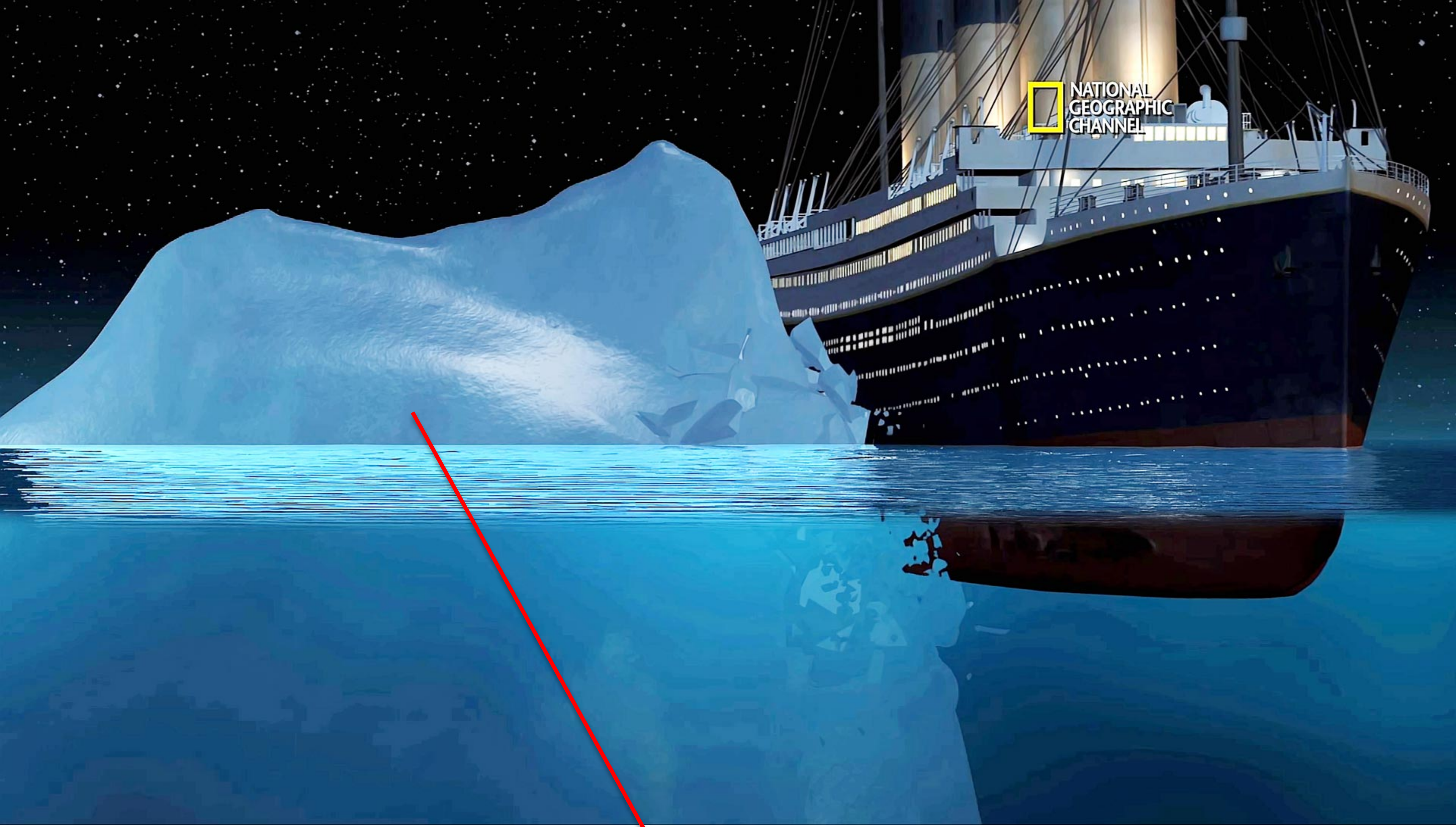
Chapter 5.
Data Cube Technology

Meng Jiang

CS412 Summer 2017:

Introduction to Data Mining

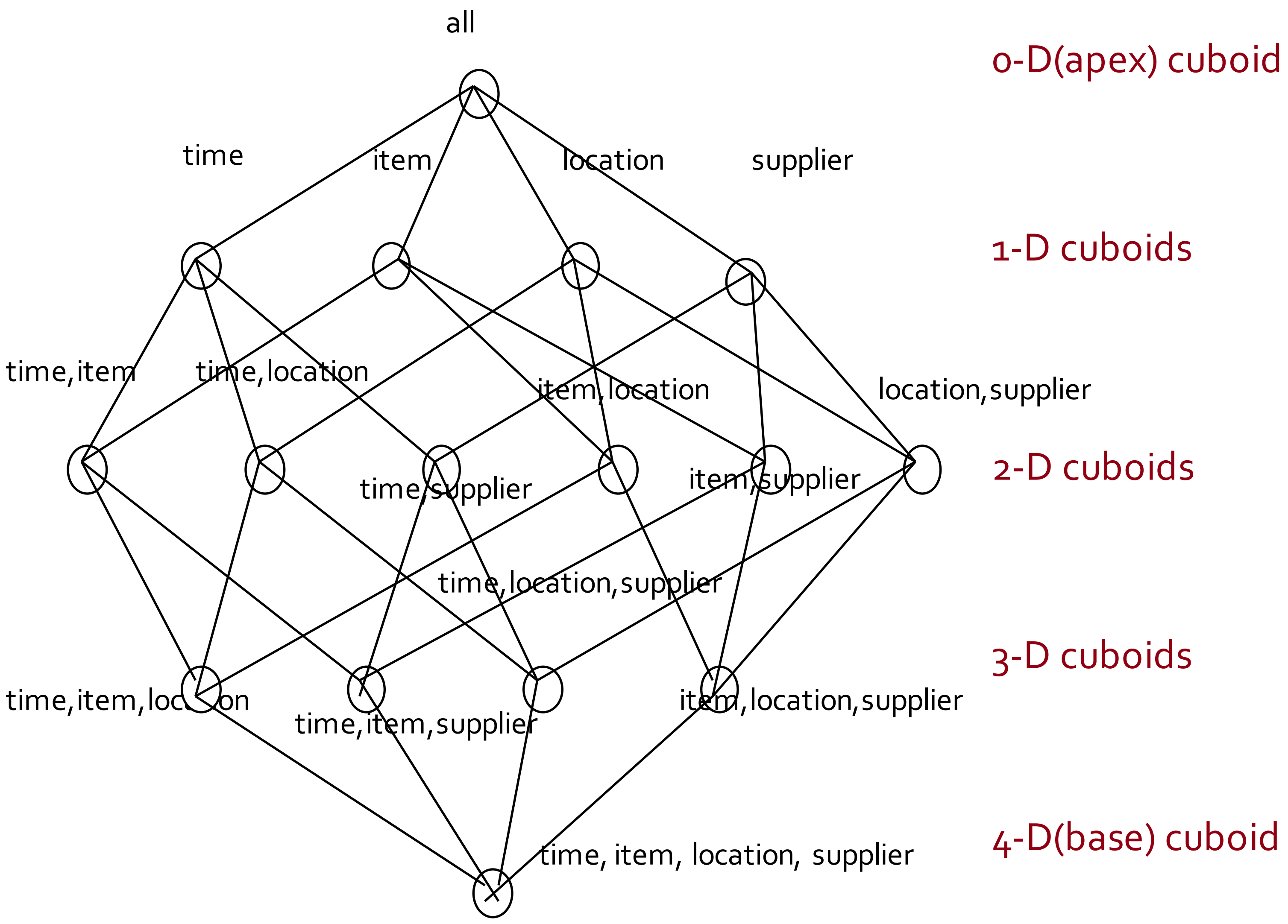


What the hell is that?!!

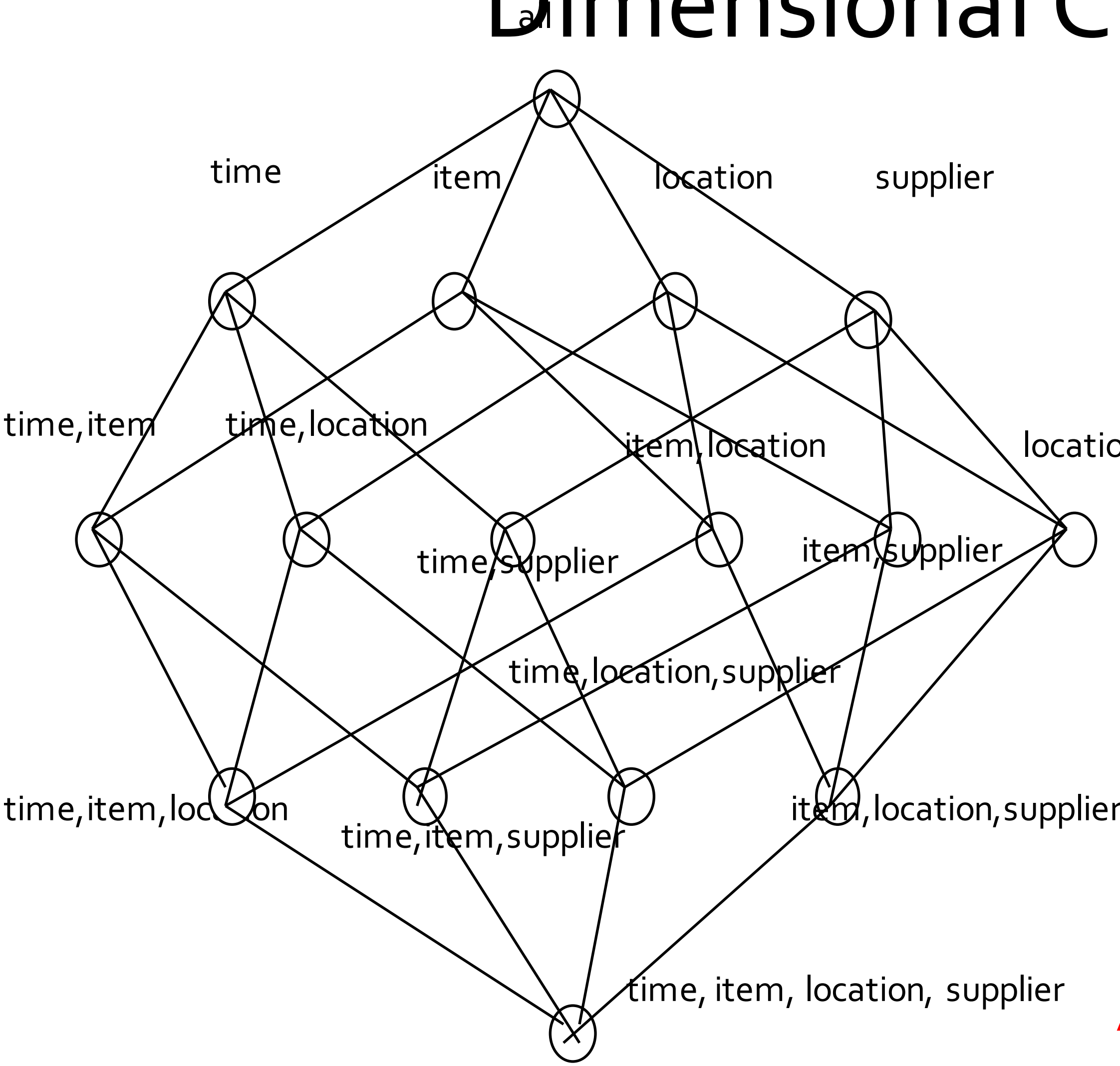
Data Cube Technology

- **Data Cube Computation: Basic Concepts**
- Data Cube Computation Methods
- Multidimensional Data Analysis in Cube Space

Cuboids in Data Cube



Cells in Data Cube across Different Dimensional Cuboids



Base vs. aggregate cells

(9/15, cheese, Urbana, KRAFT)

(9/15, cheese, Urbana, *)

(*, cheese, Urbana, *)

(*, cheese, *, *)

(*, *, *, *)

Parent vs. child cells

(9/15, *, Urbana, *)

(9/15, cheese, Urbana, *)

OR

(9/15, cheese, Urbana, *)

(9/15, cheese, Urbana, KRAFT)

Ancestor vs. descendant cells

(9/15, *, Urbana., *)

(9/15, cheese, Urbana, KRAFT)

Cells in Cuboids of the Same Dimensions

Parent vs. child cells

(9/15, cheese, Illinois, *)
(9/15, cheese, Urbana, *)

Sibling cells

(9/15, cheese, Urbana, *)
(9/15, cheese, Chicago, *)

Reference [see page 77] Multi-Dimensional, Phrase-Based Summarization in Text Cubes
http://sites.computer.org/debull/A16sept/p74.pdf

"Each parent cell is found by changing exactly one non-∗ dimension value in cell c into its parent value.
Each child cell is found by either changing one ∗ value into non-∗ or by replacing it by one of the child values.
Each sibling cell must share one parent with cell c."

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"?

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: $\{(a_1, a_2, a_3, ..., a_{100}), (a_1, a_2, b_3, ..., b_{100})\}$
  - How many aggregate cells if "having count >= 1"?
    - 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"?
    - Answer: $(2^{101} - 2) - 4$
  - What about the iceberg cells, (i.e., with condition: "having count >= 2")?

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

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"?
    - Answer: $(2^{101} - 2) - 4$
  - What about the iceberg cells, (i.e., with condition: "having count >= 2")?
    - Answer: 4

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!)

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

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
- **Cube Shell:** The ***cuboids*** involving only a ***small # of dimensions***, e.g., 2
  - Idea: Only compute cube shells, other dimension combinations can be computed on the fly

Data Cube Technology

- Data Cube Computation: Basic Concepts
- **Data Cube Computation Methods**
- Multidimensional Data Analysis in Cube Space

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, ……

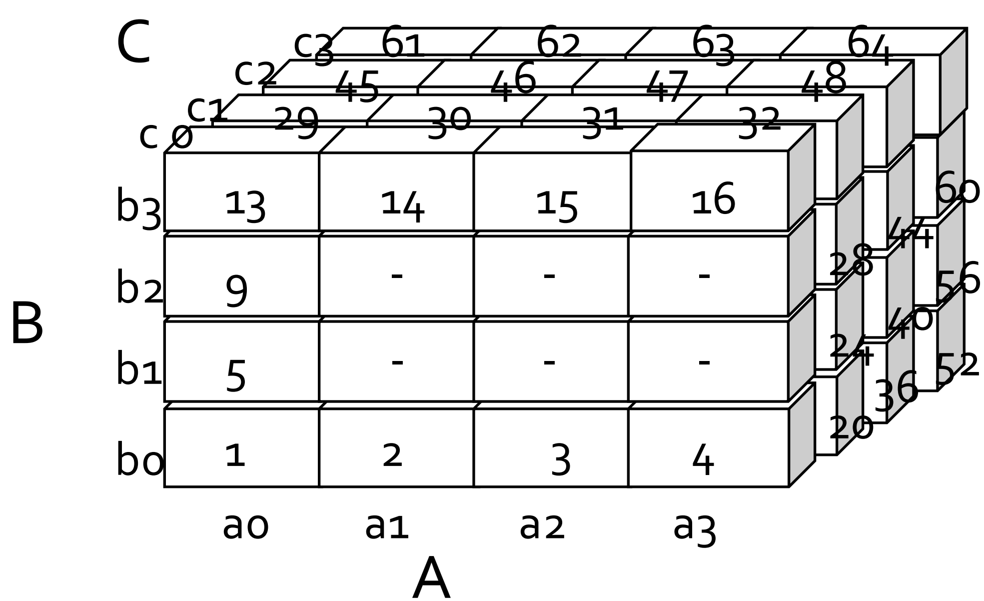
Efficient Data Cube Computation: General Heuristics

- Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples
- Aggregates may be computed from previously computed aggregates, rather than from the base fact table
  - Smallest-child: computing a cuboid **from the smallest**, previously computed cuboid
  - Cache-results: caching results of a cuboid from which other cuboids are computed to **reduce disk I/Os**
  - Amortize-scans: computing **as many as possible** cuboids at the same time to **amortize disk reads**
  - Share-sorts: **sharing sorting costs** cross multiple cuboids when **sort-based method** is used
  - Share-partitions: **sharing the partitioning cost** across multiple cuboids when **hash-based algorithms** are used

S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, S. Sarawagi. On the computation of multidimensional aggregates. VLDB'96

Cube Computation: Multi-Way Array Aggregation (MOLAP)

- 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



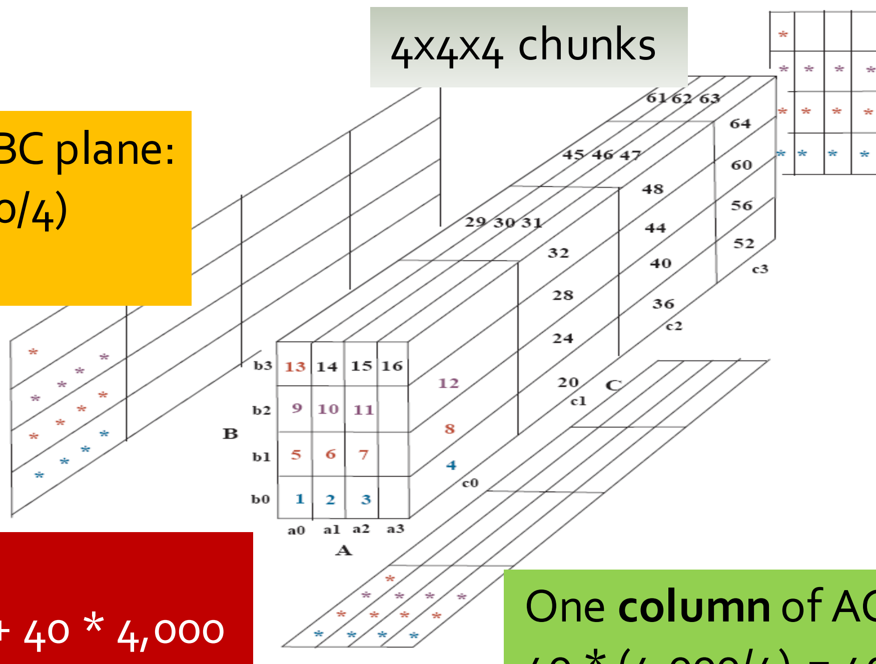
What is the best traversing order to do multi-way aggregation?

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?



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

Entire AB plane:
40 * 400
= 16,000

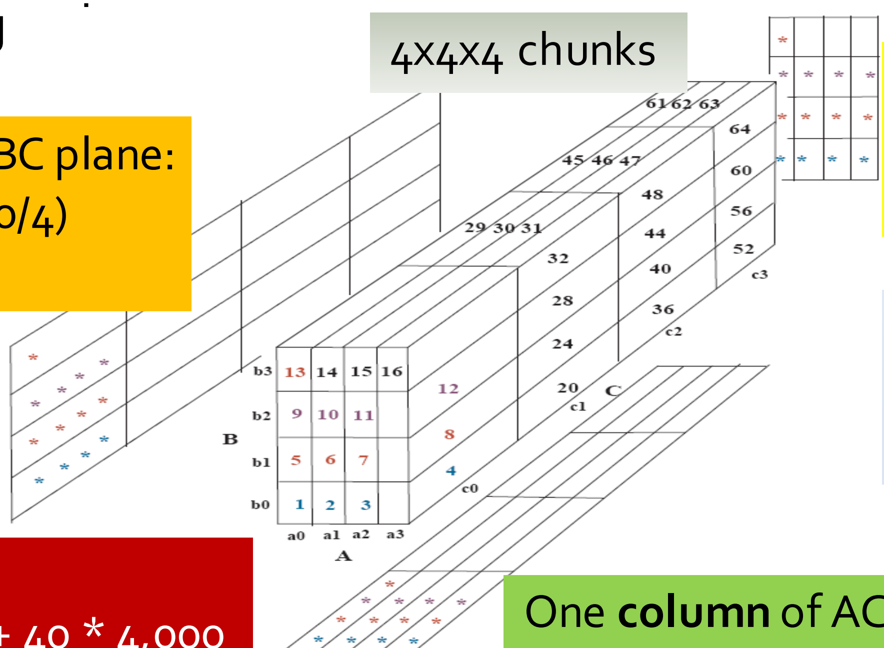
A: 40 (location),
B: 400 (item),
C: 4,000 (time)

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

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



Entire AB plane:
40 * 400
= 16,000

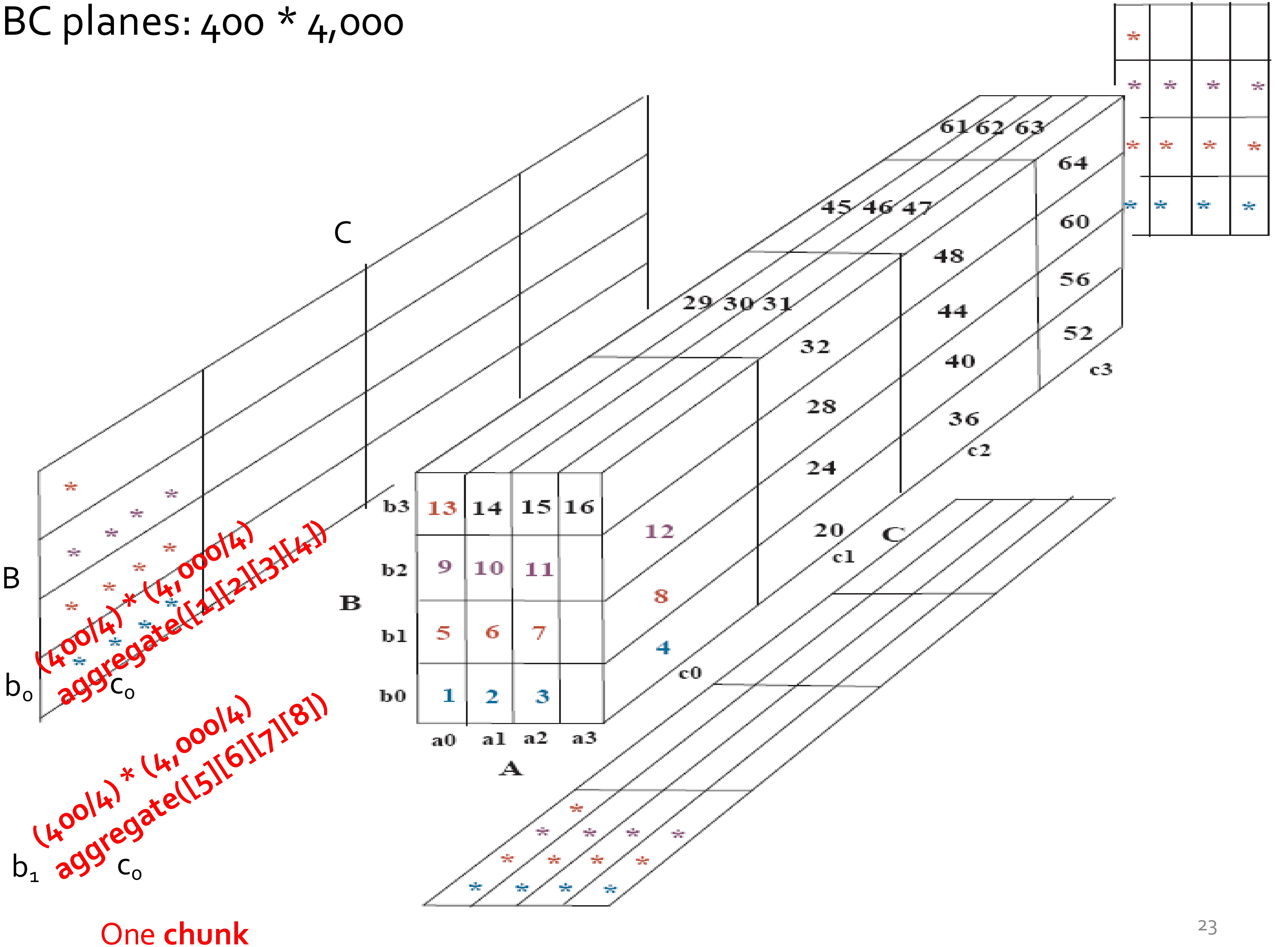
One **chunk** of BC plane:
(400/4) * (4,000/4)
= 100,000

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

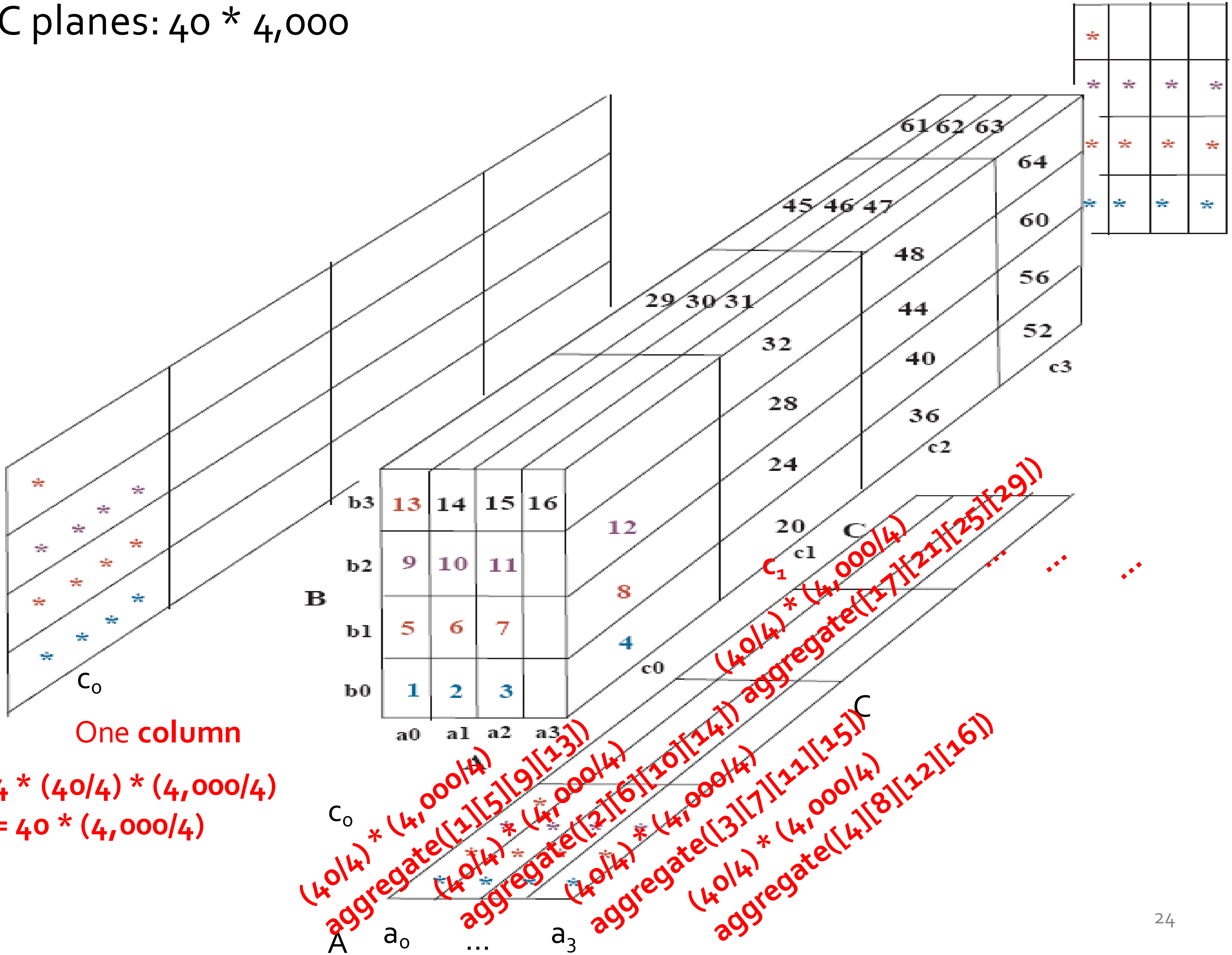
Min memory size: ?
< 40*400 + 400*4,000 + 40 * 4,000
= 1,776,000

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

BC planes: 400 * 4,000



AC planes: 40 * 4,000



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

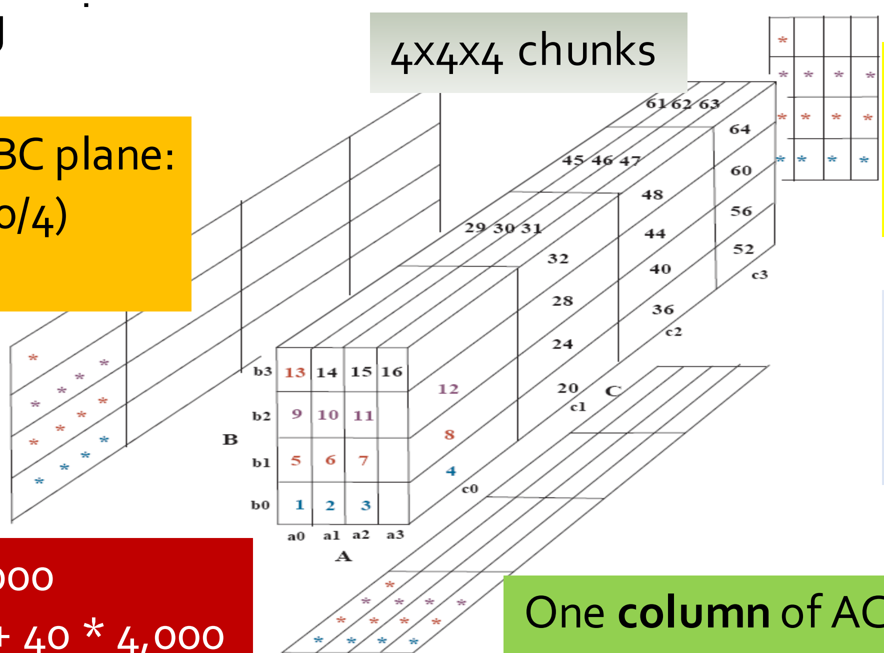
Multi-way Array Aggregation (3-D to 2-D)

- 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

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

Entire AB plane:
40 * 400
= 16,000



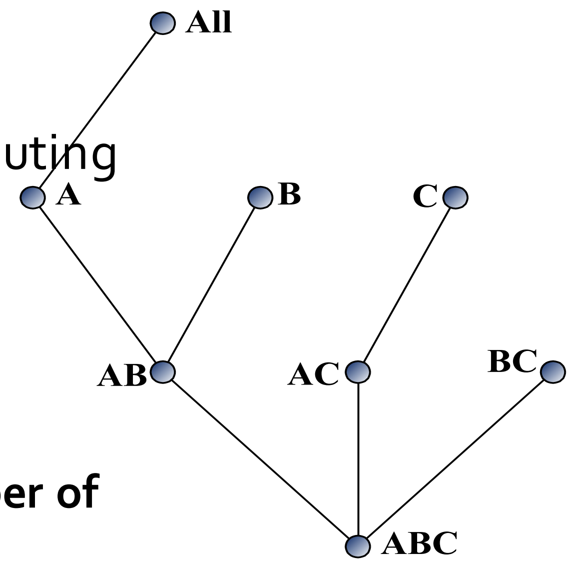
A: 40 (location),
B: 400 (item),
C: 4,000 (time)

Min memory size: 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
- Intermediate aggregate values are re-used for computing ancestor cuboids
- 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 (e.g., BUC) should be used



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

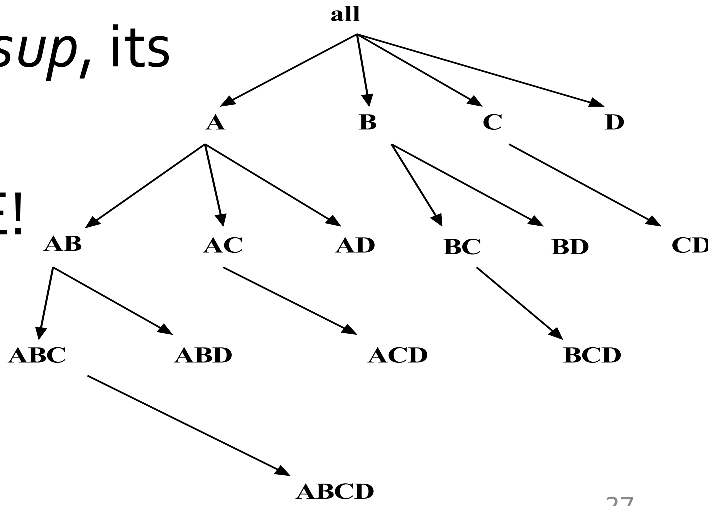
Cube Computation: Computing in Reverse Order

- BUC (Beyer & Ramakrishnan, SIGMOD'99)

  BUC: acronym of Bottom-Up (cube) Computation

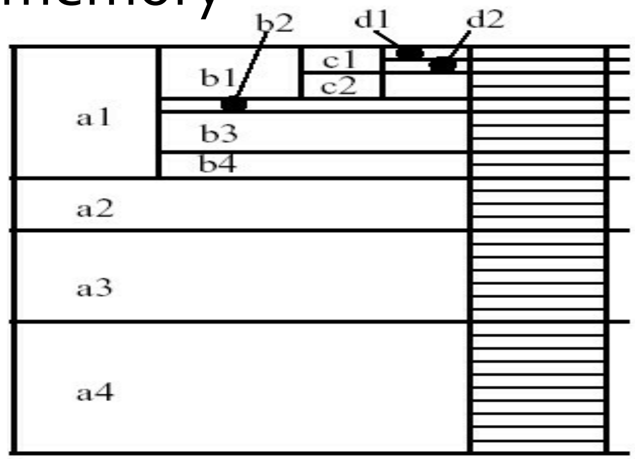
  (Note: It is "**top-down**" in our view since we put Apex cuboid on the top!)

- Divides dimensions into partitions and facilitates **iceberg pruning (it works now!)**
  - If a partition does not satisfy *min_sup*, its **descendants** can be pruned
  - If *min_sup* = 1 Þ compute full CUBE!
- No simultaneous aggregation



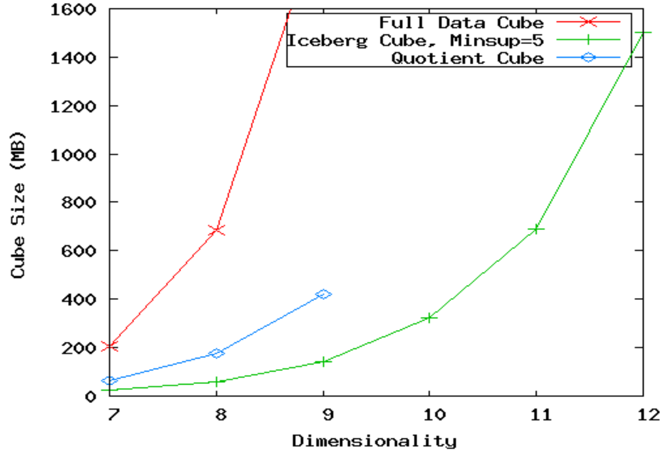
BUC: Partitioning and Aggregating

- Usually, entire data set cannot fit in main memory
- Sort *distinct* values
  - partition into blocks that fit
- Continue processing
- Optimizations
  - Partitioning
    - External Sorting, Hashing, Counting Sort
  - Ordering dimensions to encourage pruning
    - Cardinality, Skew, Correlation
  - Collapsing duplicates
    - Cannot do holistic aggregates anymore!



High-Dimensional OLAP? —The Curse of Dimensionality

- High-D OLAP: Needed in many applications
  - Science and engineering analysis
  - Bio-data analysis: thousands of genes
  - Statistical surveys: hundreds of variables
- None of the previous cubing method can handle high dimensionality!
  - Iceberg cube and compressed cubes: only delay the inevitable explosion
  - Full materialization: still significant overhead in accessing results on disk
- **A shell-fragment approach:** X. Li, J. Han, and H. Gonzalez, High-Dimensional OLAP: A Minimal Cubing Approach, VLDB'04



A curse of dimensionality: A database of 600,000 tuples. Each dimension has cardinality of 100 and *zipf* of 2.

Fast High-Dimensional OLAP with Minimal Cubing

- Observation: OLAP occurs only on a small subset of dimensions at a time
- Semi-Online Computational Model
  - Partition the set of dimensions into **shell fragments**
  - Compute data cubes for each shell fragment while retaining **inverted indices** or **value-list indices**
  - Given the pre-computed **fragment cubes**, dynamically compute cube cells of the high-dimensional data cube *online*
- Major idea: Tradeoff between the amount of **pre-computation** and the speed of **online computation**
  - Reducing computing high-dimensional cube into **pre-computing** a set of **lower dimensional** cubes
  - **Online re-construction** of original high-dimensional space
  - Lossless reduction

Computing a 5-D Cube with 2-Shell Fragments

- Example: Let the cube aggregation function be **count**

tid	**A**	**B**	**C**	**D**	**E**
1	a1	b1	c1	d1	e1
2	a1	b2	c1	d2	e1
3	a1	b2	c1	d1	e2
4	a2	b1	c1	d1	e2
5	a2	b1	c1	d1	e3

Attribute Value	TID List	List Size
a1	1 2 3	3
a2	4 5	2
b1	1 4 5	3
b2	2 3	2
c1	1 2 3 4 5	5
d1	1 3 4 5	4
d2	2	1
e1	1 2	2
e2	3 4	2
e3	5	1

- Divide the 5-D table into 2 shell fragments:
  - (A, B, C) and (D, E)
- Build traditional invert index (TID) or RID list

Shell Fragment Cubes: Ideas

- Generalize the 1-D inverted indices to **multi-dimensional inverted indices** in the data cube sense
- Compute **all cuboids for data cubes ABC and DE** while retaining the inverted indices

Attribute Value	TID List	List Size
a1	1 2 3	3
a2	4 5	2
b1	1 4 5	3
b2	2 3	2
c1	1 2 3 4 5	5
d1	1 3 4 5	4
d2	2	1
e1	1 2	2
e2	3 4	2
e3	5	1

Cell	Intersection	TID List	List Size
a1 b1	1 2 3 ∩ 1 4 5	1	1
a1 b2	1 2 3 ∩ 2 3	2 3	2
a2 b1	4 5 ∩ 1 4 5	4 5	2
a2 b2	4 5 ∩ 2 3	ϕ	0

Cell	Intersection	TID List	List Size
d1 e1	?	?	?
d1 e2	?	?	?
d1 e3	?	?	?
d2 e1	?	?	?

Answer: http://hanj.cs.illinois.edu/pdf/vldb04_hdolap.pdf

Shell Fragment Cubes: Size and Design

- Given a database of T tuples, **D dimensions**, and F shell fragment size, the fragment cubes' space requirement is:
  - For **F < 5**, the growth is sub-linear $O\left(T\left\lceil\frac{D}{F}\right\rceil(2^F - 1)\right)$
- Shell fragments do not have to be disjoint
- Fragment groupings can be arbitrary to allow for maximum online performance
  - Known common combinations (e.g.,<city, state>) should be grouped together
- Shell fragment sizes can be adjusted for optimal balance between offline and online computation

Data Cube Technology

- Data Cube Computation: Basic Concepts
- Data Cube Computation Methods
- **Multidimensional Data Analysis in Cube Space**

Data Mining in Cube Space

- Data cube greatly increases the analysis bandwidth
- Four ways to **interact OLAP-styled analysis and data mining**
  - Using cube space to define **data space** for mining
  - Using OLAP queries to generate **features and targets** for mining, e.g., multi-feature cube
  - Using data-mining models as **building blocks** in a multi-step mining process, e.g., prediction cube
  - Using data-cube computation techniques to **speed up** repeated model construction
    - Cube-space data mining may require building a model for each candidate data space
    - Sharing computation across model-construction for different candidates may lead to efficient mining

Complex Aggregation at Multiple Granularities: Multi-Feature Cubes

- Multi-feature cubes (Ross, et al. 1998): Compute complex queries involving multiple dependent aggregates at multiple granularities
- Ex. Grouping by all subsets of {item, region, month}, find the **maximum price** in 2010 for each group, and the **total sales** among all maximum price tuples

```
select item, region, month, max(price), sum(R.sales)
from purchases
where year = 2010
cube by item, region, month: R
such that R.price = max(price)
```

- Continuing the last example, **among the max price tuples**, find the **min and max shelf life**, and find the fraction of the total sales due to tuple that have **min shelf life** within the set of all max price tuples

Discovery-Driven Exploration of Data Cubes

- Discovery-driven exploration of huge cube space (Sarawagi, et al.'98)
  - Effective navigation of large OLAP data cubes
  - Pre-compute measures indicating exceptions, guide user in the data analysis, at all levels of aggregation
  - Exception: significantly different from the value anticipated, based on a statistical model
  - Visual cues such as background color are used to reflect the degree of exception of each cell
- Kinds of exceptions
  - **SelfExp: represents the surprise value of the cell relative to other cells at same level of aggregation**
  - **InExp: represents the degree of surprise somewhere beneath this cell if we drill down from the cell**
  - **PathExp: represents the degree of surprise for each drill-down path from the cell**

Exceptions: SelfExp, InExp, PathExp

Product	(All)
Region	(All)

SelfExp

Sum of Sales	Month											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Total		2%	0%	2%	2%	4%	3%	0%	-8%	0%	-3%	4%

InExp

Region	(All)

Avg.Sales	Month											
Product	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Birch-B		10%	-7%	3%	-4%	15%	-12%	-3%	1%	42%	-14%	-10%
Chery-S		1%	1%	4%	3%	5%	5%	-9%	-12%	1%	-5%	5%
Cola		-1%	2%	3%	4%	9%	4%	1%	-11%	-8%	-2%	7%
Cream-S		3%	1%	6%	3%	3%	8%	-3%	-12%	-2%	1%	10%
Diet-B		1%	1%	-1%	2%	1%	2%	0%	-6%	-1%	-4%	2%
Diet-C		3%	2%	5%	2%	4%	7%	-7%	-12%	-2%	-2%	8%
Diet-S		2%	-1%	0%	0%	4%	2%	4%	-9%	5%	-3%	0%
Grape-S		1%	1%	0%	4%	5%	1%	3%	-9%	-1%	-8%	4%
Jolt-C		-1%	-4%	2%	2%	0%	-4%	2%	6%	-2%	0%	0%
Kiwi-S		2%	1%	4%	1%	-1%	3%	-1%	-4%	4%	0%	1%
Old-B		4%	-1%	0%	1%	5%	2%	7%	-10%	3%	-3%	1%
Orang-S		1%	1%	3%	4%	2%	1%	-1%	-1%	-6%	-4%	9%
Sasprla		-1%	2%	1%	3%	-3%	5%	-10%	-2%	-1%	1%	5%

Product	Diet-S

Avg.Sales	Month											
Region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
C		0%	-2%	0%	1%	4%	1%	5%	-6%	2%	-2%	-2%
E		0%	2%	-8%	7%	0%	5%	-40%	10%	-33%	2%	8%
S		0%	-1%	3%	-2%	2%	-2%	19%	-1%	12%	-1%	0%
W		5%	1%	0%	-2%	6%	6%	2%	-17%	9%	-7%	2%

Market	(All)
Product	Cola

Avg.Sales	Month											
Region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
C		3%	1%	4%	1%	4%	10%	-11%	-14%	-3%	5%	11%
E		-3%	3%	4%	4%	13%	2%	0%	-10%	-13%	-3%	8%
S		2%	-1%	1%	9%	6%	3%	21%	-15%	1%	-5%	4%
W		-2%	2%	2%	4%	12%	1%	1%	-9%	-11%	-4%	6%

Summary

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
  - Multi-Way Array Aggregation
  - BUC
  - High-Dimensional OLAP with Shell-Fragments
- Multidimensional Data Analysis in Cube Space
  - Multi-feature Cubes
  - Discovery-Driven Exploration of Data Cubes

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*