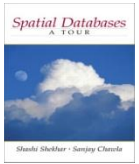
Ch. 5: Query Processing and Optimization

5.1 Evaluation of Spatial Operations
5.2 Query Optimization
5.3 Analysis of Spatial Index Structures
5.4 Distributed Spatial Database Systems
5.5 Parallel Spatial Database Systems
5.6 Summary



Types of queries

- Point Query- Name a highlighted city on a digital map.
  - Return one spatial object out of a table
- Range Query- List all countries crossed by of the river Amazon.
  - Returns several objects within a spatial region from a table
- Nearest Neighbor: Find the city closest to Mount Everest.
  - Return one spatial object from a collection
- Spatial Join: List all pairs of overlapping rivers and countries.
  - Return pairs from 2 tables satisfying a spatial predicate

R-tree query processing:Filter-Refining

- Processing a spatial query Q
  - Filter step : find a superset S of object in answer to Q
    - Using approximate of spatial data type and operator
  - Refinement step : find exact answer to Q reusing a GIS to process S
    - Using exact spatial data type and operation

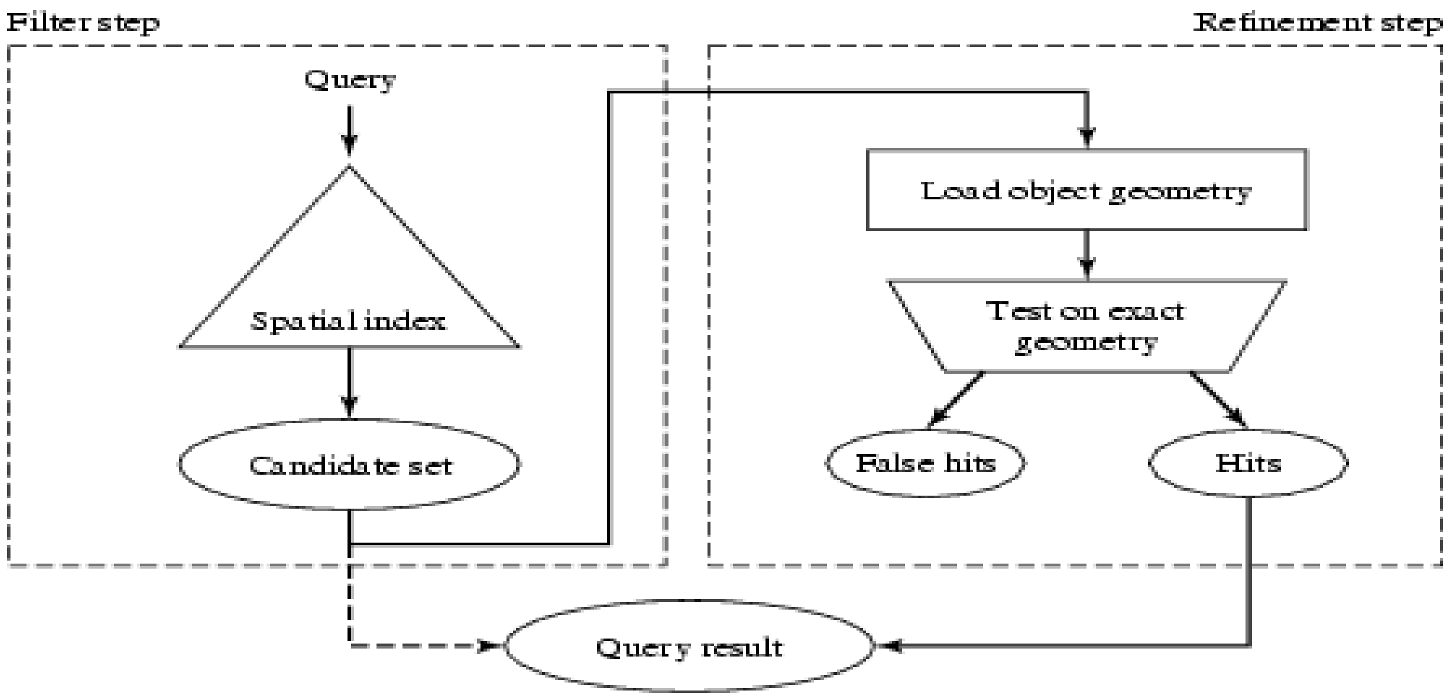


Fig 5.1

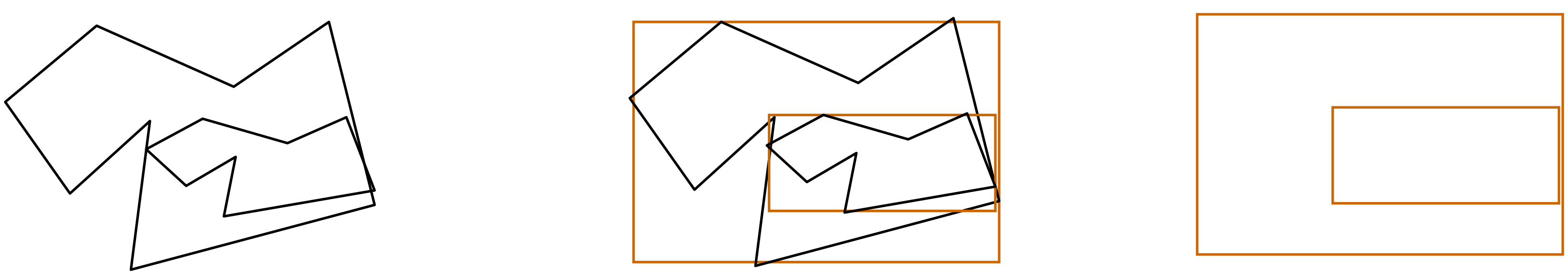
Approximate Spatial Data types

- Approximating spatial data types
  - Minimum orthogonal bounding rectangle (MOBR or MBR)
    - approximates line string, polygon, …
    - See Examples below (Bblack rectangle are MBRs for red objects)
  - MBRs are used by spatial indexes, e.g. R-tree
  - Algorithms for spatial operations MBRs are simple

- Q? Which OGIS operation (Table 3.9, pp. 66) returns MBRs ?

Approximate Spatial Operations

- Approximating spatial operations
  - SDBMS processes MBRs for refinement step
  - Overlap predicate used to approximate topological operations
  - Example: inside(A, B) replaced by
    - overlap(MBR(A), MBR(B)) in filter step
    - See picture below - Let A be outer polygon and B be the inner one
    - inside(A, B) is true only if overlap(MBR(A), MBR(B))
    - However overlap is only a filter for inside predicate needing refinement later



R-trees:Range search

pseudocode:

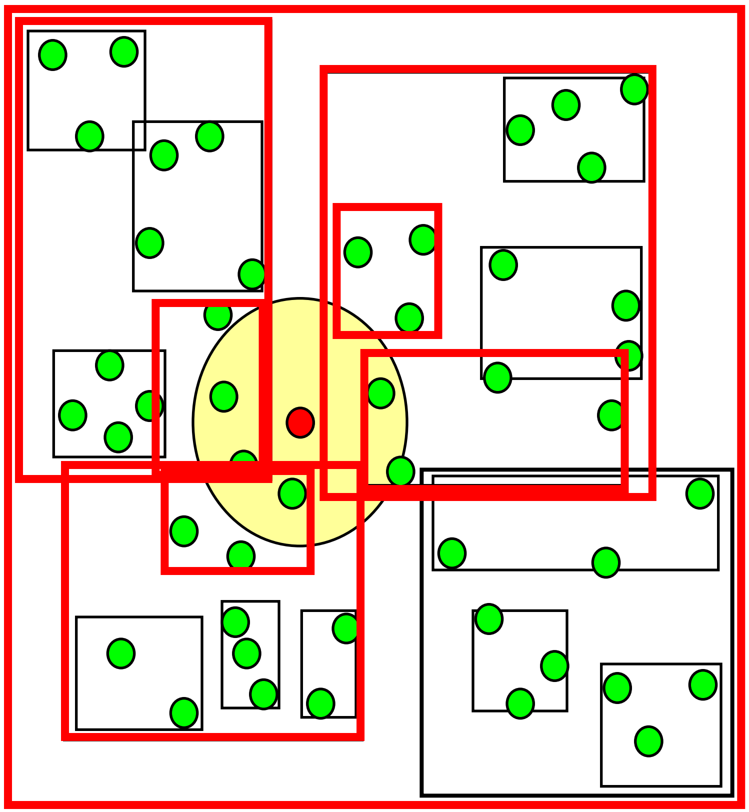
check the root

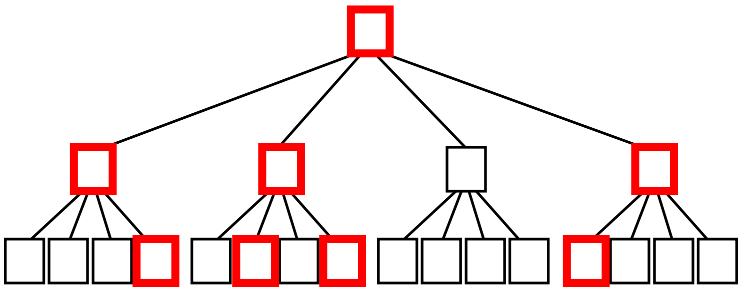
for each branch,

if its MBR intersects the query rectangle

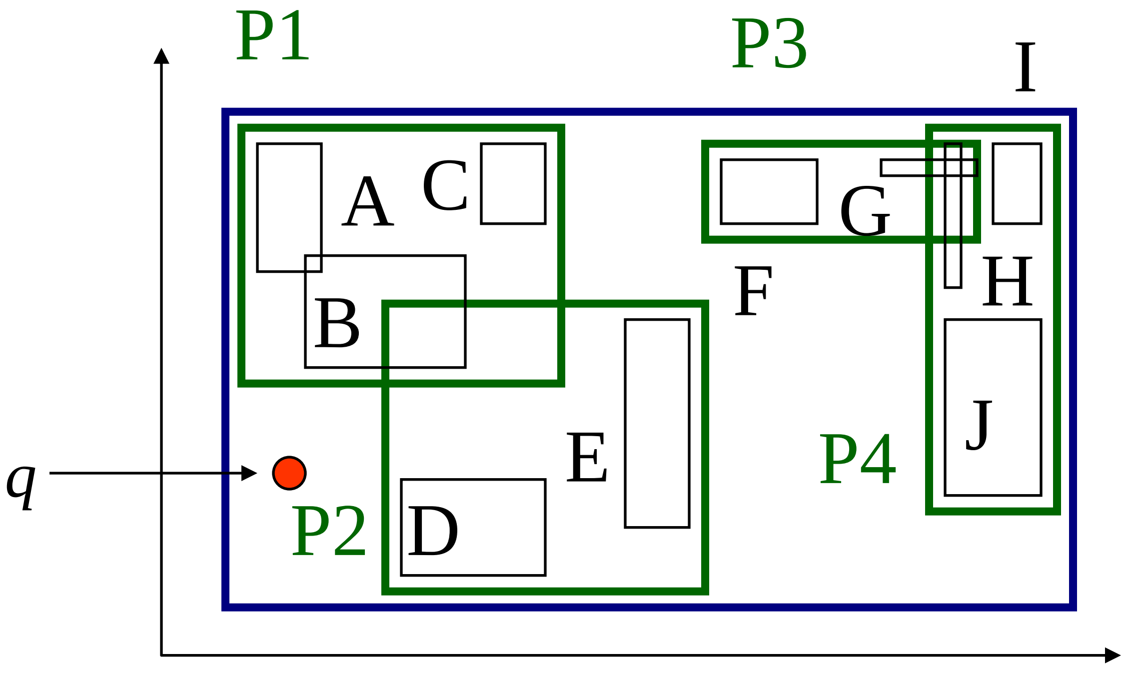
apply range-search (or print out, if this is a leaf)

Example (DFS searching)



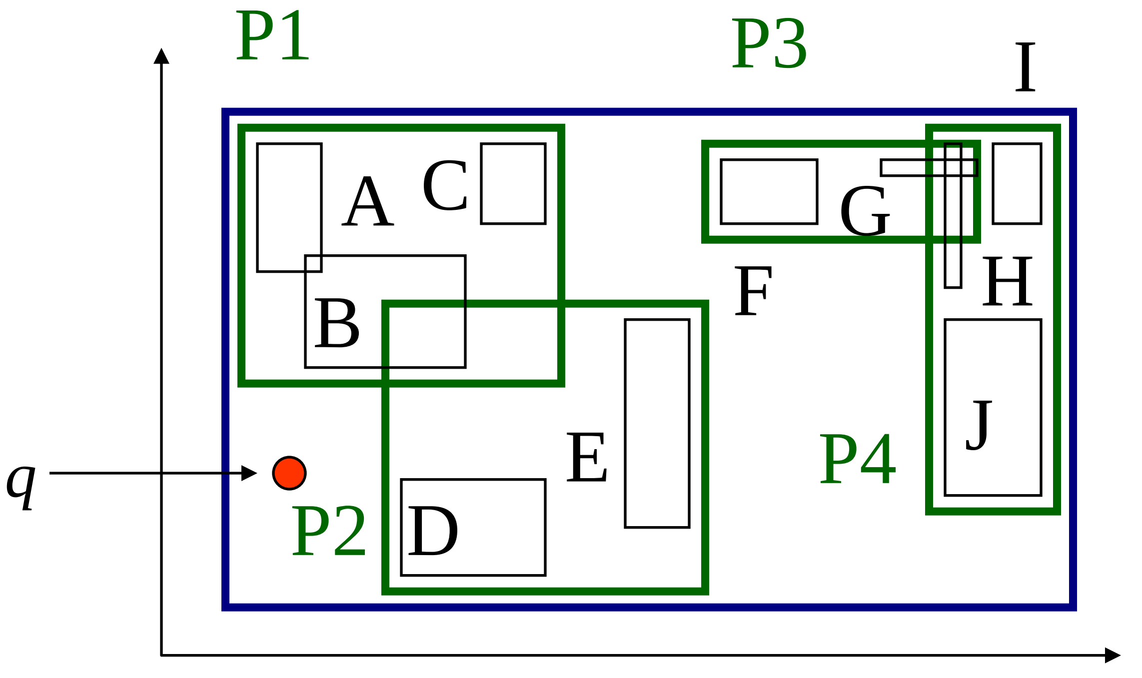


R-trees: NN search



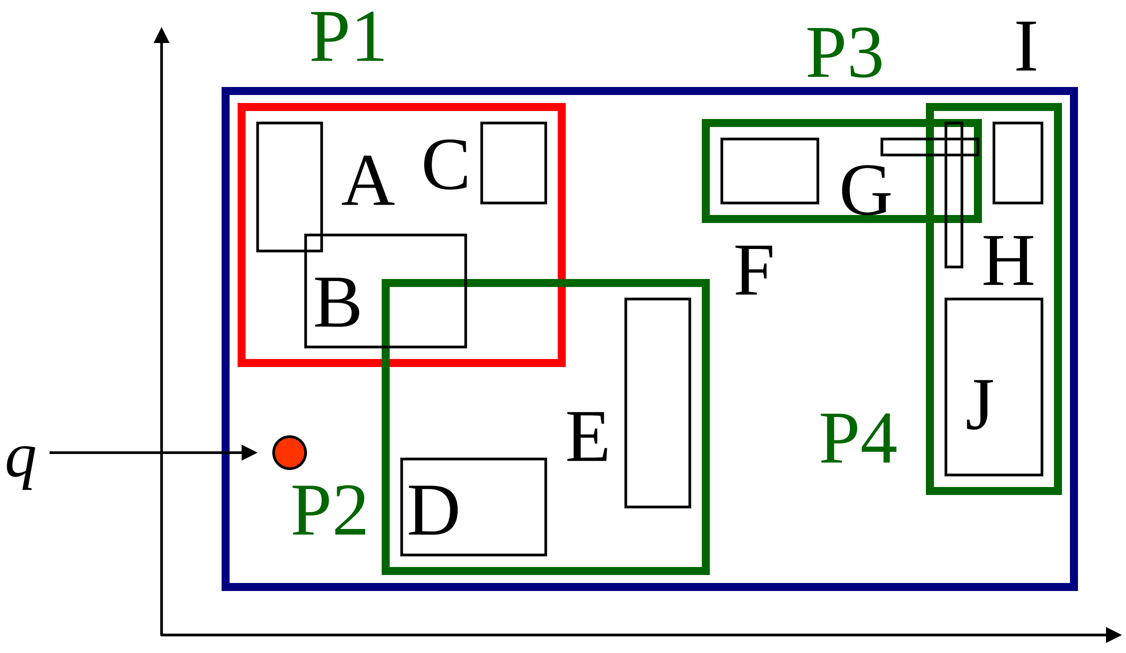
R-trees: NN search

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



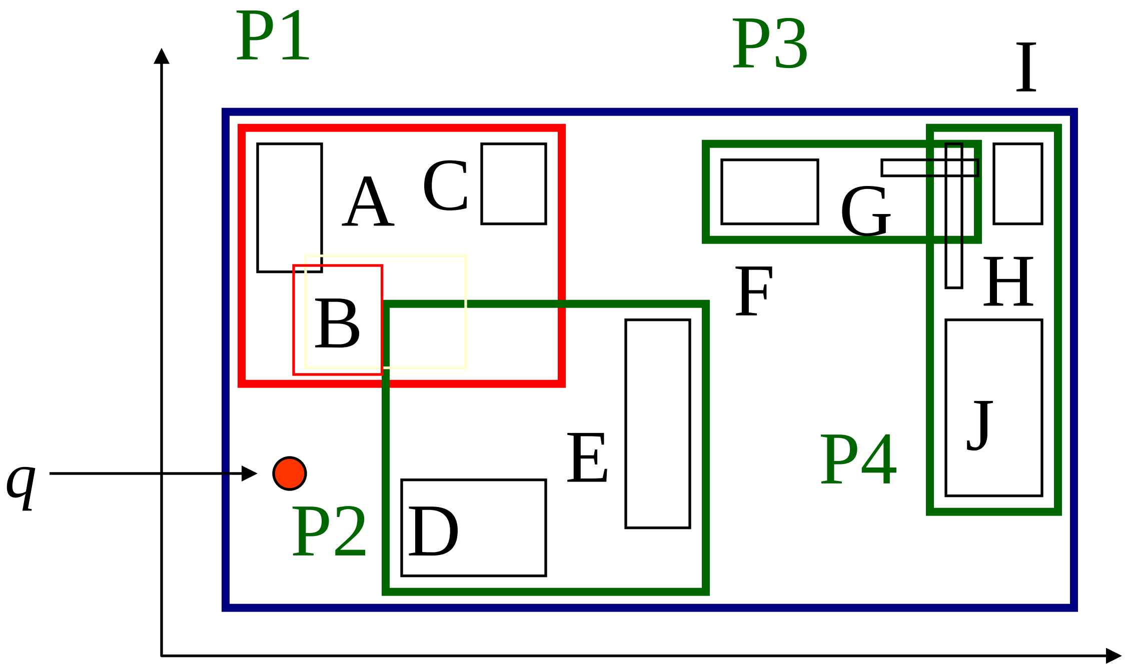
R-trees: NN search (simple algorithm)

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



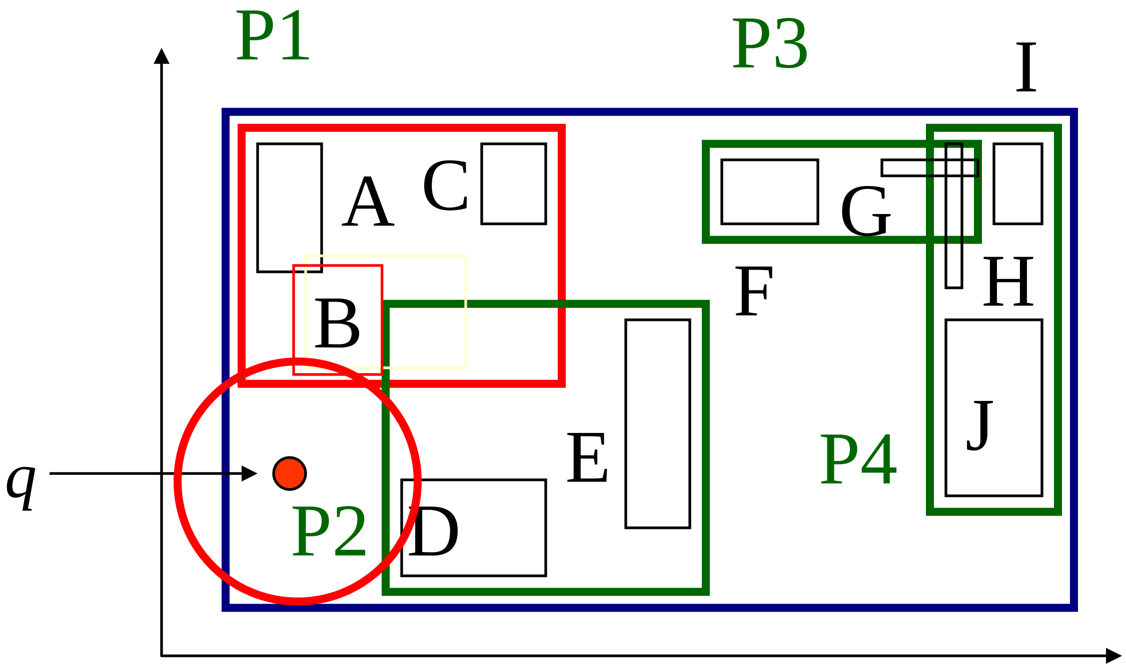
R-trees: NN search (simple algorithm)

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



R-trees: NN search (simple algorithm)

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

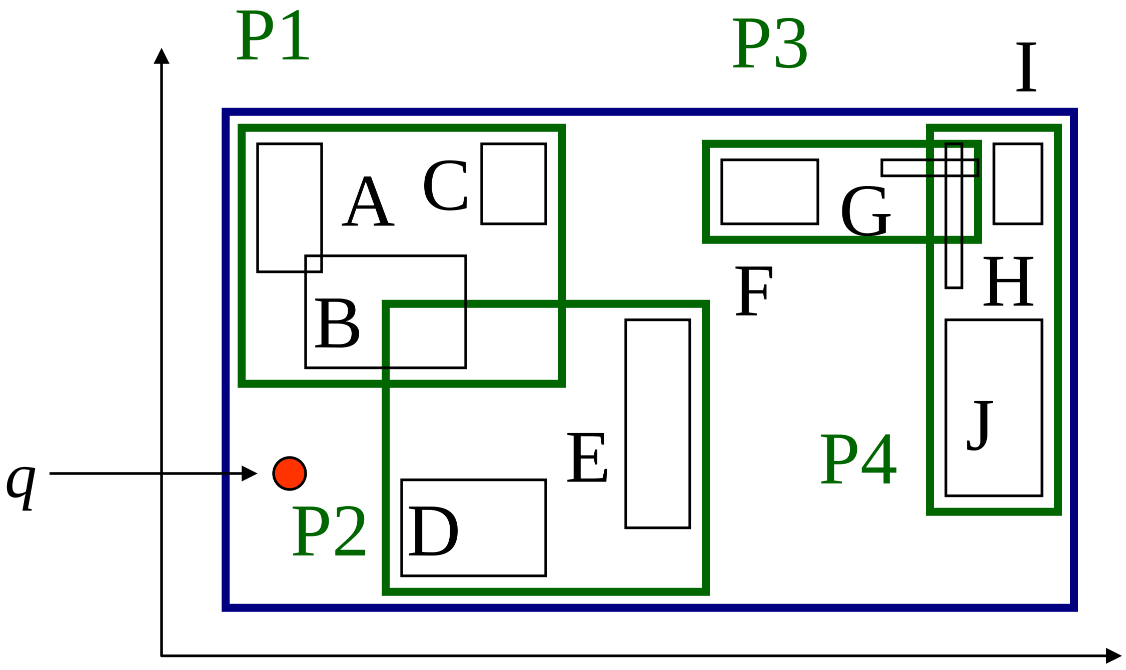


R-trees: NN search (better algorithm)

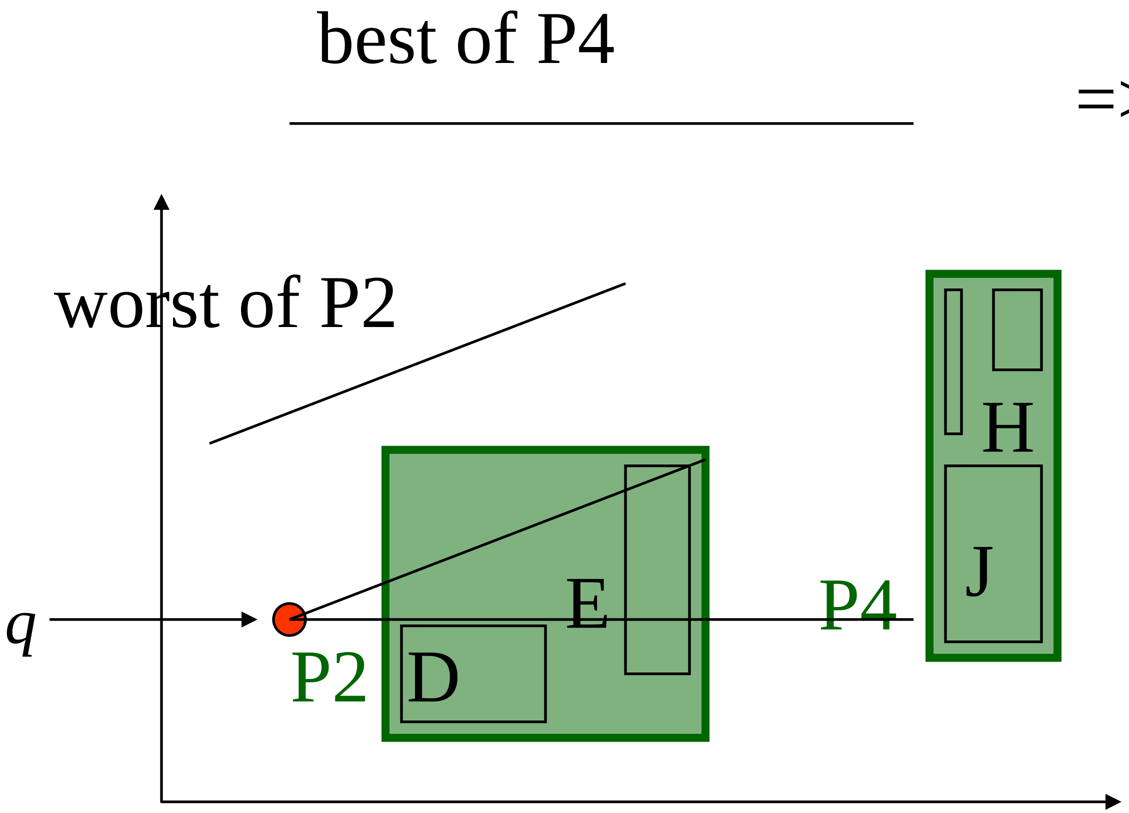
- Priority queue, with promising MBRs, and their best and worst-case distance
- Main idea: Every face of any MBR contains at least one point of an actual spatial object!

R-trees: NN search (better algorithm)

consider only P2 and P4, for illustration



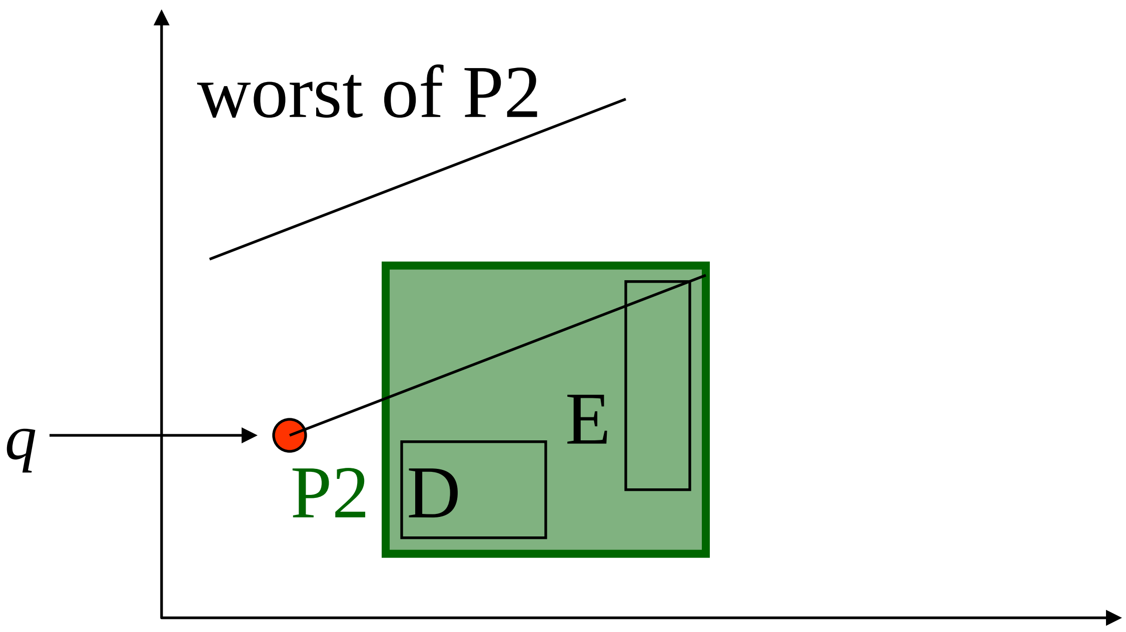
R-trees: NN search (better algorithm)



=> P4 is useless for 1-nn

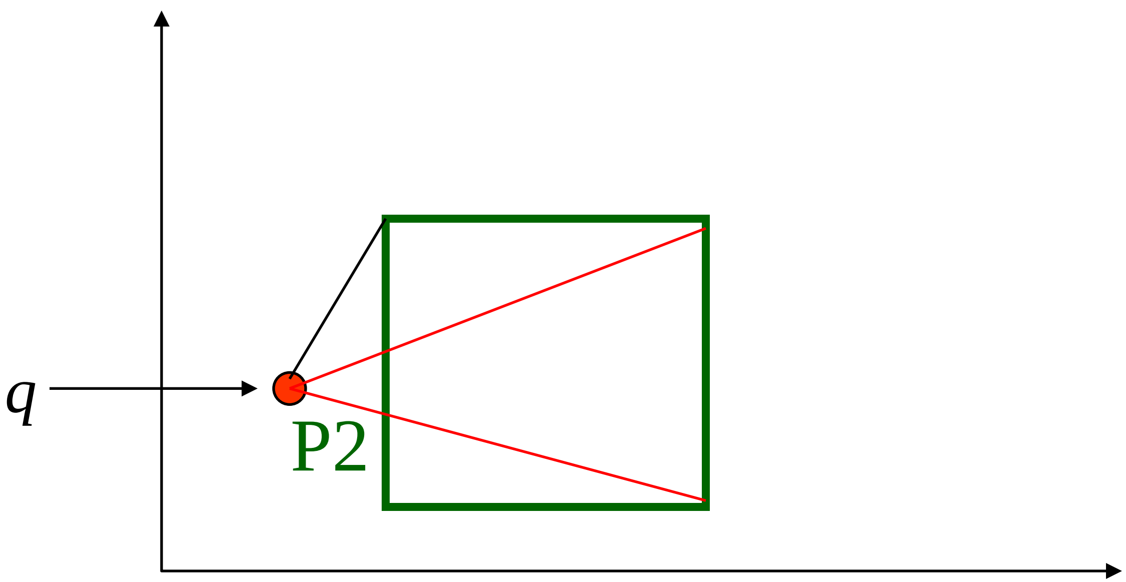
R-trees: NN search (better algorithm)

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



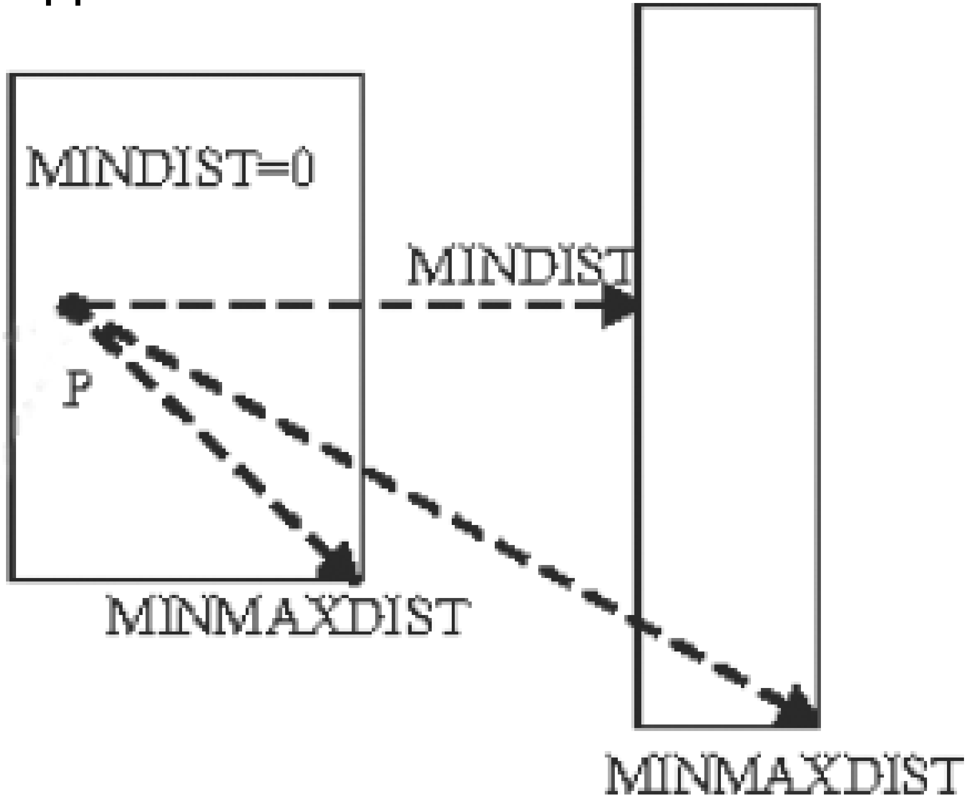
R-trees: NN search (better algorithm)

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



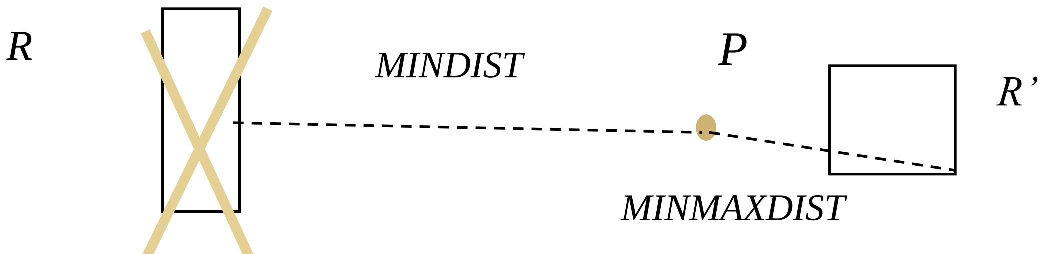
MINDIST, MINMAXDIST

- MINDIST(P, R) = min possible distance of P from R
- MINMAXDIST = the min of the max possible distances from P to a vertex of R
- Lower and an upper bound on the actual distance of R from P

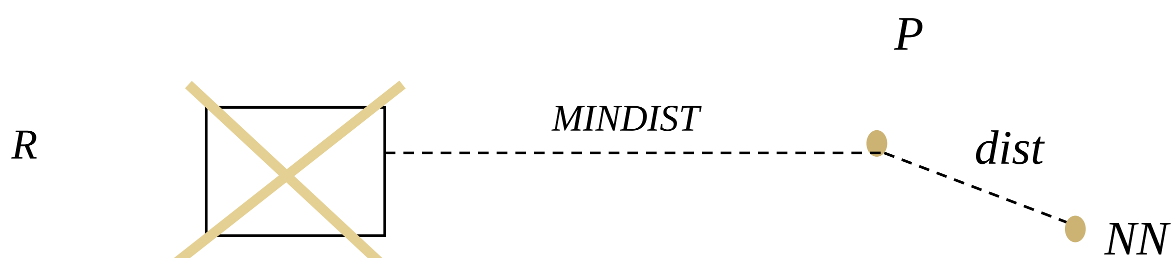


Pruning with MINDIST and MINMAXDIST

Downward pruning: MINDIST(P, R) > MINMAXDIST(P, R') => discard M



Upward pruning: MINDIST(P, R) > Dist(P, currNN) => discard visit to R



Order of searching

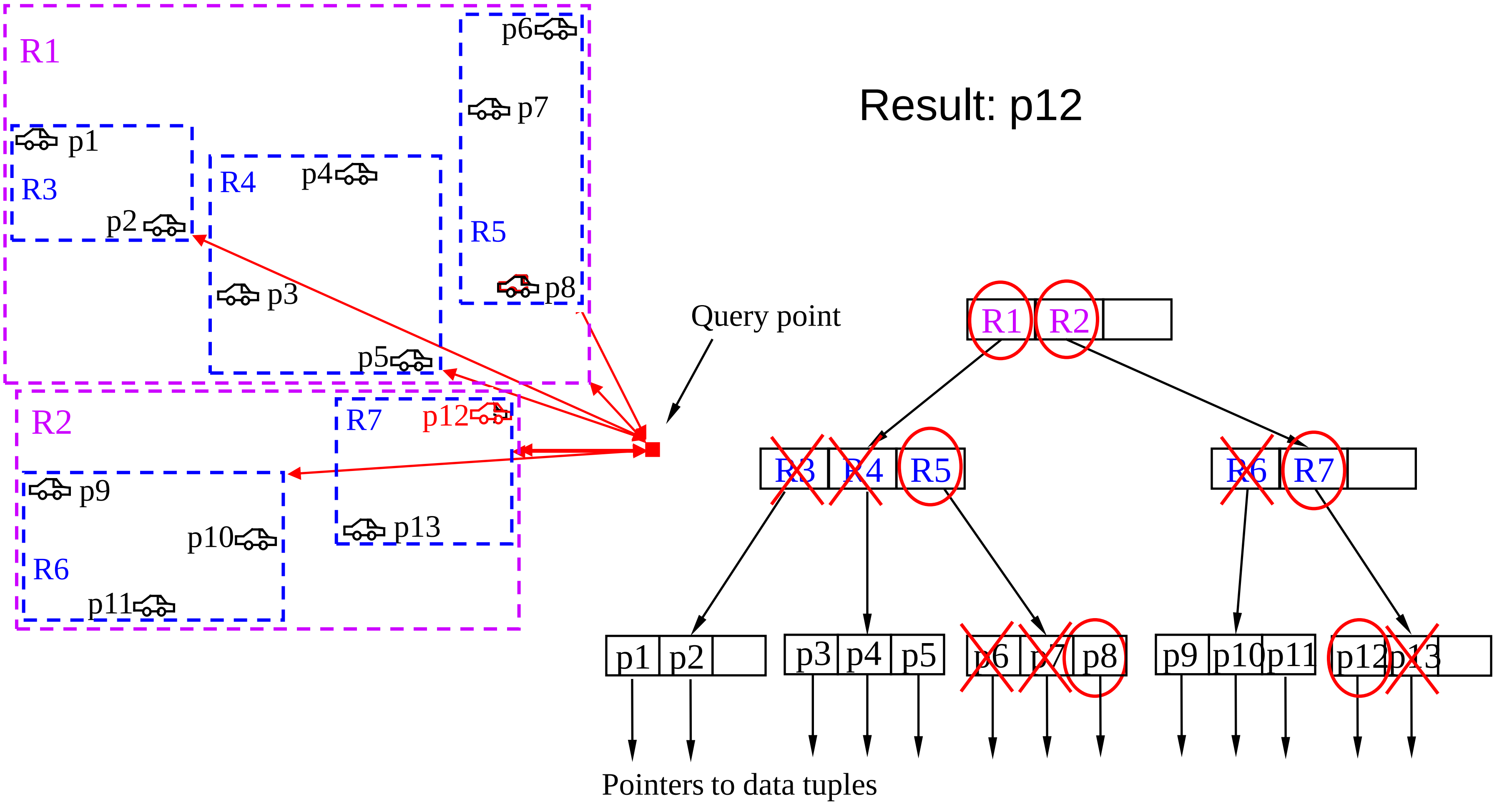
- Depth first order
  - Inspect children in MINDIST order
  - For each node in the tree keep a list of nodes to be visited
  - Prune some of these nodes in the list
  - Continue until the lists are empty

Branch and bound NN-search algorithm

```
Procedure NNSearch(Node, Point, Nearest)
  if Node.type == LEAF
      for i=1 to Node.count
          dist = objectDIST(Point, Node.branch[i].rect)
          if dist < Nearest.dist
              Nearest.dist = dist
              Nearest.rect = Node.branch[i].rect
          endif
      endfor
  else
      genBranchList(branchList)
      sortBranchList(branchList)
      last = pruneBranchList(Node, Point, Nearest, branchList)
      for i = 1 to last
          newNode = Node.branch[branchList[i]]
          NNSearch(newNode, Point, Nearest)
          last = pruneBranchList(Node, Point, Nearest, branchList)
      endfor
  endif
  end
```

NN example

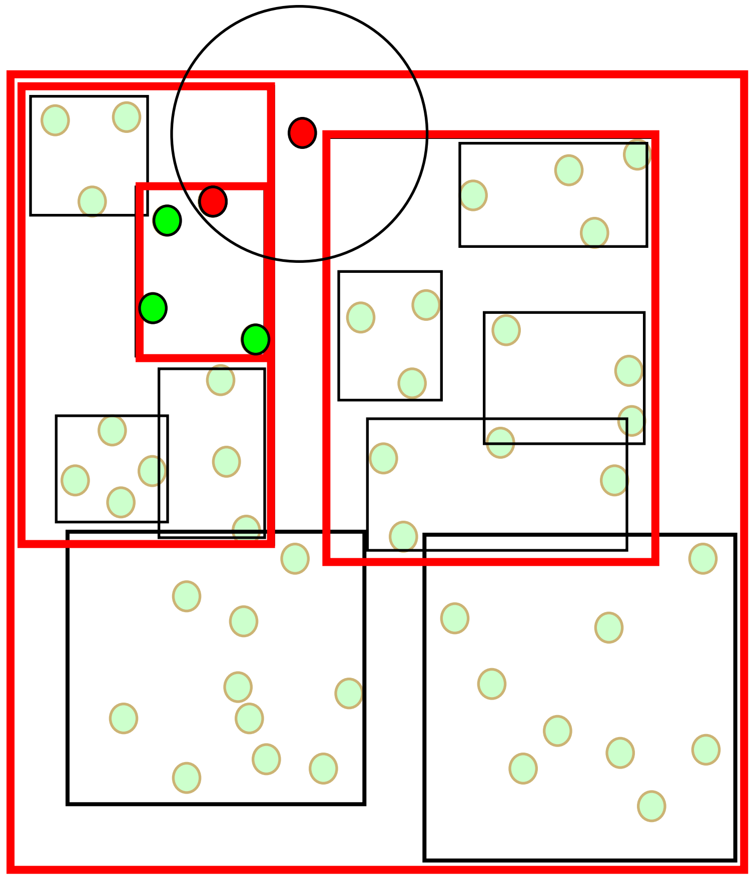
Result: p12



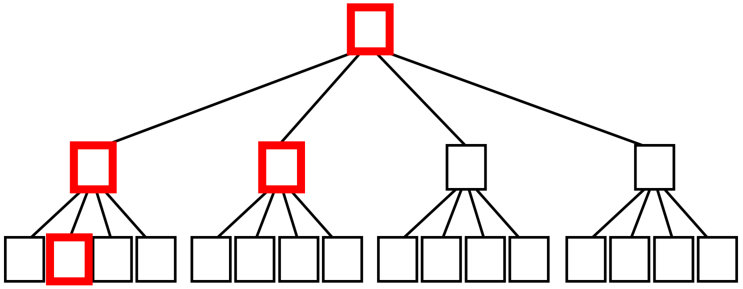
Optimal Strategy for NN search

- Global order
  - 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

Optimal NN: example

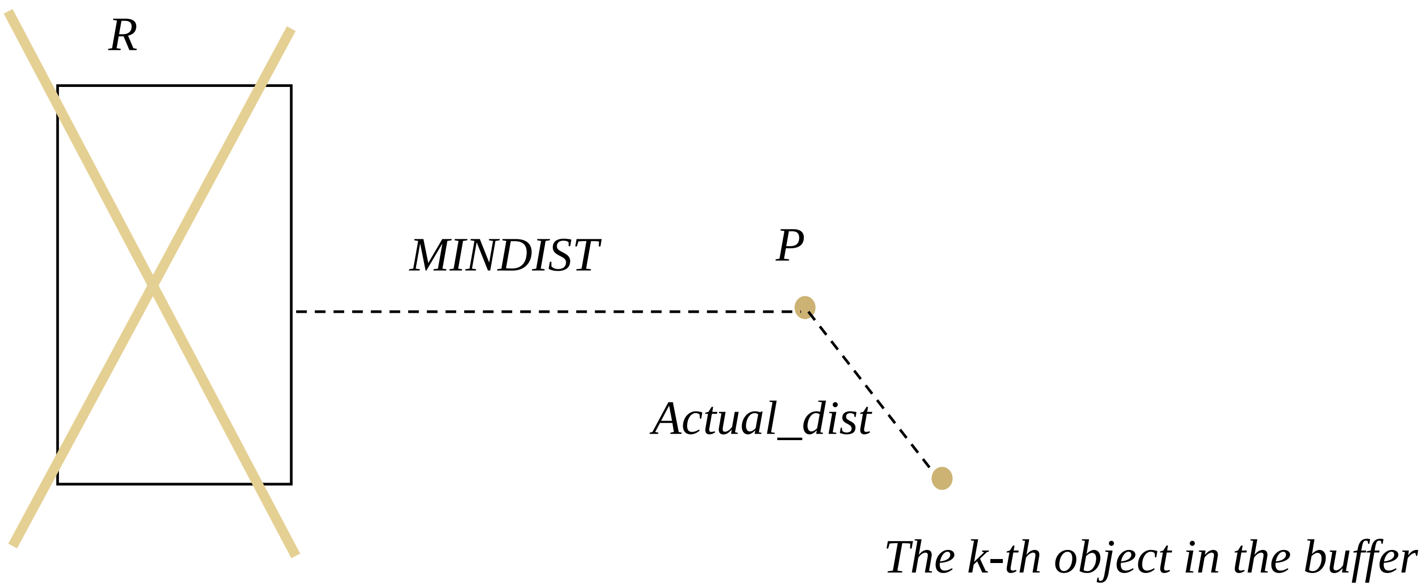


4 page accesses



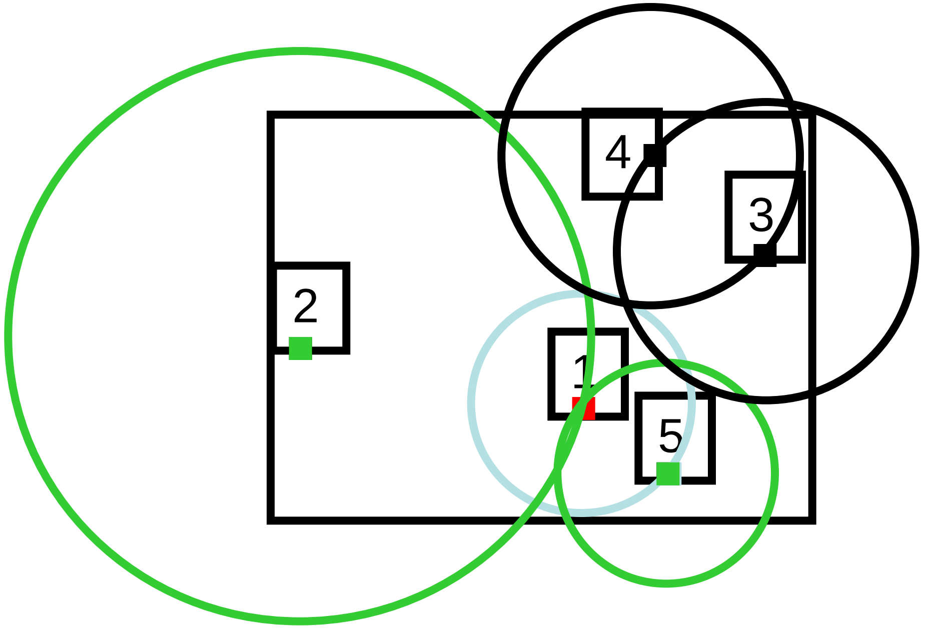
Generalize to k-NN

- Keep a sorted buffer of at most *k* current nearest neighbors
- Pruning is done according to the distance of the furthest nearest neighbor in this buffer
- Example:



RNN Queries

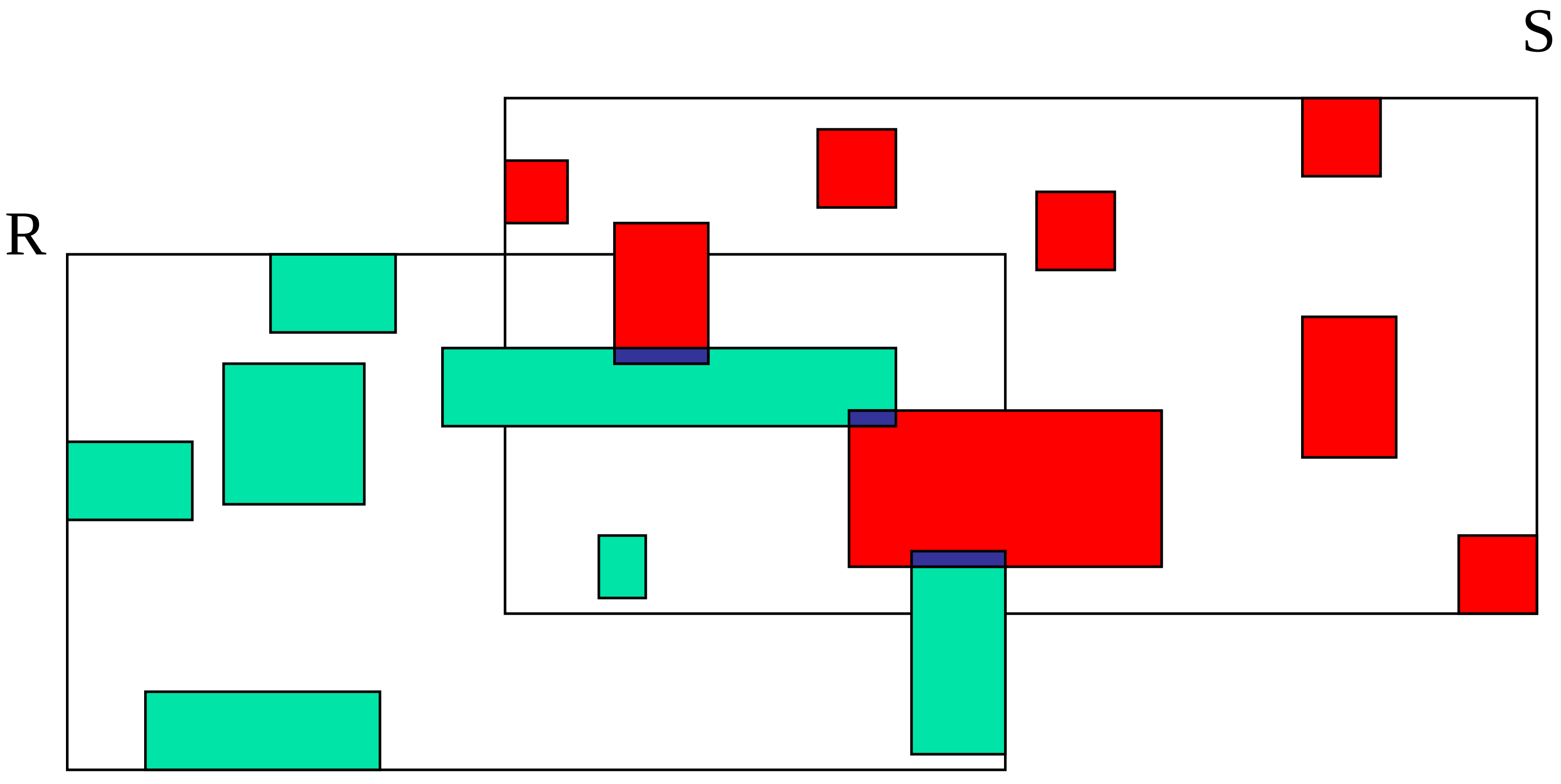
- *Nearest neighbor (NN) query* – find an object(s) that is closest to a query point.
- *Reverse Nearest Neighbor (RNN) query* – find objects that have a query point as their nearest neighbor



Spatial Joins

- Recall Spatial Join Example:
  - List all pairs of overlapping rivers and countries.
  - Return pairs from 2 tables satisfying a spatial predicate
- Naïve algorithm
  - Nested loop:
    - Test all possible pairs for spatial predicate
    - All rivers are paired with all countries

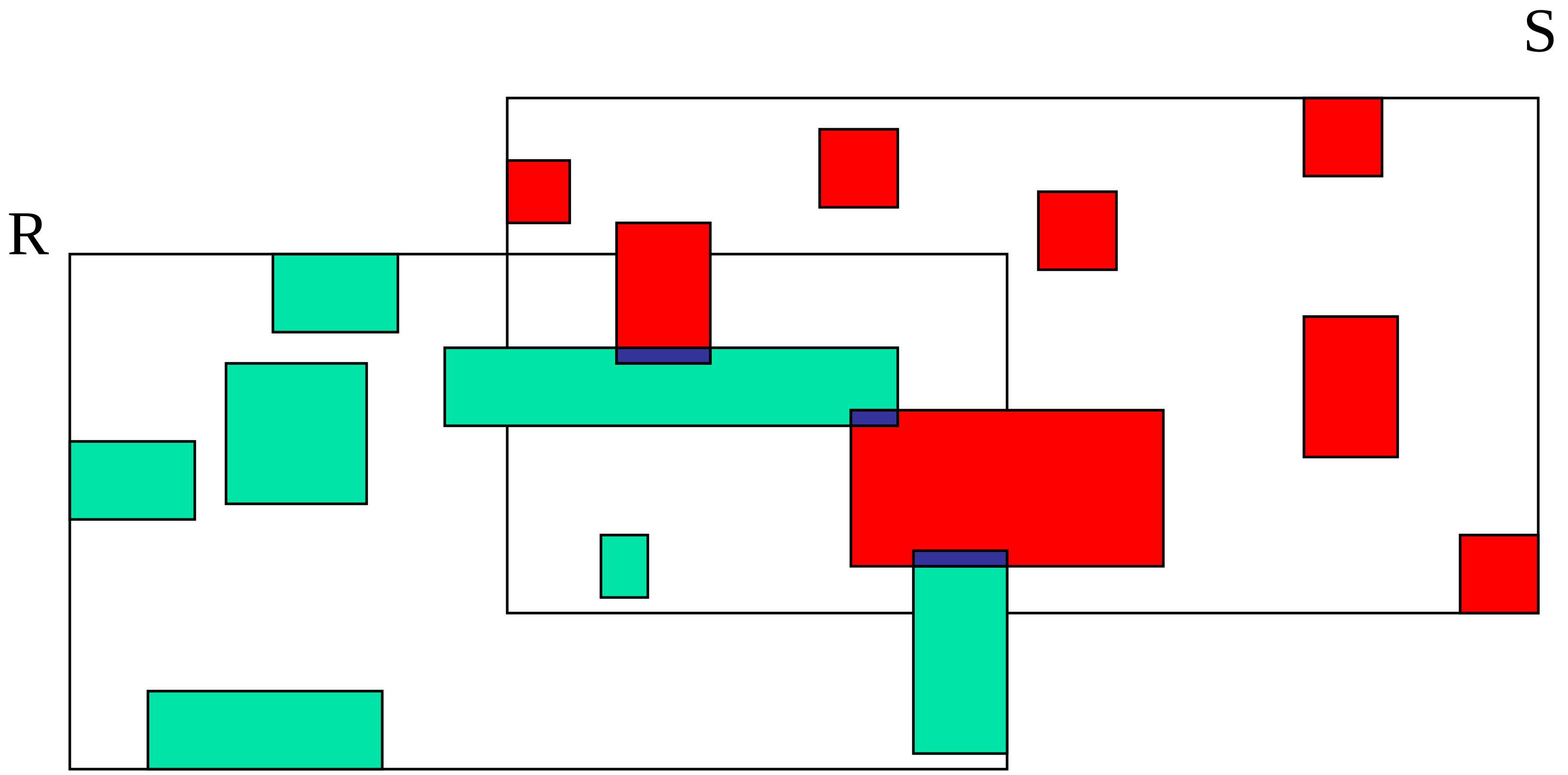
R-tree: Spatial Join



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

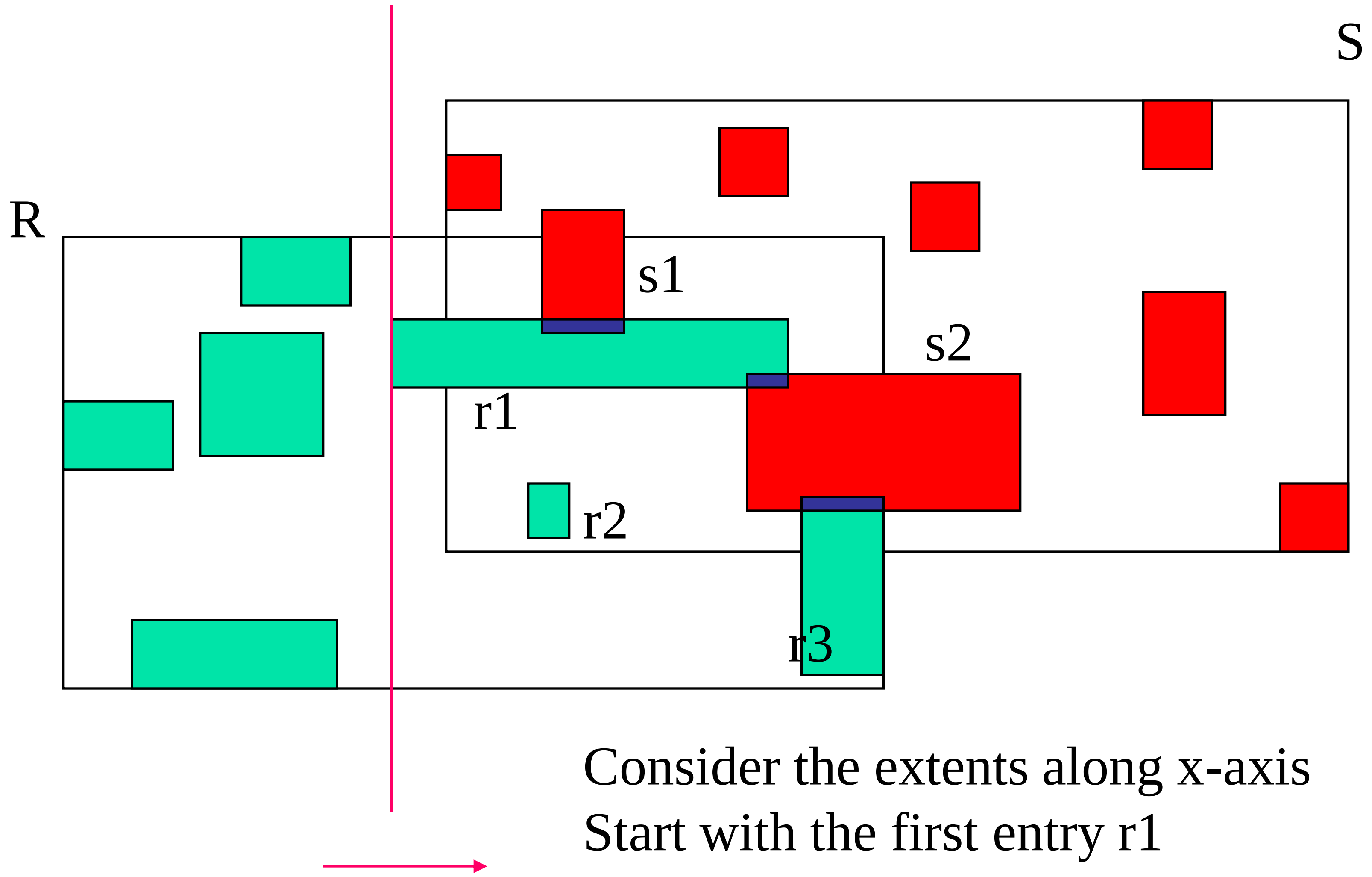
Reducing CPU bottleneck



Join2(R,S,IntersectedVol)

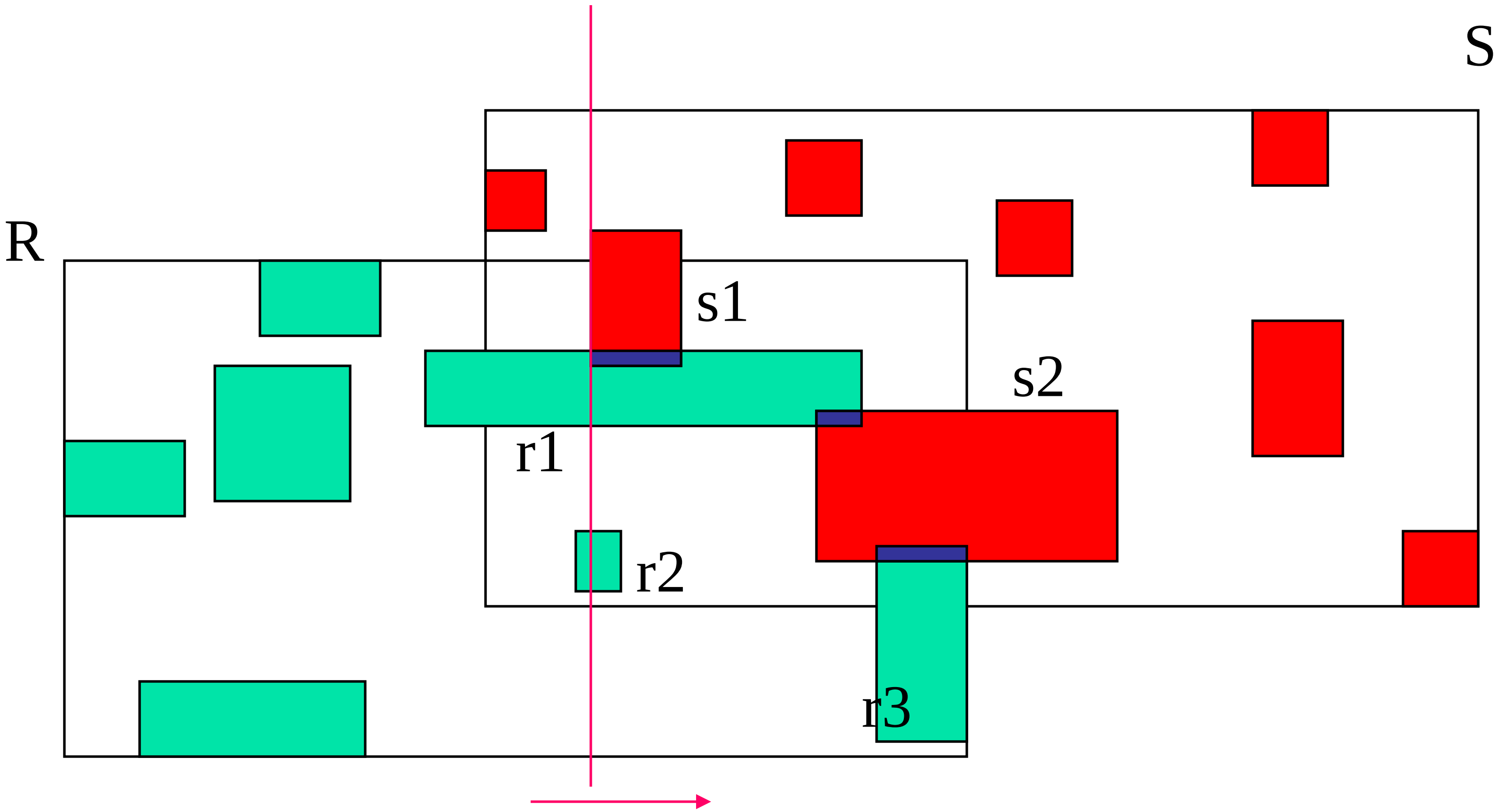
- Repeat
  - Find a pair of intersecting entries E in R and F in S that overlap with IntersectedVol
  - If R and S are leaf pages then add (E,F) to result-set
  - Else Join2(E,F,CommonEF)
- Until all pairs are examined
- 14+6 comparisons instead of 49
- In general, number of comparisons equals
  - size(R)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

Using Plane Sweep



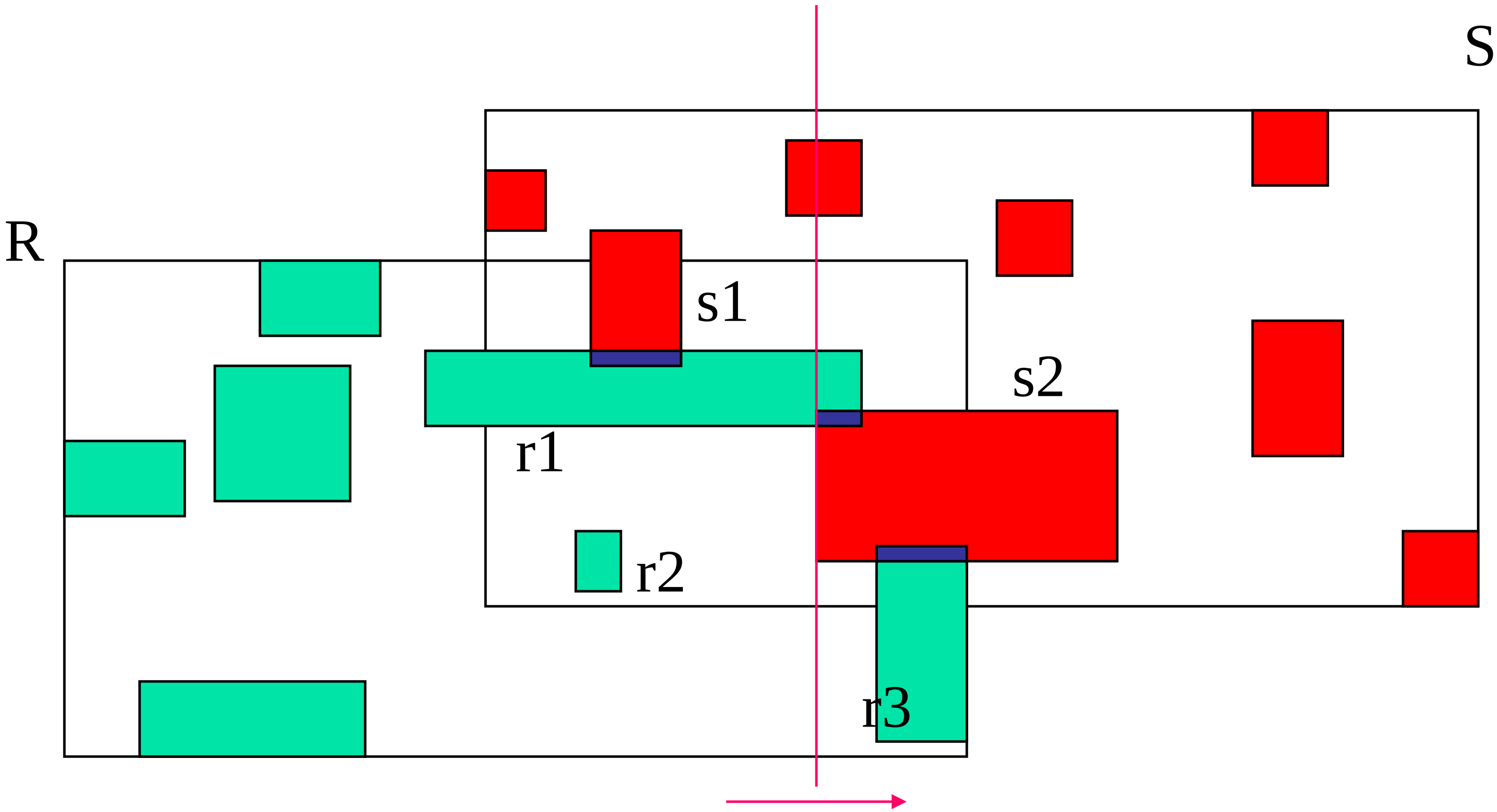
Consider the extents along x-axis
Start with the first entry r1
sweep a vertical line

Using Plane Sweep



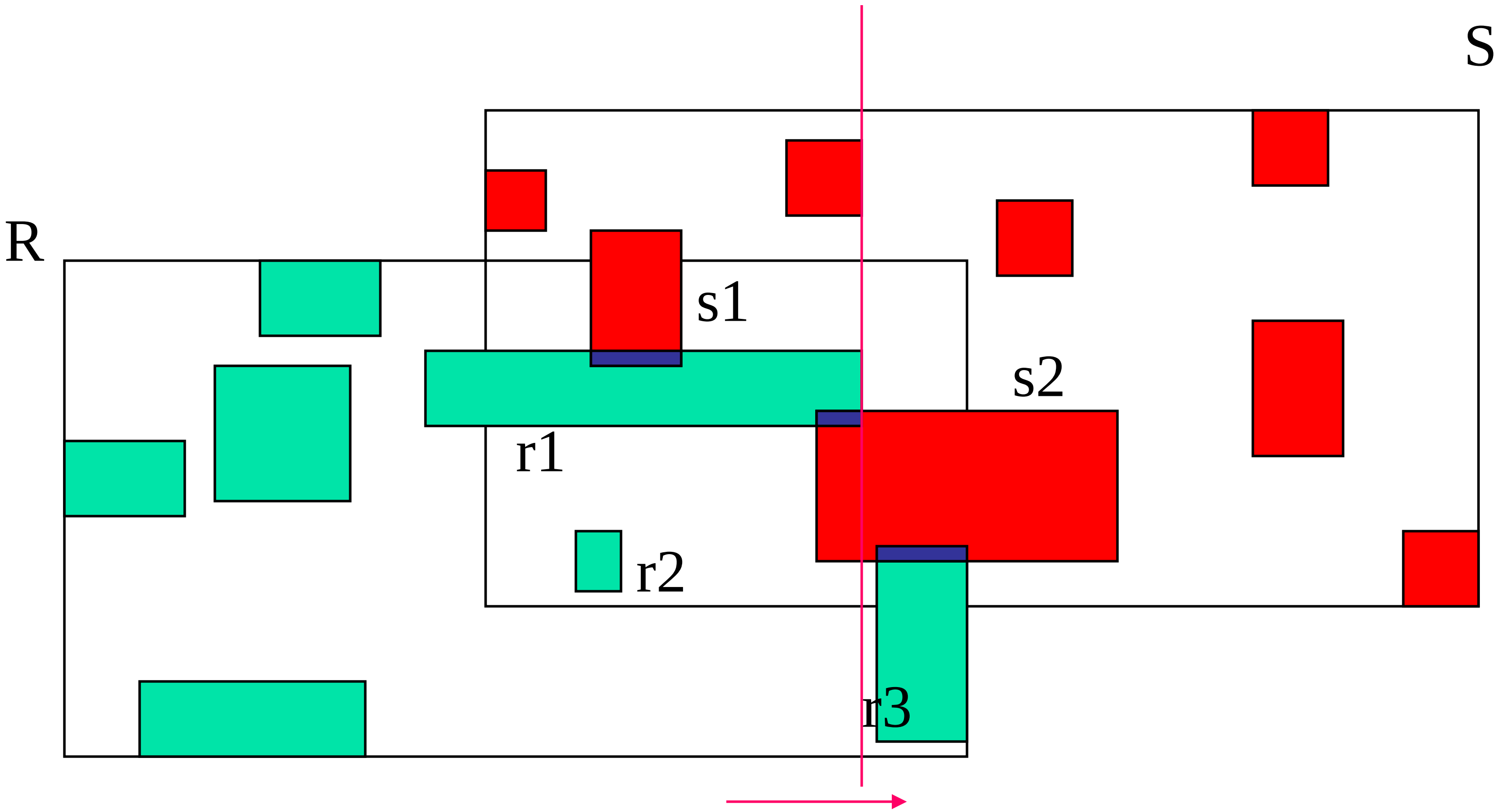
Check if (r1,s1) intersect along y-dimension
Add (r1,s1) to result set

Using Plane Sweep



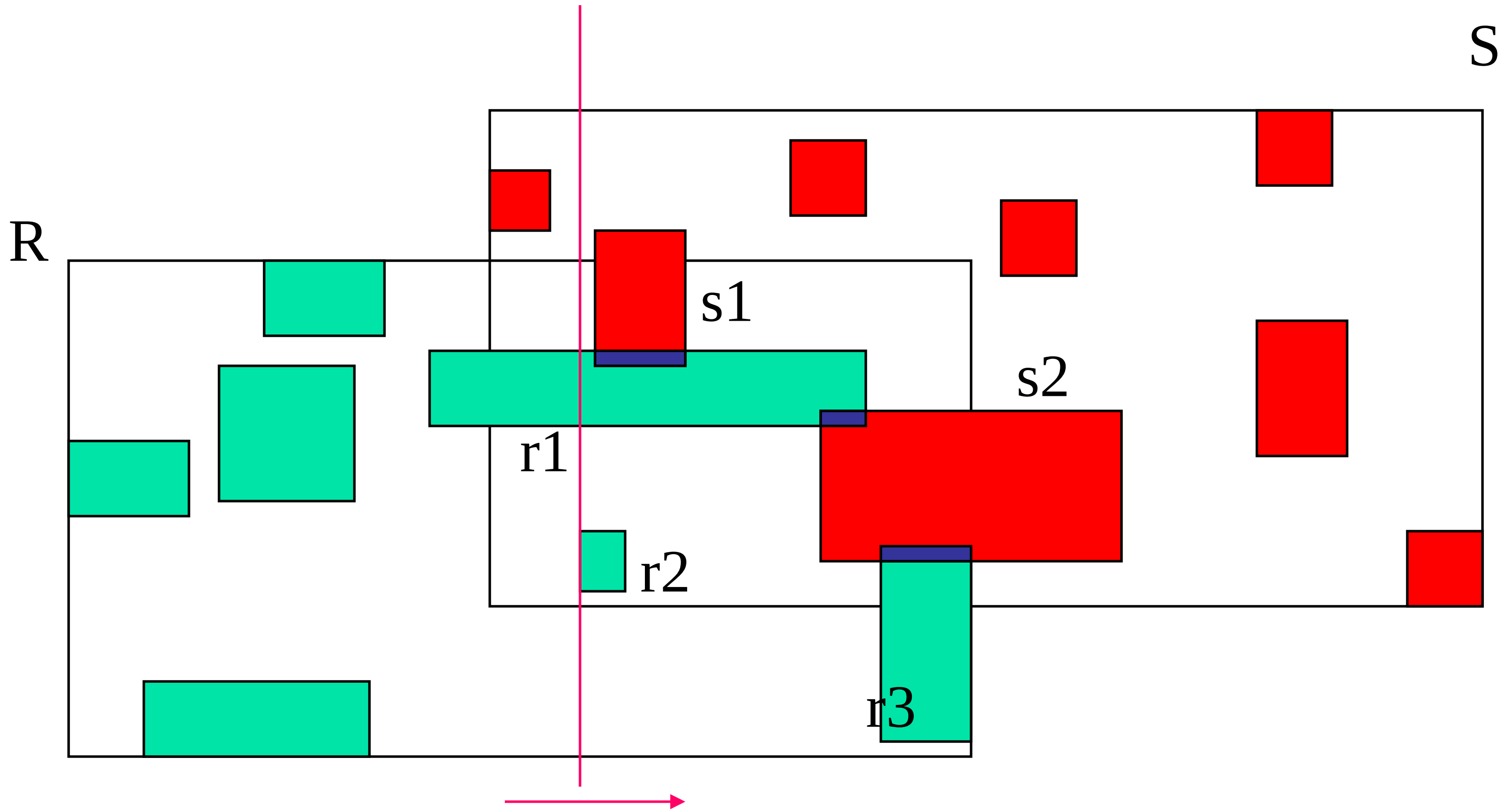
Check if (r1,s2) intersect along y-dimension
Add (r1,s2) to result set

Using Plane Sweep



Reached the end of r1
Start with next entry r2

Using Plane Sweep



Reposition sweep line and repeat …

R-trees: performance analysis

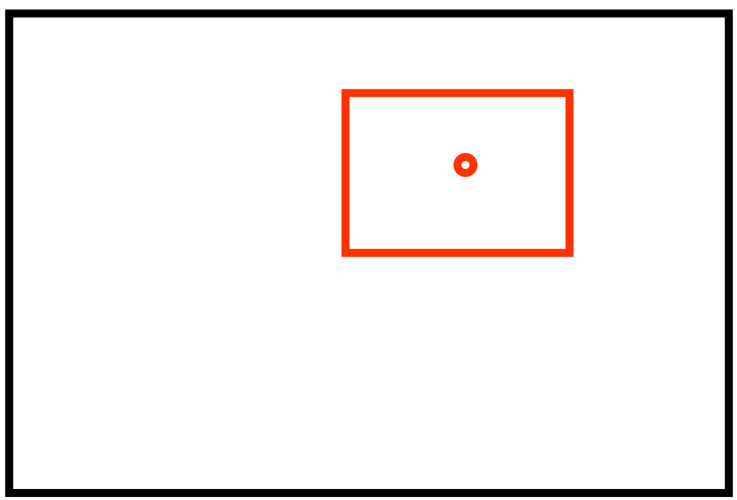
- How many disk (=node) accesses we'll need for
  - range
  - nn
  - spatial joins
- why does it matter?

R-trees: performance analysis

- A: because we can design split etc algorithms accordingly; also, do query-optimization
- motivating question: on, e.g., split, should we try to minimize the area (volume)? the perimeter? the overlap? or a weighted combination? why?

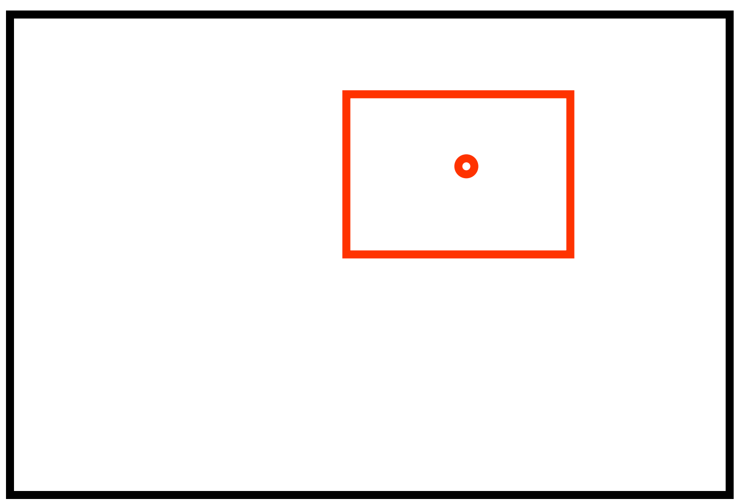
R-trees: performance analysis

- How many disk accesses for range queries?
  - query distribution wrt location?
  - "   "   wrt size?



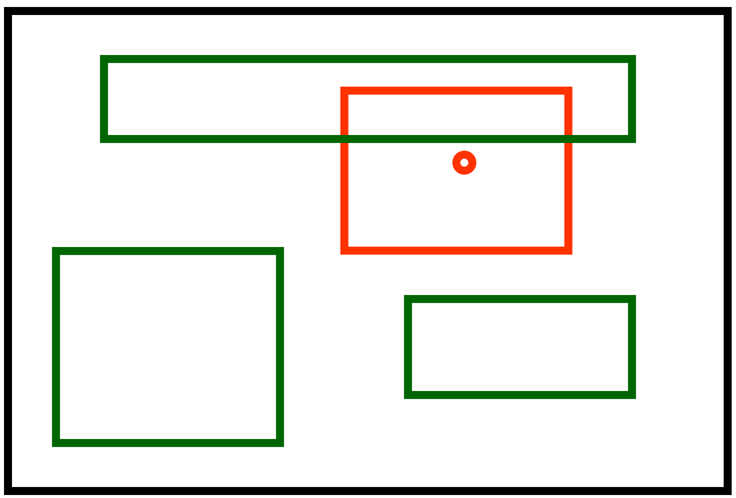
R-trees: performance analysis

- How many disk accesses for range queries?
  - query distribution wrt location? uniform; (biased)
  - "   " wrt size? uniform



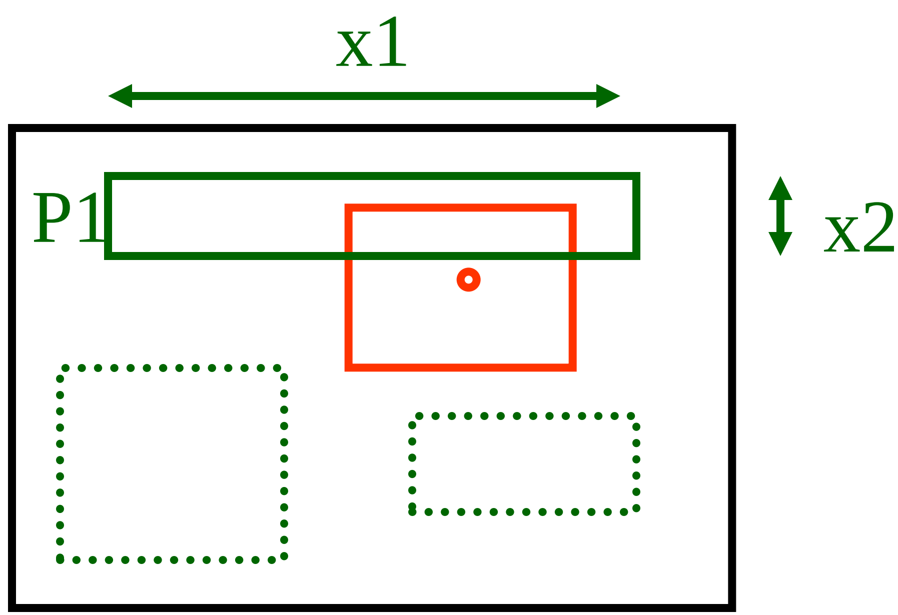
R-trees: performance analysis

- easier case: we know the positions of parent MBRs, eg:



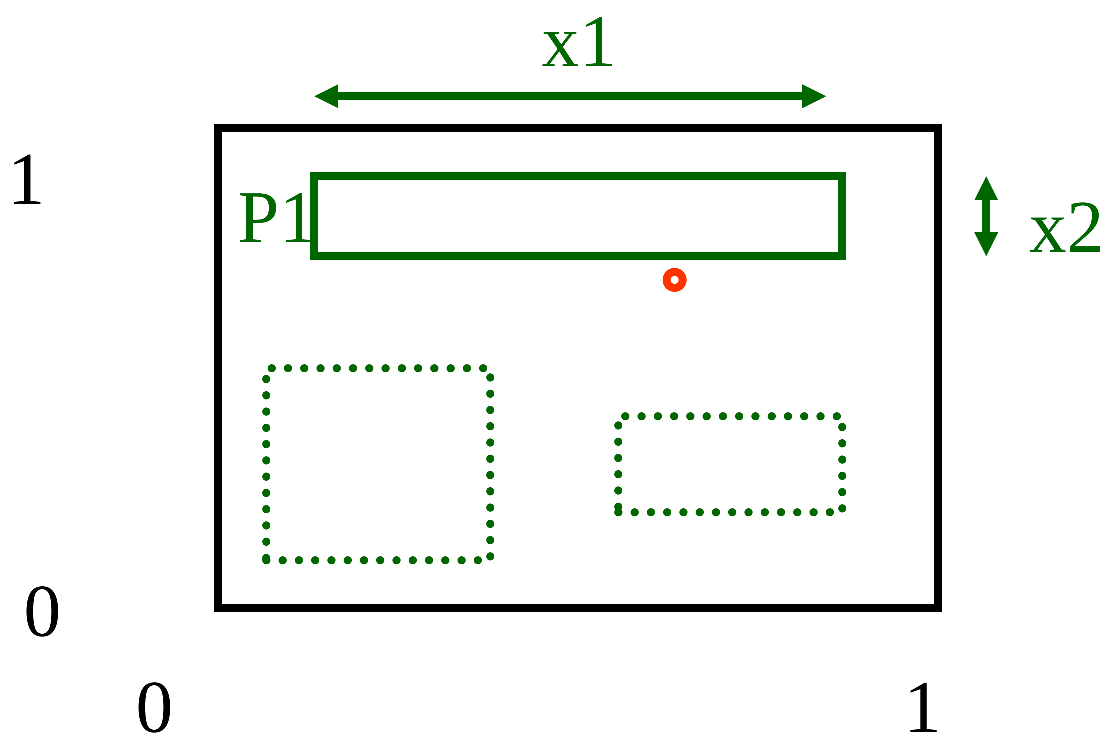
R-trees: performance analysis

- How many times will P1 be retrieved (unif. queries)?



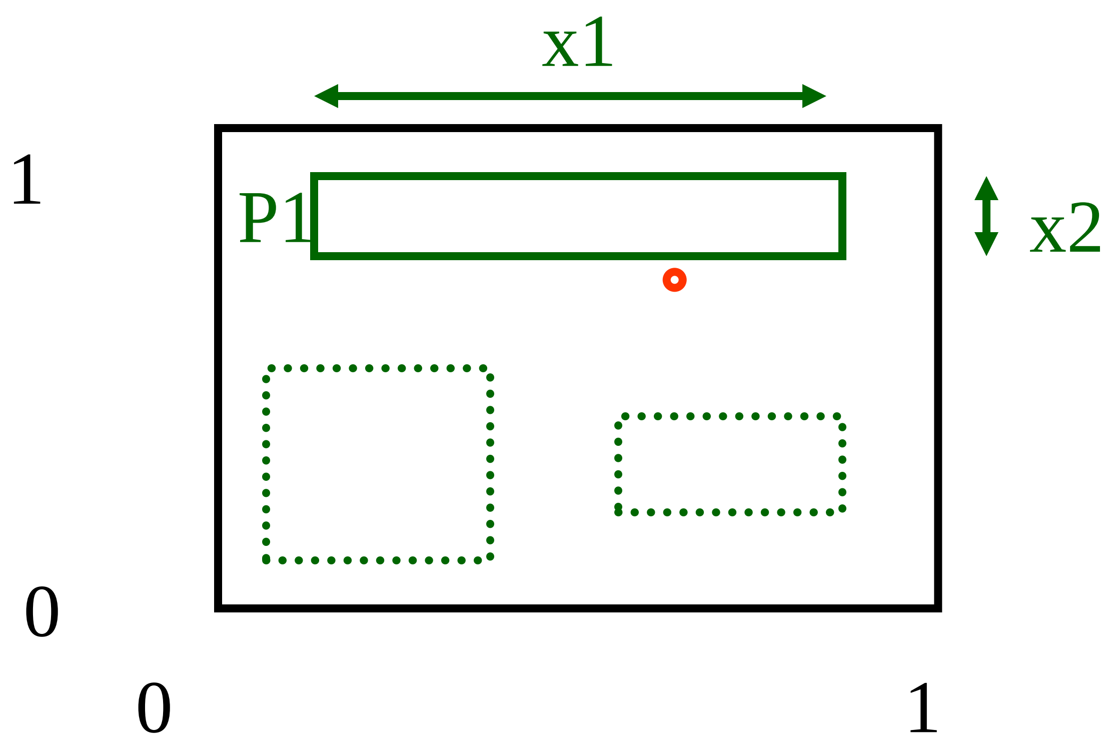
R-trees: performance analysis

- How many times will P1 be retrieved (unif. POINT queries)?



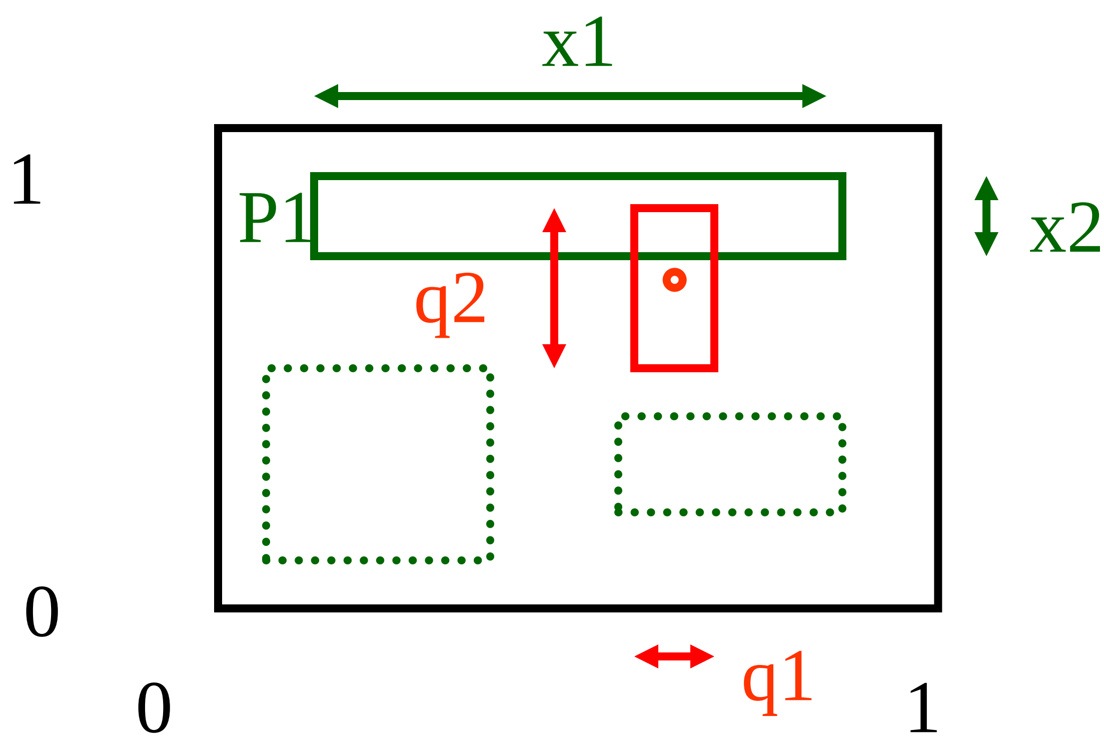
R-trees: performance analysis

- How many times will P1 be retrieved (unif. POINT queries)? A: x1*x2



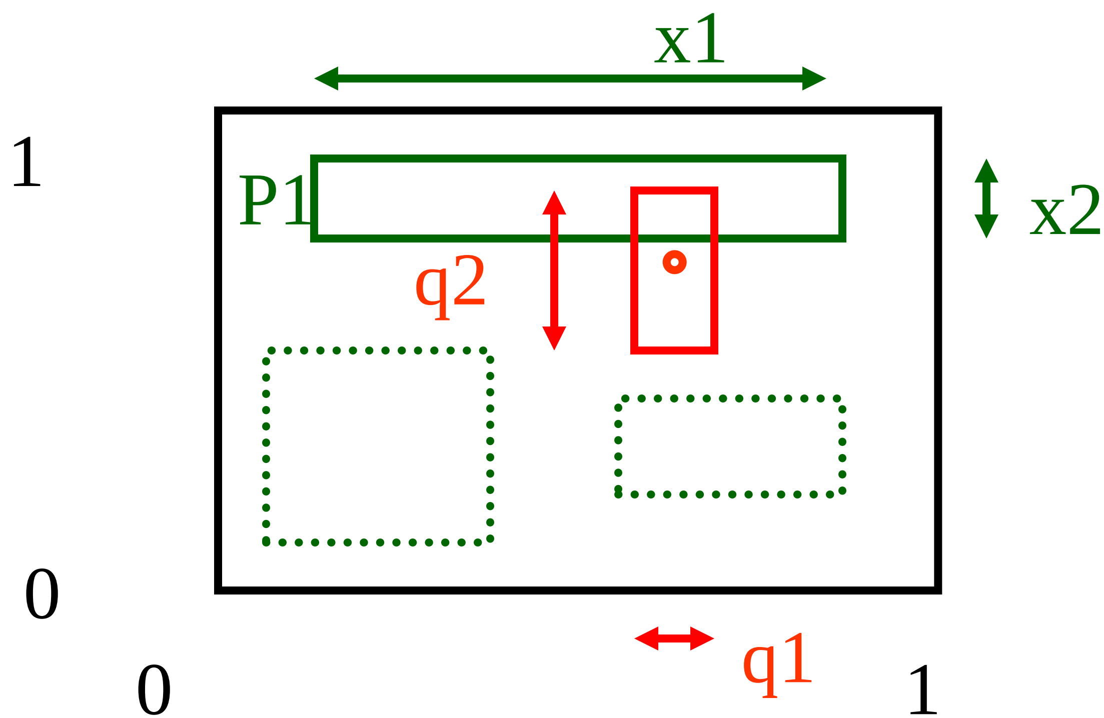
R-trees: performance analysis

- How many times will P1 be retrieved (unif. queries of size q1xq2)?



R-trees: performance analysis

- How many times will P1 be retrieved (unif. queries of size q1xq2)? A: (x1+q1)*(x2+q2)



R-trees: performance analysis

- Thus, given a tree with n nodes (i=1, ... n) we expect

$$DA(q_1, q_2) = \sum_i^n (x_{i,1} + q_1)(x_{i,2} + q_2)$$

$$= \sum_i^n x_{i,1} * x_{i,2} +$$

$$q_1 \sum_i^n x_{i,2} + q_2 \sum_i^n x_{i,1}$$

$$+ q_1 * q_2 * n$$

R-trees: performance analysis

- Thus, given a tree with n nodes (i=1, ... n) we expect

$$DA(q_1, q_2) = \sum_{i}^{n} (x_{i,1} + q_1)(x_{i,2} + q_2)$$

$$= \sum_{i}^{n} x_{i,1} * x_{i,2} + \quad \longrightarrow \text{ 'volume'}$$

$$q_1 \sum_{i}^{n} x_{i,2} + q_2 \sum_{i}^{n} x_{i,1} \quad \longrightarrow \text{ 'surface area'}$$

$$+ q_1 * q_2 * n \quad \longrightarrow \text{ count}$$

'overlap' does not seem to matter

R-trees: performance analysis

Conclusions:

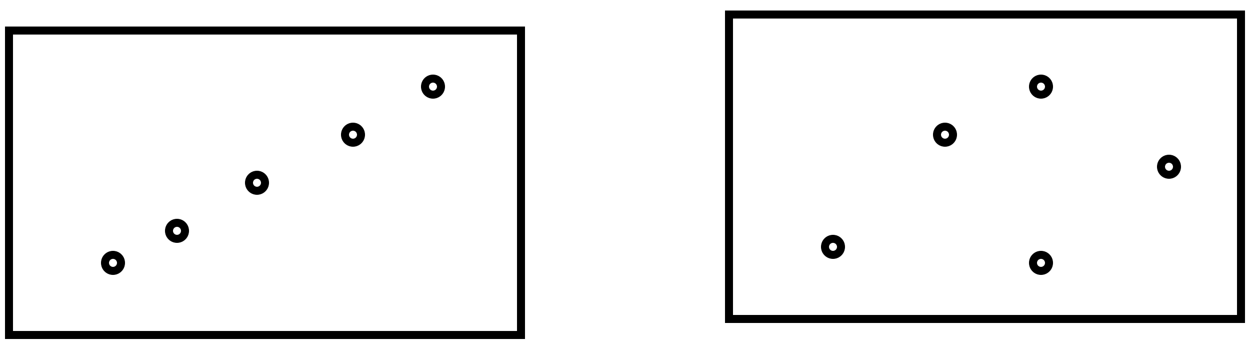
- splits should try to minimize area and perimeter
- ie., we want few, small, square-like parent MBRs
- rule of thumb: shoot for queries with q1=q2 = 0.1 (or =0.05 or so).

R-trees: performance analysis

Range queries - how many disk accesses, if we just now that we have

- *N* points in *n*-d space?

A: can not tell! need to know distribution



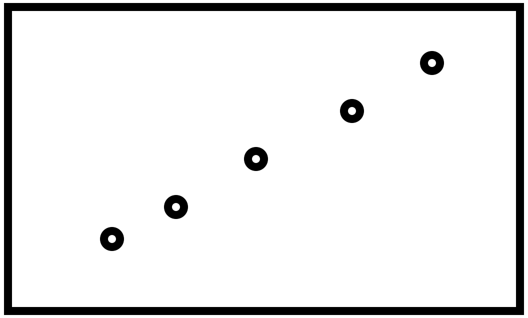
R-trees: performance analysis

What are obvious and/or realistic distributions?

A: uniform

A: Gaussian / mixture of Gaussians

A: self-similar / fractal. Fractal dimension ~ intrinsic dimension



R-trees–performance analysis

- Assuming Uniform distribution:

$$DA(q) = 1 + \sum_{j=1}^{1+h} \{ (\sqrt{D_j} + q\sqrt{\frac{N}{f^j}})^2 \}$$

where

And D is the density of the dataset, f the fanout [TS96], N the number of objects

$$D_j = \{1 + \frac{\sqrt{D_{j-1}} - 1}{\sqrt{f}}\}^2$$