Query Execution in Column-Oriented Database Systems

Daniel Abadi

SIGMOD Jim Gray Doctoral Dissertation Award

Research Advisor: Sam Madden

July 2nd, 2009

Acknowledgments

- Mike Stonebraker (C-Store paper)
- Miguel Ferreira (Compression Paper)
- Stavros Harizopoulos (Performance Paper)
- Daniel Myers, David DeWitt (Tuple reconstruction paper)
- Adam Marcus (RDF Paper)
- Nabil Hachem (Column-stores vs row-stores paper)

Acknowledgments

- C-Store Team
  - Brandeis University
    - Mitch Cherniack
    - Nga Tran
    - Adam Batkin
    - Tien Hoang
  - Brown University
    - Stan Zdonik
    - Alexander Rasin
    - Tingjian Ge
  - UMass Boston
    - Pat O'Neil
    - Betty O'Neil
    - Xuedong Chen
  - MIT (people on previous page, plus Amerson Lin, Edmond Lau, and Velen Liang)

Acknowledgments

Sam Madden before serving as my PhD Advisor

Sam Madden after getting me through the PhD program

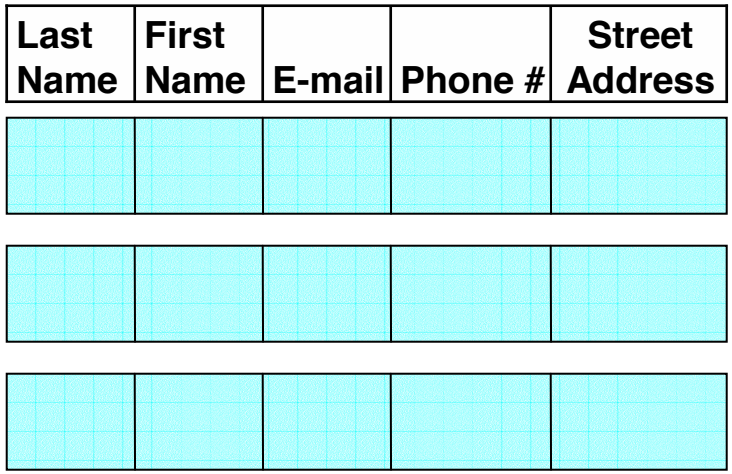
Related Dissertations

- P.A. Boncz. Monet: A Next Generation DBMS Kernel For Query-Intensive Applications (2002). PhD Advisor: Martin Kersten.

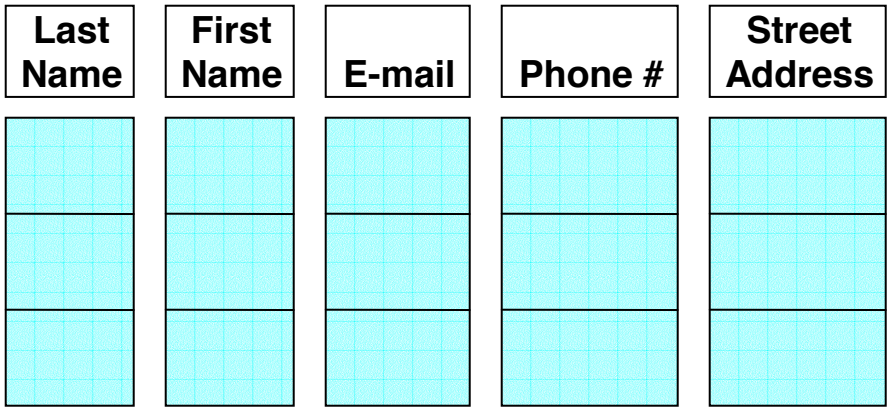
- Coming soon:
  - Marcin Zukowski (CWI)
  - Stratos Idreos (CWI)

Row vs. Column-Stores

Row Store



Column Store



+ Easy to add a new record
– Might read in unnecessary data

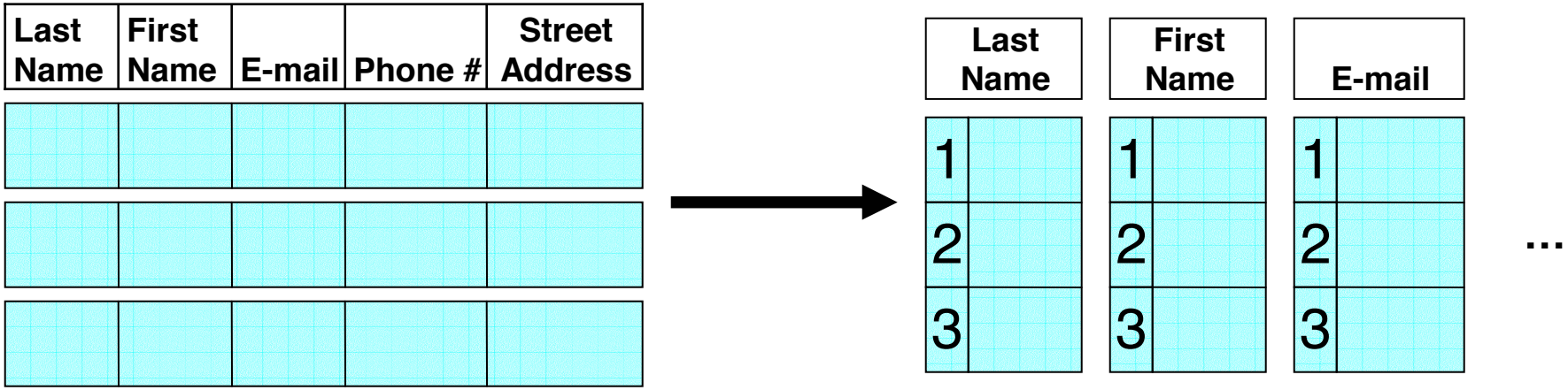
+ Only need to read in relevant attributes
– Tuple writes might require multiple seeks

Column-Stores

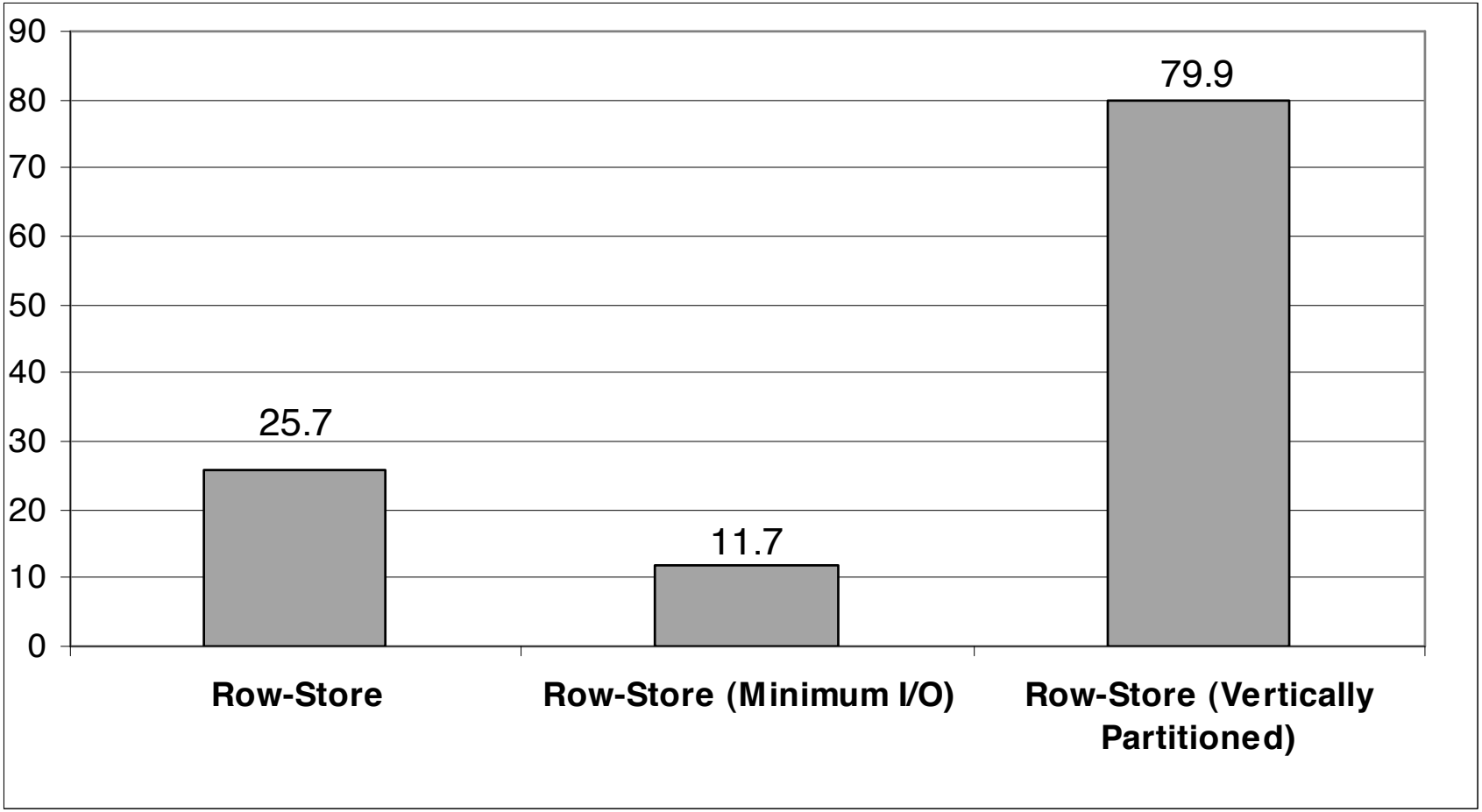
- Really good for read-mostly data warehouses
  - Lot's of column scans and aggregations
  - Writes tend to be in batch

Column-Store History

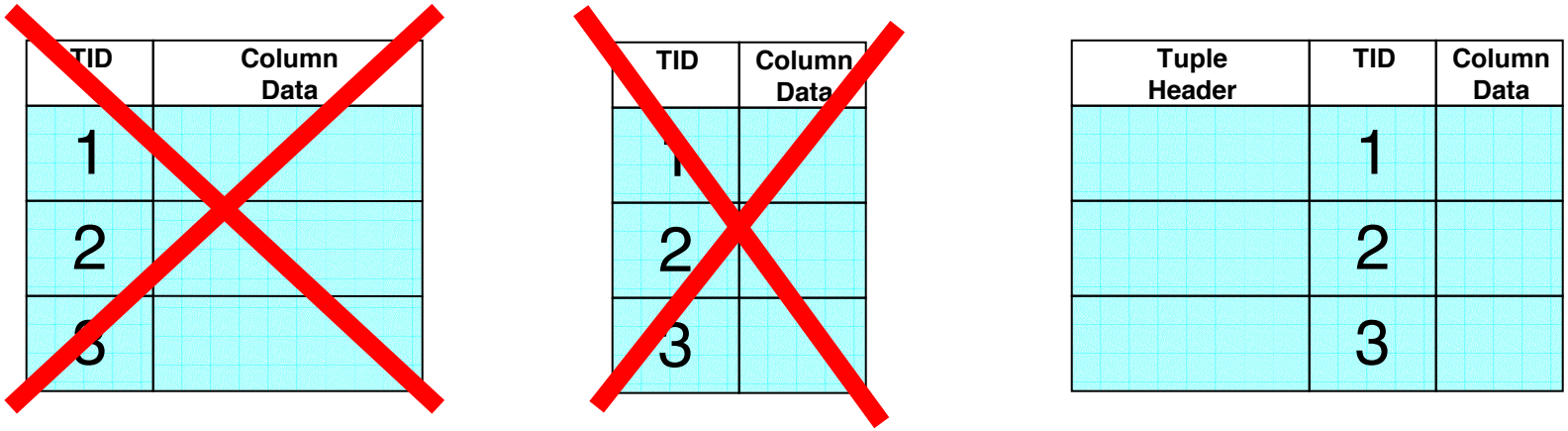
1970s/80s: Decomposed Storage Model (DSM)/Vertical Partitioning



SSBM Averages



Tuple Size



- Queries touch 3 - 4 foreign keys in fact table, 1- 2 numeric columns

Complete fact table takes up ~4 GB (compressed)

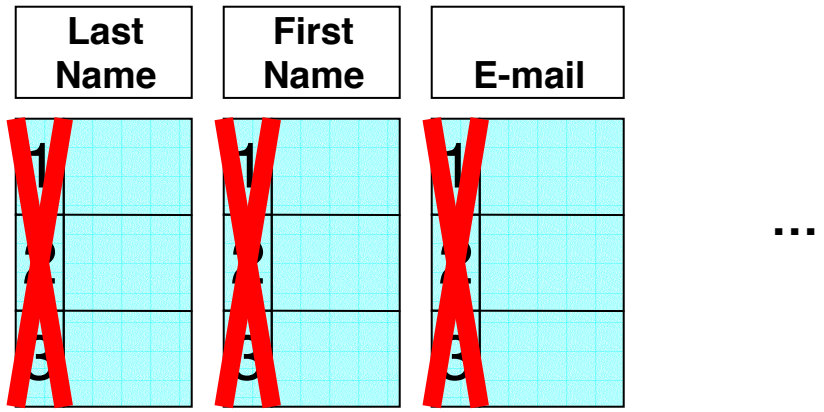
Vertically partitioned tables take up 0.7 - 1.1 GB (compressed)

Query Optimizer

- Each column is a separate table
- Each attribute accessed in a query becomes a join
- Optimizer becomes hopelessly overwhelmed

Tuple ID problem easy to solve

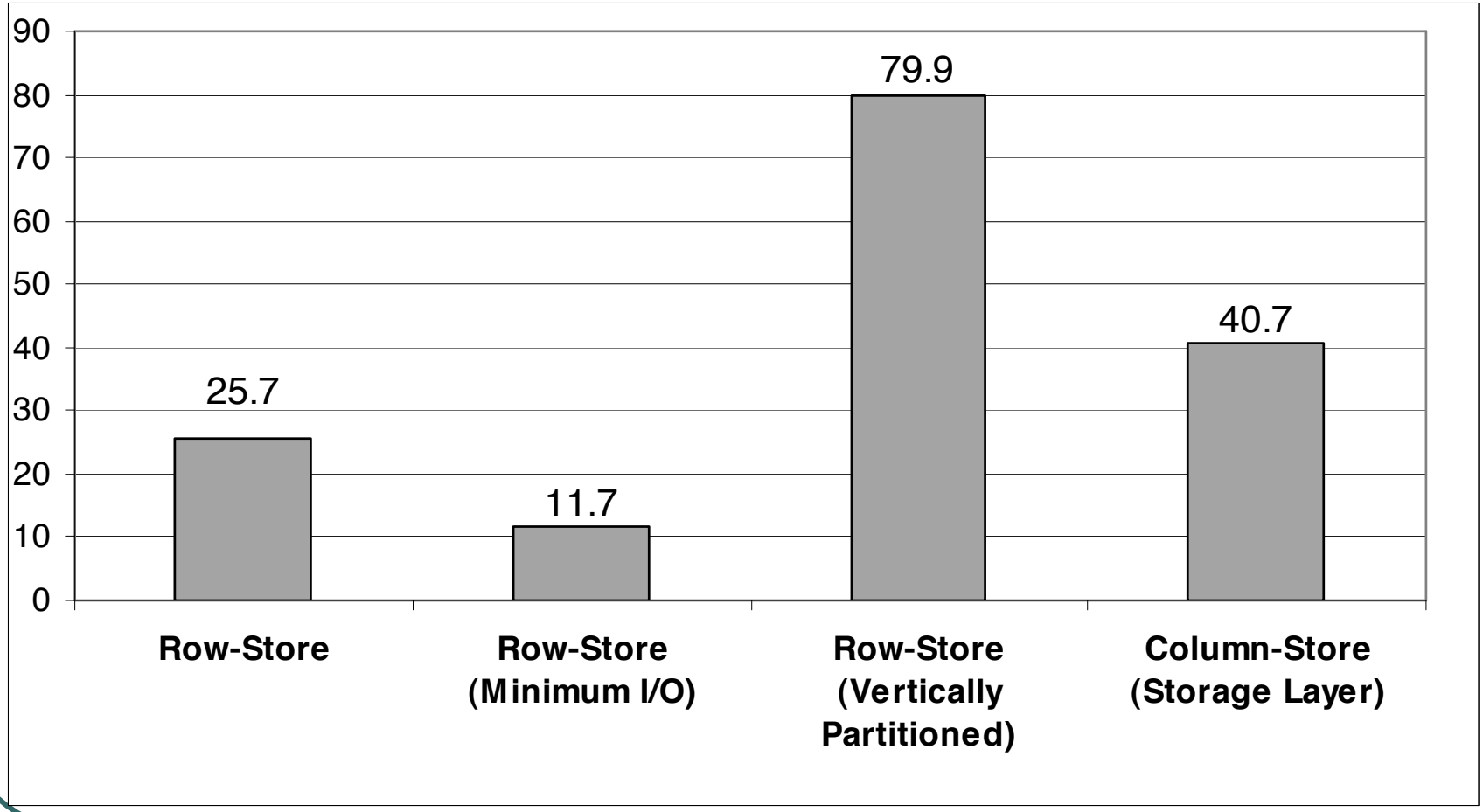
- Store columns in original tuple order, remove Tuple IDs
- Store tuple header in separate column



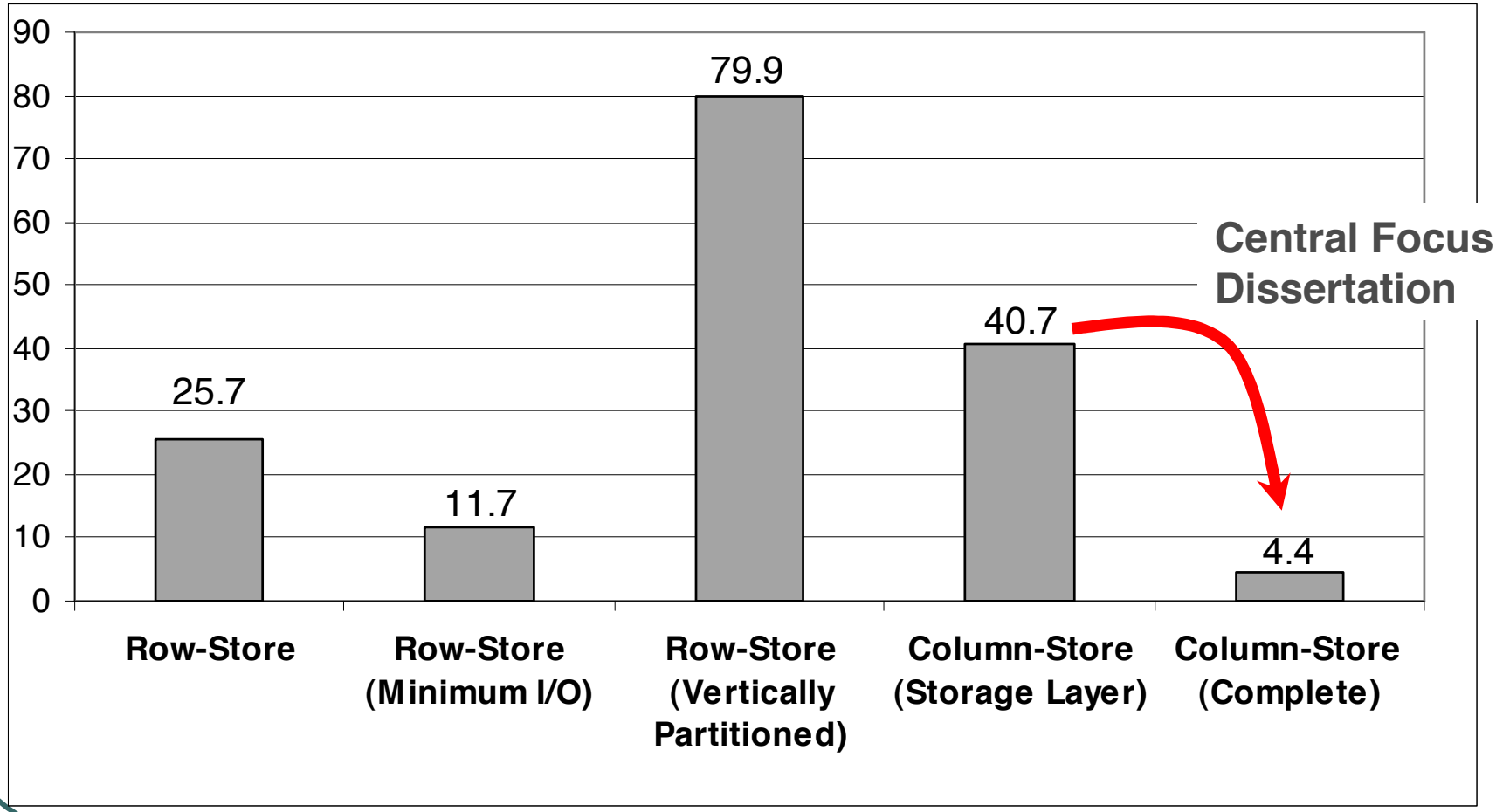
Heuristics To Solve Optimizer Problem

- For each query, figure out what attributes will be accessed
- For each table, first join all accessed attributes into normal row-store tuples for that table
- Then proceed with regular row-store optimization and execution

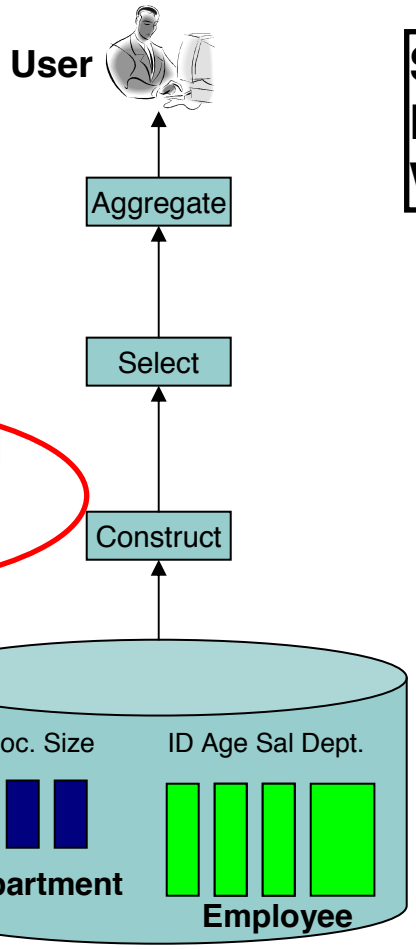
Result After These Simple Fixes



Final Result After Dissertation



Overview Of Dissertation




```
SELECT AVG(salary)
FROM Employee
WHERE Age > 50
```

3 Carefully Placed Construction Operators

4 Benchmark Performance on SSBM

5 Benchmark Performance on RDF

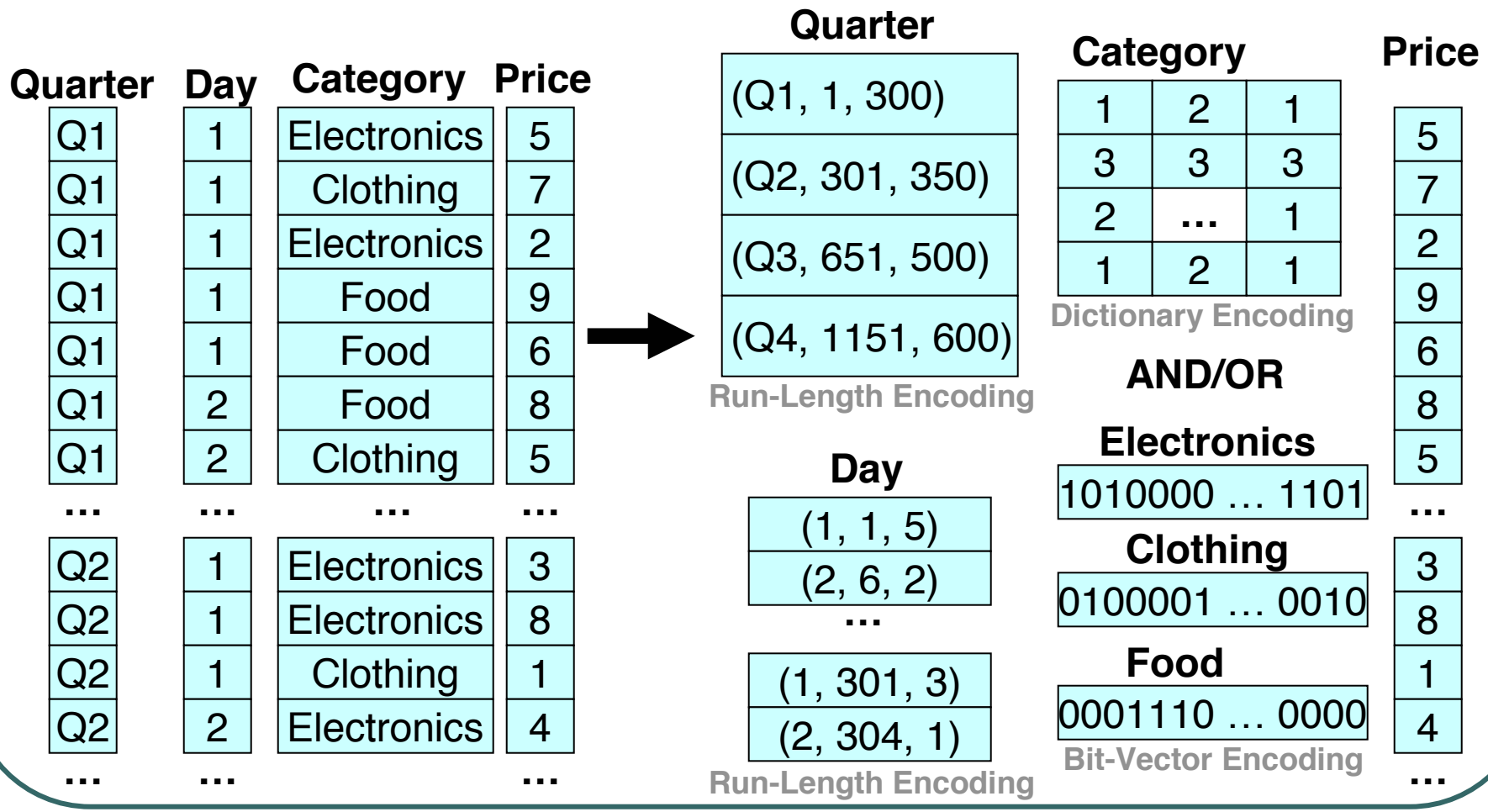
2 Build Compression into Executor

1 Architecture: Storage Layer, Data Flow, High Performance Data Access and Iteration

Compression

- **Trades I/O for CPU**
- **Increased column-store opportunities:**
  - **Higher data value locality in column stores**
  - **More Techniques Useful**
  - **Can use extra space to store multiple copies of data in different sort orders**

Column-Oriented Compression



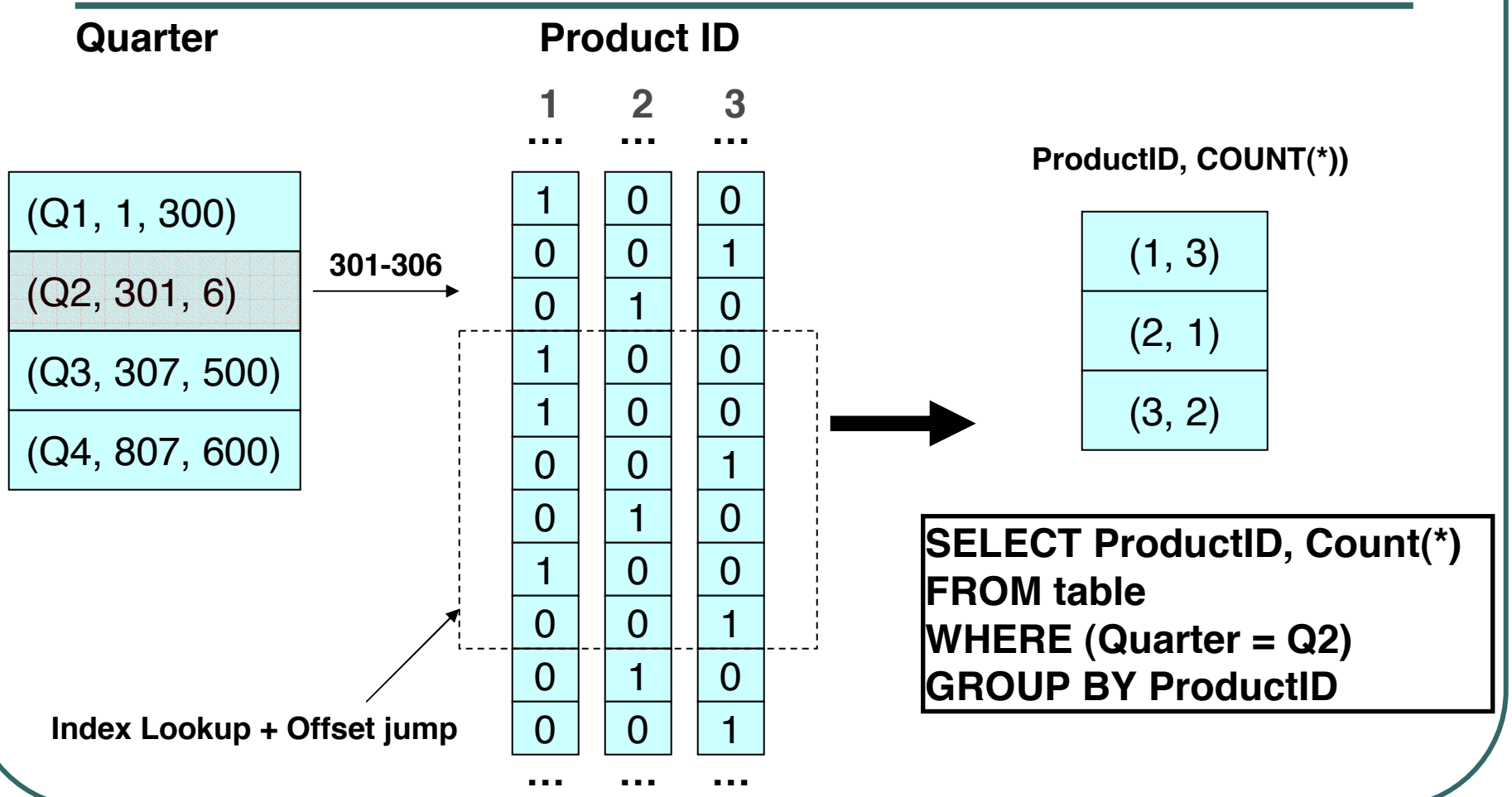
Other Compression Techniques

- **Null Suppression**
- **Differential or Frame of Reference**
- **Huffman**
- **Arithmetic**
- **Lempel-Ziv**

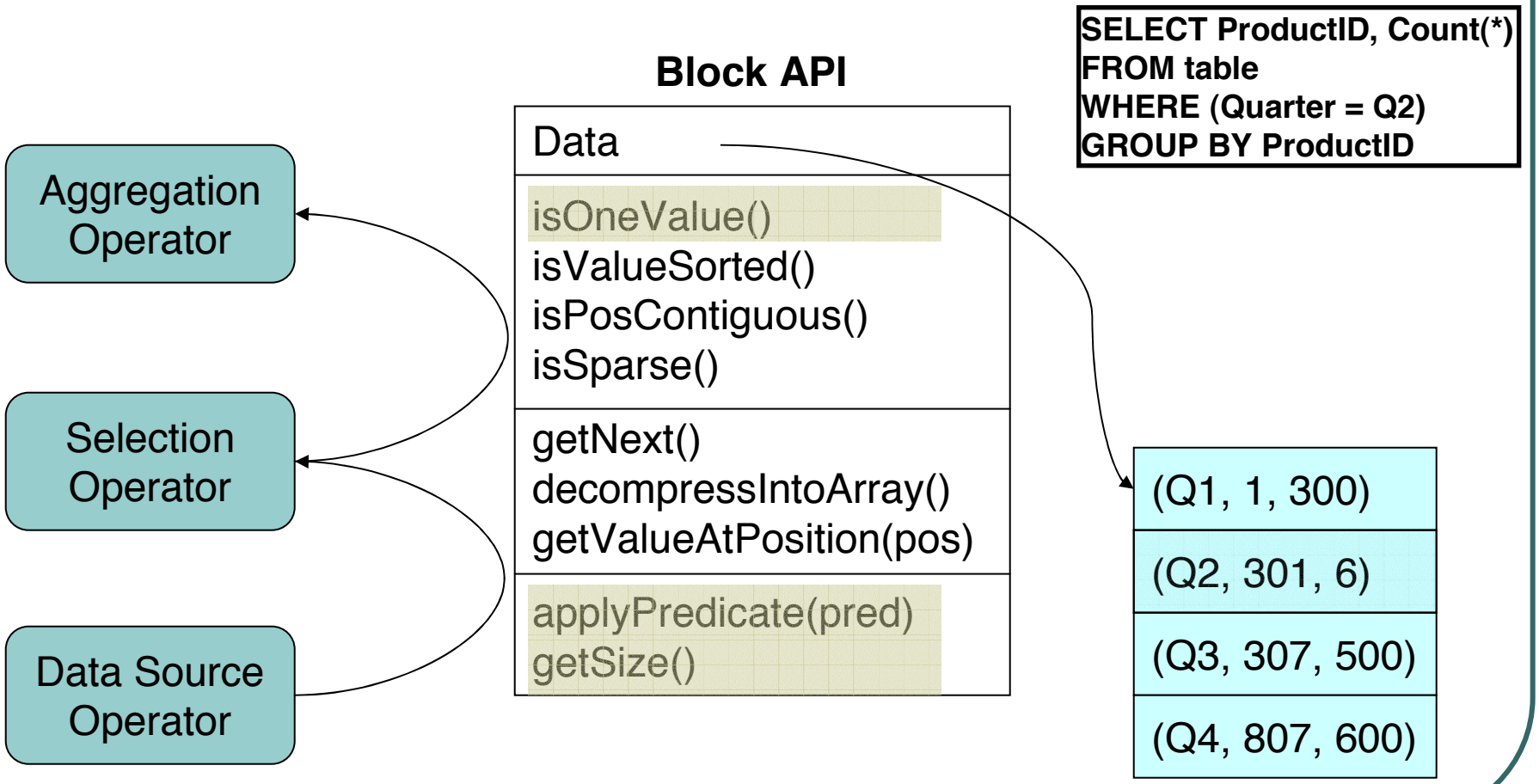
Operating Directly on Compressed Data

- **I/O - CPU tradeoff is no longer a tradeoff**
- **Reduces memory–CPU bandwidth requirements**
- **Opens up possibility of operating on multiple records at once**

Operating Directly on Compressed Data



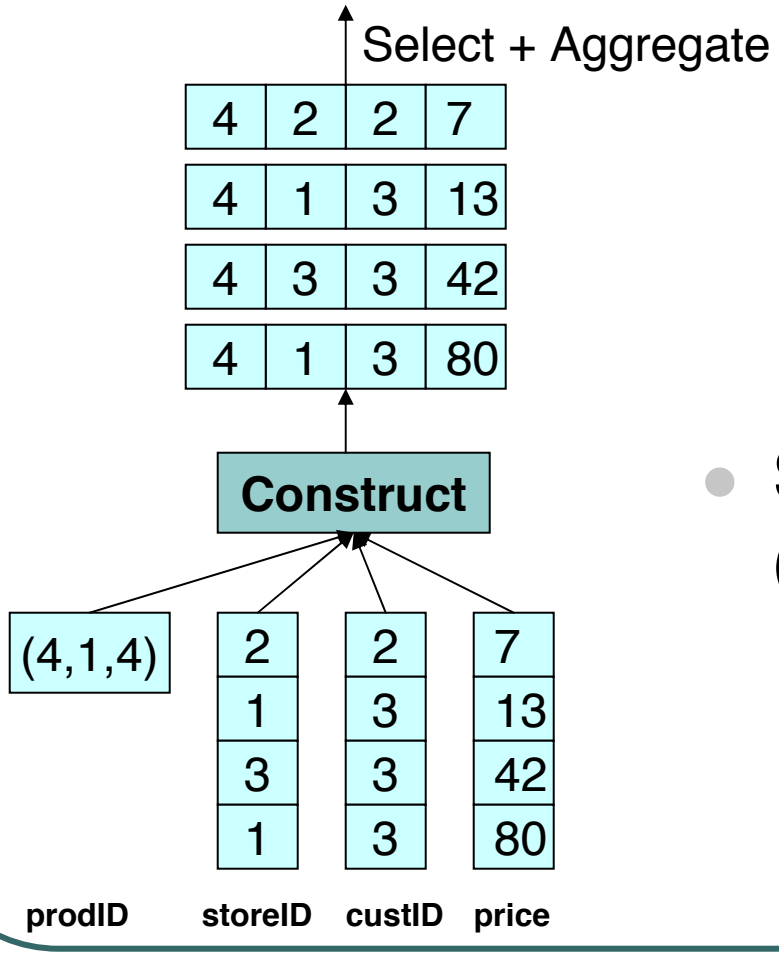
Operating Directly on Compressed Data



When should tuples be constructed?

- **At some point in query plan, tuples must be constructed, "materialization"**
  - **Early materialization = create rows at beginning of query plan**
  - **Late materialization = operate on columns as long as possible**

When should tuples be constructed?

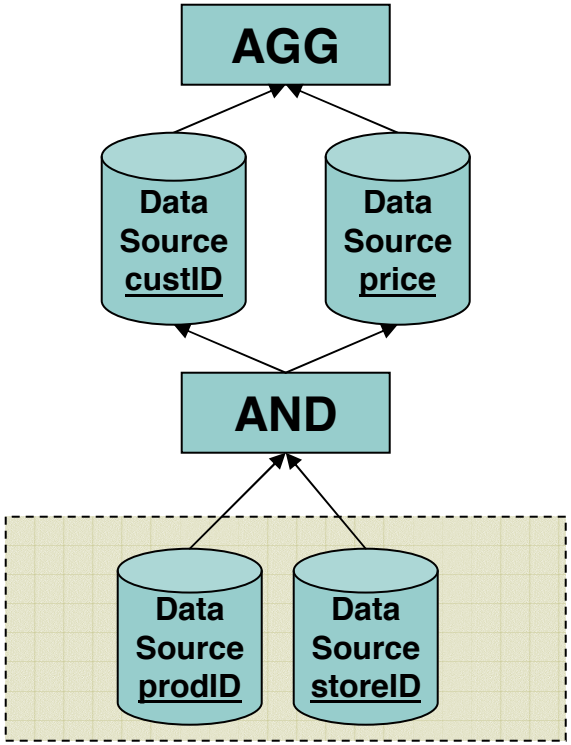


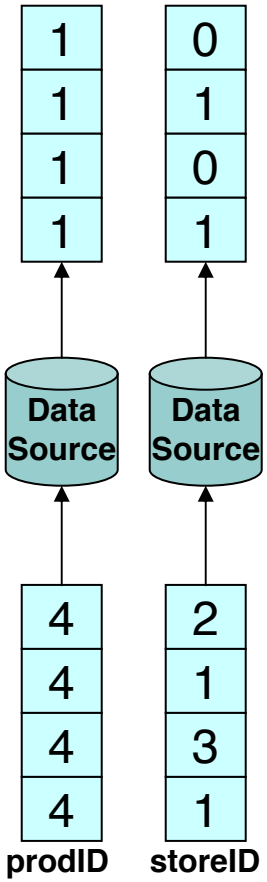
QUERY:

```
SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

- **Solution 1: Create rows first (EM). But:**
  - **Need to construct ALL tuples**
  - **Need to decompress data**
  - **Poor memory bandwidth utilization**

Solution 2: Operate on columns



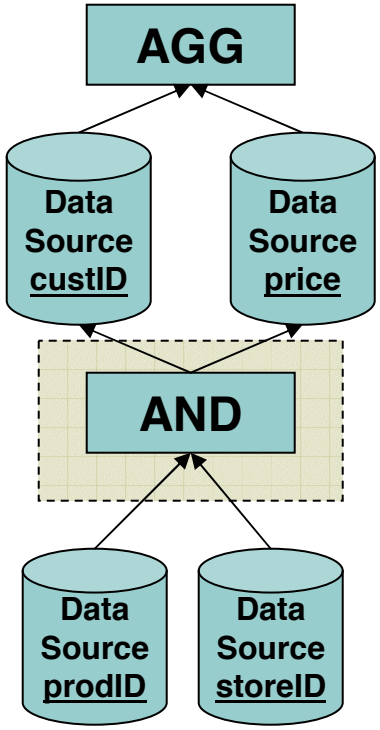


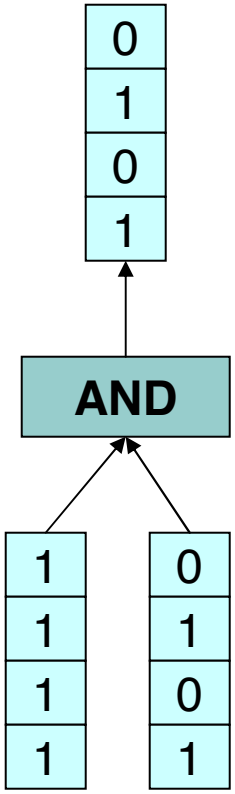
QUERY:

```
SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

Solution 2: Operate on columns



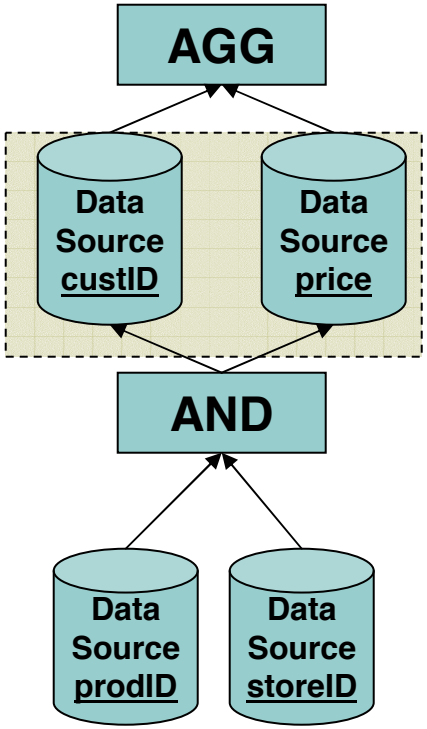


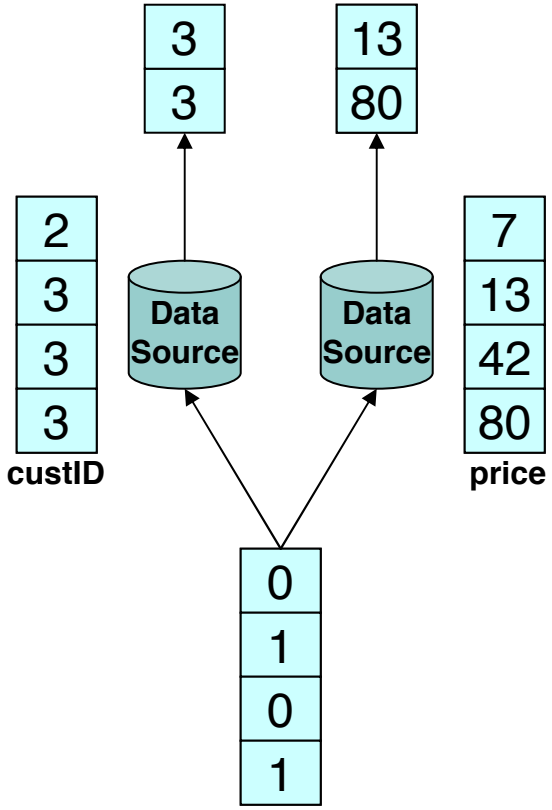
QUERY:

```
SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

Solution 2: Operate on columns



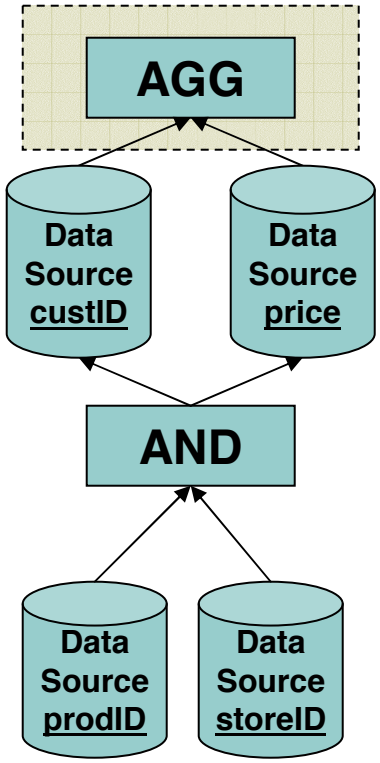


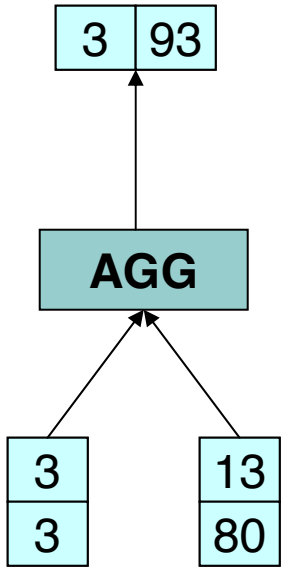
QUERY:

```
SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

Solution 2: Operate on columns





QUERY:

```
SELECT custID,SUM(price)
FROM table
WHERE (prodID = 4) AND
      (storeID = 1) AND
GROUP BY custID
```

prodID	storeID	custID	price
4	2	2	7
4	1	3	13
4	3	3	42
4	1	3	80

Materialization Chapter

- **Describes how to implement LM and EM plans in a column-store (C-Store)**
- **Explores the tradeoffs between the two types of plans using:**
  - **An analytical model**
  - **Experimental results**
- **Presents heuristics to help a plan generator choose between EM and LM plans**

Summary of Contributions

- Introduced a query executer that:
  - Is optimized for column-oriented data layout
  - Designed for compressed data
  - Constructs tuples at the right time
- Bunch of other innovations
  - Invisible Join
  - Optional block processing
  - Pseudo-iterative execution pipelining
- Orders of magnitude performance gains
  - Factor of 9-10 faster than naïve column-store
  - 1-2 orders of magnitude faster than row-store

For More, See:

- C-Store: A Column-Oriented DBMS (VLDB 2005)
- Integrating Compression and Execution in Column-Oriented Database Systems (SIGMOD 2006)
- Performance Tradeoffs in Read-Optimized Databases (VLDB 2006)
- Materialization Strategies in a Column-Oriented DBMS (ICDE 2007)
- Column Stores for Wide and Sparse Data (CIDR 2007)
- Scalable Semantic Web Data Management Using Vertical Partitioning (VLDB 2007)
- Column-Stores vs. Row-Stores: How Different Are They Really? (SIGMOD 2008)
- SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management (VLDBJ 2009)

Come Join the Yale DB Group!

More Plugs

- Blog:
  - http://dbmsmusings.blogspot.com/
- Twitter:
  - @abadid
- New Research Project:
  - HadoopDB (see VLDB paper)