数据库系统

哈尔滨工业大学（深圳）

第18讲 数据库索引

本讲学习什么?

基本内容

1. 为什么需要索引与什么是索引
2. 索引的简单分类
3. B+树索引
4. 散列索引

重点与难点

●理解索引的作用，掌握应用索引改进数据库查询性能的方法

●理解不同类型索引的概念：稠密索引与稀疏索引，主索引与辅助索引，聚簇索引与非聚簇索引，倒排索引，多级索引等

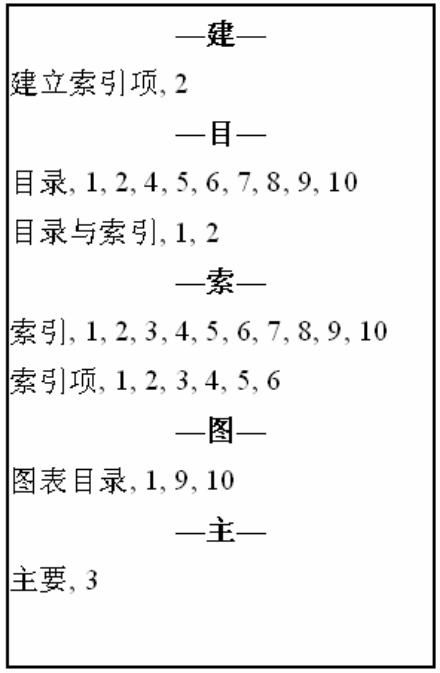
●理解B+树索引，怎样建立、维护和利用B+树索引(算法层面)

●理解散列索引，包括静态散列索引与动态散列索引(算法层面)

什么是及为什么需要索引

书籍、词典/字典中的词汇表(索引表)

- ✓词汇表等包含两部分：词条，词条在文件中的页码
- ✓词汇表等通常按词条进行某种方式的排序
- ✓目的是通过对“有序的小数据量的词汇表”的快速查找，发现词条在 “大数据量书籍中”出现的位置。



—建—

建立索引项, 2

—目—

目录, 1, 2, 4, 5, 6, 7, 8, 9, 10

目录与索引, 1, 2

—索—

索引, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

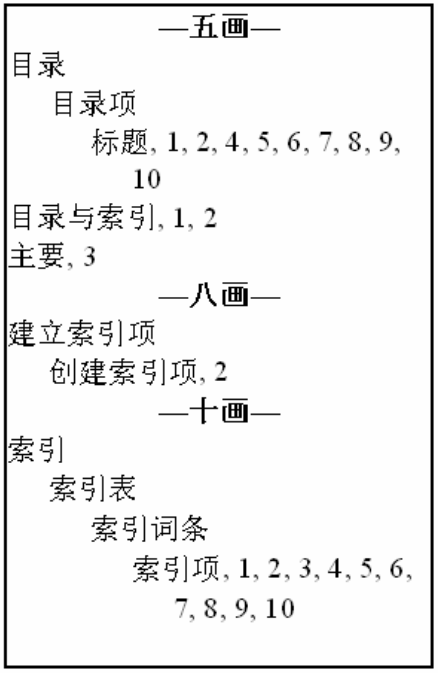
索引项, 1, 2, 3, 4, 5, 6

—图—

图表目录, 1, 9, 10

—主—

主要, 3



—五画—

目录

目录项

标题, 1, 2, 4, 5, 6, 7, 8, 9, 10

目录与索引, 1, 2

主要, 3

—八画—

建立索引项

创建索引项, 2

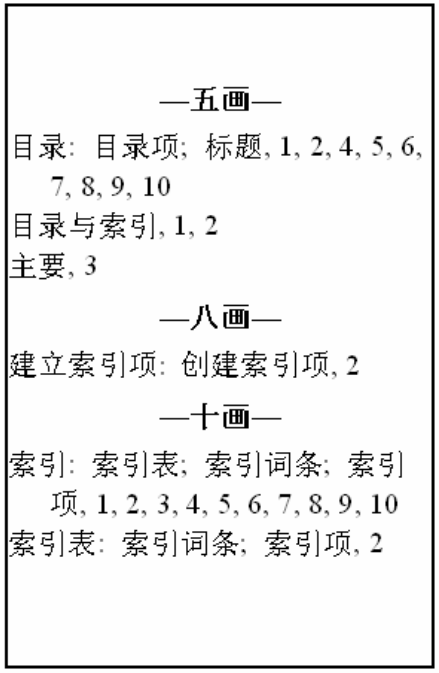
—十画—

索引

索引表

索引词条

索引项, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



—五画—

目录: 目录项; 标题, 1, 2, 4, 5, 6, 7, 8, 9, 10

目录与索引, 1, 2

主要, 3

—八画—

建立索引项: 创建索引项, 2

—十画—

索引: 索引表; 索引词条; 索引项, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

索引表: 索引词条; 索引项, 2

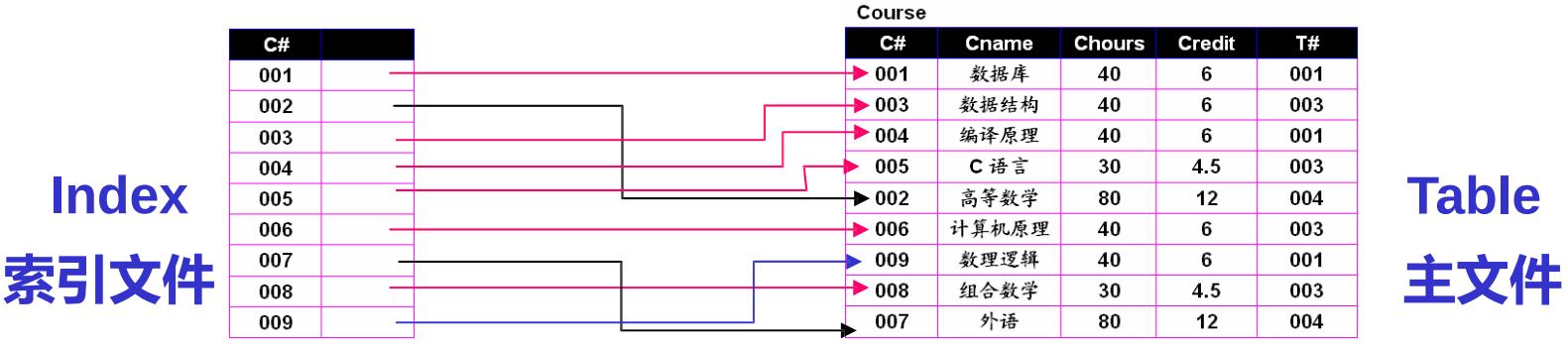
索引概念

索引： 定义在存储表(Table)基础之上，无需检查所有记录，快速定位所需记录的一种**辅助存储结构**，由一系列存储在磁盘上的**索引项**(index entries)组成，每一索引项又由两部分构成:

- ✓ **索引字段**：由Table中某些列**(通常是一列)**中的值串接而成。索引中**通常**存储了索引字段的每一个值(也有不是这样的)。索引字段类似于词典中的**词条。**
- ✓ **行指针**：指向Table中包含索引字段值的记录在磁盘上的存储位置。行指针类似于词条在书籍、词典中出现的**页码。**

●存储索引项的文件为**索引文件**，相对应，存储表又称为**主文件**

Index-field	pointer



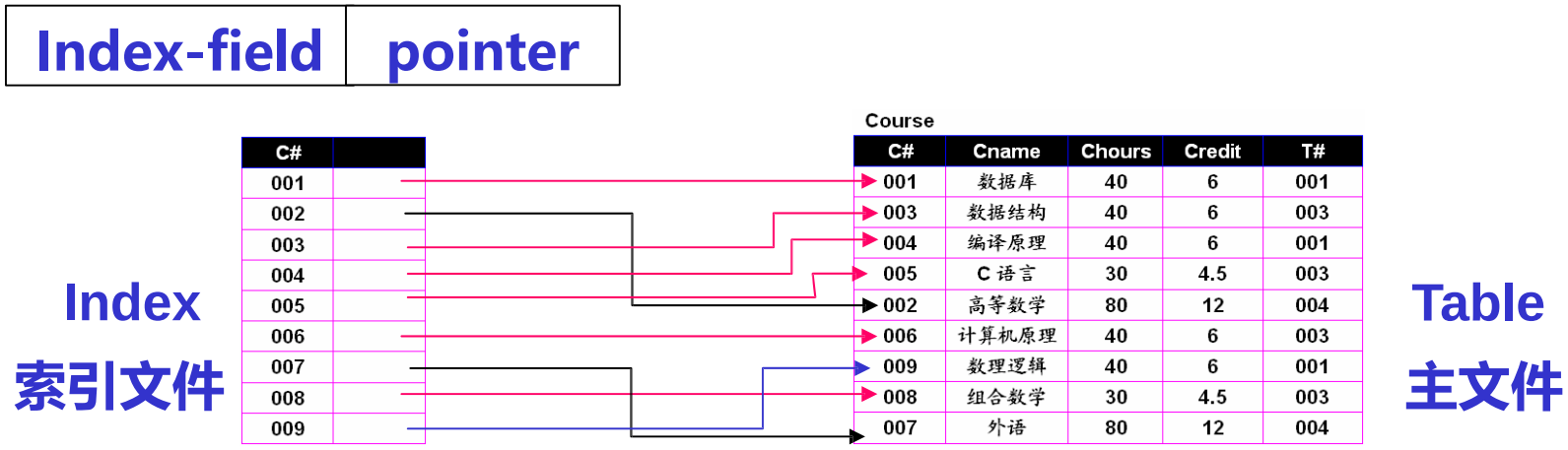
索引一般性特点

●索引文件是一种辅助存储结构，其**存在与否不改变存储表的物理存储结构**；目的提高存储表的访问速度。

●**索引文件组织方式有两种：**(相对照的，主文件组织有堆文件、排序文件、散列文件、聚簇文件等多种方式)

✓**排序索引文件**(Ordered indices): 按索引字段值的某一种顺序组织存储

✓**散列索引文件**(Hash indices)：依据索引字段值使用散列函数分配散列桶的方式存储

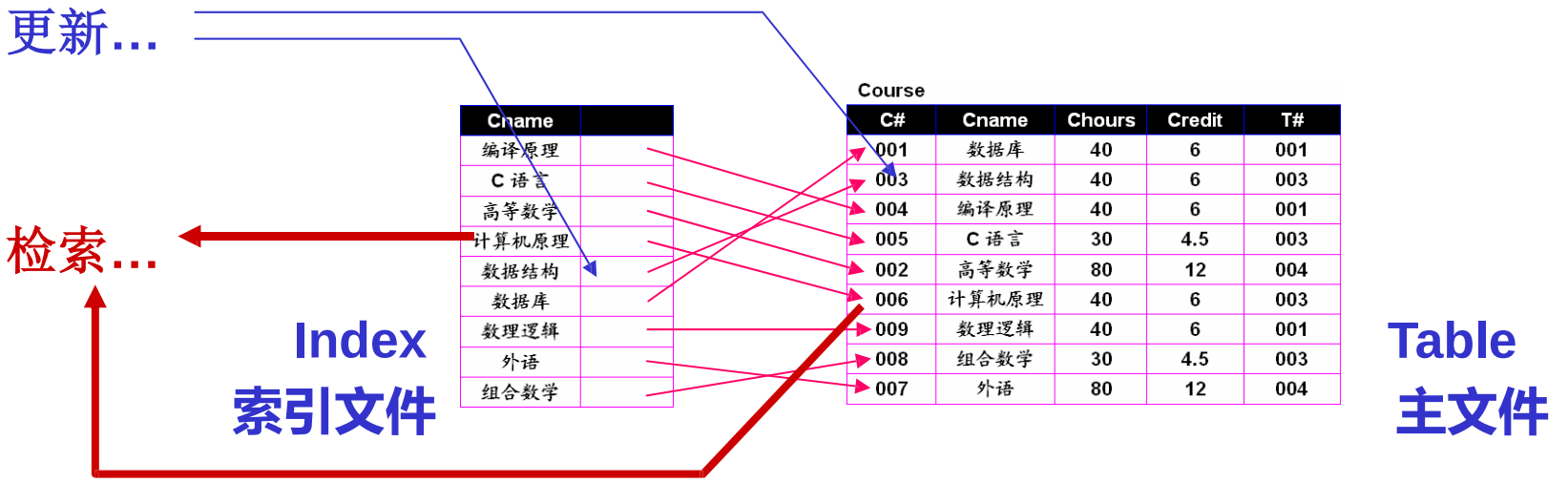


索引一般性特点

●在一个表上，可针对不同属性或属性组合，建立不同的索引文件，可建立多个索引文件。（索引字段的值可以是Table中的任何一个属性的值或任何多个属性值的组合值）

●索引文件比主文件小很多。通过检索一个小的索引文件(可全部装载进内存)，快速定位后，再有针对性的读取非常大的主文件中的有关记录

●有索引时，更新操作必须同步更新索引文件和主文件。否则...



索引应用的评价问题

关于索引应用的评价

- 索引技术应用使检索效率大幅度提高，但同时其也增加了存储空间、使维护负担加重(不仅要维护主文件，而且要维护索引文件)
- **衡量索引性能好坏：**
  - ✓**访问时间**
  - ✓**插入时间**
  - ✓**删除时间**
  - ✓**空间负载**
  - ✓**支持存取的有效性**, 比如：支持的是属性的限定值(是否符合单一值)，还是支持属性的限定范围的值(是否符合一定范围)

对哪些属性建立索引?

对经常出现在检索条件、**连接条件、分组计算条件中的属性可建立索引**

SELECT…FROM…WHERE…GROUP BY…

索引概念区分

几个概念的澄清

- 字段、排序字段、索引字段
- **码(Key)、主码(Primary Key), 又称为表键(Table Key)**---具有唯一性和最小性
- **排序码**(Order Key)---对主文件进行排序存储的那些属性或属性组
- **索引码**(Index Key)---即索引字段，不一定具有唯一性
- **搜索码**(Search Key)---在主文件中查找记录的属性或属性集

(教材中把“索引字段”统称为“索引码(Index key)”)请注意区分

- **主文件** vs. **索引文件**

讲义中

Index-field	pointer

教材中

Index-key	pointer

SQL语言中的索引创建与维护

SQL语言关于索引的基本知识

●当定义Table后，如果定义了主键，则系统将自动创建主索引，利用主索引对Table进行快速定位、检索与更新操作;

●索引可以由用户创建，也可以由用户撤消

●当索引被创建后，无论是主索引，还是用户创建的索引，DBMS都将自动维护所有的索引，使其与Table保持一致，即：当一条记录被插入到Table中后，所有索引也自动的被更新

●当Table被删除后(drop table), 定义在该Table上的所有索引将自动被撤消

创建维护索引语句

●索引可以由用户在任何属性上创建

X/OPEN SQL中关于索引的语句

➢创建索引：

```
CREATE  [unique]  INDEX  indexname
        ON  tablename ( colname [asc | desc]
                {, colname [asc | desc] . . .} );
```

●示例：在student表中创建一个基于Sname的索引

```
create  index  idxSname  on  student(sname);
```

●示例：在student表中创建一个基于Sname和Sclass的索引

```
create  index  idxSnamcl  on  student(sname, sclass);
```

●如某索引不再需要，则可通过撤消命令，撤消用户创建的索引

```
DROP  INDEX  indexname;
```

索引应用要注意效果

●选择哪些属性创建索引，以及如何创建与维护索引，如何利用索引改善数据库的运行性能，是DBA(数据库管理员)的重要职责。

●是否建立和在哪些属性上建立索引需要考虑：访问时间、插入时间、删除时间与空间负载。 既要改善性能，又要控制代价。

●建立索引还需考虑索引的类型：索引如何支持存取的有效性

比如：支持的是属性的限定值(是否符合单一值)，还是支持属性的限定范围的值(是否符合一定范围)

对经常出现在检索条件、连接条件、分组计算条件中的属性可建立索引

SELECT…FROM…WHERE…GROUP BY…

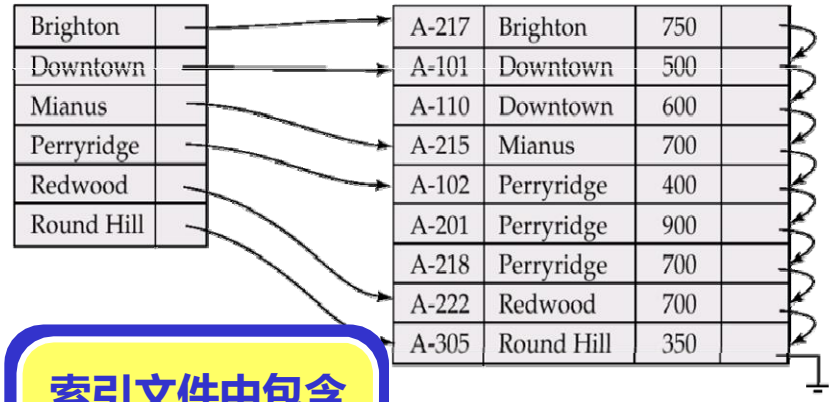
稠密索引与稀疏索引

稠密索引与稀疏索引

❑对于主文件中每一个记录(形成的每一个索引字段值)，都有一个索引项和它对应，指明该记录所在位置。这样的索引称稠密索引(dense index)

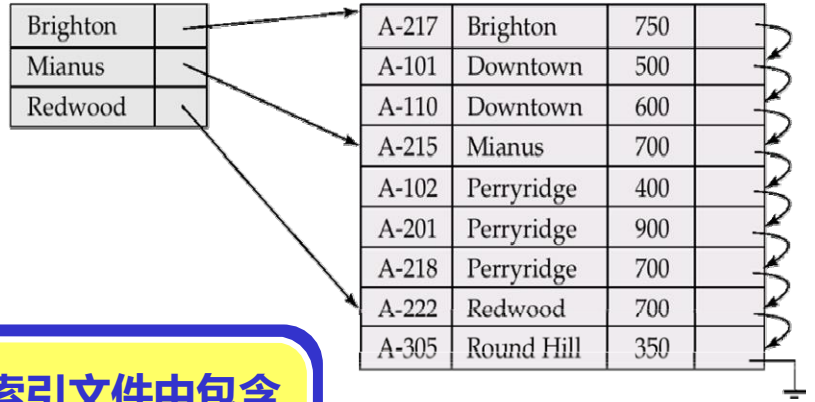
❑对于主文件中部分记录(形成的索引字段值)，有索引项和它对应，这样的索引称非稠密索引(undense index)或稀疏索引(sparse index)

稠密索引



索引文件中包含了主文件对应字段的**所有**不同值

稀疏索引



索引文件中包含了主文件对应字段的**部分**不同值

稀疏索引定位记录

稀疏索引如何定位记录

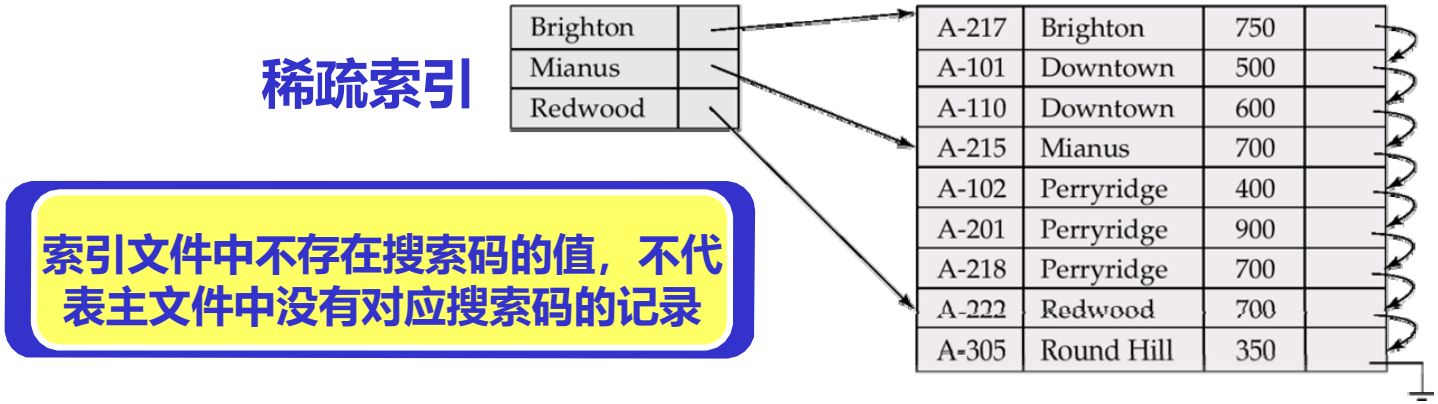
✓定位索引字段值为 K的记录，需要

- ❖首先找相邻的小于K的，最大索引字段值，所对应的索引项
- ❖从该索引项所对应的记录，开始顺序进行Table的检索

✓稀疏索引的使用要求—主文件必须是按对应索引字段属性排序存储

✓相比稠密索引：空间占用更少，维护任务更轻，但速度更慢

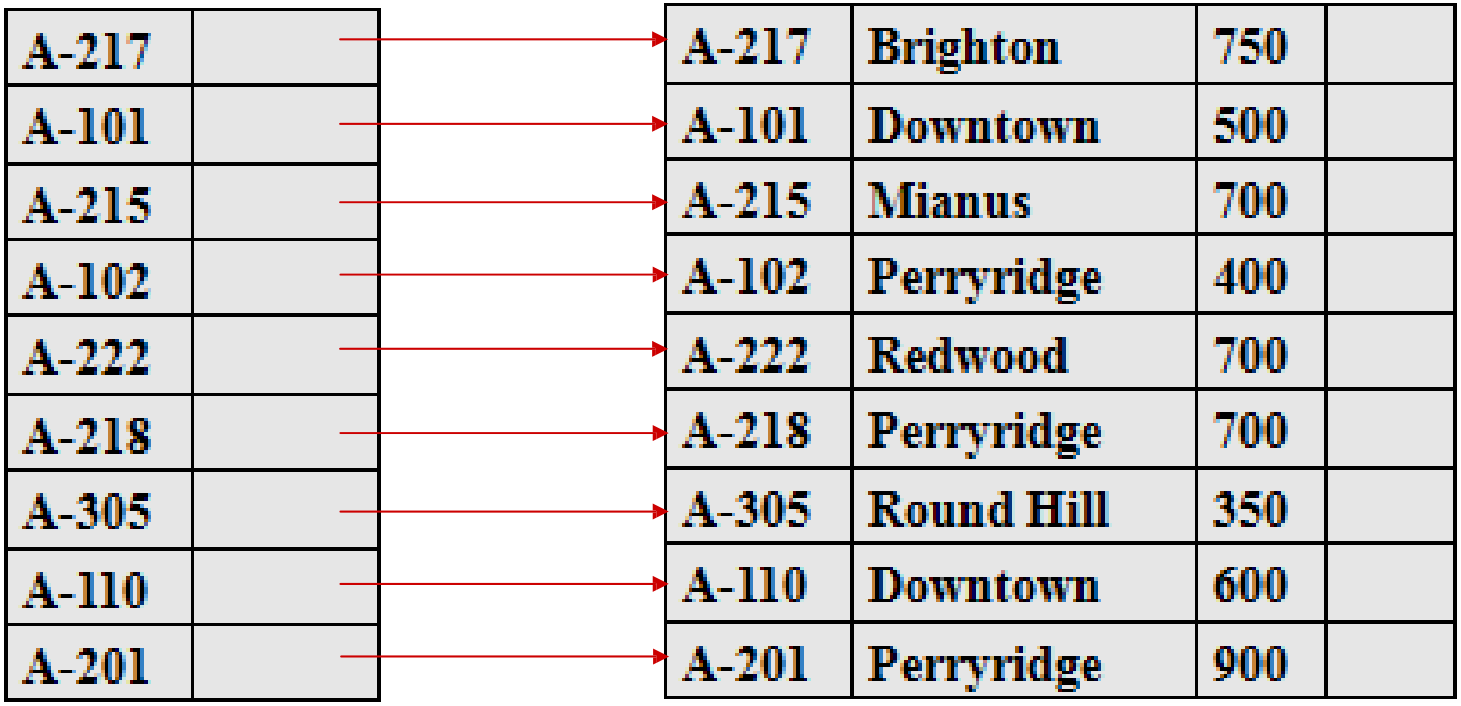
✓平衡: 索引项不指向记录指针，而是指向记录所在存储块的指针，即每一存储块有一个索引项，而不是每条记录有一索引项----**主索引**



稠密索引定位记录

候选键属性的稠密索引—先查索引，然后再依据索引读主文件

无论是候选键属性的稠密索引，还是非候选键属性的稠密索引：
索引文件中不存在搜索码的值，那么主文件中没有对应搜索码的记录

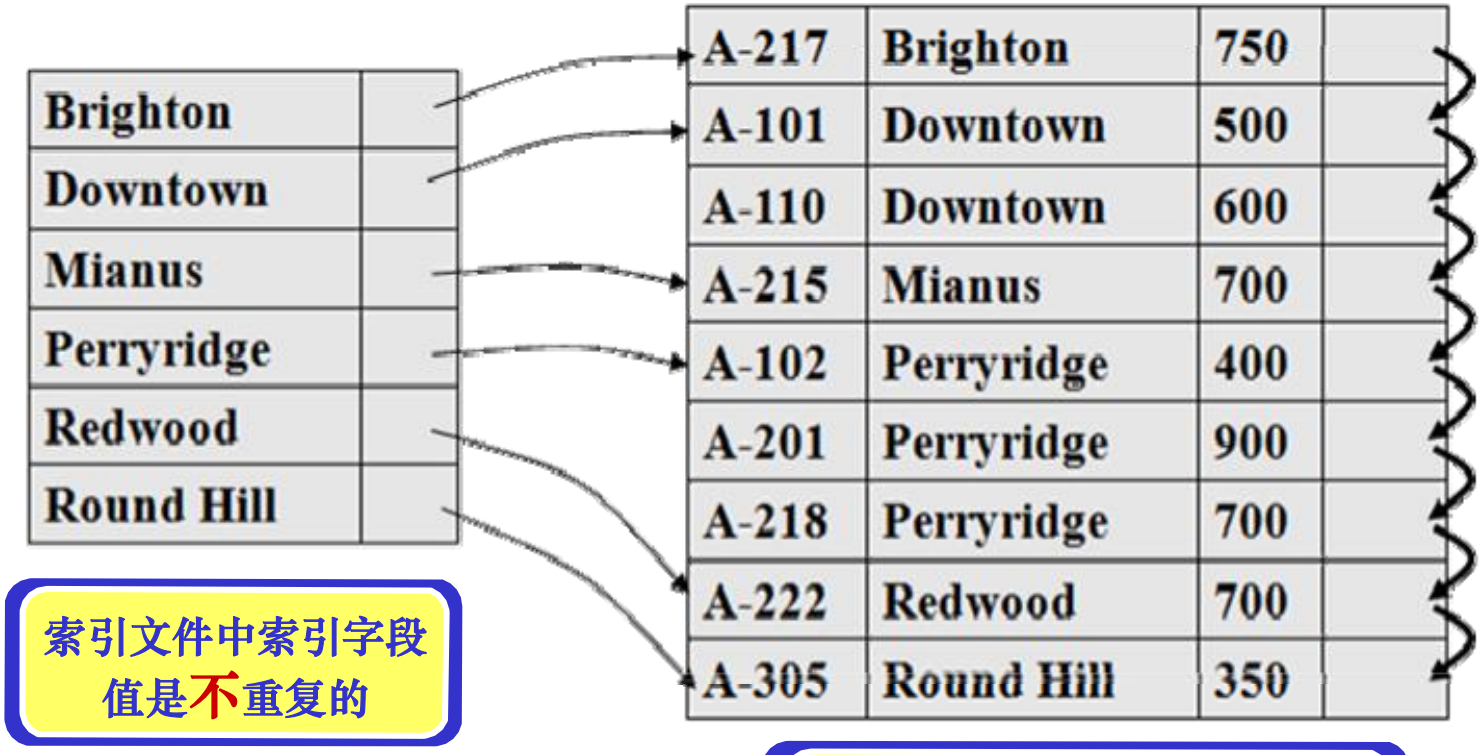


A-217	
A-101	
A-215	
A-102	
A-222	
A-218	
A-305	
A-110	
A-201	

A-217	Brighton	750	
A-101	Downtown	500	
A-215	Mianus	700	
A-102	Perryridge	400	
A-222	Redwood	700	
A-218	Perryridge	700	
A-305	Round Hill	350	
A-110	Downtown	600	
A-201	Perryridge	900	

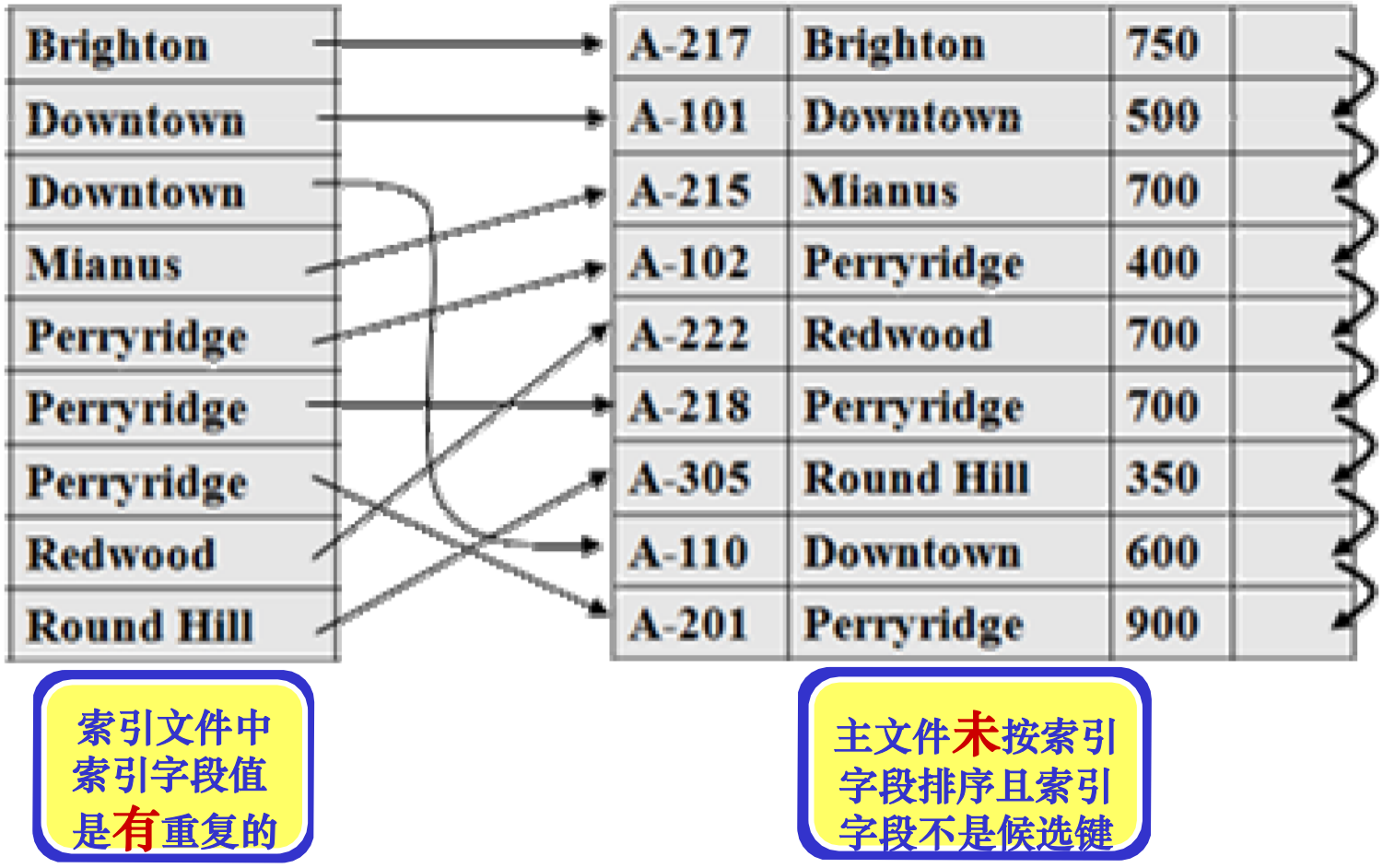
稠密索引定位记录

非候选键属性的稠密索引（I）



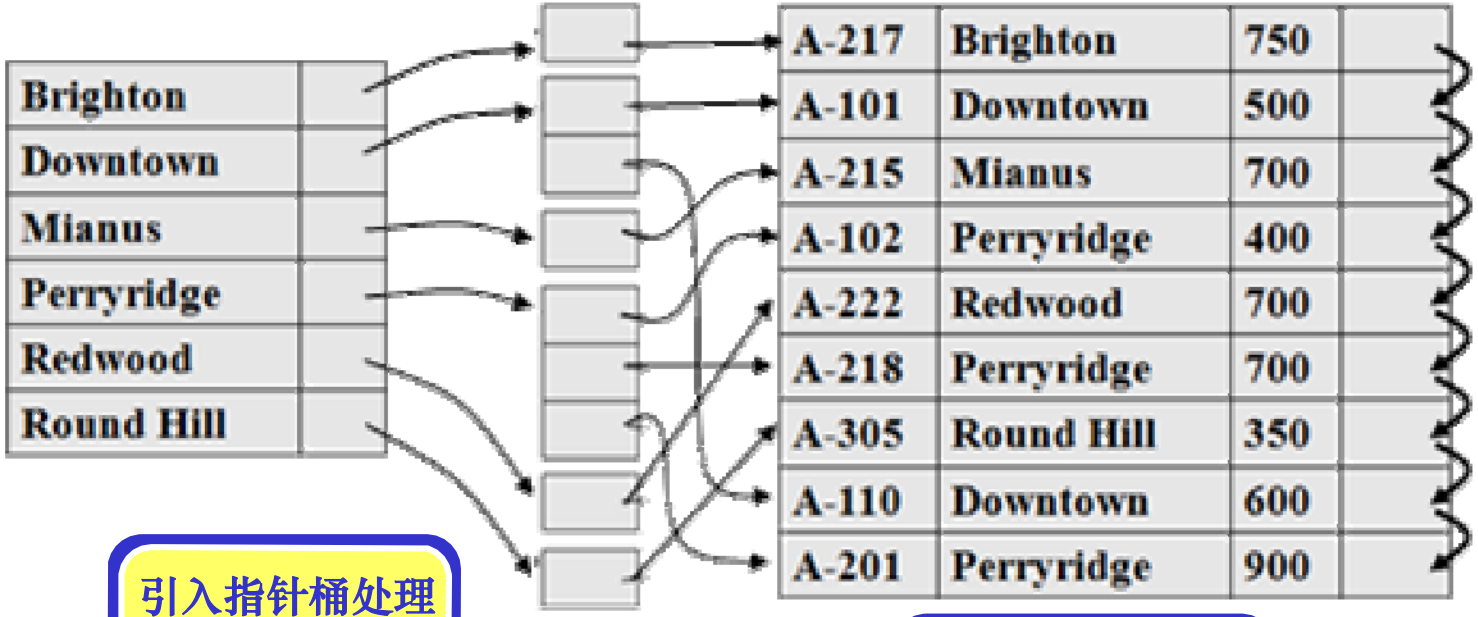
稠密索引定位记录

非候选键属性的稠密索引（II）



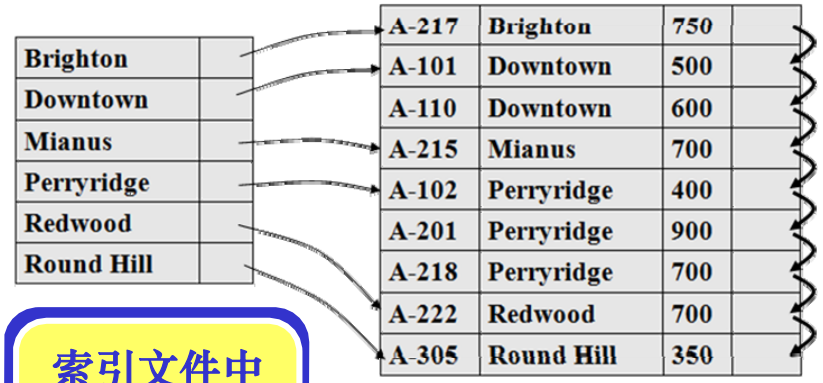
稠密索引定位记录

非候选键属性的稠密索引（III）



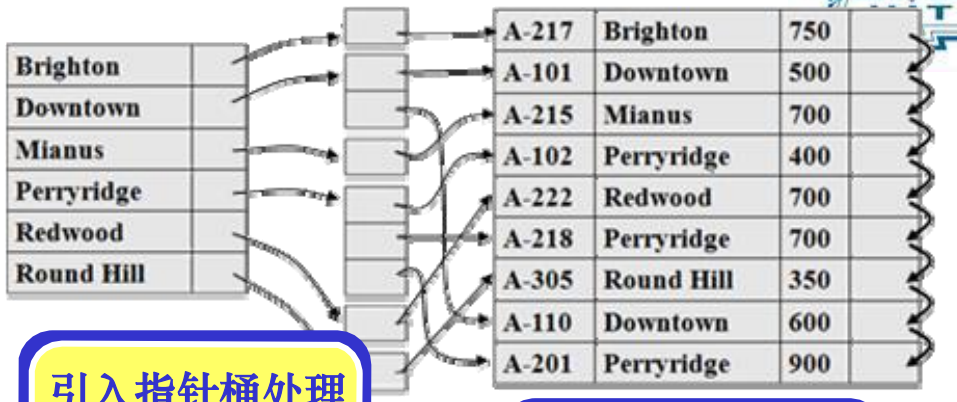
稠密索引定位记录

非候选键属性的稠密索引的三种情况之对比



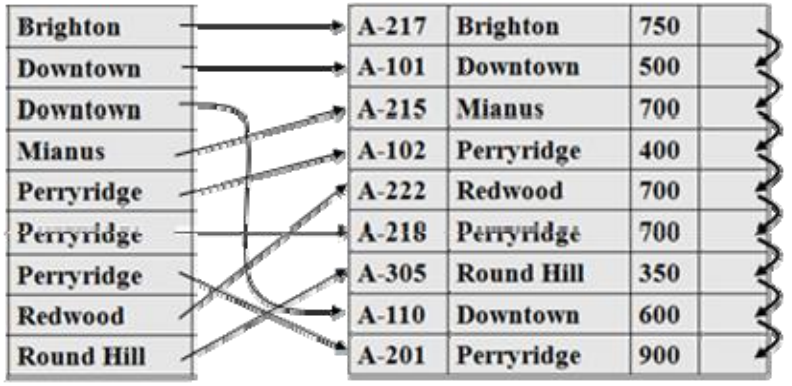
索引文件中索引字段值是**不**重复的

主文件按索引字段**排序**且索引字段不是候选键



引入指针桶处理非候选键索引的多记录情况

主文件**未**按索引字段排序且索引字段不是候选键



索引文件中索引字段值是**有**重复的

主文件**未**按索引字段排序且索引字段不是候选键

主索引与辅助索引

主索引 **通常是对每一存储块有一个索引项，索引项的总数和存储表所占的存储块数目相同，存储表的每一存储块的第一条记录，又称为锚记录(anchor record), 或简称为块锚(block anchor)**

- **主索引的索引字段值为块锚的索引字段值，而指针指向其所在的存储块。**
- 主索引是按索引字段值进行排序的一个有序文件，**通常**建立在有序主文件的基于主码的排序字段上，即主索引的索引字段与主文件的排序码(主码)有对应关系
- 主索引是稀疏索引。

主索引文件

C#	
001	
004	
007	

块锚主码值 **块指针**

主文件

C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
002	高等数学	80	12	004
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
006	计算机原理	40	6	003
007	外语	80	12	004
008	组合数学	30	4.5	003
009	数理逻辑	40	6	001

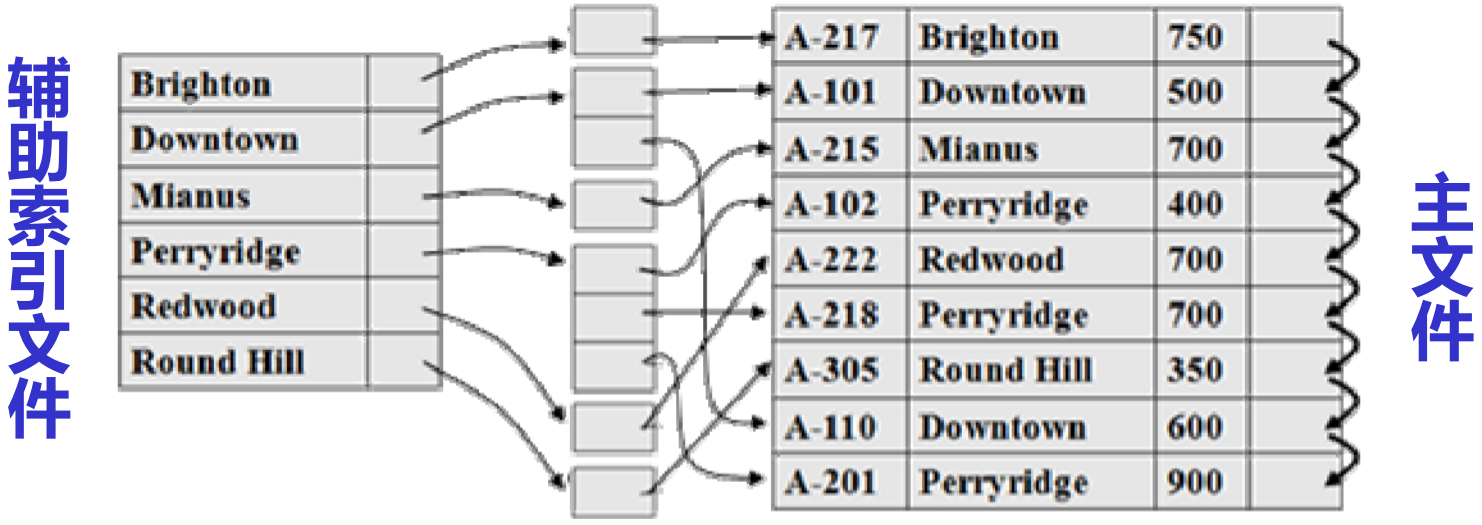
辅助索引

辅助索引是定义在主文件的任一或多个非排序字段上的辅助存储结构。

●辅助索引通常是对某一非排序字段上的每一个不同值有一个索引项：索引字段即是该字段的不同值，而指针则指向包含该记录的块或该记录本身；

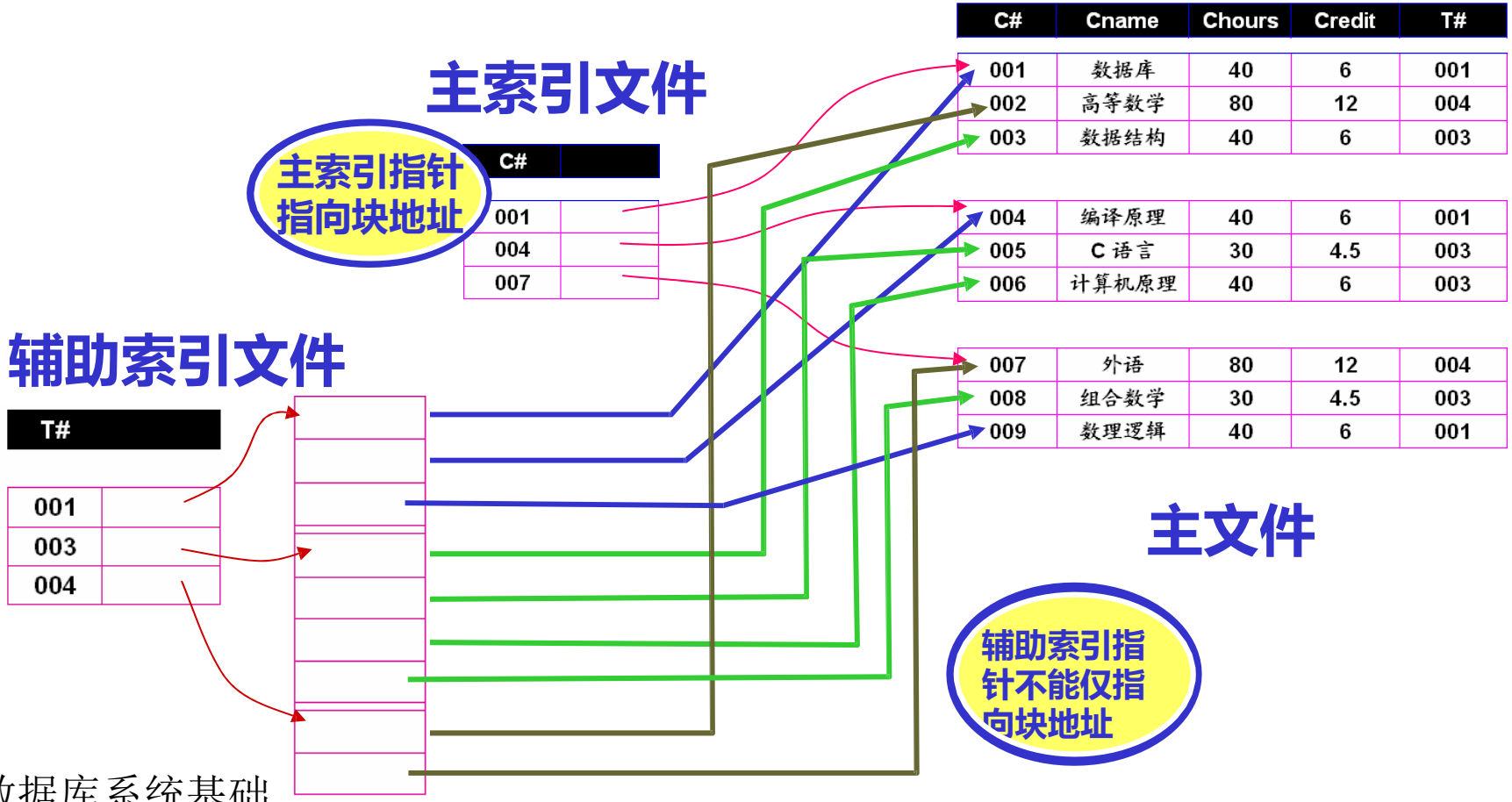
●当非排序字段为索引字段时，如该字段值不唯一，则要采用一个类似链表 的结构来保存包含该字段值的所有记录的位置。

●辅助索引是稠密索引，其检索效率有时相当高。



主索引 vs. 辅助索引

- 一个主文件仅可以有一个主索引，但可以有多个辅助索引
- 主索引通常建立于主码/排序码上面；辅助索引建立于其他属性上面
- 可以利用主索引重新组织主文件数据，但辅助索引不能改变主文件数据
- 主索引是稀疏索引，辅助索引是稠密索引



其他类型的索引

聚簇索引—是指索引中邻近的记录在主文件中也是临近存储的；

非聚簇索引—是指索引中邻近的记录在主文件中不一定是邻近存储的。

T#是“非主码字段”，有重复值，排序存储

聚簇索引文件

T#	
001	
003	
004	

聚簇字段值　　块指针

主文件

C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
004	编译原理	40	6	001
009	数理逻辑	40	6	001
008	组合数学	30	4.5	003
003	数据结构	40	6	003
005	C 语言	30	4.5	003
006	计算机原理	40	6	003
002	高等数学	80	12	004
007	外语	80	12	004

主文件基于“非主码的字段”T#排序存储

非聚簇索引文件

T#是“非主码字段”，有重复值，排序存储

T#	
001	
001	
001	
003	
003	
003	
003	
004	
004	

索引字段值　　记录指针

主文件

C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
002	高等数学	80	12	004
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
006	计算机原理	40	6	003
007	外语	80	12	004
008	组合数学	30	4.5	003
009	数理逻辑	40	6	001

主文件没有基于“非主码的字段”T#排序存储

聚簇索引和非聚簇索引

聚簇索引—是指索引中邻近的记录，在主文件中也是临近存储的；

非聚簇索引—是指索引中邻近的记录，在主文件中不是邻近存储的。

❑如果主文件的某一**排序**字段**不是主码**，则该字段上每个记录取值便不唯一，此时该字段被称为**聚簇字段**；**聚簇索引通常是定义在聚簇字段上。**

❑聚簇索引通常是对聚簇字段上的每一个不同值有一个索引项（索引项的总数和主文件中聚簇字段上不同值的数目相同），索引字段即是聚簇字段的不同值，由于有相同聚簇字段值的记录可能存储于若干块中，则索引项的指针指向其中的第一个块。

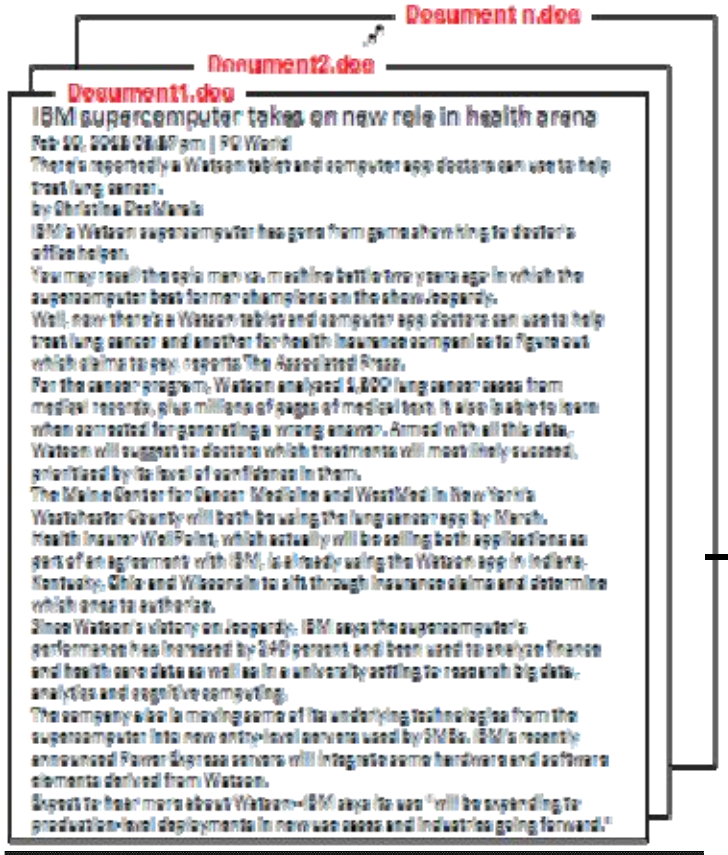
❑一个主文件只能有一个聚簇索引文件，但可以有多个非聚簇索引文件

❑主索引通常是聚簇索引(但其索引项总数不一定和主文件中聚簇字段上不同值的数目相 同，其和主文件存储块数目相同)；辅助索引通常是非聚簇索引。

❑主索引/聚簇索引是能够决定记录存储位置的索引；而非聚簇索引则只能用于查询，指出已存储记录的位置。

倒排索引

正排：一个文档包含了哪些词汇?
#Doc1, { Word1,Word2,... }
倒排：一个词汇包含在哪些文档中
Word1, { #Doc1, #Doc2, ... }



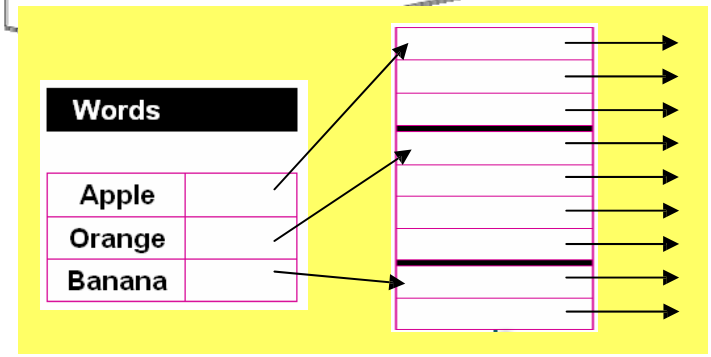
查找文档

对所有文档建立

关键词查询

关键词索引表---倒排索引

关键词, (#所在文档，出现次数，<出现位置...>)
health, (#Document1.doc, 1次， <8>),
(#Document3.doc, 3次， <10,62, 182>)
IBM, (#Document1.doc, 2次, <1,10>),
(#Document2.doc, 5次, <1,10,100,240,500>)
Supercomputer,(#Document1.doc, 3次, <2, 12,38>
Watson, (#Document1.doc, 1次， <11>),
(#Document4.doc, 3次， <15,81, 202>)



怎样按照关键词找到相应的文档呢?

其他结构索引

其他结构的索引

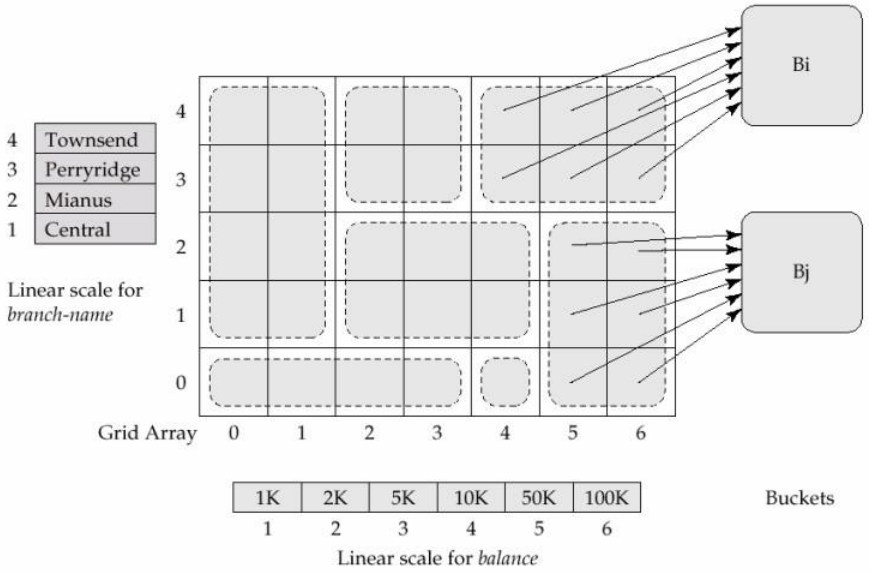
❑ **多级索引**：当索引项比较多时，可以对索引再建立索引，依此类推，形成多级索引

✓ 常见的多级索引形式，如**B树/B+树索引**，以树型数据结构来组织索引项等

● **多属性索引**：索引字段由Table的多个属性值组合在一起形成的索引

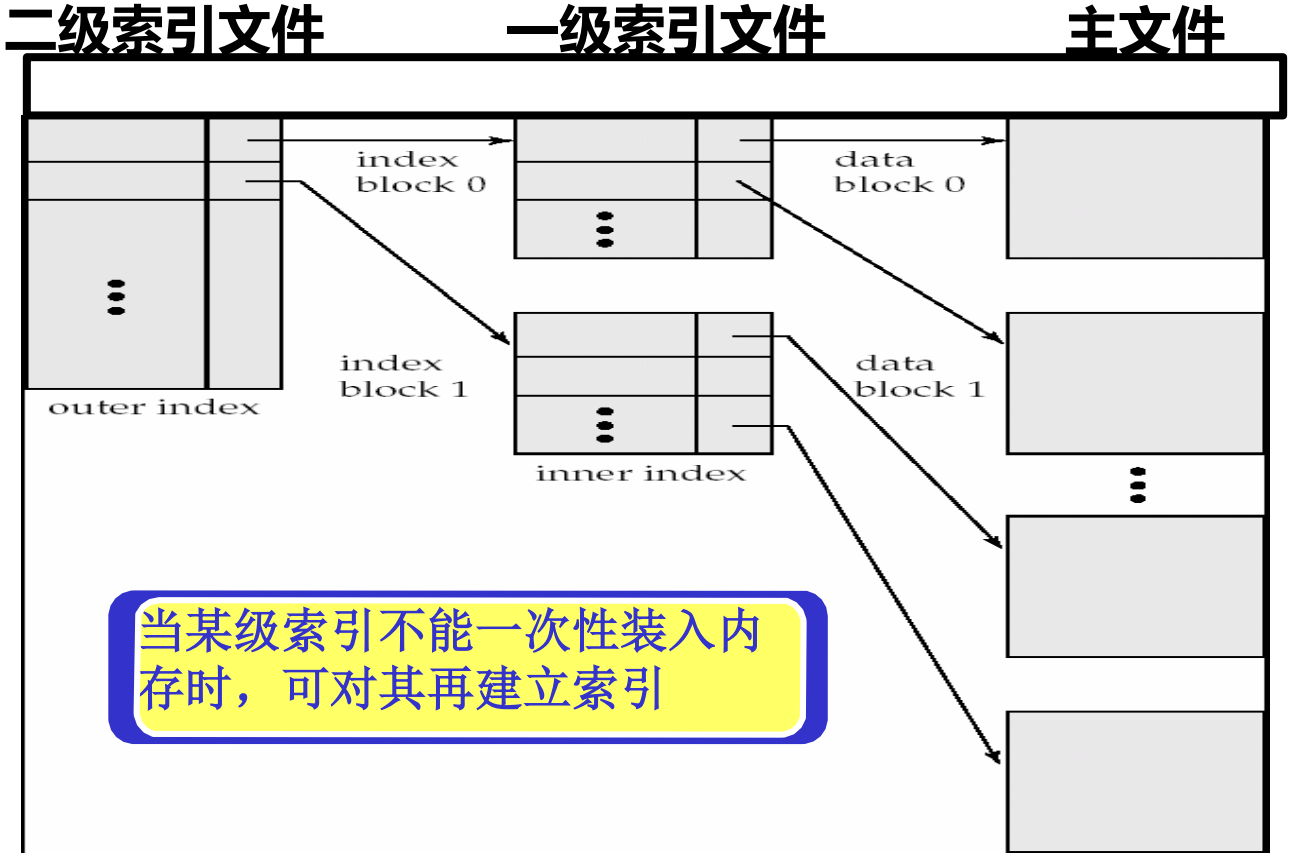
● **散列索引**：使用散列技术组织索引

● **网格索引**(Grid file)：使用多索引字段进行交叉联合定位与检索



B+树(B+ Tree)

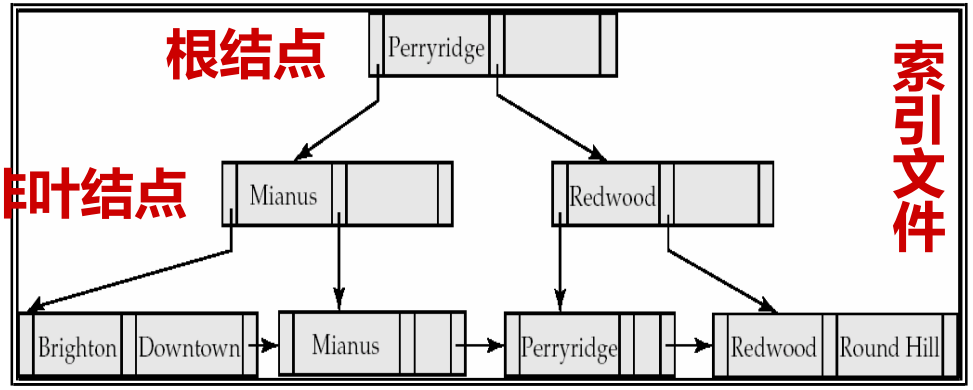
多级索引：当索引项比较多时，可以对索引再建立索引，依此类推，形成多级索引。



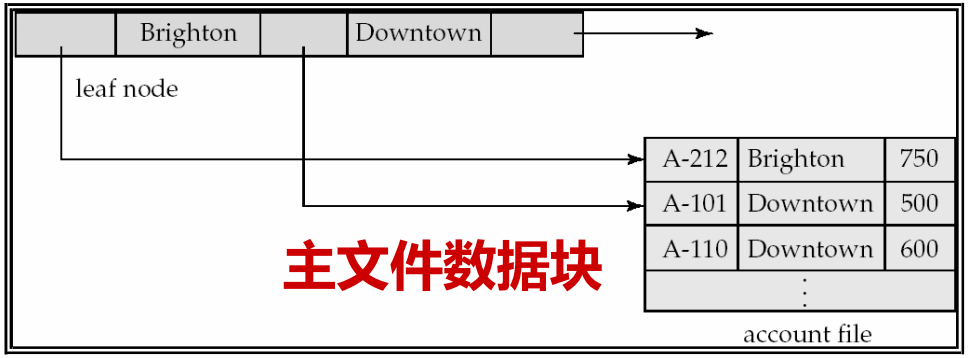
B+树索引：一种以树型数据结构来组织索引项的多级索引

一块中索引项的组织

- Ki—索引字段值
- Pj—指针，指向索引块或数据块或**数据块中记录的指针**



索引文件的叶子结点



- **能够自动保持与主文件大小相适应的树的层次**
- **每个索引块的指针利用率都在50%-100%之间**

B+树(B+ Tree)

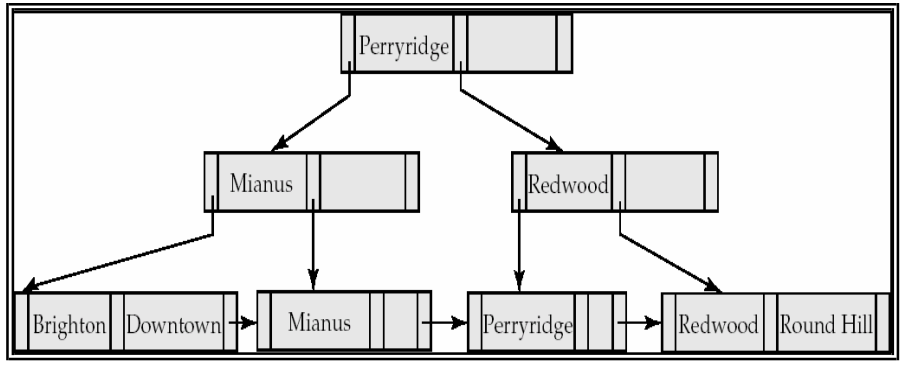
(3)B+树的存储约定

一块中存放多少个索引项(n的大小)?

P_1	K_1	P_2	...	P_{n-1}	K_{n-1}	P_n

- 有n-1个索引项(<索引字段值K_i, 指针P_i>)+1个指针(P_n);
- 索引字段值x在$K_{i-1} \le x < K_i$ 的由P_i指向；而$K_i \le x < K_{i+1}$的由P_{i+1}指向。

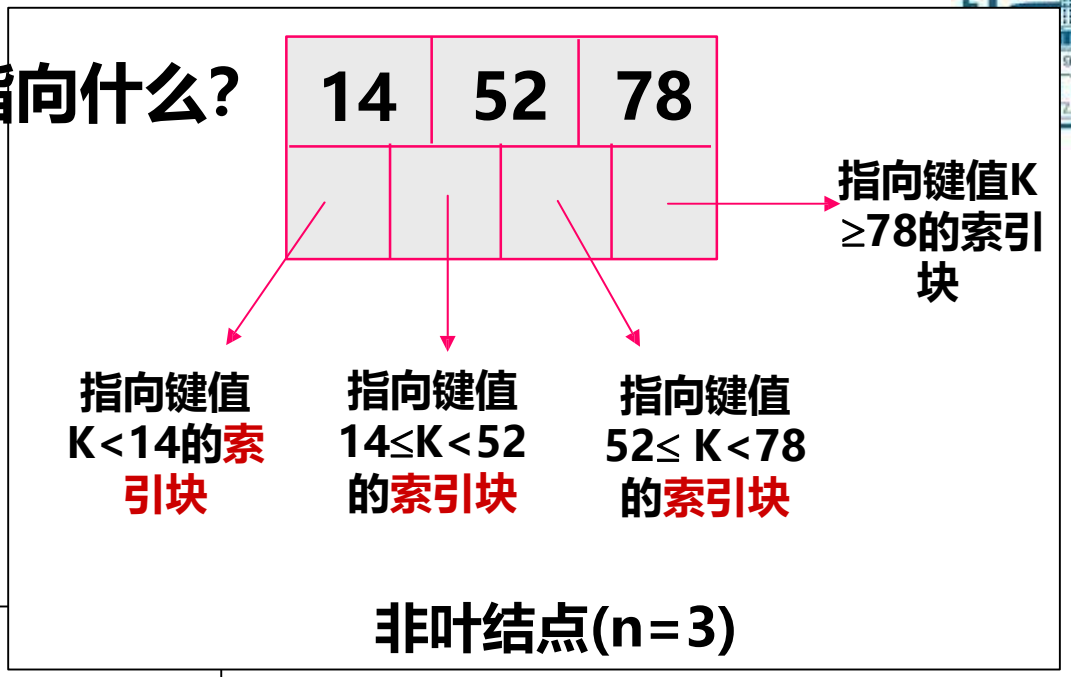
示例 存储块 = 4096 Byte
整数型索引字段值= 4 Byte
指针 = 8 Byte
则：n应满足 **4(n-1)+8n<=4096**
n取最大值
n=341

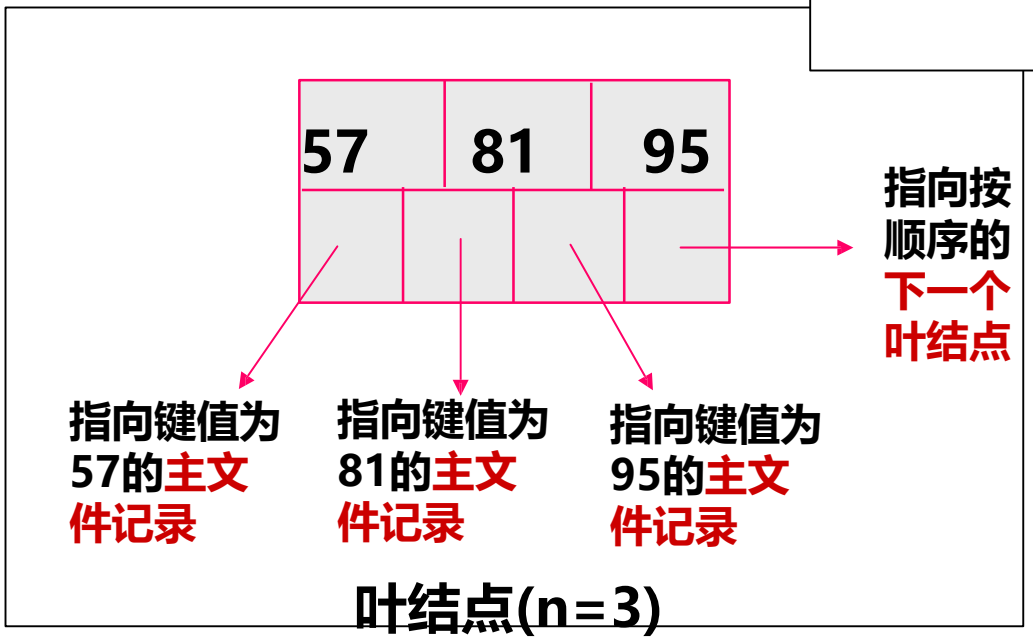


叶结点和非叶节点的指针指向什么?

•非叶结点指针指向索引块，叶结点指针指向主文件的数据块或数据记录

叶结点的最后一个指针始终指向其下一个数据块



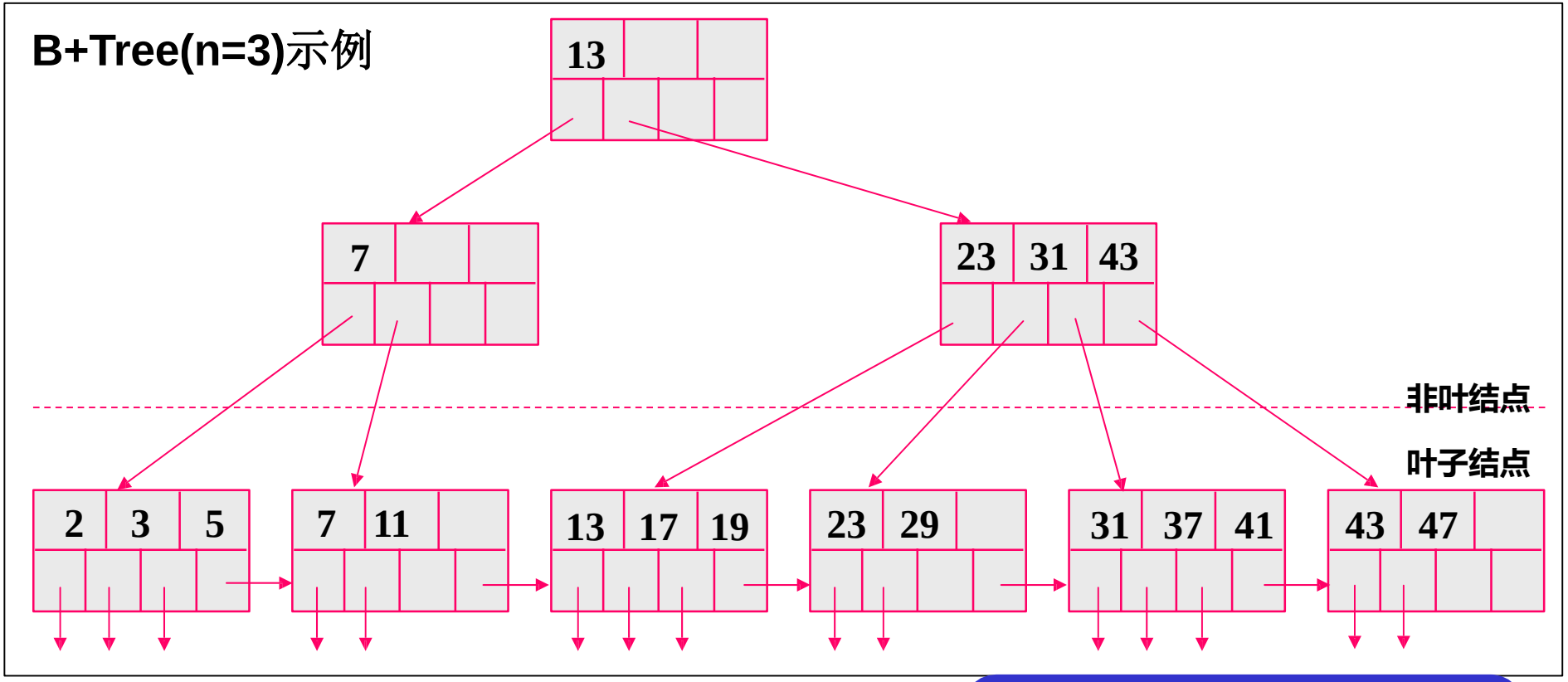


•一索引块实际使用的索引指针个数d，满足(根结点除外)：

$n/2 \le d \le n$。

•根结点至少2个指针被使用。

•索引字段值重复出现于叶结点和非叶结点；



•指向主文件的指针仅出现于叶结点；

•所有叶结点即可覆盖所有键值的索引；

•索引字段值在叶结点中是按顺序排列的；

仅叶结点块的集合就是主文件完整的索引，即：最低一级(或一层)索引

B+树(B+ Tree)

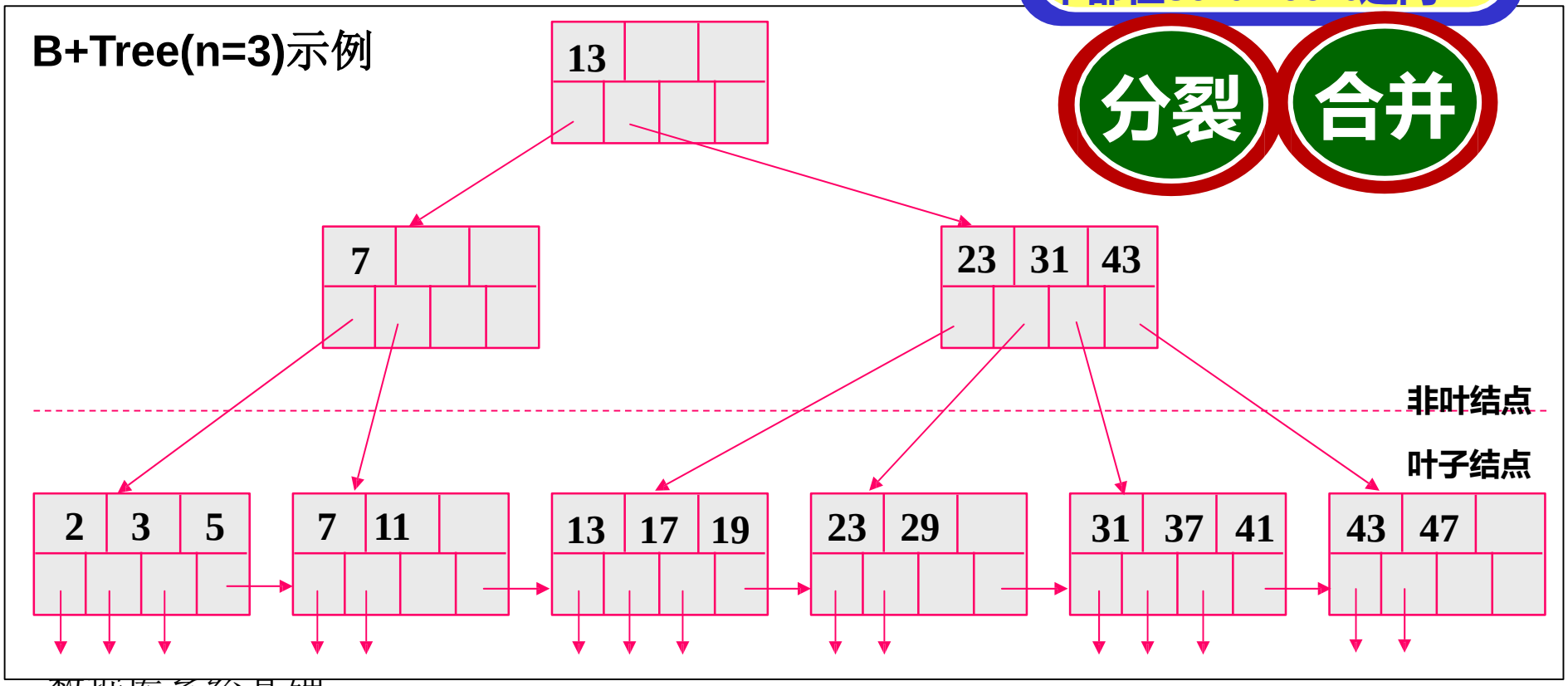
(3)B+树的存储约定

- 级数(或层数)相同--平衡。**如何保证?**
- 插入/删除记录时，伴随着结点的分裂与合并;
- 分裂与合并将调整部分结点块中的索引项

(需有算法支持)

- 能够自动保持与主文件大小相适应的树的层次
- 每个索引块的指针利用率都在50%-100%之间

分裂 合并

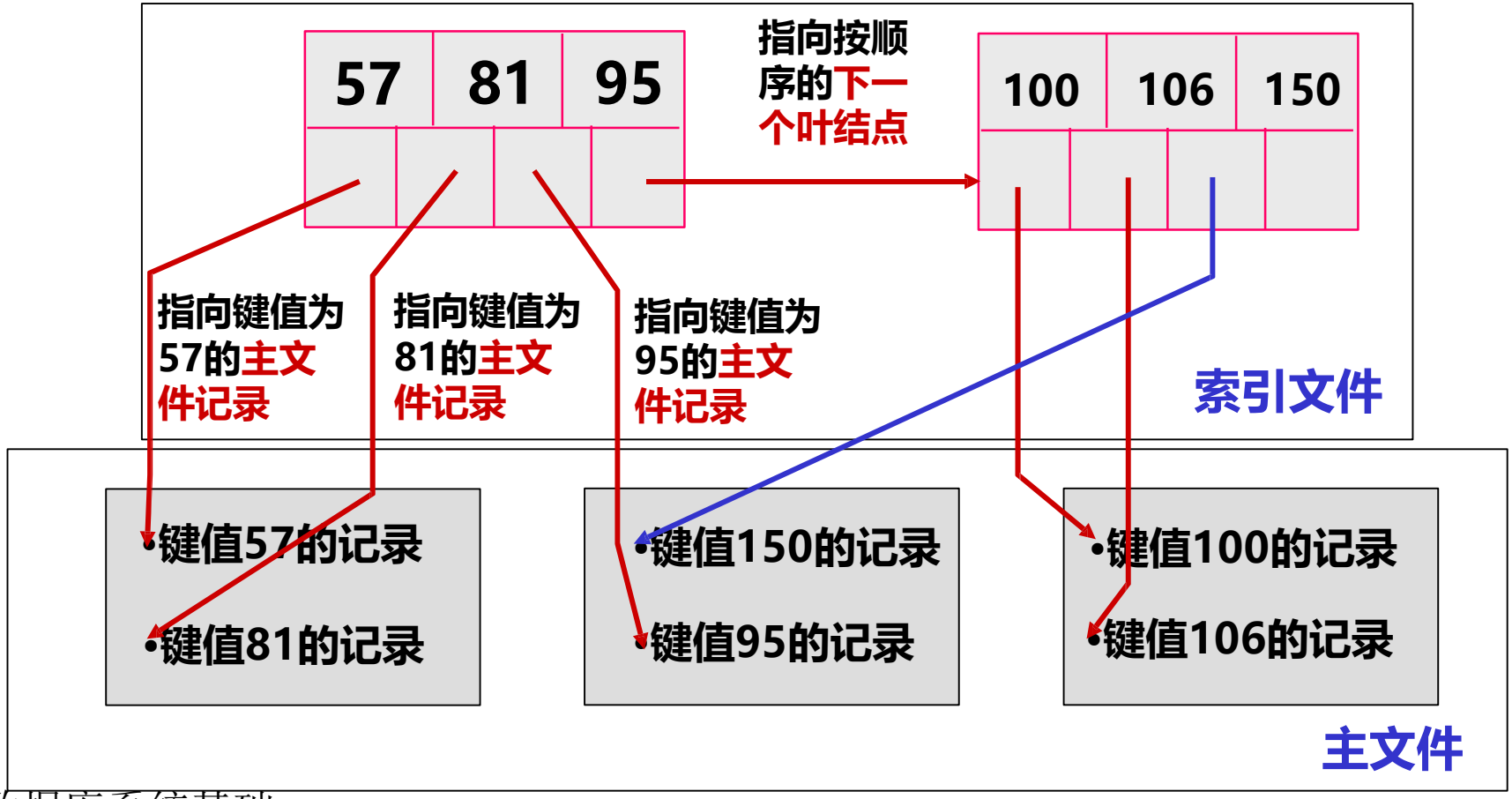


B+树(B+ Tree) (4)用B+树

建立不同的索引

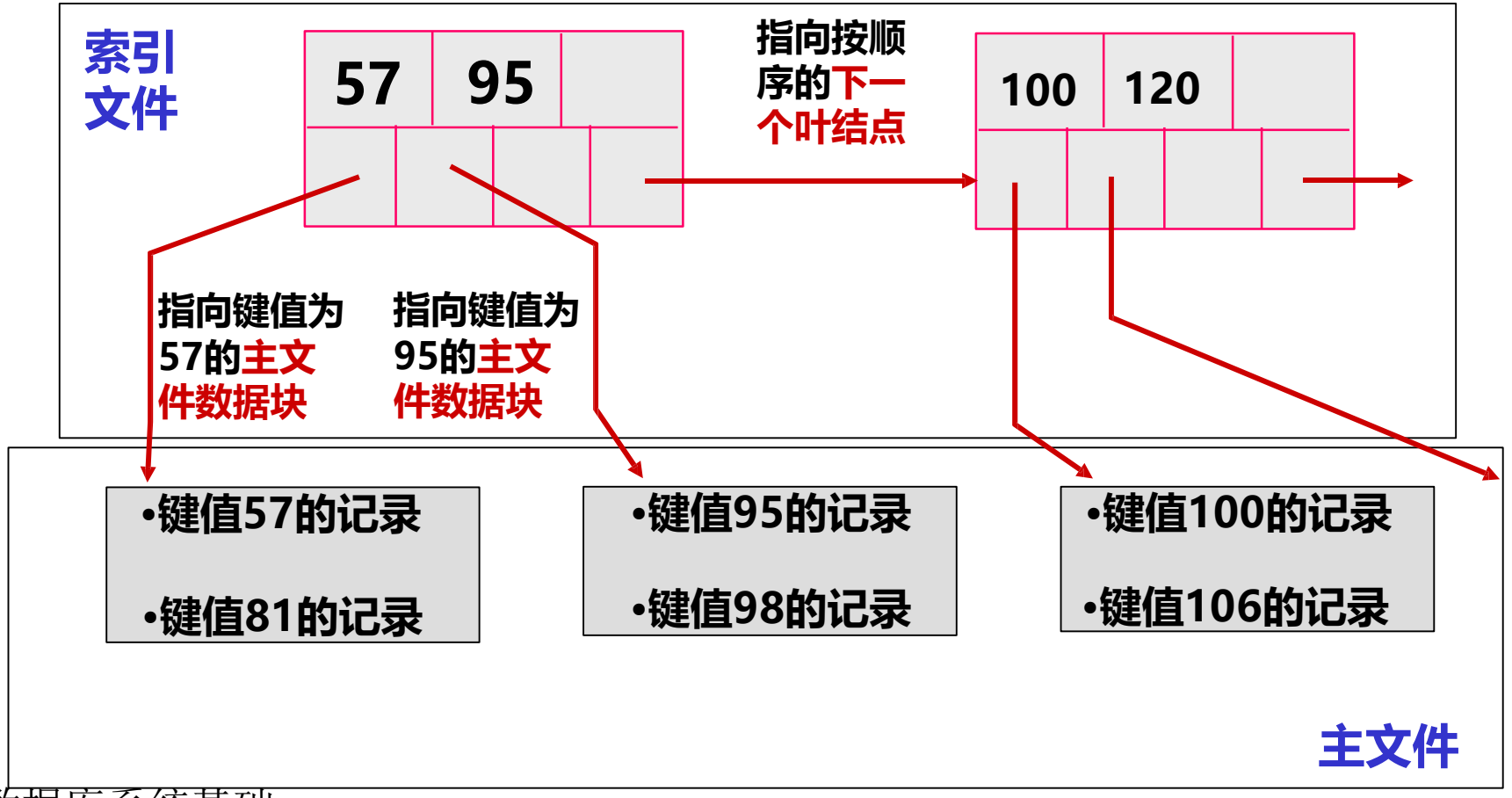
I. 用B+树建立键属性稠密索引

•索引字段是主文件的主键，索引是稠密的。主文件可以按主键排序，也可以不按主键排序。指针指向的是记录。



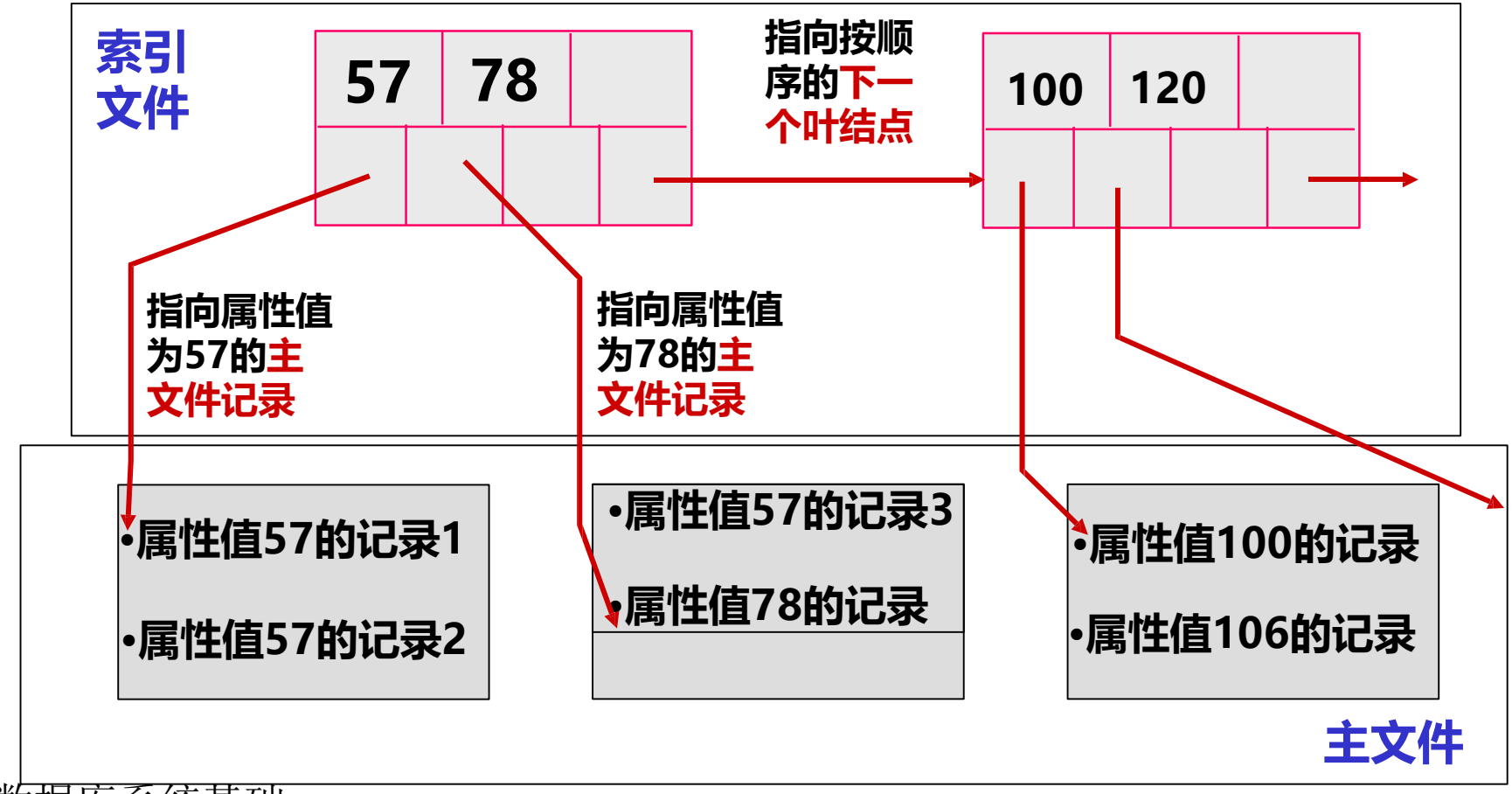
II. 用B+树建立稀疏索引(或主索引)

- 索引字段是主文件的主键，索引是稀疏的。主文件必须按主键排序。
- 指针指向的是数据块



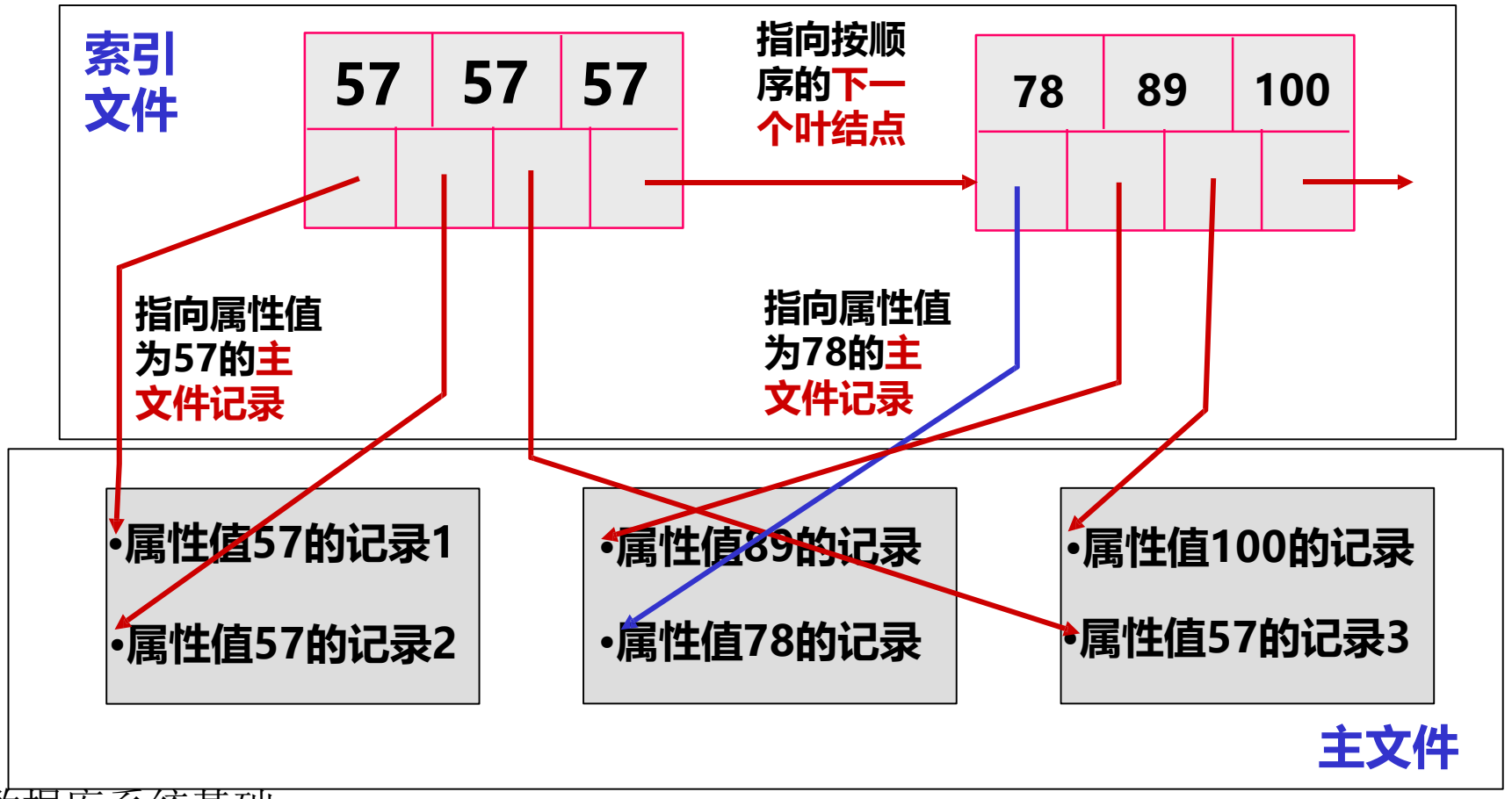
III. 用B+树建立非键属性稠密索引

- 索引字段是主文件的非键属性，索引是稠密的。主文件按非键属性排序
- 索引文件的索引字段是无重复的。指针指向的是记录。



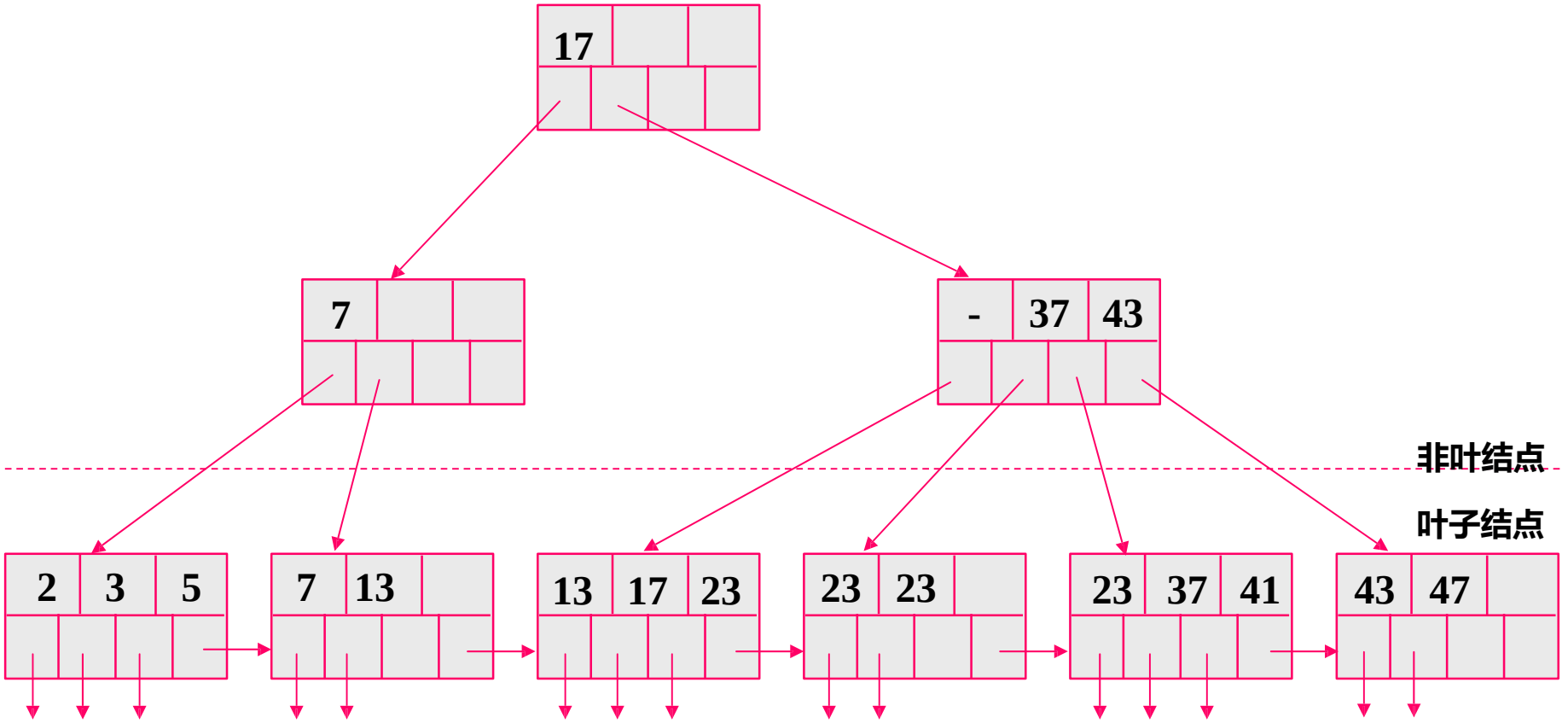
IV. 用B+树建立非键属性稠密索引

- 索引字段是主文件的非键属性。主文件不按此非键属性排序
- 索引文件的索引字段值是有重复的。指针指向的是记录。



IV. 用B+树建立非键属性稠密索引

•索引字段带有重复值的B+树的非叶结点示例



一棵带重复键值的B+Tree

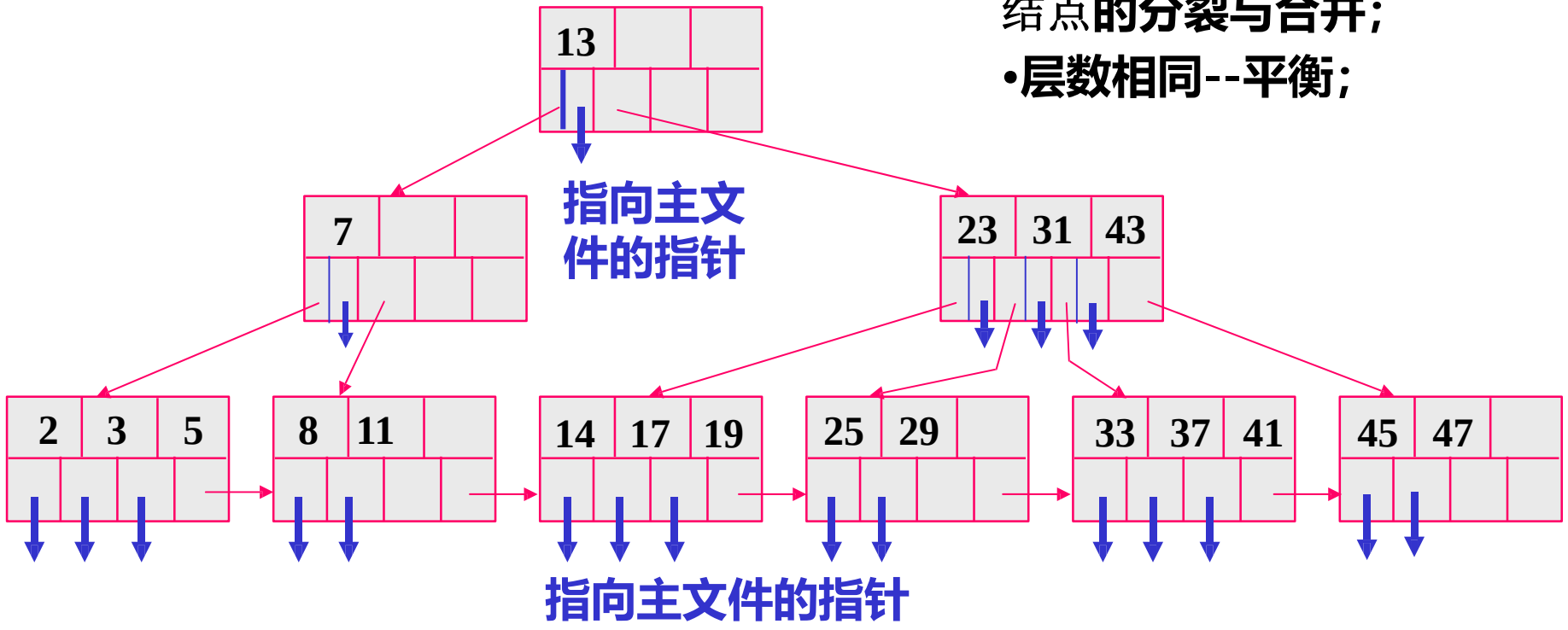
B+树(B+ Tree)

(5)B+树 vs. B树

B Tree

- 索引字段值仅出现一次或者在叶结点或者在非叶结点；
- 指向主文件的指针出现于叶结点或非叶结点；
- 所有结点才能覆盖所有键值的索引。

- 插入记录时，伴随着结点**的分裂与合并**；
- **层数相同--平衡**；

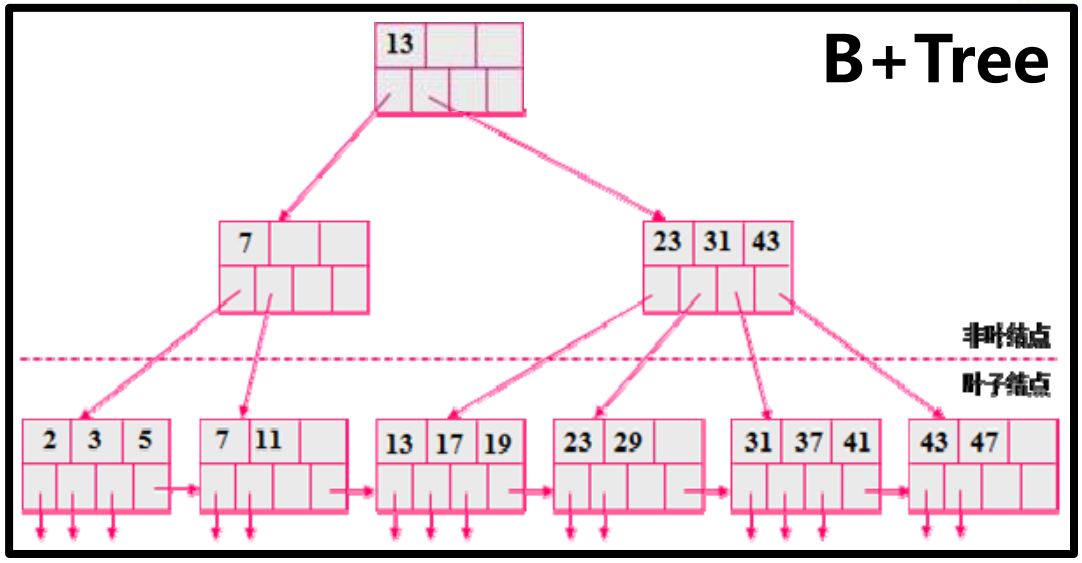


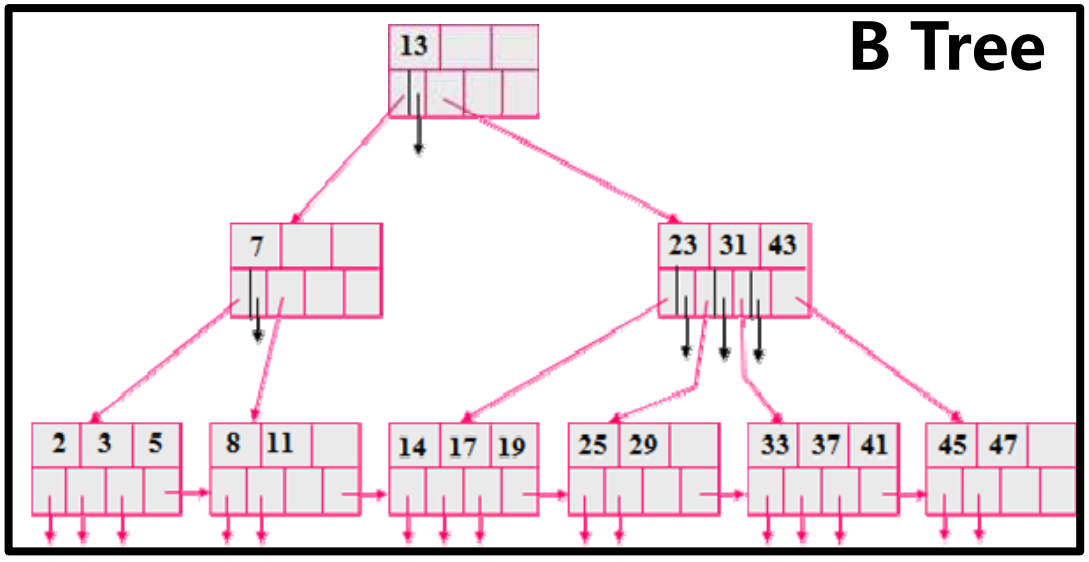
B+树(B+ Tree)

(5)B+树 vs. B树

B Tree vs. B+ Tree

- 一块中存放的索引项个数是否相同?
- 索引字段值都出现在哪里?
- 指向主文件的指针存在哪里?
- 分裂与合并的方法是否一致?





算法可参见教材

利用B+树检索的算法

```
function find( value V )
  /* 假设没有重复码返回叶结点 C 和索引 i 使得 C.P_i 指向第一条搜索码…
    置 C = 根结点
    while C 不是叶结点 begin
      令 i = 满足 V≤C.K_i 的最小值
      if 没有这样的 i then begin
        令 P_m = 结点中最后一个非空指针
        置 C = C.P_m
      end
        else if (V = C.K_i)
          then 置 C = C.P_{i+1}
        else C = C.P_i /* V < C.K_i */
      end
      /* C 是叶结点 */
      设 i 是满足 K_i = V 的最…
      if 有这样的 i 存在
        then 返回(C, i)
        else 返回空 /* 没有码值等于 V 的记录存在 */

procedure printAll(value V)
  /* 打印所有搜索码值等于 V 的记录 */
    置 done = false
    置(L, i) = findFirst(V)
    if((L, i)为空)return
    repeat
      repeat
        打印指针 L.P_i 指向的记录
        置 i = i+1
      until(i > L 中码的数目或 L.K_i > V)
      if(i > L 中码的数目)
        then L = L.P_n
        else 置 done = true;
    until (done or L 为空)
```

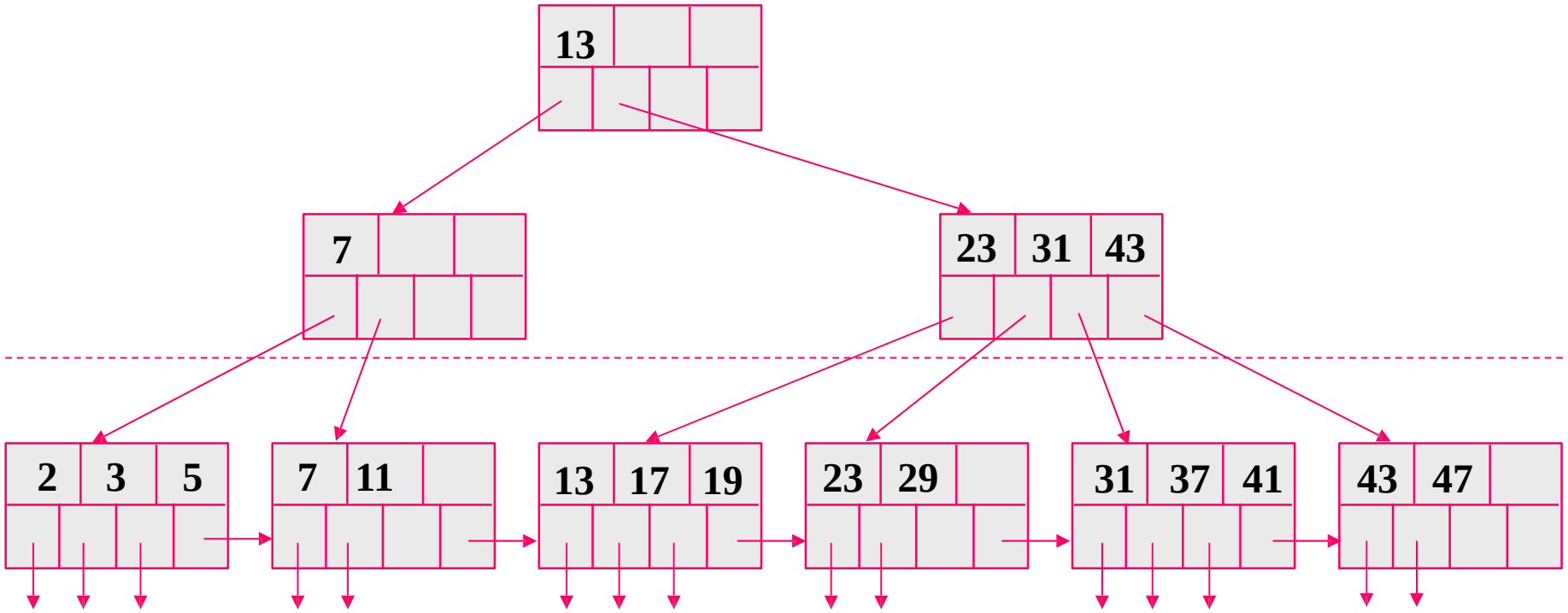
利用B+树增加记录的算法

利用B+树删除记录的算法

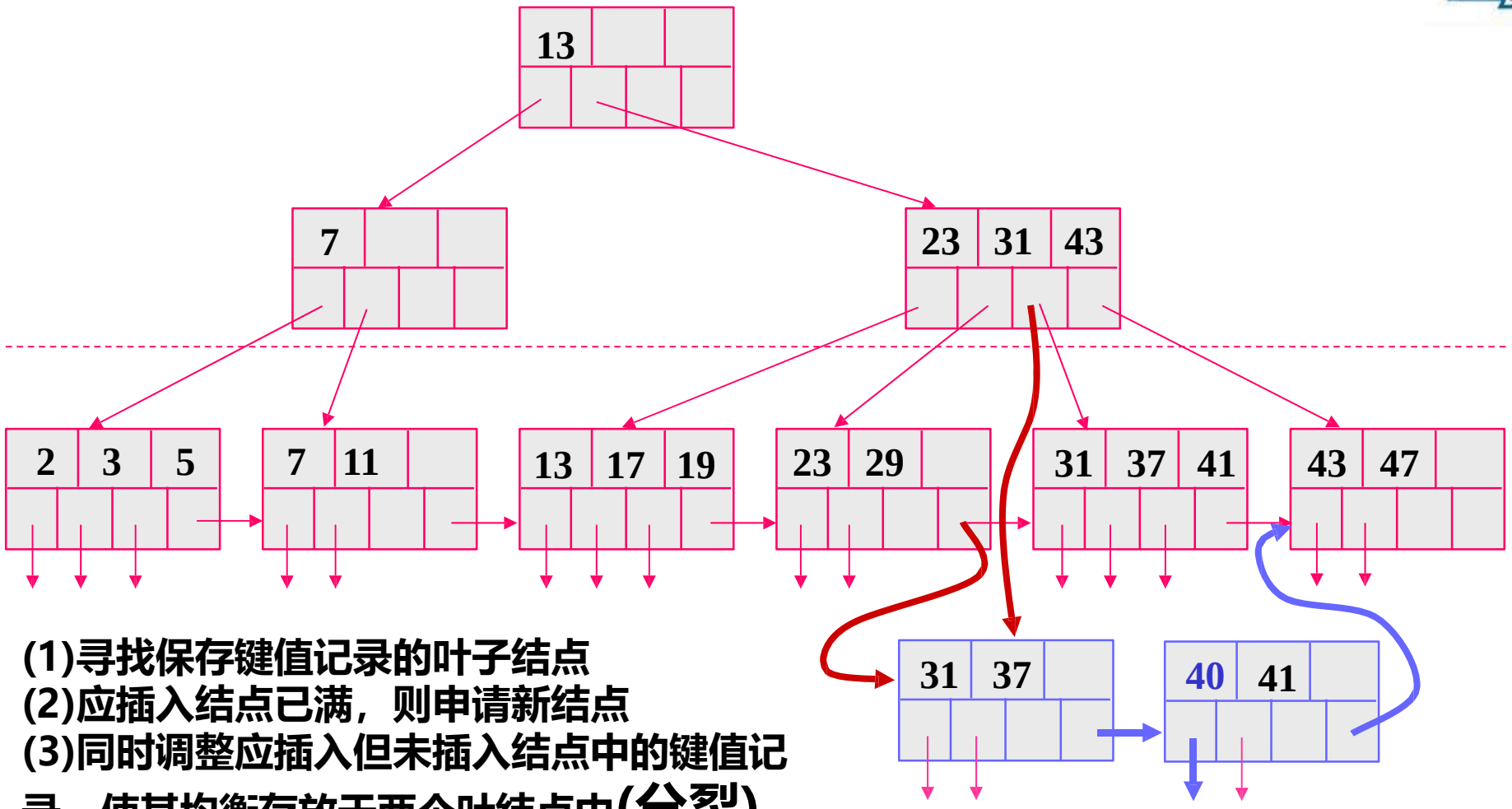
如何利用B+树进行检索？ 如何利用B+树进行增加和删除操作？ B+树的增加和删除操作时如何进行分裂和合并？ 什么条件下分裂与合并？ 分裂与合并时如何调整B+树的指针？

B+树之键值插入与结点分裂过程示意

问题1：在下列B+树中插入键值为40的记录

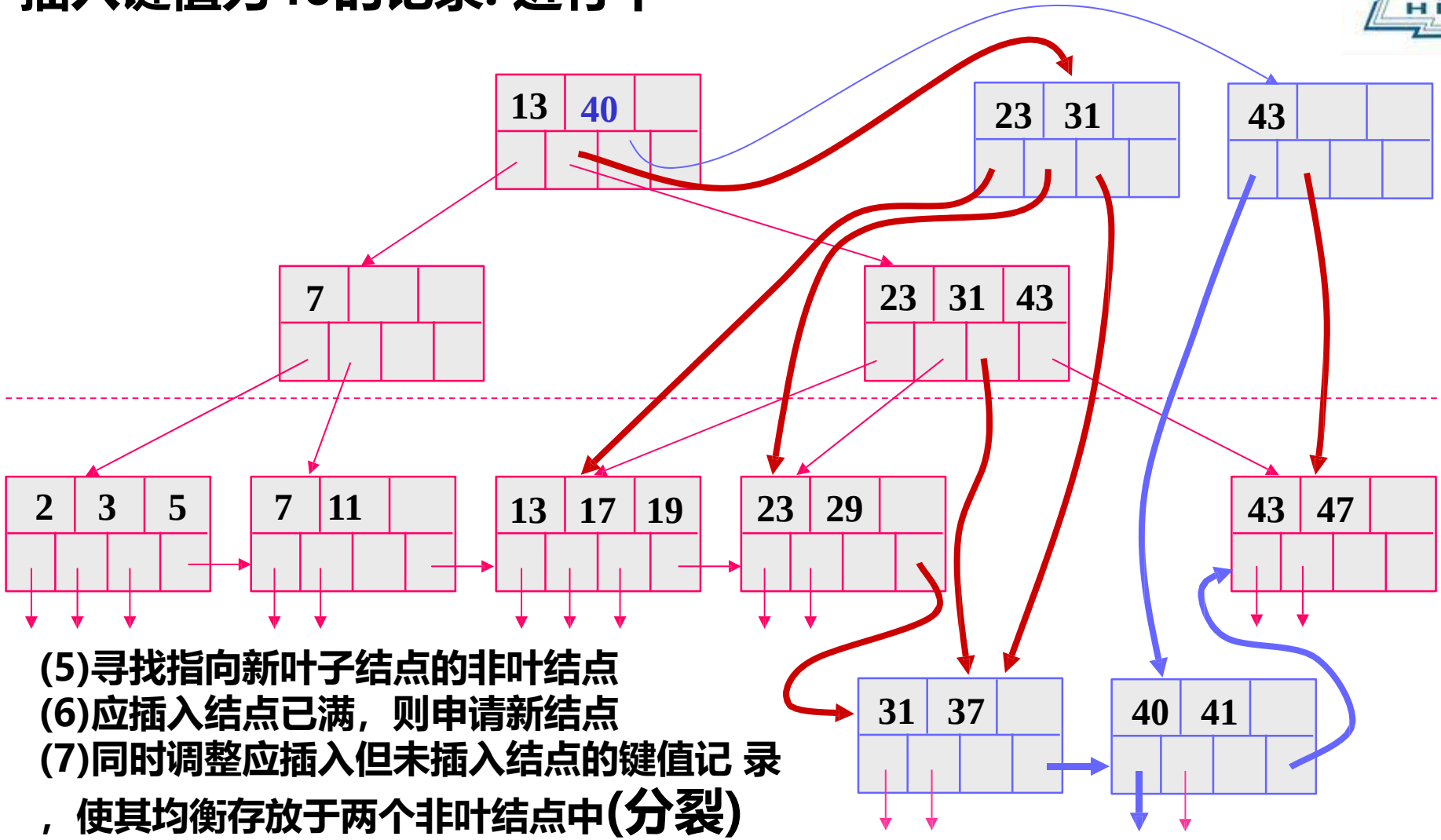


插入键值为40的记录: 进行中



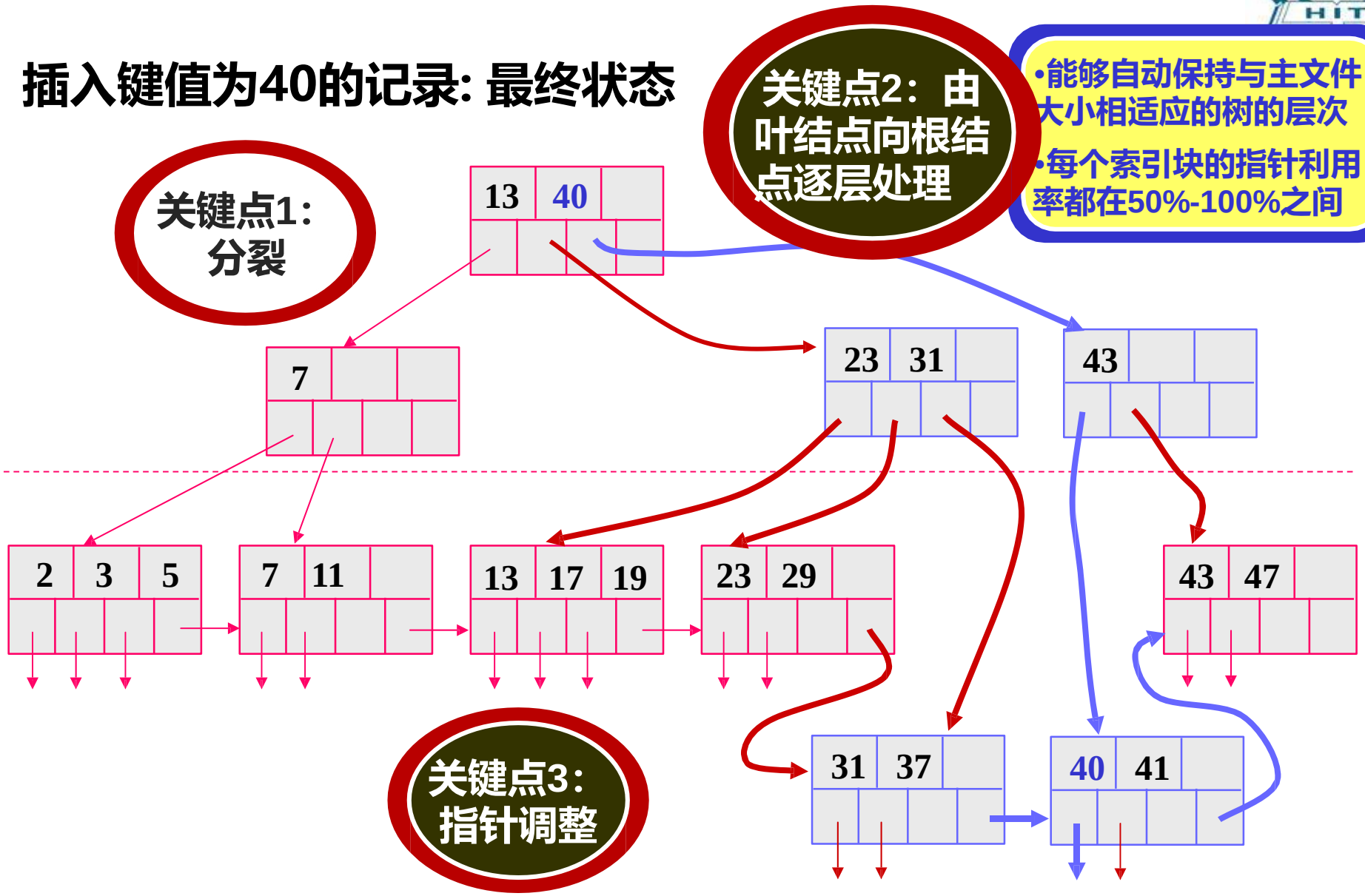
(1)寻找保存键值记录的叶子结点
(2)应插入结点已满，则申请新结点
(3)同时调整应插入但未插入结点中的键值记录，使其均衡存放于两个叶结点中(分裂)
(4)调整指针使其指向新叶子结点

插入键值为40的记录: 进行中



(5)寻找指向新叶子结点的非叶结点
(6)应插入结点已满，则申请新结点
(7)同时调整应插入但未插入结点的键值记 录，使其均衡存放于两个非叶结点中(分裂)
(8)调整各结点指针使其指向正确

插入键值为40的记录: 最终状态



将算法表达出来

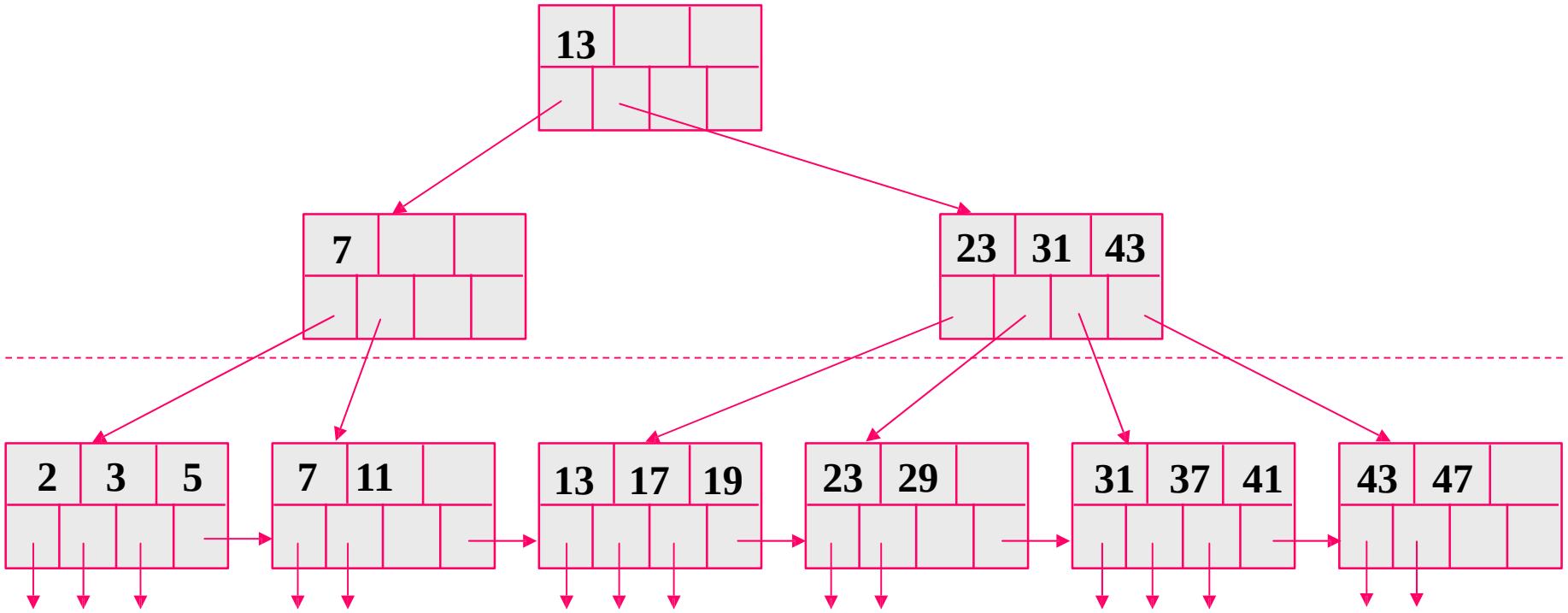
```
procedure insert(value K, pointer P)
    if (树为空) 创建一个空叶结点 L, 同时它也是根结点
    else 找到应该包含值 K 的叶结点 L
    if (L 所含搜索码少于 n-1 个)
        then insert_in_leaf (L, K, P)
        else begin  /* L 已经含有 n-1 个搜索码了, 分裂 L */
            创建结点 L'
            把 L.P1, …, L.K(n-1) 复制到可以存储 n 个(指针, 搜索码值)对的内存块 T 中
            insert_in_leaf (T, K, P)
            令 L'.Pn = L.Pn; 令 L.Pn = L'
            从 L 中删除 L.P1, …, L.P(n-1), L.K(n-1), L.Pn
            把 T.P1, …, T.K(⌈n/2⌉) 从 T 复制到 L 中, L 以 L.P1 作为开始
            把 T.K(⌈n/2⌉), …, T.Kn 从 T 复制到 L'中, L'以 L'.P1 作为开始
            令 K'表示 L'中最小搜索码值
            insert_in_parent (L, K', L')
        end

procedure insert_in_leaf (node L, value K, pointer P)
    if (K 比 L.K1 小)
        then 把 P、K 插入 L 中, 紧接在 L.P1 前面
        else begin
            令 Ki 表示 L 中小于等于 K 的最大搜索码值
            把 P、K 插入 L 中, 紧跟在 L.Ki 前面
        end
procedure insert_in_parent (node N, value K', pointer N')
    if (N 是树的根结点)
        then 在 L 中插入(V, P)
        else begin
            创建结点 R 包含 N、K'、N'  /* N 和 N'都是指针 */
            令 R 成为树的根结点
            return
        end
令 P=N 的父结点
if (P 包含的指针少于 n 个)
    then 将(K', N')插到 N 后面
    else begin /* 分裂 P */
            将 P 复制到可以存放 P 以及(K', N')的内存块 T 中
            创建结点(K', N')插入 T 中, 紧跟在 N 后面
            删除所有 P 中所有项; 创建结点 P'
            把 T.P1, …, T.K(⌈n/2⌉) 复制到 P 中
令 K"= T.K(⌈n/2⌉)
把 T.P(⌈n/2⌉+1), …, T.Pn+1 复制到 P'中
insert_in_parent (P, K", P')
    end
```

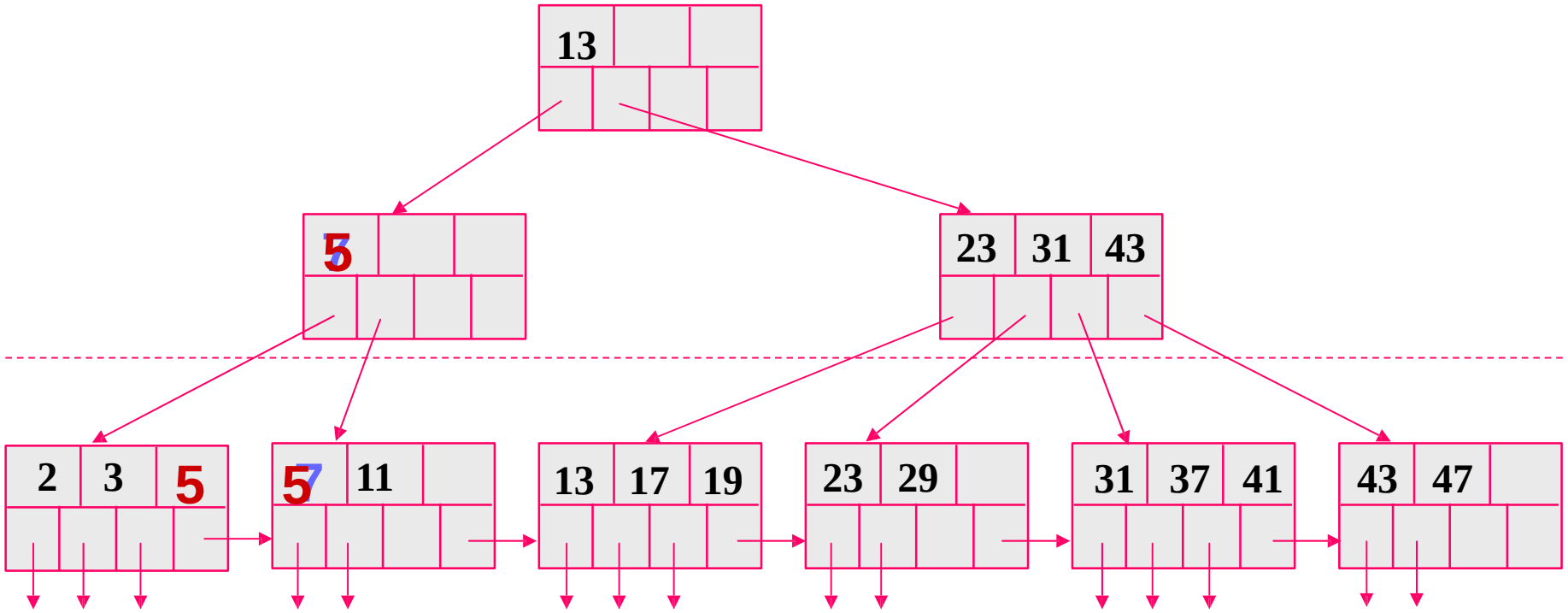
此算法可参见教材

B+树之键值删除与结点合并过程示意

问题1：在下列B+树中删除键值为7的记录

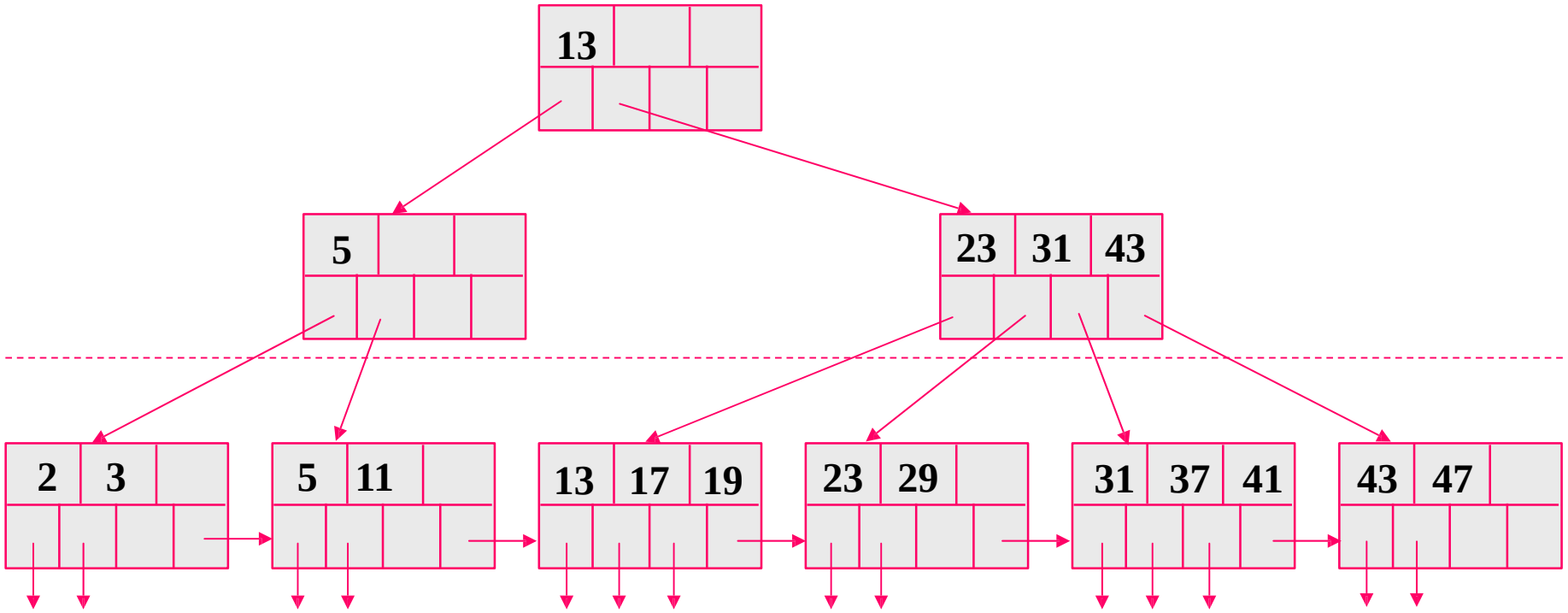


删除键值为7的记录: 进行中



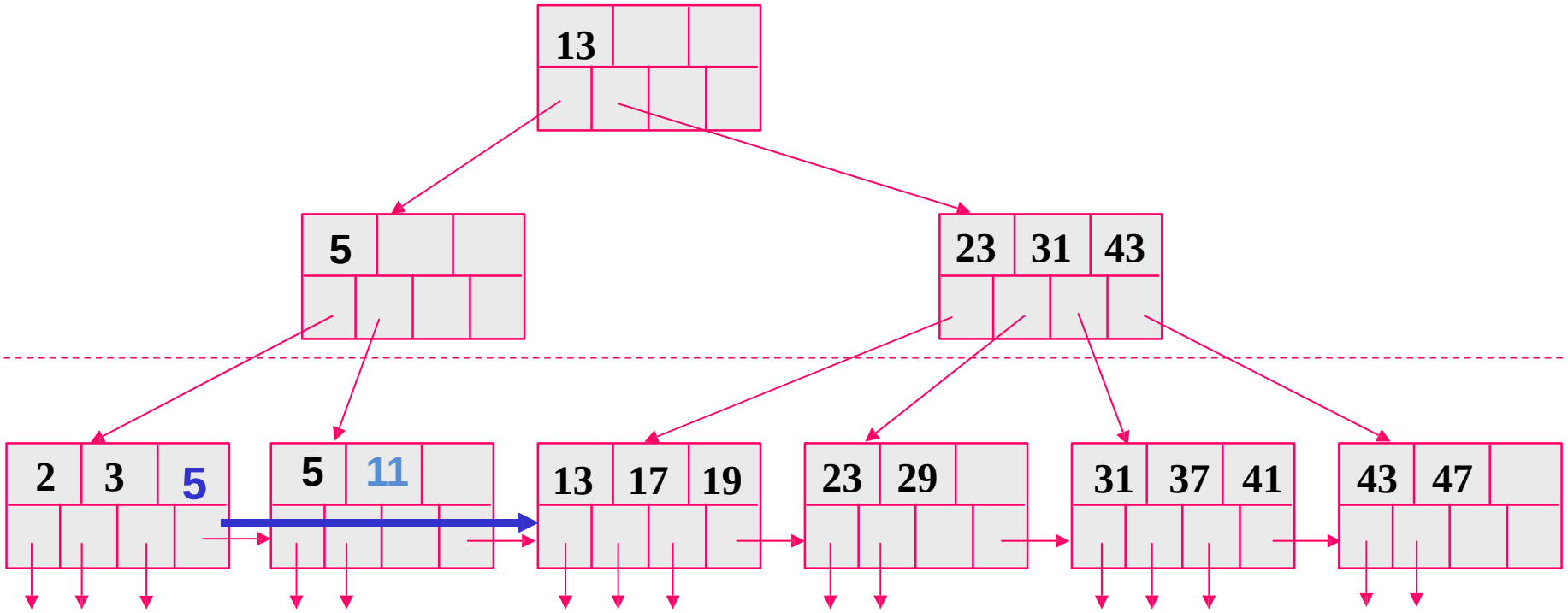
(1)叶子结点中寻找等于键值的记录，删除相应的指针及主文件中对应的记录
(2)调整其左侧(或右侧)结点及本结点中的键值记录，使其均衡存放于两个叶结点中
(3)如有调整，则进一步调整其上层非叶结点，重新确定其键值，以满足大于等于键值的记录都在其右侧

删除键值为7的记录: 最终状态



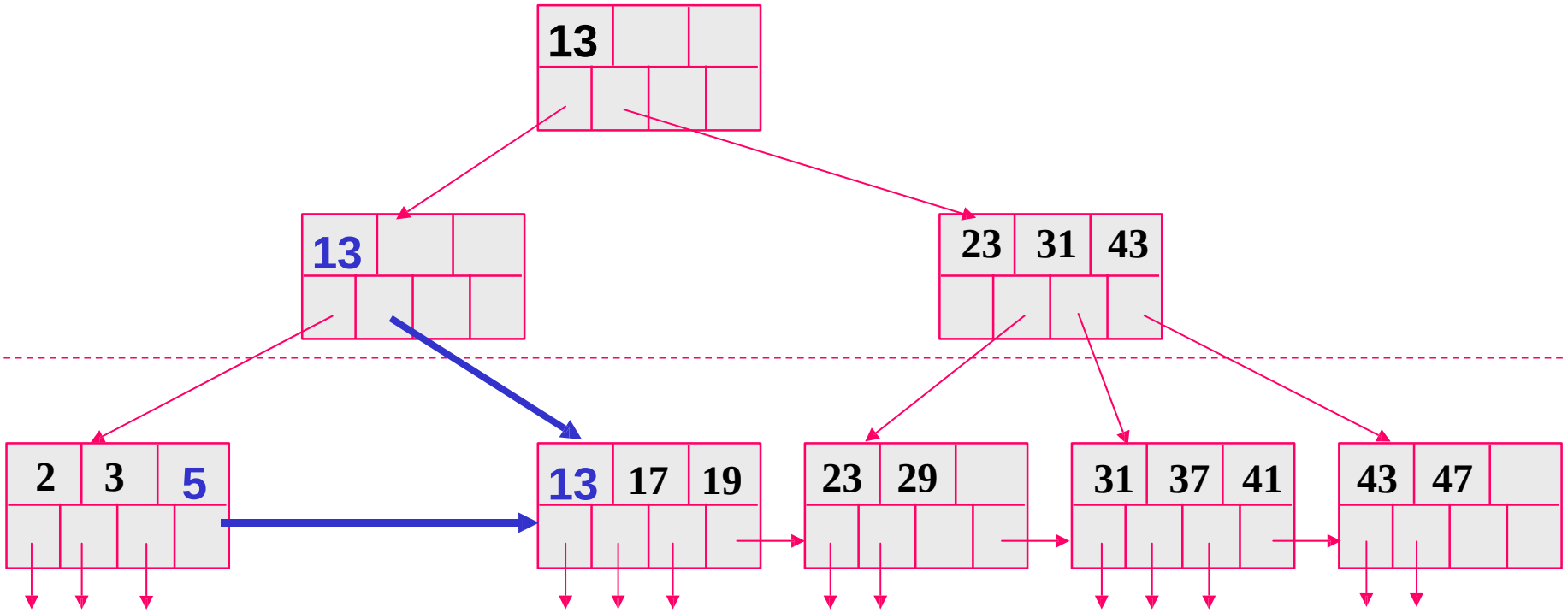
问题2：在此B+树中删除键值为11的记录

删除键值为11的记录: 进行中



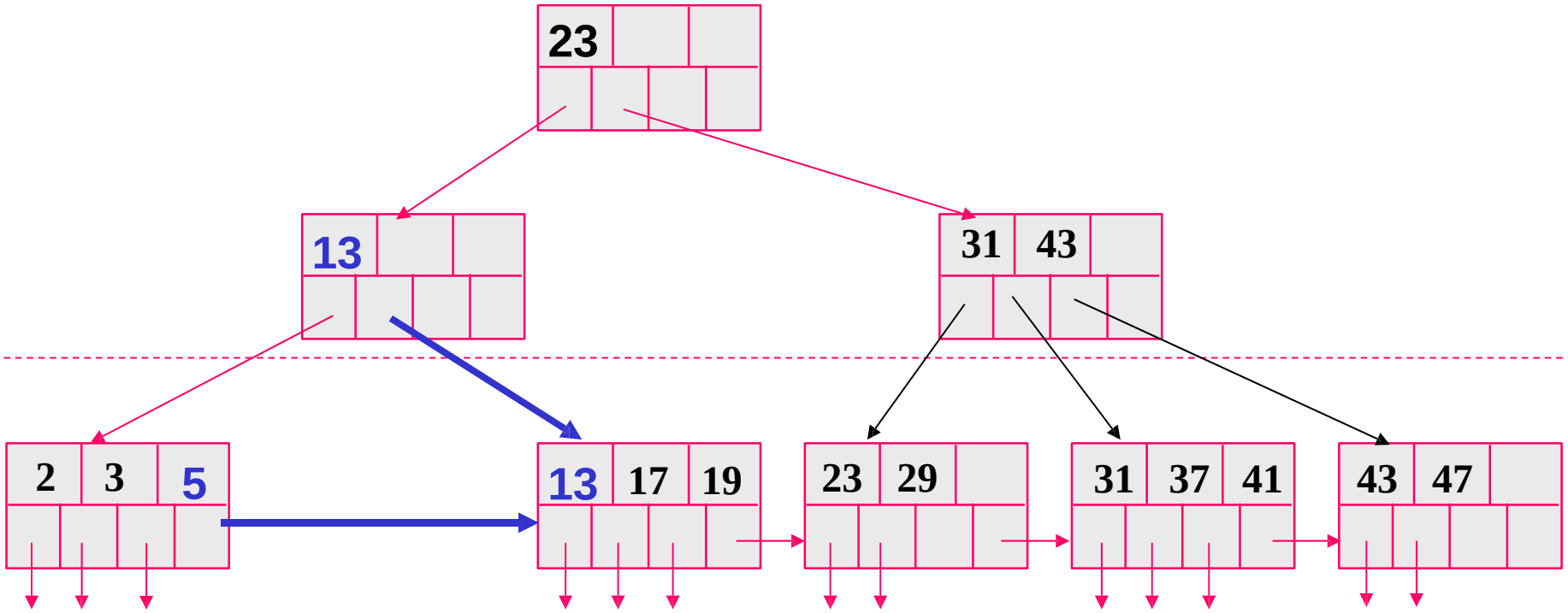
(1)叶子结点中寻找等于键值的记录，删除相应的指针及主文件中对应的记录
(2)将本结点的剩余键值，移动到左侧(或右侧)结点。此空结点将被删除。调整叶结点指针

删除键值为11的记录: 进行中



(3)如有调整，则进一步调整其上层非叶结点，重新确定其键值，以满足大于等于键值的记录都在其右侧。依次向上调整，直到根结点。

删除键值为11的记录: 进行中



(3)如有调整，则进一步调整其上层非叶结点，重新确定其键值，以满足大于等于键值的记录都在其右侧。依次向上调整，直到根结点。

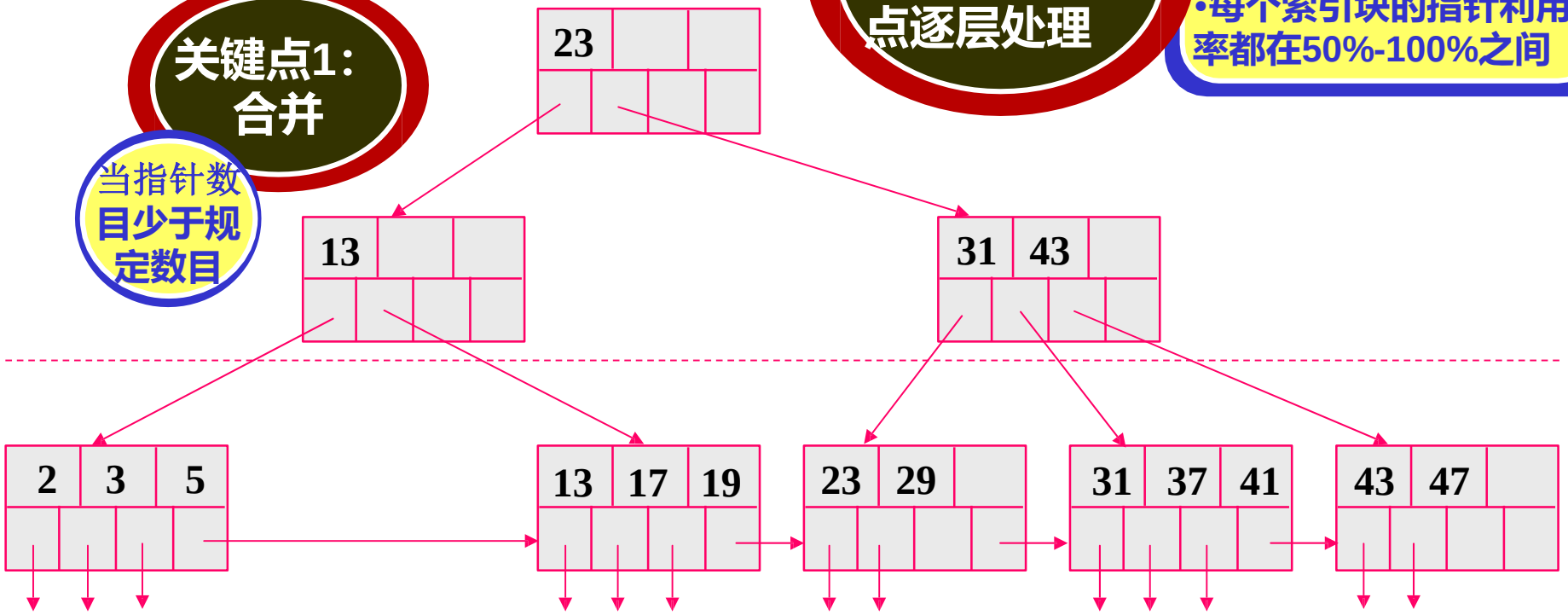
删除键值为11的记录: 最终状态

关键点1：合并

当指针数目少于规定数目

关键点2：由叶结点向根结点逐层处理

- 能够自动保持与主文件大小相适应的树的层次
- 每个索引块的指针利用率都在50%-100%之间



关键点3：指针调整

将算法表达出来

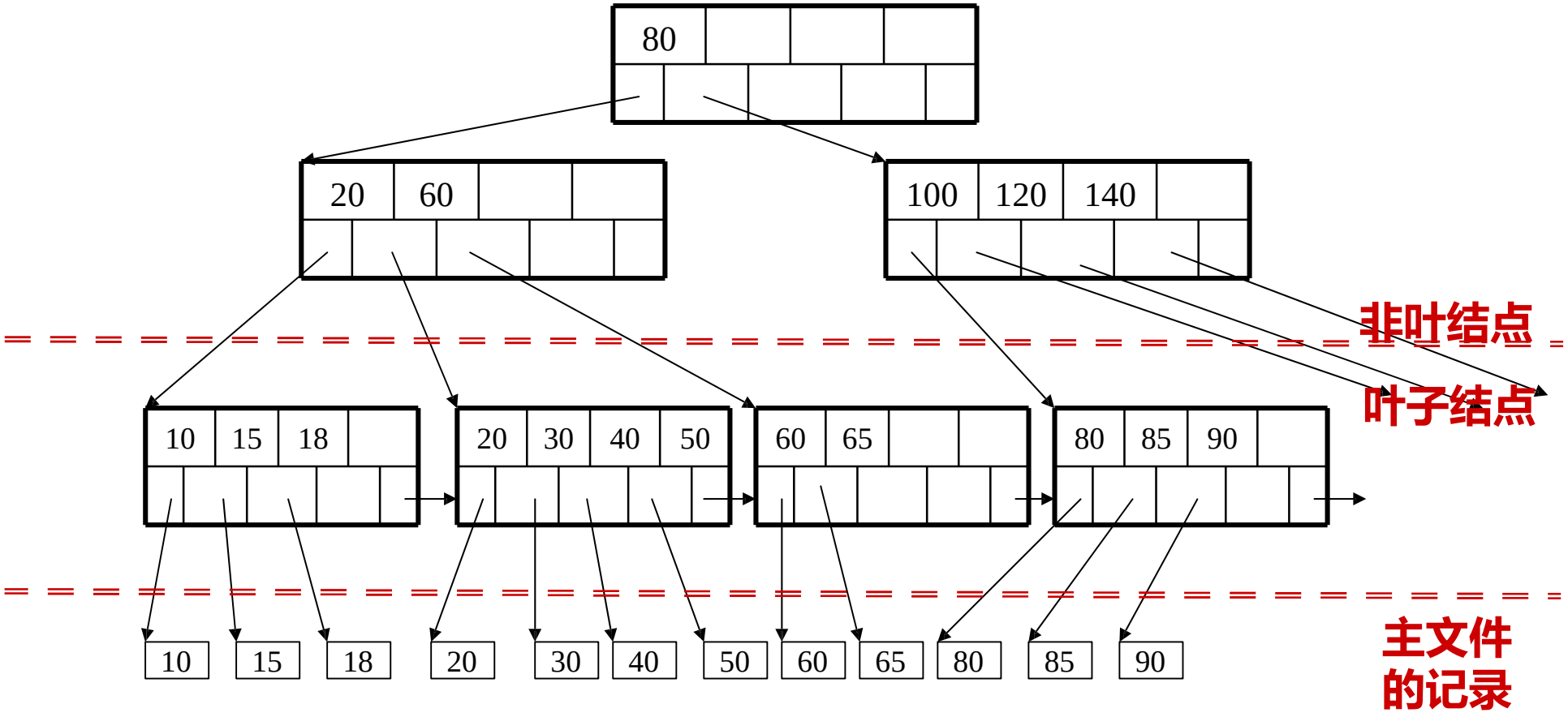
```
procedure delete (value K, pointer P)
    找到包含(K, P)的叶结点 L
    delete_entry(L, K, P)

procedure delete_entry(node N, value K, pointer P)
    从 N 中删除(K, P)
    if (N 是根结点 and N 只剩下一个子结点)
    then 使 N 的子结点成为该树的新的根结点并删除 N
    else if (N 中值/指针太少)  then begin
        令 N'为 parent(N)的前一子女或后一子女
        令 K'为 parent(N)中指针 N 和 N'之间的值
        if (N 和 N'中的项能放在一个结点中)
        then begin  /* 合并结点 */
            if  (N 是 N'的前一个结点) then swap_variables(N, N')
            if  (N 不是叶结点)
                then 将 K'以及 N 中所有指针和值附加到 N'中
                else 将 N 中所有(K_i, P_i)对附加到 N'中; 令 N'.P_n = N.P_n
            delete_entry (parent(N), K', N); 删除结点 N
        end
    else begin /* 重新分布: 从 N'借一个索引项 */
        if (N'是 N 的前一个结点) then begin
            if (N 是非叶结点) then begin
                令 m 满足: N'.P_m 是 N'的最后一个指针
                从 N'中去除(N'.K_{m-1}, N'.P_m)
                插入(N'.P_m, K')并通过将其他指针和值右移使之成为
                    N 中的第一个指针和值
                用 N'.K_{m-1}替换 parent(N)中的 K'
            end
            else begin
                令 m 满足: (N'.P_m, N'.K_m)是 N'中的最后一个指针/值对
                从 N'中去除(N'.P_m, N'.K_m)
                插入(N'.P_m, N'.K_m)并通过将其他指针和值右移使之成为
                    N 中的第一个指针和值
                用 N'.K_m 替换 parent (N)中的 K'
            end
        end
        else  …与 then 的情况对称…
    end
  end
end procedure
```

此算法可参见教材

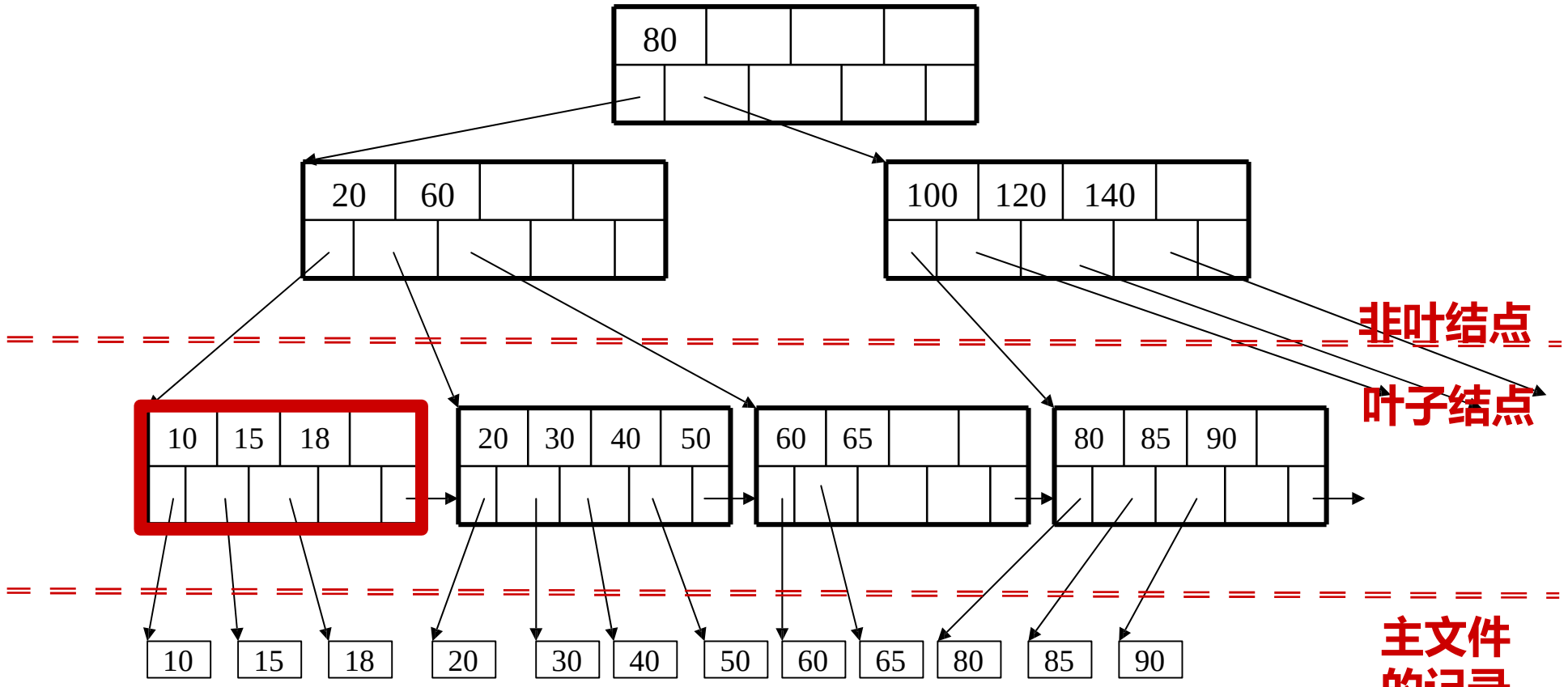
B+树之分裂与合并过程再示例

示例：初始B+树



问题：插入键值为19的记录

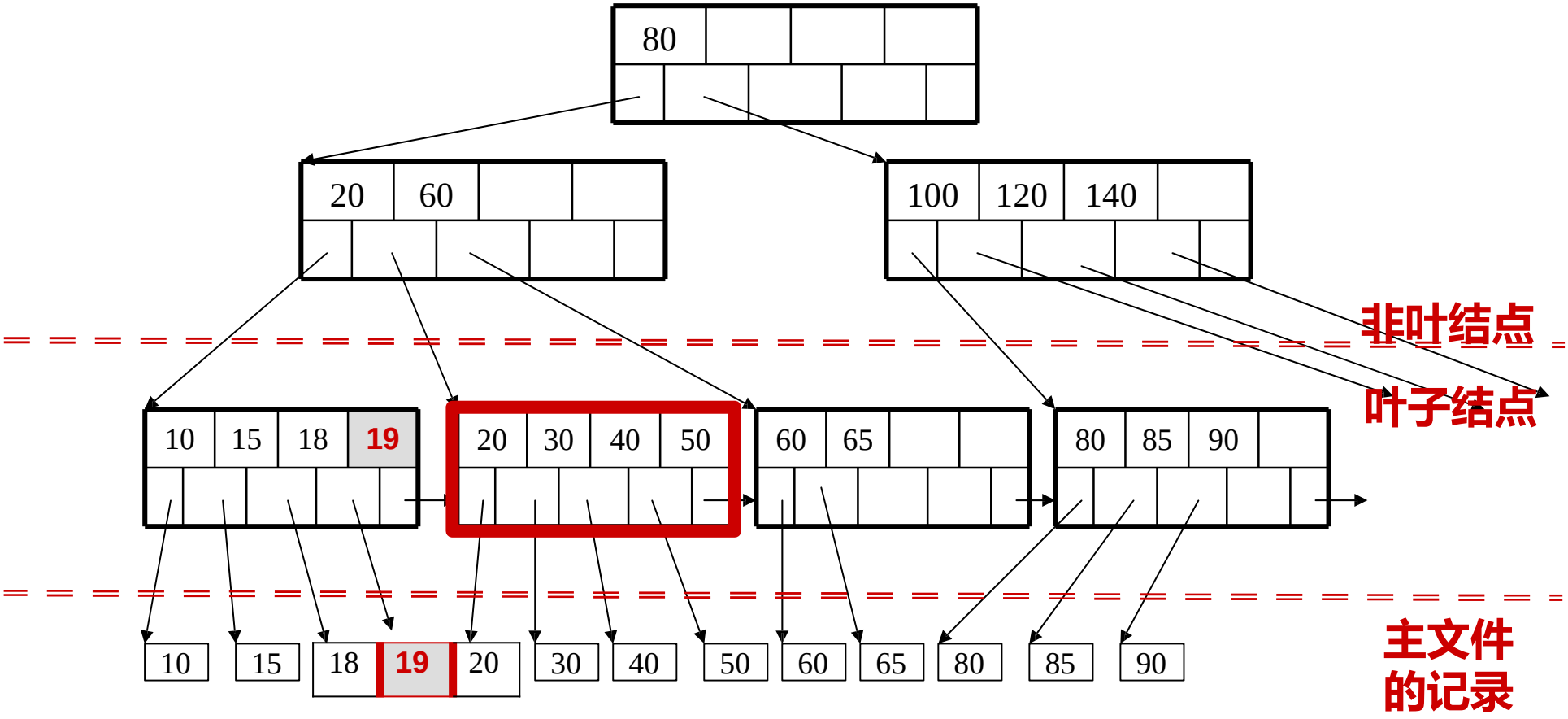
- 先定位待插入键值的叶子结点：从根结点开始向下；
- 检查叶子结点是否已满?
- 如未满，则可直接插入



键值为19的记录插入后

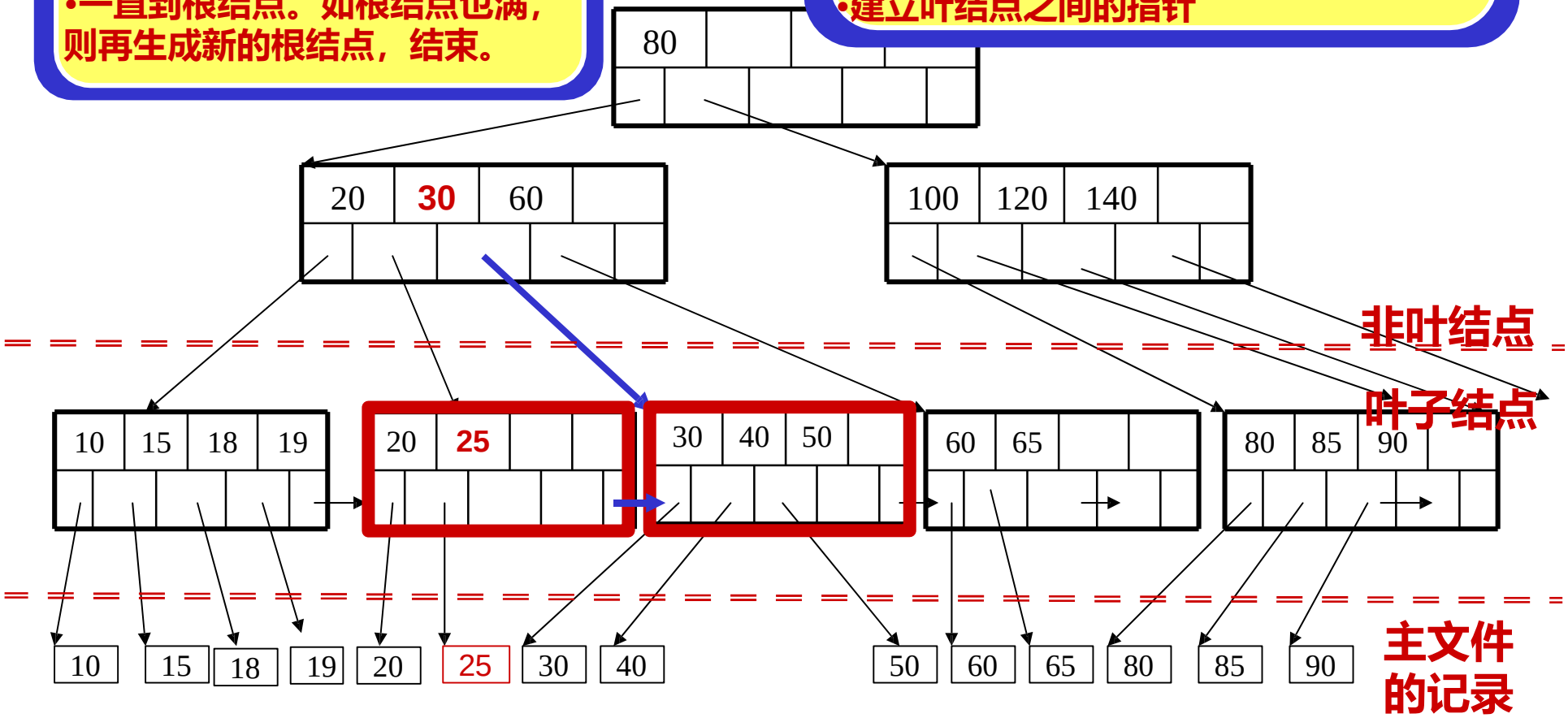
问题：再插入键值为25的记录

- 先定位待插入键值的叶子结点：从根结点开始向下；
- 检查叶子结点是否已满?
- 如已满，则需分裂该结点为两个



•分裂结点，将所涉及的键值 准均分为两个结点中存储(包含待插入的)
•插入键值为25的键值并指向主文件记录
•调整父结点的指针及键值：在父结点中插入键值为30的索引项
•建立叶结点之间的指针

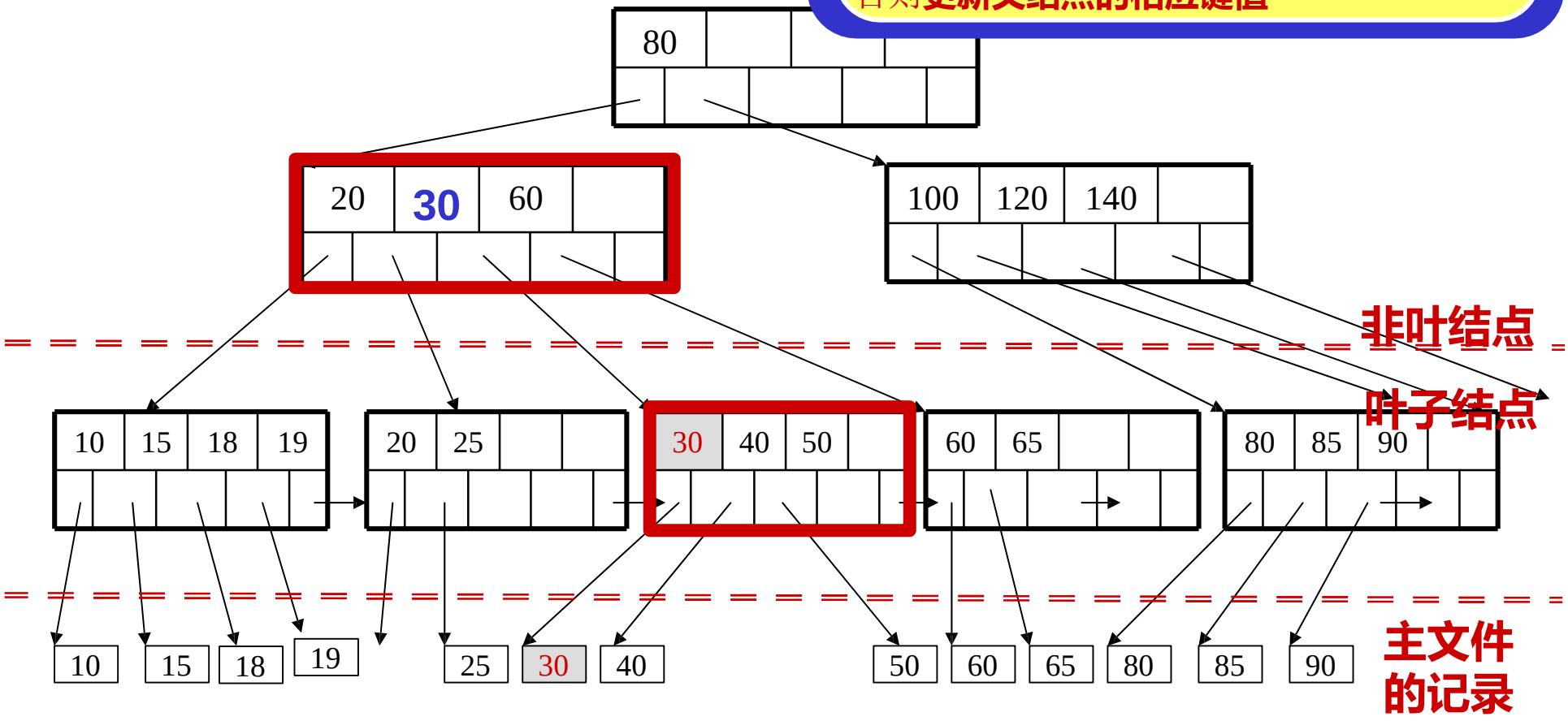
•如父结点已满，则如此继续将其分裂为两个结点，保存相应键值
•一直到根结点。如根结点也满，则再生成新的根结点，结束。



键值为25的记录插入后

问题：删除键值为30的记录

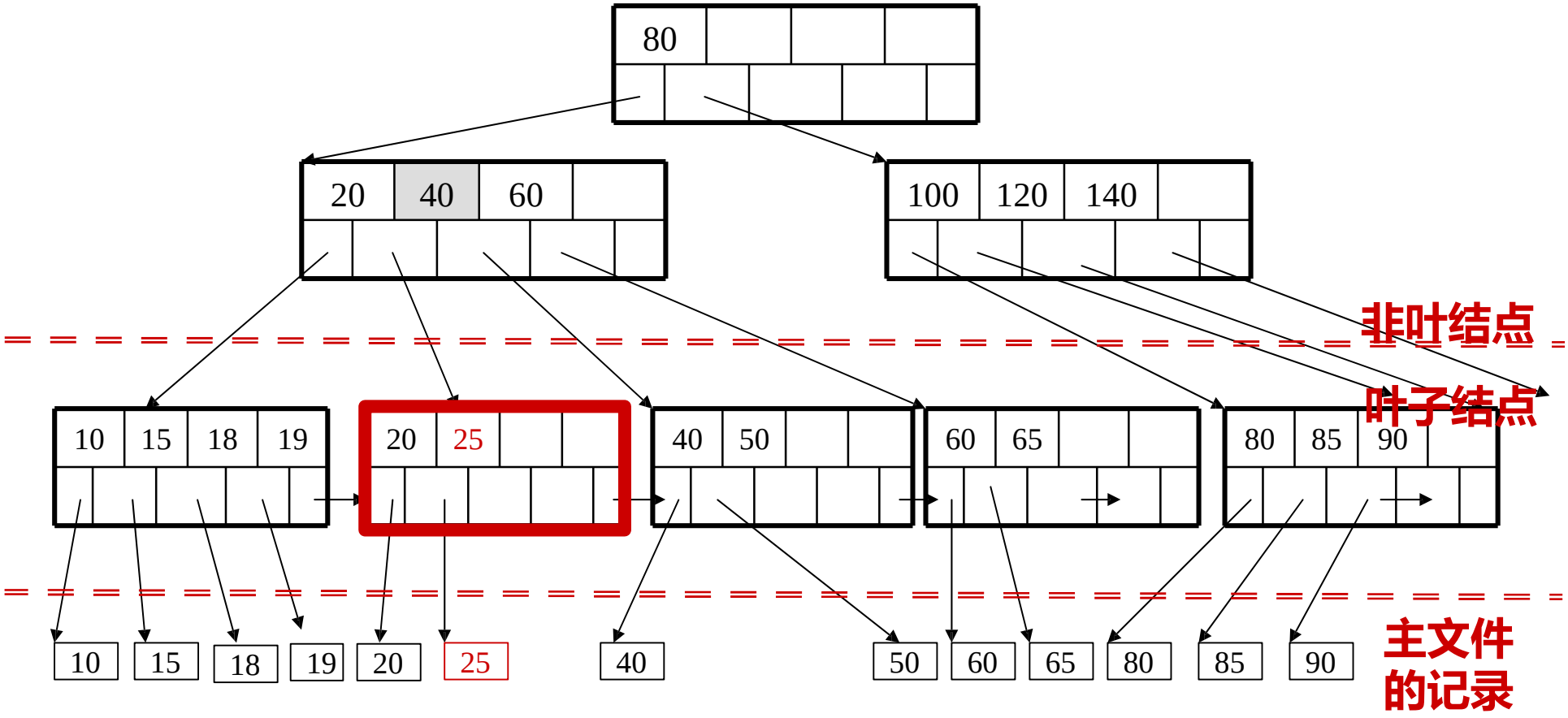
- 先定位待删除键值的叶子结点：从根结点开始向下；
- 删除键值及其主文件记录?
- 如指针数目不少于规定数目，则可以结束
- 否则更新父结点的相应键值



键值为30的记录删除后

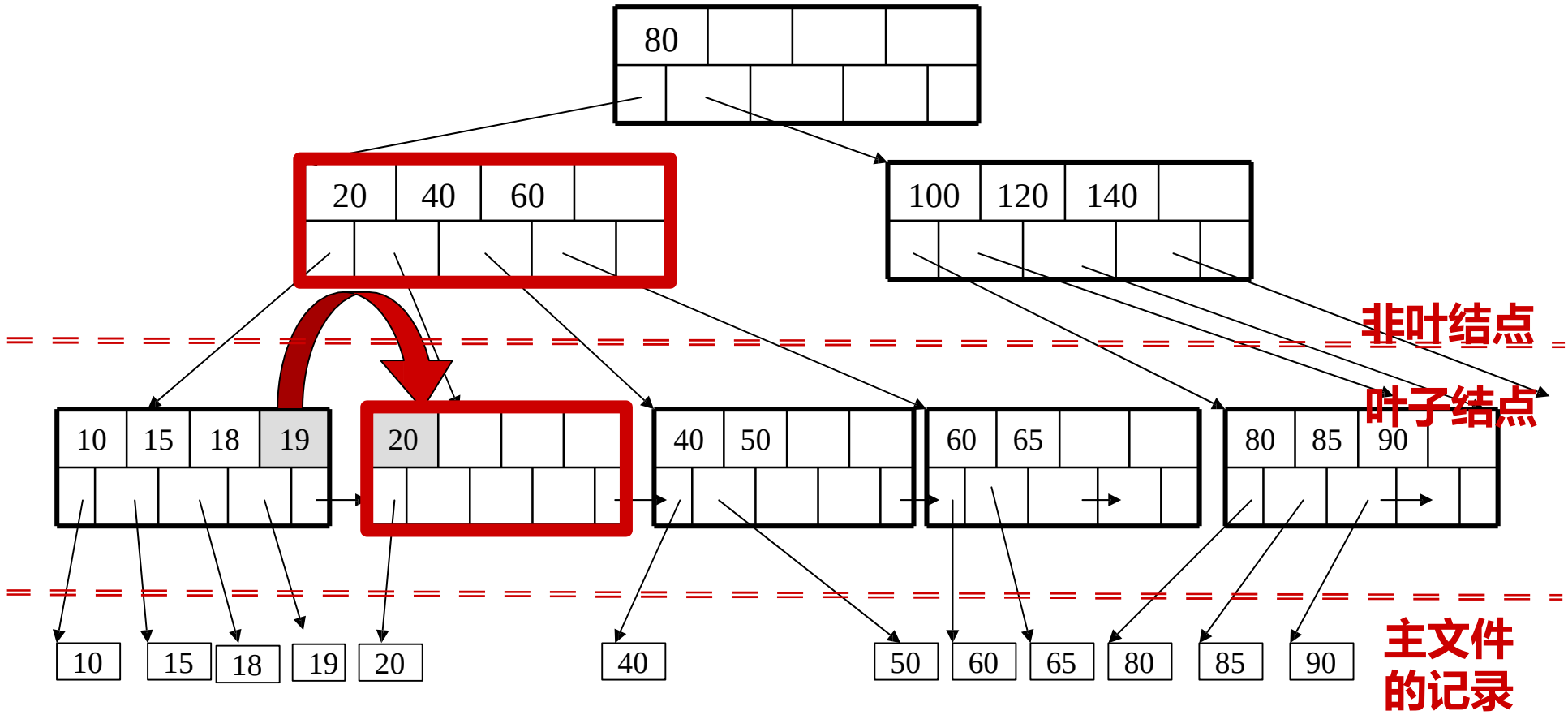
问题：删除键值为25的记录

•先定位待删除键值的叶子结点：从根结点开始向下；删除键值及其主文件记录?



键值为25的记录删除后

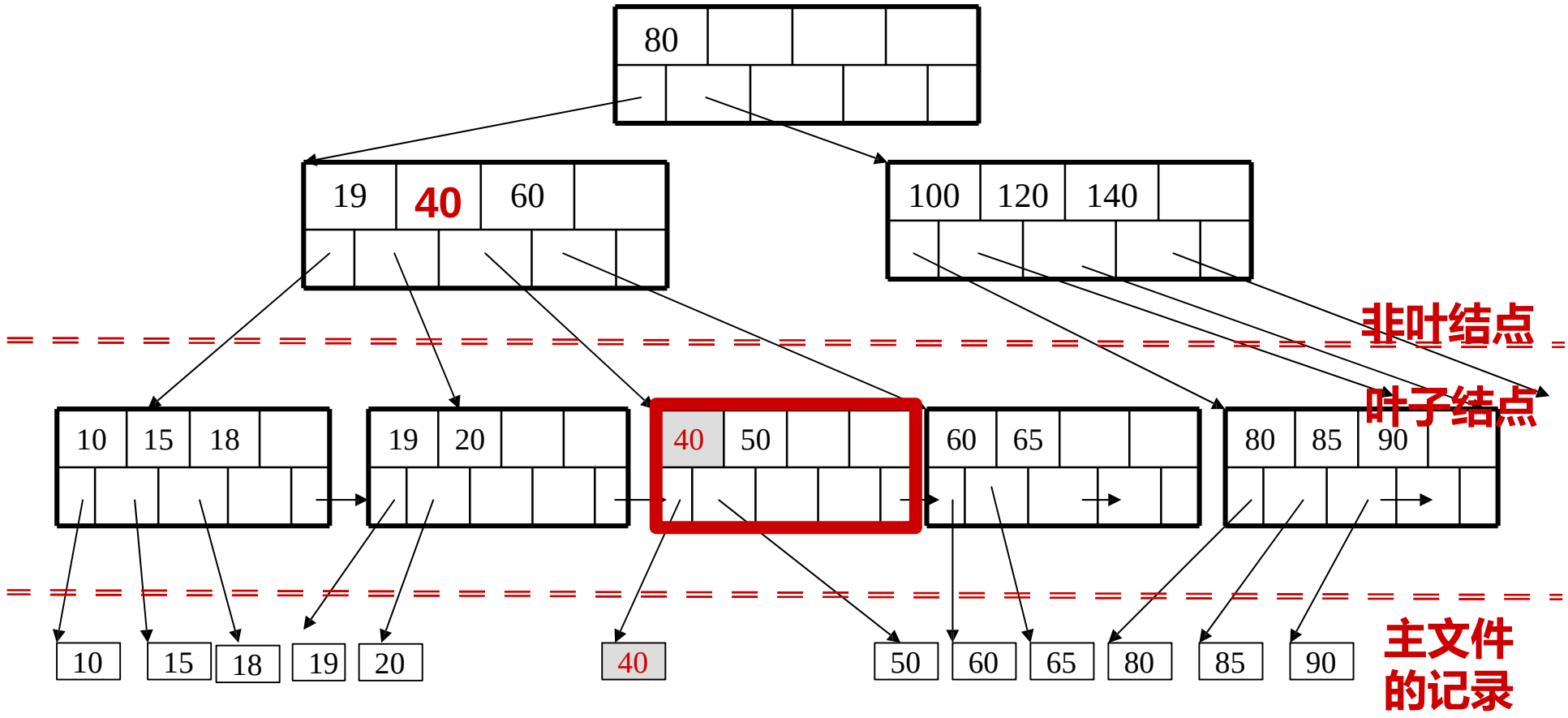
- 键值删除后，如指针数目少于规定数目，则需要合并
- 从相邻结点能否转移一些键值到该结点，如可以，则转移；并更新父结点之相应键值



转移键值后

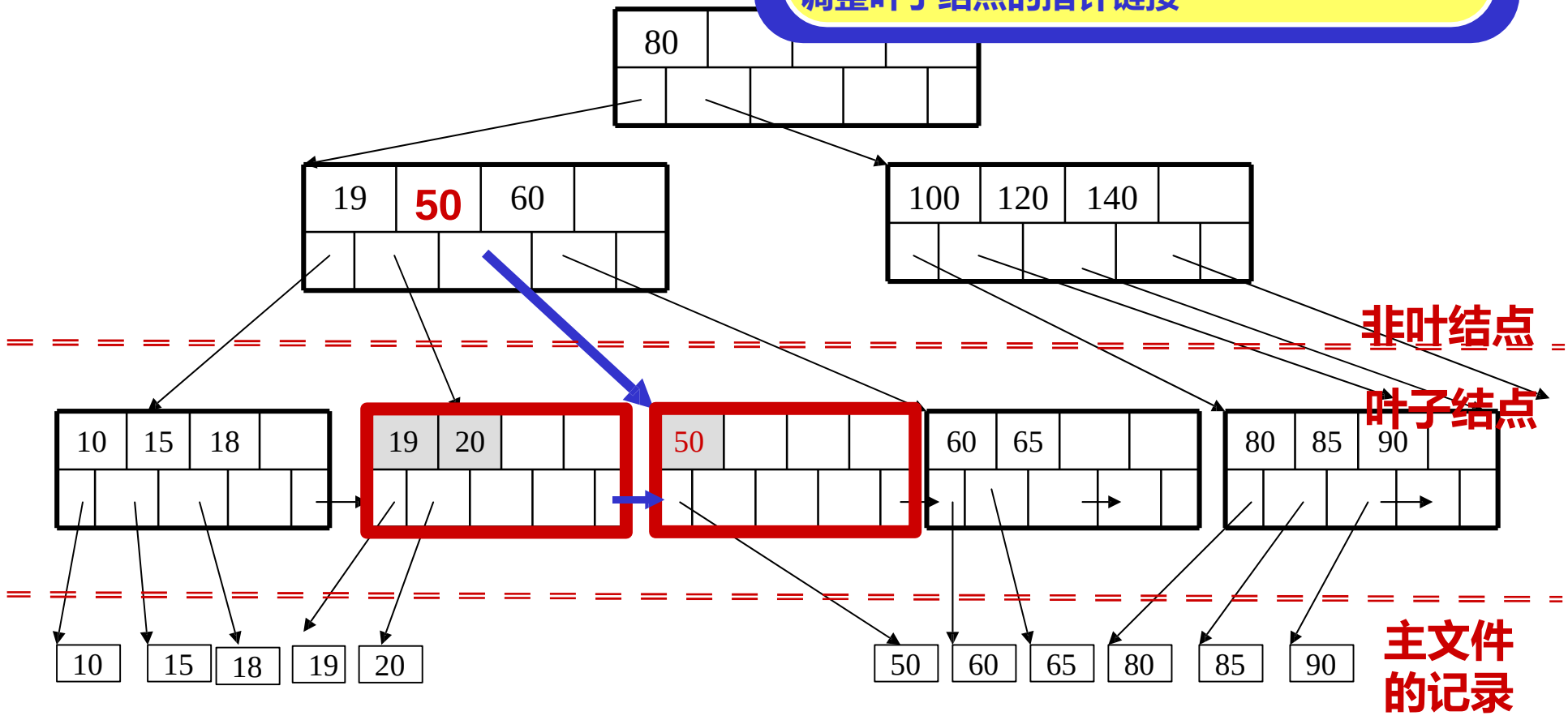
问题：删除键值为40的记录

- 先定位待删除键值的叶子结点：从根结点开始向下；删除键值及其主文件记录?
- 如是第一个，则需调整父结点的键值



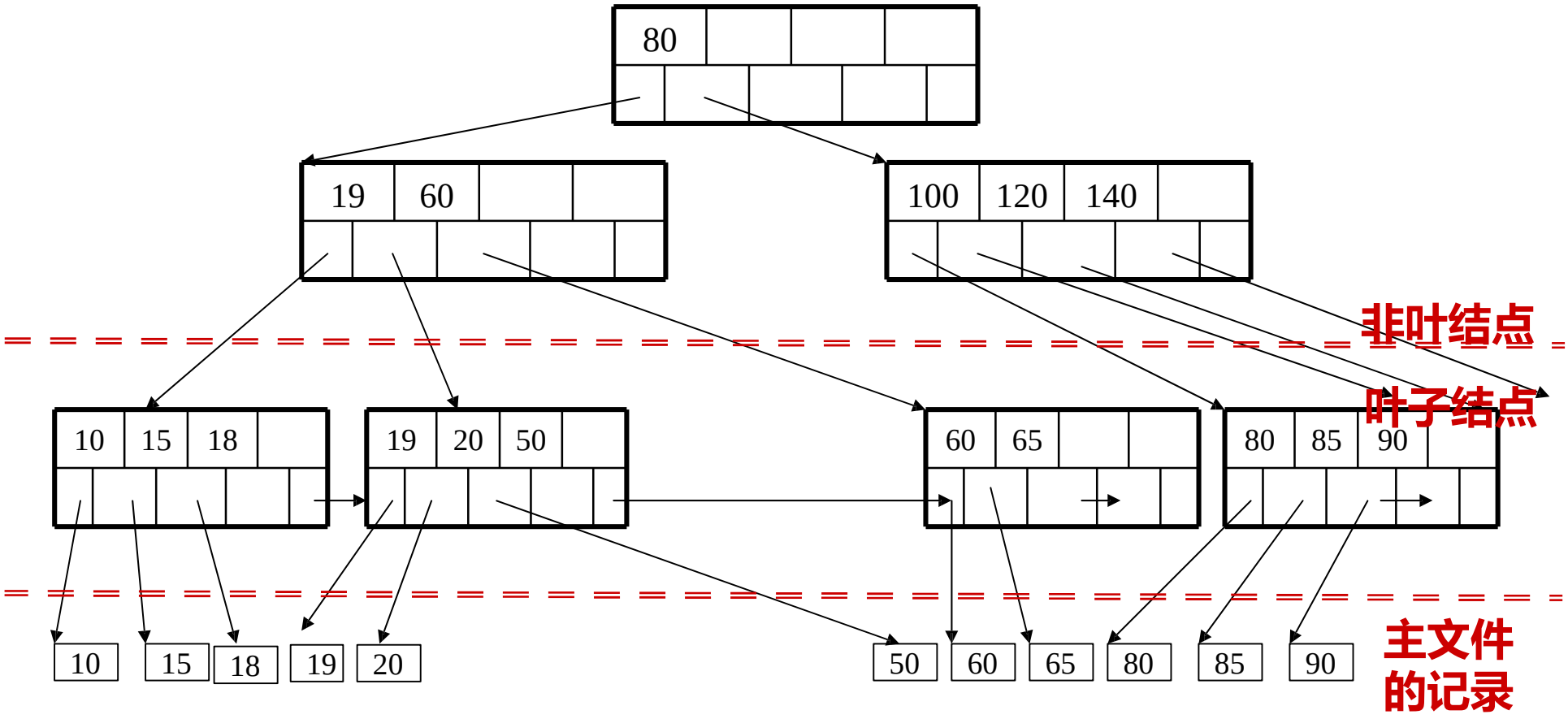
键值为40的记录删除后

- 键值删除后，如指针数目少于规定数目，则需要合并
- 如不能从相邻结点转移键值到该结点，则考虑结点合并。合并后调整父结点键值及次序。
- 调整叶子结点的指针链接



结点合并后

- 父结点，如在删除掉索引项及指针后，指针数目小于规定的数目，则继续前面的步骤处理，直至根结点；
- 如指针不小于规定的数目，则结束。



插入记录时：

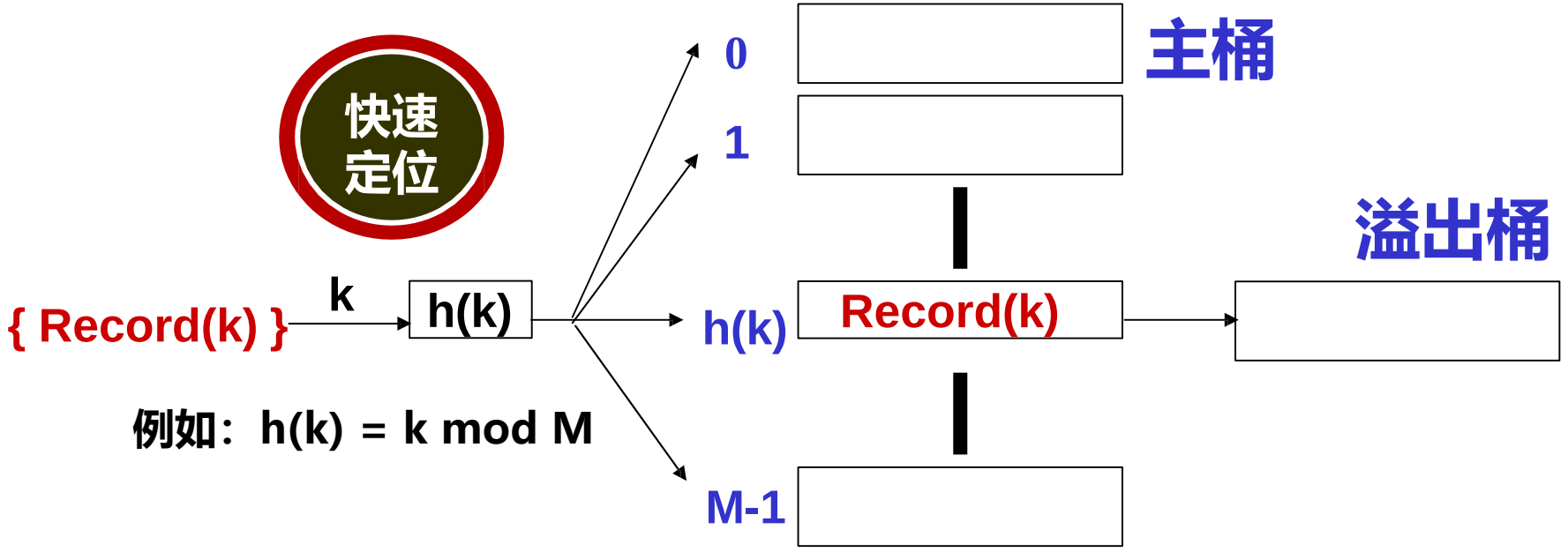
- 可能需要分裂：当索引块充满时，需要分裂。
- 分裂可能引发连锁反应，由叶结点直至根结点。
- 分裂后需要仔细调整索引键值及指针。
- 注意叶子结点之间链接关系的调整。

删除记录时：

- 可能需要合并：合并记录—从相邻结点中转移索引项(仍旧是两个结点块)
- 可能需要合并：两个结点块合并成一个结点块。
- 合并可能引发连锁反应，由叶结点直至根结点。
- 合并后需要仔细调整索引键值及指针。
- 注意叶子结点之间链接关系的调整。

散列

- 有M个桶，每个桶是有相同容量的存储(可以是内存页，也可以是磁盘块)
- 散列函数 h(k)，可以将键值k映射到 { 0, 1, ..., M-1 }中的某一个值
- 将具有键值k的记录Record(k) 存储在对应h(k)编号的桶中



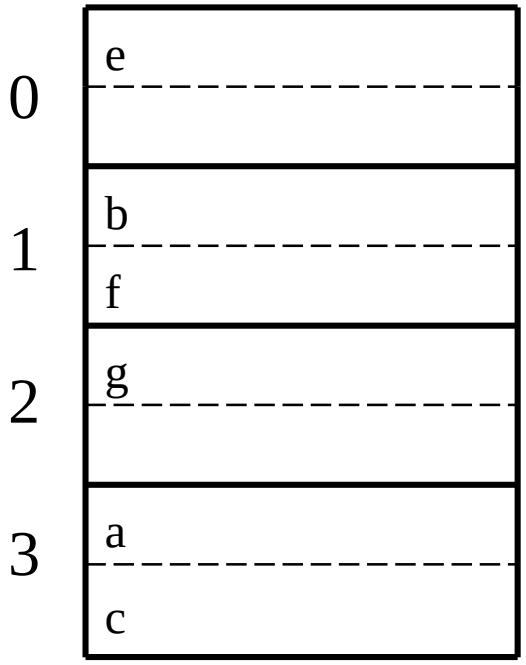
目标：选择一个合适的散列函数，将一个Record集合(每个Record都包含一个关键字k) 均匀地映射到M个桶中。即：对于集合中任一个关键字，经散列函数映射到地址集合中任何一个地址的概率是近乎相等的。

散列索引

➢内存数据可采用散列确定存储页，主文件可采用散列确定存储块，索引亦可采用散列确定索引项的存储块

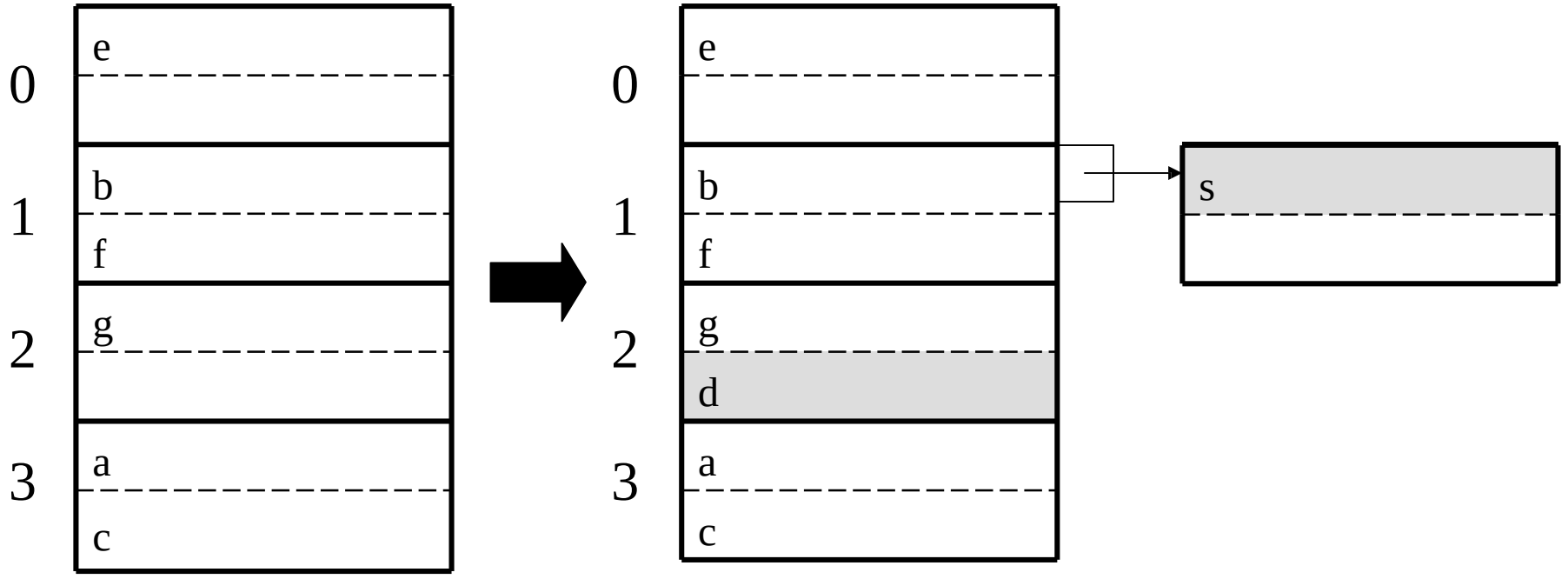
➢M个桶。一个桶可以是一个存储块，亦可是若干个连续的存储块。

示例：假设 1存储块可存放2个键值及其指针
M=4; 1个桶为 1 个存储块
h(x)满足:
✓h(e)=0 ； h(b)=h(f)=h(s)=1
✓h(d) =h(g)=2 ； h(a)=h(c)=3



●**问：如何查询键值 a的索引项?**
✓计算 h(a)=3
✓读取 3号桶，获得键值a的索引项.
需要1个磁盘块读取

- 问：如何插入键值d的索引项?
  - ✓计算h(d)=2
  - ✓如2号桶有空间，则将索引项d插入2号桶中
- 问：如何插入键值s的索引项?
  - ✓计算 h(s)=1
  - ✓1号桶无空间，则申请一溢出桶，插入s

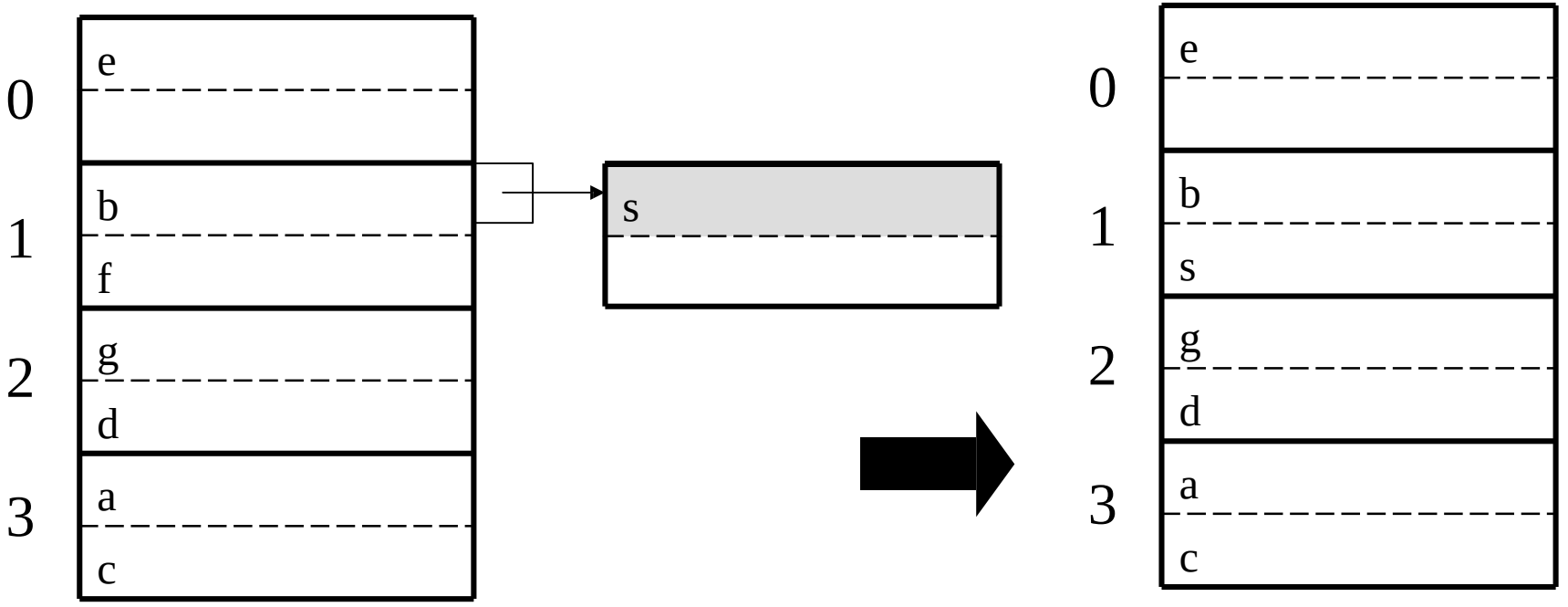


●**问：如何删除键值f.**

✓计算h(f)=1

✓删除1号桶中的键值f

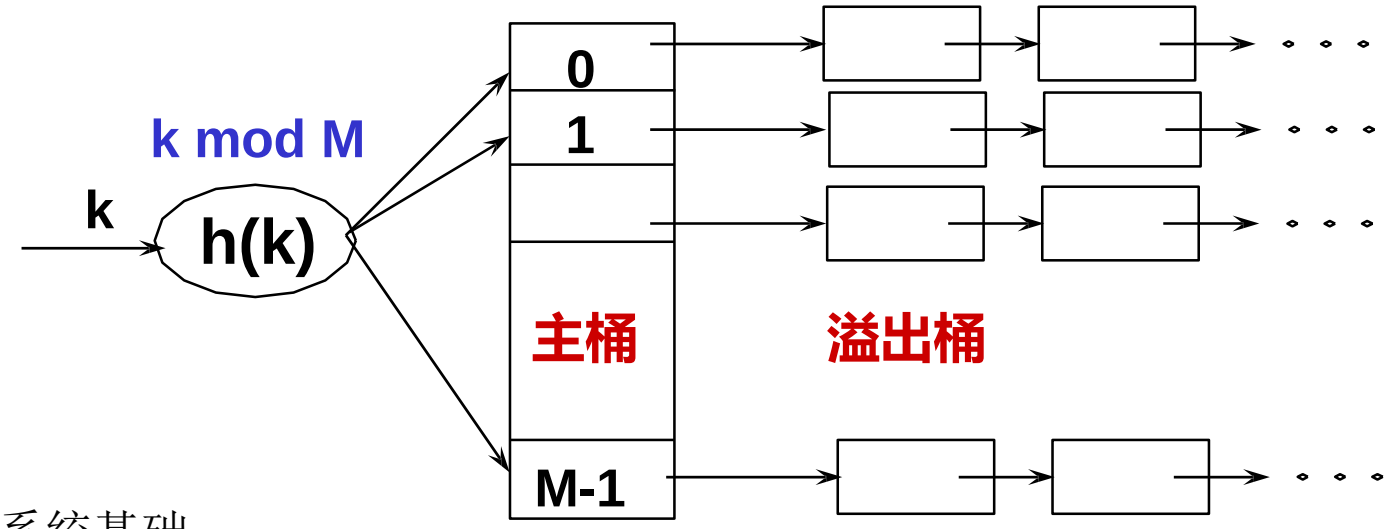
✓将溢出桶中的s合并到主桶中，删除溢出桶



✓**散列索引的目标**：最好是没有溢出桶，每一个散列值仅有一个桶。读写每一个键值都只读写一个存储块。

➢**均匀分布如何做到？** **期望将所有数据分布均匀地存储于M个桶中，使每一个桶的数据成为具有某种特征值h(k)的数据集合。---散列函数的选择。**

✓**桶的数目M如何计算？** **在键值几倍于桶的数目时，每个散列值都可能多于一个桶，形成一个主桶和多个溢出桶的列表，此时需要二次检索：先散列找到主桶号，再依据链表逐一找到每个溢出桶。---桶的数目的确定。**



桶的数目M是固定值----静态散列索引

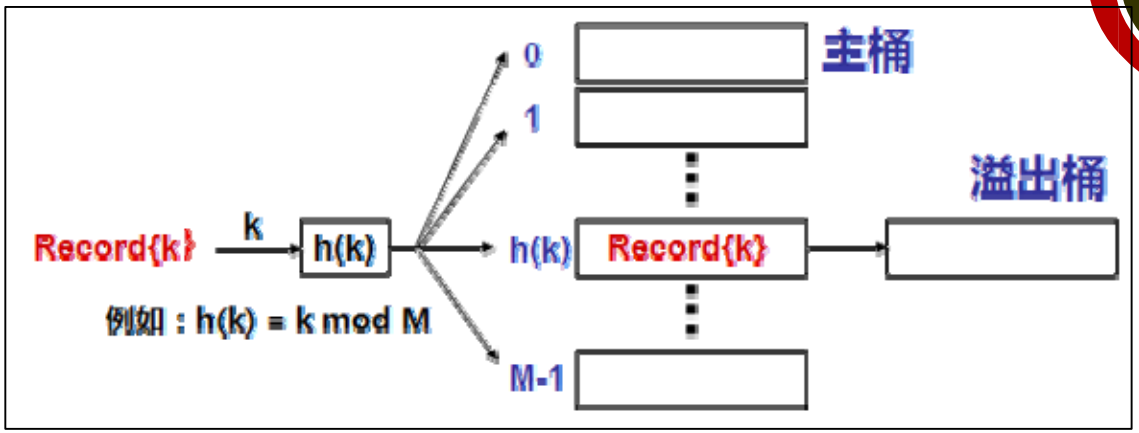
•如果桶的数目M不变：M过大，则浪费；M过小，则将产生更多的溢出桶，增加散列索引检索的时间。

桶的数目随键值增多，动态增加----动态散列索引

•h(k)是和桶的数目M相关的。M的变化会否影响原来存储的内容呢?

•是否需要将原来已经散列-存储的数据按新的桶数重新进行散列-存储呢?

这个如何处理呢?



可扩展散列索引

(1)拟解决的问题

桶的数目随键值增多，动态增加----**动态散列索引**

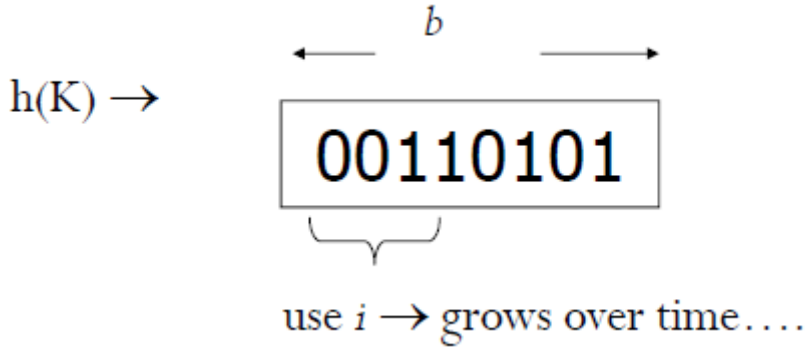
- **可扩展散列索引**
- **线性散列索引**

可扩展散列索引

•为桶引入一间接层，即用一个指向块的指针数组来表示桶，而不是用数据块本身组成的数组来表示桶

•指针数组能增长，其长度总是2的幂。因而数组每增长一次，桶的数目就翻倍。不过，并非每个桶都有一个数据块；如果某些桶中的所有记录可以放在一个块中，则这些桶可能共享一个块。

•散列函数h为每个键计算出一个K位二进制序列，该K足够大，比如32。但是桶的数目总是使用从序列第一位或最后一位算起的若干位，此位数小于K,比如说i位。也就是说，当i是使用的位数时，桶数组将有2^i个项。



可扩展散列索引
(1)基本思想示例

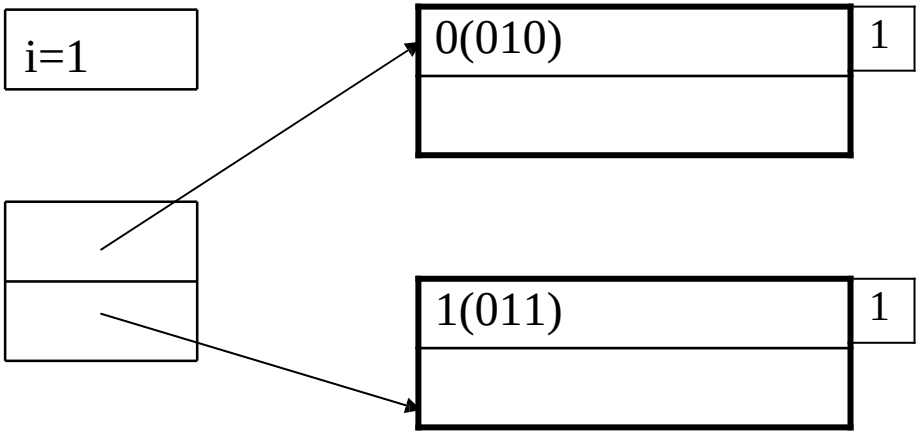
参数k

参数i

参数n=2^i

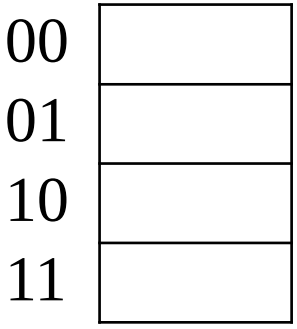
i为散列函数当前已经使用到的最多位数。即当前的桶数为2^i.

右上角标记本块散列函数使用位数。--局部

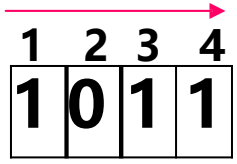


可变的桶
(指针数组)
由2^0个，2^1个，2^2个…
逐渐增加至2^k个

k为散列函数所可能使用的最多位数。即可能的最大桶的数目为2^k，逐渐被使用。



i的位次



k位二进制位

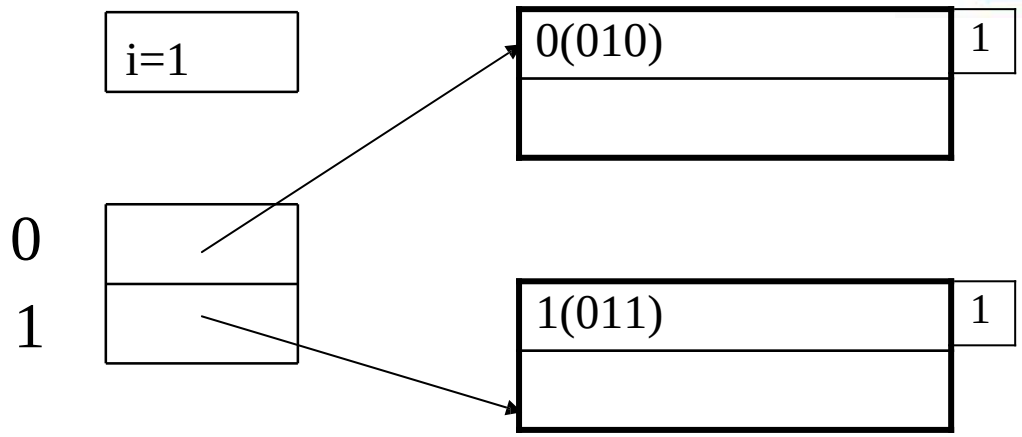
索引块
取前*i*位，按照前i位的值找到对应的索引块

可扩展散列索引
(2)操作示例

k=4

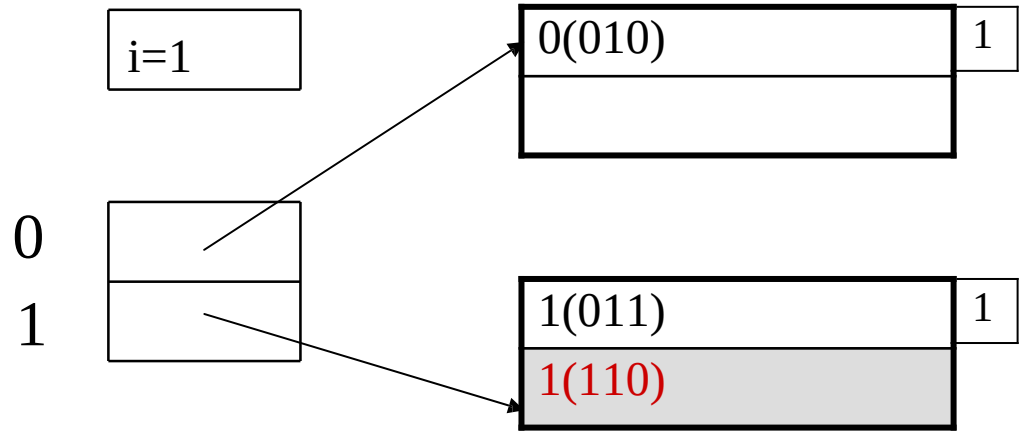
i=1

n=2^i=2



•插入 1110

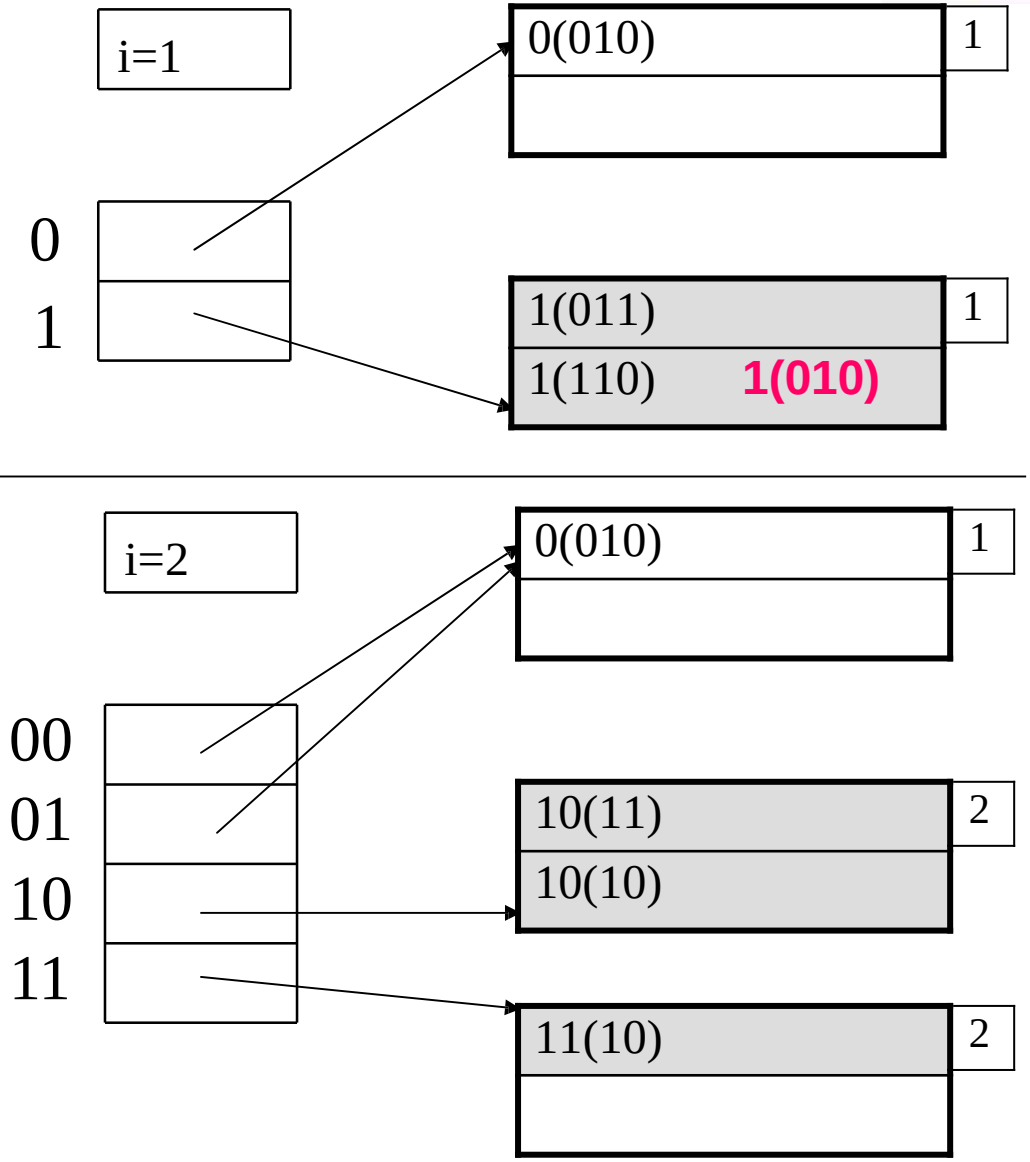
•取前i位，确定索引块。
•如有空间，则存储



可扩展散列索引
(2)操作示例

- 插入 1010

- 取前i位，确定索引块。
- 如已满，则需要扩展散列桶，进行分裂：
  - ✓ i增加1
  - ✓重新散列该块的数据到两个块中。其他不变

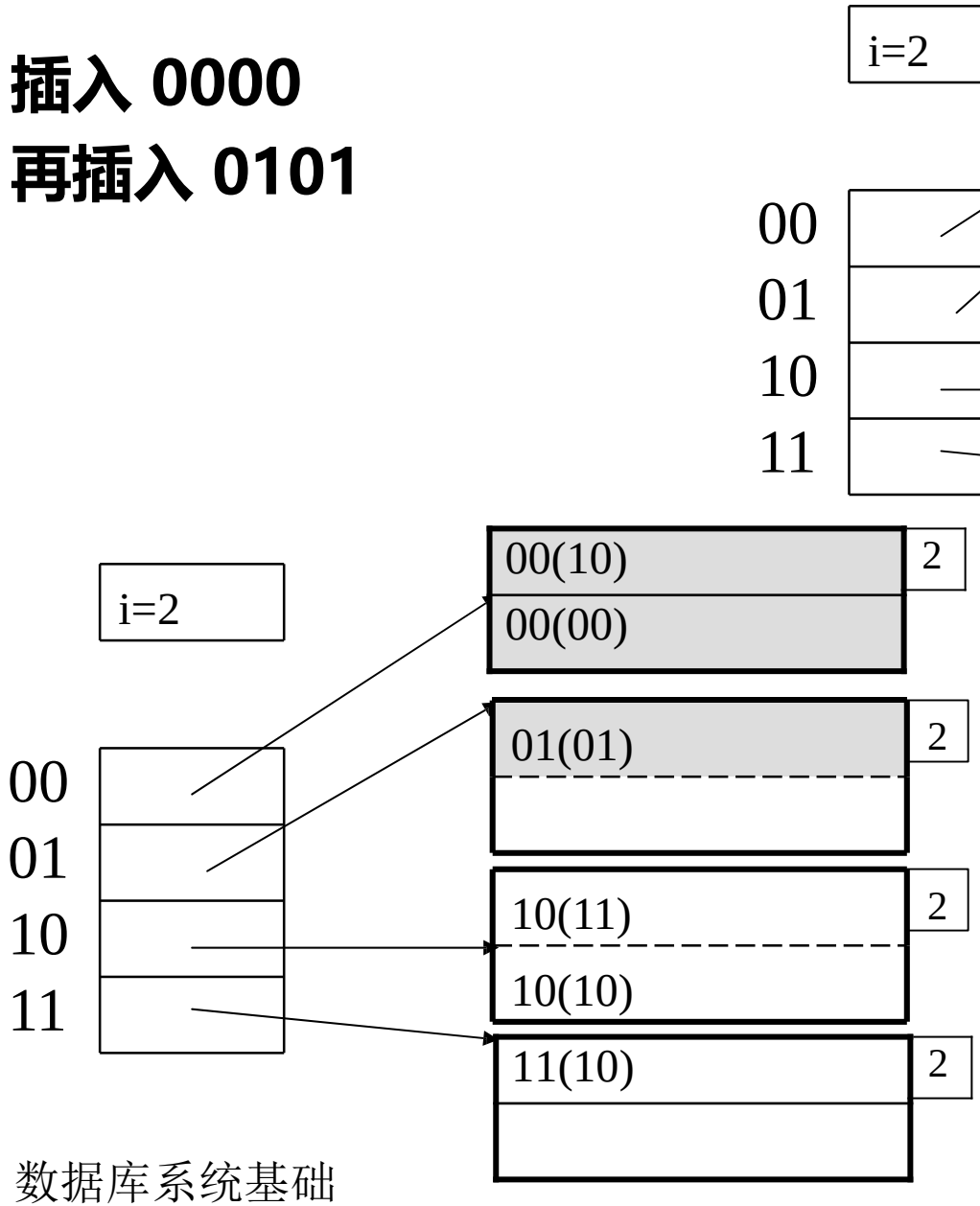


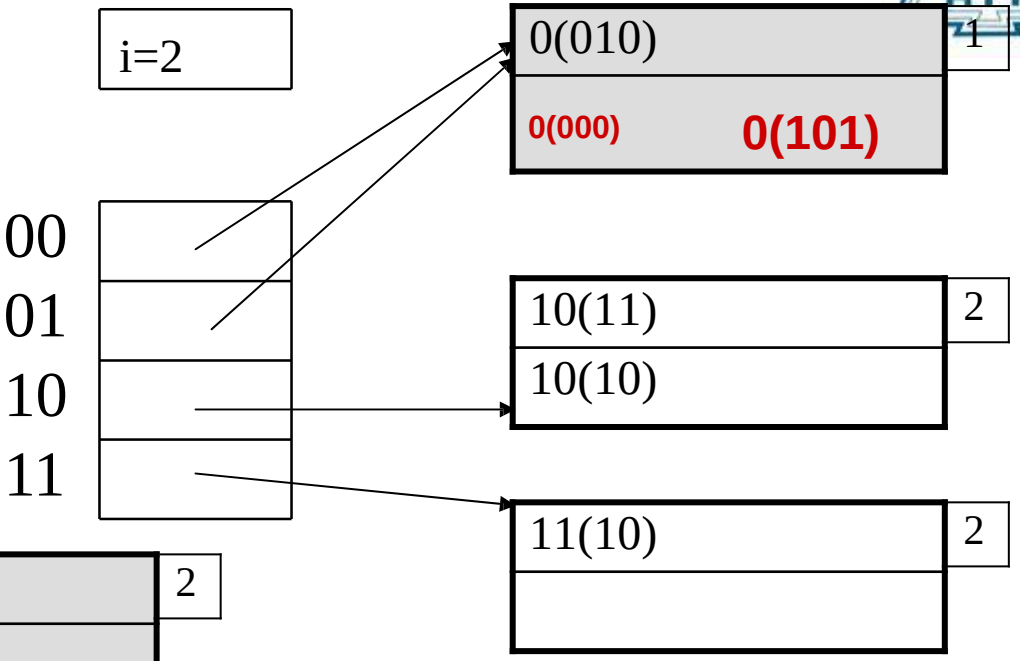
k=4

i=2

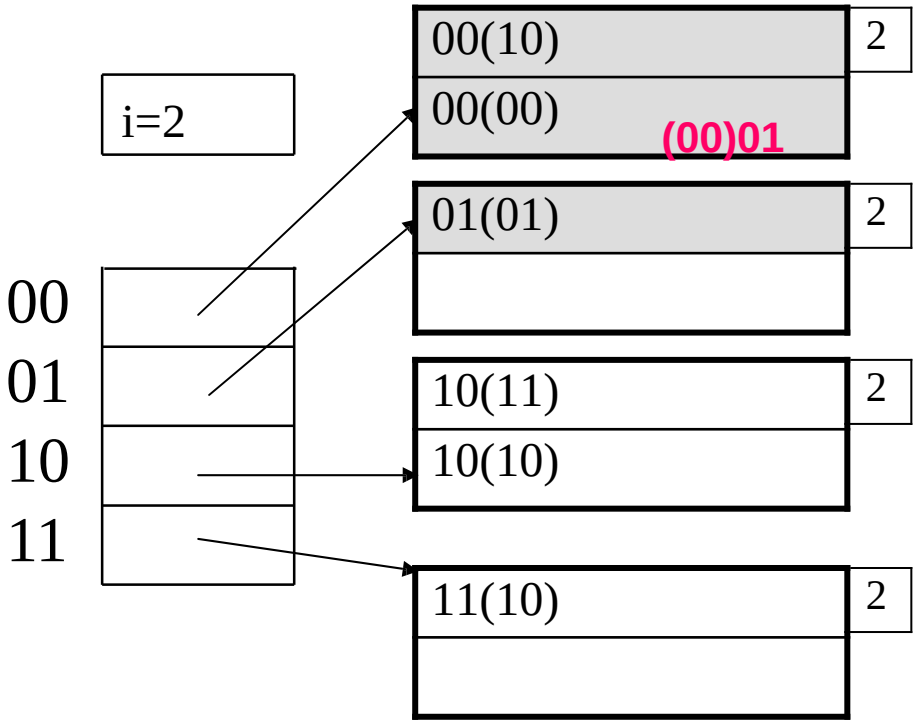
$n=2^i=4$

插入 0000
再插入 0101

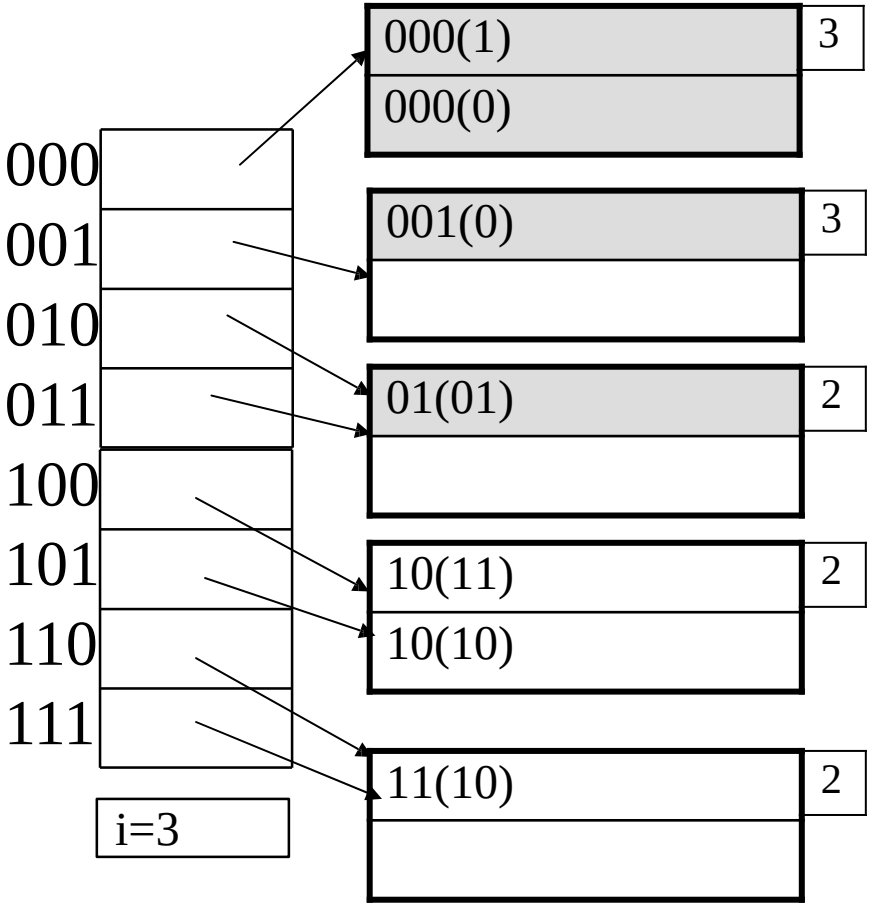




- 取前i位，确定索引块。
- 如未满，则存入。
- 如已满，则需要扩展散列桶，进行分裂：
  - ✓暂时i可能无需增加
  - ✓重新散列该块的数据到两个块中。其他不变



插入 0001



- 取前i位，确定索引块。
- 如未满，则存入。
- 如已满，则需要扩展散列桶，进行分裂：
  - ✓此时i需增加1
  - ✓重新散列该块的数据到两个块中。其他不变

问题

•当桶数组需要翻倍时，要做大量的工作(当i很大时)；

•当桶数翻倍后，其在主存中可能就装不下了，或者要占用更大的空间

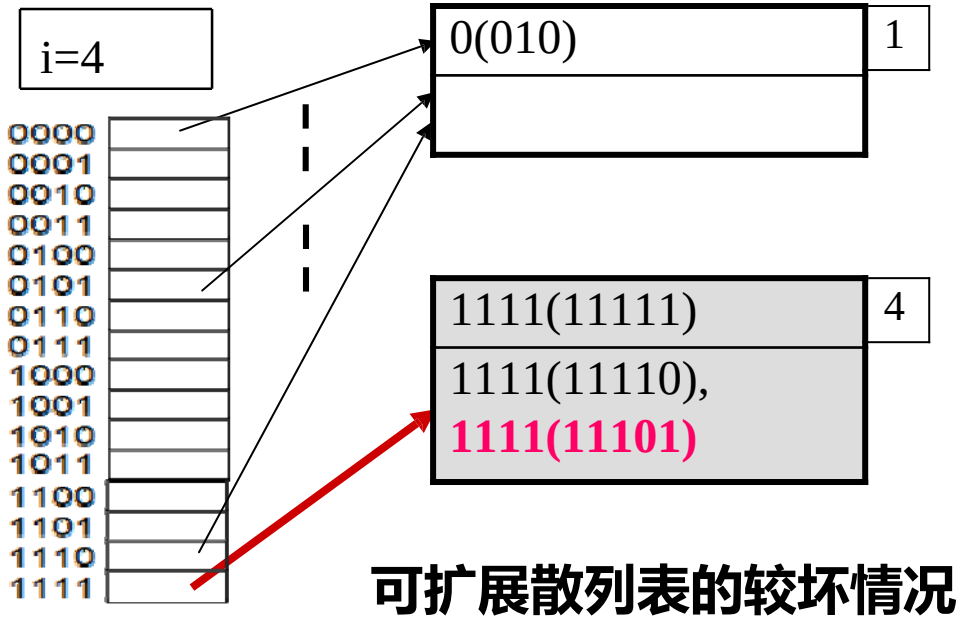
•如果每块的记录数很少，那么很有可能某一块的分裂比在逻辑上需要的分裂时间提前很多。例如：块中存放2个记录，即使记录总数远小于2^{20}, 但也可能出现三个记录的前20位二进制位序列一样，在这种情况下，将不得不使用i=20和100万个桶数组。

i=1时3个数组在一个桶 中；要分裂；则2个桶

i=2时三个数组仍旧在一个桶中；要分裂，则2^2个桶

i=3时三个数组仍旧在一个桶中；要分裂，则2^3个桶

… …



可扩展散列表的较坏情况

线性散列索引

•桶数n的选择：总是使存储块的平均记录数，保持与存储块所能容纳的记录总数成一个固定的比例，例如80%。超过此比例，则桶数增长1块，分裂。

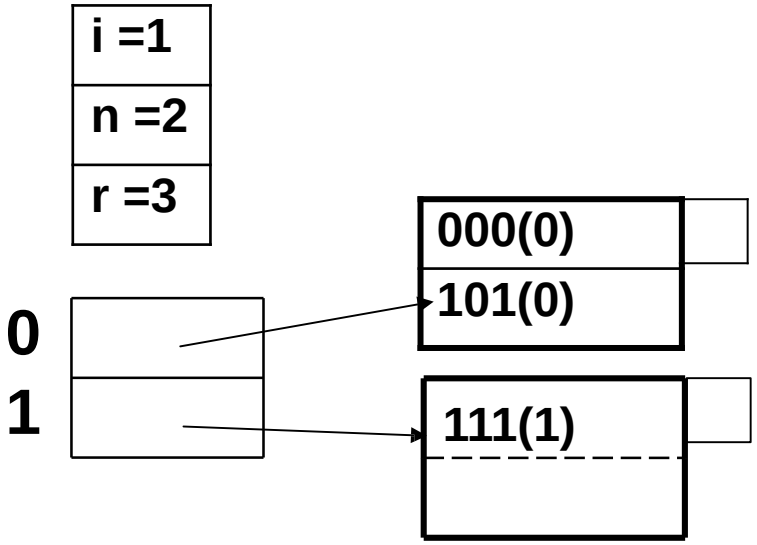
---线性 增长，每次增1。

•存储块并不总是可以分裂，所以允许有溢出块，尽管每个桶的平均溢出块数远小于1。

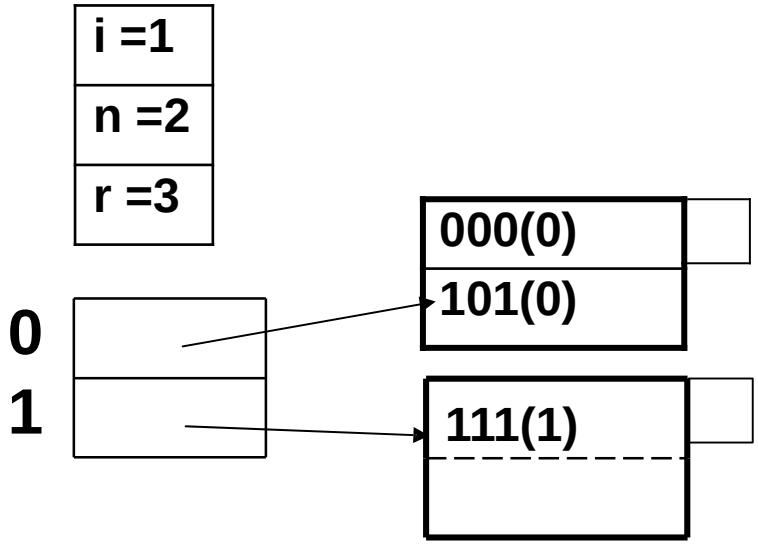
•用来做桶数组项序号的二进制位数是$\lceil \log_2 n \rceil$，其中n是当前的桶数。**这些位总是从散列函数得到的位序列的右端(即低位)开始取。**

•假定散列函数值的i位为桶数组项编号，且有一个键值为K的记录 想要插入到编号为$a_1a_2...a_i$的桶中，即$a_1a_2...a_i$是h(K)的后i位。把$a_1a_2...a_i$当 作二进制整数，设它为m。

- ✓**如果m<n，则编号为m的桶存在，并把记录存入该桶中。**
- ✓**如果n≤m<2^i，那么桶还不存在，因此我们把记录存入桶m-2^{i-1}，也就是当我们把a_1(它肯定是1)改为0时对应的桶。**

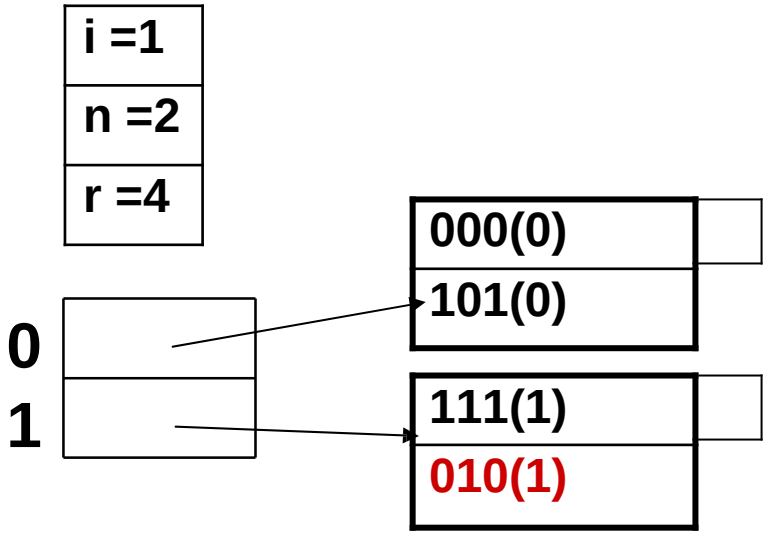


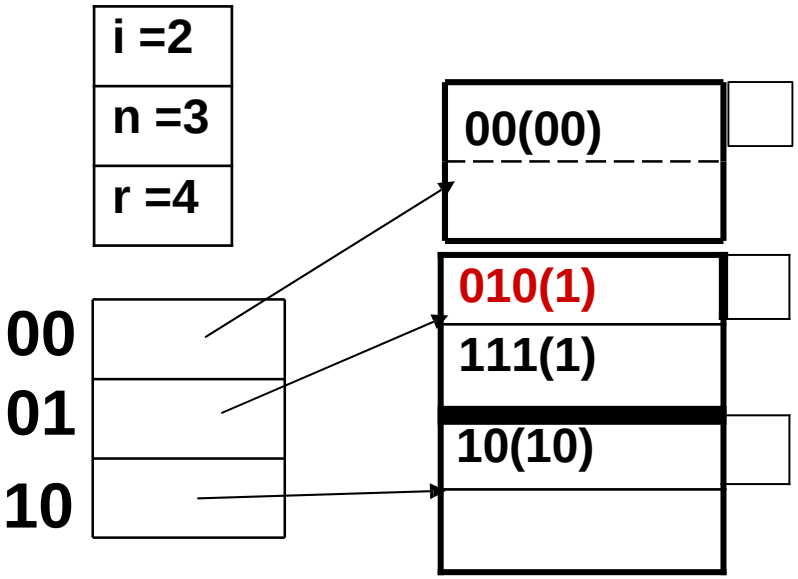
i-当前使用的散列函数的位数
n-当前的桶数
r-当前散列表中的记录总数
要求 r<=1.7n



i-当前使用的散列函数的位数
n-当前的桶数
r-当前散列表中的记录总数
要求 r<=1.7n

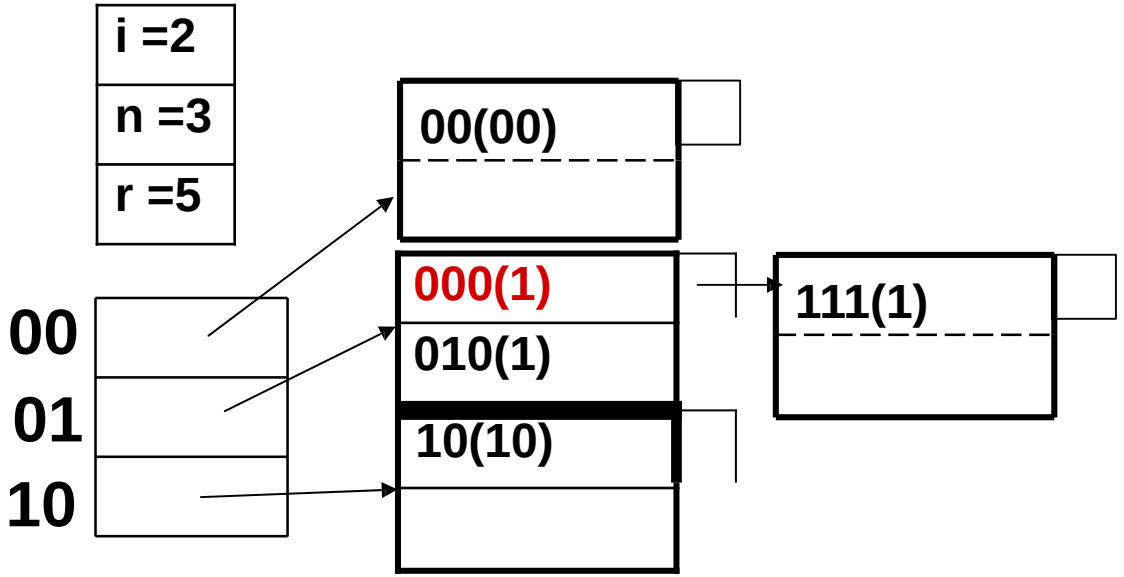
- 插入 0101的记录,如右图。
- 此时r/n=4/2=2>1.7, 桶数需要增1, 即要增加一桶,n=3
- 需要分裂。10桶要由00桶分裂而来(低位相同而最高位不同的那一个桶)

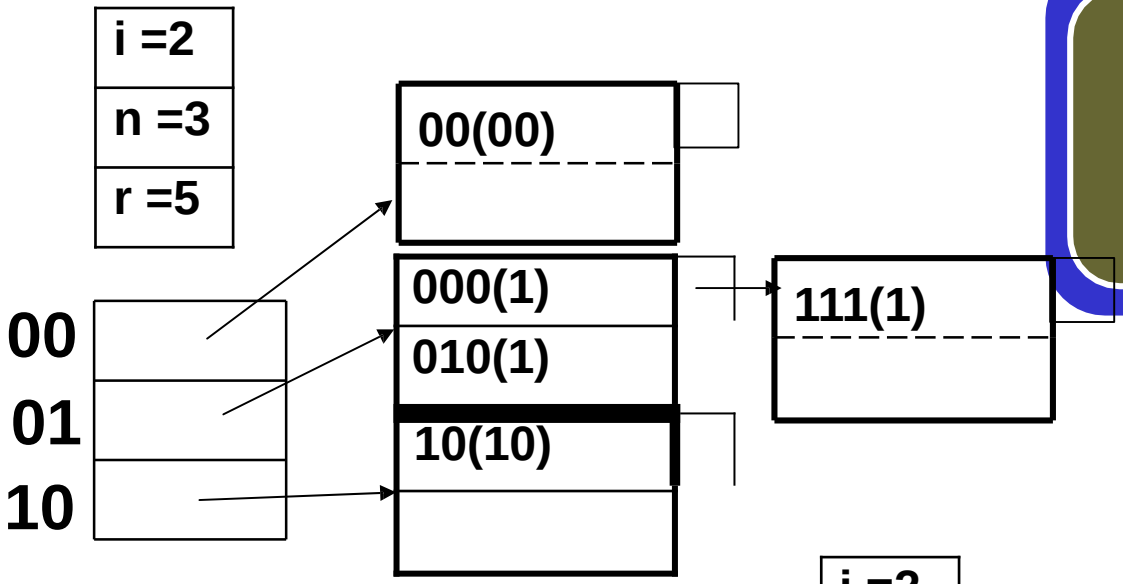




i-当前使用的散列函数的位数
n-当前的桶数
r-当前散列表中的记录总数
要求 r<=1.7n

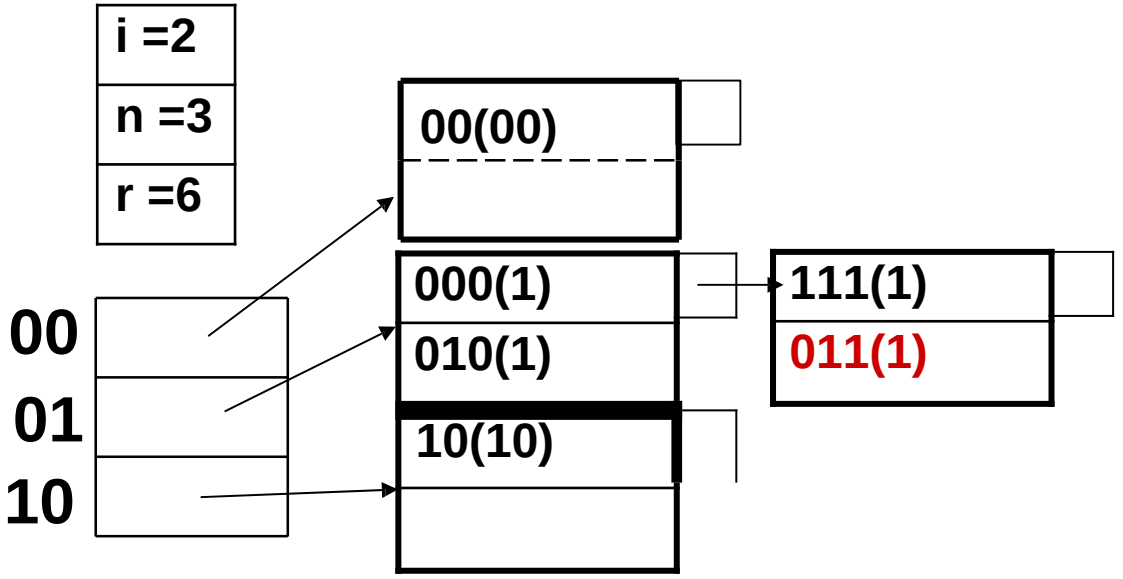
- 插入 0001的记录,如右图。
- 此时r/n=5/3=1.66<1.7, 桶数无需增加

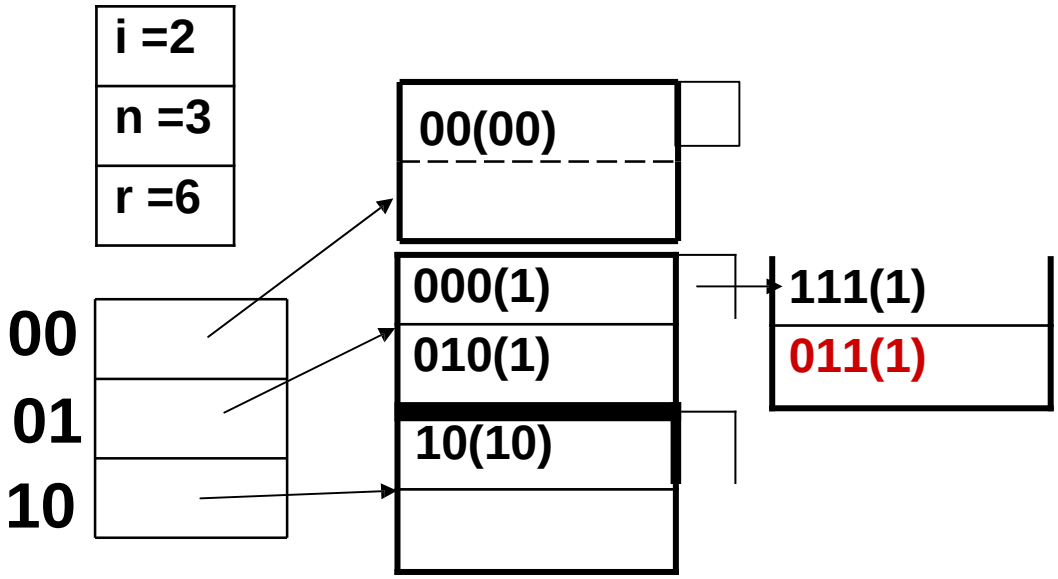




i-当前使用的散列函数的位数
n-当前的桶数
r-当前散列表中的记录总数
要求 r<=1.7n

- 插入 0111的记录,如右图。
- 因没有11桶，则插入01桶。
- 此时r/n=6/3=2>1.7, 桶数需要增1, 即要增加一桶11
- 需要分裂。11要由01桶分裂而来(低位相同而最高位不同的那一个桶)

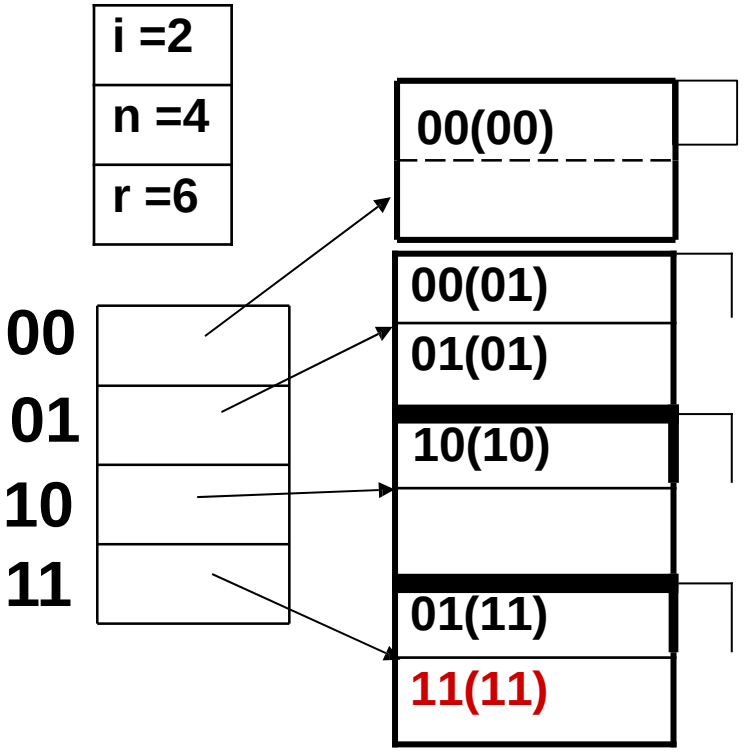


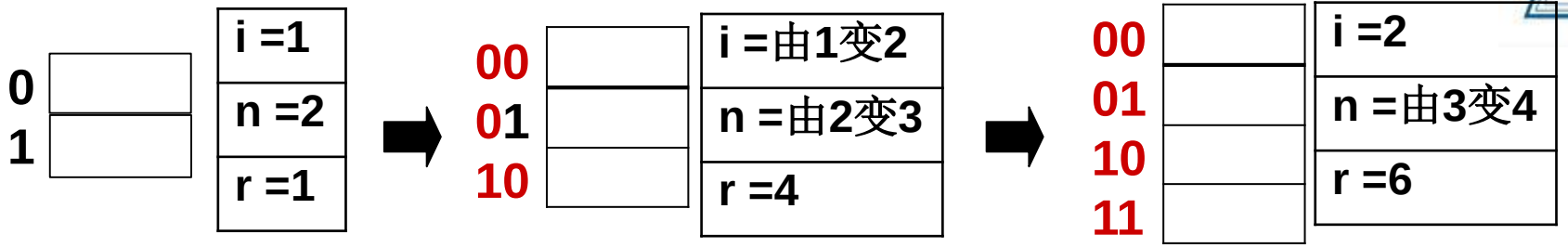


i-当前使用的散列函数的位数
n-当前的桶数
r-当前散列表中的记录总数
要求 r<=1.7n

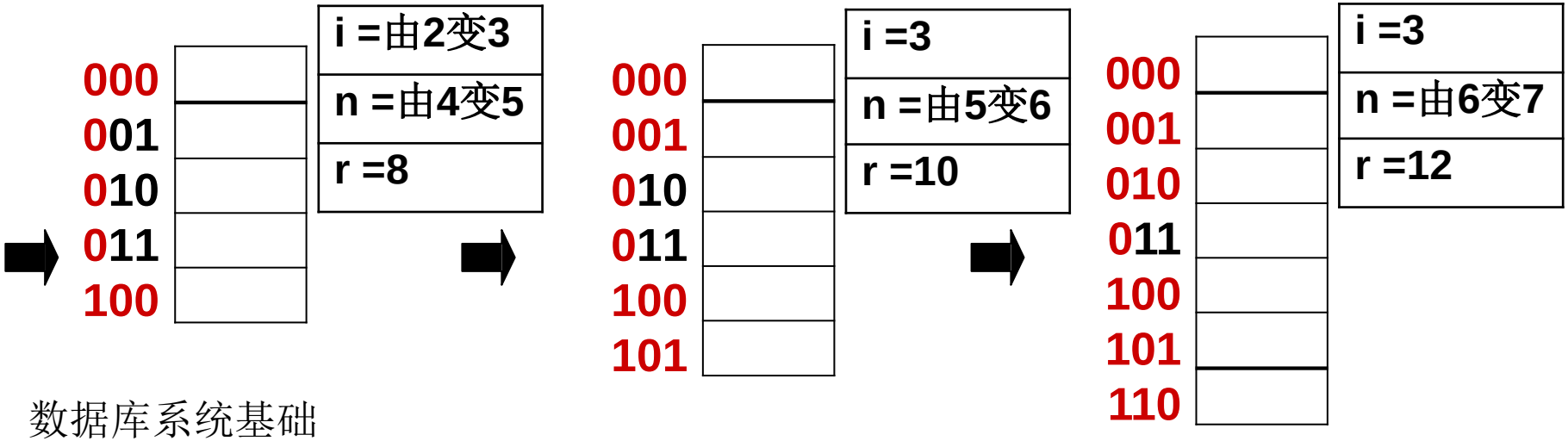
分裂前

分裂后

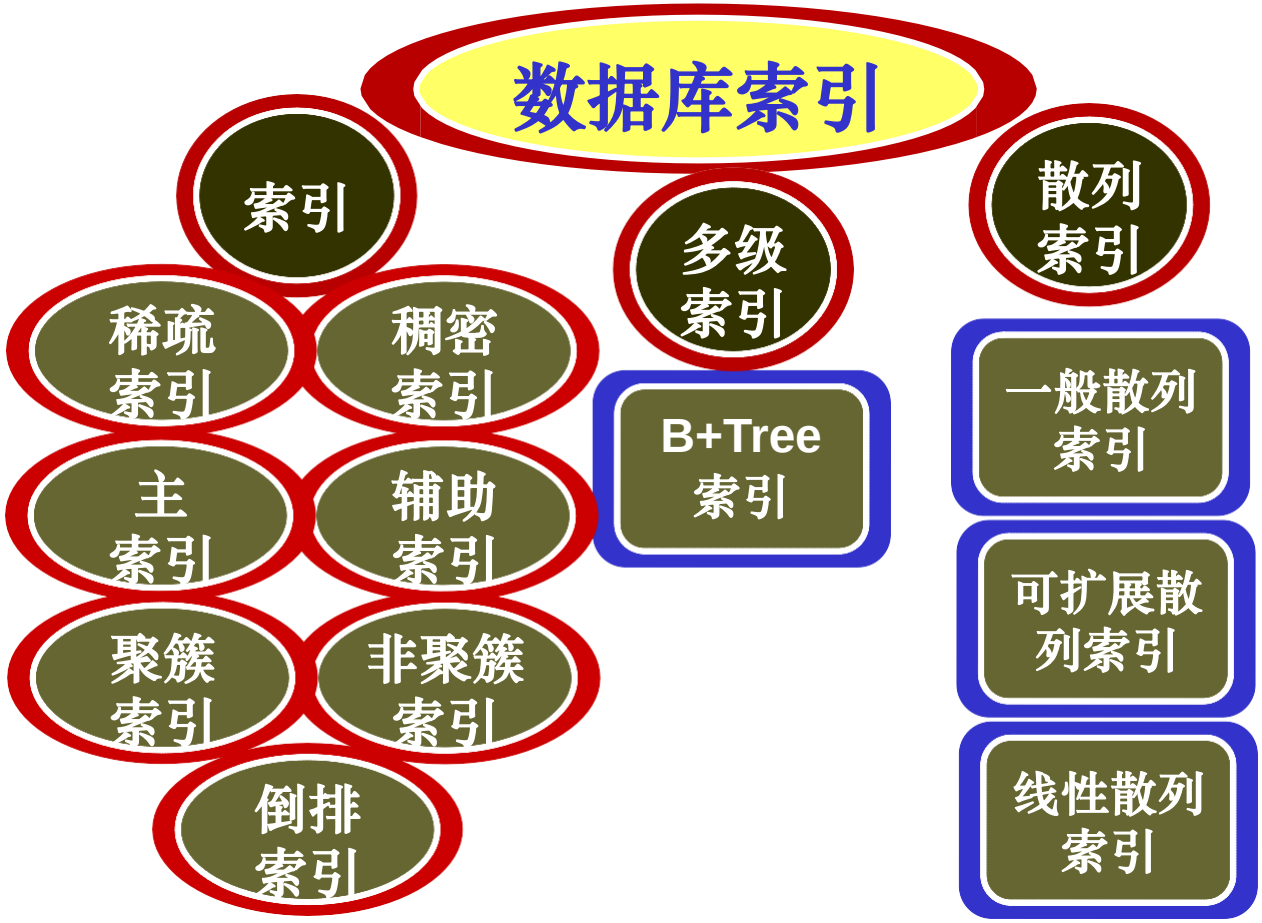




- 键值由低位向高位次序使用。对应索引项存储在由桶数组指针指向的存储块
- 当r/n(当前索引项数除以桶数)大于一定比例时，按次序增加1个桶
- 假设该桶为$1a_2a_3\ldots a_i$，则该桶由$0a_2a_3..a_i$对应的块分裂而来
- 当桶数超过2^i个桶时，则使 i增1。



回顾本讲学习了什么?


数据库索引
索引
稀疏索引
稠密索引
主索引
辅助索引
聚簇索引
非聚簇索引
倒排索引
多级索引
B+Tree索引
散列索引
一般散列索引
可扩展散列索引
线性散列索引