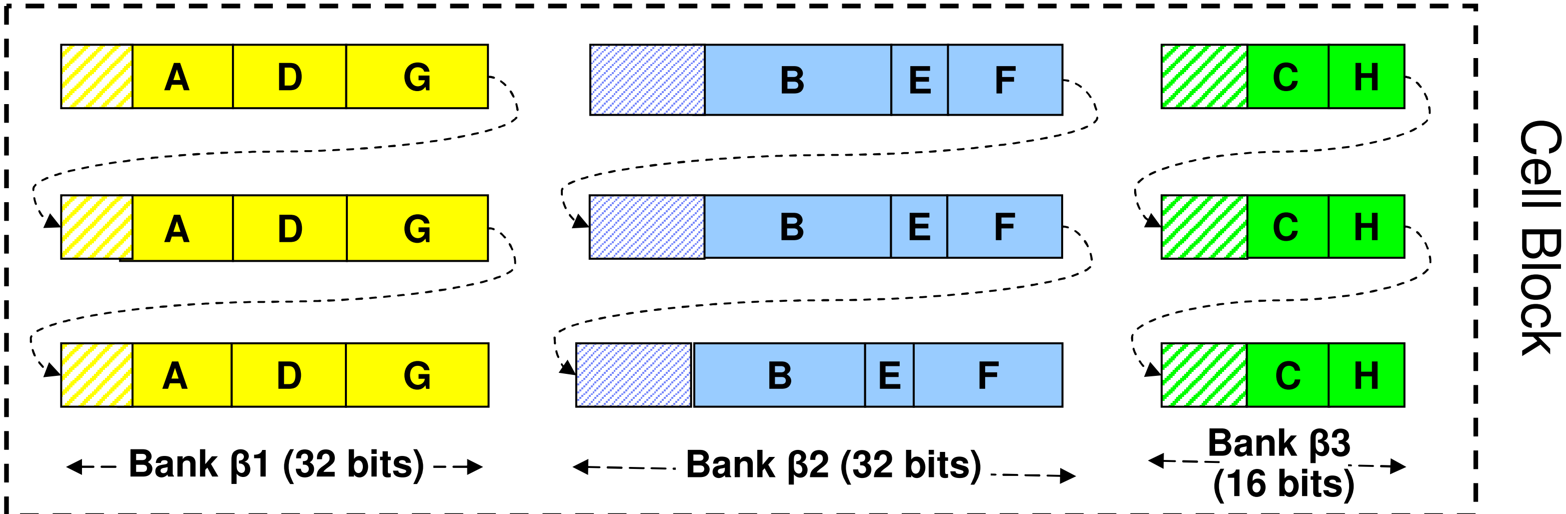
- **ISAO uses** snapshot semantics **(batch updates), common in BI**
- **System maintains a** currentEpoch **number (monotone increasing)**
  - Think of it as a batch or version number
  - Prevents seeing incomplete updates, without needing locking
  - Bumped (N++) atomically after each batch of inserts & deletes completes
- **Tables have two new columns**
  - **startEpoch** – epoch in which that row inserted
  - **endEpoch** – epoch in which that row deleted (initially Infinity)
- **Queries are automatically appended with two predicates:**
  - startEpoch < currentEpoch AND
  - endEpoch > currentEpoch
- **Encoding of updated values**
  - If value is in dictionary, use that encoding
  - Otherwise, create a new one in a special cell, called the "catch-all" cell
    - Maybe more bits than optimal, but can handle new values
    - Assigned in order values inserted (<u>not</u> order-preserving encoding)

IBM Smart Analytics Optimizer (ISAO) – Agenda

- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- **ISAO Architecture**
- **It's All About Performance!**
- **From the User's Perspective**
- **What's the Big Deal?**
- **Behind the Curtain – The Query Engine Technology**
- ISAO vs. Column Stores
- **Conclusions**

Banks and Tuplets

- A *bank* is a vertical partition of a table, containing a subset of its columns
  - Assignment of columns to banks is cell-specific, since column's length
    - Varies from cell to cell, but
    - Is fixed within a cell
  - Banks contain
    - Concatenations of the fixed-length column codes
    - Padded to the nearest word length (8 / 16 / 32 / 64 bits).
    - We call these word-sized units ***tuplets*.**
- ISAO's bank-major layouts are a hybrid of row-major and column-major



IBM Smart Analytics Optimizer vs. a Column Store

Aspect	Column Store	IBM Smart Analytics Optimizer
Compression	**Every column padded to word boundary** **➔ more padding/column** **➔ worse compression**	**Multiple columns / word** **➔ less padding overhead**
Query Processing	▪ **Like having an index on every column** ▪ **To answer query:** ▪ Determine list(s) of matching records ▪ Intersect these lists on RID	▪ **Can skip blocks based upon predicates** ▪ **To answer query:** ▪ Do table scan
Updating	▪ **Insert requires:** ▪ Separate updates to every column **➔ Multiple random I/Os, 1/column**	▪ **Insert requires:** ▪ Single update to each bank, 1 / bank **➔ One I/O to one cell block**
Evaluation Matches Hardware?	Evaluation doesn't match w/ Hardware: ▪ **Index navigation involves random accesses** ▪ **Index navigation involves branches** ▪ **Predicate evaluation has to be done serially**	Evaluation matches with Hardware ▪ **Scan does sequential memory access** ▪ **Almost no branches** ▪ **Simultaneous predicate evaluation** ▪ **SIMD predicate evaluation**

Refereed Publications in Top 3 Professional Conferences

- VLDB 2008: ***"Main-Memory Scan Sharing for Multi-core CPUs",*** **Lin Qiao, Vijayshankar Raman, Frederick Reiss, Peter Haas, Guy Lohman**
- VLDB 2008: ***"Row-Wise Parallel Predicate Evaluation",*** **Ryan Johnson, Vijayshankar Raman, Richard Sidle, Garret Swart**
- VLDB 2006**:** ***"How to wring a table Dry: Entropy Compression of Relations and Querying Compressed Relations",*** **Vijayshankar Raman, Garret Swart**
- SIGMOD 2007**:** ***"How to barter bits for chronons: compression and bandwidth trade offs for database scans",*** **Allison L. Holloway, Vijayshankar Raman, Garret Swart, David J. DeWitt**
- ICDE 2008**:** ***"Constant-time Query Processing",*** **Vijayshankar Raman, Garret Swart, Lin Qiao, Frederick Reiss, Vijay Dialani, Donald Kossmann, Inderpal Narang, Richard Sidle**

VLDB **= International Conference on Very Large Data Bases**
SIGMOD **= ACM SIGMOD International Conference on Management of Data**
ICDE **= IEEE International Conference on Data Engineering**

Summary – Not Your Father's Database!

- **Radical changes are happening in hardware**
  - Large, cheap memories
  - Multi-core processors promise cheap, massive CPU parallelism
- **IBM Smart Analytics Optimizer exploits these trends:**
  - Special-purpose accelerator (BI only, snapshot semantics, no transactions)
  - Main-memory DBMS
  - Massive parallelism of commodity multi-core hardware (blade center format)
  - Query processing on compressed values!
  - Cache-conscious algorithms
- **IBM Smart Analytics Optimizer speeds up your problem queries the most!**
- **IBM Smart Analytics Optimizer is an appliance that is transparent to the user**
  - Minimal set-up
  - Applications need not change
  - Tuning not needed!
  - Lower TCO

Questions?

धन्यवाद
Hindi

多謝
Traditional Chinese

ขอบคุณ
Thai

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

Thank You
English

Obrigado
Brazilian Portuguese

Danke
German

Grazie
Italian

多谢
Simplified Chinese

Merci
French

நன்றி
Tamil

ありがとうございました
Japanese

감사합니다
Korean

BACKUP SLIDES

1B. DB2 Automatically Defines Mart as Accelerator Query Tables (AQTs)

- ***AQTs look like MQT definitions***

```
CREATE TABLE DSN8910.MYAQT AS (
      SELECT …
      FROM …
      WHERE …
 )
 DATA INITIALLY DEFERRED
 REFRESH DEFERRED
 MAINTAINED BY USER
 ENABLE QUERY OPTIMIZATION
 IN ACCELERATOR DWASYS1
```

- DB2 uses AQTs to decide which queries to route to ISAO
- AQT content is only available on ISAO

2. SQL Routing: DB2 Automatically Routes Queries to ISAO

- DB2 Optimizer uses AQTs (like MQTs) to decide which queries can be routed to ISAO, and which cannot.
- **Automatic -- User <u>need not</u>:**
  - Change SQL in the application
  - Be aware of whether DB2 routes an individual query

<u>Query:</u>

```
SELECT SUM(REV)
FROM SALES LEFT OUTER JOIN DIM1
    ON...
    INNER JOIN DIM2 ON...
    LEFT OUTER JOIN DIM3 ON...
WHERE DIM1.Brand = ‚Levis‘
    AND DIM3.Year = 2009
GROUP BY P_Group, Quarter, Region
```

Query-time routing

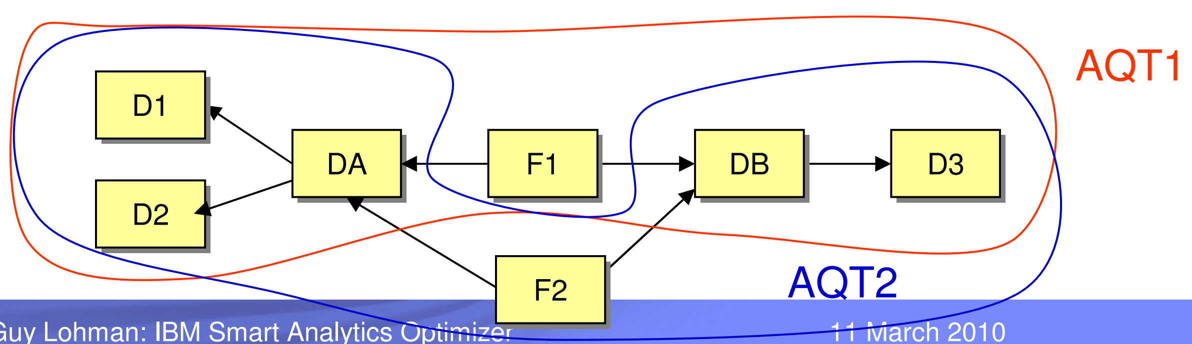
<u>Routed Query:</u>

```
SELECT  SUM(REV)
FROM MYAQT A.
WHERE A.Brand = ‚Levis‘
    AND  A.Year = 2009.
GROUP  BY A.P_Group, A.Quarter, A.Region
```

Supported Schemas

- **A *mart* consists of a set of tables, together with their referential constraints**
  - Fact tables are considered to be the tables having the highest join depth
- **A mart may contain multiple fact tables**
- **However, ISAO cannot handle queries that cross mart boundaries**
- **Load-time (de-normalization) join requires:**
  - Dimension join cols **must have unique constraint** / **primary key**
    - ➔ **Dimension : Fact cardinality is 1 : 0..n**
    - ➔ **n:m joins are supported as run-time joins only (cannot be pre-joined)**

  WHY: require loss-less join in case a dimension table is omitted from the query



Examples of Queries with Multiple Query Blocks

✓Derived table **(nested table expression)**

```
SELECT * FROM
(SELECT  C1+C2 FROM TA) TX
```

✓**Derived table (Common Table Expression (CTE))**

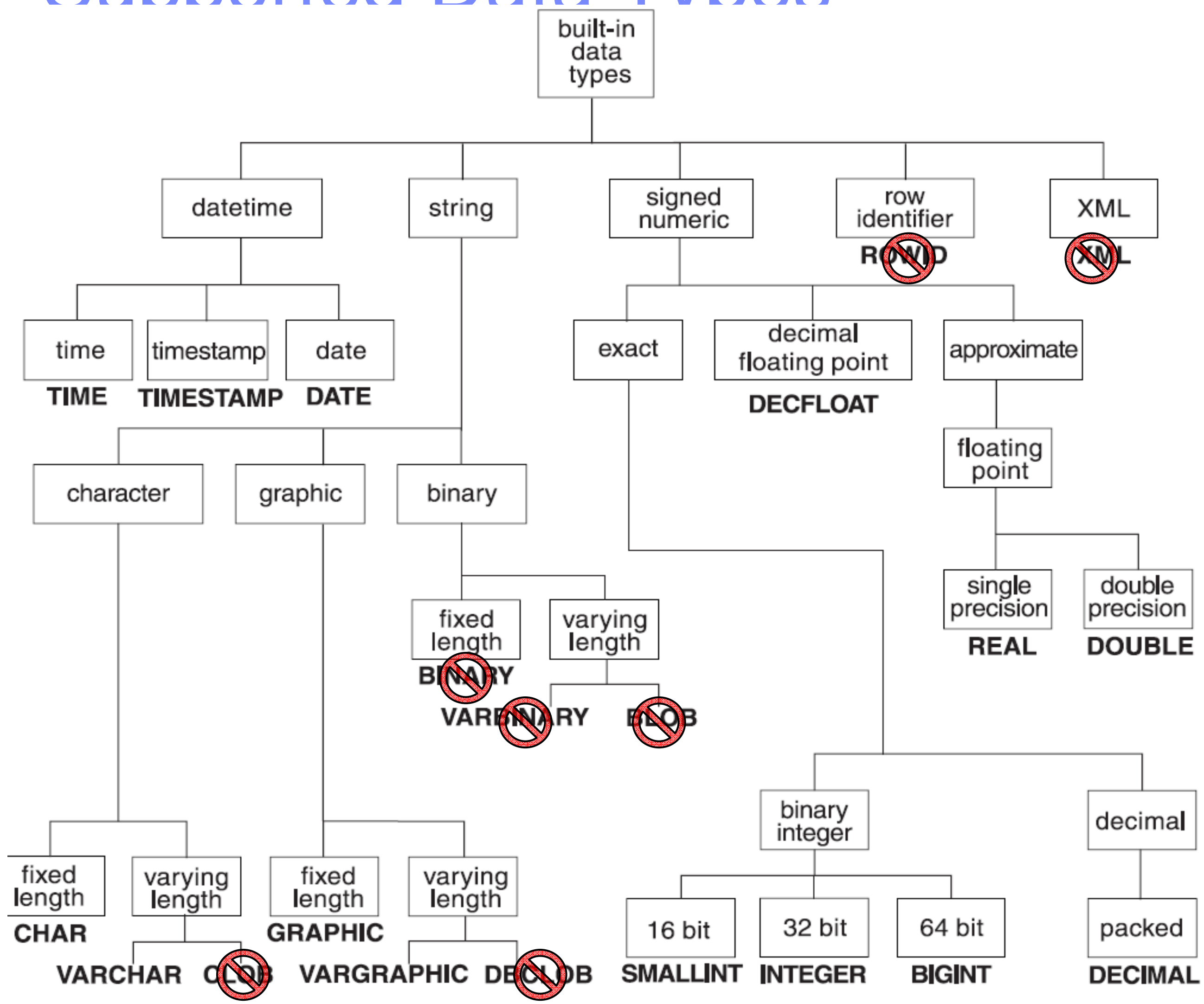
```
WITH DTOTAL (deptno, totalpay) AS
        (SELECT deptno, sum(salary+bonus)
        FROM DSN8810.EMP GROUP BY deptno)
SELECT deptno
FROM DTOTAL
WHERE totalpay = (SELECT max(totalpay) FROM DTOTAL);
```

✓**EXISTS or IN predicate with Subquery**

```
SELECT  ... FROM    ...    WHERE   ...
 AND    (A11.STORE_NUMBER IN
                (SELECT  C21.STORE_NUMBER
                 FROM    USRT004.VL_CSG_STR   C21
                 WHERE   C21.CSG_NUMBER IN (4643) ))
```

✓**UNION, INTERSECT, and EXCEPT**

Supported Data Types



Not supported:

- Any kind of LOB
- ROWID
- XML
- Binary data

Reasons a query might not be routed to ISAO

- **Because the accelerator or AQT is currently disabled**
- **Because a query contains**
  - References to a table or column not in the accelerated mart
  - Unsupported data type
  - Unsupported syntax
    - E.g., Subselect or FULL OUTER JOIN
  - CURRENT REFRESH AGE = 0
- **Because the query does not reference a fact table**
- **Because the DB2 Optimizer decides that DB2 can do better**
  - DB2 has a cost-based threshold
  - E.g., Query with selective predicate on indexed column is executed in DB2

EXPLAIN Will Tell You Why

- A new EXPLAIN table is added to show:
  - Whether or not a query block is eligible for automatic query rewrite
    - If not eligible, why it's not eligible
  - If eligible for automatic query rewrite:
    - Which materialized / accelerated query tables were considered
    - For each one not chosen, why not chosen.
- DDL for this new EXPLAIN table:

```
CREATE TABLE DSN_QUERYBLOCKINFO_TABLE(
   QUERYNO INTEGER NOT NULL WITH DEFAULT,
   QBLOCKNO SMALLINT NOT NULL WITH DEFAULT,
   ...
   QB_REASON SMALLINT NOT NULL WITH DEFAULT,
   QB_INFO CLOB(2MB) NOT NULL WITH DEFAULT,
) CCSID UNICODE;
```

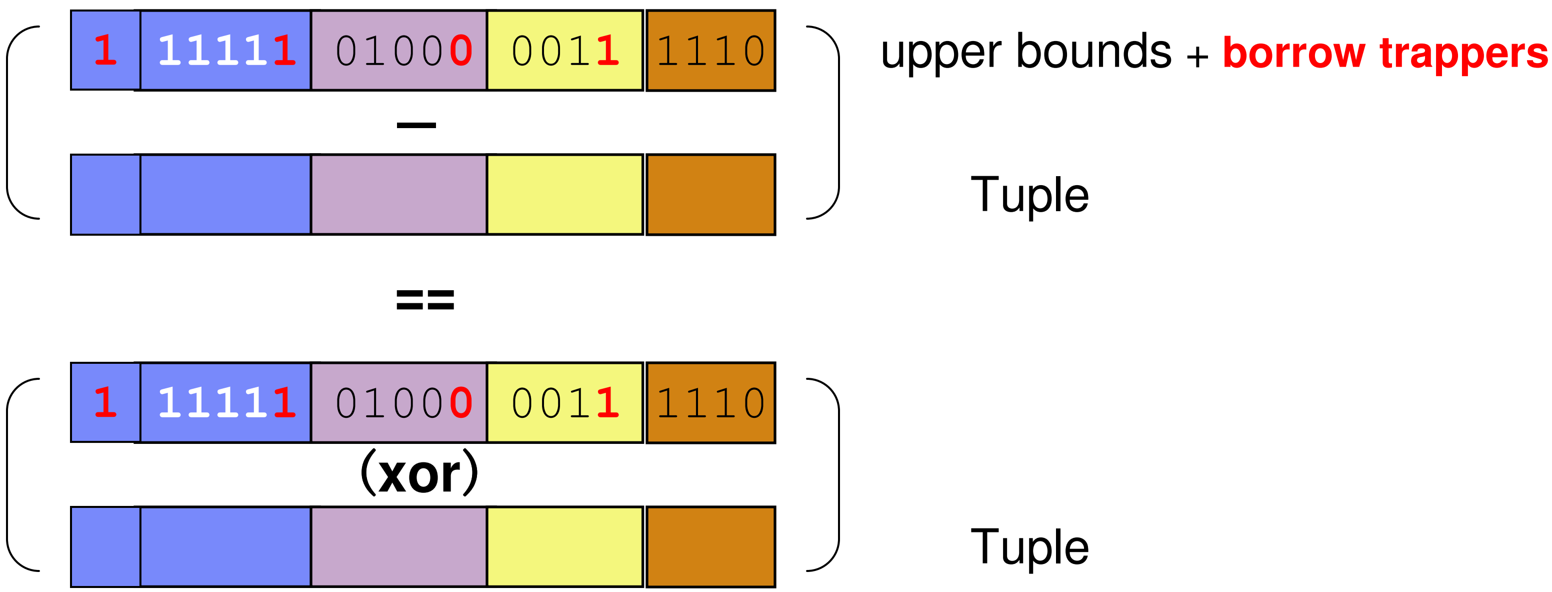
  - Column "QB_INFO" contains (in XML format) objects, functions, etc. that caused an AQT to not be chosen.

Evaluation of Range Predicates on Encoded Values

B<=‘CA’ and C < 17 and D <= ‘Q4’

Translate value query to encoded query
(exploits ***order-preserving code***)

A <= 11111 and B<=01000 and C <=0011 and D<=1110



General result: (UB – Tuple) xor (Tuple – LB)   ==   UB xor LB

Different Data Stores for Different Workloads

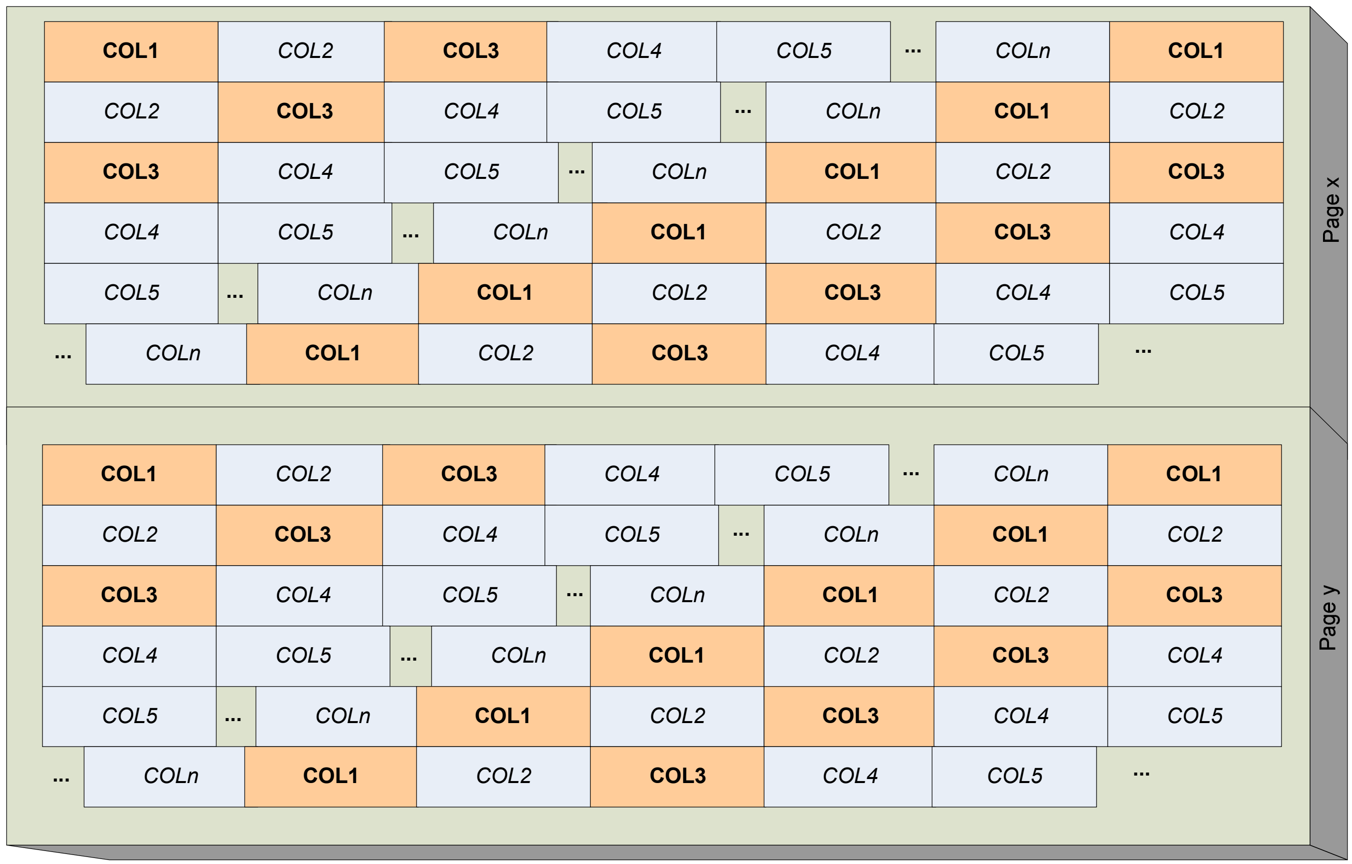
- **Transactional workloads typically:**
  - Normally fetch and use all attributes (columns) of a row
  - E.g., for a CUSTOMERS table, wouldn‘t fetch the Street_Name without also fetching the House_Number or ZIP_Code.
  - Fetches few, very specific records of a table

- **Data Warehouse workloads typically:**
  - Have very wide tables, having multiple measure columns
  - Almost never query all attributes of the rows
  - Fetch only a small subset of any table‘s columns.
  - Need to access and aggregate many rows per table

COL1	COL2	**COL3**	COL4	COL5	···	COLn

Row Stores

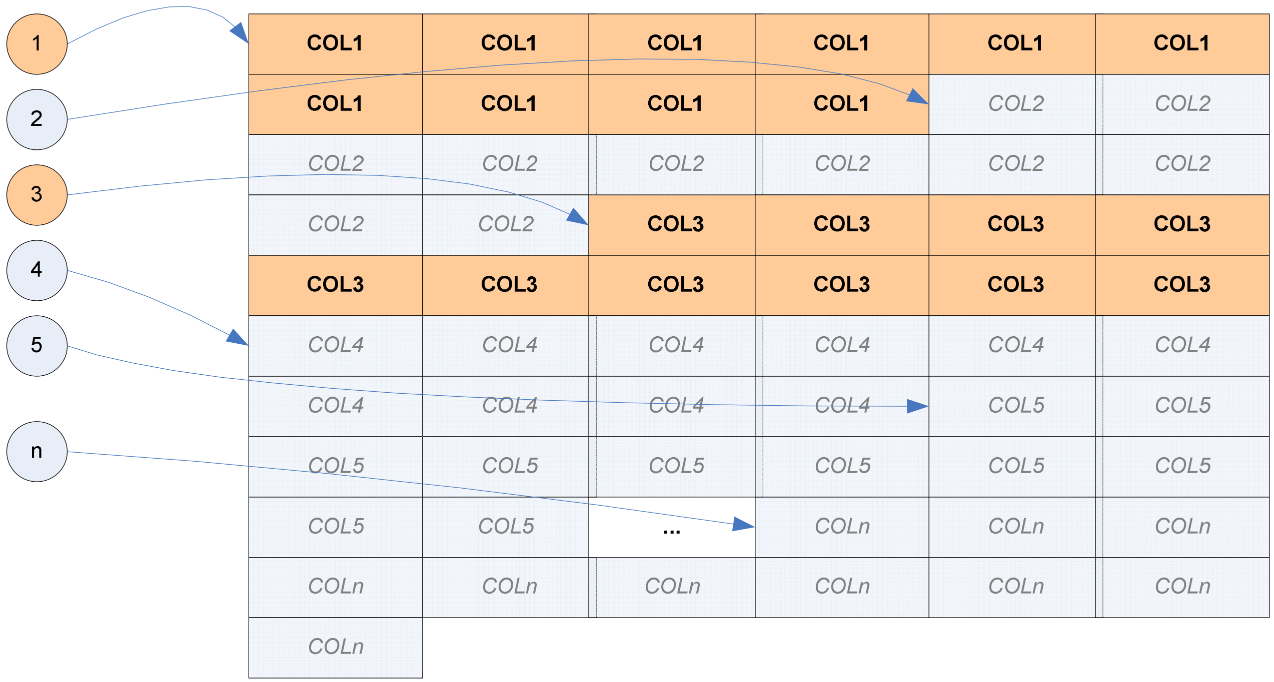
- In traditional DBMS, we use a **Row Store:**
- Each row is stored complete
- Multiple rows are stored sequentially in I/O-optimized data structures.
- Even if only a few attributes are required, the complete row still needs to be fetched and de-compressed.
- Most of the data is moved and de-compressed without even being used!



While a **Row Store** is very efficient for transactional workloads, it is sub-optimal for analytical workloads, for which only a subset of the attributes is fetched!

Column Store Layout

- Query Engines, which are optimized for analytical queries, tend to use a **Column Store** approach:
- In a **Column Store**, the data of a specific column is stored sequentially, before the data of the next column begins.
- If attributes are not required for a specific query execution, they can simply be skipped completely, saving any I/O and de-compression effort!

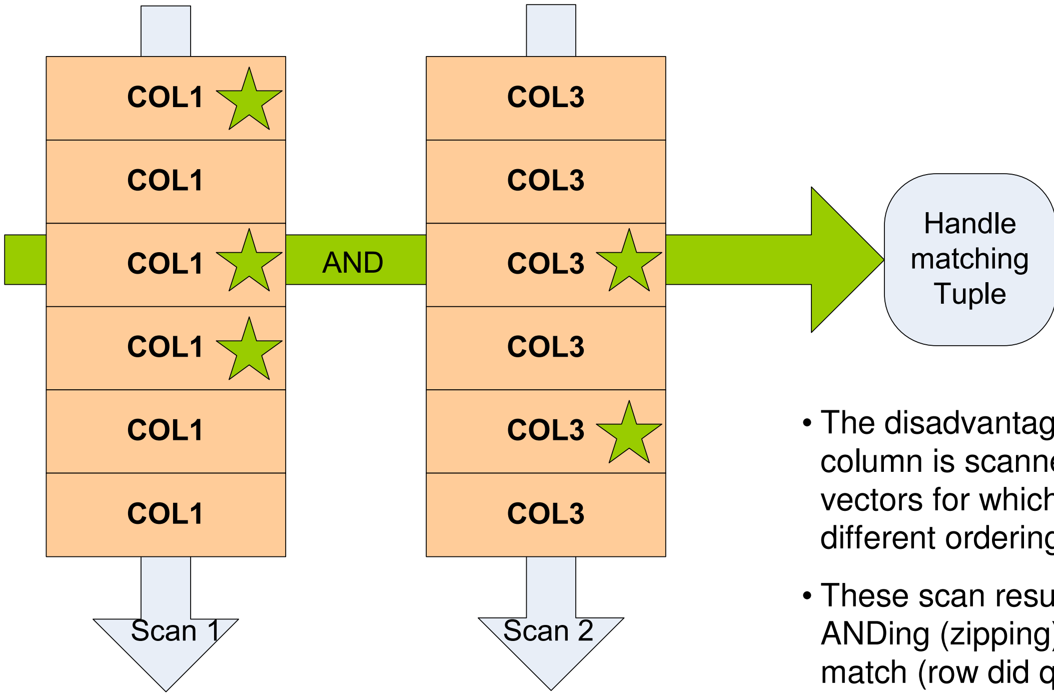


In a **Column Store**, each column is also typically re-ordered on that column's domain, and compressed sequentially, to permit prefix and run-length encoding.

- This is optimized for sequential scans of the data.
- But random access to specific attributes performs very poorly.

This is normally compensated for by limiting the number of rows per column before the next column is stored. (The data is split into blocks.)

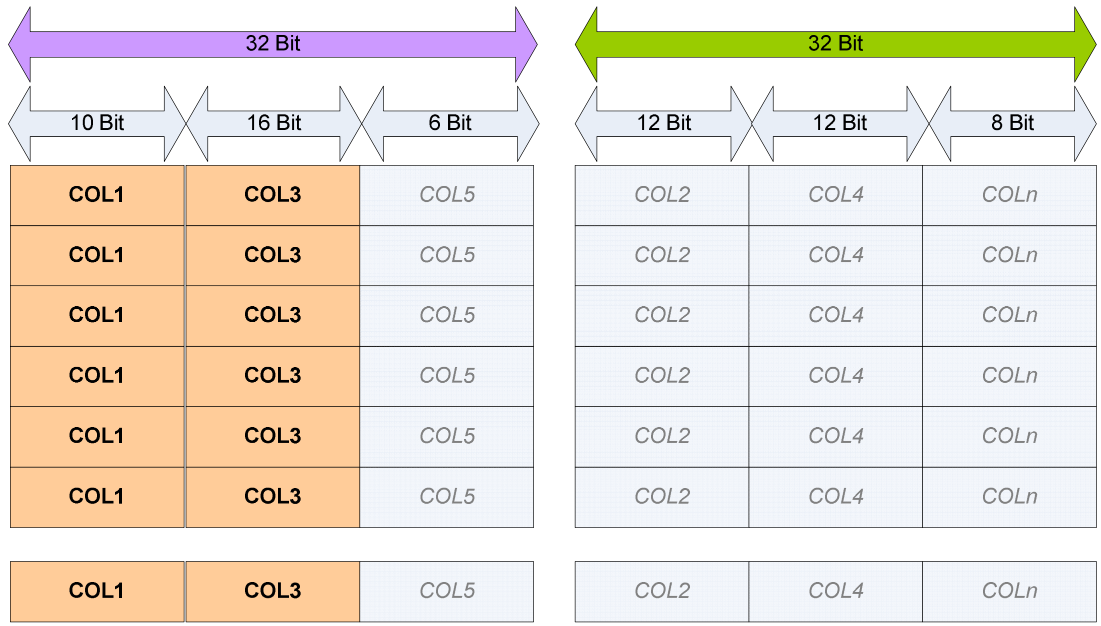
Zipping Columns Back Together in Column Stores



- The disadvantage of a **Column Store** approach is that each column is scanned independently, resulting in multiple result vectors for which predicates did or did not match, and may have different orderings (not RID order).
- These scan results from each column need to be combined by ANDing (zipping) them together to figure out if all predicates did match (row did qualify), before the corresponding measure columns can be accessed for processing.
- This ANDing is a siginificant and complicated process, whose cost increases with row count and number of columns.
- The access to the measure columns for processing (i.e., aggregation) is a „random access" that doesn't perform well on pure **Column Stores**.
- The width of a compressed column often doesn't match the processor architecture, resulting in considerable padding.

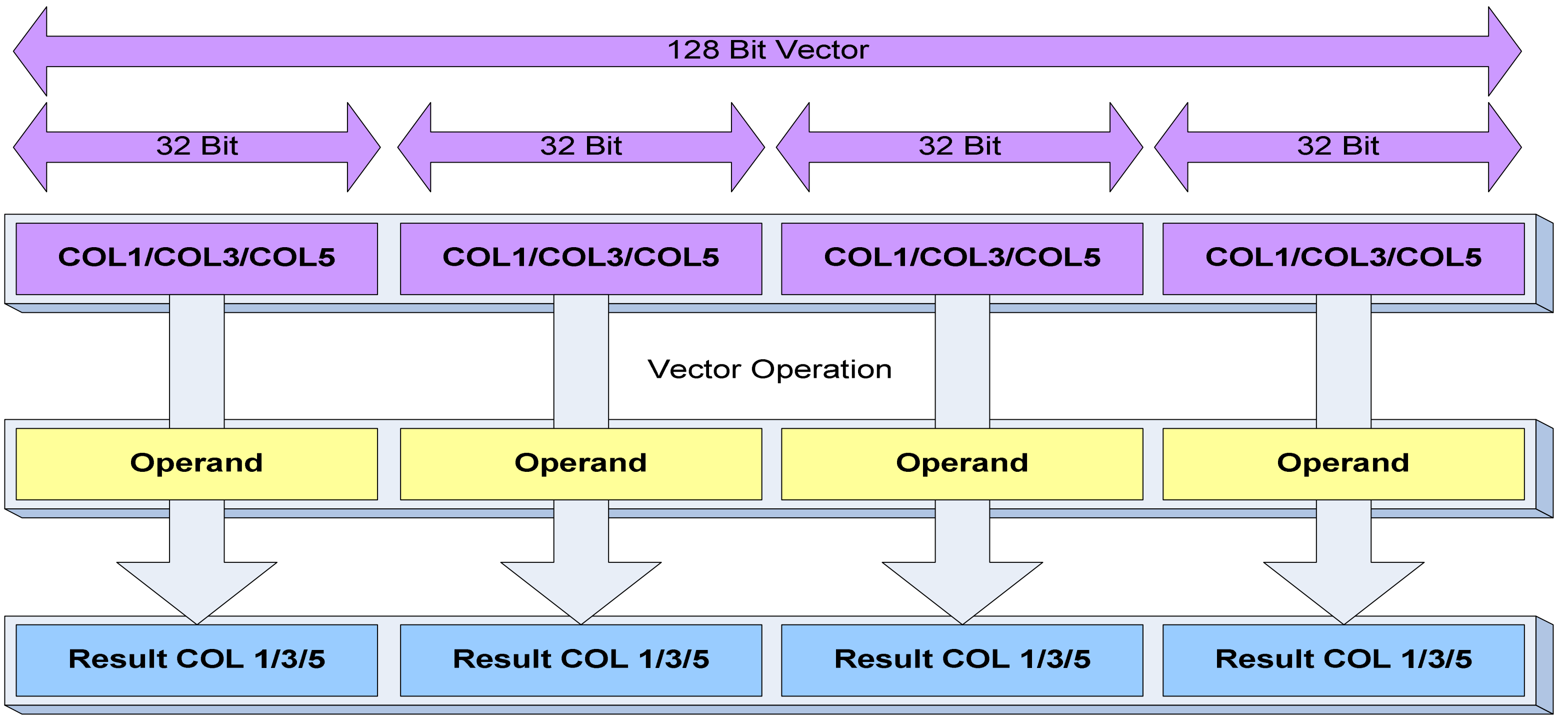
Register Store: A Hybrid of Row Store & Column Store

- Within a **Register Store**, several columns are grouped together.
- The sum of the width of the compressed columns doesn‘t exceed a register-compatible width. This could, for example, be 32 or 64 bits, for a 64-bit system. It doesn‘t matter how many columns are placed within the register-wide data element.
- It is beneficial to place commonly-used columns within the same register-wide data element. But this requires advance knowledge about the workload that will be executed (runtime statistics).
- Having multiple columns within the same register-wide data element saves having to AND (zip together) those columns.



The **Register Store** is an optimization of the **Column Store** approach that makes the best use of existing hardware. It saves the time-consuming re-shuffling of small data elements at run-time that **Column Stores** require. The **Register Store** also exploits vectorization well (next slide).

Register Stores Facilitate SIMD Parallelism

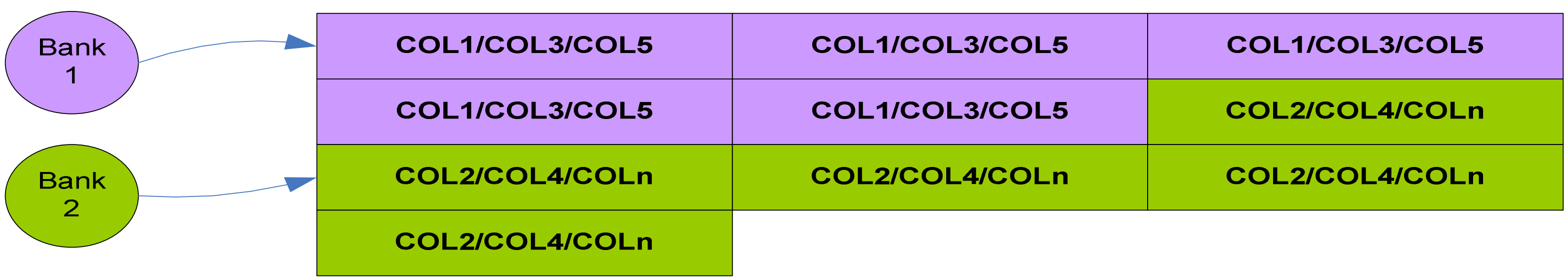


Packing multiple rows from the same bank into a 128-bit register permits
yet another level of parallelism – SIMD (Single-Instruction, Multiple Data)!

Register Store Optimizes a Column Store

The **Register Store** stores the register-wide, multi-column vertical partitions just as a **Column Store** would store each column. This permits:

- Optimized sequential access
- Exploitation of vector operations
- Less ANDing efforts (so fewer random access lookup operations)



The IBM Smart Analytics Optimizer is a Register Store

- A hybrid of a traditional Row Store and a Column Store
- Optimized for extremely efficient **scans** over data.