第七章 空间存储与索引

陶煜波

计算机科学与技术学院

第六章 关系数据库设计理论回顾

数据库设计一般要经过以下几个步骤

- 需求分析阶段 → 数据流图和数据字典 (软工)
- 概念结构设计阶段 → E-R图/UML图
- 逻辑结构设计阶段 → 关系数据库模式 (规范化)
- 数据库物理设计阶段 → 存储方式、索引和用户权限
- 数据库实施阶段
- 数据库运行和维护阶段

(数据结构基础)

第六章 关系数据库设计理论回顾

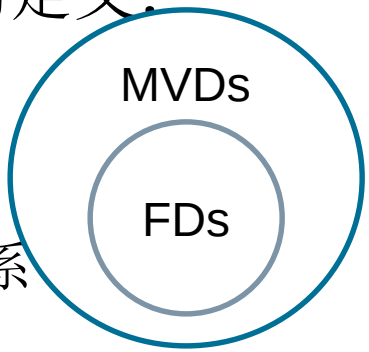
- 数据依赖
  - 关系R中，属性集A和属性集B存在函数依赖的定义：

$$\forall t, u \in R:$$
$$t[A_1, \ldots A_n] = u[A_1, \ldots A_n] => \ t[B_1, \ldots B_m] = u[B_1, \ldots B_m]$$

    - 计算属性的闭包判断是否为码 (key)
    - 完全非平凡函数依赖
    - 函数依赖规则：splitting, combining, trivial-dependency, transitive
    - 指定关系的函数依赖集合：Minimal set of completely nontrivial FDs such that all FDs that hold on the relation follow from the dependencies in the set
  - 关系R中，属性集A和属性集B存在多值依赖的定义：

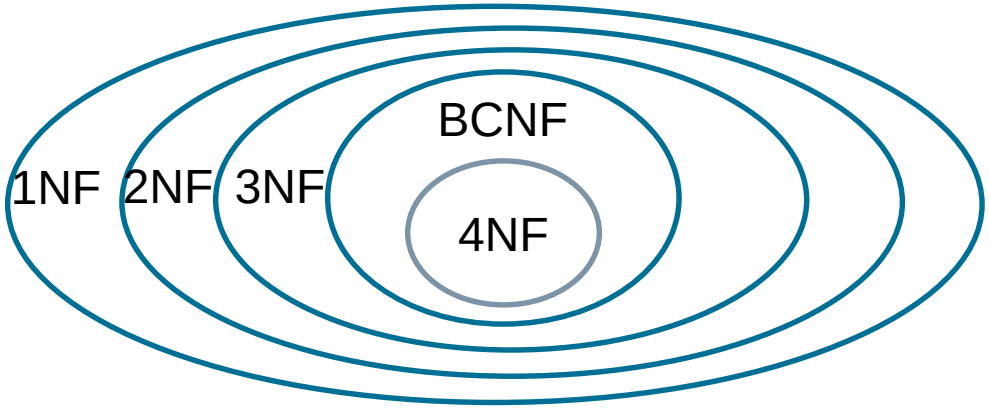
$$\forall t, u \in R: t[\bar{A}] = u[\bar{A}]\ then$$
$$\exists v \in R: v[\bar{A}] = t[\bar{A}]\ and$$
$$v[\bar{B}] = t[\bar{B}]\ and$$
$$v[rest] = u[rest]$$

    - 解决方法：E/R转换关系时，将多值属性转换为关系



第六章 关系数据库设计理论回顾

- 数据依赖 → 设计异常
  - 数据冗余
  - 更新、插入、删除异常



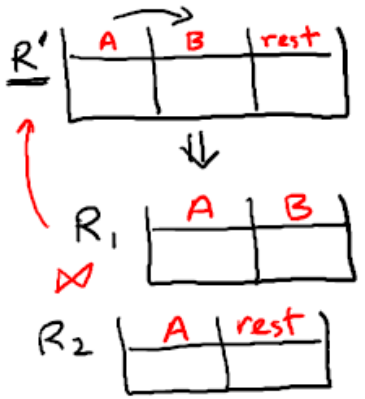
- 数据依赖 → 设计异常 → 关系分解
  - 要求：no anomalies, no lost information (lossless join property)
  - BC范式：Relation R with FDs is in BCNF if for each A→B, A is a key
  - 第4范式：Relation R with MVDs is in 4NF if for each nontrivial A→→B, A is a key
  - 设计考虑：保留函数依赖，考虑查询效率，避免过度分解
  - 空间数据的独特性：shape → other attributes(大部分情况)

第六章 关系数据库设计理论回顾

BCNF decomposition algorithm

Input: relation R + FDs for R

Output: decomposition of R into BCNF relations with "lossless join"



Compute keys for R (using FDs)

Repeat until all relations are in BCNF:

- Pick any R' with $\bar{A} \rightarrow \bar{B}$ that violates BCNF
- Decompose R' into $R_1(\bar{A}, \bar{B})$ and $R_2(\bar{A}, rest)$
- Compute FDs for R_1 and R_2
- Compute keys for R_1 and R_2

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

参考教材：
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
    - 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

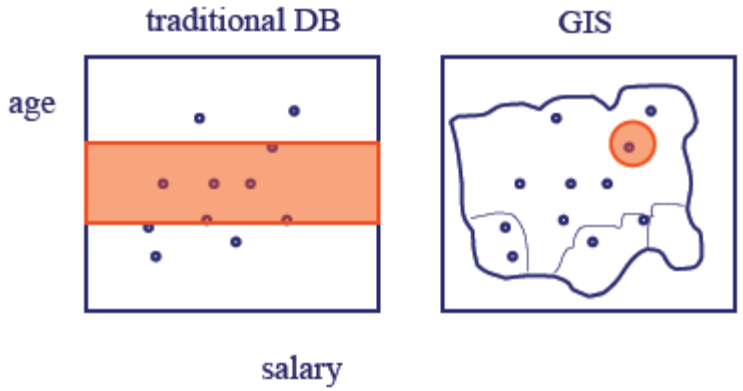
- Database concepts
  - Conceptual data model - entity, (multi-valued) attributes, relationship, …
  - Logical model - relations, atomic attributes, primary and foreign keys
  - Physical model - secondary storage hardware, file structures, indices, …
- Examples of physical model concepts from relational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

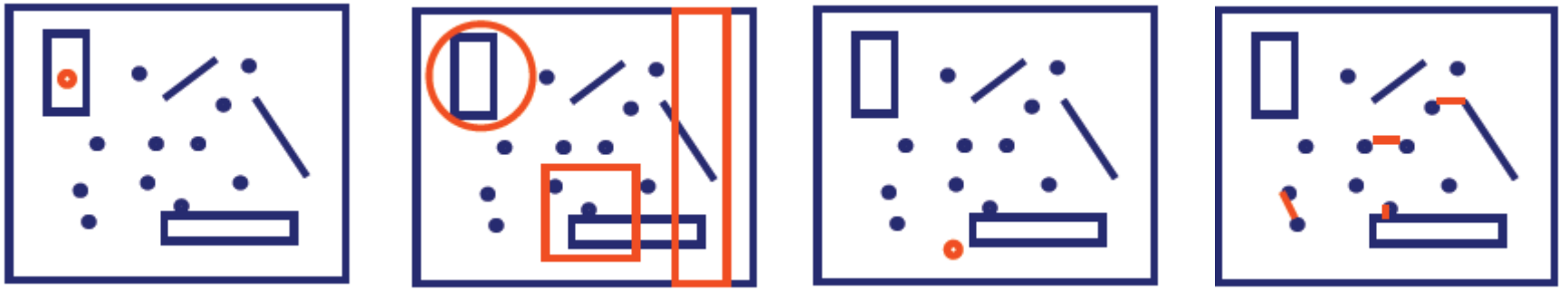
- Is relational DBMS physical data model suitable for spatial data?
- Reusing relational physical data model concepts
  - Space filling curves define a total order for points
  - This total order helps in using ordered files, search trees
  - But may lead to computational inefficiency
- New spatial techniques
  - 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

7.1 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

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

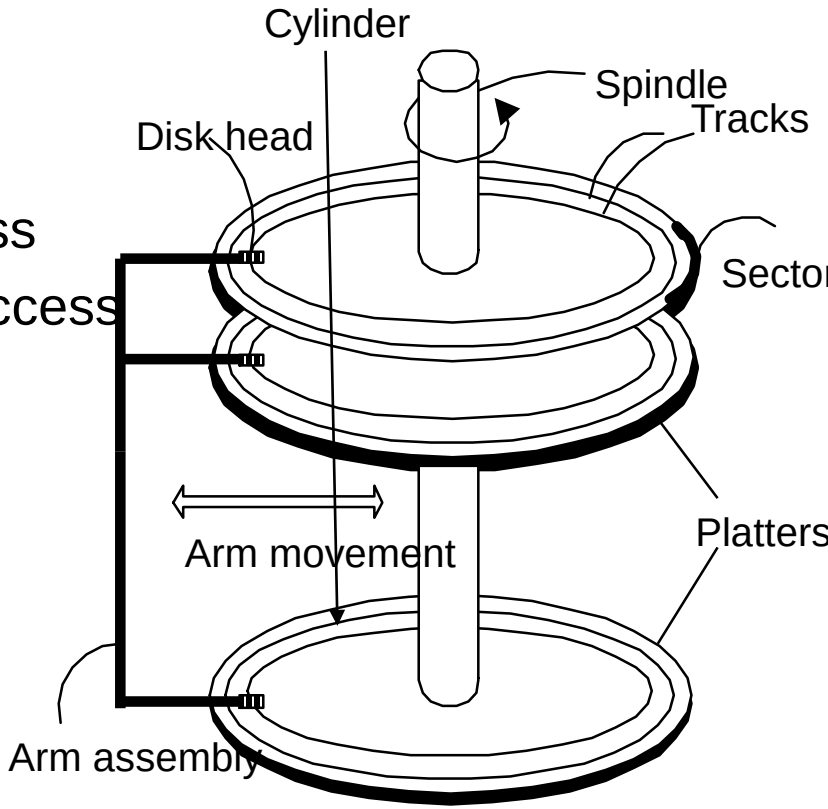
参考教材：
Spatial Databases: A Tour, Chapter 4
15-826: Multimedia Databases and Data Mining
空间数据库管理系统概论，第六章

Storage Hierarchy in Computers

- 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.

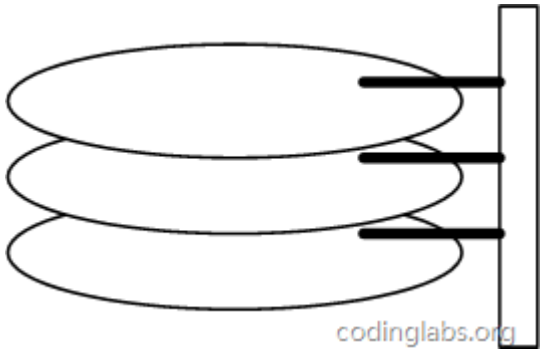
Secondary Storage Hardware: Disk Drives

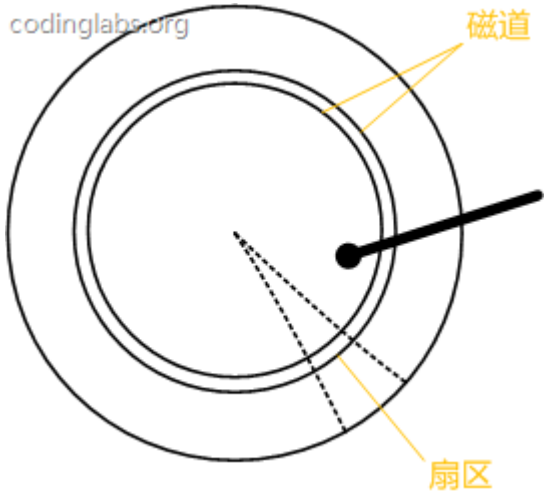
- Disk
  - Slow: Sequential access (although fast sequential reads)
  - Durable: Once on disk, data is safe
  - Cheap
- Memory
  - Fast
    - ~10x faster for sequential access
    - ~100, 000x faster for random access
  - Volatile: Data can be lost
  - Expensive



Secondary Storage Hardware: Disk Drives

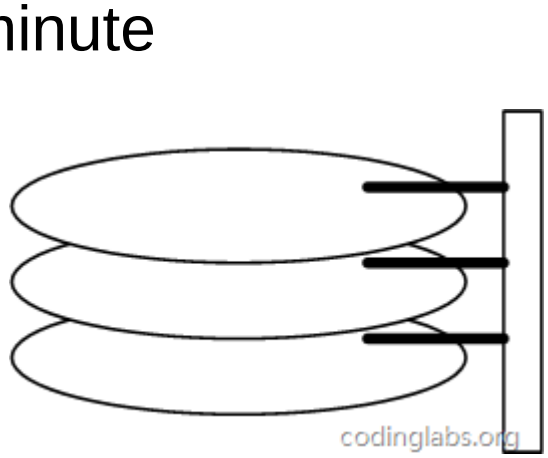
- Disk concepts
  - Multiple platters are mounted on a spindle
  - Circular platters with magnetic storage medium
  - Platters are divided into concentric tracks
  - Tracks are divided into sectors
  - A sector size may a few kilo-Bytes

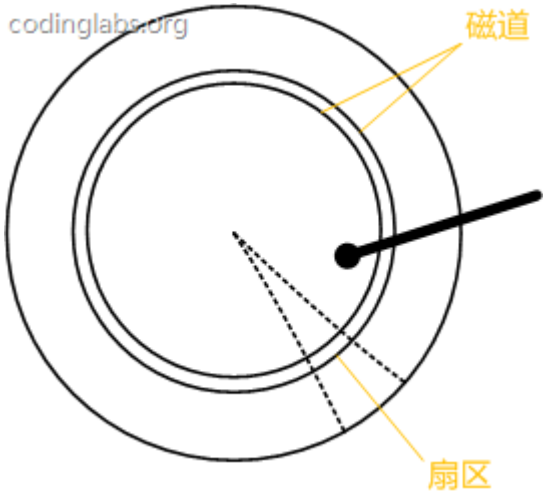




Secondary Storage Hardware: Disk Drives

- 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
  - 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 track
  - Reason: Minimize seek cost in scanning the entire data set

Using Disk Hardware Efficiently

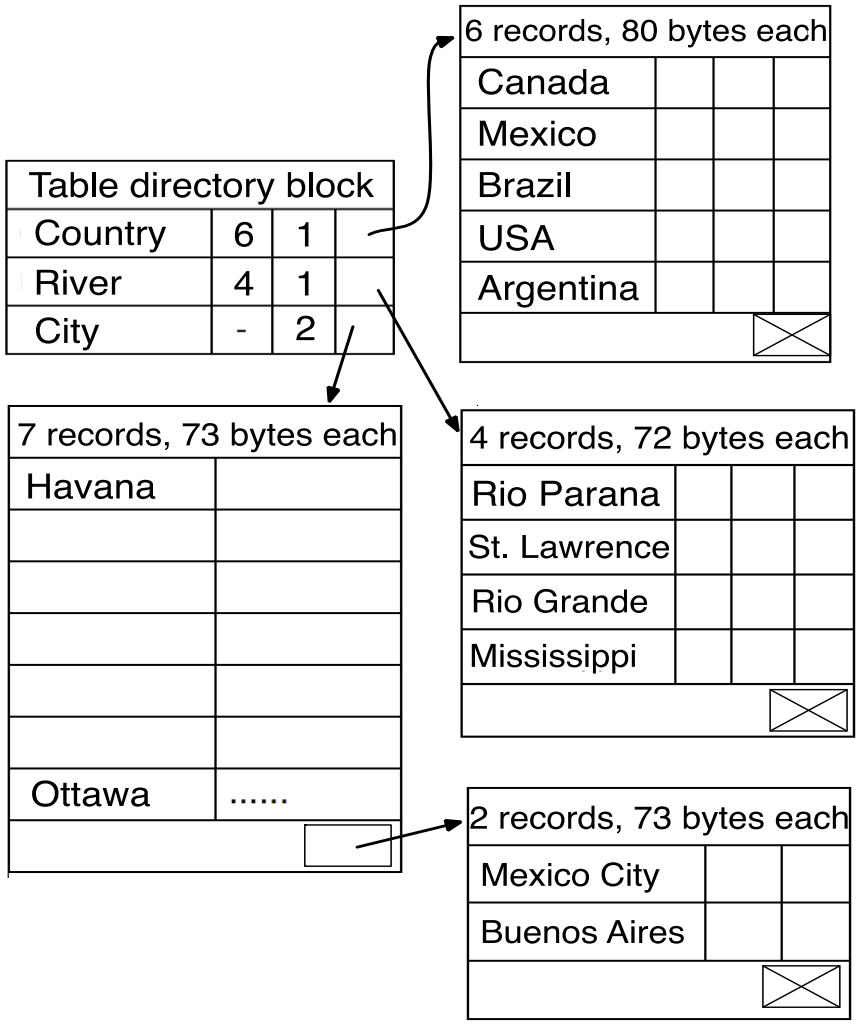
- Sequential scan is much faster than random reads
  - Good: read blocks 1, 2, 3, 4, 5, …
  - Bad: read blocks 2342, 11, 321, 9, …
- Rule of thumb
  - Random reading 1-2% of the file ≈ sequential scanning the entire file

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



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

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

参考教材：
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

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

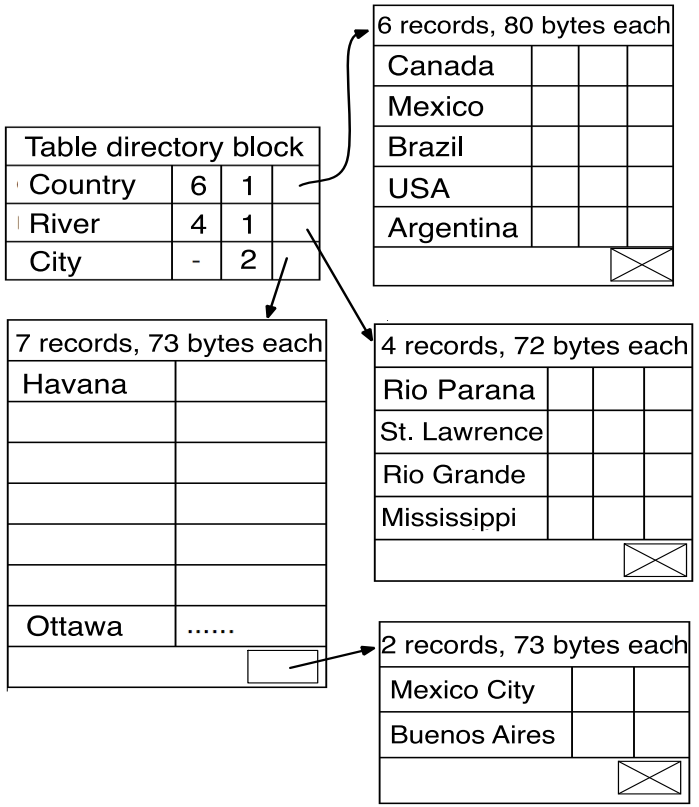
- Examples: Country, River, City
  - Find(Name = Canada) on Country table returns record about Canada
  - FindNext() on Country table returns record about Cuba
    - Since Cuba is next value after Canada in sorted order of Name
  - Insert(record about Panama) into Country table
    - Adds a new record
    - 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 (空间数据)
- 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

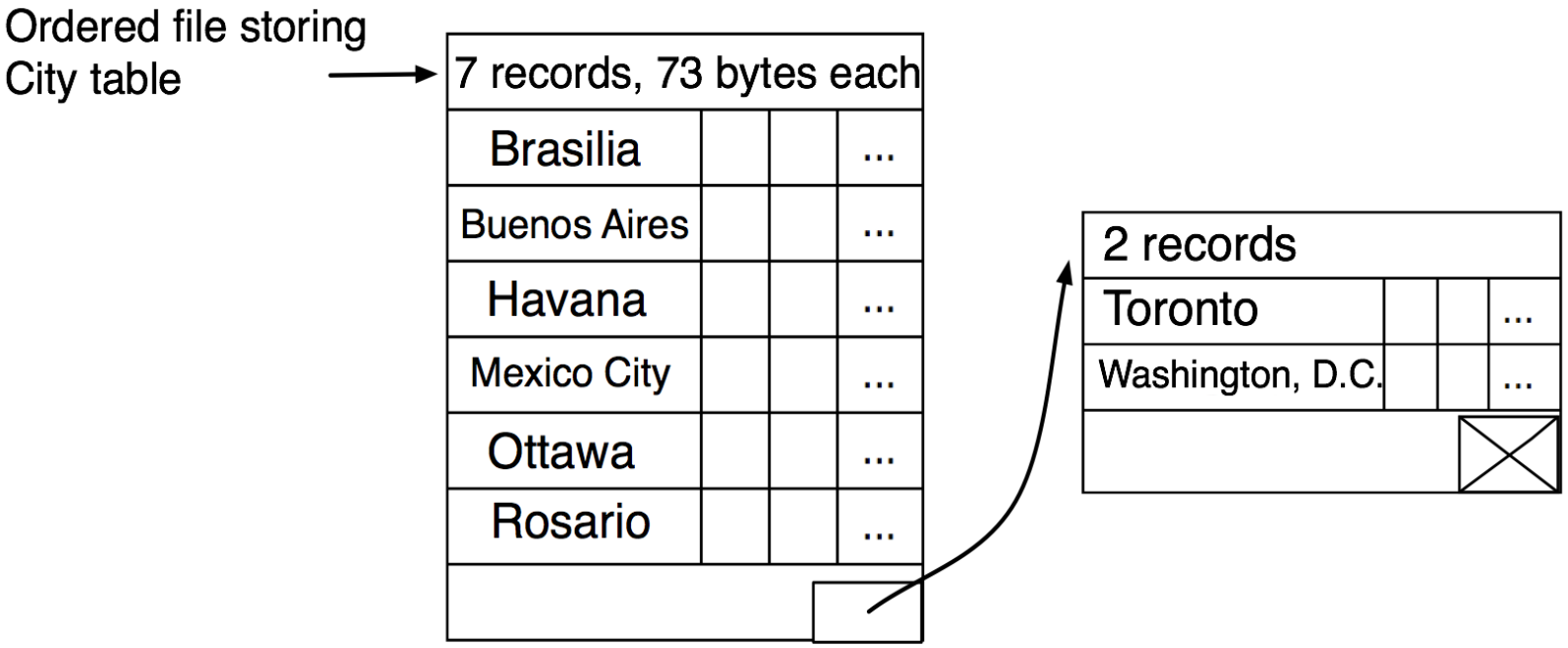
File Structures: Heap

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



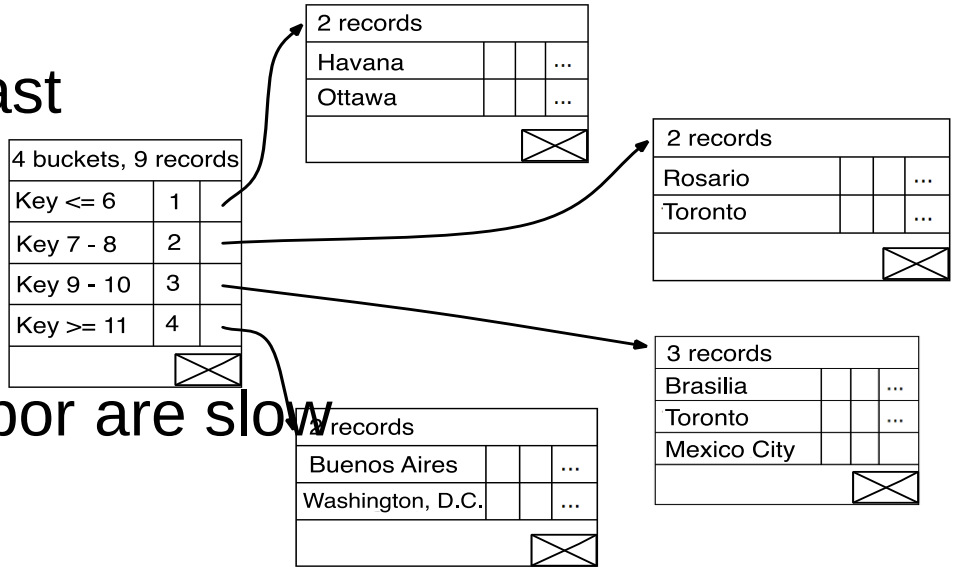
File Structures: Ordered

- 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



Heap、Ordered和Hash文件结构：
参考数据结构中的无序数组、有序数组和散列表

7.3 数据文件组织小结

- 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
- Common file structures and advantages
  - Heap or unordered or unstructured
  - Ordered
  - Hashed
  - Clustered (空间数据)

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

参考教材：
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

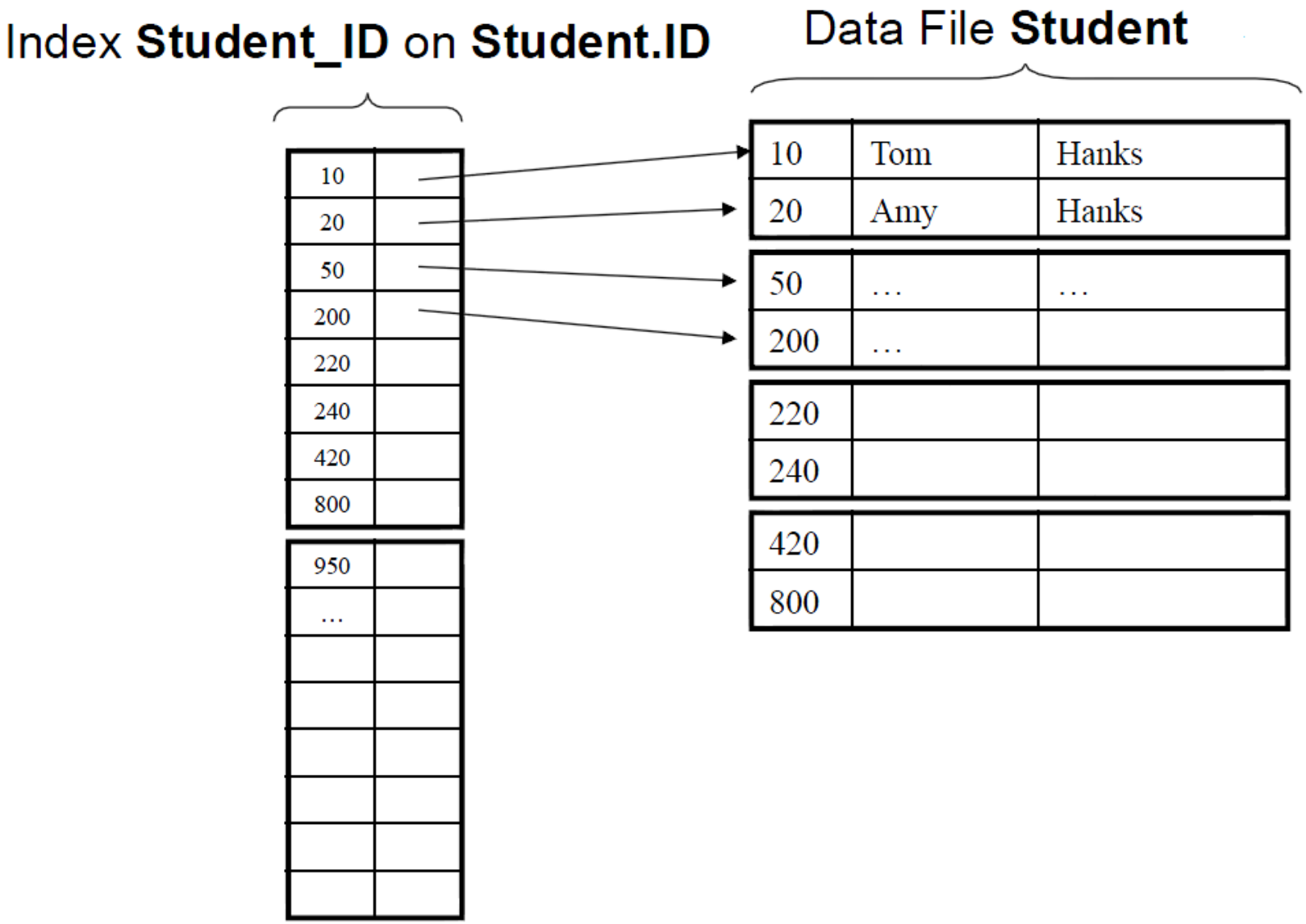
Data File vs. Index File

- 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

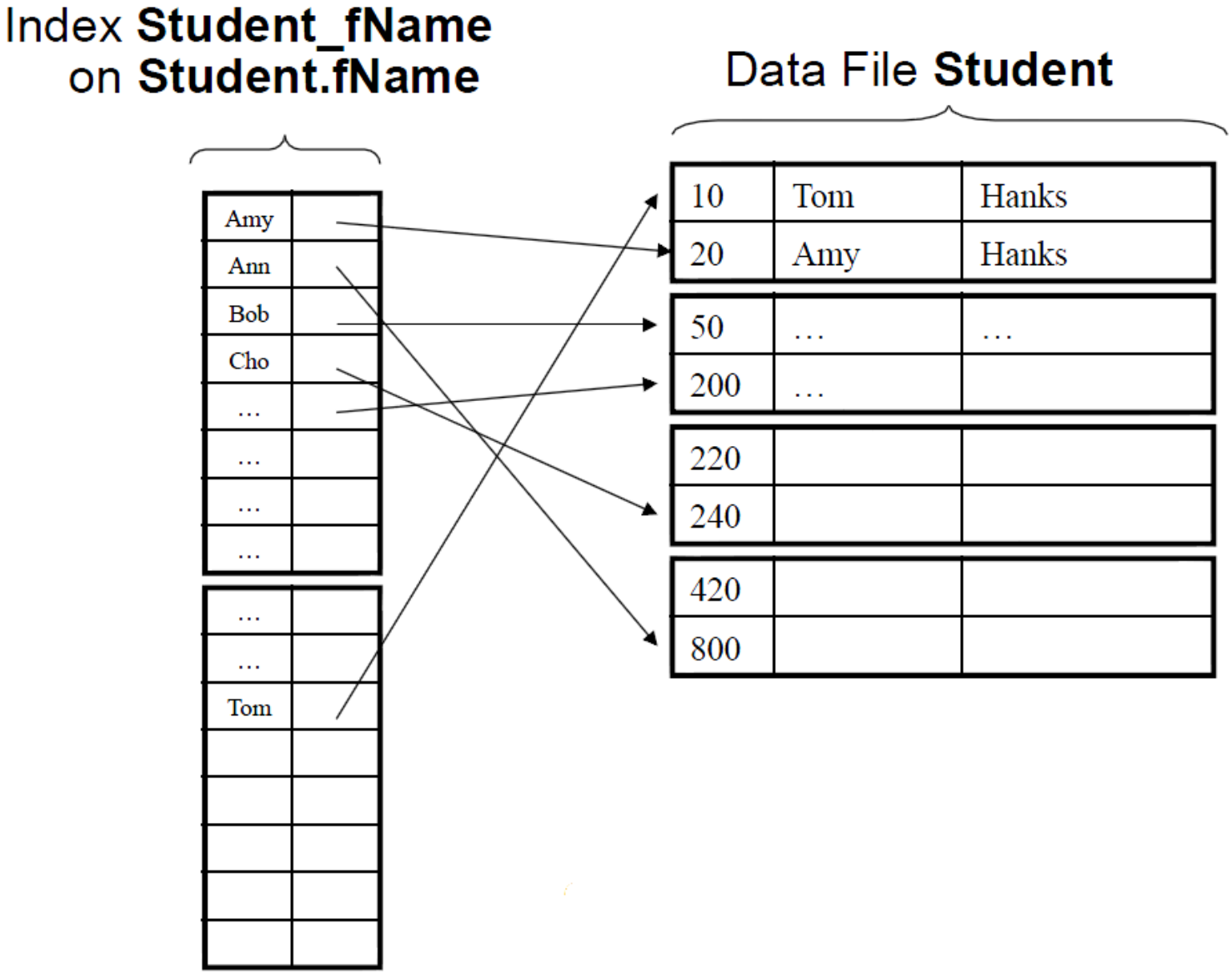
Different Keys

- 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
    - Question: A table can have at most one primary index. Why?
    - 空间属性适合作为primary key？
  - Key of the sequential file – how the data file sorted, if at all
  - Index key – how the index is organized

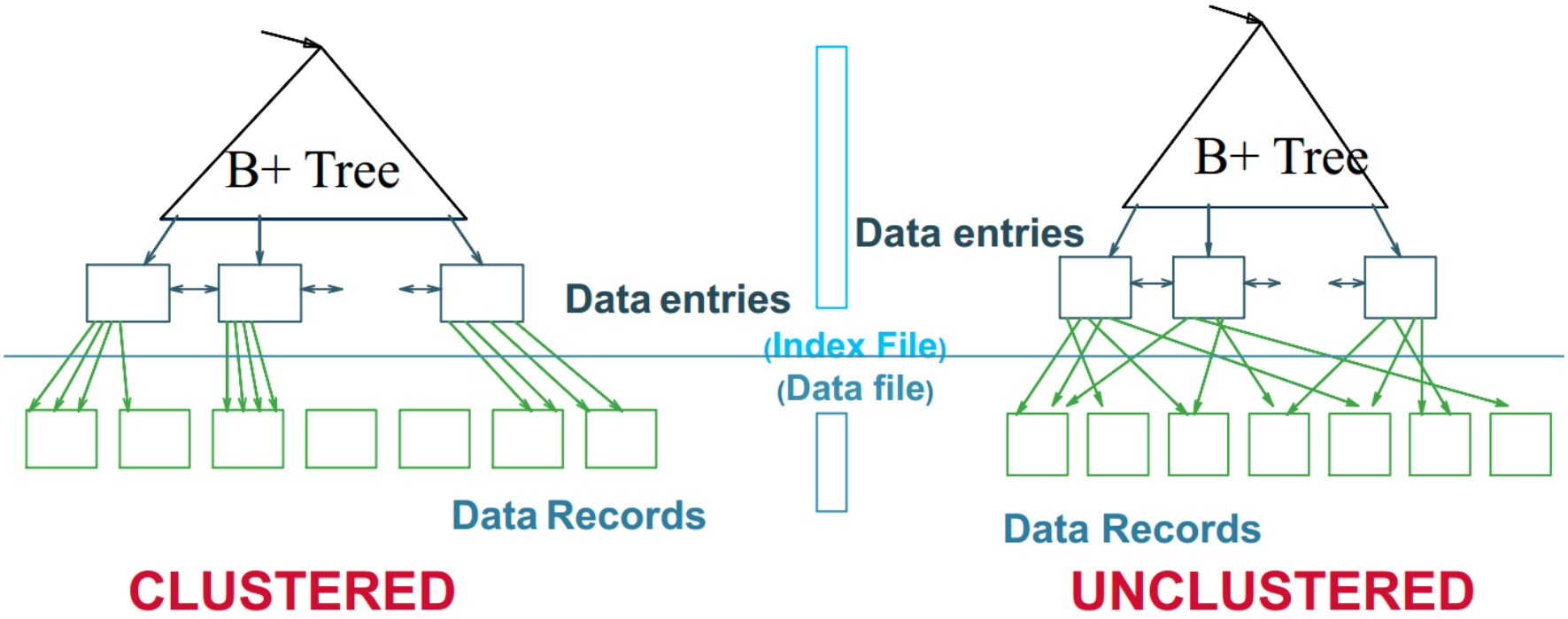
Examples


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

聚集与非聚集索引



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

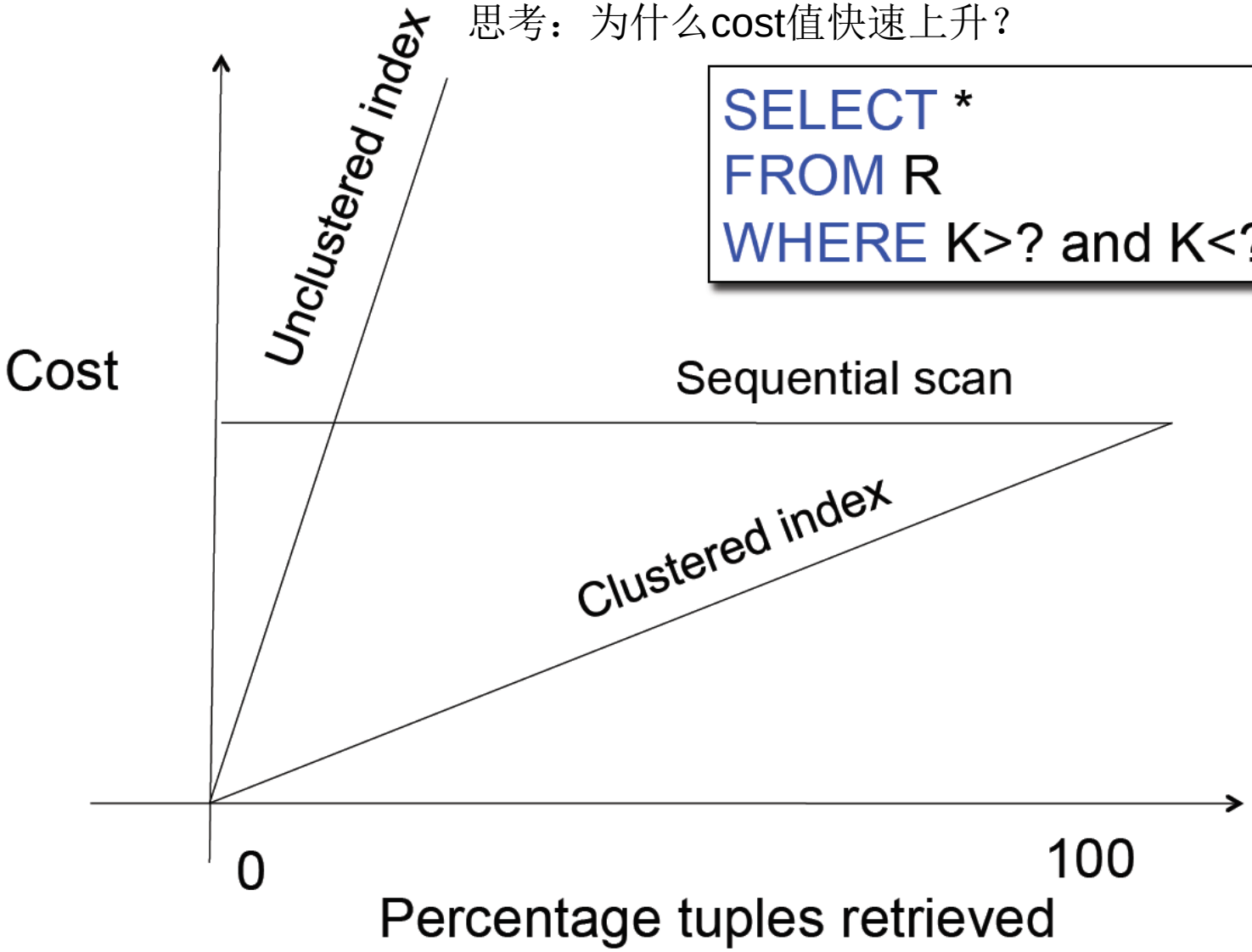
聚集与非聚集索引

- 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？

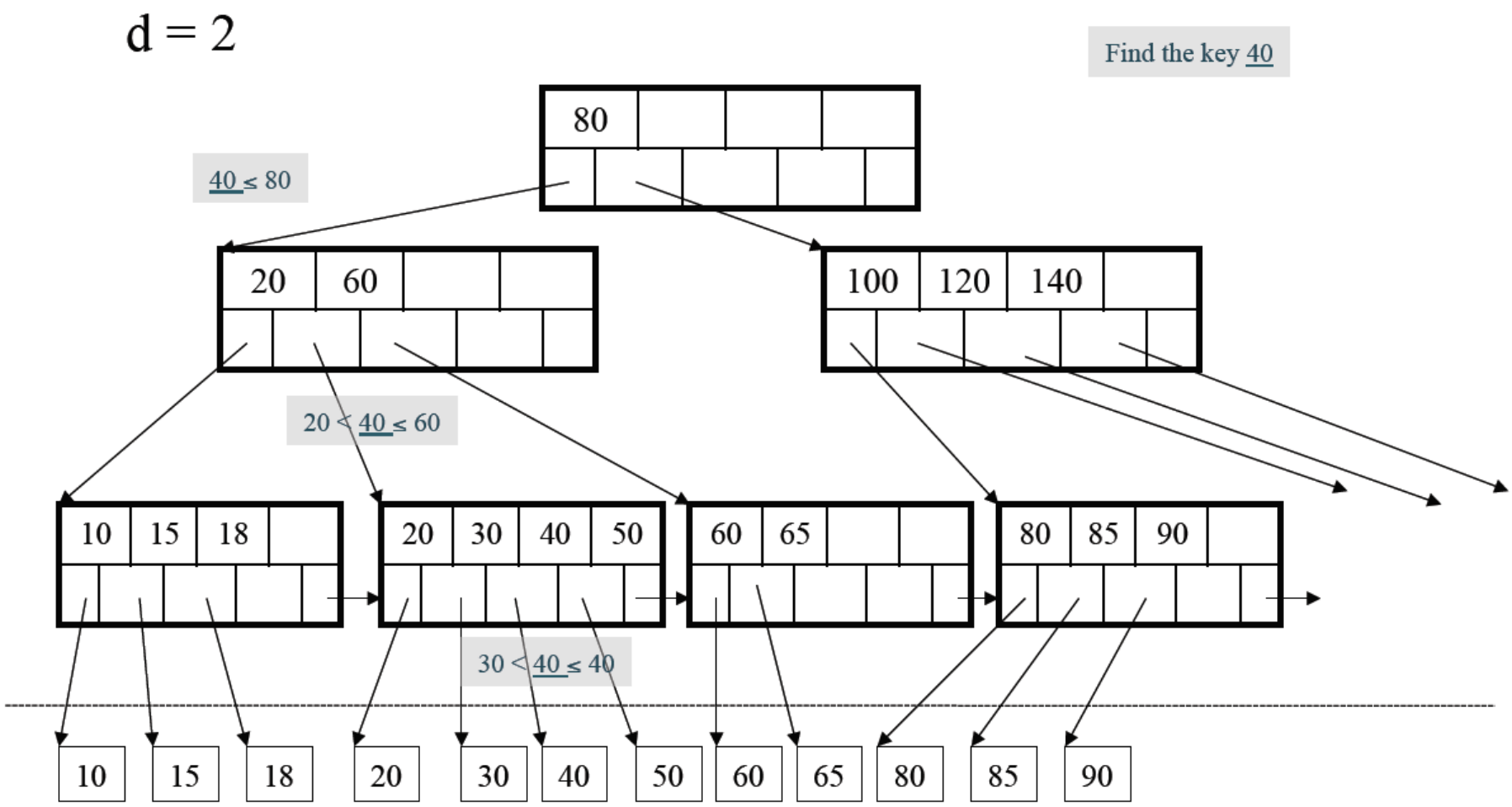
```
SELECT *
FROM R
WHERE K>? and K<?
```



索引的组织

- 索引的组织 (数值索引)
  - 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

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？

Index in PostgreSQL

- R(A int), S(B int)
- create index iR on R(A);
- pgAdmin
  - -- Index: public.ir
  - -- DROP INDEX public.ir;
  - CREATE INDEX ir
    ON public.r
    USING btree (a);

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

SQL 窗口

```
-- Index: public.ir

-- DROP INDEX public.ir;

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

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

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 *

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)

算法复杂度、Merge Sort、Hash等知识在数据结构基础课程中介绍

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/

SDBM索引使用

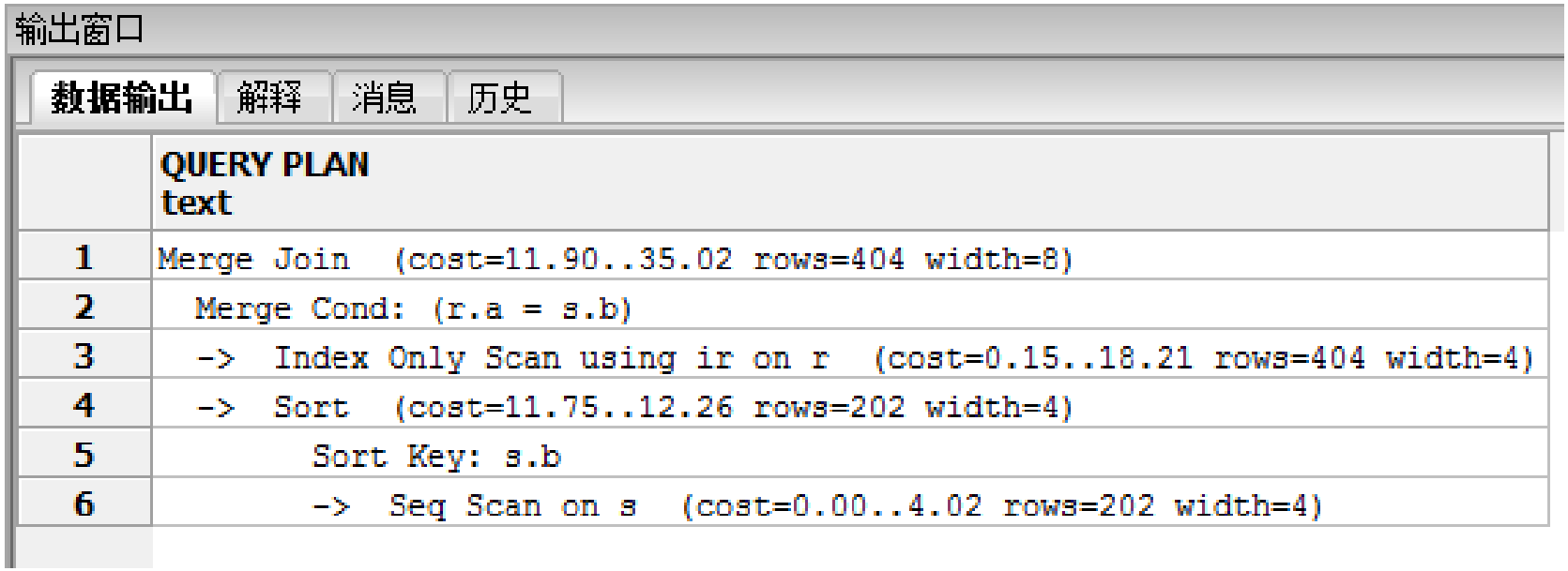
```
Select *
From S, SC
Where S.Sno = SC.Sno and SC.Cno > 300
```

- 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 *
```

PostgreSQL Example

- 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)

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
- 创建了数值索引，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？

7.4 数值索引小结

- 数值索引
  - 概念区分：data file vs. index file, primary key vs. index key, clustered vs. non-clustered
  - B+ trees: most popular
- 创建索引：create index 索引名 on 关系(属性列表)
- 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？

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
  - 7.5.1 Z-Curve
  - 7.5.2 Hilbert Curve
  - 7.5.3 Usage & Algorithm
  - 7.5.4 Regions
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

参考教材：
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

7.5 空间填充曲线

- 关系数据库
  - 数据类型：数字、字符、日期、货币等
    - 特点：1D，能够排序
  - 数据文件组织：heap, ordered, hash
  - 数值索引：B-trees, hash
- 重用关系数据库物理数据模型
  - 空间数据：无序(nD) → 有序(1D)
  - 解决方法：空间填充曲线 (Z-Curve, Hilbert Curve)
  - 重用数据文件组织：heap, ordered, hash
    - 也称为clustered
  - 重用数值索引：B-trees, hash

Spatial File Structures: Clustering

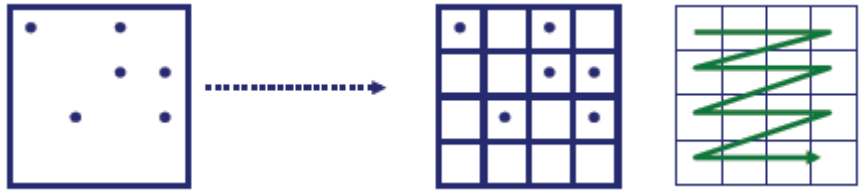
- Motivation
  - Ordered files are not natural for spatial data
  - Clustering records in a sector by space filling curve is an alternative
  - In general, clustering groups records
    - Accessed by common queries into common disk sectors
    - To reduce I/O costs for selected queries
- Clustering using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

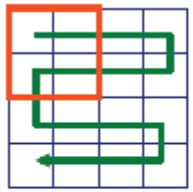
空间填充曲线

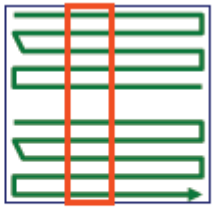
- Q: how would you organize, e.g., *n*-dim points, on disk? (*C* points per disk page)
  - Hint: reduce the problem to 1-d points
- Q1: why?
  - A1: B-trees
- Q2: how?
  - A2: assume finite granularity (e.g., 2^{32}x2^{32}; 4x4 here)
- Q2.1: how to map n-d cells to 1-d cells?
  - A2.1: row-wise
  - Q: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



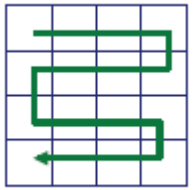
空间填充曲线

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



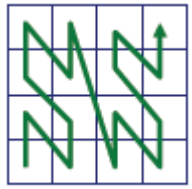


2^32

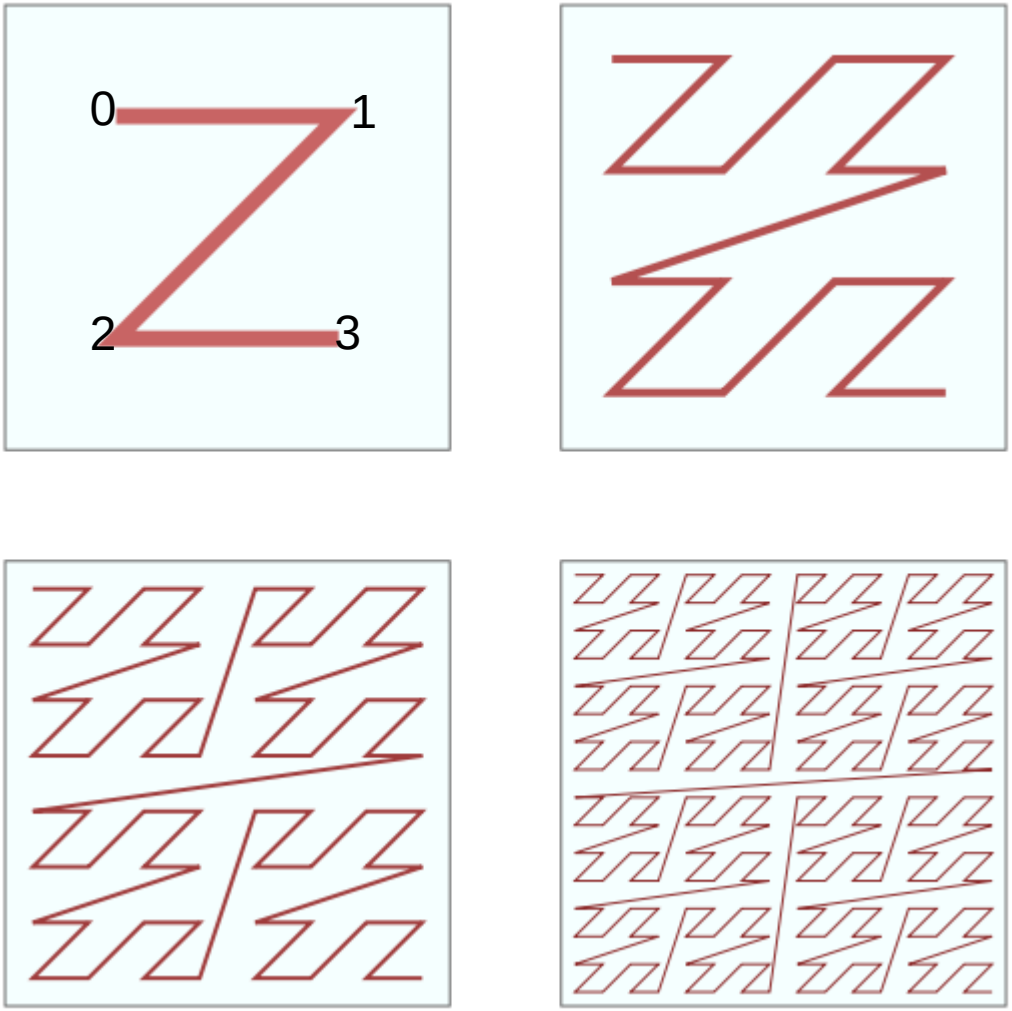


2^32

- 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



7.5.1 Z-Curve

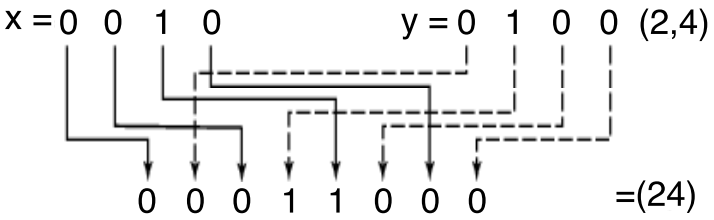


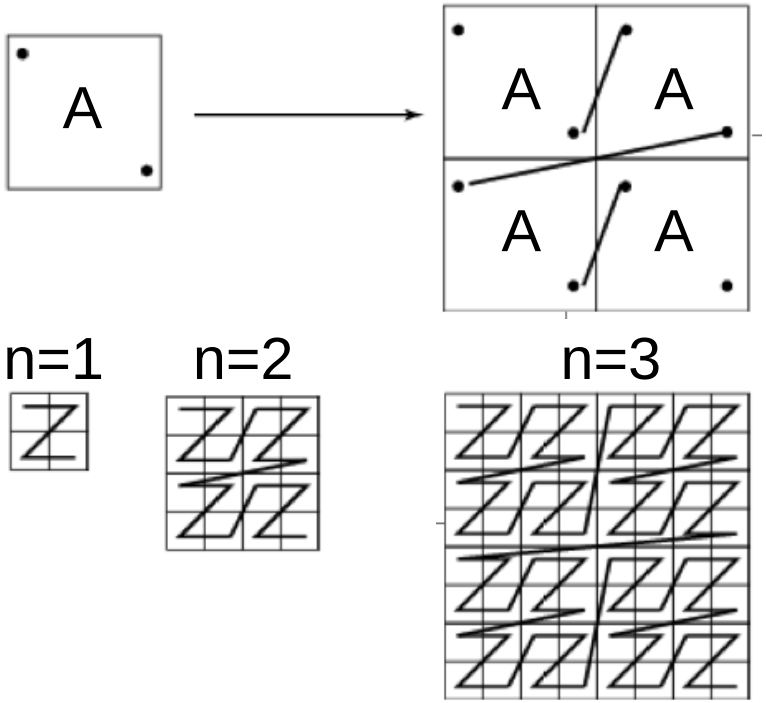
http://en.wikipedia.org/wiki/Z-order_curve

7.5.1 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

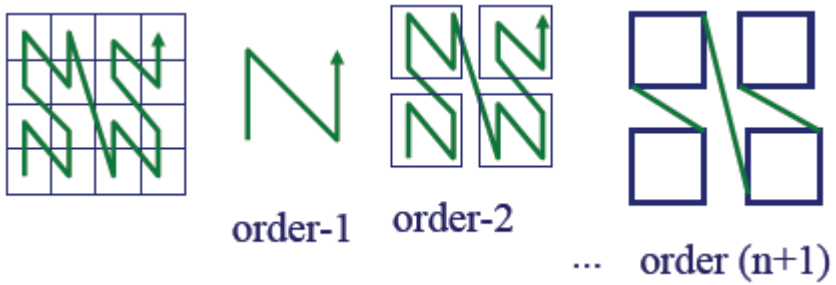
(重用ordered files和B-trees) □



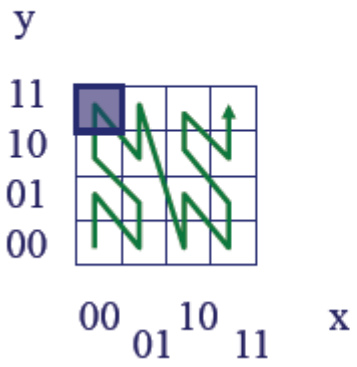


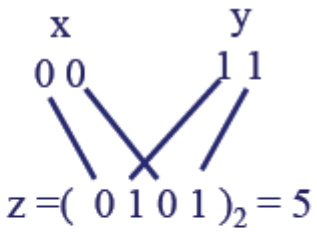
7.5.1 Z-Curve

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



- A2: bit-shuffling



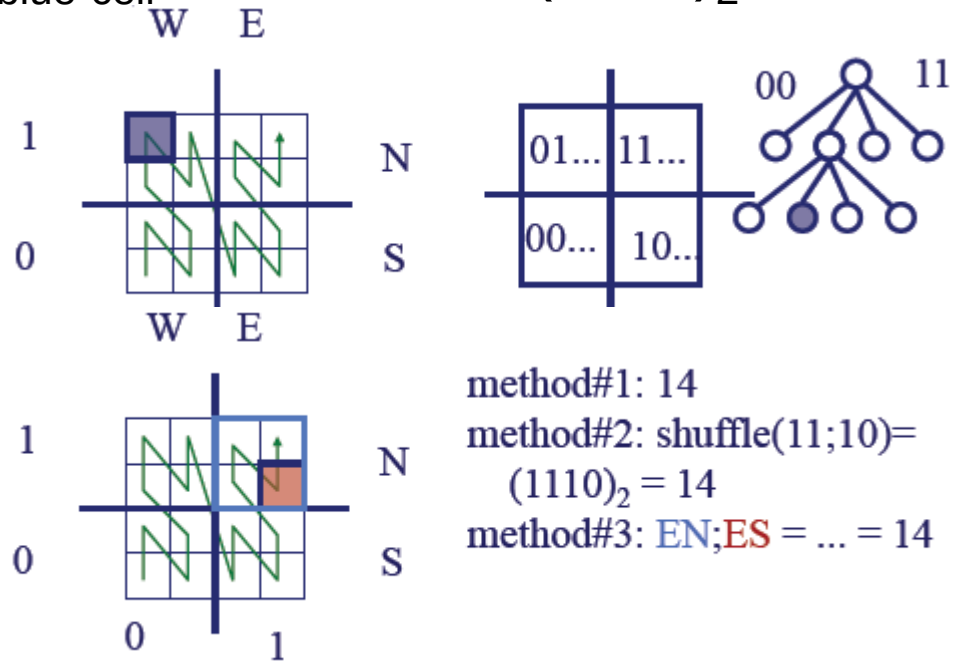


How about the reverse:

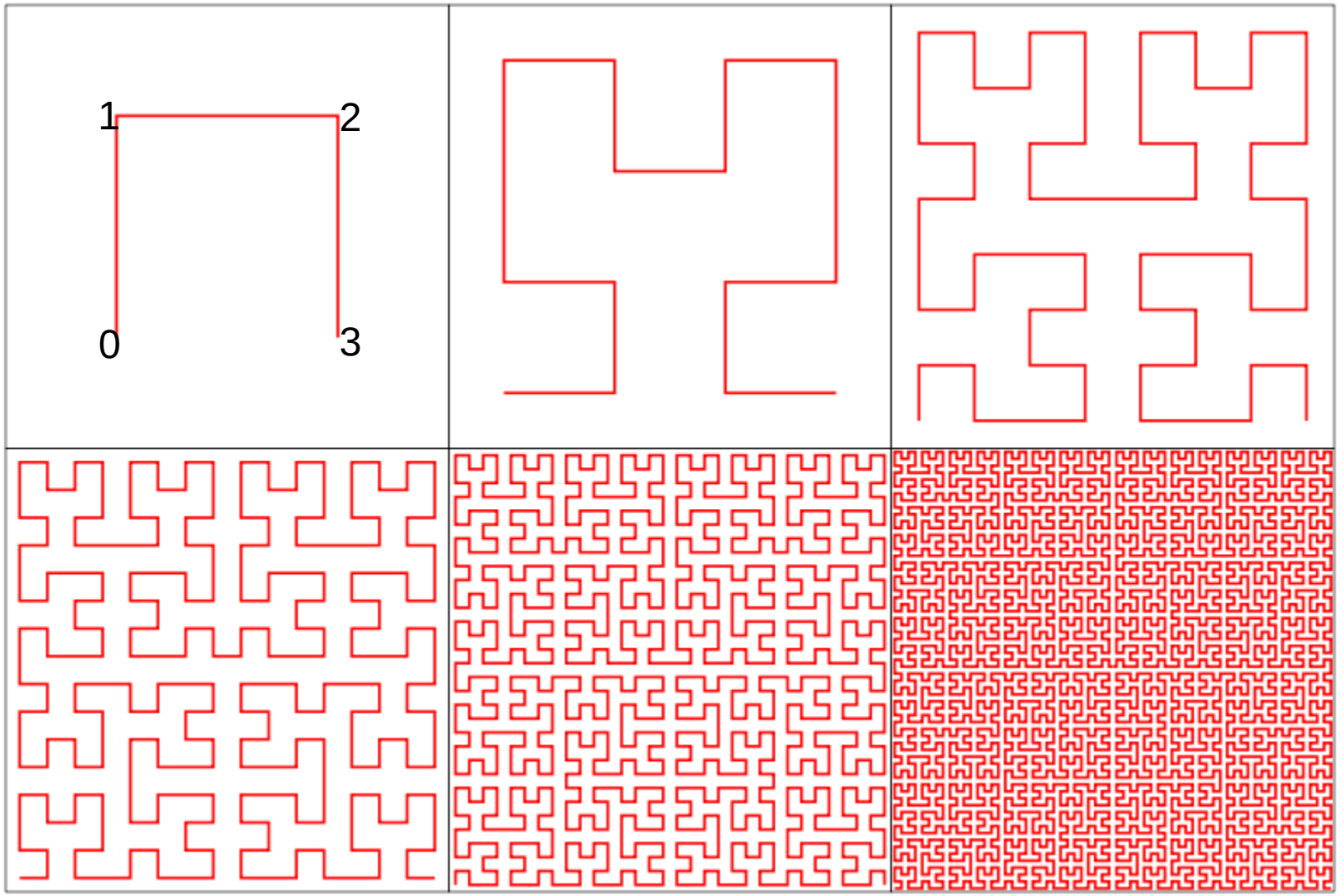
$(x,y) = g(z)$?

7.5.1 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



7.5.2 Hilbert Curve

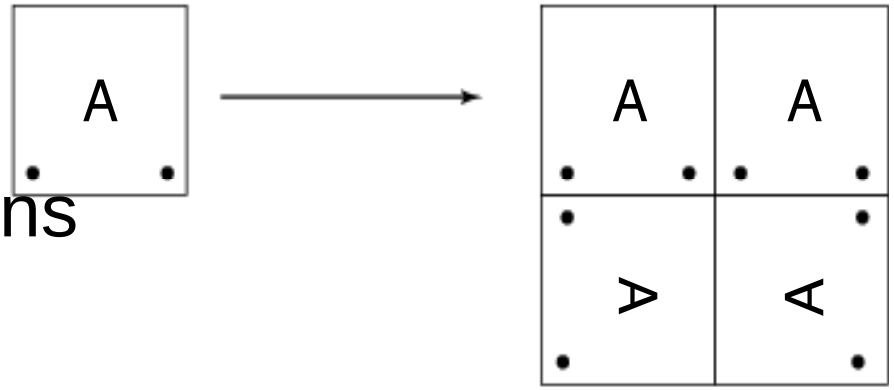


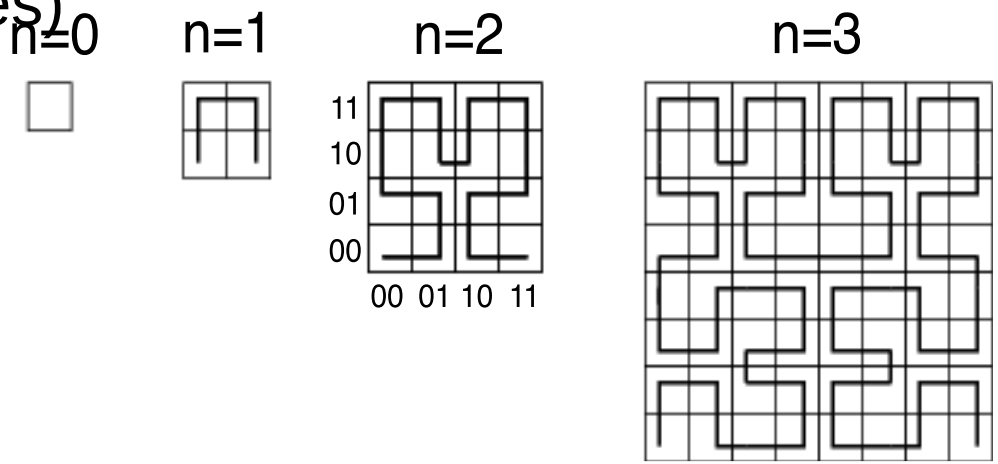
http://en.wikipedia.org/wiki/Hilbert_curve

7.5.2 Hilbert Curve

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

(重用ordered files和B-trees)





Converting (X,Y) to a Hilbert Value

1. Input n-bit binary representations of X and Y
2. Interleave bits of X and Y into one string S
3. Split S into an array of 2 bit strings
4. Convert 2-bit strings as follows:
   “00” = 0, “01” = 1, “10” = 3, “11” = 2
5. Rotate/Reflect: Scan array entries j left to right:
   - If j = 0: switch following occurrences of 1 to 3 and following occurrences of 3 to 1
   - If j = 3: switch following occurrences of 0 to 2 and following occurrences of 2 to 0
6. Convert each array entry to binary
7. Calculate decimal value

	Example **A**	Example **B**
1.	X = 001 , Y = 000	X=100, Y=001

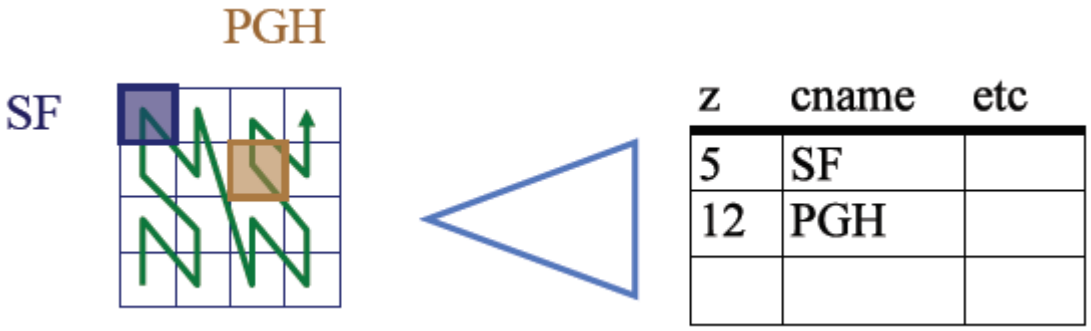
Z-Curve vs. Hilbert Curve

- 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？

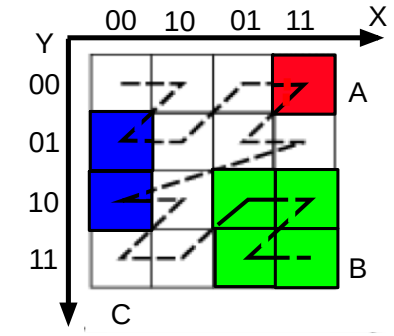
7.5.3 Usage & Algorithms

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



Z-order and Extended Objects

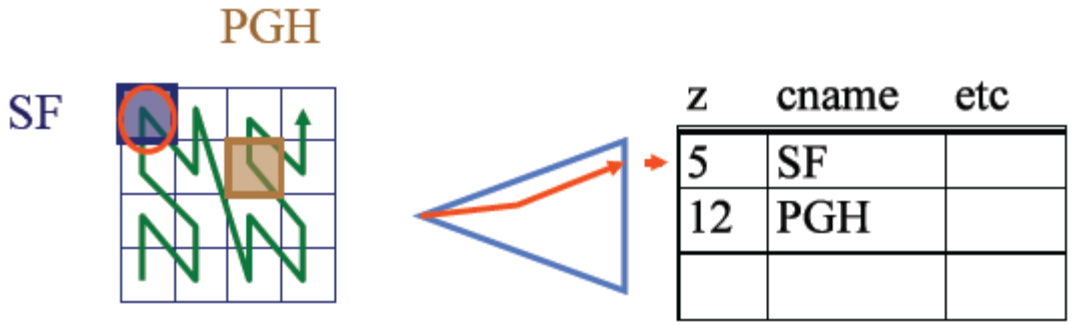
- Example
  - Left part shows a map with spatial object A, B, C
  - Right part and Left bottom part Z-values within A, B and C
  - Note C gets z-values of 2 and 8, which are not close
  - Exercise: Compute z-values for B



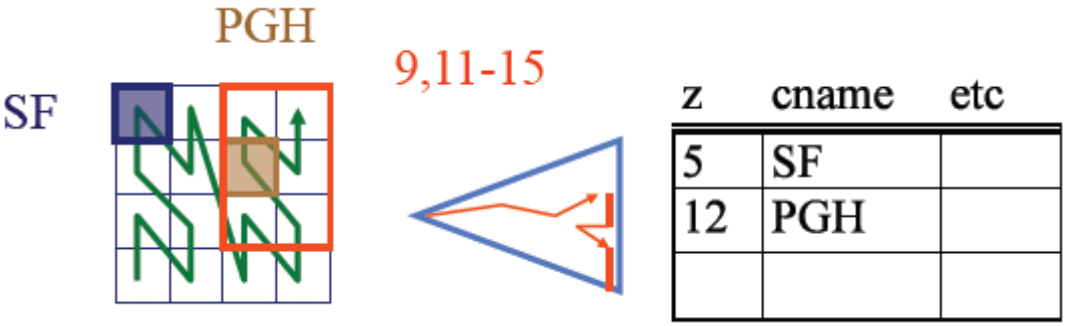
Object	Points	y	x	interleave	z-value
A	1	00	11	0101	5
B	1				
	2				
	3				
	4				
C	1	01	00	0010	2
	2	10	00	1000	8

7.5.3 Usage & Algorithms

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

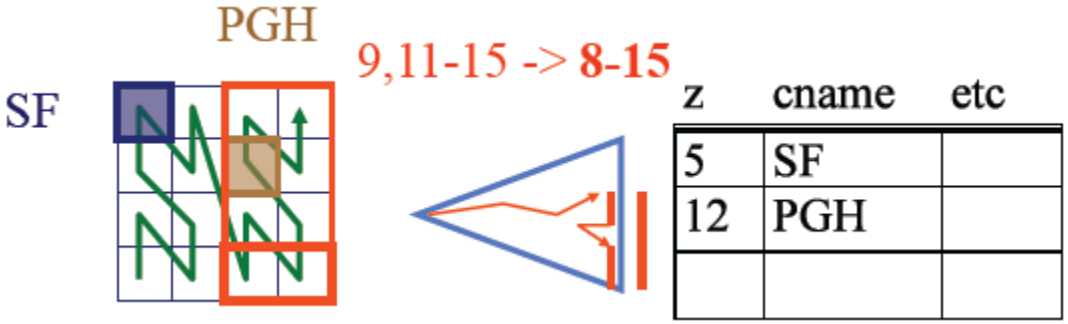


- Q2: range queries?
- A: compute ranges of z-values; use B-tree

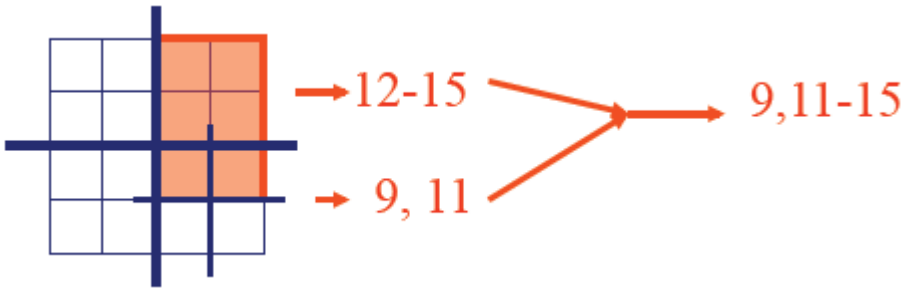


7.5.3 Usage & Algorithms

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

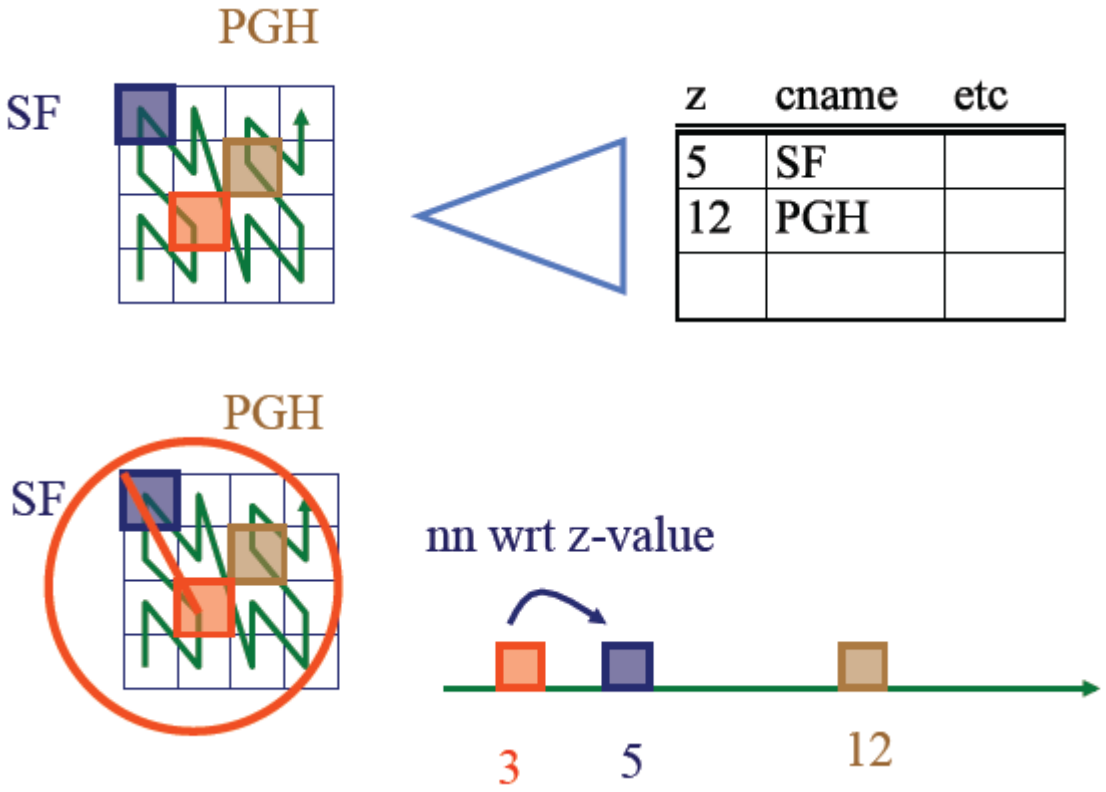


- Q2'': range queries – how to break a query into ranges?
- A: recursively, quadtree-style; decompose only non-full quadrants



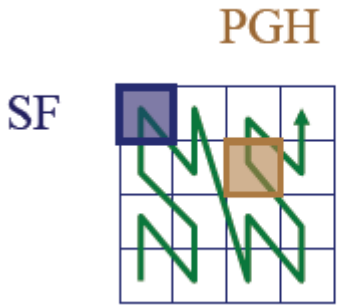
7.5.3 Usage & Algorithms

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



7.5.3 Usage & Algorithms

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

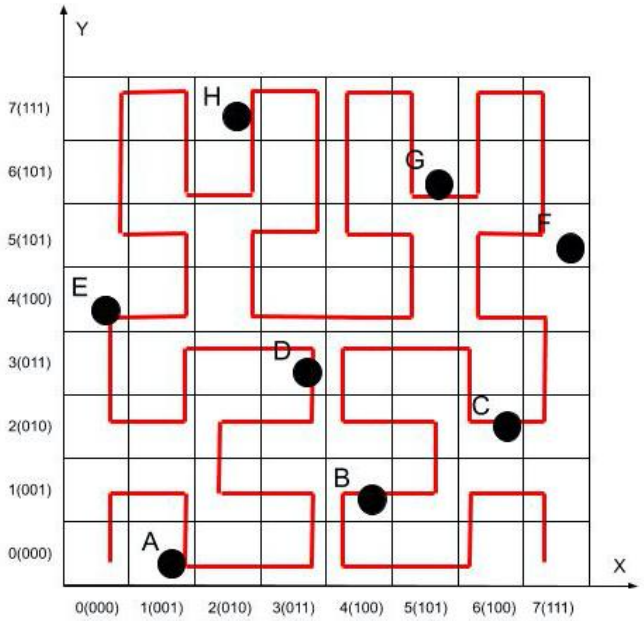


(we'll see 'spatial joins' later: *find all PA counties that intersect a lake)*

7.5.3 Usage & Algorithms

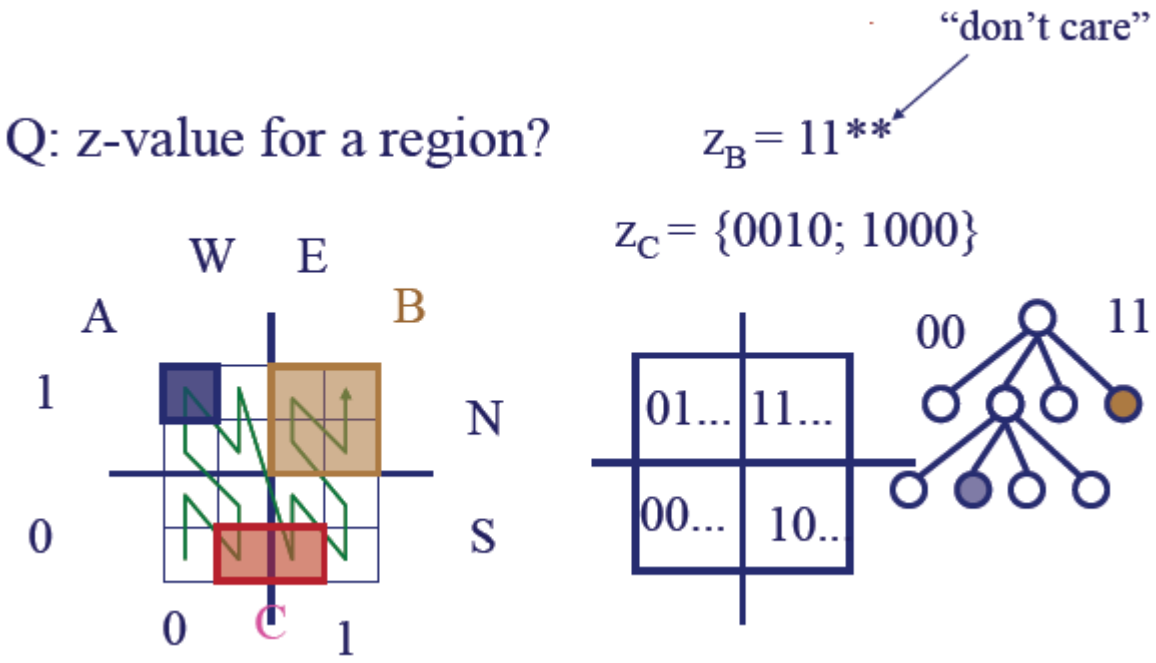
Index	Key	Hilbert
0	A	3
1	D	10
2	E	16
3	H	25
4	G	39
5	F	44
6	C	50
7	B	57

- Point query using Hilbert-Curve (无B-trees)
- Sort keys by ascending H-Curve values
  - Ex. Data = (A, B, C, … , H) in picture
- Binary search on Hilbert value of search key
  - Ex.: Search key = (001, 000)
    - Hilbert value = 3
    - Check index 3
    - Check index 1
    - Check index 0
  - Ex.: Search Key = (000, 100)
    - Hilbert Value = 16
    - Check index 3
    - Check index 1
    - Check index 2



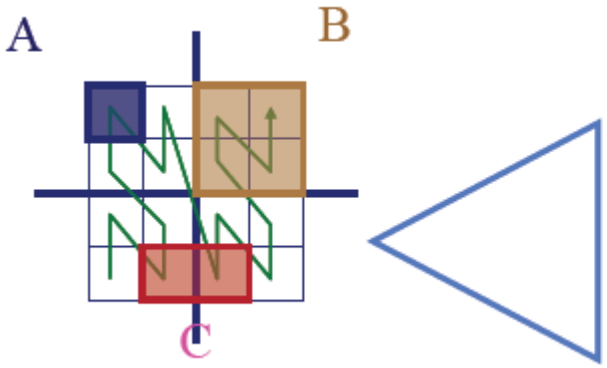
7.5.4 Regins

- Q: z-value for a region? (以Z-Curve为例)
- A: 1 or more z-values; by quadtree decomposition



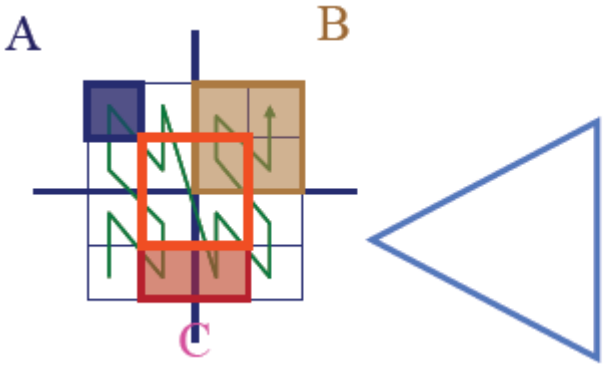
7.5.4 Regins

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



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

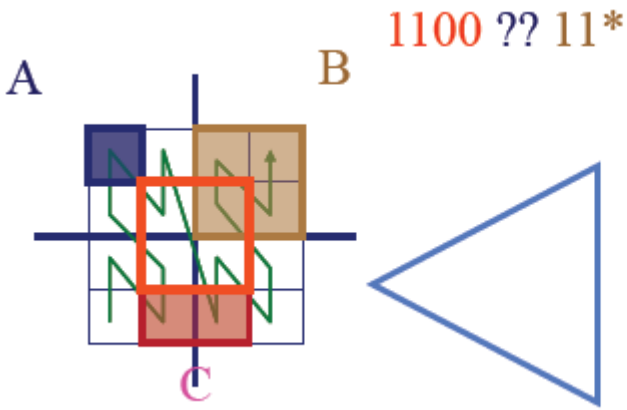
- Q: how to search (range query)
- A: break query in z-values; check B-tree



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

7.5.4 Regins

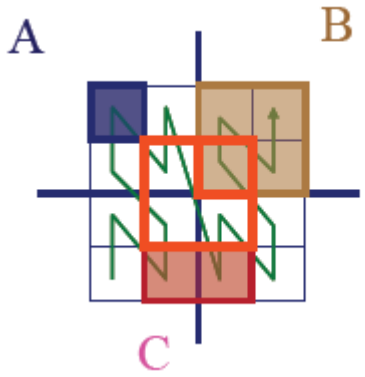
- Almost identical to range queires for point data, except for the “don’t cares” – i.e.,
  - z1 = 1100 ?? 11** = z2
- Specifically: does z1 contain/avoid/intersect z2?
- Q: what is the criterion to decide?



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

7.5.4 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

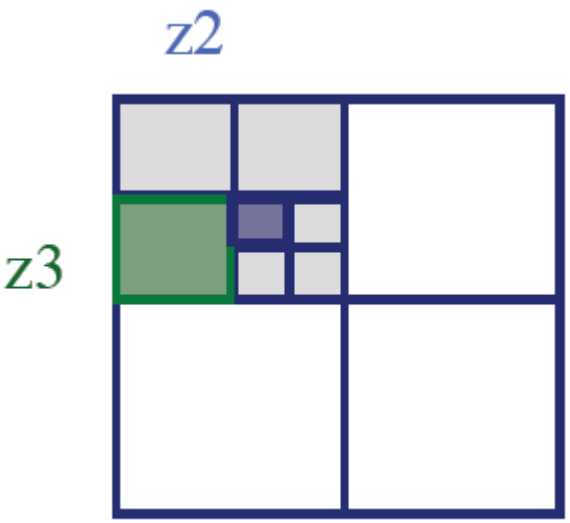


1100 ?? 11**

region of z1:
completely contained in
region of z2

7.5.4 Regins

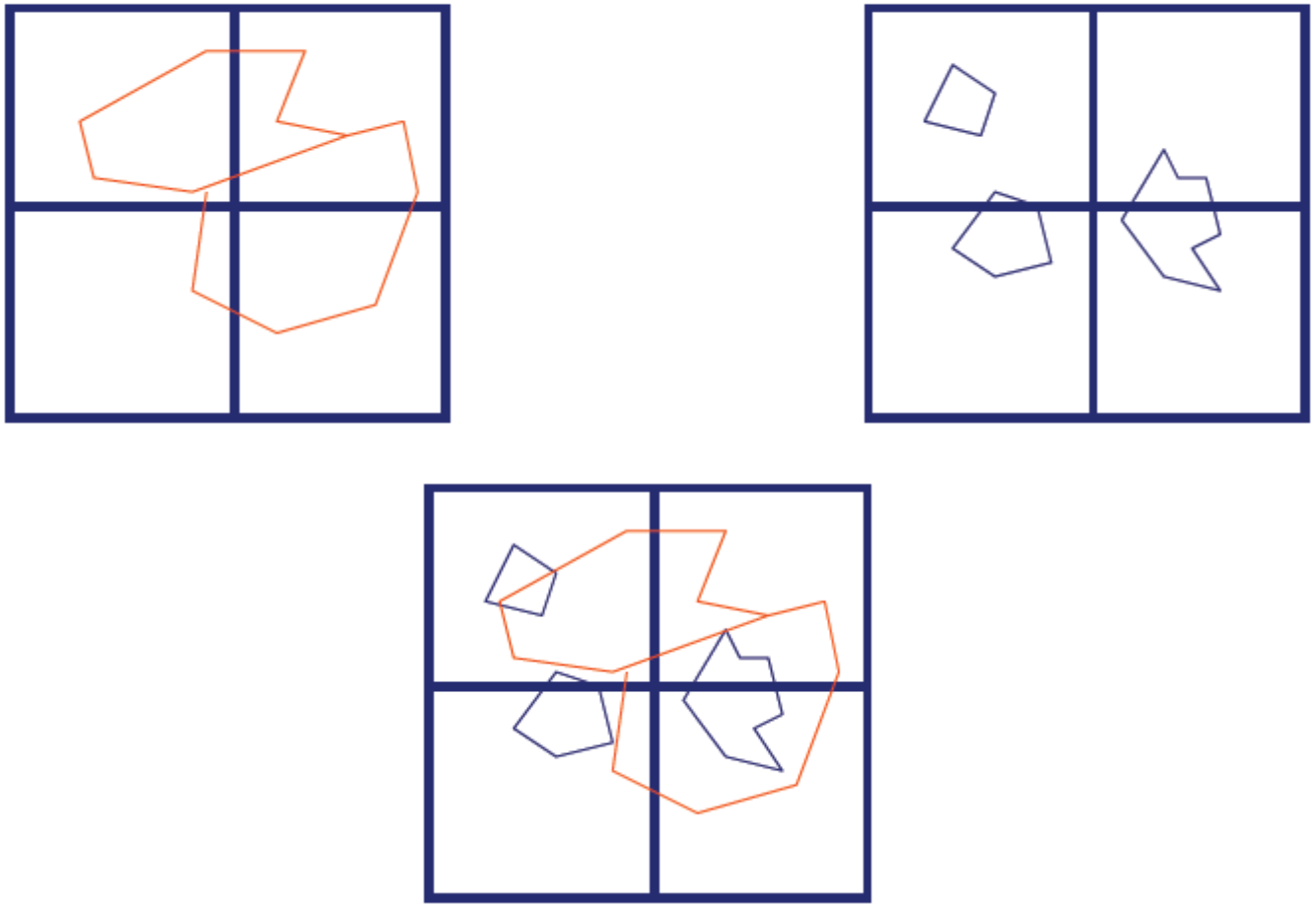
- Drill (True/False). Given:
  - z1= 011001**
  - z2= 01******
  - z3= 0100****
- T/F z2 contains z1
  - True (prefix property)
- T/F z3 contains z1
  - False (disjoint)
- T/F z3 contains z2
  - False (z2 contains z3)



7.5.4 Regins

- Spatial joins: find (quickly) all

counties intersecting lakes



7.5.4 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

7.5 空间填充曲线小结

- Clustering using space filling curves (nD → 1D)
  - Z-curve
  - Hilbert-curve
- Usage & Algorithms (重用关系数据库物理数据模型)
  - Z-value / H-value based sorted files + B-trees / Binary Se.
  - Point query
    - Calcualte Z value, search B-tree
  - Range query
    - Calculate Z ranges, search the smallest value in B-tree, traverse
  - Knn
    - Calculate Z value, search B-tree, traverse, range query
  - Spatial join
    - Merge the list of (sorted) z-values, looking for the prefix property

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
  - 7.6.1 网格索引
  - 7.6.2 四叉树索引
  - 7.6.3 R树索引
- 7.7 PostGIS空间索引
- 7.8 总结

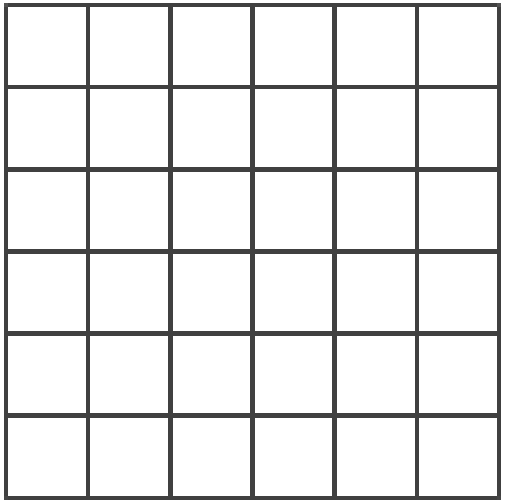
参考教材：
Spatial Databases: A Tour, Chapter 4
15-826: Multimedia Databases and Data Mining
空间数据库管理系统概论，第六章

空间索引

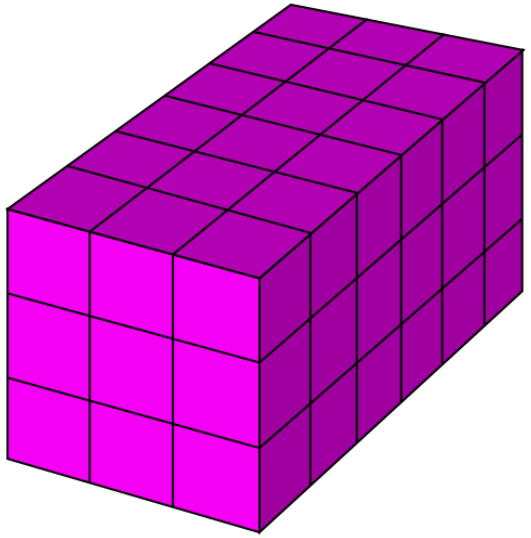
- 空间索引
  - 依据空间实体的位置和形状或空间实体之间的某种空间关系，按一定顺序排列的一种数据结构，其中包含空间实体的概要信息，如对象的标识，最小边界矩形及指向空间实体数据的指针
- 传统的数据索引技术——数值数据
  - B树，二叉树，ISAM索引，Hash索引
- 空间索引——空间数据
  - 网格索引，四叉树索引，空间填充曲线索引，R树索引

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

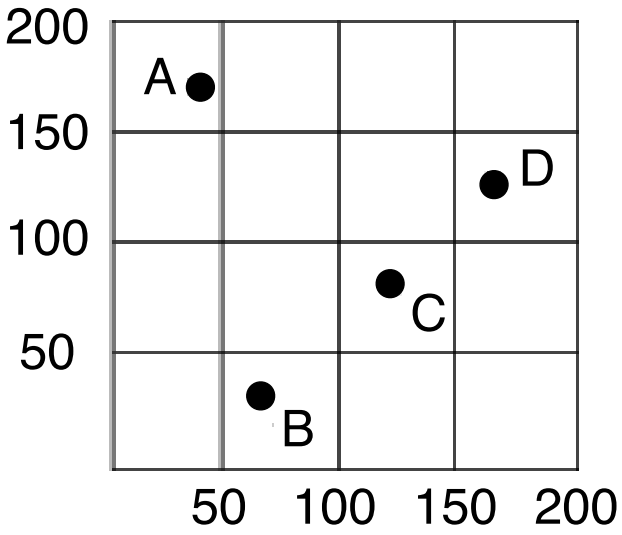


均匀网格(Cubic)



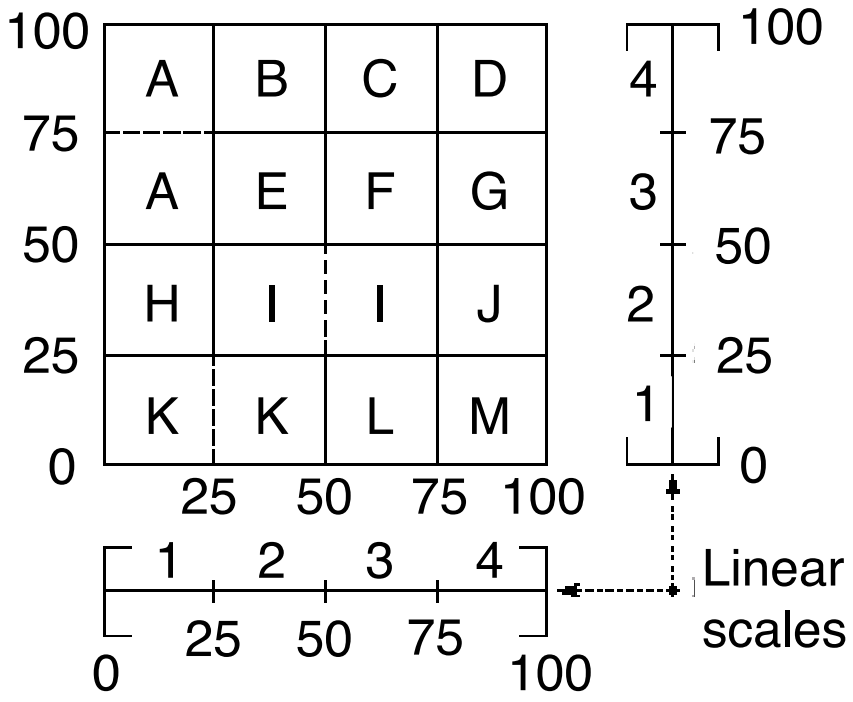
7.6.1 Grid Files

- 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



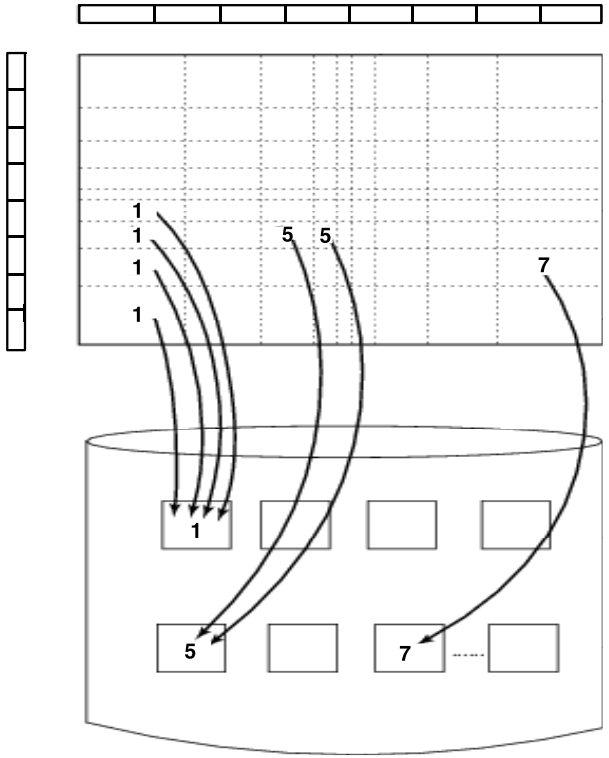
7.6.1 Grid Files

- 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



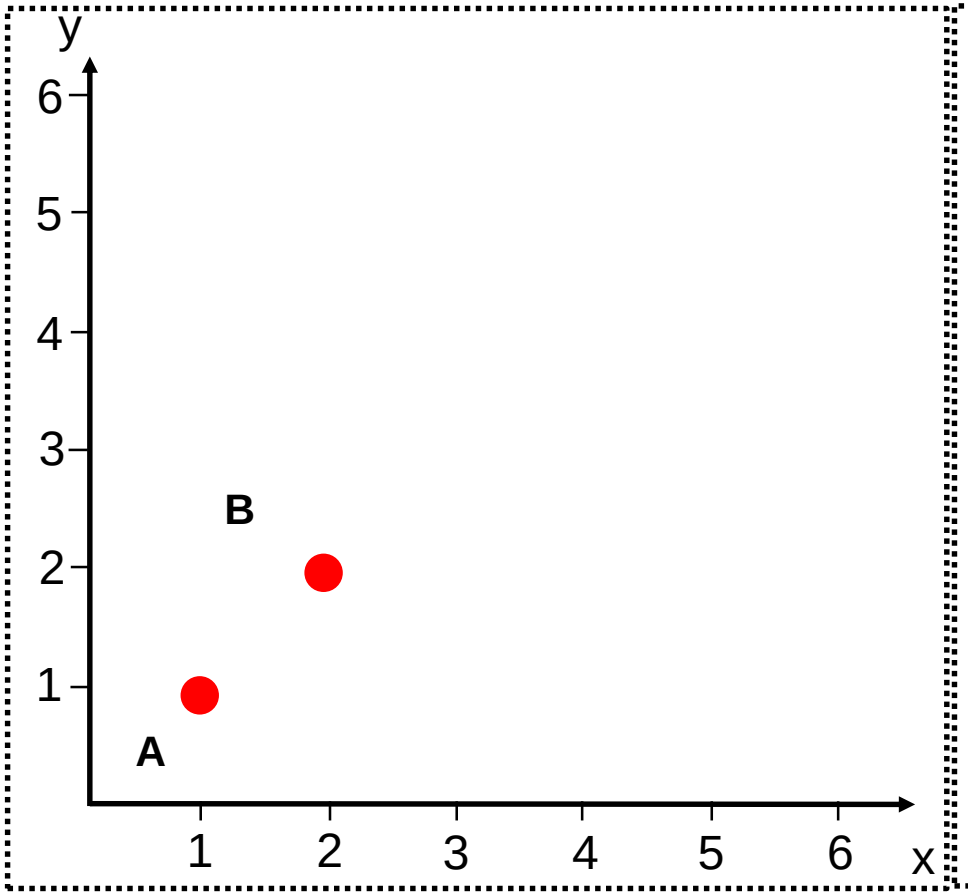
7.6.1 Grid Files

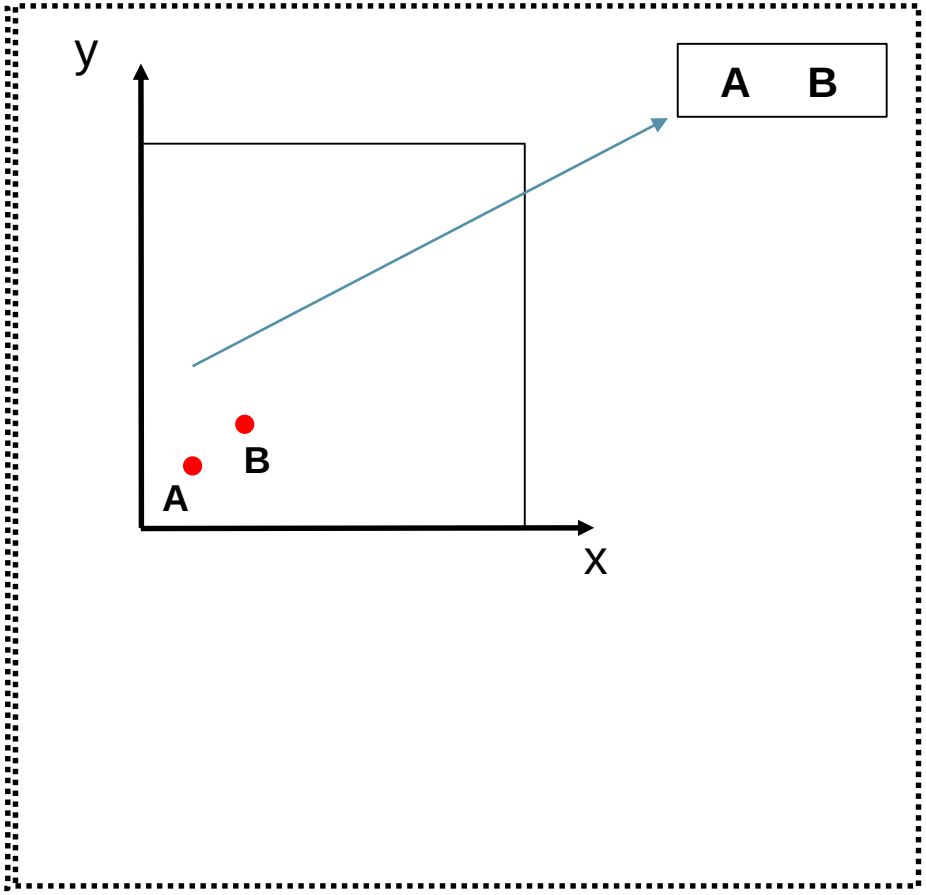
- Grid File component
  - Linear scale - row/column boundaries
  - Grid directory: cell → disk sector address
  - Data sectors on disk
- Steps for find, nearest neighbor
  - Search linear scales
  - Identify selected grid directory cells
  - Retrieve selected disk sectors
- Performance overview
  - Scales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



7.6.1 Grid Files

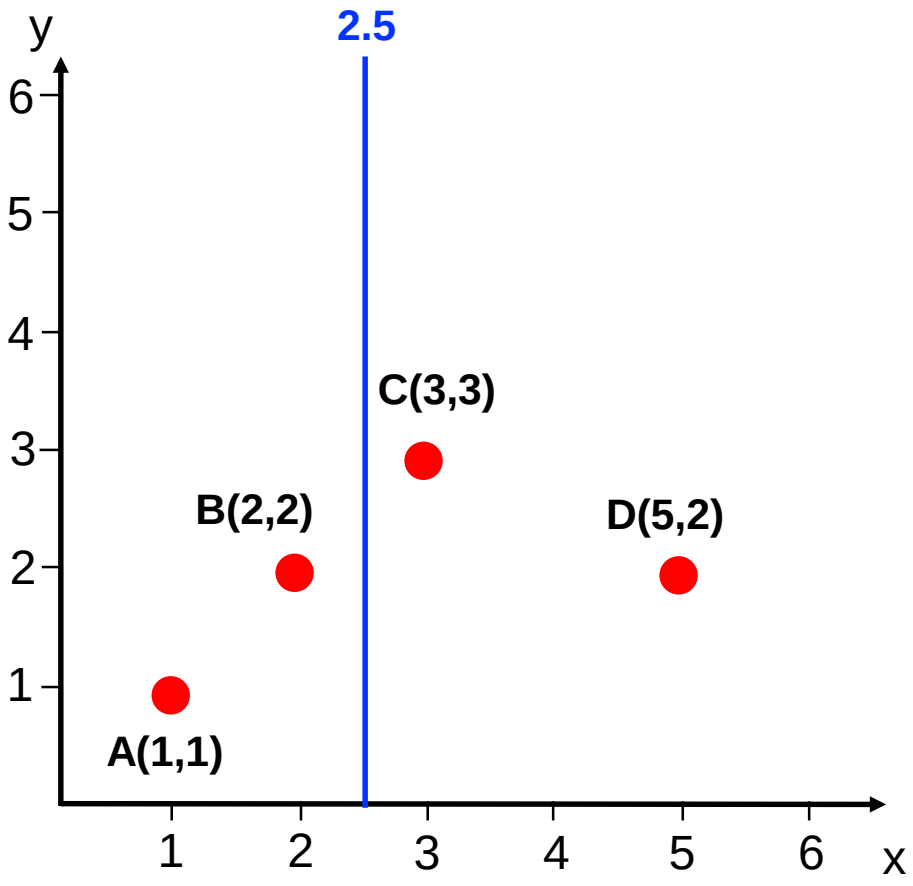
- Simple Insert Example (Blocking factor = 2)

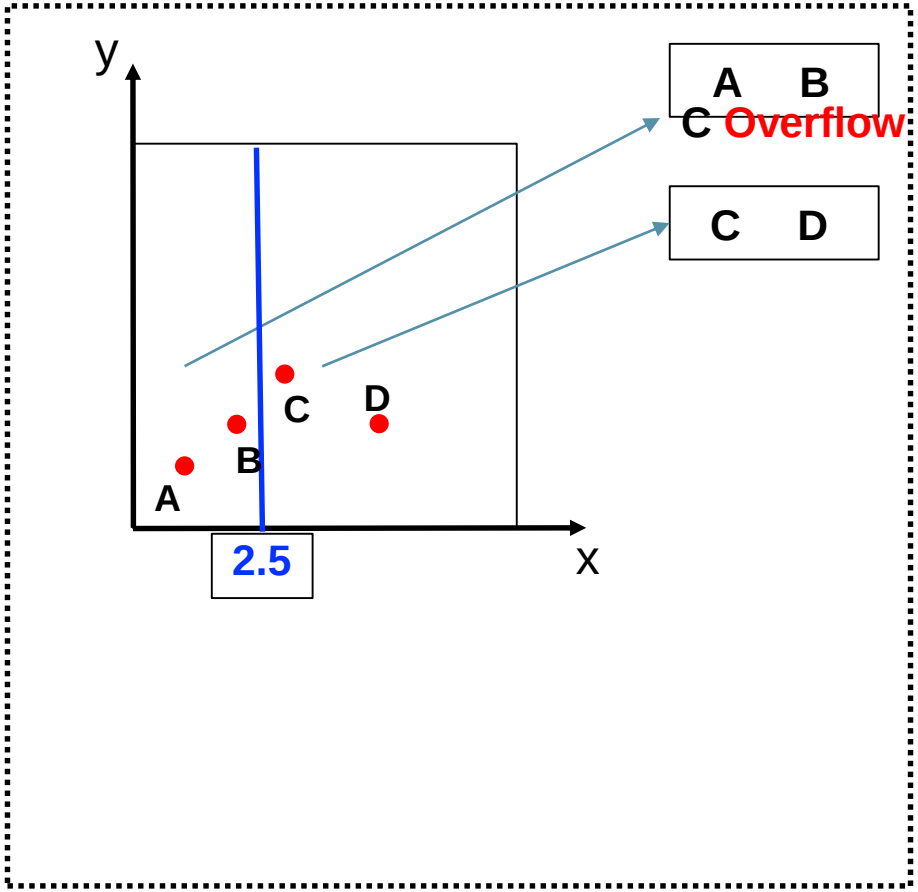




7.6.1 Grid Files

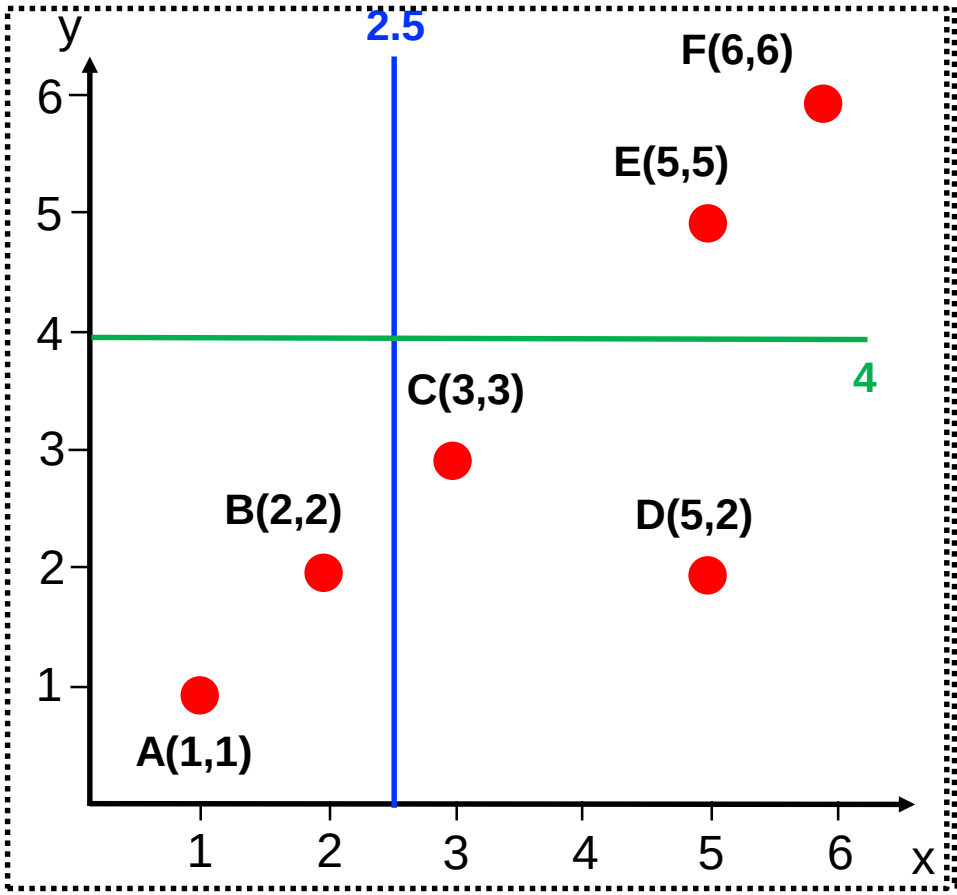
- Simple Insert Example (Blocking factor = 2)

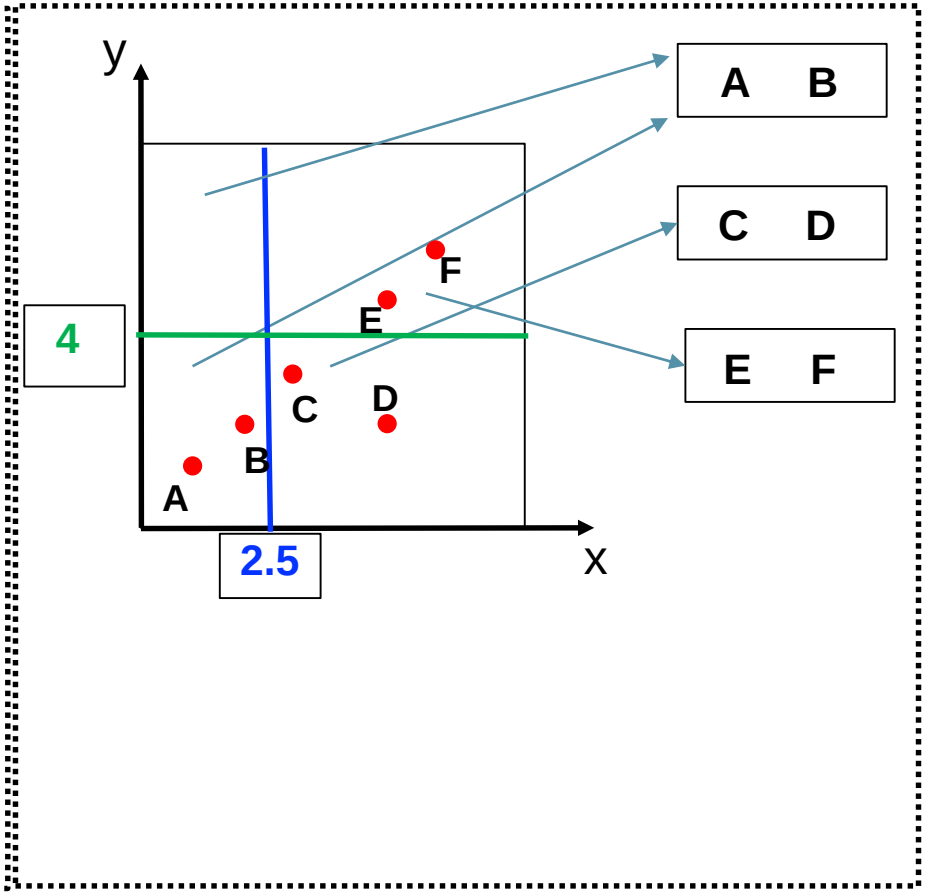




7.6.1 Grid Files

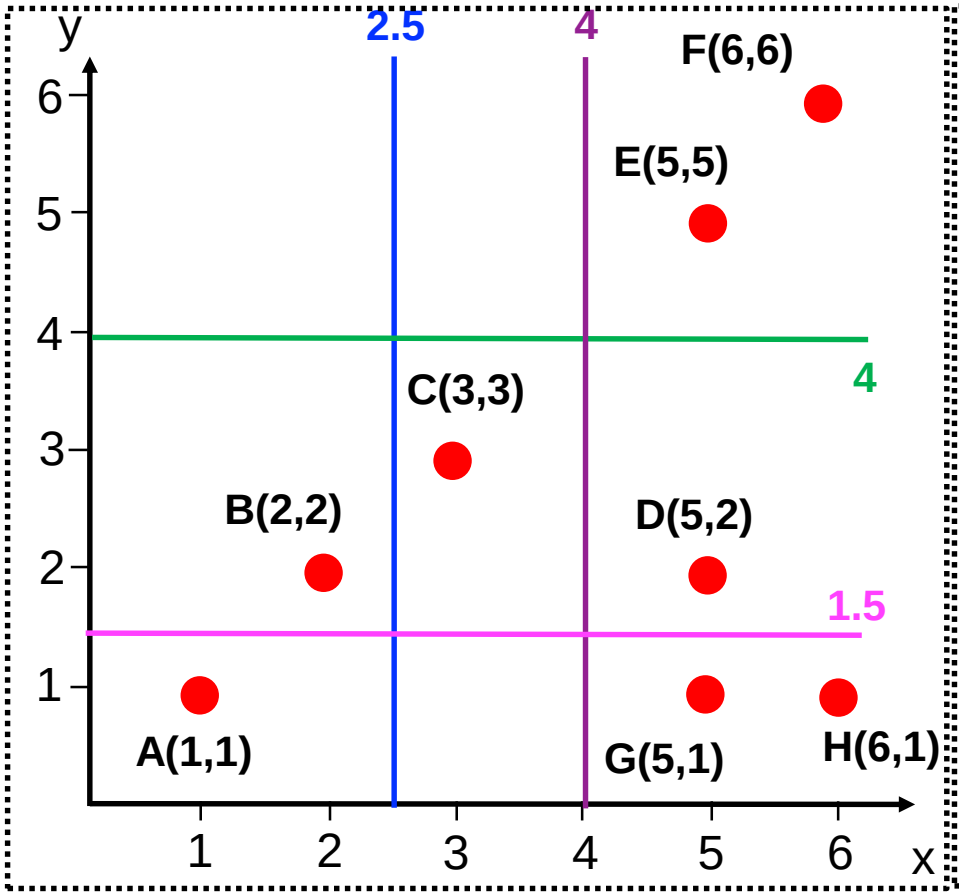
- Simple Insert Example (Blocking factor = 2)

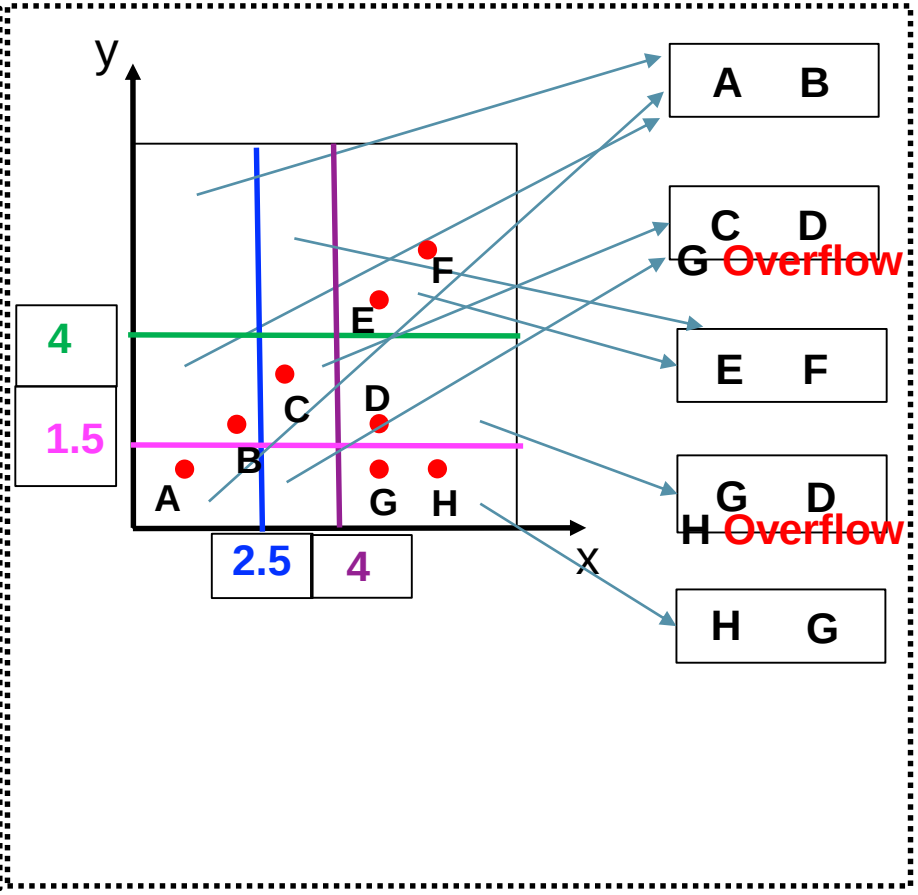




7.6.1 Grid Files

- Simple Insert Example (Blocking factor = 2)



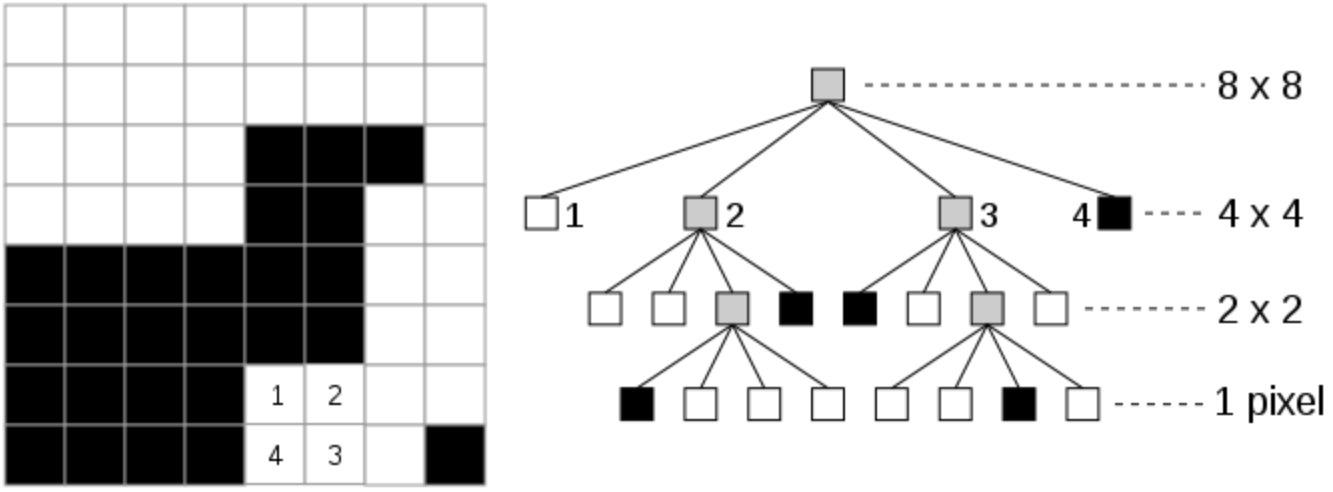


7.6.2 四叉树索引

- 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

7.6.2 四叉树索引

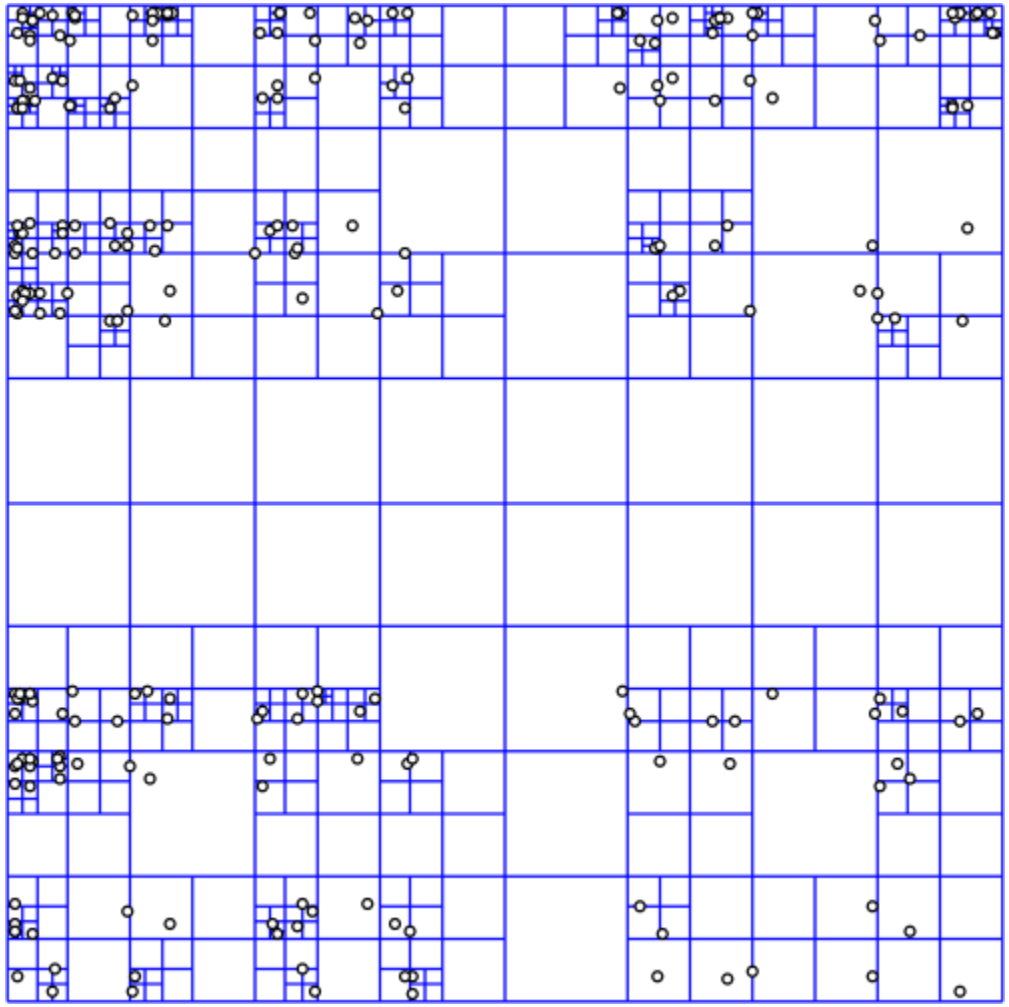
- 四叉树举例



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？

7.6.2 四叉树索引

- 四叉树举例
- 与网格差别？



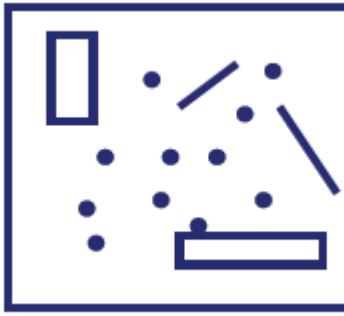
7.6.2 四叉树索引

- 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

7.6.3 R-Tree

7.6.3.1 Main idea: R-Tree, R+Tree, node format
7.6.3.2 Algorithms: insertion/split, deletion
7.6.3.3 Search: object, range, nn, spatial joins

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

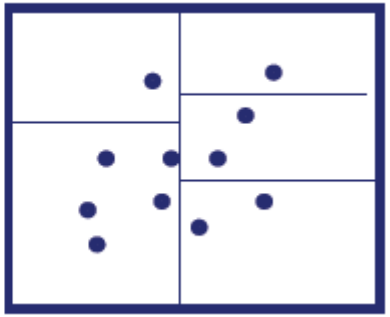


- Z-Curve: cuts regions to pieces -> dup. eli
- How would we avoid that?
- Idea: try to extend/merge B-trees and k-d trees
  - 网格：均匀划分、满(2^n 叶节点)四叉树
  - 四叉树：均匀划分、自适应划分
  - k-d树：非均匀划分、自适应划分

k-d trees等知识在图形学课程中介绍

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



7.6.3.1 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

思考：几何属性为主键在某些DBMS中可能会报什么错误？

  - Use a hierarchical collection of rectangles to organize spatial data
  - Generalizes B-tree to spatial data sets

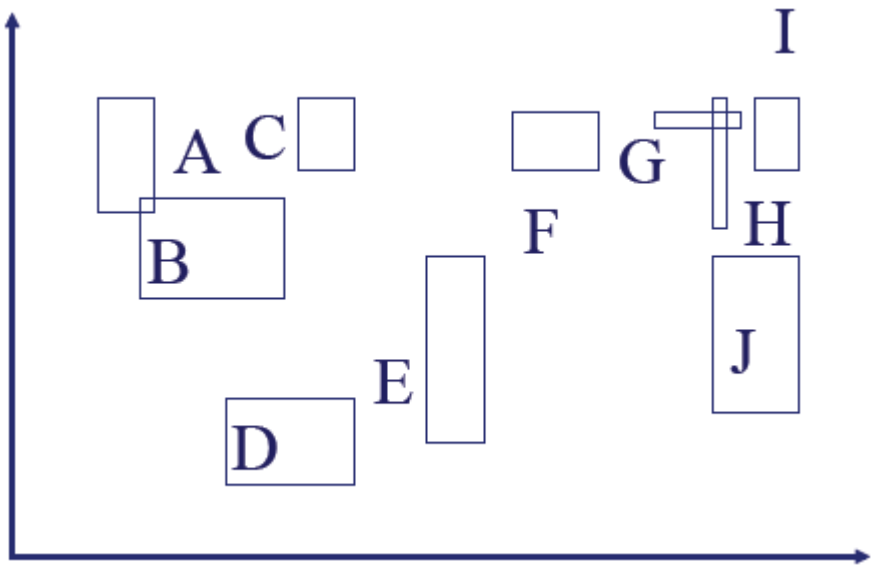
- Classifying members of R-tree family

分类标准：几何对象在页节点是否重复

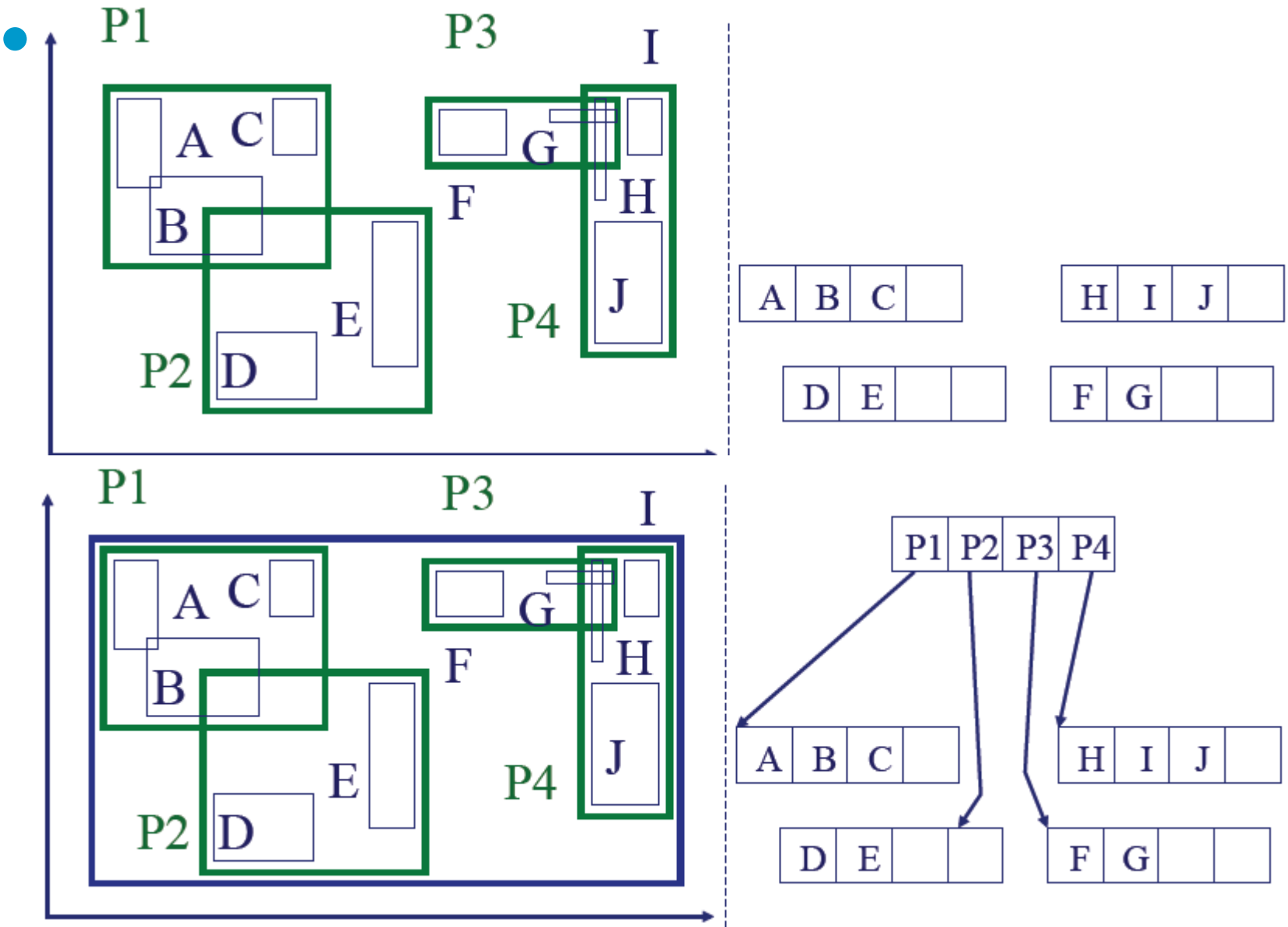
  - Handling of large spatial objects
    - Allow rectangles to overlap - R-tree
      - Grid, Quadtree, Z-Curve划分的区域都不相互重叠
    - Duplicate objects but keep interior node rectangles disjoint - R+tree
      - 几何对象会在Grid, Quadtree, Z-Curve划分的区域上重复
  - Selection of rectangles for interior nodes
    - Greedy procedures - R-tree, R+tree
    - Procedure to minimize coverage, overlap - packed R-tree
  - Other criteria exist

R-Trees – Main Idea

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

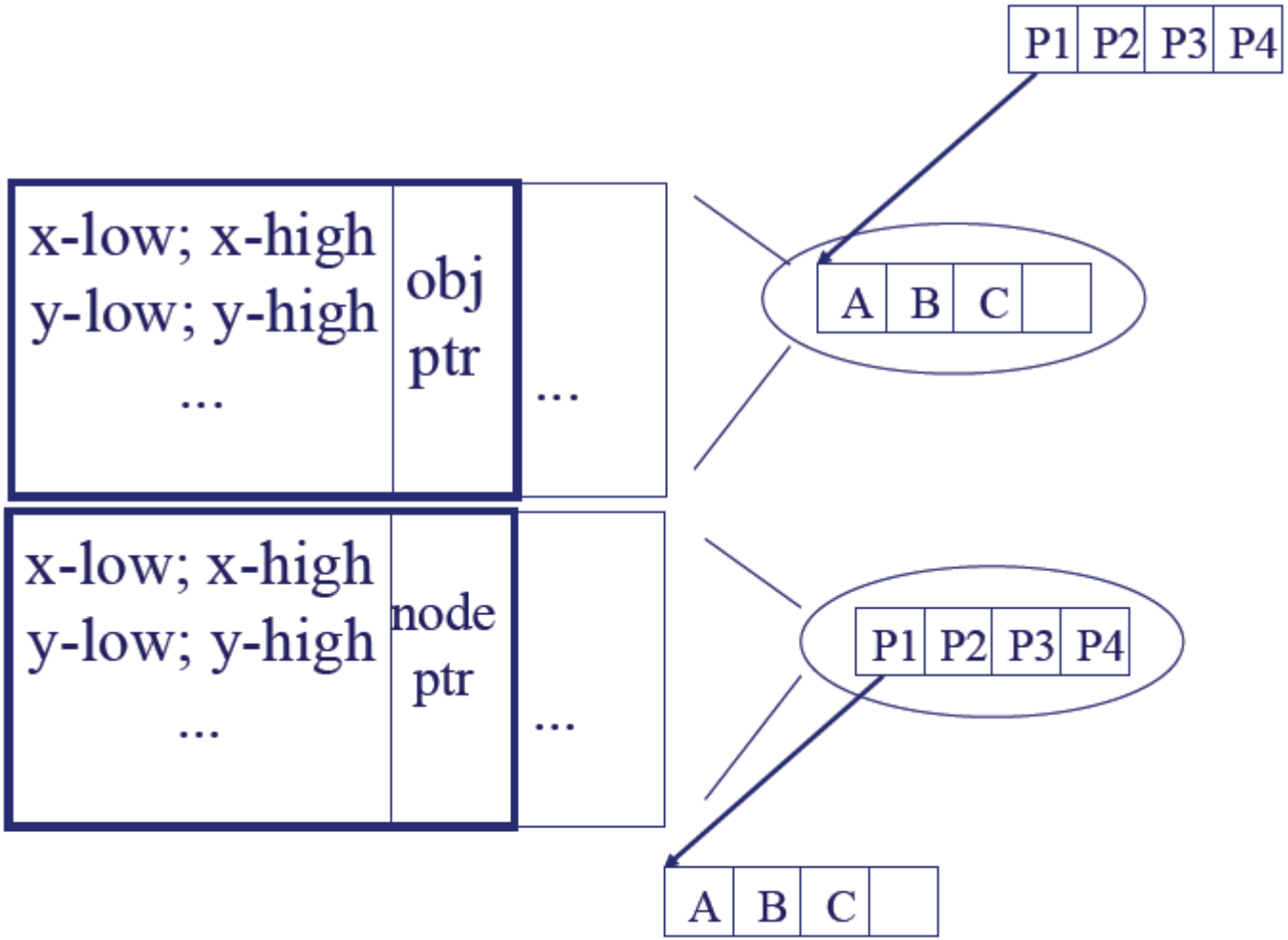


R-Trees – Main Idea



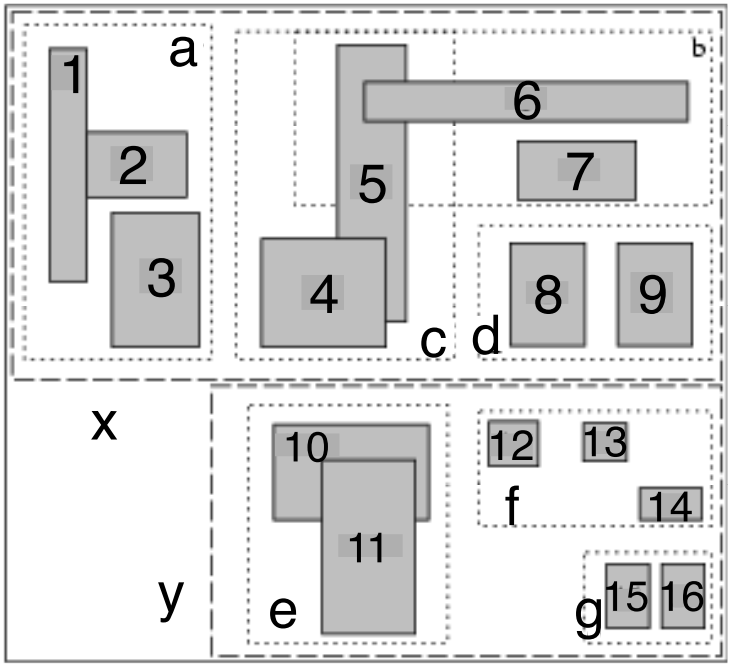
R-Trees – Node Format

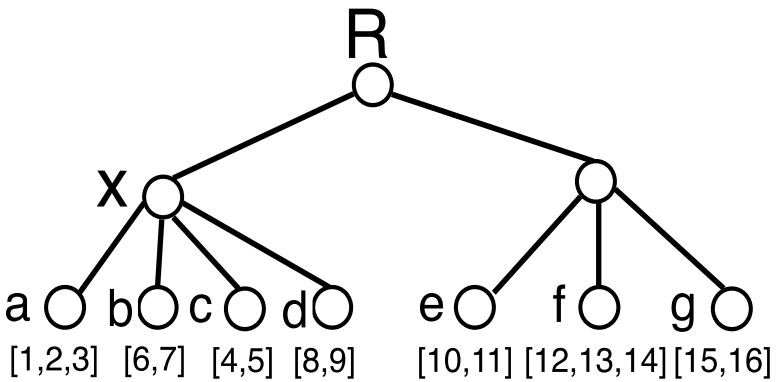
- {(MBR; obj-ptr)} for leaf nodes
- {(MBR; node-ptr)} for non-leaf nodes



R-Trees – Properties

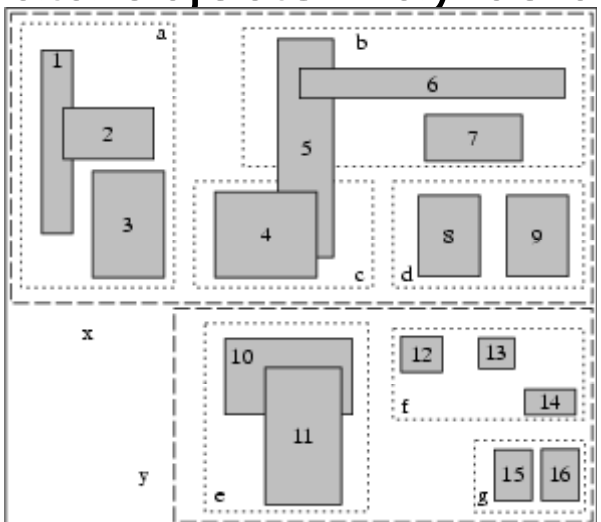
- Properties of R-trees
  - Balanced
    - Nodes are rectangle
    - Child's rectangle within parent's
  - Possible overlap among rectangles!

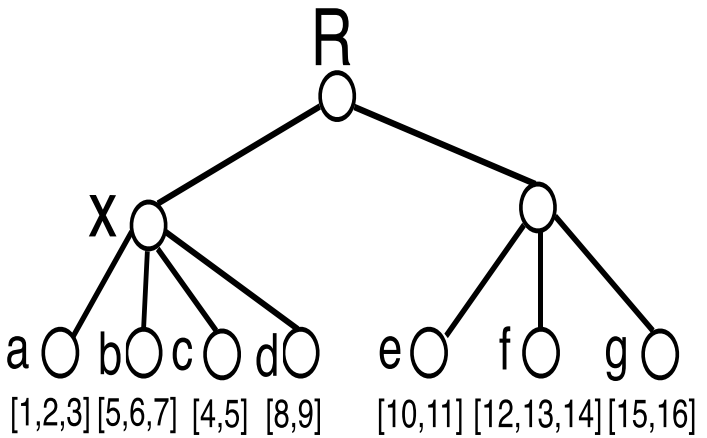




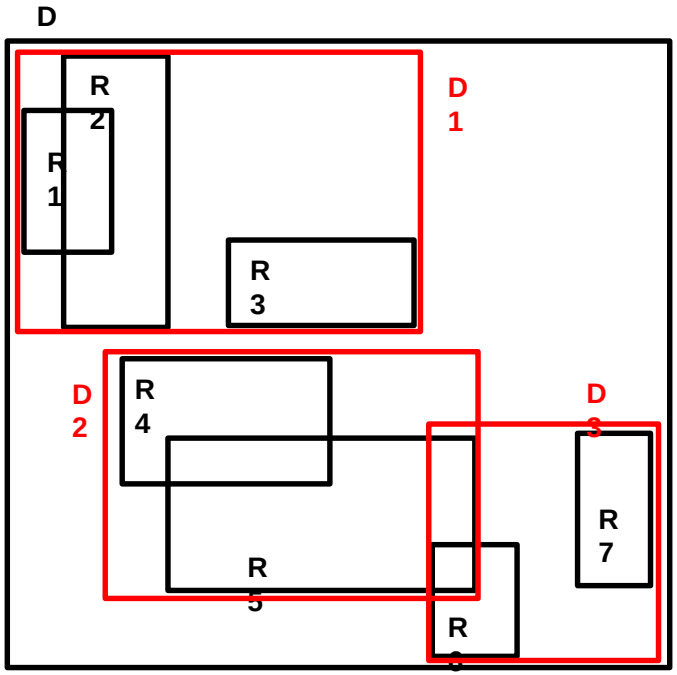
R+Trees – Properties

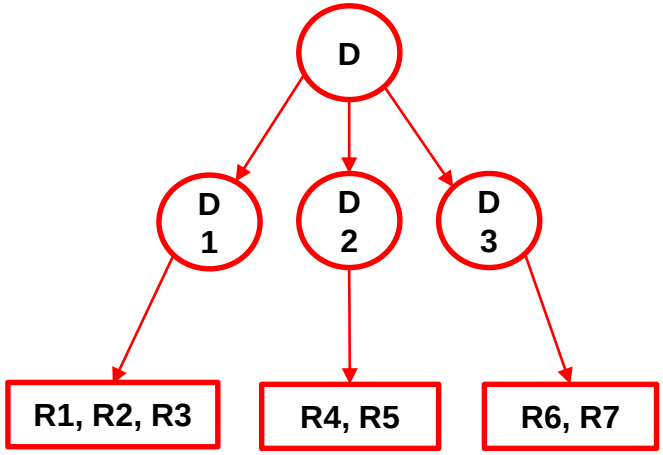
- Properties of R+trees
  - Balanced
  - Interior nodes are rectangle
    - Child's rectangle within parent's
    - Disjoint rectangles
  - Leaf nodes - MBR of polygons or lines
    - Leaf's rectangle overlaps with parent's
  - Data objects may be duplicated across leafs



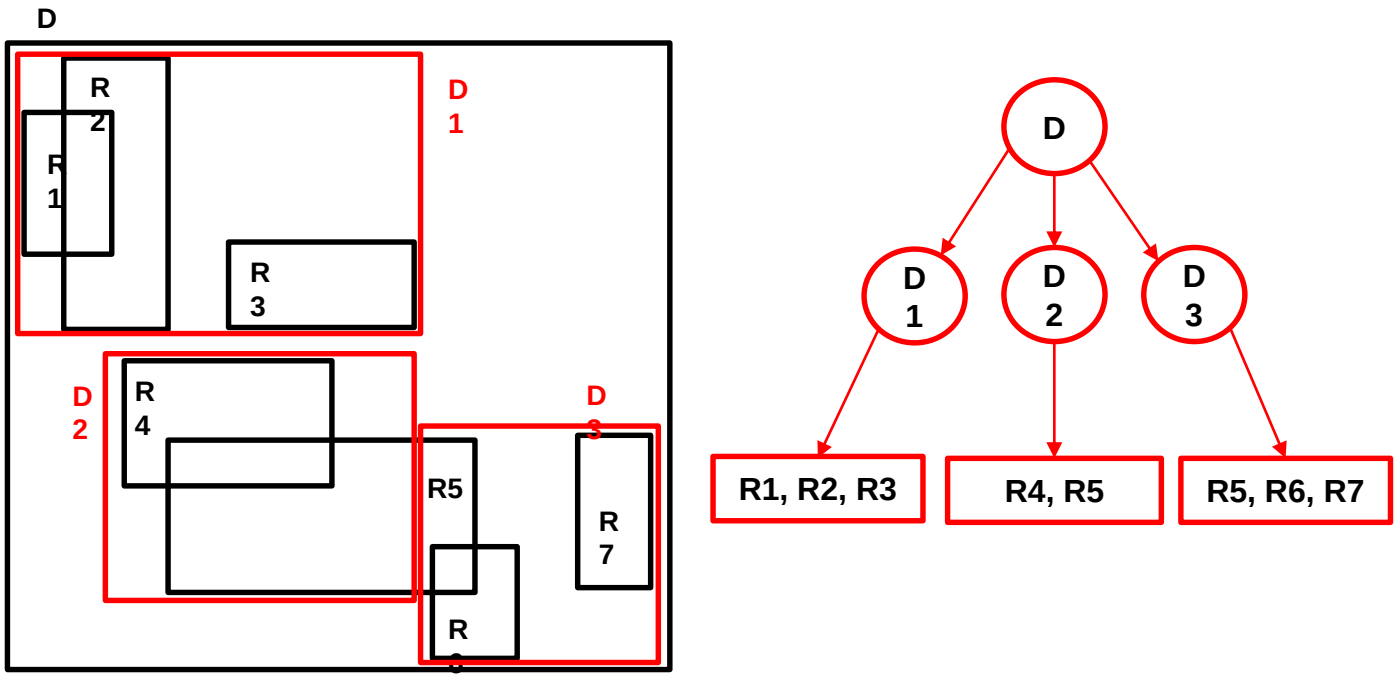


Example: R-Tree



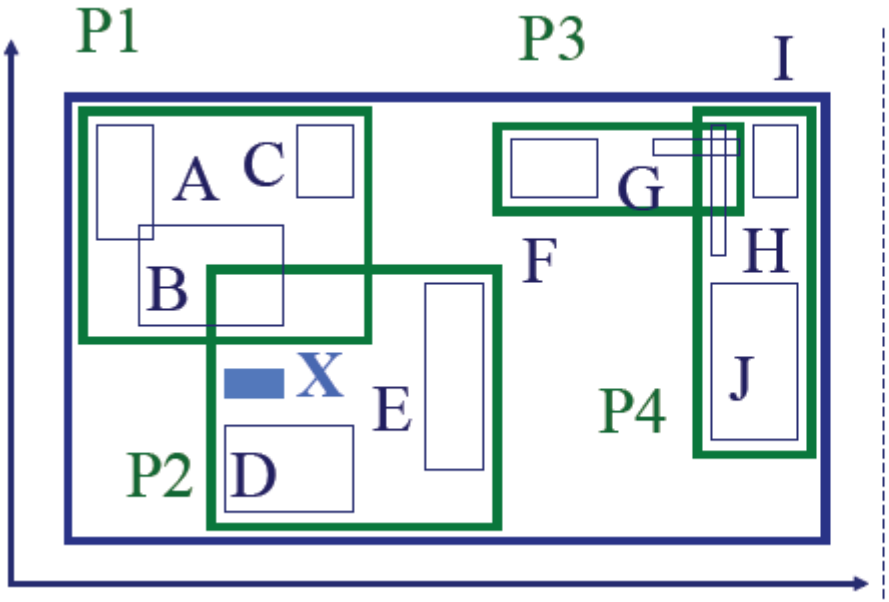


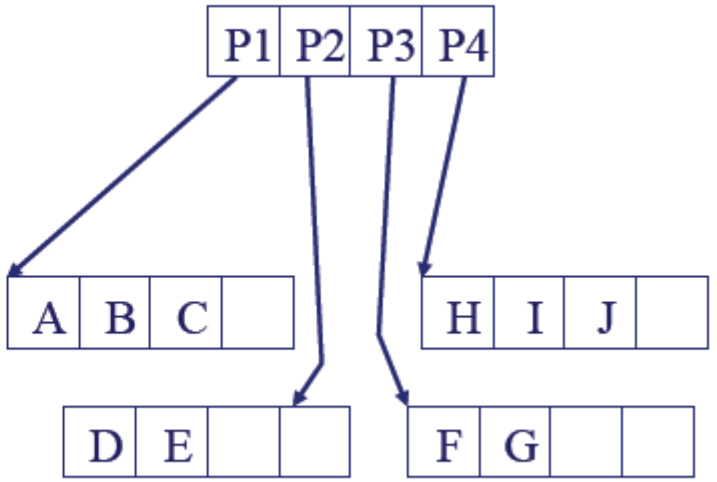
Example: R+Tree

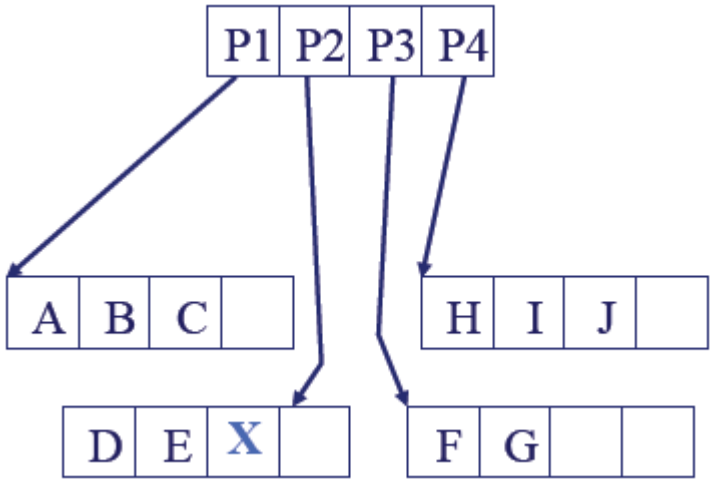


7.6.3.2 R-Trees – Insertion

- Eg., rectangle ‘X’

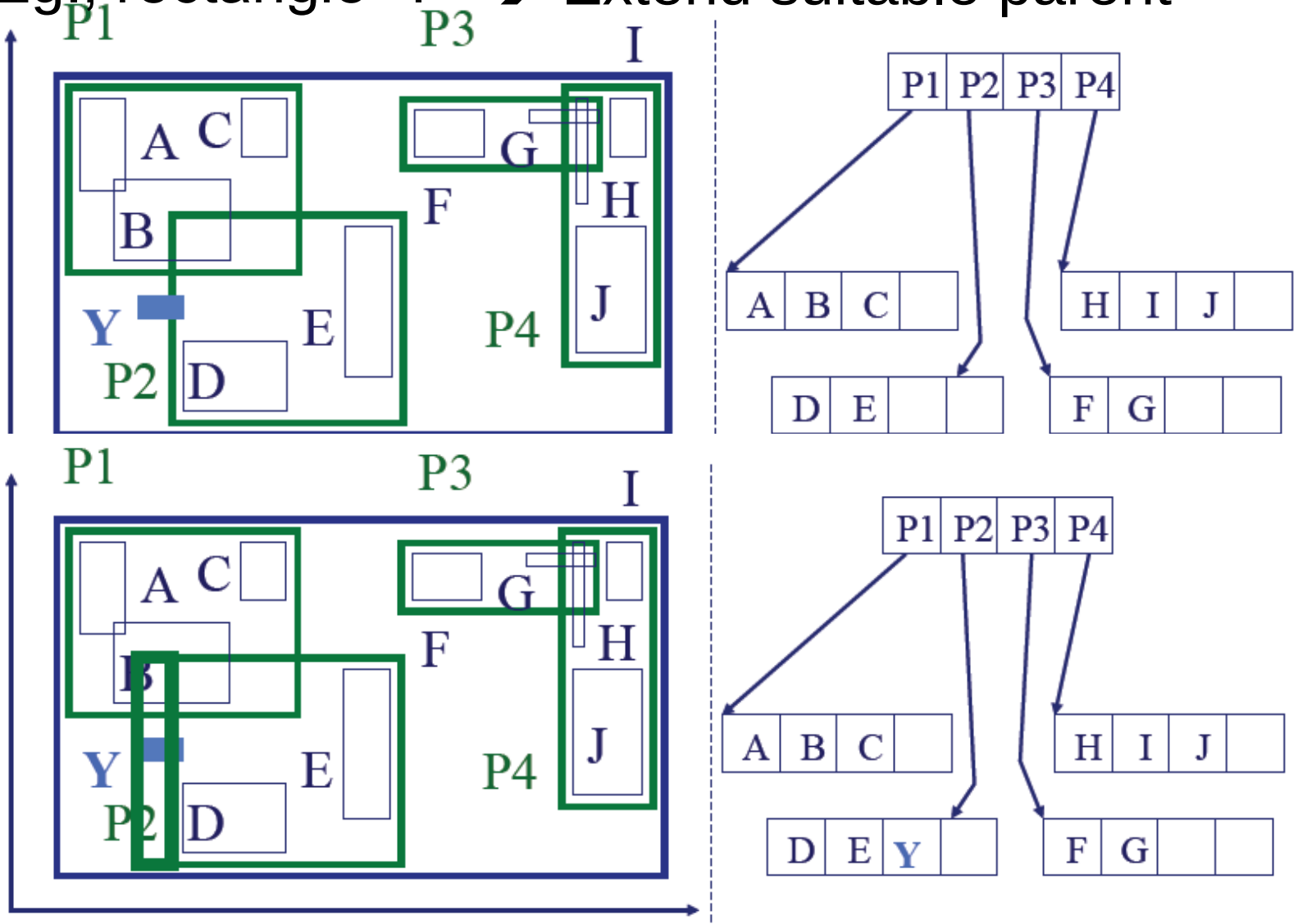






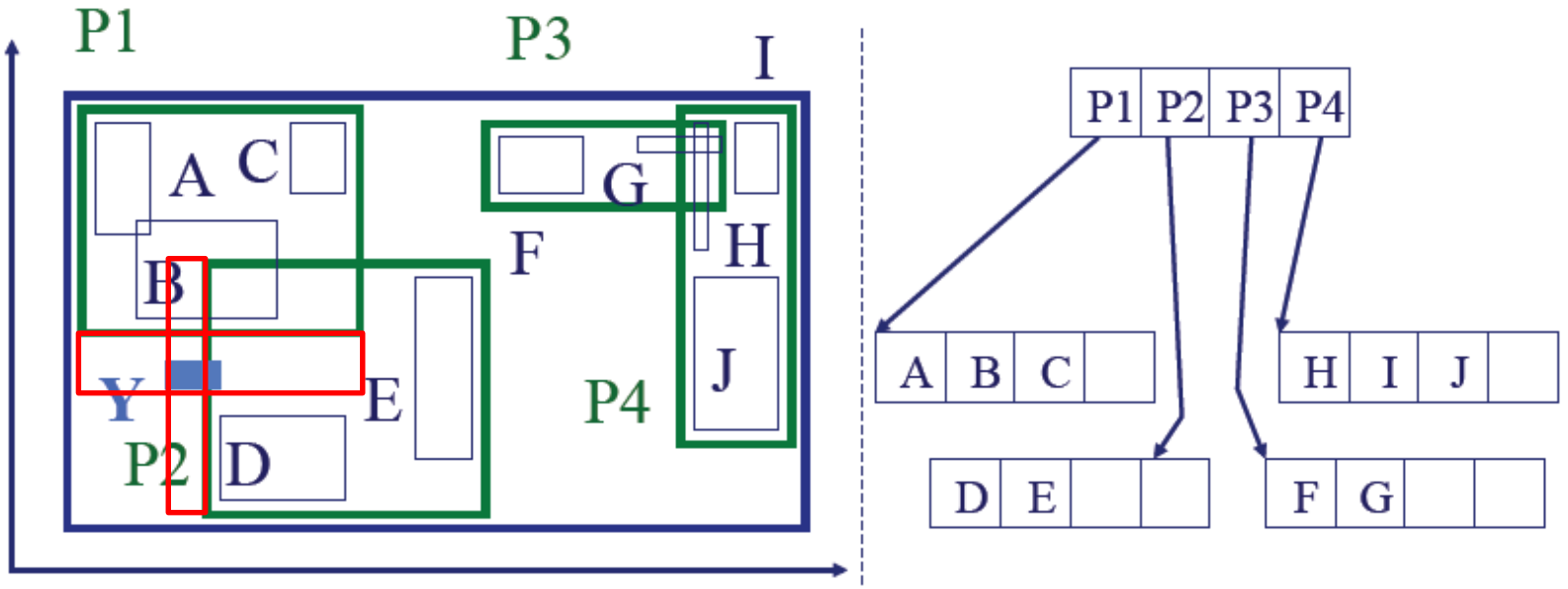
R-Trees – Insertion

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



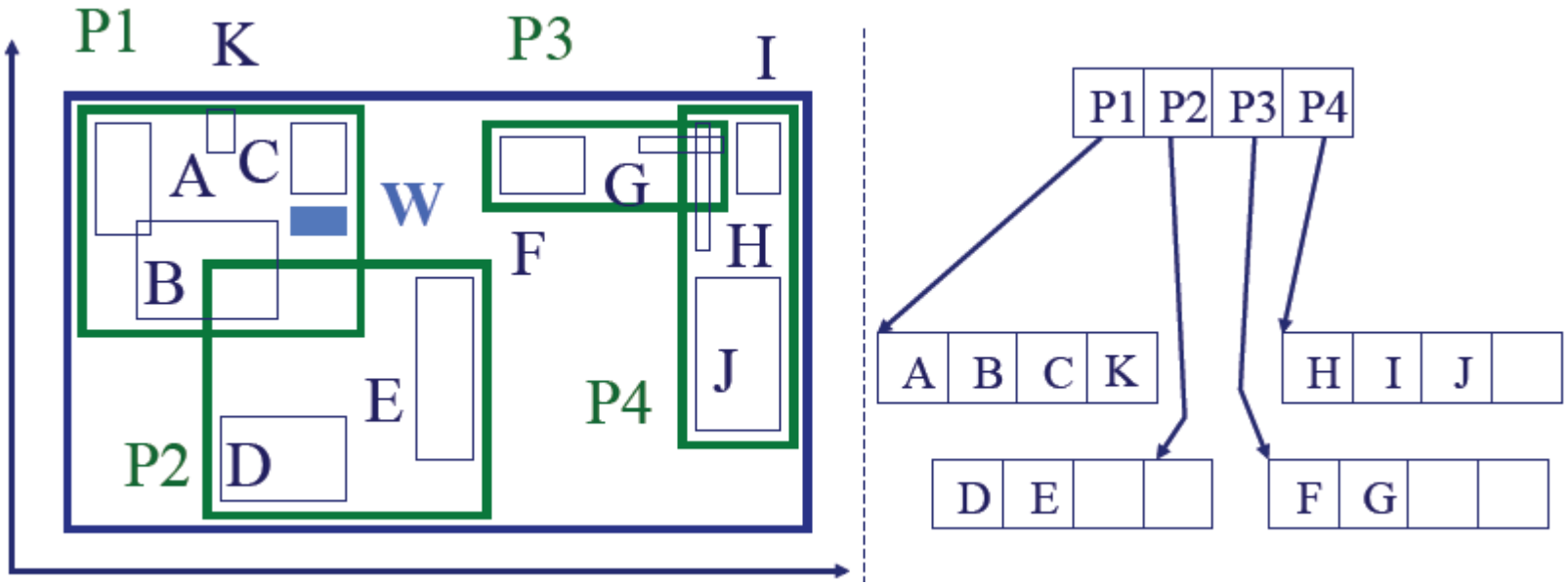
R-Trees – Insertion

- Eg., rectangle ‘Y’: extend suitable parent
- Q: how to measure ‘suitability’?
- A: increase in area (volume)
  - 节点新增加的面积越小越好



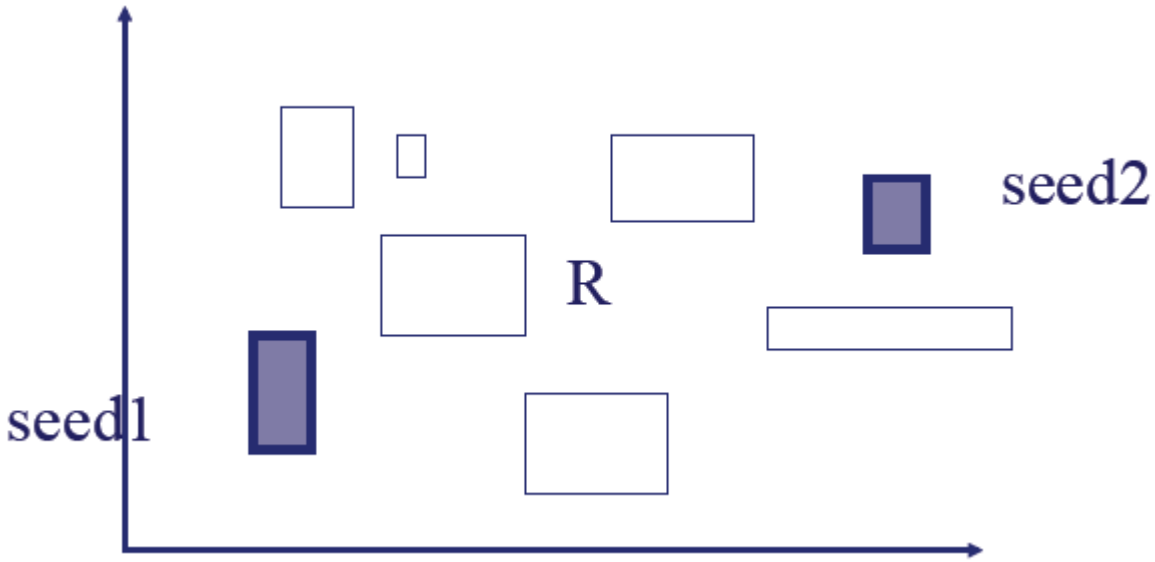
R-Trees – Insertion

- Q: what if there is no room? how to split?
- 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

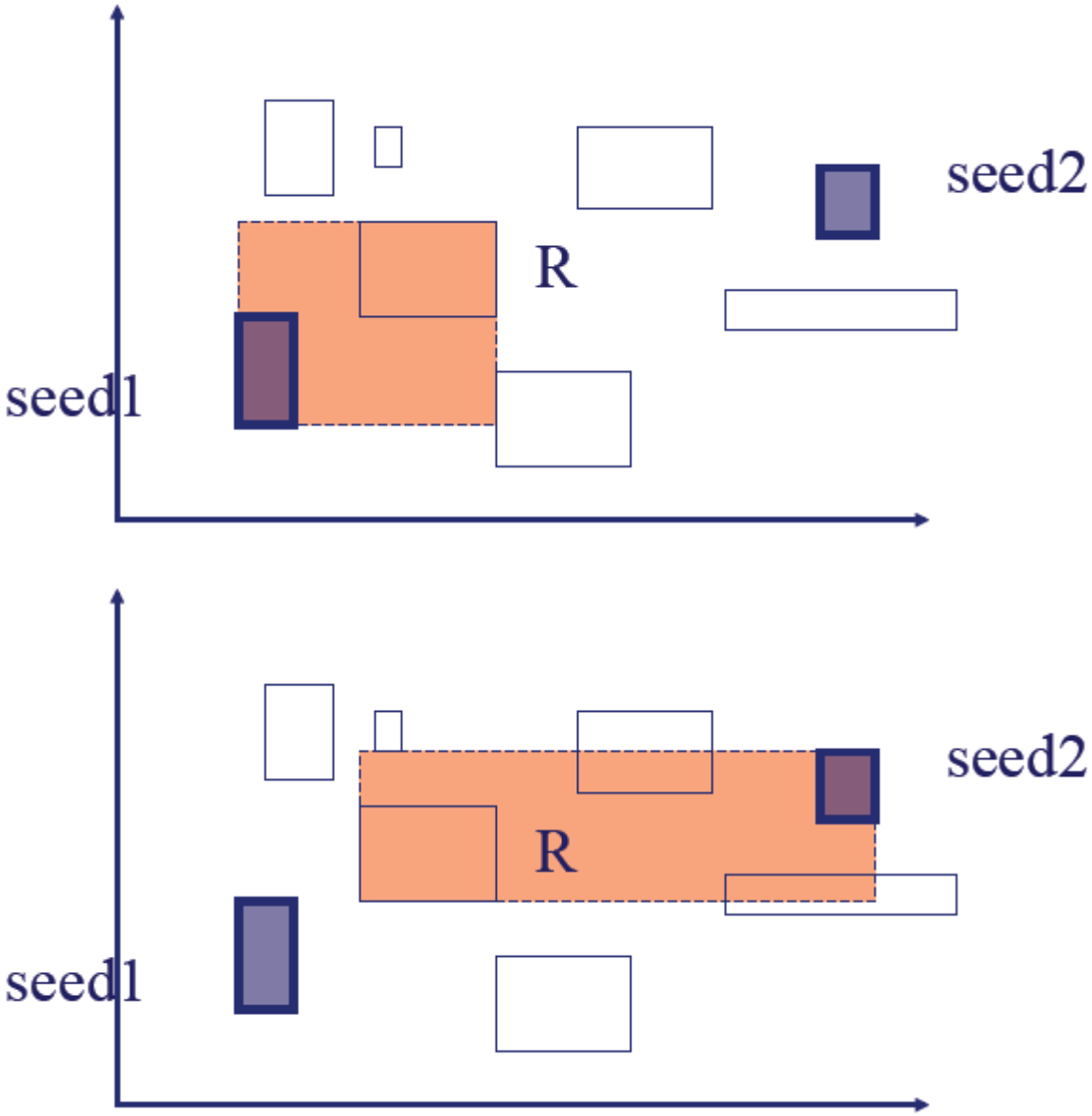


R-Trees – Insertion & Split

- Pick two rectangles as ‘seeds’;
- Assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- Q: how to measure ‘closeness’?
- A: 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

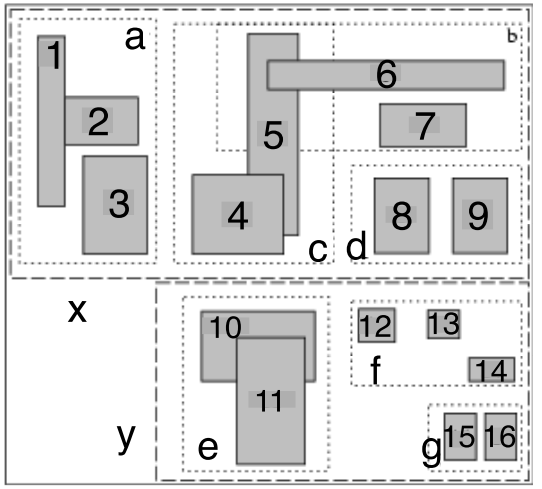
- Observations
  - Many more split algorithms exist

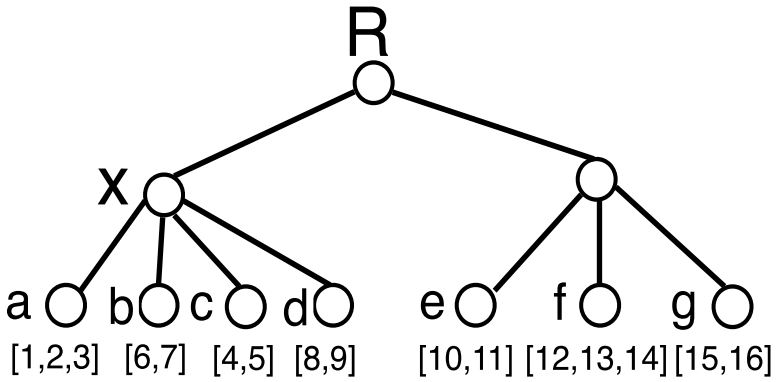
R-Trees – Deletion

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

7.6.3.3 R-Trees – Object Search

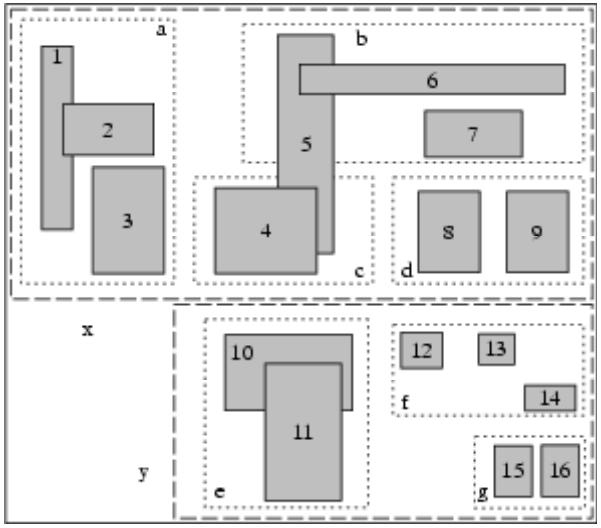
- Implementation of find operation
  - Search root to identify relevant children
  - Search selected children recursively
- Exercise: find record for rectangle 5
  - Root search identifies child x
  - Search of x identifies children b and c
  - Search of b does not find object 5
  - Search of c find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

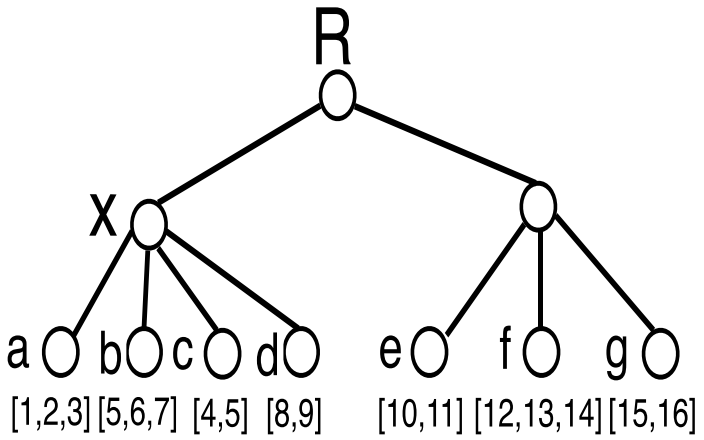




R+Trees – Object Search

- 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

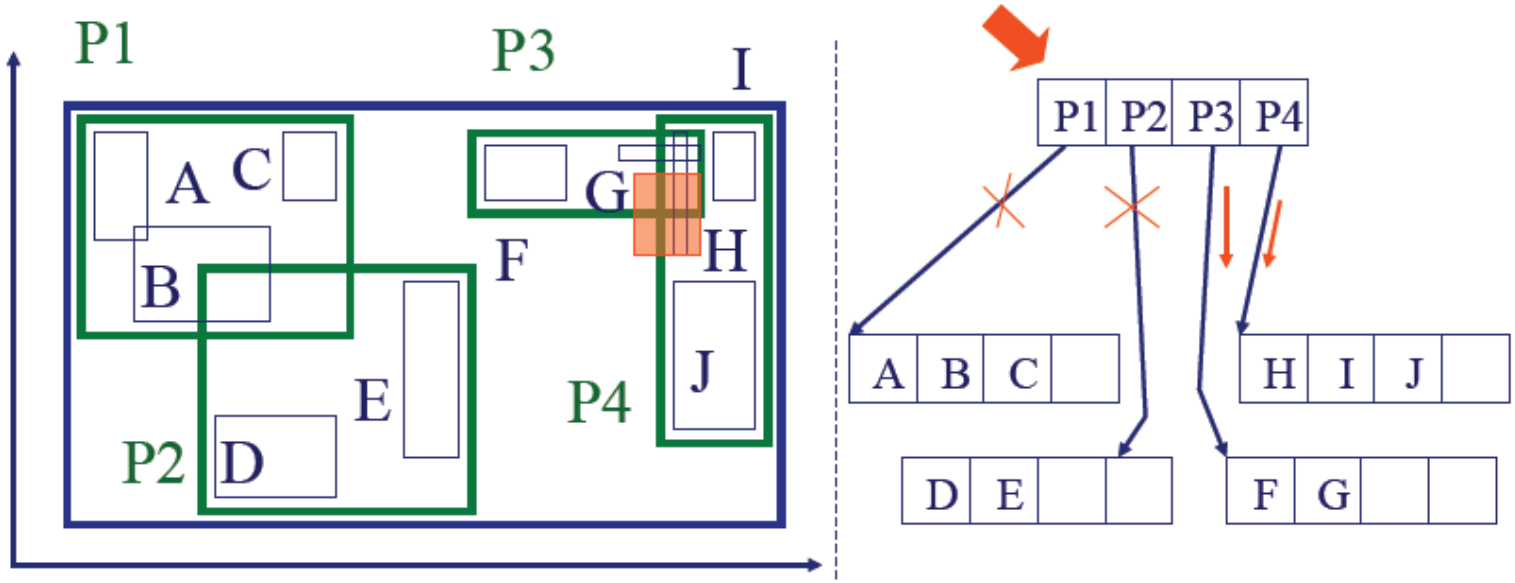




空间关系查询时，先比较包围盒，再真正空间关系比较

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

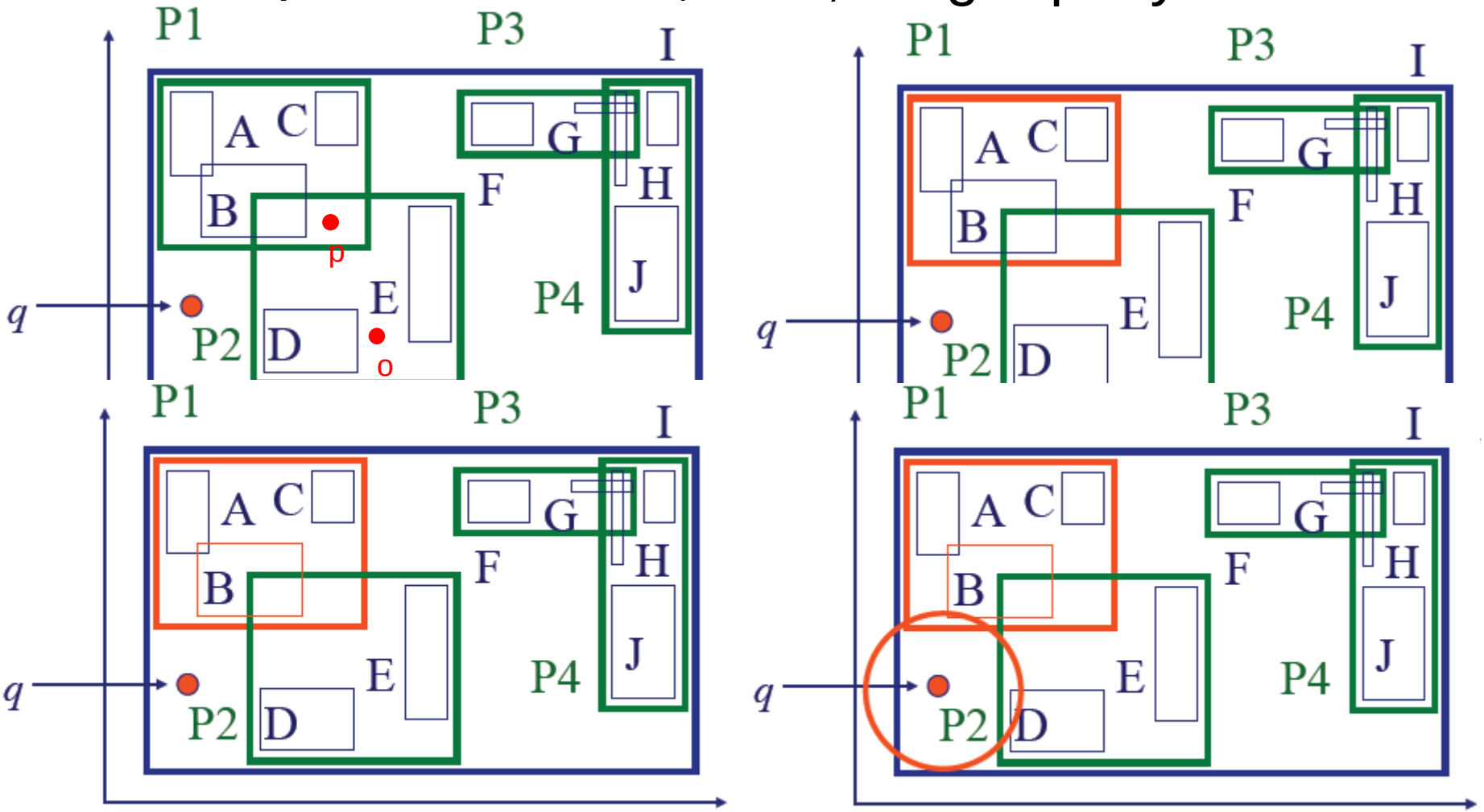


R-Trees – Range Search

- 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

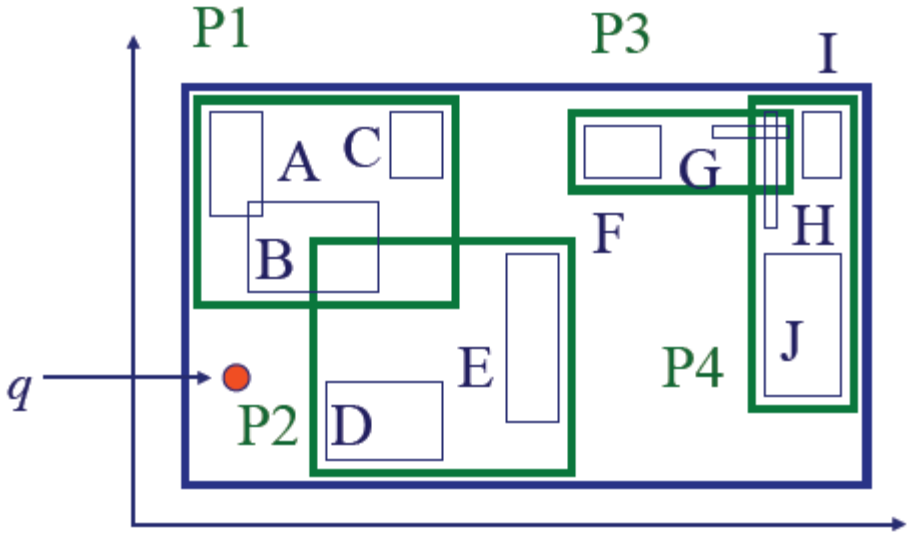
R-Trees – NN Search 思考：range query的半径合理吗？

- 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

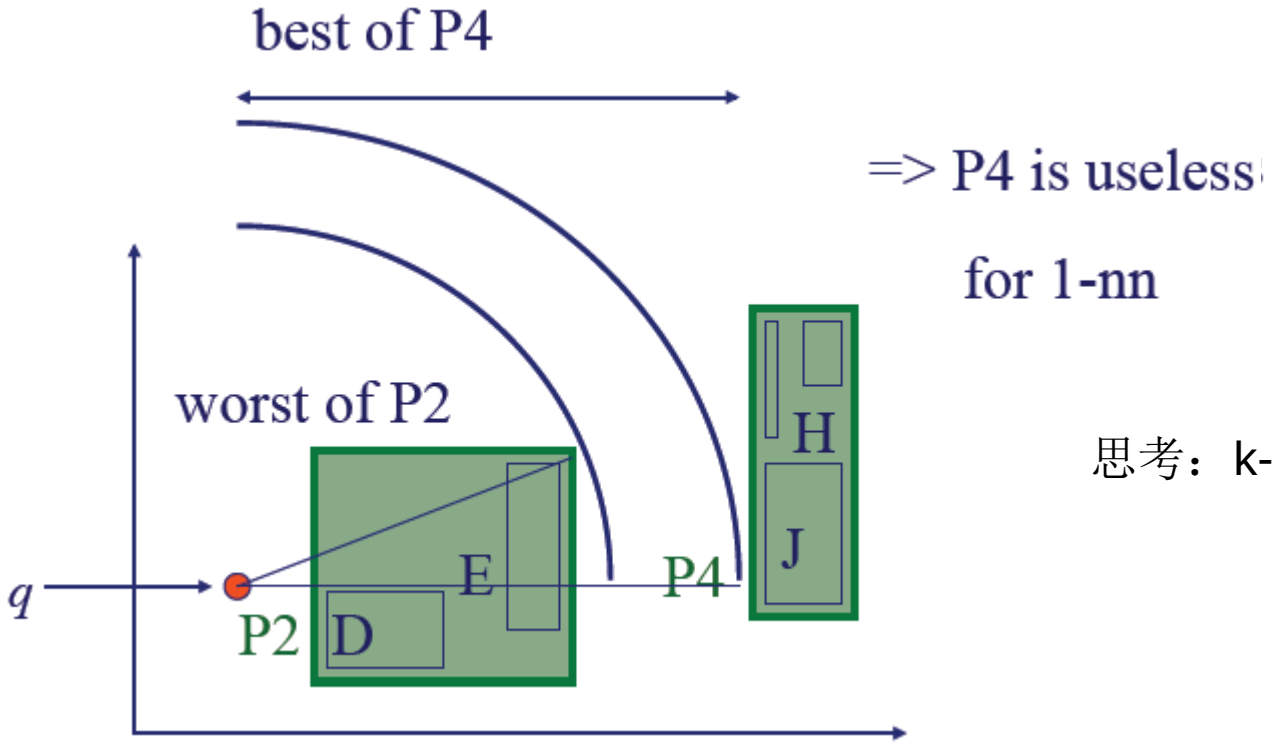


思考：q只需和4个节点判断距离，为什么？

与四叉树不同，所有叶节点都有几何要素

R-Trees – NN Search

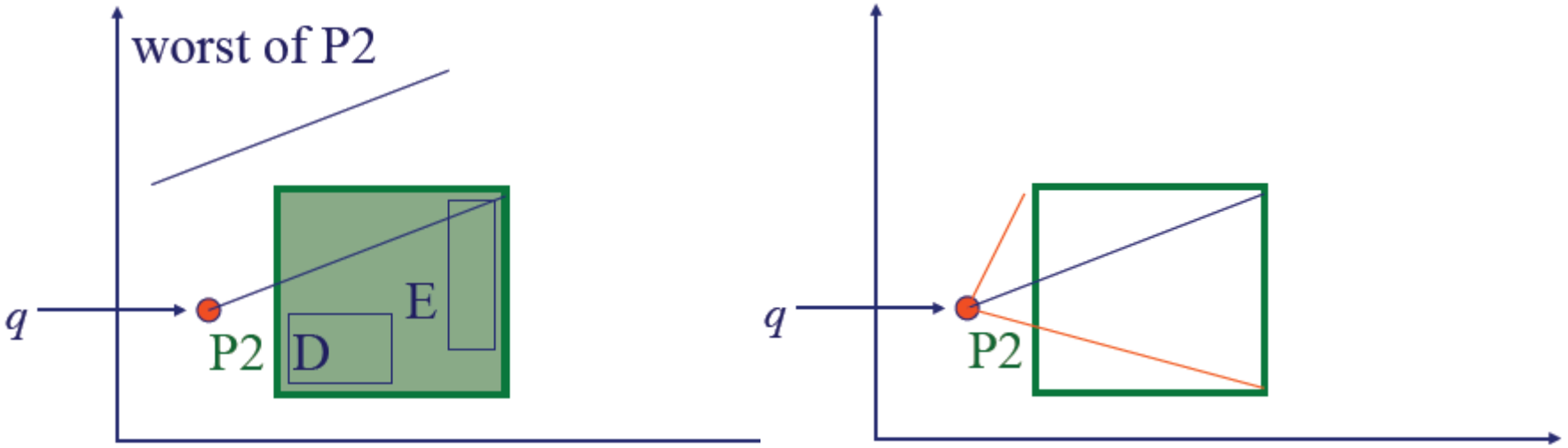
- A2: [Roussopoulos+, sigmod95]:
  - Priority queue, with promising MBRs, and their best and worst-case distance



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!



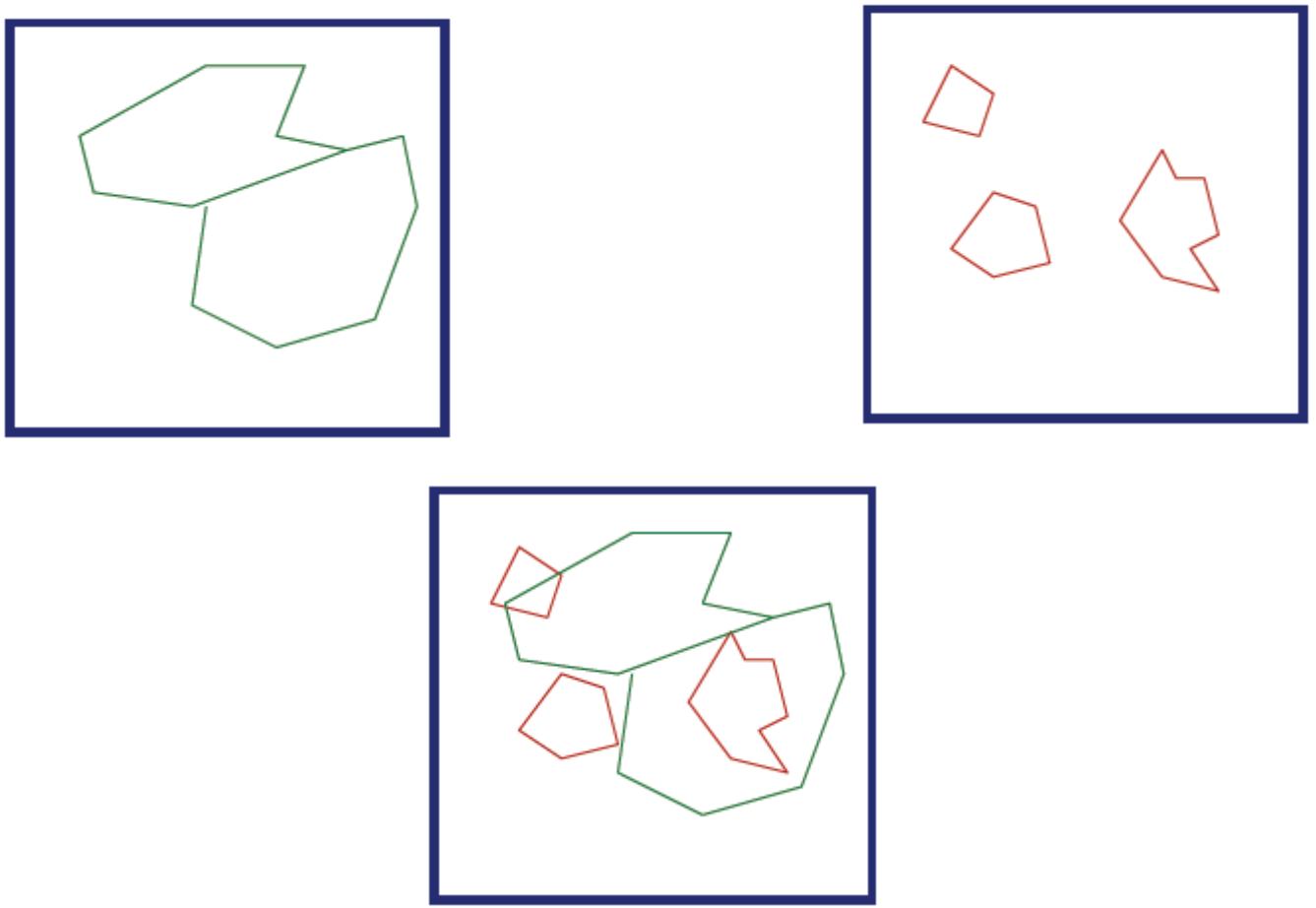
R-Trees – NN Search

- 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

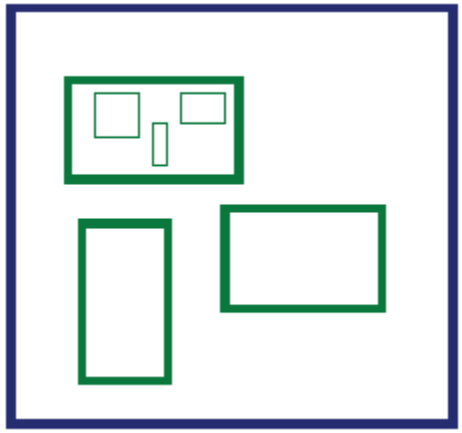
- Spatial joins: find (quickly) all

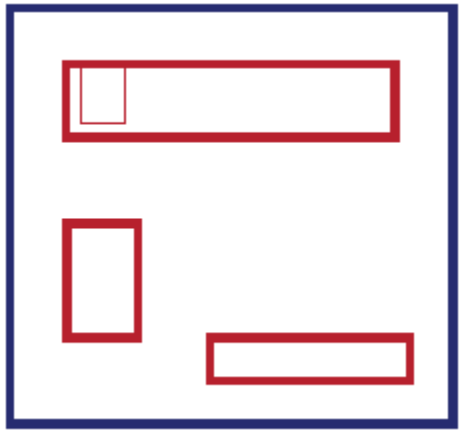
counties intersecting lakes

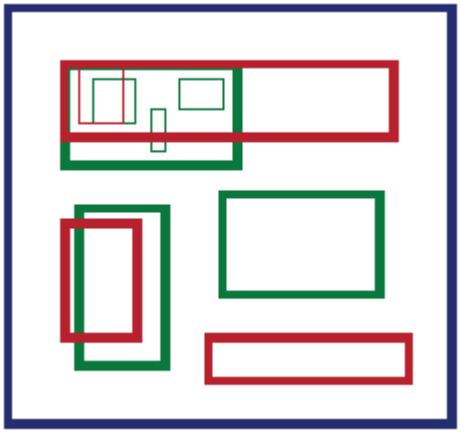


R-Trees – Spatial Joins

- Assume that they are both organized in R-trees







R-Trees – Spatial Joins

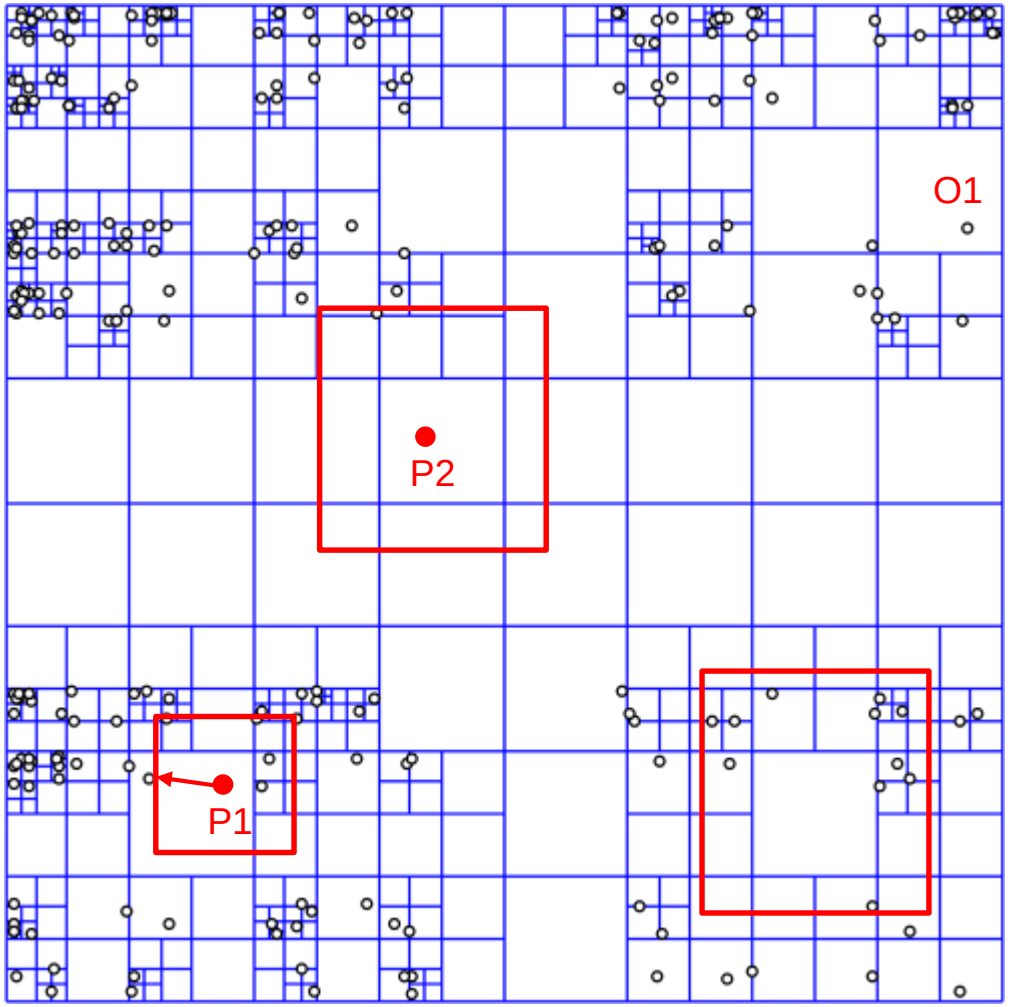
- for each parent P1 of tree T1
  - for each parent P2 of tree T2
    - if their MBRs intersect,
      - process them recursively (ie., check their children)

R-Trees – 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

四叉树索引

- 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



第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

参考教材：
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

PostGIS中的索引

- B-trees
  - Spatial data can be sorted along a space-filling curve, Z-order curve or Hilbert curve
- GiST (Generalized Search Trees)
  - Break data into “things to one side”, “things which overlap”, “things which are inside”
  - Use an R-Tree index implemented on top of GiST
- BRIN (Block Range Index)
  - Store only the bounding box englobing all the geometries contained in all the rows in a set of table blocks
- SP-GiST (Space-Partitioned GiST)
  - Many overlapping objects

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

PostGIS中的索引

```
SQL 窗口
-- Index: public.ir

-- DROP INDEX public.ir;

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

与数值索引比较

- 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
- 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

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

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)

第七章 空间存储与索引

- 7.1 物理数据模型
- 7.2 数据物理存储
- 7.3 数据文件组织
- 7.4 数值索引
- 7.5 空间填充曲线 (重用关系数据库物理模型)
- 7.6 空间索引 (新的空间技术)
- 7.7 PostGIS空间索引
- 7.8 总结

参考教材：
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

空间存储与索引总结

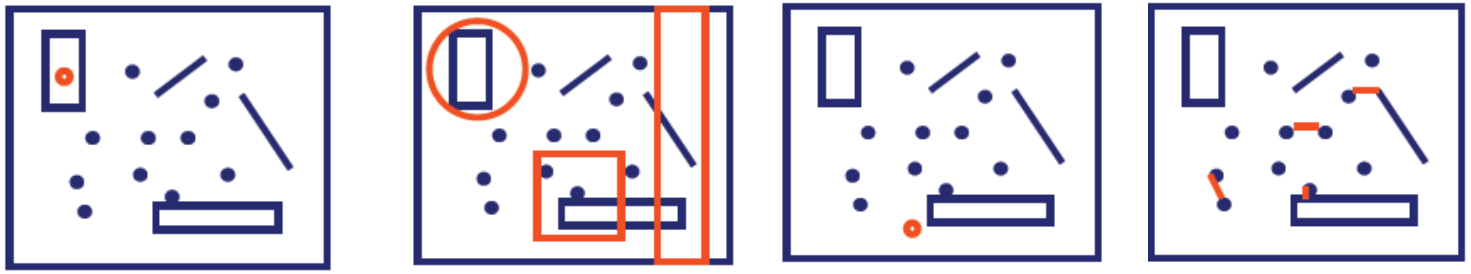
- Physical data models in relational database
  - Sequential scan is much faster than random reads (2%)
  - Total cost = I/O cost + CPU cost
  - Data file: heap (unordered), ordered, hashed, clustered
  - Data file vs. index file
  - Indices: B-Tree, Hash table, …
  - Primary key vs. index key, clustered index vs. unclustered index
- Query plan: Nested loop join / Merge join / Hash join
  - 查询规划：explain / explain analysis

空间存储与索引总结

- Two approaches to support spatial data and queries
  - Reusing relational physical data model concepts
    - Space filling curves define a total order for points, such as Z-Curve, Hilbert Curve
    - This total order helps in using ordered files, search trees
  - New spatial techniques
    - Spatial indices, such as grids, quadtree, R-Tree
    - Provide better computational performance

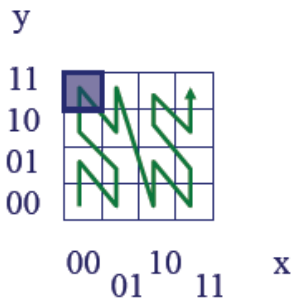
空间存储与索引总结

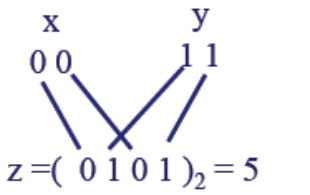
- Common Queries
  - Point query: Find all rectangles containing a given point
  - Range query: Find all objects within a query rectangle
  - Nearest neighbor: Find the point closest to a query point
  - Intersection query: Find all the rectangles intersecting a query rectangle (spatial join)

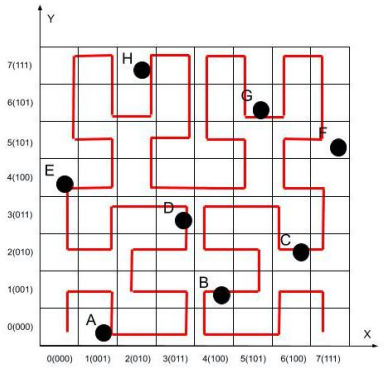


空间存储与索引总结

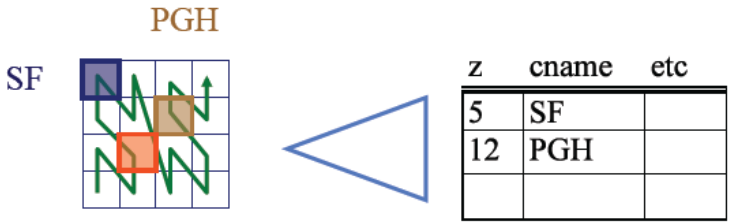
- Space-filling Curves (重用关系数据库的物理模型)
  - Z-Curve
  - Hilbert Curve
  - Sorted file + B+tree





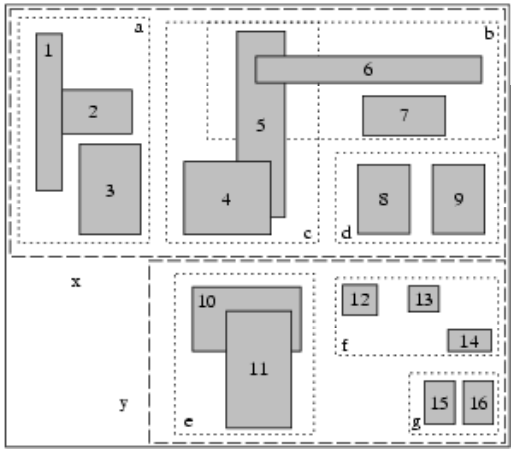


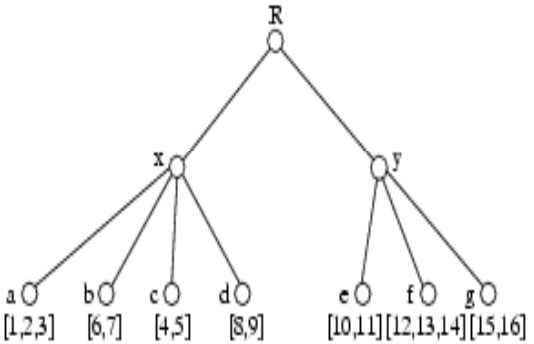
  - Point query
    - Calculate z/h value, search B+tree
  - Range query
    - Calculate z/h value, search the smallest value in B+tree, traverse
  - NN
    - Calculate z/h value, search B-tree, traverse, range query
  - Spatial join
    - Merge the list of (sorted) z/h-values, looking for the prefix property



空间存储与索引总结

- R-Tree
  - Properties
    - Node overlapping
  - Insertion/Deletion
    - Increase in area
  - Point query
    - Traverse R-Tree (multiple children nodes)
  - Range query
    - Traverse R-Tree (multiple children nodes)
  - NN
    - Depth-first search, calculate the range, range query
  - Spatial join
    - Nested loop join, traverse R-Tree





空间存储与索引总结

- 创建索引
  - Create [unique][clustered] Index <索引名> On <表名>(<属性列名>)
    - 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
  - Create Index <索引名> On <表名> Using GiST (<几何列名>)
    - 能对非空间数据属性构建空间索引？
- 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)
  - 能够利用空间索引的函数：Equals, Intersects, Touches, Crosses, Within, Contains, Overlaps, DWithin
    - 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

空间存储与索引总结

- 如何选择属性索引？
  - 看join，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
- 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

空间存储与索引总结

- 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