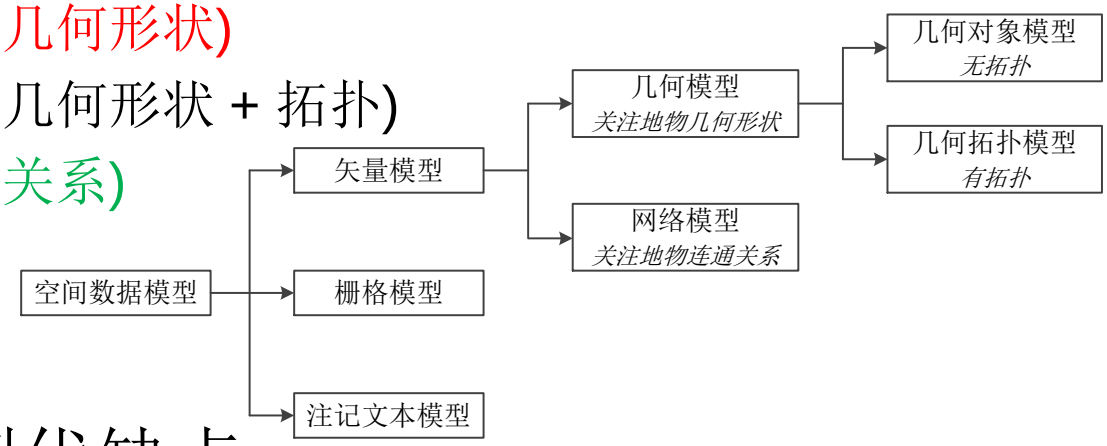
第五章 空间存储与索引

陶煜波
计算机科学与技术学院

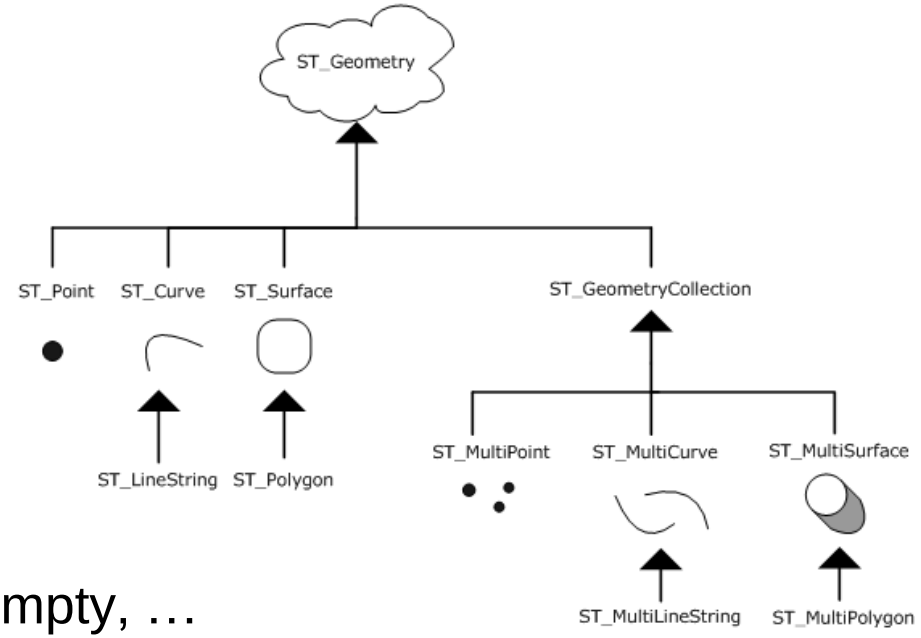
几何对象模型与查询回顾

- 空间数据模型分类
  - 矢量模型
    - 几何对象模型 (地物几何形状)
    - 几何拓扑模型 (地物几何形状 + 拓扑)
    - 网络模型 (地物连通关系)
  - 栅格模型
  - 注记文本模型
- 矢量模型和栅格模型优缺点
- 概念模型 → 逻辑模型 → 物理模型



几何对象模型

- 对象关系数据库：数据+方法（C++中的类）
- 几何对象层次关系
  - 坐标维数和几何维数
  - 边界、内部、外部
  - 九交矩阵
- 几何对象方法
  - 常规方法 (12种)
    - Dimension, Boundary, IsEmpty, …
  - 常规GIS分析方法 (7种)
    - Distance, Buffer, ConvexHull, Intersection, Union, Difference, SymDifference
  - 空间查询方法 (11种)
    - 包含于(within): 若$a \cap b = a$，且$\mathrm{I}(a) \cap E(b) = \emptyset$，则a包含于b内



空间拓扑关系

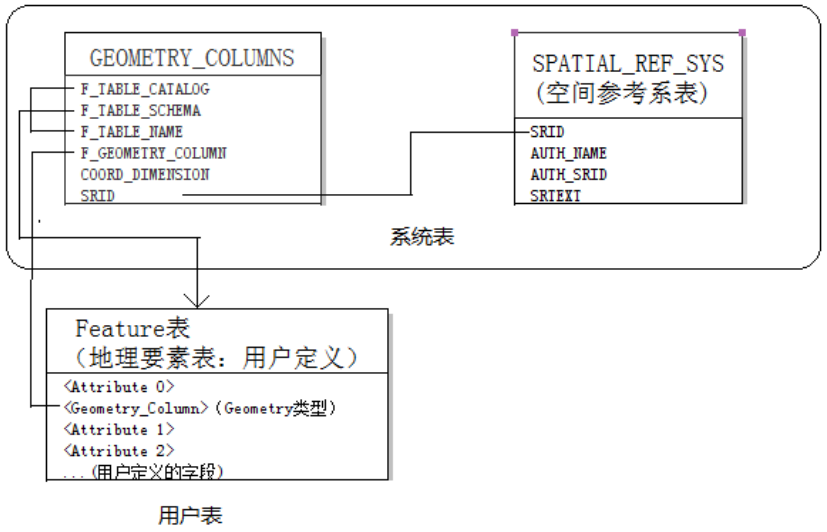
空间关系	定义	九交矩阵
Equals	$a \subseteq b$，$a \supseteq b$	TFFFTFFFT
Disjoint	$a \cap b = \emptyset$	FF*FF****
Touches	$\mathrm{I}(a) \cap I(b) = \emptyset$，$a \cap b \neq \emptyset$	FT******* F**T***** F***T****
Crosses	$\mathrm{I}(a) \cap I(b) \neq \emptyset$ ， $a \cap b \neq a$， $a \cap b \neq b$	T*T****** (点/线, 点/面, 线/面) 0******** (线/线)
Within	$a \cap b = a$，$\mathrm{I}(a) \cap E(b) = \emptyset$	T*F**F***
Overlaps	Dim(I(a)) = Dim(I(b)) = Dim(I$(a) \cap I(b)$)，$a \cap b \neq a$，$a \cap b \neq b$	T*T***T** (点/点, 面/面) 1*T***T** (线/线)
Contains	a.Contains(b) ←→ b.Within(a)	?
Intersects	a.Intersects(b) ←→ !b.Disjoint(b)	?

几何对象模型

- 空间查询方法 (11种)
  - Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps, Relates
  - LocateAlong, LocateBetween
- 空间关系 (8种)
  - 相离(disjoint), 相交(intersects), 相等(equals)
    - a.Intersects(b) ↔ !a.Disjoint(b)
  - 交叠(overlaps), 包含于(within), 包含(contains)
    - a.Contains(b) ↔ b.Within(a)
    - a.Equals(b) → a.Overlaps(b) ?
  - 相接(touches), 穿越(crosses)
    - a.Touches(b) (或a.Crosses(b))→ a.Intersects(b) ?
    - a.Touches(b) → a.Crosses(b) ?

几何对象模型

- 逻辑模型
  - 基于预定义数据类型的实现
    - numeric和BLOB
  - 基于扩展几何类型的实现
    - Geometry类
- 表模式
  - 系统表
    - GEOMETRY_COLUMNS和SPATIAL_REF_SYS
  - 用户表
    - Feature和Geometry
- 物理模型
  - WKB和WKT



基于扩展Geometry类型的要素表模式

空间数据库

- 空间数据库 = 对象关系/关系数据库 + 空间扩展
  - Oracle + Oracle Spatial
  - SQL Server + SQL Server Spatial
  - PostgreSQL + PostGIS
  - MySQL + MySQL Spatial
  - SQLite + SQLite Spatialite
- PostGIS
  - 提供了空间数据类型、空间函数和空间索引
  - Geometry (Point/Line/Polygon/Multixxx, 空间参考系)
  - ST_XXX
  - GiST

应用举例：打车软件

- 需要哪些空间数据？
  - 出租车空间位置
    - Taxi(ID, driverID, ……, status, pos(Point, 4326))
  - 乘客空间位置
    - User(ID, name, ……, pos(Point, 4326))
  - 道路
    - Road(ID, name, ……, line(LineString, 4326))
- 乘客(ID = A)的附近1公里内的空车？(注意距离单位)
  - Select T.ID, T.position
    From Taxi T, User U
    Where U.ID = A and ST_Distance(T.pos, U.pos) < 1000
     and T.status = 0;
- 出租车(ID = B)附近1公里内的乘客叫车？

应用举例：打车软件

- 需要哪些空间数据？
  - Taxi(ID, driverID, ……, status, pos(Point, 4326))
  - User(ID, name, ……, pos(Point, 4326))
  - Road(ID, name, ……, line(LineString, 4326))
- 利用出租车数量评估道路拥堵情况，出租车在道路100米内认为在该道路上
  - Select R.ID, count(*)
    From Taxi T, Road R
    Where ST_Distance(T.pos, T.line, false) < 100
    Group by R.ID;
  - 是否输出当前没有出租车所在的道路？
  - 这样的评价方式合理吗？

应用举例：打车软件

- 通常，数据库保留所有时间上的信息，即
  - Taxi(ID, driverID, ……, status, pos(Point, 4326), time)
  - User(ID, name, ……, pos(Point, 4326), time)
  - Road(ID, name, ……, line(LineString, 4326))
- 应如何修改上述SQL语句？
- 出租车(ID = B)附近1公里内的乘客叫车？
  - Select T.ID, T.position
    From Taxi T, User U
    Where T.ID = B and ST_Distance(T.pos, U.pos) < 1000
     and T.time ……
     and U.time ……
  - 这样的实现方式合理吗？

个人空间数据

- iPhone GPS
  - http://vimeo.com/26600798
- Atlas of the Habitual
  - http://www.tlclark.com/Atlas-of-the-Habitual
- Basketball
  - https://flowingdata.com/tag/basketball/
- Game
  - http://game.academy.163.com/library/2015/2/12/17721_498621.html
  - 百度图片搜索“pokemon go map”
- 挑战6 基于车辆轨迹数据的证据收集
- 挑战7 基于移动轨迹数据的群体发现

追你到天涯海角

http://vimeo.com/26600798

SuperMap杯全国高校GIS大赛2019

- 应用分析组
  - a) 通过缓冲区分析功能计算道路拓宽可能波及的居民地面积，基于此类分析和统计制定道路拓宽方案
  - b) 通过物流配送功能分析住宅小区内送水的最优路线，并结合业主的作息时间，制定小区内送水的最佳方案
  - c) 通过栅格分析功能，对某一地区历年降水分布情况进行统计，分析该地区的降水变化情况
  - d) 通过选址分区功能，对某一地区进行分析，计算还需要增加多少超市或银行才可以实现全覆盖

ChinaVis 2019 数据可视分析挑战赛

- 主题：时空移动轨迹可视分析 http://chinavis.org
  - 作品提交截止日期：2019年6月10号
- 挑战1 – 智能场馆安全运营（360企业安全集团）
  - 数据：无线传感器技术获取参会人员的实时位置信息
    - 请您通过分析数据，推测会议期间主会场和各分会场的日程安排。（建议参赛者回答此题文字不多于800字，图片不多于5张）
    - 请您分析会议期间会场内的人员类型，总结各类型人员的移动规律。（建议参赛者回答此题文字不多于1000字，图片不多于10张）
    - 请您找出至少5个会议期间值得关注的异常事件。（建议参赛者回答此题文字不多于1000字，图片不多于10张）
    - 您认为这次会议在组织和管理方面有哪些不足？（建议参赛者回答此题文字不多于500字，图片不多于3张）
- 挑战2 – 待定（滴滴出行）

第五章 空间存储与索引

- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

References:

Spatial Databases: A Tour, Chapter 4

15-826: Multimedia Databases and Data Mining

空间数据库管理系统概论，第六章

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

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

What is a Physical Data Model?

- 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!

思考：删除主键和外键能否提高数据导入效率？

Concepts in a Physical Data Model

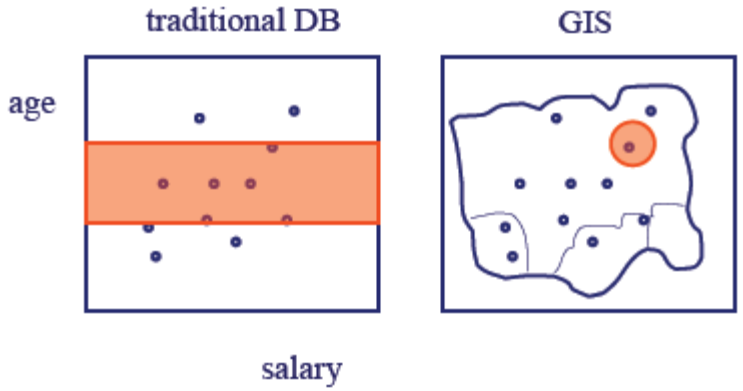
- Database concepts
  - Conceptual data model - entity, (multi-valued) attributes, relationship, …
  - Logical model - relations, atomic attributes, primary and foreign keys
  - Physical model - secondary storage hardware, file structures, indices, …
- Examples of physical model concepts from relational 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, ...

Physical Data Model for SDBMS

- Is relational DBMS physical data model suitable for spatial data?
  - Relational DBMS has simple values like numbers
  - Sorting, search trees are efficient for numbers
  - These concepts are not natural for spatial data (e.g. points in a plane)
- Solutions
  - Reusing relational physical data model concepts
  - New spatial techniques



Physical Data Model for SDBMS

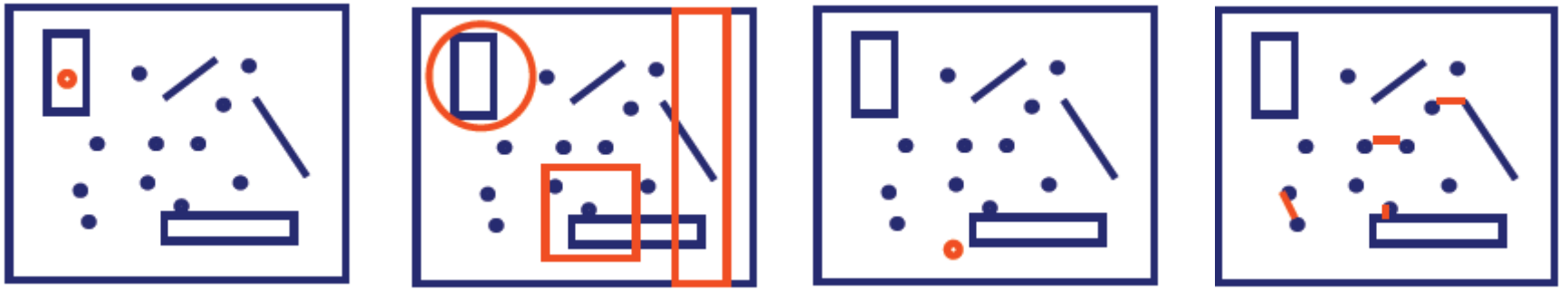
- Is relational DBMS physical data model suitable for spatial data?
- Reusing relational physical data model concepts
  - Space filling curves define a total order for points
  - This total order helps in using ordered files, search trees
  - But may lead to computational inefficiency
- New spatial techniques
  - Spatial indices, e.g. grids, hiearchical collection of rectangles
  - Provide better computational performance

Common Assumptions for SDBMS Physical Model

- Spatial data
  - Dimensionality of space is low, e.g. 2 or 3
  - Data types: OGIS data types
  - Approximations for extended objects (e.g. linestrings, polygons)
    - Minimum Orthogonal Bounding Rectangle (MOBR or MBR)
    - MBR(O) is the smallest axis-parallel rectangle enclosing an object O
  - Supports filter and refine processing of queries
- Spatial operations
  - OGIS operations, e.g. topological, spatial analysis
  - Many topological operations are approximated by Overlap
  - Common spatial queries - listed in next slide

Common Spatial Queries and Operations

- Physical model provides simpler operations needed by spatial queries!
- Common Queries
  - Point query: Find all rectangles containing a given point
  - Range query: Find all objects within a query rectangle
  - Nearest neighbor: Find the point closest to a query point
  - Intersection query: Find all the rectangles intersecting a query rectangle (spatial join)



Common Spatial Queries and Operations

- Physical model provides simpler operations needed by spatial queries!
- Common Queries
  - Point query: Find all rectangles containing a given point
  - Range query: Find all objects within a query rectangle
  - Nearest neighbor: Find the point closest to a query point
  - Intersection query: Find all the rectangles intersecting a query rectangle
- Common operations across spatial queries
  - Find: retrieve records satisfying a condition on attribute(s)
  - FindNext: retrieve next record in a dataset with total order
    - After the last one retrieved via previous find or findnext
  - Nearest neighbor of a given object in a spatial dataset

5.1 物理数据模型小结

- Learn basic concepts in physical data model of SDBMS
- Review related concepts from physical DM of relational DBMS
- Reusing relational physical data model concepts
  - Space filling curves define a total order for points
  - This total order helps in using ordered files, search trees
  - But may lead to computational inefficiency!
- New techniques
  - Spatial indices, e.g. grids, hiearchical collection of rectangles
  - Provide better computational performance

第五章 空间存储与索引

- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

Storage Hierarchy in Computers

- Computers have several components
  - Central Processing Unit (CPU)
  - Input, output devices, e.g. mouse, keyword, monitors, printers
  - Communication mechanisms, e.g. internal bus, network card, modem
  - Storage Hierarchy
- Types of storage Devices
  - Main memories - fast but content is lost when power is off
  - Secondary storage - slower, retains content without power
  - Tertiary storage - very slow, retains content, very large capacity

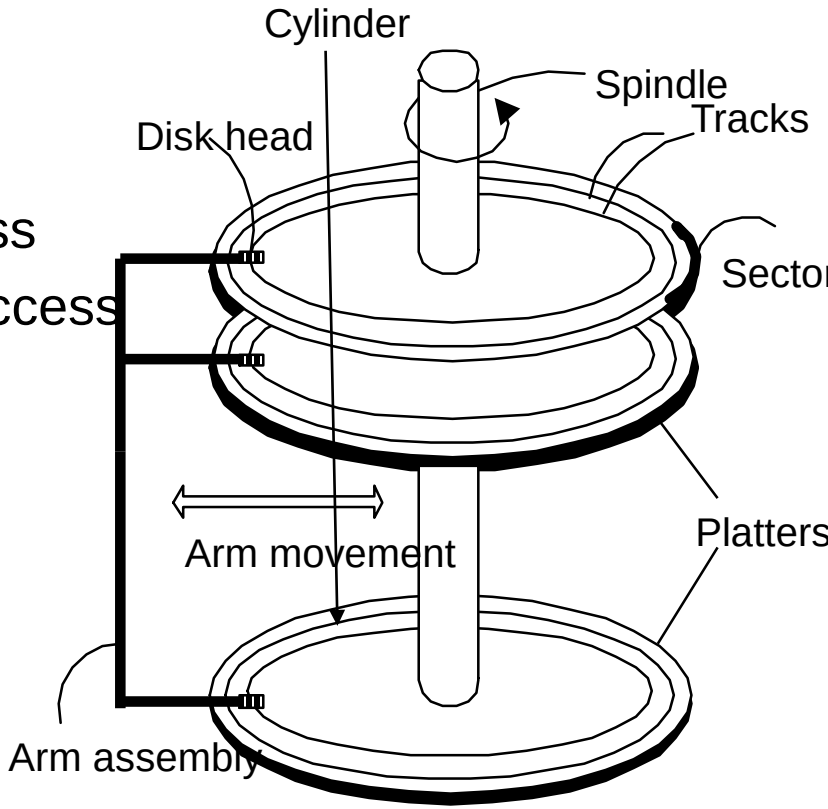
https://lagunita.stanford.edu/courses/Engineering/CS101/Summer2014/about

Storage Hierarchy in Computers

- Computers have several components
- Types of storage Devices
  - Main memories - fast but content is lost when power is off
  - Secondary storage - slower, retains content without power
  - Tertiary storage - very slow, retains content, very large capacity
- DBMS usually manage data
  - On secondary storage, e.g. disks, solid state disk
  - Use main memory to improve performance
  - User tertiary storage (e.g. tapes) for backup, archival etc.

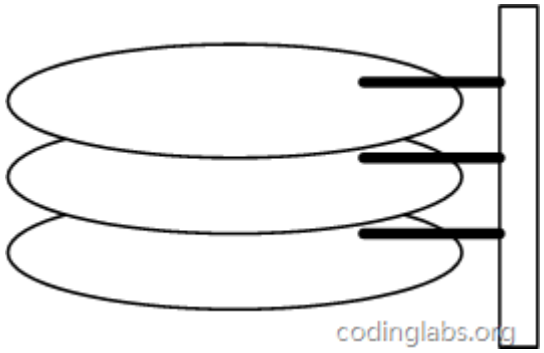
Secondary Storage Hardware: Disk Drives

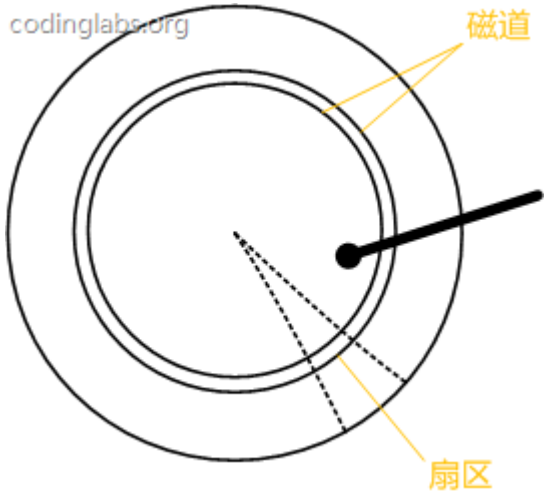
- Disk
  - Slow: Sequential access (although fast sequential reads)
  - Durable: Once on disk, data is safe
  - Cheap
- Memory
  - Fast
    - ~10x faster for sequential access
    - ~100, 000x faster for random access
  - Volatile: Data can be lost
  - Expensive



Secondary Storage Hardware: Disk Drives

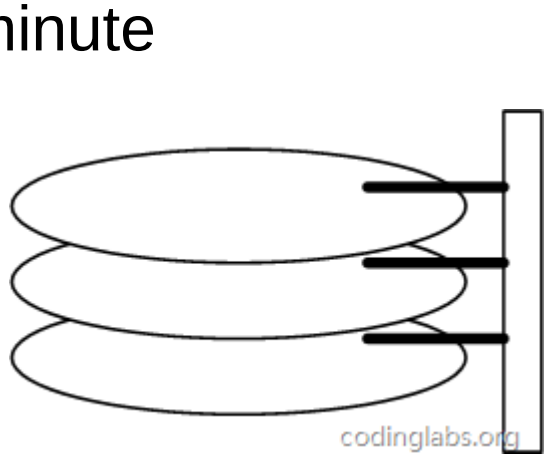
- Disk concepts
  - Circular platters with magnetic storage medium
  - Multiple platters are mounted on a spindle
  - Platters are divided into concentric tracks
  - A cylinder is a collection of tracks across platters with common radium
  - Tracks are divided into sectors
  - A sector size may a few kilo-Bytes

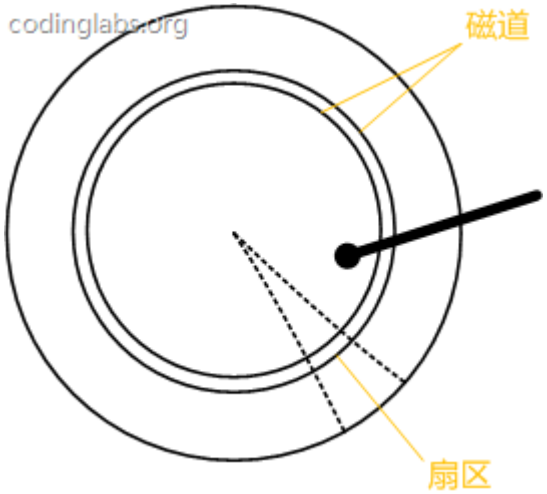




Secondary Storage Hardware: Disk Drives

- Disk concepts
- Disk drive concepts
  - Disk heads to read and write
  - There is disk head for each platter (recording surface)
  - A head assembly moves all the heads together in radial direction
  - Spindle rotates at a high speed, e.g. thousands revolution per minute





Secondary Storage Hardware: Disk Drives

- Disk concepts
- Disk drive concepts
- Accessing a sector has three major steps
  - Seek: Move head assembly to relevant track
  - Latency: Wait for spindle to rotate relevant sector under disk head
  - Transfer: Read or write the sector
  - Other steps involve communication between disk controller and CPU

Using Disk Hardware Efficiently

- Disk access cost are affected by
  - Placement of data on the disk
  - Fact: seek cost > latency cost > transfer
  - A few common observations follow
- Size of sectors

区分文件逻辑存储结构和物理存储结构

  - Larger sector provide faster transfer of large data sets
  - But waste storage space inside sectors for small data sets
- Placement of most frequently accessed data items
  - On middle tracks rather than innermost or outermost tracks
  - Reason: minimize average seek time

Using Disk Hardware Efficiently

- Disk access cost are affected by
- Size of sectors
- Placement of most frequently accessed data items
  - On middle tracks rather than innermost or outermost tracks
  - Reason: minimize average seek time
- Placement of items in a large data set requiring many sectors
  - Choose sectors from a single cylinder
  - Reason: Minimize seek cost in scanning the entire data set

Using Disk Hardware Efficiently

- Sequential scan is much faster than random reads
  - Good: read blocks 1, 2, 3, 4, 5, …
  - Bad: read blocks 2342, 11, 321, 9, …
- Rule of thumb
  - Random reading 1-2% of the file ≈ sequential scanning the entire file; this is decreasing over time (because of increased density of disks)

Software View of Disks: Fields, Records and File

- Views of secondary storage (e.g. disks)
  - Hardware views - Discussed in last few slides
  - Software views - Data on disks is organized into fields, records, files
- Concepts
  - Field presents a property or attribute of a relation or an entity
  - Records represent a row in a relational table
    - Collection of fields for attributes in relational schema of the table
  - Files are collections of records
    - Homogeneous collection of records may represent a relation
    - Heterogeneous collections may be a union of related relations

Mapping Records and files to Disk

- Records
  - Often smaller than a sector
  - Many records in a sector
- Files with many records
  - Many sectors per file
- File system
  - Collection of files
  - Organized into directories
- Mapping tables to disk
  - City table takes 2 sectors
  - Others take 1 sector each

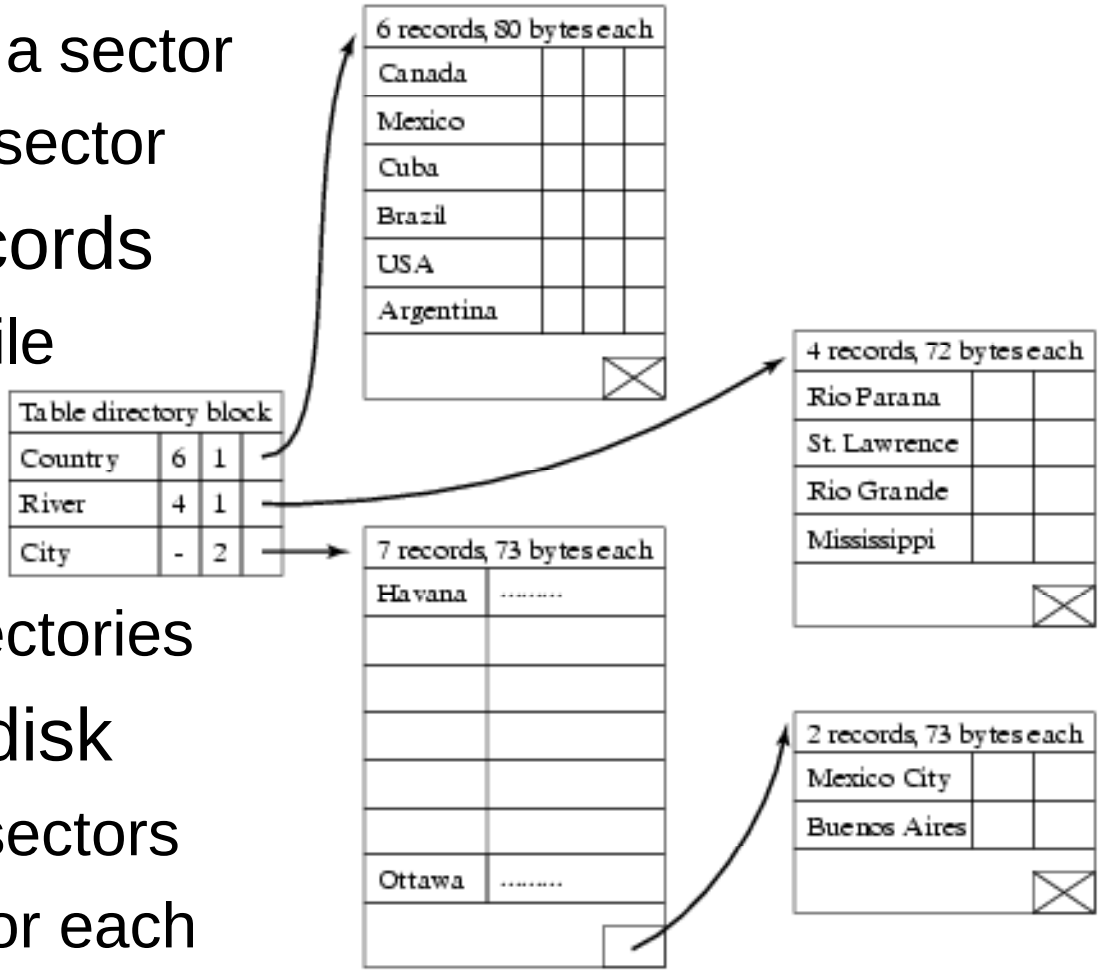


Figure 5.1

Buffer Management

- Motivation
  - Accessing a sector on disk is much slower than accessing main memory
  - Idea: Keep repeatedly accessed data in main memory buffers
    - To improve the completion time of queries
    - Reducing load on disk drive
- Buffer Manager software module decides
  - Which sectors stay in main memory buffers?
  - Which sector is moved out if we run out of memory buffer space?
  - When pre-fetch sector before access request from users?
  - These decision are based on the disk access patterns of queries

第五章 空间存储与索引

- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

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
  - Total cost = sum of I/O cost and CPU cost

Selected File Operations

- Common file operations
  - Find: key value → record matching key values
  - FindNext → Return next record after find if records were sorted
  - Insert → Add a new record to file without changing file-structure
  - Nearest neighbor of an object in a spatial dataset

Selected File Operations

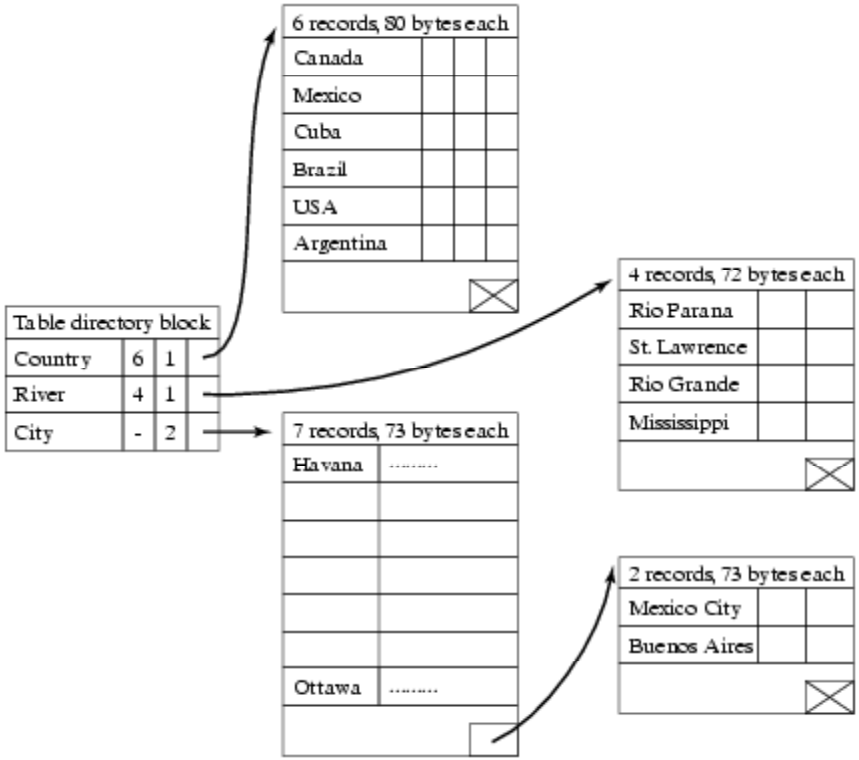
- Examples: Country, River, City
  - Find(Name = Canada) on Country table returns record about Canada
  - FindNext() on Country table returns record about Cuba
    - Since Cuba is next value after Canada in sorted order of Name
  - Insert(record about Panama) into Country table
    - Adds a new record
    - Location of record in Country file depends on file-structure
  - Nearest neighbor Argentina in country table is Brazil

Common File Structures

- Common file structures
  - Heap or unordered or unstructured
  - Ordered
    - Sorted according to some attribute(s) called key
  - Hashed
  - Clustered
- Basic Comparison of Common File Structures
  - Heap file is efficient for inserts and used for logfiles
    - But find, findnext, etc. are very slow
  - Ordered file oranization are very fast for findnext
    - And pretty competent for find, insert, etc
  - Hashed files are efficient for find, insert, delete etc.
    - But findnext is very slow

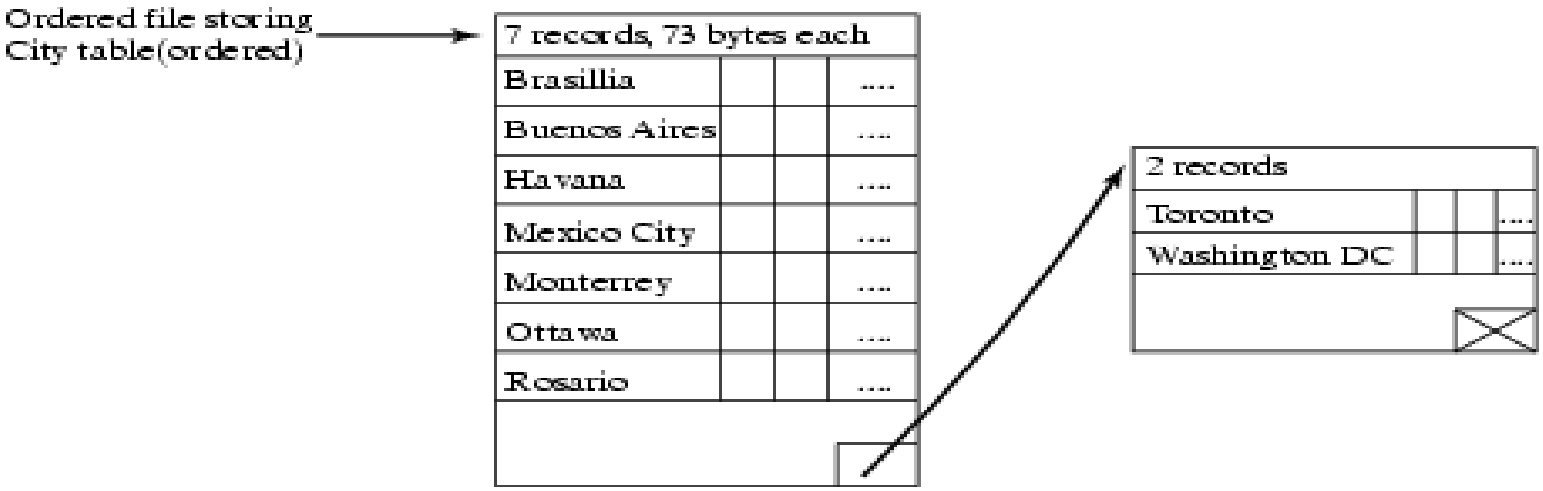
File Structures: Heap

- Heap
  - Records are in no particular order
  - Insert can simple add record to the last sector
  - Find, findnext, nearest neighbor scan the entire files



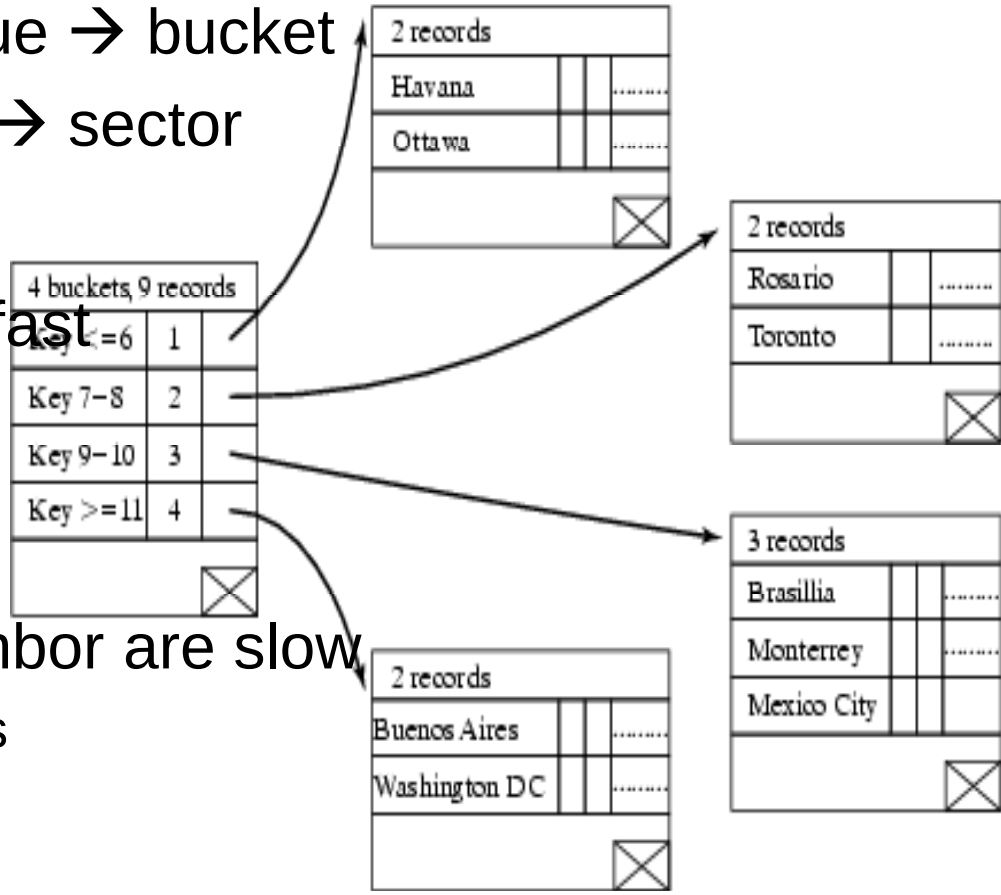
File Structures: Ordered

- Ordered
  - Records are sorted by a selected field
  - FindNext can simply pick up physically next record
  - Find, insert, delete may use binary search, is very efficient
  - Nearest neighbor processed as a range query



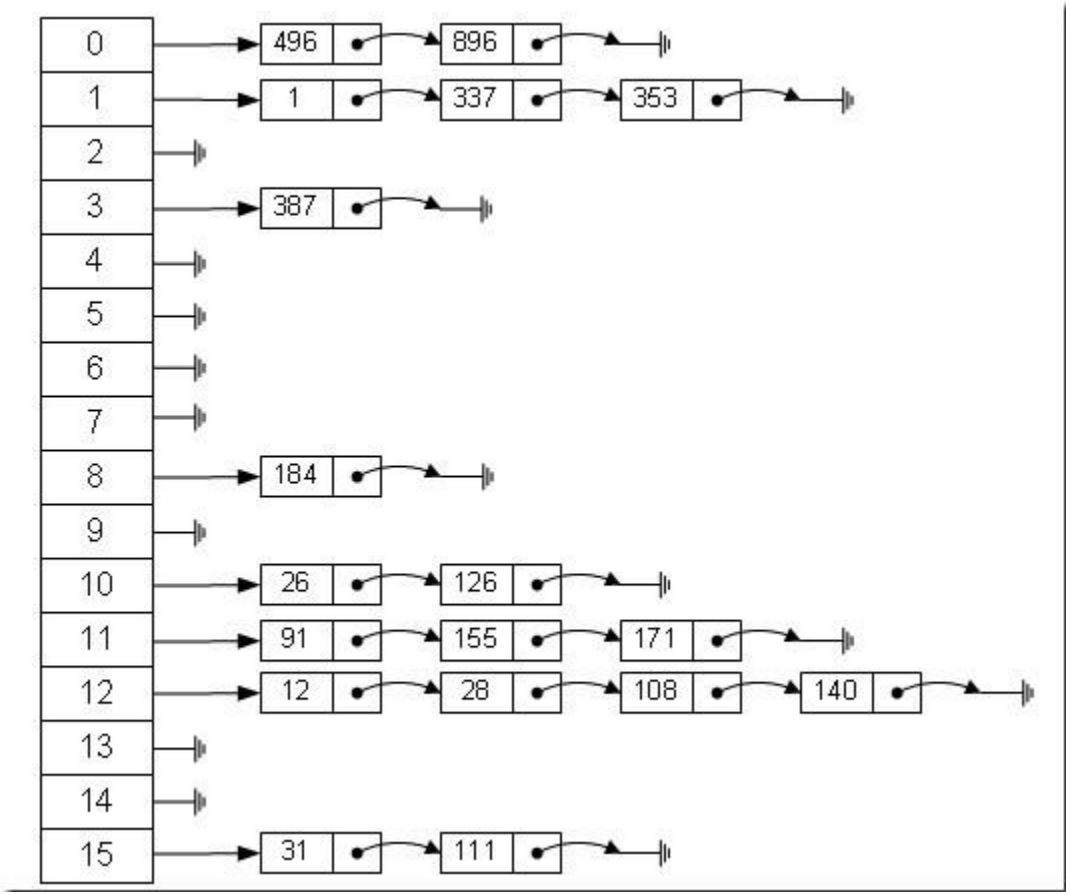
File Structure: Hash

- Components of a Hash file structure
  - A set of buckets (sectors)
  - Hash function : key value → bucket
  - Hash directory: bucket → sector
- Operations
  - Find, insert, delete are fast
    - Compute hash function
    - Lookup directiry
    - Fetch relevant sector
  - FindNext, nearest neighbor are slow
    - No order among records



File Structure: Hash

- 元素特征转变为数组下标的方法就是散列法
  - index = value % 16



Spatial File Structures: Clustering

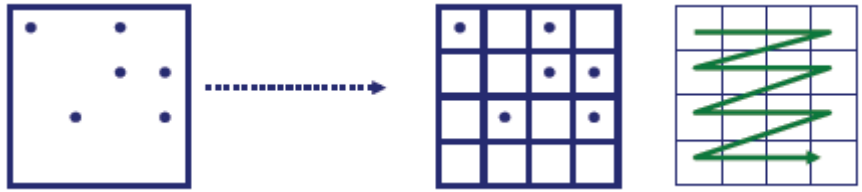
- Motivation
  - Ordered files are not natural for spatial data
  - Clustering records in a sector by space filling curve is an alternative
  - In general, clustering groups records
    - Accessed by common queries into common disk sectors
    - To reduce I/O costs for selected queries
- Clustering using space filling curves
  - Z-curve
  - Hilbert-curve

空间填充曲线

- 空间填充曲线(space-filling curve)
  - 降低空间维度的方法
  - 一条连续曲线，自身没有任何交叉
  - 通过访问所有单元格来填充包含均匀网格的四边形
    - Z曲线，Hilbert曲线
- 为了将数据空间循环分解到更小的子空间，我们需要引入m阶曲线
  - m阶曲线是基本曲线的每个网格被m-1阶曲线填充

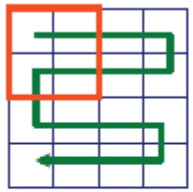
空间填充曲线

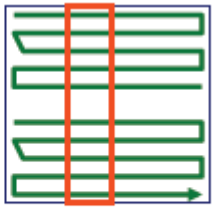
- Q: how would you organize, e.g., *n*-dim points, on disk? (*C* points per disk page)
  - Hint: reduce the problem to 1-d points (!!)
- Q1: why?
  - A1: B-trees!
- Q2: how?
  - A2: assume finite granularity (e.g., 2^{32}x2^{32}; 4x4 here)
- Q2.1: how to map n-d cells to 1-d cells?
  - A2.1: row-wise
  - Q: is it good?
  - A: great for ‘x’ axis; bad for ‘y’ axis



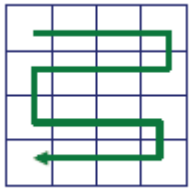
空间填充曲线

- Q: How about the ‘snake’ curve?
  - A: still problems:



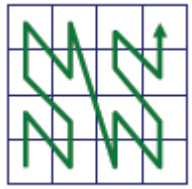


2^32

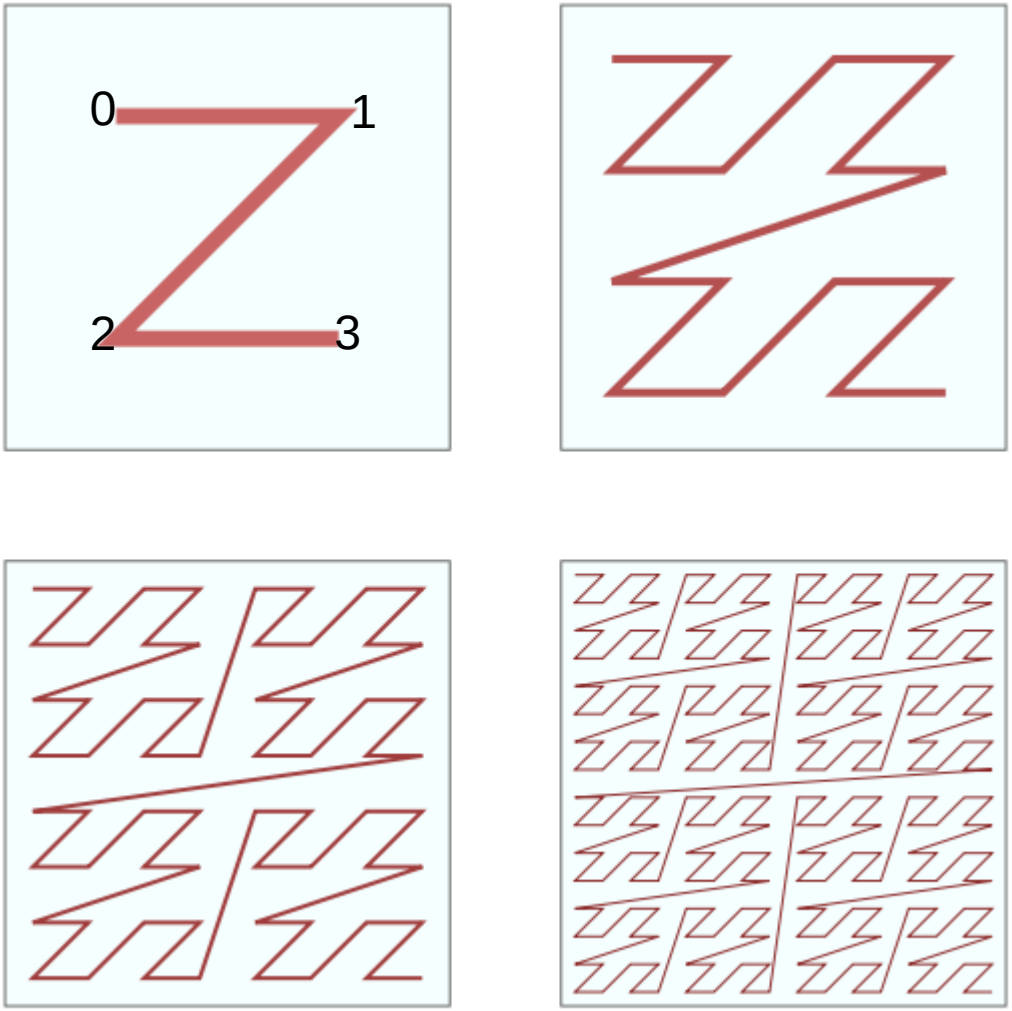


2^32

- Q: Why are those curves ‘bad’?
  - A: no distance preservation (~ clustering)
- Q: solution?
  - A: z-ordering/bit-shuffling/linear-quadtrees
  - ‘Looks’ better
    - Few long jumps
    - Scoops out the whole quadrant before leaving it
    - a.k.a. space filling curves



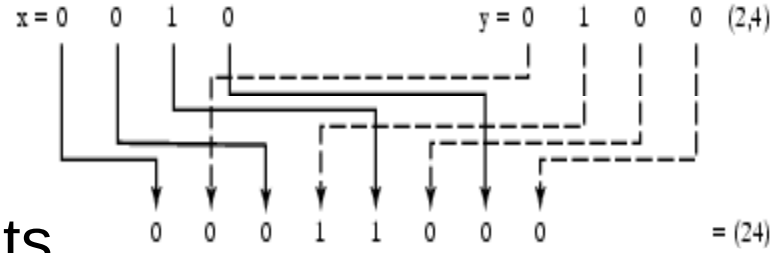
Z-Curve

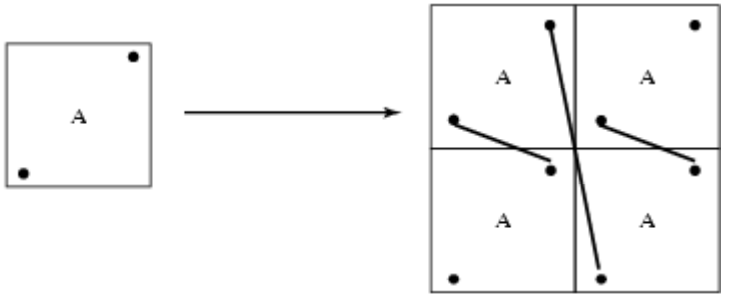


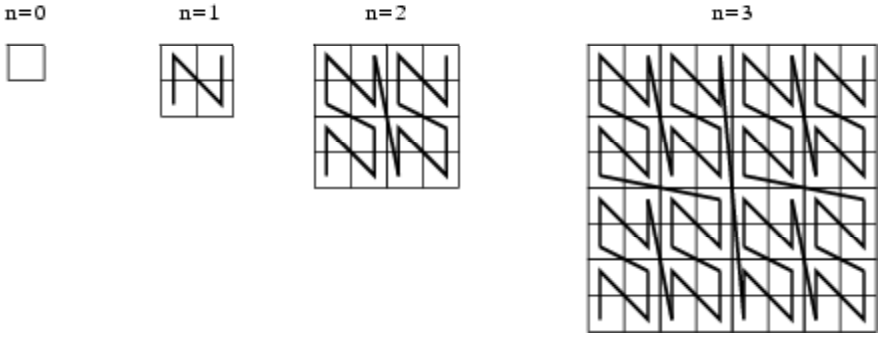
http://en.wikipedia.org/wiki/Z-order_curve

Z-Curve

- What is a Z-curve?
  - A space filling curve
  - Generated from interleaving bits
    - x, y coordinate
  - Alternative generation method
  - Connecting points by z-order
    - Looks like Ns or Zs
- Implementing file operations
  - Similar to ordered files

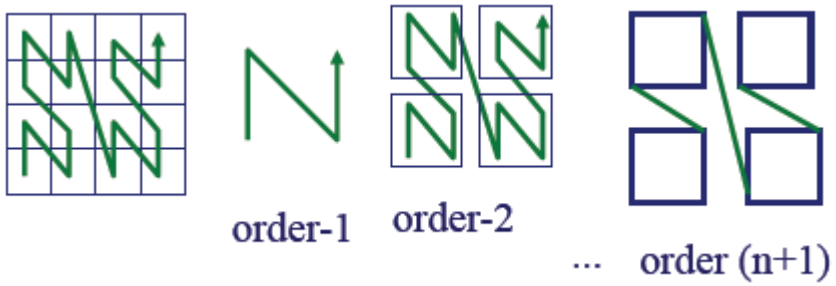




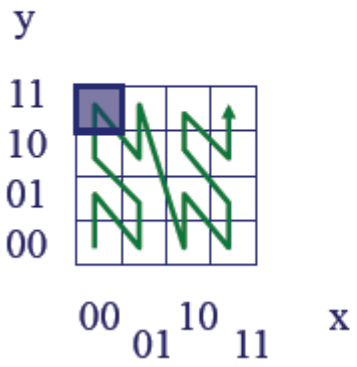


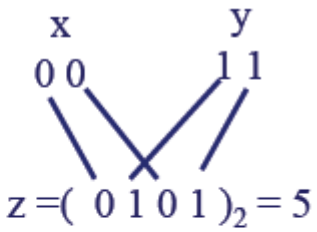
Z-Curve

- How to generate this curve ($z = f(x,y)$)?
- A1: ‘z’ (or ‘N’) shapes, recursively



- A2: bit-shuffling



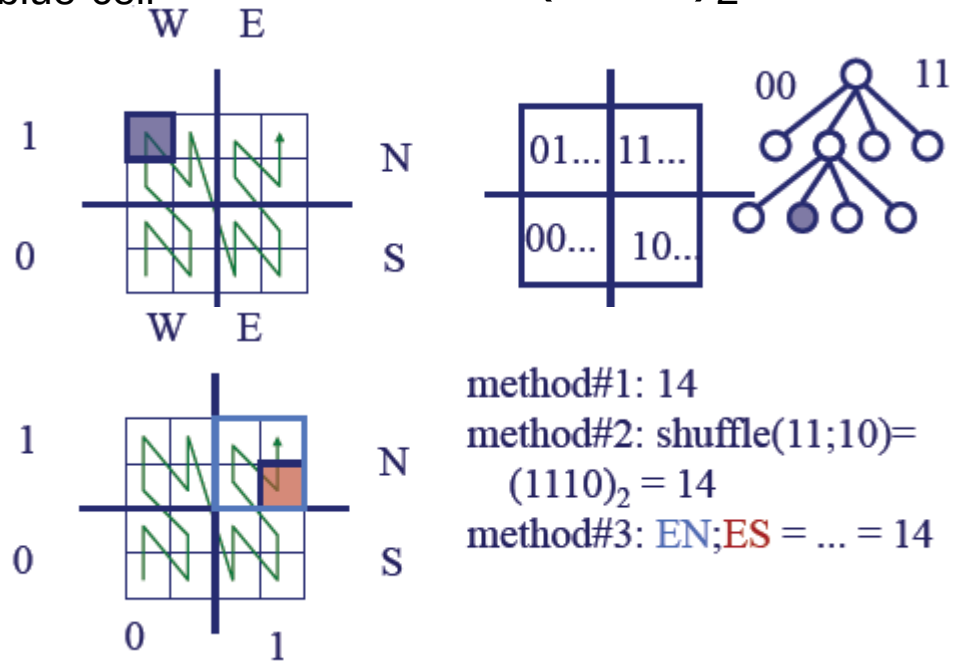


How about the reverse:

$(x,y) = g(z)$?

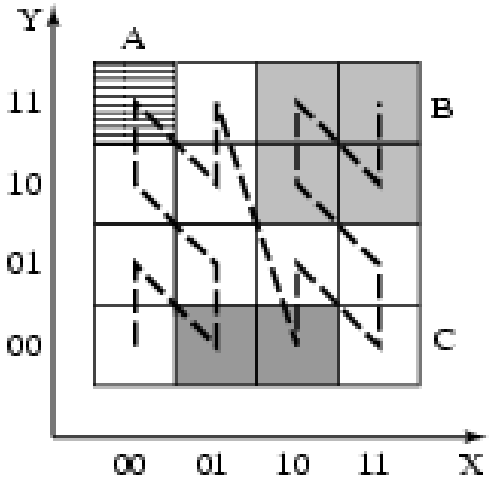
Z-Curve

- How to generate this curve ($z = f(x,y)$)?
- A1: ‘z’ (or ‘N’) shapes, RECURSIVELY
- A2: bit-shuffling
- A3: linear-quadtrees : assign N->1, S->0 e.t.c.
  - Eg.: $z_{\text{blue-cell}} = WN;WN = (0101)_2 = 5$

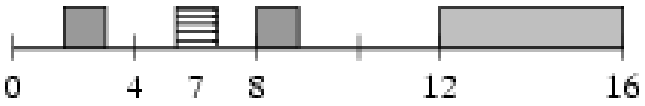


Z-Values

- Example
  - Left part shows a map with spatial object A, B, C
  - Right part and Left bottom part Z-values within A, B and C
  - Note C gets z-values of 2 and 8, which are not close
  - Exercise: Compute z-values for B

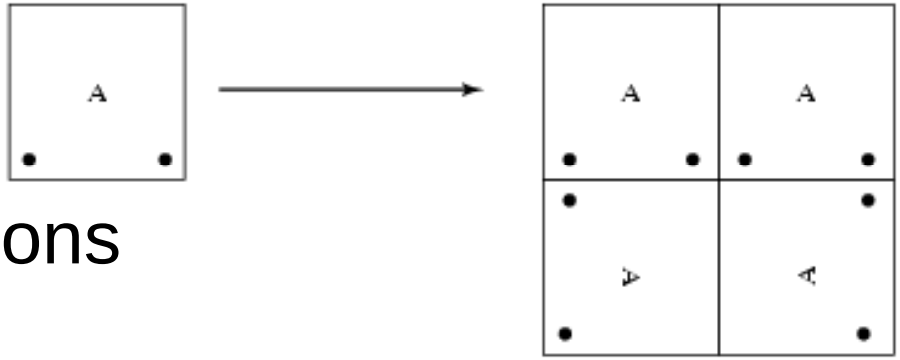


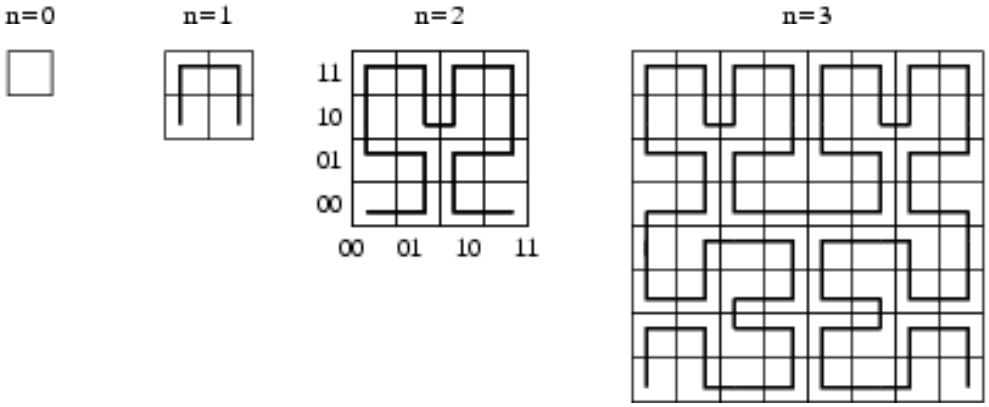
Object	Points	x	y	interleave	z-value
A	1	00	11	0101	5
B	1				
	2				
	3				
	4				
C	1	01	00	0010	2
	2	10	00	1000	8



Hilbert Curve

- A space filling curve
- More complex to generate
  - Due to rotations
  - Illustration on next slide
- Implementing file operations
  - Similar to ordered files





Calculating Hilbert Values

x \ y	00	01	10	11
00	0000	0010	1001	1010
01	0001	0011	1000	1011
10	0100	0110	1100	1110
11	0101	0111	1101	1111

(a)

x \ y	00	01	10	11
00	00	03	30	33
01	01	02	31	32
10	10	13	20	23
11	11	12	21	22

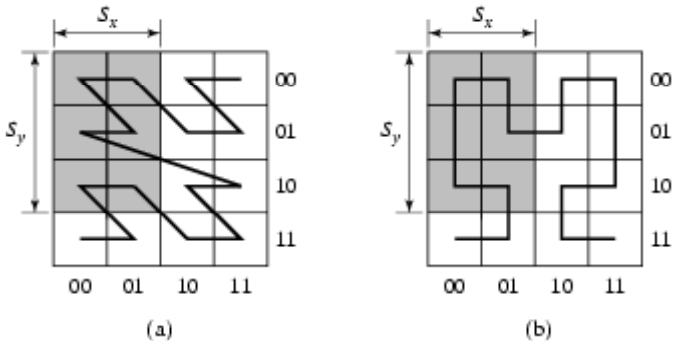
(b)

x \ y	00	01	10	11
00	00	01	32	33
01	03	02	31	30
10	10	13	20	23
11	11	12	21	22

(c)

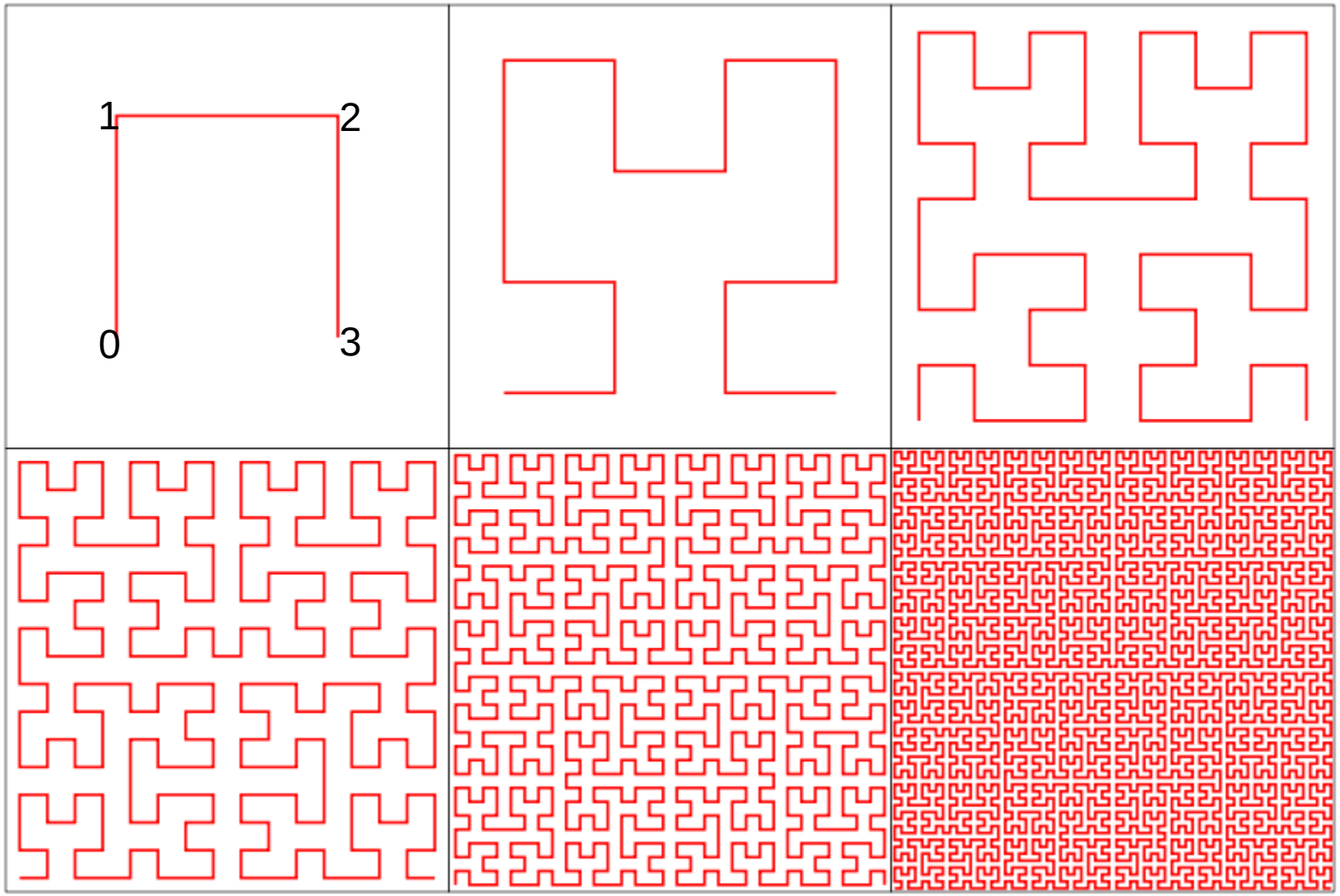
x \ y	00	01	10	11
00	0	1	14	15
01	3	2	13	12
10	4	7	8	11
11	5	6	9	10

(d)



Z-Curve to Hilbert Curve

Hilbert Curve



http://en.wikipedia.org/wiki/Hilbert_curve

Z-Curve vs. Hilbert Curve

- Z曲线和Hilbert曲线共同特点：
  - 填充曲线值临近的网格，其空间位置通常也相对临近
  - 任何一种空间排列都不能完全保证二维数据空间关系的维护（编号相邻，空间位置可能很远）
- 不同点：
  - Hilbert曲线的数据聚集特性更优，Z曲线的数据聚集特性较差
  - Hilbert曲线的映射过程较复杂，Z曲线的映射过程较简单

思考：哪类曲线更容易出现编号相邻，空间位置可能很远？

基于空间填充曲线的文件组织(补充)

- 基于空间填充曲线的有序文件也是提高空间查询效率的方法
- 数据库中有无序文件(unordered file)、散列文件(hashed file)和有序文件(ordered file)
- 无序文件：最简单的记录组织形式，也叫堆(heap)文件，其记录并没有特定的顺序
  - 优点：快速数据插入（追加在文件末尾）
  - 缺点：数据查找效率（可能需要遍历整个文件）

基于空间填充曲线的文件组织(补充)

- 有序文件：根据给定的主码域对记录进行组织，即按主码域的排列顺序存储记录
  - 可以使用折半查找算法根据给定主码属性值查找记录，也可以实现范围查找
    - 即用折半查找法查到第一个符合条件的记录，然后扫描后续记录，直到找到最后一个满足条件的记录为止
  - 优点：查找效率高
  - 缺点：在数据插入时，需要根据插入数据的主码属性值调整其后记录的存储位置

基于空间填充曲线的文件组织(补充)

- 散列文件：利用散列函数将事先选择的主码域的值映射到一个散列单元中，属于不同单元的记录存储在不同的磁盘页面中
  - 根据给出的主码属性值查找数据时，先通过散列函数算出所查数据所位于的单元，再访问相应的磁盘页面
  - 对点查询、插入和删除非常高效
  - 但不适合范围查询

基于空间填充曲线的文件组织(补充)

- 数据库中也常将这种决定数据文件中记录的存放位置的所以称为“主索引”（primary index），而将那些决定数据文件中记录的存放位置、额外存储的索引文件称为“辅助索引”（alternate index）
- 基于空间填充曲线的有序文件的实质是利用空间填充曲线在一定程度上保持编码的空间对象邻近性的特点，让编码邻近的两个空间对象的磁盘存储位置也邻近
- 由于数据库系统通常按页加载数据，在加载一个数据时与其存储在同一个页面中的空间对象也被加载到内存中

基于空间填充曲线的文件组织(补充)

- 空间位置邻近的两个几何对象被同时选中的概率也是比较大的，因此，当系统需要读邻近的数据时，就可以直接从内存中读到，减少磁盘的读写次数，提高了内存的命中率
- 利用空间邻近的对象物理存储也邻近的特征，使得空间邻近的对象被同时检索到的概率也高，从而提高内存数据的命中率、提高检索效率

5.3 数据文件组织小结

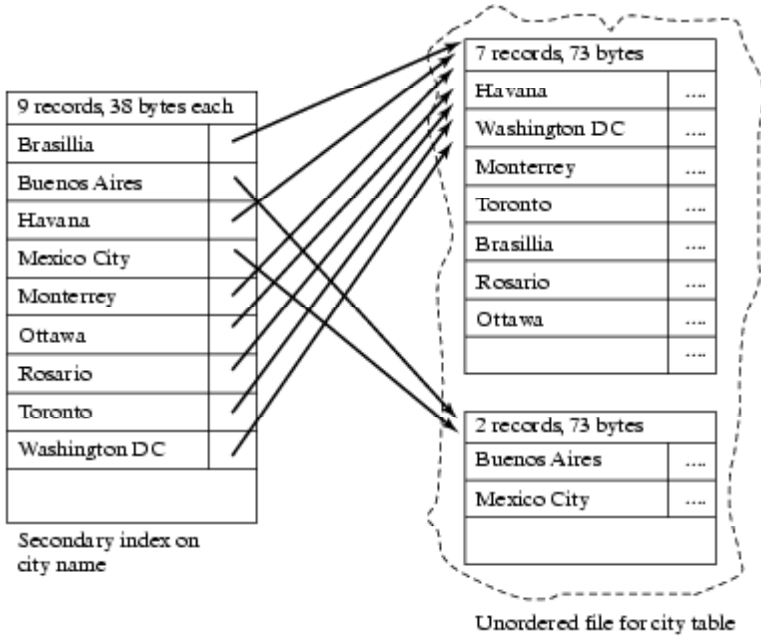
- Common file structures
  - Heap or unordered or unstructured
  - Ordered
  - Hashed
  - Clustered
- Clustering using space filling curves (n-D → 1D)
  - Z-curve
  - Hilbert-curve
- In general, clustering groups records
  - Accessed by common queries into common disk sectors
  - To reduce I/O costs for selected queries

第五章 空间存储与索引

- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

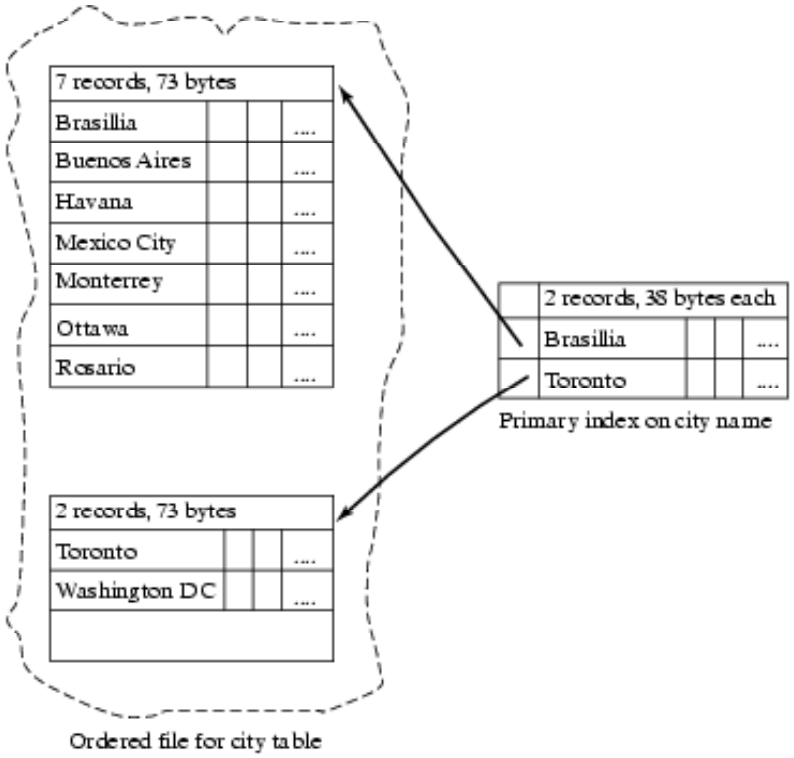
What is an Index?

- Concept of an index
  - Auxiliary file to search a data file
- Index records have
  - Key value
  - Address of relevant data sector
- Index records are ordered
  - find, findnext, insert are fast
- Note assumption of total order
  - On values of indexed attributes



Classifying Indexes

- Classification criteria
  - Data-file-structure
  - Key data type
  - Others
- Secondary index
  - Heap data file
  - 1 index record per data record
- Primary index
  - Data file ordered by indexed attribute
  - 1 index record per data sector
- Question: A table can have at most one primary index. Why?



Attribute Data Types and Indices

- Index file structure depends on data type of indexed attribute
  - Attributes with total order
    - Example, numbers, points ordered by space filling curves
    - B-tree is a popular index organization
  - Spatial objects (e.g. polygons)
    - Spatial organization are more efficient
    - Hundreds of organizations are proposed in literature
    - Two main families are Grid Files and R-trees

How DBMS Answer This Query?

- S – 学生关系，SC – 学生选课关系
- Select *

  From S, SC

  Where S.Sno = SC.Sno And SC.Cno > 300
- Nested-Loop Join
- For sc in SC
  - If sc.Cno > 300 then
    - For s in S
      - If s.Sno = sc.Sno
        - » Output *

思考：DBMS如何使用索引？

How DBMS Answer This Query?

- Select *

  From S, SC

  Where S.Sno = SC.Sno And SC.Cno > 300
- Merge Join
- Sort S on Sno
- Sort SC on Sno (and filter on Cno > 300)
- Merge join S, SC on S.Sno = SC.Sno
- For (s, sc) in merged_result output *

How DBMS Answer This Query?

- Select *

  From S, SC

  Where S.Sno = SC.Sno And SC.Cno > 300
- Hash-Join
- Create a hash-table
- For s in S
  - Insert s in the hash-table on s.Sno
- For sc in SC
  - If Cno > 300
    - Then probe sc.Sno in hash-table
      - If match found
      - Then output *

PostgreSQL Example

- R(A int), S(B int)
- explain select * from R, S where A = B order by A;

输出窗口

数据输出 | 解释 | 消息 | 历史

	QUERY PLAN text
1	Merge Join (cost=22.51..27.57 rows=404 width=8)
2	Merge Cond: (r.a = s.b)
3	-> Sort (cost=10.75..11.26 rows=202 width=4)
4	Sort Key: r.a
5	-> Seq Scan on r (cost=0.00..3.02 rows=202 width=4)
6	-> Sort (cost=11.75..12.26 rows=202 width=4)
7	Sort Key: s.b
8	-> Seq Scan on s (cost=0.00..4.02 rows=202 width=4)

思考：为什么最后没有排序？

输出窗口

数据输出 | 解释 | 消息 | 历史

	QUERY PLAN text
1	Sort (cost=10.03..10.29 rows=101 width=8)
2	Sort Key: r.a
3	-> Hash Join (cost=3.27..6.67 rows=101 width=8)
4	Hash Cond: (r.a = s.b)
5	-> Seq Scan on r (cost=0.00..2.01 rows=101 width=4)
6	-> Hash (cost=2.01..2.01 rows=101 width=4)
7	-> Seq Scan on s (cost=0.00..2.01 rows=101 width=4)

思考：cost的作用？

思考：插入什么数据，获得上述两种不同结果？

How DBMS Answer This Query?

- Which plan is best?
- Nested loop join
  - $O(N^2)$
  - Could be O(N) when few courses > 300
- Merge join
  - O(NlogN)
  - Could be O(N) if tables already sorted
- Hash join
  - O(N)

SparkSQL - 有必要坐下来聊聊Join
http://mp.weixin.qq.com/s/z427L-lCb34KaGJCLq6wbQ

表连接操作

- Nested Loop
  - 对于被连接的数据子集较小的情况，Nested Loop是个较好的选择
  - 若Join字段上没有索引查询优化器一般就不会选择Nested Loop
  - 扫描一个表（外表），每读到一条记录，就根据Join字段上的索引去另一张表（内表）里面查找
  - 内表（一般是带索引的大表）被外表（也叫“驱动表”，一般为小表——不仅相对其它表为小表，而且记录数的绝对值也较小，不要求有索引）驱动
  - 外表返回的每一行都要在内表中检索找到与它匹配的行，因此整个查询返回的结果集不能太大（大于1 万不适合）

http://www.jasongj.com/2015/03/07/Join1/

表连接操作

- Hash Join
  - 做大数据集连接时的常用方式，但只能应用于等值连接
  - 使用两个表中较小（相对较小）的表利用Join Key在内存中建立散列表，然后扫描较大的表并探测散列表，找出与Hash表匹配的行
  - 适用于较小的表完全可以放于内存中的情况，这样总成本就是访问两个表的成本之和
  - 在表很大的情况下并不能完全放入内存，这时优化器会将它分割成若干不同的分区，不能放入内存的部分就把该分区写入磁盘的临时段，此时要求有较大的临时段从而尽量提高I/O 的性能
  - 能够很好的工作于没有索引的大表和并行查询的环境中，并提供最好的性能

http://www.jasongj.com/2015/03/07/Join1/

表连接操作

- Merge Join
  - 通常情况下Hash Join的效果都比排序合并连接要好，然而如果两表已经被排过序，在执行排序合并连接时不需要再排序了，这时Merge Join的性能会优于Hash Join
  - Merge join的操作通常分三步：
    - 对连接的每个表做table access full
    - 对table access full的结果进行排序
    - 进行merge join对排序结果进行合并
  - 在全表扫描比索引范围扫描再进行表访问更可取的情况下，Merge Join会比Nested Loop性能更佳
    - 当表特别小或特别巨大的时候，实行全表访问可能会比索引范围扫描更有效
  - Merge Join可适于于非等值Join（>，<，>=，<=，但是不包含!=，也即<>）

http://www.jasongj.com/2015/03/07/Join1/

表连接操作

类别	Nested Loop	Hash Join	Merge Join
使用条件	任何条件	等值连接（=）	等值或非等值连接(>，<，=，>=，<=)，‘<>’除外
相关资源	CPU、磁盘I/O	内存、临时空间	内存、临时空间
特点	当有高选择性索引或进行限制性搜索时效率比较高，能够快速返回第一次的搜索结果	当缺乏索引或者索引条件模糊时，Hash Join比Nested Loop有效。通常比Merge Join快。在数据仓库环境下，如果表的纪录数多，效率高	当缺乏索引或者索引条件模糊时，Merge Join比Nested Loop有效。非等值连接时，Merge Join比Hash Join更有效
缺点	当索引丢失或者查询条件限制不够时，效率很低；当表的纪录数多时，效率低	为建立哈希表，需要大量内存。第一次的结果返回较慢	当缺乏索引或者索引条件模糊时，Merge Join比Nested Loop有效。非等值连接时，Merge Join比Hash Join更有效

http://www.jasongj.com/2015/03/07/Join1/

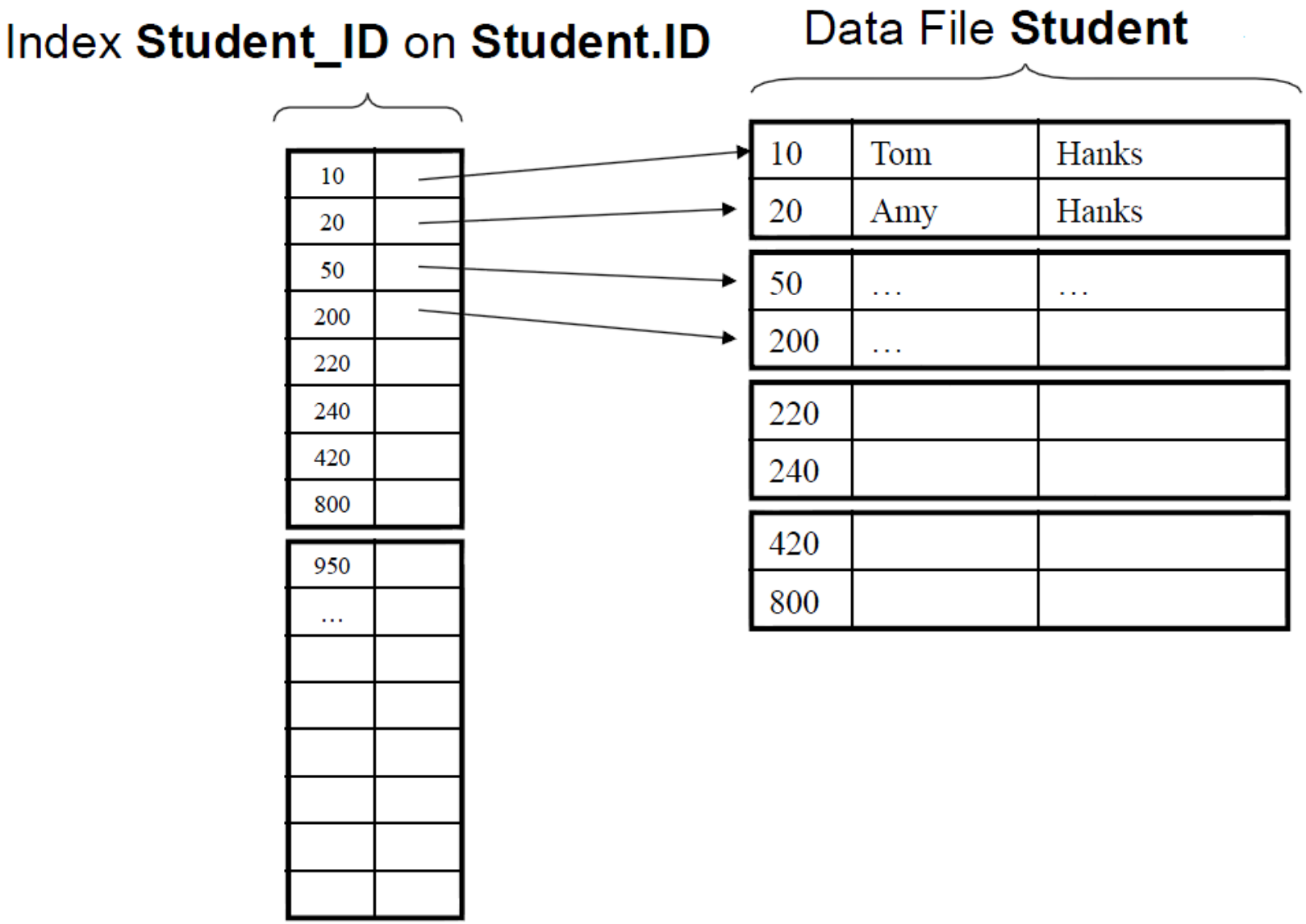
Data File vs. Index File

- Data file types
  - Head file (unsorted)
  - Sequential file
    - Sorted according to some attribute(s) called key
- An additional file
  - Allows fast access the records in the data file given a search key
  - The index contains (key, value) pairs
    - The key = an attribute value (e.g. Sno or name)
    - The value = a pointer to the record
  - Could have many indexes for on table

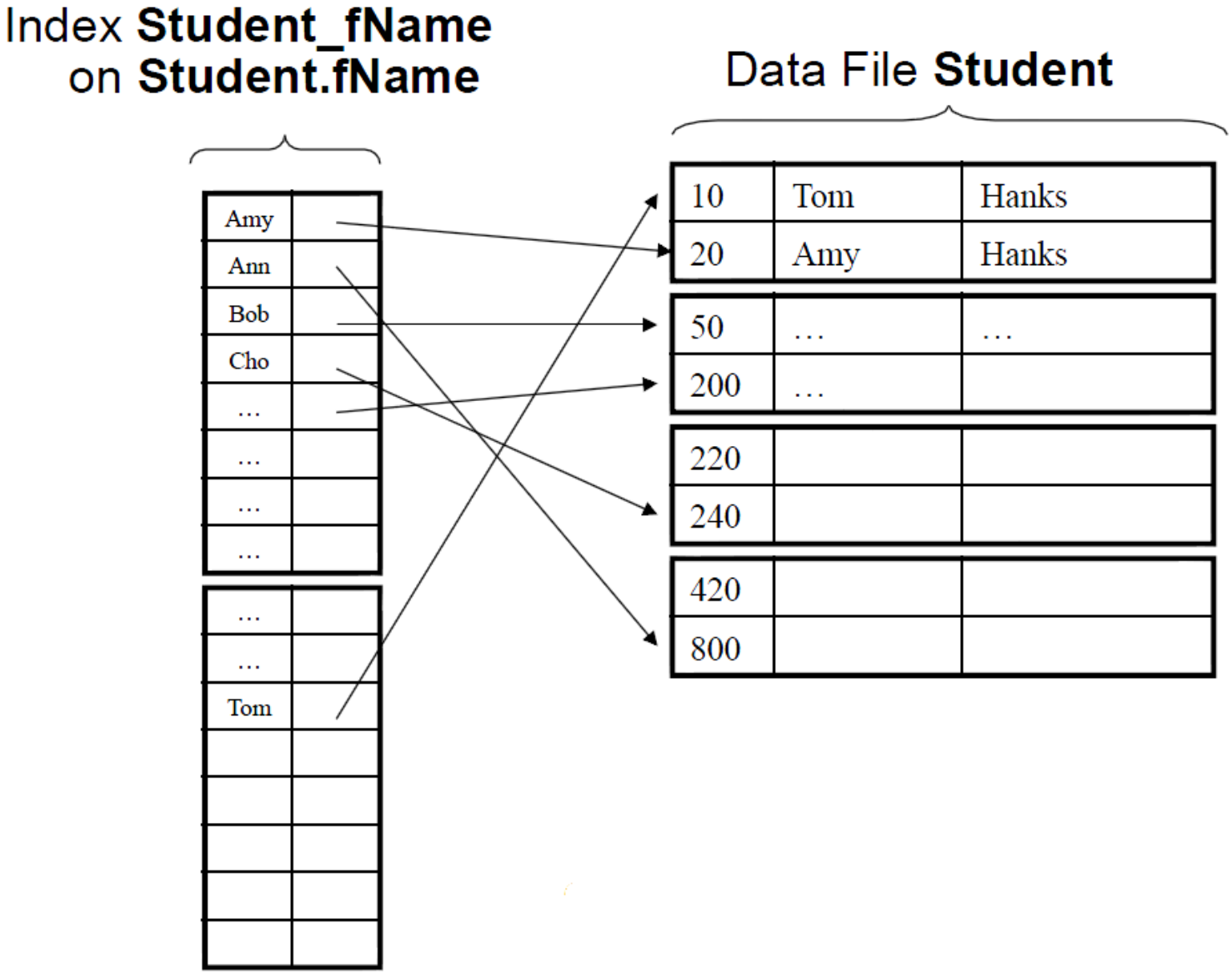
Different Keys

- Different keys
  - Primary key – uniquely identifies a tuple
    - 有些DBMS为关系的单属性primary key自动创建索引
  - Key of the sequential file – how the data file sorted, if at all
  - Index key – how the index is organized

Examples


Index **Student_ID** on **Student.ID**
Data File **Student**
10
20
50
200
220
240
420
800
950
…
10 Tom Hanks
20 Amy Hanks
50 … …
200 …
220
240
420
800

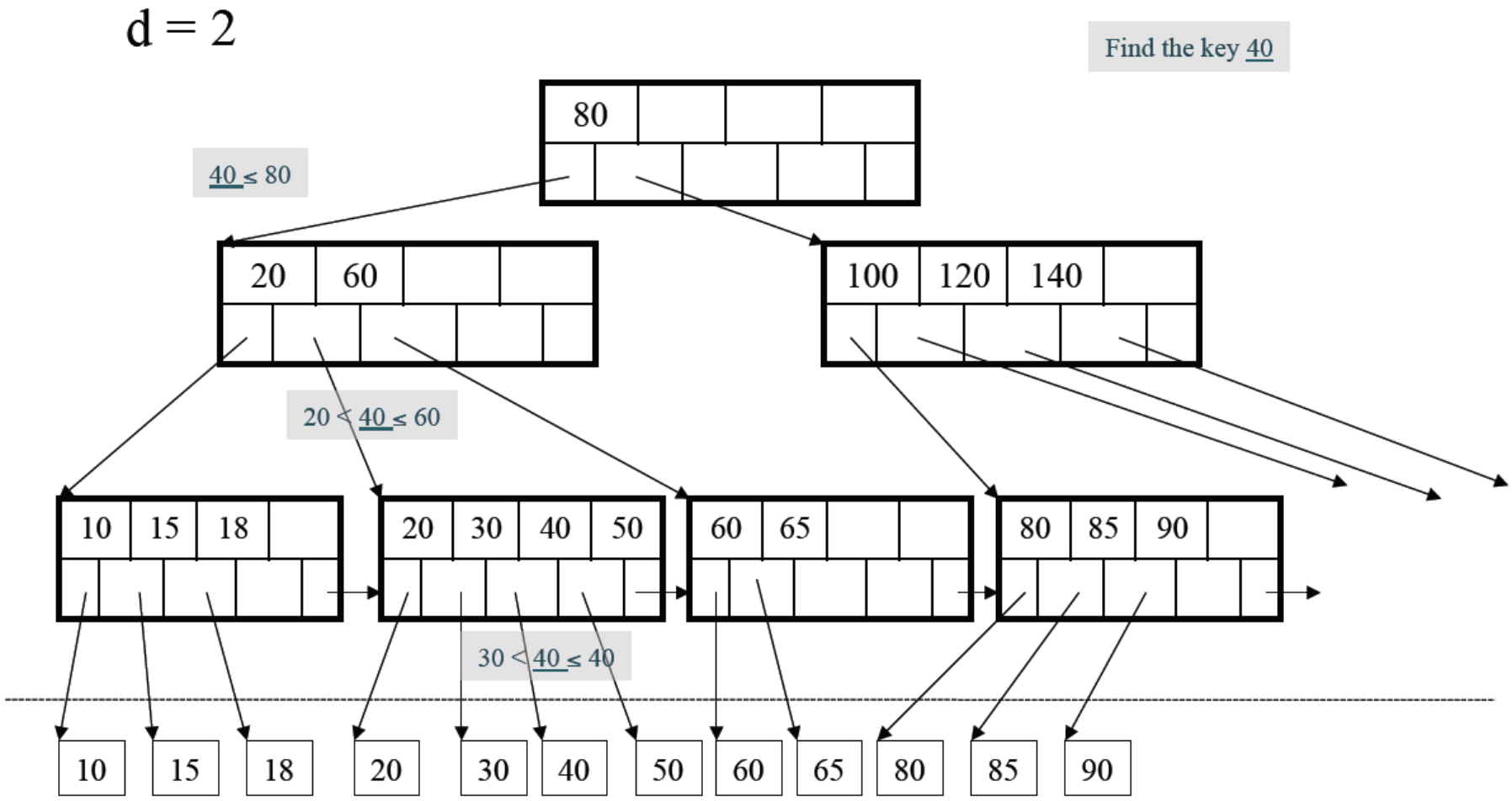
Examples


Index **Student_fName**
on **Student.fName**
Data File **Student**
Amy
Ann
Bob
Cho
…
…
…
…
…
…
Tom
10
Tom
Hanks
20
Amy
Hanks
50
…
…
200
…
220
240
420
800

索引的组织

- 索引的组织(属性索引)
  - Hash table
  - B+ trees: most popular
    - They are search trees, but they are not binary instead have higher fanout
  - Specialized indexes: bit maps, R-trees, inverted index

B+ Tree



B+ Tree Visualization: http://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Index in PostgreSQL

- R(A int), S(B int)
- create index iR on R(A);
- pgAdmin

explain 查询规划
explain analyze 查询规划 + 实际查询时间 (不返回结果)

http://www.postgresql.org/docs/current/static/sql-createindex.html

  - -- Index: public.ir
  - -- DROP INDEX public.ir;
  - CREATE INDEX ir
    ON public.r
    USING btree (a);

SQL 窗口

```
-- Index: public.ir

-- DROP INDEX public.ir;

CREATE INDEX ir
  ON public.r
  USING btree
  (a);
```

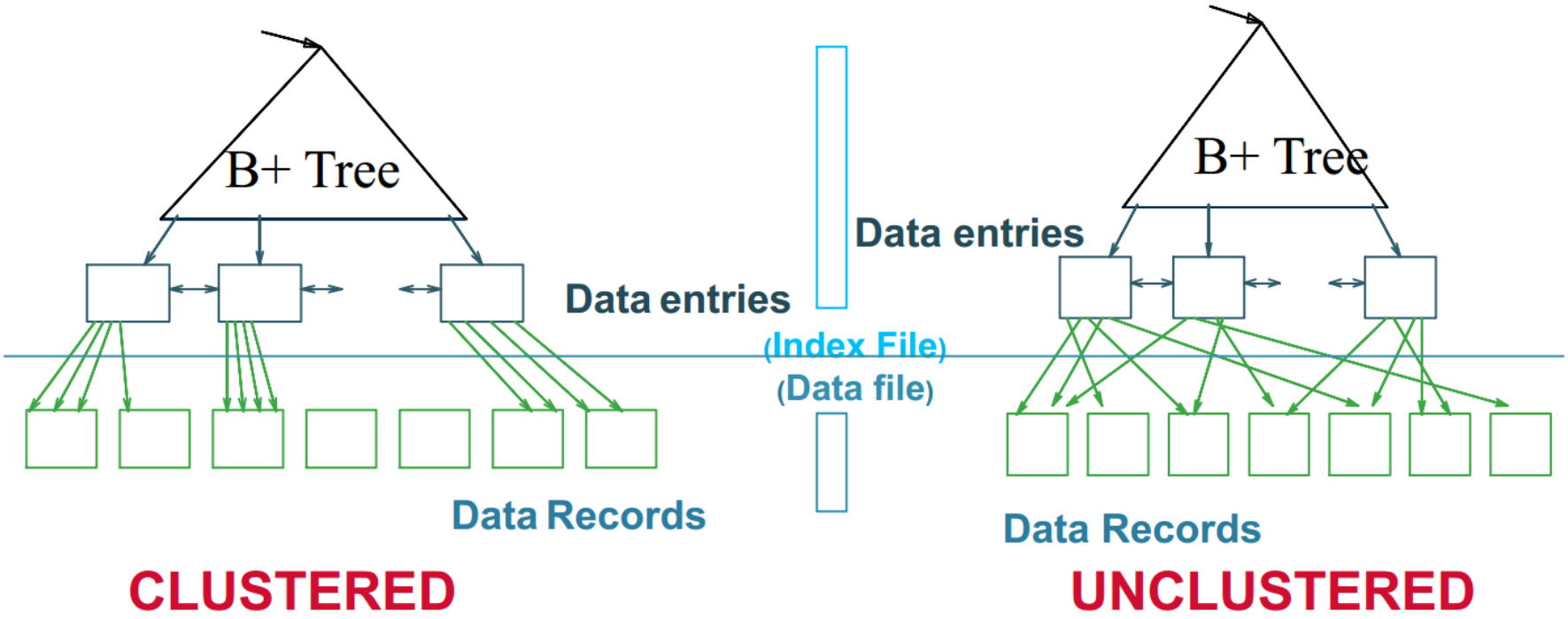
- explain select * from R, S where A = B order by A;

输出窗口

数据输出 | 解释 | 消息 | 历史

	QUERY PLAN text
1	Merge Join (cost=11.90..35.02 rows=404 width=8)
2	Merge Cond: (r.a = s.b)
3	-> Index Only Scan using ir on r (cost=0.15..18.21 rows=404 width=4)
4	-> Sort (cost=11.75..12.26 rows=202 width=4)
5	Sort Key: s.b
6	-> Seq Scan on s (cost=0.00..4.02 rows=202 width=4)

聚集与非聚集索引



Every table can have **only one** clustered and **many** unclustered indexes

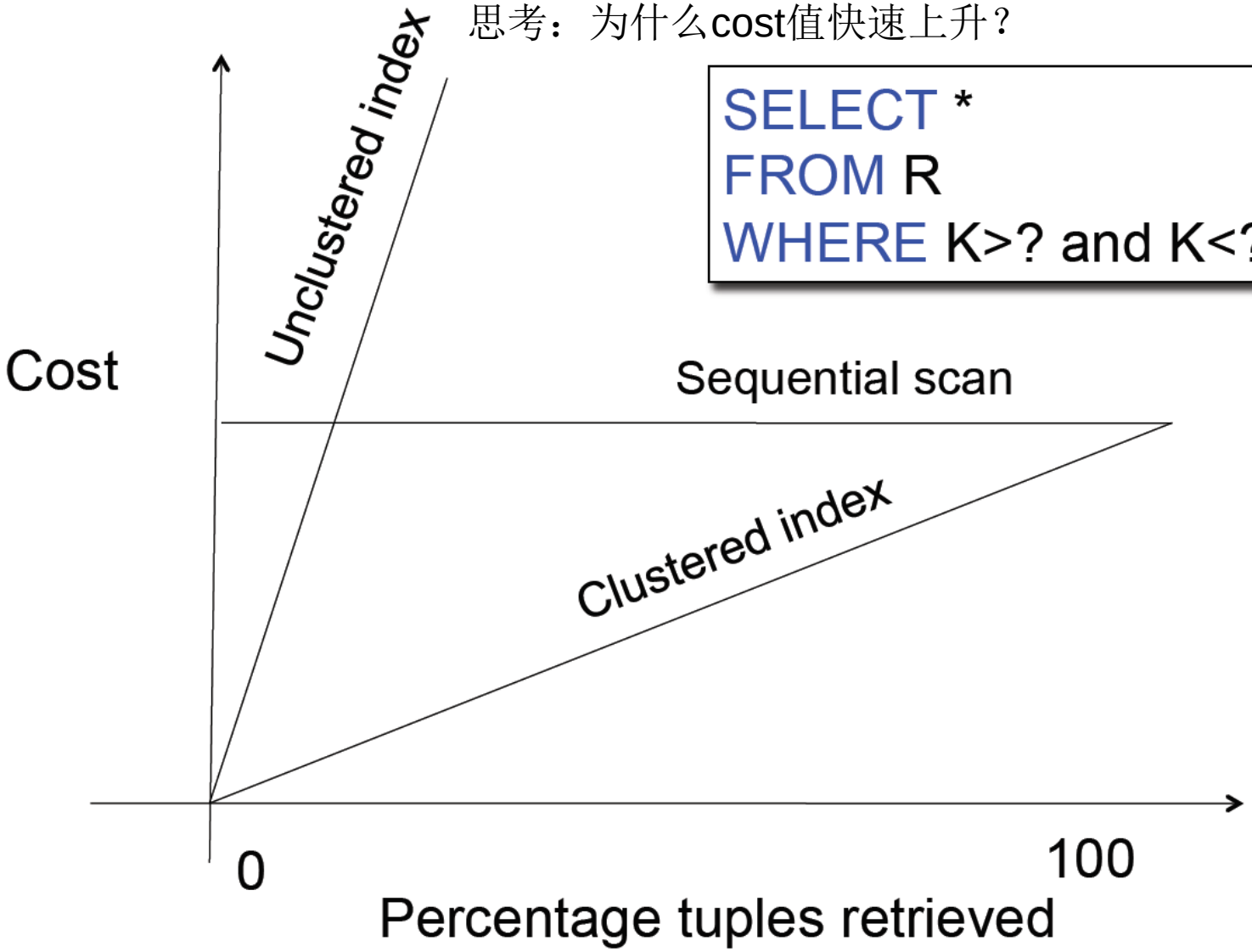
聚集与非聚集索引

- 聚集与非聚集
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data
  - Range queries benefit mostly from clustered index

聚集与非聚集索引

思考：为什么cost值快速上升？

```
SELECT *
FROM R
WHERE K>? and K<?
```



Primary/Secondary

- Primary/Secondary
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2
    - means the same as clustered /unclustered
- Organization B+ tree or Hash table

创建索引

思考：SQL Server和PostgreSQL都可以建unique和clustered索引？

- 创建索引：CREATE [UNIQUE] [CLUSTERED] INDEX <索引名> ON <表名>(<列名>[<次序>][,<列名>[<次序>]]…);
  - 用<表名>指定要建索引的基本表名字
  - 索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
  - 用<次序>指定索引值的排列次序，升序：ASC，降序：DESC。缺省值：ASC
  - UNIQUE表明此索引的每一个索引值只对应唯一的数据记录
  - CLUSTERED表示要建立的索引是聚簇索引
  - 聚簇索引的顺序就是数据的物理存储顺序，而非聚簇索引的顺序与数据物理排列顺序无关

创建索引

- 为学生-课程数据库中的S，C，SC三个表建立索引。其中S表按学号升序建唯一索引，C表按课程号升序建唯一索引，SC表按学号升序和课程号降序建唯一索引
  - CREATE UNIQUE INDEX Stusno ON S (Sno ASC);
  - CREATE UNIQUE INDEX Coucno ON C (Cno ASC);
  - CREATE UNIQUE INDEX SCno ON SC(Sno ASC, Cno DESC);
- 删除索引：DROP INDEX <索引名>;
  - 删除索引时，系统会从数据字典中删去有关该索引的描述
  - 例删除S表的Stusno索引：DROP INDEX Stusno;

思考：一个表可以建立几个聚集索引？

SDBM索引使用

```
Select *
From S, SC
Where S.Sno = SC.Sno and SC.Cno > 300
```

- Assume the database has indexes on
  - Index_SC_Cno = index of SC.Cno
  - Index_Student_Sno = index on Student.Sno

```
for y in index_SC_Cno where y.Cno > 300 (index selection)
    for x in Student where x.Sno = y.Sno (index join)
        output *
```

选择什么属性建索引？

- Index selection problem
  - Given a table, and a “workload”, decide which indexes to create
- Who does index selection
  - The database administrator DBA
  - Semi-automatically, using a database administration tool
- Make some attribute K a search key if the WHERE clause contains
  - An exact match on K
  - A range predicate on K
  - A join on K

选择什么属性建索引？

- Basic index selection guidelines
  - Consider queries in workload in order of importance
  - Consider relations accessed by query
    - No point indexing other relations
  - Look at WHERE clause for possible search key
  - Try to choose indexes that speed-up multiple queries
- 创建了属性索引，但DBMS在具体查询时不一定使用
  - 为什么？
  - DBMS如何判断是否应该使用索引？

选择什么属性建索引？

- Index selection: Multi-attribute keys
- Consider creating a multi-attribute key on K1, K2, …, if
  - WHERE clause has matches on K1, K2, …
    - But also consider separate indexes
  - SELECT clause contains only K1, K2, …
    - A covering index is one that can be used exclusively to answer a query, e.g. index R(K1, K2) cover the query: SELECT K2 FROM R WHERE K1 = 55

思考：R索引能加速SELECT K2 FROM R WHERE K2 = 55？

5.4 属性索引小结

- 索引的组织(属性索引)
  - Hash table
  - B+ trees: most popular
  - Specialized indexes: bit maps, R-trees, inverted index
  - 区分clustered vs. non-clustered, primary vs. secondary
- Query plan: Nested loop join / Merge join / Hash join
- 数据库何时使用索引
  - An exact match on K
  - A range predicate on K
  - A join on K
- 选择什么属性创建索引？
  - 考虑因素有哪些？

第五章 空间存储与索引

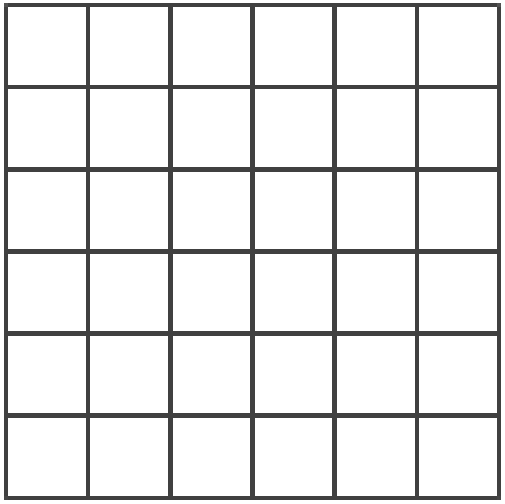
- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

空间索引

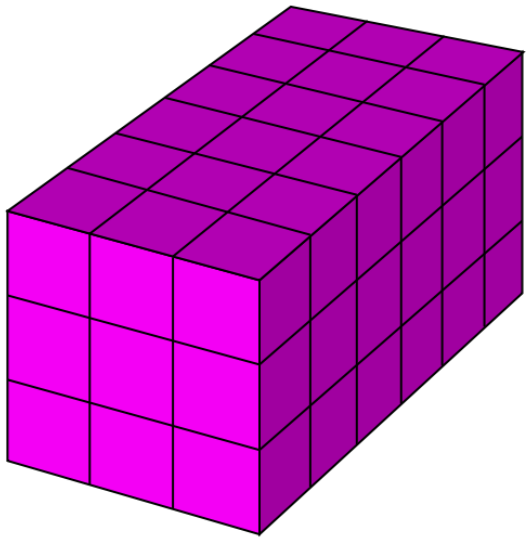
- 空间索引
  - 依据空间实体的位置和形状或空间实体之间的某种空间关系，按一定顺序排列的一种数据结构，其中包含空间实体的概要信息，如对象的标识，最小边界矩形及指向空间实体数据的指针
- 传统的数据索引技术——属性数据
  - B树，二叉树，ISAM索引，Hash索引
- 空间索引——空间数据
  - 网格索引，四叉树索引，空间目标排序索引，R树索引

空间索引

- 第二代和第三代GIS系统对空间对象采用的索引方式没有太大的变化
- 将空间索引整合进RDBMS中的实现方式使得高维的空间索引有可能被合理的使用在数据库内核的查询优化整个流程中，实现更好的查询性能
- 常用的空间索引
  - 网格索引
  - 四叉树索引
  - 空间填充曲线索引
  - R树索引及其变体



均匀网格 (Cubic)

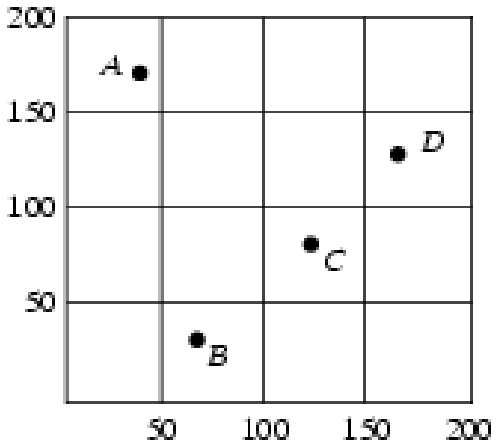


5.5 空间索引

- 5.5.1 网格索引
- 5.5.2 四叉树索引
- 5.5.3 空间填充曲线索引
- 5.5.4 R树索引及其变体

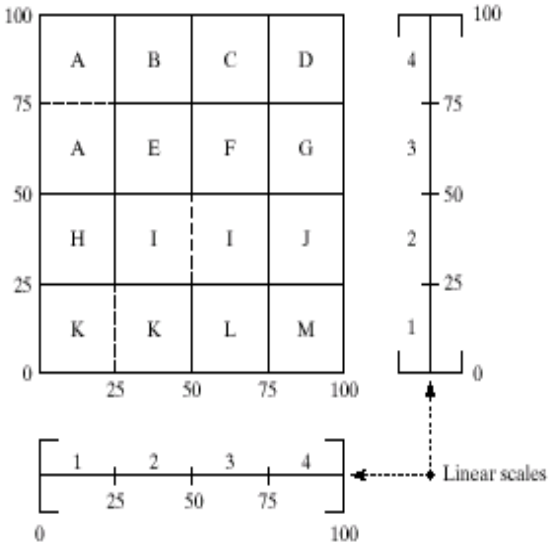
Ideas Behind Grid Files

- Basic idea - Divide space into cells by a grid
  - Example: latitude-longitude, ESRI Arc/SDE
  - Store data in each cell in distinct disk sector
  - Efficient for find, insert, nearest neighbor
  - But may have wastage of disk storage space
  - Non-uniform data distribution over space



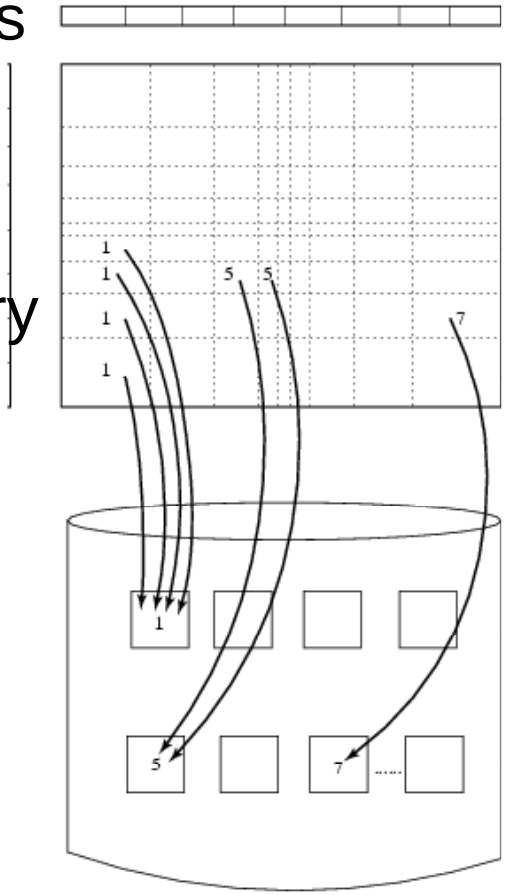
Ideas Behind Grid Files

- Refinement of basic idea into Grid Files
  - 1. Use non-uniform grids
    - Linear scale store row and column boundaries
  - 2. Allow sharing of disk sectors across grid cells
    - See next slide



Grid Files

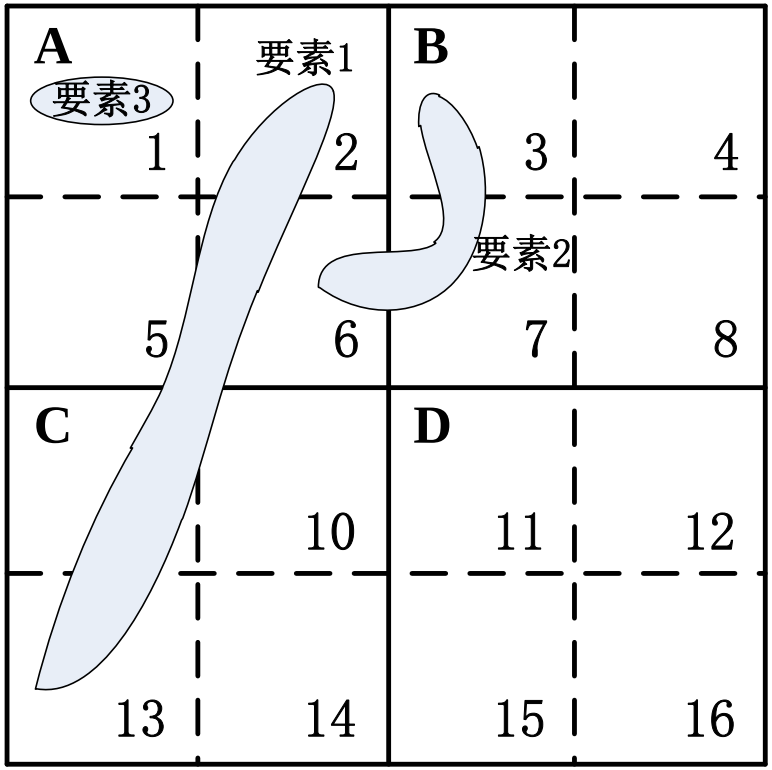
- Grid File component
  - Linear scale - row/column boundaries
  - Grid directory: cell → disk sector address
  - Data sectors on disk
- Operation implementation
  - Scales and grid directory in main memory
  - Steps for find, nearest neighbor
    - Search linear scales
    - Identify selected grid directory cells
    - Retrieve selected disk sectors
- Performance overview
  - Efficient in terms of I/O costs
  - Needs large main memory for grid directory



网格索引(补充)

- 网格索引的基本思想是将研究区域用横竖线划分大小相等和不等的网格，每个网格可视为一个桶（bucket），记录落入每一个网格区域内的空间实体编号
- 进行空间查询是，先计算出查询对象所在网格，再在该网格中快速查询所选空间实体

网格索引(补充)



索引1	
网络号	要素号
A	1, 2, 3
B	2
C	3

索引2	
网络号	要素号
1	3
2	1
3	2
5	1
6	1, 2
7	2
9	1
10	1
13	1

网格索引(补充)

- 网格索引优点：简单，易于实现，具有良好的可扩展性
- 缺点：网格大小影响网格索引检索性能
- 理想的网格大小是使网格索引记录不至于过多，同时每个网格内的要素个数的均值与最大值尽可能地少
- 获得较好的网格划分：
  - 可以根据用户的多次试验来获得经验最佳值
  - 可以通过建立地理要素的大小和空间分布等特征值来定量确定网格大小

5.5 空间索引

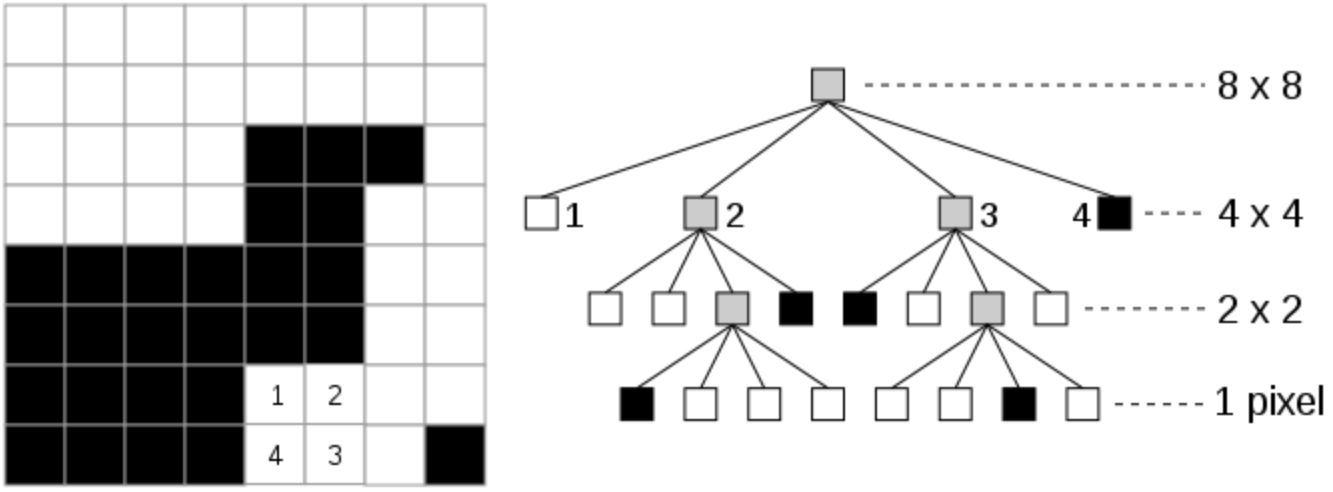
- 5.5.1 网格索引
- 5.5.2 四叉树索引
- 5.5.3 空间填充曲线索引
- 5.5.4 R树索引及其变体

四叉树索引

- 四叉树索引是为了实现要素真正被网格分割，同时保证桶内要素不超过一个量而提出的一种空间索引方法
- 首先将整个数据空间分割成为四个相等的矩阵，分别对应西北（NW），东北（NE），西南（SW），东南（SE）四个象限
- 若每个象限内包含的要素不超过给定的桶量则停止，否则对超过桶量的矩形再按照同样的方法进行划分，直到桶量满足要求或者不再减少为止，最终形成一颗有层次的四叉树

四叉树索引

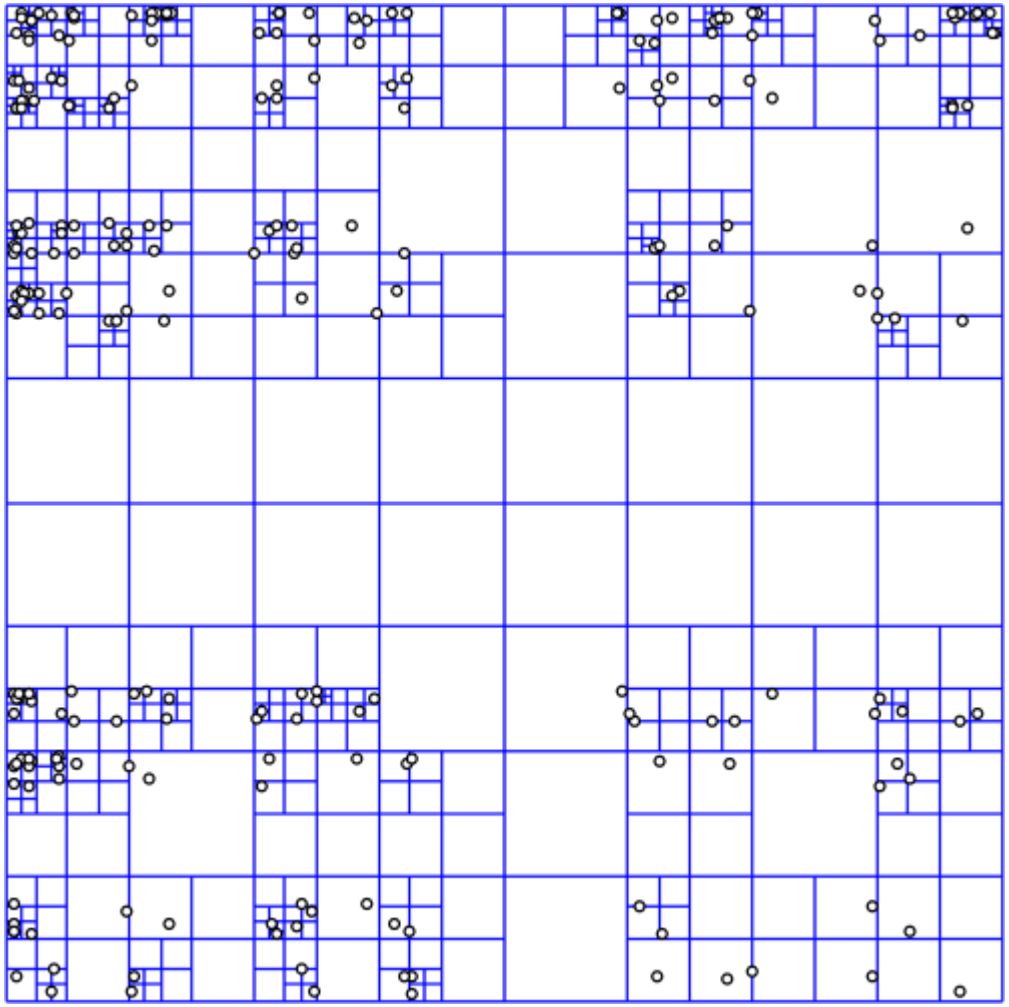
- 四叉树举例



思考：给定几何，如何遍历四叉树，获得与给定几何相交的候选几何对象？

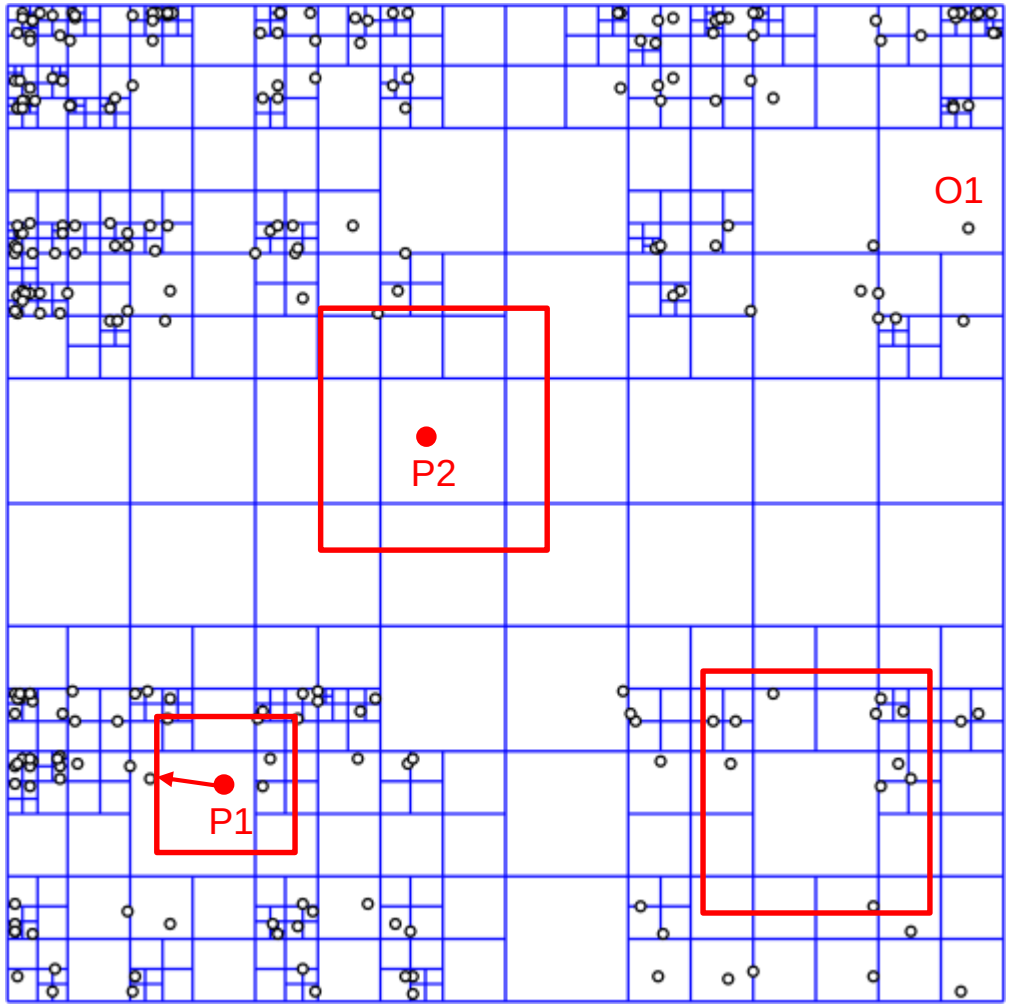
四叉树索引

- 四叉树举例



四叉树索引

- 实习3对几何对象包围盒构建索引，递归构造四叉树，使得叶节点几何对象数目小于给定数值
- 点查询
  - O1
- 区域查询
  - 包围盒是否相交
  - 几何是否相交（去重）
- KNN查询
  - P1 → 最短的最大距离
  - P2 → 区域大小？
- Spatial Join
  - 道路与站点的join



四叉树索引

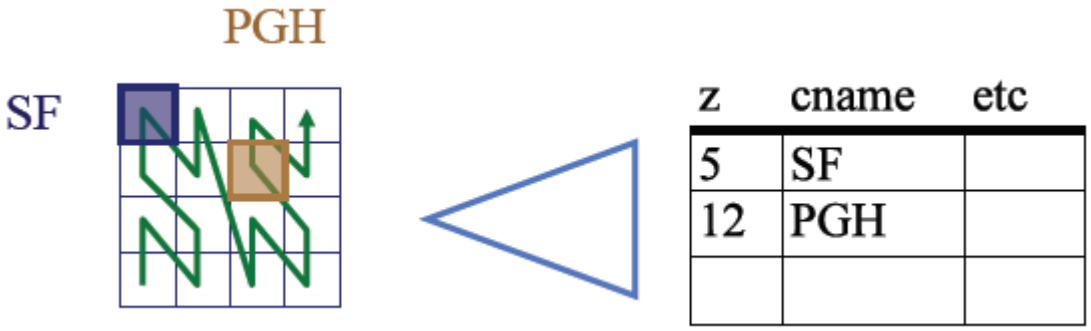
- 四叉树优缺点
  - 与网格索引相比，四叉树在一定程度上实现了地理要素真正被网格分割，保证了桶内要素不超过某个量，提高了检索效率
  - 对于海量数据，四叉树的深度会很深，影响查询效率
  - 可扩展性不如网格索引：当扩大区域时，需要重新划分空间区域，重建四叉树，当增加或删除一个对象，可能导致深度加一或减一，叶节点也有可能重新定位

5.5 空间索引

- 5.5.1 网格索引
- 5.5.2 四叉树索引
- 5.5.3 空间填充曲线索引
  - Main idea - 3 methods (P49-50)
  - Use w/ B-trees;
  - Algorithms (range, knn queries, spatial join ...)
  - Non-point (eg., region) data
  - Analysis; Variations
- 5.5.4 R树索引及其变体

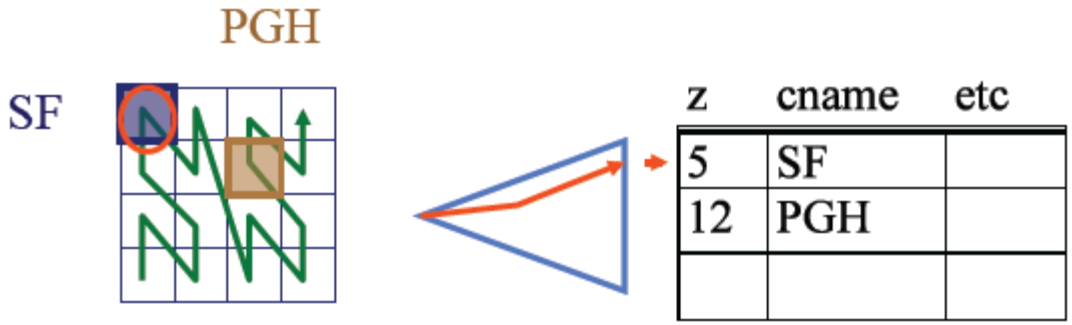
Z-Curve – Usage & Algorithms

- Q1: How to store on disk?
- A: treat z-value as primary key; feed to B-tree
- Major advantages with B-tree
  - Already inside commercial systems (no coding/debugging)
  - Concurrency & recovery is ready

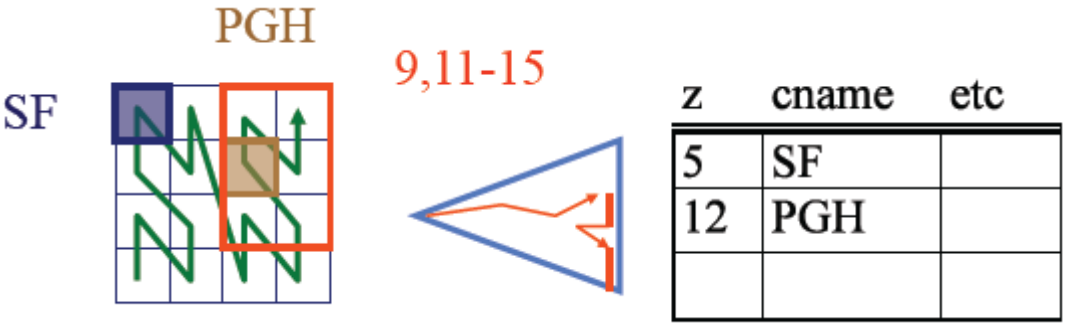


Z-Curve – Usage & Algorithms

- Q2: queries? (eg.: find city at (0, 3))
- A: find z-value; search B-tree

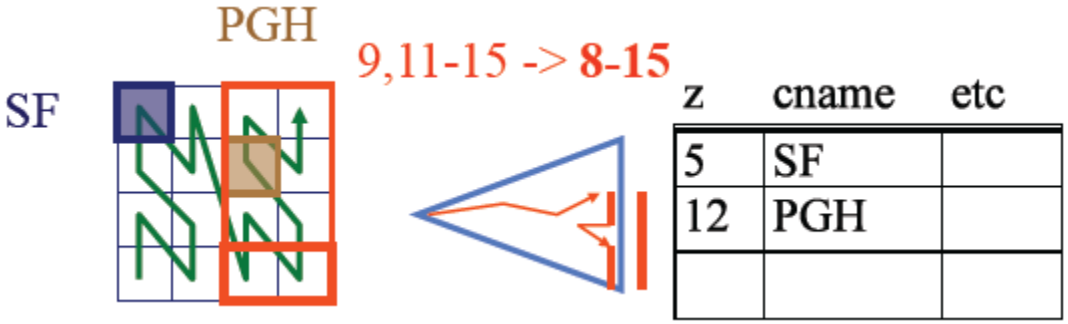


- Q2: range queries?
- A: compute ranges of z-values; use B-tree

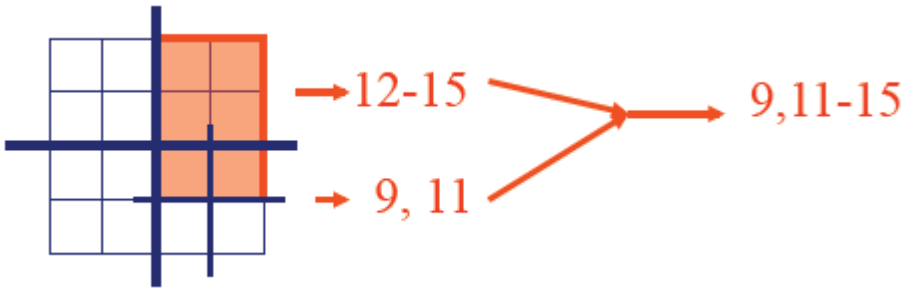


Z-Curve – Usage & Algorithms

- Q2': range queries – how to reduce # of qualifying of ranges?
- A: augment the query!

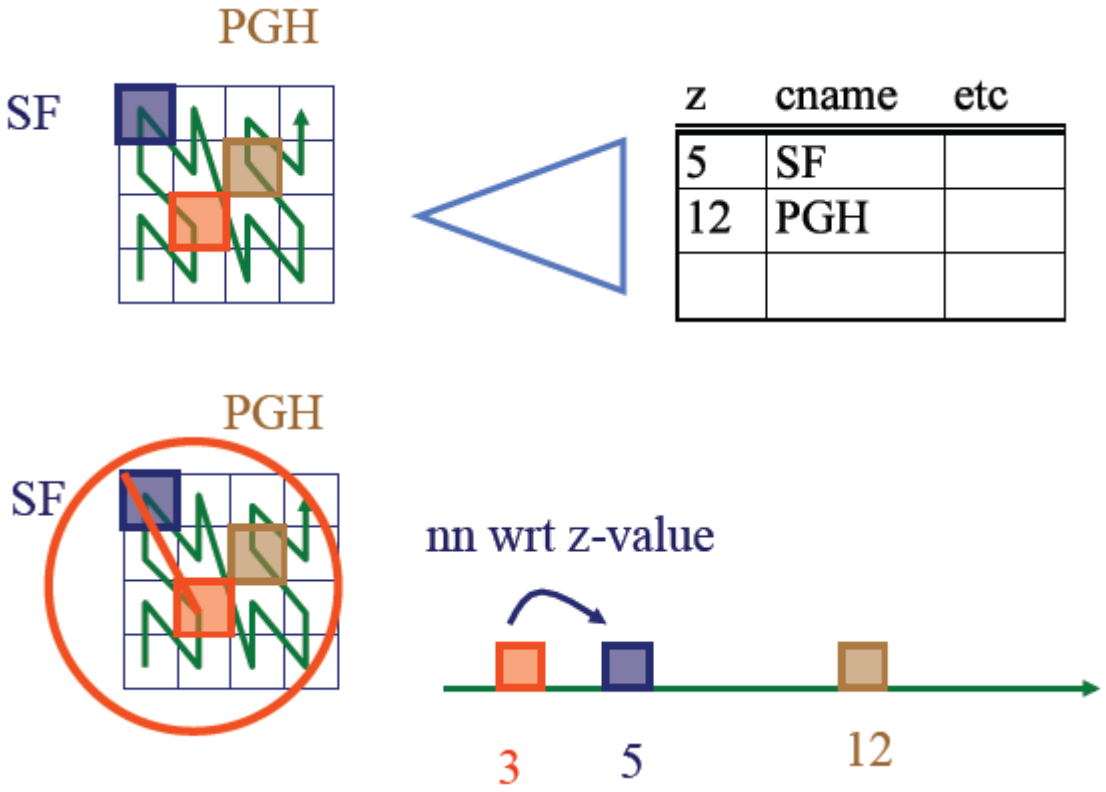


- Q2'': range queries – how to break a query into ranges?
- A: recursively, quadtree-style; decompose only non-full quadrants



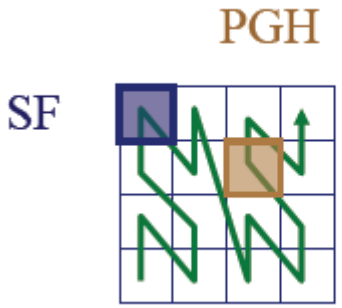
Z-Curve – Usage & Algorithms

- Q3: k-nn queries? (say, 1-nn)
- A: traverse B-tree; find nn wrt z-values and ask a range query



Z-Curve – Usage & Algorithms

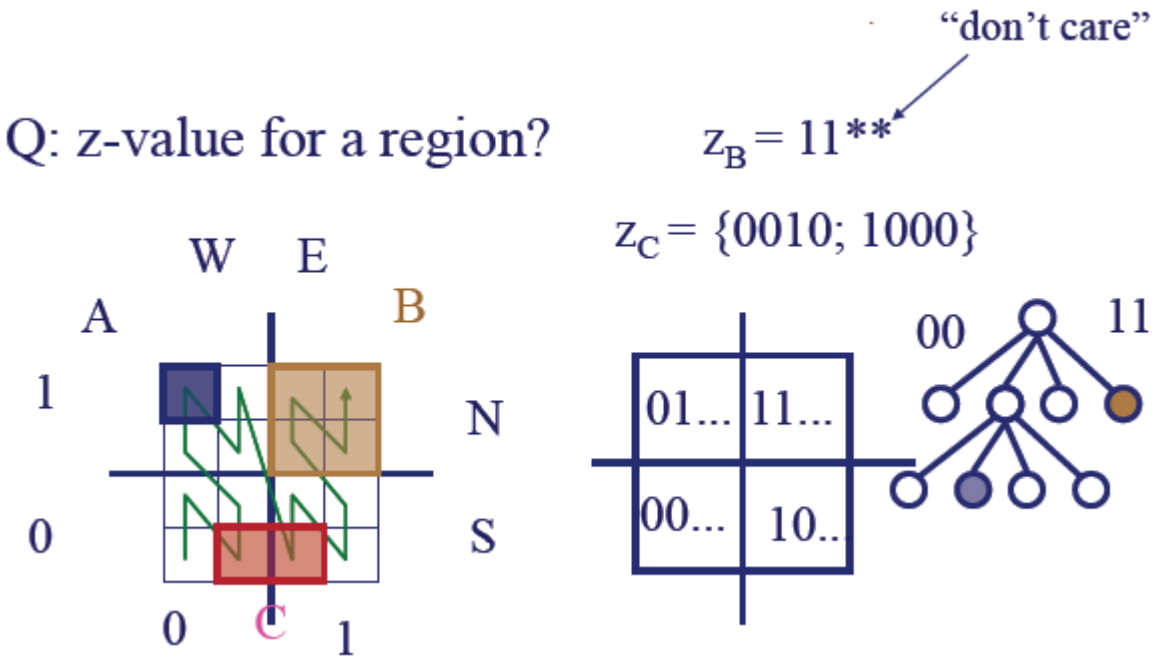
- Q4: all-pairs queries? (all pairs of cities within 10 miles from each other?)



(we'll see 'spatial joins' later: *find all PA counties that intersect a lake)*

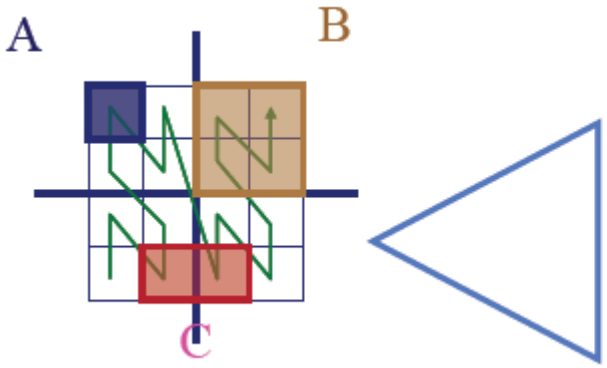
Z-Curve – Regins

- Q: z-value for a region?
- A: 1 or more z-values; by quadtree decomposition



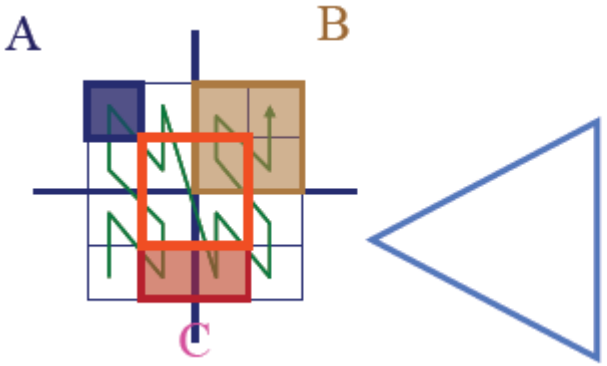
Z-Curve – Regins

- Q: how to store in B-tree? A: sort (* < 0 < 1)



z	obj-id	etc
0010	C	
0101	A	
1000	C	
11**	B	

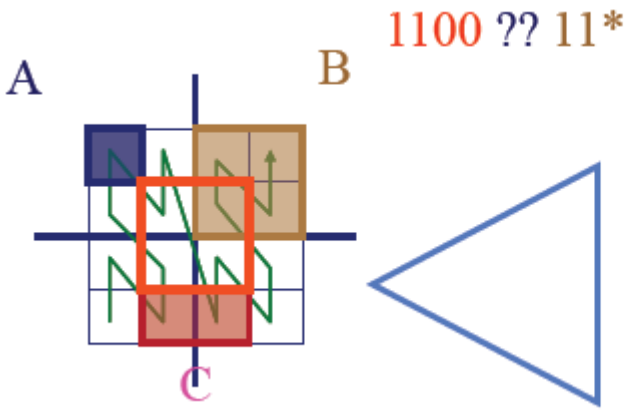
- Q: how to search (range query)
- A: break query in z-values; check B-tree



z	obj-id	etc
0010	C	
0101	A	
1000	C	
11**	B	

Z-Curve – Regins

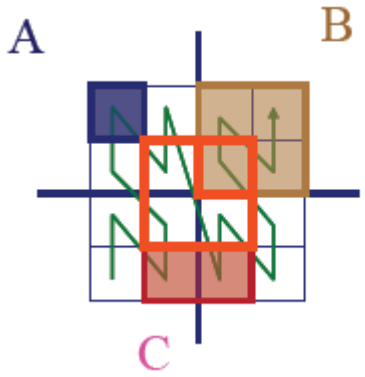
- Almost identical to range queires for point data, except for the “don’t cares” – i.e.,
  - z1 = 1100 ?? 11** = z2
- Specifically: does z1 contain/avoid/intersect z2?
- Q: what is the criterion to decide?



z	obj-id	etc
0010	C	
0101	A	
1000	C	
11**	B	

Z-Curve – Regins

- Q: what is the criterion to decide?
- A: prefix property: let r1, r2 be the correspoinding regions, and let r1 be the smallest (=> z1 has fewest '*'s. Then:
  - r2 will either contain completely, or avoid completely r1.
  - It will contrain r1, if z2 is the prefix of z1

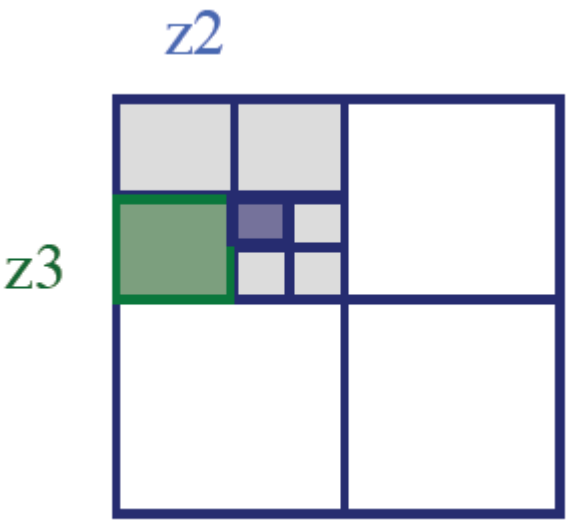


1100 ?? 11**

region of z1:
completely contained in
region of z2

Z-Curve – Regins

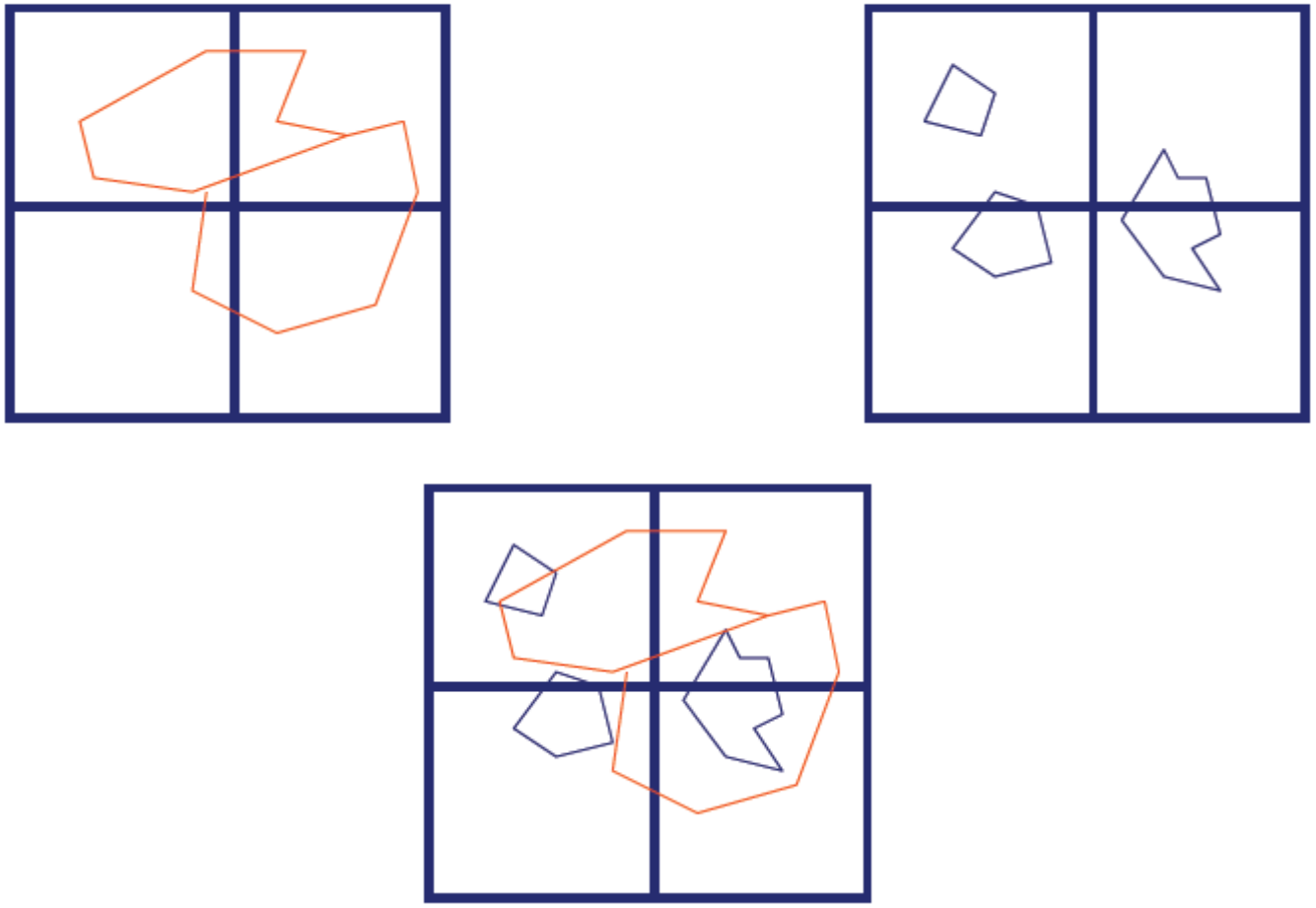
- Drill (True/False). Given:
  - z1= 011001**
  - z2= 01******
  - z3= 0100****
- T/F z2 contains z1
  - True (prefix property)
- T/F z3 contains z1
  - False (disjoint)
- T/F z3 contains z2
  - False (z2 contains z3)



Z-Curve – Regins

- Spatial joins: find (quickly) all

counties intersecting lakes



Z-Curve – Regins

- Spatial joins: find (quickly) all counties intersecting lakes
  - Naïve algorithm: O(N * M)
  - Something faster?

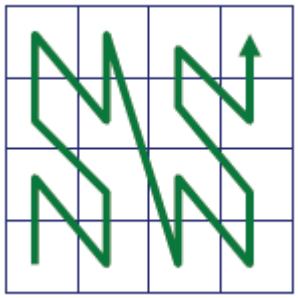
z	obj-id	etc
0010	ALG	
…	…	
1000	WAS	
11**	ALG	

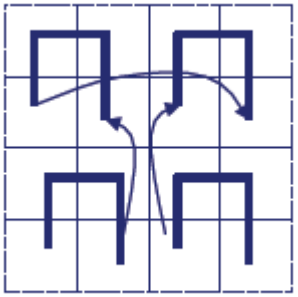
z	obj-id	etc
0011	Erie	
0101	Erie	
…		
10**	Ont.	

- Solution: merge the list of (sorted) z-values, looking for the prefix property
  - '*' needs careful treatment
  - Need dumplication elimination

Z-Curve – Variations

- Q: is z-curve the best we can do?
- A: probably not – occasional long ‘jumps’
- Q: then?
- A1: Gray codes



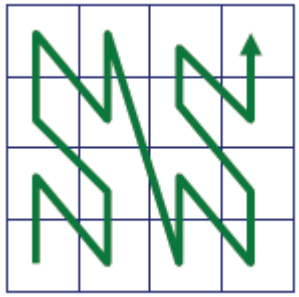


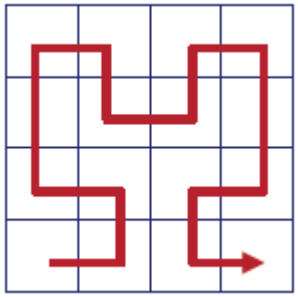
Ingenious way to spot flickering LED

0	00	000	0
1	01	001	1
	11	011	2
	10	010	3
		110	4
		111	5
		101	6
		100	7

Z-Curve – Variations

- A1: Gray codes
- A2: Hilbert curve (a.k.a, Hilbert-Peano curve)
  - Looks better (never long jumps)



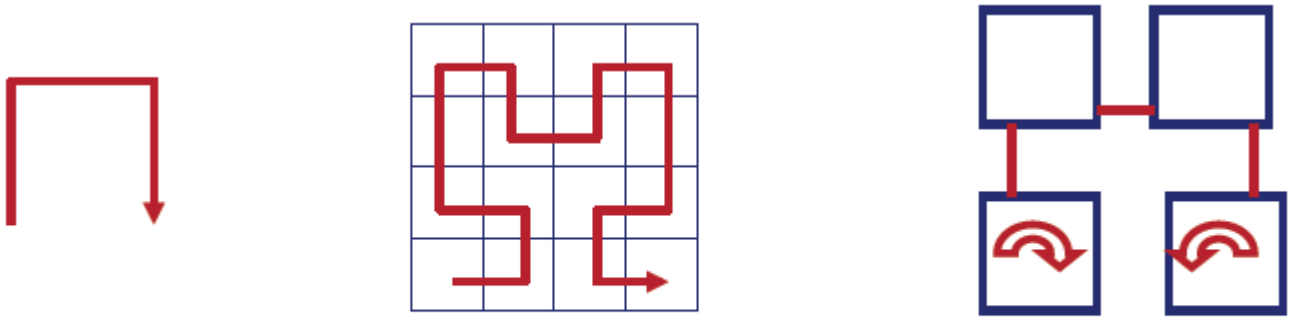


David Hilbert
(1862-1943)

Giuseppe Peano
(1858-1932)

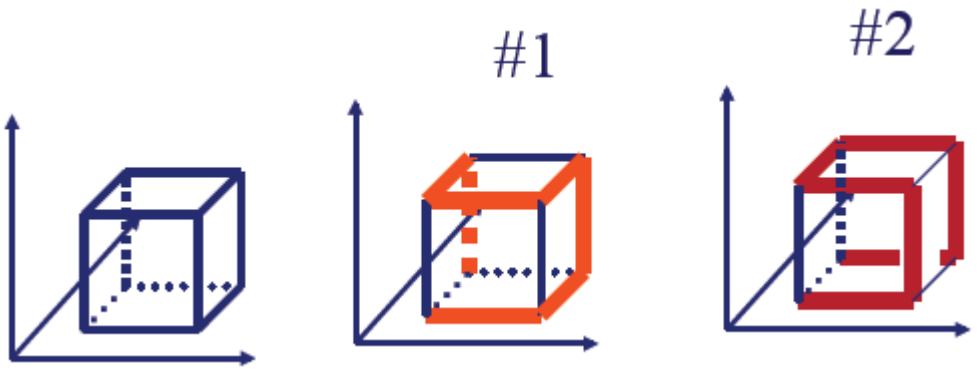
Z-Curve – Variations

- Q: function for Hilbert Curve (h = *f*(*x*, *y*))?
- A1: bit-shuffling, followed by post-processing, to account for rotations. Linear on # bits (P66)
  - 实习3基于bit-shuffling实现Z-Curve函数(z = *f*(*x*, *y*))
- A2: recursively, quadtree-style; rotation for next order
  - 实习3基于quadtree-style，实现order不断递增时的Hilbert Curve (h = *f*(*x*, *y*))



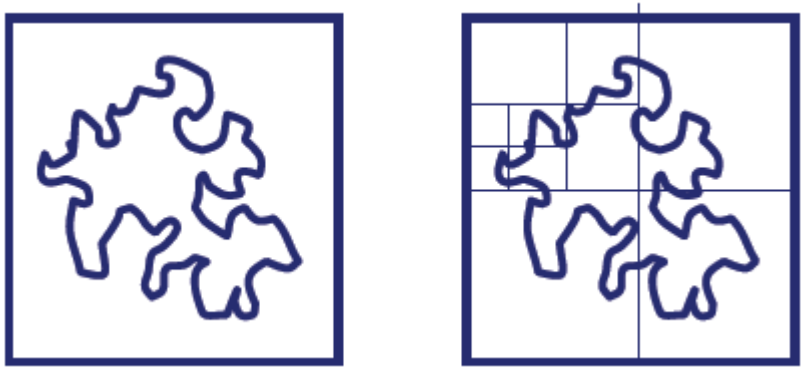
Z-Curve – Variations

- Q: how about Hilbert curve in 3D? nD?
- A: exists (and is not unique!). Eg., 3D order-1 Hilbert curves (Hamiltonian paths on cube)



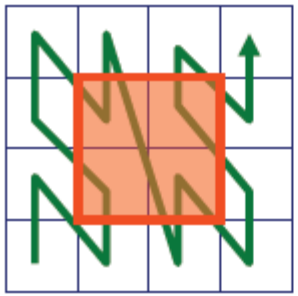
Z-Curve – Analysis

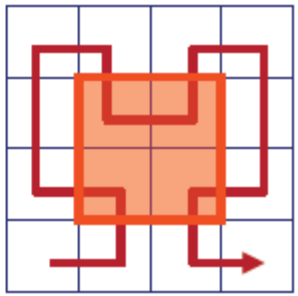
- Q: how many pieces (‘quad-tree blocks’) per region?
- A: proportional to perimeter (surface etc)
- Q: should we decompose a region to full detail (and store in B-tree)?
- A: no! approximation with 1-3 pieces/z-values is best [Orenstein90]

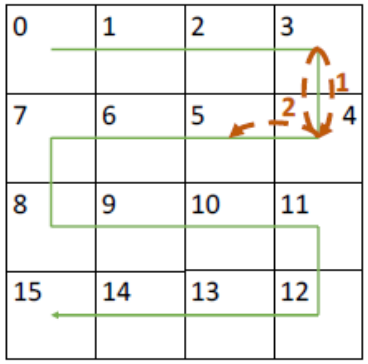


Z-Curve – Analysis

- Q: how to measure the “goodness” of a curve?
- A: e.g., avg. # of runs, for range queries







4 runs

3 runs

(#runs ~ #disk accesses on B-tree)

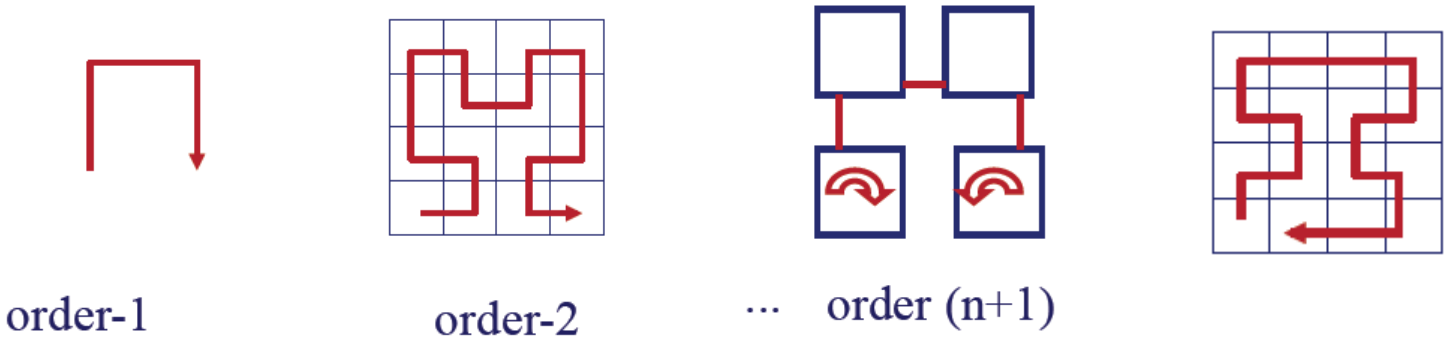
实习3 pair-distance

$$D = \sum_{i=0}^{n-3} (d(p_i, p_{i+1}) + d(p_i, p_{i+2}))$$

- Q: so, is Hilbert really better?
- A: 27% fewer runs, for 2D (similar for 3D)
- There are formulas for #runs, #of quadtree blocks [Jagadish; Moon+ etc]

Z-Curve – Fun Observations

- Hilbert and Z-ordering curves: “space filling curves”
  - Eventually, they visit every point in nD space
  - They show that the plane has as many points as a line



- Hilbert (like) curve for video encoding
  - Given a frame, visit its pixels in randomized hilbert order; compress; and transimit
  - In general, Hilbert curve is great for preserving distances, clustering, vector quantization etc

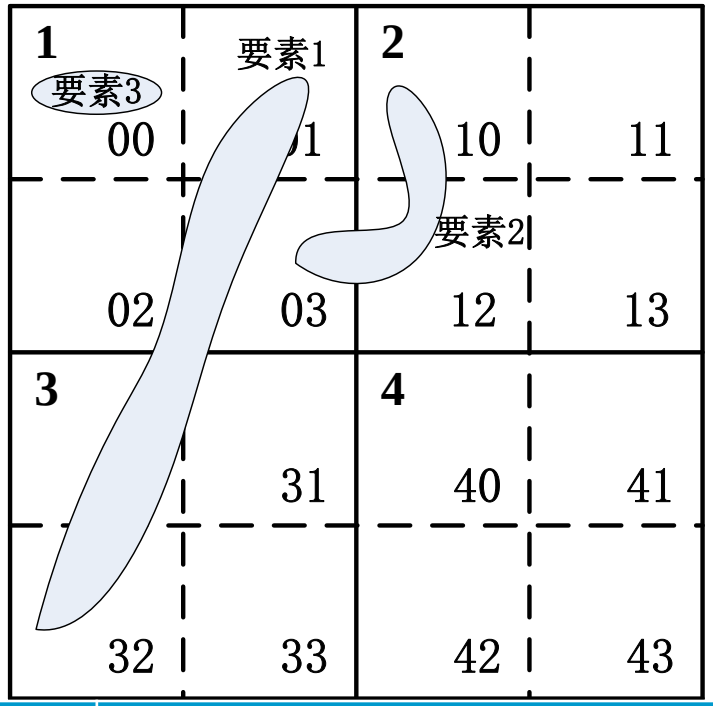
基于空间填充曲线的索引(补充)

- 多维空间索引结构的设计一般来说，难于线性索引结构的设计，原因在于多维空间中无法用线性顺序来保持空间对象之间的位置关系
- 多维空间索引结构设计：寻找一种映射关系，它至少能够在一定程度上保持空间对象之间的位置关系

基于空间填充曲线的索引(补充)

- 基于空间填充曲线的索引 [高维数据降到一维]
- 空间填充曲线的实质是按照某种策略将数据空间细分为许多格子，并根据一定的方法为这些网格编码，每个网格对应一个唯一的编码，这样可以将多维的空间数据降维到一维空间中，再利用现有的DBMS中较为成熟的一维索引技术对空间填充码建立索引
- 当进行空间查询时，先把给定的查询区域转换为覆盖此区域的一系列网格的编码，再到一维的网格编码索引中，找到对应的记录作为过滤步的候选集

基于空间填充曲线的索引(补充)



编码	属性	几何对象	所在网格编码
1	…	<WKB_Geometry>	0
2	…	<WKB_Geometry>	15
3	…	<WKB_Geometry>	1
4	…	<WKB_Geometry>	16

基于空间填充曲线的索引(补充)

- 与网格索引的区别
- 网格索引并不强调一定用DBMS已有的索引来检索，故其在实现上不一定要遵循关系数据库的一些规范化准则，如非原子属性
- 基于空间填充曲线的索引对象所在的网格编码是唯一的，由于二维几何对象（线、面）有一定的延展性，就难以用某一个固定阶的曲线来填充，故在实现中尝尝需要用到不同阶的填充曲线编码
- SQL Server 2008采用基于多阶Hilbert填充曲线的空间索引

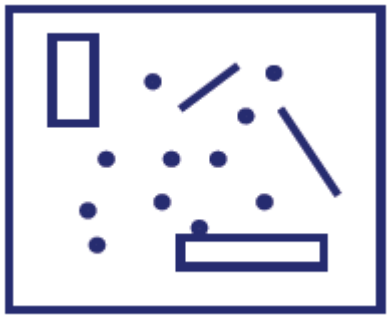
5.5 空间索引

- 5.5.1 网格索引
- 5.5.2 四叉树索引
- 5.5.3 空间填充曲线索引
- 5.5.4 R树索引及其变体
  - Main idea: R-Tree, R+Tree, node format
  - Algorithms: insertion/split, deletion
  - Search: object, range, nn, spatial joins
  - Performance analysis (自学内容，不作要求)
  - Variations (packed; hilbert;...) (自学内容，不作要求)

Problem

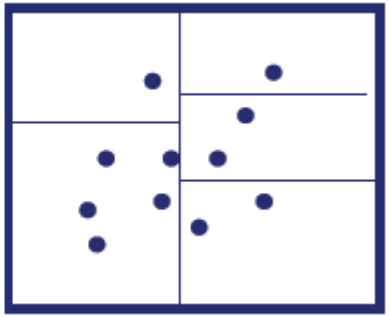
- Given a collection of geometric objects (points, lines, polygons, …)
- Organize them on disk, to answer spatial queries (range, nn, etc)

- Z-Curve: cuts regions to pieces -> dup. elim.
- How would we avoid that?
- Idea: try to extend/merge B-trees and k-d trees



K-d-B-trees

- [Robinson, 81]: if f is the fanout, split poinset in f parts; and so on, recursively
- But: insertions/deletions are tricky (splits may propagate downwards and upwards)
- No guarantee on space utilization



R-Trees – Main Idea

- [Guttman 84] Main idea: allow parents to overlap!
  - Guaranteed 50% utilization
  - Easier insertion/split algorithms
  - Only deal with Minimum Bounding Rectangles MBRs

Antonin Guttman
[http://www.baymoon.com/~tg2/]

注意：R树仅存储和比较几何对象的包围盒，获得包围盒重叠/相交后(filter, 与R树有关)，这些包围盒的真实几何再与区域查询框/查询点计算是否真的重叠/相交(refine, 与R树无关)

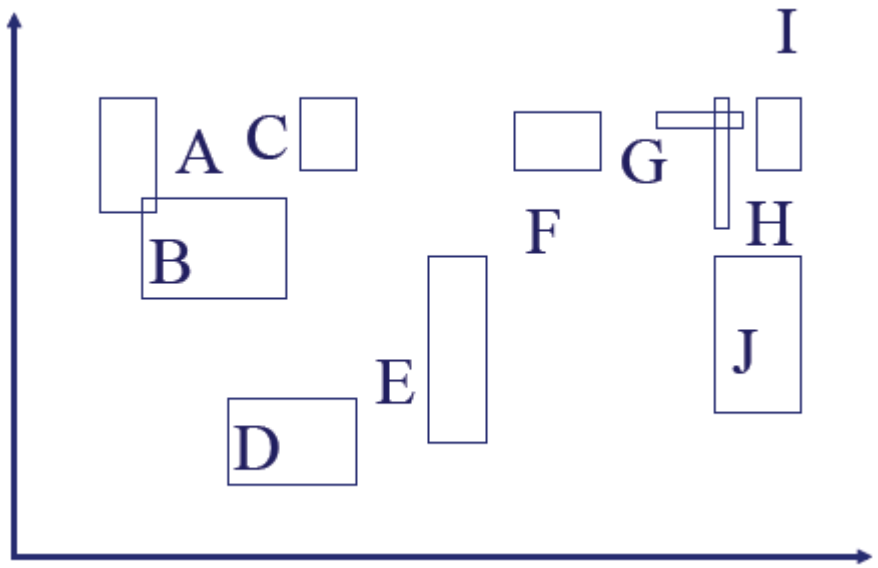
R-Trees – Main Idea

- Basic Idea
  - Use a hierarchical collection of rectangles to organize spatial data
  - Generalizes B-tree to spatial data sets
- Classifying members of R-tree family
  - Handling of large spatial objects
    - Allow rectangles to overlap - R-tree
    - Duplicate objects but keep interior node rectangles disjoint - R+tree
  - Selection of rectangles for interior nodes
    - Greedy procedures - R-tree, R+tree
    - Procedure to minimize coverage, overlap - packed R-tree
  - Other criteria exist

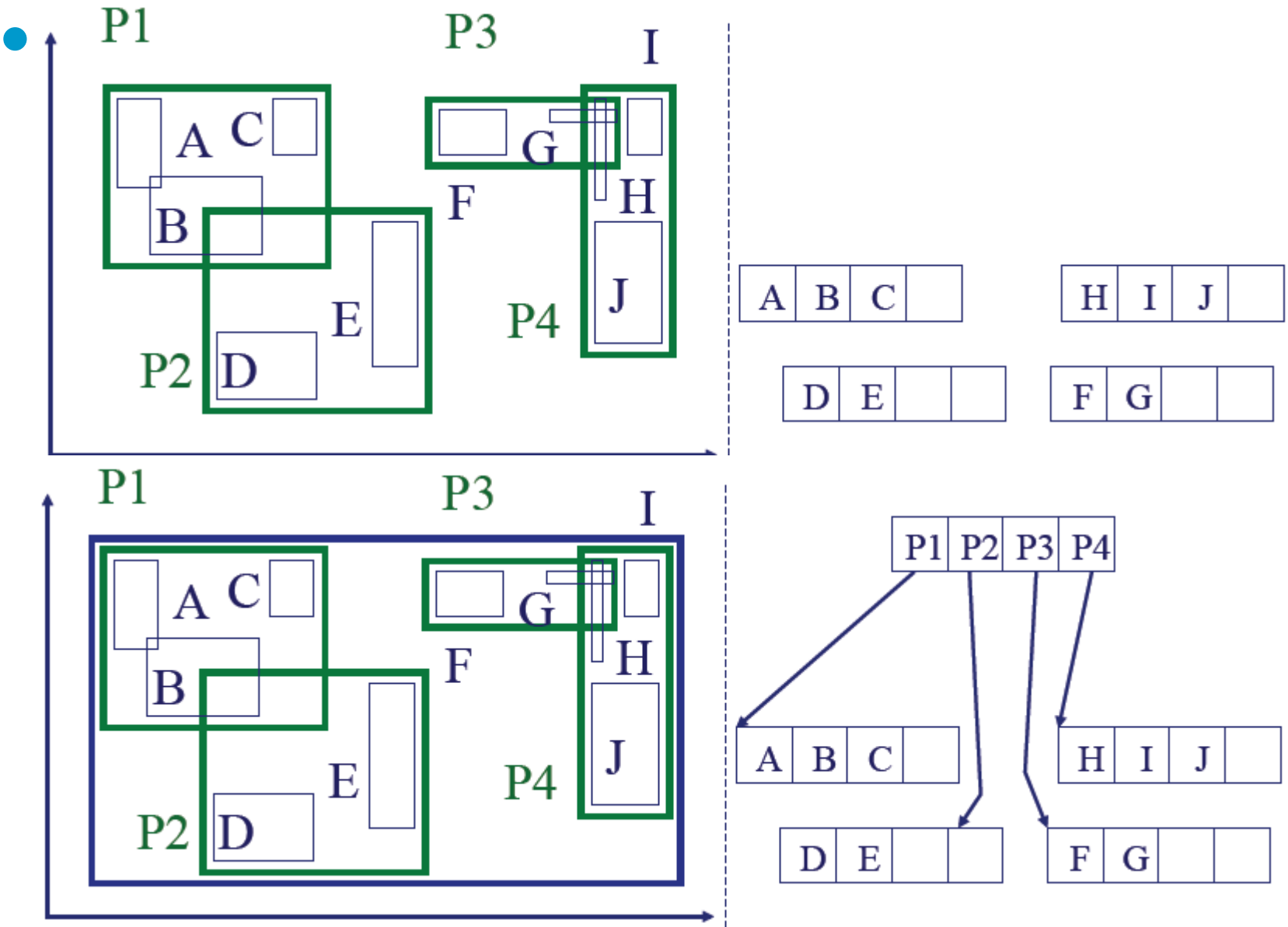
思考：几何属性为主键在某些DBMS中可能会报什么错误？

R-Trees – Main Idea

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

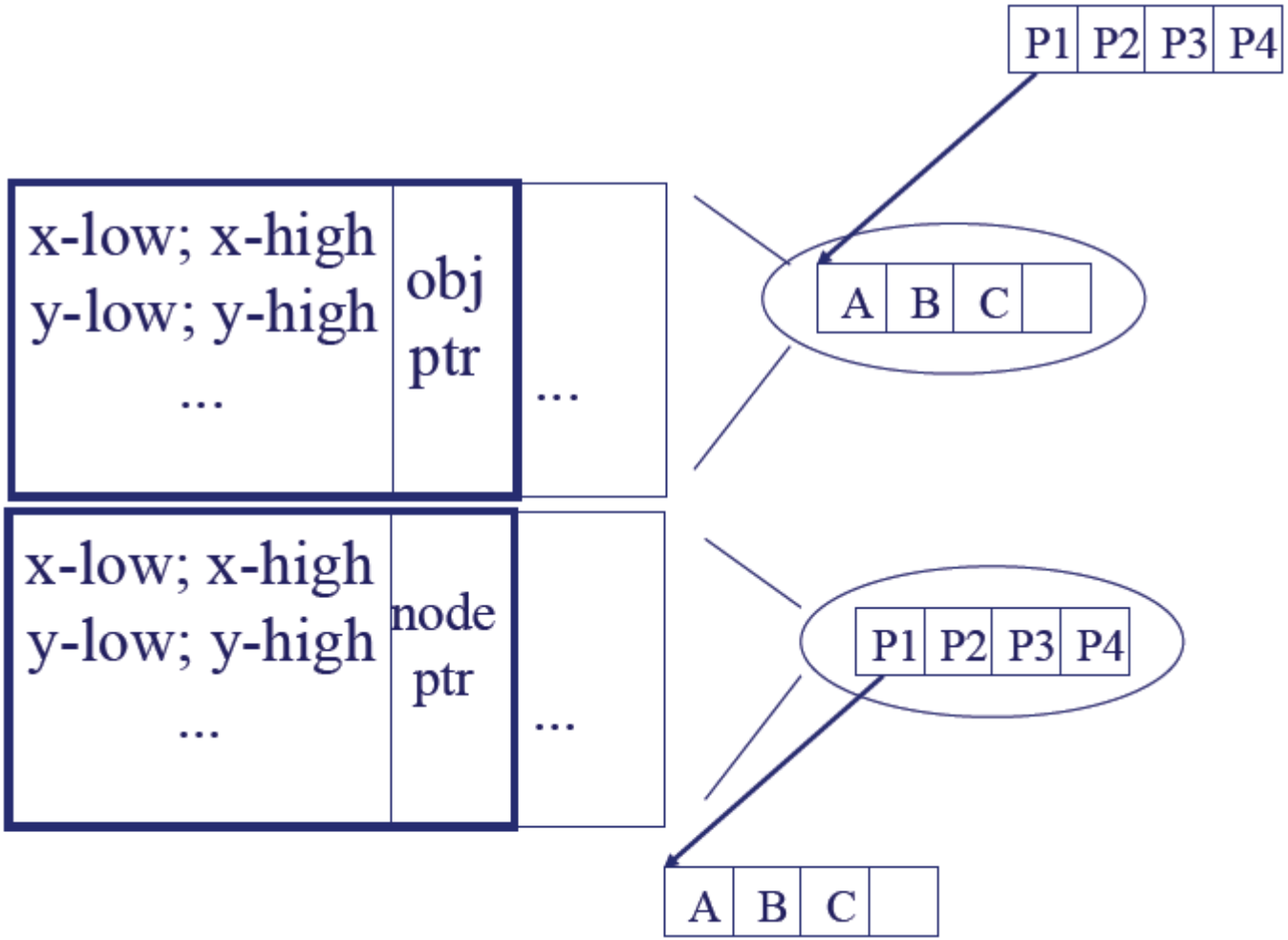


R-Trees – Main Idea



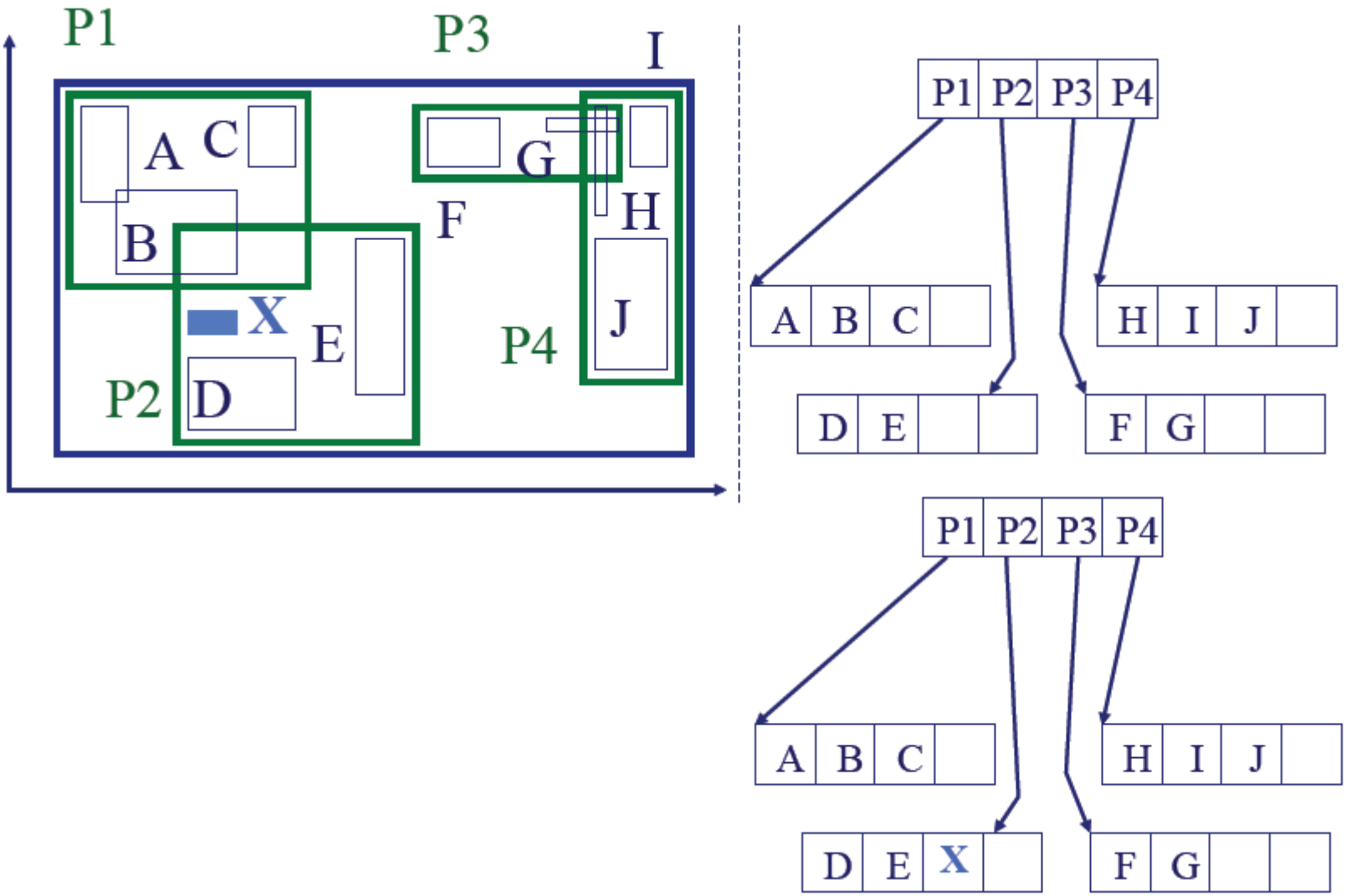
R-Trees – Node Format

- {(MBR; obj-ptr)} for leaf nodes
- {(MBR; node-ptr)} for non-leaf nodes



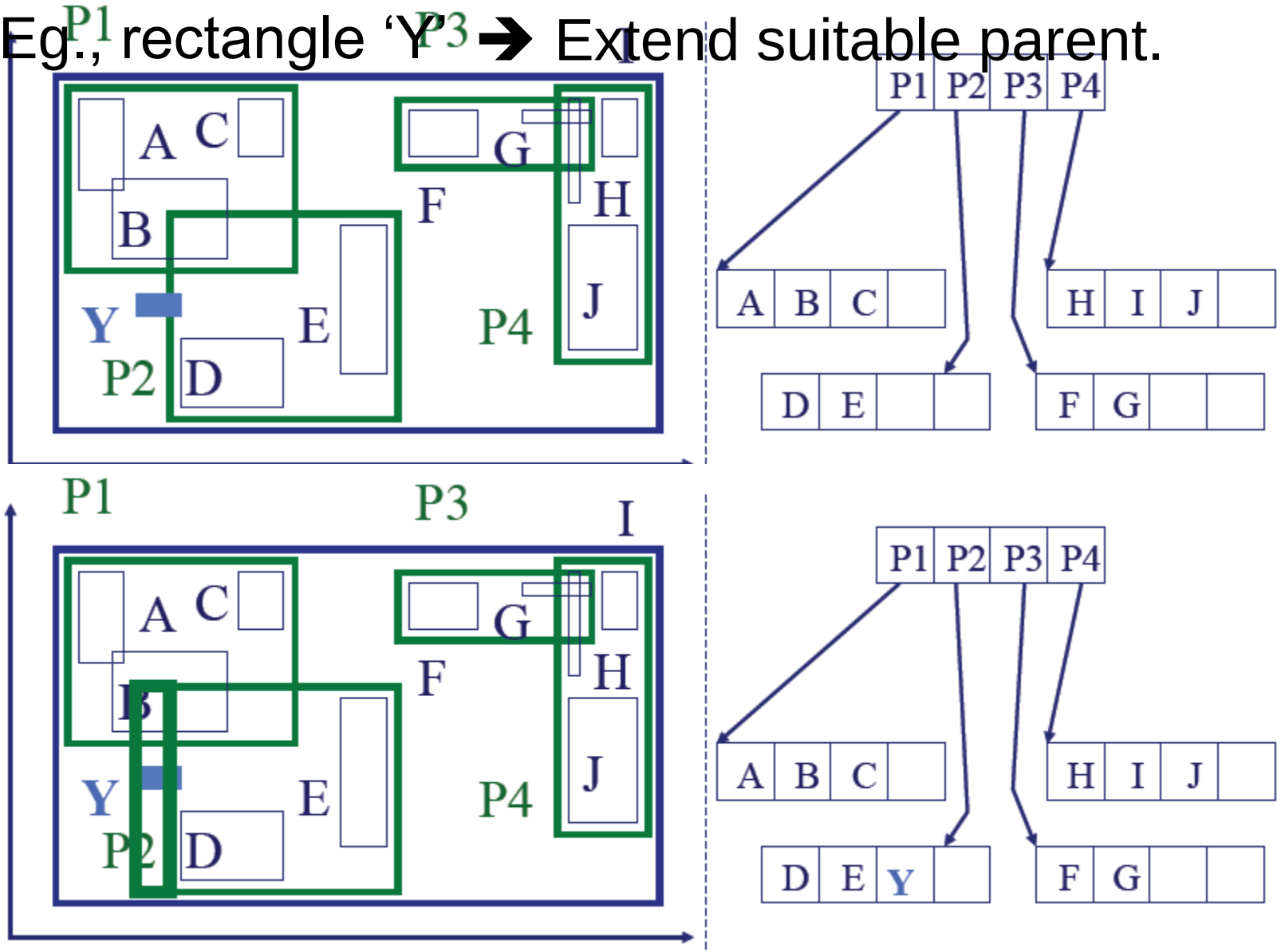
R-Trees – Insertion

- Eg., rectangle ‘X’



R-Trees – Insertion

- Eg., rectangle ‘Y’ ➔ Extend suitable parent.

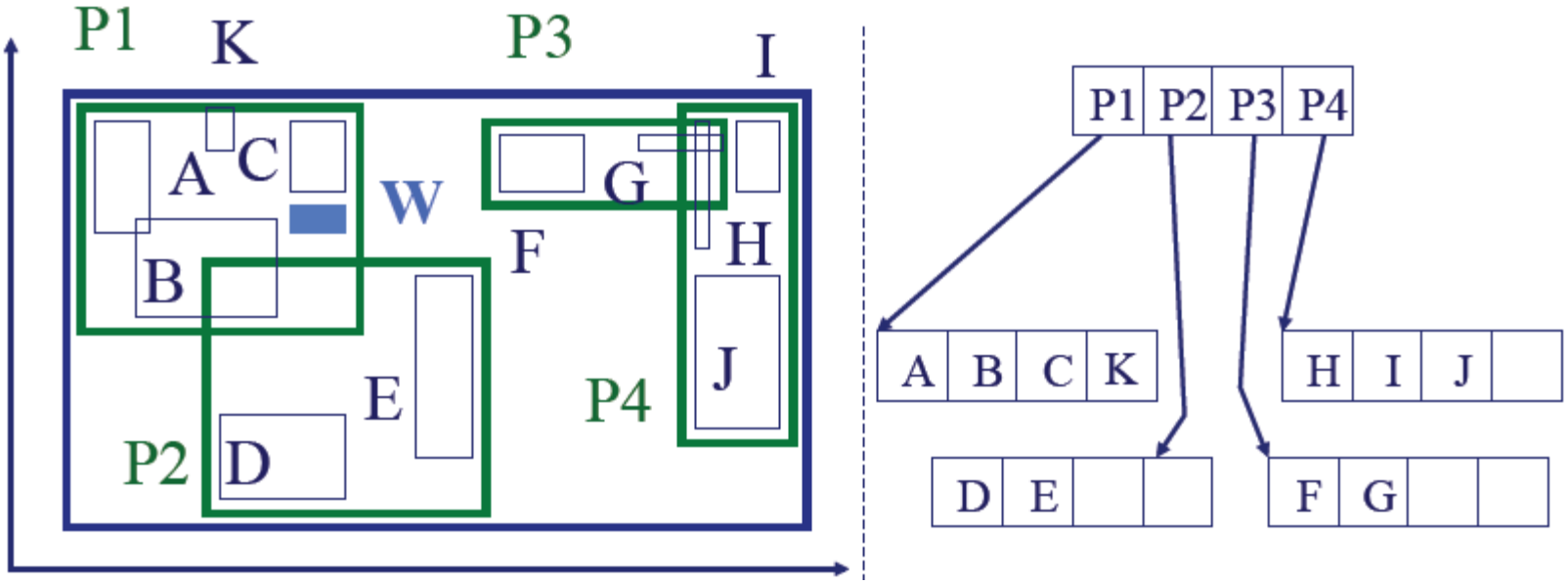


R-Trees – Insertion

- 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?

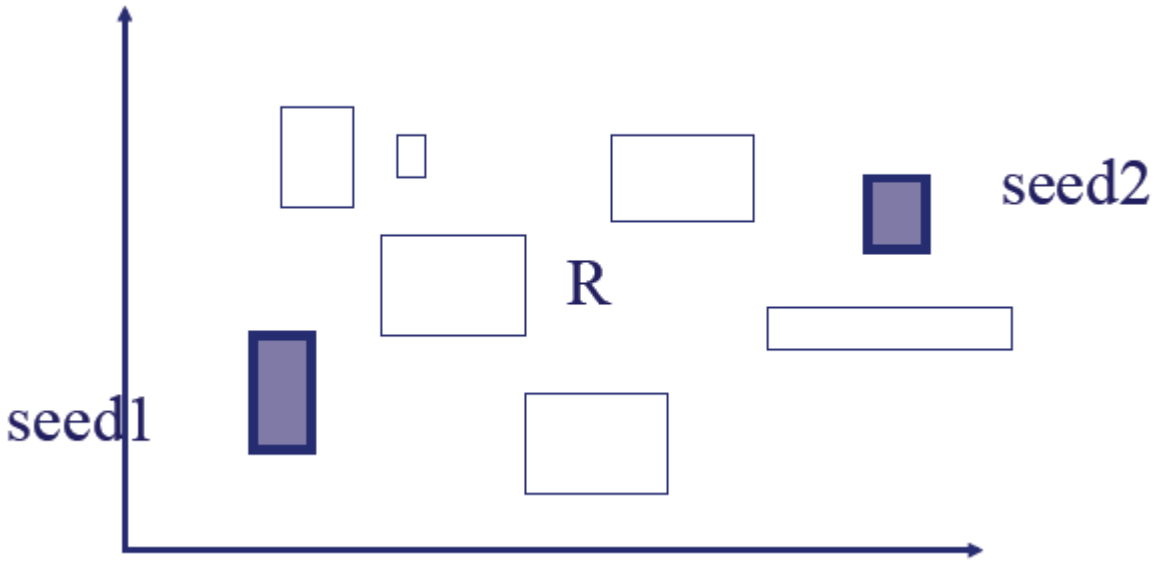
R-Trees – Insertion

- Eg., rectangle ‘W’, focus on ‘P1’ - how to split?
  - A1: plane sweep, until 50% of rectangles
  - A2: ‘linear’ split
  - A3: quadratic split
  - A4: exponential split

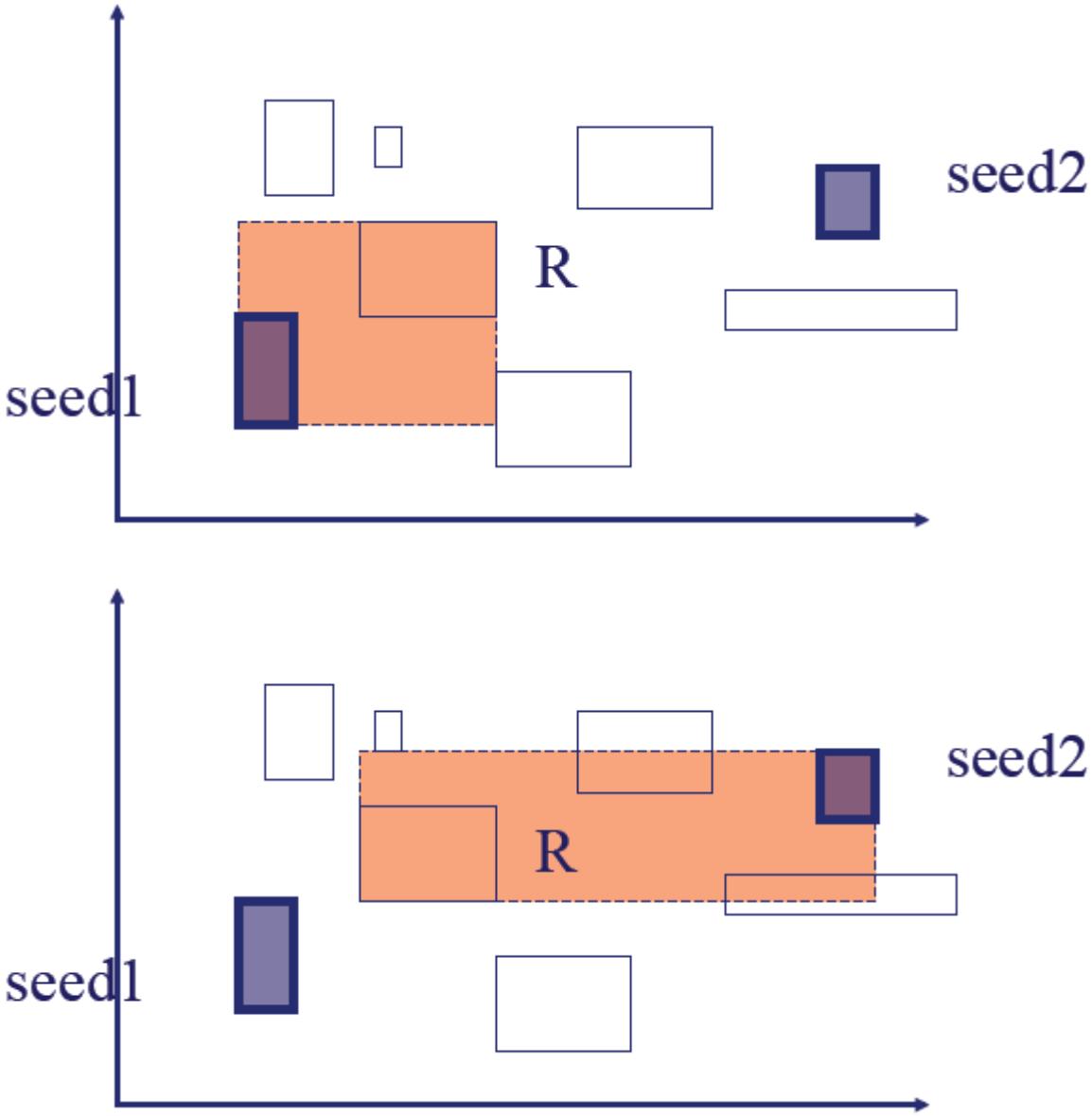


R-Trees – Insertion & Split

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



R-Trees – Insertion & Split



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

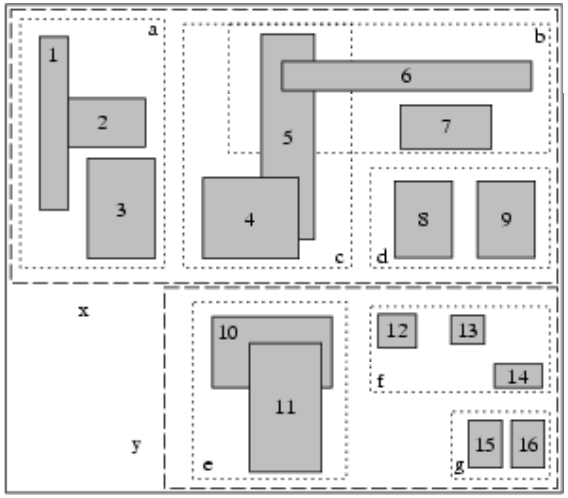
- Observations
  - Many more split algorithms exist

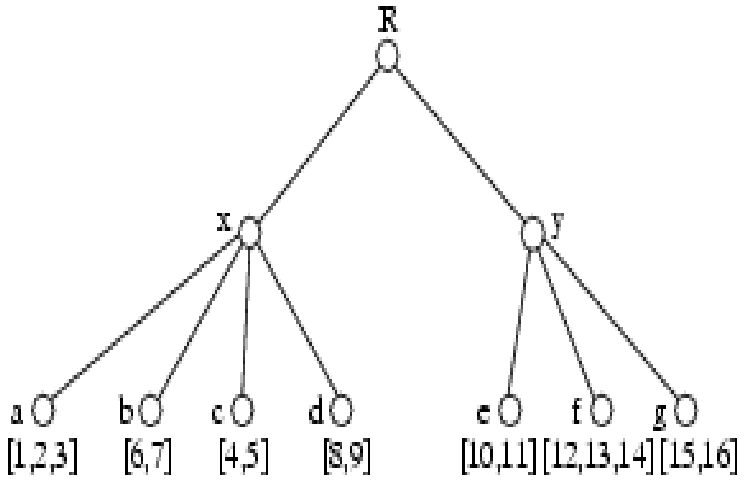
R-Trees – Deletion

- Delete rectangle
- If underflow
  - Temporarily delete all siblings (!)
  - Delete the parent node and
  - Re-insert them

R-Trees – Properties

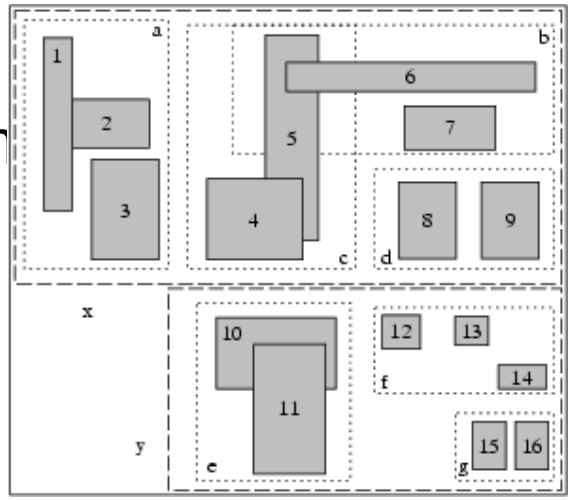
- Properties of R-trees
  - Balanced
    - Nodes are rectangle
    - Child's rectangle within parent's
  - Possible overlap among rectangles!

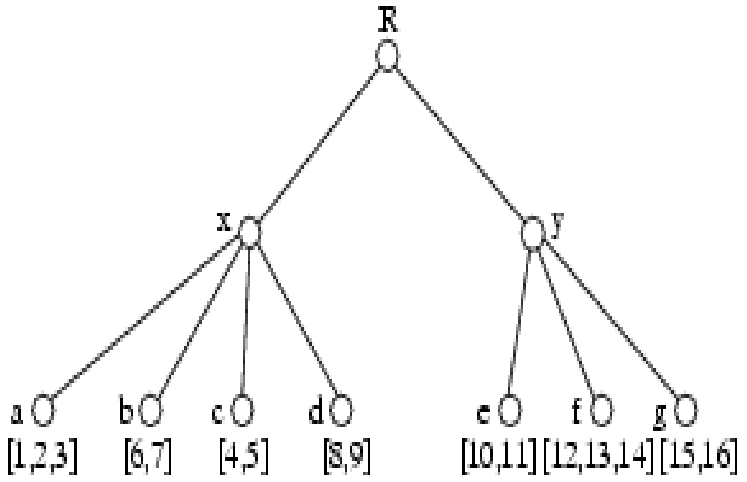




R-Trees – Object Search

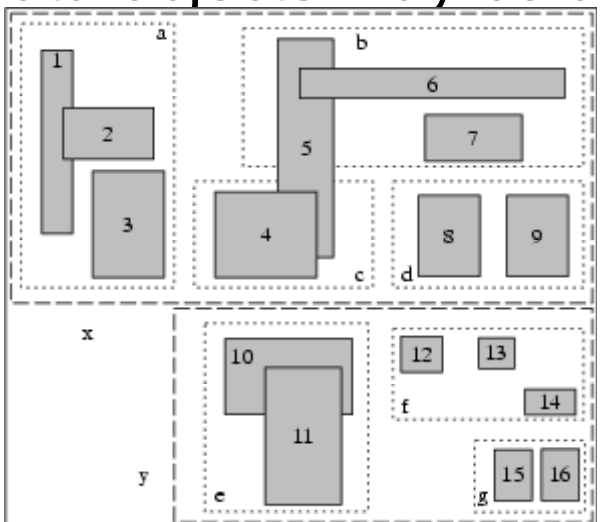
- Implementation of find operation
  - Search root to identify relevant children
  - Search selected children recursively
- Exercise: find record for rectangle 5
  - Root search identifies child x
  - Search of x identifies children b and c
  - Search of b does not find object 5
  - Search of c find object 5

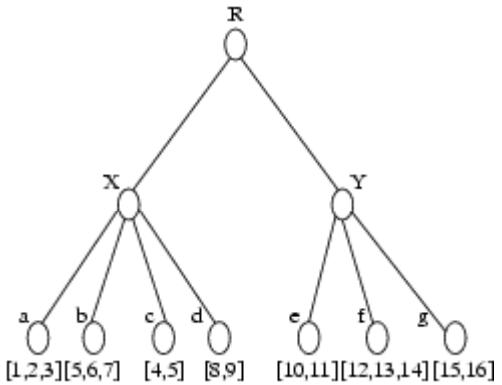




R+Trees – Properties

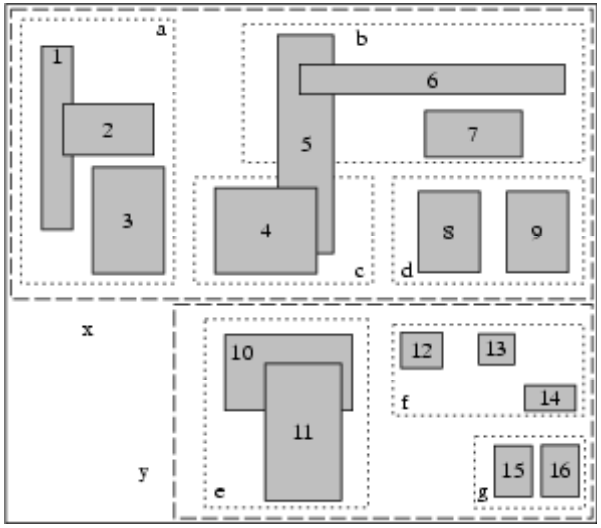
- Properties of R+trees
  - Balanced
  - Interior nodes are rectangle
    - Child's rectangle within parent's
    - Disjoint rectangles
  - Leaf nodes - MOBR of polygons or lines
    - Leaf's rectangle overlaps with parent's
  - Data objects may be duplicated across leafs

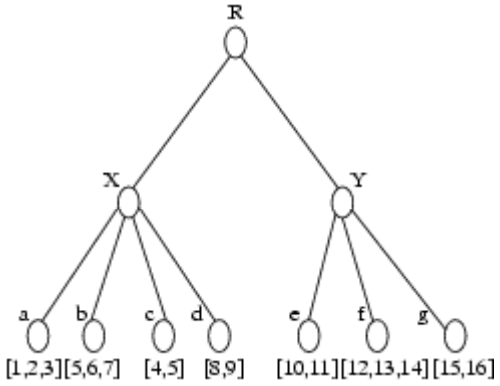




R+Trees – Object Search

- Find operation - same as R-tree
  - But only one child is followed down
- Exercise: find record for rectangle 5
  - Root search identifies child x
  - Search of x identifies children b and c
  - Search either b or c to find object 5

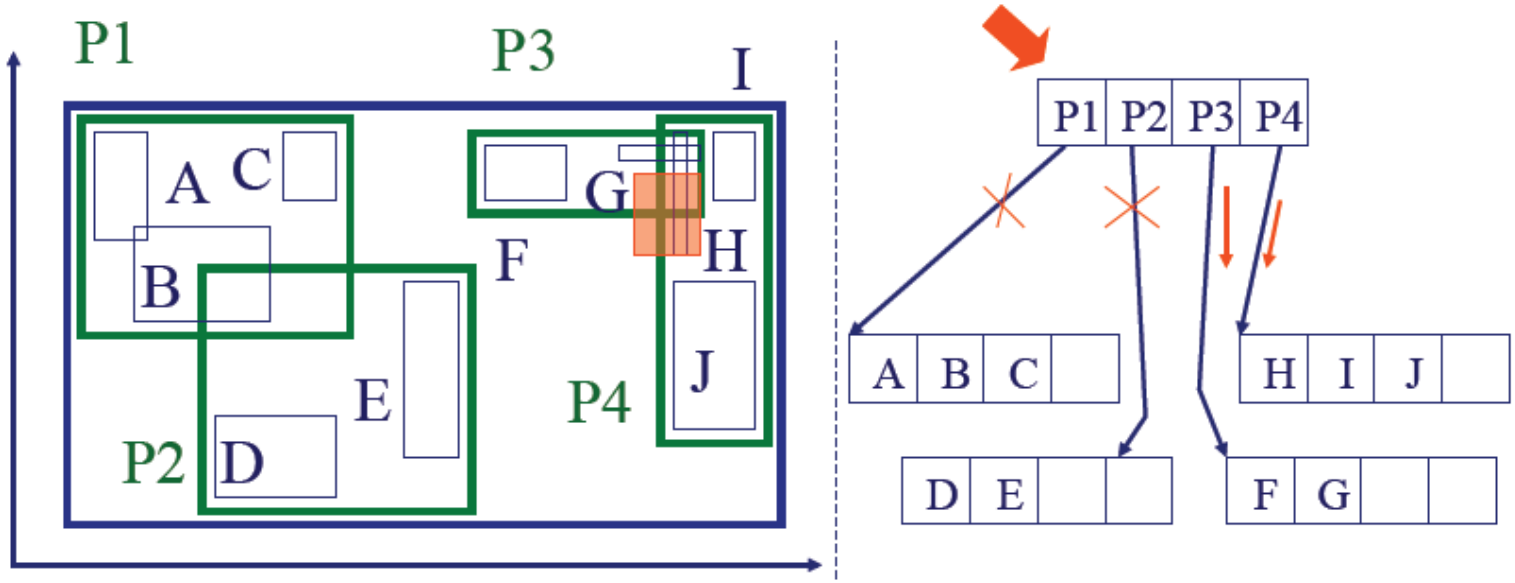




空间关系查询时，先比较包围盒，再真正空间关系比较

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.)
  - A point query may follow multiple branches.
  - Everything works for **any** dimensionality

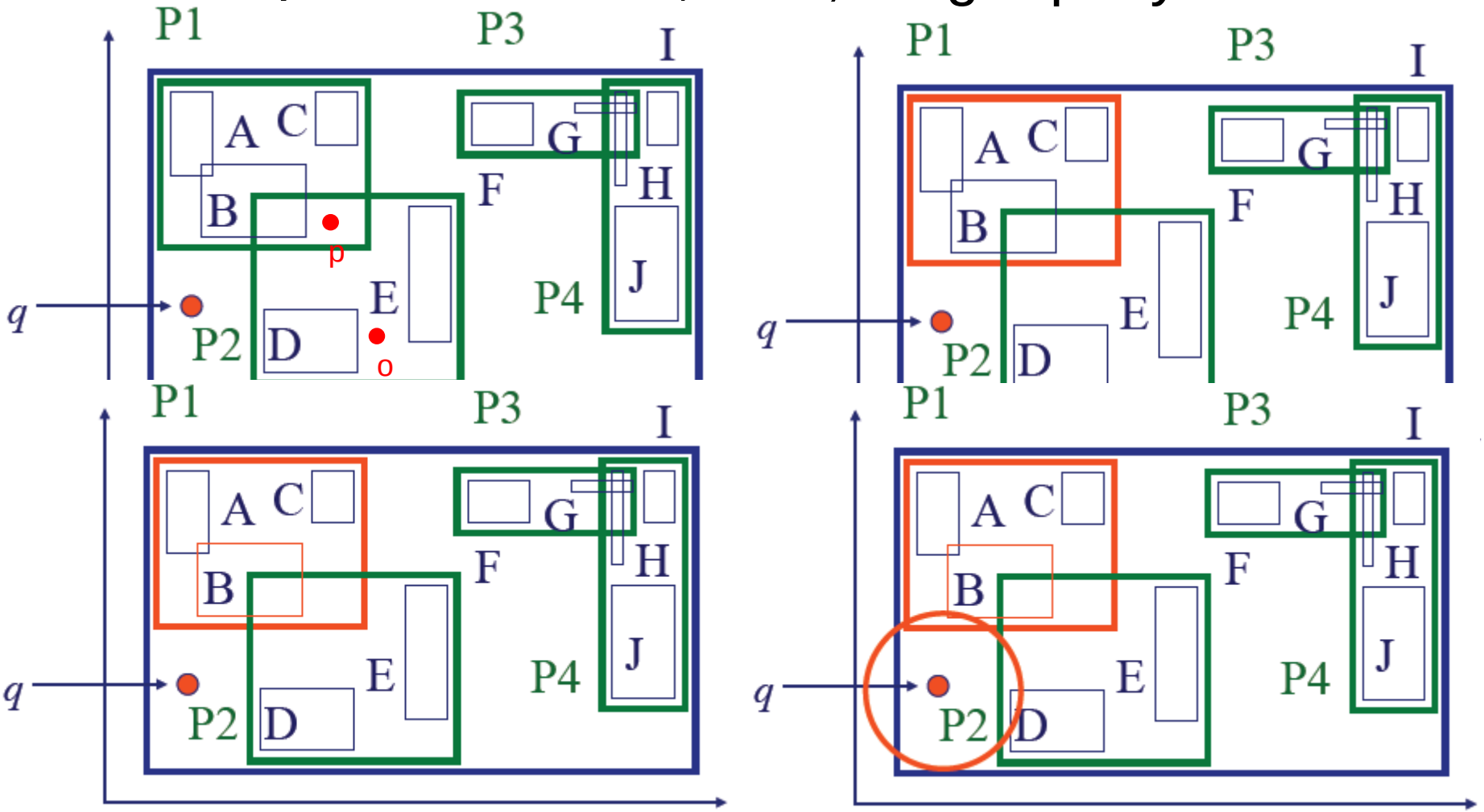


R-Trees – Range Search

- Pseudocode:
- Check the root
  - For each branch,
    - If its MBR intersects the query rectangle
      - Apply range-search, if this is a interior node
      - Print out object if its MBR intersects the query rectangle, if this is a leaf

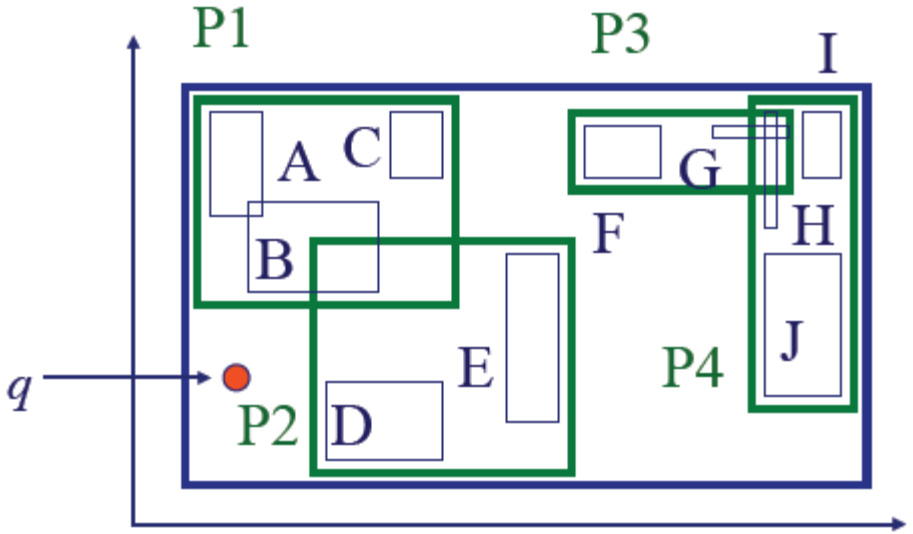
R-Trees – NN Search 思考：range query的半径合理吗？

- Q: How? (find near neighbor; refine...) o, p, q
- A1: depth-first search; then, range query



R-Trees – NN Search

- Q: How? (find near neighbor; refine...)
- A2: [Roussopoulos+, sigmod95]:
  - Priority queue, with promising MBRs, and their best and worst-case distance
- Main idea: consider only P2 and P4, for illustration

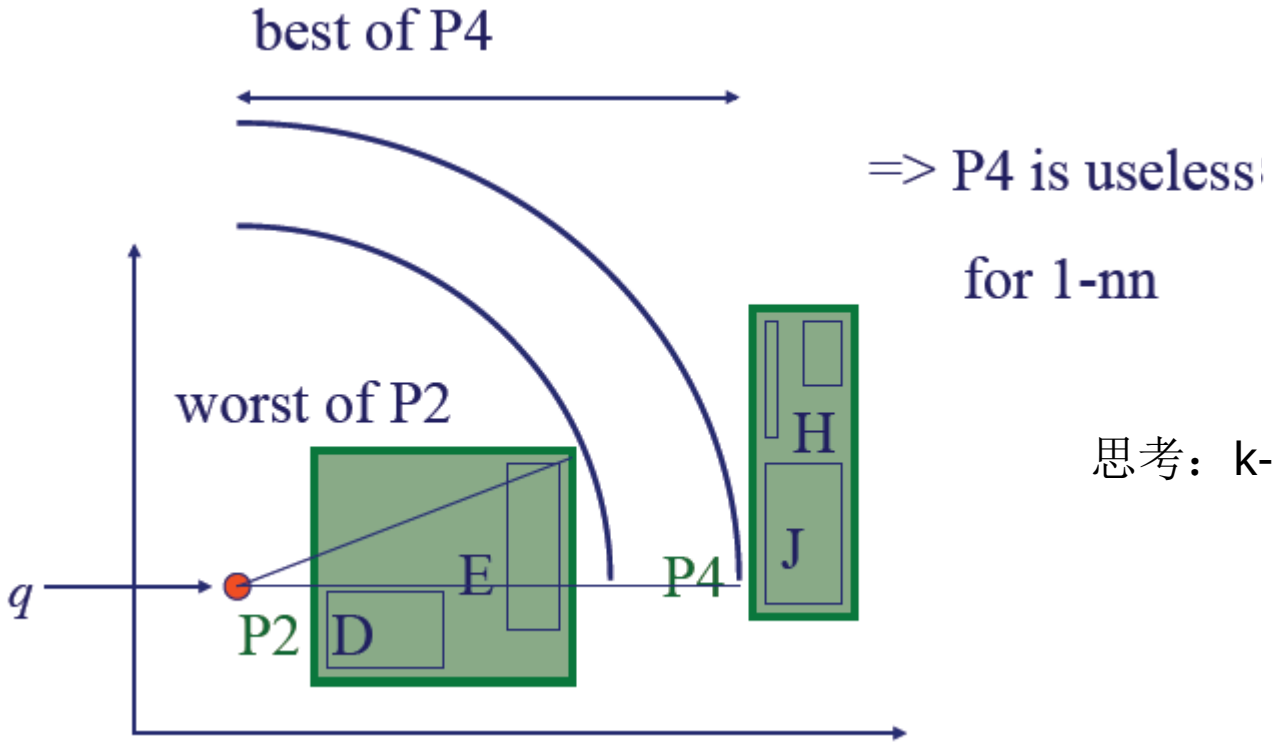


思考：q只需和4个节点判断距离，为什么？

与四叉树不同，所有叶节点都有几何要素

R-Trees – NN Search

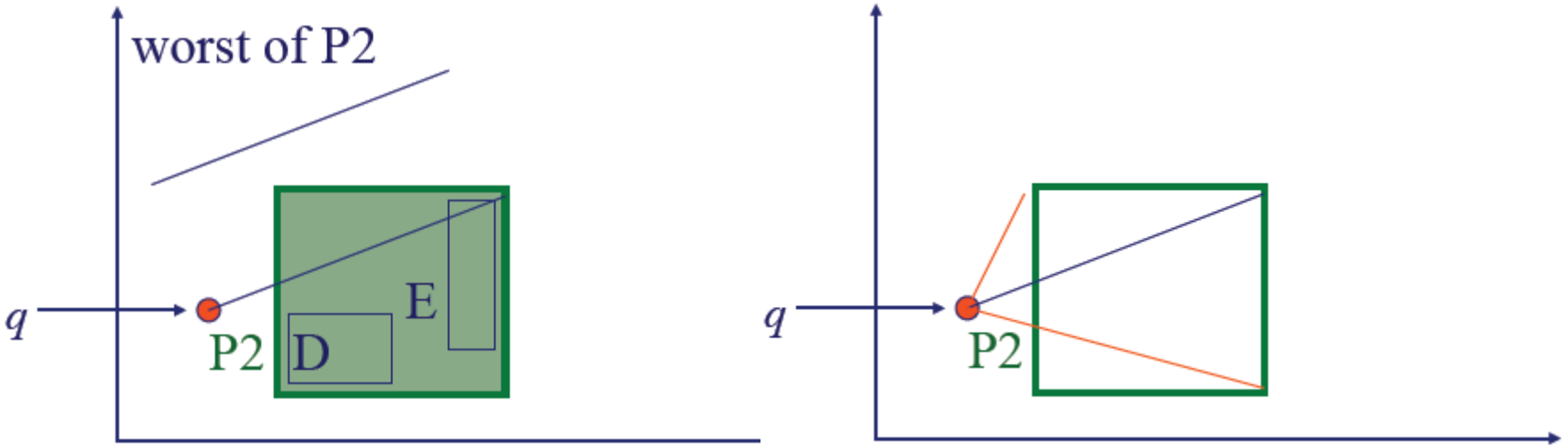
- A2: [Roussopoulos+, sigmod95]:
  - Priority queue, with promising MBRs, and their best and worst-case distance



思考：k-nn查询扩展？

R-Trees – NN Search

- Q: what is really the worst of, say, P2?
- A: the smallest of the two red segments!



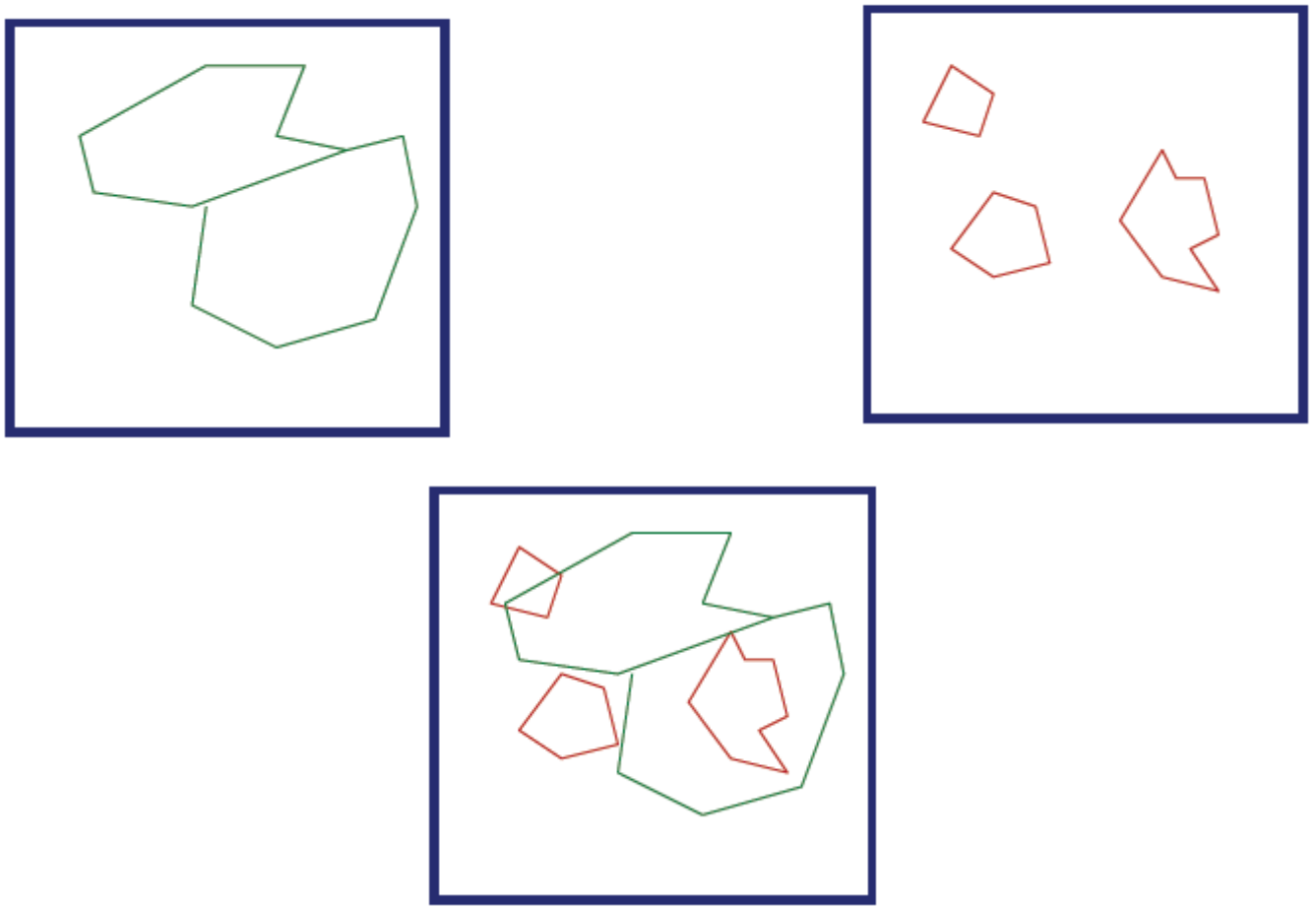
R-Trees – NN Search

- Variations: [Hjaltason & Samet] incremental nn:
  - Build a priority queue
  - Scan enough of the tree, to make sure you have the *k* nn
  - Find the (*k+1)*-th, check the queue, and scan some more of the tree
- ‘optimal’ (but, may need too much memory)

R-Trees – Spatial Joins

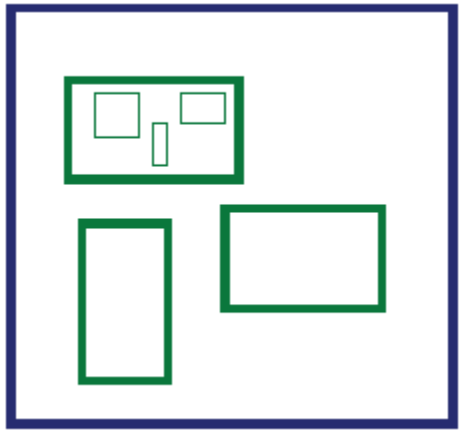
- Spatial joins: find (quickly) all

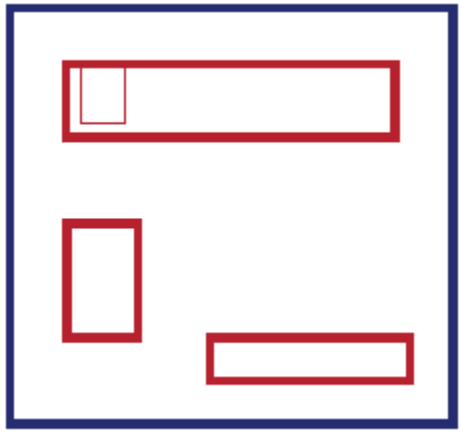
counties intersecting lakes

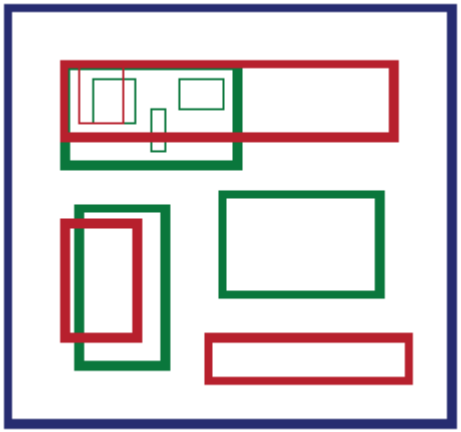


R-Trees – Spatial Joins

- Assume that they are both organized in R-trees:







R-Trees – Spatial Joins

- 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 Rtrees: [Koudas+ Sevcik], e.t.c.)

R-Trees – Performance Analysis

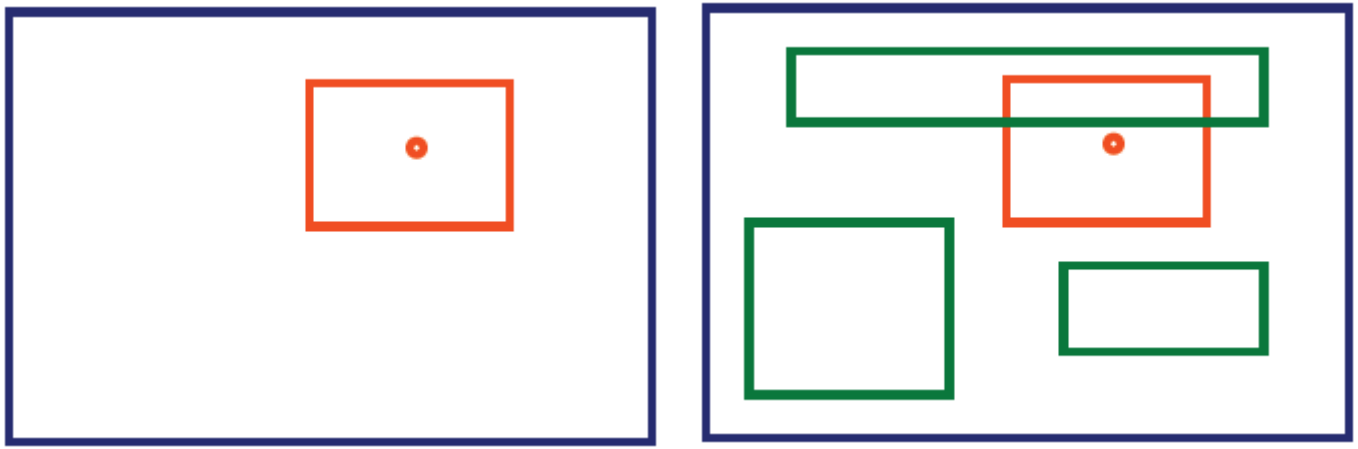
- How many disk (=node) accesses we'll need for
  - range
  - nn
  - spatial joins
- Q: Why does it matter?
- A: because we can design split etc algorithms accordingly; also, do query optimization

注意：Performance analysis和Variations为自学内容

- Motivating question: on, e.g., split, should we try to minimize the area (volume)? The perimeter? the overlap? or a weighted combination? why?

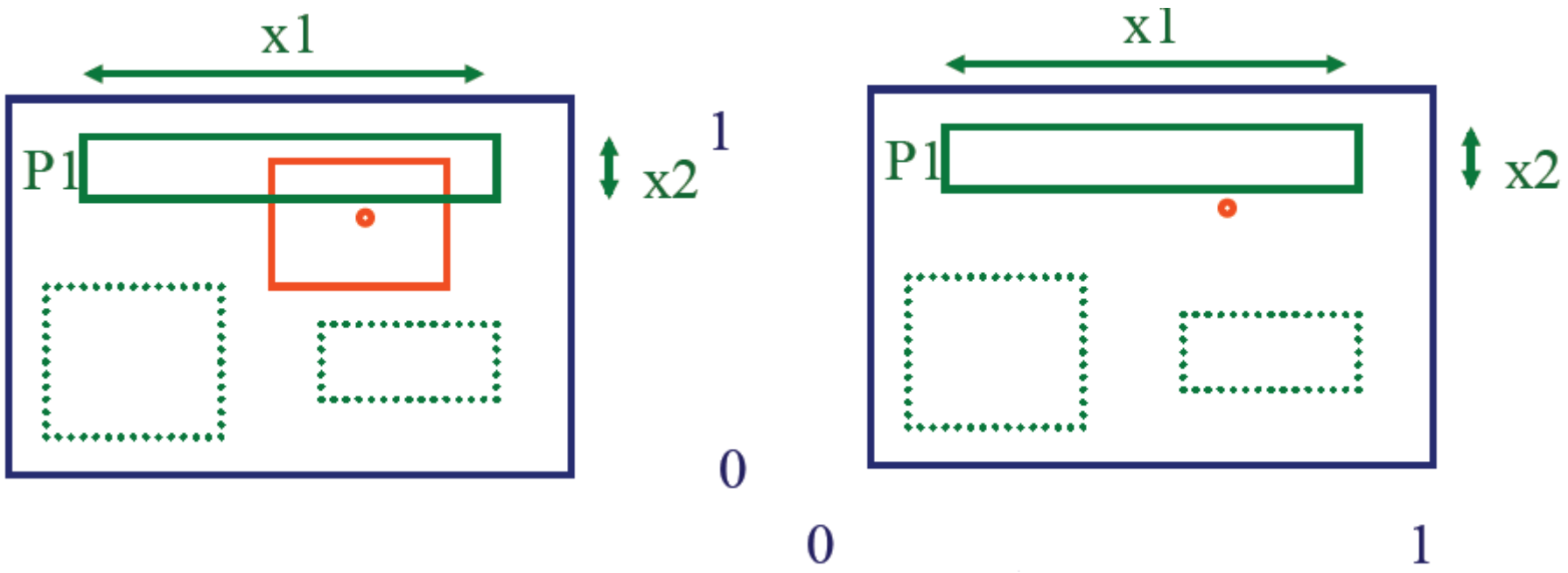
R-Trees – Performance Analysis

- How many disk accesses for range queries?
  - Query distribution wrt location? uniform; (biased)
- Easier case: we know the positions of parent MBRs, eg:



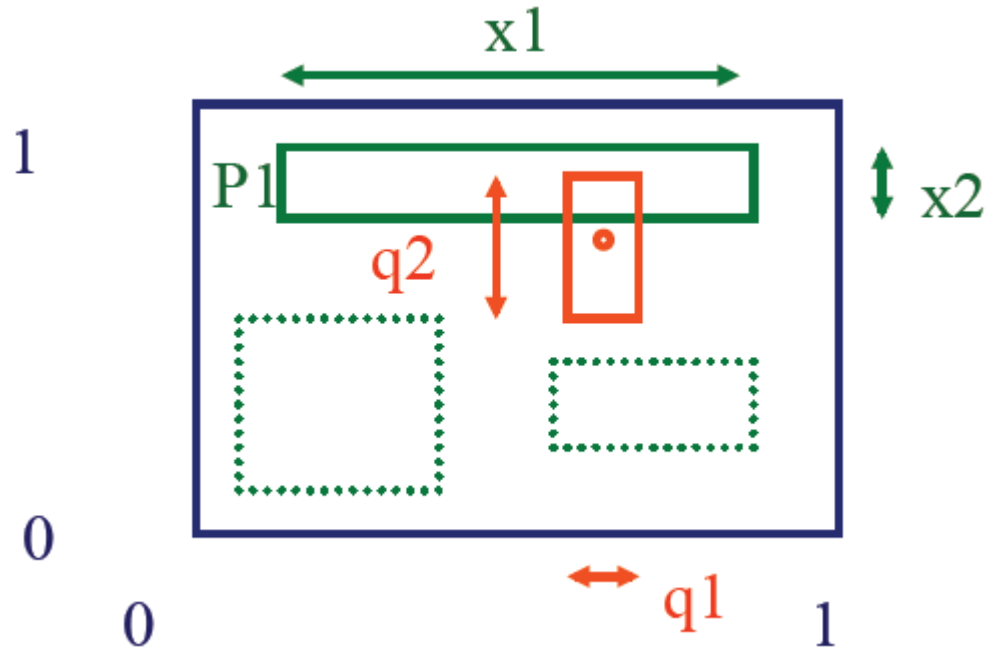
R-Trees – Performance Analysis

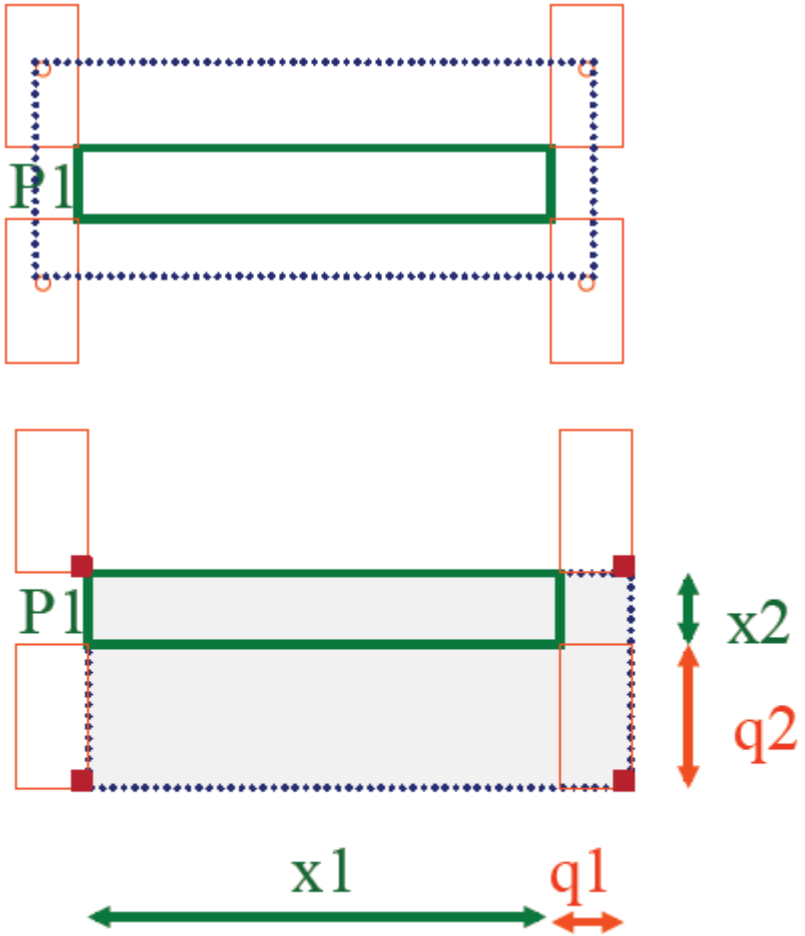
- How many times will P1 be retrieved (unif. queries)?
  - A: x1*x2



R-Trees – Performance Analysis

- How many times will P1 be retrieved (unif. queries of size q1xq2)?
  - A: (x1+q1)*(x2+q2)





R-Trees – Performance Analysis

- Thus, given a tree with N nodes (i=1, ... N) we expect
- #DiskAccesses(q1,q2) =

  sum (xi,1 + q1) * (xi,2 + q2)
- = sum (xi, 1 * xi, 2) + // volume

  q2 * sum (xi, 1) + // surface area

  q1 * sum (xi, 2) // surface area

  q1 * q2 * N // count

R-Trees – Performance Analysis

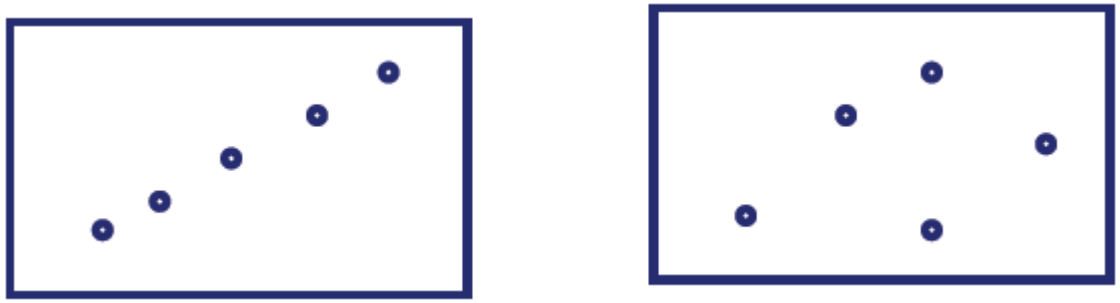
- Observations:
  - For point queries: only volume matters
  - For horizontal-line queries: (q2=0): vertical length matters
  - Ror large queries (q1, q2 >> 0): the count N matters
  - Overlap: does not seem to matter
  - Formula: easily extendible to *n* dimensions
  - (for even more details: [Pagel +, PODS93], [Kamel+, CIKM93])

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).

R-Trees – Performance Analysis

- Range queries - how many disk accesses, if we just now that we have
  - *N* points in *n*-d space?
- A: can not tell! need to know distribution
- What are obvious and/or realistic distributions?
- A: uniform
- A: Gaussian / mixture of Gaussians
- A: self-similar / fractal. Fractal dimension ~ intrinsic dimension



R-Trees – Performance Analysis

- Formulas for range queries and k-nn queries: use fractal dimension [Kamel+, PODS94], [Korn+, ICDE2000] [Kriegel+, PODS97]

R-Trees – Variations

- Guttman's R-trees sparked much follow-up work
  - Can we do better splits?
    - i.e, defer splits?
  - What about static datasets (no ins/del/upd)?
  - What about other bounding shapes?

R-Trees – Variations

- A: R*-trees [Beckmann+, SIGMOD90]
  - Defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
  - Which ones to re-insert?
  - How many? A: 30%
- Q: Other ways to defer splits?
- A: Push a few keys to the closest sibling node (closest = ??)

R-Trees – Variations

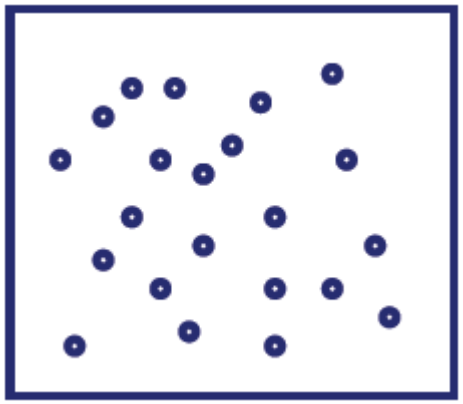
- R*-trees: Also try to minimize area AND perimeter, in their split.
- Performance: higher space utilization; faster than plain R-trees. One of the most successful R-tree variants

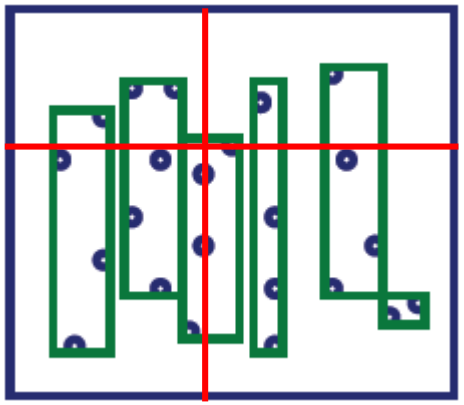
R-Trees – Variations

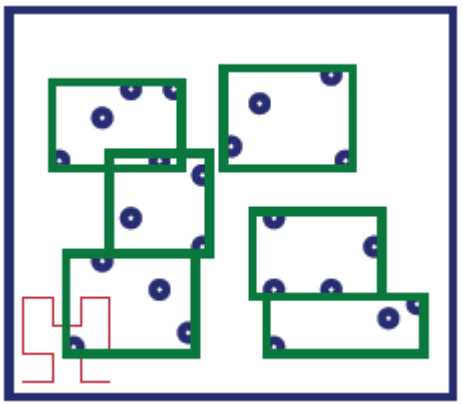
- Guttman's R-trees sparked much follow-up work
  - Can we do better splits?
    - i.e, defer splits?
  - What about static datasets (no ins/del/upd)?
    - Hilbert R-trees
  - What about other bounding shapes?

R-Trees – Variations

- What about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep great for queries on ‘x’; terrible for ‘y’
- A: plane-sweep on HILBERT curve!





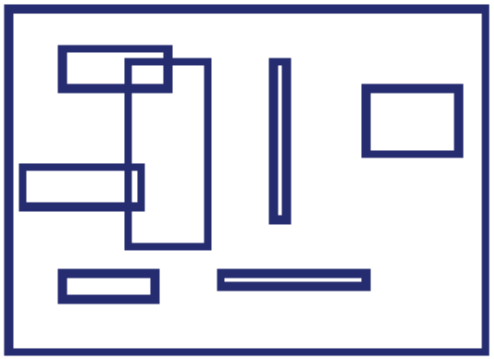


R-Trees – Variations

- Guttman's R-trees sparked much follow-up work
  - Can we do better splits?
    - i.e, defer splits?
  - What about static datasets (no ins/del/upd)?
    - Hilbert R-trees
    - Handling regions
    - Performance/discusion
  - What about other bounding shapes?

R-Trees – Variations

- What if we have regions, instead of points?
- i.e., how to impose a linear ordering (‘hvalue’) on rectangles?
- A1: h-value of center
  - With h-values, we can have deferred splits, 2-to-3 splits (3-to-4, etc)
  - Experimentally: faster than R*-trees (reference: [Kamel Faloutsos vldb 94])
- A2: h-value of 4-d point (center, x-radius, y-radius)
- A3: ...

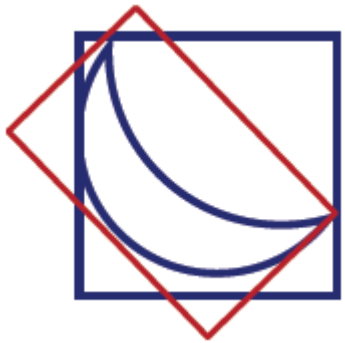


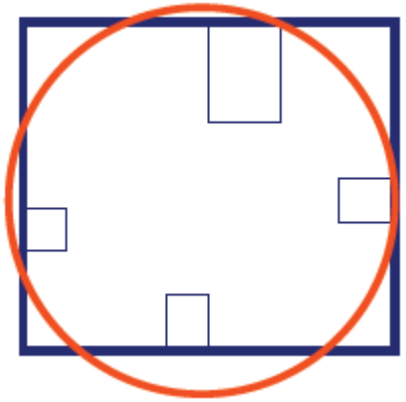
R-Trees – Variations

- Guttman's R-trees sparked much follow-up work
  - Can we do better splits?
    - i.e, defer splits?
  - What about static datasets (no ins/del/upd)?
    - Hilbert R-trees
    - Handling regions
    - Performance/discusion
  - What about other bounding shapes?

R-Trees – Variations

- What about other bounding shapes? (and why?)
- A1: arbitrary-orientation lines (cell-tree) [Guenther]
- A2: P-trees (polygon trees) (MB polygon: 0, 90, 45, 135 degree lines)
- A3: L-shapes; holes (hB-tree)
- A4: TV-trees [Lin+, VLDB-Journal 1994]
- A5: SR-trees [Katayama+, SIGMOD97] (used
- in Informedia)





R-Trees – Conclusions

- Popular method; like multi-d B-trees
- Guaranteed utilization; fast search (low dim's)
- Used in practice:
  - Oracle spatial (R-tree default; z-curve, too)
    docs.oracle.com/html/A88805_01/sdo_intr.htm
  - IBM-DB2 spatial extender
  - Postgres: create index … using [rtree | gist]
  - Sqlite3: www.sqlite.org/rtree.html
- R* variation is popular

R-Trees – 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.

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

Ibrahim Kamel and Christos Faloutsos, *Hilbert R-tree: An improved R-tree using fractals* VLDB, Santiago, Chile, Sept. 12-15, 1994, pp. 500-509.

Lin, K.-I., H. V. Jagadish, et al. (Oct. 1994). “The TV-tree An Index Structure for High-dimensional Data.” VLDB Journal 3: 517-542.

Pagel, B., H. Six, et al. (May 1993). *Towards an Analysis of Range Query Performance*. Proc. of ACM SIGACTSIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Washington, D.C.

Robinson, J. T. (1981). The k-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proc. ACM SIGMOD.

Roussopoulos, N., S. Kelley, et al. (May 1995). Nearest Neighbor Queries. Proc. of ACM-SIGMOD, San Jose, CA.

R-Trees – Other Resources

- Code, papers, datasets etc:

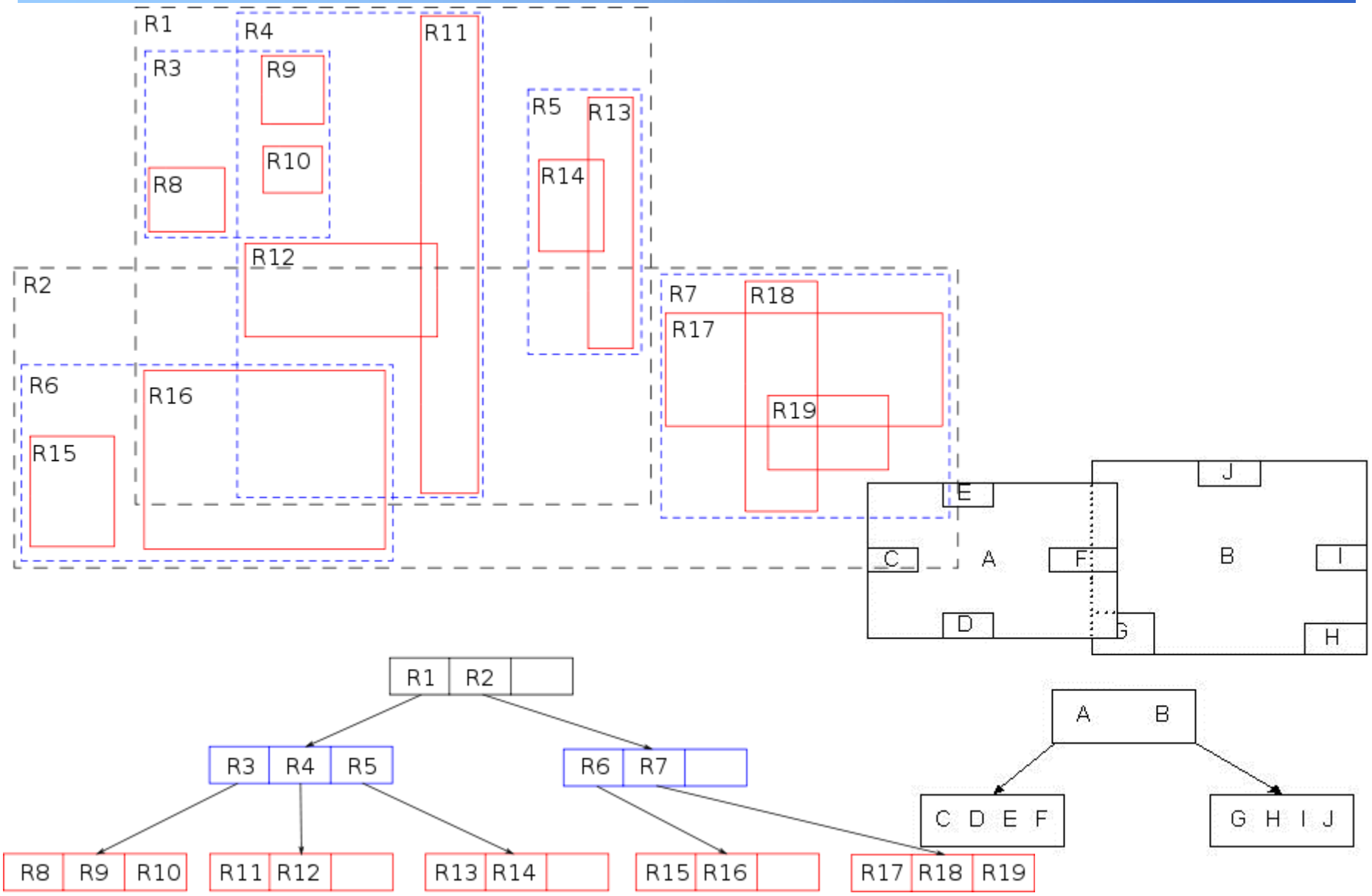
  www.rtreeportal.org/
- Java applets and more info:

  donar.umiacs.umd.edu/quadtree/points/rtrees.html

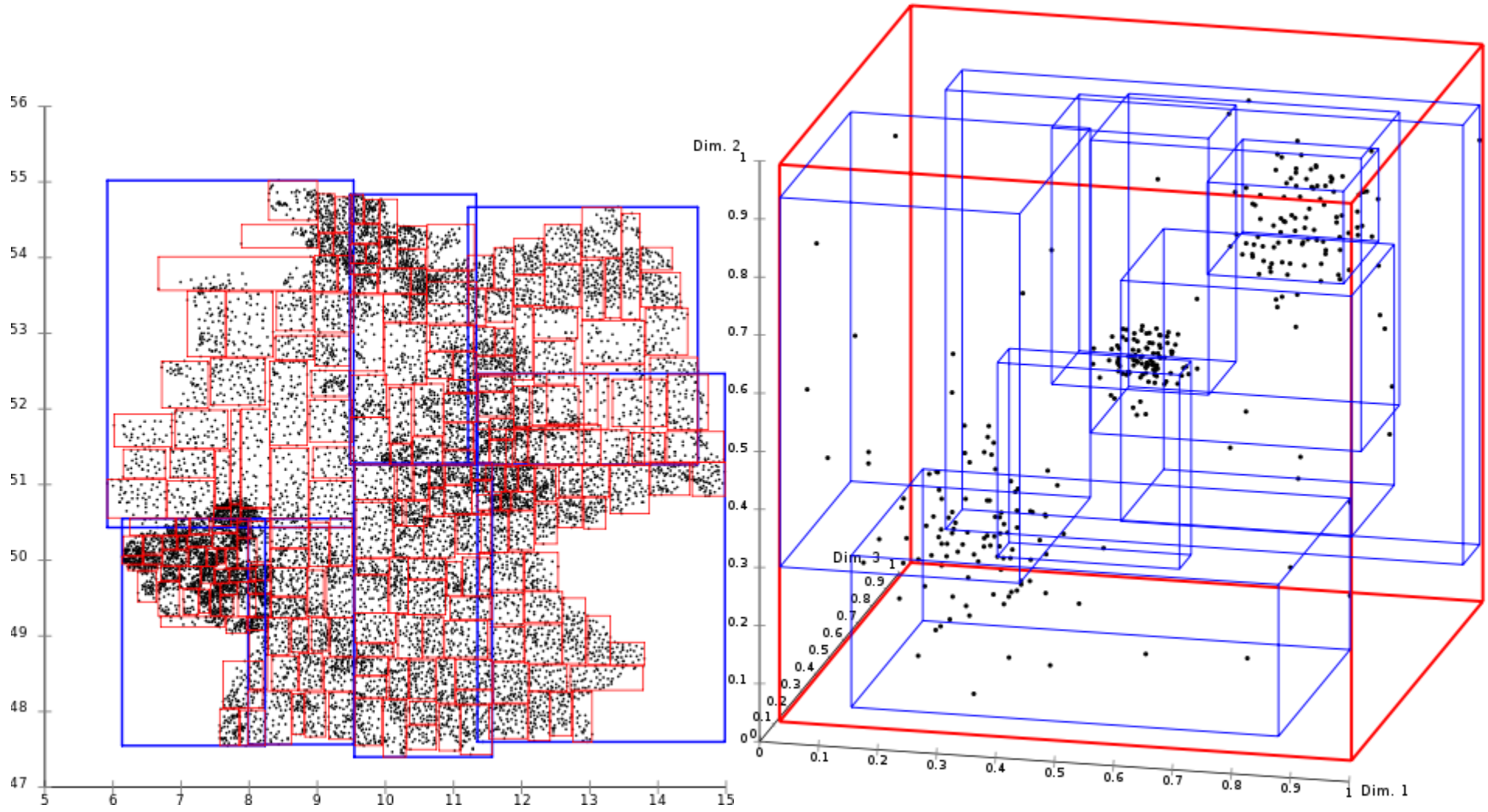
R树索引(补充)

- R树索引是最早支持扩展对象存取的方法之一，R树是一个高度平衡树，是B树在k维空间上的自然扩展
- R树用空间对象的最小边界矩形（MBR）来逼近其几何形状，采用空间聚集的方式把相邻近的空间实体划分到一起，组成更高一级的节点
- 在更高一级又根据这些节点的最小外包进行聚集，划分形成更高一级的节点，直到所有的实体组成一个根节点

R树索引(补充)


R1
R4
R11
R3
R9
R5
R13
R10
R14
R8
R12
R2
R7
R18
R17
R6
R16
R19
R15
E
J
C
A
F
B
I
D
H
R1 R2
R3 R4 R5
R6 R7
R8 R9 R10
R11 R12
R13 R14
R15 R16
R17 R18 R19
A B
C D E F
G H I J

R树索引(补充)


56
55
54
53
52
51
50
49
48
47
5
6
7
8
9
10
11
12
13
14
15
Dim. 2
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Dim. 3
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Dim. 1

R树索引(补充)

- R树特点：
  - 除根节点外，每个叶结点包含m~M条索引记录（其中 $m \leq M/2$）
  - 每个叶结点上记录了空间对象的MBR和元组标识符
  - 除根结点外，每个中间结点至多有M个子结点，至少有m个子结点
  - 每个非叶结点记录了MBR，子结点指针，其MBR为空间上包含其子结点中矩形的最小外包矩形
  - 若根节点不是叶结点，则至少包含2个子结点
  - 所有叶结点出现在同一层中
  - 所有MBR的边与一个全局坐标系的坐标轴平行

R树索引(补充)

- 查询空间对象
  - 从根节点开始，判断根节点的MBR与查询区域是否相交，若相交则遍历其子结点，否则停止
  - 在遍历子结点时，若子结点为非叶结点，则重复上述操作；若是叶子结点，则检查其MBR与查询区域是否相交
  - 若相交，则将其视为查询候选集，再根据元祖标识符提取其精确的几何信息，进行精炼步的运算

R树索引(补充)

- R树优缺点
  - 提高了空间分区节点的利用效率；降低了树的深度，提高了检索效率
- 非叶结点的MBR允许重叠，会导致同一空间查询出现多条查询路径的情况，构建一颗高效的R树：
  - 非叶结点MBR的面积尽可能小，其中不被其下级结点覆盖的面积尽可能的小
  - 各非叶结点MBR的重叠尽可能小
  - 非叶结点MBR的周长尽可能小
  - 尽可能提高每个结点的子结点的数目

R树索引(补充)

- R^+树采用对象分割技术，避免了兄弟结点的重叠，但同时也带了其他问题，如冗余存储，在构造过程中，节点MBR的增大会引起向上和向下的分裂，导致一系列复杂的连锁更新操作
- R*根据一系列结点分裂优化准则，设计了结点强制重插技术。提高了R树的空间利用率，减少了结点分裂次数，改善了树结构，但同时也增加了CPU的计算代价。
- Hilbert R树利用Hilbert分形曲线对k维空间数据进行一维线性排序，进而对树结点进行排序，以获得面积、周长最小化的树结点，以提高结点存储利用率，优化R树结构。
- Compact R树由于其特殊的分裂算法，该变体可以达到几乎100%的存储效率，并导致结点分裂的次数明显减少，易于实现和维护，但是检索性能仅与R树相仿。
- cR树，采用通用的“k-means”聚类算法，把传统的两路分裂改进为由聚类技术支持的多路分裂。适合数据密集型环境且实现算法简单，易于维护。
- 球树(sphere trees)，CP树，cell树，P树，DR树，位图R树……

四叉树与R树索引比较(补充)

- 四叉树与R树索引比较[Kothrui et al. 2002]
  - 对于10英里半径的查询窗口而言，R树性能优于四叉树2-3倍
  - 随着查询窗口的增加，两个索引性能的差异减少
  - 对于特定的查询，比如外包框相交，四叉树优于更好滴近似效果使得性能更优
  - 对于距离查询，R树优于四叉树3倍
  - 对于更新，R树基本是线性的，四叉树取决于几何对象的大小及复杂度
  - 对于点数据和大多数其他的数据，两者存储开销基本相同

四叉树与R树索引比较(补充)

- Quadtree和网格索引一般用于基于简单多边形几何对象，更新密集，高并发的或者特定的（如touch运算）查询应用中。而且，对于Quadtree来说，用户需要对块的分级进行调整以获取最好的查询性能。R树一般是自调整的，而且一般情况下具有较优的性能，比较适合用于内嵌式数据库中加速空间数据的检索性能

第五章 空间存储与索引

- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

PostGIS中的索引

- PostgreSQL支持3种索引
  - B-Tree [1维]
    - 数值、字符串、日期
  - R-Tree
    - Break data into rectangles, sub-rectangles, sub-sub-rectangles……
    - PostgreSQL R-Tree实现没有GiST实现得鲁棒
  - GiST (Generalized Search Trees)
    - Break data into “things to one side”, “things which overlap”, “things which are inside”
    - PostGIS uses an R-Tree index implemented on top of GiST

PostGIS中的索引

- 创建GiST索引(2D-index)

Create Index [indexname] On [tablename] Using **GiST** ([geometryfield])

- 创建GiST索引 (n-dimensional, PostGIS 2.0以上)

Create Index [indexname] On [tablename] Using **GiST** ([geometryfield] gist_geometry_ops_nd)

- 创建空间索引非常费时
  - 100万行的几何表，在300MHz Solaris机器上，创建GIST索引需要大约1小时

SQL 窗口

```
-- Index: public.ir

-- DROP INDEX public.ir;

CREATE INDEX ir
  ON public.r
  USING btree
  (a);
```

与属性索引比较

http://postgis.net/docs/using_postgis_dbmanagement.html#idp69832064

PostGIS中的索引

- 在创建索引之后，通知PostgreSQL收集表的统计数据，以便优化查询规划

Vacumm Analyze [tablename] [(columnname)]

-- This is only needed for PostgreSQL 7.4 installations and below

Select Update_Geometry_Stats ([tablename], [columnname]

- GIST索引与R-Tree相比，优点：
  - ‘null safe’，也就是说能够索引包含null值的列
  - 支持‘lossiness’，特别是在处理GIS对象大小大于PostgreSQL 8k页表
    - Lossiness使PostgreSQL仅需在索引中存储重要信息，如包围盒

思考：为什么属性索引不用vacumm analysis？

PostGIS中的索引

- GiST索引一旦建立，查询规划器会自动决定是否利用索引来加速查询，但是PostgreSQL查询规划器并没有优先使用GiST索引
  - 所以有时可以用空间索引，但仍然扫描全表
- 解决方法
  - Make sure statistics are gathered about the number and distributions of values in a table, to provide the query planner with better information to make decisions around index usage [Vaccum Analyze]

PostGIS中的索引

- 解决方法
  - Vacuum Analyze
  - If vacuuming does not work, you can force the planner to use the index information by using the SET ENABLE_SEQSCAN=OFF command
    - You should only use this command sparingly, and only on spatially indexed queries: generally speaking, the planner knows better than you do about when to use normal B-Tree indexes. Once you have run your query, you should consider setting ENABLE_SEQSCAN back on, so that other queries will utilize the planner as normal

PostGIS中的索引

- 解决方法
  - Vacuum Analyze
  - Set Enable_Seqscan = OFF
  - If you find the planner wrong about the cost of sequential vs index scans try reducing the value of random_page_cost in postgresql.conf or using SET random_page_cost=#. Default value for the parameter is 4, try setting it to 1 or 2. Decrementing the value makes the planner more inclined of using Index scans

PostGIS中的索引

- explain select C.name, count(*)
- from ne_10m_admin_0_countries C, ne_10m_populated_places P
- where ST_Within(P.geom, C.geom) group by C.name order by C.name

输出窗口

数据输出 | 解释 | 消息 | 历史

	QUERY PLAN text
1	Sort (cost=497105.20..497105.84 rows=255 width=10)
2	Sort Key: c.name
3	-> HashAggregate (cost=497092.46..497095.01 rows=255 width=10)
4	Group Key: c.name
5	-> Nested Loop (cost=0.00..496886.56 rows=41179 width=10)
6	Join Filter: ((p.geom && c.geom) AND st_contains(c.geom, p.geom))
7	-> Seq Scan on ne_10m_admin_0_countries c (cost=0.00..72.55 rows=255 width=34734)
8	-> Materialize (cost=0.00..629.15 rows=7343 width=32)
9	-> Seq Scan on ne_10m_populated_places p (cost=0.00..592.43 rows=7343 width=32)

输出窗口

数据输出 | 解释 | 消息 | 历史

	QUERY PLAN text
1	Sort (cost=1411.56..1412.20 rows=255 width=10)
2	Sort Key: ne_10m_admin_0_countries.name_long
3	-> HashAggregate (cost=1398.82..1401.37 rows=255 width=10)
4	Group Key: ne_10m_admin_0_countries.name_long
5	-> Nested Loop (cost=0.15..1192.93 rows=41179 width=10)
6	-> Seq Scan on ne_10m_admin_0_countries (cost=0.00..72.55 rows=255 width=34734)
7	-> Index Scan using icity on ne_10m_populated_places (cost=0.15..4.38 rows=1 width=32)
8	Index Cond: (geom && ne_10m_admin_0_countries.geom)
9	Filter: st_contains(ne_10m_admin_0_countries.geom, geom)

哪些空间函数能够利用空间索引加速

- 空间索引加速
  - 空间函数通常计算量大，扫描全表两两计算比较耗时
  - 先用Envelope判断两个几何是否有关系，如果有，再进行空间函数
- A. 常规方法(12种)：Dimension, CordinateDimension, GeometryType, SRID, Envelope, AsText, AsBinary, isEmpty, isSimple, is3D, IsMeasured, Boundary
  - 单个几何，不能利用索引
- B. 常规GIS分析方法(7种)：Distance, Buffer, ConvexHull, Intersection, Union, Difference, SymDifference
  - 两个几何计算，需要获得精确结果，比如交集

哪些空间函数能够利用空间索引加速

- 空间索引加速
  - 空间函数通常计算量大，扫描全表两两计算比较耗时
  - 先用Envelope判断两个几何是否有关系，如果有，再进行空间函数
- C. 空间查询方法(8种)：Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps
  - 两个几何关系判断，仅需返回True/False
  - 可以利用索引快速判断获得False
    - 如果Envelope不相交，那两个几何也肯定不相交，返回False
  - Disjoint和Relate在当前PostGIS实现中不会使用索引
  - Implicit bounding box overlap operators
  - http://postgis.net/docs/using_postgis_dbmanagement.html#gist_indexes

哪些空间函数能够利用空间索引加速

- ST_Distance不能利用空间索引
  - Select ST_Distance(A.geom, B.geom) From A, B
    - 两个几何的精确计算
  - Select A.name, B.name From A, B Where ST_Distance(A.geom, B.geom) < 10
    - 仅用ST_Distance判断两个几何关系
    - 何时可以改用ST_DWithin加速？
- ST_Disjoint不能利用空间索引
  - 如何利用其它空间函数进行加速？
- Lecture4 4.5.8 几何操作符
  - 操作符进行空间操作的对象必须有空间索引，也就说空间操作符是与空间索引绑定的(A？B)

现有空间数据库产品的索引方式

- 现有空间数据库产品的索引方式
  - DB2 Spatial Extender提供基于网络的三层空间索引，该索引技术是基于传统的分层B树索引形成，与ArcSDE的优化网格索引类似
  - MySQL能够以创建常规索引相同的方式创建空间索引，不同之处在于将创建常规索引的SQL语句用Spatial关键词进行扩展，而删除索引完全相同

空间扩展模块	空间索引类型
Oracle Spatial	R树，Quadtree
DB2 Spatial Extender	R树，Spherical Voronoi Tessalation
Informix Spatial	R树
SQL Server Spatial	4级网格索引（基于B树实现）
MySQL Spatial	R树（Quadratic Splitting）
PostGIS	R树（基于GiST）

现有空间数据库产品的索引方式

- 现有空间数据库产品的索引方式
  - PostGIS提供基于GiST框架的空间索引R树用于空间数据的快速访问。GiST R树采用新线性结点分裂算法，与PostgreSQL内置的采用二次结点分裂算法的R树相比：空值安全；支持松散索引

空间扩展模块	空间索引类型
Oracle Spatial	R树，Quadtree
DB2 Spatial Extender	R树，Spherical Voronoi Tessalation
Informix Spatial	R树
SQL Server Spatial	4级网格索引（基于B树实现）
MySQL Spatial	R树（Quadratic Splitting）
PostGIS	R树（基于GIST）

第五章 空间存储与索引

- 5.1 物理数据模型
- 5.2 数据物理存储
- 5.3 数据文件组织
- 5.4 属性索引
- 5.5 空间索引
- 5.6 PostGIS空间索引
- 5.7 发展趋势

Trends

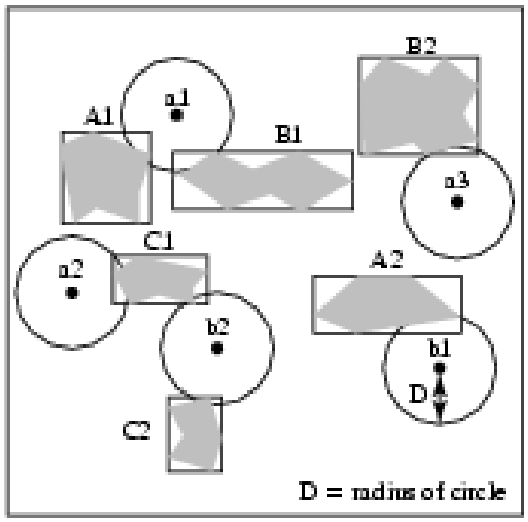
- New developments in physical model
  - Use of intra-object indexes
  - Support for multiple concurrent operations
  - Index to support spatial join operations
- Use of intra-object indexes
  - Motivation: large objects (e.g. polygon boundary of USA has 1000s of edges
  - Algorithms for OGIS operations (e.g. touch, crosses)
    - Often need to check only a few edges of the polygon
    - Relevant edges can be identified by spatial index on edges
  - Uniqueness
    - Intra-object index organizes components within a large spatial object
    - Traditional index organizes a collection of spatial objects

Trends - Concurrency Support

- Why support concurrent operations?
  - SDBMS is shared among many users and applications
  - Simultaneous requests from multiple users on a spatial table
    - Serial processing of request is not acceptable for performance
    - Concurrent updates and find can provide incorrect results
- Concurrency control idea for R-tree index
  - R-link tree: Add links to chain nodes at each level
  - Use links to ensure correct answer from find operations
  - Use locks on nodes to coordinate conflicting updates

Trends - Join Index

- Ideas
  - Spatial join is a common operation
  - Expensive to compute using traditional indexes
  - Spatial join index pre-computes and stores id-pairs of matched rows across tables
  - Speeds up computation of spatial join



(a) Spatial attribute of R and S

R Relation (Facility Location)

ID	Location (X,Y)	Non-Spatial Data
a1	(7.9, 16.7)	(...)
a2	(3.4, 11.4)	(...)
a3	(19.5, 13.1)	(...)
b1	(18.7, 6.4)	(...)
b2	(9.5, 7.1)	(...)

S Relation (Forest-Stand Boundary)

ID	MOBR (X_{LL} Y_{LL} X_{UR} Y_{UR})	Non-Spatial Data
A1	(3, 12.2, 6.6, 16)	(...)
A2	(13.4, 7.7, 19.5, 10)	(...)
B1	(7.6, 12.7, 15, 5.2)	(...)
B2	(15.5, 15, 20.4, 19)	(...)
C1	(5.1, 8.9, 9.1, 10.9)	(...)
C2	(7.5, 2.9, 7, 15.1)	(...)

Join-Index

R.ID	S.ID
a1	A1
a1	B1
a2	C1
a3	B2
b1	A2
b2	C1
b2	C2

(b) R and S relation table and join-index

空间存储与索引总结

- 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

属性索引与空间索引

- 属性索引 (非空间数据索引)
  - B+tree, Hash, …
  - create unique index Stusno ON S (Sno asc);
  - create unique index Coucno ON C (Cno asc);
  - create unique index SCno ON SC(Sno asc, Cno desc);
    - 具体数据结构取决于数据库系统实现
    - 如果了解数据属性特点，可以在创建索引时指定数据结构
  - 有些数据库系统自动为primary key创建索引，或按primary key进行sorted file存储
- 空间索引
  - create index countries_geom_idx on countries using gist(geom);
  - 能建在非空间数据属性上？

PostGIS索引

- GiST索引一旦建立，查询规划会自动决定是否利用索引来加速查询，但是PostgreSQL查询规划器并没有优化使用GiST索引
  - 所有有时可以用空间索引，但仍然扫描全表
- 解决方法
  - Vacuum Analyze
  - Set Enable_Seqscan = OFF
  - If you find the planner wrong about the cost of sequential vs index scans try reducing the value of random_page_cost in postgresql.conf or using SET random_page_cost=#.

何时选择属性索引？

- Make some attribute K a search key if the WHERE clause contains
  - An exact match on K
  - A range predicate on K
  - A join on K
  - Order by / group by
- 如何选择非空间索引？
  - 看where，group by，order by使用的属性，根据属性的特点(唯一或非唯一)和用法(=，>或<，<>)创建索引
  - 当同一个关系使用多个属性的等值判断，可以创建多属性索引加速

何时选择空间索引？

- 空间函数和空间索引 – 减少不必要的两两空间处理
  - where子句调用空间函数时，可能会使用空间索引
    - 是否真的调用，取决于查询规划器得到的全表扫描和使用索引扫描的cost值
  - select子句调用空间函数时，通常不会使用空间索引
    - select子句中所有空间操作都是必要的
  - 索引是建在基表上，通过不会在查询结果上使用空间索引
- 空间索引只能创建在空间数据，尽可能创建在静态空间数据，基于几何要素的包围盒Envelope通过overlap判断，获得需要真正几何操作的候选集(filter step)

查询规划

- 如何查看关系中已创建的索引？
  - PostgreSQL通常基于主键以sorted file进行存储，即主键本身是primary index，
  - PostGIS shapefile loader缺省选项是对geometry创建空间索引
- 学会看查询规划，DBMS基于cost模型(与数据和SQL语句有关)预测使用或不使用索引所需时间，选择cost最小的查询规划进行真正的数据查询，由于是近似的cost，查询时间不一定最短