CSE 544
Principles of Database Management Systems

Alvin Cheung

Fall 2015

Lecture 8 - Data Warehousing and Column Stores

Announcements

- Shumo office hours change
  - See website for details
- HW2 due next Thurs
  - Please start soon!
- Project meetings this week
  - Sign up on doodle if you haven't already

Where We Are

- Data models
  - Relational
  - IMS / Codasyl
  - Unstructured

- Query processing
  - Algorithms for relational operators
  - Indexing and physical design

- Queries that real-world users write
  - **Data warehousing**
  - Transaction processing

Where We Are

- What queries do real people write?
  - **Data warehousing**
    - Column stores (today)
    - Parallel databases (Thursday)
    - Programming models (next week)
  - Transaction processing

References

- Data Cube: A Relational Aggregation Operator Generalizing Group By, Cross-Tab, and Sub-Totals.

  Jim Gray et. al. Data Mining and Knowledge Discovery 1, 29-53. 1997

- Database management systems.

  Ramakrishnan and Gehrke.

  Third Ed. **Chapter 25**

Why Data Warehouses?

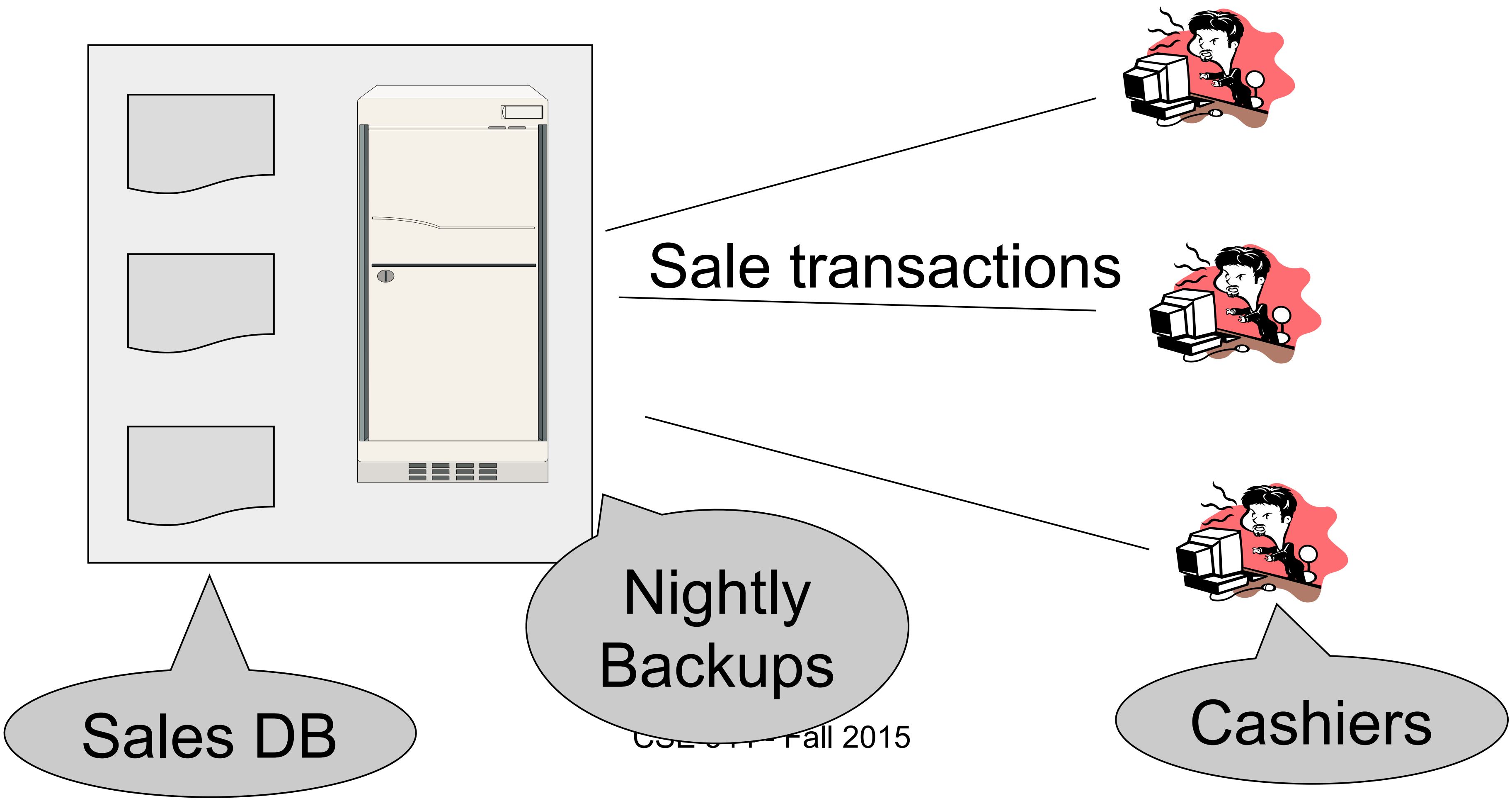
- DBMSs designed to manage operational data
  - Goal: support every day activities
  - Online transaction processing (OLTP)
  - Ex: Tracking sales and inventory of each Wal-mart store

- Enterprises also need to analyze and explore their data
  - Goal: summarize and discover trends to support decision making
  - Online analytical processing (OLAP)

- To support OLAP consolidate all data into a **warehouse**

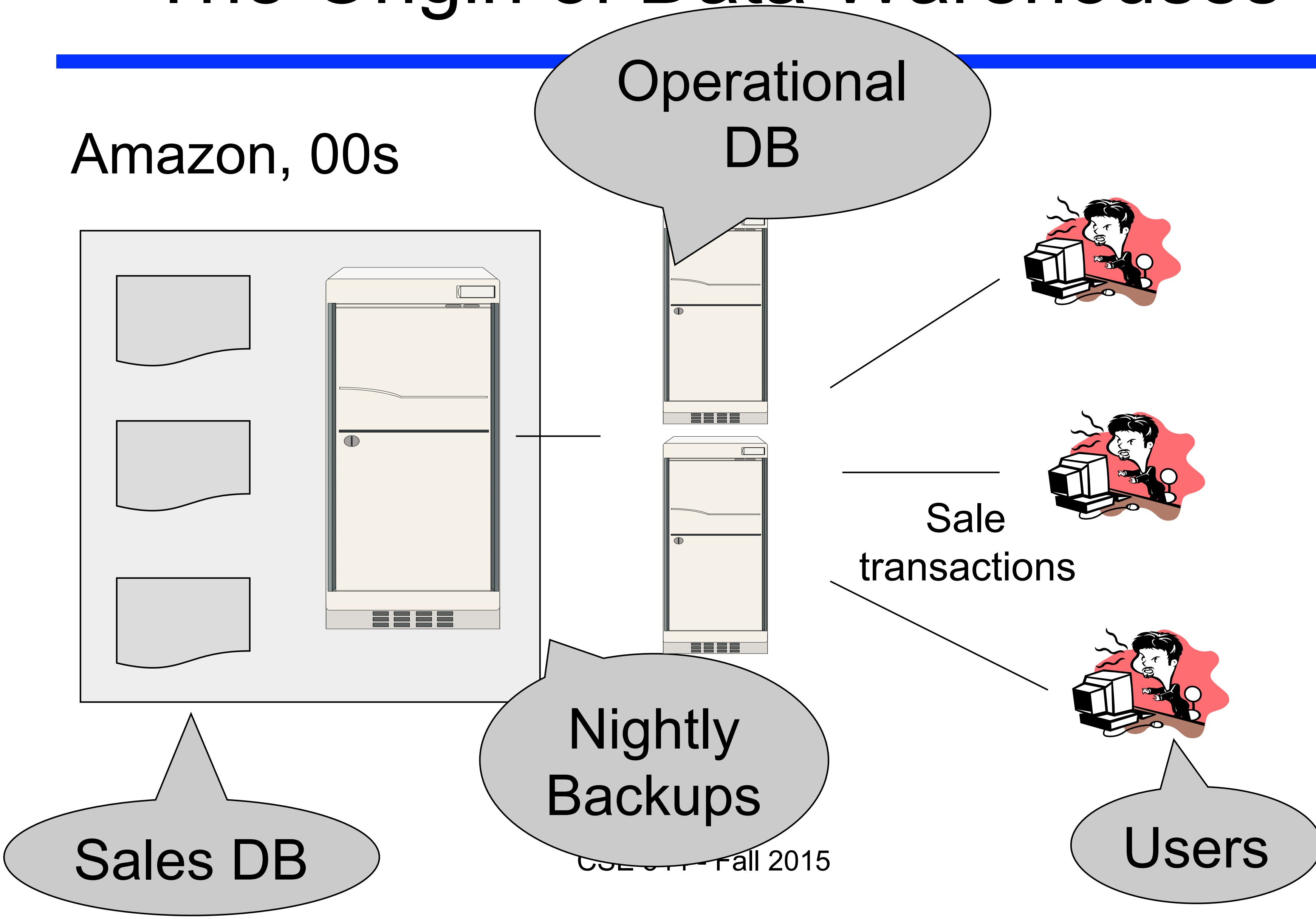
The Origin of Data Warehouses

Walmart, 90s



The Origin of Data Warehouses

Amazon, 00s



Data Warehouse Overview

- Consolidated data from many sources
  - Must create a single unified schema
  - The warehouse is like a materialized view
- Very large size: terabytes of data are common
- Complex read-only queries (no updates)
- Fast response time is (not as) important
  - Compared to transaction processing

Creating a Data Warehouse

- **Extract** data from distributed operational databases
- **Clean** to minimize errors and fill in missing information
- **Transform** to reconcile semantic mismatches
  - Performed by defining views over the data sources
- **Load** to materialize the above defined views
  - Build indexes and additional materialized views
- **Refresh** to propagate updates to warehouse periodically
- This is known as the **ETL pipeline**

Alternative: Distributed DBMS

- User submits a query at one site
- Query is defined over data located at different sites
  - Different physical locations
  - Different types of DBMSs
- Query optimizer finds the best distributed query plan
  - Query executes across all the locations
  - Results shipped to query site and returned to user
- (Stay tuned for the next lecture!)

Back to Warehouses: Outline

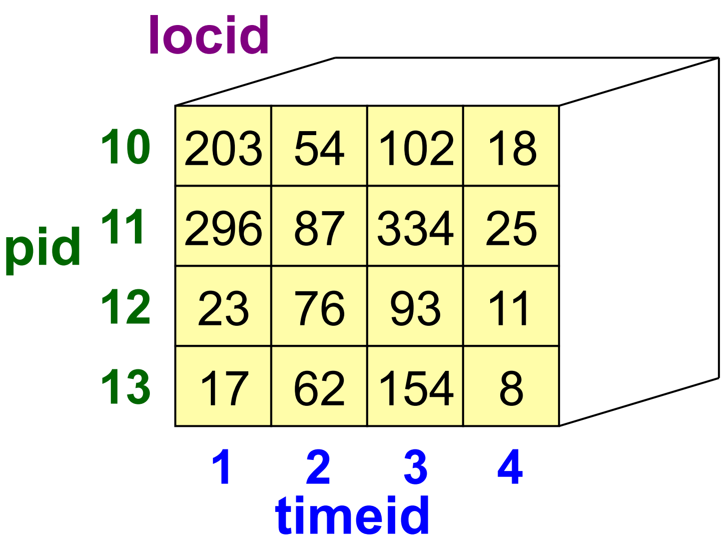
- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Data Analysis Cycle

- Formulate query that extracts data from the database
  - Typically ad-hoc complex query with group by and aggregate
- Visualize the data (e.g., spreadsheet)
  - Dataset is an N-dimensional space
- Analyze the data
  - Identify "interesting" subspace by aggregating other dimensions
  - Categorize the data an compare categories with each other
  - Roll-up and drill-down on the data

Multidimensional Data Model

- Focus of the analysis is a collection of measures
  - Example: Wal-mart sales
- Each measure depends on a set of dimensions
  - Example: product (pid), location (lid), and time of the sale (timeid)



Slicing: equality selection on one or more dimensions

Dicing: range selection

Star Schema

Representing multidimensional data as relations (ROLAP)

Product

pid	pname	category	price

Location

locid	city	state	country

Facts table: Sales

In BCNF

Sales

pid	timeid	locid	sales

Dimensions tables

- Product
- Location
- Times

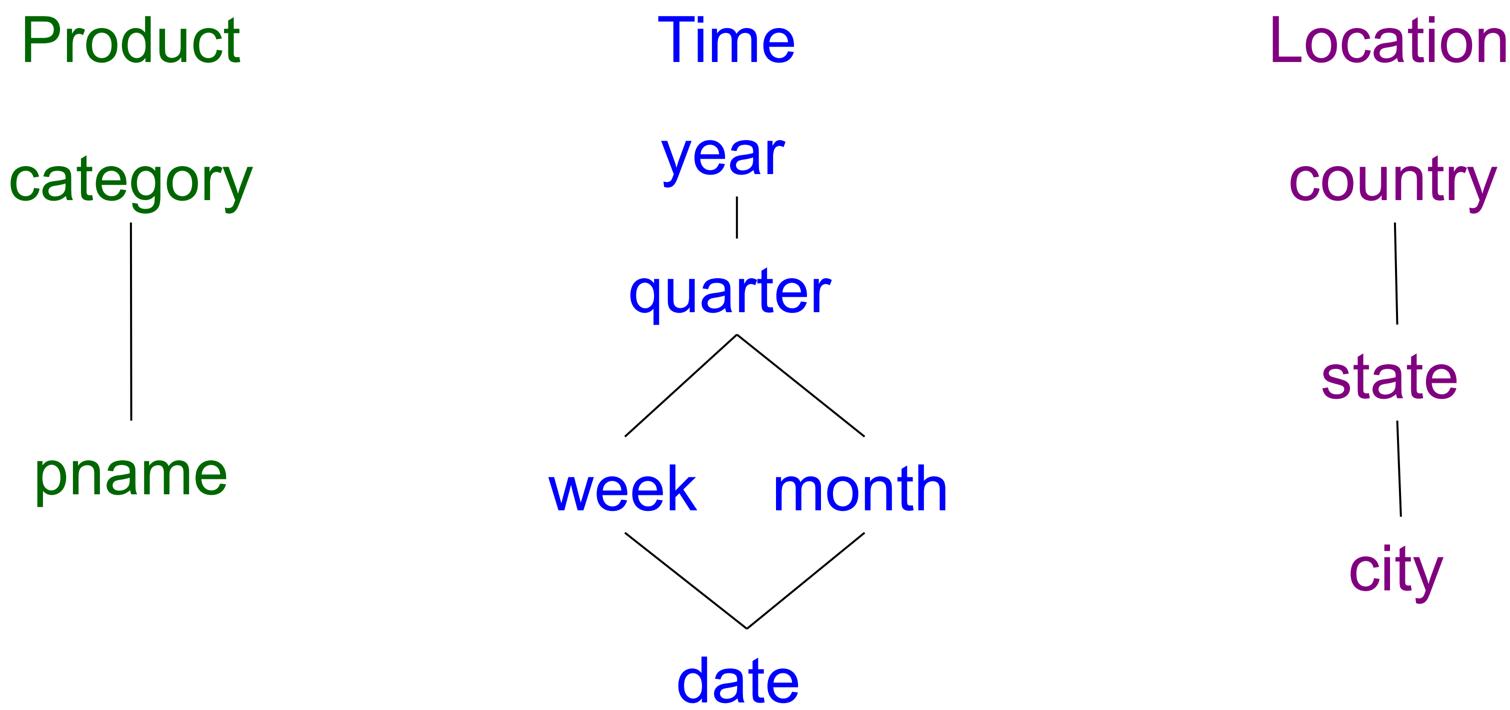
Not necessarily normalized

timeid	date	week	month	quarter	year

Times

Dimension Hierarchies

Dimension values can form a hierarchy described by attributes



Desired Operations

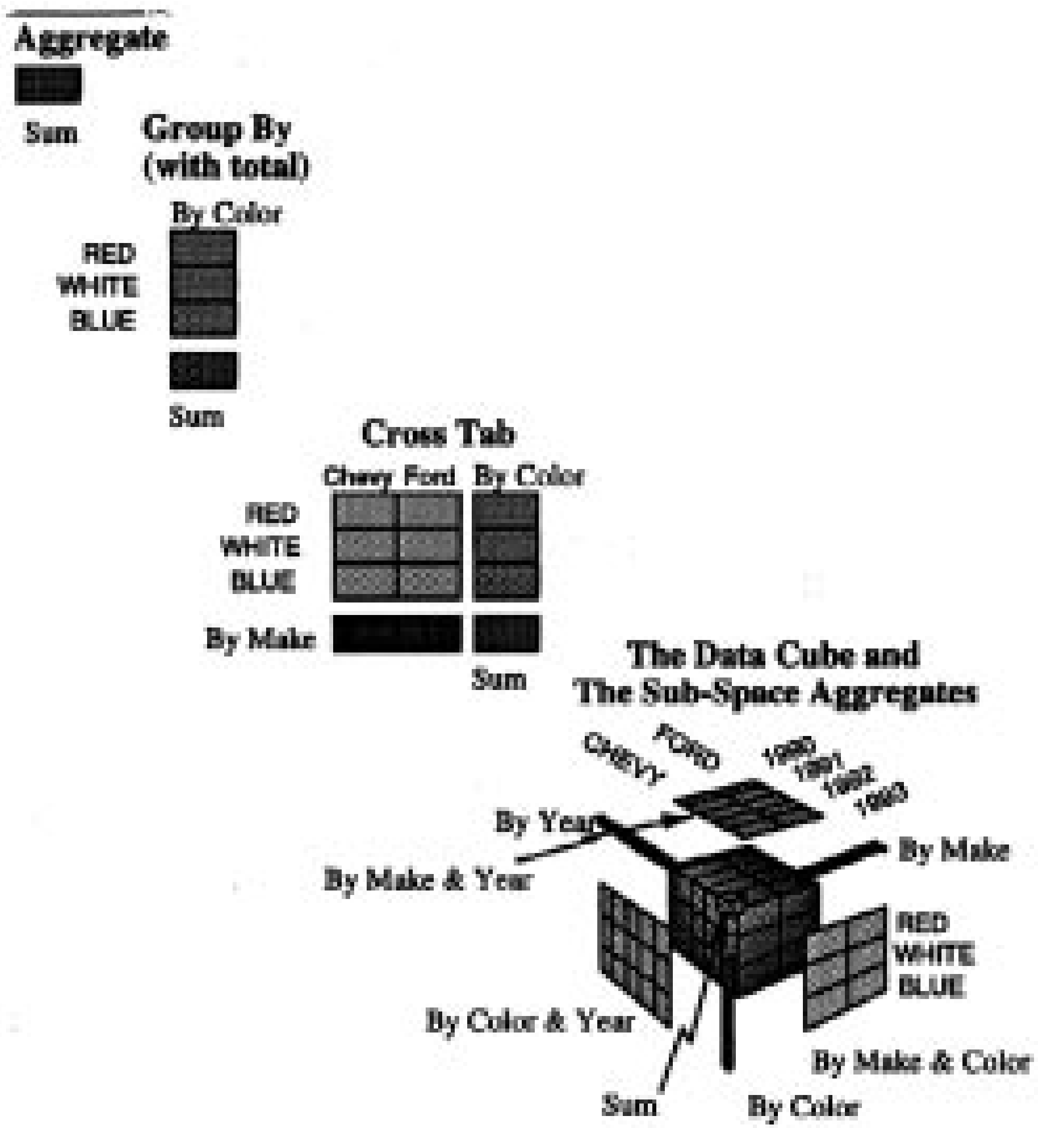
- Histograms (agg. over computed categories)
  - Problem: awkward to express in SQL (paper p.34)
- Summarize at different levels: roll-up and drill-down
  - Ex: total sales by day, week, quarter, and year
- Pivoting
  - Ex: pivot on location and time
  - Result of pivot is a cross-tabulation
  - Column values become labels

	WI	CA	Total
2005	500	200	**700**
2006	150	850	**1000**
2007	250	400	**650**
Total	**900**	**1450**	**2350**

Challenge 1: Representation

- Problem: How to represent multi-level aggregation?
  - Ex: Table 3 in the paper need 2^N columns for N dimensions!
  - Ex: Table 4 has even more columns!

Challenge 1: Representation



```
SELECT Model, Year, Color, SUM(sales) AS Sales
FROM Sales
WHERE Model in ('Ford', 'Chevy')
  AND Year BETWEEN 1990 AND 1992
GROUP BY CUBE Model, Year, Color;
```

SALES

Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

CUBE

DATA CUBE

Model	Year	Color	Sales
Chevy	1990	blue	62
Chevy	1990	red	5
Chevy	1990	white	95
Chevy	1990	ALL	154
Chevy	1991	blue	49
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	ALL	198
Chevy	1992	blue	71
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	ALL	156
Chevy	ALL	blue	182
Chevy	ALL	red	90
Chevy	ALL	white	236
Chevy	ALL	ALL	508
Ford	1990	blue	63
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	ALL	189
Ford	1991	blue	55
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	ALL	116
Ford	1992	blue	39
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	ALL	128
Ford	ALL	blue	157
Ford	ALL	red	143
Ford	ALL	white	133
Ford	ALL	ALL	433
ALL	1990	blue	125
ALL	1990	red	69
ALL	1990	white	149
ALL	1990	ALL	343
ALL	1991	blue	106
ALL	1991	red	104
ALL	1991	white	110
ALL	1991	ALL	314
ALL	1992	blue	110
ALL	1992	red	58
ALL	1992	white	116
ALL	1992	ALL	284
ALL	ALL	blue	339
ALL	ALL	red	233
ALL	ALL	white	369
ALL	ALL	ALL	941

Challenge 1: Representation

- Problem: How to represent multi-level aggregation?
  - Ex: Table 3 in the paper need 2^N columns for N dimensions!
  - Ex: Table 4 has even more columns!
  - And that's without considering any hierarchy on the dimensions!
- Solution: special "all" value

T.year	L.state	SUM(S.sales)
2005	WI	**500**
2005	CA	**200**
2005	ALL	**700**
...	...	**...**
ALL	ALL	**2350**

Note: SQL-1999 standard uses NULL values instead of ALL

Challenge 2: Computing Aggregations

- Need 2^N different SQL queries to compute all aggregates
  - Expressing roll-up of a single column and cross-table queries is thus daunting
  - Cannot optimize all these independent queries

- Solution: CUBE and ROLLUP operators

Outline

- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Data Cube

- CUBE is the N-dimensional generalization of aggregate

- Cube in SQL-1999

```
SELECT T.year, L.state, SUM(S.sales)
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid and S.locid=L.locid
GROUP BY CUBE (T.year,L.state)
```

- Creating a data cube requires generating the power set of the aggregation columns

Rollup

- Rollup produces a subset of a cube

- Rollup in SQL-1999

```
SELECT T.year,T.quarter, SUM(S.sales)
FROM Sales S, Times T
WHERE S.timeid=T.timeid
GROUP BY ROLLUP (T.year,T.quarter)
```

- Will aggregate over each pair of (year,quarter), each year, and total, but will **not** aggregate over each quarter

Computing Cubes and Rollups

- Naive algorithm
  - For each new tuple, update each of 2^N matching cells
- More efficient algorithm
  - Use intermediate aggregates to compute others
  - Relatively easy for distributive and algebraic functions
- Updating a cube in response to updates is more challenging

Outline

- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Indexes

- Bitmap indexes: good for sparse attributes (few values)

M	F
0	1
1	0
1	0

custid	name	gender	rating
10	Alice	F	3
11	Bob	M	4
12	Chuck	M	1

1	2	3	4
0	0	1	0
0	0	0	1
1	0	0	0

- Join indexes: to speed-up specific join queries
  - Example: Join fact table F with dimension tables D1 and D2
  - Index contain triples of rids $<r_1,r_2,r>$ from D_1, D_2, and F that join
  - Alternatively, two indexes, each one with pairs $<v_1,r>$ or $<v_2,r>$ where v_1, v_2 are values of tuples from D_1, D_2 that join with r

Materialized Views

- How to choose views to materialize?
  - Physical database tuning

- How to keep view up-to-date?
  - Could recompute entire view for each update: expensive
  - Better approach: incremental view maintenance
  - Example: recompute only affected partition

  - How often to synchronize? Periodic updates (at night) are typical
    - Think back in the case of Walmart

Outline

- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Additional Extensions for Decision Support

- Window queries

```
SELECT L.state, T.month, AVG(S.sales) over W AS movavg
FROM Sales S, Times T, Locations L
WHERE S.timeid = T.timeid AND S.locid=L.locid
WINDOW W AS (PARTITION BY L.State
             ORDER BY T.month
             RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
             AND INTERVAL '1' MONTH FOLLOWING)
```

- Top-k queries: optimize queries to return top k results
- Online aggregation: produce results incrementally

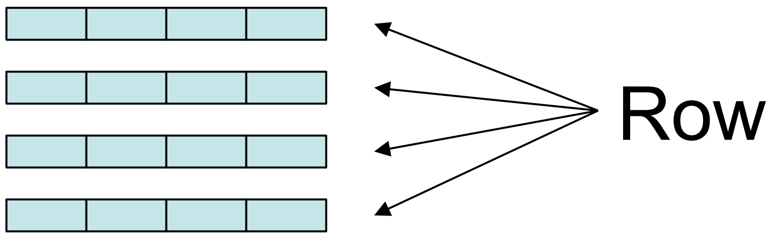
Leveraging Column Stores

References

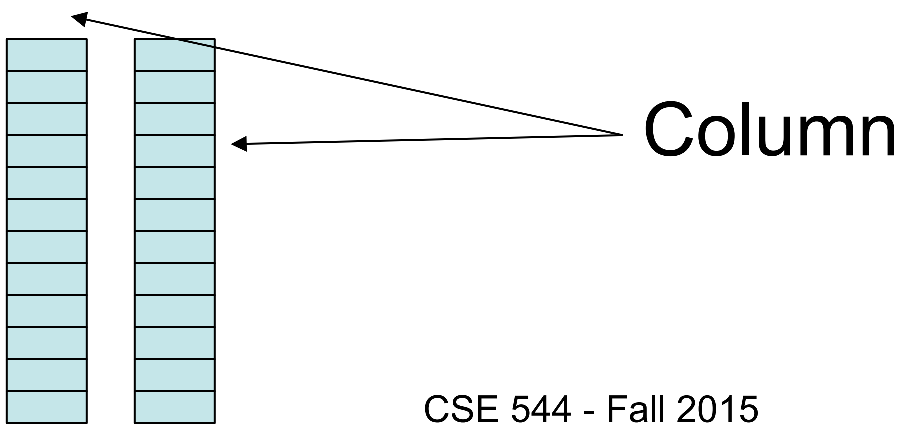
- The Design and Implementation of Modern Column-Oriented Database Systems Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, Samuel Madden. Foundations and Trends® in Databases (Vol 5, Issue 3, 2012, pp 197-280).

Main Idea

- Most DBMS (up till now) store each tuple using row-major order

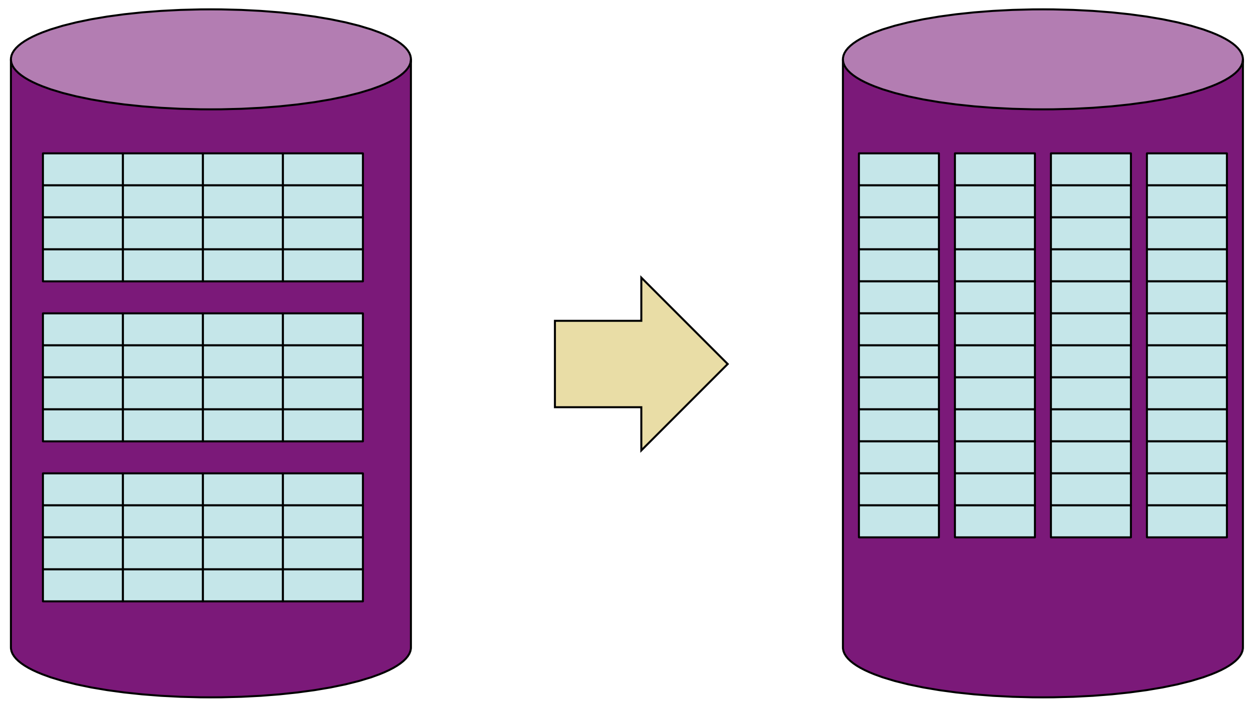


- Store tuples by column-major order instead?



Why?

From Row-Store to Column-Store

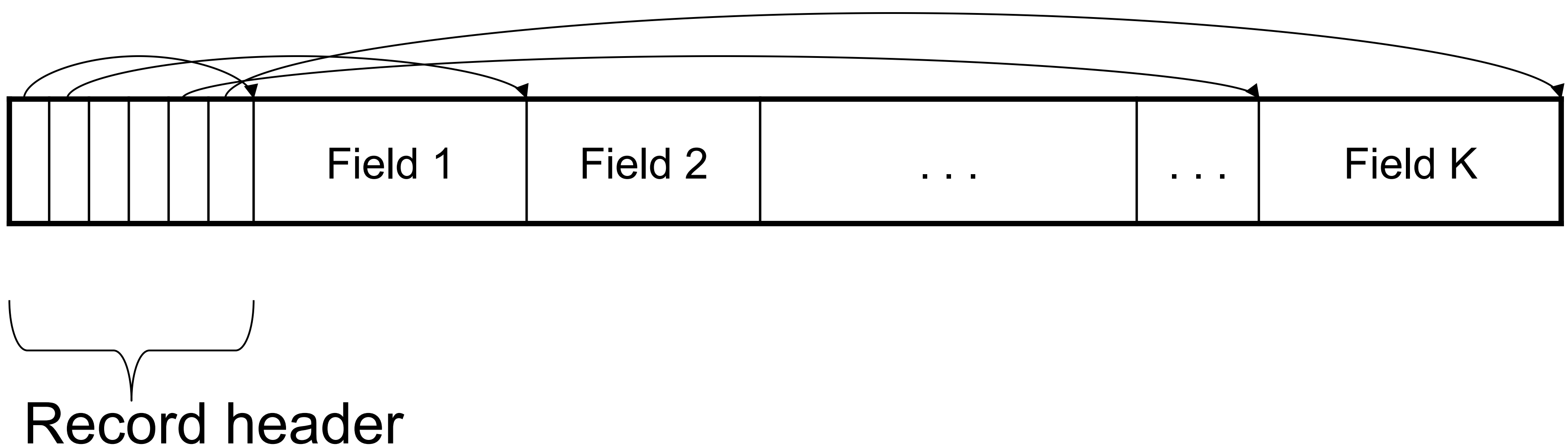


Rows stored contiguously on disk (+ tuple headers)

Columns stored contiguously on disk (no headers needed)

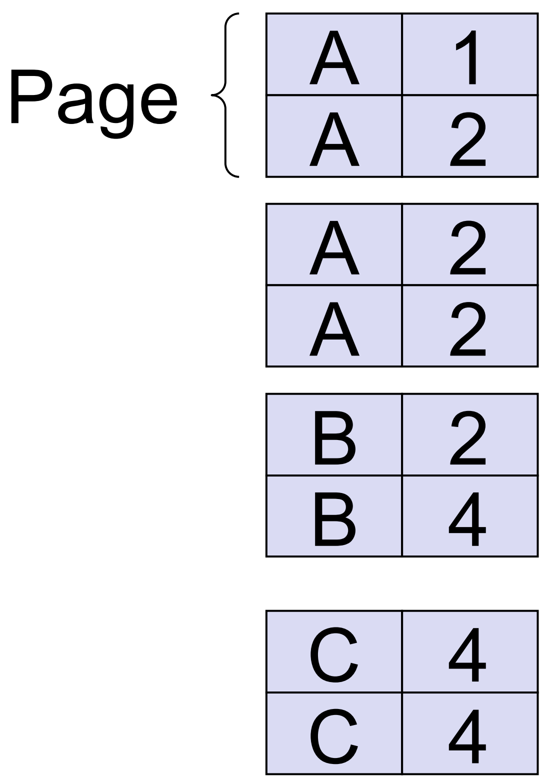
Recall Record Formats in Row Stores

Variable length records

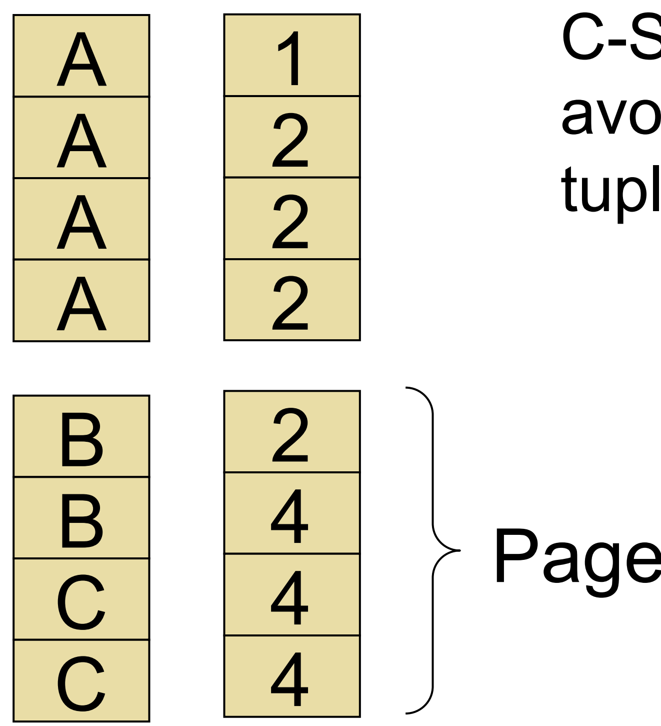


More Detailed Example

Row-based
(4 pages)



Column-based
(4 pages)



C-Store also avoids large tuple headers

Column-Store Optimizations

Numbers from earlier paper and **C-Store system**: “Column-Stores vs. Row-Stores: How Different Are they Really?” Abadi et. al. SIGMOD’08.

- Vectorized processing / Block iteration (1.5X)
  - Pass blocks of values between ops instead of individual tuples
- Compression: e.g., run-length encoding of columns (10X)
- Late tuple materialization (3X improvement)
  - Process individual columns as long as possible
  - Merge columns into complete tuples as late as possible
- Invisible joins (1.5X)

Compression Example

Row-based (4 pages)

Page

A	1
A	2

A	2
A	2

B	2
B	4

C	4
C	4

Column-based (4 pages)

A	1
A	2
A	2
A	2

B	2
B	4
C	4
C	4

Page

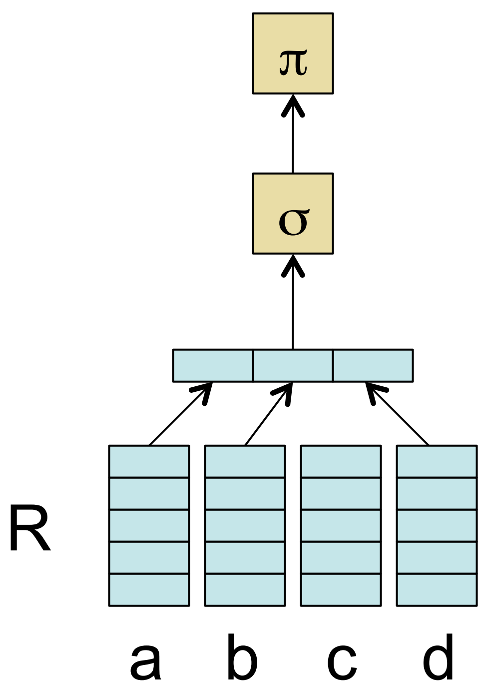
Compressed (2 pages)

4XA	1X1
2XB	4X2
2XC	5X4

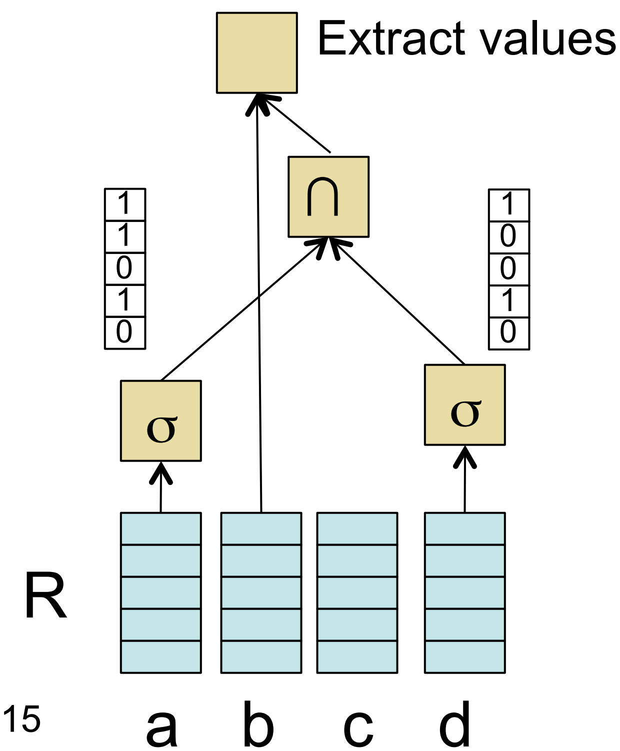
Late Tuple Materialization

Ex: `SELECT R.b from R where R.a=X and R.d=Y`

Early materialization



Late materialization



Late Tuple Materialization

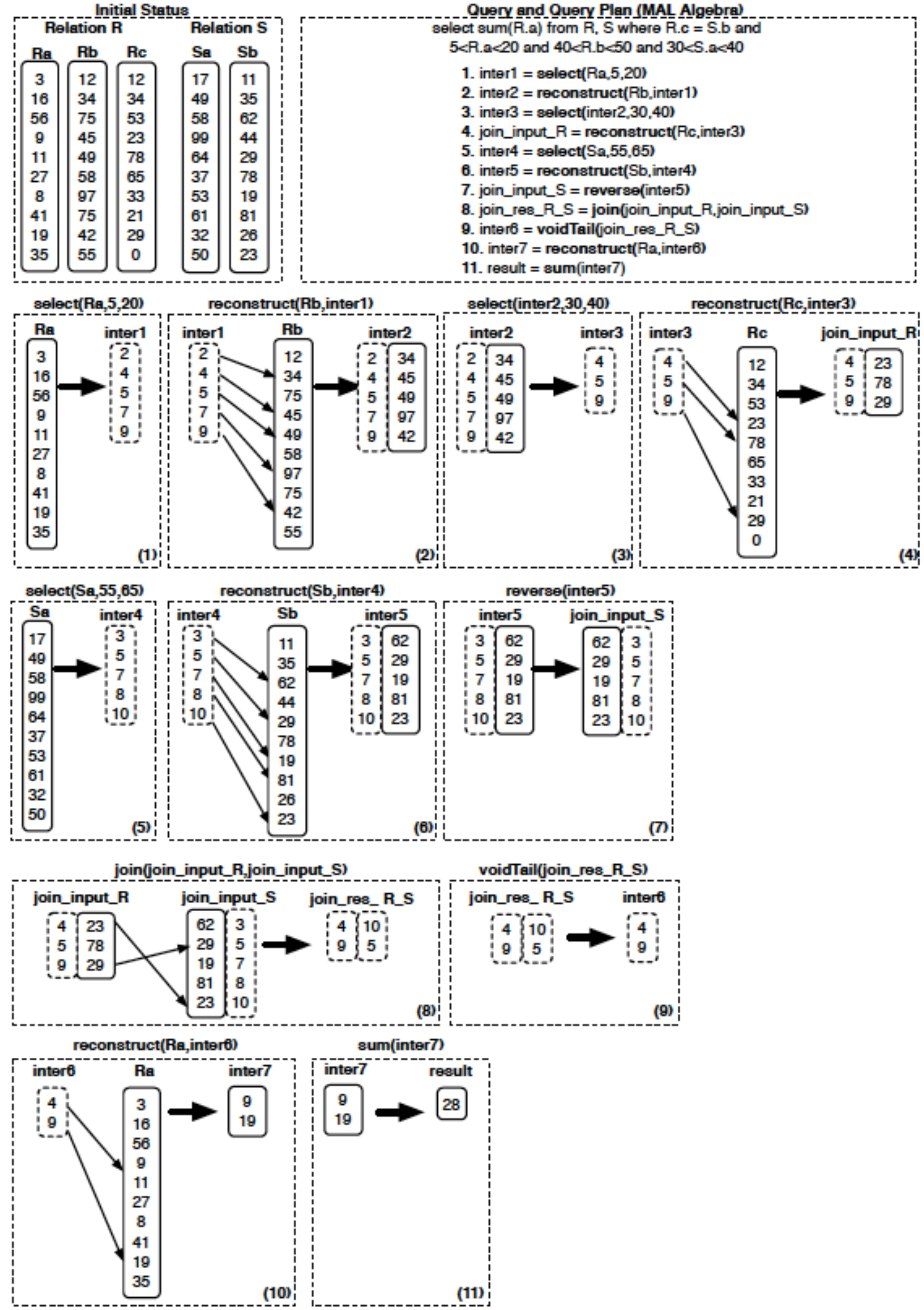


Figure 4.1: An example of a select-project-join query with late materialization.

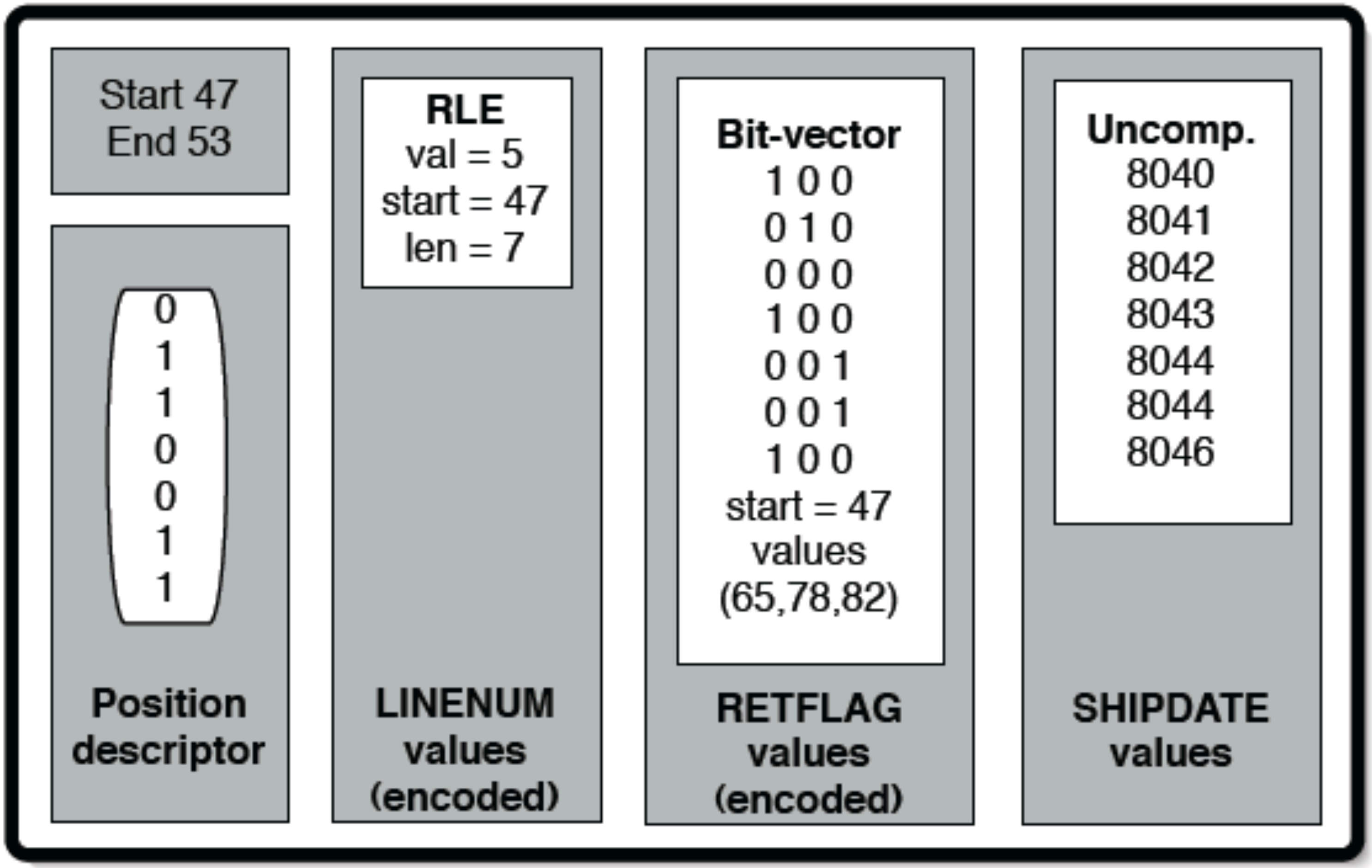
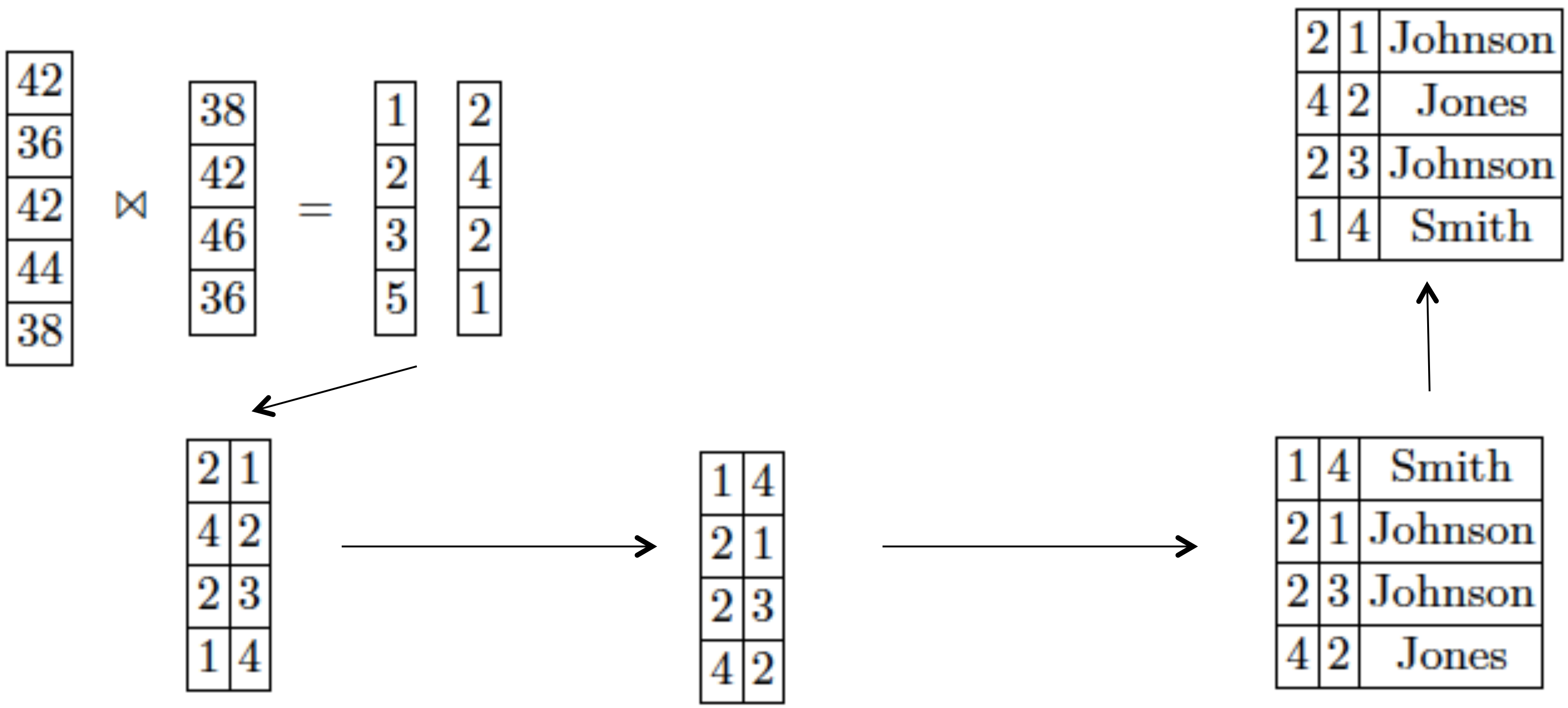


Figure 4.2: An example multi-column block containing values for the SHIPDATE, RETFLAG, and LINENUM columns. The block spans positions 47 to 53; within this range, positions 48, 49, 52, and 53 are active (i.e., they have passed all selection predicates).

Joins

```
SELECT emp.age, dept.name
FROM emp, dept
WHERE emp.dept_id = dept.id
```



Simulating a Column-Store DBMS in a Row-Store DBMS

- Vertical partitioning
  - Two-column tables: (key, attribute)

- Index-only plans
  - Create a B+ tree index on each attribute
  - Answer queries using indexes only, without reading actual data

- Materialized views
  - Each view contains a subset of columns

Conclusion

- Column-store DBMS outperforms row-store DBMS
  - Measured on a data warehousing benchmark (SSBM)
- Late materialization and compression are key factors
- Difficult to simulate a column-store in a row-store
  - Tuple overheads cause data blow-up
  - Column joins are expensive
  - Hard to get the DBMS to “do the right thing” (e.g., index plans)
- Not the end of the story, however, … see CIDR’09 paper