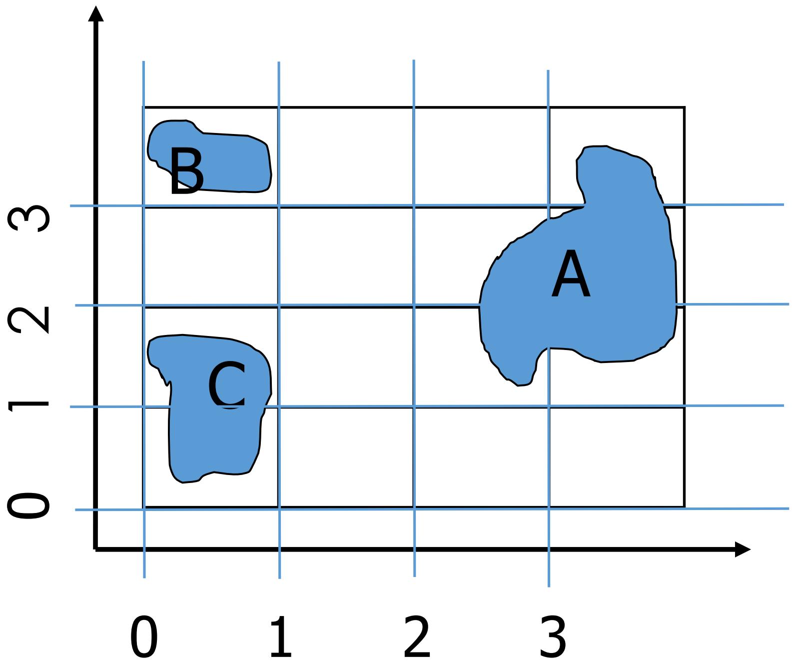
Introduction to Spatial Computing CSE 555

Spatial Indexing Techniques for Secondary Memory

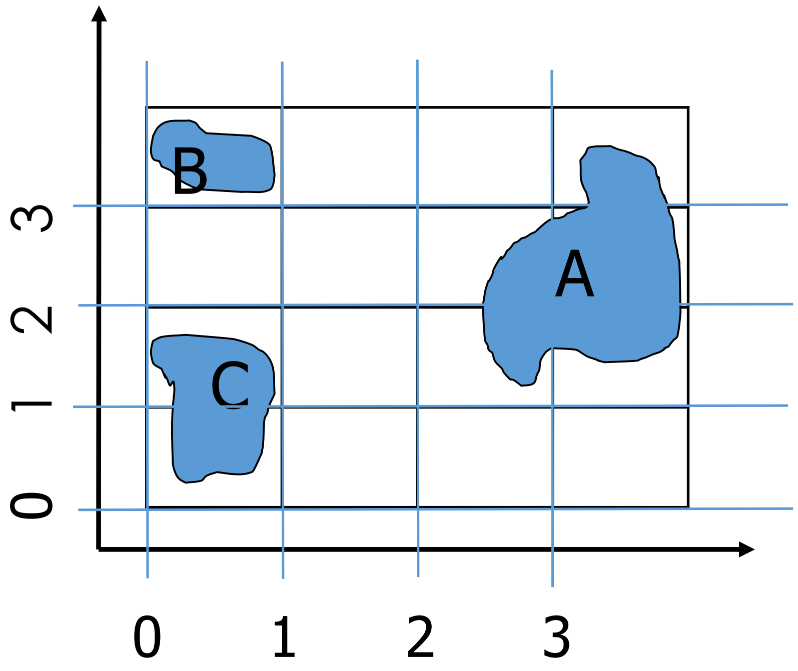
Some slides adapted from Spatial Databases: *A Tour by Shashi Shekhar Prentice Hall (2003)*

Scenario for Designing Spatial Indexes



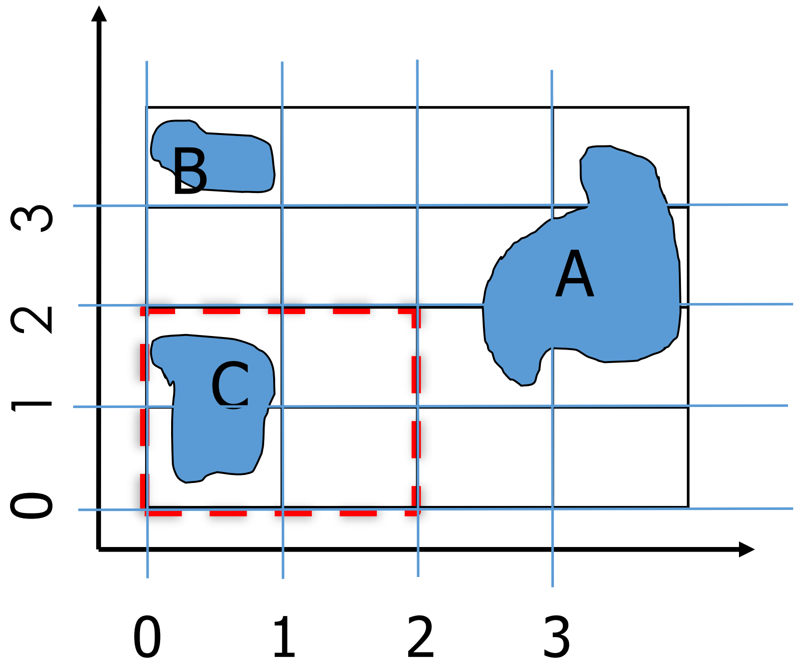
- **Goal:** Store spatial objects A,B and C in storage system such that following queries can be executed efficiently.
- **Point Queries:**
- **Range Queries:**
- **Nearest Neighbor Queries**
- **Spatial Joins:**

Scenario for Designing Spatial Indexes



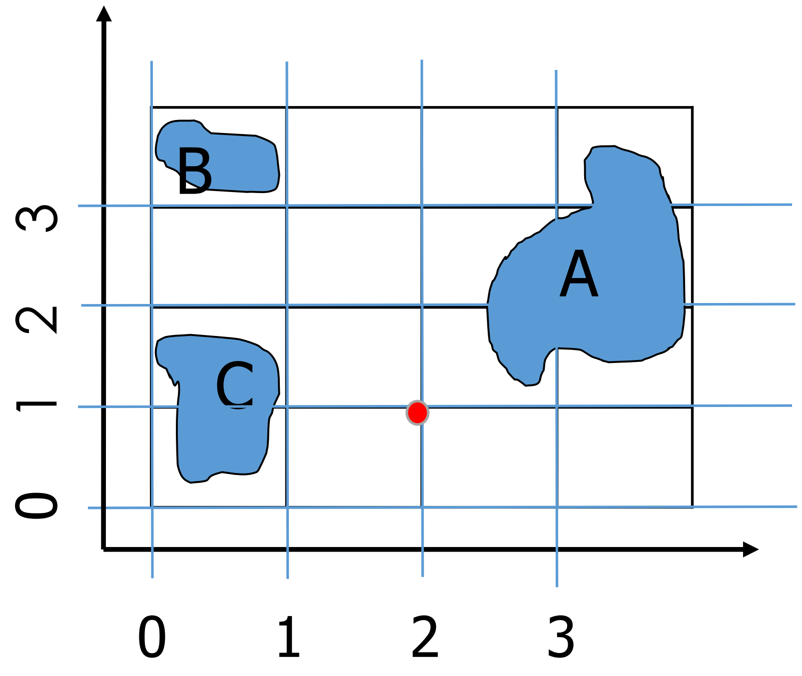
- **Goal:** Store spatial objects A,B and C in storage system such that following queries can be executed efficiently.
- **Point Queries:**
  - *Given an object search if it exists in the database or not*
  - *Example: Return the spatial object located at (3,2)*
- **Range Queries:**
- **Nearest Neighbor Queries**
- **Spatial Joins:**

Scenario for Designing Spatial Indexes



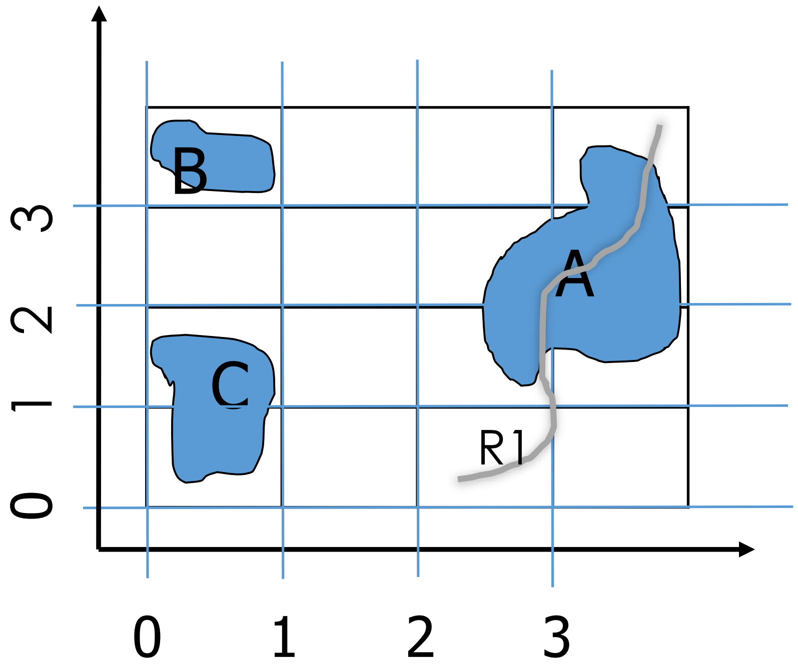
- **Goal:** Store spatial objects A,B and C in storage system such that following queries can be executed efficiently.
- **Point Queries:**
- **Range Queries:**
  - Return the objects which lie within the defined range of x and y
  - Example: return objects which lie in the rectangle defined by 0<x<2 and 0<y<2
- **Nearest Neighbor Queries**
- **Spatial Joins:**

Scenario for Designing Spatial Indexes



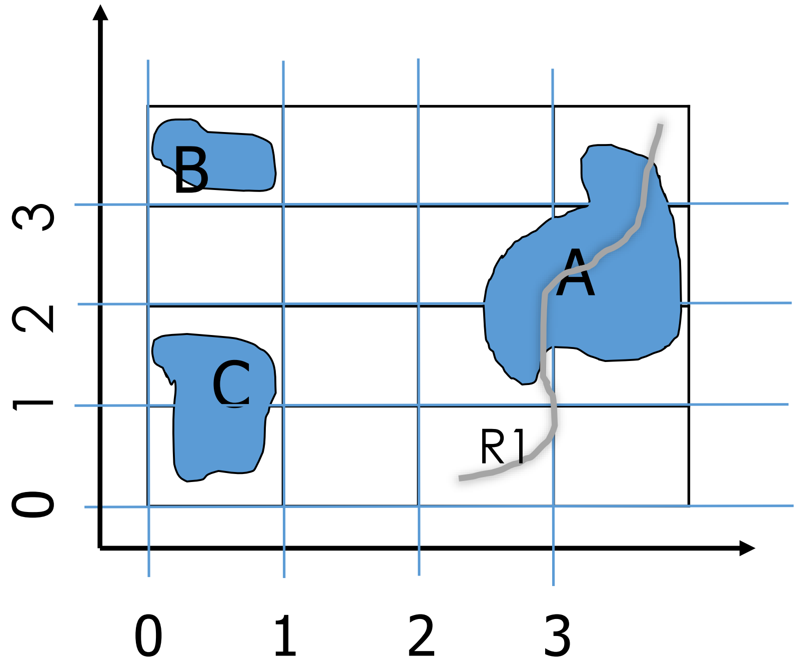
- **Goal:** Store spatial objects A,B and C in storage system such that following queries can be executed efficiently.
- **Point Queries:**
- **Range Queries:**
- **Nearest Neighbor Queries**
  - Find the nearest spatial object (or k nearest spatial objects) of the point (2,1)
- **Spatial Joins:**

Scenario for Designing Spatial Indexes



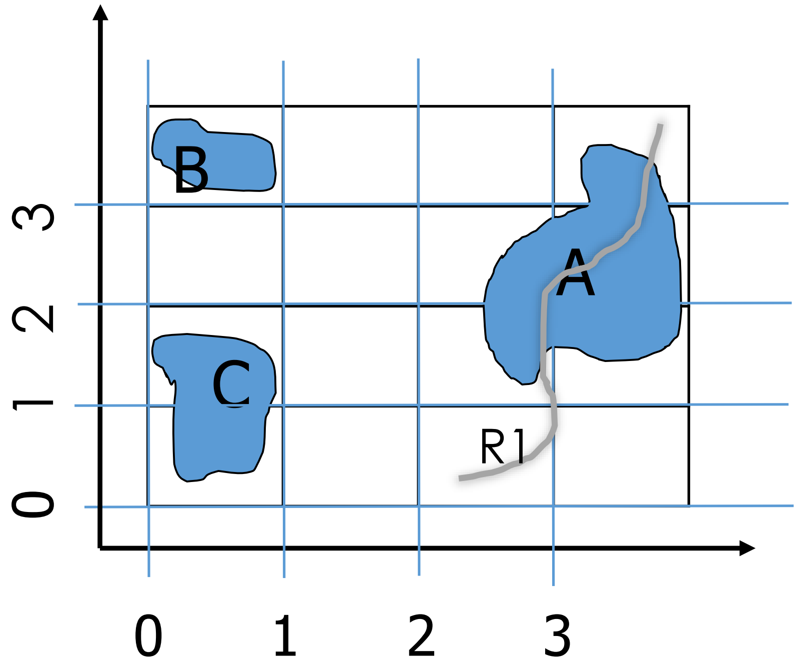
- **Goal:** Store spatial objects A,B and C in storage system such that following queries can be executed efficiently.
- **Point Queries:**
- **Range Queries:**
- **Nearest Neighbor Queries**
- **Spatial Joins:**
  - Find the spatial objects which intersect the object R1

Scenario for Designing Spatial Indexes



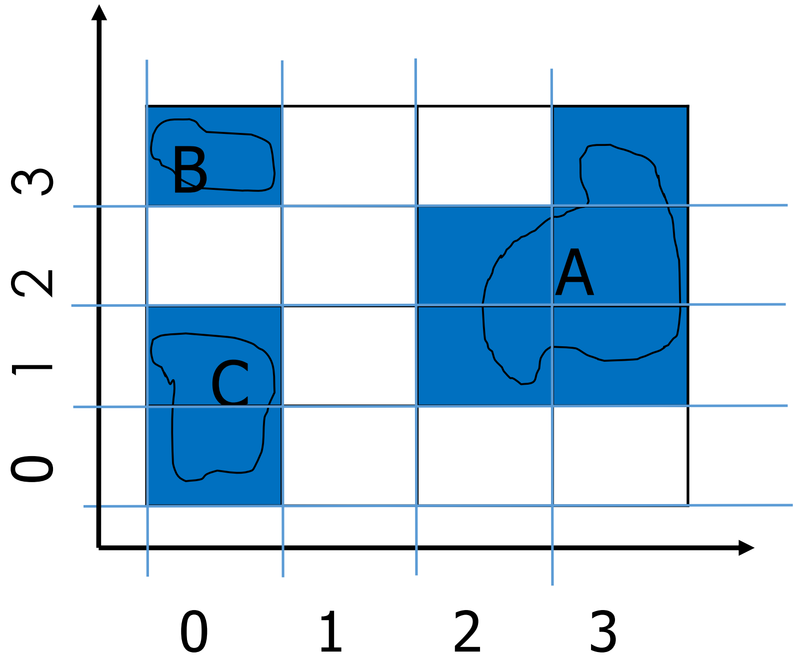
- Had these objects been a 1-dimensional in nature, e.g., real numbers, strings etc.
- A simple B+ tree would be constructed over these.
- Can easily get O(log n) complexity for all the queries (except the join query) mentioned in the previous slides.

Scenario for Designing Spatial Indexes



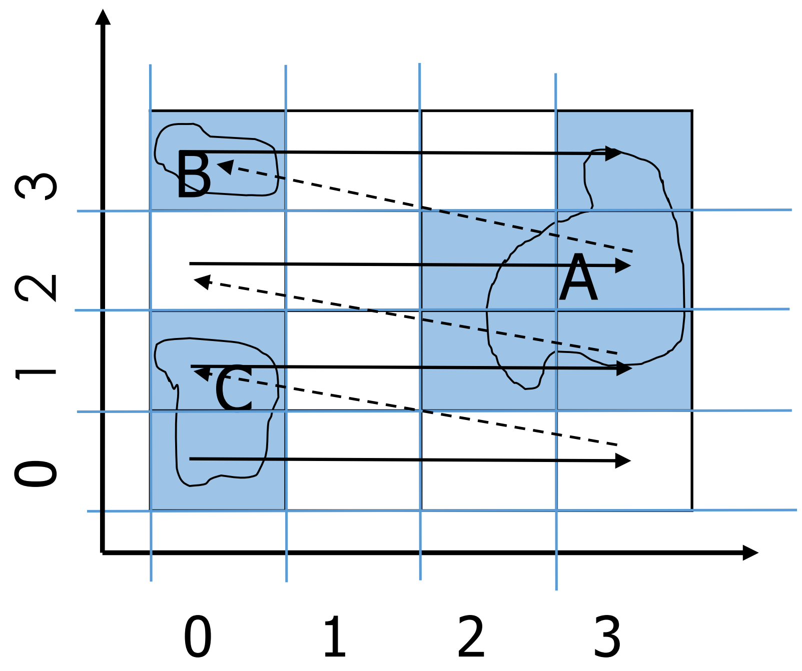
- How to get ordering in 2-Dimensions?
- Once we get ordering we can try B+ tree again for spatial objects.

Towards Getting an Order Basics



- Approximate objects with cells.
- Helps in getting a continuous space to work with easer to handle.
- Would have to map back whenever necessary (for the queries and results).

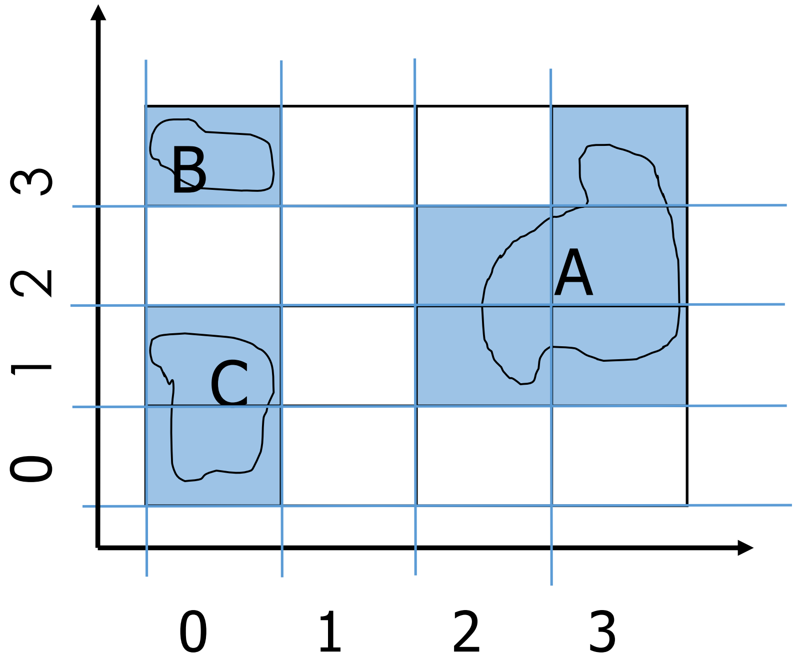
Towards Getting an Order



First Attempt

- **Order on Y then X: (0,0) (1,0) (2,0) (3,0) (0,1) (1,1) (2,1) (3,1) (0,2) (1,2) (2,2) (3,2) (0,3) (1,3) (2,3) (3,3)**

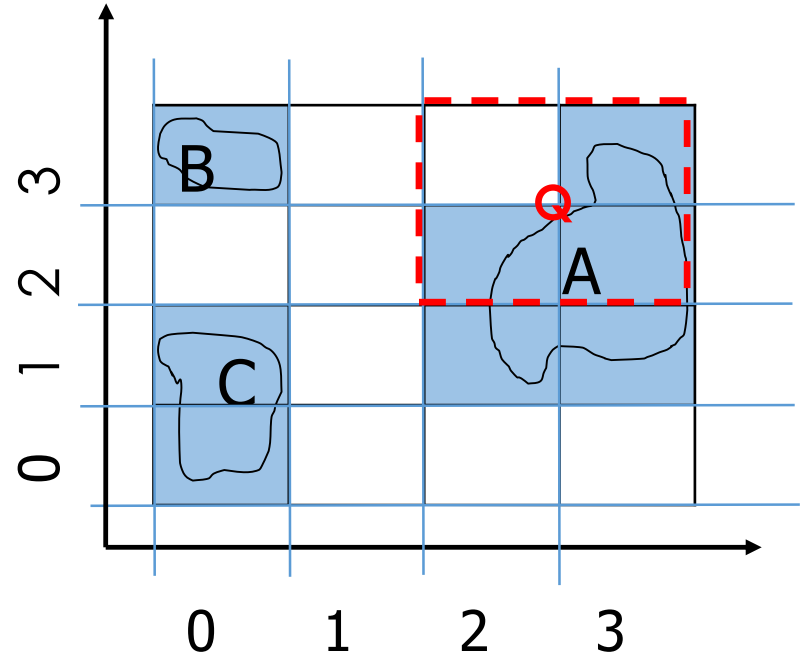
Towards Getting an Order



First Attempt

- **Order on Y then X: (0,0) (1,0) (2,0) (3,0) (0,1) (1,1) (2,1) (3,1) (0,2) (1,2) (2,2) (3,2) (0,3) (1,3) (2,3) (3,3)**
- Insert tuples <(0,0),C>; <(0,1),C>; <(2,1),A>; <(3,1),A>; <(2,2),A>; <(3,2),A>; <(0,3),B>; <(3,3),A>; in a B+ tree.
- These would be order of leaves in the B+ tree

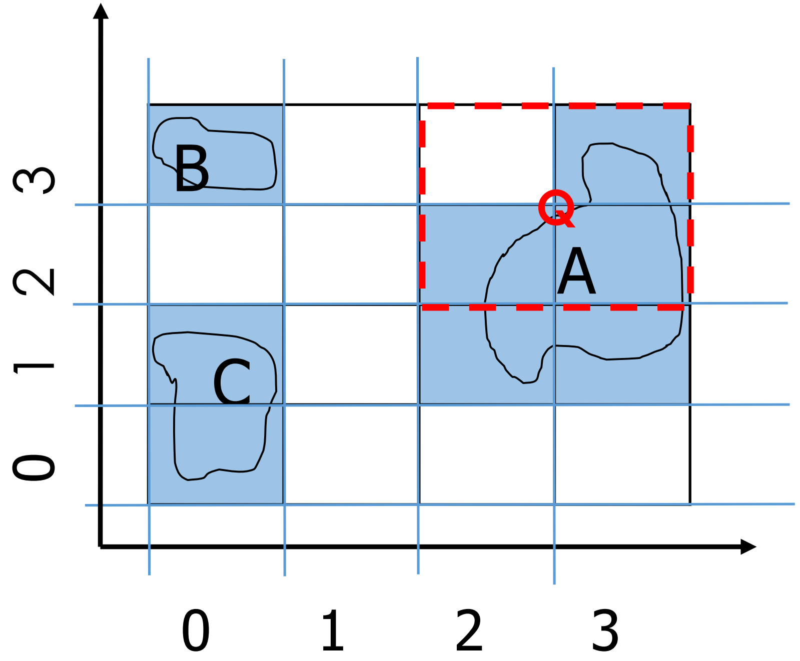
Towards Getting an Order



First Attempt (Y then X)

- **Insert tuples <(0,0),C>; <(0,1),C>; <(2,1),A>; <(3,1),A>; <(2,2),A>; <(3,2),A>; <(0,3),B>; <(3,3),A>; in a B+ tree.**
- **Range Query: Retrieve the objects whose 2=<x=<3 and 2=<y=<3**

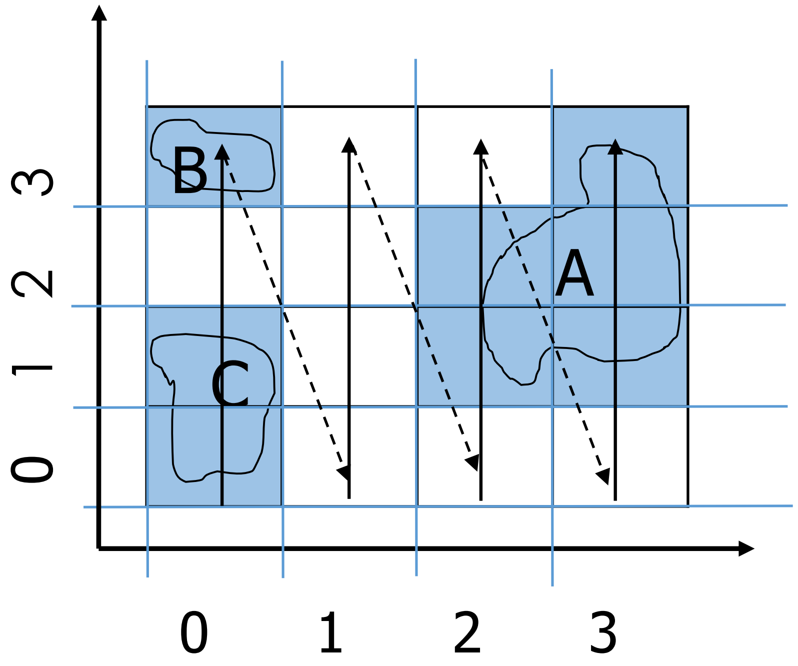
Towards Getting an Order



- **First Attempt (Y then X)**
  - **Insert tuples <(0,0),C>; <(0,1),C>; <(2,1),A>; <(3,1),A>; <(2,2),A>; <(3,2),A>; <(0,3),B>; <(3,3),A>; in a B+ tree.**
  - **Range Query: Retrieve the objects whose 2=<x=<3 and 2=<y=<3**

Not really in the range but still got in

Towards Getting an Order

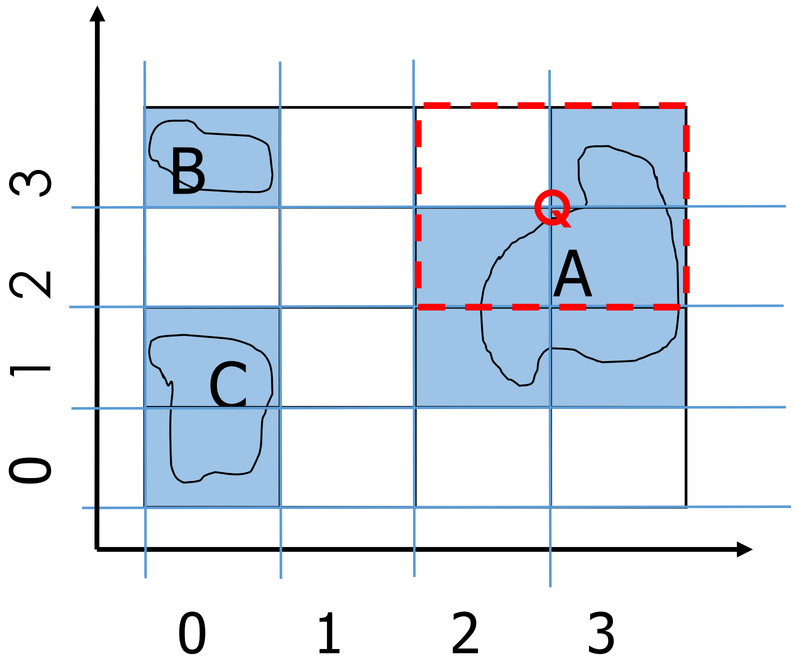


Second Attempt (X then Y)

Order on X then Y:

(0,0) (0,1) (0,2) (0,3) (1,0) (1,1) (1,2) (1,3) (2,0) (2,1) (2,2) (2,3) (3,0) (3,1) (3,2) (3,3)

Towards Getting an Order



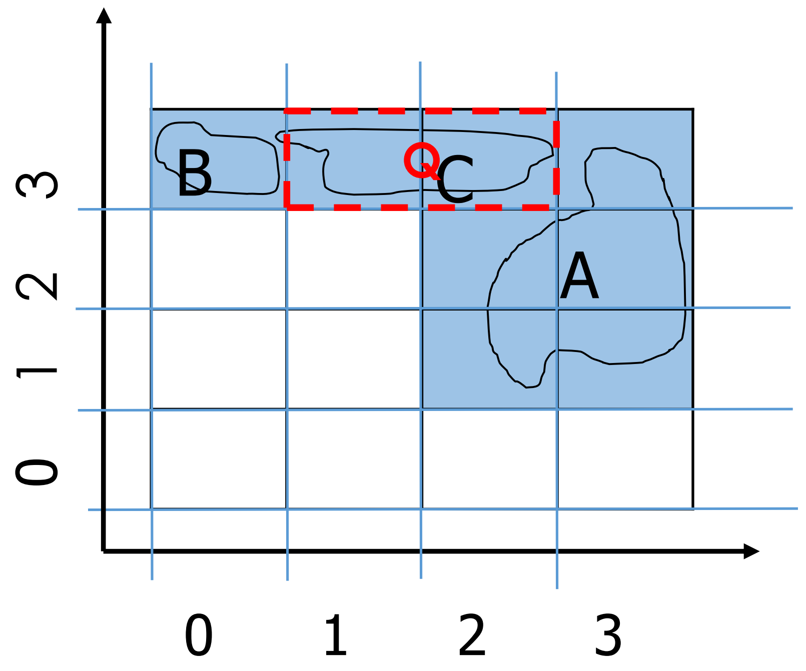
Second Attempt (X then Y)

Order on X then Y:

(0,0) (0,1) (0,2) (0,3) (1,0) (1,1) (1,2) (1,3) (2,0) (2,1) (2,2) (2,3) (3,0) (3,1) (3,2) (3,3)

Range Query 2<x<3 & 2<y<3:
Little better this time

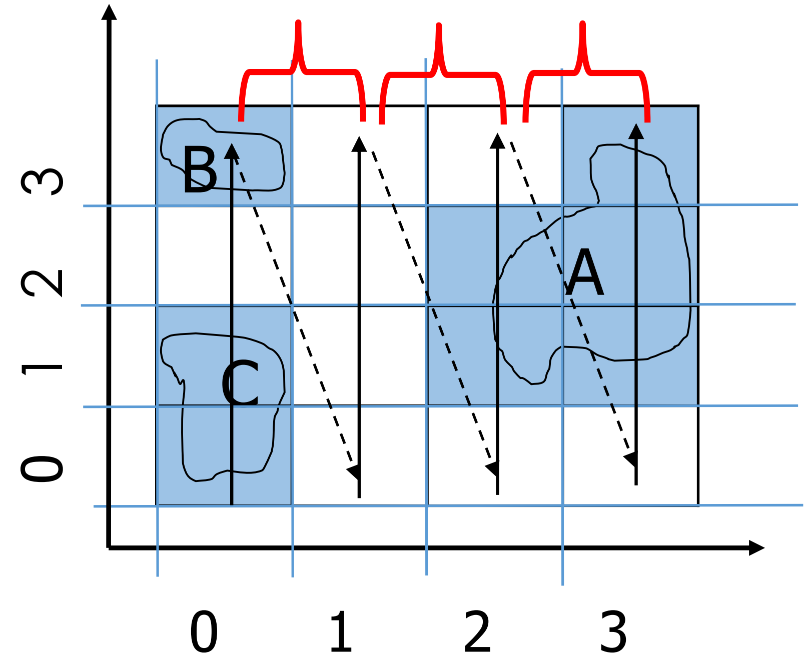
How about in this scenario?



Range Query: Retrieve all objects in this range 1=<x=<2 & y=3 ?

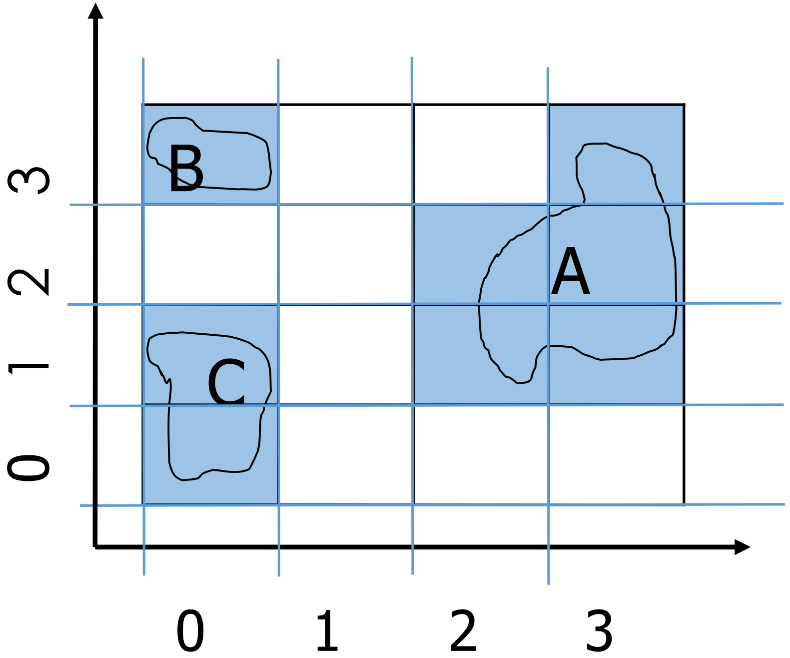
Towards Getting an Order

Neighboring Cells but far apart in the ordering



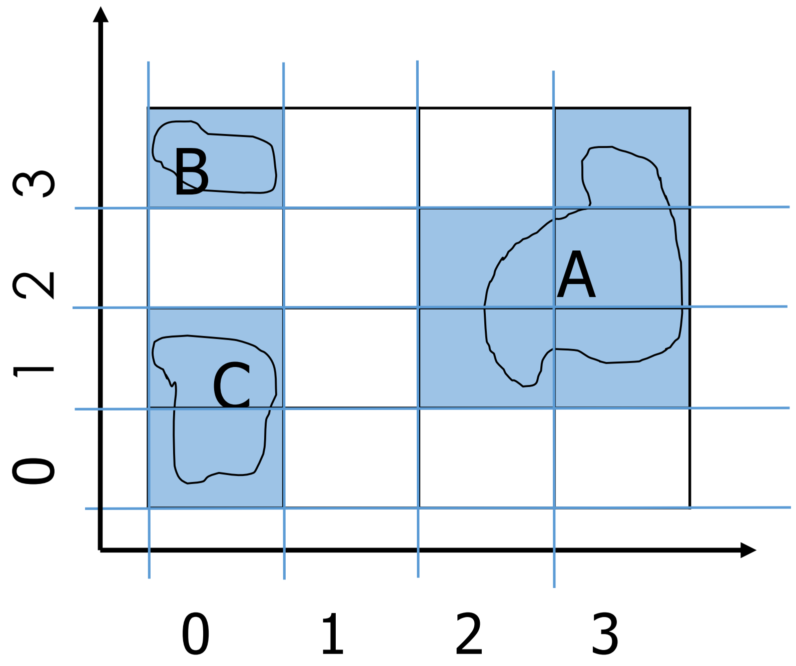
- **Problem with these orderings**: Cells which are close to each other might get spread out and occupy places quite far from each other.
- Need a ordering which can preserve spatial locality in both x and y directions as much as possible!
- **Cannot get 100%**

Z-Order curve



3	(0,3)	(1,3)	(2,3)	(3,3)
2	(0,2)	(1,2)	(2,2)	(3,2)
1	(0,1)	(1,1)	(2,1)	(3,1)
0	(0,0)	(1,0)	(2,0)	(3,0)
	0	1	2	3

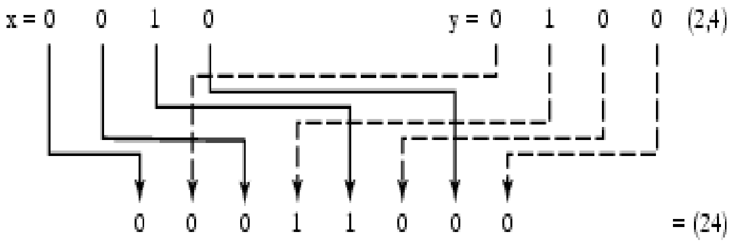
Z-Order curve

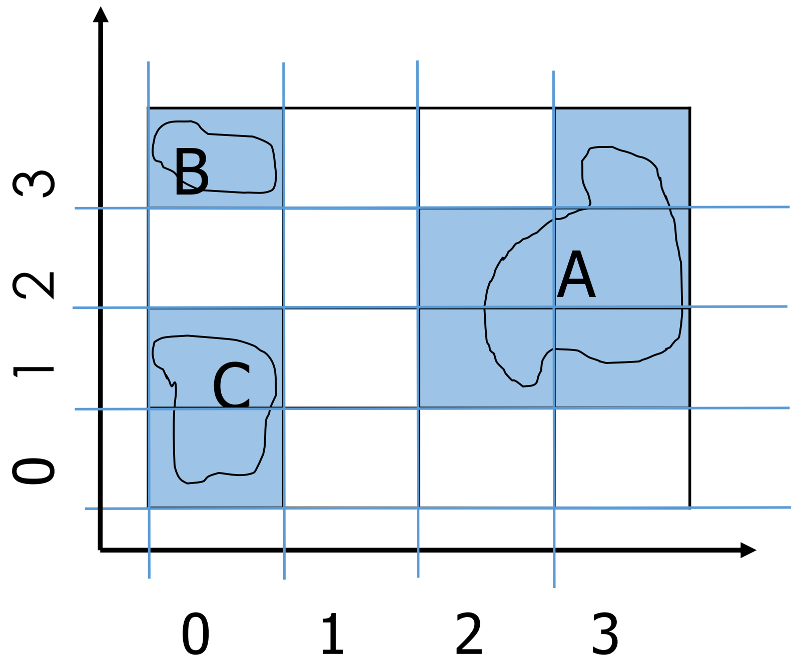


	0	1	2	3
3	00 11	01 11	10 11	11 11
2	00 10	01 10	10 10	11 10
1	00 01	01 01	10 01	11 01
0	00 00	01 00	10 00	11 00

Write the X and Y coordinates in Binary Form

Z-Order curve

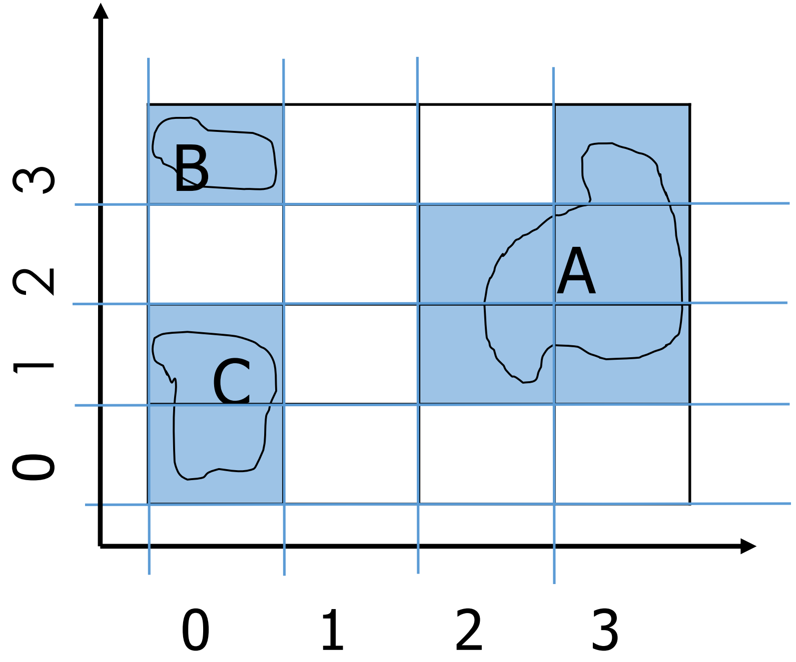




	0	1	2	3
3	**0101**	**0111**	**1101**	**1111**
2	**0100**	**0110**	**1100**	**1110**
1	**0001**	**0011**	**1001**	**1011**
0	**0000**	**0010**	**1000**	**1010**

Interleave them to create one string

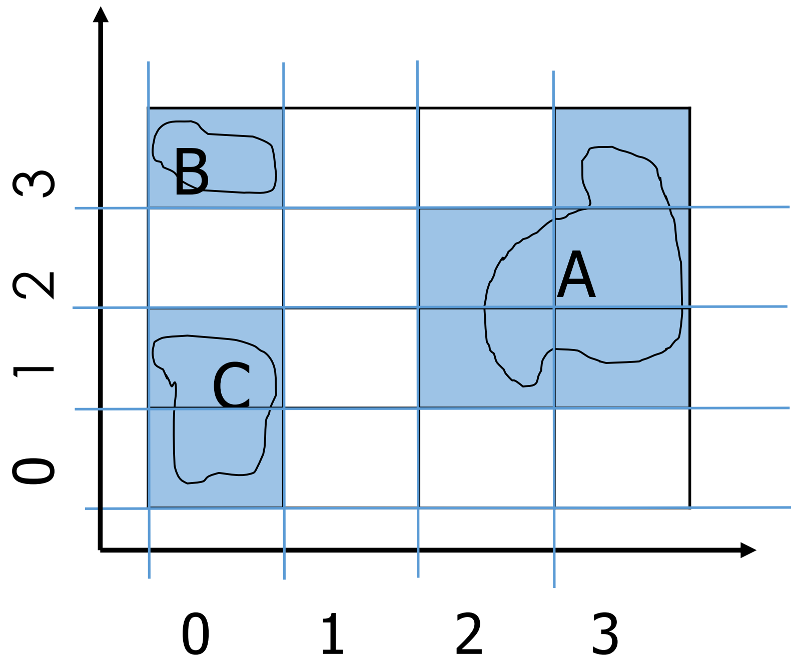
Z-Order curve



y \ x	0	1	2	3
3	5	7	13	15
2	4	6	12	14
1	1	3	9	11
0	0	2	8	10

Convert the bit strings to its corresponding decimal

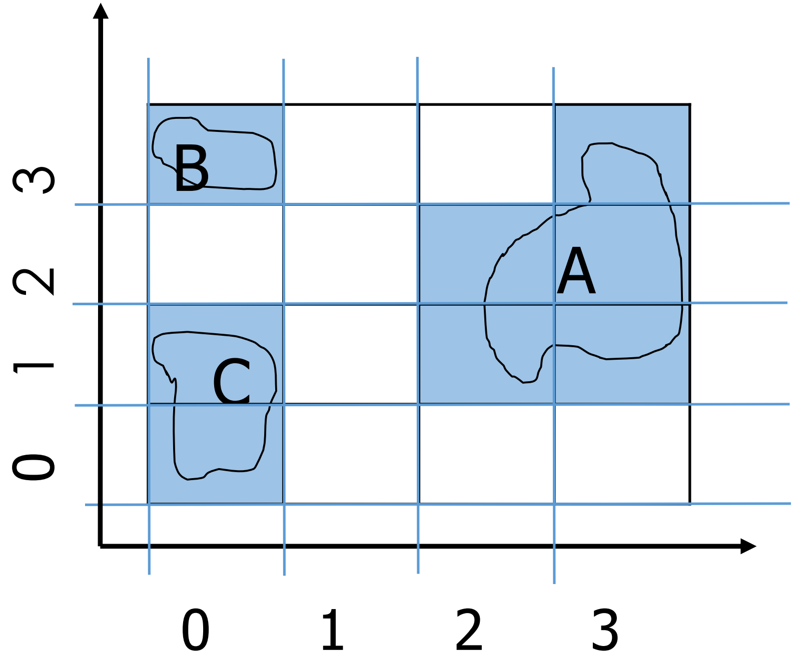
Z-Order curve

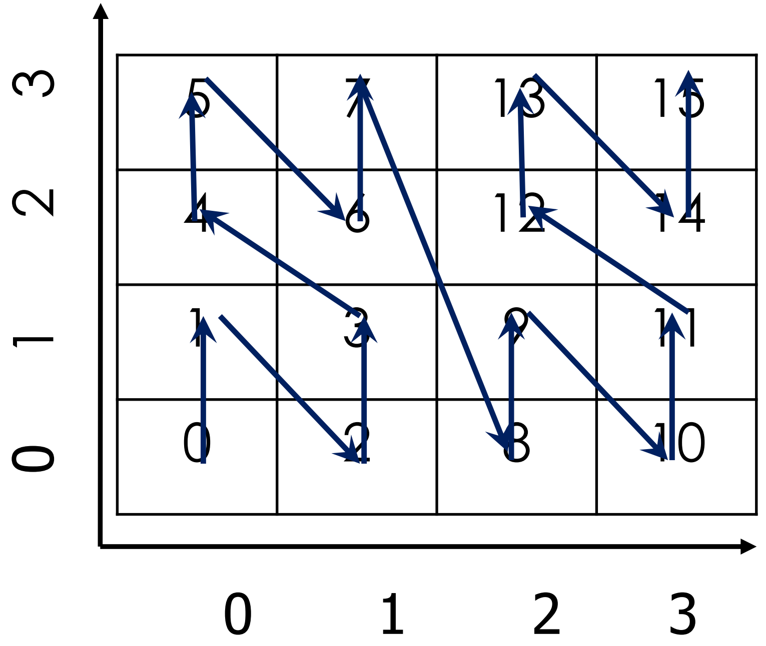


y \ x	0	1	2	3
3	5	7	13	15
2	4	6	12	14
1	1	3	9	11
0	0	2	8	10

This is the order of cells from this process

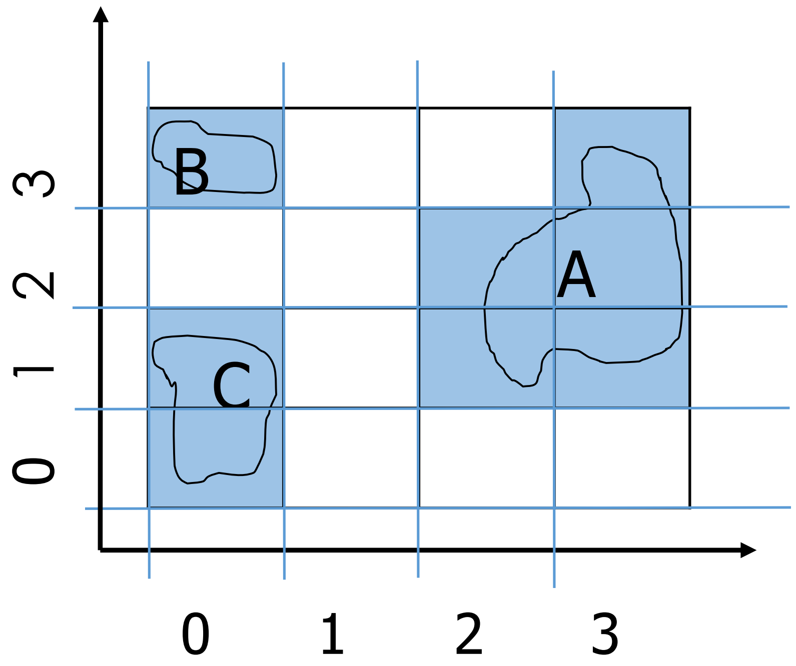
Z-Order curve

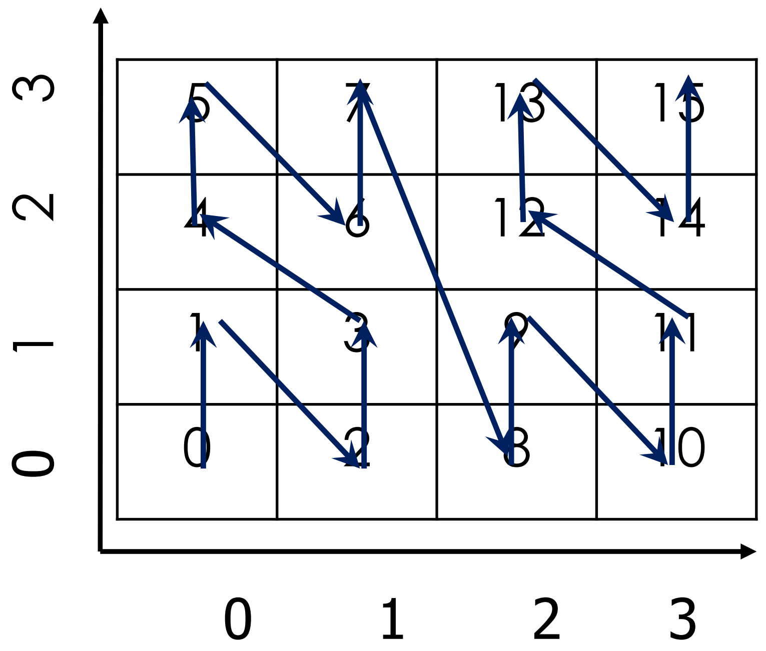




This is the order of cells from this process

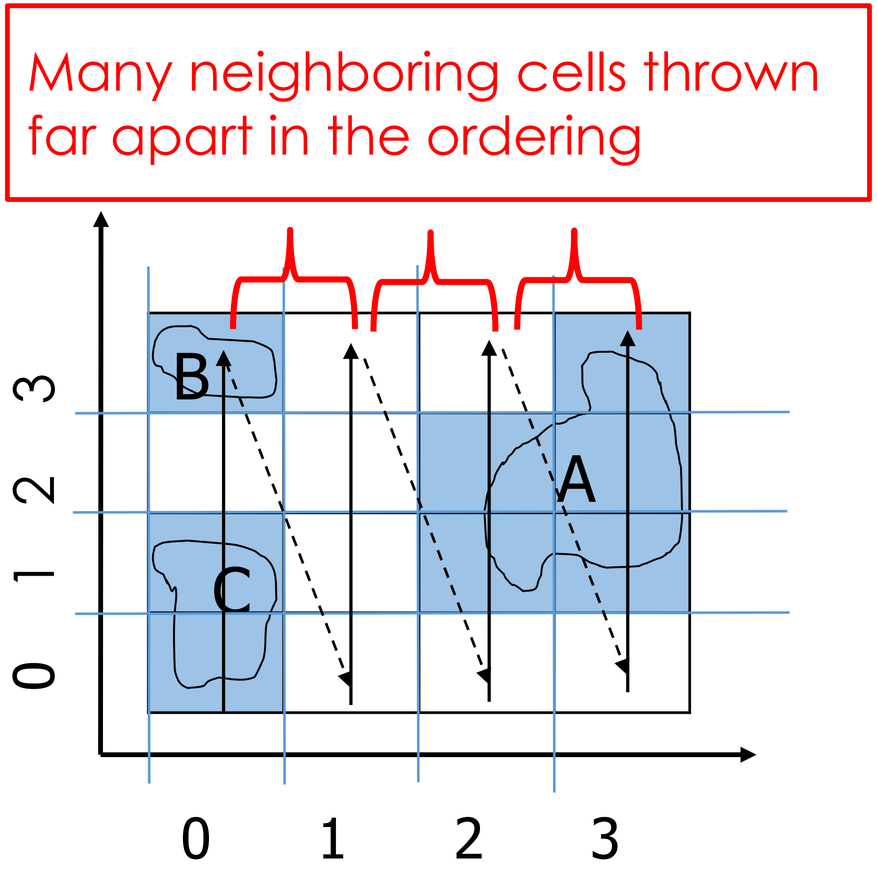
Z-Order curve



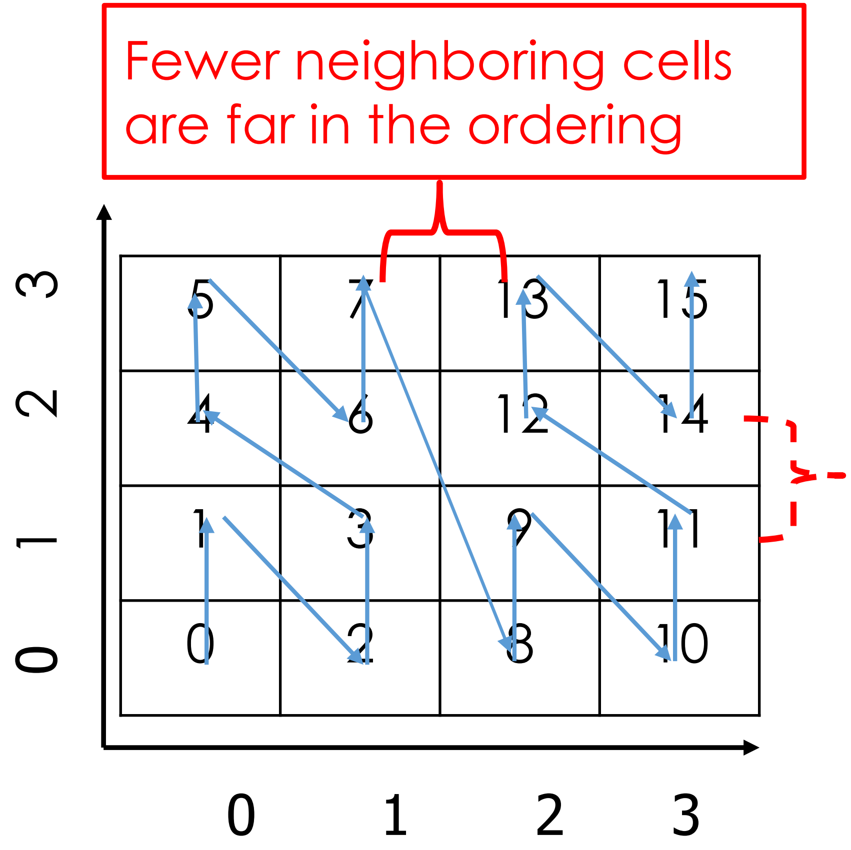


Visually its looks like we have Zs on our map.
Hence the name Z-order curve!!

Z-Order curve

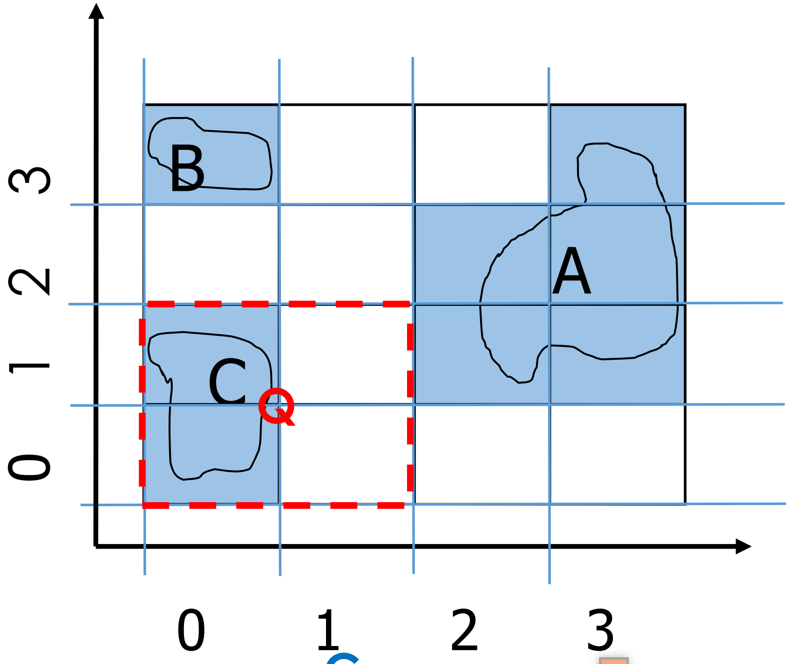


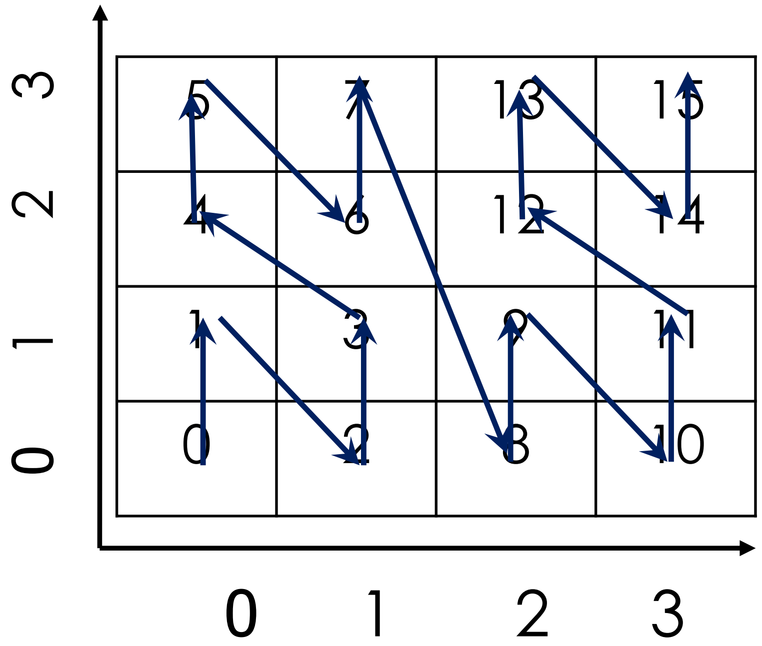
Ordering:
X followed by Y



Z-ordering

Z-Order curve: Range Query

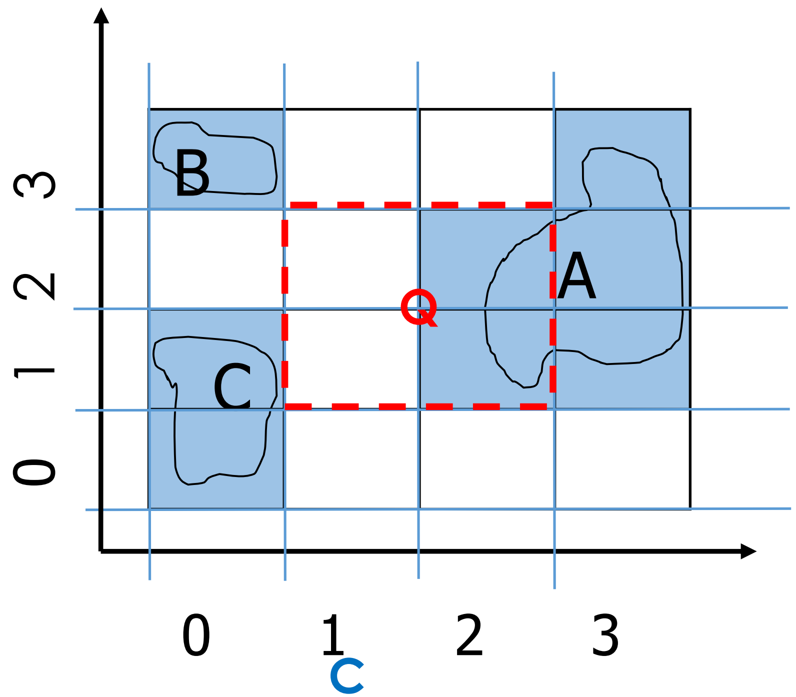


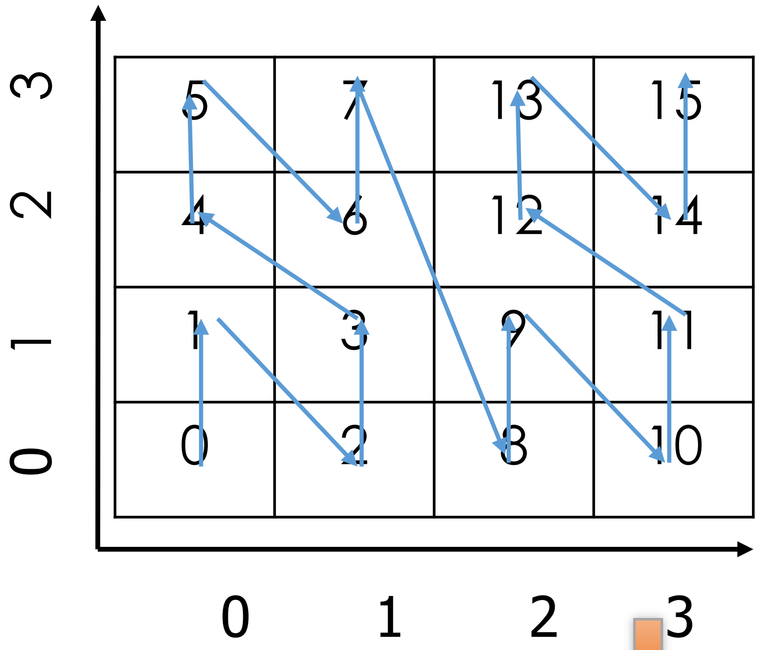


- **Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)**

C — (0,0) (0,1); B — (0,3); A — (2,1); A — (3,1) (2,2); A — (3,2) (3,3)

Z-Order curve: Range Query





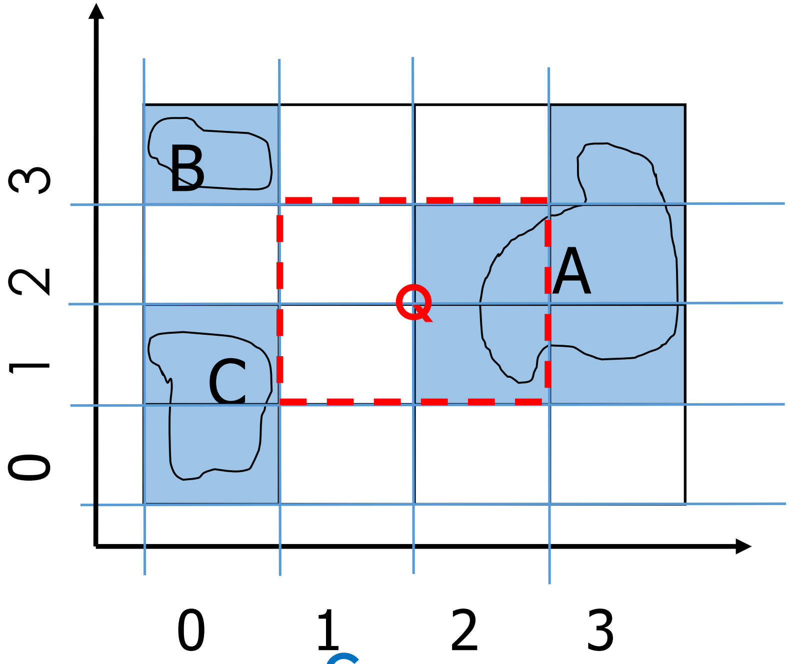
C

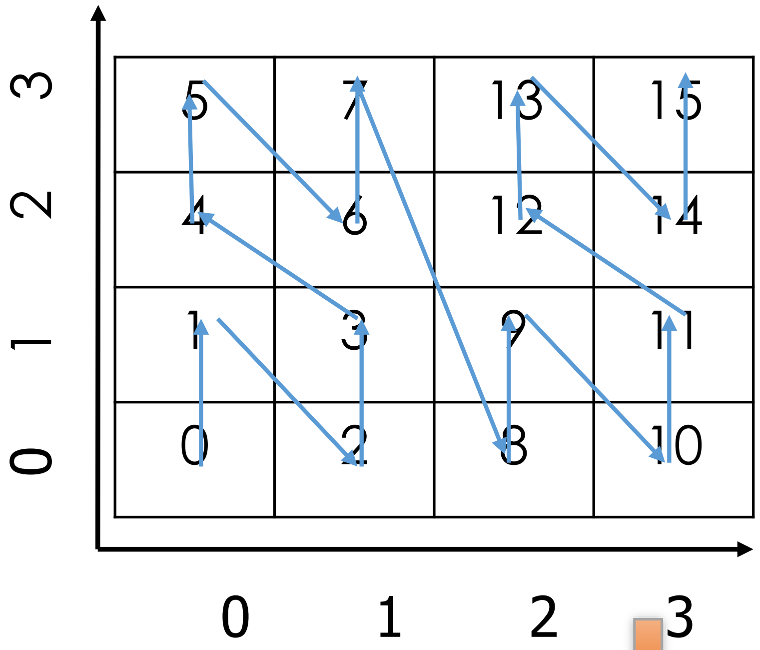
- **Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)**

B A A A

Will this Approach of executing Range Query always give correct answer??

Z-Order curve: Range Query





C

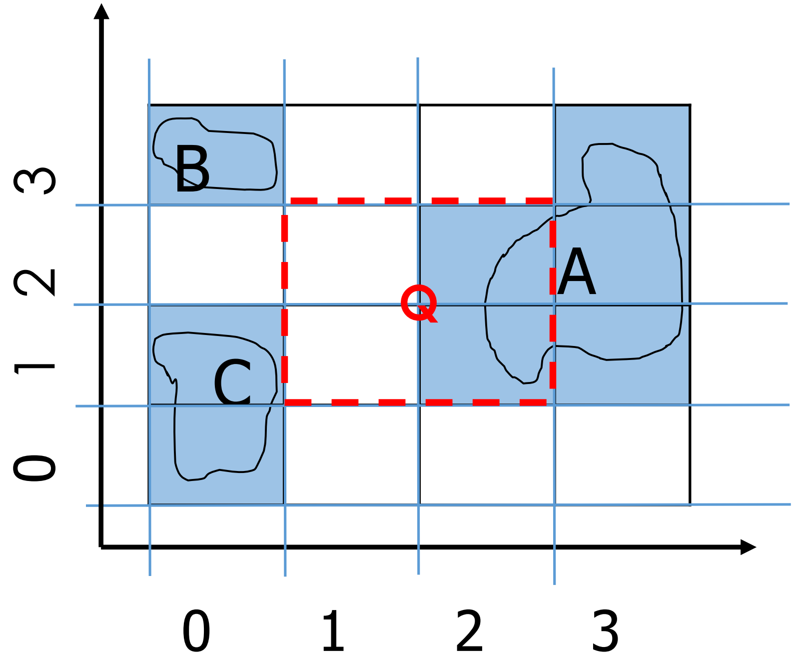
- **Z-order**: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)

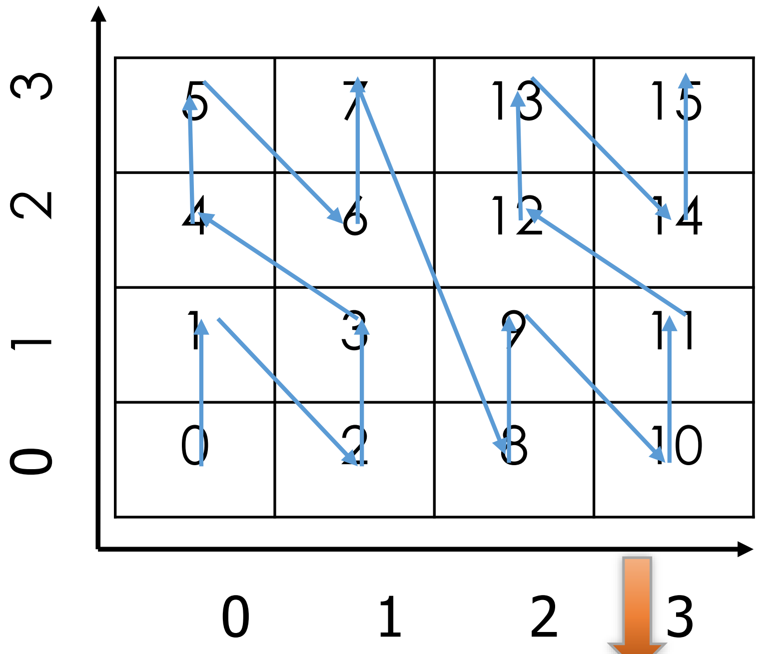
B A A A

It may also include some objects which are not part of answer. Need a second step to clean those out.

Correctness of Range Query on Z-Order curves

- Consider again our previous example:



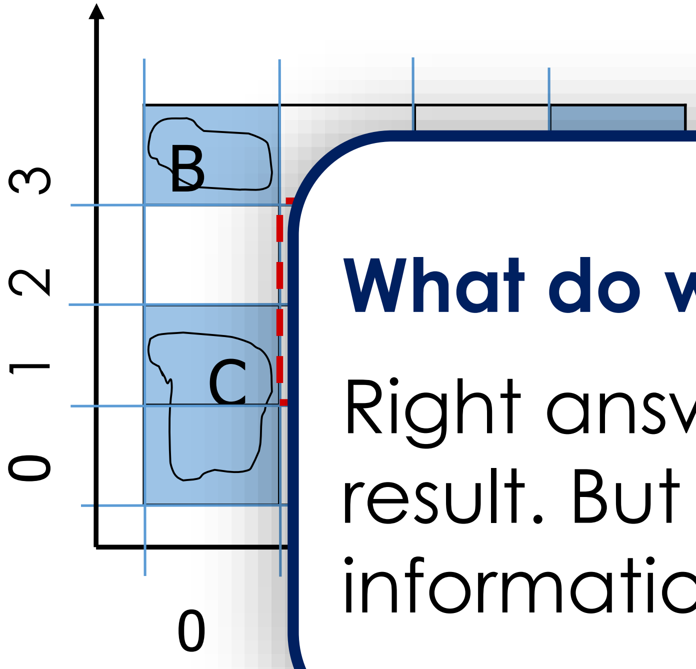


- **Z-order:** (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)

- **Retrieved all records within this range and cross checked the result.**

Correctness of Range Query on Z-Order curves

- Consider again our previous example:



What do we mean by Correctness?

Right answer would certainly be in the result. But it may contain some other information also which need to cleaned

- **Z-order:** (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)

- **Retrieved all records within this range and cross checked the result.**

Correctness of Range Query on Z-Order curves

Proof Sketch:

- **Z-order**: (0,0) (0,1) **(1,0) (1,1) (0,2)** (0,3) **(1,2) (1,3) (2,0)** **(2,1)** **(3,0)** **(3,1)** **(2,2)** **(2,3)** **(3,2) (3,3)**
- Retrieved all records within this range and cross checked the result.
- For this approach to be correct we need to prove that all the cells which are in the query rectangle of (1,1) and (2,2) are between 4 and 9.

Correctness of Range Query on Z-Order curves

<u>Proof Sketch:</u>

- **Without loss of generalization let:**
  - LL = (xmin, ymin) is the lower left of the query rectangle
  - UR = (xmax, ymax) is the upper right of the query rectangle.
  - Then we **need to prove** that all the cells with **(xmin < x < xmax) and (ymin < y < ymax)** will have their **Z-values between z-values of LL and UR.**

Correctness of Range Query on Z-Order curves

<u>Proof Sketch</u>:

- Take two cell coordinates numbers: (x1,y1) and (x2, y2)
- **Case I: x2 > x1 and y1 = y2**
  - If x2 is greater than x1 that it will have "1" in at least one higher position in binary form
  - Which means it will get "1" in at least one higher position in its z-value.
  - Implies that it will have a higher z-value.

Correctness of Range Query on Z-Order curves

Proof Sketch:

- **Case II: y2 > y1 and x1 = x2**
  - If y2 is greater than y1 that it will have "1" in at least one higher position in binary form
  - Which means it will get "1" in at least one higher position in its z-value
  - Implies it will have a higher z-value.

Correctness of Range Query on Z-Order curves

<u>Proof Sketch:</u>

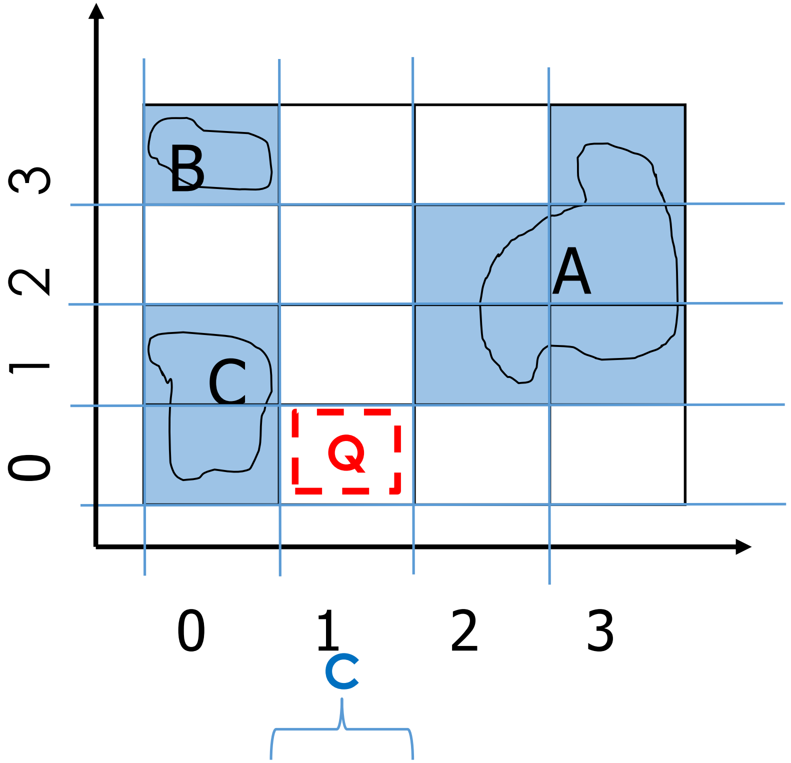
- **Case III: x2 > x1 and y2 > y1**
  - Similar argument of getting "1" in at least one higher position in its z-value
  - Implies it will have a higher z-value

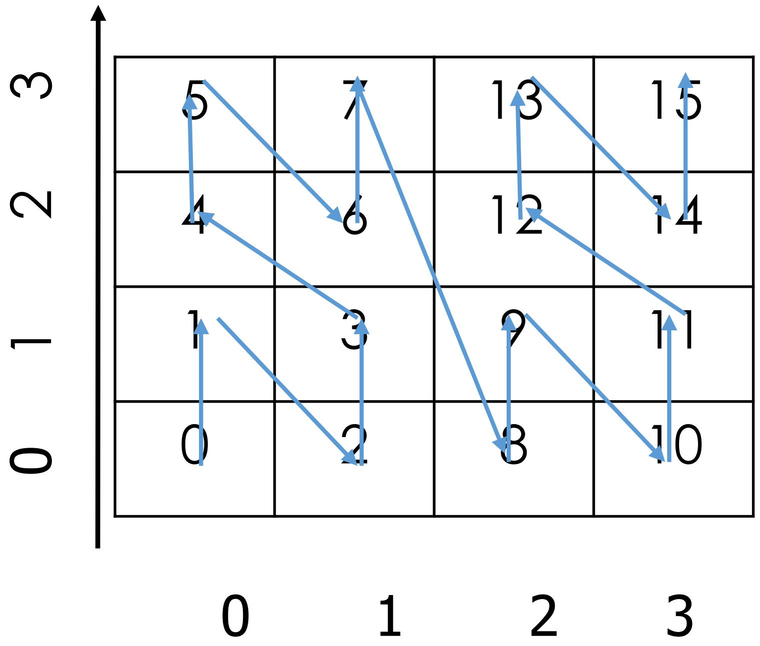
Correctness of Range Query on Z-Order curves

Proof Sketch:

- Now take any cell (x, y) inside the query rectangle defined by LL and UU
- Using our previous argument z-value of (x,y) would be greater than z-value of LL and smaller that z-value of UR
- Basically we switch (x1,y1) and (x2, y2) with (x,y), LL, and UR to make a argument.

Z-Order curve: K-Nearest Neighbor Query



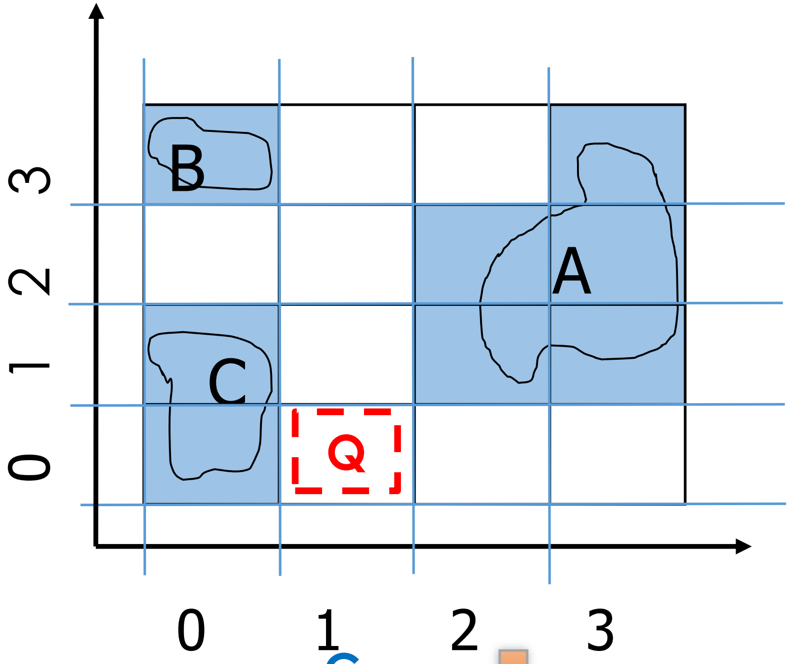


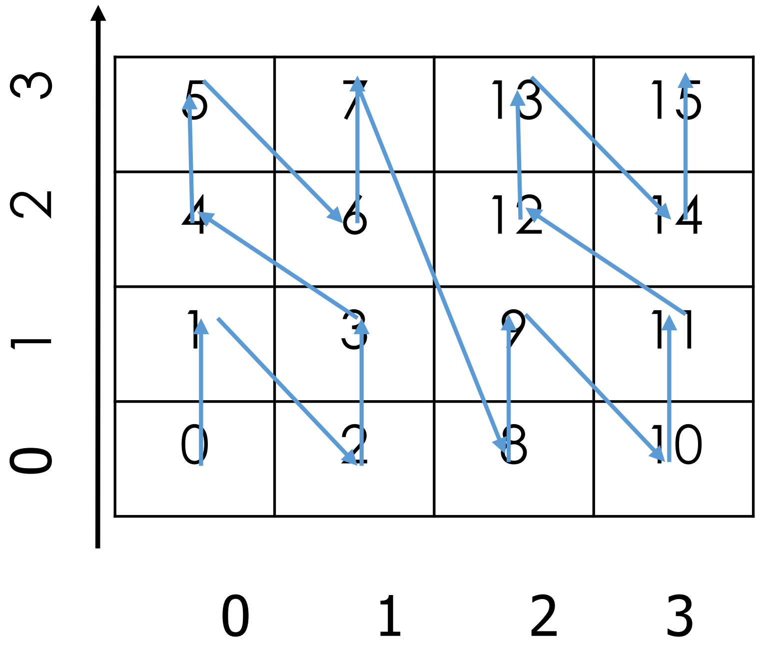
- **Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)**

B A A A

Query: What are the two closest neighbors of query point Q?

Z-Order curve: K-Nearest Neighbor Query



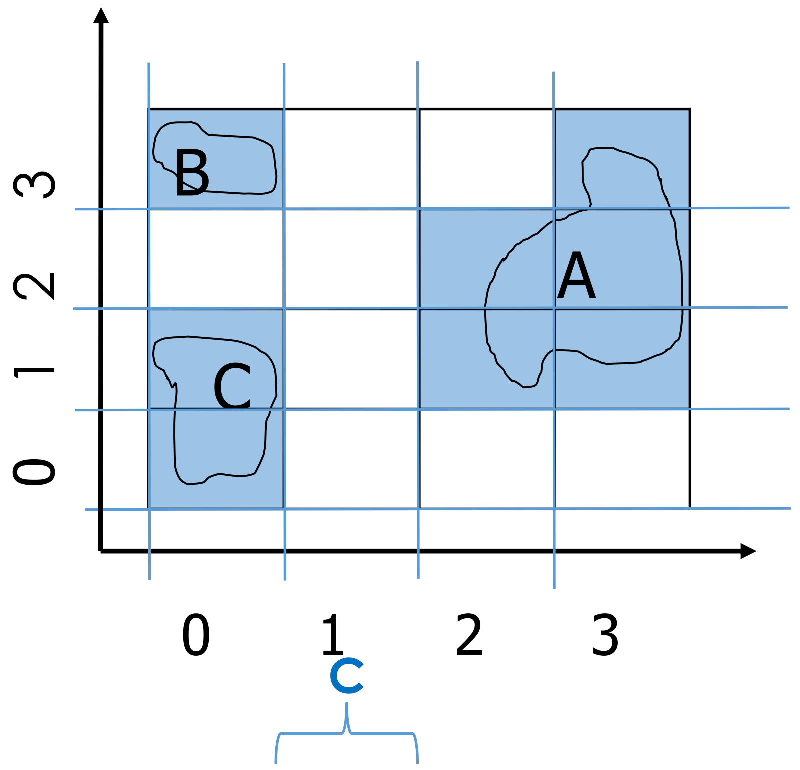


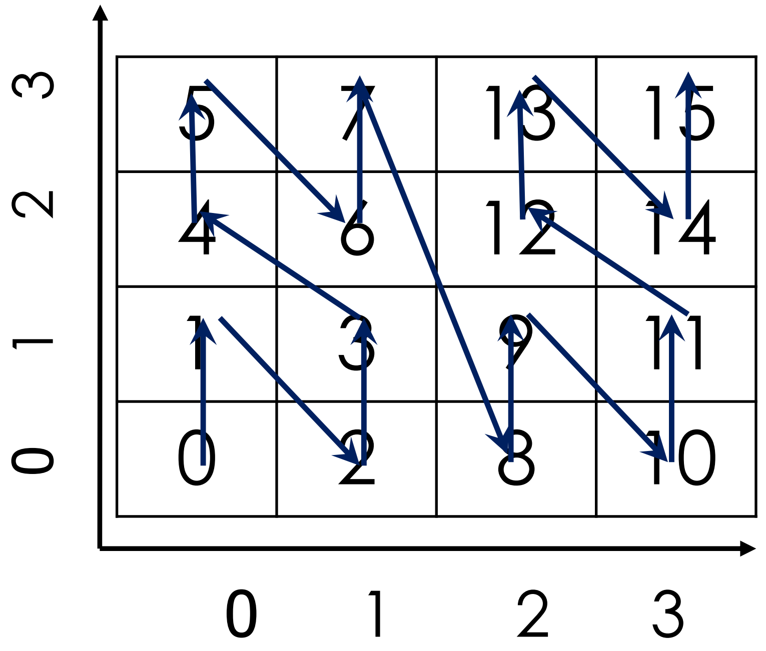
- Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)

C — (0,0) (0,1); Q — (0,1) … (0,3); B — (0,3); A — (2,1); A — (3,1) (2,2); A — (3,2) (3,3)

Nearest to Q we have object B and C in the Z-Order
Relative distances in Z-order don't match up real ones

Z-Order curve: KNN Query for K=1



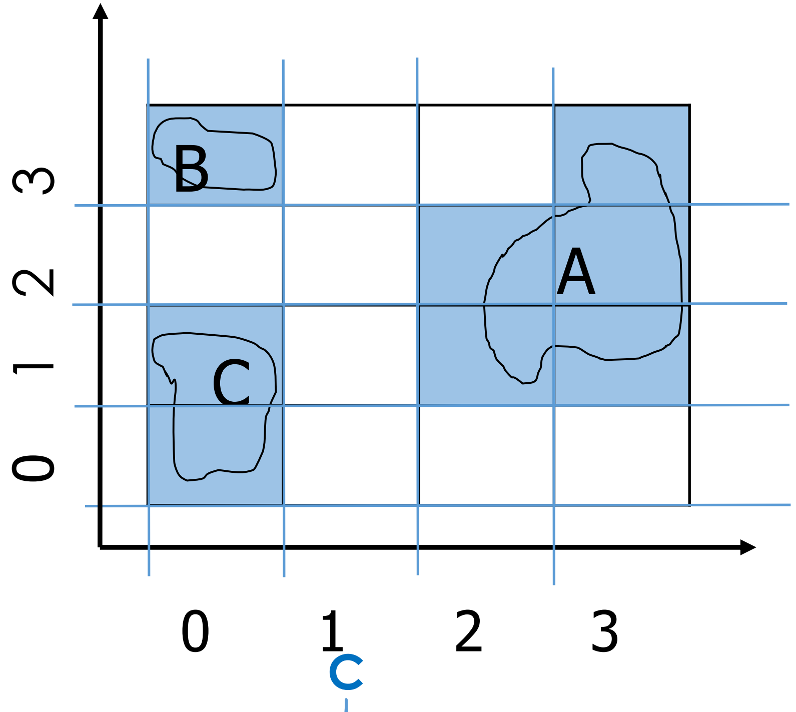


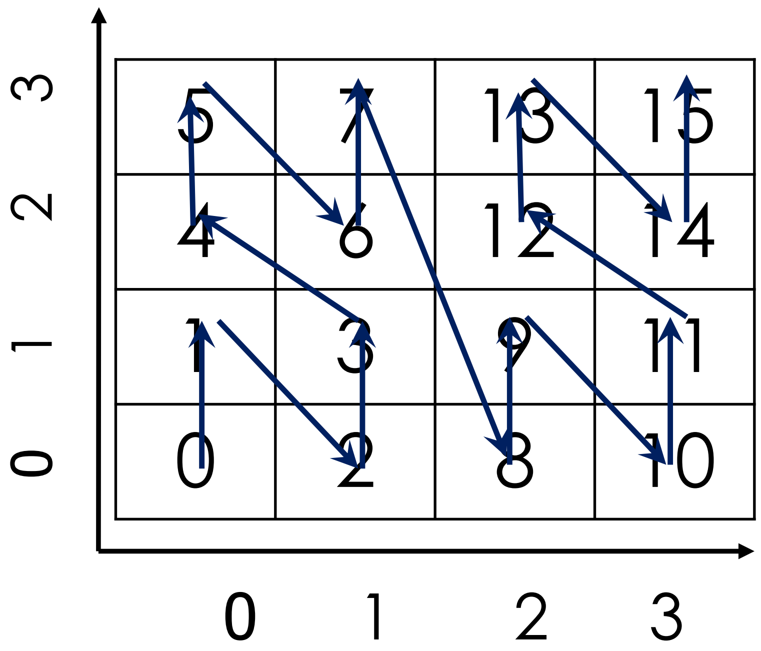
- **Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)**

B A A A

What about 1-nearest neighbor? Any Luck?

Z-Order curve: KNN Query for K=1



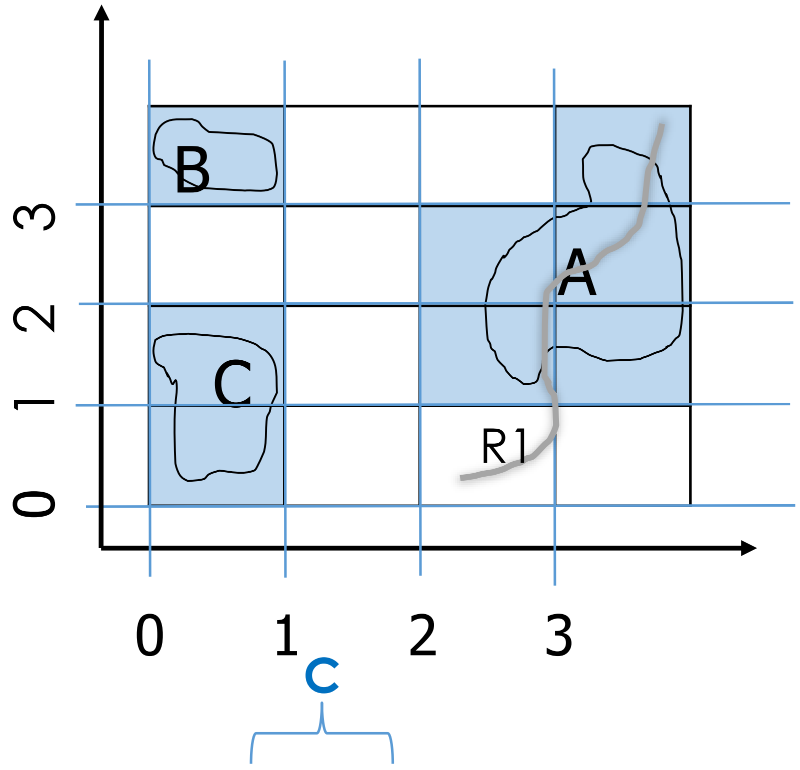


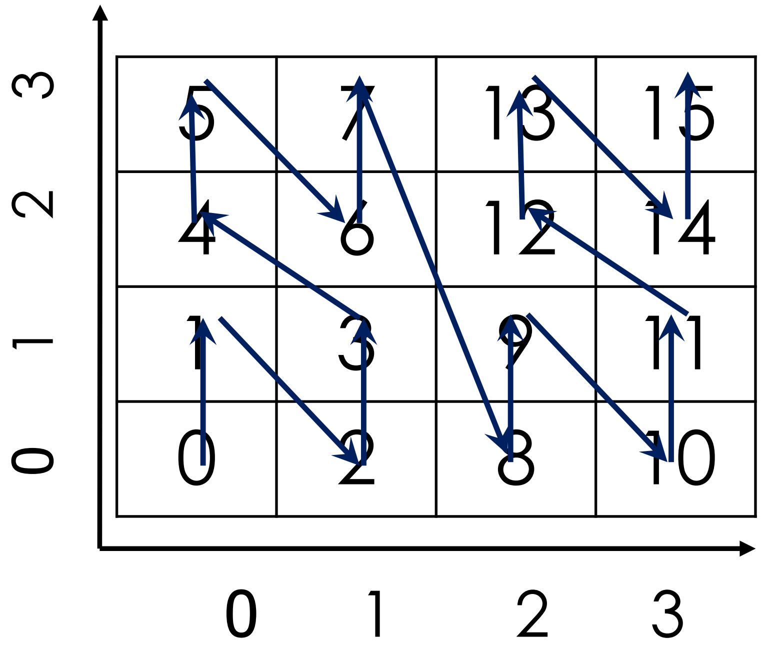
- **Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)**

B A A A

Get the NN from the z-values and issue a range query where range is a circle, query point as the center and radius is the distance to closest Z-value

Z-Order curve: Algorithm for Spatial Join?



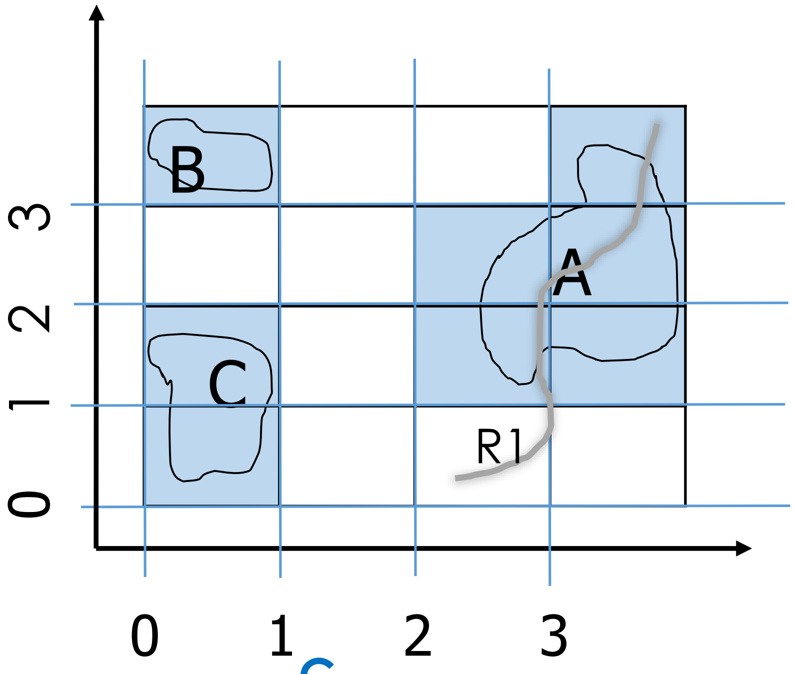


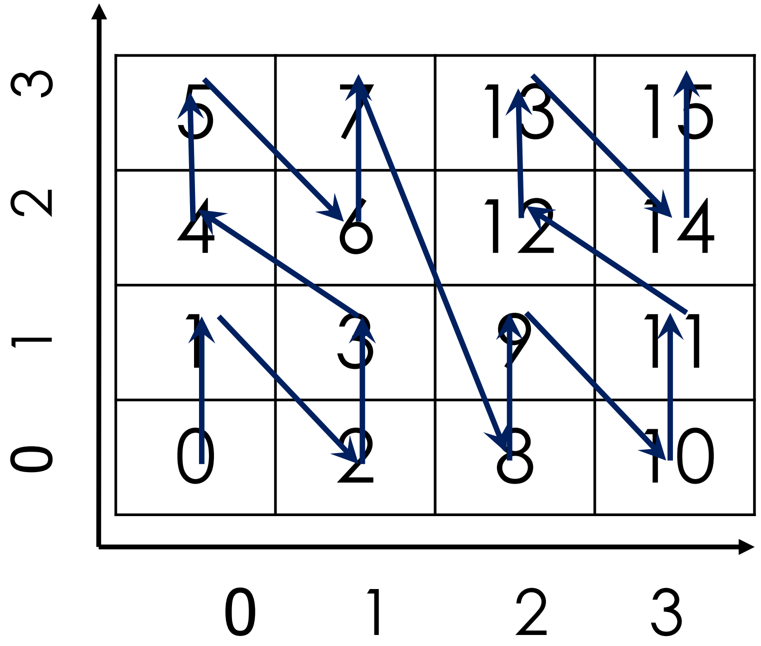
- **Z-order: (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)**

B A A A

Which Spatial Object overlaps with river R1?

Z-Order curve: Algorithm for Spatial Join?





C

- **Z-order:** (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)

B A A A

Sorted Z-order values of R1: (2,0) (2,1) (2,2) (3,2) (3,3)
Can be posed as a range query with end points as (2,0) (3,3)

Z-Curves in larger spaces

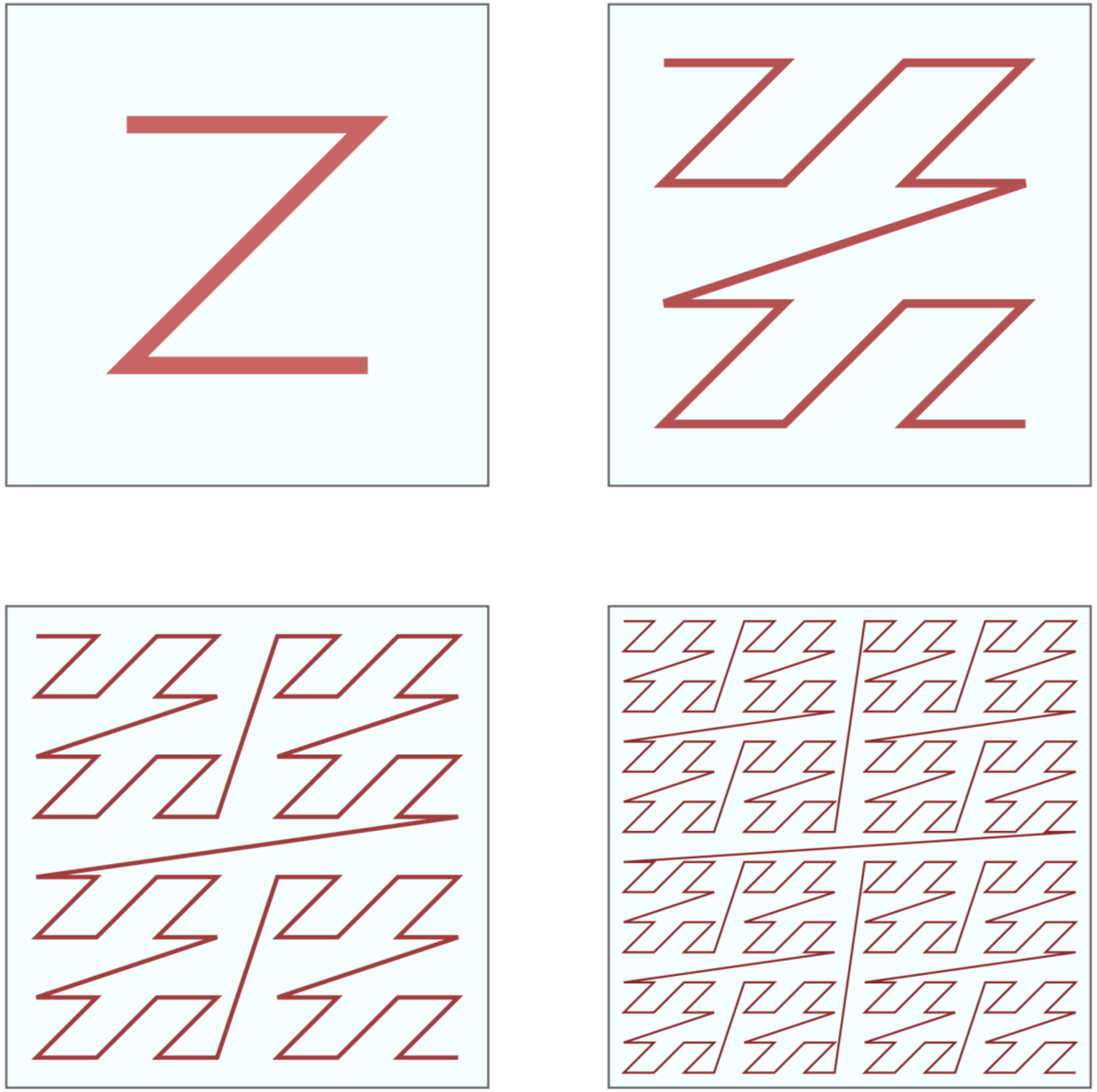


Image source: wikipedia

Analytical Analysis of Z-Order curves

- **Confusion Matrix:**

		True Condition	
		Pos	**Neg**
Predicted Condition	**Pos**	True Positive	False Positive
	Neg	False Negative	True Negative

- **Precision:**

$$\textbf{\textit{Precision}} = \frac{\textbf{\textit{True Positive}}}{\textbf{\textit{True Positive+False Positive}}}$$

- **Recall:**

$$\textbf{\textit{Recall}} = \frac{\textbf{\textit{True Positive}}}{\textbf{\textit{True Positive+False Negative}}}$$

Analytical Analysis of Z-Order curves

- **Confusion Matrix:**

		True Condition: Pos	True Condition: Neg
Predicted Condition	Pos	True Positive	False Positive
	Neg	False Negative	True Negative

Thoughts on Precision and Recall of the initial step of previous range query algorithm?

- **Precision:**

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- **Recall:**

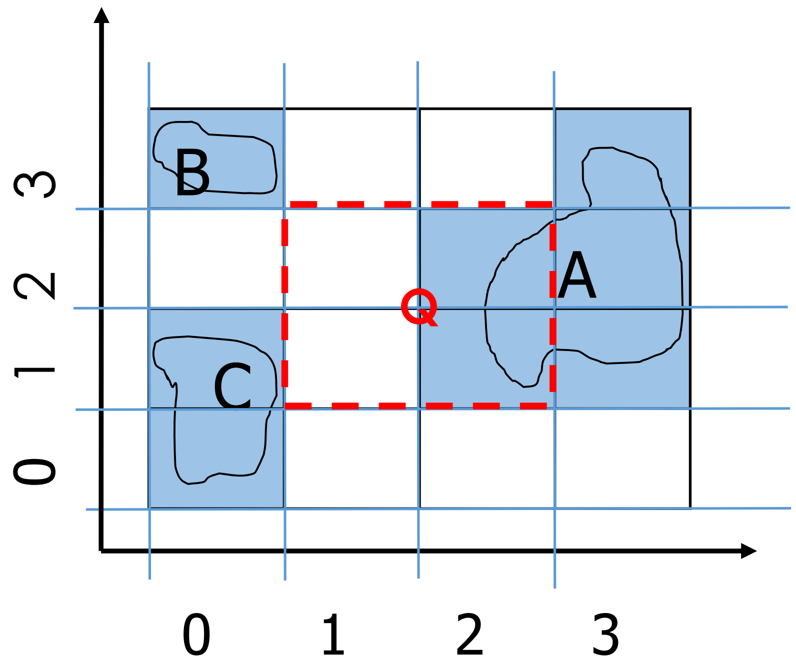
$$Recall = \frac{True\ Positive}{True\ Positive + False\ \ Negative}$$

Analytical Analysis of Z-Order curves

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ \ Negative}$$

Thoughts on Precision and Recall of the first step of the range query algorithm?

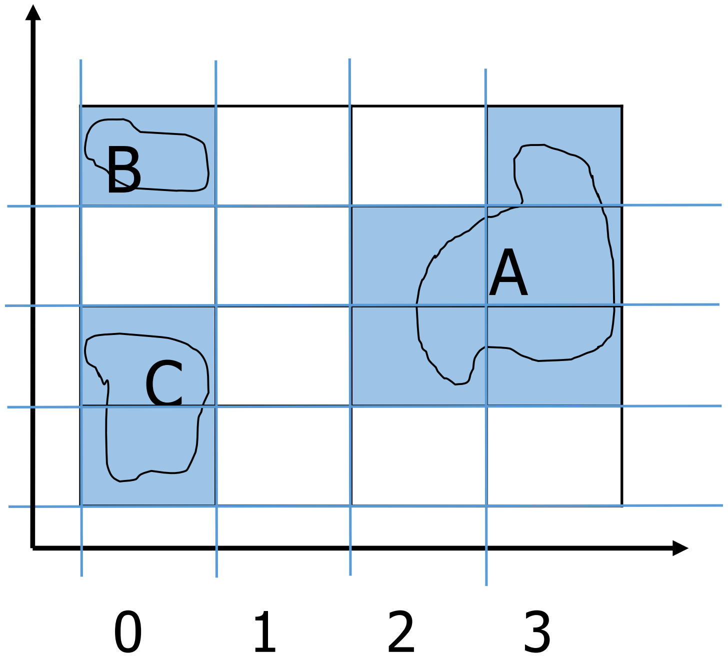


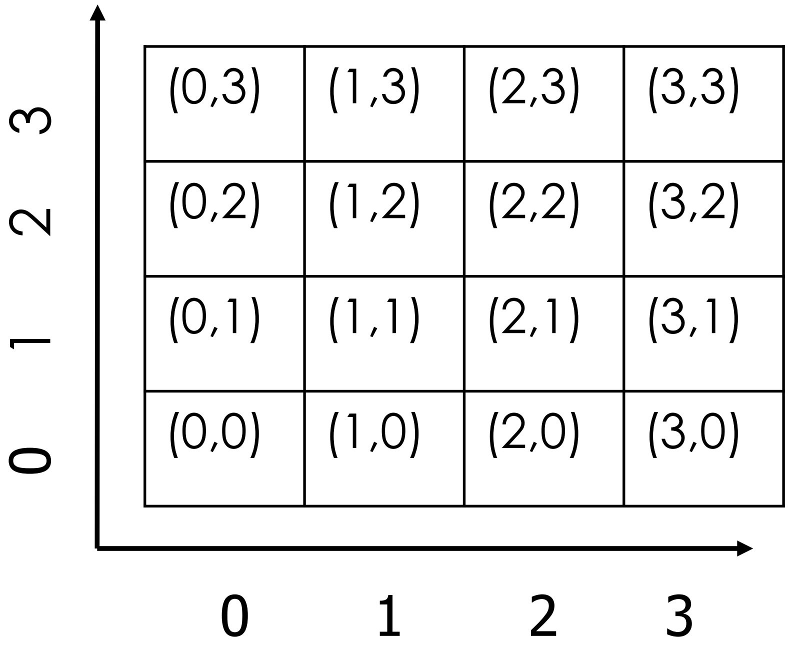
- **Z-order:** (0,0) (0,1) (1,0) (1,1) (0,2) (0,3) (1,2) (1,3) (2,0) (2,1) (3,0) (3,1) (2,2) (2,3) (3,2) (3,3)

Hilbert Curves

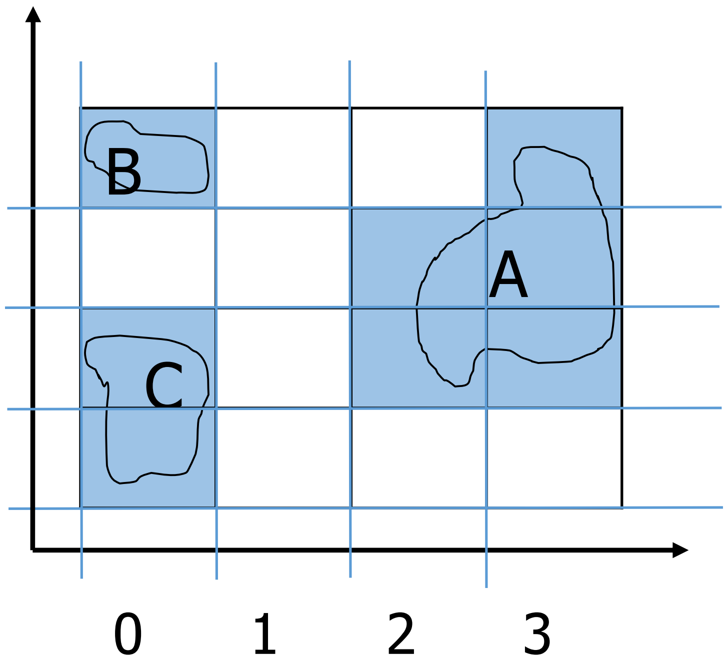
- **Step1:** Read in the n-bit binary representation of the x and y coordinates.
- **Step 2:** Interleave bits of the two binary numbers into one string
- **Step3:** Divide the string into from left to right into 2-bit strings
- **Step4**: Assign decimal values: "00" as 0; "01" as 1; "10" as 3; "11" as 2 and put into an array is the same order as the strings occurred.
- **Step5:** For each number j in the array
  - If j==0 then switch every following occurrence of 1 to 3 and vice-versa
  - If j==3 then switch every following occurrence of 0 to 2 and vice-versa
- **Step6:** Convert each number in the array to its binary representation (2-bit strings), concatenate from left to right and convert to decimal.

Hilbert Curves





Hilbert Curves (Step 1)



	0	1	2	3
3	00 11	01 11	10 11	11 11
2	00 10	01 10	10 10	11 10
1	00 01	01 01	10 01	11 01
0	00 00	01 00	10 00	11 00

Write the X and Y coordinates in Binary Form

Hilbert Curves (Step 2)

	0	1	2	3
3	00 11	01 11	10 11	11 11
2	00 10	01 10	10 10	11 10
1	00 01	01 01	10 01	11 01
0	00 00	01 00	10 00	11 00

	0	1	2	3
3	0101	0111	1101	1111
2	0100	0110	1100	1110
1	0001	0011	1001	1011
0	0000	0010	1000	1010

Interleave them to create one string

Hilbert Curves (Step 3)

	0	1	2	3
3	0101	0111	1101	1111
2	0100	0110	1100	1110
1	0001	0011	1001	1011
0	0000	0010	1000	1010

→

	0	1	2	3
3	01 01	01 11	11 01	11 11
2	01 00	01 10	11 00	11 10
1	00 01	00 11	10 01	10 11
0	00 00	00 10	10 00	10 10

Divide the string into from left to right into 2-bit strings

Hilbert Curves (Step 4)

	0	1	2	3
3	01 01	01 11	11 01	11 11
2	01 00	01 10	11 00	11 10
1	00 01	00 11	10 01	10 11
0	00 00	00 10	10 00	10 10

	0	1	2	3
3	11	12	21	22
2	10	13	20	23
1	01	02	31	32
0	00	03	30	33

Assign decimal values: "00" as 0; "01" as 1; "10" as 3; "11" as 2

Hilbert Curves (Step 5)

3	11	12	21	22
2	10	13	20	23
1	01	02	31	32
0	00	03	30	33
	0	1	2	3

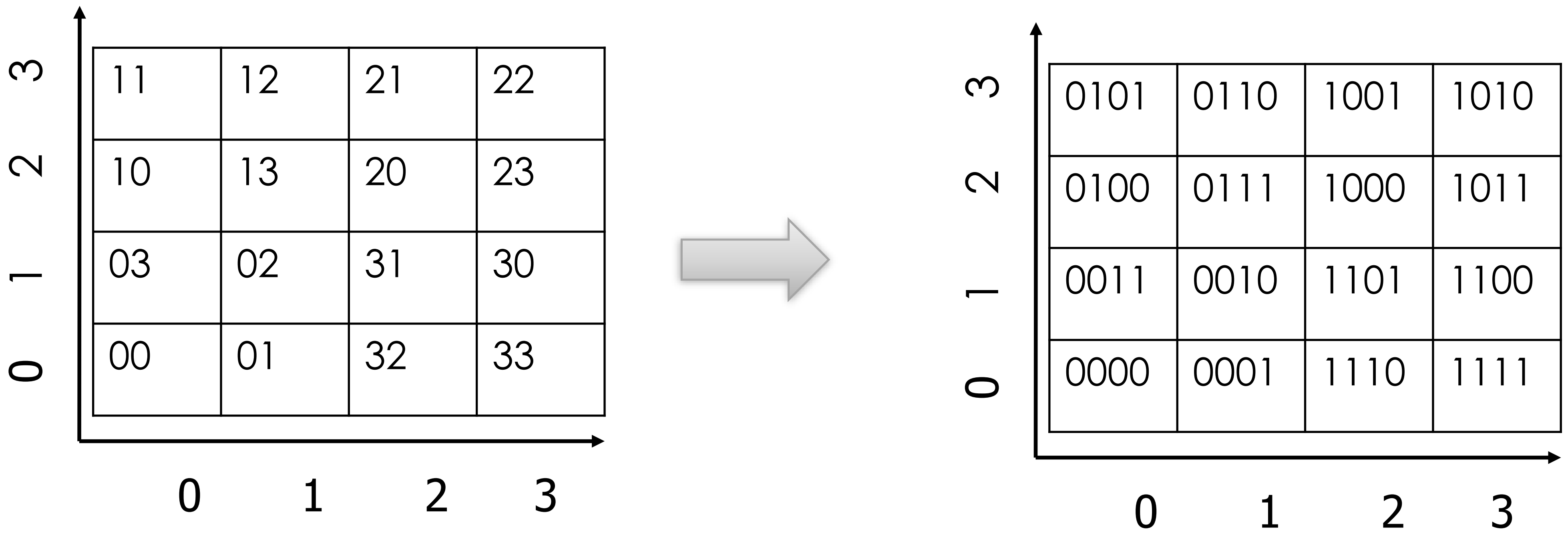
→

3	11	12	21	22
2	10	13	20	23
1	03	02	31	30
0	00	01	32	33
	0	1	2	3

If j==0 then switch every following occurrence of 1 to 3 and vice-versa

If j==3 then switch every following occurrence of 0 to 2 and vice-versa

Hilbert Curves (Step 6)



Concatenate and
Convert to Binary

Hilbert Curves (Step 6)

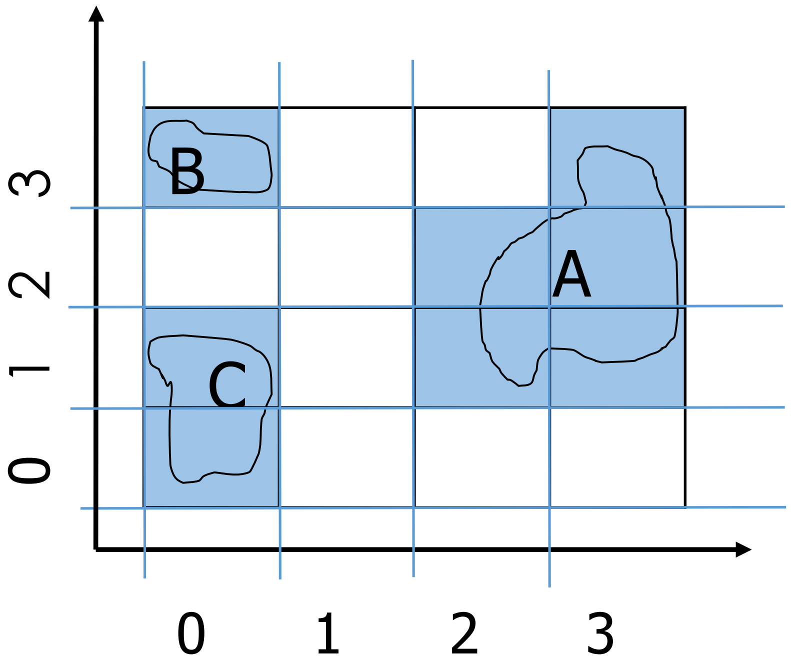
	0	1	2	3
3	0101	0110	1001	1010
2	0100	0111	1000	1011
1	0011	0010	1101	1100
0	0000	0001	1110	1111

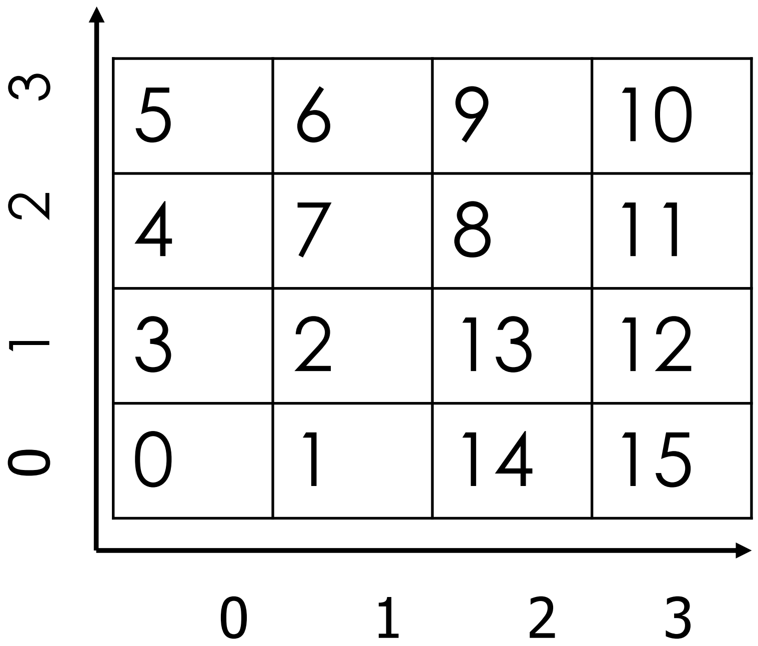
→

	0	1	2	3
3	5	6	9	10
2	4	7	8	11
1	3	2	13	12
0	0	1	14	15

Concatenate and
Convert to Binary

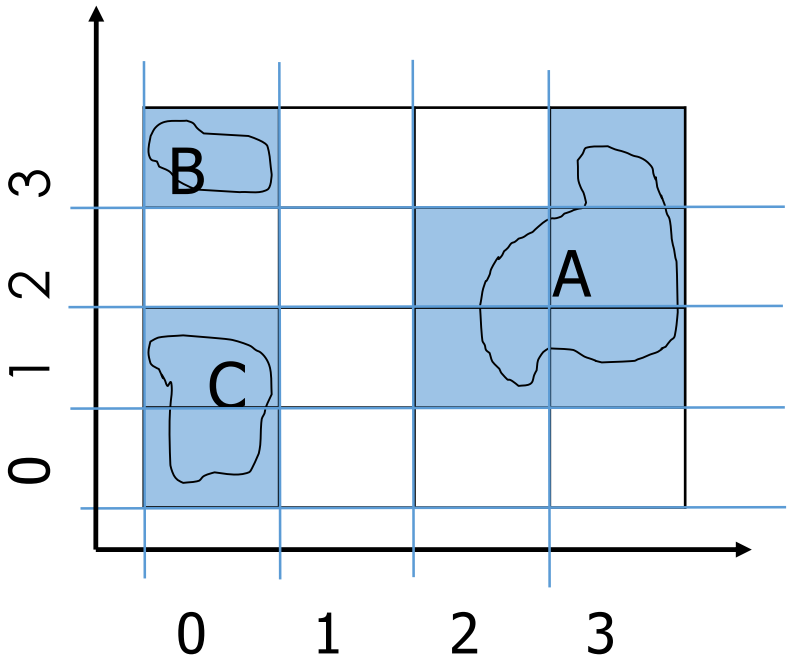
Hilbert Curves (Step 6)

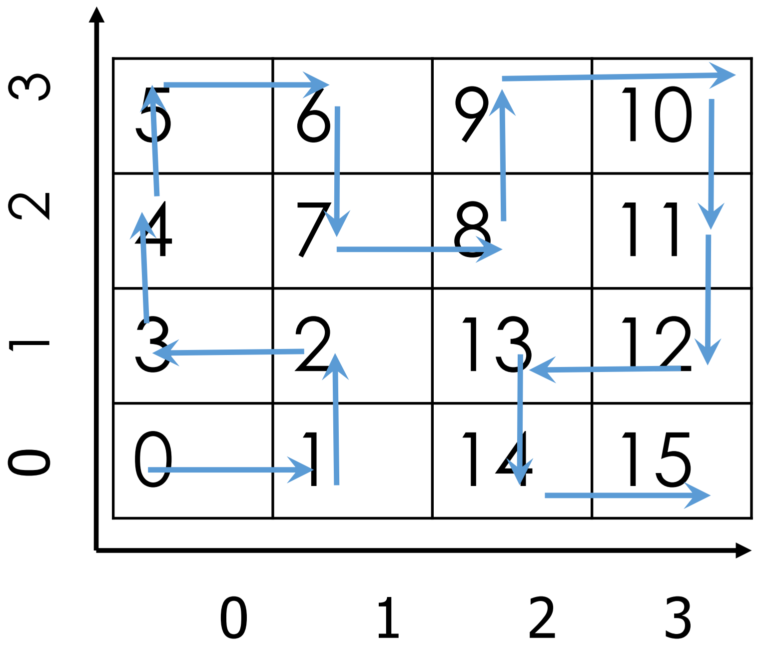




- **Hilbert-curve: (0,0) (1,0) (1,1) (0,1) (0,2) (0,3) (1,3) (1,2) (2,2) (2,3) (3,3) (3,2) (3,1) (2,1) (2,0) (3,0)**

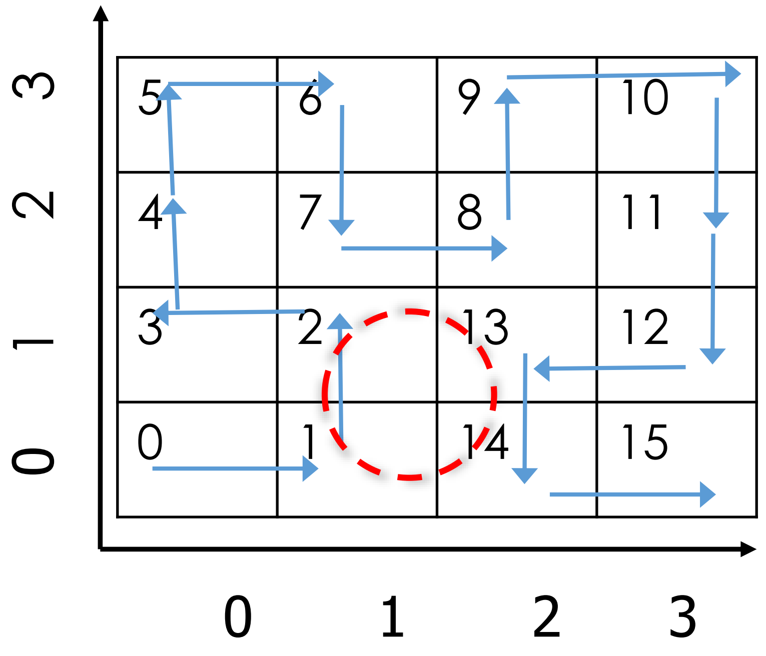
Hilbert Curves (Step 6)



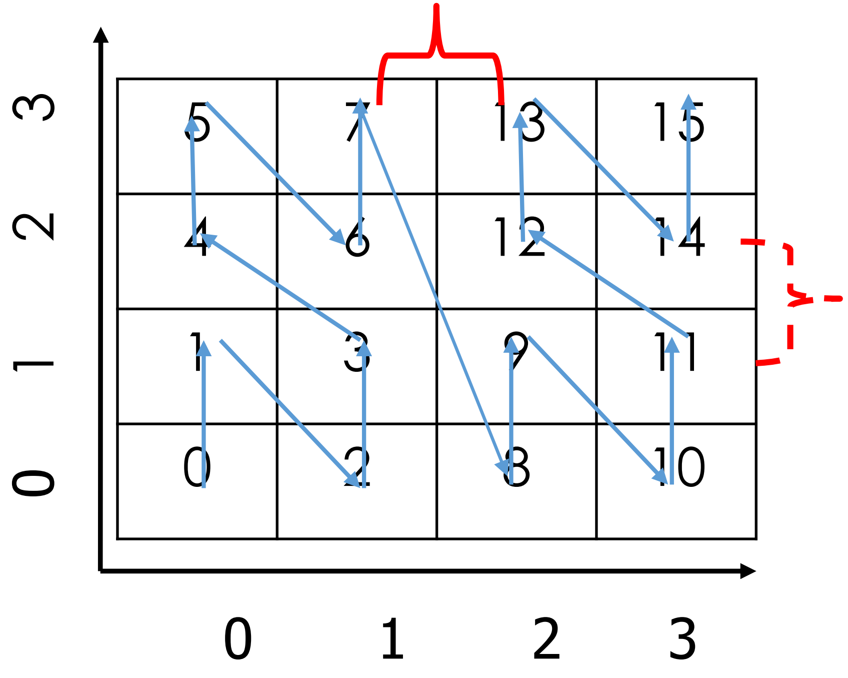


- **Hilbert-curve**: (0,0) (1,0) (1,1) (0,1) (0,2) (0,3) (1,3) (1,2) (2,2) (2,3) (3,3) (3,2) (3,1) (2,1) (2,0) (3,0)

Hilbert Curves Vs Z-Curves

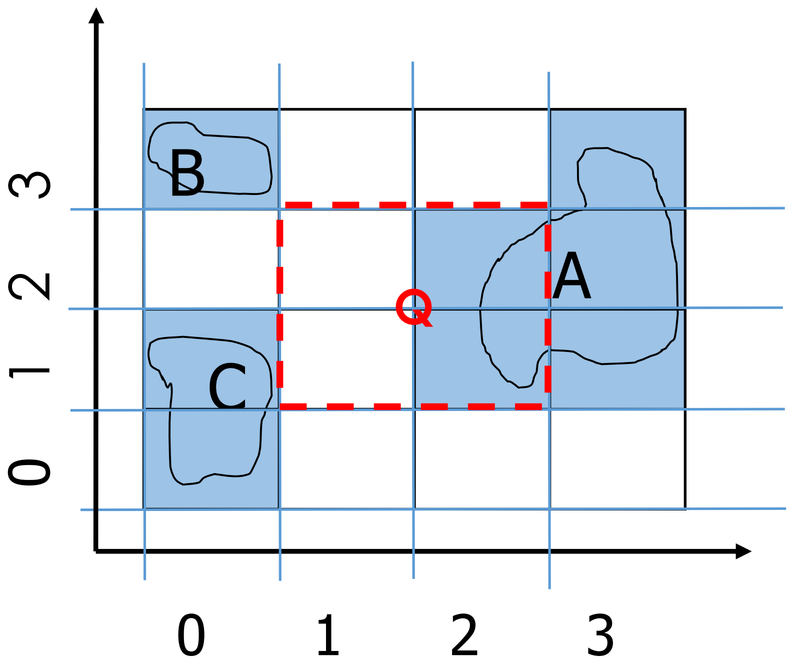


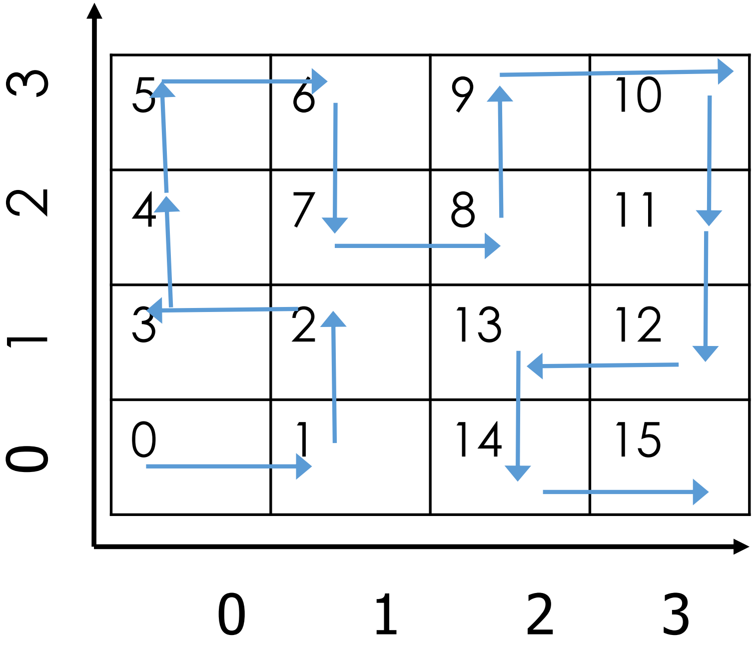
Hilbert Curve



Z-ordering

Hilbert- curve: Range Query





C

- **Hilbert-curver: (0,0) (1,0) (1,1) (0,1) (0,2) (0,3) (1,3) (1,2) (2,2) (2,3) (3,3) (3,2) (3,1) (2,1) (2,0) (3,0)**

B A A

Need to pic min and max Hilbert curve values for this range !

Hilbert Curves in larger spaces

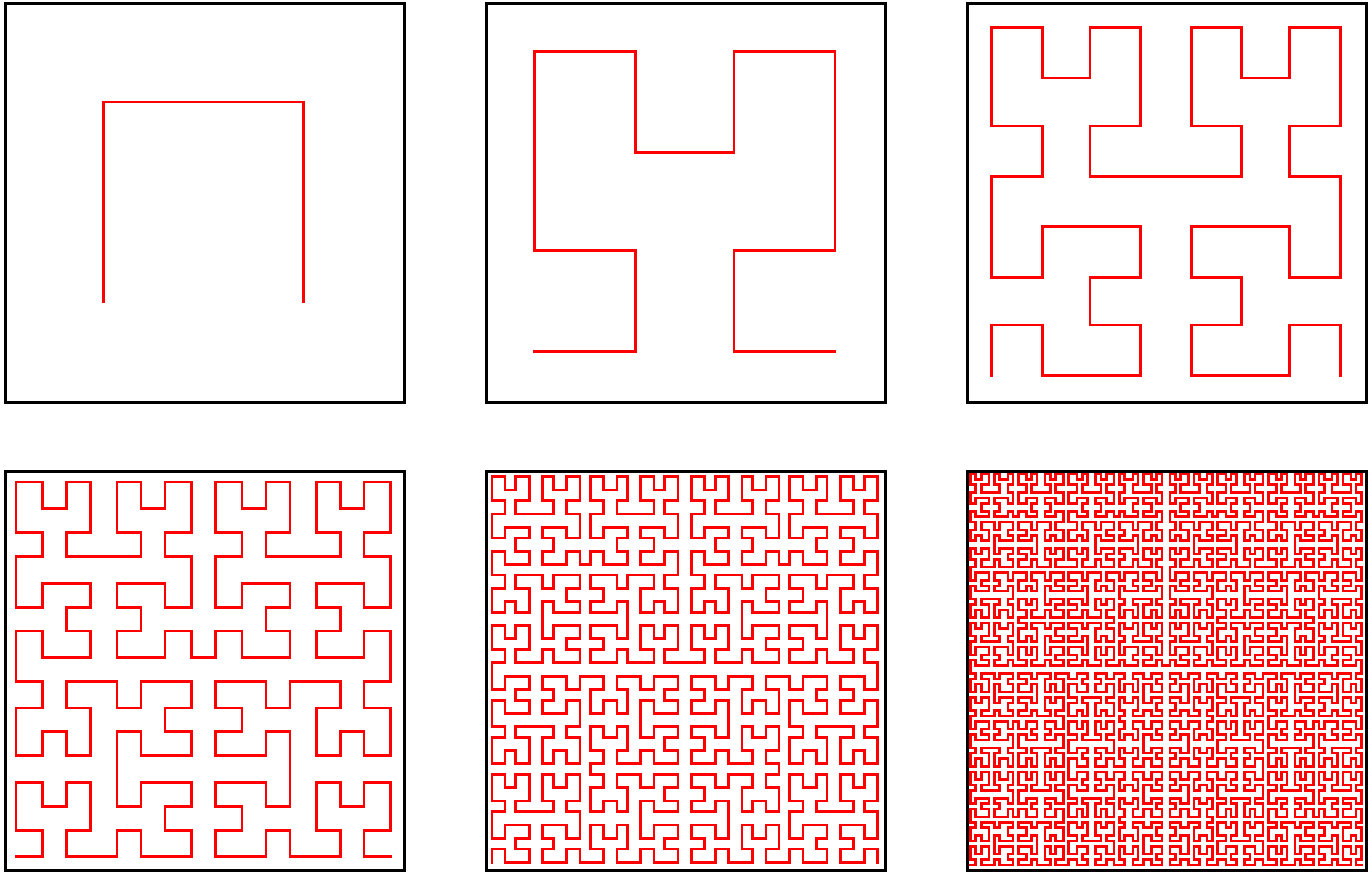


Image source and more details at: http://www.bic.mni.mcgill.ca/~mallar/CS-644B/hilbert.html

Contemplating the Hilbert Curve Algo

- **Step1:** Read in the n-bit binary representation of the x and y coordinates.
- **Step 2:** Interleave bits of the two binary numbers into one string
- **Step3:** Divide the string into from left to right into 2-bit strings
- **Step4:** Assign decimal values: “00” as 0; “01” as 1; “10” as 3; “11” as 2 and put into an array is the same order as the strings occurred.
- **Step5:** For each number j in the array
  - If j==0 then switch every following occurrence of 1 to 3 and vice-versa
  - If j==3 then switch every following occurrence of 0 to 2 and vice-versa
- **Step6:** Convert each number in the array to its binary representation (2-bit strings), concatenate from left to right and convert to decimal.

Contemplating the Hilbert Curve Algo

	0	1	2	3
3	01 01	01 11	11 01	11 11
2	01 00	01 10	11 00	11 10
1	00 01	00 11	10 01	10 11
0	00 00	00 10	10 00	10 10

→

	0	1	2	3
3	11	12	21	22
2	10	13	20	23
1	01	02	31	32
0	00	03	30	33

Output after Step 4

Contemplating the Hilbert Curve Algo

Say we Skip Step 5 and Jump to step 6

3	11	12	21	22
2	10	13	20	23
1	01	02	31	32
0	00	03	30	33
	0	1	2	3

3	01 01	01 10	10 01	10 10
2	01 00	01 11	10 00	10 11
1	00 01	00 10	11 01	11 10
0	00 00	00 11	11 00	11 11
	0	1	2	3

Step 6: We convert nums to binary, concatenate and then convert to decimal

Contemplating the Hilbert Curve Algo

Say we Skip Step 5 and Jump to step 6

y \ x	0	1	2	3
3	01 01	01 10	10 01	10 10
2	01 00	01 11	10 00	10 11
1	00 01	00 10	11 01	11 10
0	00 00	00 11	11 00	11 11

→

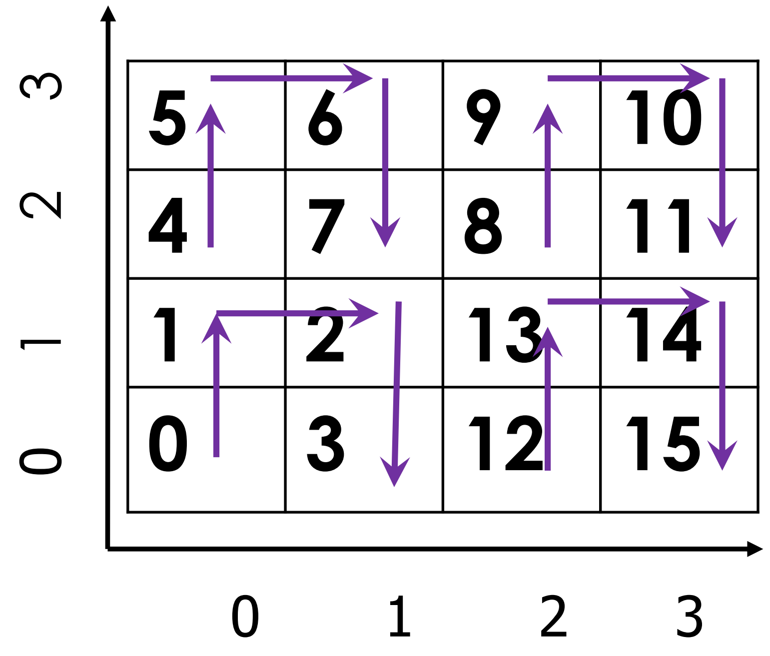
y \ x	0	1	2	3
3	5	6	9	10
2	4	7	8	11
1	1	2	13	14
0	0	3	12	15

Step 6: We convert nums to binary, concatenate and then convert to decimal

Contemplating the Hilbert Curve Algo

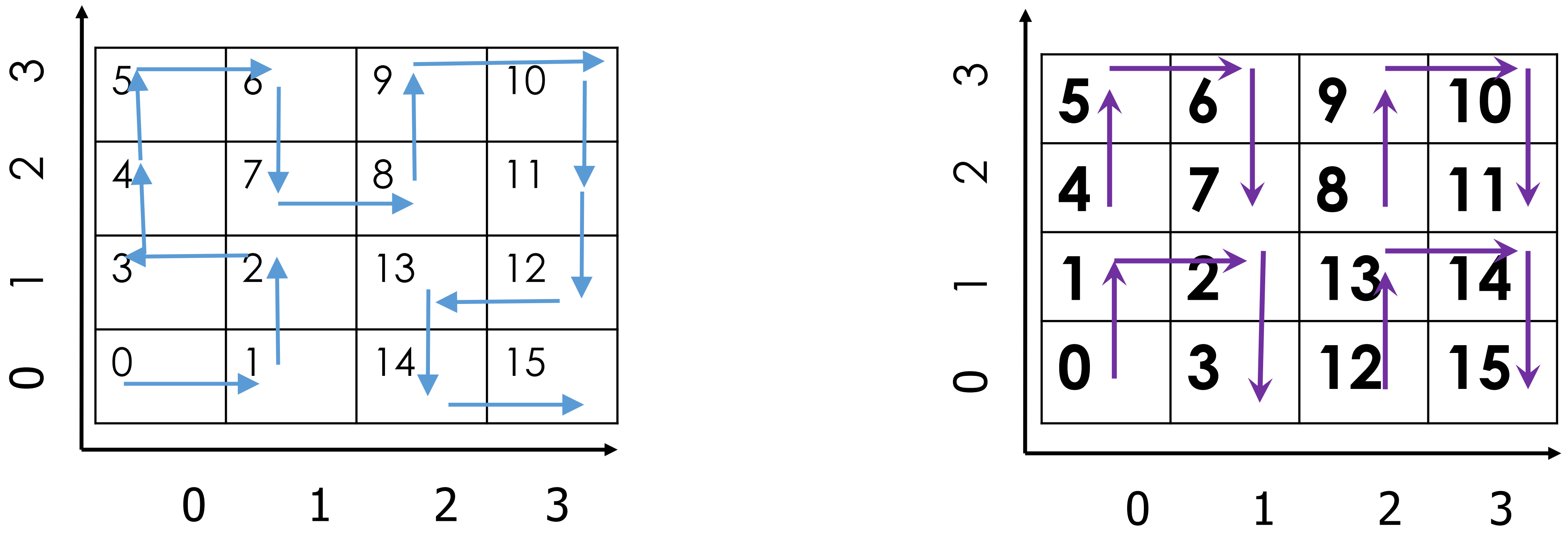
Say we Skip Step 5 and Jump to step 6

	0	1	2	3
3	01 01	01 10	10 01	10 10
2	01 00	01 11	10 00	10 11
1	00 01	00 10	11 01	11 10
0	00 00	00 11	11 00	11 11



Step 6: We convert nums to binary, concatenate and then convert to decimal

Contemplating the Hilbert Curve Algo



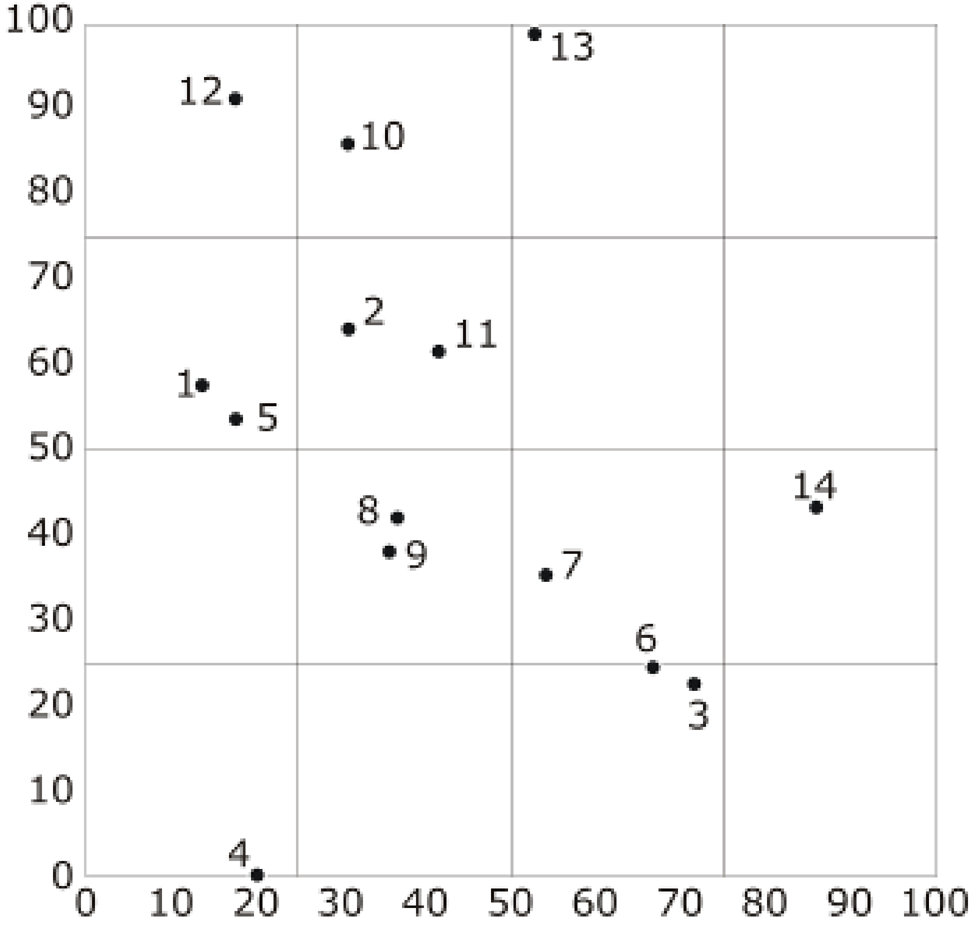
Step 5 seems to be taking care of the rotation and the reflection of the basic shape inverted cup!!!

Addressing challenges of 2-Dimenions more directly

Grid Files

Basic idea- Divide space into cells by a grid

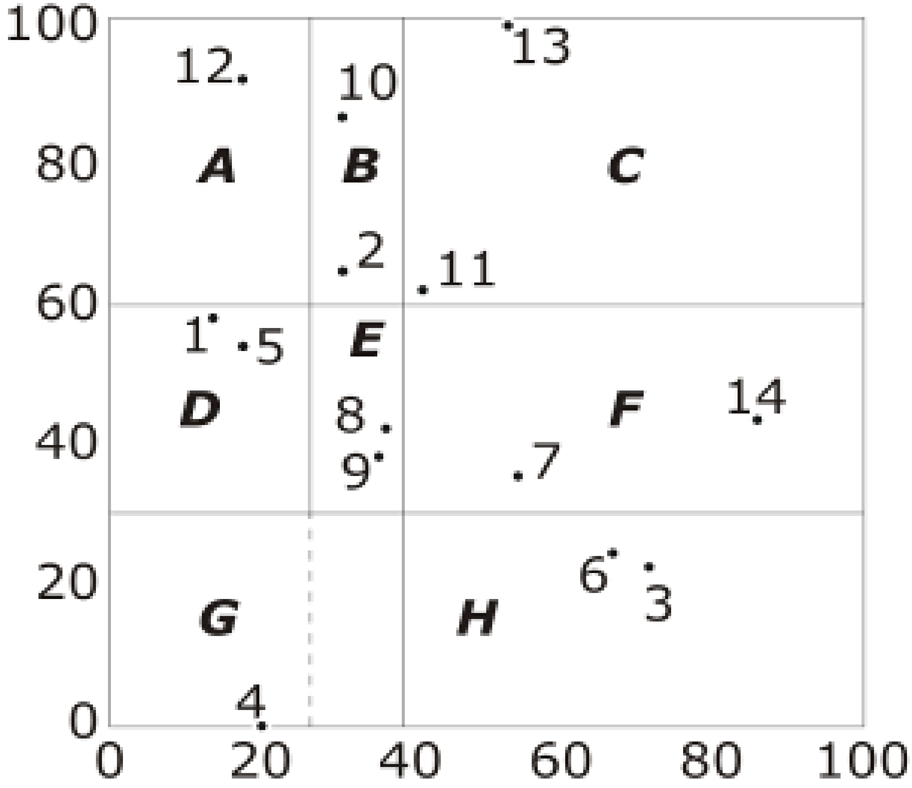
- Store data in each cell in distinct disk page
- A directory structure needed
- Efficient for find, insert, range and nearest neighbor
- May have wastage of disk storage space
- **Non-uniform data distribution over space ??**



Grid Files

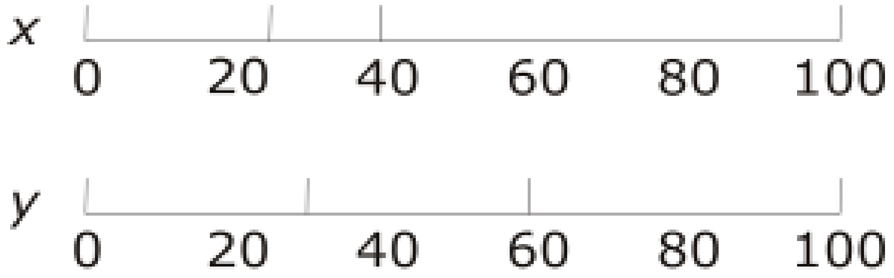
Refinement of basic idea into Grid Files

- Use non-uniform grids
- Linear scale store row and column boundaries
- Allow sharing of disk pages across grid cells



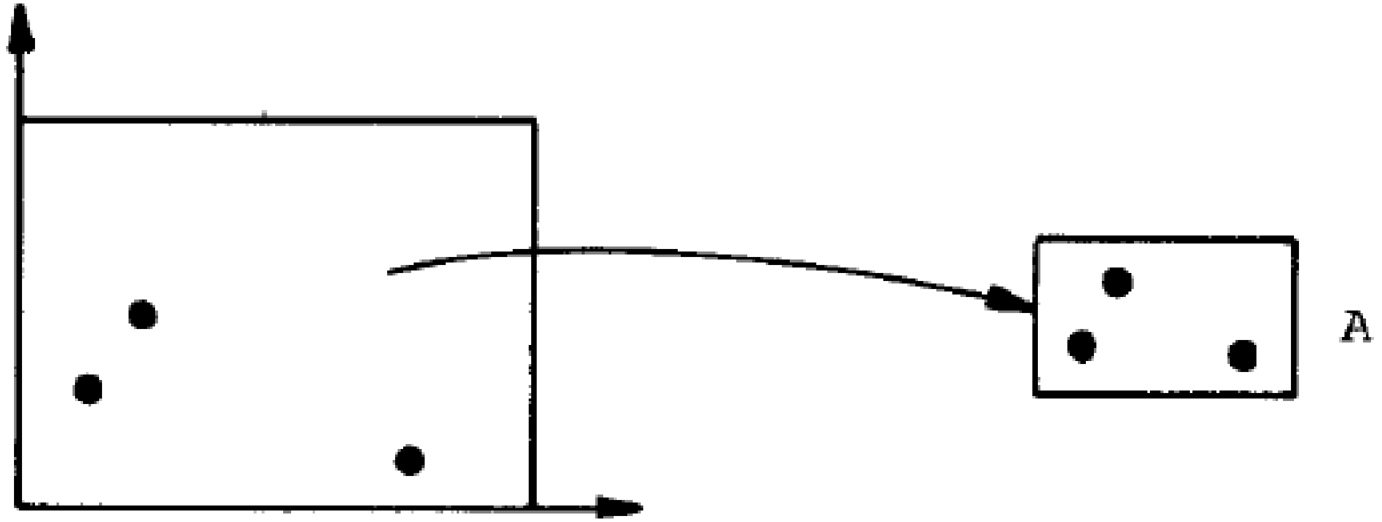
A	B	C
D	E	F
G	G	H

Grid directory



Grid Files (insertion example)

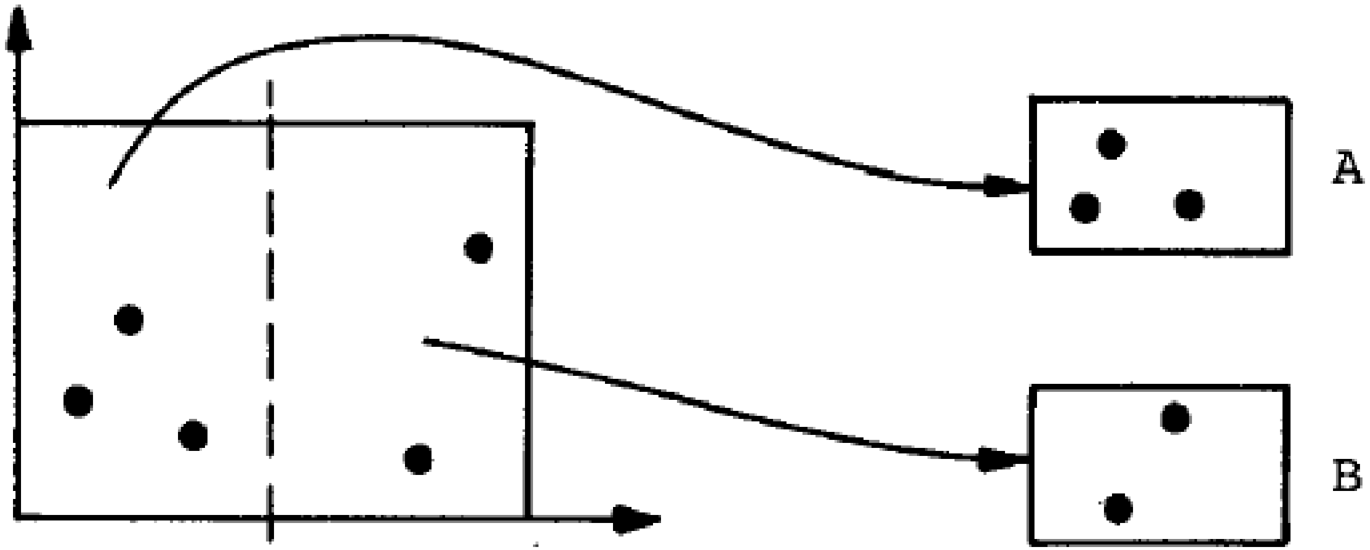
- Capacity of bucket = 3



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

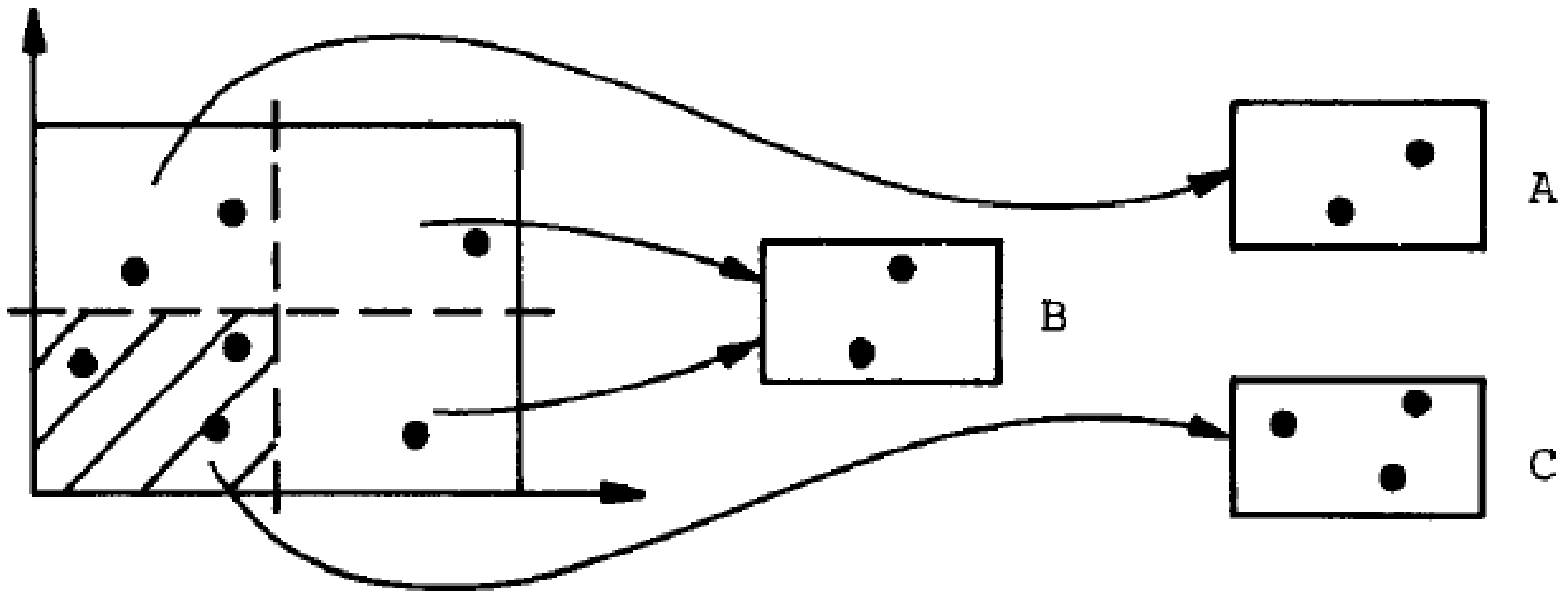
- When the bucket overflows we split it.
- A new bucket is made.
- Records that lie in one half of the space are moved to the new bucket.



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

- Bucket A overflows again.

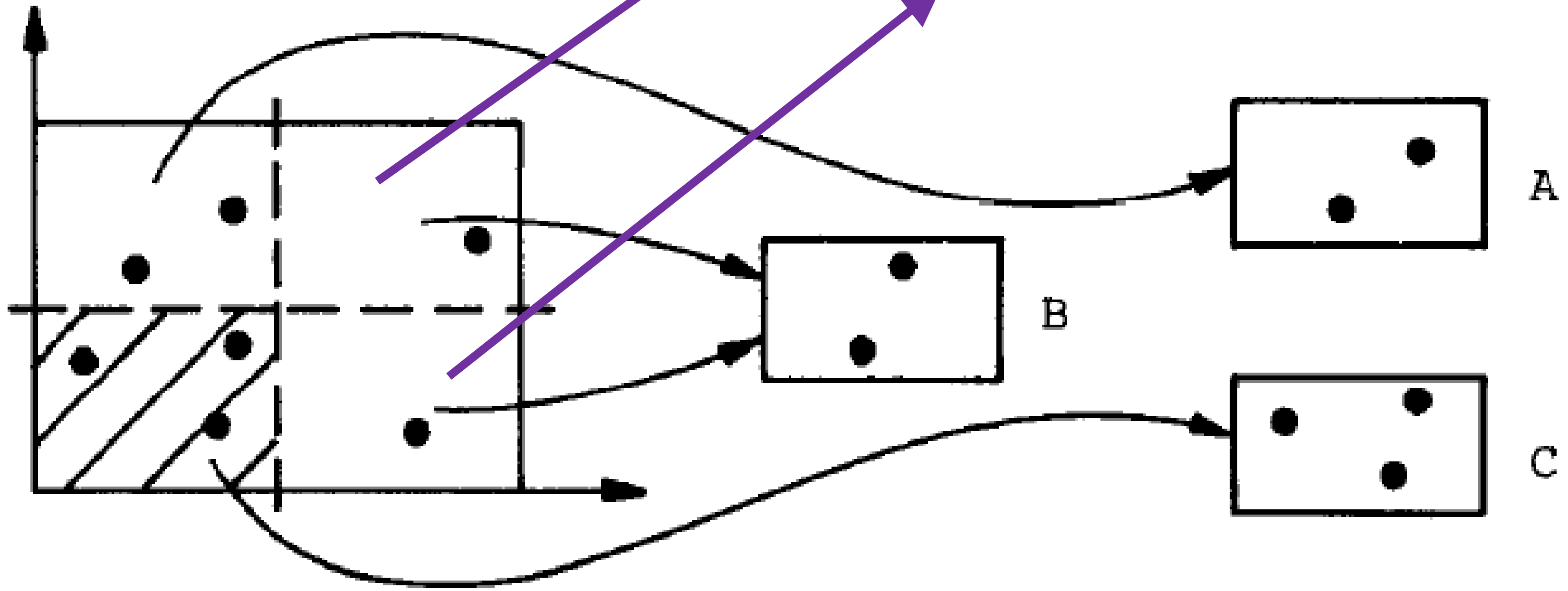


J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

- Bucket A overflows again.

Very Imp: Splitting of A is full horizontal split, i.e., region of B is also split. But B was not overflowing, so both buckets still point to B only

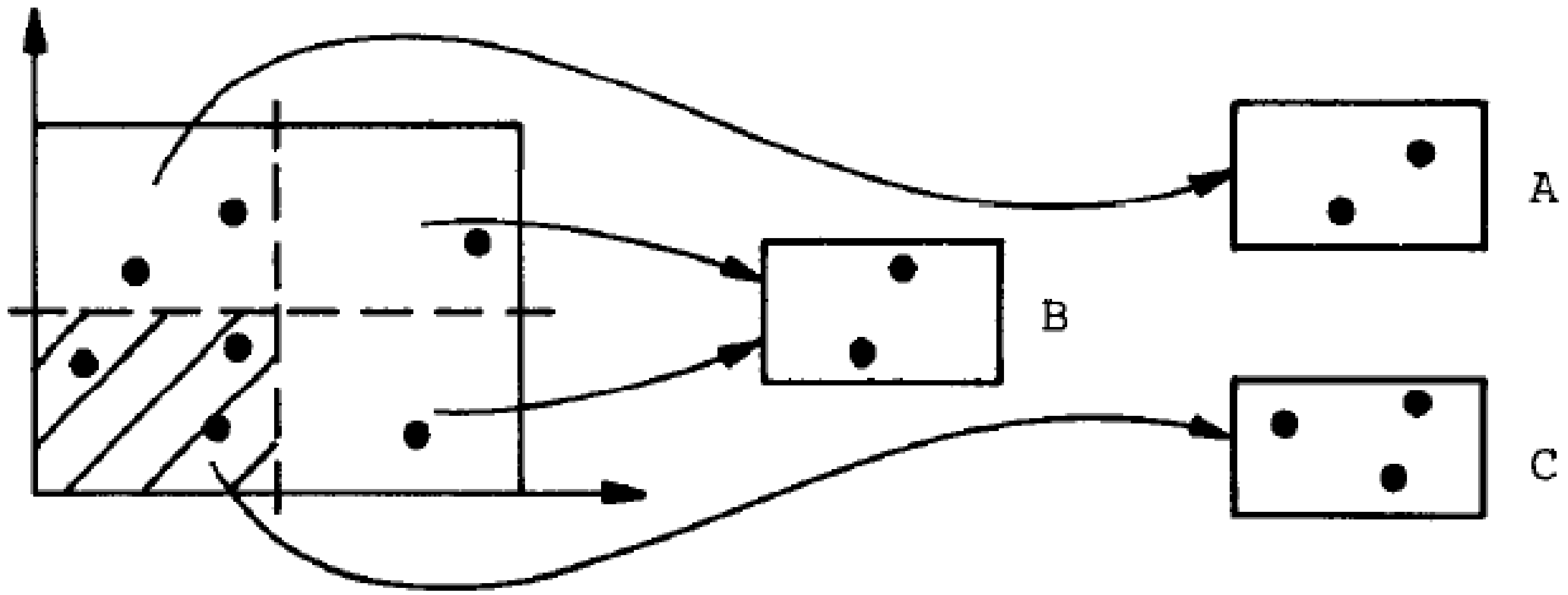


J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

- Bucket A overflows again.

In Grid files, data space which are the buckets is different from the geographic spread of the data.

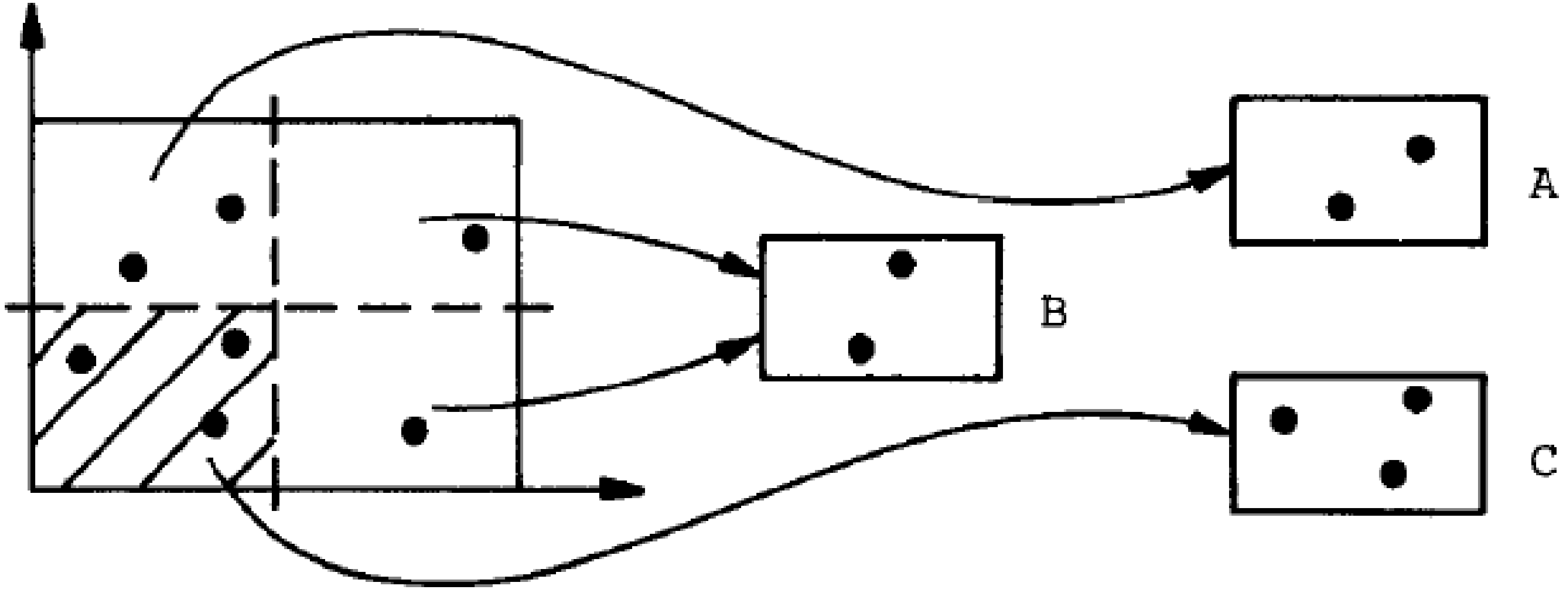


J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

- Bucket A overflows again.

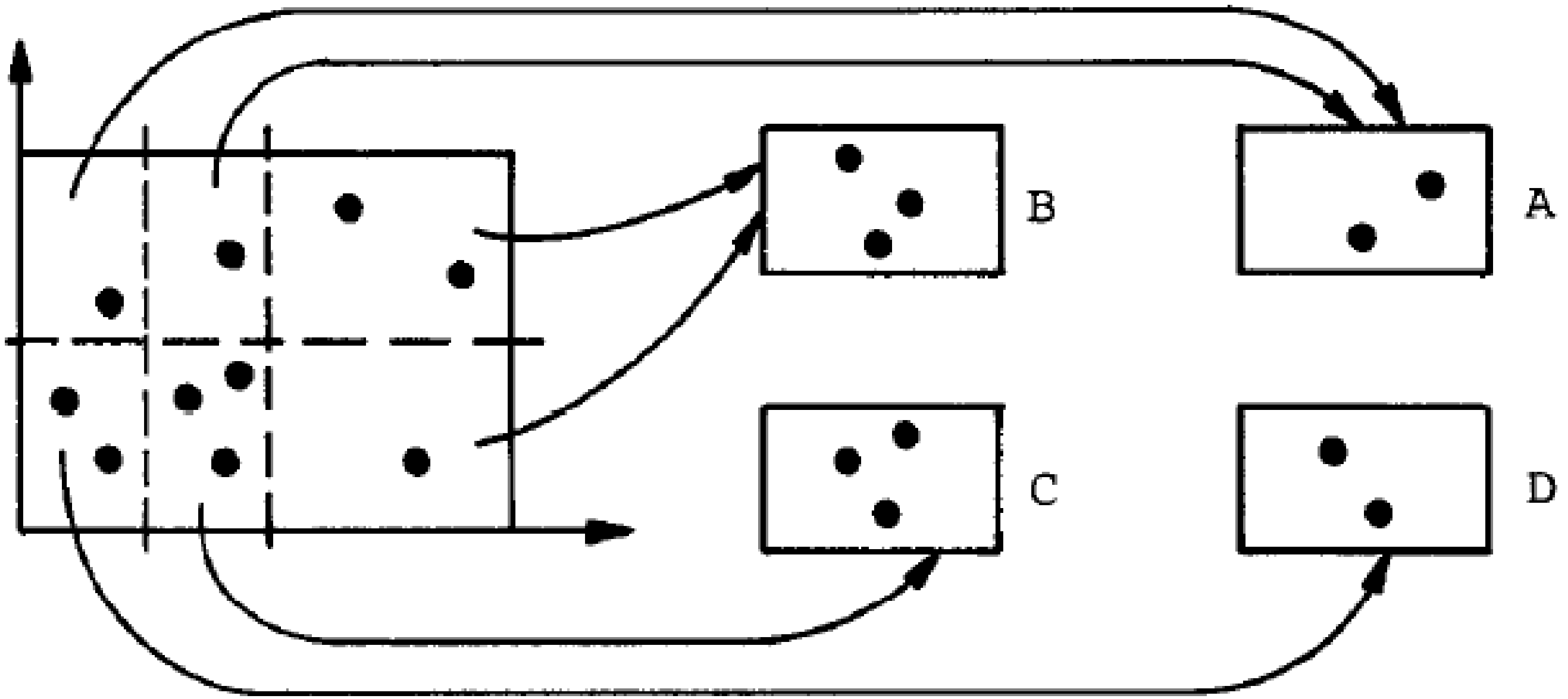
Splits in any dimension are made through and trough out. This makes the task of maintain linear scales easy



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

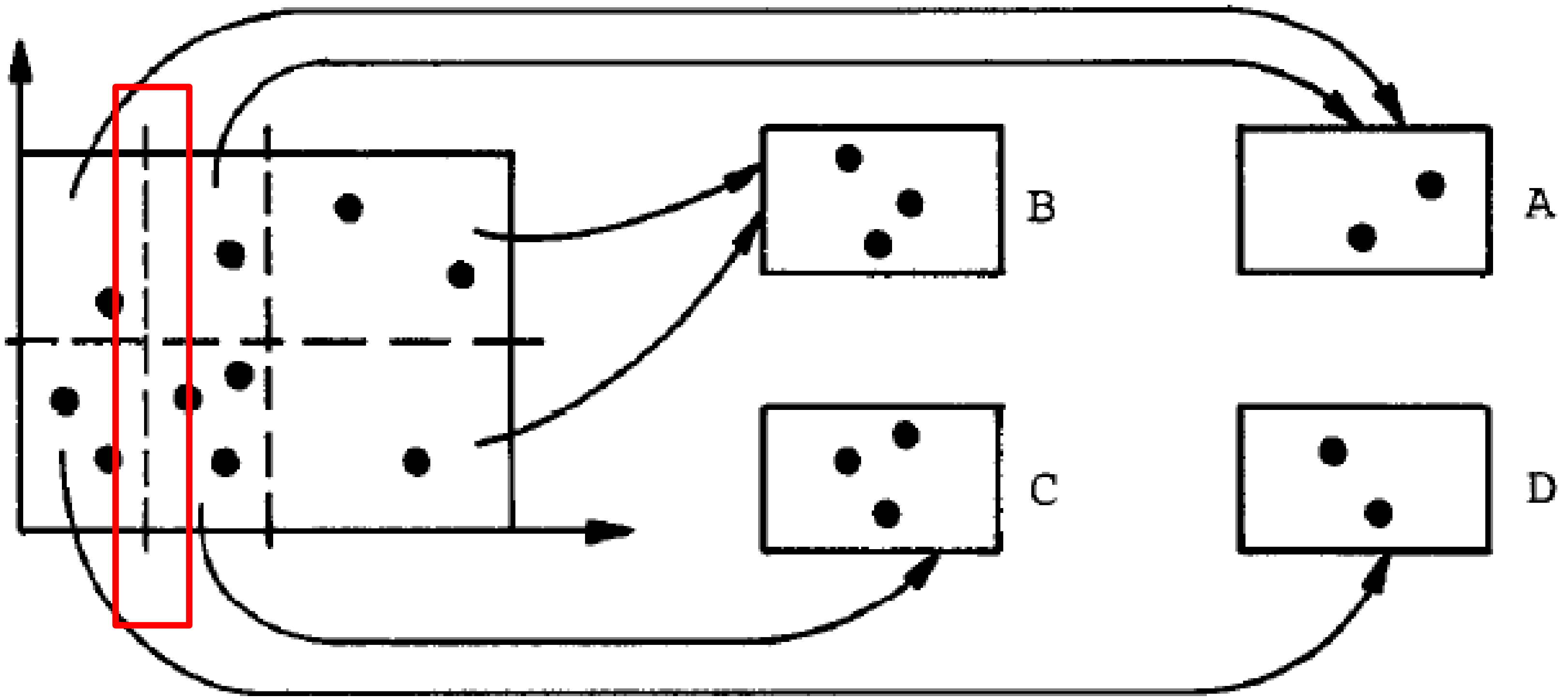
- One more split.



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (insertion example)

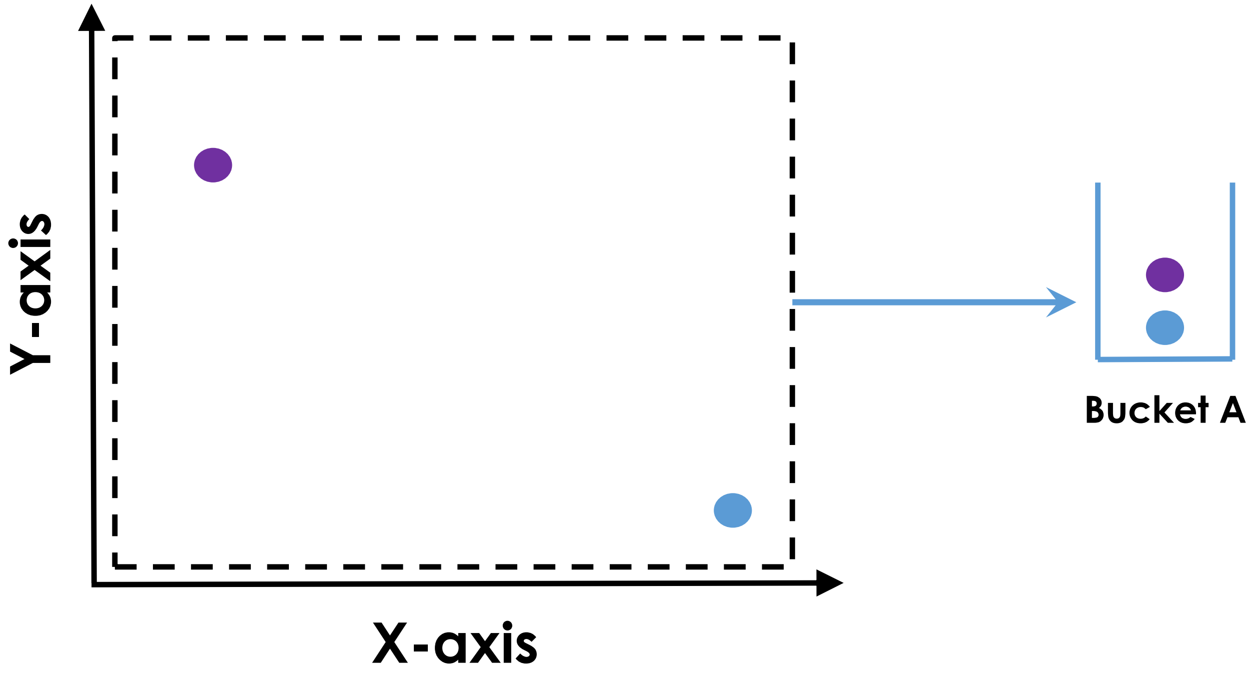
- One more split.
- Note that splits in any dimension are made through and trough.



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

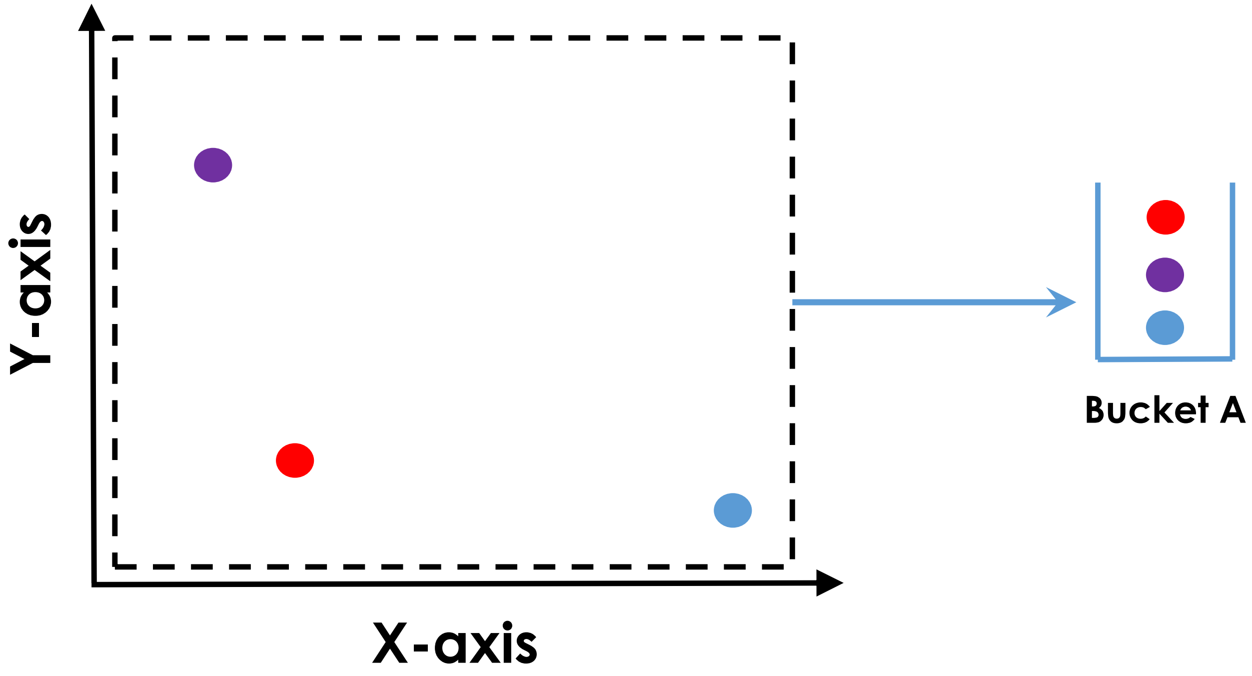
Grid Files (Another example)

- Assume Bucket size = 3



Grid Files (Another example)

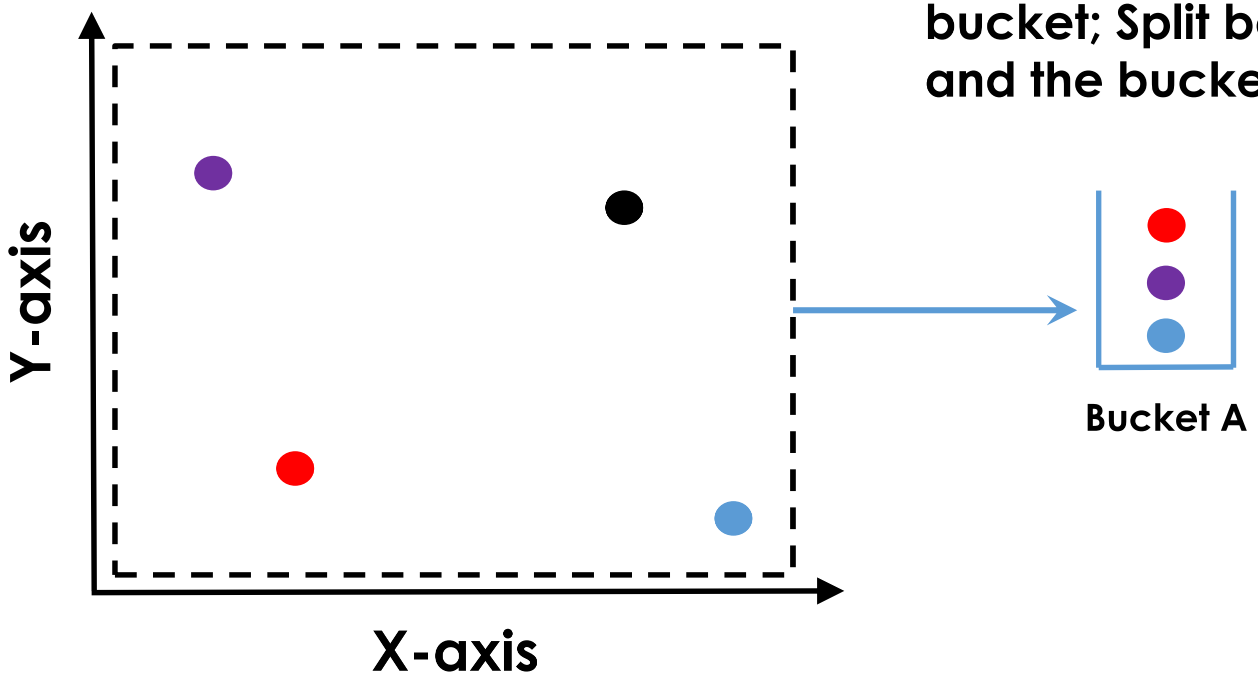
- Assume Bucket size = 3



Grid Files (Another example)

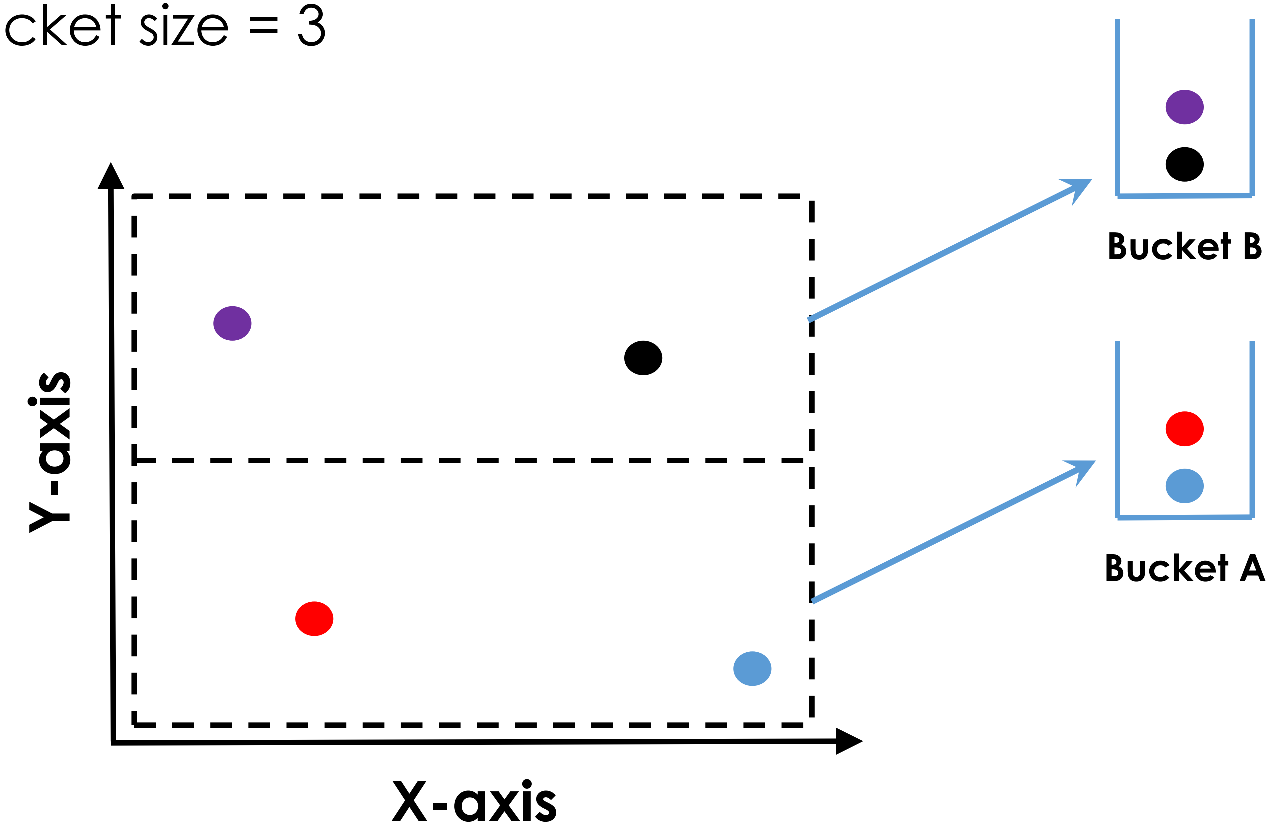
- Assume Bucket size = 3

Overflow! Create a new bucket; Split both scales and the bucket



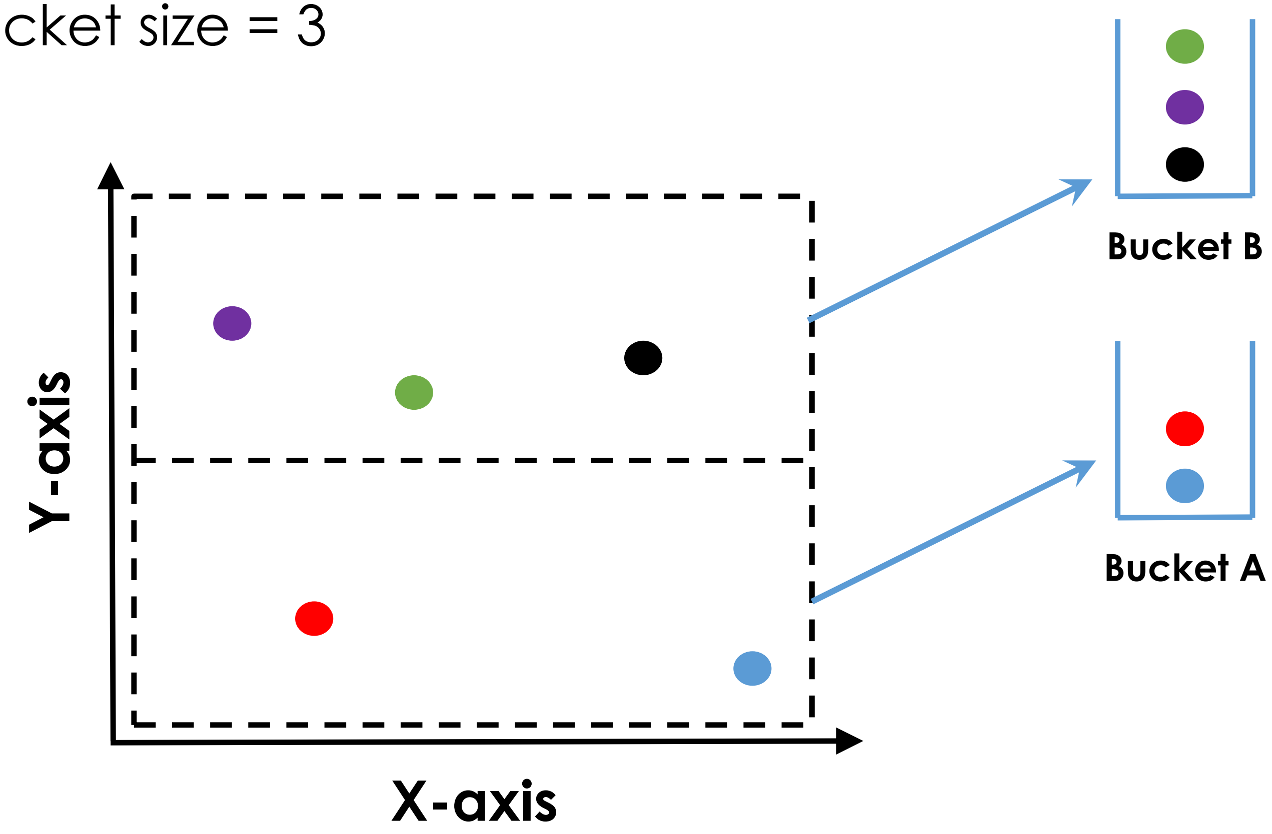
Grid Files (Another example)

- Assume Bucket size = 3



Grid Files (Another example)

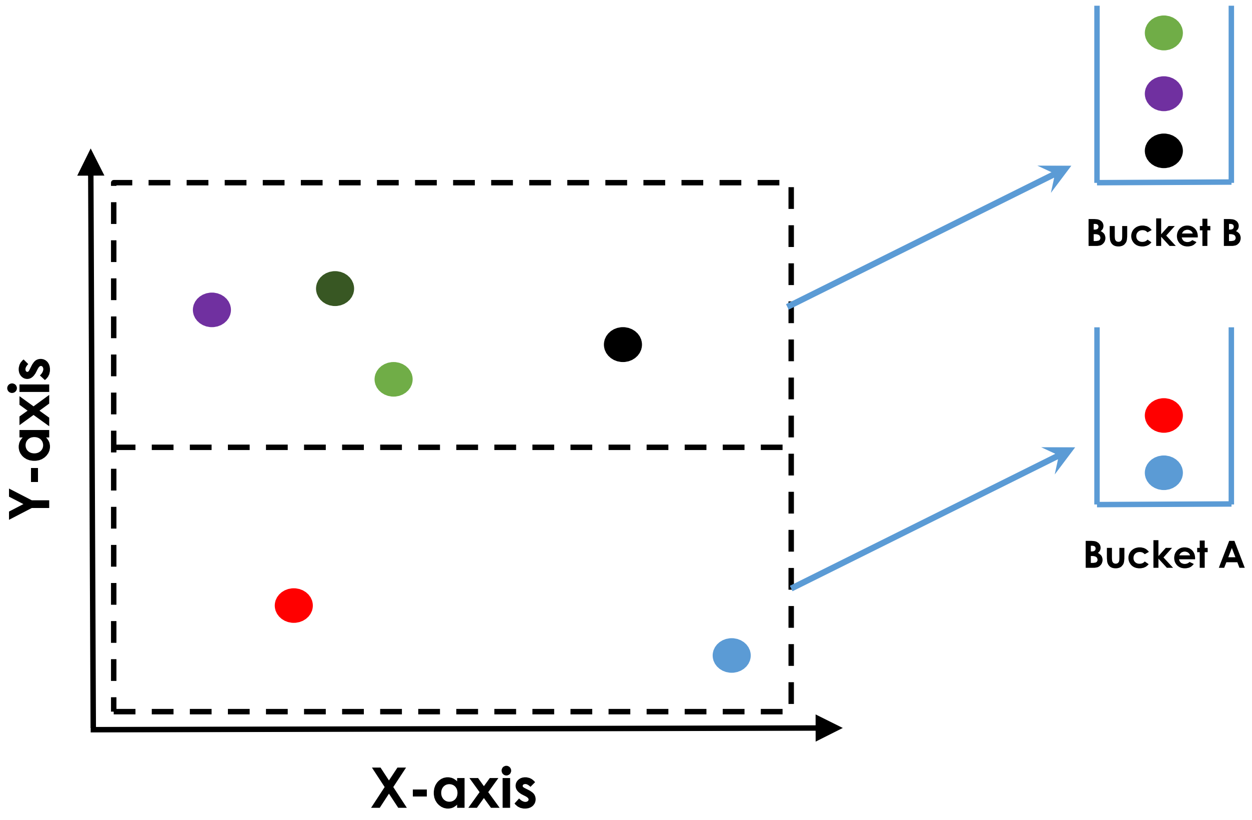
- Assume Bucket size = 3



Grid Files (Another example)

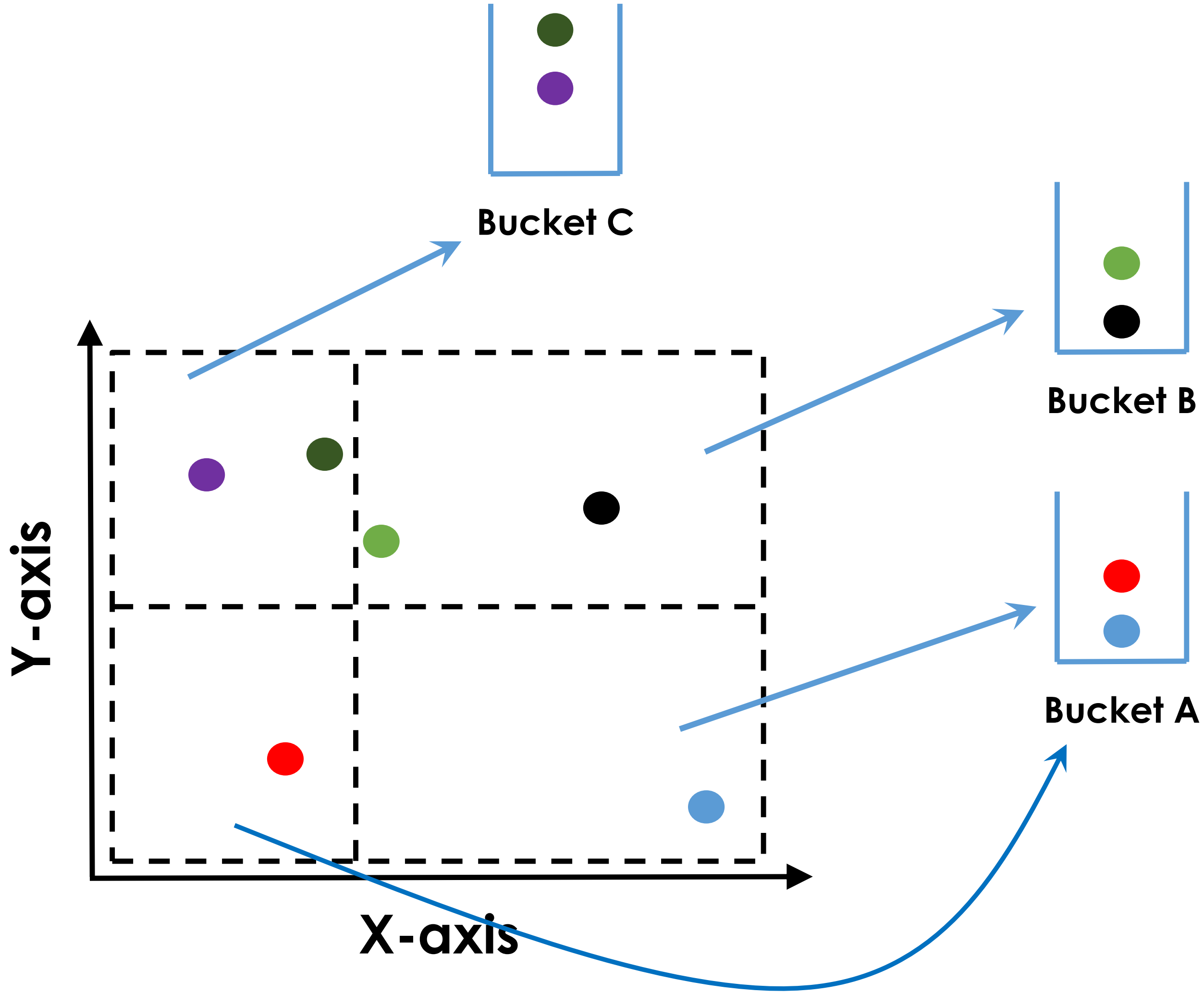
- Assume Bucket size = 3

Overflow! Create a new bucket; Split both scales and the bucket.



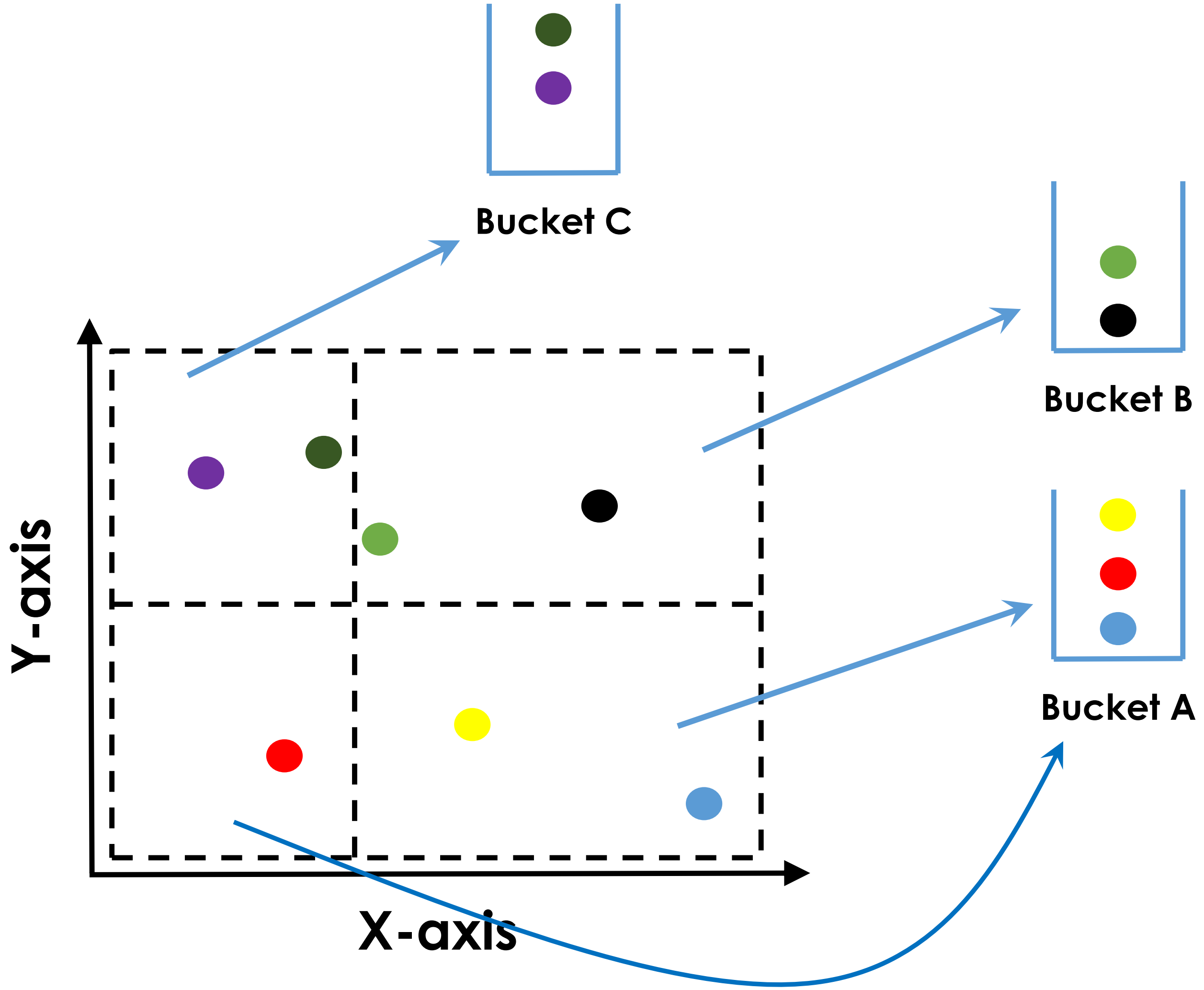
Grid Files (Another example)

- Assume Bucket size = 3



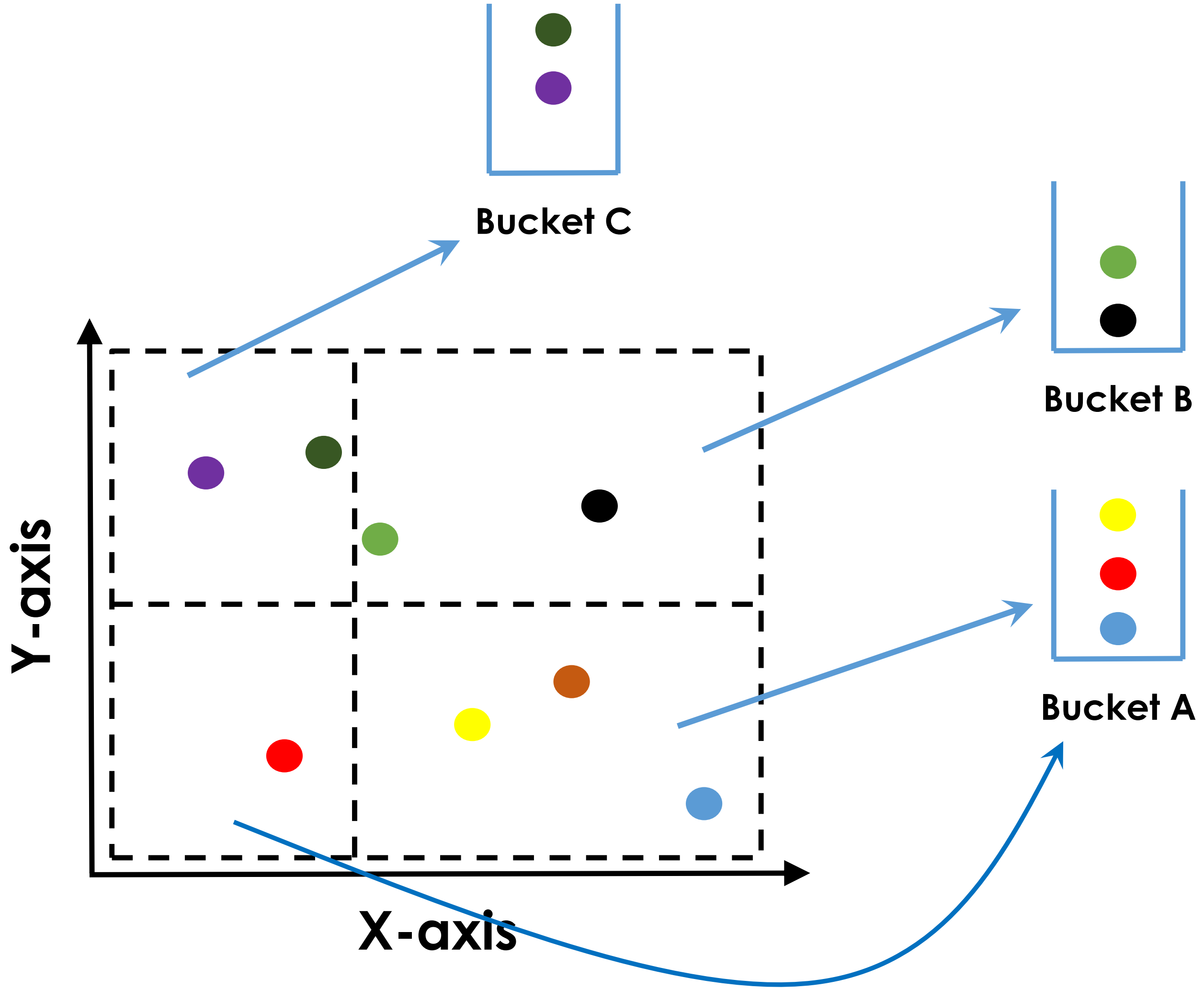
Grid Files (Another example)

- Assume Bucket size = 3



Grid Files (Another example)

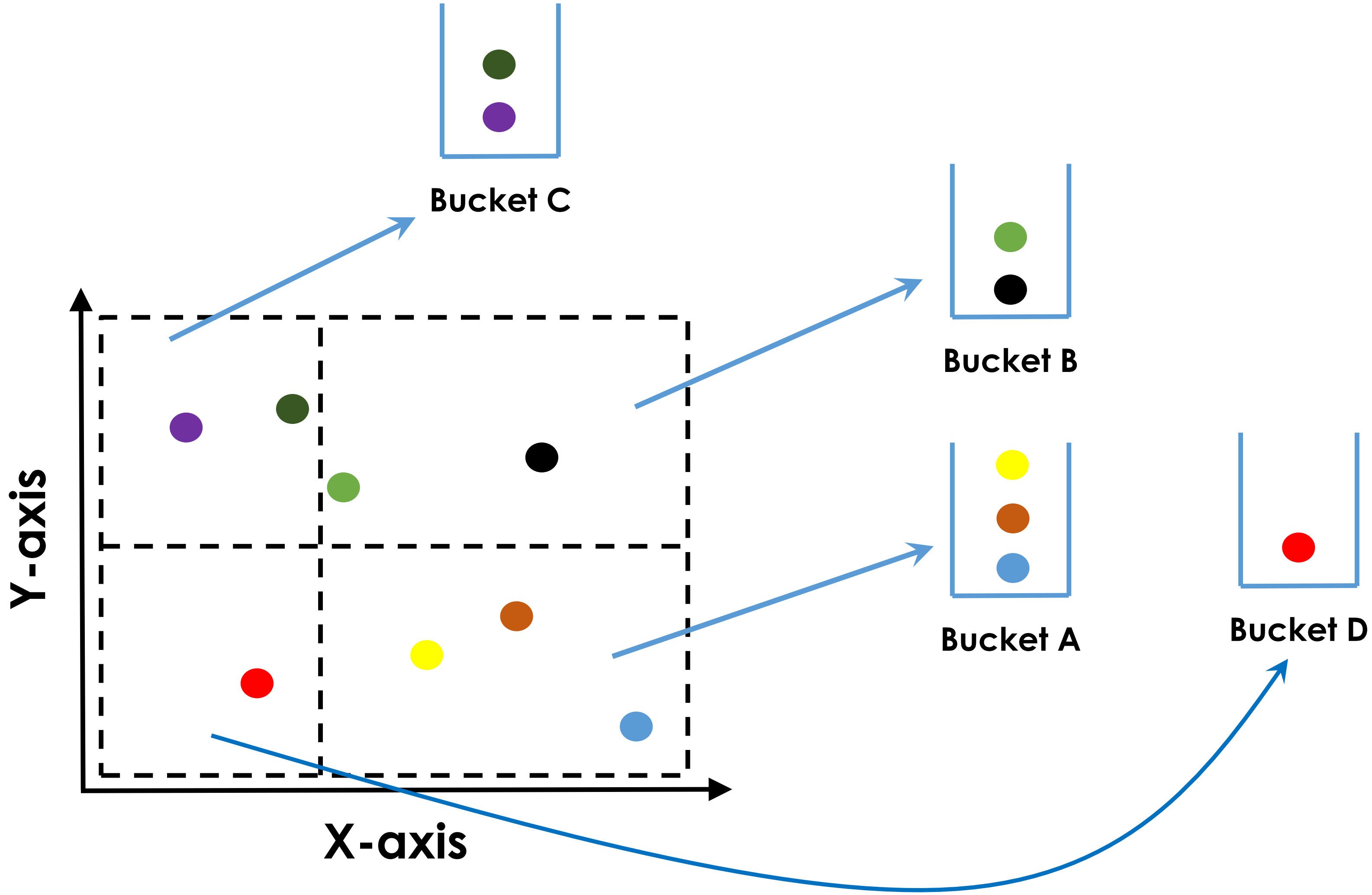
- Assume Bucket size = 3



Overflow! Create a new bucket. Split bucket A.

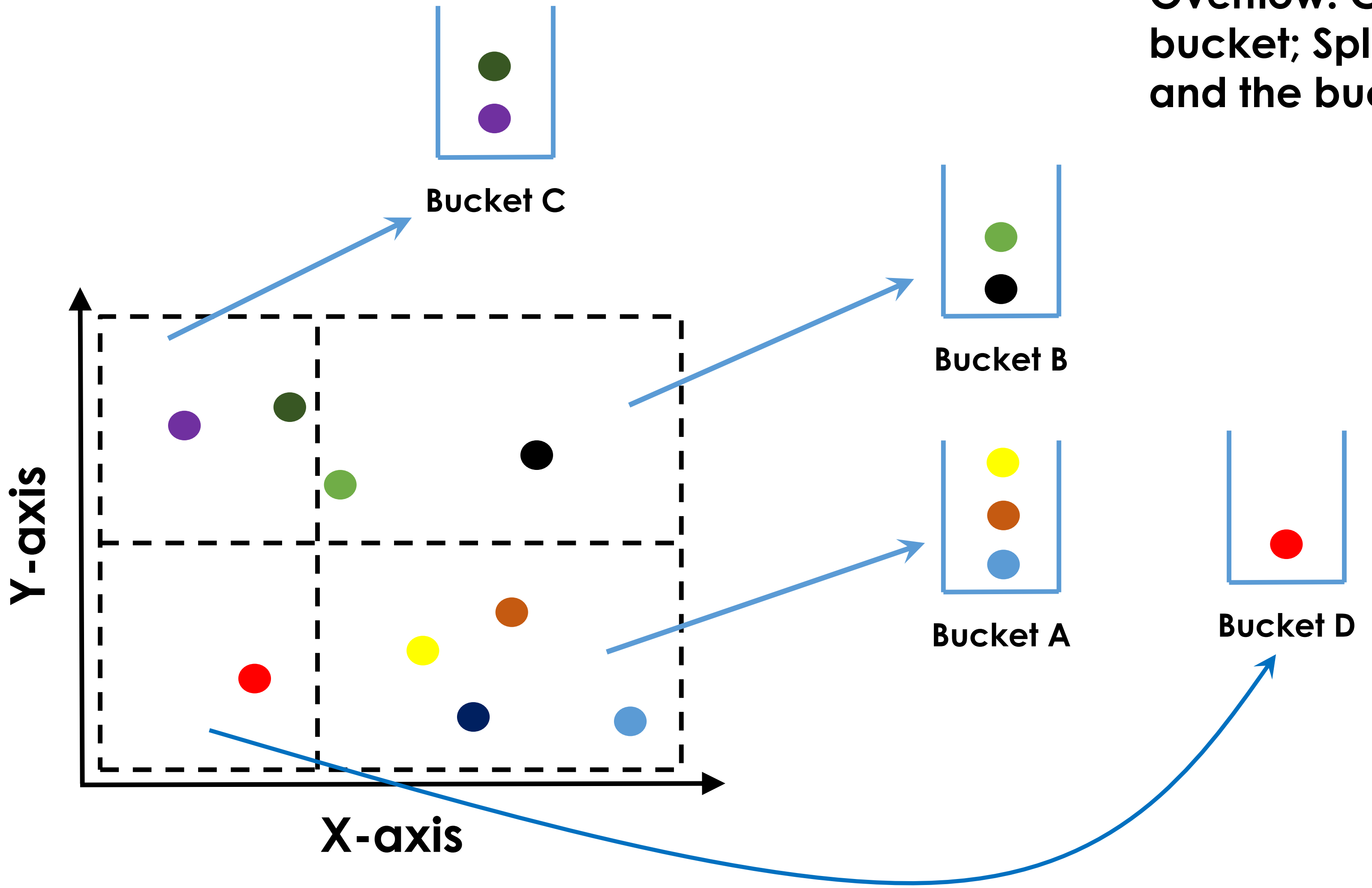
Grid Files (Another example)

- Assume Bucket size = 3



Grid Files (Another example)

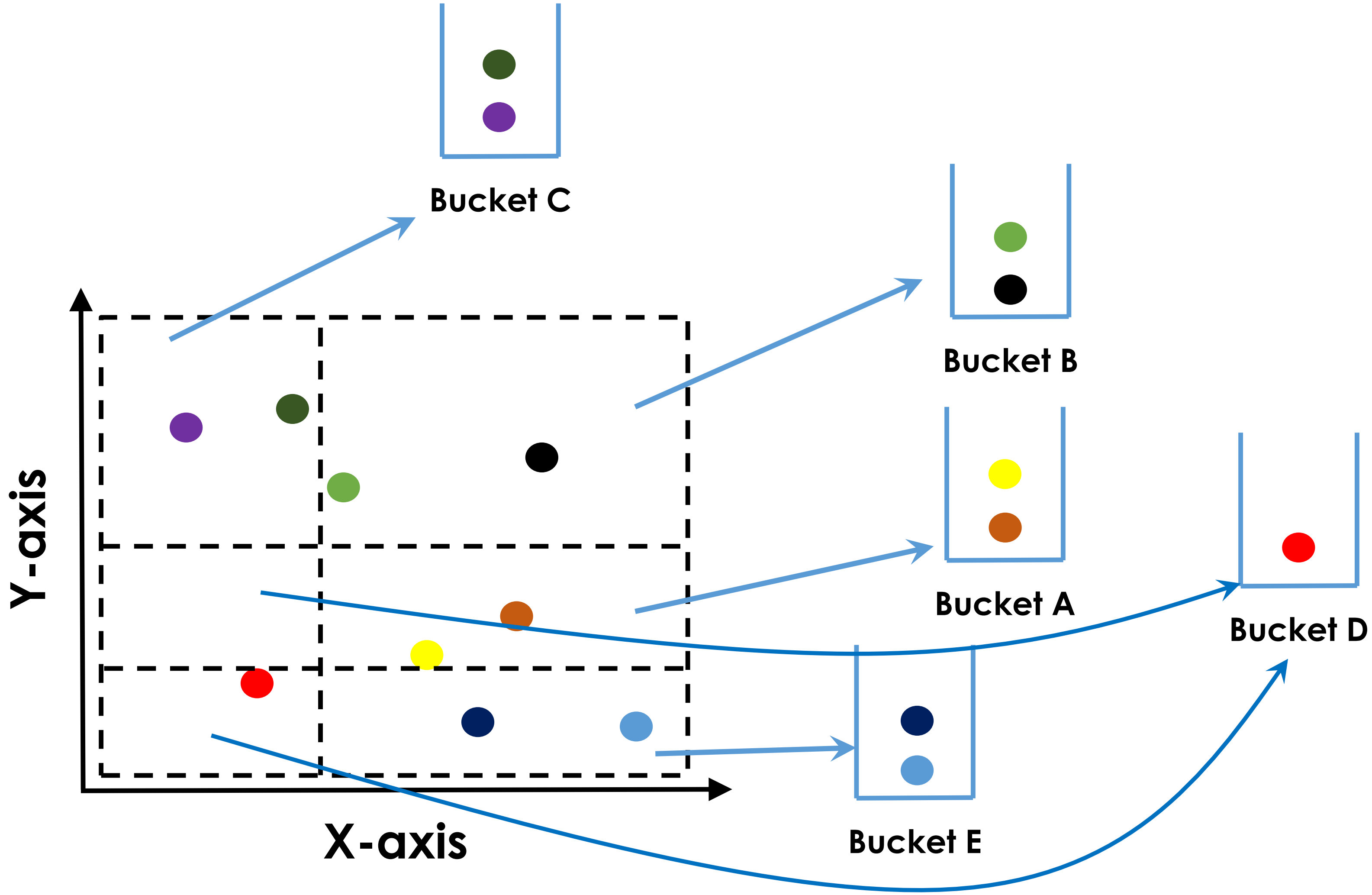
- Assume Bucket size = 3



Overflow! Create a new bucket; Split both scales and the bucket.

Grid Files (Another example)

- Assume Bucket size = 3



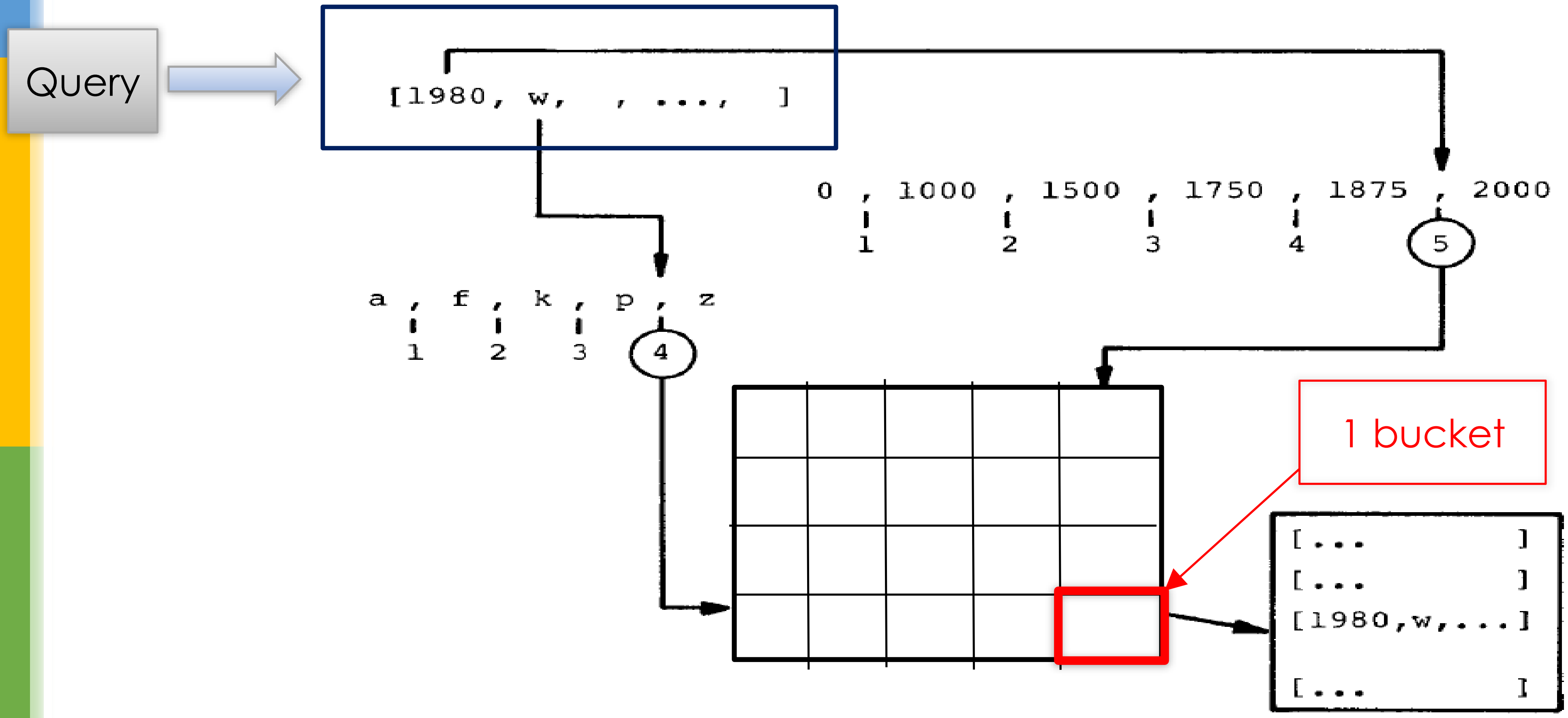
Grid Files (Splitting Policies)

- **Splits:**
  - Can happen during insertion.
  - Overflow of a bucket corresponding to a grid partition leads to a split.
  - Can also happen if bucket containing records from several grid partition fills up.
  - Splitting dimension can be changed alternatively.
  - Splitting point may not always be the middle point, other algorithms are also possible.

J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (Querying example)

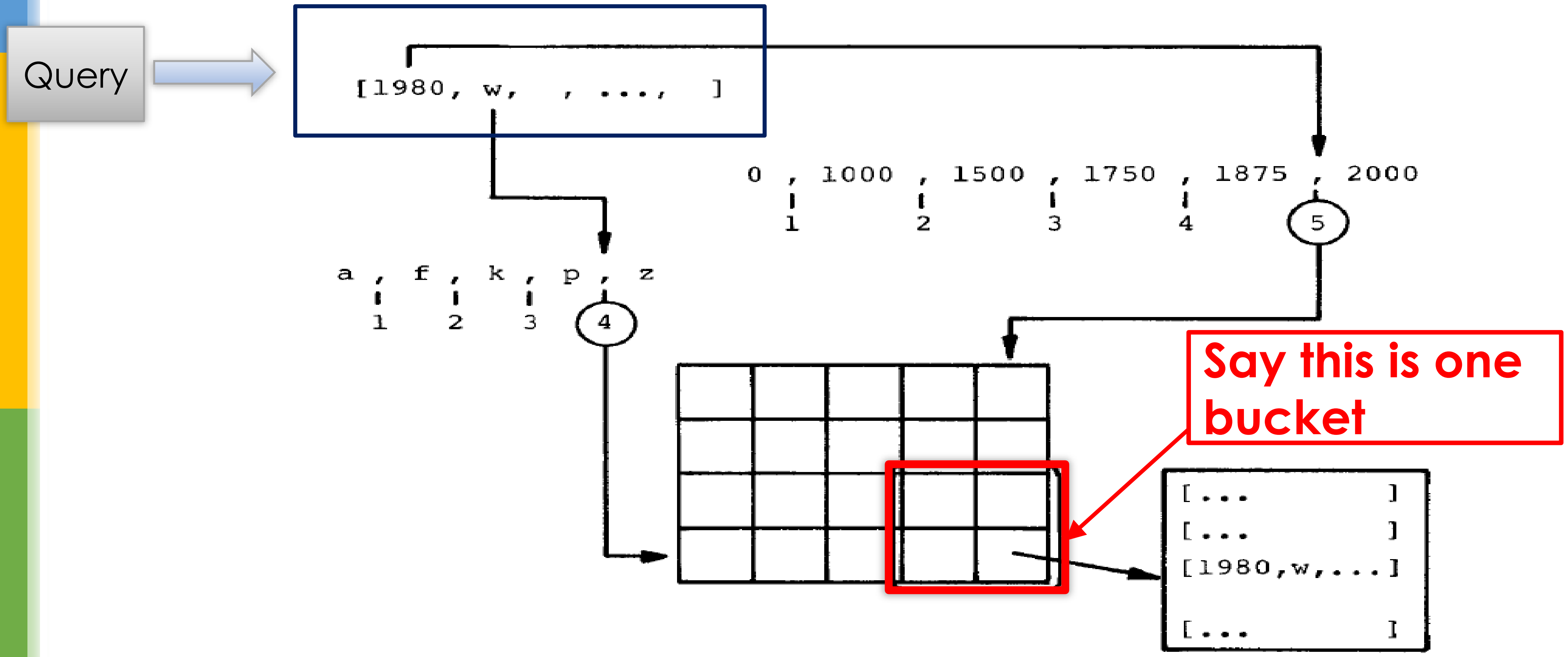
- X-partitions (0,1000,1500,1750,1875,2000)
- Y-partitions (a, f, k, p, z).



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (Querying example)

- X-partitions (0,1000,1500,1750,1875,2000)
- Y-partitions (a, f, k, p, z).

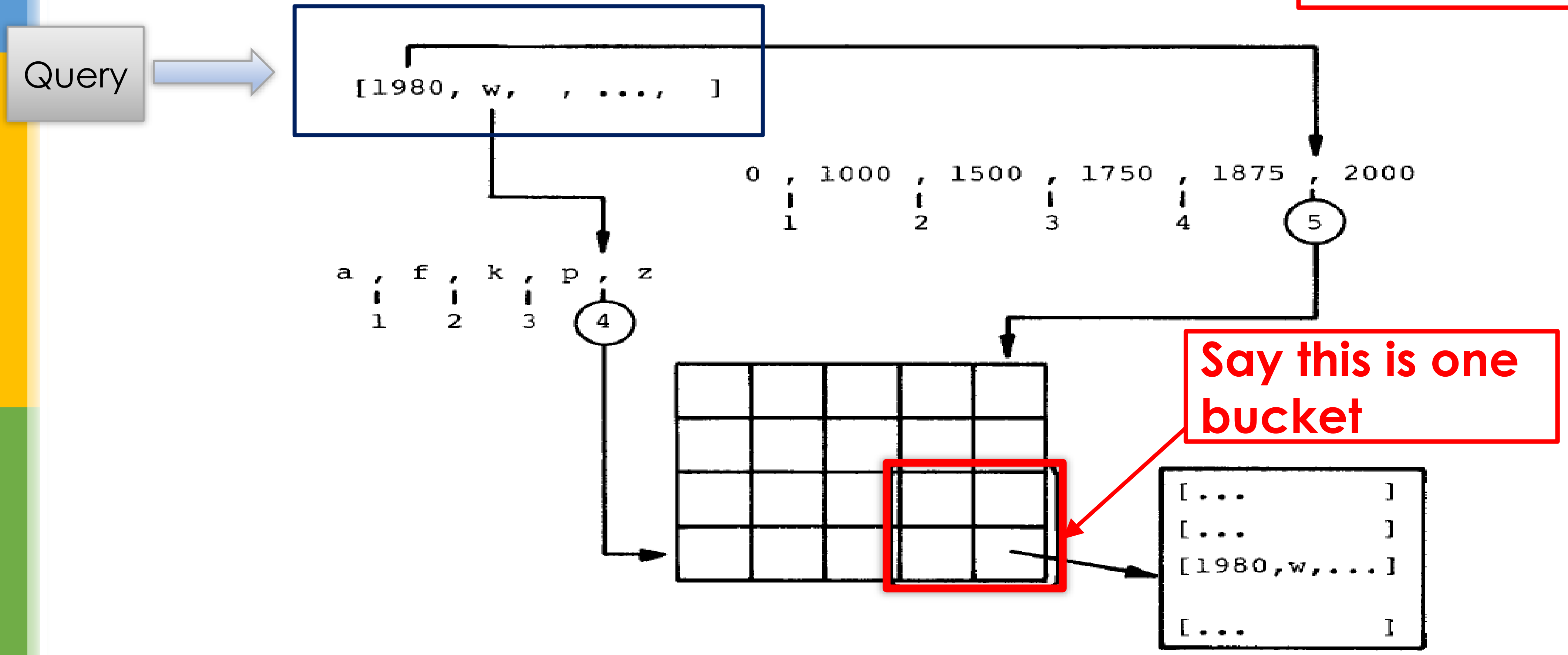


J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (Querying example)

- X-partitions (0,1000,1500,1750,1875,2000)
- Y-partitions (a, f, k, p, z).

Thoughts on Precision and Recall of the initial step of this algorithm?



J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994

Grid Files (Merging Policies)

- **Merging:**
  - Happens when data is being deleted.
  - Buckets may be merged in case of underflow.
  - Multiple policies can be developed for merging.
  - Details beyond the scope of this course.
  - Interested readers can refer the paper for details.

J. Nievergelt and H. Hinterberger. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems, Vol. 9, No. 1, March 1994