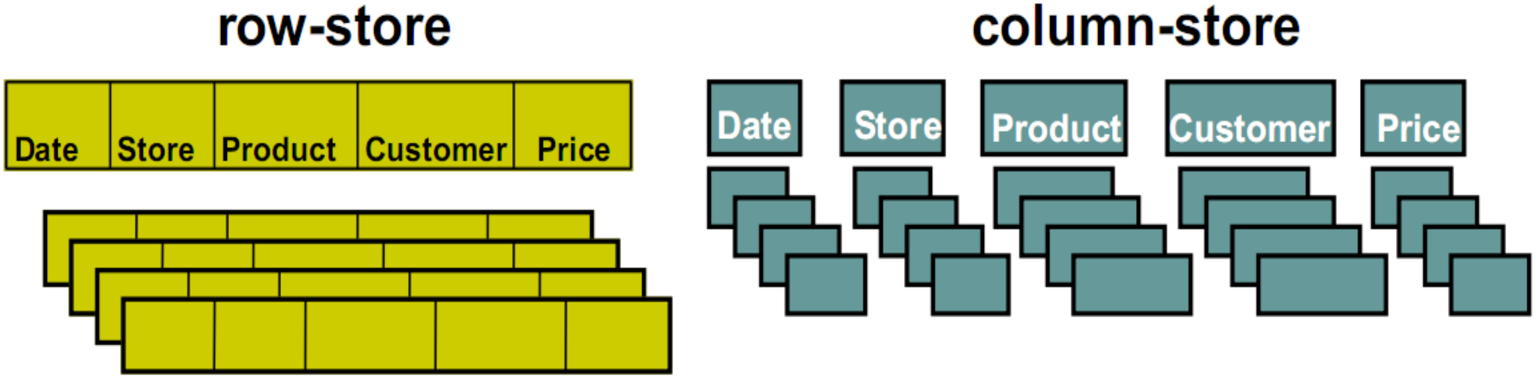
An invention in 2000s: Column Stores for OLAP

ΠΜΣ "Ερευνητικές Κατευθύνσεις στην Πληροφορική"

Επεξεργασία και Ανάλυση Δεδομένων

SPRING SEMESTER 2020

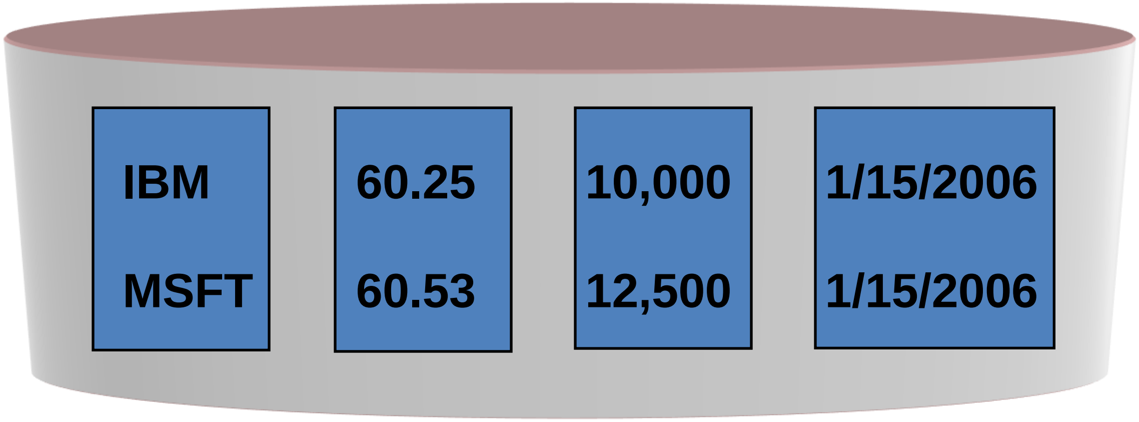
Row Store and Column Store (logical level)



- In a row store, data are stored in the disk tuple-by-tuple.
- In a column store, data are stored in the disk column by column
- Columnar DBMS are special purpose databases and are not designed to replace general purpose RDBMS.

Row Store vs Column Store

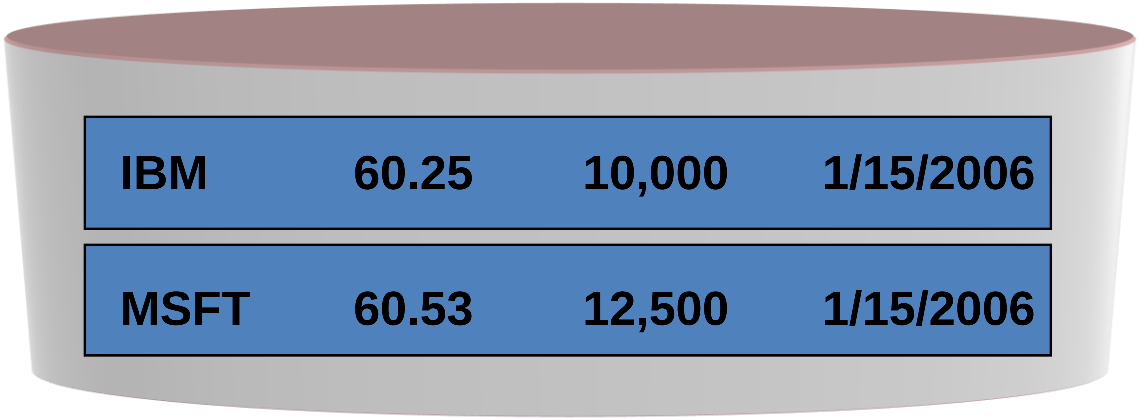
Column Store:



IBM	60.25	10,000	1/15/2006
MSFT	60.53	12,500	1/15/2006

Used in: Sybase IQ, **Vertica**

Row Store:



IBM	60.25	10,000	1/15/2006
MSFT	60.53	12,500	1/15/2006

Used in: Oracle, SQL Server, DB2, Netezza,…

Row Store and Column Store

For example the query

```
SELECT account.account_number,
        sum (usage.toll_airtime),
        sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in (‘AE’. ‘AA’)
AND usage.toll_price > 0
AND source.type != ‘CIBER’
AND toll.rating_method = ‘IS’
AND usage.invoice_date = 20051013
GROUP BY account.account_number
```

Row-store: one row = 212 columns!

Column-store: 7 attributes

Row Store and Column Store

Row Store	Column Store
(+) Easy to add/modify a record	(+) Only need to read in relevant data
(-) Might read in unnecessary data	(-) Tuple writes require multiple accesses

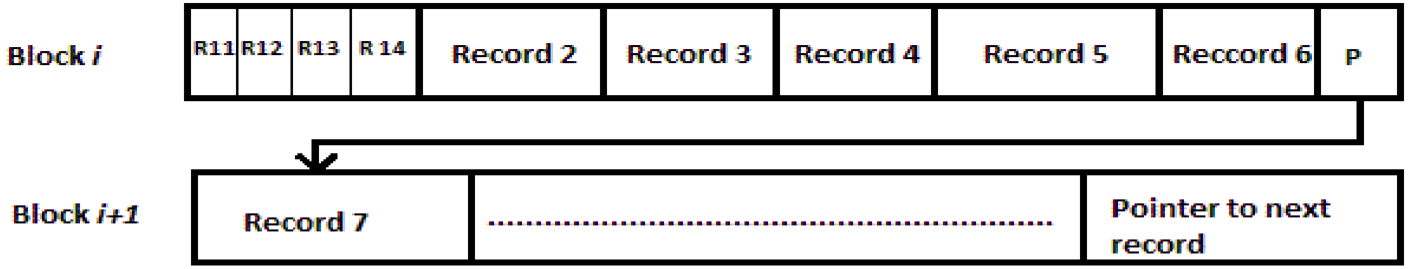
- So column stores are suitable for read-mostly, read-intensive, large data repositories

Columnar Database Systems

- Stores content by columns rather than row.
- The 2-D data represented at conceptual level will be mapped to 1-D data structure at physical level.
- Row-by –Row approach keeps all the information about one entity together.
- Column – by –Column approach keeps all attribute information together.
- Column oriented databases handle fixed length data

RDBMS vs. Columnar Oriented DBMS (Physical Level)

Row oriented

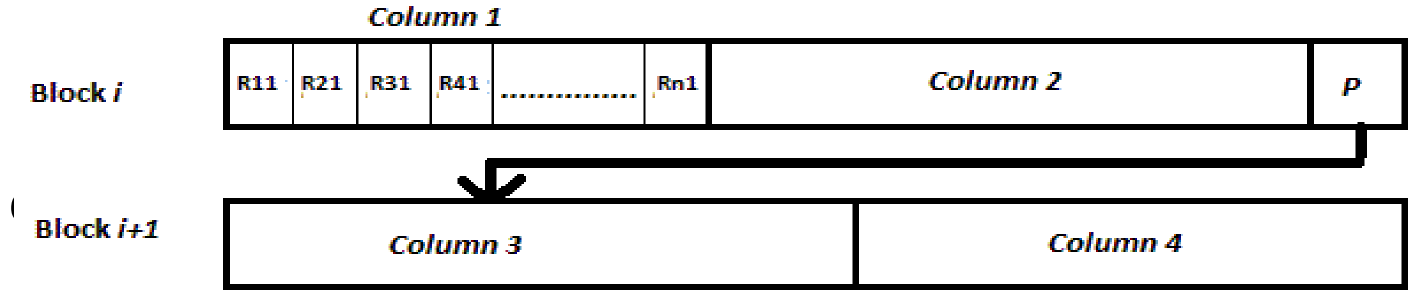


Block *i*

R11 R12 R13 R 14 Record 2 Record 3 Record 4 Record 5 Reccord 6 P

Block *i+1*

Record 7 .... Pointer to next record



Column 1

Block *i*

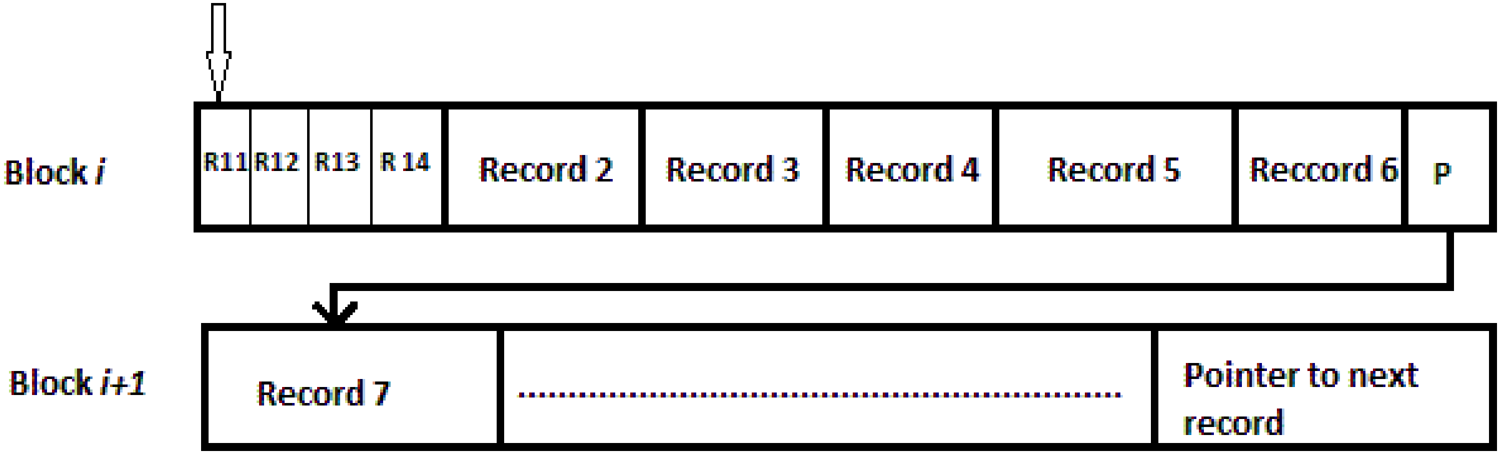
R11 R21 R31 R41 .... Rn1 *Column 2* *P*

Block *i+1*

Column 3 *Column 4*

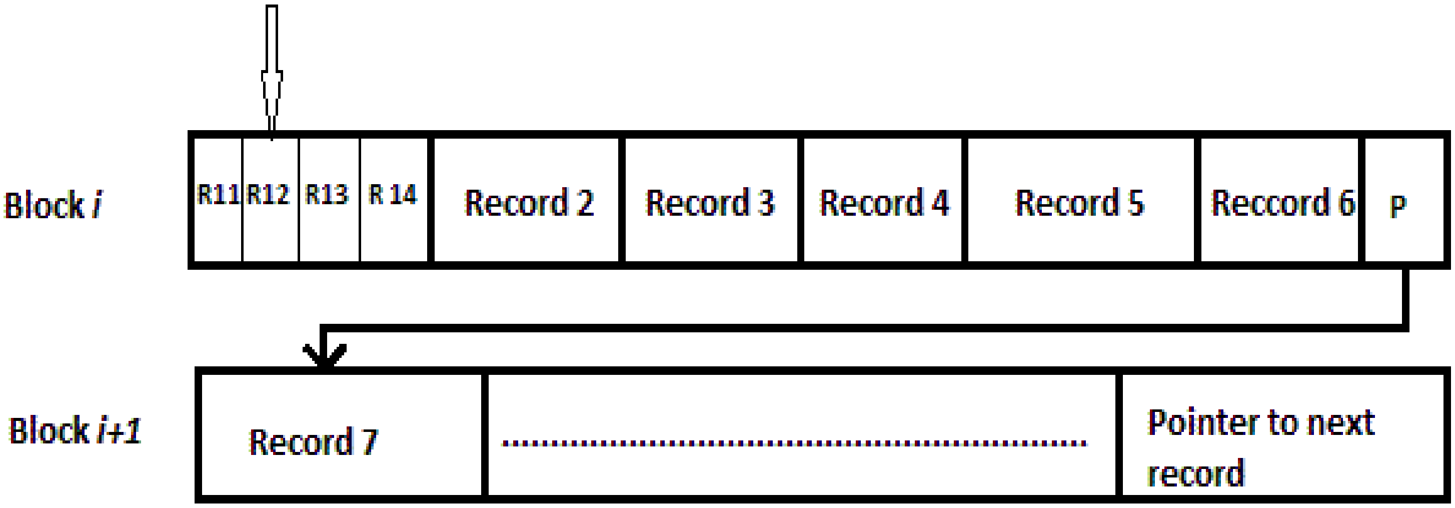
Query Execution (Row oriented)

Select * from Employee_database;



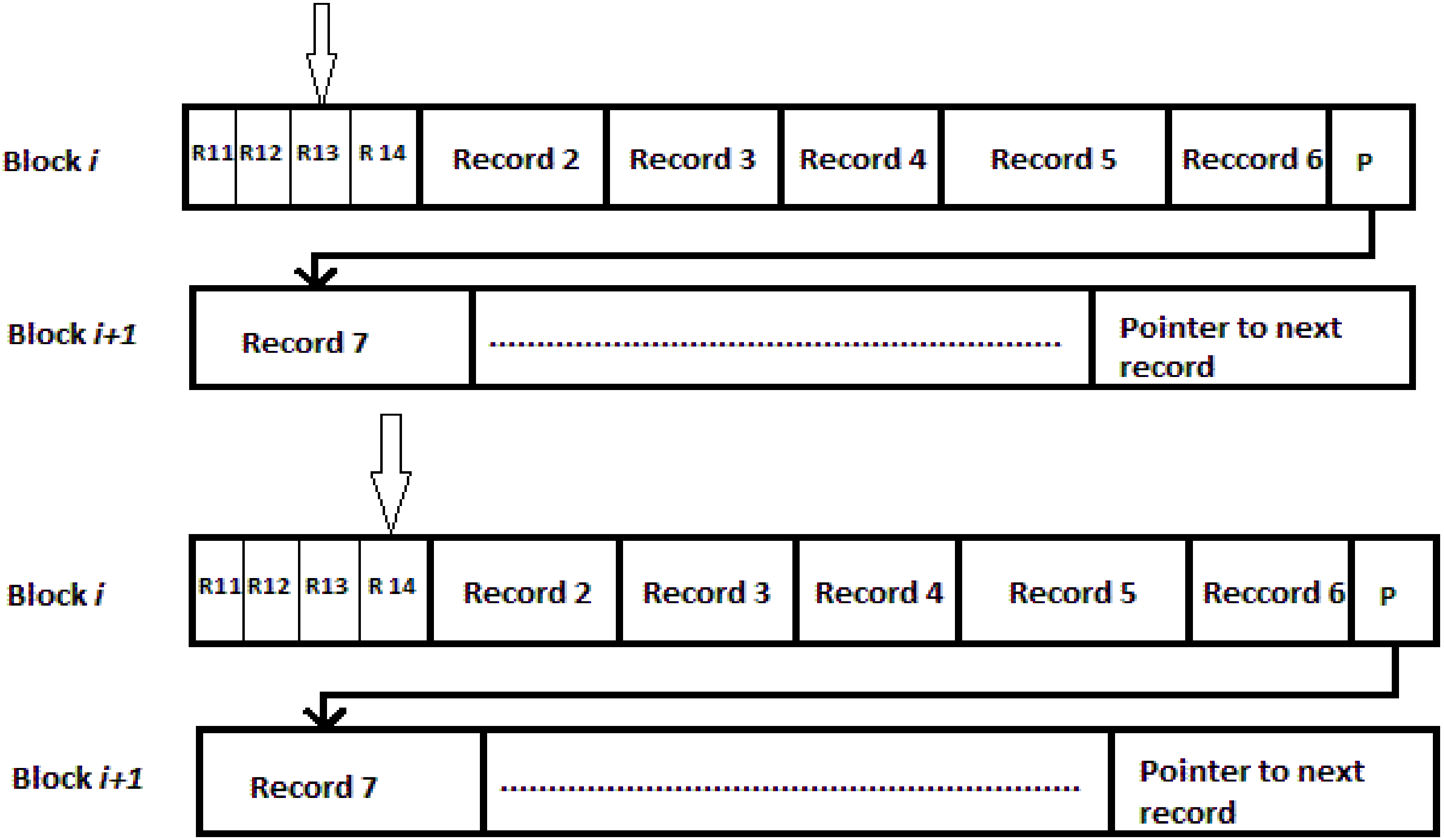
Query Execution

Select * from Employee_database;



Query Execution

Select * from Employee database;

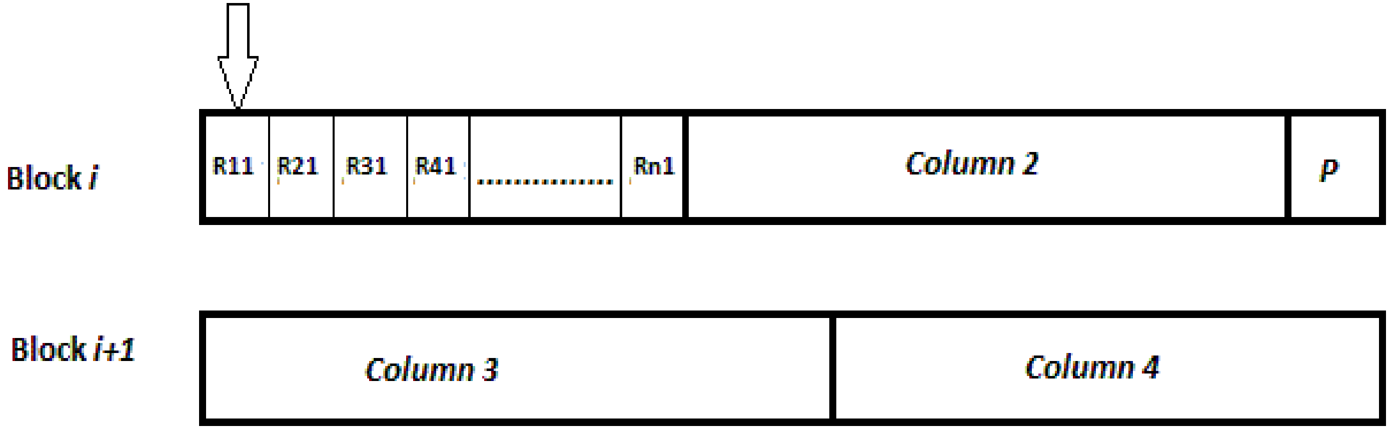


Why Column Oriented Database?

- Most data warehousing applications make more number of reads and lesser number of writes.
- They mostly retrieve and analyze fewer columns compared to the several number of columns that actually exist.
- Row oriented databases have the overhead of seeking through all columns.
- Row oriented data warehouses still persistent.

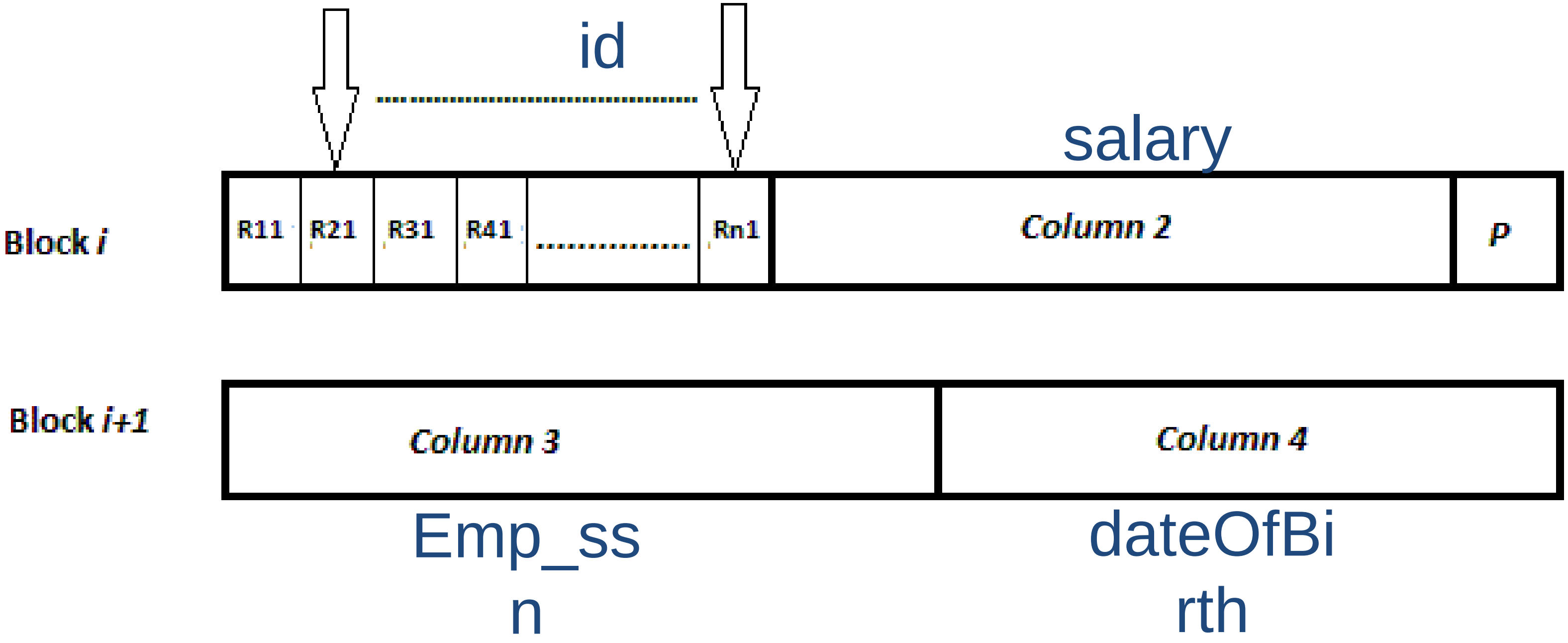
Query Execution (Columnar Databases)

- Select count(E.id) from Employee_Database



Query Execution (Columnar Databases)

- Select count(E.id) from Employee_Database;



Tradeoffs

- Row oriented databases work well for granularity at the entity level.
- Column oriented databases work well for granularity at the attribute level.
- Row oriented – Optimal write time and abundant reading overhead for retrieval of subset queries.
- Column oriented – Optimal read time for subset retrieval queries, bad write performance.

Applications

- Majorly applicable for Data warehouses and Business Intelligence– Required more analytical processing rather transaction processing (Read More and Write Less).
- Online Analytical Processing (At the attribute level)
- Decision making
- Analyzing unorganized BIG DATA with improved granularity
- Data Marts development
- Data Mining
- Latest – Assistance to law enforcement agency, SecureAlert

Data model (Vertica/C-Store)

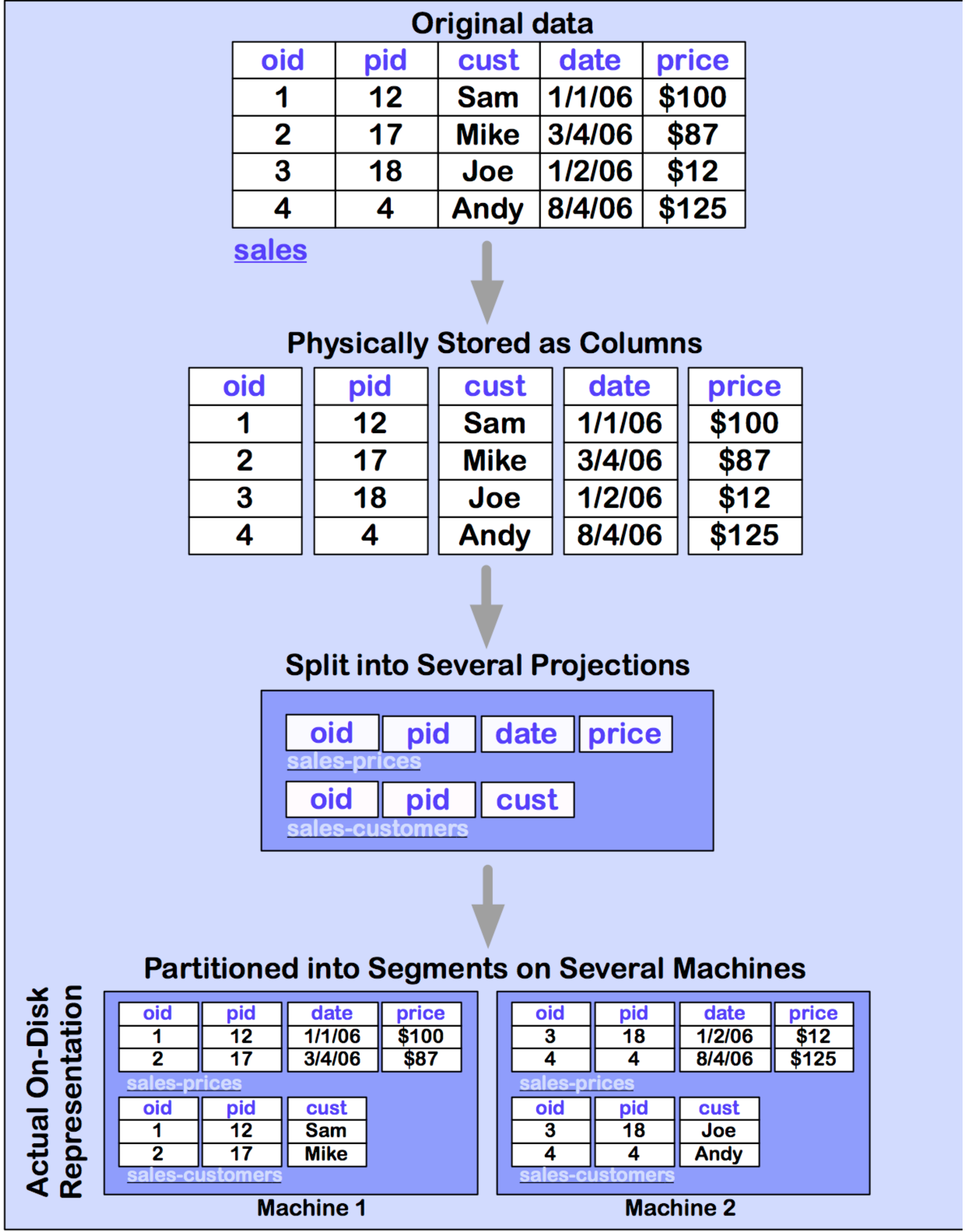
- Same as relational data model
  - Tables, rows, columns
  - Primary keys and foreign keys
  - **Projections**
    - From single table
    - Multiple joined tables
- Example

Normal relational model

EMP(name, age, dept, salary)
DEPT(dname, floor)

Possible C-store model

EMP1 (name, age)
EMP2 (dept, age, DEPT.floor)
EMP3 (name, salary)
DEPT1(dname, floor)



Original data

oid	pid	cust	date	price
1	12	Sam	1/1/06	$100
2	17	Mike	3/4/06	$87
3	18	Joe	1/2/06	$12
4	4	Andy	8/4/06	$125

sales

Physically Stored as Columns

oid
1
2
3
4

pid
12
17
18
4

cust
Sam
Mike
Joe
Andy

date
1/1/06
3/4/06
1/2/06
8/4/06

price
$100
$87
$12
$125

Split into Several Projections

oid	pid	date	price

sales-prices

oid	pid	cust

sales-customers

Partitioned into Segments on Several Machines

Actual On-Disk Representation

oid	pid	date	price
1	12	1/1/06	$100
2	17	3/4/06	$87

sales-prices

oid	pid	cust
1	12	Sam
2	17	Mike

sales-customers

Machine 1

oid	pid	date	price
3	18	1/2/06	$12
4	4	8/4/06	$125

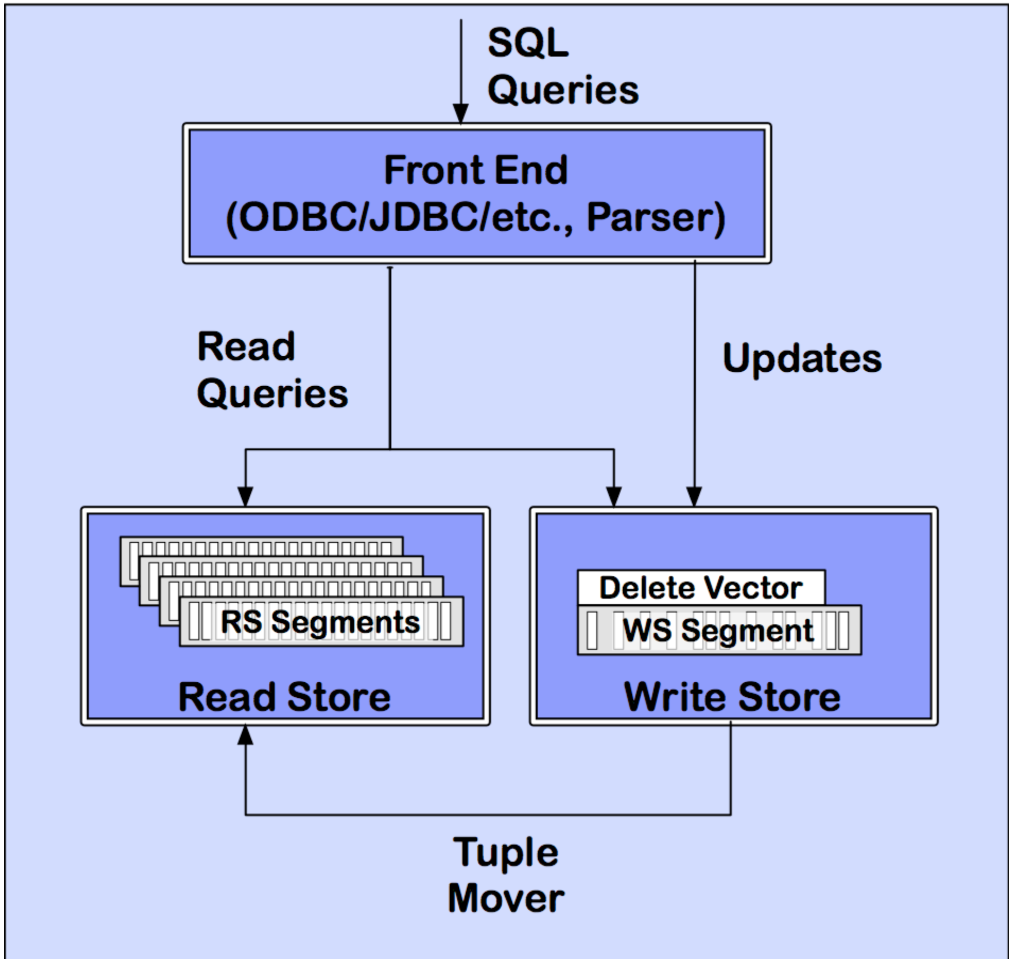
sales-prices

oid	pid	cust
3	18	Joe
4	4	Andy

sales-customers

Machine 2

C-Store/Vertica Architecture (from vertica Technical Overview White Paper)



Compression

- Trades I/O for CPU
  - Increased column-store opportunities:
  - Higher data value locality in column stores
  - Techniques such as run length encoding far more useful

Benefits in query processing

- Selection – has more indices to use
- Projection – some “projections” already defined
- Join – some projections are materialized joins
- Aggregations – works on required columns only

Summary: the performance gain

- Column representation – avoids reads of unused attributes
- Storing overlapping projections – multiple orderings of a column, more choices for query optimization
- Compression of data – more orderings of a column in the same amount of space
- Query operators operate on compressed representation

List of Column Databases

- Vertica/C-Store
- SybaseIQ
- MonetDB
- LucidDB
- HANA
- Google’s Dremel
- Parcell-> Redshit (Another Cloud-DB Service)