Database Management Systems

Lecture 12

Parallel Databases*

Spatial Databases*

* not among the exam topics

Sabina S. CS

Parallel Databases

Parallel Database Systems

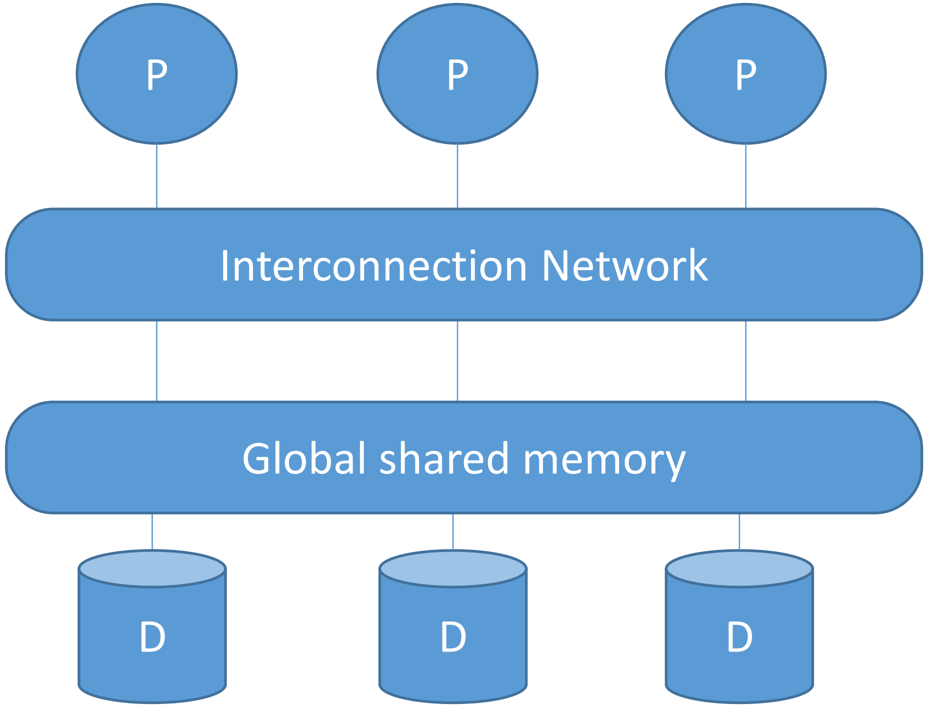
- performance improvement
    - parallelize operations:
        - loading data
        - building indexes
        - query evaluation
    - data can be distributed, but distribution is dictated solely by performance reasons

Parallel Databases - Architectures

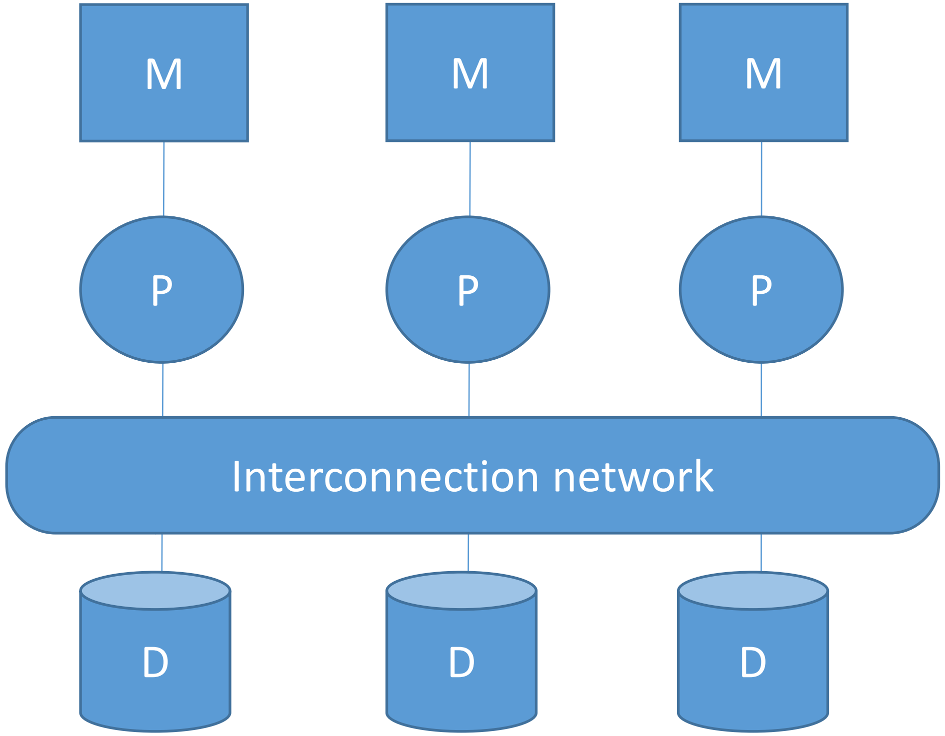
- *shared-memory*
- *shared-disk*
- *shared-nothing*

Parallel Databases - Architectures

- *shared-memory*
  - several CPUs:
    - attached to an interconnection network
    - can access a common region in the main memory

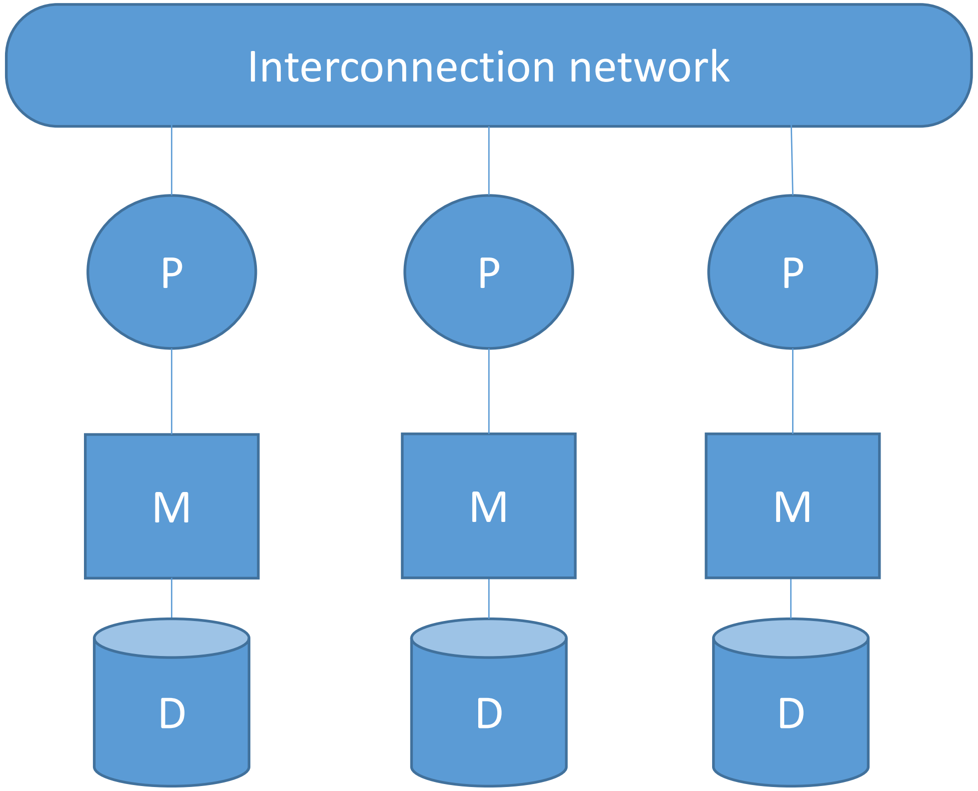


- *shared-disk*
  - a CPU:
    - its own private memory
    - can access all disks through a network



Parallel Databases - Architectures

- *shared-nothing*
  - a CPU:
    - its own local main memory
    - its own disk space
  - 2 different CPUs cannot access the same storage area
  - CPUs communicate through a network



Interference

- specific to shared-memory and shared-disk architectures
- add CPUs:
  - increased contention for memory and network bandwidth

  => existing CPUs are slowing down
- main reason that led to the shared-nothing architecture, currently considered as the best option for large parallel database systems

The Shared-Nothing Architecture

- linear speed-up & linear scale-up

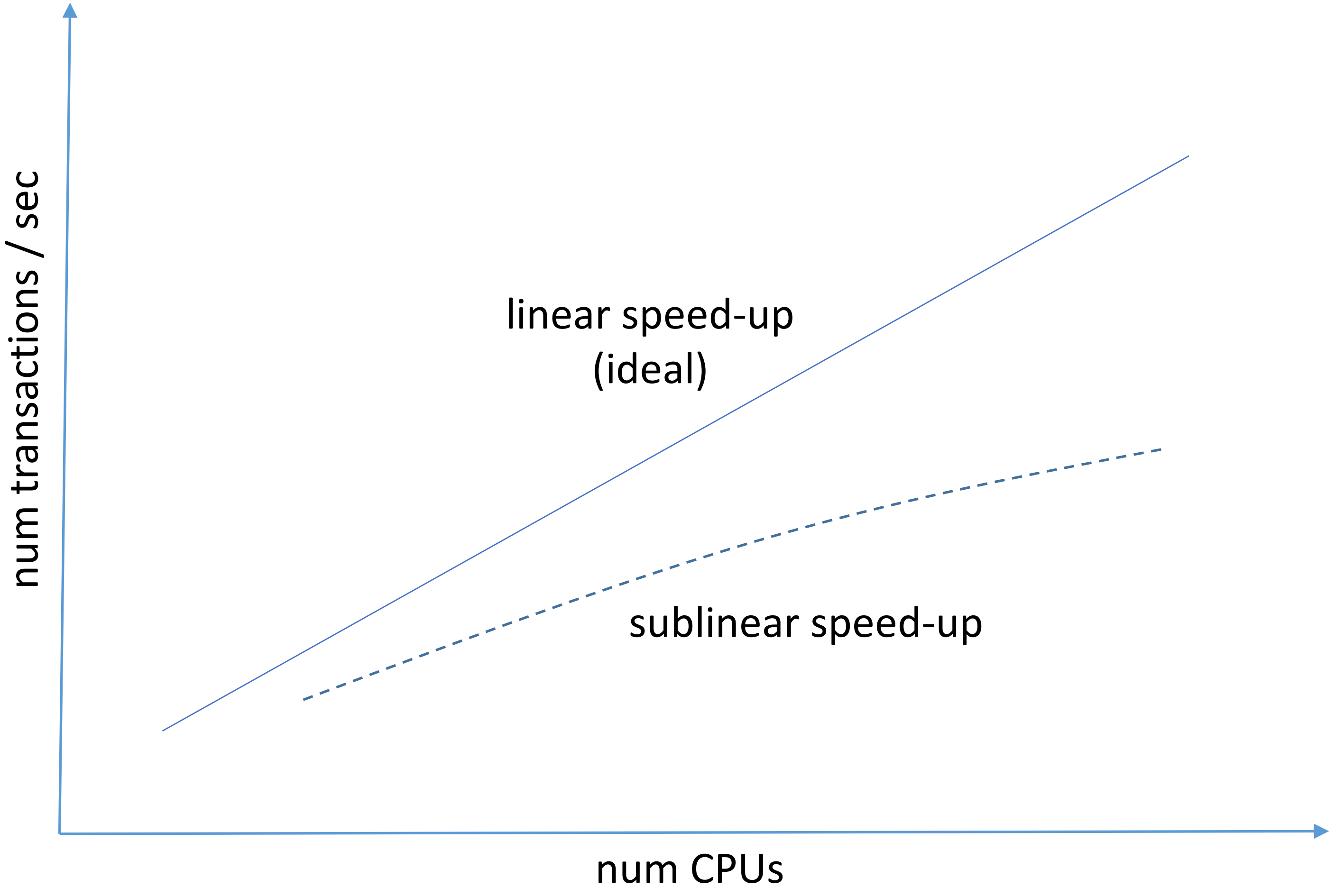
- linear speed-up
    - required processing time for operations decreases proportionally to the increase in the number of CPUs and disks

- linear scale-up
    - num. of CPUs and disks grows proportionally to the amount of data

  => performance is sustained

The Shared-Nothing Architecture

- speed-up

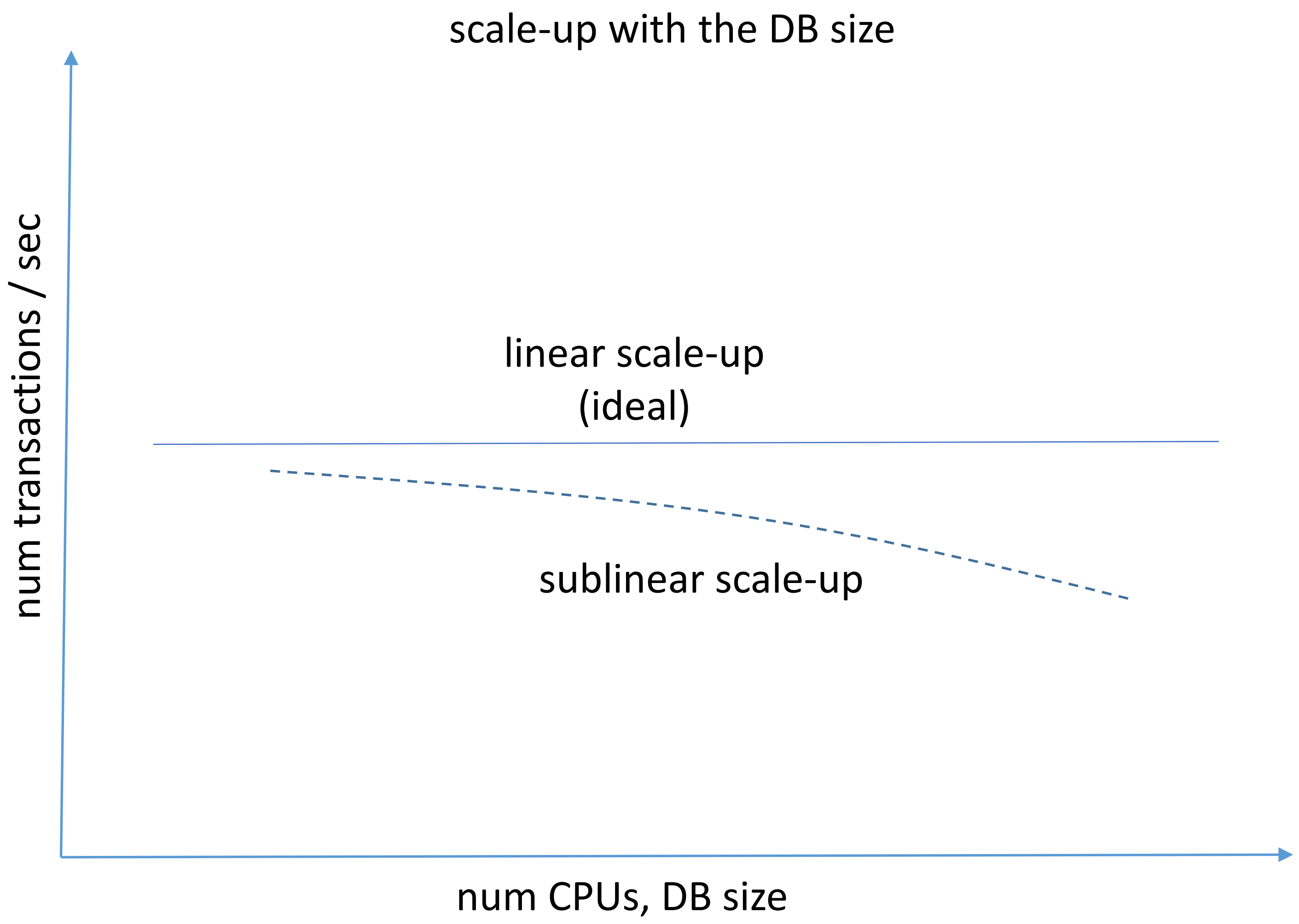


- DB size - fixed
- add CPUs

=> more transactions can be executed per second

The Shared-Nothing Architecture

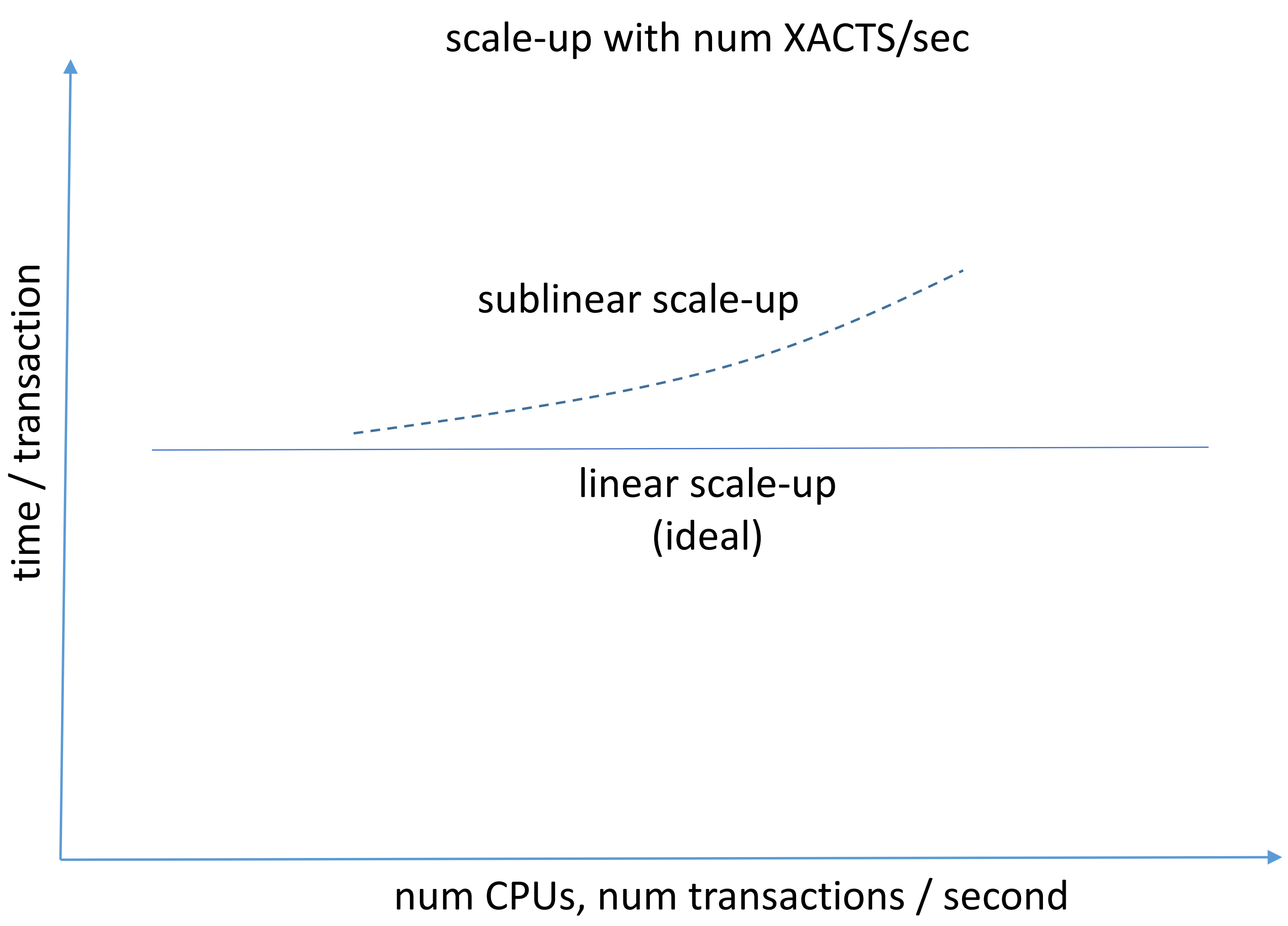
- scale-up



- the number of transactions executed per second, as the DB size and the number of CPUs increase

The Shared-Nothing Architecture

- scale-up



* alternative

- add CPUs as the number of transactions executed per second increases
- evaluate the time required for a transaction

Parallel Query Evaluation

- context
  - DBMS based on a shared-nothing architecture
- evaluate a query in a parallel manner
- operators in an execution plan can be evaluated in parallel
  - 2 operators are evaluated in parallel
  - one operator is evaluated in a parallel manner
- an operator is said to *block* if it doesn't produce results until it consumes all its inputs (e.g., sorting, aggregation)
- pipelined parallelism
  - an operator consumes the output of another operator
  - limited by blocking operators

Parallel Query Evaluation

- parallel evaluation on partitioned data
  - every operator in a plan can be evaluated in a parallel manner by partitioning the input data
  - partitions are processed in parallel, the results are then combined
- processor = CPU + its local disk

Parallel Query Evaluation – Data Partitioning

- horizontally partition a large dataset on several disks
- partitions are then read / written in parallel
- *round-robin* partitioning
    - n processors
    - the i^{th} tuple is assigned to processor i % n
- hash partitioning
    - determine the processor for a tuple t
        - apply a hash function to t ((some of) its attributes)
- range partitioning
    - n processors
    - order tuples conceptually
    - choose n ranges for the sorting key values s.t. each range contains approximately the same number of tuples
    - tuples in range i are assigned to processor i

Parallel Query Evaluation – Data Partitioning

- queries that scan the entire relation
    - round-robin partitioning - suitable
- queries that operate on a subset of tuples
    - equality selection, e.g., age = 30
        - tuples partitioned on the attributes in the selection condition, e.g., age
        - hash and range partitioning are better than round-robing (one can access only the disks containing the desired tuples)
    - range selection, e.g., 20 < age < 30
        - range partitioning is better than hash partitioning (it’s likely that the desired tuples are grouped on several processors)

Parallelizing Individual Operations

- context
  - DBMS based on a shared-nothing architecture
- each relation is horizontally partitioned on several disks
- scanning a relation
  - pages can be read in parallel
  - obtained tuples can than be reunited
  - similarly – obtain all tuples that meet a selection condition

Parallelizing Individual Operations

- sorting
  - v1
    - each CPU sorts the relation fragment on its disk
    - subsequently, the sorted tuple sets are merged
  - v2
    - redistribute tuples in the relation using range partitioning
    - each processor sorts its tuples with a sequential sorting algorithm => several sorted runs on the disk
    - merge runs => sorted version of the set of tuples assigned to the current processor
    - obtain the entire sorted relation
      - visit processors in an order corresponding to their assigned ranges and scan the tuples

Parallelizing Individual Operations

- sorting
  - v2
    - challenges
      - range partitioning – assign approximately the same number of tuples to each processor
      - a processor that receives a disproportionately large number of tuples will limit scalability

Spatial Databases

Types of Spatial Data and Queries

- spatial data
  - multidimensional points
  - lines
  - rectangles
  - cubes
  - etc.
- spatial extent (SE)
  - region of space occupied by an object
  - characterized by location + boundary

Types of Spatial Data and Queries

- DBMS
    - point data
    - region data

- point data
    - collection of points in a multidimensional space
    - point
        - SE – only location
        - no space, area, volume
    - direct measurements (e.g., MRI)
    - transforming data objects (e.g., feature vectors)

Types of Spatial Data and Queries

- region data
  - collection of regions
  - region - SE
    - location
      - position of a fixed anchor point for the region
    - boundary (line, surface)
  - geometric approximations to objects, built with line segments, polygons, spheres, cubes, etc.
  - e.g., in geographic apps
    - line segments – roads, railways, rivers, etc.
    - polygons – lakes, counties, countries, etc.

Types of Spatial Data and Queries

- spatial queries
    - spatial range queries
    - nearest neighbor queries
    - spatial join queries

    - spatial range queries
        - associated region
            - location + boundary
        - find all regions that overlap / are contained within the specified range
        - e.g., Find all cities within 150 km of Constanța.
        - e.g., Find all rivers in Bihor.

Types of Spatial Data and Queries

- spatial queries
  - nearest neighbor queries
    - e.g., Find the 5 cities that are nearest to Paris.
    - usually, answers are ordered by proximity

  - spatial join queries
    - e.g., Find all cities near Bucharest.
    - e.g., Find pairs of cities within 150 km of each other.
    - meaning of queries can be determined by the level of detail in the representation of objects
    - e.g., relation in which each tuple is a point that represents a city
      - answer the queries with a self join
      - join condition – distance between 2 tuples in the query result

Types of Spatial Data and Queries

- spatial queries
  - spatial join queries
    - e.g., cities have a boundary

    => meaning of queries and evaluation strategies become more complex
      - cities whose centroids are within 150 km of each other?
      - or cities whose boundaries come within 150 km of each other?

Applications

- relation R with k attributes seen as a collection of k-dimensional points
- Geographic Information Systems (GIS)
  - point & region data
  - e.g., a map that contains the locations of several small objects (points), highways (lines), cities (regions)
  - range, nearest neighbor, join queries
- Computer-aided design (CAD) systems, medical imaging systems
  - store spatial objects
  - point & region data
  - most frequent queries - range & join
  - spatial integrity constraints

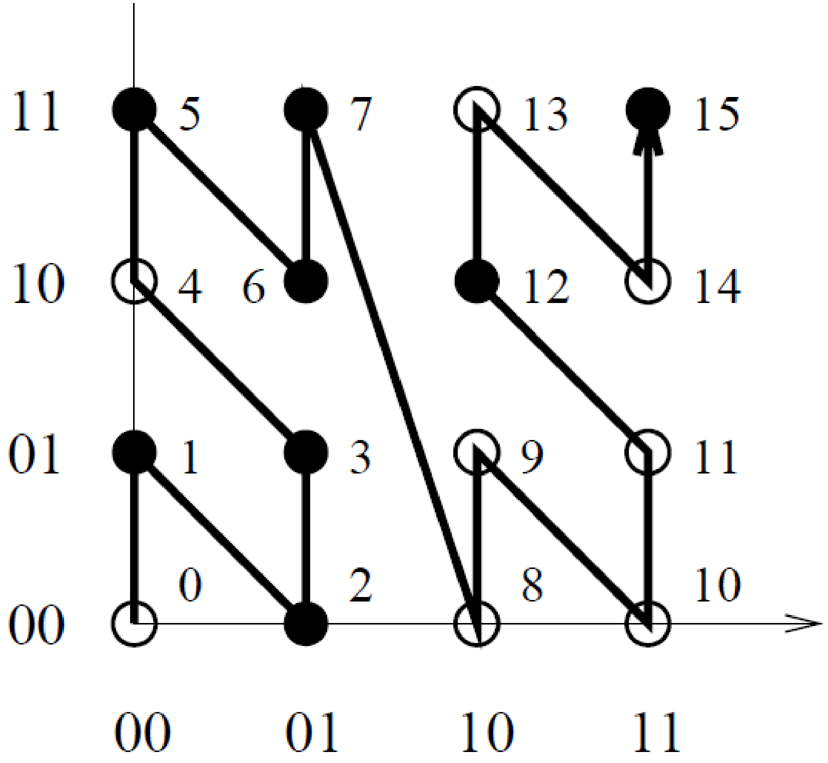
Applications

- multimedia DBs
  - multimedia objects
    - images, text, *time-series* data (e.g., audio)
  - object
    - point in a multidimensional space
  - similarity of 2 multimedia objects
    - distance between the corresponding points
  - similarity queries seen as nearest neighbor queries
  - point data & nearest neighbor queries – most common

Indexing - *space-filling curves*

- assumption
  - any attribute value can be represented with a fixed num. of bits, e.g., k bits

  => max. num. of values per dimension = 2^k
- the point with X = 01, Y = 11 has Z-value = 0111, obtained by interleaving the bits of X and Y
  - the 8^{th} point visited by the space-filling curve, which starts at point X=00, Y=00
- points in the dataset are stored in the order of their Z-values
- Z-ordering curve
  - linear ordering on the domain



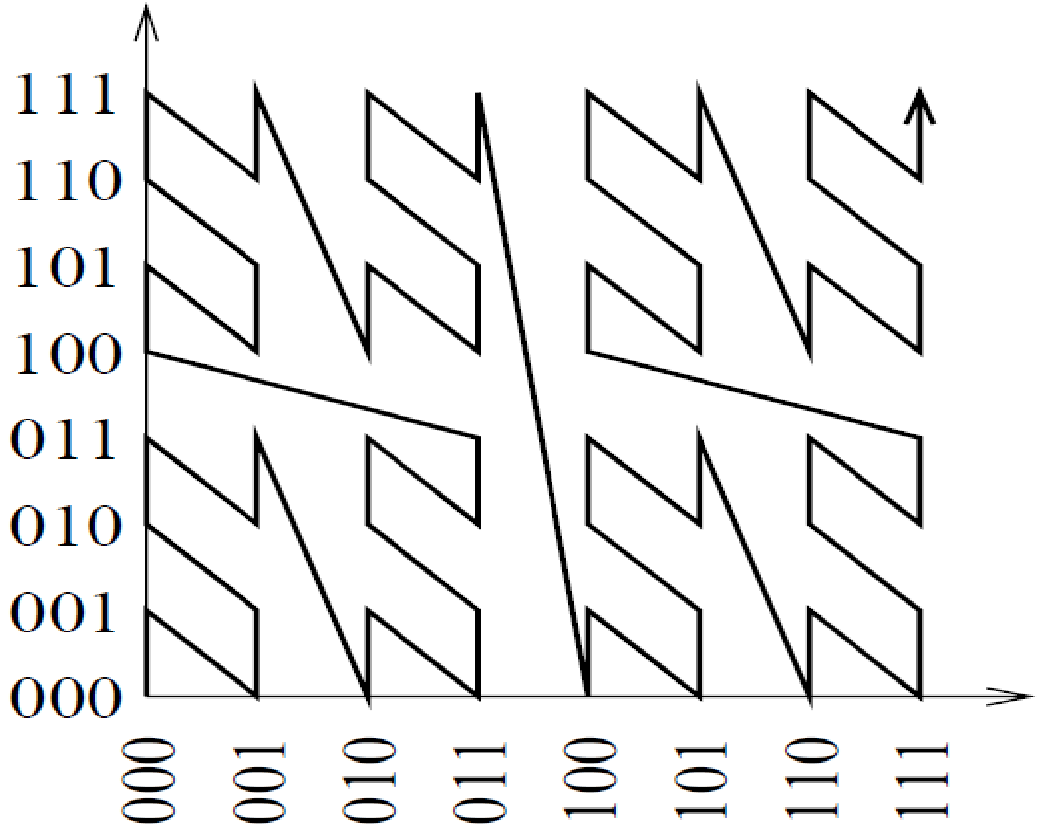
Z-ordering with 2 bits

Indexing - *space-filling curves*

- Z-ordering curve
  - the curve visits all the points in a quadrant before moving on to the next quadrant; all points in a quadrant are stored together
- indexing
  - B+ tree
  - search key: Z-value
  - store the Z-value of the point (along with the point)
  - I / D / search point
    - compute Z-value
    - I / D / search into / from / the B+ tree
  - unlike when using traditional B+ tree-based indexing, points are clustered together by spatial proximity in the X-Y space
  - spatial queries in the X-Y space become linear range queries
    - efficient evaluation on the B+ tree organized by Z-values

Indexing - *space-filling curves*

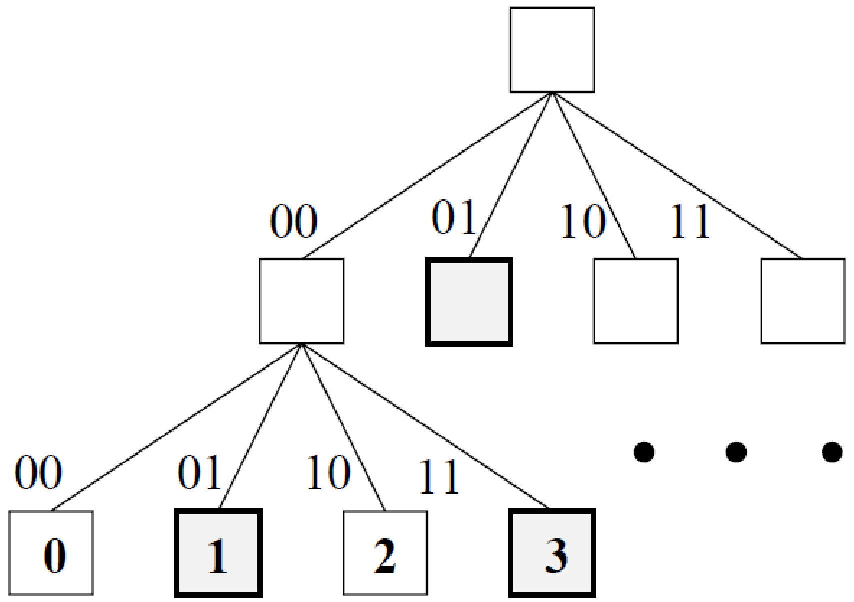
- Z-ordering curve when k = 3

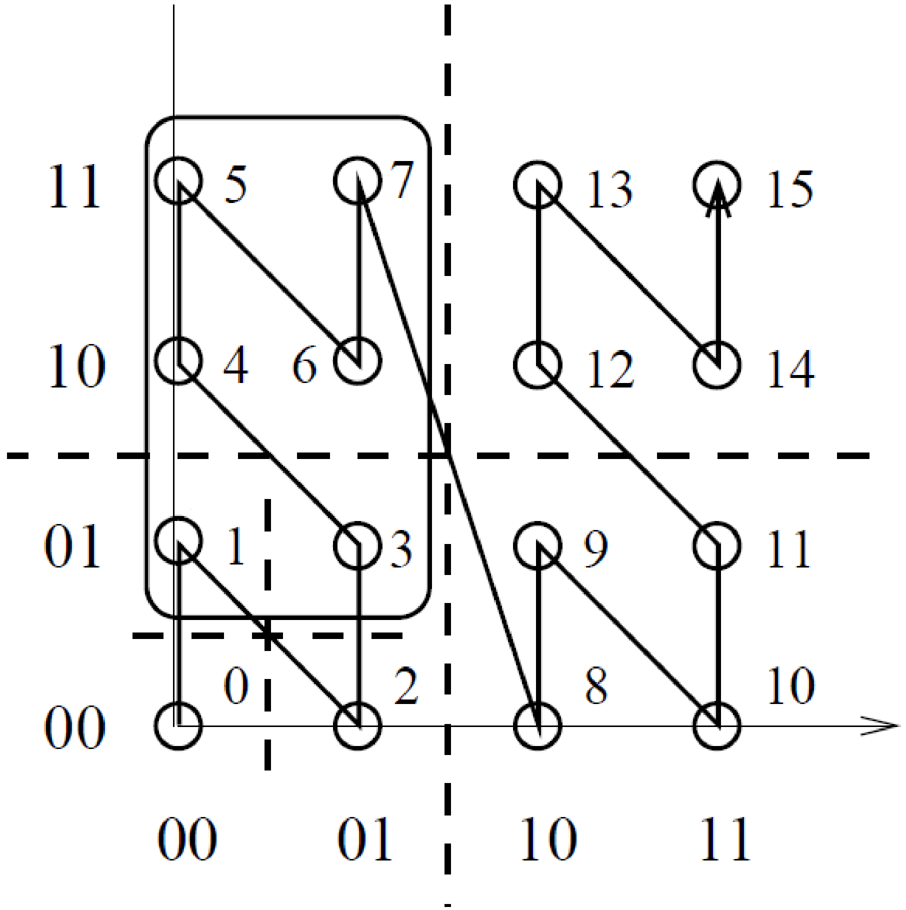


Z-ordering with 3 bits

Region Quad Trees

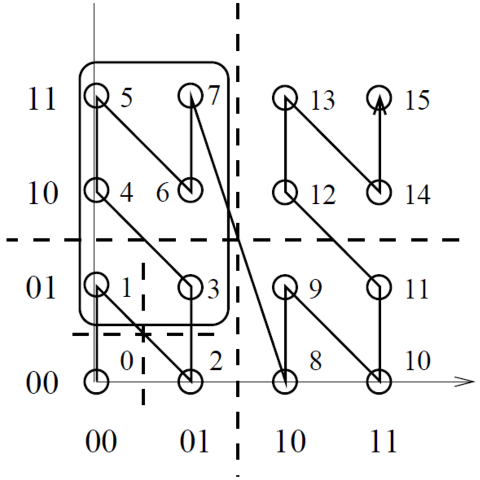
- region data
- Z-ordering recursively decomposes space into quadrants and subquadrants
- the structure of a Region Quad tree directly corresponds to the recursive decomposition of the data space
- each node in the tree corresponds to a square region in the data space

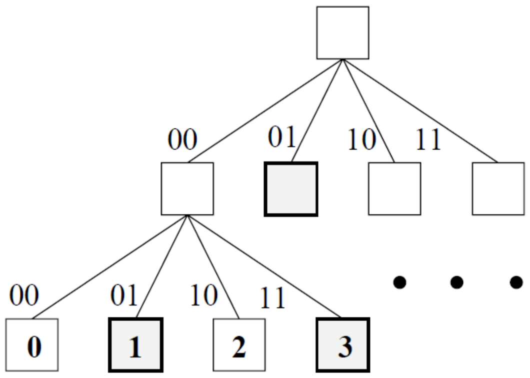




Region Quad Trees

- root – entire data space
- internal node – 4 children
- rectangle object R stored by the DBMS
  - all points in the 01 quadrant of the root and the points with Z-values 1 and 3
  - store 3 records: <0001, R>, <0011, R>, <01, R>
  - records clustered and indexed by the 1st field (the Z-value) in a B+ tree
- use B+ trees to implement Region Quad trees
- generalization - k dimensions => at every node, the space is partitioned into 2^k subregions





References

- [Ra02] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (3rd Edition), McGraw-Hill, 2002
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga09] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book (2nd Edition), Pearson Education, 2009
- [Ra02S] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, Slides for the 3rd Edition, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Si19] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts (7th Edition), McGraw-Hill, 2019
- [Si19S] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, Slides for the 7th Edition, http://codex.cs.yale.edu/avi/db-book/
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html