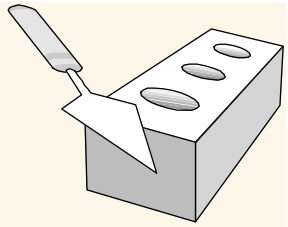
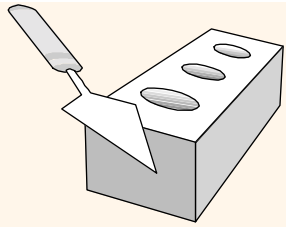


OLAP (Online Analytical Processing): Prefix-sum Data Cubes

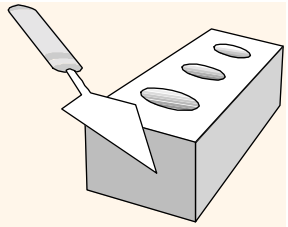
Excerpt from OLAP Presentation by Cyrus Shahabi

Content

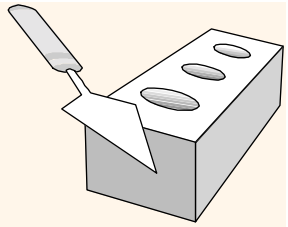


- Introduction to Multidimensional Databases (from A.R. 20 and 21)
- Focus Application: OLAP
  - Prefix-Sum Data Cube (from A.R. 16)
  - Dynamic Data Cube (from A.R. 17)
  - Iterative Data Cube (from A.R. 18)
  - Wavelet-based approaches
    - Compact Data Cube (from A.R. 19)
    - ProPolyne (from A.R. 22 and 23)

Multidimensional Databases



- During the past decade, the *multidimensional data model* emerged for use when the objective is online analysis of data rather than performing transactions.
- Multidimensional databases view data as multidimensional *cubes* that are particularly well suited for data analysis.
- Multidimensional data models have three important data analysis application areas:
  - *Data warehouses* are large repositories that integrate and abstract data from several sources in an enterprise for analysis.
  - *Online analytical processing* (OLAP) systems provide fast answers for queries that aggregate large amounts of detail data to find overall trends.
  - *Data mining* applications seek to discover knowledge by searching semi-automatically for previously unknown patterns and relationships in multidimensional databases.

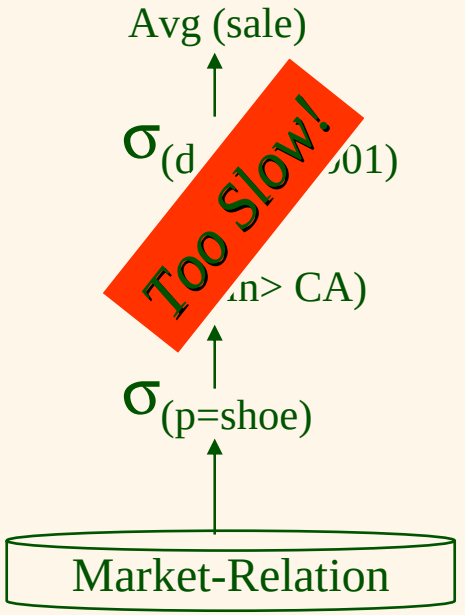


Focus Application: On-Line Analytical Processing (OLAP)

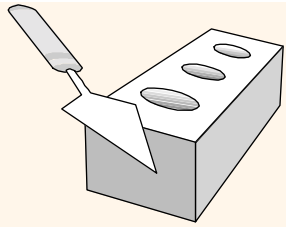
- Multidimensional data sets:
  - Dimension attributes (e.g., Store, Product, Data)
  - Measure attributes (e.g., Sale, Price)
- Range-sum queries
  - Average sale of shoes in CA in 2001
  - Number of jackets sold in Seattle in Sep. 2001
- Tougher queries:
  - Covariance of sale and price of jackets in CA in 2001 (correlation)
  - Variance of price of jackets in 2001 in Seattle

Market-Relation

Store Location	Product	Date	Sale	Price
LA	Shoes	Jan. 01	$21,500	$85.99
NY	Jacket	June 01	$28,700	$45.99
⋮	⋮	⋮	⋮	⋮

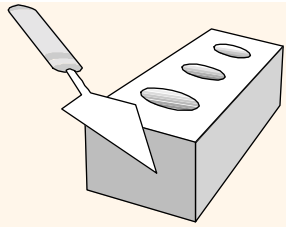


Definitions

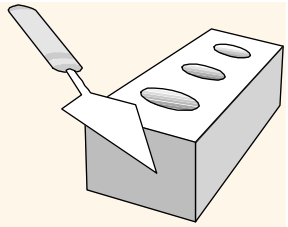


- Multidimensional databases view data as cubes that generalize spreadsheets to any number of dimensions.
- In addition, cubes support hierarchies in dimensions and formulas without duplicating their definitions.
- A collection of related cubes comprises a multidimensional database or data warehouse.
- Dimensions are used for selecting and aggregating data at the desired level of detail.
- A dimension is organized into a hierarchy composed of numerous levels, each representing a level of detail required by the desired analyses.
- Each instance of the dimension, or dimension value, belongs to a particular level.

Definitions (cont'd)



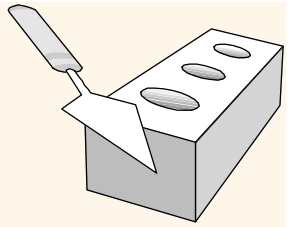
- Facts represent the subject—the interesting pattern or event in the enterprise that must be analyzed
- In most multidimensional data models, facts are implicitly characterized by their combination of dimension values
- In a multidimensional database, measures generally represent the properties of the fact that the user wants to optimize. Measures then take on different values for various dimension combinations
- A fact consists of two components:
  - a number of numerical properties, such as the sales or price
  - a formul*a,* usually a simple aggregation function such as sum, that can combine several measure values into one representative fact
- The property and formula are chosen to provide a meaningful value for all combinations of aggregation levels



Range Queries in OLAP Data Cubes

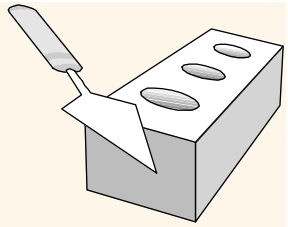
Ching-Tien Ho, Rakesh Agrawal, Nimord Megiddo, Rammakrishnan Srikant
IBM Almaden Research Center

Outline



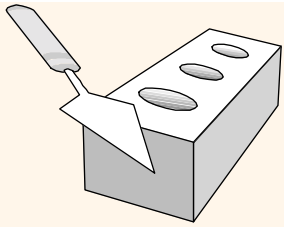
- Introduction
- The Basic Range-Sum Algorithm
- The Blocked Range-Sum Algorithm
- The Batch-Update Algorithm for Range-Sum Queries
- Choosing Dimension, Cuboids and Block Sizes
- Conclusion

Introduction



- Example for MDDB
  - Assume the data cube has four functional attributes (dimensions): age, year, state and insurance type.
  - The data cube will have age*year*state*type cells, with each cell containing the total revenue (the measure attribute) for the corresponding combination of these four attributes

Introduction (cont'd)



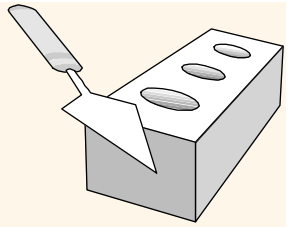
Dimensional Attribute ← Customer_age

Measure Attribute ↓

Date ↑

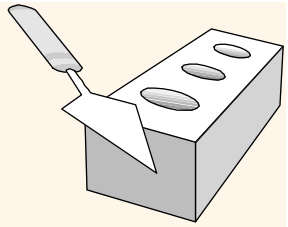
Index	*17*	*18*	*19*	*20*	*21*	*22*	*23*
10/24	$300	$400	$700	$500	$250	$600	$150
10/25	$120	$130	$100	$240	$300	$100	$300
10/26	$110	$170	$200	$600	$500	$500	$300
10/27	$800	$550	$790	$800	$470	$500	$250
10/28	$250	$230	$600	$600	$250	$230	$170
10/29	$170	$390	$440	$500	$250	$360	$190
10/30	$370	$500	$800	$950	$560	$590	$390

Introduction (cont'd)



- Range Queries
  - That apply a given aggregation operation over selected cells where the selection is specified as continuous ranges in the domains of some of the attributes.
  - Example: Sum, Max, Count and Average

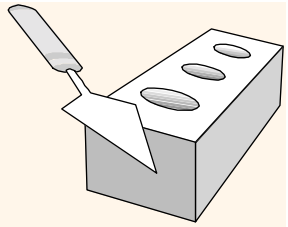
The Basic Range-Sum Algorithm



- Pre-computed *Prefix-Sum* array P
  - Let P be a d-dimensional array of size $N = n_1 * n_2 * \ldots * n_d$
  - Then we define P as:

$$P[x, y] = Sum(0 : x, 0 : y) = \sum_{i=0}^{x} \sum_{j=0}^{y} A[i, j]$$

Construction Example

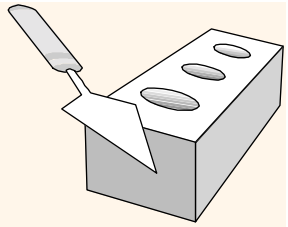


Array *A* (original array)						
Index	0	1	2	3	4	5
0	3	5	1	2	2	3
1	7	3	2	6	8	2
2	2	4	2	3	3	5

Array *P* (Prefix-Sum array)						
Index	0	1	2	3	4	5
0	3	8	9	11	13	16
1	10	18	21	29	39	44
2	12	24	29	40	53	63

P(1,1)=A(0,0)+A(0,1)+A(1,0)+ A(1,1)

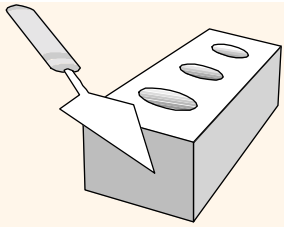
Querying



- Theorem 1: Query Complexity

  Proves how any range-sum of A can be computed from up to 2^d appropriate elements of P.

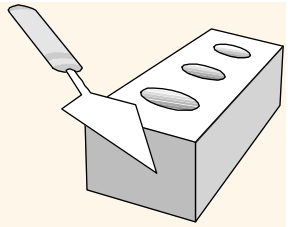
Querying Example



Question:How can we get the sum of this area?

Array *A* (original array)						
Index	0	1	2	3	4	5
0	3	5	1	2	2	3
1	7	3	2	6	8	2
2	2	4	2	3	3	5

Querying Example (cont'd)

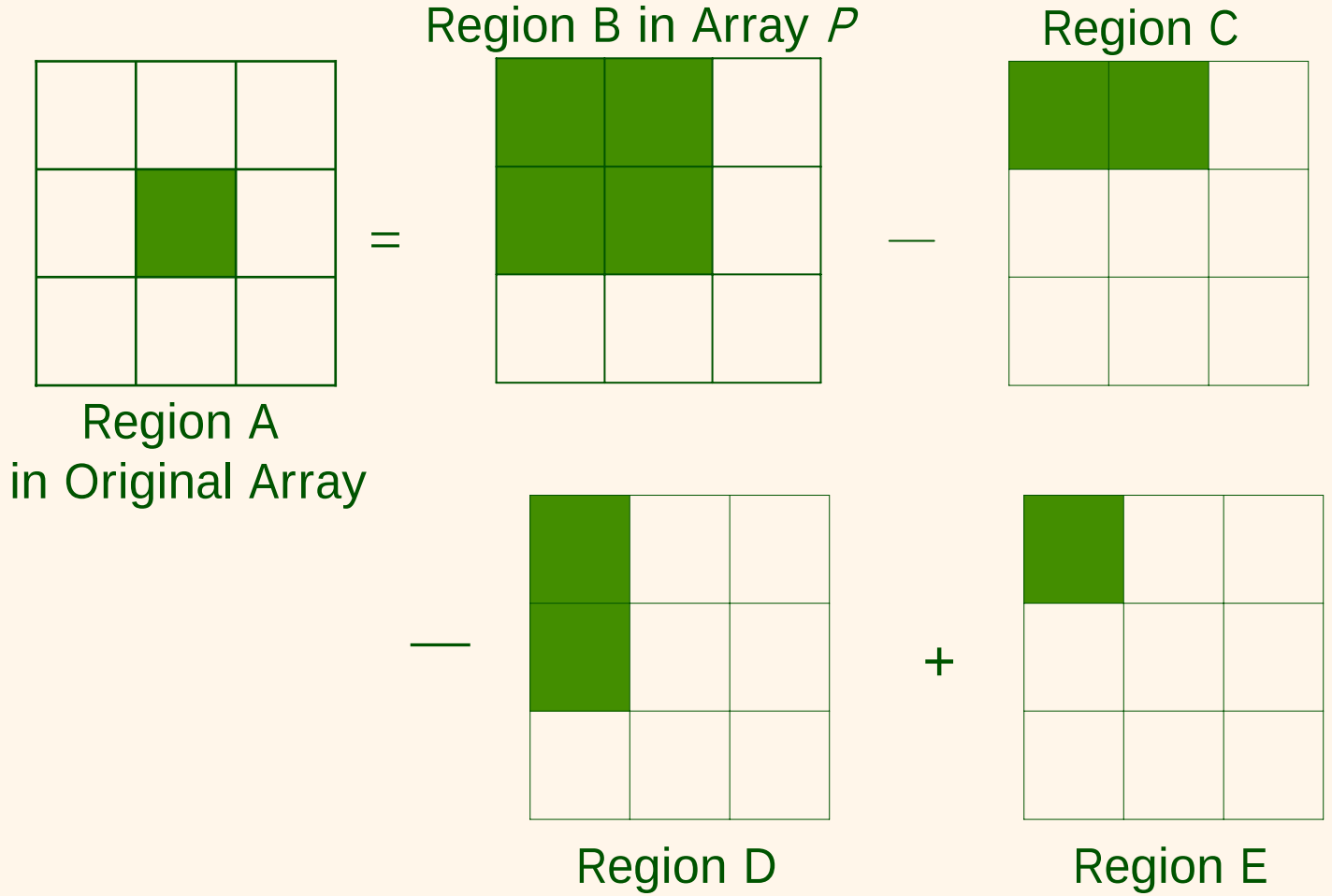


- When d=2, the range-sum Sum(l_1:h_1,l_2:h_2) can be obtained in three computation steps as

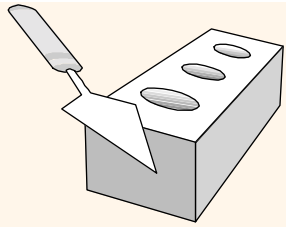
$$\text{Sum}(l_1{:}h_1,l_2{:}h_2) = P[h_1,h_2] - P[h_1,l_2-1] - P[l_1-1,h_2] + P[l_1-1,l_2-1]$$

$2^{2-}1=3$ steps

Querying Example (cont'd)



Querying Example (cont'd)



The range-sum Sum(2:3,1:2) = P [3,2]-P [3,0]-P [1,2]+P [1,0]
= 40-11-24+8=13

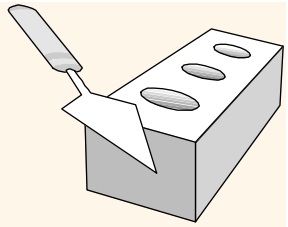
Array A (original array)

Index	0	1	2	3	4	5
0	3	5	1	2	2	3
1	7	3	2	6	8	2
2	2	4	2	3	3	5

Array P

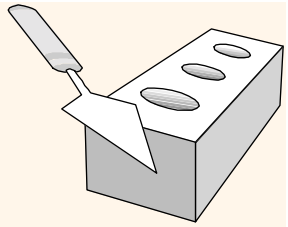
Index	0	1	2	3	4	5
0	3	8	9	11	13	16
1	10	18	21	29	39	44
2	12	24	29	40	53	63

The Blocked Range-Sum Algorithm



- To save space as a trade-off to time is to keep Prefix-Sums at a coarser-grained (block) level.
- Store the Prefix-Sum only when every index is either one less than some multiple of *b*, or is the last index.
- If the blocked algorithm is used, the original array A cannot be dropped

Construction Example

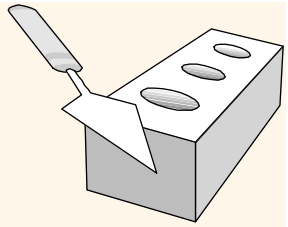


- For Example, in the two-dimensional case, only *P* [b-1,b-1], *P* [b-1,2b-1], … *P* [b-1,nb-1], *P* [2b-1,b-1], *P* [2b-1,2b-1], …

Example of the blocked Prefix-Sum array *P* with *b*=2

Array *P*						
Index	0	1	2	3	4	5
0	-	-	-	-	-	-
1	-	18	-	29	-	44
2	-	24	-	40	-	63

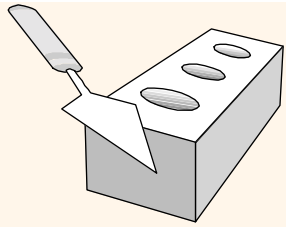
Querying



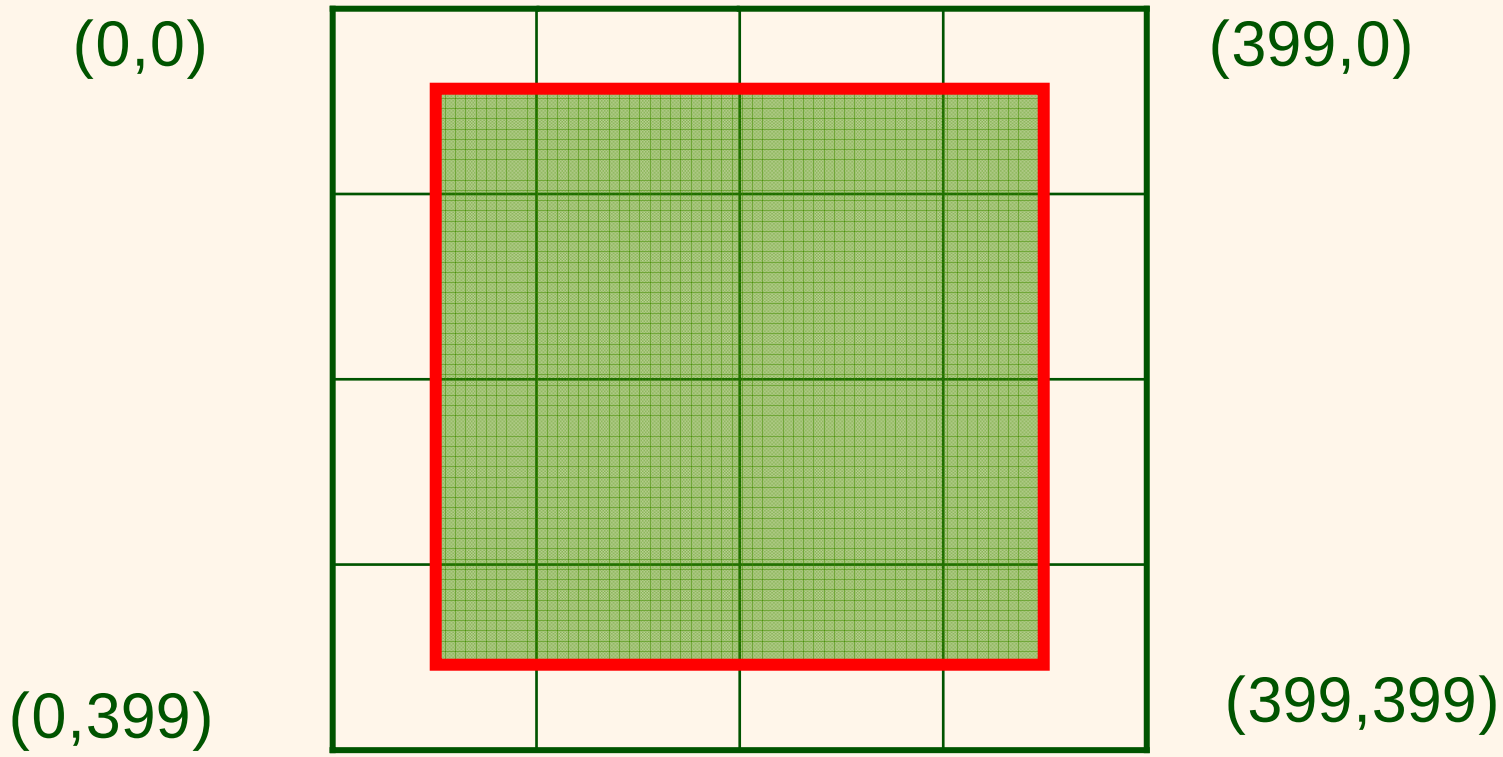
- How to compute Sum(l_1:h_1,l_2:h_2)?

  Intuitively, decompose the range in each dimension into $\mathbf{3}^{d}$ disjoining sub-ranges where the middles sub-range is properly aligned with the block structure.

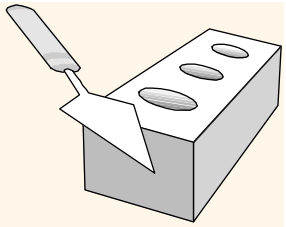
Querying (cont'd)



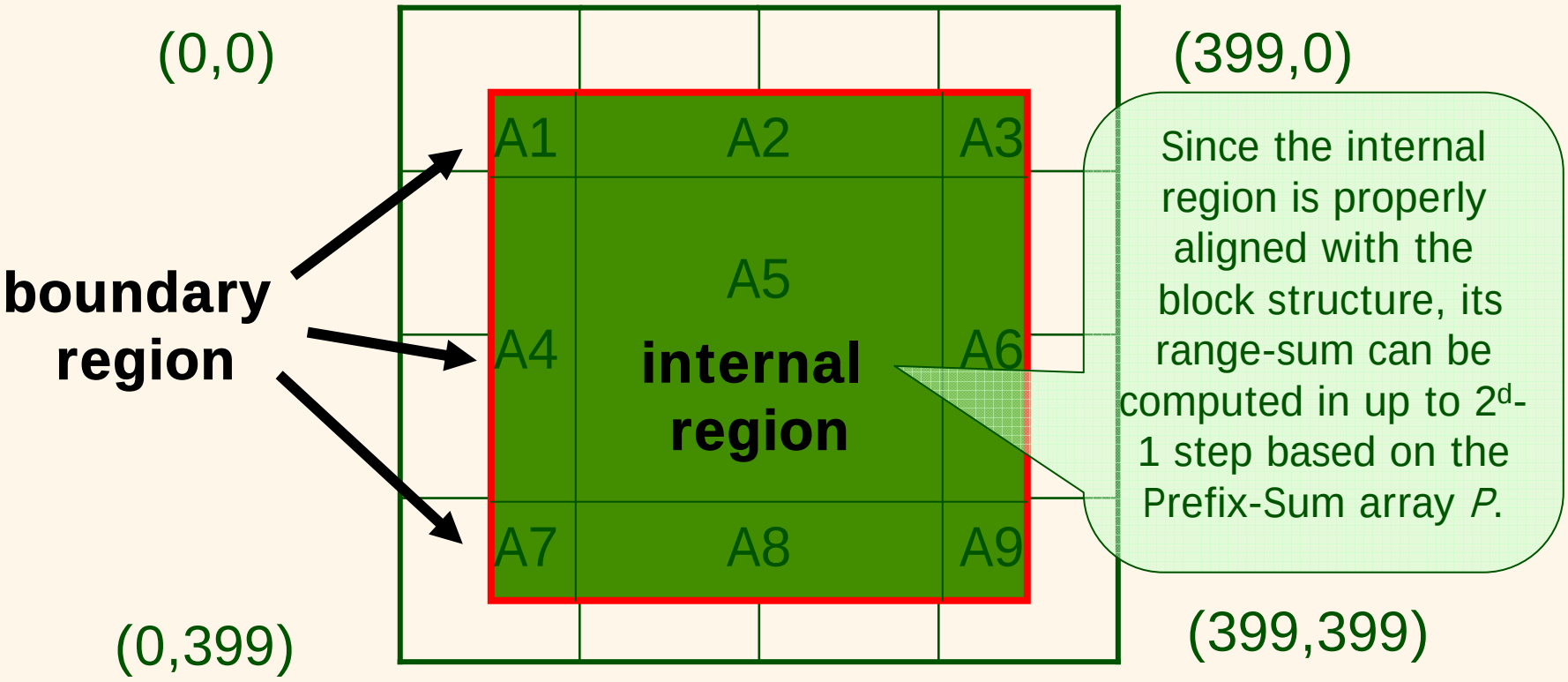
- Query: Sum (50:350,50:350)



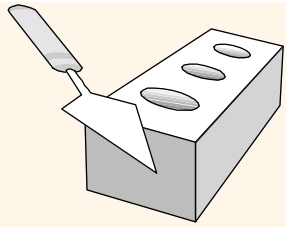
Querying (cont'd)



❖ This diagram shows the $3^2=9$ decomposed regions for this query. A5 is an **internal region** and all other 8 regions are **boundary regions**.

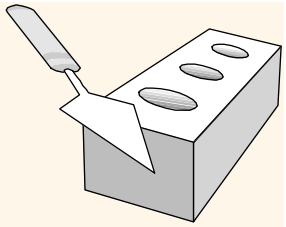


Querying (cont'd)



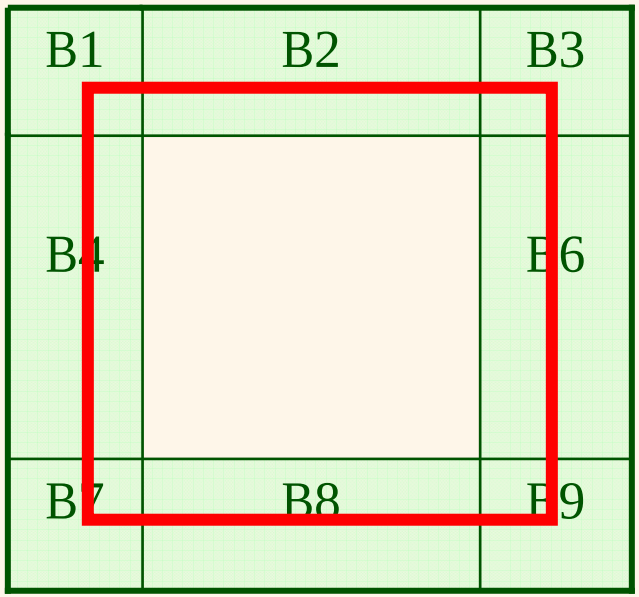
- For any boundary region, there are two possible methods for computation:
  1. Sum up all elements of A corresponding to the boundary regions.
  2. One can sum up all elements of A corresponding to the complement region, then subtract the sum from the range-sum for the super-block region.The range-sum for a super-block region can be computed in 2^d-1 step using Prefix-Sum array P.

Querying (cont'd)

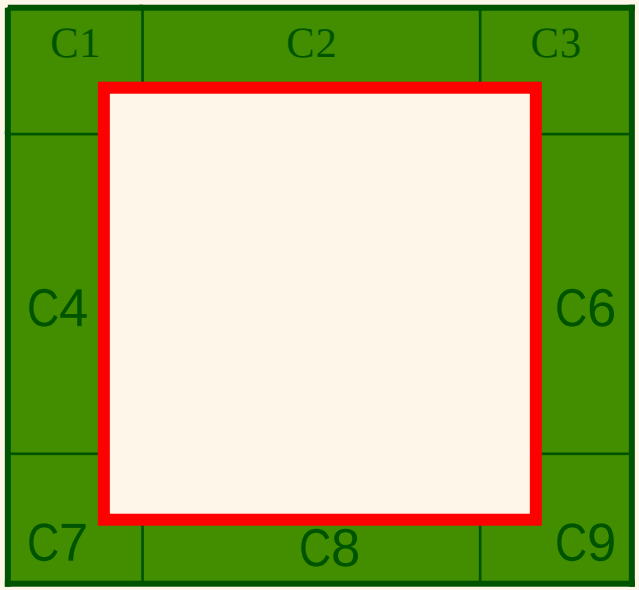


The boundary region = super-block region
– complement region

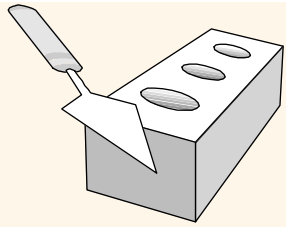
super-block region



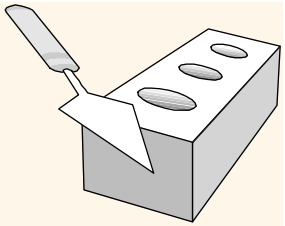
complement region



Querying (cont'd)

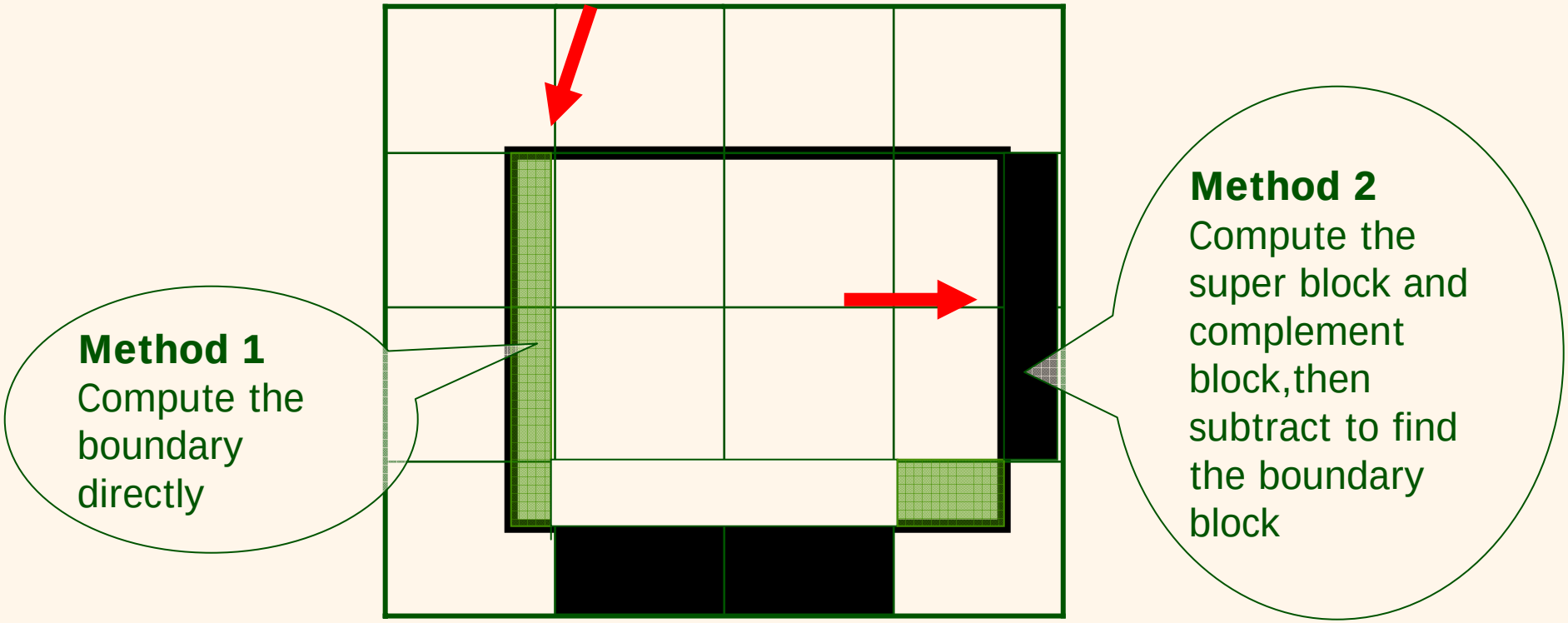


- To minimize the time complexity, the algorithm will choose the first method when the volume of the target region is smaller than or equal to "the volume of its complement region plus 2d-1"; and will choose the second method, otherwise.
- The choice is made per boundary region independently.

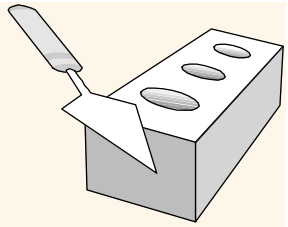


Example Query

- Consider the query Sum(75:374,100:357)

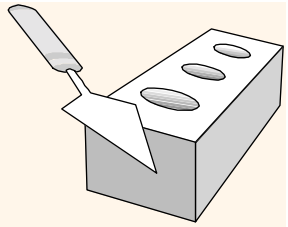


Batch-Update



- Assuming block size=1, when a query updates an array element $A[x_1, \ldots, x_d]$, all $P[y_1, \ldots, y_d]$ where $y_j \geq x_j$, will be affected.
- Batch-Update
  - In a typical OLAP environment, updates to the data cube are accumulated over a period of time, then these updates are performed together as a batch at the end of each period.
- Assume a model that *k* update queries are issued successively before the next read-only query is issued
- Idea: batch updates together and perform a "combined" update to array *P*.

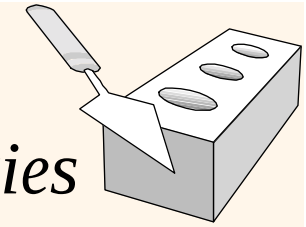
Batch-Update (cont'd)



Prefix-Sum array P of size N^2

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	3	8	9	11	13	17	23	26
1	10	18	21	29	93	50	57	62
2	12	***24**	29	40	53	67	78	88
3	15	29	35	51	67	86	99	117
4	19	35	15	61	80	103	123	142
5	21	40	50	75	95	126	151	172
6	25	49	61	93	115	154	182	206
7	27	55	69	103	127	168	205	230

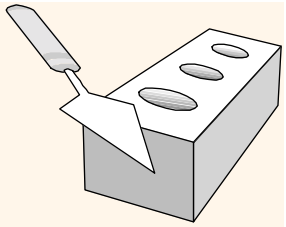
The Batch-Update Algorithm for Range-Sum Queries



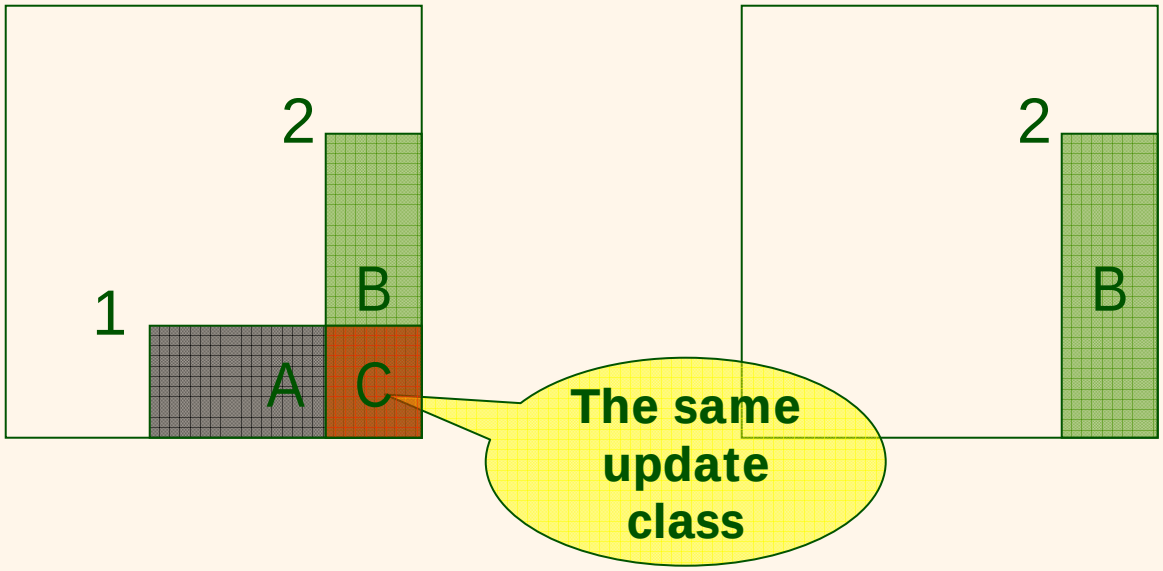
- For each update in the form of (location of the element of A, value-to-add), update the corresponding A element right away
  - For d=1, updates in the form (u_i, v_i)

- Queue an update form to be used later for a combined update of *P*

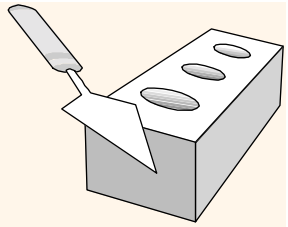
Update Classes



- Two elements are in the same ***update-class*** if they are affected by the same subset of k updates.

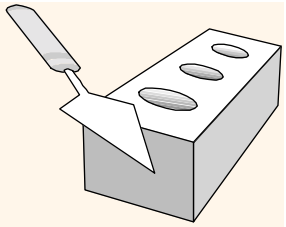


Update Classes (cont'd)

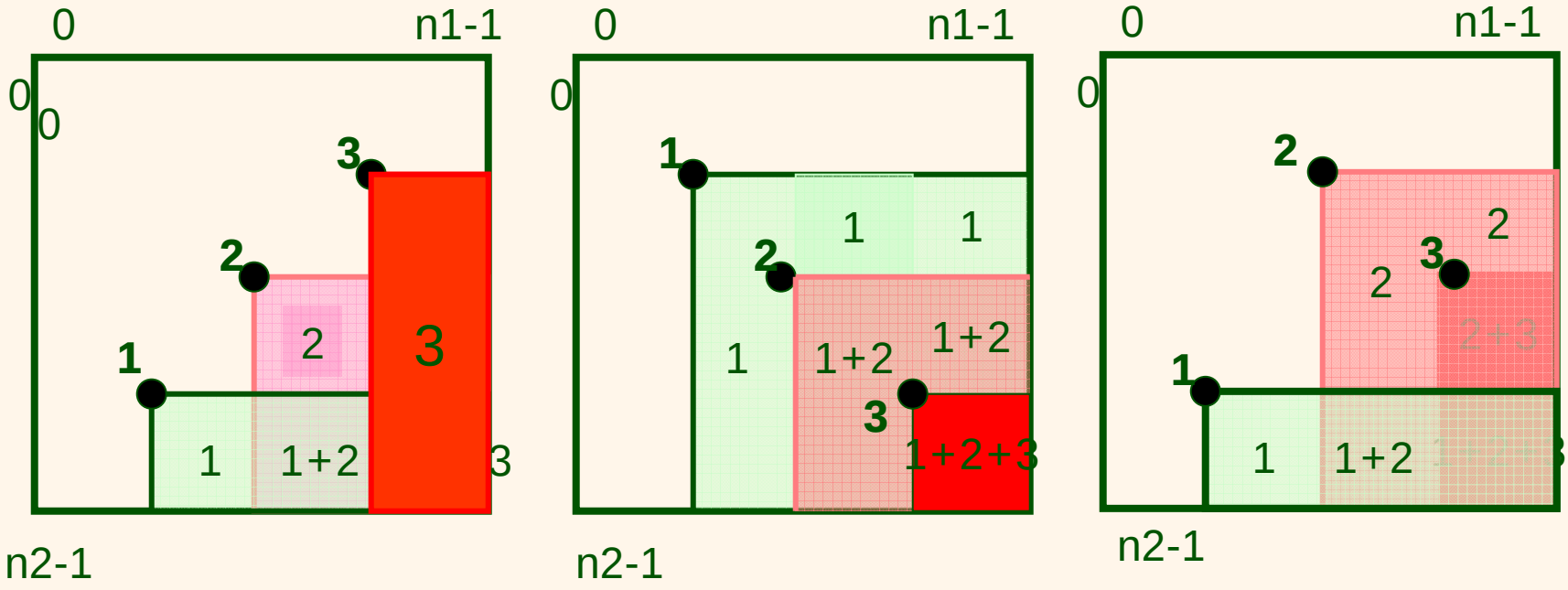


- The goal is to group all "affected" elements of array *P* into minimum number of disjoint regions
  - **Property1**: All elements of *P* in the same region are in the same update-class.
  - **Property2**: Each region has a shape of a *d*-dimensional rectangular cube.

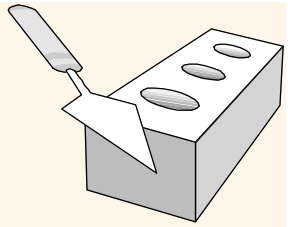
Update Classes (cont'd)



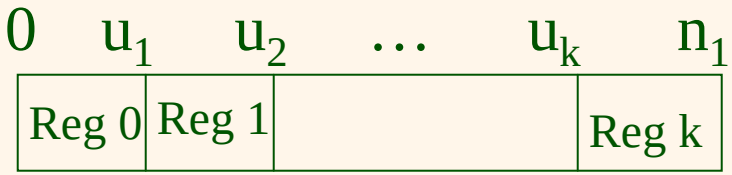
- Example of partitioned region for k=3 and d=2



Batch-Update Using Update Classes

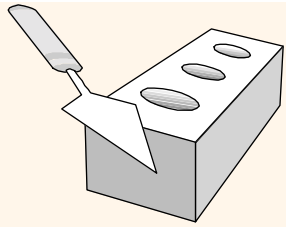


- Assume d=1
- Sort the indices of k updates in the ascending order: u_1, u_2, …, u_k
- Partition the index space of P into k+1 adjoining region according to the sorted k indices:



- All elements of P in the same region are in the same update-class.
- Except region 0, all other regions are affected.
- Perform combined updates one region at a time starting from region 1.

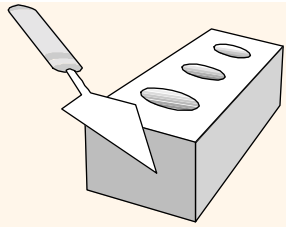
Batch-Update Using Update Classes (cont'd)



- For d > 1, recursively call a number of batch-update algorithms for dimensions d-1

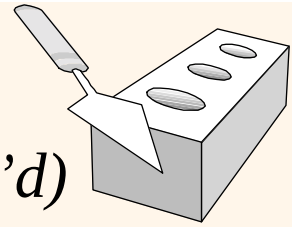
- Theorem: The batch-update algorithm will group all affected elements of P into up to $\prod_{j=0}^{d-1} \frac{(n+j)}{d!}$ regions with the two properties described before and perform the k batch-updates correctly.

Batch-Update with Blocked Prefix-Sum



- The batch-update algorithm for b>1
  1. For every *d*-dimensional block of from b*b*…*b, sum up all values-of-add for all updates of A in the block
     - Thus, the index space of A has been contracted by a factor of b in every dimension
  2. Apply the batch-update algorithm
     - Treating each block of A as one element of A
     - Treating the combined values-to-add for each block as a value-to-add for each element of A
     - Treating the blocked Prefix-Sum array *P* as the basic Prefix-Sum array *P*

Batch-Update with Blocked Prefix-Sum (cont'd)

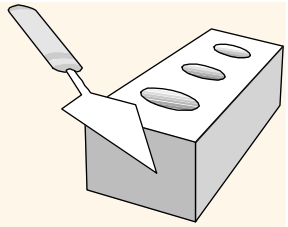


Array *A*						
Index	0	1	2	3	4	5
0	3	5	1	2	2	3
1	7	3	2	6	8	2
2	2	4	2	3	3	5

Array *P*						
Index	0	1	2	3	4	5
0	-	-	-	-	-	-
1	-	18	-	29	-	44
2	-	24	-	40	-	63

Array *P*						
Index	1	3	5	7	9	11
1	18	29	44	-	-	-
2	24	40	63	-	-	-

Conclusion



- For speeding up range sum queries: pre-compute multidimensional Prefix-Sums of the data cube. Then, any range-sum query can be answered by accessing 2^d appropriate Prefix-Sums.