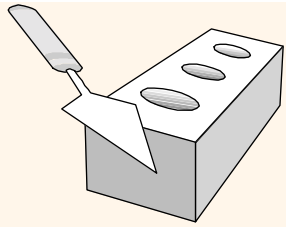
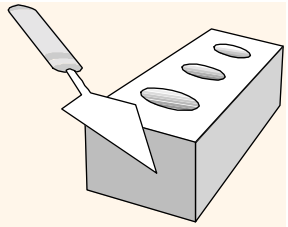


OLAP (Online Analytical Processing): Dynamic Data Cubes

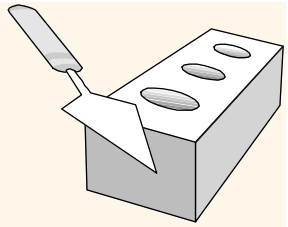
Excerpt from Presentation by S. Geffner

Content



- Introduction to Multidimensional Databases (from A.R. 20 and 21)

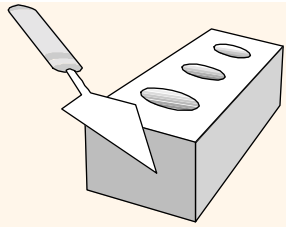
- Focus Application: OLAP
  - Prefix-Sum Data Cube (from A.R. 16)
  - **Dynamic Data Cube (from A.R. 17)**
  - Iterative Data Cube (from A.R. 18)
  - Wavelet-based approaches
    - Compact Data Cube (from A.R. 19)
    - ProPolyne (from A.R. 22 and 23)



The Dynamic Data Cube

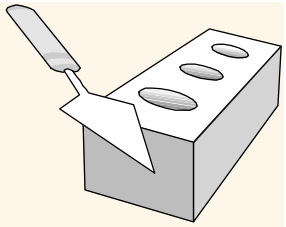
S. Geffner, D. Agrawal, and A. EL Abbadi

Outline



- Problem Description
- Dynamic Data Cube
- Improving Update
- Conclusion

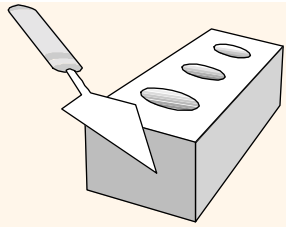
Problem Description: with original array A



Size= N^2

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

Problem Description: with original array A



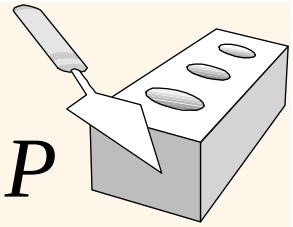
- Complexity:
  - Arbitrary range queries : $O(n^d)$
  - Update: $O(1)$

where:

d: # of dimension

n: size in each dimension

Problem Description: with prefix-sum array P



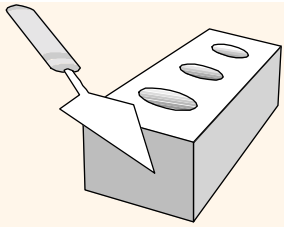
Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	3	5	1	2	2	4	6	3
1	7	3	2	6	8	7	1	2
2	2	4	2	3	3	3	4	5
3	3	2	5	3	5	2	8	
4	2	3	3	3	7	1		
5	2	3	3	6	1	8	5	2
6	4	5	2	7	1	9	3	3
7	2	4	2	2	3	1	9	1

Original array A

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	3	8	9	11	13	17	23	26
1	10	15	21	29	35	50	57	62
2	12	24	29	40	51	67	78	88
3	15	29	35	51	67	86	99	117
4	19	35	45	61	80	103	123	142
5	21	40	53	75	95	126	151	172
6	25	49	61	93	115	154	182	206
7	27	55	69	103	127	168	205	230

Precompute array P

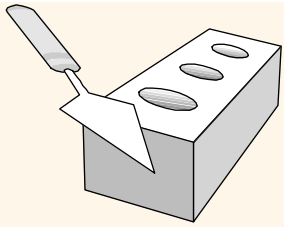
Query with Prefix-sum?



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

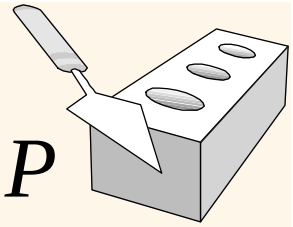
Update with Prefix-sum?



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

Problem Description: with prefix-sum array P



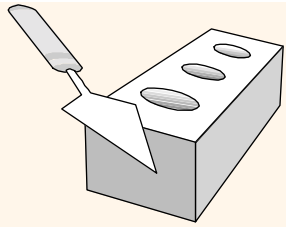
- Complexity:
  - Arbitrary range queries : *O(1)*
  - Update in worst case: $O(n^d)$

where:

d: # of dimension

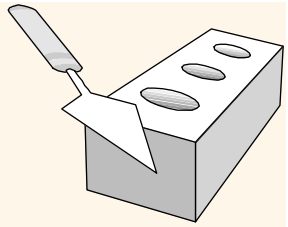
n: size in each dimension

Solution: Dynamic Data Cube



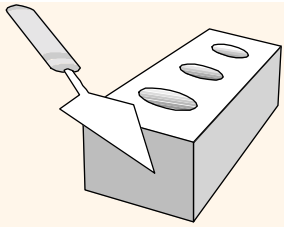
- Same as for Prefix-sum, DDC can also be applied to obtain count (special case of SUM) and average (SUM/COUNT).
- Generalization: applicable to any binary operator "+" which has an inverse operator "–" such that a+b-b=a
- We focus on SUM

Basic Data Structure: Overlay Box



- Overlay Box
  - Step1: A set of disjoint rectangles of equal size that completely partition cells of array A
  - Step2: Each box stores exactly $(k^d-(k-1)^d)$ values, where
    - k: The length of the overlay box in each dimension
    - d: # of dimension

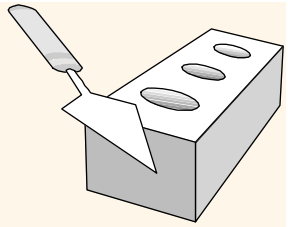
Overlay Box: Step 1



Original array A

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

Overlay Box: Step 2

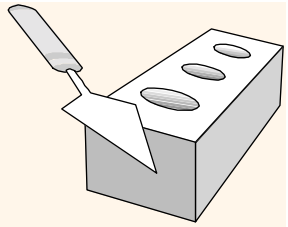


	Index	0	1	2	3	4	5	6	7
k	0				11				15
	1				29				33
	2				40				48
	3	15	29	35	51	16	35	48	66
k	4				10				15
	5				24				31
	6				42				47
	7	12	26	34	52	8	30	54	61

k k

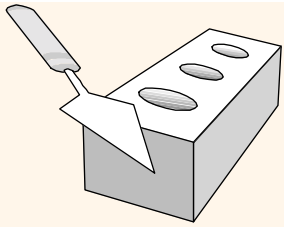
n=8, k=n/2=4, # of values in the overlay box=$(k^d-(k-1)^d)$

Dynamic Data Cube



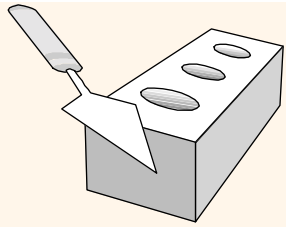
- A tree structure which **recursively** partitions original array A into overlay boxes
- Each overlay box will contain information regarding relative sums of the corresponding regions of A
- Organize overlay boxes into a tree to recursively partition array A into **non-overlapping** regions
- Each node forms children by dividing its range in each dimension **in half**

How to Construct DDC?



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

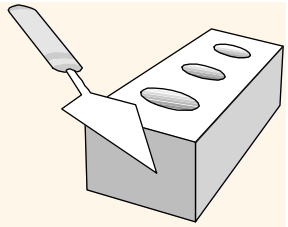
How to Construct DDC? Level 2



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

n=8, k=n/2, # of values in the overlay box=$(k^d-(k-1)^d)$

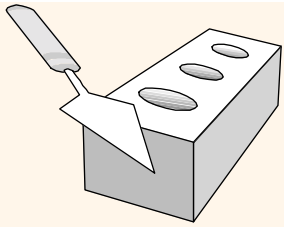
How to Construct DDC? Level 2



Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0				11				15
1				29				33
2				40				48
3	15	29	35	51	16	35	48	66
4				10				15
5				24				31
6				42				47
7	12	26	34	52	8	30	54	61

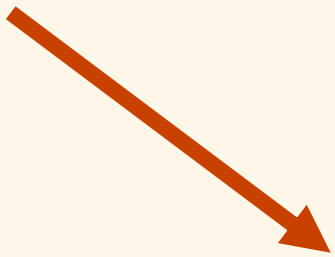
n=8, k=n/2, # of values in the overlay box=$(k^d-(k-1)^d)$

How to Construct DDC? Level 1



Index	*0*	*1*	*2*	*3*
0				11
1				29
2				40
3	15	29	35	51

k=n/2=4

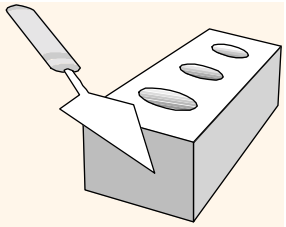


	8		3
10	18	3	11
	6		5
5	11	3	11

k=n/4=2

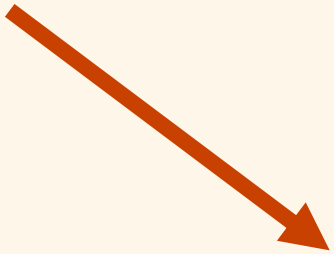
# of values in the overlay box=$(k^d-(k-1)^d)$ =3

How to Construct DDC? Level 0



Index	*0*	*1*	*2*	*3*
0		8		3
1	10	18	3	11
2		6		5
3	5	11	3	11

k=n/4=2



3	5
7	3

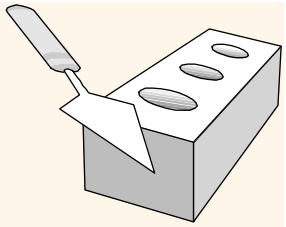
1	2
2	6

k=n/8=1

3	5
7	3

2	3
1	5

The DDC Hierarchy



Level 2

(Root node)

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0				11				15
1				29				33
2				40				48
3	15	29	35	51	16	35	48	66
4				10				15
5				24				31
6				42				47
7	12	26	34	52	8	30	54	61

Level 1

	8		3
10	18	3	11
	6		5
5	11	3	11

	6		9
10	21	7	12
	6		9
6	14	6	19

	6		4
6	11	4	13
	9		9
6	15	4	13

	7		8
4	16	12	15
	10		6
4	14	12	16

Level 0

(Leaf node)

3	5
7	3

1	2
2	6

2	4
8	7

6	3
1	2

2	4
3	2

2	3
1	5

3	3
3	5

4	5
2	8

4	2
2	3

1	3
3	6

3	4
1	8

7	1
5	2

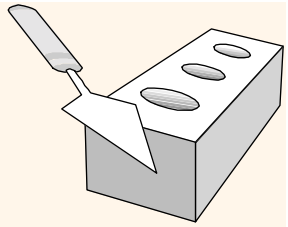
4	5
2	4

2	7
2	2

1	9
3	1

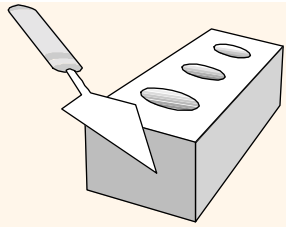
3	3
9	1

How to Query DDC?



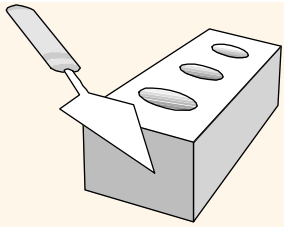
Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0				11				15
1				29				33
2				40				48
3	15	29	35	51	16	35	48	66
4				10				15
5				24			*	31
6				42				47
7	12	26	34	52	8	30	54	61

How to Query DDC? (cont'd)



Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0				11				15
1				29				33
2				40				48
3	15	29	35	**51**	16	35	**48**	66
4				10				15
5				**24**			*	31
6				42				47
7	12	26	34	52	8	30	54	61

How to Query DDC? (cont'd)



Level 2

(Root node)

	0	1	2	3	4	5	6	7
0				11				15
1				29				33
2				40				48
3	15	29	35	**51**	16	35	**48**	66
4				10				15
5				**24**			*	31
6				42				47
7	12	26	34	52	8	30	54	61

51+48+24+16+12=151

Level 1

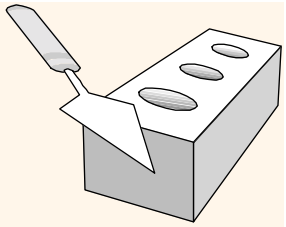
	8		3
10	18	3	11
	6		5
5	11	3	11

	6		9
10	21	7	12
	6		9
6	14	6	19

	6		4
6	11	4	13
	9		9
6	15	4	13

	7		8
4	**16**	***12**	15
	10		6
4	14	12	16

How to Update DDC?



Level 2

(Root node)

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0				11				15
1				29				33
2				40				48
3	15	29	35	51	16	35	48	66
4				10				15
5				24				31
6				42				47
7	12	26	34	52	8	30	54	61

Level 1

	8		3
10	18	3	11
	6		5
5	11	3	11

	6		9
10	21	7	12
	6		9
6	14	6	19

	6		4
6	11	4	13
	9		9
6	15	4	13

	7		8
4	16	12	15
	10		6
4	14	12	16

Level 0

(Leaf node)

3	5
7	3

1	2
2	6

2	4
8	7

6	3
1	2

2	4
3	2

2	3
1	5

3	3
3	5

4	5
2	8

4	2
2	3

1	3
3	6

3	4
1	8

7	1
5	2

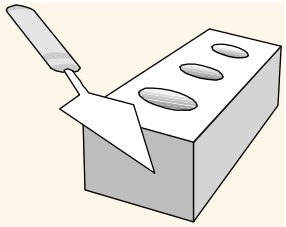
4	5
2	4

2	7
2	2

1	9
3	1

3	3
9	1

How to Update DDC? (cont'd)



Level 2

(Root node)

	0	1	2	3	4	5	6	7
0				11				15
1				29				33
2				40				48
3	15	29	35	51	16	35	48	66
4				10				15
5				24			*	31
6				42				47
7	12	26	34	52	8	30	54	61

Level 1

	8		3
10	18	3	11
	6		5
5	11	3	11

	6		9
10	21	7	12
	6		9
6	14	6	19

	6		4
6	11	4	13
	9		9
6	15	4	13

	7		8
4	16	12	15
	10		6
4	14	12	16

Level 0

(Leaf node)

3	5
7	3

1	2
2	6

2	4
8	7

6	3
1	2

2	4
3	2

2	3
1	5

3	3
3	5

4	5
2	8

4	2
2	3

1	3
3	6

3	4
1	8

7	1
*5	2

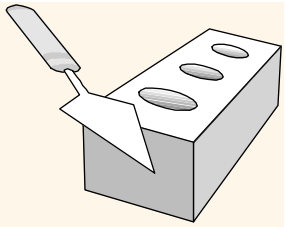
4	5
2	4

2	7
2	2

1	9
3	1

3	3
9	1

How to Update DDC? (cont'd)



Level 2

(Root node)

	0	1	2	3	4	5	6	7
0				11				15
1				29				33
2				40				48
3	15	29	35	51	16	35	48	66
4				10				15
5				24				32
6				42				48
7	12	26	34	52	8	30	55	62

Level 1

	8		3
10	18	3	11
	6		5
5	11	3	11

	6		9
10	21	7	12
	6		9
6	14	6	19

	6		4
6	11	4	13
	9		9
6	15	4	13

	7		8
4	16	13	16
	10		6
4	14	12	16

Level 0

(Leaf node)

3	5
7	3

1	2
2	6

2	4
8	7

6	3
1	2

2	4
3	2

2	3
1	5

3	3
3	5

4	5
2	8

4	2
2	3

1	3
3	6

3	4
1	8

7	1
6	2

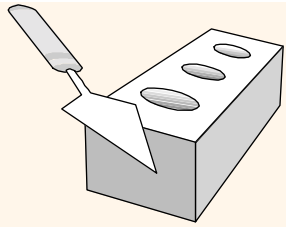
4	5
2	4

2	7
2	2

1	9
3	1

3	3
9	1

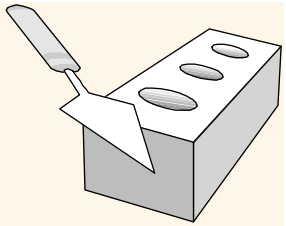
Query and Update Complexity with DDC



- Query cost for arbitrary range queries : *O(log n)*

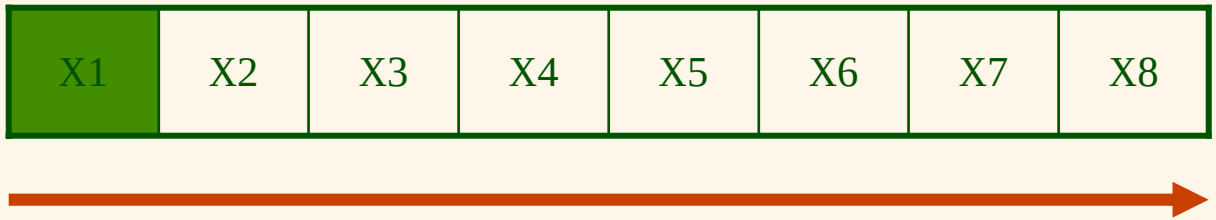
- Update cost in worst case should be evaluated considering:
  1. *O(log n)* hierarchy traversal cost
     - Whichever cell you update, only one overlay box is updated at each tree level
  2. The cost of updating the values in the relevant overlay box at each level

  The 2nd cost might be high depending on the placement of the updated cell. Hence, the total cost may become as bad as *O(n)*

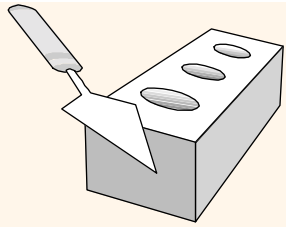


Improving Update: Problem

- The high update cost of the **overlay boxes** is a consequence of dependencies between **successive row sum values**



Improving Update: Solution

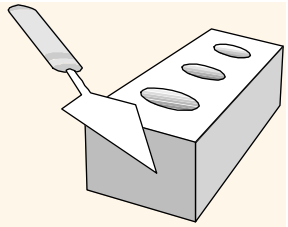


Storing row sum values in an array

Store row sum values separately in

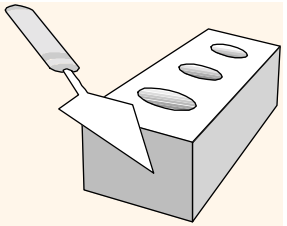
Cumulative B Tree (B^C Tree)

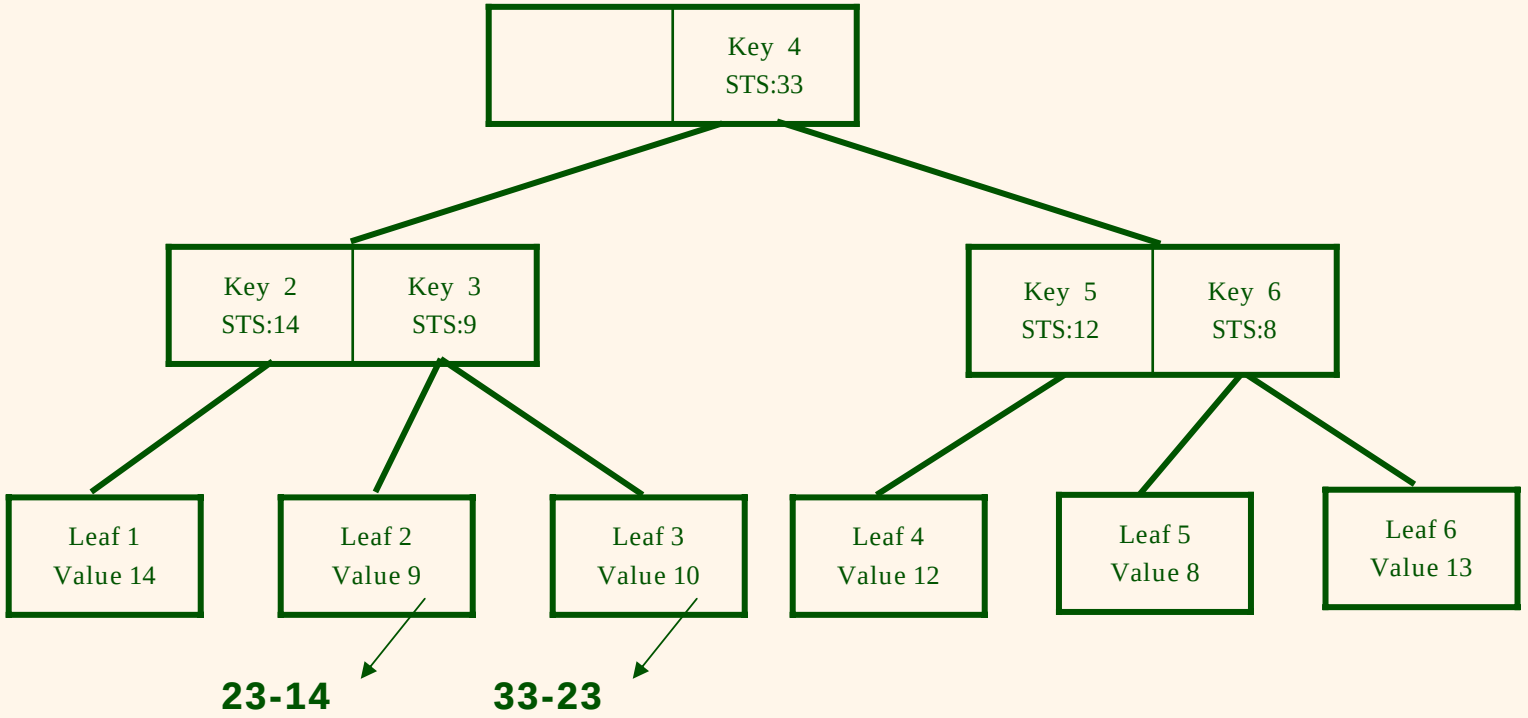
How to Construct B^C Tree?



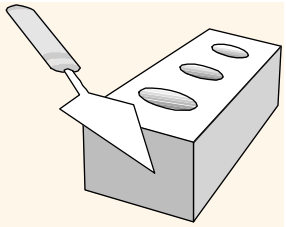
- Each leaf of the B^C tree corresponds to one row-sum cell
- Interior nodes of the B^C tree maintain subtree sums (STS)
- For each node entry, the STS stores the sum of the subtree from the left branch associated with leaf value

How to Construct B^C Tree?

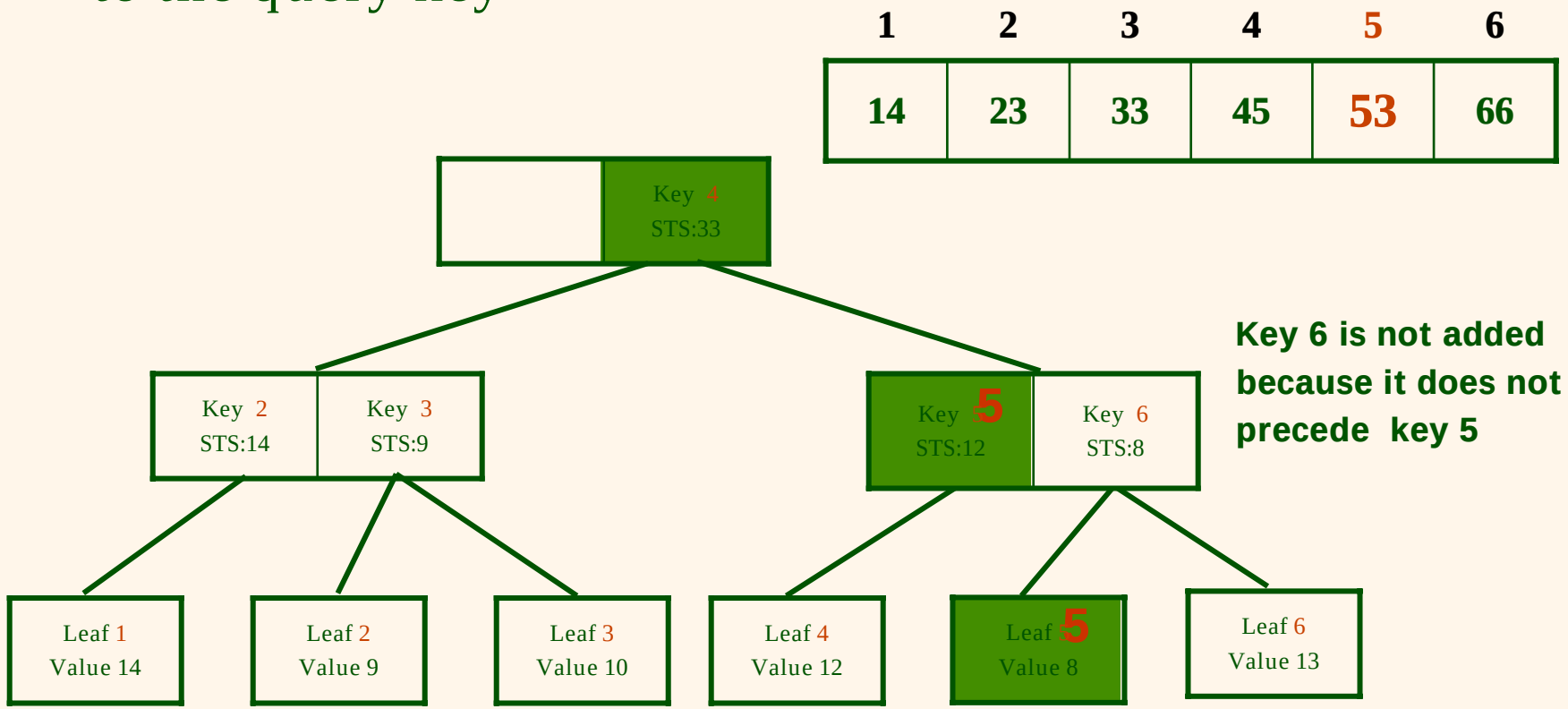




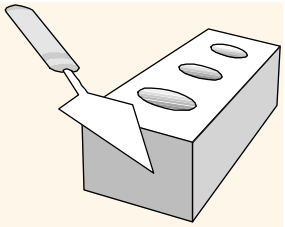
Query with B^C Tree?



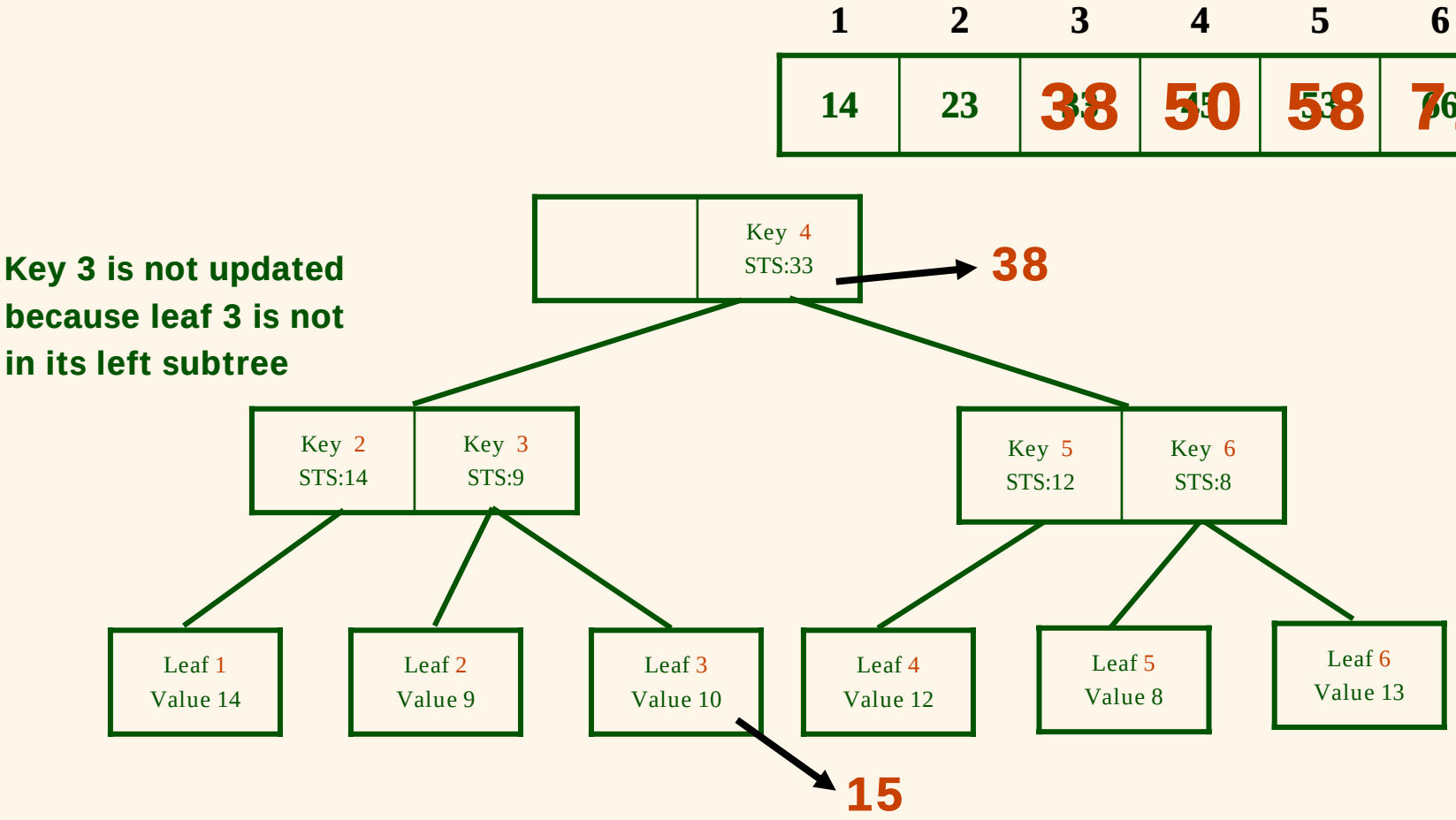
- Traverse the tree using the cell's index as the key
- If descending to a node's right branch, we sum each preceding STS in the node with the key less than or equal to the query key



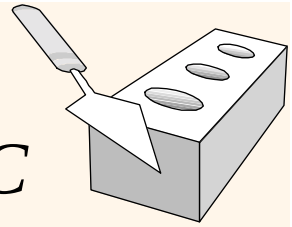
Update with B^C Tree?



- Reflect the change using a bottom-up method

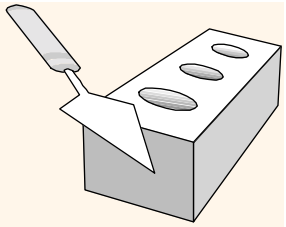


Query and Update Complexity with Improved DDC



- A two dimensional overlay box has two groups of row sum values, each of which is one dimensional
- In general, an overlay box of *d* dimensions has *d* groups of row sum values and each group is (*d-1*) dimensional
- Hence, both query and update complexity is $O(\log^d n)$

Conclusion



Method	Query	Update
Naïve approach	$O(n^d)$	$O(1)$
Prefix sum	$O(1)$	$O(n^d)$
DDC	$O(\log^d n)$	$O(\log^d n)$