hB-tree and LSD-tree

Indexing hierarchies (hB-tree)

- A multi-attribute index structure corresponds to the hB-tree.
- It is derived from the K-D-B-tree, but has additional desirable properties.
- The hB-tree (holey brick tree – akytos plytos medis) is similar to K-D-B-tree, except that splitting of the node is done based on ***multiple attributes***

Indexing hierarchies (hB-tree)

- The external ***enclosing regions*** and several cavities called ***extracted regions.***
- The hB-tree internal node search and growth processes are precisely analogous to the corresponding processes in B-trees
- The intra-node processes are unique

hB-trees - basics

- A new multi-attribute index structure called the hB-tree is introduced.
- It is derived from the K-D-B-tree but has additional desirable properties.
- The hB-tree internode search and growth processes are precisely analogous to the corresponding processes in B-trees.
- The intra-node processes are unique.

hB-trees - basics

- A k-d-tree is used as the structure within nodes for very efficient searching. Node splitting requires that this k-d-tree be split.
- This produces nodes which no longer represent brick-like regions in k-space, but that can be characterized as holey bricks, bricks in which sub-regions have been extracted.
- It guarantees hB-tree users decent storage utilization, reasonable size index terms, and good search and insert performance.

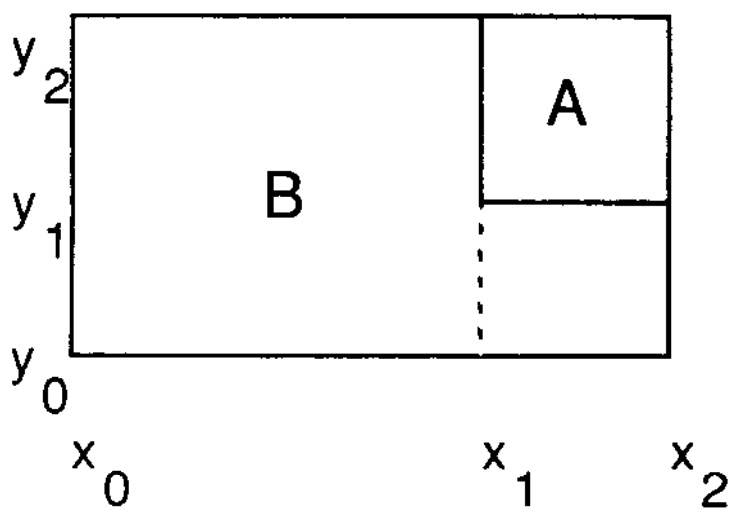
Indexing hierarchies (hB-tree)

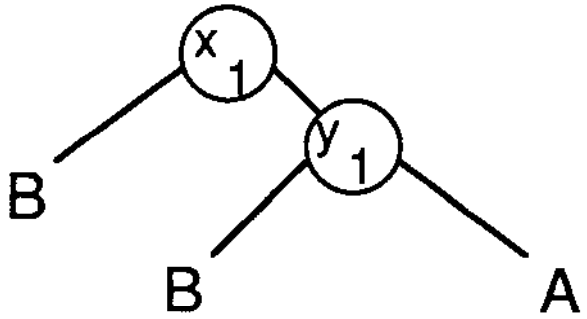
- We distinguish between
  - data nodes, that are pages which contain the records of the database and
  - index nodes that contain k-d-trees.
- The data nodes are the leaves of the hB-tree.
- The index nodes are the internal nodes of the hB-tree.
- The hB-tree grows from the leaves and has all leaves at the same level, just as a B-tree does.

Indexing hierarchies (hB-tree)

- Specific node structure:
  - the k-d tree is for internal structure of the index node - within hB-tree index nodes is to organize information about lower levels of the hB-tree.
  - The k-d tree also assists in organizing data nodes of hB-trees.

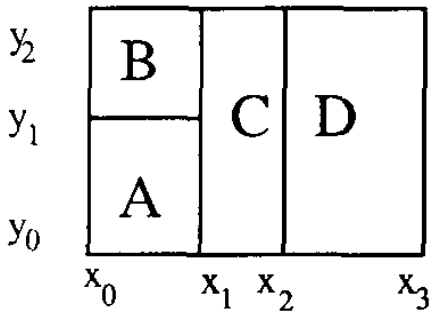
Indexing hierarchies (hB-tree)



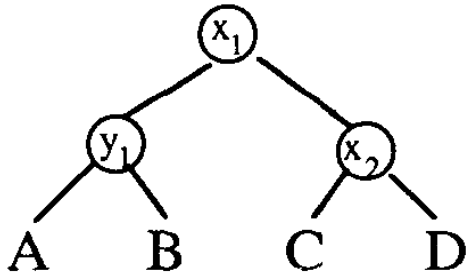


- A holey brick is represented via k-d-tree. A holey brick is a brick from which a smaller brick has been removed.
- Two leaves of the k-d tree are required to reference the holey brick region denoted by B.

Indexing hierarchies (hB-tree)



5 (a)



5 (b) : k-d tree

A : x_0 x_1 y_0 y_1

B : x_0 x_1 y_1 y_2

C : x_1 x_2 y_0 y_2

D : x_2 x_3 y_0 y_2

5 (c) : Boundaries

- Bricks share boundaries. For a k-d tree, a boundary is typically checked only once.

Indexing hierarchies (hB-tree)

- Searching using the hB-tree
- Exact Match Queries:
  - one follows a unique path from the root of the hB-tree down the hB-tree to a data page.
  - The number of hB-tree nodes accessed is the height of the hB-tree.
- Range Queries:
  - A range query specifies a range of values for one or more of the attributes.

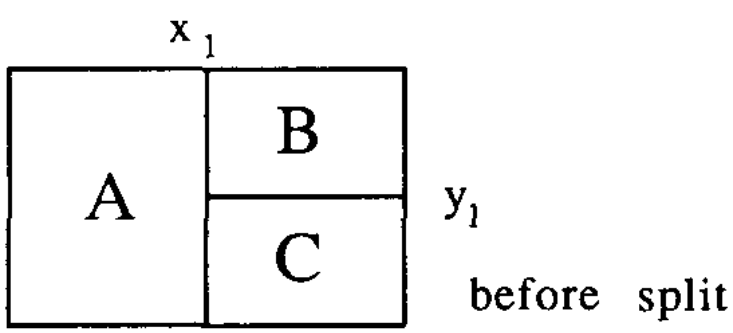
Indexing hierarchies (hB-tree)

- If a comparison value in a k-d-tree node is greater than all the same attribute's values in the search range, one goes to the left.
- If it is smaller, one goes to the right.
- If a comparison value is in the middle of a search range, one follows both the left and right branches.

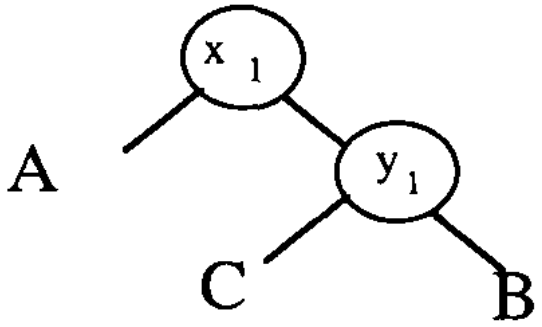
- Several hB-tree nodes at each level may be accessed.
- Region Data: Finding all the regions D contained in a given region R is a range query.

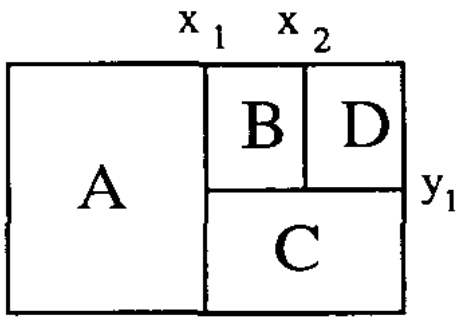
Indexing hierarchies (hB-tree)

- Data node splitting:

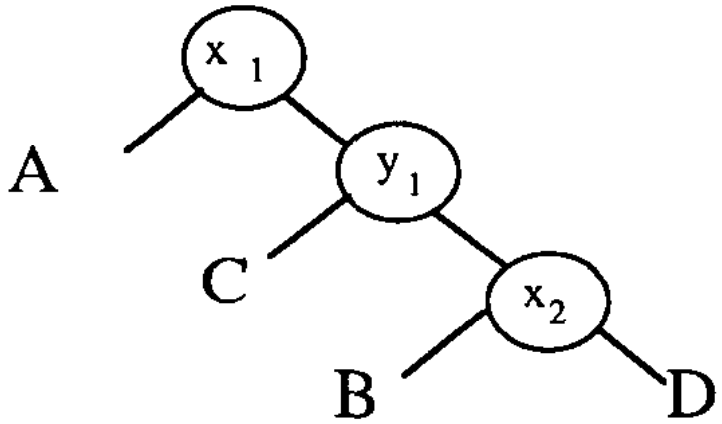


before split



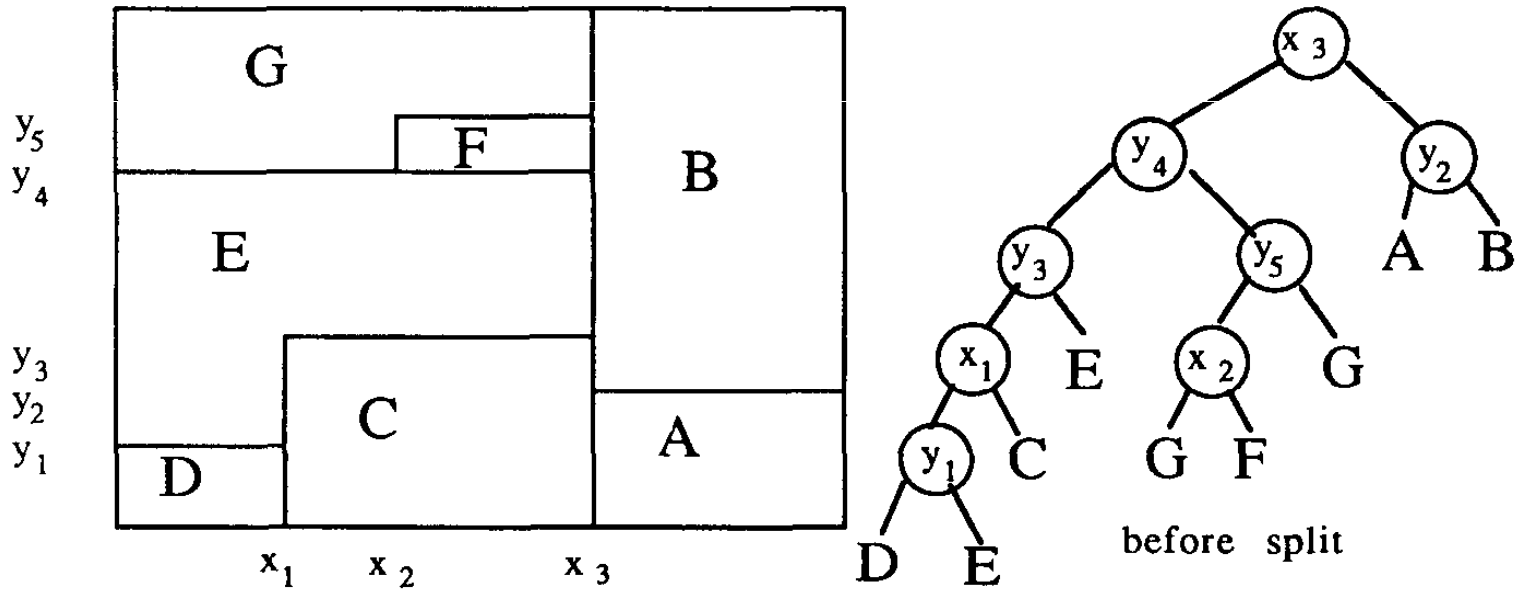


after split



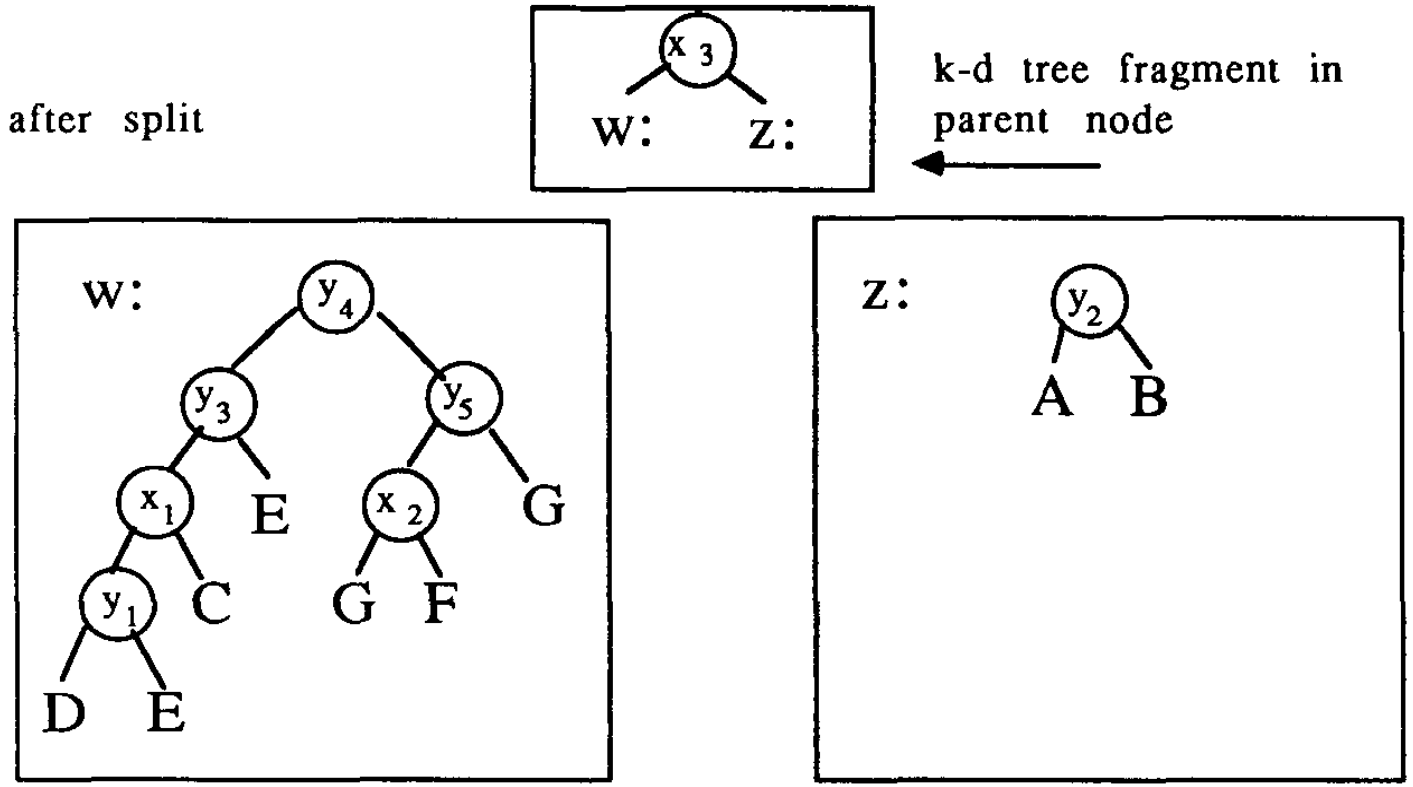
Indexing hierarchies (hB-tree)

Index node splitting (primary data):



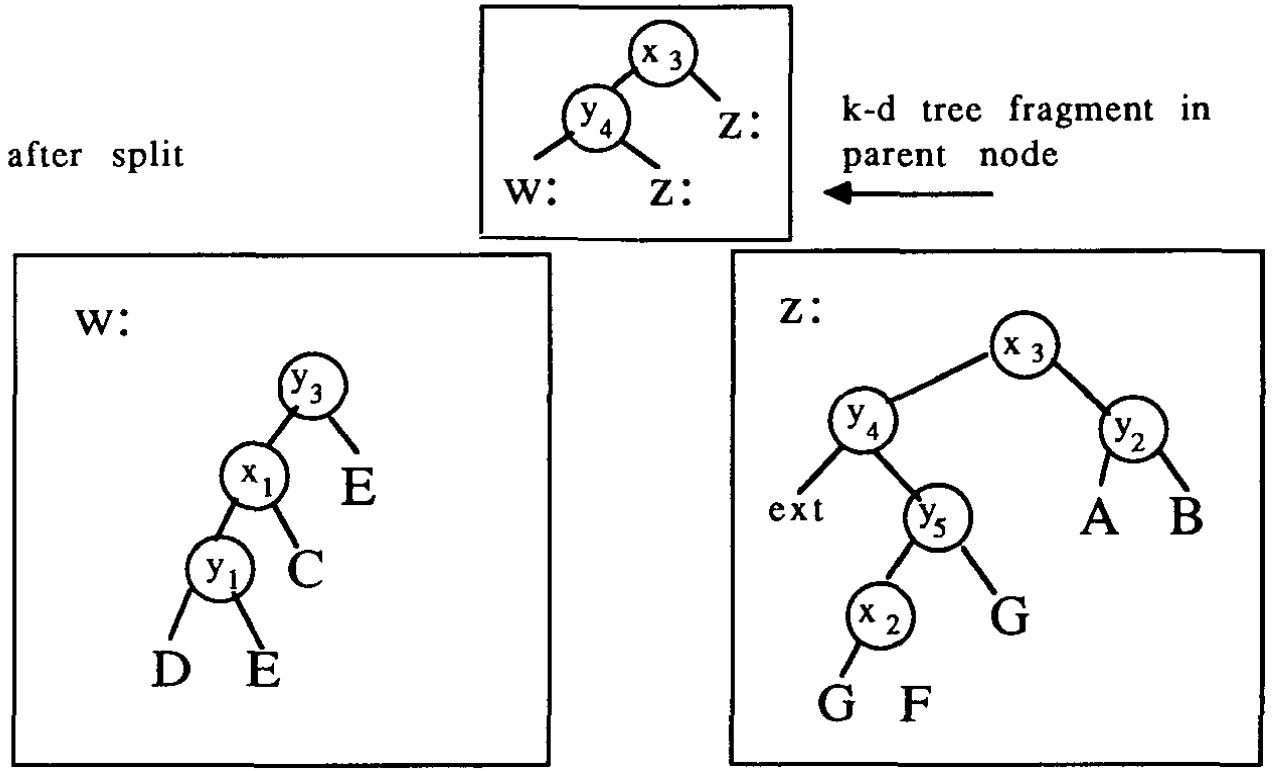
Indexing hierarchies (hB-tree)

Index node splitting (poor fan-out, low node utilization :



Indexing hierarchies (hB-tree)

Index node splitting (suitable subtree):

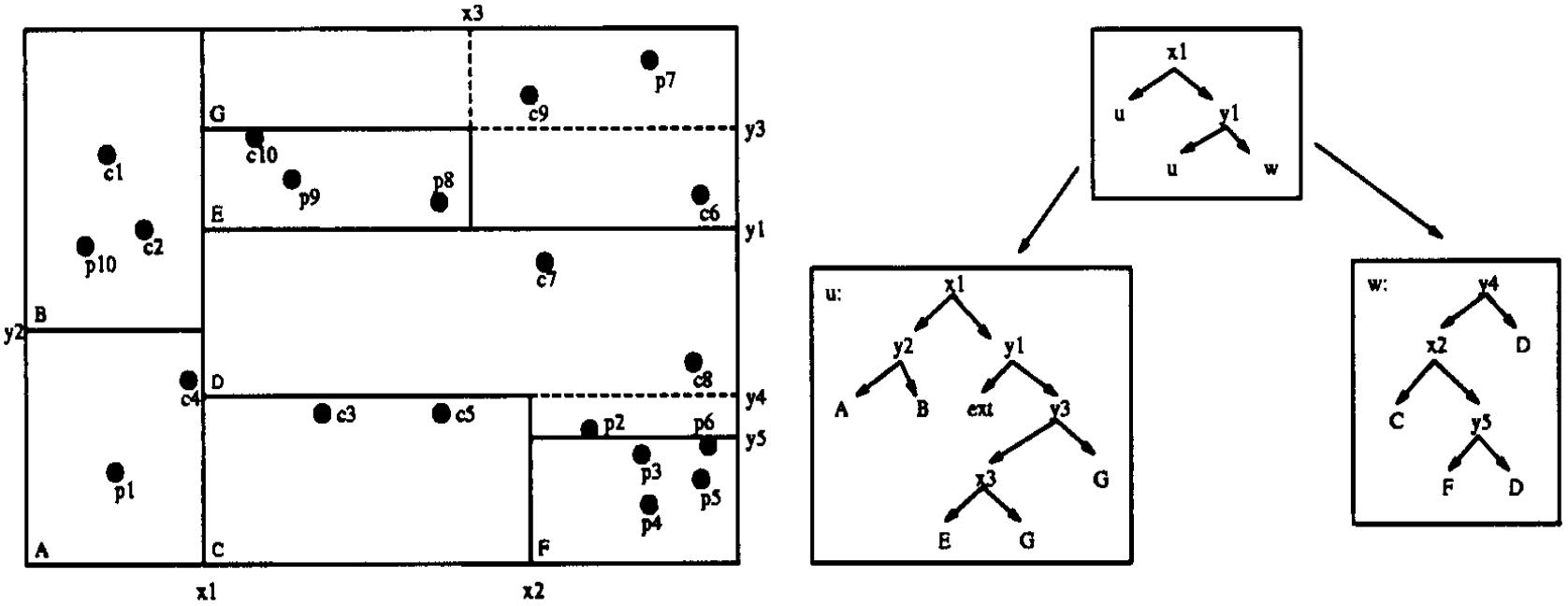


Indexing hierarchies (hB-tree)

- A k-d tree is used as the structure within nodes for very efficient searching.
- Node splitting requires that this k-d tree be split.
- This produces nodes which no longer represent brick-like regions in k-space, but that can be characterized as holey bricks, bricks in which subregions have been extracted.
- The intranode processes are unique.
- Results guarantee hB-tree users decent storage utilization, reasonable size index terms, and good search and insert performance.

Indexing by subspaces (hB-tree)

hB-tree



Indexing hierarchies (LSD-tree)

- **The lsd tree: spatial access to multidimensional point and non-point objects**
- Local Split Decision tree (LSD tree) - data structure supporting efficient spatial access to geometric objects.
- It performs well for all reasonable data distributions, cover quotients (which measure the overlapping of the data objects), and bucket capacities, and that it maintains multidimensional points as well as arbitrary geometric objects.

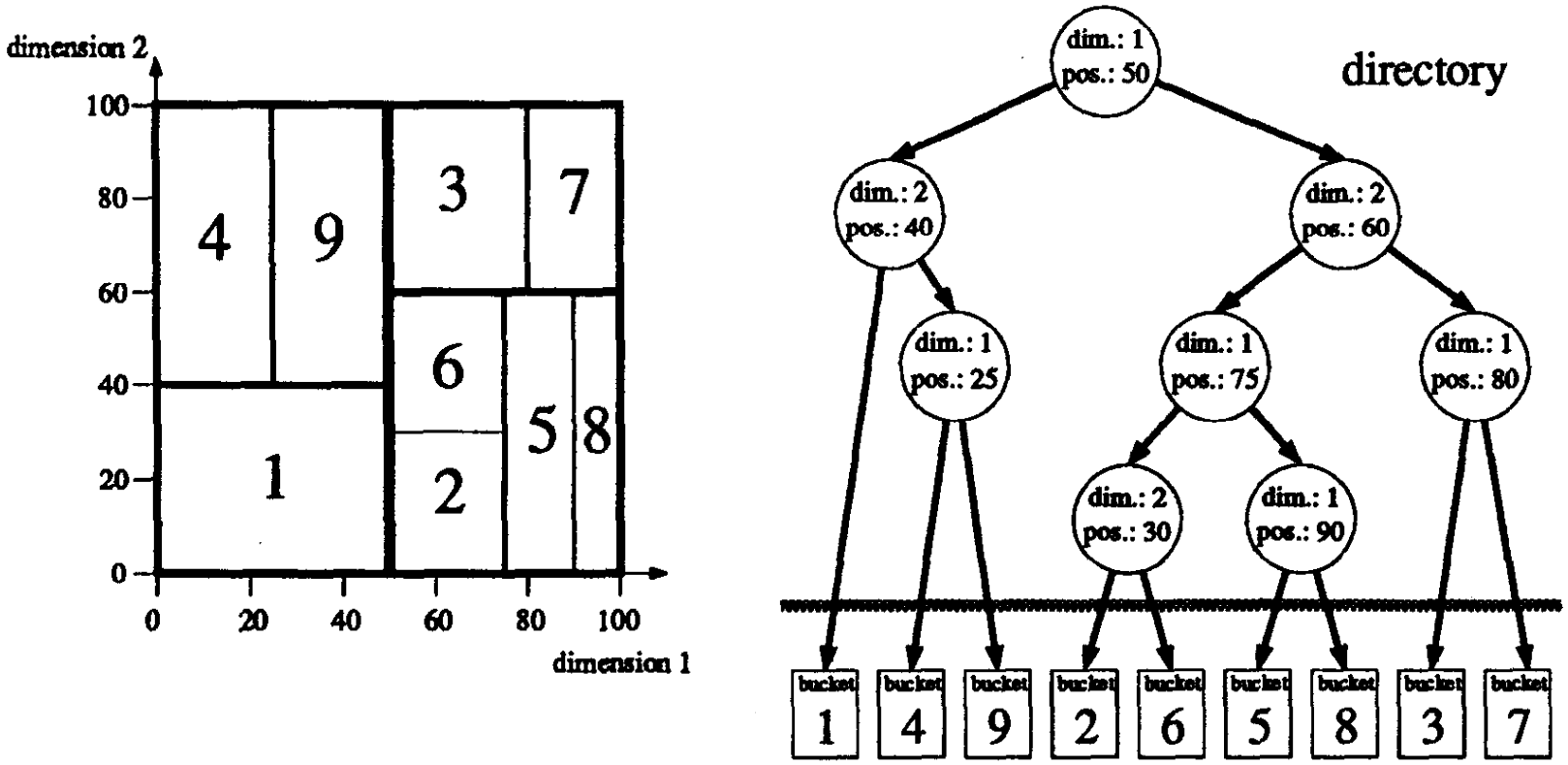
Indexing hierarchies (LSD-tree)

- These properties demonstrated by an extensive performance, evaluation make the LSD tree extremely suitable for the implementation of spatial access paths in geometric databases.
- The paging algorithm for the binary tree directory is interesting in its own right because a practical solution for the problem of how to page a (multidimensional) binary tree without access path degeneration is presented.

Indexing hierarchies (LSD-tree)

- The new structure partitions data space into pairwise disjoint cells with associated buckets of fixed size.
- The free choice of split positions is basis of graceful adaptation to arbitrary skew object distributions.
- Since a new split position can be chosen locally optimal, i.e. optimal with respect only to the cell to be split and independent from other existing cell boundaries, the new structure is called Local Split Decision tree (LSD tree)

Indexing by subspaces (LSD-tree)



LSD tree associated with the data space partition

Indexing hierarchies (LSD-tree)

- Besides the advantages of the LSD tree directory there are some drawbacks typical for multidimensional binary tree structures:
  - A multidimensional binary tree may become unbalanced, i.e. may contain long paths with almost no branches, and
  - no suitable method for paging a multidimensional binary tree is known.

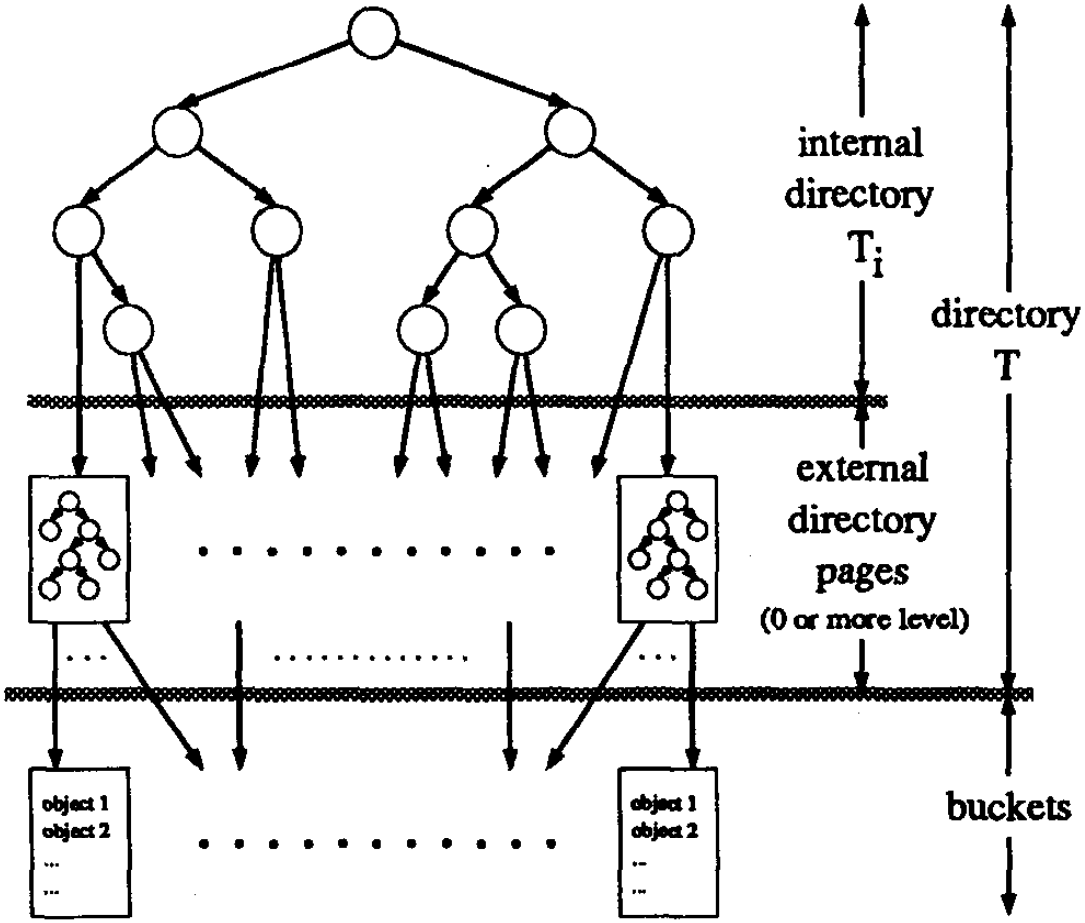
Indexing hierarchies (LSD-tree)

- To overcome these problems we introduce a paging algorithm with **external balancing property**: the number of external pages which are traversed on any two paths from the directory root to a bucket differs by at most 1.
- When geometric objects are inserted into an initially empty LSD tree the directory grows up to a size when it cannot be kept in the dedicated part of the main memory any longer.

Indexing hierarchies (LSD-tree)

- Then the paging algorithm determines a subtree to be paged on secondary storage such that the external balancing property is preserved.
- If the subtree consists of n_1 nodes, the main memory is then able to receive additional n_2 nodes until a further invocation of the paging algorithm must take place.

Overall structure of LSD-tree



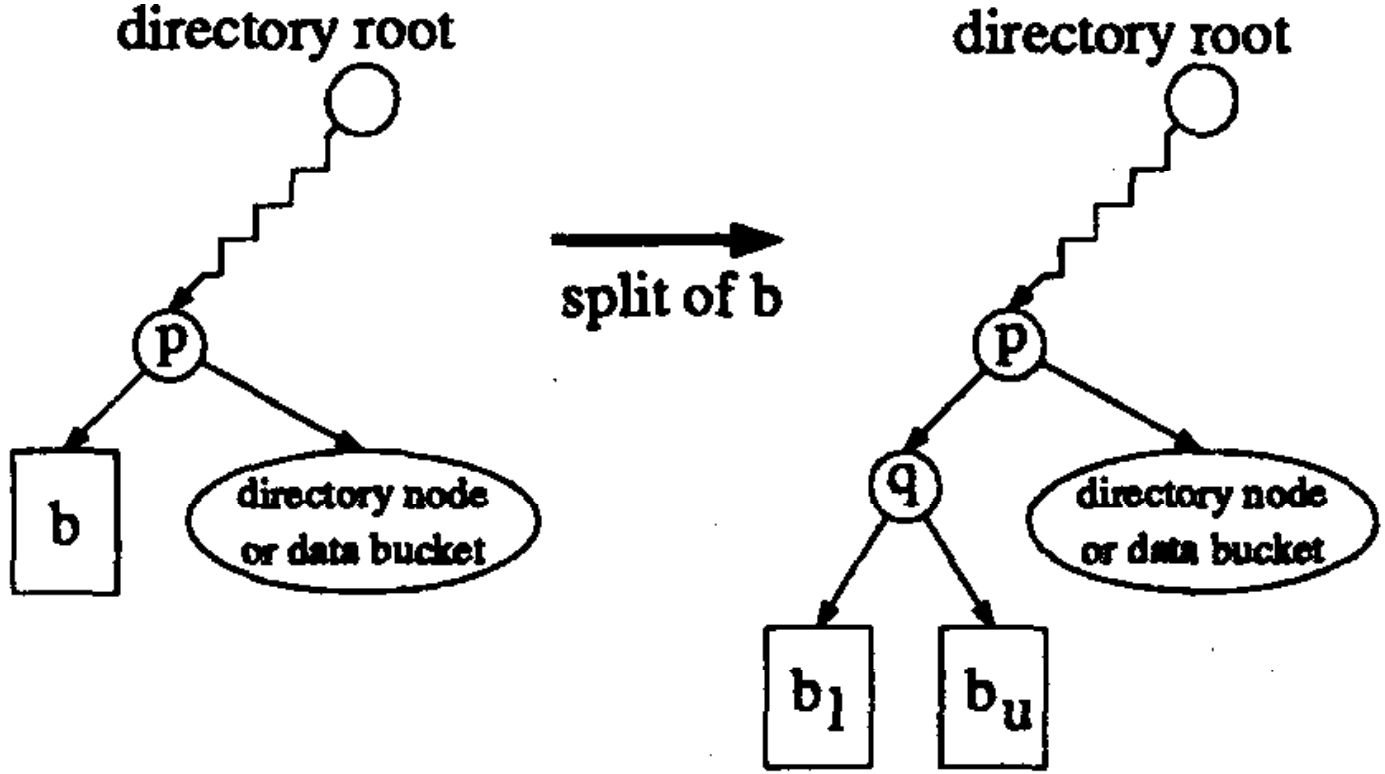
Indexing hierarchies (LSD-tree)

- The search for bucket b which will receive new object is guided by directory as in k-d-trees.
- If b does not overflow, the insertion is finished, otherwise bucket split algorithm creates two new buckets b_1 and b_2 from b according to a split strategy.

Indexing hierarchies (LSD-tree)

- In case of a bucket split the pointer in directory referencing b is changed to a pointer referencing a new directory node q, representing the split decision concerning b, i.e. the new node q is inserted into the directory by calling the directory insertion algorithm.
- The new node q points to new buckets b_1 and b_2.

Effect of a bucket split of LSD-tree



Indexing hierarchies (LSD-tree)

The paging algorithm is called - when after an insertion of an additional node the size of internal prefix tree T_i reaches the maximal possible number n_i of internal nodes.

The algorithm searches for a subtree T_s in T_i such that paging T_s preserves the external balancing property.

Indexing hierarchies (LSD-tree)

This property is preserved if T_s is a paging candidate, i.e. T_s fullfils the following properties:

- Any path from root of T_s down to a bucket contains minimal number of external directory pages (of all paths in directory T).
- The height of T_s is at most h_T

Directory before and after paging