第八章 磁盘存储器的管理

8.1 外存的组织方式

8.2 文件存储空间的管理

8.3 提高磁盘I/O速度的途径

8.4 提高磁盘可靠性的技术

8.5 数据一致性控制

- 对外存管理的主要任务和要求：

① 有效地利用外存空间

② 提高磁盘的I/O速度

③ 提高磁盘系统的可靠性

8.1 外存的组织方式

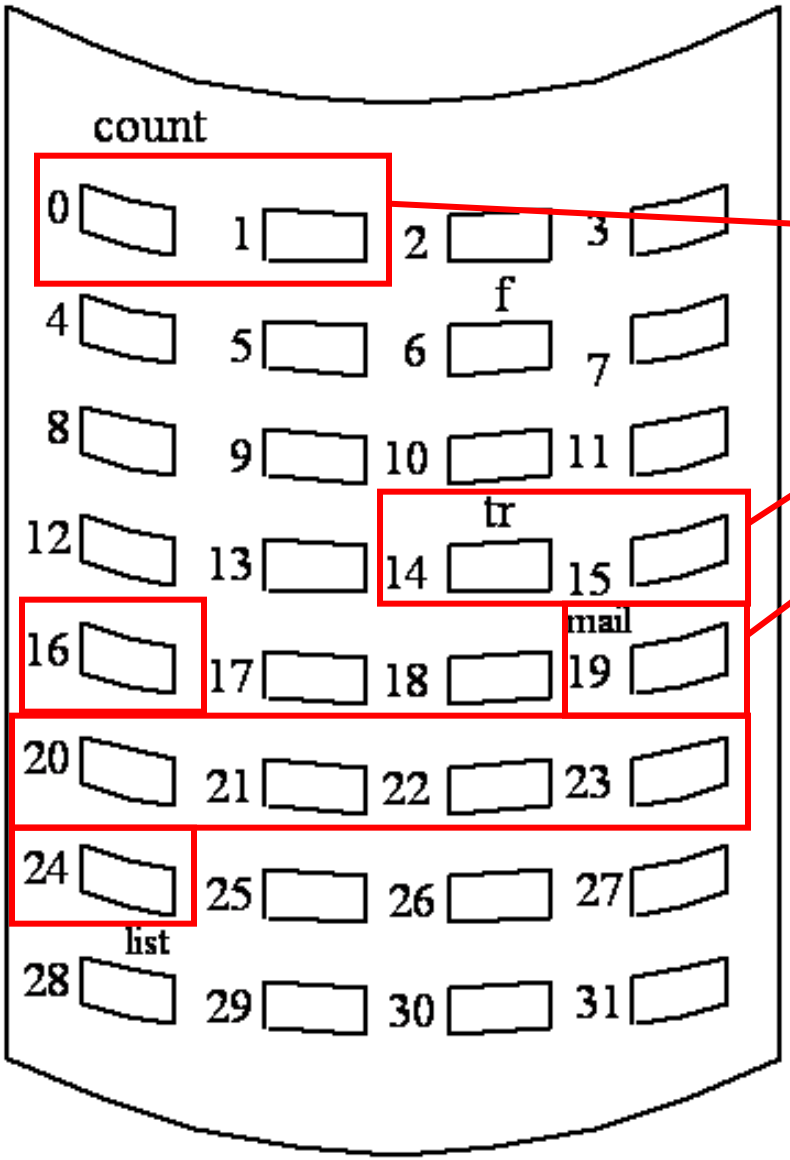
- 常用的外存组织方式：

① 连续组织方式（连续分配方式）

② 链接组织方式（链接分配方式）

③ 索引组织方式（索引分配方式）

连续组织方式



目录

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- 基本思想
  - •文件存储于连续的盘块上
- 文件结构
  - • 顺序文件结构

- 优点

  ① 顺序访问比较容易

  ② 顺序访问的速度快

- 缺点

  ① 必须要求连续的存储空间

  ② 必须事先知道文件的长度

  ③ 容易形成外存碎片

  ④ 难以应付文件的动态变化

  ⑤ 删除插入记录不灵活

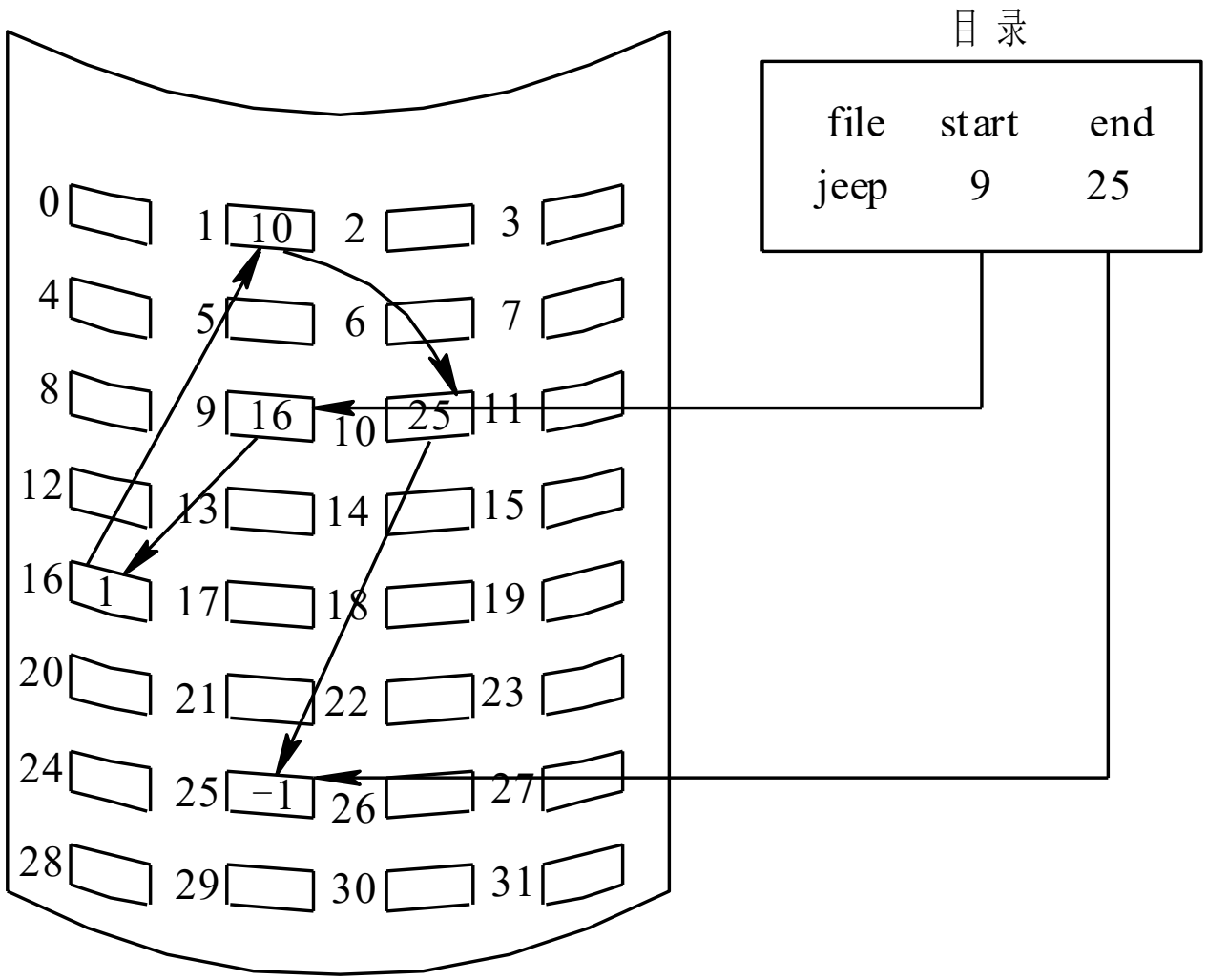
链接组织方式

- 基本思想
  - 通过盘块上的链接指针，把保存在不同盘块上的各文件部分链接起来（离散组织方式）
- 基本方法

① 隐示链接

② 显示链接

① 隐式链接


目 录
file start end
jeep 9 25
0 1 10 2 3
4 5 6 7
8 9 16 10 25 11
12 13 14 15
16 1 17 18 19
20 21 22 23
24 25 −1 26 27
28 29 30 31

- 隐式链接的缺点

① 只能顺序访问，效率低

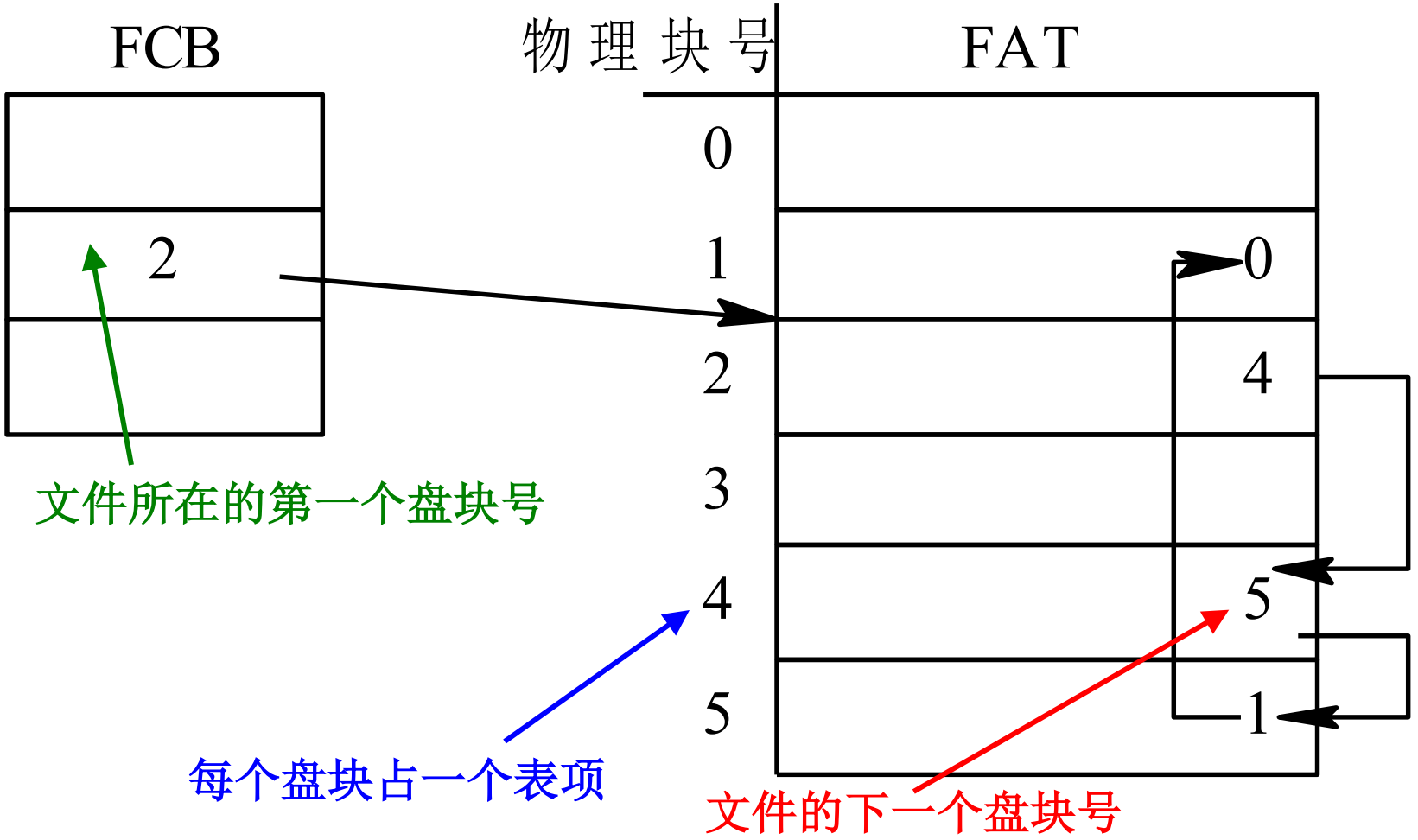
② 可靠性低，某个指针出问题，文件会丢失信息

- 解决方法
  - 以簇(包含若干相邻盘块)为单位管理，而不是以单个盘块为单位
  - 优点
    - 减少查找时间，减少指针所占存储空间
  - 缺点
    - 内部碎片会增大，改进有限

② 显式链接

- 基本思想

把各指针显式地存储到内存的文件分配表 (FAT)中

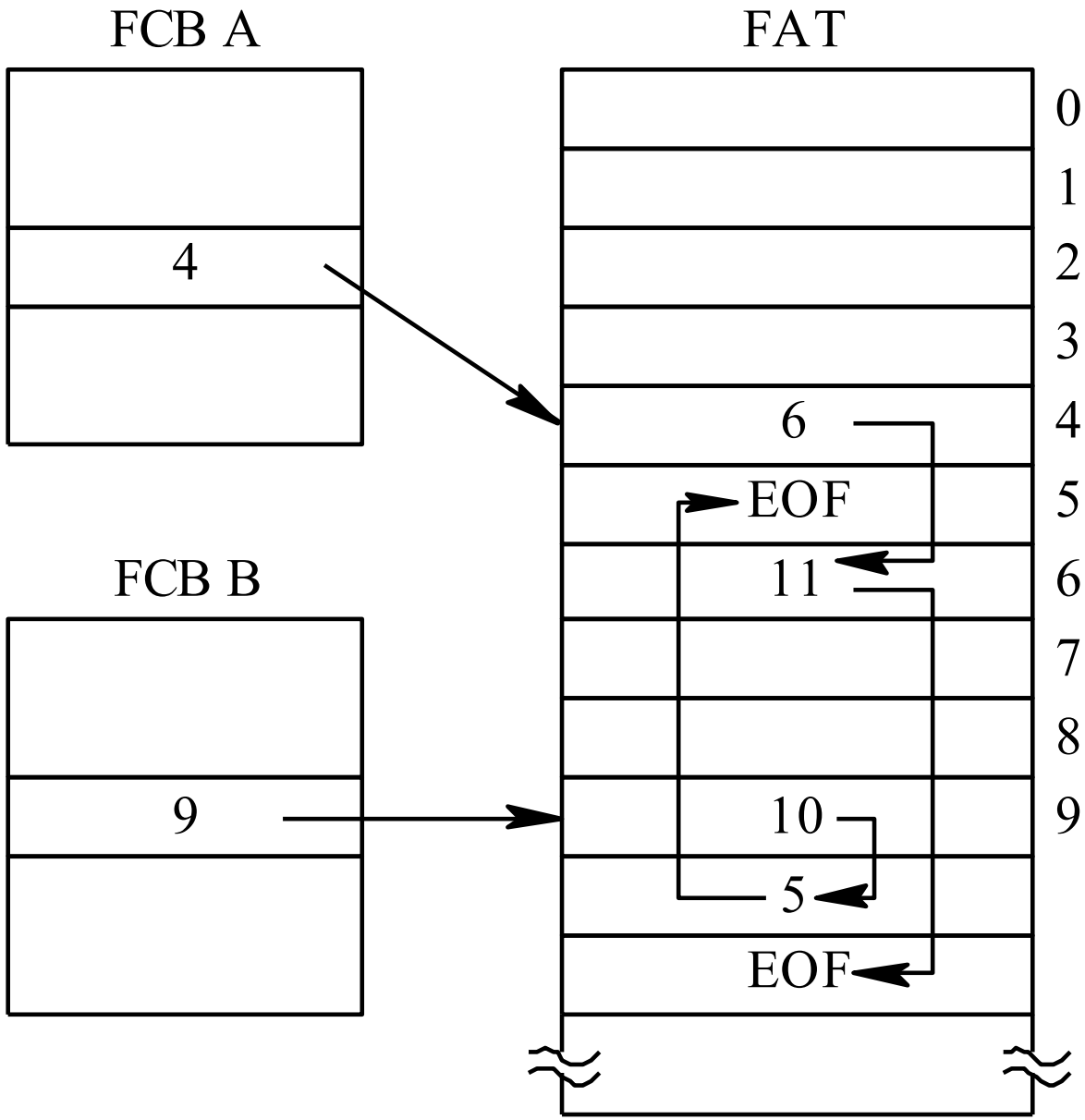


微软的FAT格式简介

- 微软早中期推出的操作系统一直采用FAT技术
- FAT的格式有多种，最常见的是FAT12、FAT16和FAT32。

1. FAT12

- MS-DOS中，最早使用12位的FAT12格式
- 以盘块为单位建立文件分配表（FAT表），来记录每个文件中所有盘块之间的链接关系
- 为保证可靠性，每个系统分区配有两张FAT表


FCB A
FAT
4
0
1
2
3
6
4
EOF
5
FCB B
11
6
7
8
9
10
9
5
EOF

- 一张FAT12的表占多大的磁盘空间？
  - 对于1.2MB的软盘，盘块大小为512B，则每个FAT表包含2.4K个表项 (1.2MB = 512B x 2.4K)
  - 由于每个表项占12位，所以FAT表占用3.6KB的存储空间 (2.4K x 12b = 3.6KB)

- **FAT12最大可以表示多大的磁盘空间？**
  - 用12位表示表项数量，所以共 2^{12}（ 即4096 ）个表项
  - 因采用盘块为基本单位，每个盘块一般为512B
  - 所以，每个逻辑分区的容量最大不能超过

    4096 X 512B = 2MB
  - 一个磁盘支持4个逻辑分区，所以使用FAT12的磁盘最大容量不能超过8MB

- **FAT12是否能支持超过8MB的磁盘呢？**
  - 可以
  - 采用簇为单位
  - 一个簇可以包括1个盘块、2个盘块、4个盘块、8个盘块 等等
  - 一个簇到底包含多少盘块与磁盘容量有关

- 以簇为单位管理的优点
  - 能适应磁盘容量的不断增大
  - 减少FAT表中的表项，使FAT表占用更少的磁盘空间
  - 减少访问FAT表的开销
- 以簇为单位管理的缺点
  - 造成簇内碎片

- **FAT12存在的主要问题**
  - 随着支持磁盘容量的增加，簇也不断增大，导致更大的簇内碎片
  - 只能支持短文件名：8字符文件名、3字符扩展名

2. FAT16

- 最多允许有2^{16}（即65536）个表项
- 故而可以将一个逻辑分区分成65536个簇
- FAT16中簇的大小可以是：4、8……、64个盘块
- 因此，FAT16可以管理的空间最大为

  65536 X 64 X 512B = 2GB

- FAT16也存在类似的问题，要支持超过2GB的逻辑分区，需要更大的簇，而簇内碎片也会增大

3. FAT32

- FAT系列文件系统的最后一个产品
- FAT表中用32位（即4个字节）表示一个簇，最多允许有2^{32}个表项
- 簇的大小固定为4KB（即每个簇大小为8个盘块）
- FAT32可以管理的空间最大为 2TB (= 4KB x 2^{32})

索引组织方式

- 事实上，打开文件时，不必将整个FAT调入内存，只需调入该文件对应的盘块号即可。
- 因此，应将每个文件对应的盘块号集中放在一起。
- 在访问到该文件时，将其所对应的盘块号一起调入内存。
- 所以，提出了“索引组织”的方法

- 基本思想
  - 为每个文件建立一个索引块(表)，把该文件占用的所有盘块号记录在该索引块中
  - 所以，可以把该索引块看做一个包含很多盘块号的数组
  - 在建立文件时，需在相应的目录项中，添加指向索引块的指针

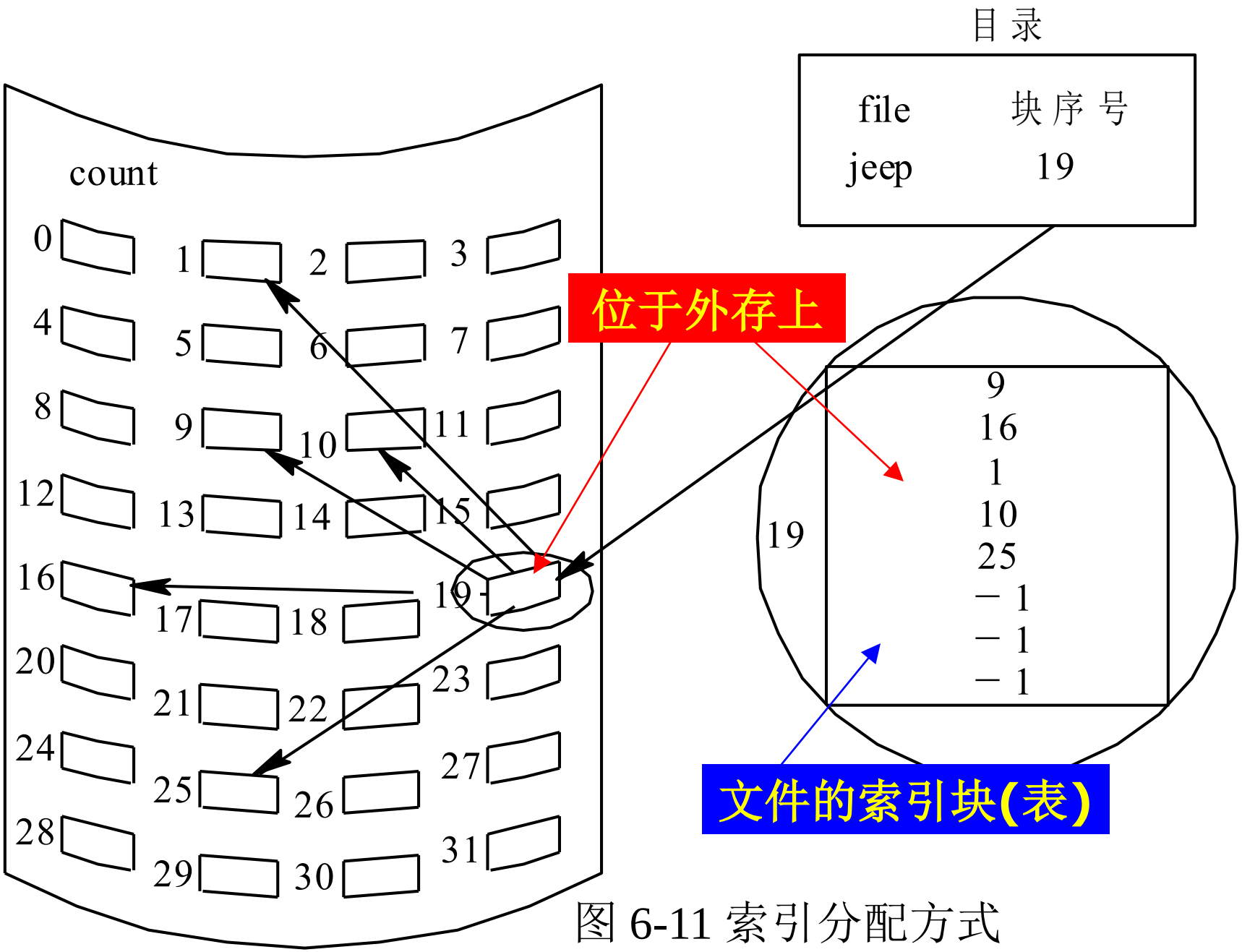
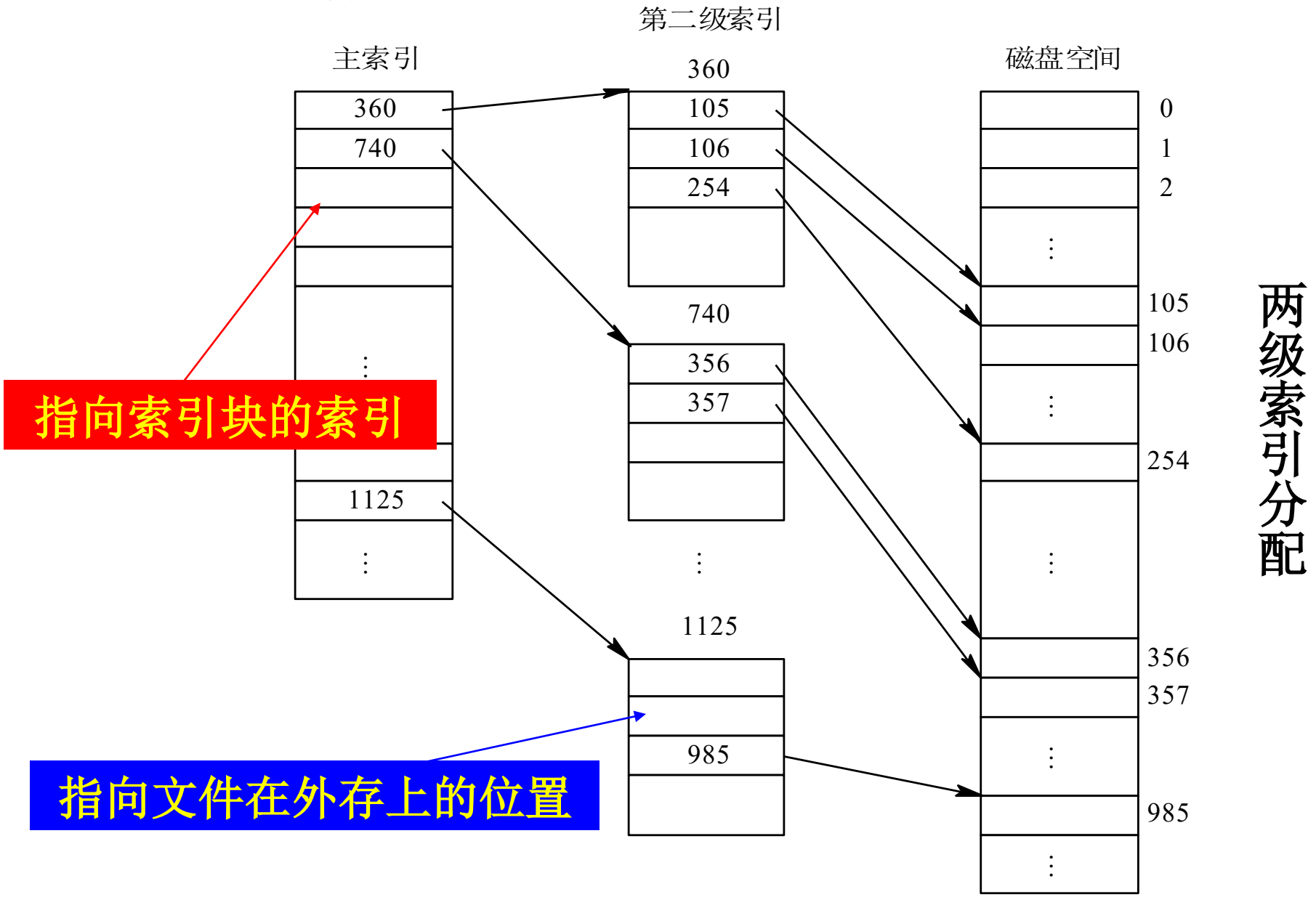

目录
file
块序号
jeep
19
count
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
位于外存上
19
9
16
1
10
25
－1
－1
－1
文件的索引块(表)

图 6-11 索引分配方式

- 优点
  - 支持直接访问，不产生碎片
  - 对于大文件来说，优于链接分配方式
- 缺点
  - 对于小文件来说，索引块利用率极低

- 多级索引组织方式

适用于大文件



两级索引分配

8.2 文件存储空间的管理

- 为给文件分配存储空间，需要知道磁盘上哪些盘块可以分配
- 所以，OS应为可分配存储空间设置相应的数据结构

1. 空闲表法

- 属于连续分配方法
- 类似于内存的动态分配方式

序号	第一空闲盘块号	空闲盘块数
1	2	4
2	9	3
3	15	5
4	—	—

空闲盘块表

2. 空闲链表法

① 空闲盘块链

- 将所有空闲空间，以盘块为单位链成一个链
- 优点：分配和回收单个盘块很简单
- 缺点：为一个文件分配盘块时，可能重复操作多次；空闲盘块链会很长。

② 空闲盘区链

- 将所有空闲盘区(可包含多个盘块)链成一个链

3. 位示图法

- 基本思想

  利用二进制的一位表示一个盘块的使用情况

  分别用“0”、“1”表示“未分配”或“已分配”

① 位示图

- 通常用m×n个位数来表示

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	0	0	0	1	1	1	0	0	1	0	0	1	1	0
2	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1
3	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0
4																
⋮																
16																

位示图

② 盘块的分配

① 顺序扫描位示图，从中找出一个或一组其值为“0”的二进制位 (“0”表示空闲)

② 将所找到的一个或一组二进制位， 转换成与之相应的盘块号。

假定找到的其值为“0”的二进制位，位于位示图的第i行、第j列，则其相应的盘块号应按下式计算：

$$b=n(i-1)+j \ ,$$

其中n代表每行的位数

③ 修改位示图， 令map［i,j］=1

③ 盘块的回收

① 将回收盘块的盘块号转换成位示图中的行号和列号。 转换公式为：

$$i = (b-1) \text{ DIV } n + 1$$

$$j = (b-1) \text{ MOD } n + 1$$

② 修改位示图:

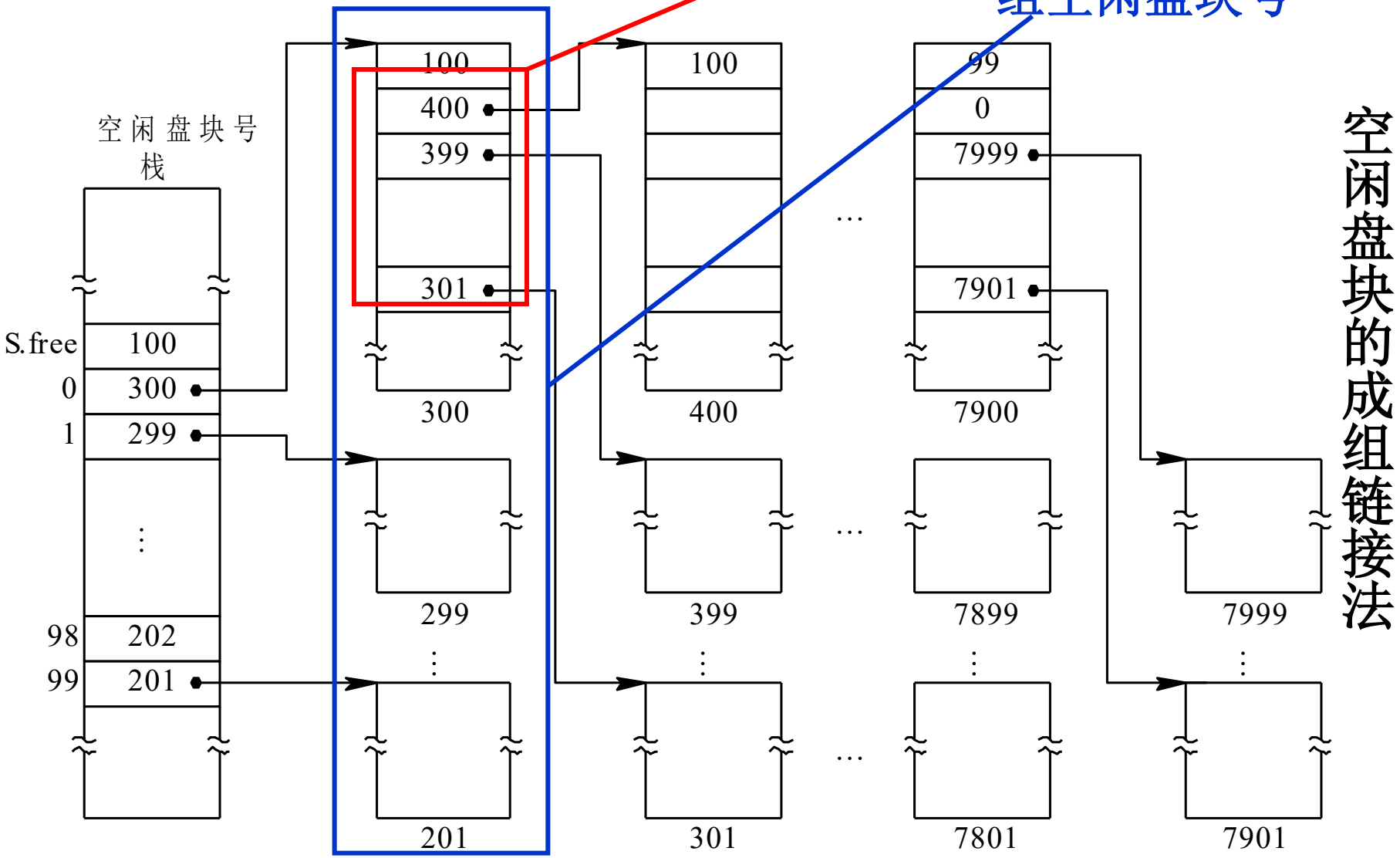
令 map［i,j］=0

④ 位示图的优点

① 很容易找到一个或一组相邻接的空闲盘块

② 位示图很小，占空间少

4. 成组链接法


指针区（下一组的盘块号）
一组空闲盘块号
空闲盘块号栈
S.free
0
1
98
99
100
300
299
202
201
100
400
399
301
300
299
201
100
399
301
400
99
0
7999
7901
7900
7899
7801
7999
7901
空闲盘块的成组链接法

- 空闲盘块的分配

① 检查“空闲盘块号栈”是否上锁，如未上锁，便从栈顶取出一个空闲盘块号，将对应的盘块分配给用户，将栈顶指针下移。

② 若该盘块已是栈底，由于该盘块号对应的盘块中记有下一组可用的盘块号。因此，须将栈底盘块号对应盘块内容读入栈中，作为新的盘块号栈，并把原栈底对应的盘块分配出去

③ 分配一个相应的缓冲区 (作为该盘块的缓冲区)

④ 把栈中的空闲盘块数减1并返回

- 回收过程如下：

① 将回收盘块的盘块号压入“空闲盘块号栈”的顶部，并将空闲盘块数加1。

② 当栈中空闲盘块号数目已达100时，表示栈已满，便将当前栈中的100个盘块号，记入新回收的盘块中，再将其盘块号作为新栈底。

8.3 提高磁盘I/O速度的途径

- 访问文件的速度是文件系统性能的重要指标
- 提高磁盘I/O速度是提高文件访问速度的一种重要手段

1. 磁盘高速缓存

- 在内存中开辟一个区域，用于某些磁盘盘块的信息
- 目的：提高磁盘I/O的速度。
- 它逻辑上属于磁盘，物理上驻留在内存中

- 磁盘高速缓存的应用
  - 当有请求要访问磁盘时，先查看要访问的盘块内容是否已在磁盘高速缓存中
  - 如果在，便可从磁盘高速缓存中访问（节省了访问磁盘的开销）
  - 如果不在，再去真正的访问磁盘，同时把本次所需的盘块内容存入磁盘高速缓存中，以备以后使用

- 设计磁盘高速缓存时，需考虑以下几个问题

① 如何将磁盘高速缓存中的数据传送给请求进程；

② 采用什么样的置换策略；

③ 何时将被修改的盘块内容写回磁盘（保证数据一致）。

- 数据交付方式
  - 将磁盘高速缓存中的数据传送给请求进程的方式
  - 两种方式：

① 数据交付：直接将磁盘高速缓存中的数据传送给请求进程（传数值）

② 指针交付：将指向磁盘高速缓存中保存数据的区域的指针交给请求进程（传地址）

- 后一种方式由于所传送的数据量少，而节省了实际传送数据的时间

- 置换算法
  - 如同请求调页(段)一样，将磁盘中的盘块数据读入磁盘高速缓存时，同样会出现因其中已装满数据而需要将某些数据换出的问题。
  - 相应地，也必然存在着采用哪种置换算法的问题。
  - 较常用的置换算法仍然是LRU等。

– 由于请求调页中的快表与磁盘高速缓存的工作情况不同，因而使得在置换算法中所应考虑的问题也有所差异。

– 现在不少OS在设计磁盘高速缓存的置换算法时，除了考虑到“最近最久未使用”这一原则外，还考虑了以下几点：

① 访问频率

② 可预见性

③ 数据的一致性：内存、磁盘内容的一致

- 有些OS将磁盘高速缓存中的盘块组成一个LRU链
- 将那些会严重影响数据一致性的盘块数据和很久都可能不再使用的盘块数据，放在链的头部
- 使之能优先被写回磁盘，以减少发生数据不一致的概率

- 周期性地写回磁盘
  - 根据LRU算法，经常被访问的数据可能会一直保留在磁盘高速缓存中，长期不会被写回磁盘。这可能因突然出现的系统故障，导致这些数据丢失。
  - UNIX专门增设一个“修改程序”，使之在后台运行，该程序周期性地调用SYNC系统调用。
  - SYNC主要功能：强制将所有在磁盘高速缓存中已修改的数据写回磁盘。

2. 提高磁盘I/O速度的其它方法

① 提前读

② 延迟写

③ 优化物理块的分布

④ 虚拟盘(RAM盘)

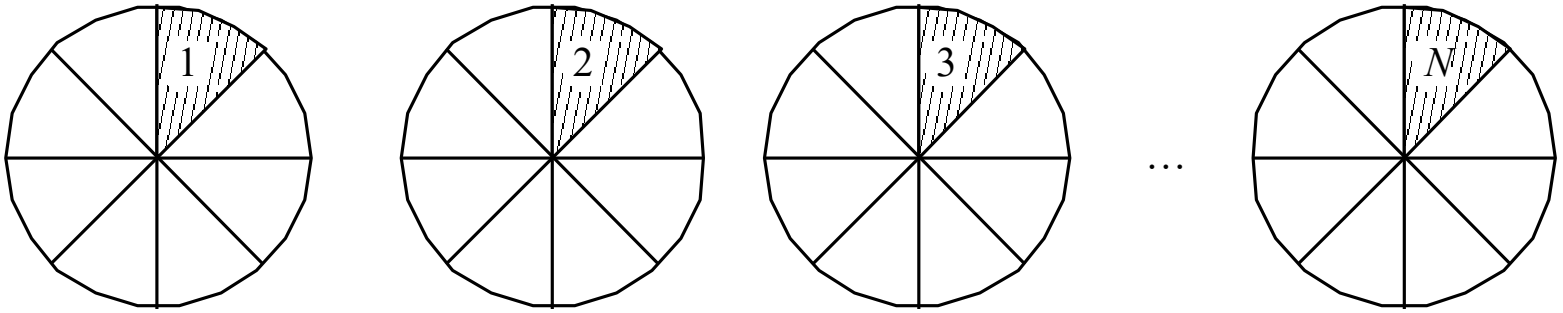
同学们自己回去看！

3. 廉价磁盘冗余阵列(RAID)

- 提出时间：1987年
- 提出者：加利福尼亚大学伯克莱分校
- 目的：
  - 利用一台磁盘阵列控制器，统一管理和控制一组(几台到几十台)磁盘驱动器，组成一个高度可靠、快速的大容量磁盘系统
- 应用：
  - 广泛应用于大型系统中

磁盘阵列柜

- 并行交叉存取
  - 目的：提高对磁盘的访问速度
  - 方法：将每个数据块分成若干子块，分别存储在各个不同磁盘中的相应位置上
  - 优点：并行存取



- **RAID的分级**
  - RAID刚推出时，被分成6级(0级—5级)，后来又增加了6级和7级。
  - 级别越高，功能越强，性能越好
  - 当然，级别越高，造价也越高。
  - 具体应用时，可根据实际情况来选购。

- **RAID的优点**

① 可靠性高

- 这是RAID最大的特点，除0级以外，其他各级都采用容错技术

② 磁盘I/O速度高

- 采用并行交叉存取技术，可并行存取信息，极大提高了磁盘I/O的速度

③ 性能/价格比高

8.4 提高磁盘可靠性的技术

- 磁盘容错技术的目的
  - 防止因系统因素（硬件故障等）导致的文件和数据丢失

1. 第一级容错技术SFT-Ⅰ

- 目标
  - 防止磁盘表面缺损造成的数据丢失

① 双份目录和双份文件分配表

– 为防止文件目录和文件分配表FAT被破坏，而造成的数据丢失。在不同的磁盘上或磁盘的不同区域中，分别建立(双份)目录表和FAT。

– 其中，一份被称为主目录、主FAT； 把另一份称为备份目录、备份FAT。

② 热修复重定向和写后读校验

- 热修复重定向
- 写后读校验方式

同学自己看看！

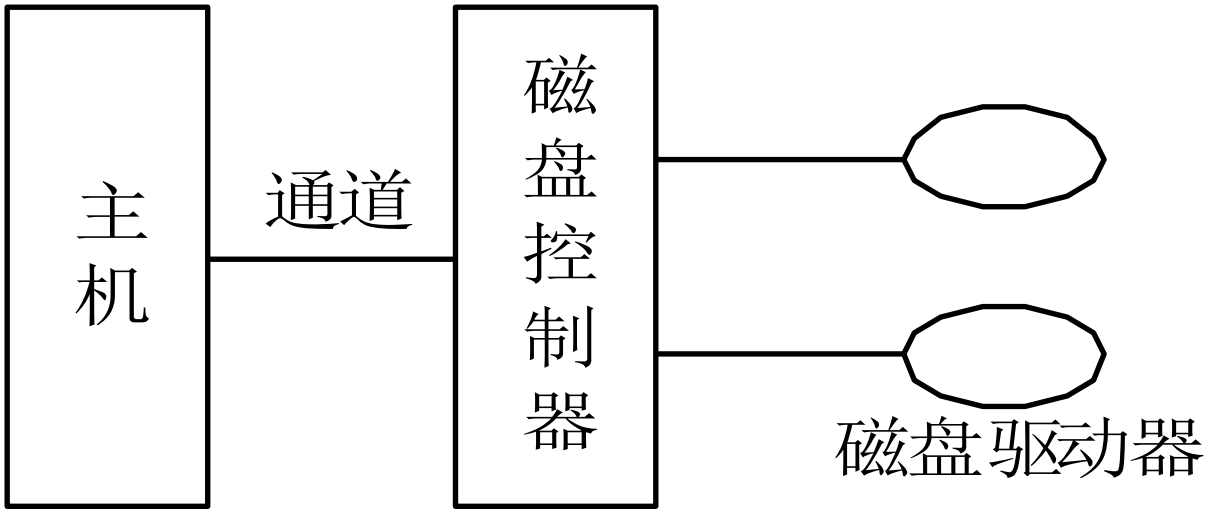
2. 第二级容错技术SFT-II

- 目标：防止磁盘驱动器或控制器损坏导致系统工作异常

① 磁盘镜像

目标：防止磁盘驱动器损坏导致系统工作异常

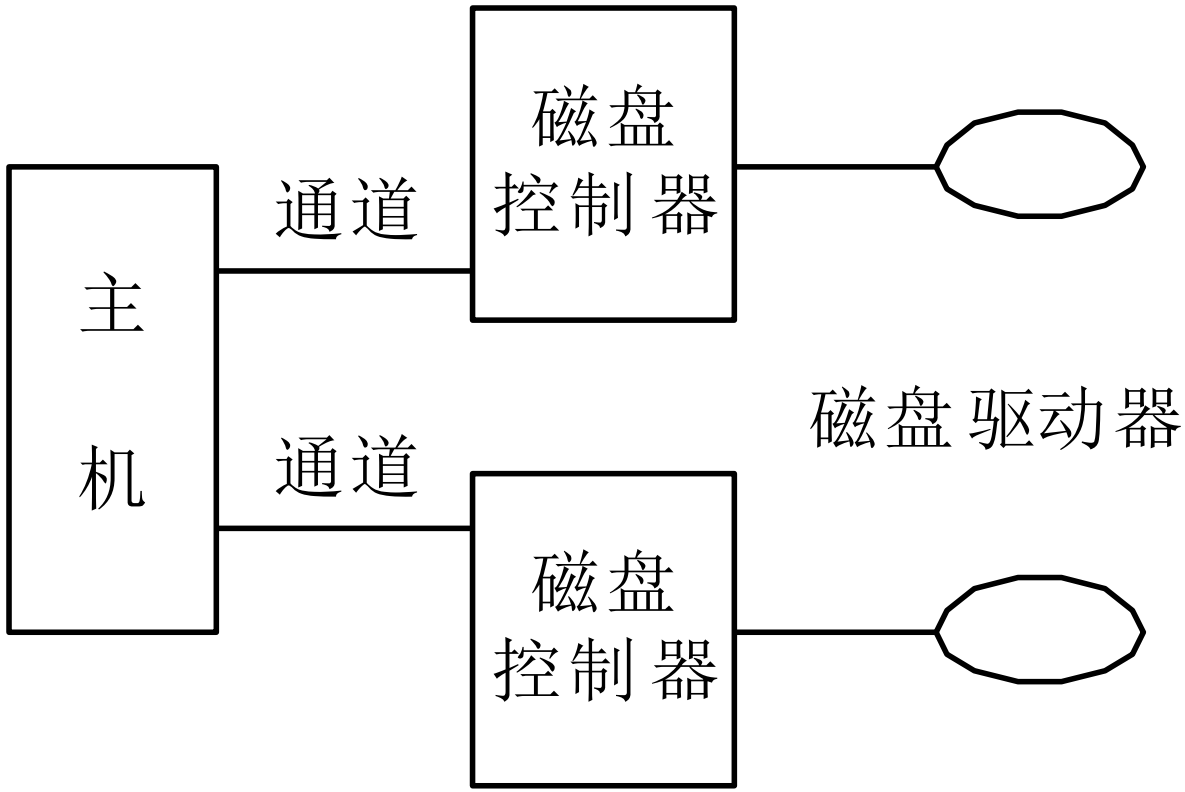
方法：写主磁盘时，也写备份磁盘



磁盘镜像示意图

② 磁盘双工

- 目的：防止磁盘控制器损坏导致系统工作异常



磁盘双工示意图

3. 基于集群技术的容错功能

- 双机热备份模式
- 双机互为备份模式
- 公用磁盘模式

4. 后备系统

同学们自己看！

文件系统安全性小结

- 为确保文件系统的安全性，可采取以下措施

① 通过存取控制机制，防止人为因素造成的文件不安全性

② 通过磁盘容错技术，防止磁盘部分故障造成的文件不安全性

③ 通过后备系统，防止自然因素造成的不安全性

8.5 数据一致性控制

1. 什么是事务（举例说明）

- 在银行转账业务中，假设从账户A转账10000元到账户B
- 系统在执行这项业务时，实际是从账户A减去10000元，再在账户B中加上10000元
- 如果从账户A减去的操作正常执行了，但在账户B中加上金额的操作因系统故障未能执行
- 这样，会导致客户账户总额中有10000元不翼而飞的错误。
- 那么，如何解决这个问题呢？

- 银行转账这类问题，实际都有一个共同特点
  - 执行一项任务需要多步操作，而这些操作要么全部执行（任务顺利完成），要么全都不执行（任务未能完成，但不会导致数据错误）
- 在OS看来，这些操作都是相互独立的，彼此之间没有任何关系
- 如何让OS知道这些操作是在执行同一项任务呢？
- 也即，如何让OS知道它们构成一个不可分割的整体，形成“原子操作”？
- 事务处理就是为了实现这个目的而提出的

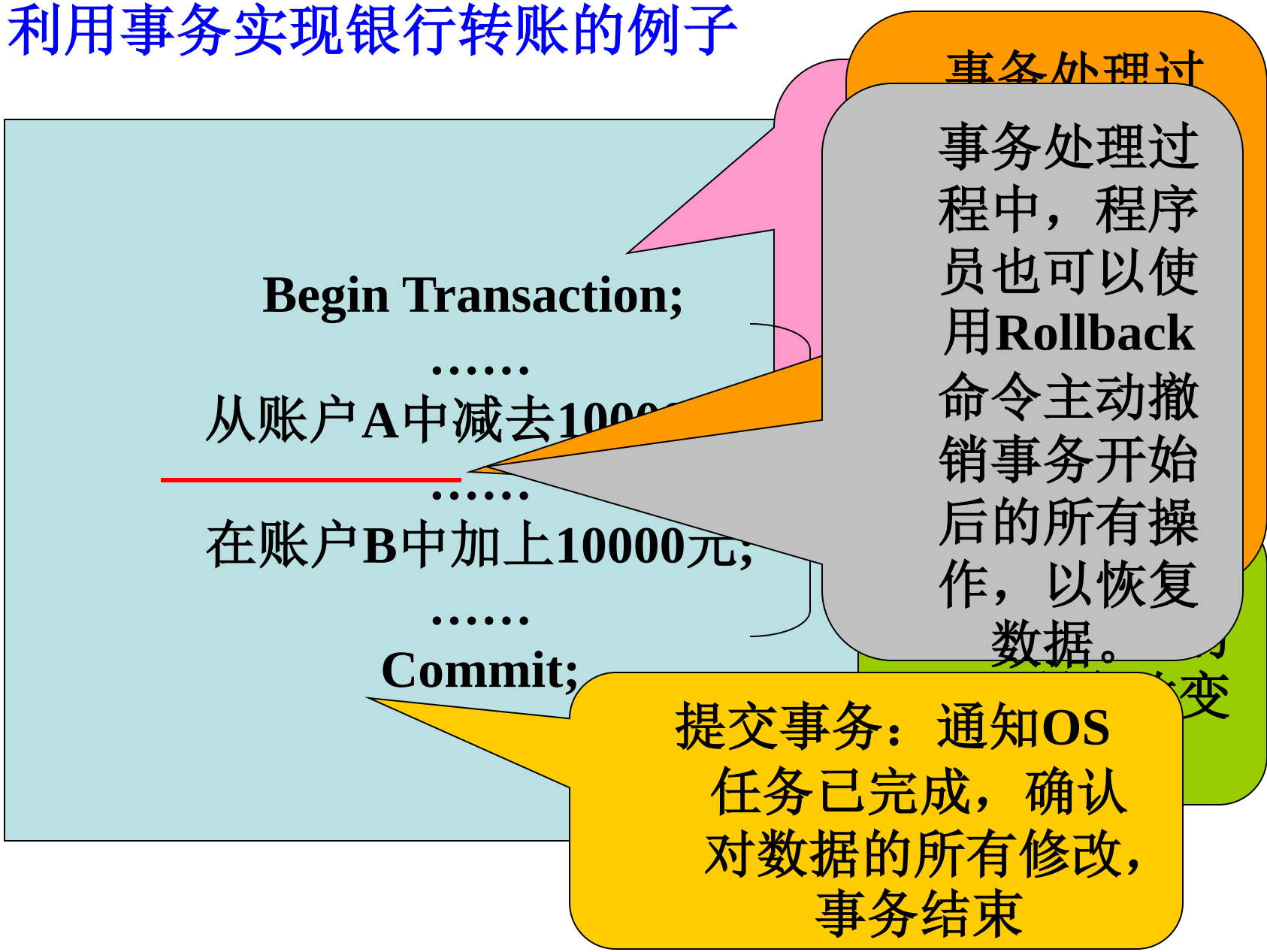
- 为实现事务处理，**OS**必须提供相应的功能：

① 开始事务 **Begin Transaction**

② 提交事务 **Commit**

③ 回滚事务 **Rollback**

- 利用事务实现银行转账的例子



- 事务的实现
  - 为了实现“原子性”，须借助称为事务记录的数据结构。这些数据结构放在稳定存储器中，以记录在事务运行时数据项修改的全部信息，又称运行记录(Log)。
  - 在事务T开始执行的时候，<T开始>记录被写到事务记录表中；
  - T执行期间，任何写操作之前，会写一条适当新记录到Log中；
  - T进行提交时，把<T提交>记录写入Log中

- 恢复算法
- OS发生故障时，数据恢复的实现需要利用以下两个过程：

① undo

– 该过程把所有被事务修改过的数据，恢复为修改前的值。

② redo

– 该过程能把所有被事务修改过的数据，设置为新值

- 需要恢复数据时， 系统会对以前事务所发生的操作进行清理：

① 事务T的所有操作都作完了，Log中必含有<T开始>、<T提交>，此时调用redo

② 事务T的部分操作未作完，Log中只含有<T开始>，但必不含有<T提交>，此时调用undo

2. 检查点

- OS发生故障时，必须检查整个Log表，以确定哪些事务需要利用redo处理，哪些事务需要undo处理
- 因为OS中存在多个并发执行的事务，所以一旦系统发生故障，在Log表中清理起来比较费时
- 引入的检查点的主要目的就是为了解决上述问题

- 检查点的工作原理是，每隔一定时间便做一次下述工作
  ① 将驻留在内存中的当前Log中的所有记录保存到外存
  ② 将驻留在内存中所有已修改数据保存到外存
  ③ 在Log中写入一条〈检查点〉记录
  ④ 每当出现一个〈检查点〉记录时，OS便执行恢复数据的操作，利用redo和undo实现恢复功能
- OS出现故障后，不需要对所有事务进行处理
- 只需对最后一个检查点之后的事务进行处理，执行相应的redo和undo

3. 并发控制

- 大家回去自己看！

练习题

1. 位示图可用于__B__

A. 文件目录的查找　B. 磁盘空间的管理

C. 主存空间的共享　D. 实现文件保护和保密

2. 事务具有__C__，须借助__C__实现

A. 分散性、硬件　B. 分散性、事务记录

C. 原子性、事务记录　D. 原子性、硬件