

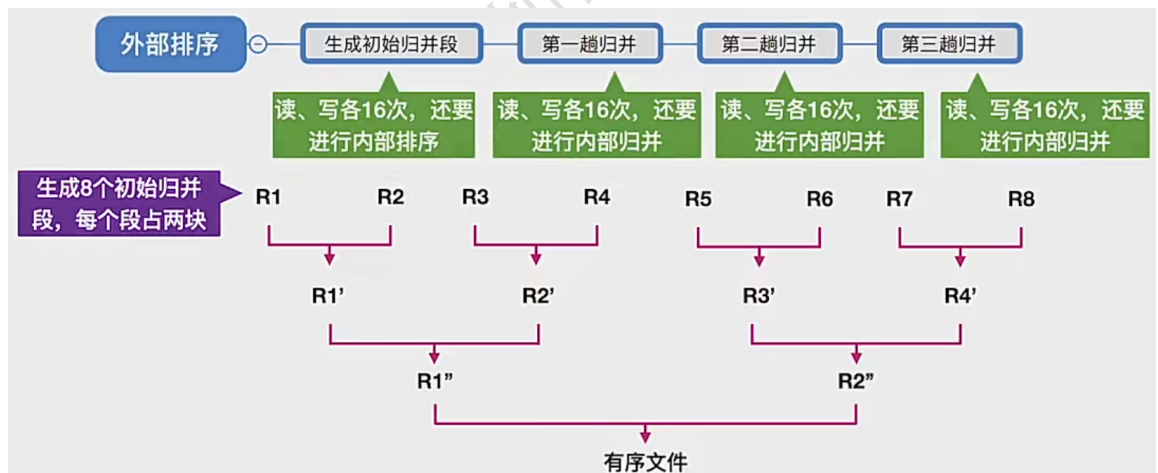
Lecture 34: 外部排序

34.1 外部排序基础

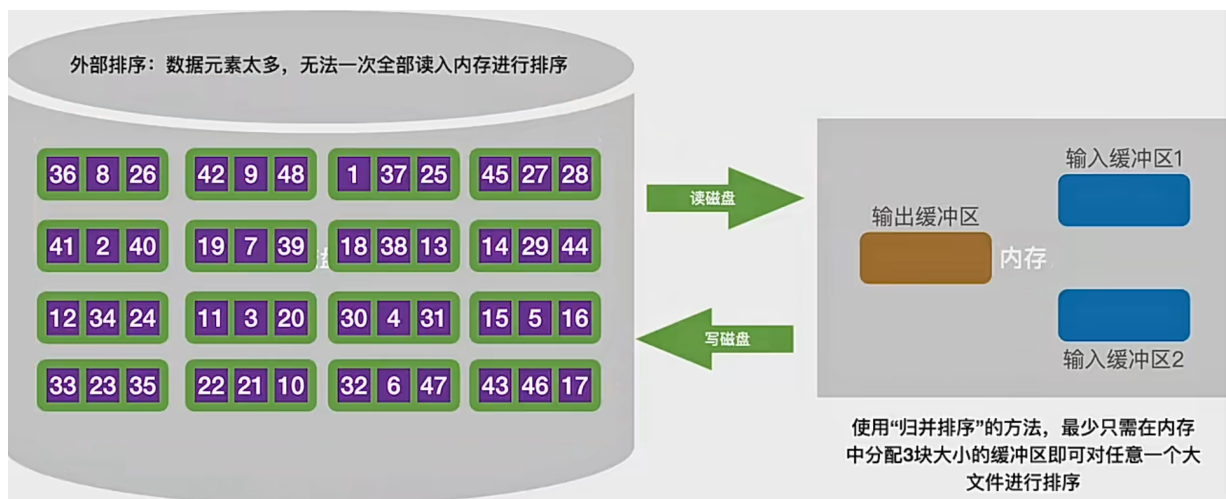
34.1.1 外部排序原理

基础概念

- 数据交换：
 - 操作系统以“块”为单位对磁盘存储空间进行管理，如：每块大小 1KB 各个磁盘块内存放着各种各样的数据；
 - 磁盘的读/写以“块”为单位数据读入内存后，才能被修改，修改完了还要写回磁盘；
- 外部排序原理：
 - 数据元素太多，无法一次性读入内存，因此需要对存在外存中的数据进行排序；
 - 举例：
 - 如果使用“归并排序”的方法，最少只需在内存中分配 3 块大小的缓冲区，即可对任意一个大文件进行排序；
 - “归并排序”要求各个子序列有序，每次读入两个块的内容，进行内部排序后写回磁盘；
- 图示：



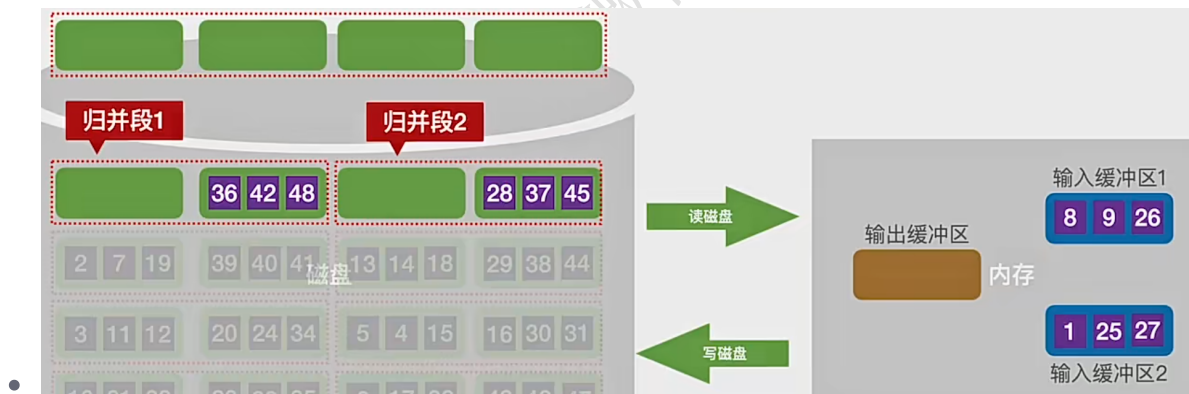
- 开销：
 - 其中 **读写外存的时间** 这个部分占大头；
 - 读、写磁盘次数 = $32 + 32 * 3 = 128$ 次 $\rightarrow$ 其中 $32 = 16 * 2$ ，表示文件总数（16）乘以 2。 $32 * 3$ 中的 3 表示归并了三趟；
 - 外部排序时间开销 = 读写外存的时间 + 内部排序所需时间 + 内部归并所需时间
- 图示：



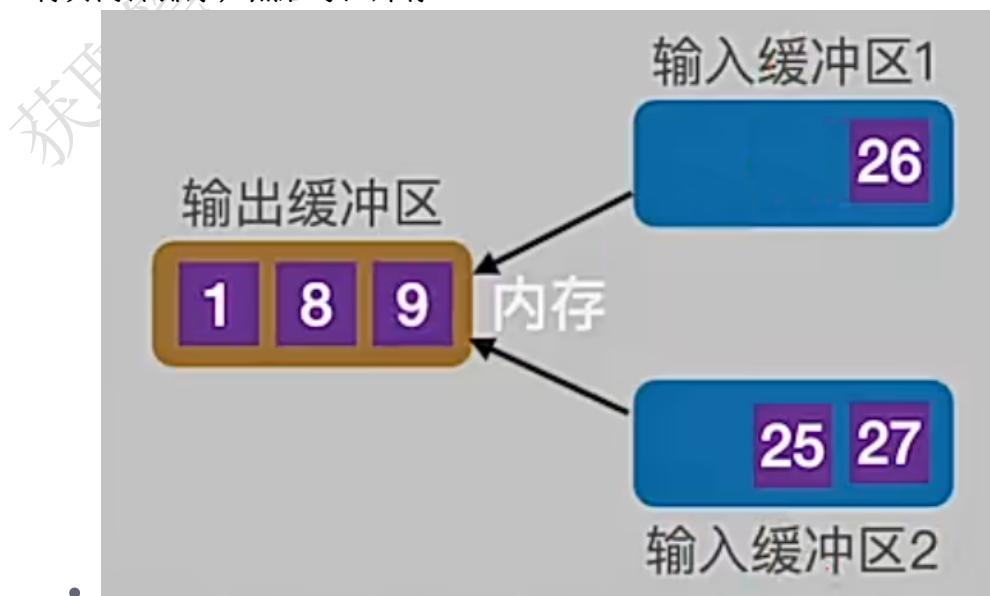
34.1.2 归并演示

对外存中数据进行二路归并

- 总结：
 - 再第一趟时，生成 8 个初始归并段，每个段占两块；
- 第一趟：
 - 图示：
 - 从两个归并段中，各自取出数据较小的那个部分，放入内存、进行内部排序；



- 将其内部排序，然后写回外存：

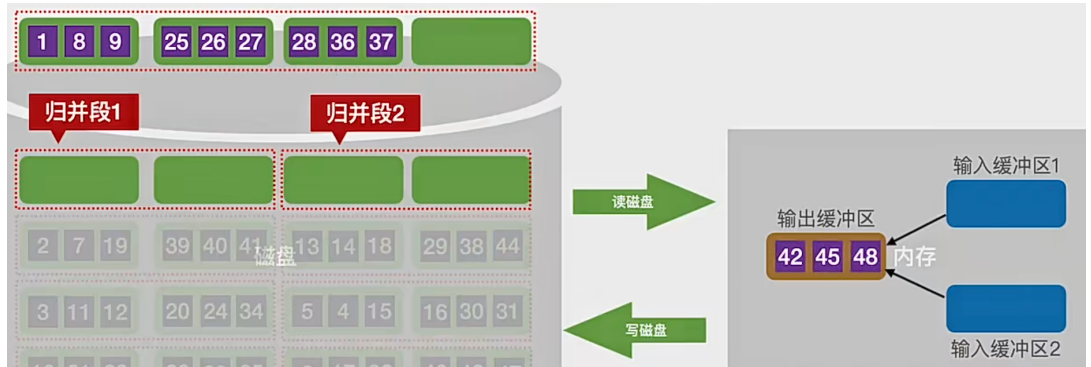


- 然后继续进行归并排序：

- 当输入缓冲区空了之后，需要从外存中的归并段再读入新的一段。比如下图中就已经把归并段 1 的第二部分也读进去了；



- 继续完成：

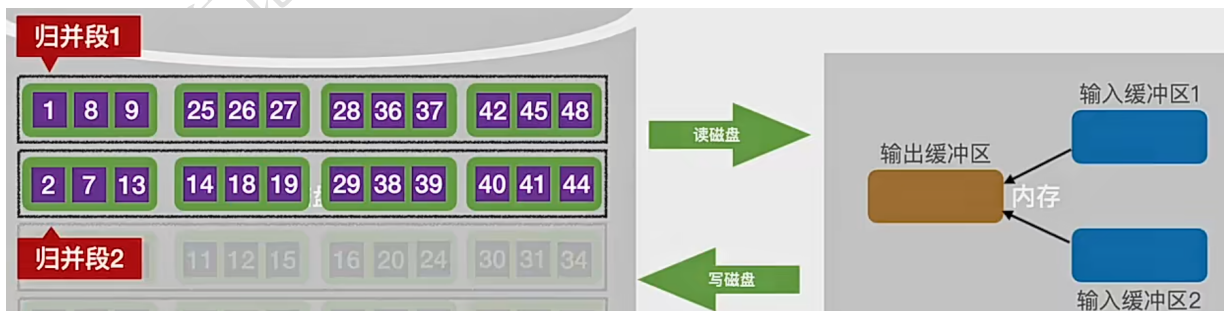


- 归并成功

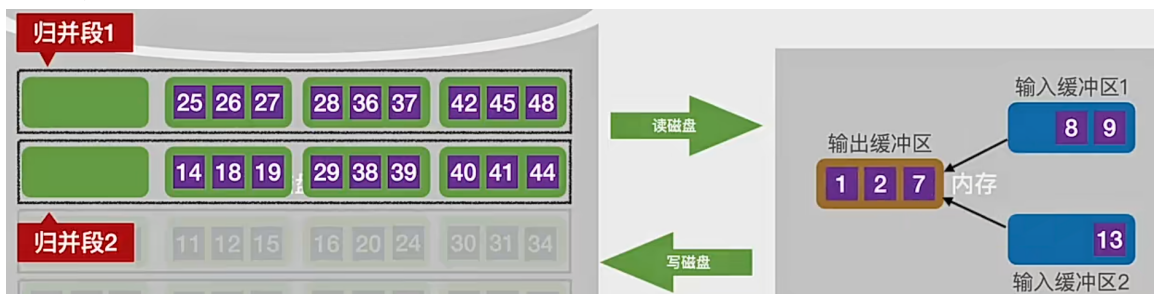


- 然后重复上述过程，对每一段对进行归并；

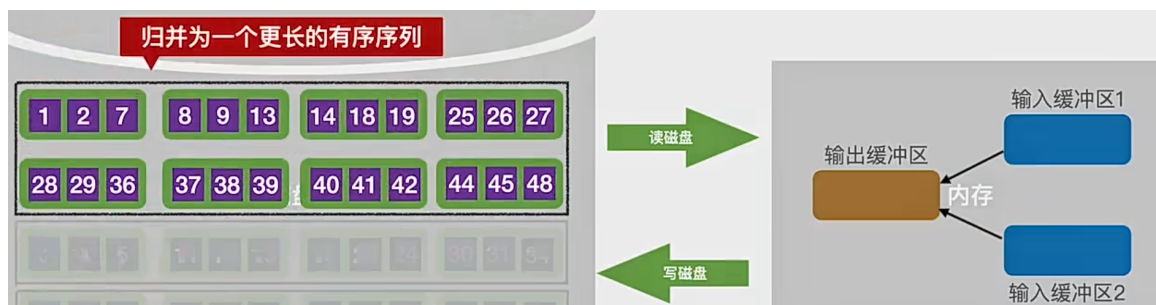
- 第二趟：



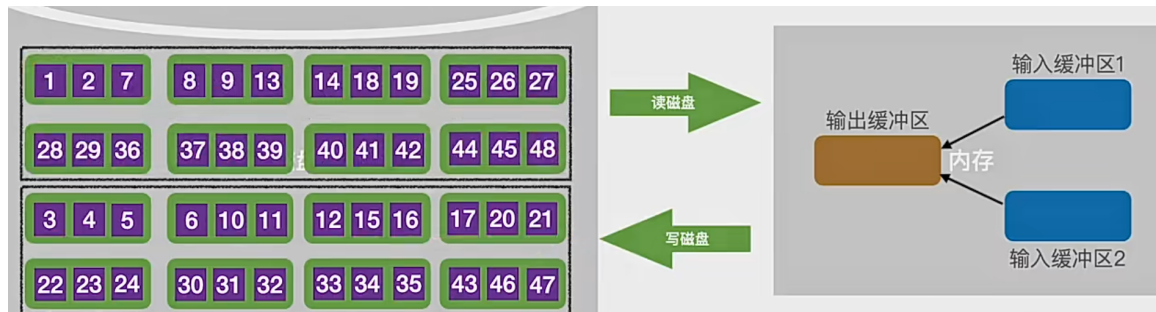
- 方法和之前一致：



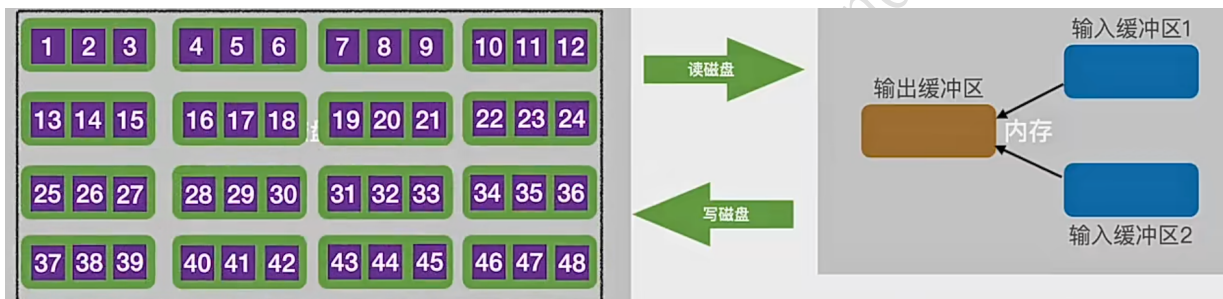
- 不断重复，得到结果：



- 继续归并：



- 第三趟：



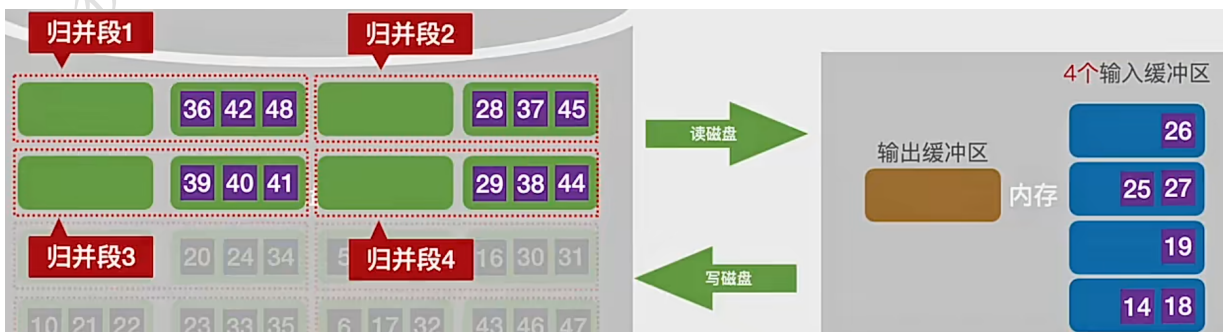
34.1.3 外部排序的优化

核心：

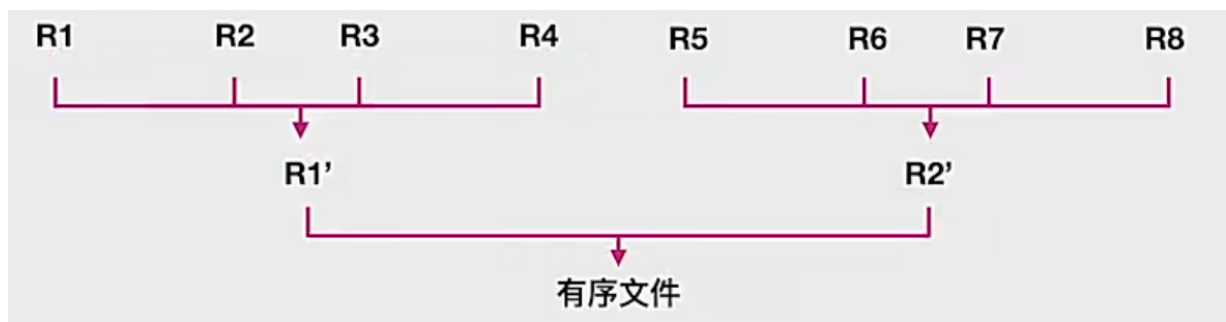
$$\text{归并趟数} = h - 1 = \lceil \log_k r \rceil$$

优化思路：多路归并 $\rightarrow$ 减少 k

- 举例：
 - 将 4 个有序归并段归并为一个；
- 图示：



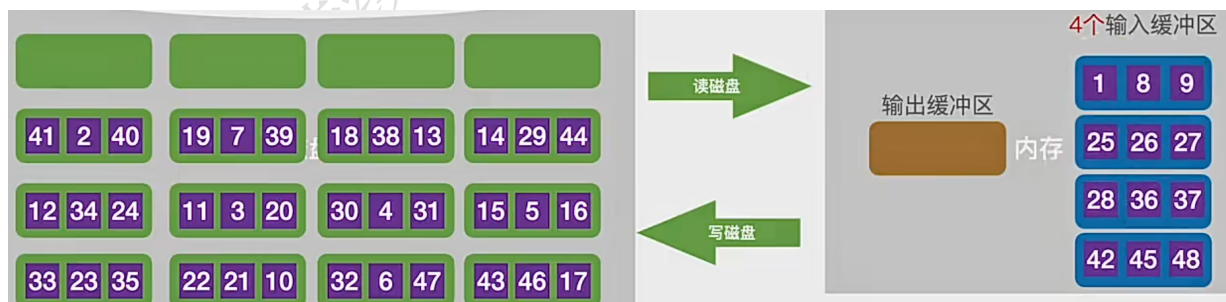
- 归并趟数：
 - 采用 4 路归并，只需进行两趟归并即可读、写磁盘次数 = $32 + 32 * 2 = 96$ 次



- 结论：
 - 使用多路归并可以减少归并的趟数，从而减少 IO 读写的次数；
- 效率：
 - 归并趟数：
 - $$\text{归并趟数} = h - 1 = \lceil \log_k r \rceil$$
 - 文字结论：
 - k 越大、r 越小，归并趟数越少，读写磁盘次数越少
- 多路归并的负面影响：
 - 1. K 路归并时，需要开辟 K 个输入缓冲区，增大内存开销；
 - 2. 每挑选一个关键字需要对比 $k-1$ 次，内部归并排序所需要花的时间增加；

优化思路：减少初始归并段数量 $\rightarrow$ 减少 r

- 概念：
 - 生成初始归并段的“内存工作区”越大，初始归并段越长；
- 结论：
 - 若能增加初始归并段的长度，则可减少初始归并段数量 r
- 图示：



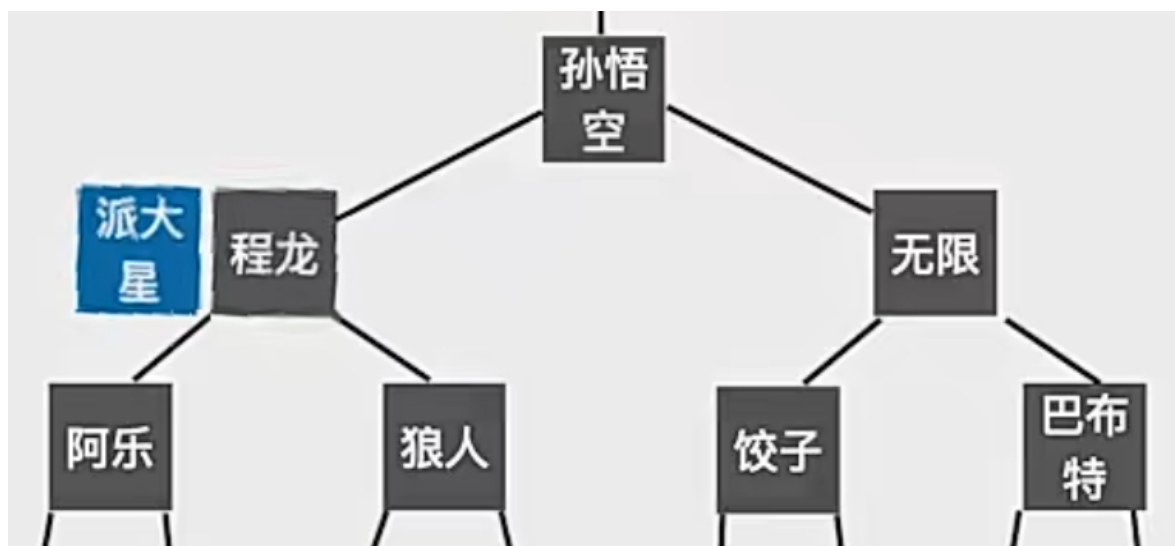
34.2 败者树

34.2.1 多路平衡归并的问题

多路平衡归并的问题

- 问题：
 - 前提 $\rightarrow$ 归并趟数 $S = \lceil \log_k r \rceil$, 归并路数 k 增加, 归并趟数 S 减小, 读写磁盘总次数减少
 - 举例 $\rightarrow$ 8 路平衡归并, 从八个归并段中选出一个最小元素需要对比关键字 7 次；
 - 原因 $\rightarrow$ 使用 k 路平衡归并策略, 选出一个最小元素需要对比关键字 $(k-1)$ 次, 导致内部归并所需时间增加；

- 优化方法 -> 败者树；
- 什么是败者树：
 - 败者树的构造 -> 失败者留在这一回合，胜利者进入下一回合比拼；
 - 构造败者树 -> 可视为一棵完全二叉树， k 个叶结点分别是当前参加比较的元素，非叶子结点用来记忆左右子树中的“失败者”，而让胜者往上继续进行比较，一直到根结点；
- 插入新的结点：
 - 假设当前插入了新的节点“派大星”，此时因为败者树已经构建好了、其余 7 个结点的胜负已经比出来了，因此“派大星”只需要先和“阿乐”比，然后再往上那个、和“程龙”比，然后再往上比，最多只需要比三次；
 - 基于已经构建好的败者树，选出新的胜者只需进行 3 场比赛；



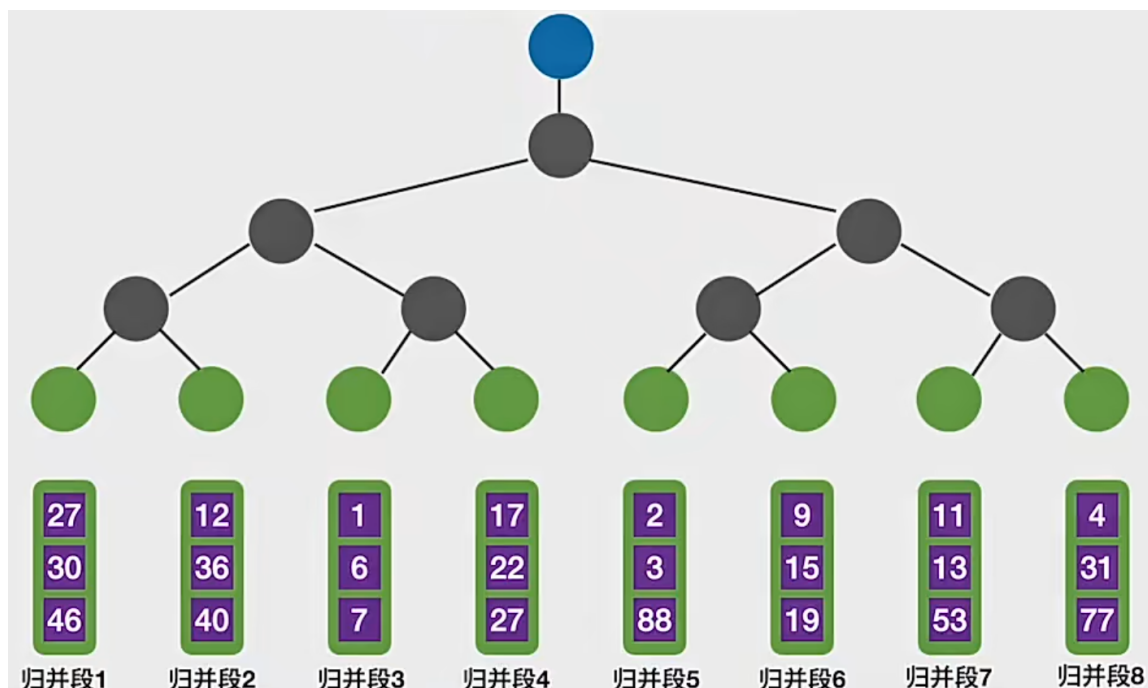
34.2.2 败者树在多路平衡归并中的应用

败者树在多路平衡归并中的应用

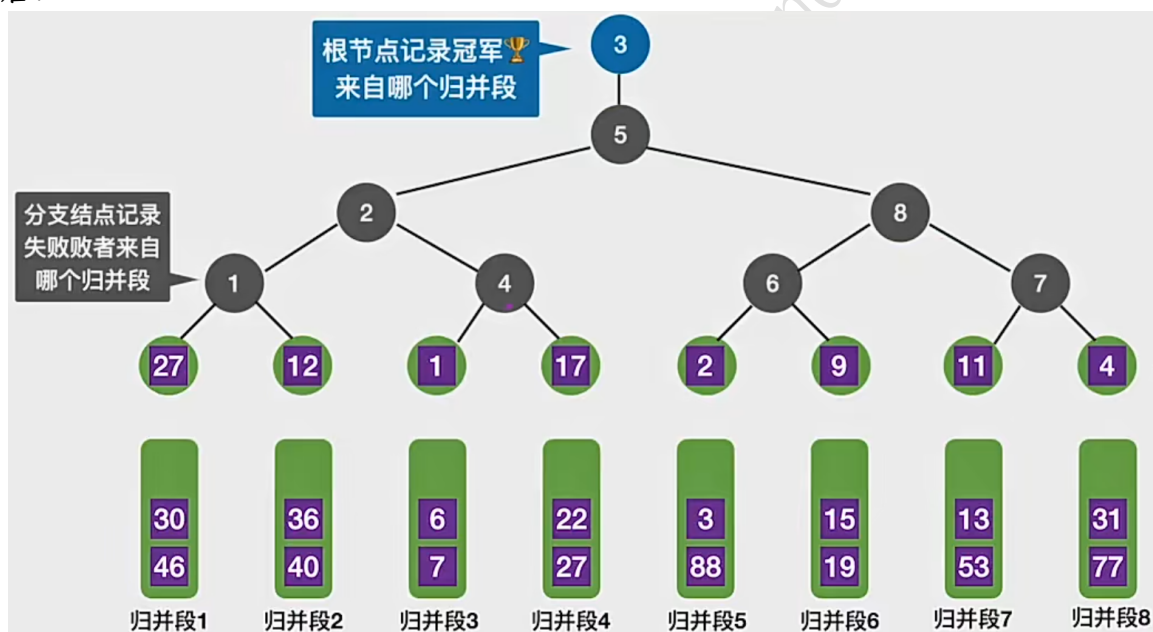
- 构建：
 - 先构建一棵败者树 -> 每个叶子结点对应一个归并段 -> 根节点记录冠军来自哪一个归并段；
 - 对于 k 路归并，第一次构造败者树需要对比关键字 $k-1$ 次；
- 总结：
 - 对于已经构建好了的败者树，选出最小元素、只需要对比 $\lceil \log_2 r \rceil$ 次
 - 一共要选择 $n-1$ 次，则完成一次内部排序的时间复杂度为：

$$(n-1) * \lceil \log_2 r \rceil$$

- 其中 r 为路数
- 和传统方法比较：
 - 优化前多路归并 -> 对于 1024 路、需要对比 1023 次；
 - 优化后多路归并 -> 对于 1024 路、在败者树中，只需要对比（树高）19 次；
- 图示：
 - 构造前：



- 构造后：

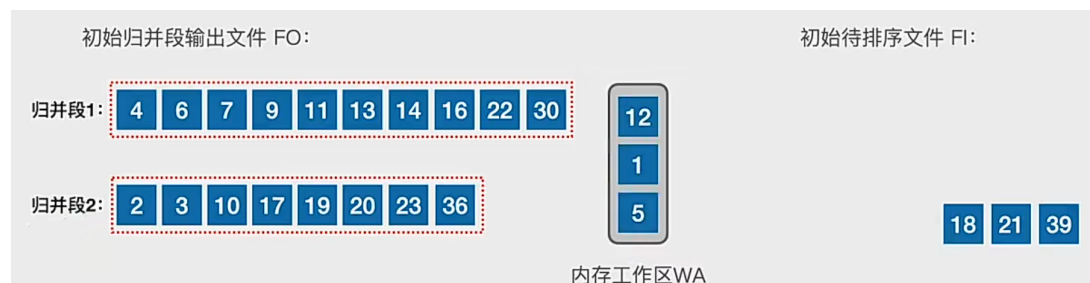


34.3 置换 - 选择排序

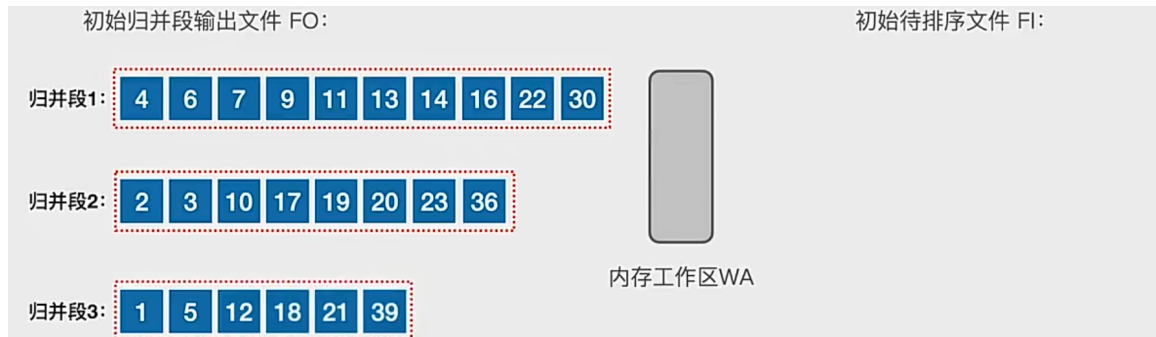
34.3.1 基础概念

概念

- 减少 r :
 - 老办法 -> 可以用一片更大的内存区域来进行内部排序 (如: 可容纳 18 个记录);
 - 置换选择排序 -> 使用置换-选择排序, 可以让每个初始归并段的长度超越内存工作区大小的限制;
- 图示: 假设用于内部排序的内存工作区只能容纳 3 个记录
 - 第一个归并段生成:
 - 将 WA 中最小的数字 4 置换出去;
 - 用 $MINMAX = 4$ 存储最小的记录;



- 第三趟：得到三个初始归并段



24.3.2 总结

总结

- 概念：
 - 设初始待排文件为 FI，初始归并段输出文件为 FO，内存工作区为 WA，FO 和 WA 的初始状态为空，WA 可容纳 w 个记录；
- 步骤：
 - 置换-选择算法的步骤如下：
 - 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 F 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，作为新的 MINIMAX 记录。
 - 6)重复 3)~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，输出一个归并段的结束标志到 FO 中去。
 - 7)重复 2)~6)，1 直至 WA 为空。由此得到全部初始归并段