Part 1: Indexing Low and High Dimensional Spaces

1. Quadtree variants
2. k-d tree
3. R-tree
4. Bounding sphere methods
5. Hybrid tree
6. Avoiding overlapping all of the leaf blocks
7. Pyramid technique
8. Methods based on a sequential scan

Simple Non-Hierarchical Data Structures

Sequential list

Name	X	Y
Chicago	35	42
Mobile	52	10
Toronto	62	77
Buffalo	82	65
Denver	5	45
Omaha	27	35
Atlanta	85	15
Miami	90	5

Inverted List

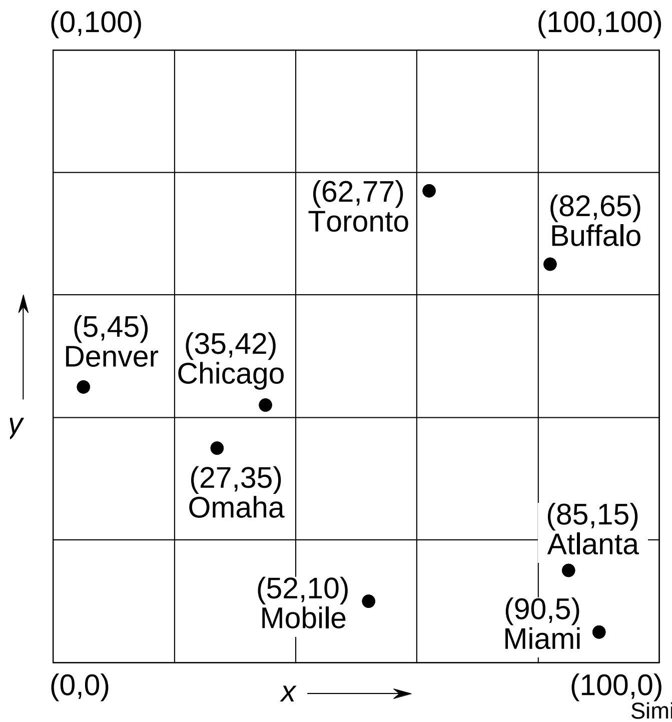
X	Y
Denver	Miami
Omaha	Mobile
Chicago	Atlanta
Mobile	Omaha
Toronto	Chicago
Buffalo	Denver
Atlanta	Buffalo
Miami	Toronto

Inverted lists:

1. 2 sorted lists
2. data is pointers
3. enables pruning the search with respect to one key

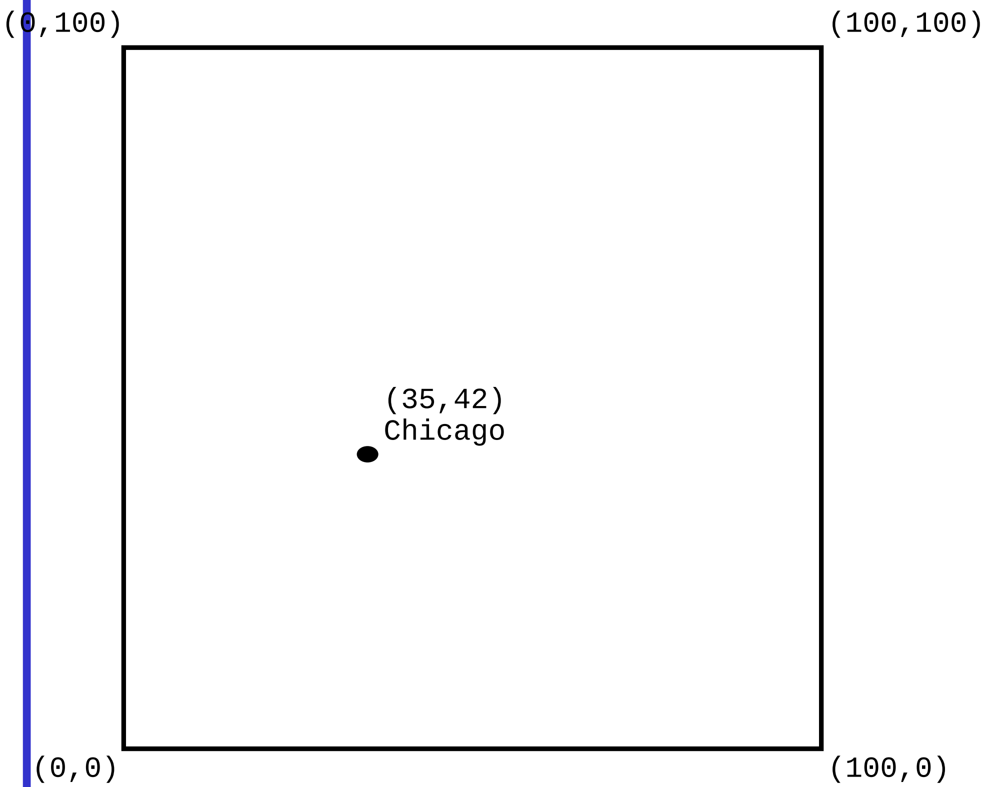
Grid Method

- Divide space into squares of width equal to the search region
- Each cell contains a list of all points within it
- Assume L_∞ distance metric (i.e., Chessboard)
- Assume C = uniform distribution of points per cell
- Average search time for k-dimensional space is $O(F \cdot 2^k)$
  - F = number of records found = C, since query region has the width of a cell
  - 2^k = number of cells examined



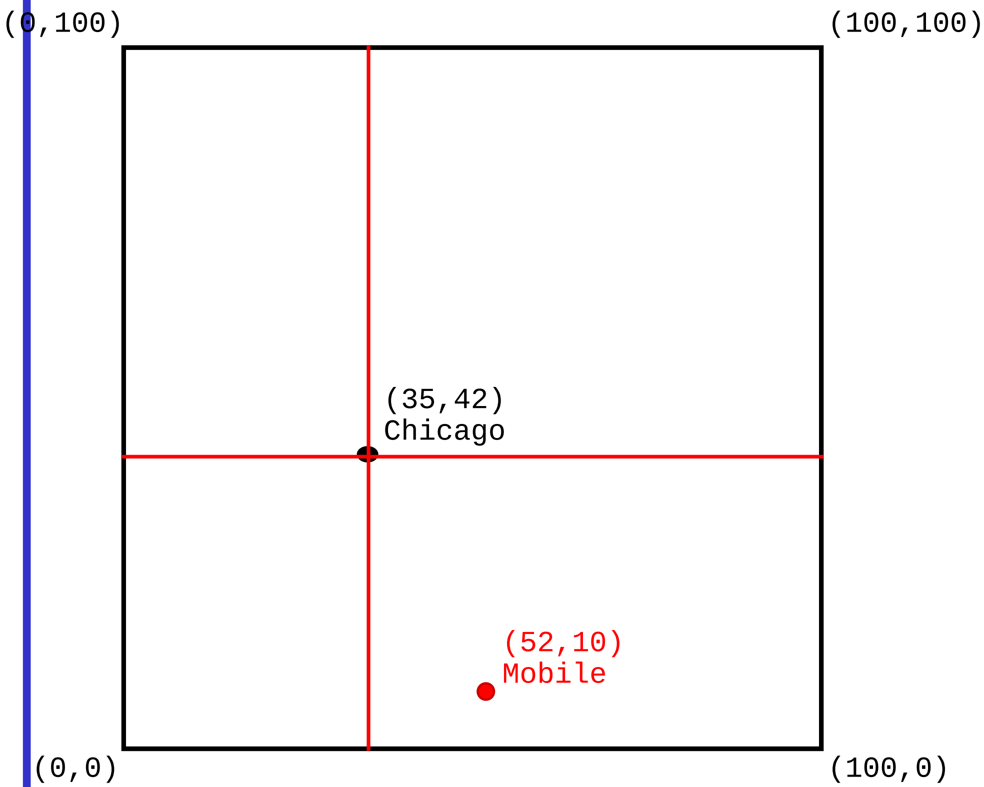
Point Quadtree (Finkel/Bentley)

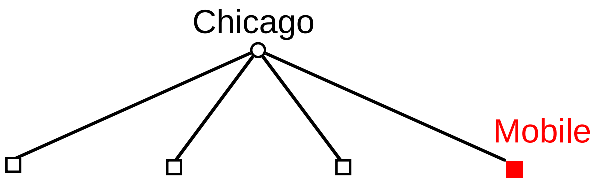
- Marriage between uniform grid and a binary search tree



Point Quadtree (Finkel/Bentley)

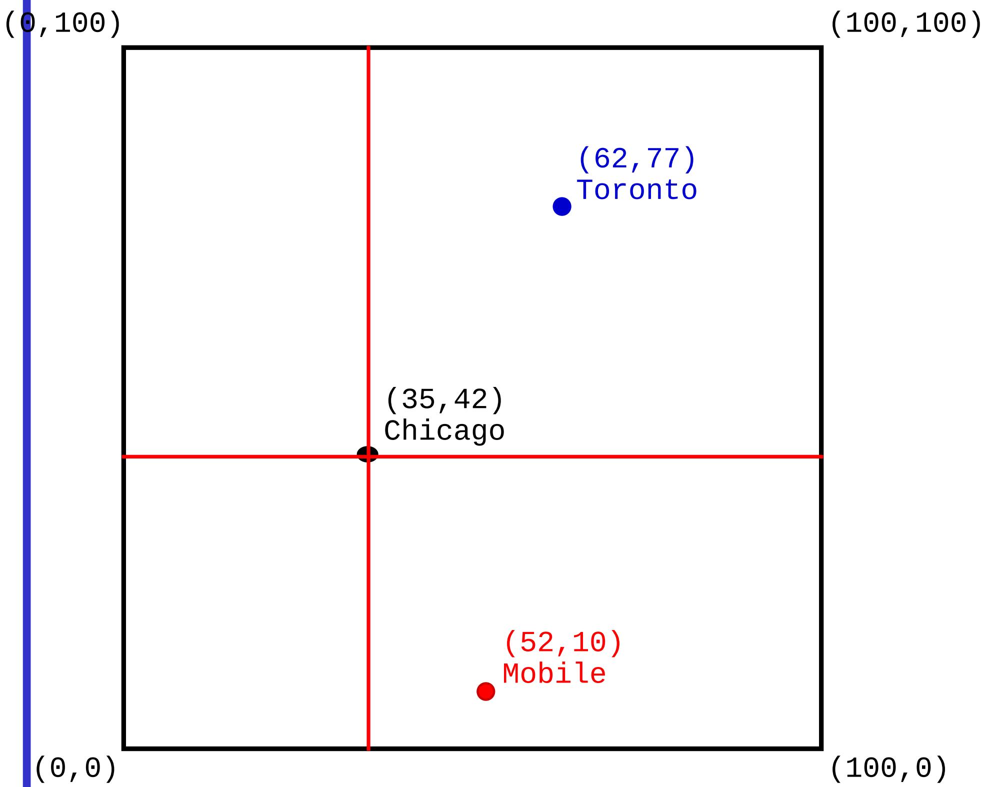
- Marriage between uniform grid and a binary search tree

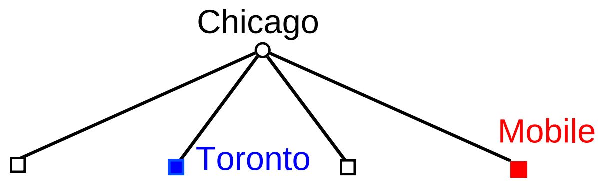




Point Quadtree (Finkel/Bentley)

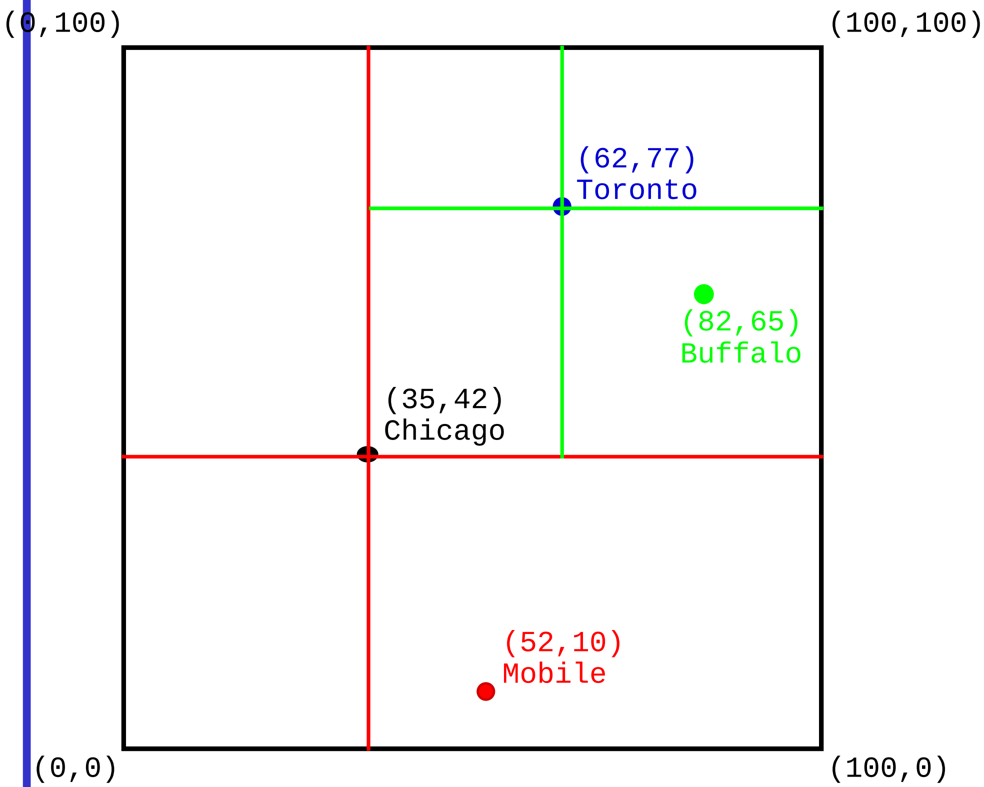
- Marriage between uniform grid and a binary search tree

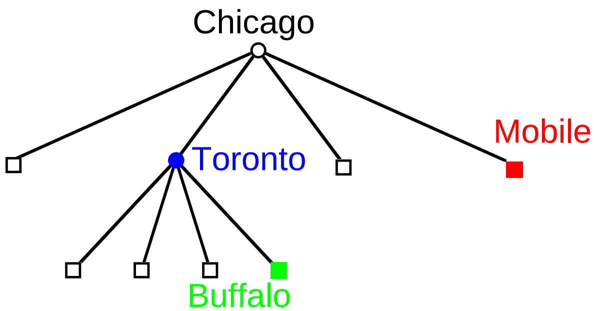




Point Quadtree (Finkel/Bentley)

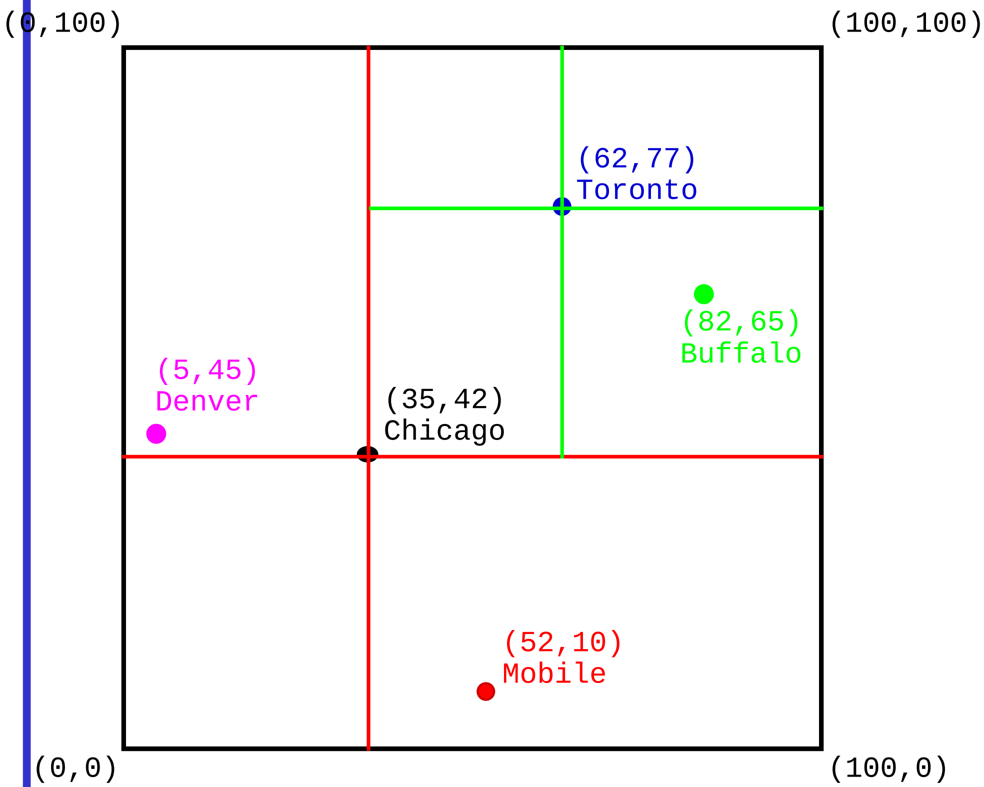
- Marriage between uniform grid and a binary search tree

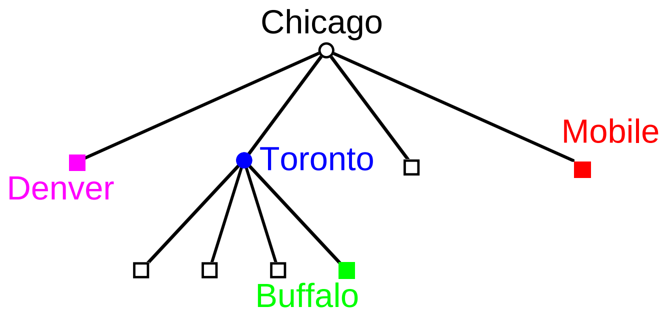




Point Quadtree (Finkel/Bentley)

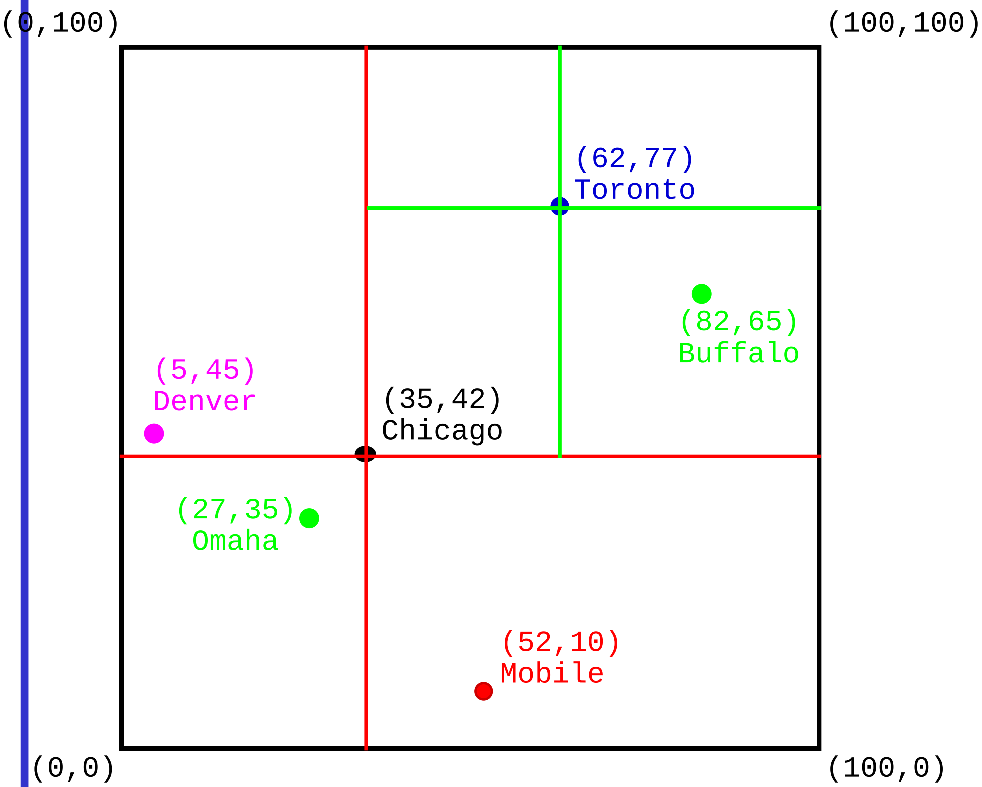
- Marriage between uniform grid and a binary search tree

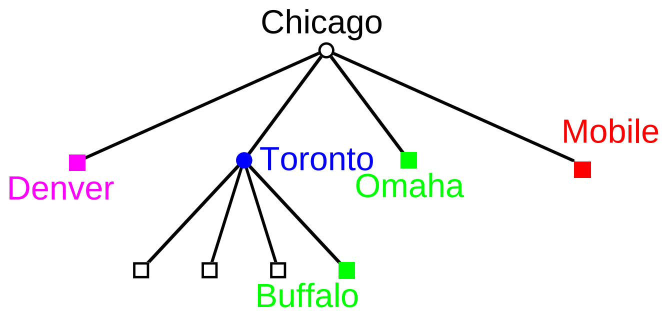




Point Quadtree (Finkel/Bentley)

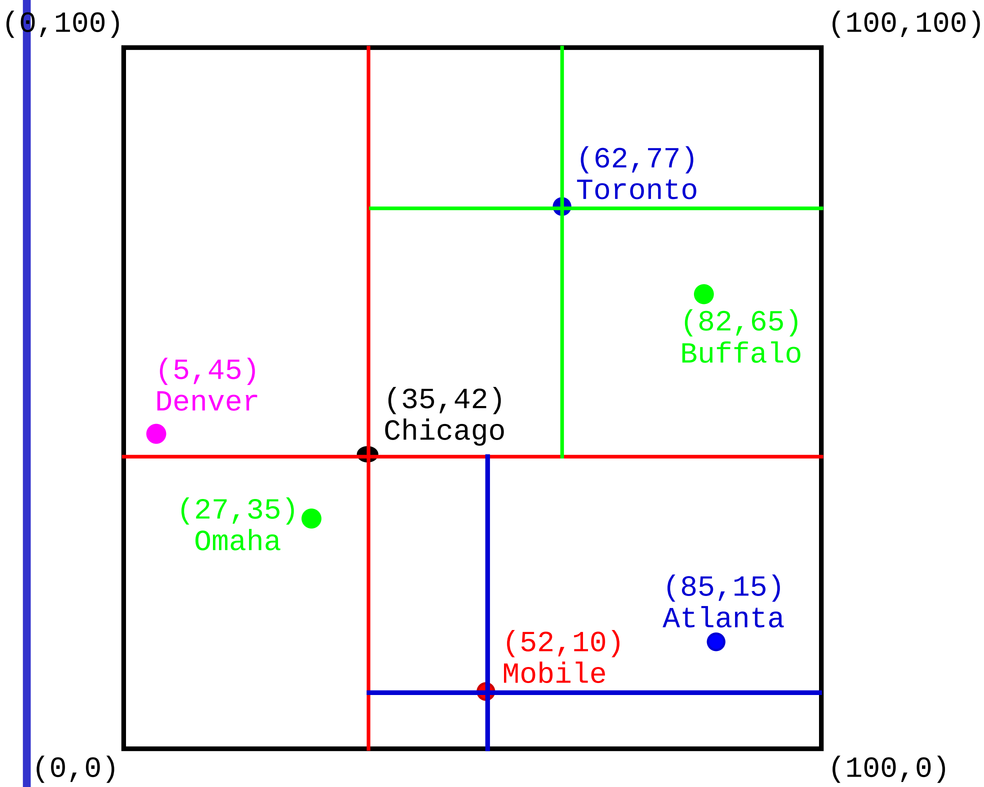
- Marriage between uniform grid and a binary search tree

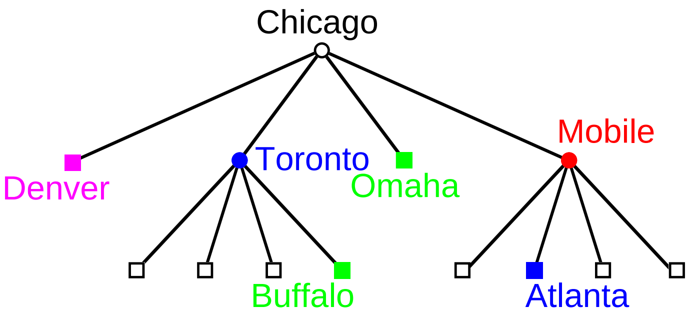




Point Quadtree (Finkel/Bentley)

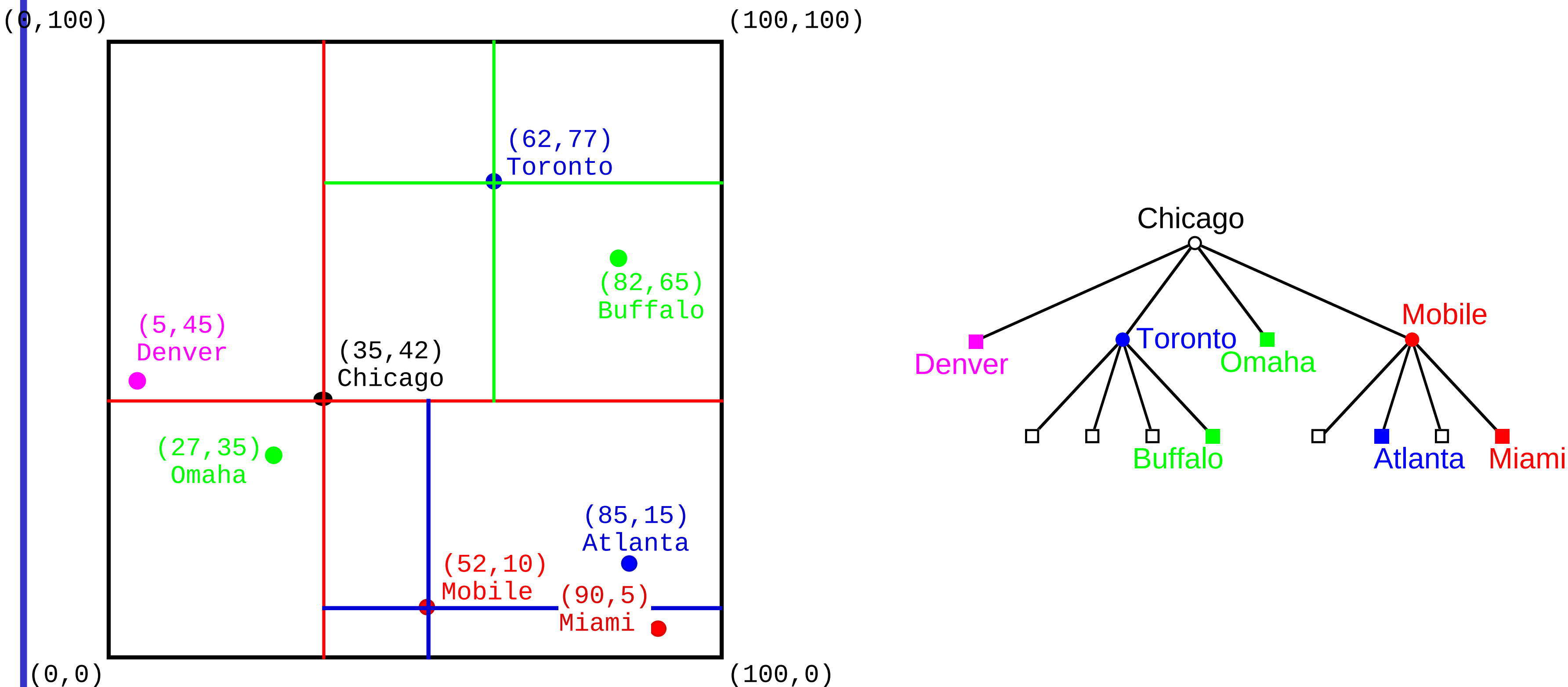
- Marriage between uniform grid and a binary search tree





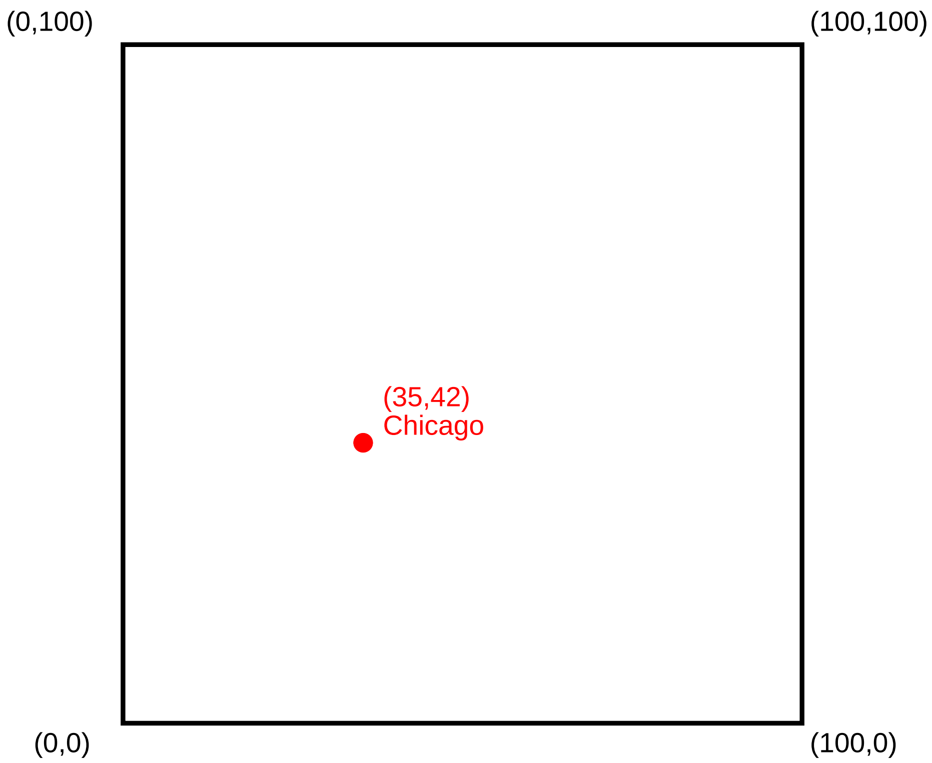
Point Quadtree (Finkel/Bentley)

- Marriage between uniform grid and a binary search tree



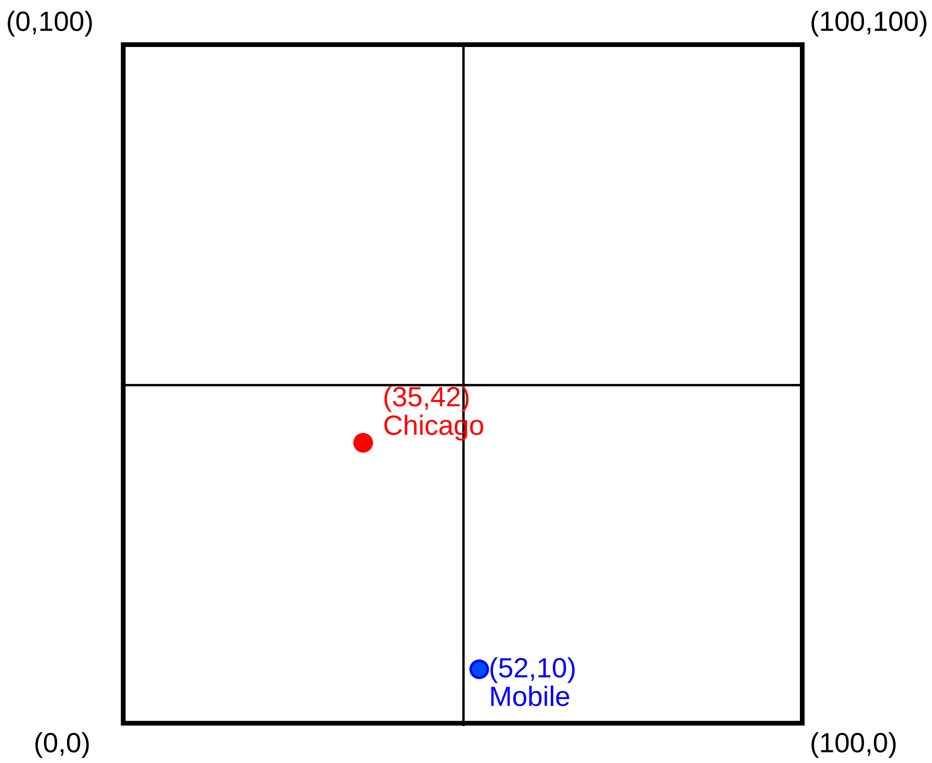
PR Quadtree

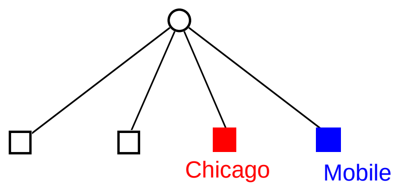
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points



PR Quadtree

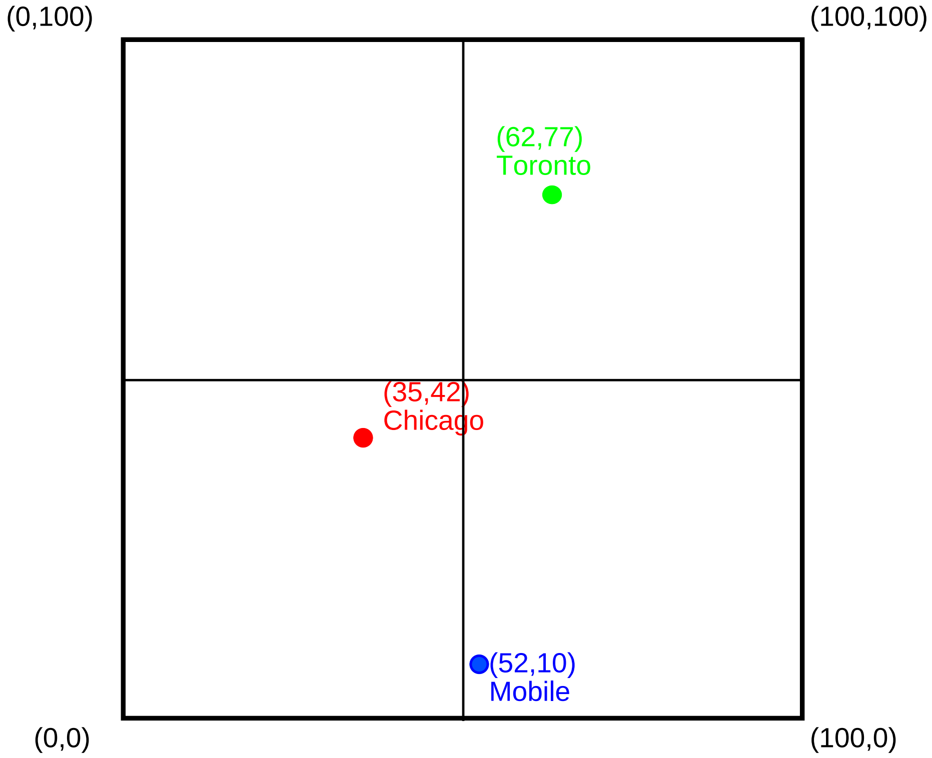
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

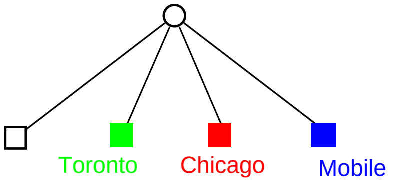




PR Quadtree

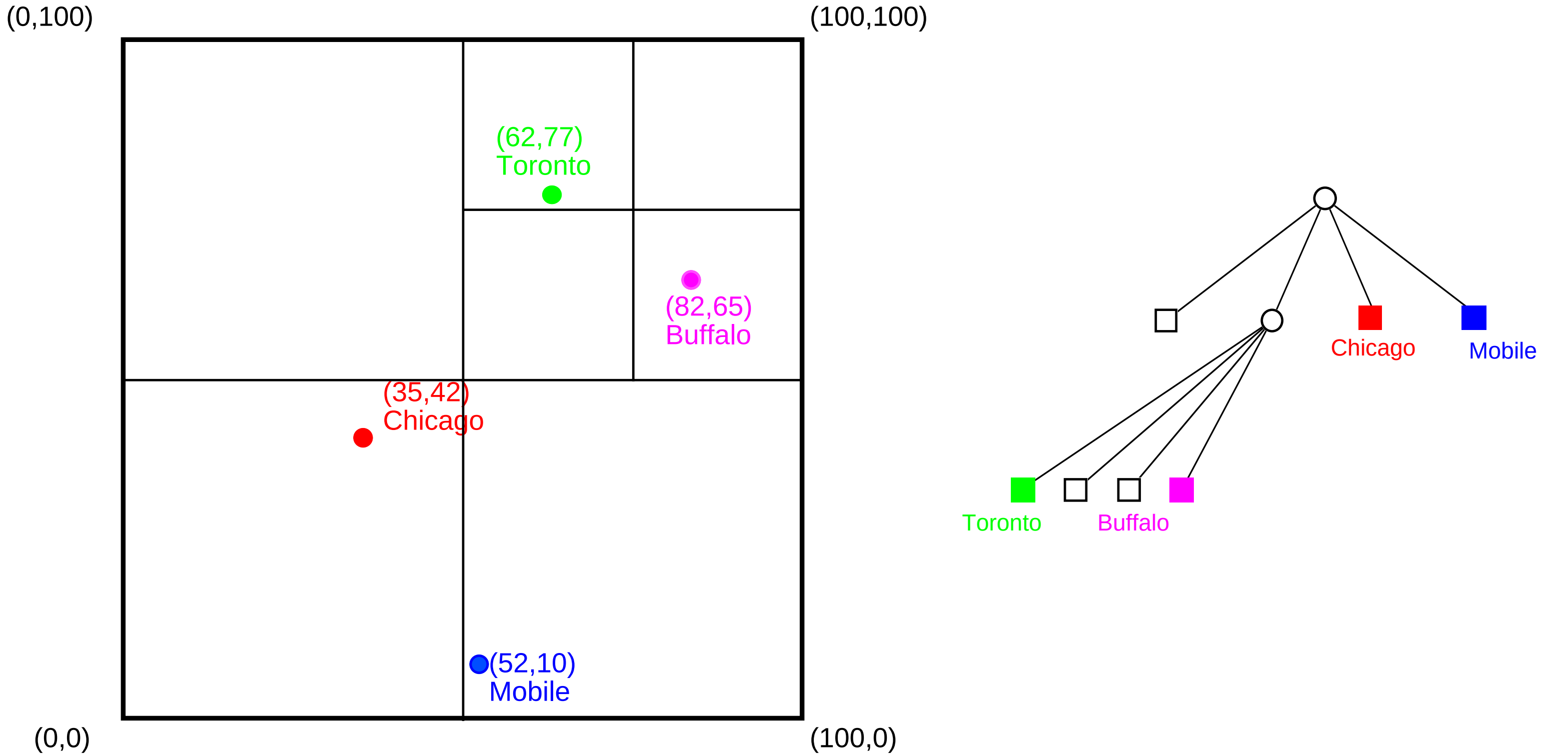
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points





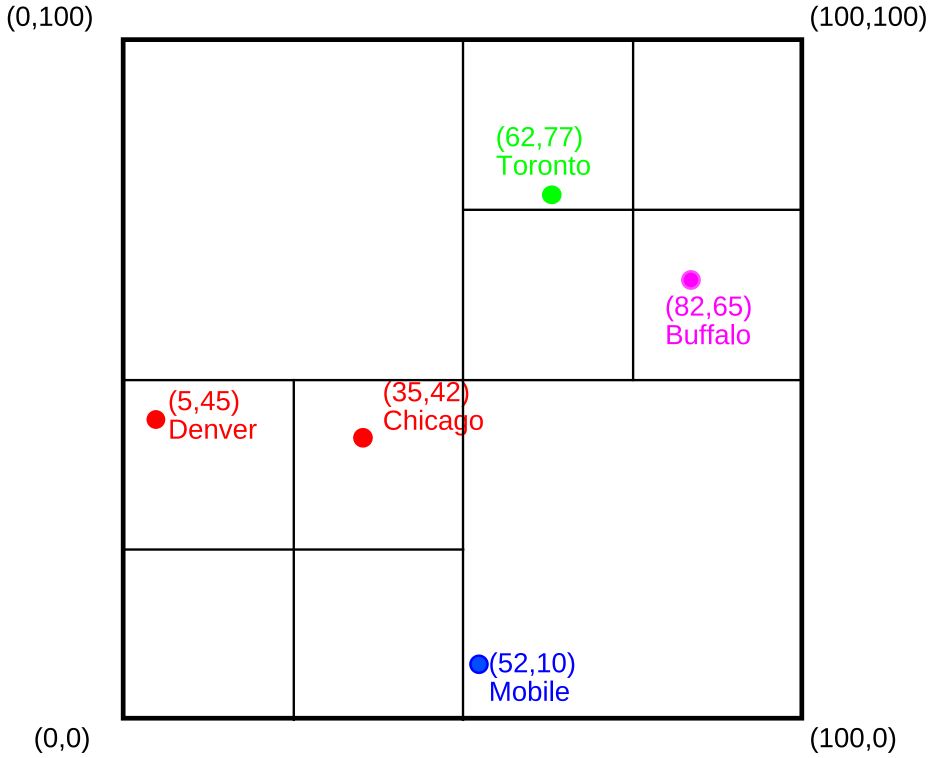
PR Quadtree

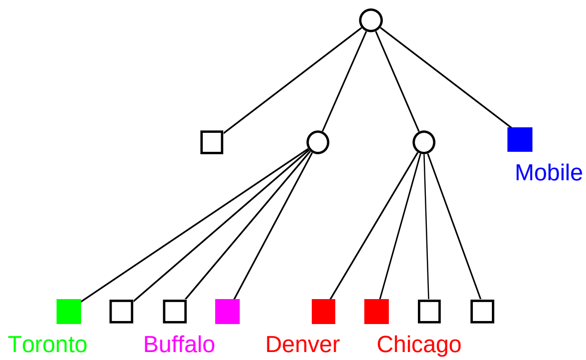
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points



PR Quadtree

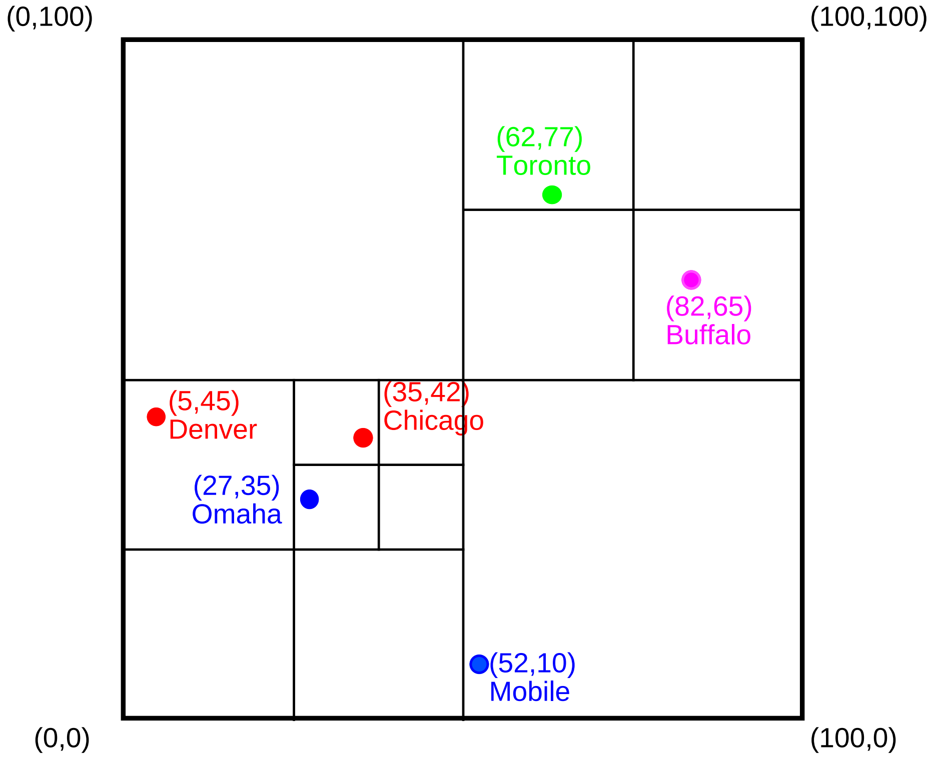
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

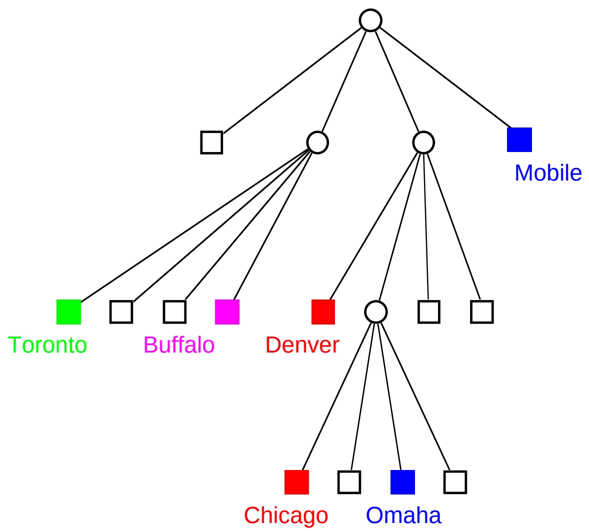




PR Quadtree

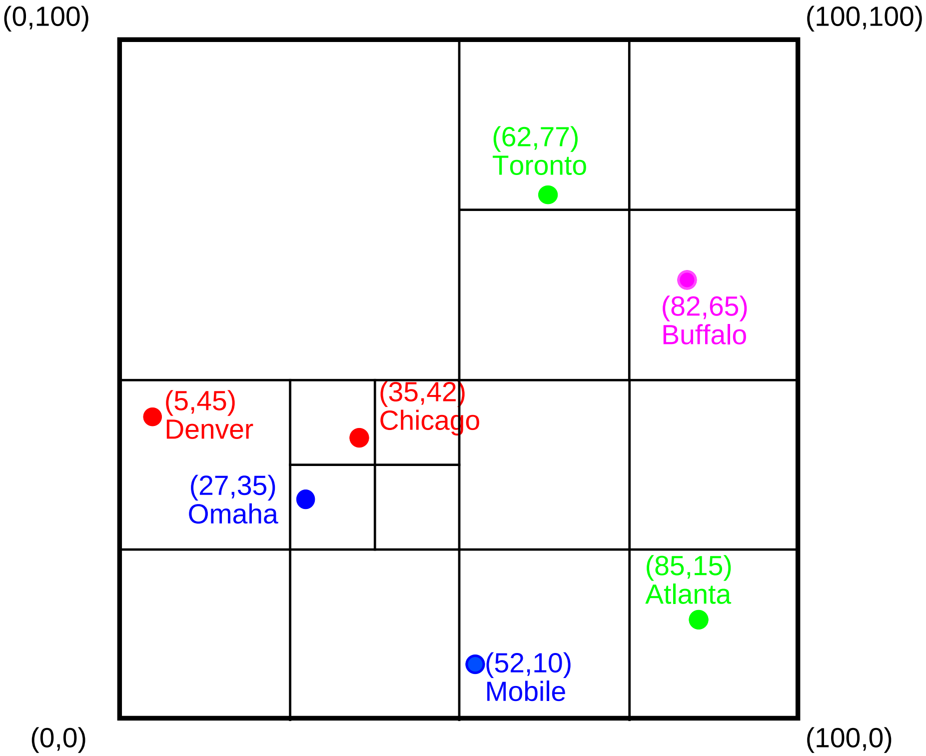
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

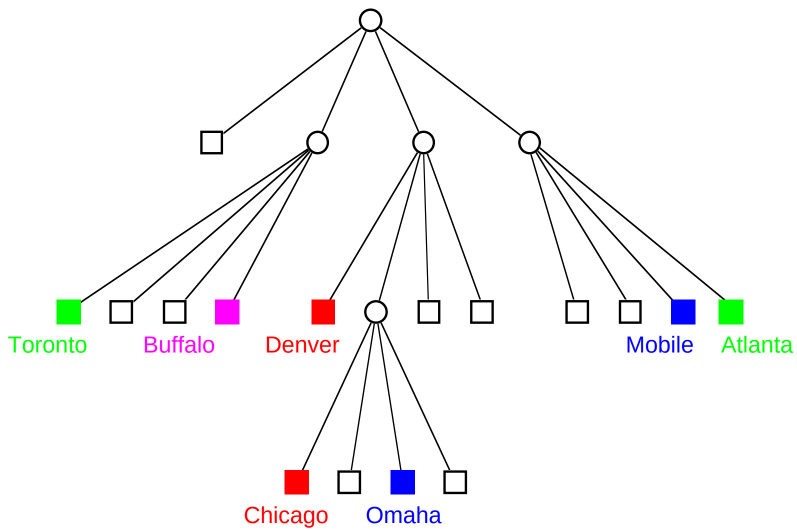




PR Quadtree

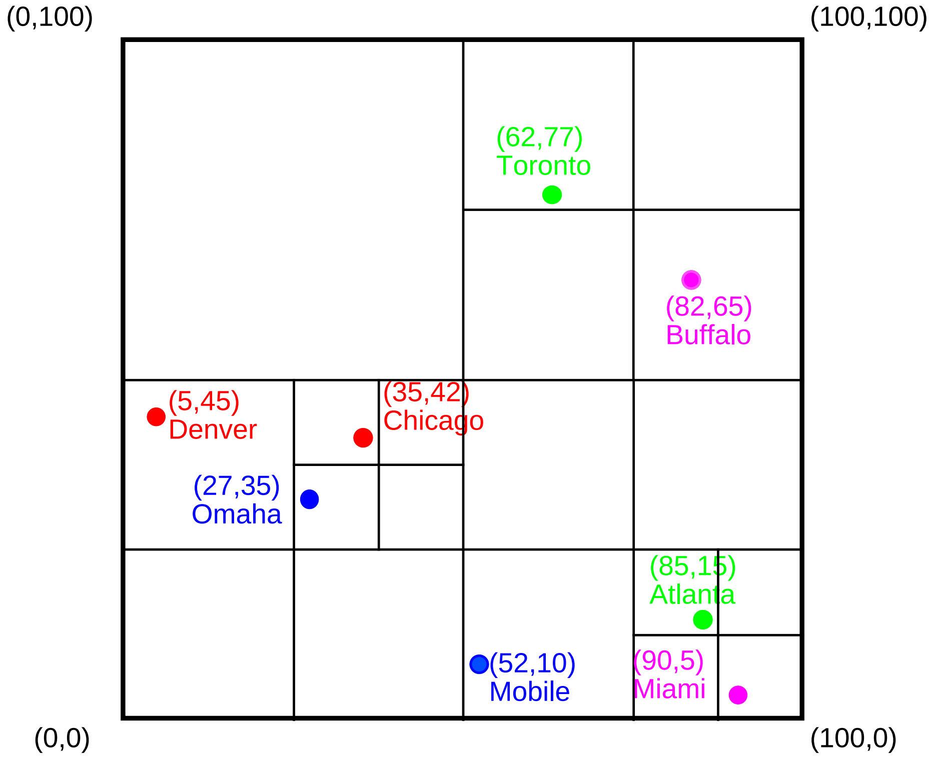
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

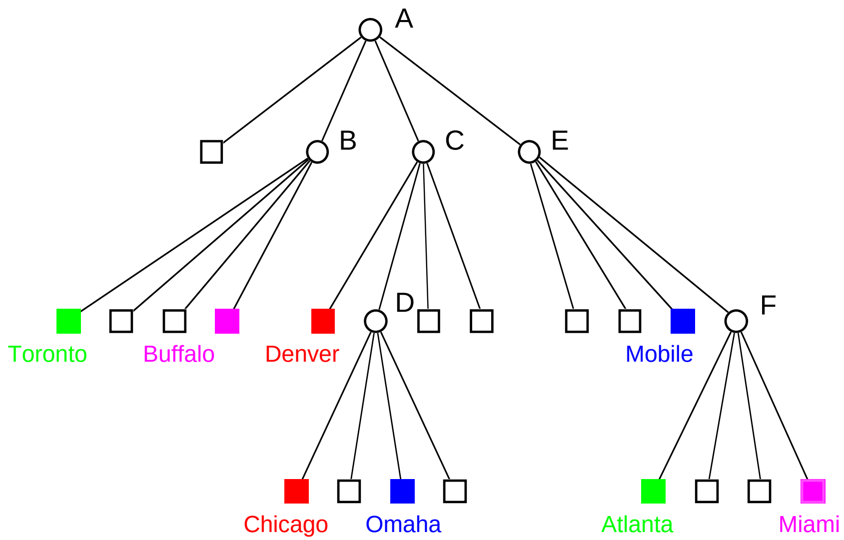




PR Quadtree

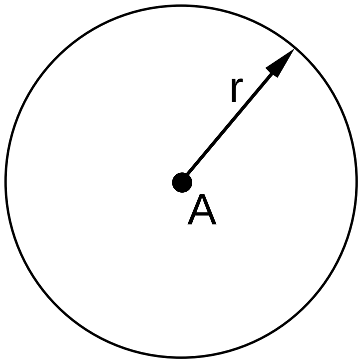
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points





Region Search

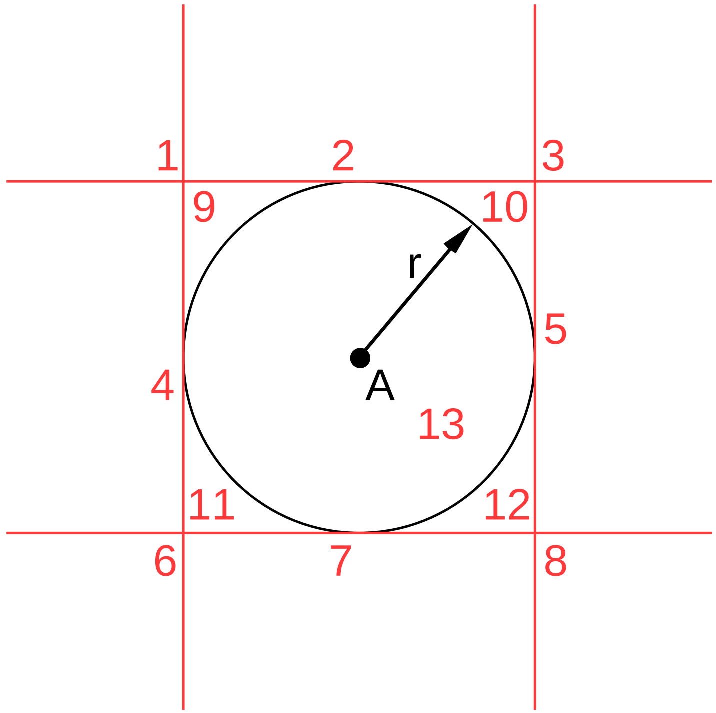
- Ex: Find all points within radius r of point A



- Use of quadtree results in pruning the search space

Region Search

- Ex: Find all points within radius r of point A

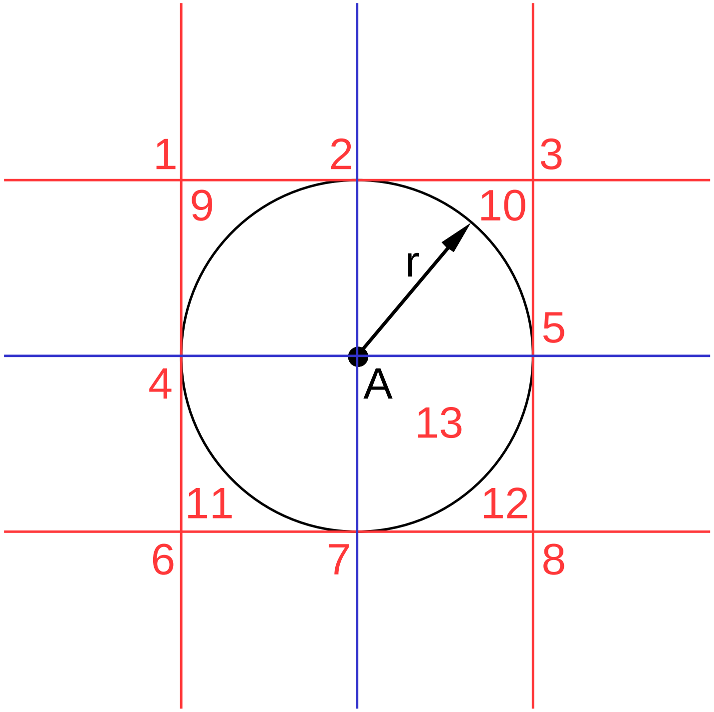


- Use of quadtree results in pruning the search space
- If a quadrant subdivision point p lies in a region l, then search the quadrants of p specified by l

1.	SE	5.	SW, NW	9.	All but NW	13.	All
2.	SE, SW	6.	NE	10.	All but NE		
3.	SW	7.	NE, NW	11.	All but SW		
4.	SE, NE	8.	NW	12.	All but SE		

Region Search

- Ex: Find all points within radius r of point A

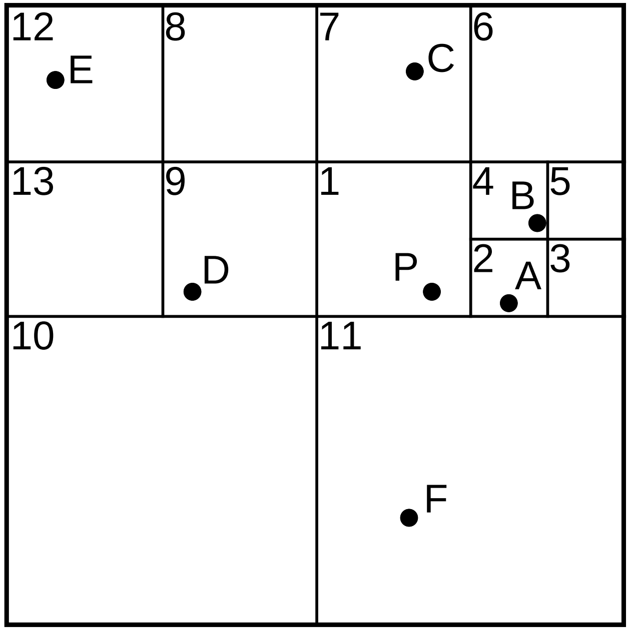


- Use of quadtree results in pruning the search space
- If a quadrant subdivision point p lies in a region l, then search the quadrants of p specified by l

1.	SE	5.	SW, NW	9.	All but NW	13.	All
2.	SE, SW	6.	NE	10.	All but NE		
3.	SW	7.	NE, NW	11.	All but SW		
4.	SE, NE	8.	NW	12.	All but SE		

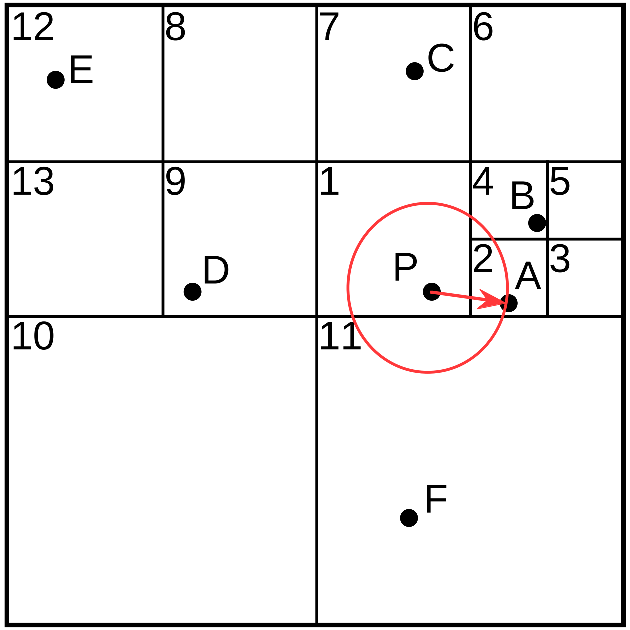
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



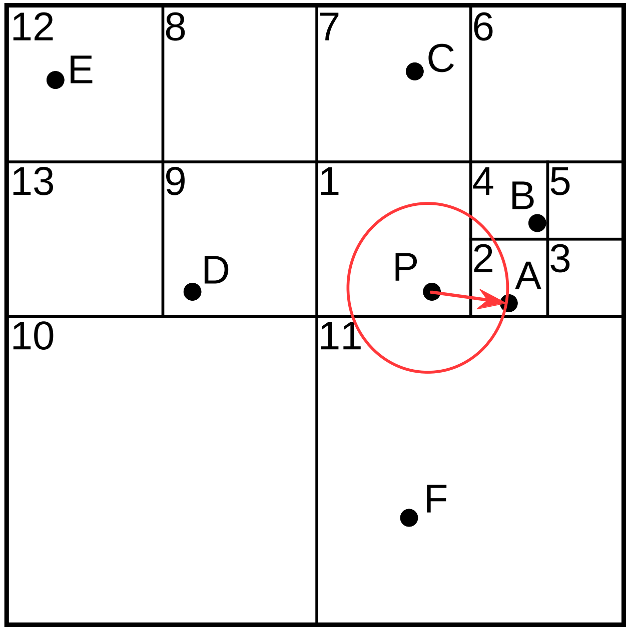
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A



Finding Nearest Object

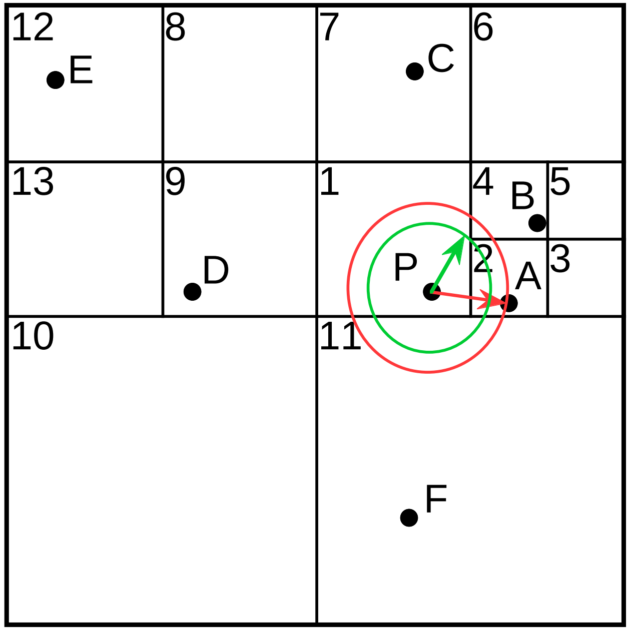
- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



1. start at block 2 and compute distance to P from A
2. ignore block 3, even if nonempty, as A is closer to P than any point in 3

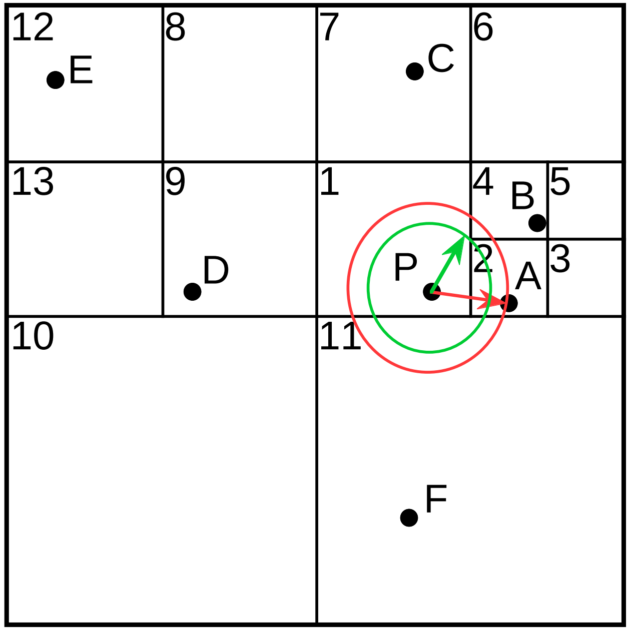
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



1. start at block 2 and compute distance to P from A
2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A

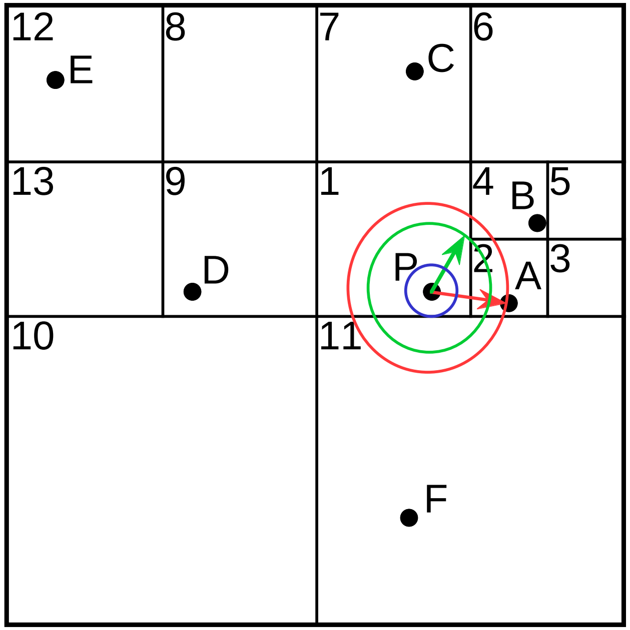
Finding Nearest Object



- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A
  2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
  3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
  4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A

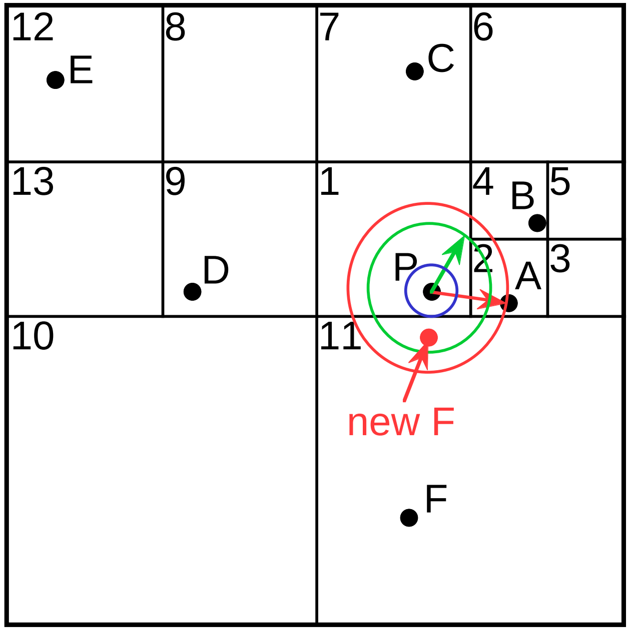
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



1. start at block 2 and compute distance to P from A
2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A
5. examine block 11 as the distance from P to the S border of 1 is shorter than distance from P to A; but, reject F as it is further from P than A

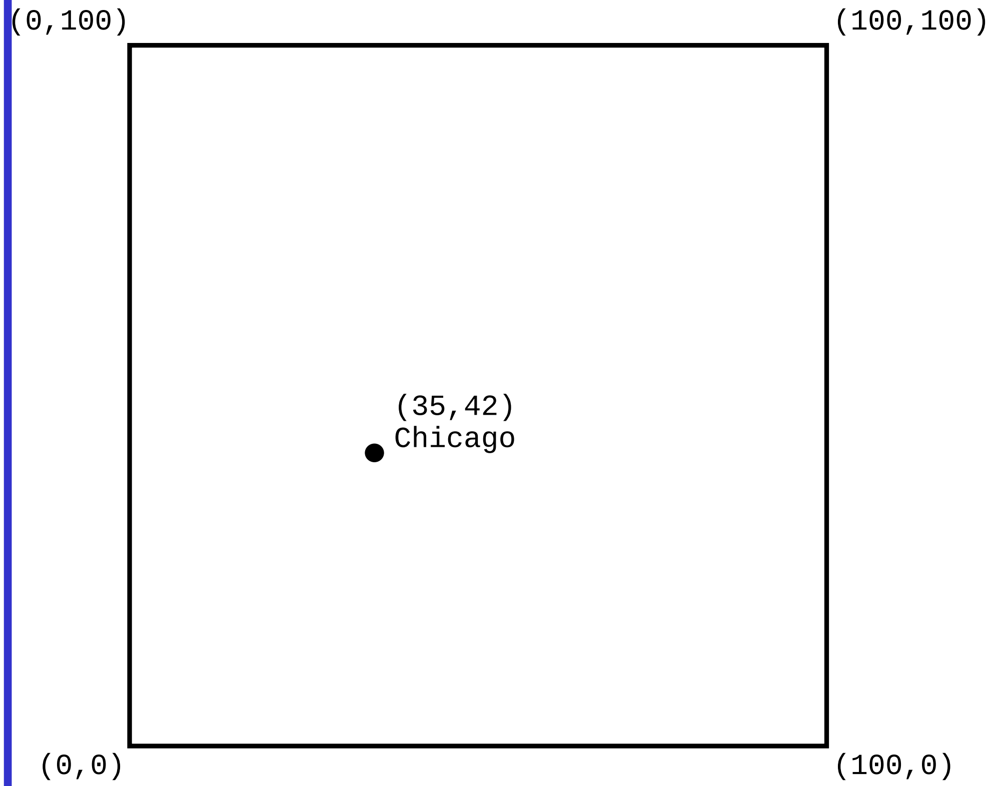
Finding Nearest Object

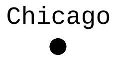


- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A
  2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
  3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
  4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A
  5. examine block 11 as the distance from P to the S border of 1 is shorter than distance from P to A; but, reject F as it is further from P than A
- If F was moved, a better order would have started with block 11, the southern neighbor of 1, as it is closest to the new F

k-d tree (Bentley)

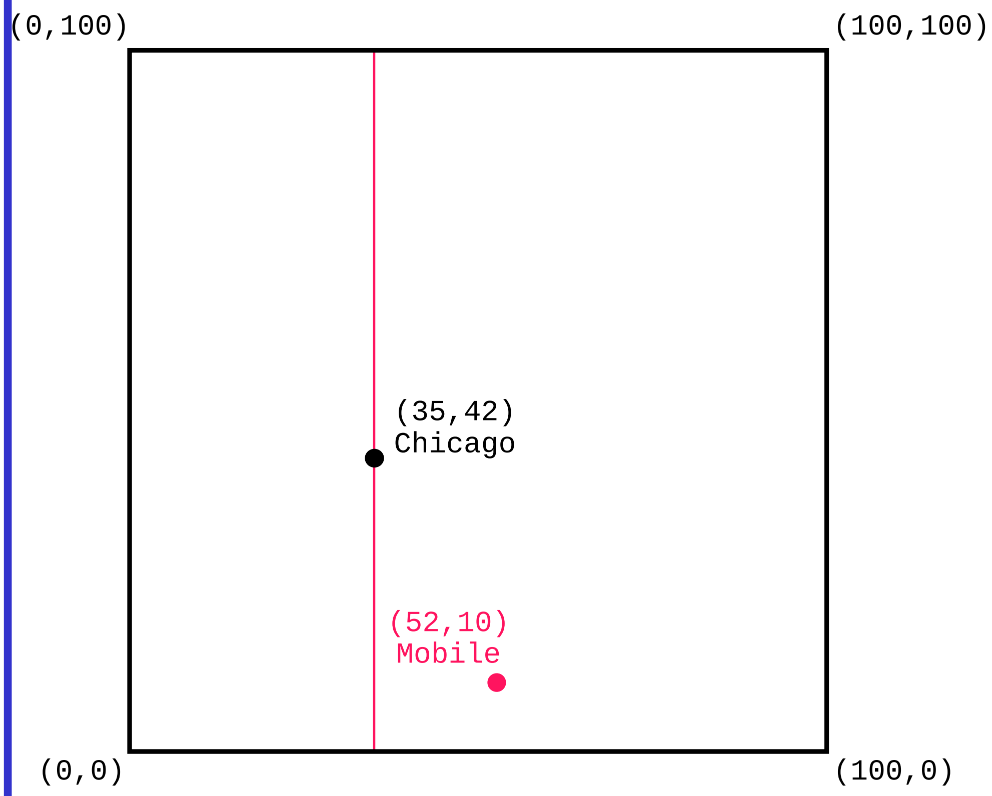
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered





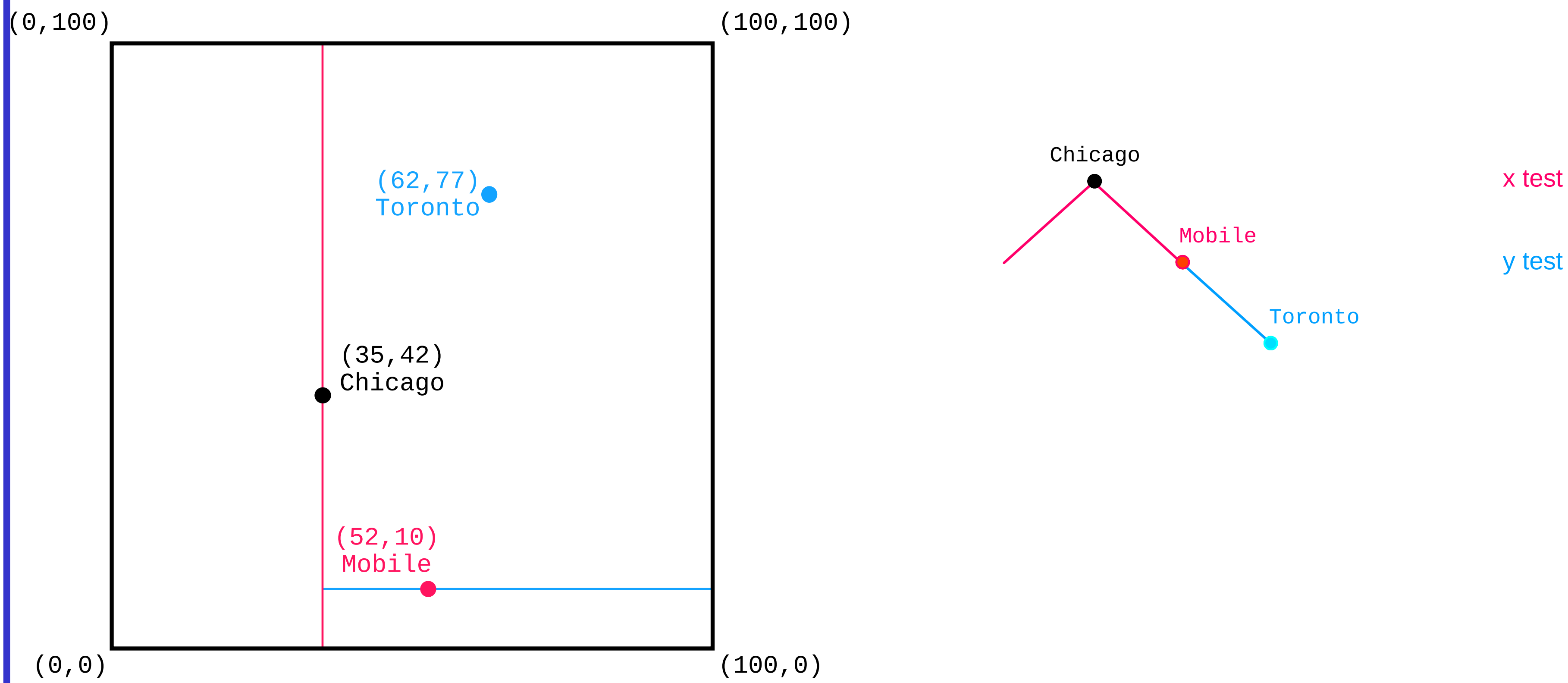
k-d tree (Bentley)

- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered



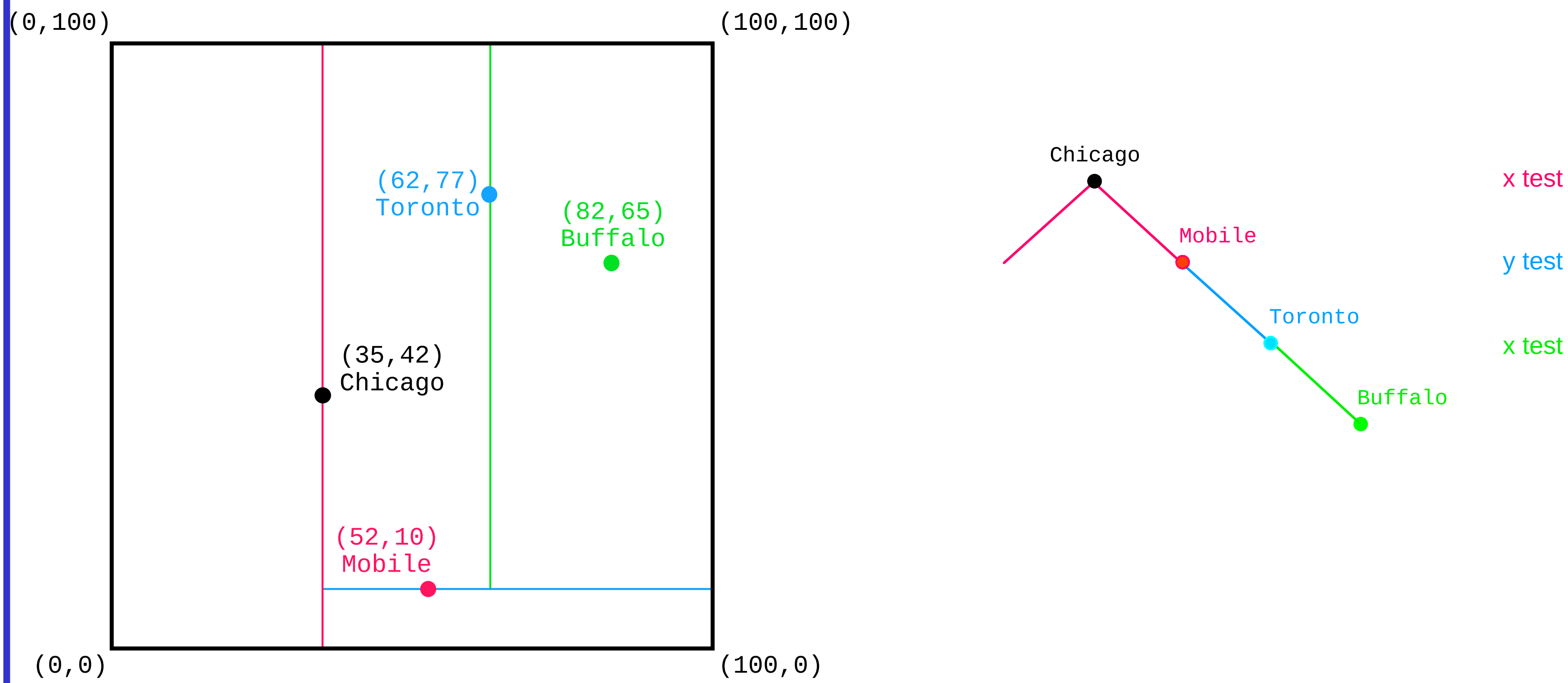
k-d tree (Bentley)

- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered



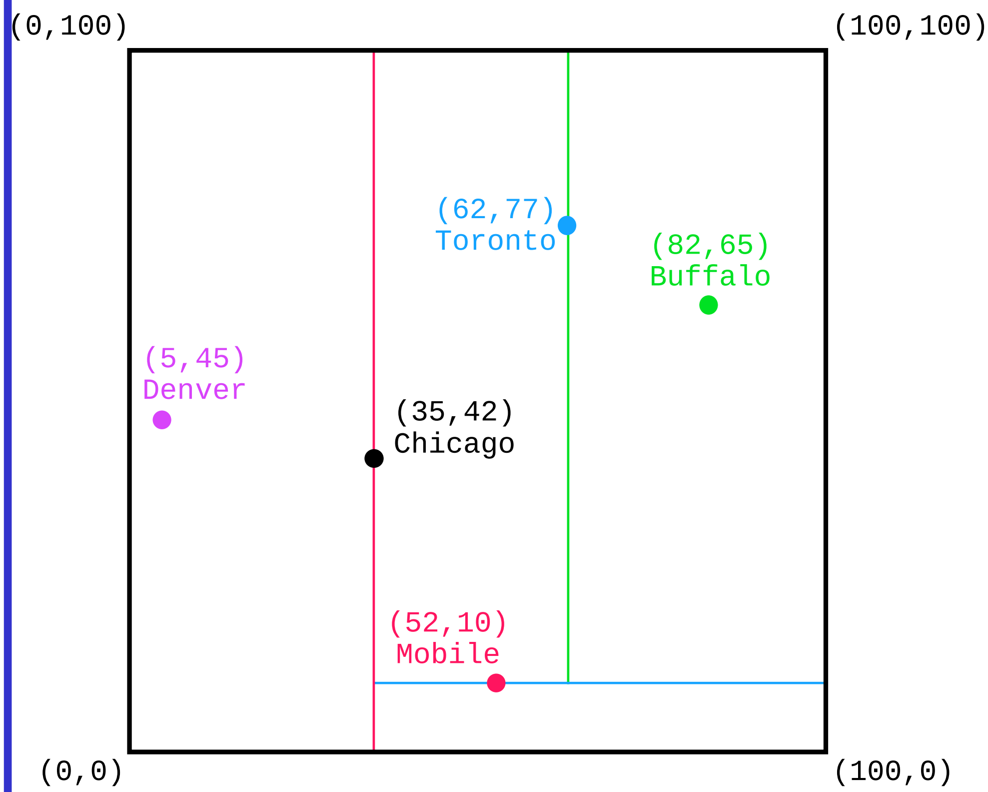
k-d tree (Bentley)

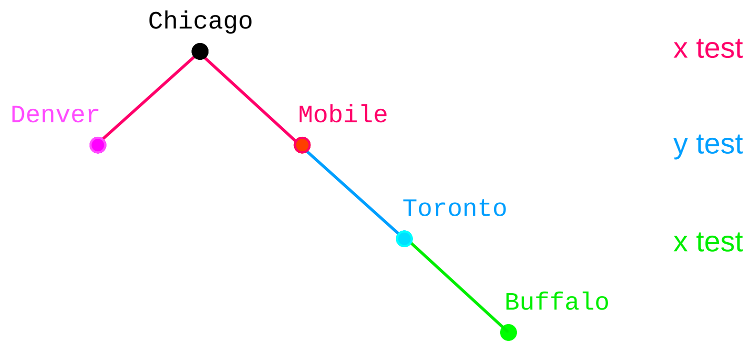
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered



k-d tree (Bentley)

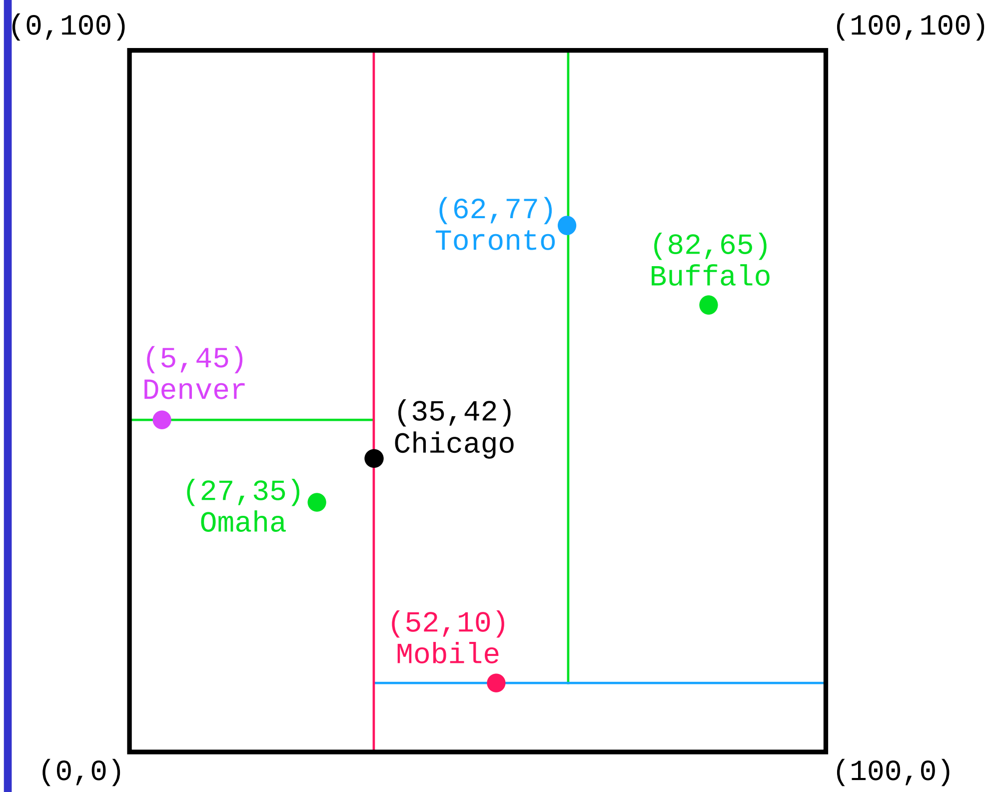
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

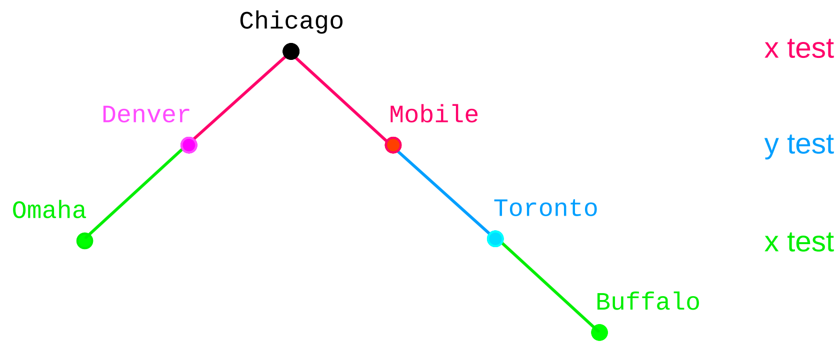




k-d tree (Bentley)

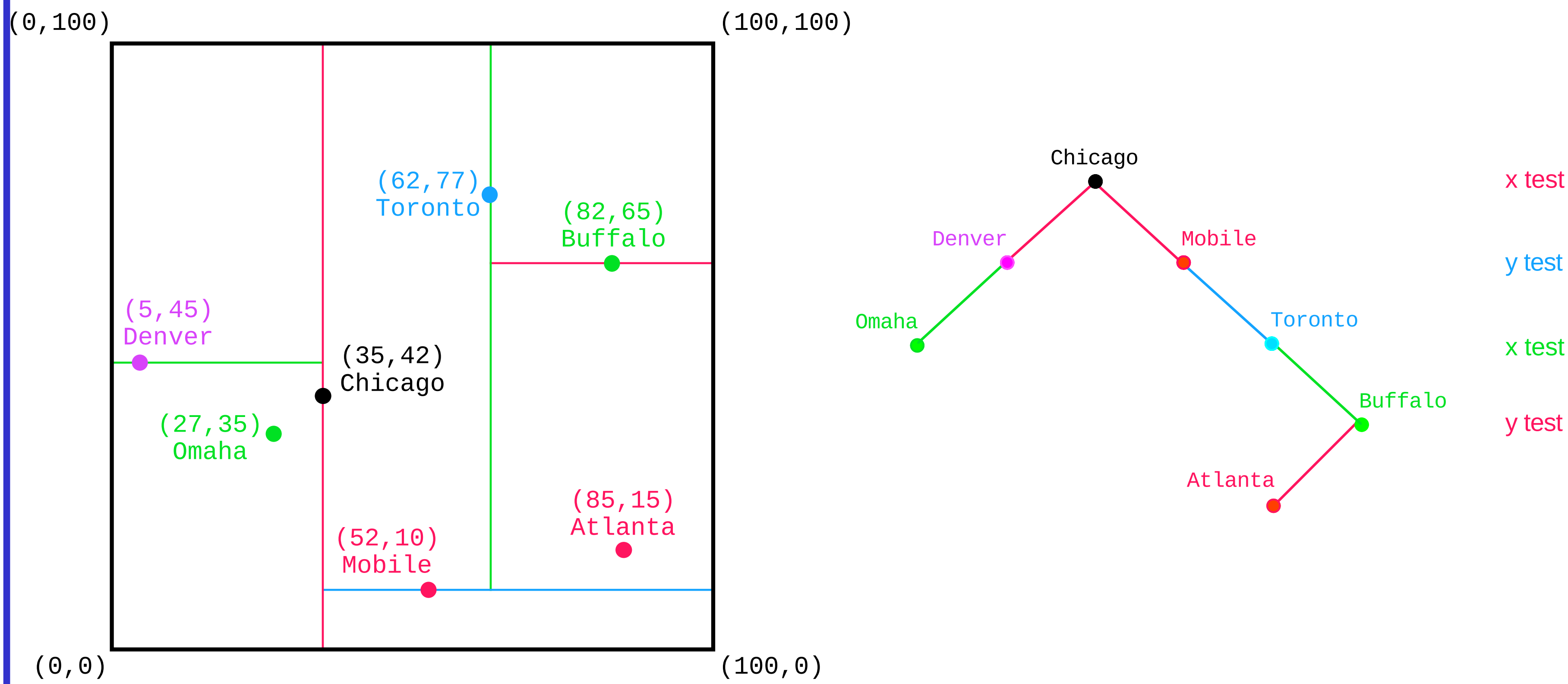
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered





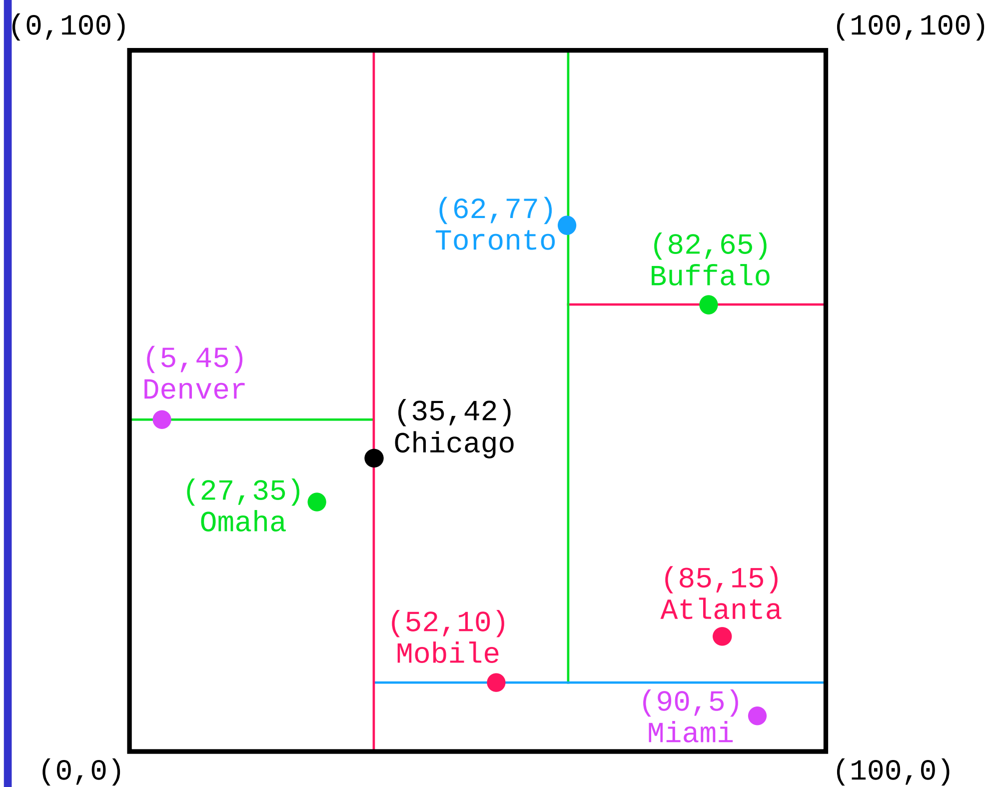
k-d tree (Bentley)

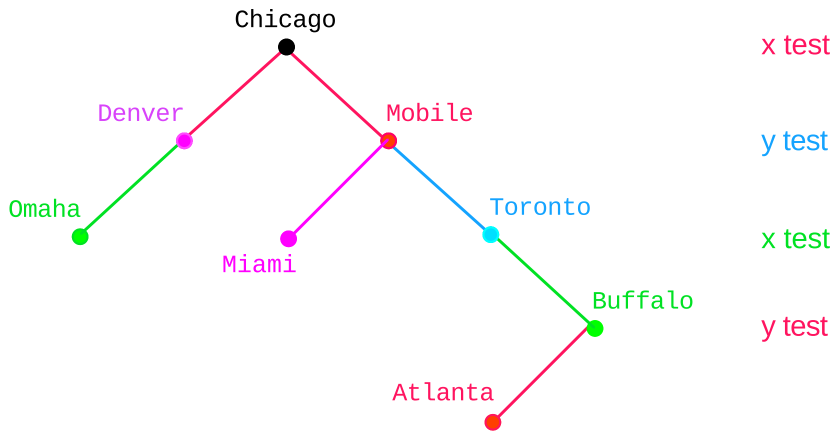
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered



k-d tree (Bentley)

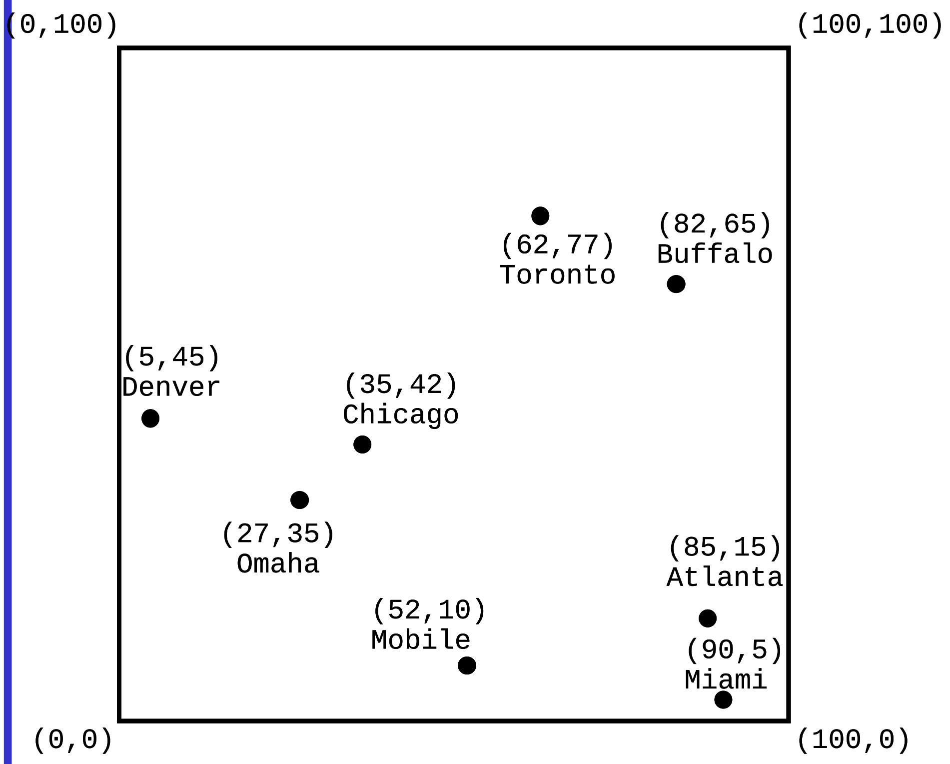
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered





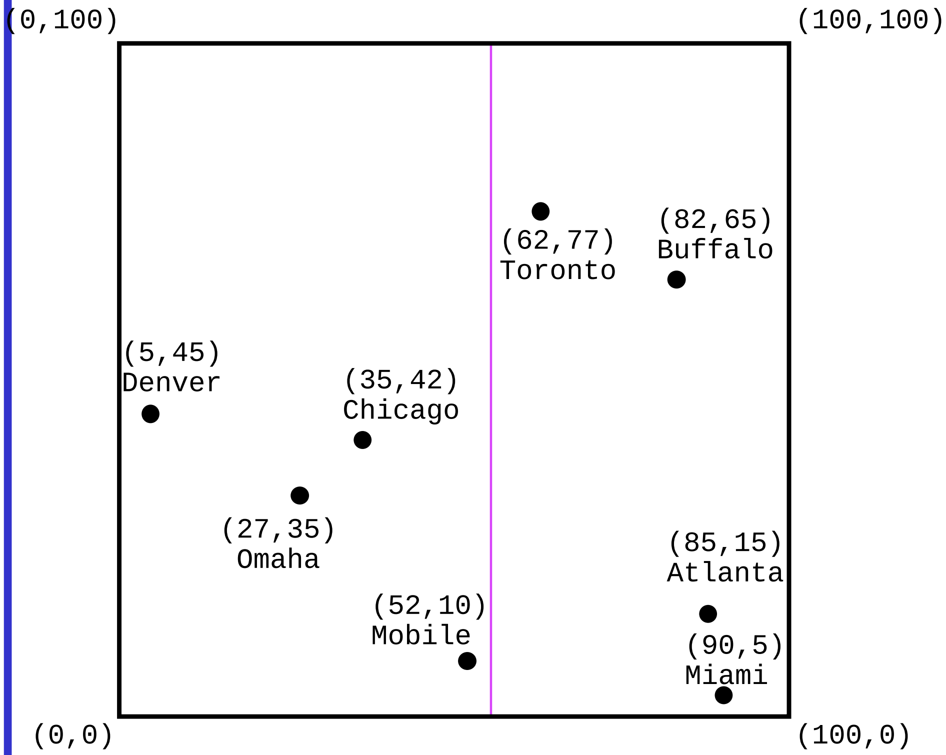
Adaptive k-d tree

- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum



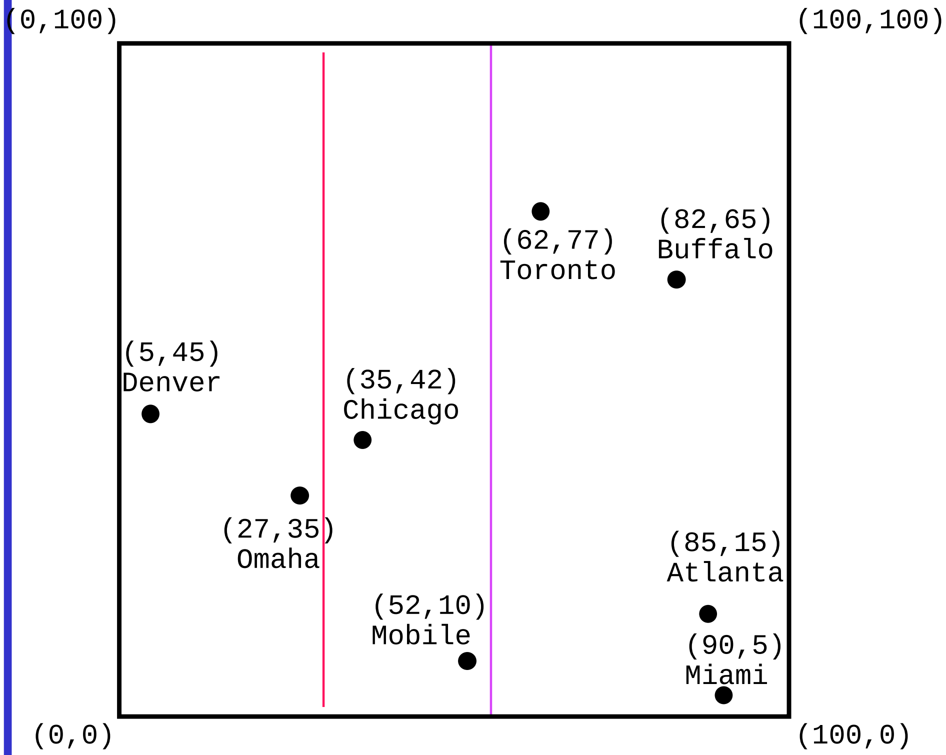
Adaptive k-d tree

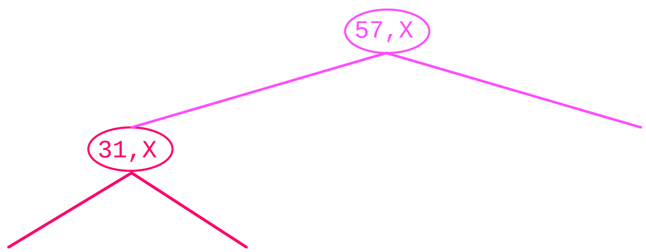
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum



Adaptive k-d tree

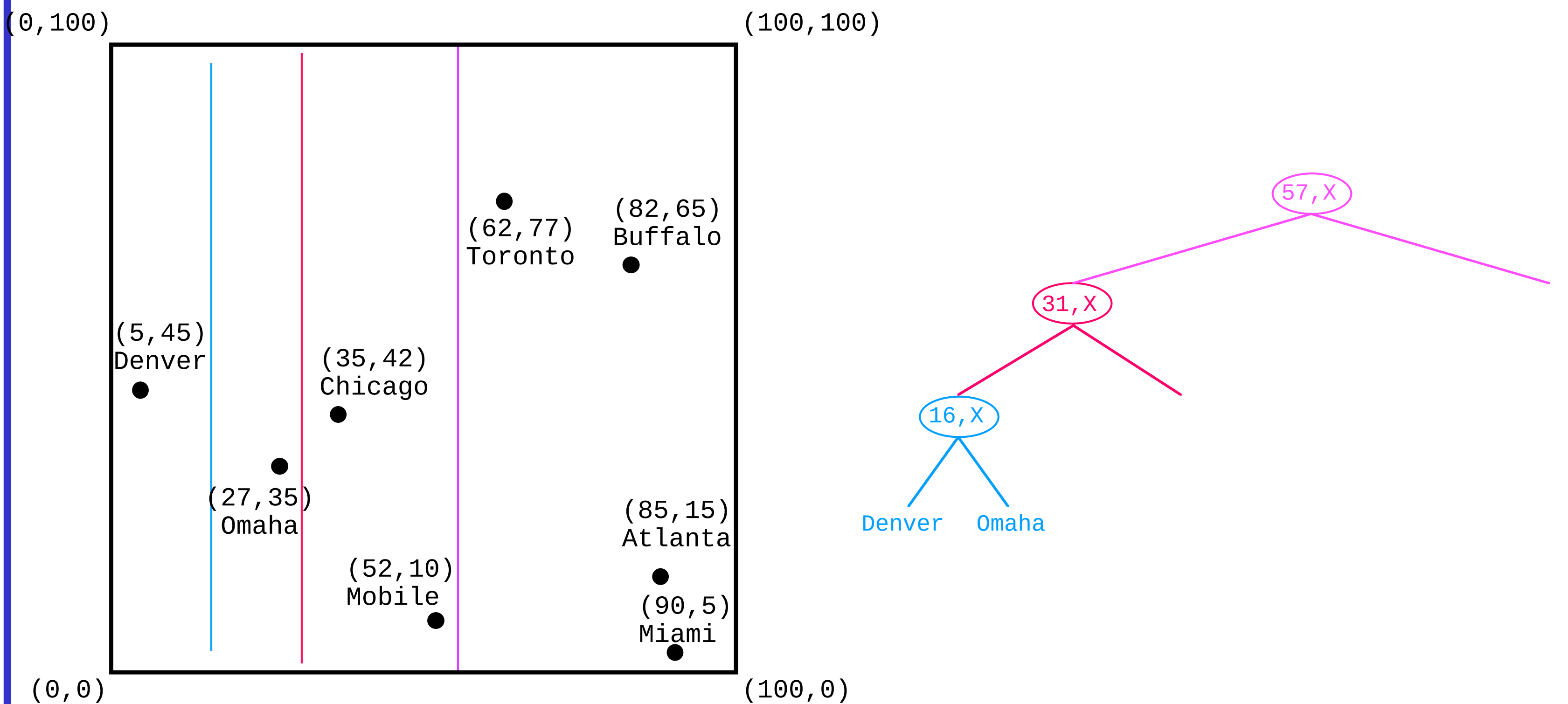
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum





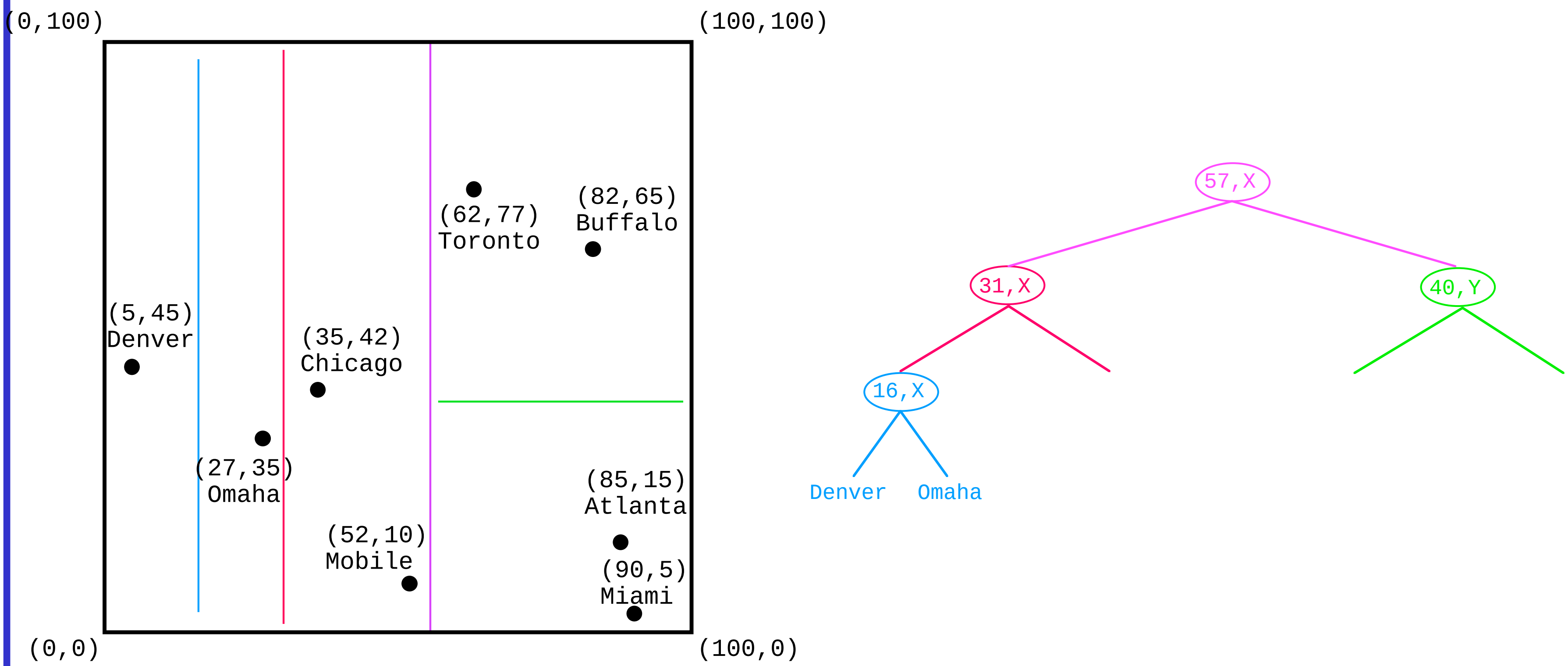
Adaptive k-d tree

- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum



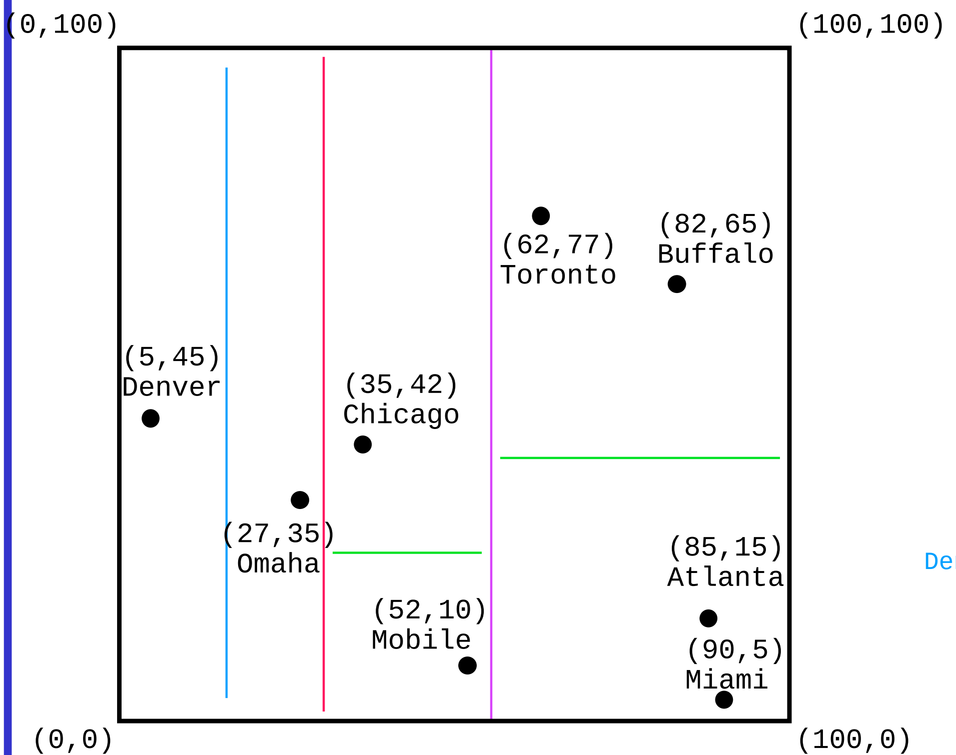
Adaptive k-d tree

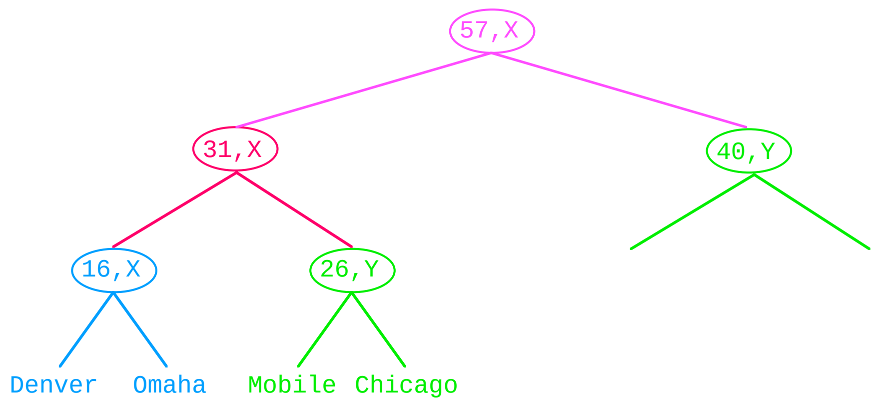
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum



Adaptive k-d tree

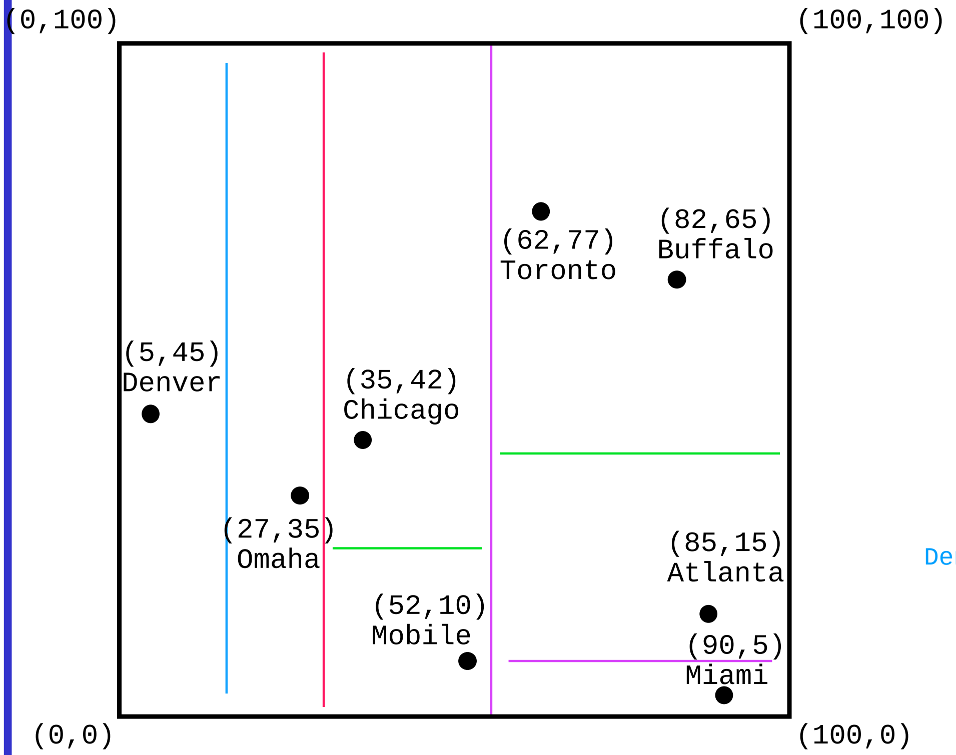
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

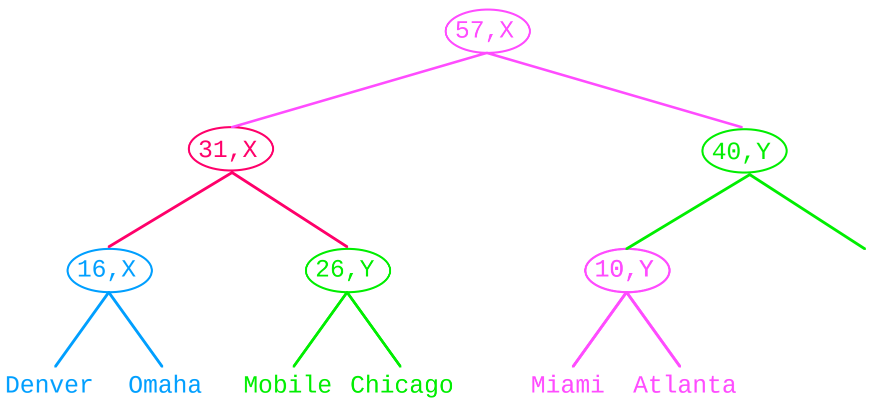




Adaptive k-d tree

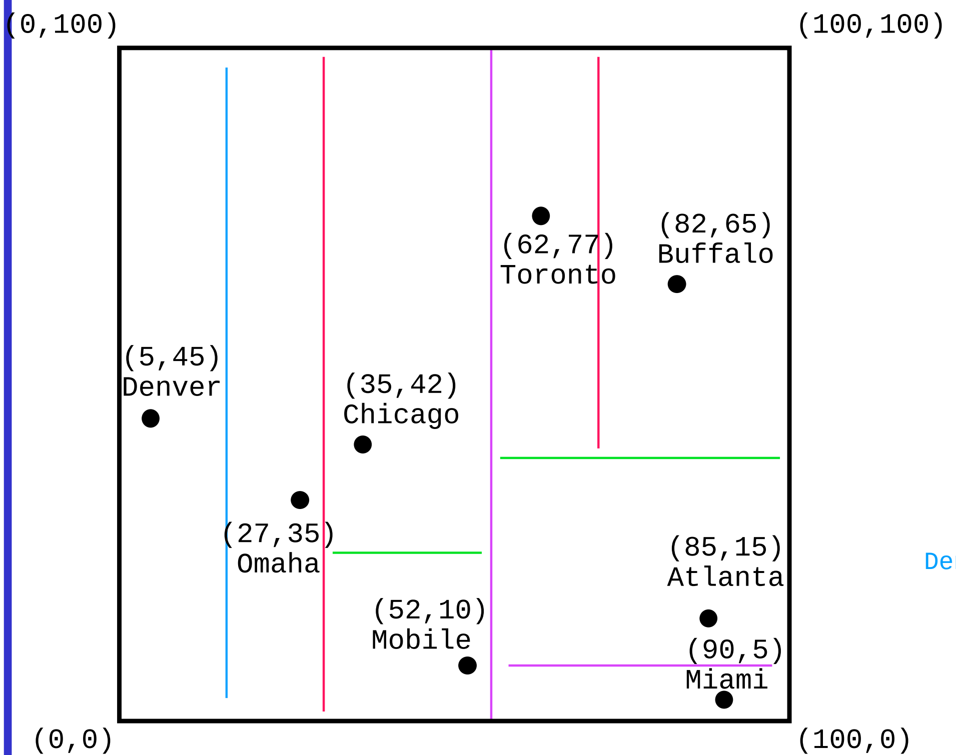
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

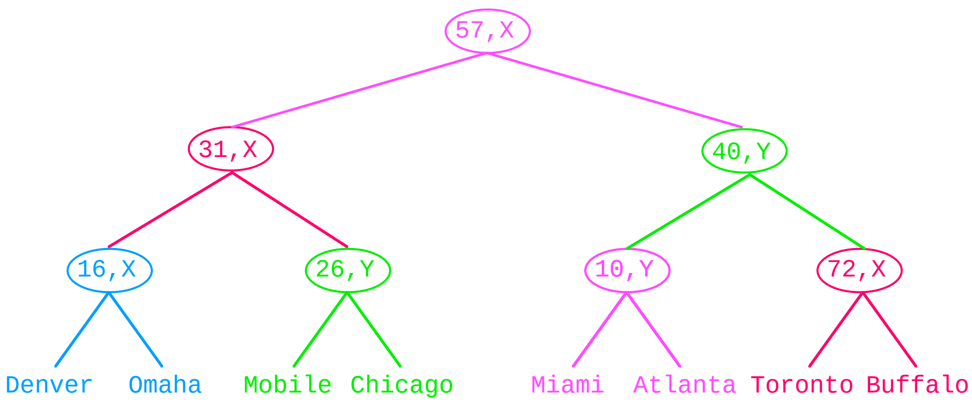




Adaptive k-d tree

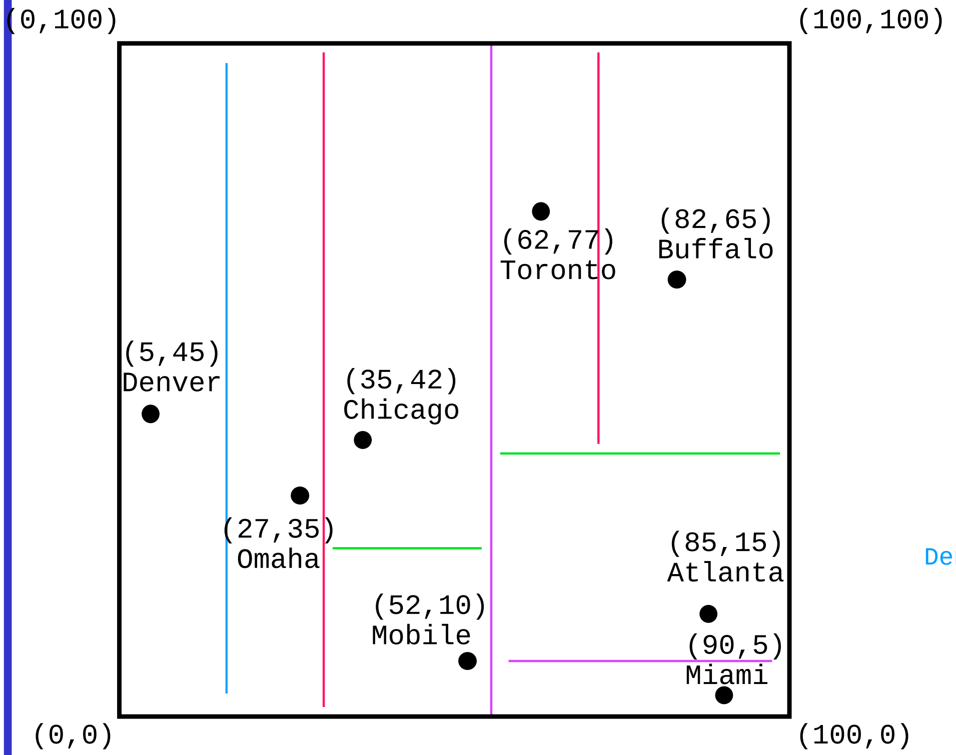
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

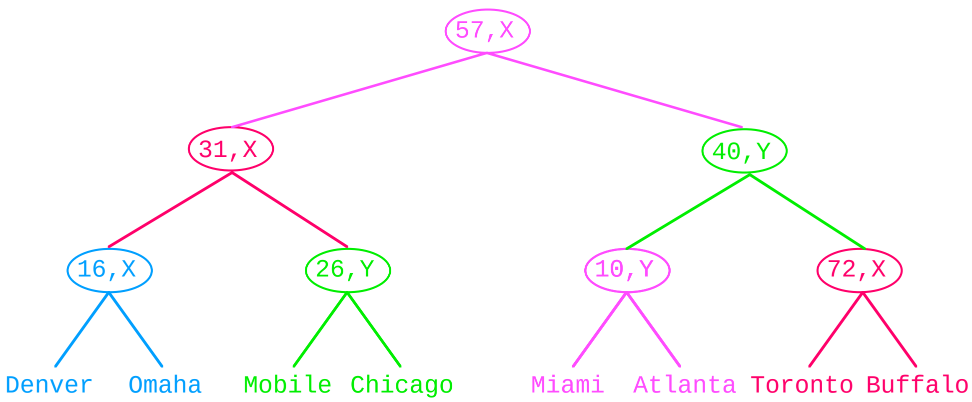




Adaptive k-d tree

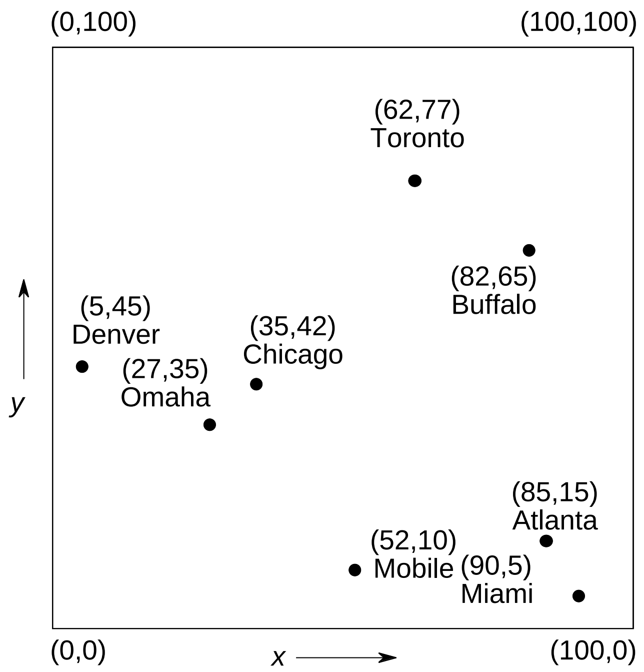
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum





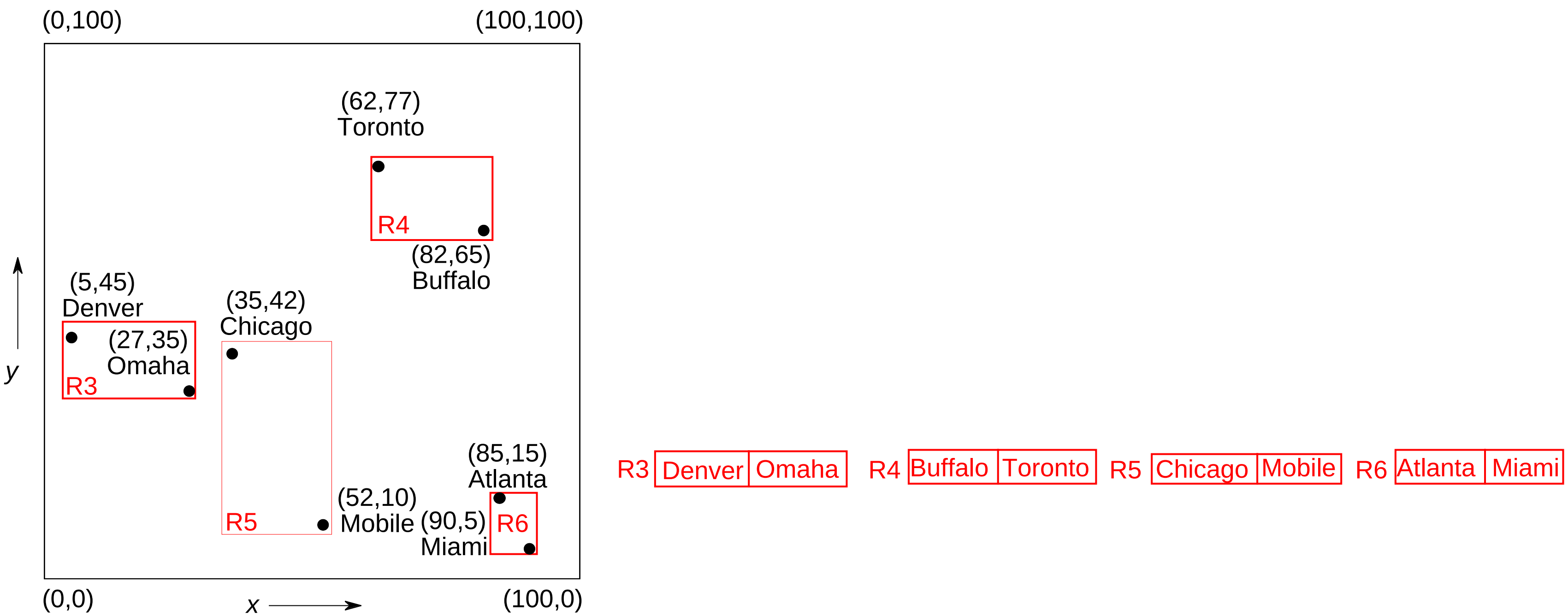
Minimum Bounding Rectangles: R-tree (Guttman)

- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



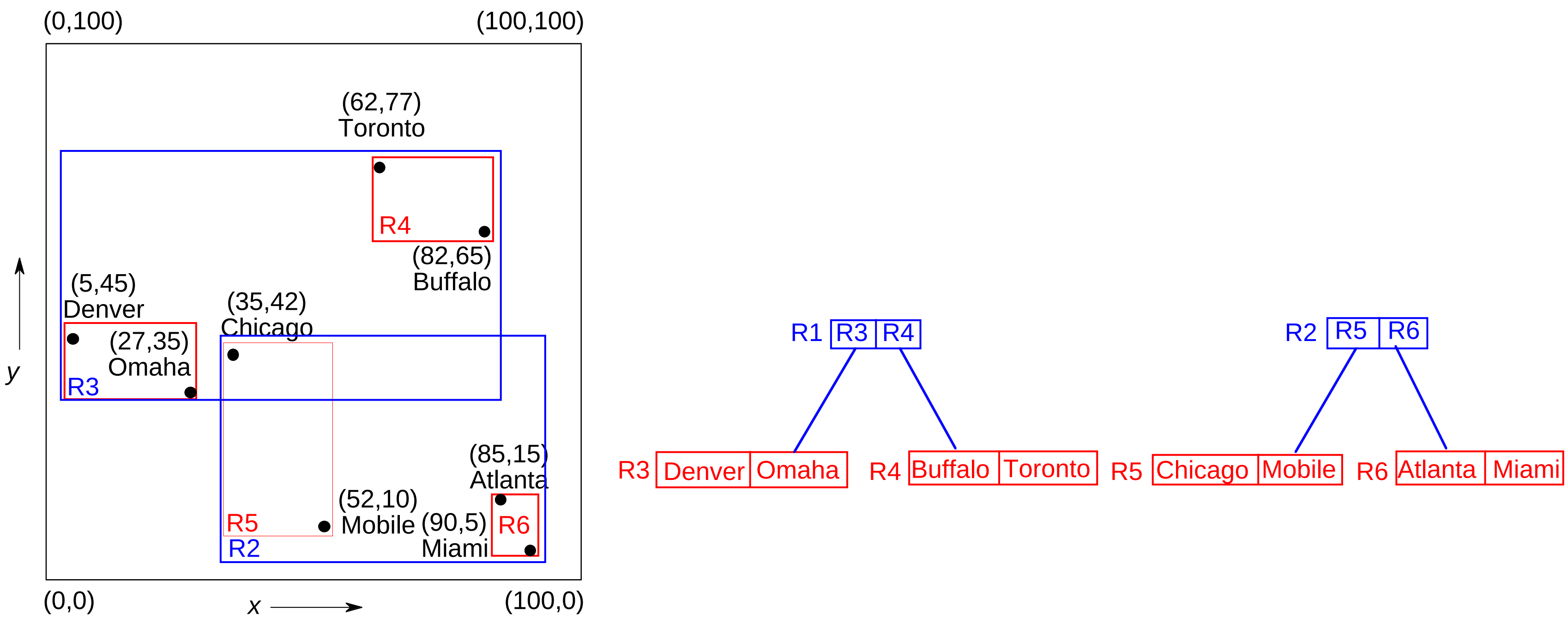
Minimum Bounding Rectangles: R-tree (Guttman)

- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



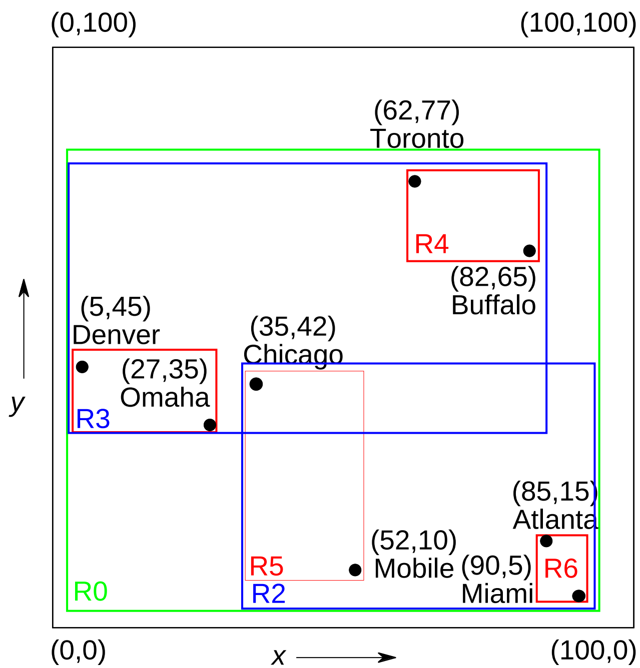
Minimum Bounding Rectangles: R-tree (Guttman)

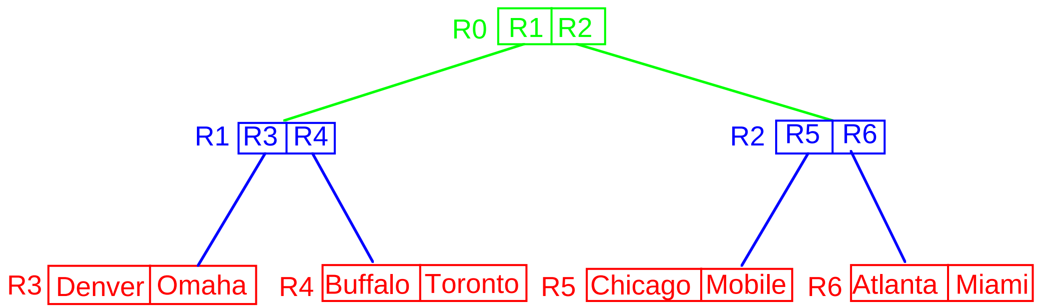
- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



Minimum Bounding Rectangles: R-tree (Guttman)

- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



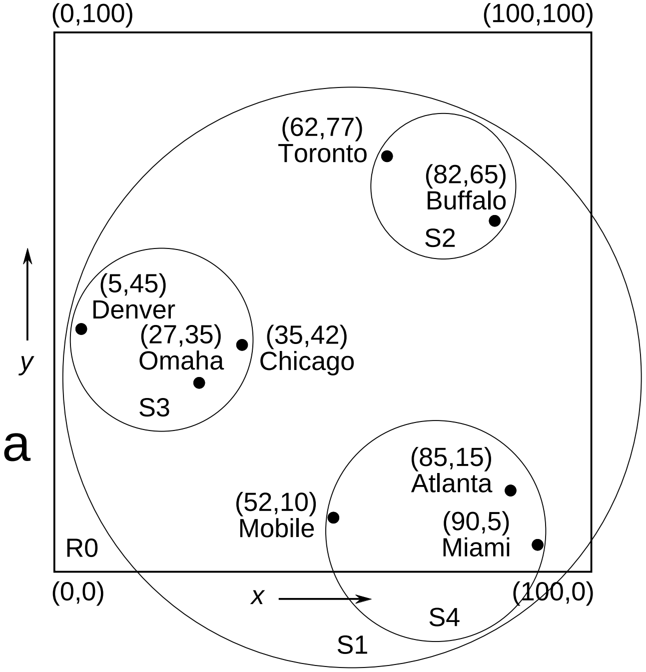


R*-tree (Beckmann et al.)

- Goal: minimize overlap for leaf nodes and area increase for nonleaf nodes
- Changes from R-tree:
  1. insert into leaf node p for which 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 (e.g., farthest from centroid)
     - known as 'forced reinsertion' and similar to 'deferred splitting' or 'rotation' in B-trees
  3. in case of true overflow, use a two-stage process (goal: low coverage)
     - determine axis along which the split takes place
       a. sort bounding boxes for each axis on low/high edge to get $2d$ lists for d-dimensional data
       b. choose axis yielding lowest sum of perimeters for splits based on sorted orders
     - determine position of split
       a. position where overlap between two nodes is minimized
       b. resolve ties by minimizing total area of bounding boxes
- Works very well but takes time due to forced reinsertion

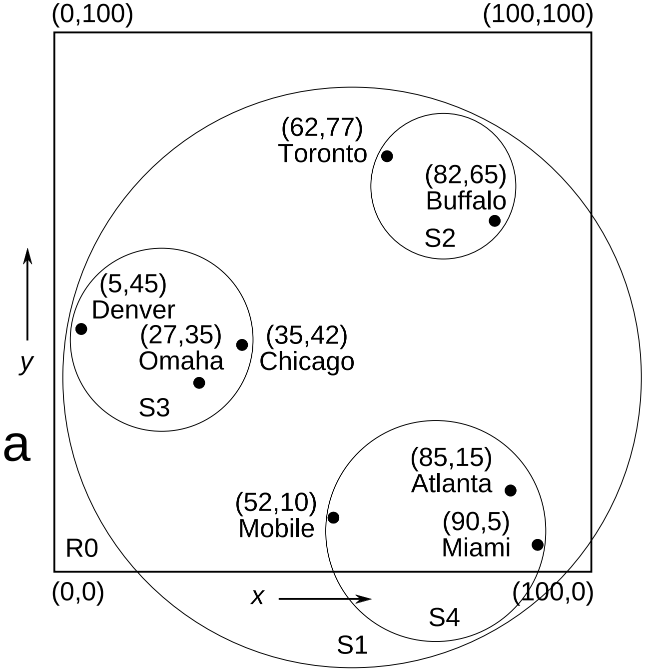
Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

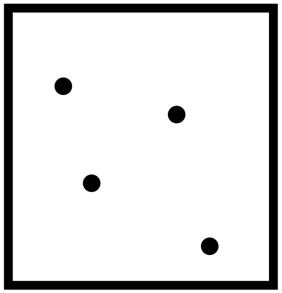


Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

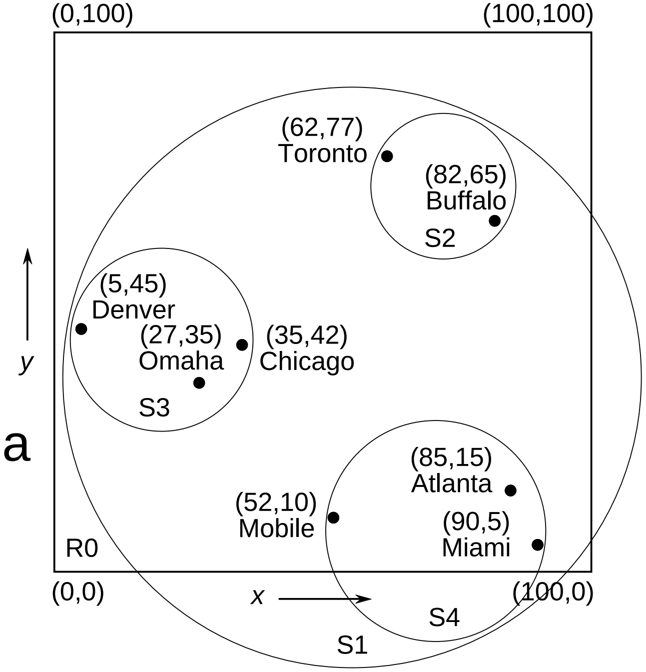


- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes

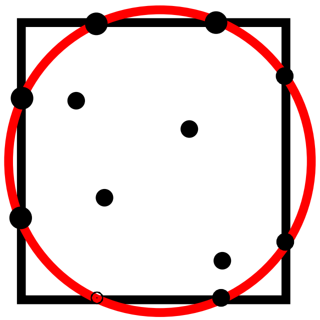


Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

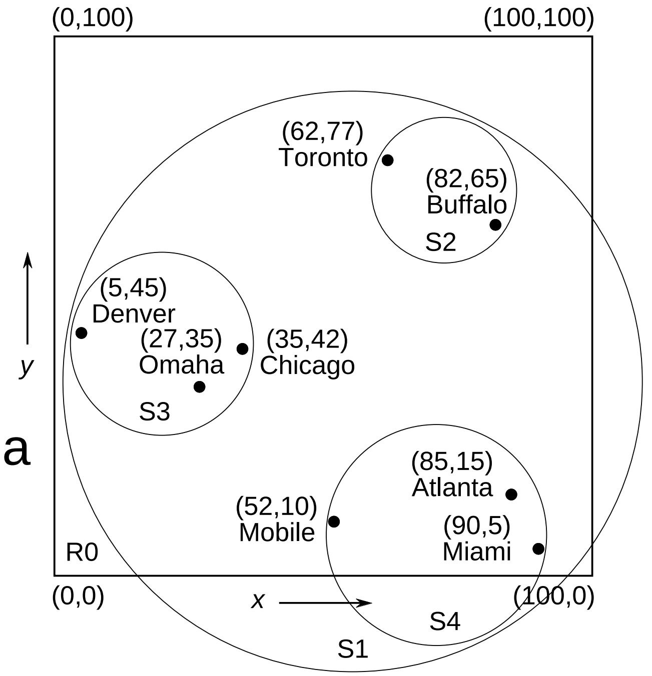


- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes

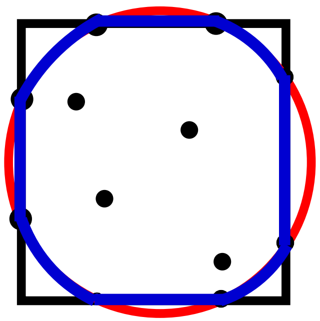


Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

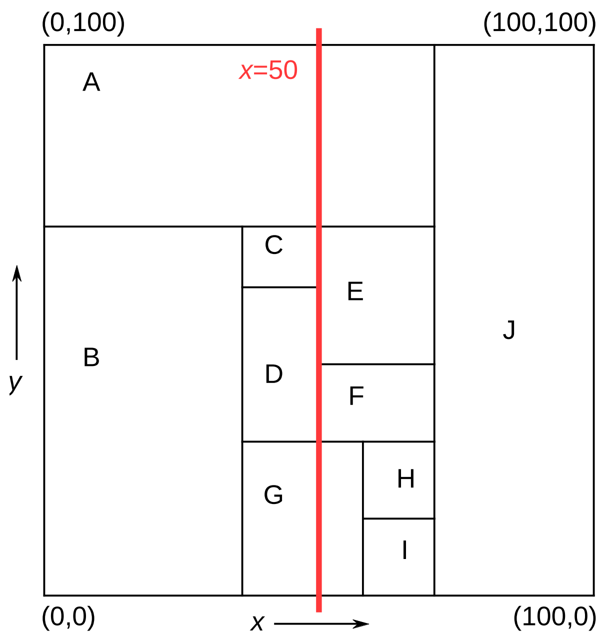


- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes



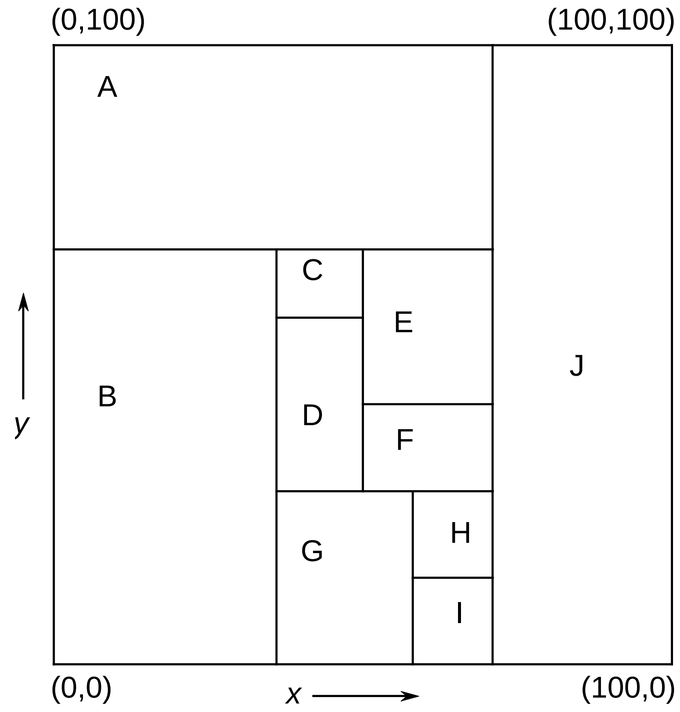
K-D-B-tree (Robinson)

- 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
- Aggregate blocks of k-d tree partition of space into nodes of 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: to get area covered by object, must retrieve all cells it occupies



Hybrid tree (Chakrabarti/Mehrotra)

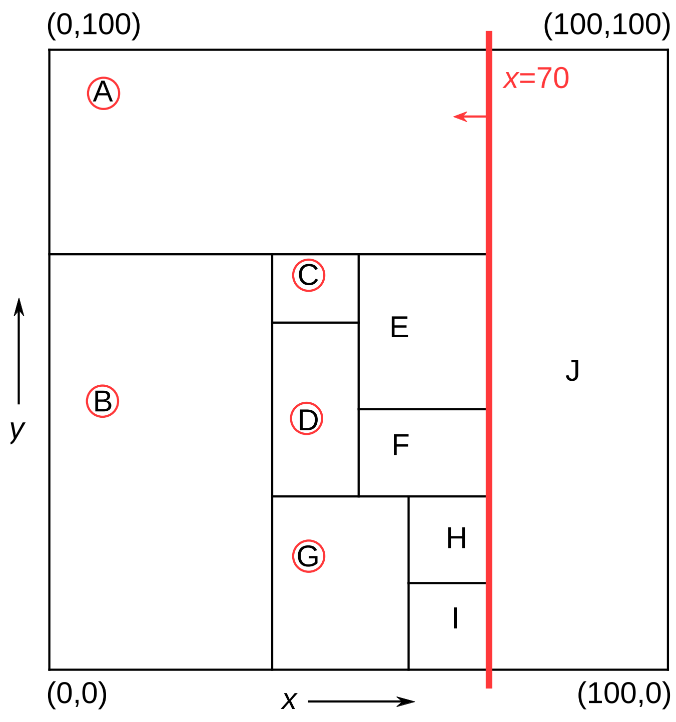
1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement



Hybrid tree (Chakrabarti/Mehrotra)

1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement

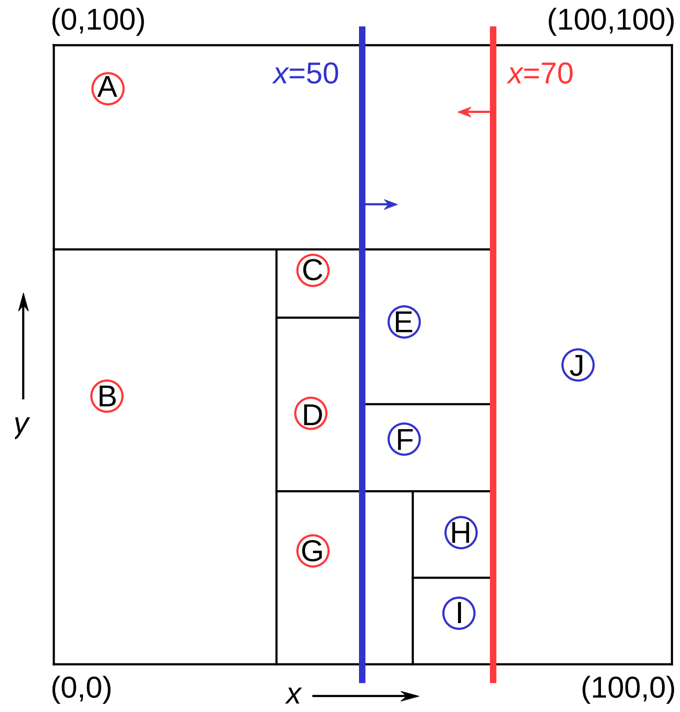
2. Add two partition lines at $x = 70$ for region low
   a. A, B, C, D, and G with region low



Hybrid tree (Chakrabarti/Mehrotra)

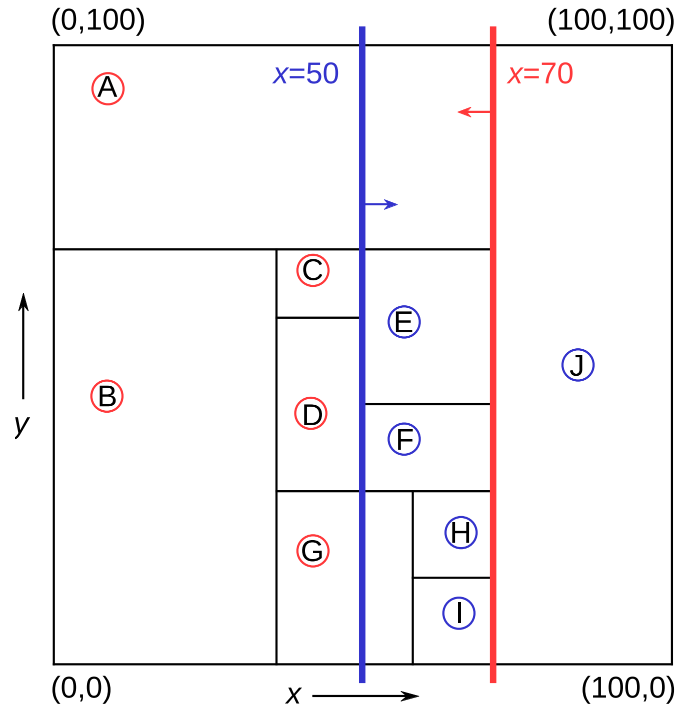
1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement

2. Add two partition lines at $x = 70$ for region low and $x = 50$ for region $high$
   a. A, B, C, D, and G with region low
   b. E, F, H, I, and J with region $high$



Hybrid tree (Chakrabarti/Mehrotra)

1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement



2. Add two partition lines at $x = 70$ for region low and $x = 50$ for region $high$
   a. A, B, C, D, and G with region low
   b. E, F, H, I, and J with region $high$

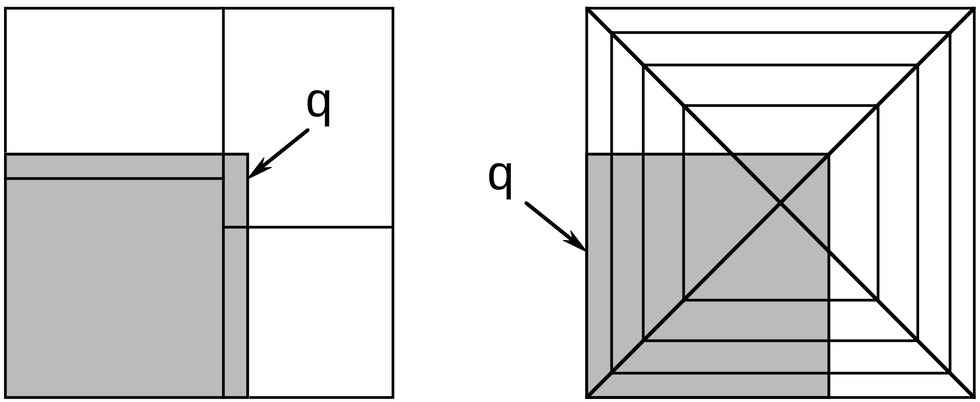
3. Associating two partition lines with each partition region is analogous to associating a bounding box with each region (also spatial k-d tree)
   - similar to bounding box in R-tree but not minimum bounding box
   - store approximation of bounding box by quantizing coordinate value along each dimension to b bits for a total of $2bd$ bits for each box thereby reducing fanout of each node (Henrich)

Avoiding Overlapping All of the Leaf Blocks

- Assume uniformly-distributed data
  1. most data points lie near the boundary of the space that is being split
     - Ex: for $d = 20$, 98.5% of the points lie within 10% of the surface
     - Ex: for $d = 100$, 98.3% of the points lie within 2% of the surface
  2. rarely will all of the dimensions be split even once
     - Ex: assuming at least $M/2$ points per leaf node blocks, and at least one split along each dimension, then total number of points N must be at least $2^d M/2$
     - if $d = 20$ and $M = 10$, then N must be at least 5 million to split along all dimensions once
  3. if each region is split at most once, and without loss of generality, split is in half, then query region usually intersects all the leaf node blocks
     - query selectivity of 0.01% for $d = 20$ leads to 'side length of query region'=0.63 which means that it intersects all the leaf node blocks
     - implies a range query will visit each leaf node block
- One solution: use a 3-way split along each dimension into three parts of proportion r, $1 - 2r$, and r
- Sequential scan may be cheaper than using an index due to high dimensions
  - We assume our data is not of such high dimensionality!

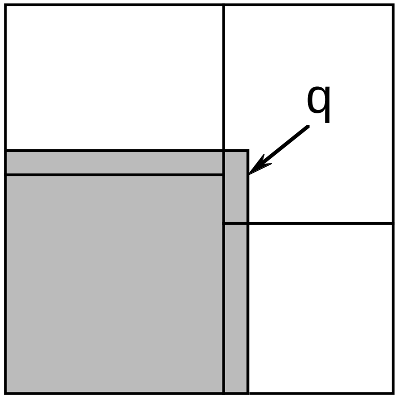
Pyramid Technique (Berchtold/Böhm/Kriegel)

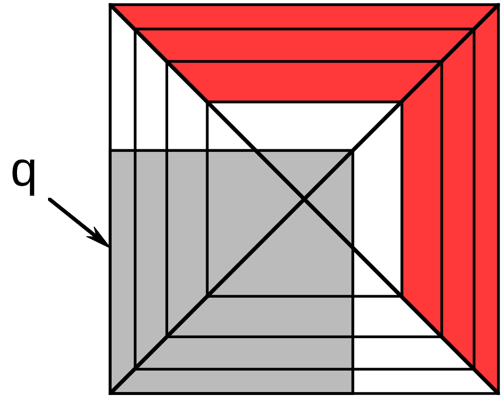
- Subdivide data space as if it is an onion by peeling off hypervolumes that are close to the boundary
- Subdivide hypercube into $2d$ pyramids having the center of the data space as the tip of their cones
- Each of the pyramids has one of the faces of the hypercube as its base
- Each pyramid is decomposed into slices parallel to its base
- Useful when query region side length is greater than half the width of the data space as won't have to visit all leaf node blocks



Pyramid Technique (Berchtold/Böhm/Kriegel)

- Subdivide data space as if it is an onion by peeling off hypervolumes that are close to the boundary
- Subdivide hypercube into $2d$ pyramids having the center of the data space as the tip of their cones
- Each of the pyramids has one of the faces of the hypercube as its base
- Each pyramid is decomposed into slices parallel to its base
- Useful when query region side length is greater than half the width of the data space as won't have to visit all leaf node blocks





- Pyramid containing q is the one corresponding to the coordinate i whose distance from the center point of the space is greater than all others
- Analogous to iMinMax method (Ooi/Tan/Yu/Bressan) with exception that iMinMax associates a point with its closest surface but the result is still a decomposition of the underlying space into $2d$ pyramids

Methods Based on a Sequential Scan

1. If neighbor finding in high dimensions must access every disk page at random, then a linear scan may be more efficient
   - advantage of sequential scan over hierarchical indexing methods is that actual I/O cost is reduced by being able to scan the data sequentially instead of at random as only need one disk seek
2. VA-file (Weber et al.)
   - use b_i bits per feature i to approximate feature
   - impose a d dimensional grid with $b = \sum_{i=1}^{d} b_i$ grid cells
   - sequentially scan all grid cells as a filter step to determine possible candidates which are then checked in their entirety via a disk access
   - VA-file is an additional representation in the form of a grid which is imposed on the original data
3. Other methods apply more intelligent quantization processes
   - VA+-file (Ferhatosmanoglu et al): decorrelate the data with KLT yielding new features and vary number of bits as well as use clustering to determine the region partitions
   - IQ-tree (Berchtold et al): hierarchical like an R-tree with unordered minimum bounding rectangles