数据结构与算法

第9章 文件管理和外排序

主讲：赵海燕
北京大学信息科学技术学院
“数据结构与算法”教学组

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6, “十一五”国家级规划教材

Why External Sorting?

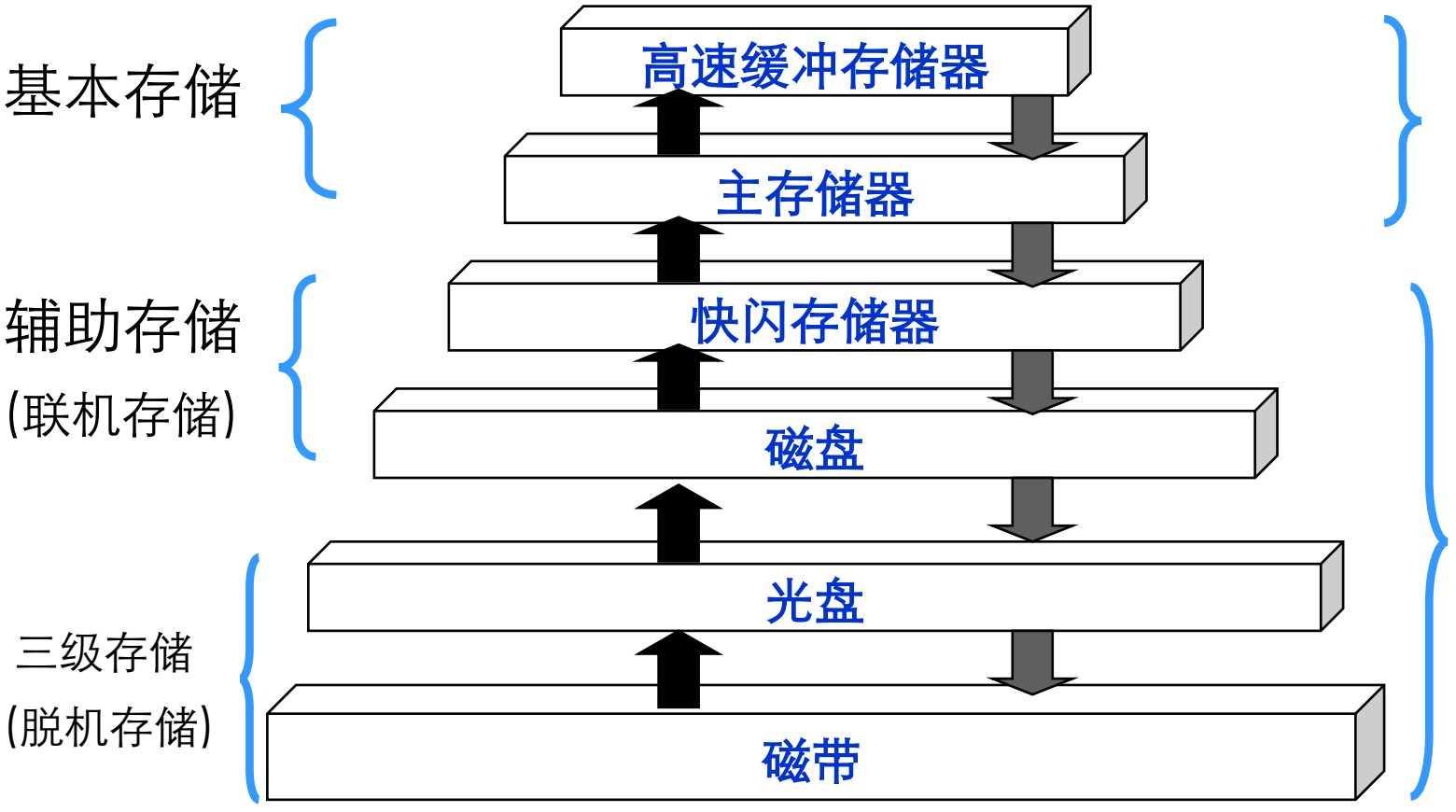
- Internal sorting
  - e.g. *Quick Sort, Heap Sort, Insertion Sort,...*
  - Very efficient but *all data needs to fit completely* into main memory
- Huge data
  - Performing sorting operations on ***amounts of data*** that are *too large to fit into main memory*
  - ***Can not*** be done ***in one step***

Why File Management?

- 内存：有限的存储资源
- 海量数据的排序
  - 不可能一次性装载到内存
  - 需外部存储器（外存）的支持，进行（多次的）内外存数据交换才能完成

- 文件结构
  - 外存中存储的数据的结构
  - 提高文件存储效率和运算效率

物理存储介质概览


基本存储
辅助存储
(联机存储)
三级存储
(脱机存储)
高速缓冲存储器
主存储器
快闪存储器
磁盘
光盘
磁带
易失性
非易失性存储

物理存储介质概览

- KB (kilo byte) 10^3B （页块)
- MB (mega byte) 10^6B (高速缓存)
- GB (giga) 10^9B (内存、硬盘)
- TB (tera) 10^{12}B (磁盘阵列)
- PB (peta) 10^{15}B (磁带库)
- EB = 10^{18}B； ZB = 10^{21}B； YB = 10^{24}B
- Googol 是 10 的 100 次方

主存储器和外存储器

- 主存储器 （primary / main memory， 简称“内存”），具有速度快、造价高、容量小等特点
  - 随机访问存储器 （Random Access Memory, 即RAM）
  - 高速缓存 （cache）
  - 视频存储器（video memory ）
- 外存储器（peripheral /secondary storage，简称“外存”），具有价低、容量大的优点
  - 硬盘
  - 软盘 、磁带

内存的优缺点

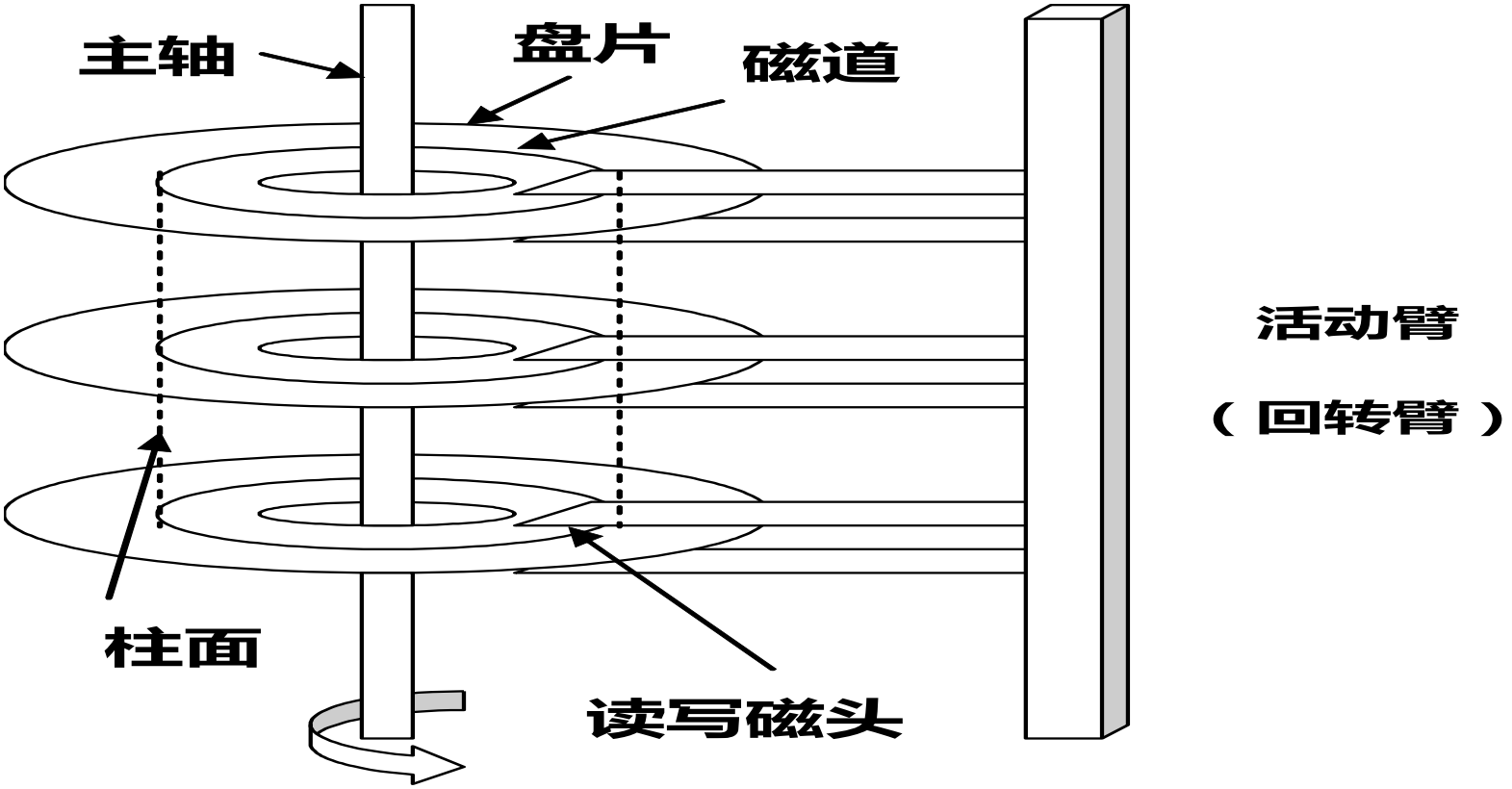
- 直接与CPU沟通，访问存储在内存地址的数据所需时间可看作是一个很小的常数

- 优点
  - 访问速度快
- 缺点
  - 造价高、存储容量小和断电后丢失数据

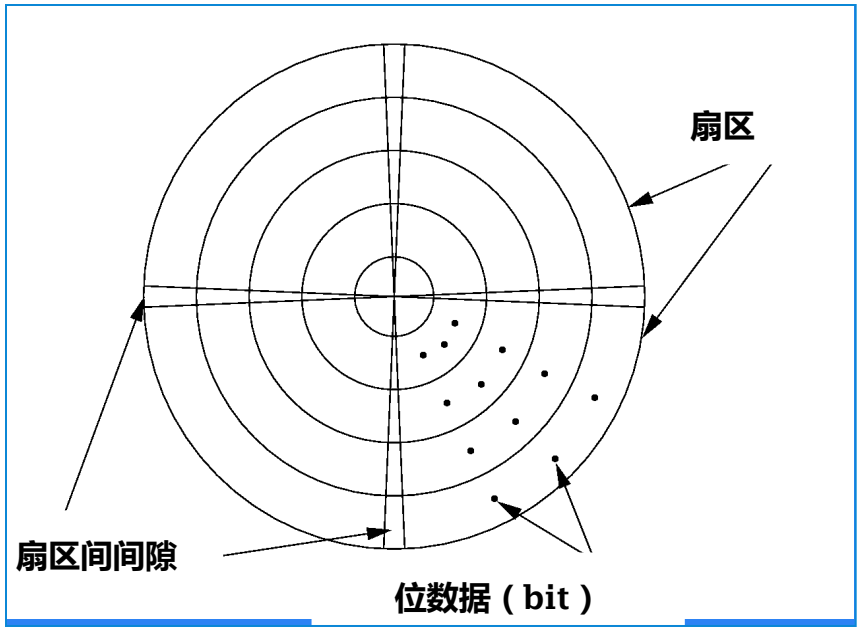
外存的优缺点

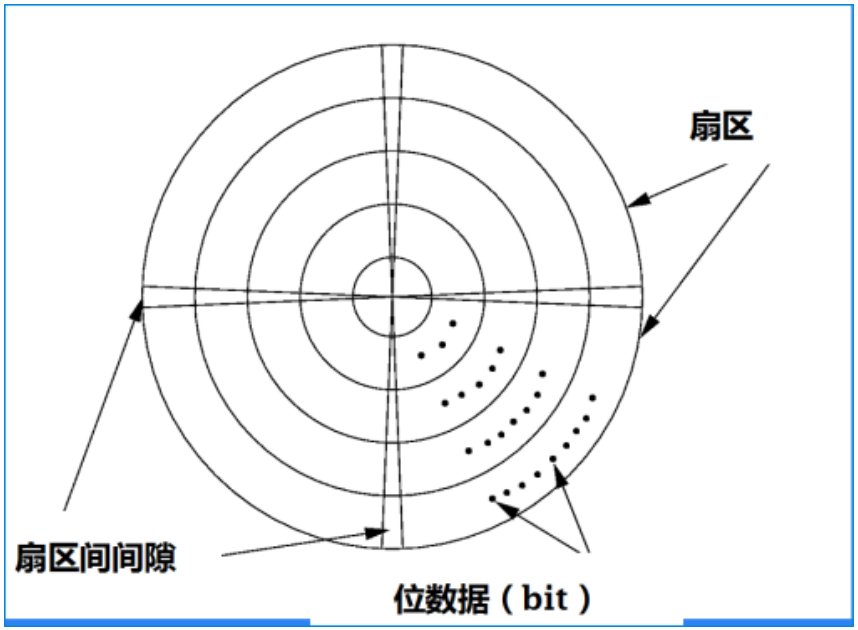
- 外存上数据需要装载到内存中才能被处理
- 优点：永久存储能力、便携性 、价廉
- 缺点：存取速度慢导致访问时间长
  - 访问磁盘中的数据比访问内存慢五六个数量级
  - 内存访问存取时间的单位是纳秒（10^{-9}秒），而外存一次访问时间则以毫秒（10^{-3}秒）或秒为数量级
- 讨论在外存的数据结构及其上的操作时，须遵循下述原则：
  - **尽量减少访外次数**

磁盘的物理结构

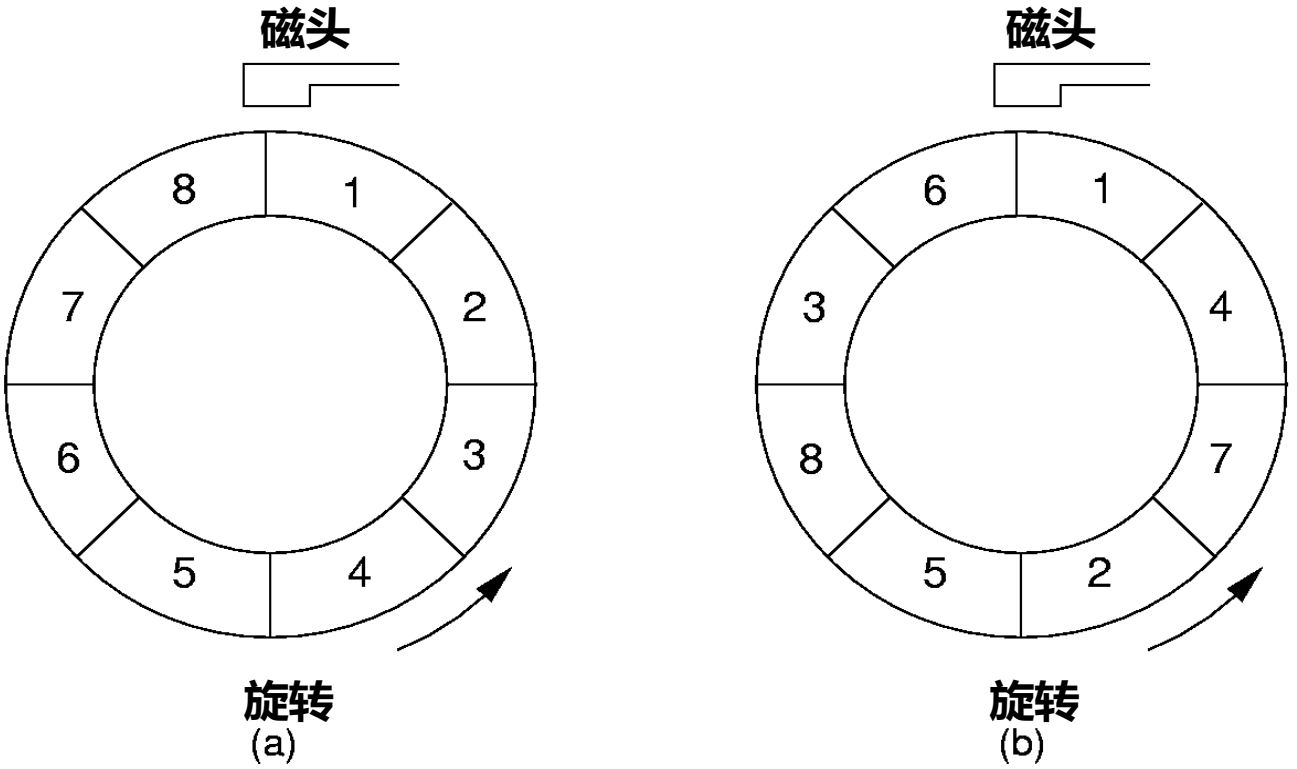


磁盘盘片的组织





磁盘磁道的组织（交错法）



（a）没有扇区交错；　　（b）以 3 为交错因子

磁盘存取步骤

- 选定某个盘片组: 电子
- 选定某个柱面： **机械**
  - 把磁头移动到该柱面，这个移动过程称为寻道（seek）
- 确定磁道： 电子
- 确定所要读写的数据在磁盘上的准确位置： **机械**
  - 这段时间一般称为旋转延迟（rotational delay 或者 rotational latency）
- 真正进行读写

磁盘访问时间估算

- 磁盘访问时间主要由寻道时间、旋转延迟时间和数据传输时间组成
  - 寻道时间： 移动磁盘臂，定位到正确磁道所需的时间
  - 旋转延迟时间： 等待被存取的扇区出现在读写头下所需的时间

外存数据访问方式

- 外存空间被划分成长度固定的存储块，称为页块(page)，每页包含一定数量的数据单元
  - 作为外存的基本存储单位，数据以页块为单位进行存取，这样可以减少外存的定位次数，从而减少外存读写的时间耗费
  - 页块大小因OS异而异， 常见：512B ，1024B，2048B，4096B， ……

缓冲区和缓冲池

- 目的：减少磁盘访问次数
- 方法：缓冲（buffering ）或缓存（caching）
  - 在内存中保留／缓冲尽可能多的块
  - 增加待访问的块在内存中找到的机会
- 缓冲区合起来称为缓冲池(buffer pool）
  - 存储在一个缓冲区中的信息常称为一个页块(page ），往往是一次 I/O 的量

替换缓冲区块的策略

- 新的页块申请缓冲区时，释放最近**最不可能被再次引用** 的缓冲区来存放新页
  - “先进先出”（FIFO）
  - “最不频繁使用”（LFU）
  - “最近最少使用”（LRU）

toDo

1. 查询当前主流硬盘的性能指标
   - 容量（G）
   - 磁盘旋转速度（rpm）
   - 交错因子
   - 寻道时间
   - 旋转延迟时间

关键码排序

- 一般而言，一条记录可能很大（可能上百个字节），而其关键码一般很小（只占几个字节）
  - 例如，学生管理条目可能存储几百个字节的信息，包括姓名、学号、地址、选课记录、成绩等；而排序码可能只是占几个字节的学号

- **方法1**：读入所有的记录，在内存排序后，再把排好序的记录写回磁盘，即每当需要处理的时候读取整个记录
  I/O太多！
- **方法2**：只读出关键码、位置的指针。关键码排好序后，可以对原文件的记录重新排列，或是不排（诸如数据库，利用索引）

关键码索引排序

- **索引文件**(index file)
  - 存储**关键码** 与 **指针**
  - 指针标识相应记录在原数据文件中的位置

- 对索引文件进行排序，减少所需的I/O操作
  - 索引文件**远小于**整个数据文件
  - 一个索引文件只针对关键码排序(key sorting)

外排序

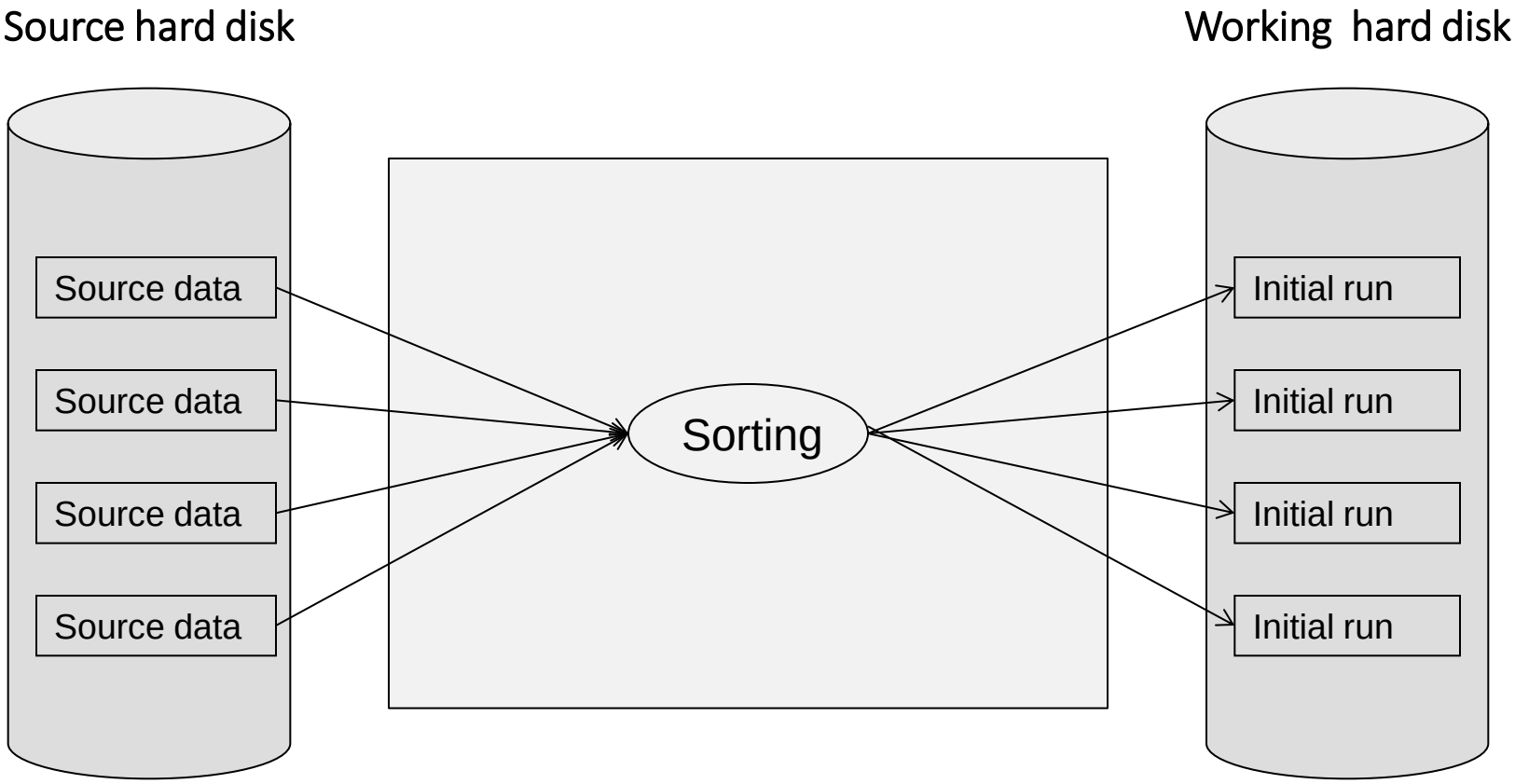
- 也称**磁盘文件排序**： 外存设备上（文件）的排序技术（数据存放在外存文件中），需要考虑外存特点，通常由**两个** 相对独立的阶段 组成
  - 根据内存的大小，将外存中的数据文件划分成**若干段**，每次将其中一段读入内存并采用内排序方法进行排序：这些已排序的段或有序的子文件称为**顺串**或**归并段**(run)
  - 处理顺串，最后形成对整个数据文件的排列文件（**归并**）

- 外排序算法的主要目标
  - 尽量**减少**读写磁盘的次数

外排序的计算模型

- 假设所有处理均在一个磁盘中完成
  - 一个基本的I/O操作读入一个数据页块到主存储器的缓冲区内，或从主存写回一块数据到磁盘
  - 在好的情况（组成整个文件的各个块按一定的顺序存储在磁盘中）下，从文件中顺序读取块比随机读取块更有效
- 基本目标为减少I/O操作

Internal Sorting & External Merging



Steps for External Sorting

1. Split the *data* into *pieces* that fit into main memory
2. Sort the *pieces* with conventional sorting algorithms into *runs*
3. Merge those so called *runs and build the completely sorted data-set*

Internal Sorting &

External Merging

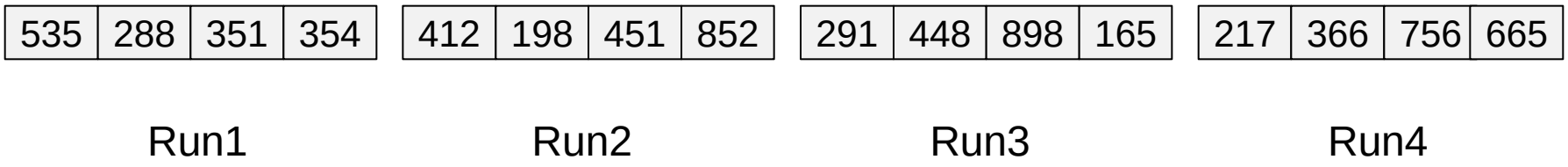
外排序：Step1

Unsorted data-set：

535	288	351	354	412	198	451	852	291	448	898	165	217	366	756	665

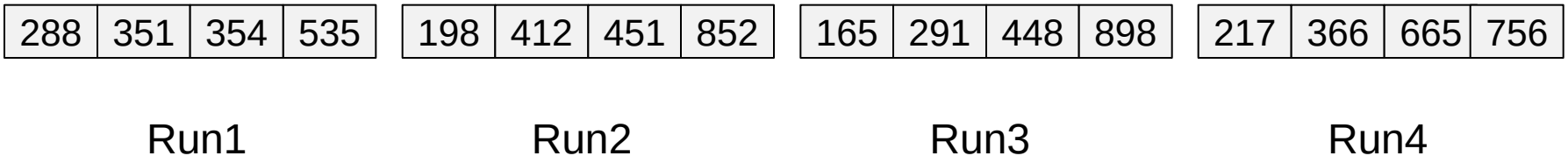
Creation of *initial runs*：

Supposed that 4 elements each fit into main memory

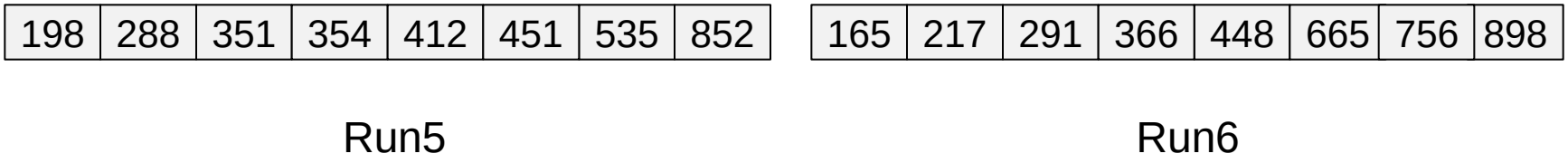


外排序: Step2

Initial runs:



Merging of initial runs:



外排序: Step3

Merged runs:

198	288	351	354	412	451	535	852

Run5

165	217	291	366	448	665	756	898

Run6

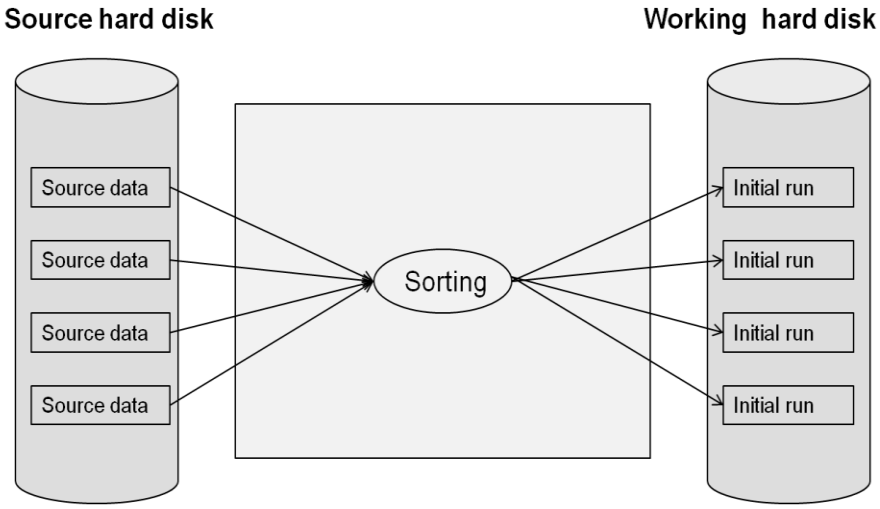
Re-merging:

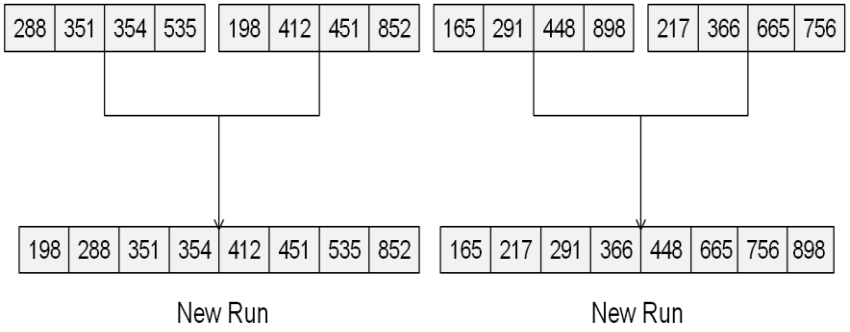
165	198	217	288	291	351	354	366	412	448	451	535	665	756	852	898

Result: After two merge procedures, formerly unsorted set is in **perfect order**

Balanced n-way-merge

The procedure is called **Balanced** **2-way-merging**





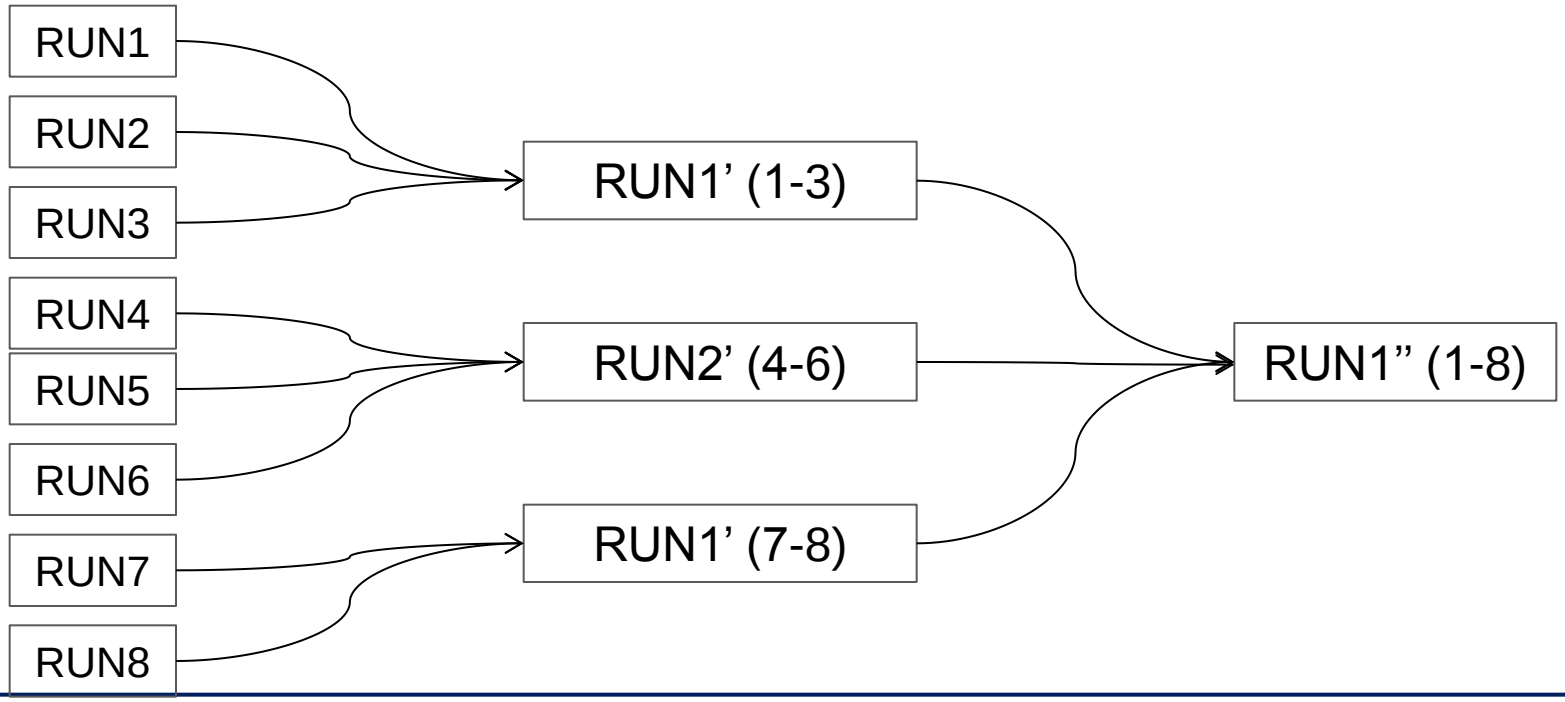
Balanced: As well source as working space is required

2-way: Out of 2 merged runs one new run is formed

Balanced n-way-merge

The merging-procedure can be certainly applied to more than two runs at each time： termed as *n-way-merge* or *multi-way merge*.

A balanced 3-way merge would be as follows:



外排序分析

- 通常，两个相对独立的阶段组成：
  - 先将文件形成尽可能长的初始顺串
  - 逐趟归并顺串，最后形成对整个数据文件的排列文件
- 所需时间由三部分组成：
  - 内部排序所需时间
  - 外存信息读写所需时间
  - 内部归并所需时间
- 关键
  - 减少外存信息的读写次数
  - 由于外排序必须在内存、外存间不断地传送数据，而外存的读写速度与内存相比要慢得多

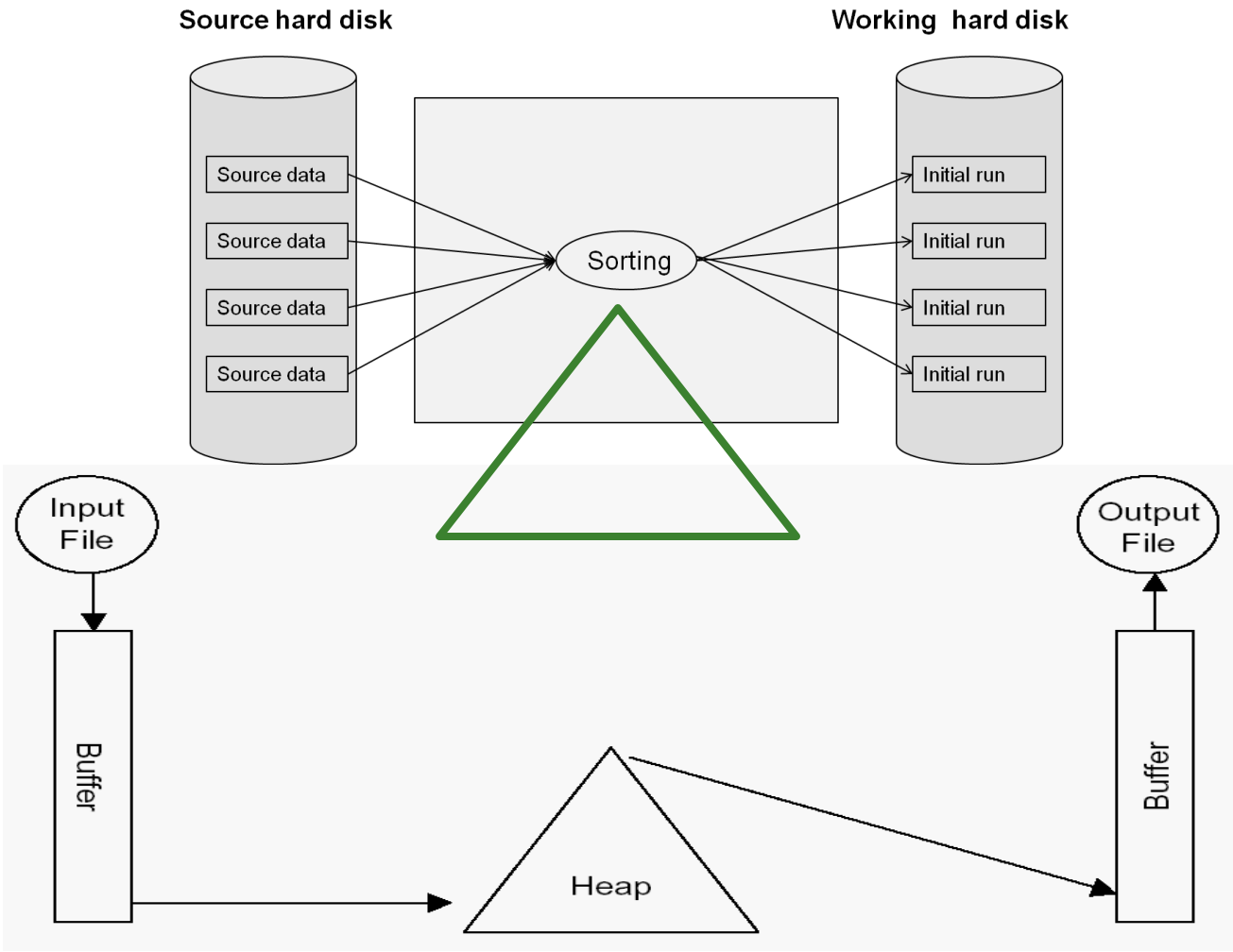
外排序分析

- 对同一个文件而言，进行外排序所需读写外存的次数与归并趟数有关系
- 假设有m个初始顺串，每次对 k 个顺串进行归并，归并趟数为$[\log_k m]$
- 为了减少归并趟数，可以从**两个方面**着手：
  - **减少**初始顺串的个数m
  - **增加**归并的顺串数量k
    - 多个磁盘

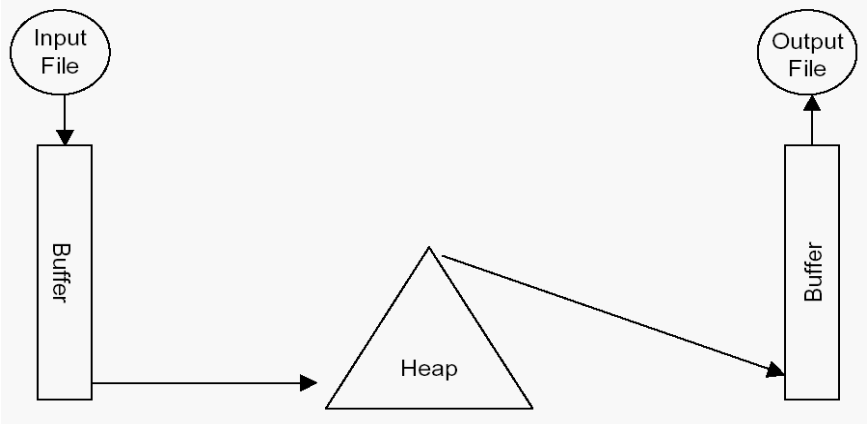
置换选择排序

- 在基本I/O基础上，尽可能产生大顺串的技术
  - 一个基本的I/O操作是读／写一块
- 堆排序的扩展
  - 内存组织：一个堆、一个输入缓冲区、一个输出缓冲区

置换选择排序


Source hard disk
Working hard disk
Source data
Source data
Source data
Source data
Sorting
Initial run
Initial run
Initial run
Initial run
Input File
Output File
Buffer
Heap
Buffer

置换选择方法



- 处理过程
  - 从**输入文件**读取一个页块（的所有记录），进入**输入缓冲区**；
  - 待排序记录放入**RAM（组织为堆）**；
  - 记录被处理后，写回到**输出缓冲区**；输出缓冲区写满的时候，把整个缓冲区写回到**磁盘块/输出文件**中；
  - 当**输入缓冲区**为空时，从**磁盘输入文件**读取下一块记录

置换选择算法

1. 初始化堆
   (a) 从输入缓冲区读M个记录放到数组RAM中
   (b) 设置堆末尾标准 LAST = M - 1
   (c) 建立一个最小值堆
2. 重复以下步骤，直到堆为空（LAST < 0）：
   (a) 把具有最小排序值的记录（根结点）送到输出缓冲区
   (b) 设R是输入缓冲区中的下一条记录。如果R的排序码值大于刚刚输出的关键码值……
      i. 那么把R放到根结点
      ii. 否则使用数组中LAST位置的记录代替根结点，然后把R放到LAST位置。设置LAST = LAST - 1
   (c) 重新排列堆，筛出根结点

置换选择示例

输出16插入29

重新排列堆

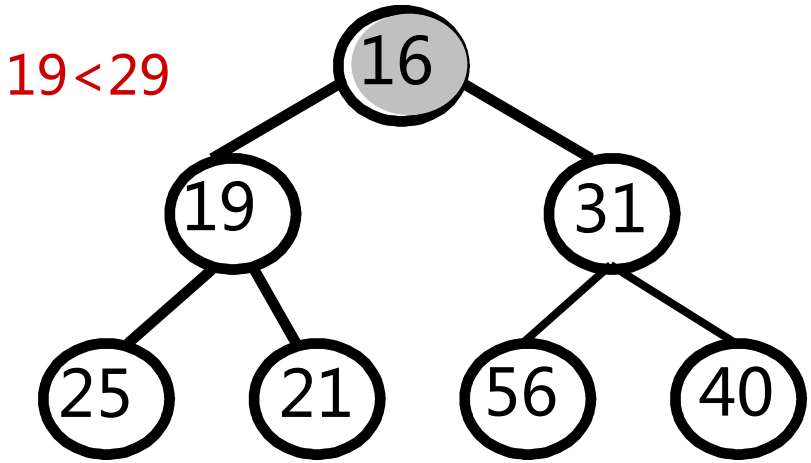
>16

输入

存储

输出

29
14
35
13



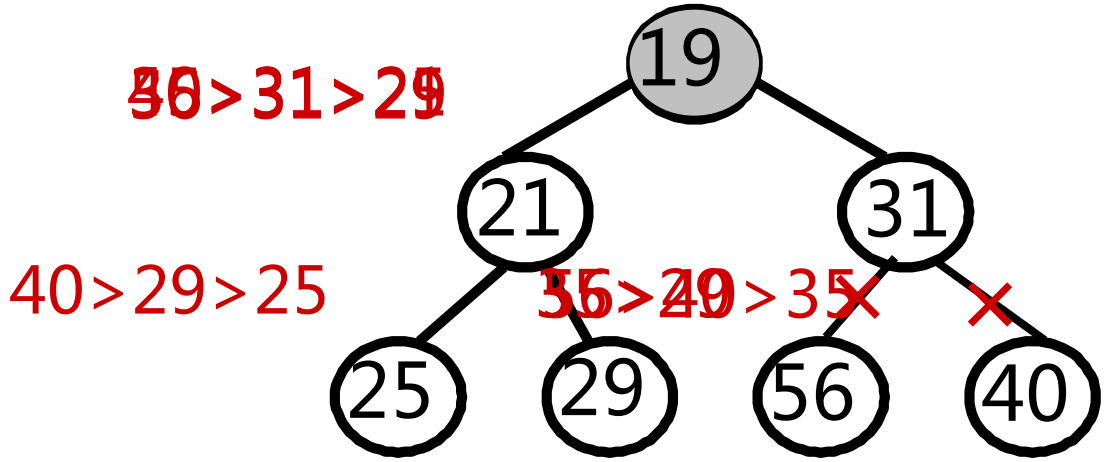
12

21<25<29

置换选择示例

输入 存储 输出

14
35
13



16
12

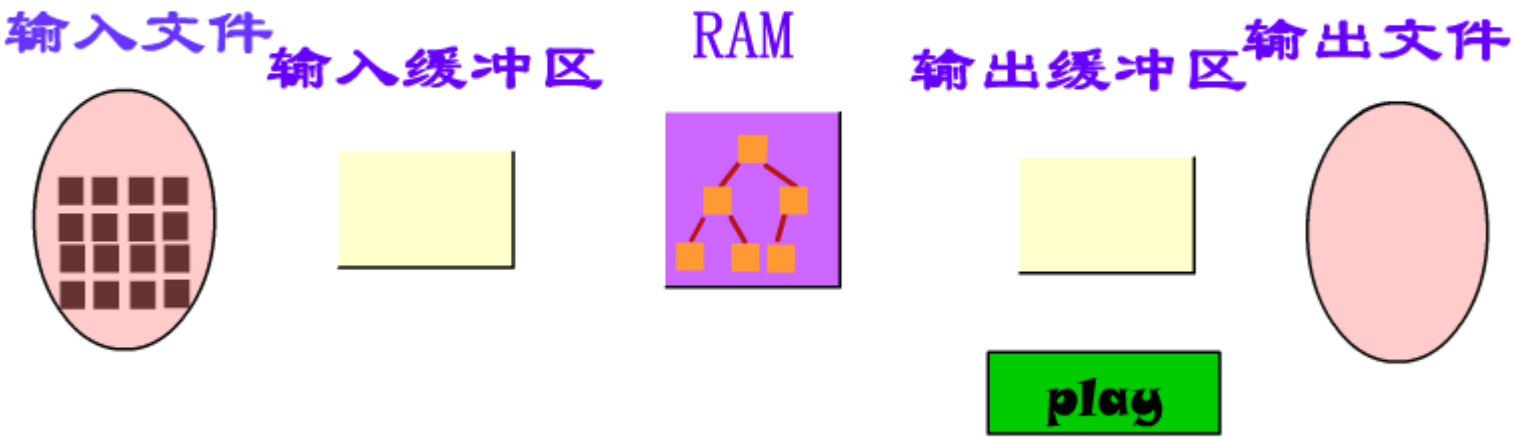
置换选择算法的实现

```
// 模板参数 Elem 代表数组中每一个元素的类型
// A 是从外存读入 n 个元素后所存放的数组
// in 和 out 分别是输入和输出文件名
template <class Elem>
void replacementSelection(Elem * A, int n, const char * in, const char * out) {
    Elem mval;                                  // 存放最小值堆的最小值
    Elem r;                                     // 存放从输入缓冲区中读入的元素
    FILE * inputFile;                           // 输入、输出文件句柄
    FILE * outputFile;
    Buffer<Elem> input;                         // 输入、输出buffer
    Buffer<Elem> output;                        // 初始化输入输出文件
    initFiles(inputFile, outputFile, in, out);
    initMinHeapArry(inputFile, n, A);           // 建堆
    MinHeap<Elem> H(A, n, n);
    initInputBuffer(input, inputFile);
```

置换选择算法的实现

```
    for (int last = (n-1); last >= 0;) {
        mval = H.heapArray[0];                          // 最小值
        sendToOutputBuffer(input, output, inputFile, outputFile, mval);
        input.read(r);                                  // 从输入缓冲区读入一个记录
        if (!less(r, mval))
            H.heapArray[0] = r;                         // r放到根结点
        else {                                          // last代替根结点，r放到last位置
            H.heapArray[0] = H.heapArray[last];
            H.heapArray[last] = r;
            H.setSize(last--);
        }
        H.SiftDown(0);                                  // 调整根结点
    }                                                   // endfor
    endUp(output, inputFile, outputFile);
}
```

置换选择排序



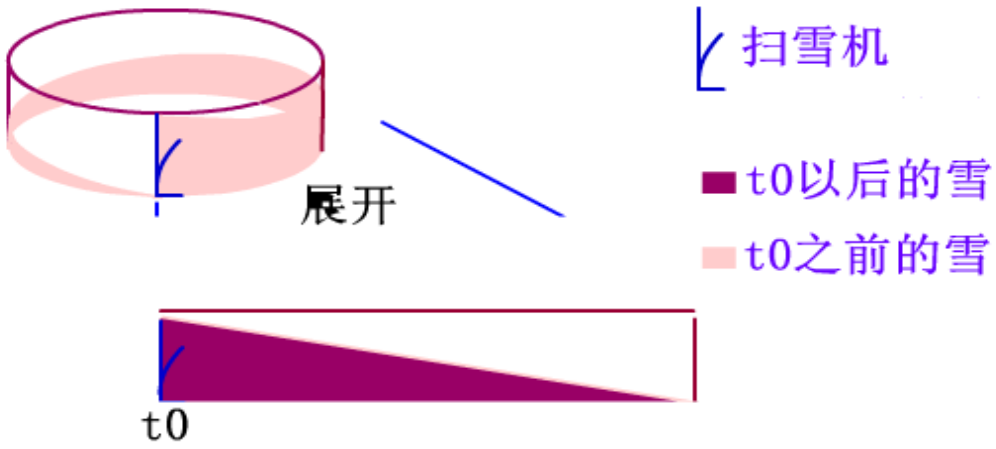
置换选择排序

- 采用置换选择算法，在扫描一遍的前提下，使得所生成的各个顺串有更大的长度
  - 减少了初始顺串的个数，有利于在合并时减少对数据的扫描遍数
- 平均情况下，这种算法可以创建长度为2M个记录的顺串，假设堆可容纳M个记录

置换选择算法的效果

- 置换选择排序算法得到的顺串长度并不相等
- 若堆的大小是M
  - 一个顺串的最小长度就是 M 个记录
  - 最佳情况，例如输入为正序，一次将整个文件生成为一个顺串
  - 平均情况，可以形成长度为2M的顺串
  - 扫雪机

扫雪机模型



click me

置换选择排序 小结

- 产生初始（大）顺串的技术
- 堆排序的扩展
  - 内存组织：一个堆、一个输入缓冲区、一个输出缓冲区

- 后续问题

  产生的这些的顺串如何形成最终的排序序列?

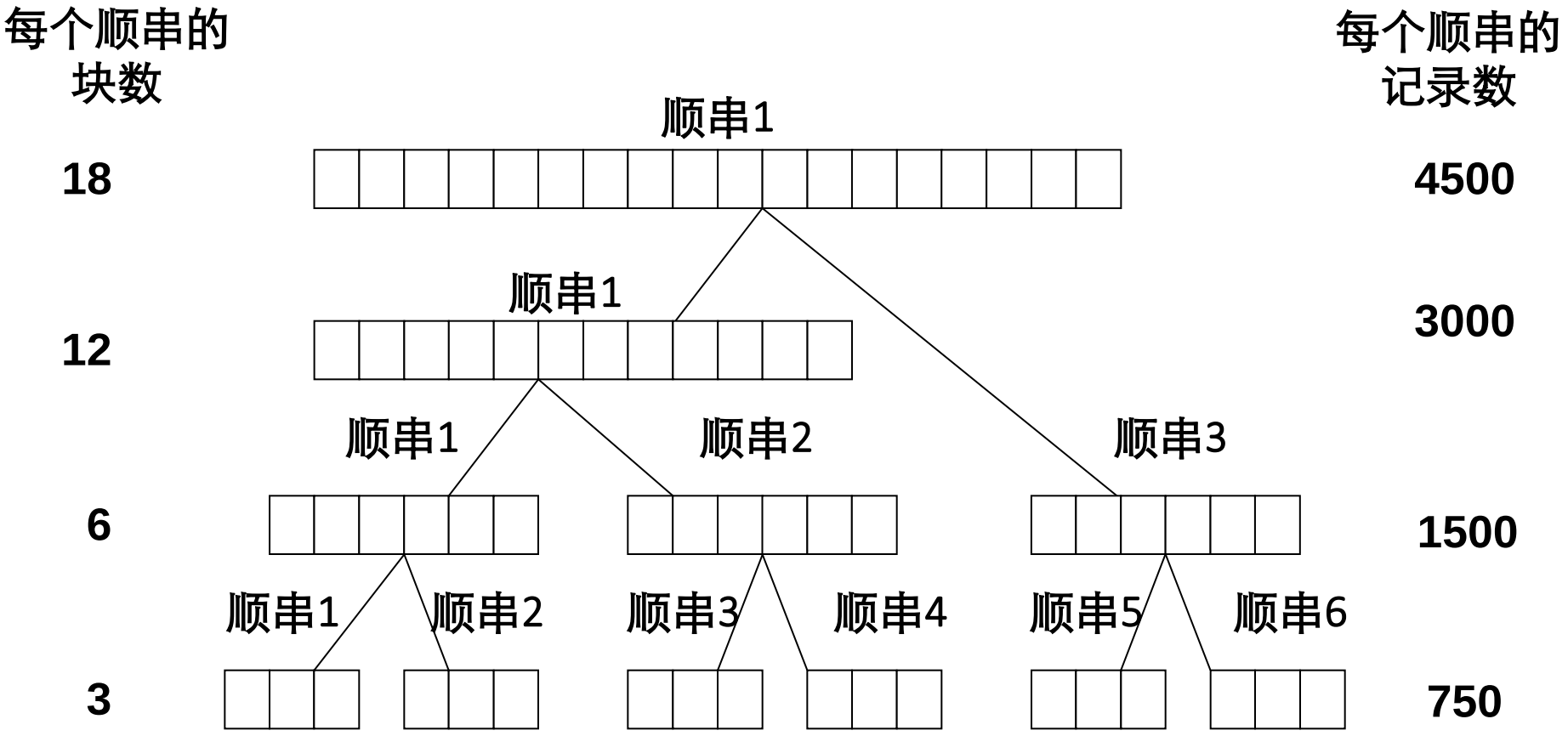
顺串 to 整体有序

- 借鉴归并排序的思想：
  - 对记录顺序完成一系列扫描，在**每趟**扫描中，归并的子序列成倍增大，如二路归并
    - **第1趟** 将长度为 **n** 的子序列归并成长度为 **2n** 的子序列
    - **第2趟**把长度为 **2n** 的子序列归并成长度为 **4n** 的子序列
    - ……

外归并排序

- 归并原理
  - 将第一阶段所生成的顺串加以合并(例如通过若干次二路合并)，直至变为一个顺串为止，即形成一个已排序的文件
  - 为一个待排文件创建**尽可能大的初始顺串**，可以大大减少扫描遍数和外存读写次数
  - **归并顺序的安排** 影响 读写次数，将初始顺串长度作为权，其实质就是 Huffman 树最优化问题

二路归并



读写各：3*6 + 6*2 + (12 + 6) = 48 次

二路归并的开销

- 扫描**趟数**
  - 页块大小
  - 待排序文件大小
- 初始顺串为m个，**扫描趟数**与**合并树高度**有关
  - 「$\log_2 m$」

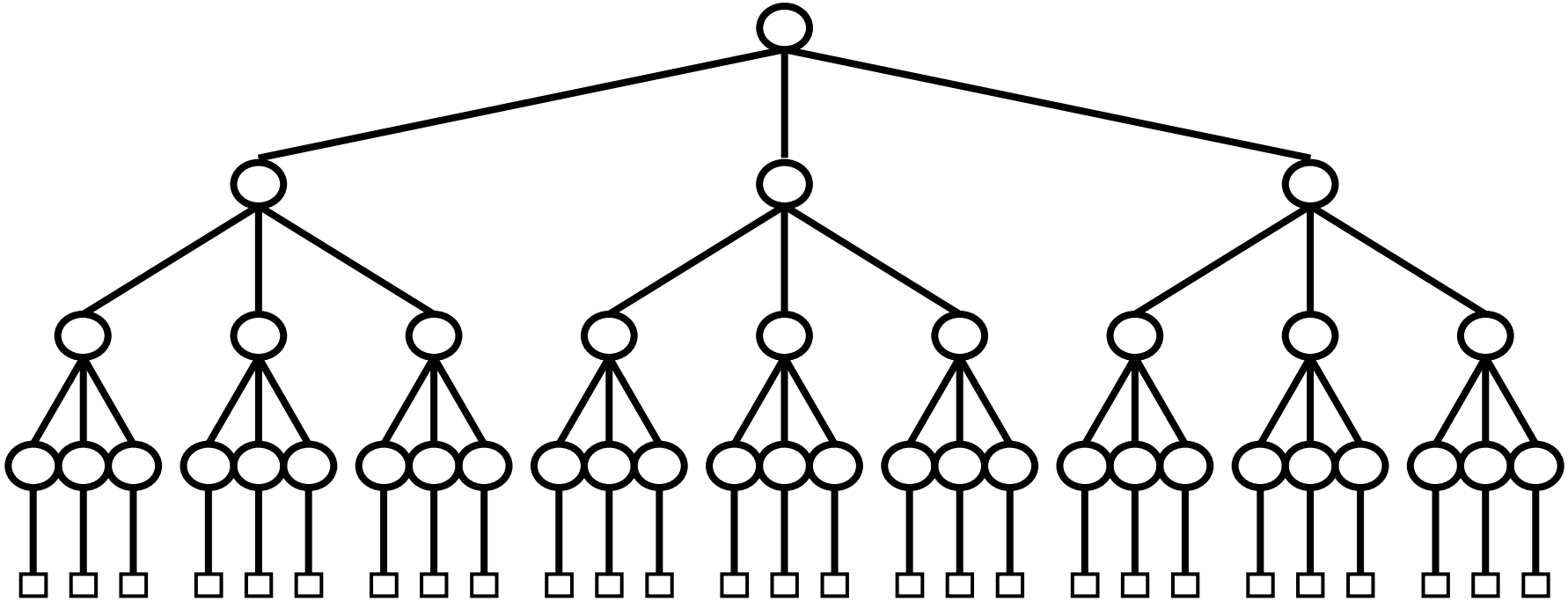
二路归并开销

- 假设上例中包含4500个记录，每个顺串750个记录
  - 共需要 4 趟二路归并把整个文件排序完毕
  - 每趟归并的过程都需把外存文件扫描一遍，即以物理块（页块）为单位对外存文件进行读写多次
    - 假设每个物理页块可以容纳250个记录。4趟归并，分别需要48次读操作、48次写操作；
  - 文件长度不变，初始顺串长度增加1倍，则可以减少1趟归并，只需 30 次读和写。故，为待排文件创建尽可能大的初始顺串，可大幅减少扫描遍数和外存读写次数
  - 归并顺序的安排也能影响读写次数，把初始顺串长度作为权，其实质就是Huffman树最优化问题

多路归并

- *K* 路归并每次将 *k* 个顺串合并成一个排好序的大顺串
  - 一般情况下，对*m*个初始顺串进行 *k* 路归并时归并趟数为 $\lceil log_k m \rceil$
  - 增加每次归并的顺串数量 *k* 可以**减少归并趟数**

多路归并



多路归并

- 在 *k* 路归并中，最直接的方法就是作 *k-1* 次比较找出所要的记录，但代价较大
  - 若得到含有 *u* 个记录的归并段需要进行 *(k-1)×(u-1)* 次比较
  - 对记录总数为 *n* 的文件进行外排序时，用 *m* 表示初始顺串的个数，那么内部归并过程进行的总比较次数

$$\lceil \log_k m \rceil \cdot (k-1) \cdot (n-1) = \lceil \log_2 m / \log_2 k \rceil \cdot (k-1) \cdot (n-1)$$

- *m*不变的情况下，随着 *k* 的增大，其值增大；某种程度上削减了由于增大 *k* 而减少归并趟数带来的好处

多路归并 vs 选择树

- 也称竞赛树，反映了一系列“淘汰赛”的结果
  - 采用顺序存储，一种完全二叉树，有两种类型：
    - 赢者树
    - 败方树
- 选择树从 k 个关键码中找出最小关键码的值所需**比较次数**为树高 $\log_2 k$，使得内部归并过程进行的总比较次数变为

$$\lceil \log_k m \rceil \cdot \lceil \log_2 k \rceil \cdot (n-1) = \left\lceil \frac{\log_2 m}{\log_2 k} \right\rceil \cdot \lceil \log_2 k \rceil \cdot (n-1) = \lceil \log_2 m \rceil \cdot (n-1)$$

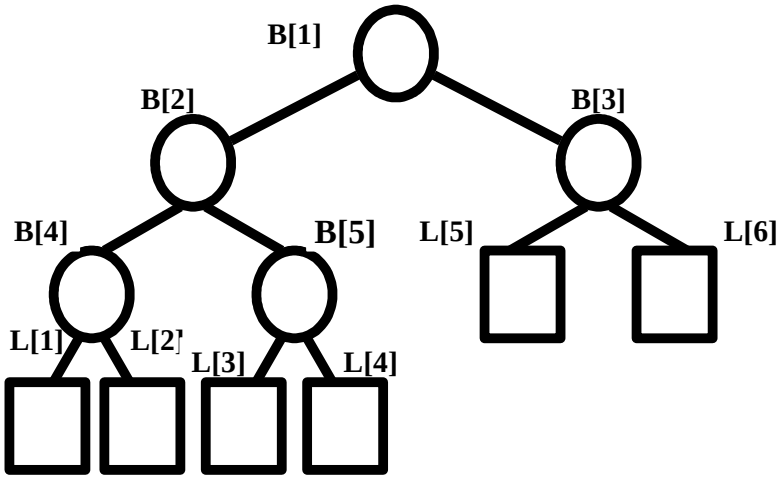
  - 此时内部比较次数与 k 无关，不随 k 的增加而增加

赢者树

- 对于 k 名选手的比赛，赢者树是一棵含有 k 个外部结点，k-1个内部结点的完全二叉树，其中每个内部结点记录了其左右子结点相应赛局的赢家
  - 叶子结点 用数组 L 表示
    - 代表各顺串在合并过程中的当前记录（图中标出了它们各自的关键码值）
  - 分支结点 用数组 B 表示
    - 每个分支结点代表其两个子结点中的赢者（关键码值较小的）所对应数组 L 的索引
  - 根结点是树中的最终赢者的索引，即为下一个要输出的记录结点

赢者树

- 外部结点的数目为 k
- LowExt 代表最底层的外部结点数目
- offset 代表最底层外部结点之上的所有结点数目
- 每一个外部节点L[i] 所对应的内部结点B[p]，i 和p之间存在如下的关系：



$$p = \begin{cases} (i + offset)/2 & i \le LowExt \\ (i - LowExt + k - 1)/2 & i > LowExt \end{cases}$$

k =6,

LowExt=4,

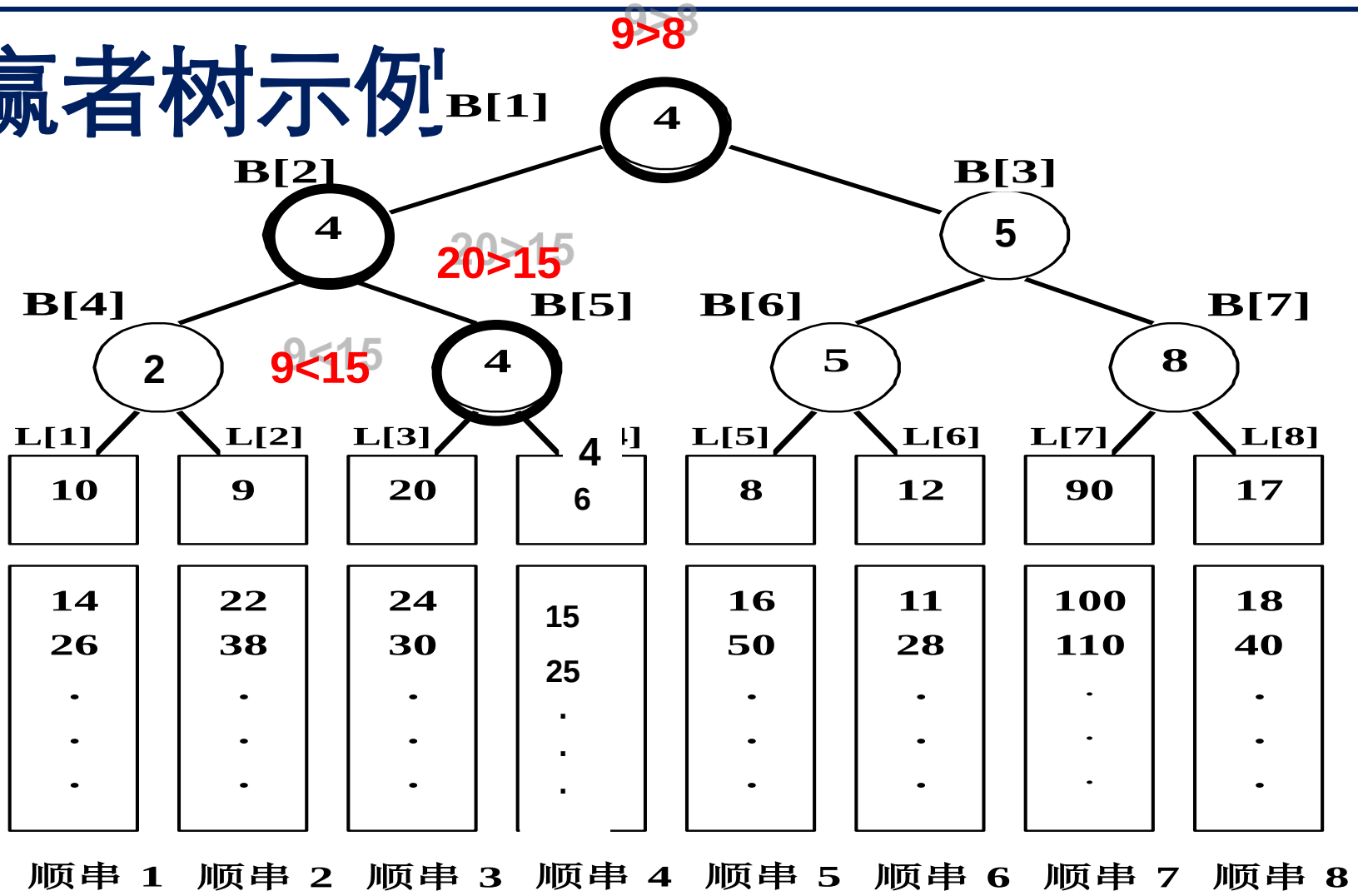
Offset=7

LowExt + Offset = 2k-1

赢者树特点

- 通过比较两个选手的分数确定一场比赛的赢家
  - 从树的最底层叶结点开始进行两两比赛，输的选手被淘汰，赢的继续向上比赛，树根记录了整个比赛的胜者
- 优点之一：若选手L[i]的分值发生改变，可方便地修改这棵赢者树
  - 只需沿着从L[i]到根结点的路径修改二叉树，而不必改变其它比赛的结果
  - 修改次数在0 ～ $\log_2 k$之间
- 重构赢者树的基础

赢者树示例



重构后的赢者树，L[4]记录具有最小的关键码值6，它所指的记录是顺串4的当前记录，该记录即为下一个要输出的记录。

根结点所指向的L[4]记录具有最小的关键码值6，它所指的记录是顺串4的当前记录，该记录即为下一个要输出的记录。为了重构这棵树，只须沿着从结点L[4]到根结点的路径重新进行比赛

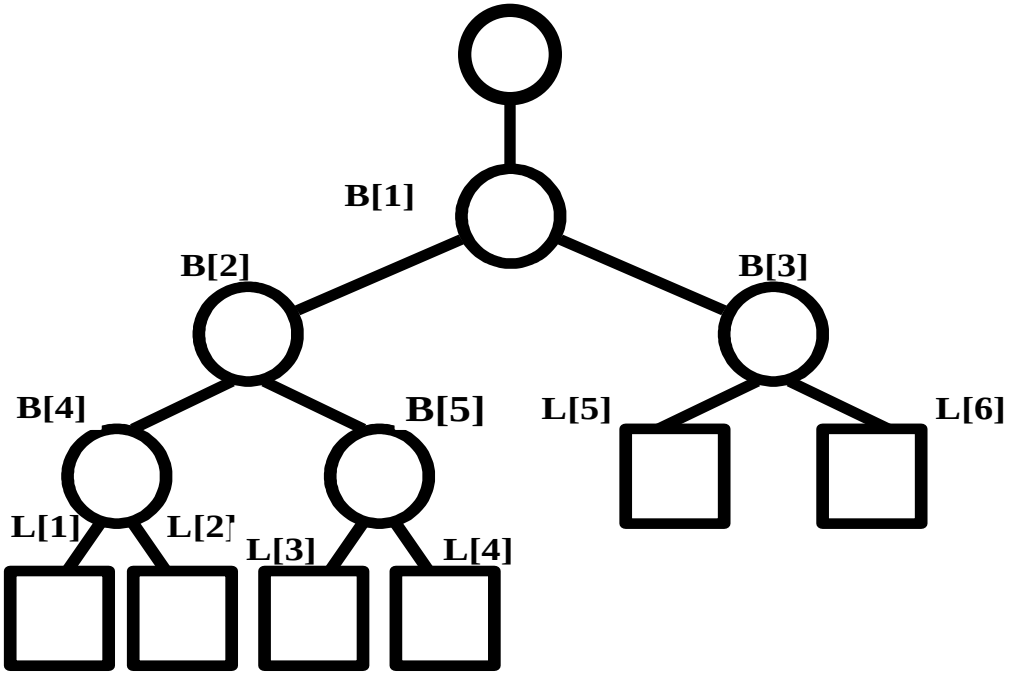
败方树

- 赢者树的一种变体
  - 父结点记录其左右子结点比赛的败者，而让获胜者去参加更高阶段的比赛
  - 根结点处加入一个结点来记录整个比赛的胜者
- 简化选择树的重构过程
  - 重构时只需与从进入树中的结点到根的路径上的结点进行比较即可，减少了从该位置到败者树的根结点的路径上确定各个索引的工作量
  - 假设L[i]的关键码值最小，当用新的关键码代替它重构选择树时，
    - 由于赢者树中记录了比赛的胜者，因此路径上的所有比赛都需要找到相应的兄弟结点进行比较
    - 败者树的分支结点由于记录了比赛的败者则只**需要往根的方向与上层父结点**所索引的关键码进行比较即可

败方树比赛过程

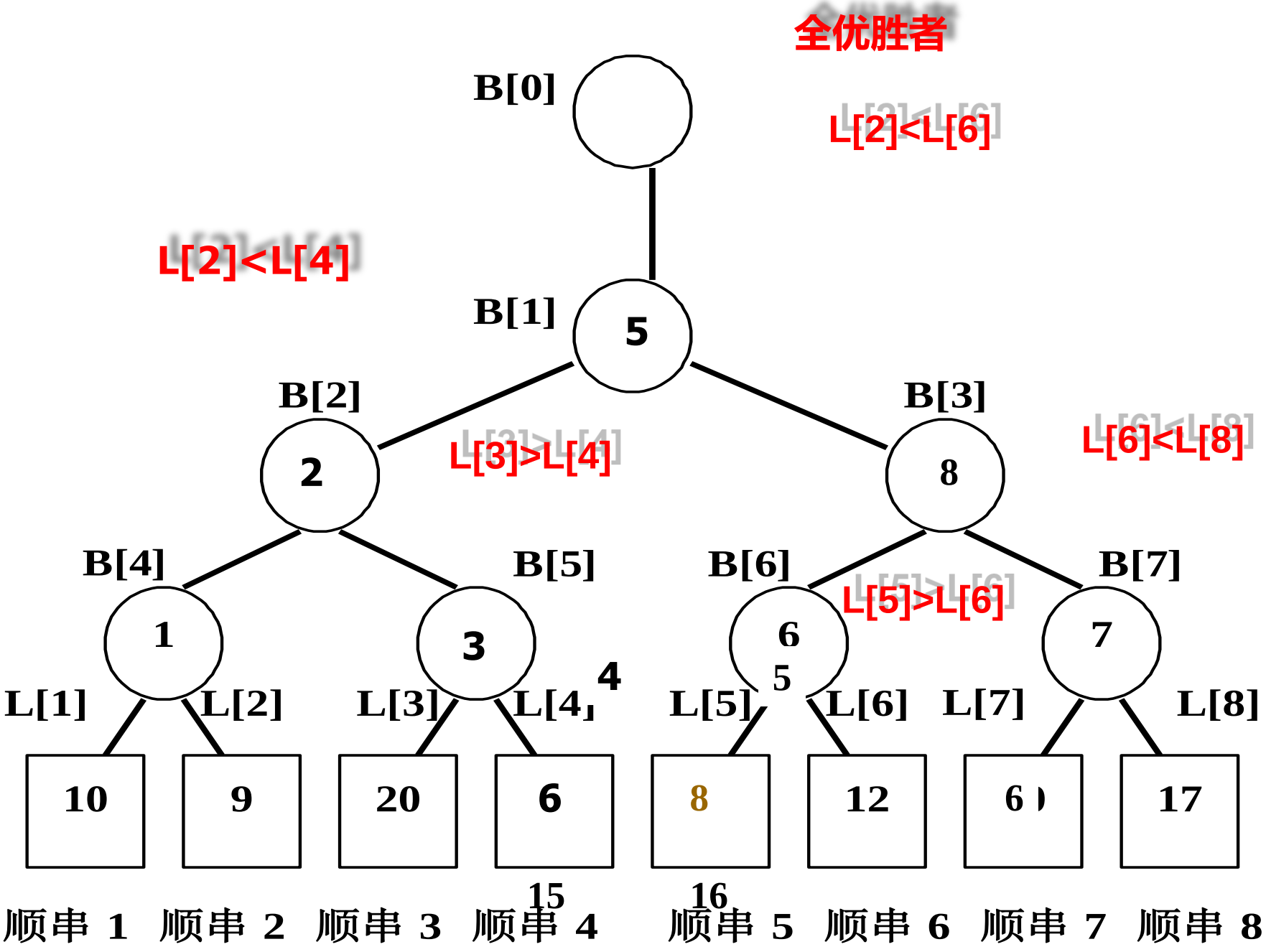
- 新进入树的结点与其父结点进行比赛
  - 败者存放在父结点中
  - 而胜者再与上一级的父结点进行比赛
- 如此，比赛不断进行直到结点B[1]处结束
  - 败者的索引放在结点B[1]
  - 胜者的索引放到结点B[0]

败方树示例



- k =6
- LowExt=4
- Offset=7

$$\begin{cases} (i + offset)/2 & i \le LowExt \\ (i - LowExt + k - 1)/2 & i > LowExt \end{cases}$$


全优胜者
B[0]
L[2]<L[6]
L[2]<L[4]
B[1]
5
B[2]
B[3]
2
L[3]>L[4]
8
L[6]<L[8]
B[4]
B[5]
B[6]
B[7]
L[5]>L[6]
1
3
6
7
5
4
L[1]
L[2]
L[3]
L[4]
L[5]
L[6]
L[7]
L[8]
10
9
20
6
8
12
6
17
15
16
顺串 1
顺串 2
顺串 3
顺串 4
顺串 5
顺串 6
顺串 7
顺串 8

多路归并的效率

假设对 *k* 个顺串进行归并

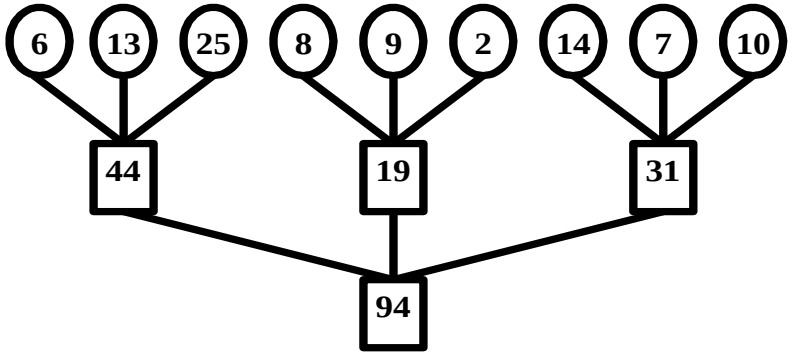
- 原始方法： $\Theta(k \cdot n)$
  - 找到每一个最小值的时间是 $\Theta(k)$
  - 产生一个大小为 *n* 的顺串的总时间是 $\Theta(k \cdot n)$
- 败方树方法： $\Theta(k + n \cdot \log k)$
  - 初始化包含 k 个选手的败方树需要 $\Theta(k)$ 的时间
  - 读入一个新值并重构败方树的时间为 $\Theta(\log k)$
  - 产生一个大小为n 的顺串的总时间为 $\Theta(k + n \cdot \log k)$

最佳归并树

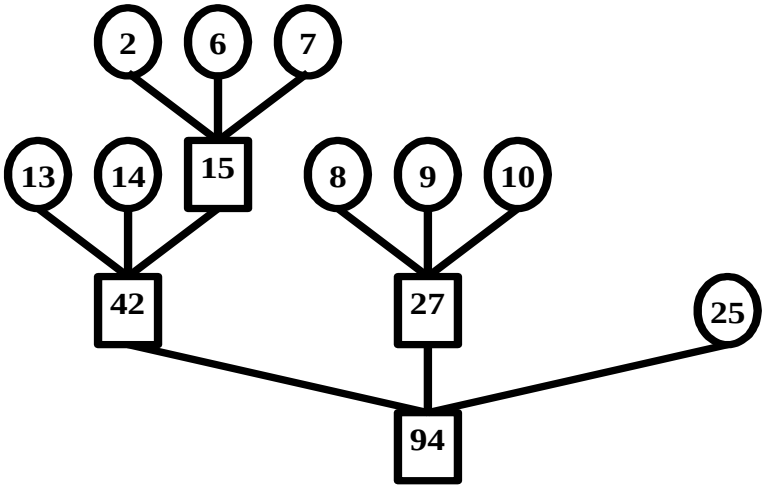
- 各初始顺串的长度不同呢?
  - 如何得到最佳的结果?
- 归并顺序的安排也能影响读写次数，把初始顺串长度作为权，其实质就是Huffman树最优化问题

最佳归并树

- 假设9个初始顺串所占的外存块数分别为6，13，25，8，9，2，14，7，10



(a)一棵普通的归并树



(b)最佳归并树

最佳归并树

（a）访外总次数为

(6+13+25+8+9+2+14+7+10)×2×2 = 376

（b）外存读/写块的次数为

(2+6+7)×3×2+(13+14)×2×2+(8+9+10)×2×2+25×2

= 356

最佳归并树

- 若进行多路归并时，**各顺串初始长度不一**，对外存扫描的次数，即执行时间会产生影响
  - 若将所有初始顺串的块数作为树的叶结点，采用K路归并则建立起一棵K-Huffman树；
  - 这样的一棵Huffman树就是最佳归并树
- 通过最佳归并树进行多路归并可使对外存的I/O降到最少，提高归并的执行效率

思考

- 是否可以用赢者树或败方树形成初始顺串?
- 是否可以用堆进行多路归并?

课堂练习☺

1. 有一组待排序的记录，其排序码为{18，5，20，30，9，27，6，14，45，22}，请问采用用直接插入排序算法和直接选择排序的比较次数各是多少?
2. 针对逆序的整数序列而言，直接插入排序、直接选择排序、冒泡排序中哪个最快?
3. 设有一个包含4500个记录的输人文件，用一台其内存至多可对750个记录进行排序的计算机对其进行排序。输人文件放在磁盘上，磁盘的每个页块可容纳 250个记录。输出文件也放在磁盘上，用以存放归并结果。由于内存中可用于排序的存储区域能够容纳750个记录，故内存中恰好能存放3个页块的记录。外归并排序伊始，每3块一组读人内存。利用某种内排序方法进行内排序，形成初始归并段，再写回外存。请画出该排序过程并计算总的磁盘读写次数。
4. 设有12个归并段，其长度分别为30，44，8，6，3，20，60，18，9，62，68，85；现欲作4路外部归并排序，试画出表示归并过程的最佳归并树，并计算总读写次数。
5. 磁盘文件采用选择法实现k路归并时，占用CPU时间是否与k相关，为什么?