Problem Statement 3

Data Structures and Algorithms Project

STR Method of Packing RTrees

Group 33

Apul Ranjan
2021A7PS2415P

Peeyush Vatsa
2021A7PS0007P

Arush Dayal
2021A7PS0011P

Chandrashekar Ramachandran
2021A7PS2451P

Nachiket Singh Kandari
2021A7PS2691P

Anti-Plagiarism Statement

We know that plagiarism means taking and using the ideas, writings, works or inventions of another as if they were one's own. We know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgement . We know that plagiarism covers this sort of use of material found in textual sources and from the Internet.

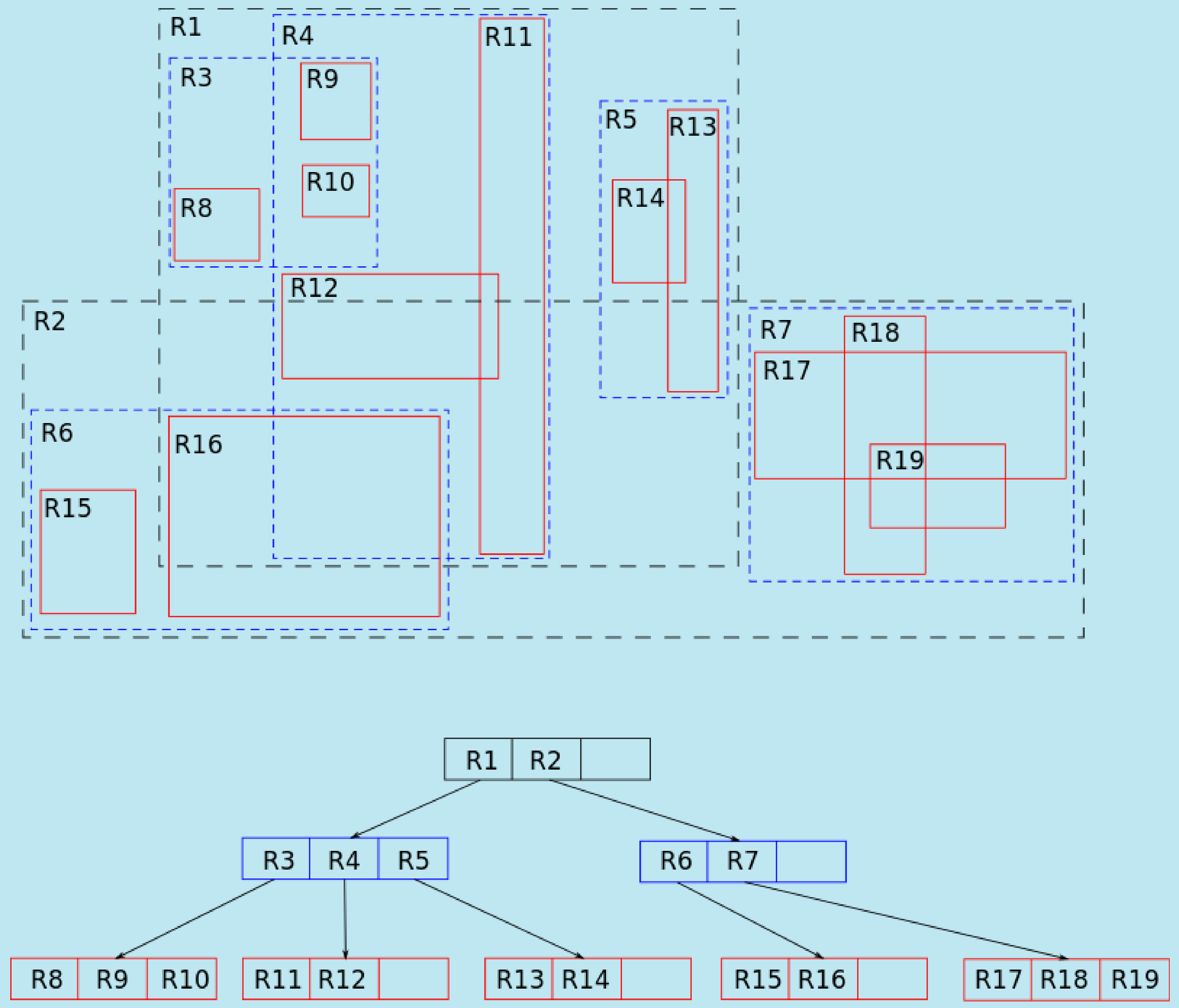
We acknowledge and understand that plagiarism is wrong.

All the codes written throughout this project have been written by us. All the work done in this project is our original work.

We have not allowed, nor will we in the future allow, anyone to copy my work with the intention of passing it off as their own work.

We also understand that if we are found to have violated this policy, we will face the consequences accordingly.

Introduction



The key idea of the data structure is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree; the "R" in R-tree is for rectangle. Since all objects lie within this bounding rectangle, a query that does not intersect the bounding rectangle also cannot intersect any of the contained objects. At the leaf level, each rectangle describes a single object; at higher levels the aggregation includes an increasing number of objects

PART 1

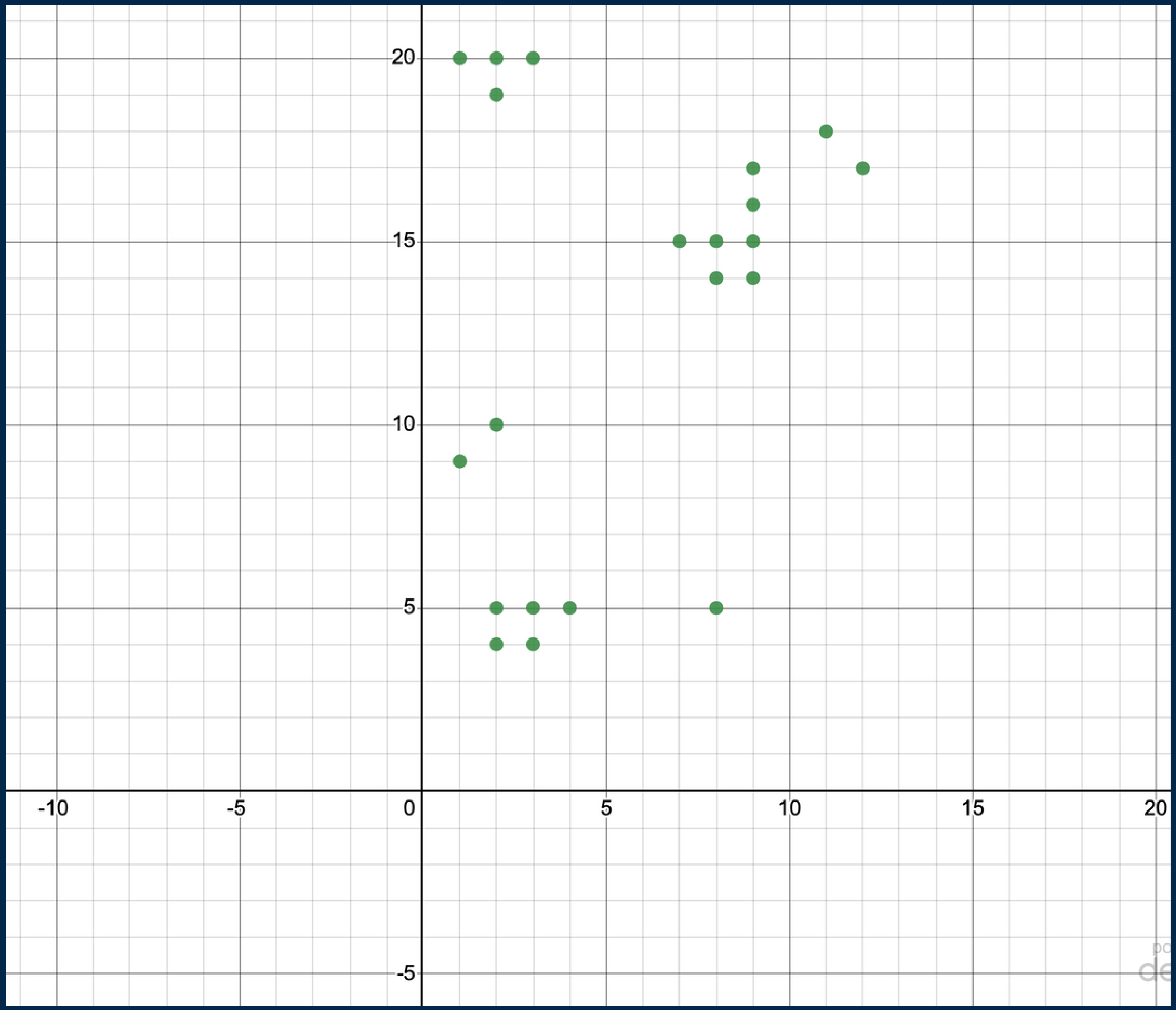
Packing Rectangles into leaf nodes

Presented by Apul Ranjan

ALGORITHM

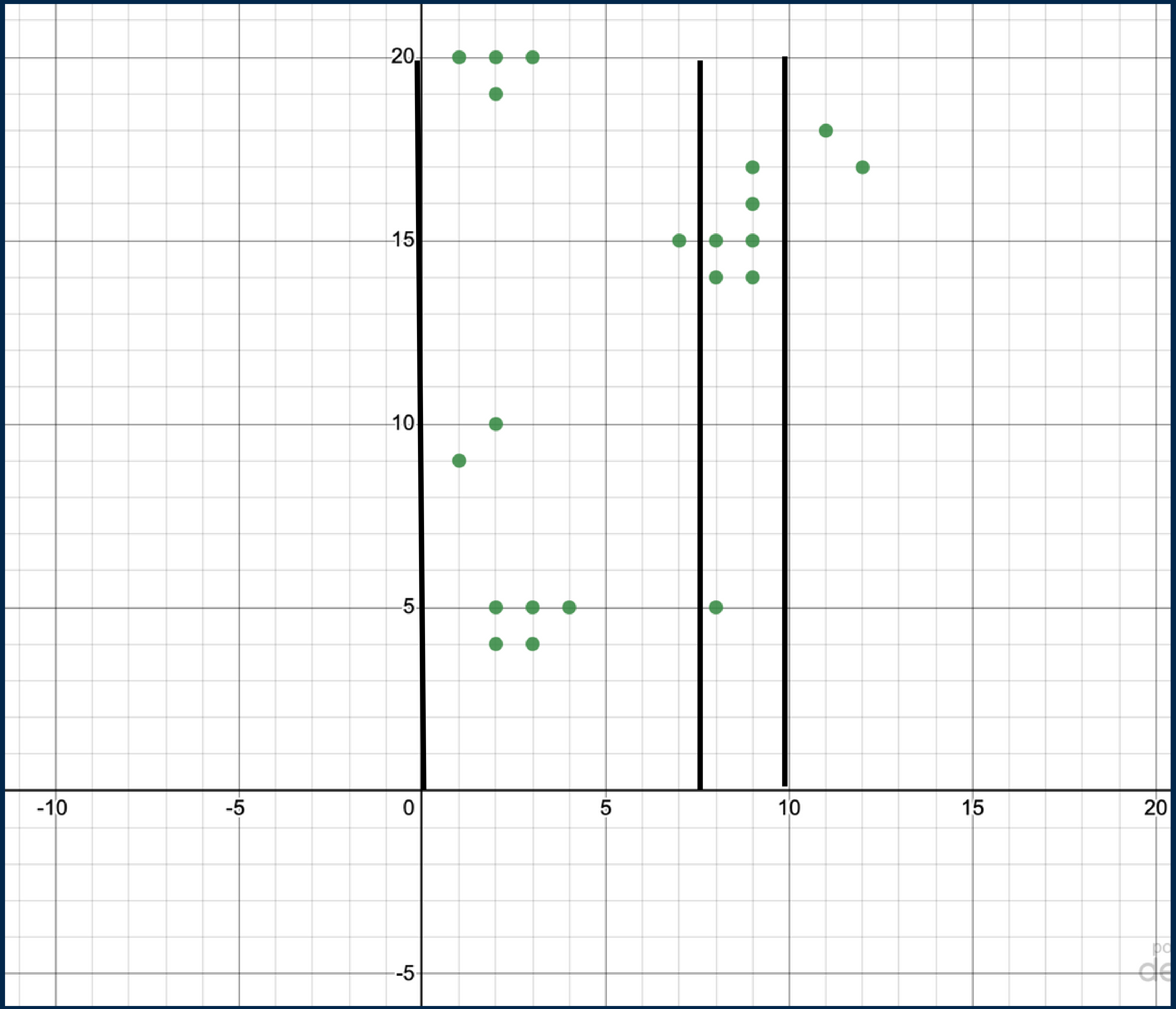
Spatial objects are in a plane and each of them is tightly enclosed in a rectangle. Assume that coordinates are for the center points of the rectangles. Sort the rectangles by x-coordinates. After that, the method tiles the data space using S = sqrt(r/M) vertical slice so that each slice contains enough rectangles to pack about sqrt(r/M) nodes. Determine the number of leaf nodes P = ceil(r / M), hence S= root(P) . Partition all sorted rectangles into S vertical slices. A slice consists of a run of S * M consecutive rectangles from the sorted list. Notice that the last slice may contain fewer than S * M rectangles but must hold at least m ones. Next, sort the rectangles of each slice by y-coordinates and pack them into the nodes by grouping them into the runs of length M (the first M rectangles into the first node, the next M into the second one, and so on).

STEP 1: SORTING DATAPOINTS BY X CO-ORDINATE (r = 21, M = 4, m = 1)

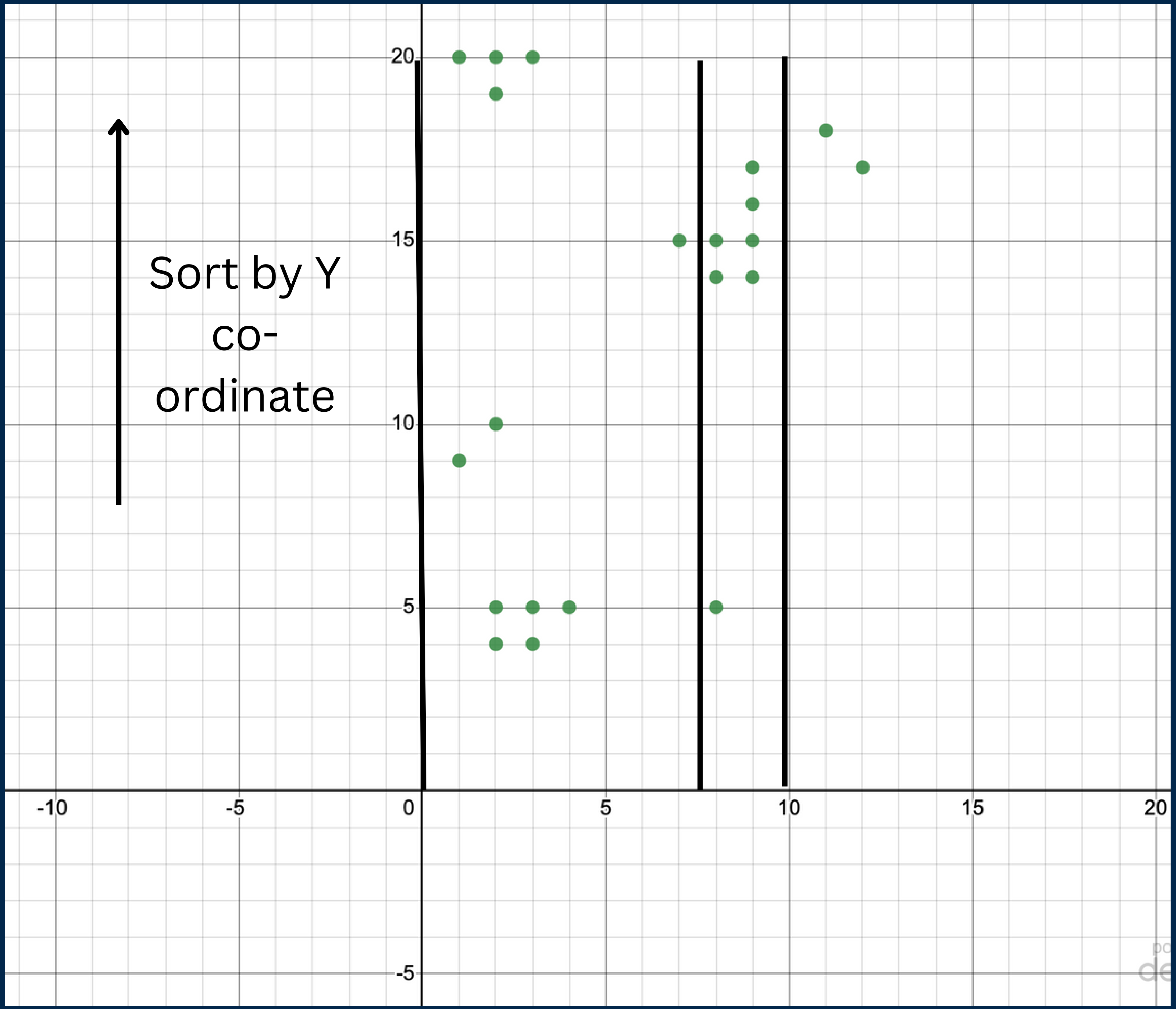


STEP 2: PUT DATAPOINTS IN SLICES

(S = 3)



STEP 3: SORT DATAPOINTS IN SLICES BY Y CO-ORDINATE



STEP 4: PACKING DATAPOINTS INTO LEAVES

(2,4)	(3,4)	(3,5)	(4,5)

NODE 1

(2,5)	(1,9)	(2,10)	(7,15)

NODE 2

(2,19)	(1,20)	(2,20)	(3,20)

NODE 3

(8,5)	(8,14)	(9,14)	(9,17)

NODE 4

(9,15)	(9,16)	(9,17)	(12,17)

NODE 5

(11,18)			

NODE 6

(P = NO OF NODES = 6)

STEP 4: PACKING DATAPOINTS INTO LEAVES

(2,4)	(3,4)	(3,5)	(4,5)

NODE 1

(2,5)	(1,9)	(2,10)	(7,15)

NODE 2

(2,19)	(1,20)	(2,20)	(3,20)

NODE 3

(8,5)	(8,14)	(9,14)	(9,17)

NODE 4

(9,15)	(9,16)	(9,17)	(12,17)

NODE 5

(11,18)			

NODE 6

NOTE: THE PACKING IS SUCH THAT THE LAST NODE CONTAINS ATLEAST m NODES

PART 2

Recursively Building an R Tree from Leaf Nodes

Presented by Peeyush Vatsa

Algorithm

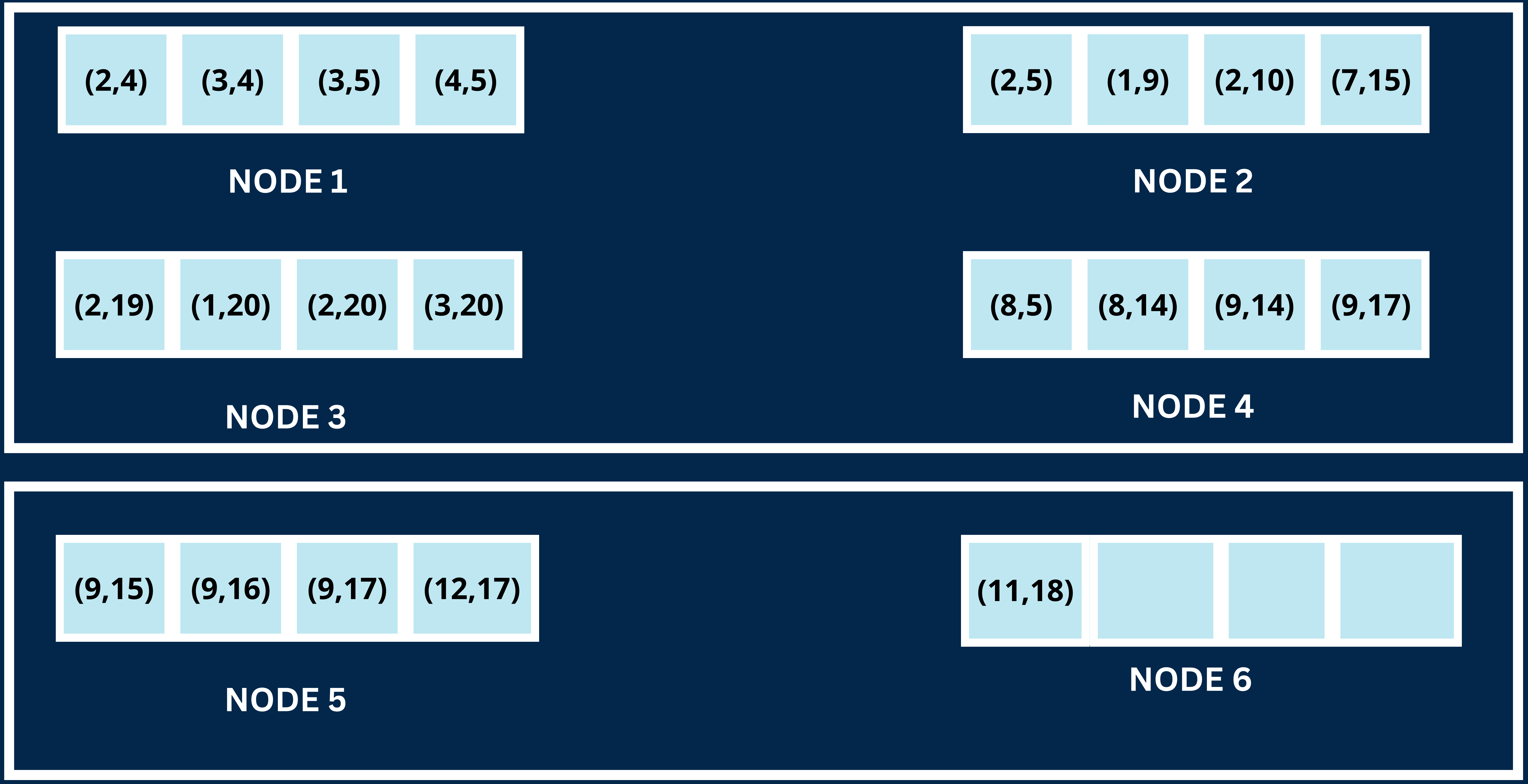
Step 1: r spatial objects are ordered in P = ceil(r / M) consecutive groups of M spatial objects, where each group of M is placed in the same leaf level node. Note that the last group may hold fewer than M hyper-rectangles but must contain at least m ones.

Step 2: From a set of consecutive leaf nodes, the process groups M successive leaf nodes into a parent node. Similar to Step 1, the last parent node may hold fewer than M leaf nodes but must contain at least m ones.

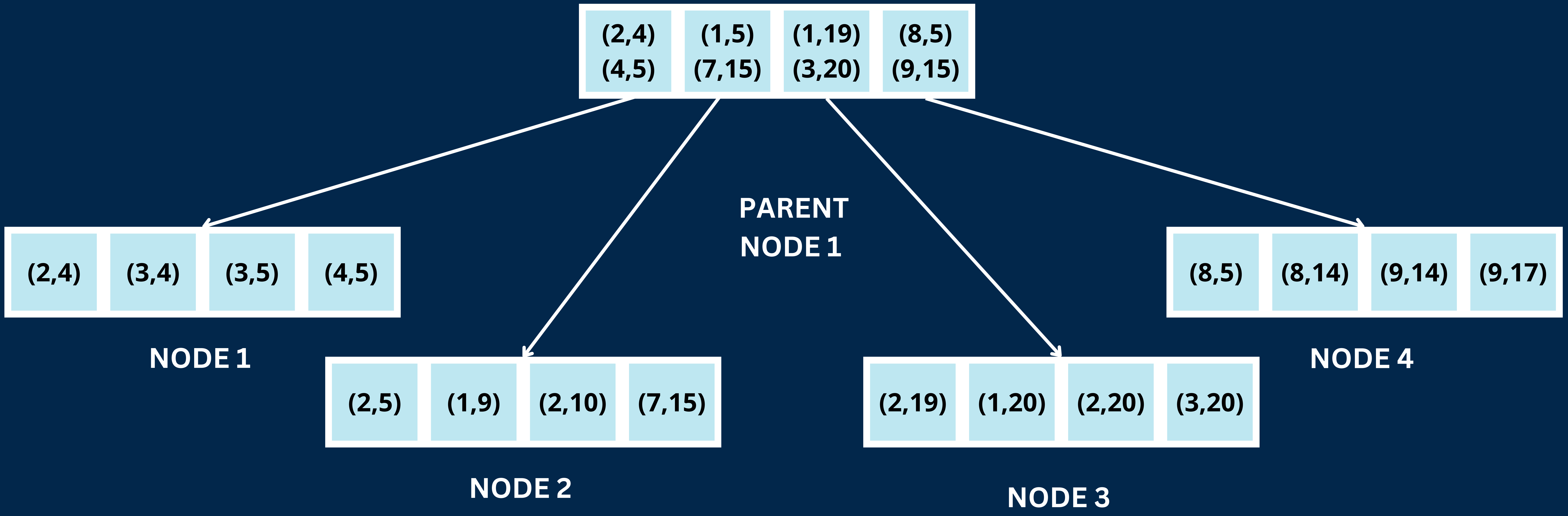
Step 3: The process recursively packs these parent nodes into the nodes at the higher level, proceeding upwards, until the root node is created.

STEP 1: NODES ARE GROUPED

Nodes are grouped together with at most M nodes in 1 group

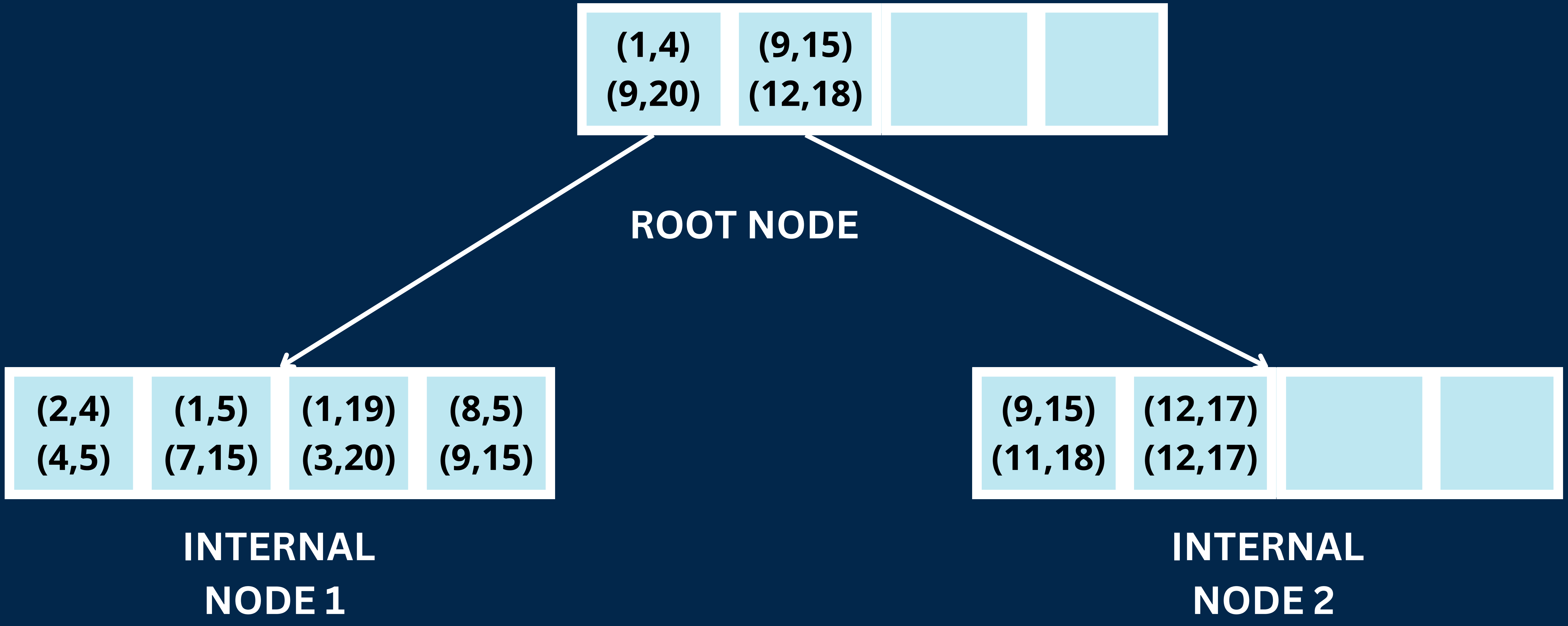


STEP 2: GROUPING M LEAF NODES INTO A PARENT NODE



STEP 3: RECURSIVELY PACKING PARENT NODES

Process is repeated till root node is created



PART 3

Searching in R Trees

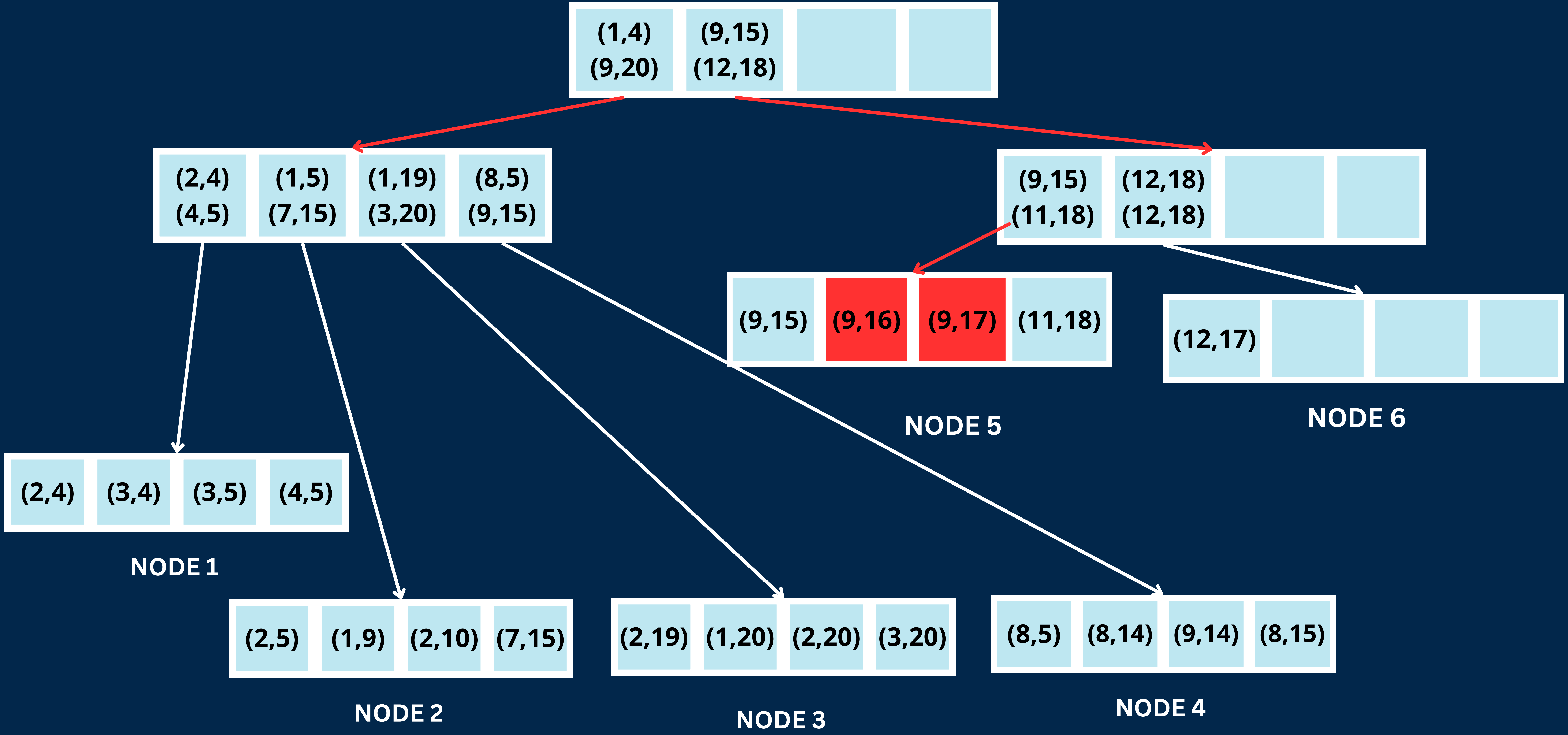
Presented by Arush Dayal

Algorithm

Assume: T is root node of an R-Tree
Goal: Find all index records whose rectangles overlap a search rectangle S.

Step 1: [Search Subtree] If T no leaf then check each entry E, whether E.I overlaps S. For all overlapping entries, start Search on the subtree (recurse step 1) whose root node is pointed to by E.p.

Step 2: [Search leaf node] If T is a leaf, then check each entry E wheather E.I overlaps S. If so, E is a suitable entry.



Consider a search query: (8,16), (9,19)

PART 4

Pre-Order Traversal of R-Tree
Presented by Nachiket

Algorithm

Step 1 : Check if the input node is NULL, and if so, return from the function.

Step 2 : If the input node is a leaf node, print out the number of entries in the node and details of each entry.

Step 3 : If the input node is not a leaf node, compute the minimum bounding rectangle (MBR) for the node, and print out the coordinates of the top right and bottom left points of the MBR.

Step 4 : Recursively traverse each child of the input node by calling the preorderTraverse function on each child node in the order they appear in the children array.

Part 5

Insertion of point in R-Tree (after STR)
Presented by Chandrashekar

How does insertion work?

1. Minimum difference and Minimum area algorithm (MDA) - according to the Guttman's paper, to insert a point the point is inserted in the leaf nodes. If it not a leaf node - then, the point goes to insert in the child node which results in minimum area expansion of MBR node and if the area expanded by two nodes are same then it inserts into the node having the minimum MBR area.
2. The algorithm has two parts insert and adjust - insert inserts the point of traverses the R-tree to find the appropriate node to go into. The adjust function is called when at any point after insertion of the point in the leaf node the node exceeds it's maximum capacity.
3. The adjust function splits the node into two and groups - by choosing largest and second largest in two different nodes and grouping others acoodring to MDA.
4. This algorithm is quadratic cost.

Algorithm

Step 1 : Start at the root node and recursively descend the tree until a leaf node is reached that has space for the new entry.

Step 2 : Insert the new entry into the leaf node.

Step 3 : If the leaf node is now overfull, split it into two smaller nodes along a split axis that minimizes the total overlap between the entries.

Step 4 : Create a new non-leaf node to hold the two smaller nodes, and adjust the parent-child relationships to reflect the new structure of the tree.

Step 5 : If the parent node is now overfull, recursively split it in the same way as the leaf node.

Step 6 : Repeat steps 3-5 until the root node is reached. If the root node is overfull, create a new root node and adjust the parent-child relationships accordingly.

Thank you