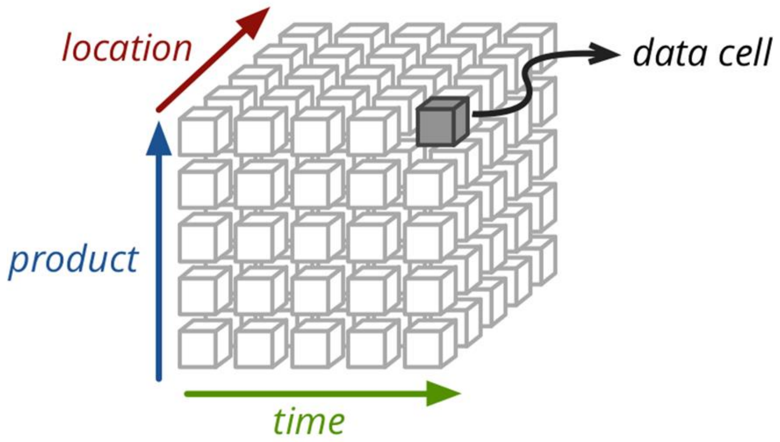
CS 412 Intro. to Data Mining

Chapter 5. Data Cube Technology

Qi Li, Computer Science, Univ. Illinois at Urbana-Champaign, 2018

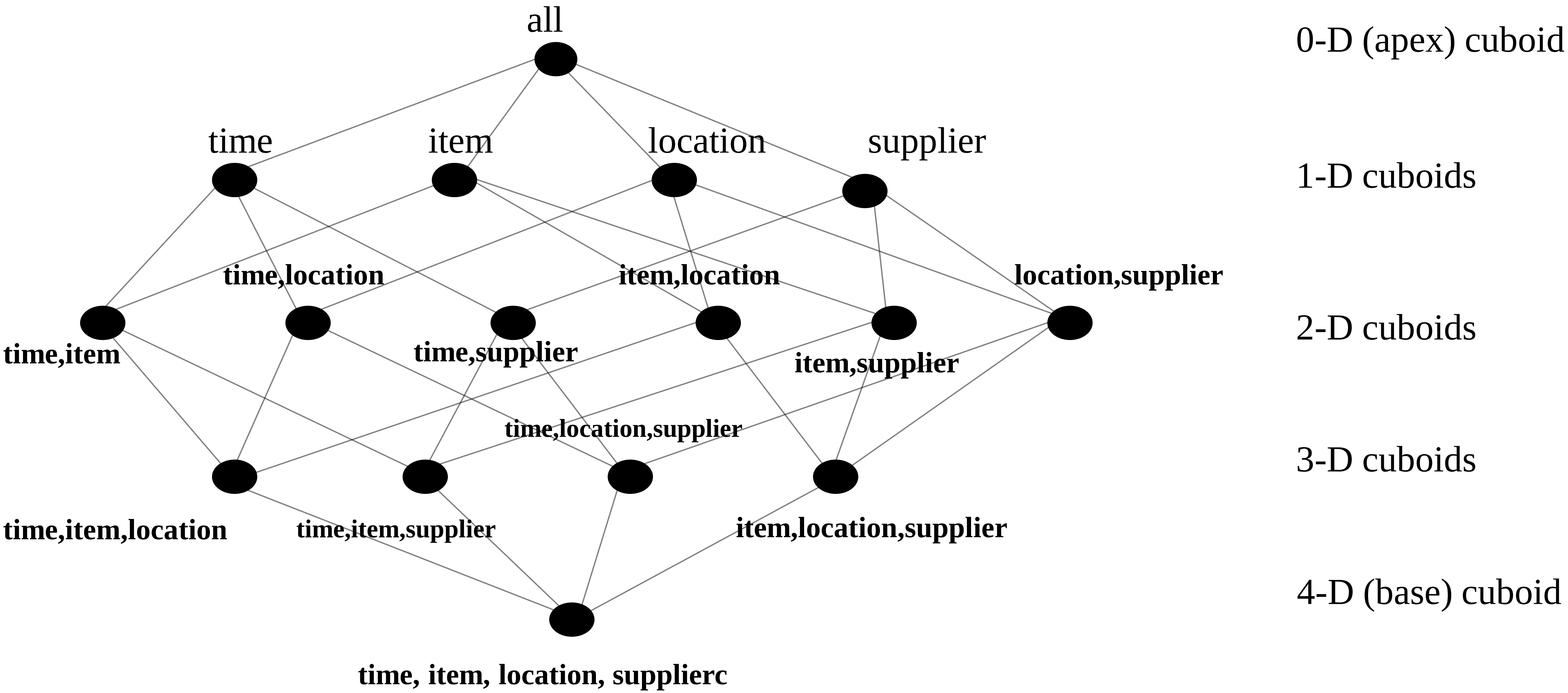
Chapter 5: Data Cube Technology

- ❑ Data Cube Computation: Basic Concepts

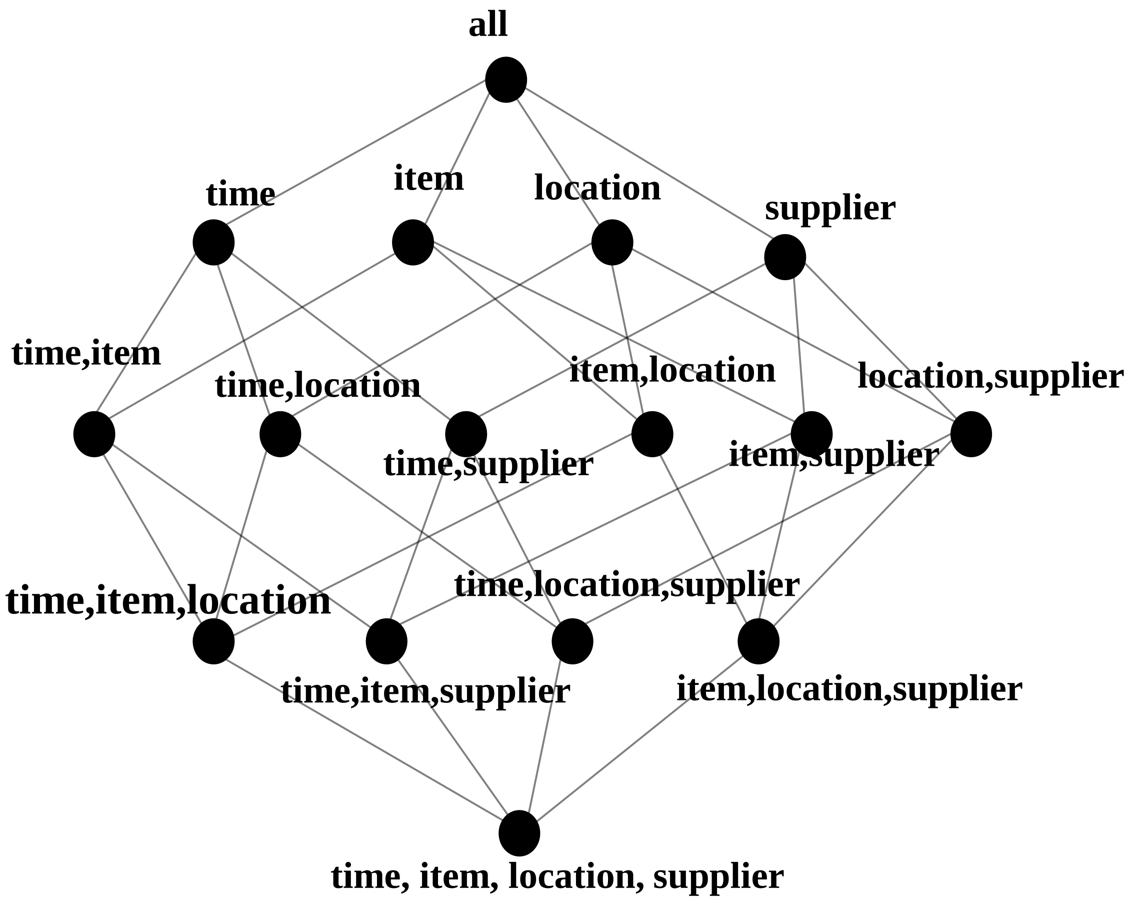


- ❑ Data Cube Computation Methods
- ❑ Processing Advanced Queries with Data Cube Technology
- ❑ Multidimensional Data Analysis in Cube Space
- ❑ Summary

Data Cube: A Lattice of Cuboids



Data Cube: A Lattice of Cuboids



- Base vs. aggregate cells
- Ancestor vs. descendant cells
- Parent vs. child cells

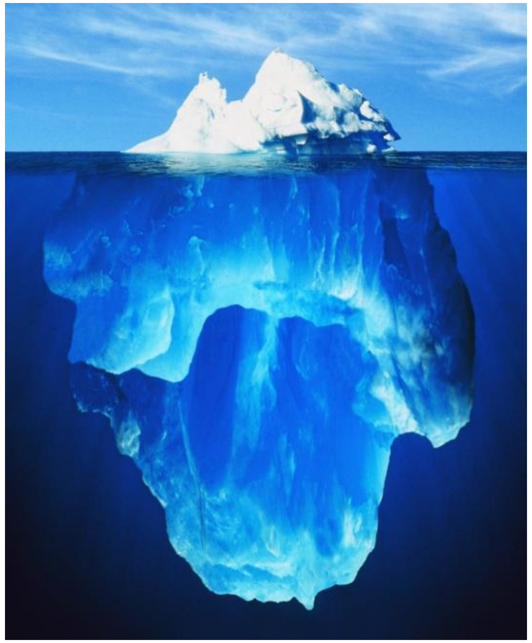
0-D (agg)	(*,*,*,*)
1-D (agg)	(*, milk, *, *)
2-D (agg)	(*, milk, Urbana, *)
2-D (agg)	(*, milk, Chicago, *)
3-D (agg)	(9/15, milk, Urbana, *)
4-D (base)	(9/15, milk, Urbana, Dairy_land)

Cube Materialization: Full Cube vs. Iceberg Cube

- Full cube vs. iceberg cube

  compute cube *sales_iceberg* **as**
  SELECT *month, city, customer_group*, **COUNT(*)**
  FROM *salesInfo*
  CUBE BY *month, city, customer_group*
  HAVING count(*) >= min support

  iceberg condition



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

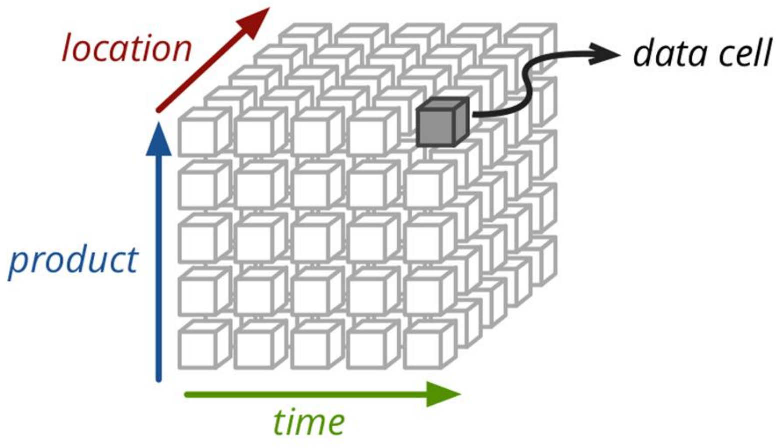
Is Iceberg Cube Good Enough? Closed Cube & Cube Shell

- 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

Q: For $(A_1, A_2, \ldots A_{100})$, how many combinations to compute?

Chapter 5: Data Cube Technology

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



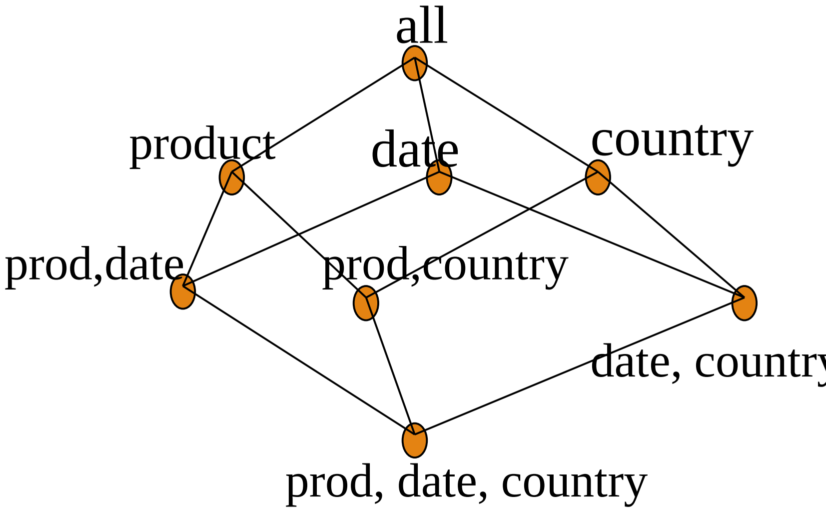
Roadmap for Efficient Computation

- General computation heuristics [1]
- Computing full/iceberg cubes: 3 methodologies
  - Bottom-Up:
    - Multi-Way array aggregation [2]
  - Top-down:
    - BUC [3]
- High-dimensional OLAP:
  - A Shell-Fragment Approach [4]
- Computing alternative kinds of cubes:
  - Partial cube, closed cube, approximate cube, ……

1. (Agarwal et al.'96)
2. (Zhao, Deshpande & Naughton, SIGMOD'97)
3. (Beyer & Ramarkrishnan, SIGMOD'99)
4. (Li, et al. VLDB'04)

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
  - **Share-sorts**
  - **Share-partitions**

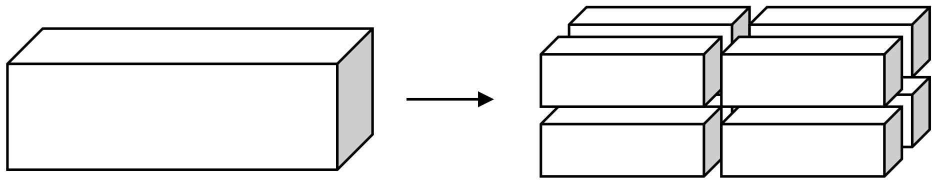


- Aggregates may be computed from previously computed aggregates, rather than from the base fact table
  - **Smallest-child:** computing a cuboid from the smallest, previously computed cuboid
  - **Cache-results:** caching results of a cuboid from which other cuboids are computed to reduce disk I/Os
  - **Amortize-scans:** computing as many as possible cuboids at the same time to amortize disk reads

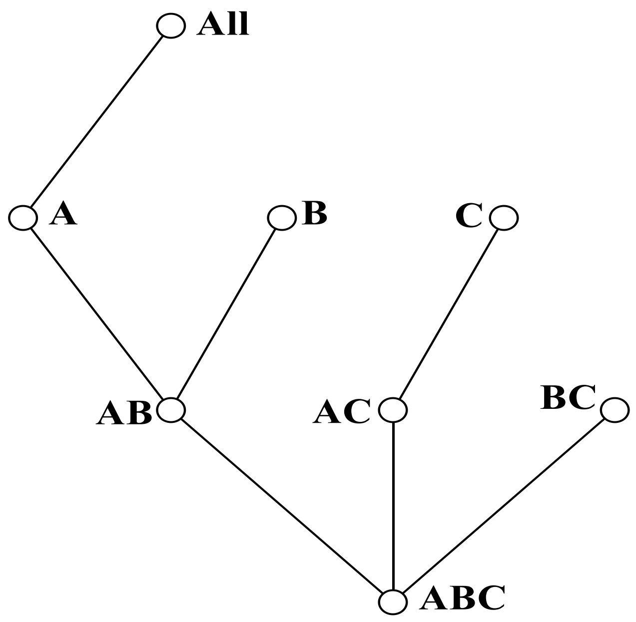
S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, S. Sarawagi. On the computation of multidimensional aggregates. VLDB'96

Multi-Way Array Aggregation (MOLAP)

- Full cube computation
- Bottom-up
- Array-based
  - Limited RAM -> Array chunking



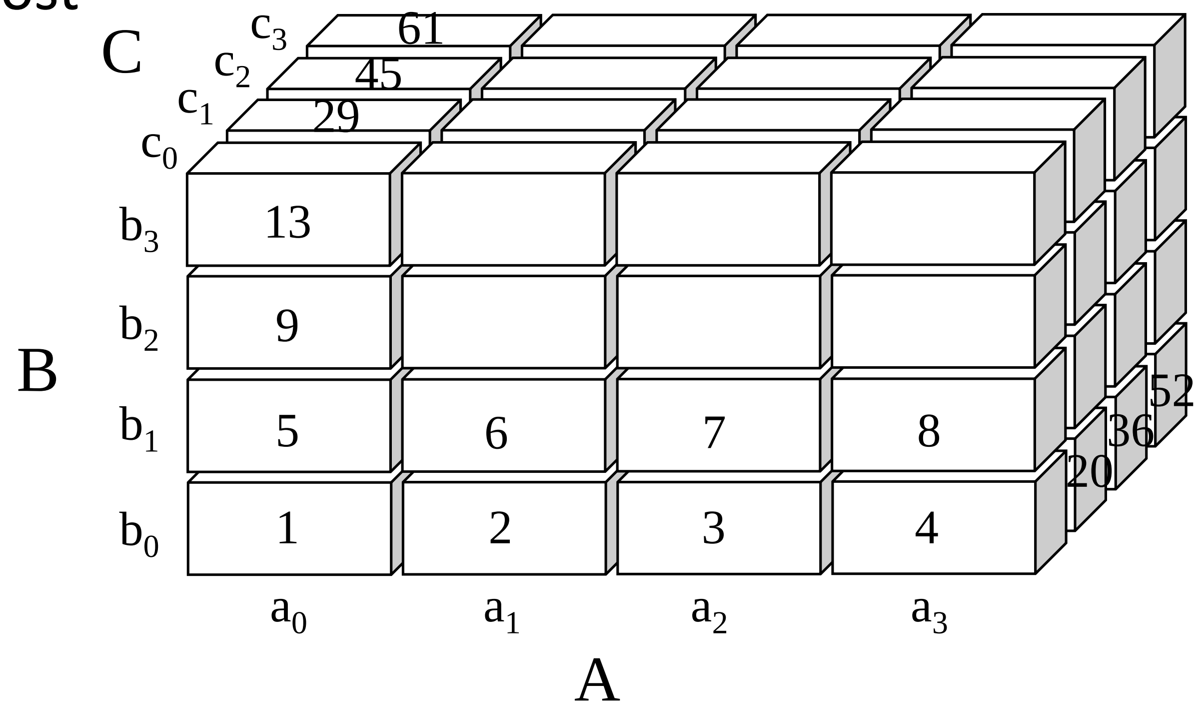
  - Each time load one chunk into memory
- How to compute aggregates efficiently?
  - Simultaneous aggregation on multiple dimensions
  - Re-use intermediate aggregate values



Multi-Way Array Aggregation (MOLAP)

- Partition arrays into *chunks* (a small subcube which fits in memory).
- 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
  - Reduces memory access and storage cost

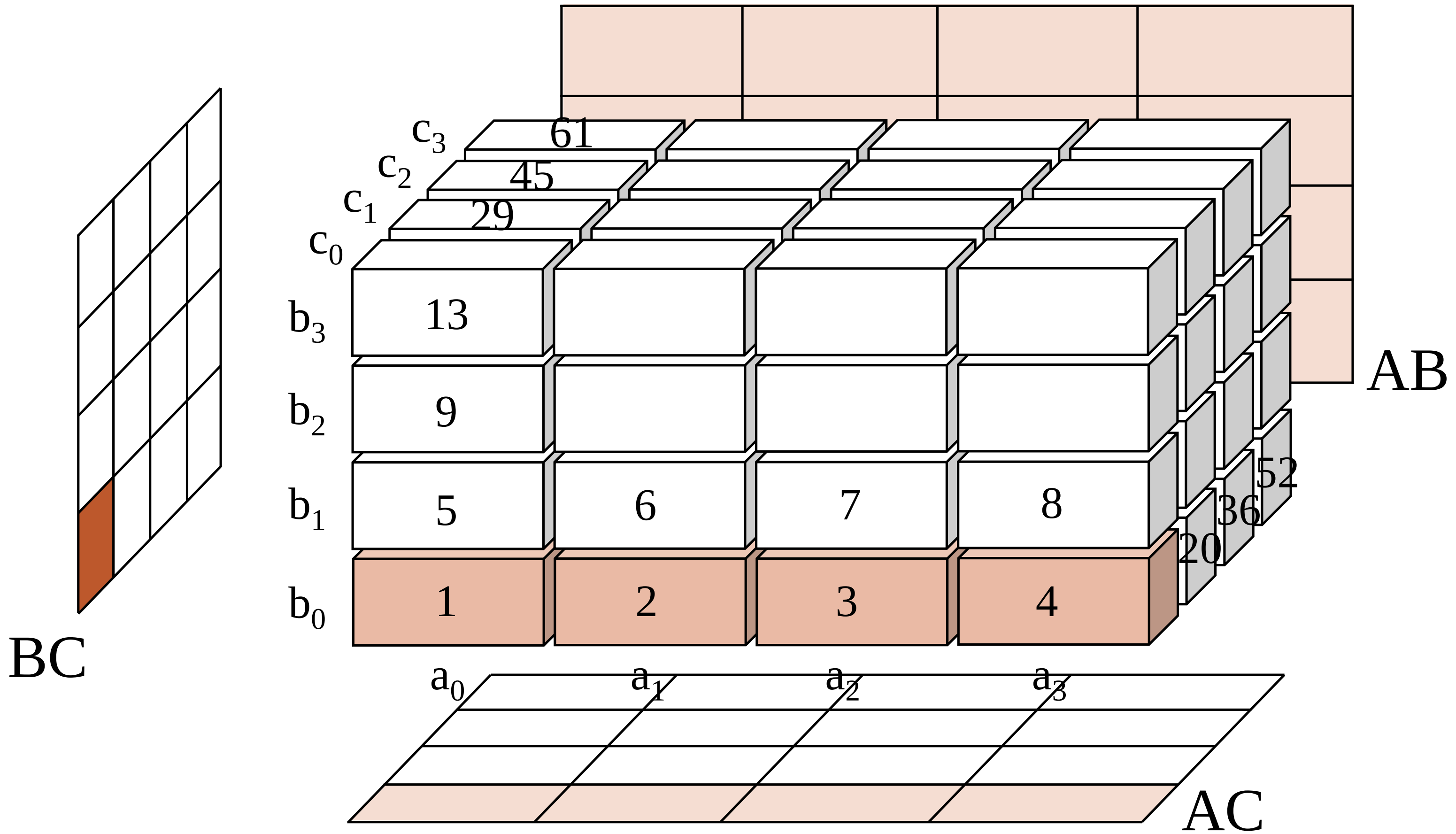
Example:
A: 4000, B: 400, C: 40
Chunk:
1000 x 100 x 10



Cube Computation: Multi-Way Array Aggregation (MOLAP)

- How to minimizes the memory requirement and reduced I/Os?
  - Suppose we scan using order: 1 – 2 – 3 – 4 – 5 – 6 – …

Example:
A: 4000, B: 400, C: 40
Chunk:
1000 x 100 x 10



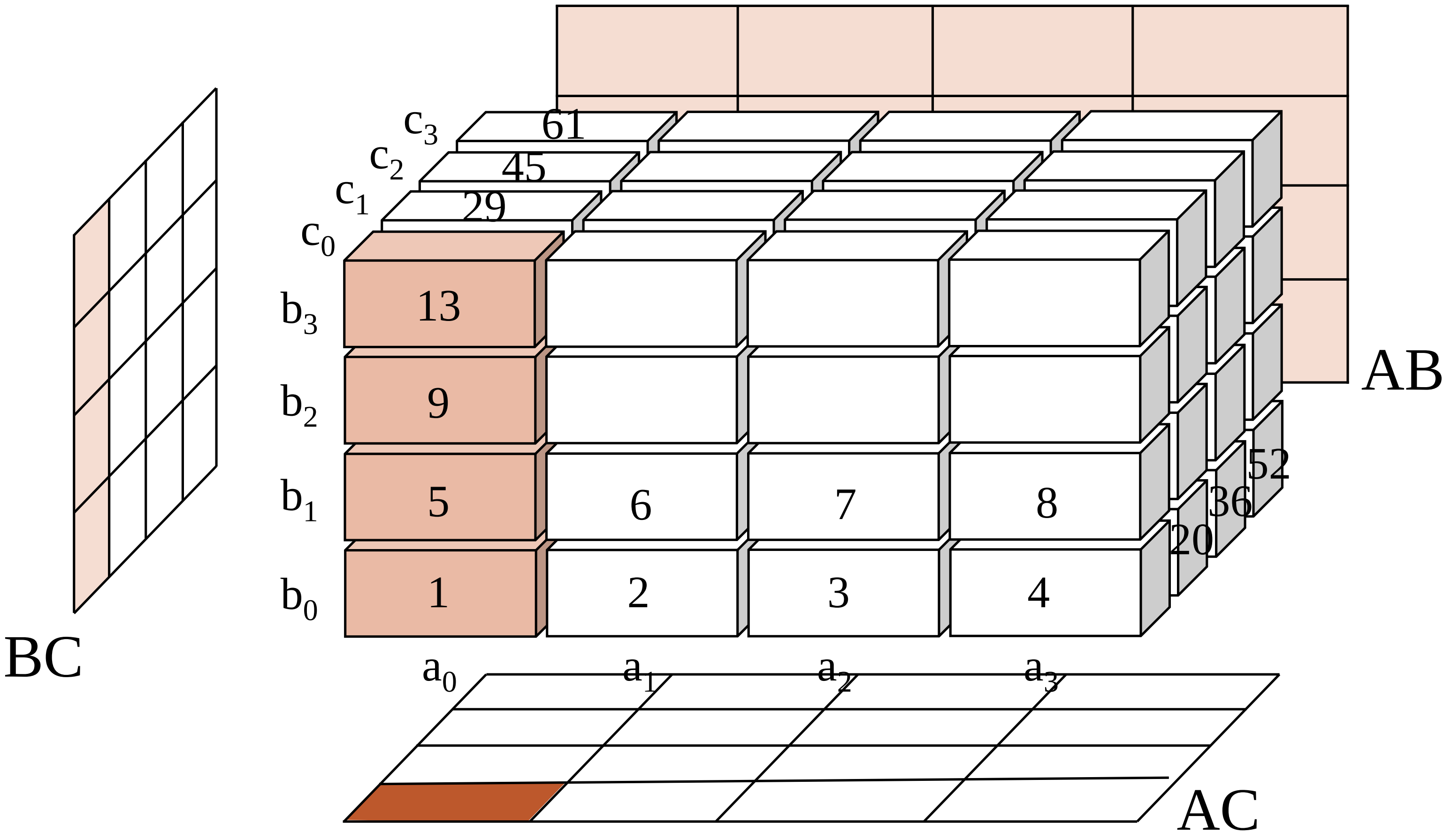
- While we scan through 1..4
  - One chunk of BC plane is fully computed
  - AB, AC plane can also be partially computed (multi-way)
- All fully computed chunks can be moved outside main memory
- Partially computed chunks must be stored in the main memory

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

Cube Computation: Multi-Way Array Aggregation (MOLAP)

- Reducing memory and I/O
  - Suppose we scan using order: 1 – 5 – 9 – 13 – 2 – 6 – …

Example:
A: 4000, B: 400, C: 40
Chunk:
1000 x 100 x 10

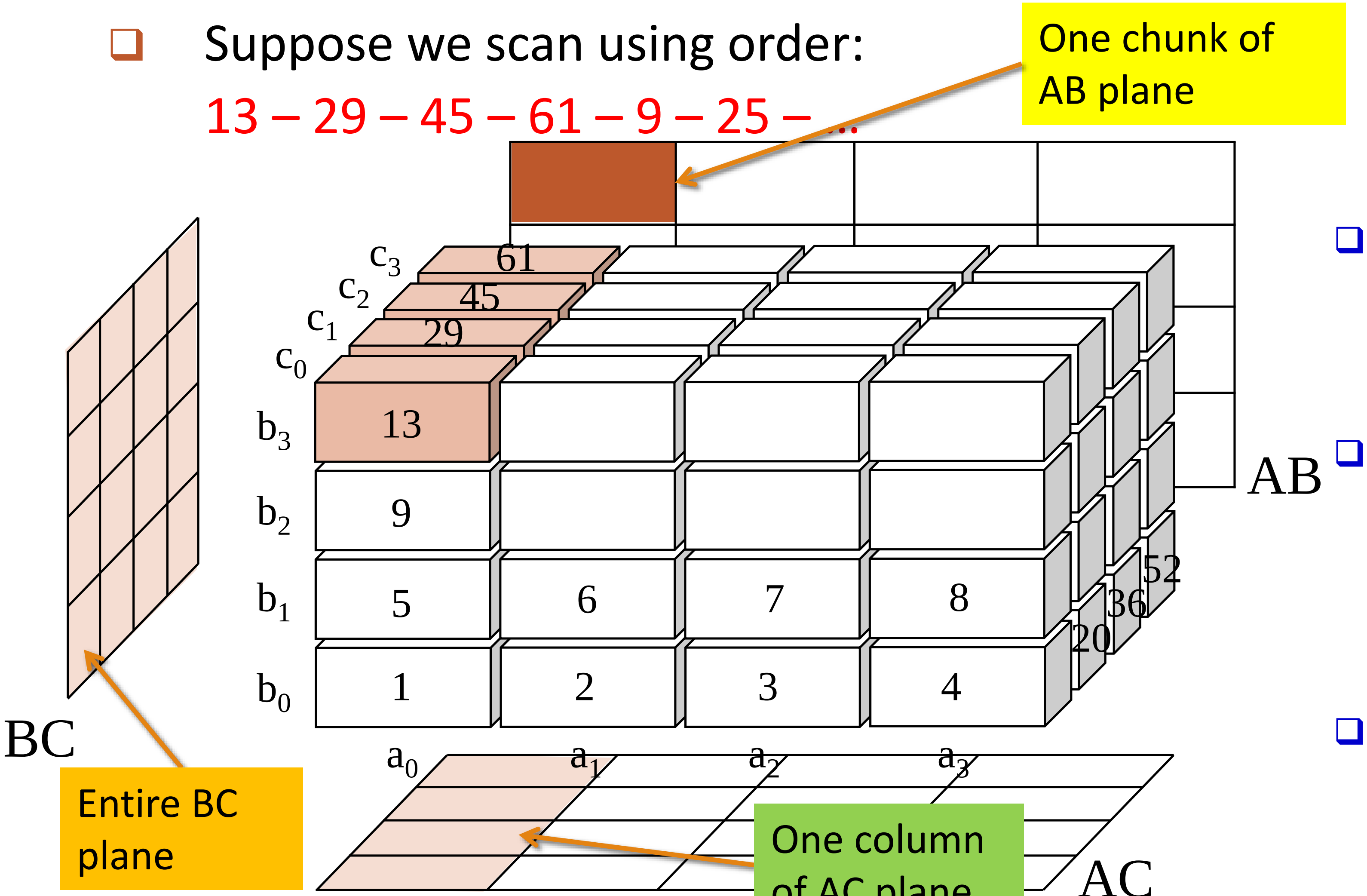


- One **row** of BC plane is fully computed
- One **chunk** of AC plane is fully computed
- All chunks in AB plane are partially computed

- All fully computed chunks can be moved outside main memory
- Partially computed chunks must be stored in the main memory

Cube Computation: Multi-Way Array Aggregation (MOLAP)

- Reducing memory and I/O
  - Suppose we scan using order:
    13 – 29 – 45 – 61 – 9 – 25 – ...



Example:
A: 4000, B: 400, C: 40
Chunk:
1000 x 100 x 10

- memory:
  - 40 x 400 (BC) + 40 x 1000 (AC) + 100 x 1000 (AB) = 156,000 units
- 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 order

Cube Computation: Multi-Way Array Aggregation (MOLAP)

- Reducing memory and I/O
  - 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 order
    - Suppose A>B>C>…

```
for a in A:
    for b in B:
        for c in C:
            …
```

  - Same methodology for computing 2-D and 1-D planes

Cube Computation: Multi-Way Array Aggregation (MOLAP)

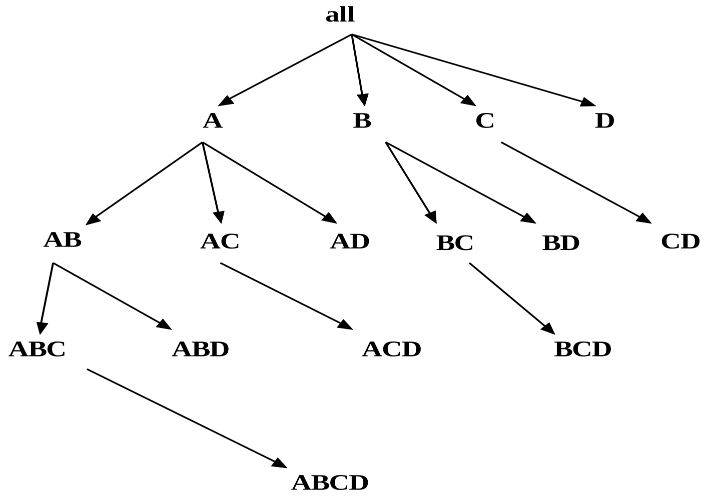
- Comments on the method
  - Input? What format?
  - Output?
  - Pro: Efficient for computing the full cube for a small number of dimensions
  - Con: If there are a large number of dimensions, “top-down” computation and iceberg cube computation methods (e.g., BUC) should be used

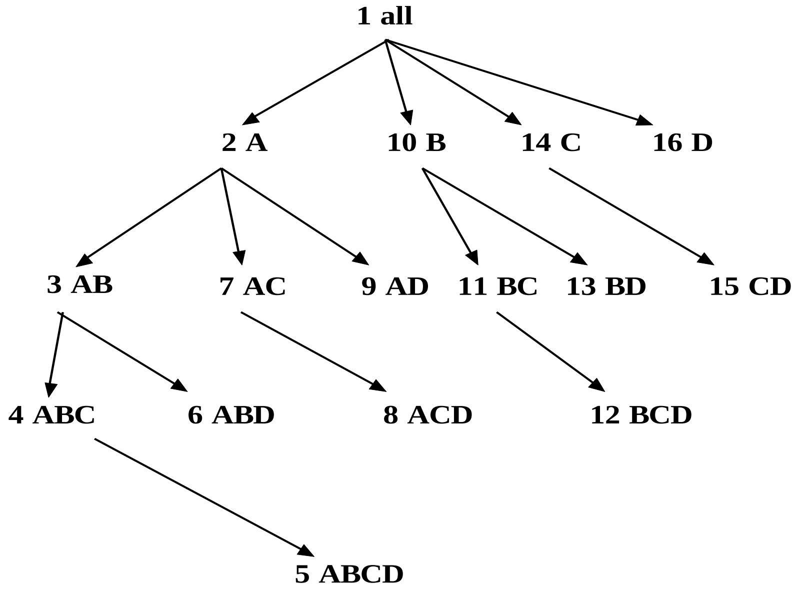
Cube Computation: Computing in Reverse Order

- Iceberg cube computation
- BUC (Beyer & Ramakrishnan, SIGMOD'99)

  BUC: acronym of Bottom-Up (cube) Computation

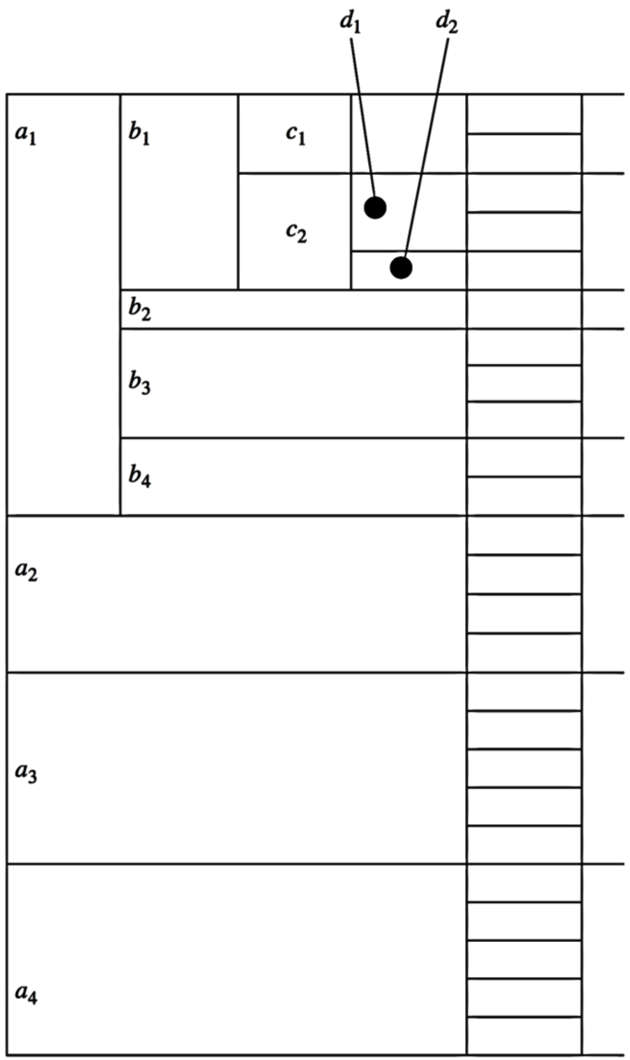
  (Note: It is "top-down" in our view since we put Apex cuboid on the top!)
- Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy *min_sup*, its descendants can be pruned
  - If *minsup* = 1 Þ compute full CUBE!
- No simultaneous aggregation



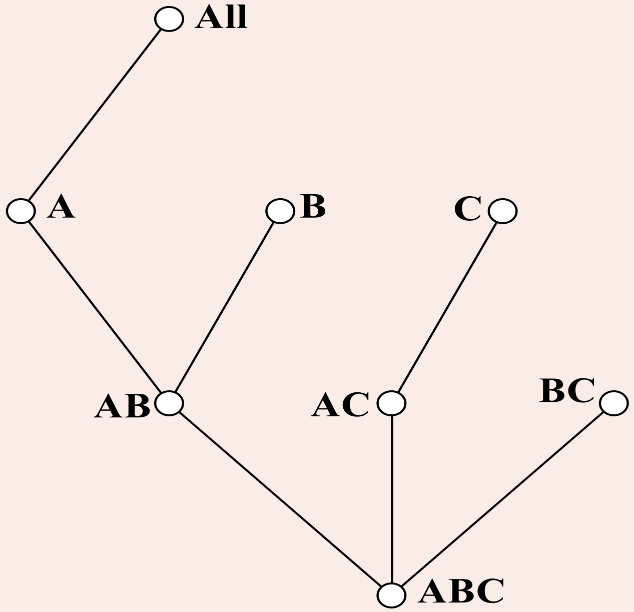


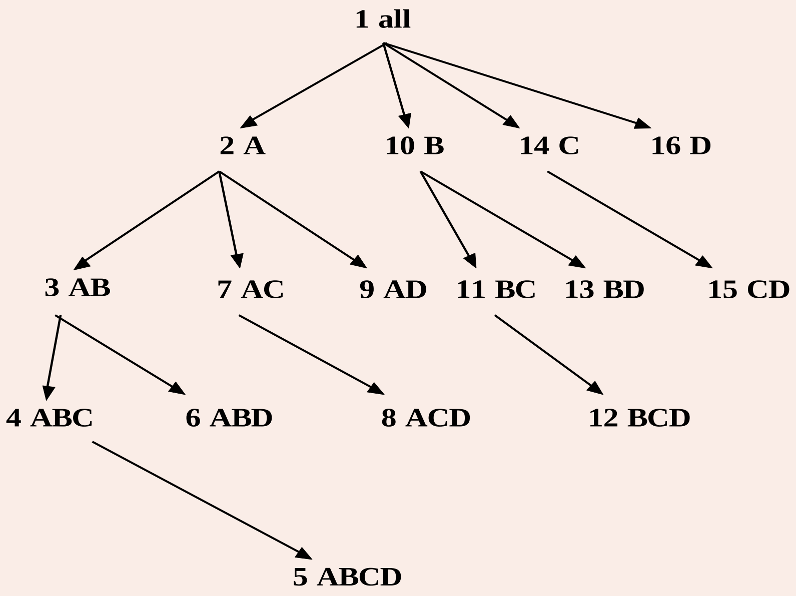
BUC: Partitioning and Aggregating

- Usually, entire data set cannot fit in main memor
- Sort *distinct* values
  - partition into blocks that fit
- Aggregation when sorting
- Continue processing
- Iceberg cube
  - If count of (a1, b1, *, *, *) < min_support
  - No need to sort on C



MultiWay VS BUC

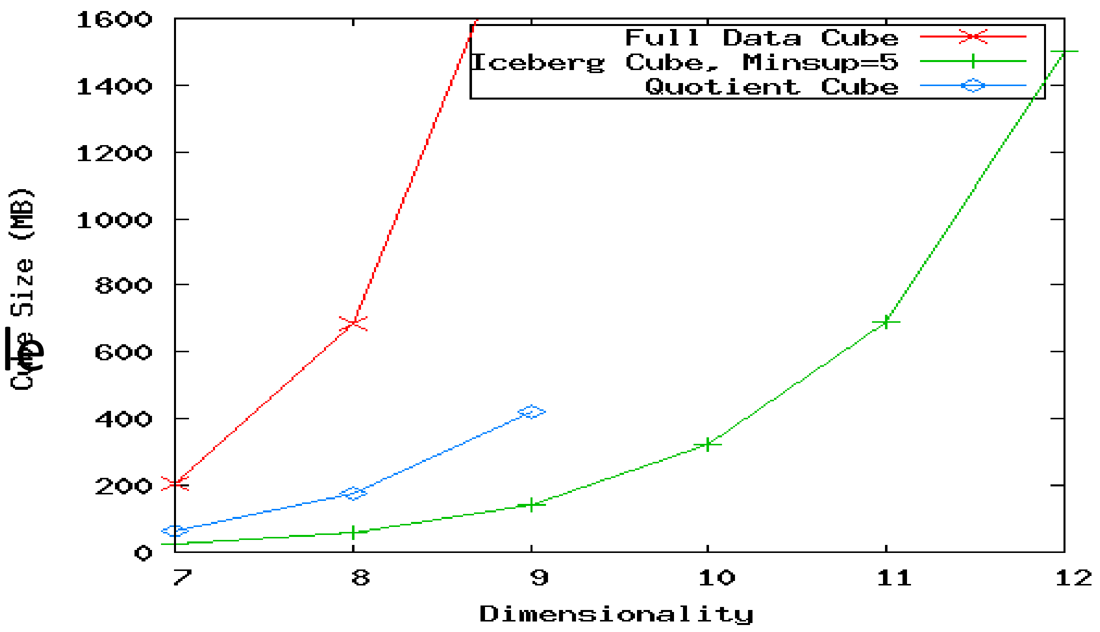
<table>
<tr><th></th><th>multiway</th><th>BUC</th></tr>
<tr><td>Input format</td><td>Multi-dimensional array</td><td>Relational database</td></tr>
<tr><td>Good for</td><td>Full cube</td><td>Iceberg cube</td></tr>
<tr><td>Key idea</td><td>Simultaneously Aggregation</td><td>Partition and sort</td></tr>
<tr><td>Calculation direction</td><td>


</td><td>


</td></tr>
</table>

High-Dimensional OLAP?—The Curse of Dimensionality

- High-D OLAP: Needed in many applications
  - 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

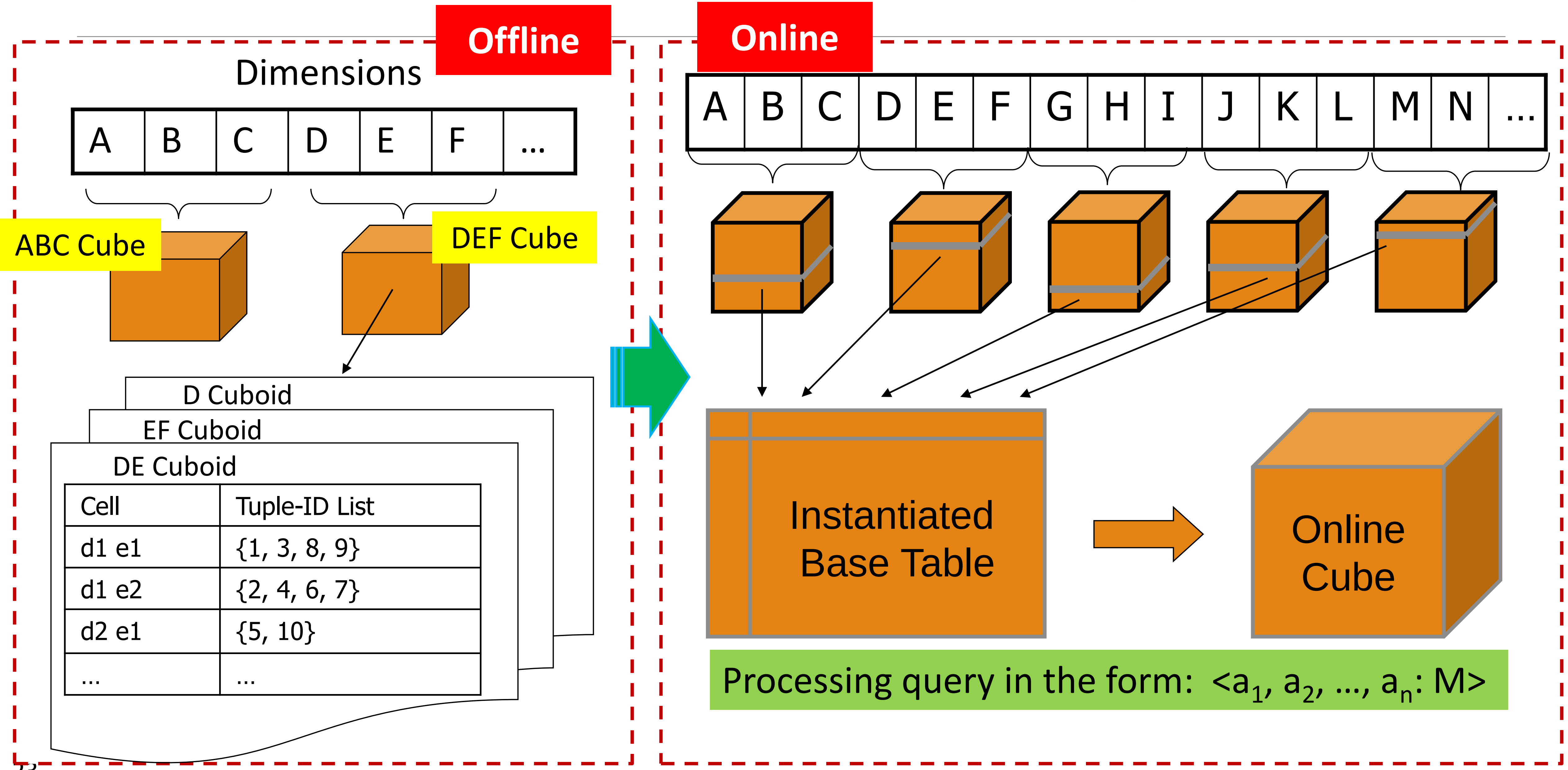


A curse of dimensionality: A database of 600k tuples. Each dimension has cardinality of 100 and *zipf* of 2.

Fast High-D 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 precomputing a set of lower dimensional cubes
  - Online re-construction of original high-dimensional space
  - Lossless reduction

Use Frag-Shells for Online OLAP Query Computation



Cell	Tuple-ID List
d1 e1	{1, 3, 8, 9}
d1 e2	{2, 4, 6, 7}
d2 e1	{5, 10}
…	…

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

- Divide the 5-D table into 2 shell fragments:
  - (A, B, C) and (D, E)
- Build traditional invert index or RID list (**1-D**)

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

Shell Fragment Cubes: Ideas

- Generalize the **1-D** inverted indices to **multi-dimensional** ones in the data cube sense
- Compute all cuboids for data cubes ABC and DE while retaining the inverted indices
  - Ex. shell fragment cube ABC contains 7 cuboids:
    - A, B, C; AB, AC, BC; ABC
- This completes the offline computation
- ID_Measure Table
  - If measures other than count are present, store in *ID_measure* table separate from the shell fragments

tid	count	sum
1	5	70
2	3	10
3	8	20
4	5	40
5	2	30

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

Shell-fragment AB

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

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: $$O\left(T\left\lceil\frac{D}{F}\right\rceil(2^F-1)\right)$$
  - For F < 5, the growth is sub-linear
- 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

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

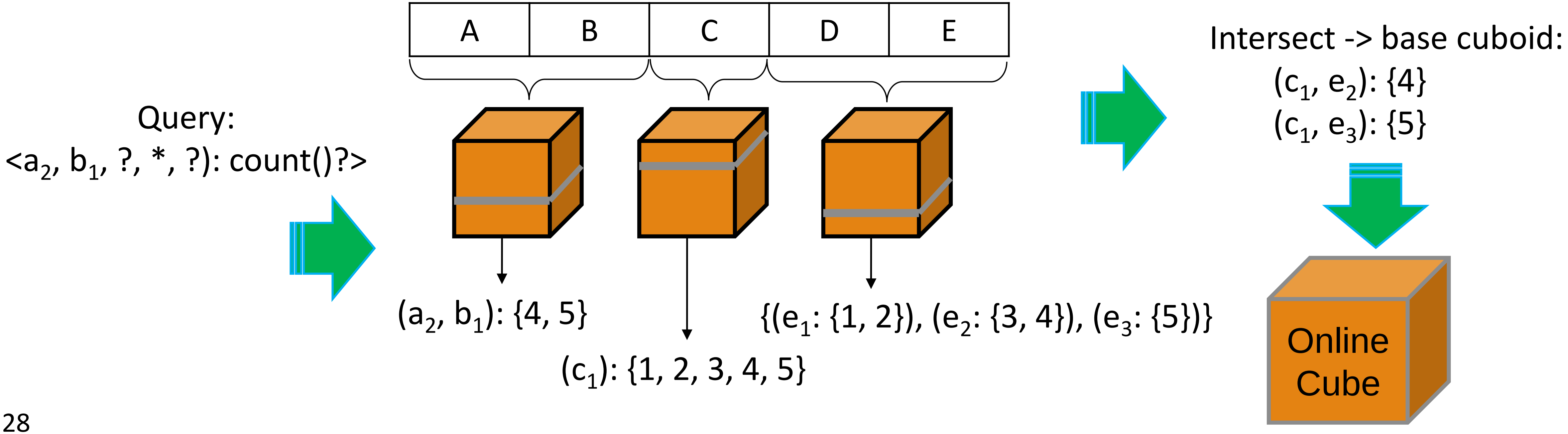
Cell	Intersection	TID List	List Size
a1 b1	1 2 3 ∩ 1 4 5	1	1
a1 b2	1 2 3 ∩ 2 3	2 3	2
a2 b1	4 5 ∩ 1 4 5	4 5	2
a2 b2	4 5 ∩ 2 3	ϕ	0

Online Query Computation with Shell-Fragments

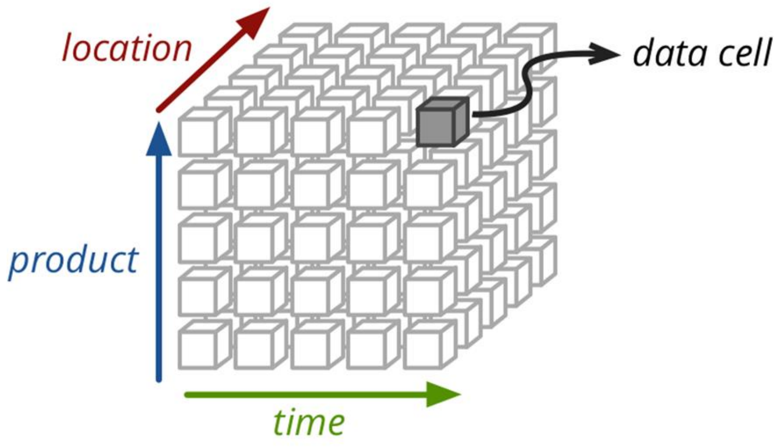
- A query has the general form: $<a_1, a_2, \ldots, a_n: M>$
- Each a_i has 3 possible values (e.g., <3, ?, ?, *, 1: count> returns a 2-D data cube)
  - Instantiated value
  - Aggregate * function
  - Inquire ? Function

Online Query Computation with Shell-Fragments

- Method: Given the materialized fragment cubes, process a query as follows
  - Divide the query into fragments, same as the shell-fragment
  - Fetch the corresponding TID list for each fragment from the fragment cube
  - Intersect the TID lists from each fragment to construct **instantiated base table**
  - Compute the data cube using the base table with any cubing algorithm



Chapter 5: Data Cube Technology



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

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 live, 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: surprise of cell relative to other cells at same level of aggregation
  - InExp: surprise beneath the cell
  - PathExp: surprise beneath cell for each drill-down path
- Computation of exception indicator can be overlapped with cube construction
  - Exceptions can be stored, indexed and retrieved like precomputed aggregates

Examples: Discovery-Driven Data Cubes

item	all
region	all

Sum of sales	month											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Total		1%	-1%	0%	1%	3%	-1	-9%	-1%	2%	-4%	3%

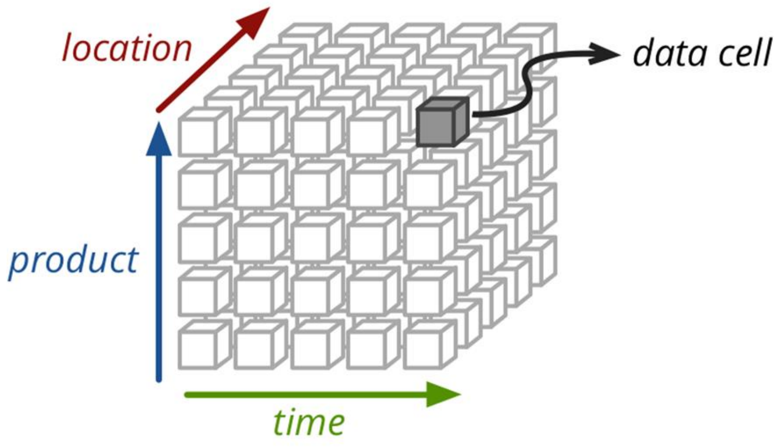
Avg sales	month											
item	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Sony b/w printer		9%	-8%	2%	-5%	14%	-4%	0%	41%	-13%	-15%	-11%
Sony color printer		0%	0%	3%	2%	4%	-10%	-13%	0%	4%	-6%	4%
HP b/w printer		-2%	1%	2%	3%	8%	0%	-12%	-9%	3%	-3%	6%
HP color printer		0%	0%	-2%	1%	0%	-1%	-7%	-2%	1%	-5%	1%
IBM home computer		1%	-2%	-1%	-1%	3%	3%	-10%	4%	1%	-4%	-1%
IBM laptop computer		0%	0%	-1%	3%	4%	2%	-10%	-2%	0%	-9%	3%
Toshiba home computer		-2%	-5%	1%	1%	-1%	1%	5%	-3%	-5%	-1%	-1%
Toshiba laptop computer		1%	0%	3%	0%	-2%	-2%	-5%	3%	2%	-1%	0%
Logitech mouse		3%	-2%	-1%	0%	4%	6%	-11%	2%	1%	-4%	0%
Ergo-way mouse		0%	0%	2%	3%	1%	-2%	-2%	-5%	0%	-5%	8%

item	IBM home computer

Avg sales	month											
region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
North		-1%	-3%	-1%	0%	3%	4%	-7%	1%	0%	-3%	-3%
South		-1%	1%	-9%	6%	-1%	-39%	9%	-34%	4%	1%	7%
East		-1%	-2%	2%	-3%	1%	18%	-2%	11%	-3%	-2%	-1%
West		4%	0%	-1%	-3%	5%	1%	-18%	8%	5%	-8%	1%

33

Chapter 5: Data Cube Technology



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

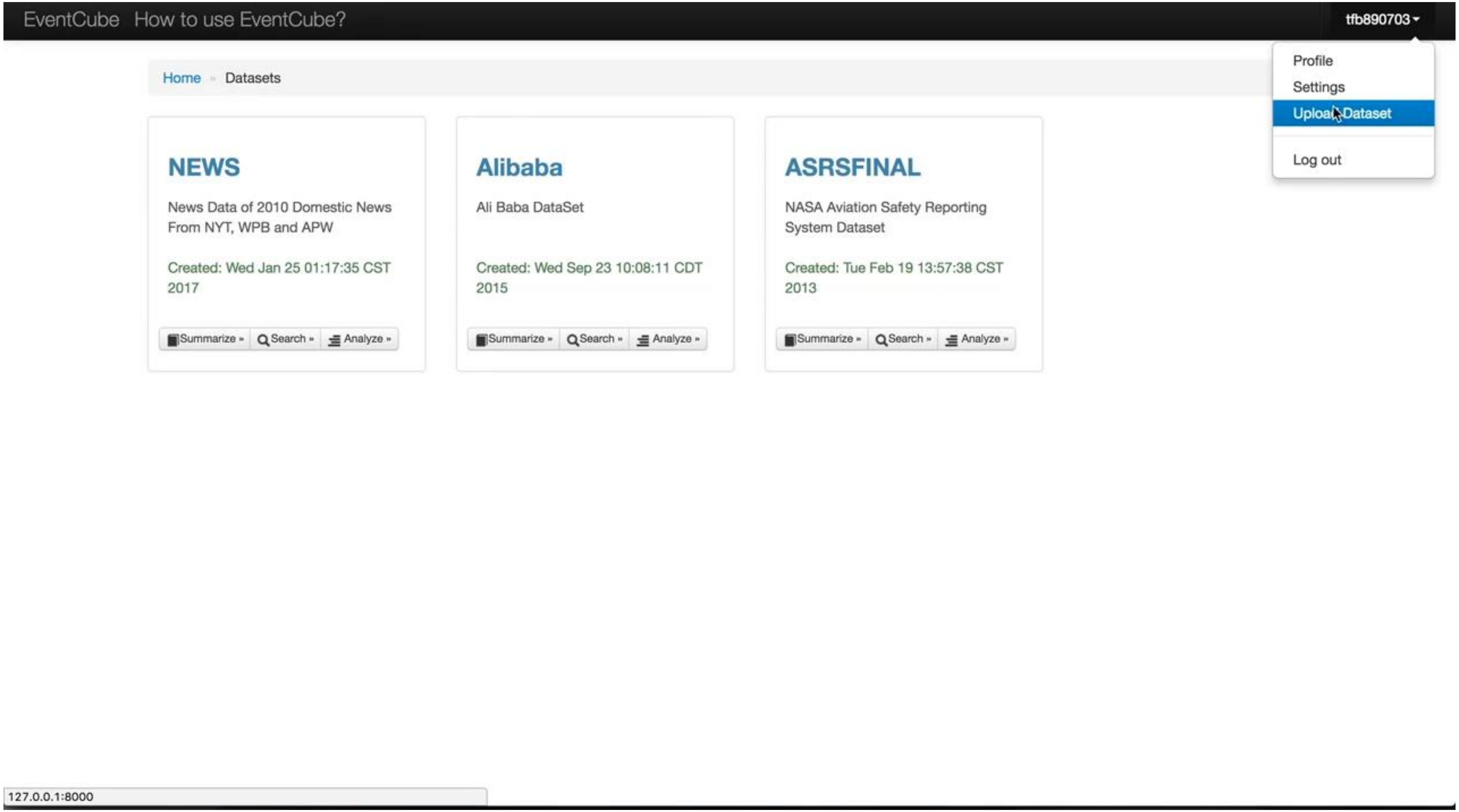
Data Cube Technology: Summary

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

Data Cube Technology: Summary

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

Text Cube


EventCube How to use EventCube?
tfb890703
Profile
Settings
Upload Dataset
Log out
Home » Datasets
NEWS
News Data of 2010 Domestic News From NYT, WPB and APW
Created: Wed Jan 25 01:17:35 CST 2017
Summarize » Search » Analyze »
Alibaba
Ali Baba DataSet
Created: Wed Sep 23 10:08:11 CDT 2015
Summarize » Search » Analyze »
ASRSFINAL
NASA Aviation Safety Reporting System Dataset
Created: Tue Feb 19 13:57:38 CST 2013
Summarize » Search » Analyze »
127.0.0.1:8000

Data Cube Technology: References (I)

- 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

Data Cube Technology: References (II)

- 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*