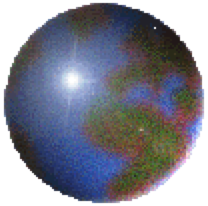
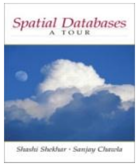
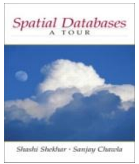


Chap4: Spatial Storage and Indexing



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!

Concepts in a physical data model

- Database concepts
  - Conceptual data model - entity, (multi-valued) attributes, relationship, …
  - Logical model - relations, atomic attributes, primary and foreign keys
  - Physical model - secondary storage hardware, file structures, indices, …
- Examples of physical model concepts from relational DBMS
  - Secondary storage hardware: Disk drives
  - File structures - sorted
  - Auxiliary search structure -
    - search trees (hierarchical collections of one-dimensional ranges)

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

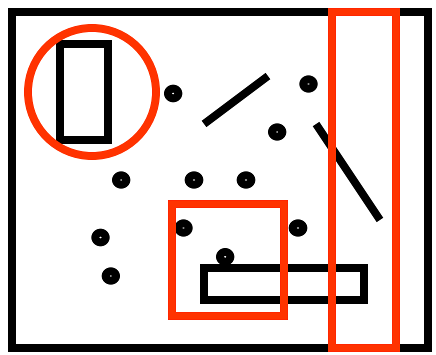
- Physical model provides simpler operations needed by spatial queries!
- *Common Queries*
  - *Range query*
  - *Nearest neighbor*
  - *Spatial-join query*
  - *Others (Closest-pair query, Color range query, etc.)*

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

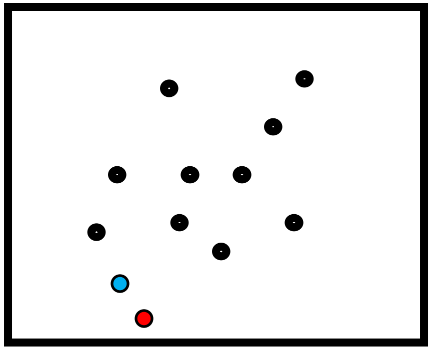


- Ex. Find all warehouses at dist < 50 Km from location (0,0)

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

Nearest neighbor query

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

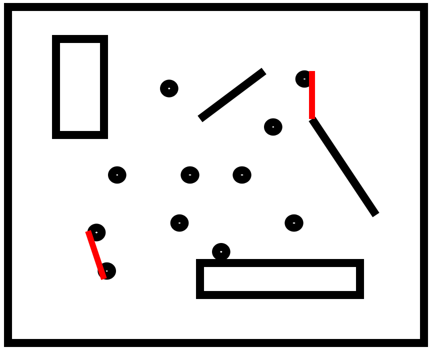


- Ex. Find the store closest to store 101

```
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

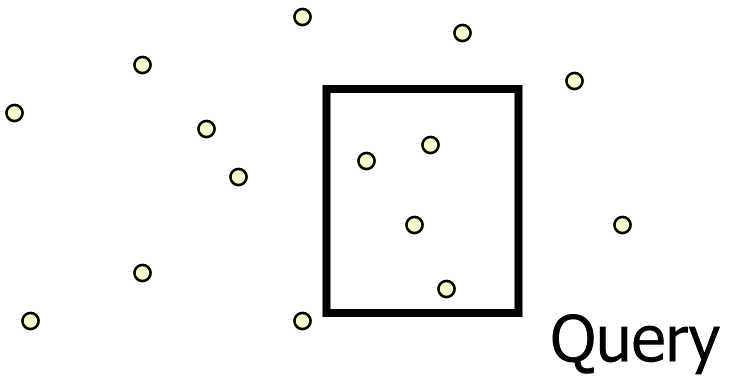
- What is a file structure?
  - A method of organizing records in a file
  - For efficient implementation of common file operations on disks
  - Example: ordered files
- Measure of efficiency
  - I/O cost: Number of disk sectors retrieved from secondary storage
  - CPU cost: Number of CPU instruction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

Point Access Methods (PAM)

- PAM: index only point data
  - Multidimensional Hashing
  - Hierarchical (tree-based) structures
  - Space filling curve

The problem

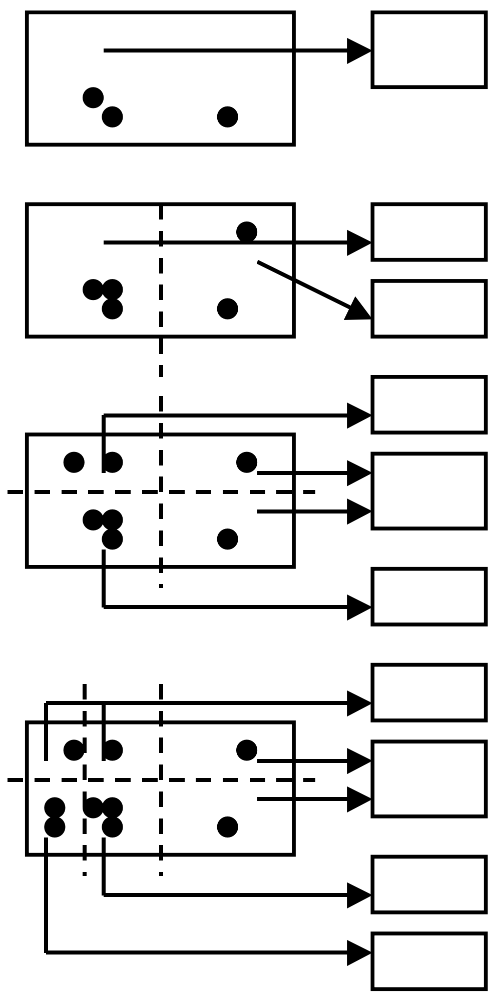
- Given a point set and a rectangular query, find the points enclosed in the query



Grid File

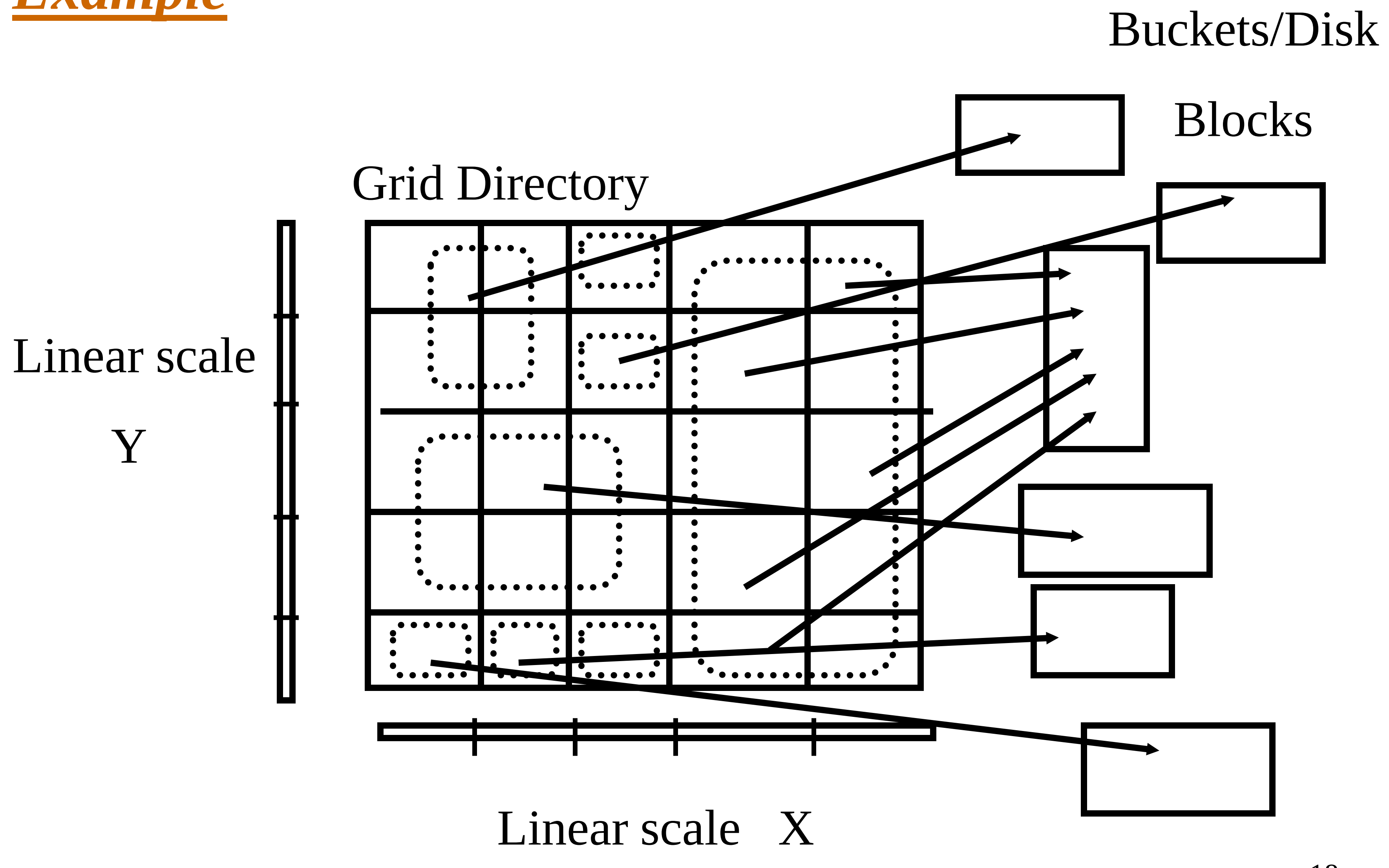
- Hashing methods for multidimensional points (extension of Extensible hashing)
- Idea: Use a grid to partition the space→ each cell is associated with one page
- Two disk access principle (exact match)

Grid File



- Select dividers along each dimension. Partition space into cells
- Dividers cut all the way.
- Each cell corresponds to 1 disk page.
- Many cells can point to the same page.
- Cell directory potentially exponential in the number of dimensions

Example


Buckets/Disk
Blocks
Grid Directory
Linear scale
Y
Linear scale X

Grid File Search

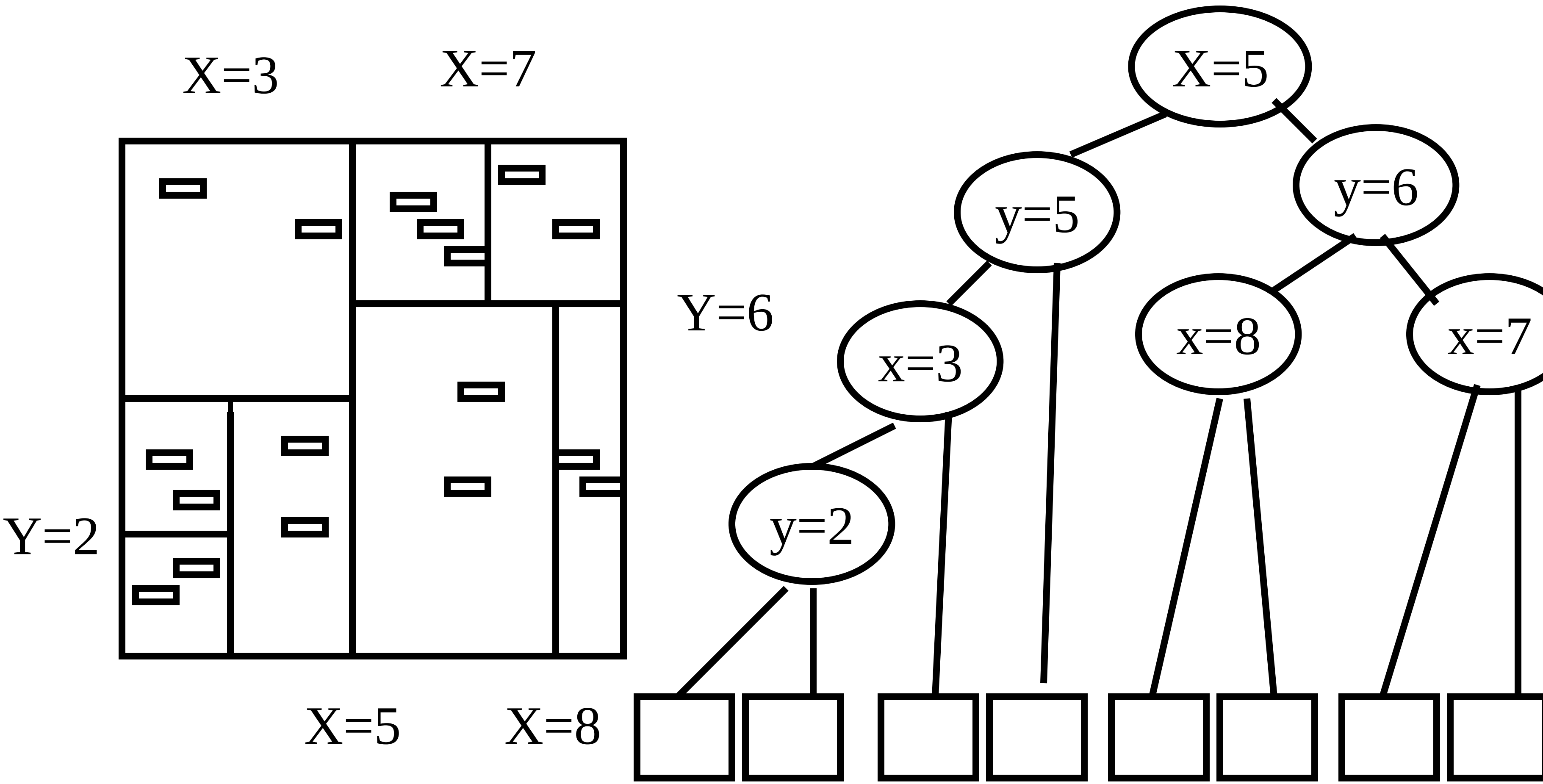
- Exact Match Search: at most 2 I/Os assuming linear scales fit in memory.
  - First use liner scales to determine the index into the cell directory
  - access the cell directory to retrieve the bucket address (may cause 1 I/O if cell directory does not fit in memory)
  - access the appropriate bucket (1 I/O)
- Range Queries:
  - use linear scales to determine the index into the cell directory.
  - Access the cell directory to retrieve the bucket addresses of buckets to visit.
  - Access the buckets.

Tree-based PAMs

- Most of tb-PAMs are based on kd-tree
- kd-tree is a main memory binary tree for indexing k-dimensional points
  - Needs to be adapted for disk model
- Levels rotate among the dimensions, partitioning the space based on a value for that dimension
- kd-tree is not necessarily balanced

Example

At each level we use a different dimension



Kd-tree properties

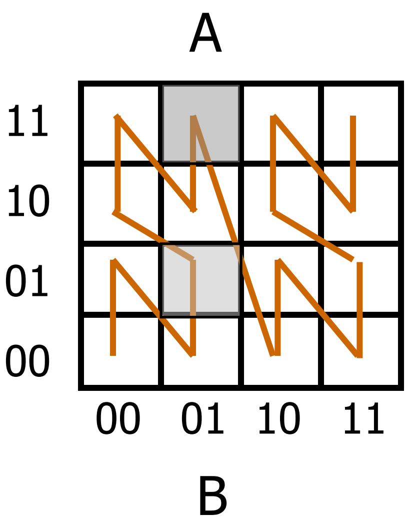
- Height of the tree O(log n)
- Search time for exact match: O(log n)
- Search time for range query: $O(n^{1/2} + k)$

Space Filling Curves: Z-ordering

- Map points from 2-dimensions to 1-dimension.
- Use a B+-tree to index the 1-dimensional points
- Basic assumption: Finite precision in the representation of each co-ordinate, K bits (2^K values)
  - The address space is a square (image) and represented as a 2^K x 2^K array
  - Each element is called a pixel

Z-ordering

- Impose a linear ordering on the pixels of the image → 1 dimensional problem

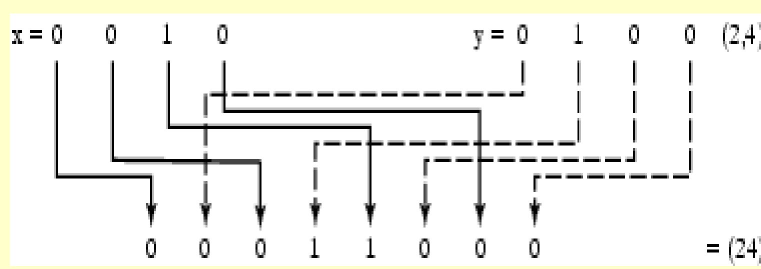


$Z_A = \text{shuffle}(x_A, y_A) = \text{shuffle}(\text{“01”}, \text{“11”})$

$= 0111 = (7)_{10}$

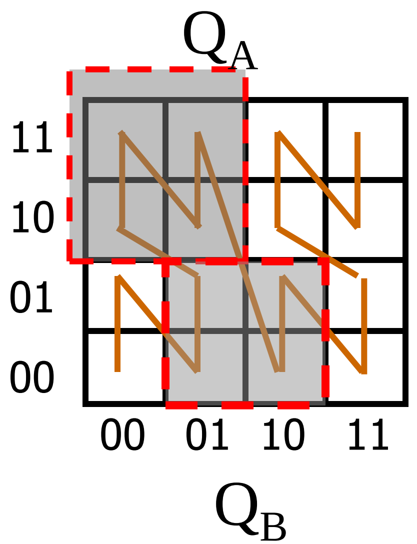
$Z_B = \text{shuffle}(\text{“01”}, \text{“01”}) = 0011$

Example of shuffling



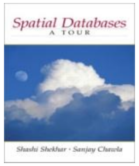
Queries

- Find the z-values that contained in the query and then the ranges



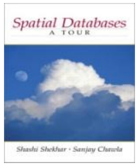
$Q_A \rightarrow$ range [4, 7]

$Q_B \rightarrow$ ranges [2,3] and [8,9]



Learning Objectives

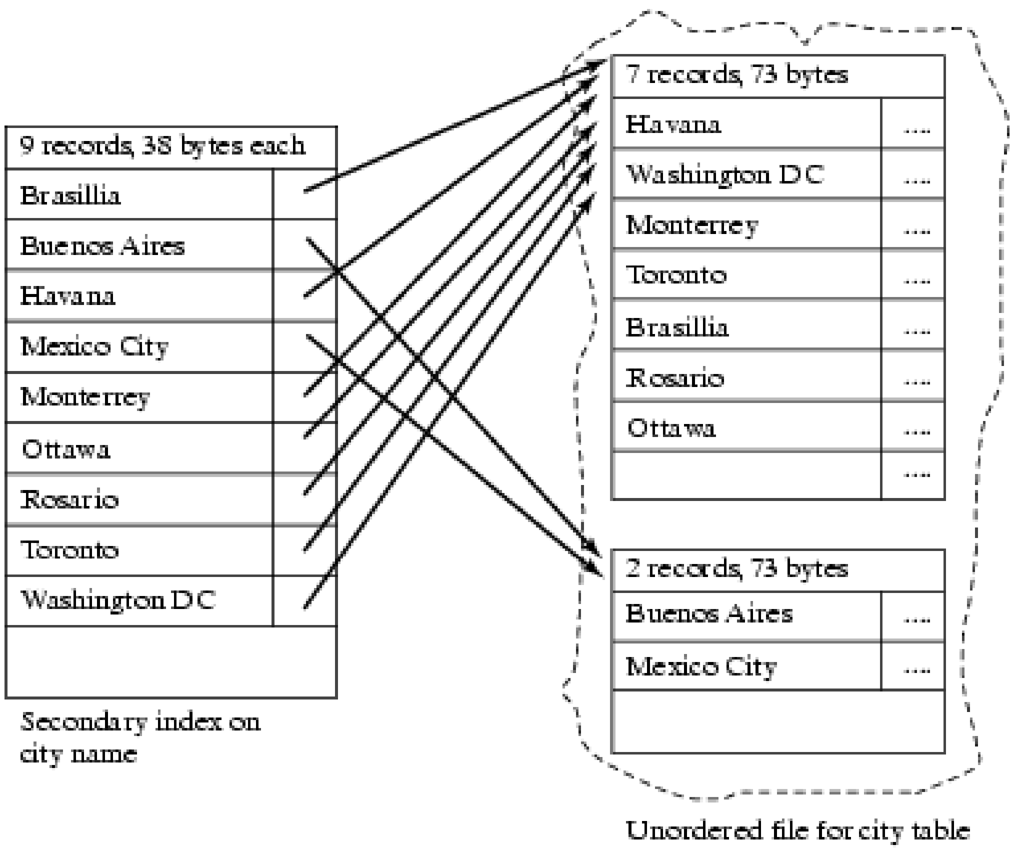
- Learning Objectives (LO)
  - LO1: Understand concept of a physical data model
  - LO2: Learn how to structure data files
  - LO3: Learn how to use auxiliary data-structures
    - Concept of index
    - Spatial indices, e.g. R-tree families
    - Focus on concepts not procedures!
- Mapping Sections to learning objectives
  - LO2 - 4.1
  - LO3 - 4.2



What is an index?

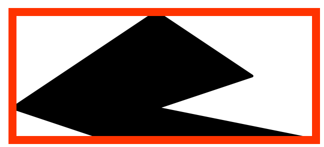
- Concept of an index
  - auxiliary file to search a data file
  - Example: Fig. 4.10
- index records have
  - key value
  - address of relevant data sector
    - see arrows in Fig. 4.10
- Index records are ordered
  - find, findnext, insert are fast
- Note assumption of total order
  - on values of indexed attributes

Fig 4.10



Spatial Access Methods (SAMs)

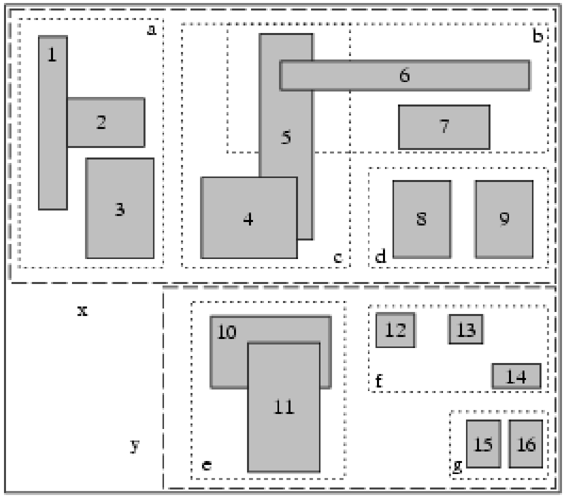
- Indexes for spatial data that have extend (not only point data)
- Use only Minimum Bounding Rectangles – **MBR**s (filtering)

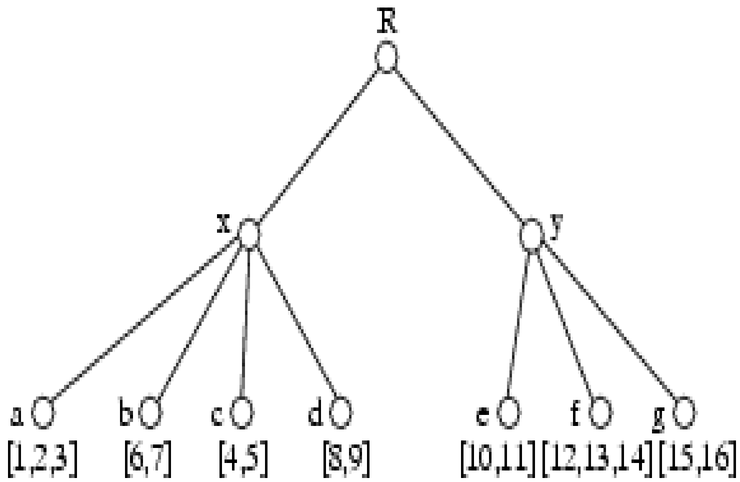


- R-tree (Guttman, 1984) is the prominent SAM
  - Implemented in Oracle, Postgres, Informix

R-Tree

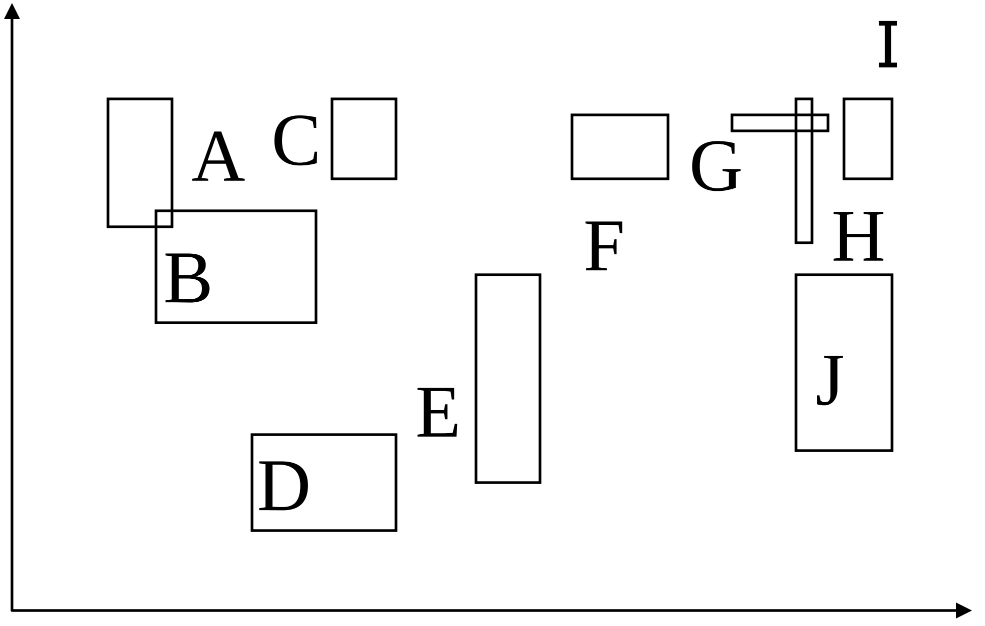
- A multi-way external memory tree
- Index nodes and data (leaf) nodes
- All leaf nodes appear on the same level
- Every node contains between m and M entries
- The root node has at least 2 entries (children)





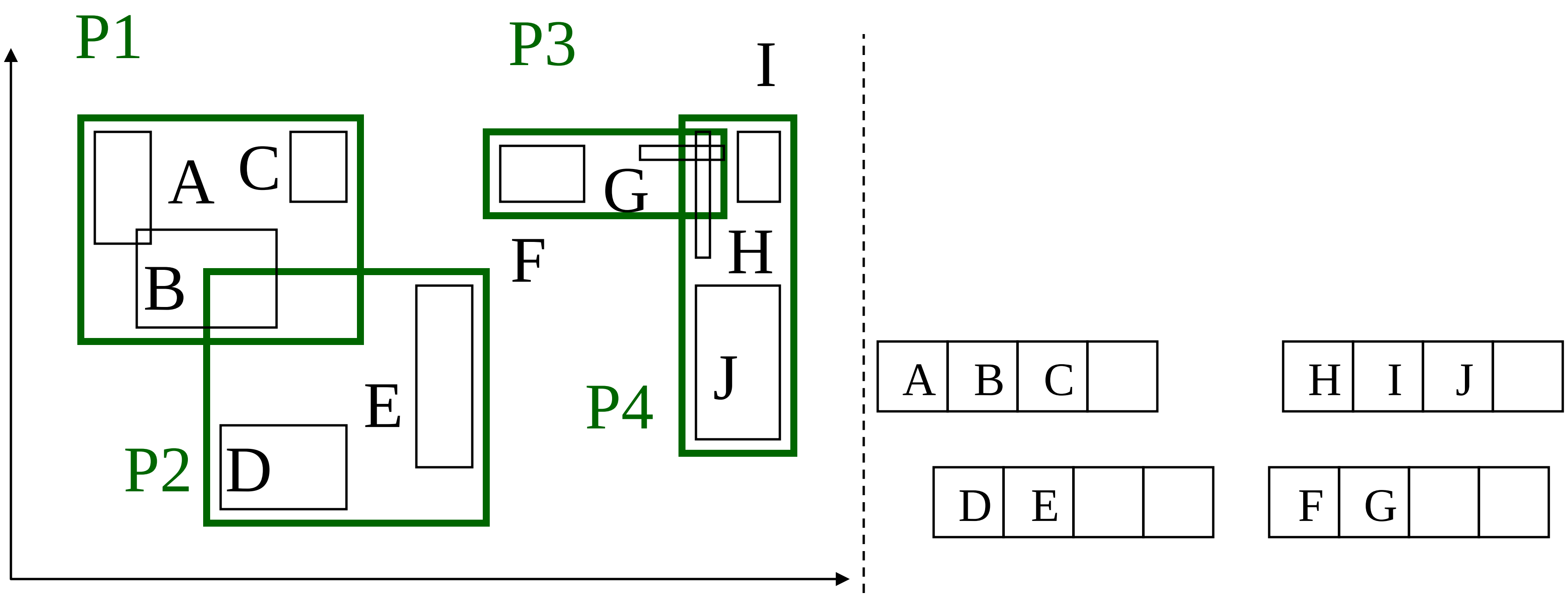
Example

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



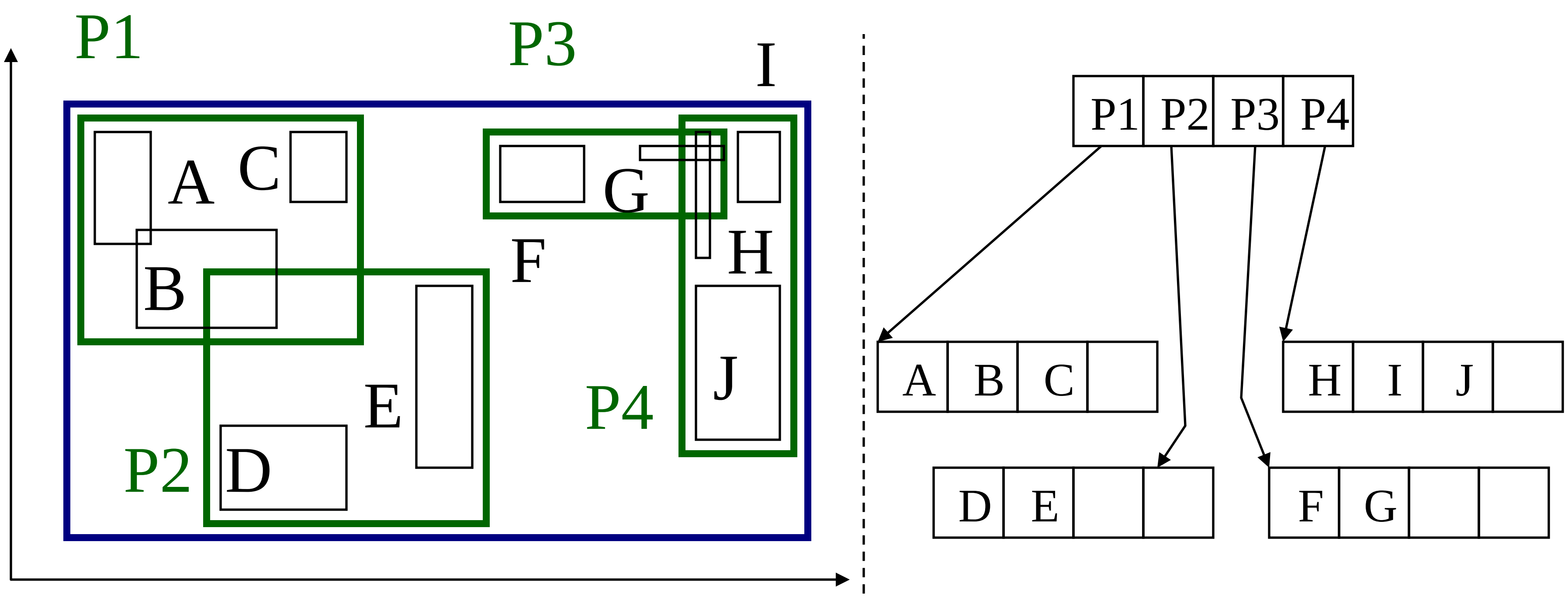
Example

- F=4

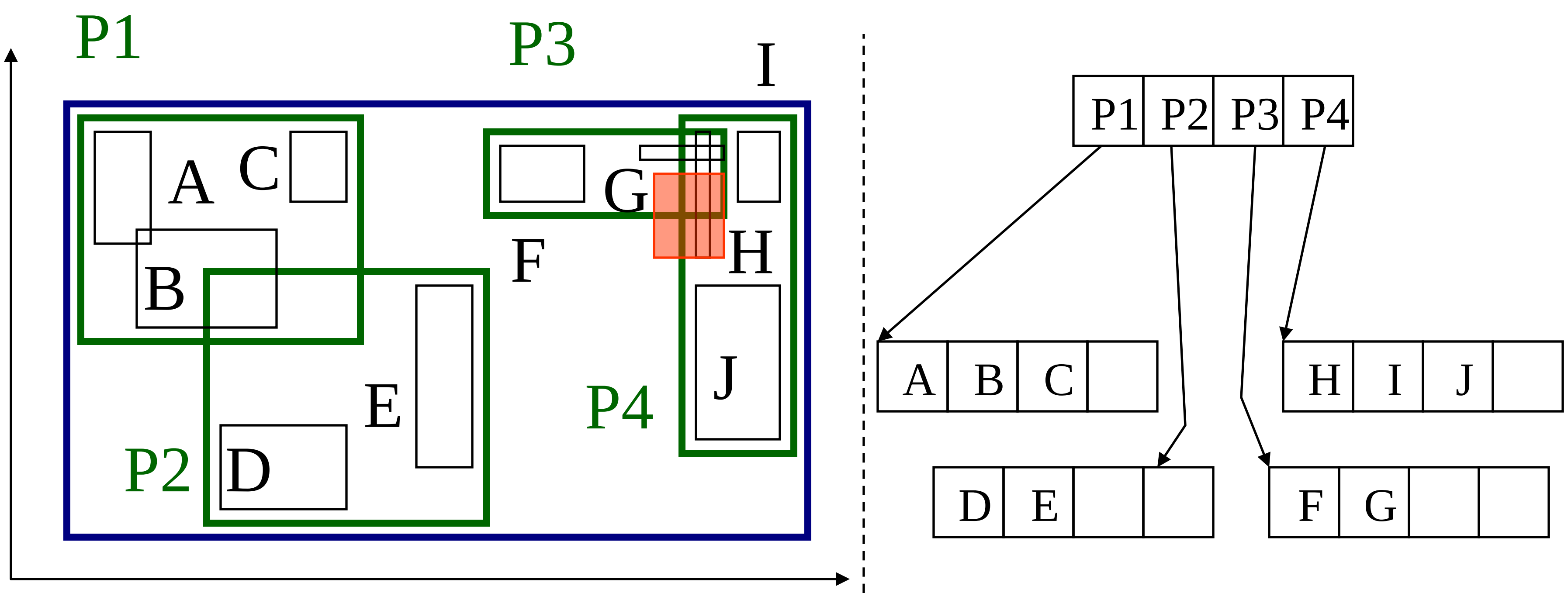


Example

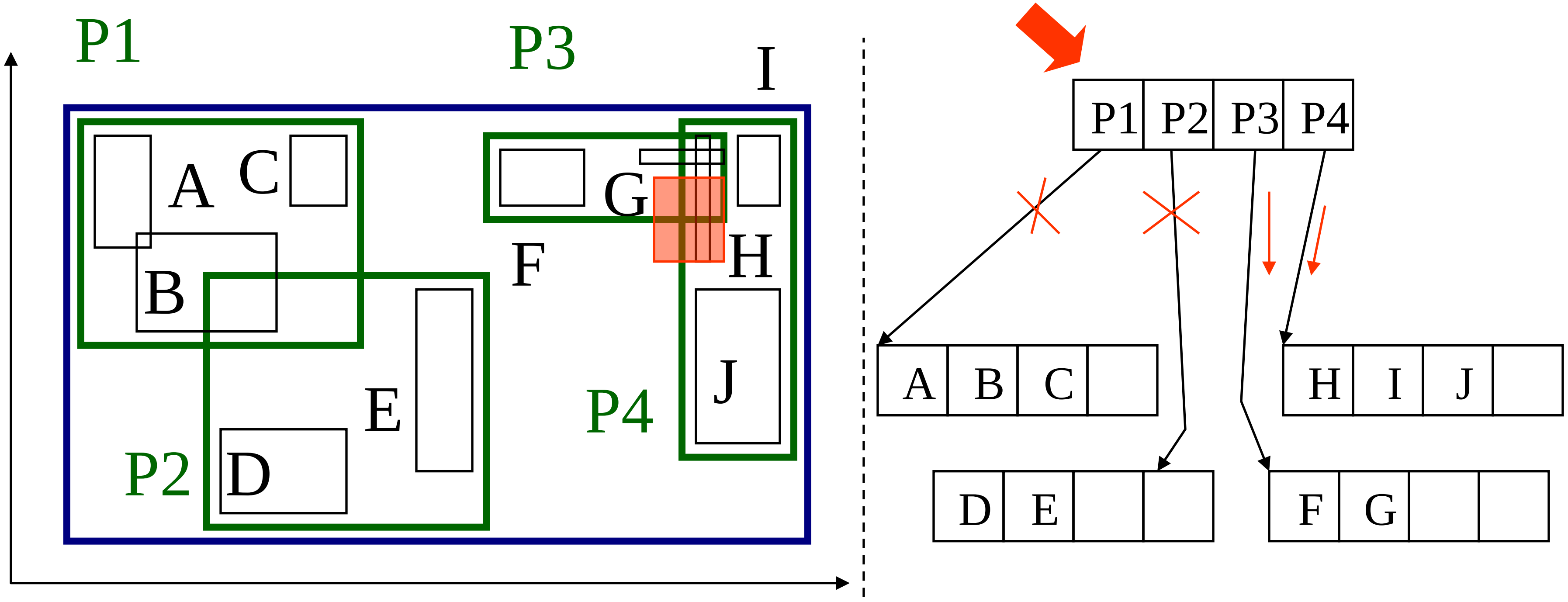
- F=4



R-trees:Search



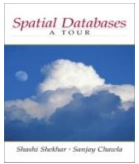
R-trees:Search



A query may follow multiple branches

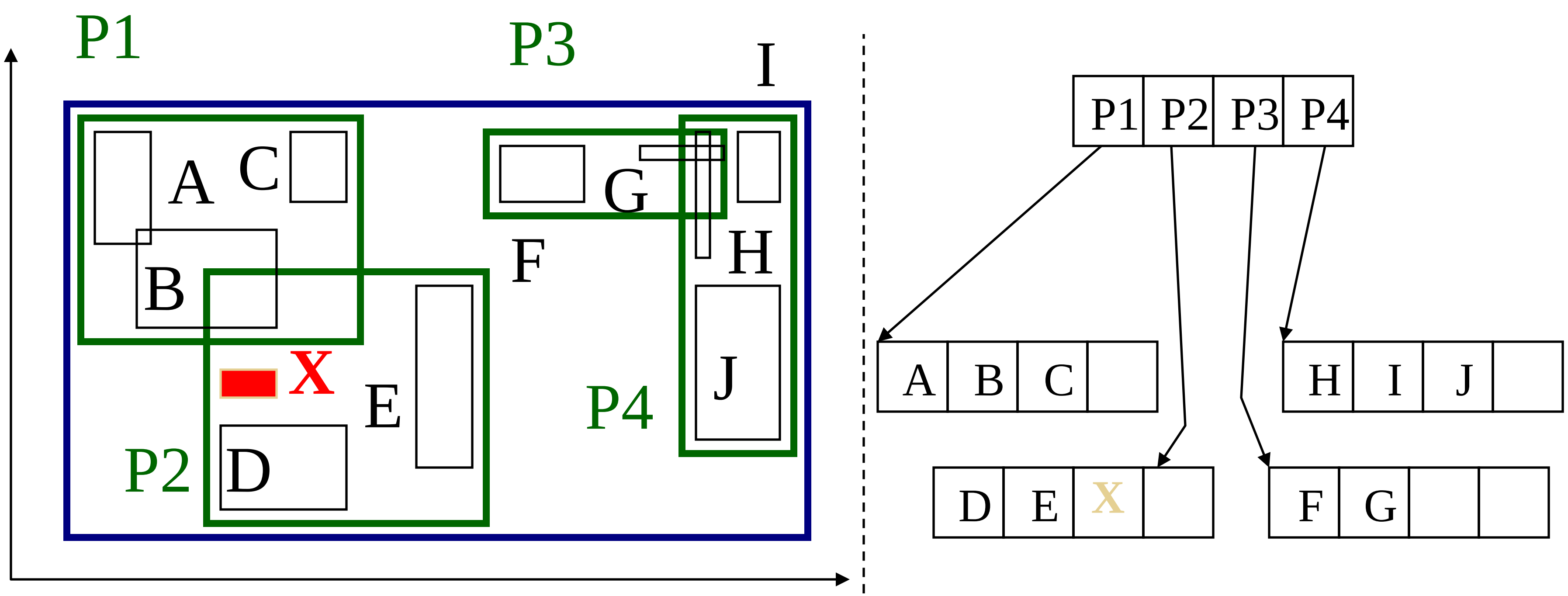
R-trees:Insertion

- 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)



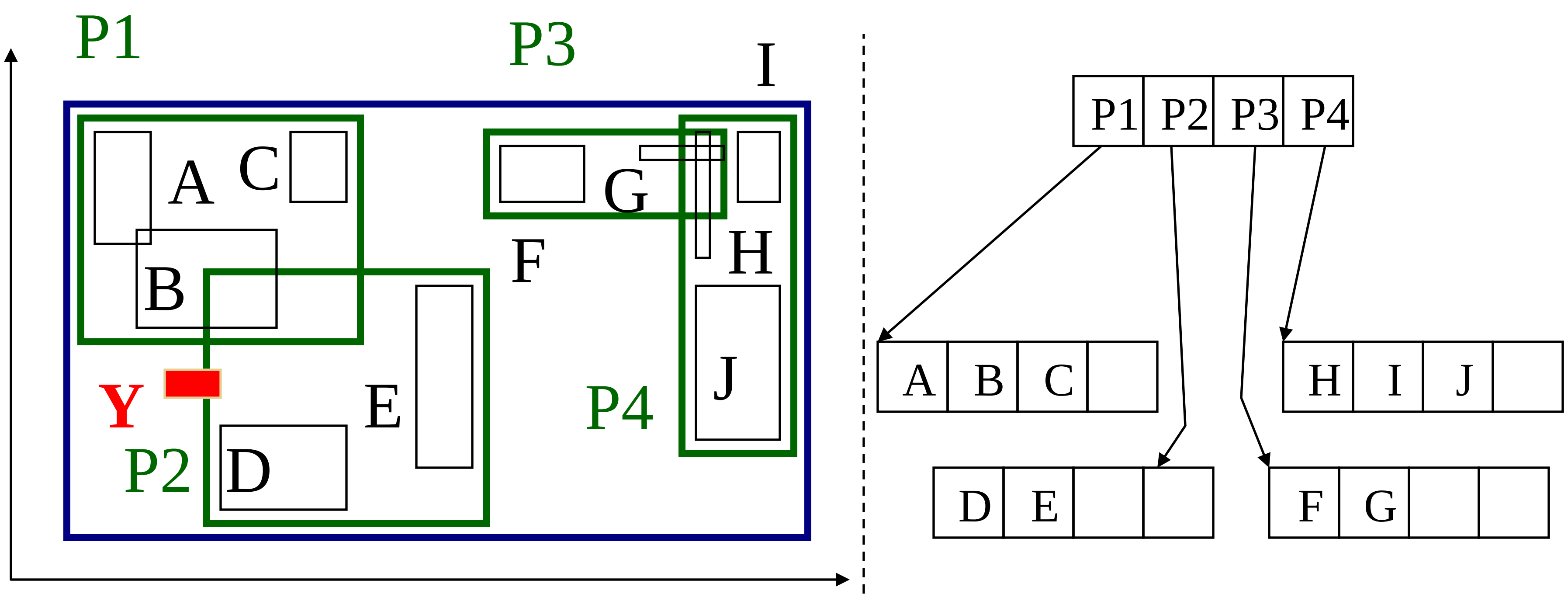
R-trees:Insertion

Insert X



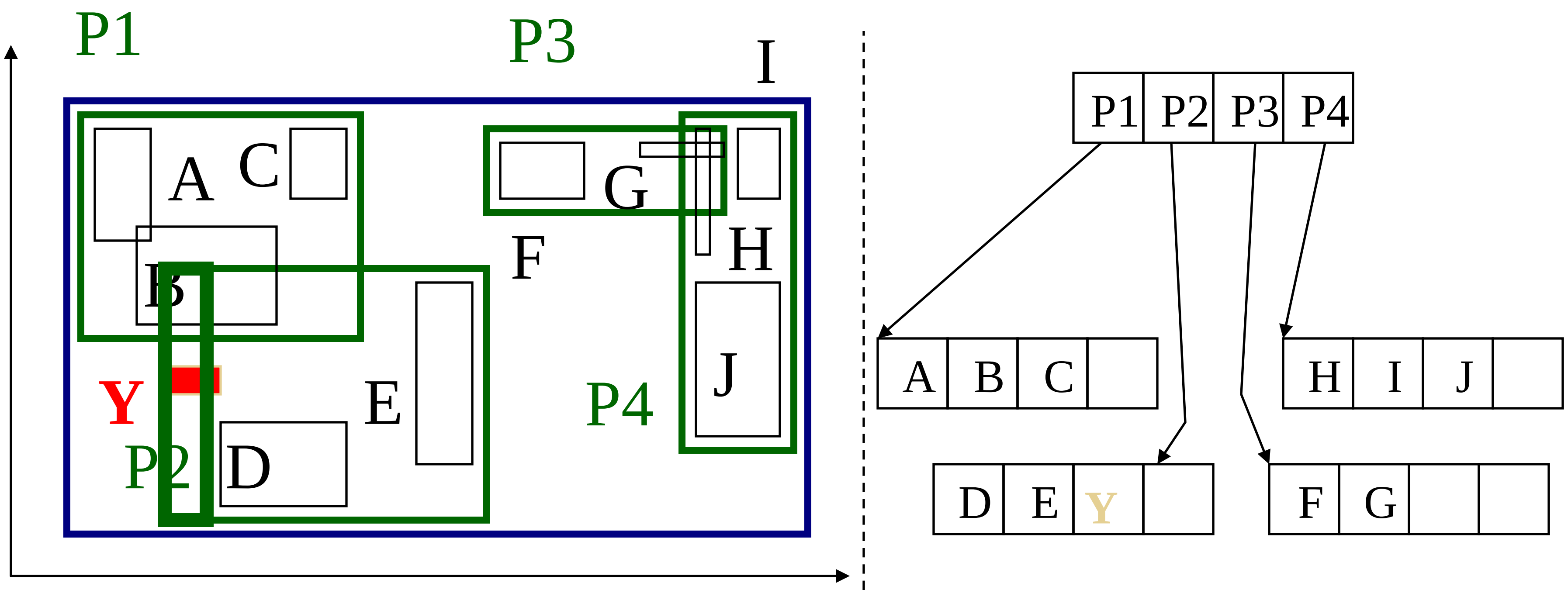
R-trees:Insertion

Insert Y



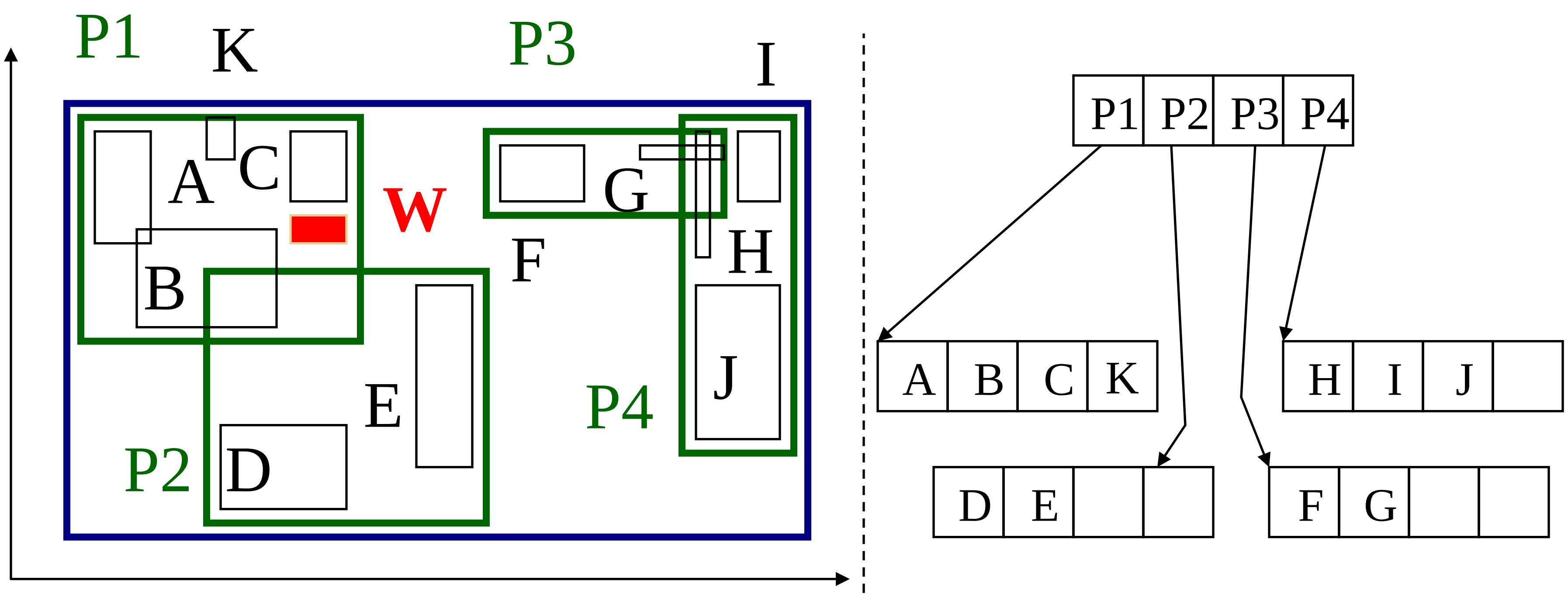
R-trees:Insertion

- Extend the parent MBR



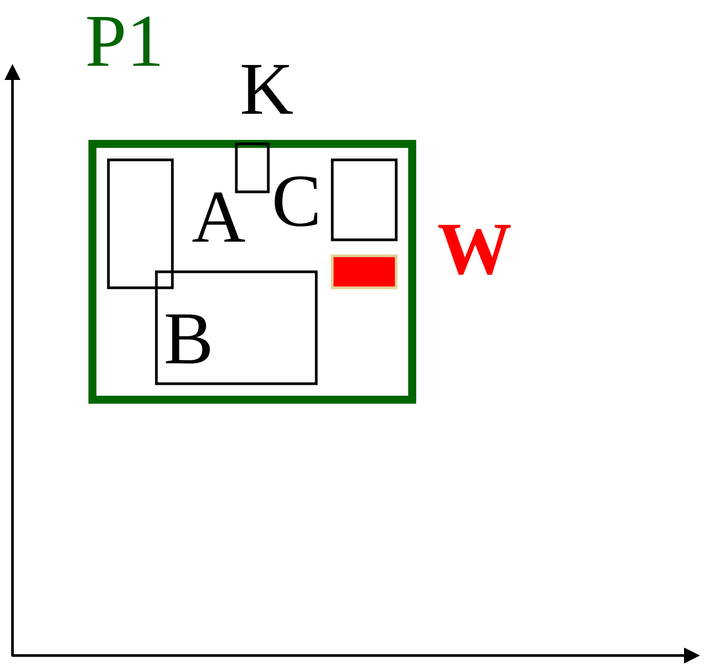
R-trees:Insertion

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



R-trees:Split

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

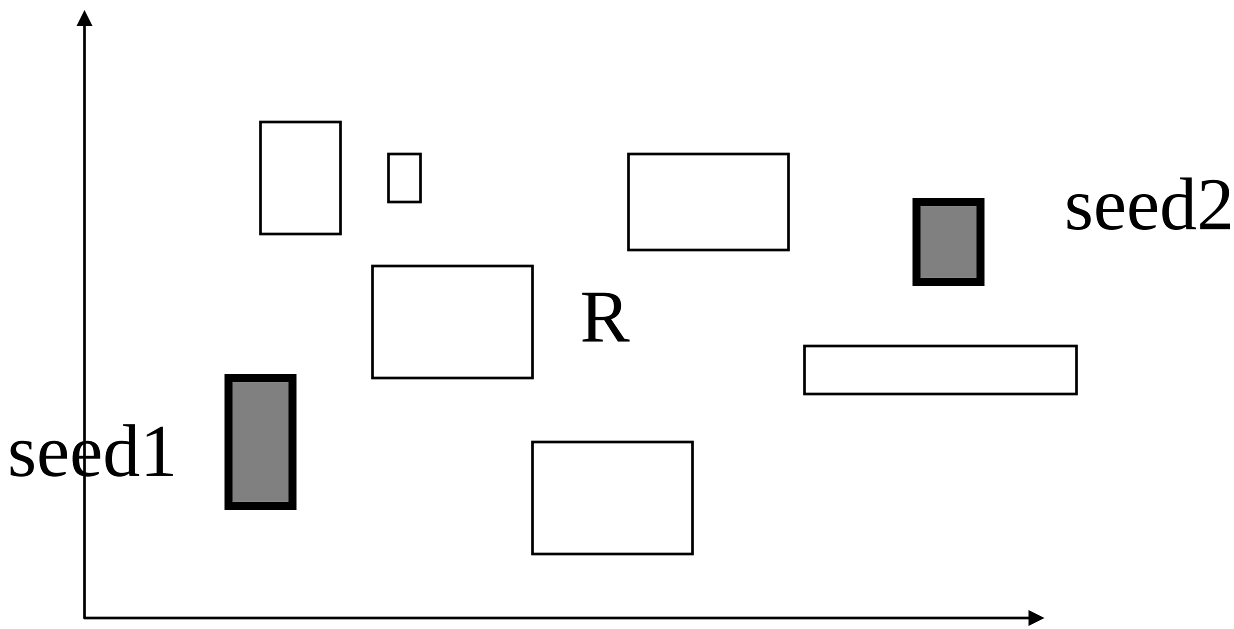


- A1: 'linear' split
- A2: quadratic split
- A3: exponential split:

  2^{M-1} choices

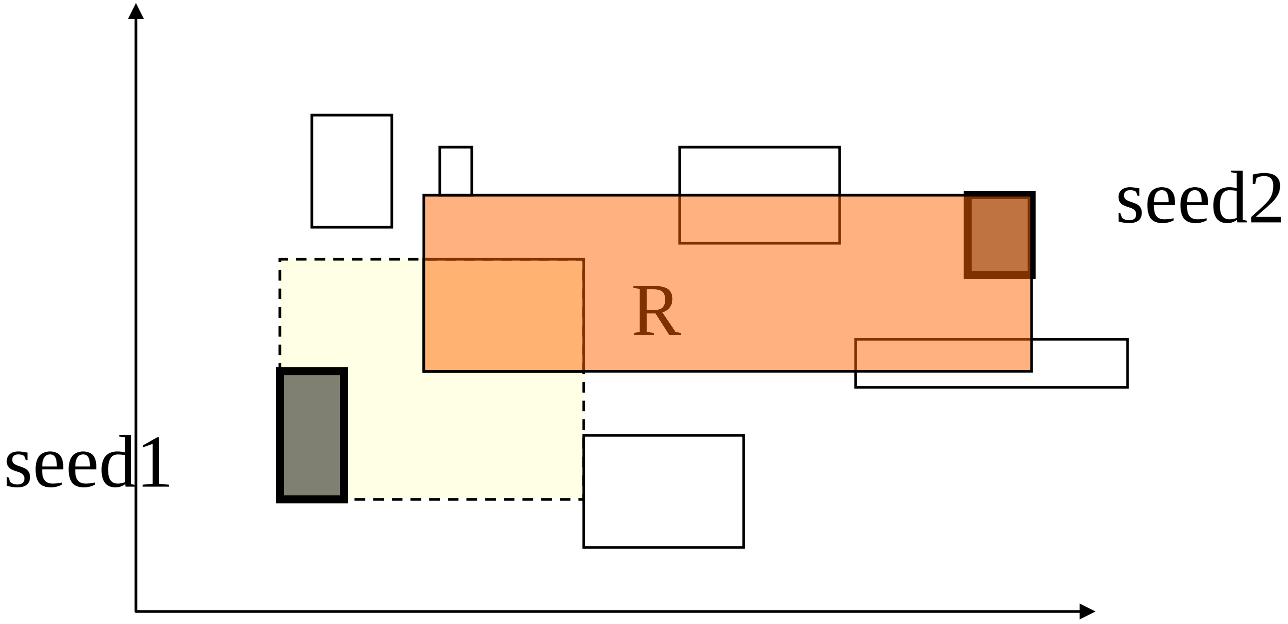
R-trees:Split

- 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

Summary

- Physical DM efficiently implements logical DM on computer hardware
  - Physical DM has file-structure, indexes
- Classical methods were designed for data with total ordering
  - fall short in handling spatial data
  - because spatial data is multi-dimensional
- Two approaches to support spatial data and queries
  - Reuse classical method
    - Use Space-Filling curves to impose a total order on multi-dimensional data
  - Use new methods
    - R-trees, Grid files