COMP5338 – Advanced Data Models

Week 10: Spatial Index

Dr. Ying Zhou
School of Computer Science

Outline

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the **University of Sydney** pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

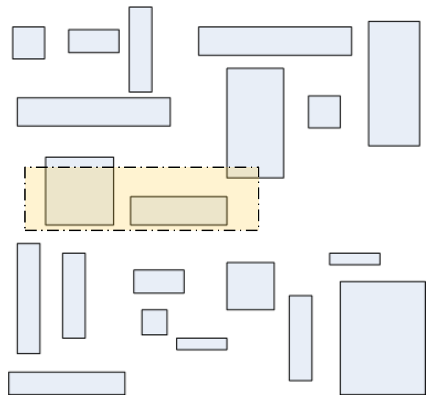
Do not remove this notice

- Index Motivation
- Hash Structure
- Tree Structure
- Space Filling Curve Techniques

Revisit: File Organization and Index Basics

- Any disk based storage systems store data in files
- Part of the file is read into memory during read/write operation
- Files are organized into fixed sized disk blocks (e.g. 4K in NTFS)
  - Each block may contain a few data records
  - It is the basic IO unit
- Index is also stored as file
  - Also consists of blocks
  - May be loaded entirely in the memory
  - Is used to decide which block(s) of the data file need to be loaded
  - It reduces disk I/O cost as well as record inspection cost

Typical Spatial Query

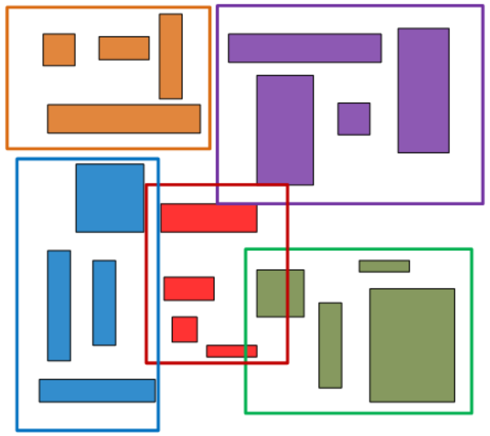


Find all rectangles that intersect a rectangular query range

Organizing Storage Block

The assumption is that the database hold mostly spatial objects and most query workload consists of spatial queries

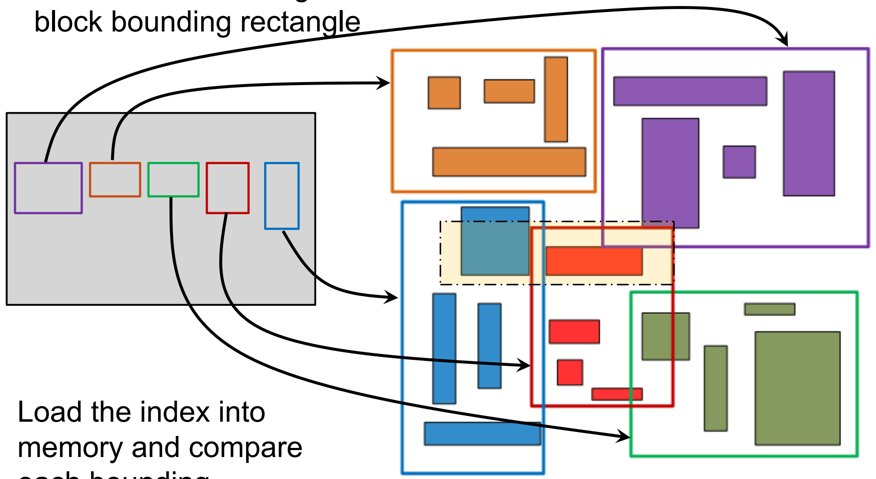
We store objects spatially close to each other in the same disk block



Suppose each storage block can hold up to 4 objects, we want to minimize the number of blocks retrieved

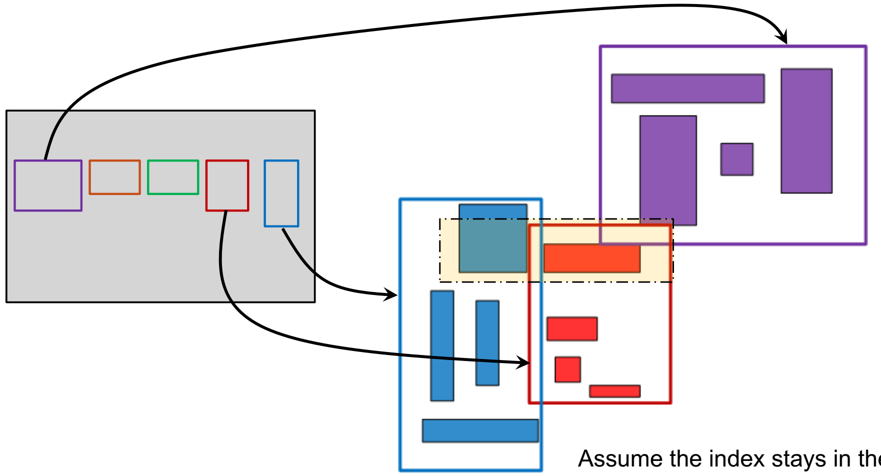
Indexing Storage Block

Create indexes using block bounding rectangle



Load the index into memory and compare each bounding rectangle with the query rectangle

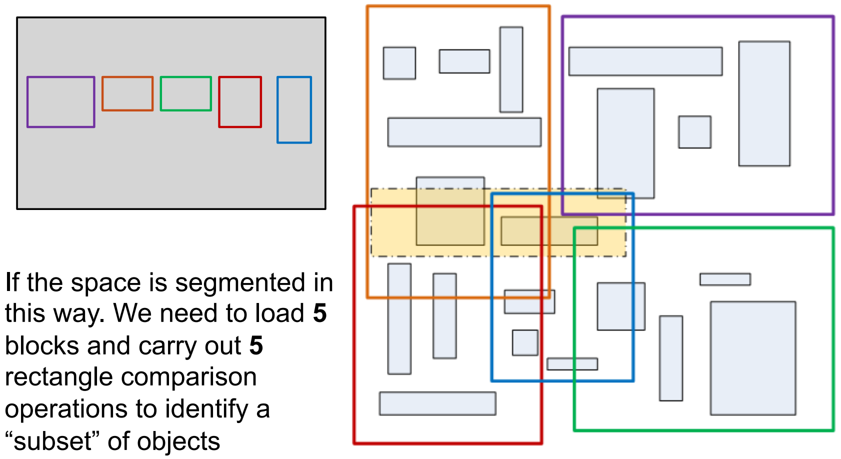
Read 3 blocks



Assume the index stays in the memory. To answer this query, **3** data blocks need to be read in memory and **5** rectangle comparison operations need to be carried out to identify a subset of objects

Two issues:
Space segmenting
Index organization structure

Issues in spatial index design



If the space is segmented in this way. We need to load **5** blocks and carry out **5** rectangle comparison operations to identify a "subset" of objects

Space segmenting method (minimize the block I/O)
Index organization structure (minimize index operation)

Outline

General Hash Index Structure

A table with record, keys are letters a through f

A hash function that maps [a,f] to [0,3]

`h(d)=0`
`h(c)= h(e)=1`
`h(b)=2`
`h(a)=h(f)=3`

Bucket array of size **`B`**, each bucket is a block that can store records

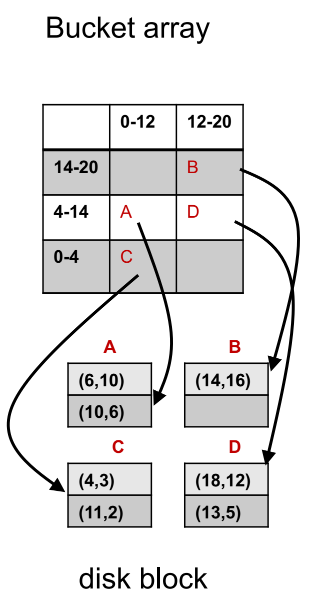
	Block
0	d
1	c
	e
2	b
3	a
	f

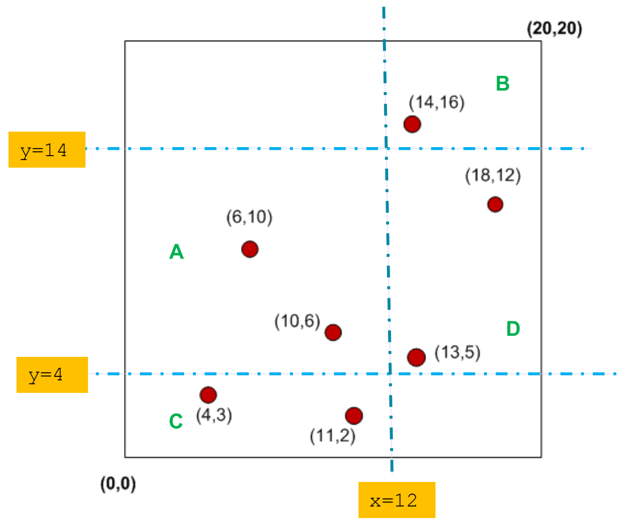
Constant cost for point query regardless of the bucket size
No use for range query.

Grid file

- Space segmenting method
  - Each dimension of the space is partitioned into *stripes* using grid lines
  - The number of grid lines in different dimensions may vary
  - The spacing between adjacent grid lines in the same dimension can be different
- Index is organized using "hash like" structure
  - Space region is like bucket in a hash table

Grid File in Two Dimensional Space

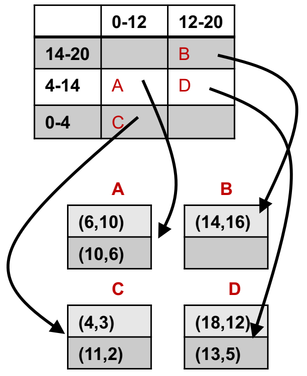




Assuming each disk block can hold up to 2 records

Index Structure

- Grid Directory
  - One dynamic *d*-dimensional array to store bucket location
  - A set of *d* one-dimensional array called **linear scales** to store grid line locations



The d-dimension array for buck location :
[
    [,B],
    [A, D],
    [C]
]

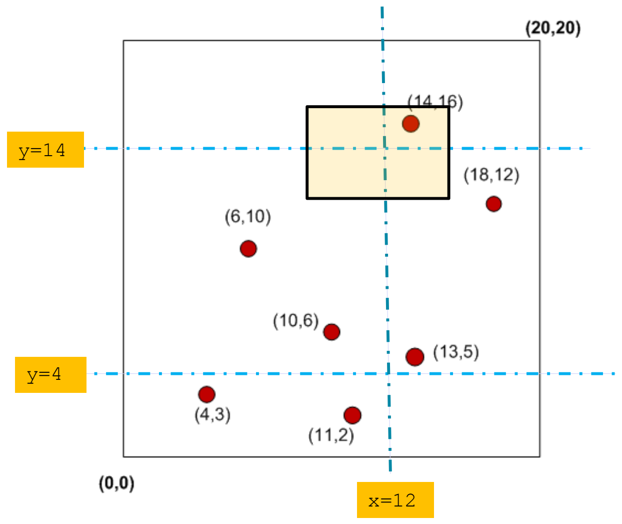
The linear scales are:
X: [0,12,20]
Y: [0,4,14,20]

Common Queries – Point Query

- Lookup of specific point
  - Load grid directory
  - Locate the proper bucket the point might be in
  - Load the bucket

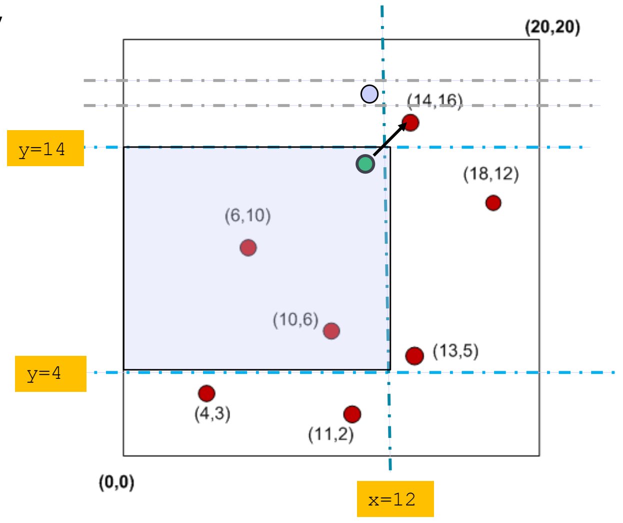
Common Queries – Range Query

- Load grid directory
- Find all buckets that interact with the query region
- Load those buckets
- May have to evaluate many candidates



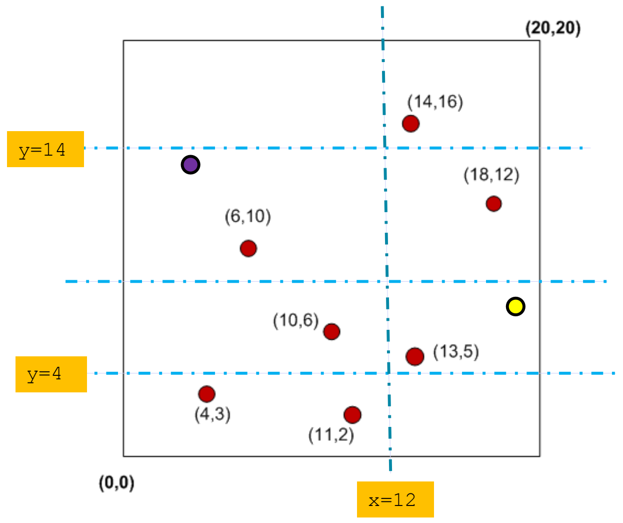
Common Queries – Nearest Neighbour

- Find the bucket the query point belongs to
- All points there would be candidate
- May have to load adjacent buckets to look for other candidates
- If the shape is much longer in one dimension than the other, it may be necessary to load buckets that are not adjacent for other candidates



Insertion into Grid Files

- Find the bucket the new point belongs to
- Add it in the bucket if there is room, otherwise
  - Add overflow block to the bucket
  - Reorganize the structure by adding or moving the grid lines

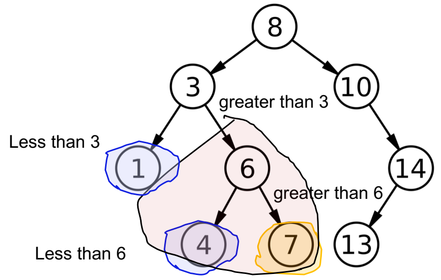


Outline

- Indexing Motivation
- Hash Structure
- **Tree Structure**
  - **Kd-Tree**
  - **Quad-Tree**
  - **R-Tree**
- Space Filling Curve Techniques

kd- Tree (K-dimensional Search Tree)

- A generalization of binary search tree to handle multidimensional data
  - Main memory data structure
  - Storage based index structure
- Binary Search Tree
  - The internal node each stores a key greater than all the keys in the node's left subtree and less than those in its right subtree.



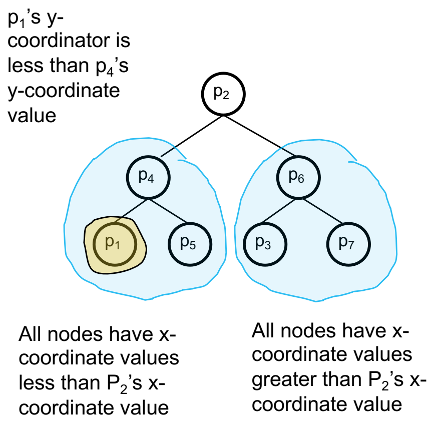
https://en.wikipedia.org/wiki/Binary_search_tree

Kd-tree

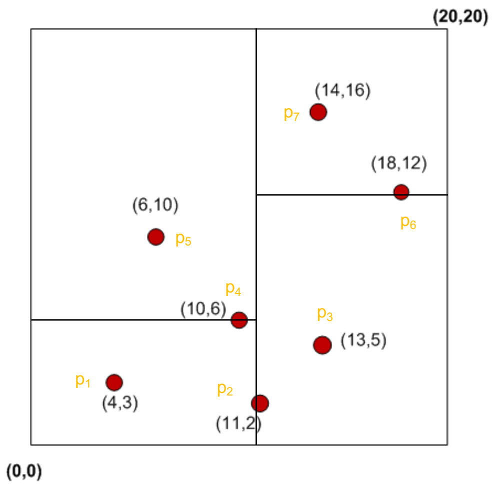
- Search key used at different level belongs to a different dimension
- In two dimension space, the x and y dimension alternates at levels
- Split the point set alternatively by x-coordinate and by y-coordinate
  - split by **`x-coordinate`**: split by a vertical line that has half the points left or <u>on</u>, and half right
  - split by **`y-coordinate`**: split by a horizontal line that has half the points below or <u>on</u>, and half above

Classic kd-tree example

- The tree nodes are the **points** in the data set, there is a predefined order of which dimension to use


p1's y-coordinator is less than p4's y-coordinate value
p2
p4
p6
p1
p5
p3
p7
All nodes have x-coordinate values less than P2's x-coordinate value
All nodes have x-coordinate values greater than P2's x-coordinate value

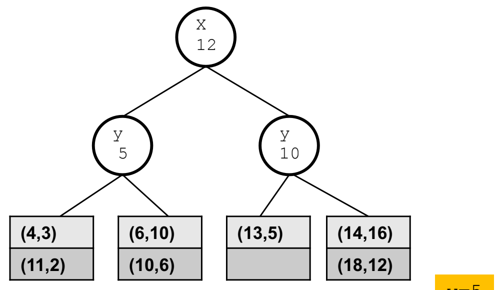
Used as in memory tree



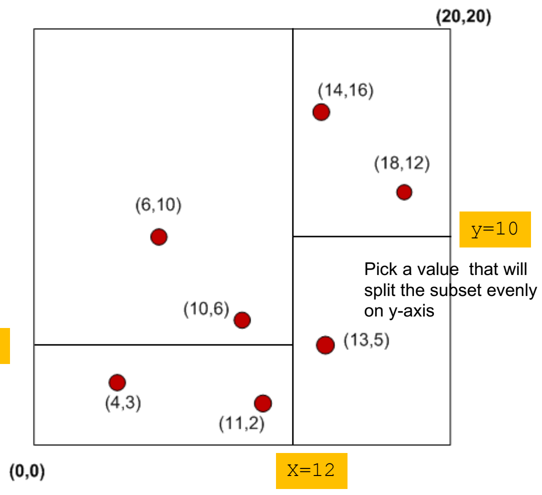
Pick a node that will split the set evenly on x-axis

Kd- tree index example

- Modification of classic kd-tree to use as **index**
  - Interior node will have only **one attribute**, a **dividing value** for that attribute, and pointers to left and right children
  - Leaves will be blocks



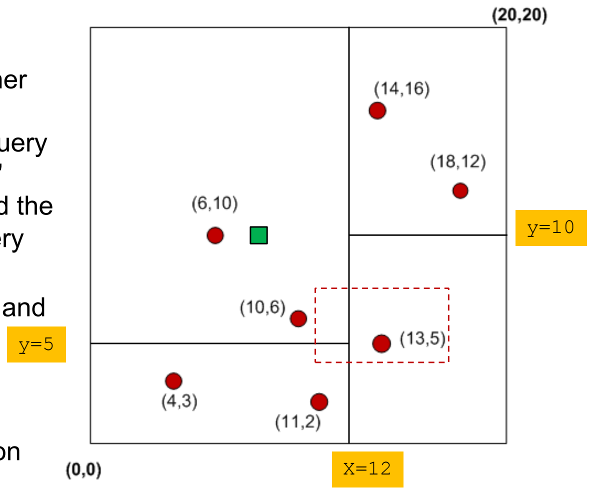
Pick a value that will split the subset evenly on y-axis



Pick a value that will split the set evenly on x-axis

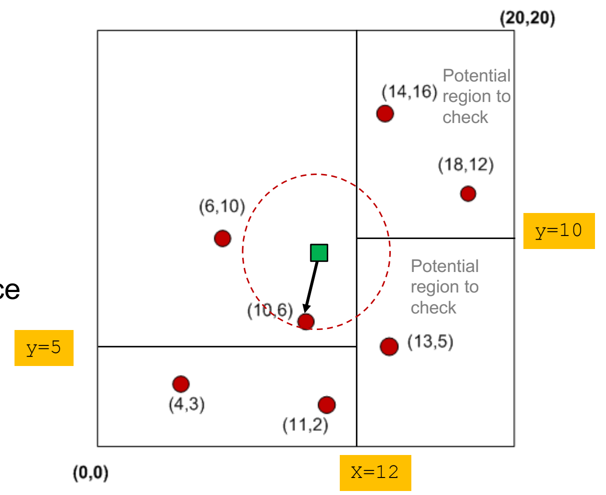
Common Queries

- Kd tree is used to store point data
- Point Query
  - Start from root, move to either the left or the right child depending on whether the query point is on the "left" or "right" side of the splitting line. Load the leaf block to check if the query point exists in the database
  - Similar process for inserting and updating points
- Range query
  - Need to check all regions intersect with the query region



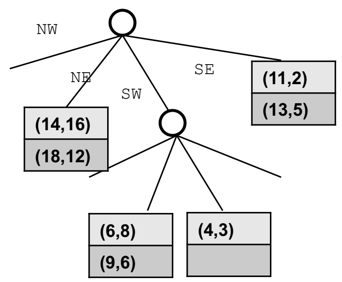
Nearest Neighbor Query

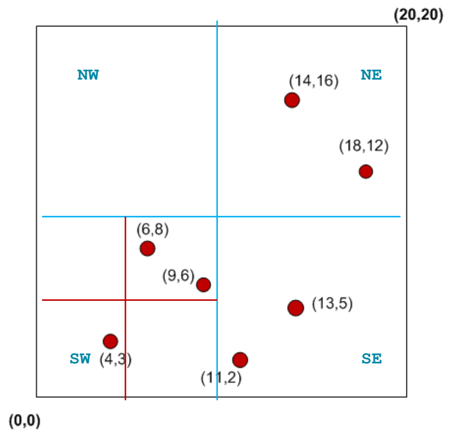
1. Use the point query process to locate the the spatial segment the query node is supposed to be in.
2. Find the current nearest neighbor in this segment and the current shortest distance
3. Identify other regions that needs to check based on the current shortest distance
4. Repeat 2-3 until no further region need to be checked



Quadtree

- In a quad tree, each interior node has exactly four children representing the four quadrants (normally square or rectangle shape) of the underlying space
- The query processing is similar to kd tree



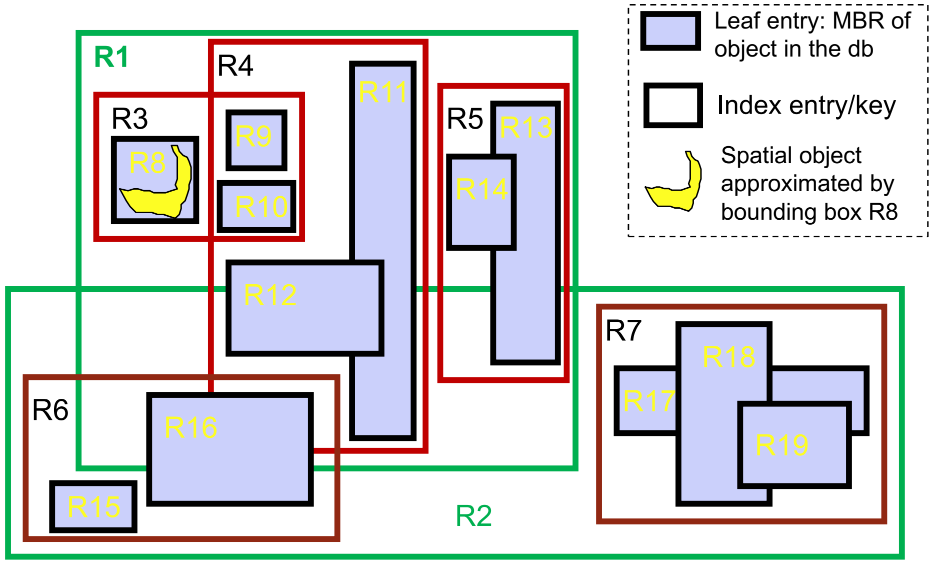


R-Tree

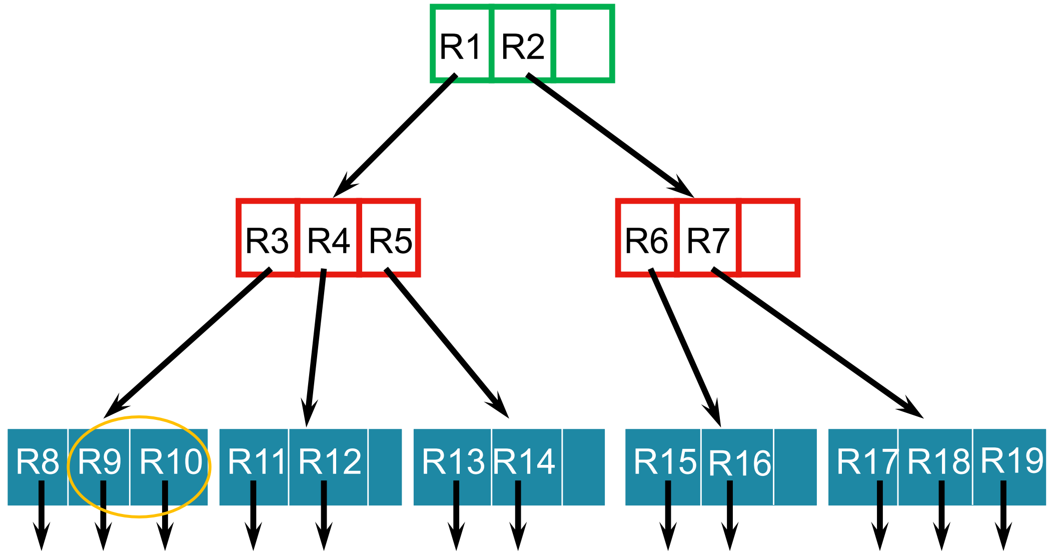
- R-Trees are hierarchical data structure based on B+ trees, except that it represents data in 2-dimensional regions
- Difference to B+ tree
  - Keys are minimum bounding rectangle (MBR) regions, instead of single value
    - No strict order of all keys
  - Interior node represents MBR's of its children
    - There is order of keys along a tree branch
  - Leaf node represents a number of MBRs of the spatial objects in the database
  - MBRs for different nodes may overlap
  - One MBR maybe covered by many upper level nodes' MBRs, but it can be associated with **only one** node

R-Tree Example -- Space

Each internal node can have maximum 3 and minimum 2 children


R1
R4
R3
R8
R9
R10
R11
R5
R13
R14
R12
R7
R18
R17
R19
R6
R16
R15
R2
Leaf entry: MBR of object in the db
Index entry/key
Spatial object approximated by bounding box R8

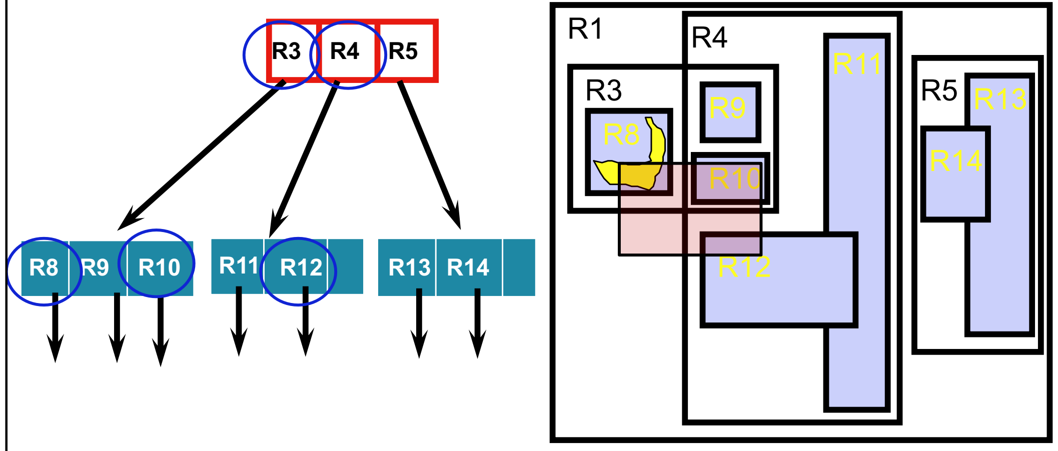
R-Tree Example – Index



Operations on R-Tree

- Most are similar to B+ tree operation
  - A search may have to examine several siblings
- Where am I query: given a location (as a point **P**), find the data region or regions the point belongs to
  - Start with root
  - Find **subregions** S containing P:
    - if S is a data region: return S
    - else: recursively search S
  - If no subregion containing P
    - Stop and return that P is not in any data region

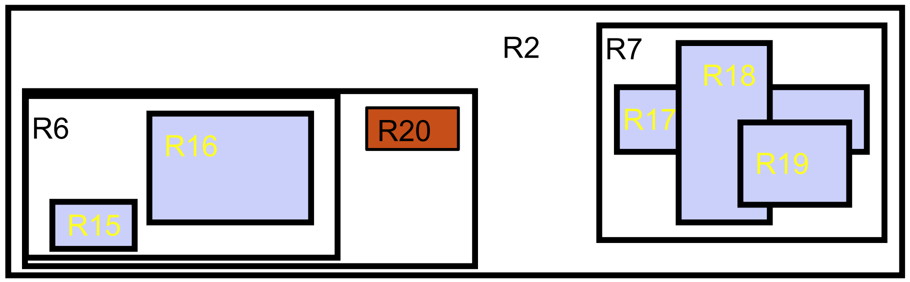
Examining multiple siblings



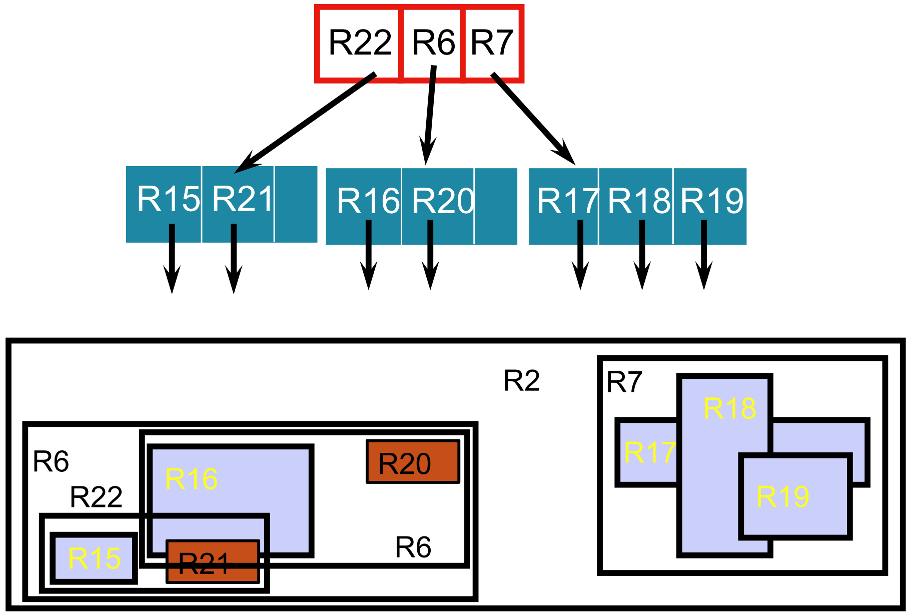
One internal node’s region

Expansion and Split Region

- Regions might be expanded or split during data insertion
- Objectives
  - minimize covered area of the containing region



Expansion and Split Region

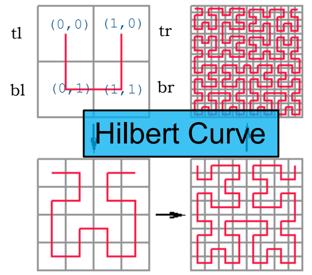


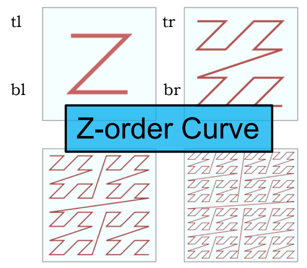
Outline

- Indexing Motivation
- Hash Structure
- Tree Structure
- **Space Filling Curve Techniques**
  - **Z-order Curve**

Space Filling Curve Techniques

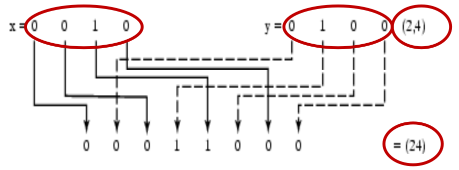
- Techniques of encoding multidimensional data as 1 dimensional value
- Relative locality of multidimensional data are preserved in most cases
- We can think of placing all the points (regions) in space in some order, on a curve
- There are many ways of ordering points





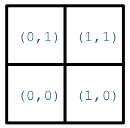
Z-order Curve

- Also called **Morton order** or **Morton code**
- Computing the z-value or order of a point is simple if coordinate can only take integer values
  - Interleaving the binary representation of the point's coordinate values
- Not all neighbours have close z-value

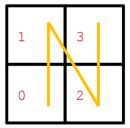


Shekhar Fig 4.6

Computing level 1 Z-value



4 points in 2 dimensional space

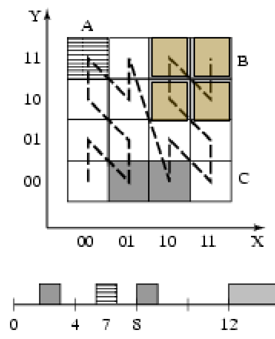


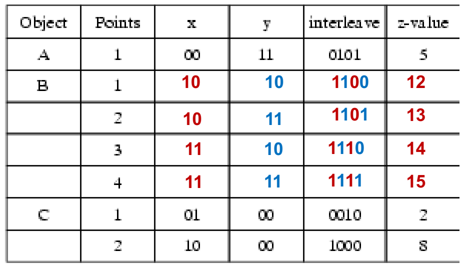
Corresponding Z-value of the 4 points

Point coordinate	Binary representation	Z-value
(0,0)	(00,00)	0000
(0,1)	(00,01)	0001
(1,0)	(01,00)	0010
(1,0)	(01,01)	0011

Example of Z-Values

- Left part shows a map with spatial object A, B, C
- Right part and Left bottom part Z-values within A, B and C
- Note C gets z-values of 2 and 8, which are not close
- Exercise: Compute z-values for B.





Object	Points	x	y	interleave	z-value
A	1	00	11	0101	5
B	1	10	10	1100	12
	2	10	11	1101	13
	3	11	10	1110	14
	4	11	11	1111	15
C	1	01	00	0010	2
	2	10	00	1000	8

Shekhar Fig 4.7

References

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Database systems : the complete book (2nd edition)
  - Chapter 14
- A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data. Boston, Massachusetts: ACM, 1984.*
- S. Shekhar and S.Chawla: *Spatial Databases: A Tour*. Prentice Hall, 2002. [http://www.spatial.cs.umn.edu/Book/]
  - Chapter 4