第三章：

文本检索技术（2）

杨建武

北京大学计算机科学技术研究所

Email:yangjianwu@icst.pku.edu.cn

检索系统介绍

Smart系统

- Smart retrieval system
- ftp://ftp.cs.cornell.edu/pub/smart/
- 最著名的、使用者最多的实验系统之一
  - 历史比较长，20世纪80年代第一个版本
  - 使用免费，且可下载源代码
- 美国康奈尔大学研发
  - 最初的研发工作由Gerard Salton教授领导
  - 后续维护工作由Chris Buckley负责
  - 最新的版本是Smart11(99年)

Smart系统

- 实现了一个完整的基于向量空间模型的文本信息检索系统
- G. Salton ed. The SMART Retrieval System—Experiments in Automatic Document RetrievalEnglewood Cliff, NJ: Prentice Hall Inc.; 1971
- G. Salton, M.J. McGill Introduction to Modern Information Retrieval McGraw-Hill Book Co. New York, NY, USA 1986

$$Sim(Q,D) = \frac{\sum_{k=1}^{t}(w_{qk} \cdot w_{dk})}{\sqrt{\sum_{k=1}^{t}(w_{qk})^2 \cdot \sum_{k=1}^{t}(w_{dk})^2}}$$

Smart系统

- 开发Smart系统的目的是为了给文本信息检索技术的研究者提供一个完善的实验平台
  - 对一组文档建立索引
  - 可对给出的查询(query)返回检索结果
  - 对结果进行评价
  - 去除stopwords（stopwords可由用户指定）
  - 去除词形变化(stemming)
  - weighting计算等
  - 用户可以根据自己的需要分别调用。

Smart系统

- 缺点
  - 只能处理大约500MB以下的文档集合，这使它在数据量达到10GB以上的TREC Web Track这样的问题时显得力不从心
  - 缺乏良好的设计说明文档，使用者常常需要自己摸索使用方法

Okapi系统

- http://www.soi.city.ac.uk/~andym/OKAPI-PACK/index.html
- 伦敦城市大学开发
- 20世纪80年代末第一版
- 运行在Unix系统上
- 不是免费的，并且不提供源代码
  - a nominal fee of 100 pounds sterling
  - the source code for Okapi for the sum of 1000 pounds sterling

Okapi系统

- 基于概率检索模型
- 经过近20年的发展，Okapi系统越来越健壮，检索精确度也越来越高
- 在TREC比赛中，有不少参加者采用Okapi系统取得了很好的成绩

BM2500

$$\sum_{T \in Q} w^1 \frac{(k_1+1)tf}{(K+tf)} \frac{(k_3+1)qtf}{(k_3+qft)}$$

$$w^1 = \log \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)}$$

Lemur Toolkit系统

- http://www.lemurproject.org/
- 卡耐基-梅隆大学（CMU）开发
- 在2001年公布了第一个公开的版本
- 设计目标：促进和帮助在文本信息检索和语言模型方面的研究
  - 在检索中引入了语言模型
  - 针对领域：特定目标检索、分布式检索、跨语言检索、文摘系统、信息过滤、文本分类
- 整个系统用C和C++语言实现，可在Unix和Windows系统下运行
  - C++, Java and C# APIs

Lemur Toolkit系统

- 不仅是一个完整的检索系统，而且是以工具包的形式提供的。
- 功能
  - 对大规模文本数据建立索引
  - 实现语言模型和其它多个检索模型的系统
- 各功能模块都有良好的封装，并提供清晰的源代码和丰富的文档说明，研究者使用它搭建自己的实验系统易如反掌。
- Free and open-source software
- Lemur Toolkit v 4.8 (2008/12/22)

Lucene系统

- Lucene是一个使用Java语言实现的全文索引软件包。
- 早先发布在www.lucene.com，后来发布在SourceForge
  - Lucene的贡献者Doug Cutting是一位资深全文索引/检索专家
    - V-Twin搜索引擎(Apple的Copland操作系统的成就之一)的主要开发者，
    - 在Excite担任高级系统架构设计师。
- 2001年年底成为APACHE基金会jakarta的一个子项目更多的人参加到Lucene的开发中来
- http://jakarta.apache.org/lucene/

Lucene系统

- Lucene是一个全文索引软件包，不是一个完整的全文索引应用，但是可以方便的嵌入到各种应用中实现针对应用的全文索引/检索功能；
- 针对Web应用，编程接口简单，但功能强大；
- 支持增量索引、多字段索引，支持各种文档格式（html,pdf,word,mp3等）；
- 提供丰富的查询功能；
- 支持多语言，也可以编写解析器(Analyzer)扩展支持其他语言。

Lucene系统

- 应用广泛
  - 微软的Outlook搜索插件用Lucene架构
  - 苹果的iTunes
  - Chinabbs

Lucene系统

- 2002年Doug Cutting开启了一个新的开源项目:nutch，基于Lucene实现的搜索引擎
  - Java开发，是Lucene Project的一个子项目，一个完整的Web搜索引擎，
    - 包括web采集，web内容分析，链接分析，分布式文件系统(NDFS)，索引和检索(采用Lucene)
  - 可扩展，基于插件式架构：
    - URL Normalizers and Filters 插件
    - 网络协议插件（HTTP,FTP等）
    - 分析器插件
    - 索引和查询插件
  - NDFS(Nutch Distributed File System)
    - 基于Google File System
    - 采用Google提出的MapReduce

Lucene的其他版本

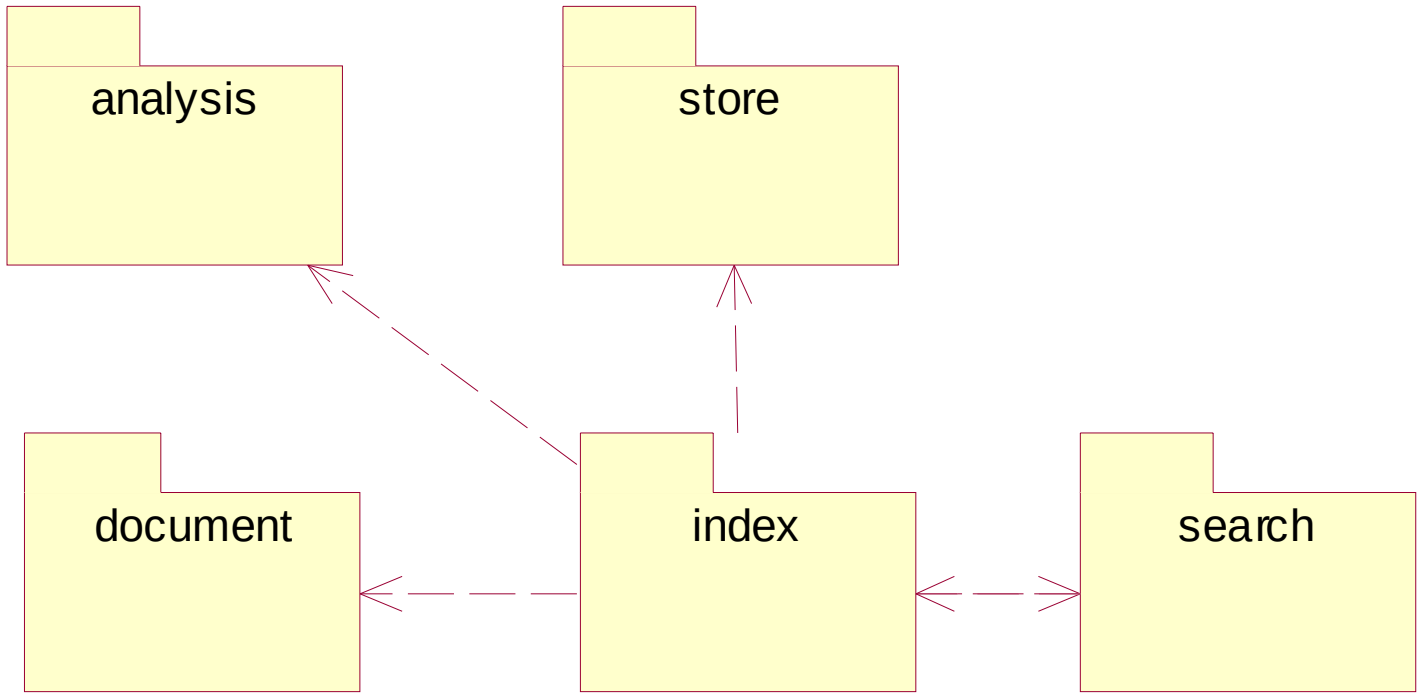
- Lucene的其他版本：
  - Lucene.Net to the C# and .NET platform utilizing Microsoft .NET Framework.
  - C++版（CLucene，非官方开发，代码可读性不是很好）

Lucene的缺点

- Lucene是由Java语言实现的，Java的解析执行方式限制了Lucene的性能
- Lucene的部分索引算法、索引合并算法和检索算法不够高效
- Lucene主要面向实际应用，在研究领域，要使用Lucene做实验还是比较困难和麻烦的
- 中文支持，目前官方只提供了简单的单字切分的分析器。

Lucene介绍

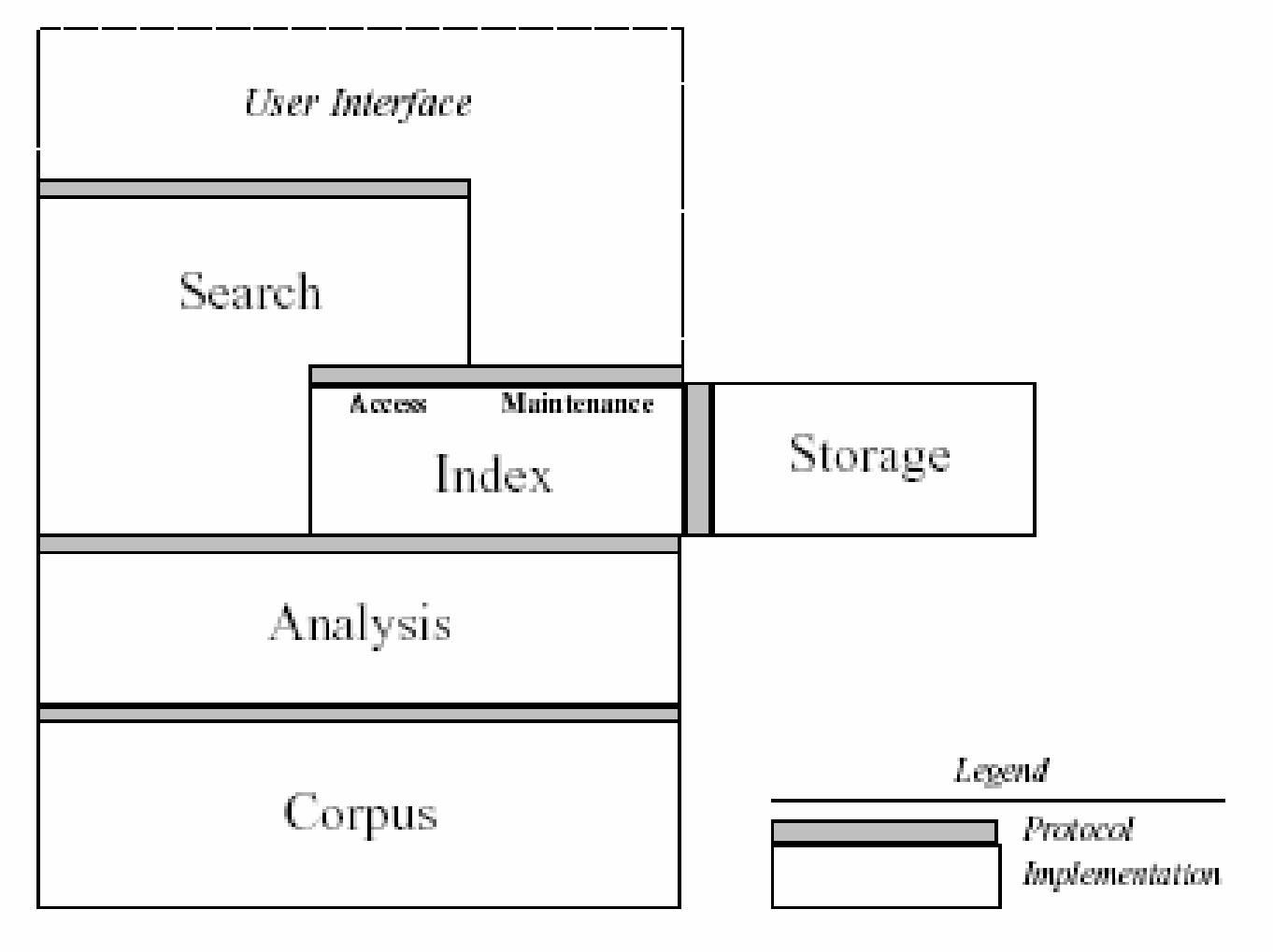
Lucene模块间依赖关系



Structure of source code

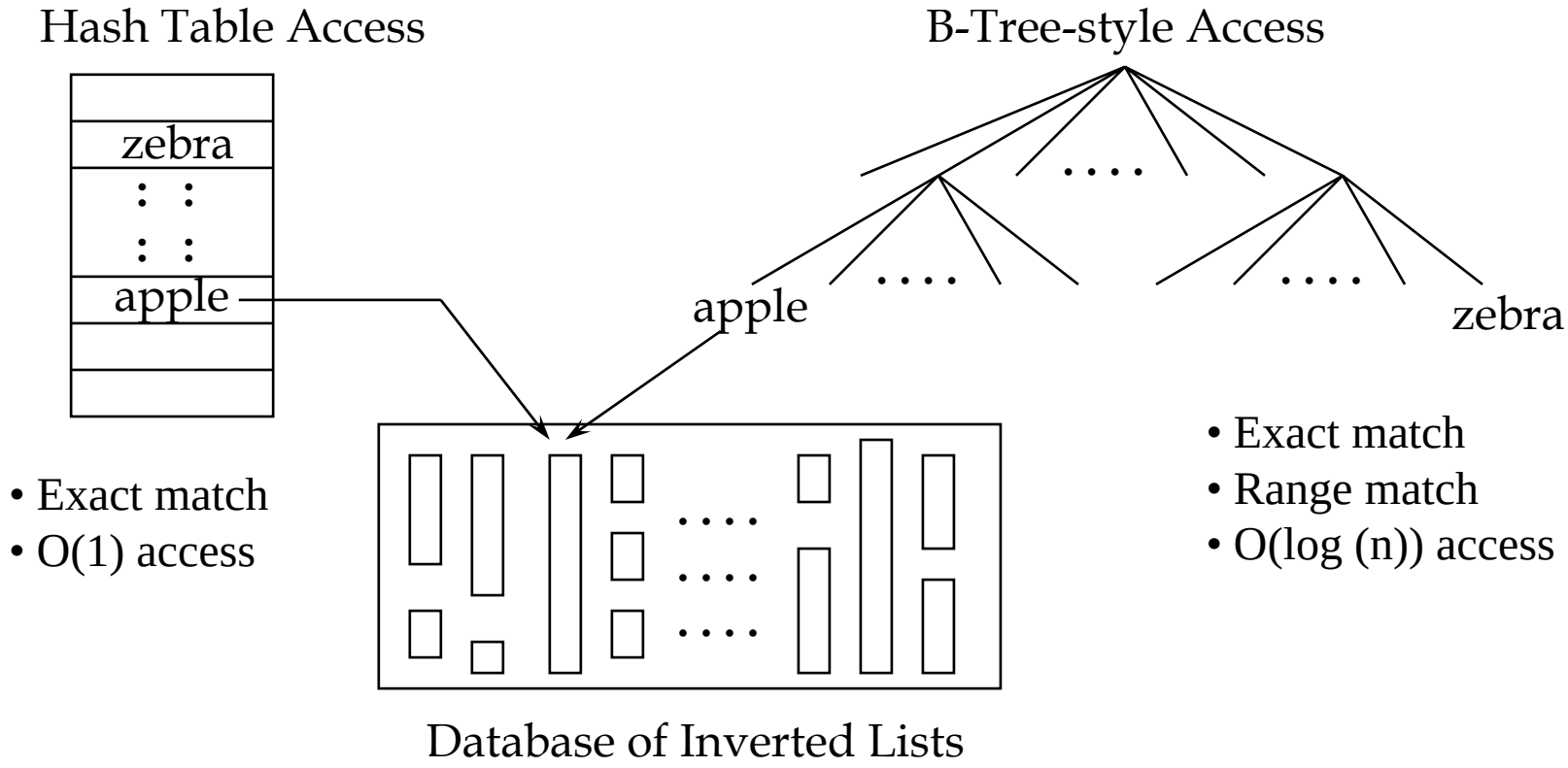
- org.apache.Lucene.index/ 索引入口
- org.apache.Lucene.search/ 检索入口
- org.apache.Lucene.analysis/ 语言分析器
- org.apache.Lucene.queryParser/查询分析器
- org.apache.Lucene.document/ 存储结构
- org.apache.Lucene.store/ 底层IO/存储结构
- org.apache.Lucene.util/ 公用数据结构

系统框架

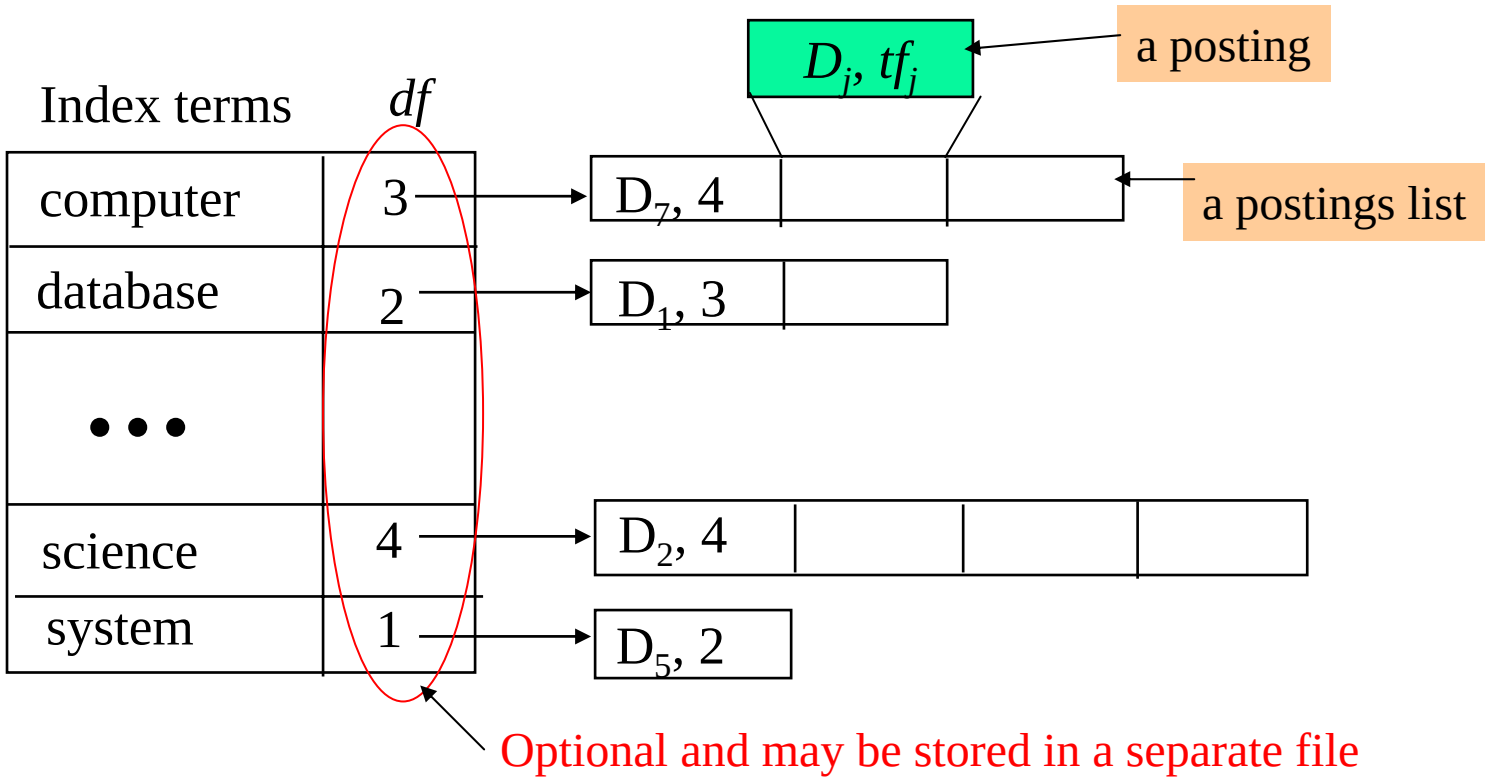

User Interface
Search
Access
Maintenance
Index
Storage
Analysis
Corpus
Legend
Protocol
Implementation

倒排索引

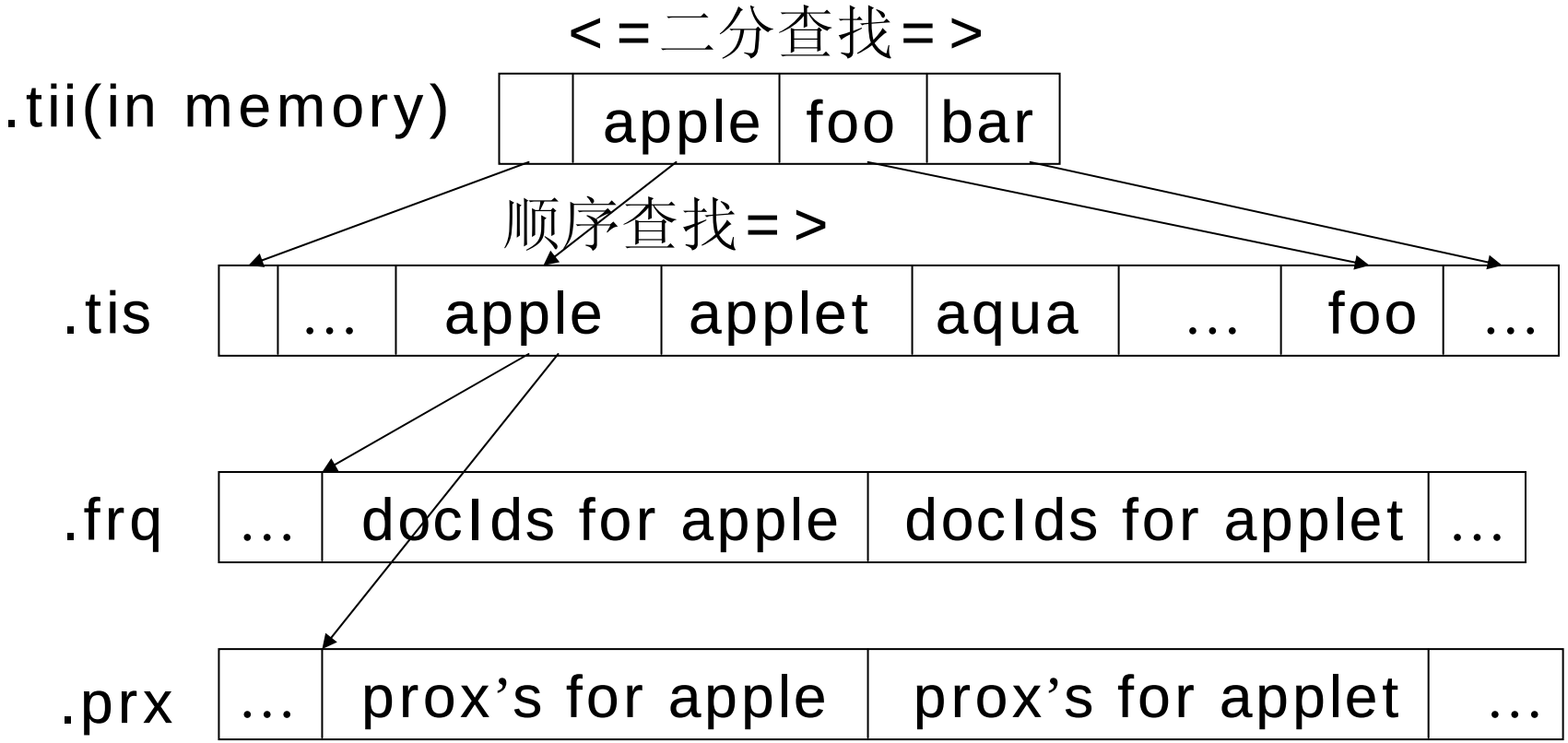
目标：快速定位索引单位所在的位置



倒排表例子


Index terms
df
computer
3
database
2
science
4
system
1
D_j, tf_j
a posting
D_7, 4
a postings list
D_1, 3
D_2, 4
D_5, 2
Optional and may be stored in a separate file

索引结构


<=二分查找=>
.tii(in memory)
apple
foo
bar
顺序查找=>
.tis
...
apple
applet
aqua
...
foo
...
.frq
...
docIds for apple
docIds for applet
...
.prx
...
prox's for apple
prox's for applet
...

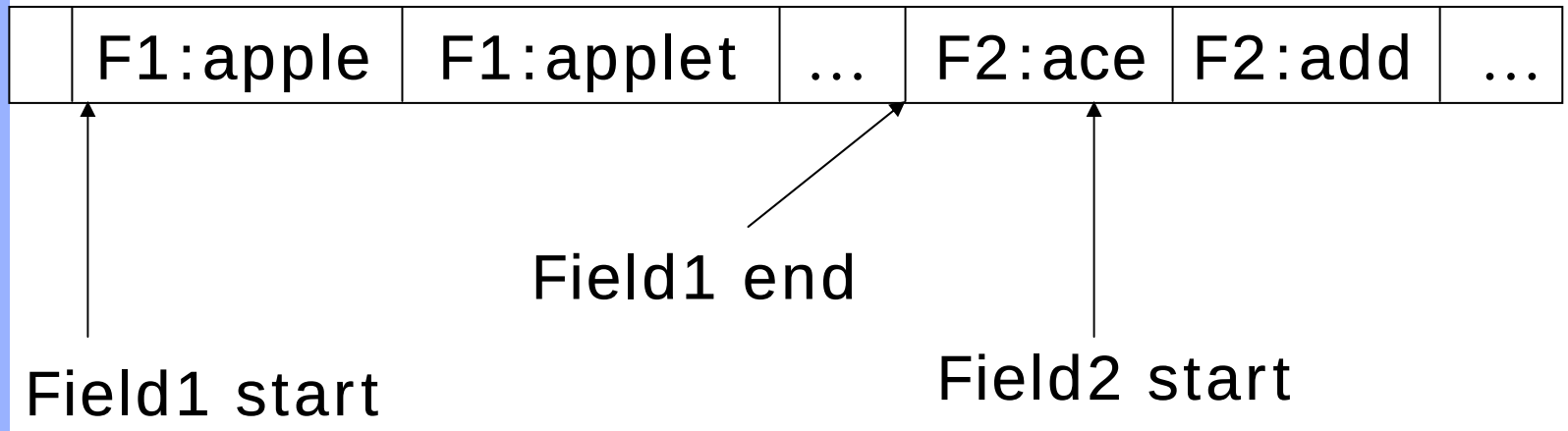
Lucene的索引文件(1)

- .tis: term infos;

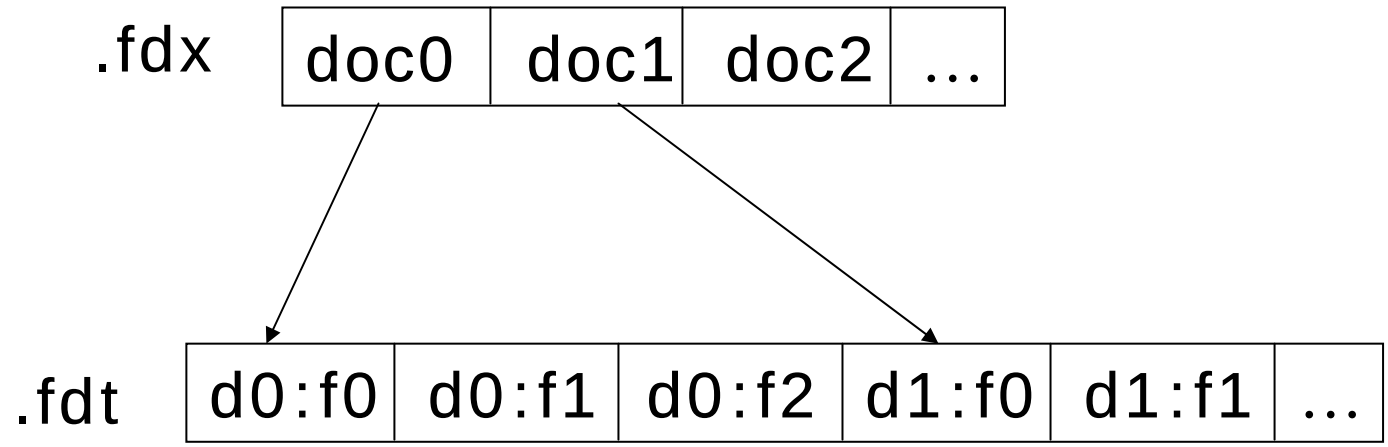
  .tii: term info index;
  - 词典(Term dictionary) 记录了文档数据中用于建索引的词,以及和该词相关的词频信息和位置信息的指针
- .frq: each term with frequency;
  - 词频数据(Term Frequency data) 对于每个词典中的词, 记录出现该词的文档编号,以及每个文档中该词的词频;
- .prx: each term position with document;
  - 词位置数据(Term Proximity data) 对于每个词典中的词, 记录该词在各个文档中出现的位置;

对多域的支持

.tis


F1:apple
F1:applet
...
F2:ace
F2:add
...
Field1 start
Field1 end
Field2 start

存储域(stored field)



Lucene的索引文件(2)

- .fnm: filed name; .fdx: field index;
- .fdt: field data; .f(field num.): field norm;
- .nrm: norm by fieldlength (=.f(num) v2.1) ;
- .del: a segment contain deletions
- segments; commit.lock; index.lock; deleteable

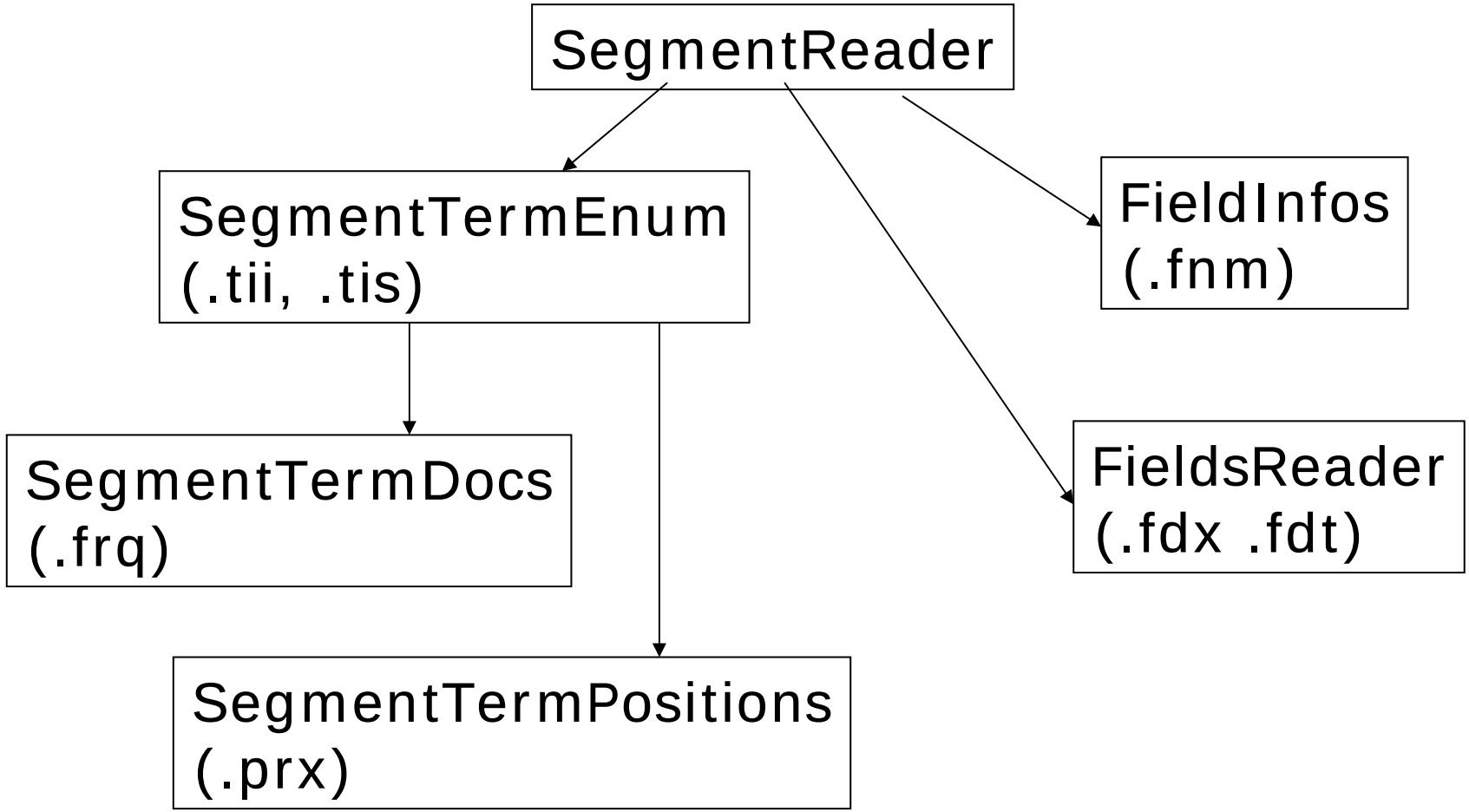
tis(tii) -> frq (prx) -> document ID

(nrm, del) -> fdx -> fdt

Segment

- 一个segment就是一个完整的索引
- 组成：
  - 索引：.tii, .tis, .frq, .prx
  - 存储：.fdx, .fdt
  - 辅助文件：
    - .fnm：字段名⇔字段编号 (fieldInfos)
    - .f0, .f1 … ：normalization vector (norms)
- Segment的特点
  - 检索算法简单，效率高
  - 难以更新

Segment classes

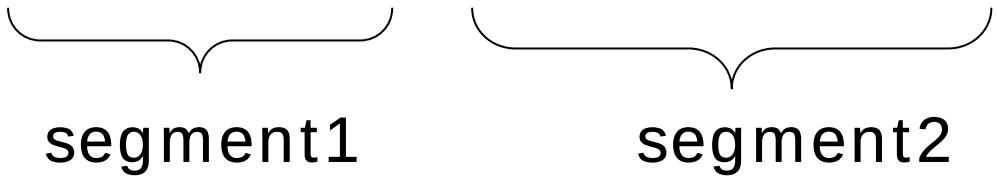


Segments

- 索引可以由多个segment(片断)组成
- 每个片断都可以独立进行检索
- 新增文档写入新片断，原有片断不变
- 所有片断检索结果的并集是最终结果
- 片断太多会影响效率：
  - 每个片断都有.tis文件
  - 在N个片断中检索，至少访问磁盘N次

docId映射

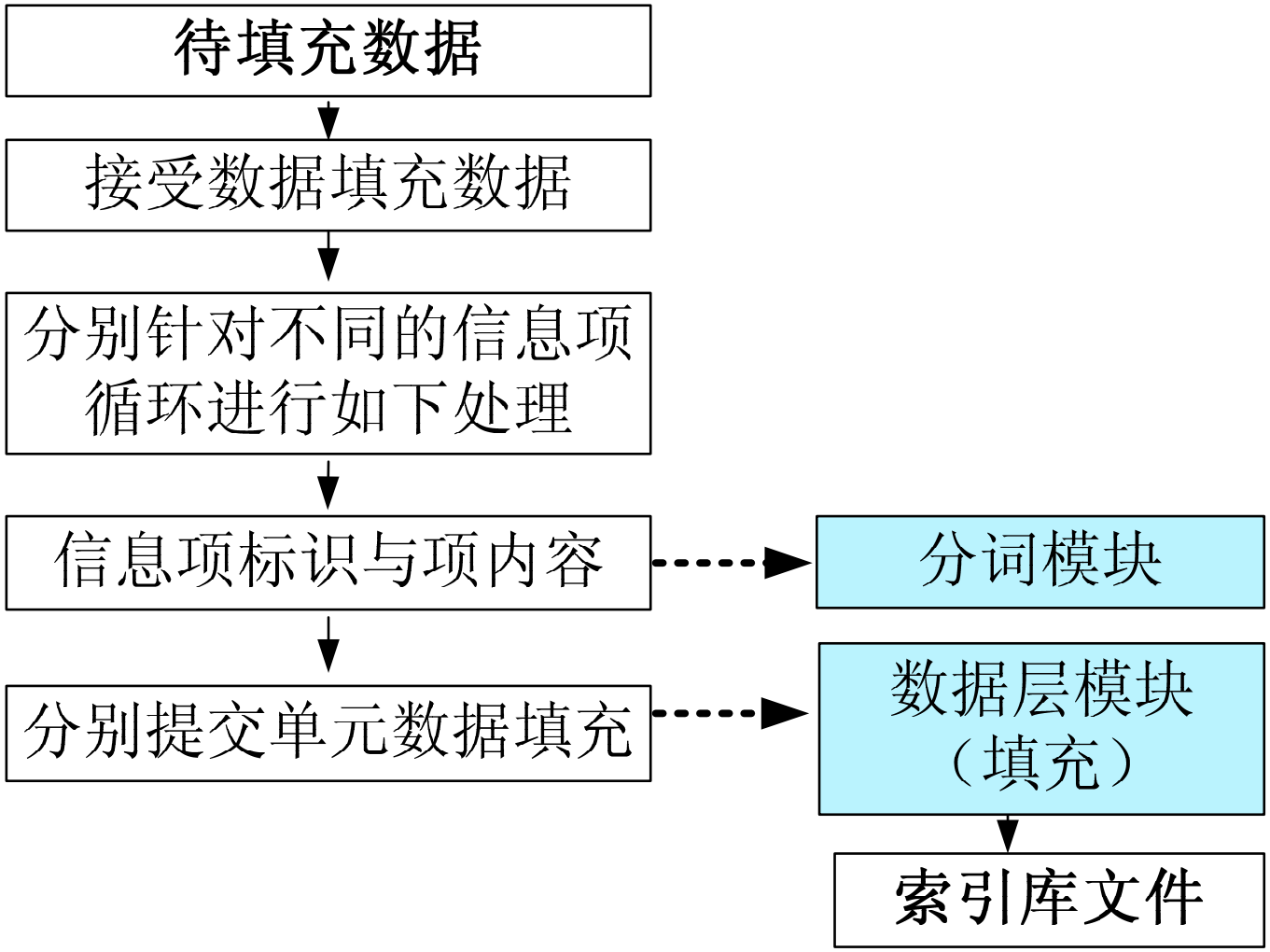
- 虽然一个索引库包含多个segment，仍然要给用户“一个segment”的印象。
- Segment1: docId [0, 1, … 99]
- Segment2: docId [0, 1, … 49]
- Segments: Segment1 + Segment2
  - docId [0, 1, …, 99, 100, 101, …, 149]



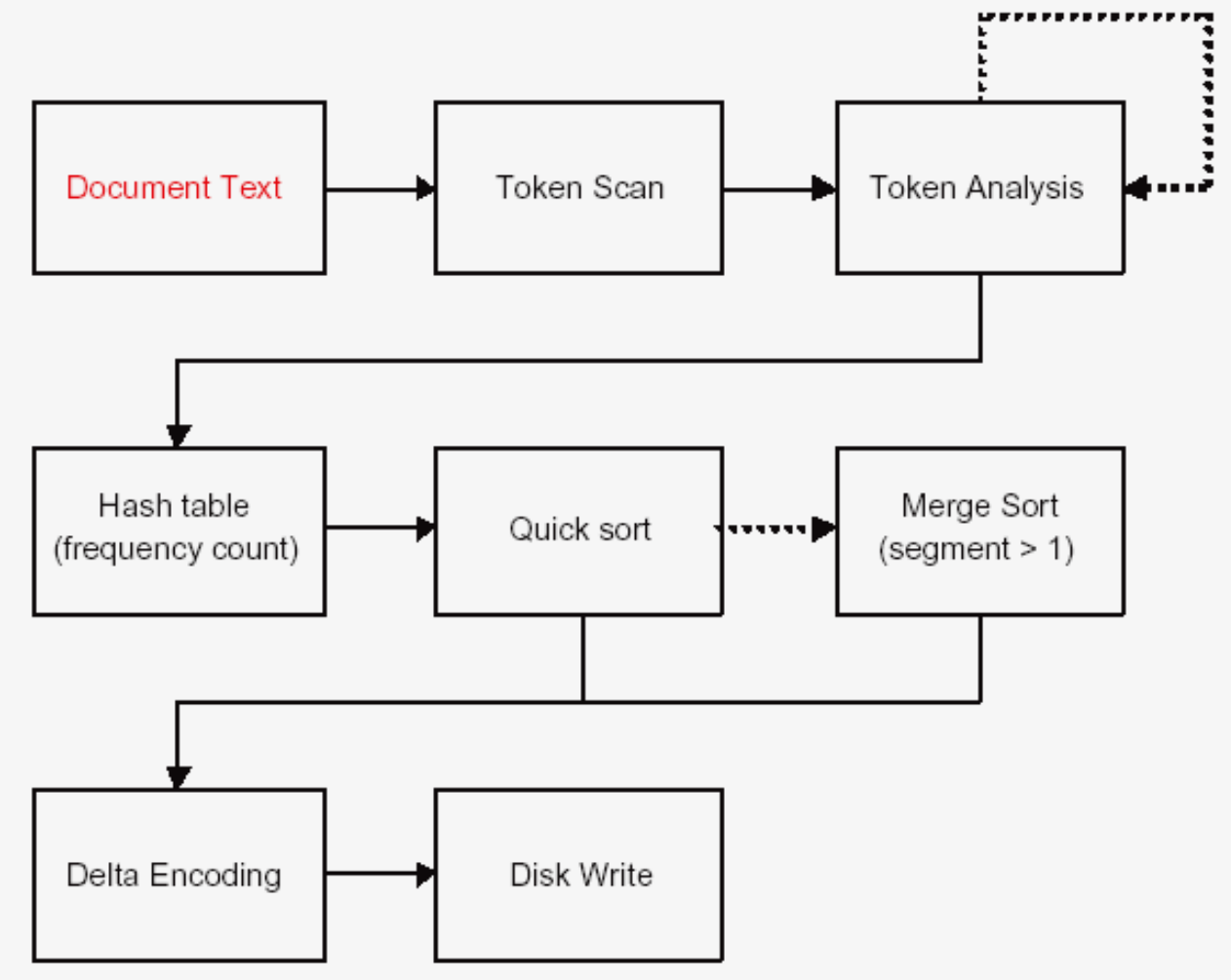
Different Field

切词	索引	存储	用途
Yes	Yes	Yes	切分词索引并存储，比如：标题，内容字段
Yes	Yes	No	切分词索引不存储，不用于返回显示，但需要检索
No	Yes	Yes	不切分索引并存储，比如：日期字段
No	No	Yes	不索引，只存储，比如：文件路径
Yes	Yes	No	只全文索引，不存储

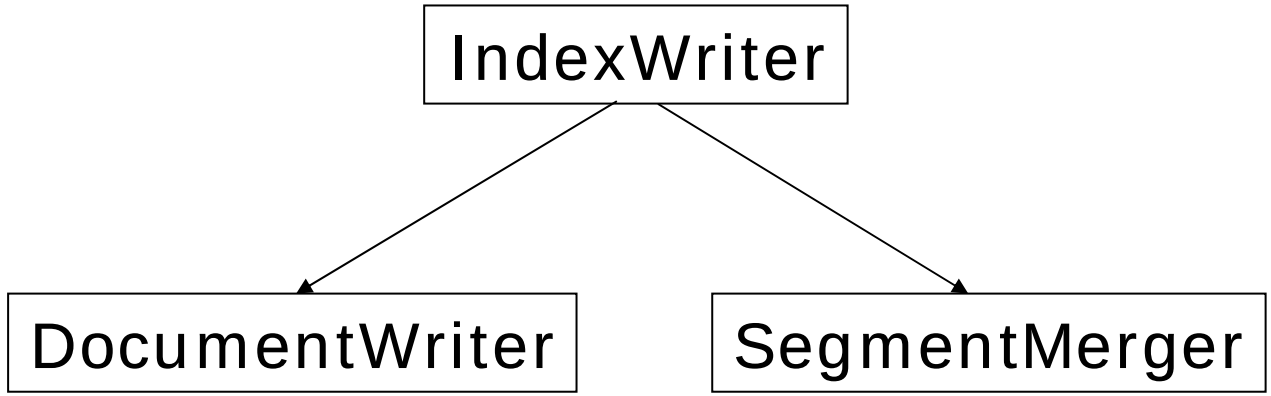
文档索引


待填充数据
接受数据填充数据
分别针对不同的信息项
循环进行如下处理
信息项标识与项内容
分词模块
分别提交单元数据填充
数据层模块
（填充）
索引库文件

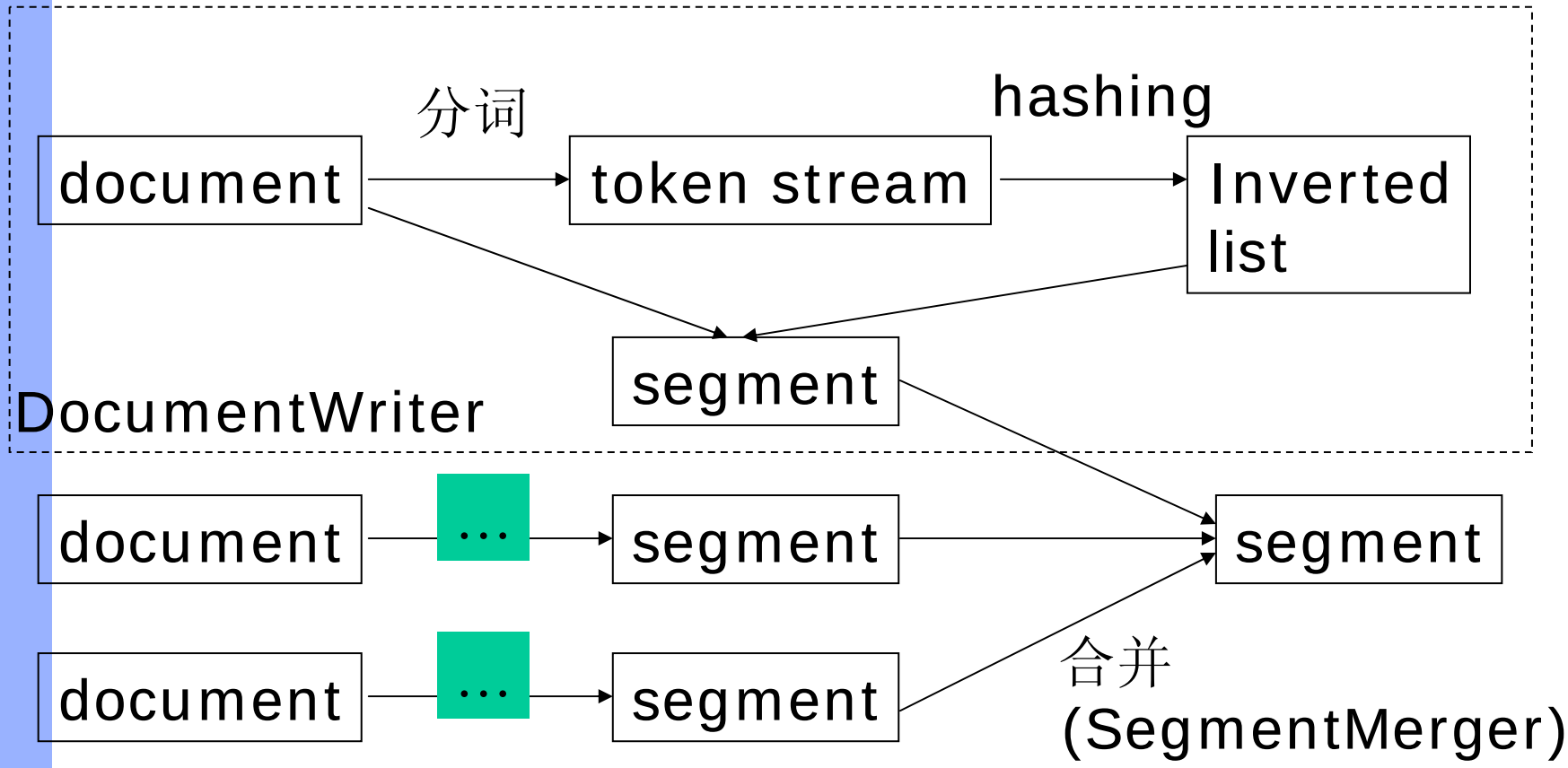
建索引过程


Document Text
Token Scan
Token Analysis
Hash table
(frequency count)
Quick sort
Merge Sort
(segment > 1)
Delta Encoding
Disk Write

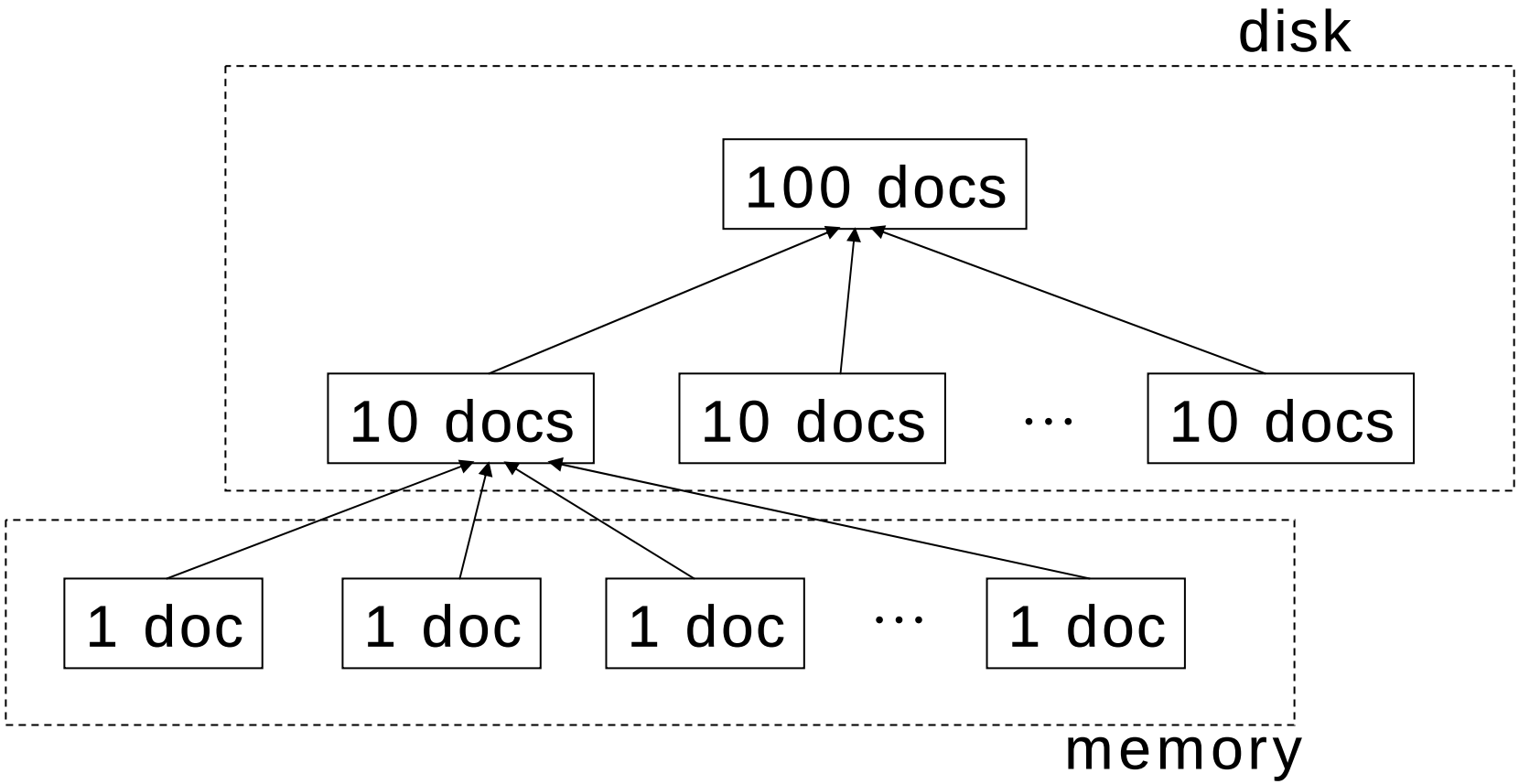
IndexWriter classes



建索引过程


分词
hashing
document
token stream
Inverted list
segment
DocumentWriter
document
…
segment
segment
document
…
segment
合并
(SegmentMerger)

合并segment


disk
100 docs
10 docs
10 docs
…
10 docs
1 doc
1 doc
1 doc
…
1 doc
memory

合并segment

Segment1 (2 docs)

- Apple
  - Doc0: prox1, prox2…
  - Doc1: prox1, prox2…
- Banana
  - Doc1: prox1, prox2…

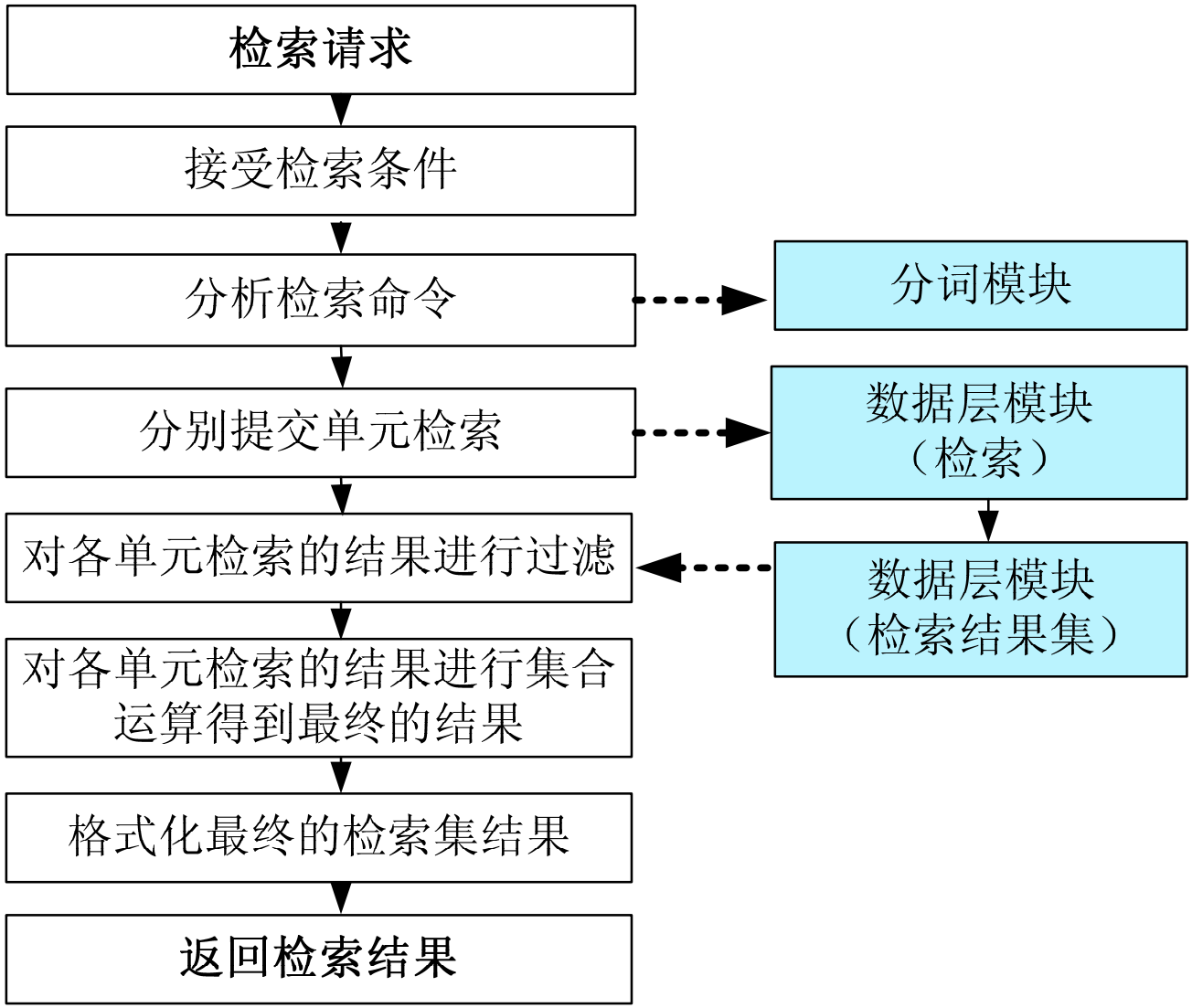
Segment2 (1 doc)

- Apple
  - Doc0: prox1, prox2...
- Candy
  - Doc0: prox1, prox2...

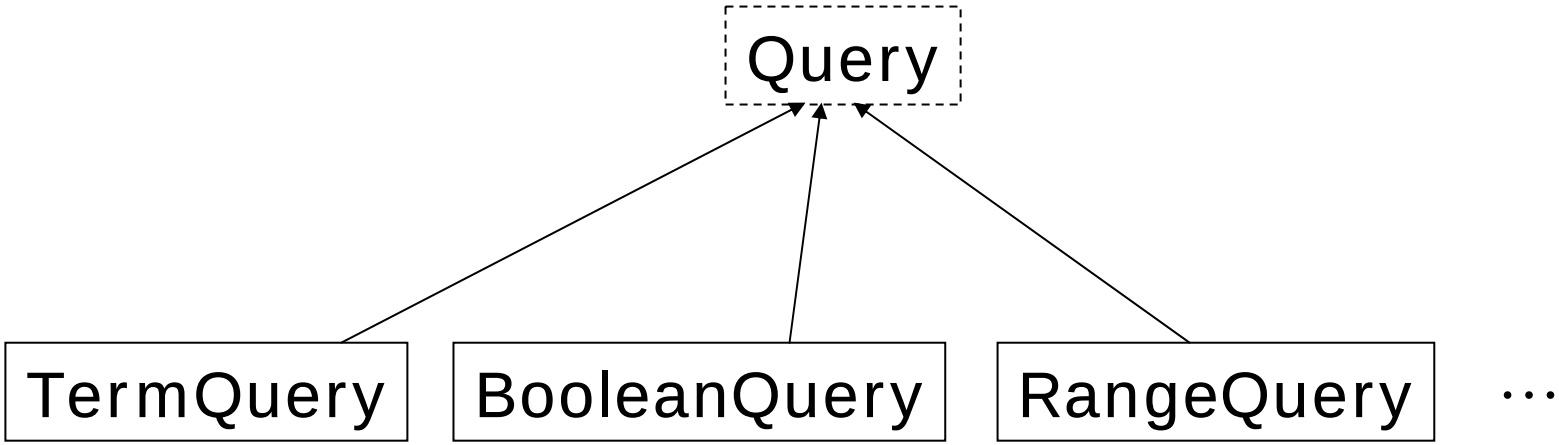
Segment1+Segment2

- Apple: D0: p1, p2... D1: p1, p2... D2: ...
- Banana: ...
- Candy: ...

检　索


检索请求
接受检索条件
分析检索命令
分词模块
分别提交单元检索
数据层模块
（检索）
对各单元检索的结果进行过滤
数据层模块
（检索结果集）
对各单元检索的结果进行集合运算得到最终的结果
格式化最终的检索集结果
返回检索结果

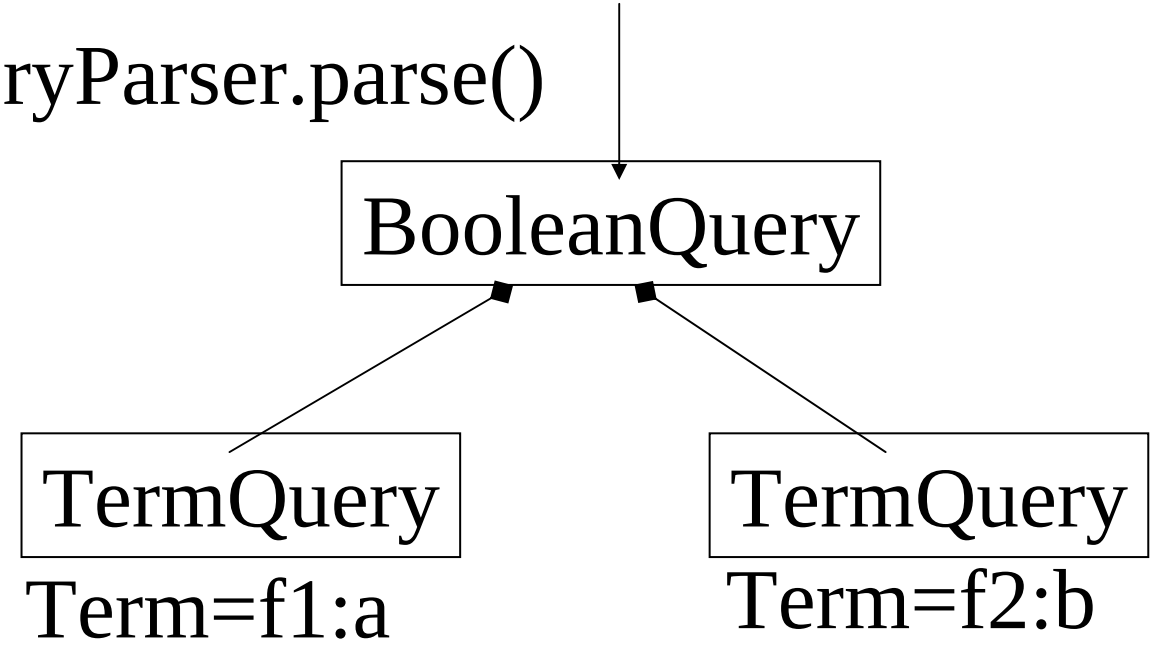
Query classes



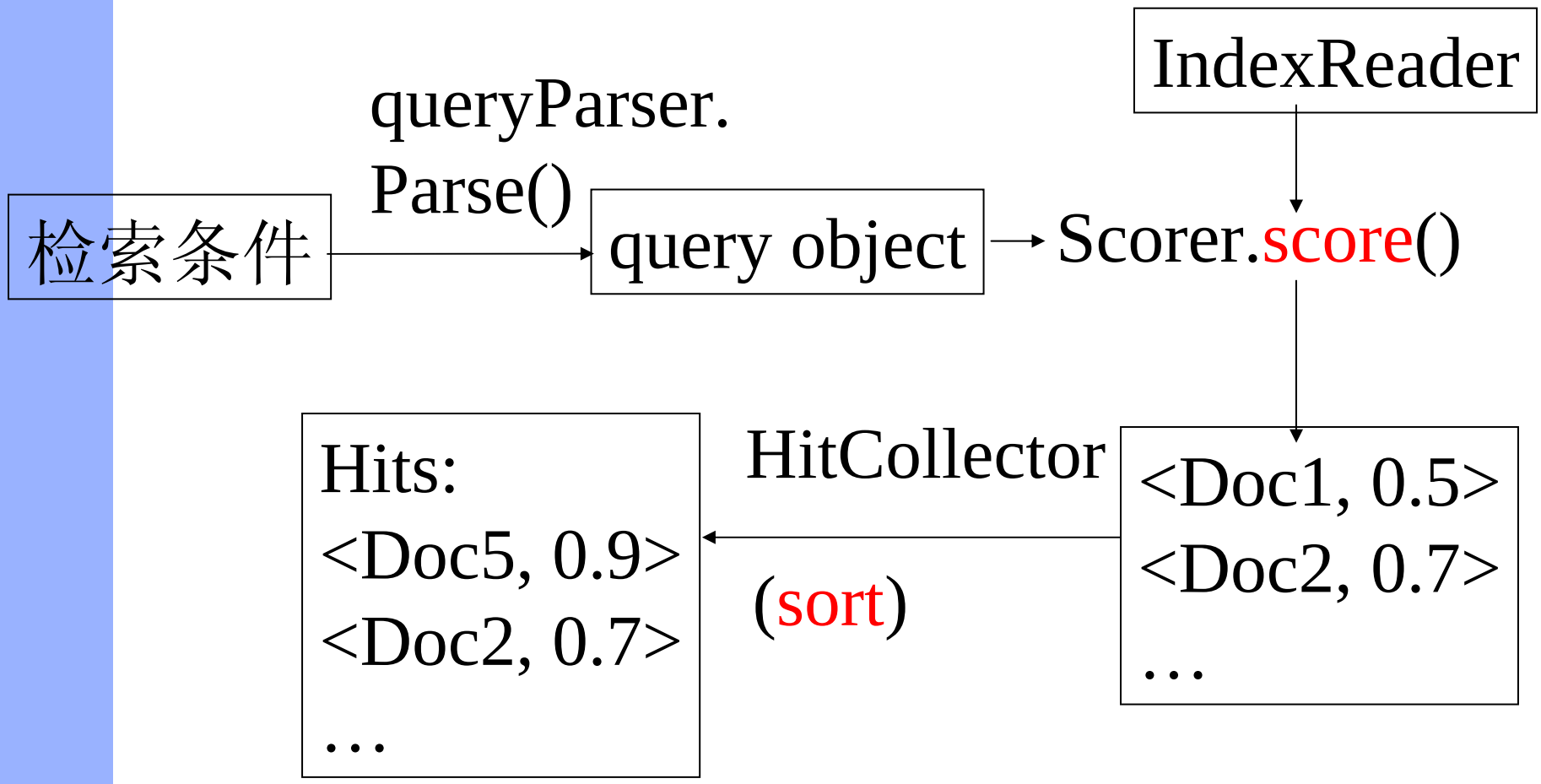
Query object

Select * from idx1 where f1:a and f2:b

queryParser.parse()



检索过程


IndexReader
queryParser.
Parse()
检索条件
query object
Scorer.score()
Hits:
<Doc5, 0.9>
<Doc2, 0.7>
…
HitCollector
(sort)
<Doc1, 0.5>
<Doc2, 0.7>
…

TermQuery

- TermQuery为Lucene支持的最简单的查询方式 Query为一个关键词Term
- TermQuery的计算公式
  - score = sqrt(freq) * idf * norm * boost
  - idf = ln(maxDoc/(docFreq + 1))+ 1.0
  - norm = fieldboost / sqrt(fieldlength)
- 说明
  - 其中的idf和boost值与文档无关，不影响排名
  - 排名因子(对于单个TermQuery)

    sqrt(freq)*fieldboost/sqrt(fieldlength)
  - fieldboost人为赋予的经验值 默认值都为1.0
    - 注意 norm在实现时只用了1个字节表示 故误差较大

BooleanQuery

- BooleanQuery是一种复合式的Query 支持多种不同Query的逻辑组合
- BooleanQuery例子
  - ＋俄罗斯　恐怖　事件　－美国
  - ＋（俄罗斯 美国）恐怖 事件
- 可以对不同的query赋予不同的boost值表示该query在整个BooleanQuery中的重要程度
  - 例如：俄罗斯3.0　恐怖2.0　事件1.0

BooleanQuery计算公式

- 计算公式
  - $score_j =$

    $coord_j * \sum_i (boost_i * idf_i * tf_{i,j} * idf_i * fieldnorm) / sqrt(\sum_i (idf_i * idf_i * boost_i * boost_i))$
  - $fieldnorm = fieldboost / sqrt(fieldlength)$
- 其中$sqrt(\sum_i (idf_i * idf_i * boost_i * boost_i))$和文档无关，不会影响文档的排名

Lucene检索模型

- 标准向量空间模型

$$Sim(d_j, q) = \frac{d_j \bullet q}{|d_j| \times |q|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

- 一种简化的空间向量模型
  - 文档中词项的权重
    - $w_{i,j} = tf_{i,j} * idf_i$
  - 查询中词项的权重
    - $w_{i,q} = boost_q * idf_q$
  - 文档向量模 $|d_j| \approx sqrt(fieldlength)$

编码

- 编码的目的：减小索引占用的磁盘空间
  - 磁盘访问是检索的主要开销
- 两个矛盾的目标
  - 高压缩比
  - 低时间开销

Delta编码

- 存储相邻数据的差，缩小数据动态范围
  - a, b, c… => a, b-a, c-b…
- Case 1: docId
  - 包含某个词的docId在.frq中是一个递增序列

    0, 2, 3, 5, 6, 10…
  - 对于高频词，相邻的docId数字接近
  - Delta编码：

    0, 2, 1, 2, 1, 4…

Delta编码

- Case 2: term
  - 词在.tis中按字典序存放

    apple, applet, application, banana …
  - 相邻词有重复部分
  - 保存差异部分的字符串+相同部分的长度
  - Delta编码:

    〈0, apple〉, 〈5, t〉, 〈4, ication〉,

    〈0, banana〉

VInt

- 存储Int类型需要4bytes
- 大部分int值很小
  - 词频：低频词远多于高频词
  - 字符串长度：大部分词不超过10
  - Delta编码后的序列范围会缩小
- 用1byte存放最常见(较小)的值，更多的byte存放不常见(较大)的值。(变长编码)

VInt

- 最高位为1表示后面还有值，为0表示到此结束
- 0~127: 1byte, 0xxxxxxx
- 128~2^14-1: 2bytes

  xxxxxxxyyyyyyy =>

  1xxxxxxx, 0yyyyyyy
- 2^14~2^21-1: 3bytes
- 2^21~2^28-1: 4bytes
- 2^28~2^32-1: 5bytes

基本数据类型与存储

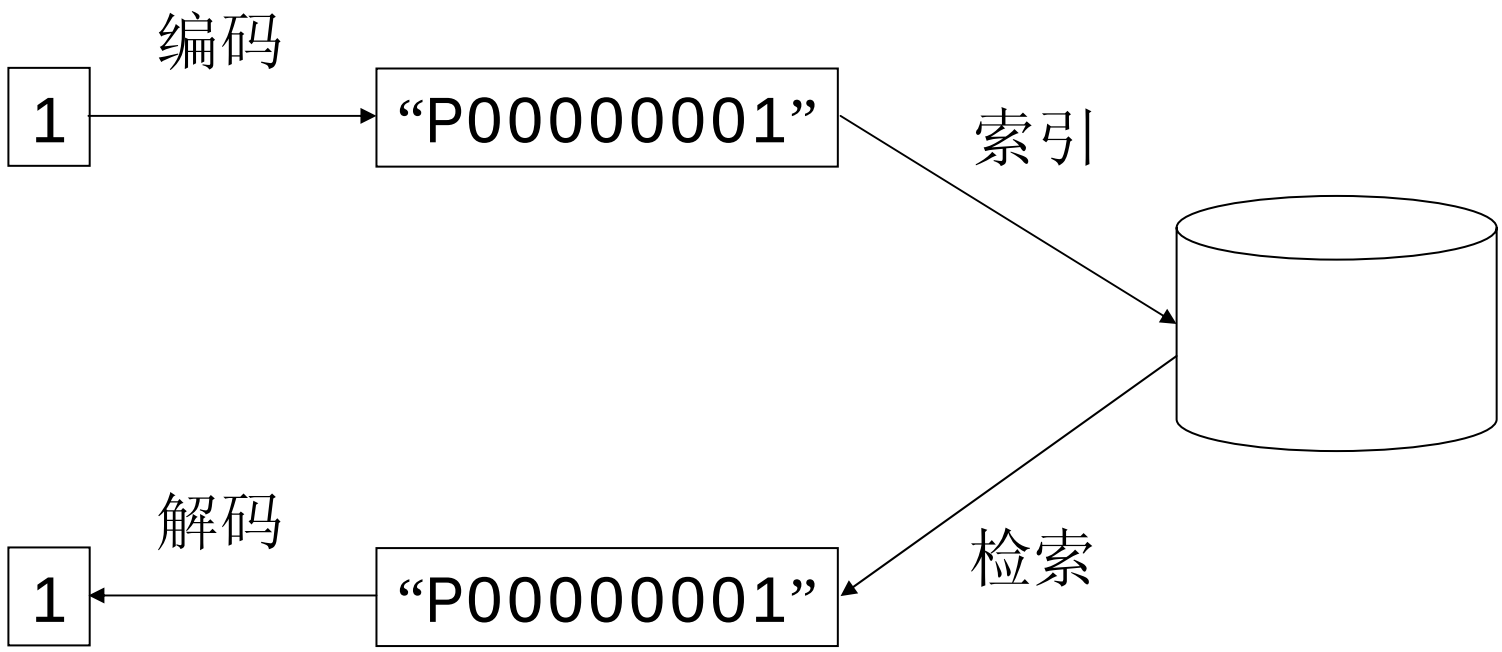
Byte: an eight-bit byte;	**VInt:** A variable-length format
UInt32: UInt32 --> <Byte>^{4}	**Chars :** UTF-8 encoding
Uint64: UInt32 --> <Byte>^{8}	**String:** String --> VInt, Chars

1: 00000001	127: 01111111
128: 10000000 00000001	16,383: 11111111 01111111
16,384: 10000000 10000000 00000001	

对数值数字类型的支持

- 索引结构/检索算法都是为文本设计的
- 为什么需要索引数值类型？
  - 脱离数据库运行
  - 尽量减少与数据库的通讯数据量
- 怎样索引？
  - 把数值编码成文本

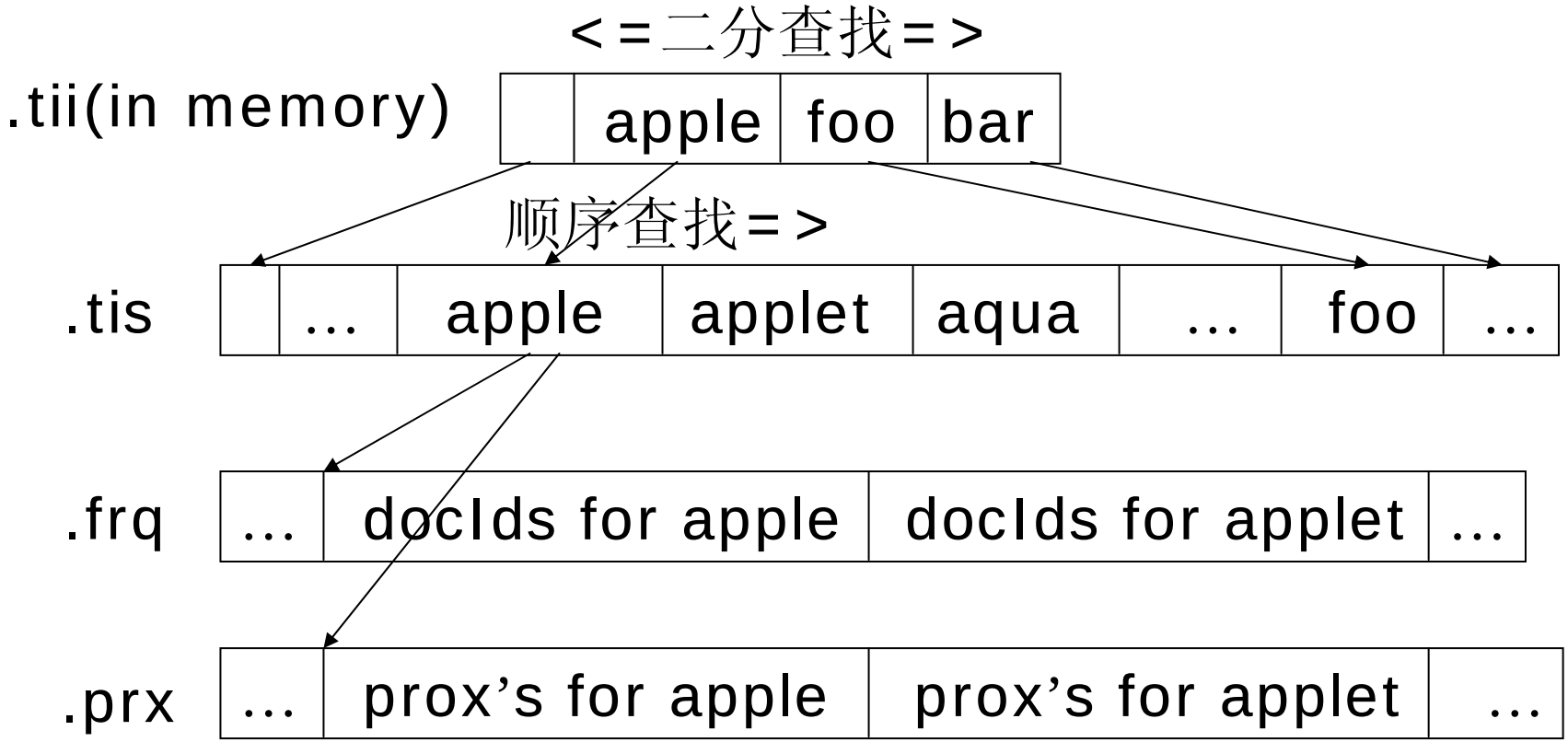
对数值数字类型的支持



对数值数字类型的支持

- 要求：保留数值的顺序
  - 索引的顺序是字典序
- 无效的编码
  - 1=>“1”
  - 11=>“11”
  - 2=>“2”
- 正确的编码
  - 1=>“00000001”
  - 2=>“00000002”
  - 11=>“00000011”

小结：索引结构


<=二分查找=>
.tii(in memory)
apple
foo
bar
顺序查找=>
.tis
...
apple
applet
aqua
...
foo
...
.frq
...
docIds for apple
docIds for applet
...
.prx
...
prox's for apple
prox's for applet
...

问题与挑战

智能信息检索

- 仍以关键字匹配查询为主。
- 利用了Web文档超文本信息。
- 部分特定知识领域的智能搜索引擎使用了机器学习和人工智能算法实现数据抽取。
- 基于自然语言理解的搜索引擎还处于低级的萌芽状态。

常见的智能检索技术（1）

- 短语搜索
  - 例如：检索“天津机场”可以检索到“天津国际机场”“天津机场”
- 同义词搜索
  - 例如：检索“程序”可以检索到“程式”
- 反义词搜索
- 简称词搜索
  - 例如：检索“人大”可以检索到“人民代表大会”“人民大学”
- 串形相似搜索
  - 例如：检索“海洋出版社”可以检索到“海燕出版社”、“大洋出版社”
- 通配符搜索
- 减词搜索
  - 例如：检索“电子防护材料”可以检索到“电子材料”、“电子保护材料”、“电子防护材料”

常见的智能检索技术（2）

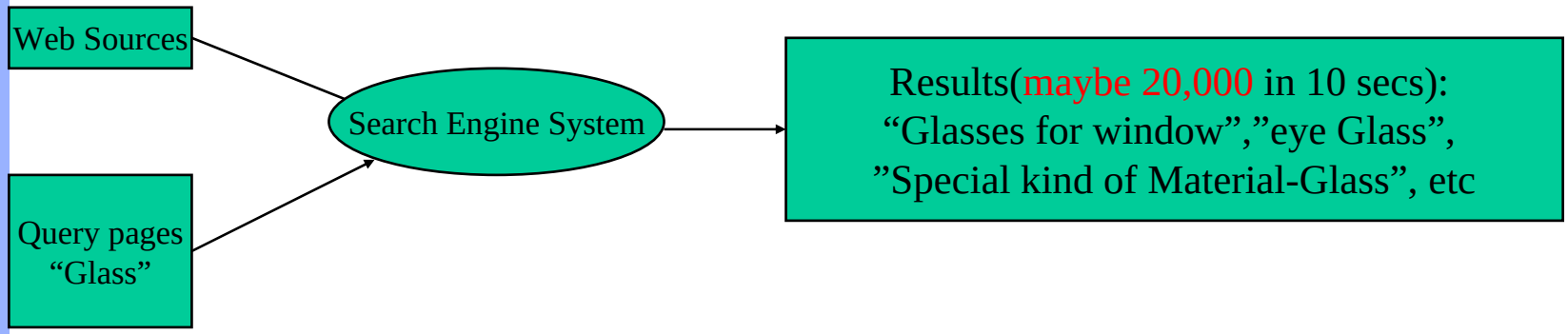
- 异形词搜索
  - 例如：检索“人材”可以检索到“人才”，检索“百叶窗”可以检索到“百页窗”
- 同音字搜索
  - 例如：检索“清和县”可以检索到“清河县”和“青河县”
- 拼音搜索
  - 例如：检索“diannao”可以检索到“电脑”
- 拼音字头搜索
  - 例如：检索“yhy”可以检索到“宇航员”
  - 检索“chp”可以检索到“纯朴”（ch p）、“彩绘铺”（c h p）
- 赘字移除搜索
  - 例如：检索“天津的汽车”可以检索到“天津汽车”
- 多语种词搜索
  - 例如：检索“windows”可以检索到“视窗”，检索“微软”可以检索到“Microsoft”

基于Semantic Web的信息检索

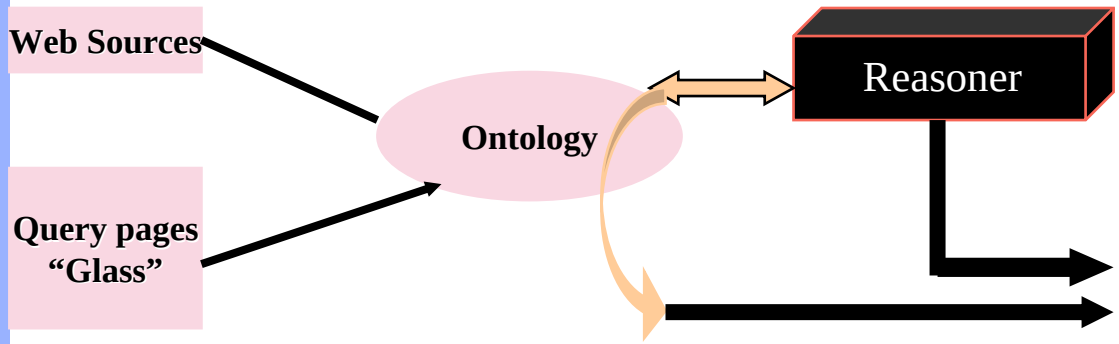
- XML和Ontology在语义Web中的使用，对Web搜索技术带来了新的契机。
- 搜索引擎的可以进行语义级别的Web分析和信息抽取。
- 问题式的检索由于使用了基于Ontology的定义，可以采用级联的方式在若干页面查找后最后给出问题的答案。
- 国内外的研究工作刚刚起步...

基于语义的检索

- Traditional Web Search application



- The Semantic Web Search Application



You get exact Results(2,3secs):
(≤10 according to user requirements)
"EyeGlasses",
"EyeGlass shop near you",
"The cost you can afford",
"The current popular style"…

全文检索的挑战

- 语言方面的挑战
  - 检索意图表达不够清楚
  - 返回的是文档列表，缺乏对答案的准确定位
  - 信息太多太杂，检索效果不佳
- 信息源方面的挑战
  - 信息来源不同，
  - 存储格式和系统不同，
  - 异构资源整合检索，用户统一访问和检索。
- 性能方面的挑战
  - 面对海量的内容数据
  - 并发检索压力，保证检索性能
- 多种检索技术的联合检索
  - 与关系数据库中数据的联合检索性能
  - XML数据的检索

技术热点

- 与自然语言处理知识的有机结合
- 用内在概念来表示文档和查询
- 建立有效的用户模型
- 建立低维度的Aspect models
  - Matrix representations typically very sparse
  - Reduce dimensionality to small # key aspects
    - Mapping contextually similar terms together
    - Latent semantic analysis

技术融合

- 信息的搜集和校对
- 内容分析
  - 切分、标注、结构化、语义分析
- 内容的索引和检索
- 内容的分类、过滤和推送
- 内容的聚类和去重
- 问答系统、文摘、信息抽取、知识挖掘
- 跨语言、跨媒体(Cross-media)的内容管理

Open Research Problems in IR

- 交互式的IR
- 个性化的IR
  - “星球大战”对政治家和儿童是不一样的
- 跨语言(Translingual IR)，
  - 用中文查询英文信息
- 跨媒体(Transmedia IR)
  - 用文字查询图像 Google→图像→“许智宏”
- 采用多种信息表示方式和比较方式
- 如何利用上下文信息和环境信息

高维索引

相似搜索概念与应用

- 从待搜索对象集合中找出与指定的查询对象相似度最大的对象子集
- 针对数据量（海量）解决效率问题
- **区别于精确查询：综合特征（相似度）**

应用

- 在海量文档库上实现快速搜索相似文档
- 对海量的图片、视音频进行基于内容的快速搜索
- 实现对海量信息库的快速消重

一维索引

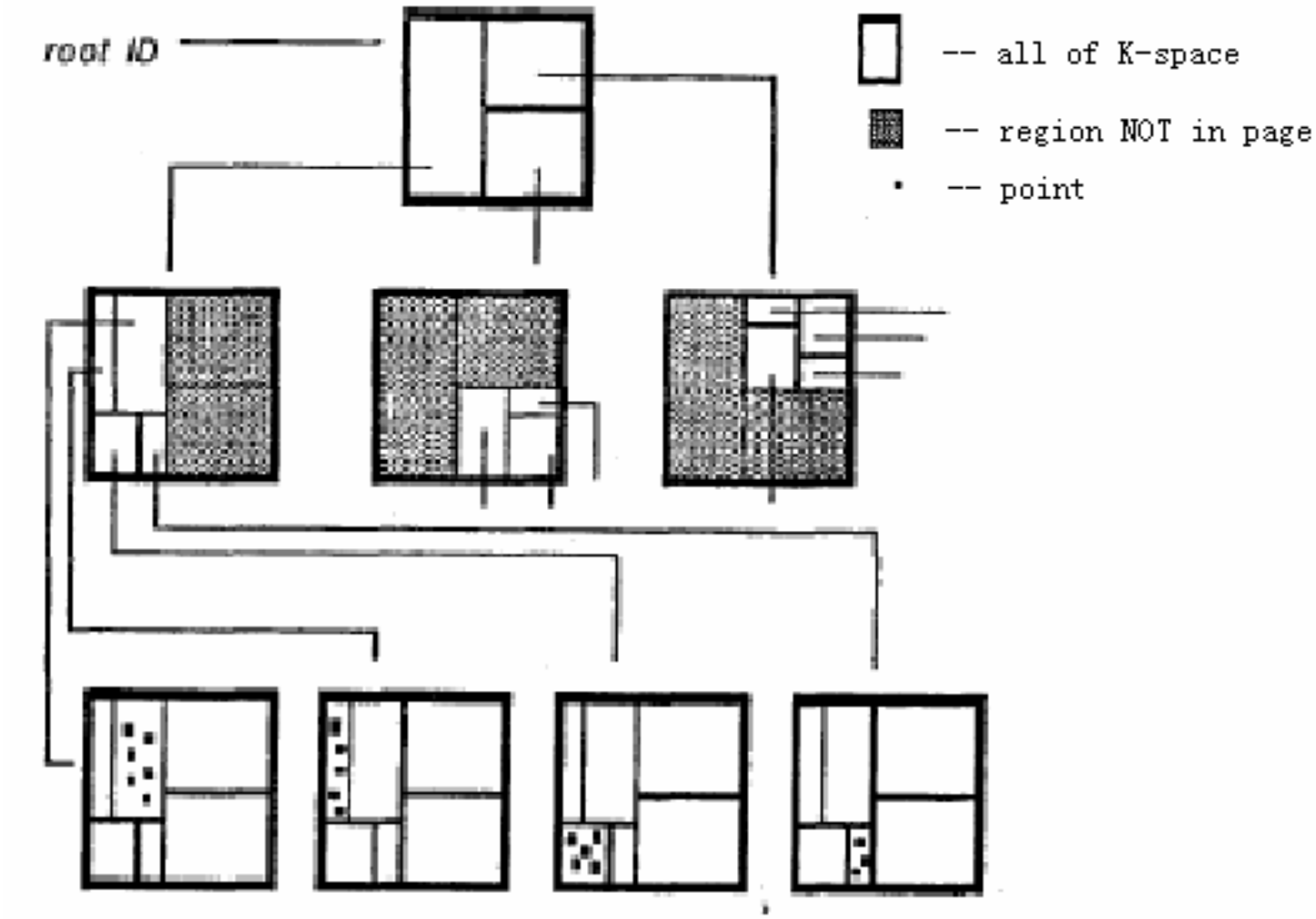
- Hash Table
  - 数值的精确匹配
  - 不能进行范围查询
- B-Trees　　ISAM
  - 键值的一维排序
  - 不能搜索多维空间

K-D-B-树

（J.T.Robinson SIGMOD’1981）

- K-D-B树是一种与B+树相似平衡树，用于多维点数据的索引结构。
- 递归的用坐标平面对搜索空间进行分割来构建树形结构。
- 每个内部节点和叶子节点都对应一个子区域，并对应一个物理存储块。
- 最大特点是在树的同一层节点对应的子空间是互相没有重叠的，从而使得任意一个点查询的路径对应单一的一条从根到叶子的路径。

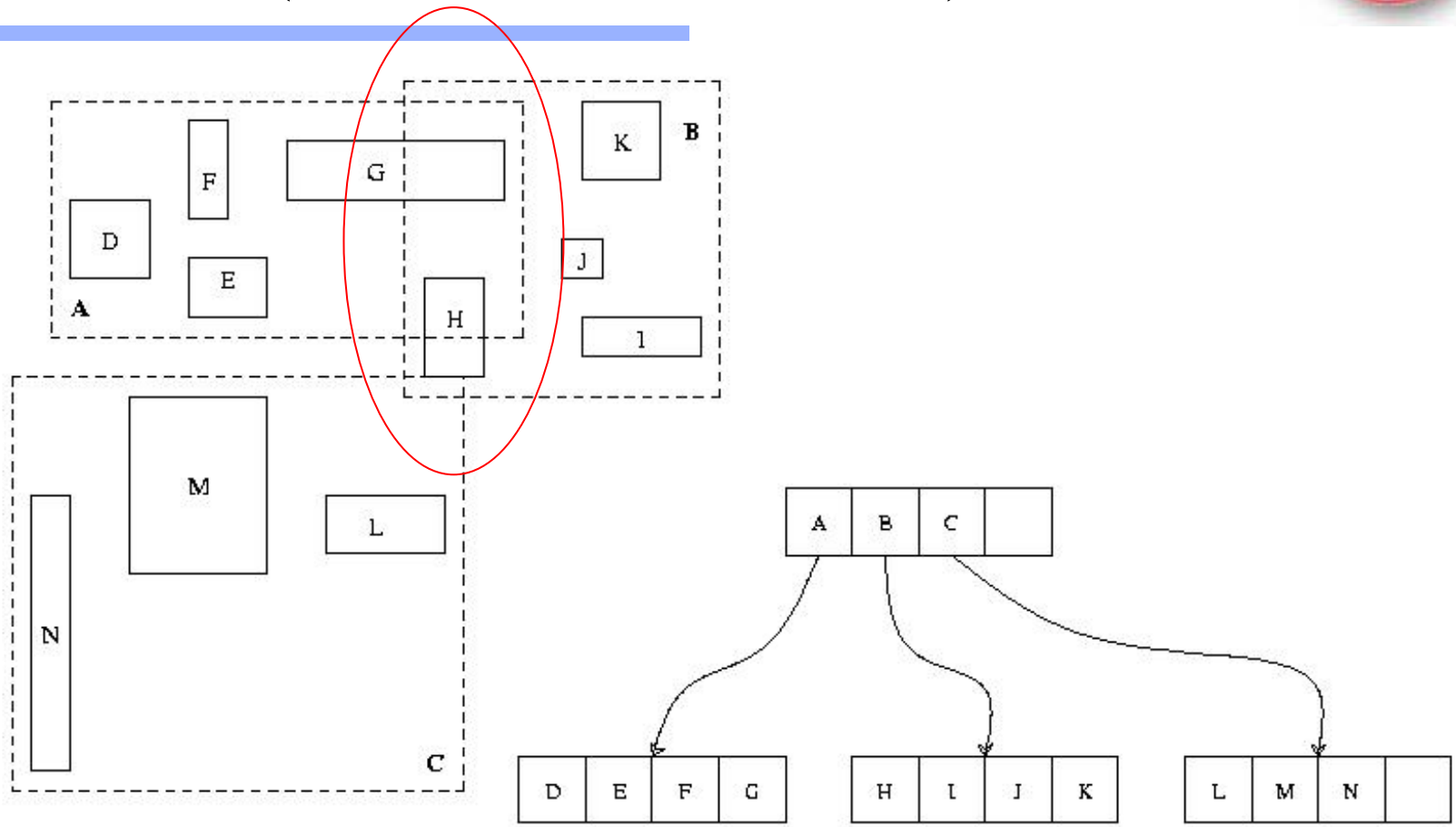
K-D-B-树



Example 2-D-B-Tree

R-tree

(A. Guttman SIGMOD'1984)



R-tree的特点

- R-tree是B-Tree对多维对象（点和区域）的扩展
- R-tree是一棵平衡树
- 一个多维对象只能被分到一个子空间中去
- 若用动态插入算法构建R-tree，在树的结点中会引起过多的空间重叠和死区（dead-space），使算法性能降低

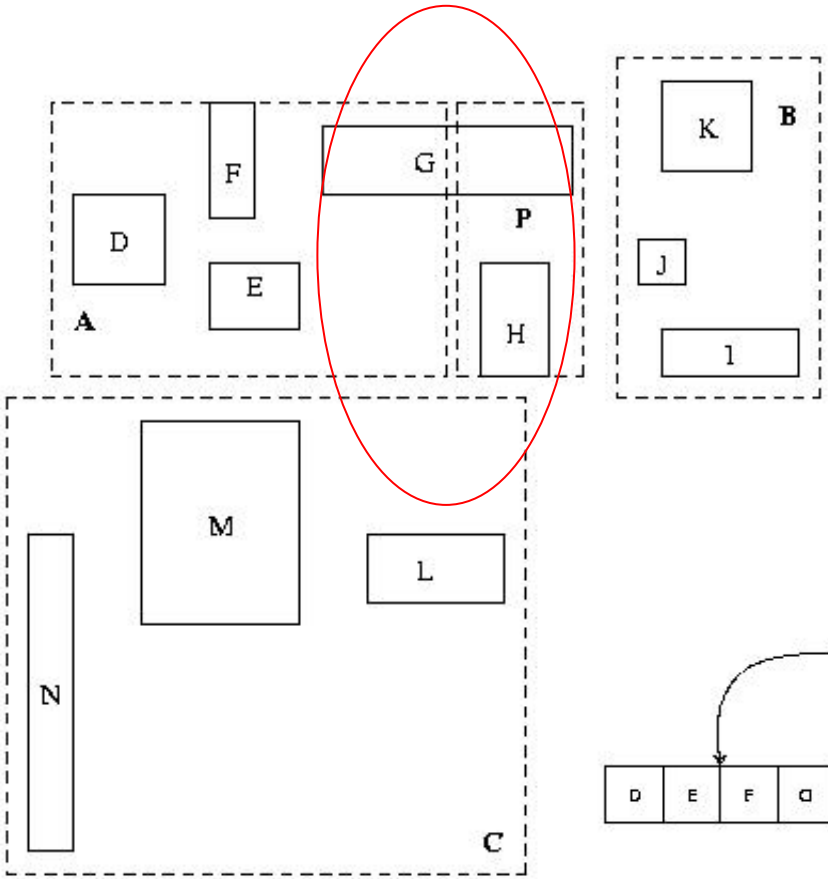
R-tree的典型算法

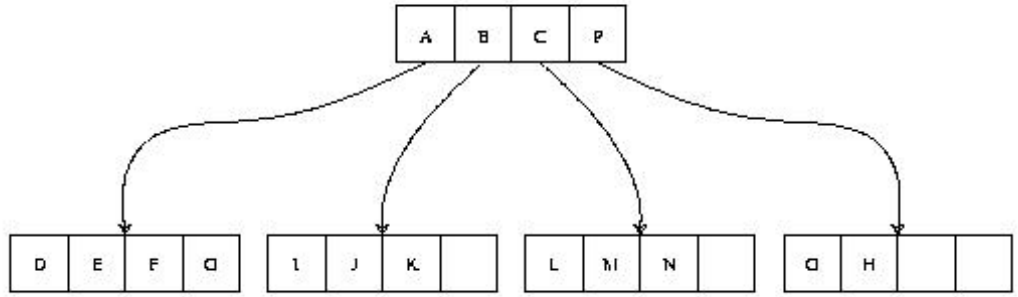
- 查找
- 插入
  - 选择叶子结点
  - 分裂结点（有多种算法）
  - 调整树
  - 必要时增加树的高度
- 删除
  - 找到包含要删除记录的叶子结点
  - 删除
  - 压缩树
  - 必要时减小树的高度
- 更新
  - 先删除老的记录索引，再插入新的记录索引

R^+-tree
(T. Sellis VLDB'1987)

- R^+-tree是K-D-B-tree对非0面积对象（不仅可以处理点，也可以处理矩形）的扩展
- 不需要覆盖整个初始空间
- 将对象标识进行复制以使结点不产生重叠
- R^+-tree比R-tree表现出更好的搜索性能（特别对点的查询），但要占据较多的存储空间
  - 避免了点查询在R-树中的多路径搜索

R^+-tree
(T. Sellis VLDB'1987)





R*-Tree
(N. Beckmann SIGMOD'1990)

- R*-Tree通过修改插入、分裂算法，并通过引入强制重插机制对R-Tree的性能进行改进。
  - R*-Tree在选择插入路径时同时考虑矩形的面积、空白区域和重叠的大小，而R-Tree只考虑面积的大小。
- R*-Tree和R-Tree允许矩形的重叠；
  - 这种重叠会使得搜索需要更多的时间，但存储效率更高。

其它R-树变种

- VAMSplit R-树通过优化分裂算法，使得分裂后所使用磁盘数据块的数目最少。
- TV-树根据多维数据中各维具有不同重要性，对维进行了减少与压缩，从而相对于R*-树提高了性能。

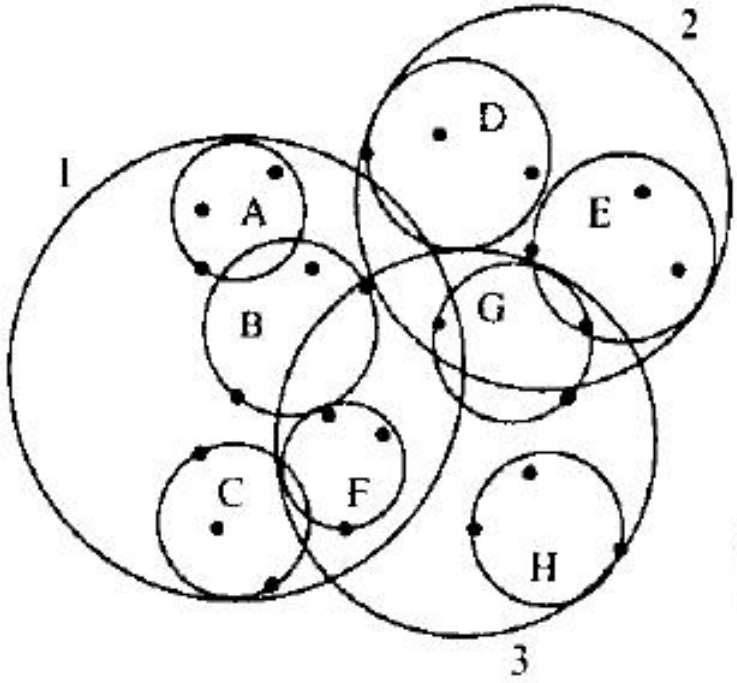
SS-Tree

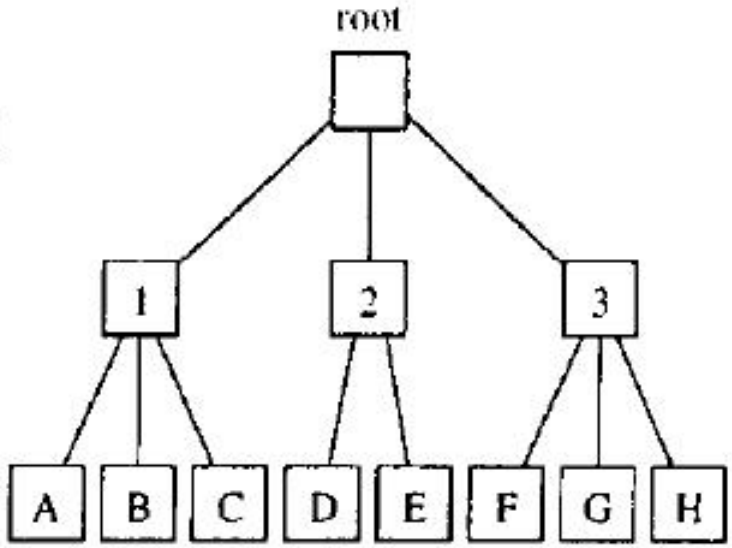
(D. A. White ICDE'1996)

- SS-Tree对R*-Tree进行了改进，通过以下措施提高了最邻近查询的性能：
  - 用最小边界园代替最小边界矩形表示区域的形状；
    - 增强了最邻近查询的性能；
    - 减少将近一半的存储空间。
  - SS-Tree改进了R*-Tree的强制重插机制。

SS-Tree

(D. A. White ICDE'1996)



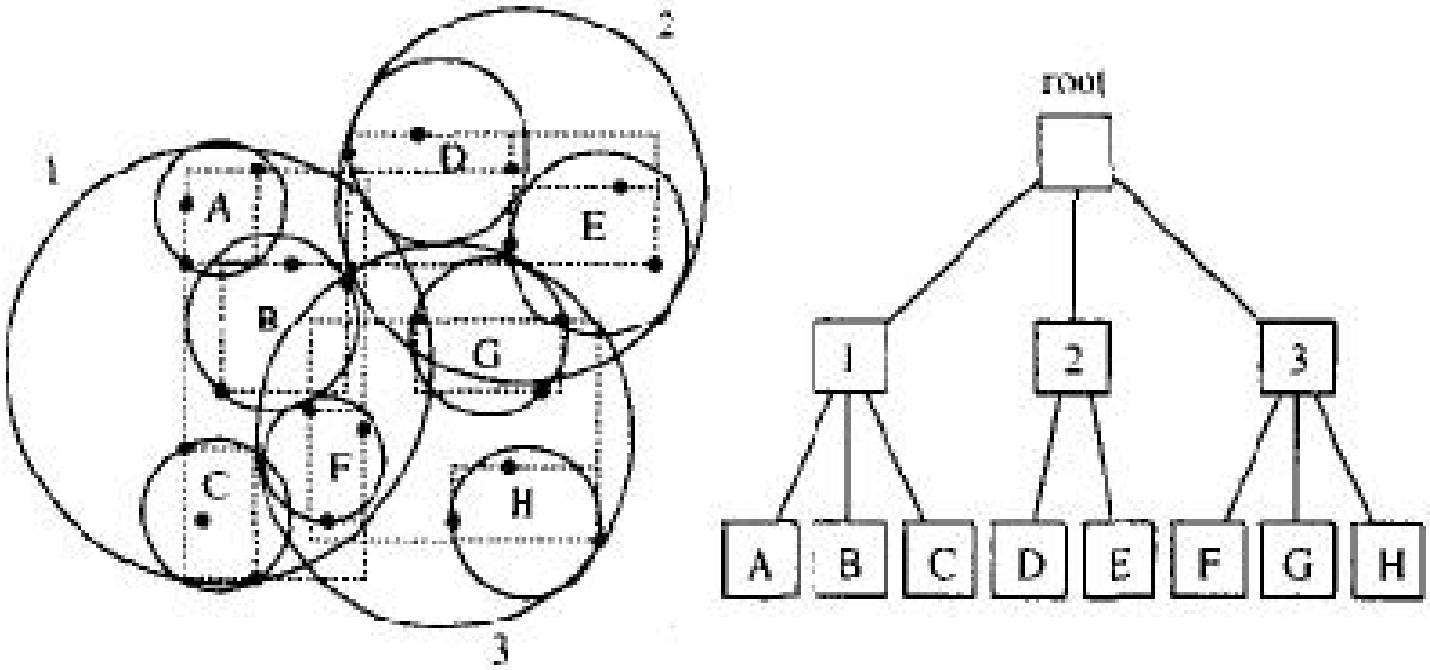


边界矩形和边界圆的比较

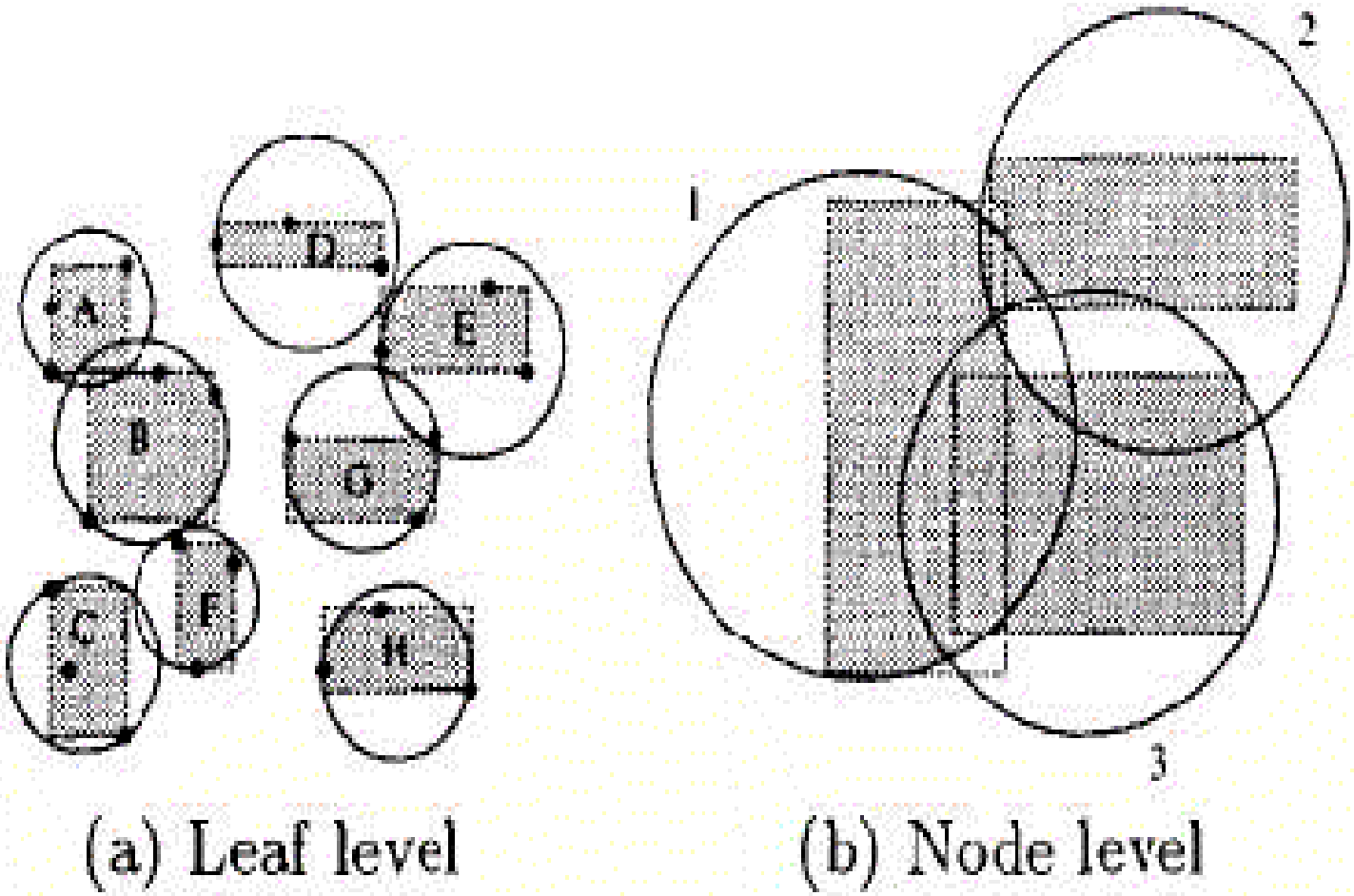
- 边界矩形的直径（对角线）比边界圆大， SS-Tree将点分到小直径区域。由于区域的直径对最邻近查询性能的影响较大，因此SS-Tree的最邻近查询性能优于R*-Tree；
- 边界矩形的平均容积比边界圆小， R*-Tree将点分到小容积区域；由于大的容积会产生较多的覆盖，因此边界矩形在容积方面要优于边界圆。

SR-Tree

(N. Katayama SIGMOD'1997)



SR-Tree的索引结构



SR-Tree的特点

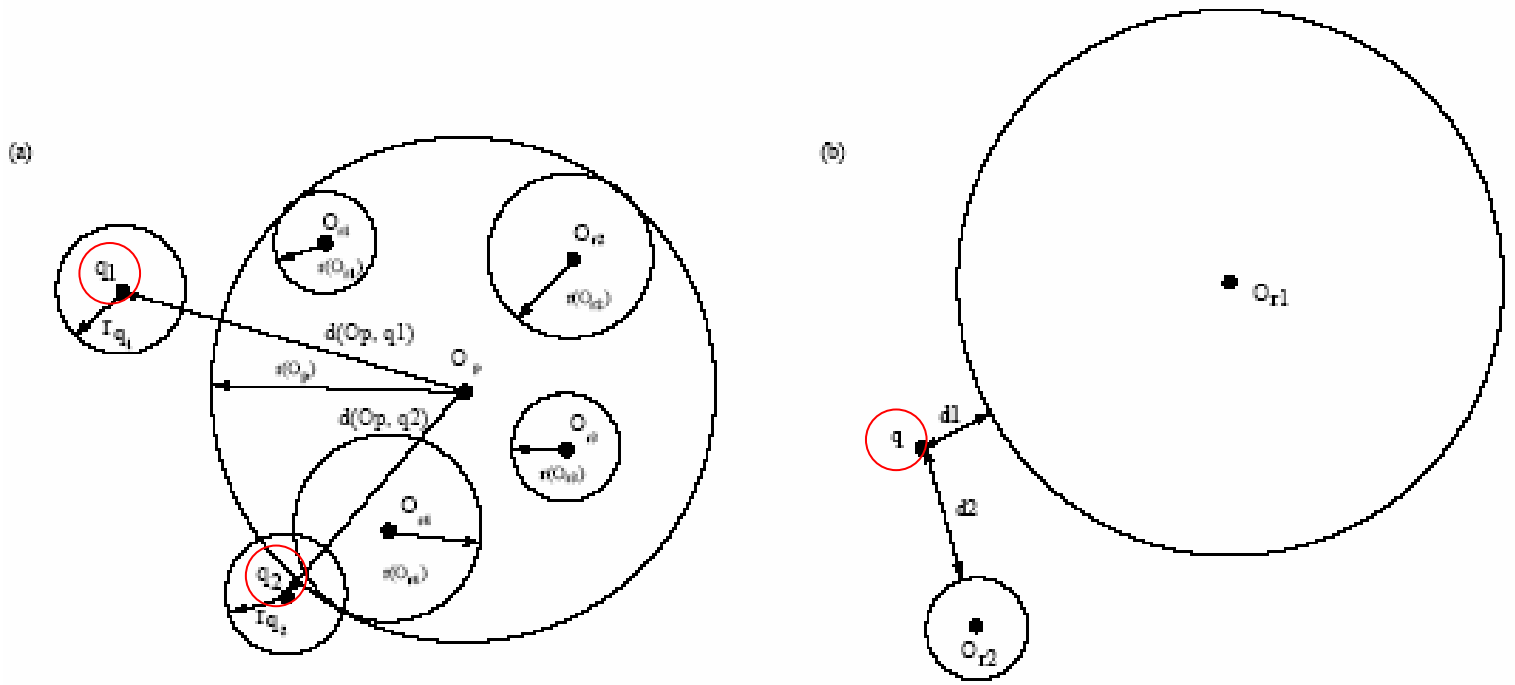
- 既采用了最小边界圆（MBS），也采用了最小边界矩形（MBR）。
- 相对于SS-Tree，减小了区域的面积，提高了区域之间的分离性。
- 相对于R*-Tree，提高了最邻近查询的性能。

M - 树

- M-树是为对非特定度量空间(a generic metric space)的数据集进行搜索而设计。
- M-树是用距离函数进行对象的索引，它既不要求向量空间也不是用线性度量(Lp metric)，从而使得搜索的应用场合大大扩大。
- 空间中对象的相似度由距离函数定义，距离函数满足非负、对称和三角不等三个基本条件。
- minimize the number of distance calculations as well as the number of disk accesses.
- It is a paged, dynamic, height-balanced tree which uses only the distance between objects for indexing.

M - 树

- The M-tree is suitable for range queries and K-nearest neighbor (K-NN) queries.



(a)Pruning nodes in the M-tree and (b)The order nodes are visited

R-Tree维度限制

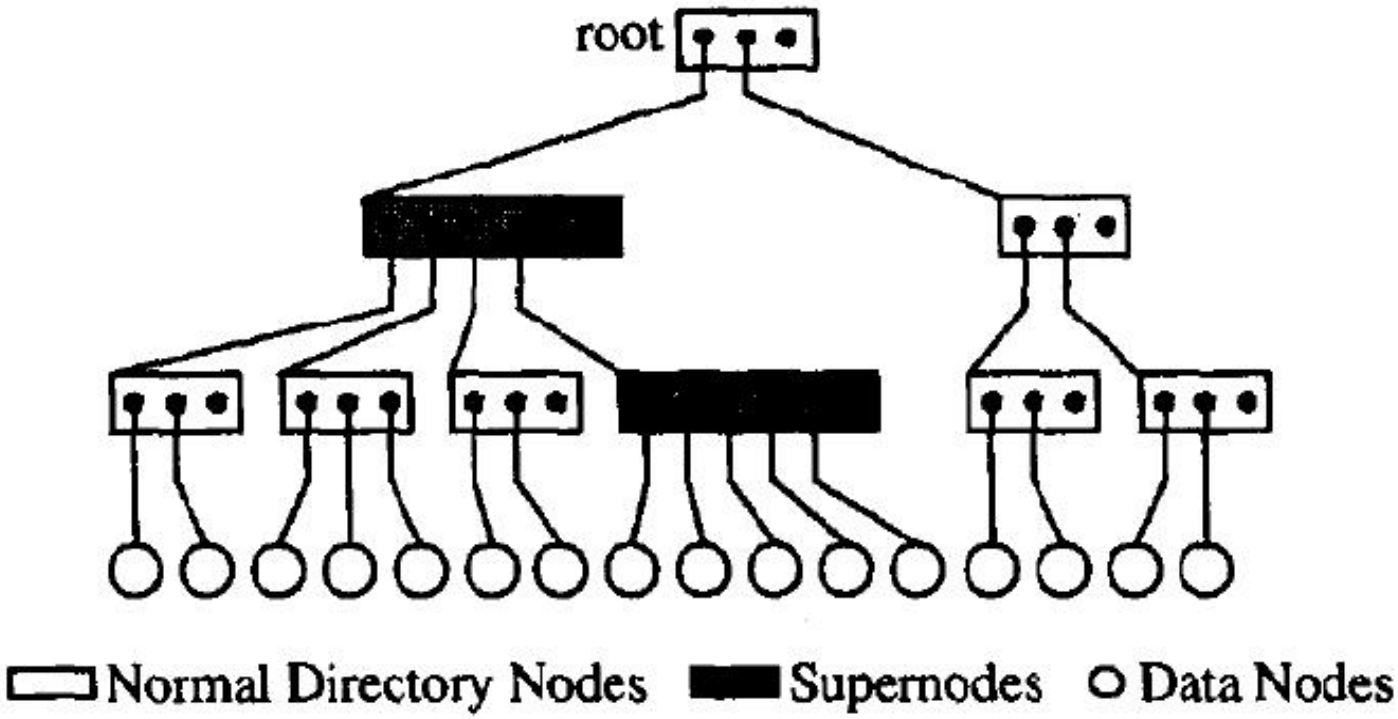
- 当维数增加到5时，R-Tree及其变种中的边界矩形的重叠将达到90%，因此在高维（high-dimension）情况（>=5)下，其性能将变得很差，甚至不如顺序扫描。

X-Tree

(S. Berchtold VLDB'1996)

- X-Tree是线性数组和层状的R-Tree的杂合体，通过引入超级结点(supernode)，大大地减少了最小边界矩形之间的重叠。提高了查询的效率。
  - X-树在侦测到所有可能的分裂都会造成大的重叠时，通过不分裂结点的策略来达到降低重叠的目的（超级结点 Super node）。
  - 问题：一个超结点的孩子个数是不固定的，且有可能超结点会很大(尤其是在维度很高的情况下)，这会使得搜索超结点的时间变长。

X-Tree的结构

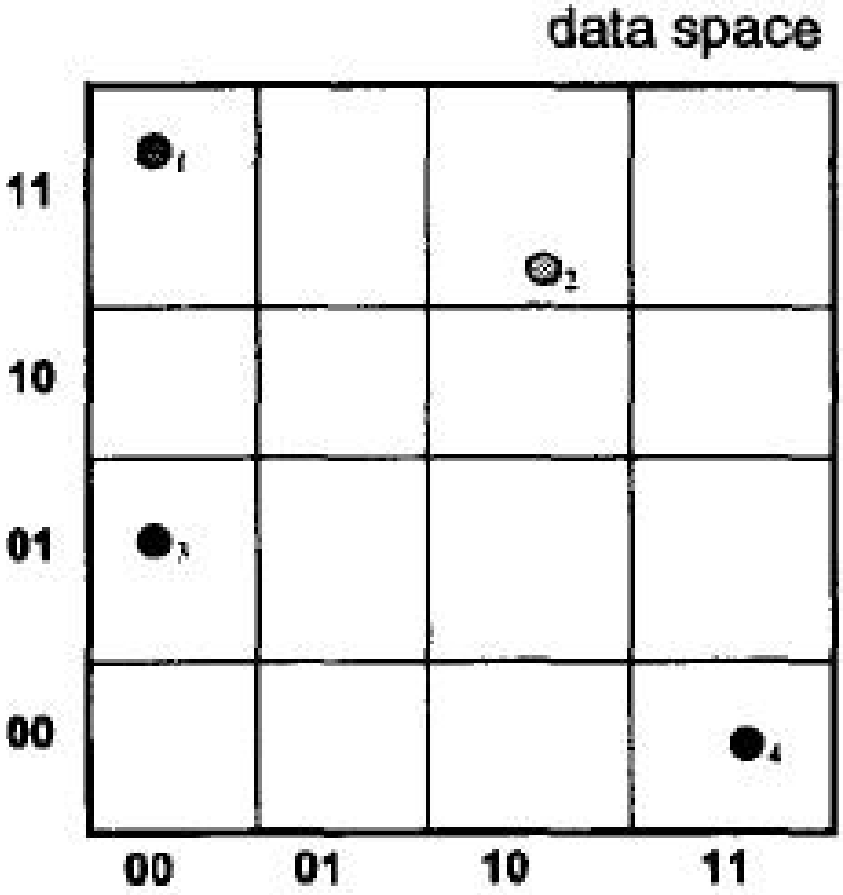


VA-File

(R. Weber VLDB'1998)

- VA-File（Vector Approximation File）
- 一种简单但非常有效的方式
- 将数据空间划分成2^b单元（cell），
  - b表示用户指定的二进制位数，每个单元分配一个位串。
- 单元内的向量用这个单元近似代替。
- VA-File本身只是这些近似体的数组。
- 查询时，先扫描VA-File，选择侯选向量，再访问向量文件进行验证。

VA-File的建立



vector data

$\bullet_1$	0.1 0.9
$\bullet_2$	0.6 0.8
$\bullet_3$	0.1 0.4
$\bullet_4$	0.9 0.1

approximation

$\bullet_1$	00 11
$\bullet_2$	10 11
$\bullet_3$	00 01
$\bullet_4$	11 00

A-Tree

(Y. Sakurai VLDB'2000)

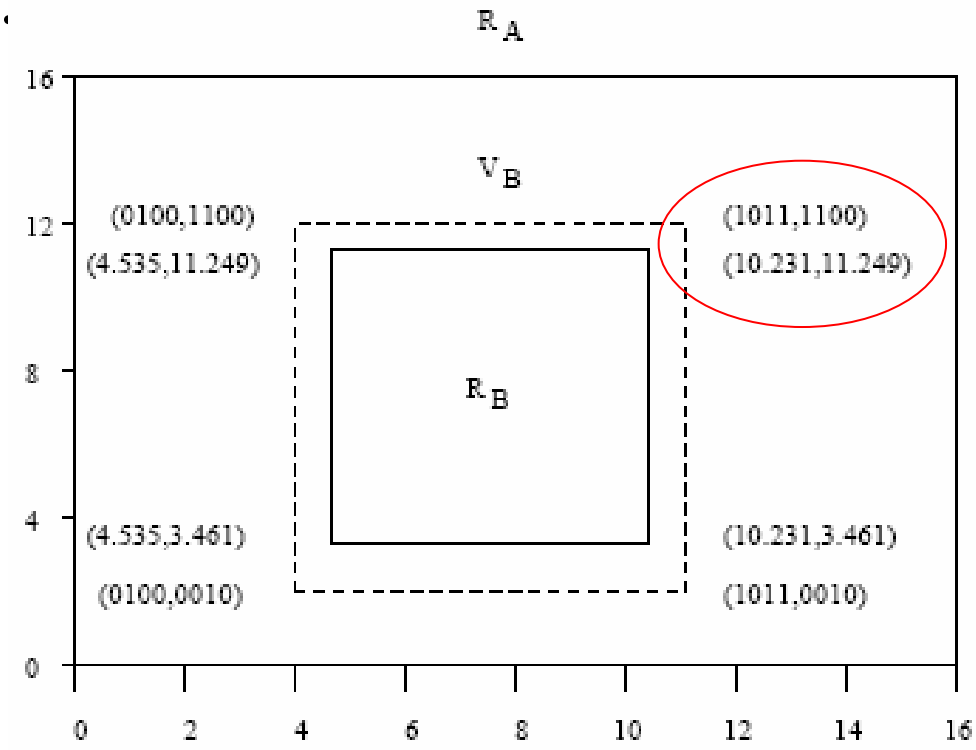
- 吸取SR-Tree和VA-File 的长处
- 引入虚拟边界矩形VBR（Virtual Bounding Rectangles）

A-Tree

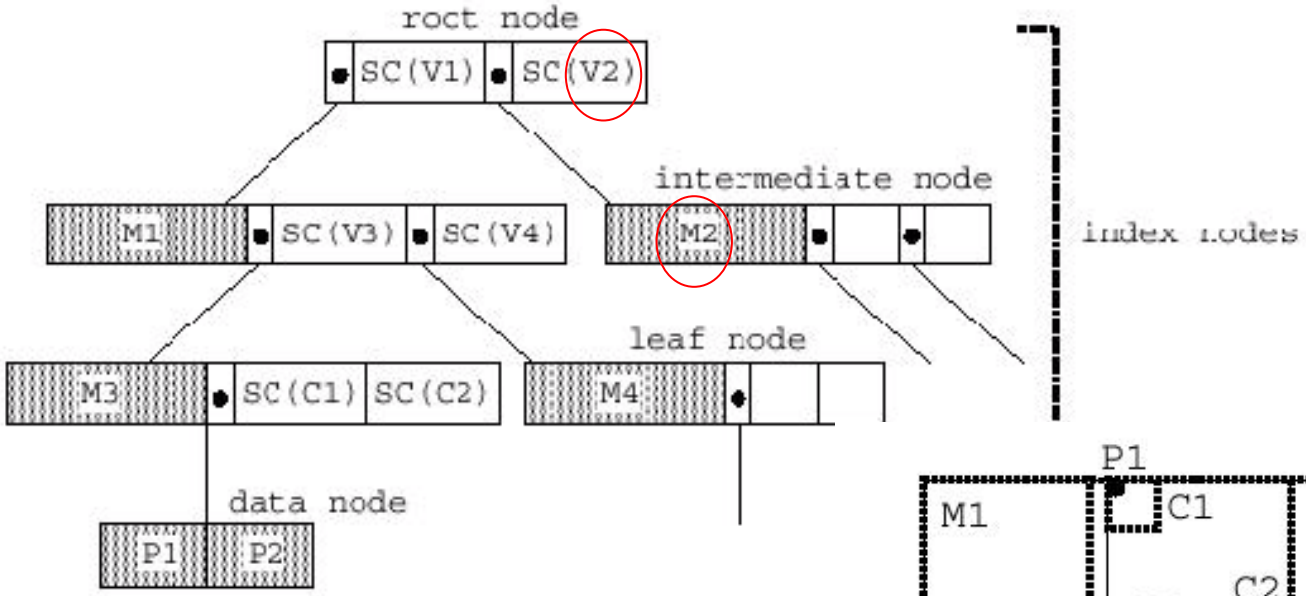
(Y. Sakurai VLDB'2000)

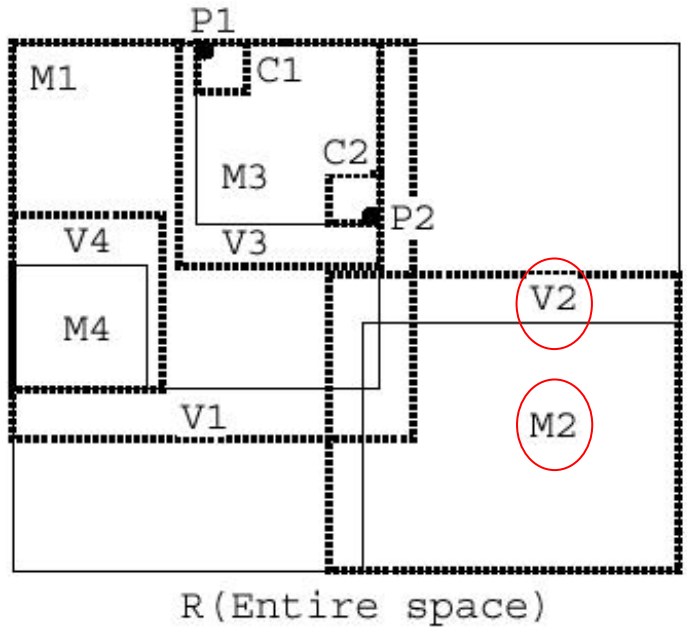
- it uses Virtual Bounding Regions (VBR) to approximate MBRs
- which can be stored more compactly to increase the fanout of the tree.

- the coordinates of VB are stored as binary
- the coordinates of RB are stored as floats.



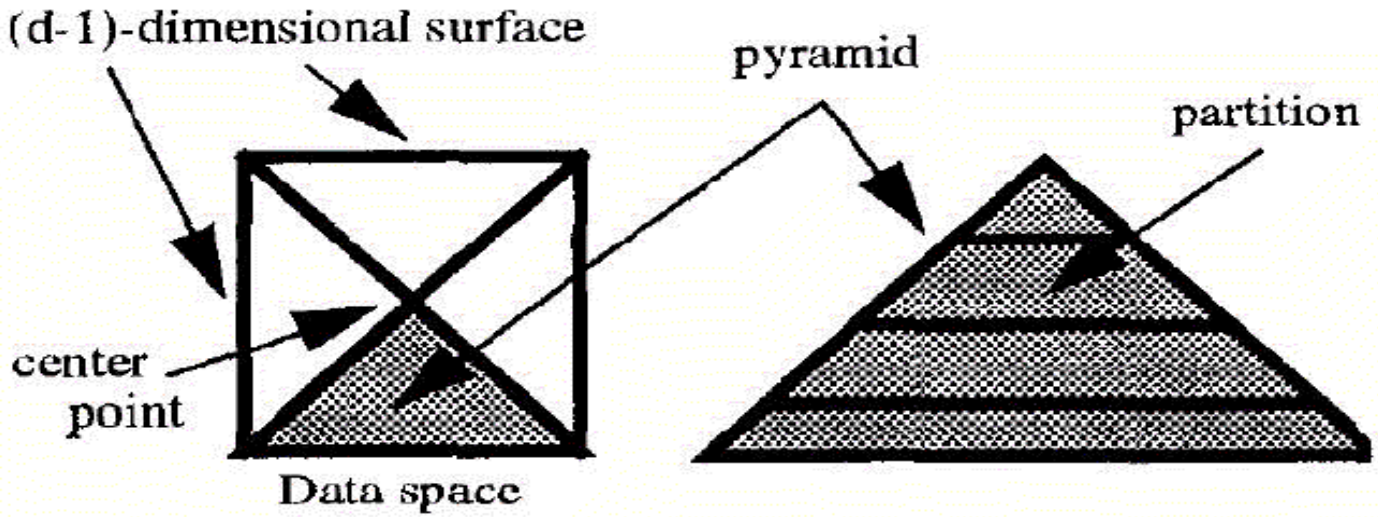
A-Tree的结构


root node
SC(V1)
SC(V2)
intermediate node
M1
SC(V3)
SC(V4)
M2
index nodes
leaf node
M3
SC(C1)
SC(C2)
M4
data node
P1
P2


P1
M1
C1
C2
M3
P2
V4
V3
V2
M4
V1
M2
R(Entire space)

Pyramid-Tree (S. Berchtold SIGMOD'1998)

- 数据分布在d-维单位超立方体内
- 将数据空间划分成2d个金字塔，这些金字塔以数据空间的中心（0.5, 0.5, …, 0.5）为顶点，以数据空间的(d-1)-维表面（立方体的面）作为基座。
- 每个金字塔划分成多个平行于基座的部分，
- 每个部分对应于B+-树的一个数据页。

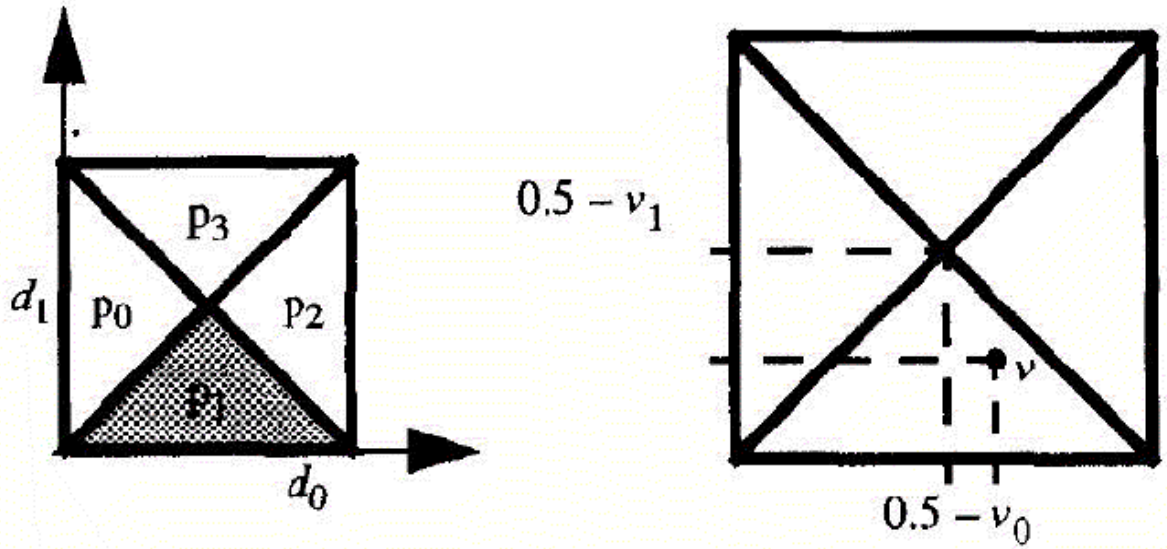


Pyramid-Tree (S. Berchtold SIGMOD'1998)

- 在第i个Pyramid中的所有的点

$$\forall j,\ 0 \le j < d, j \ne i:\ (|0.5 - v_i| \le |0.5 - v_j|) \qquad if(i < d)$$

$$\forall j,\ 0 \le j < d, j \ne (i-d):\ (|0.5 - v_{(i-d)}| \ge |0.5 - v_j|) \qquad if(i \ge d)$$

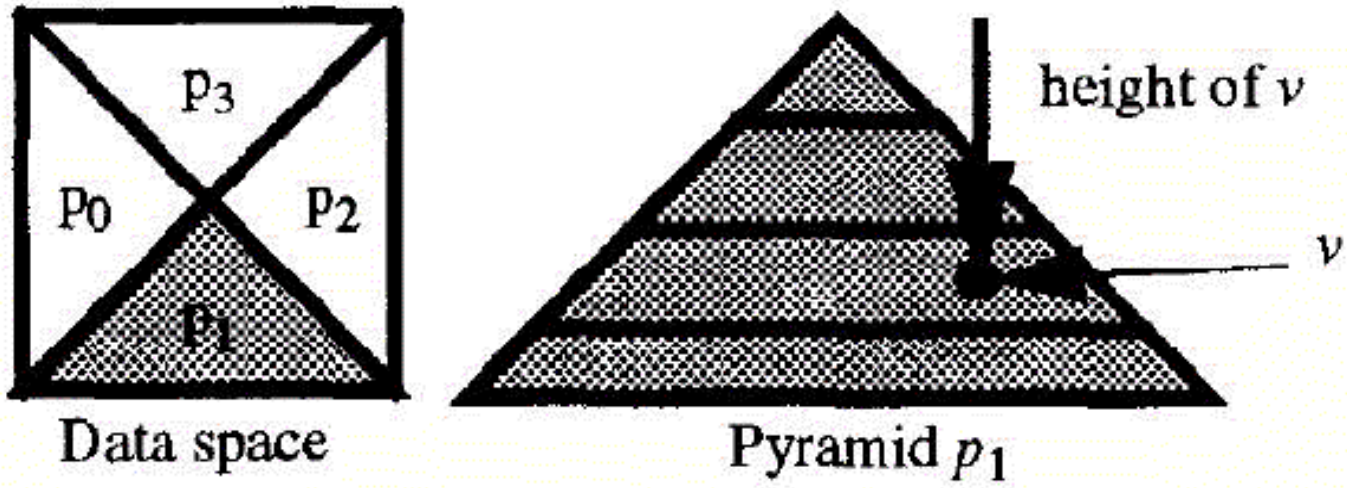


a) numbering of pyramids　　b) point in pyramid

Pyramid-Tree: 点的高度

- 给定一个d-维点v。设p_i是点v所在的金字塔。那么，点v的高度h_v定义为

$$h_v = |0.5 - v_{i \ MOD \ d}|$$



Pyramid-Tree: 点的Pyramid值

- 给定一个d-维点v。设p_i是点v所在的金字塔，h_v是v的高度，那么，v的Pyramid值定义为：

$$pv_v = (i + h_v)$$

这里i是整数，而h_v是区间[0,0.5]中的一个实数

将d-维数据点转换成1-维的数值

然后用一种有效的索引结构如B^+-树进行数值的存取。

Clindex
(C. Li, *Trans* KDE 2002)

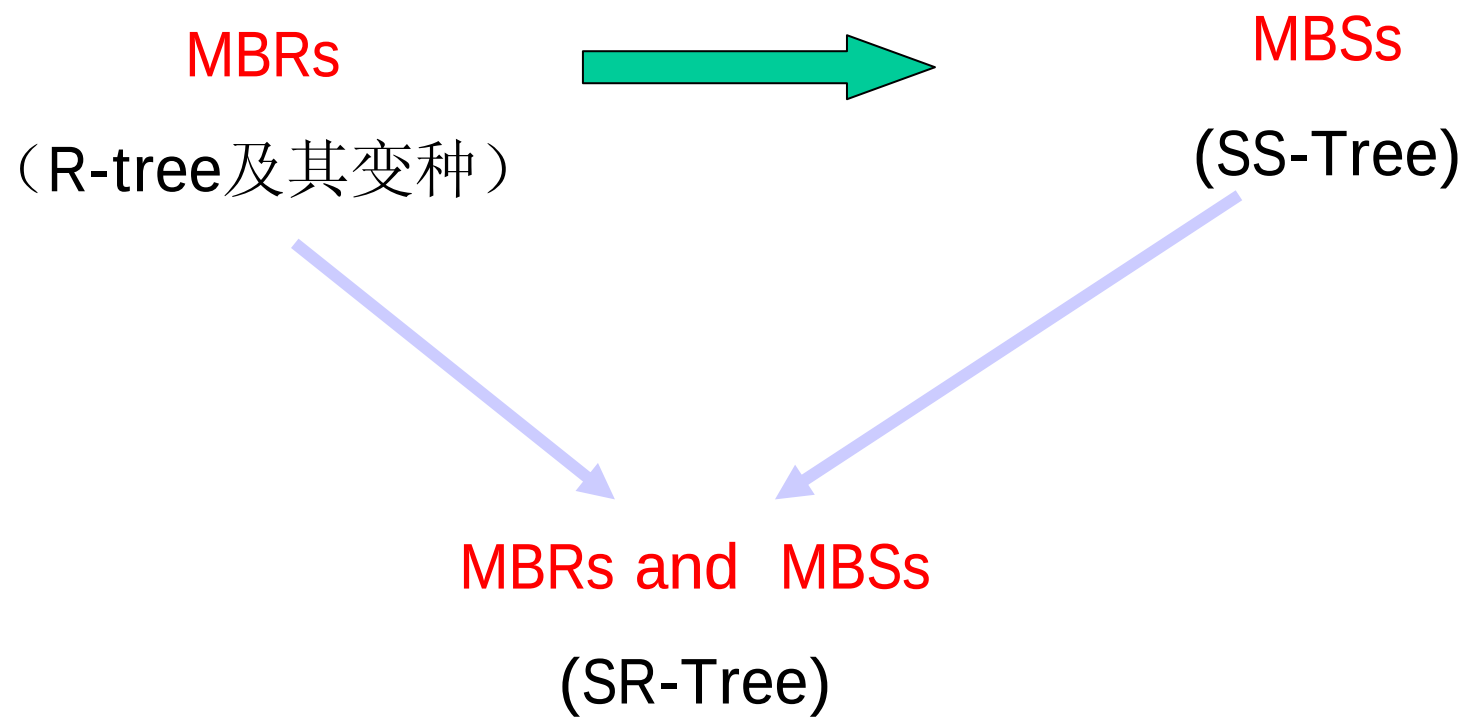
- Clindex (CLustering for INDEXing)
- 数据集首先按相似性进行聚类，每个聚类被存储在一个顺序文件中；
- 为索引聚类建立一张一维映射表；
- 对于一个查询，将查询点附近的聚类提取到内存中，并通过计算各点与查询点的距离，而获得结果。
- 这一方法大大提高了搜索的召回率及IO性能
- 主要用于对离线的静态数据集进行分析。

多维索引方法列表

- K-D-B Trees（J.T.Robinson SIGMOD’1981）
- R-tree（A. Guttman SIGMOD’1984）
- R^+-tree （T. Sellis VLDB’1987）
- R*-Tree（N. Beckmann SIGMOD’1990）
- SS-Tree（D. A. White ICDE’1996 ）
- SR-Tree（N. Katayama SIGMOD’1997）
- M-Tree（P.Ciaccia VLDB’1997）
- VA-File（R. Weber VLDB’1998）
- Pyramid-Tree(S.Berchtold SIGMOD’1998)
- A-Tree（Y. Sakurai VLDB’2000）

多维索引方法的演变

边界形状



多维索引方法的演变

索引结构

树结构

（R-Tree SS-Tree SR-Tree等）

中低维

顺序结构

（线性扫描、VA-File等)

高 维

树结构和顺序结构的混合体

（A-Tree)

多维索引方法的演变

分割方法

空间分割

（K-D-B Tree
grid file等)

数据分割

（R-Tree SR-Tree
X-Tree A-Tree等

适合数据均匀分布

多维索引方法的演变

对象的表示方法

精确表示

（R-Tree X-Tree
顺序扫描等）

近似表示

（VA-File A-Tree)

本章总结

- 信息检索概述
- 信息检索模型
- 检索质量评价
- 倒排索引
- 检索系统介绍
  - Lucene
- 问题与挑战
- 高维索引

Any Question?