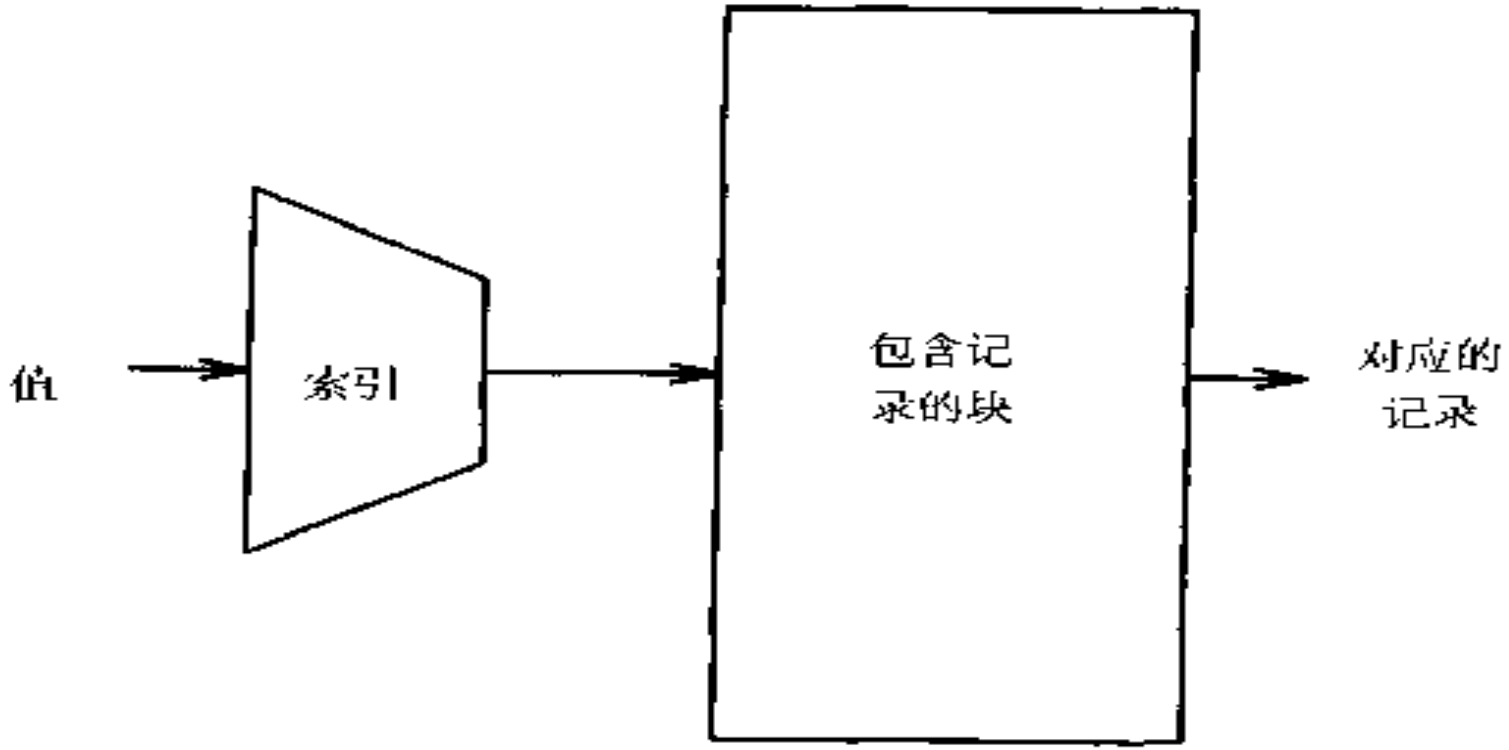
第三部分：索引实现技术

韩丽萍
计算机学院
联系邮件：liping@njupt.edu.cn

问题背景

select * from moviestar where name='Tom'



内容提要

1 顺序文件上的简单索引

2 非排序文件上的辅助索引

3 B树，可在任何文件上建立索引

4 散列（哈希）表

索引的涵义

创建时：

根据一个数据文件，生成一个索引文件，索引文件由键-指针对组成。

查询时：

首先在索引文件找到对应的键-指针，然后根据指针，在O(1)时间找到数据文件中对应的块，从块中读出记录。

顺序文件上的索引

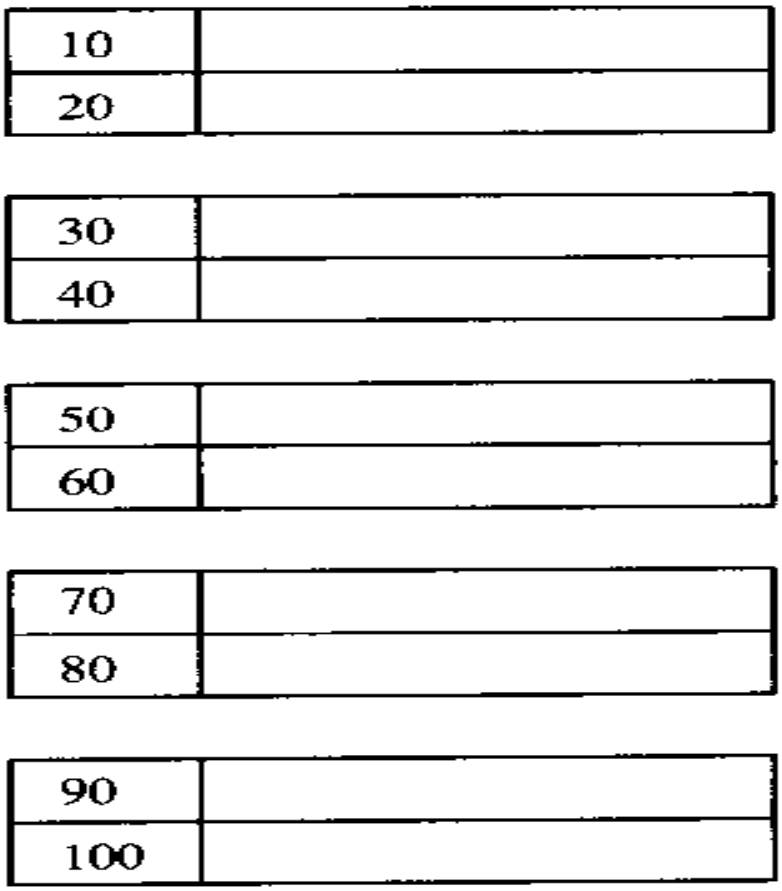
（1）根据数据文件（顺序文件）构造索引文件，索引文件存储了许多键-指针。

（2）查询时，根据索引文件中键对应的指针，到数据文件中读取对应的块。

顺序文件上的索引

顺序文件：按某个属性排序的数据文件

复习（堆文件）：数据文件中的记录根据先来后到的插入顺序存储，没有按照特定的属性值进行排序。



顺序文件上的索引

（3）稠密索引：数据文件中每个记录在索引文件中都对应一个索引项。

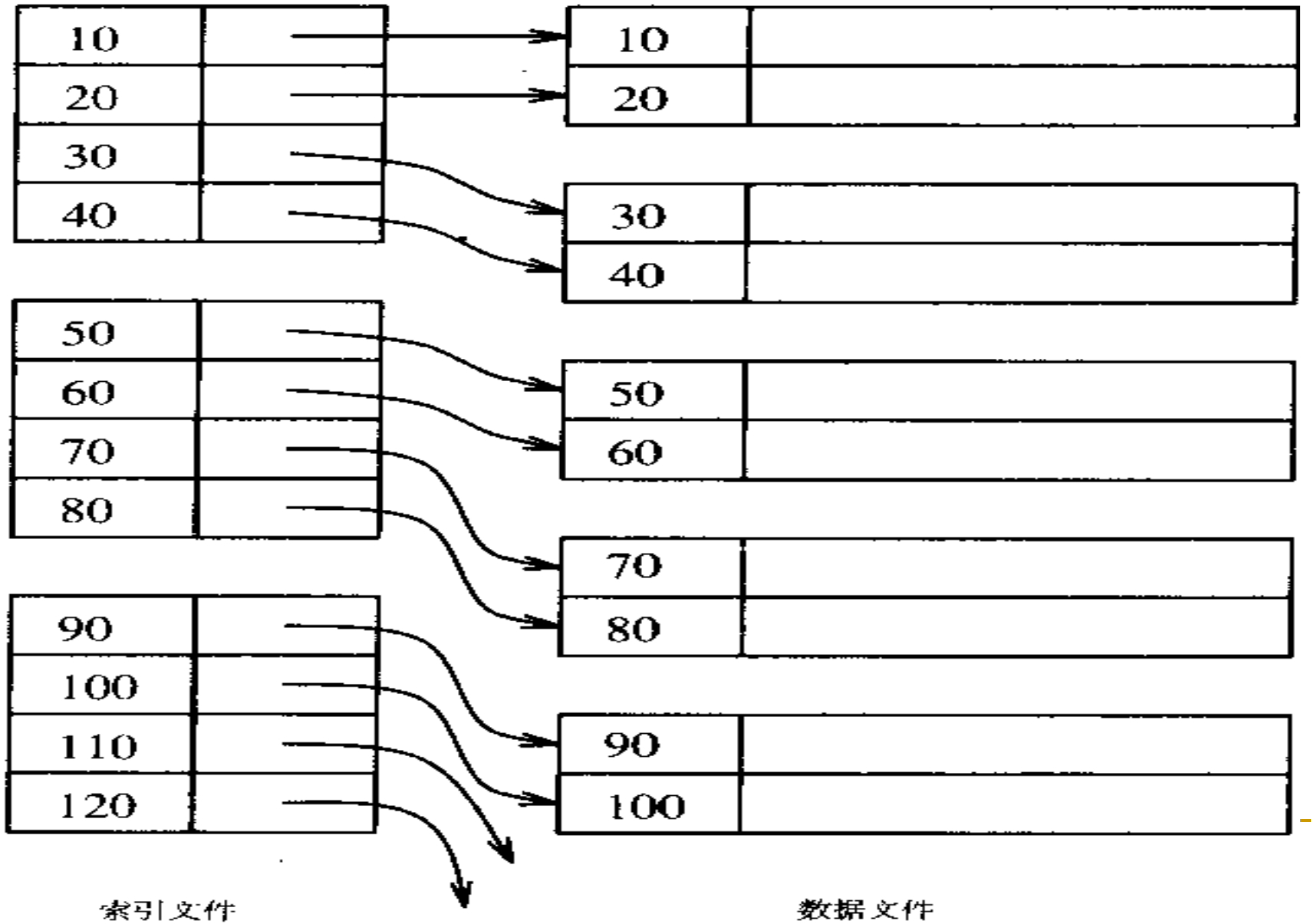
（4）稀疏索引：数据文件中若干条记录在索引文件中有一个索引项，通常一个数据块对应一个索引项。

顺序文件上的索引——稠密索引

稠密索引：稠密索引文件亦由一系列的存储块构成，每个块存放记录的键和指向记录本身的指针(比如：块号+偏移量)。

如果该索引是主索引（一个顺序文件只有一个主索引），则索引块中键的顺序与数据文件中键的排序一致，参见pp.91页图4-3

顺序文件上的索引——稠密索引



顺序文件上的索引——稠密索引

基于索引的查询为何更有效？

1 索引文件中索引块的数量一般远小于数据文件中数据块的数量

2 索引中键被排序，可以用二分法在索引文件查找K。若有n个索引块，只要查找$\log_2 n$个块。

3 索引文件如果足够小，可以在主存缓冲区中查找K，而不需要外存I/O。

举例子

一个关系有1,000,000个元组，存储块大小为4096个字节，假设一个存储块可以存放10个元组，数据文件大约是400MB。如果关系的键占30个字节，指针占8个字节，加上块头所需空间，可以在一个大小为4096的块中存放100个键-指针对。

（1）稠密索引文件（只有一层）大约40MB（10,000个块）。

（2）进一步，假定索引文件放在磁盘上用二分查找，$\log_2(10000)$大约是13，访问13～14个存储块就可以找到指定的键值。如果把一些索引块放到内存，I/O次数远低于14；如果整个索引放在主存中，则只要1个I/O。

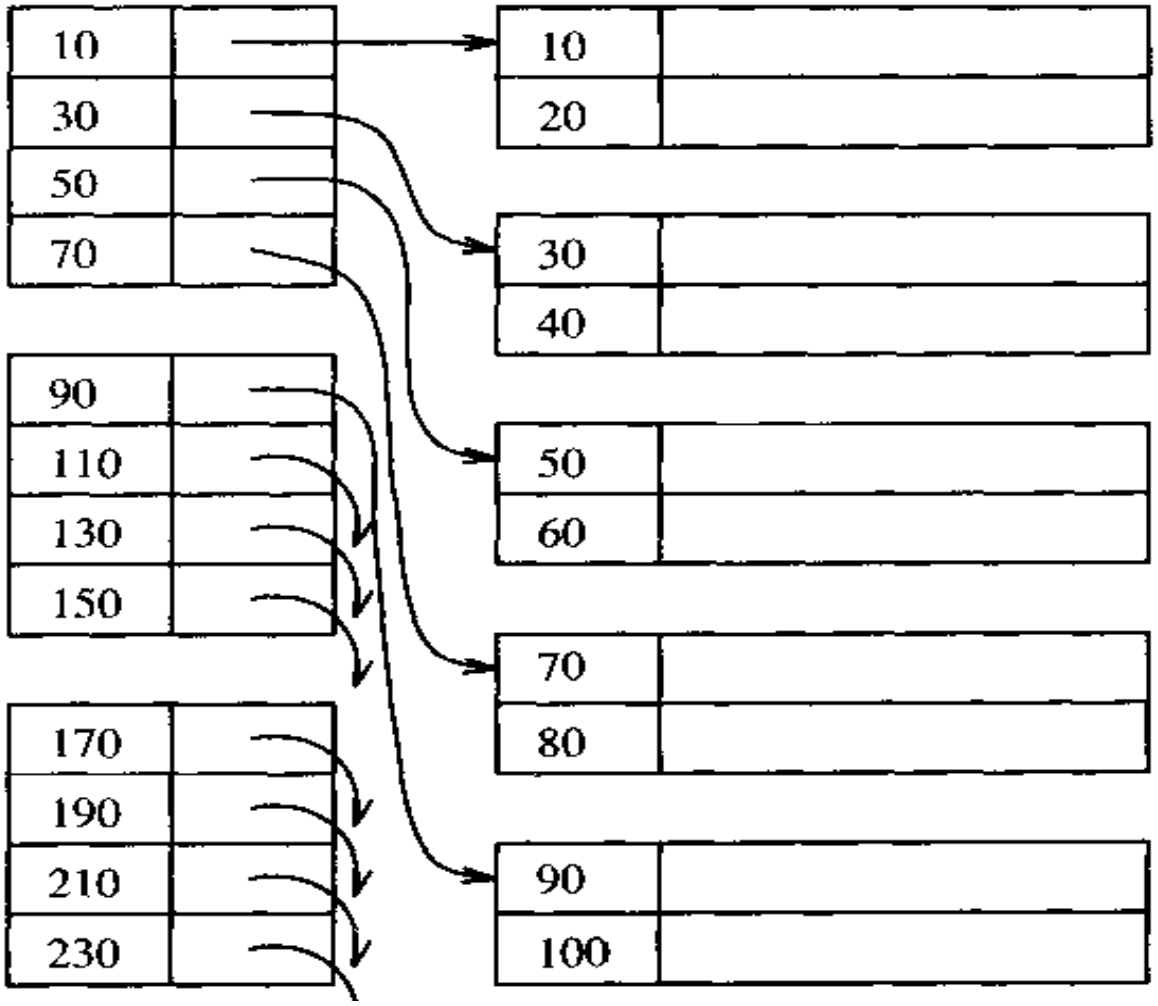
（3）自己计算不用索引的I/O次数（顺序查找100,000个块，二分查找17个块）。

顺序文件上的索引——稀疏索引

（1）稀疏索引为每个数据存储块设一个键-指针，键值是该数据块中的第一个记录的对应值。

参见pp.92图4-4

（2）查询时，先用二分查找在索引上定位小于或等于K的最大键值，然后到对应的数据文件存储块搜索是否含K.



顺序文件上的索引——稀疏索引

例子4.4
承接前面的例子:

（1）由于数据文件有100,000个存储块(也是键-指针对的数目)

（2）一个索引块可以存放100个键-指针对

（3）因此使用稀疏索引只要1,000个索引块，大约4M的大小，这个大小可以完全放入主存！

顺序文件上的索引——稀疏索引

稀疏索引和稠密索引对比：

（1）稀疏索引的大小远小于稠密索引，并且稀疏索引的指针要比稠密索引短（稀疏索引优点）。

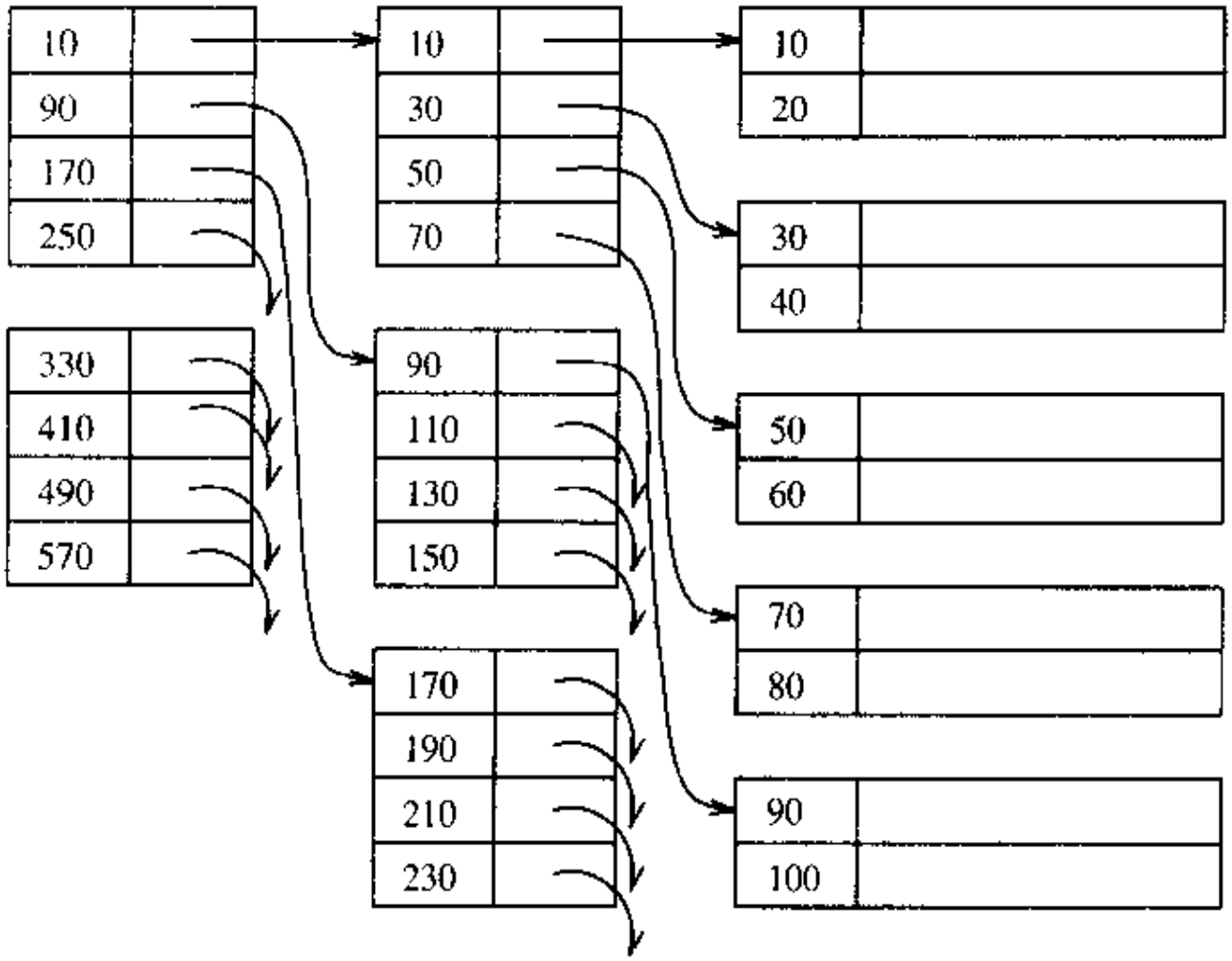
（2）稠密索引不用检索数据文件就可以回答诸如“是否存在键值为K的记录”，但稀疏索引要检索可能包含键值的所有数据块（稠密索引优点）

顺序文件上的索引——多级索引

多级索引的

（1）一级索引可以是稠密的也可以是稀疏的，

（2）二级以上的索引必须是稀疏的，否则浪费空间而无收益。



pp.94图4-5

顺序文件上的索引——多级索引

参见例子4.5

（1）假定在一级稀疏索引上建二级索引，由于一级稀疏索引占据1,000个存储块，且一个存储块可以存放100个键-指针对，则只要10个存储块来存放二级索引。

（2）二级索引可以直接放在主存中，根据二级索引，通过一个I/O取到一级索引对应的索引块，再根据该索引块取出对应的数据块。

（3）共用2个I/O。

顺序文件上的索引——重复查找键的索引

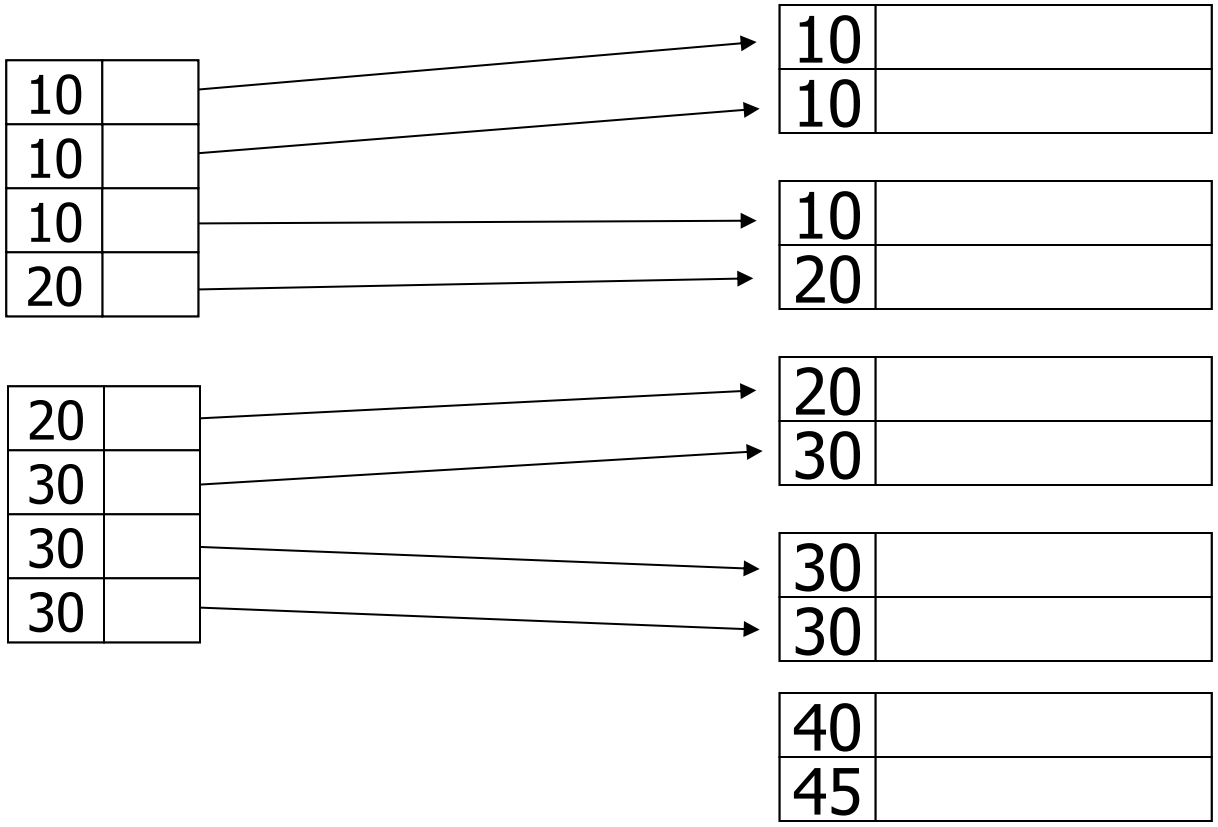
背景：数据文件的键有重复值，这种情况很常见（pp.94）

稠密索引-方法1：

每一个具有键值K的记录都设一个稠密索引项，即稠密索引文件中允许出现重复的查找键。

见下图

顺序文件上的索引——重复查找键的索引



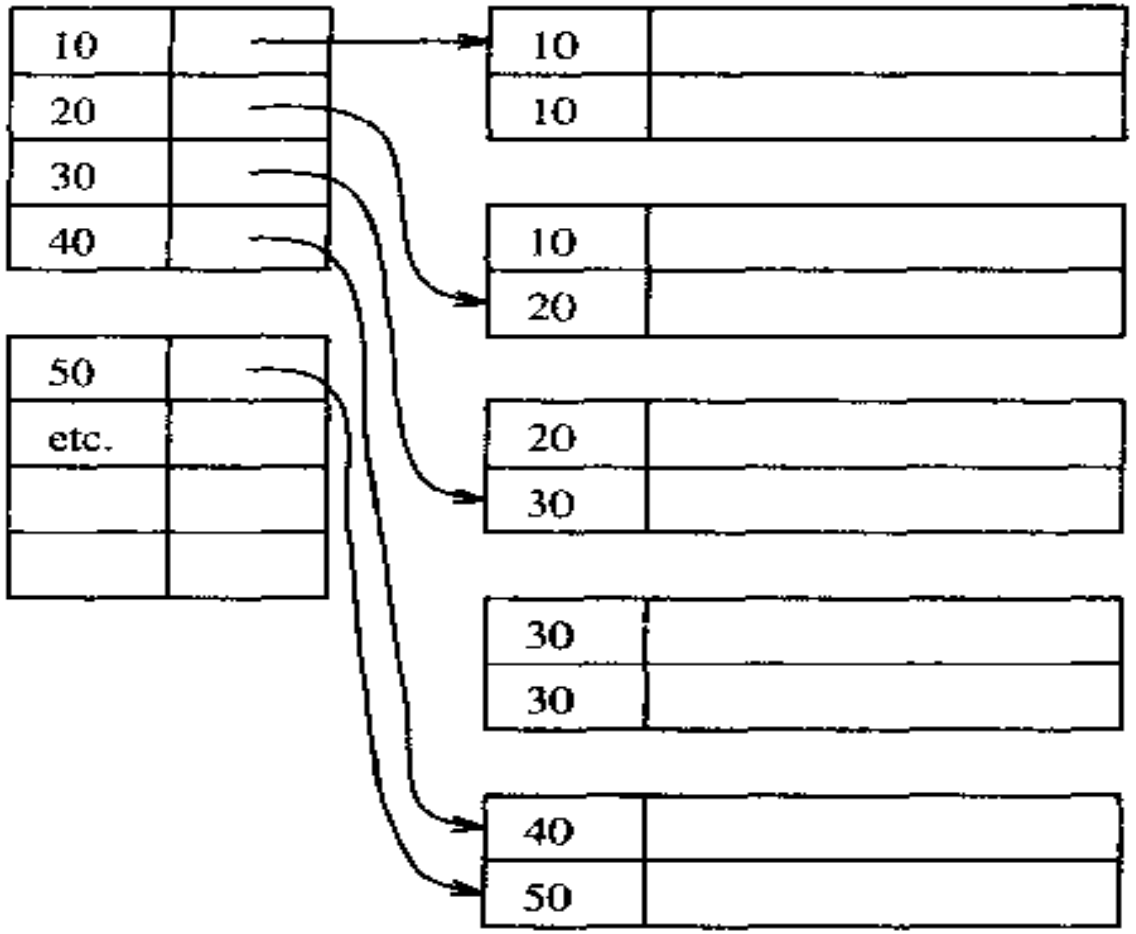
顺序文件上的索引——重复查找键的索引

稠密索引-方法2：

（1）索引的每个键值K只设一个索引项，

（2）该索引项的指针指向键值为K的第一个记录。

（3）查询K时，要在数据文件中顺序检索。如图pp.95图4-6



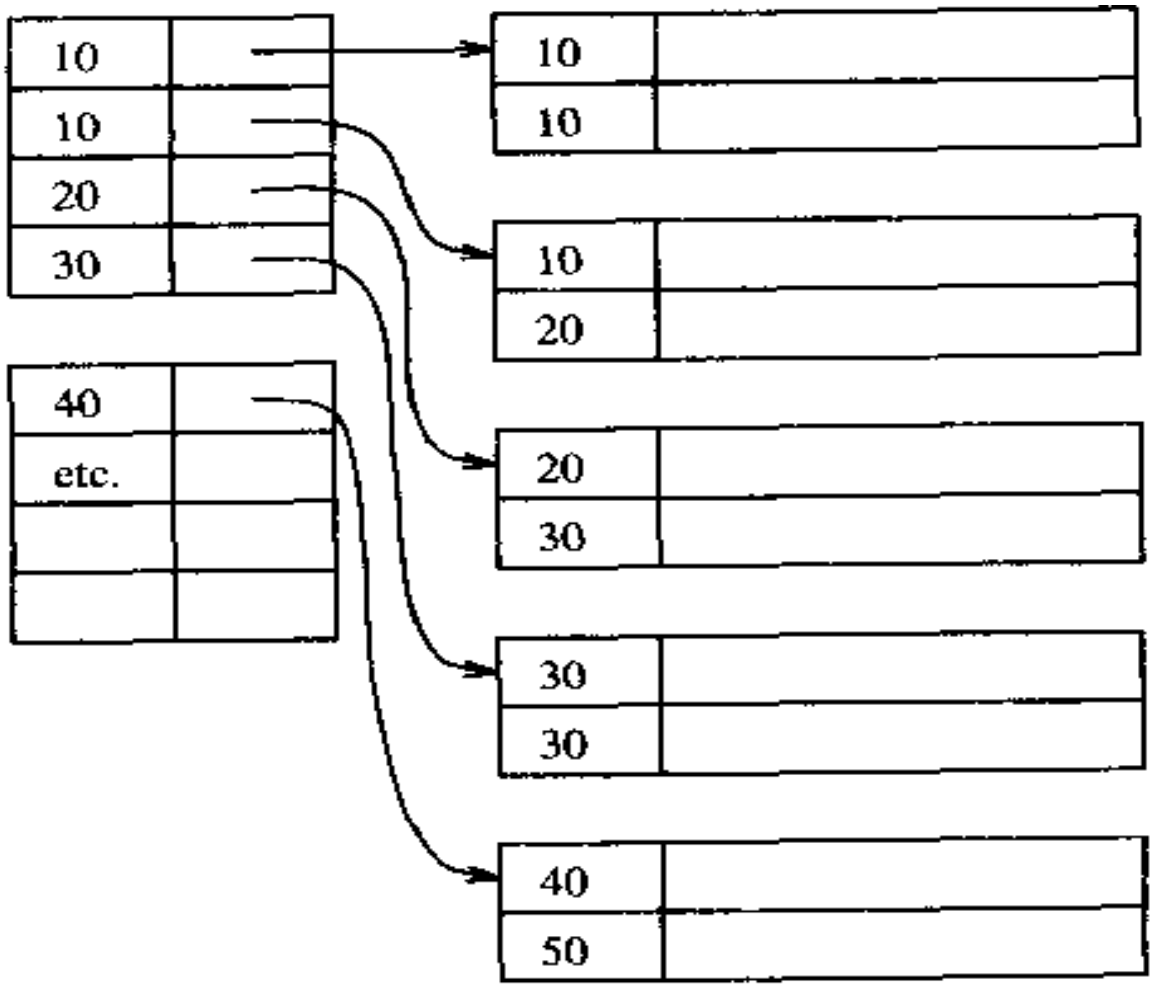
顺序文件上的索引——重复查找键的索引

稀疏索引-方法3：

（1）采用稀疏索引，

（2）它的键-指针指向数据文件中每个块的第一个查找键，

（3）参见图4-7。



顺序文件上的索引——重复查找键的索引

查找键是K的记录：

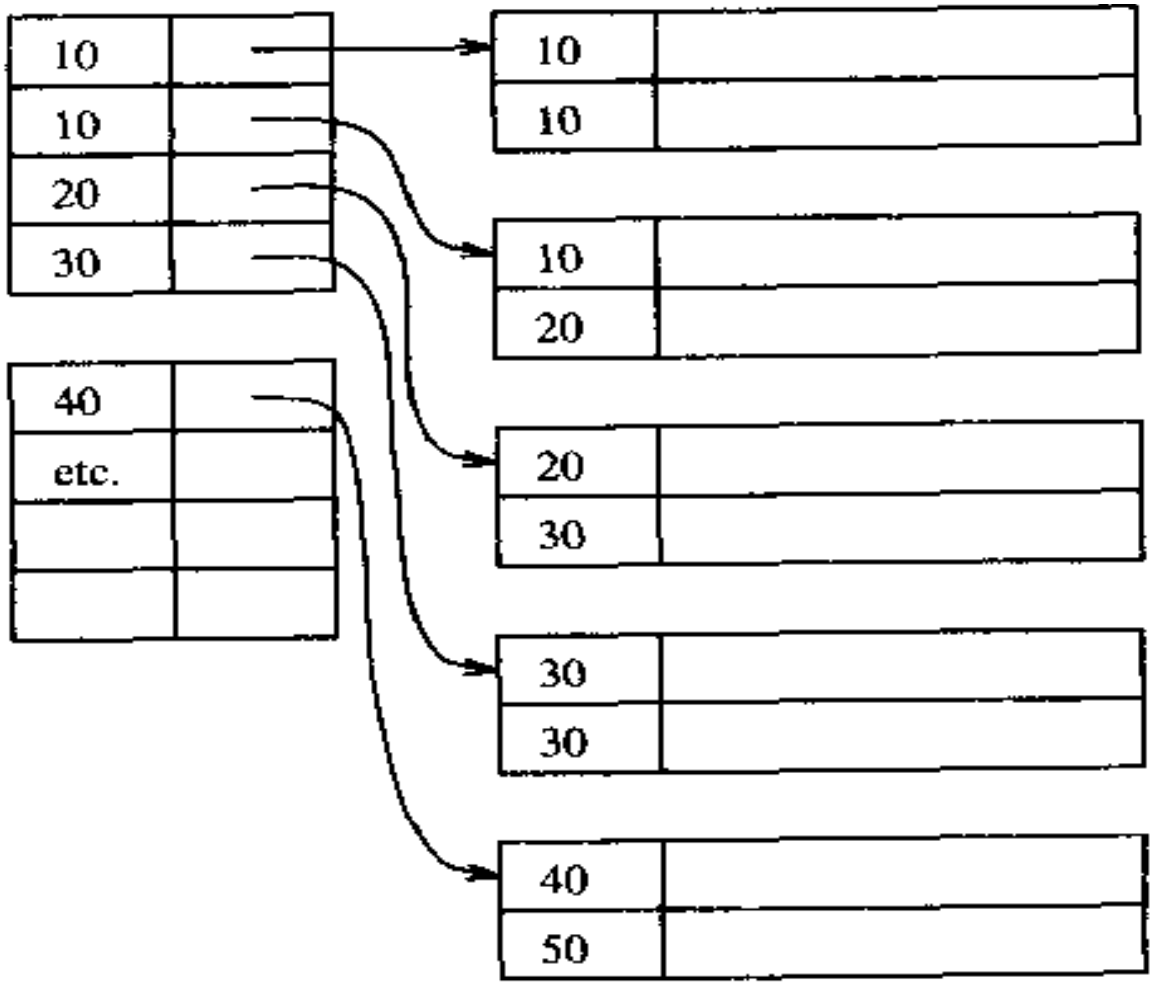
（1）先找到索引中键值等于或者严格小于K的最后一个索引键，记为E1

（2）从E1往索引起始方向找，直到碰到第一个索引项或者严格小于K的索引项，记为E2

（3）E1和E2指向的数据块中可以找到所有的键值为K的记录（注意稀疏索引指向的是数据块，不是记录）

顺序文件上的索引——重复查找键的索引

如何查找键值20或10？



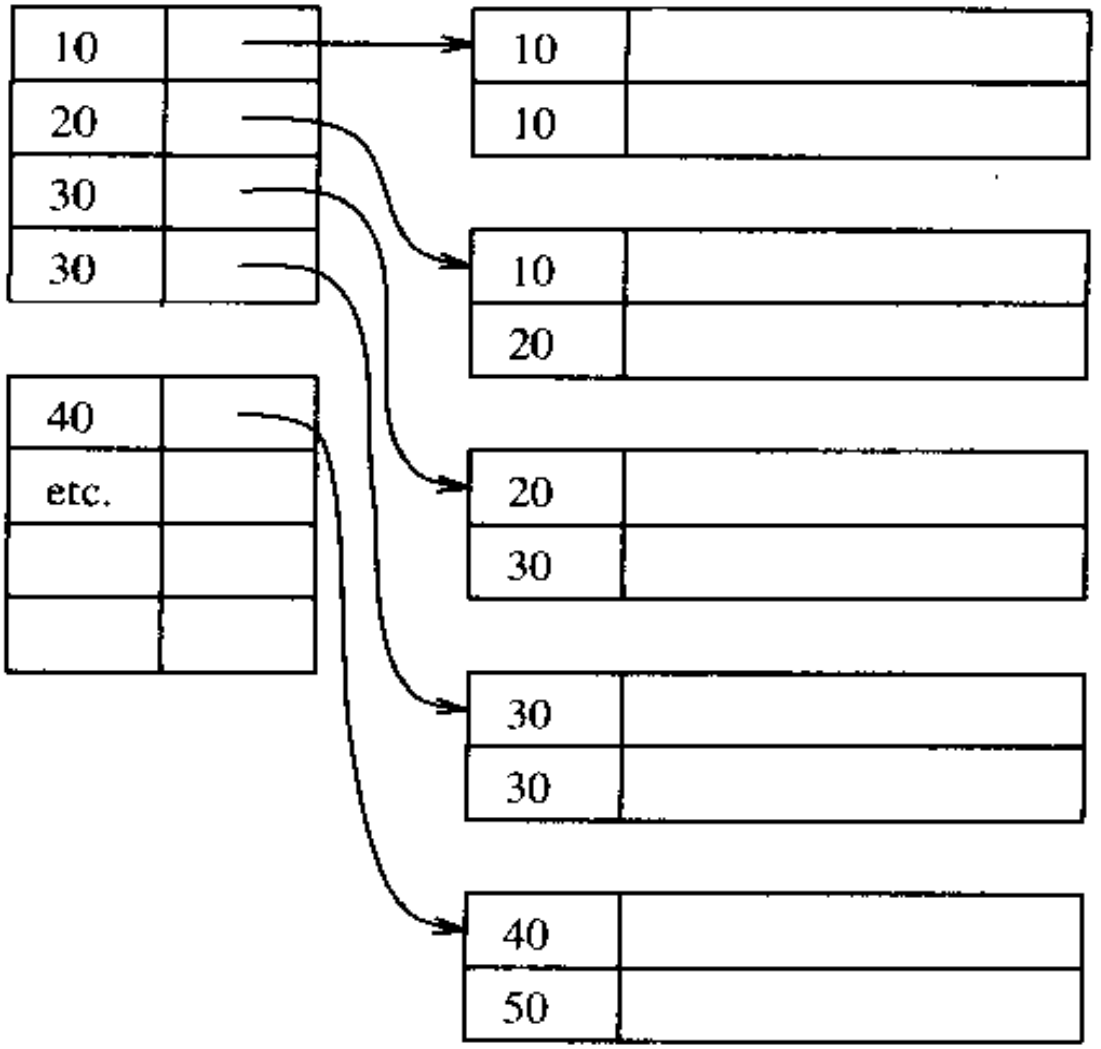
顺序文件上的索引——重复查找键的索引

稀疏索引-方法4：

采用稀疏索引，它的键-指针对仍指向块，但设定方法不同：

（a）在前面的块中没有出现的最小查找键设一个索引项

（b）整个块同一个键值也设一个索引项，参见图4-8。



顺序文件上的索引——重复查找键的索引

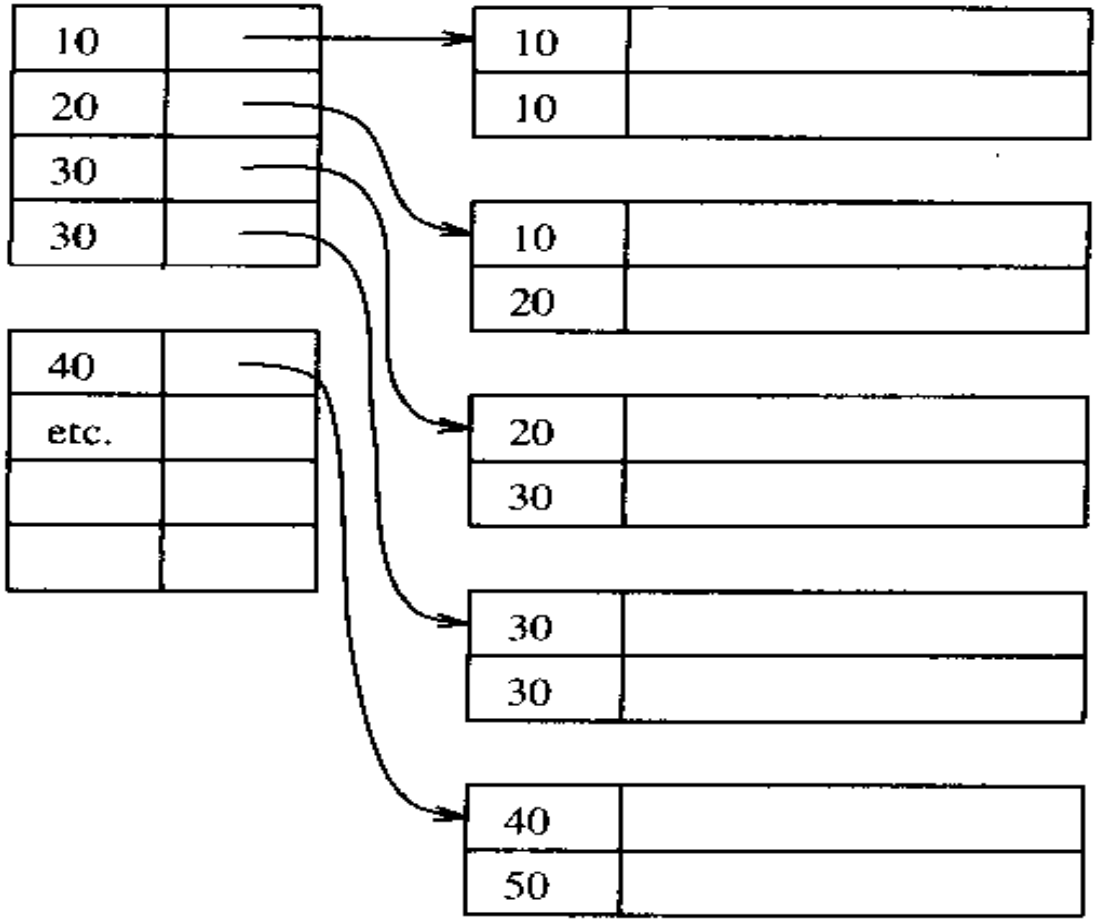
查找键值为K的记录：

（1）找到第一个键值为K的索引项，然后往后找

（2）或者是键值严格小于K的索引项，但下一个键值大于K，则在本块查找完毕即可以，如果没找到，则不必找下一块。

顺序文件上的索引——重复查找键的索引

如何查找键值30或25？



数据修改期间的索引维护

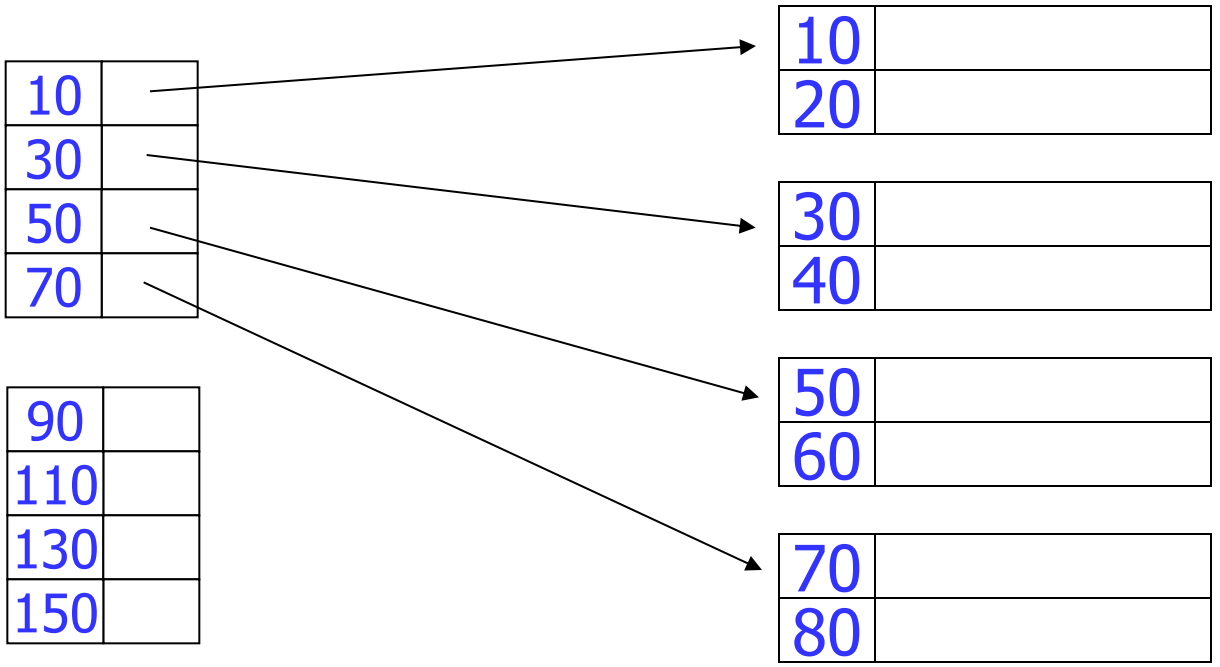
回忆数据文件的重组方式：

（1）溢出方式：需要额外空间，创建溢出块；如果溢出块不需要，则删除溢出块。注意溢出块是基本存储块的扩充，插入和删除不影响稠密索引和稀疏索引的对应索引项（当空气！）。

（2）邻近块方式（pp.84）：数据的增加和删除直接导致元组在临近块的移动，从而导致索引变化

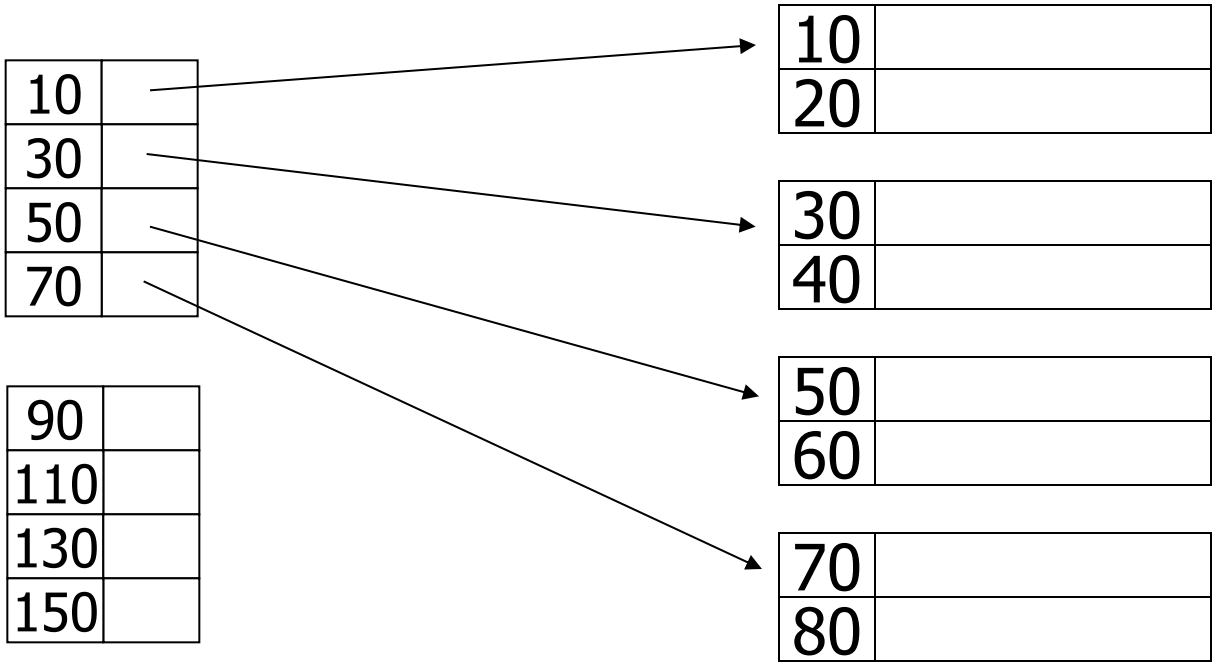
备注：先分析删除，再分析插入

删除影响稀疏索引



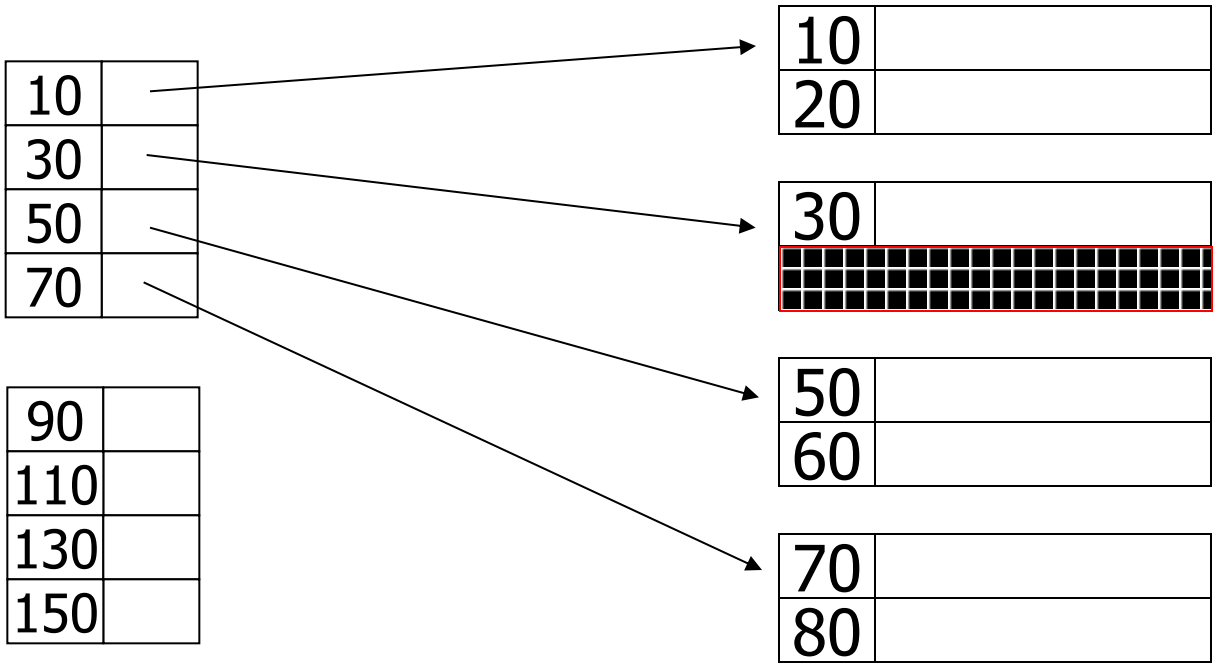
删除影响稀疏索引

– 删除记录40



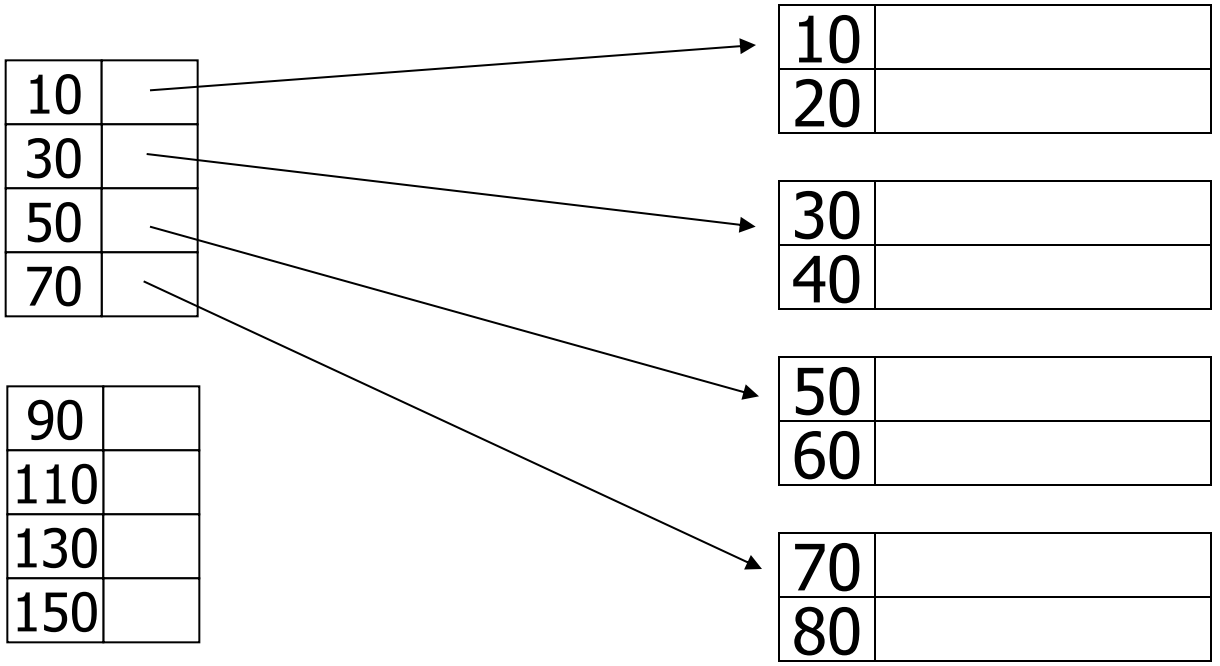
删除影响稀疏索引

– 删除记录40



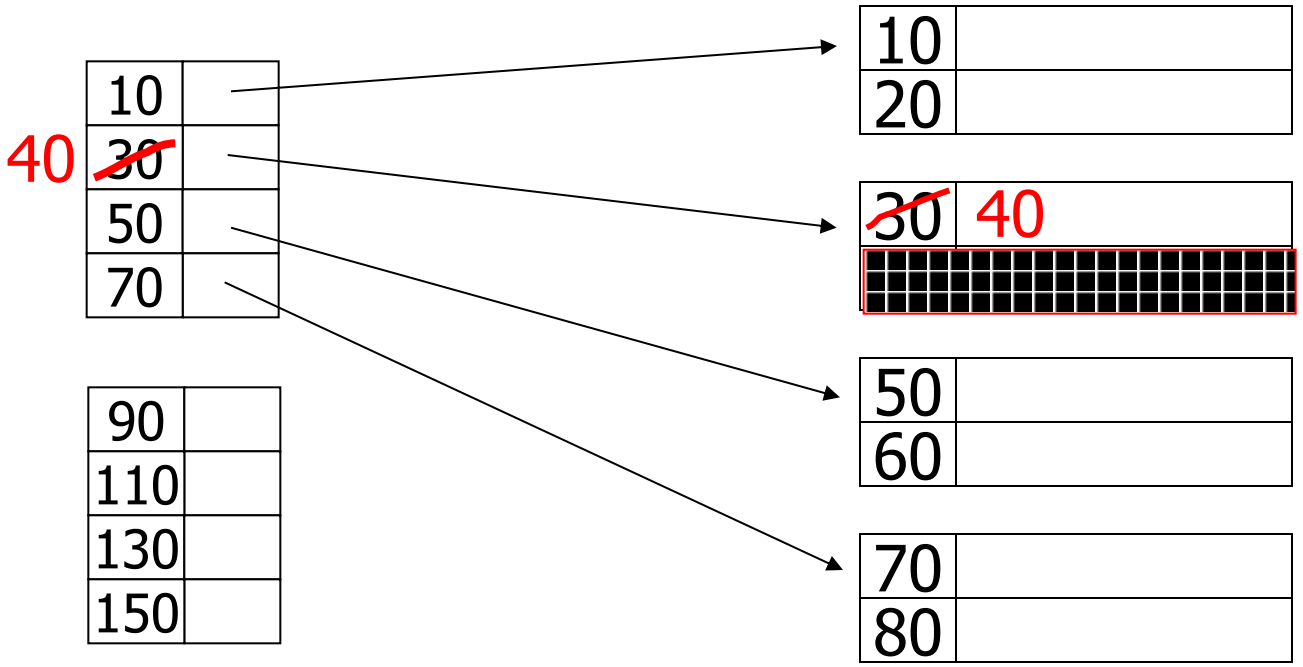
删除影响稀疏索引

– 删除记录 30



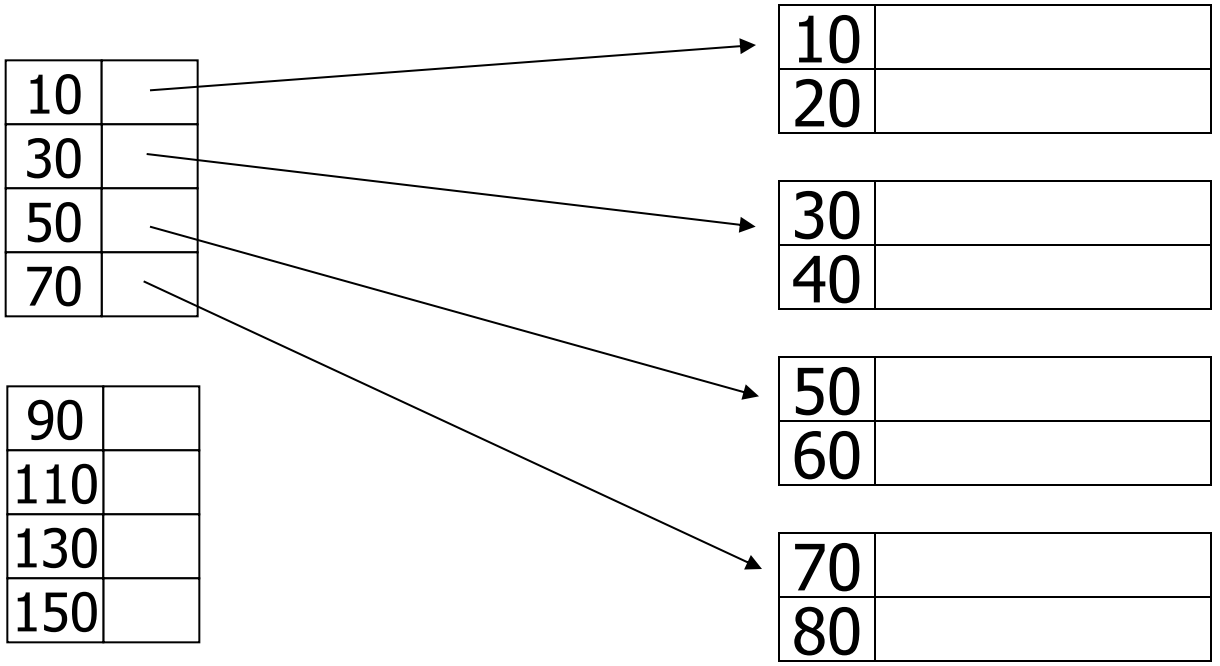
删除影响稀疏索引

– delete record 30



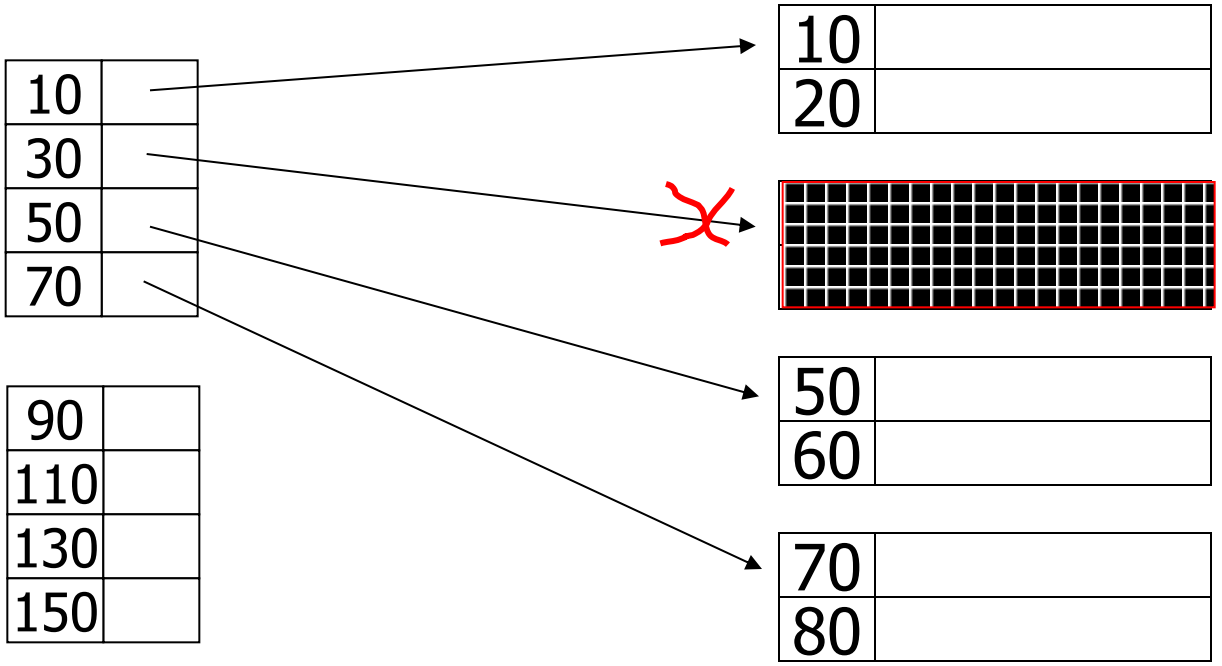
删除影响稀疏索引

– 删除记录30 和 40



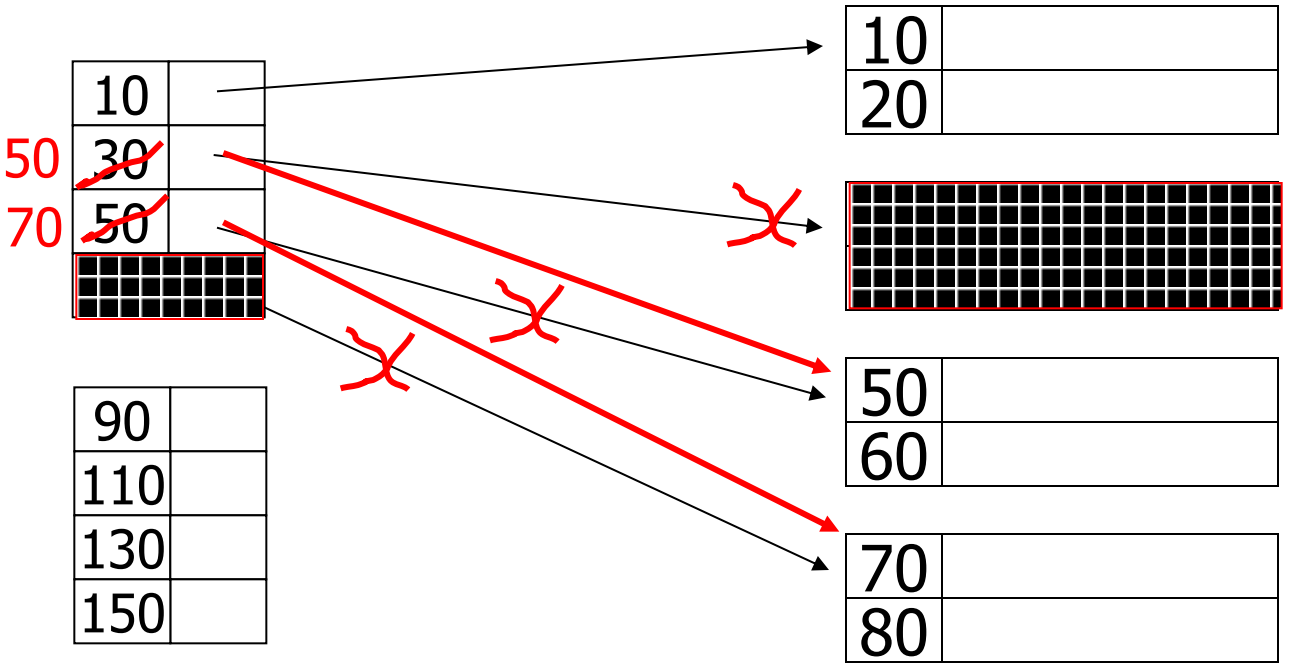
删除影响稀疏索引

– 删除记录30 和 40

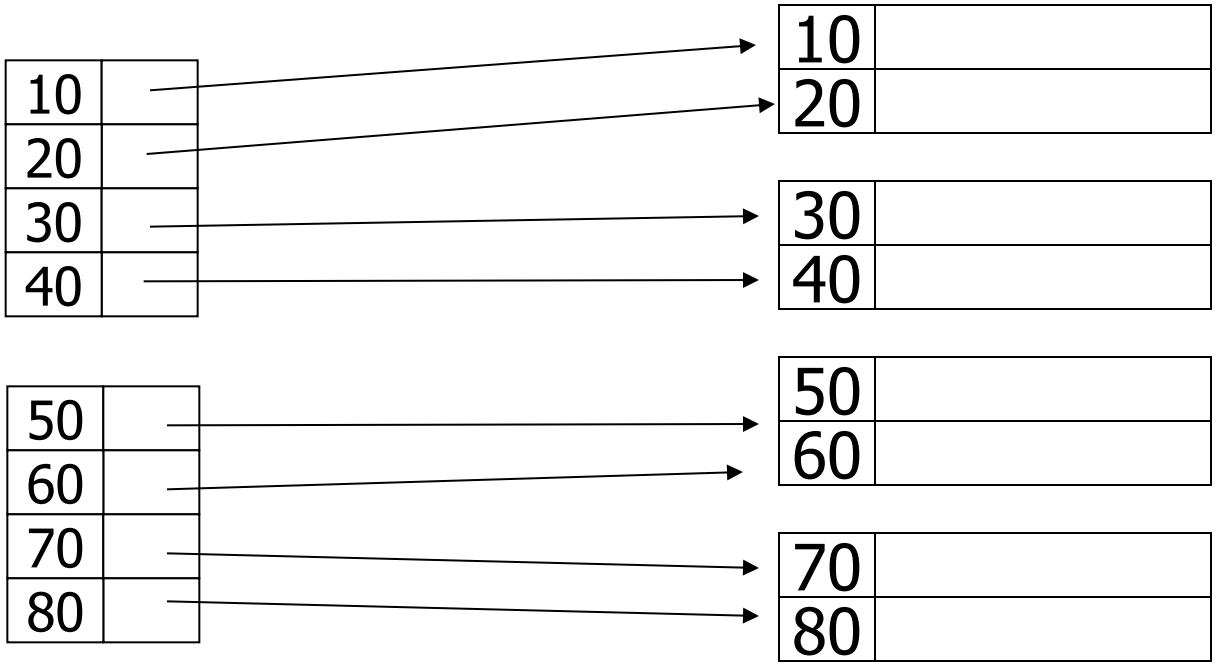


删除影响稀疏索引

– 删除记录30 和 40

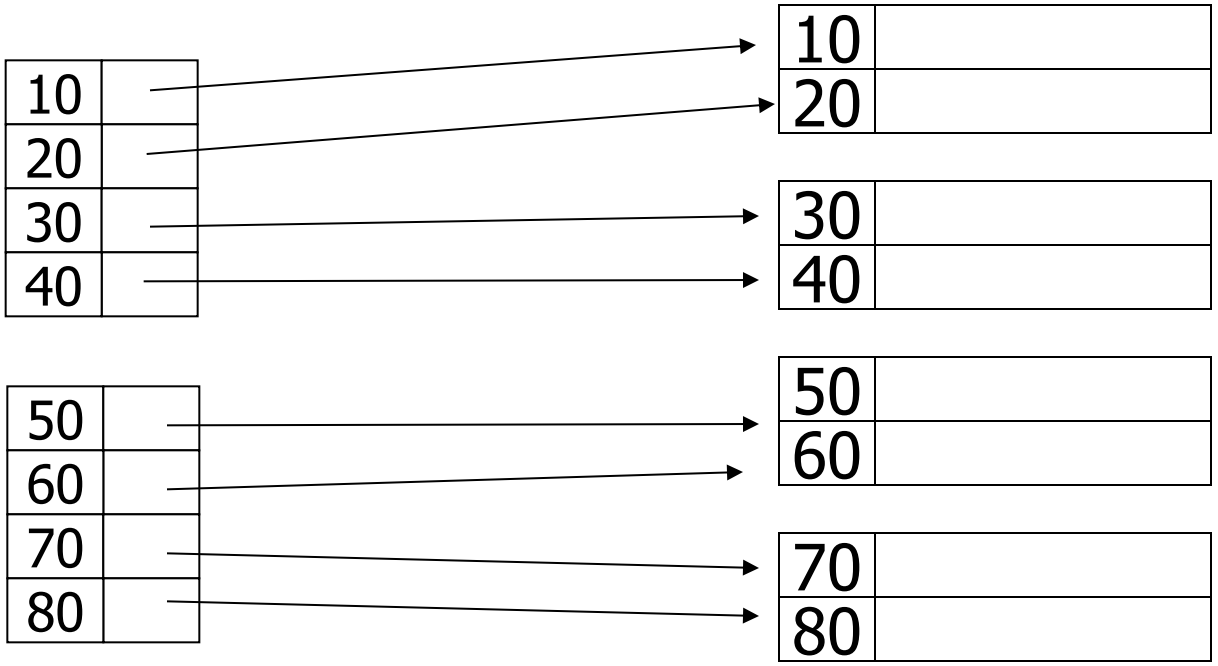


删除影响稠密索引



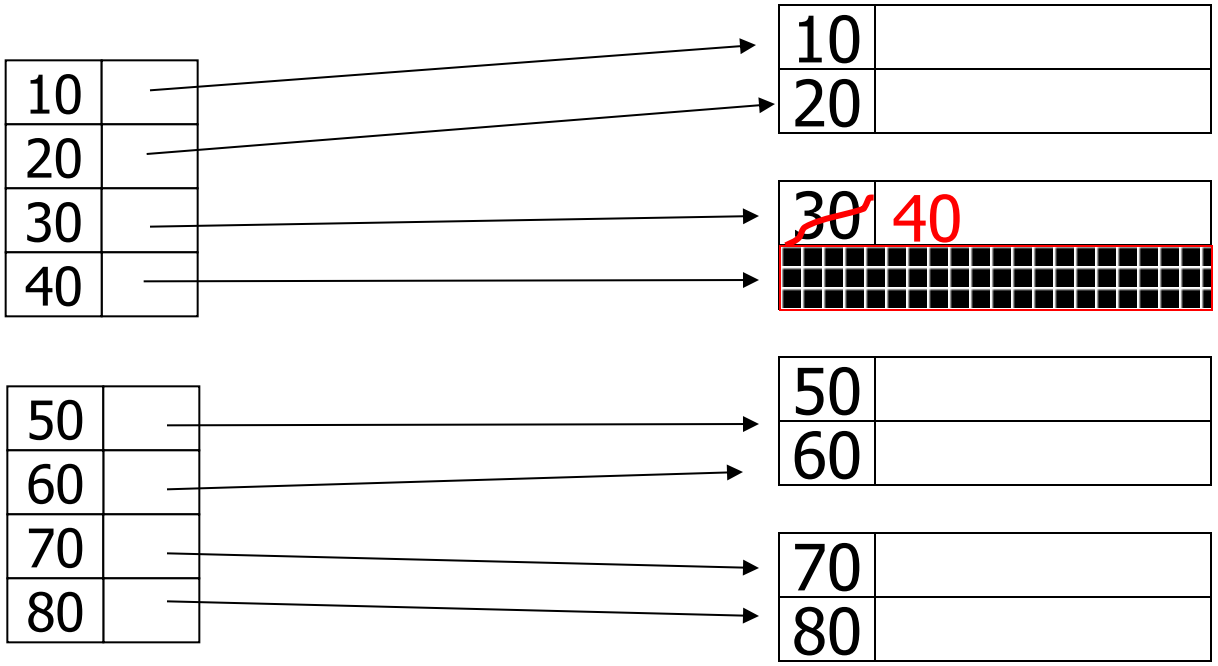
删除影响稠密索引

– 删除记录 30



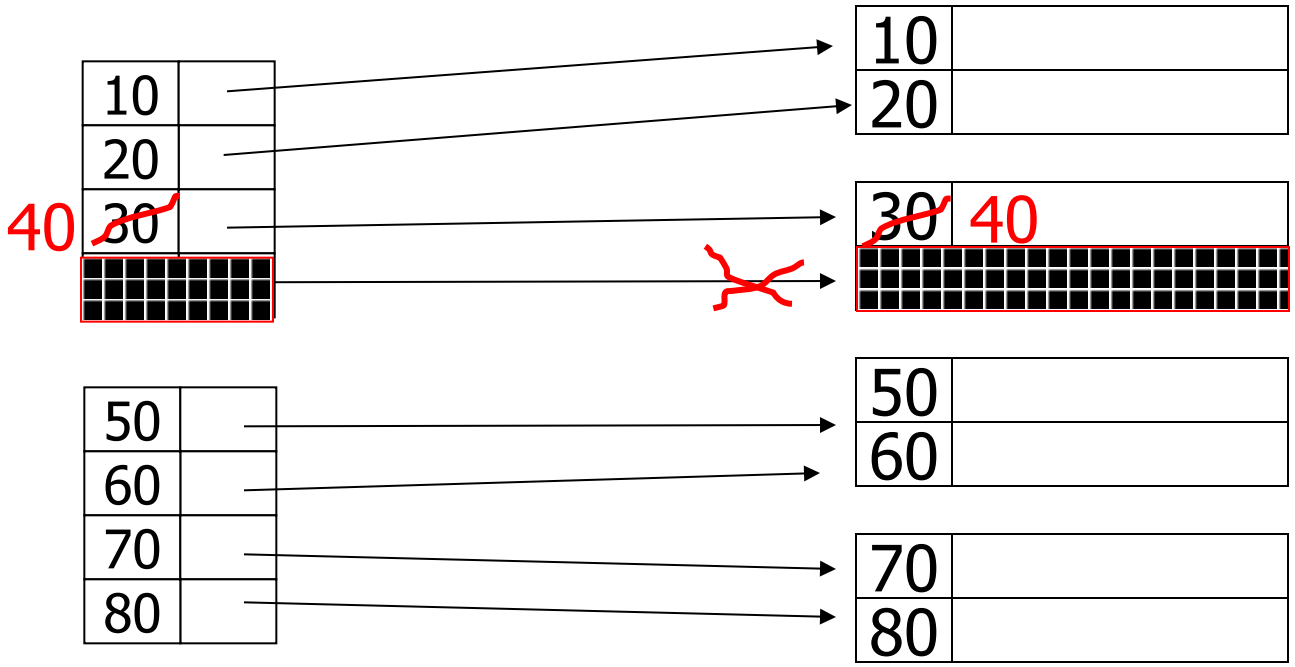
删除影响稠密索引

– 删除记录 30

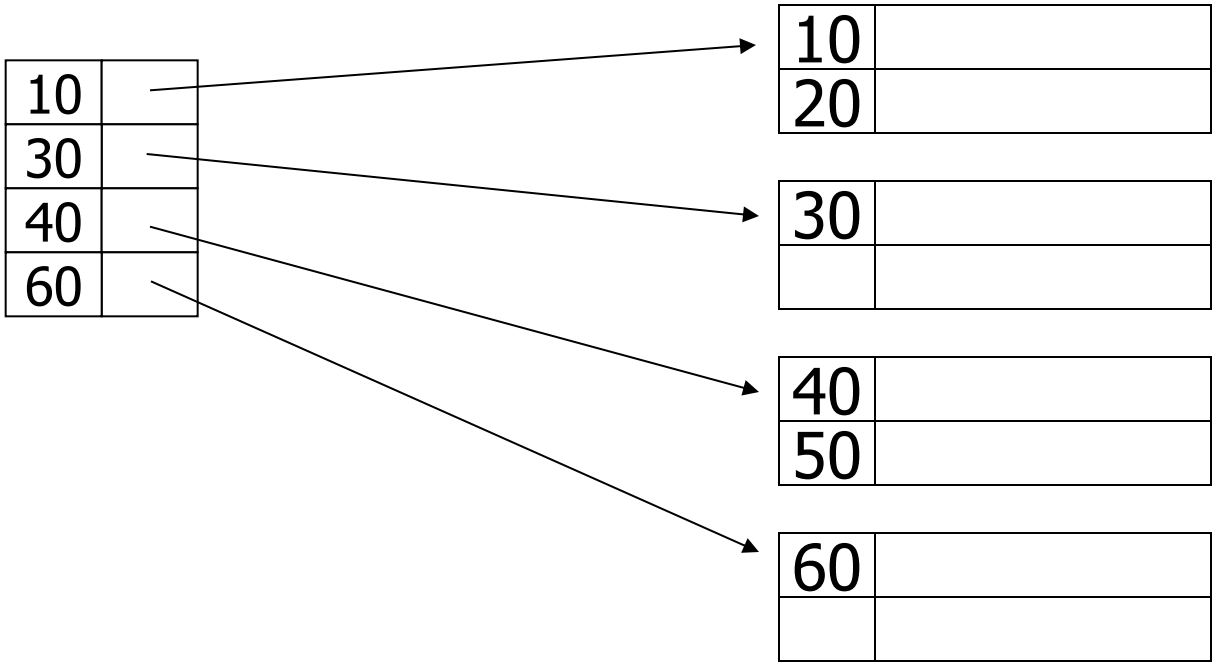


删除影响稠密索引

– 删除记录 30

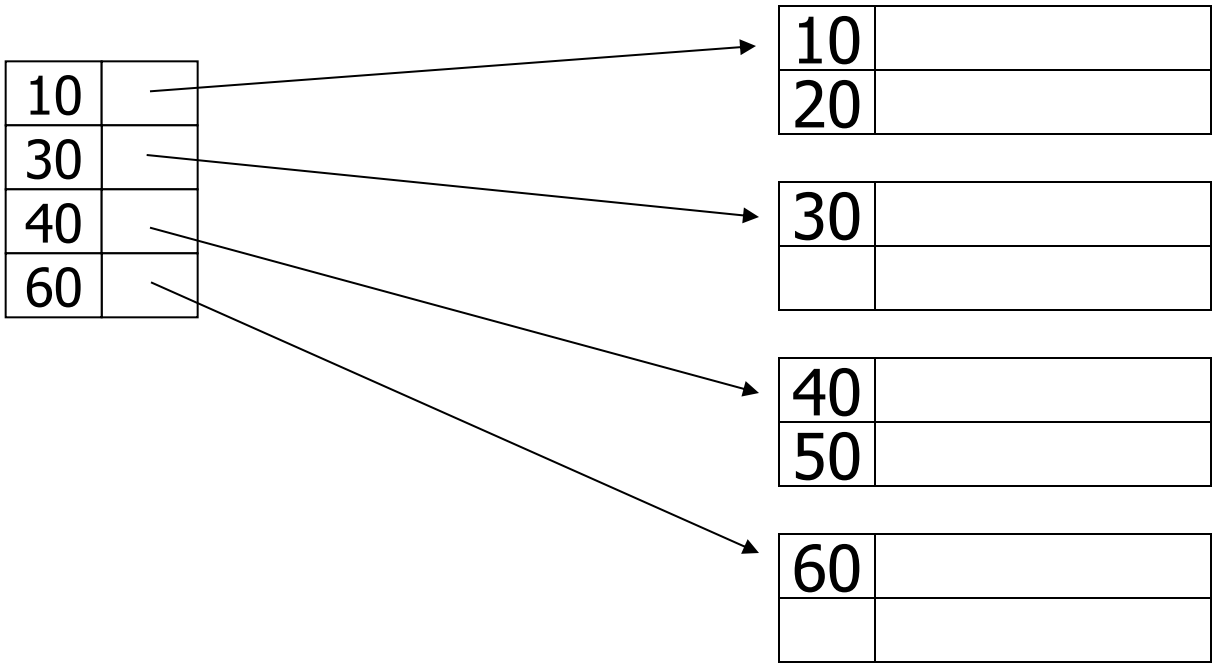


插入影响稀疏索引



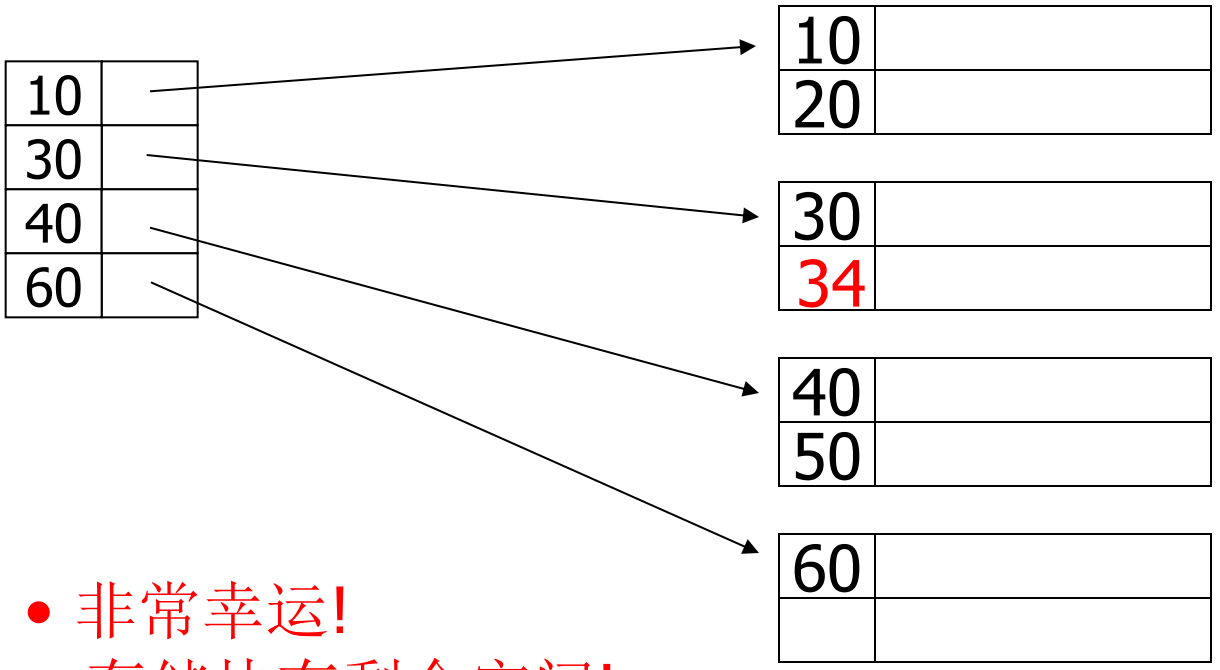
插入影响稀疏索引

– 插入记录 34



插入影响稀疏索引

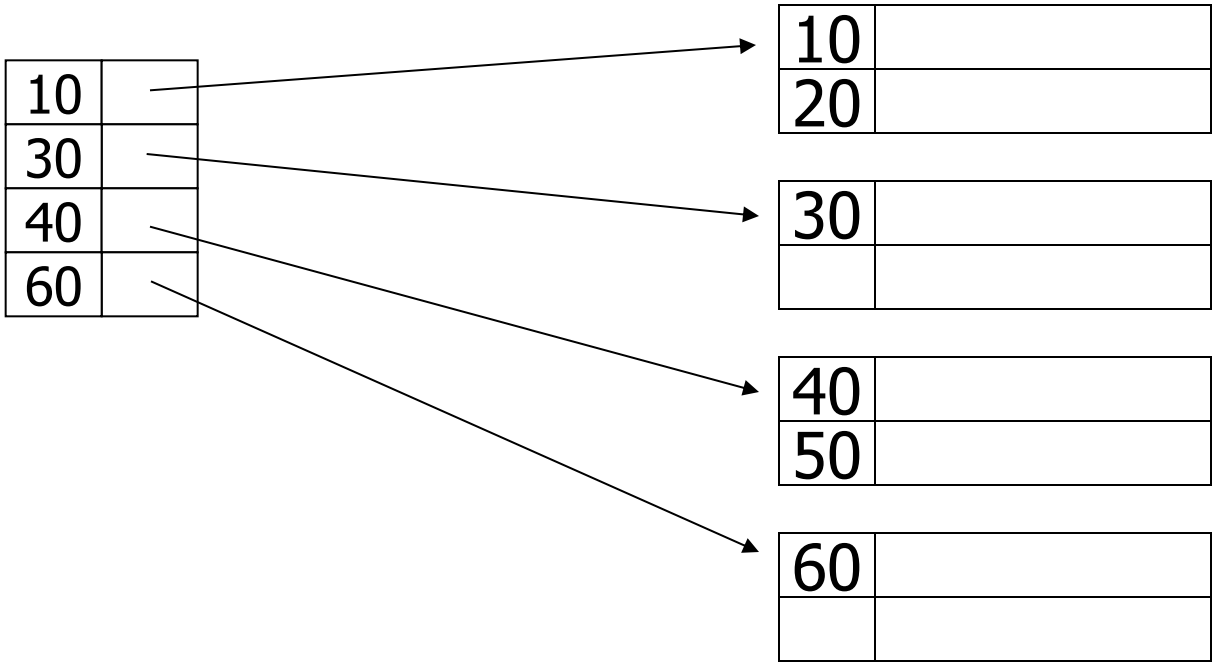
– 插入记录 34



- 非常幸运!
  存储块有剩余空间!

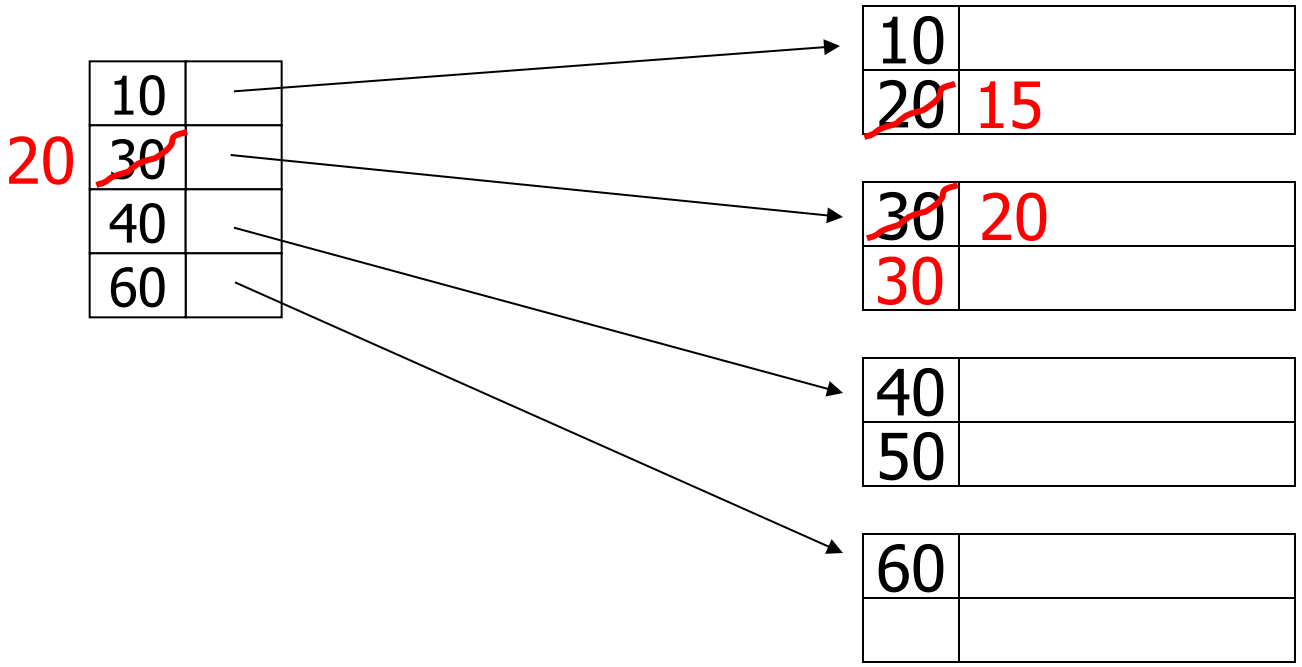
插入影响稀疏索引

– 插入记录 15



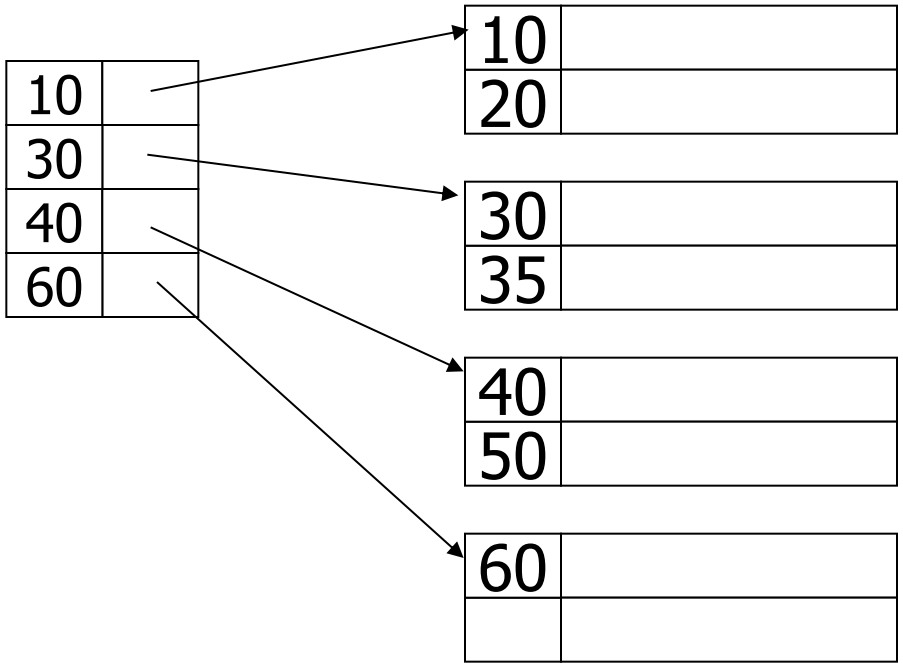
插入影响稀疏索引

– 插入记录 15



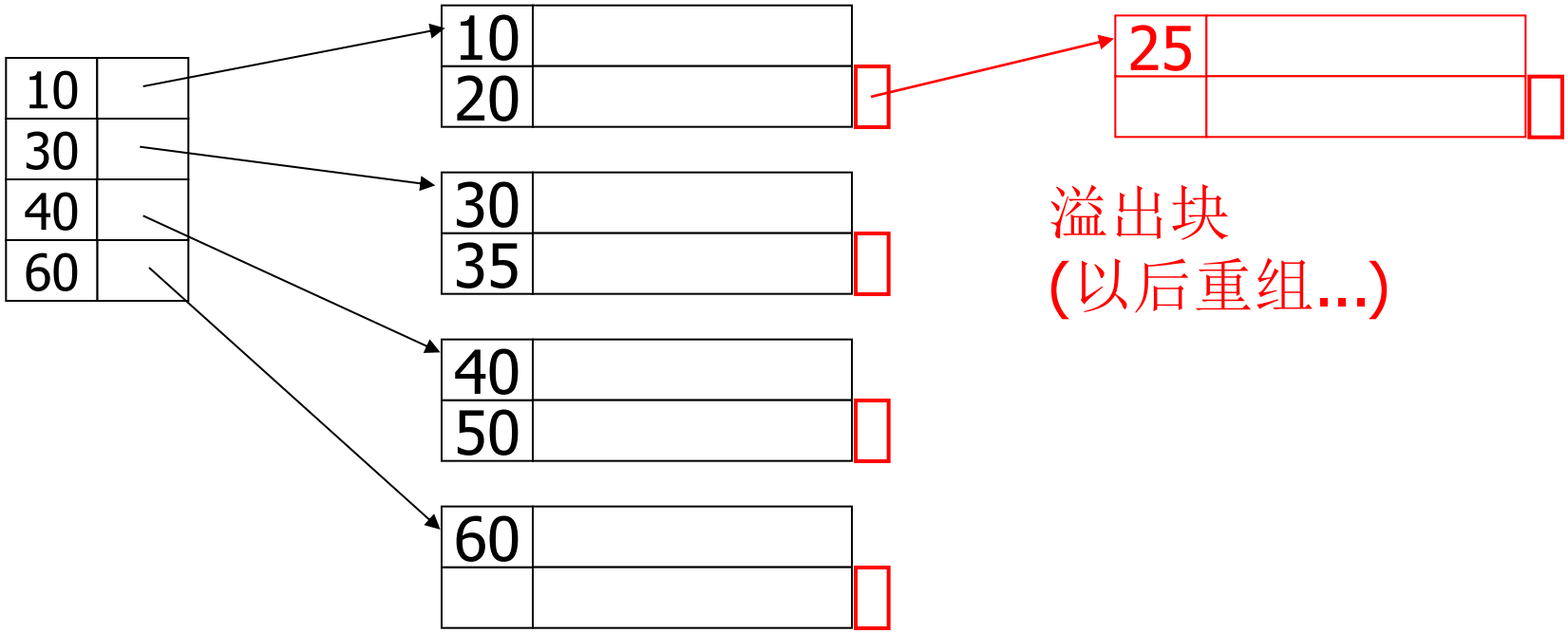
插入影响稀疏索引

– 插入记录 25



插入影响稀疏索引

– 插入记录 25



插入影响稠密索引

- 类似
- 代价更昂贵. . .

总结：

（1）对溢出块和顺序块的影响原则：稠密索引针对记录，稀疏索引针对基本块（不包含溢出块）。

（2）增加和删除记录只有影响块内的第一个记录时，稀疏索引才受影响。

（3）移动记录时，只有变成了块内的第一个记录，稀疏索引才受影响。

总结：

稠密索引维护原则：数据文件记录删除后，可以移动合并，也可只做删除标记（其他记录位置不移动）；索引文件中索引项删除后，索引项要移动进行合并（索引文件实际是顺序文件）。

稀疏索引维护原则：数据文件记录删除后，记录在块内位置移动；如果影响到块内的第一个记录，则索引项要修改。如果整个块删除，则索引文件的索引项也要进行移动合并。

总原则：索引文件内部始终进行移动、合并或修改。

插入时：利用临近空闲来移动，如果没有则用溢出块

思考题

假定一个存储块可存放5个记录，或20个键—指针对。已知有n个记录，如果表示成n的函数，创建以下两种数据文件各需要多少个数据块:1)稠密索引;2)稀疏素引?

辅助索引

（1）一个关系上只有一个主索引，但可以有多个辅助索引。

（2）主索引决定了数据文件中记录的排列顺序(因此只能一个)。

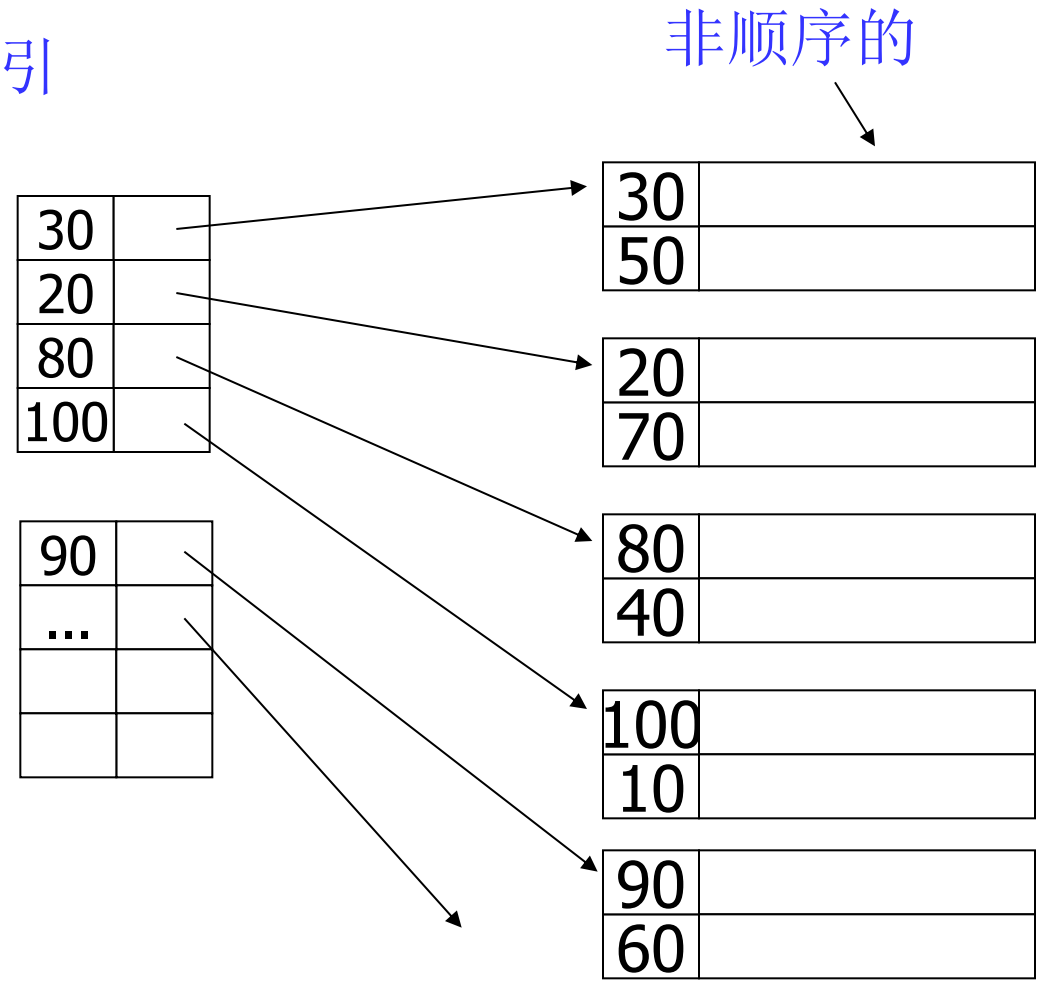
（3）辅助索引不影响数据文件中记录的存放位置，而只告诉某个记录的存放位置。

（4）直接建立在数据文件上的稀疏的辅助索引没有作用，辅助索引必须是稠密的，并且通常有重复的键值项。

参见pp.103图4-15

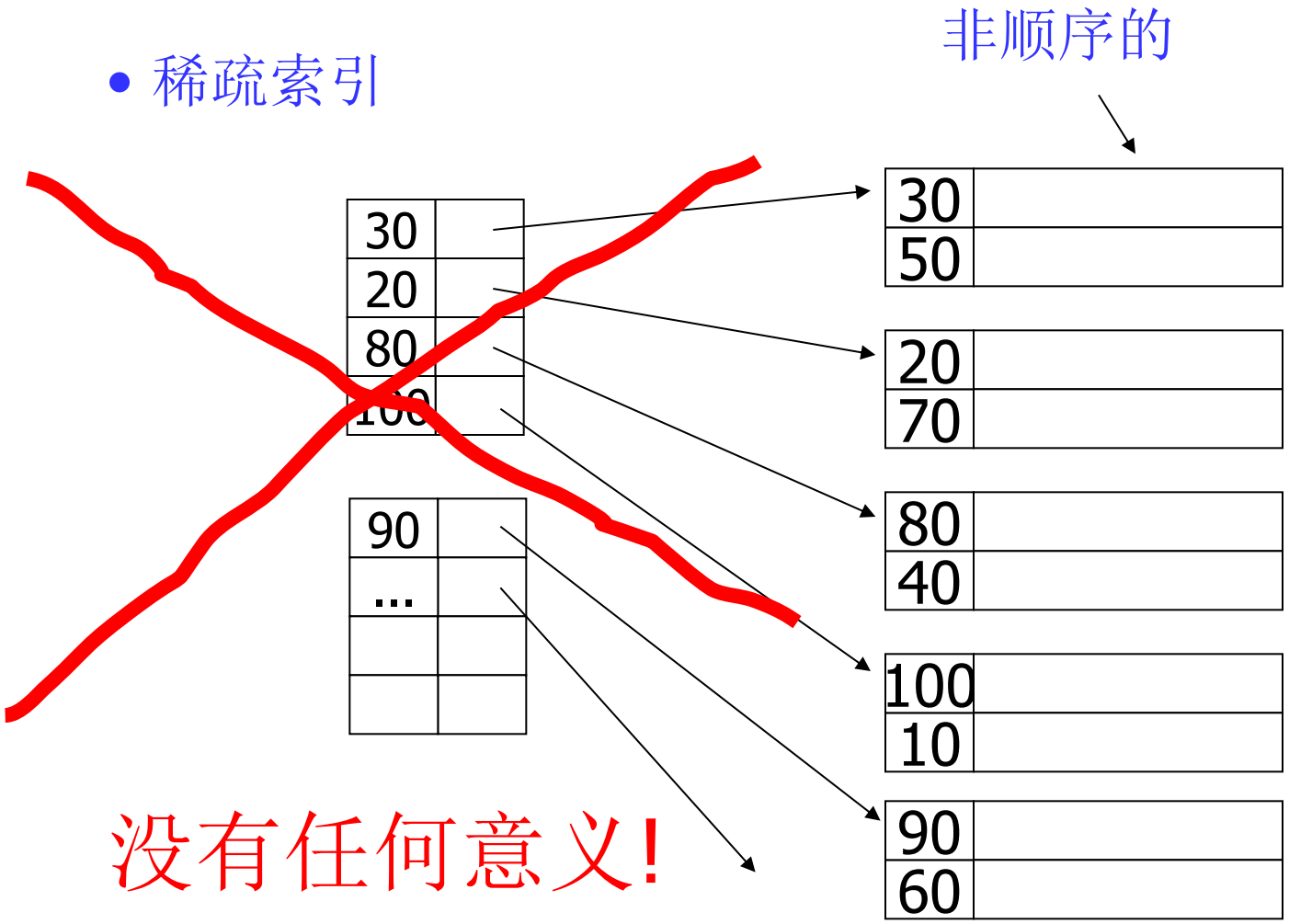
辅助索引

- 稀疏索引



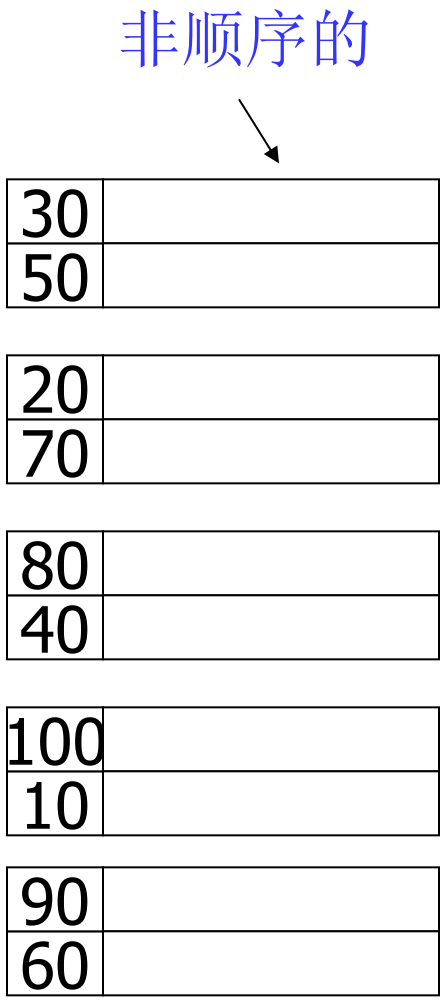
辅助索引

- 稀疏索引



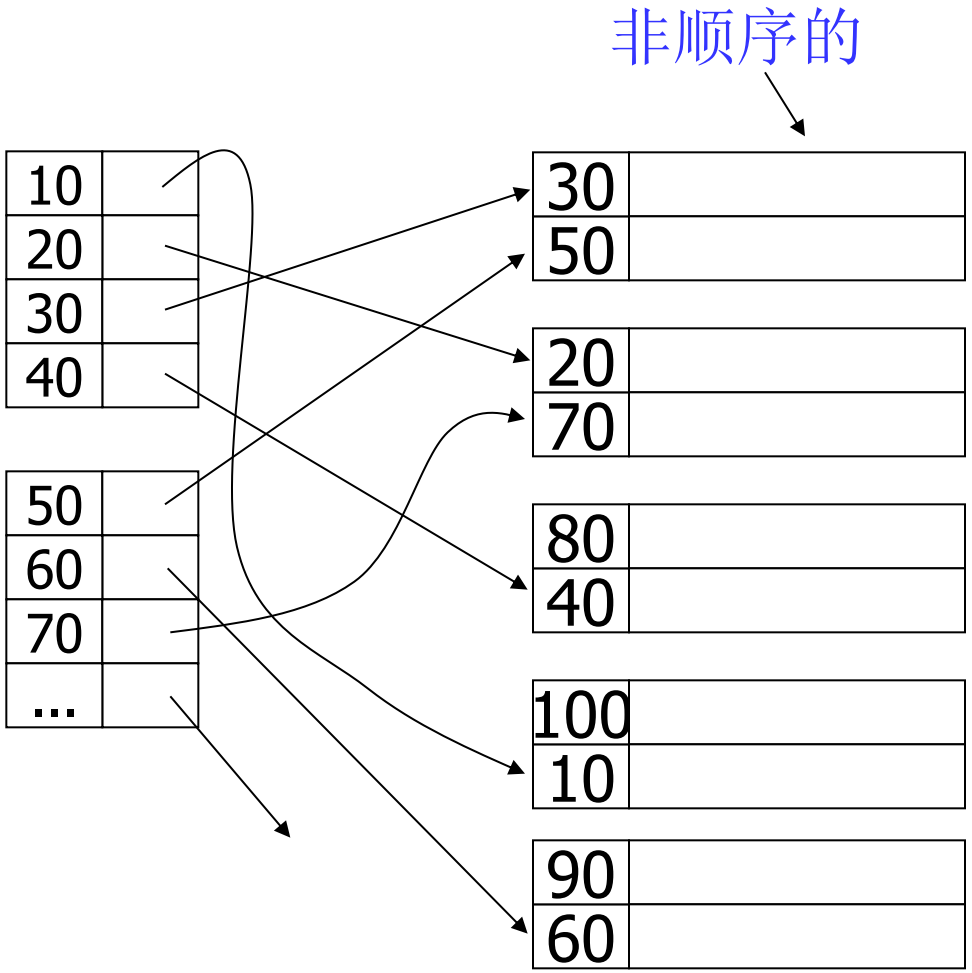
辅助索引

- 稠密索引



辅助索引

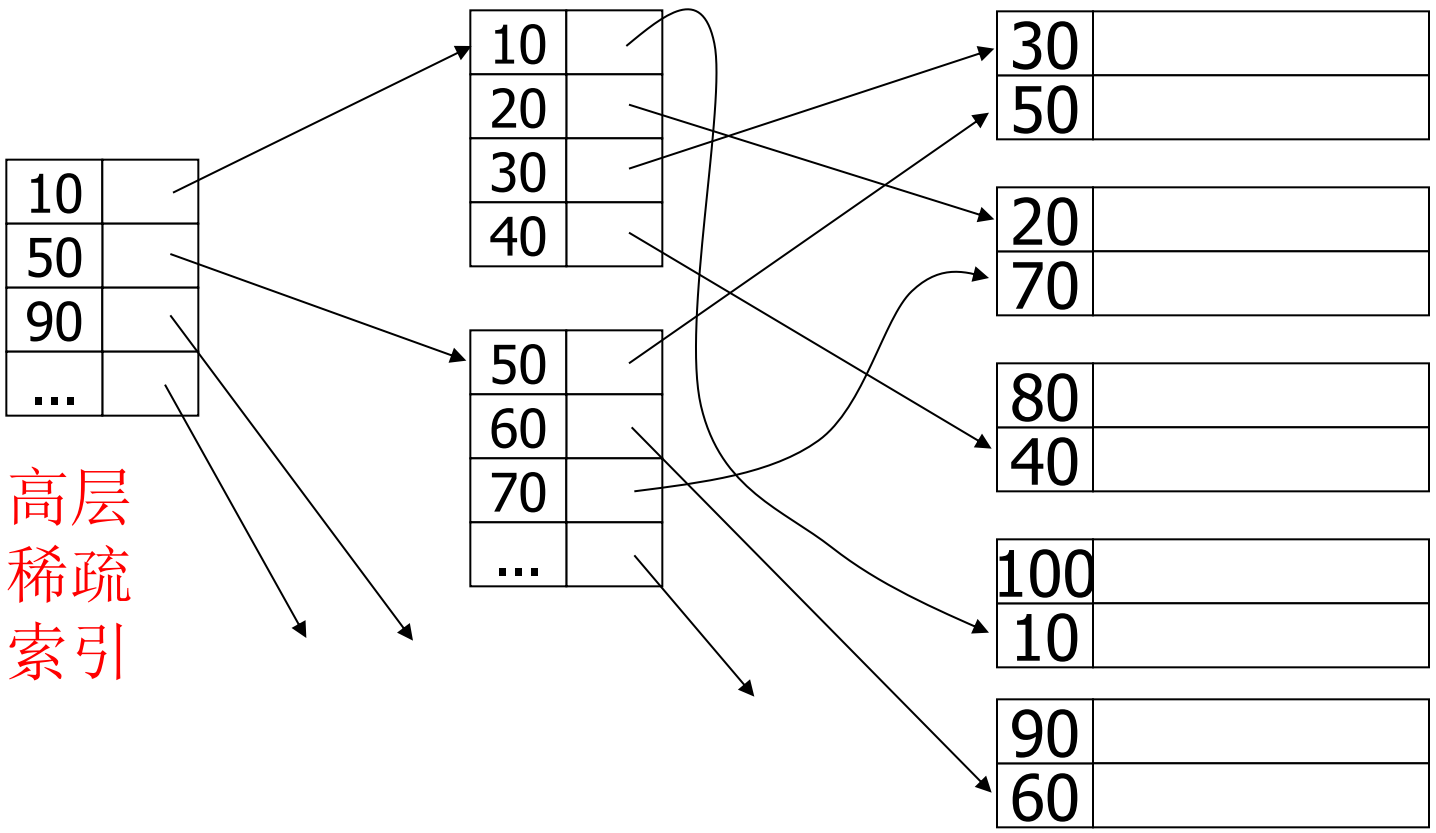
- 稠密索引



辅助索引

- 稠密索引

非顺序的



辅助索引总结

- 最底层用稠密索引
- 其余层用稀疏索引

辅助索引——应用

辅助索引除了可以辅助排序文件上非主键属性查询外，也可以作下列文件的主键索引

（1）堆文件：这种数据文件的记录没有按照某个属性排序，物理顺序由先来后到决定。

（2）聚集文件（有时翻译成聚簇文件），将多个关系的元组存储在同一个文件中。（参见下页例子）

辅助索引——聚集文件应用

举例：

Movie(title,year,length,genre,studioName,producerC#)

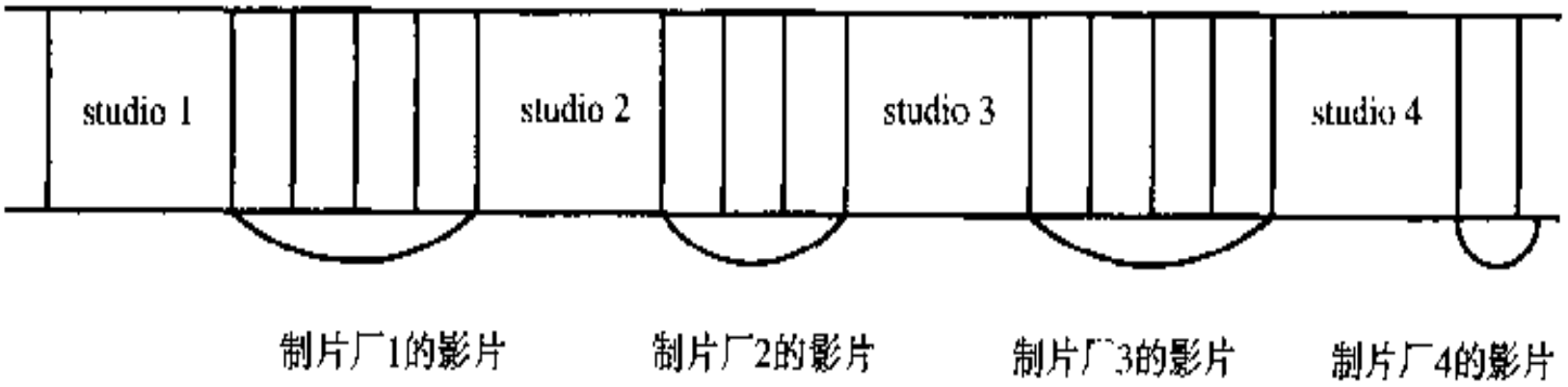
Studio(name,address,pressC#)

假定典型查询如下：

```
Select title,year
From Movie,Studio
Where pressC#='zzz' and Movie.studioName=Studio.name
```

可以为两个关系建立一个聚集文件，如pp.105图4-16，然后在pressC#上建立辅助索引，则可以大大加快查询速度。

辅助索引——聚集文件应用


studio 1
studio 2
studio 3
studio 4
制片厂1的影片
制片厂2的影片
制片厂3的影片
制片厂4的影片

重复值和辅助索引(pp.105)

20	
10	

20	
40	

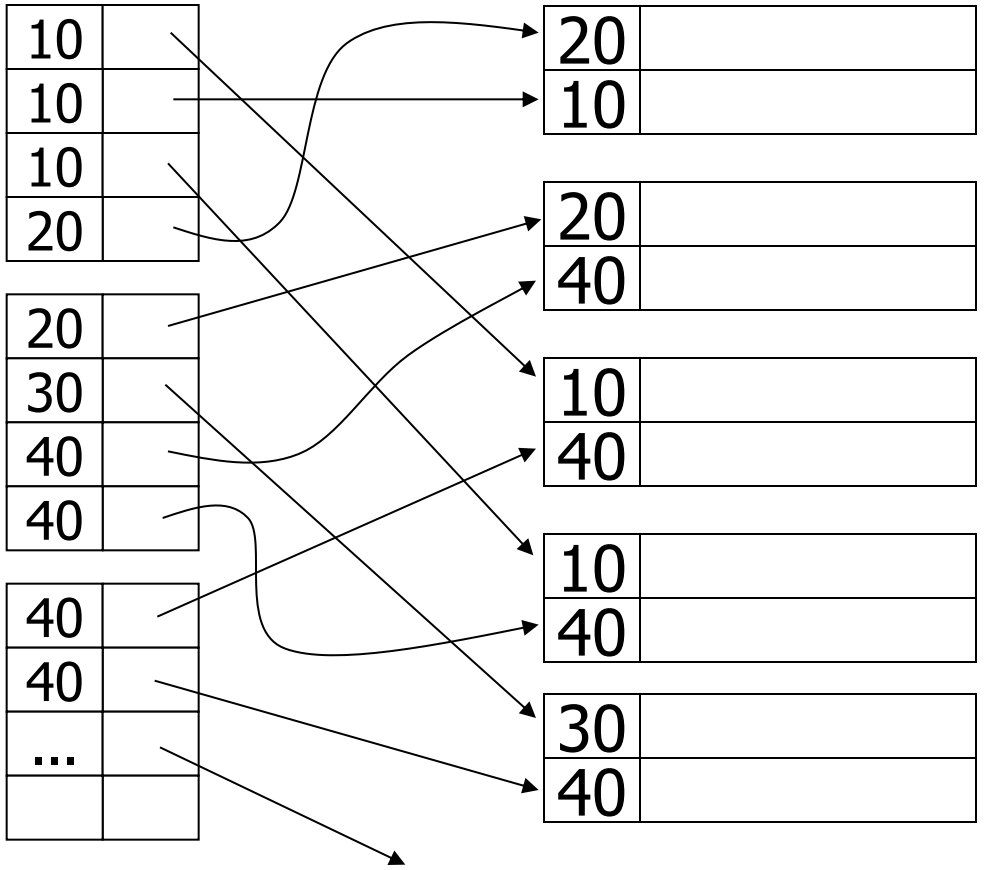
10	
40	

10	
40	

30	
40	

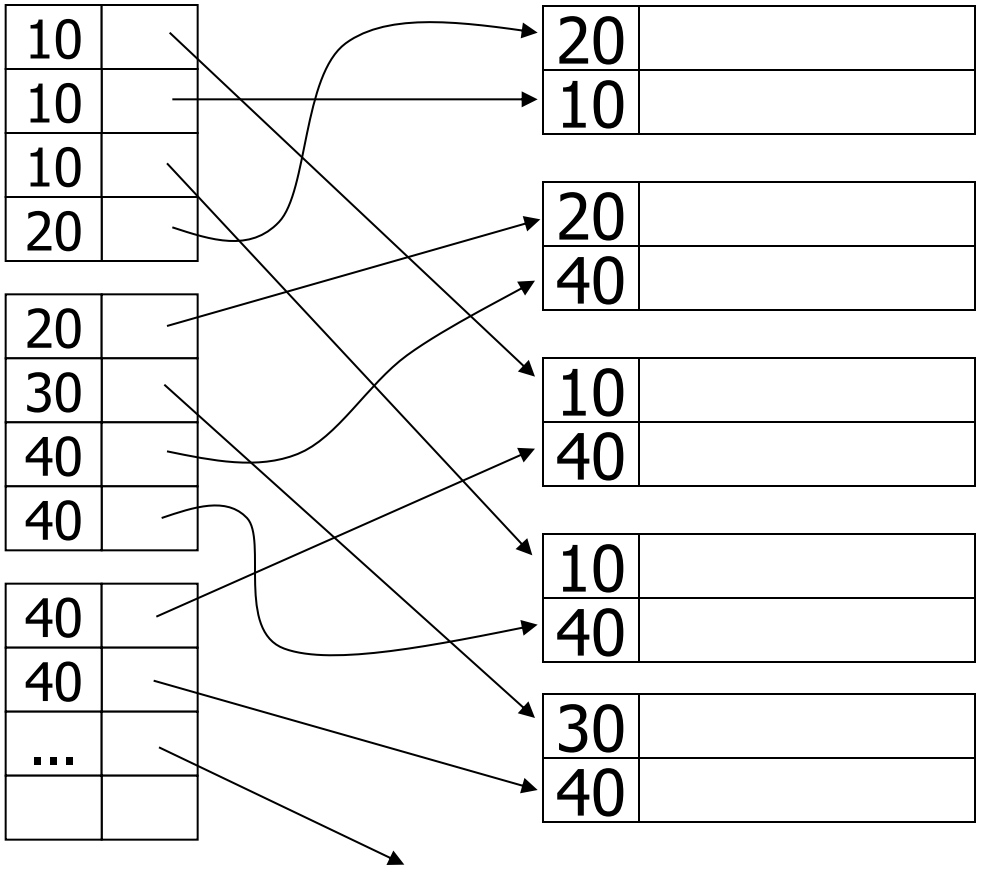
重复值和辅助索引（pp.105）

（1）一种选择...



重复值和辅助索引

（1）一种选择...



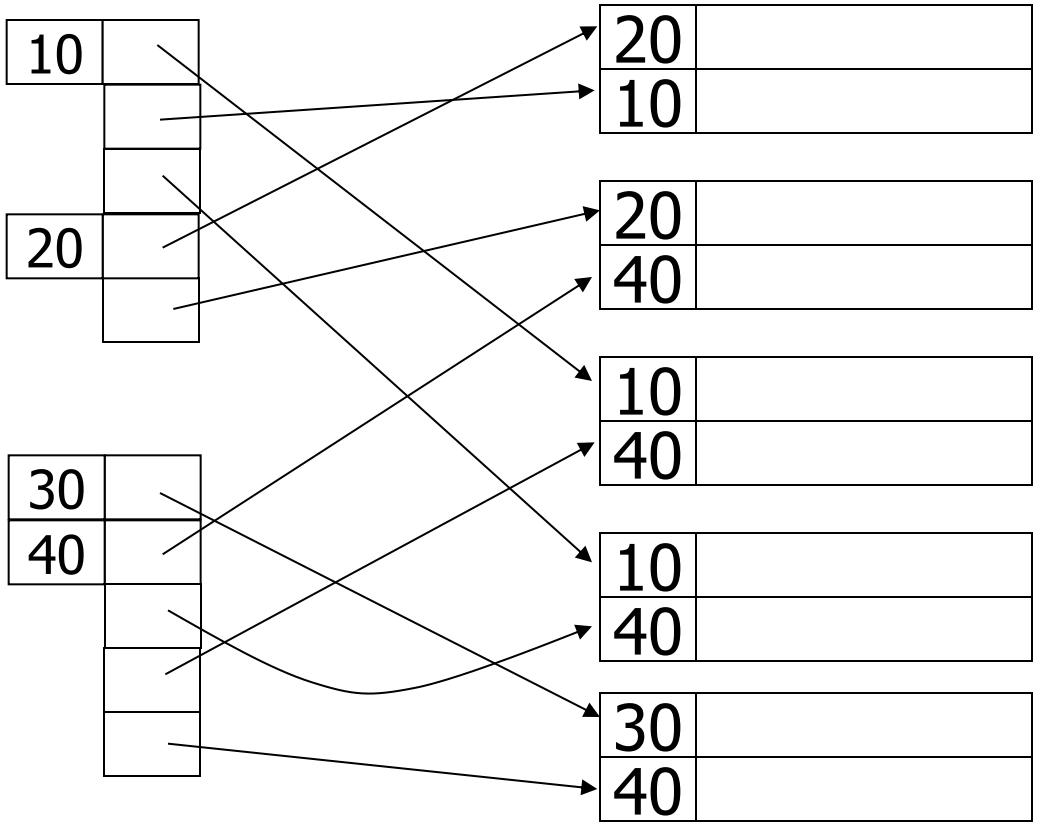
问题:

过多的代价!

- 磁盘空间
- 搜索时间

重复值和辅助索引

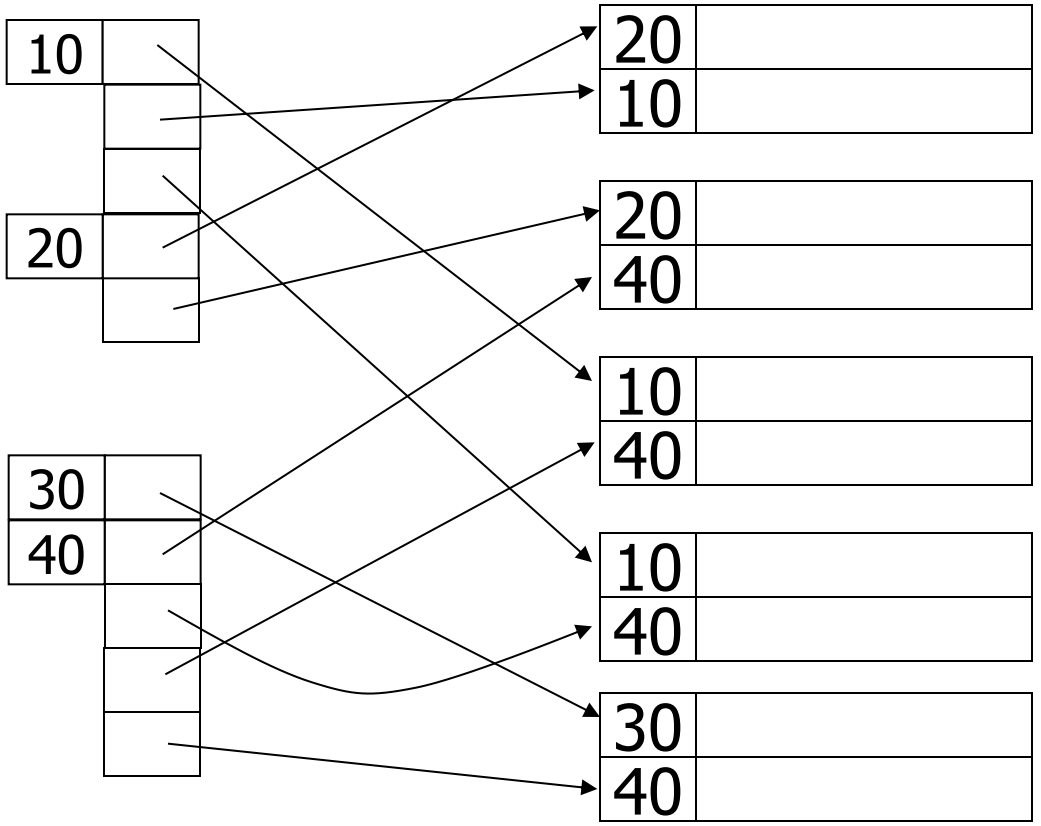
（2）另一种选择...



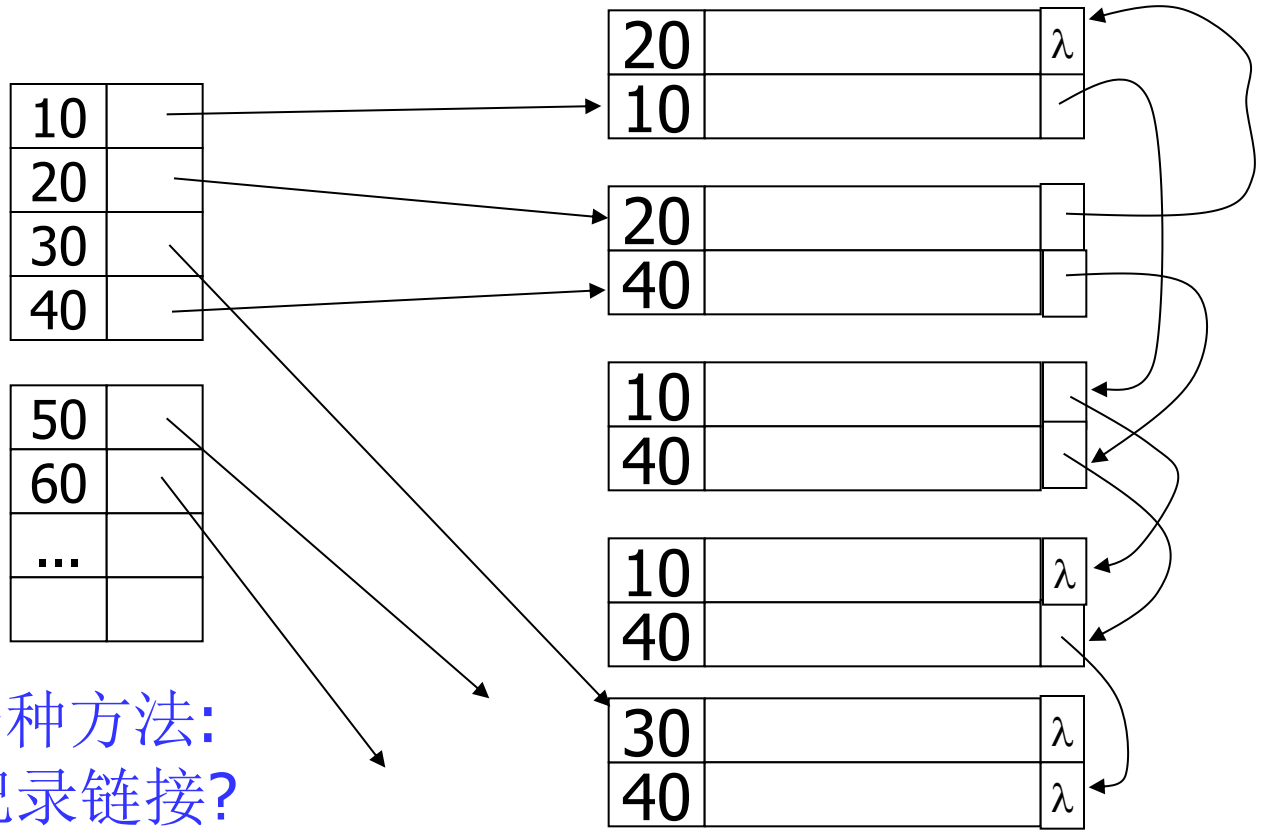
重复值和辅助索引

（2）另一种选择...

问题：
索引项中指针数目不一，导致指针项长度无法统一!

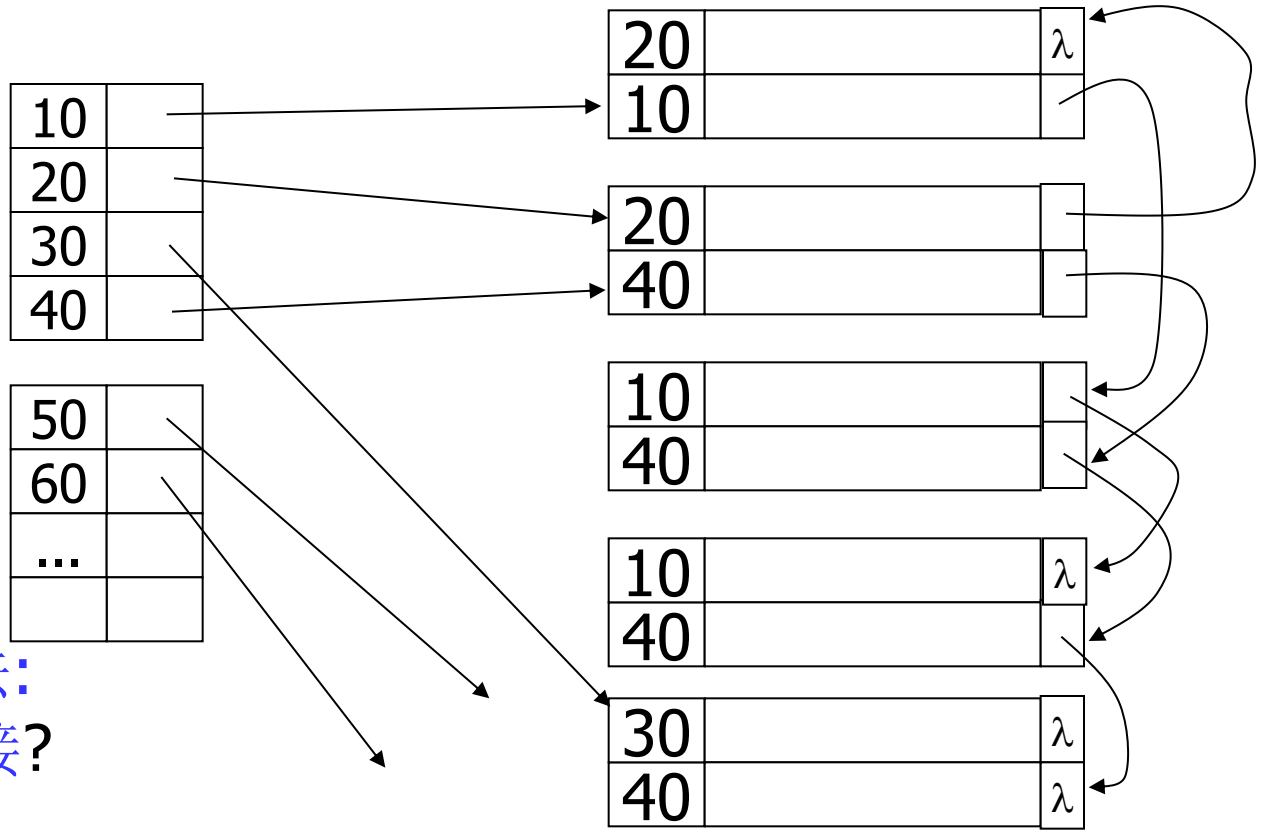


重复值和辅助索引



（3）另一种方法:
相同键的记录链接?

重复值和辅助索引



（3）另一种方法:
相同键的记录链接?

问题:

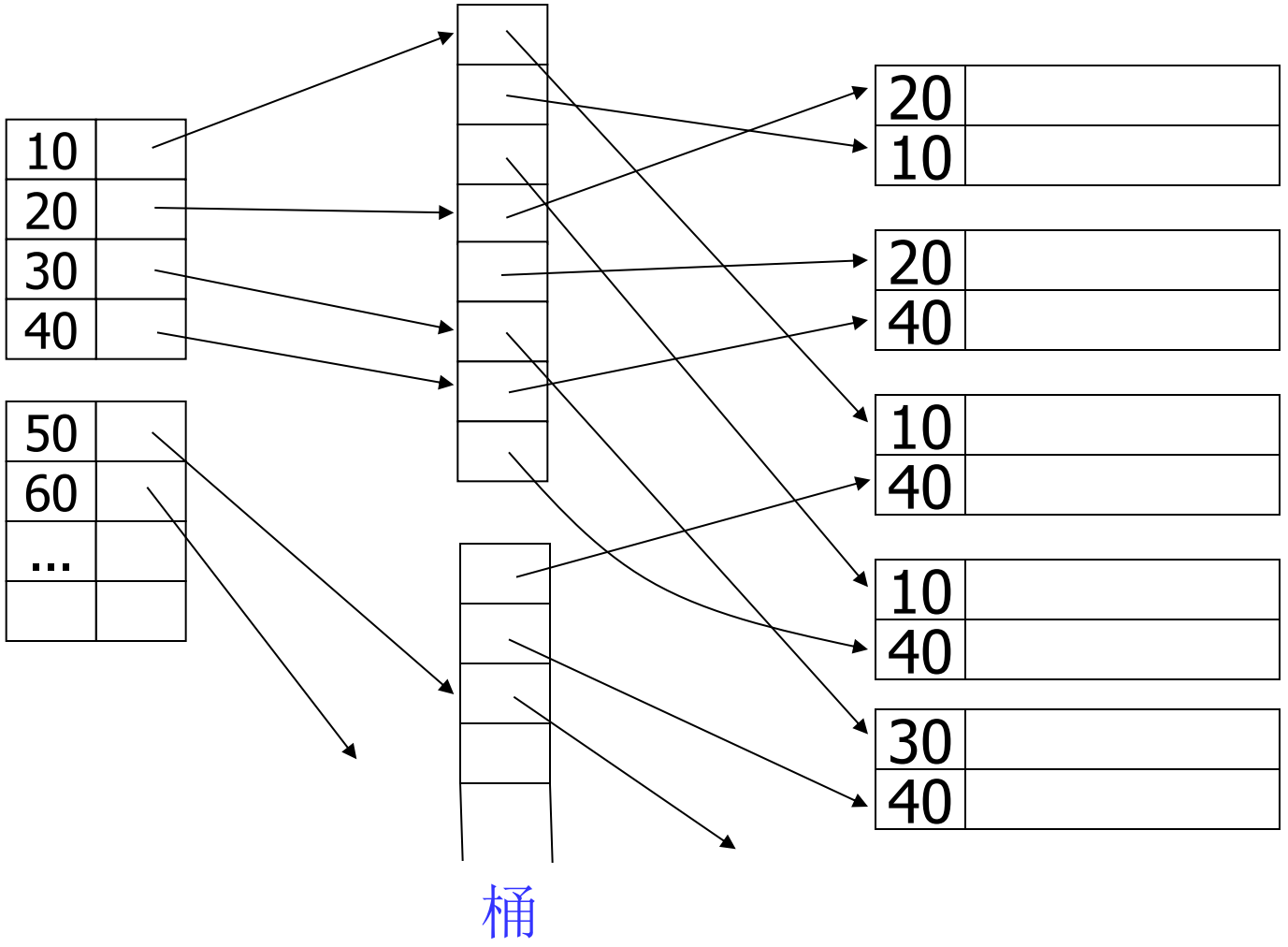
- 记录要增加新字段
- 必须依据链接寻找记录

辅助索引——索引中的间接(pp.105)

解决方案：

在辅助索引和数据文件中，引入一个桶文件，存放一个键对应的多个指针，参见pp.106图4-17.

辅助索引——索引中的间接



“桶” 为什么有用？

索引

Name: 主索引

Dept: 辅助索引

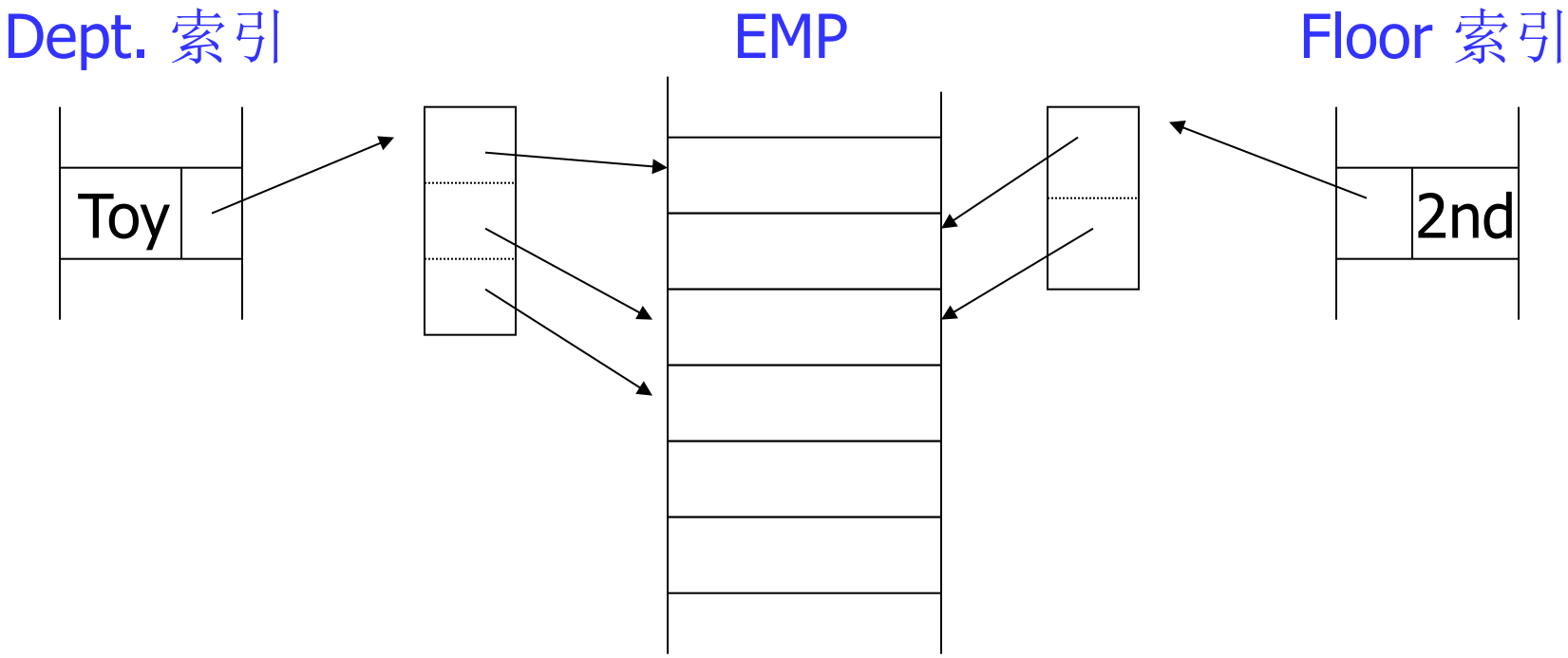
Floor: 辅助索引

表记录

EMP (Name,Dept,Floor,...)

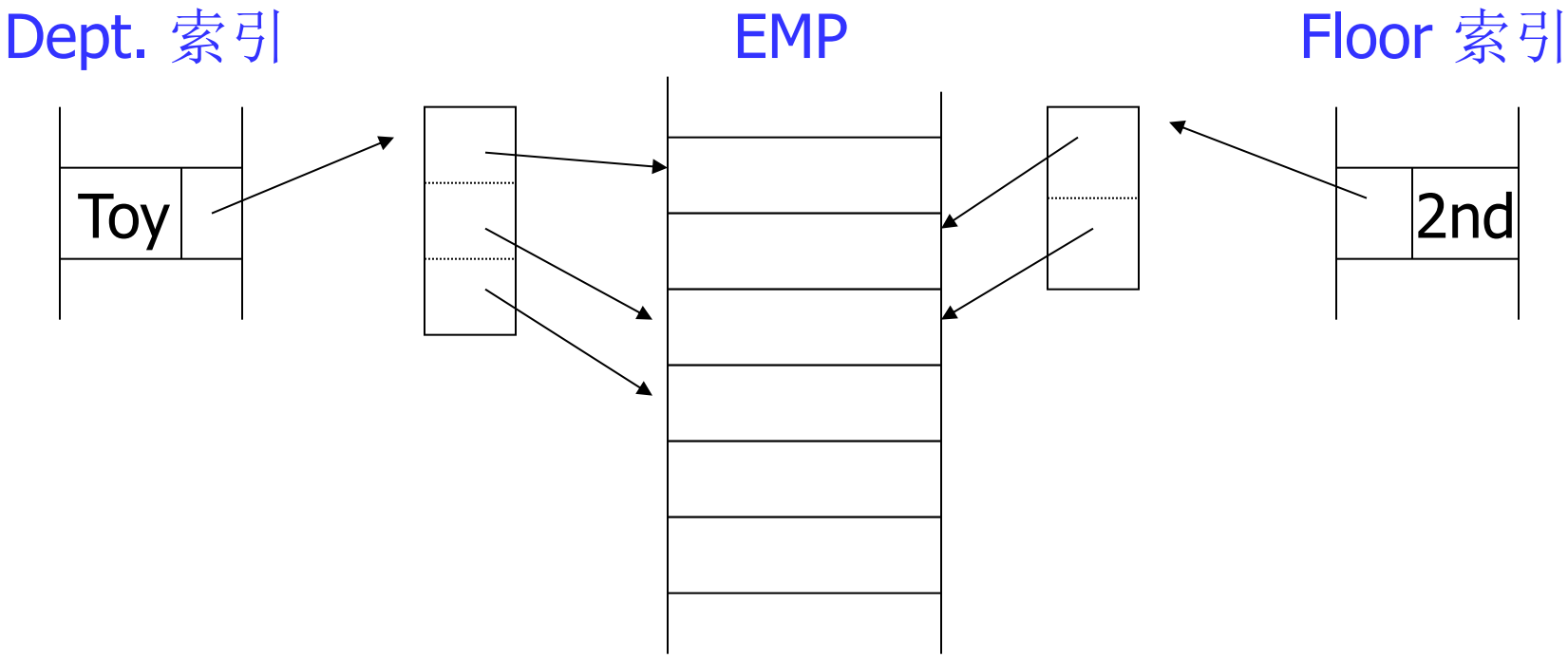
查询: 员工符合下列条件

(Toy Dept) ∧ (2nd floor)



查询: 员工符合下列条件

(Toy Dept) ∧ (2nd floor)



→ 对Toy 对应的指针和2nd 对应的指针求交集

通过索引块交集求取记录的思想在信息检索中应用

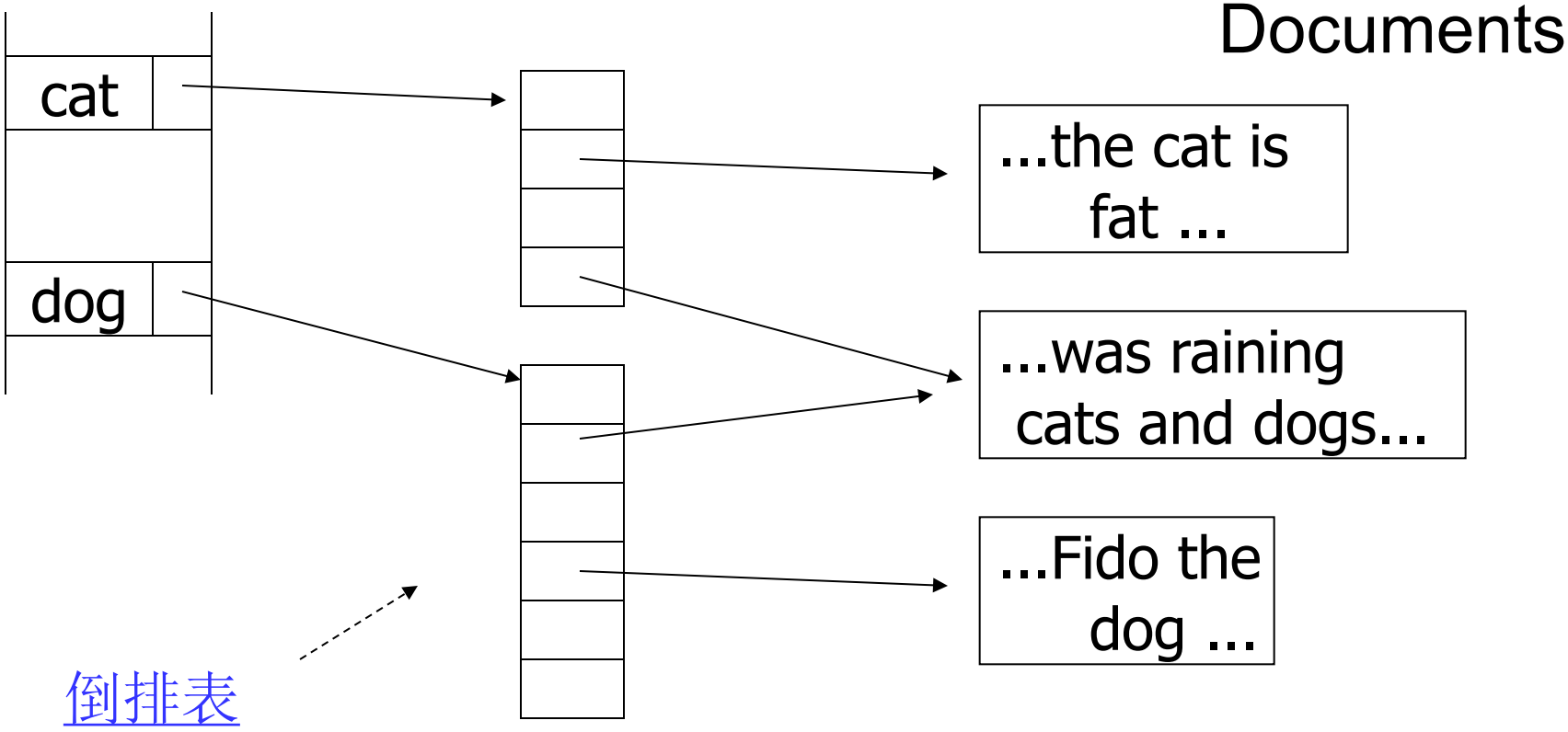
Documents

...the cat is fat ...

...was raining cats and dogs...

...Fido the dog ...

通过索引块交集求取记录的思想在信息检索中应用



辅助索引——倒排索引

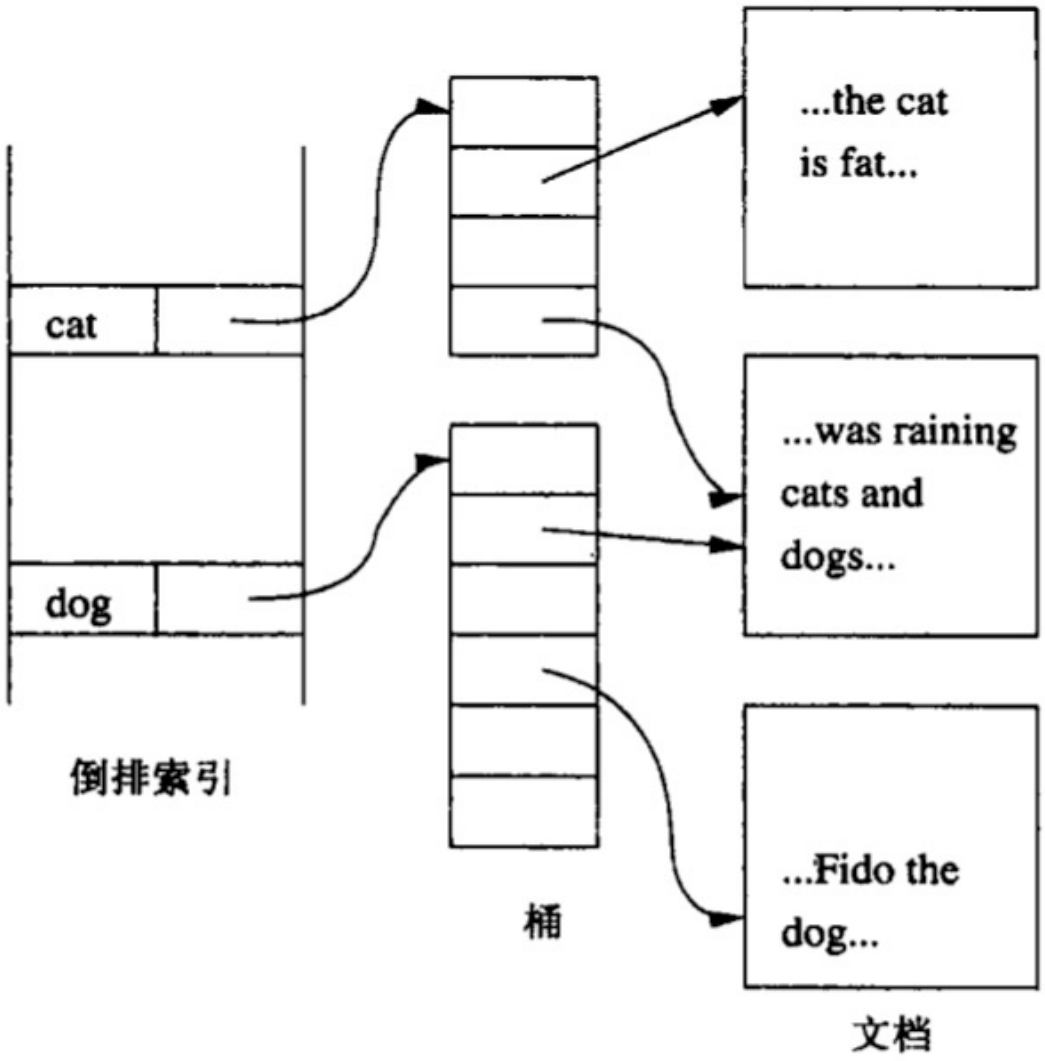
针对基于关键词的文档检索，具体要点

（1）一个文档被看作关系Doc的一个元组，这个关系有很多属性，每个属性对应于文档可能出现的一个词，每个属性可以看作是布尔型：Doc(hasCat,hasDog,…)

（2）关系Doc的每个属性上都建有辅助索引，并把所有索引合成一个，称为倒排索引，并利用间接桶来提高空间利用率。

（3）关键字可以采用“抽取词干”和删除“无用词”技术。

辅助索引——倒排索引


...the cat
is fat...
cat
...was raining
cats and
dogs...
dog
...Fido the
dog...
倒排索引
桶
文档

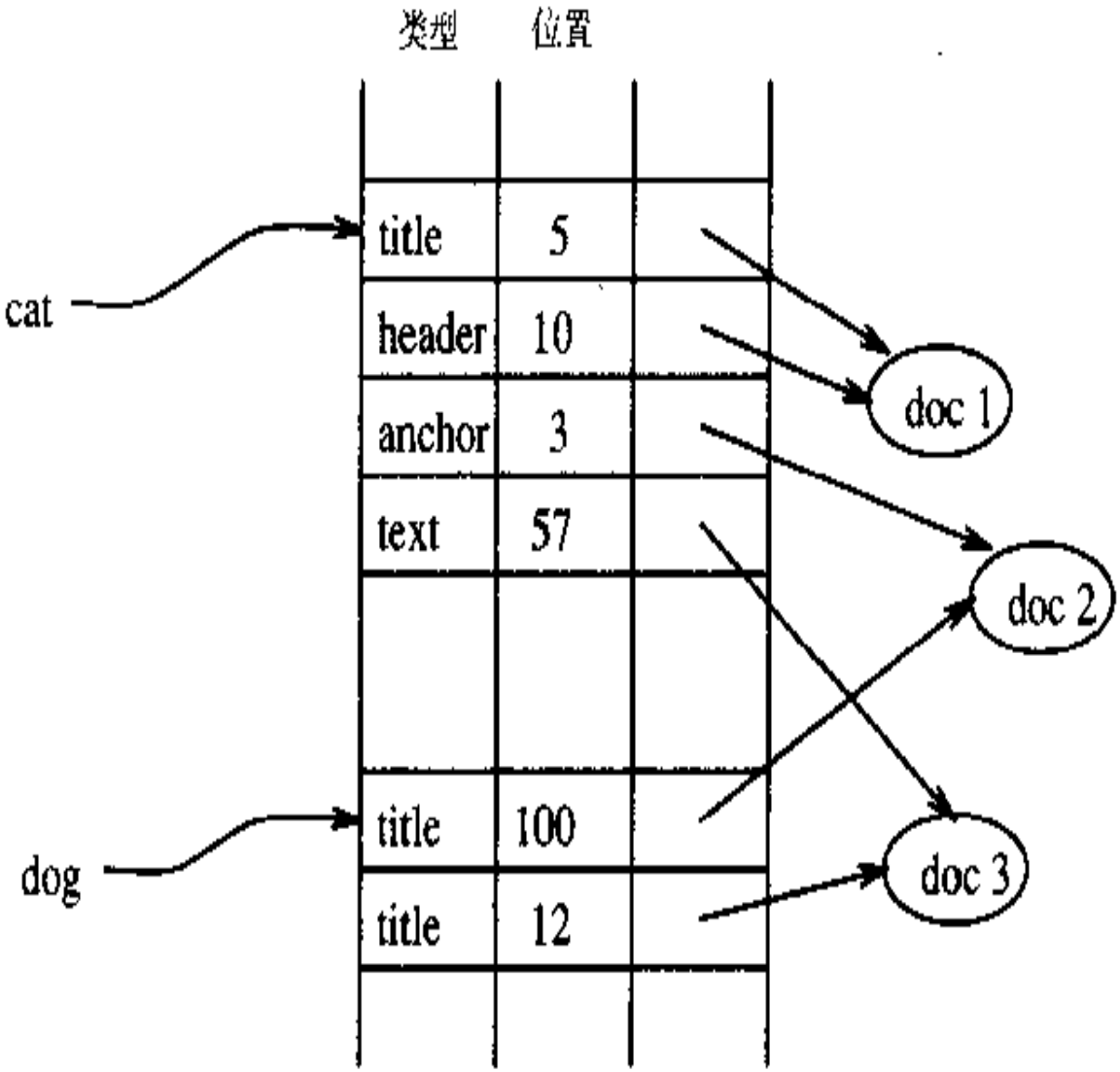
辅助索引——倒排文件

桶文件指针

（1）指针可指向文档本身

（2）指向词某一次出现

（3）指向词的多次出现，并增加其他信息，诸如类型、位置等。



总结：前述索引

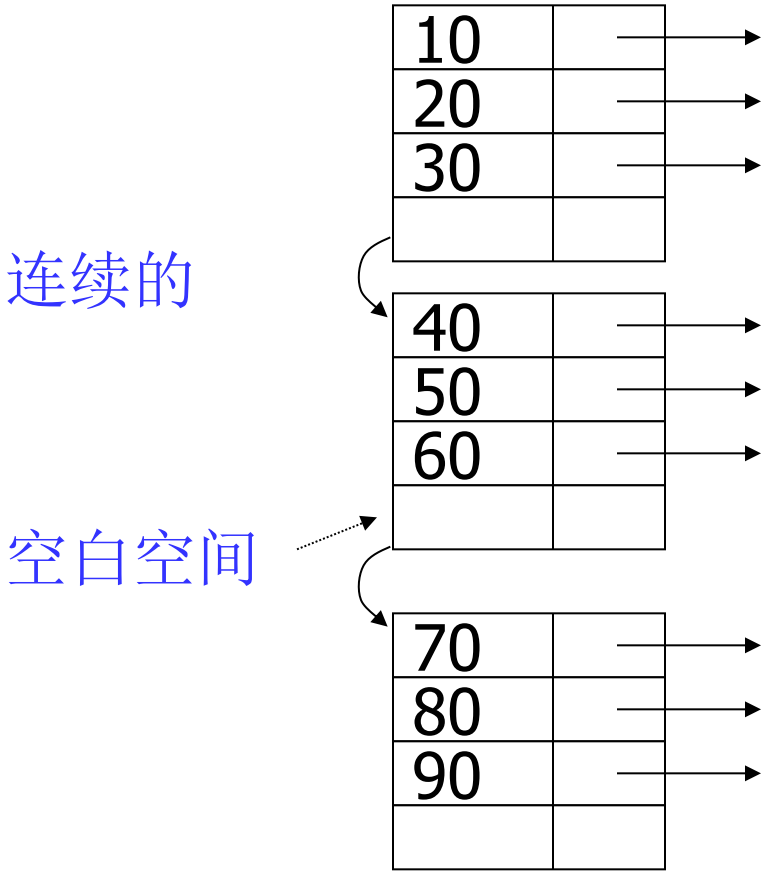
优点:

- 简单
- 索引是顺序文件，便于扫描

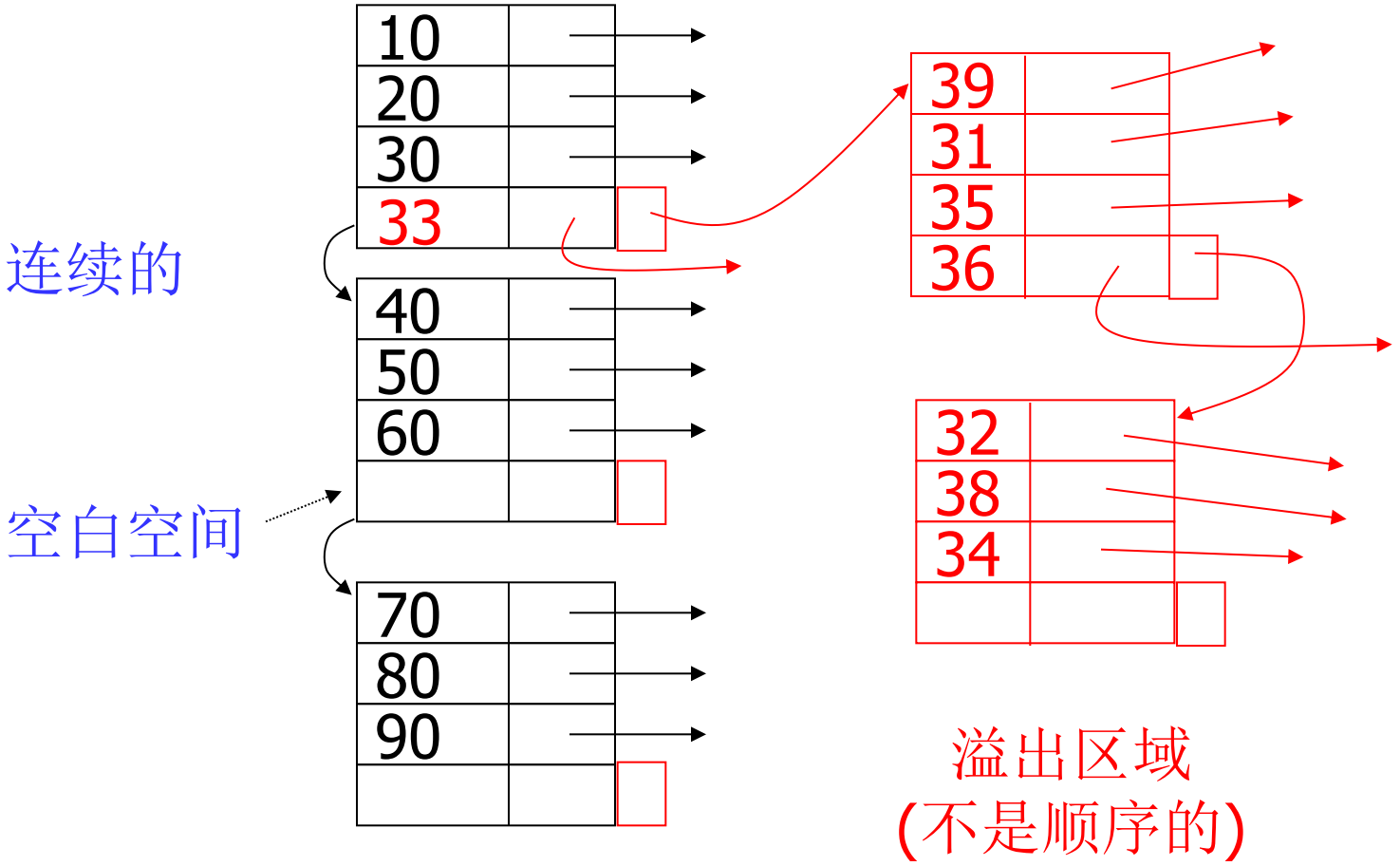
缺点：

- 插入代价高（全部重排序），并且有时
- 丢失顺序性（溢出）

例子　　索引 (顺序)



例子　　索引 (顺序)


10
20
30
33
40
50
60
70
80
90
39
31
35
36
32
38
34
连续的
空白空间
溢出区域
(不是顺序的)

- 承接: 另外一种索引
  - 放弃索引块间的*物理连续性*（前述索引整个索引顺序链接）
  - 努力取得“平衡”（各个叶子结点到根等长路径）

注意: 这种索引应是B+树，但本书都称为B 树

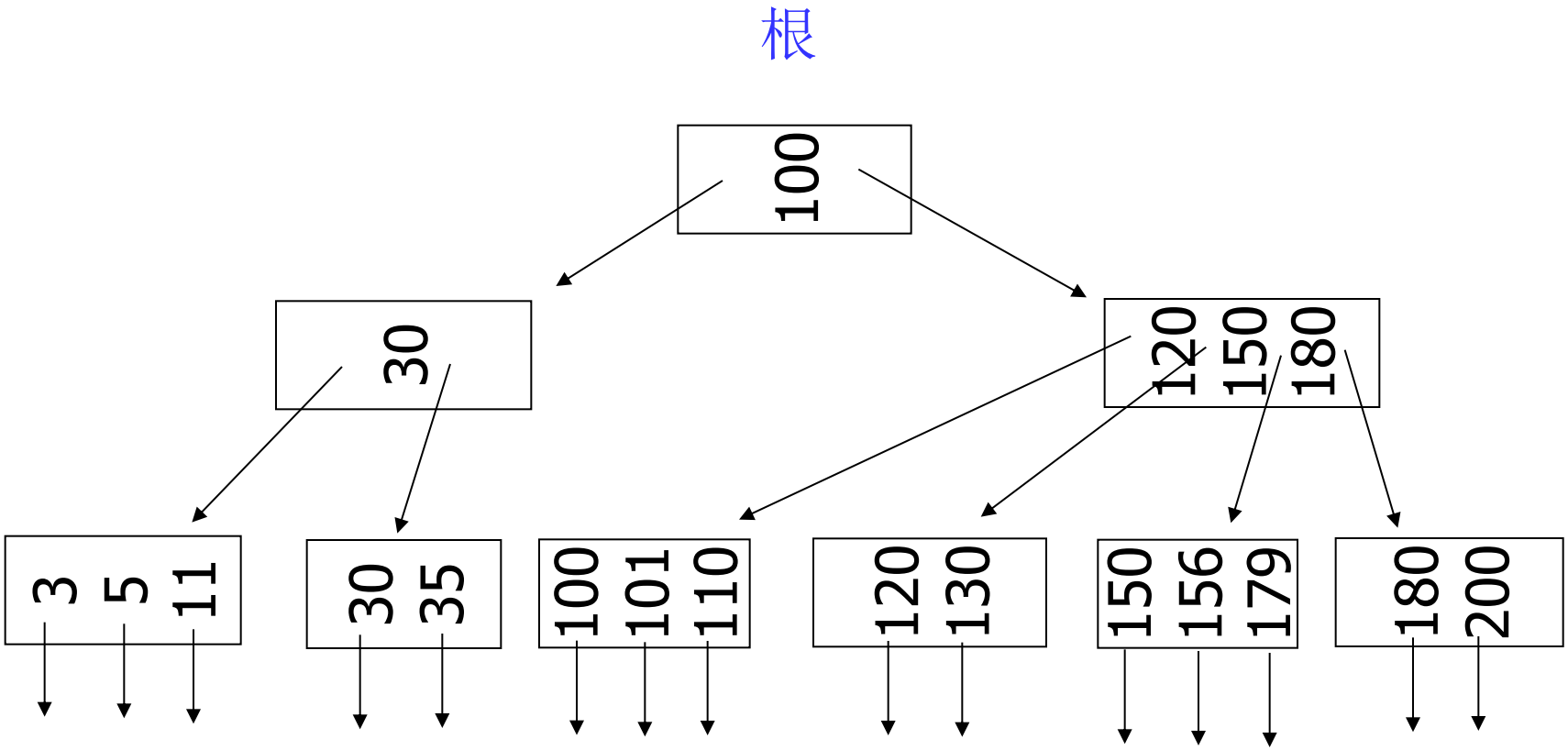
B 树
B+树

B树

（1）B树能自动保持与数据文件大小相适应的索引层次

（2）对使用的存储空间进行管理，使每个块的充满程度在半满与全满之间，不需要溢出块。

B+树例子　　n=3


根
100
30
120 150 180
3 5 11
30 35
100 101 110
120 130
150 156 179
180 200

B树——结构

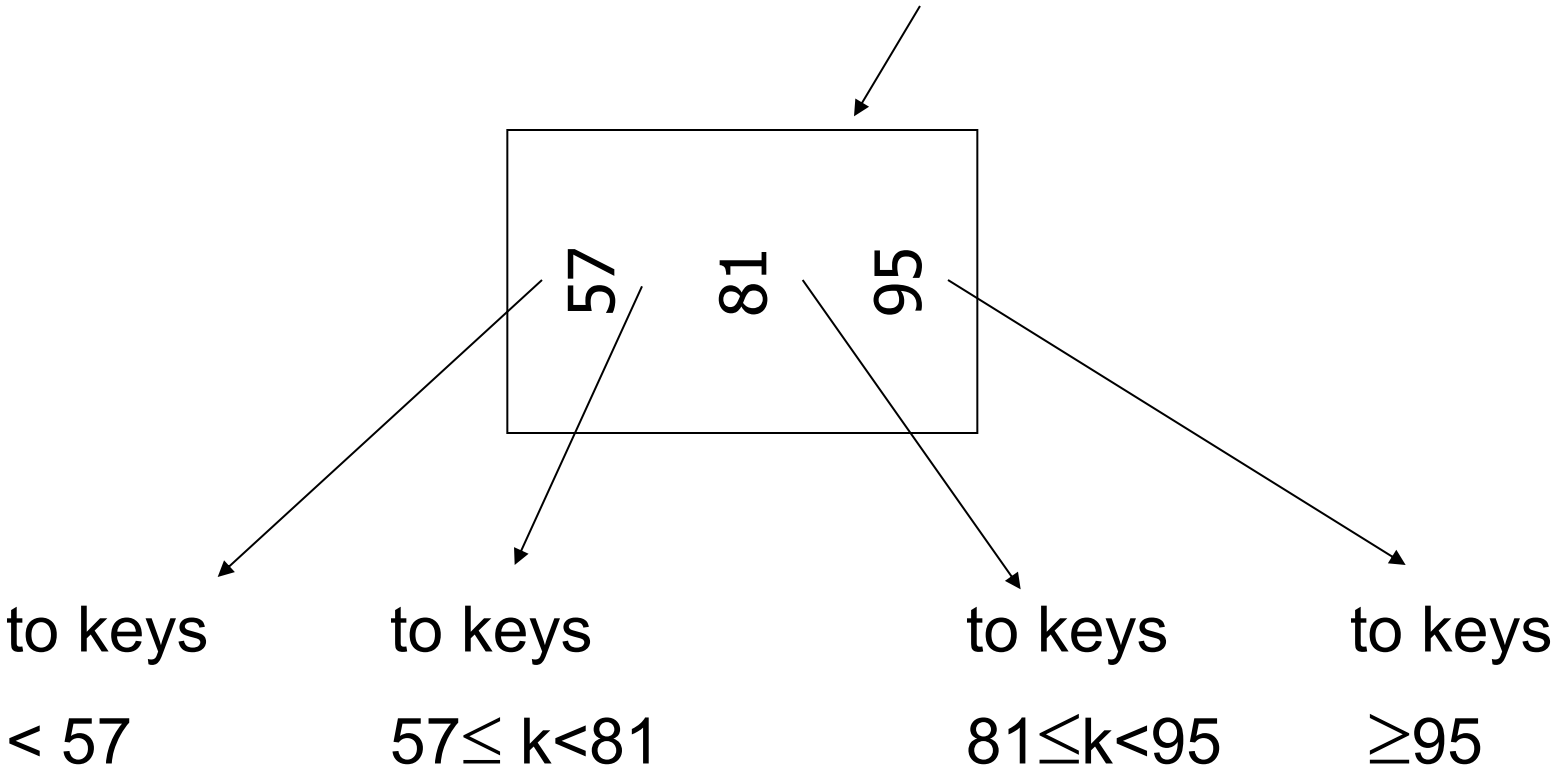
（1）B树是一颗平衡树，从树根到任一树叶的路径都一样长

（2）通常B树有三层：根、中间层和叶，但也可以有任意多层。

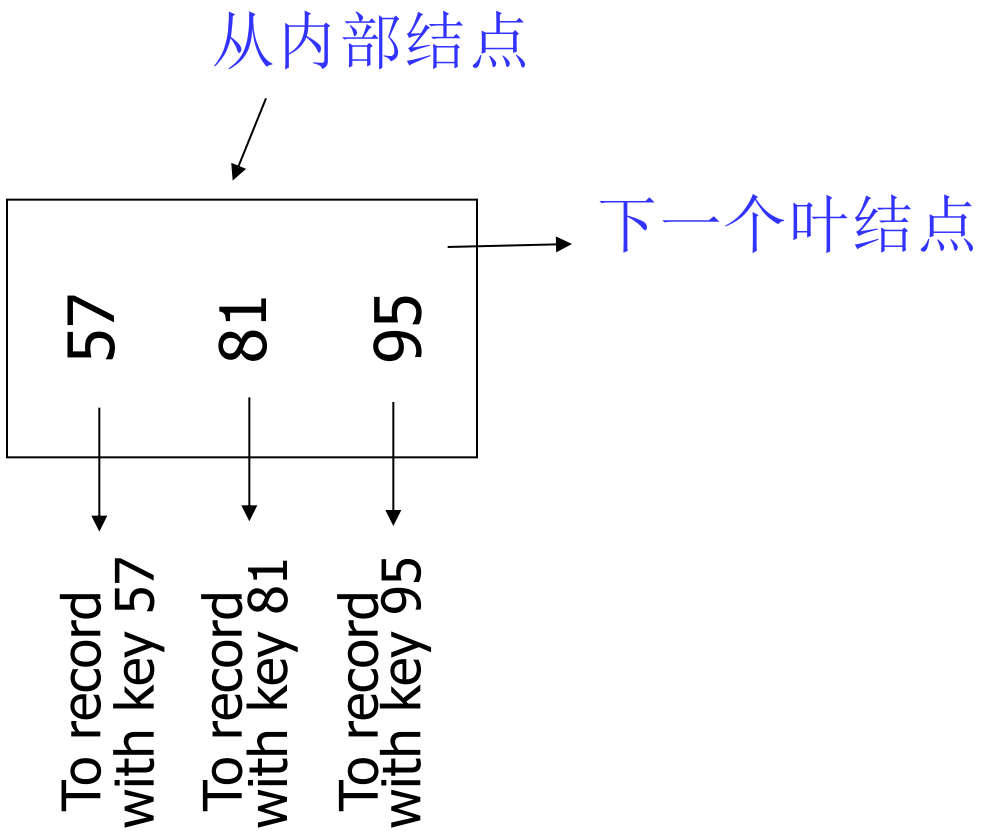
（3）每个B树都有一个关键的参数n,决定了B树中每个存储块的布局，每个存储块最多有n个键值和n+1个指针。

参见下图的B树叶结点、内部结点和全图。

内部结点示例(pp.113图4-22)



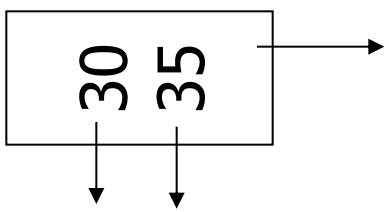
叶子结点示例(pp.112图4-21)：

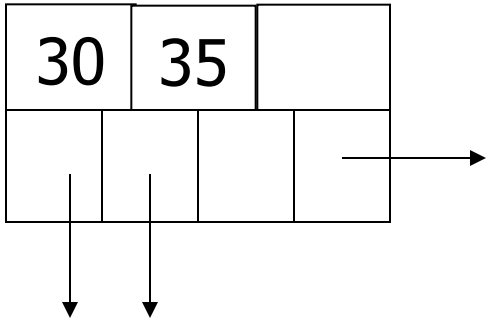


教材中标记说明

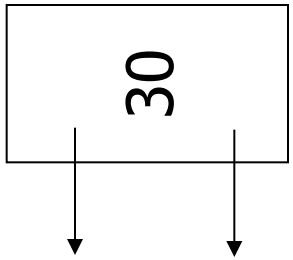
n=3

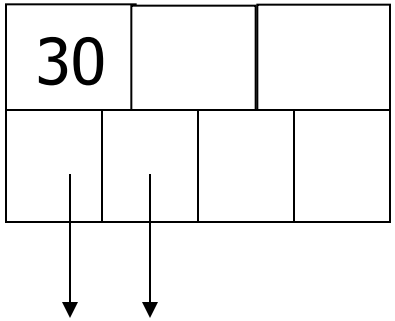
叶子结点:





内部结点（非叶子结点）：





结点的大小：

n+1 个指针

n 个键

(最大)

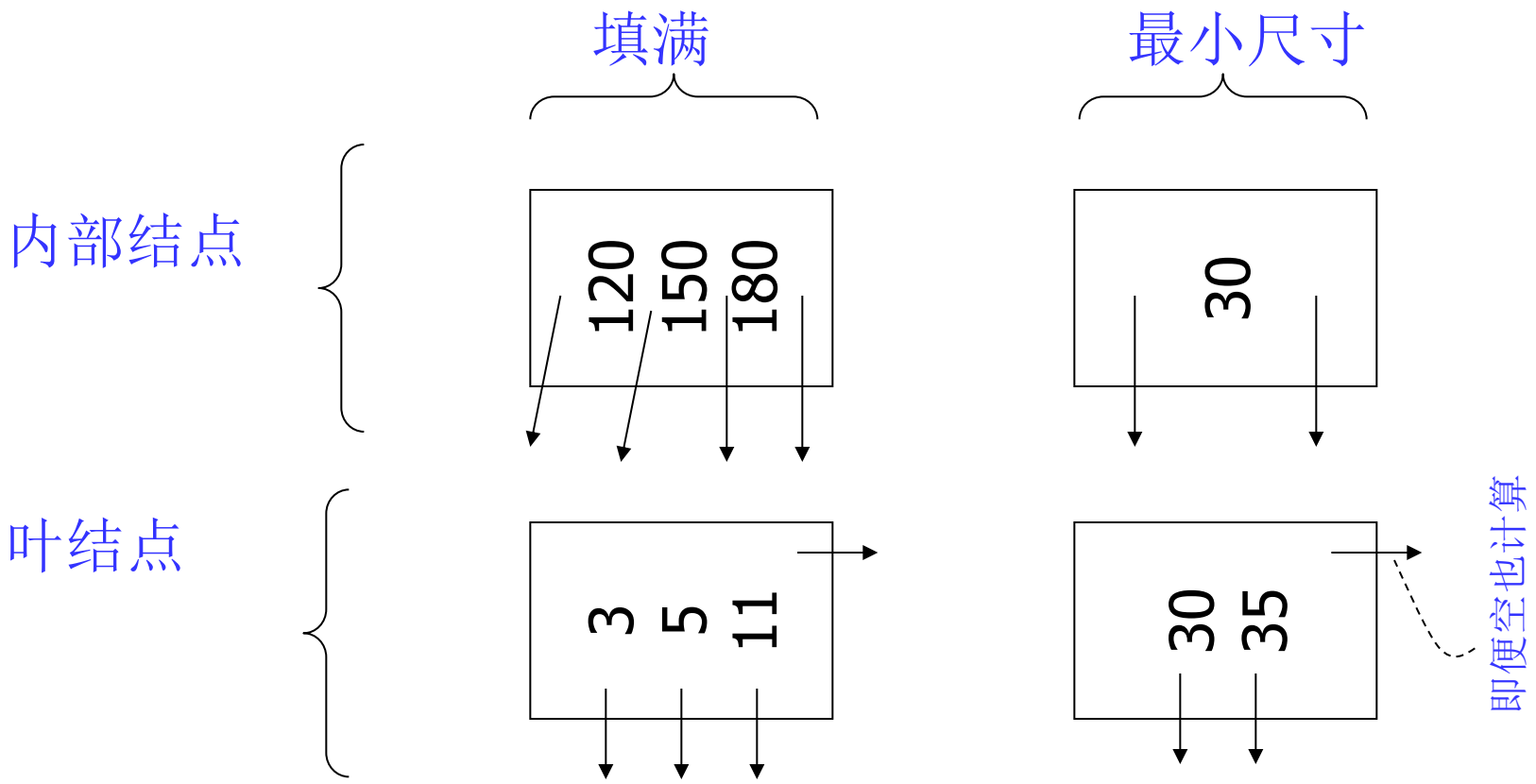
为了避免结点空间太浪费，要求：

- 每个结点最少包含

  内部结点: $\lceil (n+1)/2 \rceil$　　指针

  叶子结点: $\lfloor (n+1)/2 \rfloor$　　指针（指向数据的，不包括指向下一个叶子结点的那个指针）

n=3



B树——结构

关于存储块的限制

（1）根结点至少有两个指针（整个块只有一个键值和一个指针的平凡情况也可以）

（2）叶结点中，最后一个指针指向右边的下一个叶结点存储块；每个叶块至少包含⌊(n+1)/2⌋（下括号）个指向数据记录的指针。如果第i个指针被使用，则指向第i个键值记录；未使用的指针为空且不指向任何地方。

B树——结构

（3）在内部结点中，所有的n+1个指针都用来指向B树中下一层的块，其中至少有$\lceil (n+1)/2 \rceil$（上括号）个指针被使用。但如果是根结点，则不管n多大至少有两个指针被使用。如果有j个指针，则该块中有j-1个键值，记为$K_1,\ldots,K_{j-1}$，第一个指针指向键值小于K_1的记录部分，第二个指针指向键值大于等于K_1且小于K_2的部分,依此类推。最后，第j个指针指向键值大于等于的K_{j-1}部分。

（4）任何一个结点的键值都是按照从左到右的升序排列。

B树——结构

举例：

假定存储块大小是4096个字节，一个键占4个字节，一个指针占8个字节，由4*n+8*(n+1)≤4096，可知n最大为340。

秩为n的B树特征

(1) 所有的叶子结点在同一个最底层(平衡树)。

(2) 除最后一个指针，叶子结点中的其他指针都指向数据文件中的记录。

(3) 键和指针数目的规则

	Max ptrs	Max keys	Min ptrs→data	Min keys
内部结点(非根)	n+1	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
叶子结点(非根)	n+1	n	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
根结点	n+1	n	1	1

B树—应用

（1）B树的查找键是数据文件的主键，且索引是稠密的。对应的数据文件可以按主键排序，也可以不按照主键排序。

（2）如果数据文件按主键排序，则B树可以是稀疏索引，叶结点中为数据文件的每个块设一个键-指针对。

（3）数据文件按照非主键属性排列，且该属性是B树的查找键。B树的叶结点为数据文件中的每一个属性值K设一个键-指针对，指向排序键值为K的记录中的第一个（仍是稠密索引）。

B树——应用

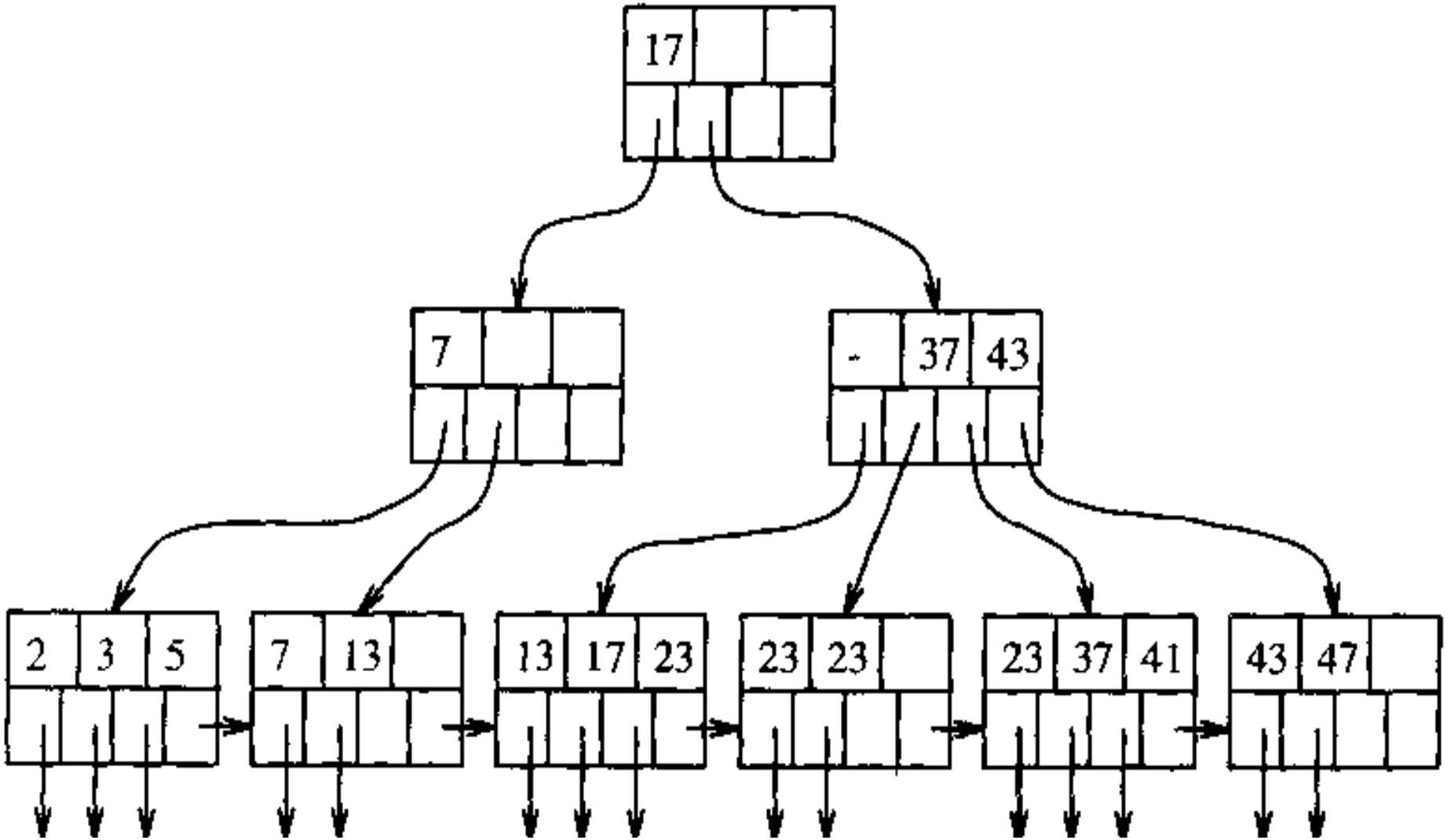
允许查找键的重复出现，内部结点中键的涵义更改如下：

假定一个内部结点的键是$K_1,K_2,...,K_n$,那么K_i将是从第i+1（K_i右侧）个指针所能访问的子树中出现的最小新键。这里的“新”，是指在树中第i+1个指针所指向的子树以左没有出现过的K_i ,但K_i在第i+1个指针指向的子树中至少出现一次。

注意：在某些情况下可能不存在这样的键，这时K_i可以为空，但对应的指针仍然需要。

B树——应用

如图：叶子节点两个13，四个23，引起根结点和根结点的第二个子结点的一系列变化。



B树——键值查找

（假定叶结点中没有重复键，且B-树是稠密索引）。

从根到叶递归查找键值为K的记录：

（基础）若处于叶子结点，就在其键值中查找，若第i个键是K，则根据第i个指针找到记录。

（归纳）若处于某个内部结点，且它的键为$K_1,K_2,\ldots,K_n$。若$K<K_1$,则为第一个子结点；如果$K_1 \le K<K_2$,则为第二个子结点，依此类推。在这个子节点上递归地运用查找过程。

B树——范围查询

在B树的叶子结点上查找在[a,b]之间的所有键值

（1）首先通过一次查找，寻找键a。不论它是否存在，都可以到达可能出现a的叶子结点，然后在该结点查找键a或者大于a的那些键

（2）如果没有发现大于b的键，就使用当前叶子结点指向下一个叶子结点的指针，并继续检索查找键和相应的指针，直到

1）找到一个大于b的键，这时停止查找；或者

2）到了叶结点的末尾，在这种情况下，转向下一个叶子结点并且重复这个过程

B+树插入

(a) 叶子结点空余

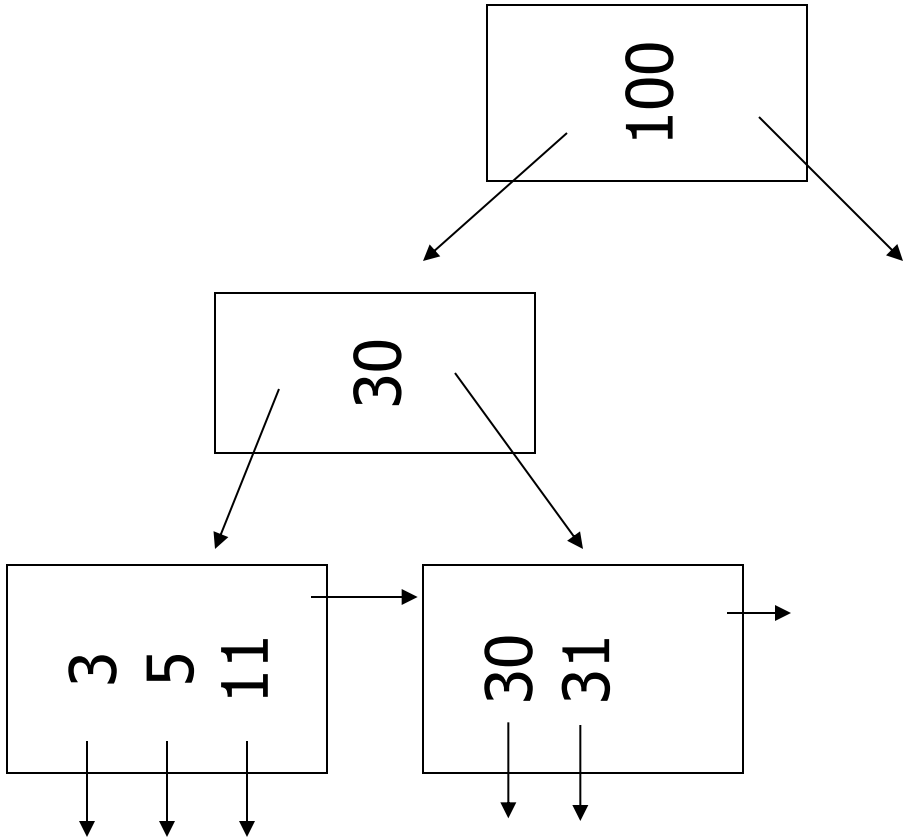
(b) 叶子结点溢出

(c) 内部结点溢出

(d) 创建新的根结点

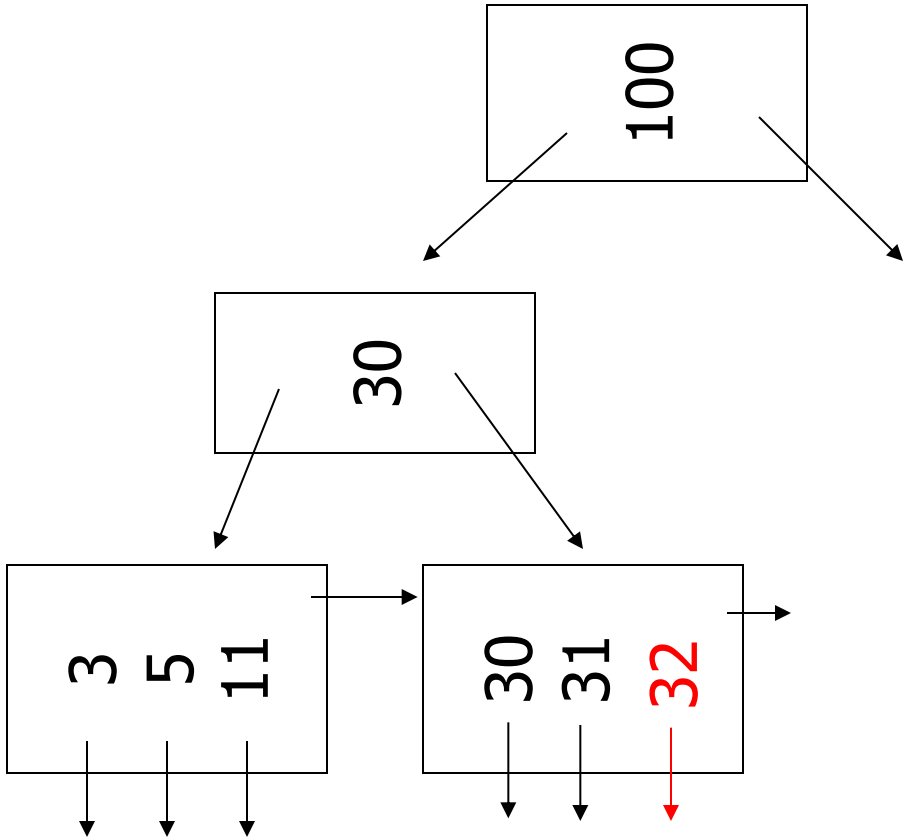
(a) 插入键 key = 32

n=3



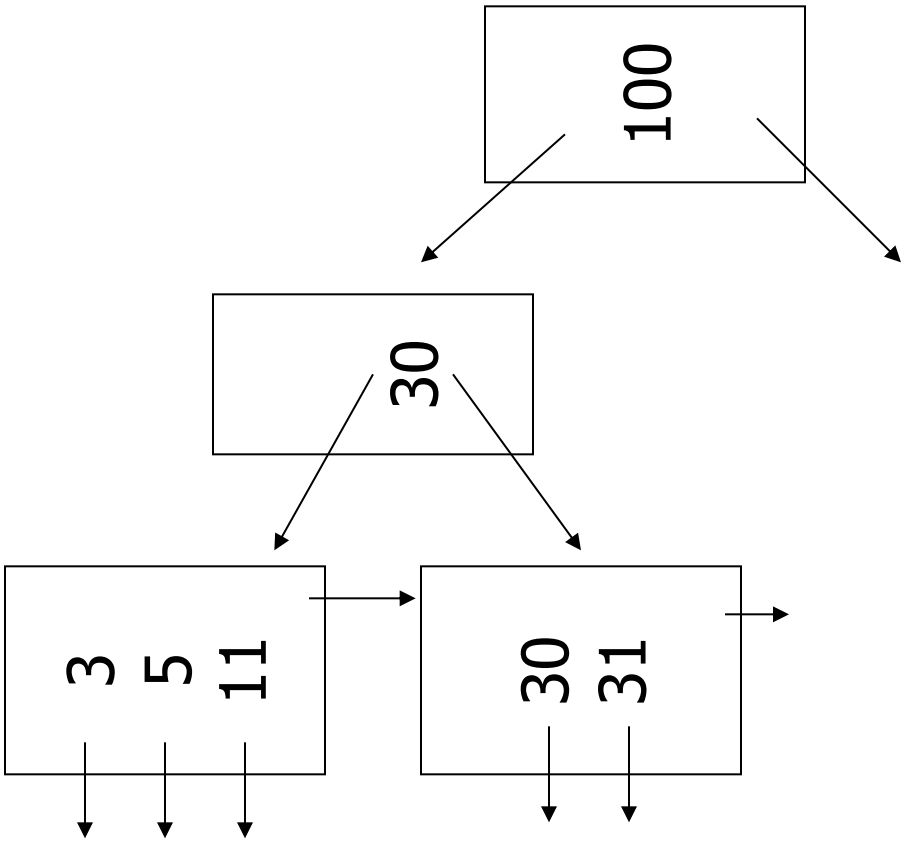
(a) 插入键 key = 32

n=3



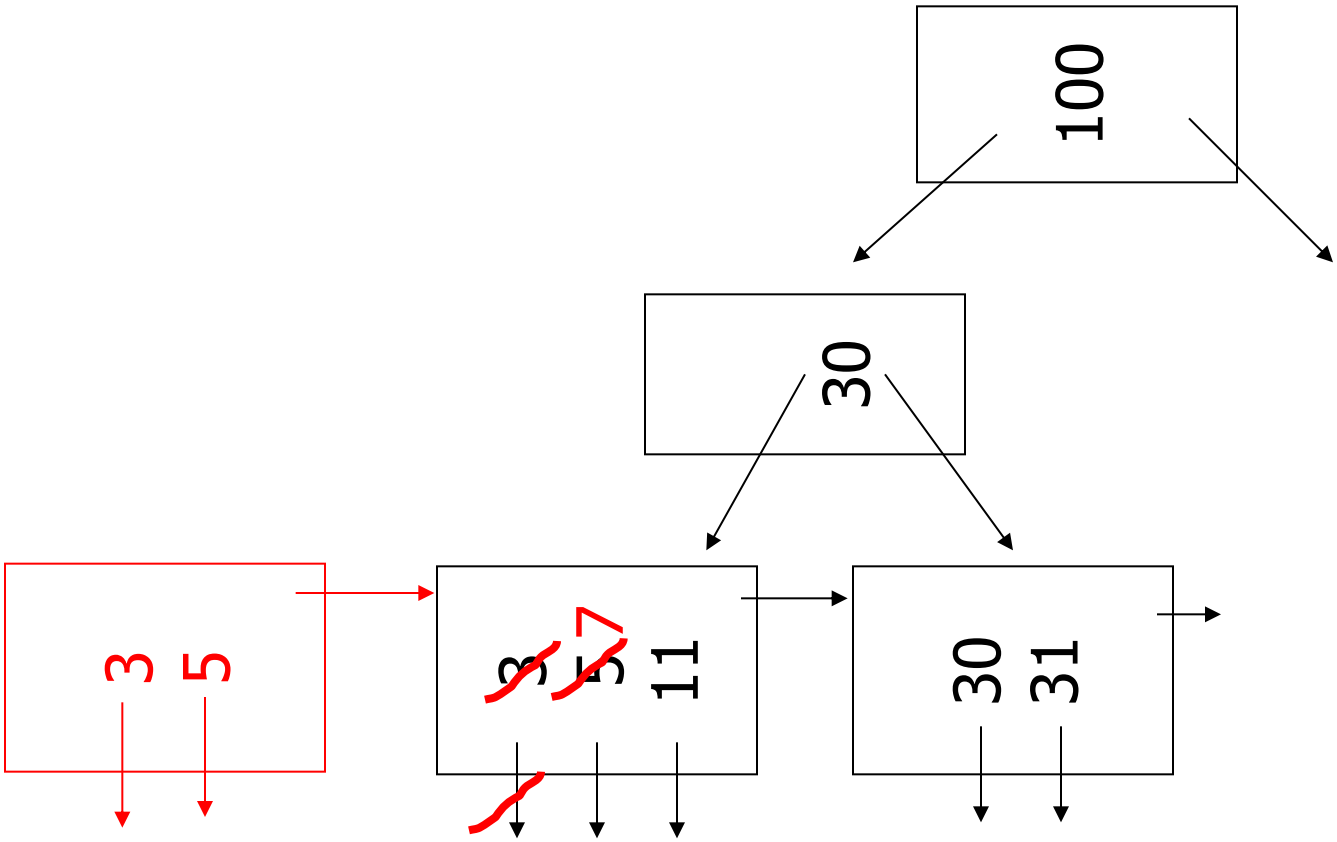
(b) 插入键 key = 7

n=3



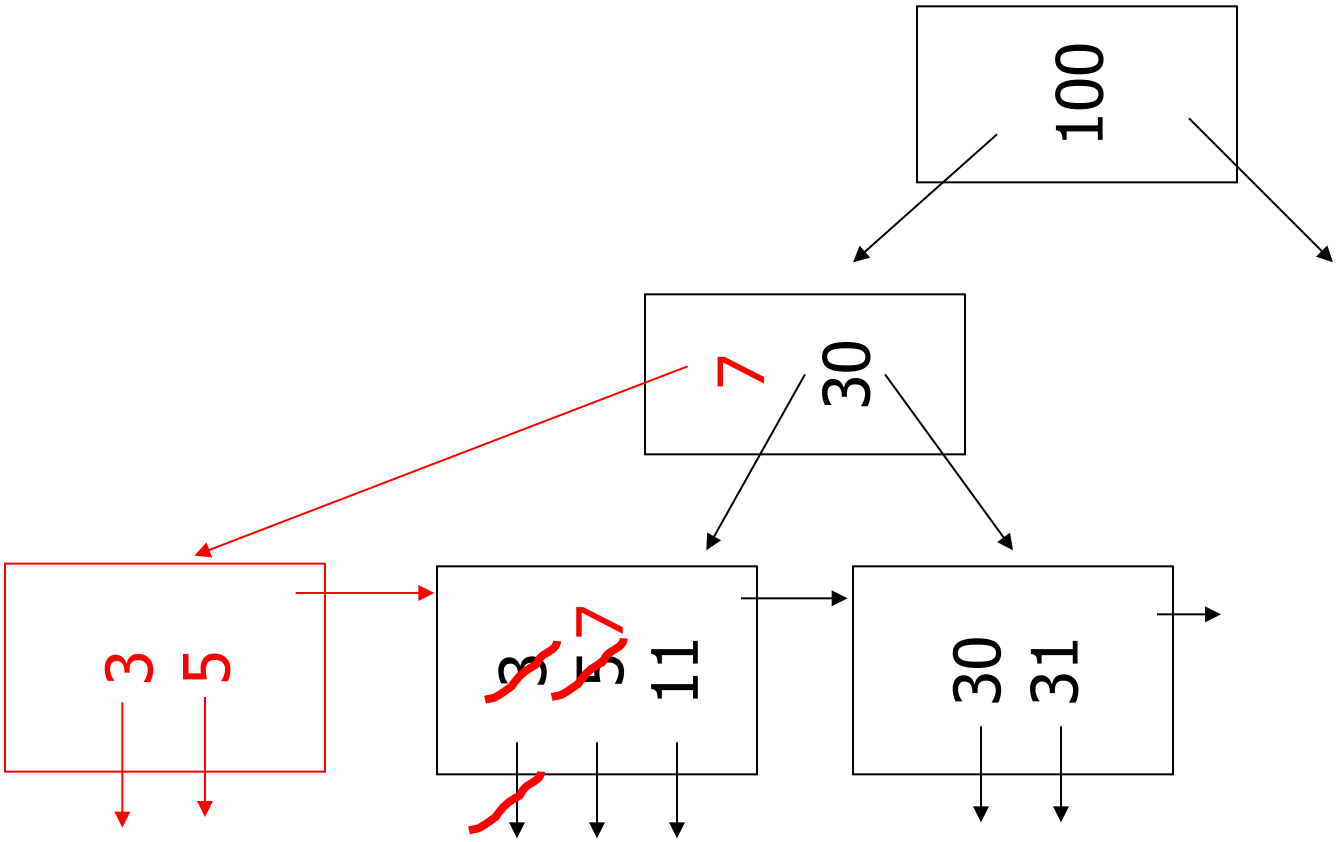
(b) 插入键 key = 7

n=3



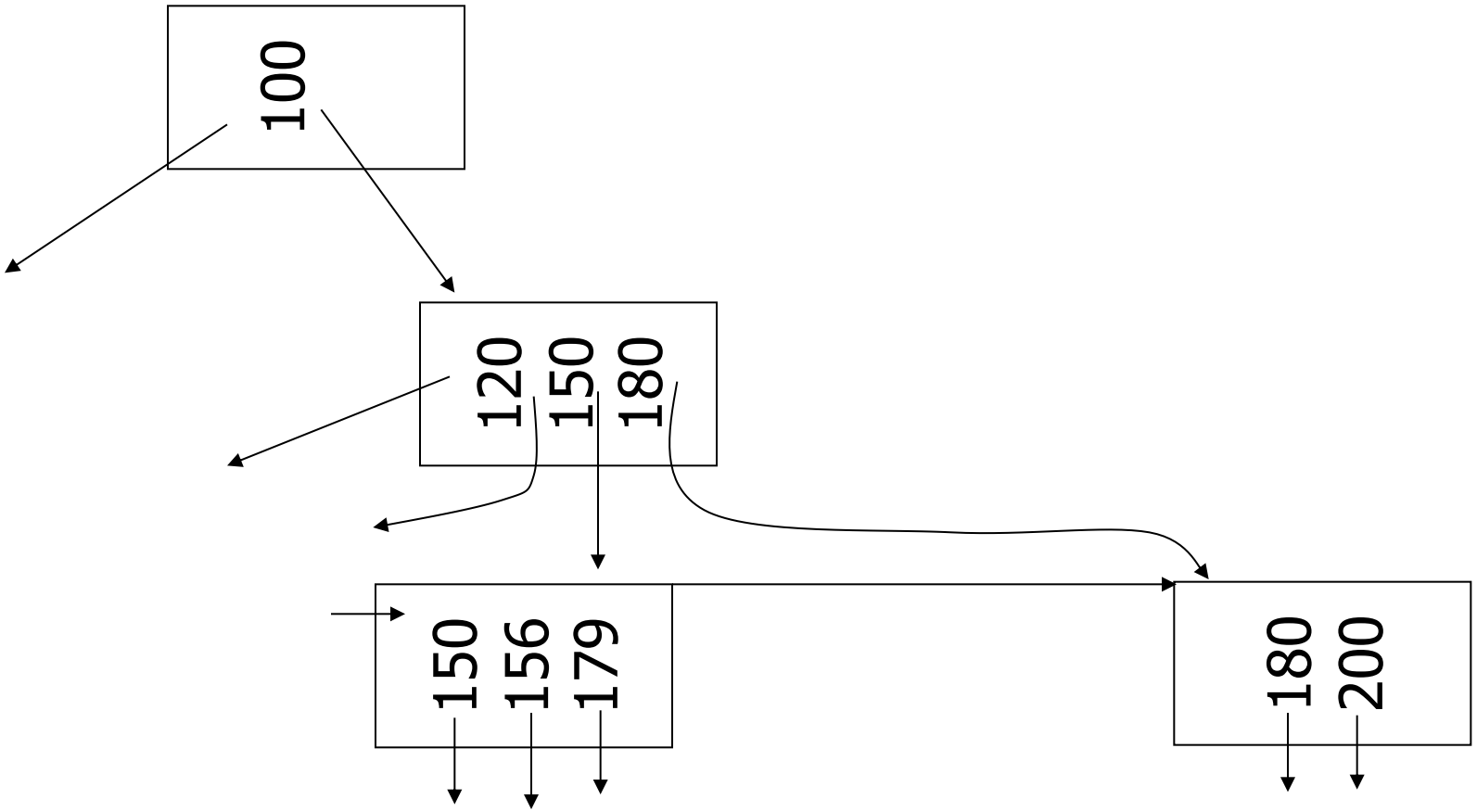
(b) 插入键 key = 7

n=3



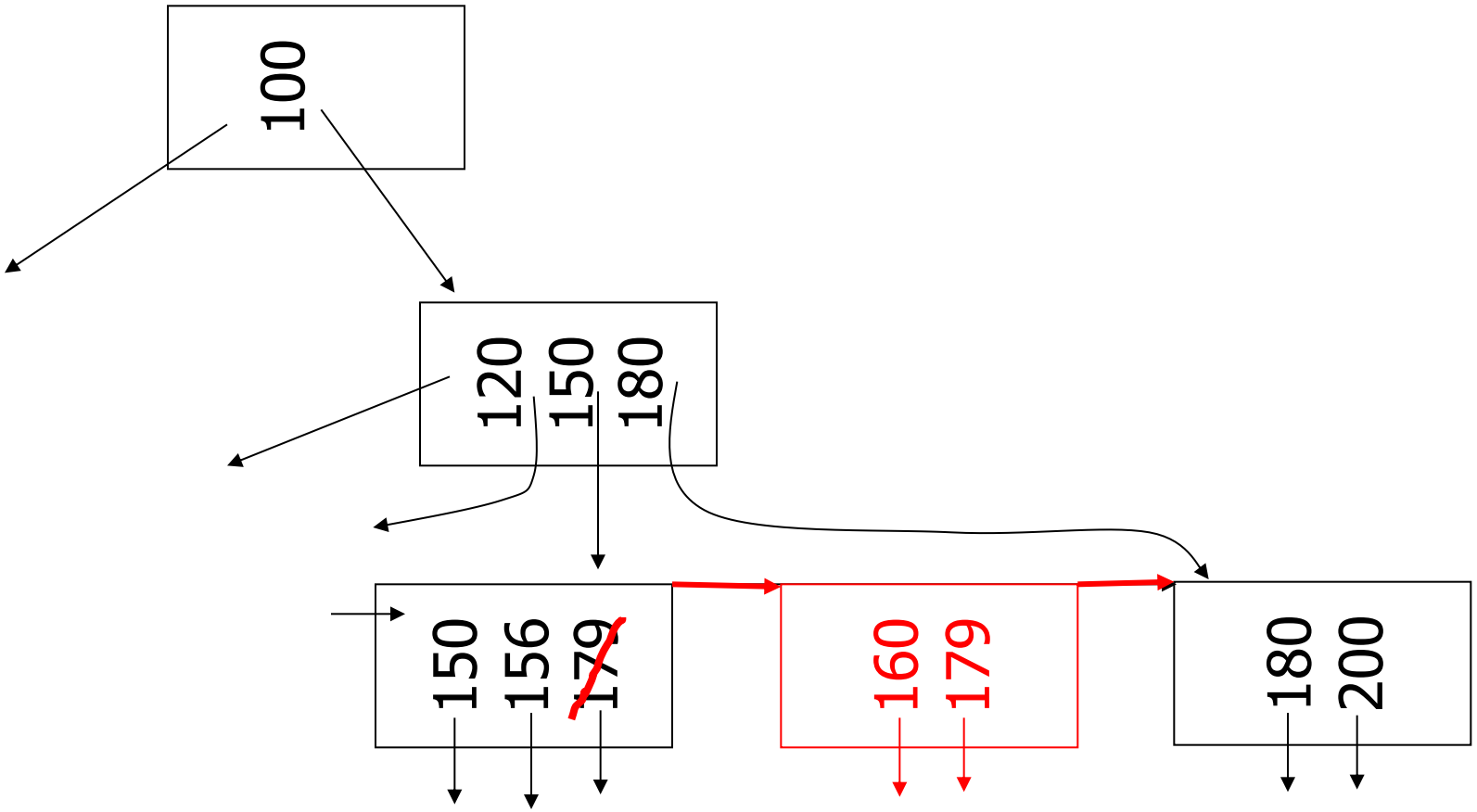
(c) 插入键 key = 160

n=3



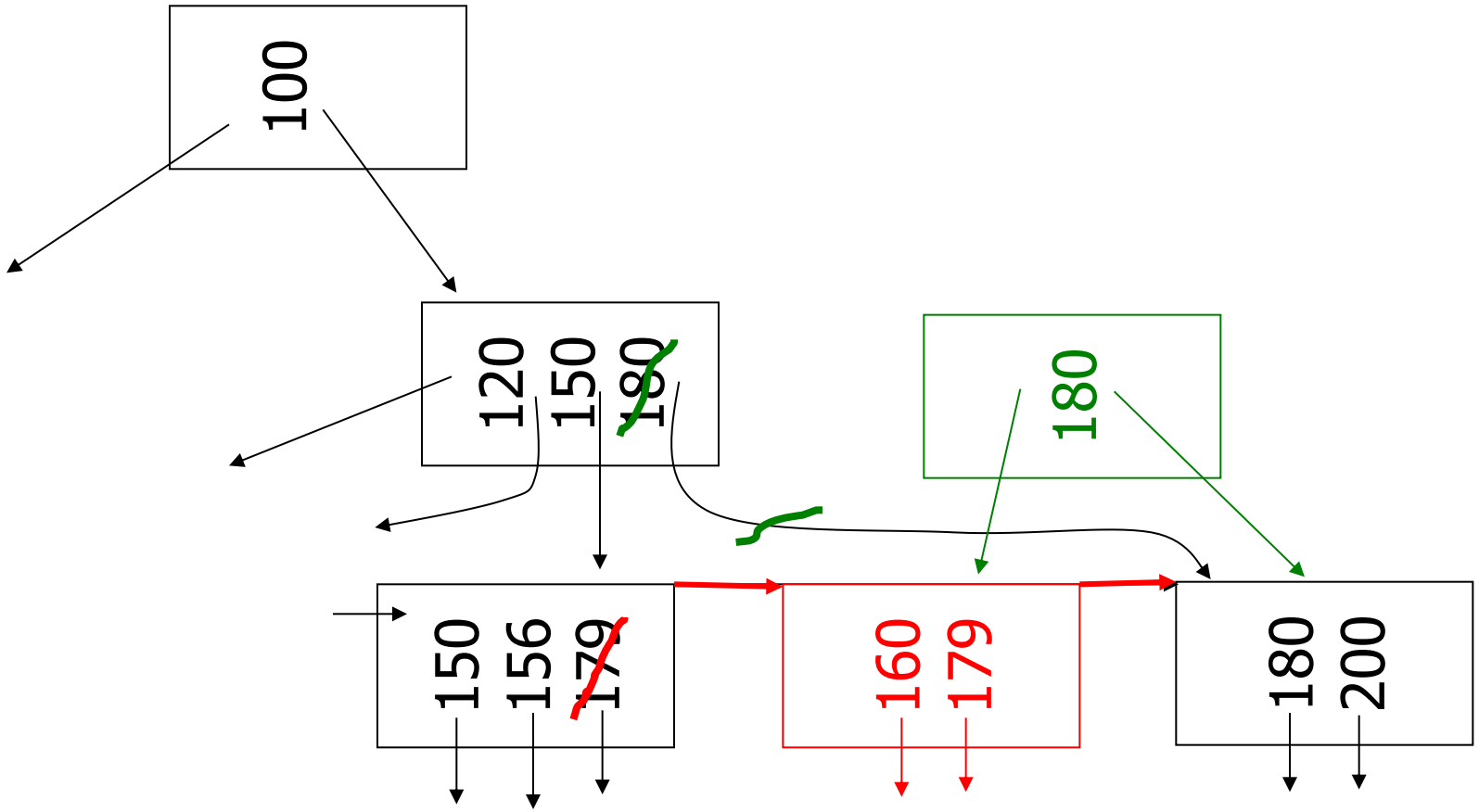
(c) 插入键 key = 160

n=3



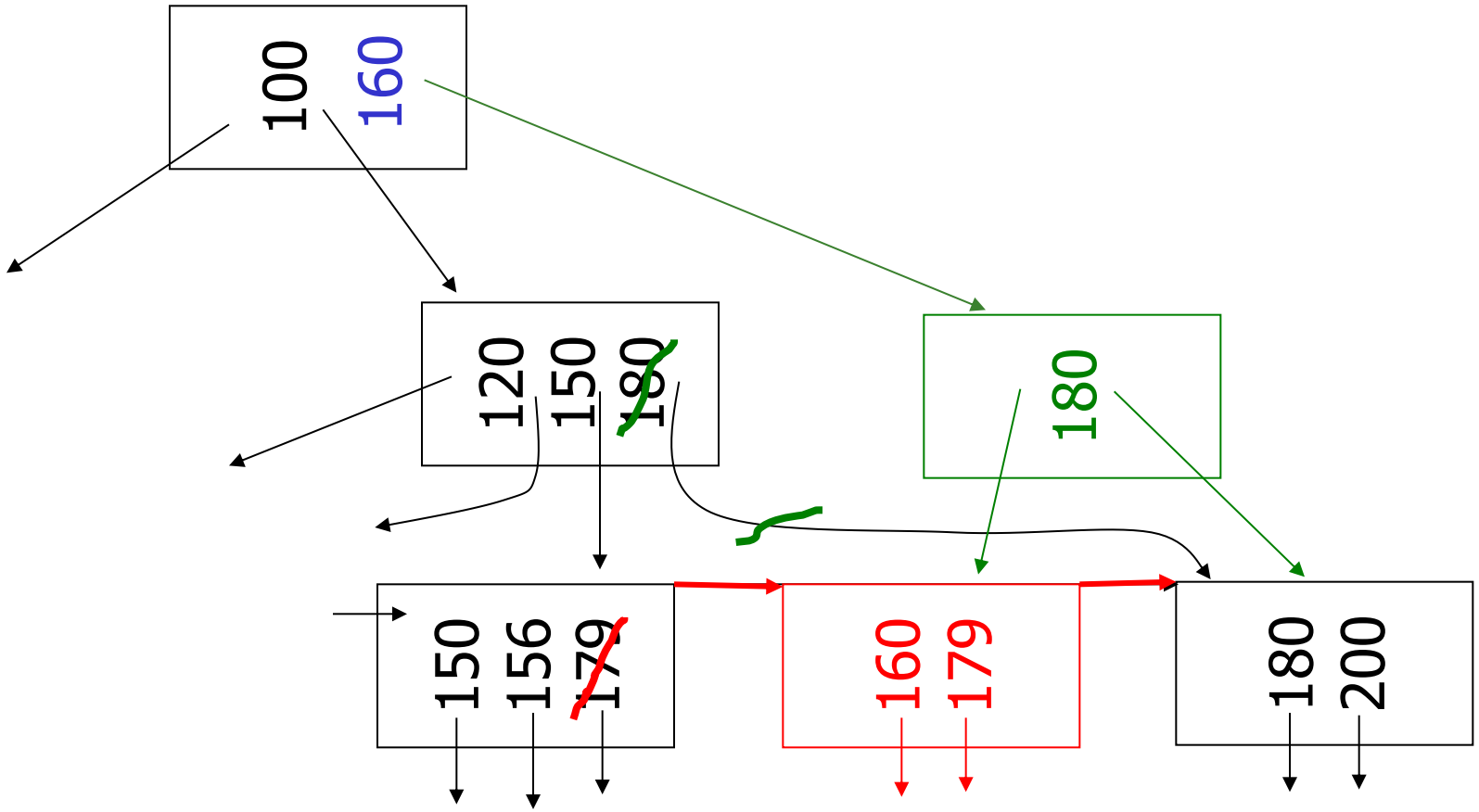
(c) 插入键 key = 160

n=3



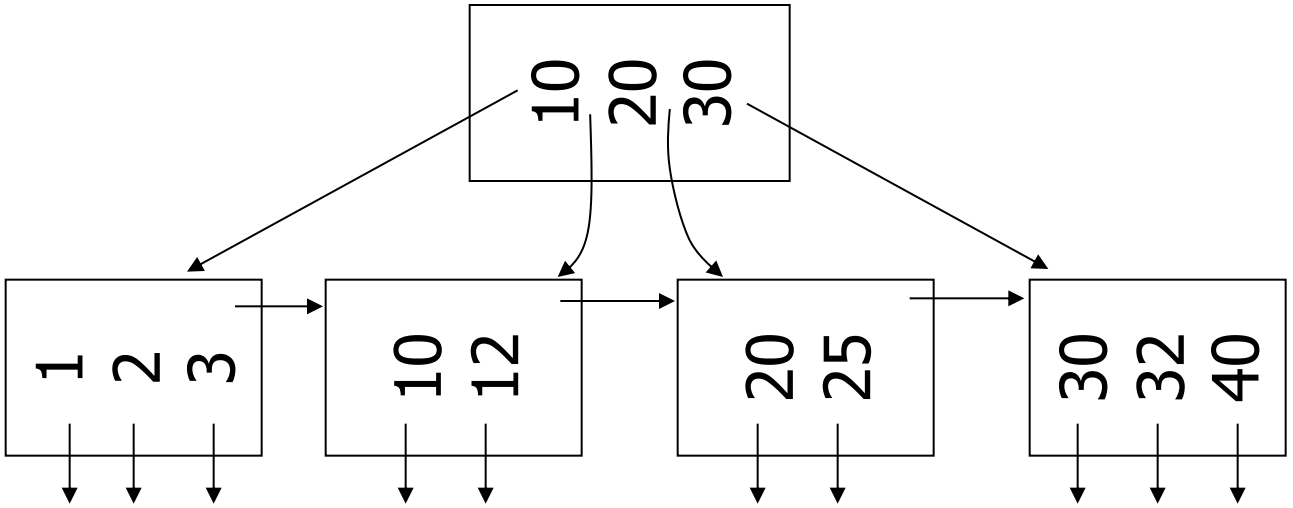
(c) 插入键 key = 160

n=3



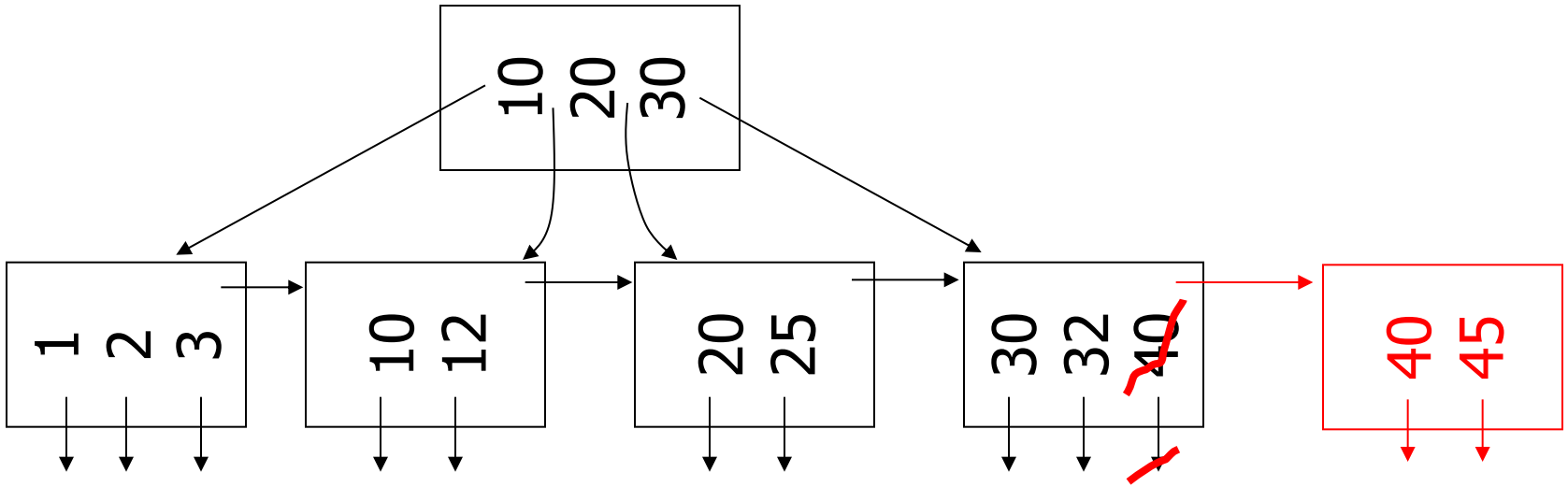
(d) 新建根，插入 45

n=3



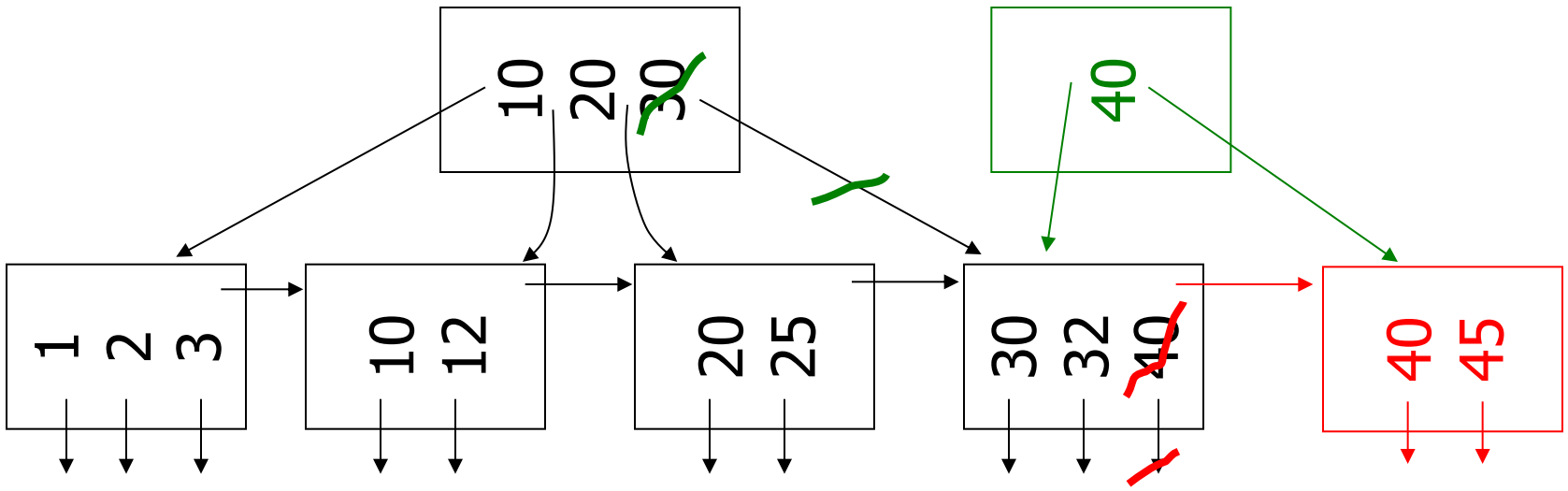
(d) 新建根，插入 45

n=3



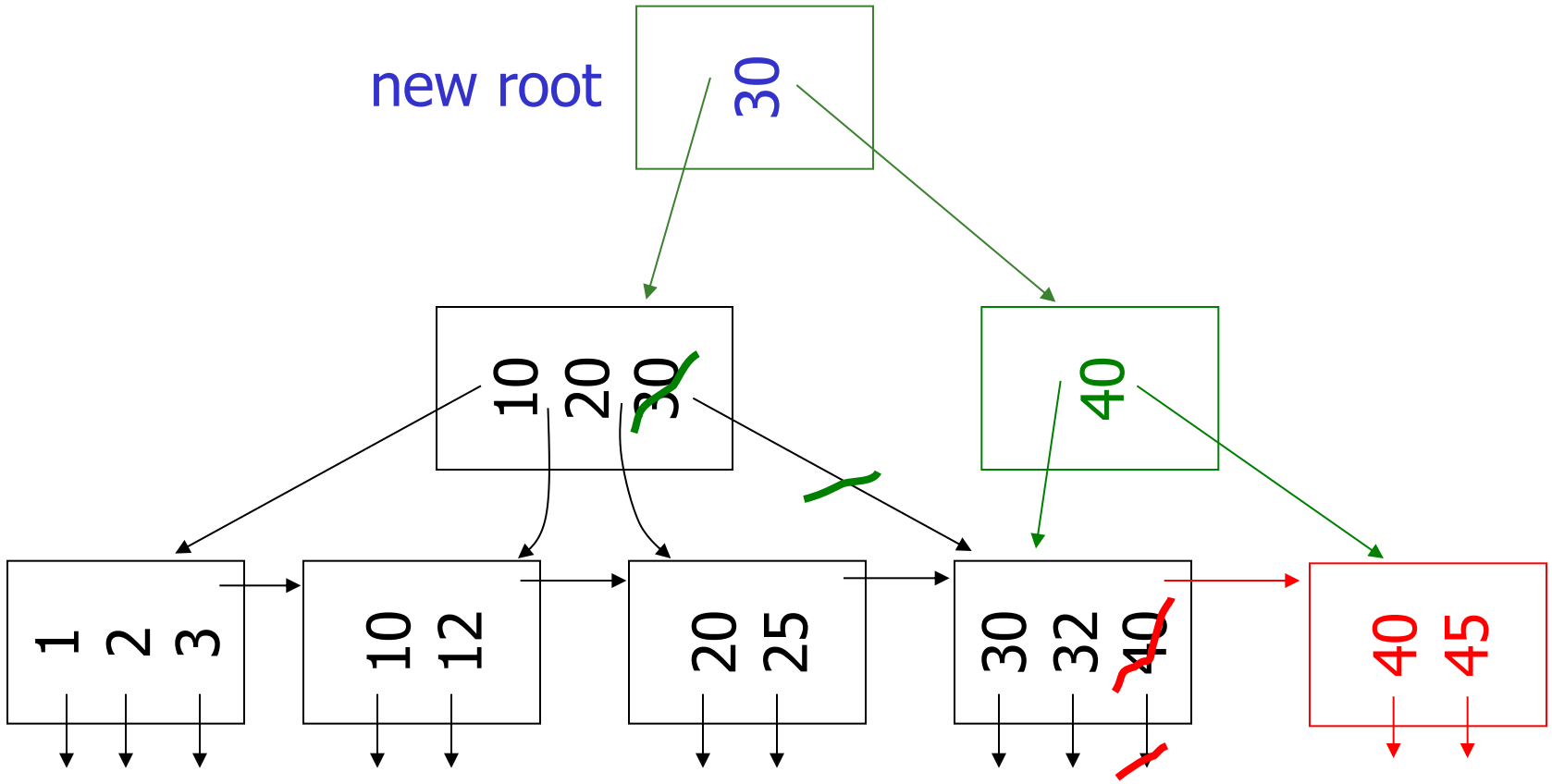
(d) 新建根，插入 45

n=3



(d)新建根，插入 45

n=3



B树——插入（总则）

原则上是递归的

（1）设法在叶结点找到空闲空间，如果有空间，就把键插入。

（2）如果空间不够，就把叶子结点分裂成两个，使每个结点有一半或者刚好超过一半的键(左侧多，右侧少)。

（3）某一层结点的分裂在上层看来，相当于在较高的层次上插入一个新的键-指针对：如果有空间，就插入；如果没有，则分裂这个父结点且向高层推进

（4）例外：如果试图插入键到根结点并且根结点没有空间，就分裂根结点成两个结点，并且在上层创建一个新的根结点。

B树——插入（叶结点）

假定N是一个容量为n的*叶结点*，且我们试图插入第(n+1)个键和它相应的指针，如下分裂

（1）创建一个新结点M，紧挨在N的右边。

（2）按照键的排列顺序，前⌈(n+1)/2⌉（上括号）个键-指针对保留在结点N中，而其他的键-指针对移到M中（左侧结点多，右侧结点少）

B树——插入（内部结点）

假定N是一个容量为n个键和n+1个指针的内部结点，并且由于下层结点的分裂。N正好又被分配给第（n+2)个指针，则执行下列步骤

（1）创建一个新结点M，作为N的兄弟，在N的右边

（2）前⌈n/2 ⌉(上括号)个键保留在结点N中，而后⌊n/2⌋(下括号)个键移到M中。注意：在中间的那个键总是被留出来，它既不在结点N中也不在结点M中。这一留出的键K会被N和M的父结点用来划分在两个结点之间的查找。

B+树删除（先借后合并原则）

(a) 不用合并 （不举例）

(b) 重新分配键（先借）

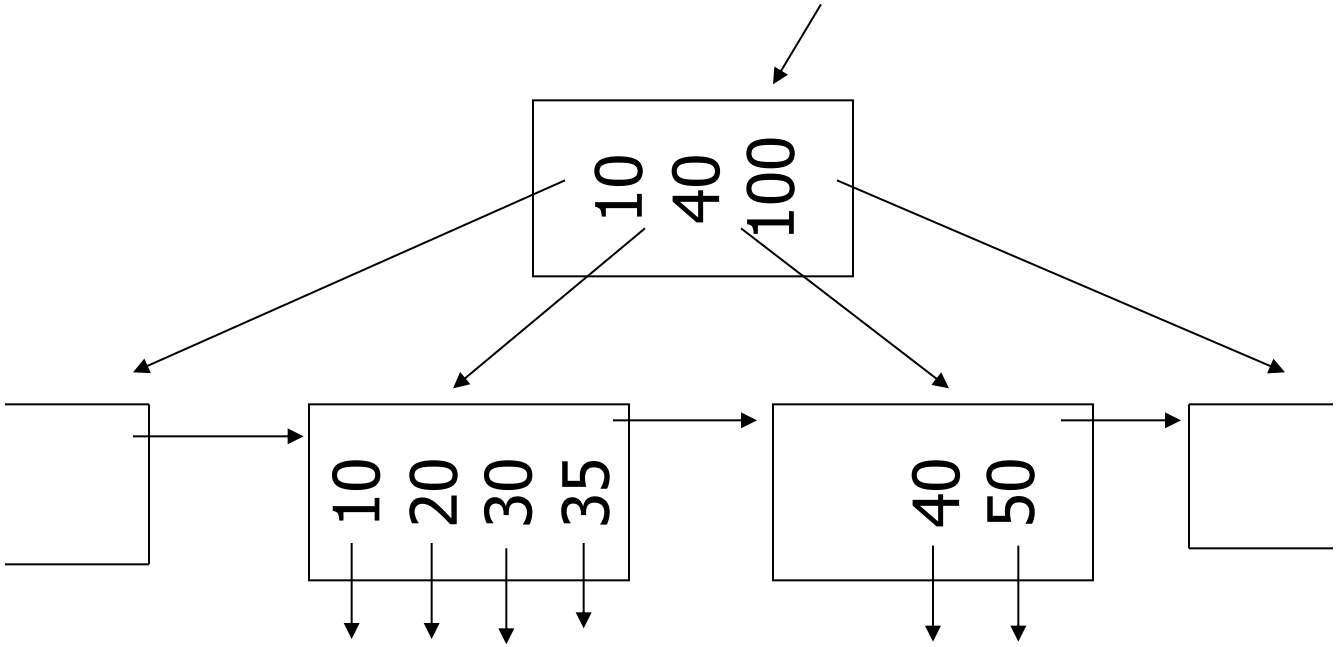
(c) 和兄弟结点直接合并（后合并）

(d) 内部结点中 (b) 或 (c) 的情况

(b) 重新分布键(先借)

n=4

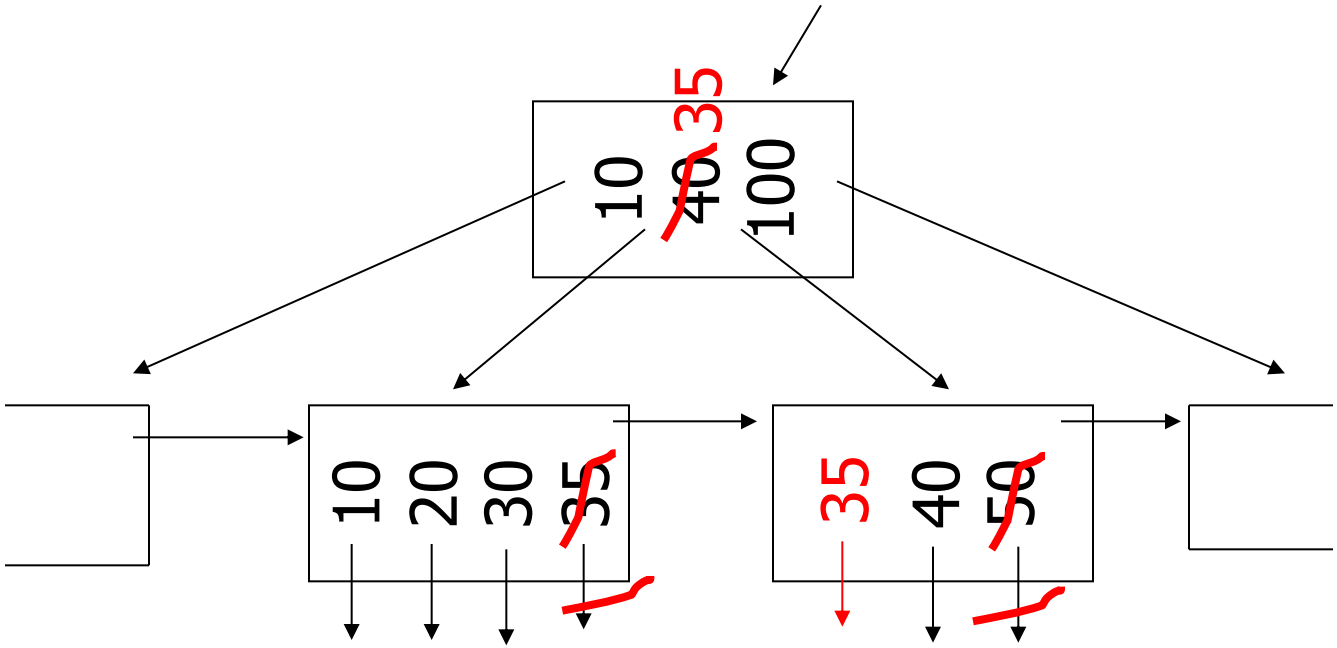
- 删除 50



(b) 重新分布键(先借)

- 删除 50

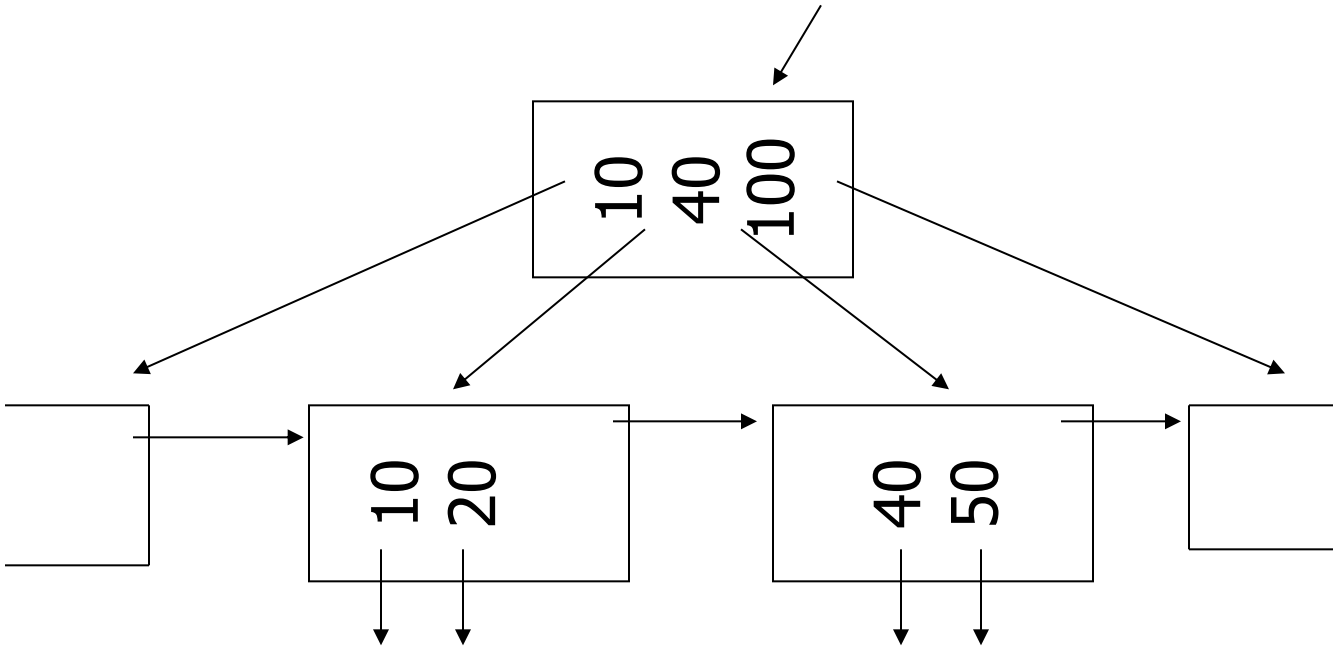
n=4



(c) 和兄弟结点合并(后合并)

n=4

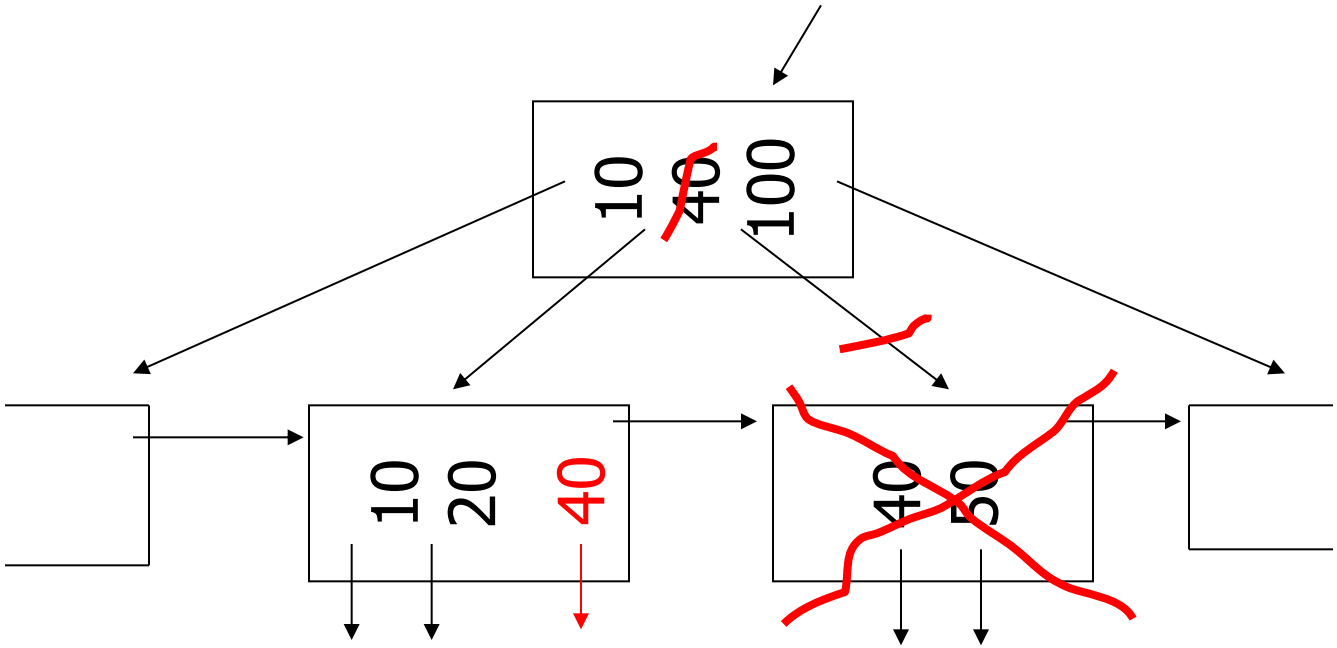
- 删除 50



(c) 和兄弟合并(后合并)

- 删除 50

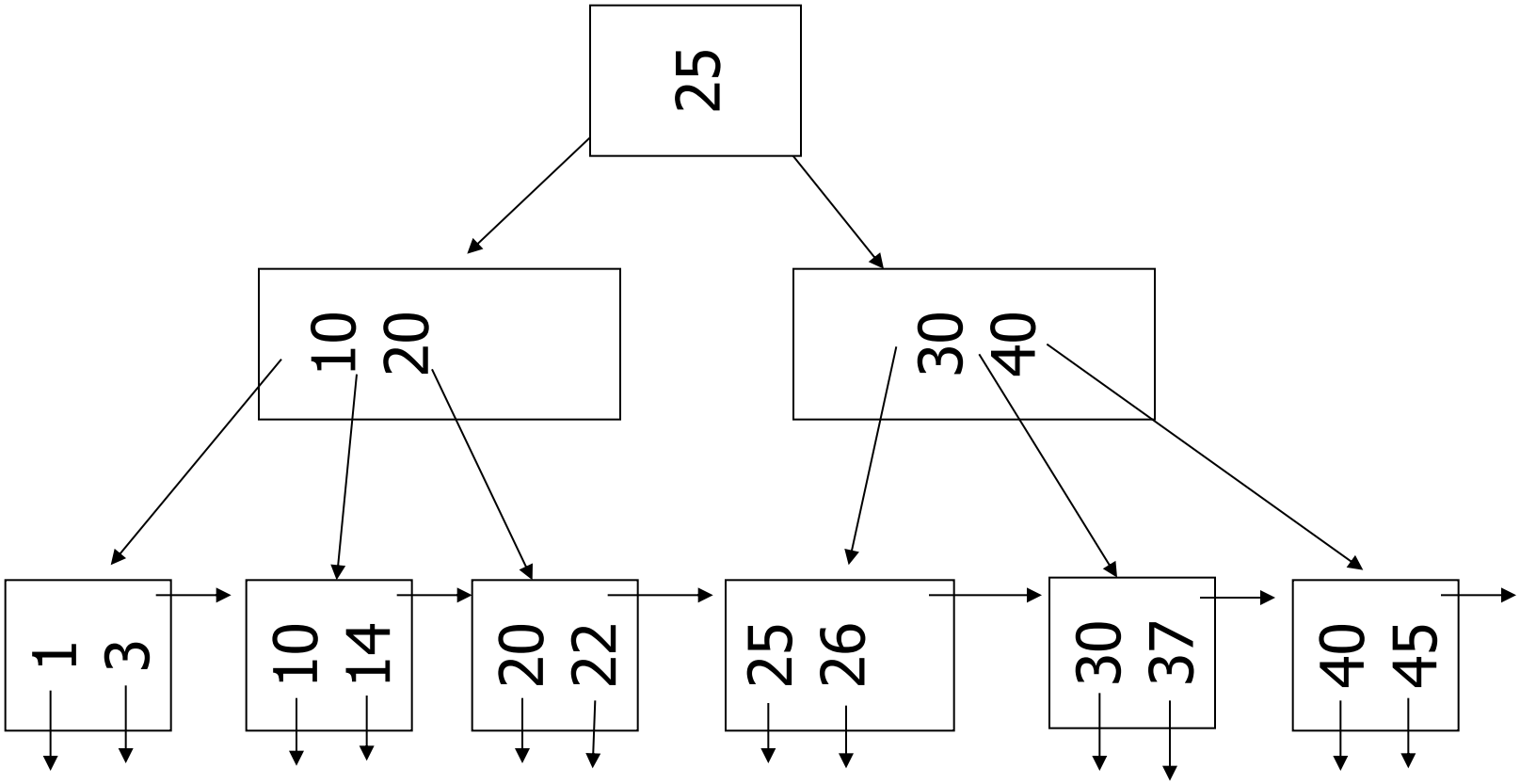
n=4



(d) 内部结点合并

– 删除 37

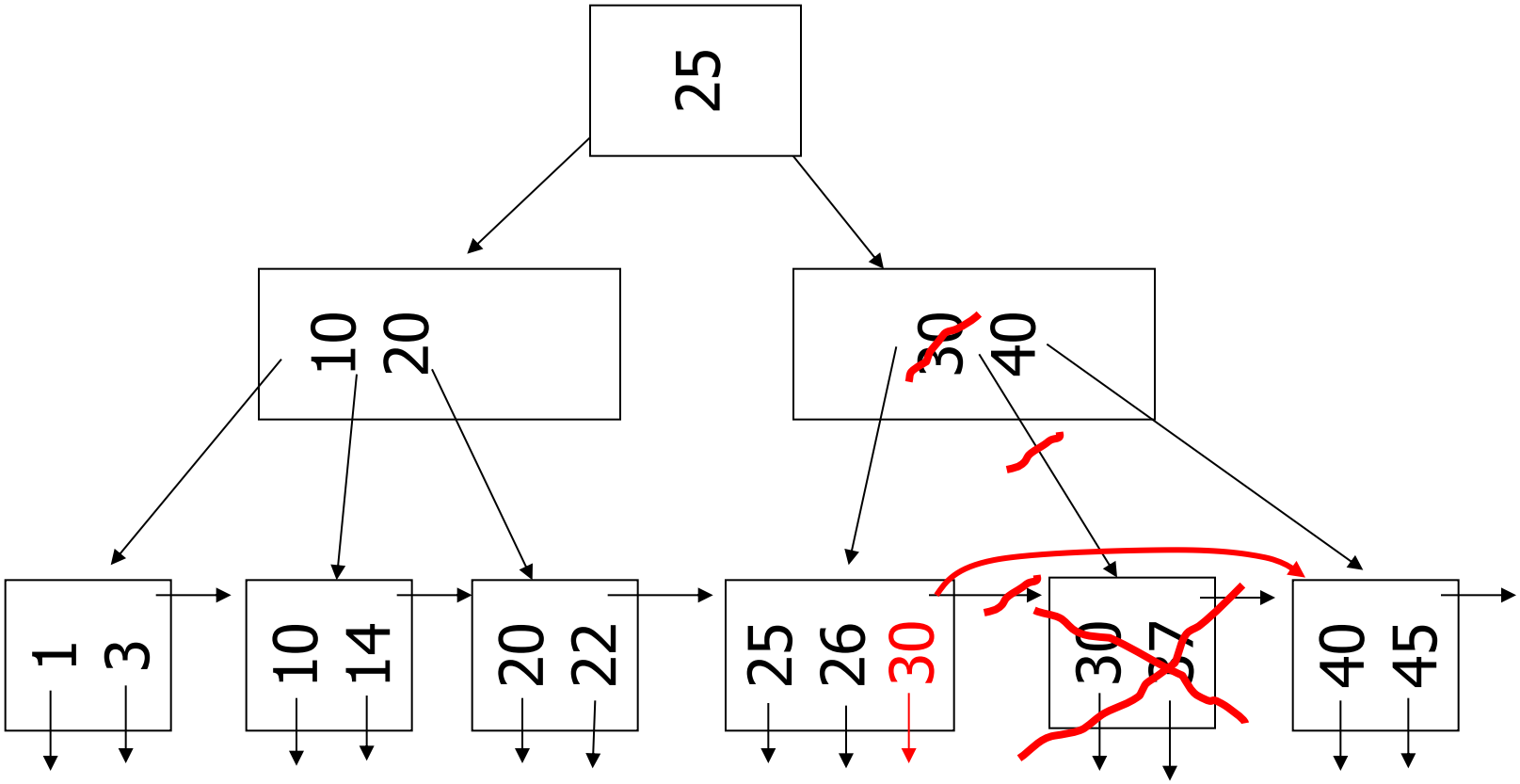
n=4



(d) 非叶子结点合并

– 删除 37

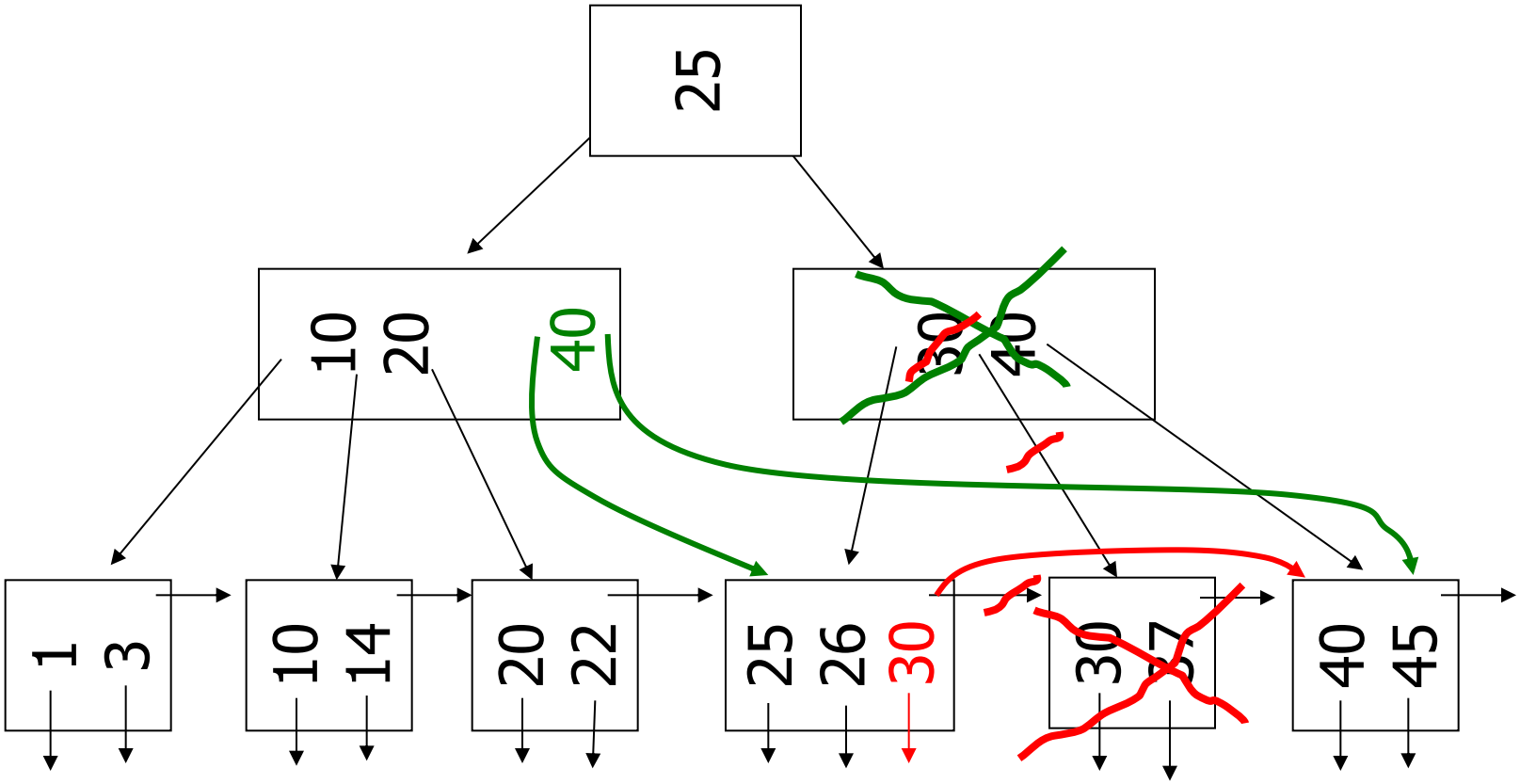
n=4



(d) 非叶子结点合并

– 删除 37

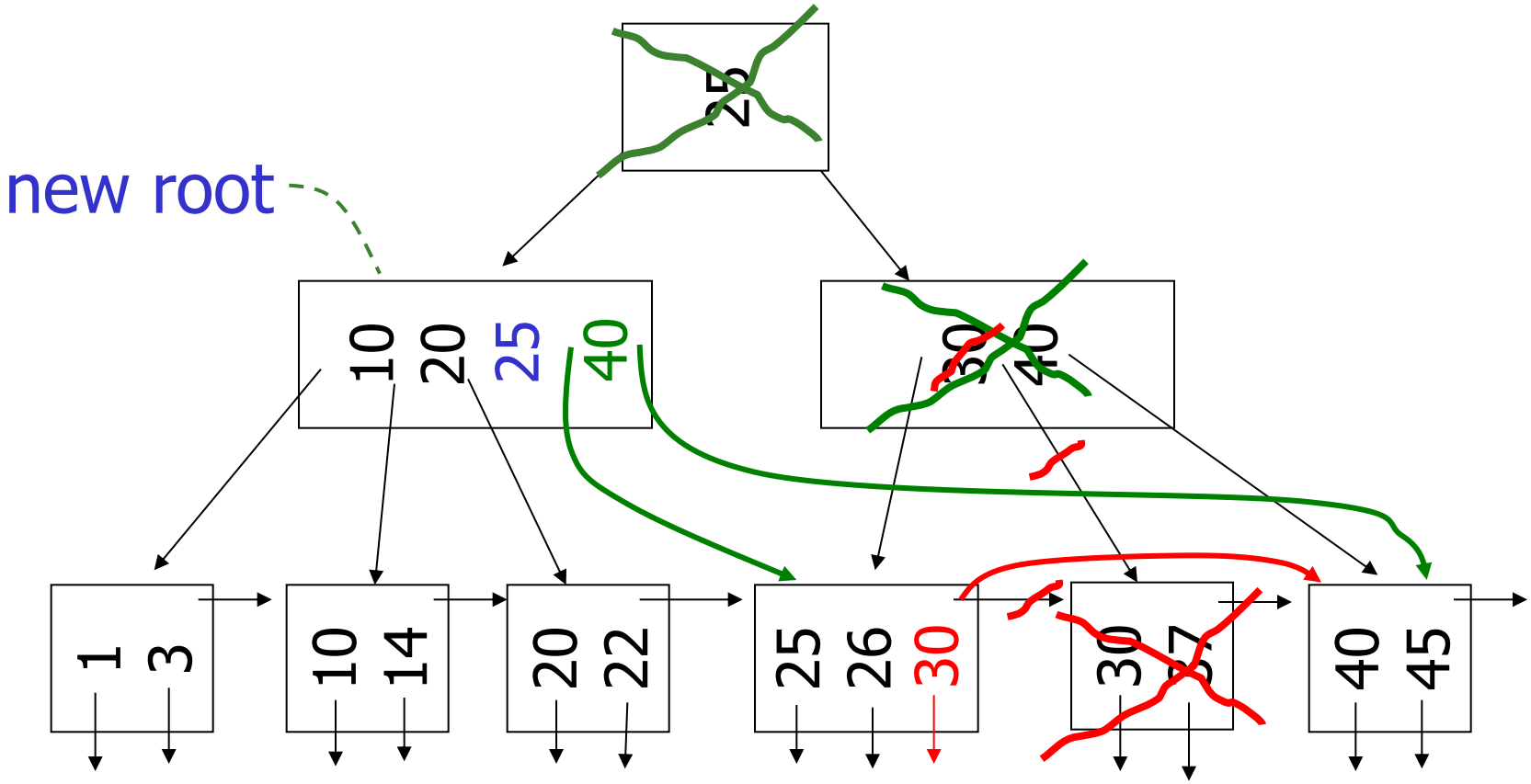
n=4



(d) 非叶子结点合并

– 删除 37

n=4



B树——删除

如果删除前具有最小的充满度，则删除后违反键数目的约束，此时要做如下处理（先借后合并）：

（1）如果与结点N相邻的兄弟中有一个键和指针数目超过最小数目，那么它的一个键和指针可以移动到N,此时对应父结点的键也要做相应处理（借的情况）

（2）如果两个相邻的兄弟中没有一个能提供额外的键给N时，此时有结点N和一个相邻的兄弟，其中一个的键数少于最小数，另外一个刚好为最小数，由于合并在一起没有超过单个结点的键和指针数目，因此可以合并。相应调整父结点，有时需要递归操作（合并的情况）。

B树——效率

由于每个块容纳的键数n（一般100以上）很大，绝大部分操作在叶子和其父结点，可以基本忽略B树的重组开销：

（1）读操作，I/O开销是B树层数加一次（数据文件一个I/O）。

（2）插入和删除操作是B树层次数加二次。

（3）通常B树三层就够了，除非数据库非常大。

（4）通过缓存根结点和第二层结点，I/O可以进一步减小