

外部排序与文件

一、外部排序

- 1、外存信息的存取
- 2、外部排序的方法
- 3、多路平衡归并的实现
- 4、置换-选择排序
- 5、最佳归并树

二、文件

- 1、文件的基本概念
- 2、顺序文件
- 3、索引文件

外部排序

1、外存信息的存取

1、外部排序：

内部排序：信息一次可全部调入内存，信息在内存中的处理时间是主要的时间耗费。

外部排序：信息量巨大，无法一次调入内存。只能驻留在带、盘、**CD-ROM** 上。特点为内存运行时间短，内、外存进行交换需要时间长。减少 I/O 时间成为主要矛盾。

2、基本术语：

- 记录 (**Record**)：数据项的集合存于内存，称之为结点。如果存之于外存，则叫做记录。原因起源于是在历史上研究管理应用和计算机科学的两部分人员的习惯。
- 域（场）：记录中的每个数据项，称之为域 (**Field**) 。
- 文件：记录的集合。
- 关键字：唯一标识记录的域，称之为关键字。
- 有序文件：文件根据关键字的大小。排成递增或递减的序列。

1、外存信息的存取

3、常用外存：

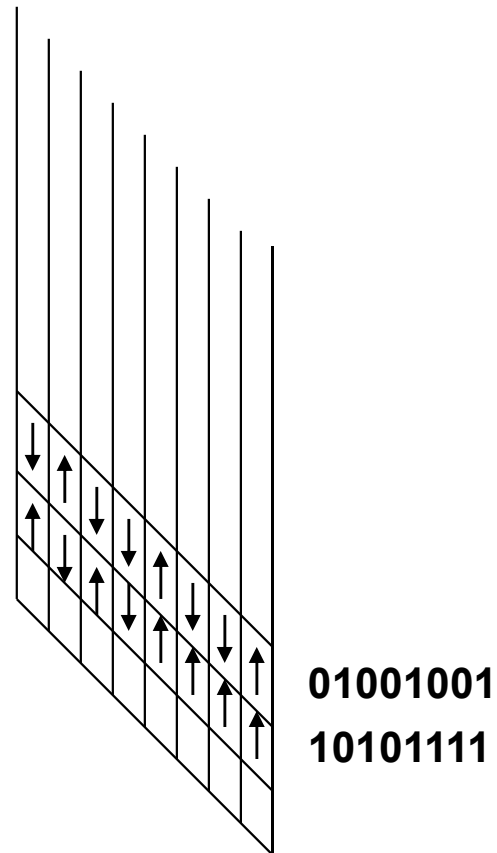
- 磁带：由磁带介质、读、写磁头、驱动器、接收盘和原始盘组成。

便宜、可反复使用、是一种顺序存取设备。

查找费时、速度慢。

- 带信息的表示：

↑ 一种磁化方向、代表**1**
↓ 另一种磁化方向，代表**0**



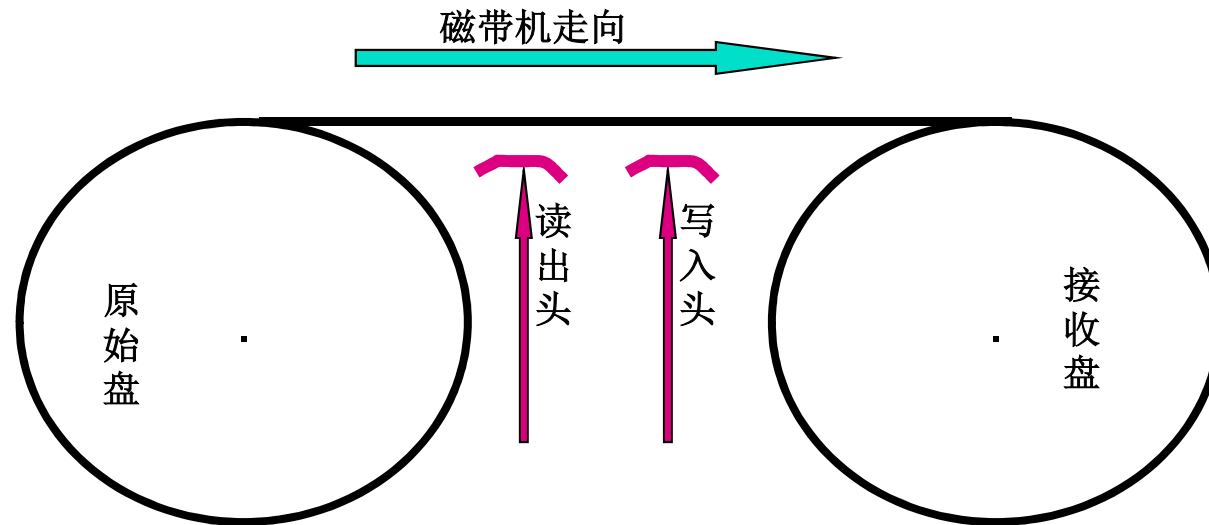
1、外存信息的存取

3、常用外存：

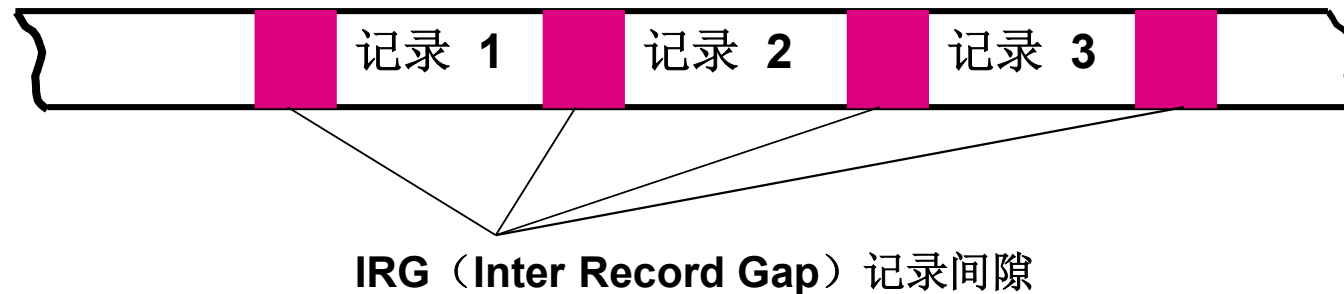
- 磁带：由磁带介质、读、写磁头、驱动器、接收盘和原始盘组成。

便宜、可反复使用、是一种顺序存取设备。

查找费时、速度慢（尤其是查找末端记录时）。



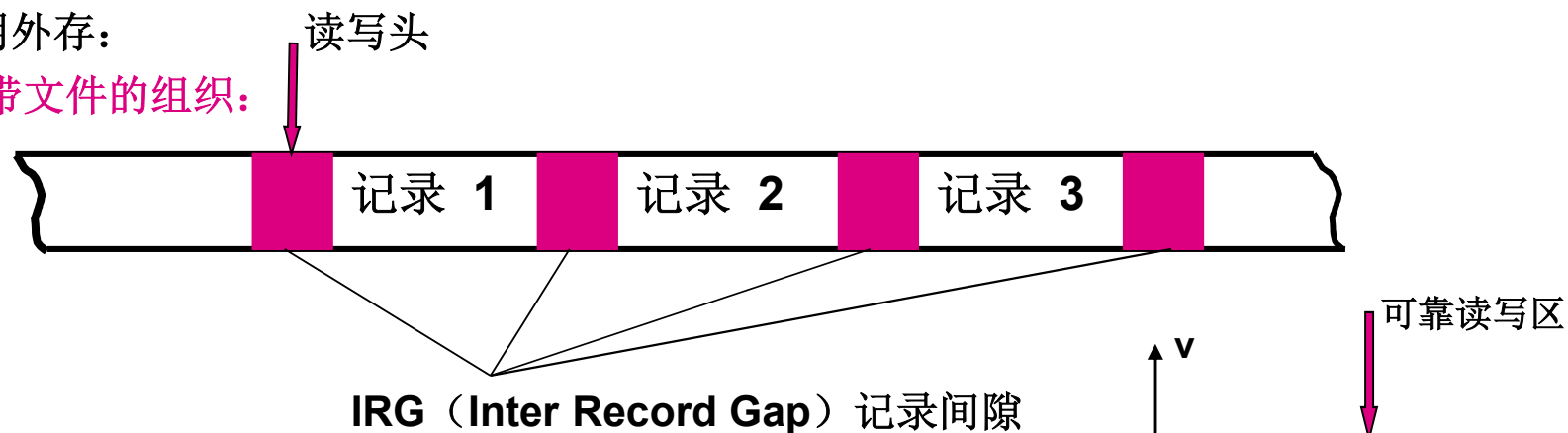
- 带文件的组织：



1、外存信息的存取

3、常用外存：

- 带文件的组织：



IRG: .5~.75 inch, 带来的问题是什么?

带的利用率下降, 如:

密度 **800 byte per inch** 的带。设每个记录有 **80 byte**, 共 **1000** 个记录。如果, **IRG = .6 inch**; 带的利用率?

$$1000 \times (80/800) = 100 \text{ inch (file)}$$

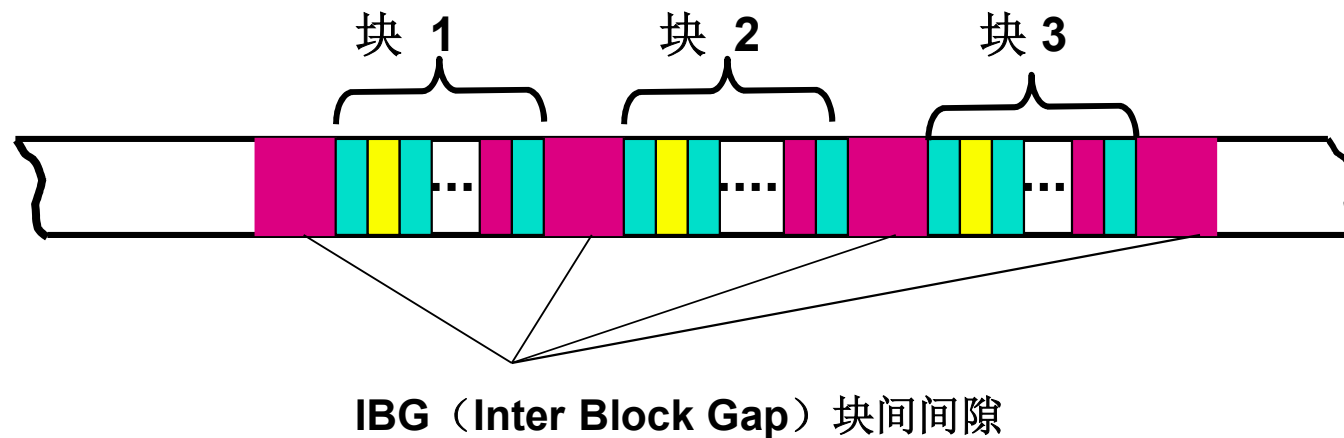
$$1000 \times 0.6 = 600 \text{ inch (total IRG)}$$

利用率 = $1/7 = 14\%$ 必须改进带的利用率!

1、外存信息的存取

3、常用外存：

- 带文件的组织的改进：



IBG: .5~.75 inch，带来的好处是什么？

每一块是一个物理记录，包含若干个逻辑记录。

B.F (块因子) = 一个物理记录包含逻辑记录的个数

带的利用率上升，如上例：

设 **B.F = 100**

$1000 / 100 \times 0.6 = 6 \text{ inch (total IBG)}$

$1000 \times 80 / 800 = 100 \text{ inch (file)}$ 利用率 = $100 / 106 = 94.3 \%$

1、外存信息的存取

3、常用外存：

- 带文件的读写时间： $T_{i/o} = t_a + n \times t_w$

t_a 延迟时间：读写头到达相应的物理块的起始位置的时间。

t_w 读/写一个字符的时间； n 字符数。

1、外存信息的存取

3、常用外存：

- 磁盘：由存取装置、读、写磁头、活动臂、盘片（磁道、扇区）、旋转主轴构成。
速度快、容量大、直接存取设备。

- 种类：

固定头磁盘：每个磁道都有一个磁头（速度快）

活动头磁盘：每个盘面共用一个磁头，
增加了找道的时间，应用广泛。

- 柱面：各盘面的直径相同的磁道的总和。

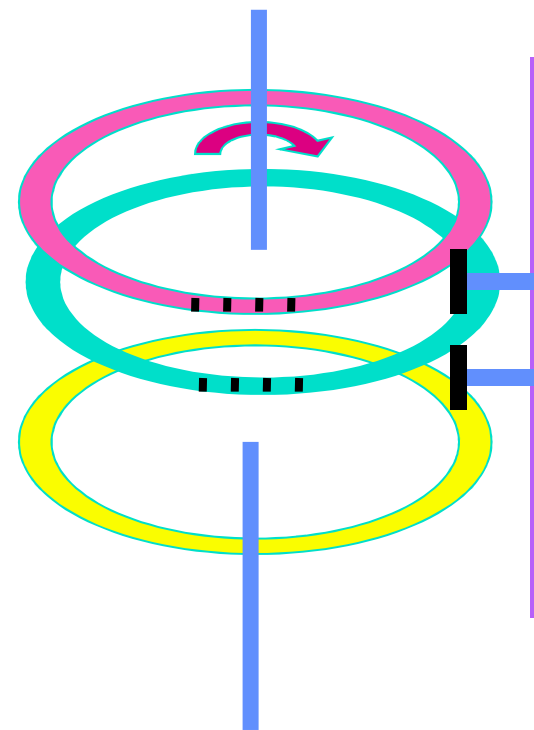
- 物理位置：柱面号、
盘面号、
块号（扇区号）

- 盘文件的读写时间： $T_{i/o} = t_{seck} + t_{la} + n \times t_{wm}$

t_{seck} ：找道时间，磁头定位的时间

t_{la} ：等待时间，等待信息块的初始位置旋转到读写头下的时间

t_{wm} ：每字符传输时间。



2、外部排序的方法

1、步骤：

- 生成合并段（run）：读入文件的部分记录到内存 → 在内存中进行内部排序 → 将排好序的这些记录写入外存，形成合并段 → 再读入该文件的下面的记录，往复进行，直至文件中的记录全部形成合并段为止。

7、15、19	⋮	8、11、13	⋮	16、23、31	⋮	5、12
---------	---	---------	---	----------	---	------
- 外部合并：将上一阶段生成的合并段调入内存，进行合并，直至最后形成一个有序的文件。
- 平衡合并分类法：被合并的初始合并段均匀分布在 K 条磁带上，即分布在 T_1 、 T_2 、..... T_k 上。对这 K 条带进行合并，将生成的中间归并段分布在 T_{K+1} 、 T_{K+2} 、..... T_{2K} 上。然后，循环往复，直至最后形成一个单一的合并段为止。

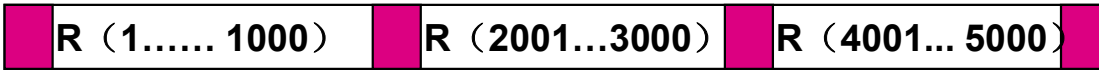
e.g: 平衡 2 路归并， $K = 2$ 。 $2K = 4$, 需 4 条磁带。六个初始合并段均匀分布在 T_1 、 T_2 上。每个合并段有 1000 个记录。 T_3 、 T_4 初始为空。合并过程如下：

初始分布：

T_1	R (1..... 1000)	R (2001...3000)	R (4001... 5001)
T_2	R (1001..... 2000)	R (3001...4000)	R (5001... 6000)
T_3	空	T_4	空

2、外部排序的方法

- 初始分布:

T_1 

T_2 

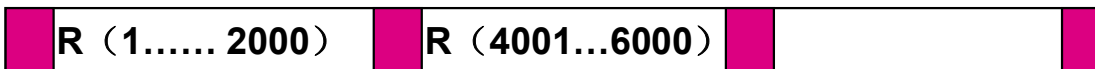
T_3 : 空

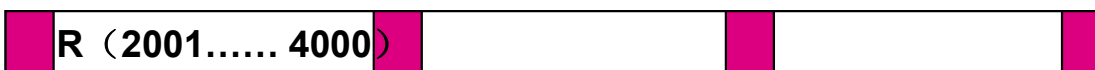
T_4 : 空

- 第一趟归并:

T_1 : 空


T_2 : 空

T_3 : 

T_4 : 

2、外部排序的方法

- 第二趟归并：

T_1 : 

T_2 : 


T_3 : 空

T_4 : 空

- 第三趟归并：

T_1 : 空

T_2 : 空

T_3 : 

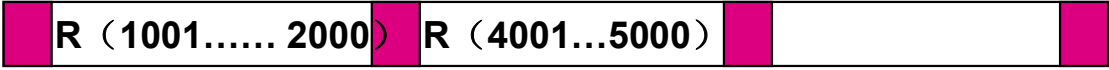
T_4 : 空

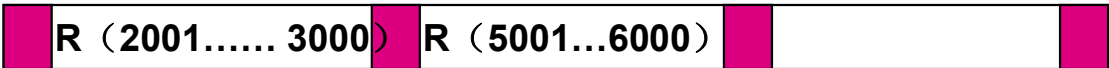
2、外部排序的方法

e.g: 平衡 3 路归并, $K = 3$ 。 $2K = 6$, 需 6 条磁带。 六个初始合并段均匀分布在 T_1 、 T_2 、 T_3 上。 每个合并段有 1000 个记录。 T_4 、 T_5 、 T_6 初始为空。 合并过程如下:

初始分布:

T_1 : 


T_2 : 

T_3 : 

T_4 : 空 T_5 : 空 T_6 : 空

• 第一趟归并:

T_1 : 空 T_2 : 空 T_3 : 空

T_4 : 

T_5 : 


T_6 : 空


2、外部排序的方法

e.g: 平衡 3 路归并, $K = 3$ 。 $2K = 6$, 需 6 条磁带。 六个初始合并段均匀分布在 T_1 、 T_2 、 T_3 上。 每个合并段有 1000 个记录。 T_4 、 T_5 、 T_6 初始为空。 合并过程如下:

- 第一趟归并:

T_1 : 空 T_2 : 空 T_3 : 空

T_4 :  $R (1..... 3000)$

T_5 :  $R (3001..... 6000)$

T_6 : 空

- 第二趟归并:

T_1 :  $R (1..... 6000)$

T_2 : 空 T_3 : 空 T_4 : 空

T_5 : 空 T_6 : 空

2、外部排序的方法

一般情况下：

外部排序所需的时间 = 内部排序（产生初始归并段）所需的时间 + 外存信息读写的时间 + 内部归并所需的时间。

其中，主要的时间花费在外存的信息读写（I/O读写）。外存的信息读写次数，主要与归并趟数相关。对于K路平衡归并：

归并趟数： $\lceil \log_k m \rceil$ 其中 k 是路数； m 是初始合并段数。在上例中：
 $\lceil \log_2 6 \rceil = 3$ 而 $\lceil \log_3 6 \rceil = 2$ 。

总的外存信息读写时间：每一趟归并时，对文件中的所有的记录都要全部读写一次。此外，还有一次生成所有合并段的时间。

为了减少归并趟数，有两种可能：

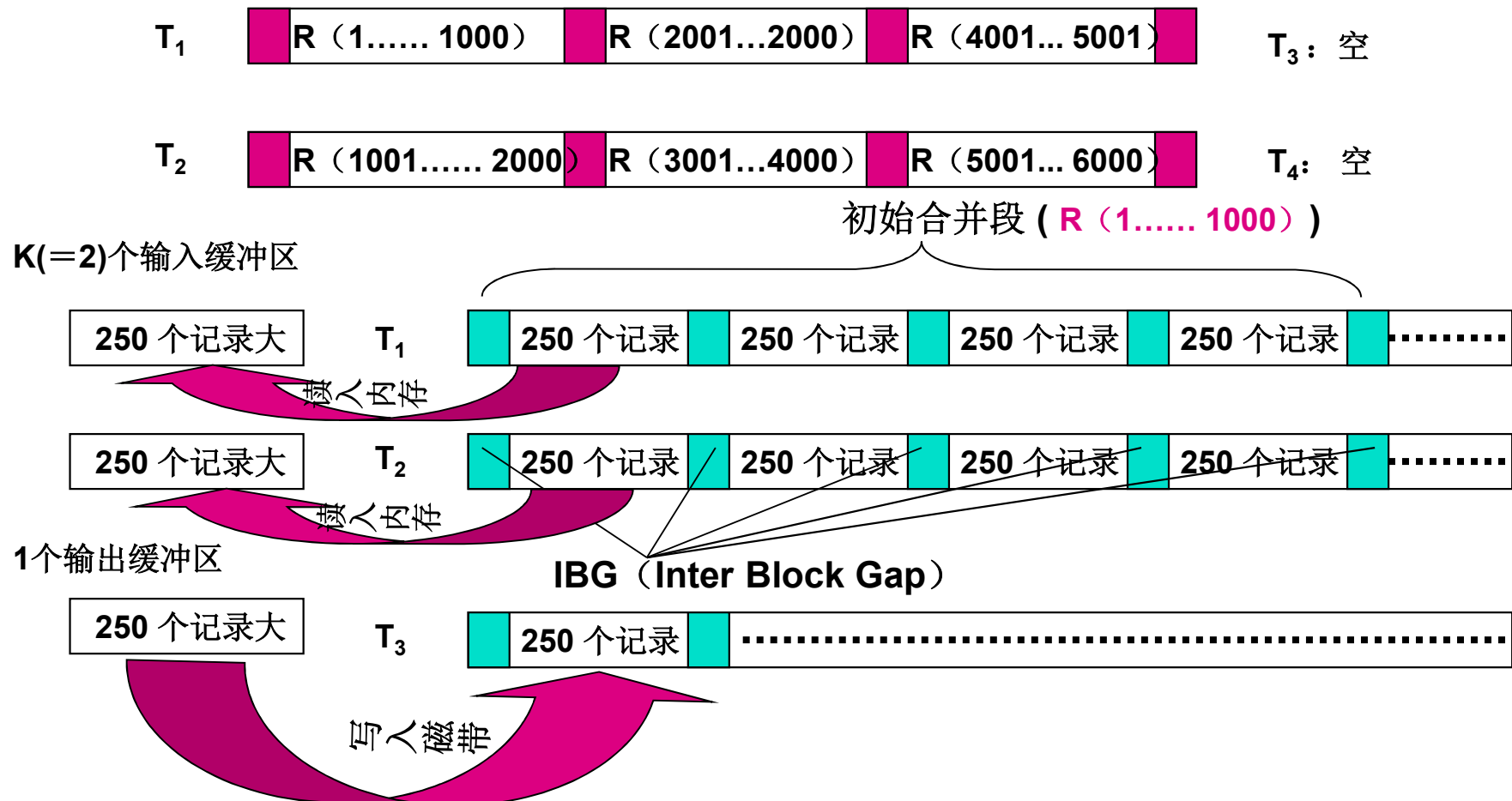
（1） K 大，趟数减少，读写记录的总数将减少。但 K 大，需要的内存将越多。

（2）减少初始合并段数 m ，将使趟数减少，读写记录的总数将减少。这就要求在内存一定且进行内部排序时，生成尽可能长的归并段，从而减少归并段的总数。

以下分别介绍这两种方法：胜者树和败者树，置换选择排序

2、外部排序的方法

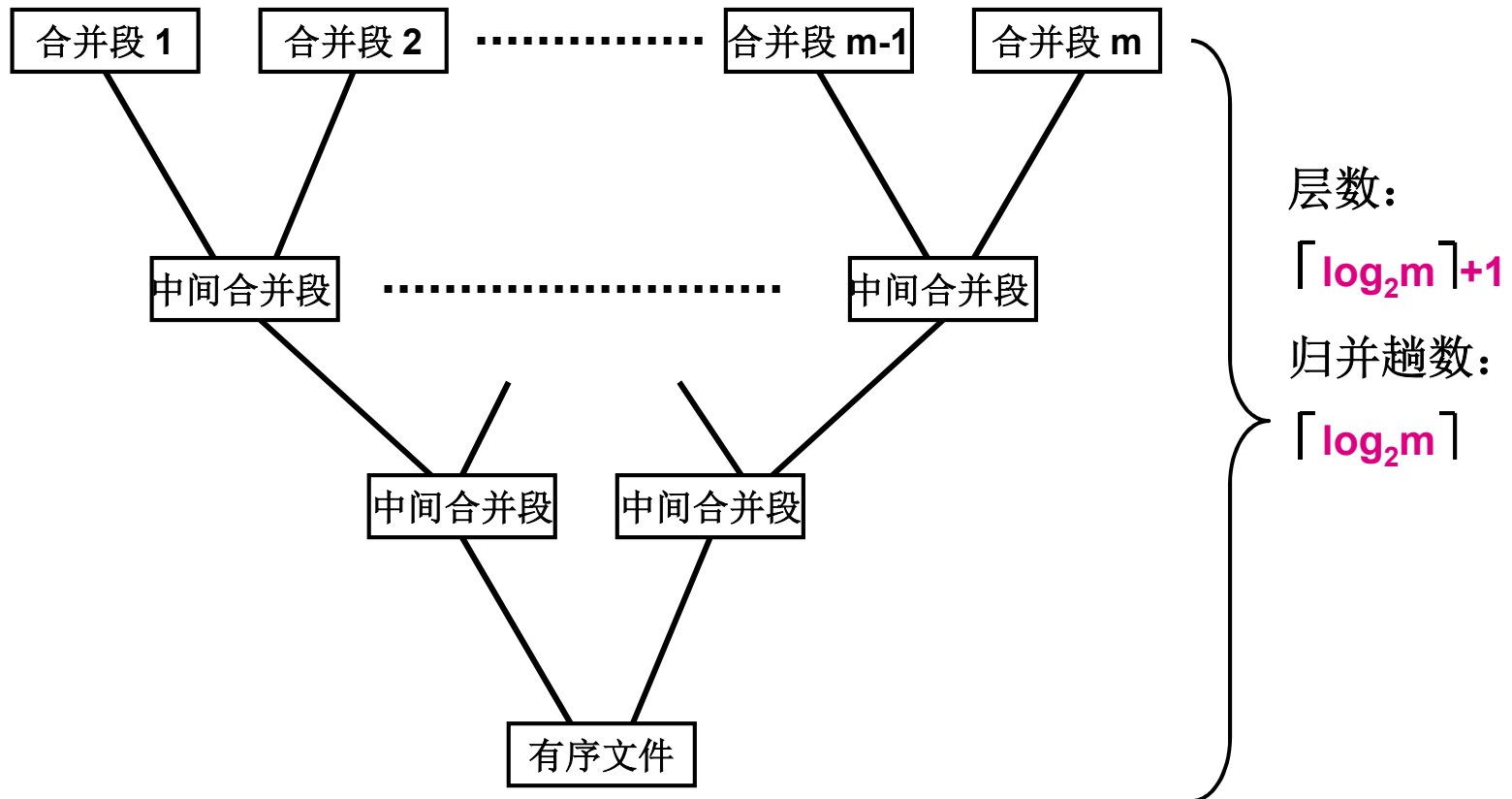
- 输入、输出缓冲区的安排: 设进行平衡 2 路归并, $K = 2$ 。 $2K = 4$, 需 4 条磁带。 六个初始合并段均匀分布在 T_1 、 T_2 。 每个合并段有 1000 个记录。 T_3 、 T_4 初始为空。 设每个物理块包含 250 个记录, 则每个初始合并段包含 4 个物理块。 K 路合并, K 个输入输入缓冲区和一个输出缓冲区。



3、多路平衡归并的实现

1、带、盘的平衡多路归并的性质：

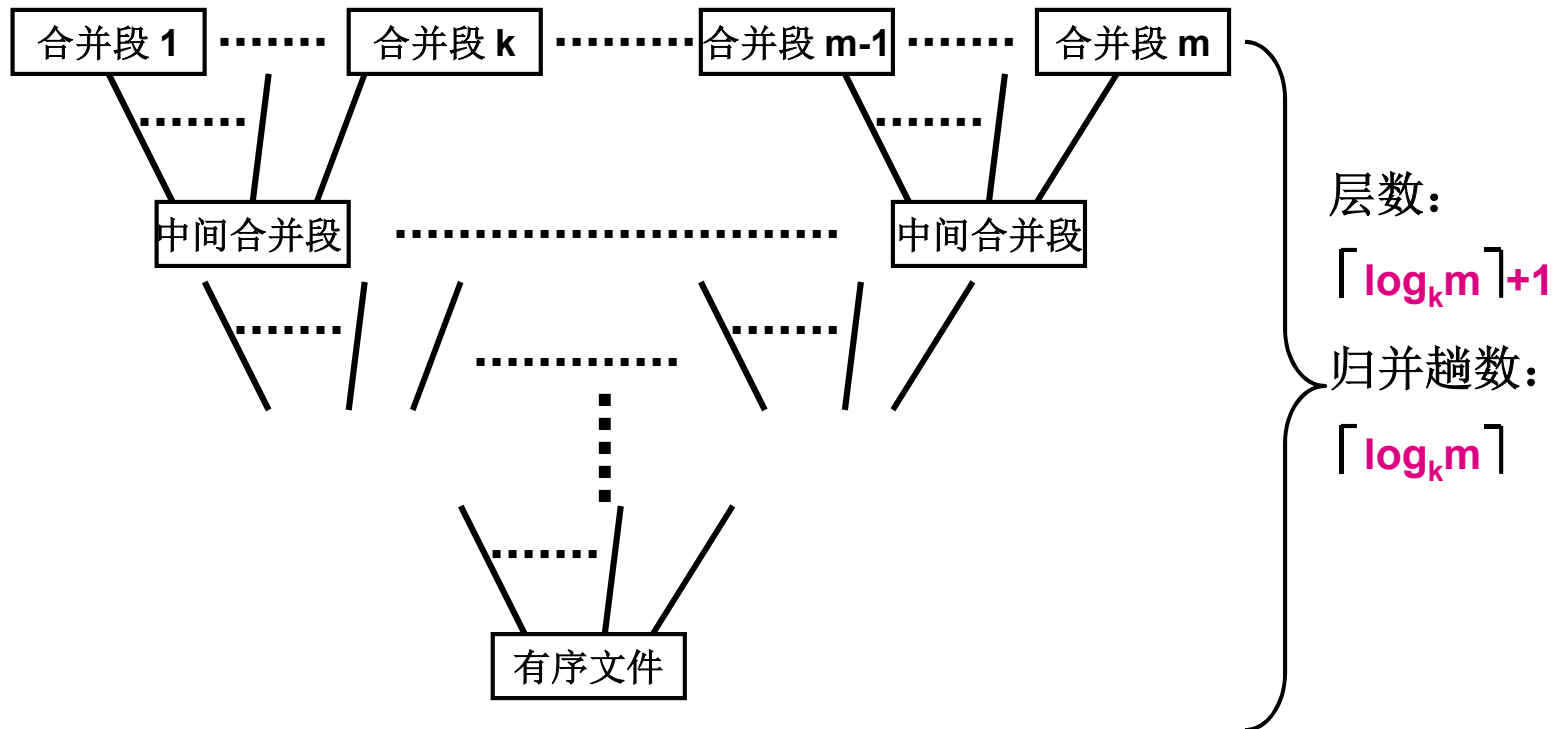
- 时间分析： $\lceil \log_k m \rceil$ 趟。 K 大，趟数减少，读写外存的总数将减少。但 K 大，会带来什么问题呢？以下，分析一下内部归并所需的时间。
- e.g: $K = 2$ 时, m 个归并串。总共 n 个记录。



3、多路平衡归并的实现

1、带、盘的平衡多路归并的性质：

- e.g: $K > 2$ 时, 趟数将会减少。 m 个归并串。总共 n 个记录。



- 设从 k 个元素中挑选一个最小的元素需 $(k-1)$ 次比较。每次比较耗费的时间代价为:

t_{mg} , 那么在进行 k 路平衡归并时, 总的内部归并时间耗费不会超过:

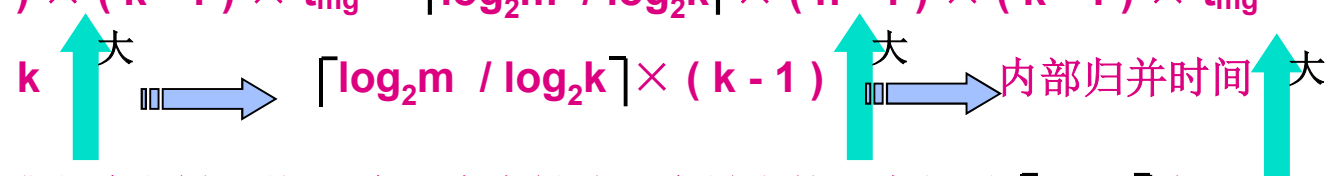
$$\lceil \log_k m \rceil \times (n - 1) \times (k - 1) \times t_{mg} = \lceil \log_2 m / \log_2 k \rceil \times (n - 1) \times (k - 1) \times t_{mg}$$

3、多路平衡归并的实现

1、带、盘的平衡多路归并的性质：

- 设从 k 个元素中挑选一个最小的元素需 $(k-1)$ 次比较。每次比较耗费的时间代价为：

t_{mg} ，那么在进行 k 平衡归并时，总的时间耗费不会超过：

$$\lceil \log_k m \rceil \times (n-1) \times (k-1) \times t_{mg} = \lceil \log_2 m / \log_2 k \rceil \times (n-1) \times (k-1) \times t_{mg}$$


- 改进：采用胜者树或者败者树，从 K 个元素中挑选一个最小的元素仅需 $\lceil \log_2 k \rceil$ 次比较，这时总的时间耗费将下降：

$$\lceil \log_k m \rceil \times \lceil \log_2 k \rceil \times (n-1) \times t_{mg}$$

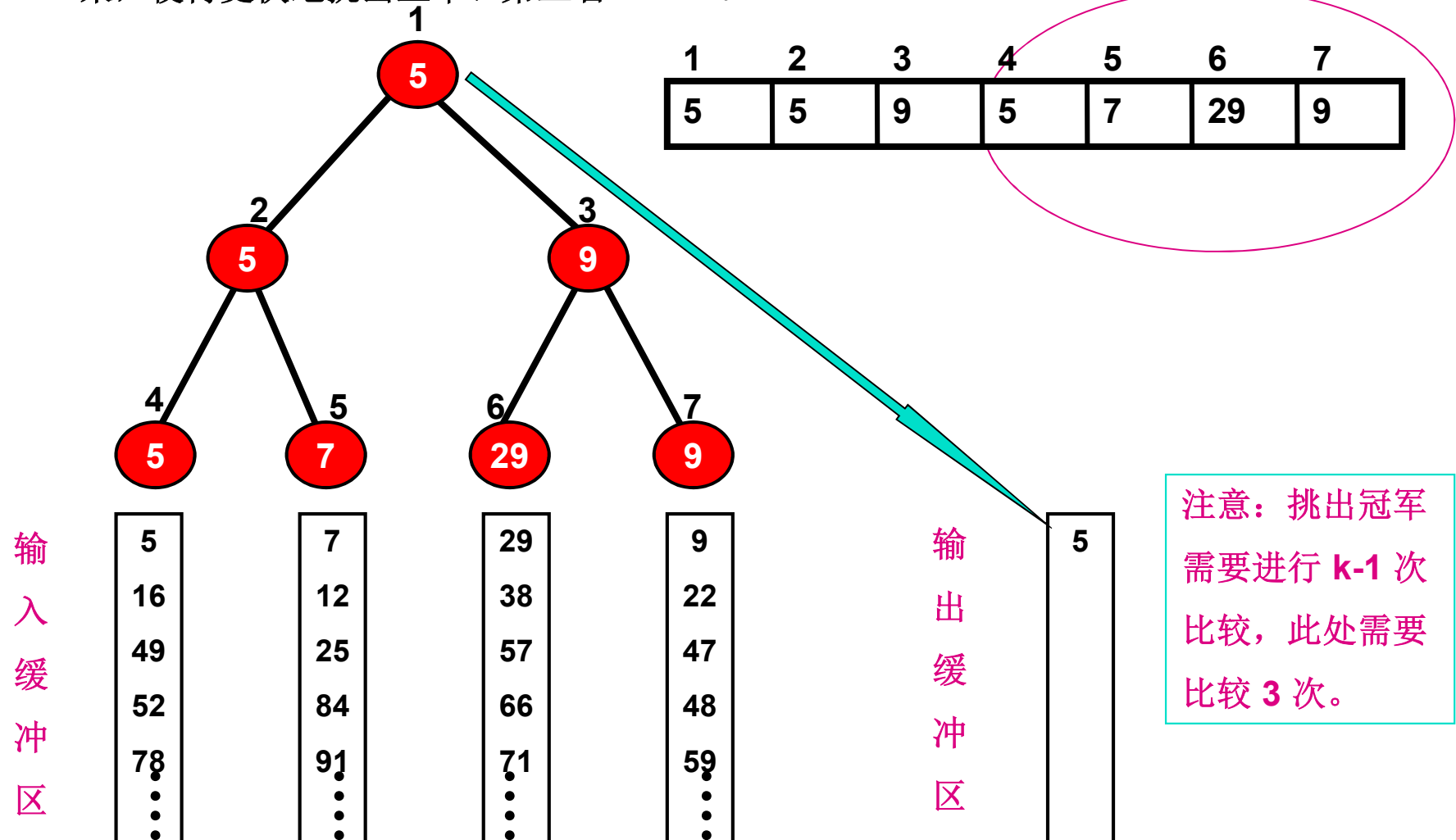
2、胜者树及其使用

胜者进入下一轮，直至决出本次比赛的冠军。决出冠军之后，充分利用上一次比赛的结果，使得更快地挑出亚军、第三名 ……。

3、多路平衡归并的实现

2、胜者树及其使用

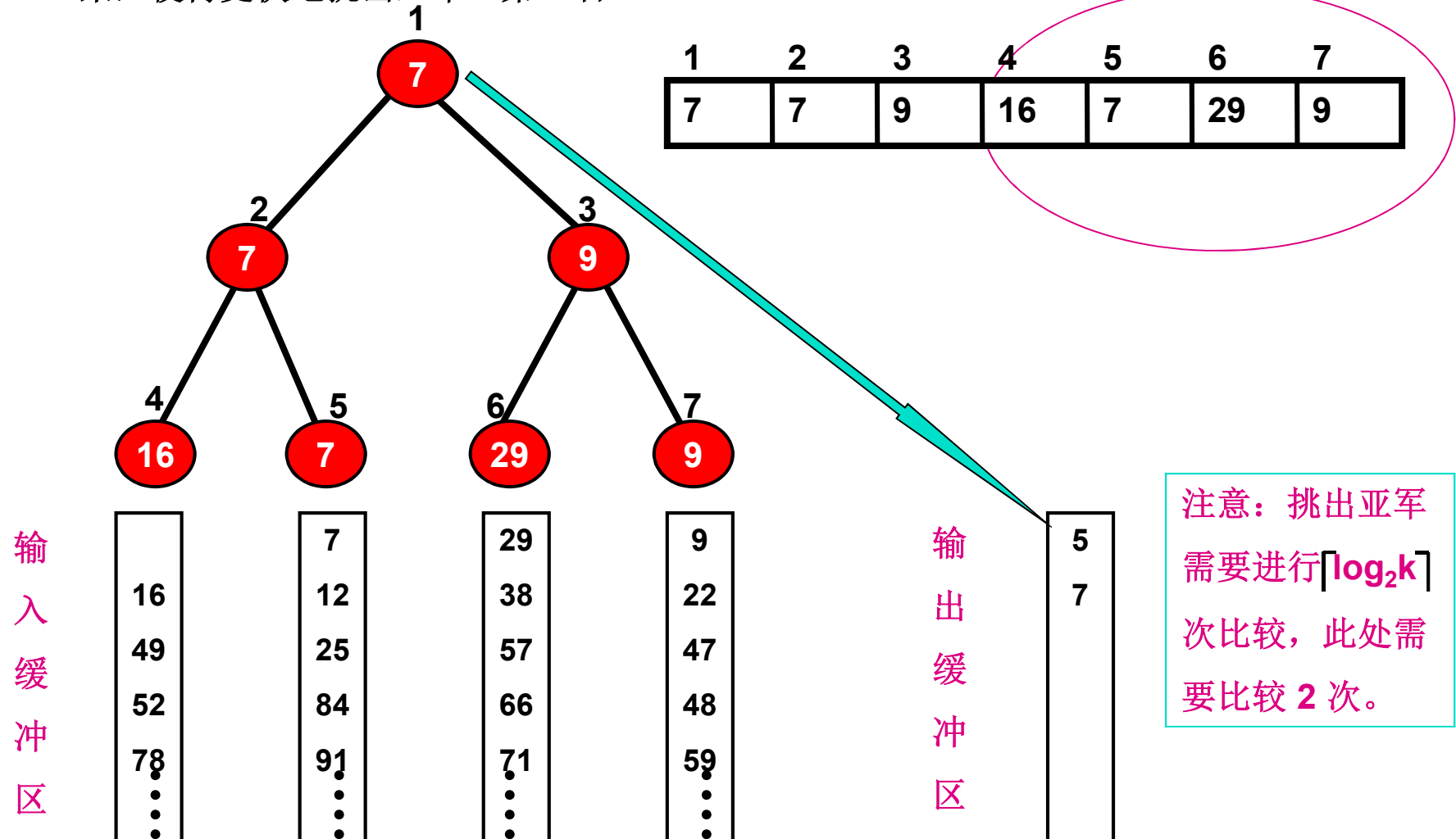
胜者进入下一轮，直至决出本次比赛的冠军。决出冠军之后，充分利用上一次比赛的结果，使得更快地挑出亚军、第三名



3、多路平衡归并的实现

2、胜者树及其使用

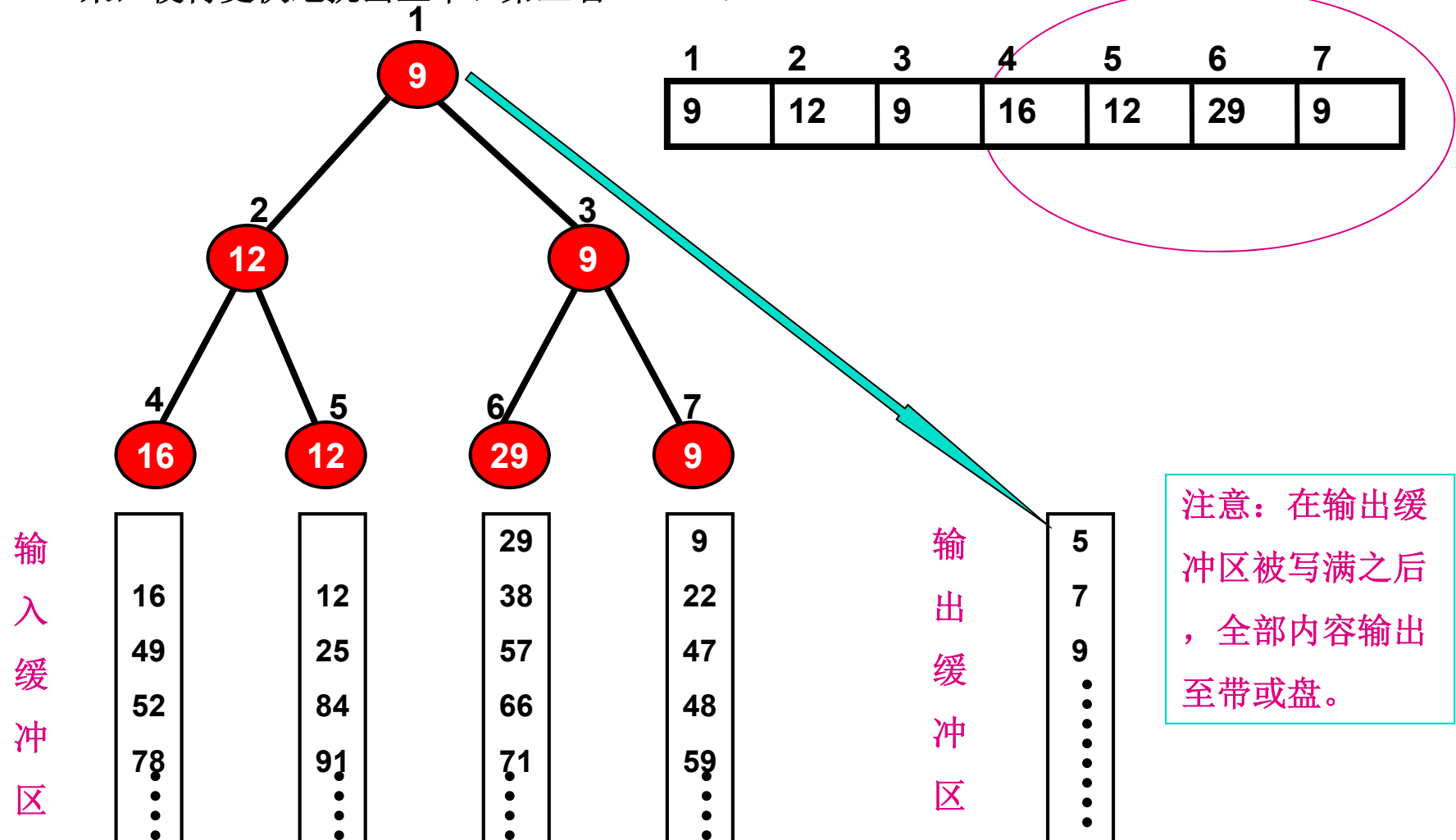
胜者进入下一轮，直至决出本次比赛的冠军。决出冠军之后，充分利用上一次比赛的结果，使得更快地挑出亚军、第三名



3、多路平衡归并的实现

2、胜者树及其使用

胜者进入下一轮，直至决出本次比赛的冠军。决出冠军之后，充分利用上一次比赛的结果，使得更快地挑出亚军、第三名



3、多路平衡归并的实现

2、胜者树及其使用

- 胜者进入下一轮，直至决出本次比赛的冠军。决出冠军之后，充分利用上一次比赛的结果，使得更快地挑出亚军、第三名。

决出第一名需比较： $k - 1$ 次

决出第二名需比较： $\lceil \log_2 k \rceil$ 次

决出第三名需比较： $\lceil \log_2 k \rceil$ 次

⋮

- 结果：采用胜者树后，从 K 个元素中挑选一个最小的元素仅需 $\lceil \log_2 k \rceil$ 次比较，这时总的时间耗费下降为：

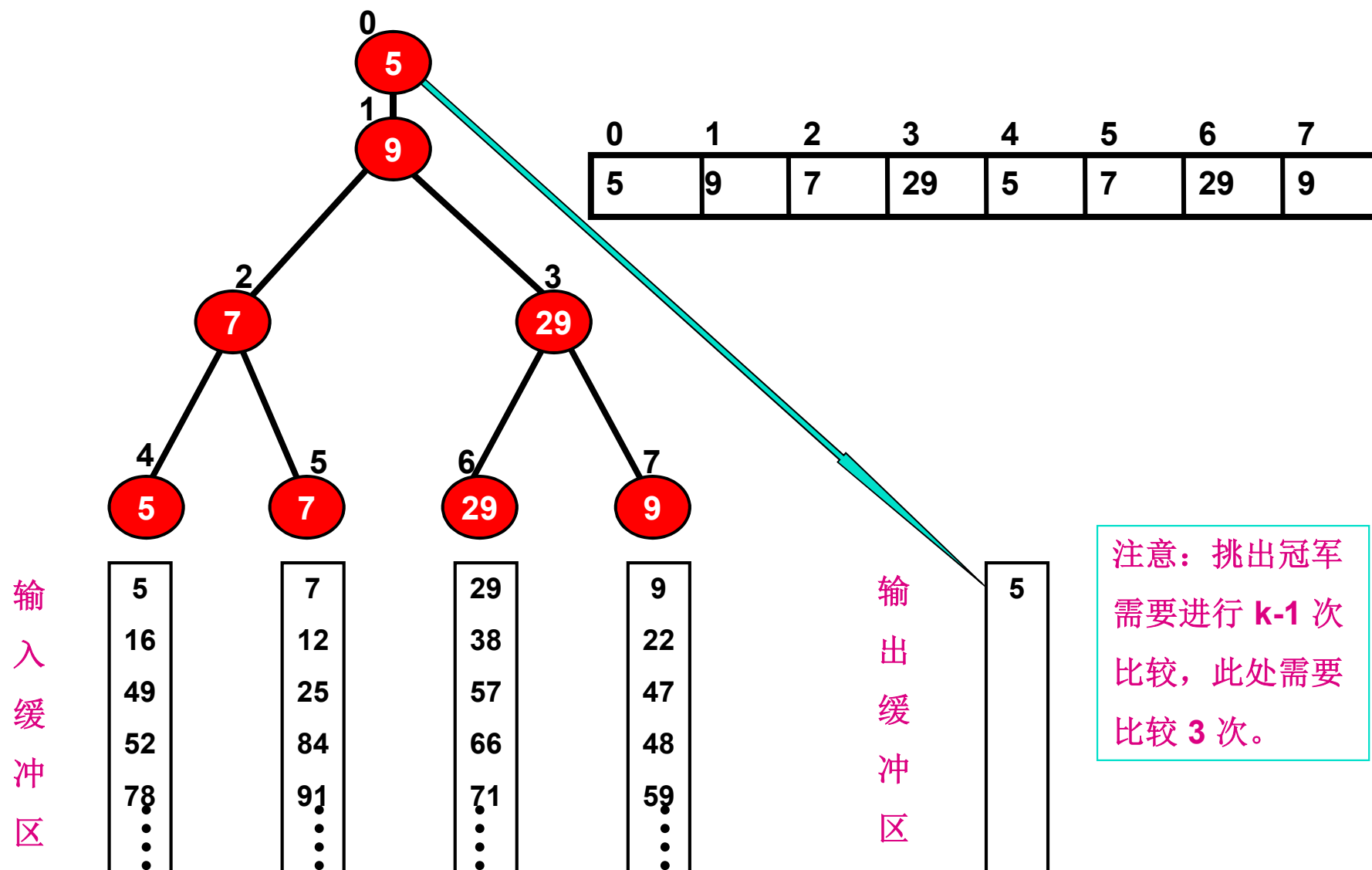
$$\lceil \log_k m \rceil \times \lceil \log_2 k \rceil \times (n - 1) \times t_{mg}$$

有意思的是该结果和 k 无关，这是通过多用空间换来的。

- 改进：采用胜者树， K 个元素中最小的元素输出之后，因为该结点在每次比较中都是胜利者，所以从根结点到它的相应的叶子结点路径上的结点都需要进行修改，为了加快程序运行的速度产生了败者树。

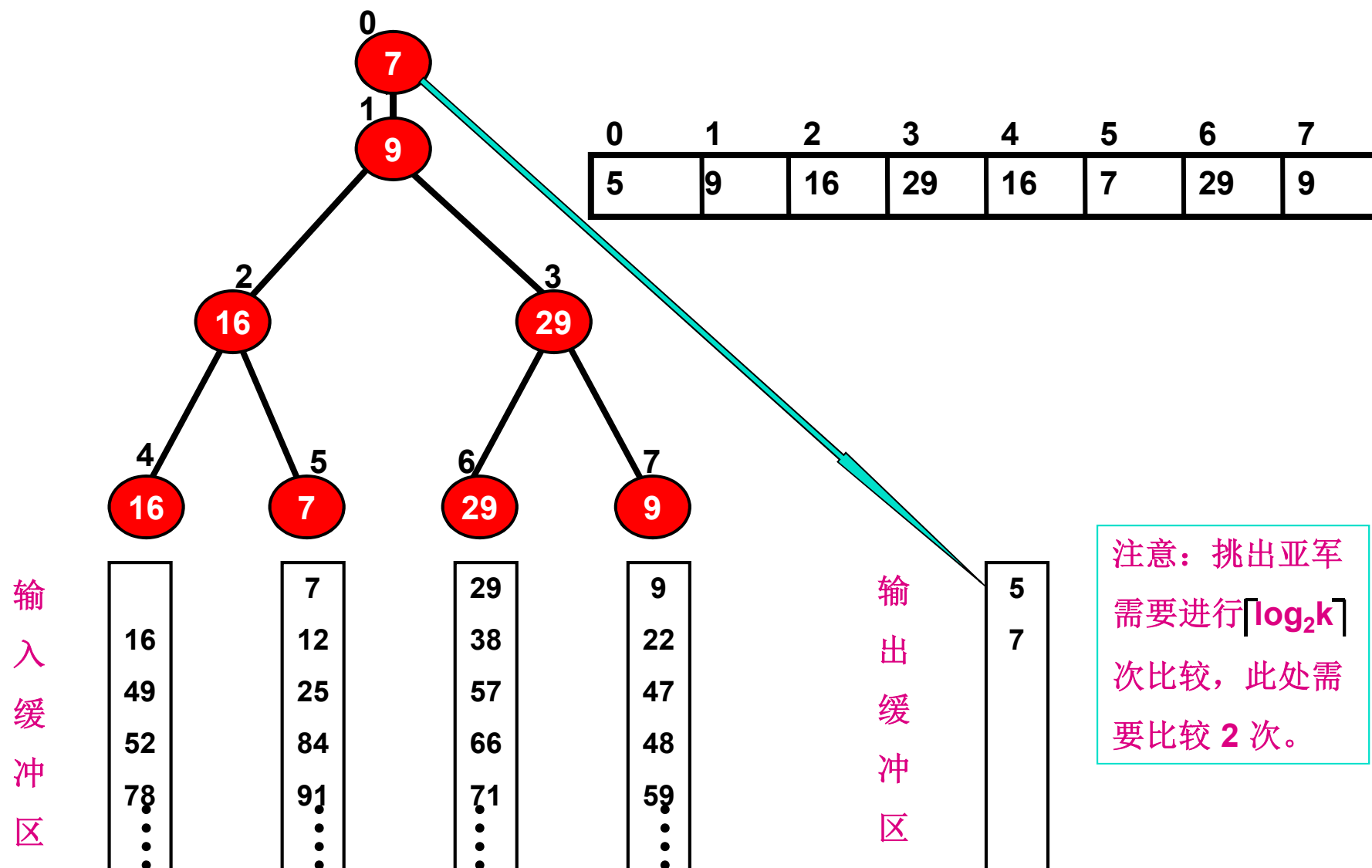
3、多路平衡归并的实现

3、败者树及其使用



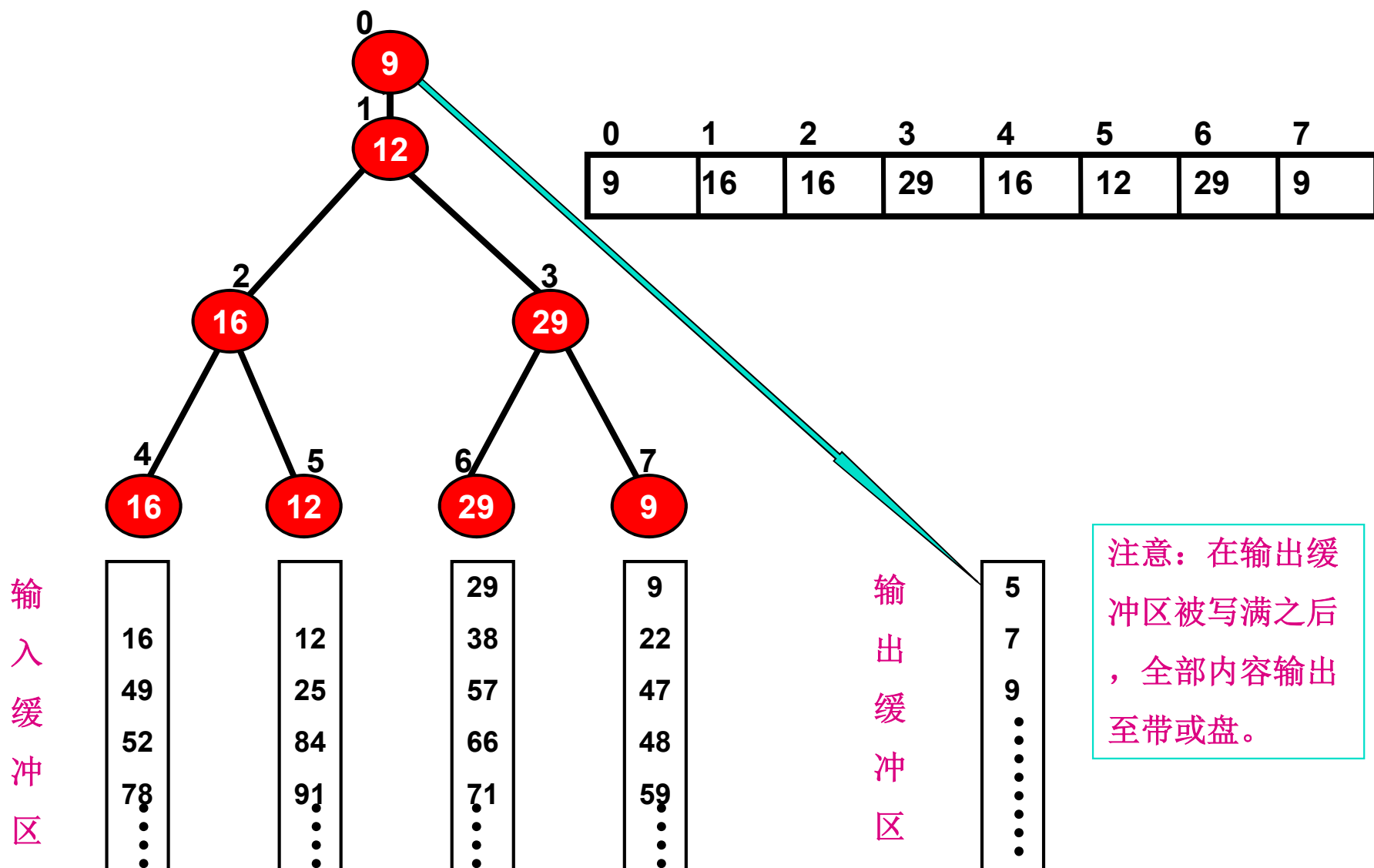
3、多路平衡归并的实现

3、败者树及其使用



3、多路平衡归并的实现

3、败者树及其使用



3、多路平衡归并的实现

3、败者树的程序实现

```
typedef int LoserTree[ k ];
```

```
typedef struct { KeyType key;  
                } Exnode, External[ k+1 ];
```

```
void K_Merge ( LoserTree &ls, External & b)
```

```
{ for ( i = 0; i < k ; ++i ) input(b[i].key);
```

```
  CreateLoserTree(ls);
```

```
  while ( b[ls[0]].key != MAXKEY )
```

```
    { q =ls[0]; output(q); input(b[q].key,q); Adjust( ls, q ); }
```

```
    output( ls[0]);
```

```
} // K_Merge
```

3、多路平衡归并的实现

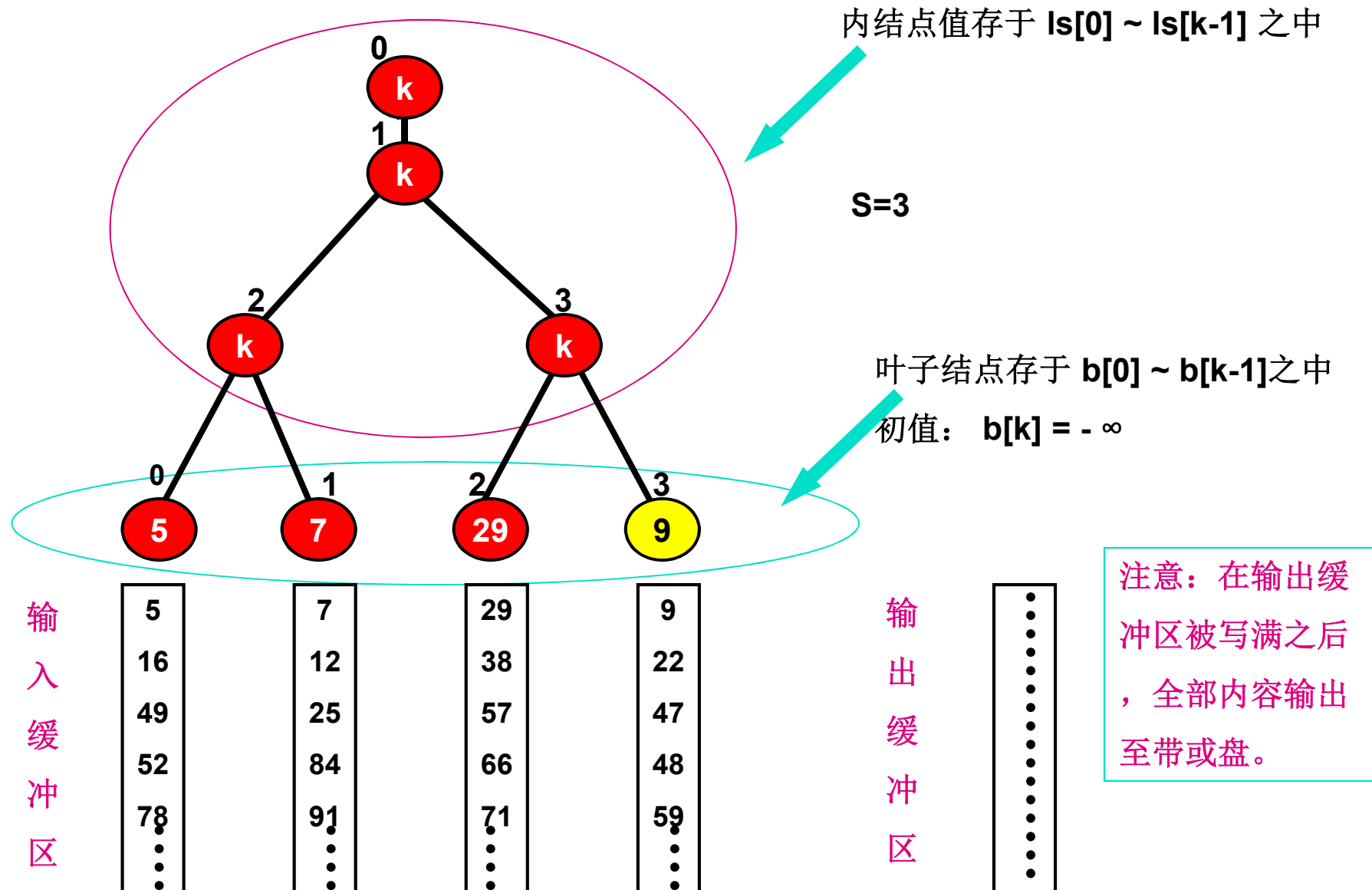
3、败者树的程序实现

```
void Adjust ( LoserTree &ls, int s)
{  t = (s + k )/2;
   while ( t >0 )
       { if (b[s].key > b[ls[t]].key) s <-> ls[t];      t = t / 2; }
   ls[0] = s;
} // Adjust
```

```
void CreateLoserTree ( LoserTree &ls )
{  b[k].key = MINKEY;
   for ( i=0; i < k; ++i ) ls[i]=k;
   for ( i=k-1; k >= 0; - -i ) Adjust(ls,i);
} // CreateLoserTree
```

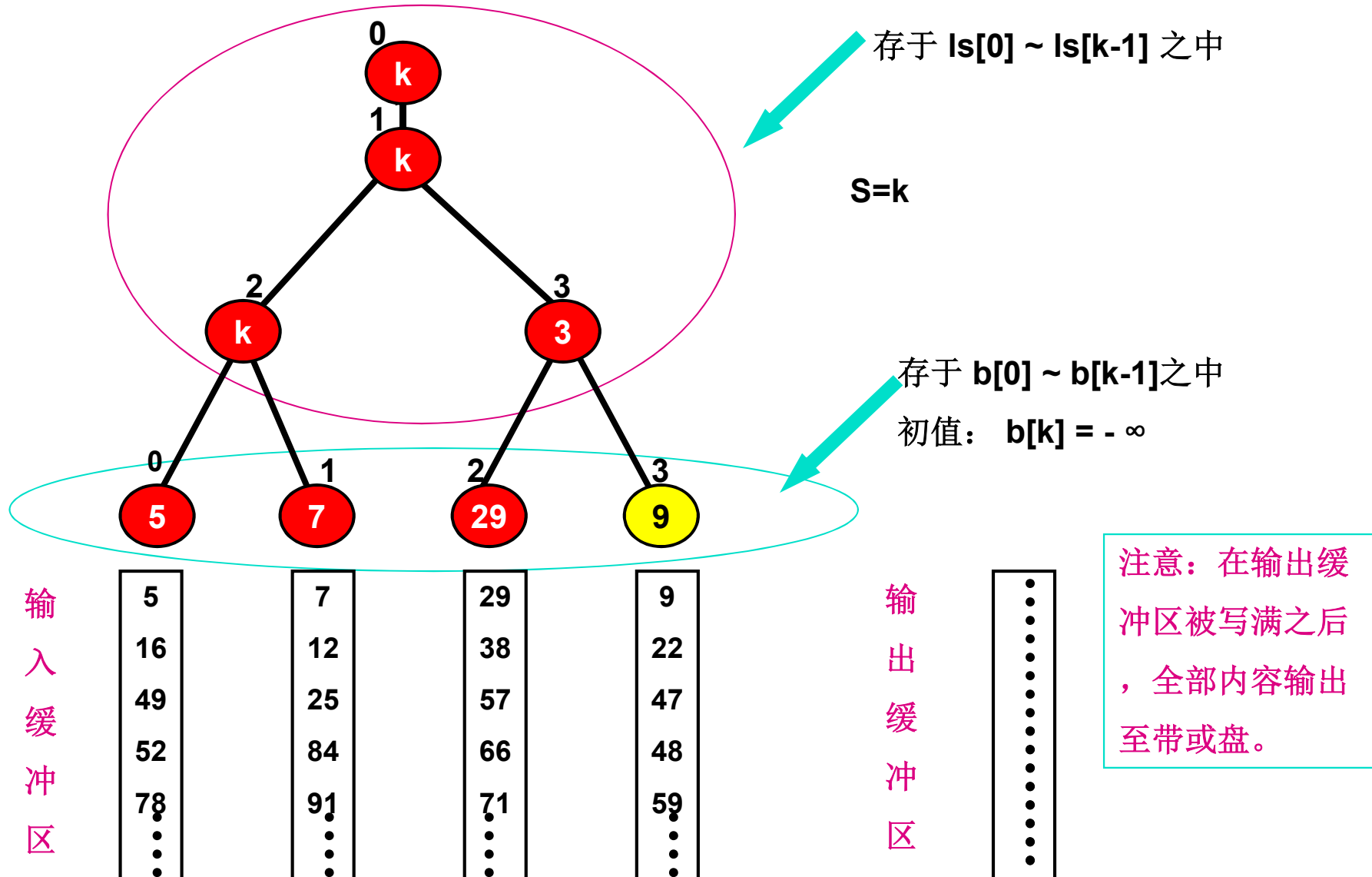
3、多路平衡归并的实现

- 败者树程序实现：



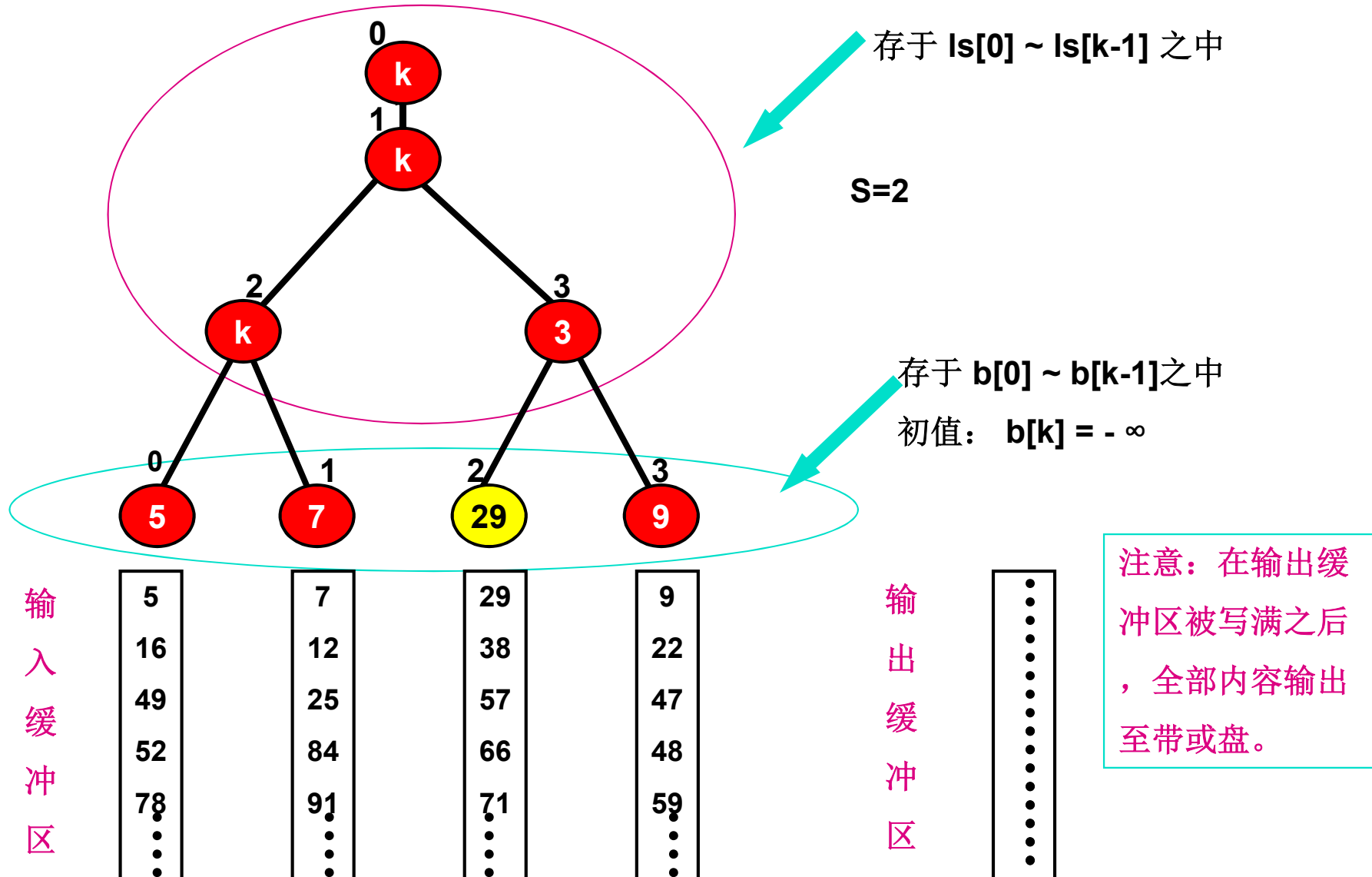
3、多路平衡归并的实现

- 败者树程序实现：



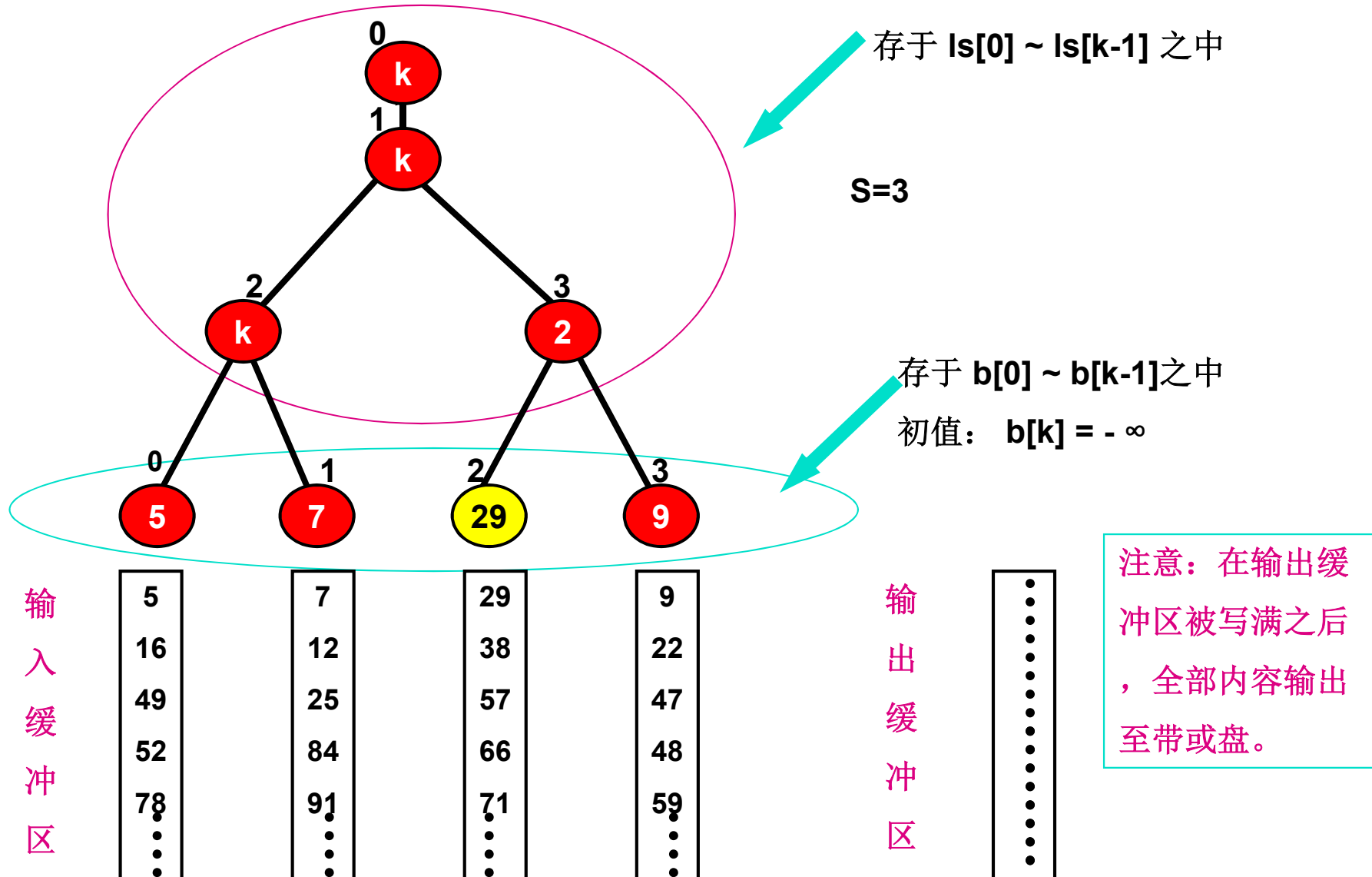
3、多路平衡归并的实现

- 败者树程序实现：



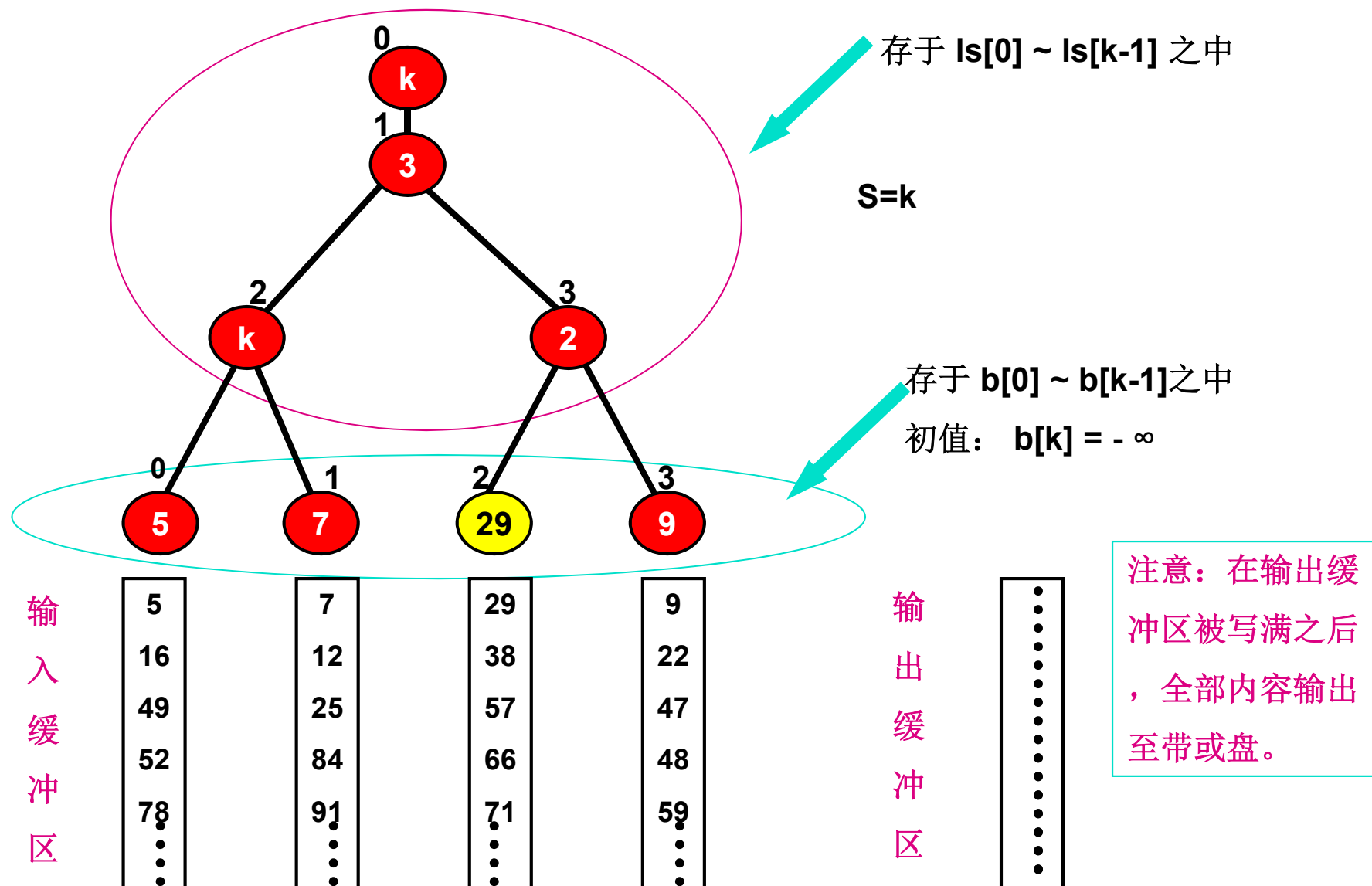
3、多路平衡归并的实现

- 败者树程序实现:



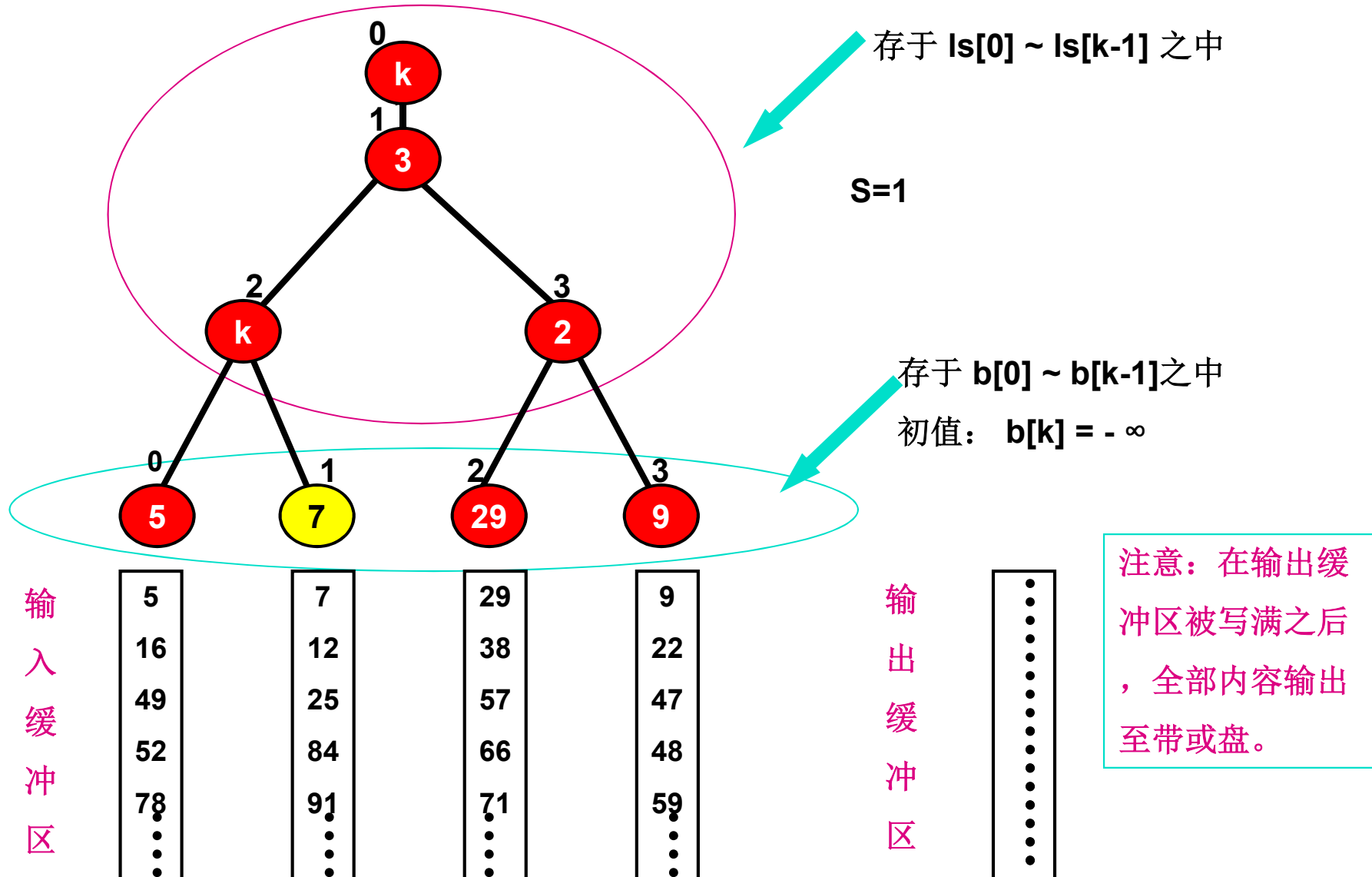
3、多路平衡归并的实现

- 败者树程序实现：



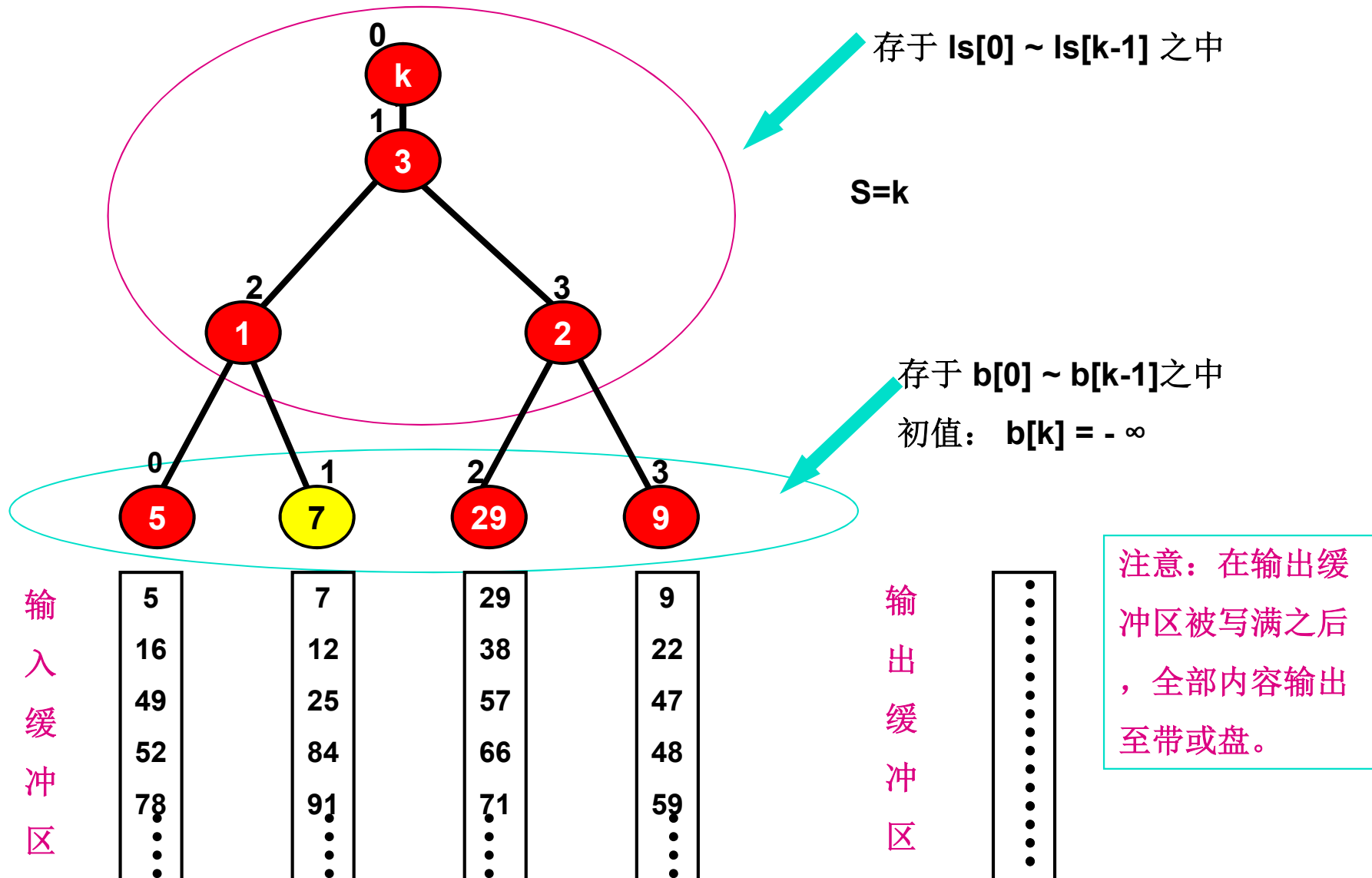
3、多路平衡归并的实现

- 败者树程序实现：



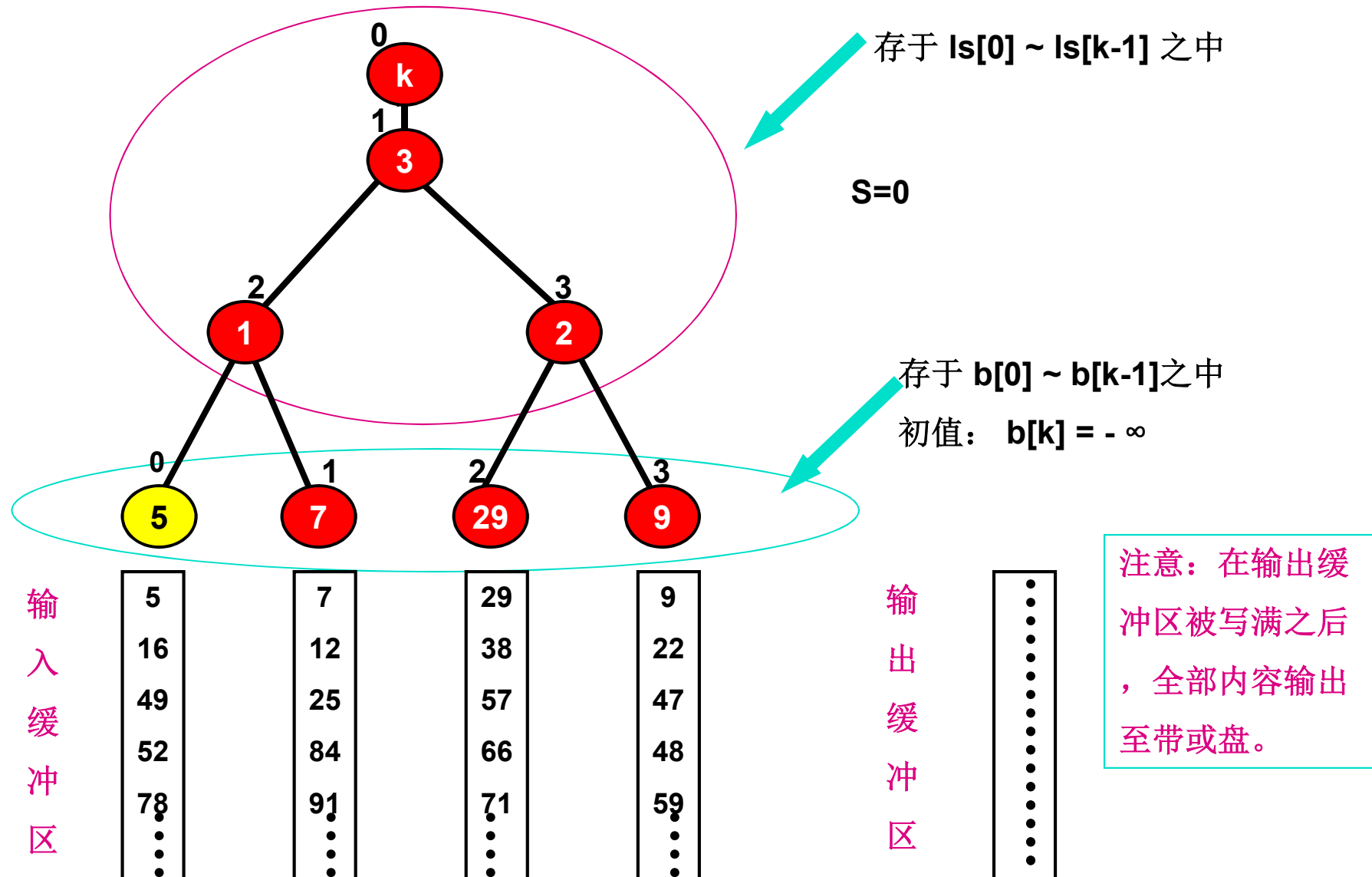
3、多路平衡归并的实现

- 败者树程序实现：



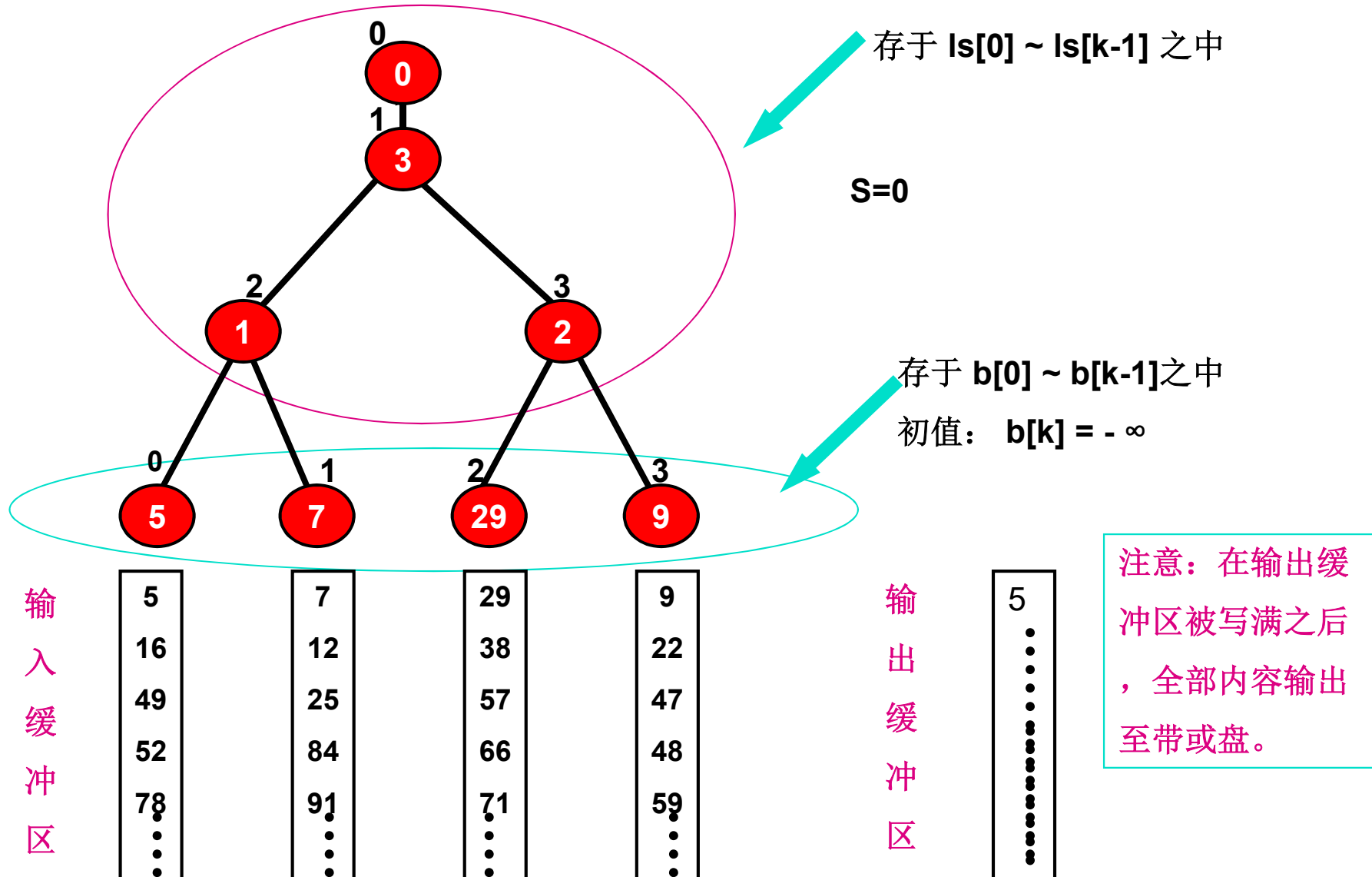
3、多路平衡归并的实现

- 败者树程序实现：



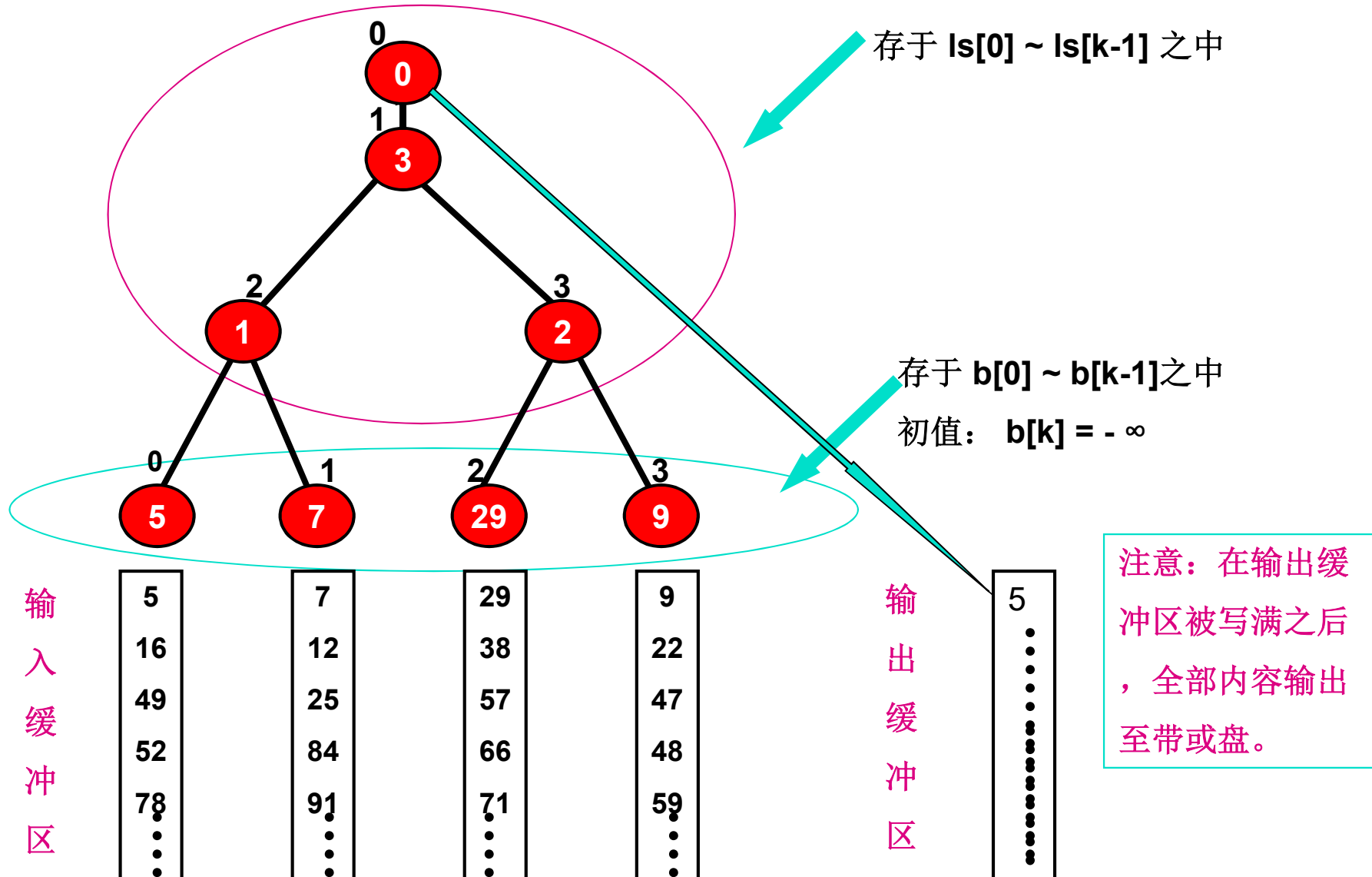
3、多路平衡归并的实现

- 败者树程序实现：



3、多路平衡归并的实现

- 败者树程序实现：



4、置换-选择排序

1、作用：产生尽可能长的初始归并段。在内存长度一定时，按照通常的排序方法，产生的初始归并段的长度只能和内存长度相同，但采用本方法之后却可以生成两倍内存长度的初始归并段（平均情况下）。

操作过程（**FI**为初始待排文件；**FO**为输出文件；**WA**为可容纳 w 个记录的内存工作区）：

- （1）从**FI**输入 w 个记录到工作区**WA**。
- （2）从**WA**中选出其中关键字取最小值的记录，记为**MINIMAX**记录。
- （3）将**MINIMAX**记录输出到**FO**中去。
- （4）若**FI**不空，则从**FI**输入下一个记录到**WA**中。
- （5）从**WA**中所有关键字比**MINIMAX**记录的关键字大的记录中选出最小关键字记录，记为新的**MINIMAX**记录。
- （6）重复（3）至（5），直至在**WA**中选不出新的**MINIMAX**记录为止，由此得到一个初始归并段，输入一个归并段的结束标志到**FO**中去。
- （7）重复（2）至（6），直至**WA**为空。由此得到全部初始归并段。

2、实例：**F1**: 31、21、19、12、26、41、11、37、15、23、29 在带 1 上

F2: 输出磁带 2

假定使用的内存只有 3 个记录长，利用置换-选择分类法产生初始合并段。

4、置换-选择排序

- 实例的实现过程： **F1**: 31、21、19、12、26、41、11、37、15、23、29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41

4、置换-选择排序

- 实例的实现过程： F1: 31、21、19、12、26、41、11、37、15、23、29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##

} 第一个初始合并段

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11
8	15、 37、 12	12

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11
8	15、 37、 12	12
9	15、 37、 23	15

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11
8	15、 37、 12	12
9	15、 37、 23	15
10	29、 37、 23	23

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11
8	15、 37、 12	12
9	15、 37、 23	15
10	29、 37、 23	23
11	29、 37	29

4、置换-选择排序

- 实例的实现过程： F1： 31、 21、 19、 12、 26、 41、 11、 37、 15、 23、 29 在带 1 上

步数	内存缓冲区内容（由 F1 输入）	输出结果（至带 F2 ）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11
8	15、 37、 12	12
9	15、 37、 23	15
10	29、 37、 23	23
11	29、 37	29
12	37	37

4、置换-选择排序

- 实例的实现过程： F1: 31、21、19、12、26、41、11、37、15、23、29 在带 1 上

步数	内存缓冲区内内容（由 F1 输入）	输出结果（至带 F2）
1	31、 21、 19	19
2	31、 21、 (12)	21
3	31、 26、 (12)	26
4	31、 41、 (12)	31
5	(11)、 41、 (12)	41
6	(11)、 (37)、 (12)	##
7	11、 37、 12	11
8	15、 37、 12	12
9	15、 37、 23	15
10	29、 37、 23	23
11	29、 37	29
12	37	37
13		##

} 第二个初始合并段

4、置换-选择排序

3、长度分析：

设内存可以存放 m 个记录，则首先从文件读入 m 记录到内存。这 m 个记录肯定可以归入本初始合并段内。这样，必须从文件中读入 m 个记录。这 m 个记录中平均有一半可以归入本合并段，一半归入下一合并段。

因此，合并段的平均长度为：

$$m + m/2 + m/4 + \dots = 2m$$

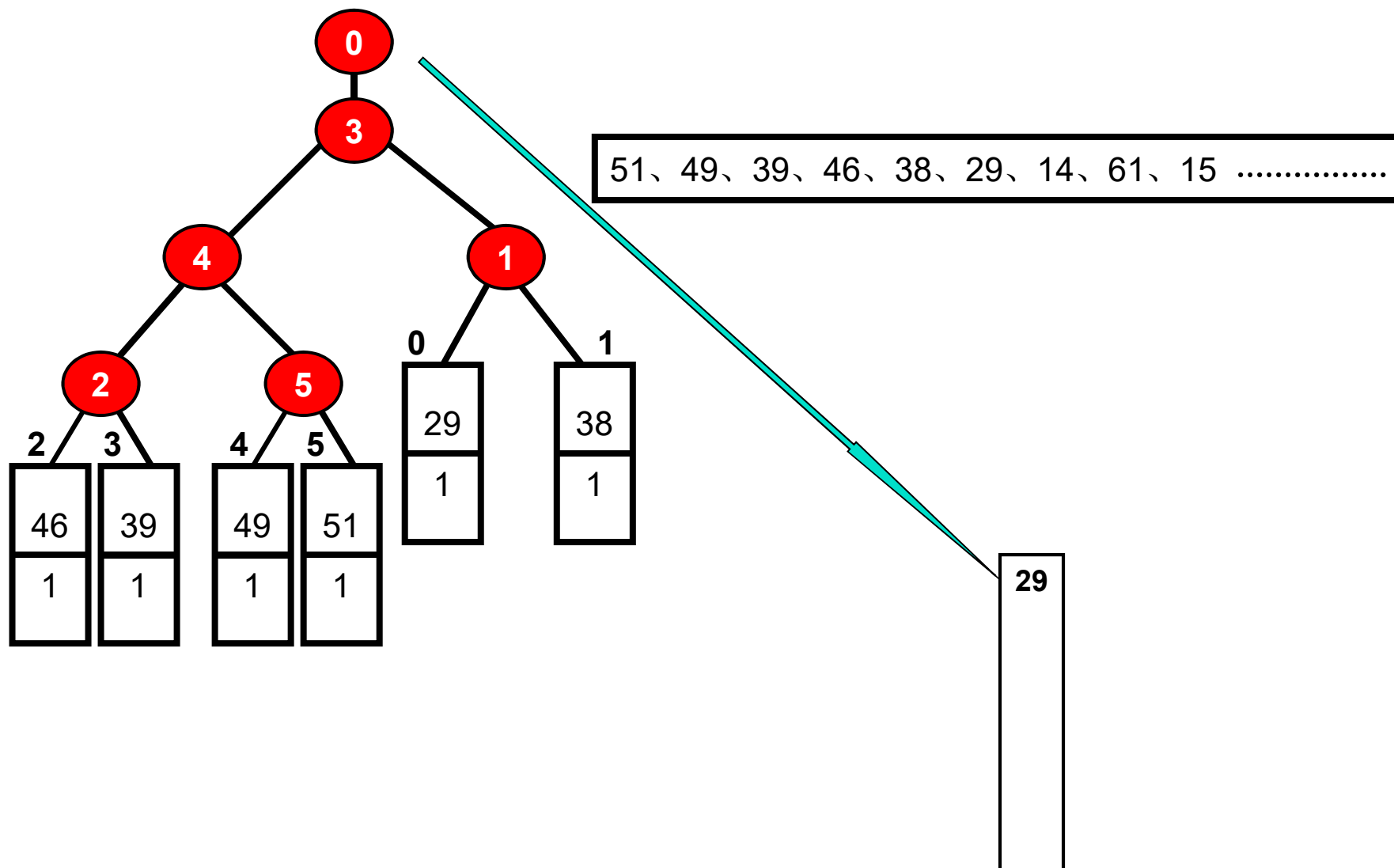
所以，在平均情况下，合并段的平均长度为 $2m$ 。

4、程序实现：

可以用败者树的办法 加以实现。

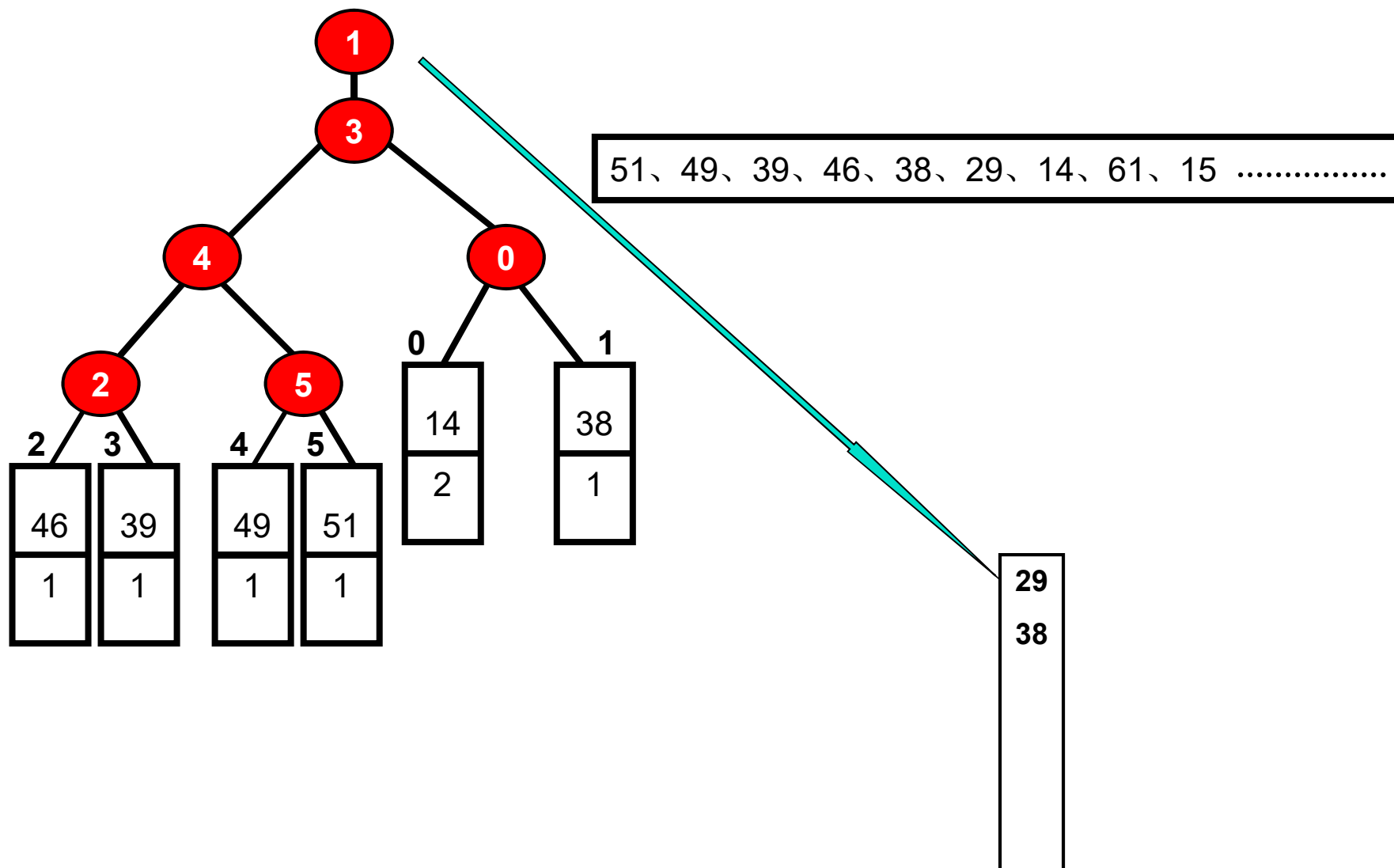
4、置换-选择排序

5、败者树实现置换-选择排序



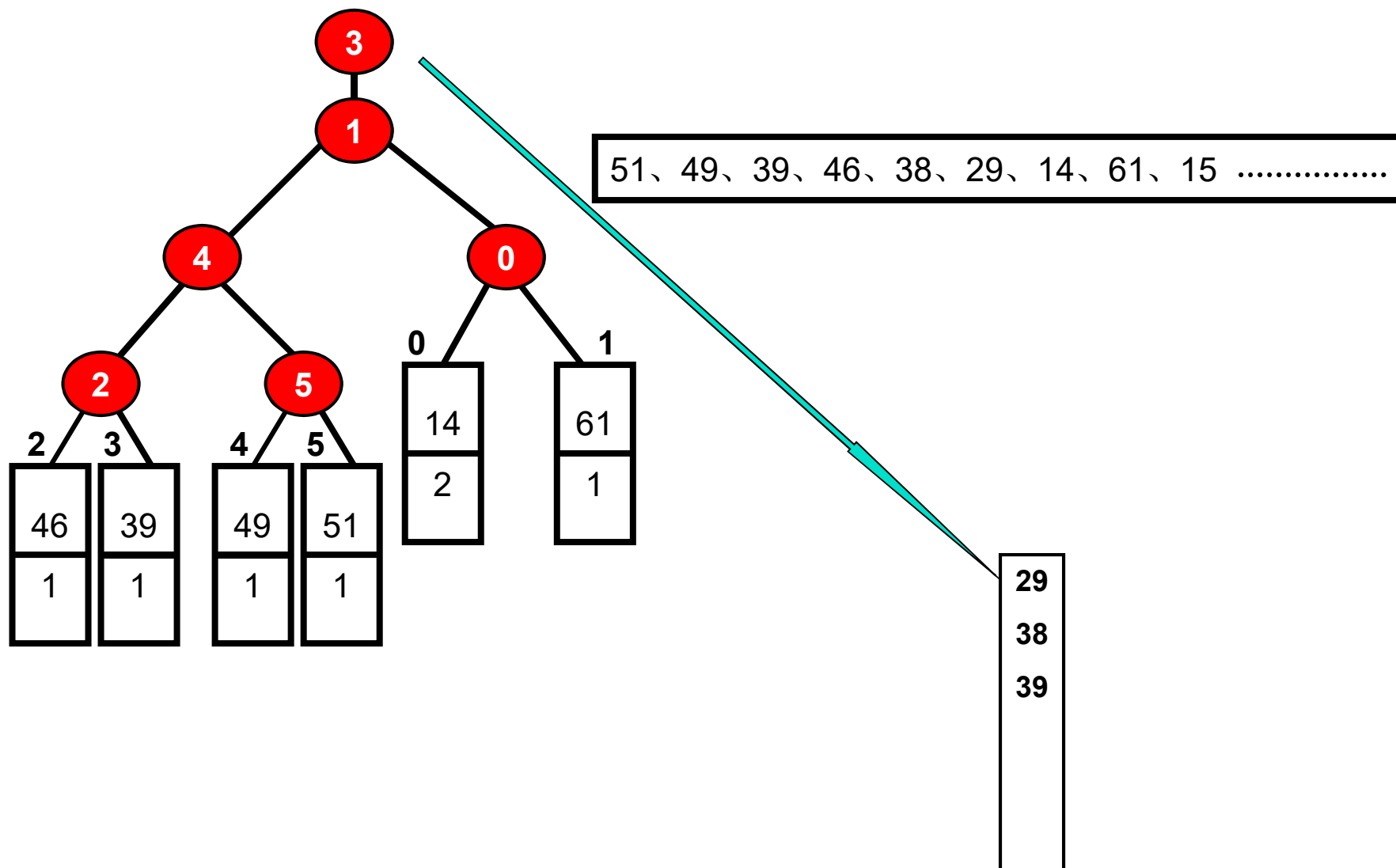
4、置换-选择排序

5、败者树实现置换-选择排序



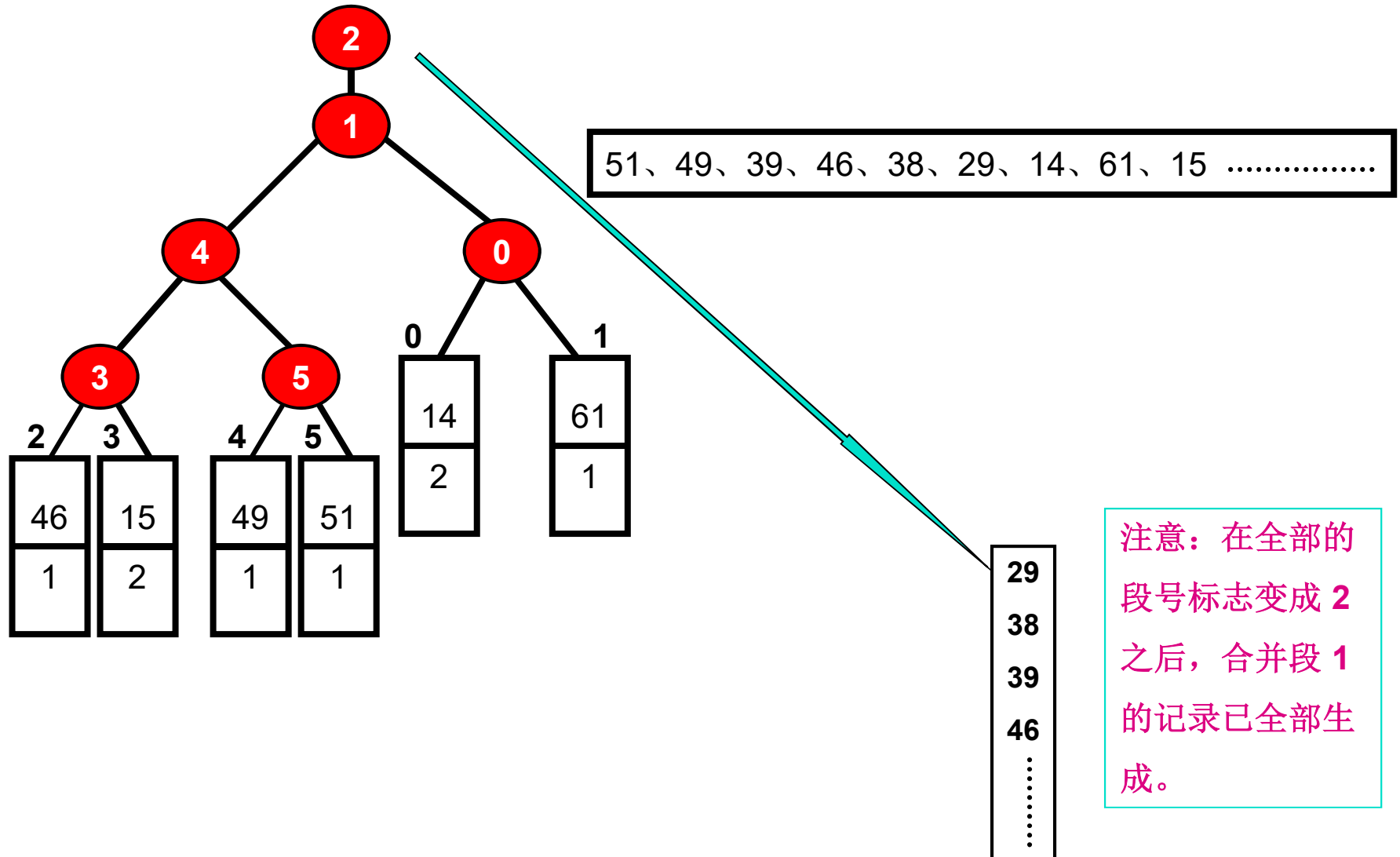
4、置换-选择排序

5、败者树实现置换-选择排序



4、置换-选择排序

5、败者树实现置换-选择排序

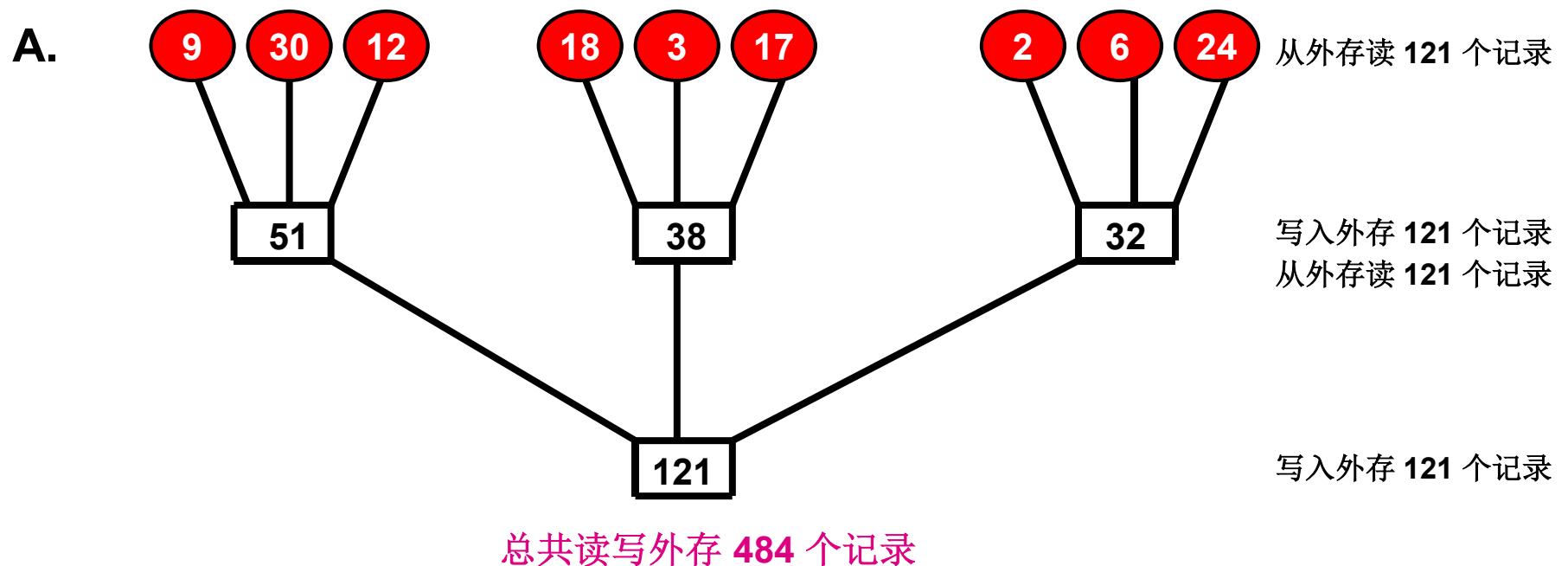


5、最佳归并树

1、最佳归并树

- 起因：由于初始归并段通常不等长。
- 目的：减少读写外存的次数。
- 限制：由于磁带寻找具有最少记录的初始归并段，必须反复倒带。所以，实用性不强。在盘的情况下，需要有段包含的记录数信息、段的位置信息等。文件如集中放置在几个相邻的柱面上的情况比较合适。

e.g:假定由置换-选择分类法生成了 9 个初始归并段，记录数分别为 9、30、12、18、3、17、 2、6、24。如果进行 3-路归并，请讨论在各种情况下的对外存的读写次数。

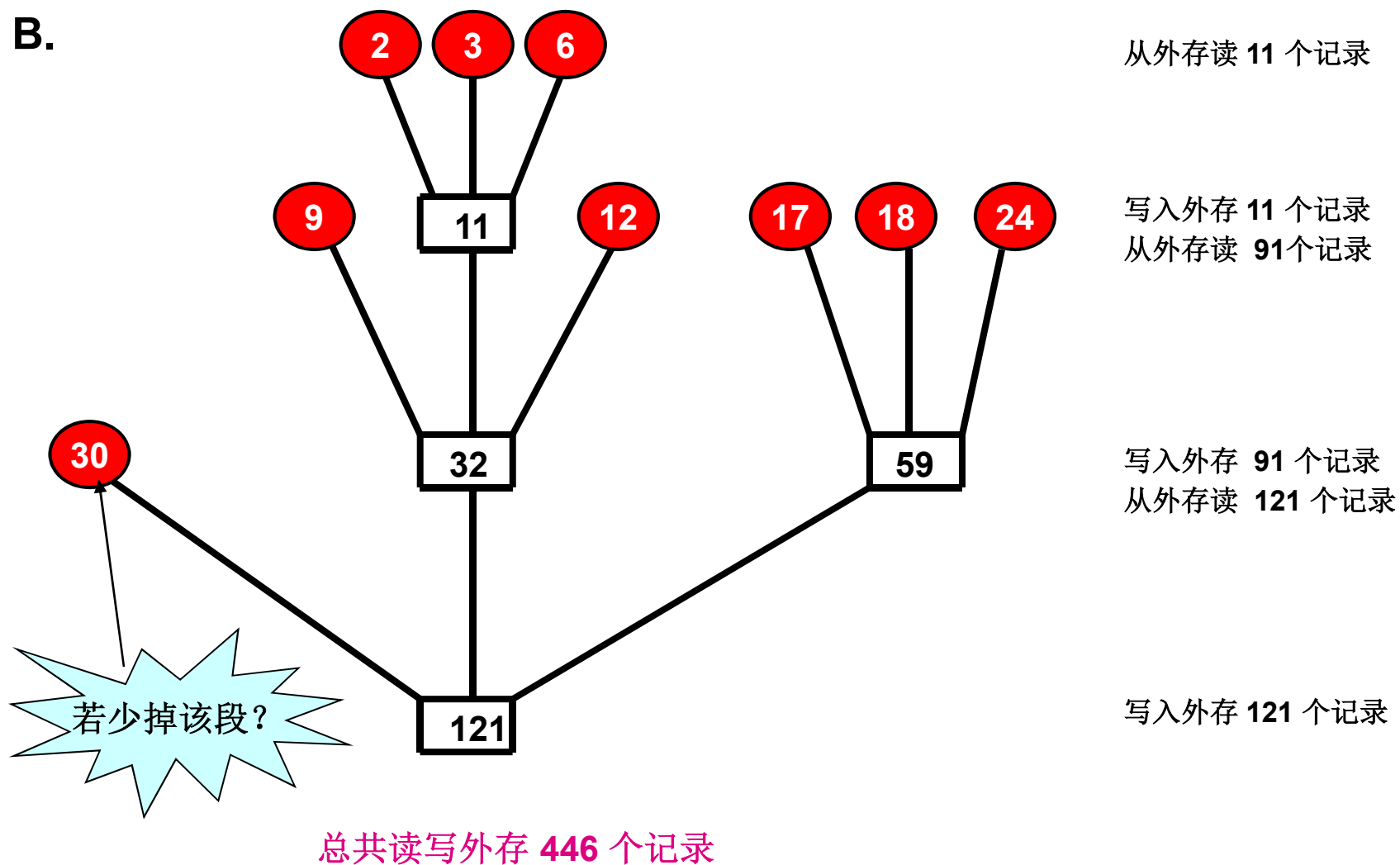


5、最佳归并树

1、最佳归并树

- 按照 **HUFFMAN** 树的思想，记录少的段最先合并。

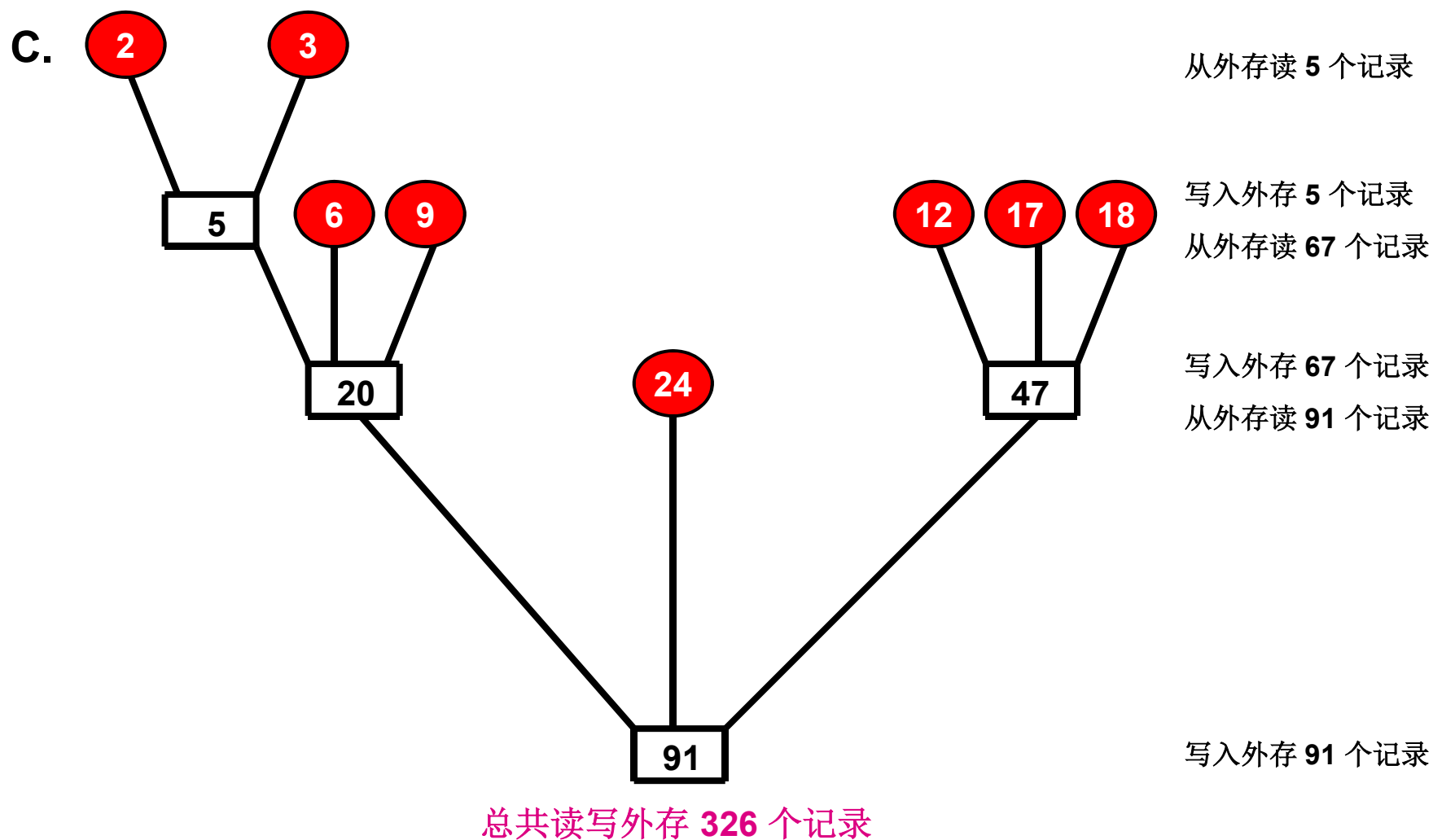
B.



5、最佳归并树

1、最佳归并树

•不够时增加虚段，虚段的关键值可以看作0。如下例所示，第一次只归并两个归并段。



5、最佳归并树

1、最佳归并树

- 虚段的补法:

解：在 K 路平衡归并时，它的归并树的模型是一棵度为 K 的树。在这棵树上的结点要么是叶子，要么是具有 K 个儿子的内部结点。设具有 K 个儿子的内部结点共有 n_k 个。初始归并段的个数为 m 个。

设 $n = n_k + m$ ，有：

从结点出发的枝条，共计有： $K \times n_k$ 根

若从进入结点结点的角度进行考虑，则共有： $n_k + m - 1$

注意：没有枝条进入根结点。

所以， $K \times n_k = n_k + m - 1$

于是： $n_k = (m - 1) / (K - 1)$

这就意味着，若 $(m - 1) \text{ MOD } (K - 1) = 0$ ，无需增加虚段。否则，要增加虚段，其数目为： $(K - 1) - (m - 1) \text{ MOD } (K - 1)$ 。

文件

1、基本概念

1、文件及其类别：

文件的类型：

1。按记录的类型分类

操作系统文件，数据库文件（单关键字文件，多关键字文件）

2。按记录长度分类

定长记录文件，不定长记录文件

1、基本概念

2、基本术语：

- 数据域（数据场）：记录中的每个数据项，称之为域或场（**Field**）
- 关键字：唯一标识记录的域，称之为关键字。辅助关键字，称之为次关键字。
- 记录（**Record**）：若干相关的数据项的集合。如果存之于外存，则叫做记录。
- 文件：记录的集合。
- 记录的物理结构和逻辑结构：

逻辑结构：记录在用户或程序员面前呈现的形式。

物理结构：记录在在物理存储器上的存储方式，是数据的物理表示和组织。

- 物理记录和逻辑记录：

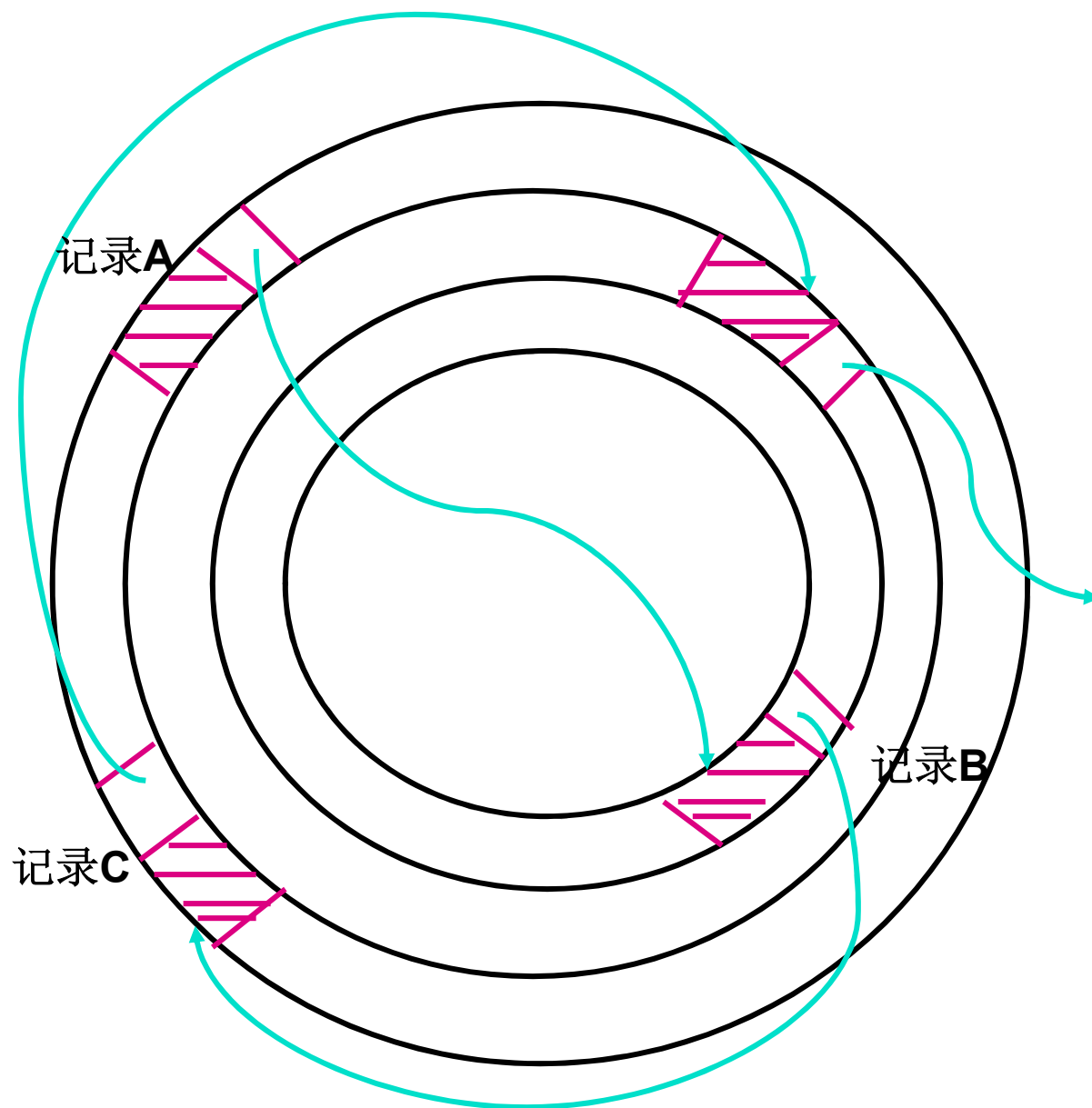
物理记录：计算机用一条 **I/O** 指令进行读写外存的基本单位。通常，对一定的设备和操作系统，大小是固定不变的。

逻辑记录：程序员加以定义，用户要求使用的。

关系： 一对一，一对多，多对一

1、基本概念

记录A
记录B
记录C
记录D
⋮



1、基本概念

3、检索和修改

- 检索：

- 顺序存取：存取下一个逻辑记录

- 直接存取：存取第 i 个逻辑记录

- 按关键字值存取相应的记录：

- 简单询问：查单个记录

- 区域询问：查多个记录

- 函数询问：满足某种条件的记录

- 布尔询问：满足布尔运算组合的询问

- 修改：插入、修改、更新

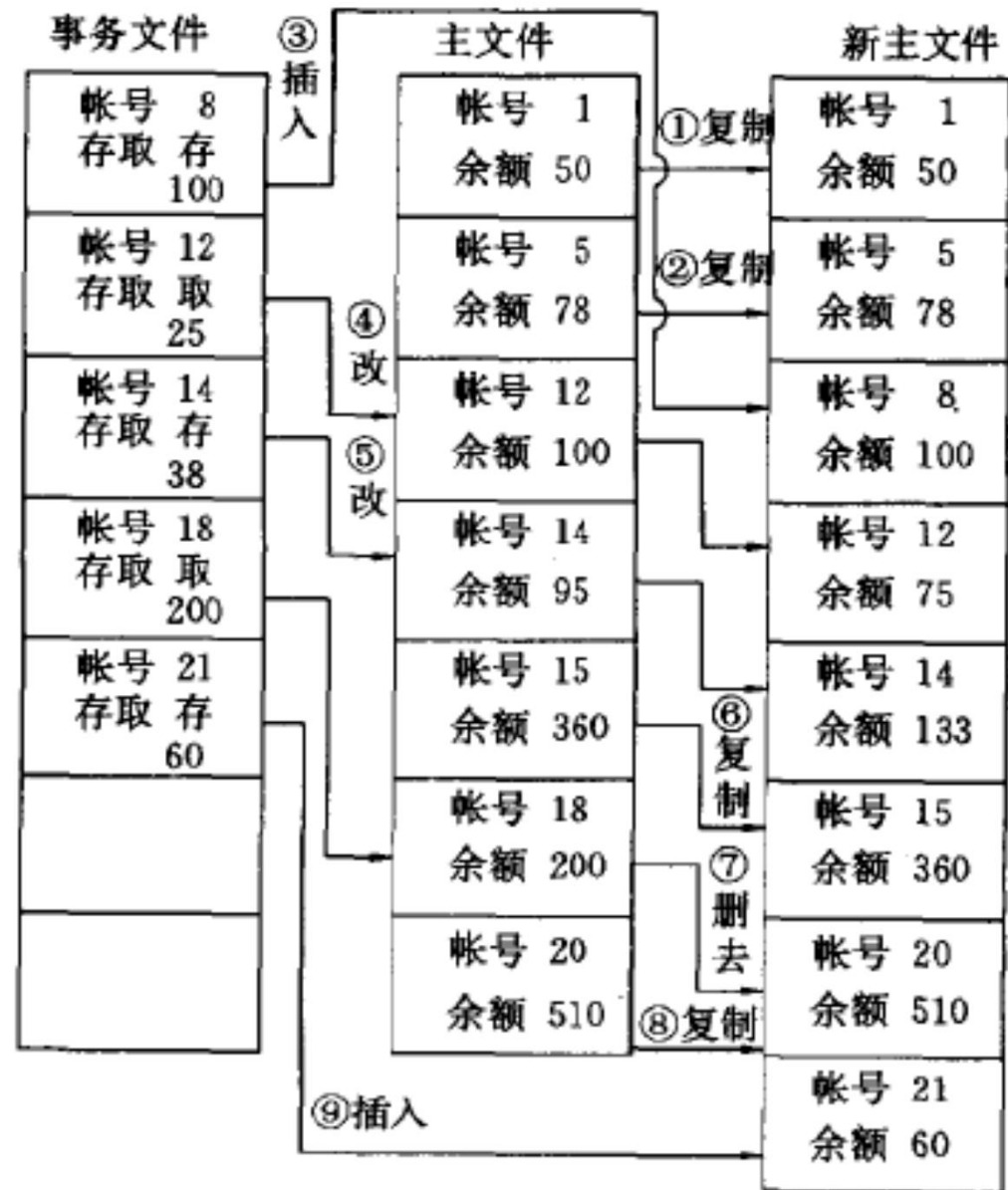
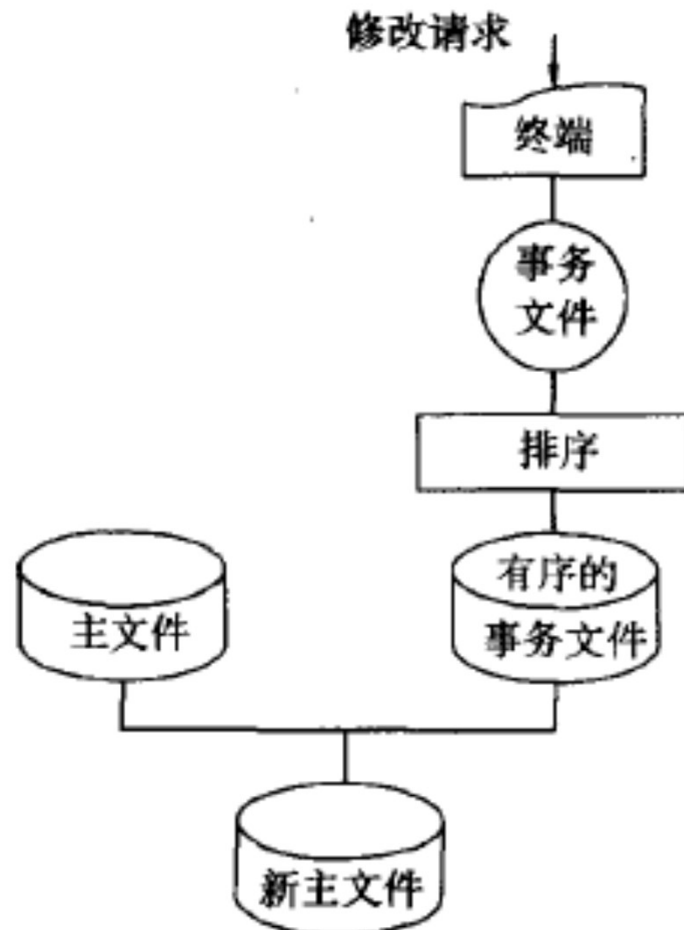
- 更新方式：实时、批量两种方式

2. 顺序文件

- 记录按其在文件中的逻辑顺序依次进入存储介质而建立，即物理记录的顺序和逻辑记录的顺序一致。
- 连续文件：次序相继的两个物理记录在存储介质上的存储位置相邻。
- 串联文件：物理记录之间的次序由指针相链。
- 顺序文件的特点：
 - 1。存取第 i 个记录，必须先搜索前 $i-1$ 个记录
 - 2。新记录只能加在文件的末尾。
 - 3。更新文件记录必须对文件进行复制。

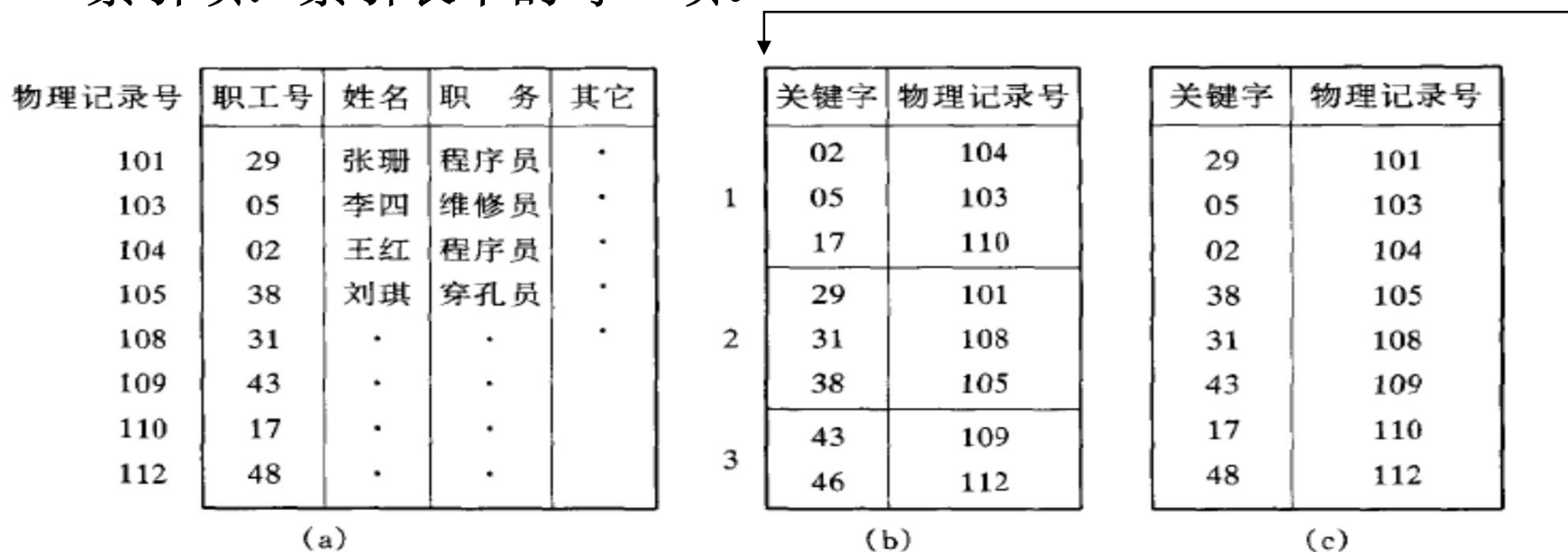
2. 顺序文件

- 典型的顺序文件如磁带文件
- 磁带文件的批处理过程：



3. 索引文件

- 除了文件本身之外，另建立一张指示逻辑记录和物理记录之间的对应关系表。
- 索引项：索引表中的每一项。

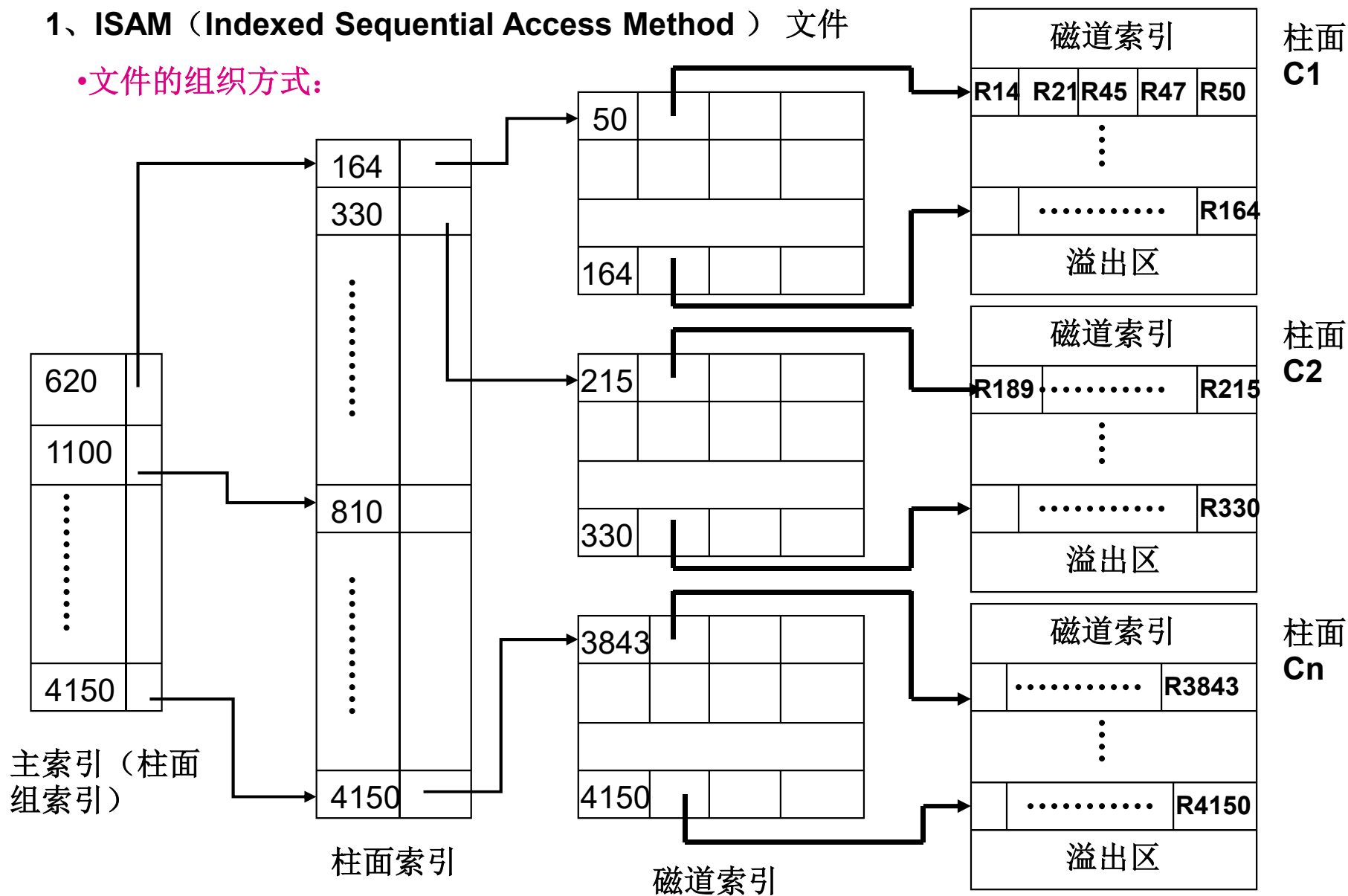


最大关键字	物理块号
17	1
38	2
46	3

4、ISAM文件和VSAM文件

1、ISAM (Indexed Sequential Access Method) 文件

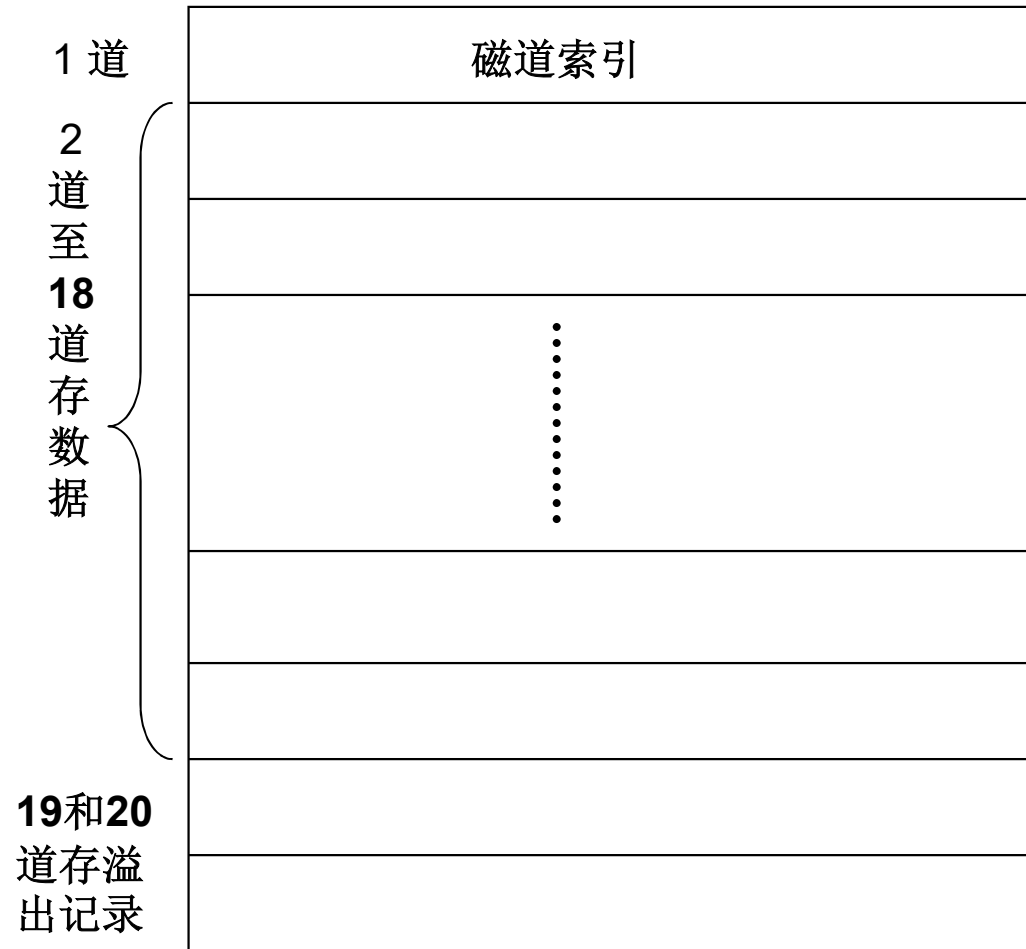
•文件的组织方式:



4、ISAM文件和VSAM文件

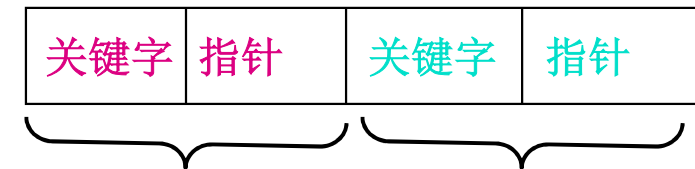
1、ISAM (Indexed Sequential Access Method) 文件

• 柱面的安排: 柱面 C1



溢出区的设置: 1、集中都存在一个公共溢出区内(用多个柱面)
2、分散每个柱面都有溢出区 3、柱面溢出区+公共溢出区

磁道索引项结构



基本索引项

关键字: 本道的最大关键字值

指 针: 本道第一个记录的地址

溢出索引项

关键字: 本道的溢出记录的最大关键字值

指 针: 本道的溢出记录的第一个溢出记录的地址

4、ISAM文件和VSAM文件

- 查找：主索引 → 柱面索引 → 磁道索引 → 磁道的第一个记录
- 插入：R14、R21、R43、R47、R50进入 2 磁道；R60、R70、R80、R85、R90进入3磁道

柱面 C1					
1 道	磁道索引				
2 道	R14	R21	R43	R47	R50
3 道	R60	R70	R80	R85	R90
	⋮				
19和20					
道存溢					
出记录					

2、3 磁道的磁道索引项

50	T2' 1		
90	T3' 1		

4、ISAM文件和VSAM文件

- 查找：主索引 → 柱面索引 → 磁道索引 → 磁道的第一个记录
- 插入：R30 进入 2 磁道之后。

柱面 C1					
1 道	磁道索引				
2 道	R14	R21	R30	R43	R47
3 道	R60	R70	R80	R85	R90
	⋮				
19和20 道存溢出记录	R50	∧			

2、3 磁道的磁道索引项

47	T2' 1	50	T19' 1
90	T3' 1		

4、ISAM文件和VSAM文件

- 查找：主索引 → 柱面索引 → 磁道索引 → 磁道的第一个记录
- 插入：R65 进入 3 磁道之后。

柱面 C1						
1 道 2 道 3 道	磁道索引					
	R14	R21	R30	R43	R47	
	R60	R65	R70	R80	R85	
19和20 道存溢 出记录	⋮					
	R50	∧	R90	∧		

2、3 磁道的磁道索引项

47	T2' 1	50	T19' 1
85	T3' 1	90	T19' 2

4、ISAM文件和VSAM文件

- 查找：主索引 → 柱面索引 → 磁道索引 → 磁道的第一个记录
- 插入：R48 进入 2 磁道之后。因 $50 > 48 > 47$ ，只能进入本道溢出区。若同时插入多个记录进入 溢出区，注意排成递减序进入溢出区。

柱面 C1

1 道	磁道索引									
2 道	R14	R21	R30	R43	R47					
3 道	R60	R65	R70	R80	R85					
	⋮									
19和20 道存溢 出记录	R50	∧	R90	∧	R48					

2、3 磁道的磁道索引项

47	T2' 1	50	T19' 3
85	T3' 1	90	T19' 2

4、ISAM文件和VSAM文件

- 删除：给被删除记录作一标记。优点：不必移动记录变更指针。缺点：周期性整理 ISAM 文件。如删除 R21。

		柱面 C1									
1 道 2 道 3 道	磁道索引										
	R14		R21		R30		R43		R47		
	R60		R65		R70		R80		R85		
	⋮										
19和20 道存溢 出记录											
	R50		^		R90		^		R48		

2、3 磁道的磁道索引项

47	T2' 1	50	T19' 3
85	T3' 1	90	T19' 2

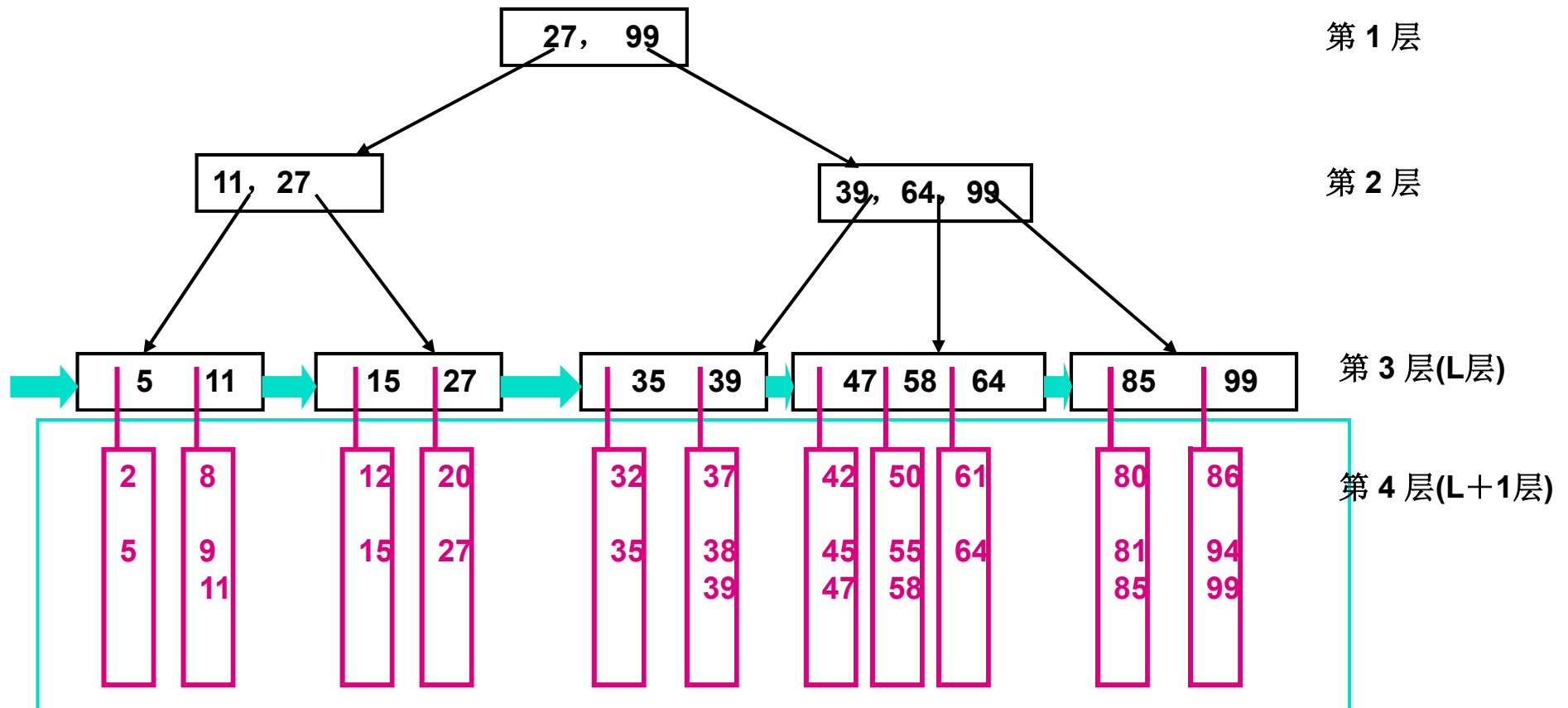
4、ISAM文件和VSAM文件

2、B+ 树

- 定义： **m** 阶 **B+** 树满足或空，或：
 - A**、根结点要么是叶子，要么至少有两个儿子
 - B**、除根结点和叶子结点之外，每个结点的儿子个数为：
 $\lceil m/2 \rceil \leq s \leq m$
 - C**、有 **k** 个儿子的非叶结点具有 **k** 个关键字
 - D**、叶子结点可以认为是外部结点，保存信息。而非叶结点可以认为是索引部分，结点中含其子树中的最大或最小结点关键字。
 - E**、叶子结点的上层结点包含关键字信息和指向相应记录的指针，且按关键字顺序相链结。

4、ISAM文件和VSAM文件

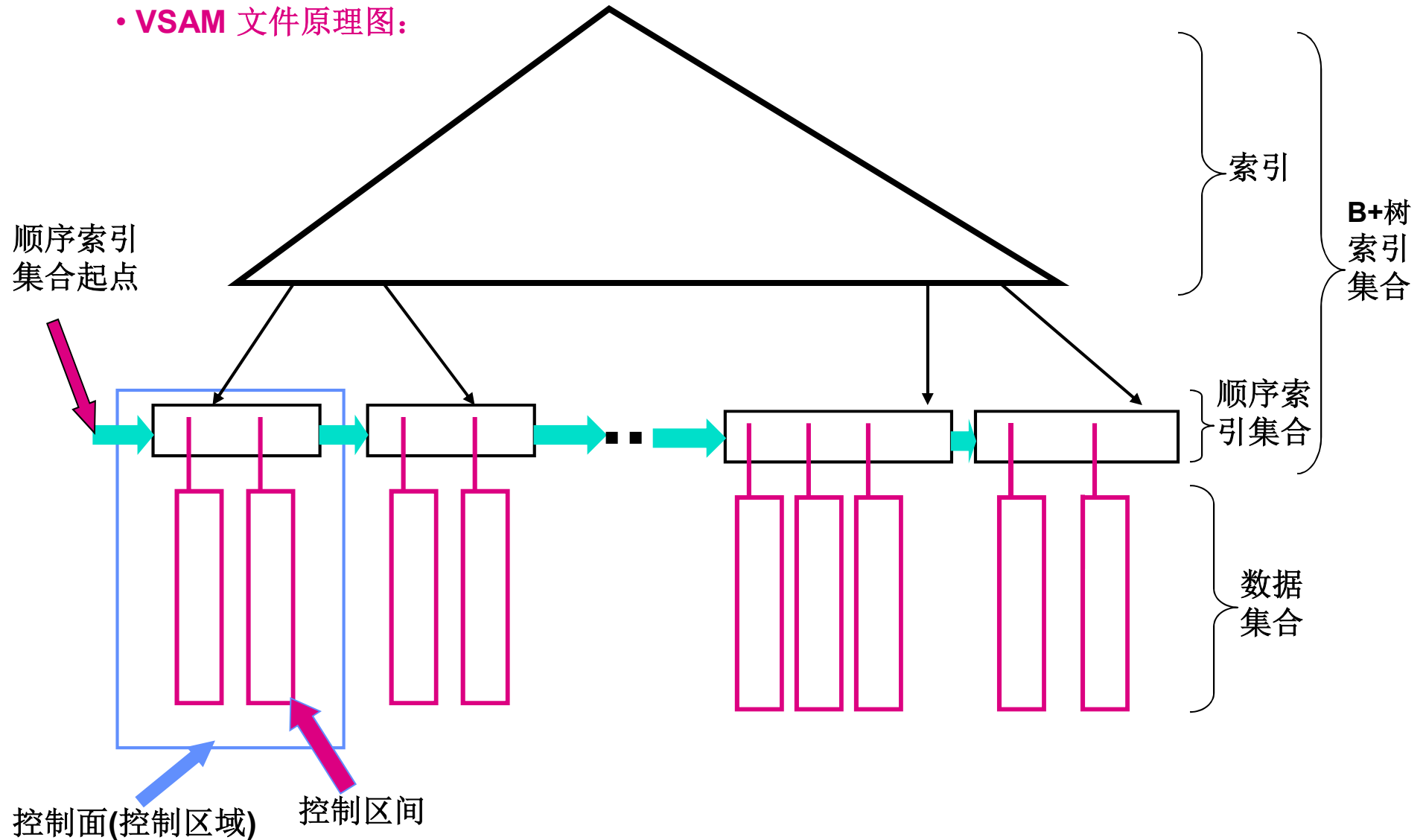
例如： $m = 3$ 阶 B+ 树。除根结点和叶子结点之外，每个结点的儿子个数至少为 $\lceil m/2 \rceil = 2$ 个；结点的关键字个数至少为 1。该 B+ 树的深度为 4。叶子结点都在第 4 层上。



4、ISAM文件和VSAM文件

3、VSAM（Virtual Storage Access Method）文件

• VSAM 文件原理图：



4、ISAM文件和VSAM文件

3、VSAM（Virtual Storage Access Method）文件

- **VSAM** 文件组织方式:

控制区间：数据集合中的一个结点，是 **VSAM** 中的基本单位（**like** 磁道 **in ISAM**）。

控制面：多个控制区间和它们的顺序索引集合中的结点形成控制面（**like** 柱面 **in ISAM**）

记录信息的控制：记录可以不定长。

记录1	记录2	记录n	记录n控制信息	记录n控制信息	控制区间的控制信息
-----	-----	-------	-----	-------	---------	-------	---------	-----------

溢出区的处理：取消溢出区。控制区间留有空间。控制面留有全空的控制区间。

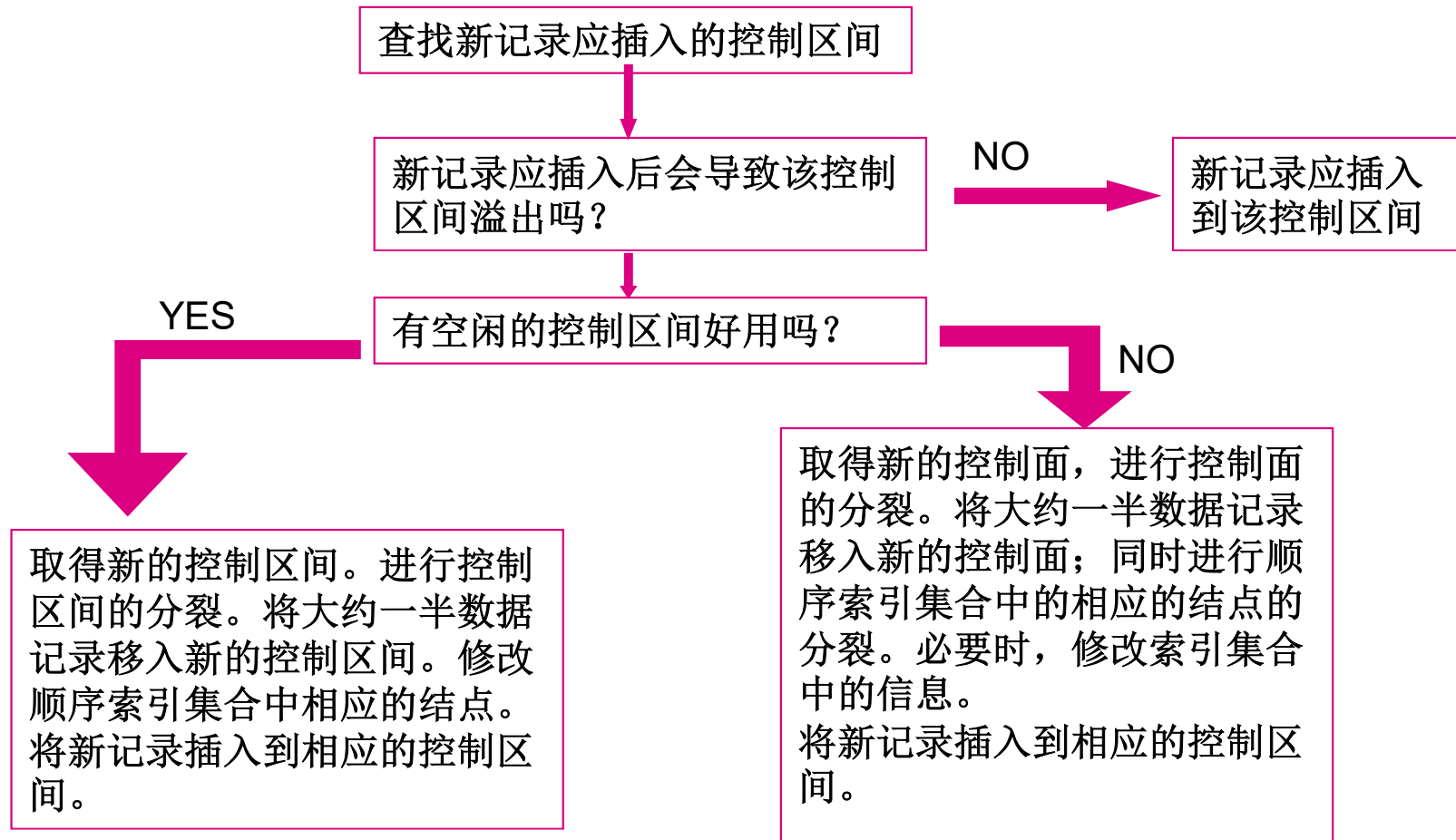
- 插入：新记录进入控制区间，必须保证有序。

插入流程图：

4、ISAM文件和VSAM文件

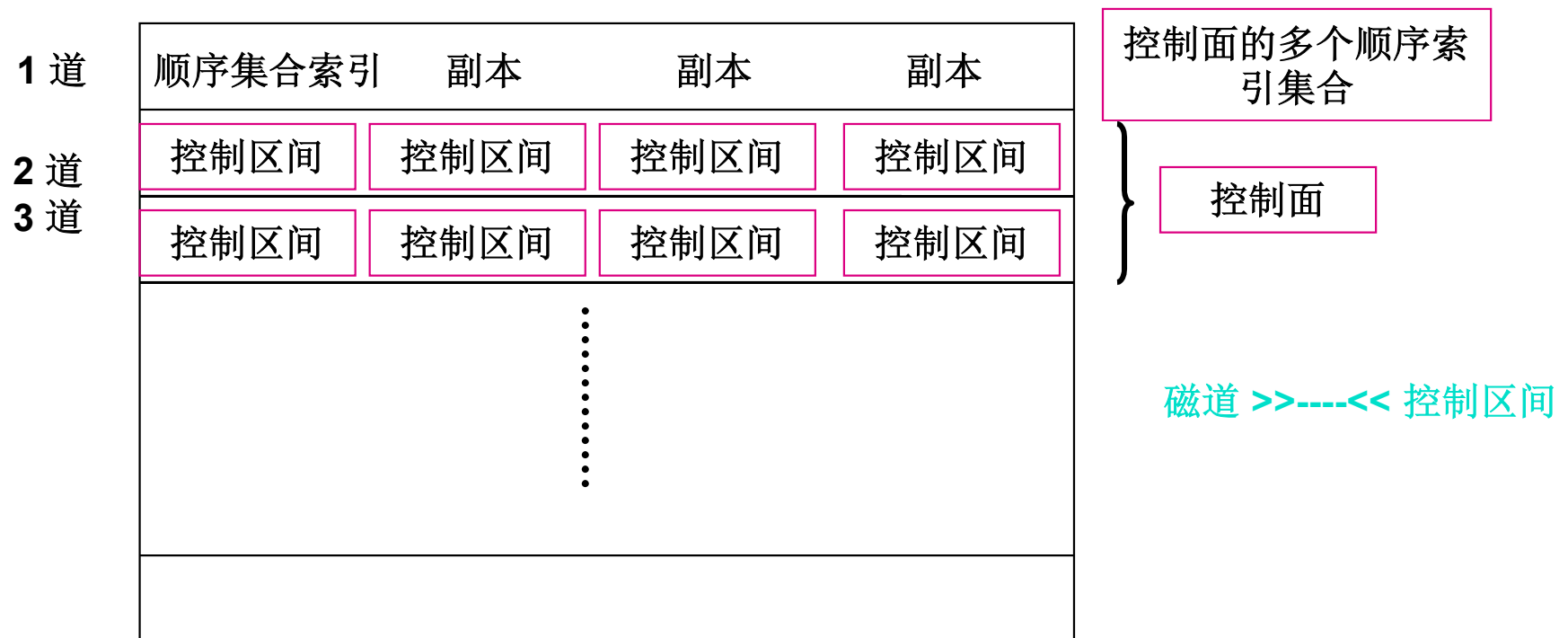
- 插入： 新记录进入控制区间，必须保证有序。

插入流程图：



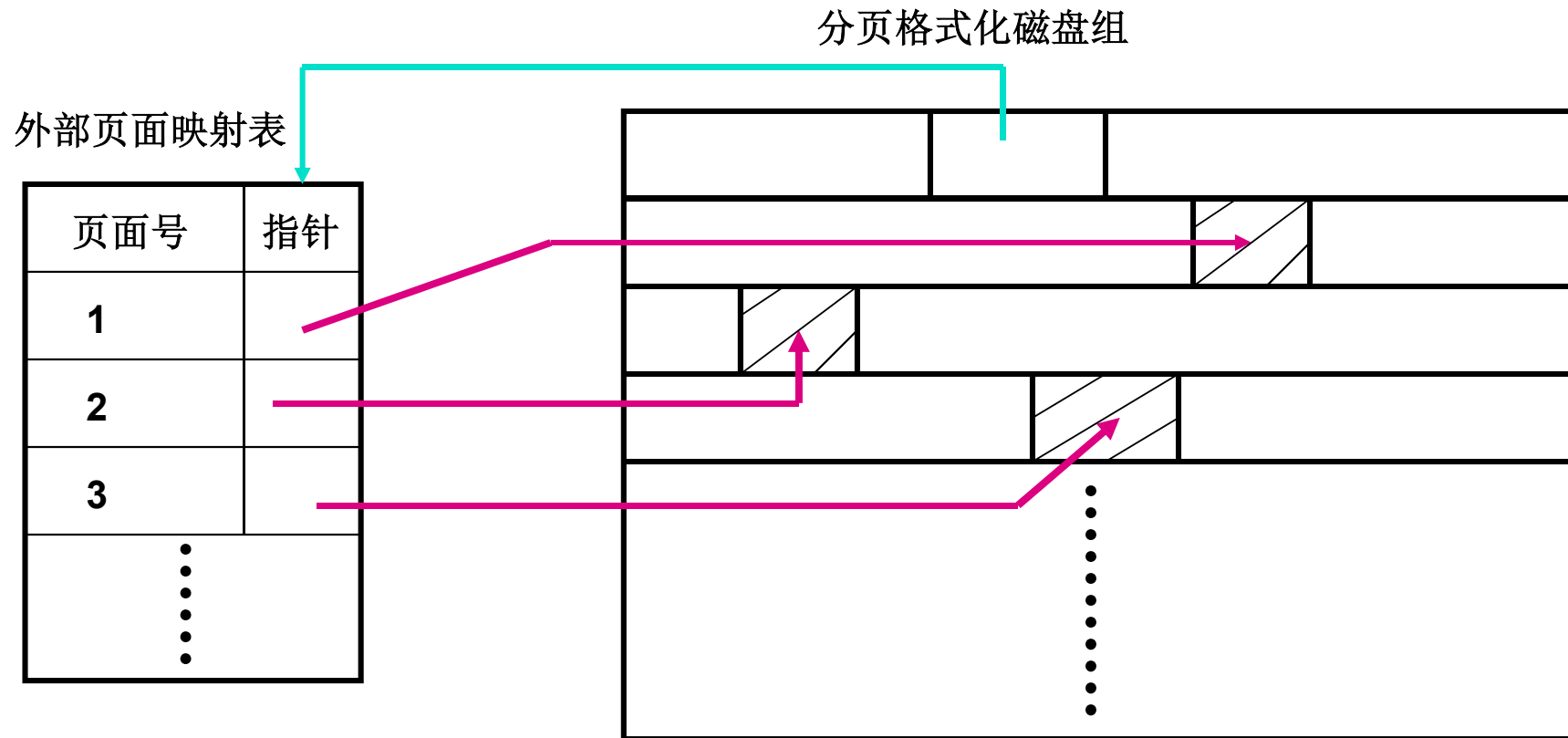
4、ISAM文件和VSAM文件

- 删除：后面的结点向前移动把空间留给新插入的记录。控制区间变空，修改相应的顺序索引集合中的索引项。
- 优缺点：动态分配于释放存储，不许对文件进行重组，插入、查找快。空间利用率低，通常只有 **50% ~ 70%**。
- 加速的考虑：设置多个副本的顺序集合索引。



4、ISAM文件和VSAM文件

- 为什么称之为 **VSAM** 文件：虚拟存储，同机器、设备无关。

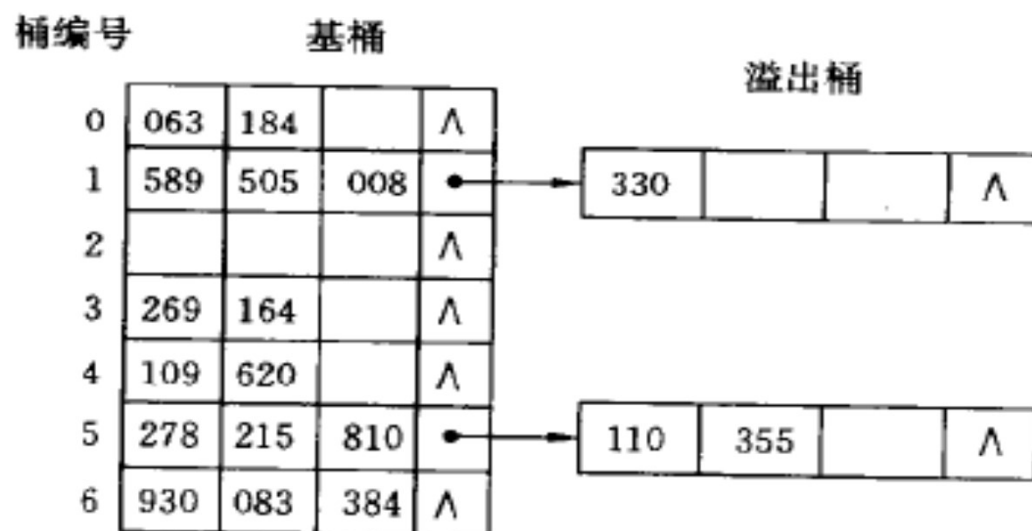


- 分页：把内外存按同样的大小进行分页。使文件技术不依赖于具体的外部设备。更换外设时，不必改变整个程序，只需改变外部页面映射表中的相对地址到绝对地址的计算程序即可。
- 控制区间 = 若干页面。控制面 = 若干控制区间。外存是内存的延伸，虚拟存储器。**VSAM** 可脱离具体的外存组织文件，所以称之为 **VSAM** 文件。

5. 直接存取文件（散列文件）

- 与**Hash**表的建立方式类似
- 优点：随机存放，记录不需要进行排序；插入、删除方便，存取速度快，不需要索引区。
- 缺点：不能进行顺序存放，经过多次的插入、删除后，容易造成文件结构不合理，溢出桶满而基桶内多数为被删除的记录。

18个记录，关键字分别为：278，109，063，930，589，184，505，269，008，083，164，215，330，810，620，110，384，355



6. 多关键字文件

- 对文件进行检索时，不仅对主关键字进行简单访问，还经常需要对次关键字进行其他类型的检索。
- 两种多关键字文件的组织方法：多重表文件，倒排文件

物理

记录号	姓 名	学 号		专 业		已修学分		选 修 课 目					
01	王 雯	1350	02	软件	02	412	03	丙	02	丁	03	丁	05
02	马小燕	1351	03	软件	07	398	07	甲	04	丙	03		
03	阮 森	1352	04	计算机	05	436	∧	乙	05	丙	04		
04	苏明明	1353	∧	应用	06	402	08	甲	06	丙	08		
05	田 永	1354	06	计算机	∧	384	02	乙	07	丁	09		
06	杨 青	1355	07	应用	09	356	10	甲	07				
07	薛平平	1356	08	软件	08	398	∧	甲	08	乙	∧		
08	崔子健	1357	∧	软件	∧	408	01	甲	09	丙	∧		
09	王 洪	1358	10	应用	10	370	05	甲	10	丁	∧		
10	刘 倩	1359	∧	应用	∧	364	09	甲	∧				

6. 多关键字文件

1、多重表文件

主关键字建立索引，次关键字也建立索引，具有同一次关键字的记录构成一个链表。

特点：

易于构造和修改，但删除记录比较繁琐，需要在每个次关键字的链表中进行删除。

主关键字	头指针
1353	01
1357	05
1359	09

(b)

次关键字	头指针	长度
350~399	06	6
400~449	04	4

(d)

次关键字	头指针	长度
软 件	01	4
计 算 机	03	2
应 用	04	4

(c)

次关键字	头指针	长度
甲	02	7
乙	03	3
丙	01	5
丁	01	4

(e)

6. 多关键字文件

1、倒排文件

次关键字的索引结构与多重表文件不同，具有相同次关键字的记录间不设指针相链，而在倒排表中该次关键字的一项中存放记录的物理记录号。

特点：

检索记录较快。

维护困难，不同关键字其记录数不同，造成倒排表的长度不等，同一倒排表中各项长度也不等。

软 件	01, 02, 07, 08
计 算 机	03, 05
应 用	04, 06, 09, 10

(a)专业倒排表

350~399	02, 05, 06, 07, 09, 10
400~449	01, 03, 04, 08

(b)已修学分倒排表

甲	02, 04, 06, 07, 08, 09, 10
乙	03, 05, 07,
丙	01, 02, 03, 04, 08
丁	01, 03, 05, 09



本章小结

✓ 熟练掌握：

- 外部存储的概念，各种外部排序的思想和算法。
充分了解各种排序算法的应用背景和优缺点。
- 文件的种类和基本概念。

✓ 重点学习：

- 胜者树、败者树和最佳归并树，与内部排序的联系，加强在实际应用中的训练。