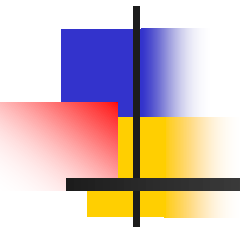
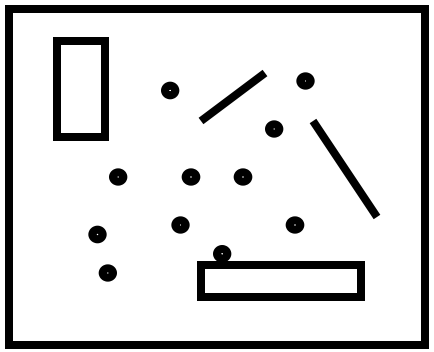


Spatial Queries

Nearest Neighbor Queries

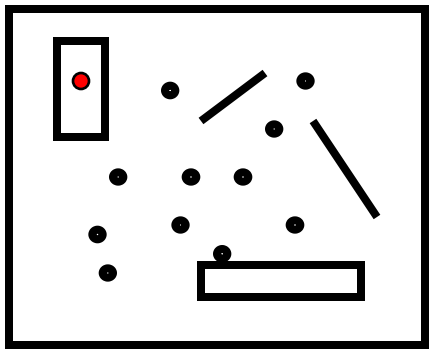
Spatial Queries

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer efficiently
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



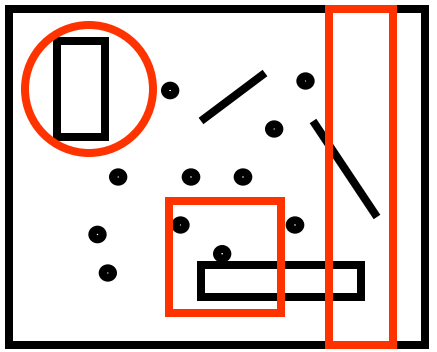
Spatial Queries

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



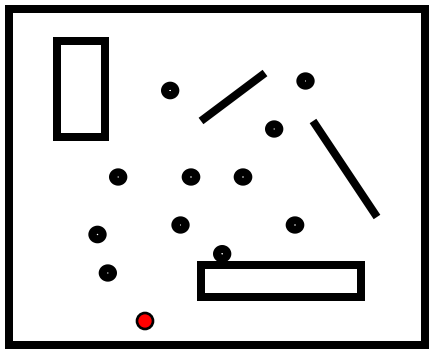
Spatial Queries

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



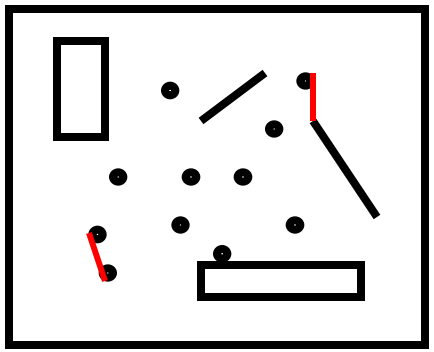
Spatial Queries

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)

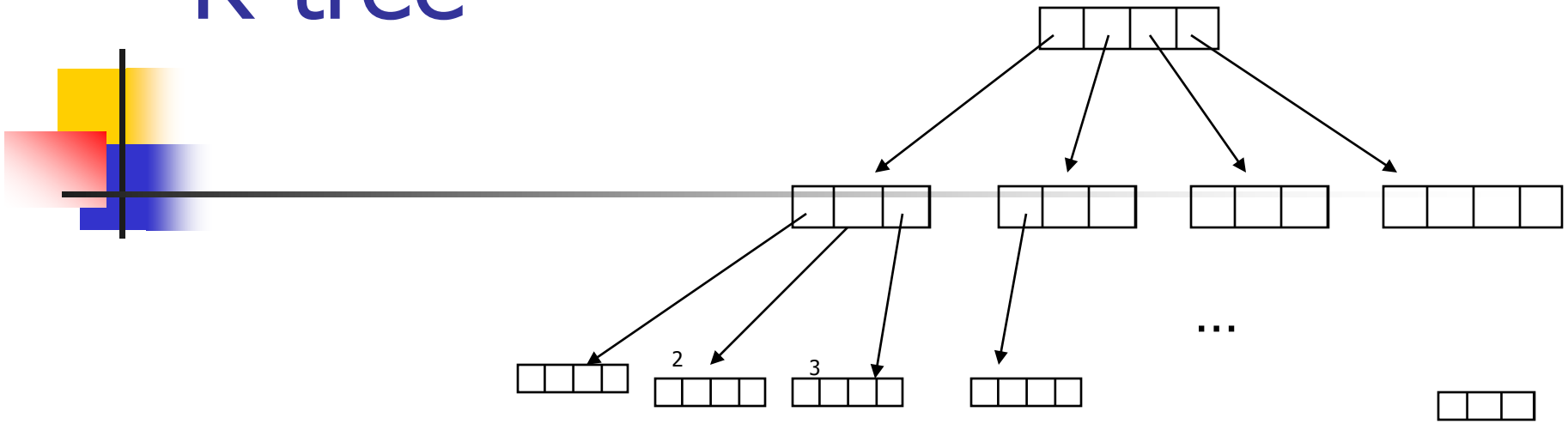


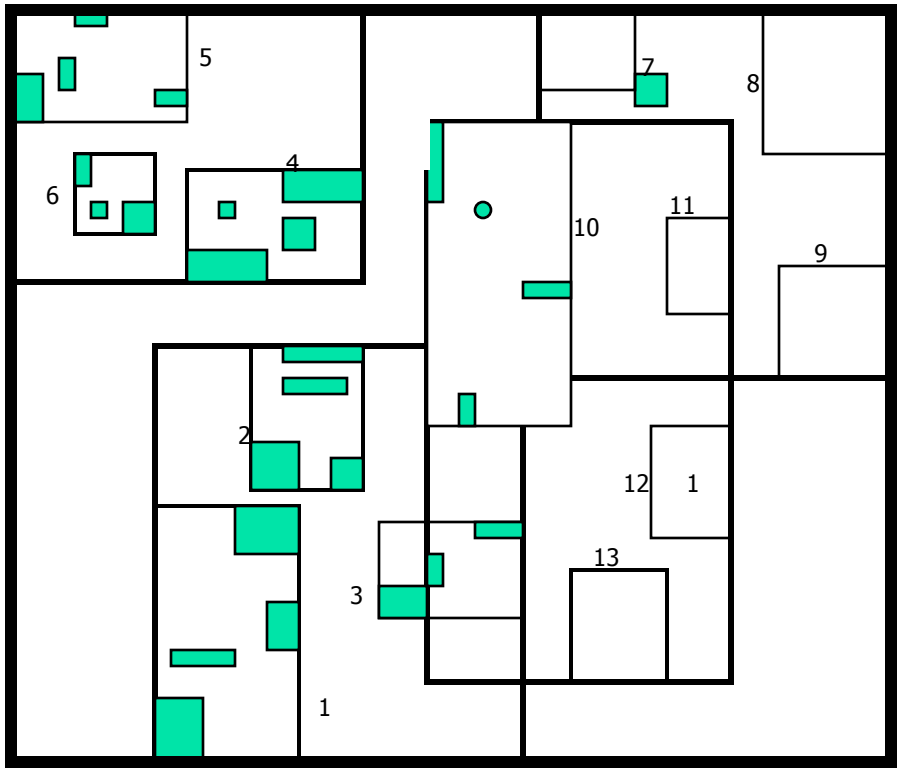
Spatial Queries

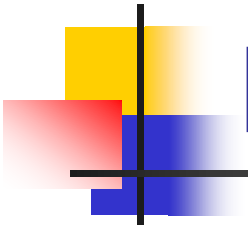
- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins ('all pairs' queries)



R-tree





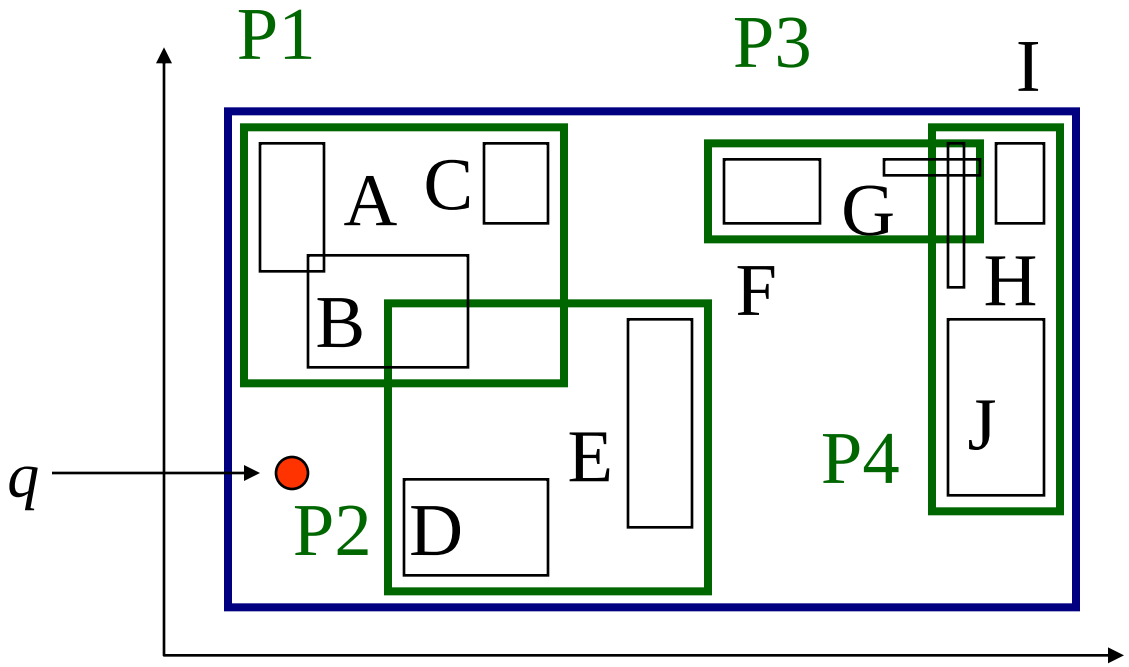


R-trees - Range search

pseudocode:

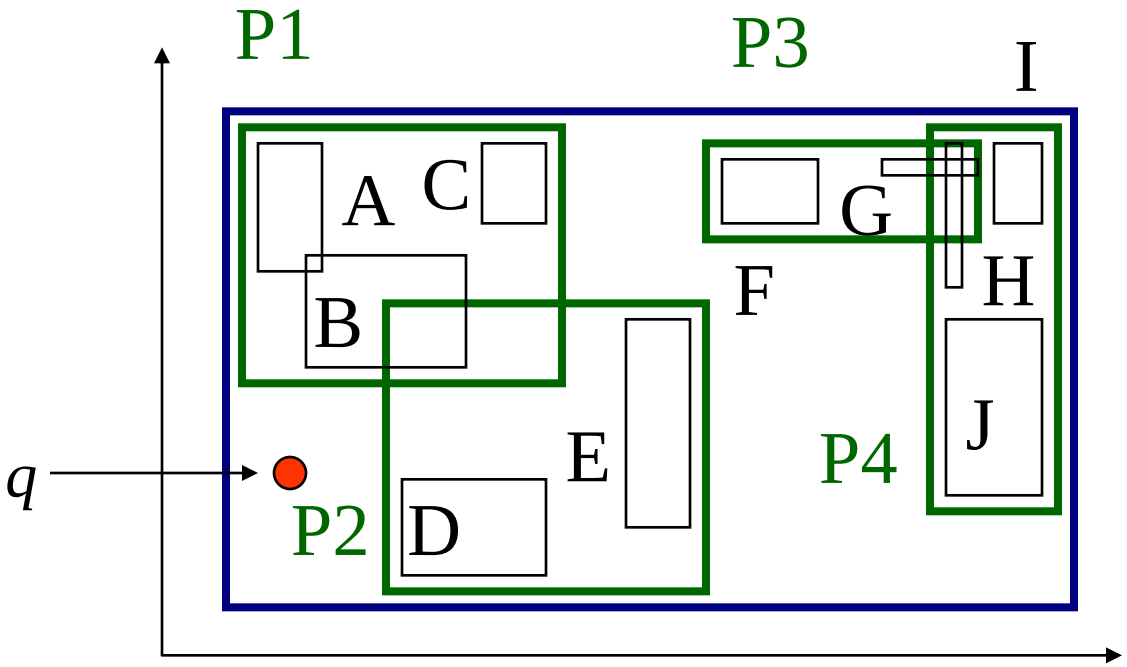
```
check the root
   for each branch,
      if its MBR intersects the query rectangle
            apply range-search (or print out, if this
                  is a leaf)
```

R-trees - NN search



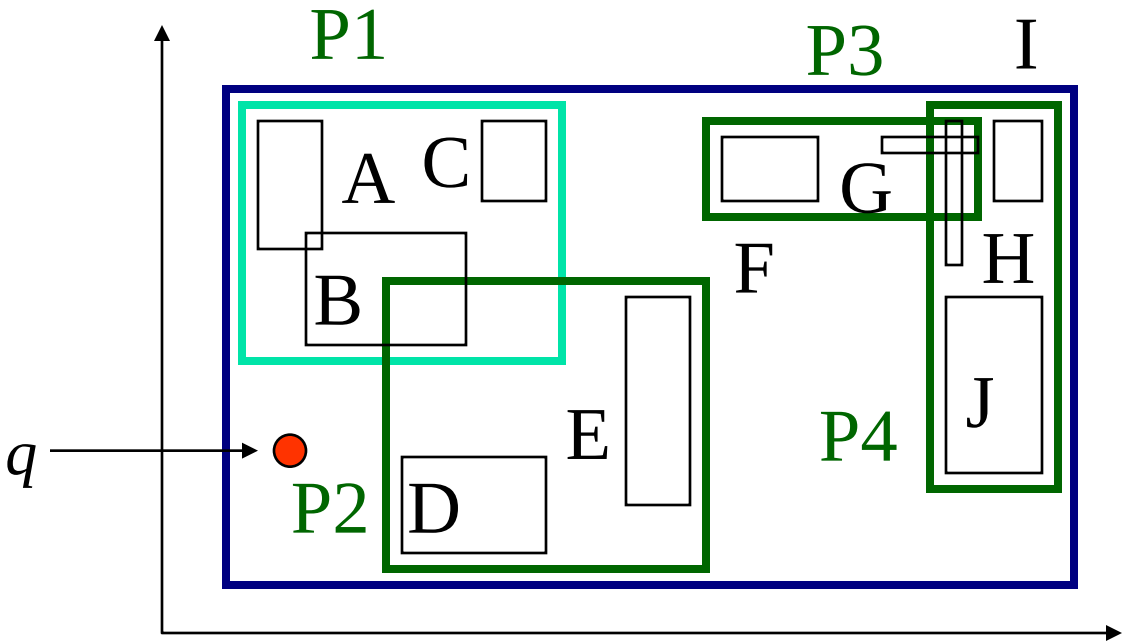
R-trees - NN search

- Q: How? (find near neighbor; refine...)



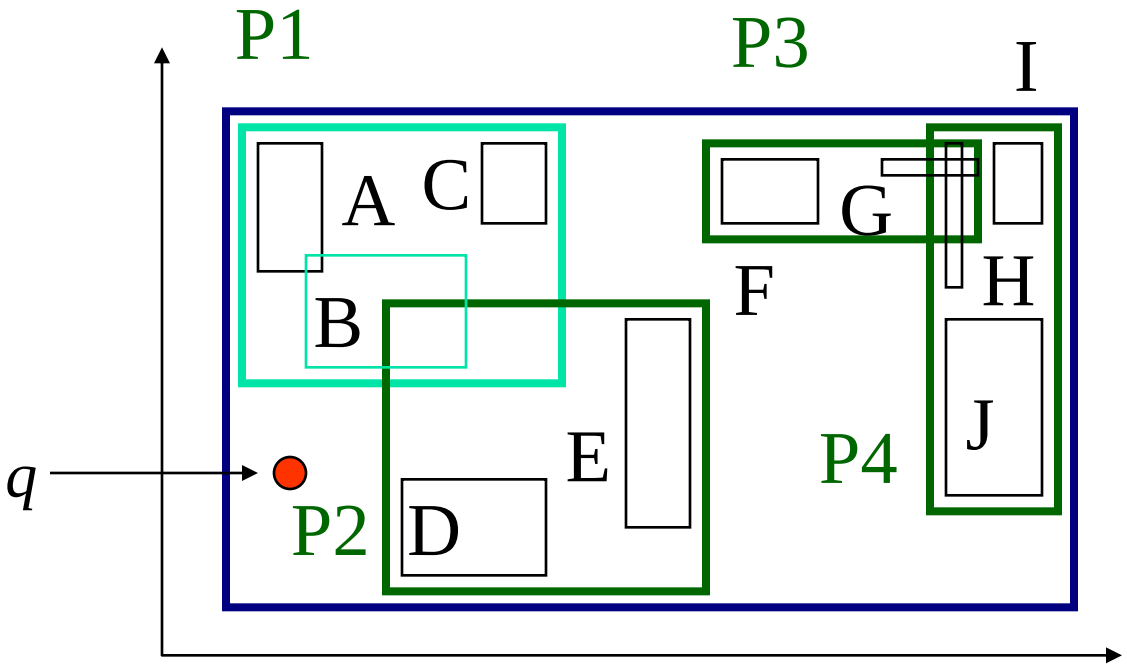
R-trees - NN search

- A1: depth-first search; then range query



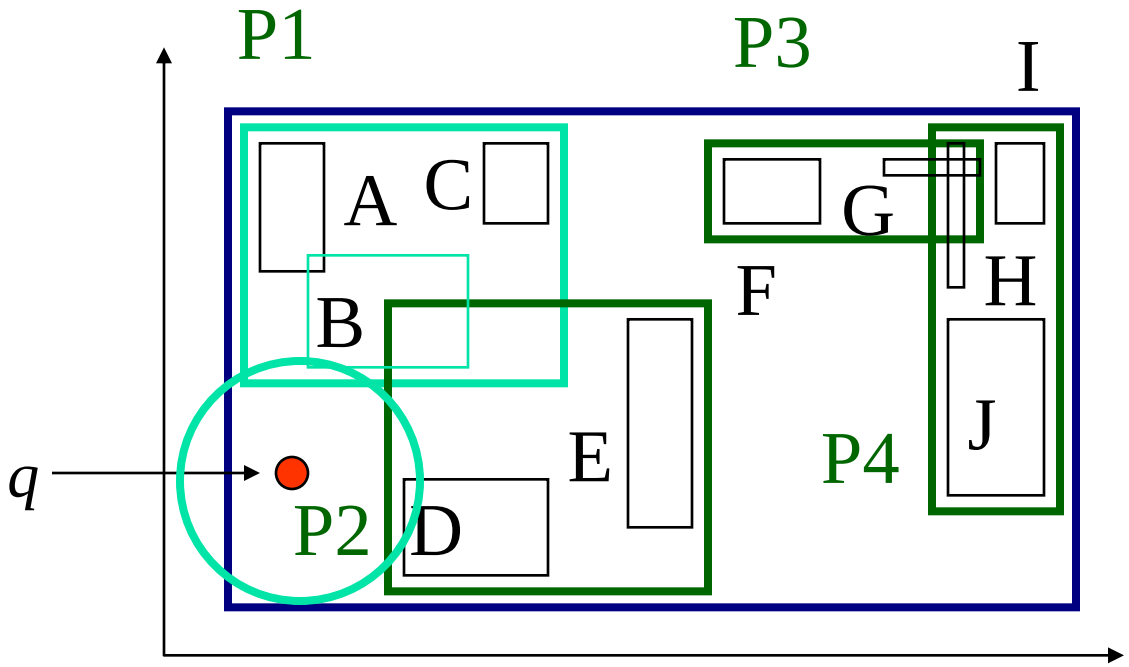
R-trees - NN search

- A1: depth-first search; then range query

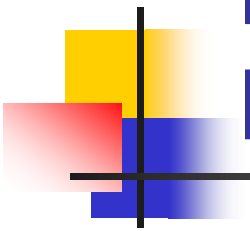


R-trees - NN search

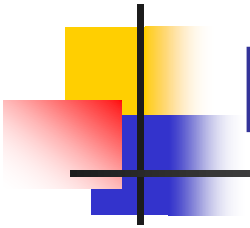
- A1: depth-first search; then range query



R-trees - NN search: Branch and Bound



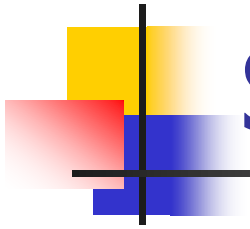
- A2: [Roussopoulos+, sigmod95]:
  - At each node, priority queue, with promising MBRs, and their best and worst-case distance
- main idea: Every face of any MBR contains at least one point of an actual spatial object!



MBR face property

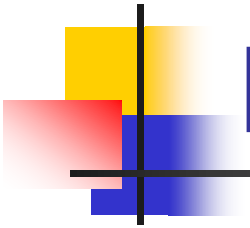
- MBR is a d-dimensional rectangle, which is the minimal rectangle that fully encloses (bounds) an object (or a set of objects)

- MBR f.p.: Every face of the MBR contains at least one point of some object in the database



Search improvement

- Visit an MBR (node) only when necessary
- How to do pruning? Using MINDIST and MINMAXDIST

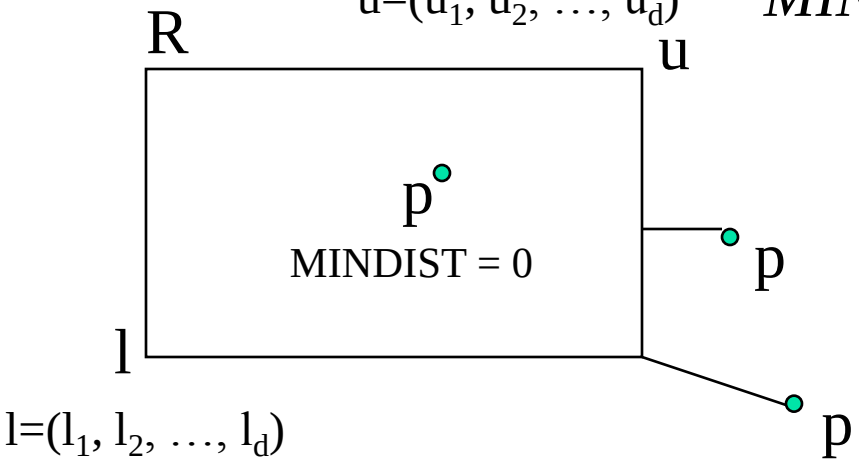


MINDIST

- MINDIST(P, R) is the minimum distance between a point P and a rectangle R
- If the point is inside R, then MINDIST=0
- If P is outside of R, MINDIST is the distance of P to the closest point of R (one point of the perimeter)

MINDIST computation

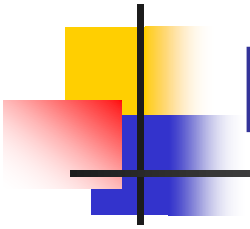
- MINDIST(p,R) is the minimum distance between p and R with corner points l and u
  - the closest point in R is at least this distance away



$$MINDIST(P,R) = \sqrt{\sum_{i=1}^{d} (p_i - r_i)^2}$$

$r_i = l_i$ if $p_i < l_i$
$= u_i$ if $p_i > u_i$
$= p_i$ otherwise

$$\forall o \in R, MINDIST(P,R) \le \|(P,o)\|$$



MINMAXDIST

- MINMAXDIST(P,R): for each dimension, find the closest face, compute the distance to the furthest point on this face and take the minimum of all these (d) distances
- MINMAXDIST(P,R) is the smallest possible upper bound of distances from P to R
- MINMAXDIST guarantees that there is at least one object in R with a distance to P smaller or equal to it.

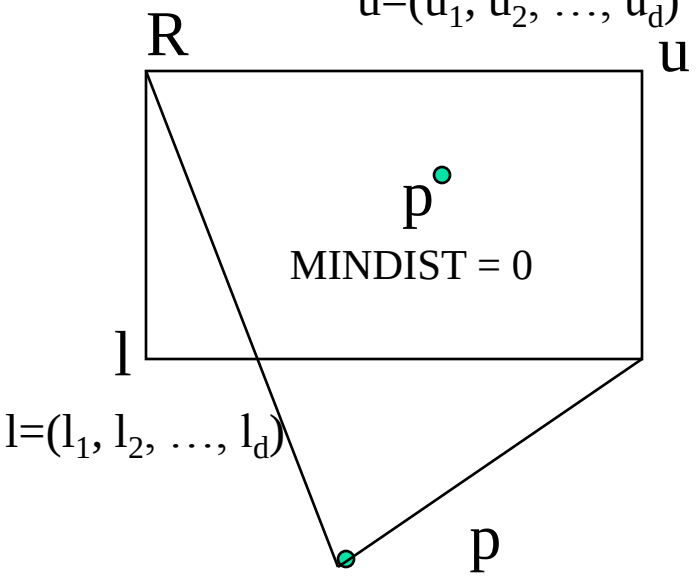
$$\exists o \in R, \left\|(P,o)\right\| \leq MINMAXDIST(P,R)$$

MINMAXDIST computation

- MINMAXDIST(p,R) guarantees there is an object within the MBR at a distance less than or equal to MINMAXDIST
  - the closest point in R is less than this distance away

$$MINMAXDIST(P,R) = \min_{1 \le k \le d} \sqrt{(|\, p_k - rm_k \,|^2 + \sum_{i \ne k, 1 \le i \le n} |\, p_i - rM_i \,|)}$$

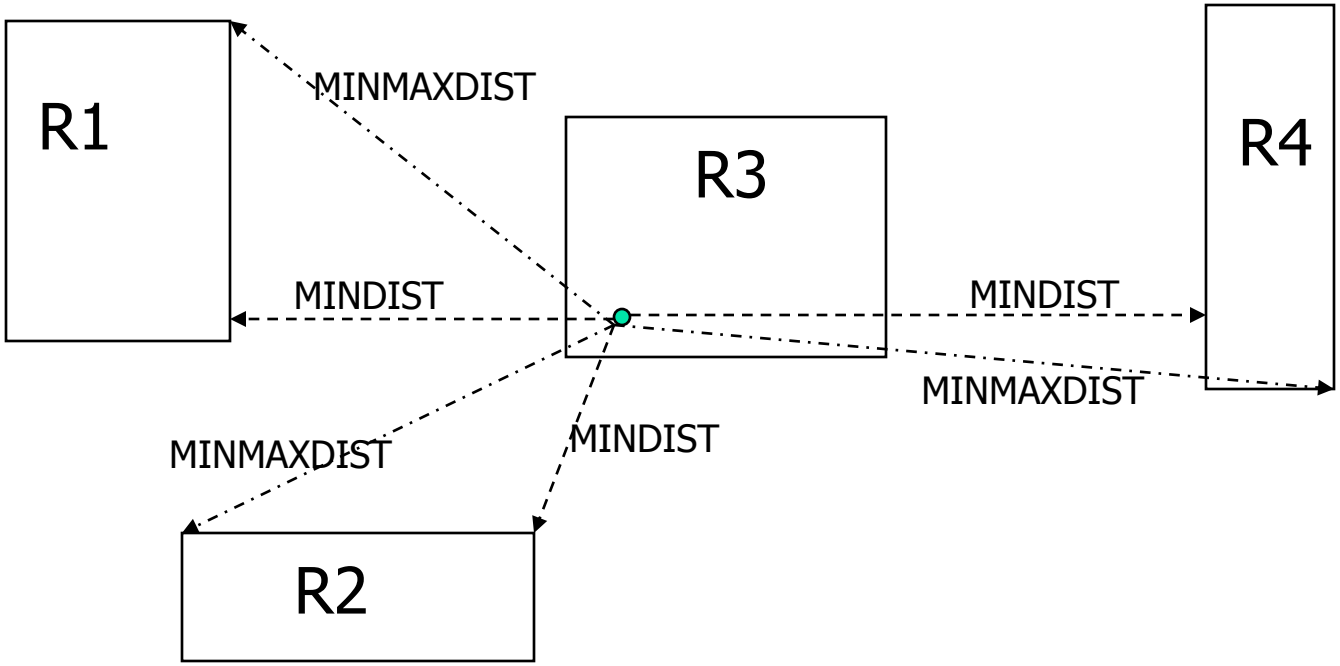
$u=(u_1, u_2, \ldots, u_d)$

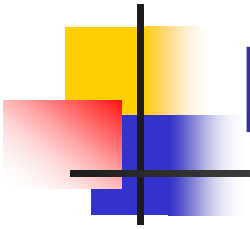


$rm_k = l_k$ if $p_k < \frac{1}{2}(l_k + u_k)$
$= t_k$ otherwise;
$rM_i = u_i$ if $p_i > \frac{1}{2}(l_i + u_i)$
$= l_i$ otherwise

MINDIST and MINMAXDIST

- MINDIST(P, R) <= NN(P) <=MINMAXDIST(P,R)



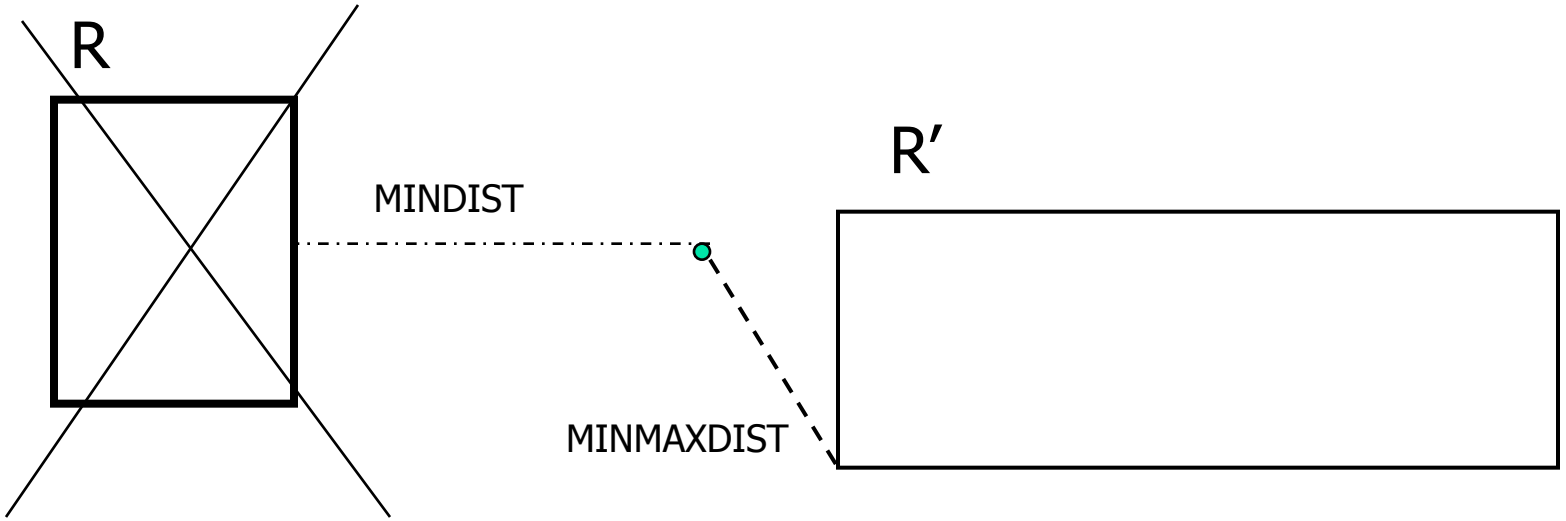


Pruning in NN search

- Downward pruning: An MBR R is discarded if there exists another R' s.t. MINDIST(P,R)>MINMAXDIST(P,R')
- Downward pruning: An object O is discarded if there exists an R s.t. the Actual-Dist(P,O) > MINMAXDIST(P,R)
- Upward pruning: An MBR R is discarded if an object O is found s.t. the MINDIST(P,R) > Actual-Dist(P,O)

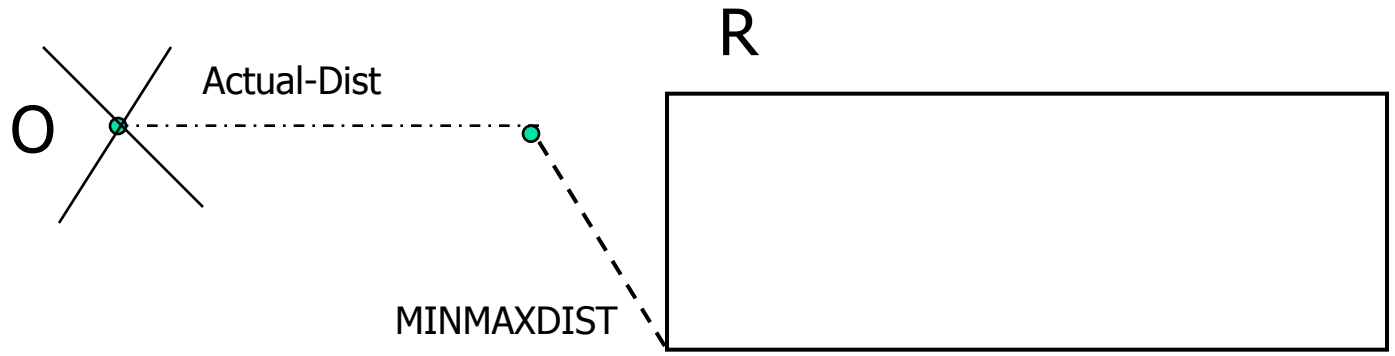
Pruning 1 example

- Downward pruning: An MBR R is discarded if there exists another R' s.t. MINDIST(P,R)>MINMAXDIST(P,R')



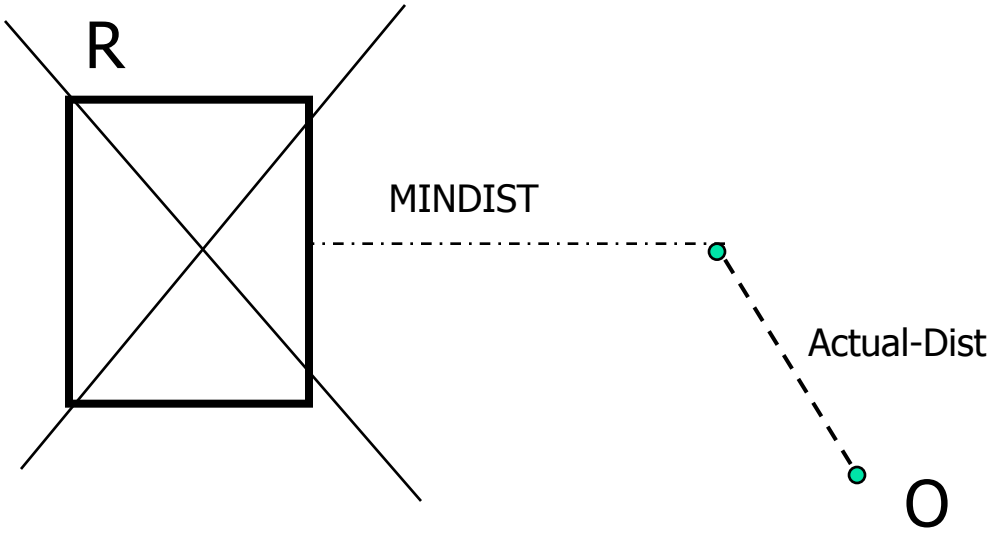
Pruning 2 example

- Downward pruning: An object O is discarded if there exists an R s.t. the Actual-Dist(P,O) > MINMAXDIST(P,R)



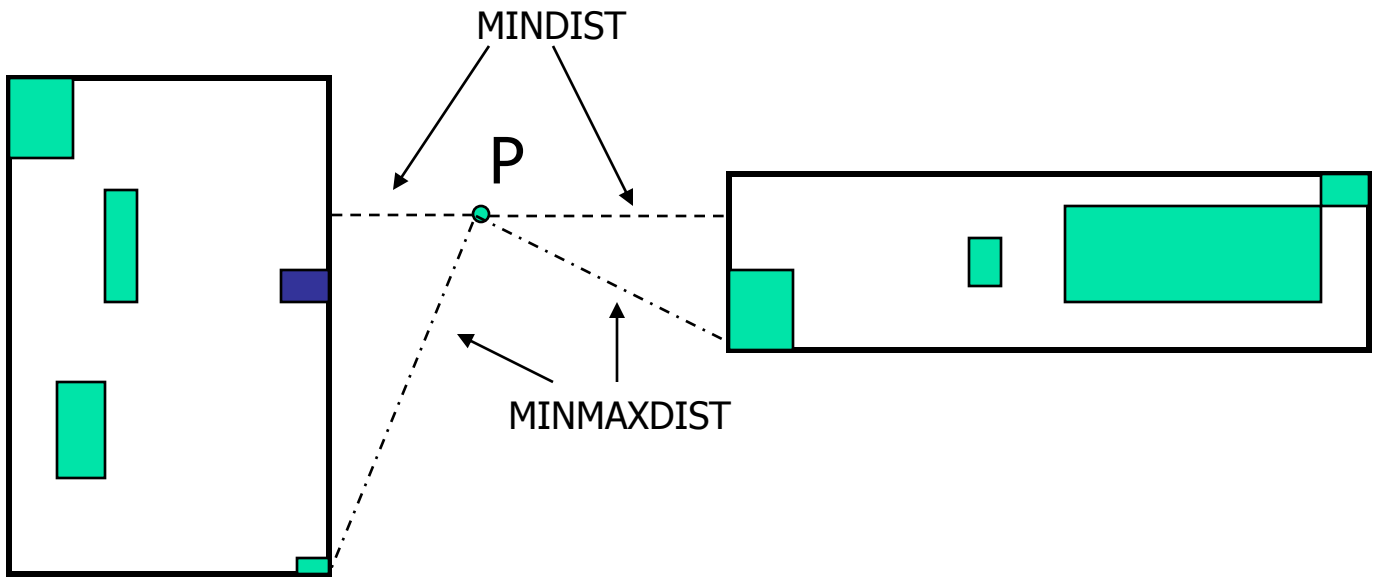
Pruning 3 example

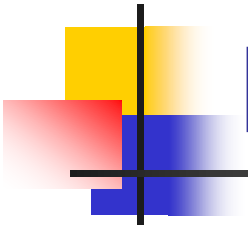
- Upward pruning: An MBR R is discarded if an object O is found s.t. the MINDIST(P,R) > Actual-Dist(P,O)



Ordering Distance

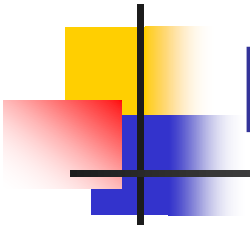
- MINDIST is an optimistic distance where MINMAXDIST is a pessimistic one.





NN-search Algorithm

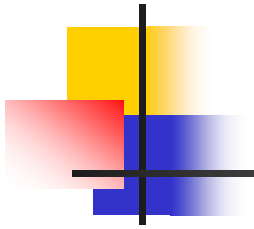
1. Initialize the nearest distance as infinite distance
2. Traverse the tree depth-first starting from the root. At each Index node, sort all MBRs using an ordering metric and put them in an **Active Branch List (ABL).**
3. Apply pruning rules 1 and 2 to ABL
4. Visit the MBRs from the ABL following the order until it is empty
5. If Leaf node, compute actual distances, compare with the best NN so far, update if necessary.
6. At the return from the recursion, use pruning rule 3
7. When the ABL is empty, the NN search returns.



K-NN search

- Keep the sorted buffer of at most k current nearest neighbors
- Pruning is done using the k-th distance

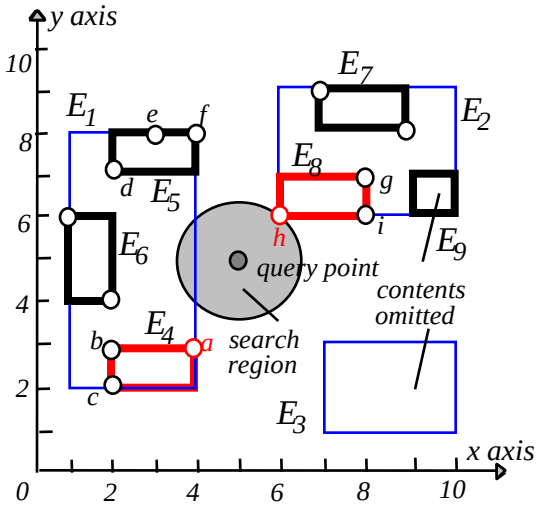
Another NN search: Best-First

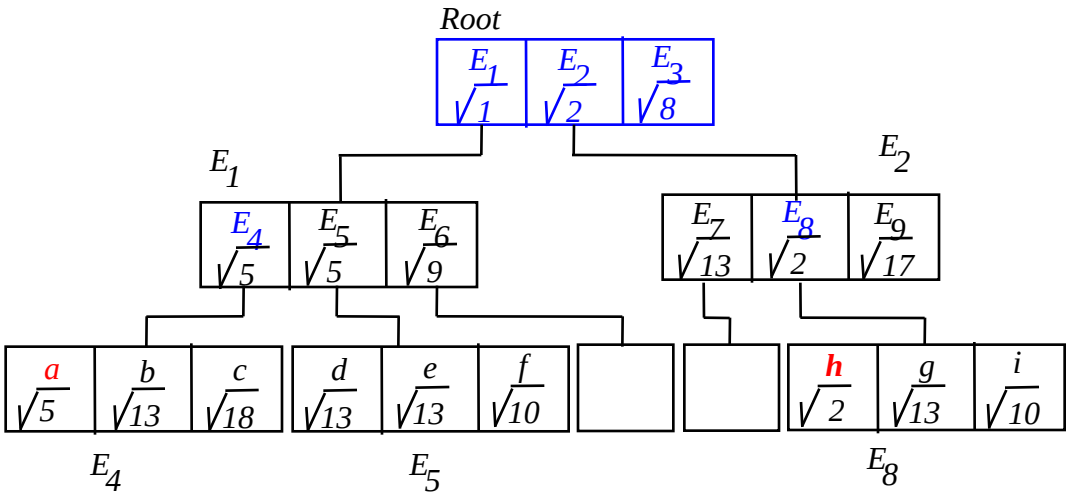


- Global order [HS99]
  - Maintain distance to all entries in a common Priority Queue
  - Use only MINDIST
  - Repeat
    - Inspect the next MBR in the list
    - Add the children to the list and reorder
  - Until all remaining MBRs can be pruned

Nearest Neighbor Search (NN) with R-Trees

Best-first (BF) algorihm:

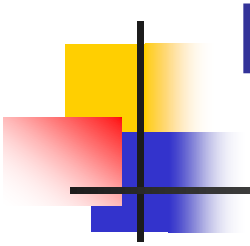




Action	*Heap*	*Result*
Visit Root	$E_1\sqrt{1}$ $E_2\sqrt{2}$ $E_3\sqrt{8}$	{empty}
follow E_1	$E_2\sqrt{2}$ $E_4\sqrt{5}$ $E_5\sqrt{5}$ $E_3\sqrt{8}$ $E_6\sqrt{9}$	{empty}
follow E_2	$E_8\sqrt{2}$ $E_4\sqrt{5}$ $E_5\sqrt{5}$ $E_3\sqrt{8}$ $E_6\sqrt{9}$ $E_7\sqrt{13}$ $E_9\sqrt{17}$	{empty}
follow E_8	$E_4\sqrt{5}$ $E_5\sqrt{5}$ $E_3\sqrt{8}$ $E_6\sqrt{9}$ $E_7\sqrt{13}$ $E_9\sqrt{17}$	$\{(h,\sqrt{2})\}$
	$E_4\sqrt{5}$ $E_5\sqrt{5}$ $E_3\sqrt{8}$ $E_6\sqrt{9}$ $i\sqrt{10}$ $E_7\sqrt{13}$ $g\sqrt{13}$	

Report h and terminate

HS algorithm



Initialize PQ (priority queue)

InesrtQueue(PQ, Root)

While not IsEmpty(PQ)

    R= Dequeue(PQ)

    If R is an object

        Report R and exit (done!)

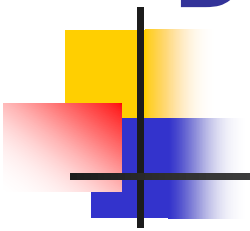
    If R is a leaf page node

        For each O in R, compute the Actual-Dists, InsertQueue(PQ, O)

    If R is an index node

        For each MBR C, compute MINDIST, insert into PQ

Best-First vs Branch and Bound



- Best-First is the "optimal" algorithm in the sense that it visits all the necessary nodes and nothing more!
- But needs to store a large Priority Queue in main memory. If PQ becomes large, we have thrashing…
- BB uses small Lists for each node. Also uses MINMAXDIST to prune some entries