HIERARCHICAL REPRESENTATIONS OF RECTANGLE DATA

Hanan Samet

Computer Science Department and
Center for Automation Research and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
e-mail: hjs@umiacs.umd.edu

RECTANGLES

- Applications
  1. geographic data
     - can approximate lakes, forests, hills, etc. by minimum enclosing rectangle
     - number of elements is usually small
     - size of collection is *O* (size of space) being represented
  2. VLSI design rule checking
     - determine if components intersect
     - ensure satisfaction of constraints (e.g., minimum separation widths, etc.)
     - size of the collection is large (on the order of millions)
     - rectangles are small — i.e., much smaller than the *O* (size of space) from which they are drawn
- Choosing a representation
  1. individual objects
  2. collection of objects
     - linked lists
     - similar to representations used for collections of point data
       a. organize data to be stored (e.g., comparative search)

          OR

       b. organize embedding space from which the data is drawn (e.g., address computation)

OPERATIONS ON RECTANGLES

1. Insertion
2. Deletion
3. Proximity queries
   - distinguish between objects and points

     point ≡ element in *d*-dimensional space from which the objects are drawn and NOT an element of the space into which the objects are mapped by the representation
   - Ex: a rectangle can be represented as a point in 4-d space yet a point in it is an element of 2-d space
   - operations:
     a. point query — find all objects that overlap a given point
     b. point query set — find all objects that overlap a given set of points (e.g., rectangle, square, circle, ?)
        - a window operation
     c. geometric join
        - relation ⊕ — e.g., intersection
        - classes of objects with subsets S_1, S_2 — e.g., rectangles
        - find all (P_1, P_2) such that $P_1 \in S_1$, $P_2 \in S_2$ and $P_1 \oplus P_2$

REPRESENTATION BY CHARACTERISTIC PARTS

1. Based on interior (e.g., region quadtree)
   - decompose into subunits where each one points to a complete description of the object
2. Based on the boundary of the object
   - polygons as ordered collections of vertices
   - polygons as collections of line segments comprising their boundaries
3. Procedural — combination of (a) and (b) by decomposition into units of smaller dimension
   - rectangle as the Cartesian product of two 1-d spheres (e.g., intervals)

Problem: sometimes a query is specified in such a way that none of the characteristic parts of an object, say O, that satisfies the query will match the query's description; yet object O does not match the query

Ex: polygon P is represented by the line segments comprising its edges and we want to know if rectangle R is within P

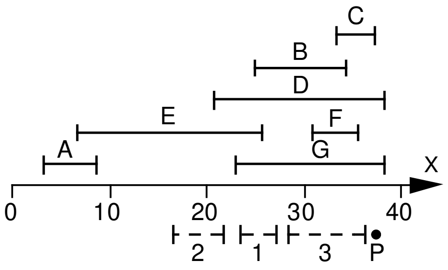
Solution: store the identity of the polygon or objects associated with each side of an edge

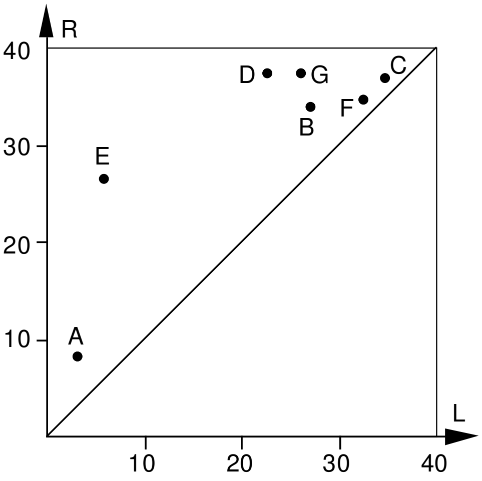
POINT-BASED RECTANGLE REPRESENTATIONS

- Parameterize the rectangles and represent the parameters as a point (termed a *representative point*) in a higher dimensional space
- Store points using data structures for multidimensional point data (e.g., k-d trees, grid file, EXCELL, etc.)
- Can represent as the Cartesian product of two one-dimensional intervals (Hinrichs and Nievergelt) where each interval is a point in a two-dimensional space

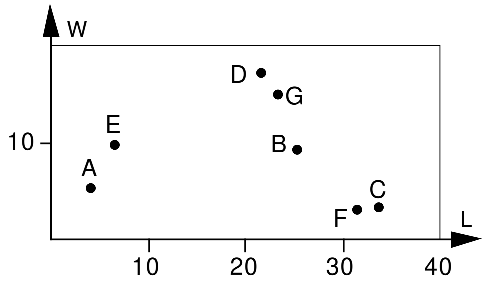
POSSIBLE INTERVAL REPRESENTATIONS

1. left (*L*) and right (*R*) endpoints of interval
   - drawback is a resulting clustering of the representative points near and above the diagonal since $L < R$

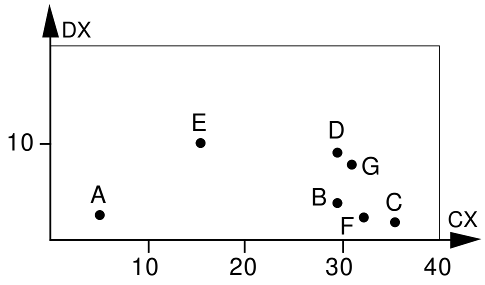




2. left endpoint of interval and width



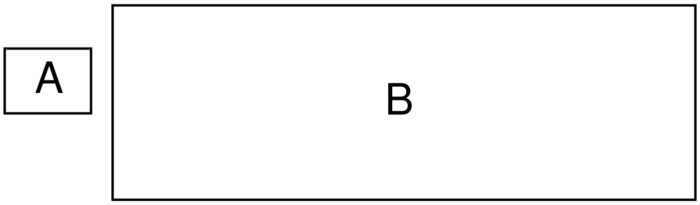
3. center of interval and radius



- Items satisfying proximity queries lie in cone-shaped regions

HIGHER-DIMENSIONAL POINT-BASED RECTANGLE REPRESENTATIONS

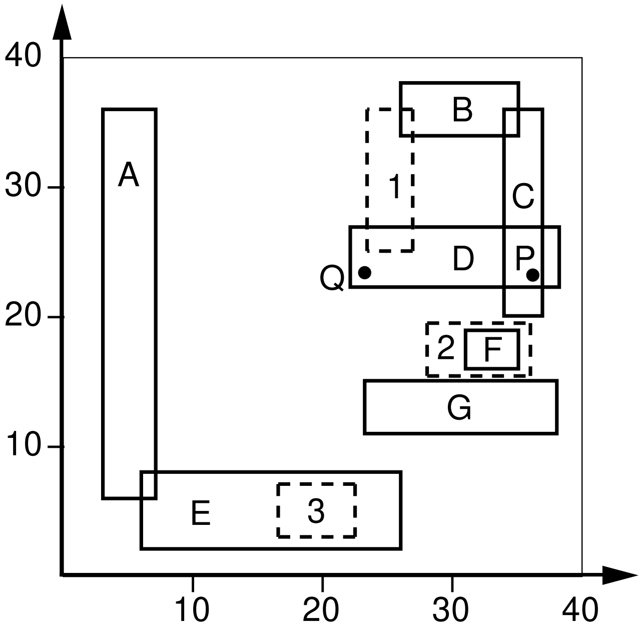
- Use a point in a four-dimensional space
- Sample representations:
  1. *x*, *y* coordinate values of diagonally opposite corners
  2. location of corner plus horizontal and vertical extents
  3. location of centroid plus half of each of the horizontal and vertical extents
- Drawback: the representative points of two rectangles that are physically close to each other in 2-d space may be very far from each other in the higher dimensional space
- Ex:



POINT QUERY

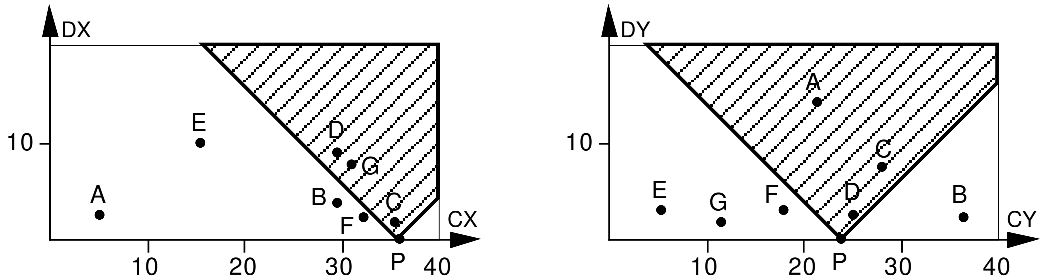
- Use Cartesian product of two one-dimensional intervals
- Represent interval by its center and radius
- Determine all intervals that contain a given point P

Ex: Point P



- Intervals have cone-shaped regions whose tip is an interval of width 0 centered at P

Horizontal Intervals = {C,D,G}   Vertical Intervals = {A,C,D}

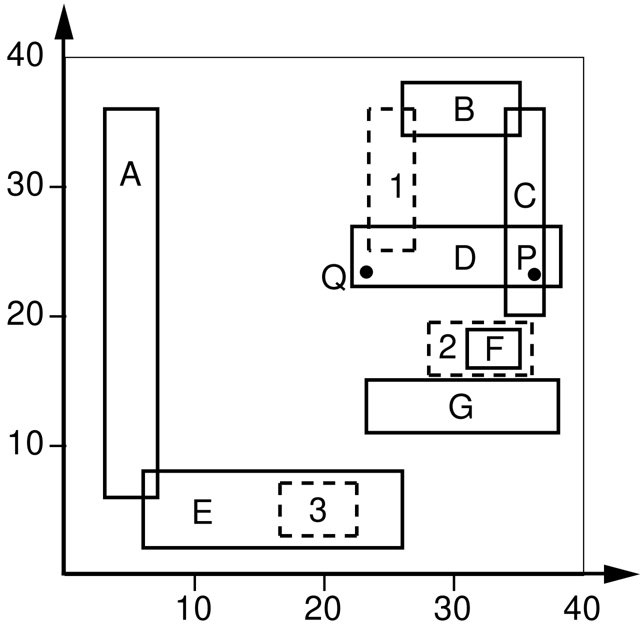


- Result is obtained by intersecting the contents of the cone-shaped regions to yield {C,D}

WINDOW QUERY

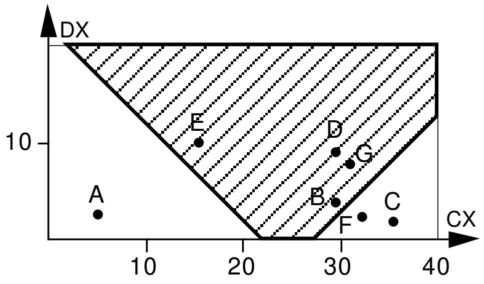
- Use Cartesian product of two one-dimensional intervals
- Represent interval by its center and radius
- Determine all intervals that overlap a given interval

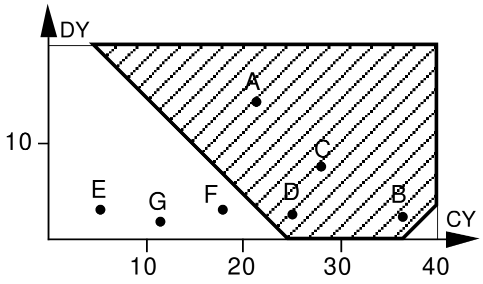
Ex: Rectangle 1



- Like an infinite number of point queries centered at the points comprising the query interval

  Horizontal Intervals = {B,D,E,G}   Vertical Intervals = {A,B,C,D}



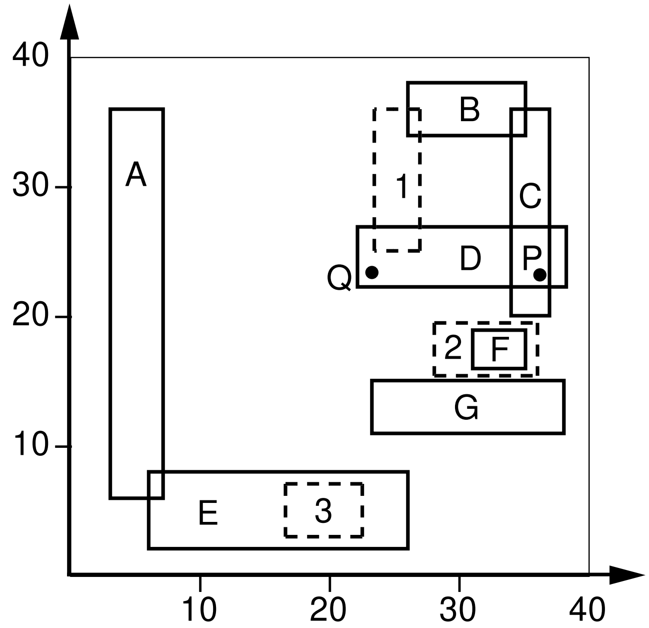


- Result is obtained by intersecting the contents of the cone-shaped regions to yield {B,D}

CONTAINMENT QUERY

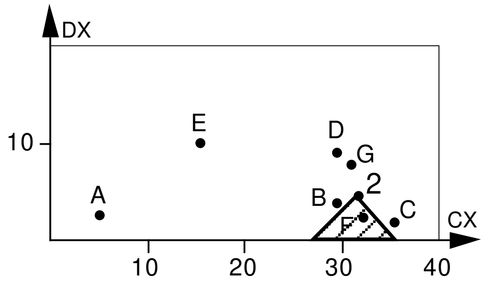
- Use Cartesian product of two one-dimensional intervals
- Represent interval by its center and radius
- Determine all intervals totally contained within a given interval

Ex: Rectangle 2

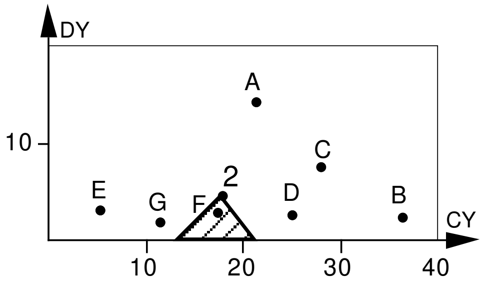


- The contained intervals form a cone-shaped region with a tip at 2 and opening in the direction of smaller extent values

Horizontal Intervals = {F}



Vertical Intervals = {F}

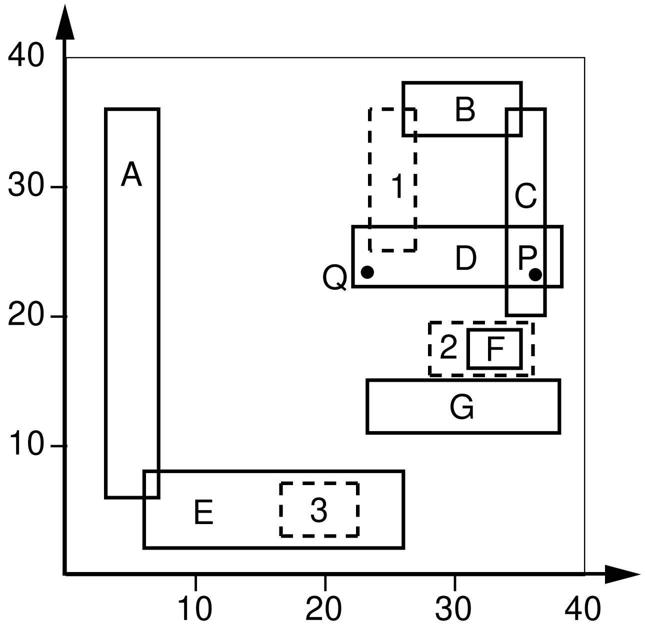


- Result is obtained by intersecting the contents of the cone-shaped regions to yield {F}

ENCLOSURE QUERY

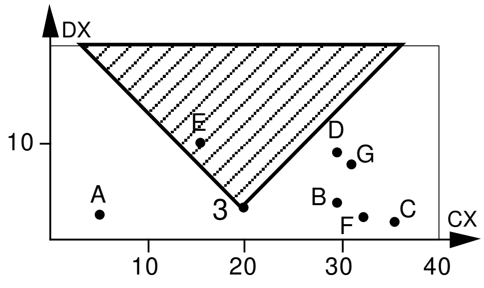
- Use Cartesian product of two one-dimensional intervals
- Represent interval by its center and radius
- Determine all intervals that enclose a given interval

Ex: Rectangle 3

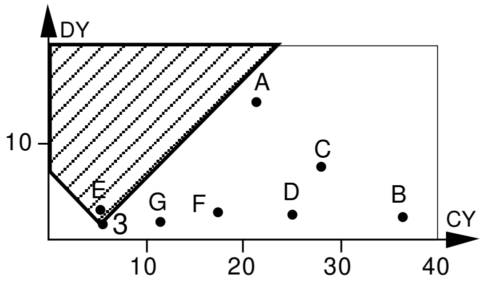


- The set of enclosing intervals form a cone-shaped region with a tip at 3 and opening in the direction of larger extent values

Horizontal Intervals = {E}



Vertical Intervals = {E}



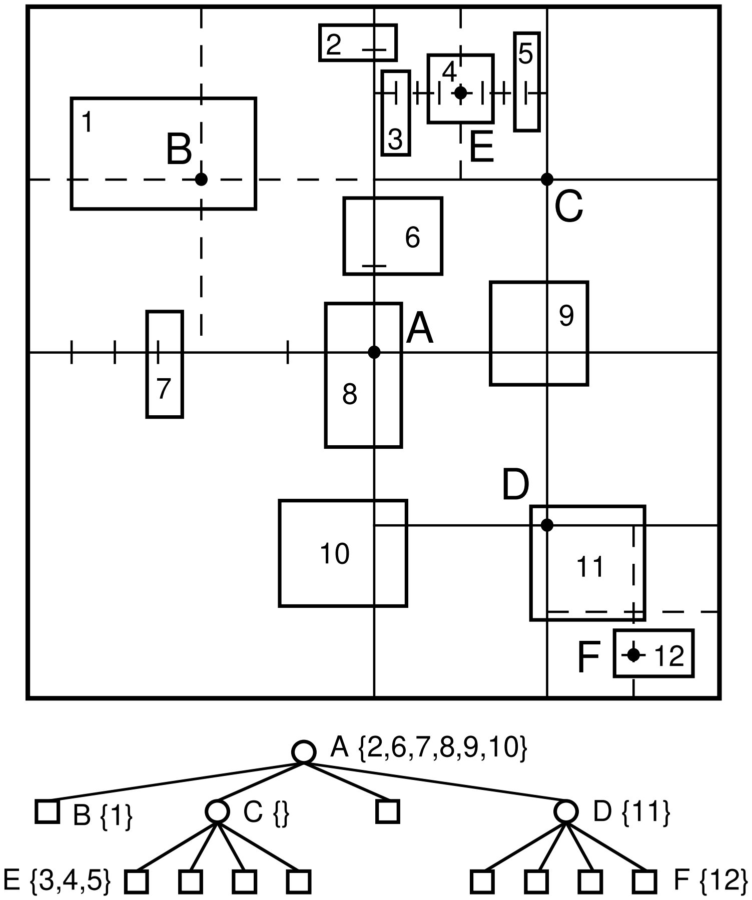
- Result is obtained by intersecting the contents of the cone-shaped regions to yield {E}

AREA-BASED METHODS

- Partition space into cells (i.e., buckets)
  1. contain pointers to rectangles that intersect them
  2. can be disjoint or allowed to overlap
     - even if allow overlap, as long as there is always at least one cell that contains an object in its entirety, then no problem with redundancy due to multiple references to the object
     - when cells overlap, the cost of query operations increases because several cells can cover a query point
- Hierarchy of minimum bounding boxes
  1. MX-CIF quadtrees
  2. R-trees
- Decompose rectangle into characteristic parts
  1. based on the interior of the rectangles
     - R^+-tree
     - rectangle quadtree (based on region quadtree — i.e., decompose until each block is completely within or completely outside a rectangle)
  2. based on the boundary of the rectangles
     - PMR quadtree — i.e., treat the boundaries of the rectangles as line segments

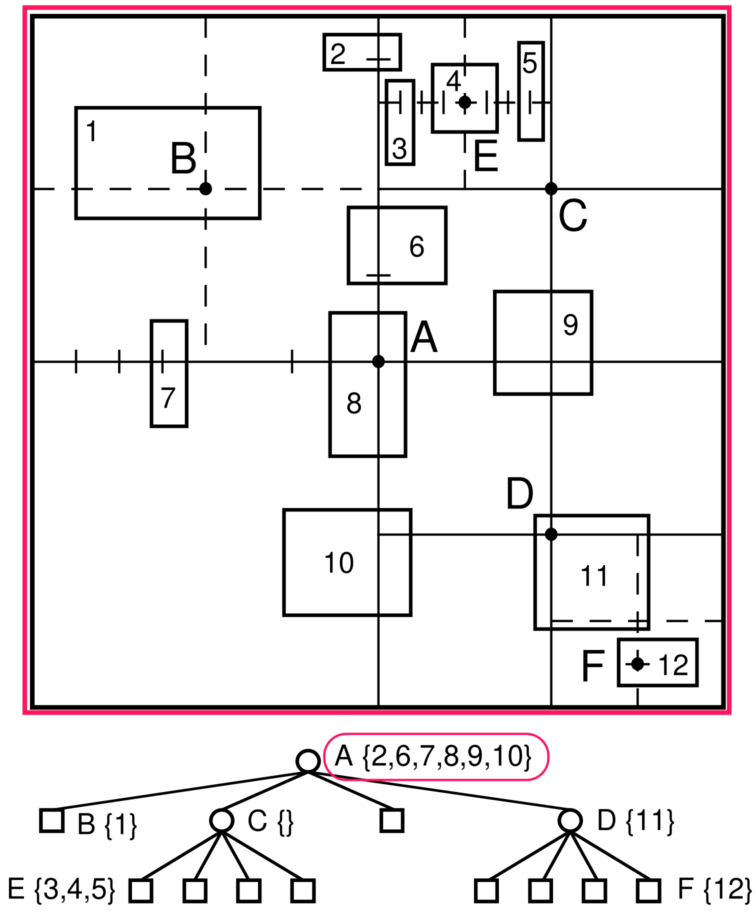
MX-CIF QUADTREE (Kedem)

1. Collections of small rectangles for VLSI applications
2. Each rectangle is associated with its minimum enclosing quadtree block
3. Like hashing: quadtree blocks serve as hash buckets



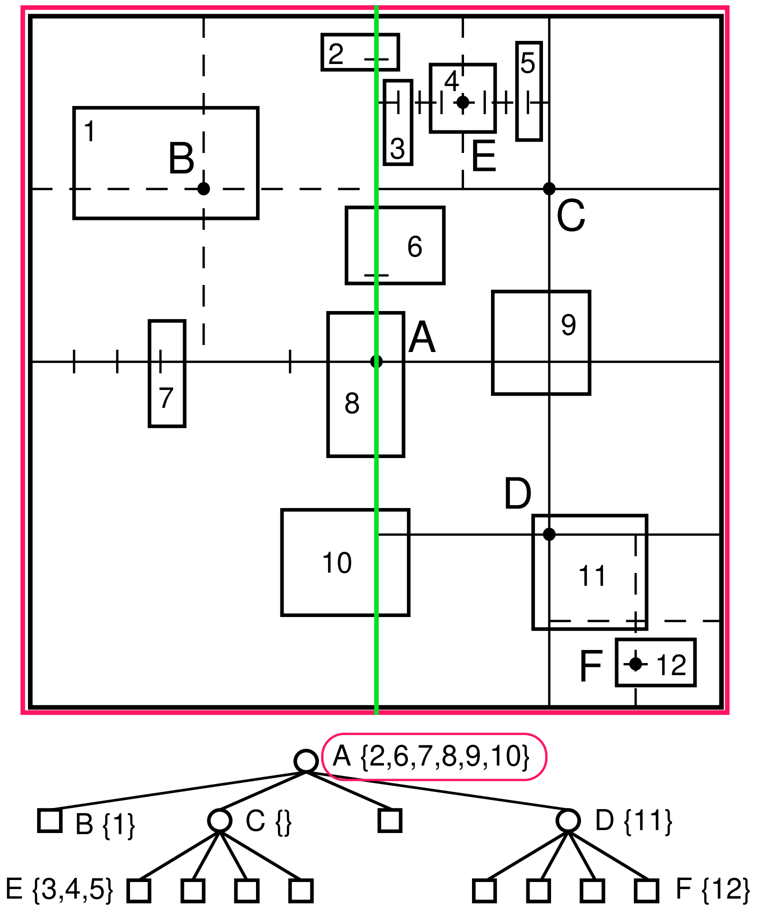
MX-CIF QUADTREE (Kedem)

1. Collections of small rectangles for VLSI applications
2. Each rectangle is associated with its minimum enclosing quadtree block
3. Like hashing: quadtree blocks serve as hash buckets
4. Collision = more than one rectangle in a block
   - resolve by using two one-dimensional MX-CIF trees to store the rectangle intersecting the lines passing through each subdivision point

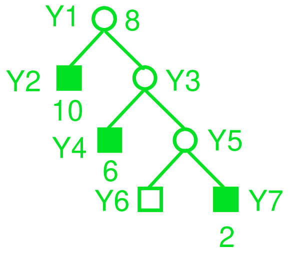


MX-CIF QUADTREE (Kedem)

1. Collections of small rectangles for VLSI applications
2. Each rectangle is associated with its minimum enclosing quadtree block
3. Like hashing: quadtree blocks serve as hash buckets
4. Collision = more than one rectangle in a block
   - resolve by using two one-dimensional MX-CIF trees to store the rectangle intersecting the lines passing through each subdivision point
   - one for y-axis

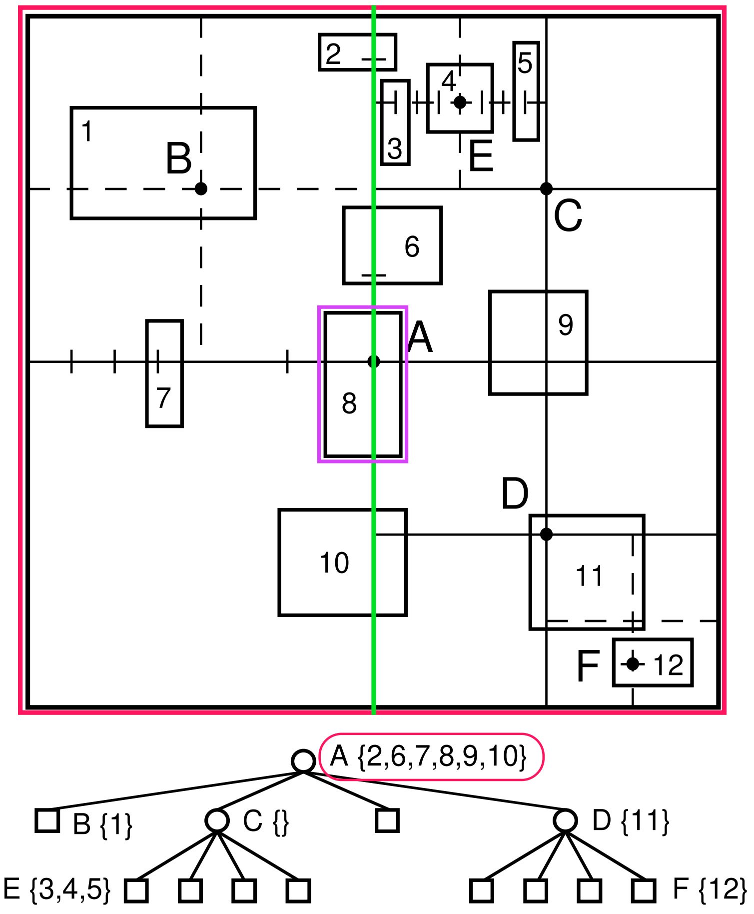


Binary tree for *y*-axis through A

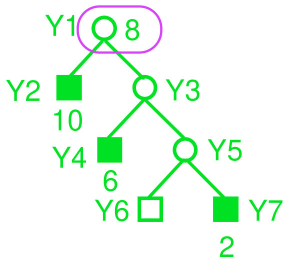


MX-CIF QUADTREE (Kedem)

1. Collections of small rectangles for VLSI applications
2. Each rectangle is associated with its minimum enclosing quadtree block
3. Like hashing: quadtree blocks serve as hash buckets
4. Collision = more than one rectangle in a block
   - resolve by using two one-dimensional MX-CIF trees to store the rectangle intersecting the lines passing through each subdivision point
   - one for y-axis
   - if a rectangle intersects both *x* and *y* axes, then associate it with the *y* axis

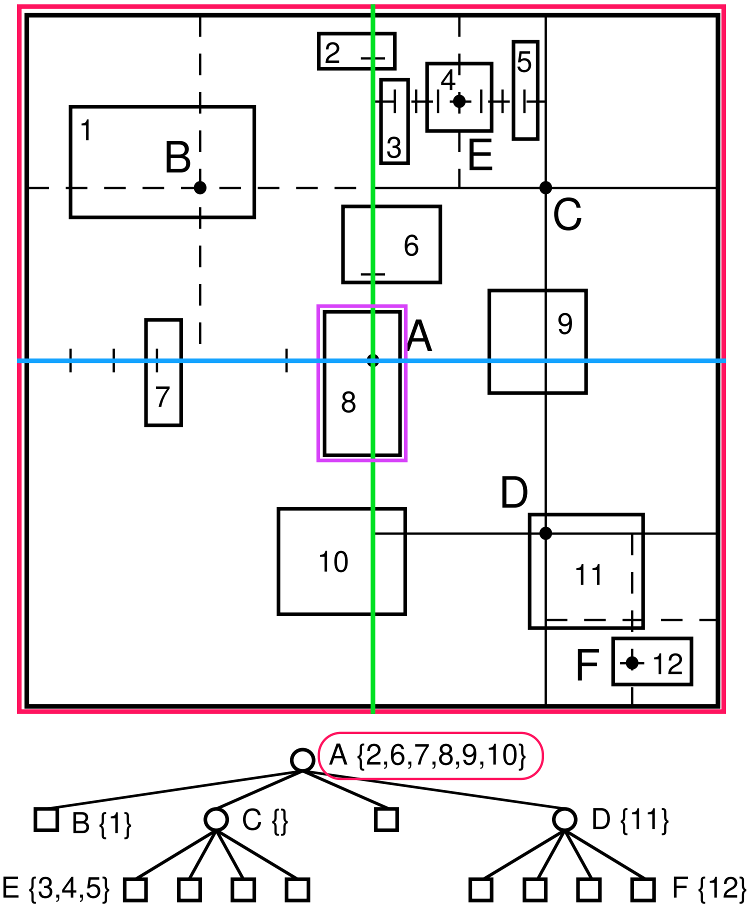


Binary tree for *y*-axis through A

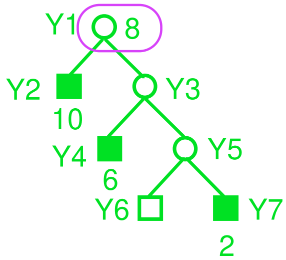


MX-CIF QUADTREE (Kedem)

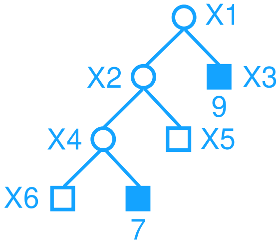
1. Collections of small rectangles for VLSI applications
2. Each rectangle is associated with its minimum enclosing quadtree block
3. Like hashing: quadtree blocks serve as hash buckets
4. Collision = more than one rectangle in a block
   - resolve by using two one-dimensional MX-CIF trees to store the rectangle intersecting the lines passing through each subdivision point
   - one for y-axis
   - if a rectangle intersects both *x* and *y* axes, then associate it with the *y* axis
   - one for x-axis



Binary tree for *y*-axis through A

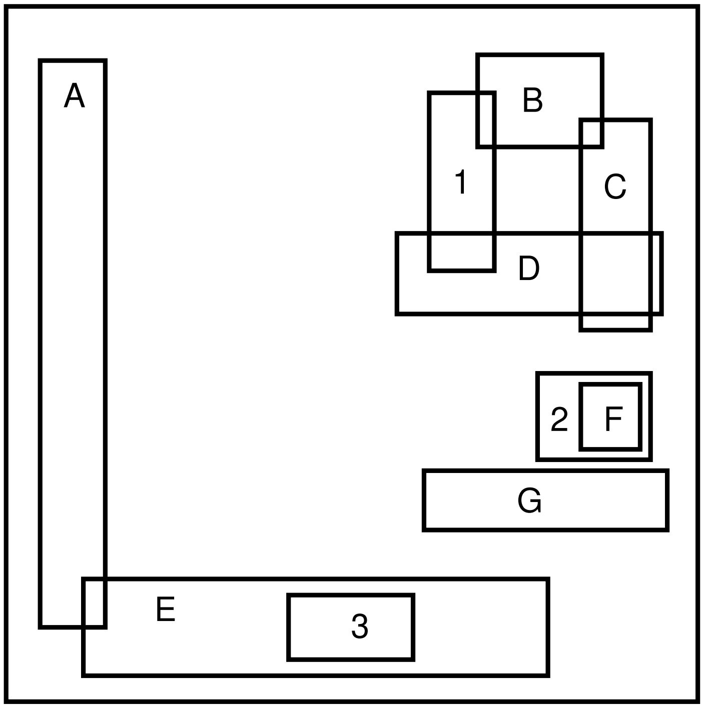


Binary tree for *x*-axis through A



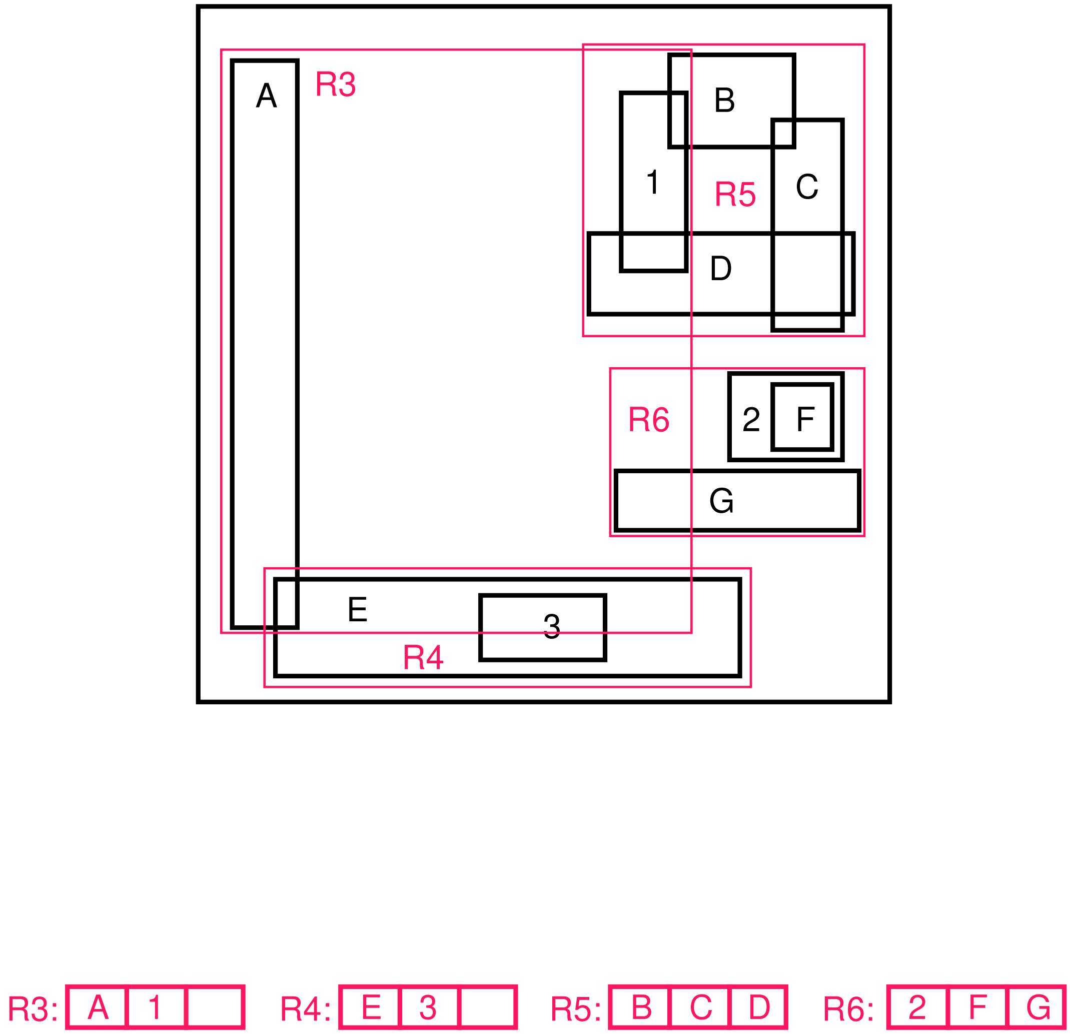
MINIMUM BOUNDING RECTANGLES

- Rectangles grouped into hierarchies, stored in another structure such as a B-tree
- Drawback: not a disjoint decomposition of space
- Rectangle has single bounding rectangle, yet area it spans may be included in several bounding rectangles
- May have to visit several rectangles to determine the presence/absence of a rectangle
- Order (m,M) R-tree
  1. between $m \leqslant \lceil M/2 \rceil$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- Ex: order (2,3) R-tree



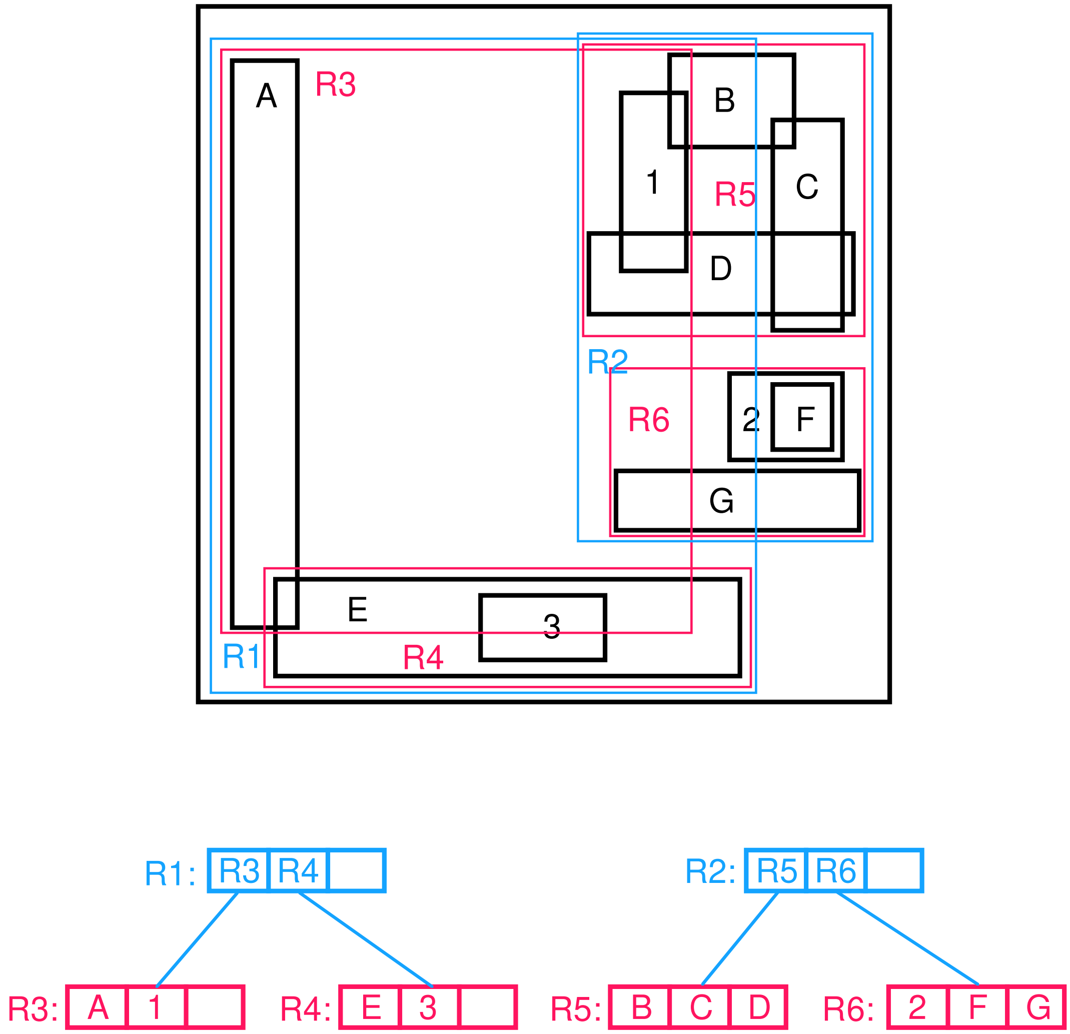
MINIMUM BOUNDING RECTANGLES

- Rectangles grouped into hierarchies, stored in another structure such as a B-tree
- Drawback: not a disjoint decomposition of space
- Rectangle has single bounding rectangle, yet area it spans may be included in several bounding rectangles
- May have to visit several rectangles to determine the presence/absence of a rectangle
- Order (m,M) R-tree
  1. between $m \leqslant \lceil M/2 \rceil$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- Ex: order (2,3) R-tree



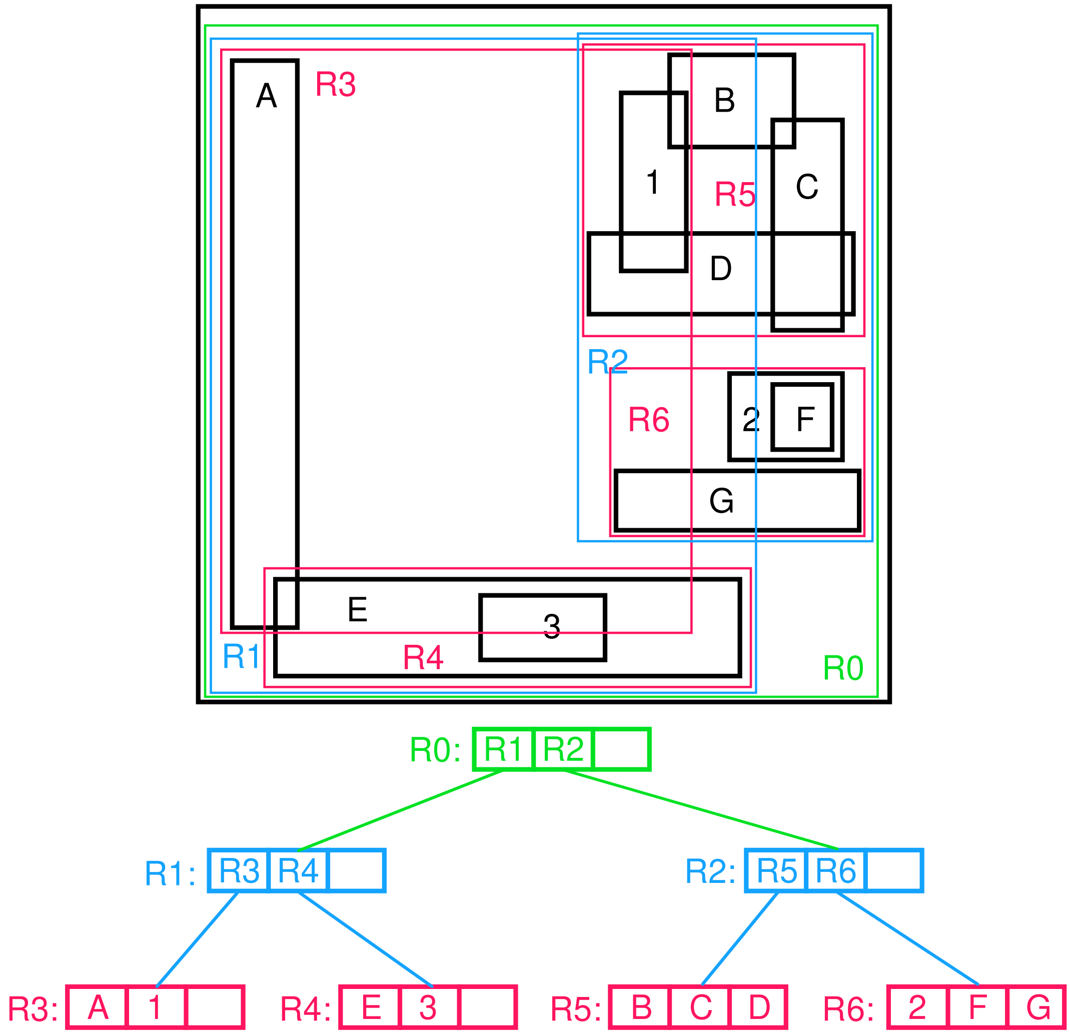
MINIMUM BOUNDING RECTANGLES

- Rectangles grouped into hierarchies, stored in another structure such as a B-tree
- Drawback: not a disjoint decomposition of space
- Rectangle has single bounding rectangle, yet area it spans may be included in several bounding rectangles
- May have to visit several rectangles to determine the presence/absence of a rectangle
- Order (m,M) R-tree
  1. between $m \leqslant \lceil M/2 \rceil$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- Ex: order (2,3) R-tree



MINIMUM BOUNDING RECTANGLES

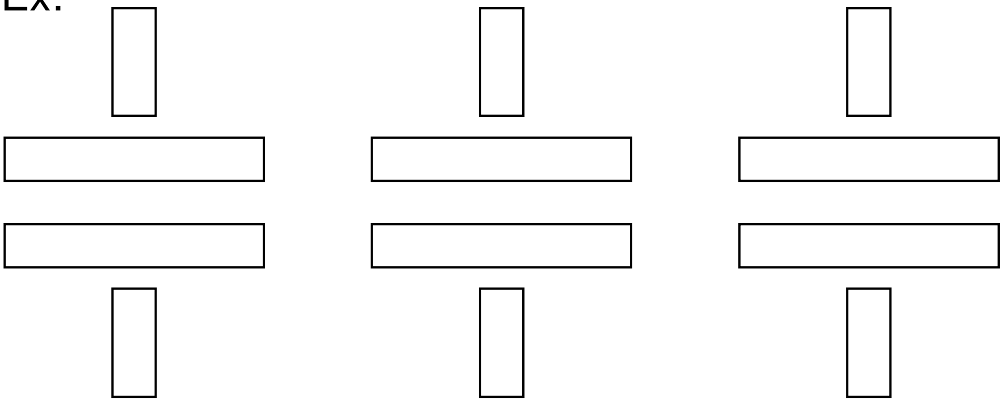
- Rectangles grouped into hierarchies, stored in another structure such as a B-tree
- Drawback: not a disjoint decomposition of space
- Rectangle has single bounding rectangle, yet area it spans may be included in several bounding rectangles
- May have to visit several rectangles to determine the presence/absence of a rectangle
- Order (*m*,*M*) R-tree
  1. between $m \leqslant \lceil M/2 \rceil$ and *M* entries in each node except root
  2. at least 2 entries in root unless a leaf node
- Ex: order (2,3) R-tree



UPDATING R-TREES

- Insertion
  1. Shape of R-tree depends on the insertion order
  2. Proceeds as in the B-tree; when overflow, must split
- Conflicting methods of distributing nodes upon a split

Ex:

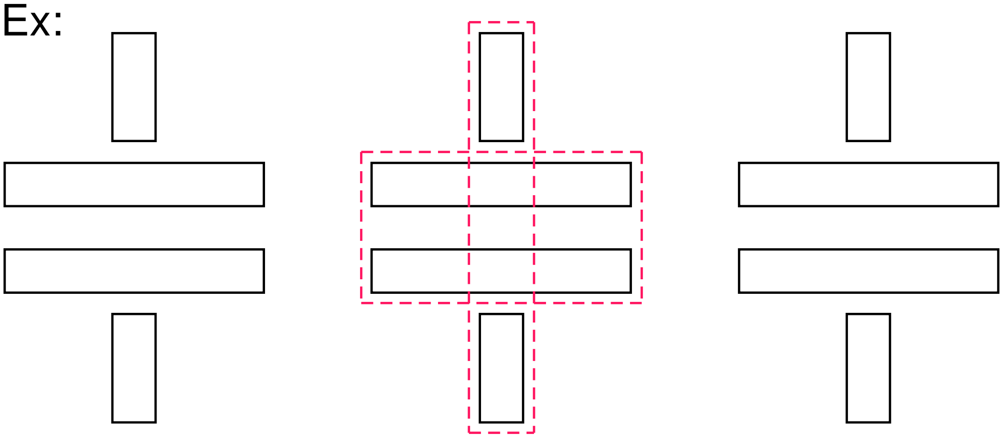


Rectangles

UPDATING R-TREES

- Insertion
  1. Shape of R-tree depends on the insertion order
  2. Proceeds as in the B-tree; when overflow, must split
- Conflicting methods of distributing nodes upon a split
  1. Goal: Reduce likelihood of visiting the nodes in later searches (i.e., minimize total area of covering rectangle for the node)

Ex:



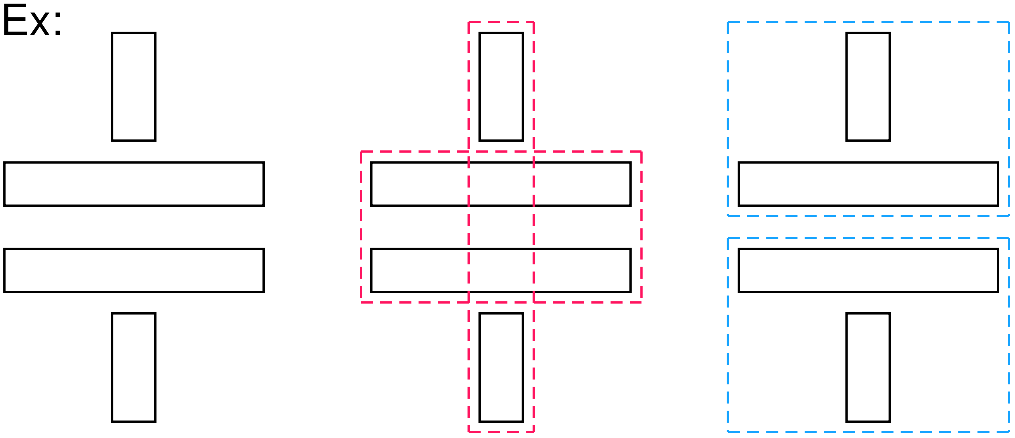
Rectangles

Goal 1

UPDATING R-TREES

- Insertion
  1. Shape of R-tree depends on the insertion order
  2. Proceeds as in the B-tree; when overflow, must split
- Conflicting methods of distributing nodes upon a split
  1. Goal: Reduce likelihood of visiting the nodes in later searches (i.e., minimize total area of covering rectangle for the node)
  2. Goal: Reduce likelihood that both nodes are visited in later searches (i.e., minimize area common to both nodes ≡ overlap)

Ex:

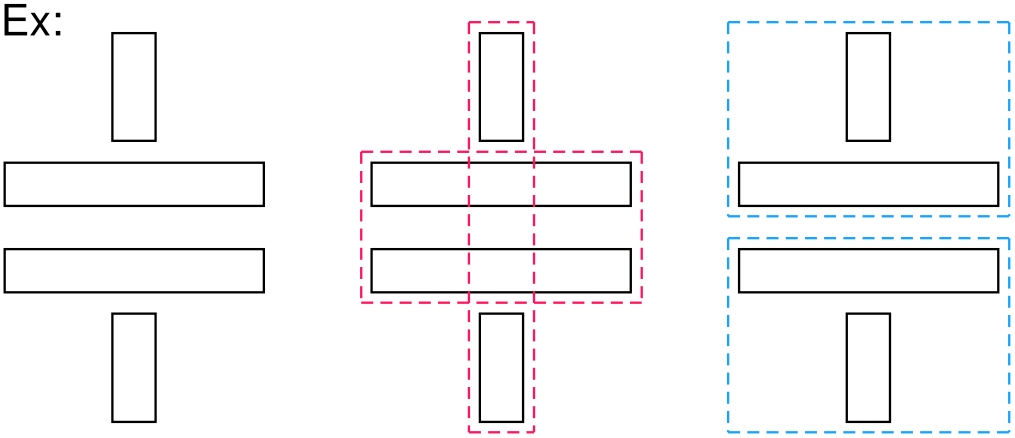


Rectangles Goal 1 Goal 2

UPDATING R-TREES

- Insertion
  1. Shape of R-tree depends on the insertion order
  2. Proceeds as in the B-tree; when overflow, must split
- Conflicting methods of distributing nodes upon a split
  1. Goal: Reduce likelihood of visiting the nodes in later searches (i.e., minimize total area of covering rectangle for the node)
  2. Goal: Reduce likelihood that both nodes are visited in later searches (i.e., minimize area common to both nodes ≡ overlap)

Ex:



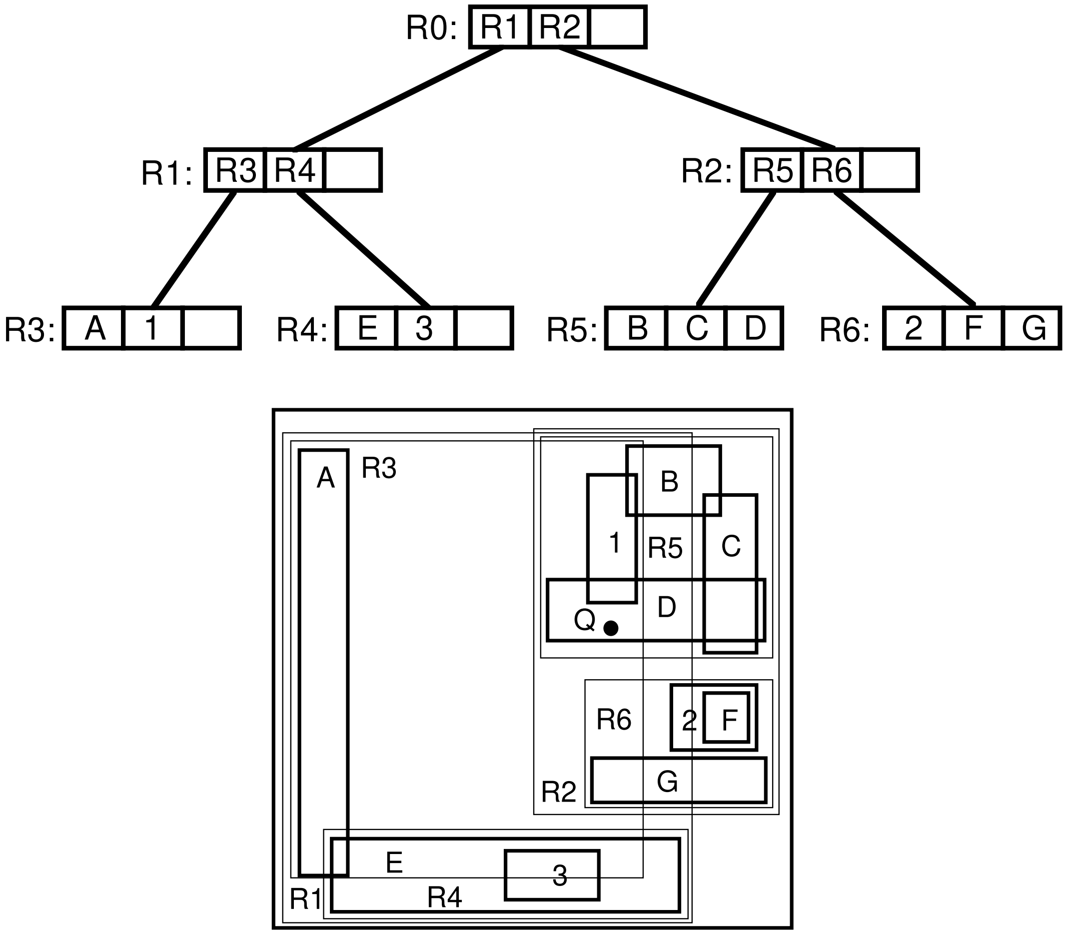
Rectangles Goal 1 Goal 2

- Deletion
  1. Delete from the leaf node and adjust covering rectangles in all nodes along the path from the root node
  2. Reinsert all nodes that underflowed rather than merging adjacent nodes
  3. Reinsertion is deemed preferable to merging because there is no concept of adjacency in an R-tree

SEARCHING FOR A RECTANGLE CONTAINING A POINT IN AN R-TREE

- Drawback is that may have to examine many nodes since a rectangle can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., D in R0, R1, R2, R3, and R5)

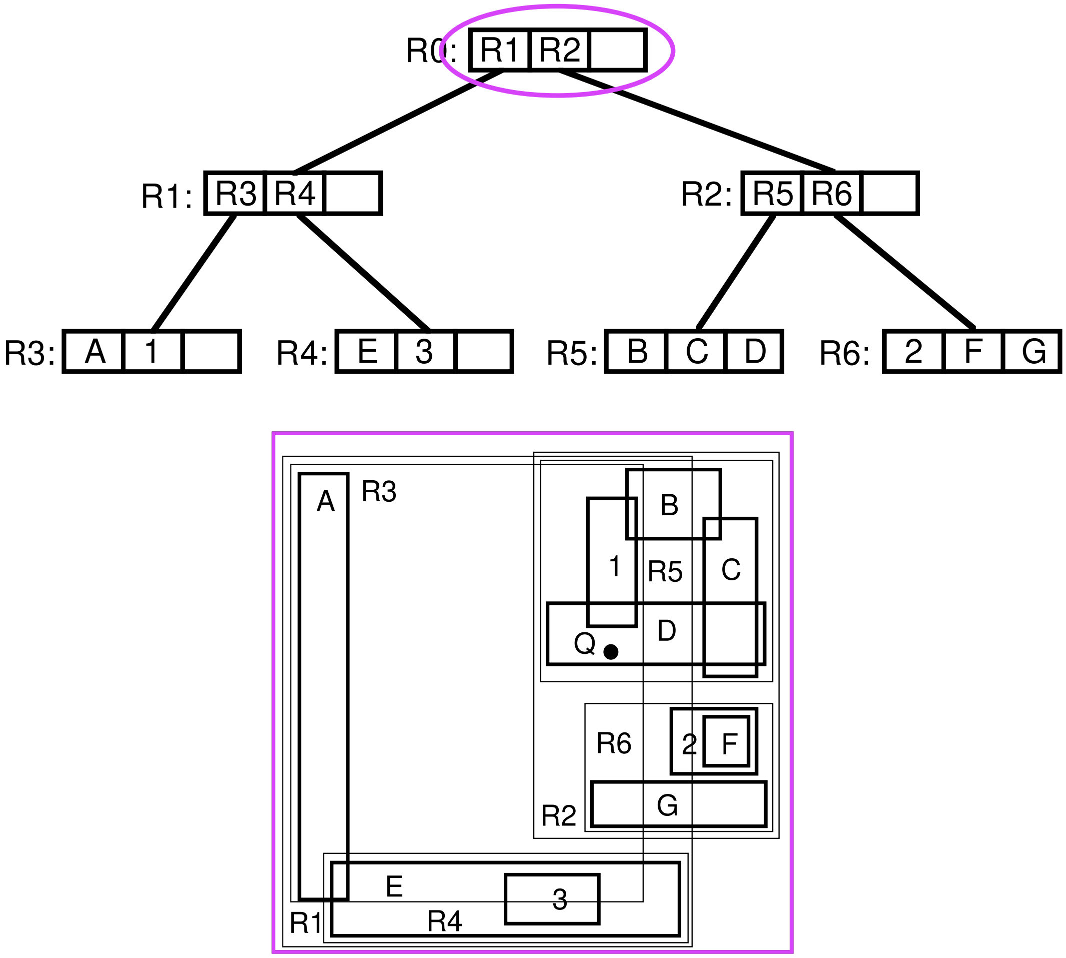
  Ex: Search for the rectangle containing point Q



SEARCHING FOR A RECTANGLE CONTAINING A POINT IN AN R-TREE

- Drawback is that may have to examine many nodes since a rectangle can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., D in R0, R1, R2, R3, and R5)

  Ex: Search for the rectangle containing point Q

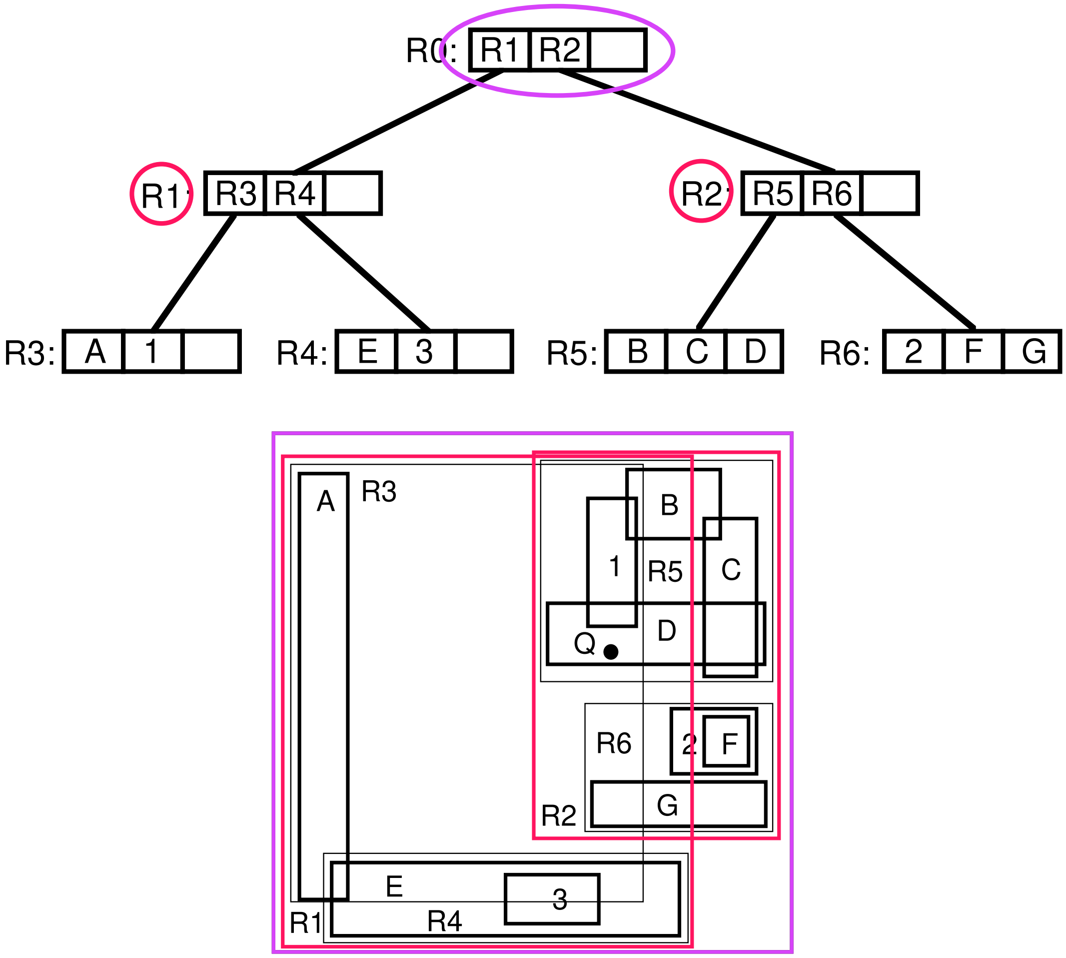


- Q is in R0

SEARCHING FOR A RECTANGLE CONTAINING A POINT IN AN R-TREE

- Drawback is that may have to examine many nodes since a rectangle can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., D in R0, R1, R2, R3, and R5)

  Ex: Search for the rectangle containing point Q

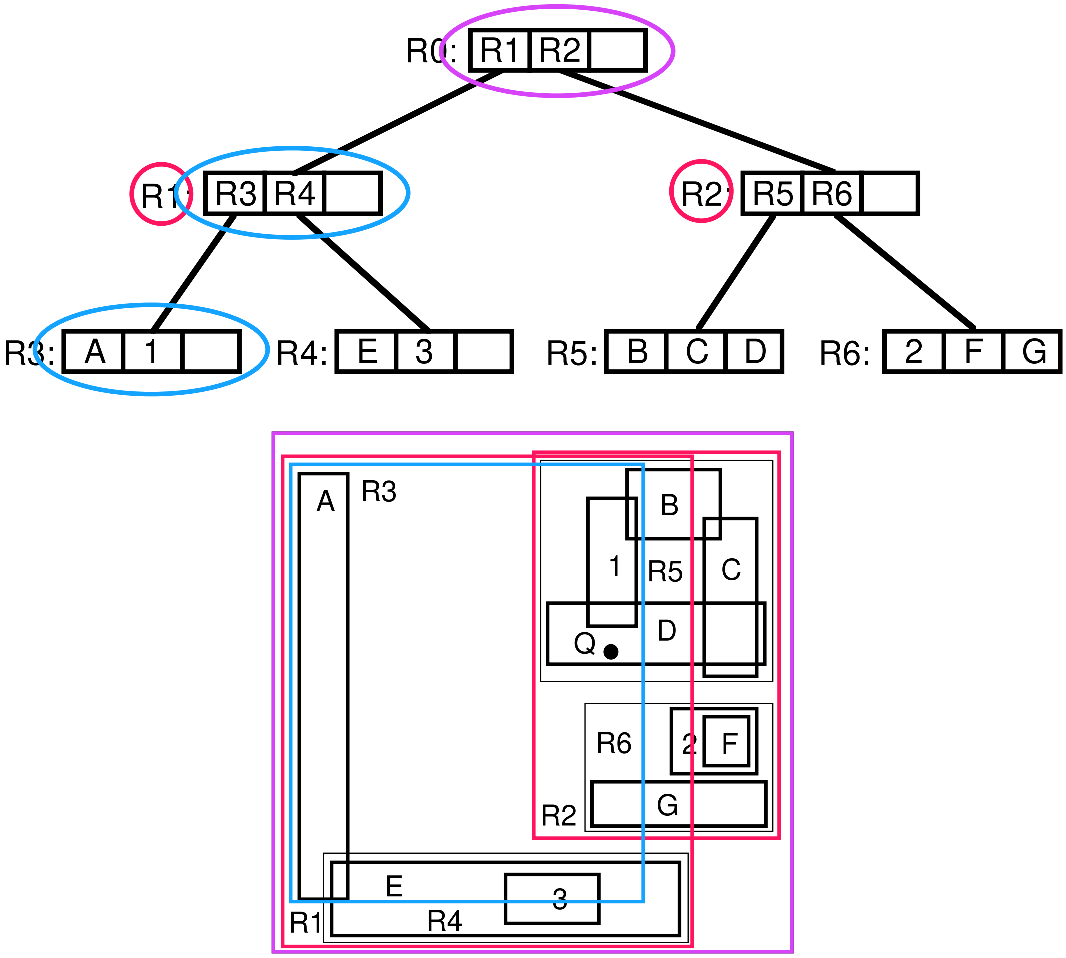


- Q is in R0
- Q can be in both R1 and R2

SEARCHING FOR A RECTANGLE CONTAINING A POINT IN AN R-TREE

- Drawback is that may have to examine many nodes since a rectangle can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., D in R0, R1, R2, R3, and R5)

  Ex: Search for the rectangle containing point Q

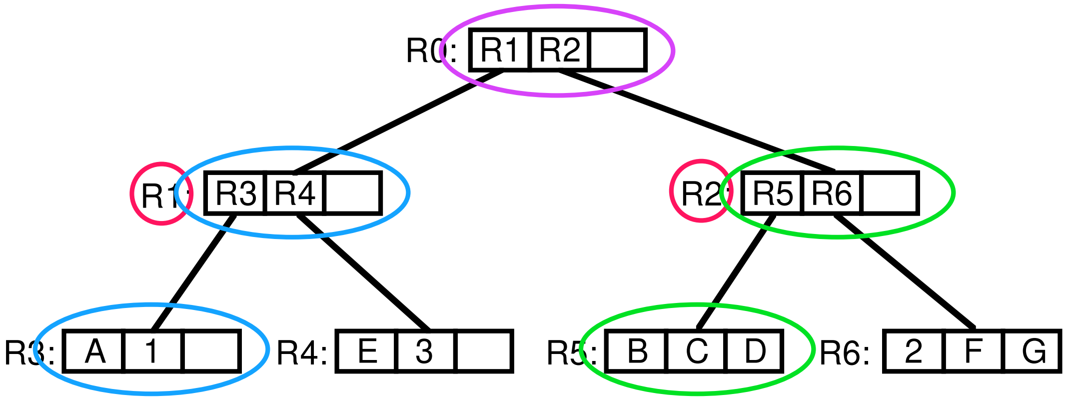


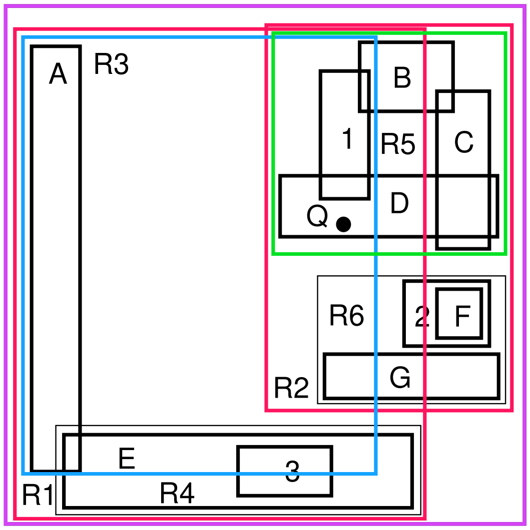
- Q is in R0
- Q can be in both R1 and R2
- Searching R1 first means that R3 is searched but this leads to failure even though Q is in a part of D which is in R3

SEARCHING FOR A RECTANGLE CONTAINING A POINT IN AN R-TREE

- Drawback is that may have to examine many nodes since a rectangle can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., D in R0, R1, R2, R3, and R5)

  Ex: Search for the rectangle containing point Q





- Q is in R0
- Q can be in both R1 and R2
- Searching R1 first means that R3 is searched but this leads to failure even though Q is in a part of D which is in R3
- Searching R2 finds that Q can only be in R5

R-TREE OVERFLOW NODE SPLITTING POLICIES

- Could use exhaustive search to look at all possible partitions
- Usually two stages:
  1. pick a pair of bounding boxes to serve as seeds for resulting nodes (‘seed-picking’)
  2. redistribute remaining nodes with goal of minimizing the growth of the total area (‘seed-growing’)
- Different algorithms of varying time complexity
  1. quadratic:
     - find two boxes j and k that would waste the most area if they were in the same node
     - for each remaining box i, determine the increase in area d_{ij} and d_{ik} of the bounding boxes of j and k resulting from the addition of i and add the box r for which $|d_{rj} - d_{rk}|$ is a maximum to the node with the smallest increase in area
     - rationale: find box with most preference for one of j, k
  2. linear:
     - find two boxes with greatest normalized separation along all of the dimensions
     - add remaining boxes in arbitrary order to box whose area is increased the least by the addition
  3. linear (Ang/Tan)
     - minimizes overlap
     - for each dimension, associate each box with the closest face of the box of the overflowing node
     - pick partition that has most even distribution
       a. if a tie, minimize overlap
       b. if a tie, minimize coverage

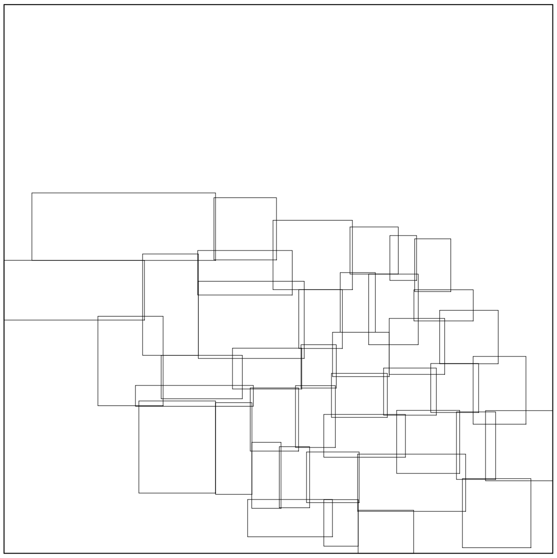
R*-TREE

- Tries to minimize overlap in case of leaf nodes and minimize increase in area for nonleaf nodes
- Changes from R-tree:
  1. insert into leaf node *p* for which the resulting bounding box has minimum increase in overlap with bounding boxes of *p*'s brothers
     - compare with R-tree where insert into leaf node for which increase in area is a minimum (minimizes coverage)
  2. in case of overflow in *p*, instead of splitting *p* as in R-tree, reinsert a fraction of objects in *p*
     - known as 'forced reinsertion' and similar to 'deferred splitting' or 'rotation' in B-trees
     - how do we pick objects to be reinserted? possibly sort by distance from center of *p* and reinsert furthest ones
  3. in case of true overflow, use a two-stage process
     - determine the axis along which the split takes place
       a. sort bounding boxes for each axis to get *d* lists
       b. choose the axis having the split value for which the sum of the perimeters of the bounding boxes of the resulting nodes is the smallest while still satisfying the capacity constraints (reduces coverage)
     - determine the position of the split
       a. position where overlap between two nodes is minimized
       b. resolve ties by minimizing total area of bounding boxes (reduces coverage)
- Works very well but takes time due to reinsertion

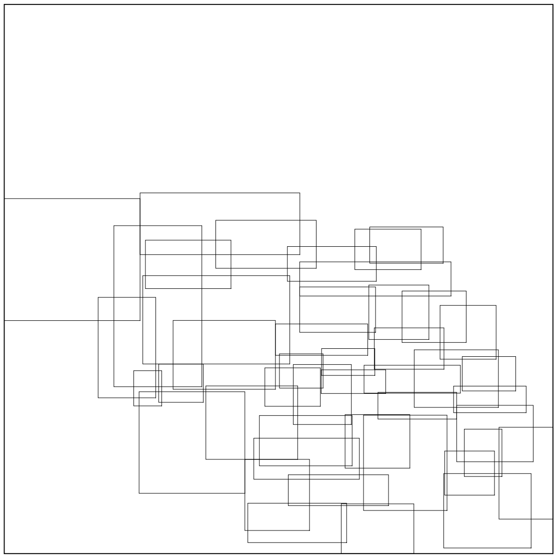
EXAMPLE OF R-TREE NODE SPLITTING POLICIES

- Sample collection of 1700 lines using m=20 and M=50

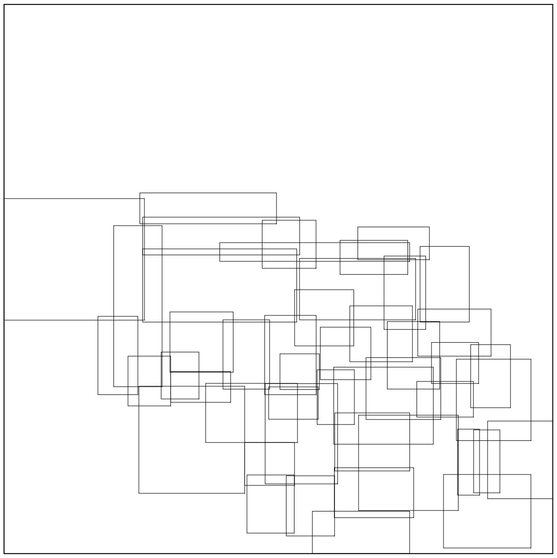
Collection of lines



R*-tree



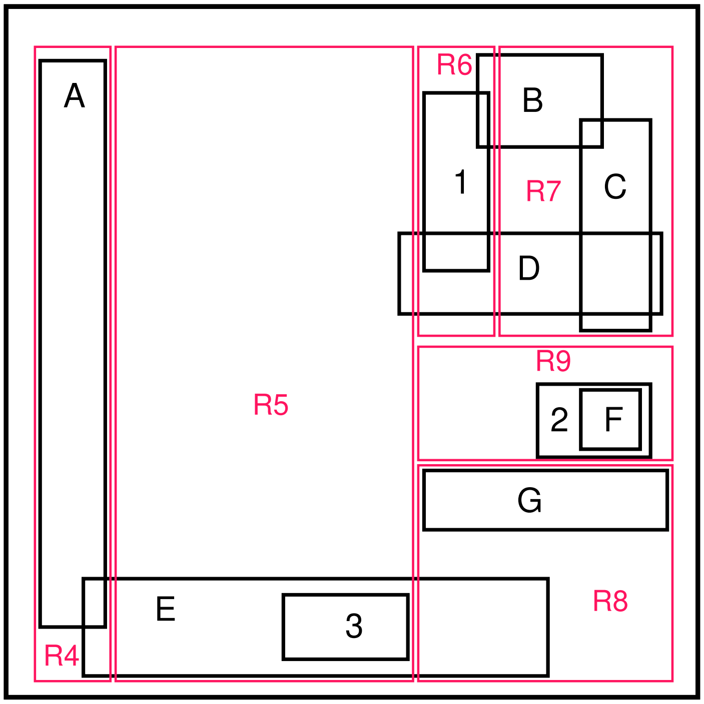
Linear



Quadratic

K-D-B-TREES

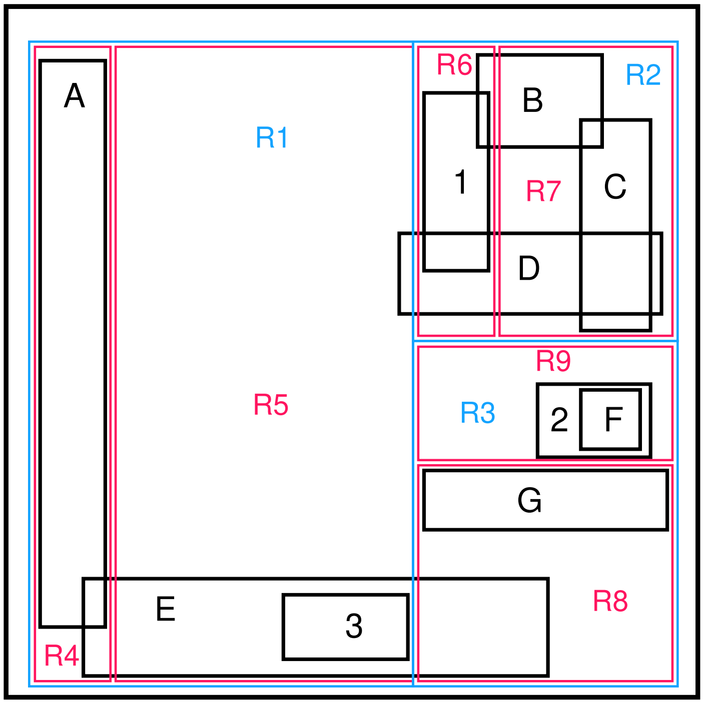
- Rectangular embedding space is hierarchically decomposed into disjoint rectangular regions
- No dead space in the sense that at any level of the tree, entire embedding space is covered by one of the nodes
- Blocks of k-d tree partition of space are aggregated into nodes of a finite capacity
- When a node overflows, it is split along one of the axes
- Originally developed to store points but may be extended to non-point objects represented by their minimum bounding boxes
- Drawback: in order to determine area covered by object, must retrieve all cells that it occupies

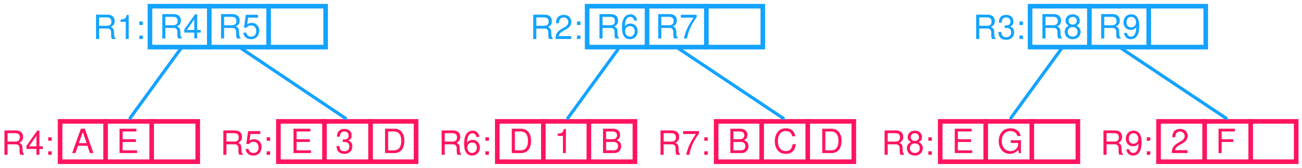


R4: A E
R5: E 3 D
R6: D 1 B
R7: B C D
R8: E G
R9: 2 F

K-D-B-TREES

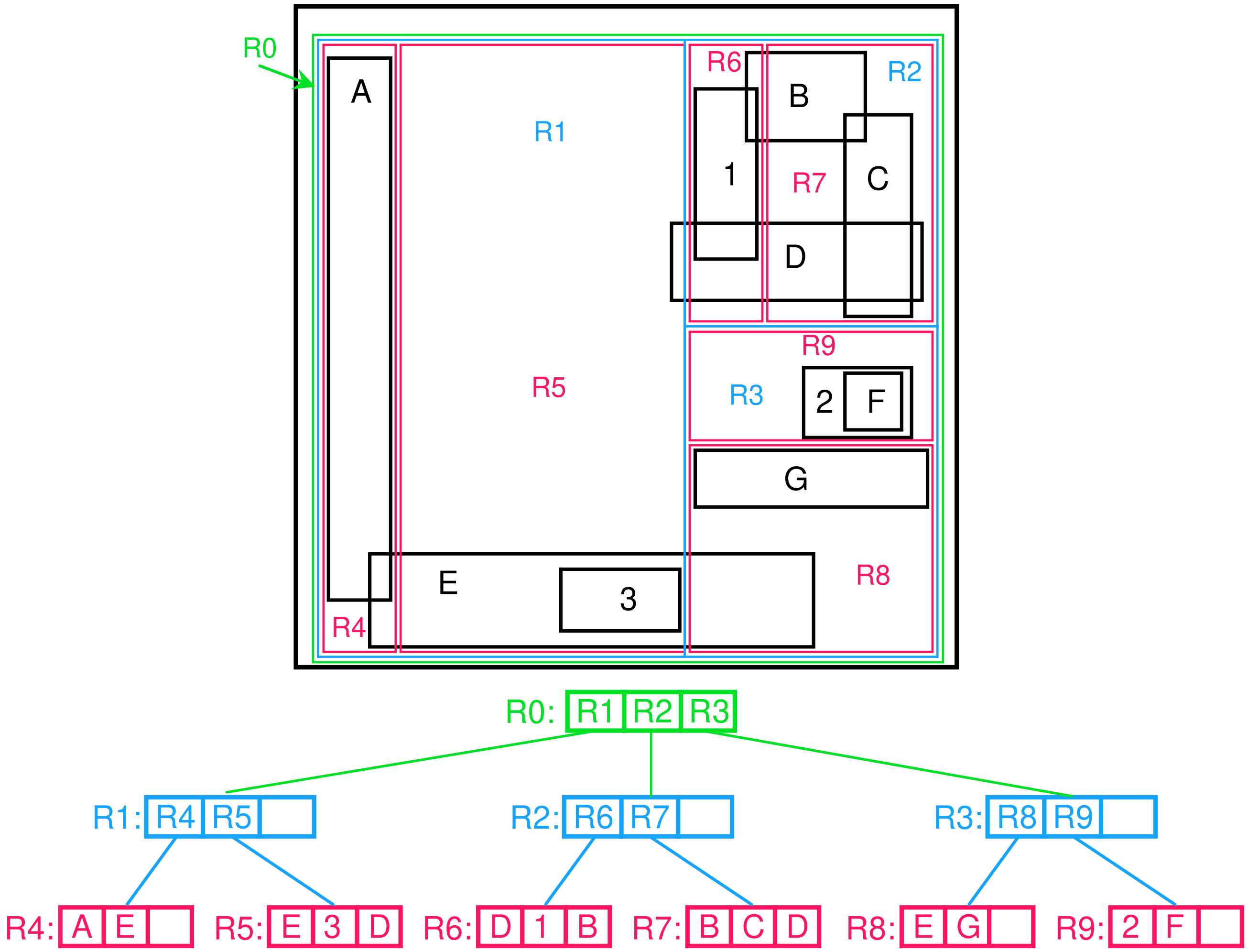
- Rectangular embedding space is hierarchically decomposed into disjoint rectangular regions
- No dead space in the sense that at any level of the tree, entire embedding space is covered by one of the nodes
- Blocks of k-d tree partition of space are aggregated into nodes of a finite capacity
- When a node overflows, it is split along one of the axes
- Originally developed to store points but may be extended to non-point objects represented by their minimum bounding boxes
- Drawback: in order to determine area covered by object, must retrieve all cells that it occupies





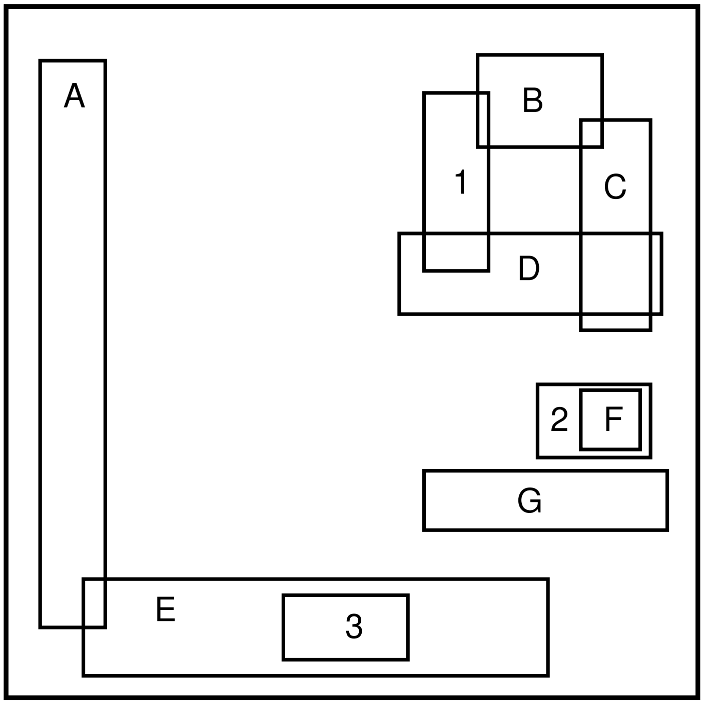
K-D-B-TREES

- Rectangular embedding space is hierarchically decomposed into disjoint rectangular regions
- No dead space in the sense that at any level of the tree, entire embedding space is covered by one of the nodes
- Blocks of k-d tree partition of space are aggregated into nodes of a finite capacity
- When a node overflows, it is split along one of the axes
- Originally developed to store points but may be extended to non-point objects represented by their minimum bounding boxes
- Drawback: in order to determine area covered by object, must retrieve all cells that it occupies



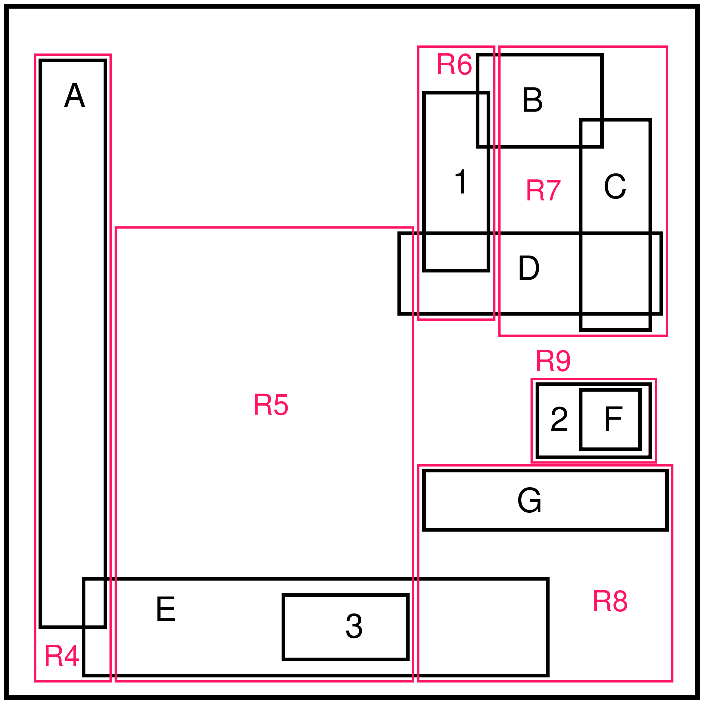
R+-TREES

- Rectangles are decomposed into disjoint subrectangles
- Subrectangles are aggregated hierarchically into larger disjoint rectangles
- Equivalent to a k-d-B-tree with difference that a rectangle at depth *i* is a minimum bounding rectangle of the contained rectangles at depth *i*+1
- Advantage over k-d-B-tree in that false hits are reduced
- Same drawback of duplicate reporting as in k-d-B-tree



R+-TREES

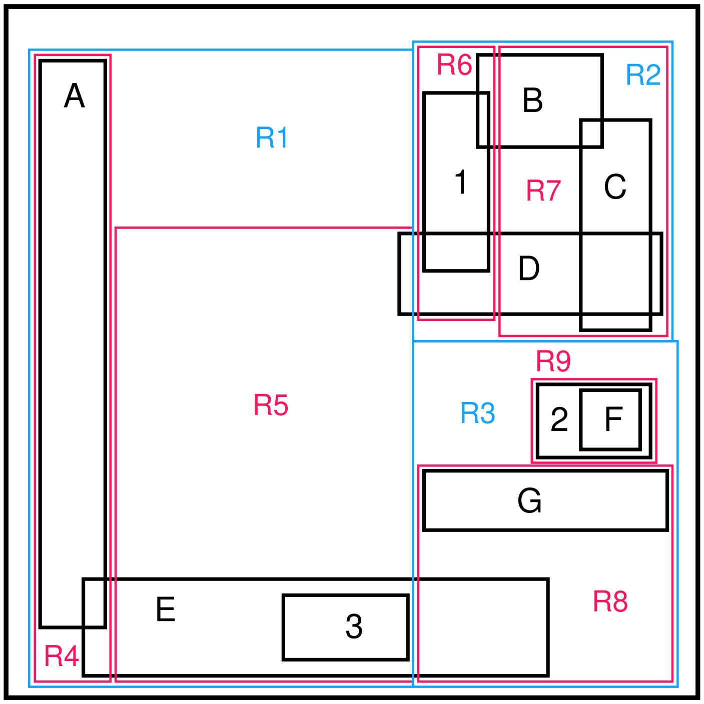
- Rectangles are decomposed into disjoint subrectangles
- Subrectangles are aggregated hierarchically into larger disjoint rectangles
- Equivalent to a k-d-B-tree with difference that a rectangle at depth *i* is a minimum bounding rectangle of the contained rectangles at depth *i*+1
- Advantage over k-d-B-tree in that false hits are reduced
- Same drawback of duplicate reporting as in k-d-B-tree

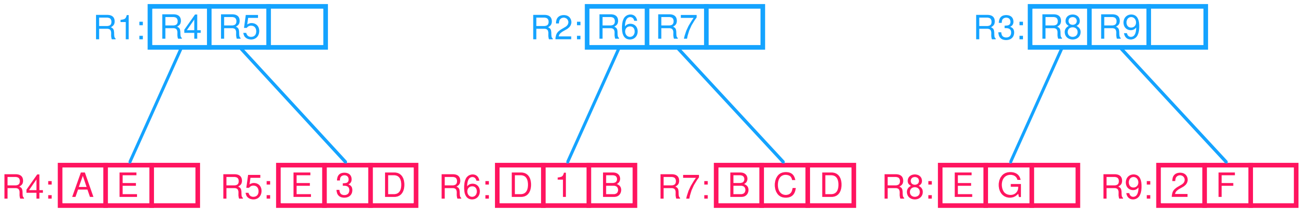


R4: A E
R5: E 3 D
R6: D 1 B
R7: B C D
R8: E G
R9: 2 F

R+-TREES

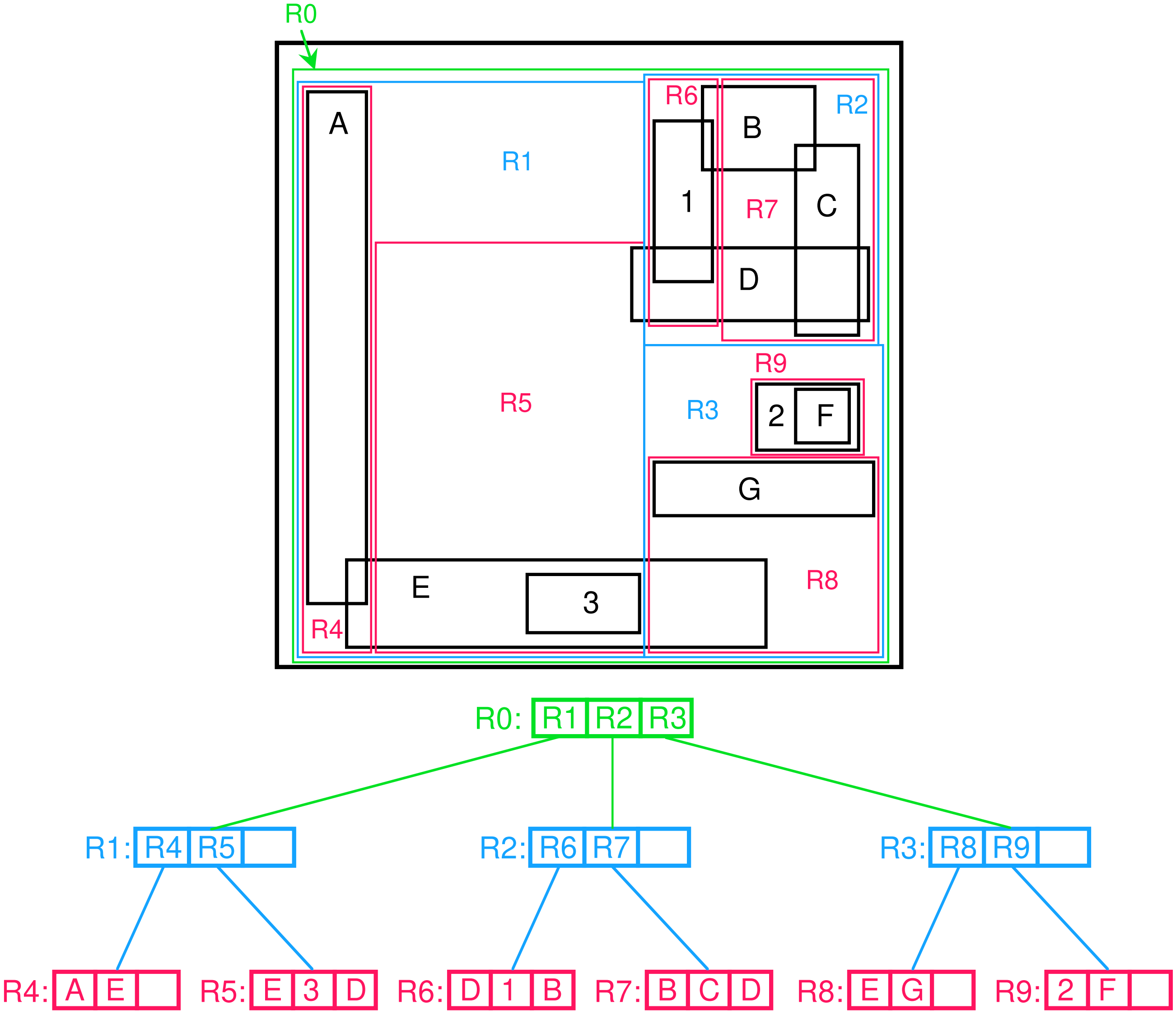
- Rectangles are decomposed into disjoint subrectangles
- Subrectangles are aggregated hierarchically into larger disjoint rectangles
- Equivalent to a k-d-B-tree with difference that a rectangle at depth *i* is a minimum bounding rectangle of the contained rectangles at depth *i*+1
- Advantage over k-d-B-tree in that false hits are reduced
- Same drawback of duplicate reporting as in k-d-B-tree





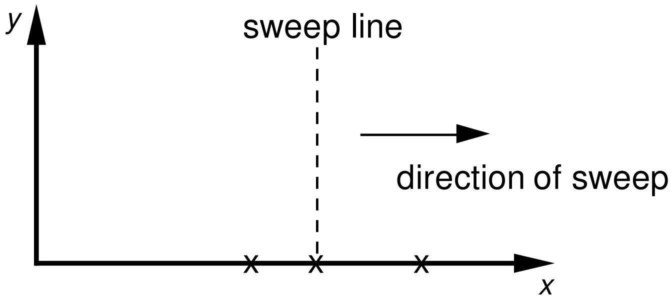
R+-TREES

- Rectangles are decomposed into disjoint subrectangles
- Subrectangles are aggregated hierarchically into larger disjoint rectangles
- Equivalent to a k-d-B-tree with difference that a rectangle at depth *i* is a minimum bounding rectangle of the contained rectangles at depth *i*+1
- Advantage over k-d-B-tree in that false hits are reduced
- Same drawback of duplicate reporting as in k-d-B-tree



PLANE-SWEEP METHODS

- Representations for use with results from computational geometry
  1. segment trees
  2. interval trees
- Solve geometric problems by sweeping a line (plane in 3-d) across the plane (space in 3-d) and halting at points of intersection with the objects being processed



x ≡ halting points

- Compute a partial solution so that at the end of the sweep we have a final solution
- Assume 2-d data
- Organize two sets of data:
  1. Halting points of the sweep line (i.e., x coordinates) which are sorted in ascending order
  2. Description of the status of the objects inserted by the current position of the sweep line
     - status is problem-dependent
     - efficiency of solution depends on the data structure
- Basically, a multidimensional sort!
- Efficiency of solutions employing plane sweep are constrained by how fast we can sort — i.e., $O(N \cdot \log_2 N)$