# 操作系统

主讲教师:范国祥

电　话：0451-86418876-811（O）
13199561265（Mobile）
E-mail：fgx@hit.edu.cn

软件学院教研室

2016.06

# 第12章 文件系统

## 主要内容

12.1 文件的目录结构

12.2 文件系统的实现

12.3 MINIX文件系统1.0实现

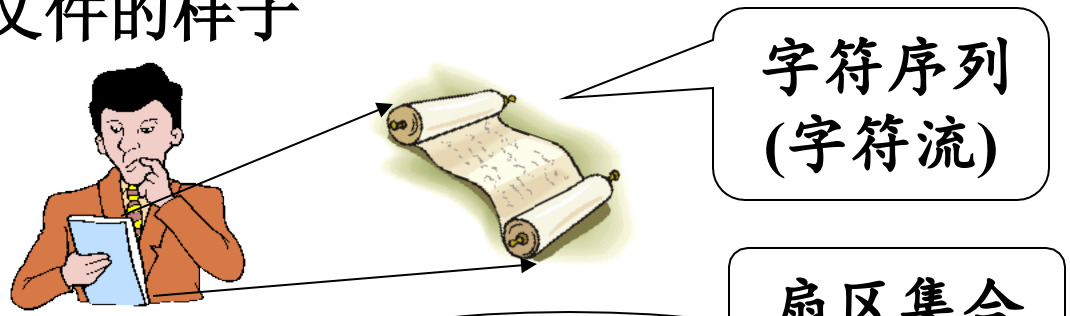
12.4 Windows的FAT文件系统实现

# 12.1 文件的目录结构

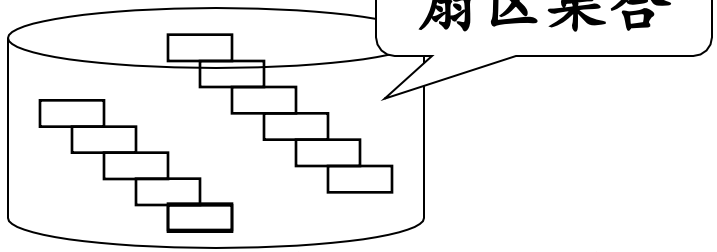
- 文件的引入回顾
- 文件盘块的三种分配方式回顾
- 文件的目录结构

# 文件的引入

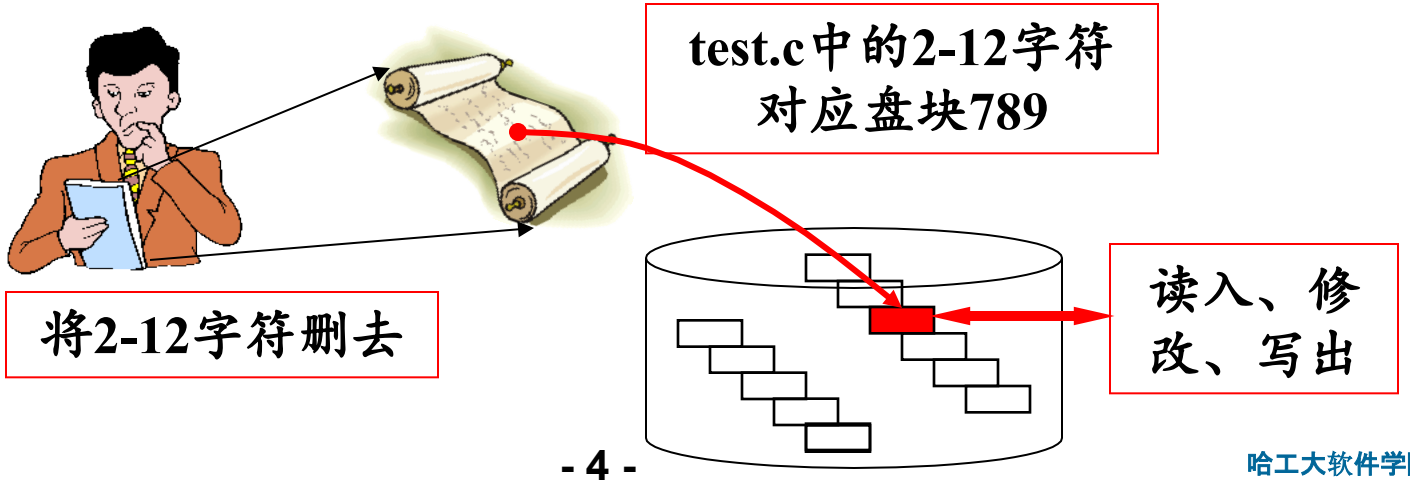
- 用户眼里文件的样子




- 磁盘上的文件的样子

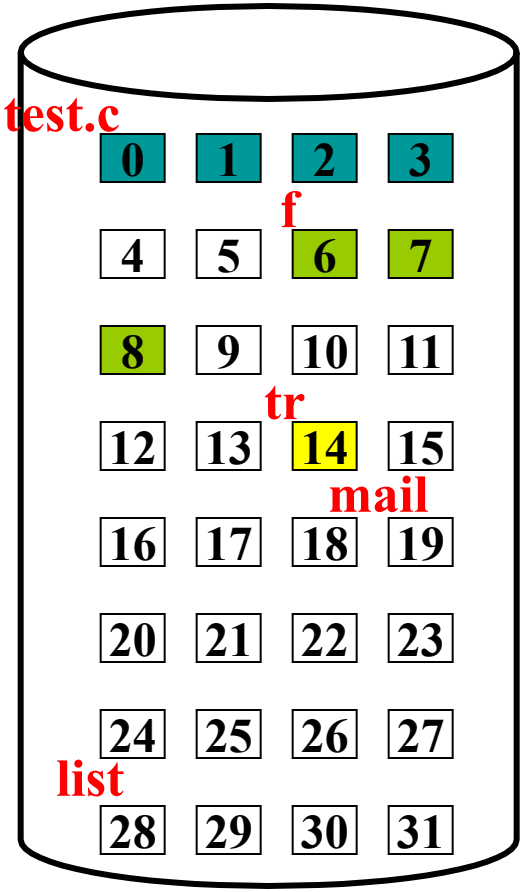



- **文件: 建立了字符流到盘块集合的映射关系**

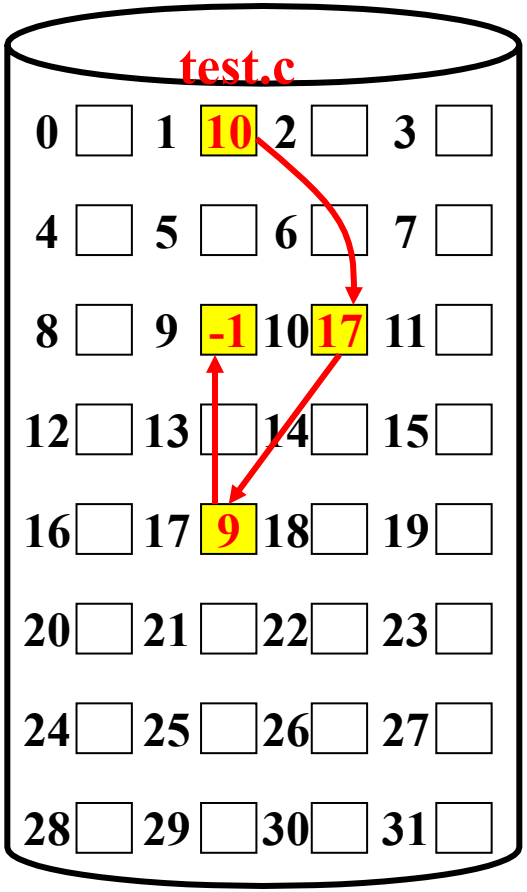


# 三种基本映射关系

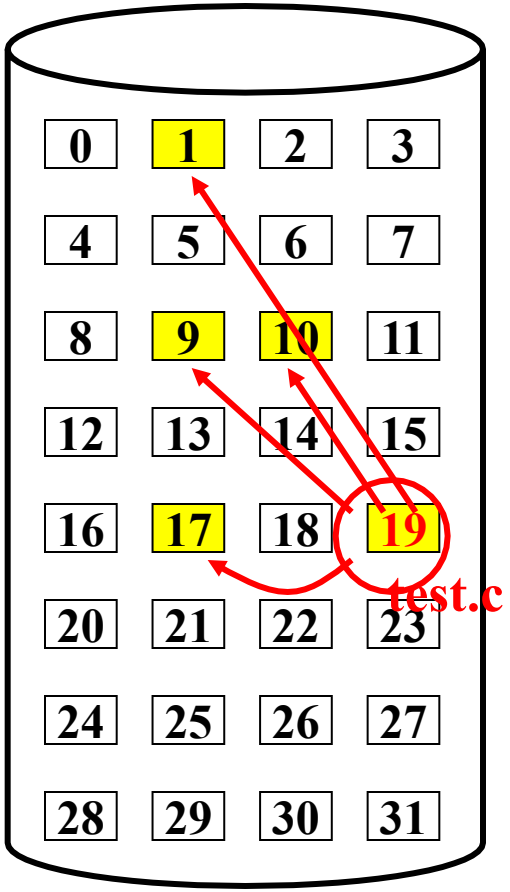
## 盘块连续分配




## 盘块链式分配

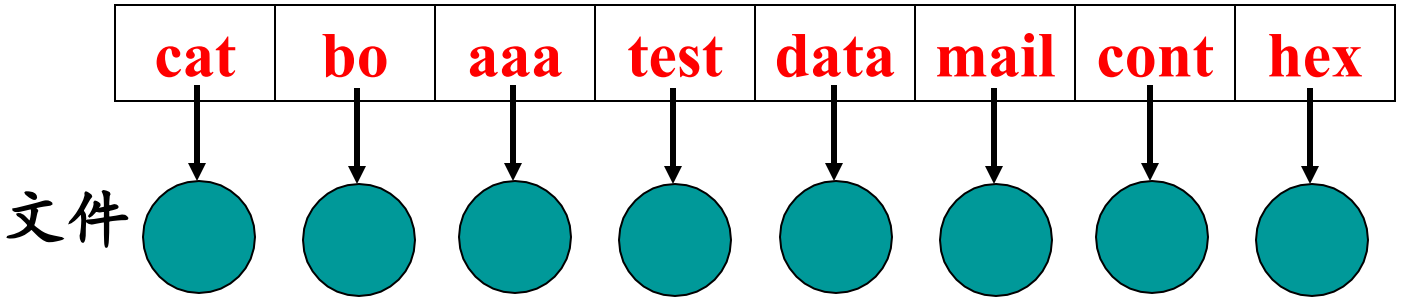



## 盘块索引分配

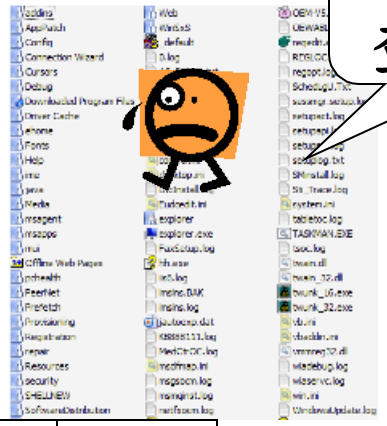


# 文件和文件系统的差别在哪里？

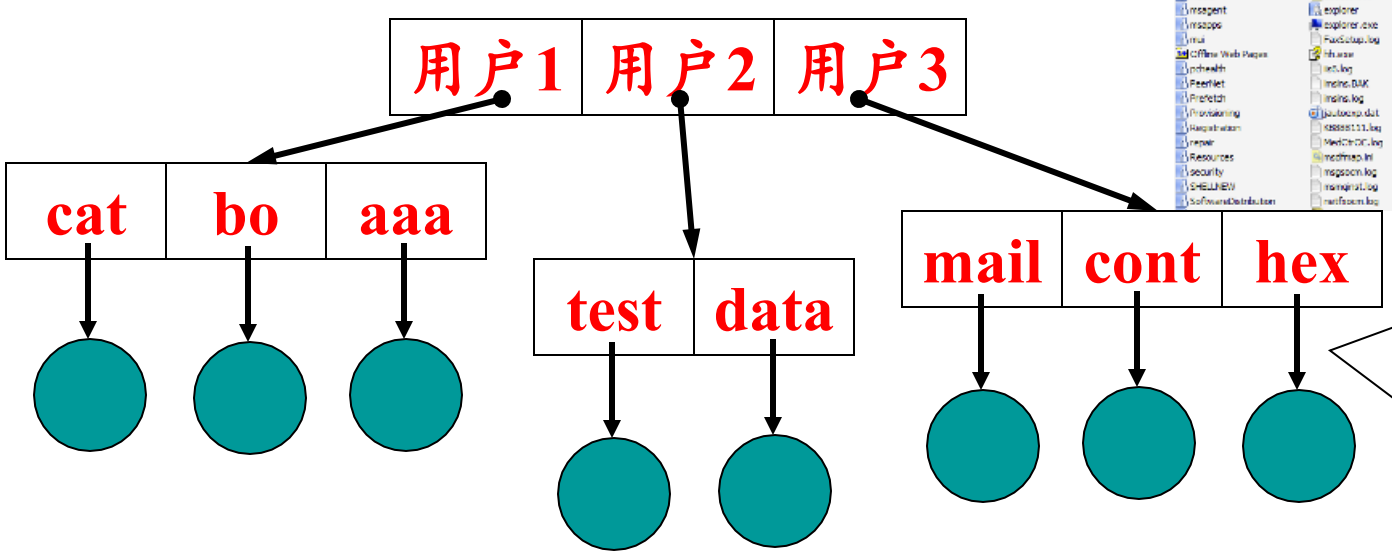
# 文件系统中有很多文件 怎么管理?




- 所有文件放在一层(一个大集合)
- 怎么办? 集合划分



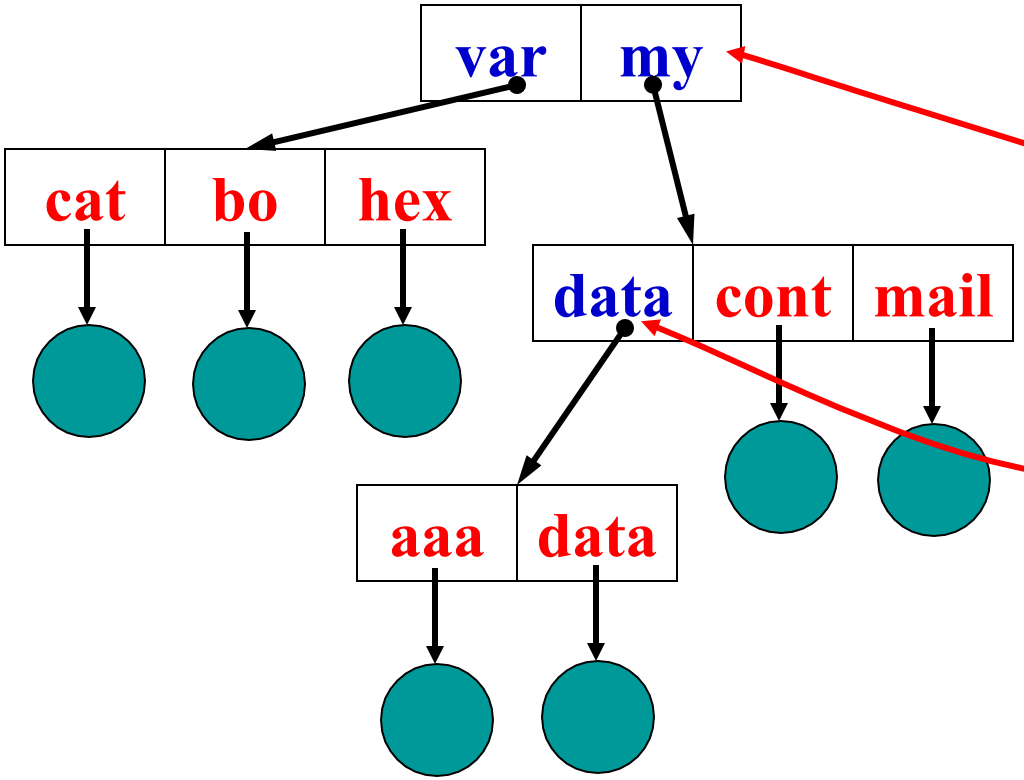
枯燥乏味
查找困难




问题依然存在: N/U个, 扩展性仍然差!

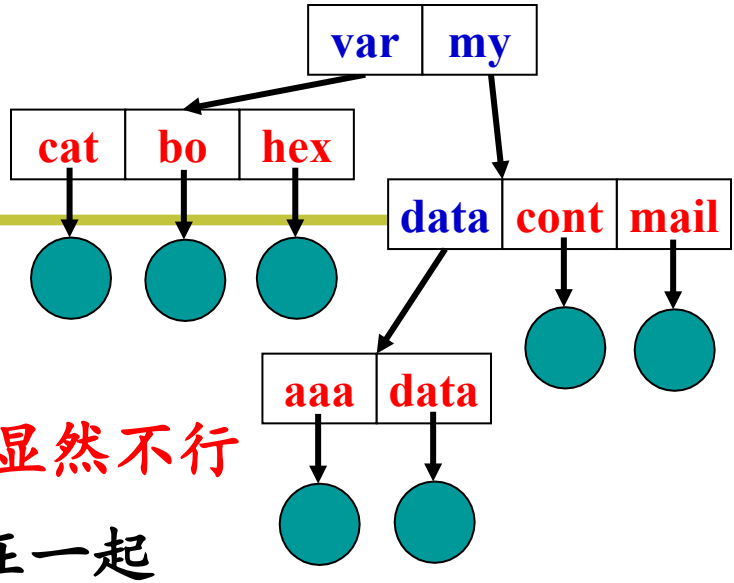
# 划分的基础上继续划分 树状目录

- 将划分后的集合再进行划分
- k次划分后，每个集合中的文件数为$O(\sqrt[k]{N})$




- 此种树状结构扩展性好、表示清晰，最常用
- 描述“文件的集合”，需要引入概念：目录
- 怎么进行“文件集合”物理表述？

# 目录的实现




- 存放“文件集合”
  - 将文件内容(盘块)放在一起… 显然不行
  - 将文件内容指针(即文件头)放在一起
  - 应该是可以的，取决于文件系统如何处理…
- 思考: 有了树状目录后会出现什么问题?
  - 出现了路径名: /my/data/aaa用来定位文件aaa
  - 路径名 ⇒ 路径的解析:

再根据文件头定位文件内容

输入/my/data/aaa，获得文件aaa的文件头

# 路径的解析(Name Resolution)

想一想?

- 输入/my/data/aaa，获得文件aaa的文件头
  - 从哪里开始? 顶层目录(根目录/)
  - 目录是什么? 应该也是一个文件，文件存放的内容是该目录中所有文件的文件头!
  - 解析/my/data/aaa :

文件系统中全是文件!

放在磁盘确定位置，可在OS初始化时读入

```
res("/my/data/aaa")
 {   fh = FileHeader("/");  //根目录的文件头
     data = ReadData(fh); fh = Find(data,"my");
     data = ReadData(fh); fh = Find(data,"data");
     data = ReadData(fh); fh = Find(data,"aaa");
     return fh;
 }
```

# 从路径解析来看目录内容

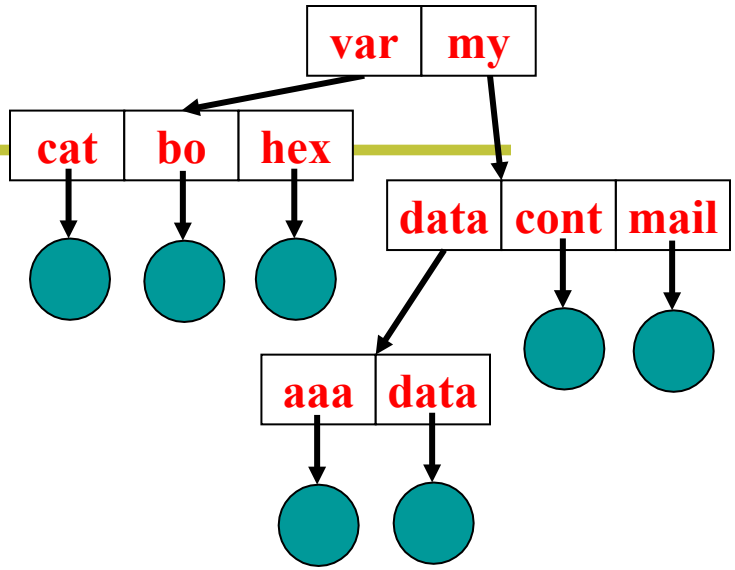
- 显然路径解析的使用频率高，因此效率很重要
  - 如何提高路径解析的效率?
  - 要使语句data=ReadData(fh);fh=Find(data,"??");
    要想效率高，data应该尽可能短
  - 所以目录文件中不应该存放文件头，应该存放指向文件头的指针
  - 文件头指针? 可将文件头连续存放(形成了数组)在磁盘的固定位置，文件头指针就是其数组项标号!

文件头尺寸也并不小

任何文件的文件头结构相同

已知基址、偏移就能找到文件头!

# 树状目录的完整实现




磁盘

文件头数组 | 数据盘块集合

“/”的文件头

根目录文件的数据块

<var, 13> | <my, 82>

目录项!

文件头数组索引号!

“my”的文件头

“my”目录文件的数据块

<data, 103> | <cont, 225> | <mail, 77>

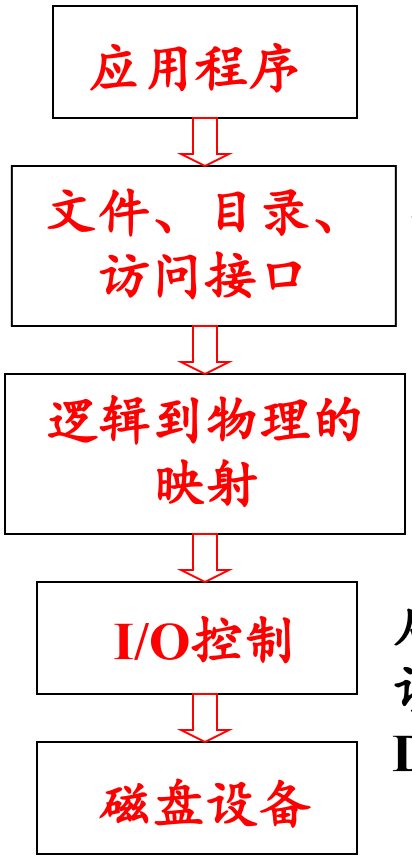
# 12.2 文件系统的实现

- 文件系统定义
- 典型的文件系统结构
- 文件分区空闲块的管理

# 描述文件系统的实质(定义)

就像将CPU资源和地址空间封装成进程一样!

- **文件系统**: 将盘块“变”成文件集，方便用户访问
  - 文件系统是将“抽象盘块”按一定格式包装的一层软件!

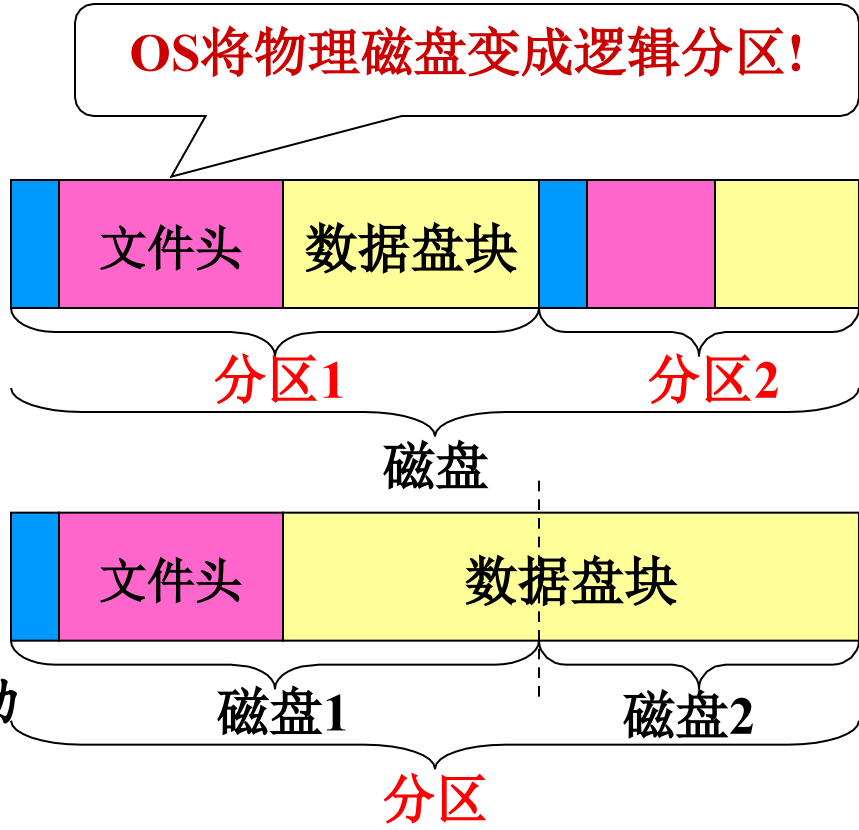



删除test.c 2-12个字节

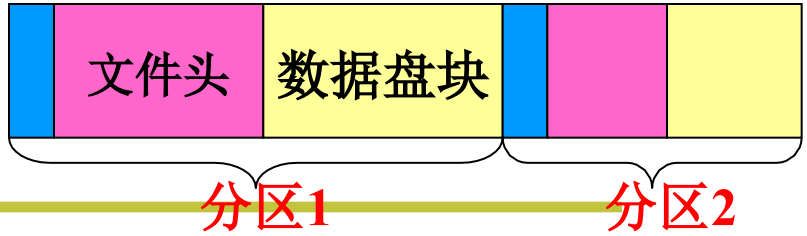
找到test.c的文件头: 需要定位根目录，解析目录...

根据文件头可以找到2-12字节对应的盘块(789)

启动DMA，等待789读入，读入后删除2-12字节再启动DMA写出



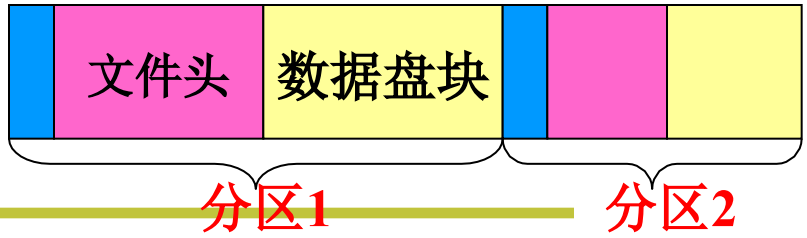
# 分区的详细结构



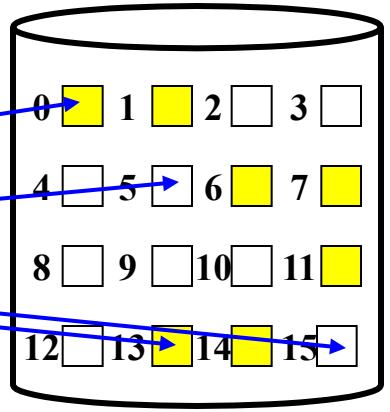

- 典型分区结构：UNIX分区的基本结构


- **引导块**：存放引导OS的信息，如果该分区中没有OS，则该块为空
- **超级块**：记录分区基本信息，即块大小；分区块数；空闲块数量、指针；空闲文件头数量、指针等
- **索引节点数组**：存放所有文件的文件头，UNIX root目录的索引节点号为2
- **数据块**：存放文件内容

# 分区空闲盘块的管理




- 有的盘块被文件使用，其它盘块如何管理?
  - 组织起来等待文件的使用！ 怎么组织?
  - 方法1：空闲位图(位向量)…



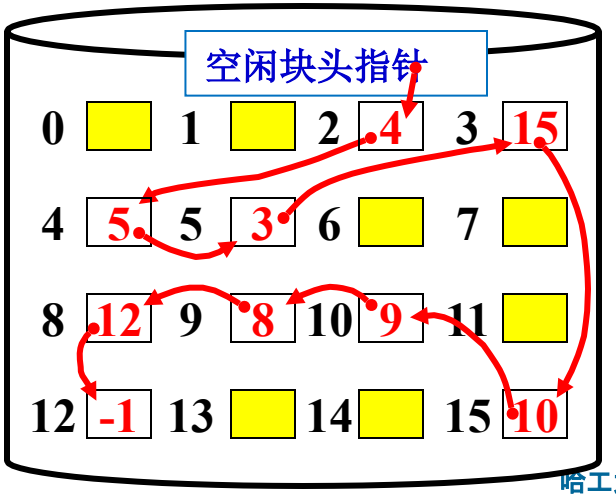

0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 1

表示磁盘块2,3,4,5,8,9,10,12空闲

可快速分配连续盘块组，但位向量很大(1G/512B=?)

  - 方法2：空闲链表

分配一个(或少量的)空闲盘块是可以高效工作，但分配多个则慢！



# 可运转的和良好运转的文件系统!

# 良好运转的文件系统应该高效

- 相比CPU和内存，磁盘读写非常慢!

```
int main(int argc, char* argv[])
{
    int  i , to, *fp, sum = 0;
    to = atoi(argv[1]);
    for(i=1; i<=to; i++)
    {
        sum = sum + i;
        fprintf(fp,“%d”, sum);
    }
}
```

fprintf用一条其他计算语句代替

```
C:\>sum 10000000
0.015000 seconds
```

$0.015/10^7$

用fprintf

```
C:\>sum 1000
0.859000 seconds
```

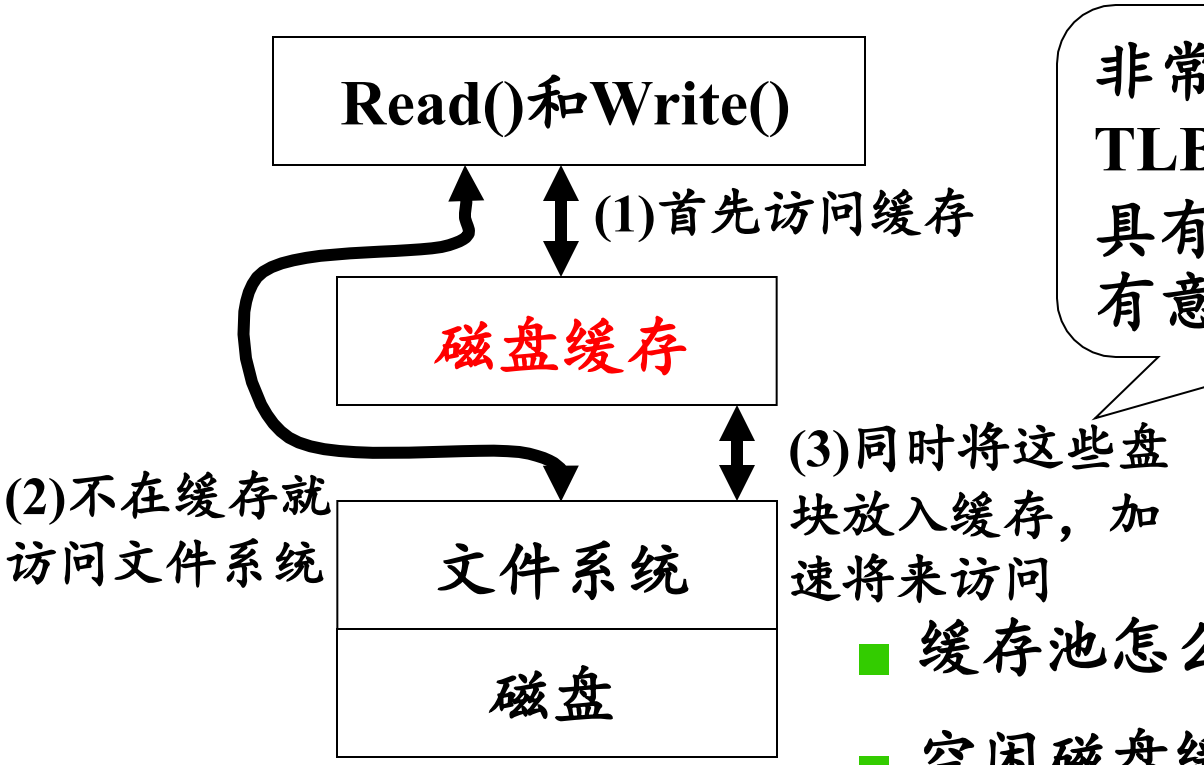
$0.859/10^3$

$5.7\times10^5 : 1$

解决速度差异问题的基本手段是?
引入缓存!(因为局部性)

# 磁盘缓存

- 在内存中缓存磁盘上的部分(很少部分)盘块




- 缓存池怎么组织？ Hashing
- 空闲磁盘缓存用完怎么办？ LRU
- 改写后的文件盘块怎么办？

Delayed Write or Write Through

# 其他的提高文件访问效率的

文件路径解析的根本做法是从“根目录”开始

- 某些目录文件的FCB可以常驻内存
  - /的FCB常驻内存，因为许多目录解析都从此处开始
  - 当前目录(cwd)的FCB可驻内存?
- ls -l的使用非常频繁

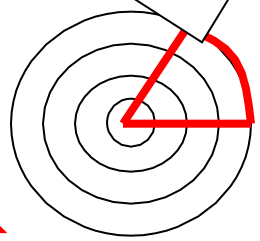
一段时间大多数文件访问集中在一个目录中(局部性)

  - 怎么才能快速执行?

    同一目录中的文件iNode放在一个柱面(组)!

  - 需要重新设计iNode、目录文件分配算法…

每个柱面组都有iNode，空闲盘块…

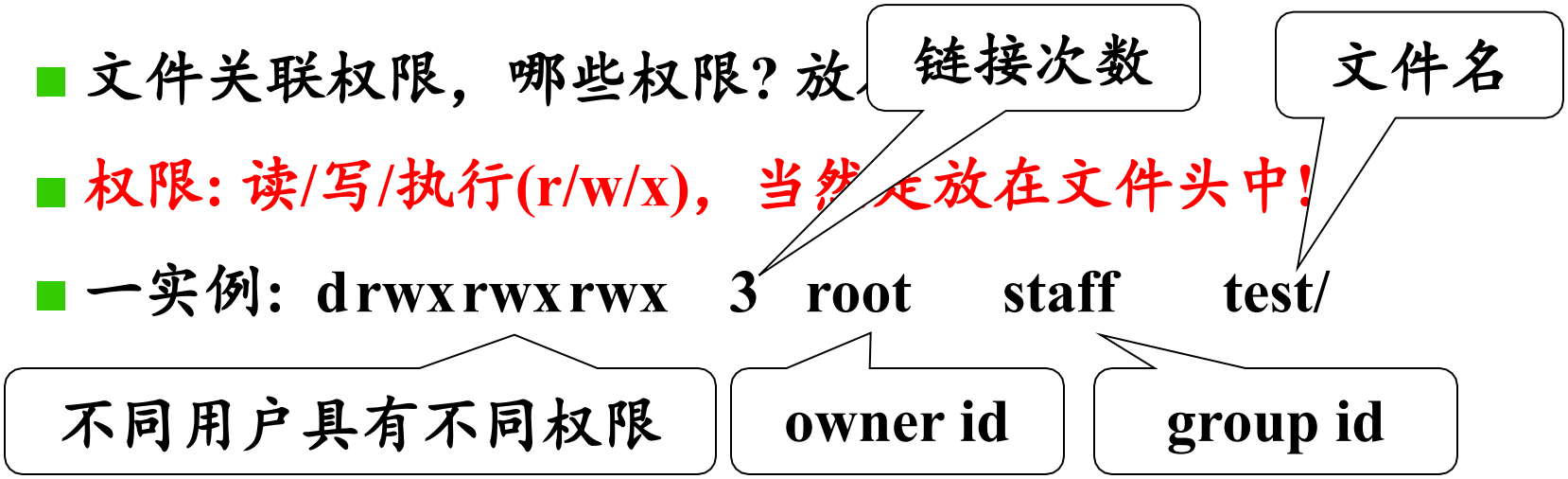


- 显然，还有许多提高文件系统效率的技术……
  - 需要自己去整理，这是操作系统课程显著特点之一

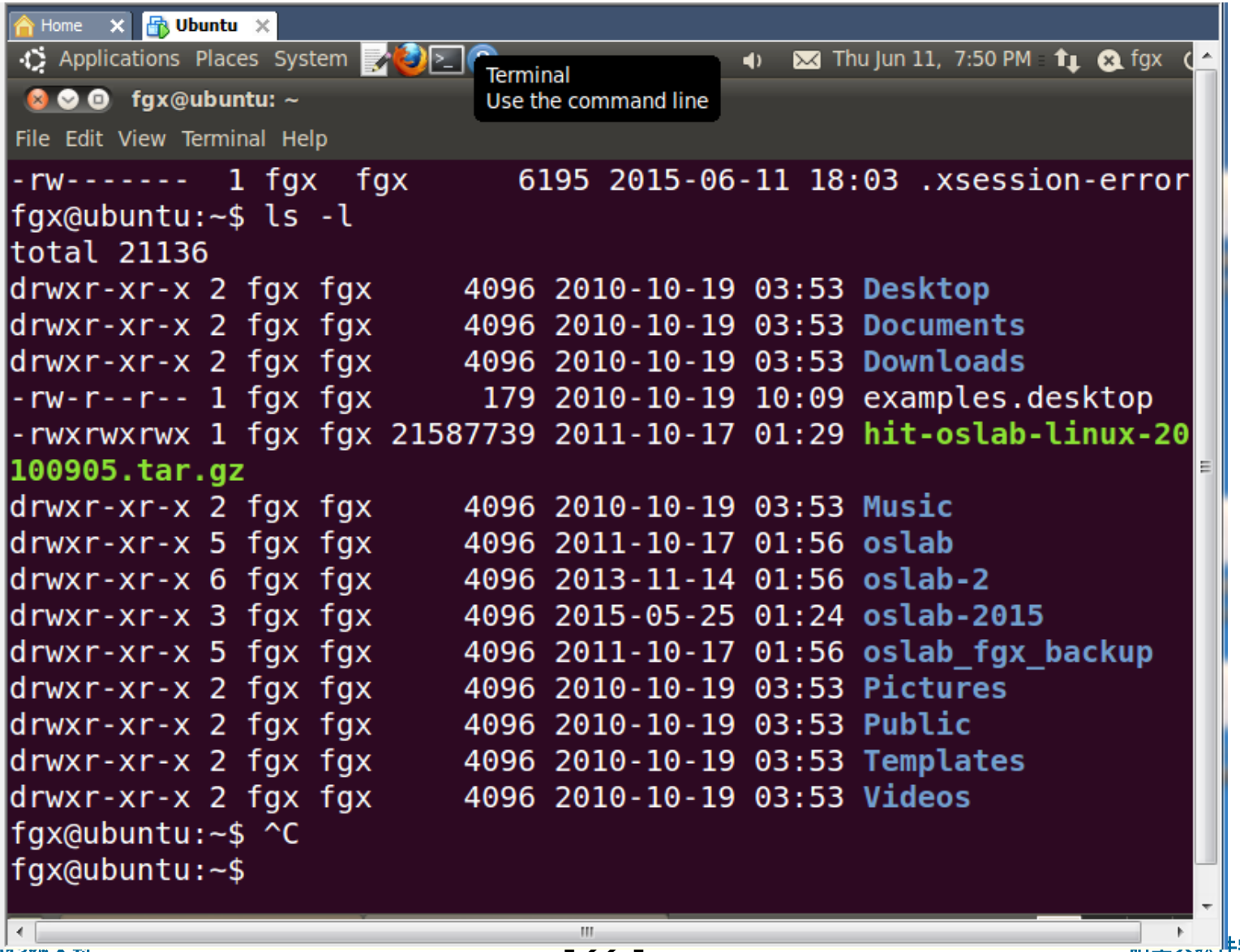
# 良好运转的文件系统应该提供保护

- 文件用来存放用户的信息，用户应该能控制对文件的访问: 如只允许读
  - 文件关联权限，哪些权限？放
  - 权限: 读/写/执行(r/w/x)，当然是放在文件头中！
  - 一实例: drwxrwxrwx 3 root staff test/
  - 如何强制执行(enforce)?
  - 访问文件是由进程发起的，进程是由用户启动的:

    (1)PCB中有uid和gid; (2)fork时设置; (3)这些信息在登录时收集(从tty); (4)文件访问时校验

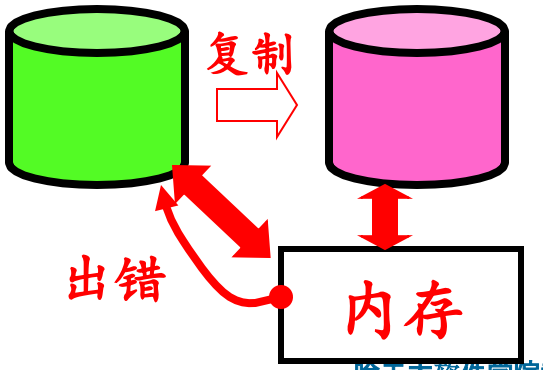


# 良好运转的文件系统应该提供保护



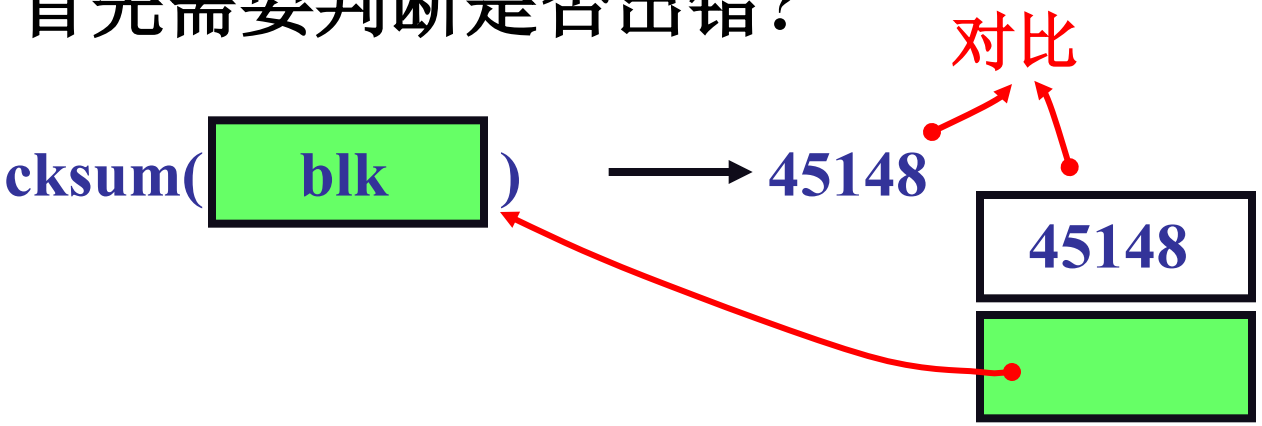
# 良好运转的文件系统似乎也应该容错


- 有过这样的经历?
  - 用户当然不希望: 早晨起来发现写了数月论文打不开了(如因下一块的link断了)...
  - 错误是难免的: 误操作、突然断电、无处不在的电磁干扰... 怎么办? 错误避免还是错误恢复...
  - RAID(Redundant Arrays of Inexpensive Disks)
  - RAID基本思想就是冗余(R): 如在镜像磁盘上并备份数据, 发现错误时拷贝整个磁盘(恢复)

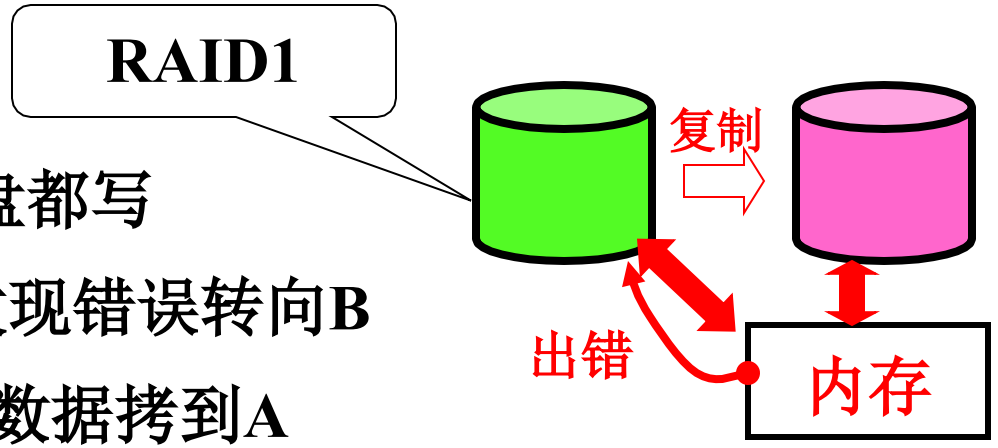


# RAID的简单实现

- 首先需要判断是否出错?




- 此时的磁盘读写




  - (1)每次写两个磁盘都写

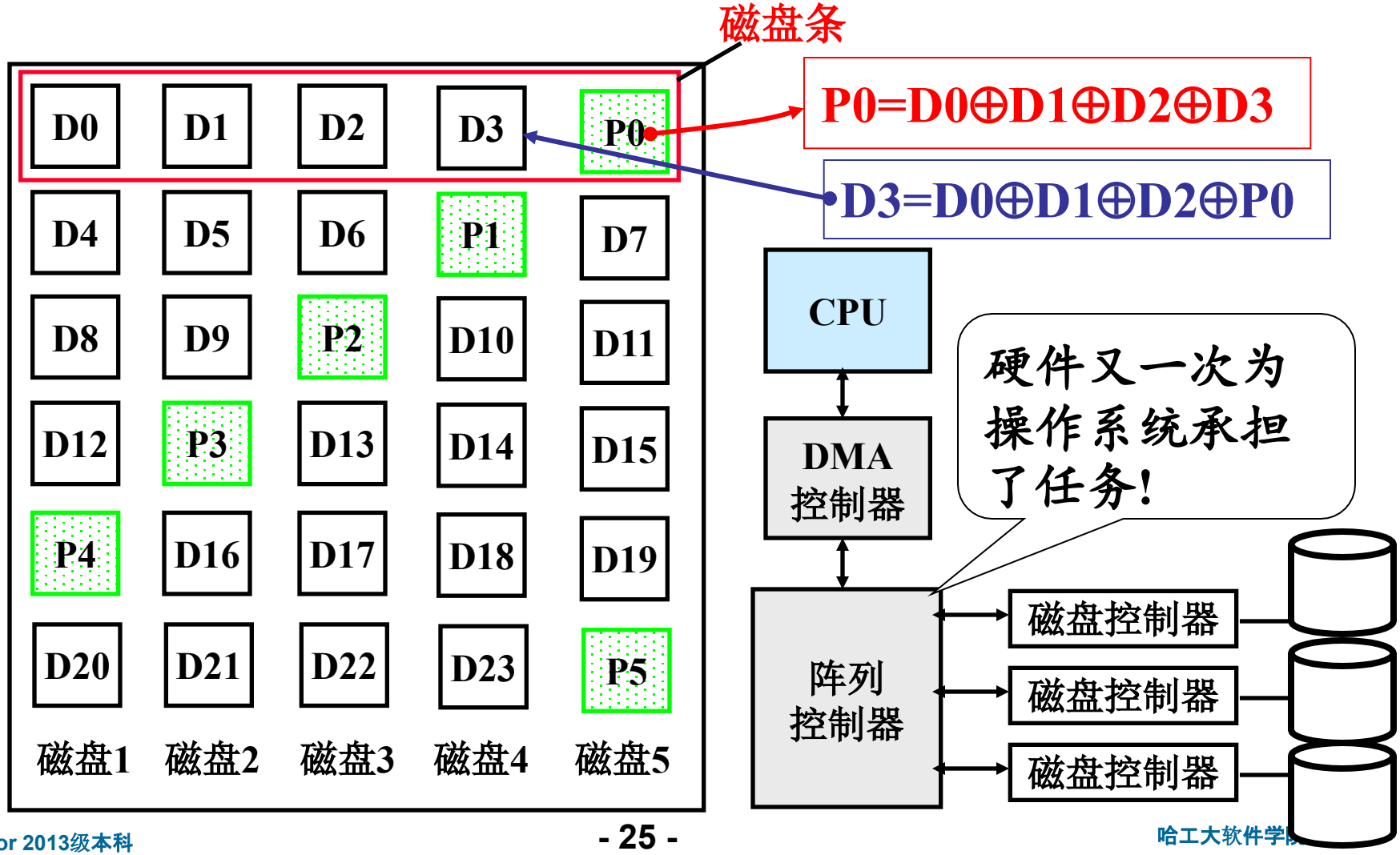
    (2)从磁盘A读，发现错误转向B

    (3)A有错时将B的数据拷到A

  - 磁盘利用率50%

# RAID5+

- 校验数据分布在多个盘上，磁盘互相恢复数据



# 实践一个可实际运转的文件系统！

# 12.3 MINIX文件系统1.0实现

- MINIX文件系统1.0实现