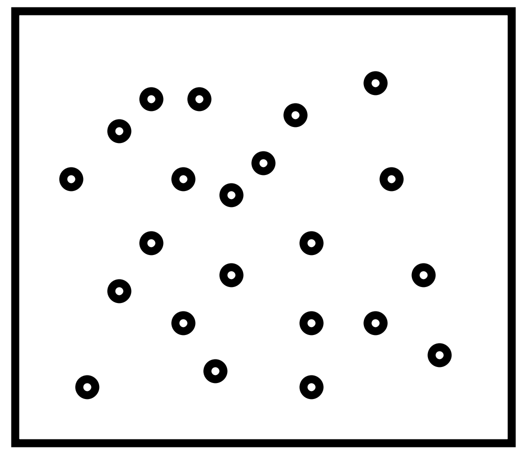
Spatial Queries

R-tree: variations

- What about static datasets?
  - (no ins/del) Hilbert
- What about other bounding shapes?

R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?

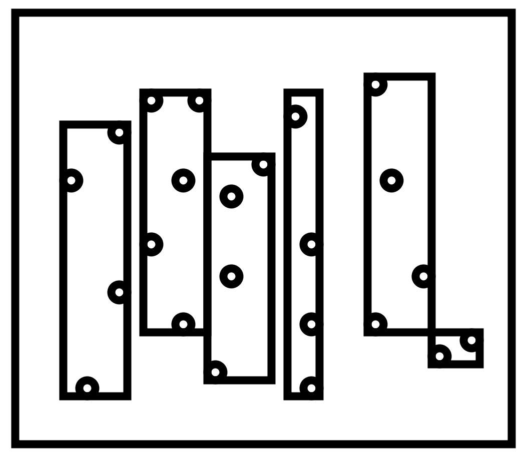


R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep

  great for queries on ‘x’;

  terrible for ‘y’

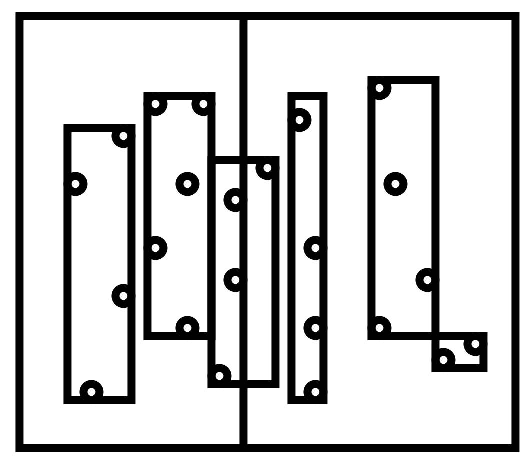


R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep

  great for queries on ‘x’;

  bad for ‘y’

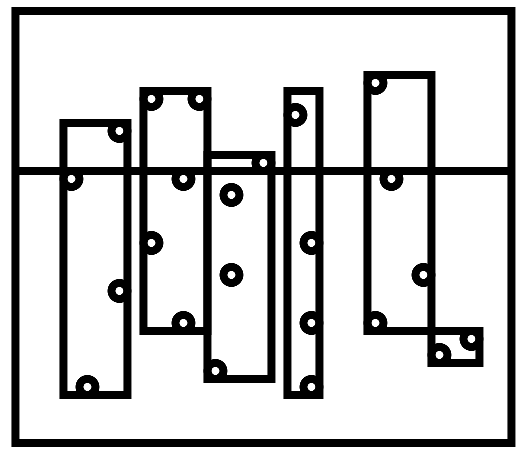


R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep

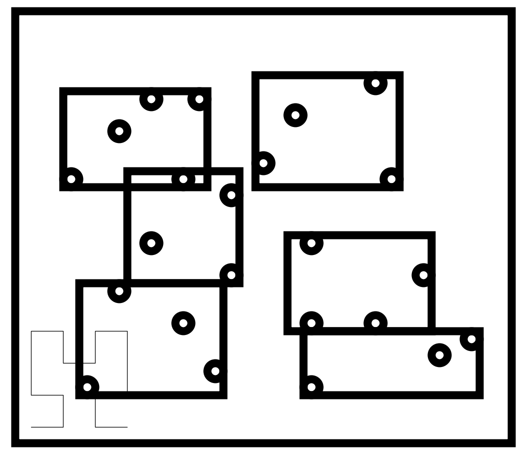
  great for queries on ‘x’;

  terrible for ‘y’
- Q: how to improve?



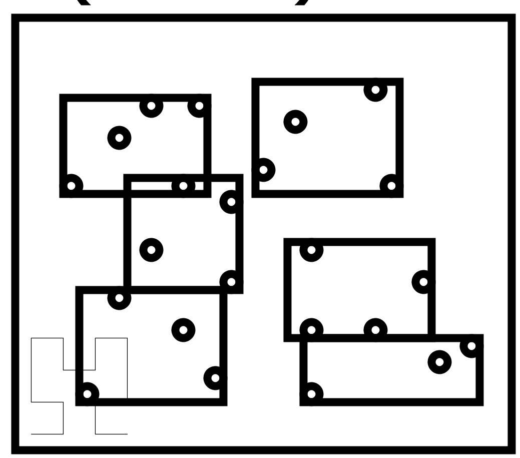
R-trees - variations

- A: plane-sweep on HILBERT curve!



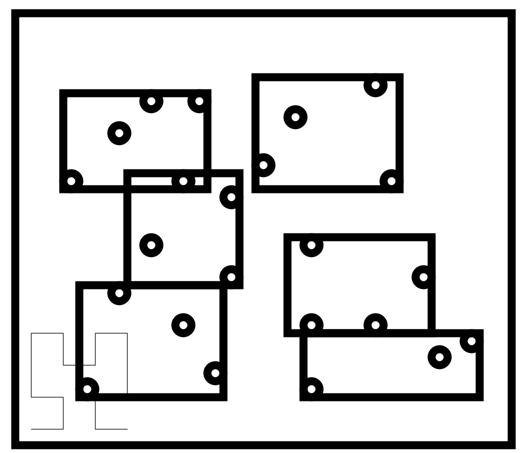
R-trees - variations

- A: plane-sweep on HILBERT curve!
- In fact, it can be made dynamic (how?), as well as to handle regions (how?)



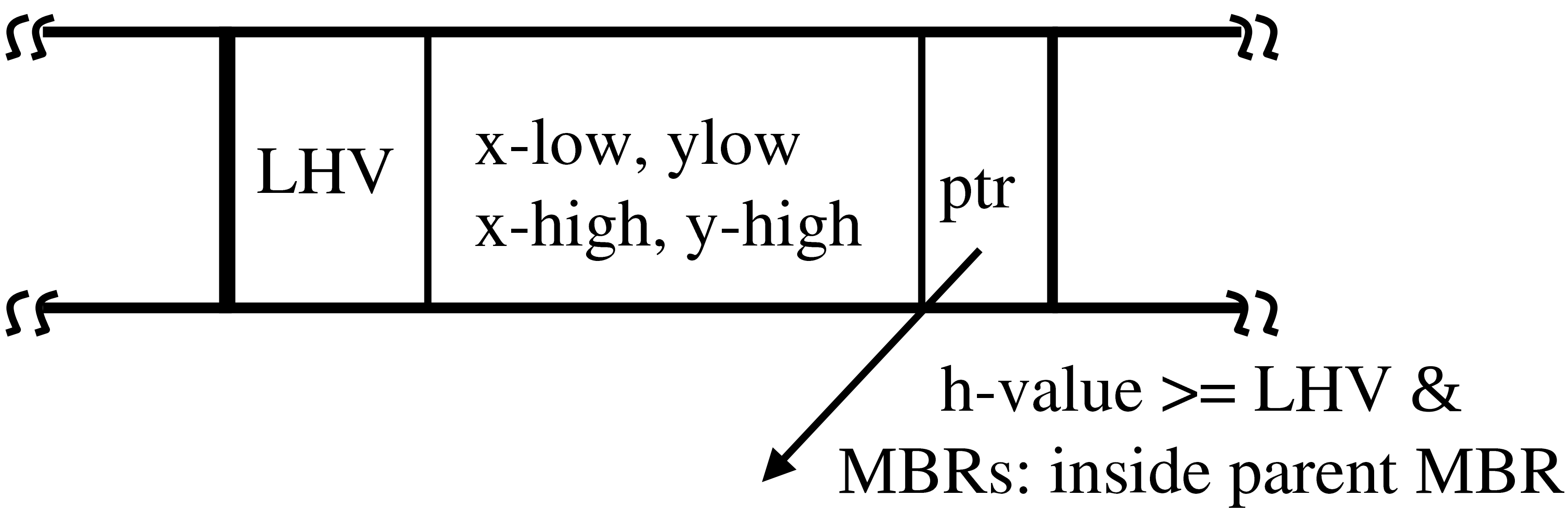
R-trees - variations

- Dynamic ('Hilbert R-tree):
  - each point has an 'h'-value (hilbert value)
  - insertions: like a B-tree on the h-value
  - but also store MBR, for searches



Hilbert R-tree

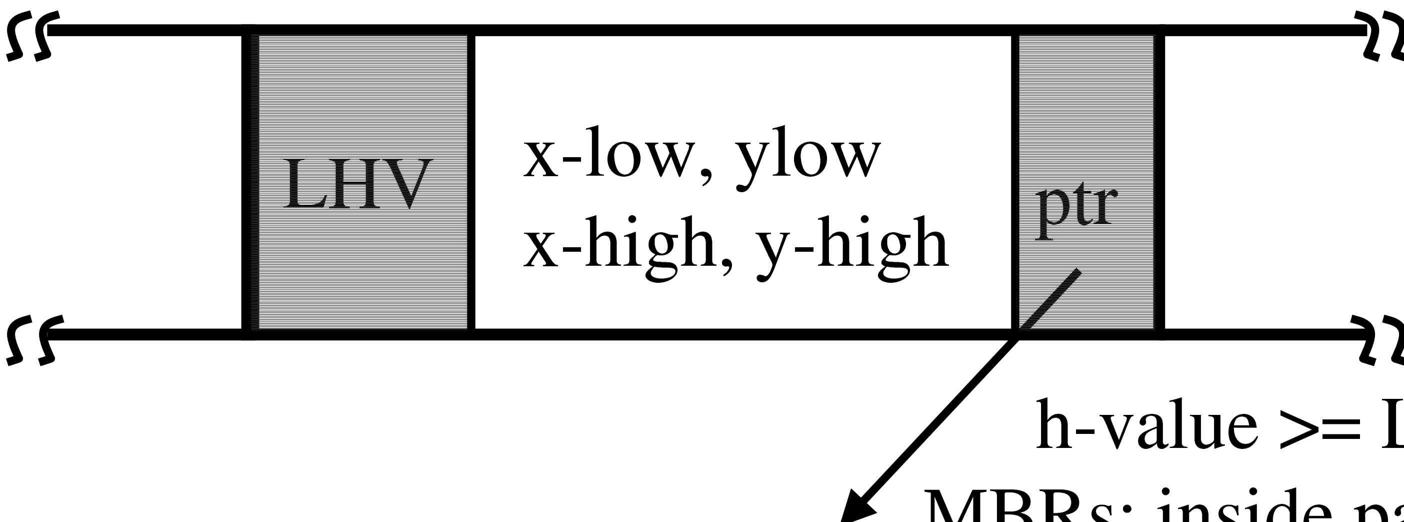
- Data structure of a node?



R-trees - variations

- Data structure of a node?

~B-tree

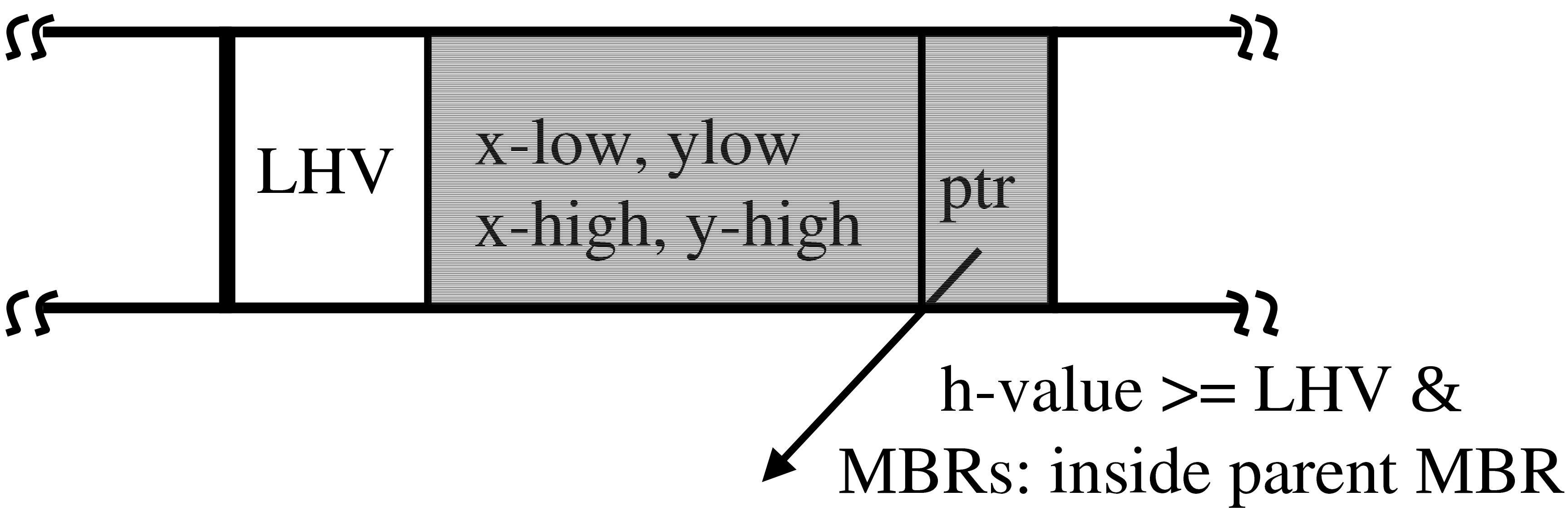


h-value >= LHV &
MBRs: inside parent MBR

R-trees - variations

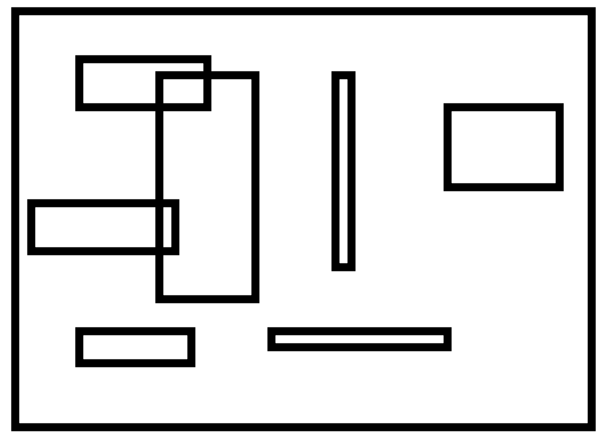
- Data structure of a node?

~ R-tree



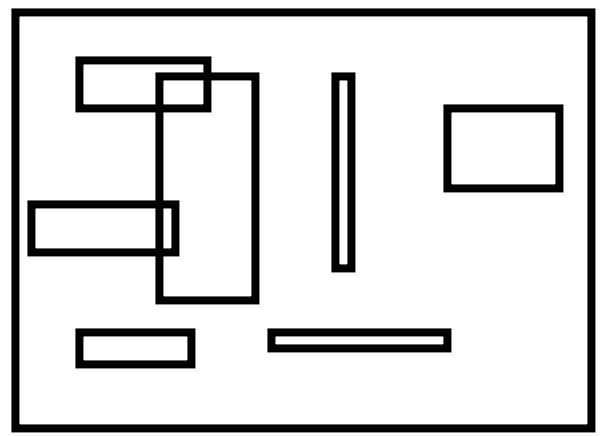
R-trees - variations

- What if we have regions, instead of points?
- I.e., how to impose a linear ordering (‘h-value’) on rectangles?



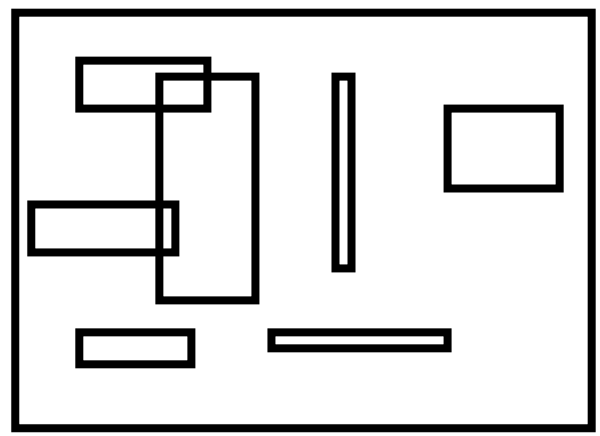
R-trees - variations

- What if we have regions, instead of points?
- I.e., how to impose a linear ordering ('h-value') on rectangles?
- A1: h-value of center
- A2: h-value of 4-d point
  (center, x-radius,
  y-radius)
- A3: ...



R-trees - variations

- What if we have regions, instead of points?
- I.e., how to impose a linear ordering (‘h-value’) on rectangles?
- **<u>A1: h-value of center</u>**
- A2: h-value of 4-d point (center, x-radius, y-radius)
- A3: ...



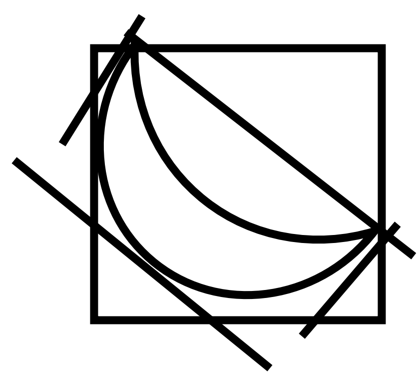
R-trees - variations

- with h-values, we can have deferred splits, 2-to-3 splits (3-to-4, etc)
  - Instead of splitting a full node, find the siblings (using the h-values) and redistribute the rectangles among the nodes. Split only when **all** siblings are full.
- experimentally: faster than R*-trees

(reference: [Kamel Faloutsos vldb 94])

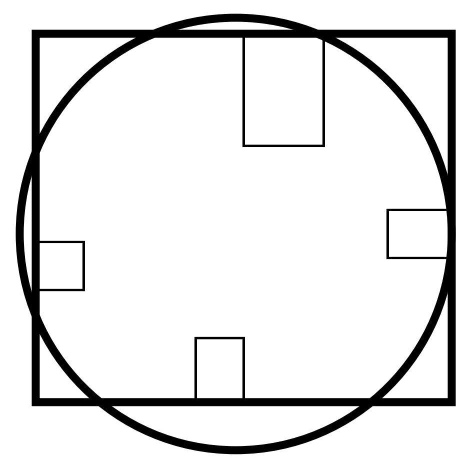
R-trees - variations

- what about other bounding shapes? (and why?)
- A1: arbitrary-orientation lines (cell-tree, [Guenther]
- A2: P-trees (polygon trees) (MB polygon: 0, 90, 45, 135 degree lines)



R-trees - variations

- A3: L-shapes; holes (hB-tree)
- A4: TV-trees [Lin+, VLDB-Journal 1994]
- A5: SR-trees [Katayama+, SIGMOD97] (used in Informedia)

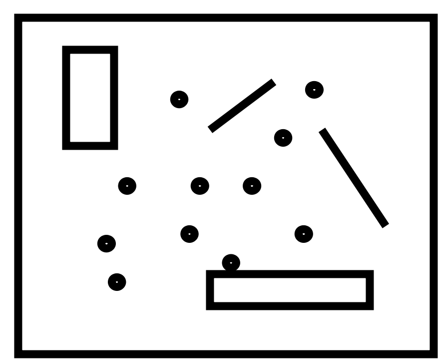


R-trees - conclusions

- Popular method; like multi-d B-trees
- guaranteed utilization
- good search times (for low-dim. at least)
- Informix ships DataBlade with R-trees

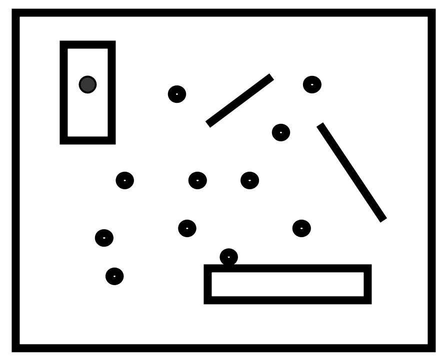
Spatial Queries

- 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)



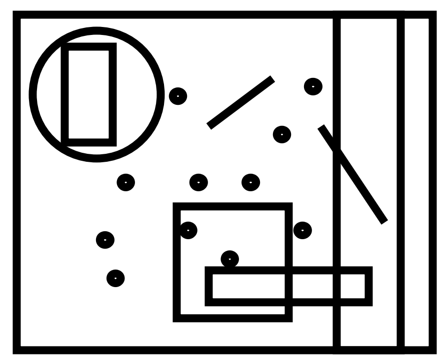
Spatial Queries

- 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)



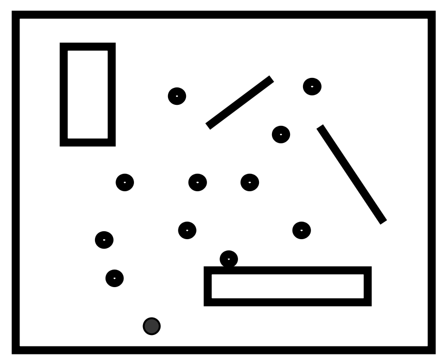
Spatial Queries

- 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)



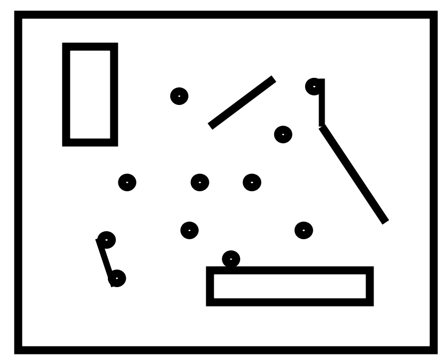
Spatial Queries

- 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)



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)

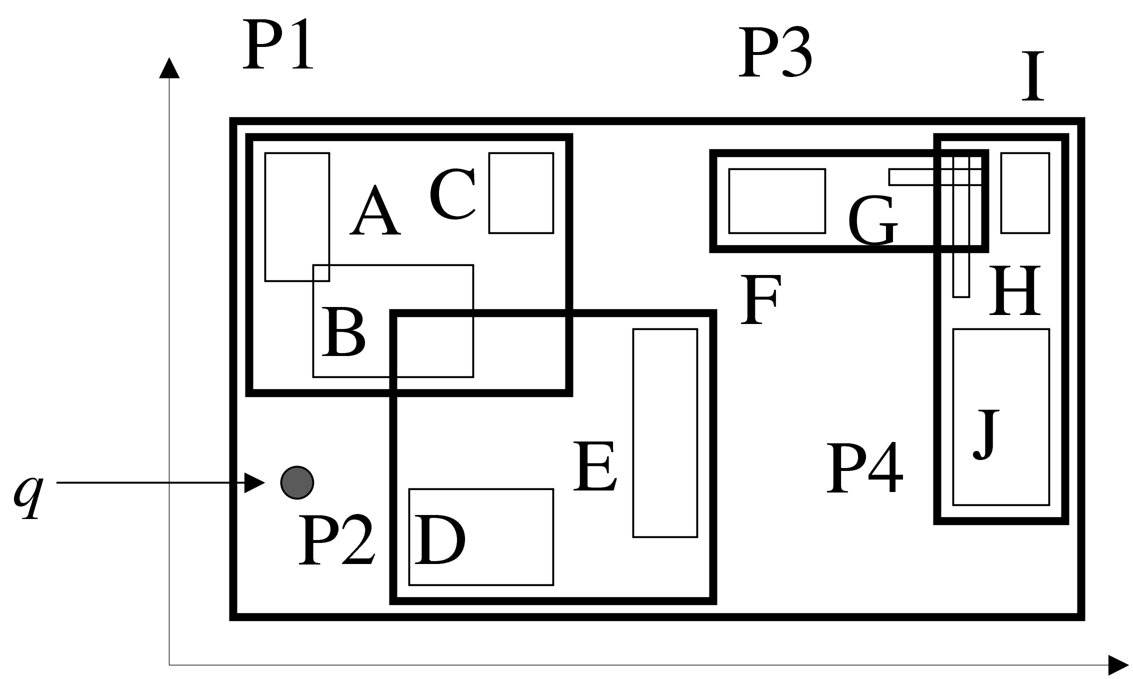


R-trees - Range search

pseudocode:

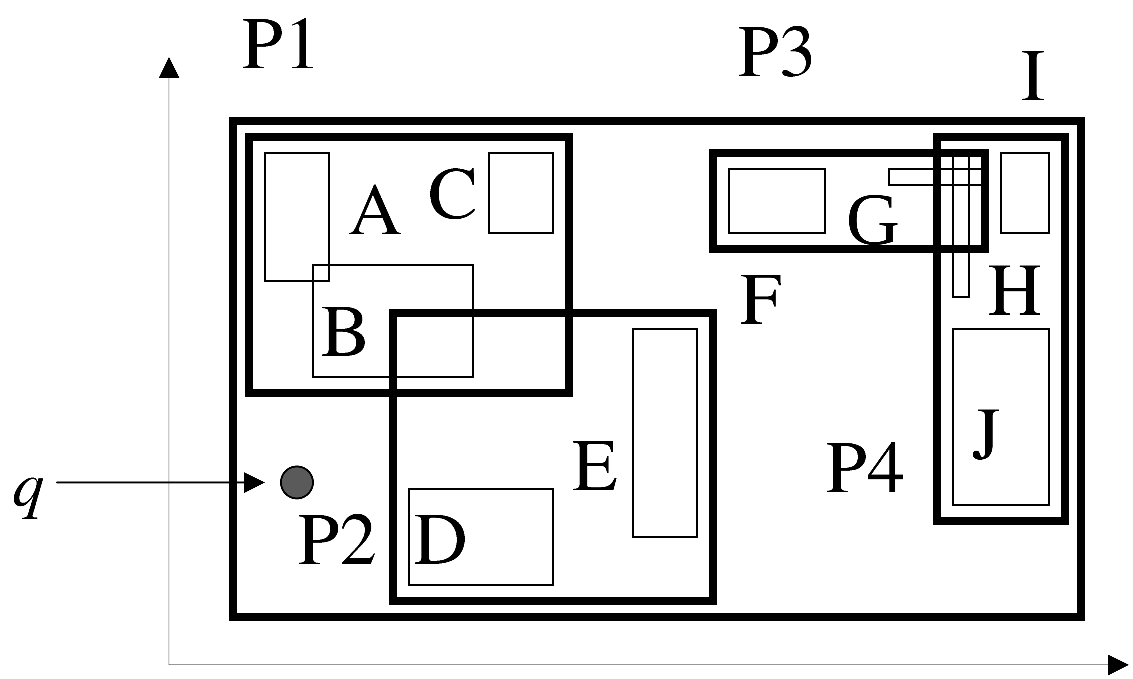
```
check the root
 for each branch,
    if its MBR intersects the query rectangle
          apply range-search (or print out, if this
                  is a leaf)
```

R-trees - NN search



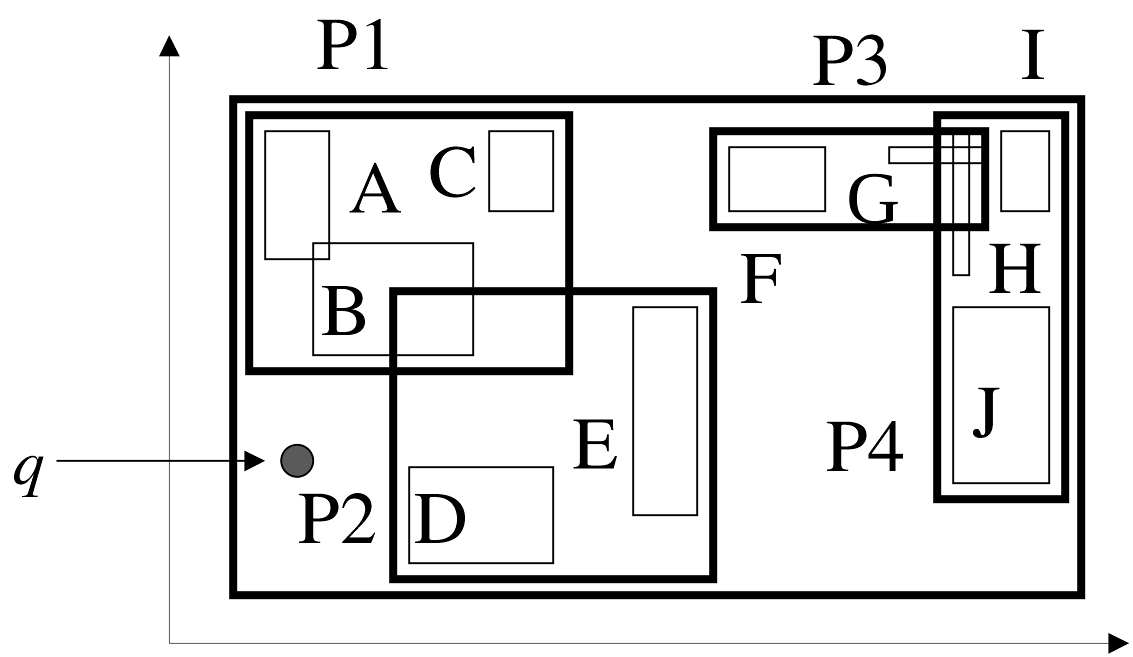
R-trees - NN search

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



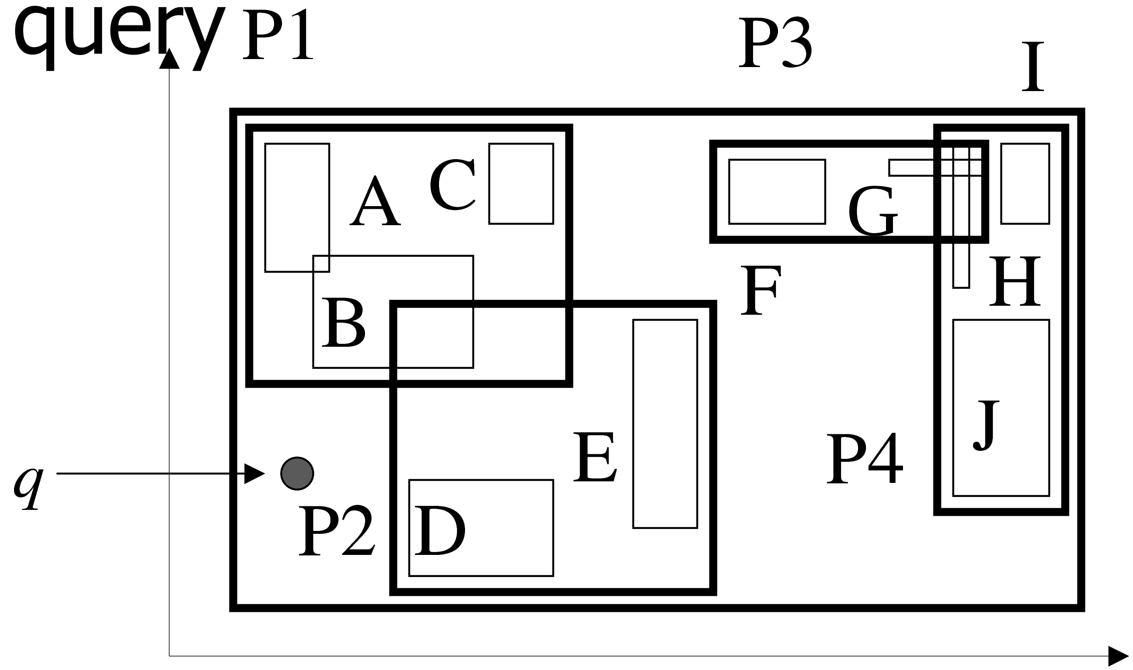
R-trees - NN search

- A1: depth-first search; then, range query



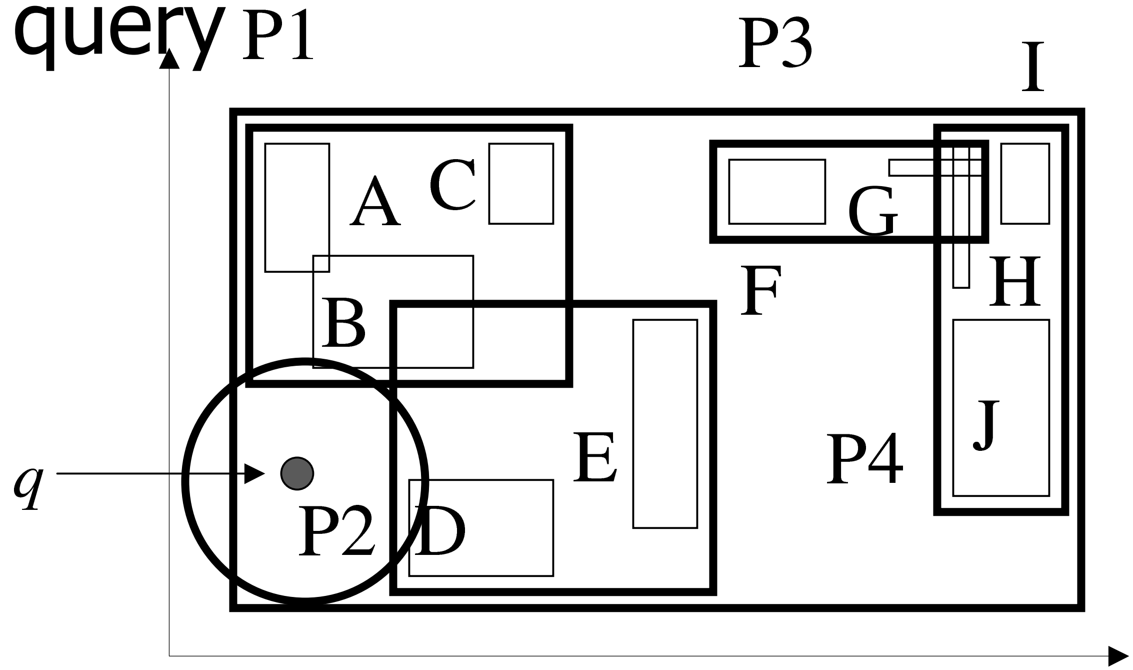
R-trees - NN search

- A1: depth-first search; then, range query



R-trees - NN search

- A1: depth-first search; then, range query

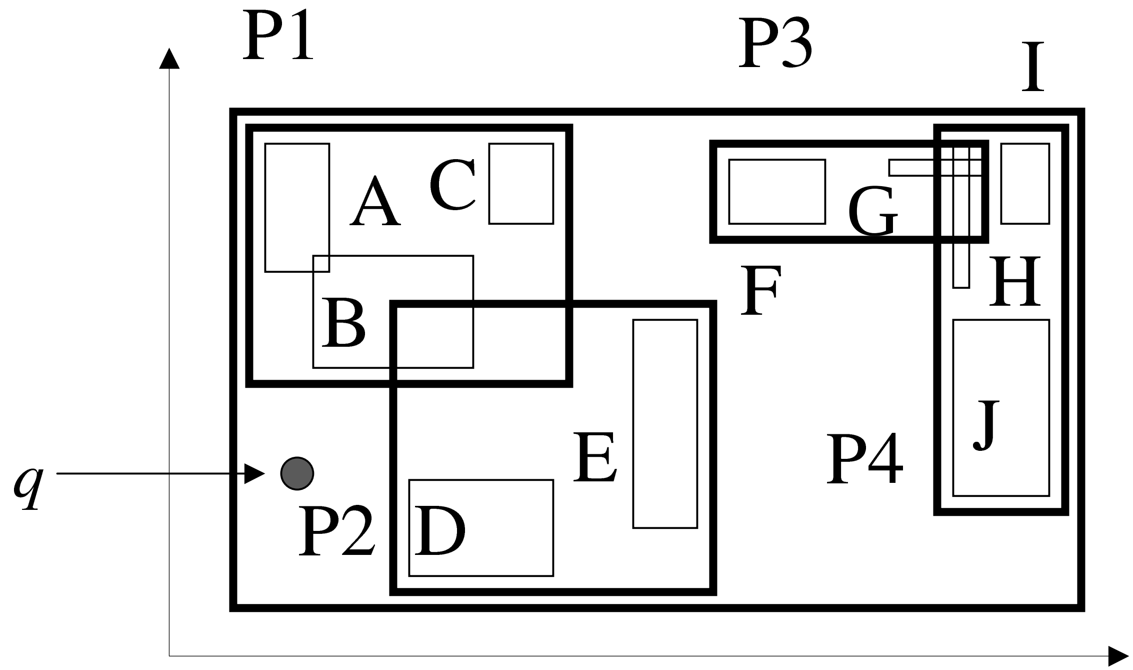


R-trees - NN search

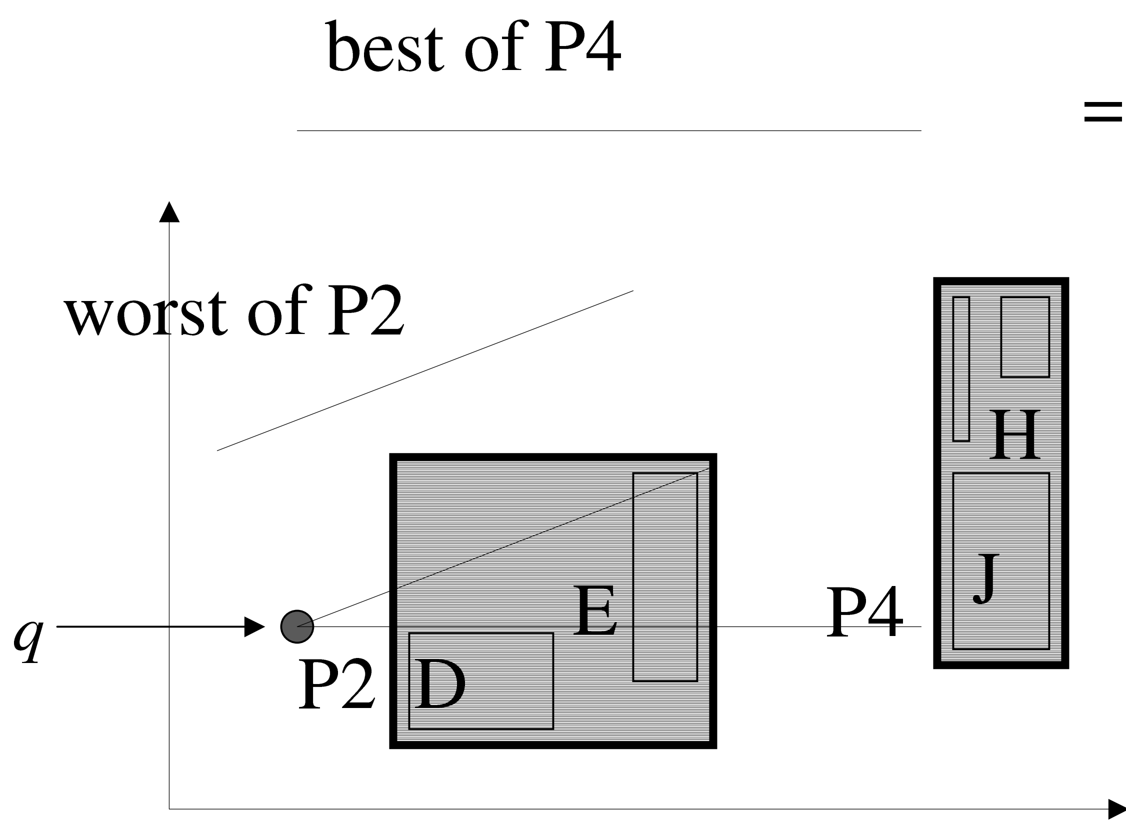
- A2: [Roussopoulos+, sigmod95]:
  - priority queue, with promising MBRs, and their best and worst-case distance
- main idea: Every face of any MBR contains at least one point of an actual spatial object!

R-trees - NN search

consider only P2 and P4, for illustration



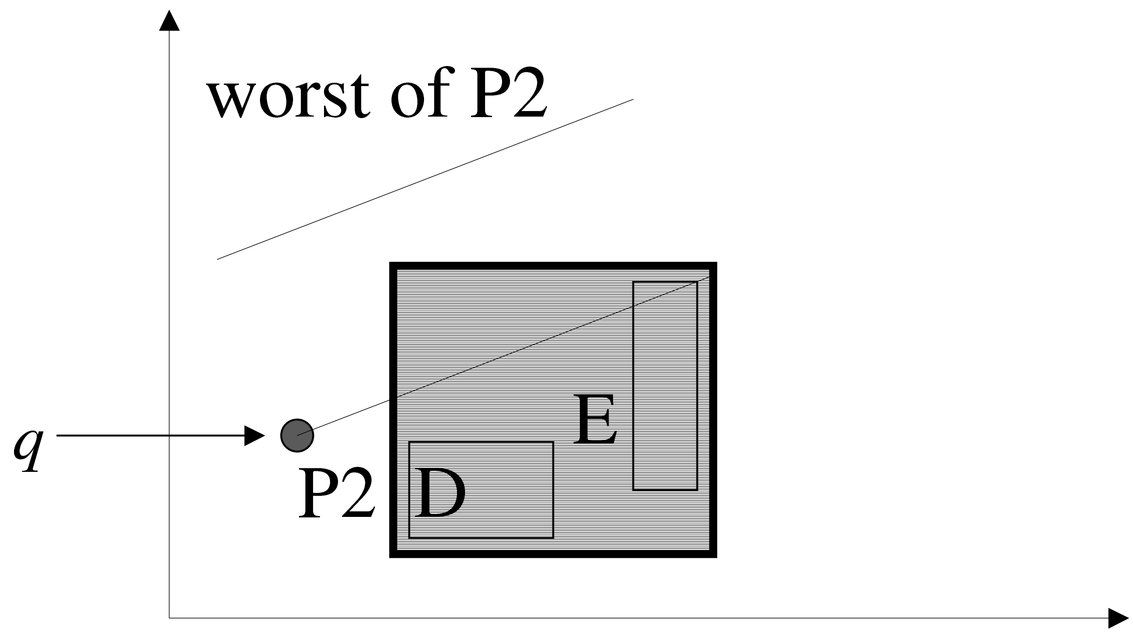
R-trees - NN search



=> P4 is useless for 1-nn

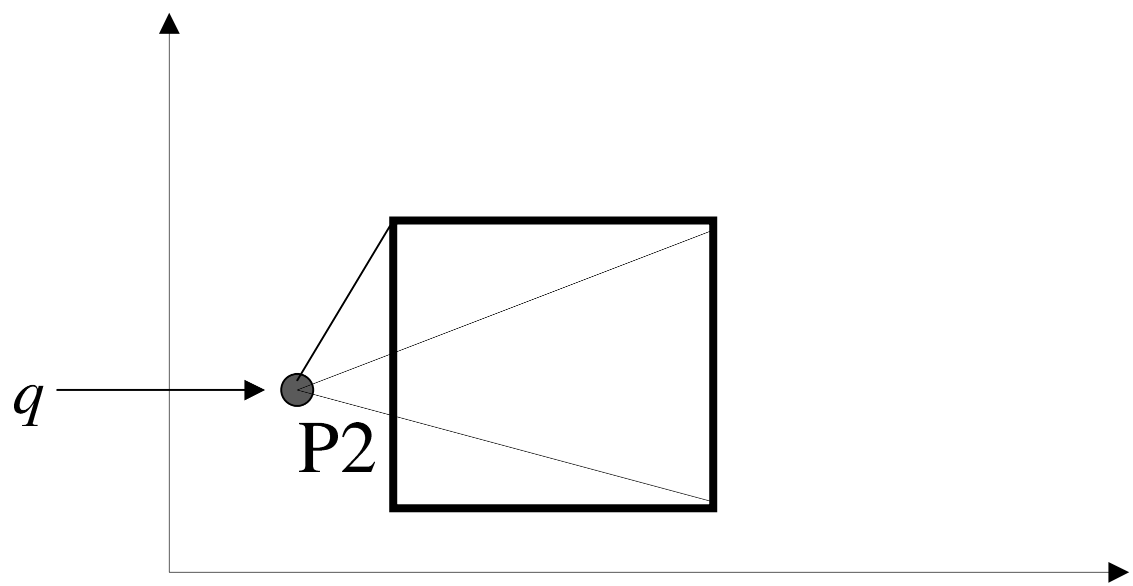
R-trees - NN search

- what is really the worst of, say, P2?



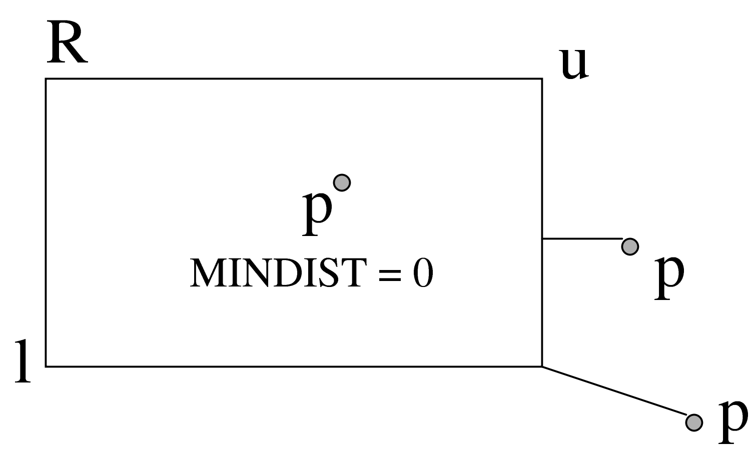
R-trees - NN search

- what is really the worst of, say, P2?
- A: the smallest of the two red segments!



Nearest-neighbor searching

- Branch and bound strategy
- Compute MINDIST and MINMAXDIST [RKV95]
- 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

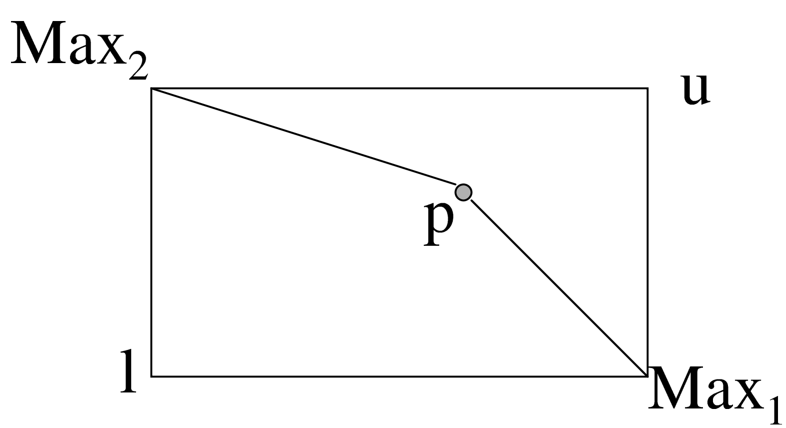


Sqrt sum $(p_i - r_i)^2$ where

$$r_i = \begin{cases} l_i & \text{if } p_i < l_i \\ u_i & \text{if } p_i > u_i \\ p_i & \text{otherwise} \end{cases}$$

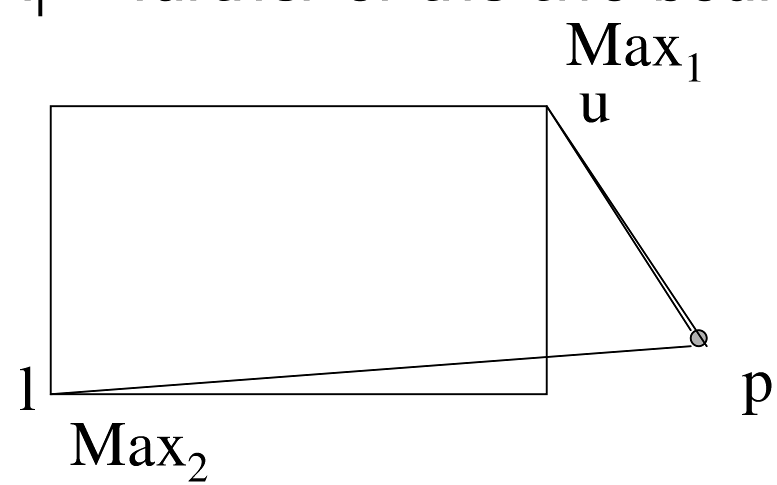
Nearest-neighbor searching

- MINMAXDIST(p,R) is the minimum of the maximum distance to each pair of faces of R
  - MaxDistanceToFace(p,R,k) = distance between p and $Max_k = (M_1, M_2, .., M_{k-1} m_k, M_{K+1}, .., M_n)$
  - m_i = closer of the two boundary points along i axis
  - M_i = farther of the two boundary points along i axis



Nearest-neighbor searching

- MINMAXDIST(p,R) is the minimum of the maximum distance to each pair of faces of R
  - MaxDistanceToFace(p,R,k) = distance between p and $Max_k = (M_1, M_2, .., M_{k-1} m_k, M_{K+1}, .., M_n)$
  - m_i = closer of the two boundary points along i axis
  - M_i = farther of the two boundary points along i axis



Pruning

- ESTIMATE := smallest MINMAXDIST(p,R)
- Prune an MBR R′ for which MINDIST(p,R′) is greater than ESTIMATE.
- Generalize to k-nearest neighbor searching
  - Maintain kth largest MINMAXDIST
  - Prune an MBR if MINDIST to it is larger than the current estimate of kth MINMAXDIST
- Can use objects to refine estimate

Order of searching

- Depth first order
  - Inspect children in MINDIST order
  - For each node in the tree keep a list of nodes to be visited
  - Prune some of these nodes in the list
  - Continue until the lists are empty

Another NN search

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

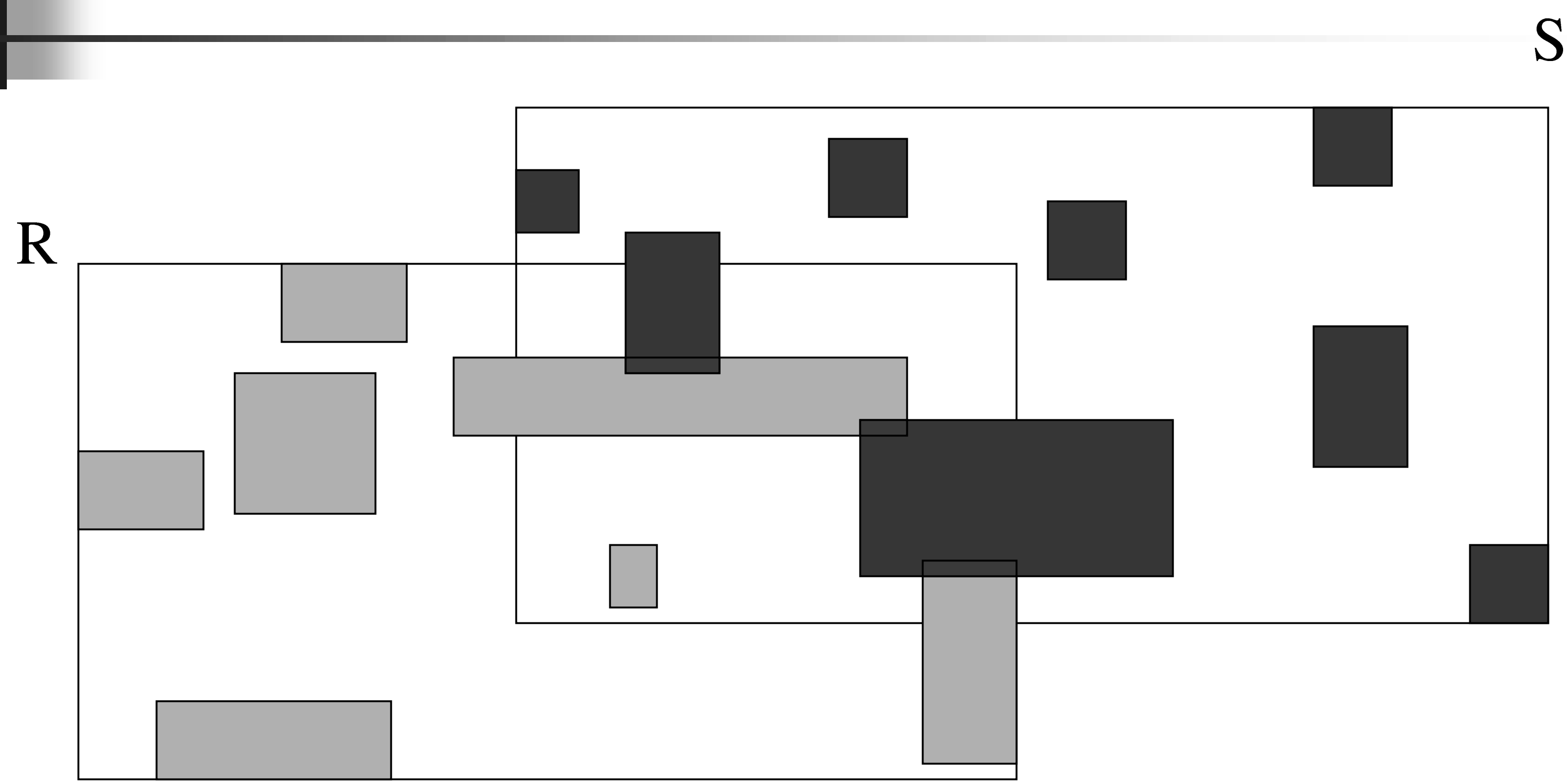
Spatial Join

- Find all parks in a city
- Find all trails that go through a forest
- Basic operation
  - find all pairs of objects that overlap
- Single-scan queries
  - nearest neighbor queries, range queries
- Multiple-scan queries
  - spatial join

Algorithms

- No existing index structures
  - Transform data into 1-d space [O89]
    - z-transform; sensitive to size of pixel
  - Partition-based spatial-merge join [PW96]
    - partition into tiles that can fit into memory
    - plane sweep algorithm on tiles
  - Spatial hash joins [LR96, KS97]
  - Sort data [BBKK01]
- With index structures [BKS93, HJR97]
  - k-d trees and grid files
  - R-trees

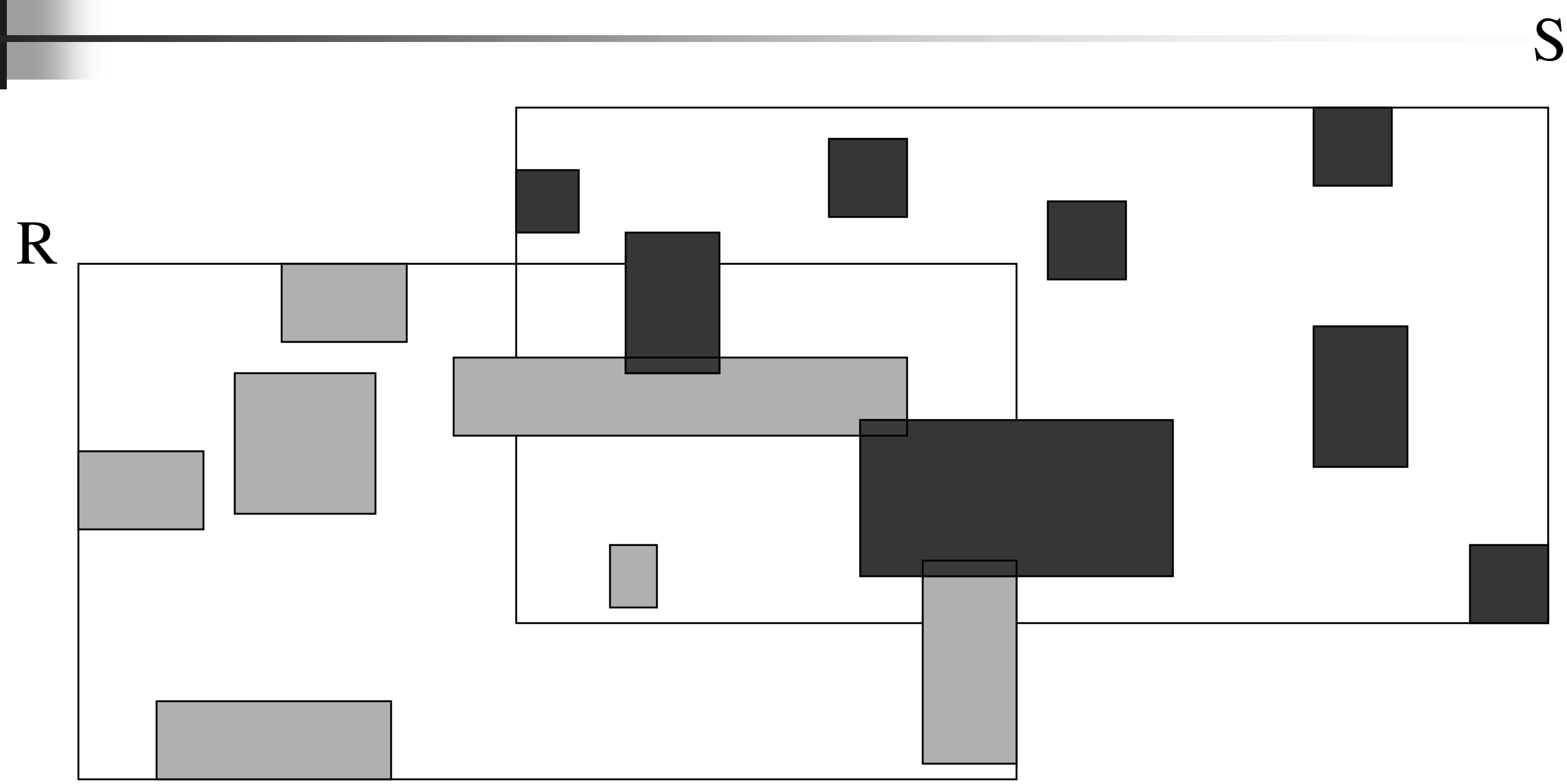
R-tree based Join [BKS93]



Join1(R,S)

- Repeat
  - Find a pair of intersecting entries E in R and F in S
  - If R and S are leaf pages then add (E,F) to result-set
  - Else Join1(E,F)
- Until all pairs are examined
- CPU and I/O bottleneck

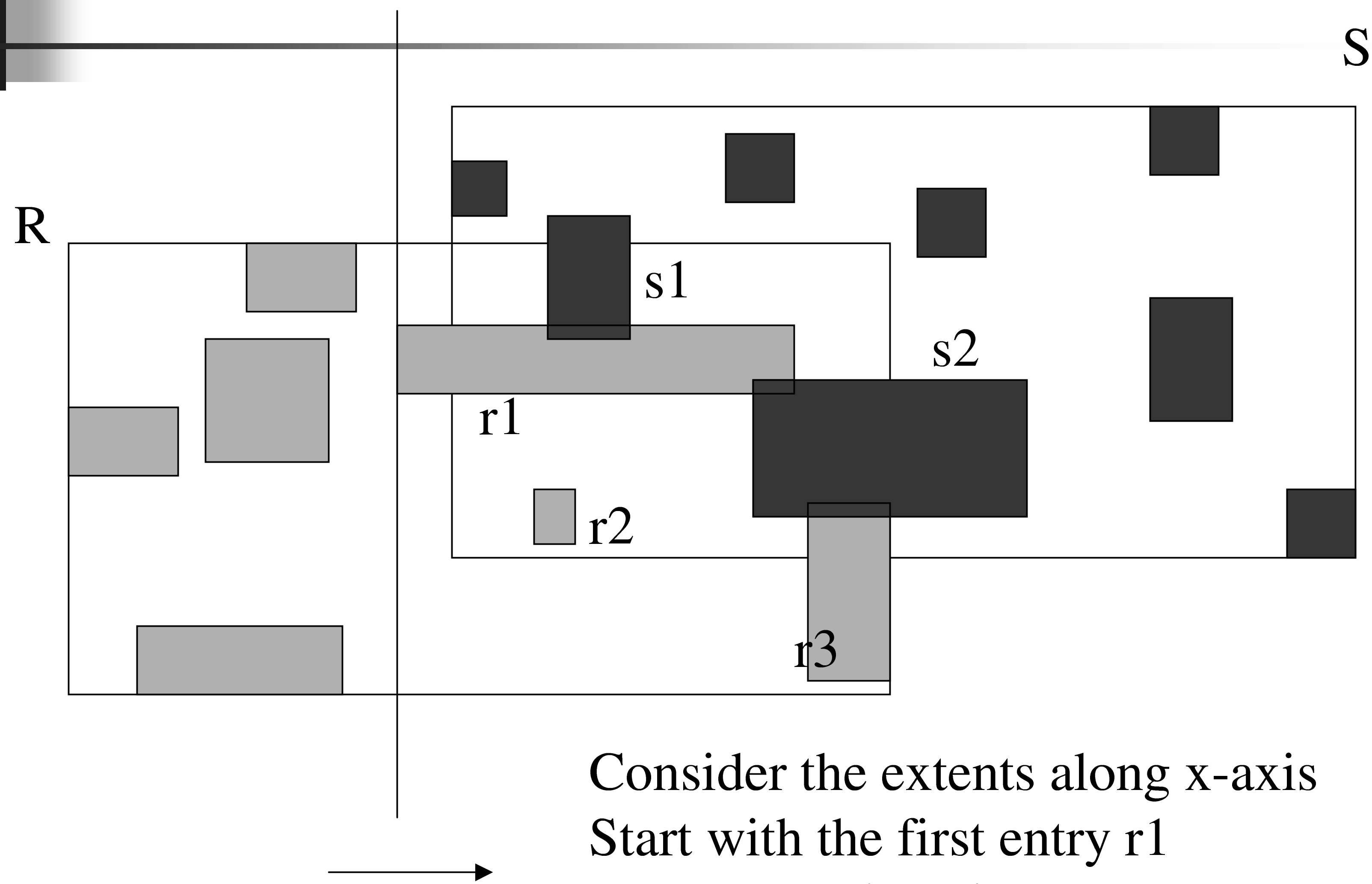
Reducing CPU bottleneck



Join2(R,S,IntersectedVol)

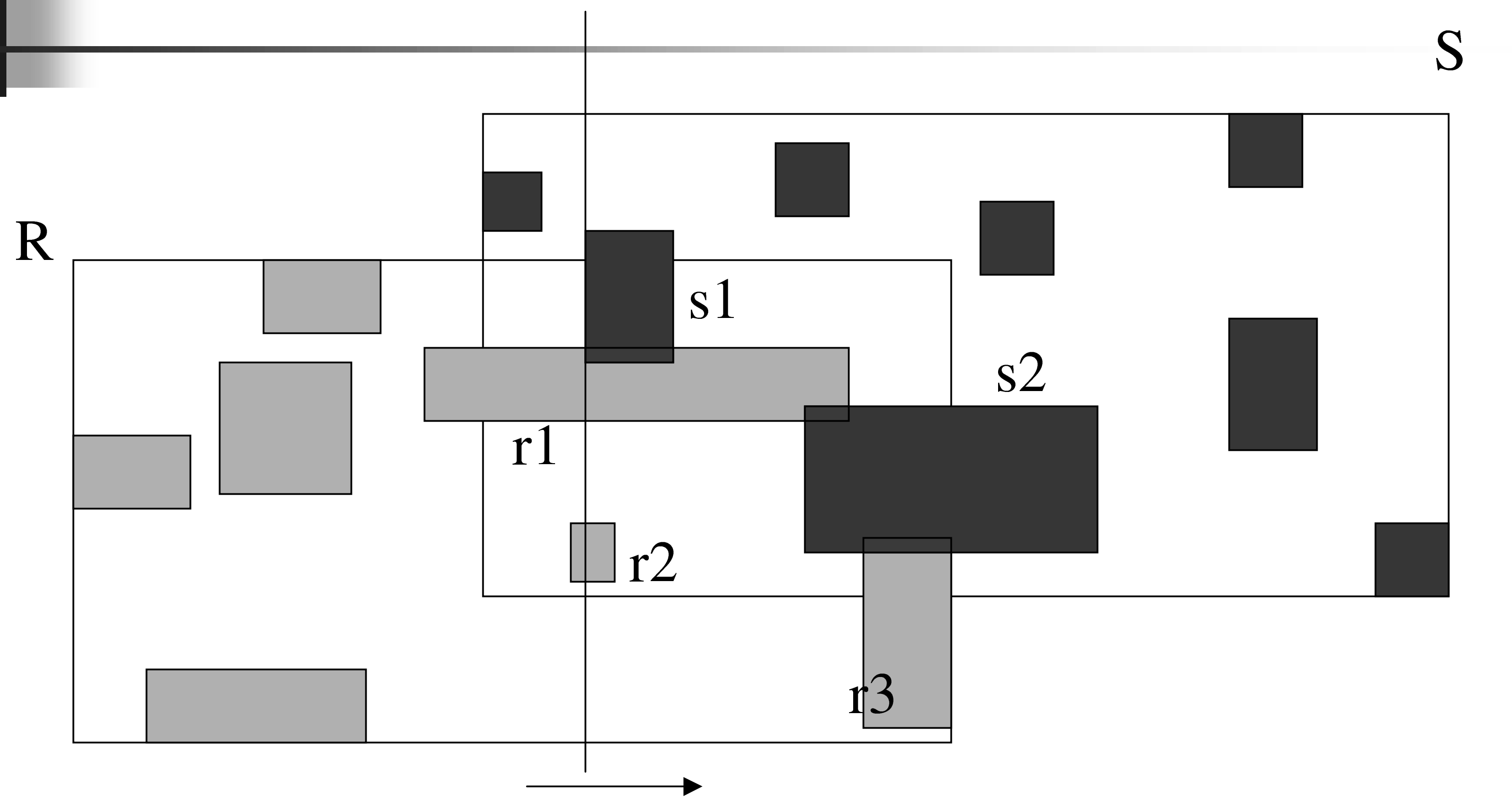
- Repeat
  - Find a pair of intersecting entries E in R and F in S that overlap with IntersectedVol
  - If R and S are leaf pages then add (E,F) to result-set
  - Else Join2(E,F,CommonEF)
- Until all pairs are examined
- 14+6 comparisons instead of 49
- In general, number of comparisons equals
  - size(R) + size(S) + relevant(R)*relevant(S)
- Reduce the product term

Using Plane Sweep



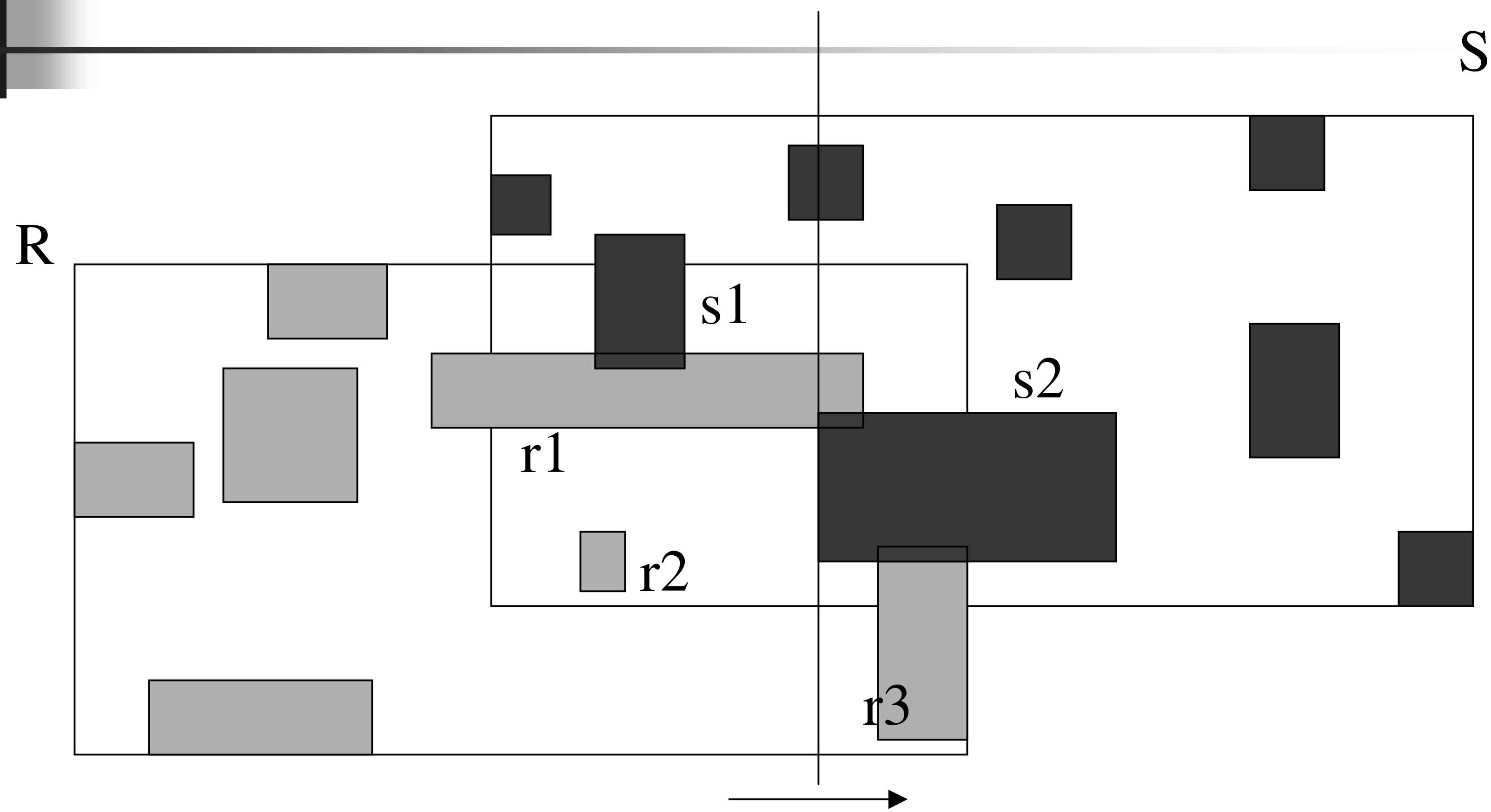
Consider the extents along x-axis
Start with the first entry r1
sweep a vertical line

Using Plane Sweep



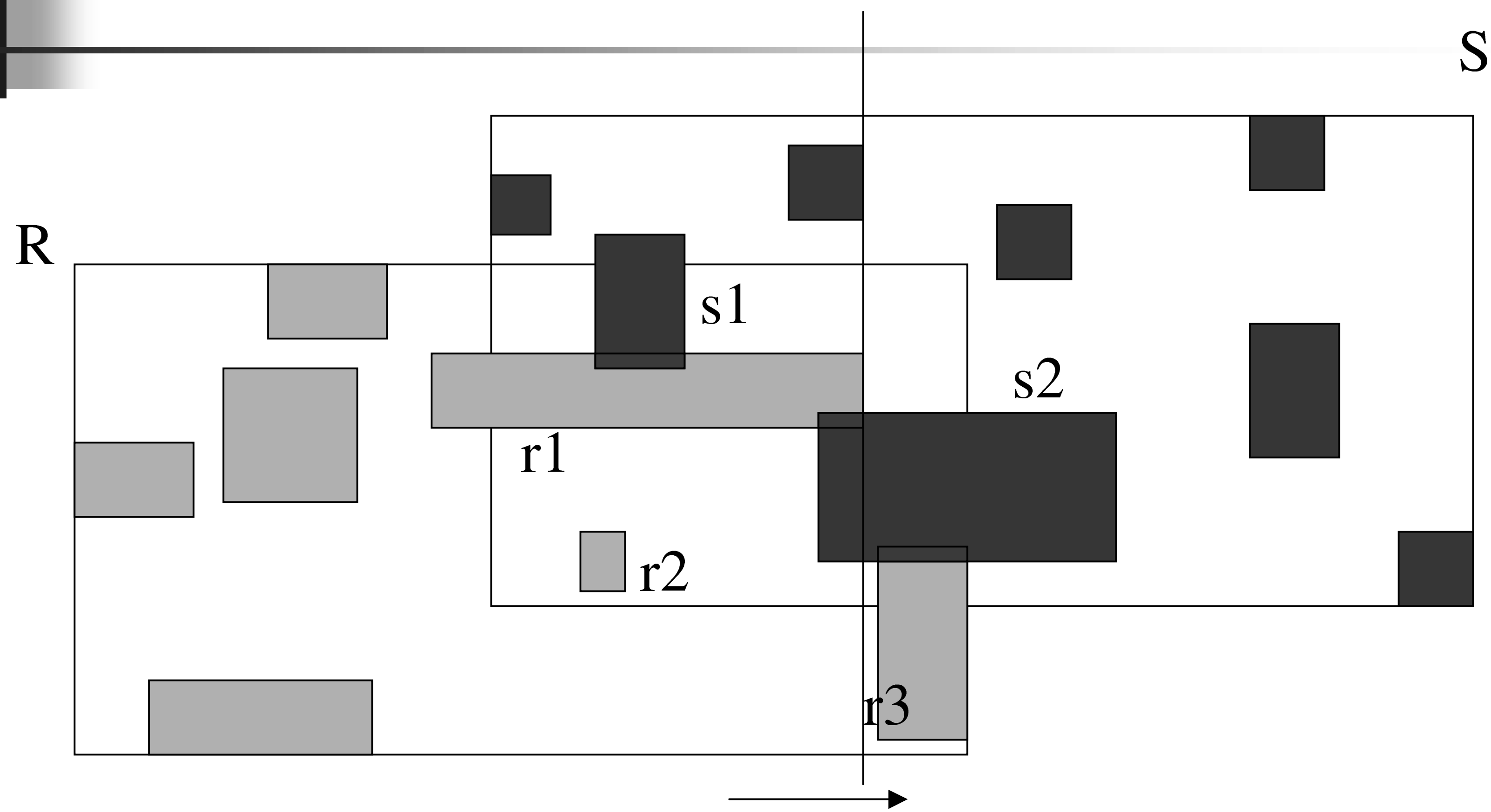
Check if (r1,s1) intersect along y-dimension
Add (r1,s1) to result set

Using Plane Sweep



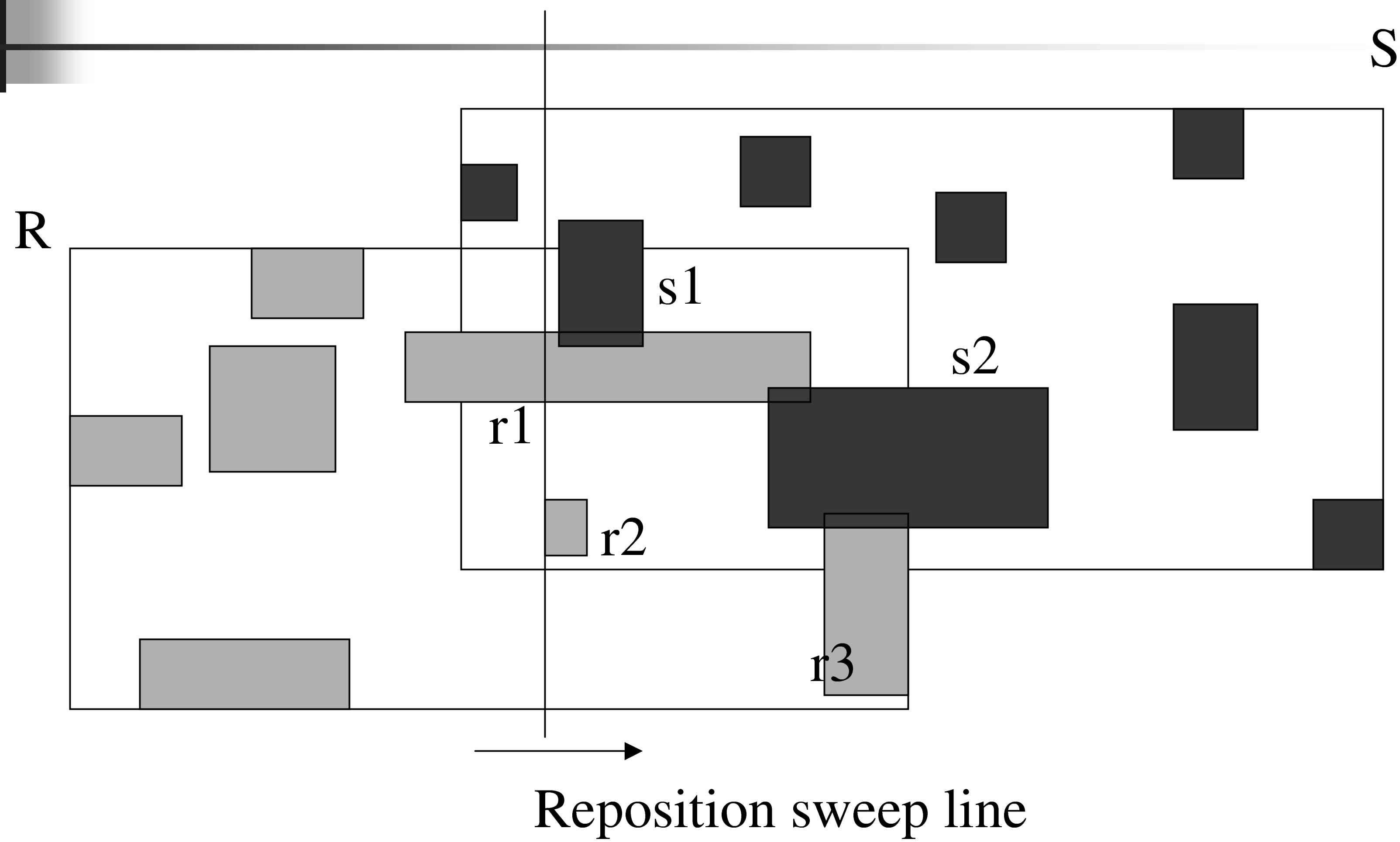
Check if (r1,s2) intersect along y-dimension
Add (r1,s2) to result set

Using Plane Sweep

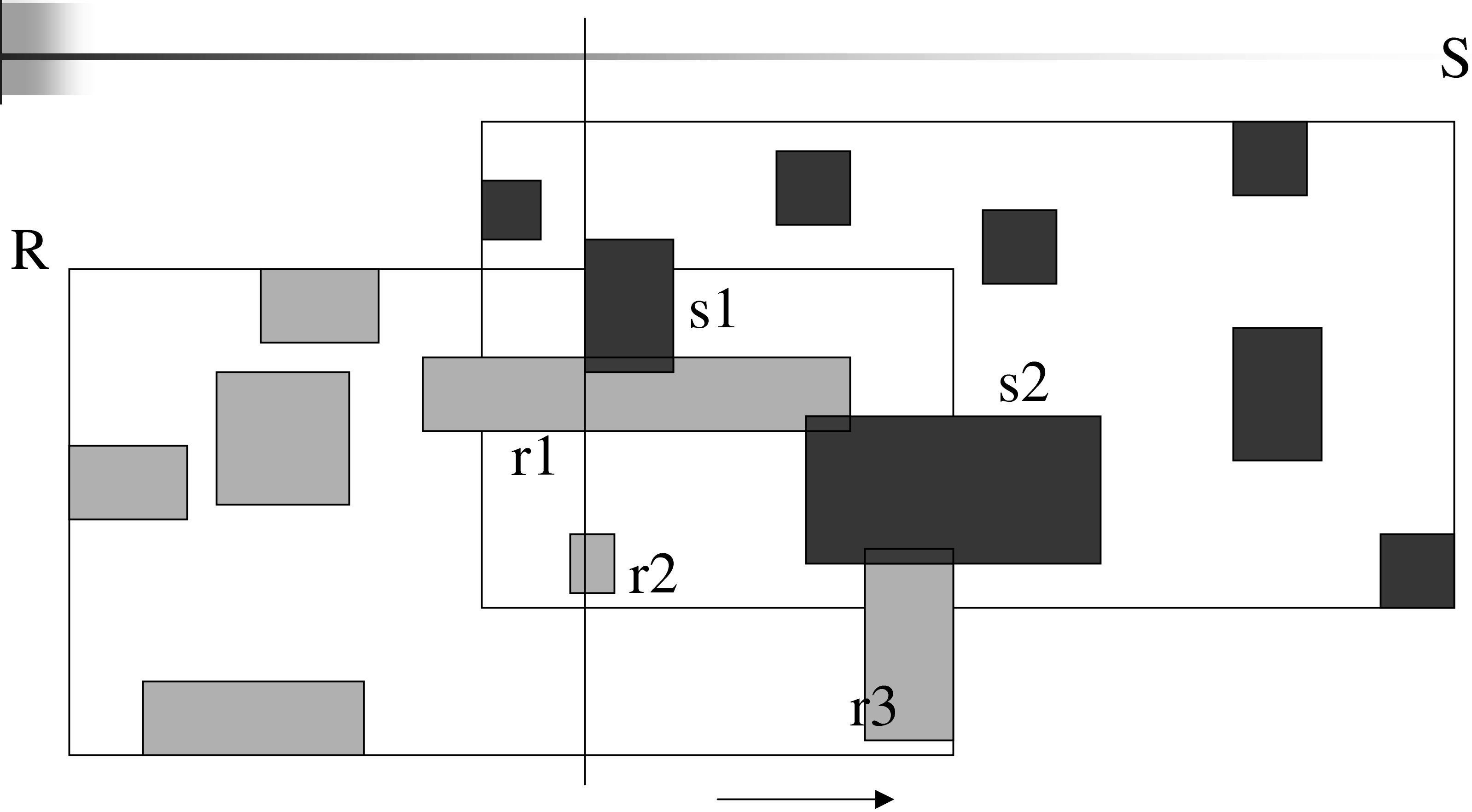


Reached the end of r1
Start with next entry r2

Using Plane Sweep

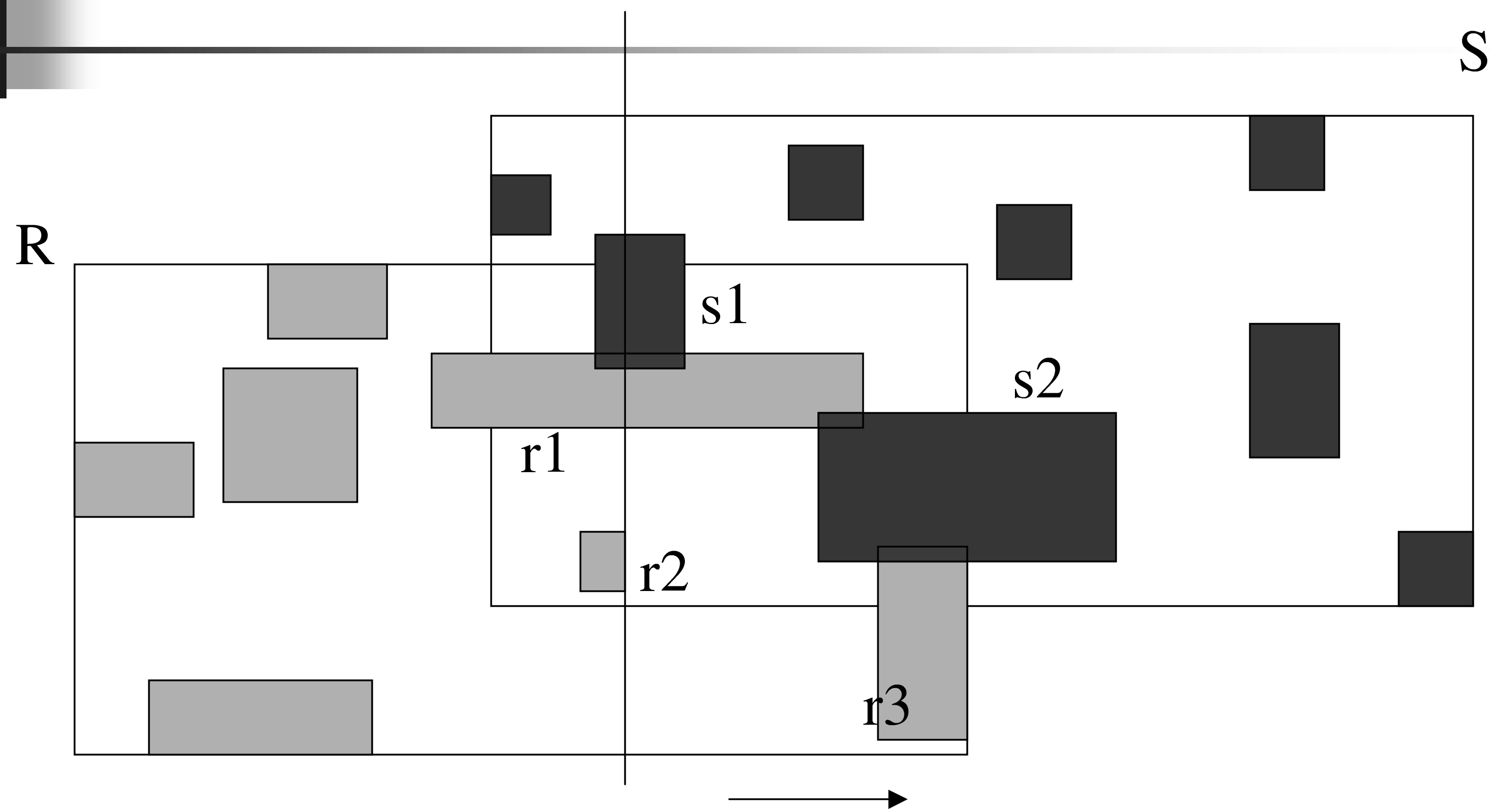


Using Plane Sweep



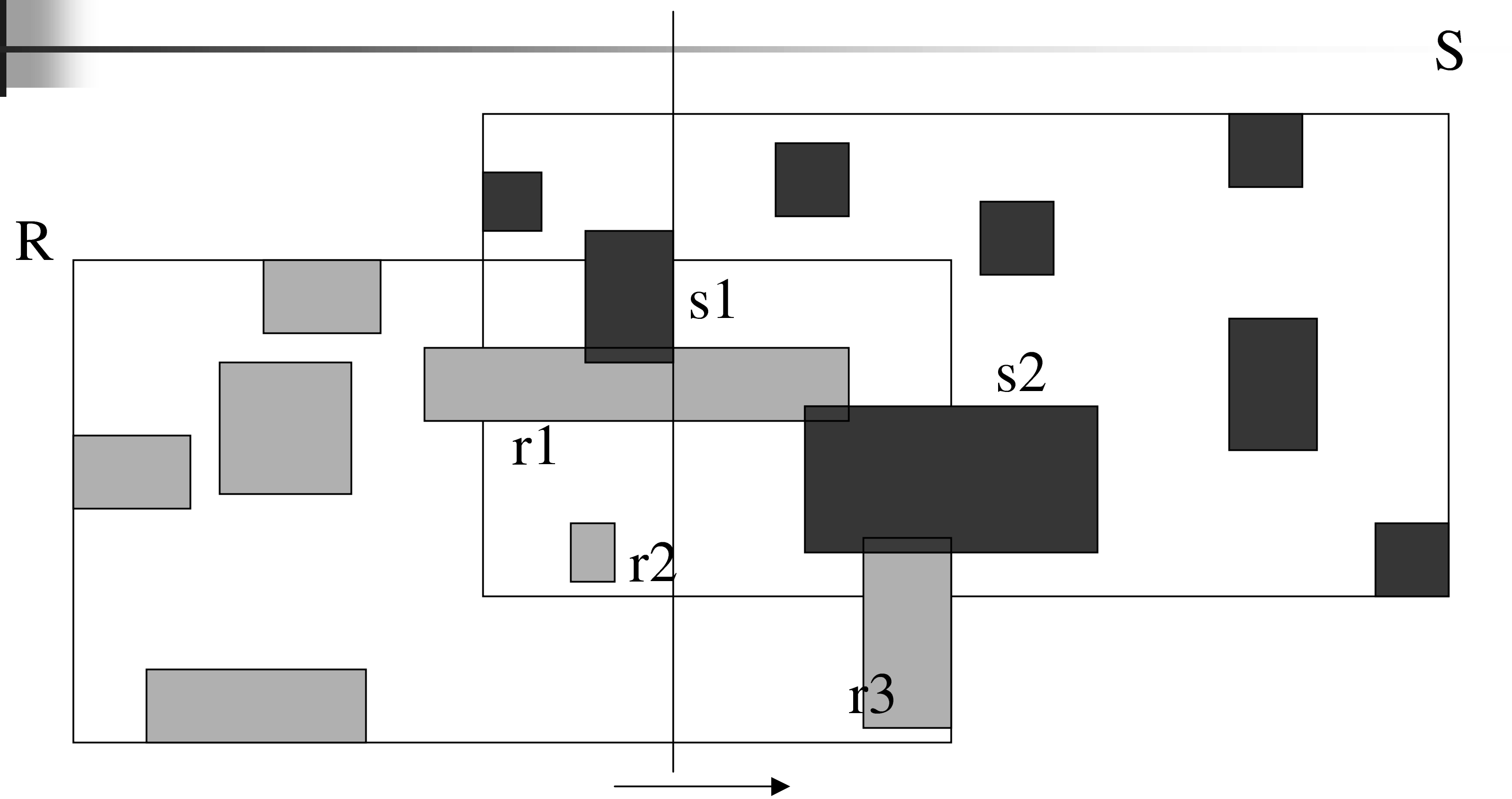
Check if r2 and s1 intersect along y
Do not add (r2,s1) to result

Using Plane Sweep



Reached the end of r2
Start with next entry s1

Using Plane Sweep

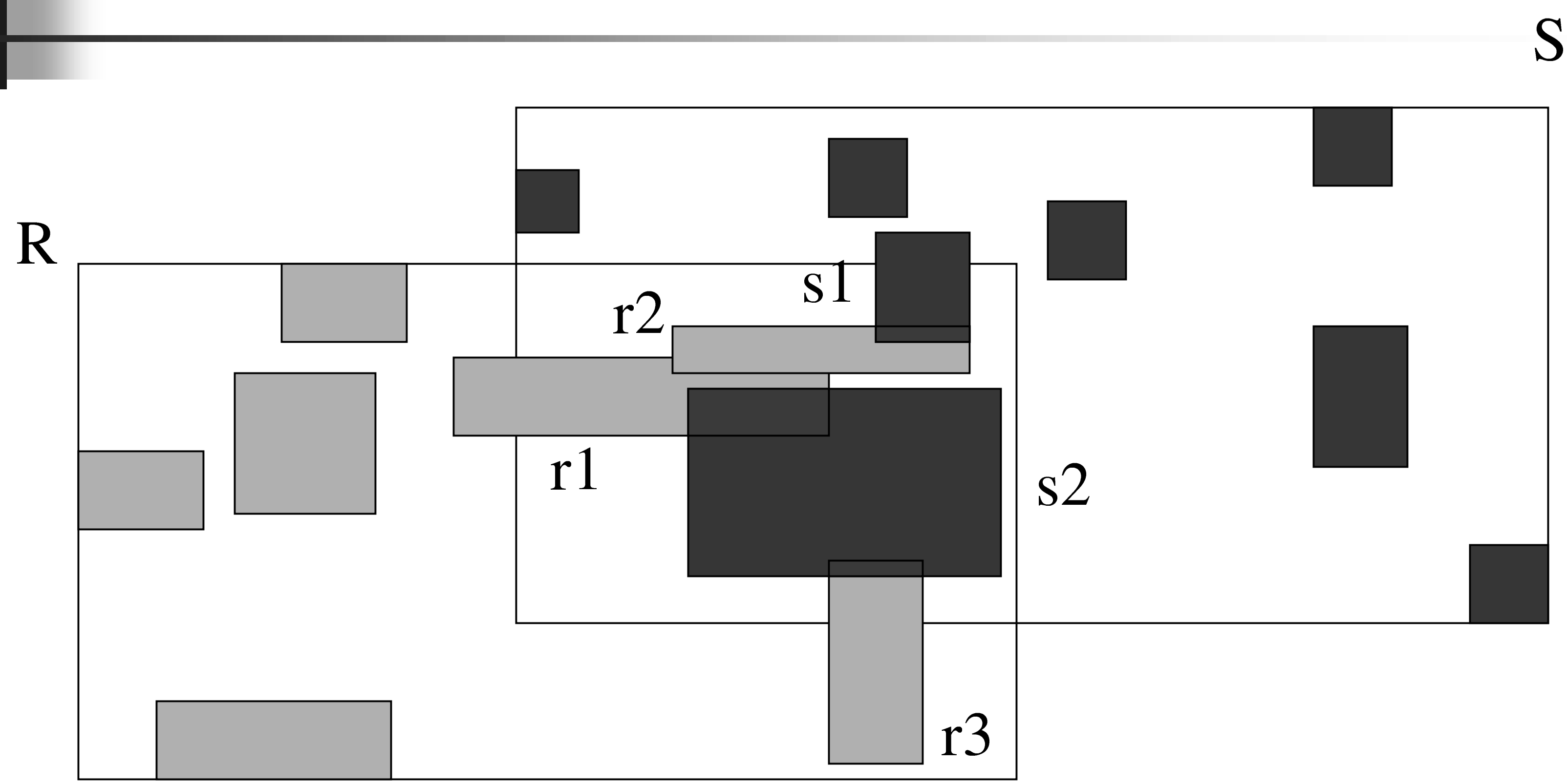


Total of 2(r1) + 1(r2) + 0 (s1)+ 1(s2)+ 0(r3) = 4 comparisons

Reducing I/O

- Read schedule r1, s1, s2, r3
- Every subtree examined only once
- Consider a slightly different layout

Reducing I/O

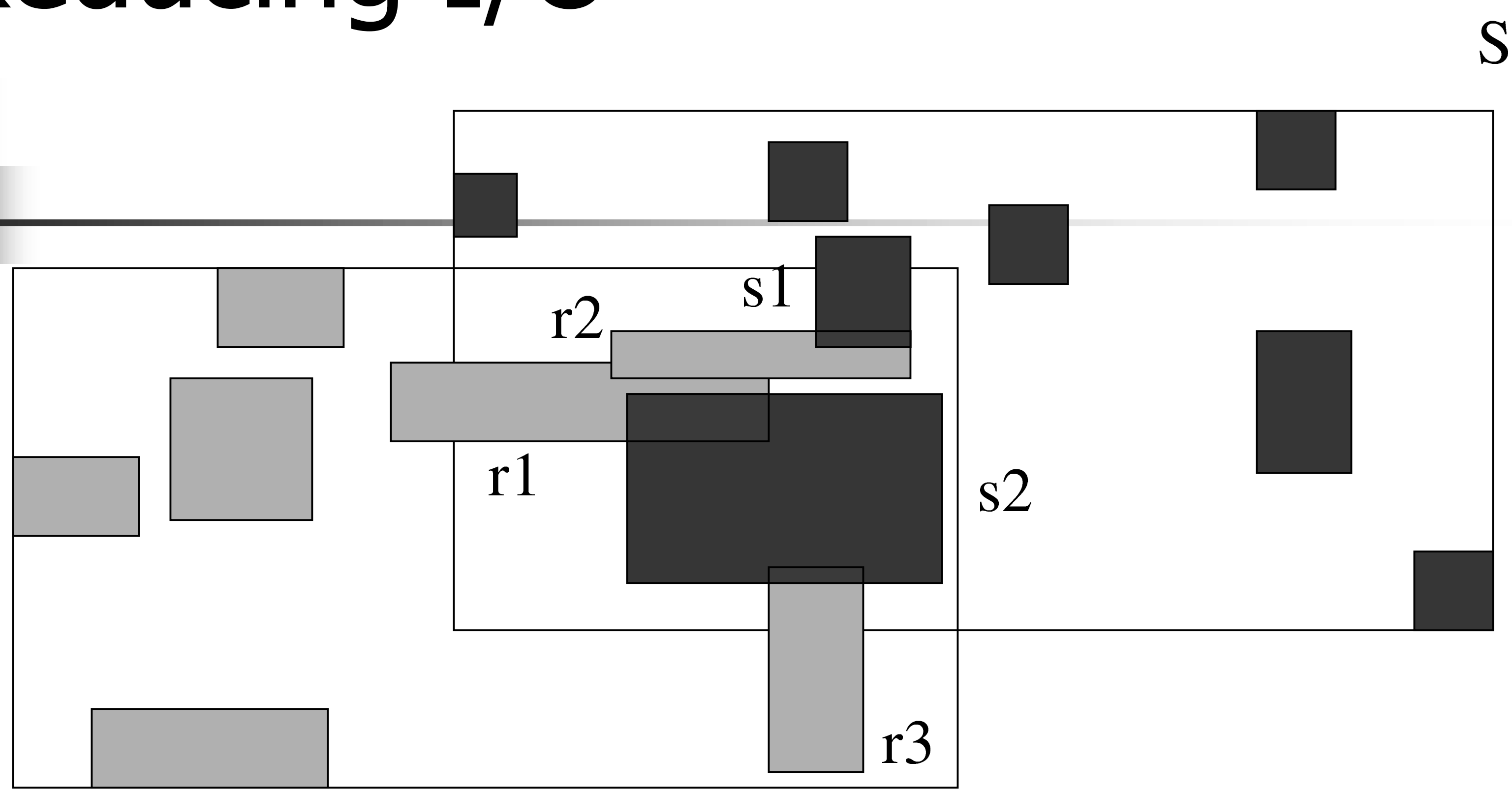


Read schedule is r1, s2, r2, s1, s2, r3
Subtree s2 is examined twice

Pinning of nodes

- After examining a pair (E,F), compute the degree of intersection of each entry
  - degree(E) is the number of intersections between E and unprocessed rectangles of the other dataset
- If the degrees are non-zero, pin the pages of the entry with maximum degree
- Perform spatial joins for this page
- Continue with plane sweep

Reducing I/O



After computing join(r1,s2),
degree(r1) = 0
degree(s2) = 1
So, examine s2 next
Read schedule = r1, s2, r3, r2, s1
Subtree s2 examined only once

References

- [SK98] Optimal multi-step k-nearest neighbor search, T. Seidl and H. Kriegel, SIGMOD 1998: 154--165.
- [BBKK01] Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data, C. Bohm, B. Braunmüller, F. Krebs and H.-P. Kriegel, SIGMOD 2001.
- [RKV95] Roussopoulos N., Kelley S., Vincent F. *Nearest Neighbor Queries*. Proceedings of the ACM-SIGMOD International Conference on Management of Data, pages 71-79, 1995.

References

- [HS99] G. R. Hjaltason and H. Samet, Distance browsing in spatial databases, *ACM Transactions on Database Systems* 24, 2 (June 1999), 265-318
- [O89] Jack A. Orenstein: Redundancy in Spatial Databases. SIGMOD Conference 1989: 294-305
- [PW96] Jignesh M. Patel, David J. DeWitt: Partition Based Spatial-Merge Join. SIGMOD Conference 1996: 259-270
- [LR96] Ming-Ling Lo, Chinya V. Ravishankar: Spatial Hash-Joins. SIGMOD Conference 1996: 247-258

References

- [KS97] Nick Koudas, Kenneth C. Sevcik: Size Separation Spatial Join. SIGMOD Conference 1997: 324-335
- [HJR97] Yun-Wu Huang, Ning Jing, Elke A. Rundensteiner: Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations. VLDB 1997, 396-405
- [BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, Bernhard Seeger: Efficient Processing of Spatial Joins Using R-Trees. SIGMOD Conference 1993: 237-246