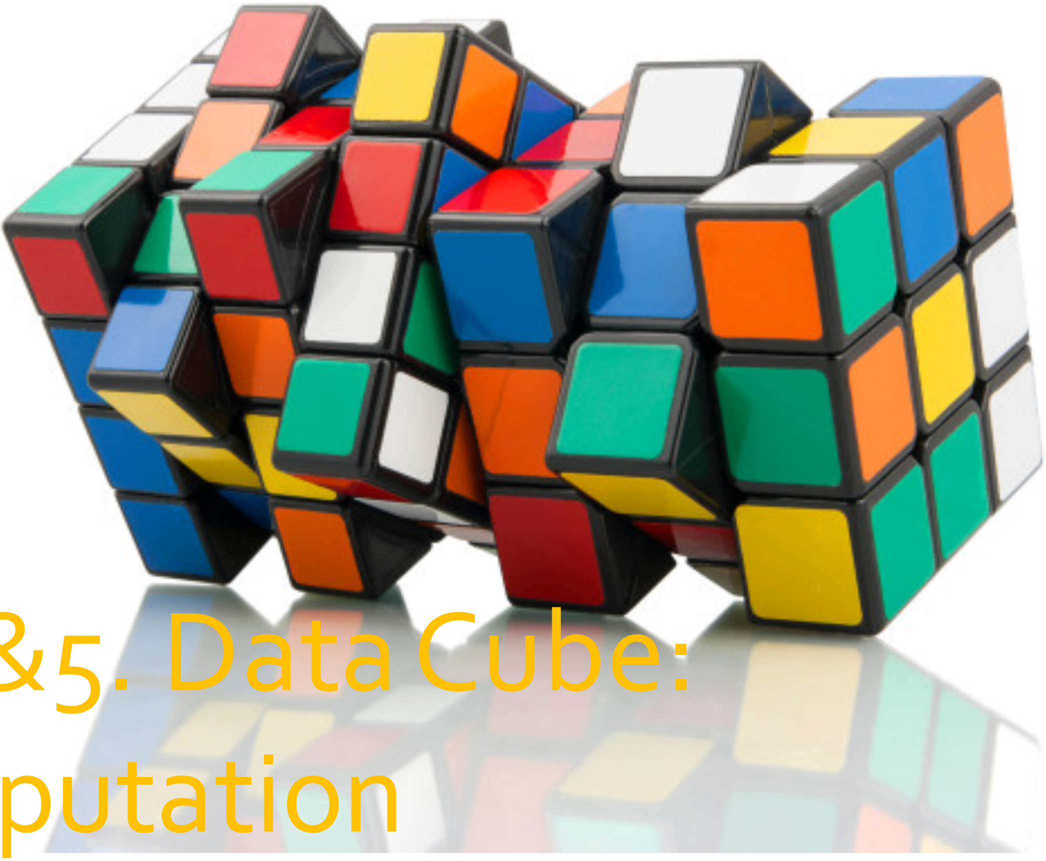


Chapter 4&5. Data Cube: Cube Computation

Meng Jiang

CSE 40647/60647 Data Science Fall 2017

Introduction to Data Mining

Data Cube Technology

- **Data Cube Computation: Basic Concepts**
- Data Cube Computation Methods

Efficient Data Cube Computation

- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell
  - How many cuboids in an n-dimensional cube with L_i levels?
- Materialization of data cube
  - **Full materialization**: Materialize every (cuboid)
  - **No materialization**: Materialize none (cuboid)
  - **Partial materialization**: Materialize some cuboids
    - Which cuboids to materialize?
      - Selection based on size, sharing, access frequency, etc.

Q: What do they dislike the most?

Iceberg

Cube Materialization: Full Cube vs. Iceberg Cube

- Full cube vs. iceberg cube

  compute cube sales iceberg as

  select date, product, city, department, count(*)

  from salesInfo

  cube by date, product, city

  having count(*) >= min support
- Compute *only* the **cells** whose **measure** satisfies the iceberg condition
- Only a small portion of cells may be "above the water" in a sparse cube
- Ex.: Show only those cells whose **count** is no less than 100

Why Iceberg Cube?

- Advantages of computing iceberg cubes
  - No need to save nor show those cells whose value is below the threshold (iceberg condition)
  - Efficient methods may even avoid computing the un-needed, intermediate cells
  - Avoid explosive growth
- Example: A cube with 100 dimensions
  - Suppose it contains only 2 base cells: $\{(a_1, a_2, a_3, \ldots, a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$
  - How many aggregate cells if "having count >= 1" ("non-empty"?
    - Answer: $(2^{101} - 2) - 4$

Suppose it contains only 2 base cells: $\{(a_1, a_2, a_3, \ldots., a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$

How many aggregate cells if "having count >= 1"?

For $\{(a_1, a_2, a_3 \ldots, a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$, the total # of non-base cells should be 2 * (2^{100} – 1) – 4.

This is calculated as follows:

- (a1, a2, a3 . . . , a100) will generate 2^{100} - 1 non-base cells
- (a1, a2, b3, . . . , b100) will generate 2^{100} - 1 non-base cells

Among these, 4 cells are overlapped and thus minus 4 so we get:

2*2^{100} - 2 – 4

These 4 cells are:

- (a1, a2, *, …, *): 2
- (a1, *, *, …, *): 2
- (*, a2, *, …, *): 2
- (*, *, *, …, *): 2

Why Iceberg Cube?

- Advantages of computing iceberg cubes
  - No need to save nor show those cells whose value is below the threshold (iceberg condition)
  - Efficient methods may even avoid computing the un-needed, intermediate cells
  - Avoid explosive growth
- Example: A cube with 100 dimensions
  - Suppose it contains only 2 base cells: $\{(a_1, a_2, a_3, \ldots, a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$
  - How many aggregate cells if "having count >= 1" ("non-empty"?
    - Answer: $(2^{101} - 2) - 4$
  - What are the iceberg cells, (i.e., with condition: "having count >= 2")?
    - Answer: 4

Suppose it contains only 2 base cells: $\{(a_1, a_2, a_3, \ldots., a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$

How many iceberg cells if "having count >= 2"?

For $\{(a_1, a_2, a_3 \ldots, a_{100}), (a_1, a_2, b_3, \ldots, b_{100})\}$, the total # of non-base cells should be 2 * (2^{100} – 1) – 4.

This is calculated as follows:

- (a1, a2, a3 . . . , a100) will generate 2^{100} - 1 non-base cells
- (a1, a2, b3, . . . , b100) will generate 2^{100} - 1 non-base cells

Among these, 4 cells are overlapped and thus minus 4 so we get:

2*2^{100} - 2 – 4

These 4 cells are:

- (a1, a2, *, …, *): 2
- (a1, *, *, …, *): 2
- (*, a2, *, …, *): 2
- (*, *, *, …, *): 2

Is Iceberg Cube Good Enough? Closed Cube & Cube Shell

- Let cube P have only 2 base cells: $\{(a_1, a_2, a_3 \ldots, a_{100}):10, (a_1, a_2, b_3, \ldots, b_{100}):10\}$
  - How many cells will the iceberg cube contain if "having count(*) ≥ 10"?
    - Answer: $2^{101} - 4$ (still too big!)
- **Close cube:**
  - A cell c is ***closed*** if there exists no cell *d*, such that *d* is a descendant of *c*, and *d* has the same measure value as *c*
    - Ex. The same cube P has only 3 closed cells:
  - $\{(a_1, a_2, *, \ldots, *): 20, (a_1, a_2, a_3 \ldots, a_{100}): 10, (a_1, a_2, b_3, \ldots, b_{100}): 10\}$
  - A ***closed cube*** is a cube consisting of only closed cells

Data Cube Technology

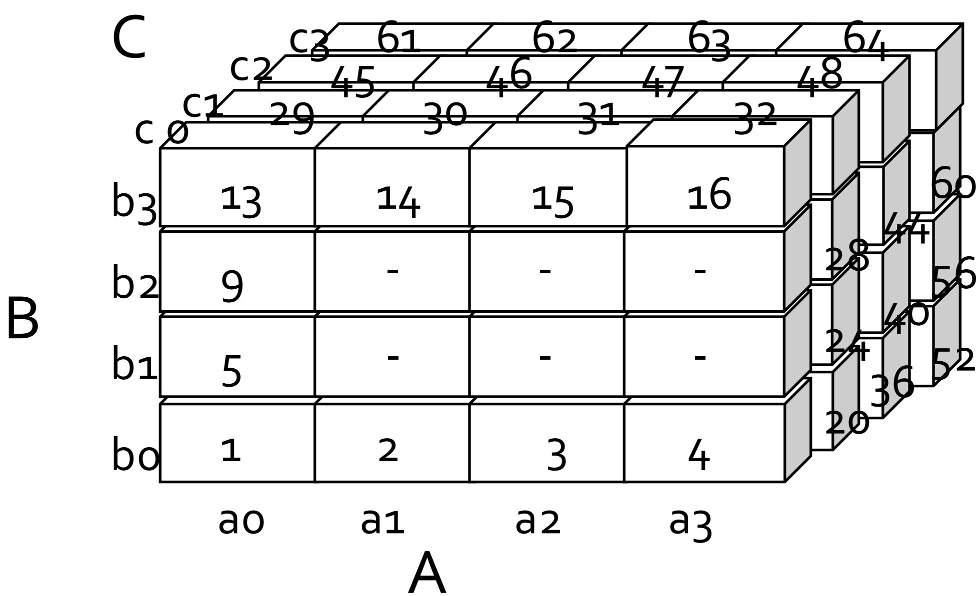
- Data Cube Computation: Basic Concepts
- **Data Cube Computation Methods**

Roadmap for Efficient Computation

- General computation heuristics (Agarwal et al.'96)
- Computing full/iceberg cubes: 3 methodologies
  - Bottom-Up:
    - **Multi-way array aggregation** (Zhao, Deshpande & Naughton, SIGMOD'97)
  - Top-down:
    - BUC (Beyer & Ramarkrishnan, SIGMOD'99)
  - Integrating Top-Down and Bottom-Up:
    - Star-cubing algorithm (Xin, Han, Li & Wah: VLDB'03)
- High-dimensional OLAP:
  - A shell-fragment approach (Li, et al. VLDB'04)
- Computing alternative kinds of cubes:
  - Partial cube, closed cube, approximate cube, ...

Multi-Way Array Aggregation

- Bottom-up: Partition a huge *sparse* array into *chunks* (a small subcube which fits in memory) and aggregation.
- Data addressing: Compressed *sparse array addressing* (chunk_id, offset)
- Compute **aggregates in "multiway"** by visiting cube cells in the order which **minimizes** the # of times to visit each cell, and **reduces** memory access and storage cost



What is the best traversing order to do multi-way aggregation?

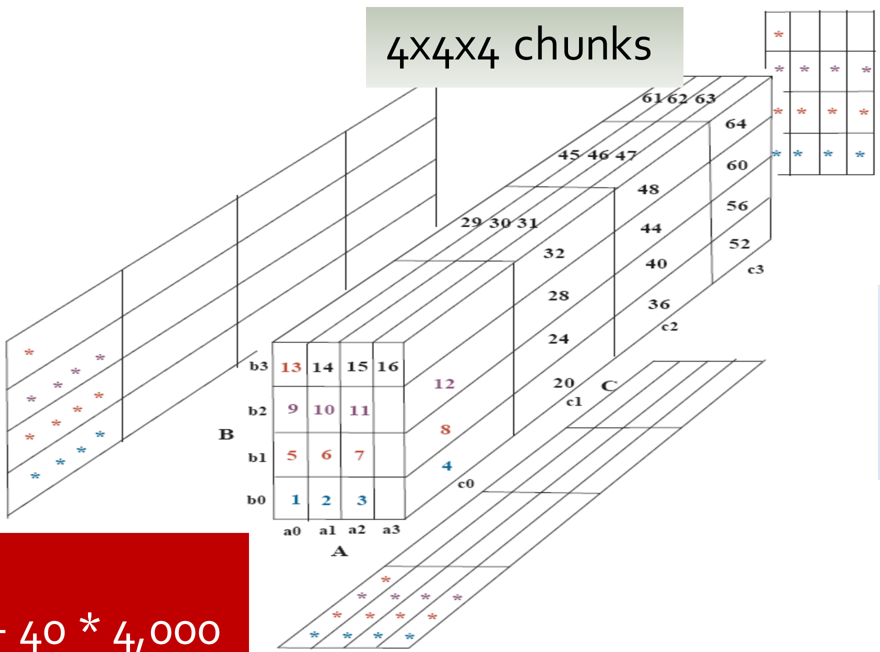
ABC → AB, BC and AC

A: 40 (location),
B: 400 (item),
C: 4,000 (time)

Multi-way Array Aggregation (3-D to 2-D)

- How much **memory cost of computation (aggregation for AB, AC, BC planes)** can we save?

4x4x4 chunks



A: 40 (location),
B: 400 (item),
C: 4,000 (time)

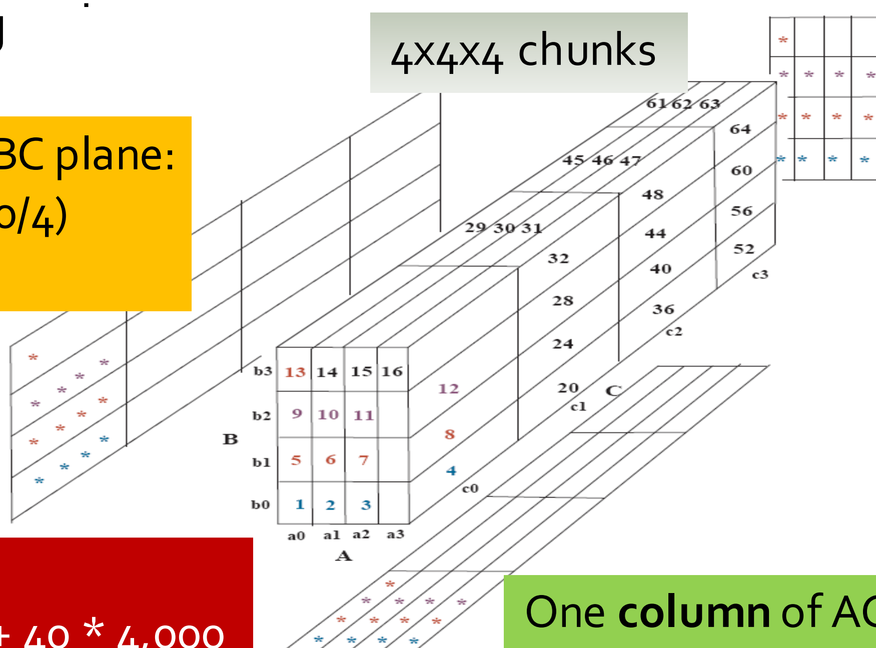
Min memory size: ?
< 40*400 + 400*4,000 + 40 * 4,000
= 1,776,000

Multi-way Array Aggregation (3-D to 2-D)

- How to minimizes the memory requirement and reduced I/Os?
  - Keep the **smallest** plane in **main memory**
  - Fetch and compute **only one chunk** at a time for the **largest** plane
  - The planes should be **sorted** and computed according to their **size** in ascending

4x4x4 chunks

One **chunk** of BC plane:
(400/4) * (4,000/4)
= 100,000



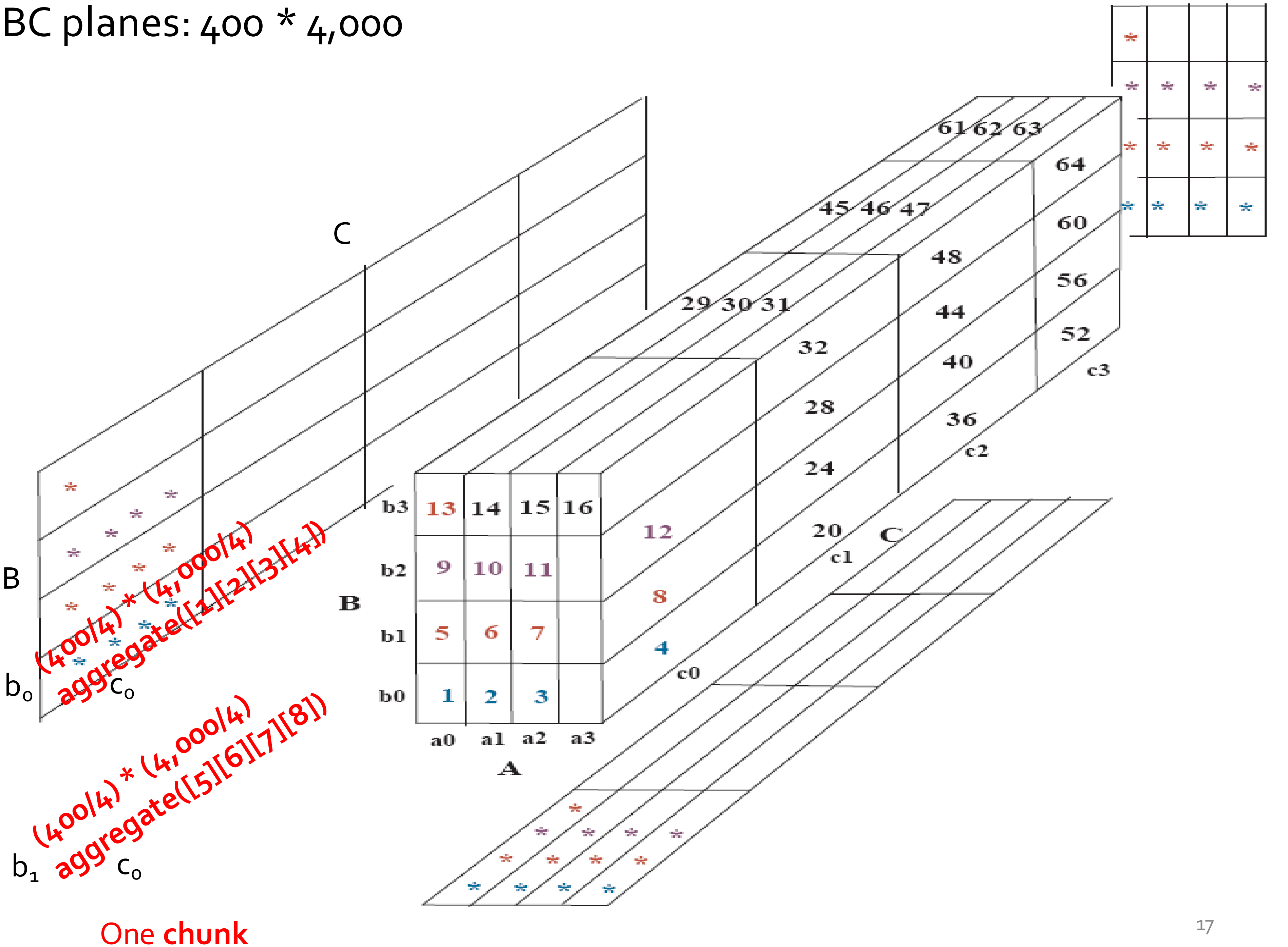
Entire AB plane:
40 * 400
= 16,000

A: 40 (location),
B: 400 (item),
C: 4,000 (time)

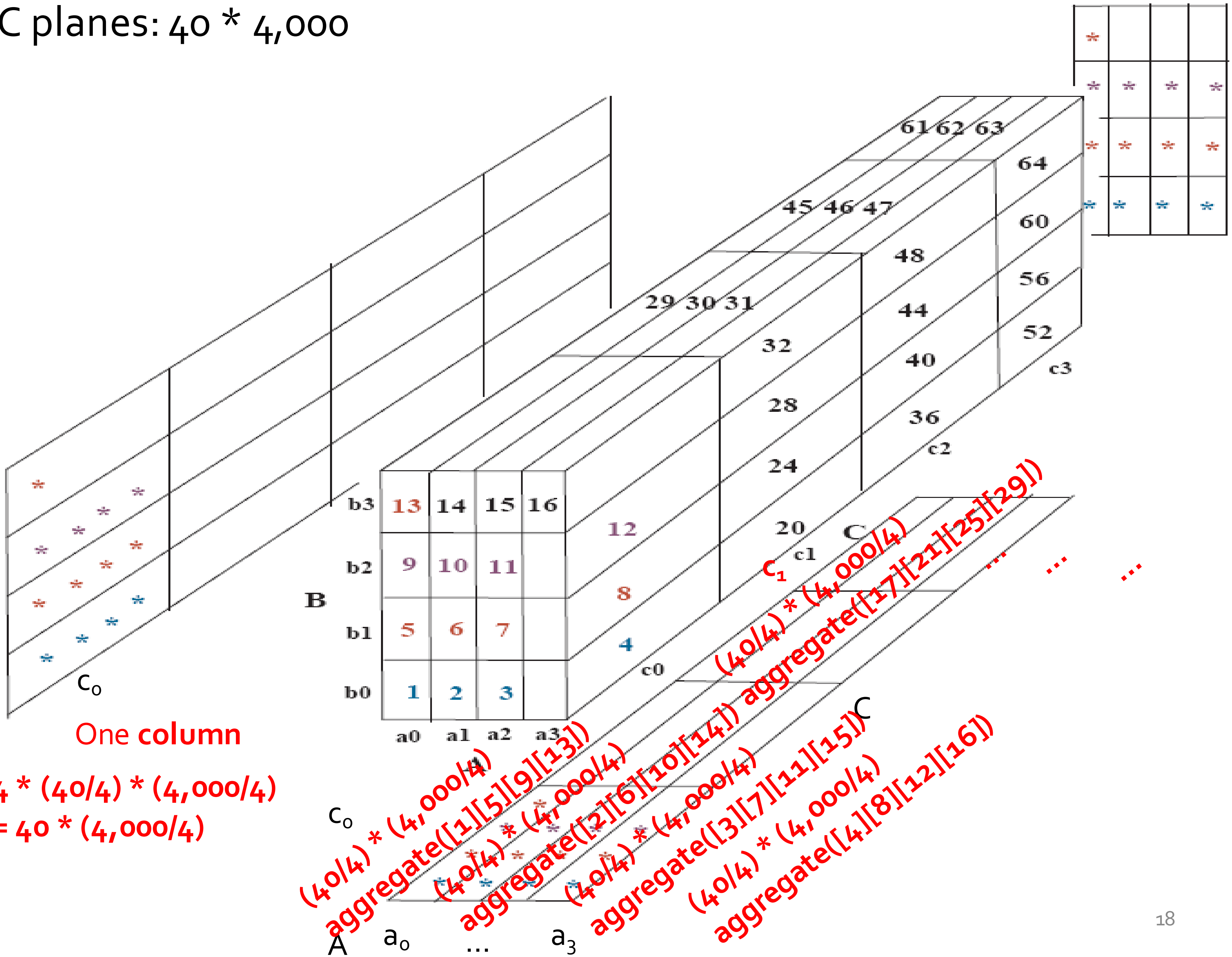
Min memory size: ?
< 40*400 + 400*4,000 + 40 * 4,000
= 1,776,000

One **column** of AC plane:
40 * (4,000/4) = 40,000

BC planes: 400 * 4,000



AC planes: 40 * 4,000



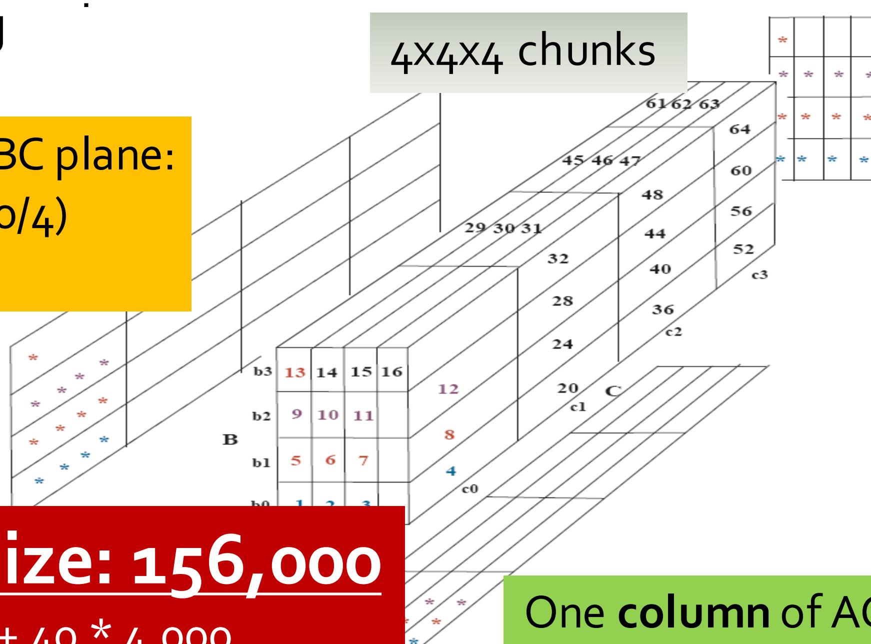
One **column**

4 * (40/4) * (4,000/4)
= 40 * (4,000/4)

Multi-way Array Aggregation (3-D to 2-D)

- How to minimizes the memory requirement and reduced I/Os?
  - Keep the **smallest** plane in **main memory**
  - Fetch and compute **only one chunk** at a time for the **largest** plane
  - The planes should be **sorted** and computed according to their **size** in ascending

4x4x4 chunks



Entire AB plane:
40 * 400
= 16,000

One **chunk** of BC plane:
(400/4) * (4,000/4)
= 100,000

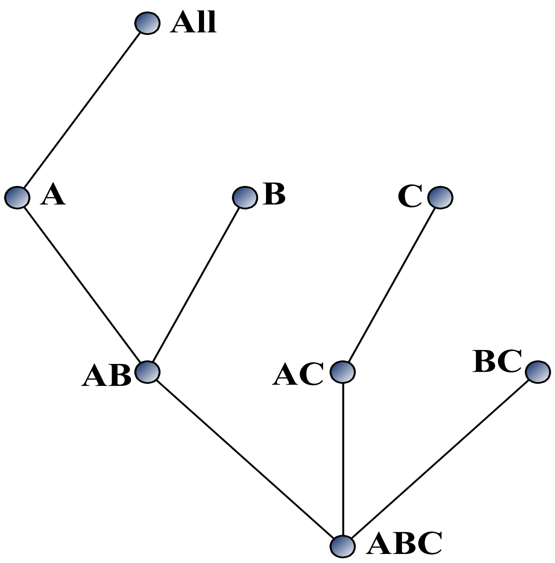
A: 40 (location),
B: 400 (item),
C: 4,000 (time)

Min memory size: 156,000
< 40*400 + 400*4,000 + 40 * 4,000
= 1,776,000

One **column** of AC plane:
40 * (4,000/4) = 40,000

Multi-Way Array Aggregation

- Array-based "**bottom-up**" algorithm (from ABC to AB,…)
- Using multi-dimensional **chunks**
- Simultaneous aggregation on multiple dimensions
- Cannot do *Apriori* pruning: No iceberg optimization
- Comments on the method
  - Efficient for computing the full cube for **a small number of dimensions**
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods should be used



Zhao, Deshpande & Naughton. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates . SIGMOD'97

Summary

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
  - Multi-Way Array Aggregation

References

- S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. VLDB'96
- K. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs.. SIGMOD'99
- J. Han, J. Pei, G. Dong, K. Wang. Efficient Computation of Iceberg Cubes With Complex Measures. SIGMOD'01
- L. V. S. Lakshmanan, J. Pei, and J. Han, Quotient Cube: How to Summarize the Semantics of a Data Cube, VLDB'02
- X. Li, J. Han, and H. Gonzalez, High-Dimensional OLAP: A Minimal Cubing Approach, VLDB'04
- X. Li, J. Han, Z. Yin, J.-G. Lee, Y. Sun, "Sampling Cube: A Framework for Statistical OLAP over Sampling Data", SIGMOD'08
- K. Ross and D. Srivastava. Fast computation of sparse datacubes. VLDB'97
- D. Xin, J. Han, X. Li, B. W. Wah, Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration, VLDB'03
- Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. SIGMOD'97
- D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. VLDB'05

References (cont.)

- R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. ICDE'97
- B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan. Prediction cubes. VLDB'05
- B.-C. Chen, R. Ramakrishnan, J.W. Shavlik, and P. Tamma. Bellwether analysis: Predicting global aggregates from local regions. VLDB'06
- Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, Multi-Dimensional Regression Analysis of Time-Series Data Streams, VLDB'02
- R. Fagin, R. V. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-structural databases. PODS'05
- J. Han. Towards on-line analytical mining in large databases. SIGMOD Record, 27:97–107, 1998
- T. Imielinski, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. Data Mining & Knowledge Discovery, 6:219–258, 2002.
- R. Ramakrishnan and B.-C. Chen. Exploratory mining in cube space. Data Mining and Knowledge Discovery, 15:29–54, 2007.
- K. A. Ross, D. Srivastava, and D. Chatziantoniou. Complex aggregation at multiple granularities. EDBT'98
- S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. EDBT'98
- G. Sathe and S. Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. *VLDB'01*