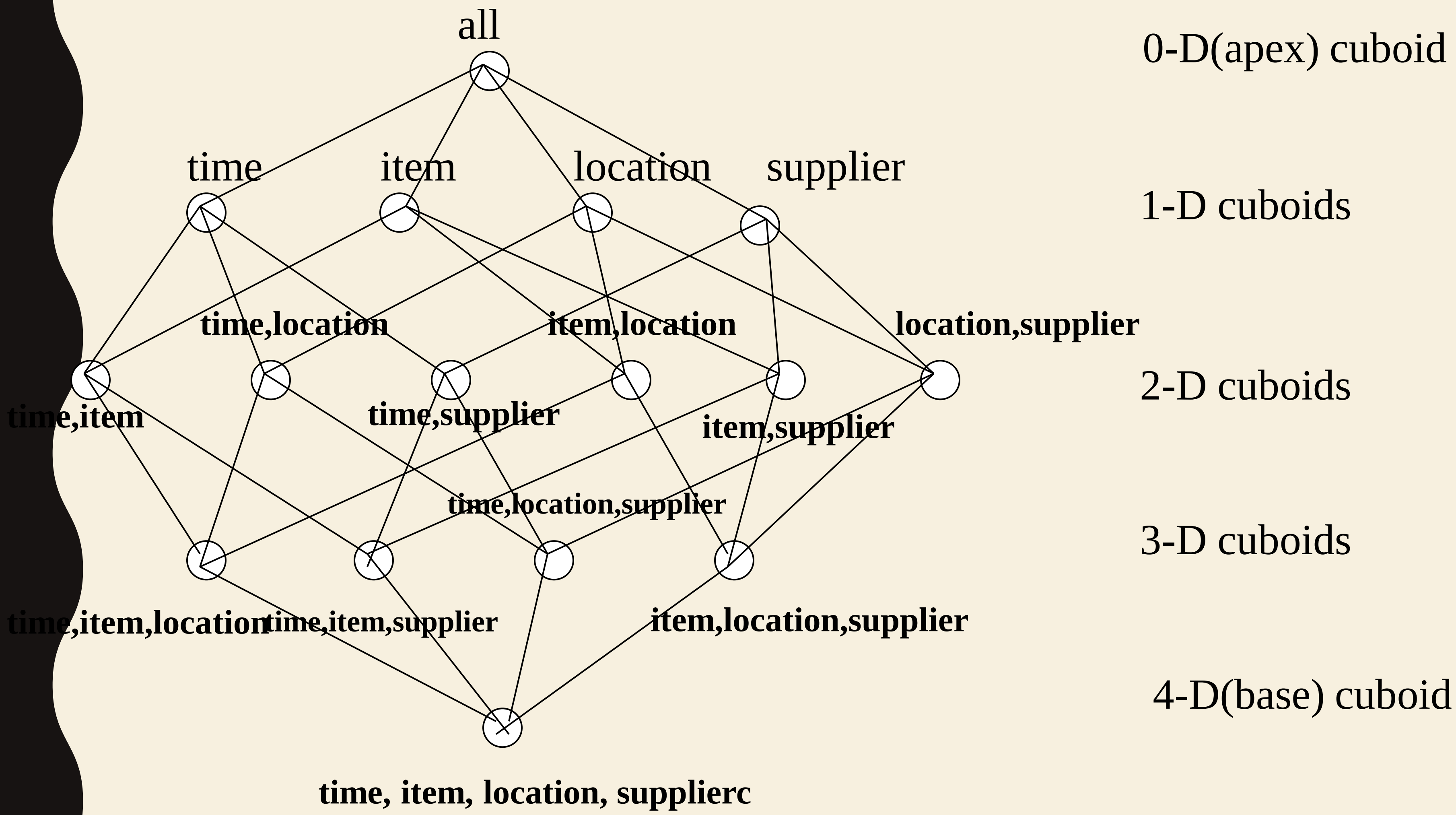
DSC603-DATA MINING

DATA CUBE TECHNOLOGY AND COMPUTATION

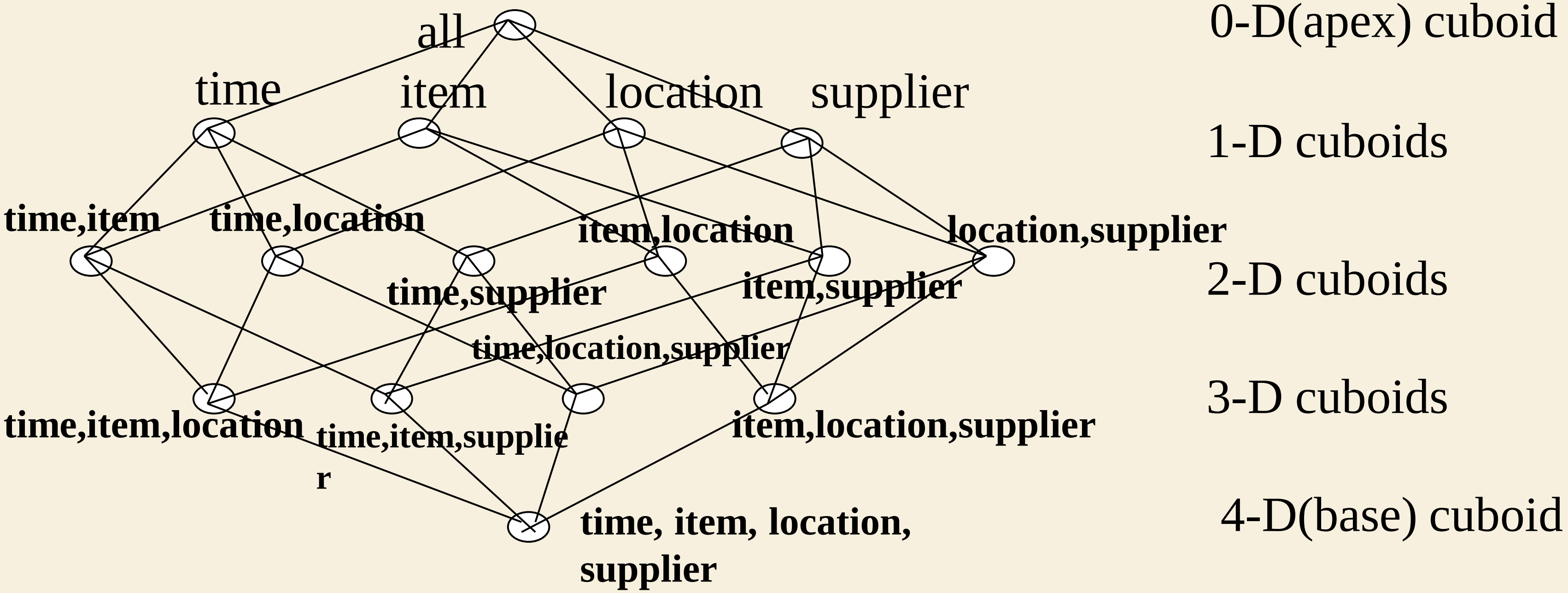
DATA CUBE TECHNOLOGY

- **Data Cube Computation: Preliminary Concepts**
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring 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
  1. (9/15, milk, Urbana, Dairy_land)
  2. (9/15, milk, Urbana, *)
  3. (*, milk, Urbana, *)
  4. (*, milk, Urbana, *)
  5. (*, milk, Chicago, *)
  6. (*, milk, *, *)

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

- Computing *only* the cuboid cells whose measure satisfies the iceberg condition
- Only a small portion of cells may be "above the water'' in a sparse cube
- Avoid explosive growth: A cube with 100 dimensions
  - 2 base cells: (a1, a2, …., a100), (b1, b2, …, b100)
  - How many aggregate cells if "having count >= 1"?
  - What about "having count >= 2"?

ICEBERG CUBE, CLOSED CUBE & CUBE SHELL

- Is iceberg cube good enough?
  - 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 have if having count(*) >= 10? Hint: A huge but tricky number!
- Close cube:
  - Closed cell c: if there exists no cell d, s.t. d is a descendant of c, and d has the same measure value as c.
  - Closed cube: a cube consisting of only closed cells
  - What is the closed cube of the above base cuboid? Hint: only 3 cells
- Cube Shell
  - Precompute only the cuboids involving a small # of dimensions, e.g., 3
  - More dimension combinations will need to be computed on the fly

For $(A_1, A_2, \ldots A_{10})$, how many combinations to compute?

ROADMAP FOR EFFICIENT COMPUTATION

- General cube computation heuristics
- Computing full/iceberg cubes: 3 methodologies
  - Bottom-Up method: Multi-Way array aggregation (Zhao and all, SIGMOD'97)
  - Top-down method:
    - BUC (Beyer and all, SIGOD'99)
    - H-cubing technique
  - Integrating Top-Down and Bottom-Up method:
    - Star-cubing algorithm (Xin and all in VLDB'03)
- High-dimensional OLAP: A Minimal Cubing Approach
- Computing alternative kinds of cubes:
  - Partial cube, closed cube, approximate cube, etc.

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

DATA CUBE TECHNOLOGY

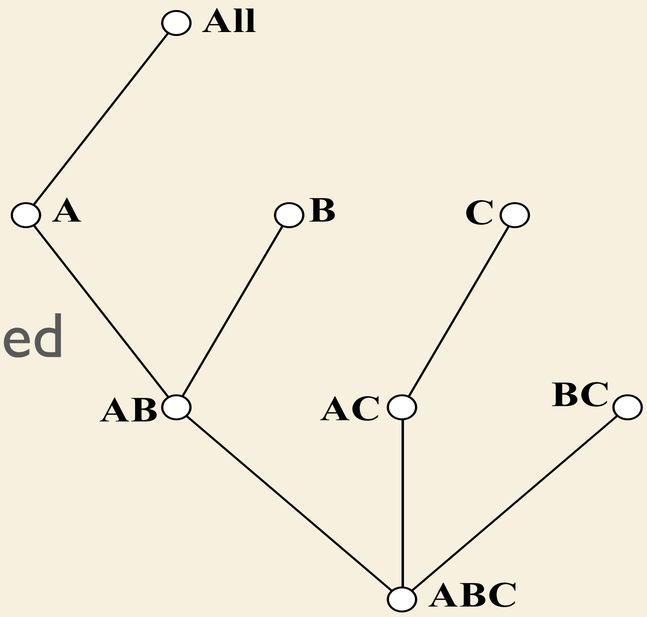
- Data Cube Computation: Preliminary Concepts
- **Data Cube Computation Methods**
- Processing Advanced Queries by Exploring Data Cube Technology
- Multidimensional Data Analysis in Cube Space
- Summary

DATA CUBE COMPUTATION METHODS

- **Multi-Way Array Aggregation**
- BUC
- Star-Cubing
- High-Dimensional OLAP

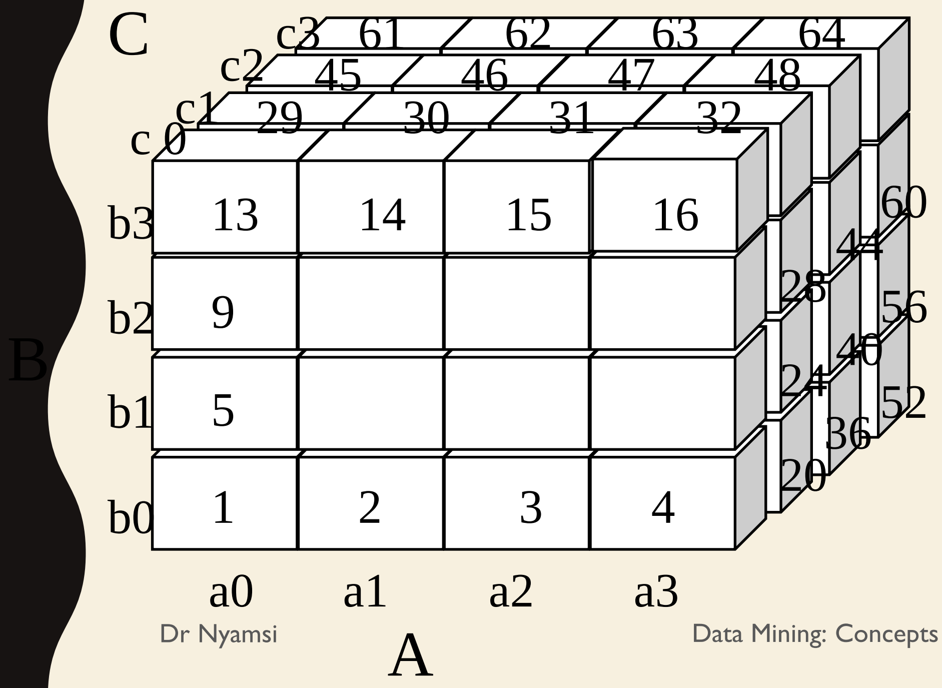
MULTI-WAY ARRAY AGGREGATION

- Array-based "bottom-up" algorithm
- Using multi-dimensional chunks
- No direct tuple comparisons
- Simultaneous aggregation on multiple dimensions
- Intermediate aggregate values are re-used for computing ancestor cuboids
- Cannot do *Apriori* pruning: No iceberg optimization



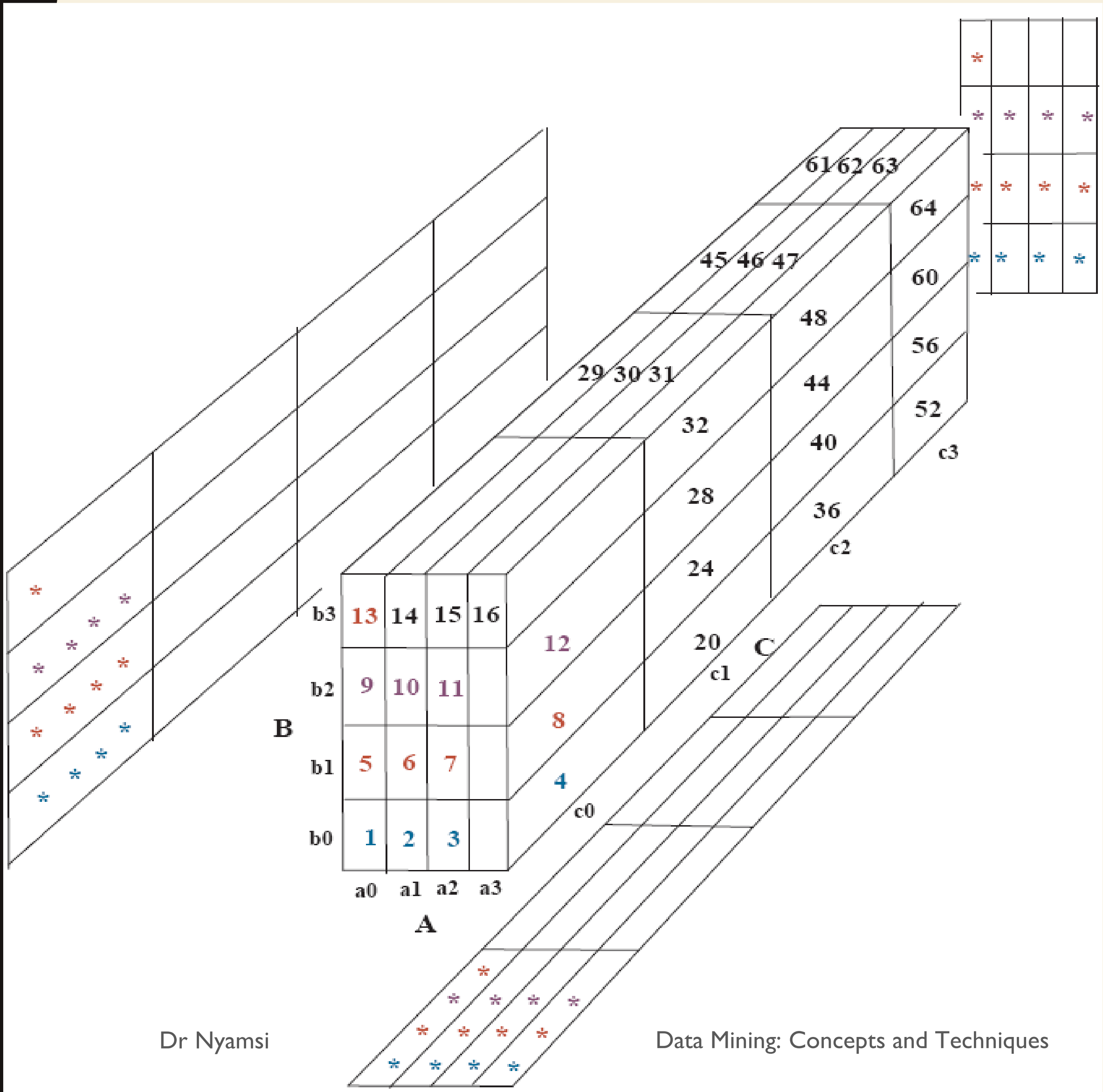
MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION (MOLAP)

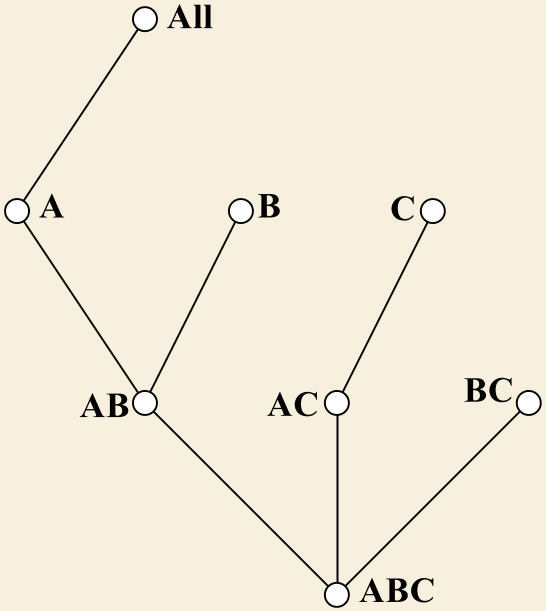
- Partition arrays into chunks (a small subcube which fits in memory).
- Compressed sparse array addressing: (chunk_id, offset)
- Compute aggregates in "multiway" by visiting cube cells in the order which minimizes the # of times to visit each cell and reduces memory access and storage cost.



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

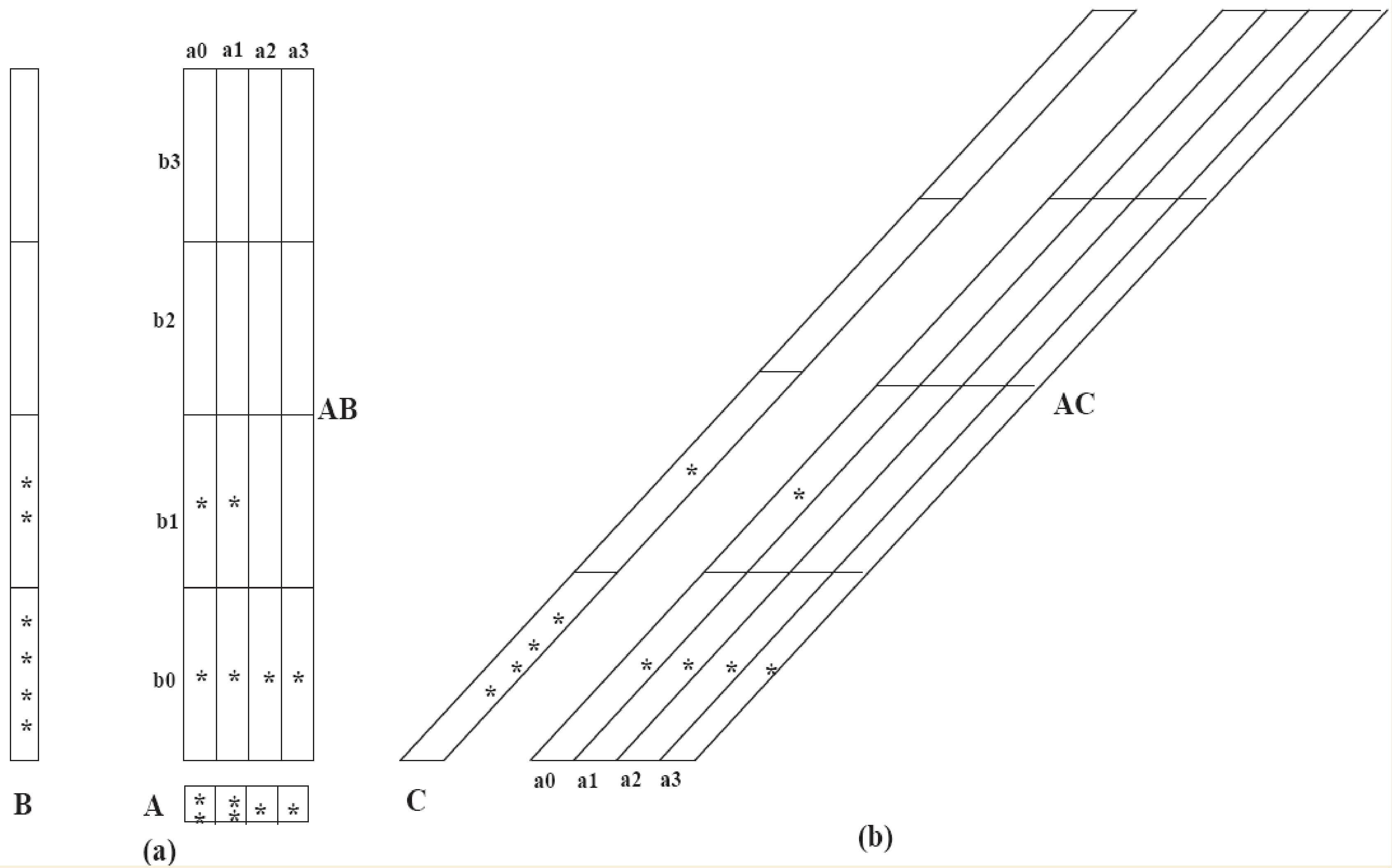
MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION (3-D TO 2-D)





- The best order is the one that minimizes the memory requirement and reduced I/Os

MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION (2-D TO 1-D)



MULTI-WAY ARRAY AGGREGATION FOR CUBE COMPUTATION (METHOD SUMMARY)

- Method: the planes should be sorted and computed according to their size in ascending order
  - Idea: keep the smallest plane in the main memory, fetch and compute only one chunk at a time for the largest plane
- Limitation of the method: computing well only for a small number of dimensions
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods can be explored

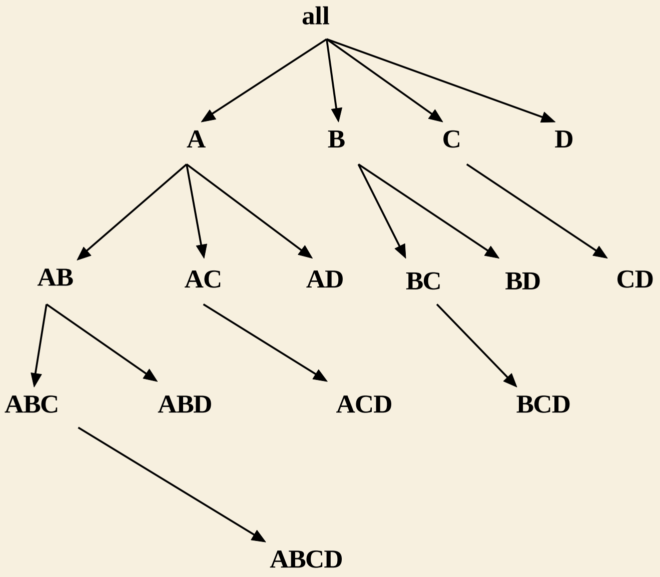
DATA CUBE COMPUTATION METHODS

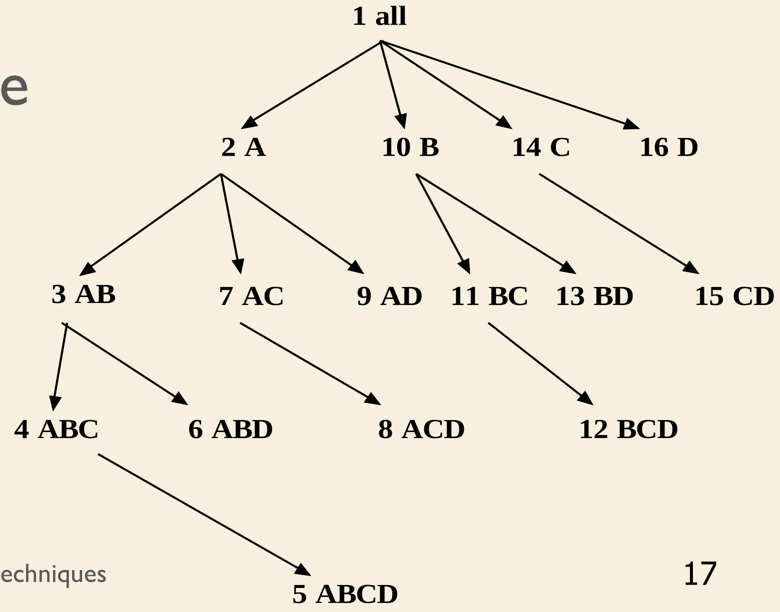
- Multi-Way Array Aggregation
- **BUC**
- Star-Cubing
- High-Dimensional OLAP

BOTTOM-UP COMPUTATION (BUC)

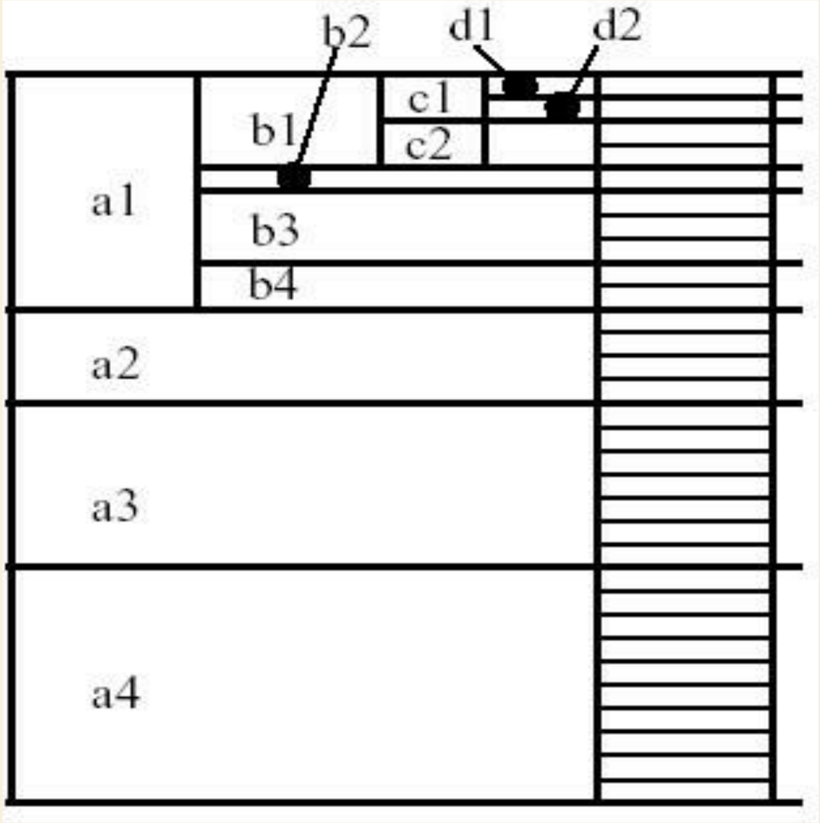
- BUC (Beyer & Ramakrishnan, SIGMOD'99)
- Bottom-up cube computation

  (Note: top-down in our view!)
- Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy *min_sup*, its descendants can be pruned
  - If *minsup* = 1 ⇒ compute full CUBE!
- No simultaneous aggregation





BUC: PARTITIONING



- Usually, entire data set 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
    - Can't do holistic aggregates anymore!

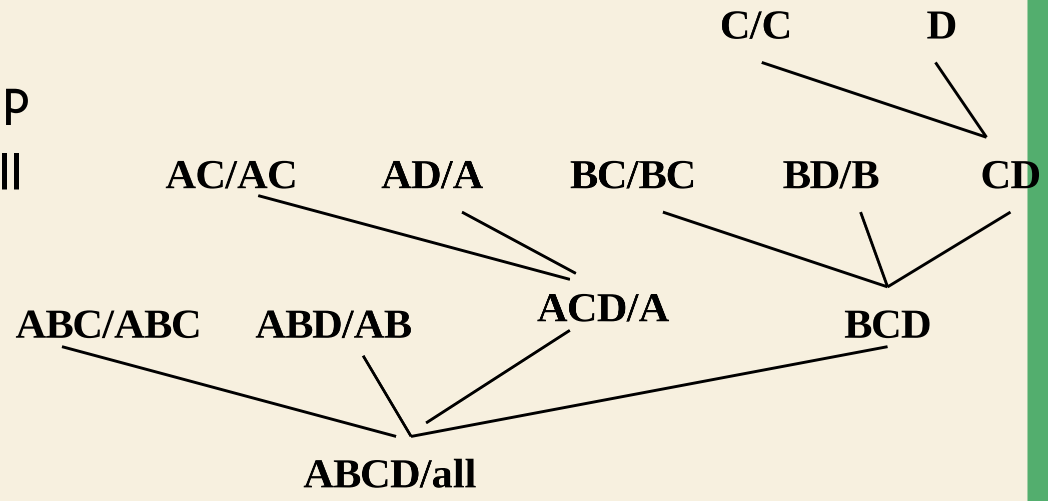
DATA CUBE COMPUTATION METHODS

- Multi-Way Array Aggregation
- BUC
- **Star-Cubing**
- High-Dimensional OLAP

STAR-CUBING: AN INTEGRATING METHOD

- D. Xin, J. Han, X. Li, B. W. Wah, Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration, VLDB'03
- ***Explore shared dimensions***
  - E.g., dimension A is the shared dimension of ACD and AD
  - ABD/AB means cuboid ABD has shared dimensions AB
- Allows for shared computations
  - e.g., cuboid AB is computed simultaneously as ABD

- Aggregate in a top-down manner but with the bottom-up sub-layer underneath which will allow Apriori pruning
- Shared dimensions grow in bottom-up fashion

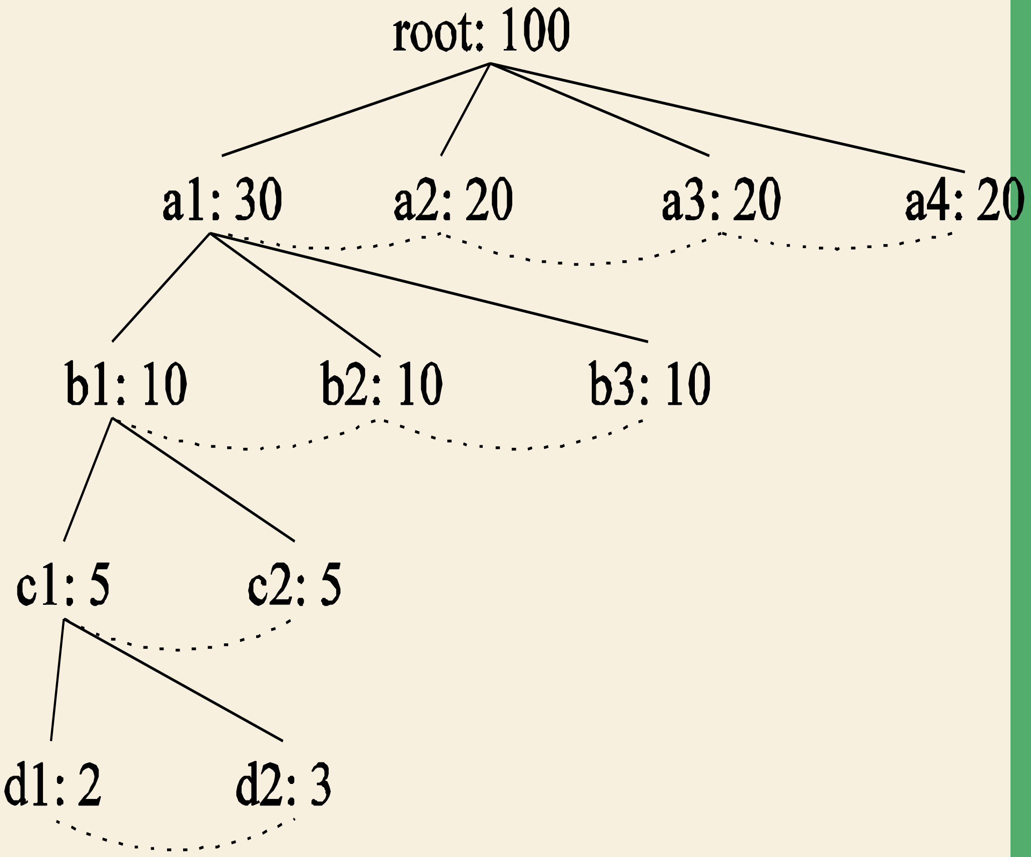


ICEBERG PRUNING IN SHARED DIMENSIONS

- Anti-monotonic property of shared dimensions
  - If the measure is *anti-monotonic*, and if the aggregate value on a shared dimension does not satisfy the *iceberg condition*, then all the cells extended from this shared dimension cannot satisfy the condition either
- Intuition: if we can compute the shared dimensions before the actual cuboid, we can use them to do `Apriori` pruning
- Problem: how to prune while still aggregate simultaneously on multiple dimensions?

CELL TREES

- Use a tree structure similar to H-tree to represent cuboids
- Collapses common prefixes to save memory
- Keep count at node
- Traverse the tree to retrieve a particular tuple



STAR ATTRIBUTES AND STAR NODES

- Intuition: If a single-dimensional aggregate on an attribute value *p* does not satisfy the iceberg condition, it is useless to distinguish them during the iceberg computation
  - E.g., b_2, b_3, b_4, c_1, c_2, c_4, d_1, d_2, d_3
- Solution: Replace such attributes by a *. Such attributes are *star attributes,* and the corresponding nodes in the cell tree are *star nodes*

A	B	C	D	Count
a1	b1	c1	d1	1
a1	b1	c4	d3	1
a1	b2	c2	d2	1
a2	b3	c3	d4	1
a2	b4	c3	d4	1

EXAMPLE: STAR REDUCTION

- Suppose minsup = 2
- Perform one-dimensional aggregation. Replace attribute values whose count < 2 with *. And collapse all *'s together
- Resulting table has all such attributes replaced with the star-attribute
- With regards to the iceberg computation, this new table is a *lossless compression* of the original table

A	B	C	D	Count
a1	b1	*	*	1
a1	b1	*	*	1
a1	*	*	*	1
a2	*	c3	d4	1
a2	*	c3	d4	1

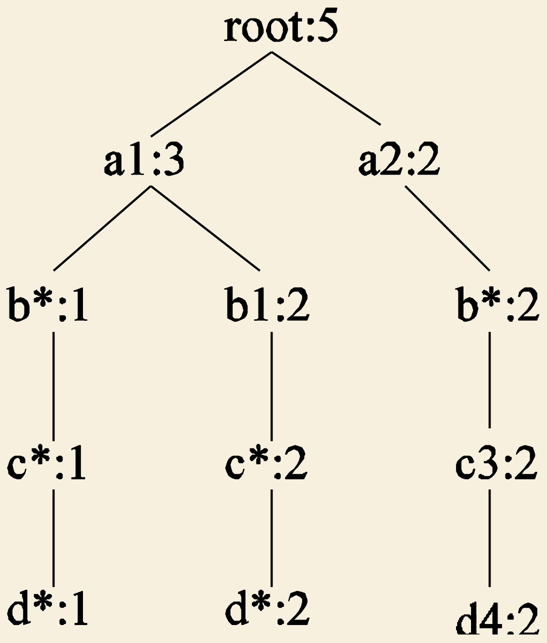
↓

A	B	C	D	Count
a1	b1	*	*	2
a1	*	*	*	1
a2	*	c3	d4	2

STAR TREE

- Given the new compressed table, it is possible to construct the corresponding cell tree—called <u>star tree</u>
- Keep a <u>star table</u> at the side for easy lookup of star attributes
- The star tree is a *lossless compression* of the original cell tree

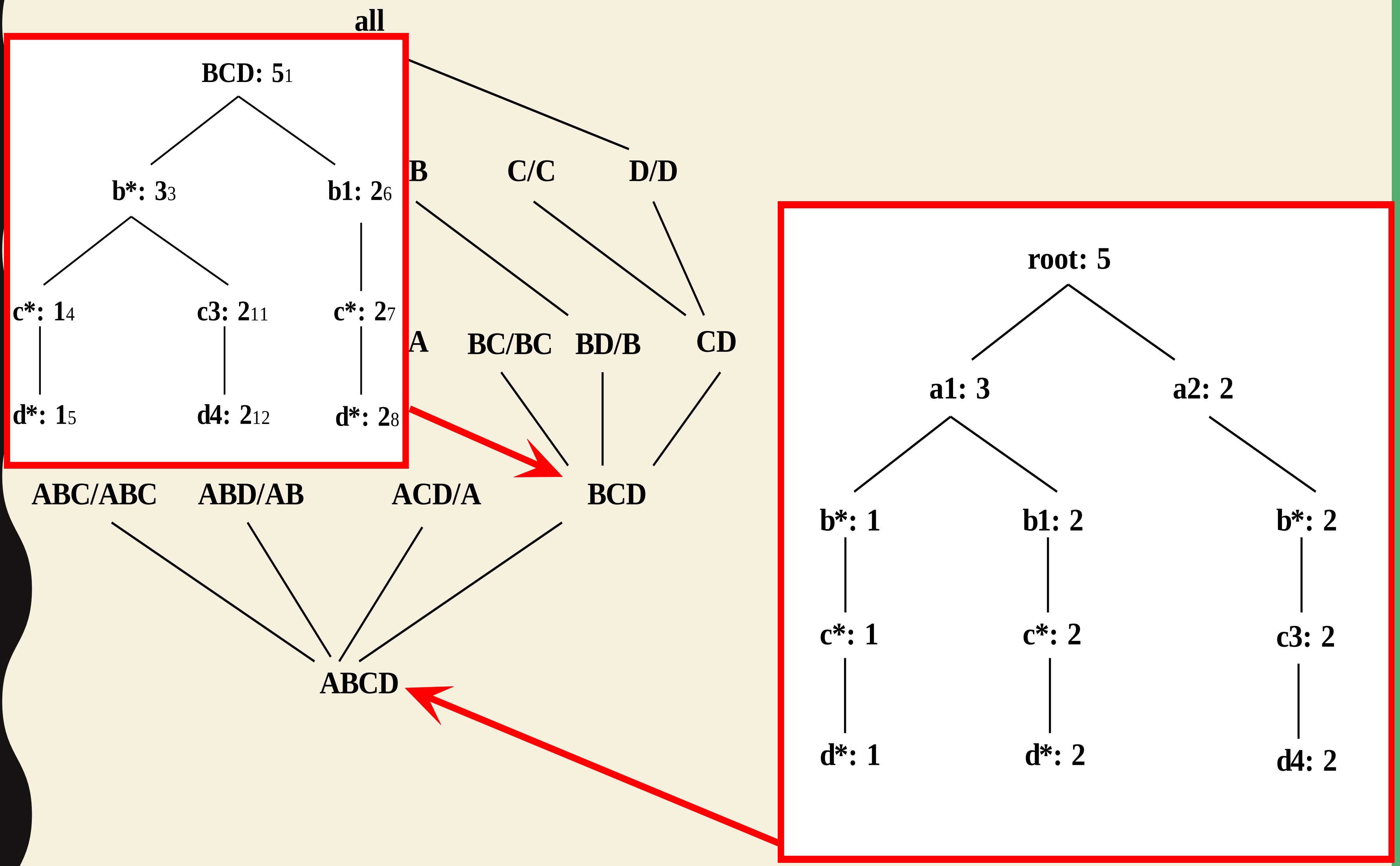
A	B	C	D	Count
a1	b1	*	*	2
a1	*	*	*	1
a2	*	c3	d4	2



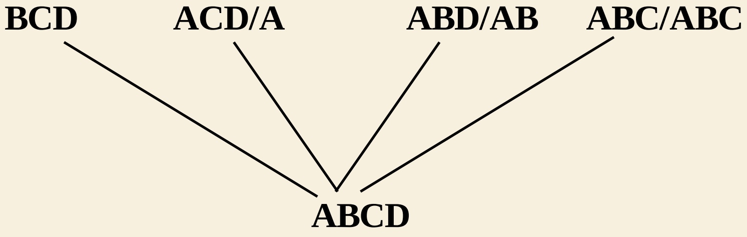
Star Table

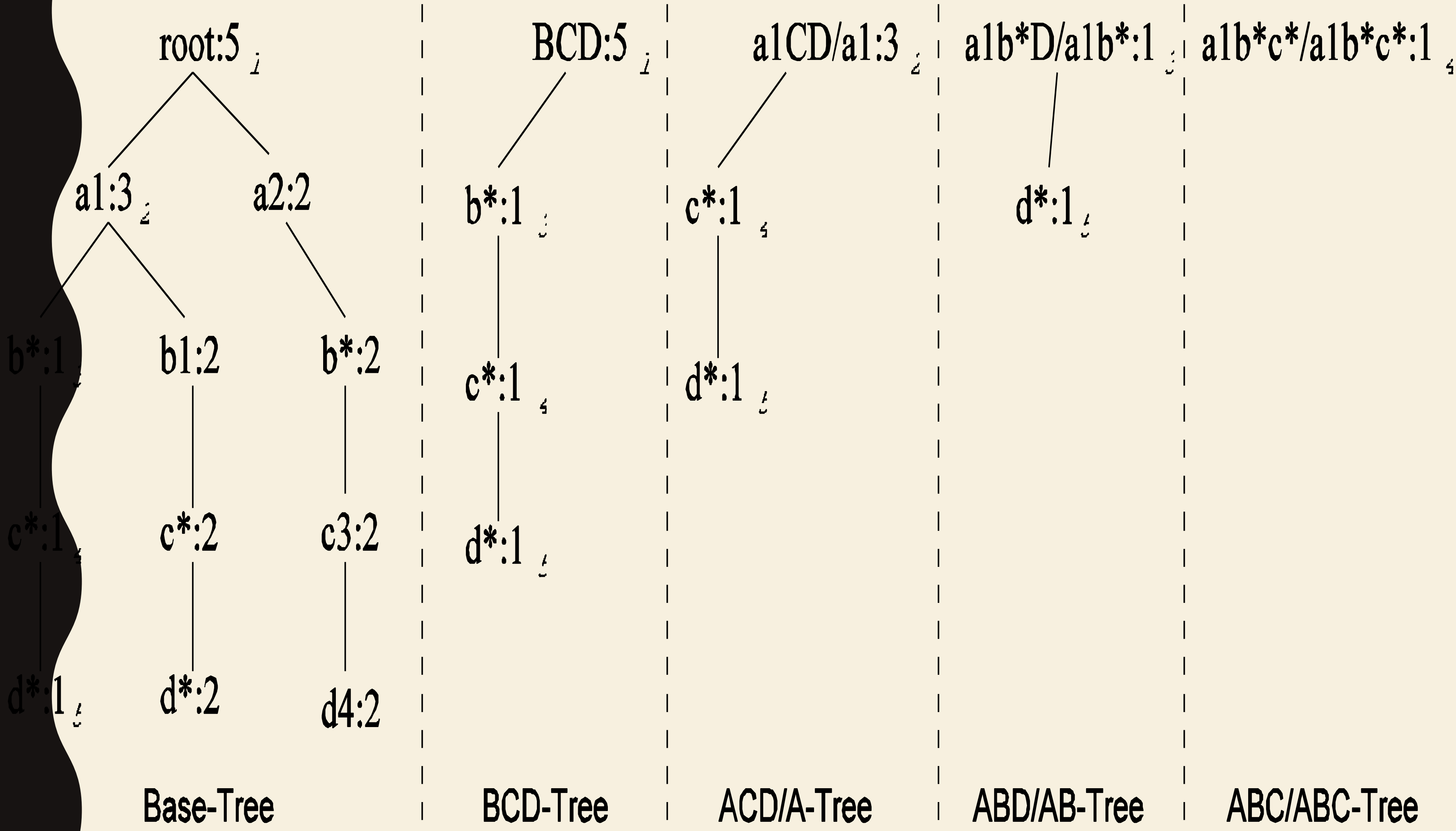
b2 → *
b3 → *
b4 → *
c1 → *
c2 → *
d1 → *
…

STAR-CUBING ALGORITHM—DFS ON LATTICE TREE

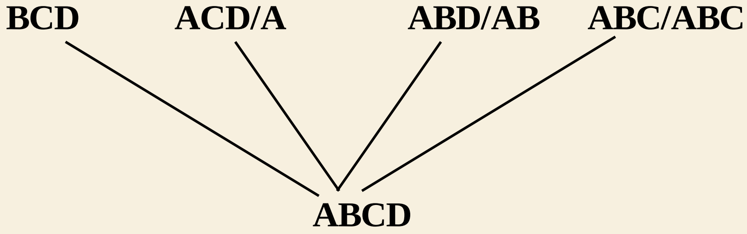


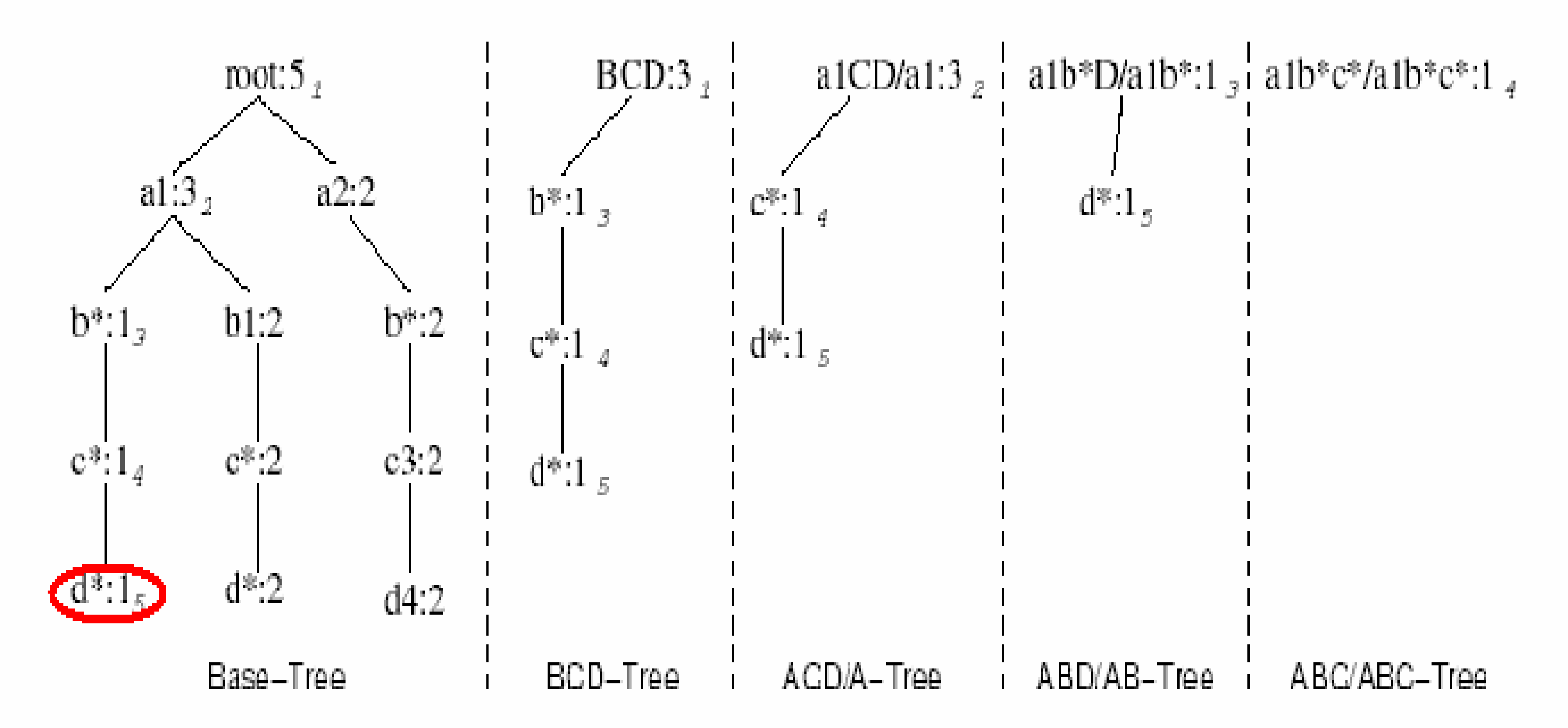
MULTI-WAY AGGREGATION





STAR-CUBING ALGORITHM
DFS ON STAR-TREE





MULTI-WAY STAR-TREE AGGREGATION

- Start depth-first search at the root of the base star tree
- At each new node in the DFS, create corresponding star tree that are descendants of the current tree according to the integrated traversal ordering
  - E.g., in the base tree, when DFS reaches `a1`, the ACD/A tree is created
  - When DFS reaches `b*`, the ABD/AD tree is created
- The counts in the base tree are carried over to the new trees
- When DFS reaches a leaf node (e.g., `d*`), start backtracking
- On every backtracking branch, the count in the corresponding trees are output, the tree is destroyed, and the node in the base tree is destroyed

MULTI-WAY STAR-TREE AGGREGATION

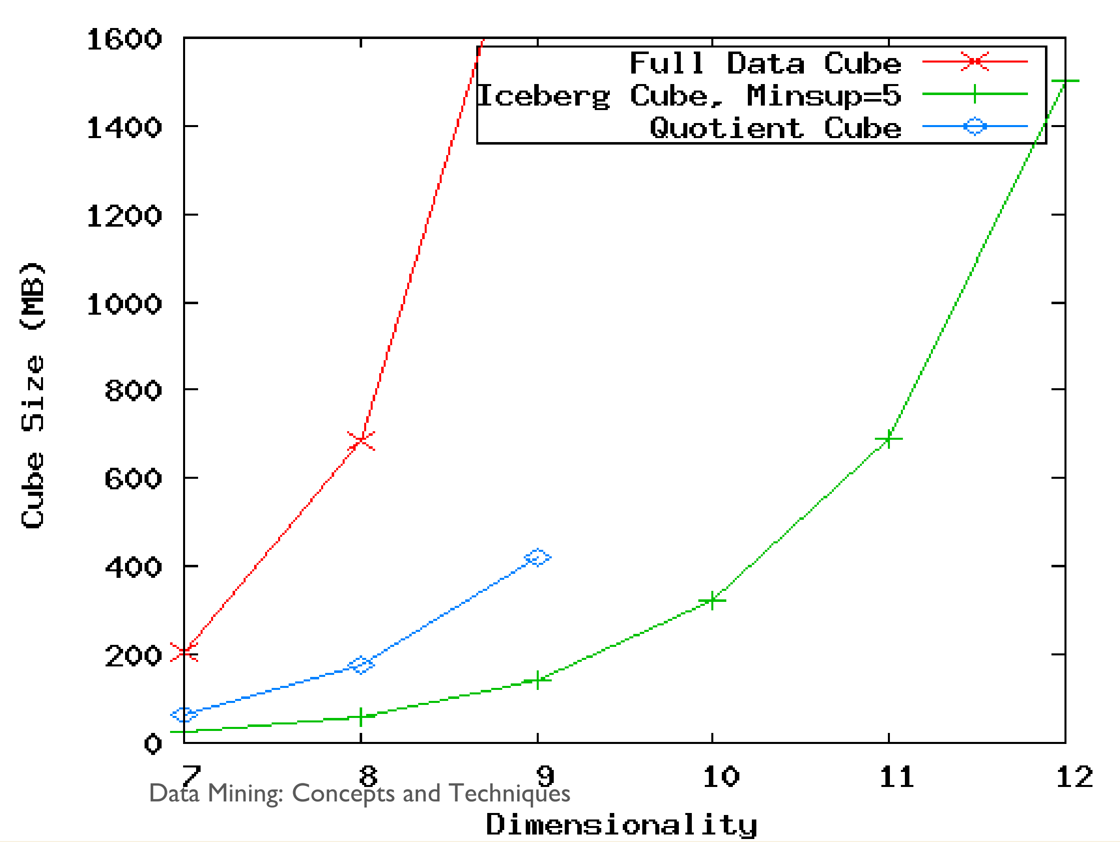
- Example
  - When traversing from `d*` back to `c*`, the `a1b*c*/a1b*c*` tree is output and destroyed
  - When traversing from `c*` back to `b*`, the `a1b*D/a1b*` tree is output and destroyed
  - When at `b*`, jump to `b1` and repeat similar process

DATA CUBE COMPUTATION METHODS

- Multi-Way Array Aggregation
- BUC
- Star-Cubing
- **High-Dimensional OLAP**

THE CURSE OF DIMENSIONALITY

- None of the previous cubing method can handle high dimensionality!
- A database of 600k tuples. Each dimension has cardinality of 100 and *zipf* of 2.



MOTIVATION OF HIGH-D OLAP

- Challenge to current cubing methods:
  - The "curse of dimensionality" problem
  - Iceberg cube and compressed cubes: only delay the inevitable explosion
  - Full materialization: still significant overhead in accessing results on disk
- High-D OLAP is needed in applications
  - Science and engineering analysis
  - Bio-data analysis: thousands of genes
  - Statistical surveys: hundreds of variables

FAST HIGH-D OLAP WITH MINIMAL CUBING

- Observation: OLAP occurs only on a small subset of dimensions at a time
- Semi-Online Computational Model

1. Partition the set of dimensions into **shell fragments**
2. Compute data cubes for each shell fragment while retaining **inverted indices** or **value-list indices**
3. Given the pre-computed **fragment cubes**, dynamically compute cube cells of the high-dimensional data cube *online*

PROPERTIES OF PROPOSED METHOD

- Partitions the data vertically
- Reduces high-dimensional cube into a set of lower dimensional cubes
- Online re-construction of original high-dimensional space
- Lossless reduction
- Offers tradeoffs between the amount of pre-processing and the speed of online computation

EXAMPLE COMPUTATION

- 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 dimensions into 2 shell fragments:
  - (`A, B, C`) and (`D, E`)

1-D INVERTED INDICES

- Build traditional invert index or RID list

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

- For example, shell fragment cube ABC contains 7 cuboids:
  - A, B, C
  - AB, AC, BC
  - ABC
- This completes the offline computation stage

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	⊗	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

ID_MEASURE TABLE

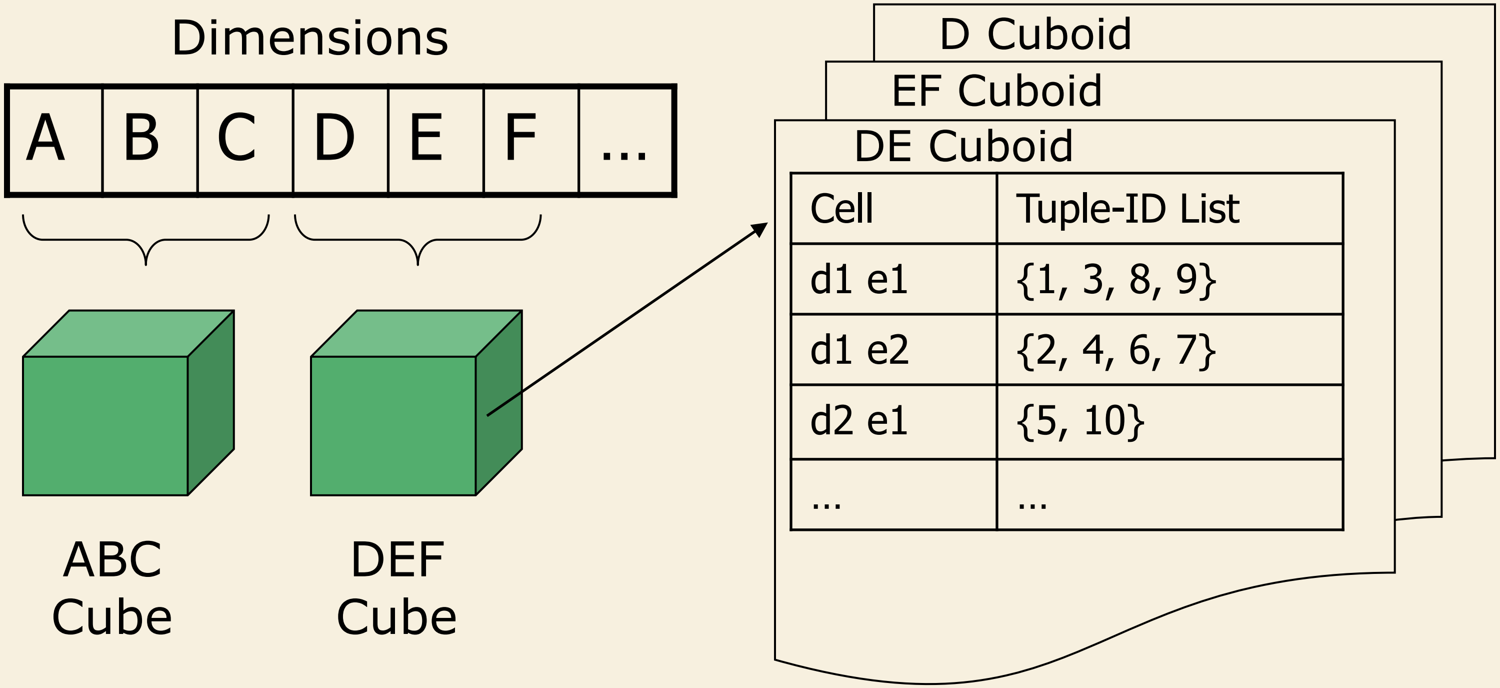
- If measures other than `count` are present, store in *ID_measure* table separate from the shell fragments

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

THE FRAG-SHELLS ALGORITHM

1. Partition set of dimension (A_1,…,A_n) into a set of *k* fragments (P_1,…,P_k).
2. Scan base table once and do the following
   a. insert <tid, measure> into ID_measure table.
   b. for each attribute value a_i of each dimension A_i
   c. build inverted index entry <a_i, tidlist>
3. For each fragment partition P_i
   a. build local fragment cube S_i by intersecting tid-lists in bottom-up fashion.

FRAG-SHELLS (2)



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

ONLINE QUERY COMPUTATION: QUERY

- A query has the general form $\langle a_1, a_2, \ldots, a_n : M \rangle$
- Each a_i has 3 possible values

1. Instantiated value
2. Aggregate * function
3. Inquire ? function

- For example, $\langle 3 \quad ? \quad ? \quad * \quad 1 : count \rangle$

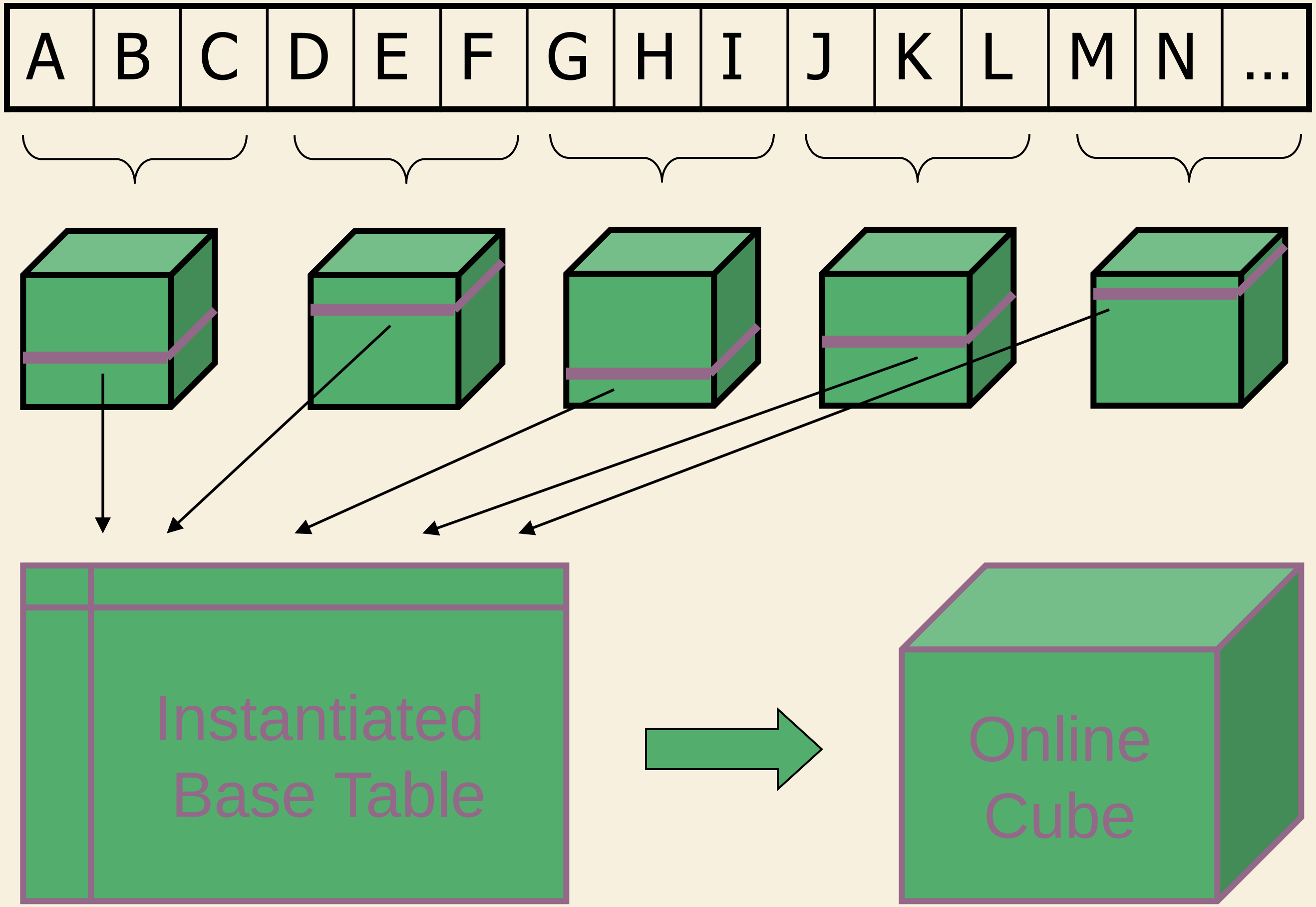
  returns a 2-D data cube.

ONLINE QUERY COMPUTATION: METHOD

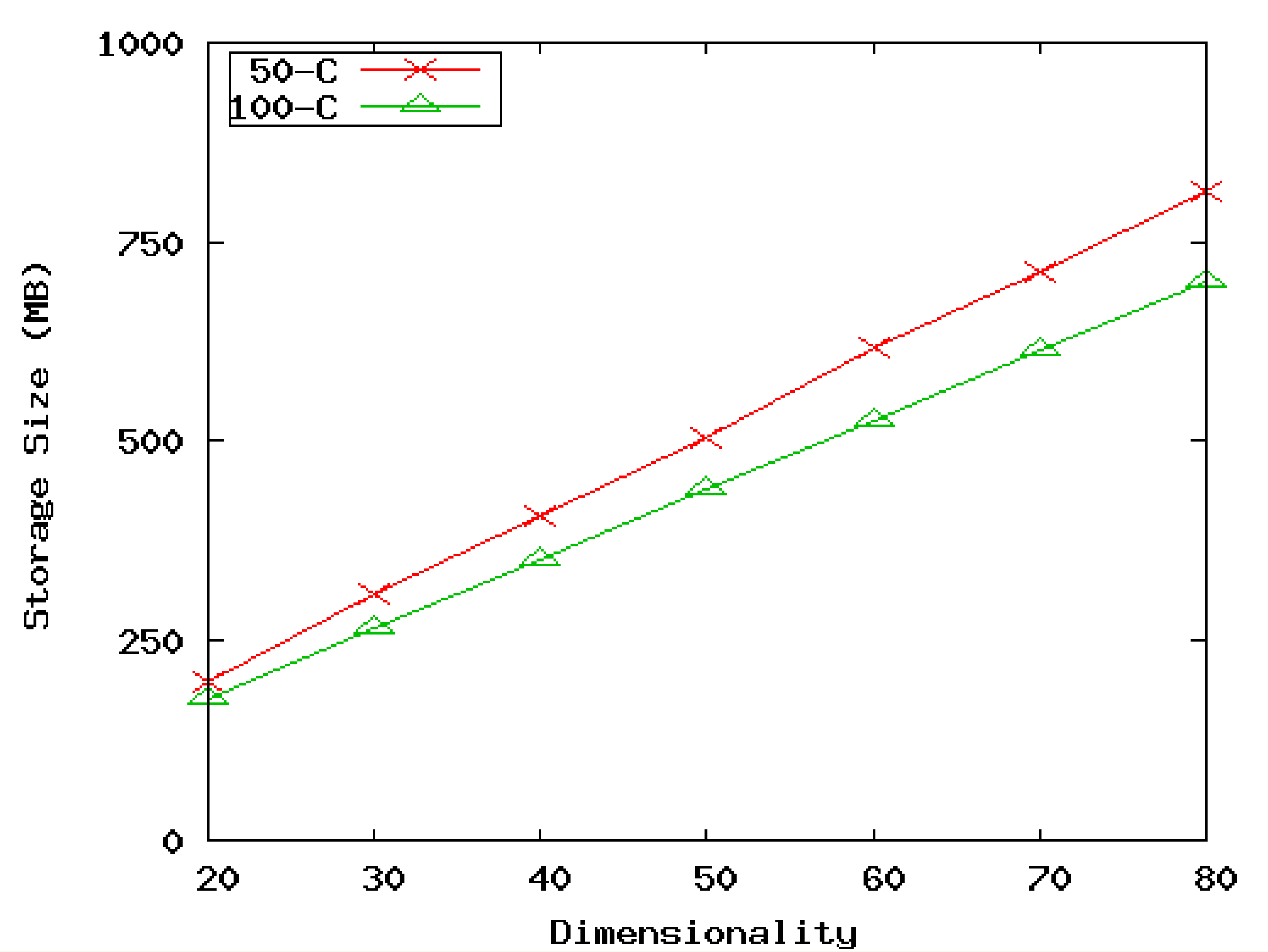
Given the fragment cubes, process a query as follows

1. Divide the query into fragment, same as the shell
2. Fetch the corresponding TID list for each fragment from the fragment cube
3. Intersect the TID lists from each fragment to construct **instantiated base table**
4. Compute the data cube using the base table with any cubing algorithm

ONLINE QUERY COMPUTATION: SKETCH



EXPERIMENT: SIZE VS. DIMENSIONALITY (50 AND 100 CARDINALITY)



- (50-C): 10^6 tuples, 0 skew, 50 cardinality, fragment size 3.
- (100-C): 10^6 tuples, 2 skew, 100 cardinality, fragment size 2.

EXPERIMENTS ON REAL WORLD DATA

- UCI Forest CoverType data set
  - 54 dimensions, 581K tuples
  - Shell fragments of size 2 took 33 seconds and 325MB to compute
  - 3-D subquery with 1 instantiate D: 85ms~1.4 sec.
- Longitudinal Study of Vocational Rehab. Data
  - 24 dimensions, 8818 tuples
  - Shell fragments of size 3 took 0.9 seconds and 60MB to compute
  - 5-D query with 0 instantiated D: 227ms~2.6 sec.

DATA CUBE TECHNOLOGY: SUMMARY

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
  - MultiWay Array Aggregation
  - BUC
  - Star-Cubing
  - High-Dimensional OLAP with Shell-Fragments

DATA CUBE TECHNOLOGY: SUMMARY

- There are other methods which are left for SPW (Student Personal Work)
  - Processing Advanced Queries by Exploring Data Cube Technology
    - Sampling Cubes
    - Ranking Cubes
  - Multidimensional Data Analysis in Cube Space
    - Discovery-Driven Exploration of Data Cubes
    - Multi-feature Cubes
    - Prediction Cubes