# Chap4: Spatial Storage and Indexing

4.1 Storage:Disk and Files
4.2 Spatial Indexing
4.3 Trends
4.4 Summary

# Learning Objectives

- Learning Objectives (LO)
  - LO1: Understand concept of a physical data model
    - What is a physical data model?
    - Why learn about physical data models?
  - LO2: Learn how to structure data files
  - LO3: Learn how to use auxiliary data-structures
- Focus on concepts not procedures!
- Mapping Sections to learning objectives
  - LO2 - 4.1
  - LO3 - 4.2

# Physical model in 3 level design?

- Recall 3 levels of database design
  - Conceptual model: high level abstract description
  - Logical model: description of a concrete realization
  - Physical model: implementation using basic components
- Analogy with vehicles
  - Conceptual model: mechanisms to move, turn, stop, ...
  - Logical models:
    - Car: accelerator pedal, steering wheel, brake pedal, …
    - Bicycle: pedal forward to move, turn handle, pull brakes on handle
  - Physical models :
    - Car: engine, transmission, master cylinder, break lines, brake pads, …
    - Bicycle: chain from pedal to wheels, gears, wire from handle to brake pads
- We now go, so to speak, “under the hood”

# What is a physical data model?

- What is a physical data model of a database?
  - Concepts to implement logical data model
  - Using current components, e.g. computer hardware, operating systems
  - In an efficient and fault-tolerant manner
- Why learn physical data model concepts?
  - To be able to choose between DBMS brand names
    - Some brand names do not have spatial indices!
  - To be able to use DBMS facilities for performance tuning
  - For example, if a query is running slow,
    - one may create an index to speed it up
  - For example, if loading of a large number of tuples takes for ever
    - one may drop indices on the table before the inserts
    - and recreate index after inserts are done!

# An interesting fact about physical data model

- Physical data model design is a trade-off between
  - Efficiently support a small set of basic operations of a few data types
  - Simplicity of overall system
- Each DBMS physical model
  - Choose a few physical DM techniques
  - Choice depends chosen sets of operations and data types
- Relational DBMS physical model
  - Data types: numbers, strings, date, currency
    - one-dimensional, totally ordered
  - Operations:
    - search on one-dimensional totally order data types
    - insert, delete, ...

# Common Spatial Queries and Operations

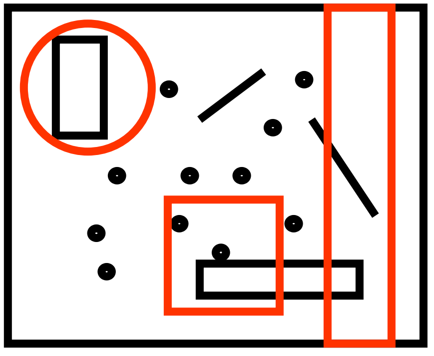
- Physical model provides simpler operations needed by spatial queries!
- *Common Queries*
  - *Range query*
  - *Nearest neighbor*
  - *Spatial-join query*
  - *Others (Closest-pair query, Color range query, etc.)*

Example schema:

- A big company with a lot of stores and warehouses
- Store(Id int, Name char(30), Location Point)
- Warehouse(Id int, Name char(30), Location Point)

# Range query

- Find all **objects** contained in a rectangle/circle

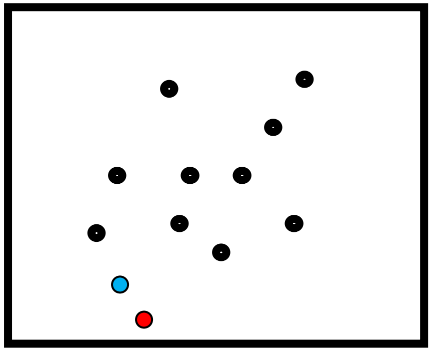


- Ex. Find all warehouses at dist < 50 Km from location (0,0)

```
Select WarehouseId
From Warehouse
Where distance(Warehouse.Location, Point(0,0)) < 50;
```

# Nearest neighbor query

- Find the **object(s)** closest to another object

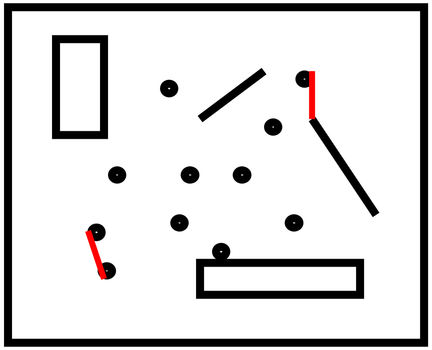


- Ex. Find the store closest to store 101

```
Select s2.Id
From Store s1, Store s2
Where s1.Id = 101 and distance(s1.Location, s2.Location) = min
               (Select distance(s1.Location, s3.Location)
                From Store s3);
```

# Spatial-join query

- Find pairs of **objects** satisfying a property



- Ex. Find all pairs of stores-warehouses with dist < 10 Km

```
Select Store.Id, Warehouse.Id
From Store, Warehouse
Where distance(Store.Location, Warehouse.Location)< 10
```

# Other types of queries

- Closest-pair query: Find the closest pair (i.e., with min distance) between a store and a warehouse
  - (Coral et al., 2000)
- Color range query: What type of objects (e.g., stores, warehouses) are inside a rectangle/circle
  - Find not the objects themselves, but their types
  - (Nanopoulos et al., 2001)
- Computational geometry has many interesting queries
  - Not all of them have been transferred to SDB realm

# Learning Objectives

- Learning Objectives (LO)
  - LO1: Understand concept of a physical data model
  - LO2: Learn how to structure data files
    - What is a file structure? Why structure files?
    - What are common structures for spatial data file?
  - LO3: Learn how to use auxiliary data-structures

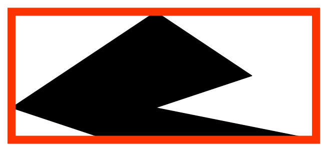
- Mapping Sections to learning objectives
  - LO2 - 4.1
  - LO3 - 4.2

# 4.1.4 File Structures

- What is a file structure?
  - A method of organizing records in a file
  - For efficient implementation of common file operations on disks
  - Example: ordered files
- Measure of efficiency
  - I/O cost: Number of disk sectors retrieved from secondary storage
  - CPU cost: Number of CPU instruction used
- Two basic categories of file structures in SDB
  - Point Access Methods (objects are strictly points)
  - Spatial Access Methods (objects have spatial extend)

# Spatial Access Methods (SAMs)

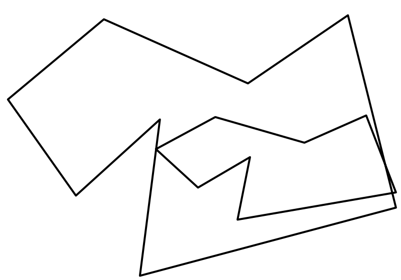
- Indexes for spatial data that have extend (not only point data)
- Use only Minimum Bounding Rectangles – **MBR**s (filtering)

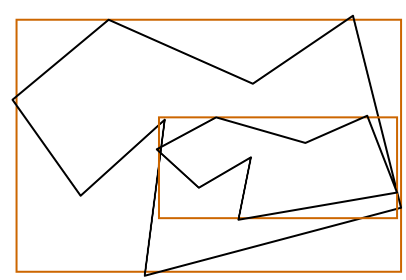


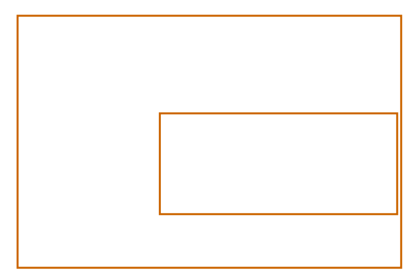
- R-tree (Guttman, 1984) is the prominent SAM
  - Implemented in Oracle, Postgres, Informix

# 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



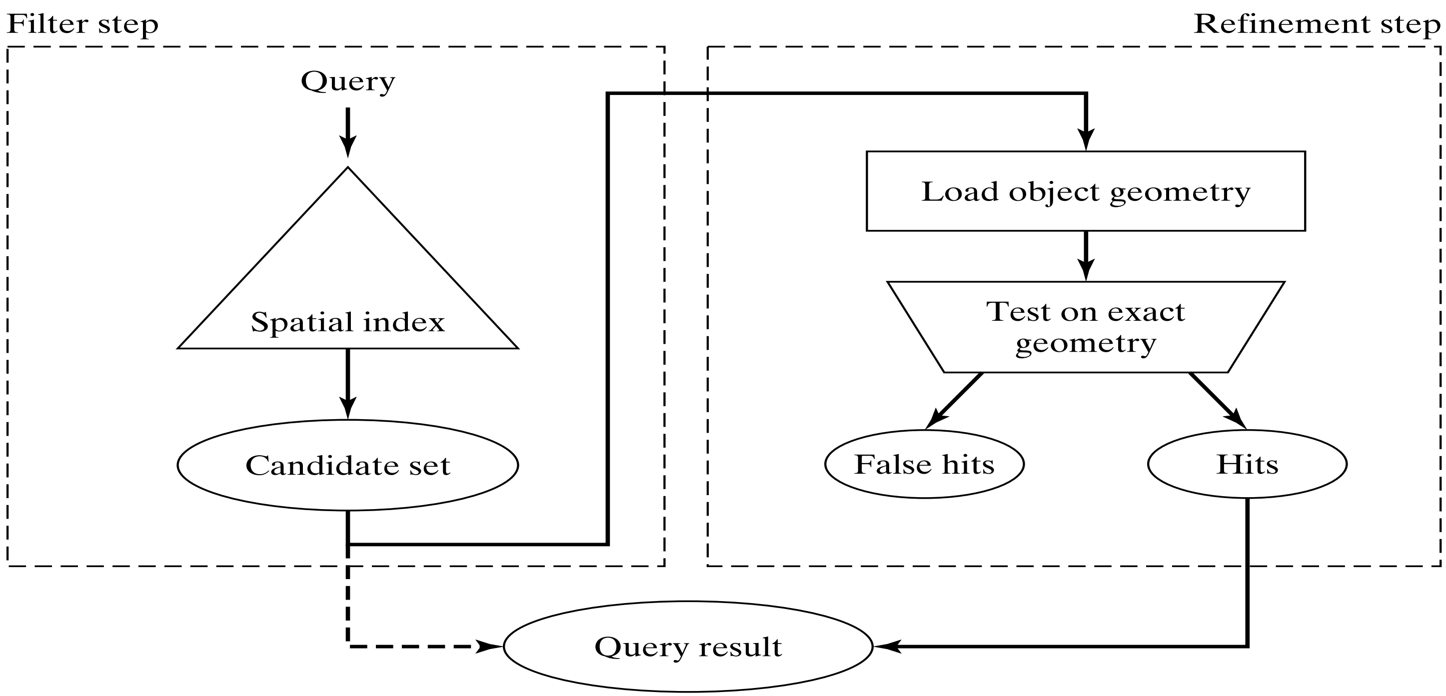




# R-tree query processing:Filter-Refining

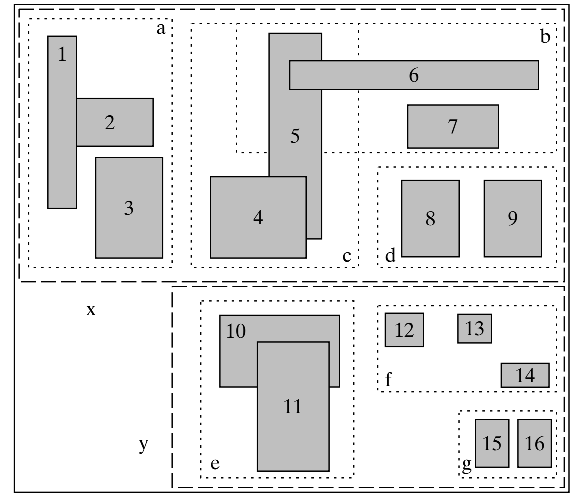
- Processing a  spatial query Q
  - Filter step : find a superset S of object in answer to Q
    - Using approximate of spatial data type and operator
  - Refinement step : find exact answer to Q reusing a GIS to process S
    - Using exact spatial data type and operation

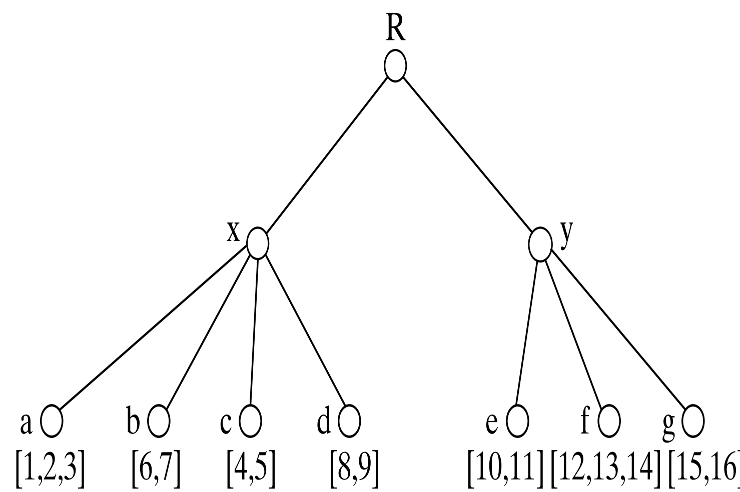
Fig 5.1



# R-Tree

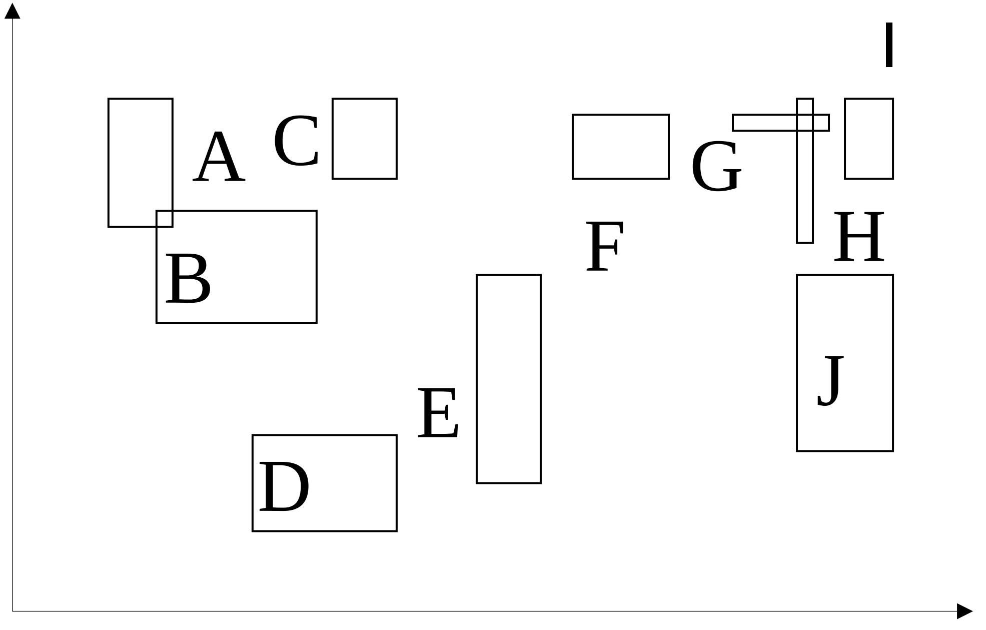
- A multi-way external memory tree
- Index nodes and data (leaf) nodes
- All leaf nodes appear on the same level
- Every node contains between m and M entries
- The root node has at least 2 entries (children)



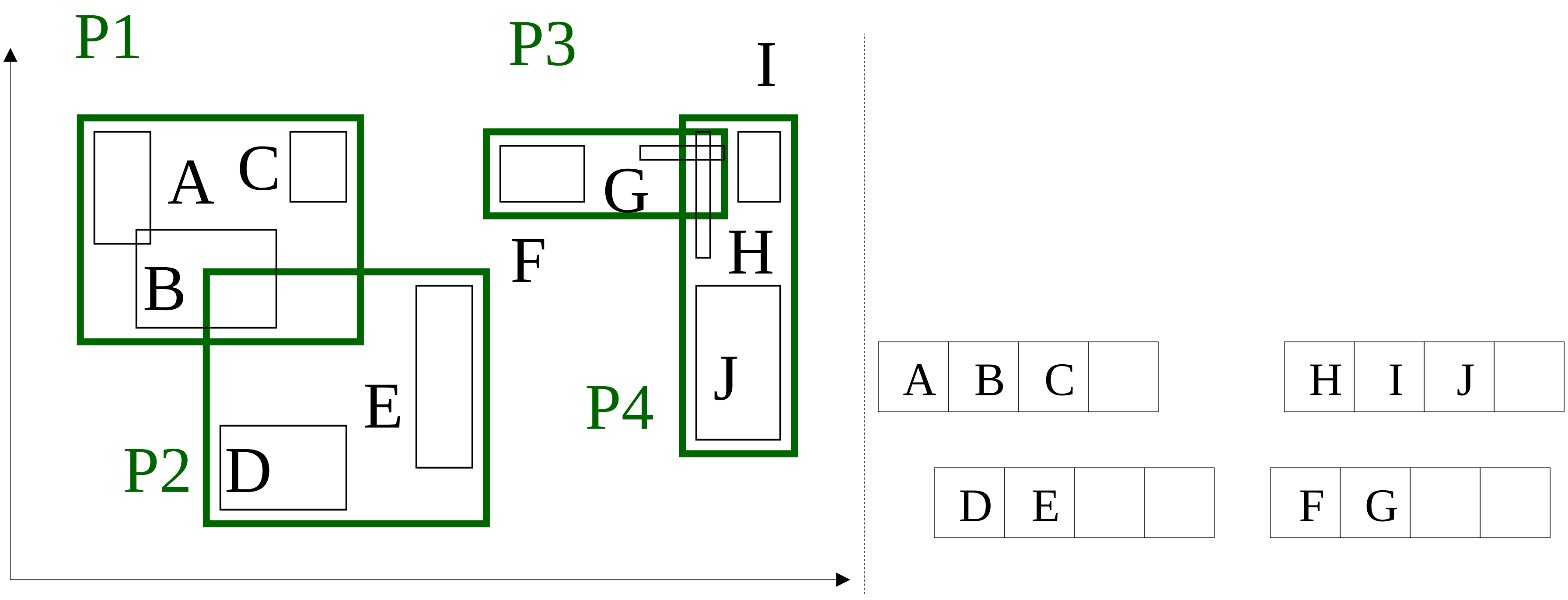
# Example

- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page



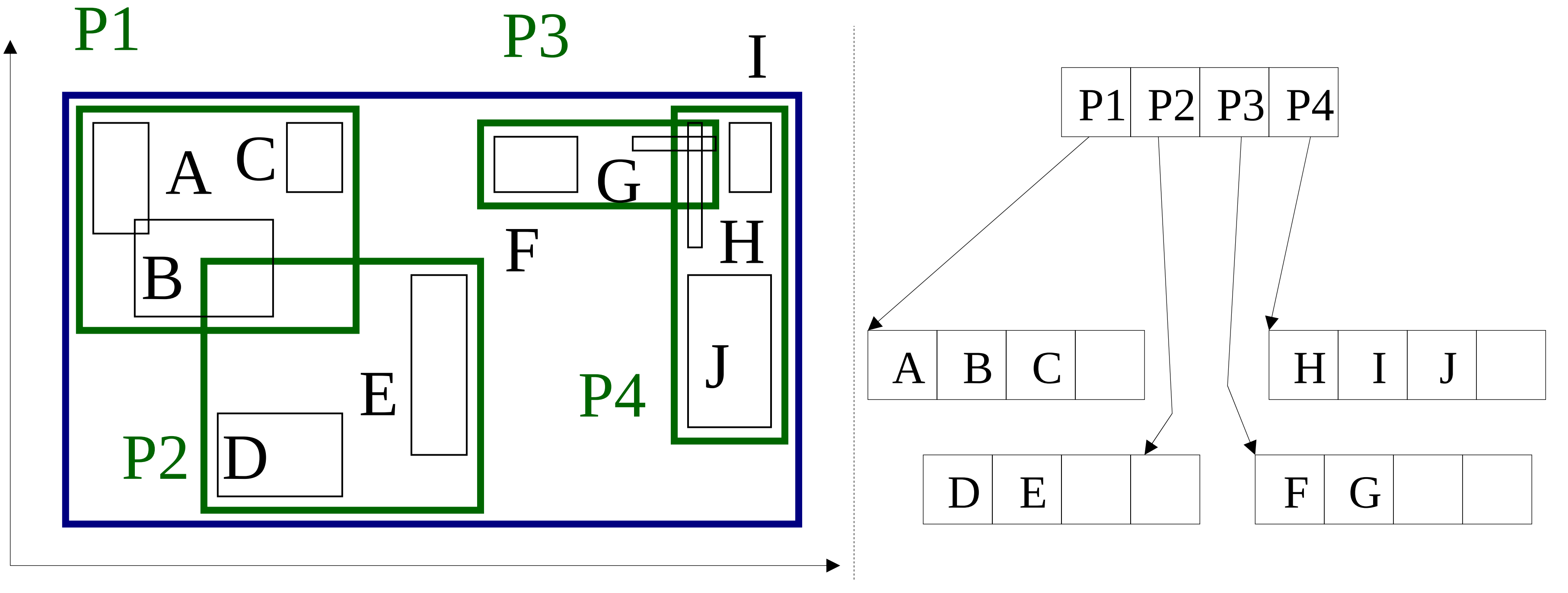
# Example

- F=4



# Example

- F=4

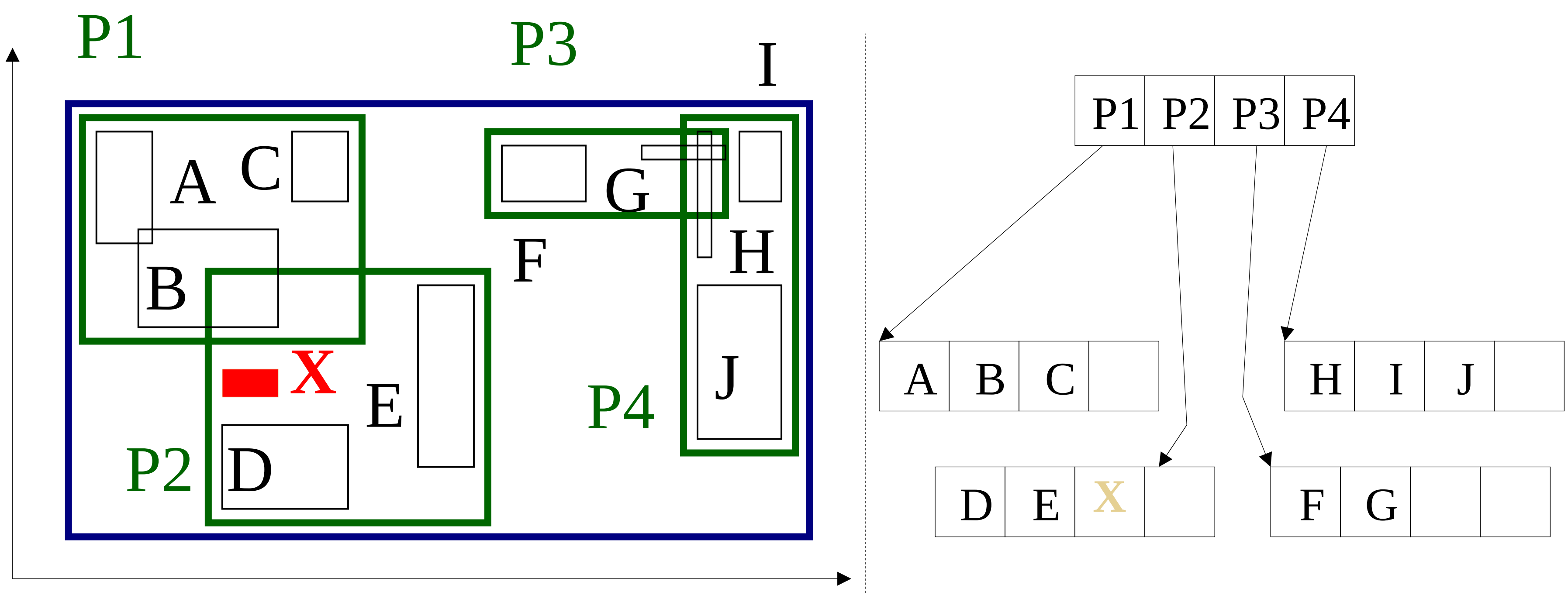


# R-trees:Insertion

- Insert new MBR in a leaf
- Find the leaf to insert by searching, starting from the root
- How to find the next node to insert the new object?
  - Using ChooseLeaf: Find the entry that needs the least enlargement to include Y. Resolve ties using the area (smallest)

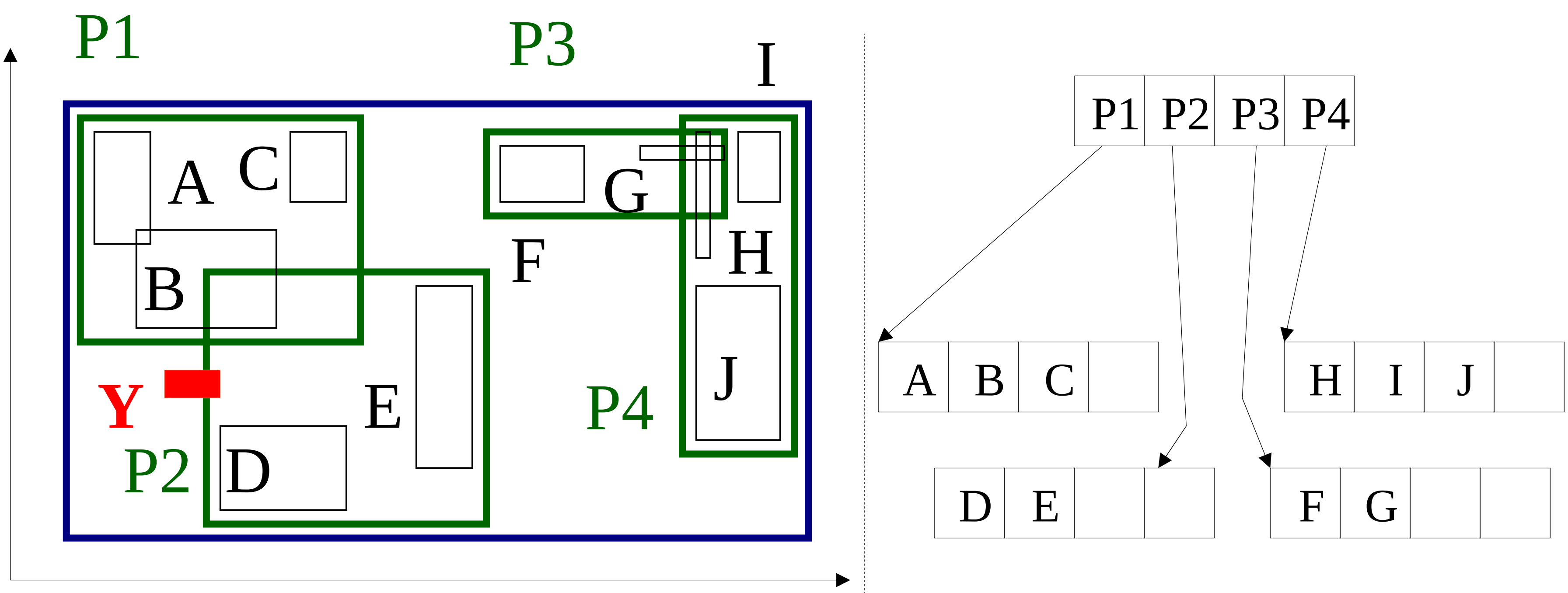
# R-trees:Insertion

Insert X



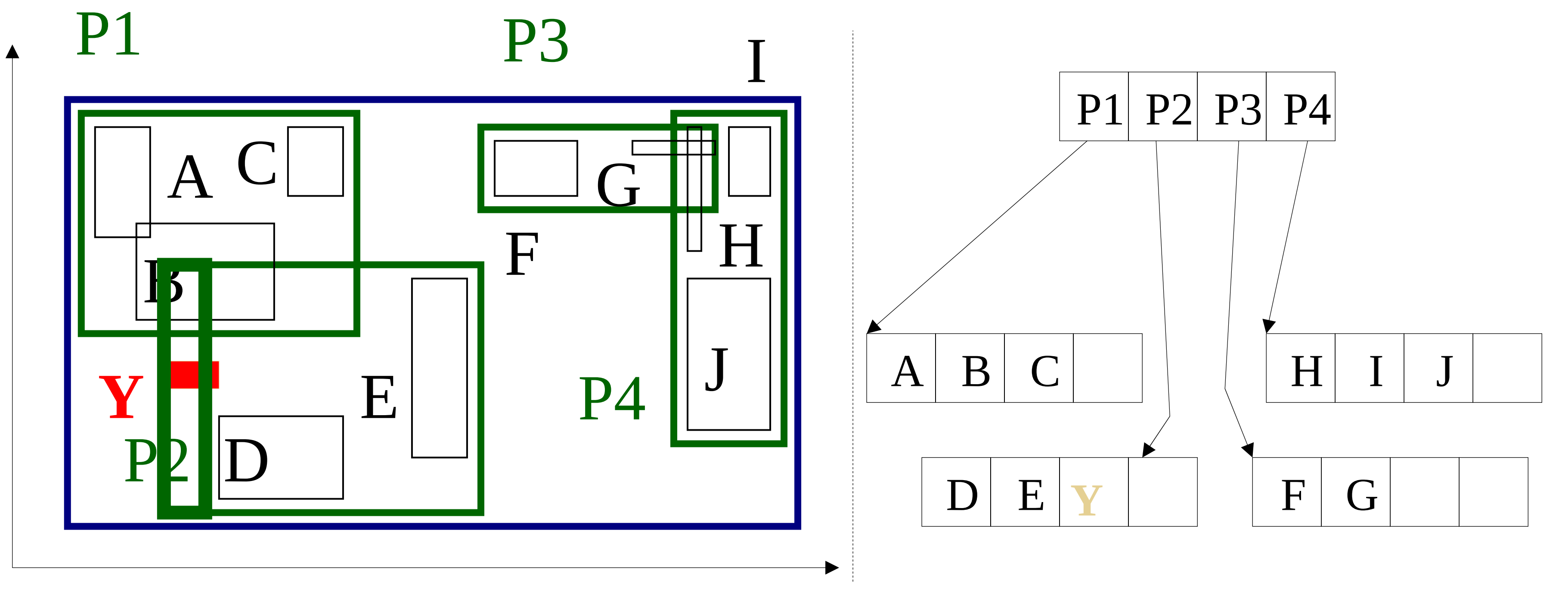
# R-trees:Insertion

Insert Y



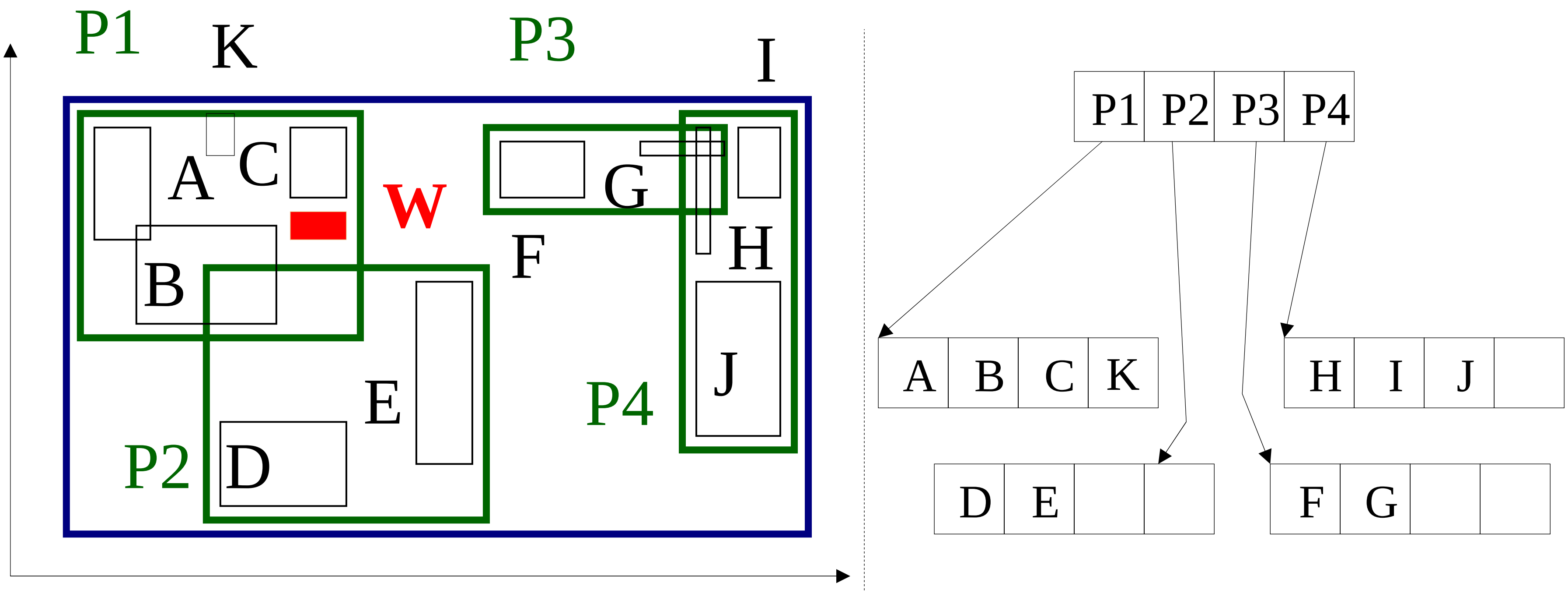
# R-trees:Insertion

- Extend the parent MBR



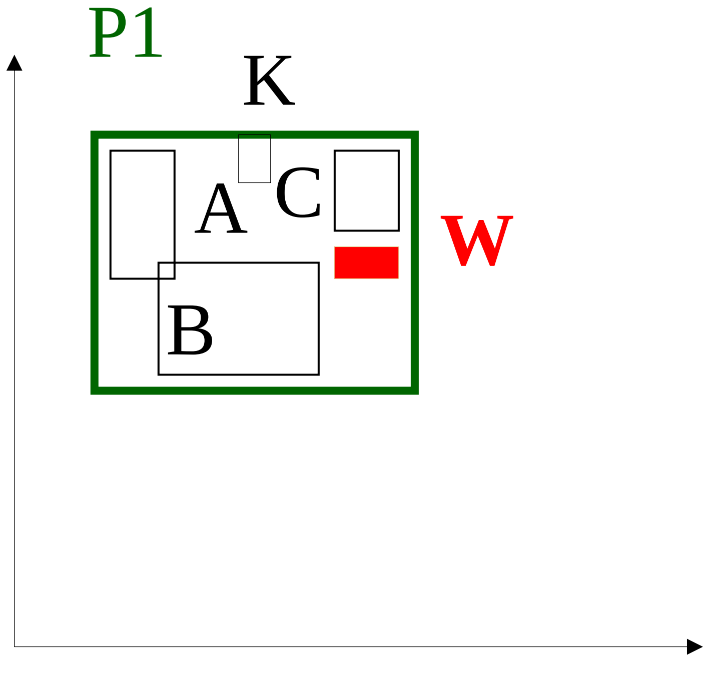
# R-trees:Insertion

- If node is full then Split : ex. Insert w



# R-trees:Split

- Split node P1: partition the MBRs into two groups.

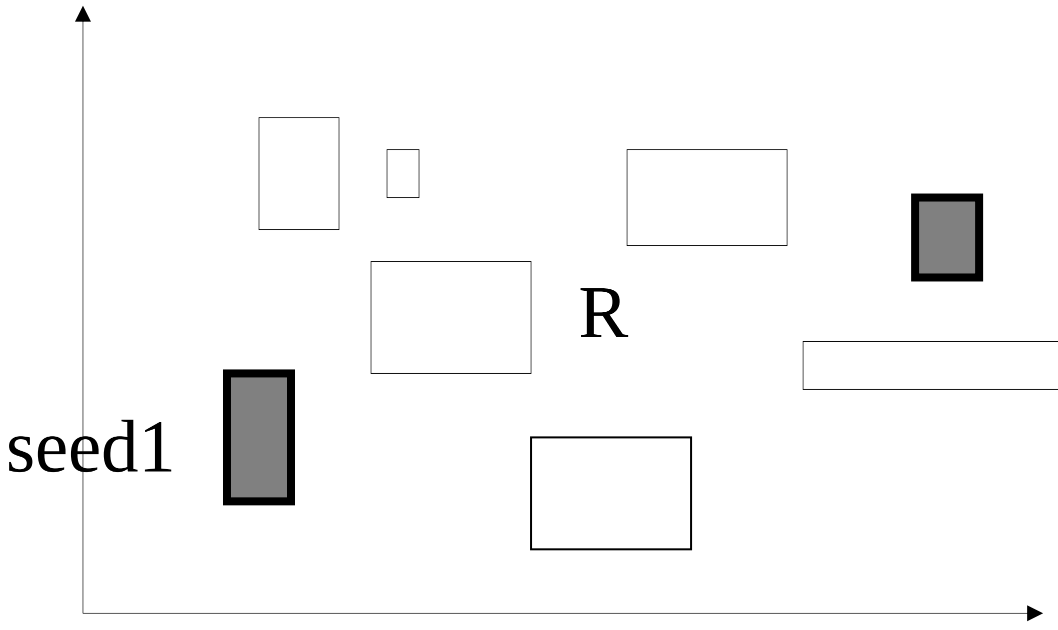



- A1: ‘linear’ split
- A2: quadratic split
- A3: exponential split:

  $2^{M-1}$ choices

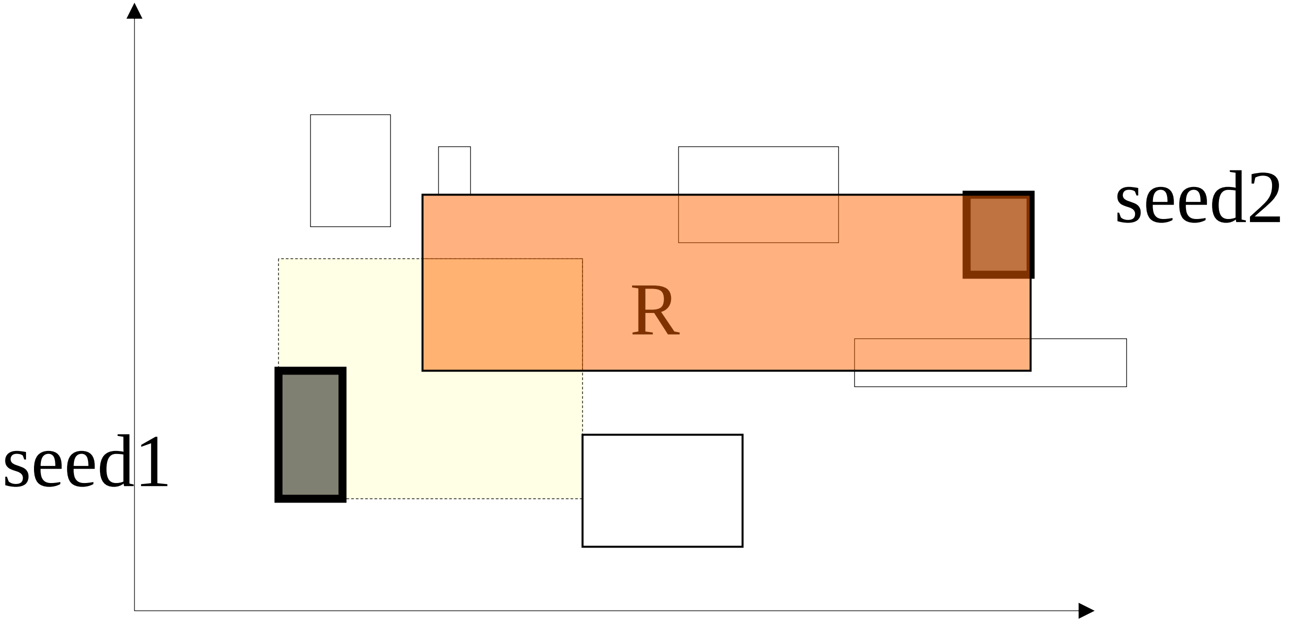
# R-trees:Split

- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’



# R-trees:Split

- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’:
- ‘closest’: the smallest increase in area



# R-trees:Split

- How to pick Seeds:
  - Linear:Find the highest and lowest side in each dimension, normalize the separations, choose the pair with the greatest normalized separation
  - Quadratic: For each pair E1 and E2, calculate the rectangle J=MBR(E1, E2) and d= J-E1-E2. Choose the pair with the largest d

# *R-trees:Insertion (the complete algorithm)*

- Use the **ChooseLeaf** to find the leaf node to insert an entry E
- If leaf node is full, then **Split,** otherwise insert there
  - Propagate the split upwards, if necessary
- Adjust parent nodes

# R-Trees:Deletion

- Find the leaf node that contains the entry E
- Remove E from this node
- If underflow:
  - Eliminate the node by removing the node entries and the parent entry
  - Reinsert the orphaned (other entries) into the tree using **Insert**

# R-trees: Variations

- R+-tree: DO not allow overlapping, so split the objects (similar to z-values)
- R*-tree: change the insertion, deletion algorithms (minimize not only area but also perimeter, forced re-insertion)
- Hilbert R-tree: use the Hilbert values to insert objects into the tree

## R-trees:Range search

pseudocode:

check the root

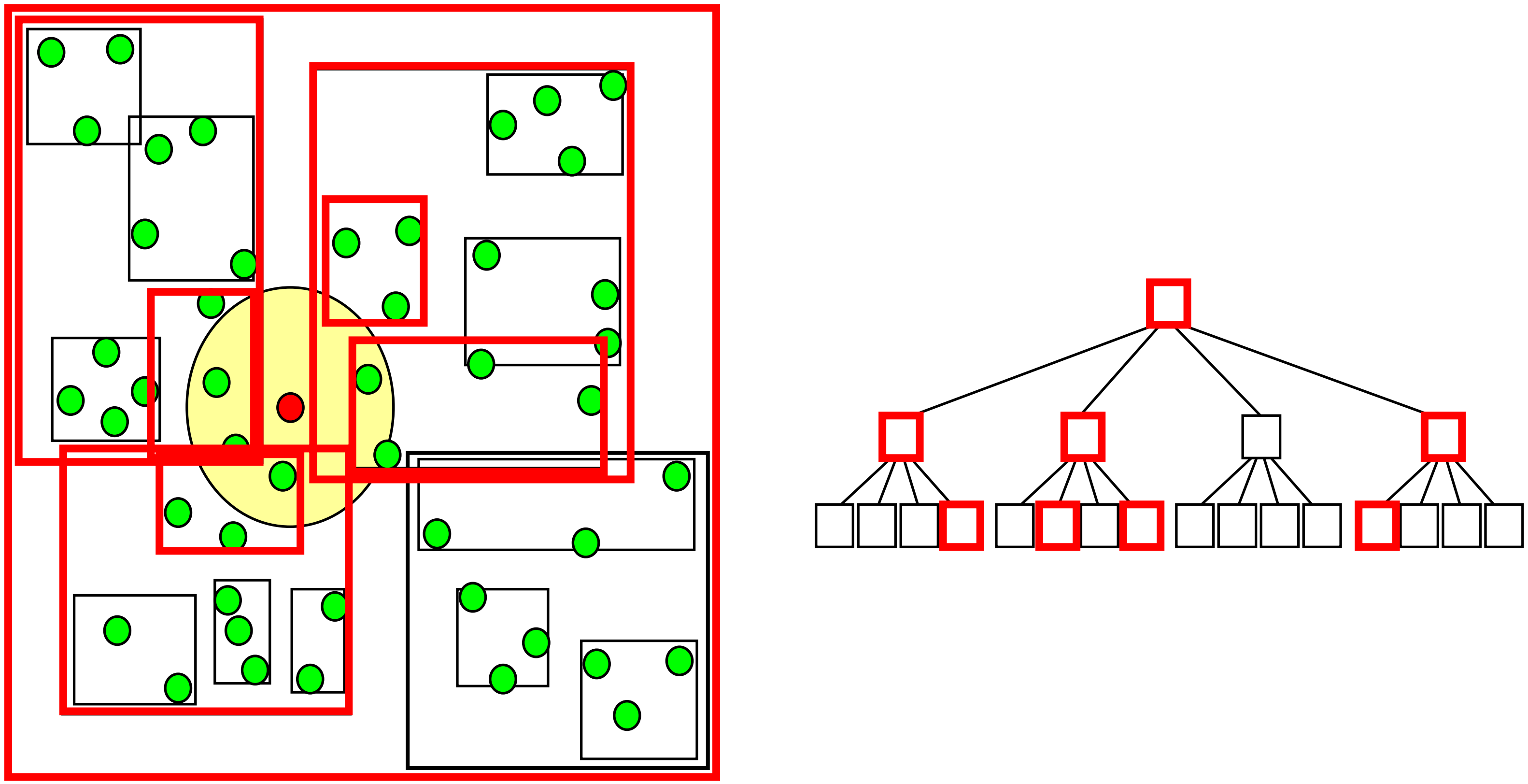
for each branch,

if its MBR intersects the query rectangle

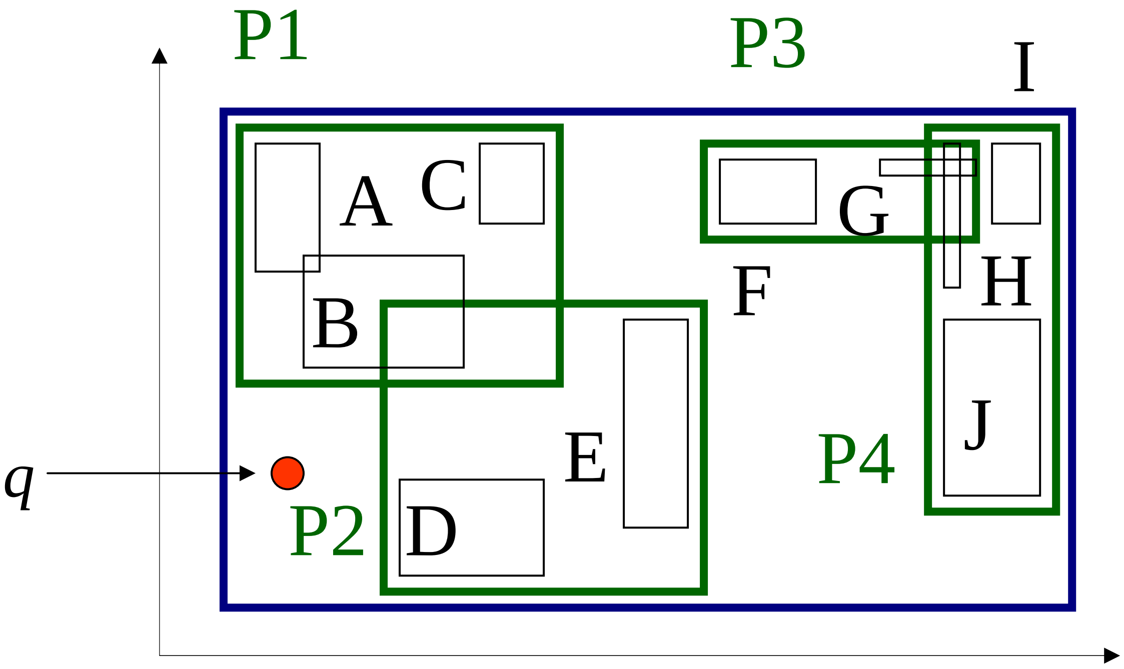
apply range-search (or print out, if this

is a leaf)

# Example (DFS searching)

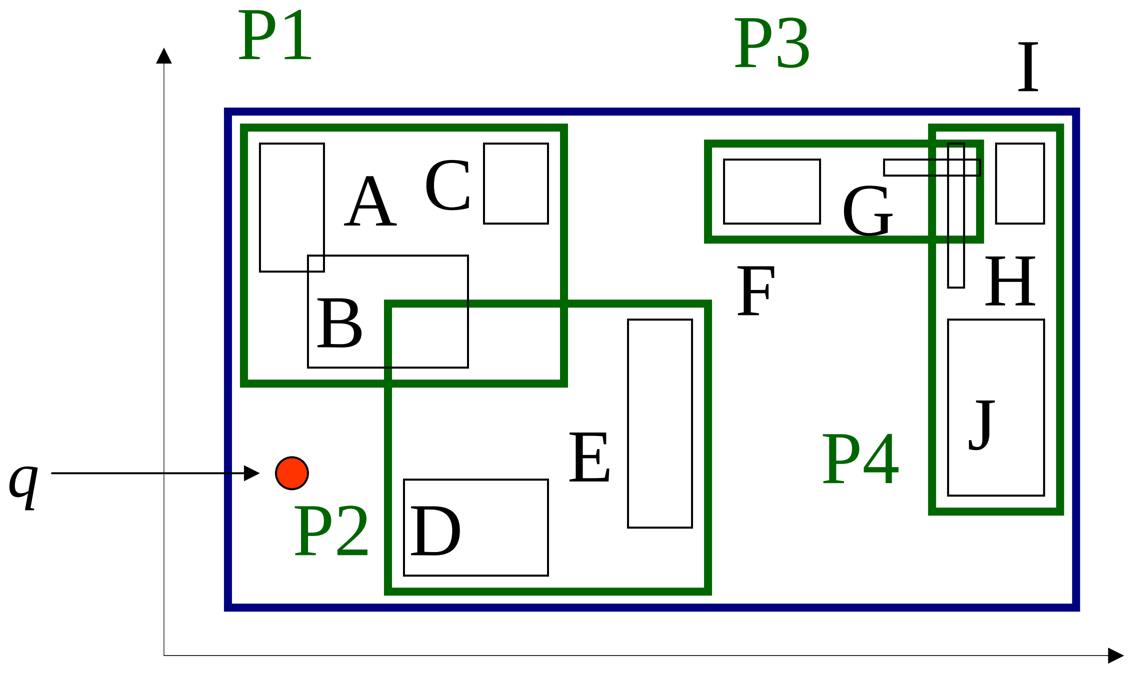


# R-trees: NN search



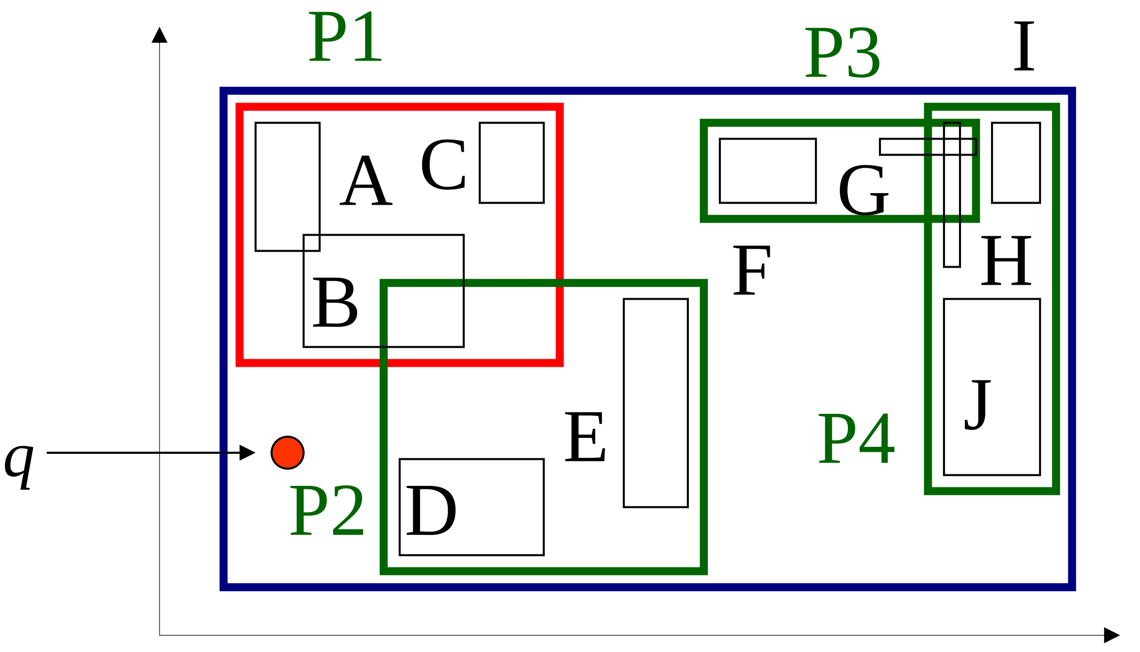
# R-trees: NN search

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



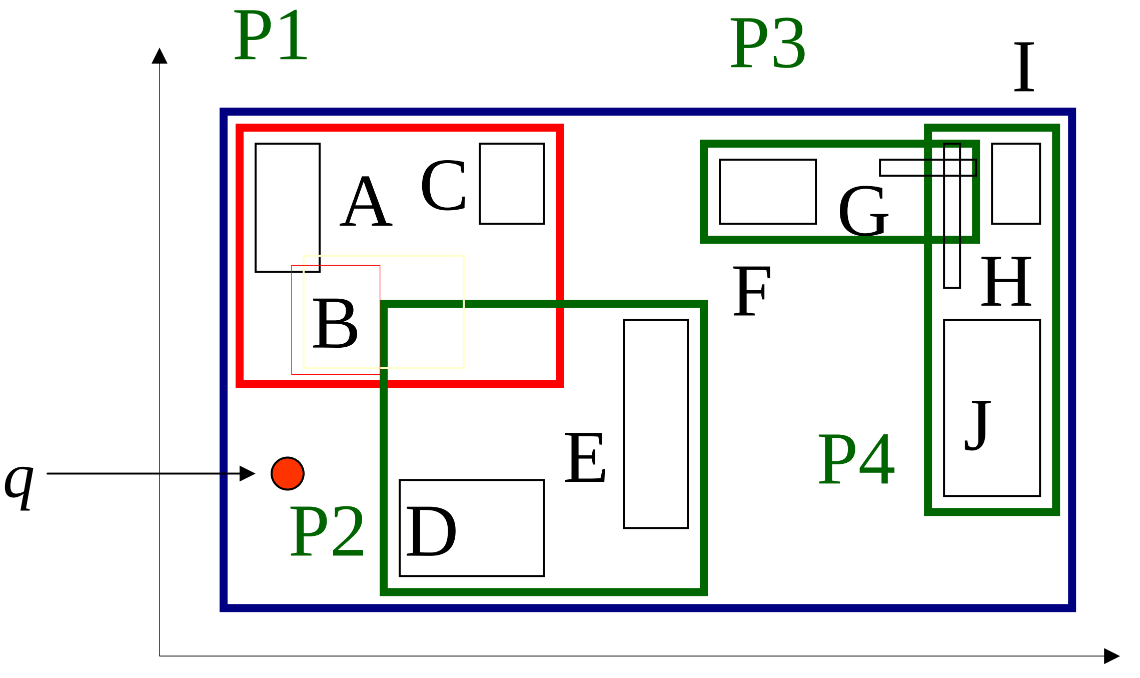
# R-trees: NN search (simple algorithm)

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



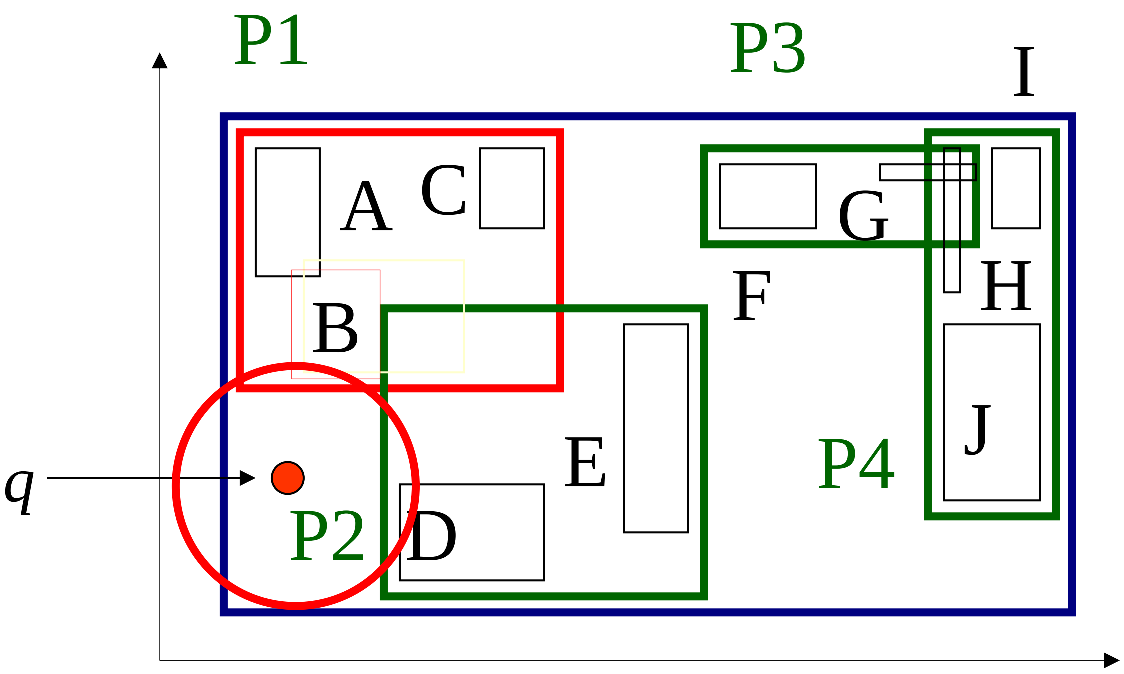
## R-trees: NN search (simple algorithm)

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



# R-trees: NN search (simple algorithm)

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

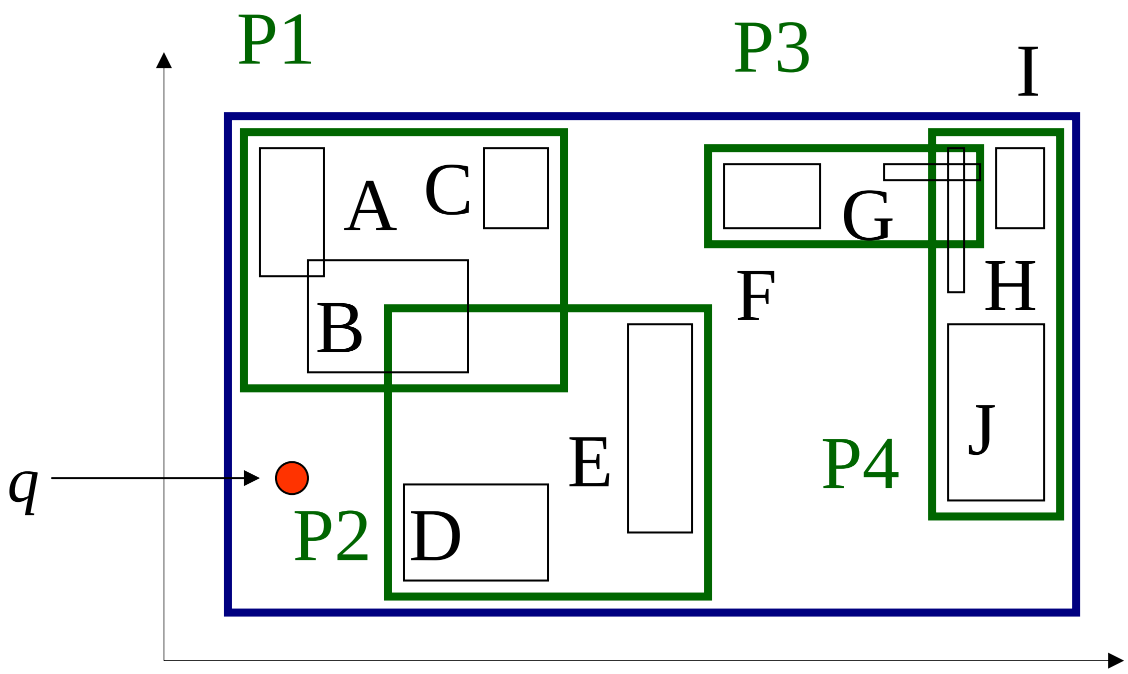


# R-trees: NN search (better algorithm)

- 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

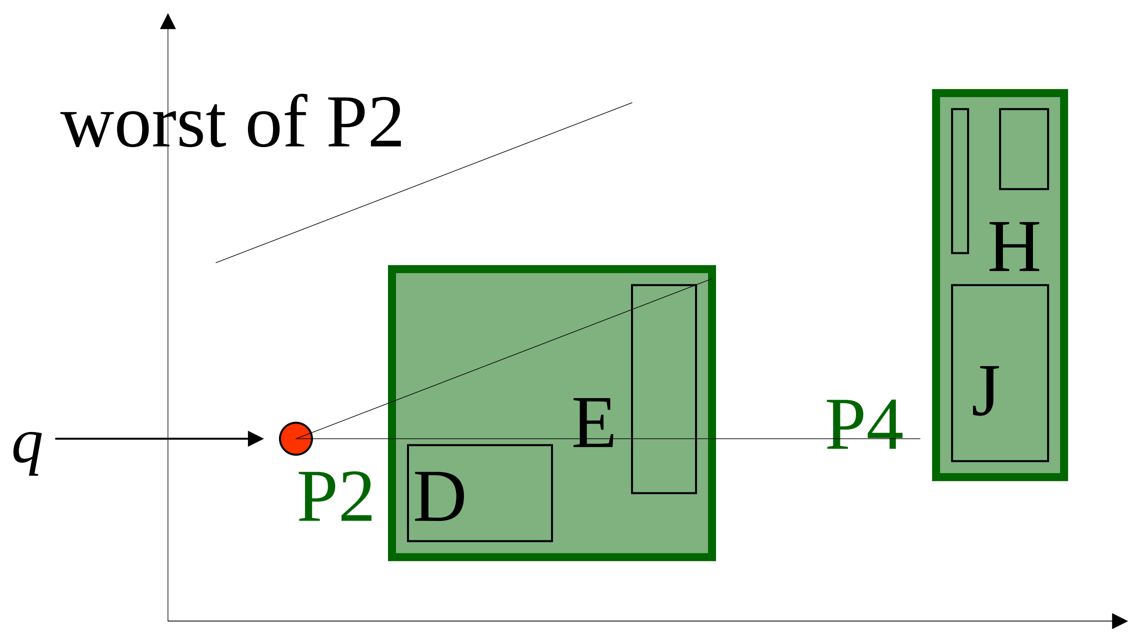


# *R-trees: NN search (better algorithm)*

best of P4

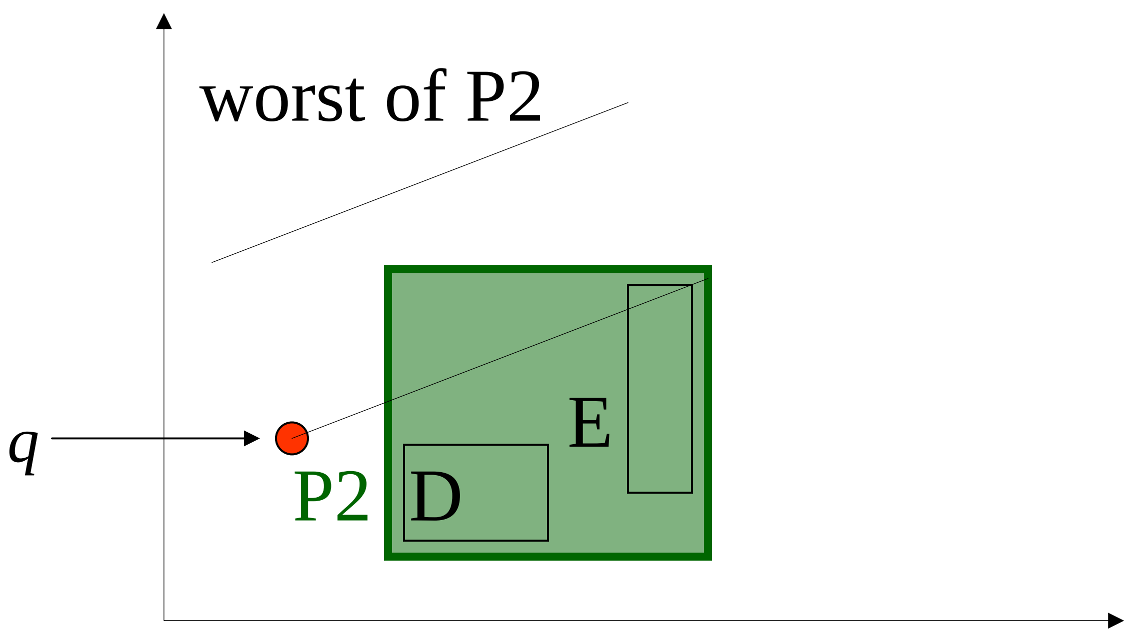
=> P4 is useless

for 1-nn



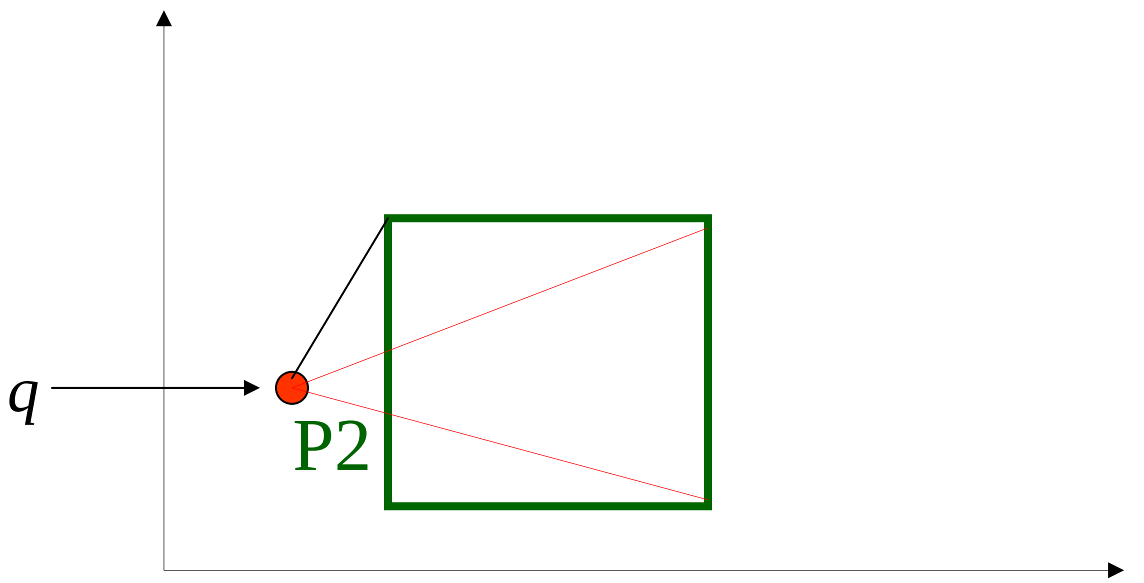
# R-trees: NN search (better algorithm)

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



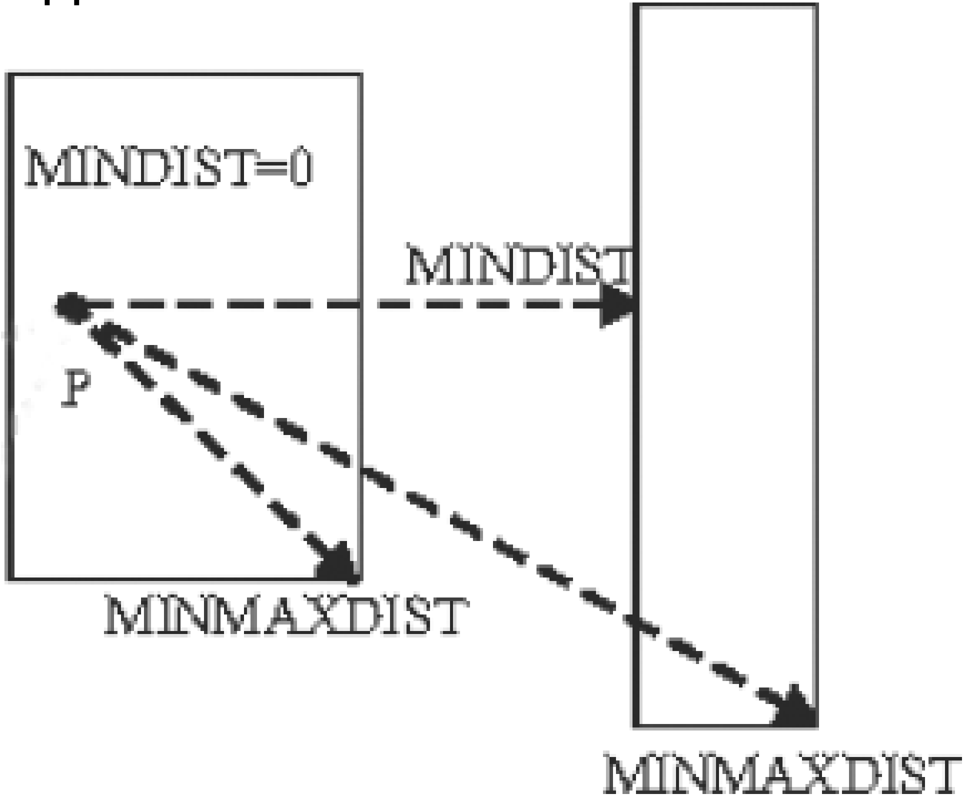
# R-trees: NN search (better algorithm)

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



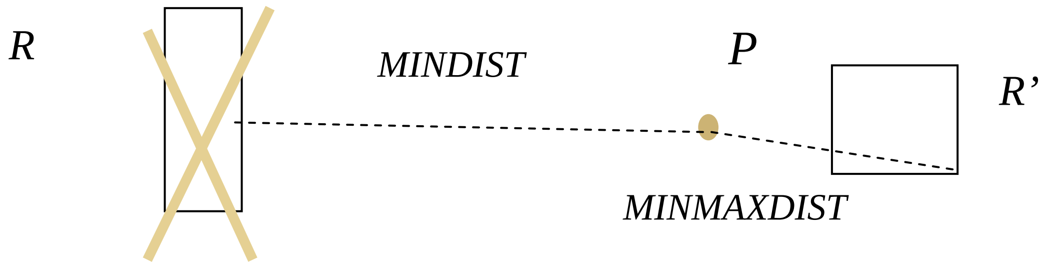
# MINDIST, MINMAXDIST

- 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

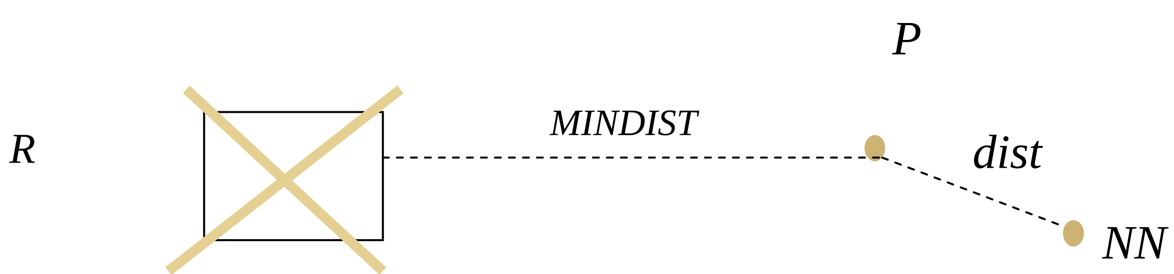


# Pruning with MINDIST and MINMAXDIST

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




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



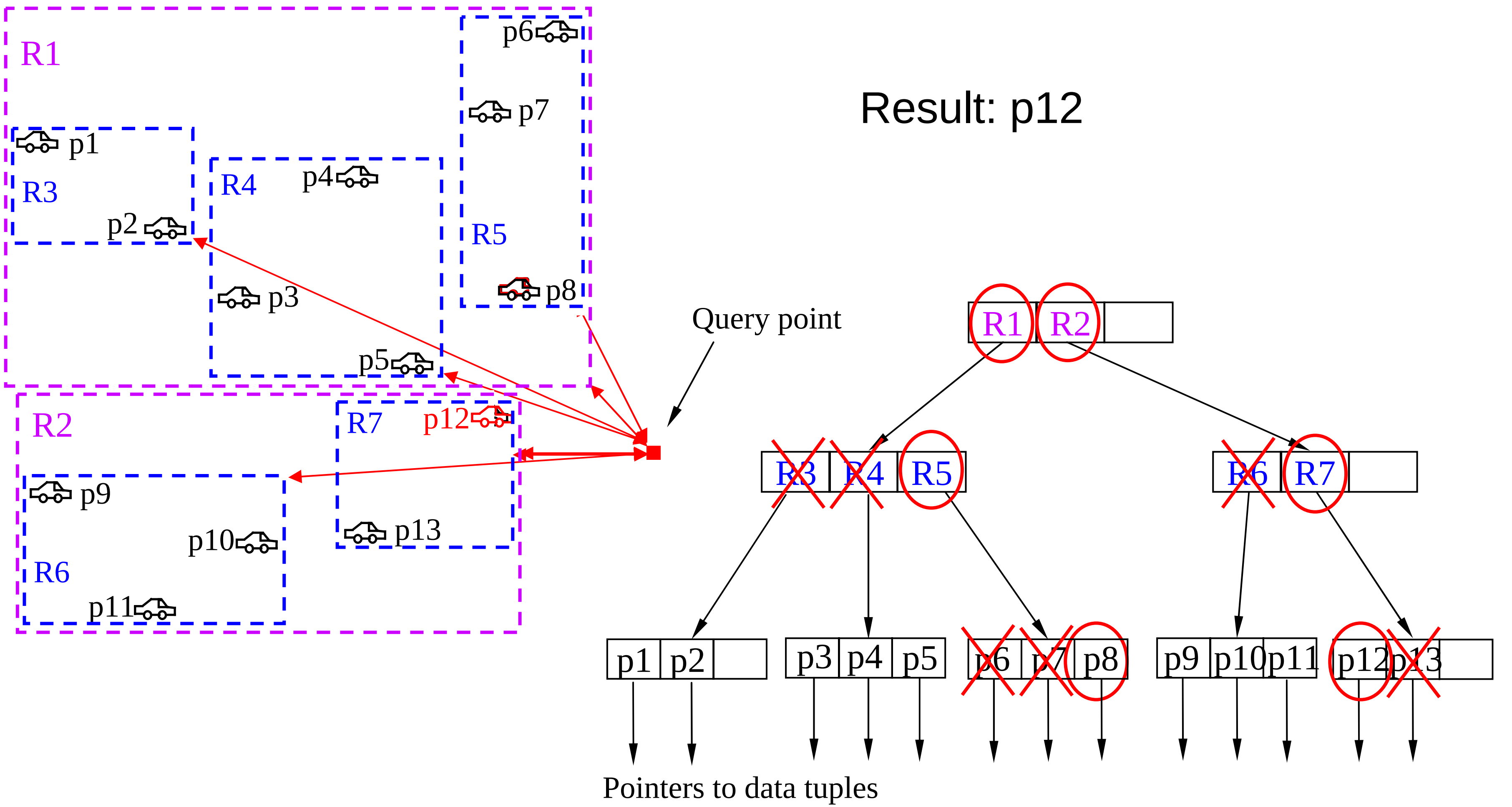
# Order of searching

- 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
```

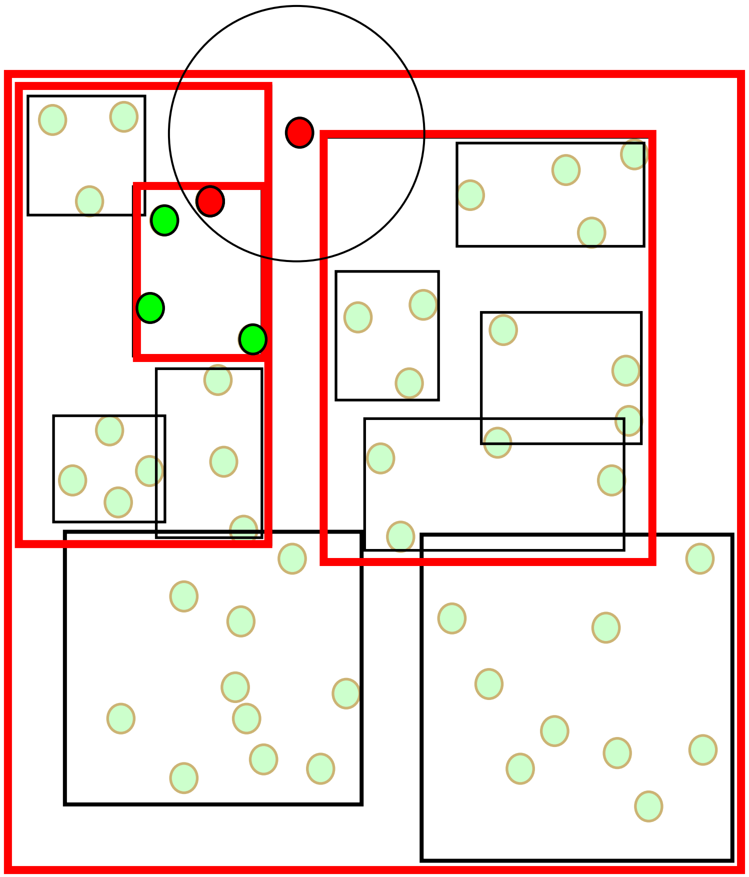
# NN example


R1
p6
p7
p1
R3
R4
p4
p2
R5
p3
p8
p5
Query point
R2
R7
p12
p9
p10
R6
p11
p13
Result: p12
R1
R2
R3
R4
R5
R6
R7
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
p11
p12
p13
Pointers to data tuples

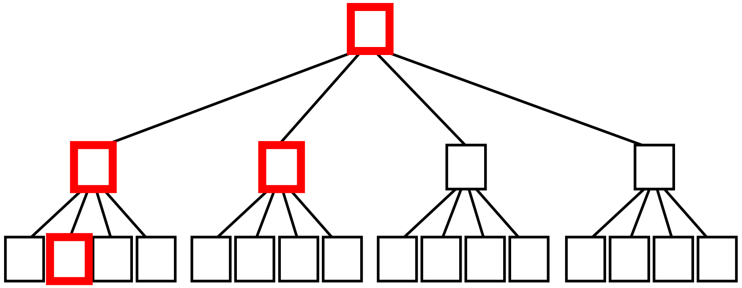
# Optimal Strategy for NN search

- 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

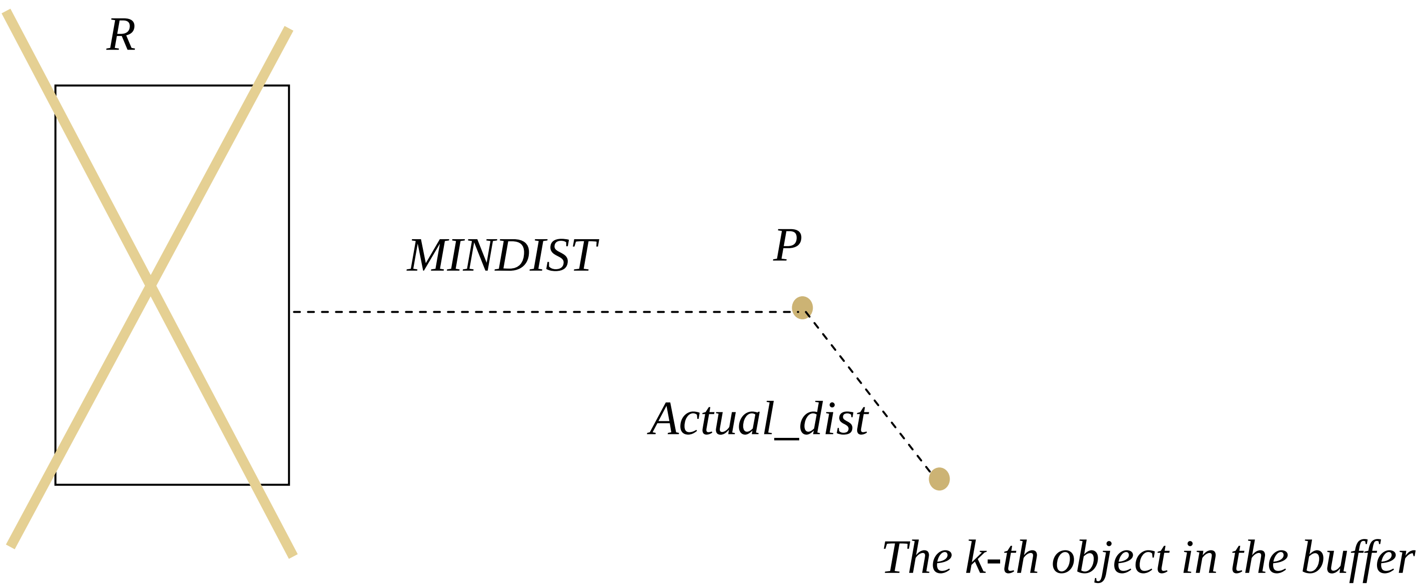


4 page accesses



# 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:



## R-trees: performance analysis

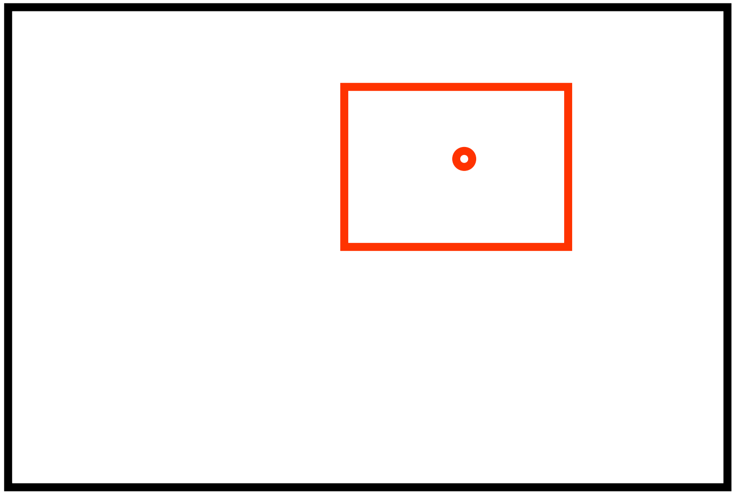
- How many disk (=node) accesses we'll need for
  - range
  - nn
  - spatial joins
- why does it matter?

# *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?

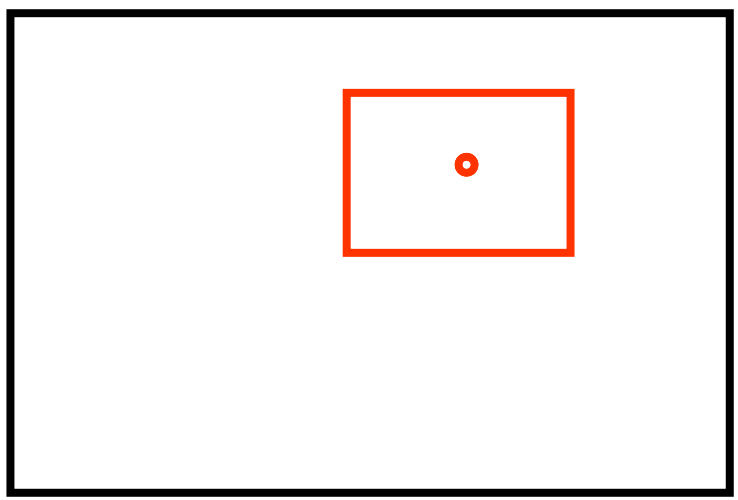
# R-trees: performance analysis

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



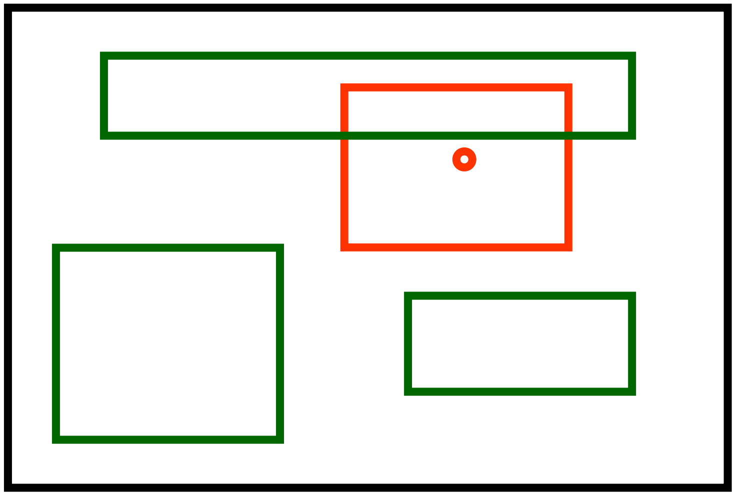
# R-trees: performance analysis

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



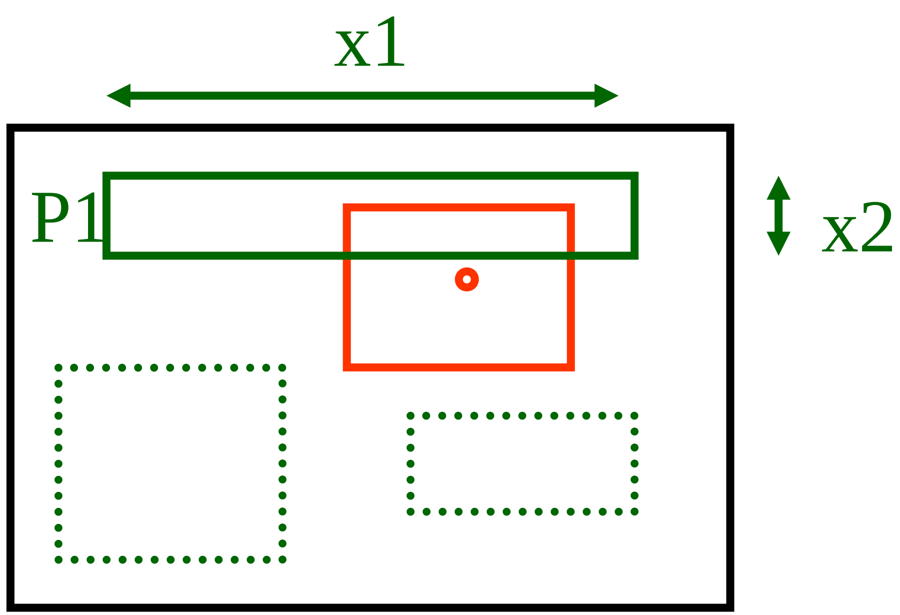
# R-trees: performance analysis

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



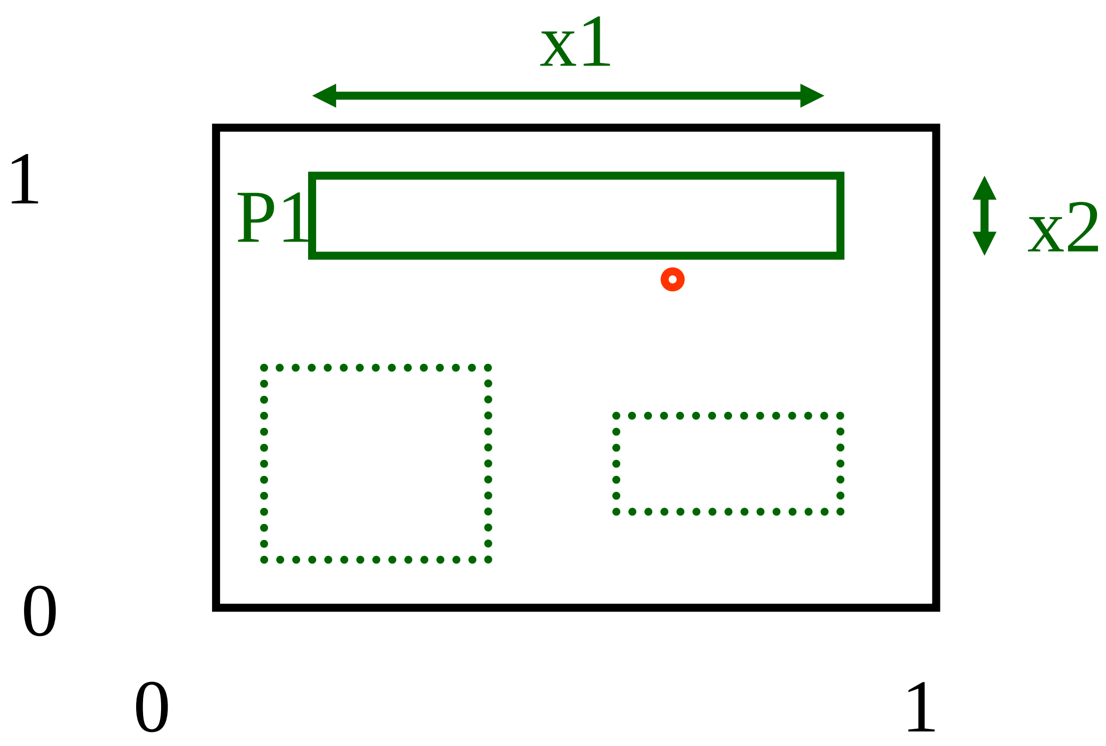
# R-trees: performance analysis

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



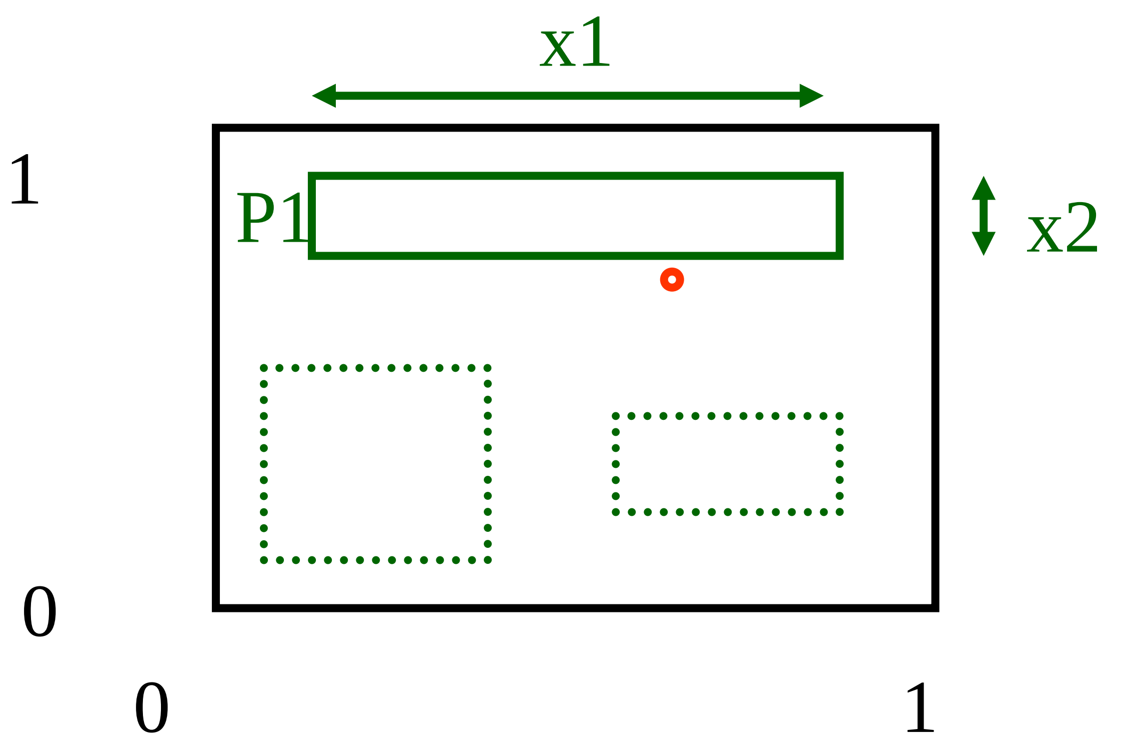
# R-trees: performance analysis

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



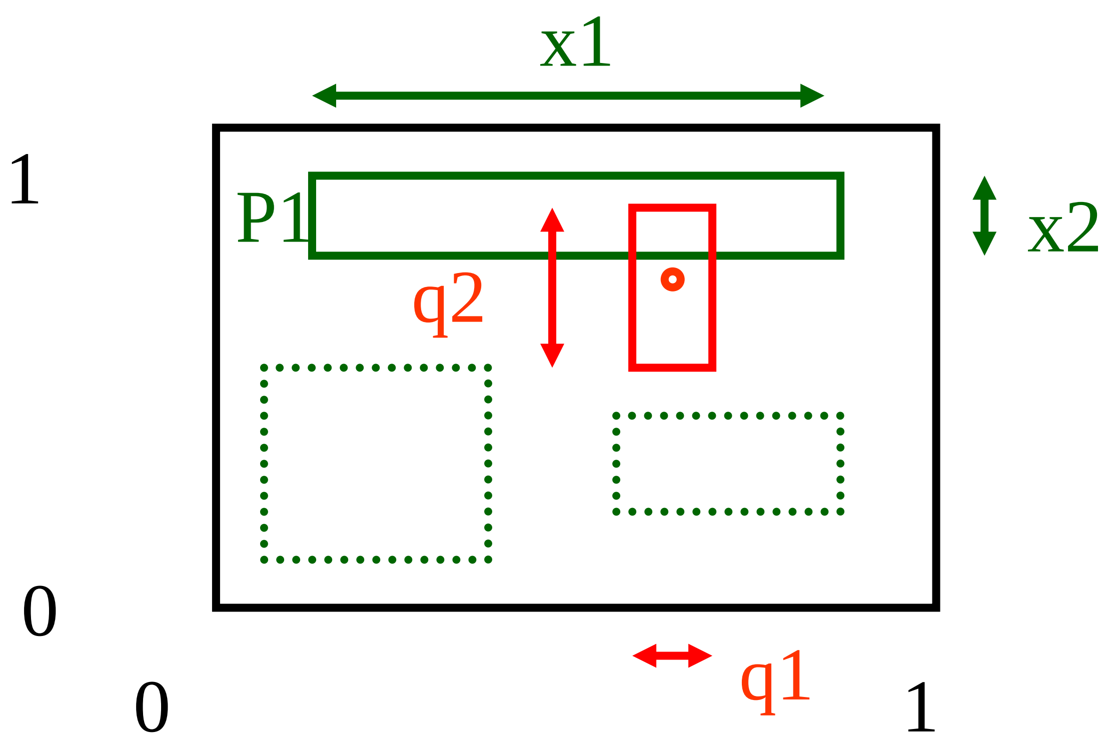
# R-trees: performance analysis

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



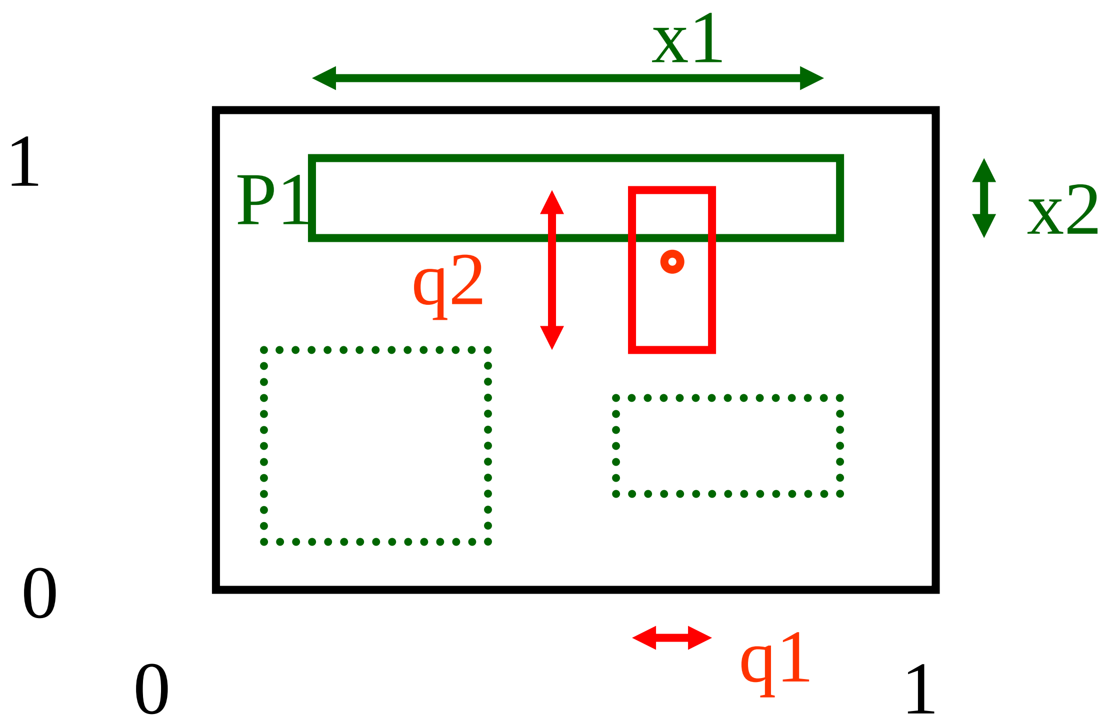
# R-trees: performance analysis

- 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

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

$$= \sum_i^n x_{i,1} * x_{i,2} + \longrightarrow \text{‘volume’}$$

$$q_1 \sum_i^n x_{i,2} + q_2 \sum_i^n x_{i,1} \longrightarrow \text{‘surface area’}$$

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

‘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).