COMP7106B
Big Data Management
Lecture 2 Spatial Data Management

Prof. Reynold Cheng

27th January, 2024

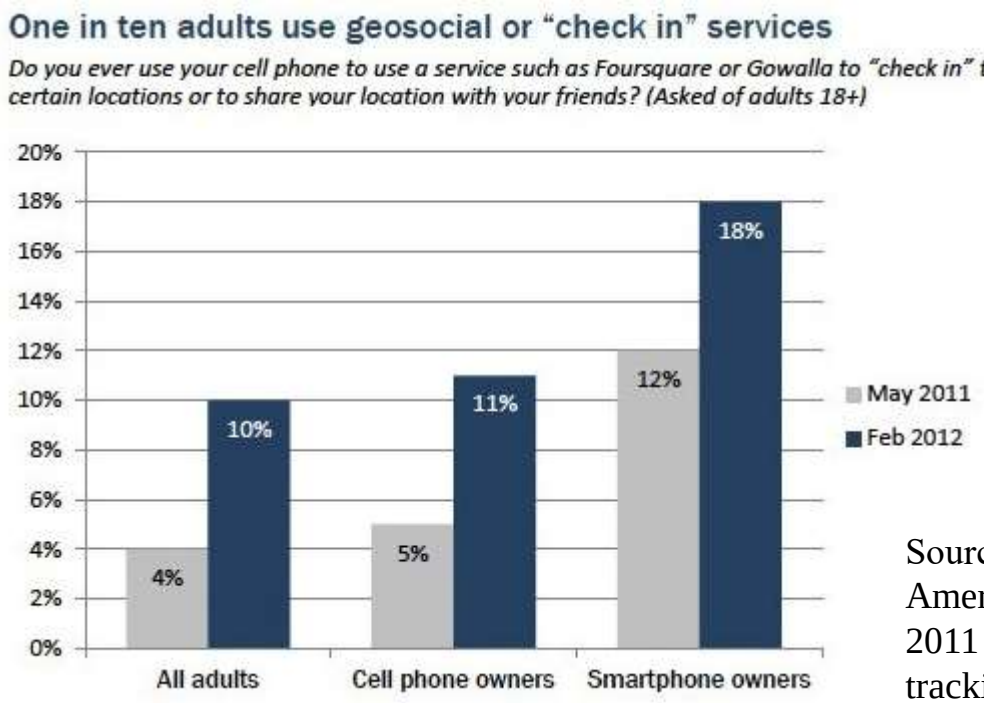
Acknowledgement: This presentation is based on Prof. Nikos Mamoulis' COMP3323 slides.

Spatial Data Management

- Spatial Data
- Spatial Relationships
- Spatial Queries
- Issues in Query Processing
- The R-tree
- Spatial Query Processing
  - Spatial Selections
  - Nearest Neighbor Queries
  - Spatial Joins

Location data

- Check-in service usage is increasing a lot. The number has been more than doubled in 9 months.



Source: Pew Research Center's Internet & American Life Project April 26-May 22, 2011 and January 20-Febrary 19, 2012 tracking surveys

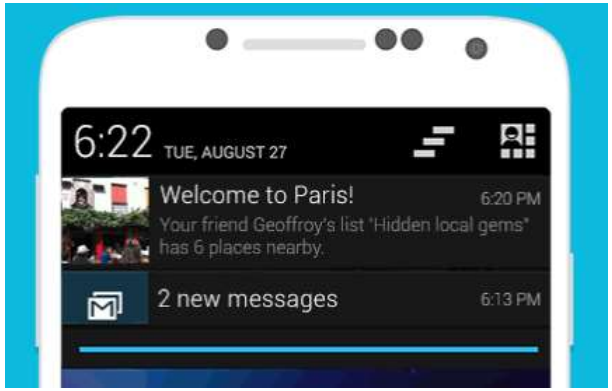
Image source: http://marketingland.com/check-in-service-usage-has-more-than-doubled-in-past-9-months-study-says-11792

Online Maps

- Google Map allows you to Identify your location and plan routes.
- Route information from other users provides suggestions for the best route.

Location-based services

- Map services on mobile phones are able to navigate users to their destination.
- A company can send SMS promotions to subscribers when they are near to their stores.
- Application can suggest a dish to you when you are in a restaurant, notify you that your friends are nearby, or suggest you for places that you may want to visit.



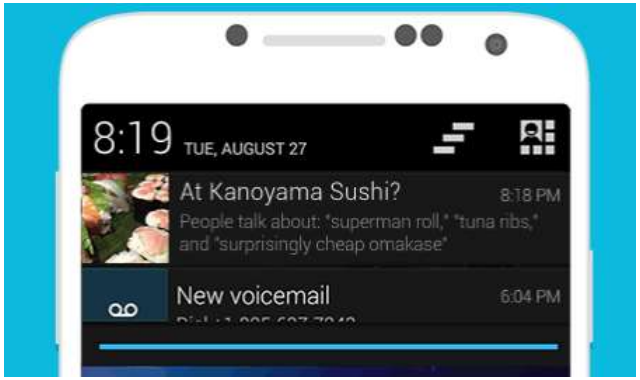


Image source: http://blog.foursquare.com/2013/08/29/a-smarter-foursquare-so-you-dont-miss-a-thing/

Location tracking

- Companies can monitor their delivery truck location with GPS installed on them.
- A bus company can also publish such information for passenger on web.

*Real time bus location in the web site of Enshu Railway Co., Ltd, Hamamatsu, Shizuoka, Japa*n

http://info.entetsu.co.jp/navi/pc/location.aspx?no=15 (Only available in Japanese)

Traffic Data Analysis

- **Loop detectors** are installed to capture real time traffic information
- Multiple data sources from different transportation companies and departments are collected
- By analyzing these data, we predict the traffic condition which can enhance the quality of suggested routes

Loop Detector

Detector Cabinet

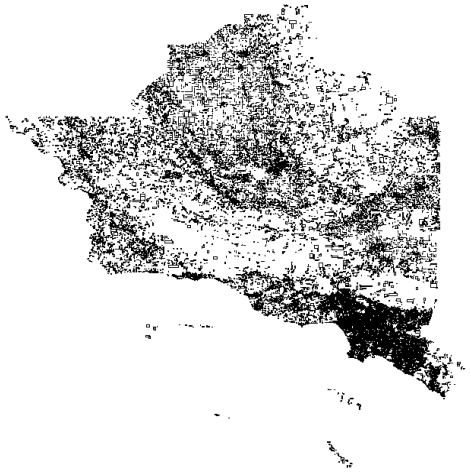
"TransDec: A Big-Data Framework for Decision-Making in Transportation Systems" by Prof. Cyrus Shahabi

Spatial Databases (Free)

- PostgreSQL with PostGIS
- Neo4J-spatial
- HadoopGIS
- Ingres
- GeoMesa

Spatial Data Management

- *Spatial Database Systems* manage large collections of multidimensional objects (typically 2D/3D)
- A *spatial object* contains (at least) one spatial attribute that describes its location and/or geometry
- A *spatial relation* is an organized collection of spatial objects of the same entity (e.g. rivers, cities, road segments)



Road segments from an area in CA

ID	Name	Type	Polyline
1	Boulevard	avenue	(10023,1094), (9034,1567), (9020,1610)
2	Leeds	highway	(4240,5910), (4129,6012), (3813,6129), (3602,6129)
…	…	…	…

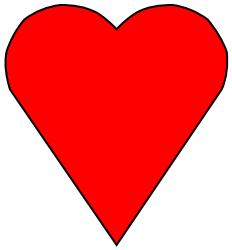
A spatial relation

Spatial Data

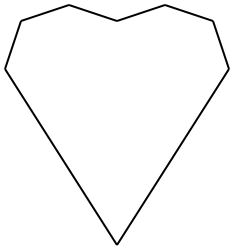
- Spatial objects may be have spatial extent or they can be points.
  - Cities in a large-scale map can be modeled by points
  - Examples of objects with extent, include rivers, forests, districts, buildings in a city, etc.

Spatial Data

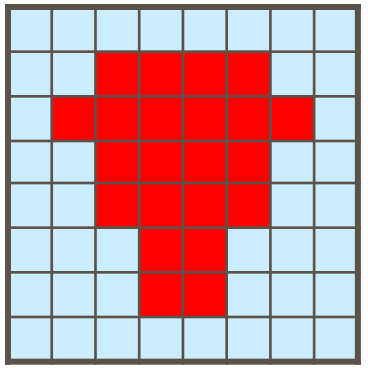
- Two ways to represent objects with spatial extent
  - Vector representation: approximation by geometric objects, e.g., polygons, poly-lines, rectangles, circles, etc.
  - Raster representation: divide the space by a fine grid and approximate the objects by a set of pixels in this grid



object



vector approximation



raster approximation

Spatial Data

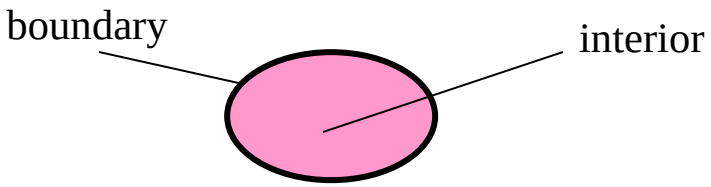
- Spatial data can be found in many applications
  - Geographic Information Systems
  - Segmented images (e.g., objects in X-rays)
  - Components of CAD constructs or VLSI circuits
  - Stars on the sky
  - …
- Spatial database systems are used by
  - Users of mobile devices (find the nearest restaurant)
  - Geographers, astrologers, life scientists, army commanders, etc.

Spatial Relationships

- A spatial relationship associates two objects according to their relative location and extent in space
  - Example: “My house is close to Central Park”
- Sometimes also called “spatial relations”. We will use the term spatial relation to refer to a database relation which stores spatial objects (see slide P. 9)
- Spatial relationships are classified to
  - topological relationships
  - distance relationships
  - directional relationships

Topological Relationships

- Each object is characterized by the space it occupies in the universe.
  - a (finite or infinite) set of pixels
- Each object has a boundary and an interior
  - boundary: the set of pixels the object occupies, that are adjacent to at least one pixel not occupied by the object
  - interior: the set of pixels occupied by the object, which are not part of its boundary



- Note: in some representation models, some objects may not have interior
  - e.g., points, line segments, etc.

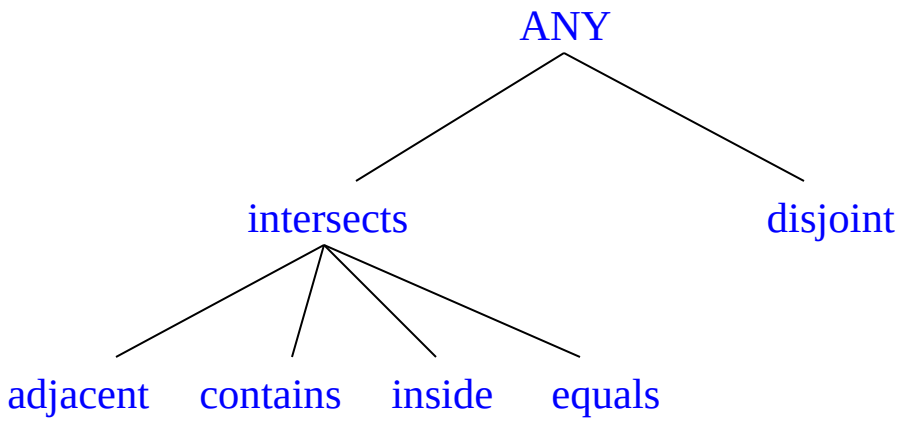
Topological Relationships

- A topological relationship between two objects is defined by a set of (set-based) relationships between their boundaries and interiors
  - E.g., o_1 is inside o_2 if interior(o_1) $\subset$ interior(o_2)
- intersects (or overlap) means any of equals, inside, contains, adjacent
- intersects $\Leftrightarrow \neg$disjoint

Topological relationship	equivalent boundary/interior relationships
$disjoint(o_1, o_2)$	
$intersects(o_1, o_2)$ (or $overlaps(o_1, o_2)$)	
$equals(o_1, o_2)$	
$inside(o_1, o_2)$	
$contains(o_1, o_2)$	
$adjacent(o_1, o_2)$	

Topological Relationships

- In fact, there is a hierarchy of topological relationships:



Distance Relationships

- Distance relationships associate two objects based on their geometric distance.
- The geometric distance is often called Euclidean distance.
- Distance is usually abstracted (i.e., discretized) into the human mind.
- Example (distances in a city)
  - 0-100m: near
  - 100m-1km: reachable
  - 1km-10km: far
  - >10km: very far
- Distance relationships are expressed either explicitly or by some abstract distance class

Directional Relationships

- Directional relationships associate two objects based on their relative orientation according to a global reference system
- Example: My house is north of the river
  - relationship can also be a number:
    - e.g. house 96 degrees relative to river
- Examples of directional relationships:
  - north, south, east, west, northeast, etc.
  - left, right, above, below, front, behind, etc.
- Topological, distance, and directional relationships can be combined :
  - My house is disjoint with the park, 100 meters north of it

Spatial Queries

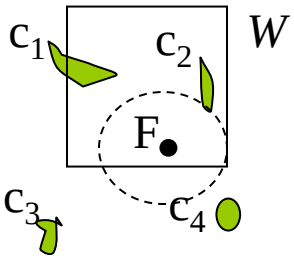
- Applied on one (or more) spatial relations
- Retrieve objects (or combinations of objects) satisfying some spatial relationships
  - between them or
  - with a reference query object
- Examples:
  - Nearest neighbor query: What is the nearest city to my current location?
  - Spatial join: Find all pairs of hotels and restaurants within 100m distance from each other

Spatial Queries

- Range query (spatial selection, window query)

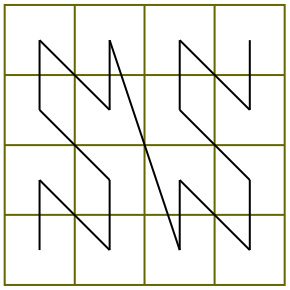
  e.g. find all cities that *intersect* window *W*

  Answer set: {c_1, c_2}

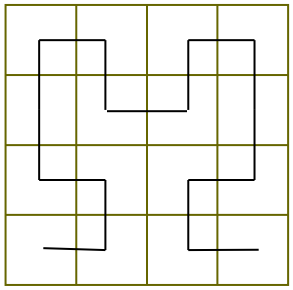


What Is Special About Spatial

- Dimensionality: There is no total ordering of objects in the multidimensional space that preserves spatial proximity



z-curve



Hilbert-curve

space-filling curves

- Complex spatial extent: The spatial extent adds to the complexity of object clustering into disk pages imposed by dimensionality

- No standard (closed) definitions of spatial operations and algebra

What Is Special About Spatial

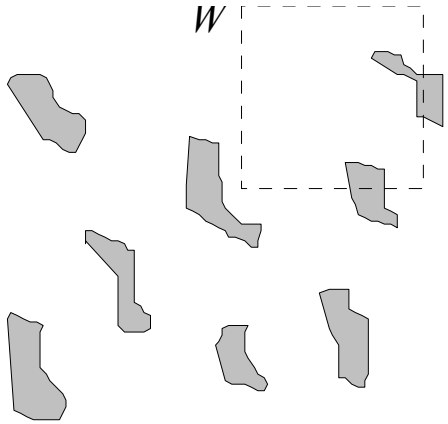
- Implication:
  - Relational indexes (e.g. B+-trees) and query processing methods (e.g. sort-merge join, hash-join) are not readily applicable to spatial data
  - New, *spatial access methods* (SAMs) for spatial data have to be defined
- SAMs index spatial objects and facilitate efficient processing of simple spatial query types (i.e., range queries)

Two-step Spatial Query Processing

- Evaluating spatial relationships on geometric data is slow; a *spatial object* is approximated by its *minimum bounding rectangle* (MBR)
- The *spatial query* is then processed in two steps:
  1. *Filter step*: The MBR is tested against the query predicate
  2. *Refinement step*: The exact geometry of objects that pass the filter step is tested for qualification
- Example for spatial intersection joins:

Two-step Spatial Query Processing

- Example 2:
- Problem: Find all objects that intersect the query window *W*



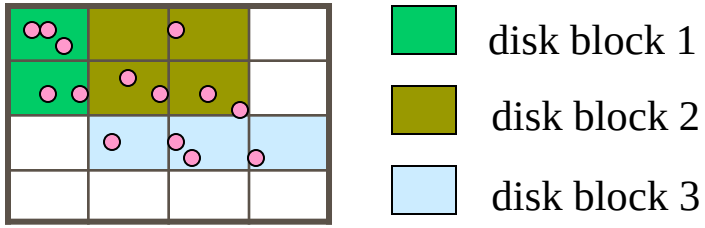
(a) objects and a query

Spatial Access Methods

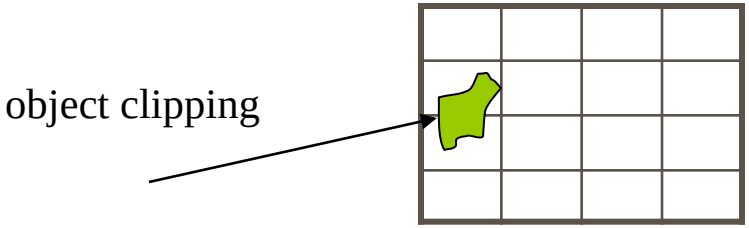
- The problem of indexing spatial data is challenging
  - No dynamic access method with good theoretical worst-case guarantees for range queries
- Therefore Spatial Access Methods aim at the minimization of the expected cost
- Early SAMs focused on indexing of multidimensional points
  - Examples: the grid file, the k-d-B-tree
- Point access methods are not effective for extended objects
  - Objects may need to be *clipped* into several parts which leads to data redundancy and affects performance negatively

Spatial Access Methods

- Point access methods (dynamically) divide the space into disjoint partitions and group the points according to the regions they belong
  - E.g., the grid-file

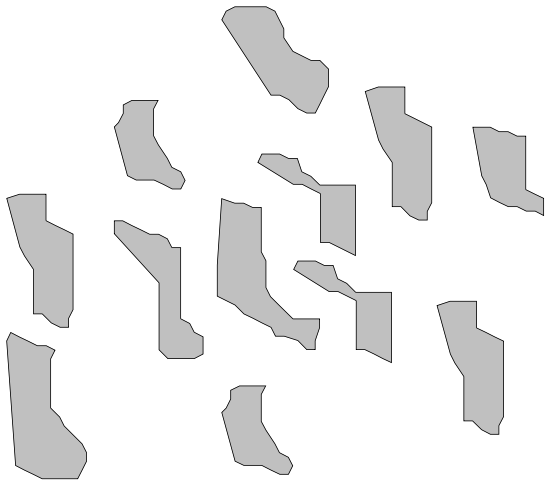


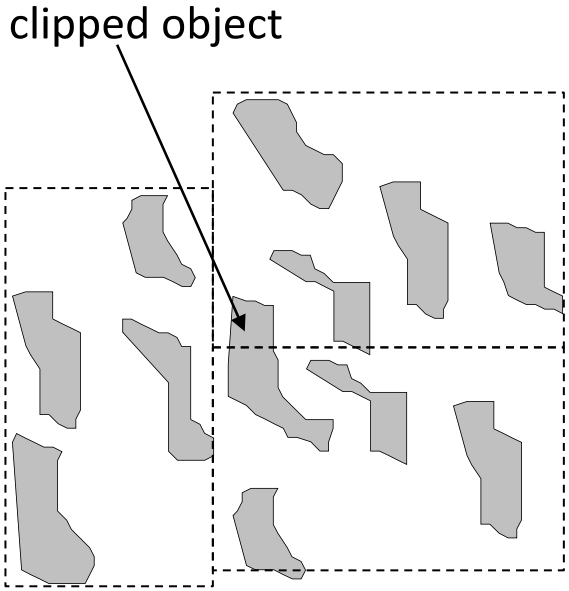
- Point access methods are not effective for extended objects
  - Objects may need to be *clipped* into several parts which leads to data redundancy



Spatial Access Methods

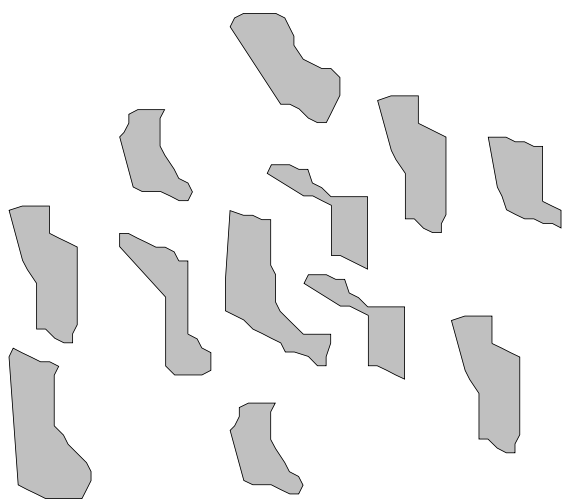
- Object clipping can be avoided if we allow the regions of object groups to overlap
- Problem:
  - Group the objects below into 3 groups of 4 objects each such that the MBRs of the groups do not overlap
  - not possible!

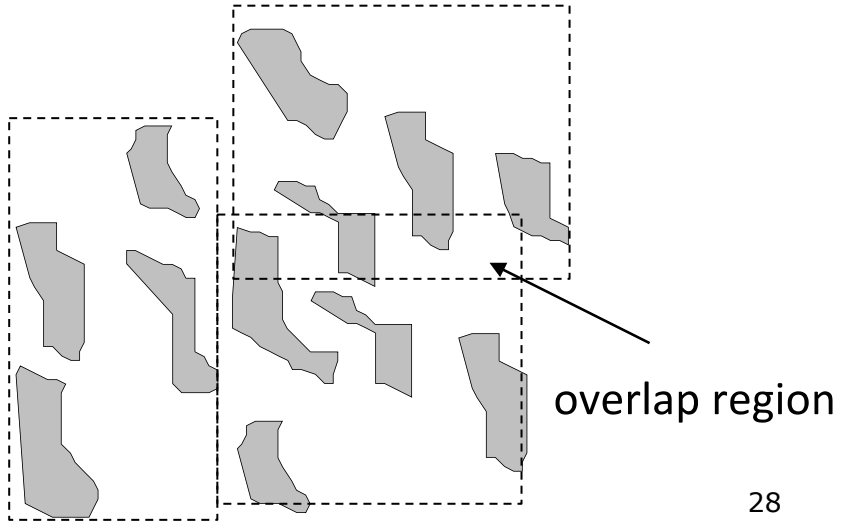




Spatial Access Methods

- Better:
  - Group the objects below into 3 groups of 4 objects each such that the MBRs of the groups have the minimum overlap
  - Hard optimization problem
  - We can seek for a fast, suboptimal solution





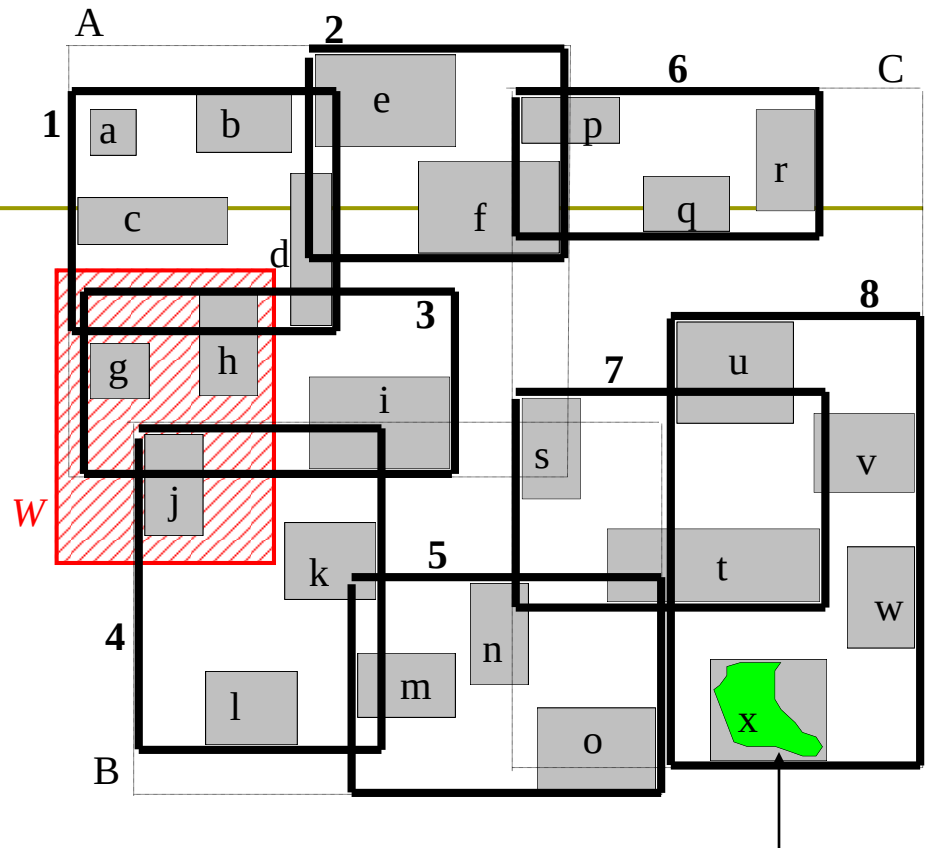
The R-tree

- Groups object MBRs to disk blocks hierarchically
- Each group of objects (a disk block) is a leaf of the tree
- The MBRs of the leaf nodes are grouped to form nodes at the next level
- Grouping is recursively applied at each level until a single group (the root) is formed

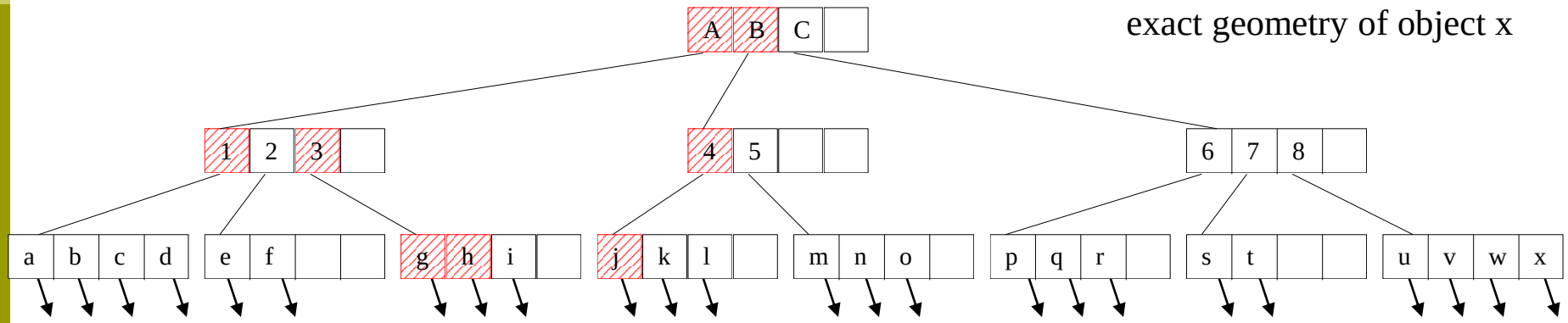
The R-tree

- Leaf node entries: <MBR, object-id>
- Non-leaf node entries: <MBR, ptr>
- The MBR of a non-leaf node entry is the MBR of all entries in the node pointed by it
- Parameters (except root):
  - M (max no of entries per node)
  - m (min no of entries per node)
  - m <= M/2
  - usually m=0.4M
- Root has at least two children
- All leaves in same level (balanced tree)
- 1 node → 1 disk block

The R-tree

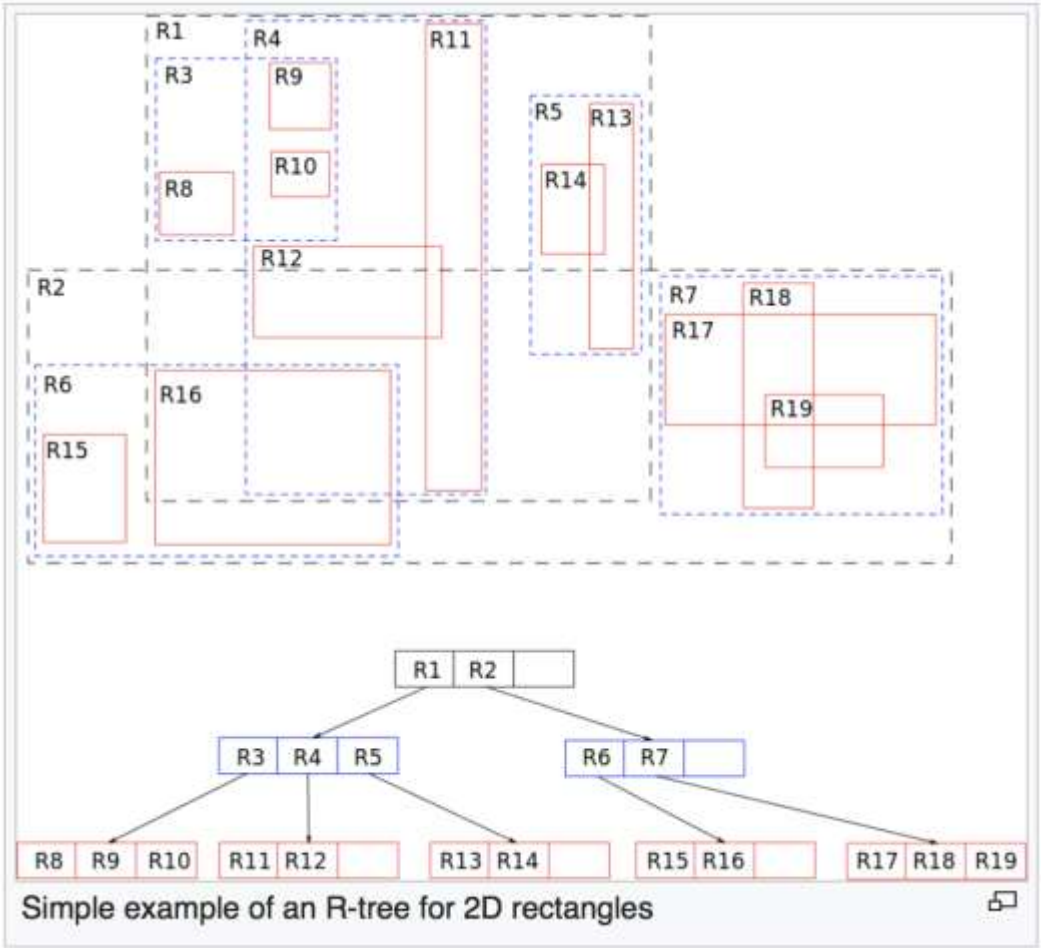


exact geometry of object x



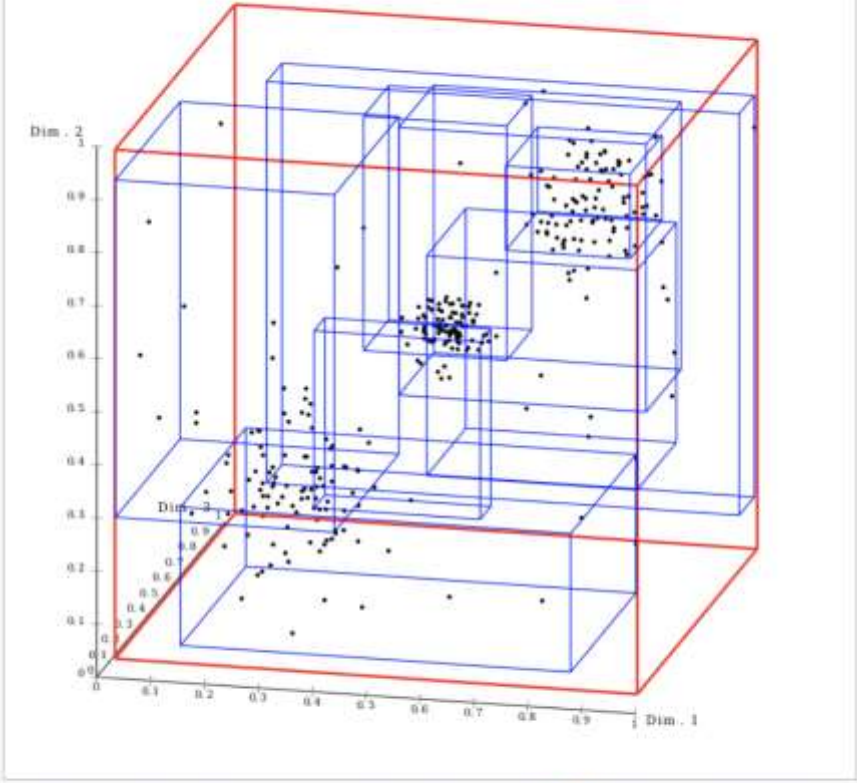
pointers to object locations in the spatial relation (for accessing the exact geometry and other attributes)

The R-tree



https://en.wikipedia.org/wiki/R-tree

The R-tree



Visualization of an R*-tree for 3D points using ELKI (the cubes are directory pages)

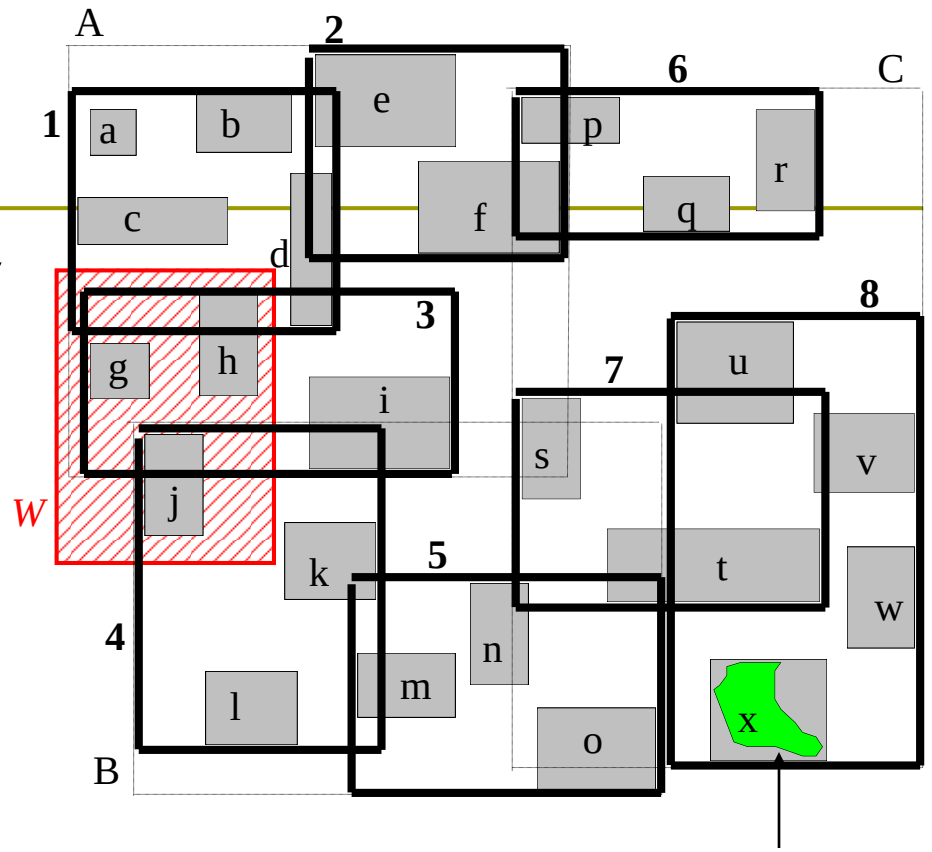
https://en.wikipedia.org/wiki/R-tree

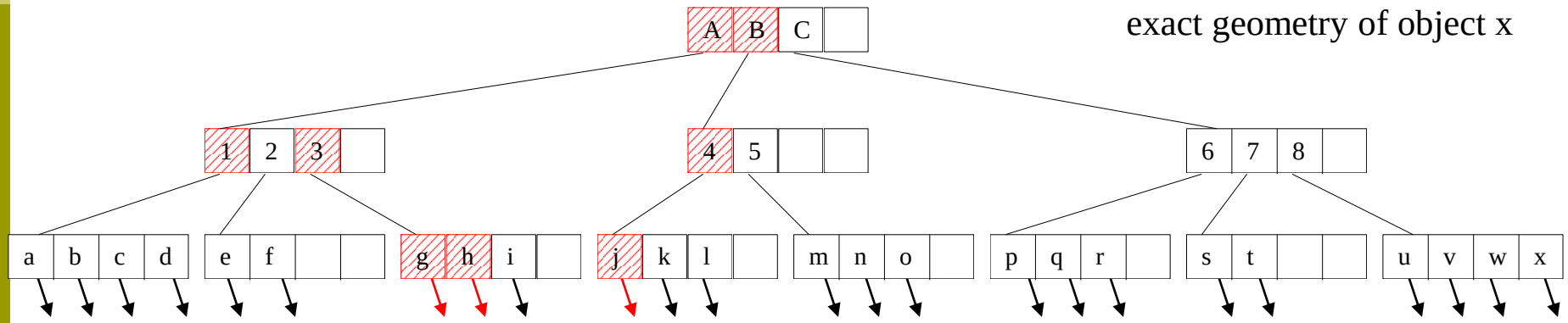
Range searching using an R-tree

- Range_query(query *W*, R-tree node *n*):
  - If n is not a leaf node
    - For each index entry *e* in *n* such that e.MBR intersects *W*
      - visit node *n'* pointed by *e.ptr*
      - Range_query(*W*, *n'*)
  - If n is a leaf
    - For each index entry *e* in *n* such that e.MBR intersects *W*
      - visit object o pointed by *e.object-id*
      - test range query against exact geometry of o; if o intersects W, report o
- May follow multiple paths during search
- Different search predicates are used for different relationships with *W*.
  - What if we want to find all objects inside W?

Searching

- *W* is a window query (range intersection query)





Construction of the R-tree

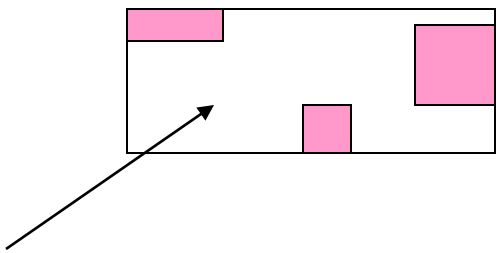
- Dynamically constructed/maintained
- Insertions/deletions interleave with search operations
- Insertion similar to B^+-tree
  - However special optimization algorithms have to be designed for
    - choosing the path where a new MBR is inserted
    - splitting overflown nodes
- Underflows in deletions are handled by
  - deleting the underflown leaf node
  - re-inserting the remaining entries

R*-tree: an optimized version of the R-tree

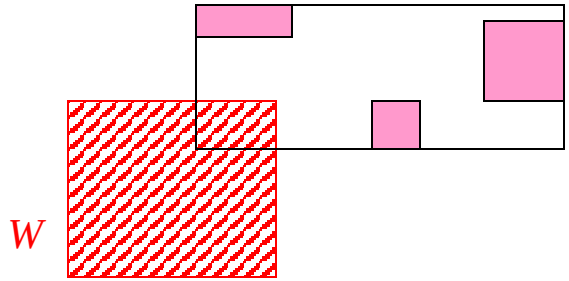
- The R-tree and the R*-tree differ only in the insertion algorithm
- The improved insertion algorithm aims at constructing a tree of high quality
- A good tree has
  - nodes with small MBRs
  - nodes with small overlap
  - nodes that look like squares
  - nodes as full as possible

Optimization Criteria

- Minimize the area covered by an index rectangle
  - → small area means small dead space



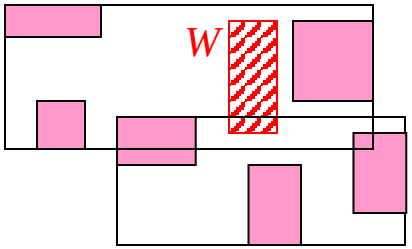
dead space



window query intersects node MBR, but no object!

Optimization Criteria

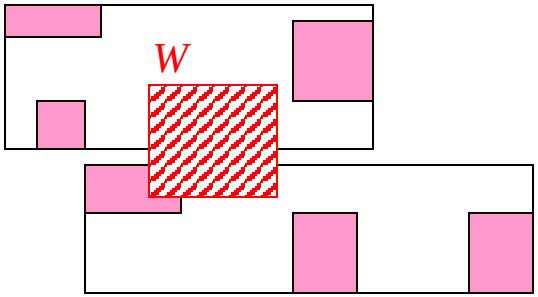
- Minimize overlap between node MBRs
  - → Minimizes the number of traversed paths



Both nodes intersect query!

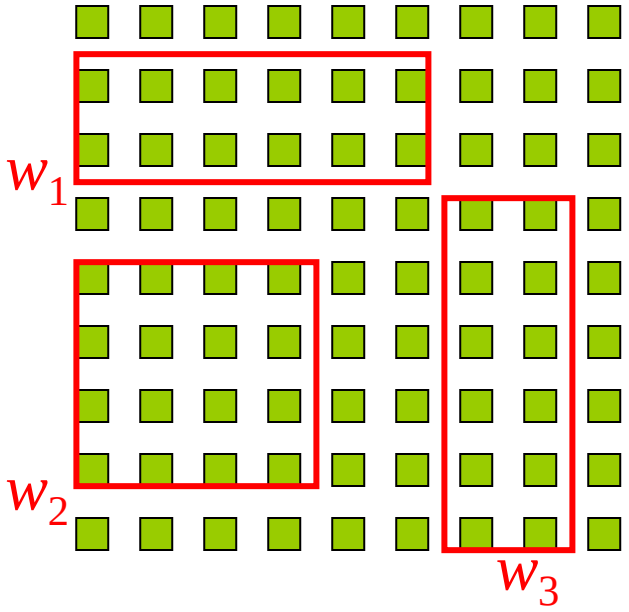
Optimization Criteria

- Minimize the margins of node MBRs
  - → Square-like nodes, smaller number of intersections for a random query, better structure



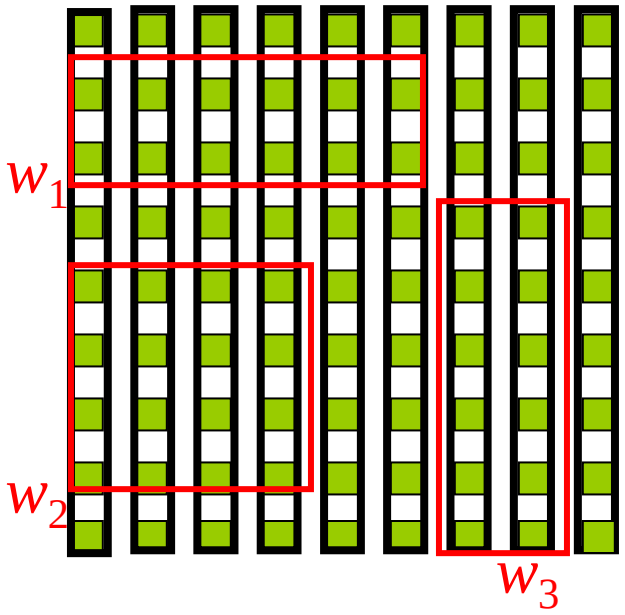
Effect of Margins Minimization

- Problem:
  - group the rectangles below into groups of 9, such that the expected number of group MBRs intersected by a query is minimized



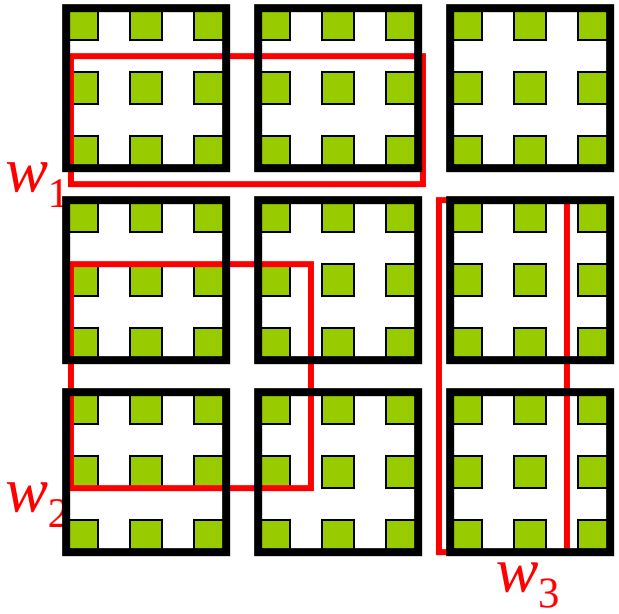
Effect of Margins Minimization

- Grouping 1 (minimal areas of group MBRs)
  - bad grouping for queries w_1, w_2



Effect of Margins Minimization

- Grouping 2 (minimization of margins)
  - minimizes the expected number of groups touching a random query



Optimization Criteria

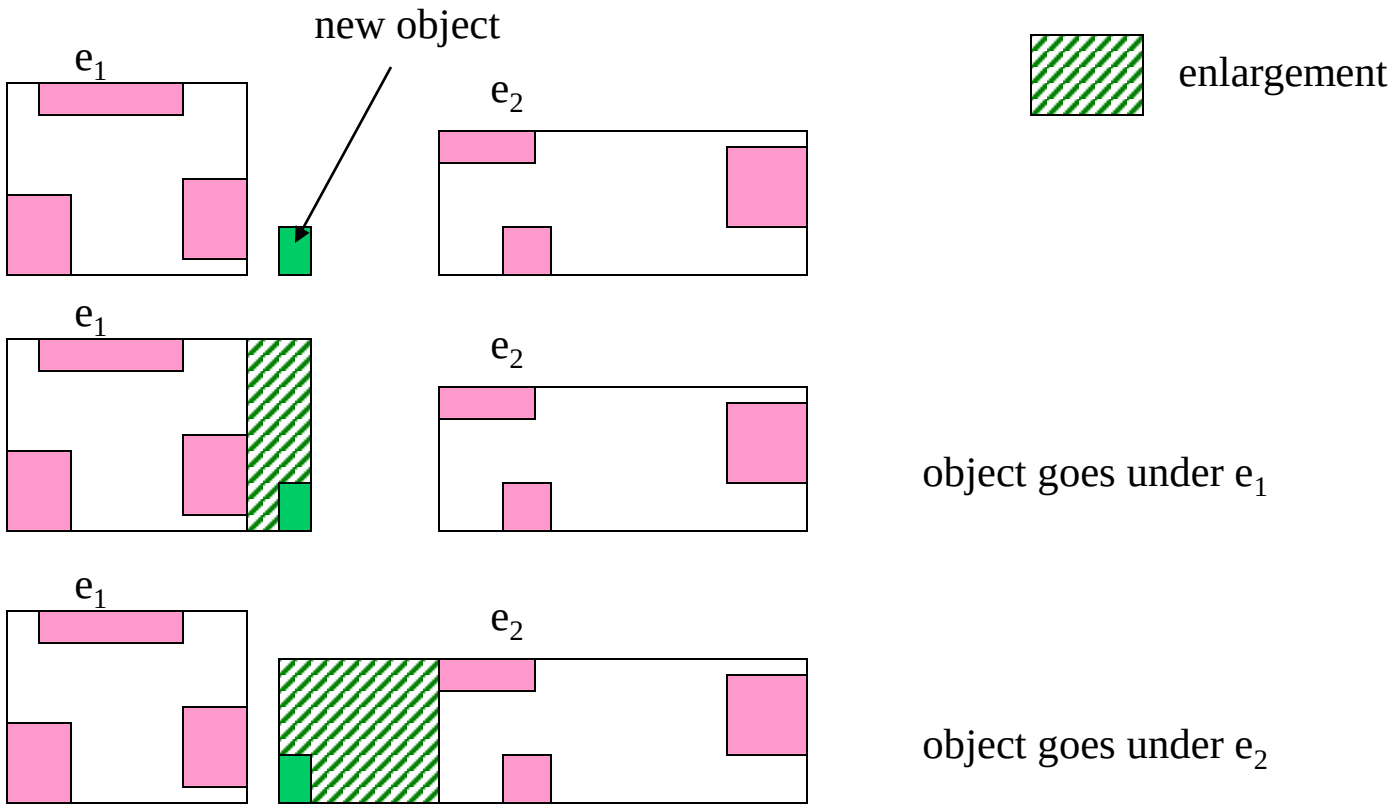
- Optimize the storage utilization
  - → Nodes in tree should be filled as much as possible
  - → Minimizes tree height and potentially decreases dead space
- Sometimes it is impossible to optimize all the above criteria at the same time!

Insertion heuristics

- When inserting a new entry e into the tree we follow one path from the root to a leaf
- The entry is then inserted to the leaf
- Issue: How to choose the path?
  - Let n be the current node, under which the new entry e will be inserted.
    - Follow subtree pointed by entry e of n:
      - whose MBR is enlarged the least after insertion (n is non-leaf)
      - whose MBR enlargement will cause the minimum overlap with other entries of the same node (n is leaf)
      - break any ties by choosing MBR with the minimum area

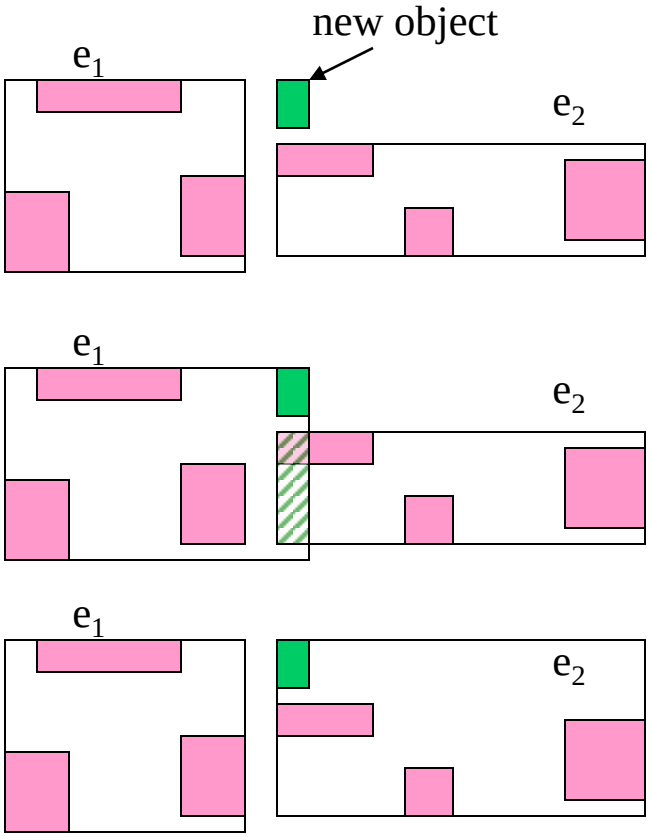
Insertion heuristics

- MBR enlargement after insertion



Insertion heuristics

- MBR overlap after insertion



overlap

object goes under e_1

object goes under e_2

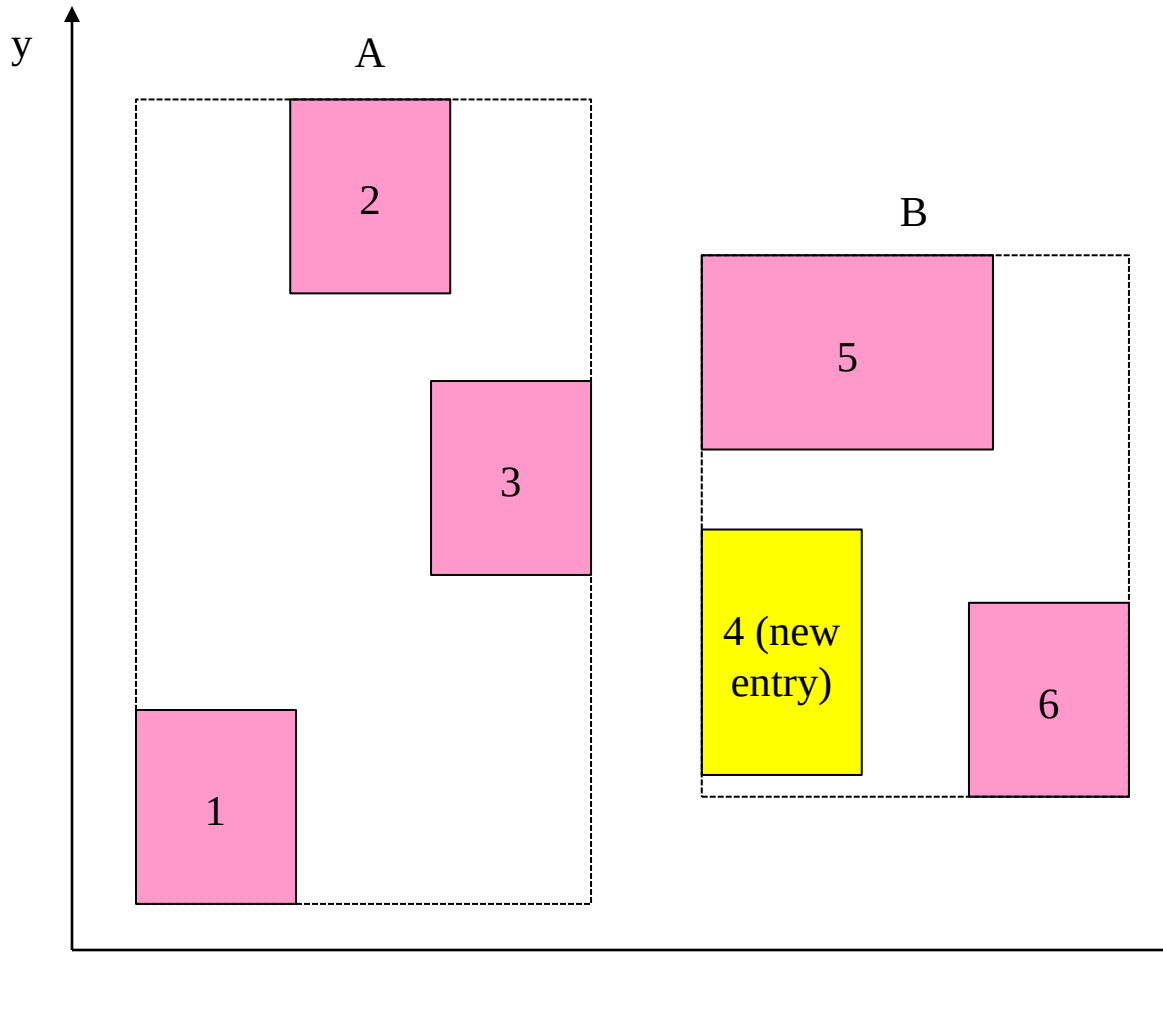
Node splitting

- If a node overflows we need to split it
- Issue: distribute (fast!) a set of rectangles into two nodes such that the areas, overlap, and margins are minimized.
  - Have to give weight on some optimization criteria (conflicting)
  - Have to perform splitting fast
  - Distribution may not be even (m<0.5M)
- Sort the rectangles with respect to one axis (x or y) and then find the best split distribution from the sorted lists

1. Determine the split axis

- For each axis (i.e. x and y axis)
  - Sum=0;
  - sort entries by the lower value, then by upper value
  - for each sorting (e.g. lower value)
    - for k=m to M+1-m
      - place first k entries in group A, and the remaining ones in group B
      - Sum = Sum + margin(A) + margin(B)
- Choose axis with the minimum Sum

Determine the split axis (M=5, m=2)



No. objects = 6

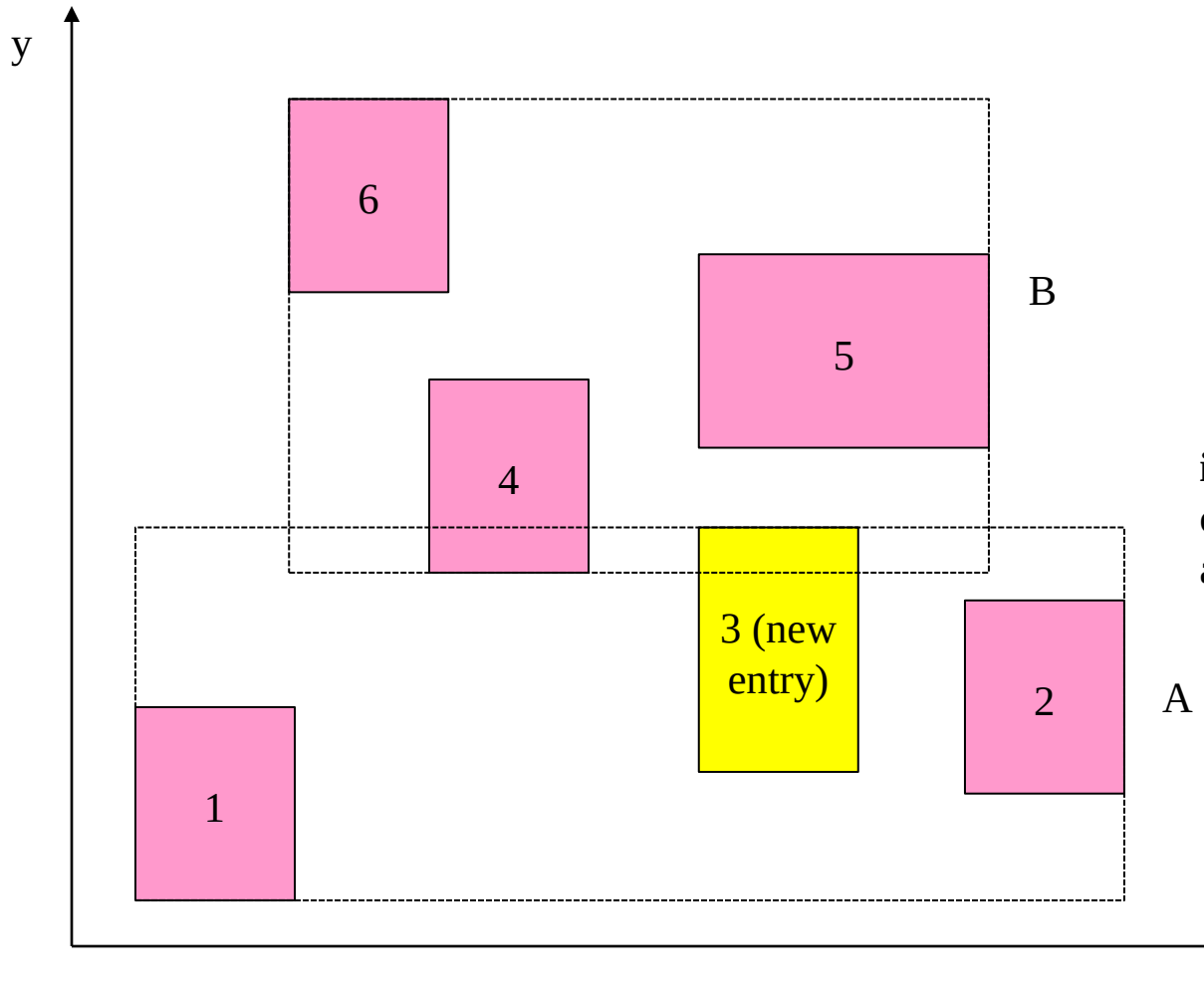
Sorted in x-axis

k=3

Sum_x=margin(A) + margin(B)

(Get the total Sum_x for all k in [2, 4])

Determine the split axis (M=5, m=2)



Sorted in y-axis

k=3

Sum_y=margin(A) + margin(B)

(Get the total Sum_y for all k in [2, 4])

if $Sum_x > Sum_y$ **then** choose y-axis as splitting axis.

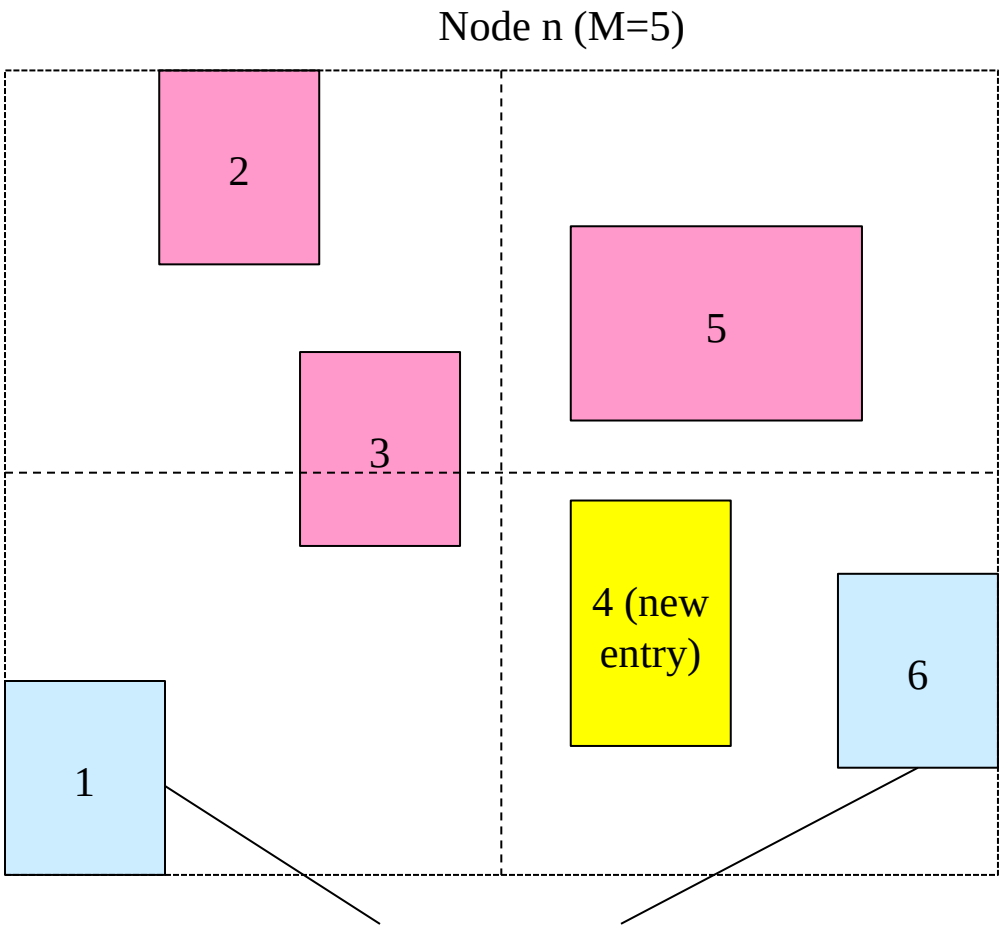
2. Distribute entries along axis

- Along the split axis, choose the distribution with minimum overlap
- If there are multiple groupings with minimal overlap choose <A,B> such that area(A)+area(B) is minimized

Insertion heuristics: Forced Reinsert

- Forced Reinsert:
  - When R*-tree node *n* overflows, instead of splitting n immediately, try to see if some entries in *n* could possibly fit better in another node
  - Find the 30% furthest entries from the center of the group
  - Re-insert them to the tree (not to be repeated if another overflow occurs)
- Slightly more expensive, but better tree structure:
  - less overlap
  - more space is utilized (more full nodes)

Forced Reinsert

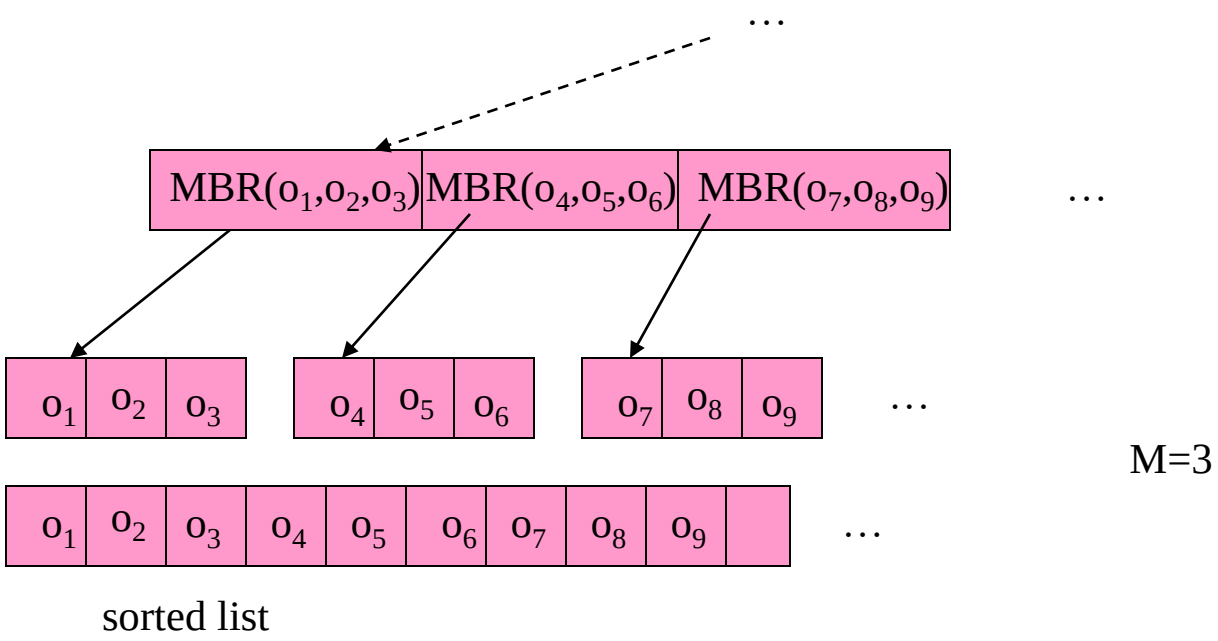


Bulk-loading R-trees

- Given a static set S of rectangles, build an R-tree that indexes S.
- Method 1: iteratively insert rectangles into an initially empty tree
  - tree reorganization is slow
  - tree nodes are not as full as possible: more space occupied for the tree
- Method 2 (x-sorting): bulk-load the rectangles into the tree using some fast (sort or hash-based) process
  - R-tree is built fast
  - good space utilization

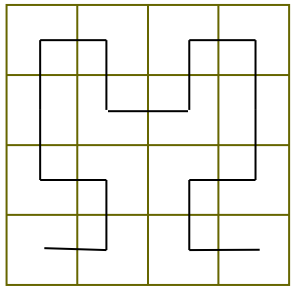
Bulk-loading R-trees

- Method 2 (x-sorting): Sort using only one axis
  - sort rectangles using the x-coordinate of their center
  - pack M consecutive rectangles in leaf nodes
  - build tree bottom-up



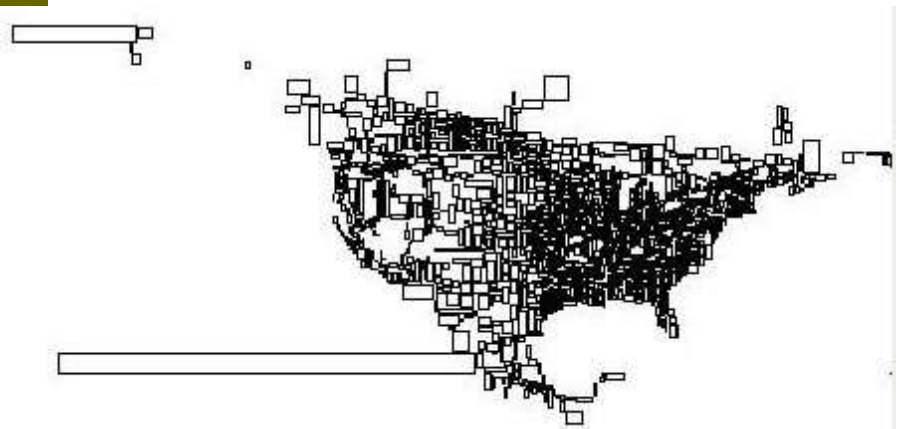
Bulk-loading R-trees

- Method 2 (x-sorting) results in leaf nodes that are have long stripes as MBRs
- Method 3 (Hilbert sorting): use a space-filling curve to order the rectangles
  - Results in better structure than x-sorting

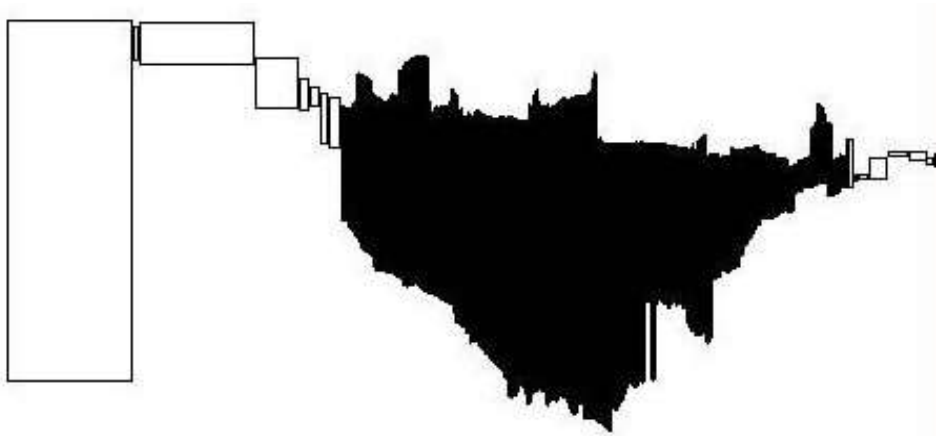


Hilbert-curve

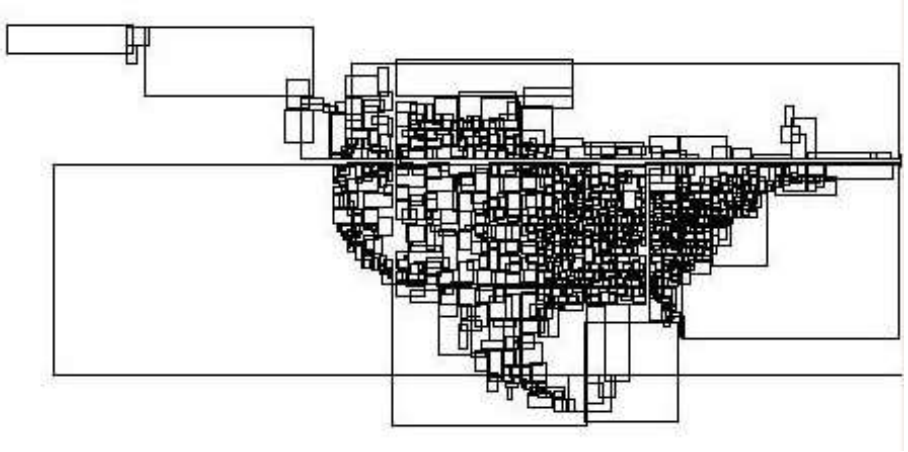
R-tree leaf nodes by different construction methods



(a) R*-tree insertion



(b) *x*-sorting



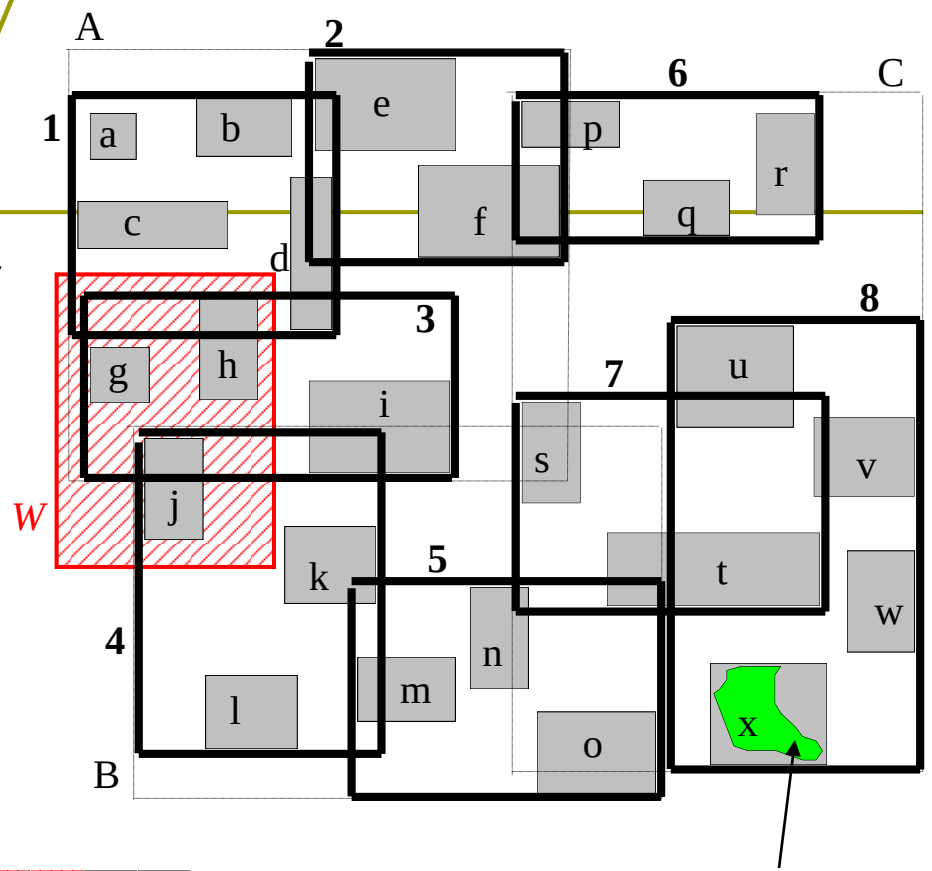
(c) Hilbert sorting

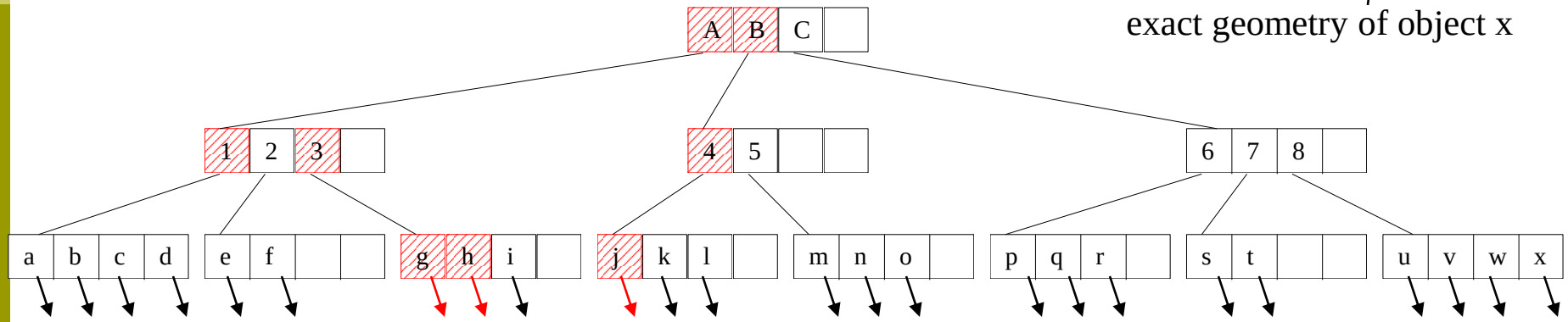
Spatial Data Management

- Spatial Data
- Spatial Relationships
- Spatial Queries
- Issues in Query Processing
- The R-tree
- Spatial Query Processing
  - Spatial Selections
  - Nearest Neighbor Queries
  - Spatial Joins

Spatial Selection/ Range Searching

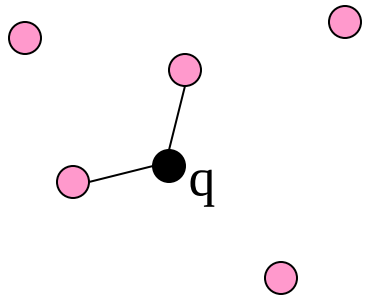
- *W* is a window query (range intersection query)





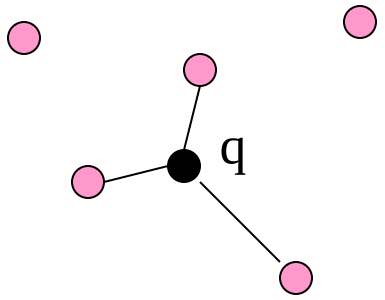
Nearest neighbor search

- Basic problem:
  - Given a spatial relation R and a query object q, find the nearest neighbor of q in R
  - Formally:
    - NN(q,R) = o $\in$ R: dist(q,o) $\leq$ dist(q,o'), $\forall$ o' $\in$ R
- Note:
  - We can have more than one NN (with equal minimum distance)



Nearest neighbor search

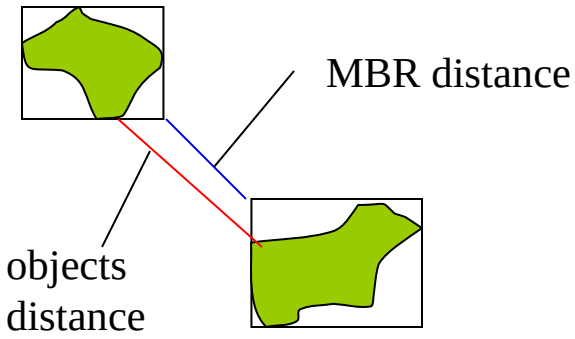
- Generalized problem:
  - Given a spatial relation R, a query object q, and a number k <|R|, find the k-nearest neighbors of q in R
  - Formally:
    - NN(q,k,R) = S $\subset$ R : |S| = k, dist(q,o) $\leq$ dist(q,o'), $\forall$ o $\in$ S $\forall$ o' $\in$ R-S
- Note:
  - We can have more than one k-NN sets (with multiple possible equidistant furthest points in them)
- Simplification
  - NN (and k-NN) operations return any NN (and k-NN sets)
  - We usually focus on point-sets



3-NN(q,R)

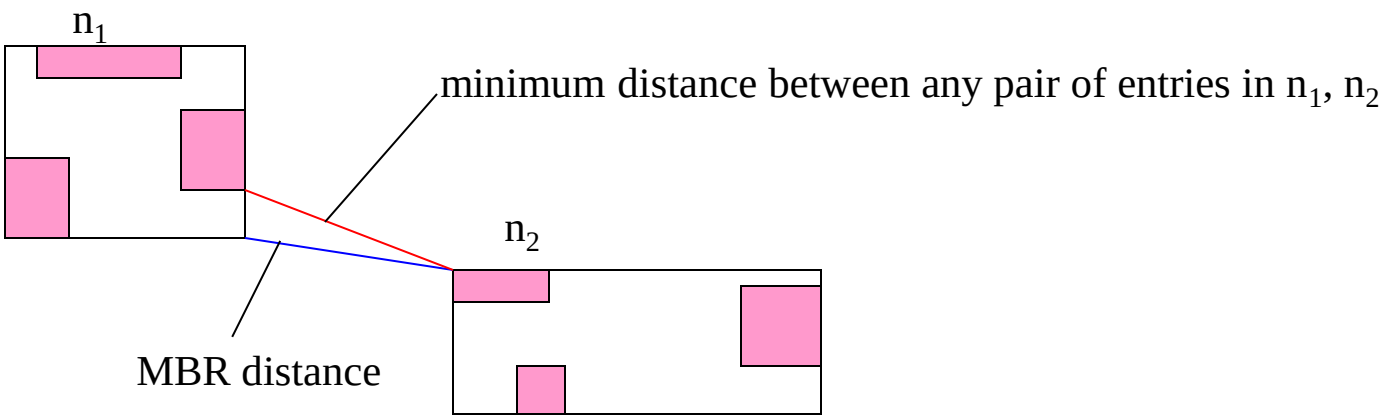
Distance measures and MBRs

- Distances between MBRs lower-bound the distances between the corresponding objects
  - $dist(\text{MBR}(o_i), \text{MBR}(o_j)) \leq dist(o_i, o_j)$



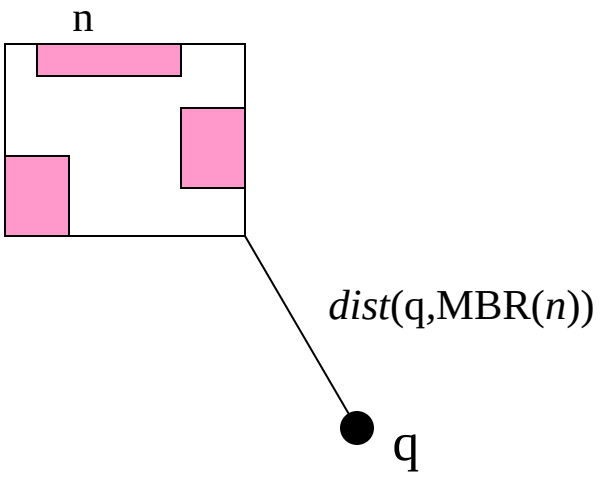
Distance measures and MBRs

- Distances between R-tree node MBRs lower-bound the distances between the entries in them
  - $dist(\text{MBR}(n_i), \text{MBR}(n_j)) \leq dist(e_i.\text{MBR}, e_j.\text{MBR})$, $\forall\ e_i \in n_i, e_j \in n_j$



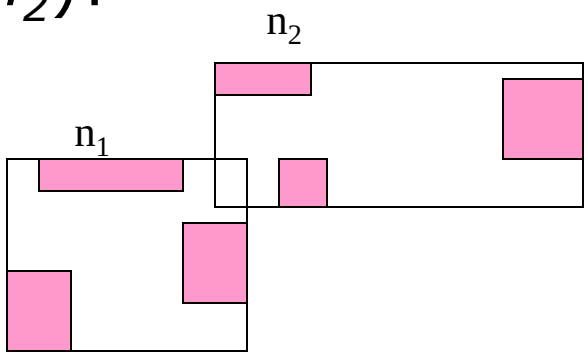
Distance measures and MBRs

- The distance between a query object q and an R-tree node MBR lower-bounds the distances between q and the objects indexed under this node
  - *dist*(q,MBR(*n*)) ≤ *dist*(q,o) ∀ *o* indexed under *n*



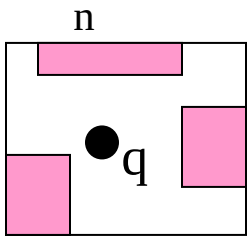
Questions

- What is the distance between MBR(n_1) and MBR(n_2)?



$dist(\text{MBR}(n_1),\text{MBR}(n_2)) = ?$

- What is the distance between q and MBR(n)?

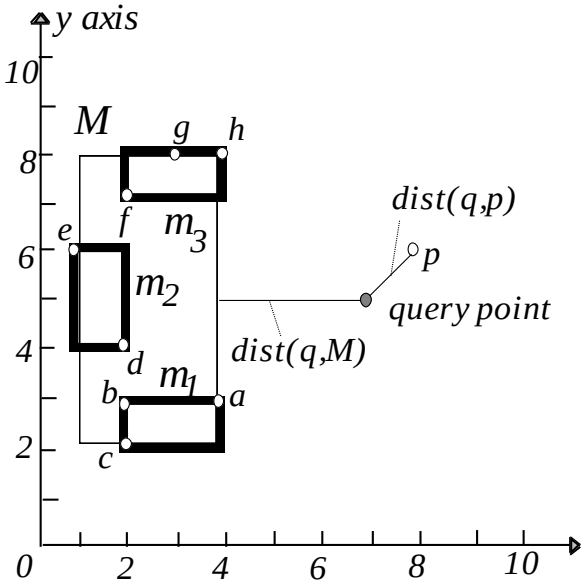


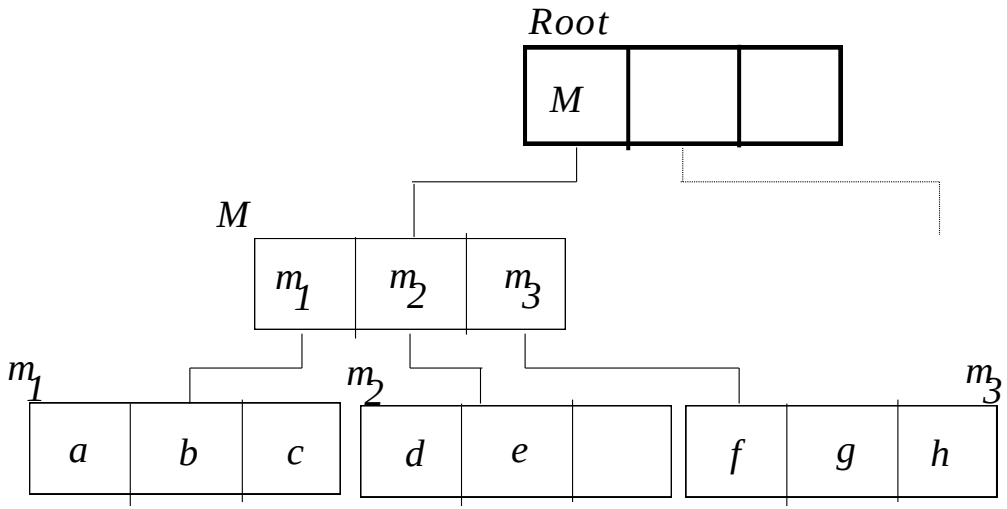
$dist(\text{q},\text{MBR}(n)) = ?$

- What can we conclude from these distances?

Using MBR distances to guide/prune search in an R-tree

- Problem: find the NN of q
- Do we need to look for it in node M if we know dist(q,p)?





Depth-first NN search using an R-tree

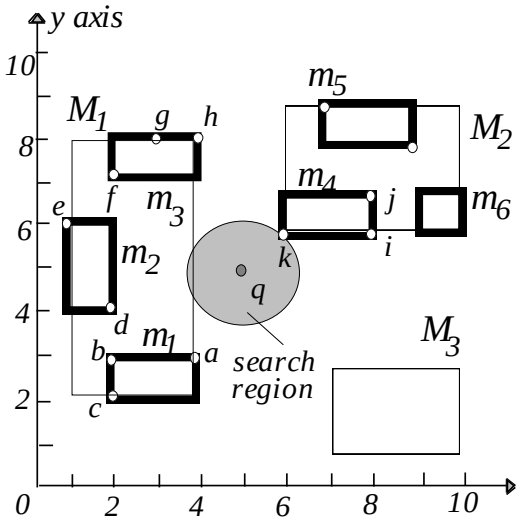
- Start from the root and visit the node nearest to q
- Continue recursively, until a leaf node n_l is visited.
- Find the NN of q in n_l.
- Continue visiting other nodes after backtracking as long there are nodes closer to q than the current NN.

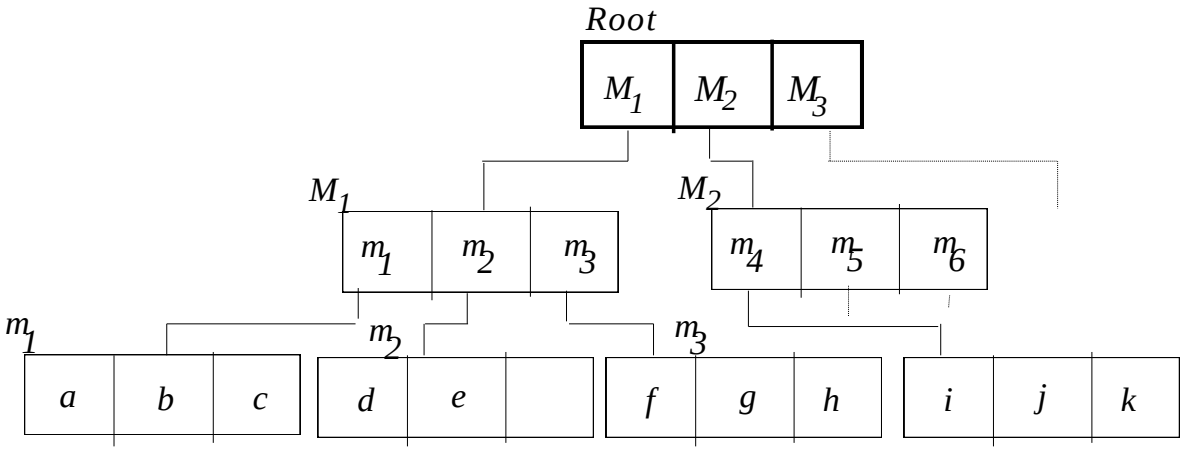
Depth-first NN search using an R-tree

```
function DF_NN_search(object q, Node n, object o_NN)
    if n is a leaf node then
        for each entry e ∈ n
            if dist(q, e.MBR) < dist(q, o_NN) then
                /* e.MBR is closer to q than its current NN*/
                o := object with address e.ptr;
                if dist(o, q) < dist(q, o_NN) then
                    /* found new NN */
                    o_NN := o;
    else /* n is not a leaf node */
        for each entry e ∈ n
            if dist(q, e.MBR) < dist(q, o_NN) then
                /* it is possible that n indexes the NN */
                n' := R–tree node with address e.ptr;
                DF_NN_search(q, n');
```

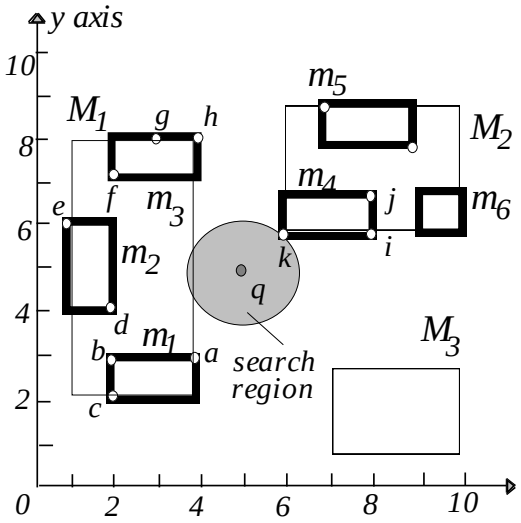
Initially an imaginary object with $dist(q,o_{NN}) = \infty$

Depth-first NN search using an R-tree





Depth-first NN search using an R-tree



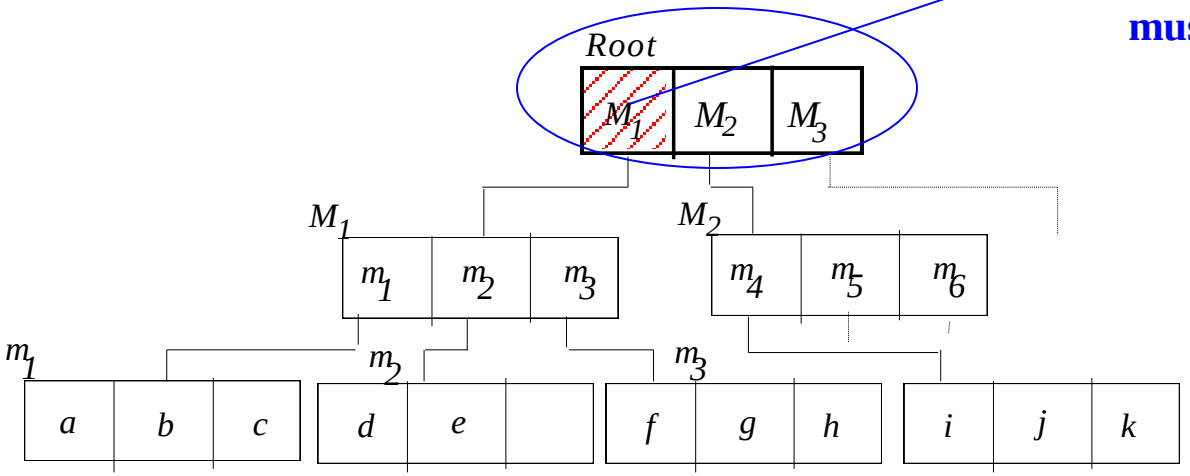
o_{NN} = NULL

$dist(q,o_{NN})= \infty$

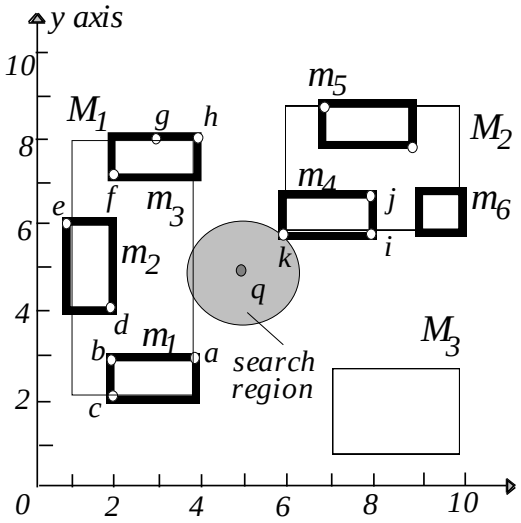
1. visit root

$dist(q,M_1)<dist(q,oNN)$

must visit node M_1



Depth-first NN search using an R-tree



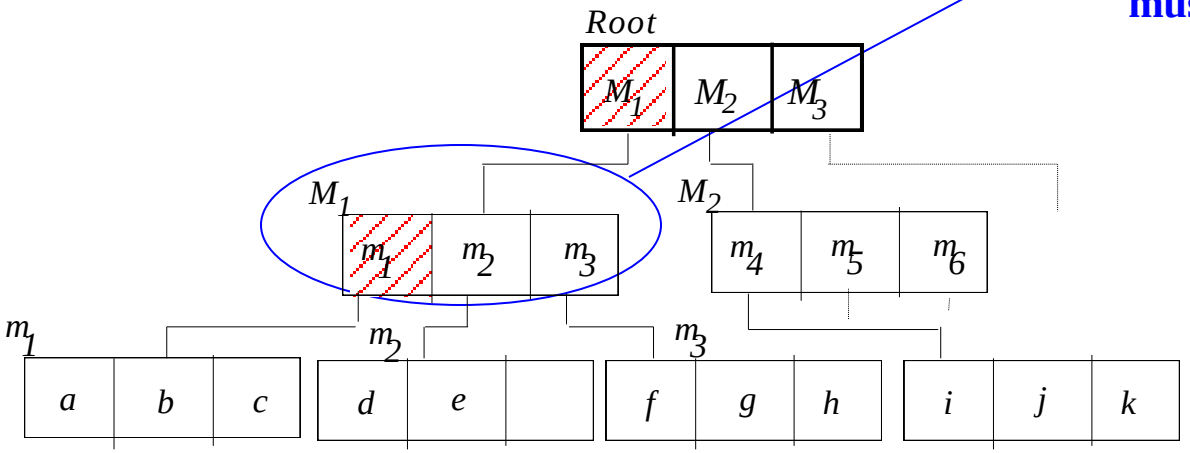
o_{NN} = NULL

$dist(q,o_{NN})= \infty$

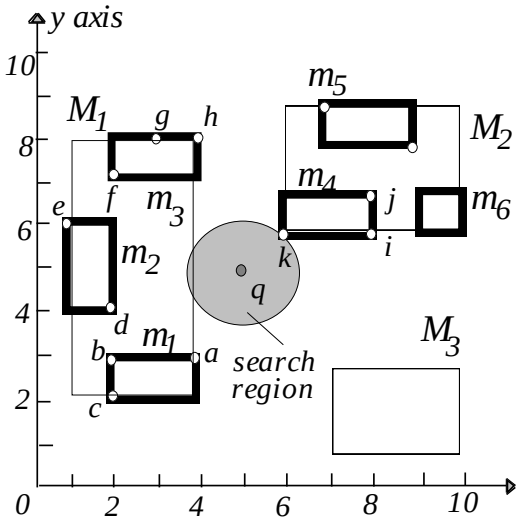
2. visit M_1

$dist(q,m_1)<dist(q,oNN)$

must visit node m_1



Depth-first NN search using an R-tree



o_{NN} = NULL

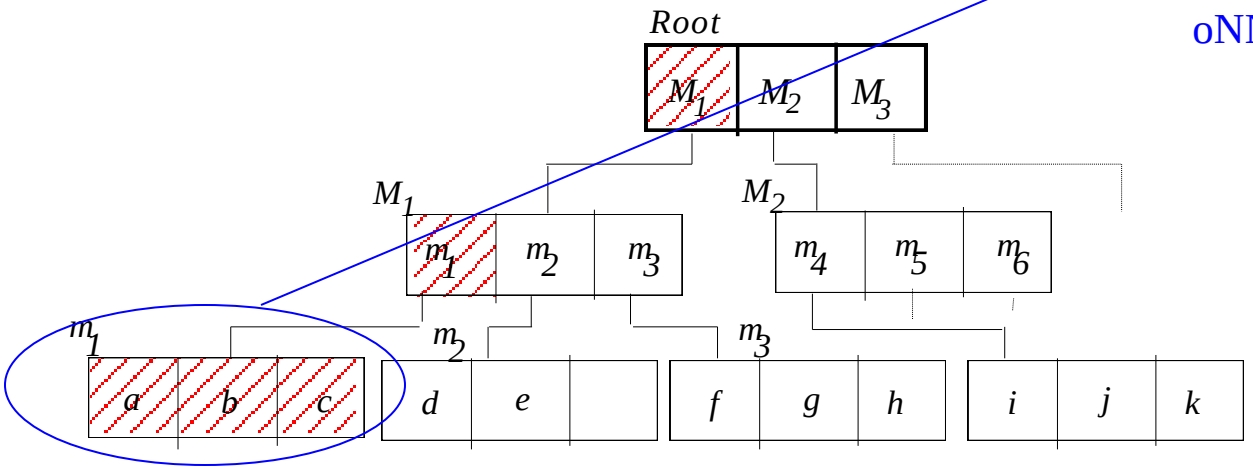
$dist(q,o_{NN}) = \infty$

3. visit m_1

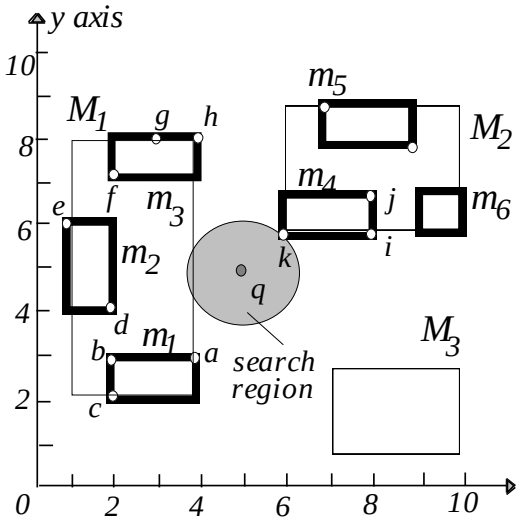
check a,b,c

found new NN:

oNN = a, $dist(q,o_{NN}) = \sqrt{5}$

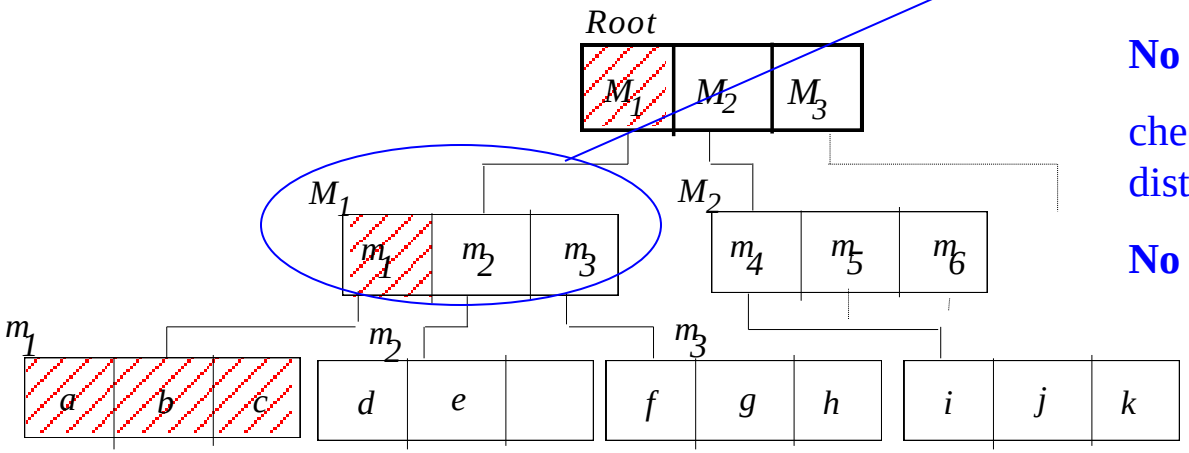


Depth-first NN search using an R-tree



o_{NN} = a

dist(q,o_{NN})= √5



4. backtrack to M_1

check m_2

dist(q,m_2) = 3 ≥ √5:

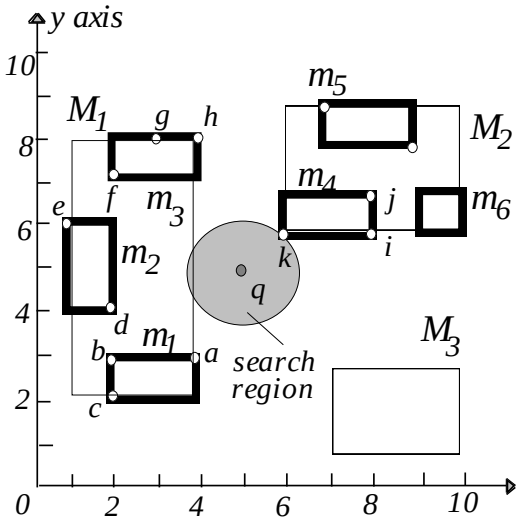
No need to visit node m_2

check m_3

dist(q,m_3) = √5 ≥ √5:

No need to visit node m_3

Depth-first NN search using an R-tree



o_{NN} = a

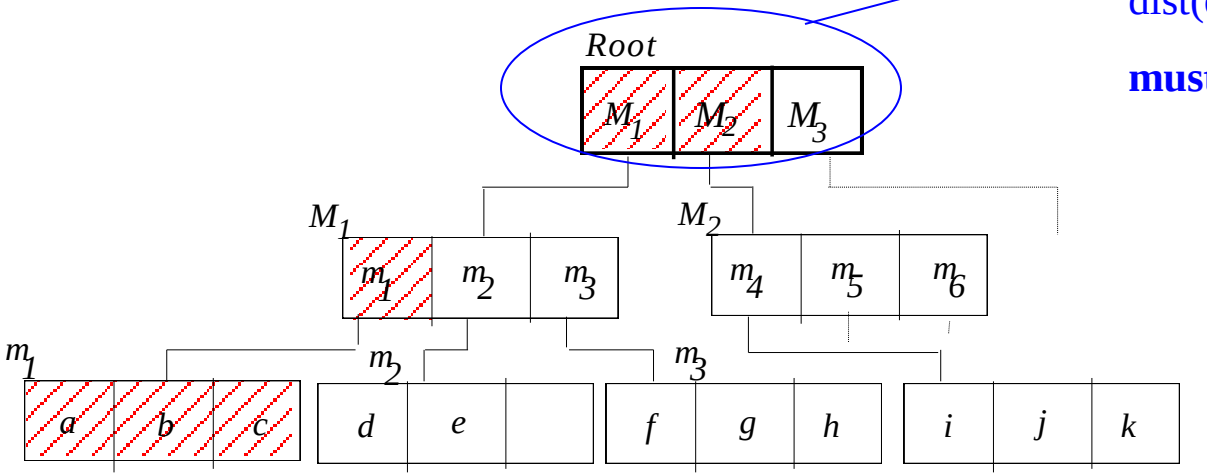
dist(q,o_{NN})= √5

5. backtrack to root

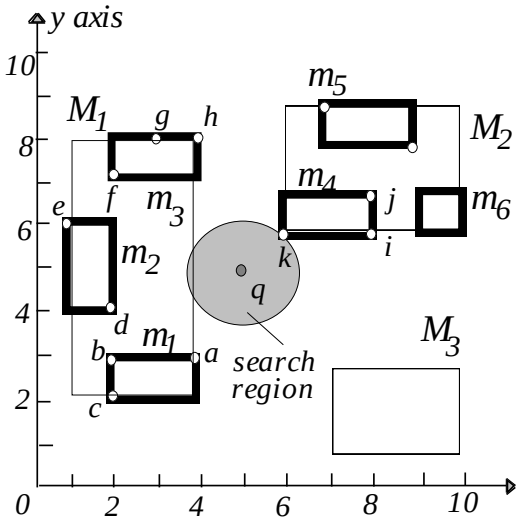
check M_2

dist(q,M_2) = √2 < √5:

must visit node M_2



Depth-first NN search using an R-tree



o_{NN} = a

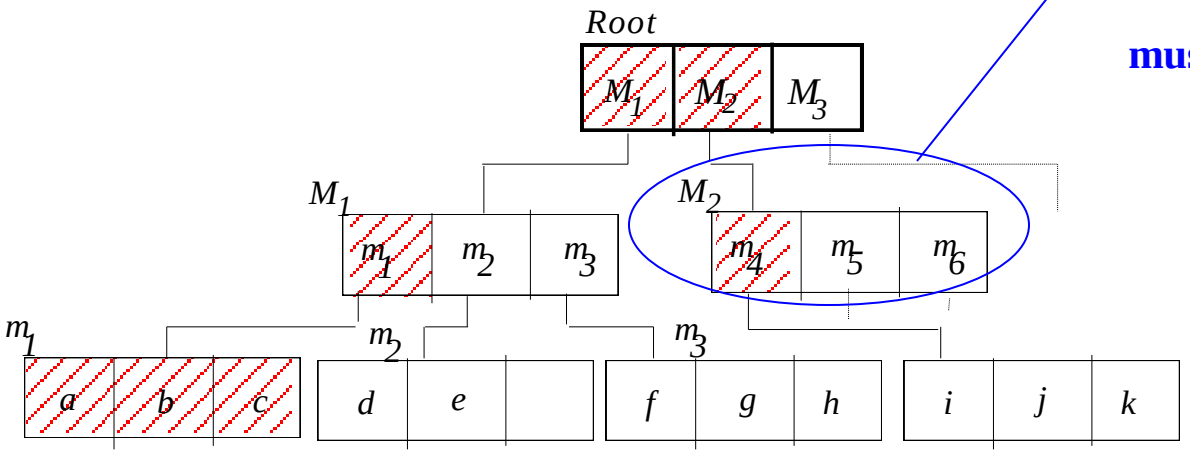
dist(q,o_{NN})= √5

6. visit M_2

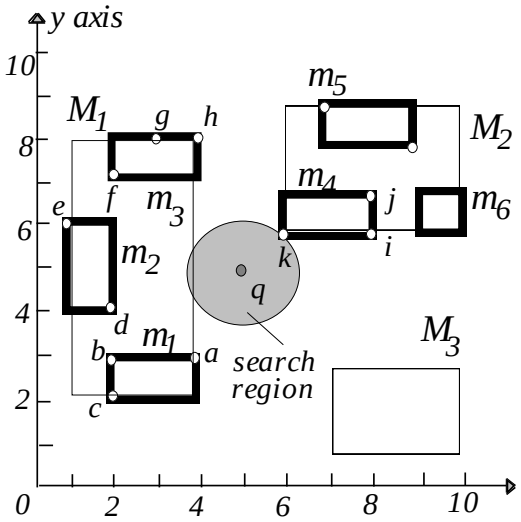
check m_4

dist(q,m_4) = √2 < √5:

must visit node m_4



Depth-first NN search using an R-tree



o_{NN} = a

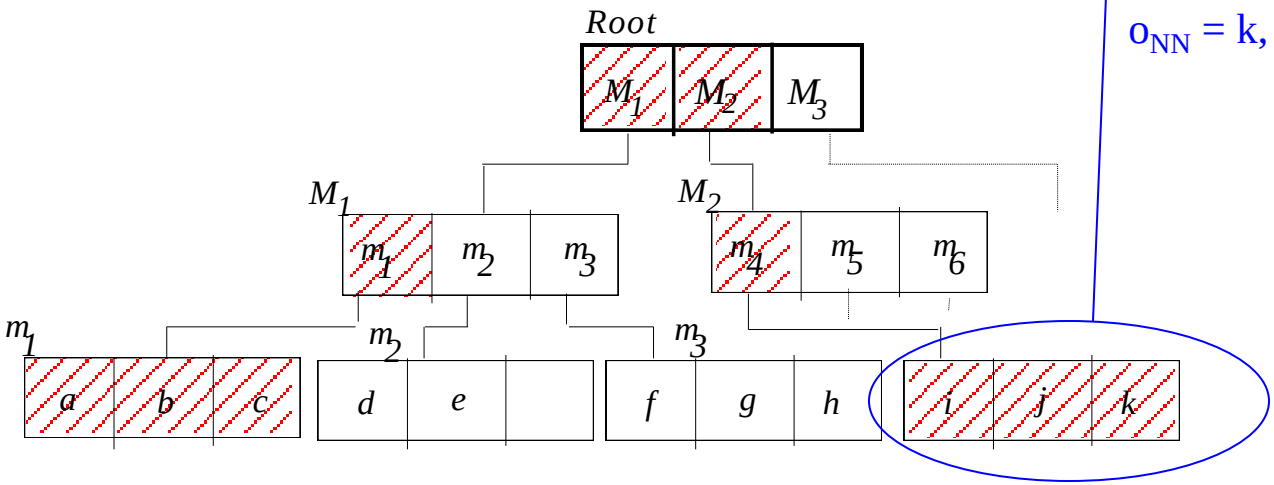
dist(q,o_{NN})= √5

7. visit m_4

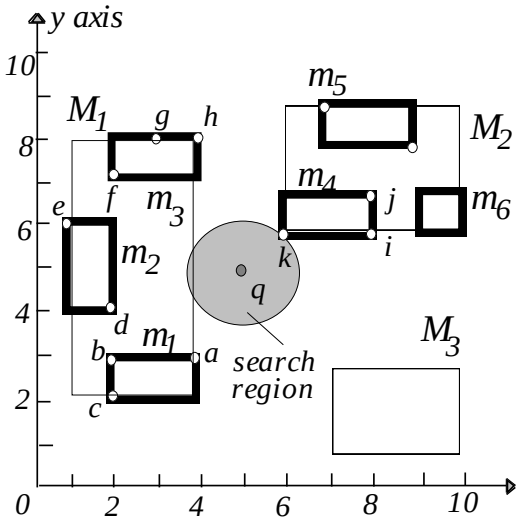
check i,j,k

found new NN:

o_{NN} = k, dist(q,o_{NN}) = √2

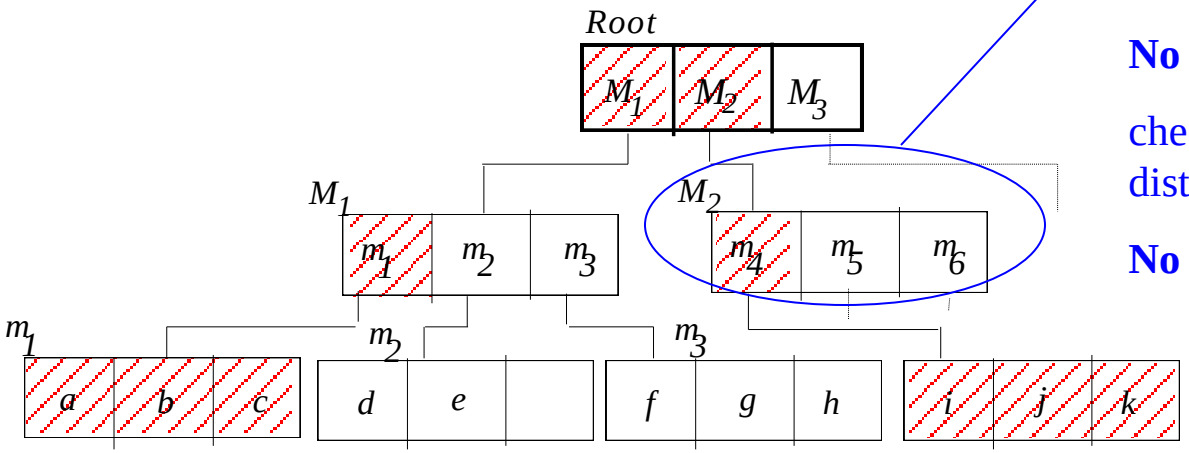


Depth-first NN search using an R-tree



o_{NN} = k

dist(q,o_{NN})= √2



8. backtrack to M_2

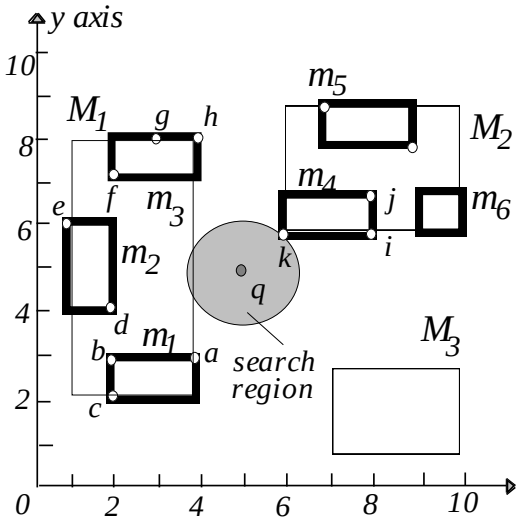
check m_5
dist(q,m_5) ≥ √2:

No need to visit node m_5

check m_6
dist(q,m_6) ≥ √2:

No need to visit node m_6

Depth-first NN search using an R-tree



o_{NN} = k

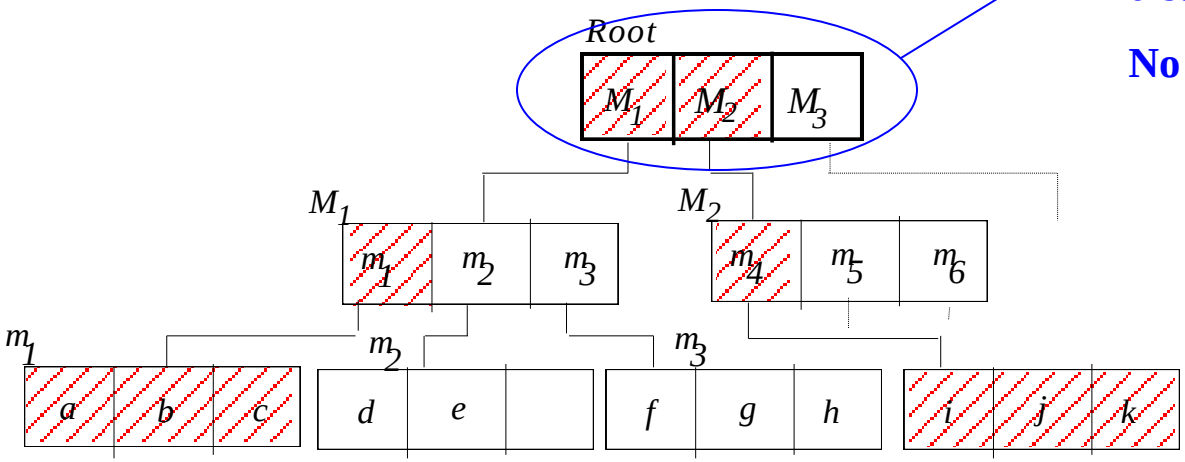
dist(q,o_{NN})= √2

9. backtrack to root

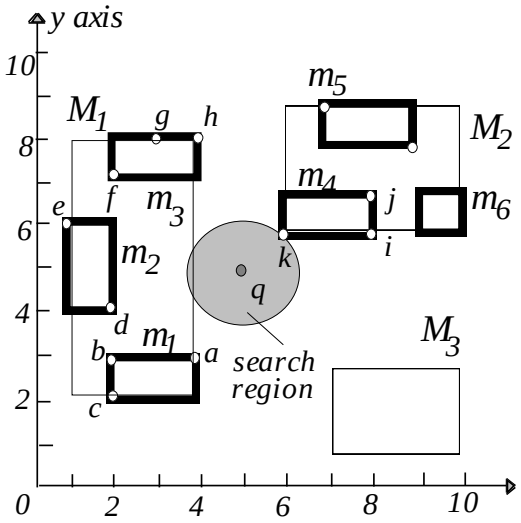
check M_3

dist(q,M_3) ≥ √2:

No need to visit node M_3



Depth-first NN search using an R-tree



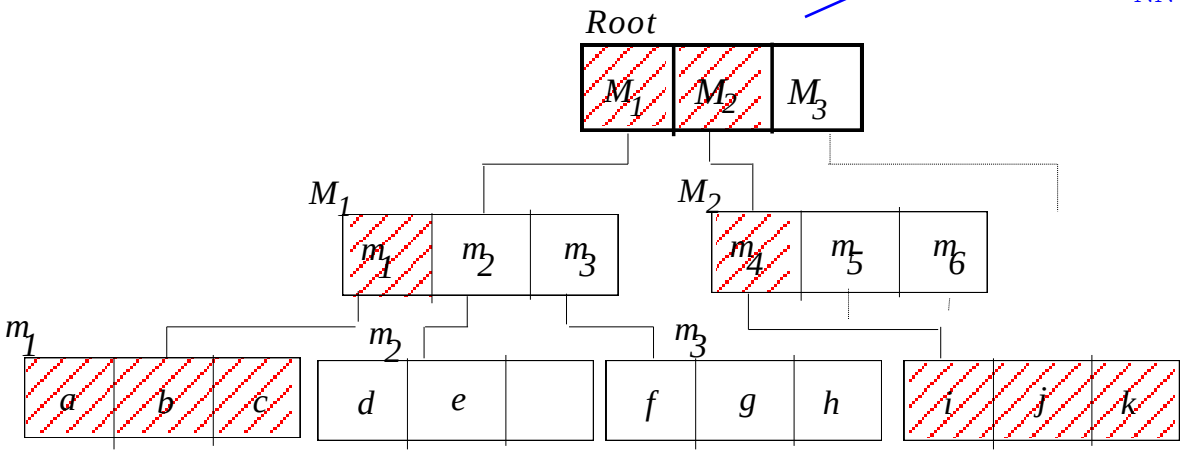
o_{NN} = k

dist(q,o_{NN})= √2

10. backtrack from root

Algorithm terminates

o_{NN} =k with dist(q,o_{NN})= √2 found



Notes on Depth-first NN search

- Large space can be pruned by avoiding visiting R-tree nodes and their sub-trees
- Should order the entries of a node in increasing distance from q to maximize potential for a good NN found fast
- Can be easily adapted for k-NN search (how?)
- Requires at most one tree path to be currently in memory – good for small memory buffers
  - Characteristic of all depth-first search algorithms
  - Recall that the range search algorithm is also DF
- However, does not visit the least possible number of nodes
- Also, not incremental – more on this later…

Best-first NN search

- A more efficient algorithm (given large enough memory)
- Optimal in the number of R-tree nodes visited for a given query q
- Uses a priority queue to organize seen entries and prioritize the next node to be visited
- Adaptable for k-NN search and incremental NN search

Best-first NN search

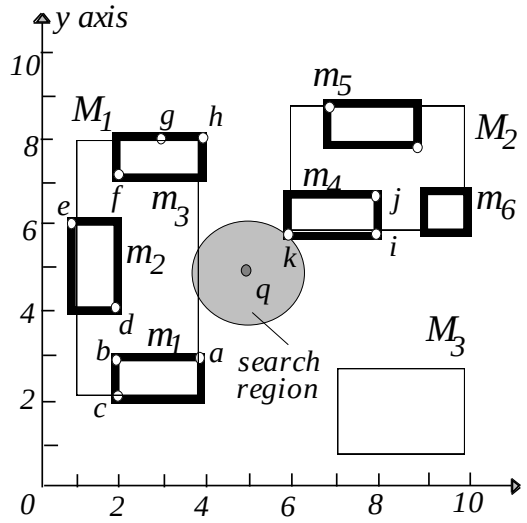
- Observation about DF-search:
  - The closest entry to q in the current node is “opened” and control is passed to the node pointed by it
  - However the entries in that node may not be the closest entries to q from those seen so far
- Idea of BF search:
  - Put all entries in a priority queue and always “open” the closest one, independently of the node that contains it
  - Thus the best (i.e., closest) entry is always visited first

Best-first NN search

```
function BF NN search(object q, R–tree R): object o_NN
    o_NN := NIL; dist(q, o_NN) := ∞
    initialize a priority queue Q;
    add all entries of R's root to Q;
    while not empty(Q) and dist(q, top(Q).MBR) < dist(q, o_NN)
        e := top(Q);
        remove e from Q;
        if e is a directory node entry then
            n := R–tree node with address e.ptr;
            for each entry e' ∈ n
                if dist(q, e'.MBR < dist(q, o_NN) then
                    add e' on Q;
        else /* e is an entry of a leaf node */
            o := object with address e.ptr;
            if dist(o, q) < dist(q, o_NN) then
                /* found new NN */
                o_NN := o;
    return o_NN;
```

Figure 3.14: Best-first nearest neighbor search using an R–tree

Best-first NN search



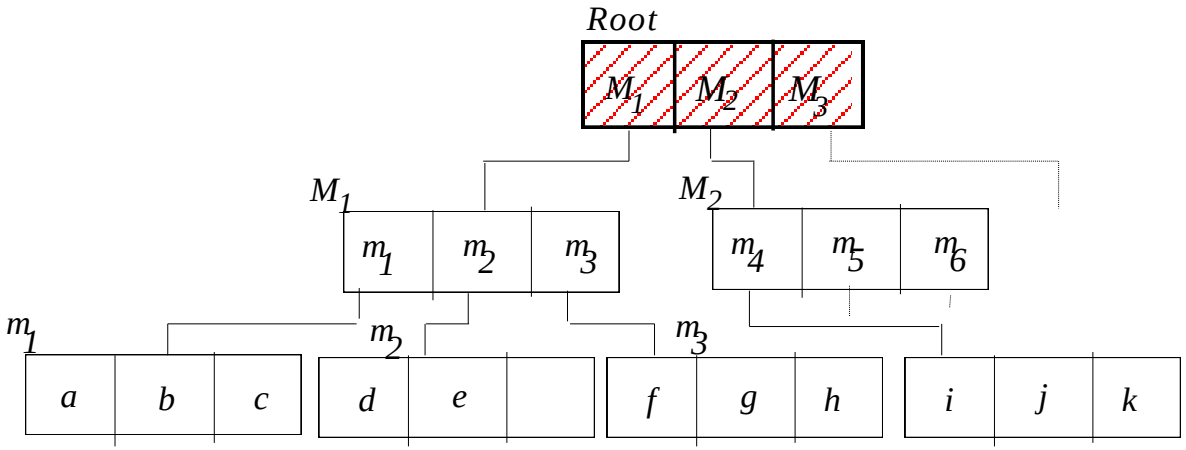
o_{NN} = NULL

$dist(q,o_{NN}) = \infty$

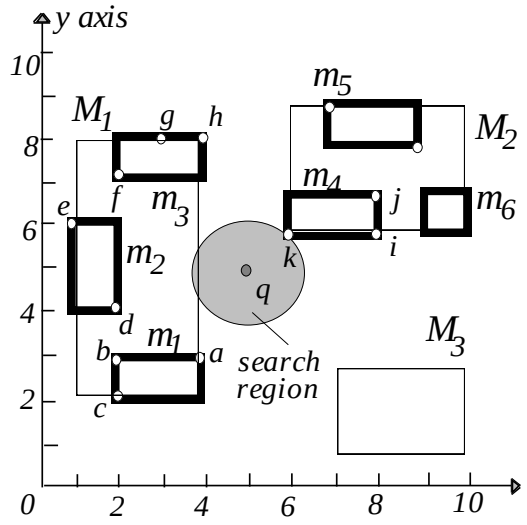
Step 1: put all entries of root on heap Q

$Q = M_1(1), M_2(\sqrt{2}), M_3(\sqrt{8})$

distance from q



Best-first NN search



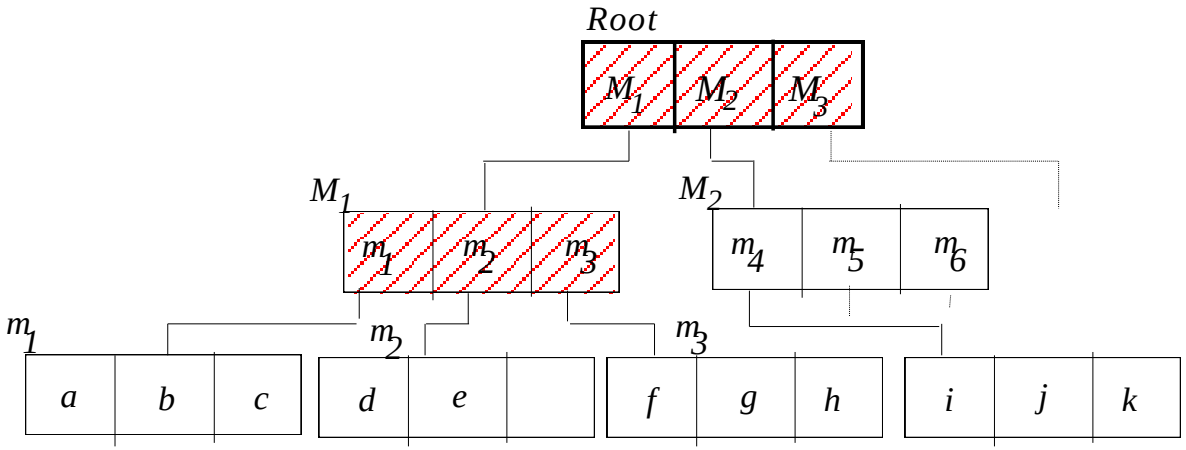
o_{NN} = NULL

$dist(q,o_{NN})= \infty$

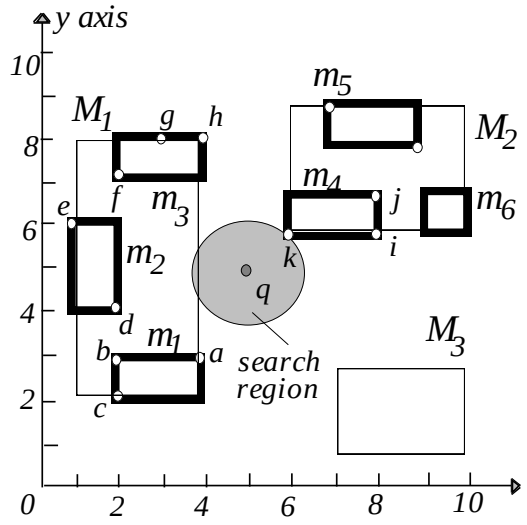
Step 2: get closest entry (top element of Q):

$M_1(1)$. Visit node M_1. Put all entries of visited node on heap Q

$Q = M2(\sqrt{2}), m1(\sqrt{5}), m3(\sqrt{5}), M3(\sqrt{8}), m2(3)$



Best-first NN search



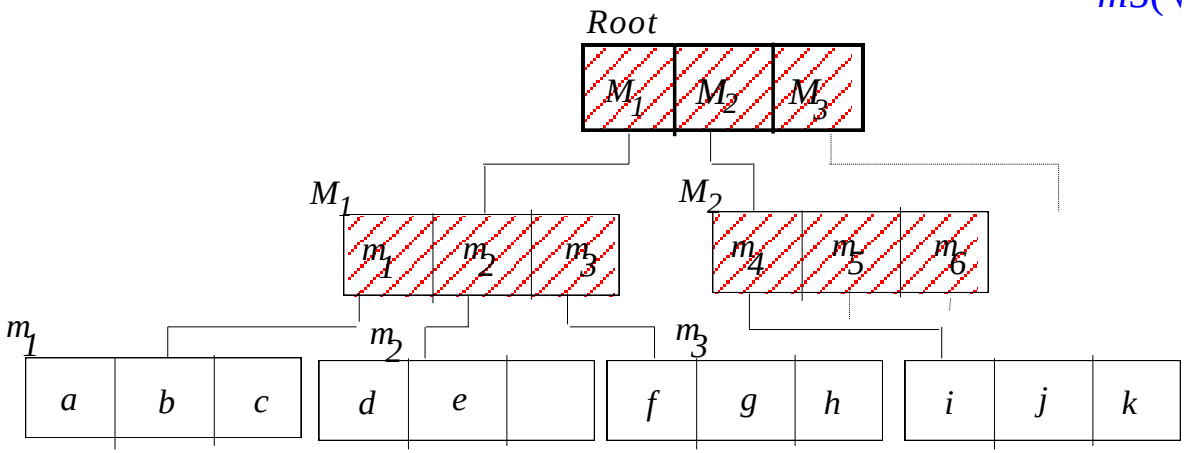
o_{NN} = NULL

$dist(q,o_{NN})= \infty$

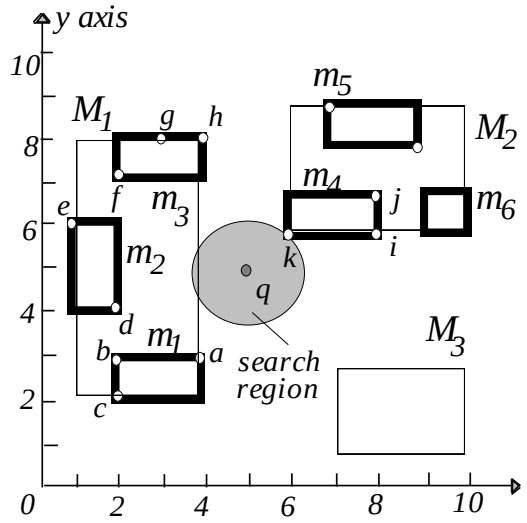
Step 3: get closest entry (top element of Q):

$M_2(\sqrt{2})$. Visit node M_2. Put all entries of visited node on heap Q

$Q = m4(\sqrt{2}), m1(\sqrt{5}), m3(\sqrt{5}), M3(\sqrt{8}), m2(3), m5(\sqrt{13}), m5(\sqrt{17})$



Best-first NN search



o_{NN} = NULL

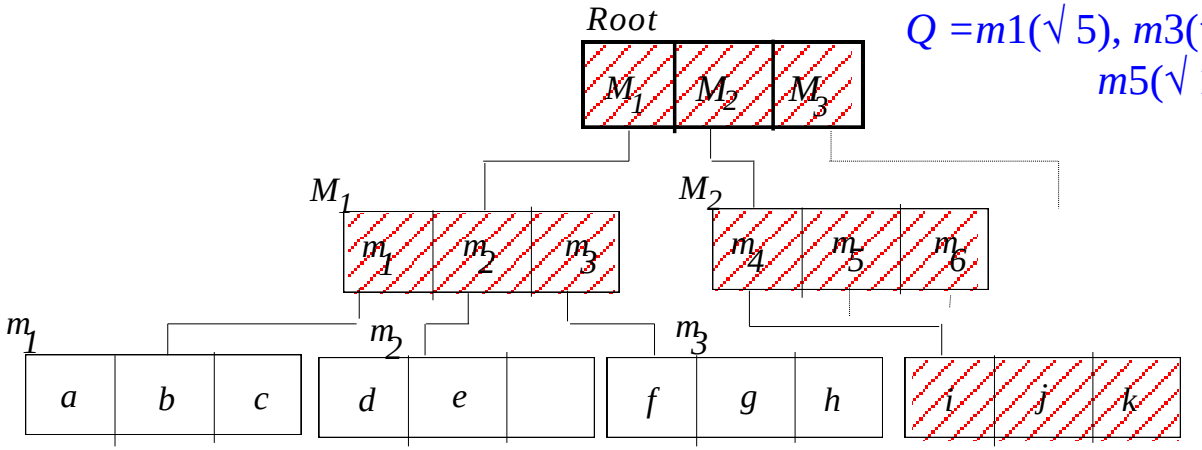
dist(q,o_{NN})= ∞

Step 4: get closest entry (top element of Q):

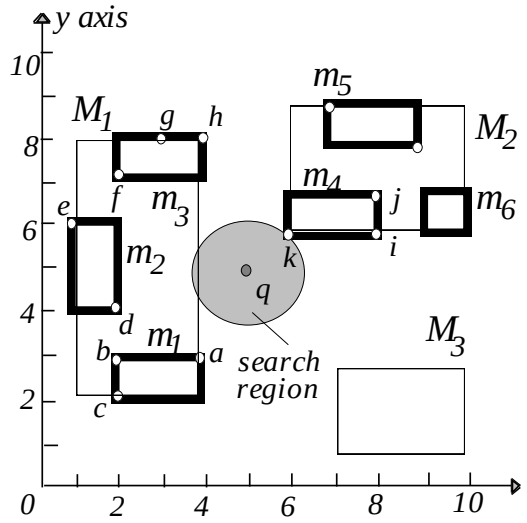
m_4(√ 2). Visit node m_4. m_4 is a leaf node, so update NN if some object in *m*4 is closer than the current NN:

o_{NN} = k, dist(q,oNN)= √2

Q =*m*1(√ 5), *m*3(√ 5), *M*3(√ 8), *m*2(3), *m*5(√ 13), *m*5(√17)



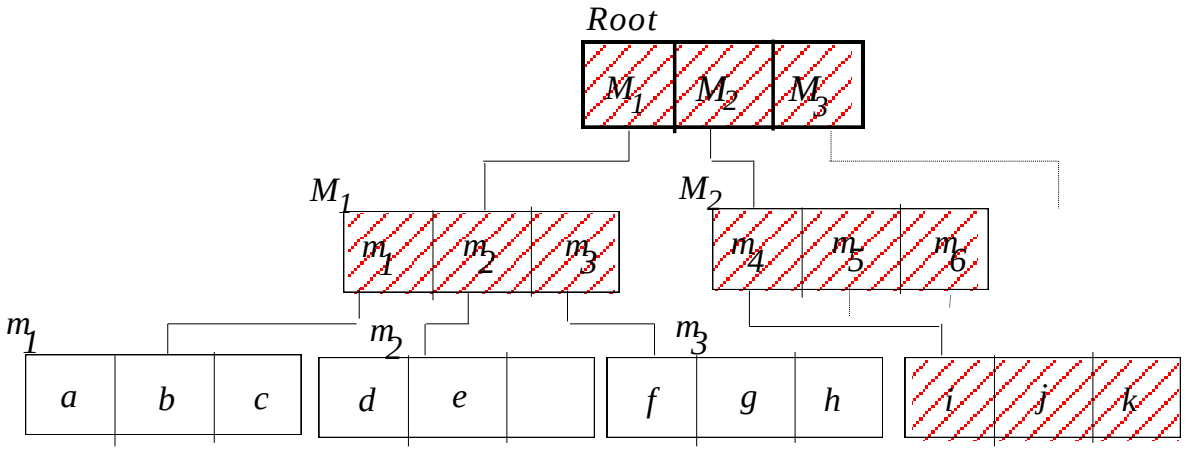
Best-first NN search



oNN = k, dist(q,oNN)= √2

Step 5: get closest entry (top element of Q):

m_1(√5). Since √5 ≥ dist(q,oNN)= √2, search stops and o_{NN} is returned as the NN of q



Notes on Best-first NN search

- In the previous example, we have visited fewer nodes compared to DF-NN algorithm
  - Only nodes whose MBR intersect the disk centered at q with radius the real NN distance are visited (see if you can you prove this)
- The algorithm can be adapted for incremental NN search
  - After having found the NN can we easily (incrementally) find the next NN without starting search from the beginning?
    - put objects on the heap
    - never prune, but wait until an object comes out
- The algorithm can be used for k-NN search
  - use a second heap to organize the NN found so far (same can be done for DF-NN)
  - no need if we just use the inc. version of the algorithm
- … but: The heap can grow very large until the algorithm terminates

Why incremental NN search?

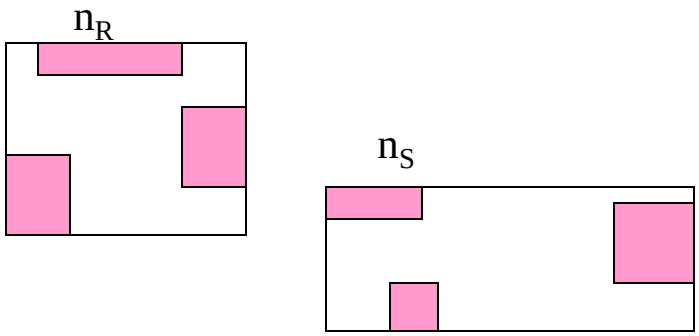
- Example 1: find the nearest large city (>10,000 residents) to my current position
  - Solution 1:
    - find all large cities
    - apply NN search on the result
    - could be slow if many such cities
    - also R-tree may not be available for large cities only
  - Solution 2:
    - incrementally find NN and check if the large city requirement is satisfied; if not get the next NN
- Example 2: find the nearest hotel; see if you like it; if not get the next one; see if you like it; …

Spatial Joins

- Input:
  - two spatial relations R, S (e.g., R=cities, S=rivers)
  - a spatial relationship θ (e.g., θ=intersects)
- Output:
  - {(r,s): $r \in R$, $s \in S$, r θ s is true}
  - Example: find all pairs of cities and rivers that intersect
- We will discuss intersection joins (like the example above)

R-tree (Intersection) Join

- Applies on two R-trees of spatial relations R and S
- Observation:
  - If a node $n_R \in R$ does not intersect $n_S \in S$, then no object $o_R \in R$ under n_R can intersect any object $o_S \in S$ under n_S
  - Node MBRs at the high level of the trees can prune object combinations to be checked



- We can use this property to design an algorithm that synchronously traverses both trees following only entry pairs that intersect.

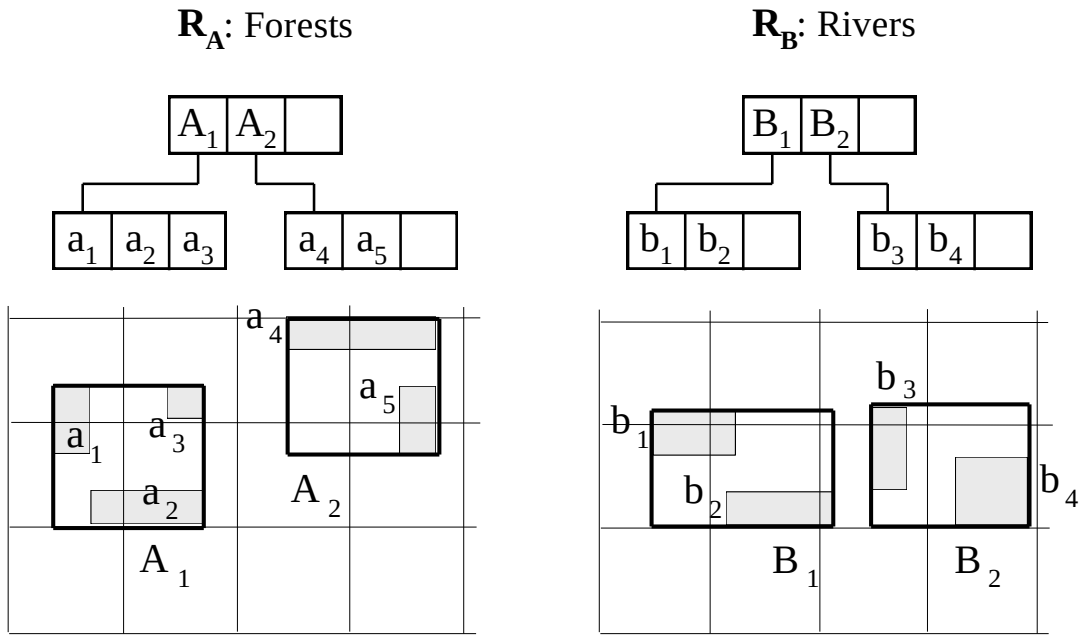
R-tree (Intersection) Join

- Initially called taking with parameters the roots of the two trees
- This pseudo-code version assumes that the trees have same height
  - easily extendable to trees of different heights

```
function RJ(Node n_R, Node n_S)
    for each e_i ∈ n_R
        for each e_j ∈ n_S, such that e_i.MBR ∩ e_j.MBR≠ ∅
            if n_R is a leaf node then /* n_S is also a leaf node */
                output (e_i.ptr,e_j.ptr); /* a pair of object-ids passing the filter step */
            else /* n_R, n_S are directory nodes */
                RJ(e_i.ptr,e_j.ptr); /* run recursively for the nodes pointed by intersecting entries*/
```

R-tree (Intersection) Join

- Example:
  - run for root(R_A), root(R_B)
  - for every intersecting pair there (e.g., A_1, B_1) run recursively for pointed nodes
  - intersecting pairs of leaf nodes are qualifying object MBR pairs

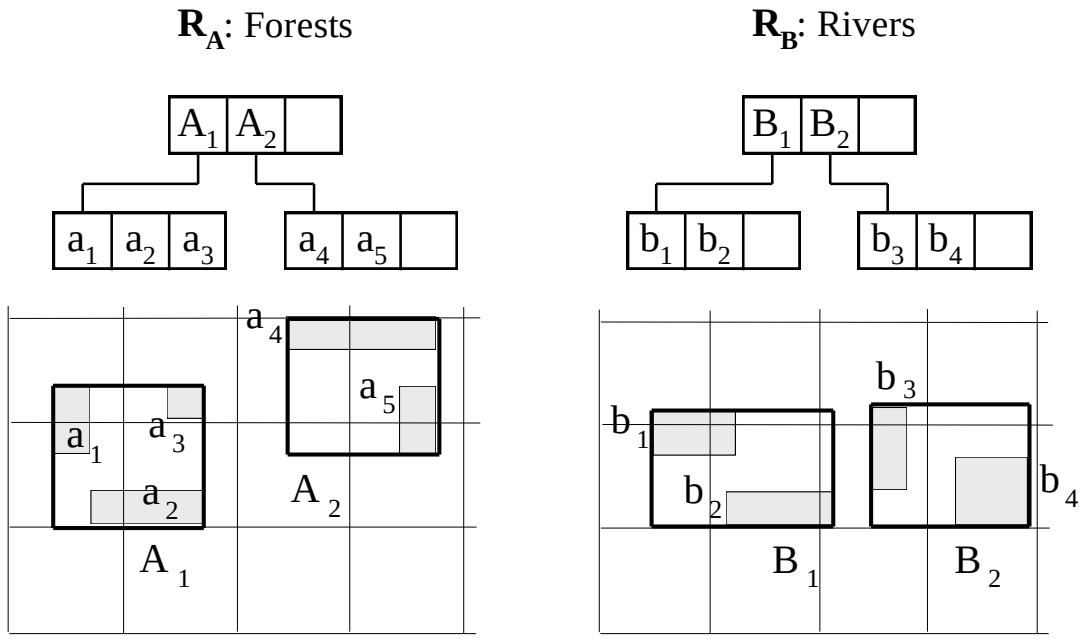


Level 1 qualifying pairs:
$\{(A_1, B_1), (A_2, B_2)\}$

Level 0 qualifying pairs:
$\{(a_1, b_1), (a_2, b_2)\}$

Computational issue

- How to decrease the cost of comparing entry pairs for a given pair of nodes
  - E.g., for (A_1, B_1), we need to compare $\{a_1,a_2,a_3\}x\{b_1,b_2\}$ = 6 pairs
  - In real R-trees nodes may have >100 entries (100x100 = 10000 entry pairs!)

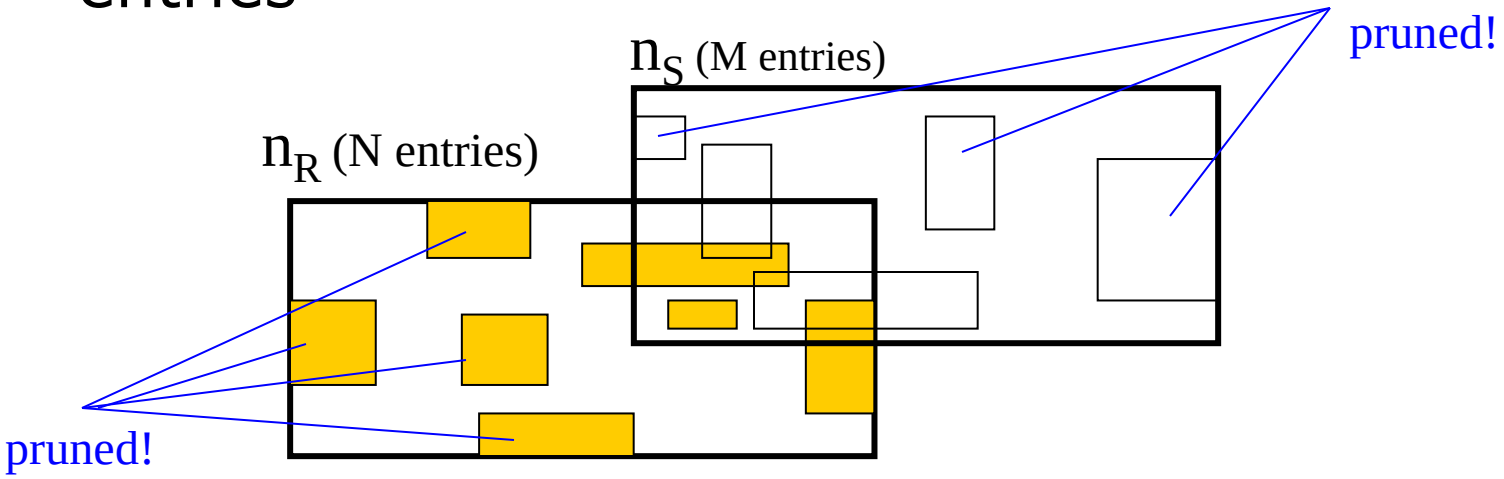


Level 1 qualifying pairs:
$\{(A_1, B_1), (A_2, B_2)\}$

Level 0 qualifying pairs:
$\{(a_1, b_1), (a_2, b_2)\}$

Optimizations (1) space restriction

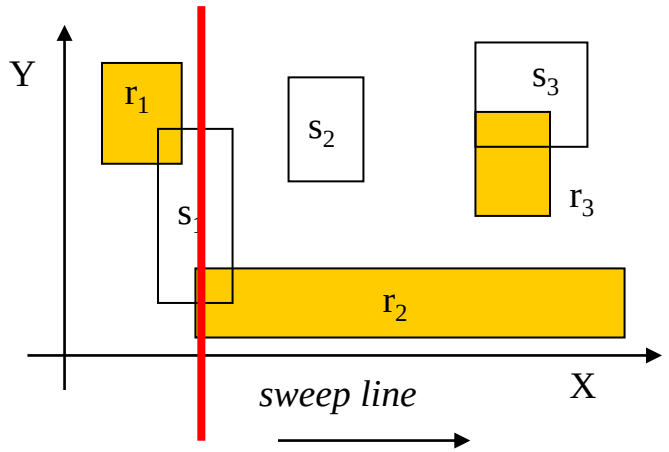
- If an entry in n_1 does not intersect the MBR of n_2 it may not intersect any entry in n_2.
- Perform two scans in n_1 and n_2 to prune such entries



Restricting the search space
cost: O(N)+O(M)

Optimizations (2) plane sweep

- Sort entries in both nodes on their lower-x value (lower bound of x-projection)
- Sweep a line to find fast all entry pairs that qualify x-intersection
  - for each of them check y-intersection



N entries ⋈ M entries

Spatial sorting and plane sweep
cost: O((N+M)·log(max(N,M))+k)

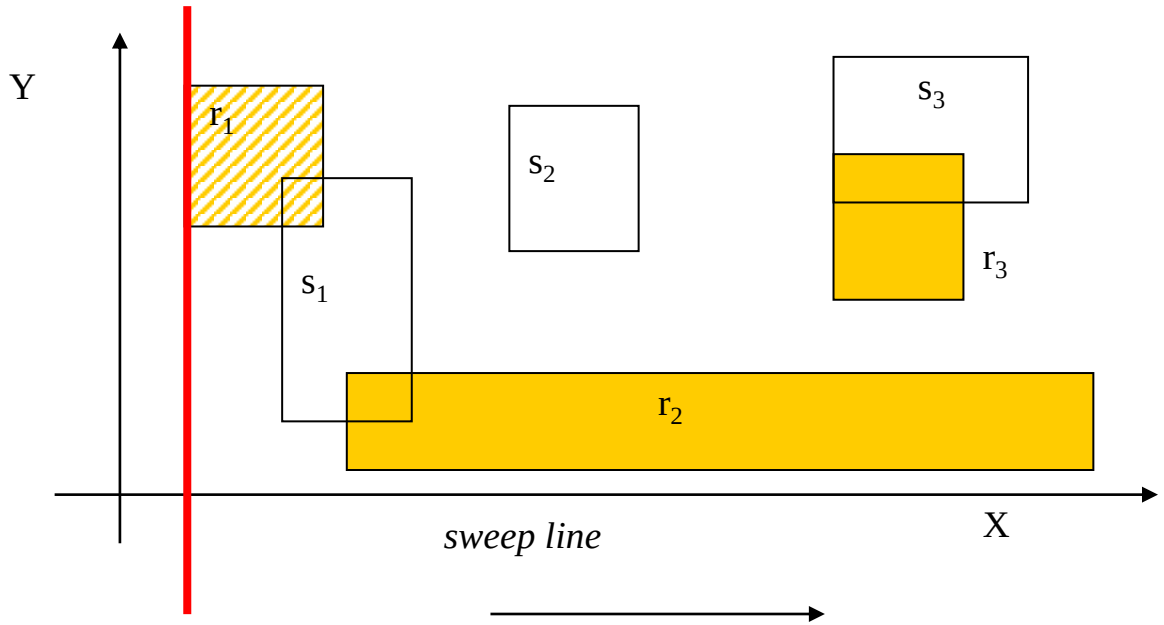
Sweep example

sorted lists:
R={r1,r2,r3}
S={s1,s2,s3}

last sweep line position in R: r1
last sweep line position in S: s1

r1<s1:
scan S from s1 until $s_i > r_1$ on x-axis
for each seen s_i check y-intersection:
pair $<r_1,s_1>$ found!

increase position in R to r2



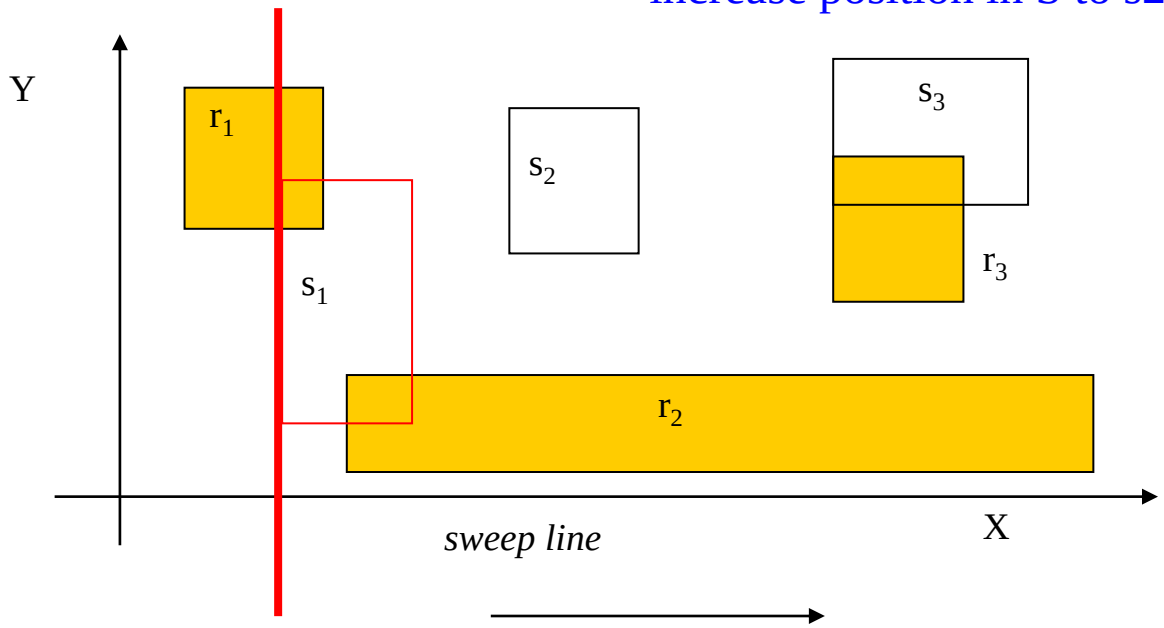
Sweep example

sorted lists:
R={r1,r2,r3}
S={s1,s2,s3}

last sweep line position in R: r2
last sweep line position in S: s1

s1<r2:
scan R from r2 until $r_i > s_1$ on x-axis
for each seen r_i check y-intersection:
pair $<r_2, s_1>$ found!

increase position in S to s2



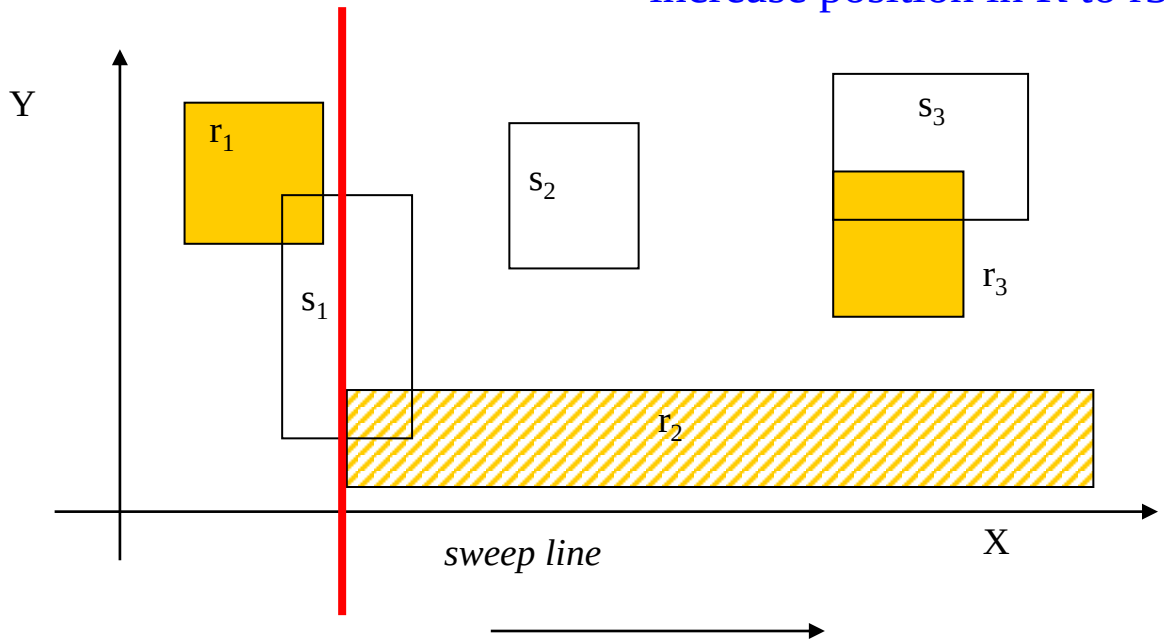
Sweep example

sorted lists:
R={r1,r2,r3}
S={s1,s2,s3}

last sweep line position in R: r2
last sweep line position in S: s2

r2<s2:
scan S from s2 until $s_i > r_2$ on x-axis
for each seen s_i check y-intersection:
no pair found!

increase position in R to r3



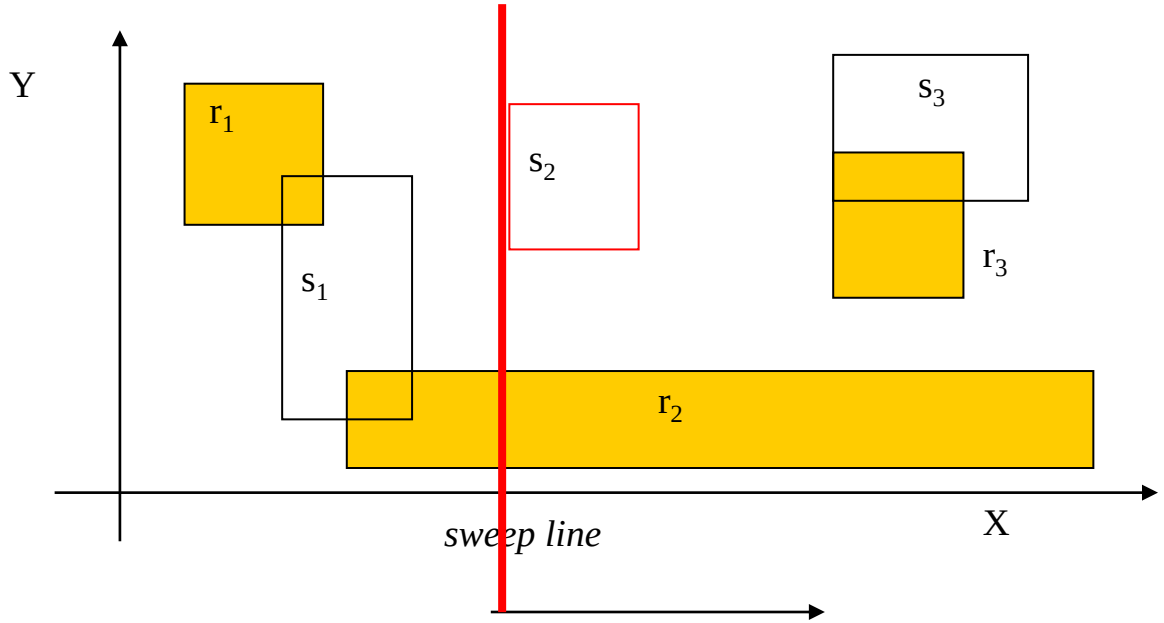
Sweep example

sorted lists:
R={r1,r2,r3}
S={s1,s2,s3}

last sweep line position in R: r3
last sweep line position in S: s2

s2<r3:
scan R from r3 until $r_i > s_2$ on x-axis
for each seen r_i check y-intersection:
no pair found!

increase position in S to s3



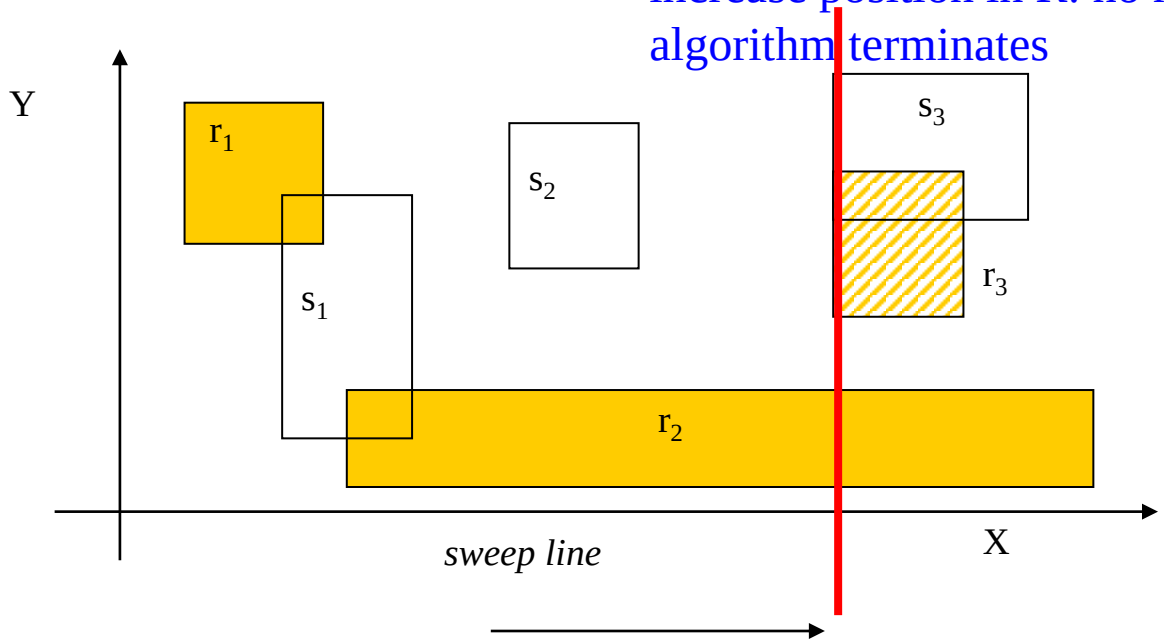
Sweep example

sorted lists:
R={r1,r2,r3}
S={s1,s2,s3}

last sweep line position in R: r3
last sweep line position in S: s3

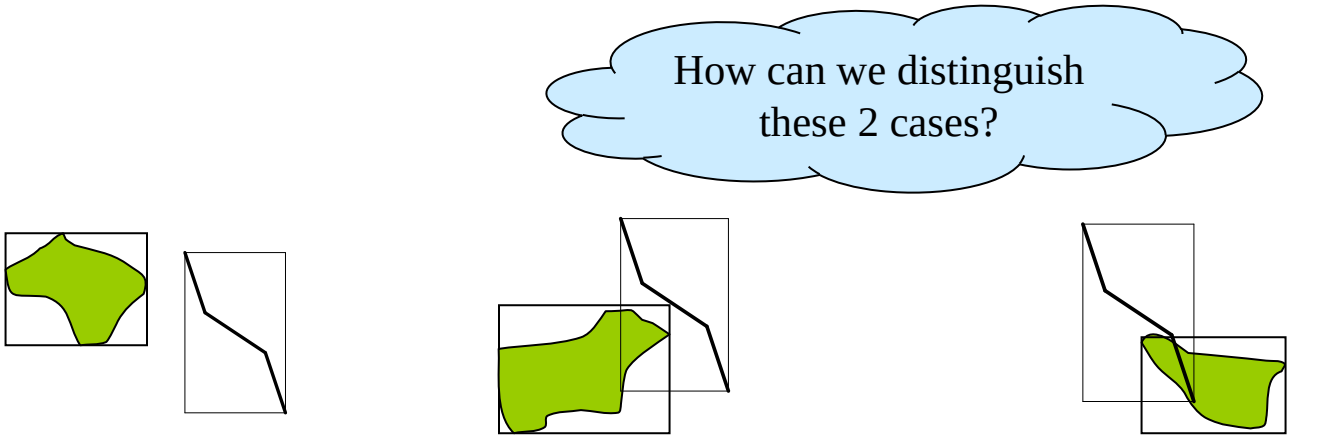
r3<s3:
scan S from s3 until $s_i > r_3$ on x-axis
for each seen s_i check y-intersection:
pair <r3,s3> found!

increase position in R: no more rectangles:
algorithm terminates



The refinement step

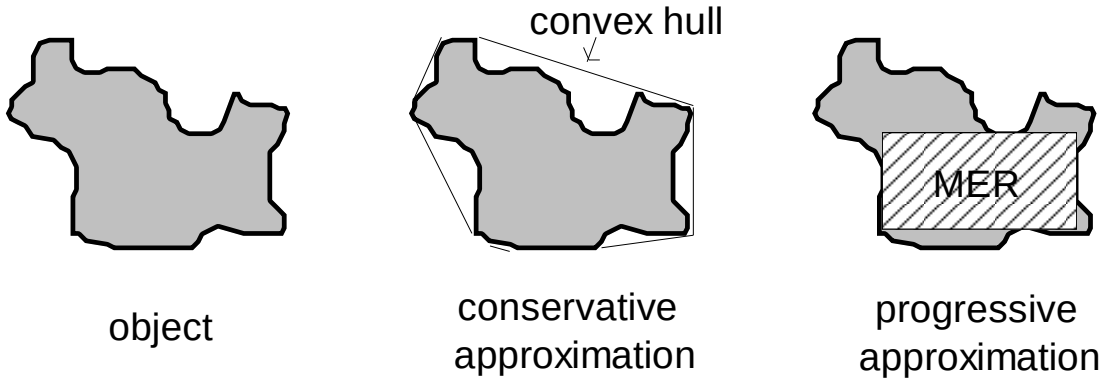
- Most join algorithms focus on the filter step; how do we process the refinement step?
- A multi-step process is adopted for spatial joins



Multi-step join processing

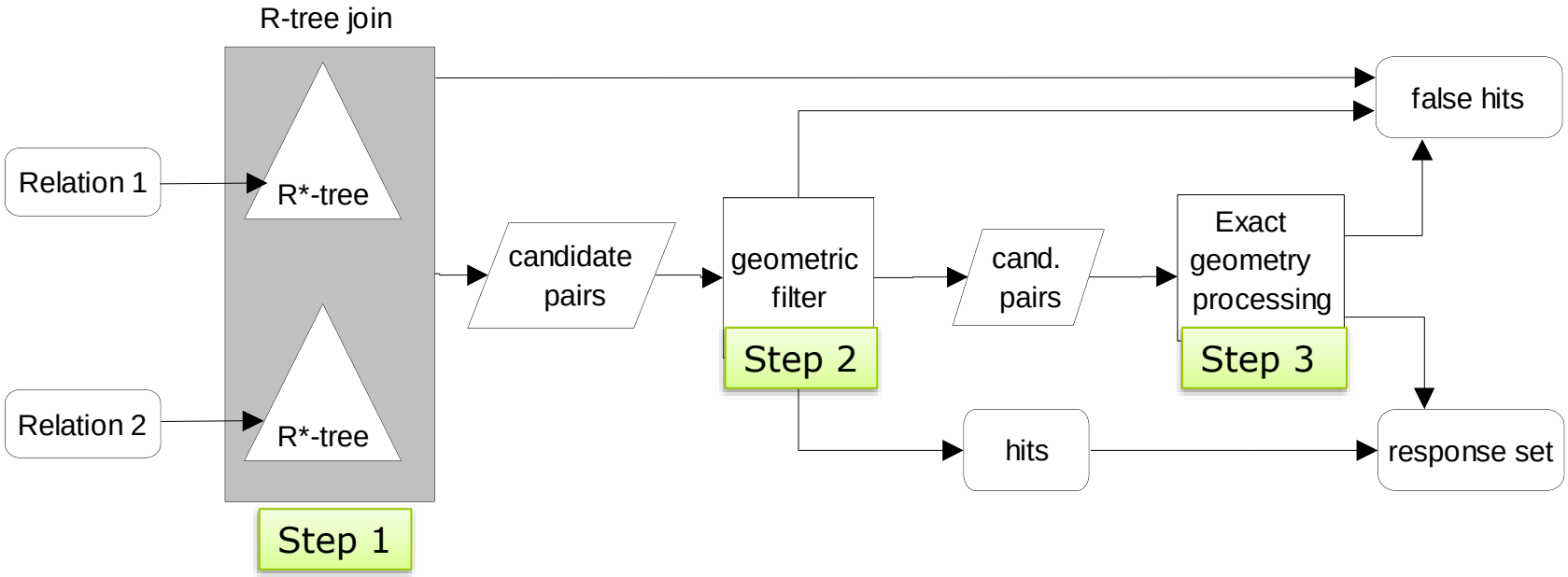
- Step 1: find MBR pairs that intersect
- Step 2: compare some more detailed approximations to make conclusions

  (a.k.a. geometric filter)
  - conservative approximations
    - e.g., convex hull
  - progressive approximation
    - e.g., maximum enclosed rectangle



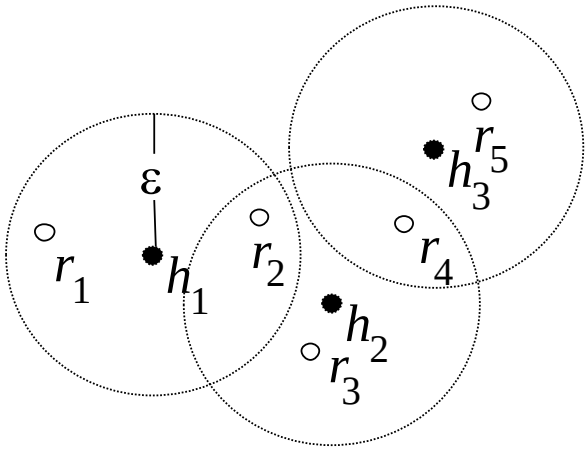
Multi-step join processing

- Step 3: if still join predicate inconclusive, perform expensive refinement step
  - can be processed by computational geometry algorithms
- Multi-step processing (R-tree join as example):



Other Spatial Joins

- Also:
  - Semi-join: Find the cities that intersect a river
  - Similarity join: Find pairs of hotels, restaurants close to each other (with distance smaller than 100m)
  - Closest pairs: Find the closest pair of hotels, restaurants
  - All-NN: For each hotel find the nearest restaurant
  - Iceberg distance join: Find hotels close to at least 10 restaurants



Summary

- Topological, distance and directional spatial relationships are used as predicates in spatial queries
- The dimensionality and extent of spatial data makes their indexing challenging
- The R-tree is the most popular spatial access method
- Popular algorithms for Nearest Neighbor and Spatial Join queries are based on the R-tree