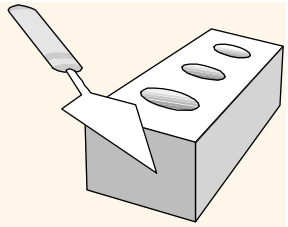
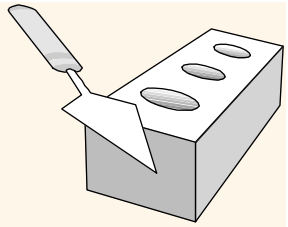


OLAP (Online Analytical Processing): Iterative Data Cubes

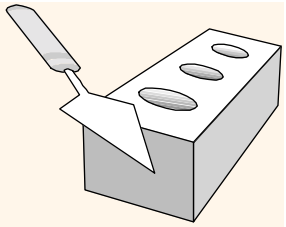
Excerpt from Presentation by M. Riedewald

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)



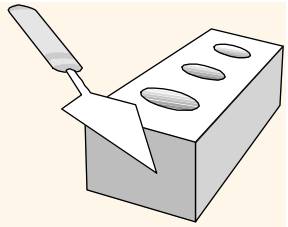
Flexible Data Cubes for Online Aggregation

Mirek Riedewald, Divyakant Agrawal, and Arm El Abbadi

Space-Efficient Data Cubes for Dynamic Environments

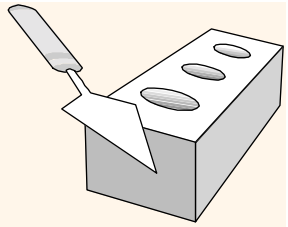
Mirek Riedewald, Divyakant Agrawal, Arm El Abbadi, and Renato Pajarola

Outline



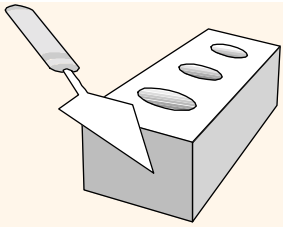
- What is IDC?
- Pre-Aggregation Techniques
- Querying and Updating an IDC
- Selecting an IDC
- IDC with more than One Dimension
- Conclusion

Iterative Data Cube (IDC)

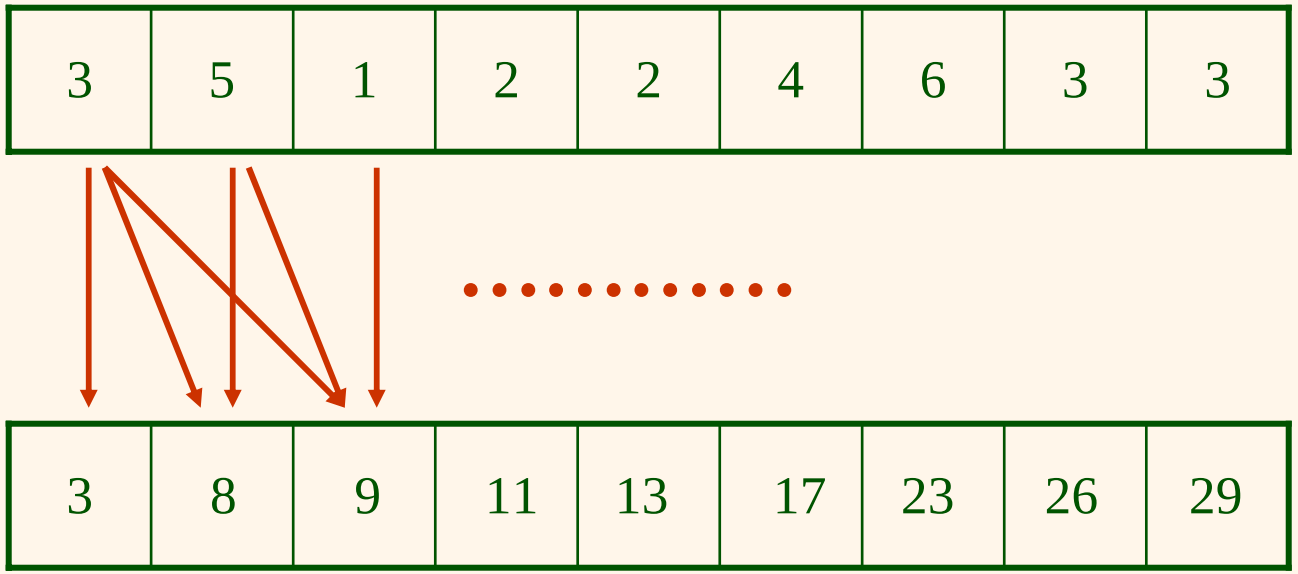


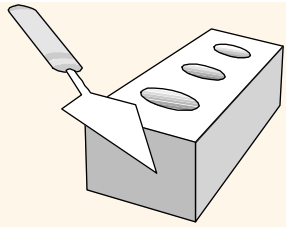
- Provides a modular framework for combining **one-dimensional** aggregation techniques to create space-optimal **high-dimensional** data cubes.
  - For each dimension a different one-dimensional technique can be selected.
  - Combining the one-dimensional techniques is easy.
- Allows a variety of cost tradeoffs between query and update.
- Generalizes some of the pre-aggregation approaches:
  - PS (Prefix Sum)
  - SRPS (Space-Efficient Relative Prefix Sum)
  - SDDC (Space-Efficient Dynamic Data Cube)

Prefix Sum (review)



Original array





Space-Efficient Relative Prefix Sum

- The data cube is partitioned into a set of disjoint hyper-rectangles of equal size (termed *boxes*)

- Any cell in box B stores the value:

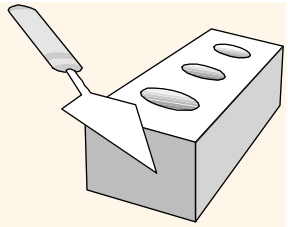
$$SUM(A[l_1, l_2, \ldots, l_d]:A[c])$$

where for all i: $1 <= i <= d$

$l_i = 0$, if $c_i = a_i$

$l_i = a_i + 1$, if $a_i + 1 <= c_i < a_i + k$

Space-Efficient Relative Prefix Sum (cont'd)



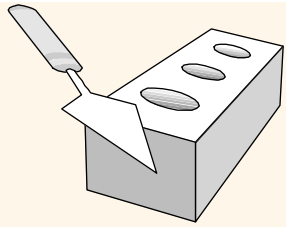
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

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

Space-Efficient Relative Prefix Sum (cont'd)



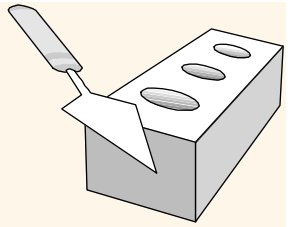
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6						
1			5						
2	9	7							
3									
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



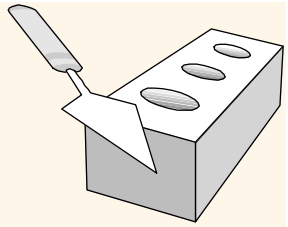
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6						
1			5						
2	9	7	11						
3									
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



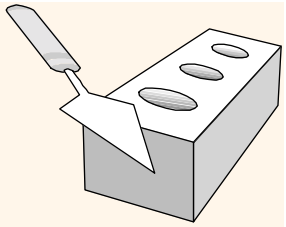
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6	11					
1			5	18					
2	9	7	11						
3									
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



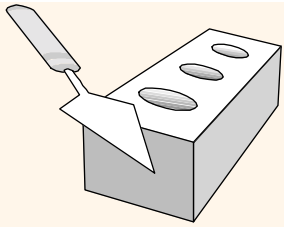
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6	11					
1			5	18					
2	9	7	11						
3	15	14							
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



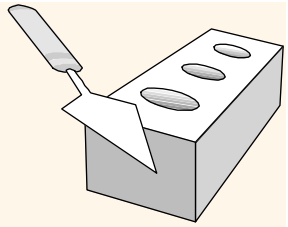
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6	11					
1			5	18					
2	9	7	11	29					
3	15	14							
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



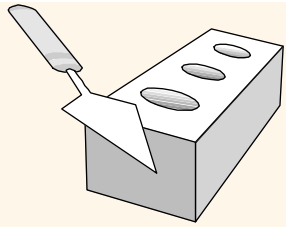
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6	11					
1			5	18					
2	9	7	11	29					
3	15	14	20						
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



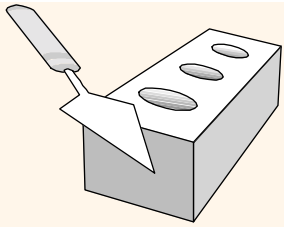
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6	11					
1			5	18					
2	9	7	11	29					
3	15	14	20	51					
4									
5									
6									
7									
8									

Space-Efficient Relative Prefix Sum (cont'd)



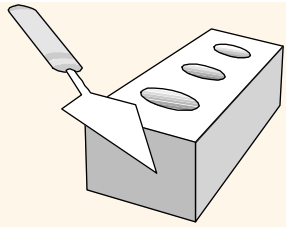
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0			6	11		6	23		6
1			5	18		15	34		6
2	9	7	11	29	11	21	55	7	18
3	15	14	20	51	16	35	99	18	34
4			3	10		7	24		4
5	6	5	9	24	4	16	52	2	6
6	25	24	36	93	21	61	182	23	47
7			6	10		4	23		4
8	7	8	13	23	6	9	42	10	19

Space-Efficient Relative Prefix Sum (cont'd)



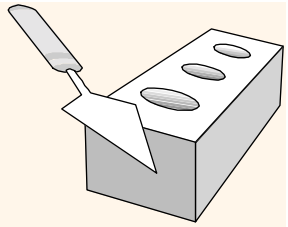
Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

	0	1	2	3	4	5	6	7	8
0	3	5	6	11	2	6	23	3	6
1	7	3	5	18	8	15	34	2	6
2	9	7	11	29	11	21	55	7	18
3	15	14	20	51	16	35	99	18	34
4	4	2	3	10	3	7	24	1	4
5	6	5	9	24	4	16	52	2	6
6	25	24	36	93	21	61	182	23	47
7	2	4	6	10	3	4	23	1	4
8	7	8	13	23	6	9	42	10	19

Space-Efficient Relative Prefix Sum (cont'd)



Original array

	0	1	2	3	4	5	6	7	8
0	3	5	1	2					
1	7	3	2	6					
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4									
5									
6									
7									
8	5	4	3	1	3	2	1	9	6

SRPS array (block size 3)

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

Border cells

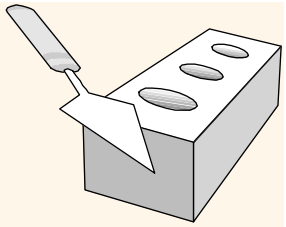
Inner cells

Border cells include cells from outside the block into their aggregation.

Inner cells store sums local to the block.

Partition into blocks of equal size

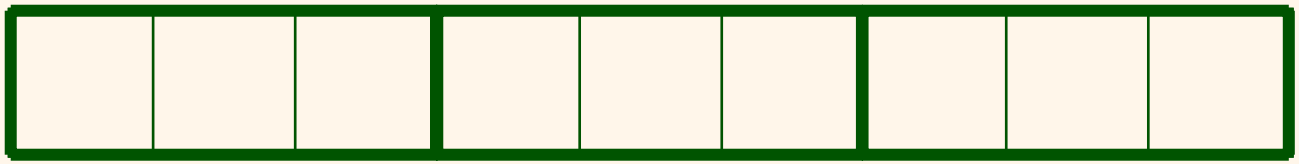
Space-Efficient Relative Prefix Sum (cont'd)



Original array

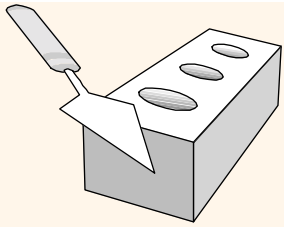
3	5	1	2	2	4	6	3	3

SRPS array

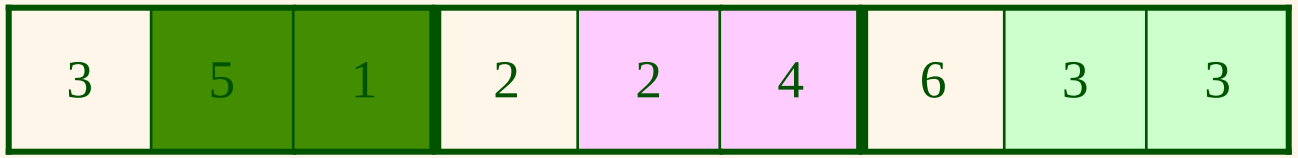


(block size 3)

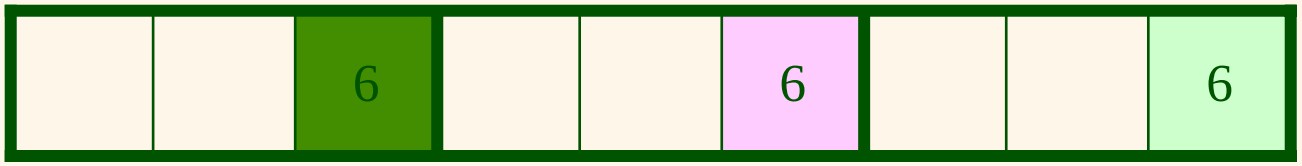
Space-Efficient Relative Prefix Sum (cont'd)



Original array

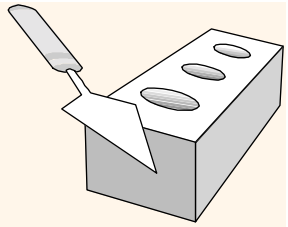


SRPS array



(block size 3)

Space-Efficient Relative Prefix Sum (cont'd)



Original array

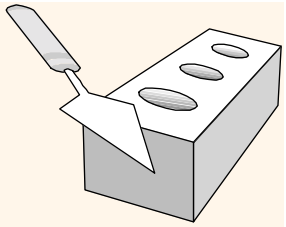
3	5	1	2	2	4	6	3	3

SRPS array

		6	11		6			6

(block size 3)

Space-Efficient Relative Prefix Sum (cont'd)



Original array

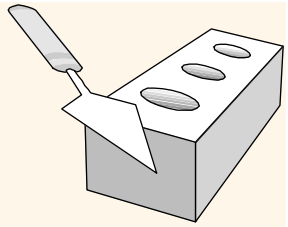
3	5	1	2	2	4	6	3	3

SRPS array

		6	11		6	23		6

(block size 3)

Space-Efficient Relative Prefix Sum (cont'd)



Original array

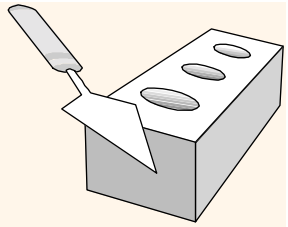
3	5	1	2	2	4	6	3	3

SRPS array

3	5	6	11	2	6	23	3	6

(block size 3)

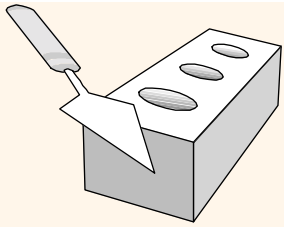
Space-Efficient Dynamic Data Cube



- Partition to boxes the same as SRPS except for the following:

1. The side-length of a box is set to $k=n/2$ (i.e., the cube is partitioned into 2^d equi-size boxes)

2. Aggregation of the border cells remain the same as SRPS, but not for inner cells:
   - The region that contains the inner cells is recursively partitioned into 2^d boxes until there is no inner cells!



Space-Efficient Dynamic Data Cube

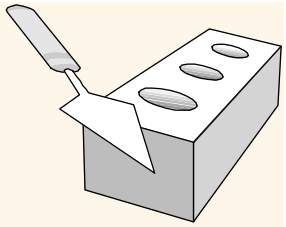
Original array

	0	1	2	3	4	5	6	7	8	9
0	3	5	1	2	2	4	6	3	3	1
1	7	3	2	6	8	7	1	2	4	2
2	2	4	2	3	3	3	4	5	7	4
3	3	2	1	5	3	5	2	8	2	1
4	4	2	1	3	3	4	7	1	3	2
5	2	3	3	6	1	8	5	1	1	2
6	4	5	2	7	1	9	3	3	4	1
7	2	4	2	2	3	1	9	1	3	3
8	5	4	3	1	3	2	1	9	6	5
9	6	1	2	4	2	1	3	1	5	2

SDDC array

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Partition into 2^d blocks of equal size d:Dimensionality



Space-Efficient Dynamic Data Cube

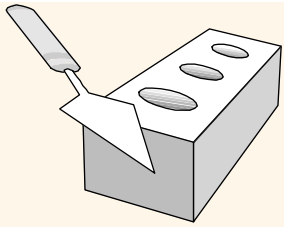
Original array

	0	1	2	3	4	5	6	7	8	9
0	3	5	1	2	2	4	6	3	3	1
1	7	3	2	6	8	7	1	2	4	2
2	2	4	2	3	3	3	4	5	7	4
3	3	2	1	5	3	5	2	8	2	1
4	4	2	1	3	3	4	7	1	3	2
5	2	3	3	6	1	8	5	1	1	2
6	4	5	2	7	1	9	3	3	4	1
7	2	4	2	2	3	1	9	1	3	3
8	5	4	3	1	3	2	1	9	6	5
9	6	1	2	4	2	1	3	1	5	2

SDDC array

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Partition into 2^d blocks of equal size d:Dimensionality



Space-Efficient Dynamic Data Cube

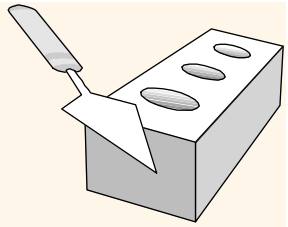
Original array

	0	1	2	3	4	5	6	7	8	9
0	3	5	1	2	2	4	6	3	3	1
1	7	3	3	4	4	4	5	2	4	3
2	2	4	2	3	3	3	4	5	7	4
3	3	2	1	5	3	5	2	8	2	1
4	4	2	1	3	3	4	7	1	3	2
5	2	3	3	6	1	8	5	1	1	2
6	4	5	2	7	1	9	3	3	4	1
7	2	4	2	2	3	1	9	1	3	3
8	5	4	3	1	3	2	1	9	6	5
9	6	1	2	4	2	1	3	1	5	2

Processing for **column** using SRPS

SDDC array

	0	1	2	3	4	5	6	7	8	9
0	3	5	6	8	10	17	6	9	12	13
1	7					33				
2	9					50				
3	12					69				
4	16					86				
5	18	19	29	54	74	126	25	45	65	77
6	4					28				
7	6					42				
8	11					60				
9	17					76				



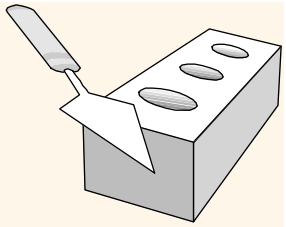
Space-Efficient Dynamic Data Cube

Original array

	0	1	2	3	4	5	6	7	8	9
0	3	5	1	2	2	4	6	3	3	1
1	7	3	2	6	8	7	1	2	4	2
2	2	4	2	3	3	3	4	5	7	4
3	3	2	1	5	3	5	2	8	2	1
4	4	2	1	3	3	4	7	1	3	2
5	2	3	3	6	1	8	5	1	1	2
6	4	5	2	7	1	9	3	3	4	1
7	2	4	2	2	3	1	9	1	3	3
8	5	4	3	1	3	2	1	9	6	5
9	6	1	2	4	2	1	3	1	5	2

SDDC array

	0	1	2	3	4	5	6	7	8	9
0	3	5	6	8	10	17	6	9	12	13
1	7	3	2	11	8	33	1	2	7	2
2	9	4		9		50	4		16	
3	12	9	5	28	14	69	7	15	35	7
4	16	2		6		86	7		11	
5	18	19	29	54	74	126	25	45	65	77
6	4	5	2	14	1	28	3	3	10	1
7	6	4		8		42	9		13	
8	11	13	7	30	7	60	13	9	39	9
9	17	1		7		76	3		9	



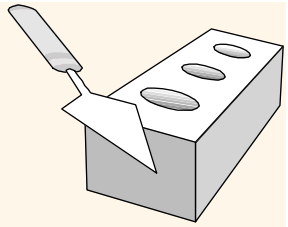
Space-Efficient Dynamic Data Cube

Original array

	0	1	2	3	4	5	6	7	8	9
0	3	5	1	2	2	4	6	3	3	1
1	7	3	2	6	8	7	1	2	4	2
2	2	4	2	3	3	3	4	5	7	4
3	3	2	1	5	3	5	2	8	2	1
4	4	2	1	3	3	4	7	1	3	2
5	2	3	3	6	1	8	5	1	1	2
6	4	5	2	7	1	9	3	3	4	1
7	2	4	2	2	3	1	9	1	3	3
8	5	4	3	1	3	2	1	9	6	5
9	6	1	2	4	2	1	3	1	5	2

SDDC array

	0	1	2	3	4	5	6	7	8	9
0	3	5	6	8	10	17	6	9	12	13
1	7	3	2	11	8	33	1	2	7	2
2	9	4	2	9	3	50	4	5	16	4
3	12	9	5	28	14	69	7	15	35	7
4	16	2	1	6	3	86	7	1	11	2
5	18	19	29	54	74	126	25	45	65	77
6	4	5	2	14	1	28	3	3	10	1
7	6	4	2	8	3	42	9	1	13	3
8	11	13	7	30	7	60	13	9	39	9
9	17	1	2	7	2	76	3	1	9	2

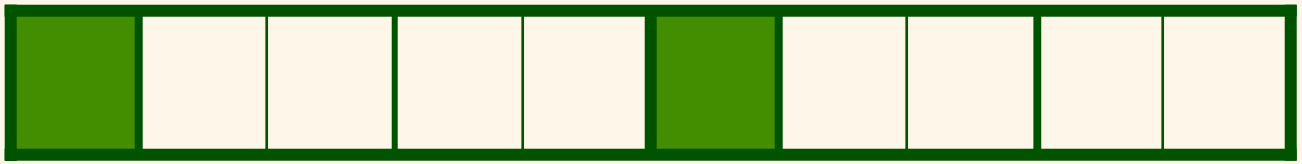


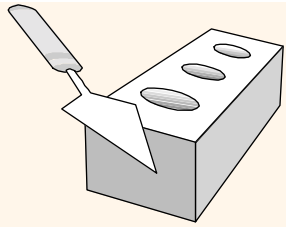
Space-Efficient Dynamic Data Cube

Original array

3	5	1	2	2	4	6	3	3	1

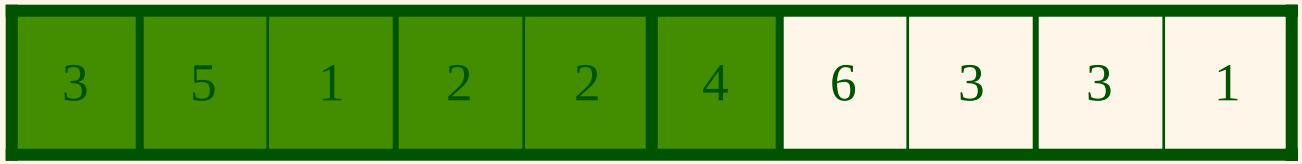
SDDC array



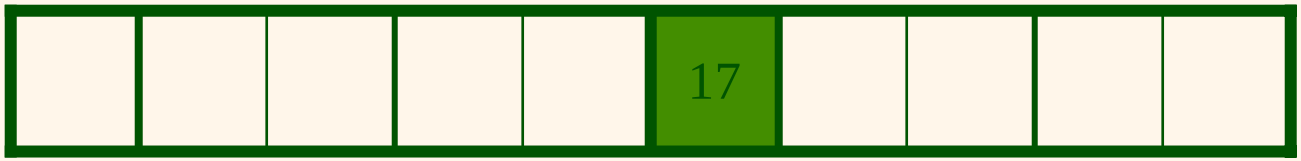


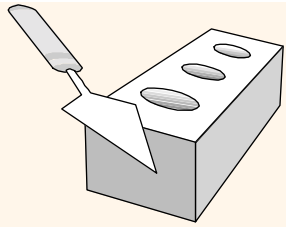
Space-Efficient Dynamic Data Cube

Original array



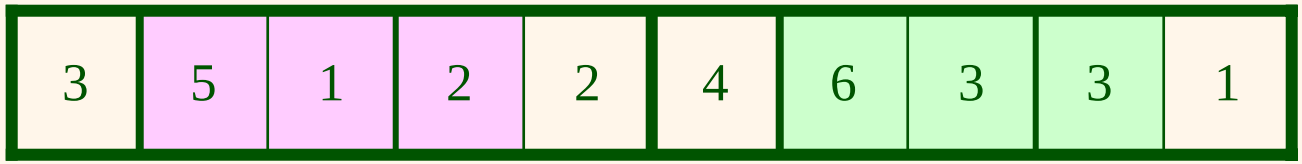
SDDC array



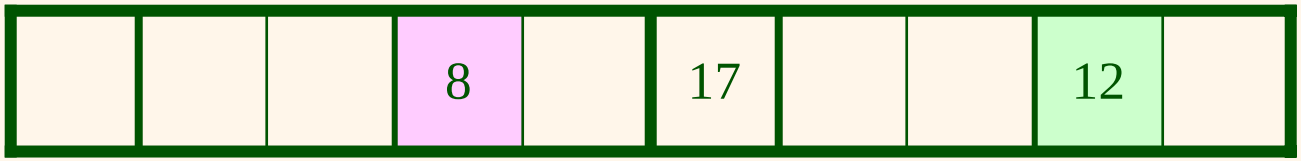


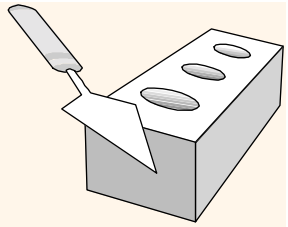
Space-Efficient Dynamic Data Cube

Original array



SDDC array





Space-Efficient Dynamic Data Cube

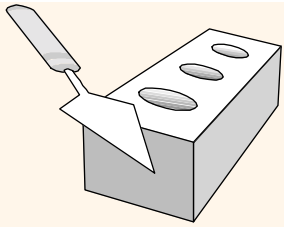
Original array

3	5	1	2	2	4	6	3	3	1

SDDC array

3	5	1	8	2	17	6	3	12	1

Query and Update Comparison (PS)



Original array A

3	5	1	2	2	4	6	3	3

PS Array

3	8	9	11	13	17	23	26	29

Query: 1+2+2+4=9 cost: 4

3	5	**1**	**2**	**2**	**4**	6	3	3

Query: 17-8=9 cost: 2

3	8	9	11	13	17	23	26	29

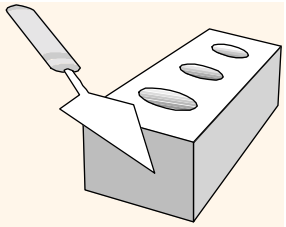
Update: A[4]=3 cost: 1

3	5	1	2	**3**	4	6	3	3

Update: A[4]=3 cost: 5

3	8	9	11	14	18	24	27	30

Query and Update Comparison (SRPS)



Original array A

3	5	1	2	2	4	6	3	3

SRPS Array (block size 3)

3	5	6	11	2	6	23	3	6

Query: 1+2+2+4=9 cost: 4

3	5	**1**	**2**	**2**	**4**	6	3	3

Query: 11+6-3-5=9 cost: 4

3	5	6	11	2	6	23	3	6

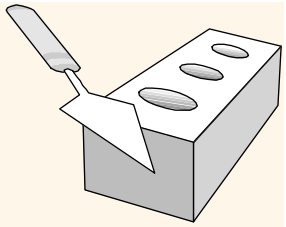
Update: A[4]=3 cost: 1

3	5	1	2	**3**	4	6	3	3

Update: A[4]=3 cost: 3

3	5	6	11	3	7	24	3	6

Query and Update Comparison (SDDC)



Original array A

3	5	1	2	2	4	6	3	3	1

SDDC Array

3	5	1	8	2	17	6	3	12	1

Query: 1+2+2+4=9 cost: 4

3	5	**1**	**2**	**2**	**4**	6	3	3	1

Query: 17-3-5=9 cost: 3

3	5	1	8	2	17	6	3	12	1

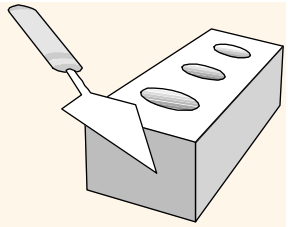
Update: A[4]=3 cost: 1

3	5	1	2	**3**	4	6	3	3	1

Update: A[4]=3 cost: 2

3	5	1	8	3	18	6	3	12	1

Query and Update Comparison

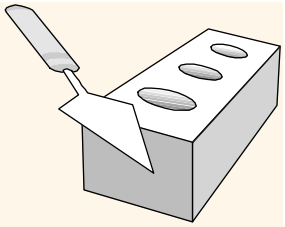


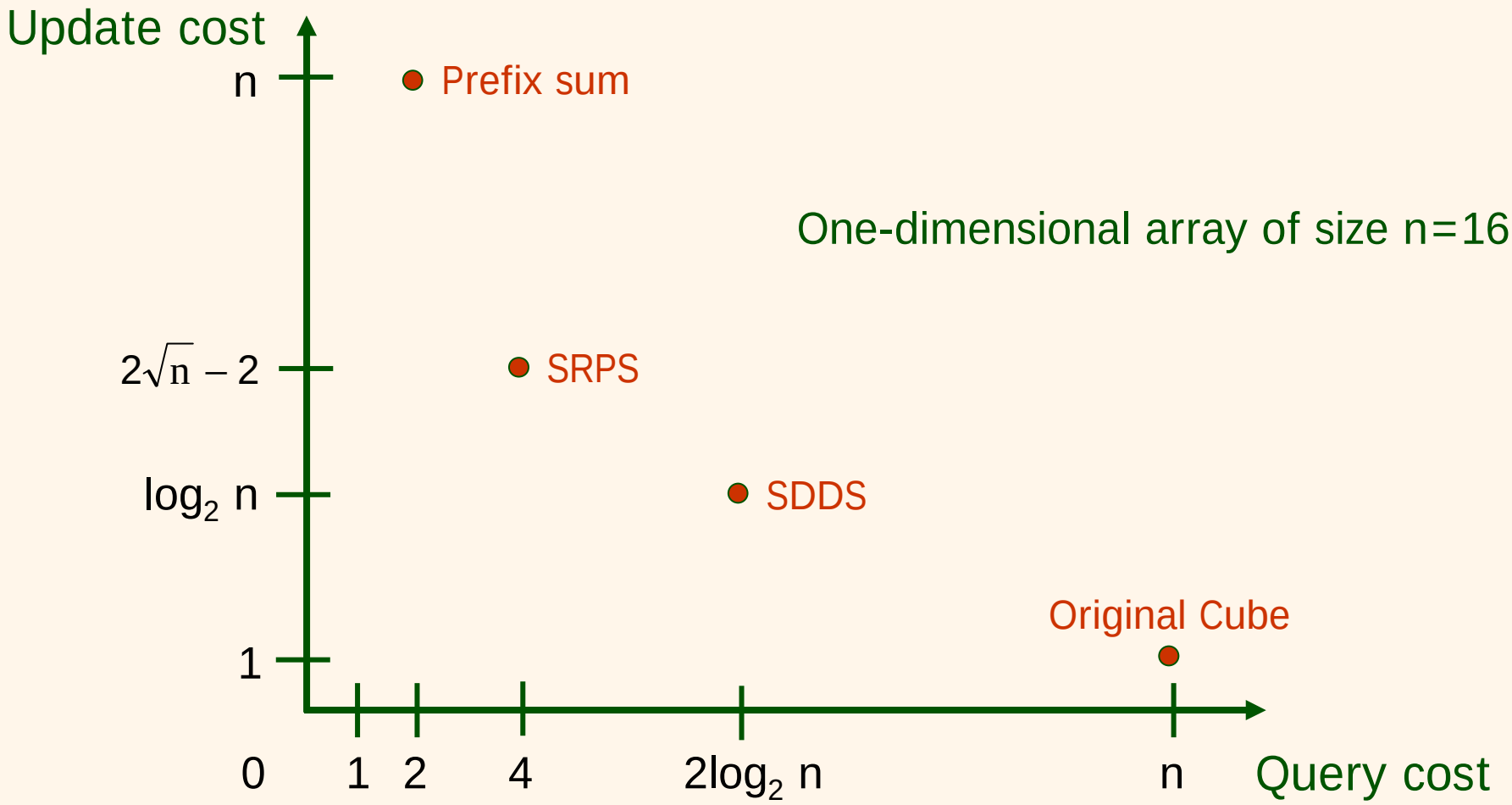
n: size of dimension

One-dimensional technique	Query cost (worst case)	Update cost (worst case)	Note
Original array	n	1	
Prefix Sum (PS)	2	n	
Space-Efficient Relative Prefix Sum (SRPS)	4	$2\sqrt{n}-2$	When n prefect square
	4	$2\sqrt{n}$	otherwise
Space-Efficient Dynamic Data Cube (SDDC)	$2[\log_2 n]$	$[\log_2 n]$	

Query-update cost tradeoffs for selected one-dimensional techniques

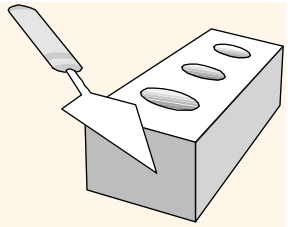
Query and Update Comparison





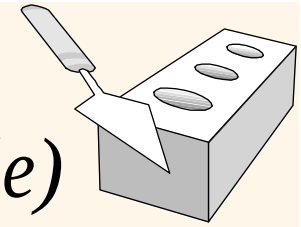
(Relative position of points can be different for other n)

Iterative Data Cube (IDC)



- Selecting a pre-aggregation method for each dimension:
  - The choice of the query-update cost tradeoff depends on the **properties of a dimension**.
  - Example guidelines:
    - If a hierarchy exists for an attribute and users typically query according to this hierarchy ⇒ Use SDDC or SPRS
    - A dimension has a few values (e.g., gender) ⇒ Use PS
- For high-dimensional data cubes one-dimensional techniques are applied along **each dimension**.

IDC with more then one dimension (example)



Original array

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

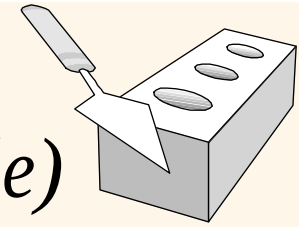
IDC array

PS →

SRPS (block size 2) ↓

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

IDC with more then one dimension (example)



Original array

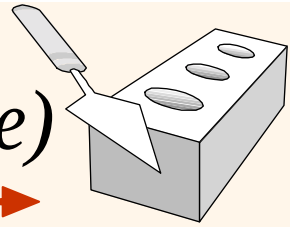
	0	1	2	3	4	5
0	3	5	1	2	2	4
1	7	3	2	6	8	7
2	2	4	2	3	3	3
3	3	2	1	5	3	5
4	4	2	1	3	3	4
5	2	3	3	6	1	8

IDC array

PS →

	0	1	2	3	4	5
0	3	8	9	11	13	17
1	7	10	12	18	26	33
2	2	6	8	11	14	17
3	3	5	6	11	14	19
4	4	6	7	10	13	17
5	2	5	8	14	15	23

IDC with more then one dimension (example)



PS

	0	1	2	3	4	5
0	3	8	9	11	13	17
1	7	10	12	18	26	33
2	2	6	8	11	14	17
3	3	5	6	11	14	19
4	4	6	7	10	13	17
5	2	5	8	14	15	23

Original array

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

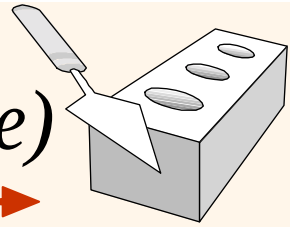
IDC array

SRPS

(block size 2)

	0	1	2	3	4	5
0	3	8	9	11	13	17
1	7	10	12	18	26	33
2	2	6	8	11	14	17
3	3	5	6	11	14	19
4	4	6	7	10	13	17
5	2	5	8	14	15	23

IDC with more then one dimension (example)



Original array

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

PS

	0	1	2	3	4	5
0	3	8	9	11	13	17
1	7	10	12	18	26	33
2	2	6	8	11	14	17
3	3	5	6	11	14	19
4	4	6	7	10	13	17
5	2	5	8	14	15	23

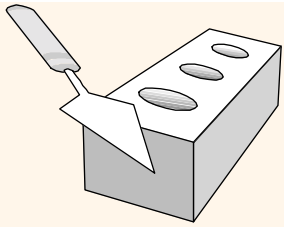
IDC array

SRPS

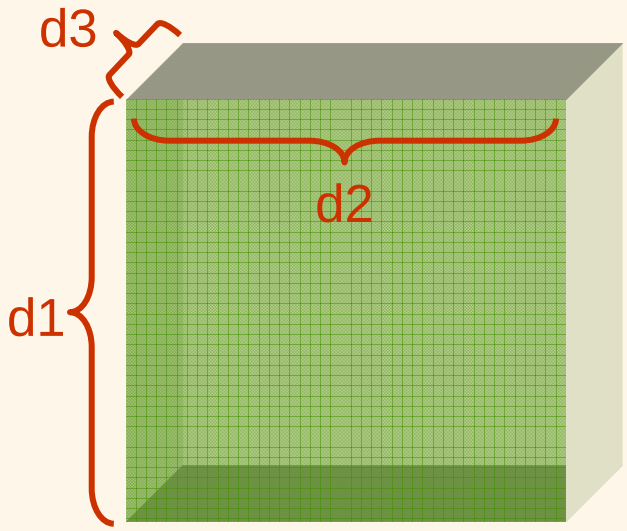
(block size 3)

	0	1	2	3	4	5
0	3	8	9	11	13	17
1	7	10	12	18	26	33
2	9	16	20	29	40	50
3	15	29	35	51	67	86
4	4	6	7	10	13	17
5	6	11	15	24	28	40

Query and Update Cost of IDC



- The worst case update and query costs of a high dimensional IDC is the **product** of the worst case costs of all the used one-dimensional techniques.



A data cube C has 3 dimensions, d1, d2 and d3:

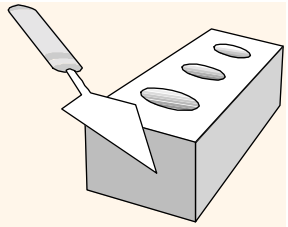
d1 size: n,
hierarchical attribute -> SDDC
d2 size: n
hierarchical attribute -> SDDC
d3 size: 2 -> PS

IDC C

worst case query cost: $2\log_2 n * 2\log_2 n * 2 = 8(\log_2 n)^2$

worst case update cost: $\log_2 n * \log_2 n * 2 = 2(\log_2 n)^2$

Conclusion



- IDC (Iterative Data Cube)
  - First pre-aggregation technique on data cube that can take the specific properties of different dimension attributes into account.
  - The different dimensions are handled independently.
- Features
  - Development, analysis and implementation are simplified.
  - A greater variety of query-update cost tradeoffs can be generated.