

15-826: Multimedia (Databases) and Data Mining

Lecture #6: Spatial Access Methods

Part III: R-trees

C. Faloutsos

Must-read material

- MM-Textbook, Chapter 5.2
- Ramakrishnan+Gehrke, Chapter 28.6
- Guttman, A. (June 1984). [R-Trees: A Dynamic Index Structure for Spatial Searching](#). Proc. ACM SIGMOD, Boston, Mass.

R-trees – impact:

- Popular method; like multi-d B-trees
- guaranteed utilization; fast search (low dim' s)
- Used in practice:
 - Oracle spatial ([R-tree](#))
 - Postgres: `create index ... using gist`
 - Databricks ([R-trees and z-order](#))
 - Sqlite3: www.sqlite.org/rtree.html
 - Python: `(pip install rtree)`



Outline

Goal: ‘Find **similar / interesting** things’

- Intro to DB
- ➔ • Indexing - similarity search
- Data Mining



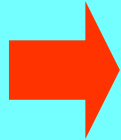
Indexing - Detailed outline

- primary key indexing
- secondary key / multi-key indexing
- spatial access methods
 - problem defn
 - z-ordering
 - ➔ – R-trees
 - ...
- text
- ...



Indexing - more detailed outline

- R-trees

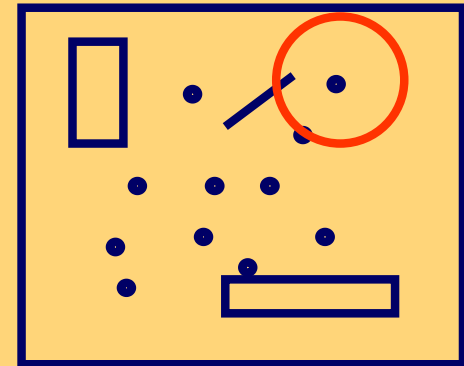


- main idea; file structure
- algorithms: insertion/split
- deletion
- search: range, nn, spatial joins
- performance analysis
- variations (packed; hilbert;...)



Spatial Access Methods - problem

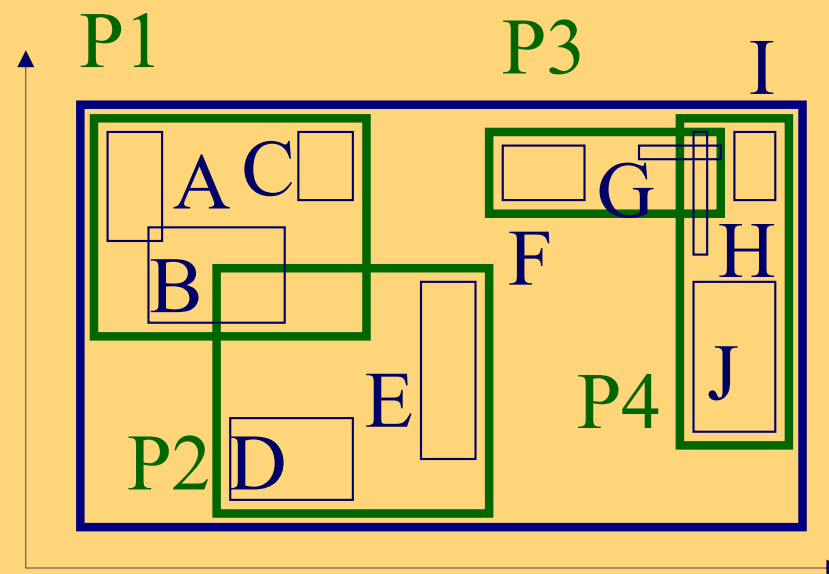
- Given a collection of geometric objects (points, lines, polygons, ...)
- Find cities within 100mi from Pittsburgh





Solution#2: R-trees

- multi-dim trees
- Allow nodes to overlap
- Guaranteed 50% utilization



R-trees

- z-ordering: cuts regions to pieces -> dup. elim.
- how could we avoid that?
- Idea: try to extend/merge B-trees and k-d trees

R-trees

- [Guttman 84] Main idea: allow parents to overlap!

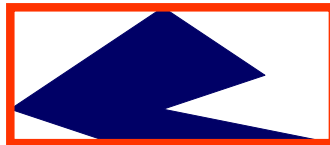


Antonin Guttman

[<https://dblp.org/pid/81/3404.html>]

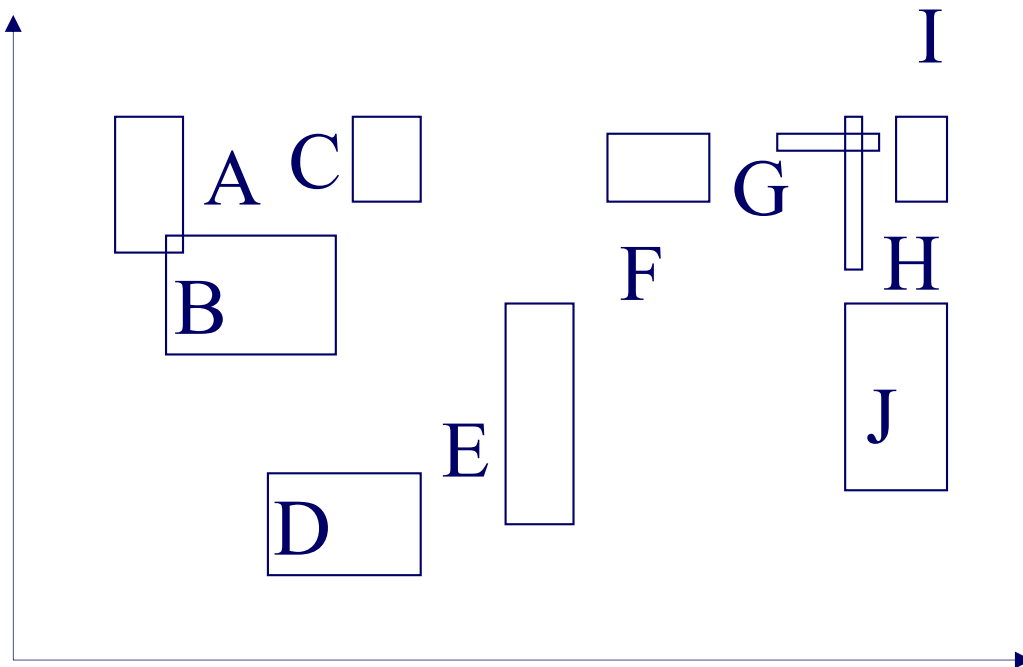
R-trees

- [Guttman 84] Main idea: allow parents to overlap!
 - $\Rightarrow$ guaranteed 50% utilization
 - $\Rightarrow$ easier insertion/split algorithms.
 - (only deal with Minimum Bounding Rectangles - **MBRs**)



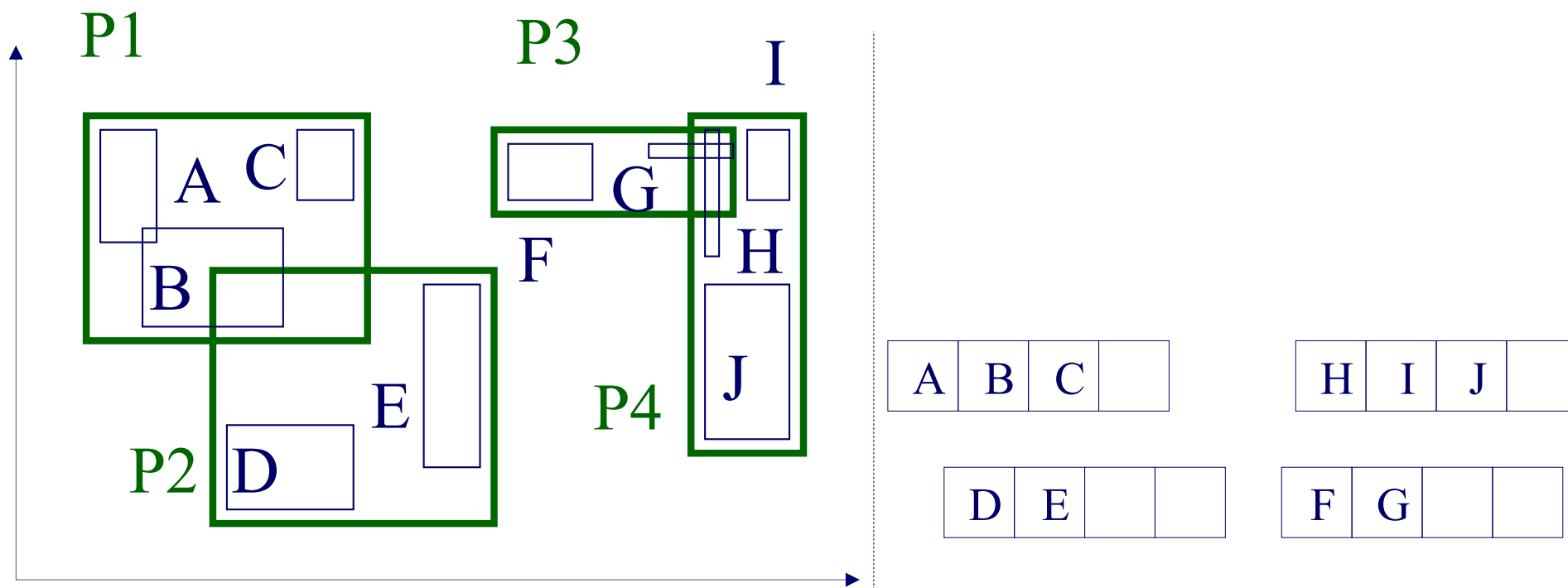
R-trees

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



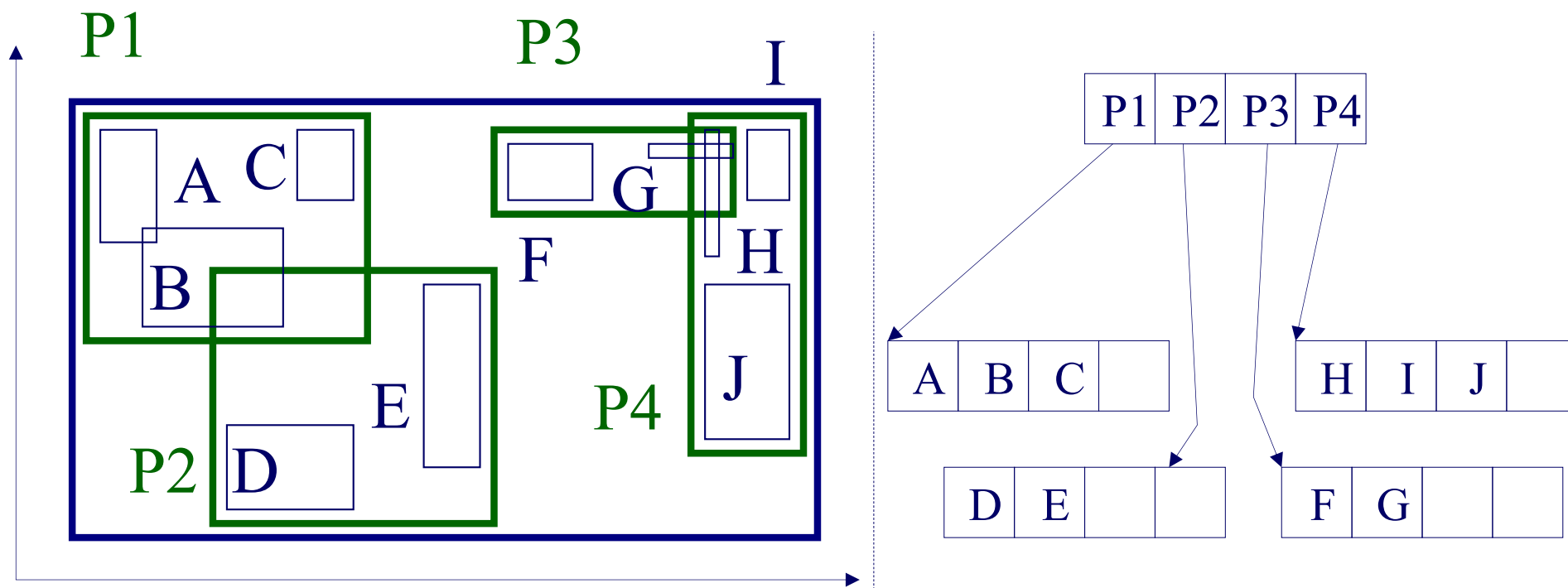
R-trees

- eg., w/ fanout 4:



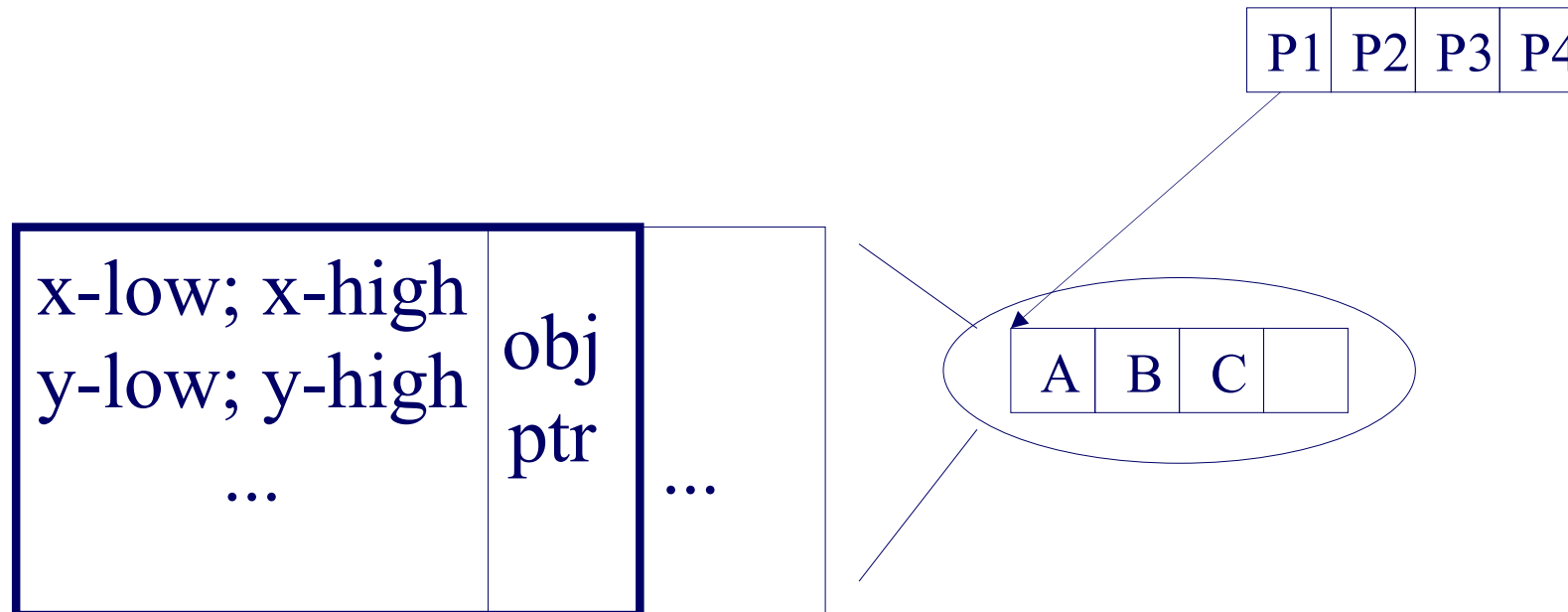
R-trees

- eg., w/ fanout 4:



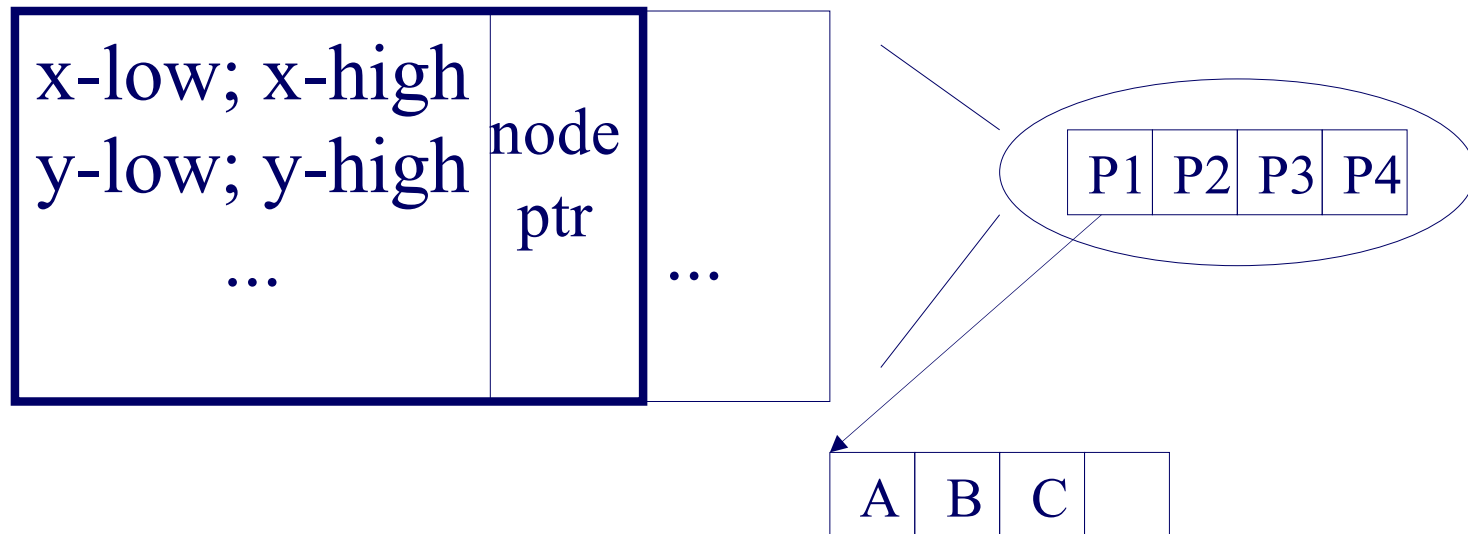
R-trees - format of nodes

- $\{(MBR; \text{obj-ptr})\}$ for leaf nodes

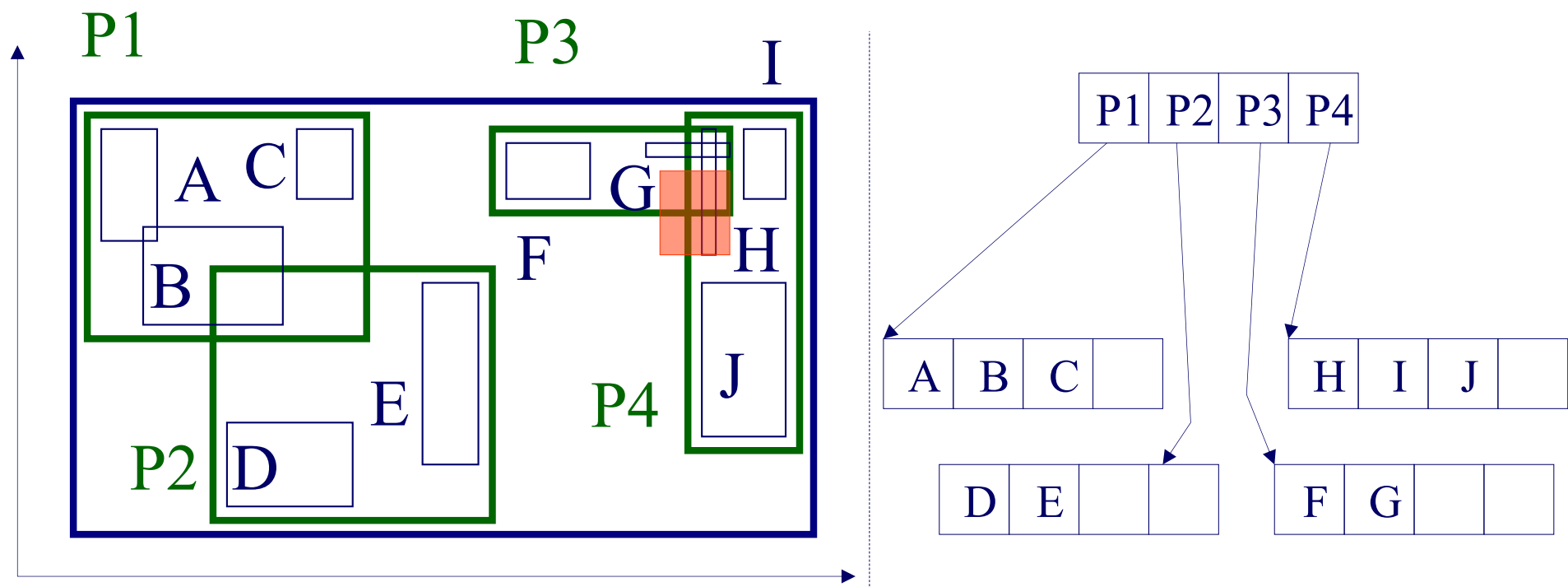


R-trees - format of nodes

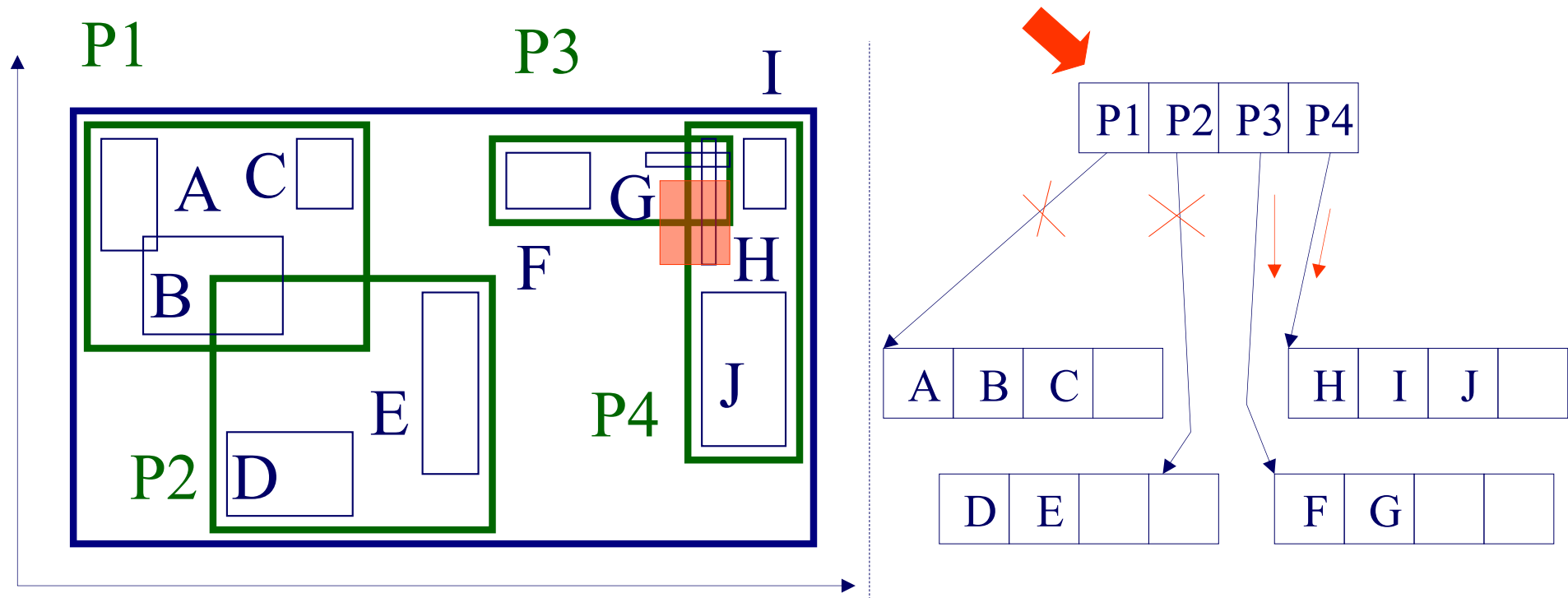
- $\{(MBR; \text{node-ptr})\}$ for non-leaf nodes



R-trees - range search?



R-trees - range search?



R-trees - range search

Observations:

- every parent node completely covers its 'children'
- a child MBR may be covered by more than one parent - it is stored under ONLY ONE of them. (ie., no need for dup. elim.)

R-trees - range search

Observations - cont' d

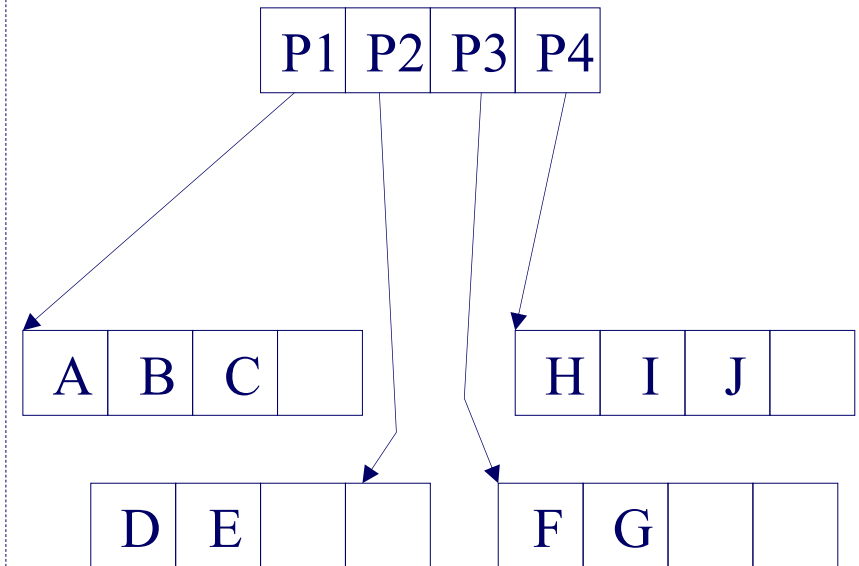
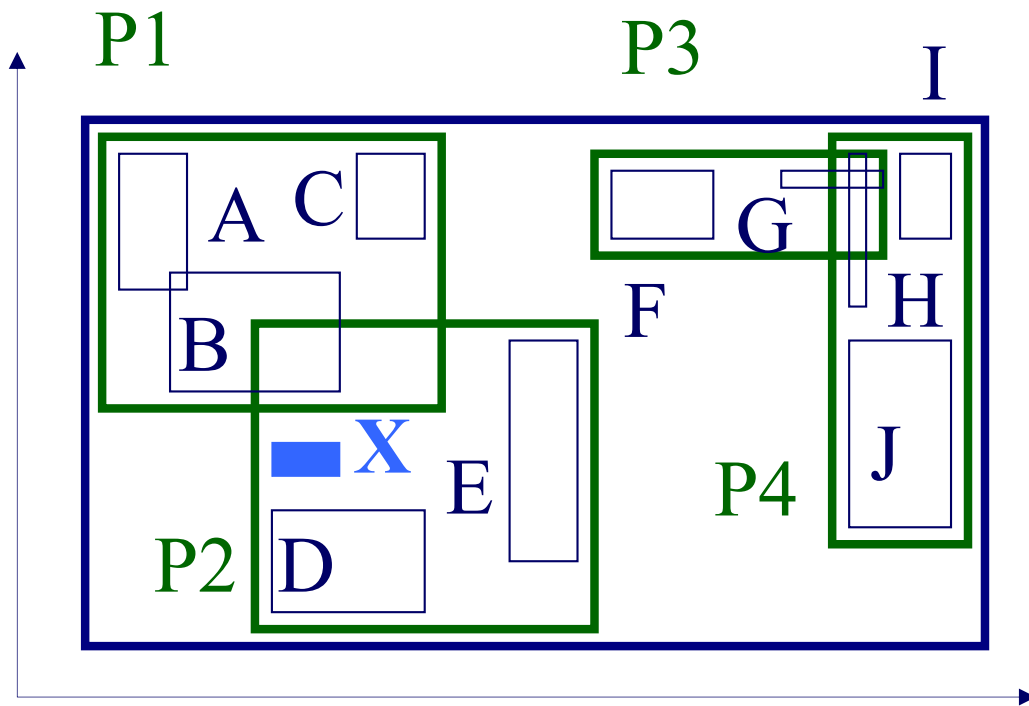
- a point query may follow multiple branches.
- everything works for **any** dimensionality

Indexing - more detailed outline

- R-trees
 - main idea; file structure
 -  – algorithms: insertion/split
 - deletion
 - search: range, nn, spatial joins
 - performance analysis
 - variations (packed; hilbert;...)

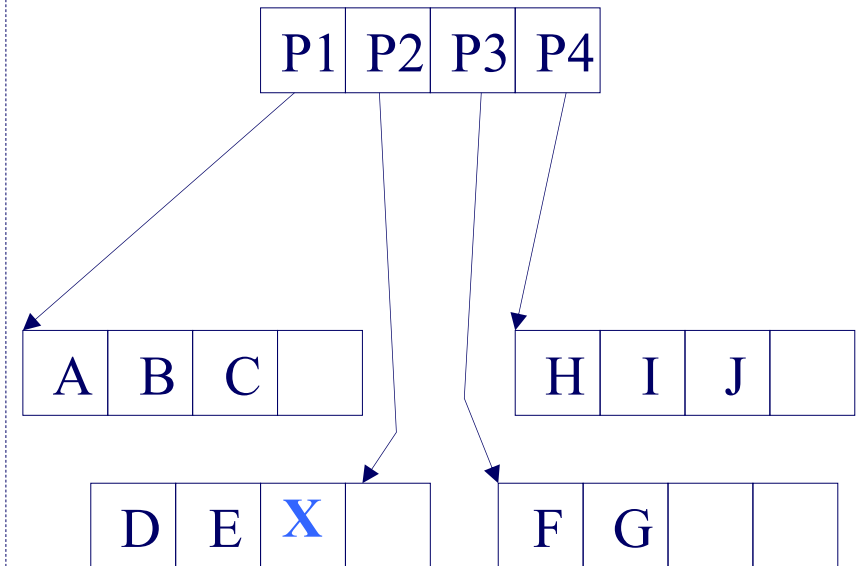
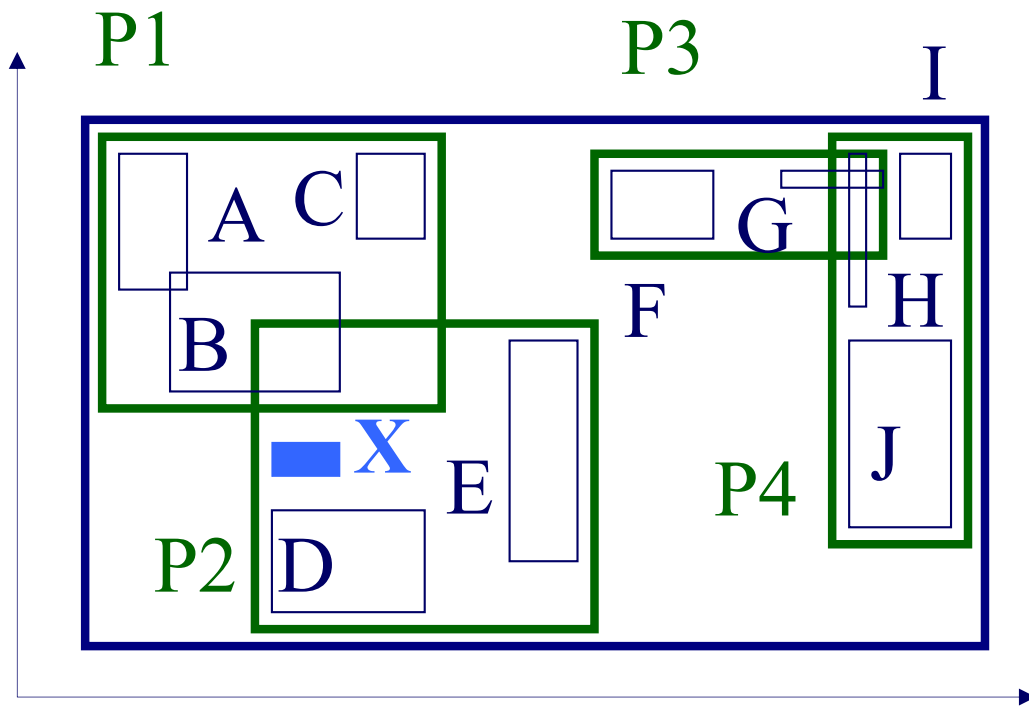
R-trees - insertion

- eg., rectangle 'X'



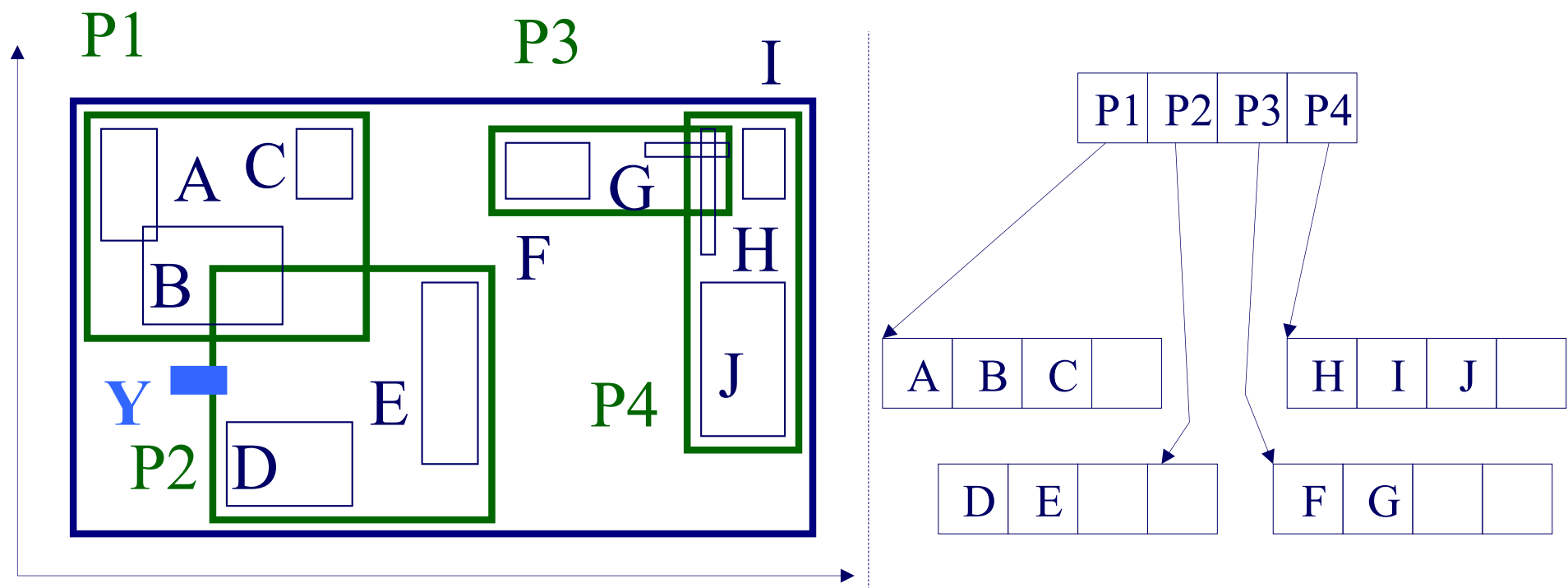
R-trees - insertion

- eg., rectangle 'X'



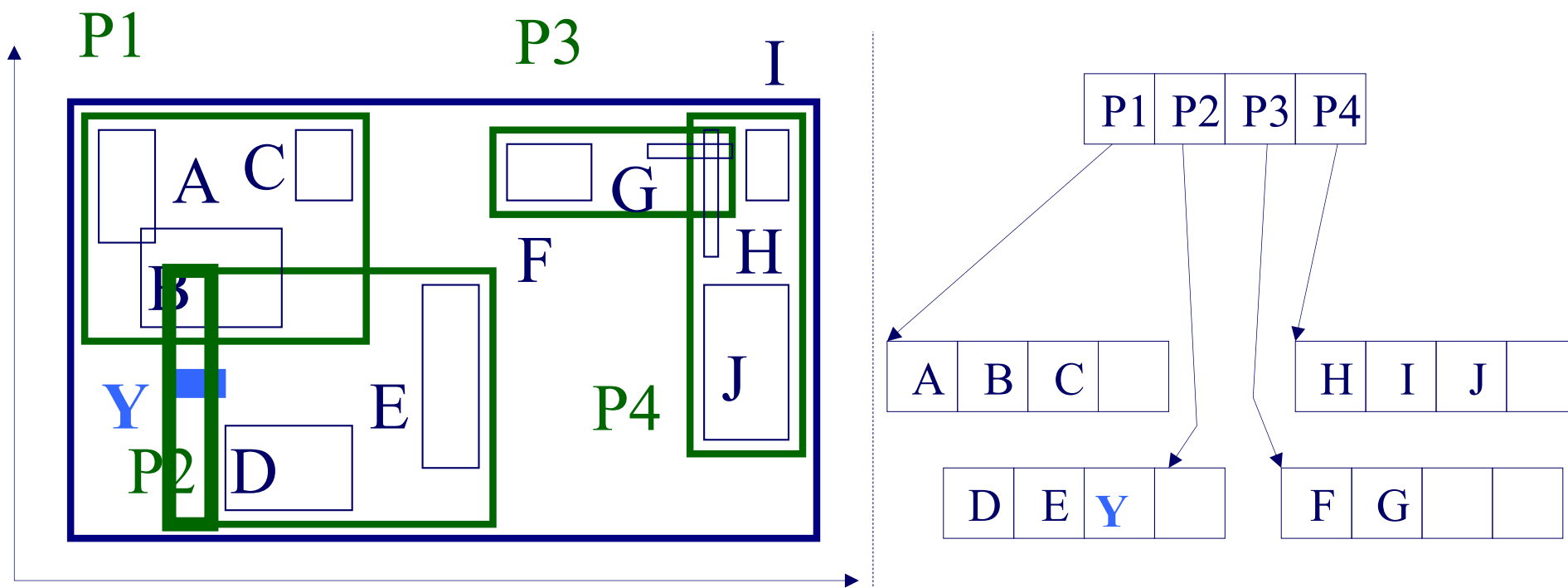
R-trees - insertion

- eg., rectangle 'Y'



R-trees - insertion

- eg., rectangle 'Y' : extend suitable parent.



R-trees - insertion

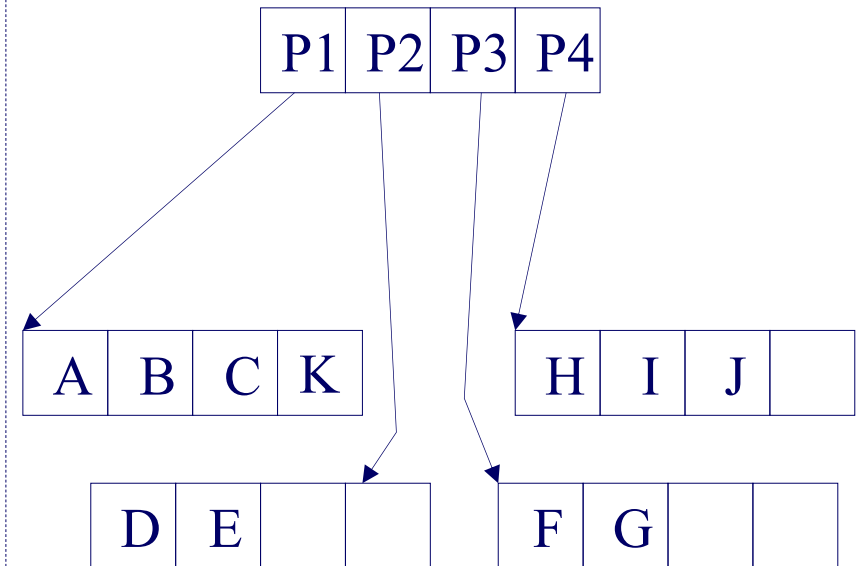
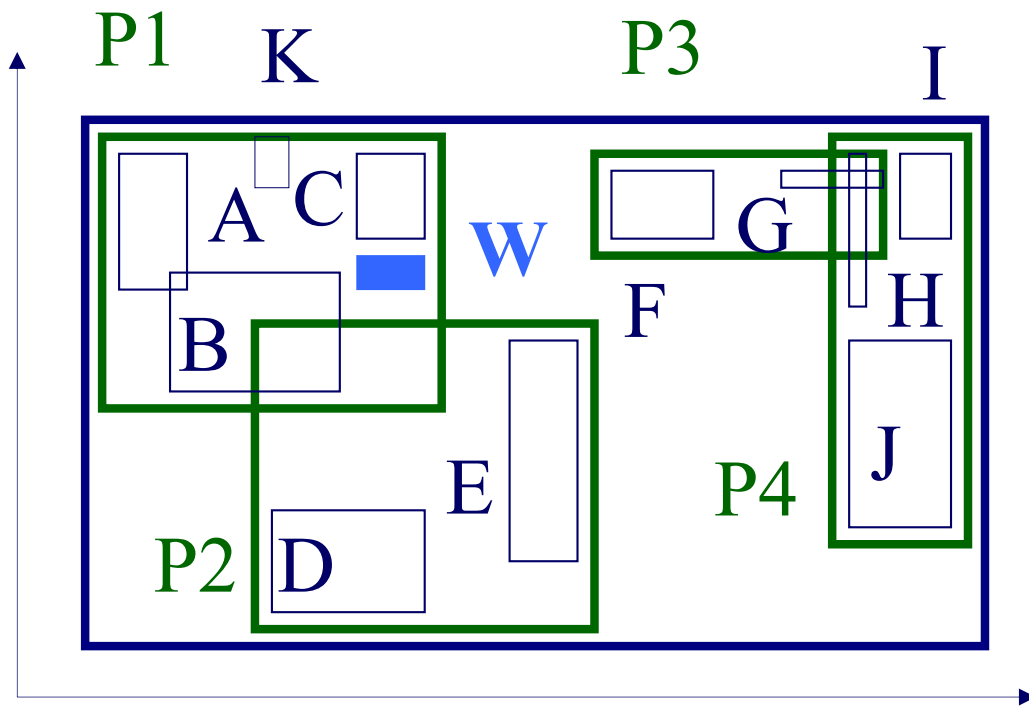
- eg., rectangle 'Y' : extend suitable parent.
- Q: how to measure 'suitability' ?

R-trees - insertion

- eg., rectangle ‘Y’ : extend suitable parent.
- Q: how to measure ‘suitability’ ?
- A: by increase in area (volume) (more details: later, under ‘performance analysis’)
- Q: what if there is no room? how to split?

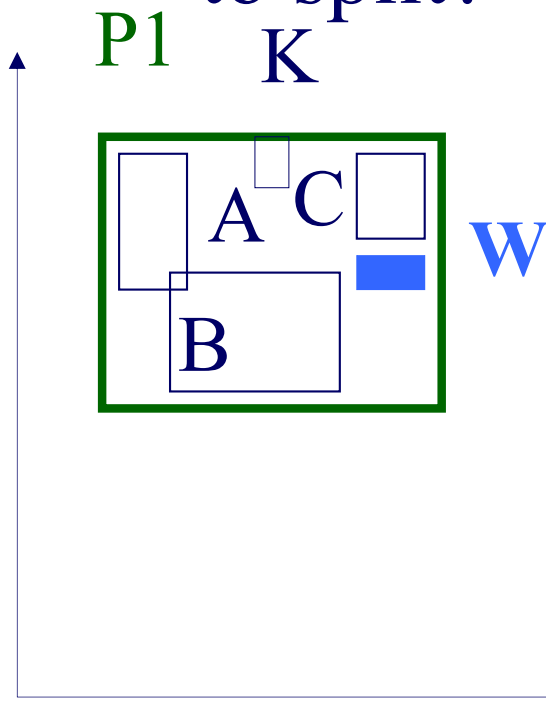
R-trees - insertion

- eg., rectangle 'W'



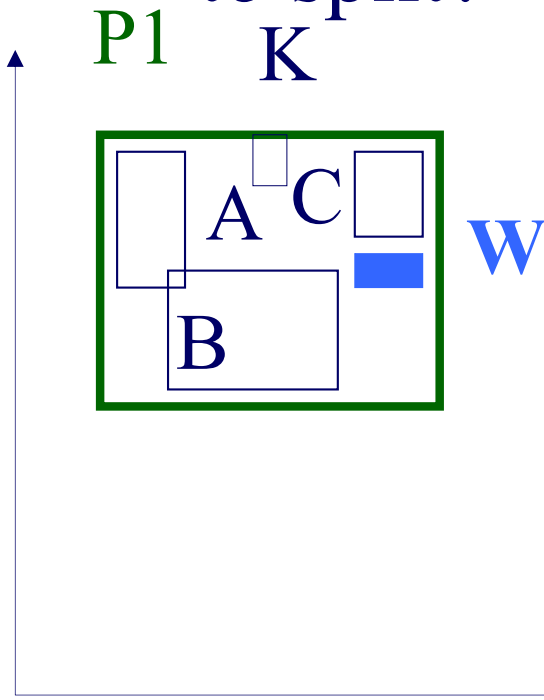
R-trees - insertion

- eg., rectangle 'W' - focus on 'P1' - how to split?



R-trees - insertion

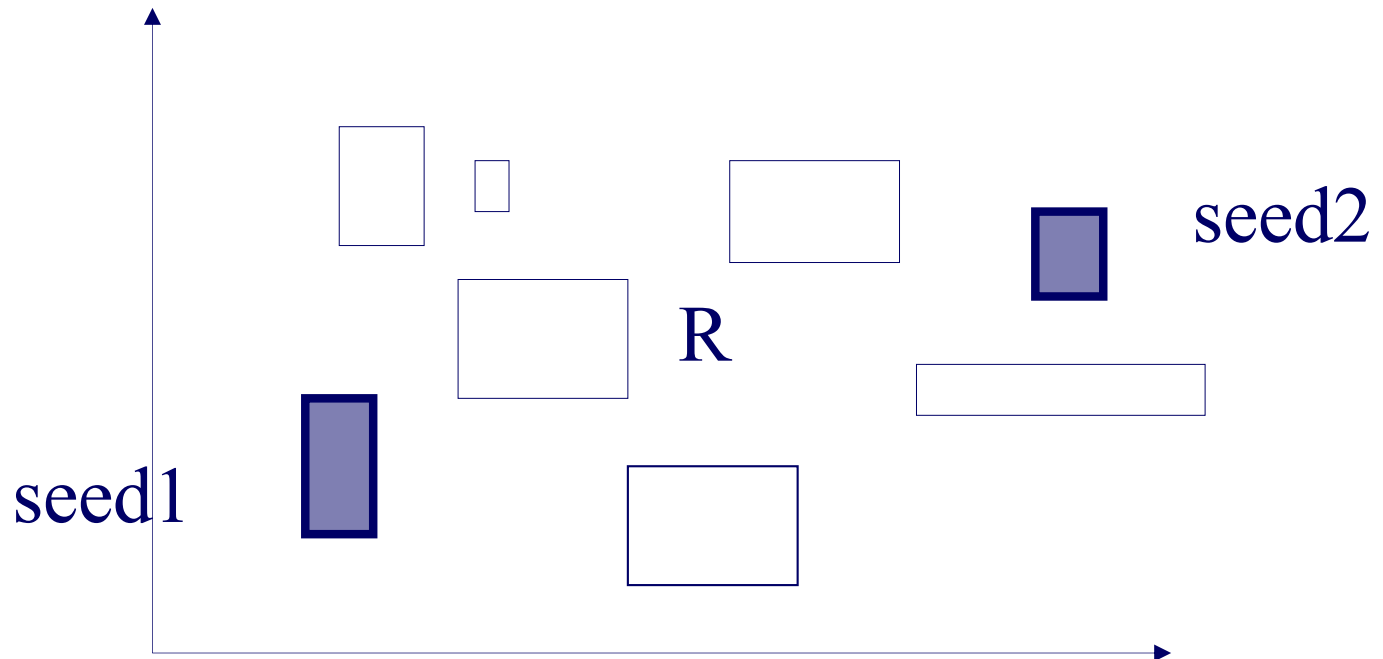
- eg., rectangle 'W' - focus on 'P1' - how to split?



- (A1: plane sweep, until 50% of rectangles)
- A2: 'linear' split
- ➔ • A3: quadratic split
- A4: exponential split

R-trees - insertion & split

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



R-trees - insertion & split

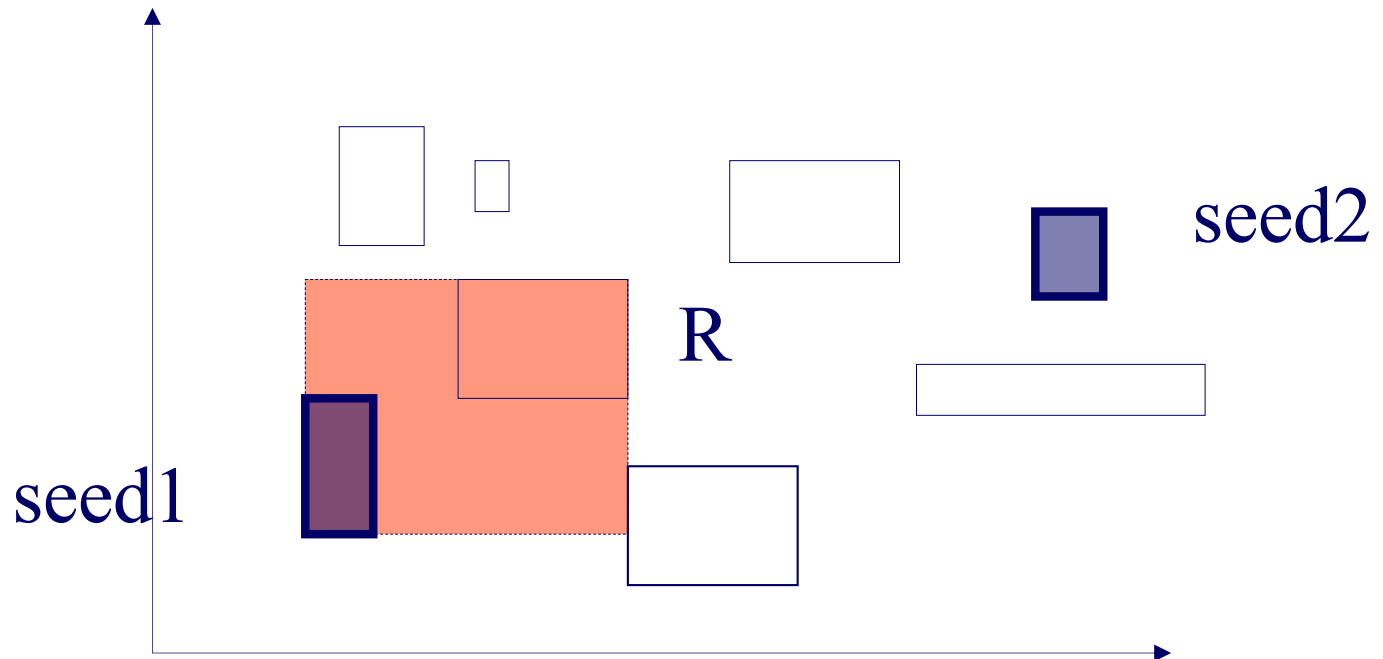
- pick two rectangles as ‘seeds’ ;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- Q: how to measure ‘closeness’ ?

R-trees - insertion & split

- pick two rectangles as ‘seeds’ ;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- Q: how to measure ‘closeness’ ?
- A: by increase of area (volume)

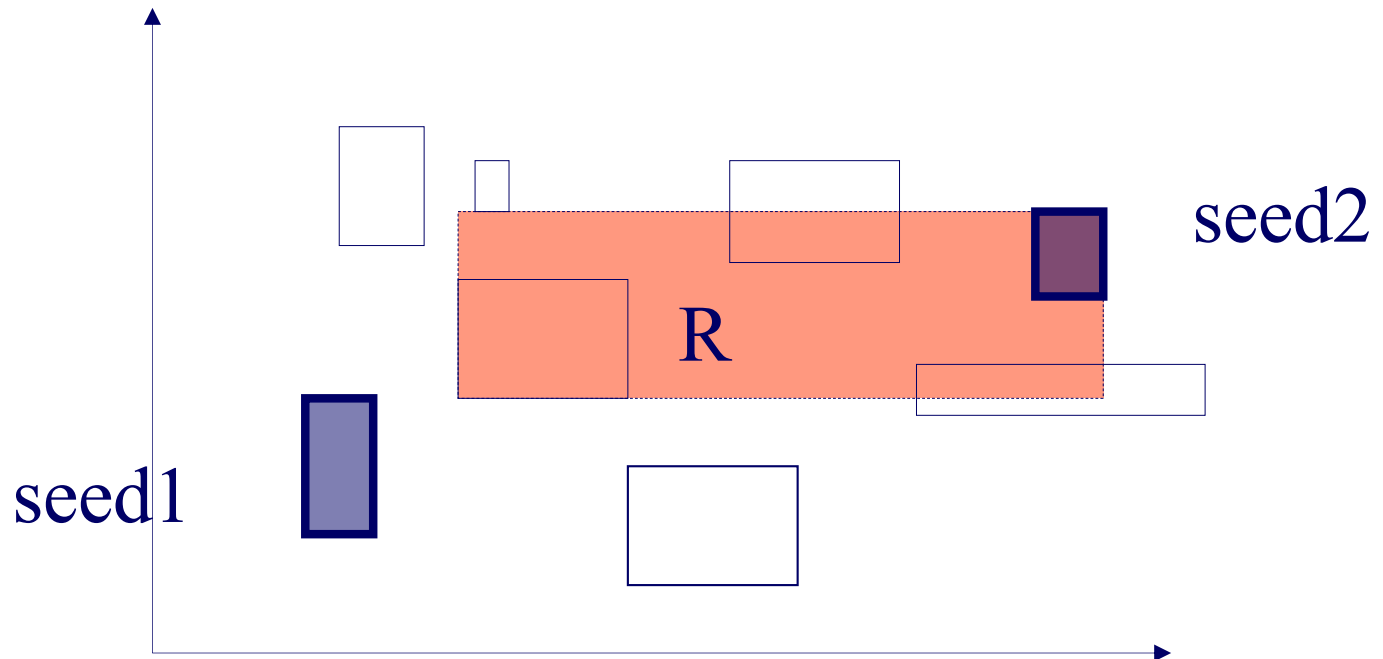
R-trees - insertion & split

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



R-trees - insertion & split

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



R-trees - insertion & split

- pick two rectangles as ‘seeds’ ;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- smart idea: pre-sort rectangles according to delta of closeness (ie., schedule easiest choices first!)

R-trees - insertion - pseudocode

- decide which parent to put new rectangle into (‘closest’ parent)
- if overflow, split to two, using (say,) the quadratic split algorithm
 - propagate the split upwards, if necessary
- update the MBRs of the affected parents.

R-trees - insertion - observations

- **many** more split algorithms exist (see refs)

Indexing - more detailed outline

- R-trees
 - main idea; file structure
 - algorithms: insertion/split
 -  – deletion
 - search: range, nn, spatial joins
 - performance analysis
 - variations (packed; hilbert;...)

R-trees - deletion

- delete rectangle
- if underflow
 - ??

R-trees - deletion

- delete rectangle
- if underflow
 - temporarily delete all siblings (!);
 - delete the parent node and
 - re-insert them



Indexing - more detailed outline

- R-trees
 - main idea; file structure
 - algorithms: insertion/split
 - deletion
 - ➔ – search: range, nn, spatial joins
 - performance analysis
 - variations (packed; hilbert;...)

R-trees - range search

pseudocode:

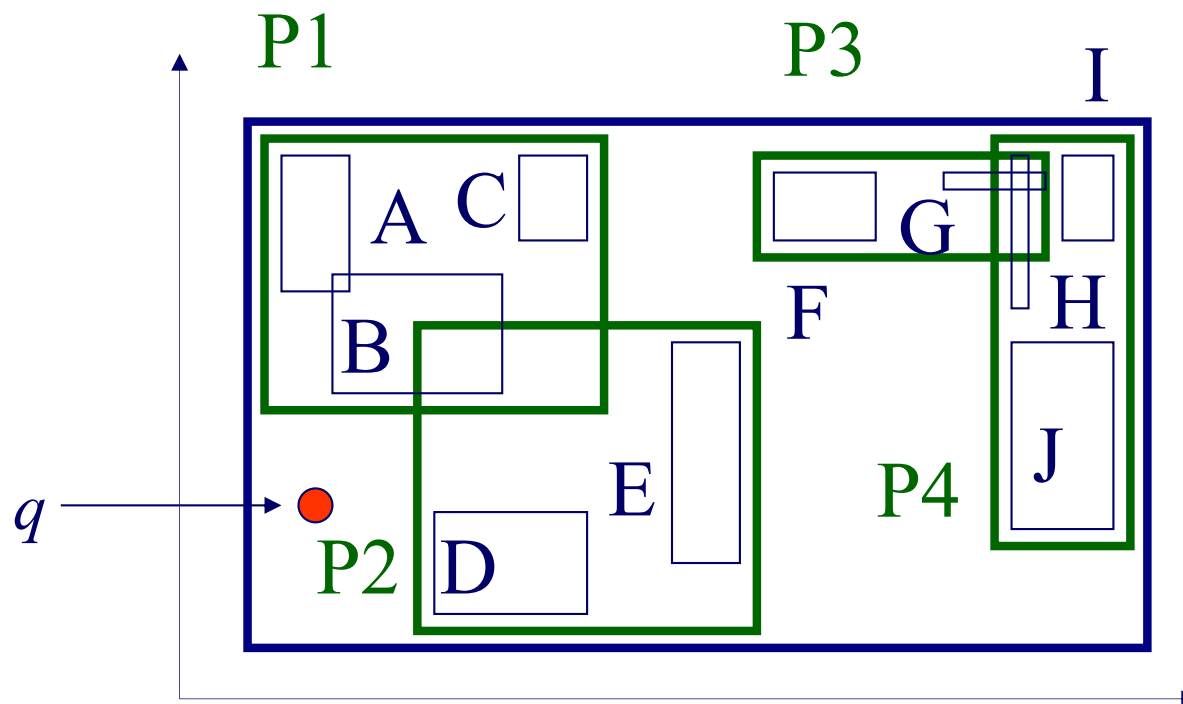
- check the root

- for each branch,

 - if its MBR intersects the query rectangle

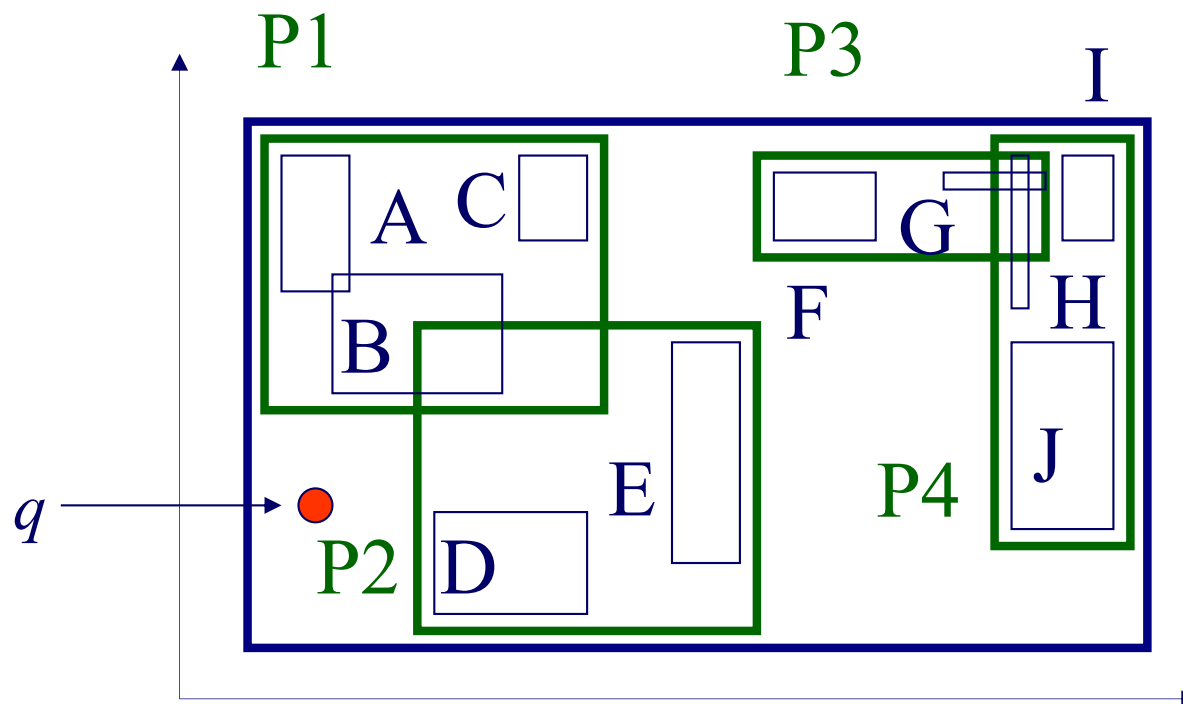
 - apply range-search (or print out, if this is a leaf)

R-trees - nn search



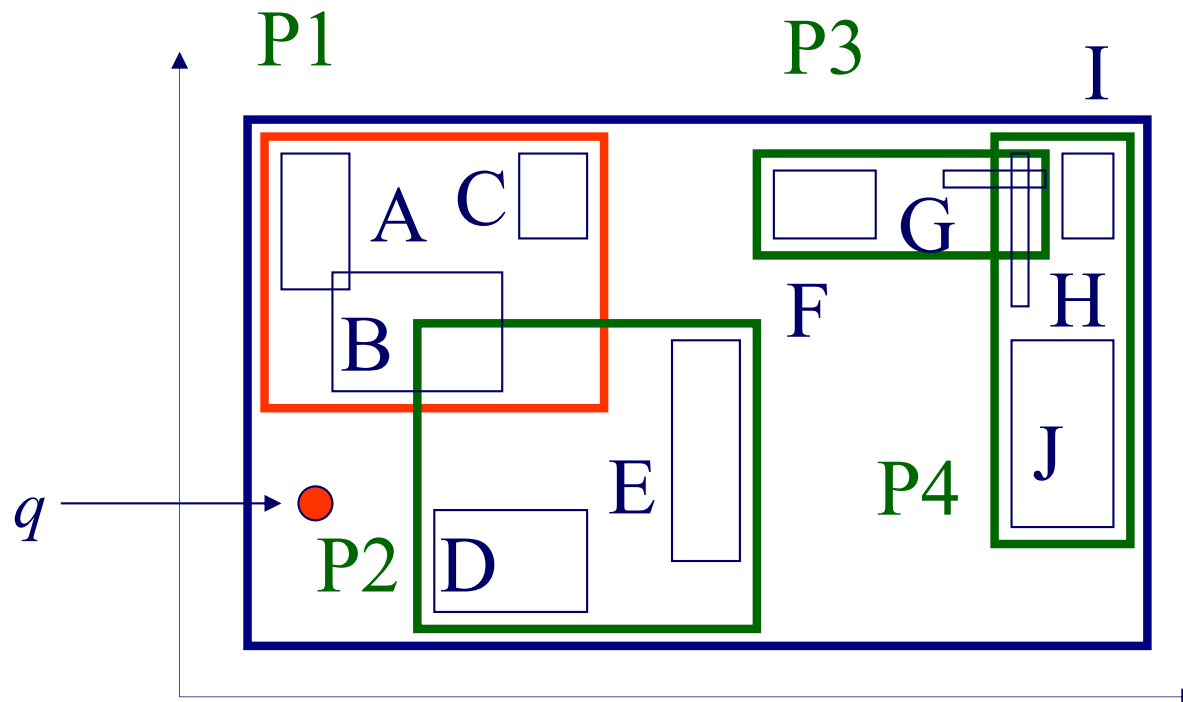
R-trees - nn search

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



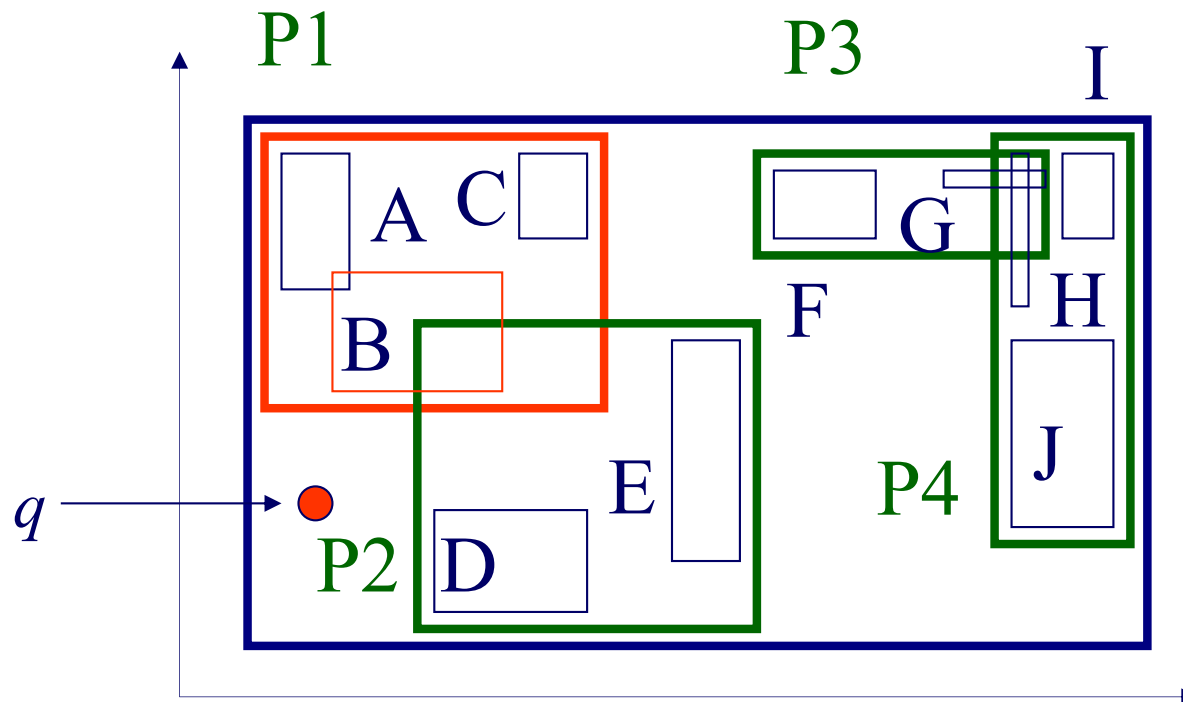
R-trees - nn search

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



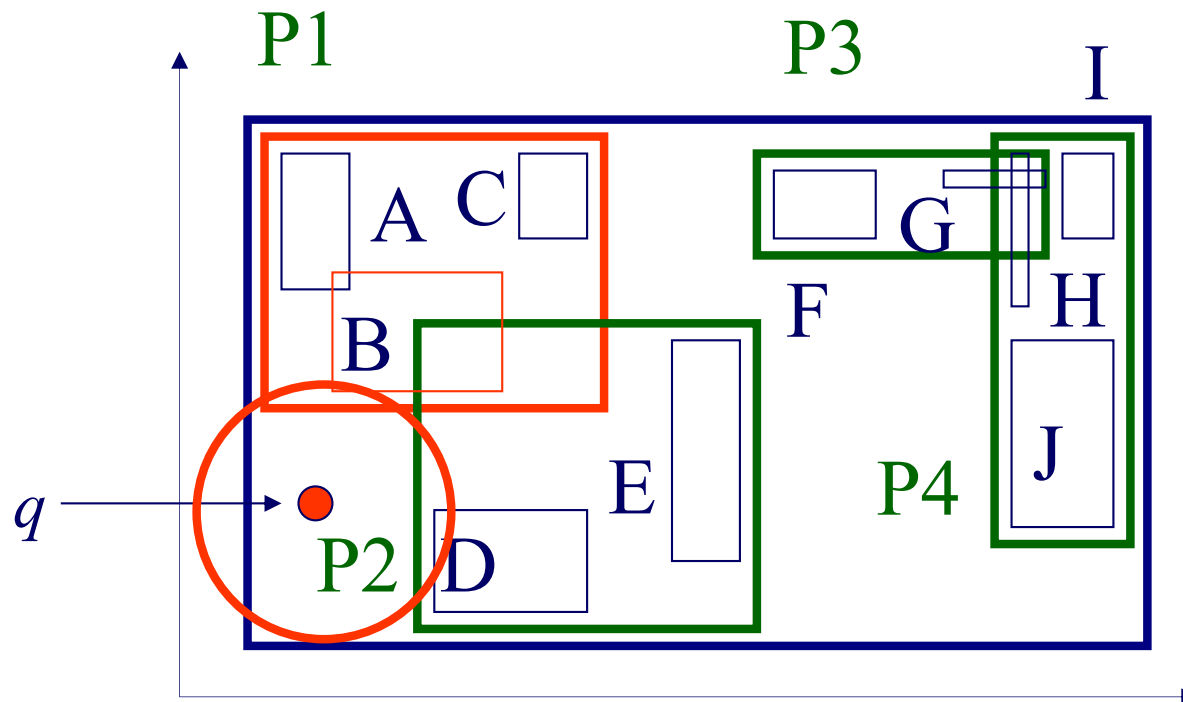
R-trees - nn search

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



R-trees - nn search

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

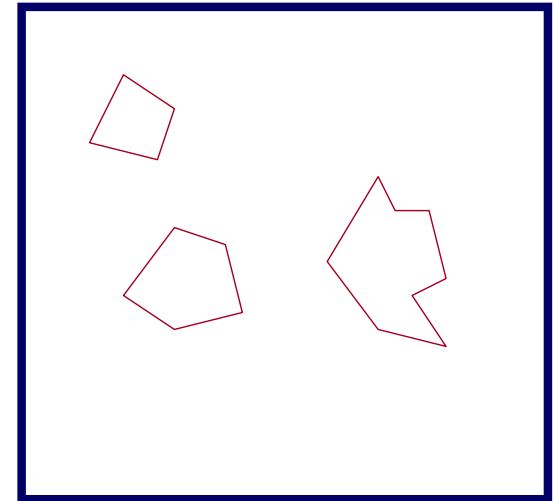
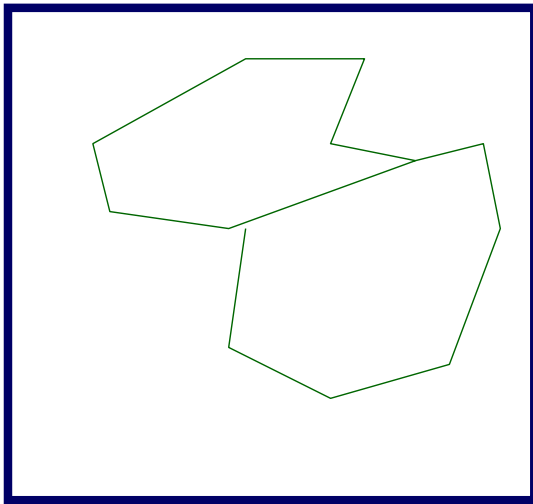


Indexing - more detailed outline

- R-trees
 - main idea; file structure
 - algorithms: insertion/split
 - deletion
 -  – search: range, nn, **spatial joins**
 - performance analysis
 - variations (packed; hilbert;...)

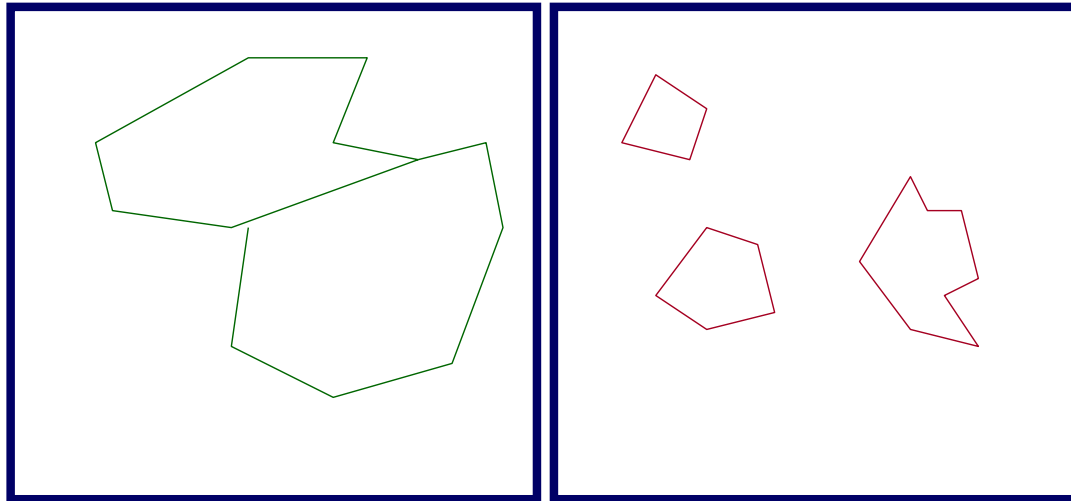
R-trees - spatial joins

Spatial joins: find (quickly) all
counties intersecting **lakes**



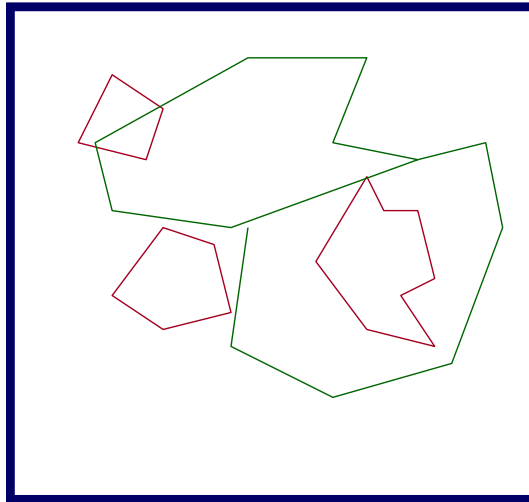
R-trees - spatial joins

Spatial joins: find (quickly) all
counties intersecting **lakes**



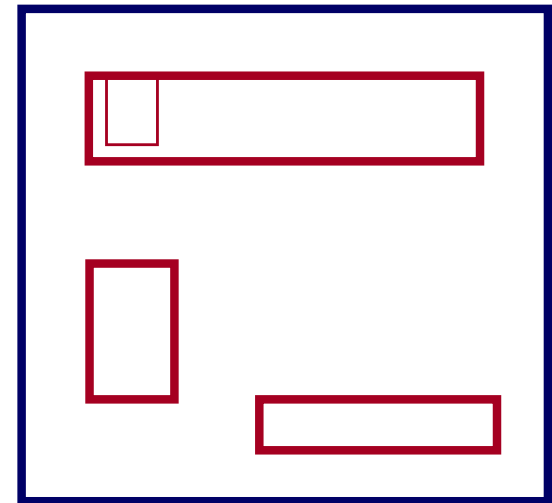
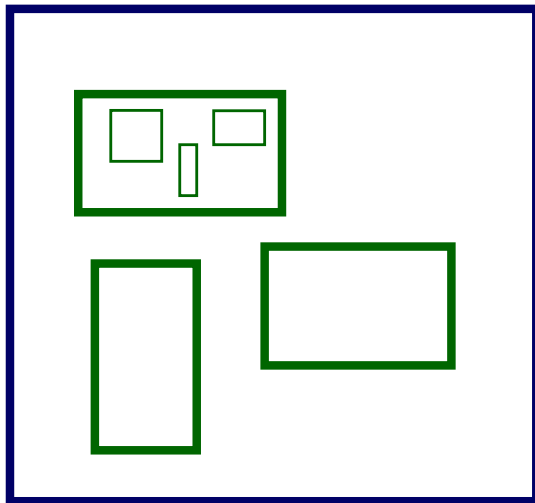
R-trees - spatial joins

Spatial joins: find (quickly) all
counties intersecting lakes



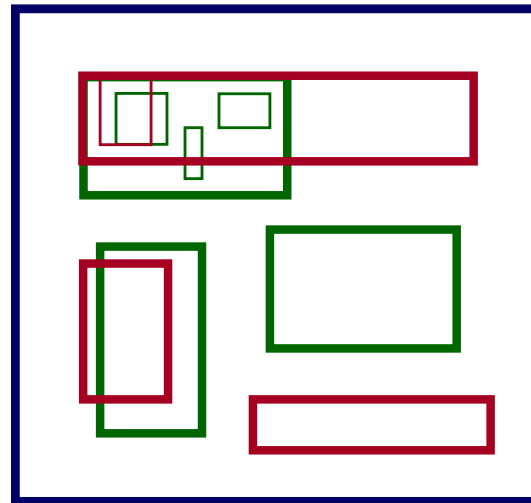
R-trees - spatial joins

Assume that they are both organized in R-trees:



R-trees - spatial joins

Assume that they are both organized in R-trees:



R-trees - spatial joins

for each parent P1 of tree T1
 for each parent P2 of tree T2
 if their MBRs intersect,
 process them recursively (ie., check their
 children)

R-trees - spatial joins

Improvements - variations:

- [Seeger+, sigmod 92]: do some pre-filtering; do plane-sweeping to avoid $N1 * N2$ tests for intersection
- [Lo & Ravishankar, sigmod 94]: ‘seeded’ R-trees
(FYI, many more papers on spatial joins, without R-trees: [Koudas+ Sevcik], e.t.c.)



Indexing - more detailed outline

- R-trees
 - main idea; file structure
 - algorithms: insertion/split
 - deletion
 - search: range, nn, spatial joins
 - ➔ – performance analysis
 - variations (packed; hilbert;...)

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

- How many disk (=node) accesses we'll need for
 - range
 - nn
 - spatial joins
- why does it matter?
- A: because we can design split etc algorithms accordingly; also, do query-optimization

R-trees - performance analysis

- How many disk (=node) accesses we'll need for
 - ➔ – range
 - nn
 - spatial joins
- why does it matter?
- A: because we can design split etc algorithms accordingly; also, do query-optimization



R-trees - performance analysis

- 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

- Thus, given a tree with N nodes ($i=1, \dots, N$) we expect

$$\text{\#DiskAccesses}(q1, q2) =$$

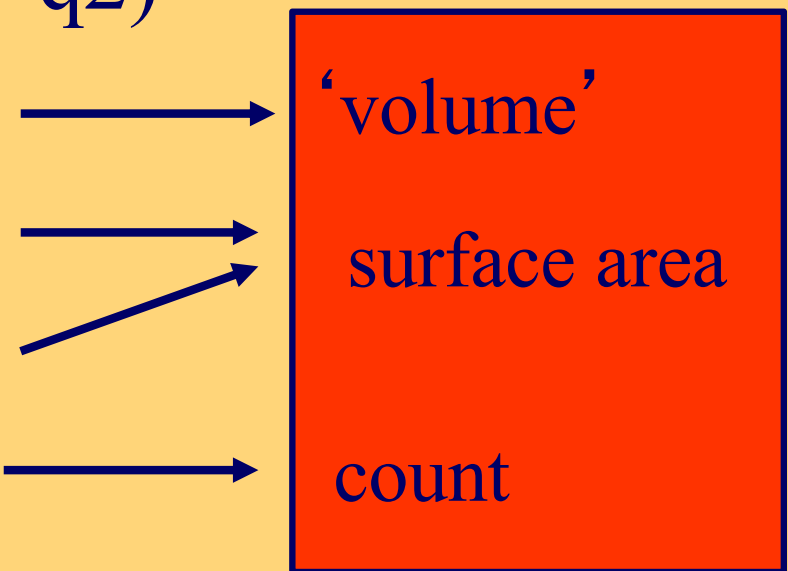
$$\text{sum} (x_{i,1} + q1) * (x_{i,2} + q2)$$

$$= \text{sum} (x_{i,1} * x_{i,2}) +$$

$$q2 * \text{sum} (x_{i,1}) +$$

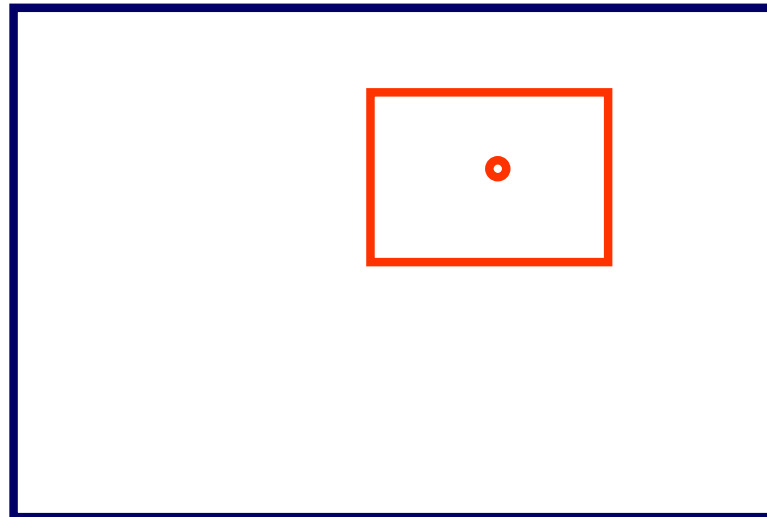
$$q1 * \text{sum} (x_{i,2})$$

$$q1 * q2 * N$$



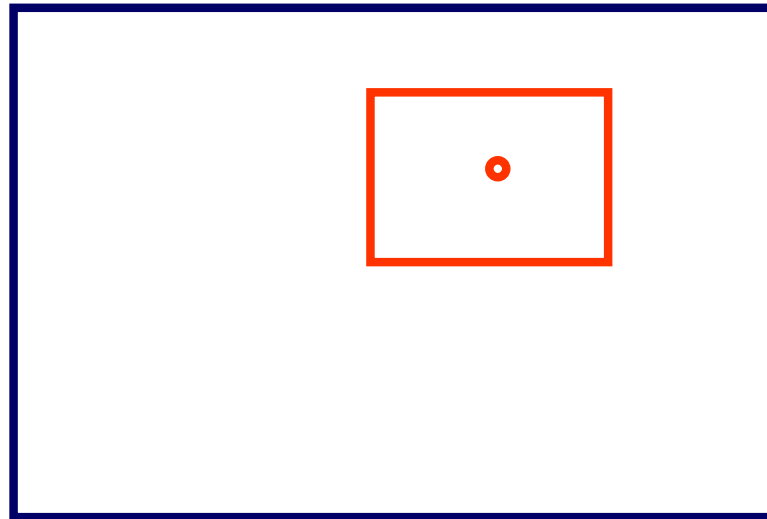
R-trees - performance analysis

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



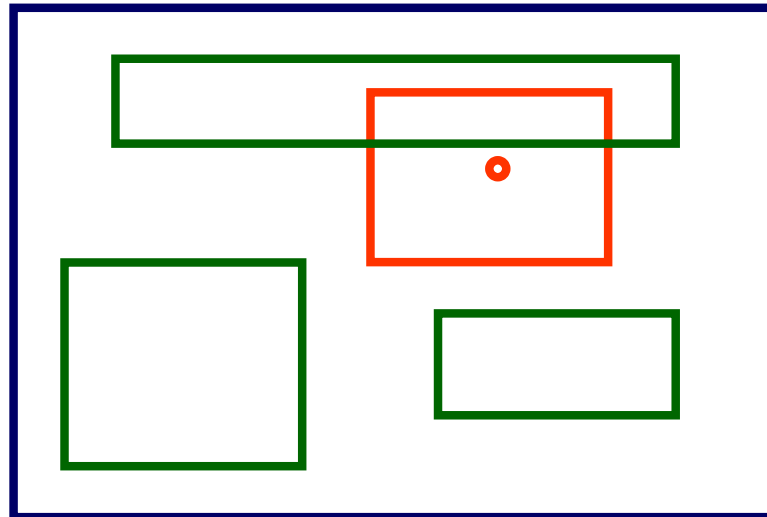
R-trees - performance analysis

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



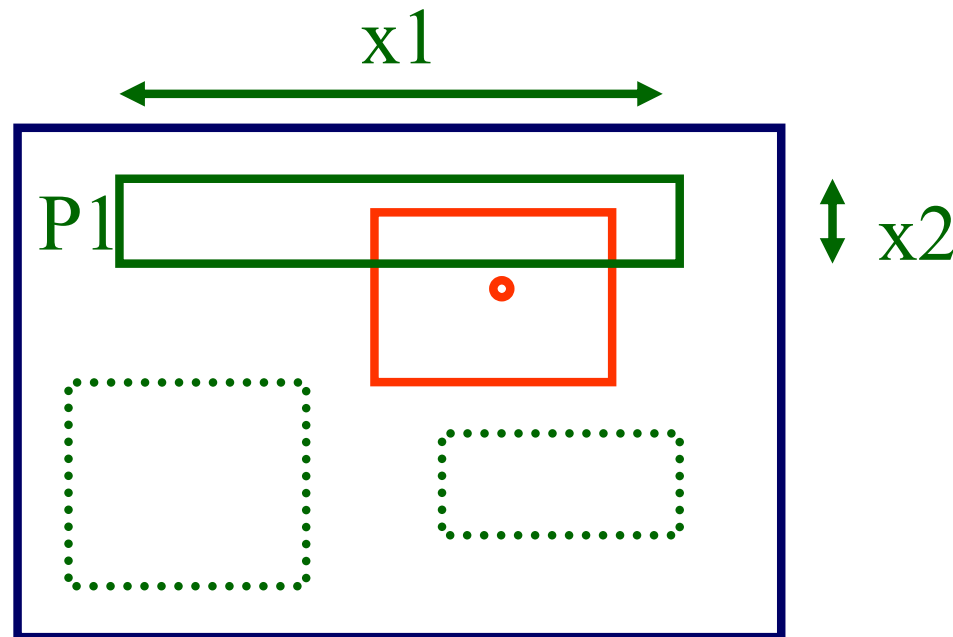
R-trees - performance analysis

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



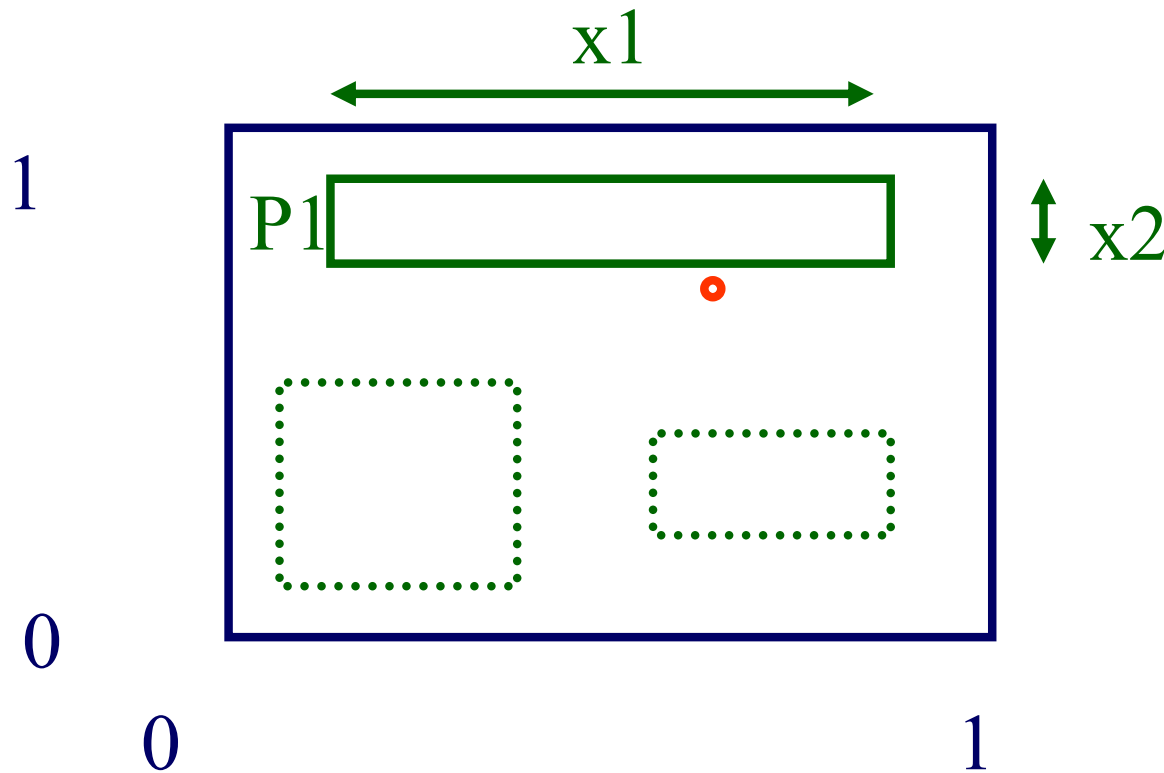
R-trees - performance analysis

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



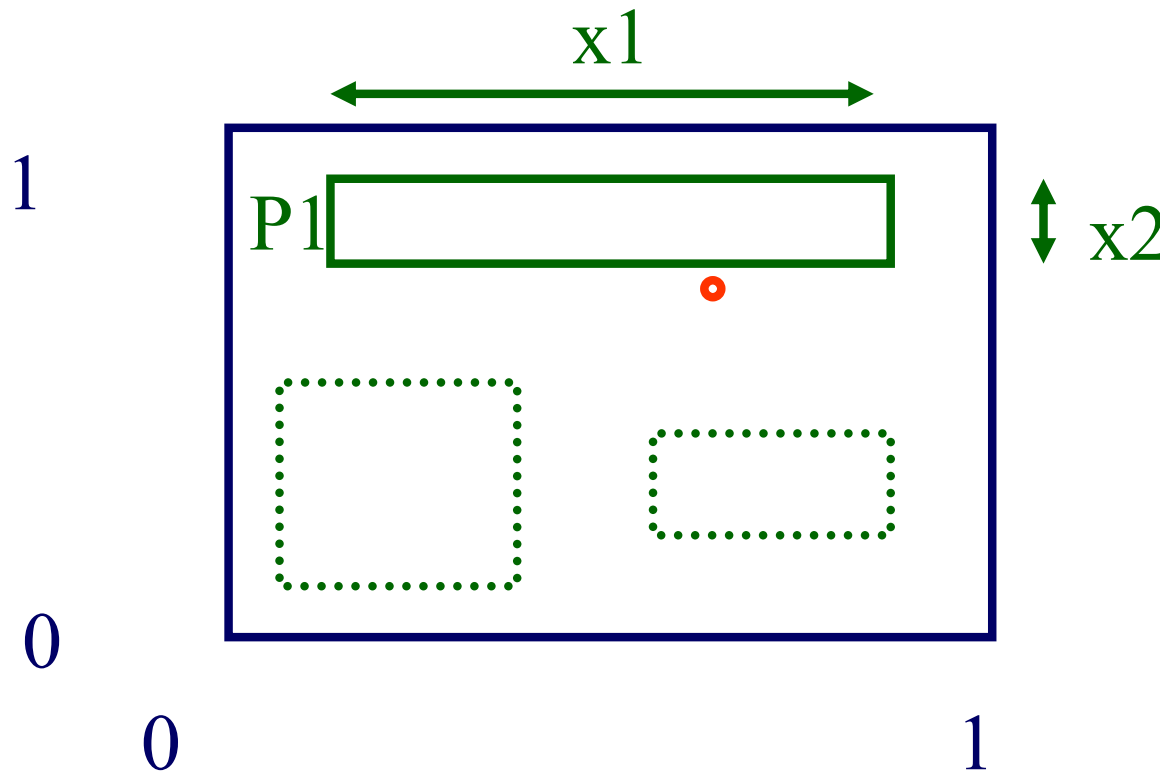
R-trees - performance analysis

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



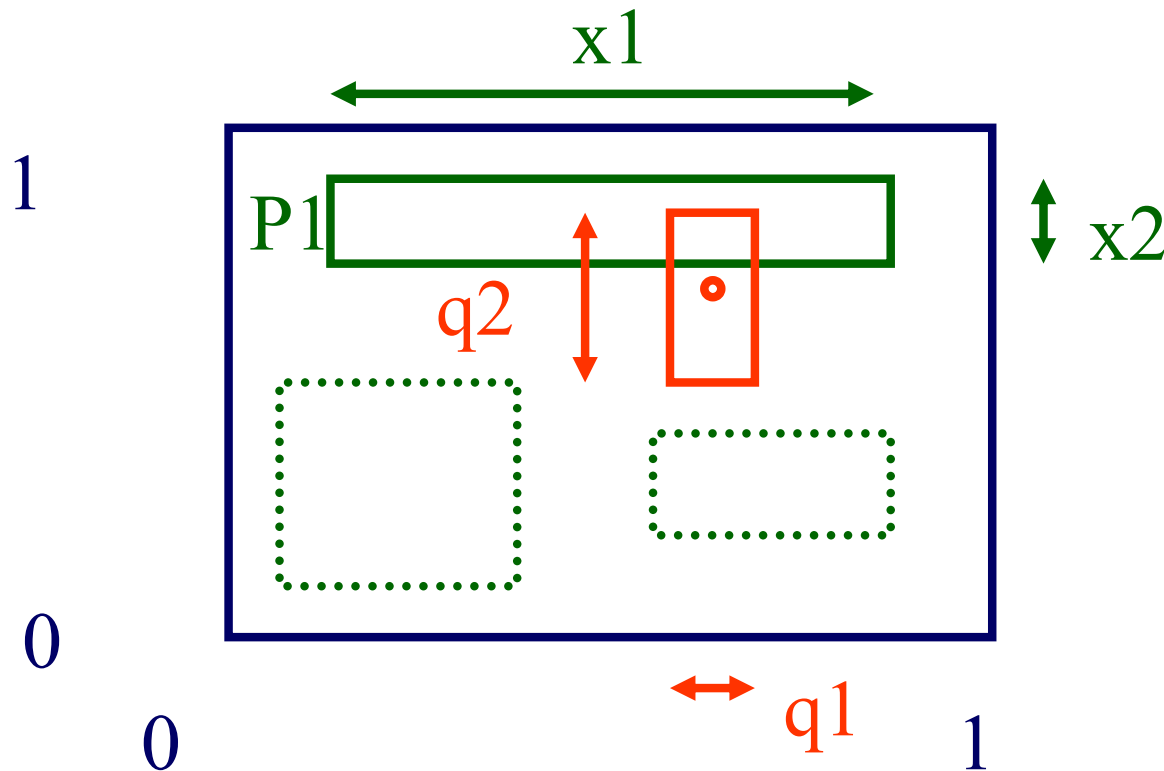
R-trees - performance analysis

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



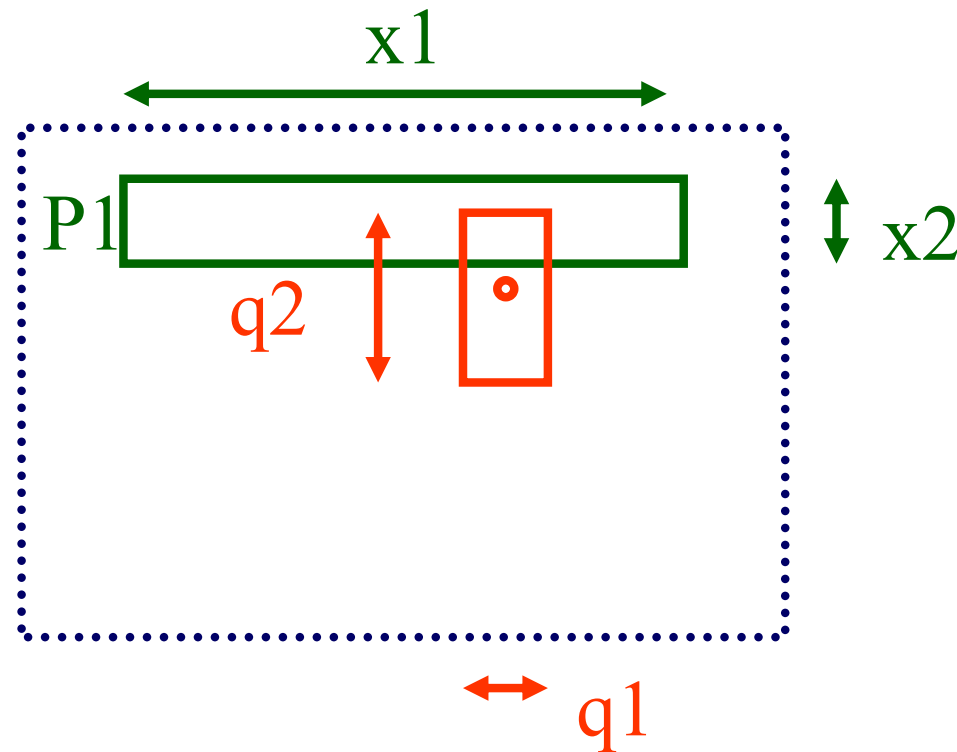
R-trees - performance analysis

- How many times will P1 be retrieved (unif. queries of size $q_1 \times q_2$)?



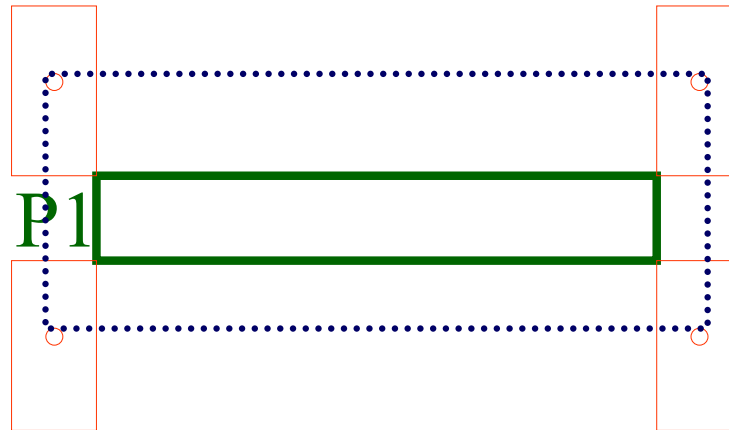
R-trees - performance analysis

- How many times will P1 be retrieved (unif. queries of size $q_1 \times q_2$)?



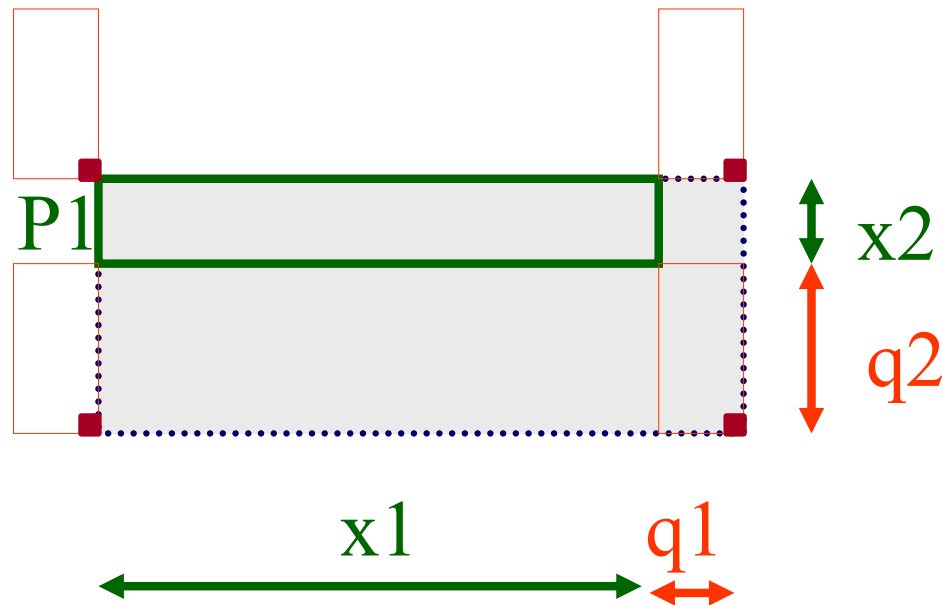
R-trees - performance analysis

- How many times will P1 be retrieved (unif. queries of size $q_1 \times q_2$)?



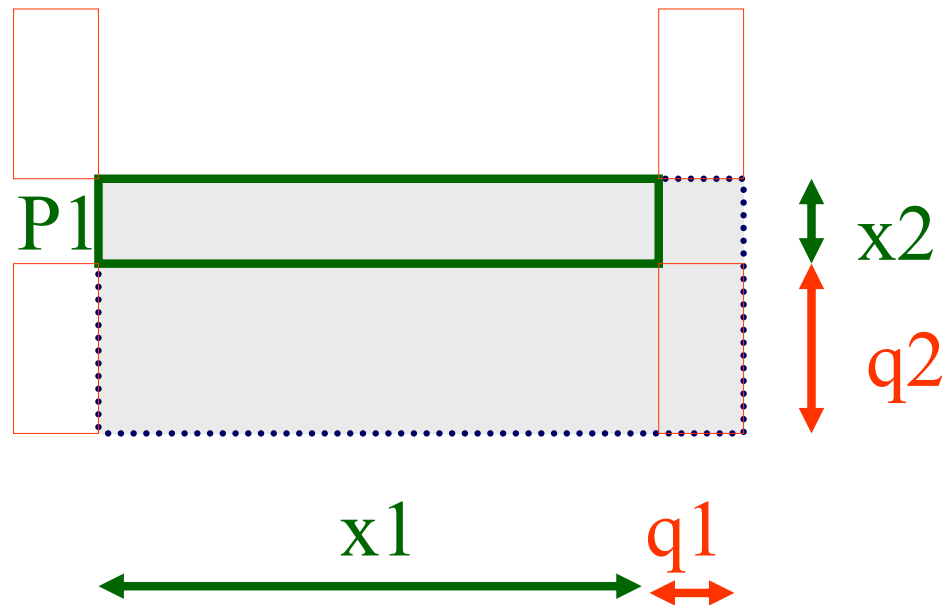
R-trees - performance analysis

- How many times will P1 be retrieved (unif. queries of size $q_1 \times q_2$)?



R-trees - performance analysis

- How many times will P1 be retrieved (unif. queries of size $q_1 \times q_2$)? A: $(x_1 + q_1) * (x_2 + q_2)$



R-trees - performance analysis

- Thus, given a tree with N nodes ($i=1, \dots, N$) we expect

$$\begin{aligned} \text{\#DiskAccesses}(q1, q2) &= \\ &\quad \text{sum} (x_{i,1} + q1) * (x_{i,2} + q2) \\ &= \text{sum} (x_{i,1} * x_{i,2}) + \\ &\quad q2 * \text{sum} (x_{i,1}) + \\ &\quad q1 * \text{sum} (x_{i,2}) \\ &\quad q1 * q2 * N \end{aligned}$$



R-trees - performance analysis

- Thus, given a tree with N nodes ($i=1, \dots, N$) we expect

$$\text{\#DiskAccesses}(q1, q2) =$$

$$\text{sum} (x_{i,1} + q1) * (x_{i,2} + q2)$$

$$= \text{sum} (x_{i,1} * x_{i,2}) +$$

$$q2 * \text{sum} (x_{i,1}) +$$

$$q1 * \text{sum} (x_{i,2})$$

$$q1 * q2 * N$$



‘volume’

surface area

count

R-trees - performance analysis

Observations:

- for point queries: only volume matters
- for horizontal-line queries: ($q_2=0$): vertical length matters
- for large queries ($q_1, q_2 \gg 0$): the count N matters

R-trees - performance analysis

Observations (cont' ed)

- overlap: does not seem to matter
- formula: easily extendible to n dimensions
- (for even more details: [Pagel +, PODS93], [Kamel+, CIKM93])



Berndt-Uwe Pagel

R-trees - performance analysis

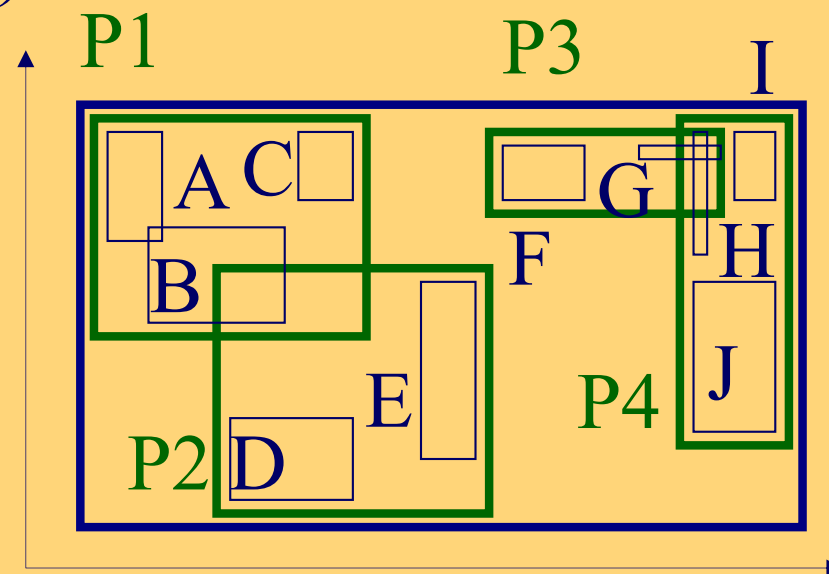
Conclusions:

- 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 $=0.5$ or so).



Solution#2: R-trees

- multi-dim trees
- Allow nodes to overlap
- Guaranteed 50% utilization – fast search (in low dim's)



R-trees – conclusions:

- Used in practice:
 - Oracle spatial ([R-tree](#))
 - Postgres: `create index ... using gist`
 - Databricks ([R-trees and z-order](#))
 - Sqlite3: www.sqlite.org/rtree.html
 - Python: `(pip install rtree)`

References

- Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger: *The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles*. ACM SIGMOD 1990: 322-331
- ➔ • Guttman, A. (June 1984). *R-Trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD, Boston, Mass.

References

- Jagadish, H. V. (May 23-25, 1990). Linear Clustering of Objects with Multiple Attributes. ACM SIGMOD Conf., Atlantic City, NJ.
- Ibrahim Kamel, Christos Faloutsos: *On Packing R-trees*, CIKM, 1993

References, cont' d

- Pagel, B., H. Six, et al. (May 1993). *Towards an Analysis of Range Query Performance*. Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Washington, D.C.
- Roussopoulos, N., S. Kelley, et al. (May 1995). Nearest Neighbor Queries. Proc. of ACM-SIGMOD, San Jose, CA.