

第7章

数据库的物理组织

数据库的物理组织

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.1 概论

➤ 研究目的

- 提高存储效率，加快存取速度

➤ 研究内容

- 磁盘的物理结构
- 文件的组织与设计

➤ 主要技术

- 索引技术
- 散列技术

数据库的物理组织

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.2 数据库的物理存储介质

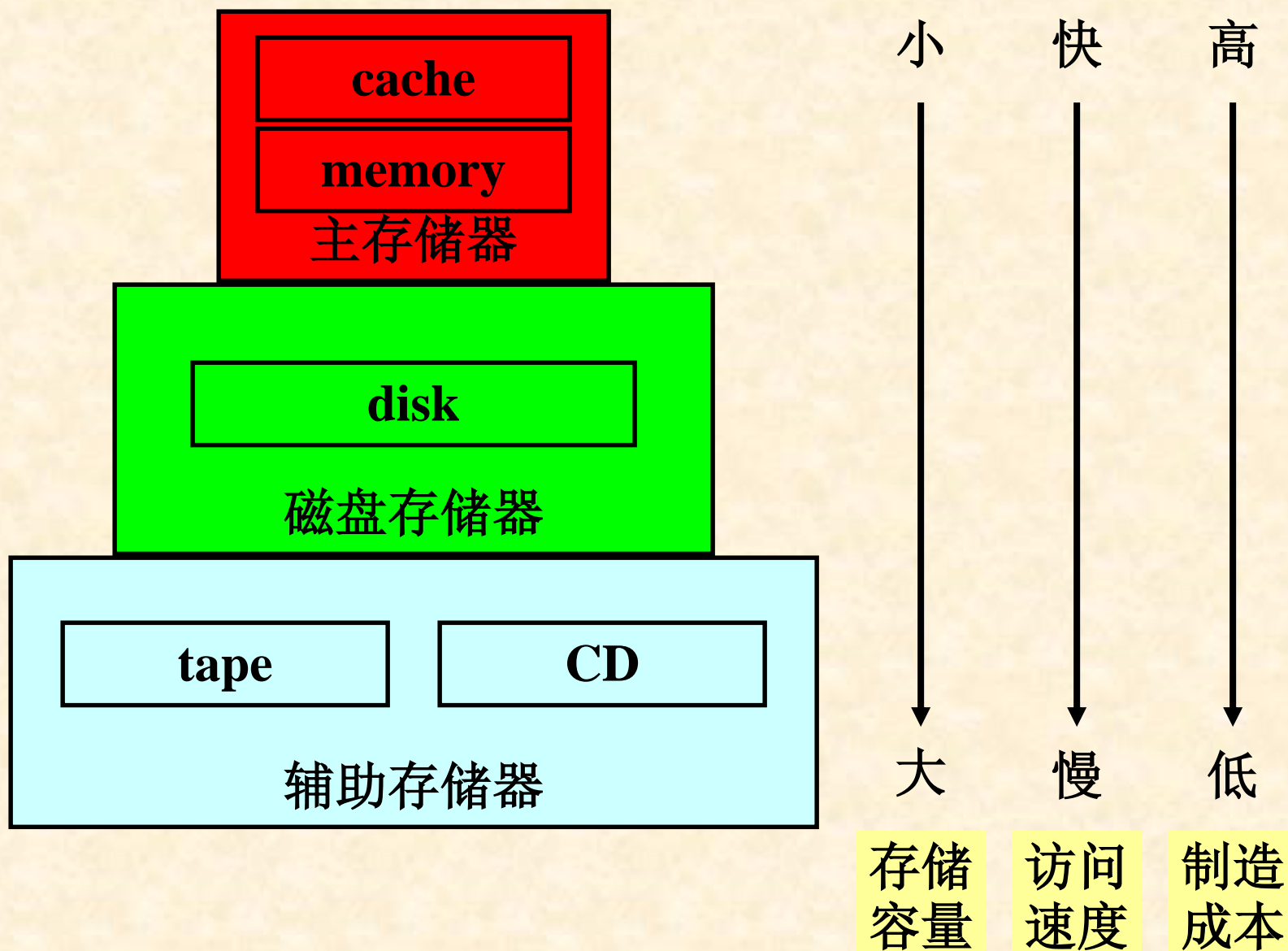
➤ 三级存储器结构

- 第一级：主存储器（main memory）
 - ✓ 包括：
 - 高速缓冲存储器（cache）
 - 主存储器（memory）
- 第二级：磁盘存储器（secondary storage）
 - ✓ 也称为：二级存储器或次级存储器
- 第三级：辅助存储器（tertiary storage）
 - ✓ 包括：
 - 磁带存储器
 - 自动光盘机
 - ✓ 是一种辅助存储设备，也称三级存储器

7.2 数据库的物理存储介质

	存储容量	访问速度	访问类型	存取单位
第一级 存储器	100MB ~ 10GB	10^{-8} 秒 ~ 10^{-7} 秒	随机	字节
第二级 存储器	10GB ~ 20^3 GB	10毫秒 ~ 30 毫秒	随机	磁盘块
第三级 存储器	10^6 GB	几秒钟 ~ 几分钟	顺序	数据块

7.2 数据库的物理存储介质



数据库的物理组织

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.3 磁盘存储器及其结构

➤ 磁盘存储器

- 一种大容量、可以直接存取的外部存储设备
 - ❖ 大容量：10GB ~ 2000GB
 - ❖ 直接存取：可以随机到达磁盘上的任何一个部位存取数据

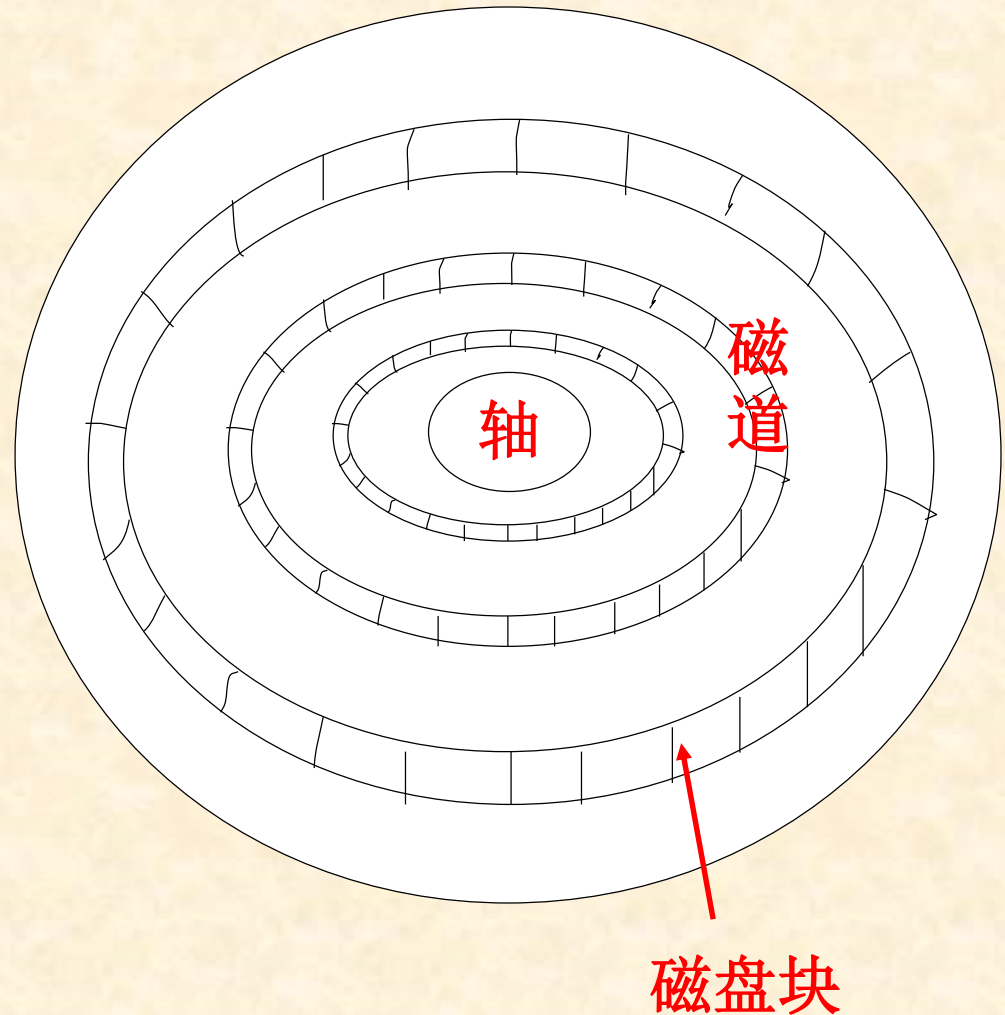
➤ 磁盘存储器的组成

- 磁盘盘片
- 磁盘驱动器

7.3 磁盘存储器及其结构

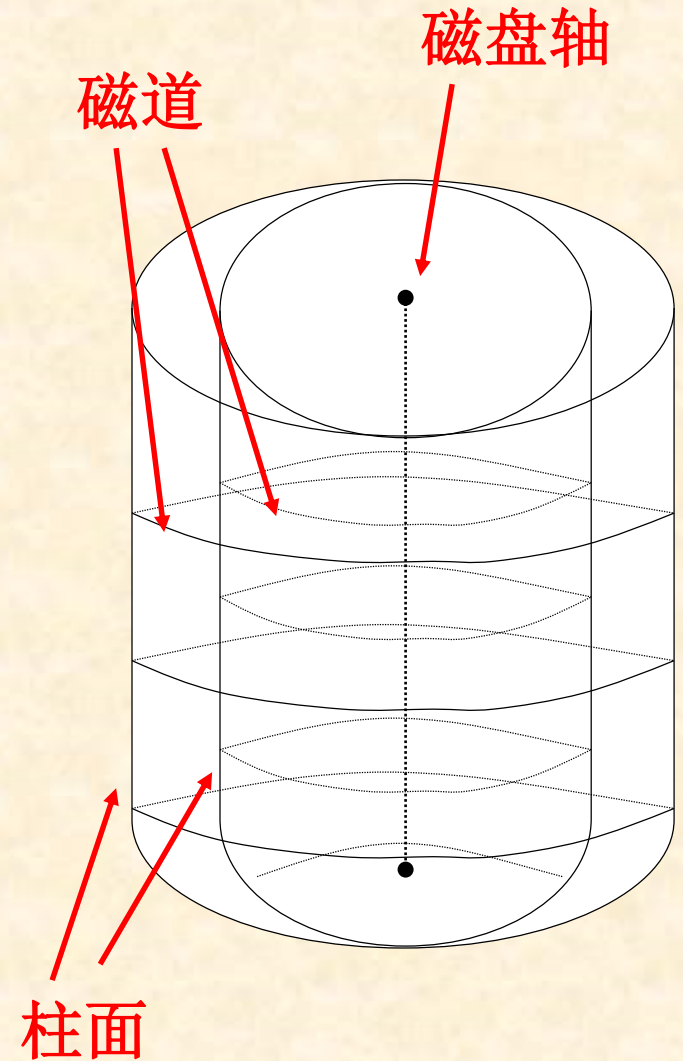
1. 磁盘盘片

- 盘片分上下两面
- 每一面又被划分成若干个磁道（**track**）
- 每个磁道是一个由两个半径不等的同心圆所构成的区域
- 每个磁道又分为若干个等长的扇区（**sector**），又称磁盘块（**block**）
- 磁盘块是磁盘与内存进行数据交换的基本单位



7.3 磁盘存储器及其结构

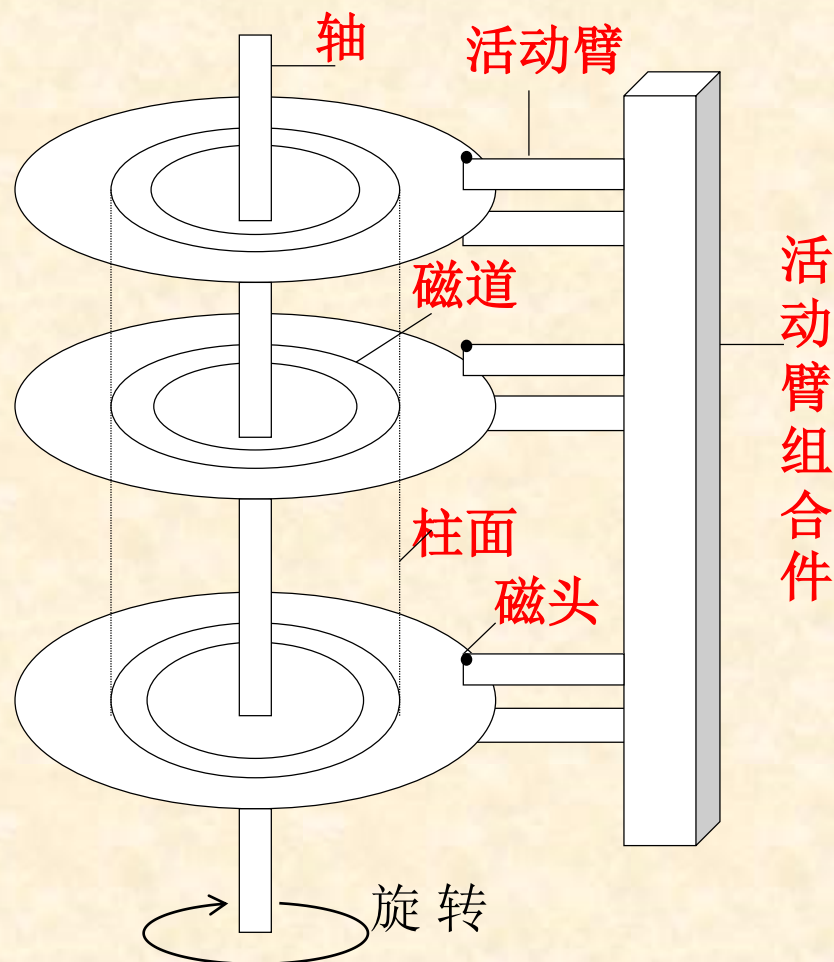
- 一个磁盘存储器往往由若干个盘片组成一个盘片组，固定在同一个主轴上
- 由具有相同半径的若干个磁道可以构成一个无形的同心圆柱体，我们称其为柱面（cylinder）
- 磁盘盘片的每一面有多少条磁道，该磁盘存储器就有多少个柱面



7.3 磁盘存储器及其结构

2. 磁盘驱动器

- 磁盘驱动器由活动臂和读写头组成
- 每个盘片有两个活动臂，分别对应上、下两面，每个活动臂的尽头有一个读/写头（或称磁头），用它读取/写盘片中的数据
- 所有的活动臂都被固定在一个活动臂组合件上，并可以通过移动活动臂组合件来使固定在活动臂上的磁头在不同的磁道之间移动，从而完成对于柱面的定位操作



7.3 磁盘存储器及其结构

3. 磁盘存储器

- 一个磁盘存储器是由盘片组以及磁盘驱动器所组成的，其中盘片组以轴为核心作不间断的旋转，而活动臂组合件则以柱面为单位做前进或后退操作
- 磁盘块（扇区）的定位操作
 - ❖ 选择柱面：通过移动活动臂组合件来进行定位
 - ❖ 选择磁道：选择活动臂（读/写头）
 - ❖ 选择磁盘块：根据盘片组的旋转定位
- 因此，每个磁盘块的物理地址由三个部分组成：
柱面号 + 磁道号 + 磁盘块号

7.3 磁盘存储器及其结构

4. 磁盘存储器的I/O操作

- **编码方式**：设一个磁盘存储器有n个柱面，每个柱面有m个磁道，每个磁道有r个磁盘块。则编码规则如下：
 - ❖ **柱面号**：由外层到内层分别为（0号柱面，1号柱面，……，n-1号柱面）
 - ❖ **磁道号**：每个柱面中的磁道从上到下分别为（0号磁道，1号磁道，……，m-1号磁道）
 - ❖ **磁盘块号**：每个磁道中的磁盘块按照旋转的方向分别为（0号磁盘块，1号磁盘块，……，r-1号磁盘块）
- 因此，该磁盘存储器中共有m*n*r个磁盘块，其中x号柱面的y号磁道的z号磁盘块的编号是：

$$x * m * r + y * r + z$$

7.3 磁盘存储器及其结构

4. 磁盘存储器的I/O操作（续）

- 磁盘的格式化

- ✓ 在每个磁盘块的头部写入：

- 该磁盘块的地址，包括：柱面号，磁道号（读/写头号），磁盘块号
 - 有关该磁盘块的状态信息

- ✓ 在同一个磁道的相邻磁盘块之间会留有一定的空隙，以利于对相邻磁盘块的顺序访问

- 磁盘的I/O操作：首先根据给出的磁盘块地址定位，然后读/写指定磁盘块上的数据，其时间开销是：

- 活动臂组合件的移动时间
 - 磁盘片的旋转定位时间
 - 读/写数据时间

固态硬盘

- **固态硬盘**（Solid State Disk、簡稱**SSD**，准确的技术称呼应为**固态驱动器**）是一种基于永久性存储器，如闪存，或非永久性存储器，同步动态随机存取存储器（**SDRAM**）的计算机外部存储设备。固态硬盘用来在便携式计算机中代替常规硬盘。虽然在固态硬盘中已经没有可以旋转的盘状机构，但是依照人们的命名习惯，这类存储器仍然被称为“硬盘”
- 和常规硬盘相比，固态硬盘具有低功耗、无噪音、抗震动、低热量的特点。这些特点不仅使得数据能更加安全地得到保存，而且也延长了靠电池供电的设备的连续运转时间
- 目前固态硬盘普及的最大问题仍然是成本和写入次数

数据库的物理组织

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.4 文件组织

➤ 数据库中的数据被组织成若干个数据文件

- 文件是记录的集合，记录是项的集合

❖ 记录的‘型’与‘值’

- 一个记录中所含有的项的描述信息被称为记录的型
- 记录值则是符合记录型要求的项值的集合，通常也简称为记录

❖ ‘记录’是文件中的一个逻辑单位，在实现时‘记录’必须被分配存储到具体磁盘块中去，从而构成一条物理记录（通常也简称为记录）

❖ 记录的大小往往与磁盘块的大小不一致，因此记录在磁盘块上的存储可以有多种方式

7.4 文件组织

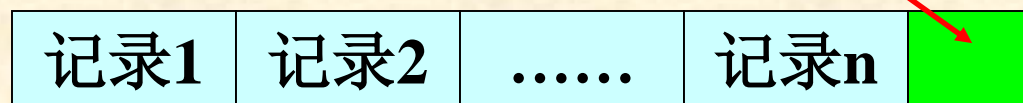
➤ 记录在磁盘块中的分配方式

- 单块单记录



无用部分

- 单块多记录

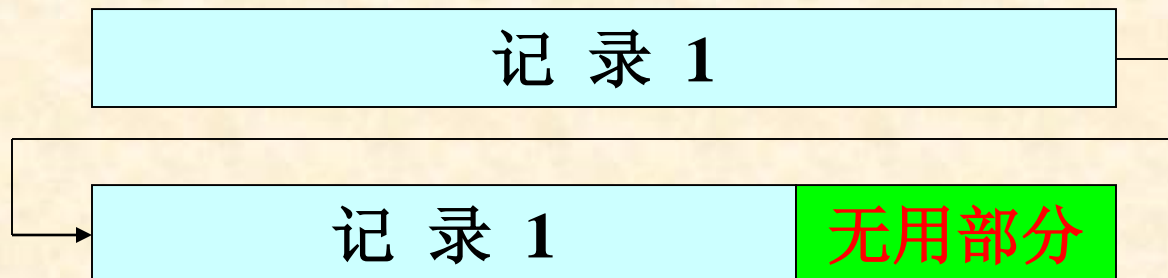


特点：在上述两种存储方式中，记录不能跨块存储，都存在存储空间浪费现象

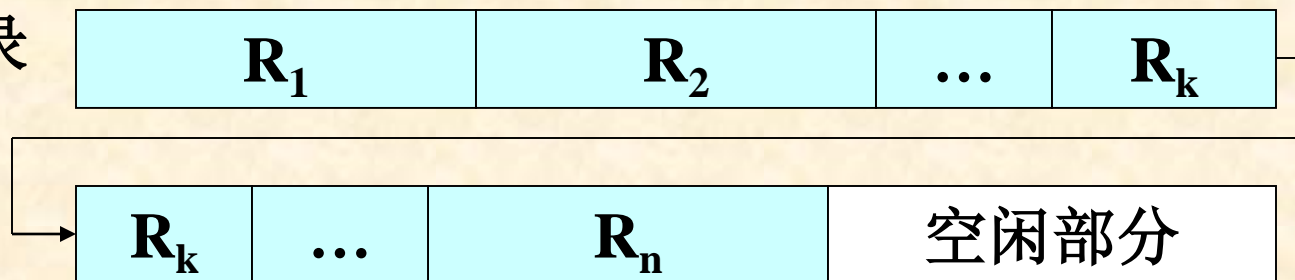
7.4 文件组织

➤ 记录在磁盘块中的分配方式（续）

▪ 多块单记录



▪ 多块多记录



特点：在上述两种存储方式中，允许记录跨块存储。在多块单记录方式下仍然有空间浪费现象，在多块多记录方式中则不存在空间浪费现象

7.4 文件组织

➤ 不同类型的应用可以采用不同的记录分配方式。
较常用的有两种方式：

- 单块多记录

- ❖ 当记录的长度远远小于磁盘块的大小时

- 多块单记录

- ❖ 当记录的长度接近或大于磁盘块的大小时

7.4 文件组织

➤ 磁盘块在磁盘上的分配

- 数据库系统使用的磁盘块有两种申请与使用方式:

1. 由文件系统负责磁盘块的申请与分配

- 采用文件系统中的数据文件来负责数据库中数据的存储，文件所需要的磁盘空间由文件系统来申请和分配使用
- 在此种方式下不存在真正意义上的磁盘管理

2. 由DBMS负责磁盘块的申请与分配

- DBMS在一开始就申请所有供数据库系统使用的磁盘块，并负责这些磁盘块的分配与使用管理

7.4 文件组织

➤ 常用的磁盘块的分配方式

	分配方式	优点	缺点
连续分配法	按连续物理地址分配	有利于文件的顺序访问	无法扩充文件所使用的存储空间
链接分配法	随机分配，磁盘块之间通过指针链接在一起	有利于文件存储空间的扩充	存取效率较差
索引分配法	通过一个逻辑块号与磁盘块地址之间的索引来维护所使用的磁盘块	空间的使用较灵活	索引文件需要额外的存储空间
集簇分配法	局部是物理上连续的，又可以通过指针链接新的磁盘块	综合了连续分配法和链接分配法的优点	

7.4 文件组织

➤ 文件中的定长记录与变长记录

- 在数据库系统中，文件中的记录有定长与变长两种组织方式

❖ **定长记录：**所有属性值都是定长的，即占用固定大小的存储空间

❖ **变长记录：**记录占用的存储空间是不断变化的。变长记录的组织方式主要适合于下述几种情况：

- 数据项所占用的存储空间大小变化较大
- 含有需要重复存储的字段值
- 含有数据量极大的字段值
- 含有可变格式的记录

7.4 文件组织

➤ 定长记录组织方式

在**dBASE**、**ACCESS**等桌面数据库系统中，一般采用定长记录、单块多记录的组织方式，而对于其中某些特殊的可变长字段，则单独组织成一个数据存储文件，并采用多块单记录的组织方式。在这类数据库系统中，一个关系对应着一个**.dbf**文件，其数据文件的结构如下：

- 第**0**号字节：状态标志S，（S%16）为系统的版本号，（S/16）表示有无备注字段
- 第**1-3**号字节：年、月、日，表示数据文件的访问时间
- 第**4-7**号字节：该文件中的记录总数
- 第**8-9**号字节：文件头的长度，即记录存储的起始地址
- 第**10-11**号字节：记录长度
- 第**12-31**号字节：未使用

7.4 文件组织

➤ 定长记录组织方式（续）

从第**32**个字节开始记录关系的定义信息，即关系模式。
每个属性的定义信息占**32**个字节，其结构如下：

- 第**0-9**个字节：字段名
- 第**10**个字节：ASCII码**0**，字段名的结束标志
- 第**11**个字节：字段类型（**C**表示字符类型，**D**表示日期类型，**L**表示逻辑类型，即逻辑真或逻辑假，**N**表示数值类型，**M**表示备注字段，即字符型的变长字段）
- 第**12-15**个字节：工作区中当前记录在该字段上的值的指针，通过该指针可以直接访问到当前记录在该字段上的值
- 第**16**个字节：字段的长度
- 第**17**个字节：小数部分的长度，只用于数值型字段
- 第**18-31**个字节：未使用

7.4 文件组织

➤ 变长记录组织方式

- 在文件中的记录有时其长度是可以变化的，如一个文件中允许存在不同记录型的记录，或允许记录型记录可以是变长的，以及记录中某些项是变长的等
- 变长记录的表示有两种形式：

❖ 变长记录的字节串表示形式

- 将每个记录看成连续字节串，然后在每个记录尾部附加特殊的标志符叫“记录尾标识符”，用以区别不同的记录

❖ 变长记录的定长表示形式

- 在文件中往往使用一个或多个定长记录来表示变长记录的方法。具体实现时有两种技术：
 - » 预留空间
 - » 指针技术

7.4 文件组织

➤ 变长记录组织方式（续）

- 这里以我们自己研制的面向对象数据库管理系统 **OMNIX**（或关系数据库管理系统 **CBase**）为例来介绍变长记录的组织方式。
- 在该数据库系统中，我们将记录中的字段分为常规字段和超长字段两大类。

❖ 超长字段

- 此种类型字段的属性值可能占用大量的存储空间，因此我们参照定长记录中的备注字段的组织方法，采用**多块多记录**的磁盘分配方法。

❖ 常规字段

- 采用**变长记录、多块多记录**的磁盘分配方法。

7.4 文件组织

➤ 变长记录组织方式（续）

- 有关记录模式等描述信息保存在数据字典中，数据文件只用于保存用户的数据主体及根据对象标识符**OID**（或记录号）所建立的索引信息。
- 数据文件中的磁盘块分为索引块和数据块两类，索引块采用**B+**树的组织方式，而数据块的组织结构如下：
 - ❖ 块头信息：16个字节
 - ❖ 块内的记录索引信息：每条记录占24个字节
 - ❖ 记录值：变长记录

7.4 文件组织

➤ OMNIX数据库的数据（磁盘）块的组织结构

- 块头：磁盘块的头部，占**16**个字节
 - 第**0**至**3**号字节：下一个数据块的地址
 - 第**4**至**7**号字节：当前数据块中的记录数
 - 第**8**至**11**号字节：当前数据块中剩余空间的大小
 - 第**12**至**15**号字节：剩余空间的尾地址

- 记录索引信息：从第**16**号字节开始保存每条记录在当前数据块内的索引定位信息，每条记录占**24**个字节。
 - 第**0**至**15**个字节：对象标识符**OID**（或记录号）
 - 第**16**至**19**个字节：记录的起始位置（在当前磁盘块内的偏移量）
 - 第**20**至**23**个字节：记录在当前数据块内的存储长度

7.4 文件组织

➤ OMNIX数据库的变长记录

- 变长记录

记录长度	字段1	字段2
------	-----	-----	-------

- 其中：

- 记录长度

- ✓ 占4个字节，用于存放该记录的总长度（物理记录所占用的存储空间大小，包括用于记录的物理存储组织的一些附加信息）

- 字段

- ✓ 用于存放属性值

7.4 文件组织

➤ OMNIX数据库的变长记录 – 字段结构

字段长度	字段值
------	-----

■ 其中：

❖ 字段长度（length）

- 占4个字节，用于记录该字段值的总长度。当记录长度为-1时，表示该字段上的值为空值（NULL）。
- 空字段值是不占用额外的存储空间的。

❖ 字段值（value）

- 用于存放属性值，根据不同类型的属性，其字段值的组织结构是不一样的。

7.4 文件组织

➤ OMNIX数据库的变长记录 – 字段值的存储

❖ 普通属性

- 即单值的常规属性，字段值即属性值。

❖ 长字段属性

- 该字段值是一个8个字节的定长字段，用于存放该属性值的存储地址（段地址和段内的偏移量）。

❖ 集合属性

- 该字段值又分为两部分
 - 元素个数
 - 各个元素的值

元素个数	元素1	元素2	元素n
------	-----	-----	-------	-----

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.5 文件记录组织

➤ 堆文件组织 (Heap file)

- 记录可以放在文件的任何位置上，一般以记录录入时间先后为序组织存储

➤ 顺序文件组织 (Sequential file)

- 记录是按其项值的升序或降序存储的

➤ 散列文件组织 (Hashing file)

- 根据记录中的某个项值通过散列函数求得的值作为记录的存储地址

➤ 聚集文件组织 (Clustering file)

- 一个文件可以存储多个关系的记录，不同关系中有联系的记录存储在一起可以提高查找速度

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.6 索引技术与散列技术

- 在数据库中数据查找的速度是至关重要的，但是当数据库中数据量增大时，数据查找速度就会受到影响，当数据量极大时，查找速度将会受到严重影响，为解决此问题需引入“索引技术”与“散列技术”

- 内容

- 顺序文件
- 索引文件
 - ❖ 在顺序文件上的索引技术
 - 稠密索引、稀疏索引、多级索引
 - ❖ 非顺序文件中的索引技术
 - ❖ 多维索引
- B/B+树文件
- HASH文件

7.6 索引技术与散列技术

一、顺序文件

- 按照某个属性(项)的取值进行排序而构成的数据文件被称为**顺序文件**

- 例（假设每个磁盘块可以存放**2**条记录）

1	记录 1	}	第 1 个磁盘块
2	记录 2		
3	记录 3	}	第 2 个磁盘块
4	记录 4		
5	记录 5	}	第 3 个磁盘块
6	记录 6		
⋮	⋮		⋮

主关键字值

7.6 索引技术与散列技术 - 顺序文件（续）

➤ [例1]

- 设一个关系有 10^6 个元组，在每个磁盘块（大小为4KB）中可存放10个这样的元组，则该关系大约占用：

10^5 个磁盘块（约400MB）

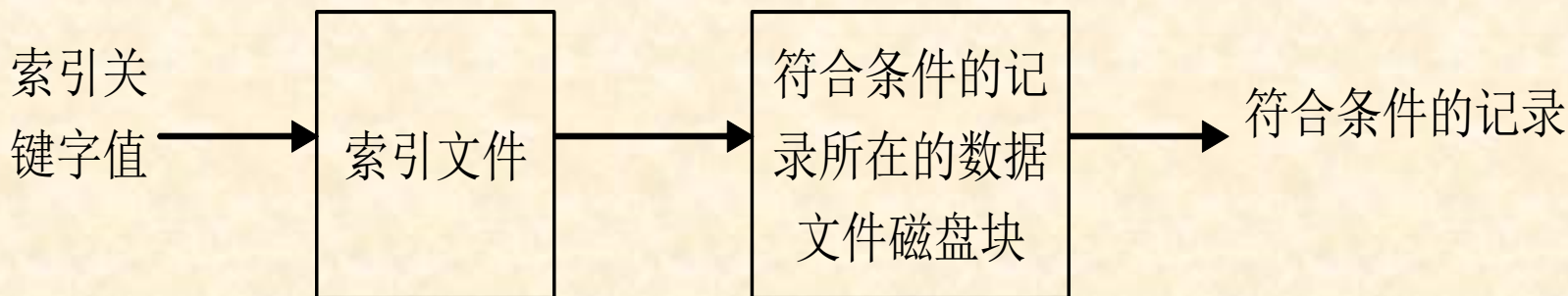
- 设该关系中的元组已经按照主关键字值从小到大的顺序构成了一个顺序文件，因此可以采用二分查找法来根据主关键字值进行记录定位
- 按照一个主关键字值进行记录定位所需要的磁盘I/O次数为：

$$\log_2 10^5 \approx 16.6 \approx 17$$

7.6 索引技术与散列技术

二、索引文件

- 建立索引文件的目的是：以记录的特征值（通常是一个或多个字段的值）为输入，能够快速找出具有该特征的记录
- 利用索引文件进行记录查找的过程：首先在索引文件中按照特征字段的值进行查找，找出具有该特征的记录的记录指针（即记录在数据文件中的存储地址），从而可以在数据文件中进行直接定位，读出所需要的记录



利用索引文件访问数据文件中的记录的示意图

7.6 索引技术与散列技术 - 索引文件（续）

- 通过建立索引文件可以大大提高在数据文件上的记录查找定位速度

❖ 索引文件的大小一般都大大小于数据文件的大小

- 索引文件主要用于记录的快速定位操作，在索引文件中一般只含有特征字段（即索引属性）上的值和记录指针。如：

属性值1	对应记录1的指针
属性值2	对应记录2的指针
属性值3	对应记录3的指针
.....

❖ 索引文件采用便于查找的特殊组织结构（如B或B+树）

- 在不同类型的数据文件上我们可以建立不同的索引文件

7.6 索引技术与散列技术

1. 在顺序文件上的索引

- 稠密索引
- 稀疏索引
- 多级索引

❖ 在讨论上述的三种索引文件时，我们假设其索引属性的值具有唯一性。否则我们可以使用：

- 具有重复键值的索引

7.6 索引技术与散列技术

➤ 稠密索引

■ 索引文件

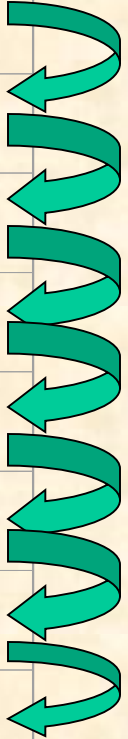
- ❖ 存放记录的关键字值以及指向记录本身的指针（记录的存储地址），并且按照关键字值的顺序进行排序
- ❖ 一个关键字值和一个记录指针构成的 **键-指针对** 我们称为一个 **索引项**：

关键字K	记录指针P
------	-------

■ 稠密索引

- ❖ 如果数据文件中的每条记录在索引文件中都存在一个相对应的索引项，则该索引被成为 **稠密索引**
- ❖ 例如：

稠密索引(续)

S [#]	指针	S [#]	SN	SD	SA	
S ₁	● →	S ₁	LU	CS	18	
S ₂	● →	S ₂	LI	CS	17	
S ₃	● →	S ₃	XU	MA	18	
S ₄	● →	S ₄	LO	CS	18	
S ₅	● →	S ₅	LIN	PH	19	
S ₆	● →	S ₆	WANG	CS	17	
S ₇	● →	S ₇	SEN	MA	17	
S ₈	● →	S ₈	EHEN	PH	18	

稠密索引

数据文件（主文件）

稠密索引(续)

- 利用稠密索引进行数据查找要比直接在数据文件上进行数据查找的速度快。其原因是：
 - 索引文件使用的磁盘块比数据文件的少，因此磁盘I/O的时间开销小
 - ❖ 一般情况下，一个磁盘块中可以存放的索引项的个数要远远大于可以存放的记录数
 - 索引文件中的索引项被按照索引关键字的值进行了排序，因此在索引文件中可采用二分查找法来提高查找速度
 - ❖ 这在无序的数据文件(堆文件)中是无法做到的
 - 索引文件可能足够小，可以放在内存中操作，从而不必访问磁盘

➤ 利用稠密索引查找关键字值为K的记录算法如下：

- 1) 采用二分查找法在索引文件中查找是否存在关键字值为K的索引项；
- 2) 如果不存在相关的索引项，则表示关键字值为K的记录不存在，查找失败。否则：
- 3) 根据找到的索引项中的记录指针到数据文件中进行直接定位，读取相应的记录

➤ [例2]

- 为[例1]中的顺序数据文件建立一个稠密索引。假设每个磁盘块可以存放**100**个索引项，则该稠密索引共有 **10^6** 个索引项，需要占用：

10^4 个磁盘块（约40MB）

- 利用该稠密索引进行记录定位需要的磁盘I/O次数为：

$$\log_2 10^4 + 1 \approx 13.3 + 1 \approx 15$$

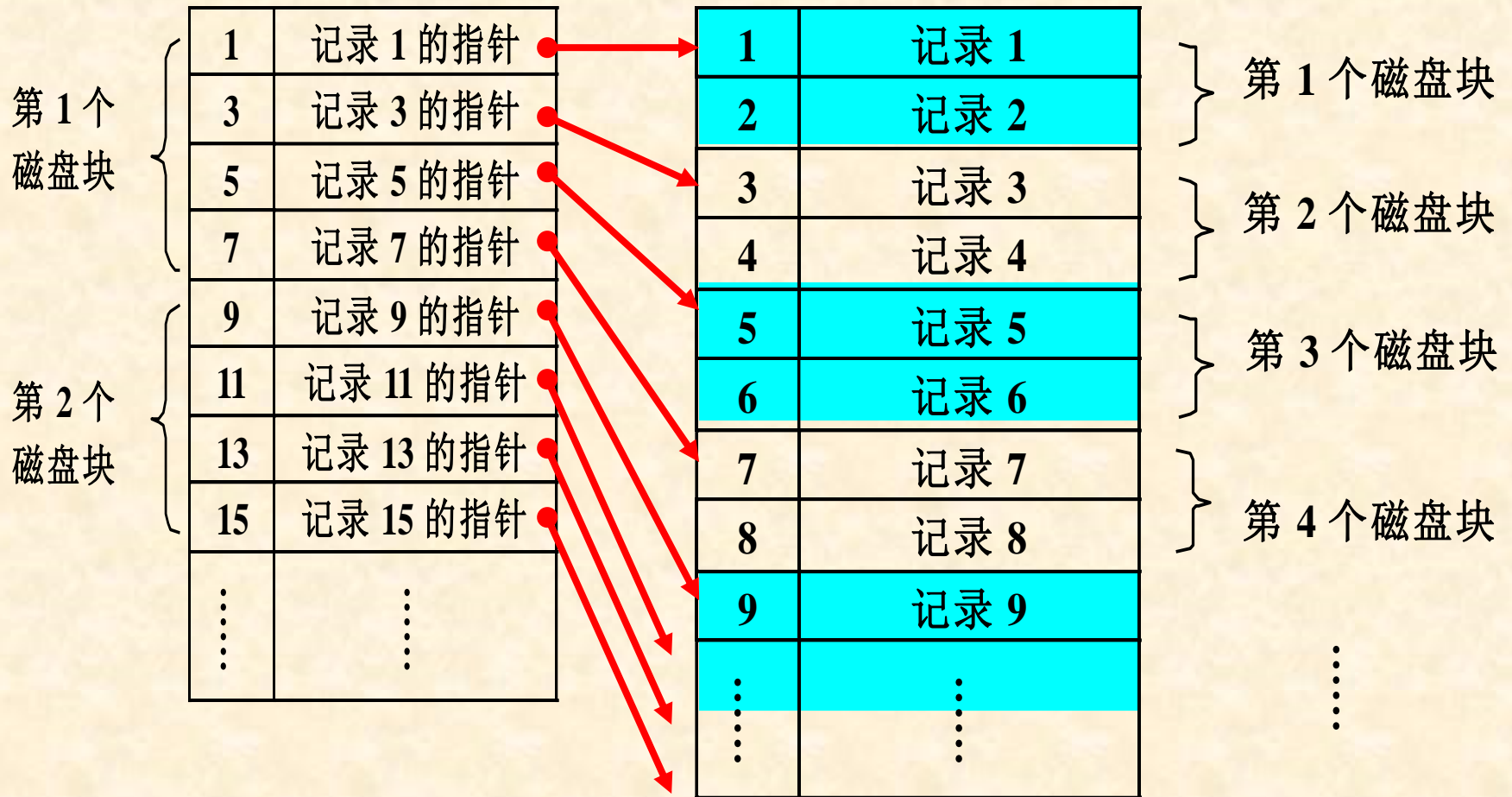
❖ 其中的1是访问数据文件的磁盘I/O

7.6 索引技术与散列技术

➤ 稀疏索引

- 如果数据文件是顺序文件，我们可以在索引文件中只为数据文件的每个磁盘块设一个索引项，记录该磁盘块中第一条数据记录的关键字值及该磁盘块的首地址
- 这样建立起来的索引文件被称为**稀疏索引**

稀疏索引(续)



稀疏索引

数据文件 (主文件)

假设每个磁盘块可以存放2条数据记录或4条索引项

稀疏索引(续)

➤ 利用稀疏索引查找关键字值为K的记录的算法如下：

- 1) 采用二分查找法在索引文件中查找关键字值小于或等于K，且最接近K的索引项；
- 2) 如果不存在相关的索引项，则表示关键字值为K的记录不存在(所有记录的索引关键字的值均大于K)，查找失败。否则：
- 3) 根据找到的索引项中的记录指针到数据文件中读取相应的磁盘块 D；
- 4) 在磁盘块 D 中利用二分查找法查找关键字值为K的记录

➤ [例3]

- 为例1中的顺序数据文件再建立一个稀疏索引。由于该数据文件共占用 10^5 个磁盘块，因此稀疏索引文件中有 10^5 个索引项，需要占用：

❖ 10^3 个磁盘块（约4MB）

- 利用该稀疏索引进行记录定位需要的磁盘I/O次数为：

❖ $\log_2 10^3 + 1 \approx 9.97 + 1 \approx 11$

➤ 稠密索引 与 稀疏索引 的区别

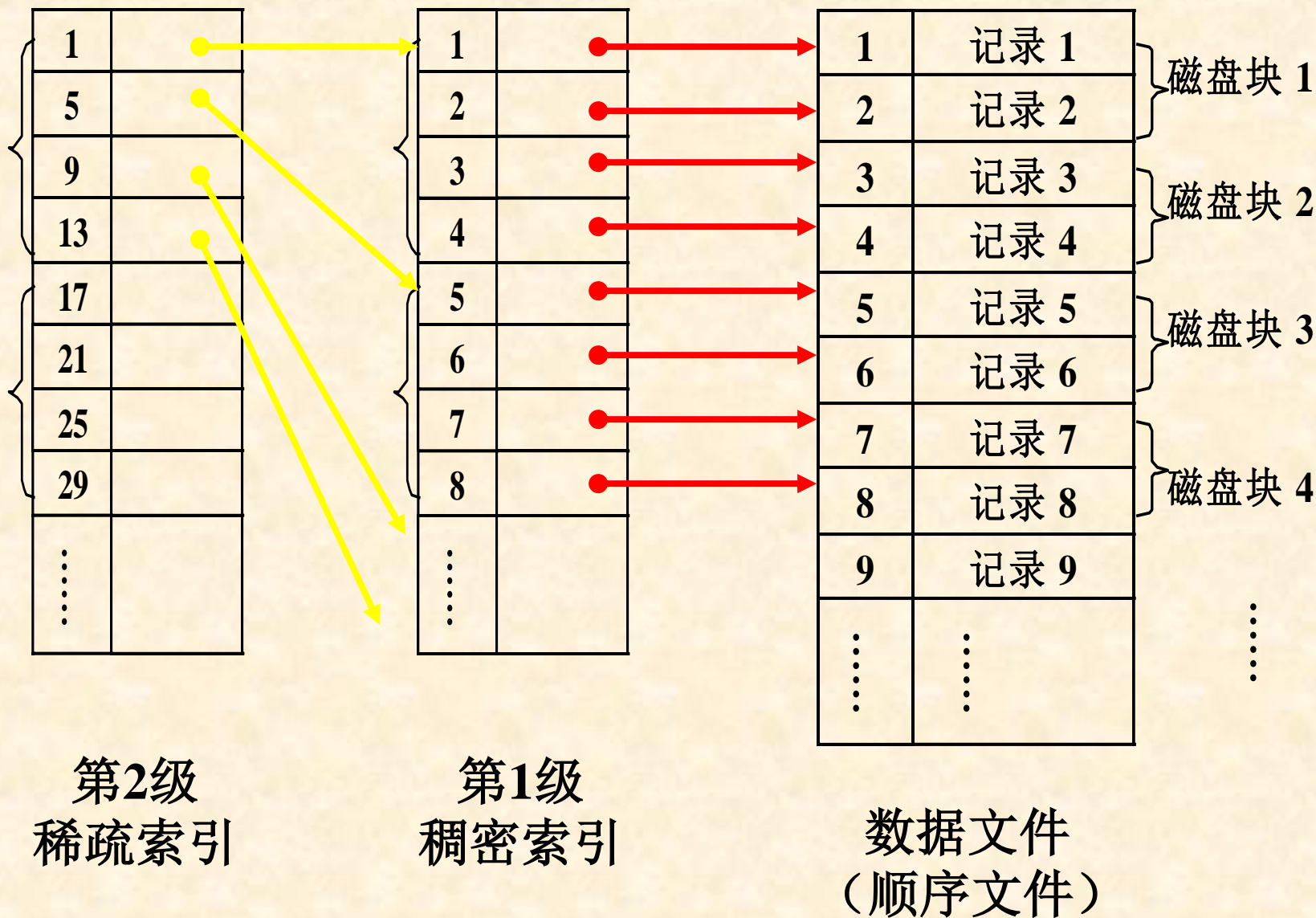
- 索引文件的定义不同
 - ❖ 稀疏索引只能用于顺序文件上的索引组织
 - ❖ 稠密索引中的每个索引项对应数据文件中的一条记录，而稀疏索引中的每个索引项则对应数据文件中的一个磁盘块
- 需要的磁盘空间大小不同
- 在记录的查找定位功能上存在差别：
 - ❖ 稠密索引：可以直接回答是否存在键值为K的记录
 - ❖ 稀疏索引：需要额外的磁盘I/O操作，即需要将相应的数据文件中的磁盘块读入内存后才能判别该记录是否存在

7.6 索引技术与散列技术

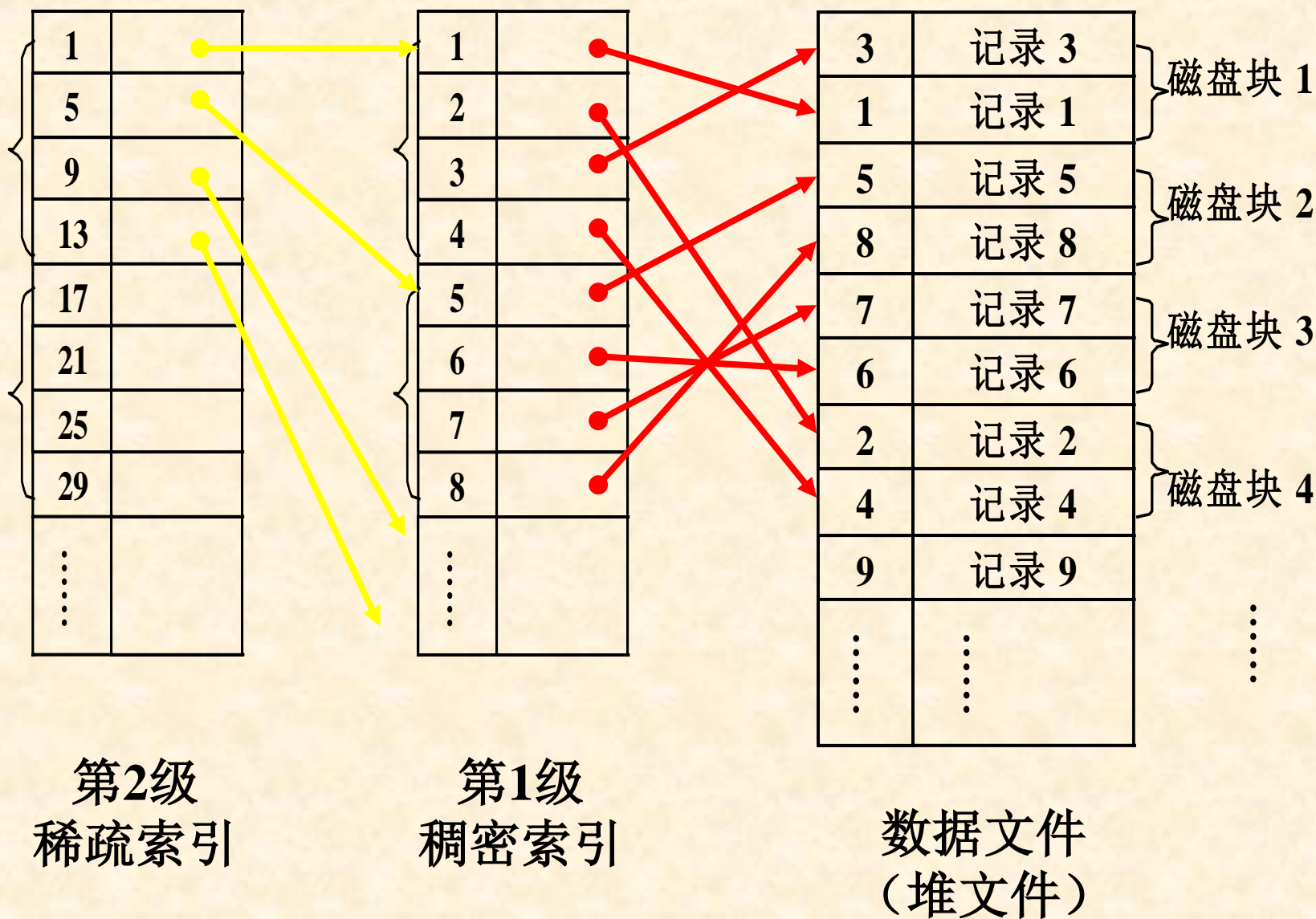
➤ 多级索引

- 索引文件本身也可能占据多个存储块，为了能够快速找到这些索引块在磁盘中的存储位置，需要引入新的索引结构，即在索引文件上再建立索引，从而构成了多级索引
 - 由于索引文件本身是顺序文件，因此索引文件上的索引组织方法采用的是稀疏索引
- 我们将直接建立在数据文件上的索引（例2、例3中建立的索引）称为第一级索引，根据第一级索引文件建立的索引称为第二级索引，依此类推，从而可以建立一个多级索引结构
 - 在多级索引组织结构中，第一级索引可以是稠密索引，也可以是稀疏索引
 - 从第二级索引开始建立的都是稀疏索引

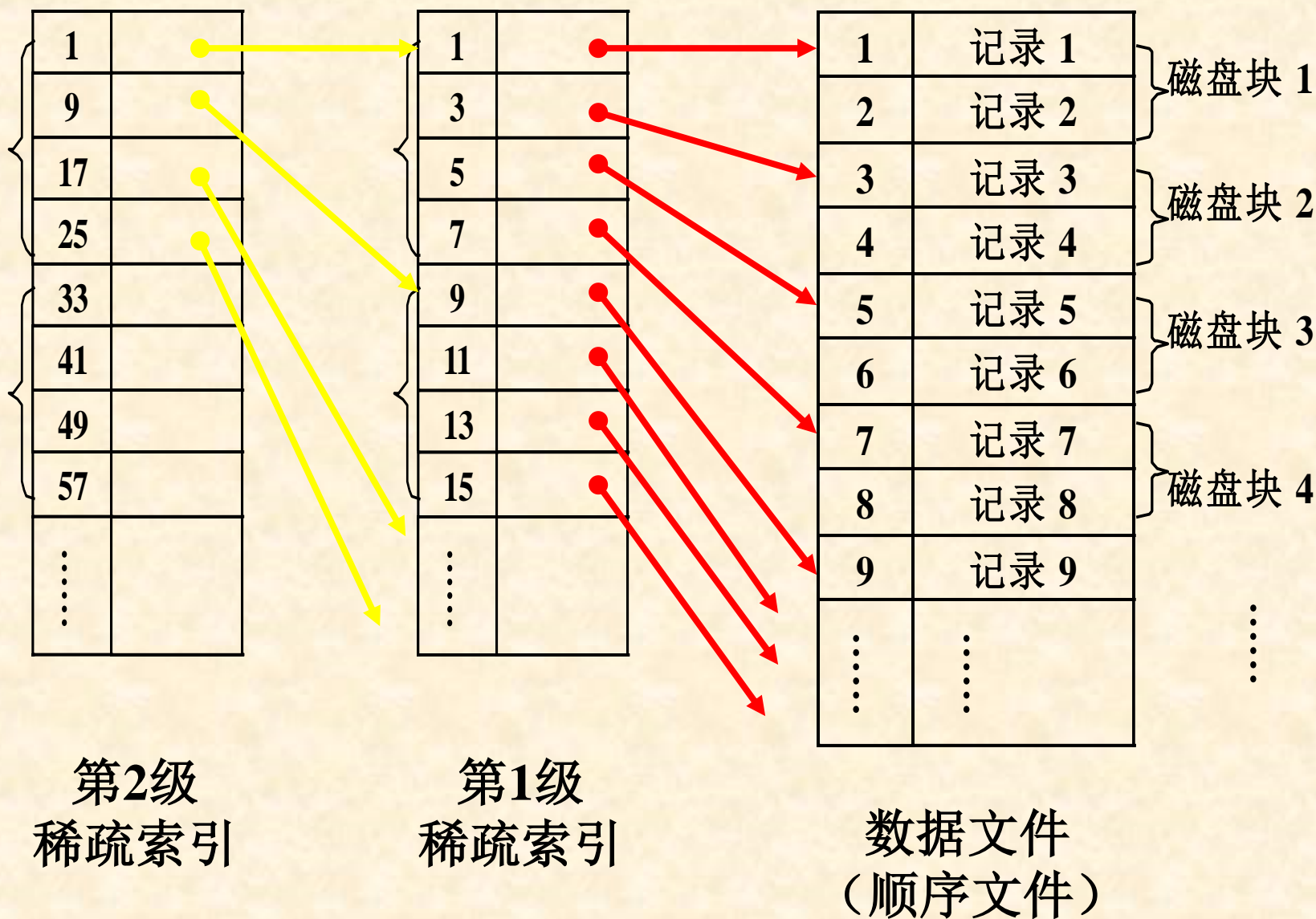
多级索引(续)



多级索引(续)



多级索引(续)



多级索引(续)

➤ [例4]

- 在例2中的稠密索引文件上再建立一个稀疏索引
- 由于例2中的稠密索引文件共占用 10^4 个磁盘块，因此新建立的稀疏索引文件中有 10^4 个索引项，需要占用：

100个磁盘块（约400KB）

➤ [例5]

- 在例3中的稀疏索引文件上也可以再建立一个稀疏索引
- 由于例3中的稀疏索引文件共占用 10^3 个磁盘块，因此新建立的稀疏索引文件中有 10^3 个索引项，只需要占用：

10个磁盘块（约40KB）

多级索引(续)

- 考虑在例4和例5中所建立的第二级索引文件，由于它们只分别占用400KB和40KB的存储空间，这样的索引文件完全可以全部放在内存中
- 在这样的二级索引文件上进行搜索定位不需要索引文件上的磁盘I/O操作，我们只要根据在二级索引中查找到的索引项到第一级索引文件中直接读取相应的索引磁盘块，并根据在该磁盘块中所找到的索引项到数据文件中直接读取相应的数据文件磁盘块或记录
- 因此，根据这样的两级索引结构进行记录的查找定位只需要2次磁盘I/O操作，大大低于我们在例2和例3中所需的磁盘I/O次数（见下表）

多级索引(续)

	索引类型	需要的磁盘空间 (包括数据文件和 索引文件)	记录查找需要的 磁盘I/O次数
例1	无索引，直接在顺序数据文件上进行记录的查找	10^5	17
例2	一级稠密索引	$10^5 + 10^4$	15
例3	一级稀疏索引	$10^5 + 10^3$	11
例4	基于一级稠密索引的 二级索引	$10^5 + 10^4 + 10^2$	2
例5	基于一级稀疏索引的 二级索引	$10^5 + 10^3 + 10^1$	2

多级索引与单级索引的性能比较

7.6 索引技术与散列技术

- 到此为止，我们介绍了基于顺序文件的稠密索引、稀疏索引和多级索引，并且索引属性是关系的关键字，具有唯一性。以此为基础，我们可以构造出索引属性不唯一以及非顺序文件上的索引。

➤ 在顺序数据文件中建立具有重复键值的索引

- 在数据文件中索引关键字的值是不唯一的，即一个索引关键字值对应着若干条记录。
- 具有重复键值的索引文件可以采用以下的组织方法：
 - ❖ 稠密索引
 - ❖ 稀疏索引

7.6 索引技术与散列技术

➤ 具有重复键值的索引文件组织

▪ 具有重复键值的稠密索引

❖ 为数据文件中的 每一个索引关键字值 K 定义一个索引项，其中的记录指针指向索引关键字值等于 K 的第一条记录

• 在索引文件中索引关键字的值是唯一的

❖ 利用该索引文件查找键值为 K 的记录时，所找到的记录指针（如果找到具有键值为 K 的索引项）指向数据文件中第一条键值为 K 的记录

❖ 由于 数据文件是顺序文件，因此，我们可以在数据文件中从找到的第一条记录开始向后扫描读取每一个键值为 K 的记录，直到出现一条键值不为 K 的记录，查找结束

具有重复键值的稠密索引(续)



稠密索引

数据文件（关键字不唯一）

7.6 索引技术与散列技术

➤ 具有重复键值的索引文件组织

■ 具有重复键值的稠密索引（续）

- ❖ 在数据文件中的记录查找可能需要跨越多个磁盘块
- ❖ 在这样的稠密索引文件中，索引关键字的值具有唯一性，且是一个按照索引关键字的取值而建立的顺序文件
- ❖ 因此我们可以按照前面介绍的方法，以该稠密索引为第一级索引，建立多级索引结构

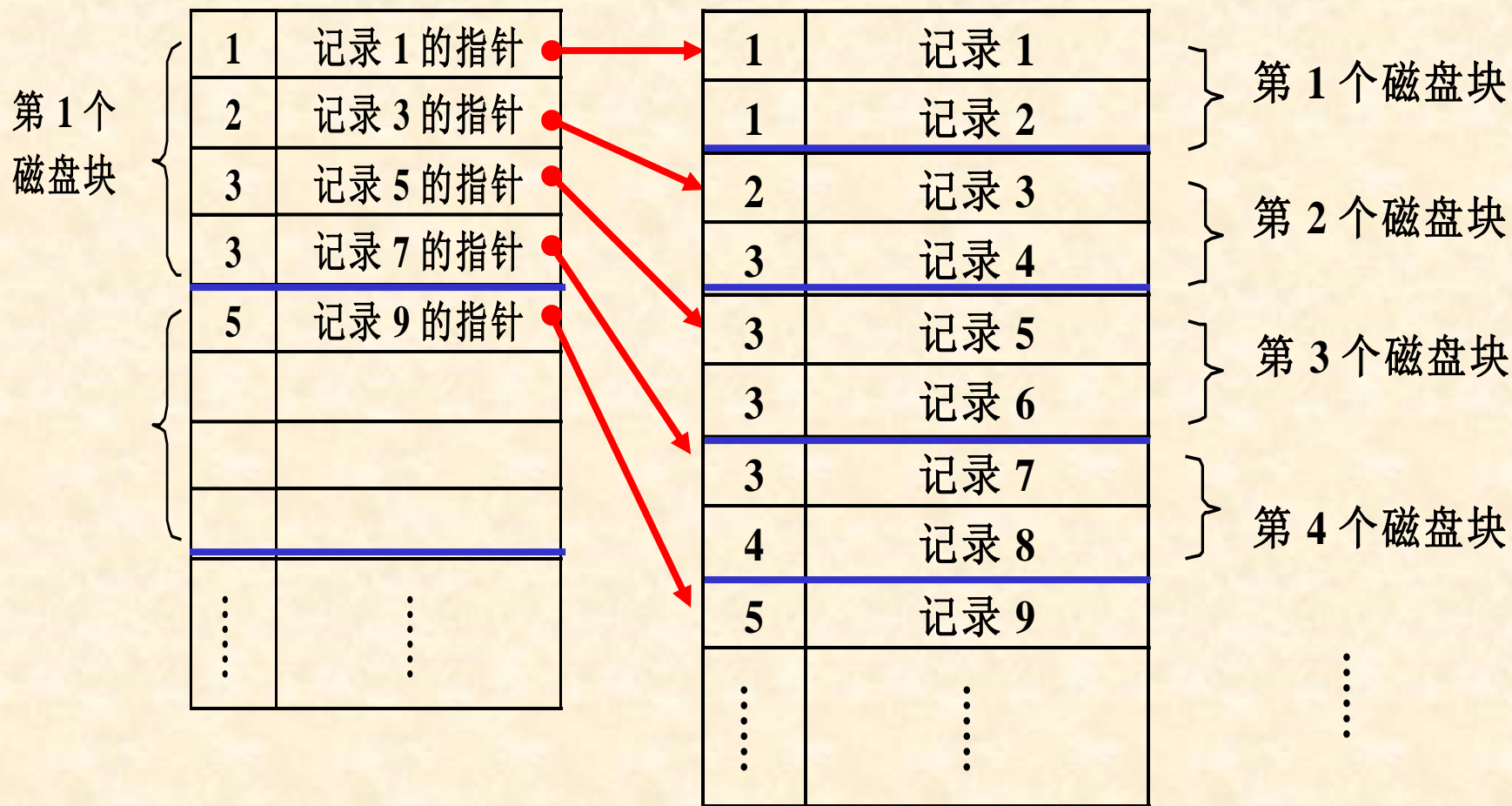
7.6 索引技术与散列技术

➤ 具有重复键值的索引文件组织（续）

■ 具有重复键值的稀疏索引

- ❖ 为顺序数据文件中的每一个磁盘块定义一个索引项，其中的索引关键字值是该磁盘块中第一条记录的索引关键字值，而记录指针则指向该磁盘块的第一条记录
- ❖ 由于具有相同键值的记录可能占用若干个磁盘块，因此该索引文件中的索引关键字值是不唯一的

具有重复键值的稀疏索引(续)



稀疏索引

数据文件（关键字不唯一）

7.6 索引技术与散列技术

➤ 具有重复键值的稀疏索引（续）

- 该稀疏索引的记录查找算法

输入：索引关键字值 K

输出：索引关键字值为 K 的记录集合 R

- 1) 查找索引关键字值小于并最接近于 K 的一个索引项 M ;
- 2) 若不存在这样的索引项，则令 M 为该索引文件的第一个索引项;
- 3) 如果 M 的索引关键字值大于 K ，则表明不存在索引关键字值为 K 的记录，查找结束；否则根据 M 中的记录指针读取数据文件中的磁盘块 D ;

具有重复键值的稀疏索引 - 记录查找算法（续）

- 4) 将记录指针指向磁盘块 D 的第一条记录，并作如下处理：
 - a) 如果当前记录的索引关键字值大于 K ，则跳转至步骤 5) ；
 - b) 如果当前记录的索引关键字值等于 K ，则将当前记录加入到结果记录集 R 中；
 - c) 如果当前记录是磁盘块 D 的最后一条记录，则：
 - 如果磁盘块 D 是数据文件中的最后一个磁盘块，则跳转至步骤 5) ，否则读取下一个磁盘块的数据并赋给变量 D ，然后返回步骤 4) 以处理新磁盘块中的记录。
 - d) 将当前记录指针指向该磁盘块中的下一条记录，并返回步骤 a) 以比较新记录中的索引关键字值；
- 5) 返回结果记录集 R ，算法结束。

7.6 索引技术与散列技术

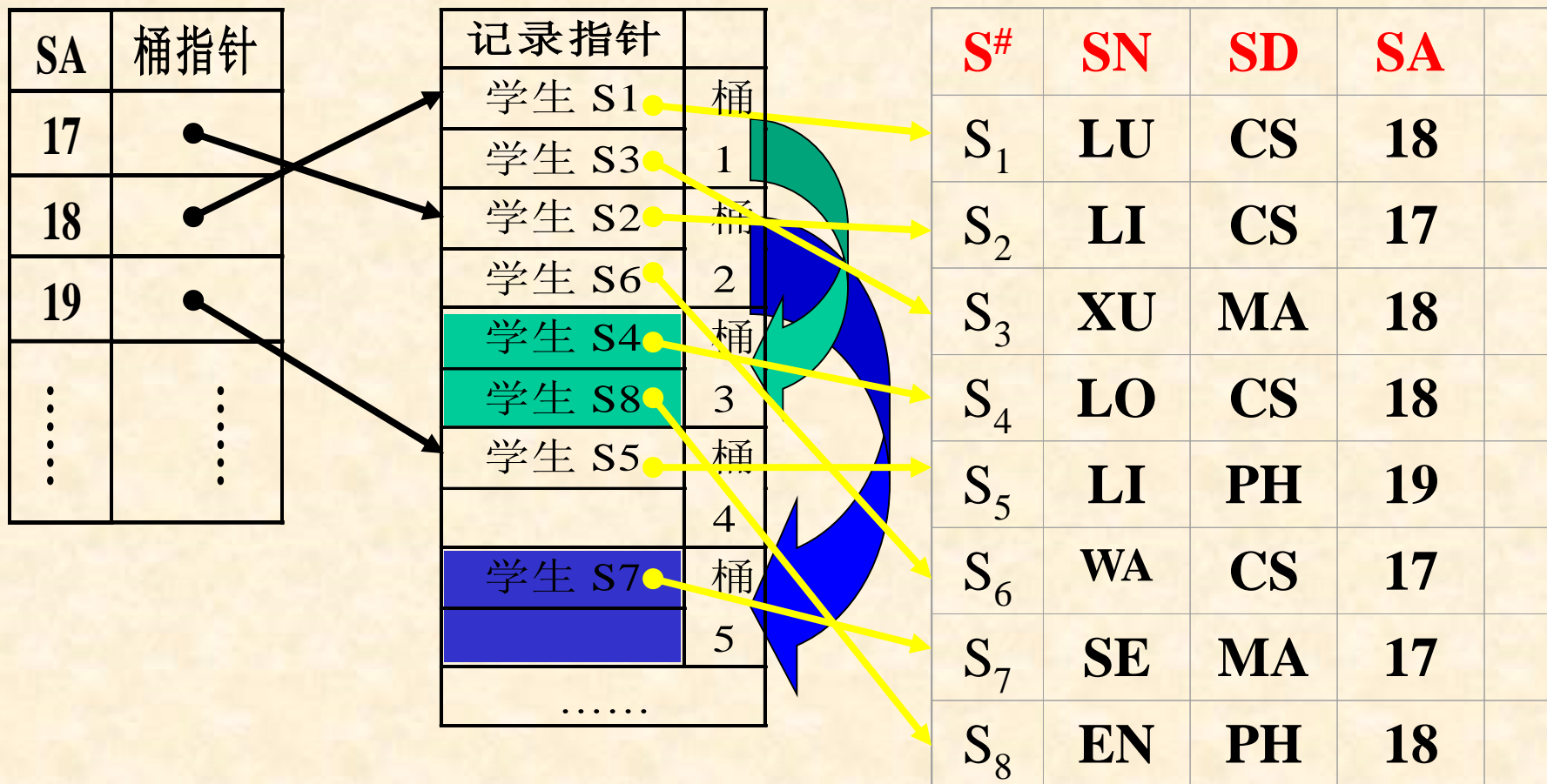
2. 非顺序文件中的索引技术

- 在数据库系统中所使用的数据文件通常都是无序的（堆文件组织），因此迫切需要利用上述的索引技术来建立非顺序文件上的索引，以提高记录查找的速度
- 如果索引关键字值在非顺序数据文件中具有唯一性，则可以按照下述步骤建立其上的索引文件：
 - ❖ 首先为非顺序数据文件建立第一级的稠密索引
 - ❖ 然后再根据需要建立该稠密索引上的多级稀疏索引

非顺序文件中的索引技术(续)

- 如果索引关键字的值不唯一，则可以通过在第一级的稠密索引和数据文件之间加一个记录指针桶(bucket)。此时索引项(K_i, P_i)中的记录指针 P_i 不再是指向数据文件中的记录，而是指向一个记录指针桶，在桶中存放着索引关键字值为 K_i 的记录的记录指针
 - ❖ 在这里，记录指针桶的大小 BSize 是预先确定的
 - ❖ 当在数据文件中插入第一条索引关键字值为 K_i 的记录 R_i 时，在索引文件中将生成一个新的索引项 (K_i, P_i)，同时系统将自动申请一个大小为 BSize 的记录指针桶 B_i ，将指针 P_i 指向该记录指针桶 B_i ，同时将当前记录 R_i 的记录指针保存在记录指针桶 B_i 中
 - ❖ 当新的索引关键字值为 K_i 的记录被加入到数据文件中时，只需要将新记录的记录指针加入到记录指针桶 B_i 中。如果记录指针桶 B_i 已满，系统将申请一个新的记录指针桶，并链接到原来的记录指针桶的后面

非顺序文件中的索引技术(续)



稠密索引

记录指针桶

索引关键字值不唯一，且数据文件没有按照索引关键字值排序存储

7.6 索引技术与散列技术

3. 多维索引

- 在上述介绍的索引文件中，我们并没有关心索引关键字的构成，它可以由关系中的单个属性上的值组成，也可以是多个属性值的一个组合
- 根据多个属性值的组合来建立的索引文件被称为多维索引

多级索引(续)

	索引类型	需要的磁盘空间 (包括数据文件和 索引文件)	记录查找需要的 磁盘I/O次数
例1	无索引，直接在顺序数据文件上进行记录的查找	10^5	17
例2	一级稠密索引	$10^5 + 10^4$	15
例3	一级稀疏索引	$10^5 + 10^3$	11
例4	基于一级稠密索引的 二级索引	$10^5 + 10^4 + 10^2$	2
例5	基于一级稀疏索引的 二级索引	$10^5 + 10^3 + 10^1$	2

多级索引与单级索引的性能比较

7.6 索引技术与散列技术

- 索引文件本身是一个顺序文件，利用索引文件或多级索引可以大大提高数据文件的访问效率
- 索引顺序文件的不足
 - ❖ 记录查找算法的效率不高 ($\log_2 N$)
 - ❖ 在数据文件是非顺序文件，或索引关键字值的变化比较频繁的情况下，索引顺序文件自身的维护非常复杂，对索引项的插入、修改和删除操作会导致索引项在索引文件中的大量移动
 - ❖ 在上述情况下，如果通过引入链接磁盘块的方法来减少索引项的移动，又会减低存储空间的利用率，并最终影响到系统的性能
 - ❖ 因此需要引入适合于索引文件的存储组织的文件结构，这就是在数据库系统中广泛使用的多级索引技术：

B/B+树索引

B/B⁺树索引文件

- B/B⁺树是一种多级索引组织方法，是适合于组织存放在外存的大型磁盘文件的一种树状索引结构。其中用得比较多的是B⁺树。下面的讨论如不特别声明都是指B⁺树索引文件
- B/B⁺树的结点划分
 - ❖ 叶结点：B/B⁺树的最下一级索引是树的叶结点
 - ❖ 内部结点：B/B⁺树中的其它结点(非叶结点)，其中：
 - 根结点：B/B⁺树的最上一级索引是树的根结点
- 在B/B⁺树中，由叶结点所构成的最下面的一级索引通常采用稠密索引，而其它层次上的索引则采用稀疏索引

B/B+树索引文件(续)

■ B⁺树的特点

❖ 平衡性

- 从树的根结点到每个叶子结点的路径都是等长的

❖ 过半性

- 每个结点（除根结点外）所对应的磁盘块至少被占用一半的存储空间

❖ 顺序性

- 既提供从根结点开始的随机查找关键字功能，也能根据索引关键字的值的排序进行顺序访问

❖ 自适应性

- 自动保持与数据文件大小相适应的索引层次

1. B+树中的结点

- 每个结点占用一个磁盘块，每棵B+树都有一个被称为秩的整型参数 n ，每个结点能容纳 n 个键和 $n+1$ 个指针，我们将 n 取得尽可能的大，以便在一个磁盘块中存放更多的索引项
- 例如：设每个磁盘块的大小是4096个字节，每个键值占4个字节，每个指针占8个字节，则满足：
$$4n + 8(n+1) \leq 4096$$
的最大 n 的值是340
- 秩为 n 的B+树的结点的结构如下：

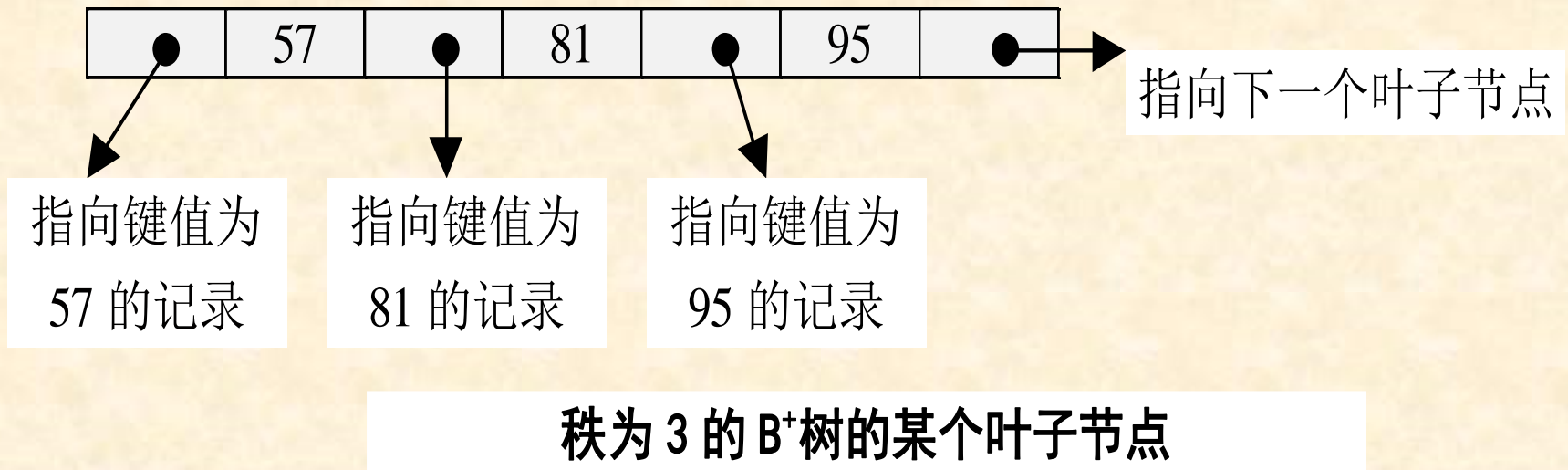
B/B+树索引文件(续)

P_1	K_1	P_2	K_2	P_m	K_m	P_{m+1}
-------	-------	-------	-------	-------	-------	-------	-----------

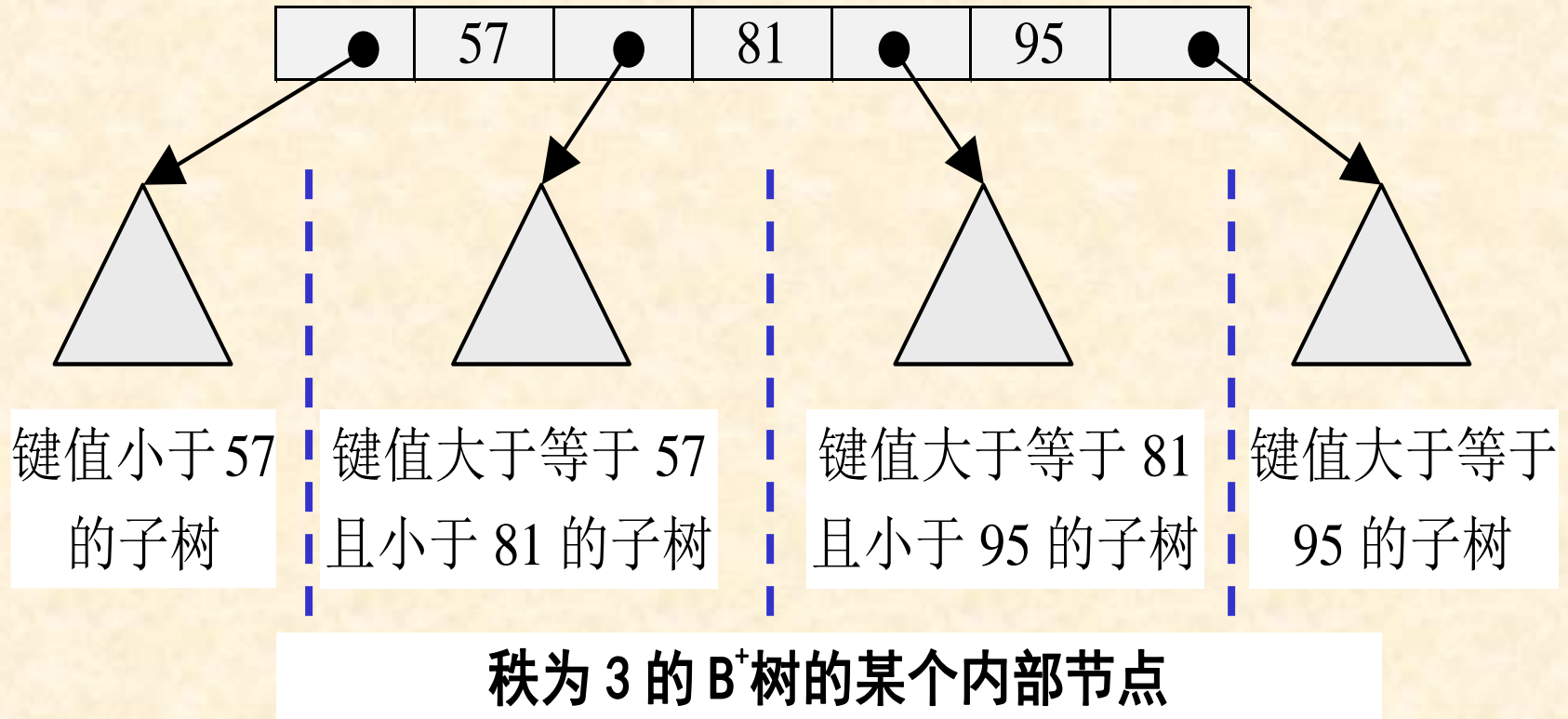
- 其中: K_1, K_2, \dots, K_m 是索引关键字值, 且 $K_1 < K_2 < \dots < K_m$
- 1) 若是叶子结点, 则: $\lfloor (n+1)/2 \rfloor \leq m \leq n$, P_1, P_2, \dots, P_m 是指向数据记录的指针, P_{m+1} 是指向其右边的下一个叶子结点的指针。其中, P_i 是指向关键字值为 K_i 的数据记录 ($i=1, 2, \dots, m$)
 - 2) 若是根结点, 则 $1 \leq m \leq n$, 否则(即内部结点):
 $\lceil (n-1)/2 \rceil \leq m \leq n$
其中: $P_1, P_2, \dots, P_m, P_{m+1}$ 分别指向另一棵子树的根结点
- ❖ 若是非叶结点, 则:
- 对 P_1 所指向的子树中的任意一个索引关键字 K , 都有 $K < K_1$
 - 对 P_{m+1} 所指向的子树中的任意一个索引关键字 K , 都有 $K \geq K_m$
 - 对 P_j 所指向的子树中的任意一个索引关键字 K , 都有 $K_{j-1} \leq K < K_j$

B/B⁺树索引文件(续)

- 例如：有一棵秩 $n=3$ 的B⁺树，则每个结点最多可放3个关键字和4个指针，每个叶子结点至少有2个关键字和3个指针，每个内部结点至少有1个关键字和2个指针。



B/B+树索引文件(续)



- a) 如果一颗B⁺树只有唯一的一个节点，那么该节点既是该B⁺树的根节点，同时也是其唯一的一个叶子节点，其结构采用叶子节点的组织方式
- b) 否则，一颗B⁺树的根节点将采用与其内部节点相同的组织方式，只是根节点中的关键字值的个数不受其下限的限制

B/B+树索引文件(续)

2. B+树中的查找

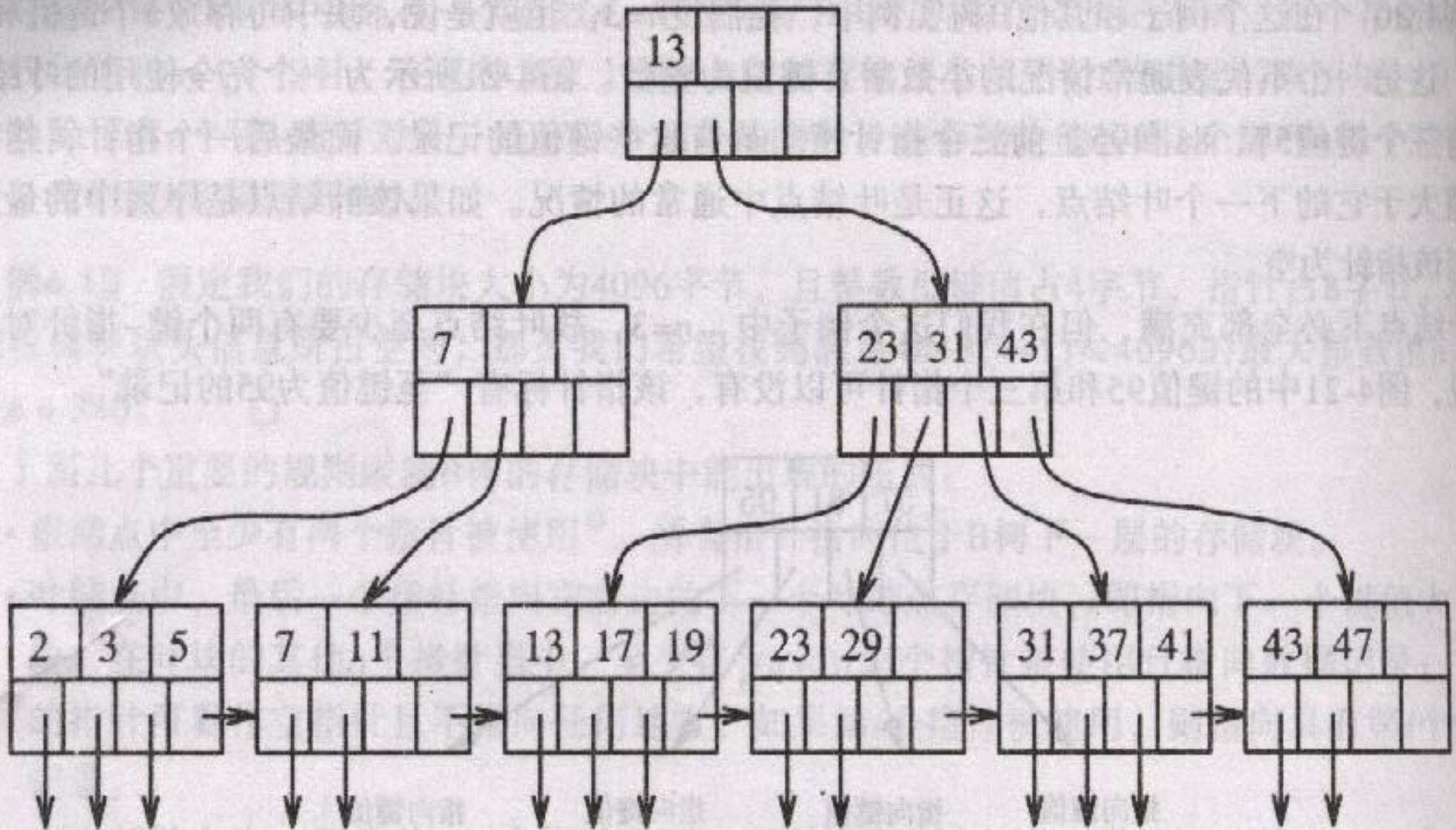
➤ 算法1: B+树上的随机查找算法

- 输入: 索引关键字值 K
- 输出: 关键字值为 K 的记录指针
- 算法:
 - 1) 结点 $N := B^+$ 树的根结点;
 - 2) 若结点 N 是叶子结点, 则转步骤 4) ;
 - 3) 设结点 N 有 m 个关键字 K_1, K_2, \dots, K_m 和 $m+1$ 个指针 P_1, P_2, \dots, P_{m+1} ,
 - ❖ 若 $K < K_1$, 则令结点 $N := P_1$ 所指向的子树的根结点;
 - ❖ 若 $K \geq K_m$, 则令结点 $N := P_{m+1}$ 所指向的子树的根结点;
 - ❖ 若 $K_j \leq K < K_{j+1}$, 则令结点 $N := P_{j+1}$ 所指向的子树的根结点;

返回步骤 2)

- 4) 设结点 N 有 m 个关键字 K_1, K_2, \dots, K_m 和 $m+1$ 个指针 P_1, P_2, \dots, P_{m+1} , 若找到一个索引关键字 $K_j = K$, 则返回记录指针 P_j , 否则返回空指针, 查找失败。

B/B+树索引文件(续)

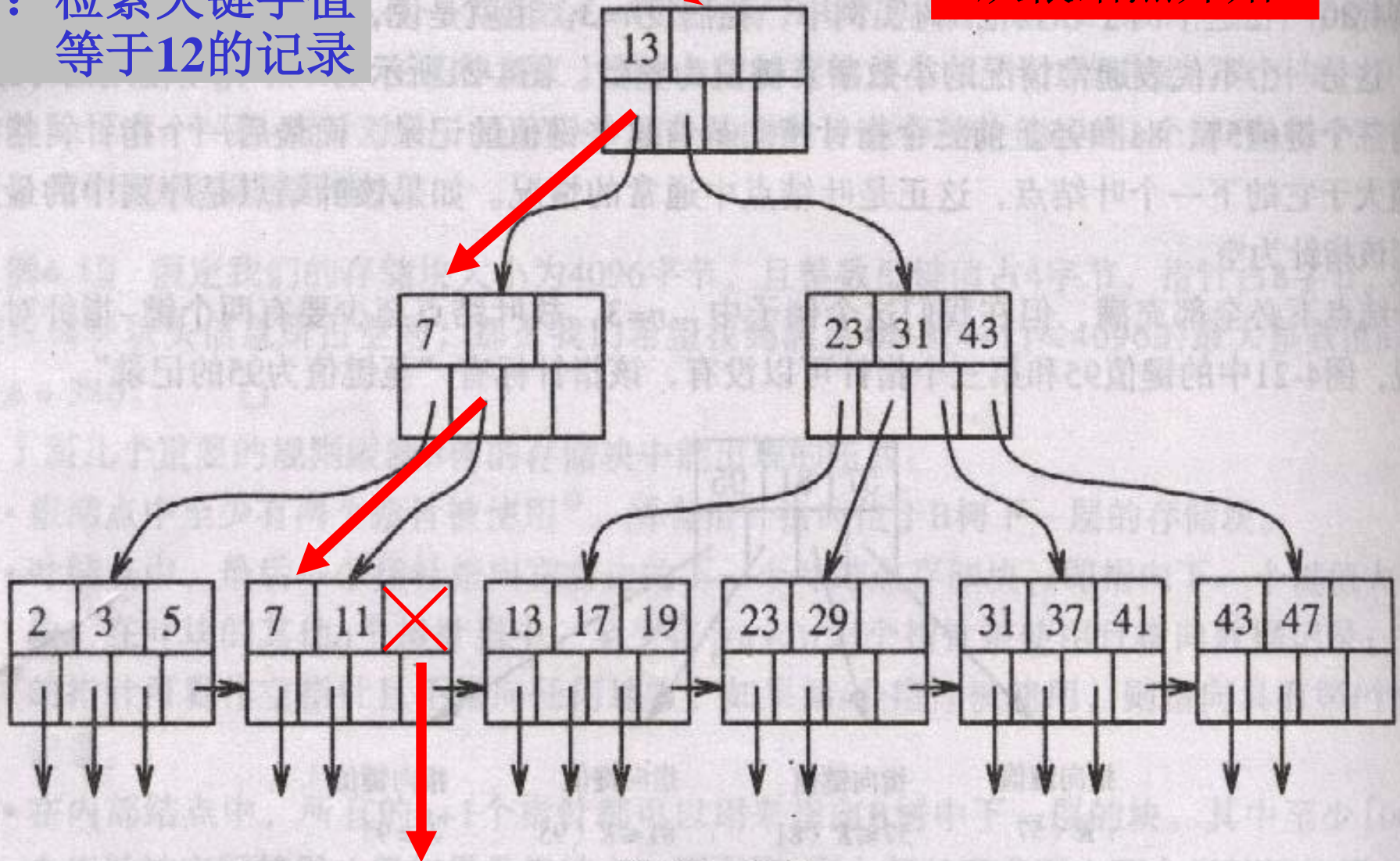


例6：由2到47之间的所有素数所构成的B+树（秩为3）

B/B+树索引文件(续)

从根结点开始

例：检索关键字值
等于12的记录

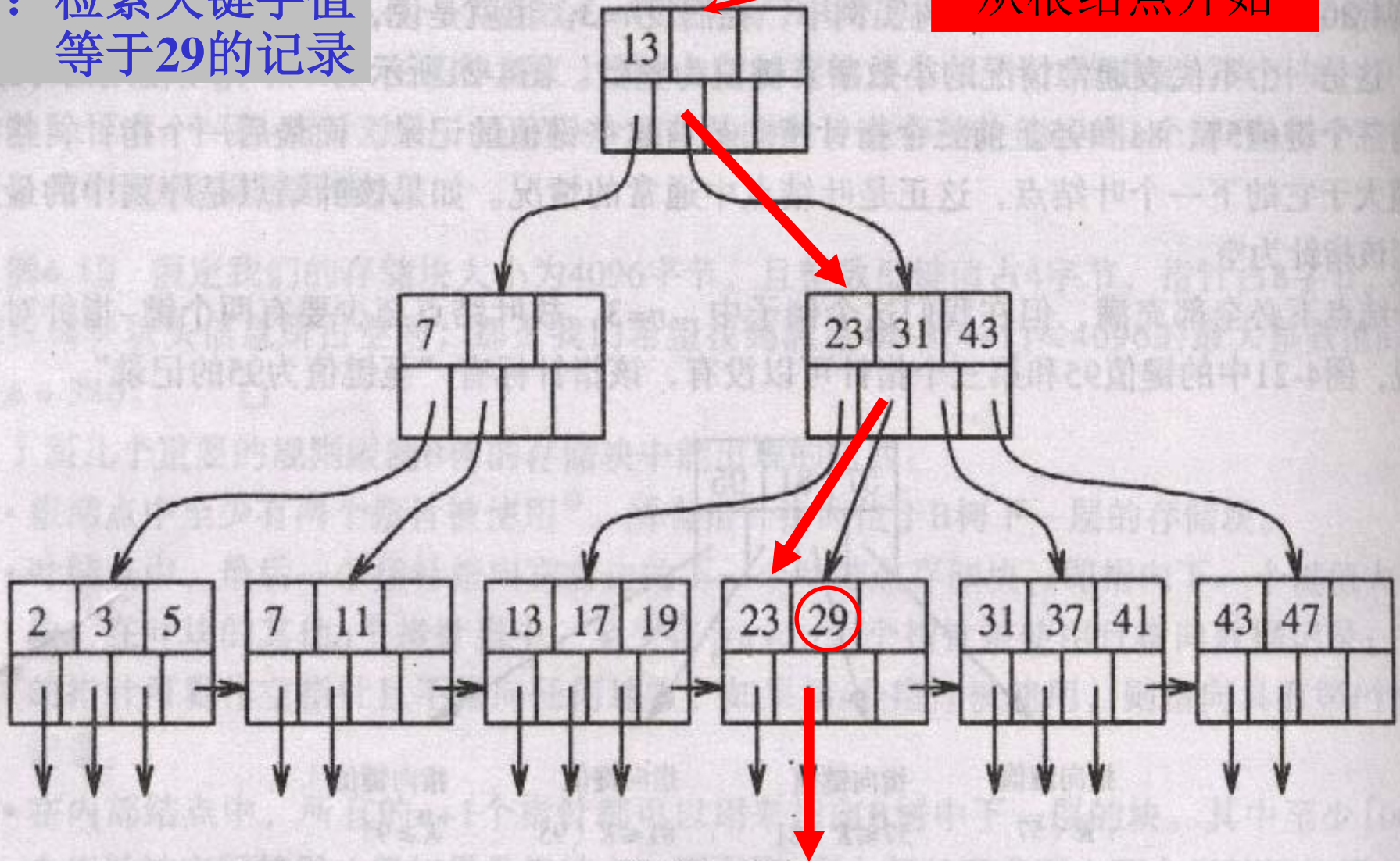


不存在关键字为12的记录

B/B+树索引文件(续)

例：检索关键字值
等于29的记录

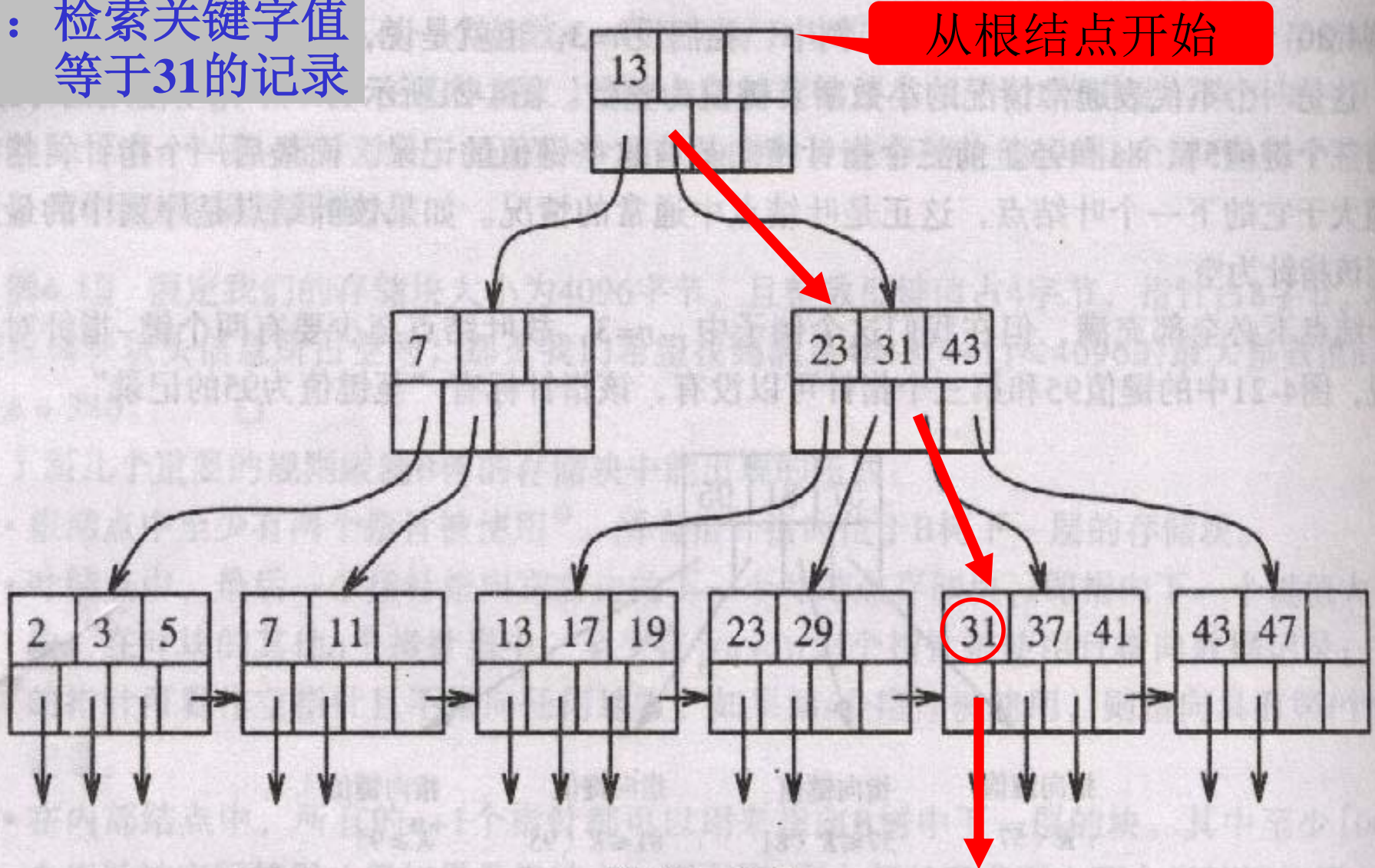
从根结点开始



指向关键字值为29的记录

B/B+树索引文件(续)

例：检索关键字值
等于31的记录



B/B+树索引文件(续)

2. B+树中的查找 (续)

➤ 算法2: B+树上的范围查询

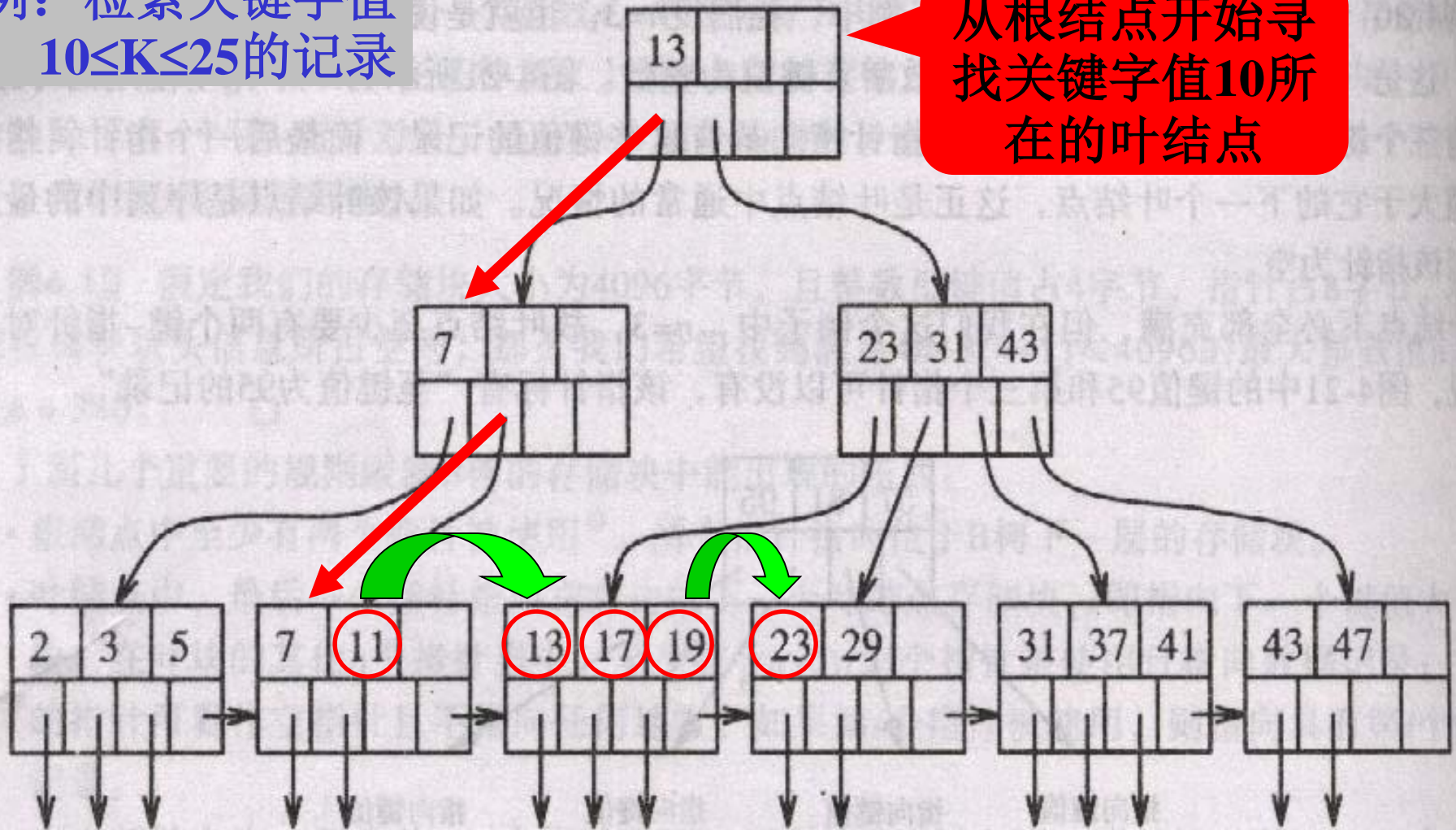
- 输入: 索引关键字值的范围 $[a, b]$
- 输出: 满足条件的记录指针集合
- 算法:
 - 1) 利用算法1找到关键字 a 所在的叶子结点 N ;
 - 2) 在结点 N 中寻找第一个大于或等于 a 的关键字 K_i 。如果没有找到, 则转步骤5;
 - 3) 在结点 N 中搜索从 K_i 开始的每一个关键字 K ;
 - 4) 对每一个关键字值 K 作如下比较:
 - ❖ 若 $K \leq b$, 则将关键字 K 所对应的记录指针加入结果集中, 并继续比较下一个关键字值;
 - ❖ 若 $K > b$, 则转步骤 7);
 - 5) 如果结点 N 的下一个叶子结点指针为空, 则转步骤 7); 否则结点 $N :=$ 下一个叶子结点;
 - 6) 从结点 N 中的第一个关键字值 K_1 开始, 返回步骤4继续比较;
 - 7) 算法终止, 返回结果集。

B/B+树索引文件(续)

- **复习思考题：**对算法2稍做调整，可以得到有关 $a < K < b$ 、 $a < K \leq b$ 、 $a \leq K < b$ 、 $K \leq a$ 、 $K \geq a$ 、 $K < a$ 、 $K > a$ 等范围查询的算法

B/B+树索引文件(续)

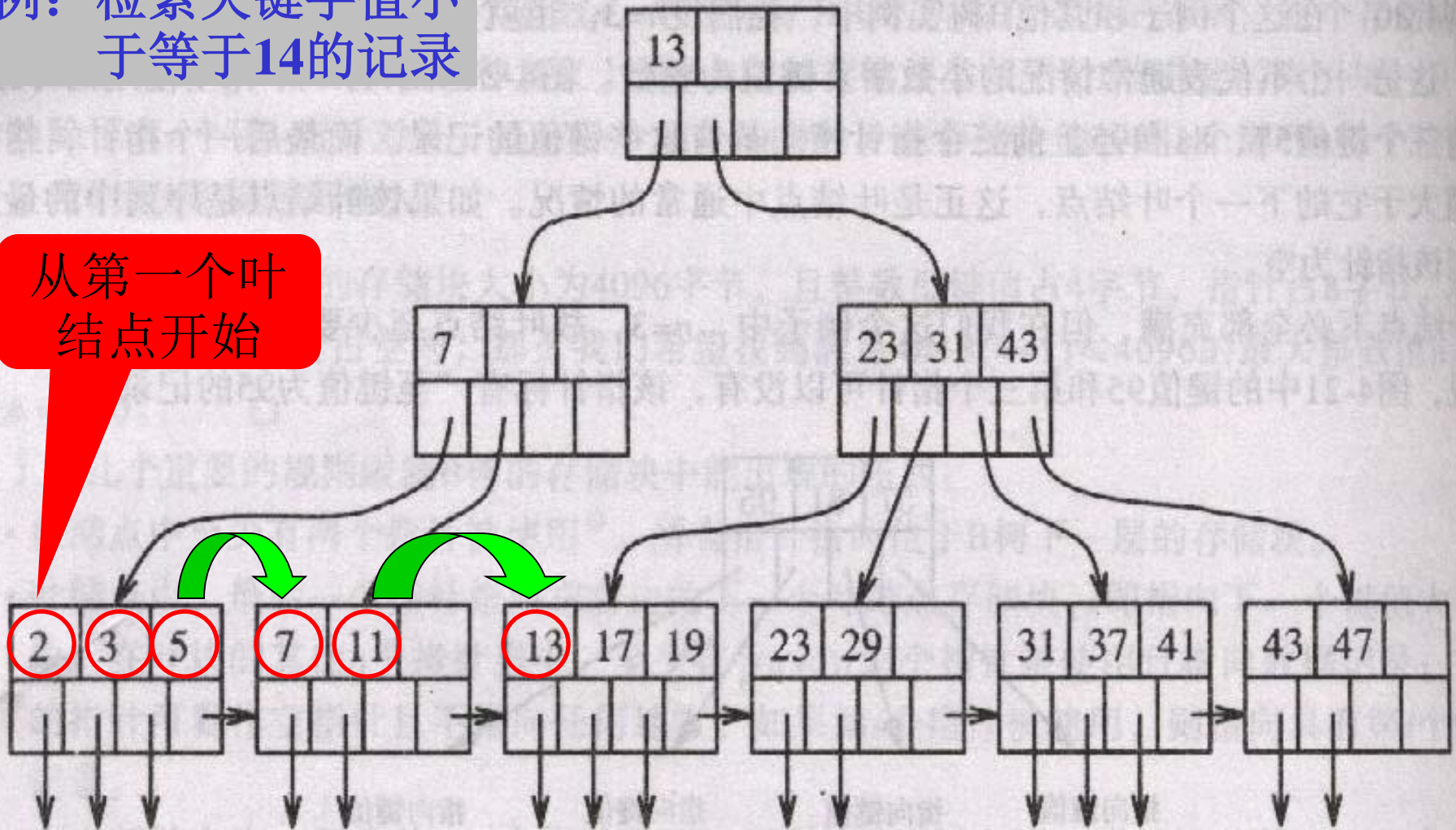
例：检索关键字值
 $10 \leq K \leq 25$ 的记录



B/B+树索引文件(续)

例：检索关键字值小于等于14的记录

从第一个叶
结点开始

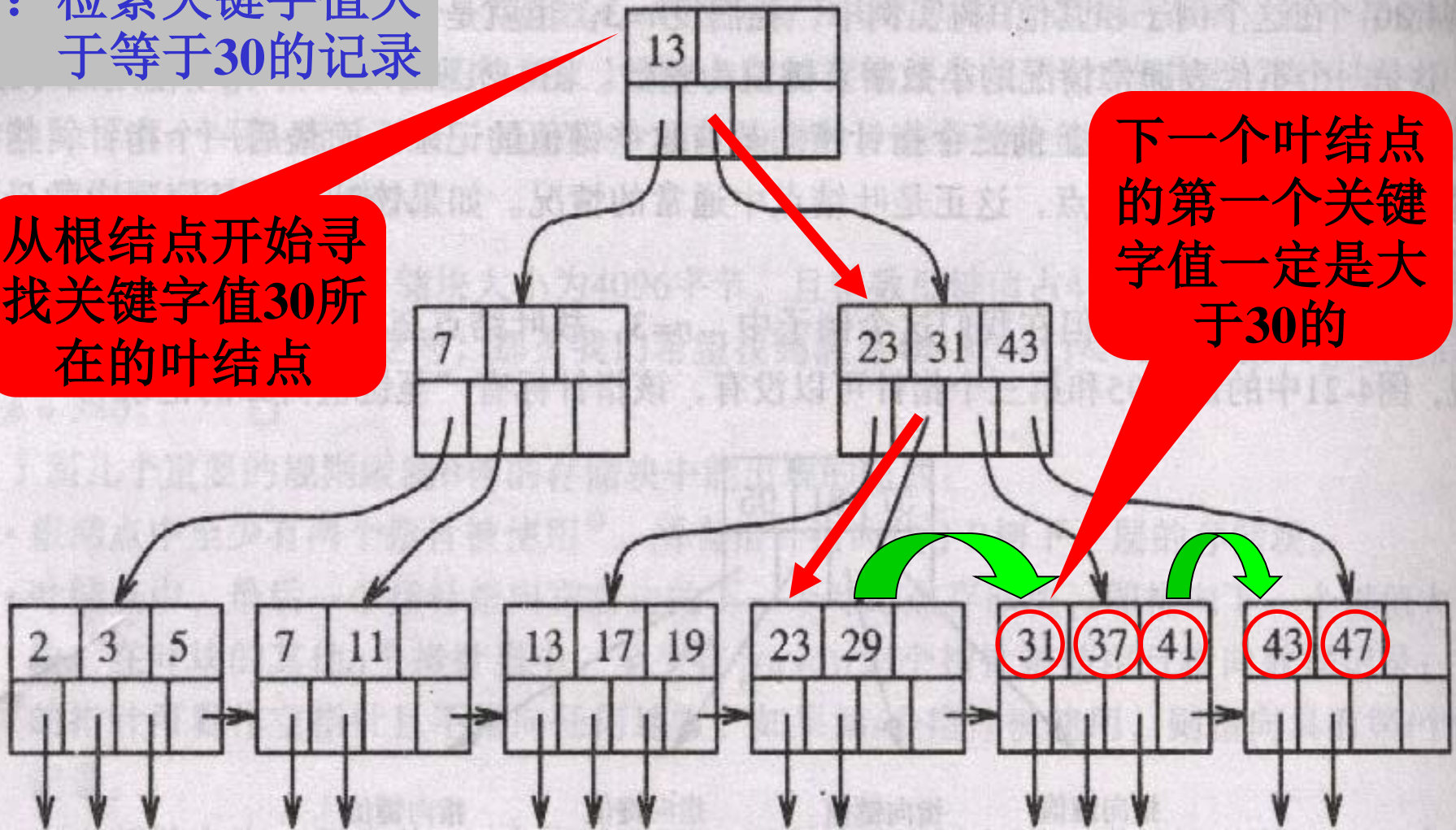


B/B+树索引文件(续)

例：检索关键字值大于等于30的记录

从根结点开始寻找关键字值30所在的叶结点

下一个叶结点的第一个关键字值一定是大于30的



3. B+树中的插入操作

➤ **算法3:** 在一个秩为 n 的B+树中插入一个关键字值为 K 的索引项（在索引文件中不允许出现重复的关键字值）

■ 算法的思想

- ❖ 寻找插入操作的最初执行结点（叶结点）
- ❖ 插入一个关键字值为 K 的索引项
- ❖ 当当前结点的关键字数目超过其上限时，当前结点将被分裂为两个结点，并将其中间关键字值插入到当前结点的父结点中去
- ❖ 父结点中的插入操作又有可能引起父节点的分裂，这样的结点分裂过程可能会一直波及到根结点，而根结点的分裂则会产生新的根结点，并导致树的高度的增加

B+树中的插入算法（续）

- 1) 利用算法1查找到关键字值 K 应该插入的叶子结点 N ;
- 2) 在结点 N 中查找关键字 K , 如果该关键字已经存在, 则不允许再做该关键字的插入操作, 转步骤 7) 。
- 3) 将由关键字 K 及其记录（子树）指针构成的索引项插入到结点 N 的相应位置上。
- 4) 如果结点 N 中的关键字数目没有超过其上限, 则转步骤 7) ; 否则将结点 N 分裂为两个结点 N 和 N' , 其中 N' 是新申请的结点。并且:

B+树中的插入算法（续）

- 如果是叶结点的分裂：
 - ❖ 则将原结点N中的后 $\lceil (n+1)/2 \rceil$ 个索引项转移到新结点 N' 中去；
 - ❖ 结点 N' 的后续叶结点指针指向原结点 N 的后续叶结点，而结点 N 的后续叶结点指针指向新结点 N' ；
 - ❖ 由新结点 N' 的第一个关键字值 K' 和指向新结点 N' 的子树指针 P' 构成一个新索引项 (K', P')

B+树中的插入算法（续）

- 如果是内部结点的分裂（令 $m1 = \lfloor (n+1)/2 \rfloor$, $m2 = \lceil (n-1)/2 \rceil$ ）
 - ❖ 则将原结点 N 中的前 $m1$ 个索引关键字值和前 $(m1+1)$ 个子树指针仍然保留在结点 N 中；
 - ❖ 后 $m2$ 个索引关键字值和 $(m2+1)$ 个子树指针转移到新结点 N' 中去；
 - ❖ 提取原结点 N 的中间关键字值 K' （第 $m1+1$ 个索引关键字值），并和指向新结点 N' 的子树指针 P' 构成一个新索引项 (K', P')

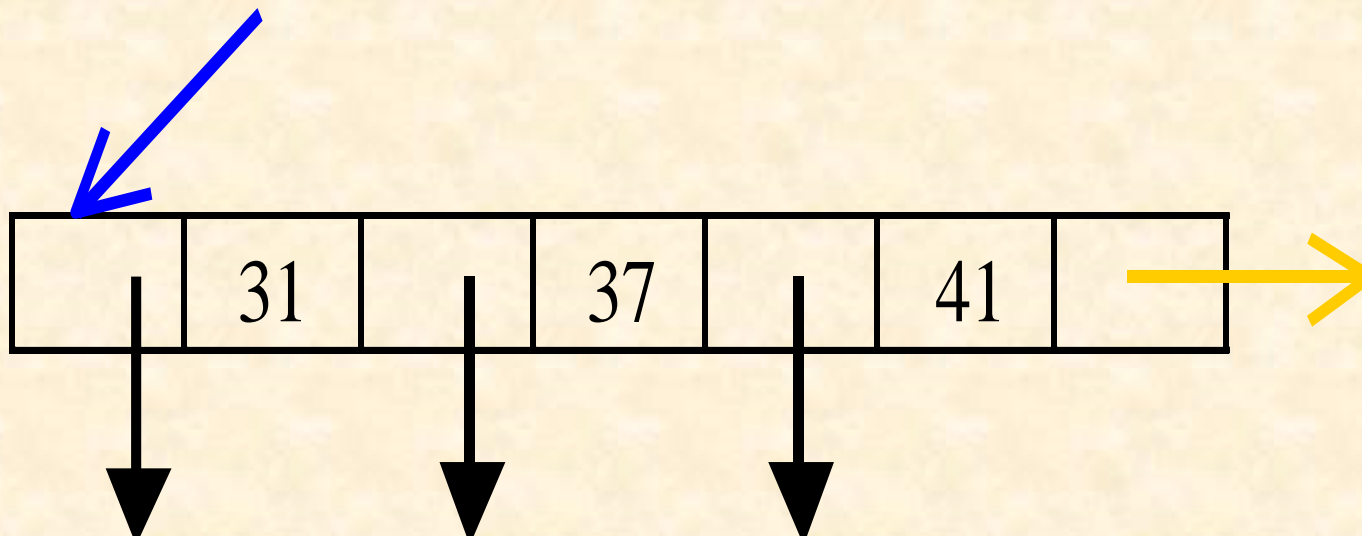
B+树中的插入算法（续）

- 5) 如果原结点 N 不是根结点，则转步骤 3)，在原结点 N 的根结点中的相应位置上插入索引关键字值 K' 和子树指针 P' 。
- 6) 原结点 N 是根结点，则申请一个新的结点 R ，并且将指向原结点 N 的子树指针 P_N 、索引关键字值 K' 和子树指针 P' 插入到新结点 R 中，并且将新结点 R 设置为该索引文件的根结点。
- 7) 算法结束。

B+树中的插入算法（续）

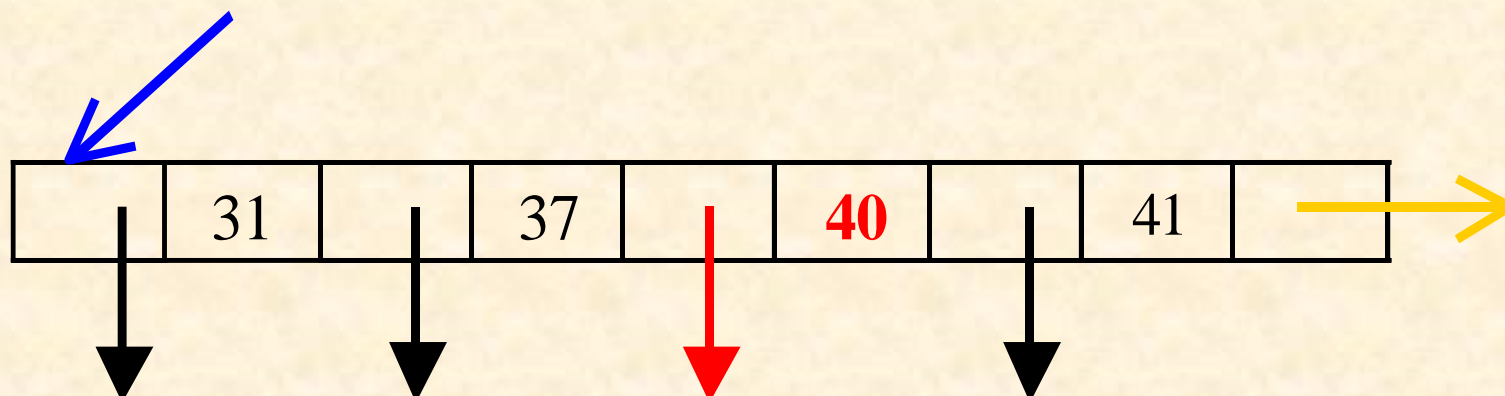
➤ **例7：**在例6给出的B⁺树中插入一个关键字40（及其记录指针）

1) 首先找到执行插入操作的叶结点



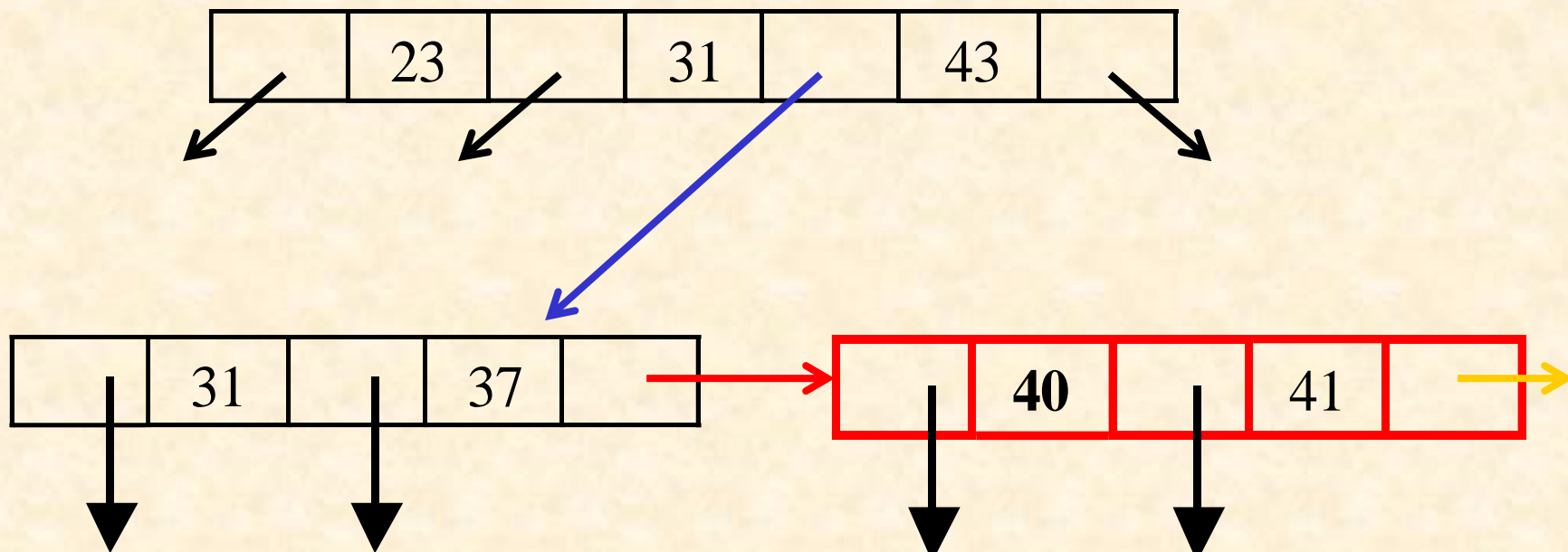
B+树中的插入算法（续）

2) 插入关键字40及其记录指针



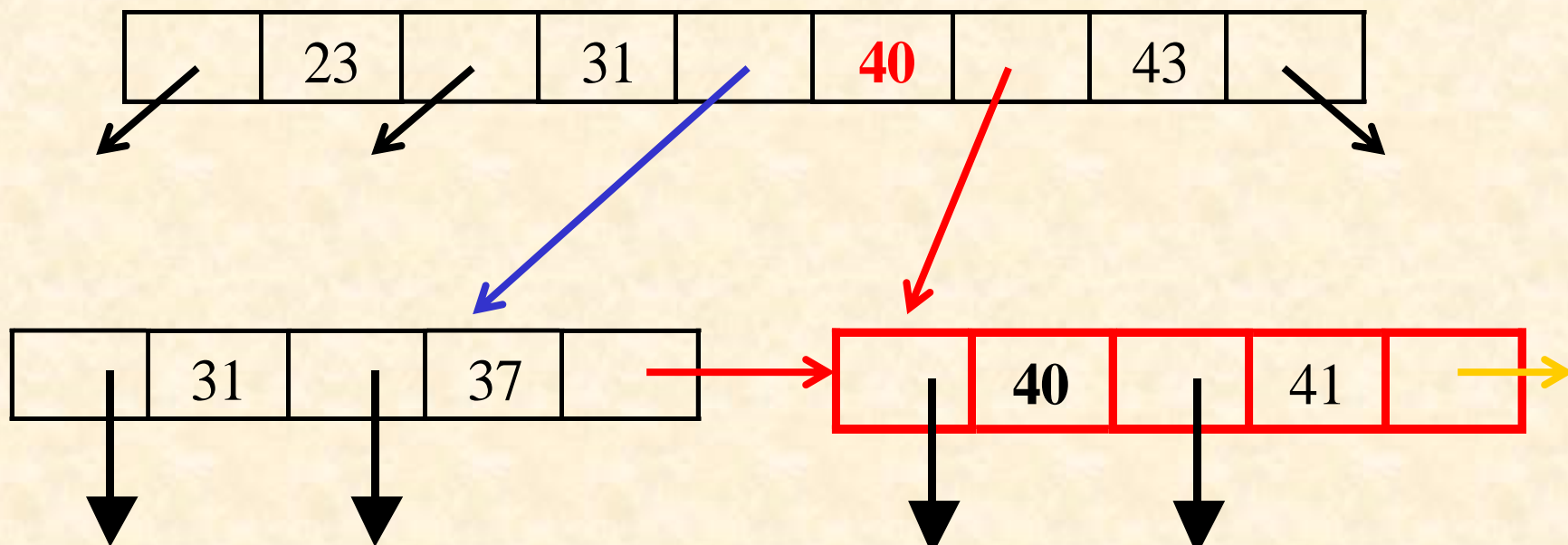
B+树中的插入算法（续）

3) 叶结点的分裂



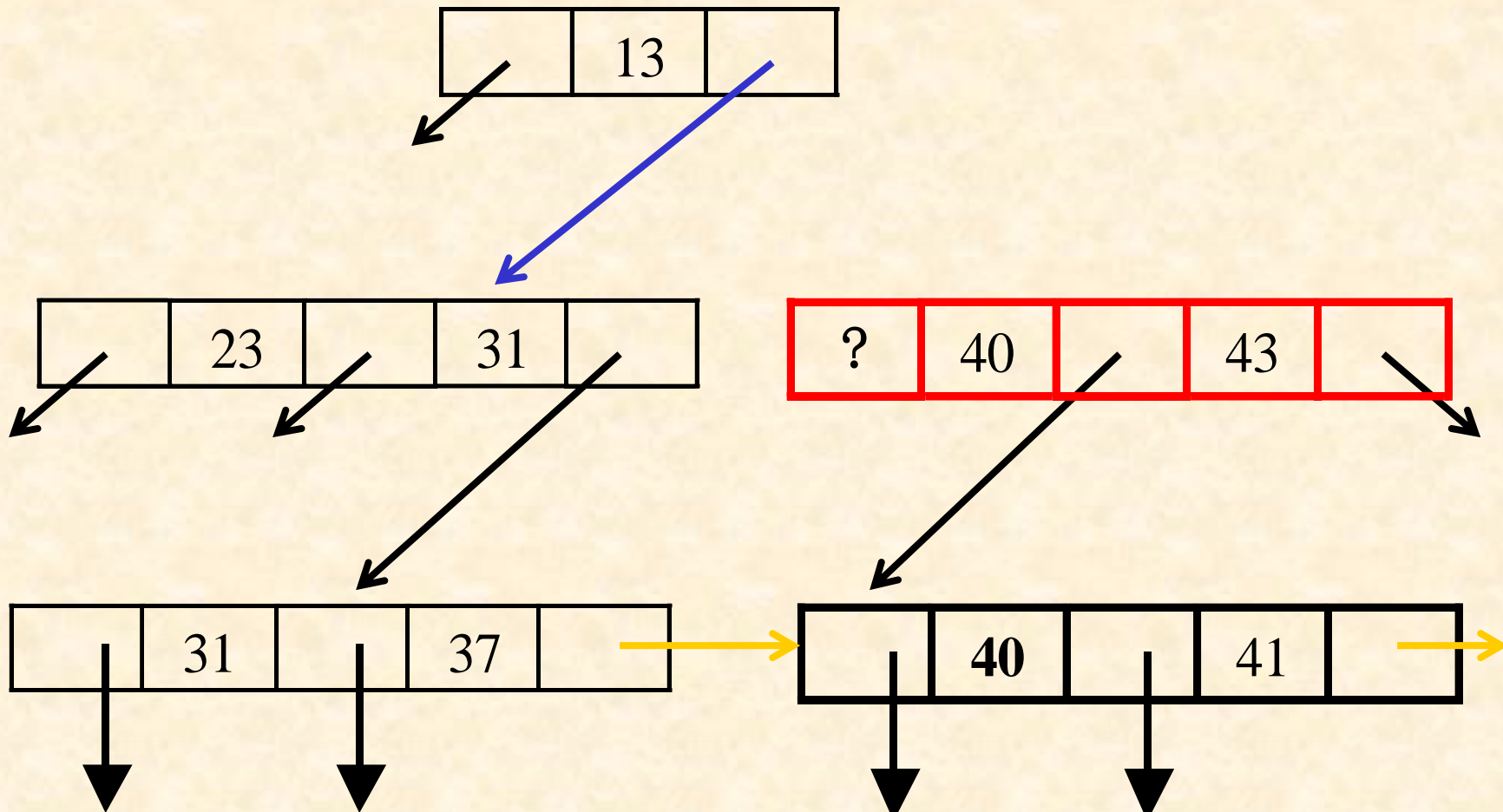
B+树中的插入算法（续）

- 4) 将新结点的第一个关键字值**40**以及新结点的指针插入到父结点中去。



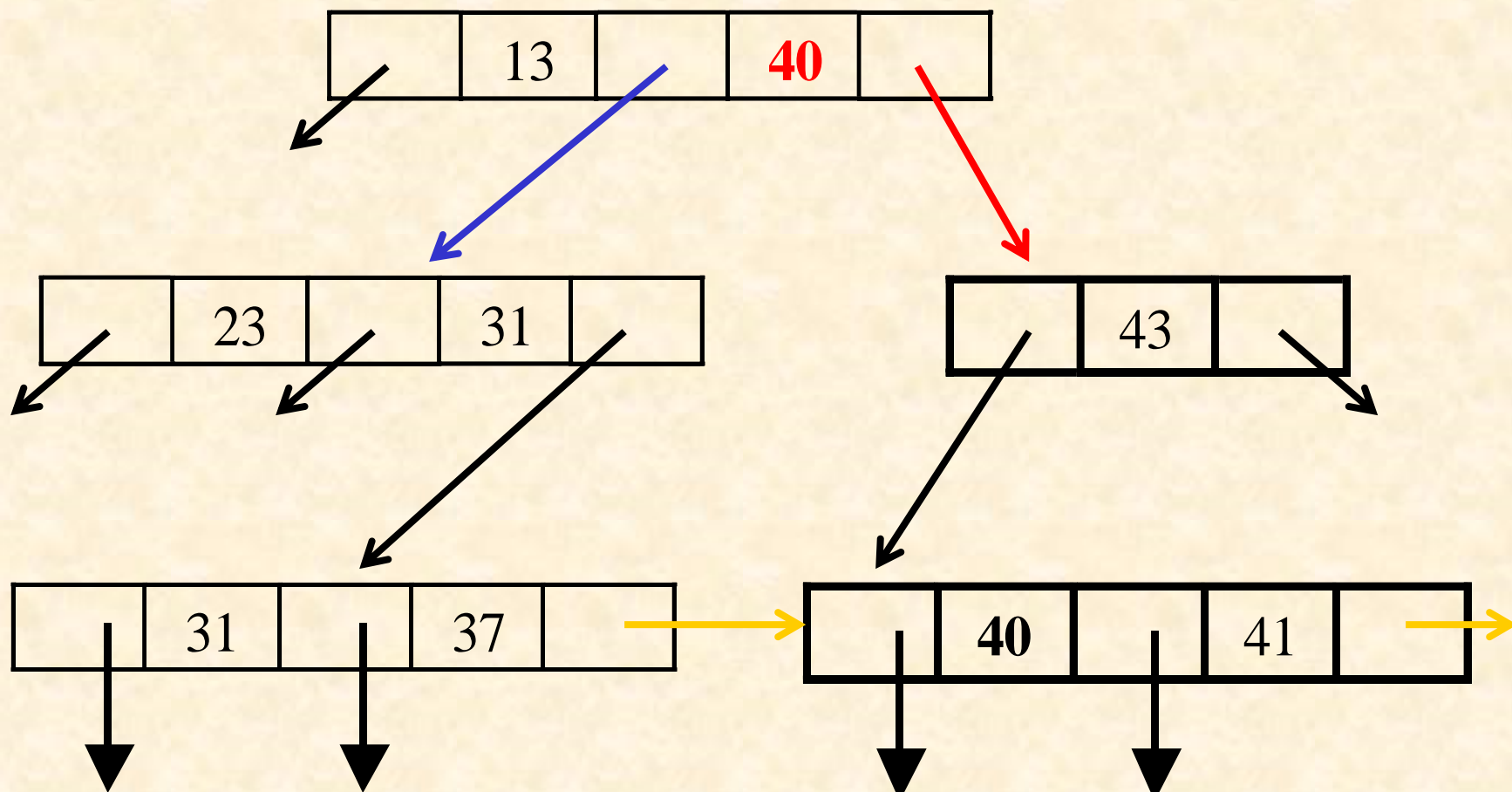
B+树中的插入算法（续）

5) 又引起父结点（内部结点）的分裂

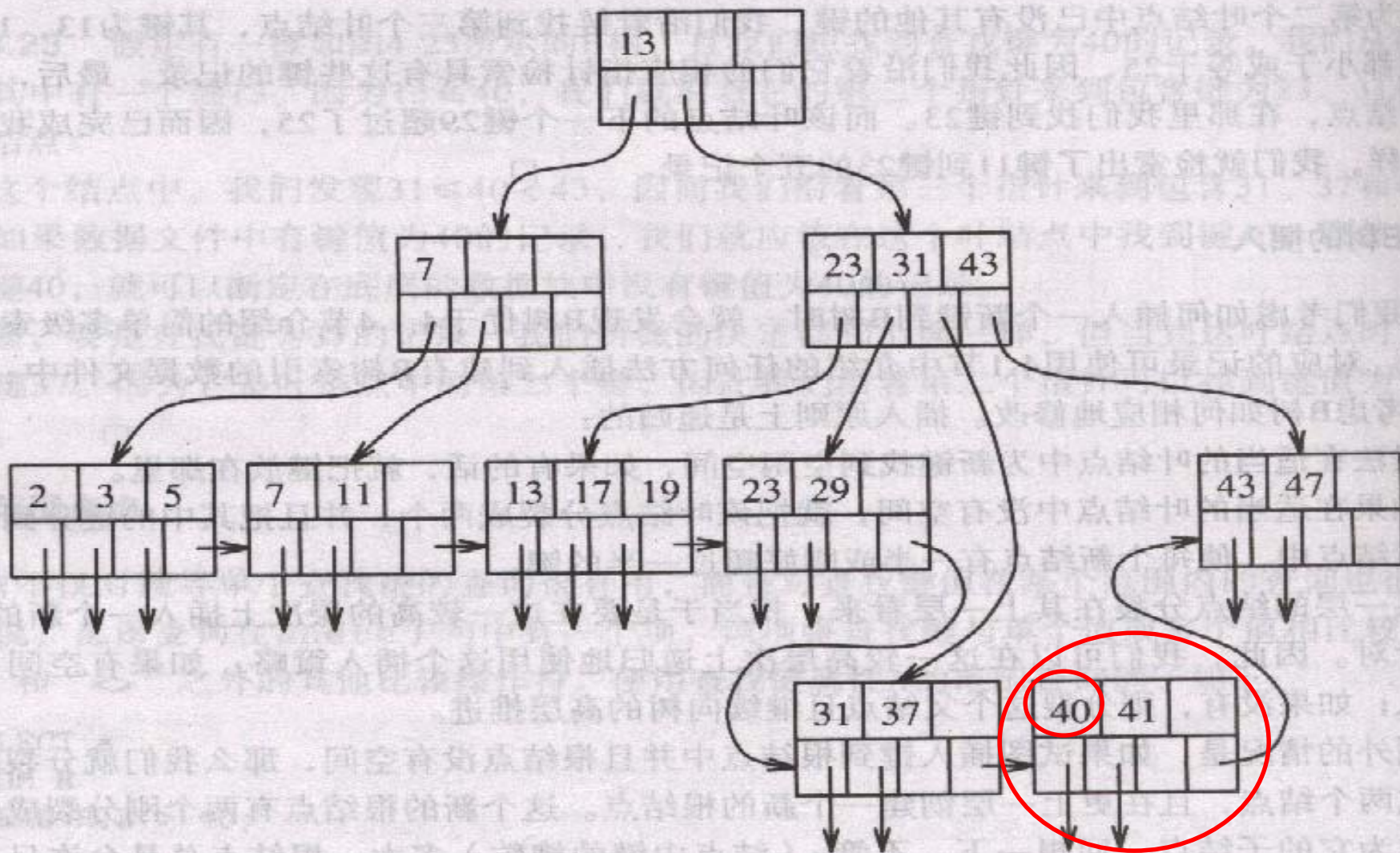


B+树中的插入算法（续）

- 6) 因父结点的分裂而导致在更高层次上的插入操作。
注意：内部结点分裂与叶结点分裂的区别。

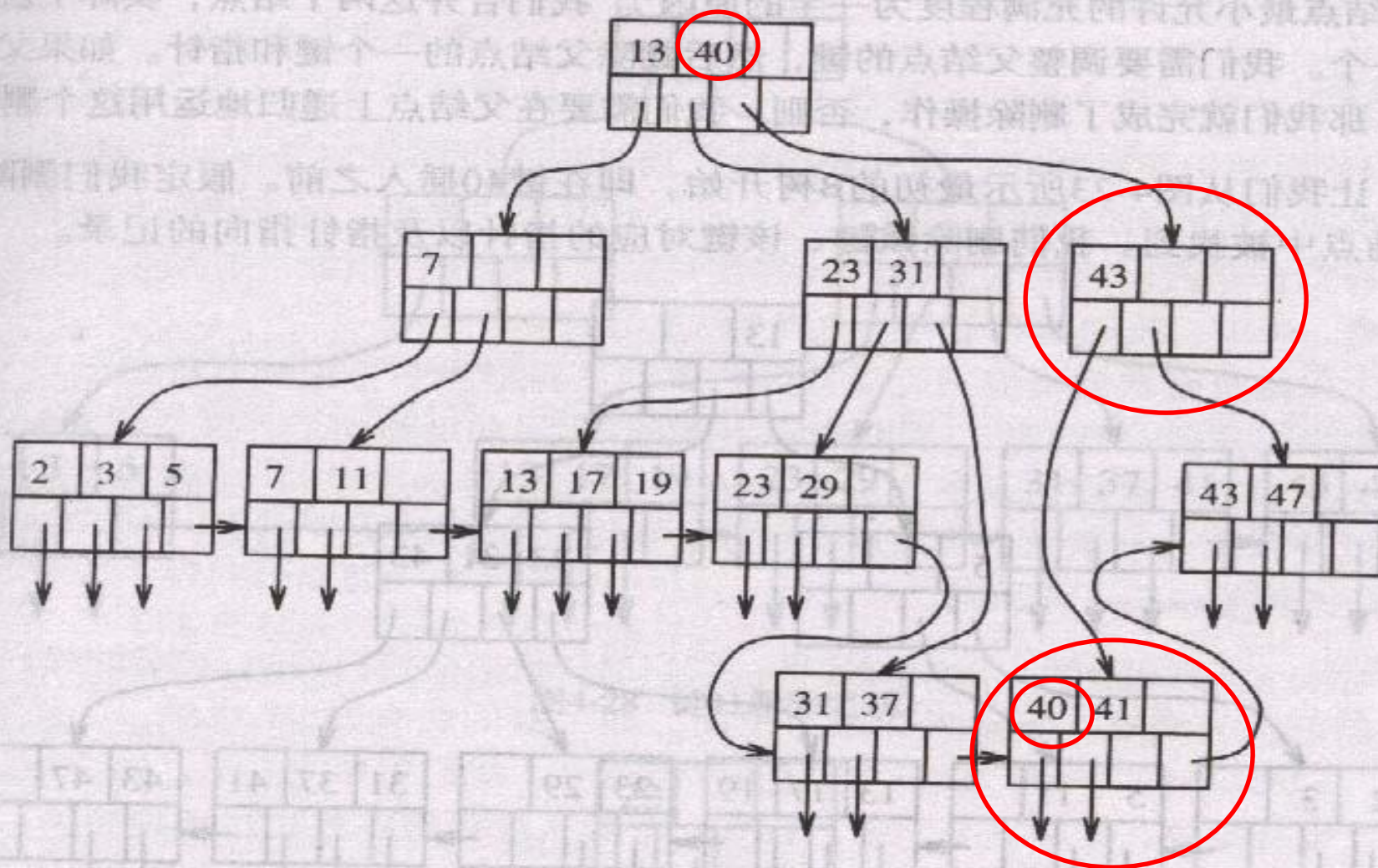


B+树中的插入算法（续）



例7：键40插入之初

B+树中的插入算法（续）



例7：键40插入完成后

4. B+树中的删除操作

算法3: 在B⁺树中删除一个关键字值为K的索引项

- 寻找删除操作的最初执行结点（叶结点）
- 删除关键字值为K的索引项
- 如果当前结点的关键字数目低于其下限时，在当前结点与其相邻的兄弟结点之间重新分配索引项，并修改父结点中的相关关键字的值。
- 如果其兄弟结点没有多余的索引项时，需要将相邻的两个兄弟结点合并为一个结点，并引起在父结点中的删除操作。
- 当根结点中的索引项个数被减为0时，将形成新的根结点，并导致树的高度减1。

B/B+树索引文件(续)

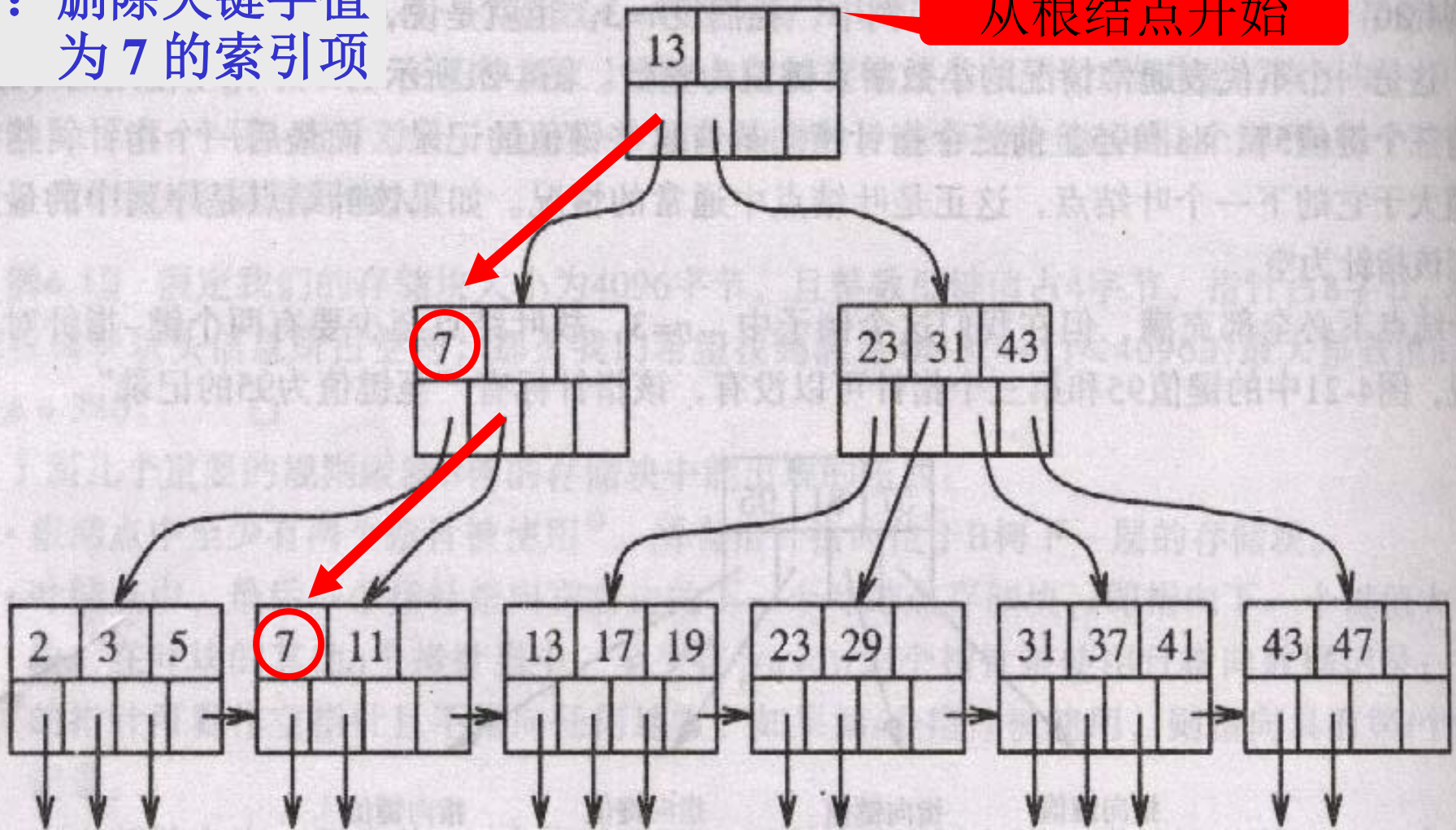
4. B+树中的删除算法 (续)

- 1) 利用算法1查找到关键字值为K的索引项所在的叶子结点N，若没有找到，则算法结束；
- 2) 在结点N中删除关键字值为K的索引项；
- 3) 若结点N中的关键字数目满足最低限度要求，则算法结束；
- 4) 检查与结点N相邻的兄弟结点（即具有同一个父结点的结点）中是否存在多余（即超过半数的）的键：
 - ❖ 若有，则从兄弟结点中移一个最近的关键字过来，同时需要修改其父结点中相应位置上的关键字值。
 - ❖ 若没有，则将结点N与其某个相邻的兄弟结点合并为一个结点（合并后的新结点中的索引关键字的个数不会超过其上限要求），并在其父结点中删除相应的索引项（转步骤 2））。
 - ❖ 在父结点中执行的删除操作又有可能引起父结点的合并。如果结点的合并操作一直波及到根结点，则会减低树的高度。

B/B+树索引文件(续)

例：删除关键字值为7的索引项

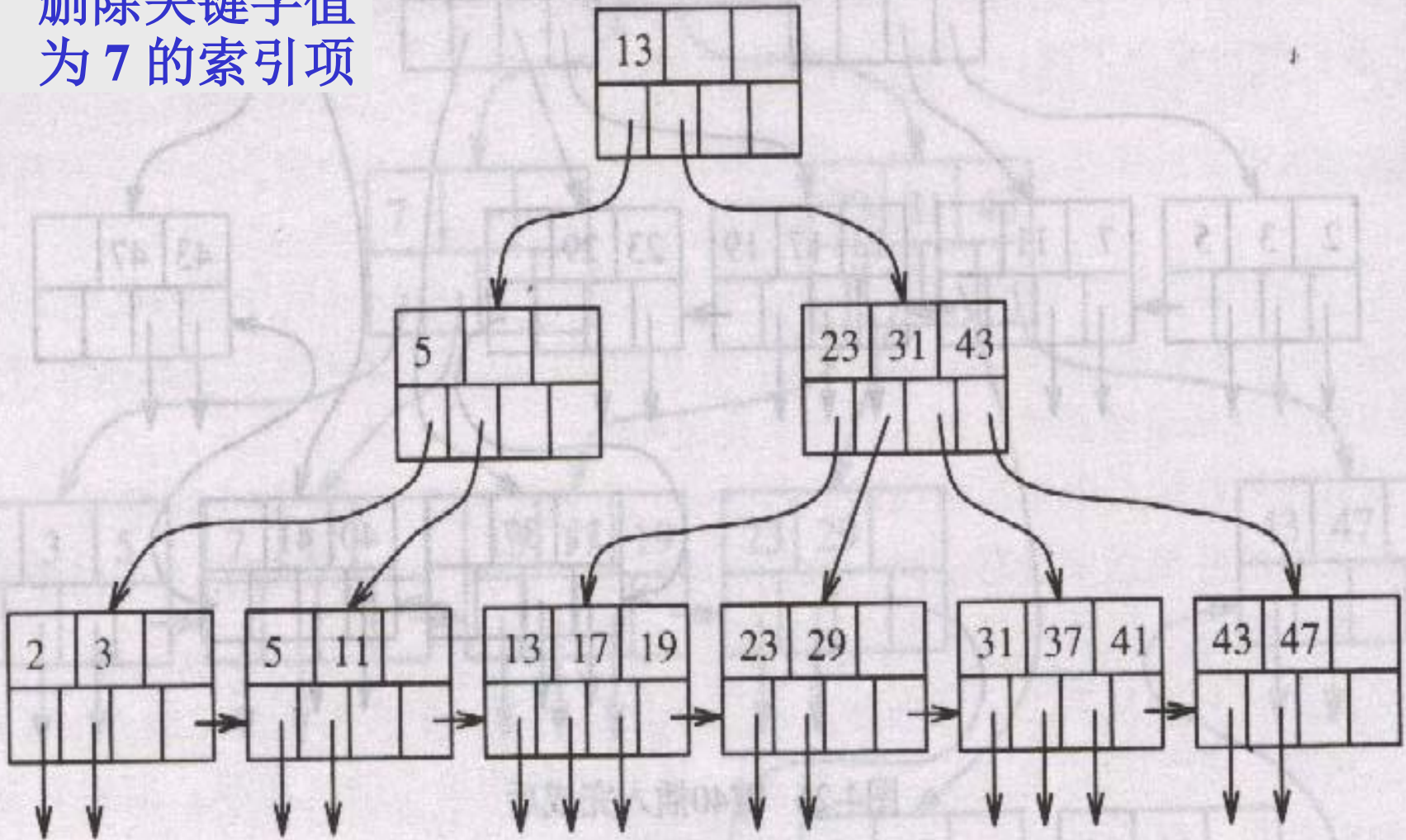
从根结点开始



首先寻找关键字值等于7的索引项所在的叶结点

B/B+树索引文件(续)

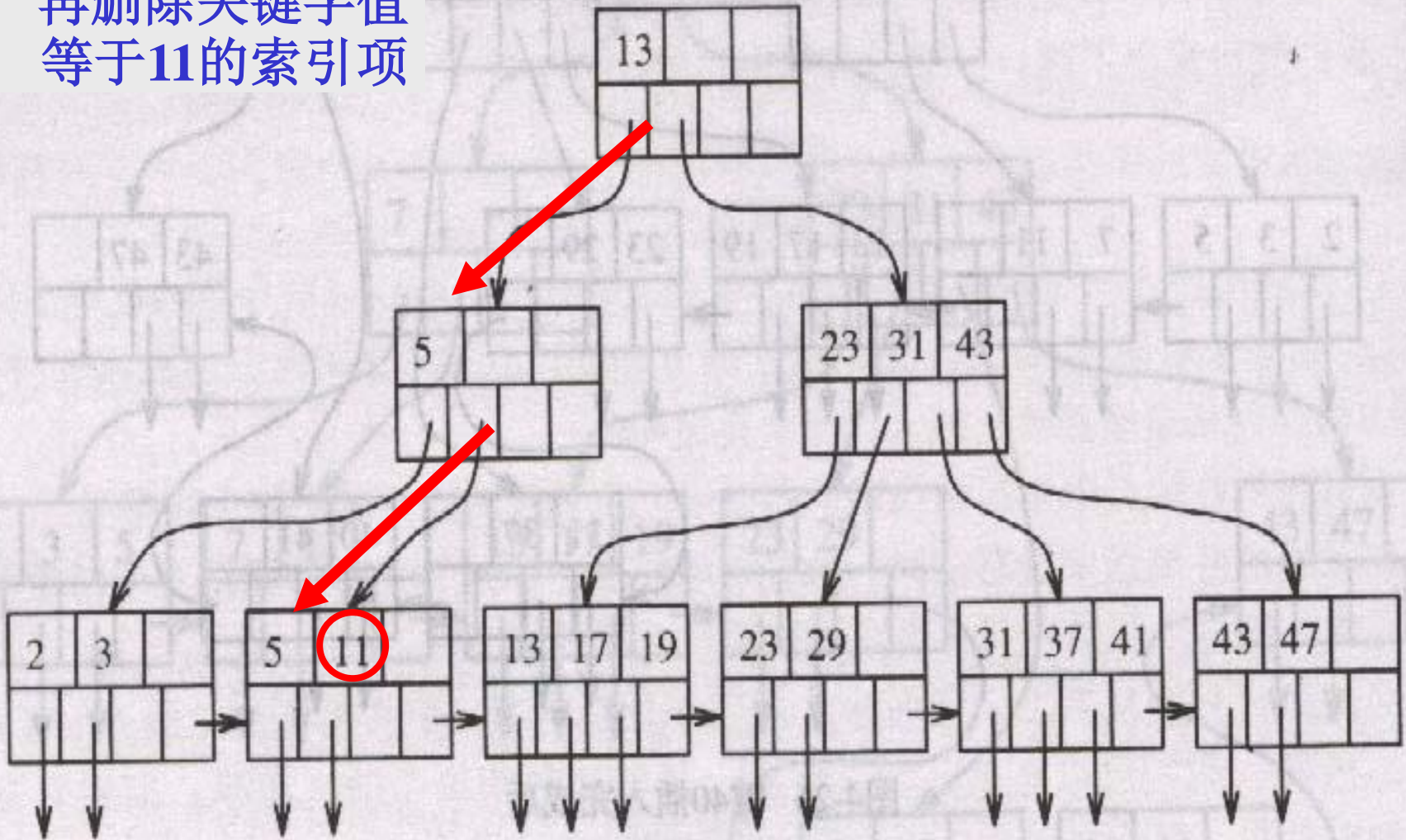
例：删除关键字值为7的索引项



删除关键字等于7的索引项后的结果

B/B+树索引文件(续)

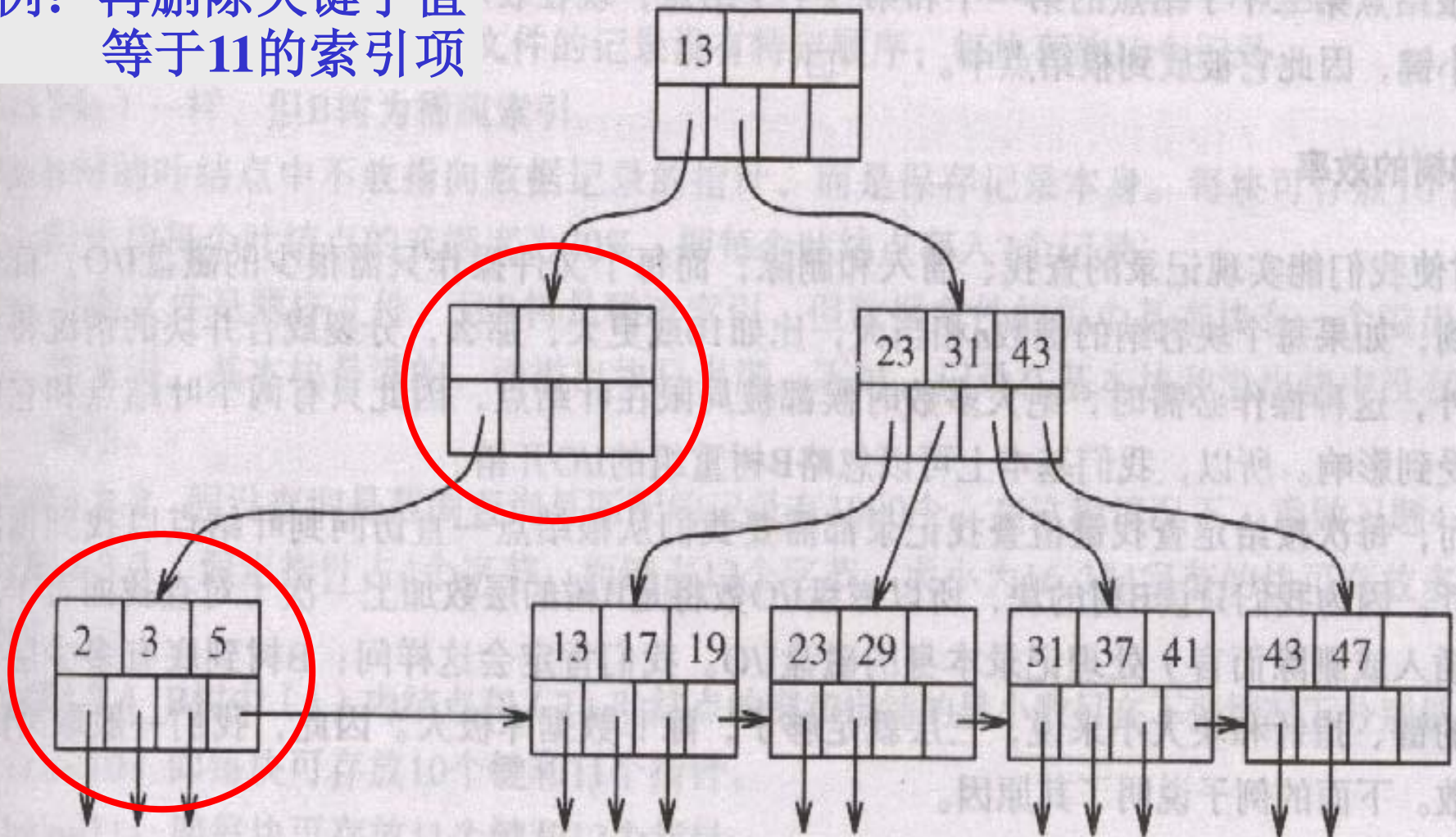
例：再删除关键字值
等于11的索引项



寻找关键字值等于11的索引项所在的叶结点

B/B+树索引文件(续)

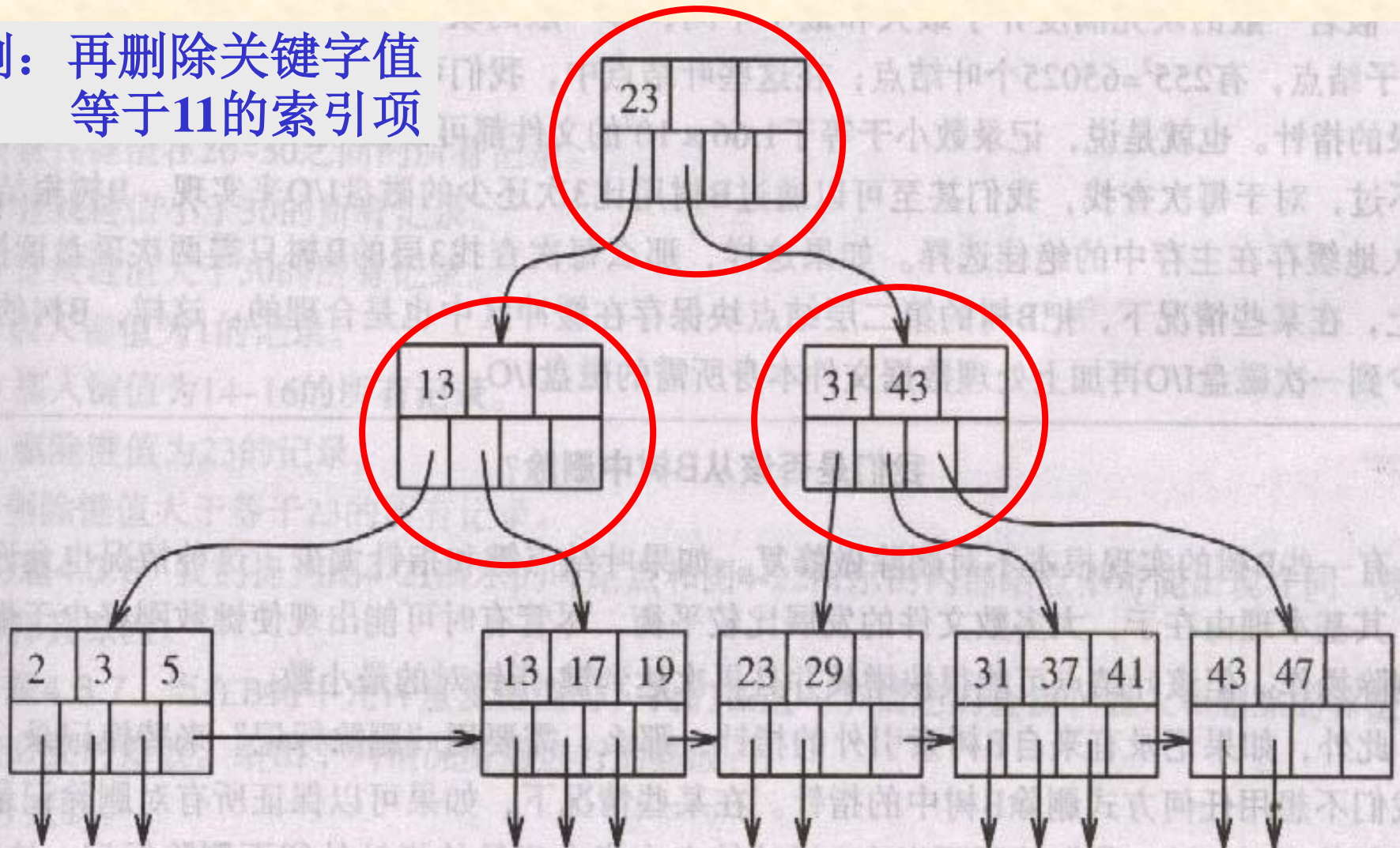
例：再删除关键字值
等于11的索引项



删除关键字11并合并相邻叶结点后的最初结果

B/B+树索引文件(续)

例：再删除关键字值
等于11的索引项

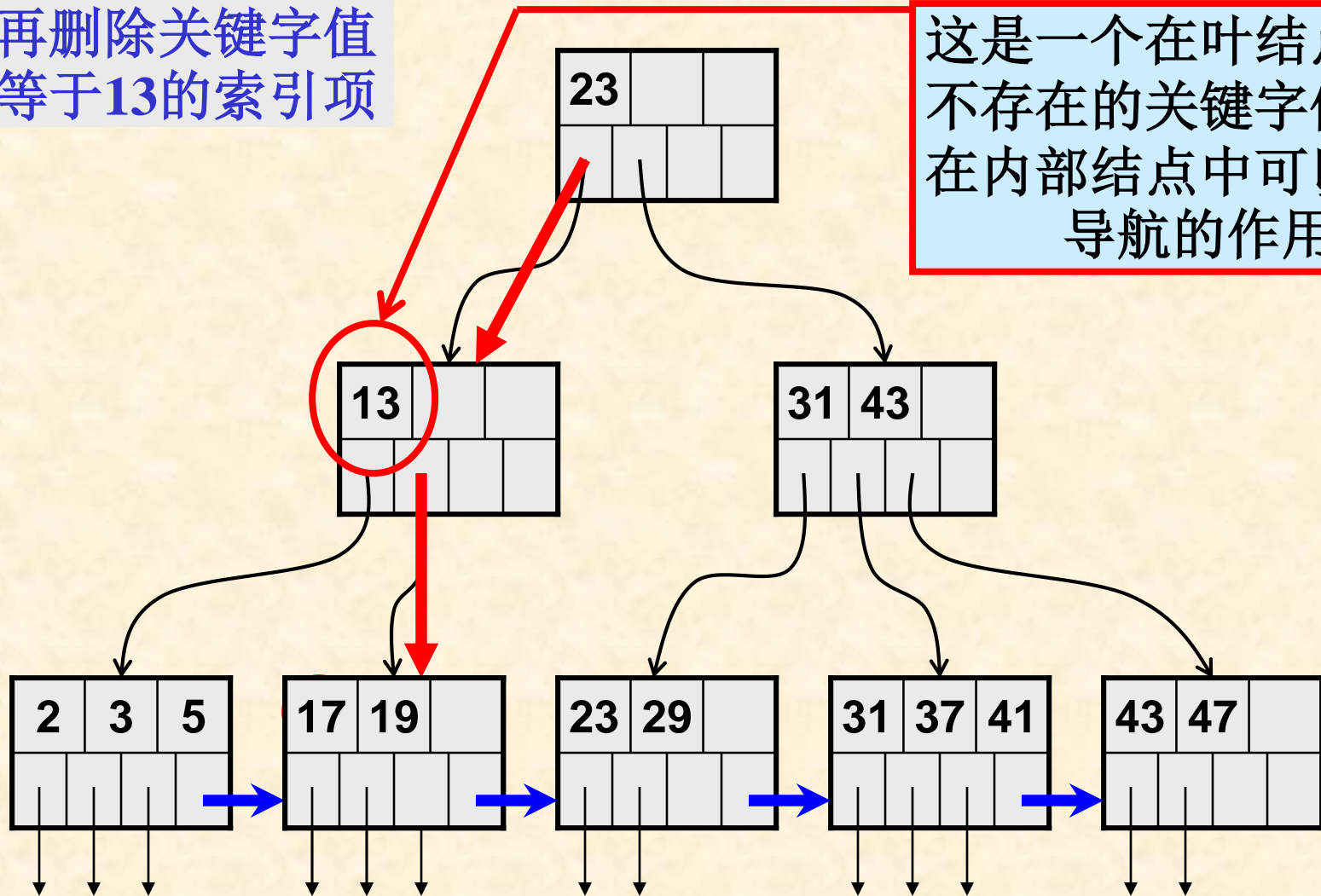


整个删除操作全部完成后的结果

B/B+树索引文件(续)

例：再删除关键字值
等于13的索引项

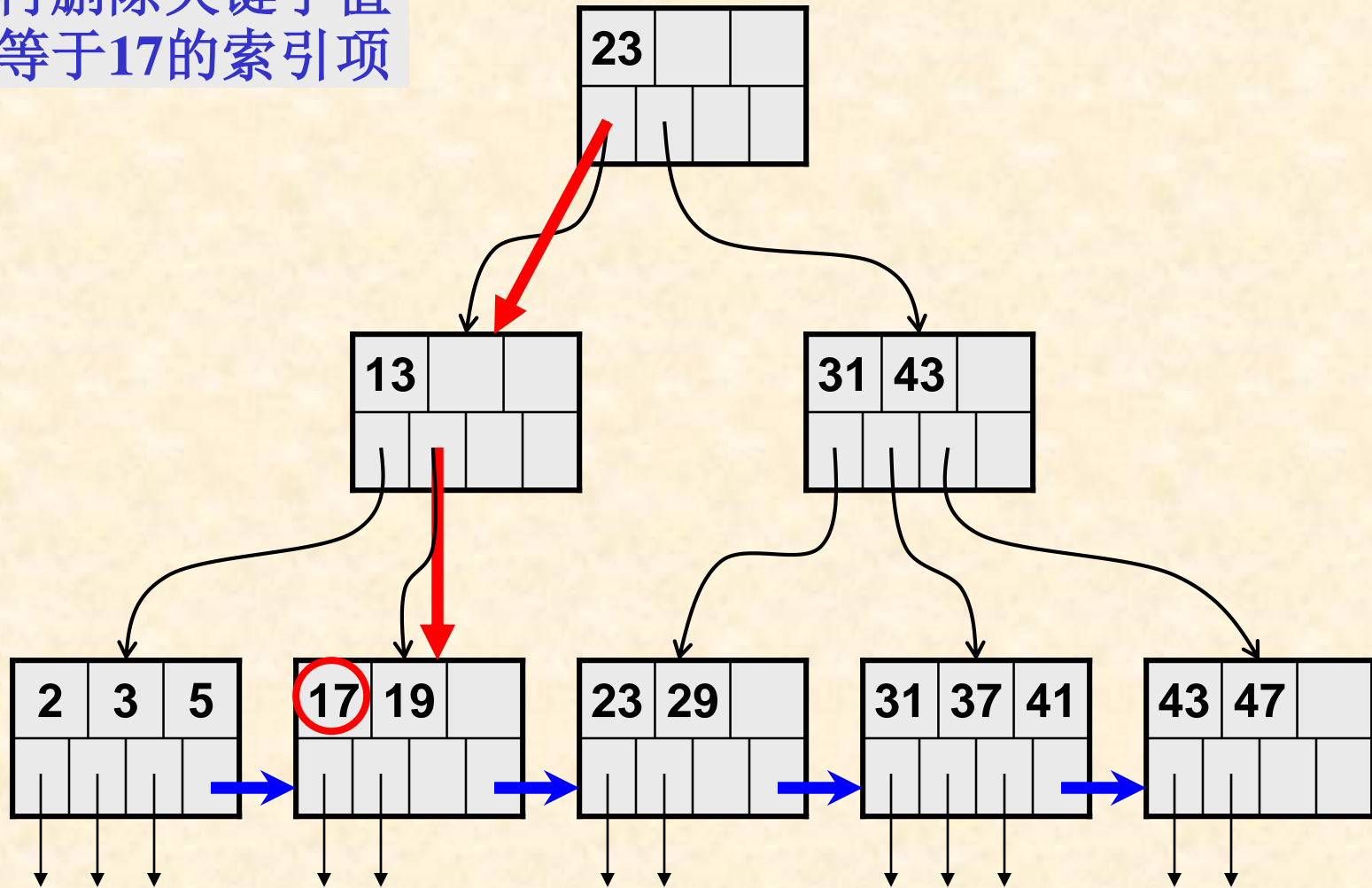
这是一个在叶结点中并不存在的
关键字值，但在内部结点中可以起到
导航的作用



删除关键字值等于13的索引项后的结果

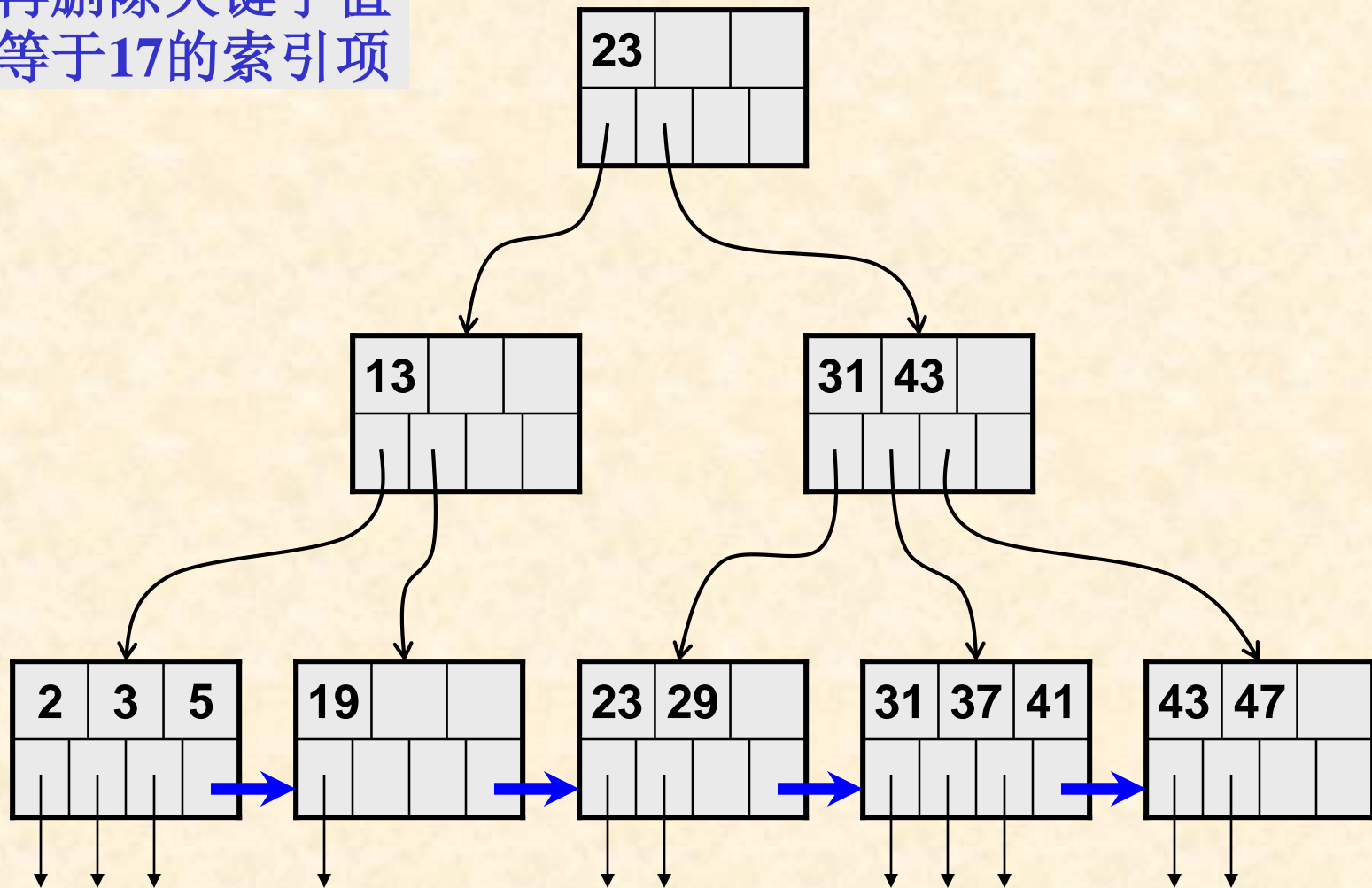
B/B+树索引文件(续)

例：再删除关键字值
等于17的索引项



B/B+树索引文件(续)

例：再删除关键字值
等于17的索引项

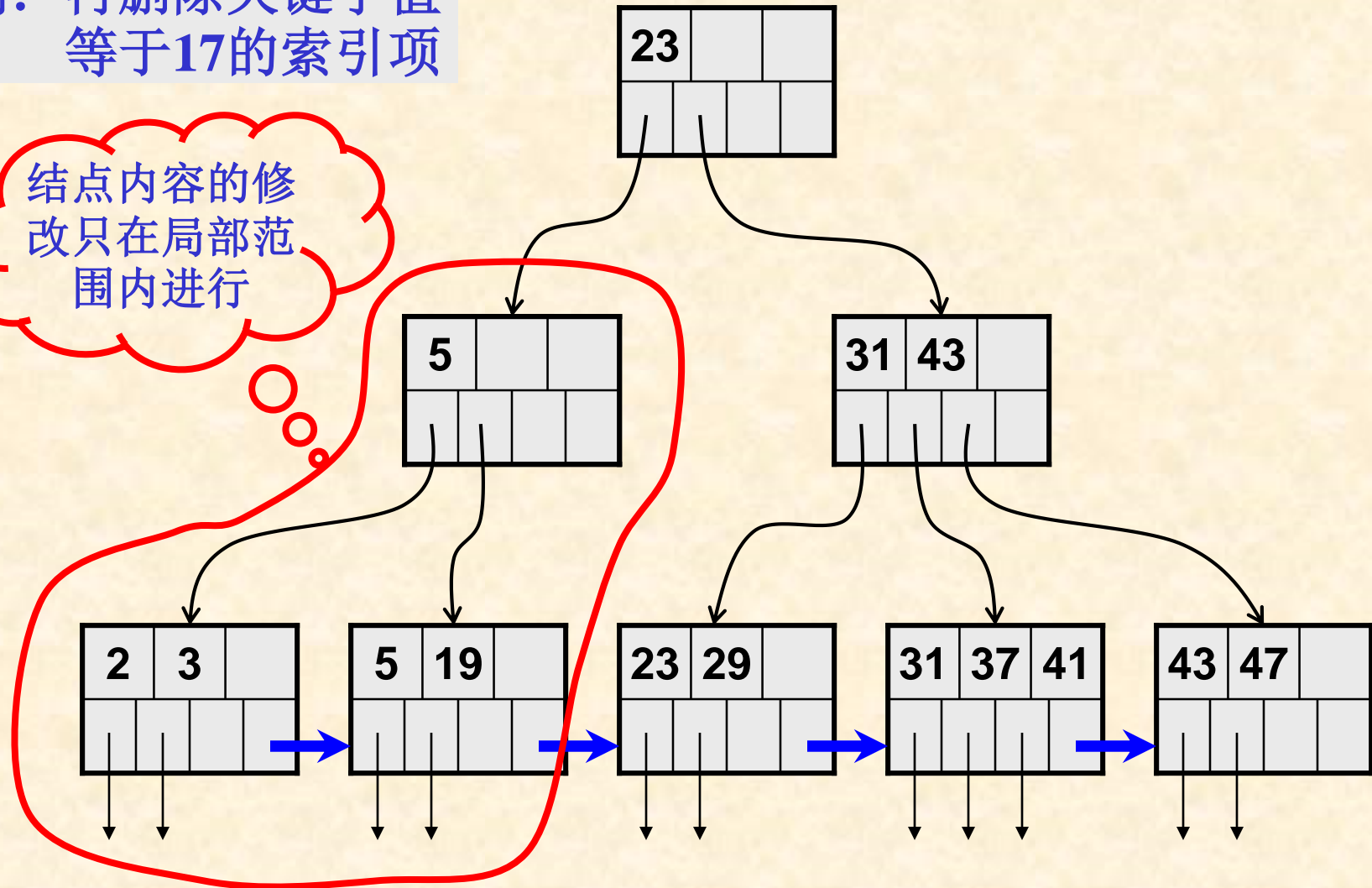


删除关键字值等于17的索引项后的最初情况

B/B+树索引文件(续)

例：再删除关键字值
等于17的索引项

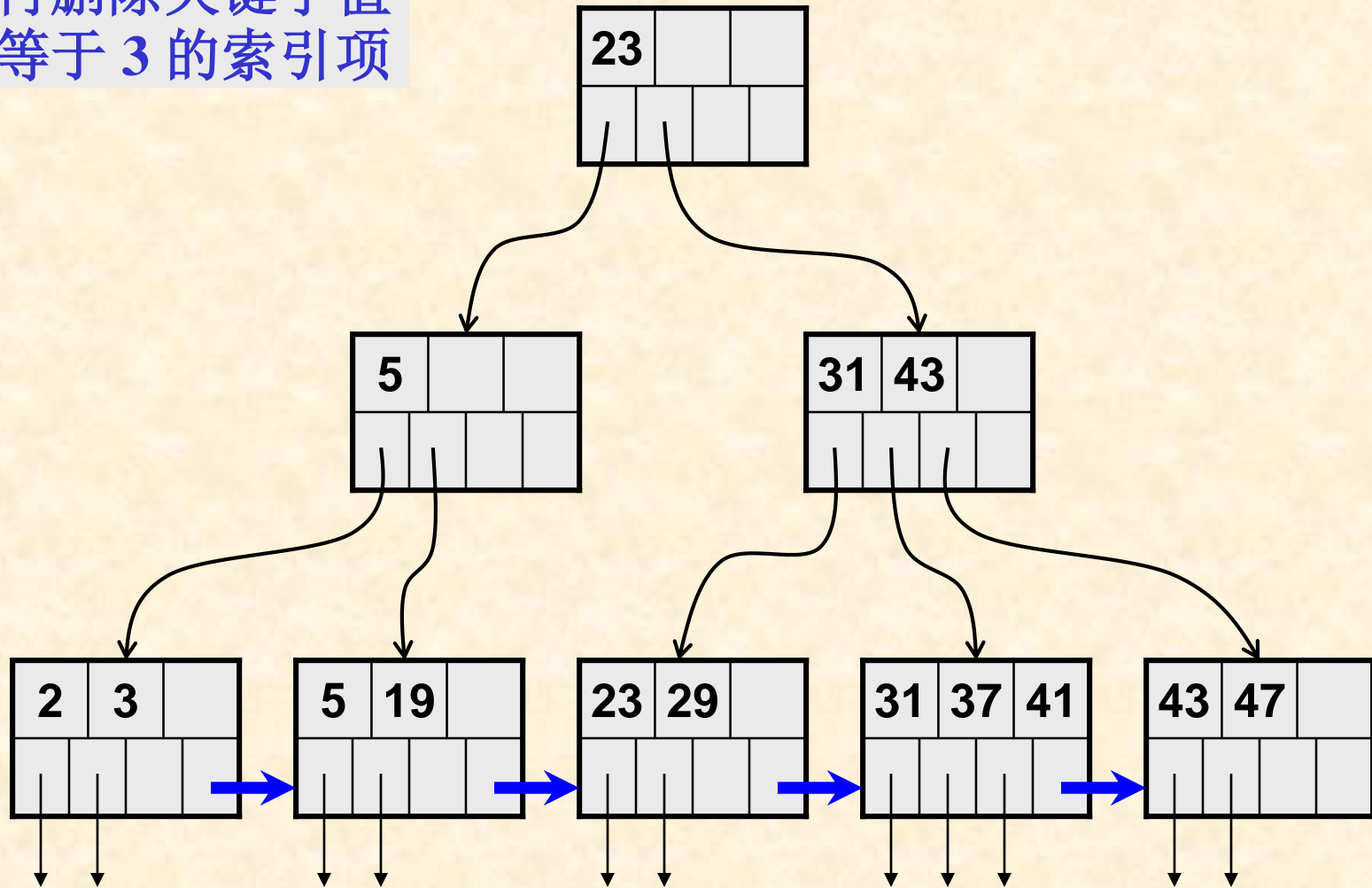
结点内容的修
改只在局部范
围内进行



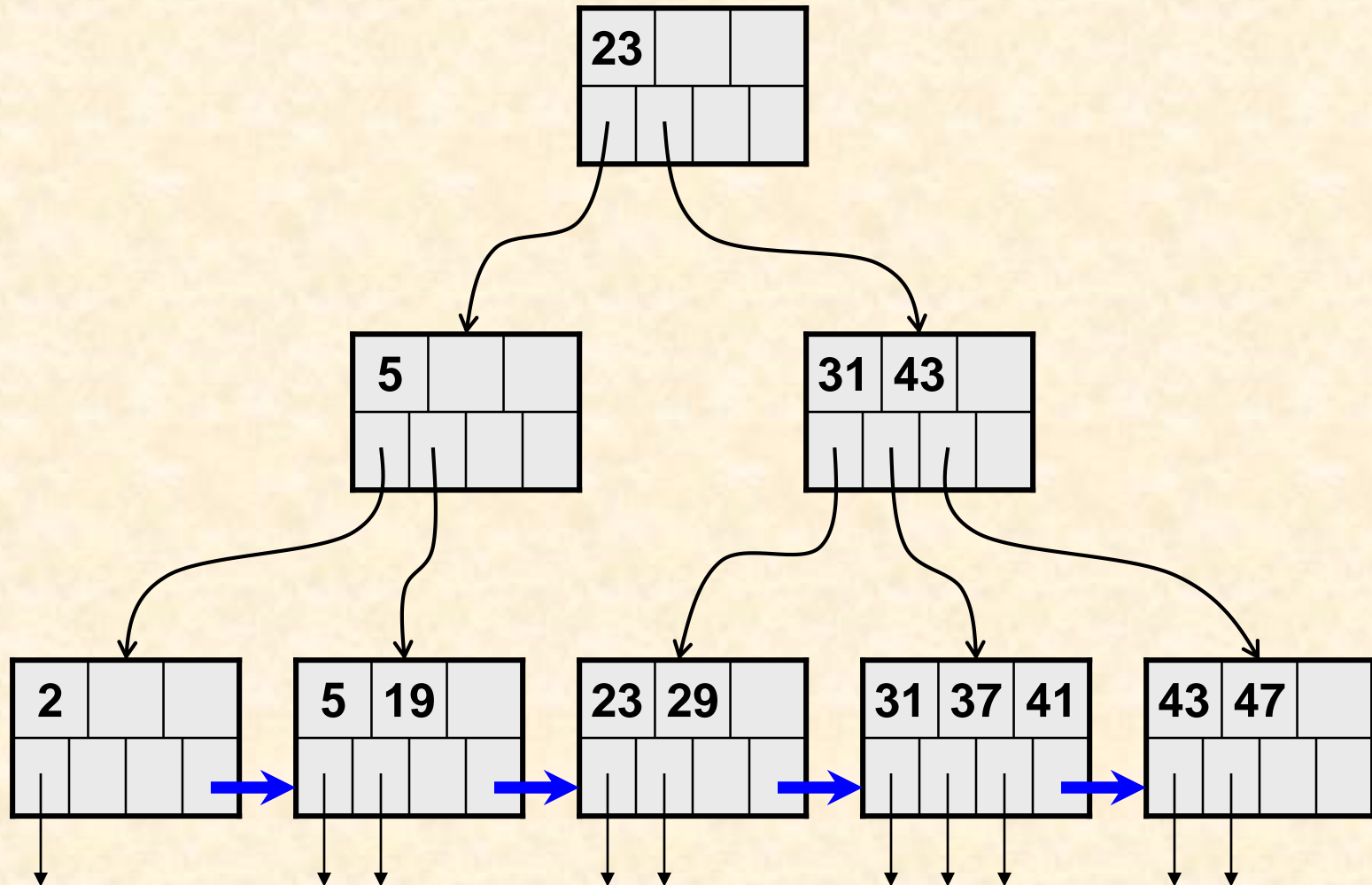
进行索引项迁移后的最终结果

B/B+树索引文件(续)

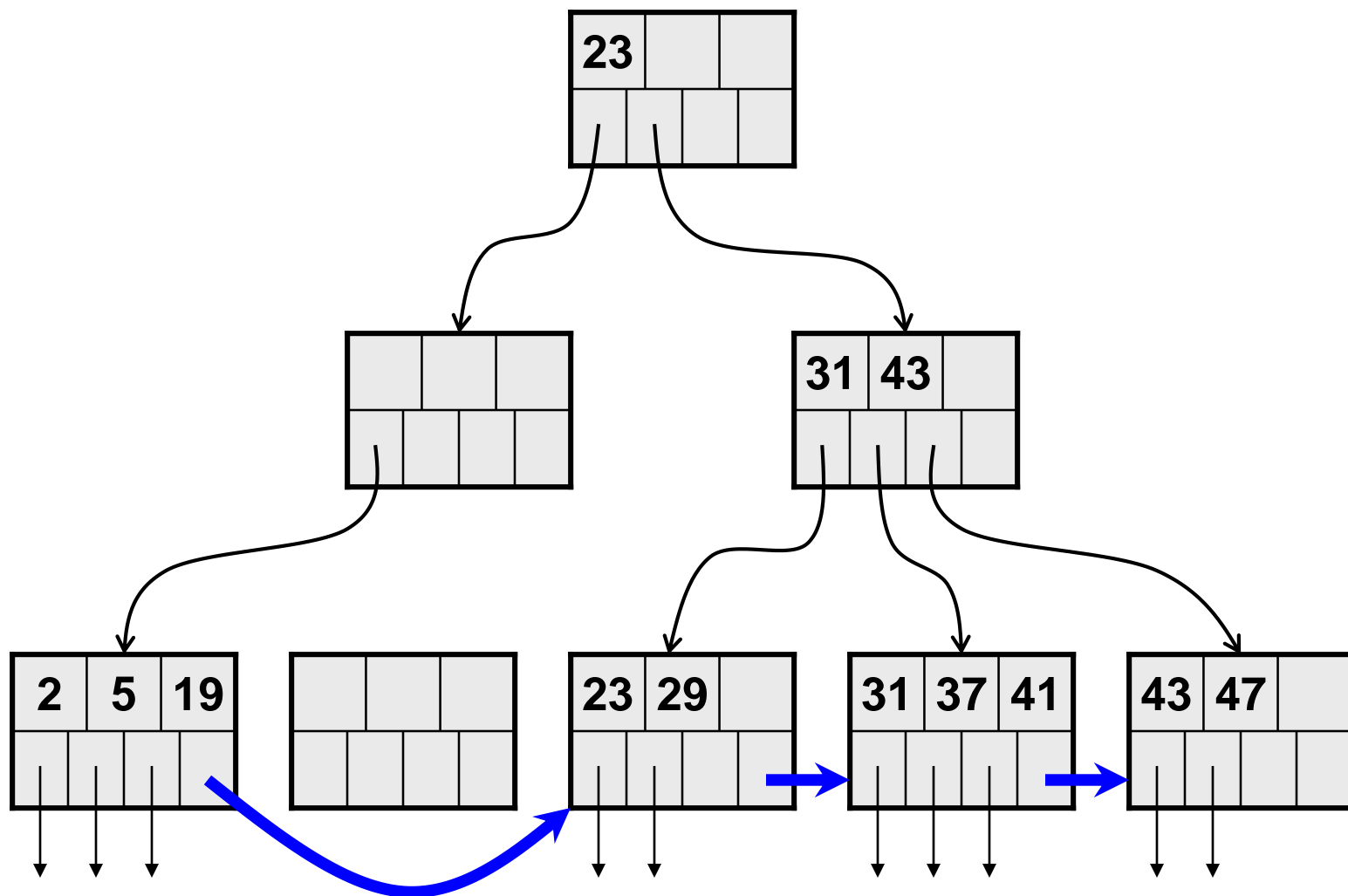
例：再删除关键字值
等于 3 的索引项



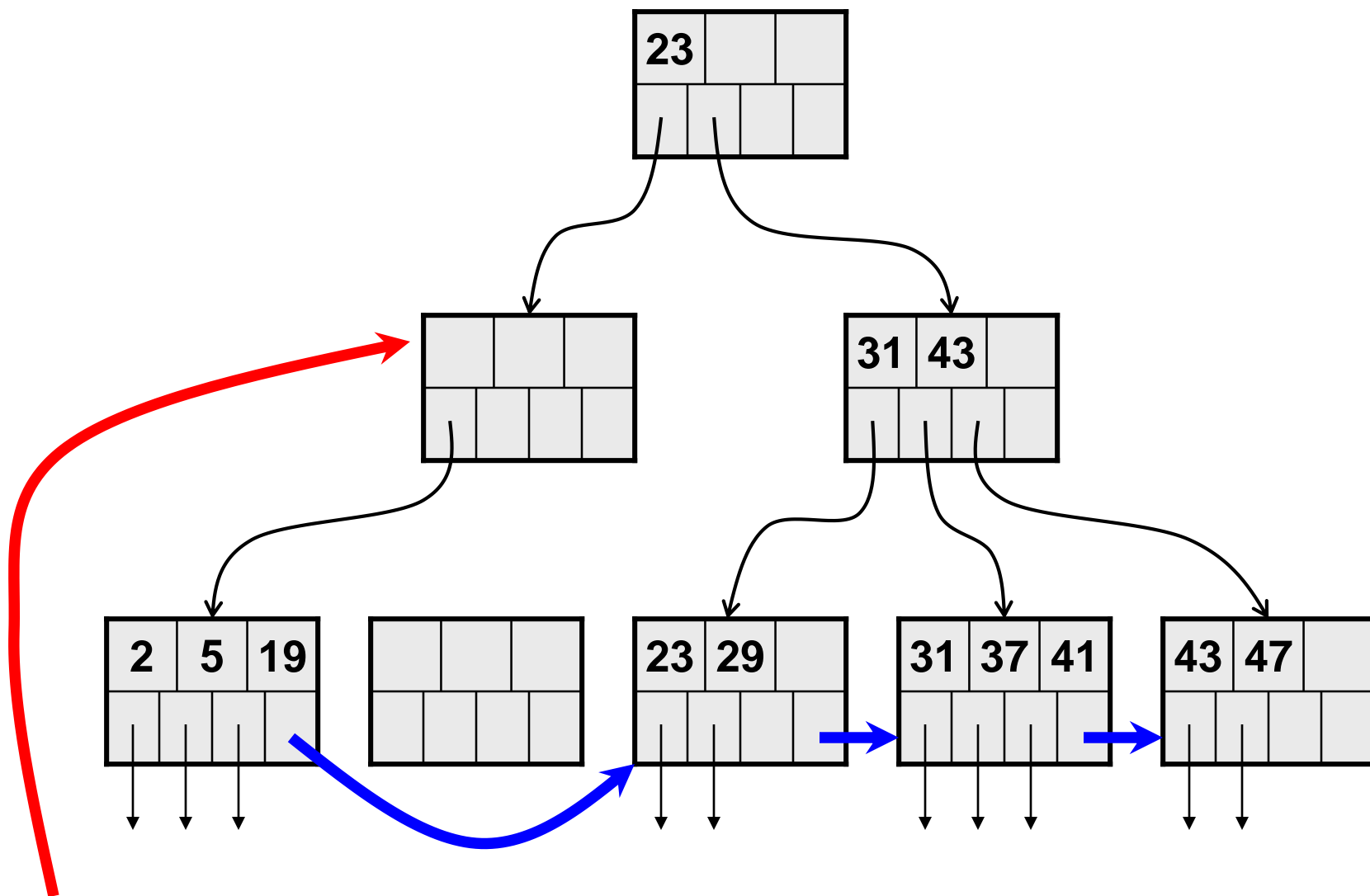
B/B+树索引文件(续)



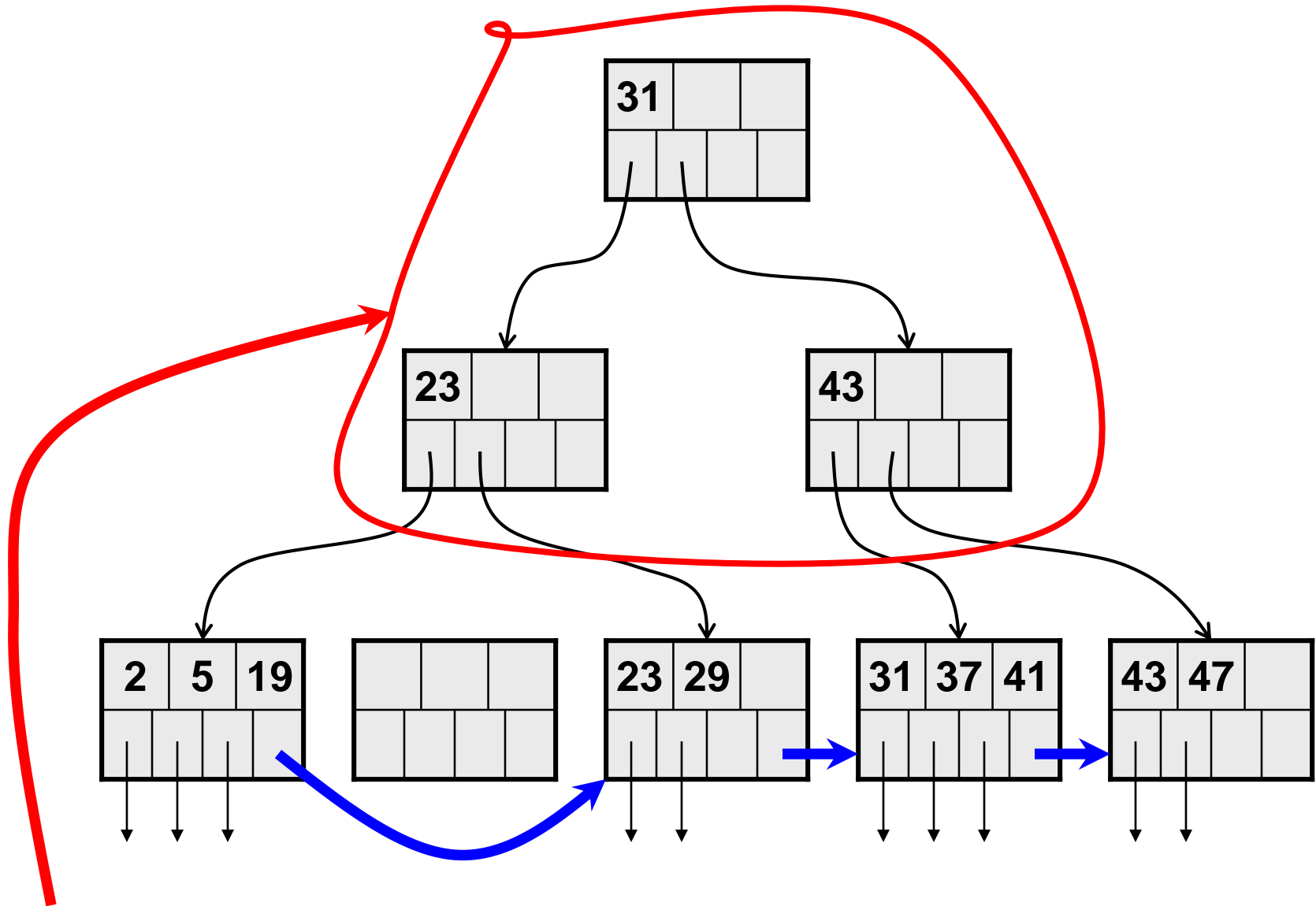
删除关键字值等于 3 的索引项后的最初结果



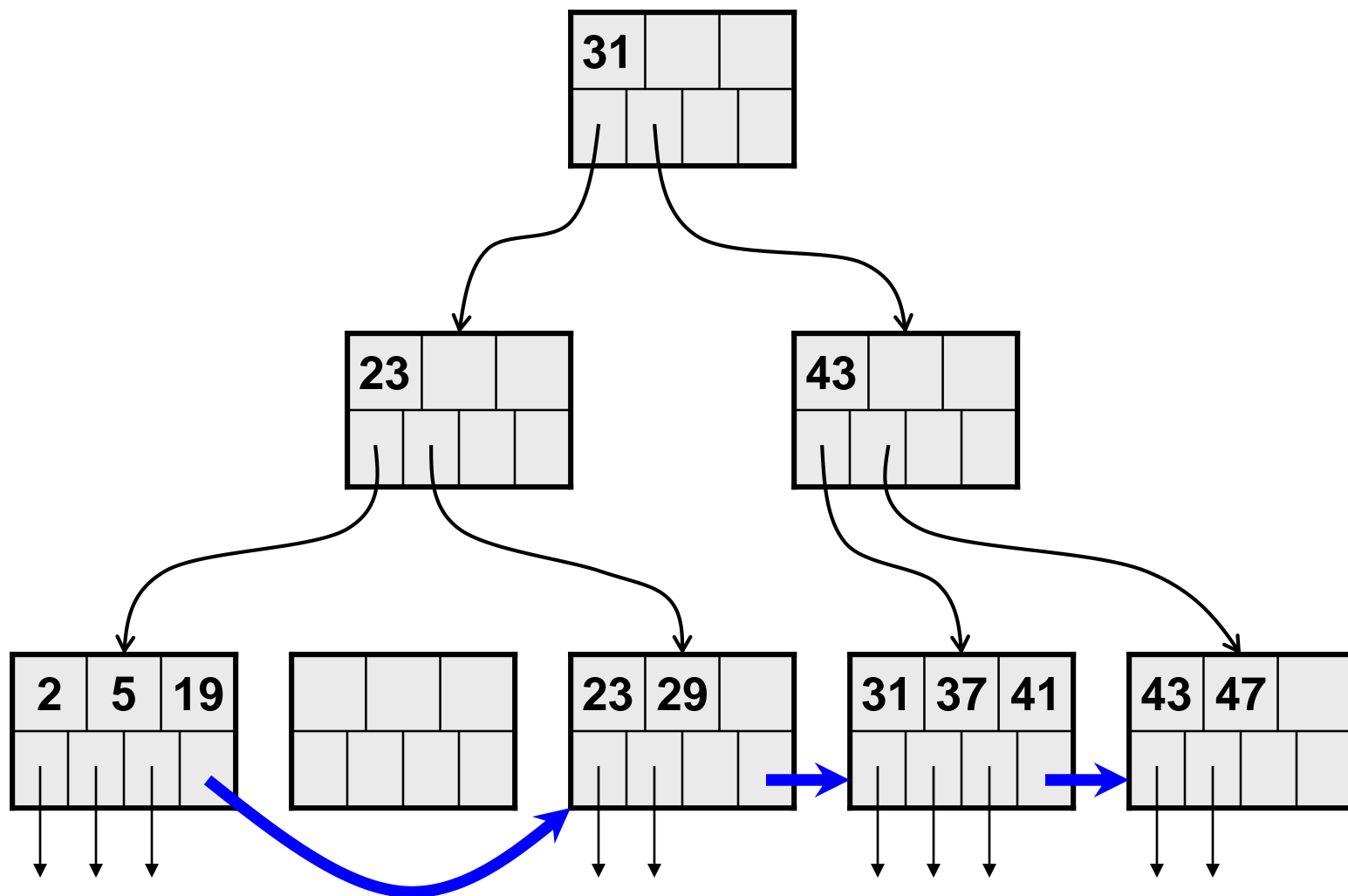
合并相邻的兄弟叶结点（将右叶结点中的索引项移到左叶结点中），抛弃已经被置空的叶结点（修改叶结点之间的指针），从而引起其父结点中的关键字值及子树指针的减1



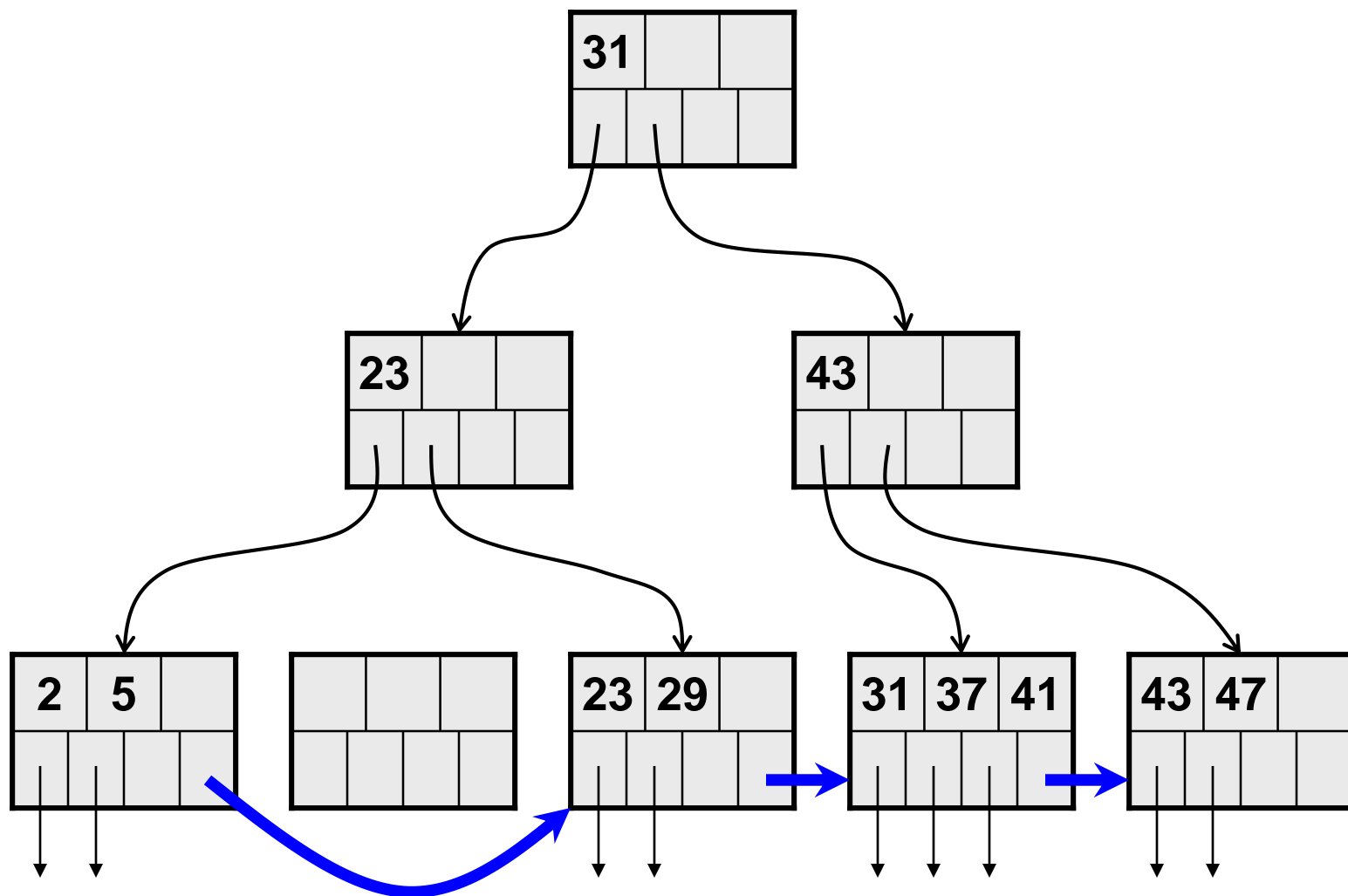
该结点中的关键字值的个数已经低于其下限值（0），需要从其相邻的某个兄弟结点中迁移一个关键字值过来



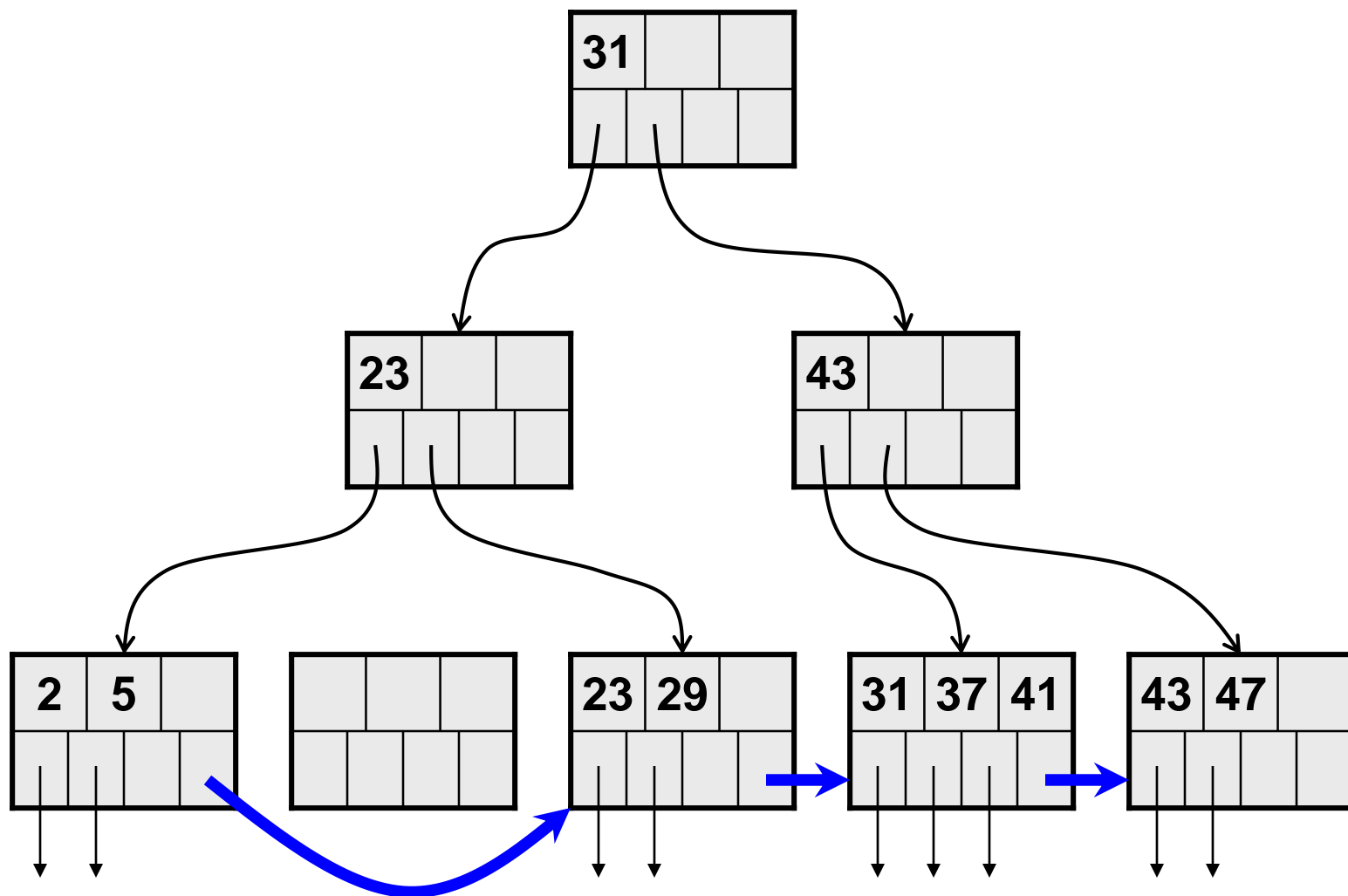
在内部结点中进行了索引项的迁移后的结果，被更新的仅限于这三个索引结点中的值



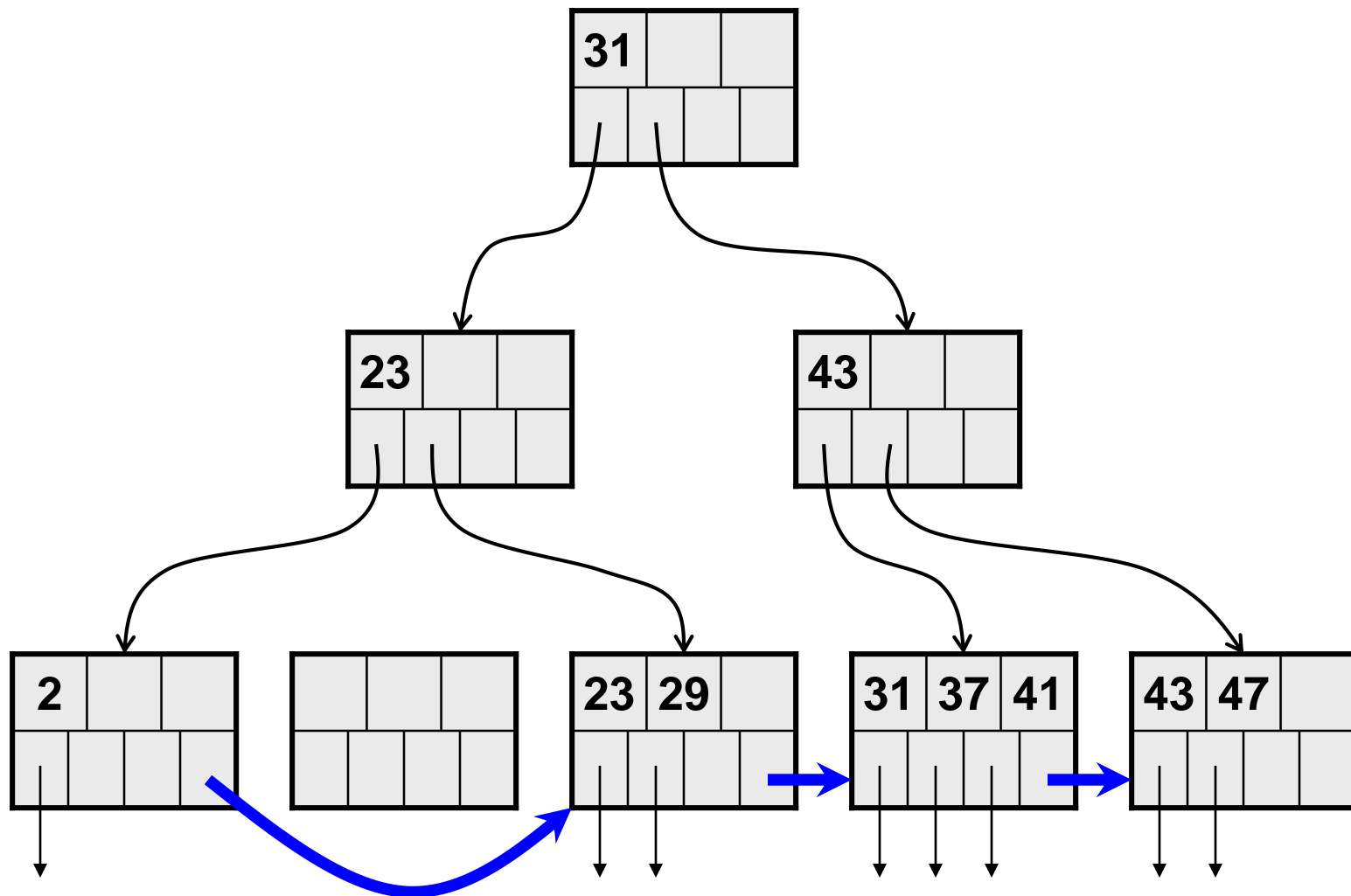
例：再删除关键字值等于 19 的索引项



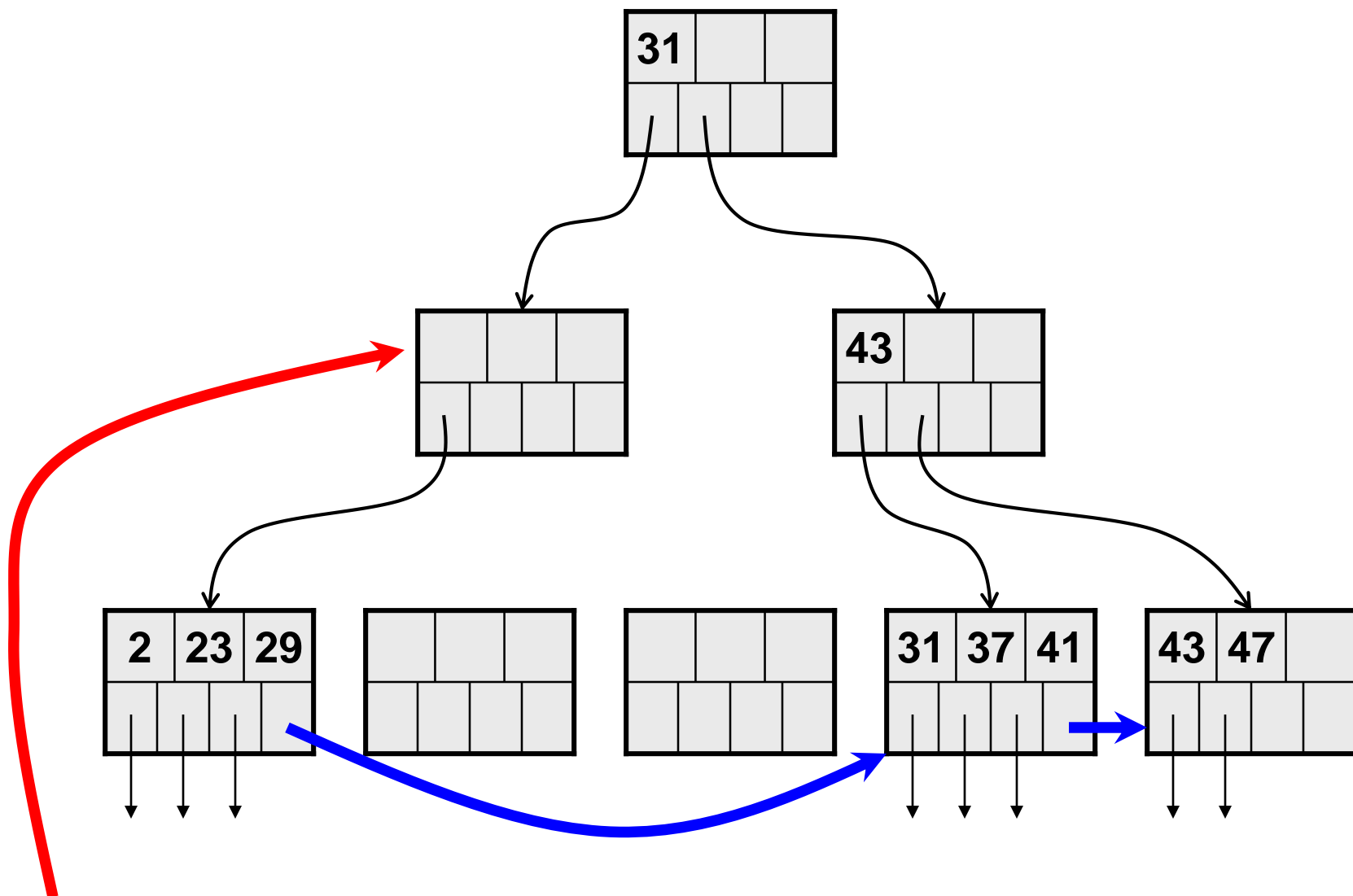
删除关键字值等于 19 的索引项后的结果



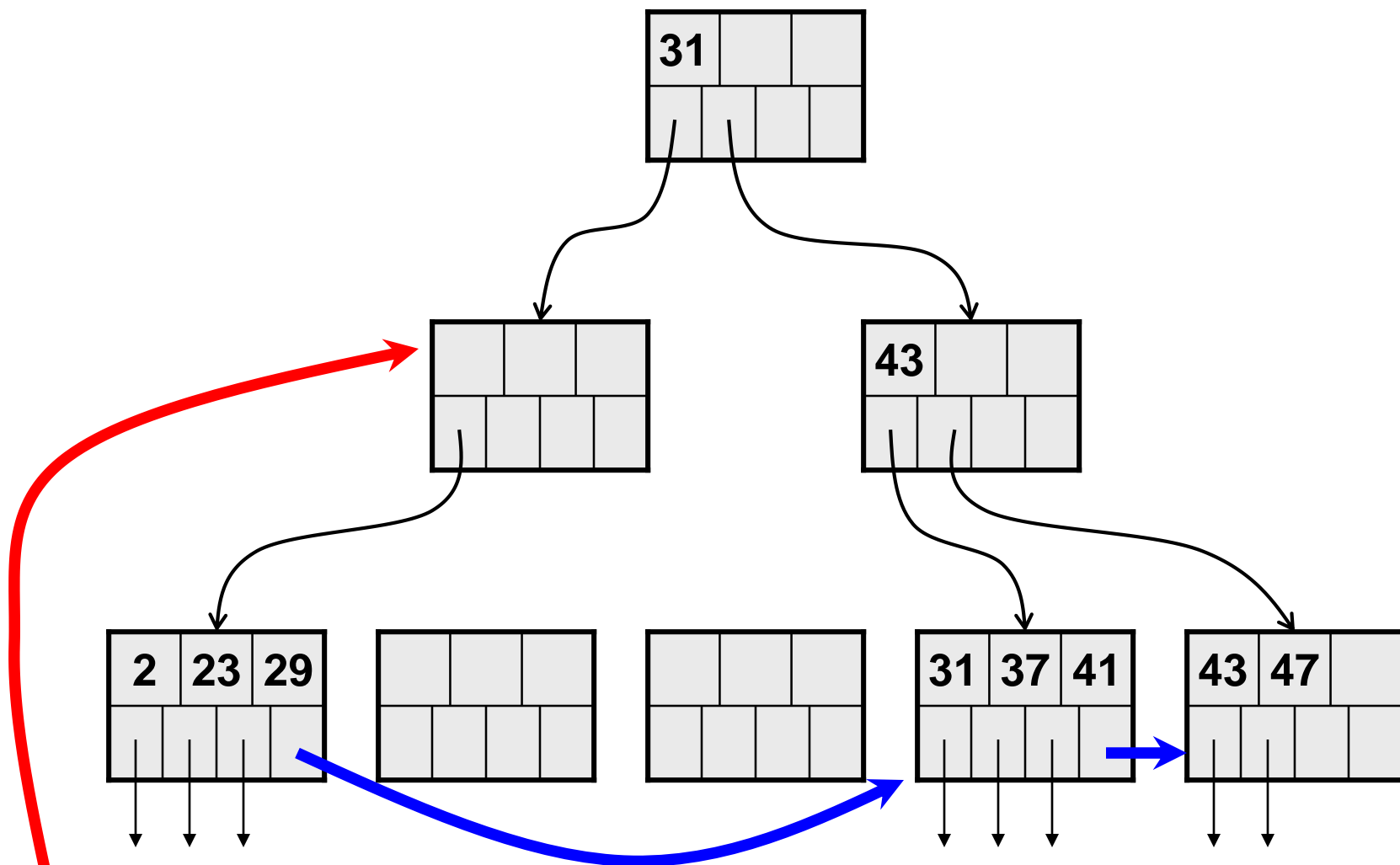
例：再删除关键字值等于 5 的索引项



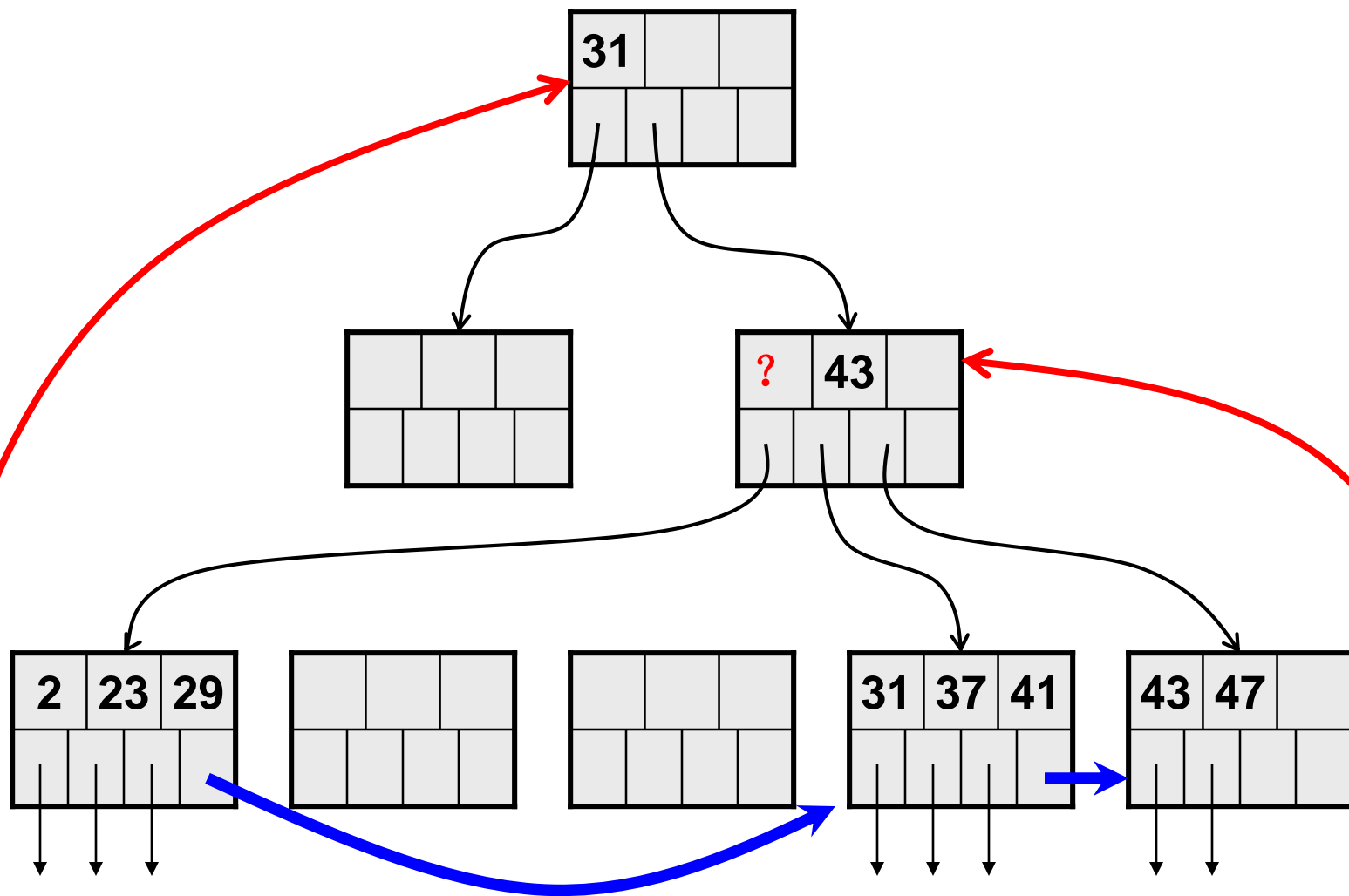
删除关键字值等于 5 的索引项后的最初结果。其中：第一个叶子结点中的关键字值的个数已经低于其下限值（2），且其相邻的兄弟结点中也没有多余的关键字值，因而必须合并它们



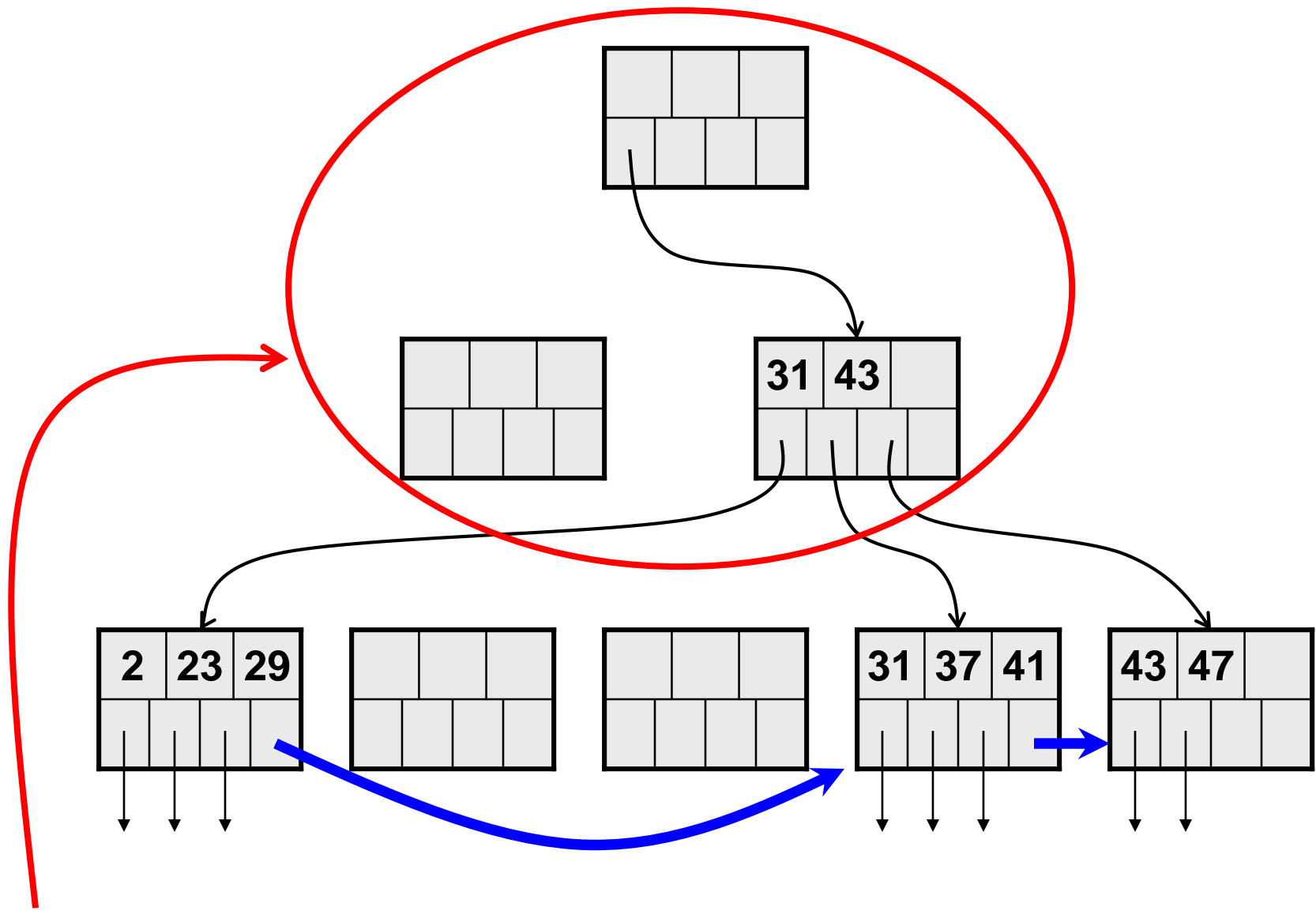
结点的合并操作必然会引起其上一层父结点中的索引项的删除操作。



该结点中的关键字值的个数也已经低于其下限值（0），且其兄弟结点中也没有多余的关键字值，因而必须与某个相邻的兄弟结点进行合并

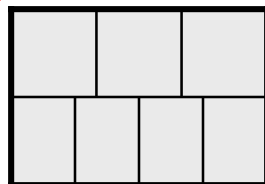


1. 内部结点在合并后需要增加一个索引关键字值，以满足子树指针等于关键字值的个数+1的要求。
2. 同时需要将指向被清空结点的子树指针从其父结点中删除（当然也需要同时从其父结点中删除一个索引关键字值）

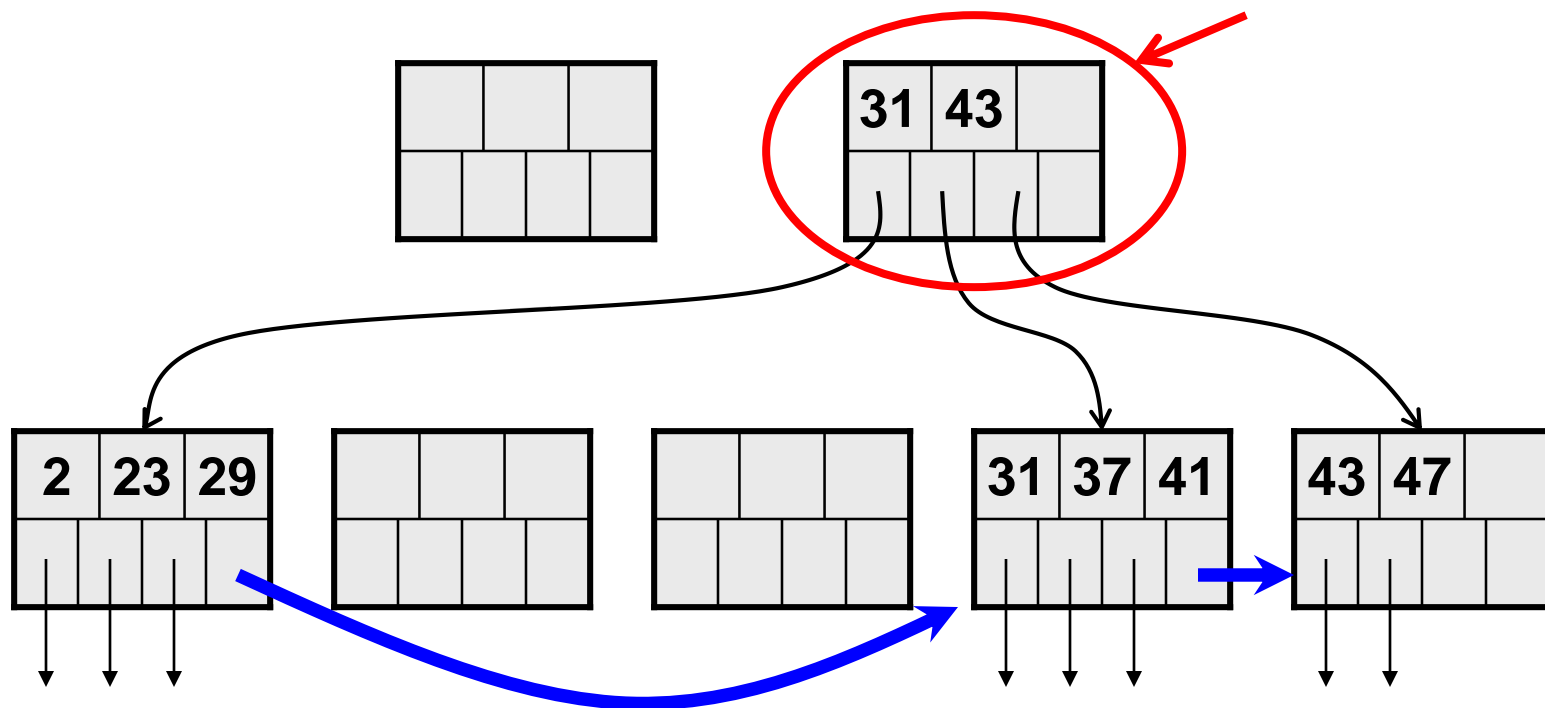


合并后要从它们的父结点中迁移一个关键字值过来，以实现前述的两点要求。

原来的根结点



新的根结点



抛弃已经为空的旧的根结点，合并后的结点将成为新的根结点，从而使整个B+树的高度下降了一层

5. B+树的效率考虑

- 随机查找的效率
 - ❖ 每次所需要的磁盘I/O次数等于B+树的高度
- 空间利用率
 - ❖ 超过50%
- B+树上的插入/删除操作

6. B+树与B树的比较

■ 组织方式不一样

- ❖ B+树：所有有效的索引关键字值都必须存储在叶结点中，其内部结点中的键值只用于索引项的查找定位。
- ❖ B树：有效的索引关键字值可以出现在B树的任意一个结点中。

■ 内部结点不同

- ❖ B+树：关键字值 + 子树指针
- ❖ B树：关键字值 + 记录指针 + 子树指针

- 因此B树结点的扇出（即一个结点可以拥有的最大子结点数且）较小，从而导致整个B树的高度大于B+树的高度。

B⁺树与B树的比较(续)

- 随机查找效率的区别
 - ❖ B⁺树：所有关键字的查找速度基本一致
 - ❖ B树：依赖于关键字所在结点的层次
- 在B树中没有提供对索引关键字的顺序扫描功能。
- B树的插入、删除操作较B⁺树复杂。

7.6.3 散列技术

➤ 基本原理

- 是一种 ‘利用散列函数 $h(K)$ 建立起数据文件中指定项值 K 与磁盘物理块间的映射关系’ 的索引技术
- 这样只要给出指定项的值 K_i 立即可通过 $h(K_i)$ 得到保存其记录的物理磁盘块地址

数据文件

...	
...	
...	
...	
...	
...	
K_i	
...	
...	
...	
...	
...	

项值 K_i

$h(K_i)$

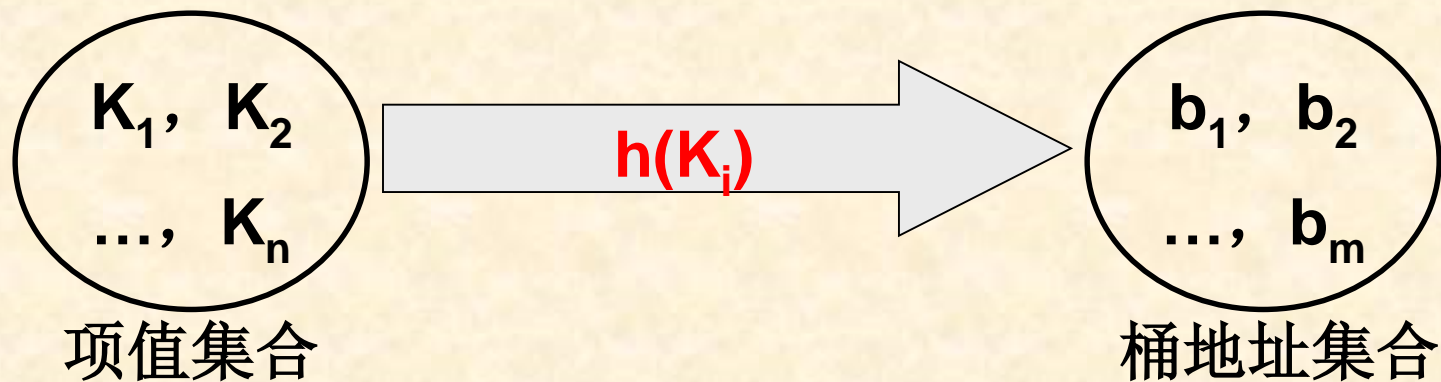
.....

7.6.3 散列技术

➤ 散列技术的实现方法

- 建立数据文件的指定项**K**以及该项值的集合
 $\{ K_1, K_2, \dots, K_n \}$
- 建立磁盘物理存储单位桶（**Bucket**）以及桶地址的集合： $\{ b_1, b_2, \dots, b_m \}$
 - ❖ 一个桶可以存放多条记录（或记录指针）
 - ❖ 一个桶可以是一个磁盘块，也可以是比磁盘块还大的物理空间
- 设计散列函数 $h(K_i)$
 - ❖ 以建立数据文件中指定项的值与桶（桶地址）之间的对应关系，即一个 K_i 通过 $h(K_i)$ 必能找到惟一的一个桶地址
 - ❖ 设计良好的散列函数将使得 n 个指定项值被平均分配到 m 个桶中去

7.6.3 散列技术



- 在散列技术中，桶的空间大小是固定的，即一个桶中可以存放的记录（指针）数是固定的
- 但是在实际应用中，记录在指定项值上的分布往往是不均衡的，从而使得在某些桶中存在空间浪费现象，而另外一些桶则存在空间溢出问题
- 当一个桶的空间溢出时，需要通过链接的方法申请“溢出桶”与其相连，以达到扩大桶空间的目的

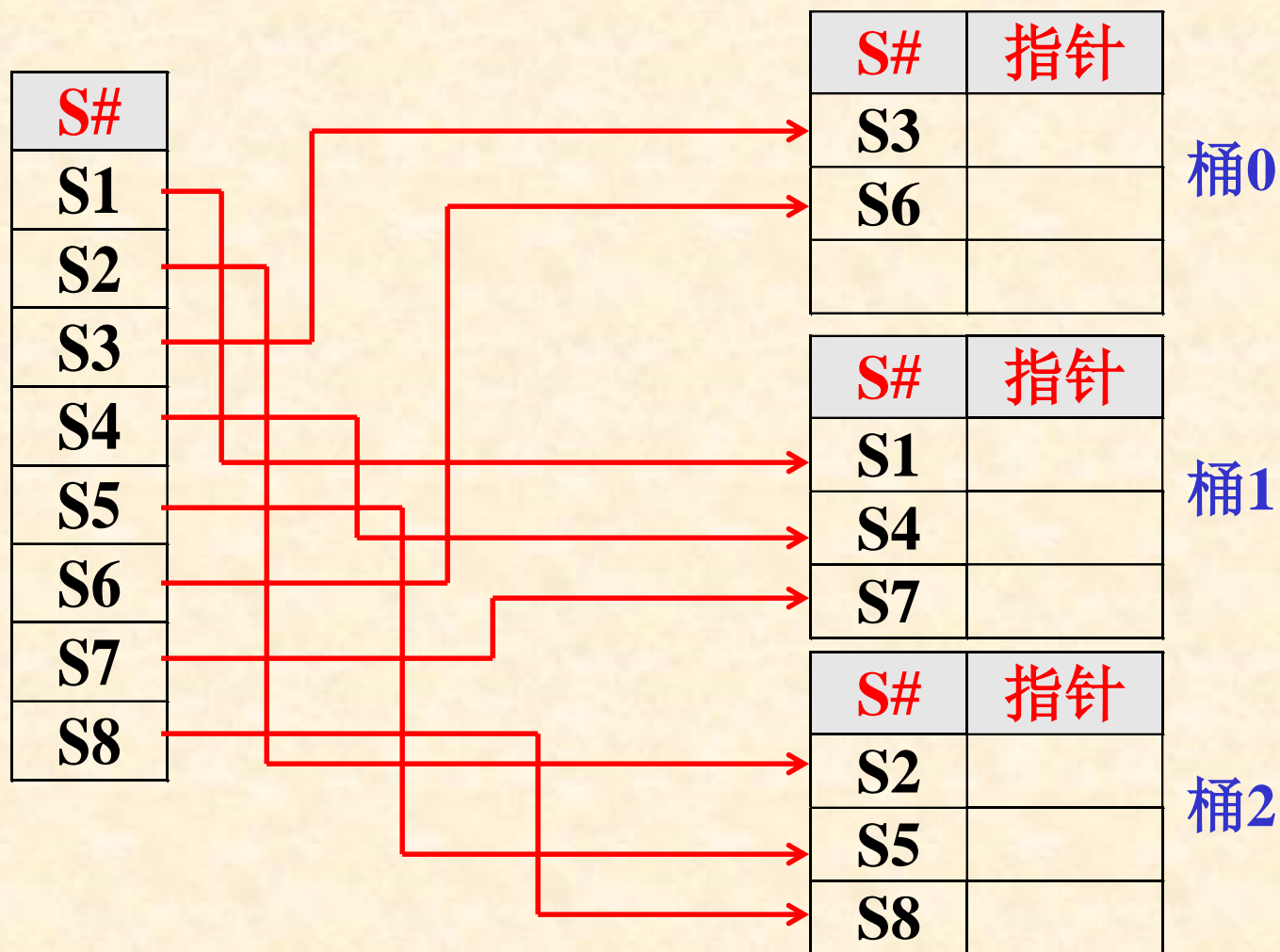
7.6.3 散列技术

➤ 散列索引

- 将散列技术与索引技术相结合形成的“散列索引”技术
- 散列索引的构造方法
 - ❖ 为数据文件中的每个索引项值建立一个索引记录
 - ❖ 将这些索引记录组织成散列结构
- 例如：为学生关系建立一个基于学号（**S[#]**）的散列索引，其中初始化桶地址的空间为{0, 1, 2}，每个桶可以存放三条索引记录，其散列函数是：

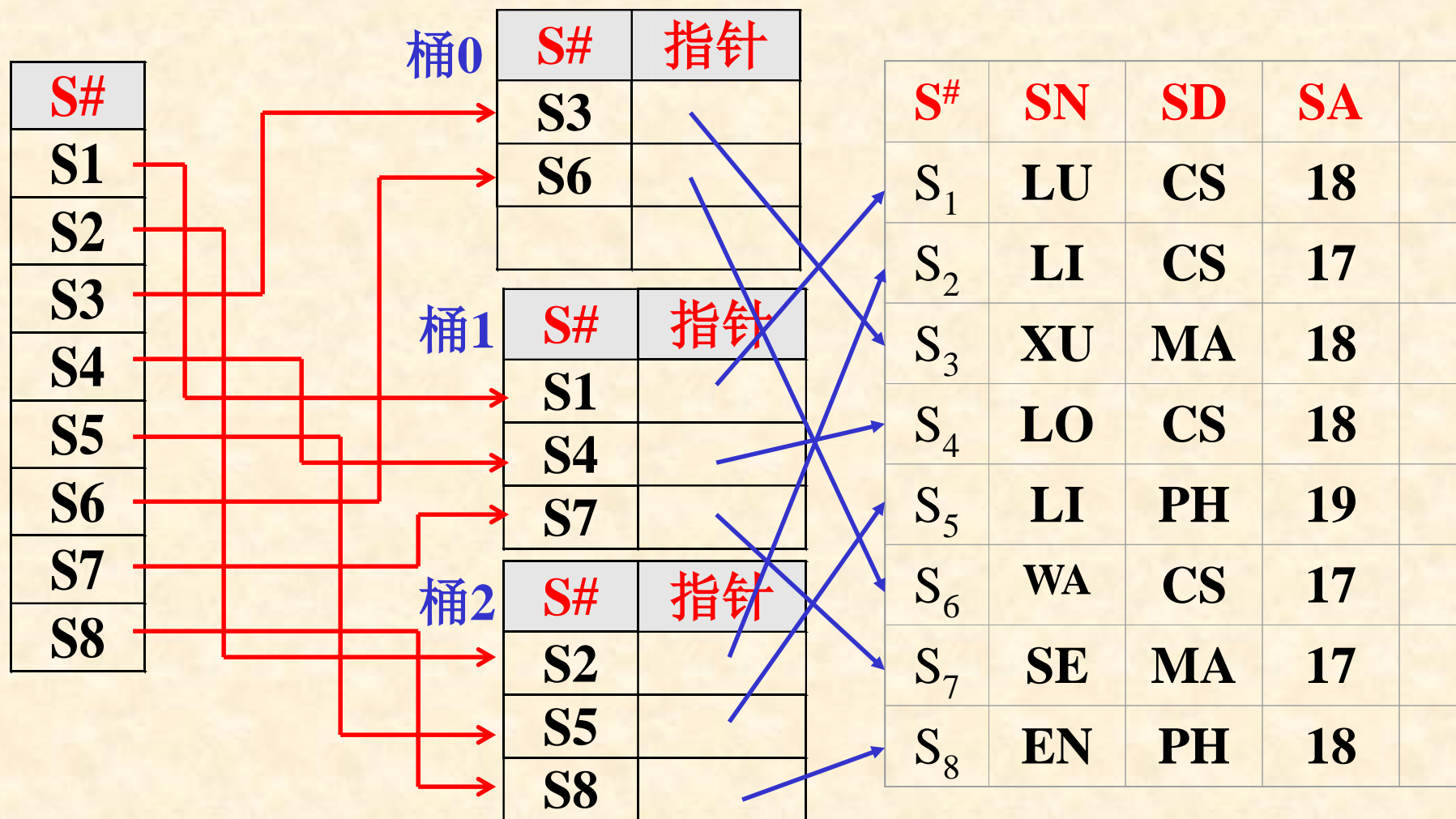
$$h(K_i) = i \bmod 3$$

7.6.3 散列技术



散列示意图

7.6.3 散列技术



散列索引示意图

7.6.3 可扩展的动态散列

➤ 静态散列一般通过增加溢出页来处理溢出问题

- 如不希望增加溢出页，也可修改散列函数，将桶的数目扩大（如扩大一倍），然后重组数据文件
- 但这种重组的代价可能很大：
 - ❖ 需要读入 n 个页，并写回 $2n$ 个页

➤ 克服这个问题的一个方法

- 引入一个仅存储桶指针的目录数组，用翻倍目录数组来取代翻倍数据桶数目
 - ❖ 由于每个目录项只含有一个桶指针，目录所需存储页一般很少，这样翻倍的代价就很小
- 每次只分裂有溢出的桶

7.6.3 可扩展的动态散列

- 引入一散列函数 h ，将索引键值映射为二进制数，并解释最后的 d 个比特位
- d 值是目录项的编码位数
 - 目录项总数为 2^d 个； d 值加1目录项数将增加一倍
 - d 值也称全局位深度（the global bit-depth）
- 每个桶有一个局部位深度（the local bit-depth）
 - 在局部位深度为 ℓ 的桶中，所存储数据项对应散列值的后 ℓ 位取值都相同。
 - 一般情况下，会有 $2^{d-\ell}$ 个目录项指向这个桶；当 $\ell=d$ 时，只有一个目录项指向这个桶。最初，所有桶都有 $\ell=d$

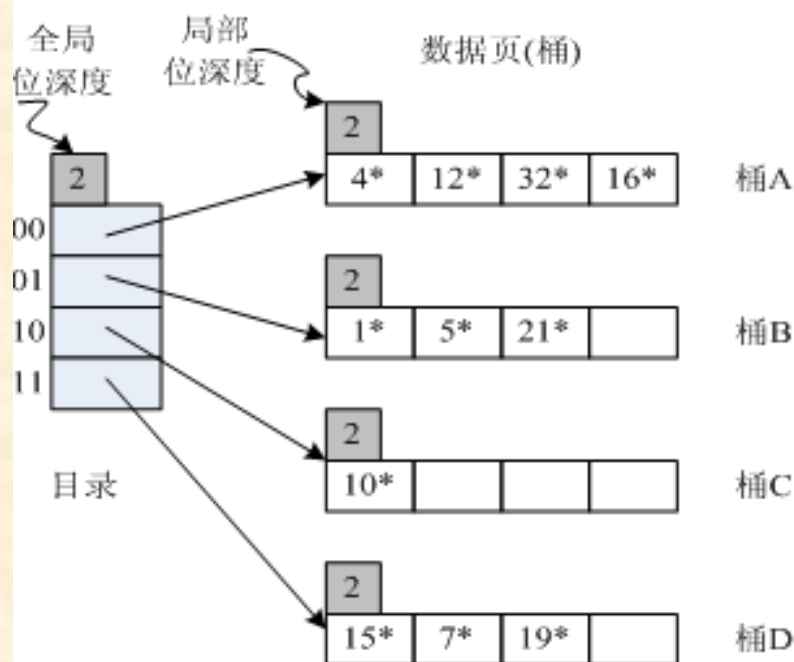
7.6.3 可扩展的动态散列

- 当向一个已满的、局部位深度为 ℓ 的桶插入数据项时，就要分裂这个桶
 - 该桶及分裂产生的映像桶： $\ell \rightarrow \ell+1$
 - 如果 $\ell+1 > d$ ，则还需要增加目录的编码位数，即要对目录进行翻倍处理
- 翻倍目录时，只需将原有的每个目录项分别复制产生1个对应的新目录项
 - 一个目录项和由它复制产生的新目录项，互称为“对应元素”
 - “对应元素”开始时指向同一个桶，只是当桶分裂后，才分别指向原桶和原桶分裂映像
- 可扩展散列存在的主要问题
 - 当散列值分布不均匀或偏斜很大时，会导致目录项数特别大和数据桶的空间利用率很低
 - 每次目录调整都是翻倍调整，目录大小扩展过快，调整不平滑

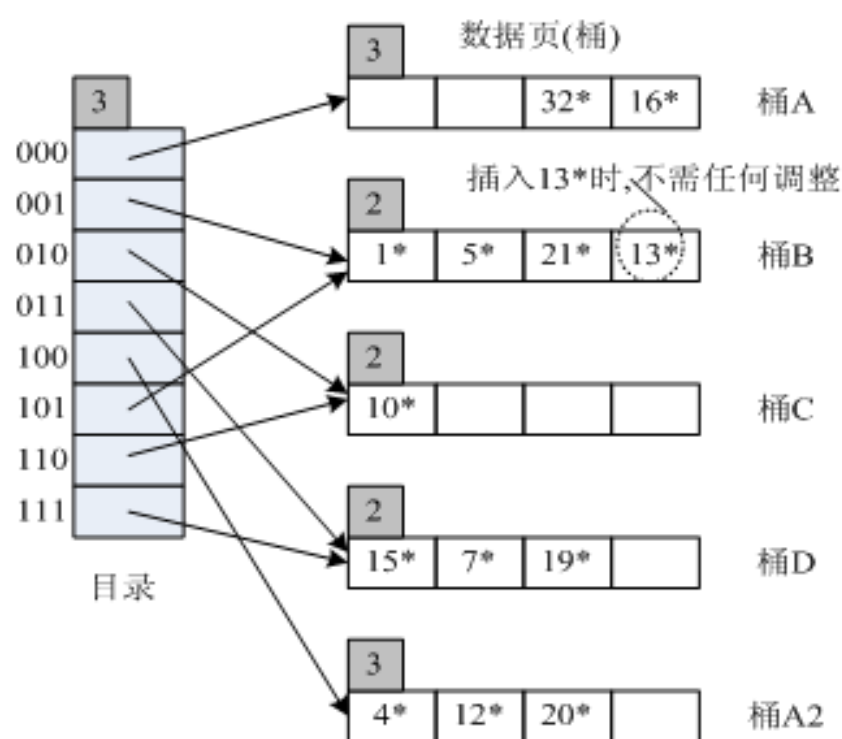
7.6.3 可扩展的动态散列

例5.5

- 每个目录项存1个指向桶的指针,每个桶可存放4个数据项
- 初始时, 总共有4个目录项, $d=2$, $l=2$ 。



(a) 一个简单的扩展散列示例



(b) 在图(a)基础上, 插入数据项13*, 20*后的情况

7.6.3 位图索引

- ◆ 是一种主要针对多键查询设计的特殊类型索引，尽管每个位图索引都是建立在单键码之上
- ◆ 为了使用位图索引，关系或数据文件中的记录必须进行顺序编号 $1, 2, \dots, n$ ，使得给定一个编号 i ，能方便检索到第 i 个数据记录

7.6.3 位图索引的结构

- 考虑一个共有 n 个记录的关系 R 和它的一个属性 A ，假设 R 的所有记录在 A 上只有 m 个的不同取值，

V_1, V_2, \dots, V_m

- R 在 A 上的位图索引

- 是一组位图向量

- ❖ $R.A$ 的每个取值 $v_j (j=1..m)$ ，分别对应一个位图向量 bit_v_j

- ❖ 共有 m 个这样的位图向量

- bit_v_j 是一宽度为 n 的二进制数，该数的第 i 位值

- ❖ $bit_v_j[i] = 1$ ，如果第 i 条记录的 $R.A$ 取值 $= v_j$ ；

- ❖ $bit_v_j[i] = 0$ ，如果第 i 条记录的 $R.A$ 取值 $\neq v_j$ ；

7.6.3 位图索引示例

Record number	name	gender	address	Income-level	gender 的位图	Income-level 的位图
0	John	m	Perryridge	L1	m 10010	L1 10000
1	Diana	f	Brooklyn	L2	f 01101	L2 01000
2	Mary	f	Jonestown	L3		L3 00100
3	Peter	m	Brooklyn	L4		L4 00010
4	Kathy	f	Perryridge	L5		L5 00001

7.6.3 位图索引的应用

➤ 多属性检索应用

1. 条件匹配查询

$$\sigma_{\text{gender}=\text{f} \wedge \text{income-level}=\text{L2}}(r)$$

2. 范围查询

$$\sigma_{\text{gender}=\text{f} \wedge \text{income-level} > \text{L2}}(r)$$

7.1 概论

7.2 数据库的物理存储介质

7.3 磁盘存储器及其结构

7.4 文件组织

7.5 文件记录组织

7.6 索引技术与散列技术

7.7 数据库与文件

7.7 数据库与文件

7.7.1 数据库中数据分类

7.7.2 数据库存储空间组织

7.7.1 数据库中数据分类

1. 数据主体及辅助数据

- 数据主体：用户数据自身
- 辅助数据：数据主体的控制信息
 - 如数据的长度、数据项的分隔符等，通常它们与数据主体是结合在一起存储的

2. 数据字典

- 有关数据的描述信息
- 数据字典的数据量较小，但使用频率高，需要较高的访问速度

7.7.1 数据库中数据分类

3. 数据间联系信息

- 数据主体内部存在着数据间的联系，需要用一定的“数据”来表示
- 这方面信息也是一类数据，在不同的数据库系统中有不同的实现方法
 - ❖ 在层次及网状数据库中，采用‘指引元’或‘邻接关系’来表示数据间的联系信息
 - ❖ 在关系数据库中，用‘外关键字’来表示数据间的联系，这类数据也被包含在数据主体中

7.7.1 数据库中数据分类

4. 数据存取路径信息

- 在层次及网状数据库中，不需要附加的数据来表示访问路径，访问方式较单一
- 在关系数据库中，一般通过索引来定义数据存取路径（如：主关键字上的索引），用户也可以通过创建新的索引文件来提供新的访问路径
 - ❖ 所有索引文件都需要占用新的存储空间

7.7.1 数据库中数据分类

5. 其它信息：日志信息、用户信息、审计信息

- 特殊的数据需要有特殊的处理办法，包括：

❖ 日志信息

- 用于记录对数据库作“更新”操作的有关信息，以对付数据库遭受破坏时恢复之用

❖ 用户信息

- 有关数据库用户登录信息以及相应的用户安全性信息

❖ 审计信息

- 用于跟踪用户是否正确使用数据库的审计信息

7.7.2 数据库存储空间组织

➤ 系统区

- 数据字典
- 日志
- 用户信息
- 审计信息

➤ 数据区

- 数据主体及其辅助数据
- 数据之间的联系信息
- 数据的访问路径

7.7.2 数据库存储空间组织

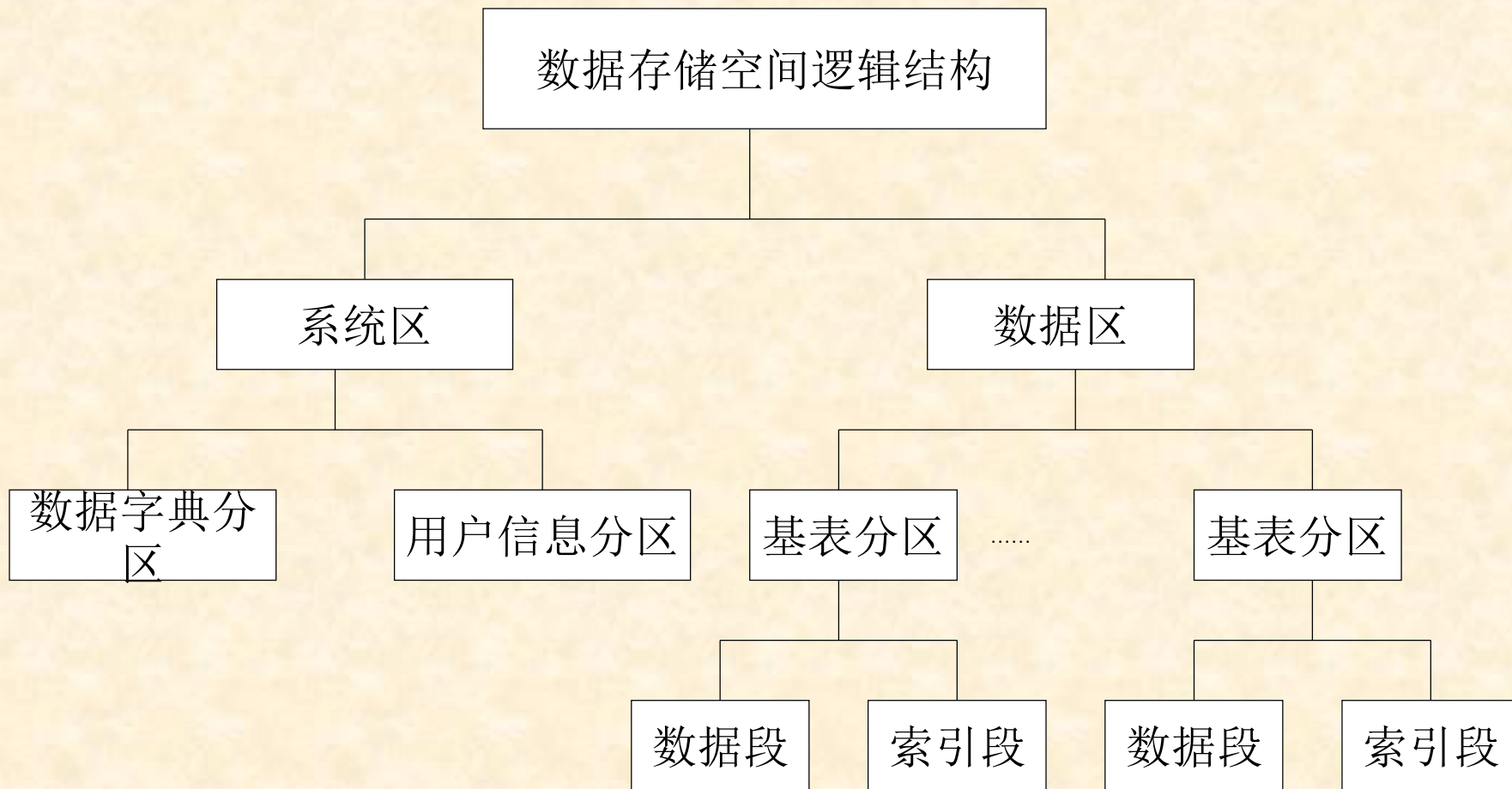


图7.22 数据存储空间逻辑结构图