Organizing Spatial Data

Spatial data records include a sense of location as an attribute.

Typically location is represented by coordinate data (in 2D or 3D).

If we are to search spatial data using the locations as key values, we need data structures that efficiently represent selecting among more than two alternatives during a search.

One approach for 2D data is to employ quadtrees, in which each internal node can have up to four children, each representing a different region obtained by decomposing the coordinate space.

There are a variety of such quadtrees, many of which are described in:

The Quadtree and Related Hierarchical Data Structures, Hanan Samet

ACM Computing Surveys, June 1984

Spatial Decomposition

In binary search trees, the structure of the tree depends not only upon what data values are inserted, but also in what order they are inserted.

In contrast, the structure of a Point-Region quadtree is determined entirely by the data values it contains, and is independent of the order of their insertion.

In effect, each node of a PR quadtree represents a particular region in a 2D coordinate space.

Internal nodes have exactly 4 children (some may be empty), each representing a different, congruent quadrant of the region represented by their parent node.

Internal nodes do not store data.

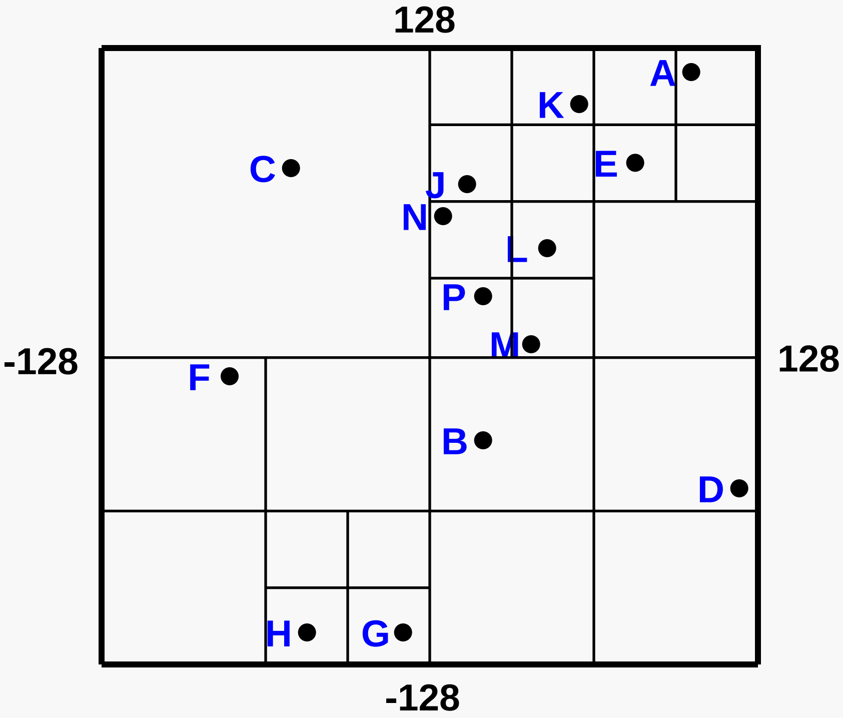
Leaf nodes hold a single data value. Therefore, the coordinate space is partitioned as insertions are performed so that no region contains more than a single point.

PR quadtrees represent points in a finitely-bounded coordinate space.

Coordinate Space Partitioning

Consider the collection of points in a 256 x 256 coordinate space:

A	(100, 125)
B	(25, -30)
C	(-55, 80)
D	(125, -60)
E	(80, 80)
F	(-80, -8)
G	(-12, -112)
H	(-48, -112)
J	(16, 72)
K	(60, 100)
L	(48, 48)
M	(36, 8)
N	(4, 60)
P	(28, 30)



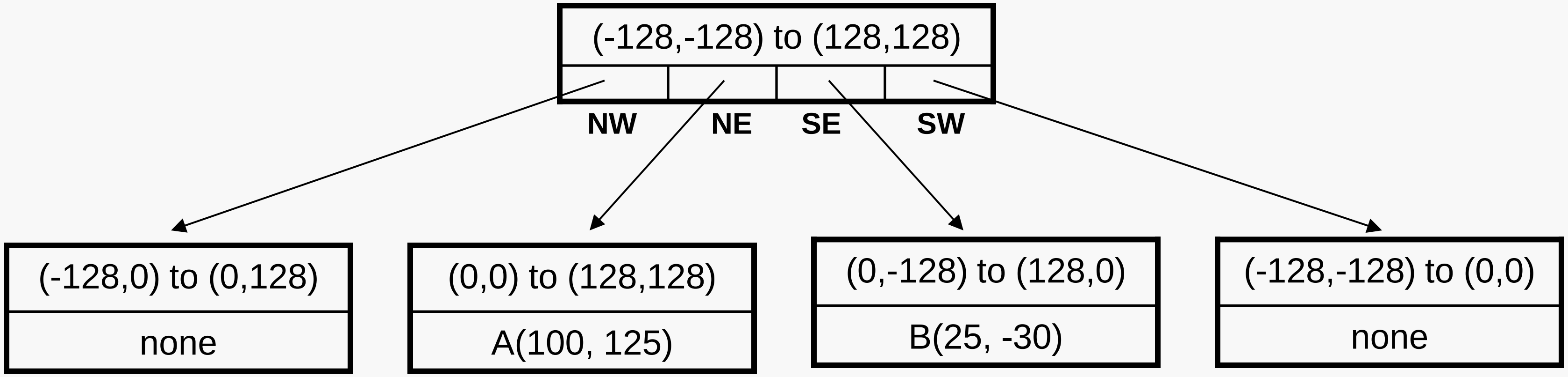
The subdivision of the coordinate space shows how it will be partitioned as the points are added to a PR quadtree.

PR Quadtree Insertion

A	**(100, 125)**
B	**(25, -30)**

Obviously inserting the first point, A, just results in the creation of a leaf node holding A.

Inserting B causes the partitioning of the original coordinate space into four quadrants, and the replacement of the root with an internal node with two nonempty children:



(-128,-128) to (128,128)

NW **NE** **SE** **SW**

(-128,0) to (0,128)
none

(0,0) to (128,128)
A(100, 125)

(0,-128) to (128,0)
B(25, -30)

(-128,-128) to (0,0)
none

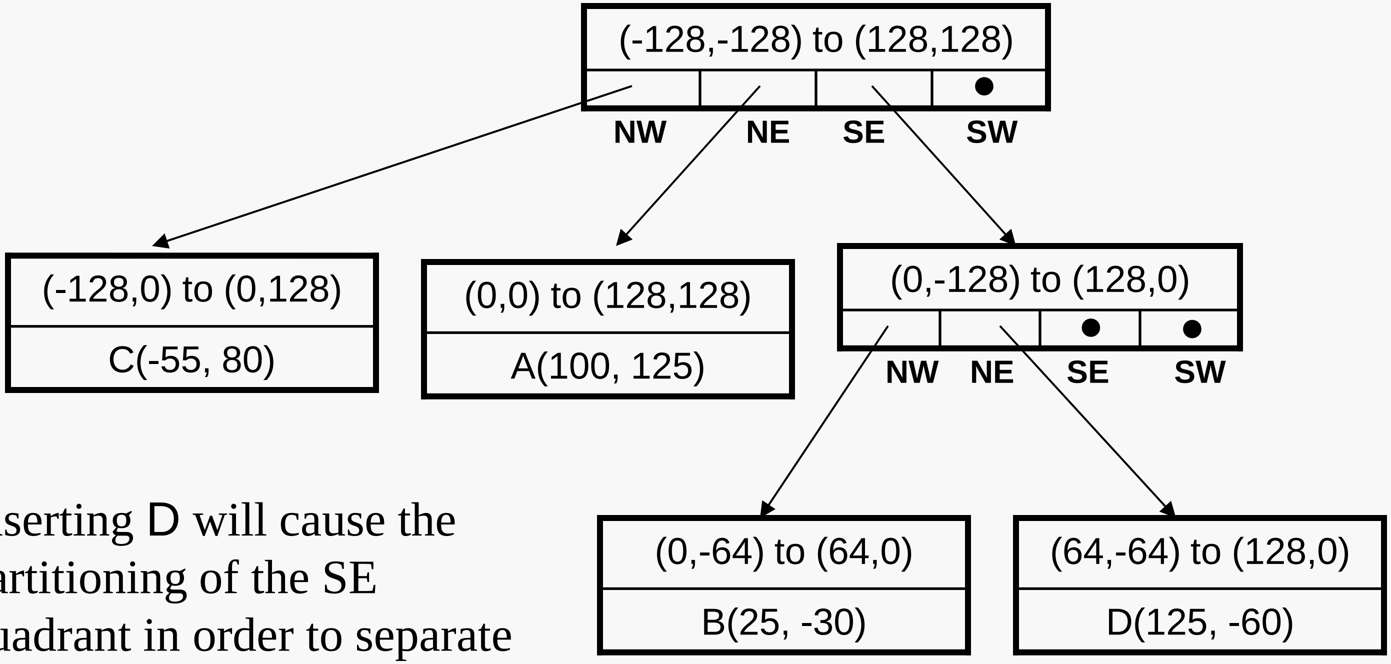
The display above shows the SW and NE corners of the regions logically represented by each node, and the data values stored in the leaf nodes.

In an implementation, nodes would not store information defining their regions explicitly, nor would empty leaf nodes probably be allocated.

Leaf Splitting During Insertion

Inserting C does not cause any additional partitioning of the coordinate space since it naturally falls into an empty leaf:

A	(100, 125)
B	(25, -30)
C	(-55, 80)
D	(125, -60)

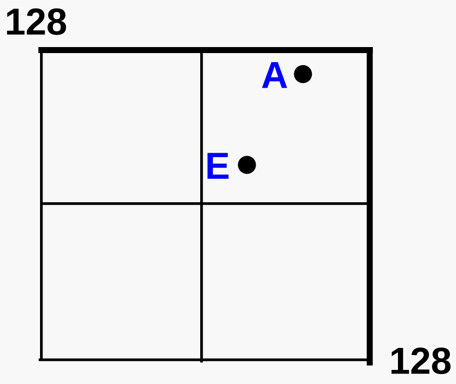


Inserting D will cause the partitioning of the SE quadrant in order to separate B and D:

Multiple Splitting

Suppose the value E(80, 80) is now inserted into the tree. It falls in the same region as the point A, (0, 0) to (128, 128).

However, dividing that region creates three empty regions, and the region (64, 64) to (128, 128) in which both A and E lie.



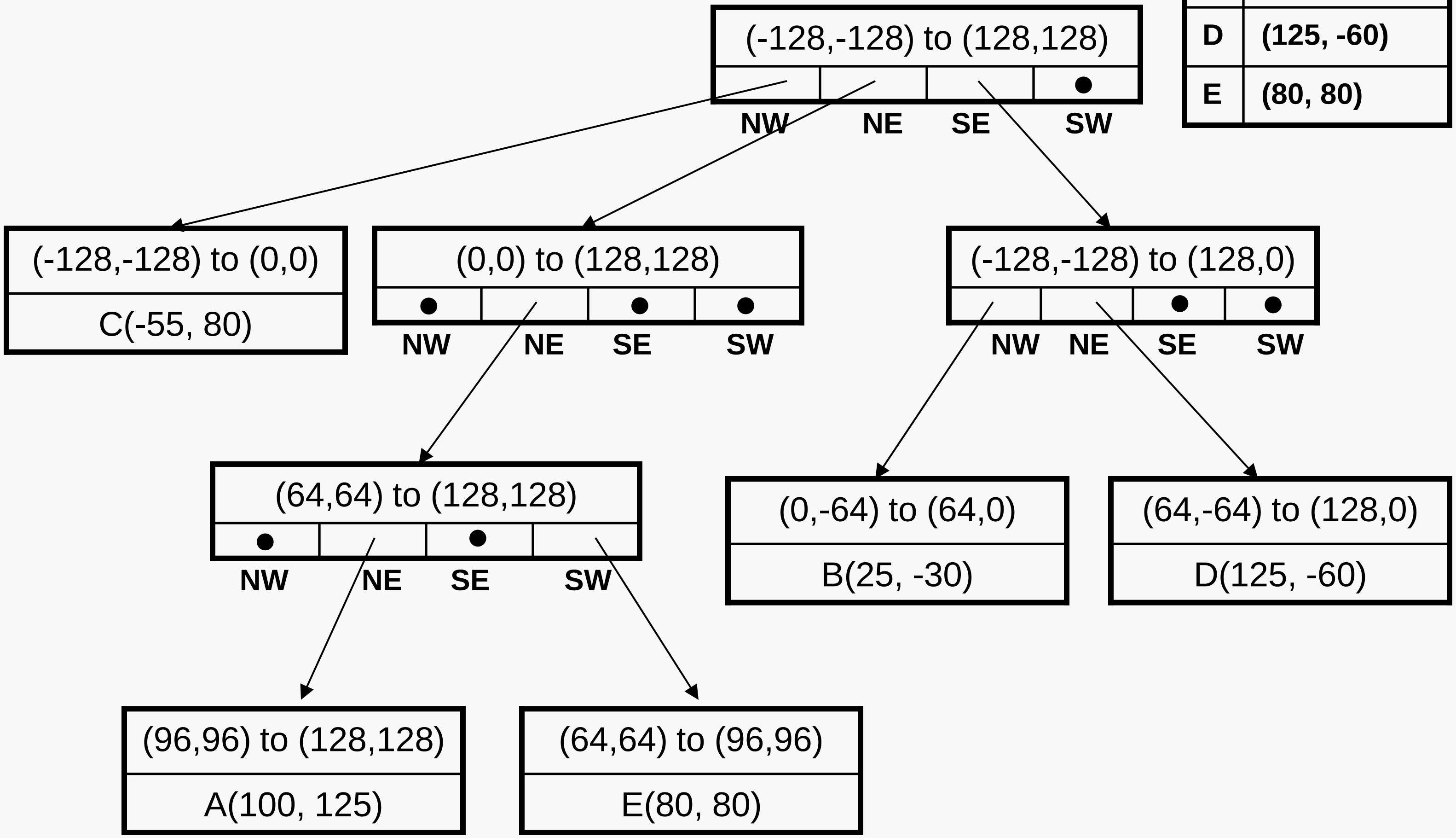
So, that region must be partitioned again. This separates A and E into two separate regions (see illustration on the slide "Coordinate Space Partitioning").

If it had not, then the region in which they both occurred would be partitioned again, and again if necessary, until they are separated.

Multiple Splitting

Inserting E results in the following tree:

A	**(100, 125)**
B	**(25, -30)**
C	**(-55, 80)**
D	**(125, -60)**
E	**(80, 80)**



PR Quadtree Insertion

Insertion proceeds recursively, descending until the appropriate leaf node (possibly empty) is found, and then partitioning and descending until there is no more than one point within the region represented by each leaf.

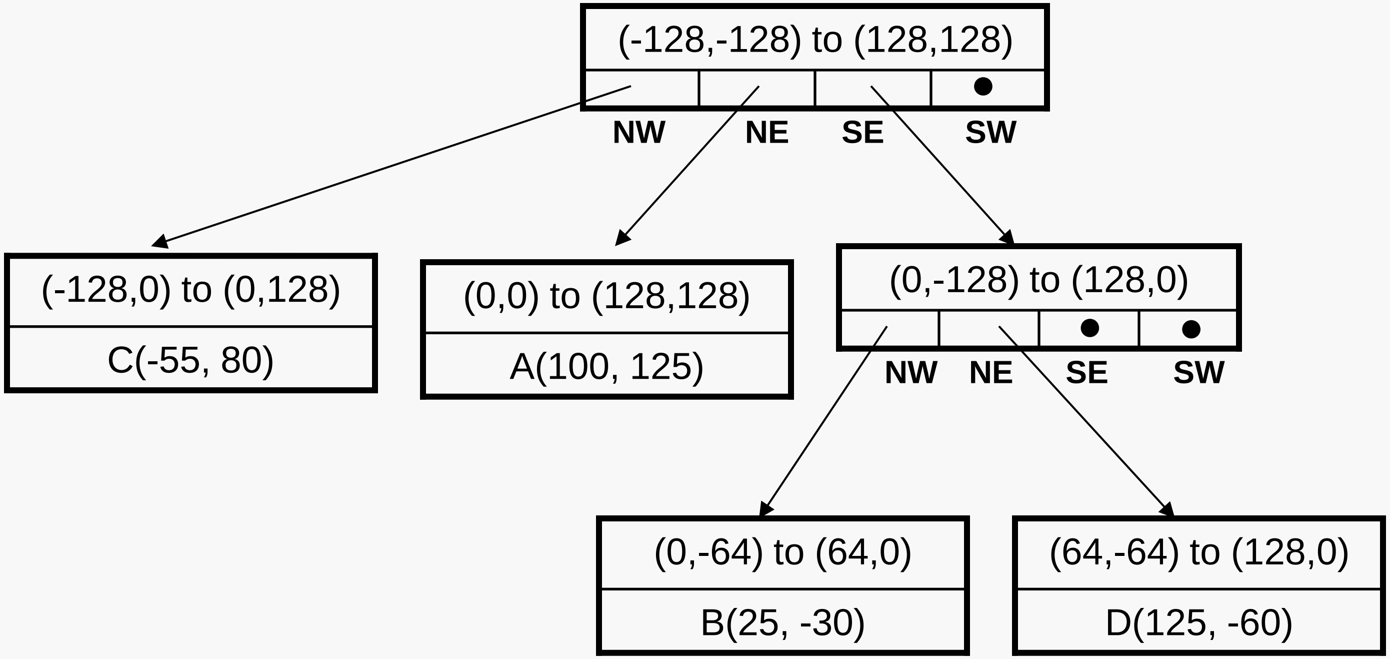
It is possible for a single insertion to add many levels to the relevant subtree, if points lie close enough together.

Of course, it is also possible for an insertion to require no splitting whatsoever.

The shape of the tree is entirely independent of the order in which the data elements are added to it.

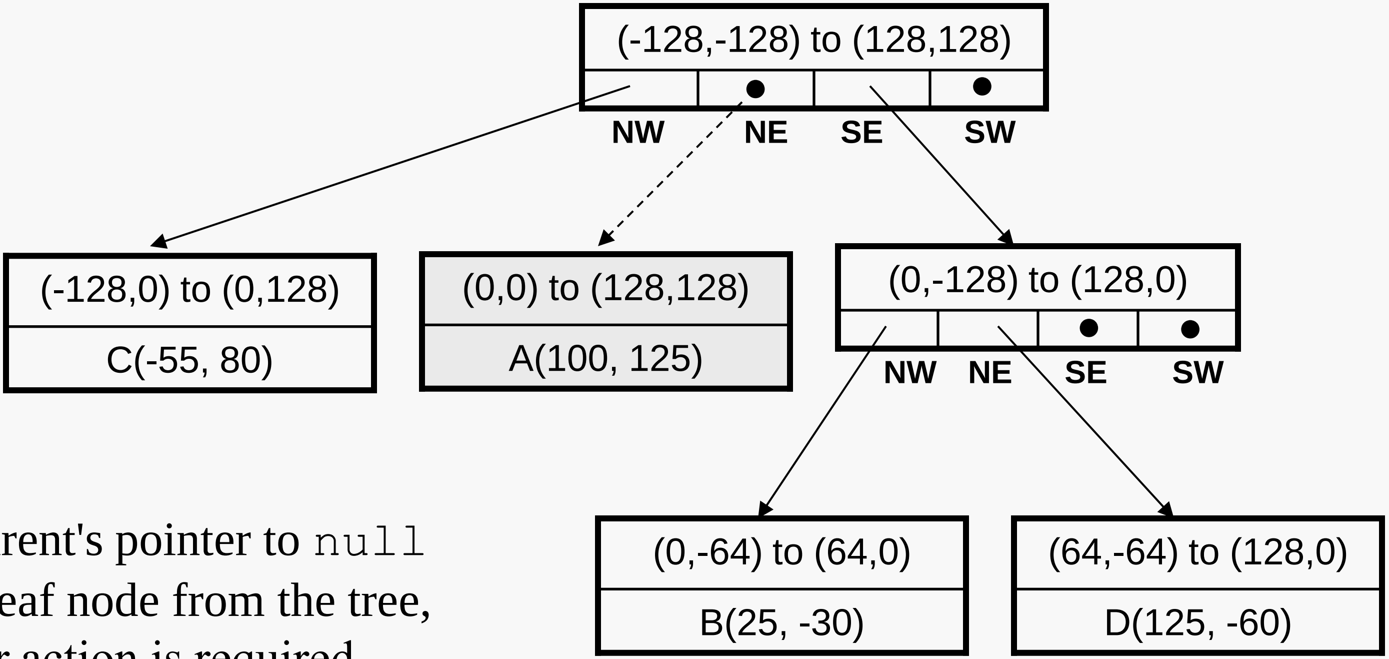
PR Quadtree Deletion

Deletion always involves removing a leaf node. Consider deleting A from the following tree:



PR Quadtree Deletion

Deletion always involves removing a leaf node. Consider deleting A from the following tree:

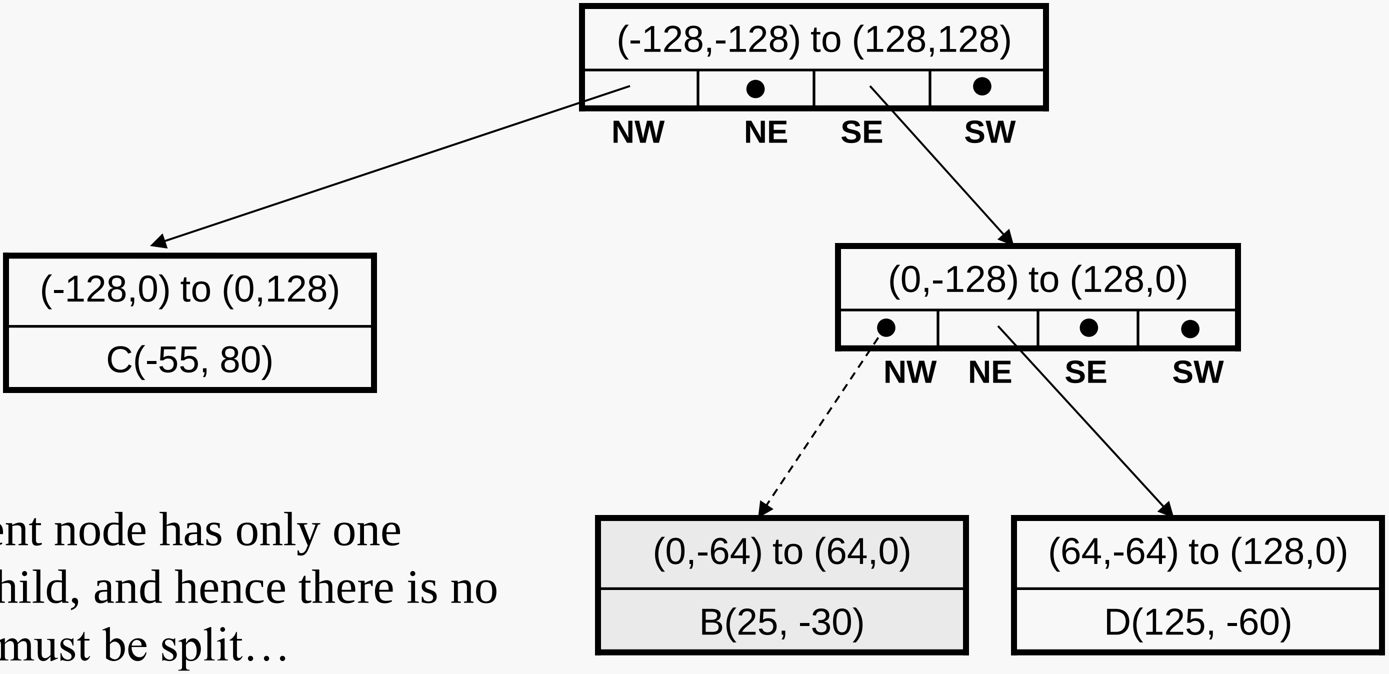


Setting the parent's pointer to `null` removes the leaf node from the tree, and no further action is required…

PR Quadtree Deletion

On the other hand, deleting a leaf node may cause its parent to "underflow":

Consider deleting B:

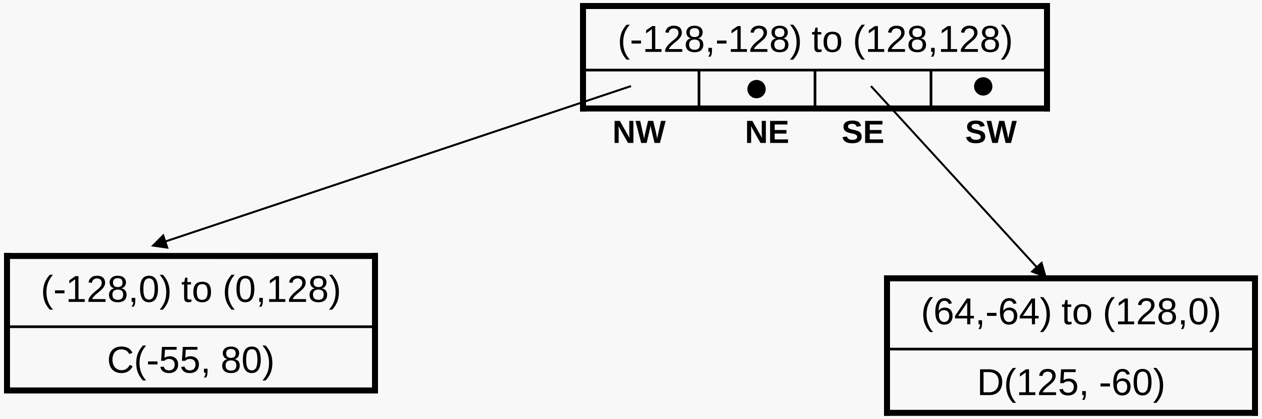


Now, the parent node has only one (nonempty) child, and hence there is no reason that it must be split…

… so, we can "contract" the branch by replacing the parent (internal) node with the remaining child (leaf)…

PR Quadtree Deletion

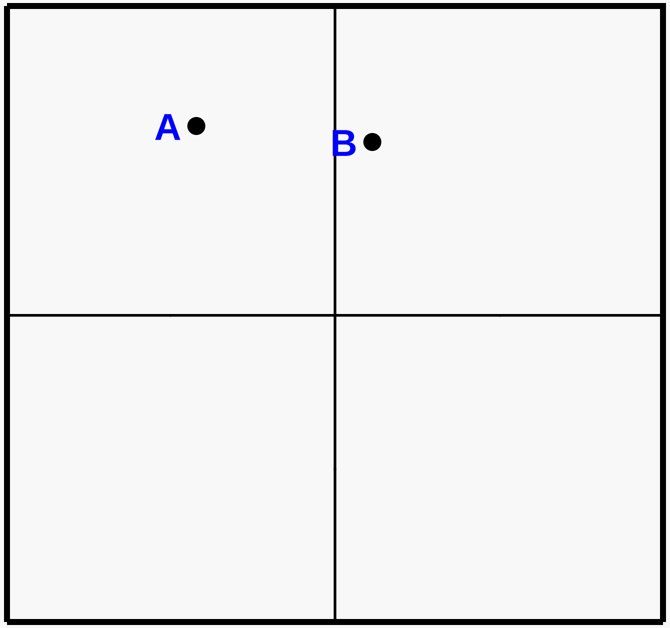
Contracting the branch results in:

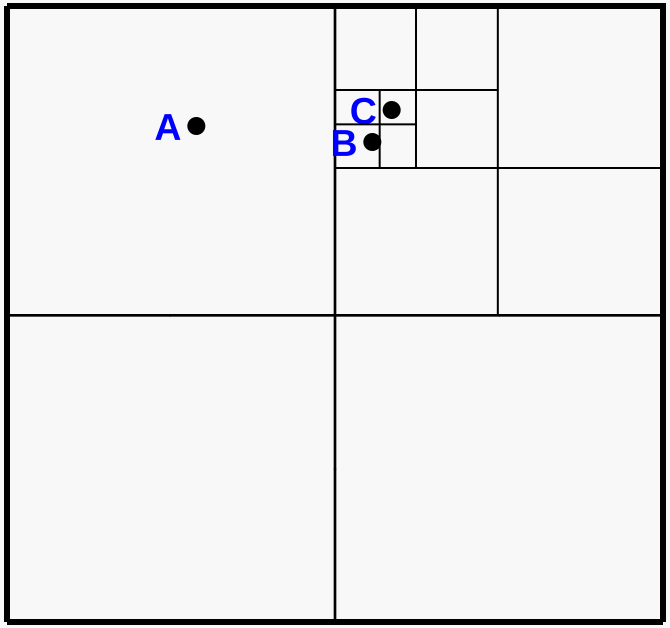


Of course, if the root node of this tree did not have another child, then the branch contraction could continue…

PR Quadtree Achilles' Heel

If a data point that is inserted lies very close to another data point in the tree, it is possible that many levels of partitioning will be required in order to separate them.





The minimum height of a PR quadtree is can be as large as $\left\lceil \log\left(\frac{\sqrt{2}s}{d}\right)\right\rceil$

where *s* is the length of a side of the "world" and *d* is the minimum distance between any two data points in the tree.

PR Quadtree Using Buckets

The problem of "stalky" PR quadtree branches can be alleviated by allowing each leaf node to store more than one data object, making the leaf a "bucket".

For example, if the quadtree leaf can store 5 data elements then it does not have to split until we have 6 data points that fall within its region.

This complicates the implementation of the tree slightly, but can substantially reduce the cost of search operations.

If buckets are used, then the minimum height of a PR quadtree is can be as large as

$$\left\lceil \log\left(\frac{\sqrt{2}s}{d}\right)\right\rceil$$

where *s* is the length of a side of the "world" and *d* is the minimum side of a square that contains more data elements than a bucket can hold.