
GRID FILE

E0 261

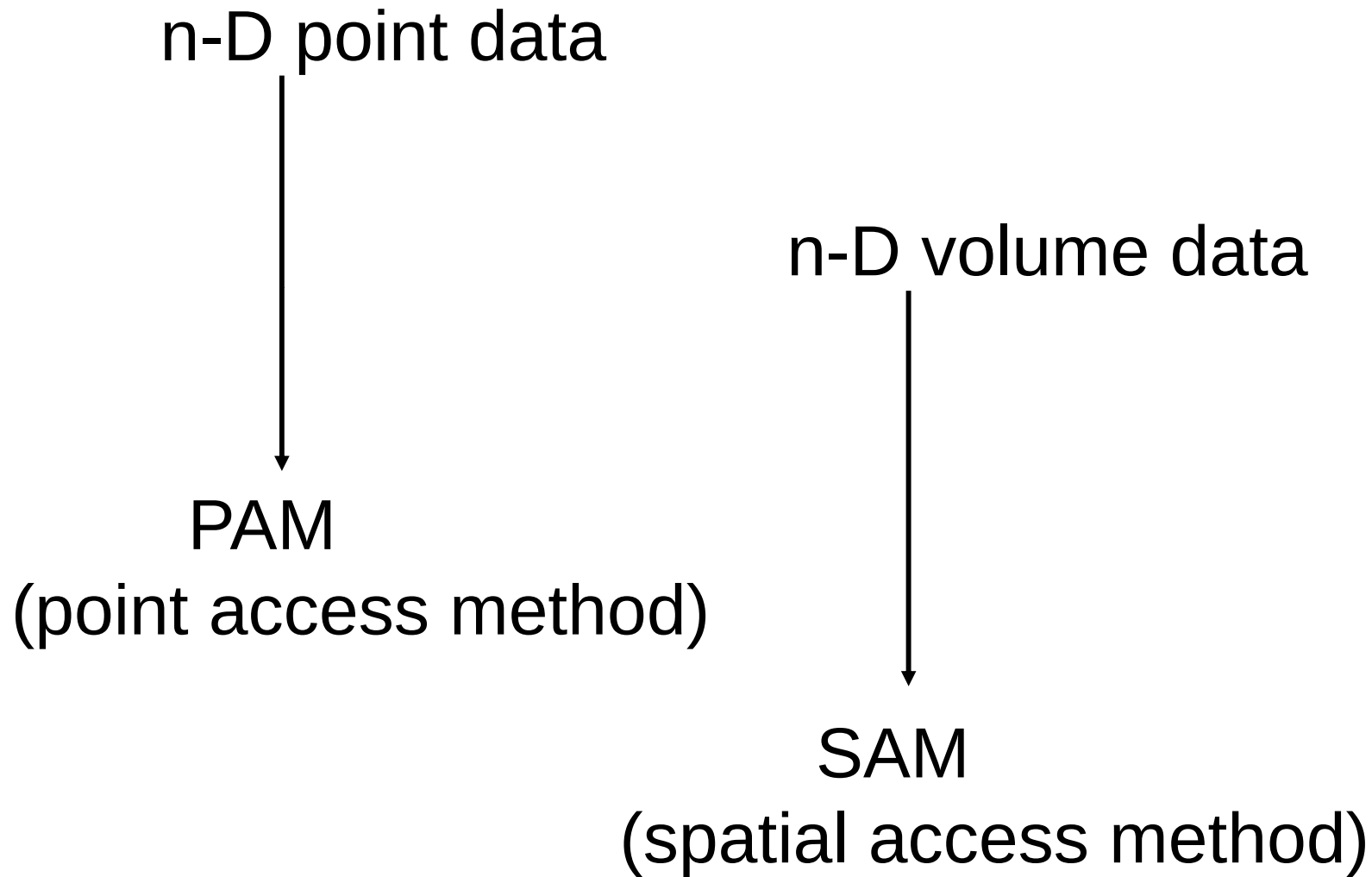
Jayant Haritsa

Computer Science and Automation

Indian Institute of Science



Multidimensional Access Methods



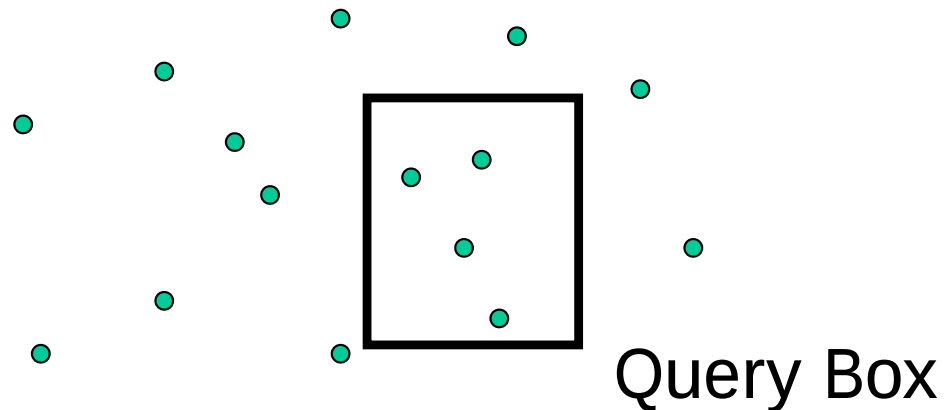
Data Geometry to Index Mappings

- 1-D PAM: B-tree
 - n-D PAM: Grid File
 - B-tree won't work since not symmetric
 - n-D SAM: R-tree
 - polygons, polylines
- LowDim
($n < 10$)
- Continuous
Domains

High-dimensional, categorical domains in
TIDS course

The n-D PAM Problem

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



Grid File Goals

- Symmetric structure (all attributes equal)
- Adaptable structure (automatically adjusts to skewed data distribution)
- Two disk access principle (exact match)
- Efficient range queries (close objects should be close in index)

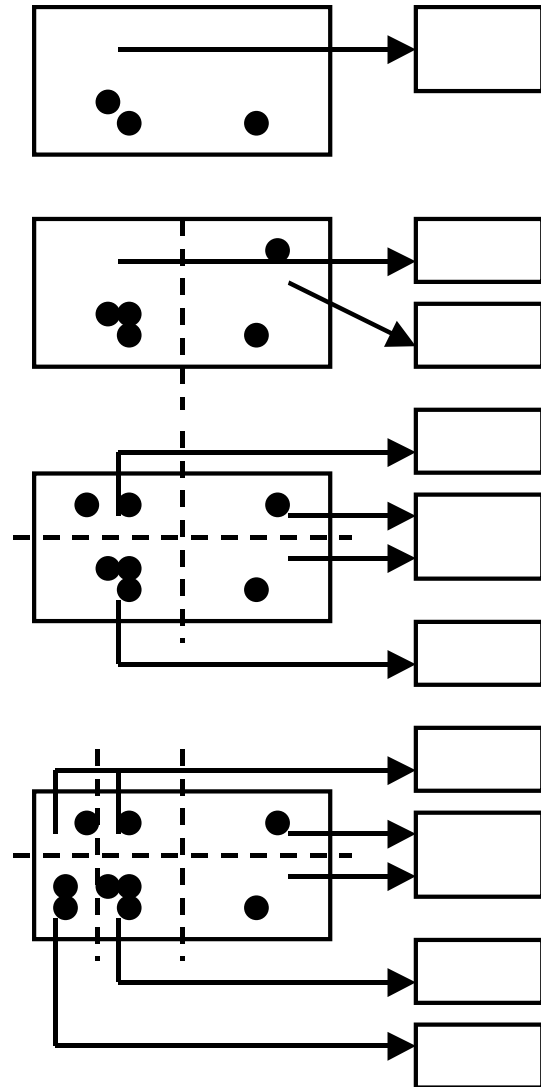


Grid File Technique

- Organize the **domain**, not the **data**
- Use a **grid** to partition the space
 - each cell of the grid is associated with a disk page
- Similar to Extendible Hashing in layout but can also handle range queries



Grid File Details



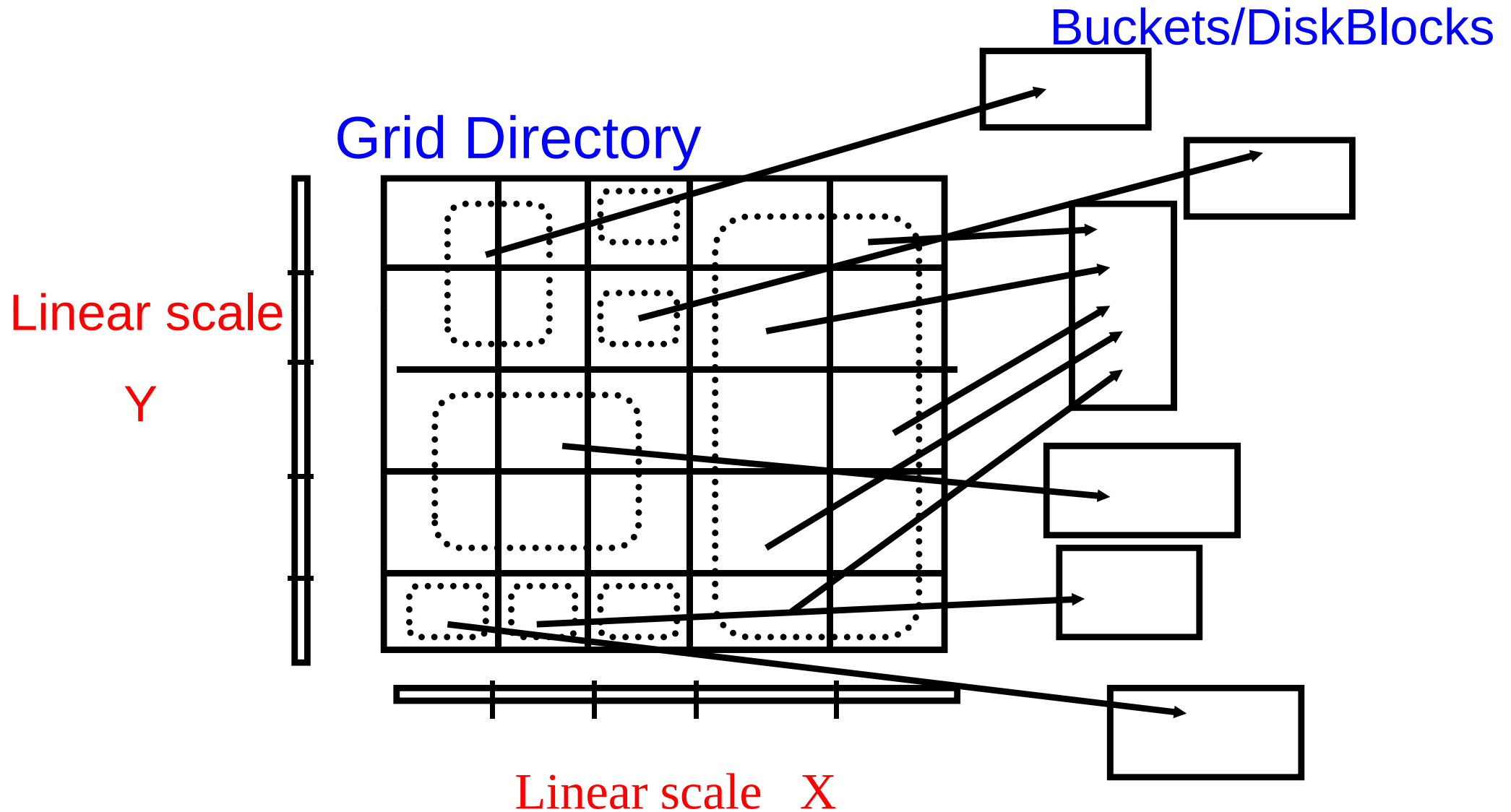
- Select dividers along each dimension. Partition space into cells
- Dividers *cut all the way*
- Each cell corresponds to 1 disk page
- Many cells can point to the same page
- Convex pages
- Cell directory potentially exponential in the number of dimensions

2-D Grid File Implementation

- Dynamic structure using a grid directory
 - Grid array: a 2-dimensional array with pointers to buckets (this array can be large, disk resident) $G(0, \dots, nx-1, 0, \dots, ny-1)$
 - Linear scales: Two 1 dimensional arrays that used to access the grid array (main memory) $X(0, \dots, nx-1), Y(0, \dots, ny-1)$



2-D Example



Grid File Search

- Exact Match Search: at most 2 I/Os assuming linear scales fit in memory.
 - First use linear 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)



Grid File Search

- Range Queries:
 - Use linear scales to determine the indexes into the cell directory
 - Access the cell directory to retrieve the bucket addresses of buckets to visit
 - Access the buckets



Grid File Insertions

- Determine the bucket into which insertion must occur.
- If space in bucket, insert.
- Else, split bucket
 - how to choose a good dimension to split?
 - cyclic, random, weighted, ...
 - how to choose a good location to split?
 - domain units, equi-occupancy, ...
- If bucket split causes the cell directory to split, do so and adjust linear scales.
- Insertion of these new entries potentially requires a complete reorganization of the cell directory---expensive!!!



Grid File Deletions

- Deletions may decrease the space utilization. Therefore, merge buckets.
- We need to decide which cells to merge and a merging threshold
- Buddy system and neighbor system
 - A bucket can merge with only one *buddy* in each dimension
 - Merge adjacent regions if the result is a rectangle
- Merge threshold ~ 70 percent occupancy



Grid Array Implementation

- Fast insertion/deletion of (n-1D) cross-sections
 - Linked lists
 - space inefficient
 - pointer chasing on disk
- K-dimensional equi-interval array
 - size based on smallest interval in each linear scale
 - multidimensional counterpart of extendible hashing directory, size doubles abruptly
 - skewed data results in large directory, since no randomization of data



Grid Array Implementation (contd)

- TIFR Mumbai approach (contiguous)
 - address computation becomes complicated
- Settle for a conventional multi-dimensional array
- Can have multi-level grid files, like in multi-level indexing
- Concurrency control is distributed, not centralized



Problems

- Grid directory maintenance
- Does not handle correlated attributes well
- Restricted to low-dimensional continuous data



END GRID FILE

E0 261

