（注意，星号后面有一空格），则rm命令会删除全部当前目录中的文件，然后报告说找不到文件.o。在MS-DOS和一些其他系统中，文件的删除仅仅是在对应目录或i节点上设置某一位，表示文件被删除，并没有把磁盘块返回到空闲表中，直到确实需要时才这样做。所以，如果用户立即发现了操作错误，可以运行特定的一个“撤销删除”（即恢复）实用程序恢复被删除的文件。在Windows中，删除的文件被转移到回收站目录中（一个特别的目录），稍后若需要，可以从那里还原文件。当然，除非文件确实从回收站目录中删除，否则不会释放空间。

4.4.4 文件系统性能

访问磁盘比访问内存慢得多。读内存中一个32位字大概要10ns。从硬盘上读的速度大约超过100MB/s，对32位字来说，大约要慢4倍，还要加上5~10ms寻道时间，并等待所需的扇面抵达磁头下。如果只需要一个字，内存访问则比磁盘访问快百万数量级。考虑到访问时间的这个差异，许多文件系统采用了各种优化措施以改善性能。本节我们将介绍其中三种方法。

1. 高速缓存

最常用的减少磁盘访问次数技术是*块高速缓存*（block cache）或者*缓冲区高速缓存*（buffer cache）。在本书中，高速缓存指的是一系列的块，它们在逻辑上属于磁盘，但实际上基于性能的考虑被保存在内存中。

管理高速缓存有不同的算法，常用的算法是：检查全部的读请求，查看在高速缓存中是否有所需要的块。如果存在，可执行读操作而无须访问磁盘。如果该块不在高速缓存中，首先要把它读到高速缓存，再复制到所需地方。之后，对同一个块的请求都通过高速缓存完成。

高速缓存的操作如图4-28所示。由于在高速缓存中有许多块（通常有上千块），所以需要有某种方法快速确定所需要的块是否存在。常用方法是将设备和磁盘地址进行散列操作，然后，在散列表中查找结果。具有相同散列值的块在一个链表中连接在一起。这样就可以沿着冲突链查找其他块。

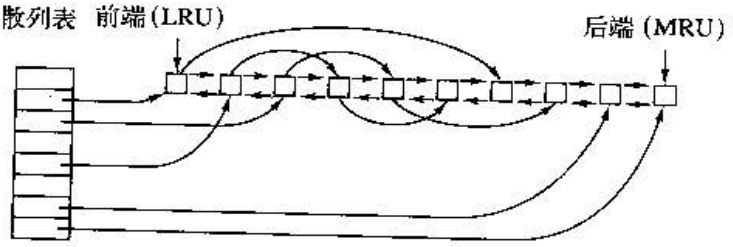

图4-28 缓冲区高速缓存数据结构

如果高速缓存已满，则需要调入新的块，因此，要把原来的某一块调出高速缓存（如果要调出的块在上次调入以后修改过，则要把它写回磁盘）。这种情况与分页非常相似，所有常用的页面置换算法在第3章中已经介绍，例如FIFO算法、第二次机会算法、LRU算法等，它们都适用于高速缓存。与分页相比，高速缓存的好处在于对高速缓存的引用不很频繁，所以按精确的LRU顺序在链表中记录全部的块是可行的。

在图4-28中可以看到，除了散列表中的冲突链之外，还有一个双向链表把所有的块按照使用时间的先后次序链接起来，近来使用最少的块在该链表的前端，而近来使用最多的块在该链表的后端。当引用某个块时，该块可以从双向链表中移走，并放置到该表的尾部去。用这种方法，可以维护一种准确的LRU顺序。

但是，这又带来了意想不到的难题。现在存在一种情形，使我们有可能获得精确的LRU，但是碰巧该LRU却又不符合要求。这个问题与前一节讨论的系统崩溃和文件一致性有关。如果一个关键块（比如i节点块）读进了高速缓存并做过修改，但是没有写回磁盘，这时，系统崩溃会导致文件系统的不一致。如果把i节点块放在LRU链的尾部，在它到达链首并写回磁盘前，有可能需要相当长的一段时间。

此外，某一些块，如i节点块，极少可能在短时间内被引用两次。基于这些考虑需要修改LRU方案，并应注意如下两点：

1) 这一块是否不久后会再次使用？

2) 这一块是否关系到文件系统的一致性？

考虑以上两个问题时，可将块分为i节点块、间接块、目录块、满数据块、部分数据块等几类。把有可能最近不再需要的块放在LRU链表的前部，而不是LRU链表的后端，于是它们所占用的缓冲区可以很快被重用。对很快就可能再次使用的块，比如正在写入的部分满数据块，可放在链表的尾部，这样它们能在高速缓存中保存较长的一段时间。