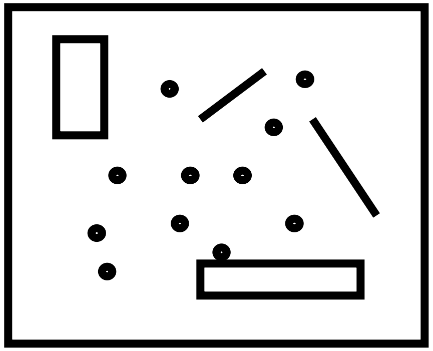
Spatial Queries

Nearest Neighbor and Join Queries

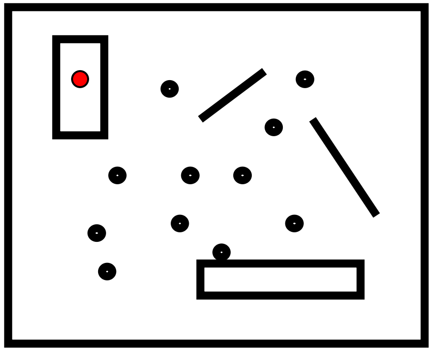
Spatial Queries

- 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)



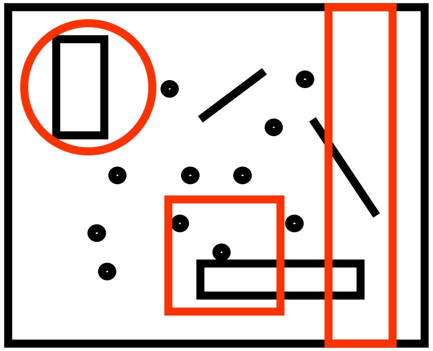
Spatial Queries

- 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)



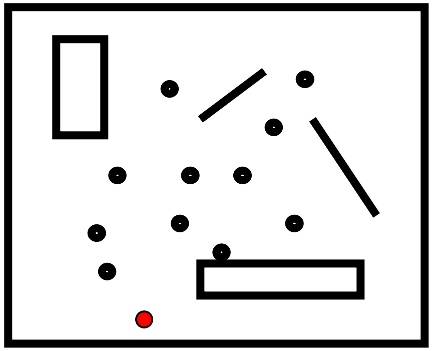
Spatial Queries

- 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)



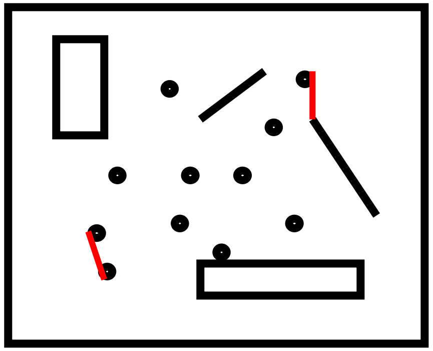
Spatial Queries

- 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)

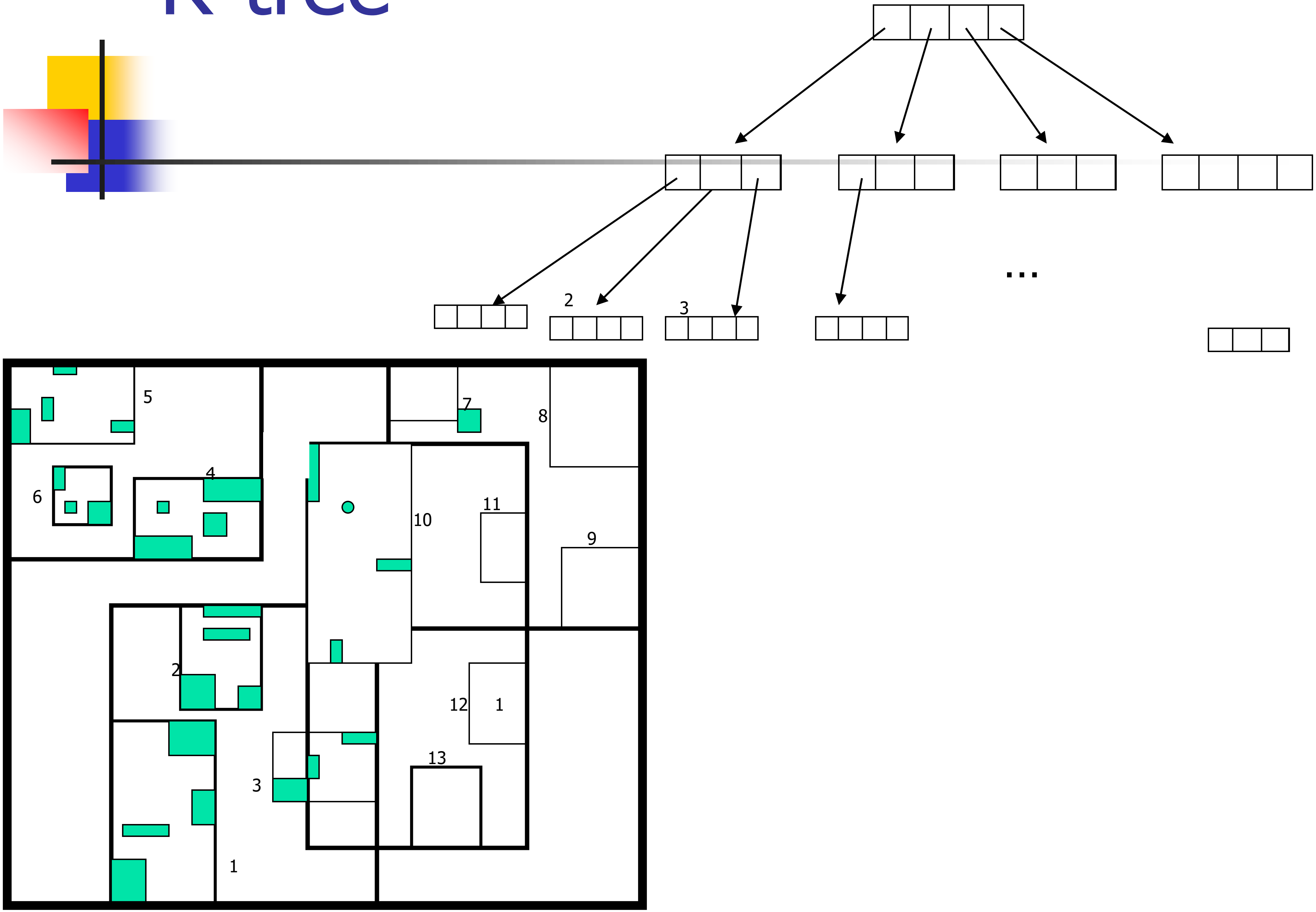


Spatial Queries

- 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

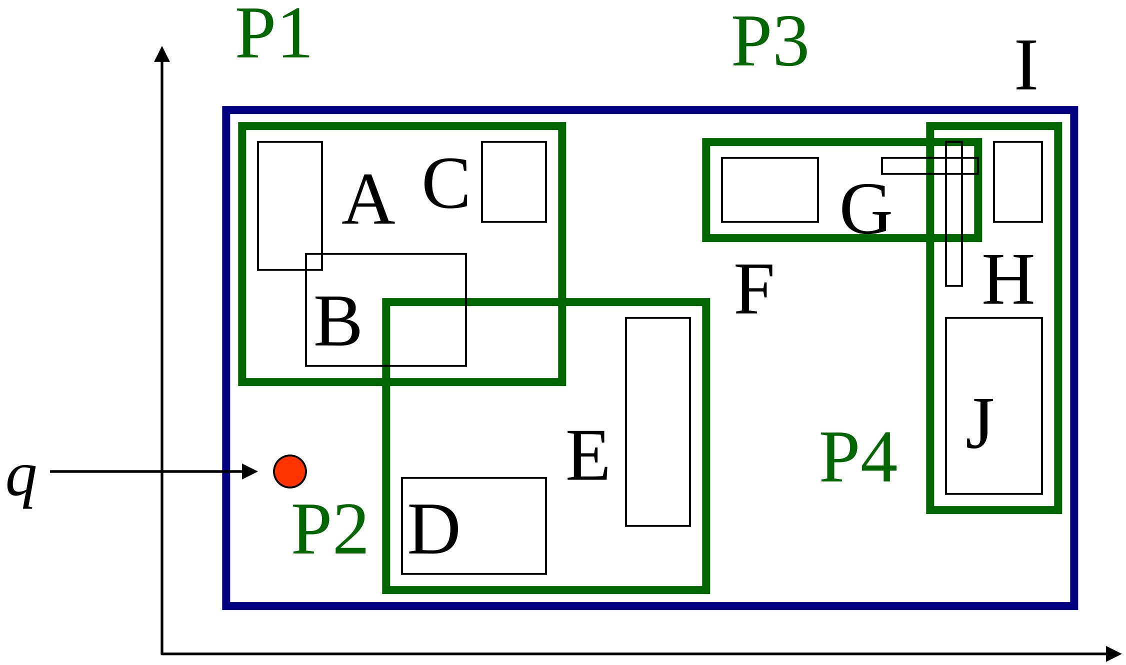


R-trees - Range search

pseudocode:

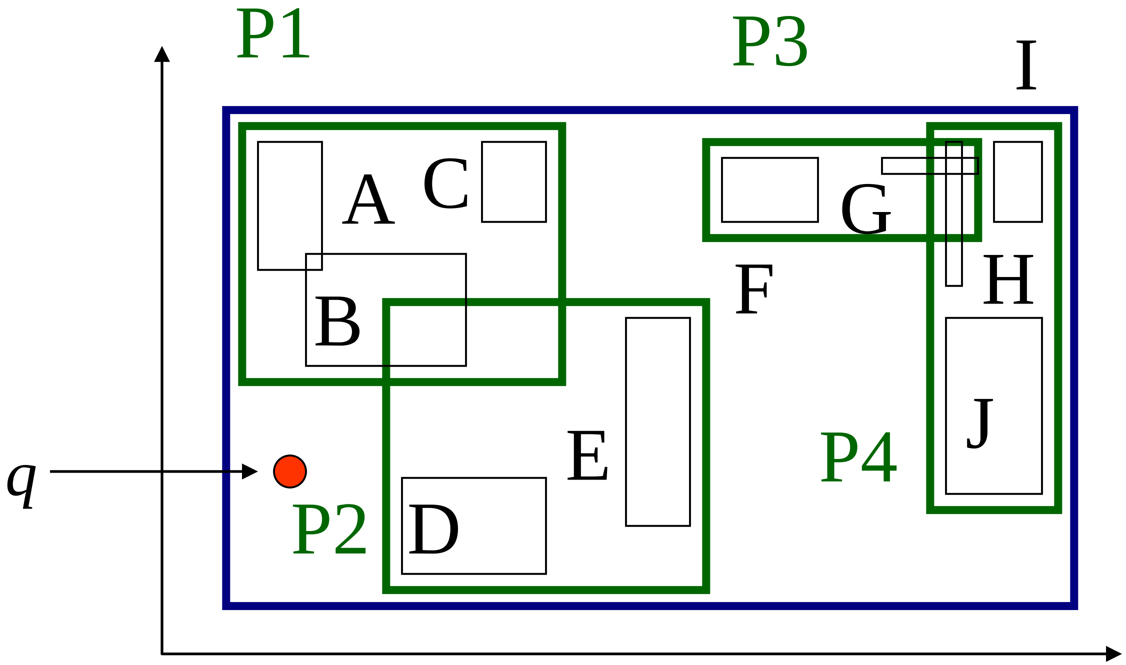
```
check the root
   for each branch,
      if its MBR intersects the query rectangle
            apply range-search (or print out, if this
                    is a leaf)
```

R-trees - NN search



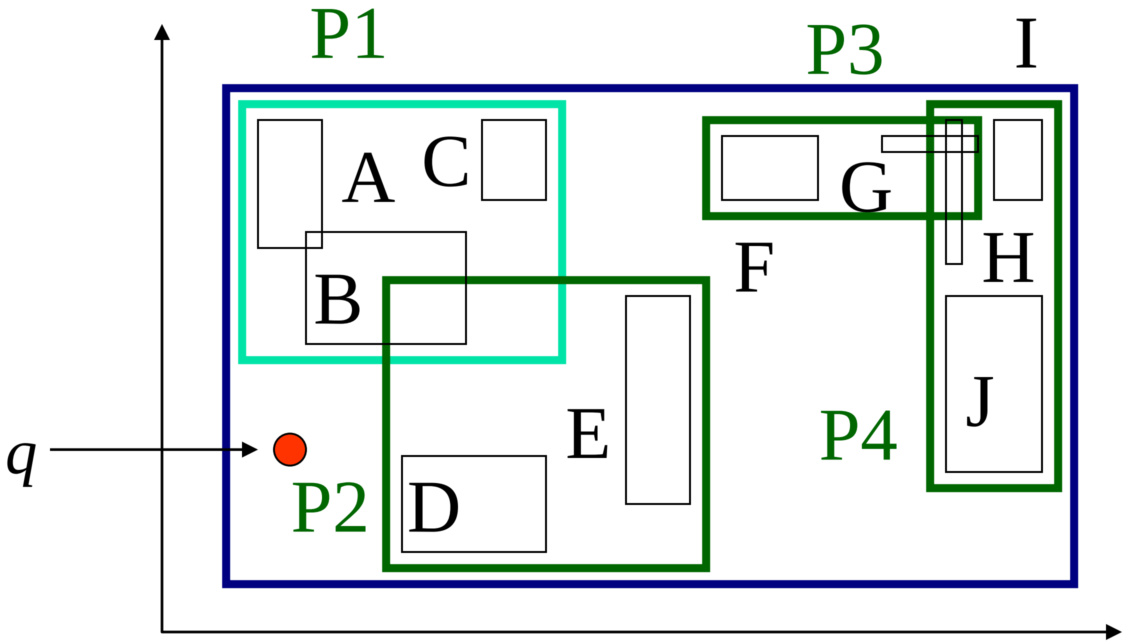
R-trees - NN search

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



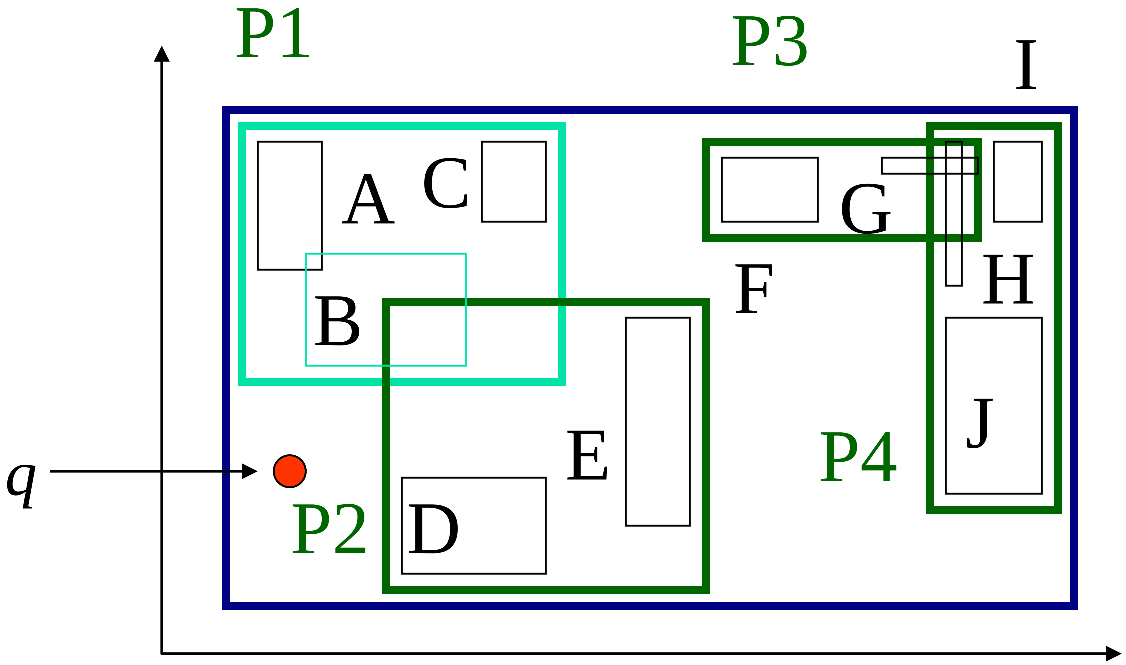
R-trees - NN search

- A1: depth-first search; then range query



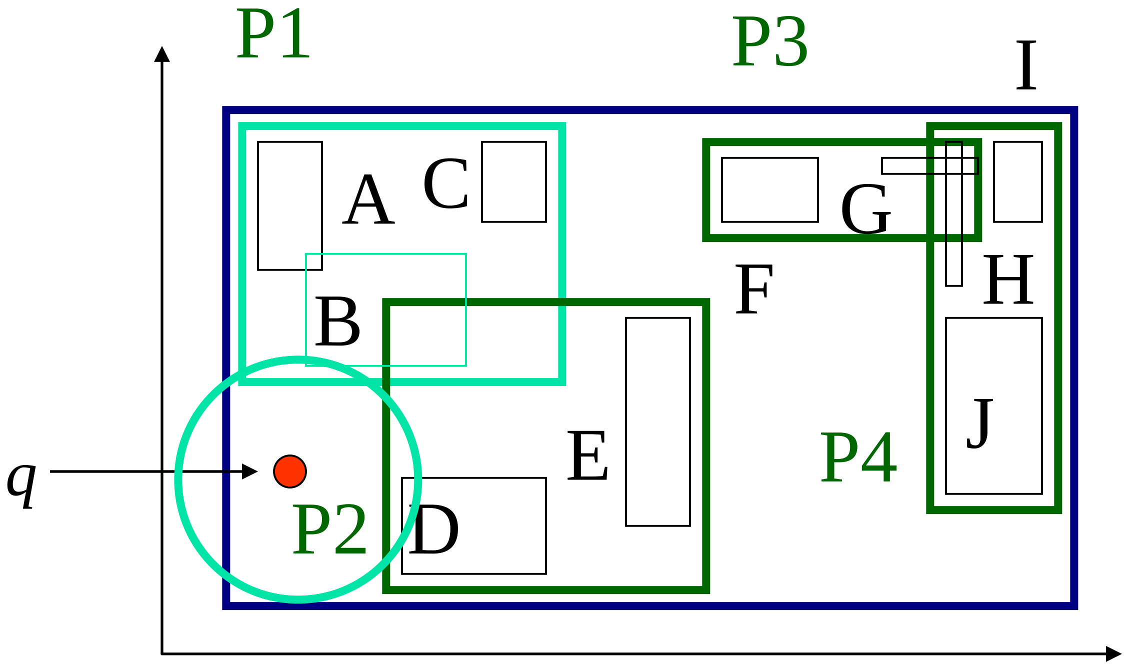
R-trees - NN search

- A1: depth-first search; then range query



R-trees - NN search

- A1: depth-first search; then range query

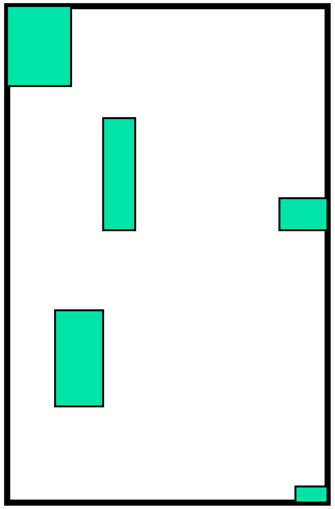


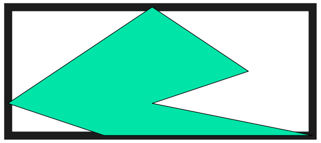
R-trees - NN search: Branch and Bound

- A2: [Roussopoulos+, sigmod95]:
  - At each node, priority queue, with promising MBRs, and their best and worst-case distance
- main idea: Every side (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

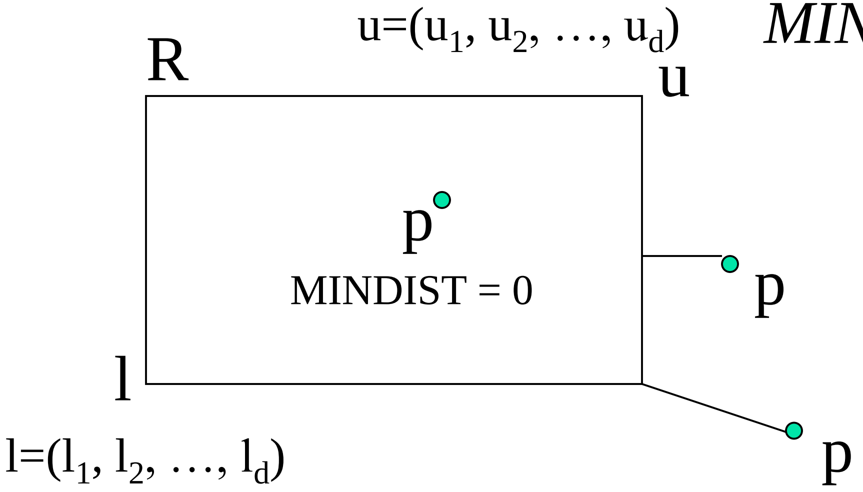
- Visit an MBR (node) only when necessary
- How to do pruning? Using MINDIST and MINMAXDIST

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



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

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

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

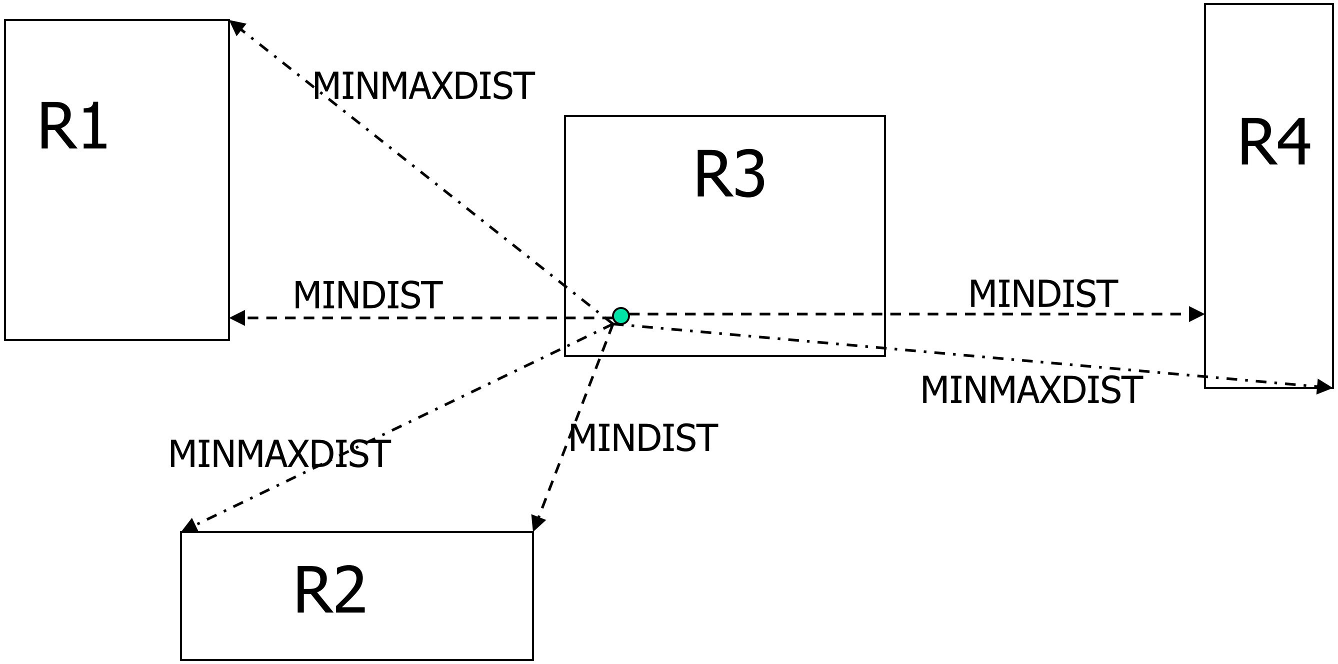
MINMAXDIST

- 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.

$$\$o\ \hat{I}\ R, \|(p,o)\|\ \text{£}\ MINMAXDIST(p,R)$$

MINDIST and MINMAXDIST

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

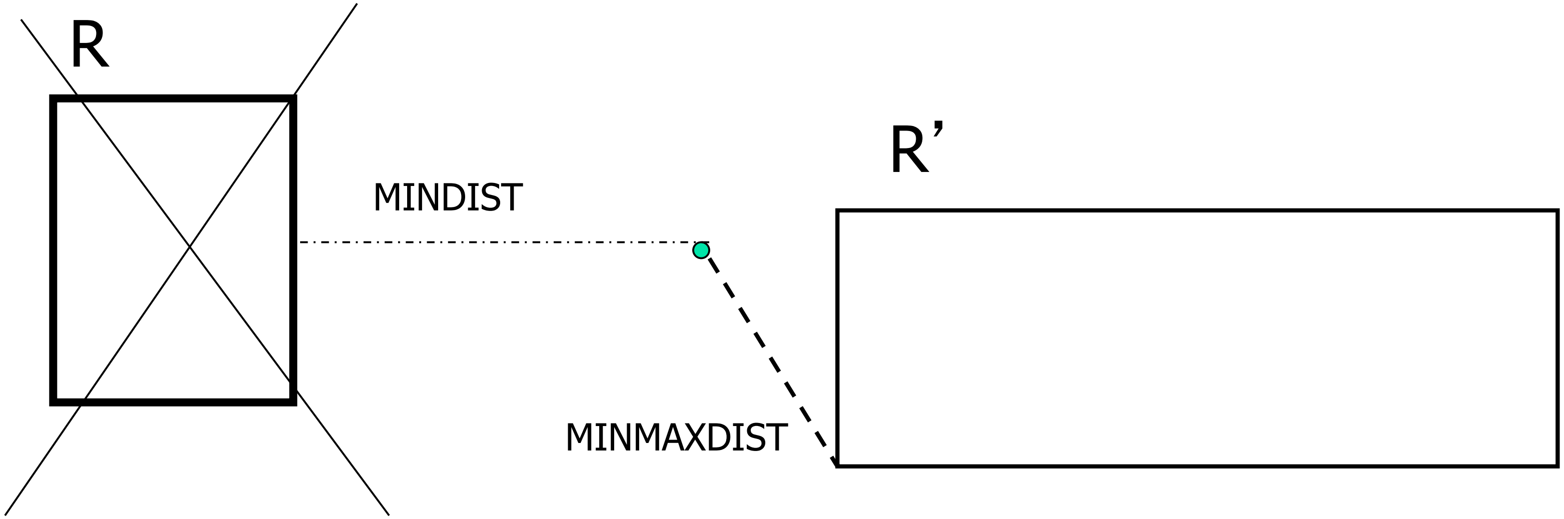


Pruning in NN search

- 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)

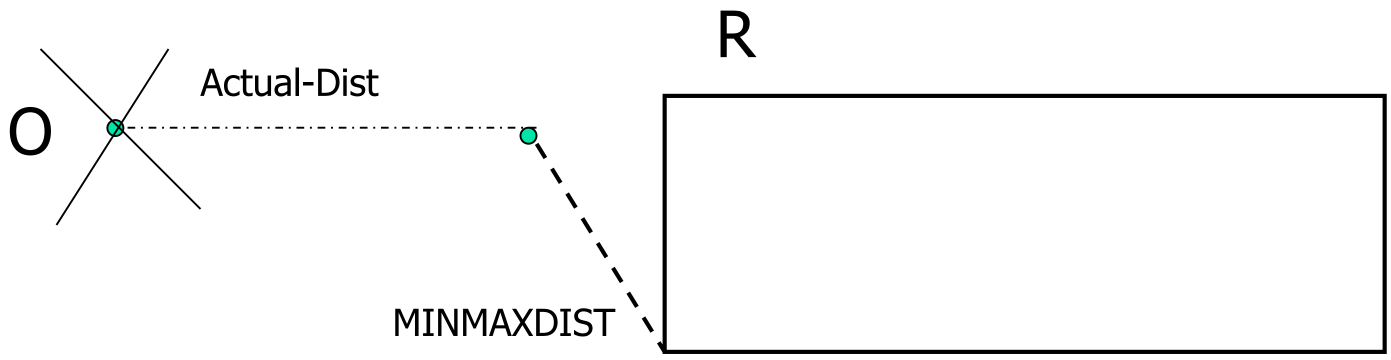
Pruning 1 example

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



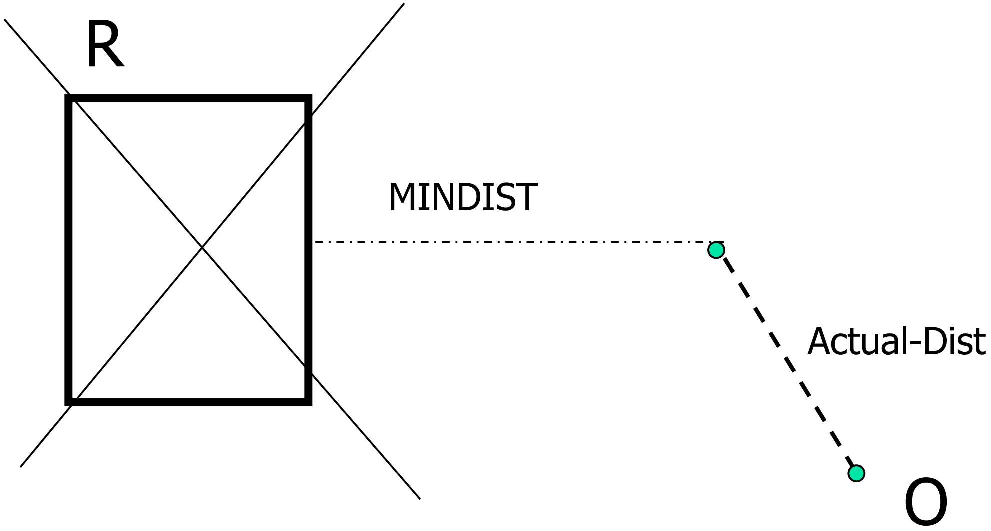
Pruning 2 example

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



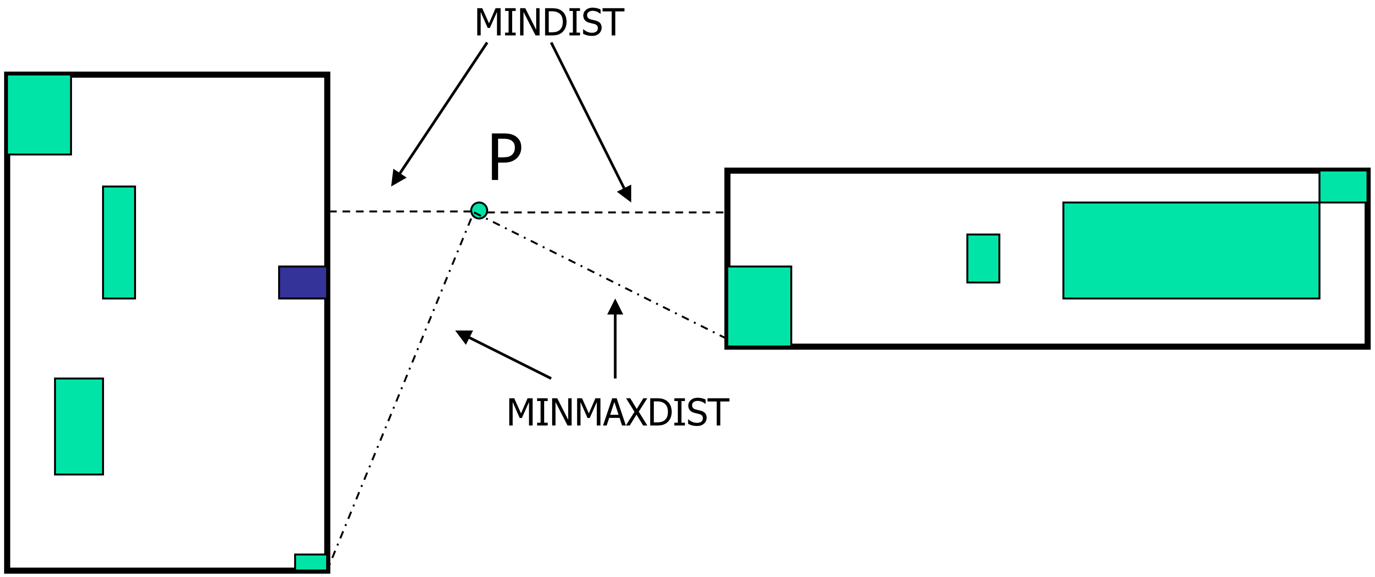
Pruning 3 example

- 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

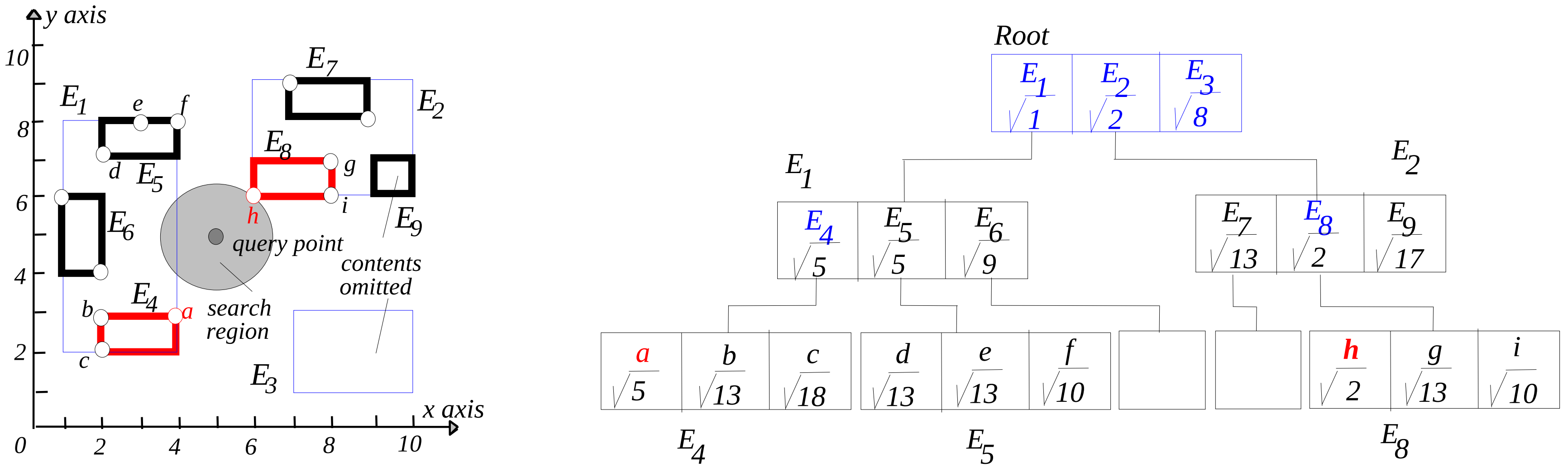
- Pruning is done using the k-th distance

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	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 $\sqrt{13}$	{empty}
	E_4 $\sqrt{5}$	E_5 $\sqrt{5}$	E_3 $\sqrt{8}$	E_6 $\sqrt{9}$	i $\sqrt{10}$	E_7 $\sqrt{13}$	g $\sqrt{13}$		{(h, $\sqrt{2}$)}

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

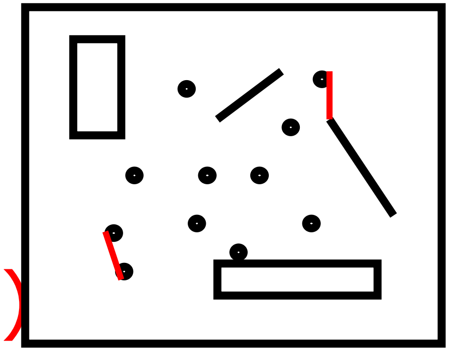
- Best-First is the “optimal” algorithm in the sense that it visits all the necessary nodes and nothing more!

- 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

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)



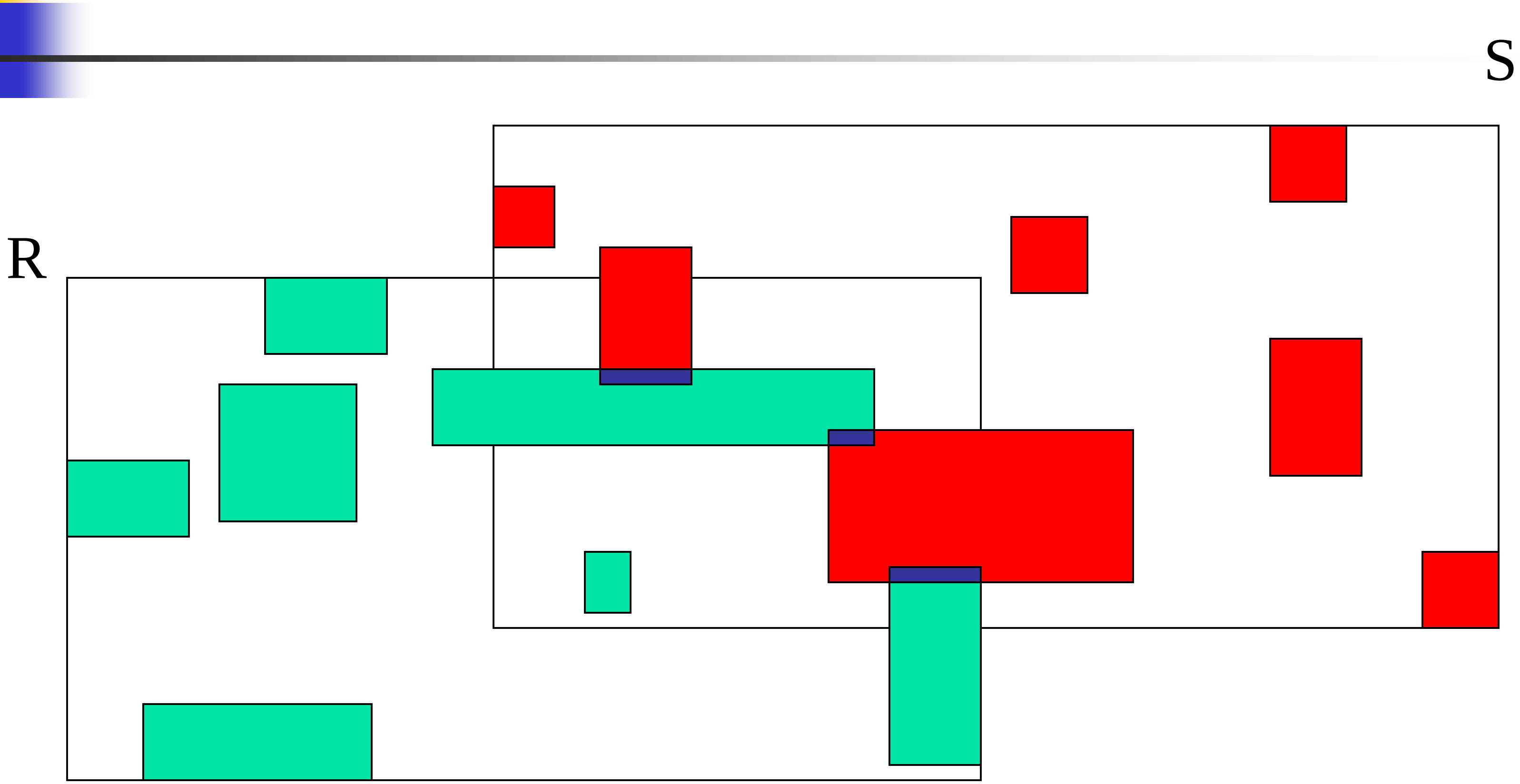
Spatial Join

- Find all parks in each city in MA
- Find all trails that go through a forest in MA
- 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 using recursive partitioning [BBKK01]
- With index structures [BKS93, HJR97]
  - k-d trees and grid files
  - R-trees

R-tree based Join [BKS93]

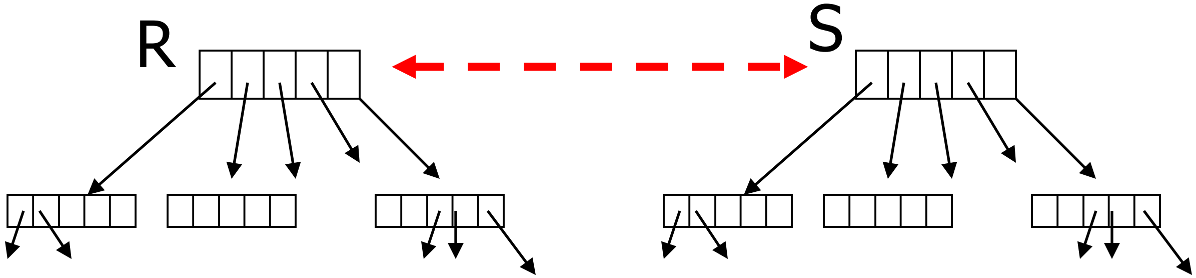


Join1(R,S)

- Tree synchronized traversal algorithm

```
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)
```

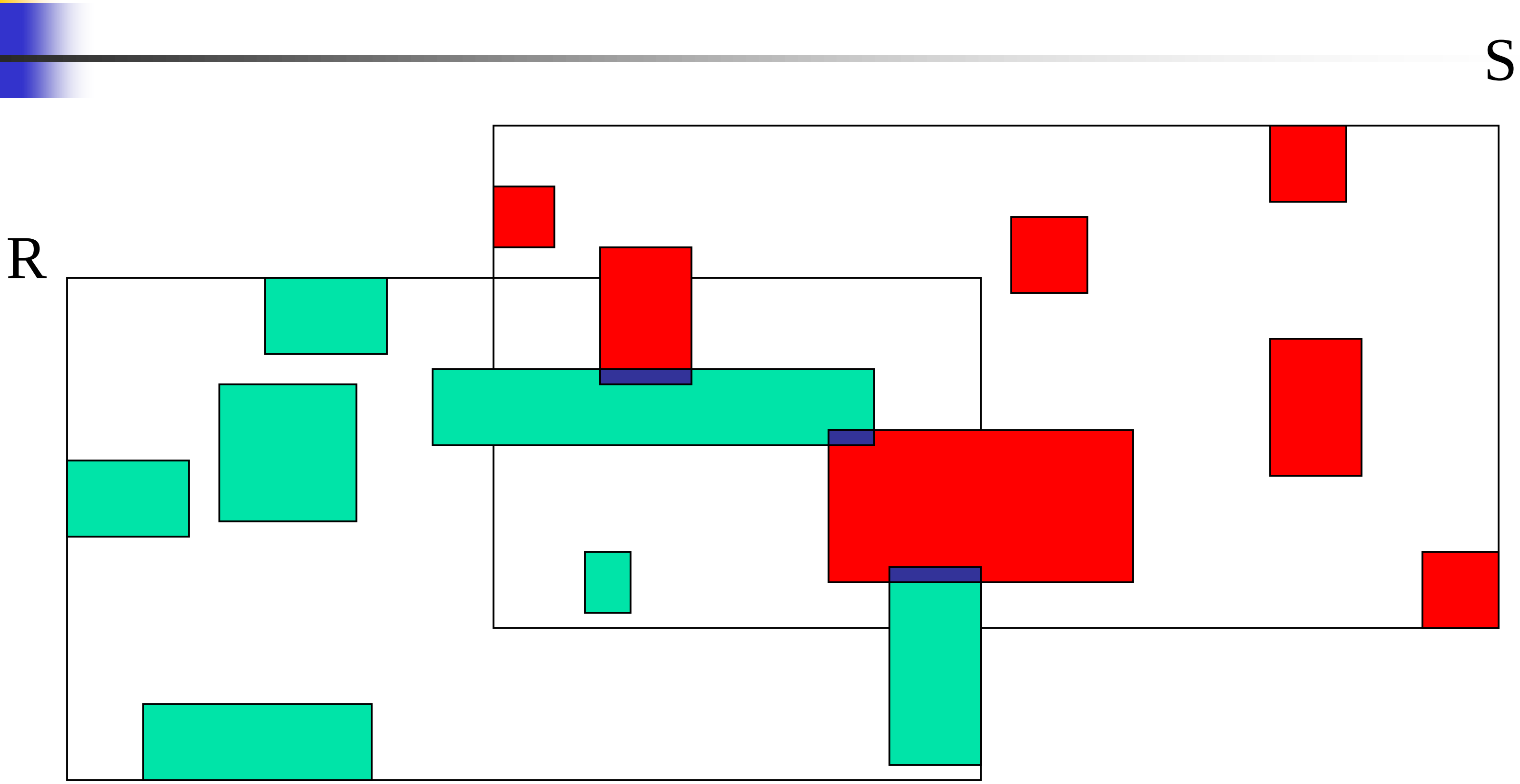
- Until all pairs are examined
- CPU and I/O bottleneck



CPU – Time Tuning

- Two ways to improve CPU – time
  - Restricting the search space
  - Spatial sorting and plane sweep

Reducing CPU bottleneck



Join2(R,S,IntersectedVol)

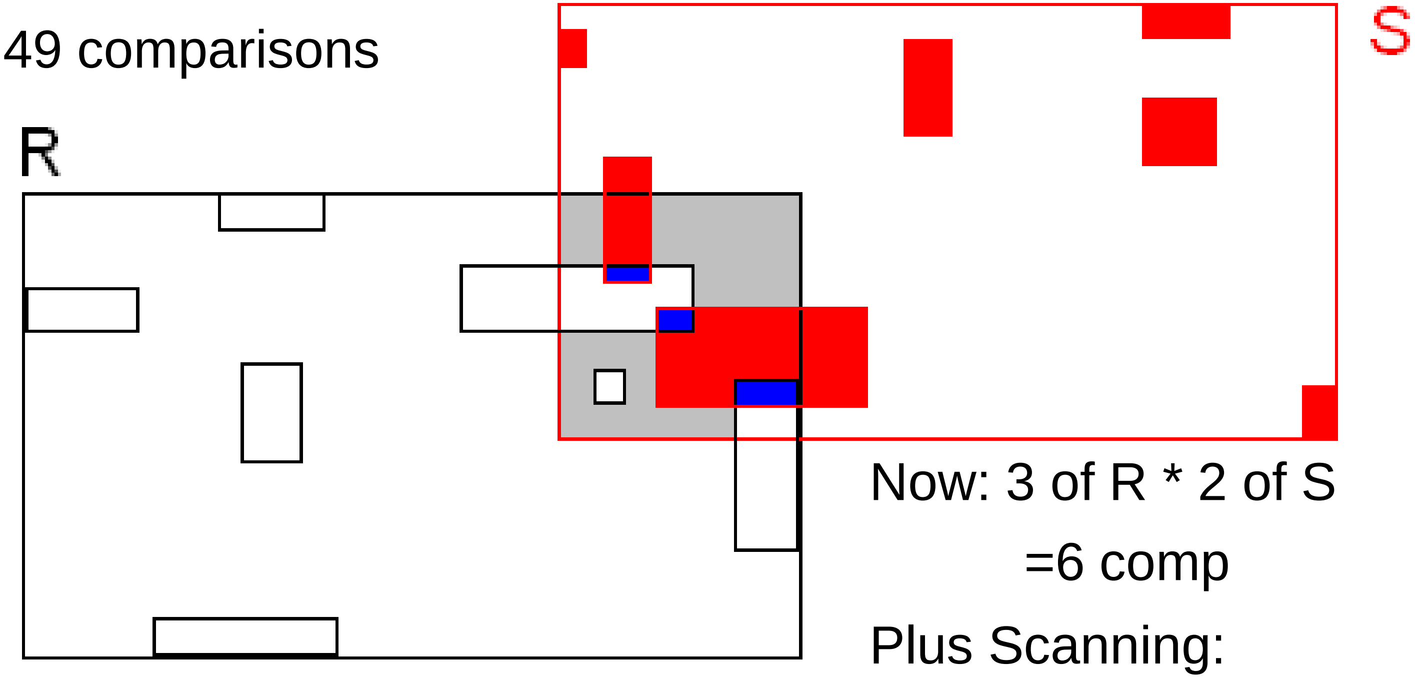
```
Join2(R,S,IV)
Repeat
    Find a pair of intersecting entries E in R and F in S that overlap with IV
    If R and S are leaf pages then
            add (E,F) to result-set
    Else  Join2(E,F,CommonEF)
```

- Until all pairs are examined
- In general, number of comparisons equals
  - size(R) + size(S) + relevant(R)*relevant(S)
- Reduce the product term

Restricting the search space

Join1: 7 of R * 7 of S

= 49 comparisons



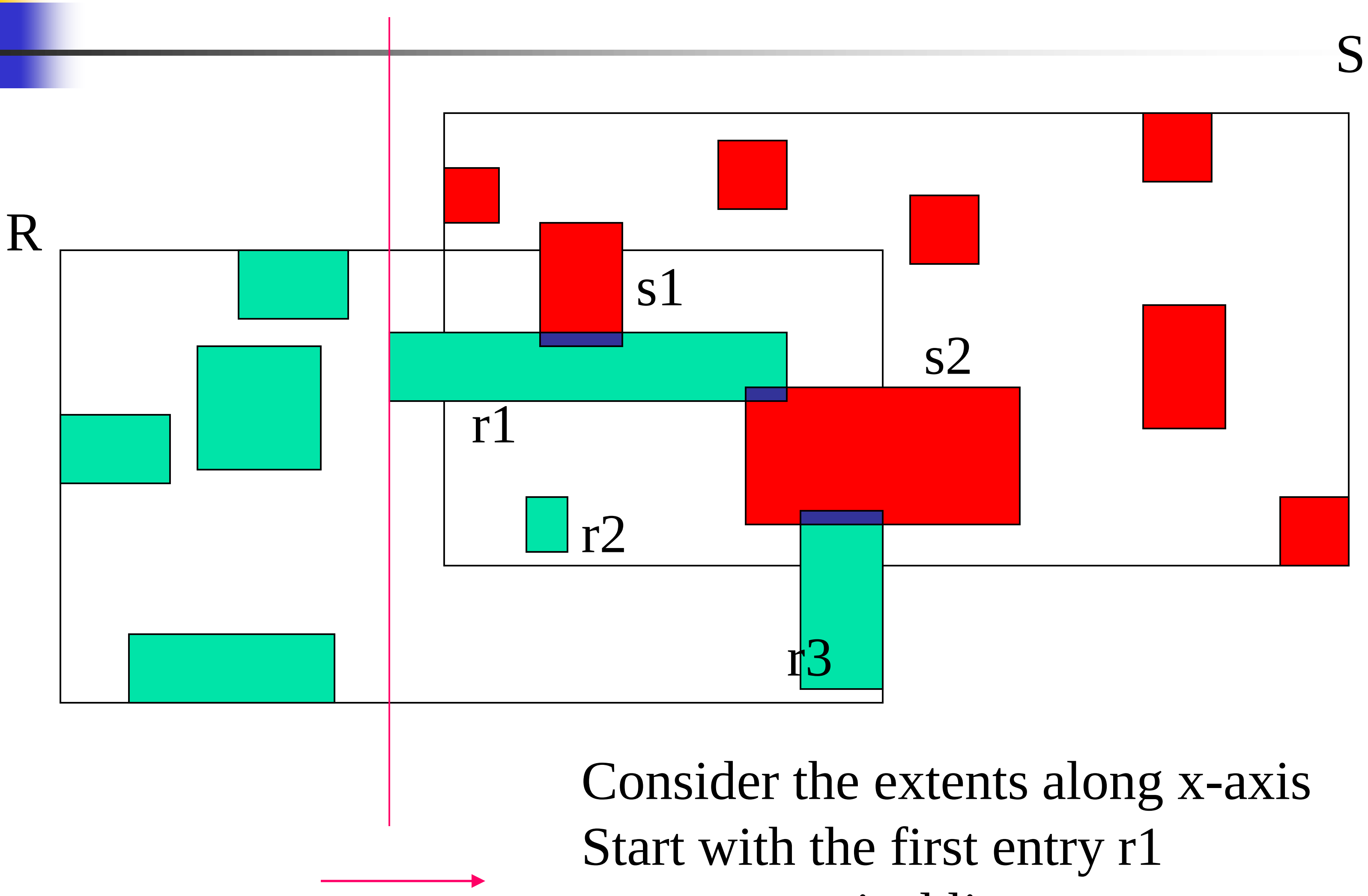
Now: 3 of R * 2 of S

=6 comp

Plus Scanning:
7 of R + 7 of S

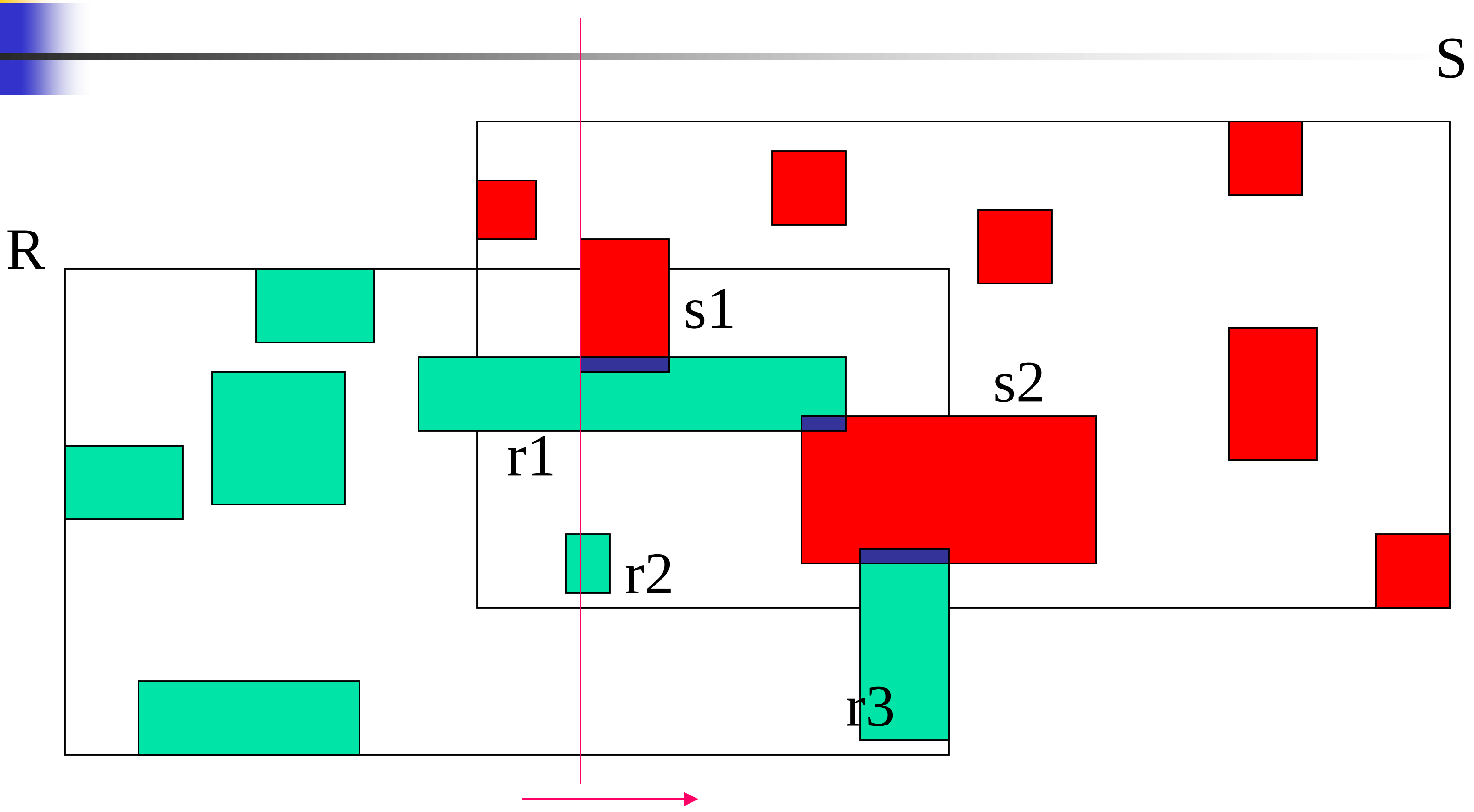
= 14 comp

Using Plane Sweep



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

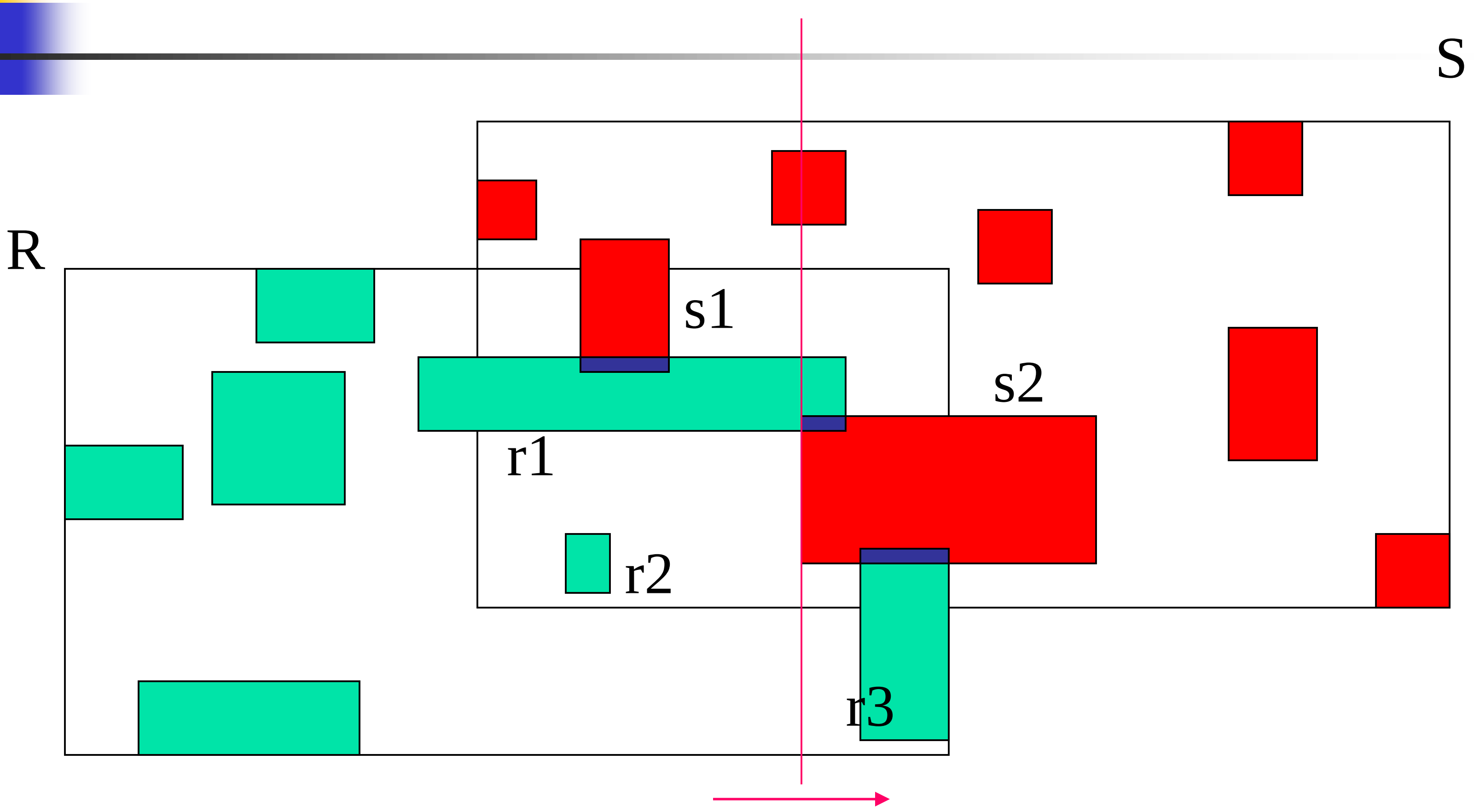
Using Plane Sweep



Check if (r1,s1) intersect along y-dimension

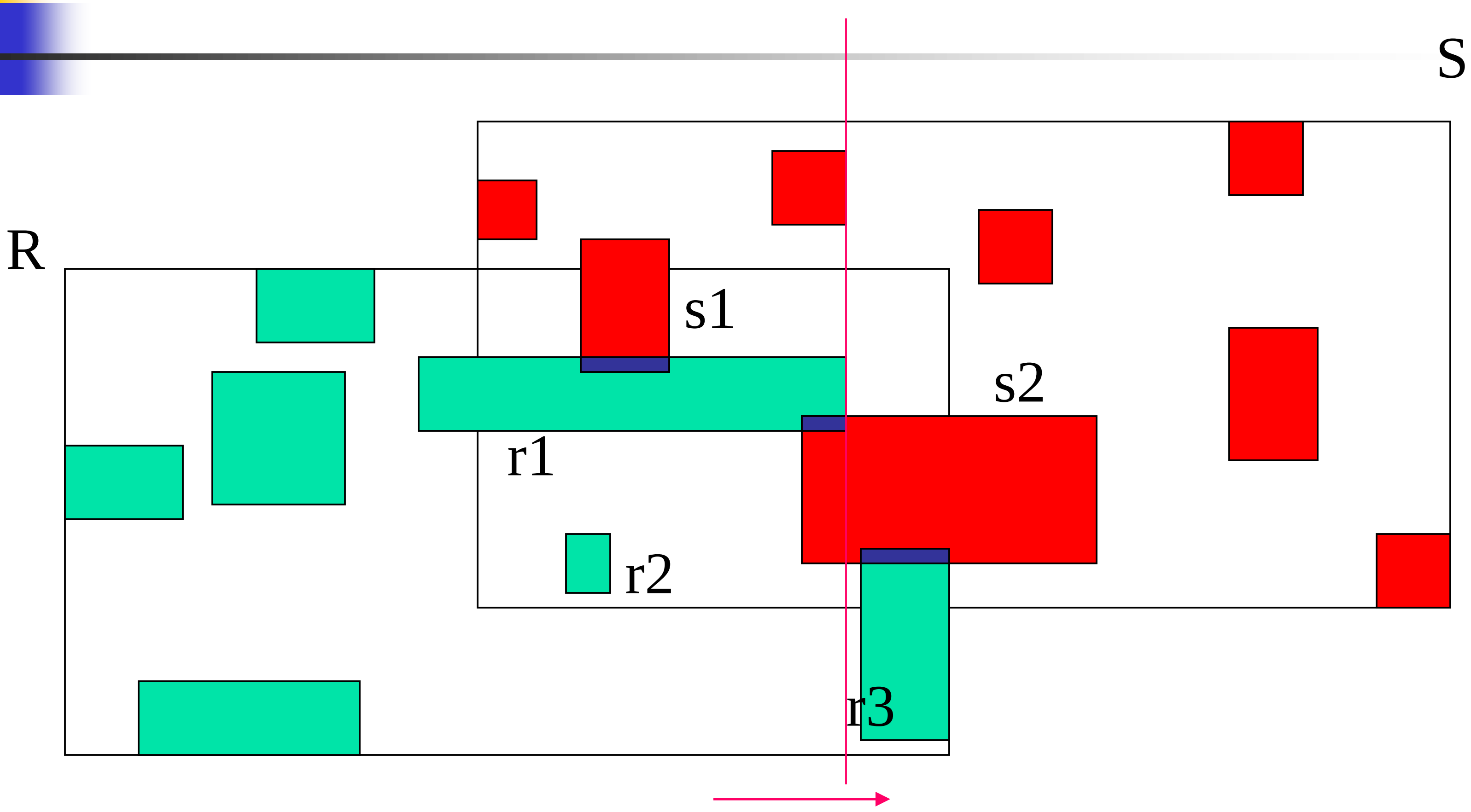
Add (r1,s1) to result set

Using Plane Sweep



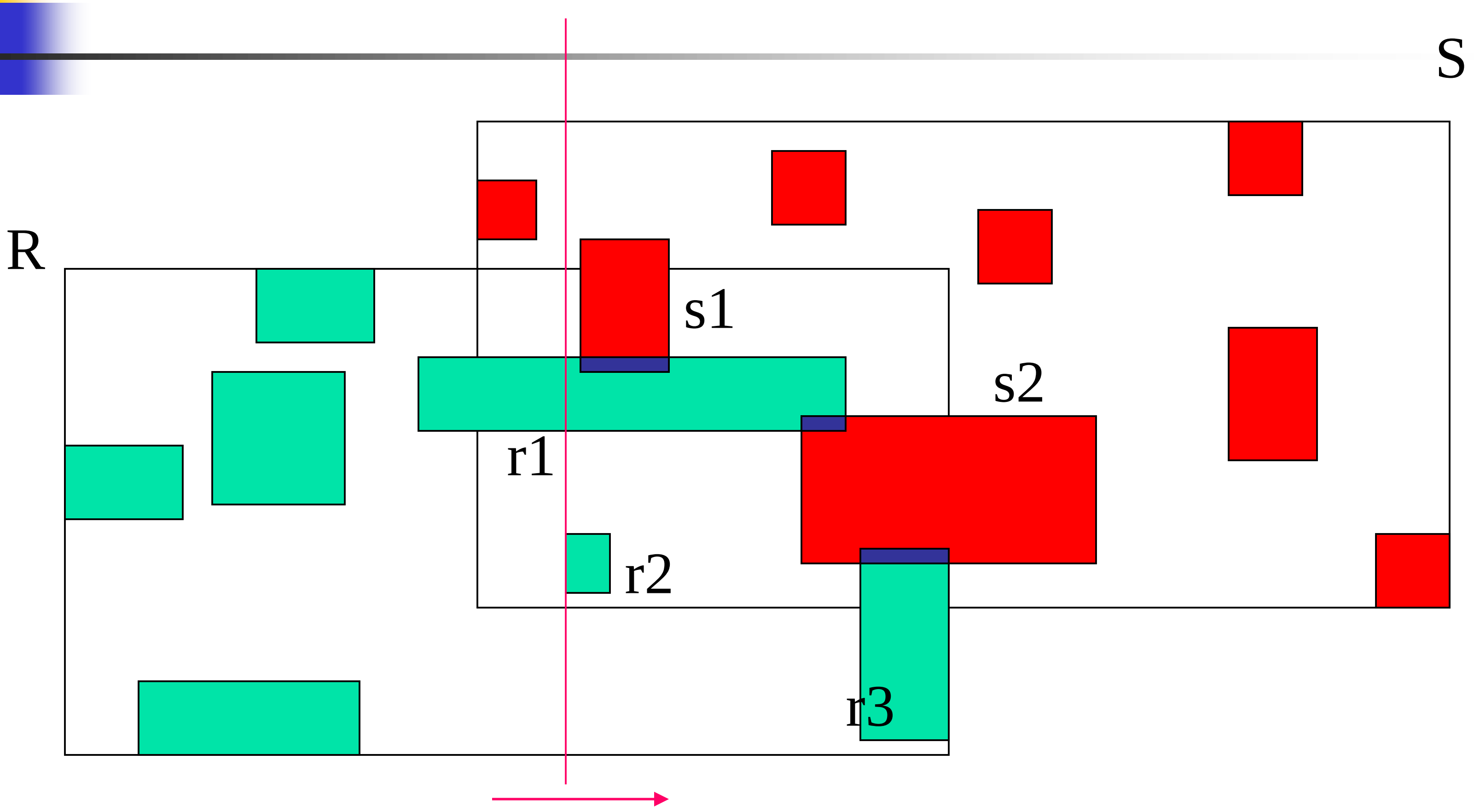
Check if (r1,s2) intersect along y-dimension
Add (r1,s2) to result set

Using Plane Sweep



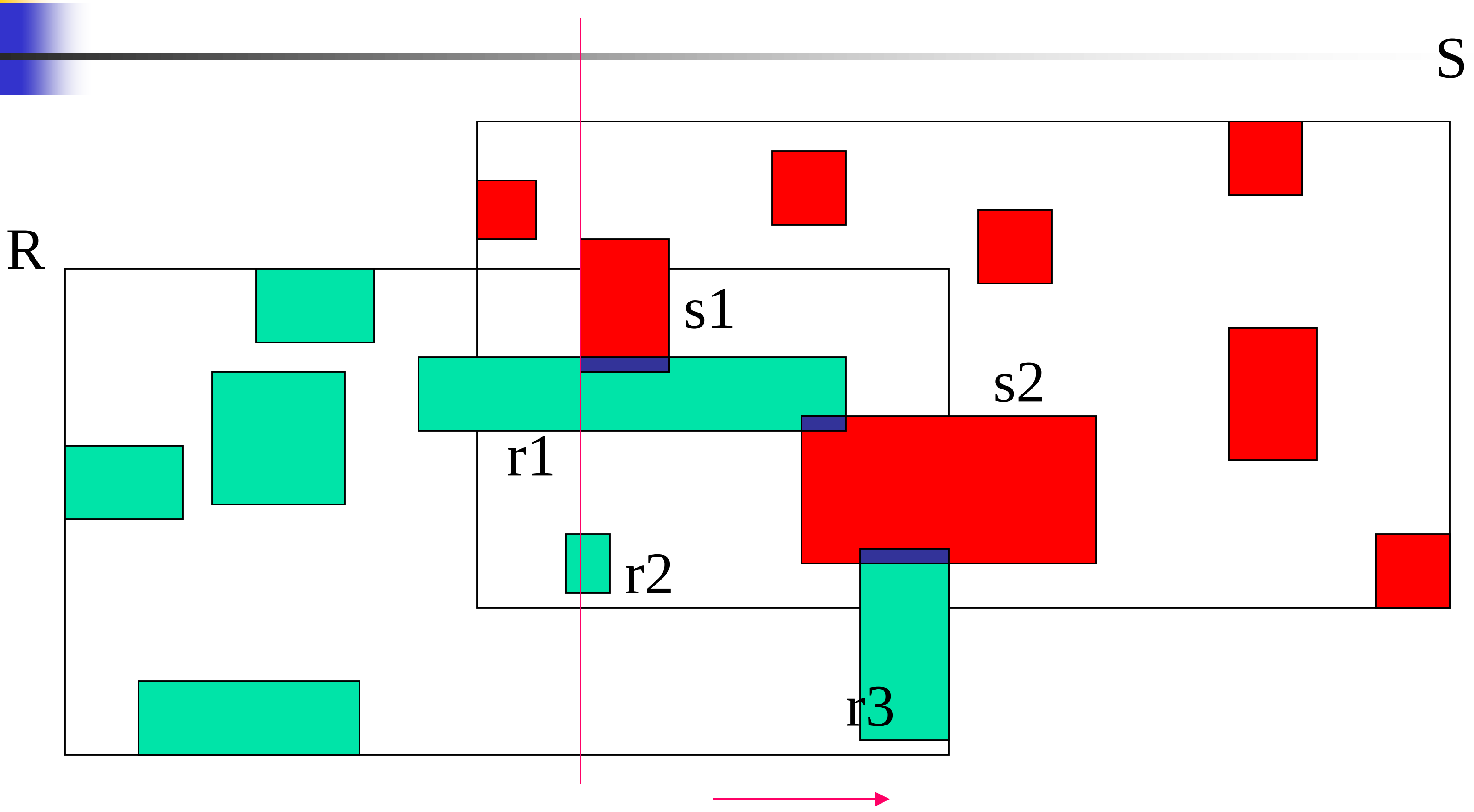
Reached the end of r1
Start with next entry r2

Using Plane Sweep



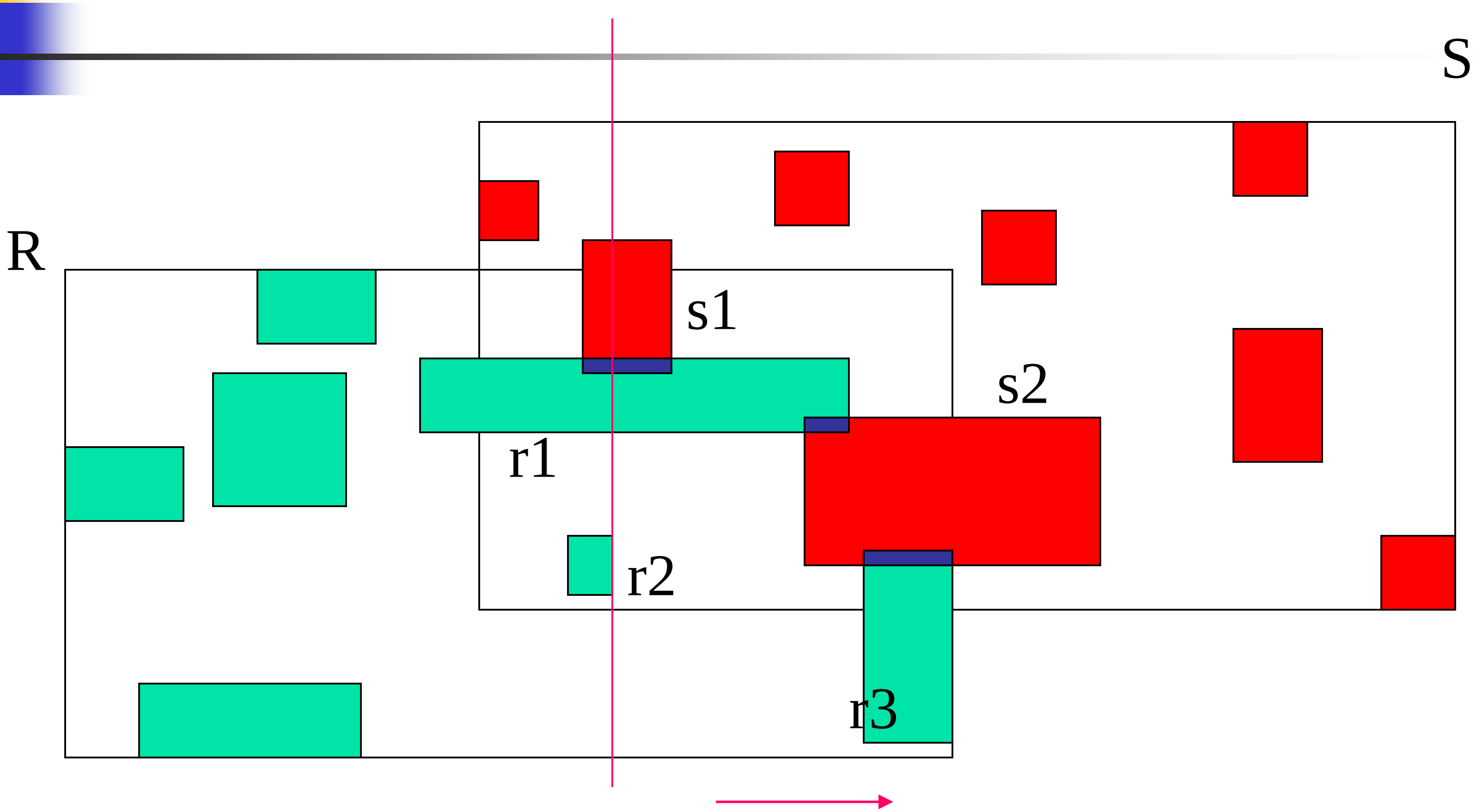
Reposition sweep line

Using Plane Sweep



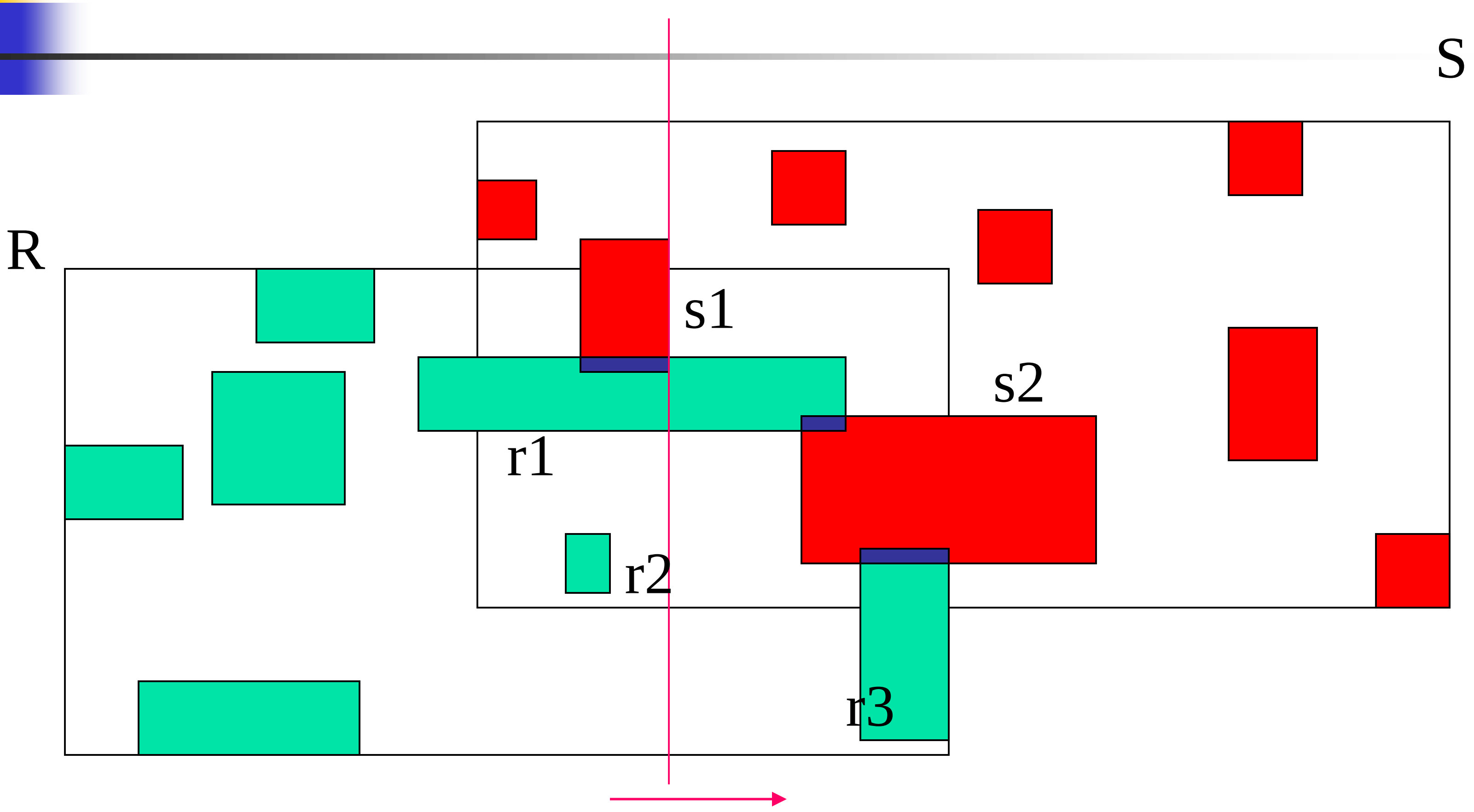
Check if r2 and s1 intersect along y
Do not add (r2,s1) to result

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

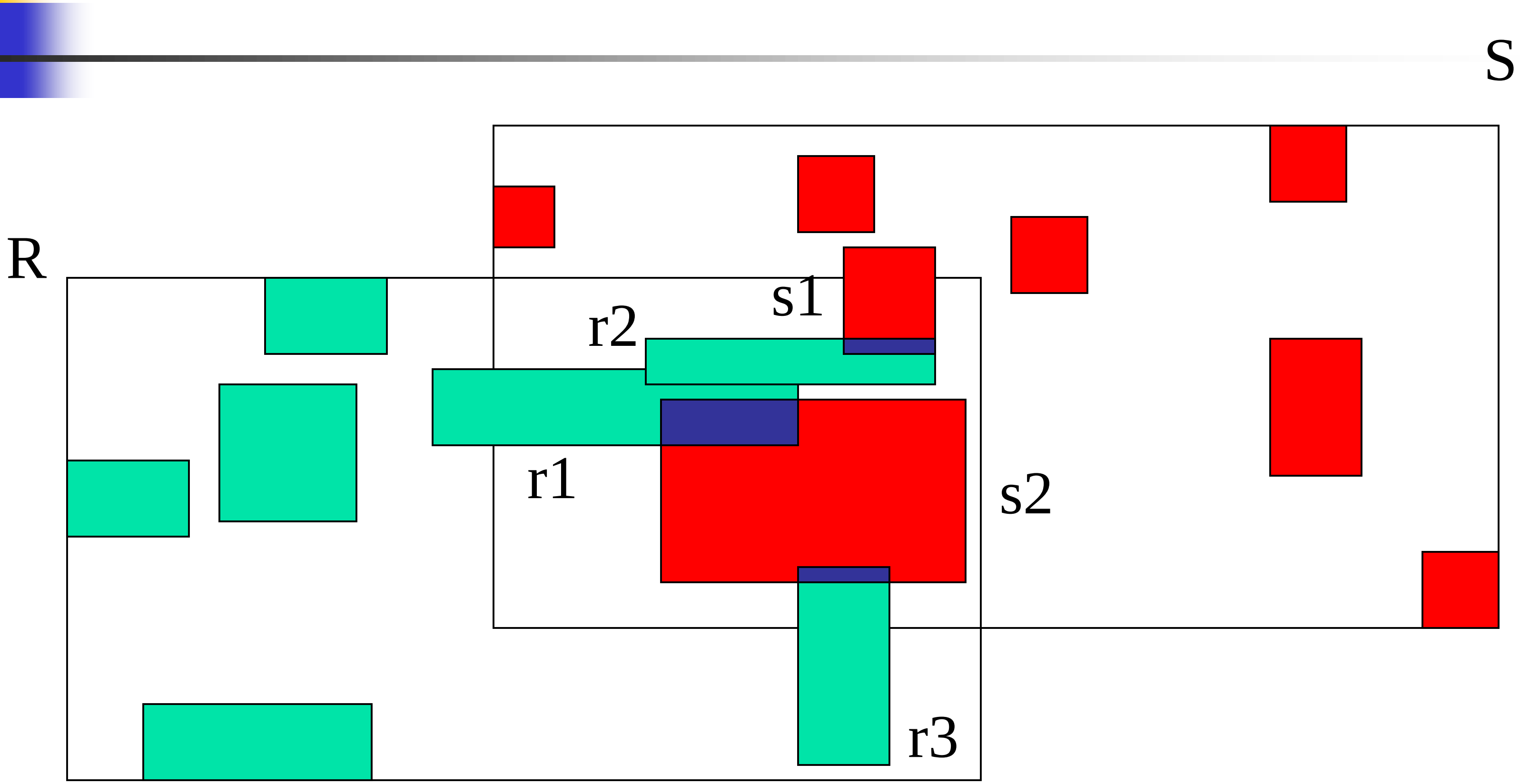
I/O Tunning

- Compute a read schedule of the pages to minimize the number of disk accesses
  - Local optimization policy based on spatial locality
- Three methods
  - Local plane sweep
  - Local plane sweep with pinning
  - Local z-order

Reducing I/O

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

Reducing I/O

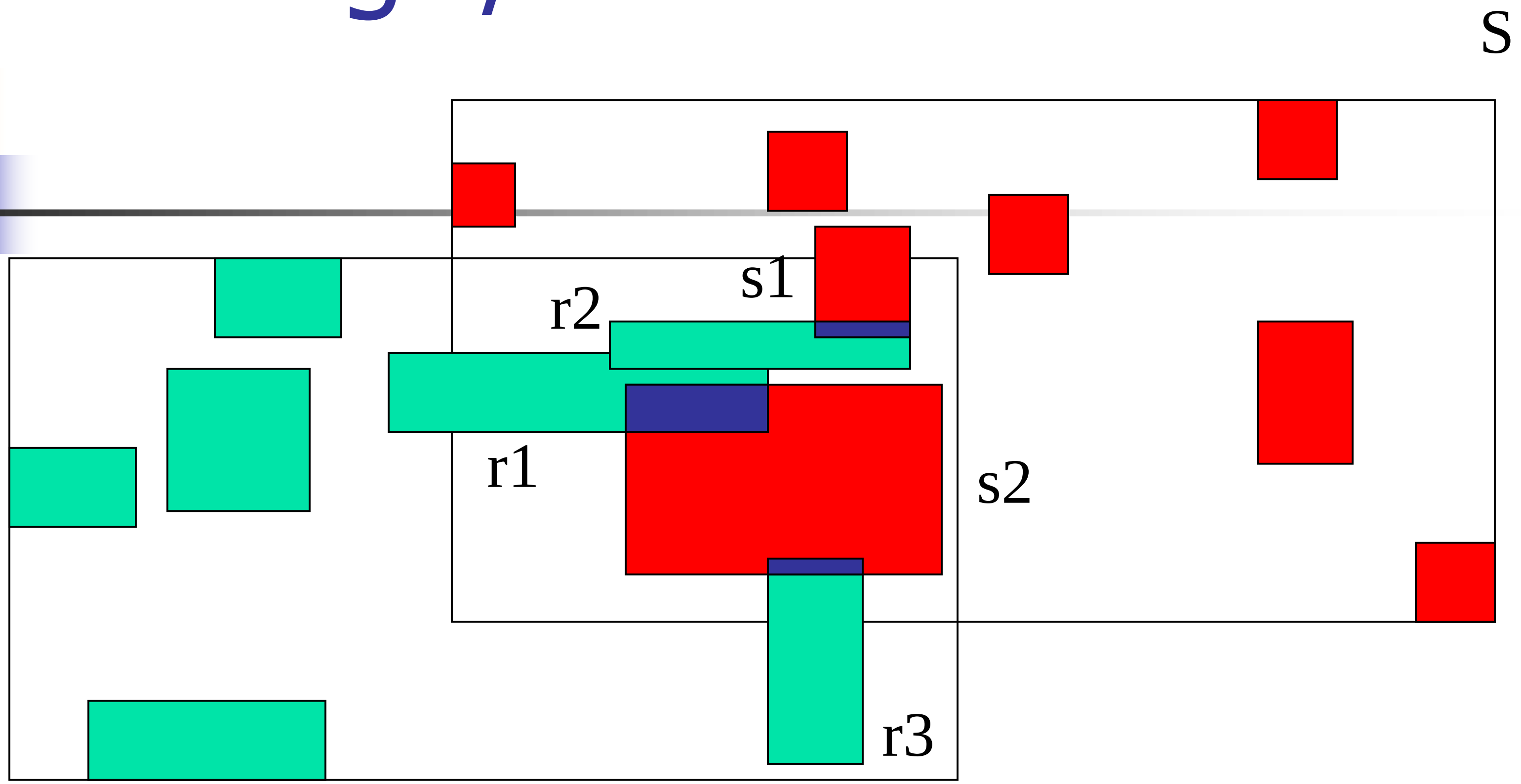


Read schedule is r1, s2, r2, s1, s2, r3
Subtree s2 is examined twice

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

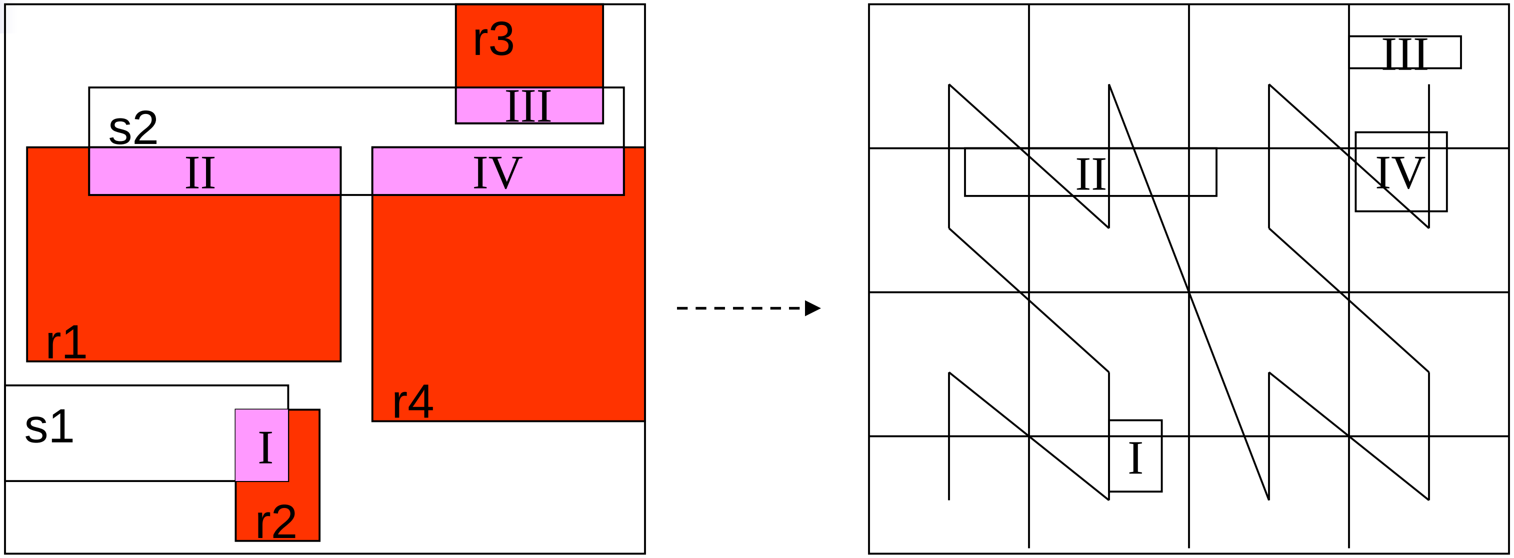


After computing join(r1,s2),
degree(r1) = 0
degree(s2) = 1
So, examine s2 next
Read schedule = r1, s2, r3, r2, s1
Subtree s2 examined only once

Local Z-Order

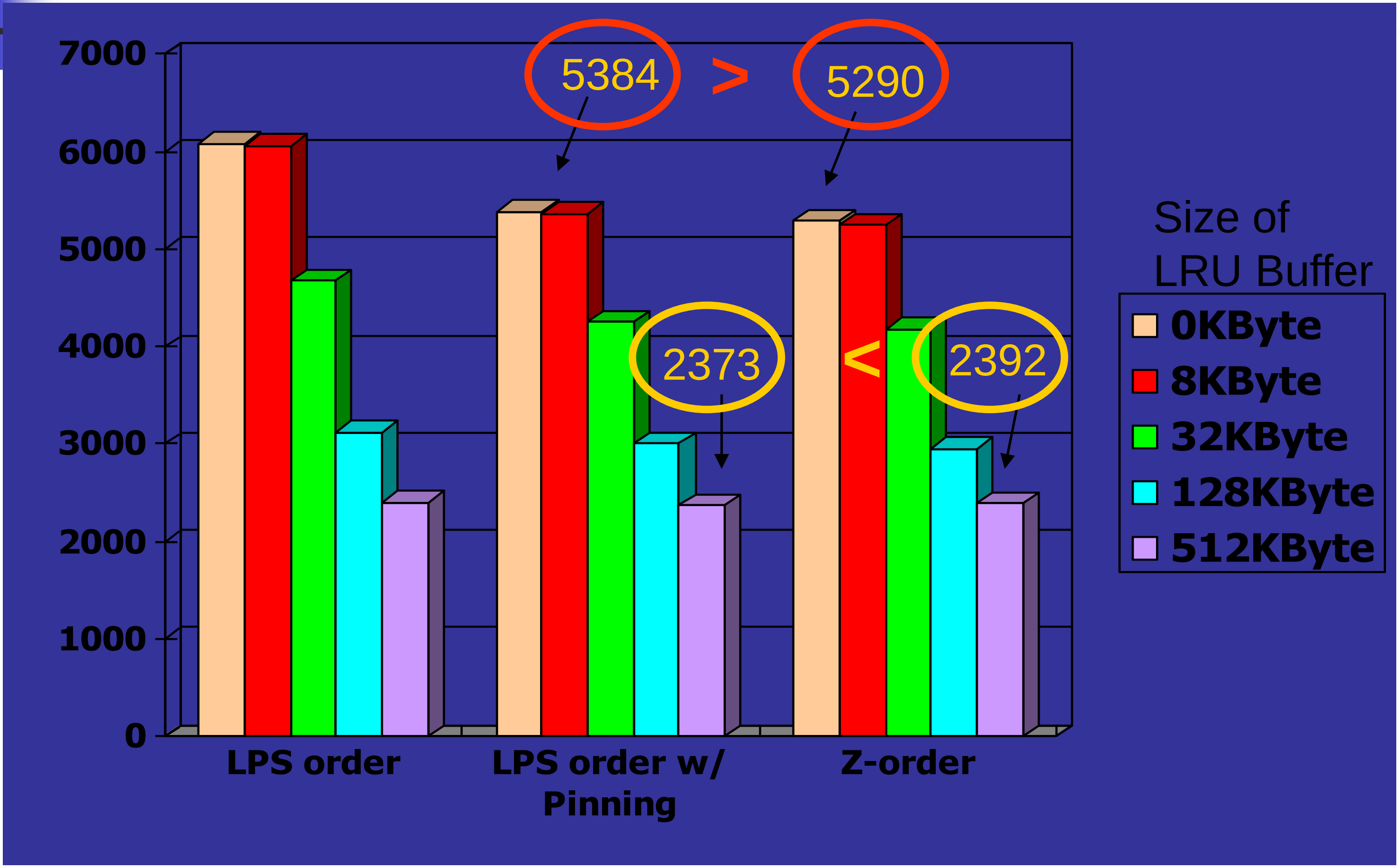
- Idea:
  1. Compute the intersections between each rectangle of the one node and all rectangles of the other node
  2. Sort the rectangles according to the Z-ordering of their centers
  3. Use this ordering to fetch pages

Local Z-ordering



Read schedule:
<s1,r2,r1,s2,r4,r3>

Number of Disk Access



LPS order
LPS order w/ Pinning
Z-order
>
<
Size of LRU Buffer
0KByte
8KByte
32KByte
128KByte
512KByte