第12章 文件系统

孙承杰

E-mail： sunchengjie@hit.edu.cn

哈工大计算学部人工智能教研室

2023年秋季学期

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

文件和文件系统的差别在哪里?

- 从字面上理解：文件系统-管理文件的系统
- 为文件的各类操作和存储（增删改查）提供简单、高效、安全和可靠的管理功能。

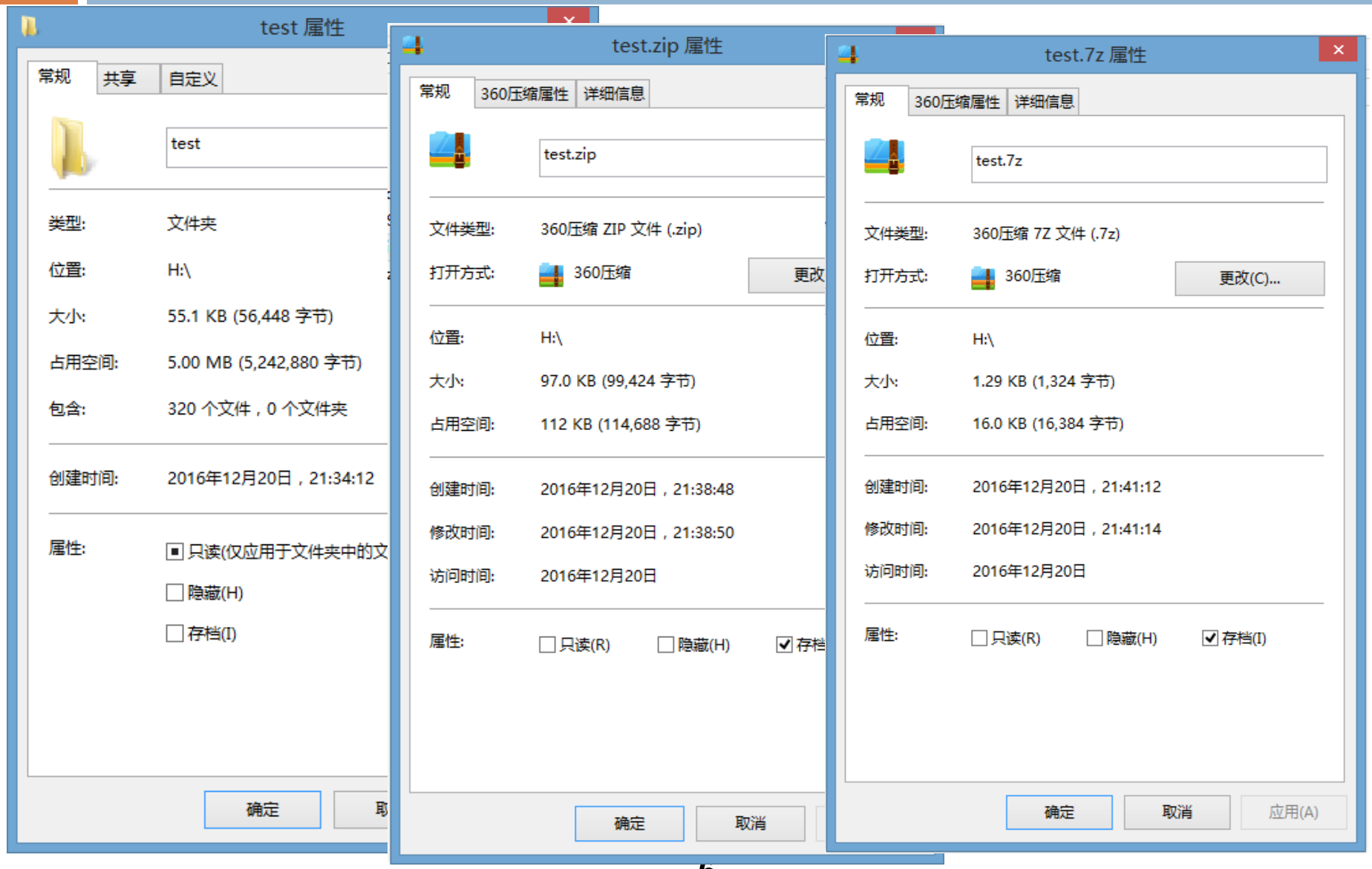
思考几个小问题

- 拷贝一个1G的文件vs拷贝1M个1k的小文件?
- 删除一个1G的文件vs删除1M个1k的小文件?
- 拷贝无数个空的文件夹速度怎么样?
- 无数个小文件总大小和占用磁盘空间大小不一致?
- 压缩后文件大小和占用磁盘空间大小更接近?
- 磁盘寻道和旋转耗时，优化方法?
- 同时剪切很多文件：1）存放到当前磁盘分区某个文件夹中；2）存放到其他磁盘分区某个文件夹中
- 针对磁盘的分区或U盘进行格式化是干什么?
- 磁盘碎片清理是干什么?

思考几个小问题


驱动器工具
此电脑
文件
计算机
查看
管理
BitLocker
优化
清理
格式化
自动播放
弹出
结束刻录
擦除此光盘
保护
管理
介质
优化
优化驱动器有助于提高电脑的运行效率。
文档
图片
视频
音乐
OneDrive
此电脑
网络
音乐
桌面
设备和驱动器 (9)
本地磁盘 (C:)
72.0 GB 可用，共 99.9 GB
本地磁盘 (D:)
60.5 GB 可用，共 73.3 GB
工作文章 (E:)
74.8 GB 可用，共 99.9 GB
各种项目 (F:)
30.4 GB 可用，共 99.9 GB
教学科研 (G:)
83.6 GB 可用，共 165 GB
各种软件 (H:)
83.8 GB 可用，共 126 GB
教学视频 (I:)
96.2 GB 可用，共 131 GB
娱乐生活 (J:)
110 GB 可用，共 133 GB
DVD 驱动器 (L:) OFFICE14
0 字节 可用，共 1.50 GB
UDF
基本
465.76 GB
联机
(D:)
73.33 GB NTFS
状态良好 (活动, 主分区)
各种软件 (H:)
126.86 GB NTFS
状态良好 (逻辑驱动器)
教学视频 (I:)
131.85 GB NTFS
状态良好 (逻辑驱动器)
娱乐生活 (J:)
133.72 GB FAT32
状态良好 (逻辑驱动器)
未分配
主分区
扩展分区
可用空间
逻辑驱动器

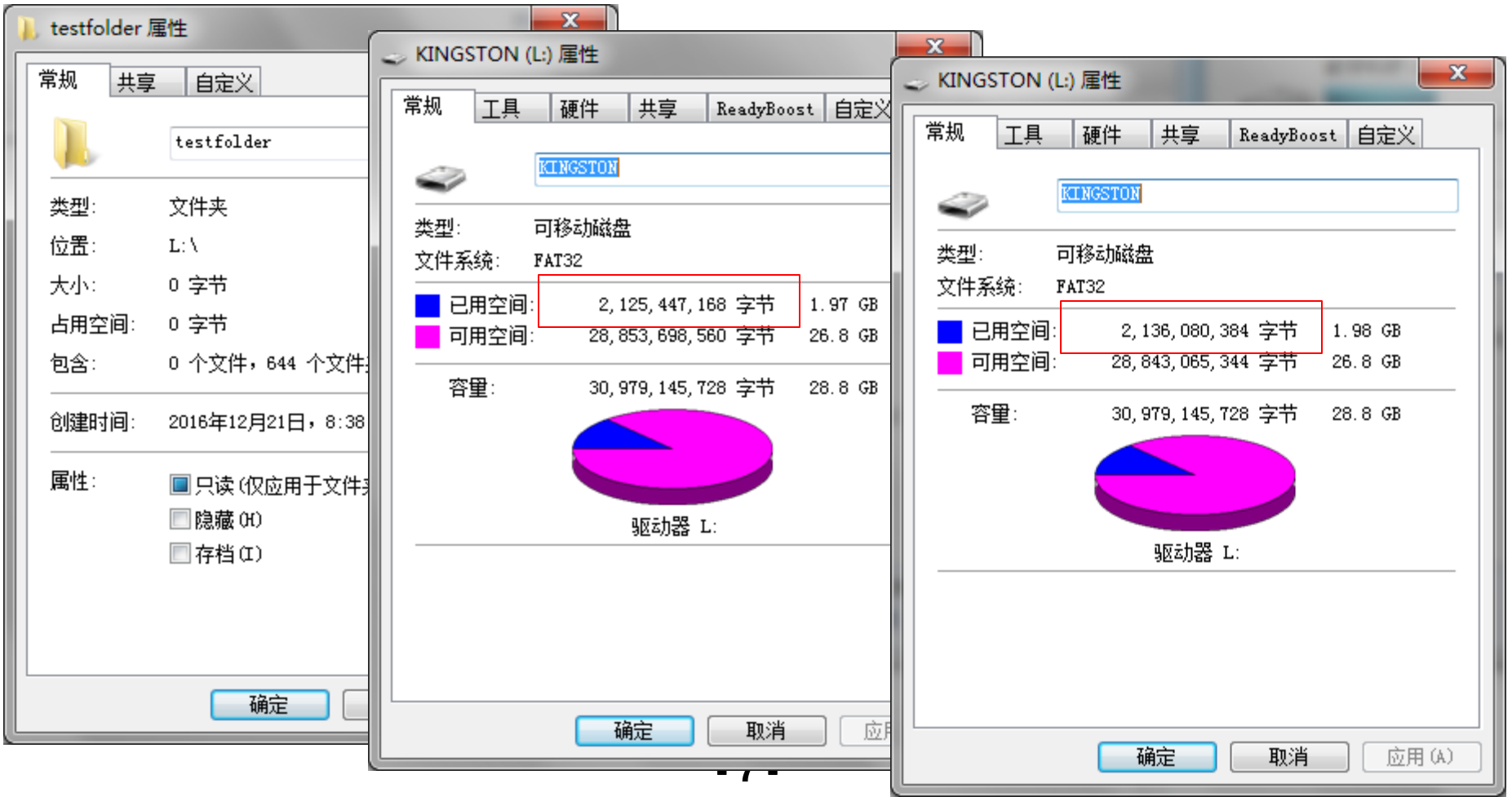
FAT32优盘小文件测试



FAT32优盘文件夹测试

- FAT32优盘建立644个空文件夹

一个空文件夹：（36080384-25447168）/644/1024=16k



磁盘优化包括哪些方面

1、如何存储?

增删改查的灵活性和速度、空间利用率

2、如何访问?

寻道和旋转：扇区→盘块→预读→缓存，多个磁盘请求要调度优化。

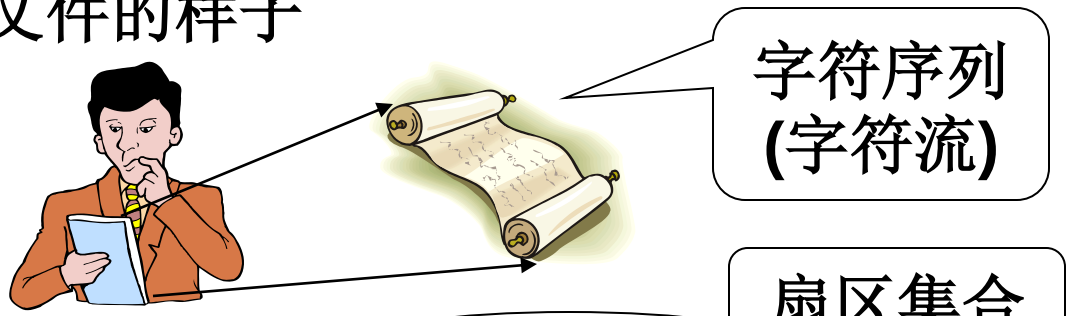
文件的存储结构、文件系统的实现方式、定期优化

12.1 文件的目录结构

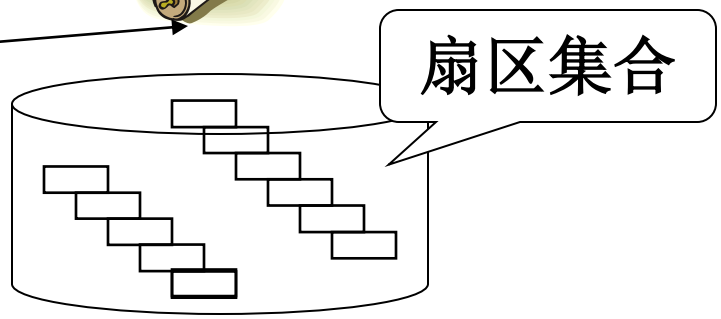
- 文件的引入回顾
- 文件盘块的三种分配方式回顾
- 文件的目录结构

文件的引入

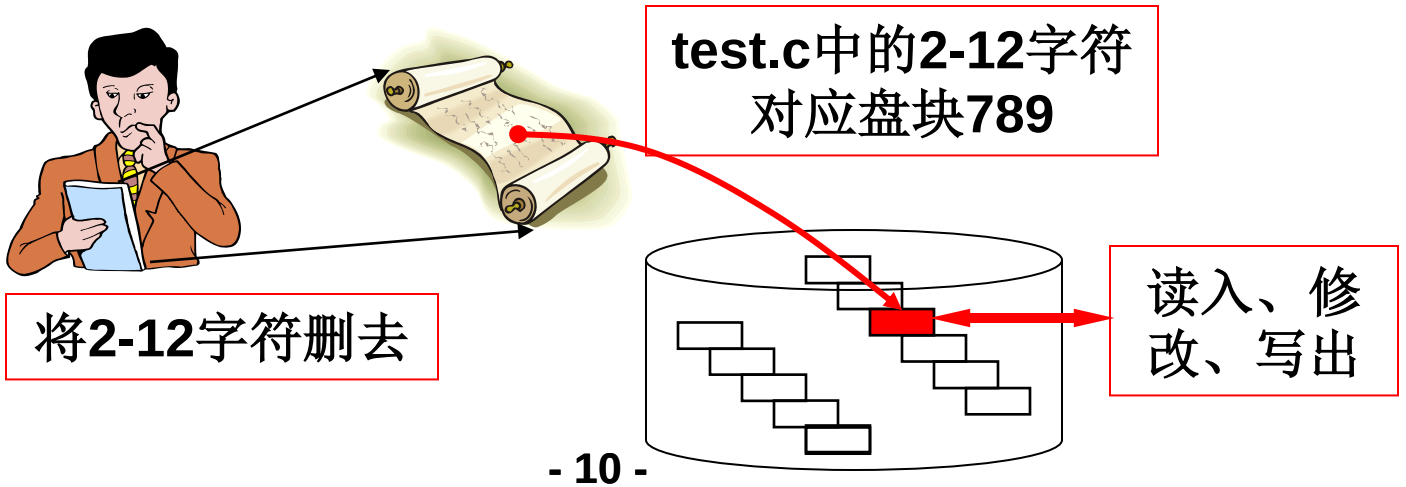
- 用户眼里文件的样子



- 磁盘上的文件的样子

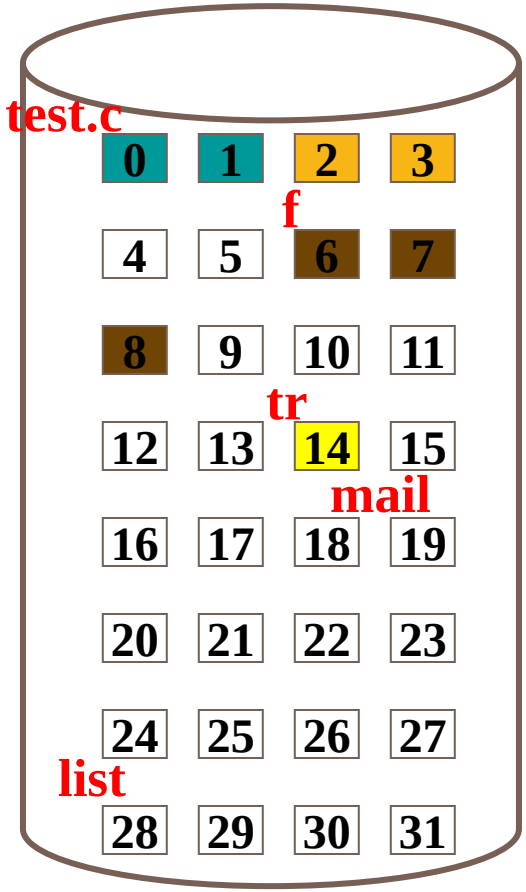


- 文件: 建立了字符流到盘块集合的映射关系

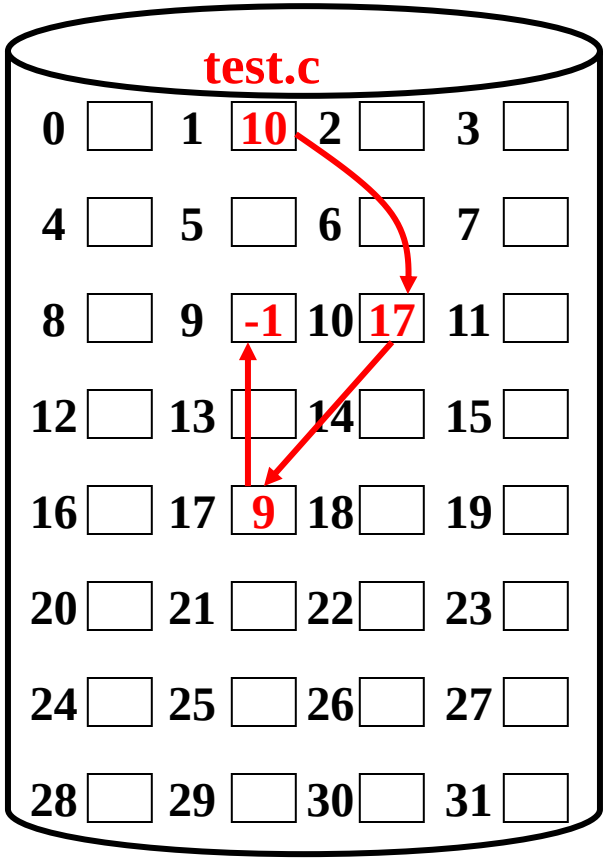


三种基本映射关系

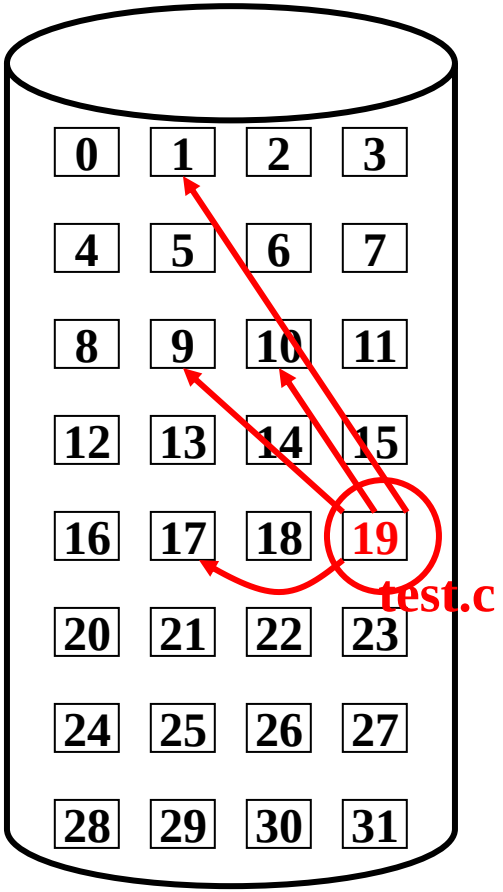
盘块连续分配



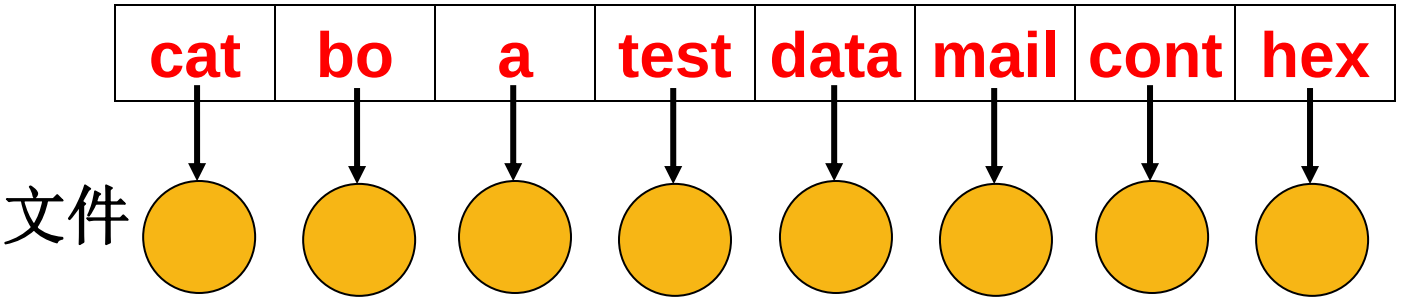
盘块链式分配



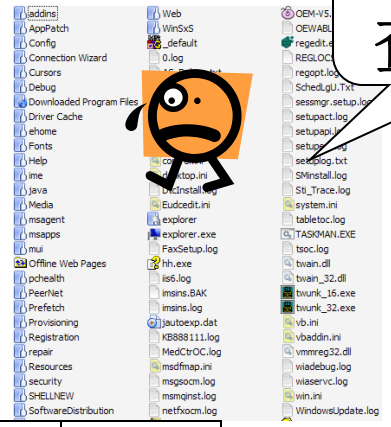
盘块索引分配



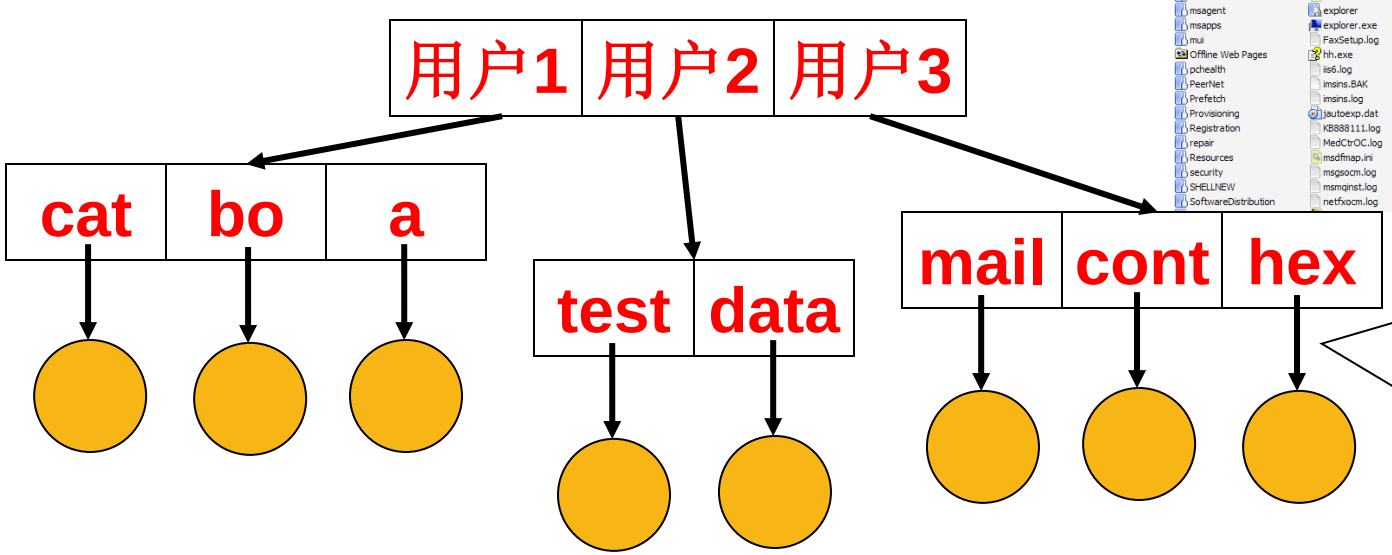
文件系统中有很多文件 怎么管理?



- 所有文件放在一层(一个大集合)
- 怎么办? 集合划分



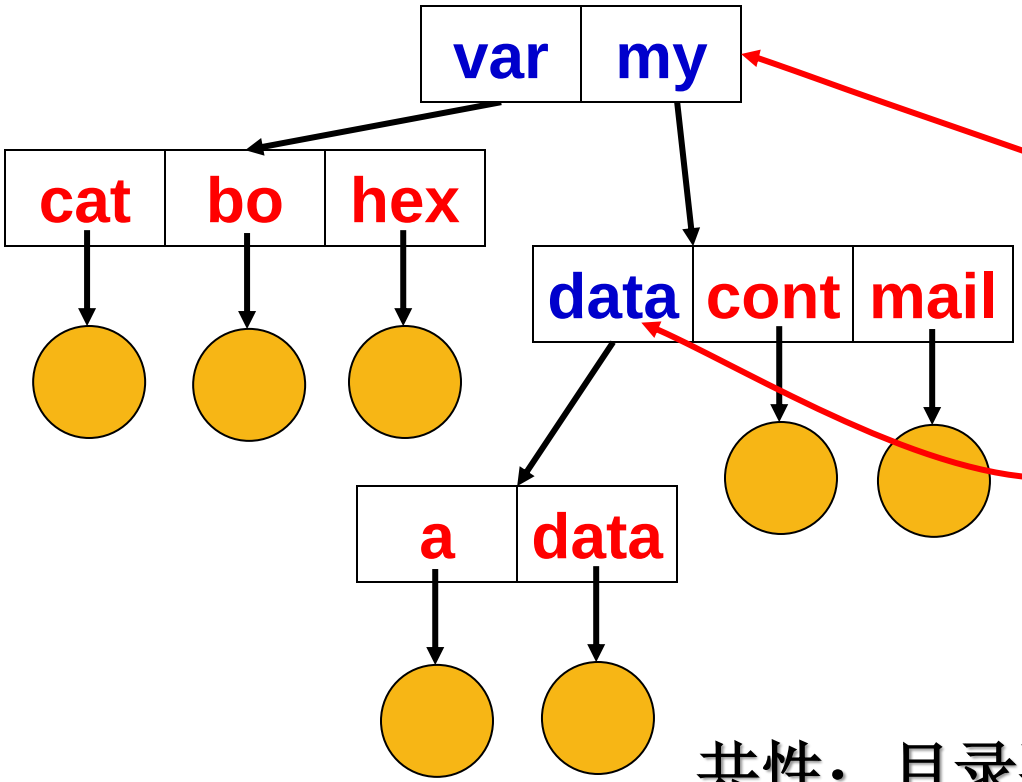
枯燥乏味
查找困难



问题依然存在: N/U，扩展性仍然差

划分的基础上继续划分　树状目录

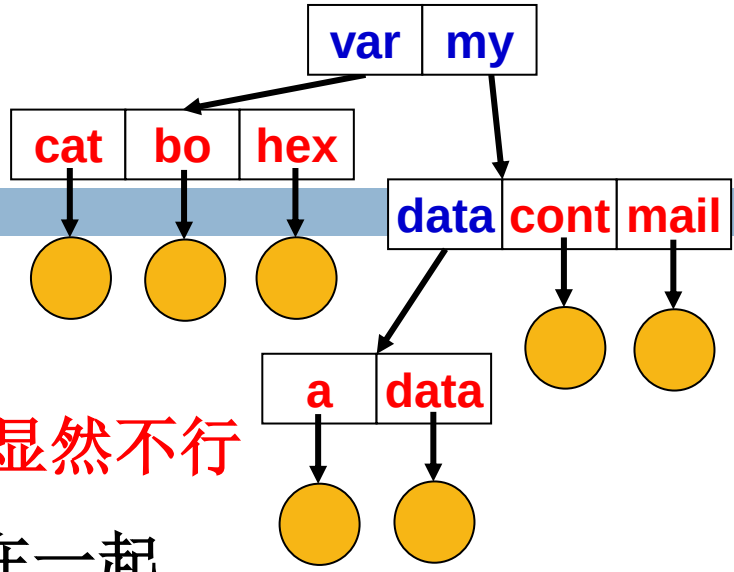
- 将划分后的集合再进行划分: k次划分后，每个集合中的文件数可以达到为O($\sqrt[k]{N}$)，比如8个文件，一次分割(4,4)，再分2次：(2,2),(2,2)，$\sqrt[3]{8} = 2$



- 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?

共性：目录要存储信息，文件要存储信息

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

- 思考: 有了树状目录后会出现什么问题?
  - 出现了路径名: /my/data/a用来定位文件a
  - 路径名 ⇒ 路径的解析: 再根据文件头定位文件内容

输入/my/data/a，获得文件a的文件头

路径的解析(Name Resolution)

想一想?

- 输入/my/data/a，获得文件a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!

文件系统中全是文件!

- 解析/my/data/a：

放在磁盘确定位置，可在OS初始化时读入!

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

从路径解析来看目录内容

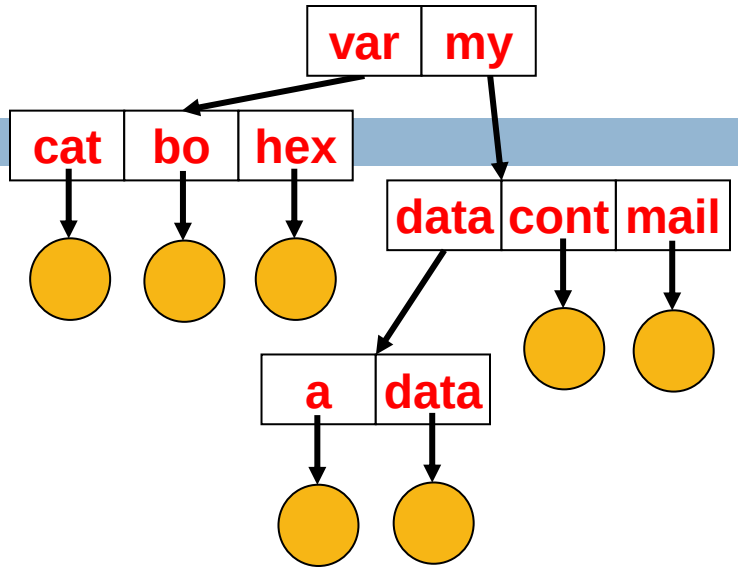
- 显然路径解析的使用频率高，因此效率很重要
  - 如何提高路径解析的效率?
  - 要使语句var=Read(fh);fh=Find(var,"??"); 效率高，var应该尽可能短!
  - 所以目录文件中不应该存放完整的文件头，可以存放指向文件头的指针
  - 文件头指针? 可将文件头连续存放(形成了数组)在磁盘的固定位置，文件头指针就是其数组项标号!

文件头尺寸也并不小

任何文件的文件头结构相同

基址已知、偏移已知就能找到文件头

树状目录的完整实现


var
my
cat
bo
hex
data
cont
mail
a
data

磁盘
文件头数组
数据盘块集合
“/”的文件头
“/”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”的数据块
<data, 103>
<cont, 225>
<mail, 77>

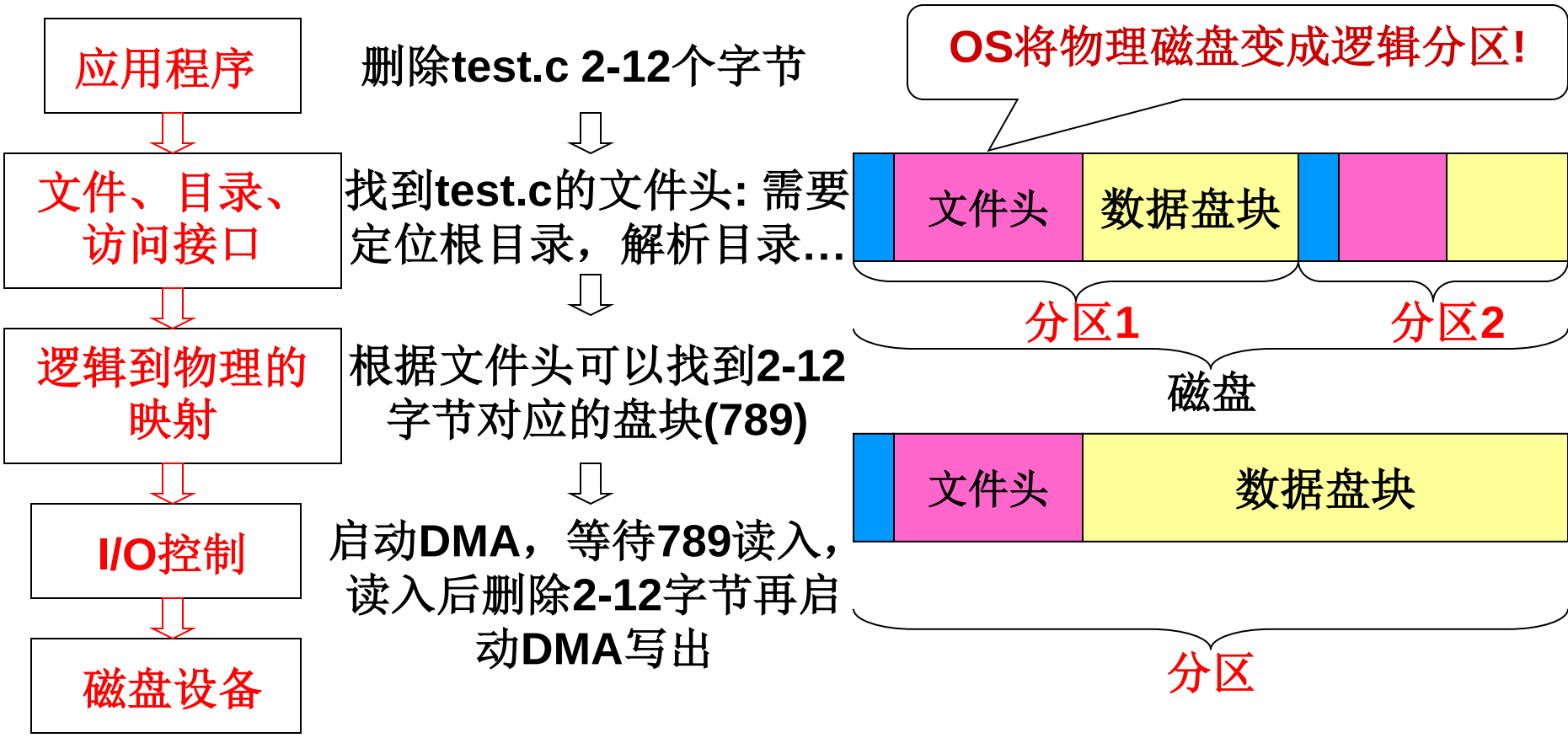
12.2 文件系统的实现

- 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

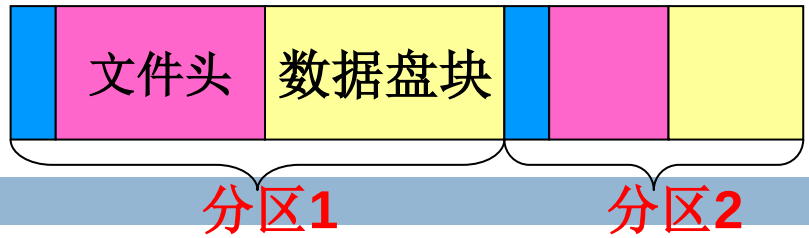
描述文件系统的实质(定义)

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“抽象盘块”的一层软件!



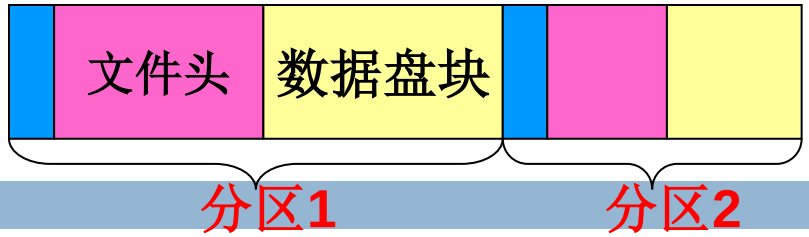
分区的详细结构



- 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：分区块数；块大小；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，文件内容

分区空闲盘块的管理

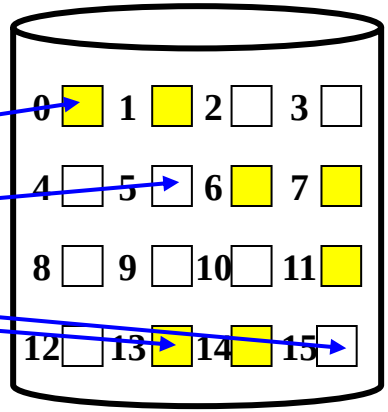


- 有的盘块被文件使用，其它盘块如何管理?
  - 组织起来等待文件的使用! 怎么组织?
  - 方法1：空闲位图(位向量)…

    0011110011101001

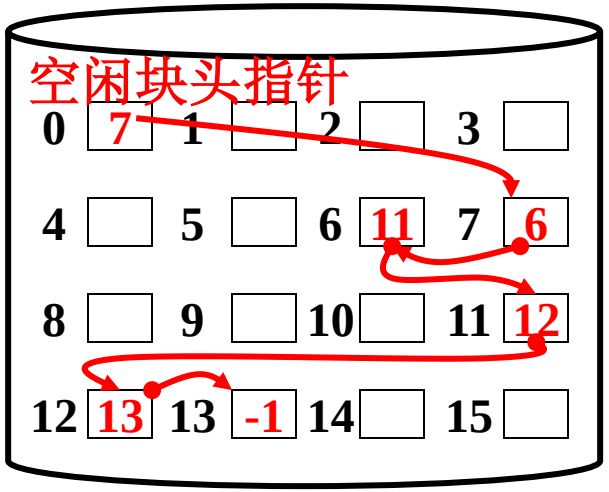
    表示磁盘块2,3,4,5,8,9,10,12空闲

    可快速分配连续盘块组，但位向量很大(1G/1k=?)



  - 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!

高效、安全、可靠

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
```

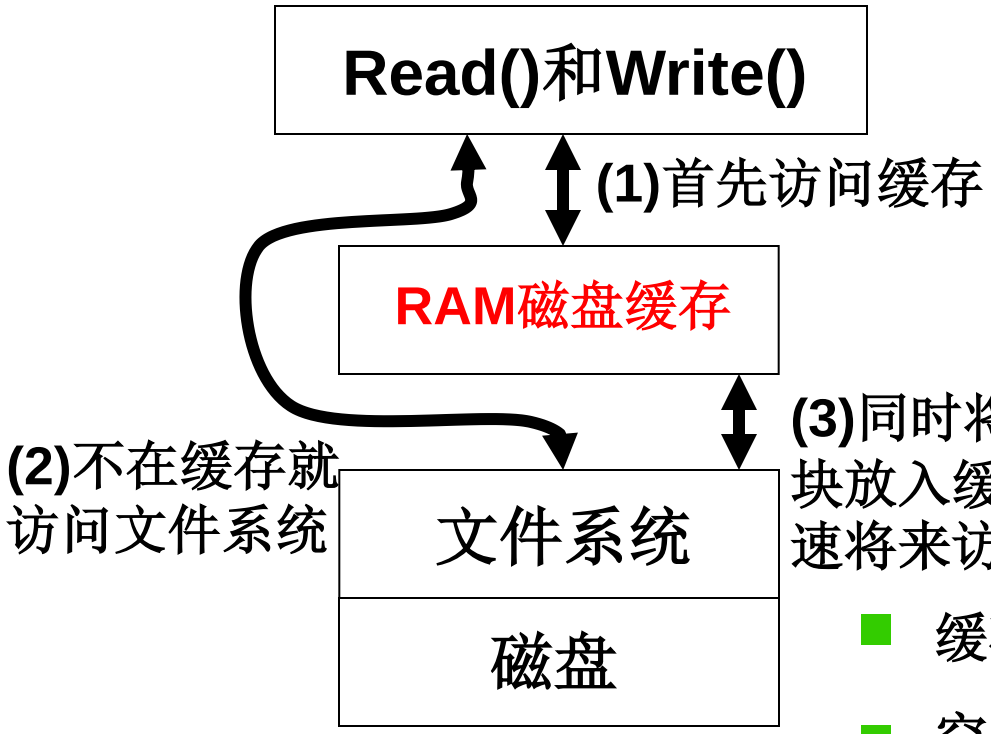
$0.859/10^3$

$5.7\times10^5 : 1$

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



非常熟悉的过程?
TLB、虚拟内存!
具有局部性性质才有意义!

Linux中的sync命令

用于强制将修改的文件内容写入磁盘

- 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?

  延迟写，直接写（缓存和磁盘都写）

  Delayed Write or Write Through

等等…还有许多细节

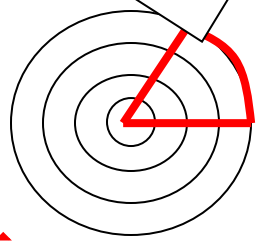
其他的提高文件访问效率的技术…

- 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)，如gcc 1.c。其FCB可驻内存?
- ls -l的使用非常频繁
  - 怎么才能快速执行?

    同一目录中的文件inode在一个柱面(组)!
  - 需要优化设计inode、目录文件分配算法…
- 显然，还有许多提高文件系统效率的技术……
  - 现在还有继续改进的空间

一段时间大多数文件访问集中在一个目录中(局部性)

每个柱面组都有inode，空闲盘块…



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? 放在哪里?
  - 权限: 读/写/执行(r/w/x)，当然是放在文件头中!
  - 一实例: drwxrwxrwx root staff test/

不同用户具有不同权限 | owner id | group id

  - 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:

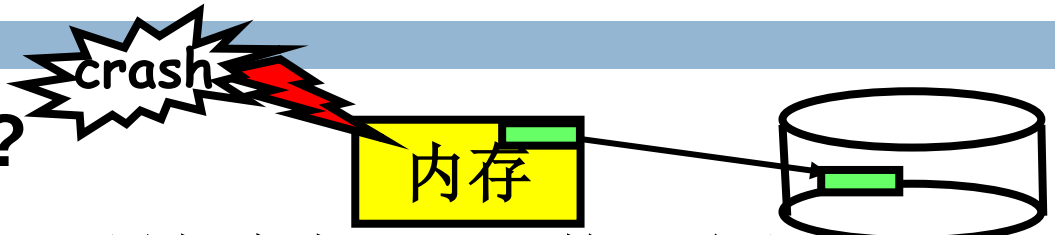
(1)PCB中有uid和gid; (2)fork时设置; (3)这些信息在登录时收集(从tty); (4)文件访问时校验

又将许多东西联系在了一起…OS魅力所在!

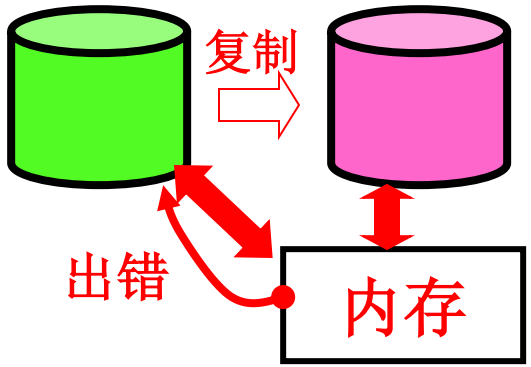
回忆:进程控制块PCB

```
struct task_struct {
/*-----------------------------------------------*/
    long state;    // 进程运行状态（-1 不可运行，0 可运行，>0 以停止）
    long counter; // 任务运行时间片，递减到 0 是说明时间片用完
    long priority;     // 任务运行优先数，刚开始是 counter＝priority
    long signal;  // 任务的信号位图，信号值=偏移+1
    struct sigaction sigaction[32];  //信号执行属性结构，对应信号将要执行的操作和标志信息
    long blocked; // 信号屏蔽码
/*--------------- various fields---------------------- */
    int exit_code;     // 任务退出码，当任务结束时其父进程会读取
    unsigned long start_code, end_code, end_data, brk, start_stack;
        // start_code      代码段起始的线性地址
        // end_code    代码段长度
        // end_data    代码段长度+数据段长度
        // brk         代码段长度+数据段长度+bss 段长度
        // start_stack     堆栈段起始线性地址
    long pid, father, pgrp, session, leader;
        // pid 进程号，father 父进程号，pgrp 父进程组号，session 会话号，leader 会话首领
    unsigned short uid, euid, suid;
        // uid 用户标 id，euid 有效用户 id，suid 保存的用户 id
    unsigned short gid, egid, sgid;
        // gid 组 id，egid 有效组 id，sgid 保存组 id
    long alarm;    // 报警定时值
```

良好运转的文件系统似乎也应该容错

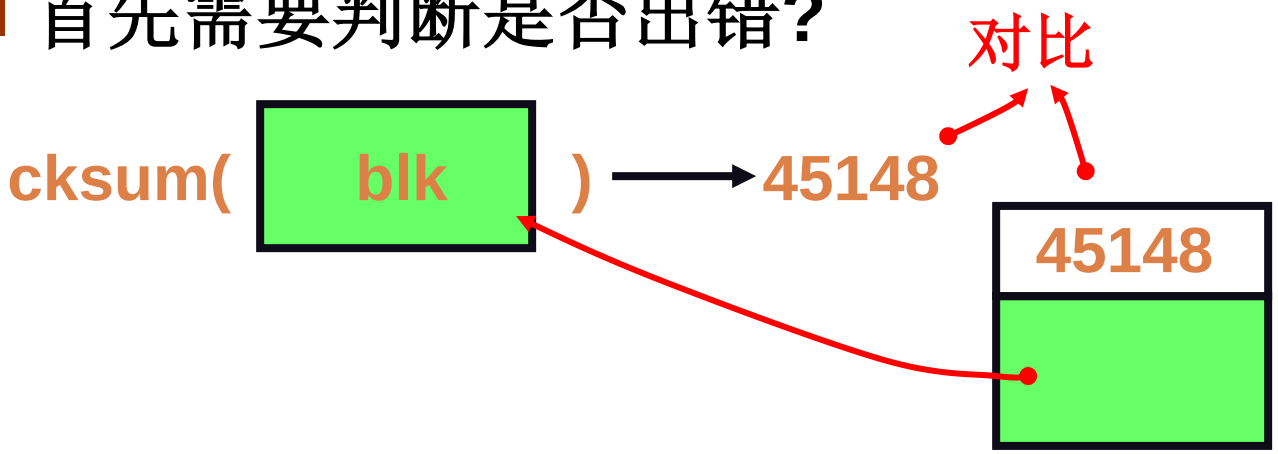


- 有过这样的经历?
  - 用户当然不希望: 早晨起来发现写了数月论文打不开了(如因下一块的link断了)…
  - 错误是难免的: 误操作、突然断电、无处不在的电磁干扰… 怎么办? 错误避免还是错误恢复…
  - **RAID**(Redundant Arrays of Independent Disks)
  - RAID基本思想就是冗余(R): 如在镜像磁盘上备份数据, 发现错误时拷贝镜像磁盘(恢复)

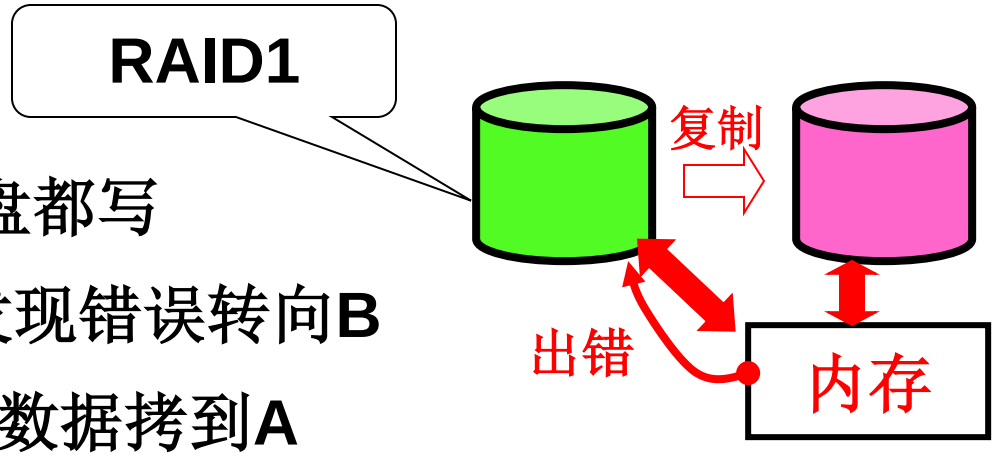


RAID的简单实现

- 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

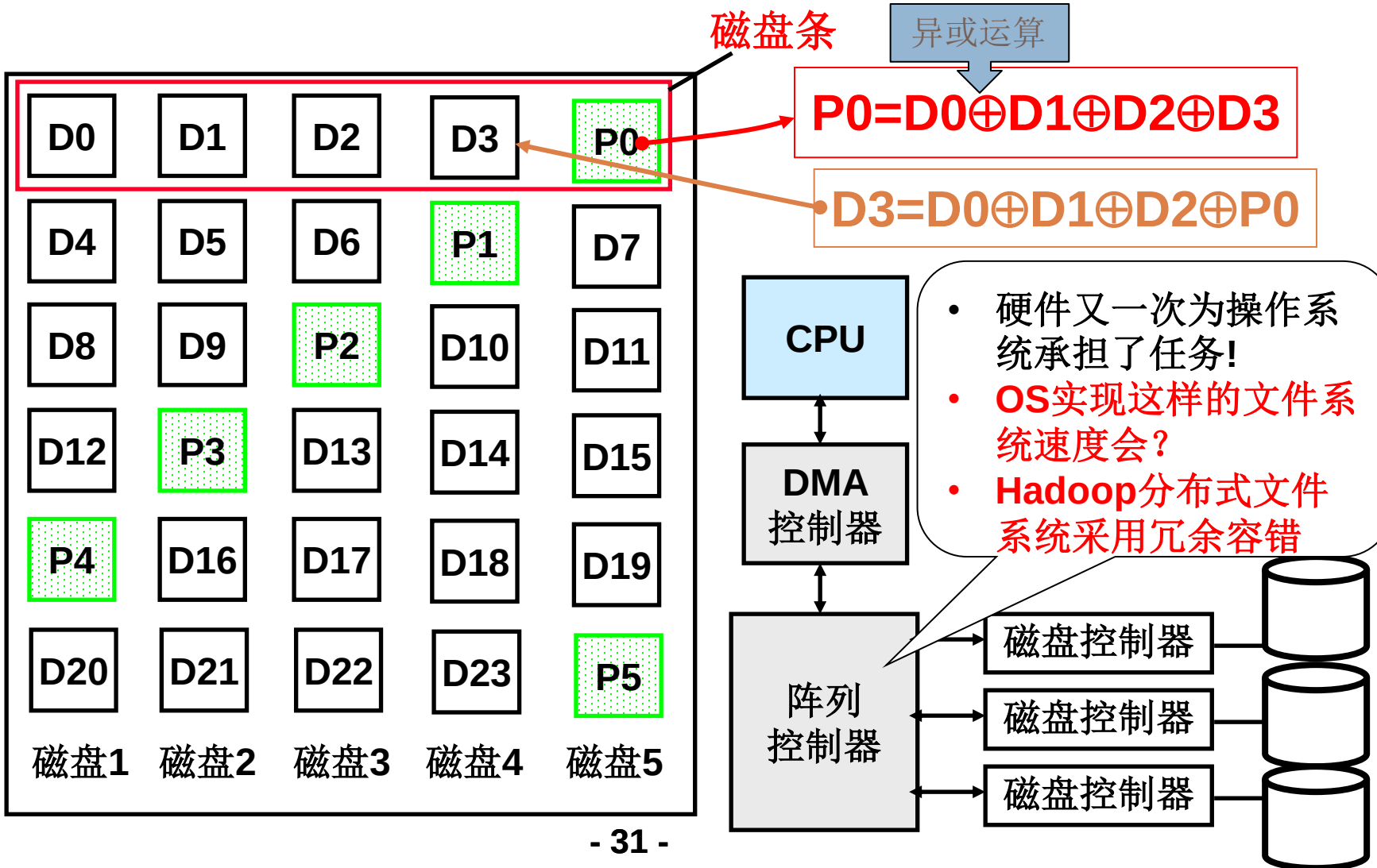
    (2)从磁盘A读，发现错误转向B

    (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



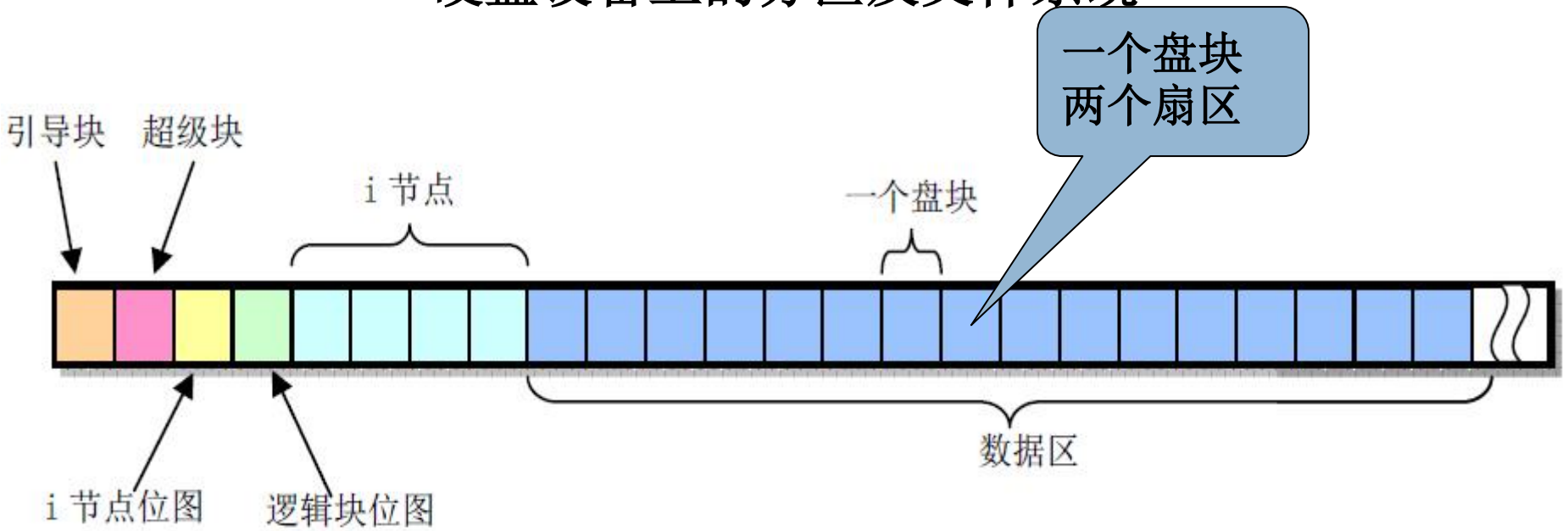
实践一个可实际运转的文件系统!

12.3 MINIX文件系统1.0实现

- MINIX，是一种基于微内核架构的类UNIX计算机操作系统，于1987年由Andrew S. Tanenbaum教授发布，它启发了Linux。
- Linus Torvalds深受Minix的启发写出了第一版本的Linux内核。
- Minix于2000年重新改为BSD授权，成为自由和开放源码软件，为全球注册商标

MINIX文件系统V1.0的实现

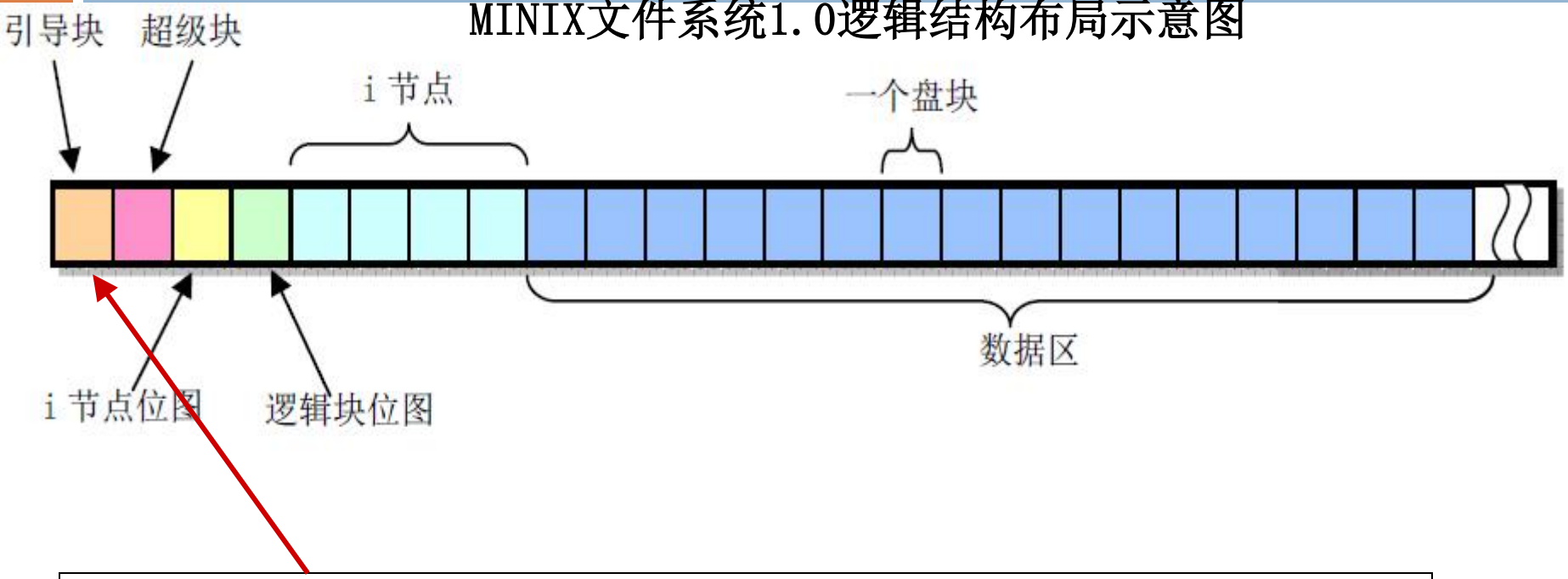
硬盘设备上的分区及文件系统



MINIX文件系统1.0逻辑结构布局示意图

MINIX文件系统V1.0的实现

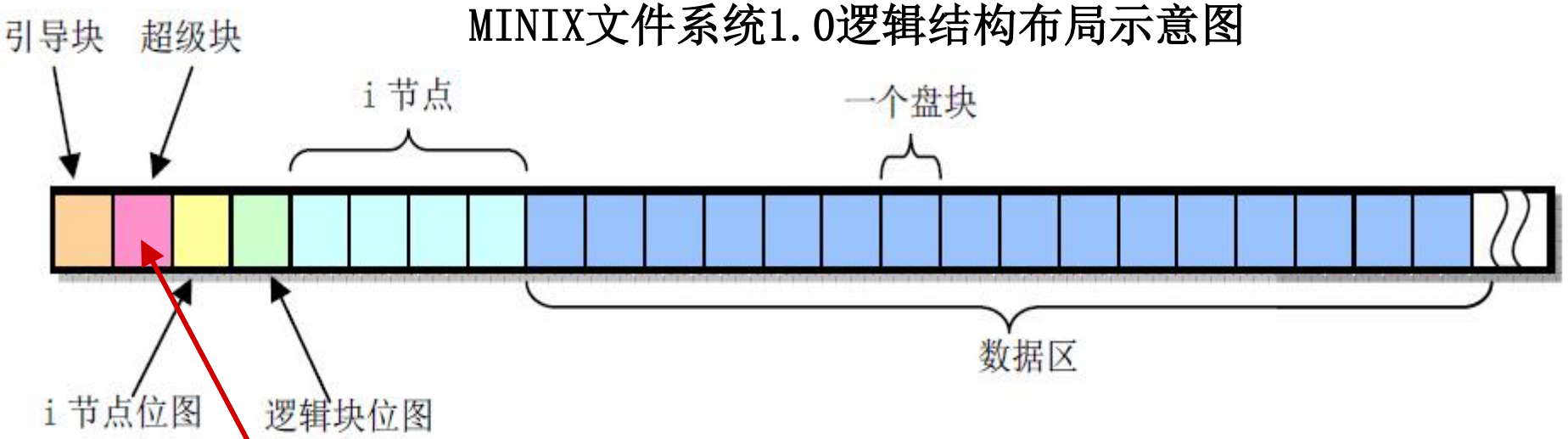
MINIX文件系统1.0逻辑结构布局示意图



- 计算机加电启动时，由ROM BOIS程序自动读入MBR，MBR找到引导块并读入引导代码和数据。
- 对于不是引导分区，该盘块空闲不用。

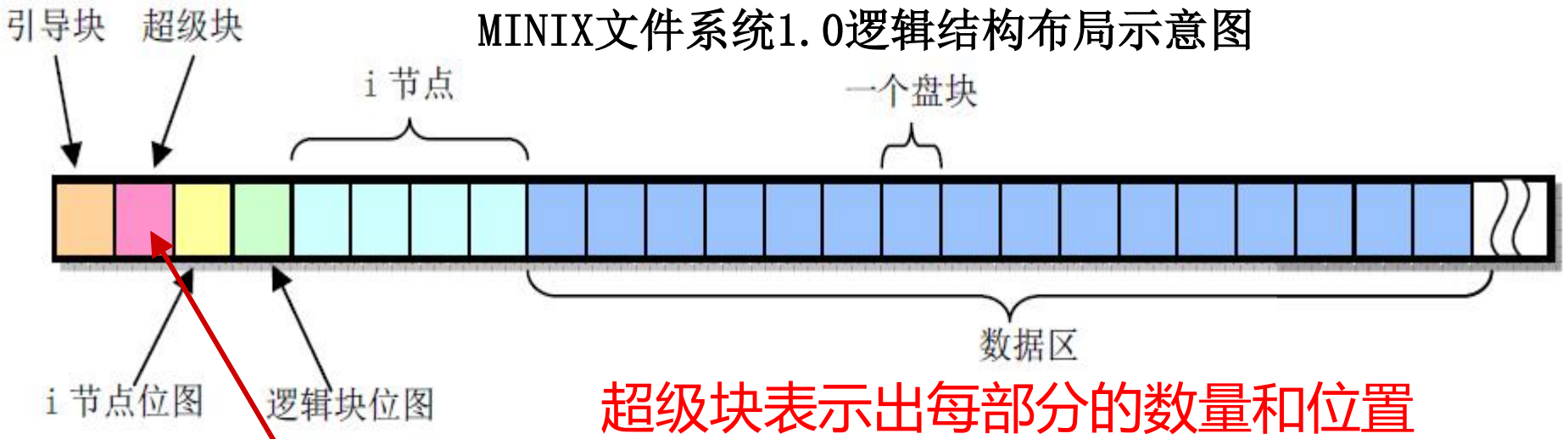
MINIX文件系统V1.0的实现

MINIX文件系统1.0逻辑结构布局示意图



- 保存分区中文件系统的结构信息

MINIX文件系统V1.0的实现



MINIX文件系统1.0逻辑结构布局示意图

超级块表示出每部分的数量和位置

字段名称	数据类型	说明
s_ninodes	short	i 节点数
s_nzones	short	逻辑块数(或称为区块数)
s_imap_blocks	short	i 节点位图所占块数
s_zmap_blocks	short	逻辑块位图所占块数
s_firstdatazone	short	数据区中第一个逻辑块块号
s_log_zone_size	short	Log_2(磁盘块数/逻辑块)
s_max_size	long	最大文件长度
s_magic	short	文件系统幻数（0x137f）

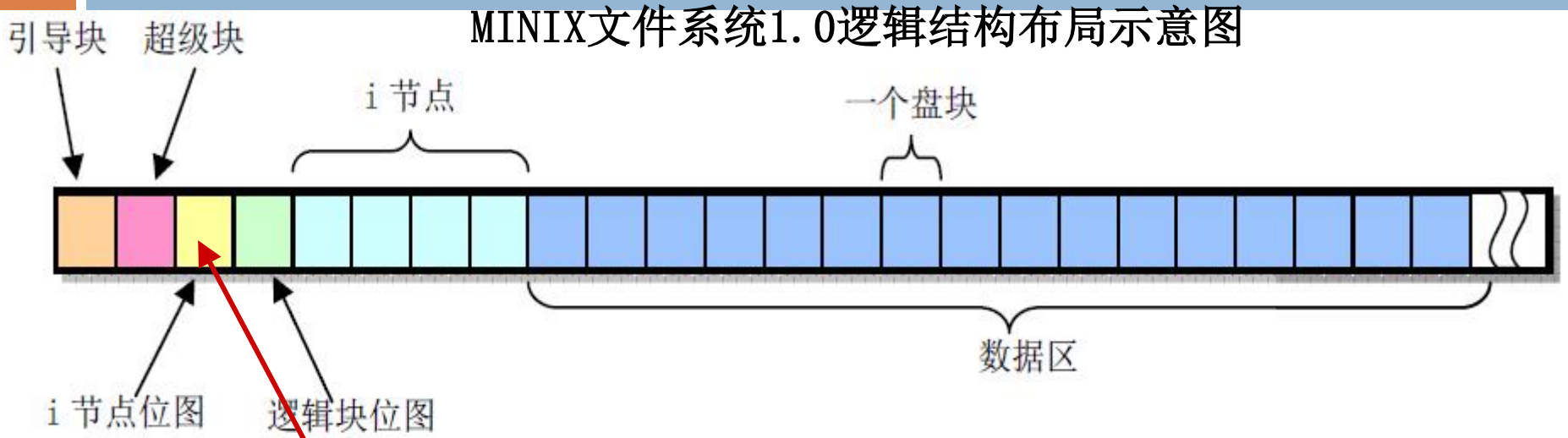
超级块数据结构

MINIX文件系统V1.0的实现

	字段名称	数据类型	说明
出现在盘上和内存中的字段	s_ninodes	short	i 节点数
	s_nzones	short	逻辑块数(或称为区块数)
	s_imap_blocks	short	i 节点位图所占块数
	s_zmap_blocks	short	逻辑块位图所占块数
	s_firstdatazone	short	数据区中第一个逻辑块块号
	s_log_zone_size	short	Log_2(磁盘块数/逻辑块)
	s_max_size	long	最大文件长度
	s_magic	short	文件系统幻数（0x137f）
仅在内存中使用的字段	s_imap[8]	buffer_head *	i 节点位图在高速缓冲块指针数组
	s_zmap[8]	buffer_head *	逻辑块位图在高速缓冲块指针数组
	s_dev	short	超级块所在设备号
	s_isup	m_inode *	被安装文件系统根目录 i 节点
	s_imount	m_inode *	该文件系统被安装到的 i 节点
	s_time	long	修改时间
	s_wait	task_struct *	等待本超级块的进程指针
	s_lock	char	锁定标志
	s_rd_only	char	只读标志
	s_dirt	char	已被修改(脏)标志

MINIX文件系统1.0超级块数据结构

MINIX文件系统V1.0的实现

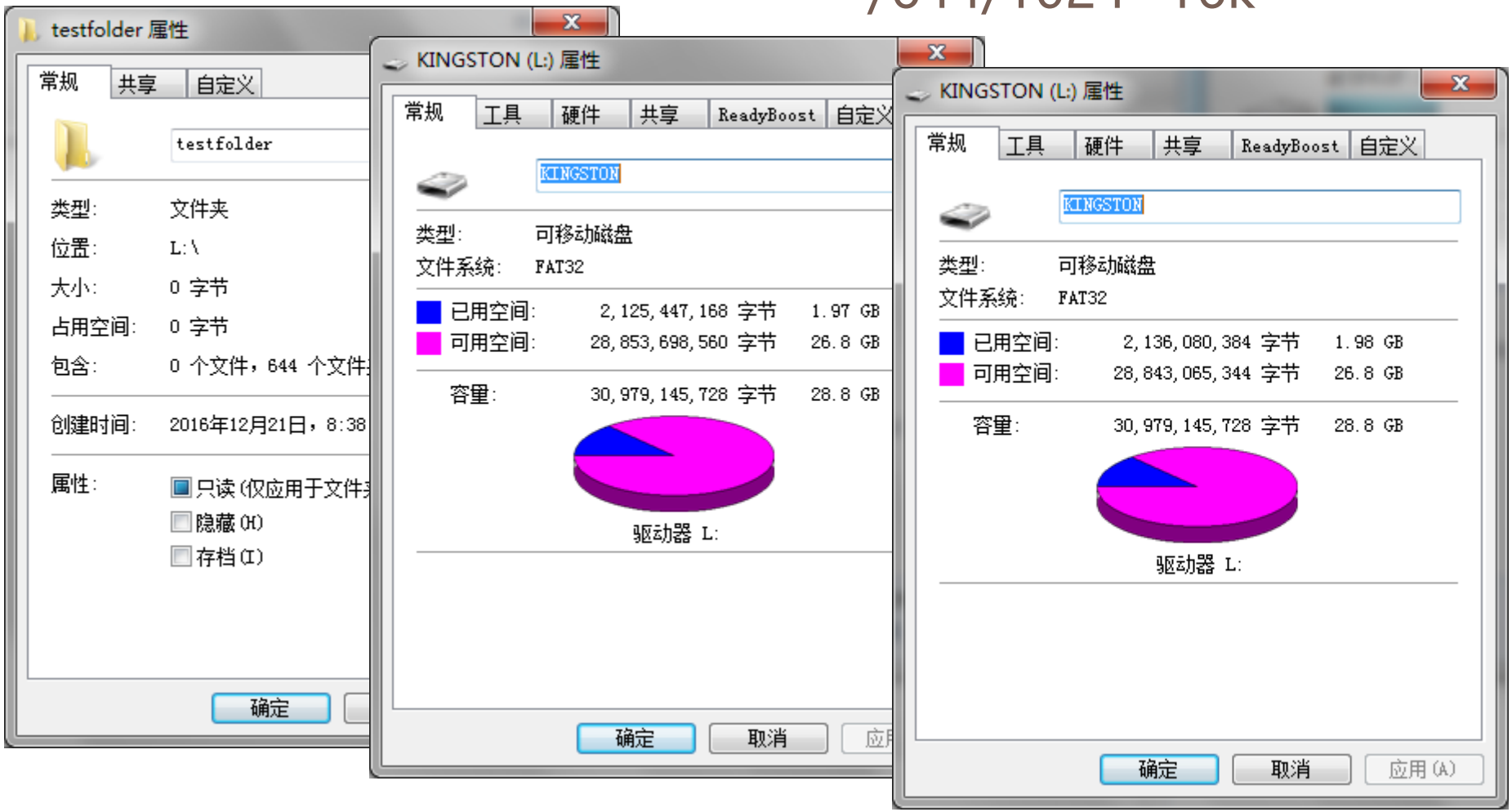


- 标记i节点的使用情况，每bit位代表1个i节点，即1个目录文件或普通文件
- 可以占用多个盘块

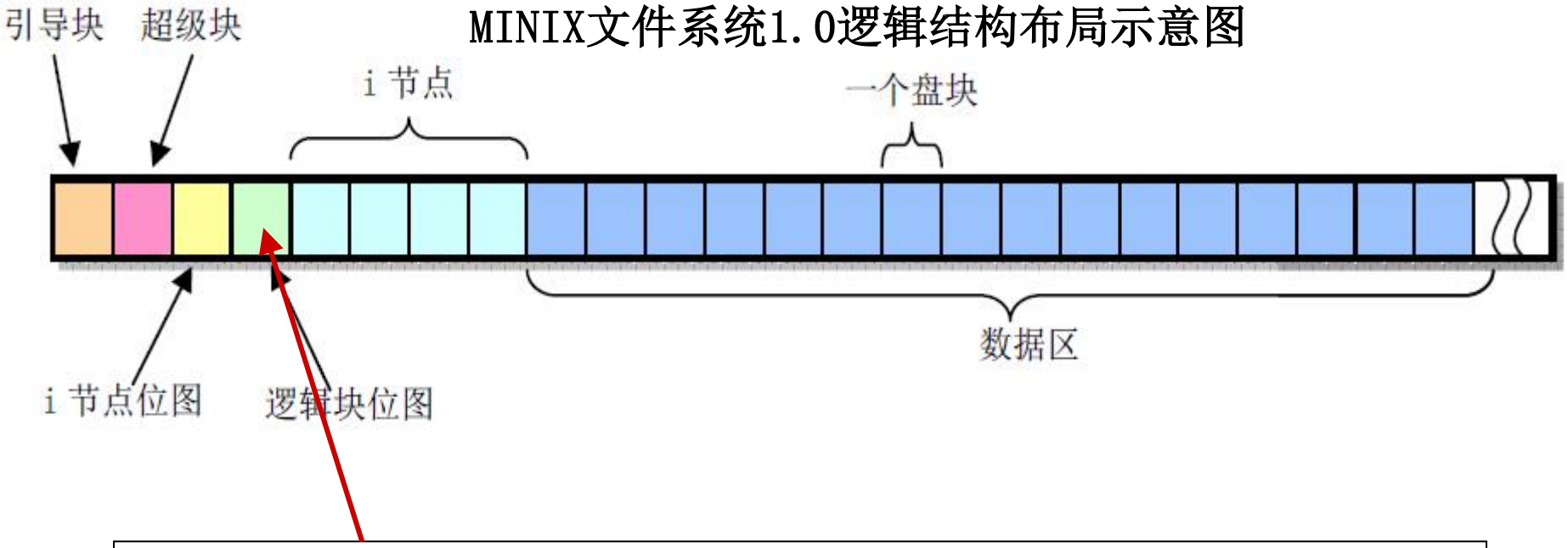
●如果建一个空目录，盘块分配情况如何?

FAT32优盘文件夹测试

- FAT32优盘建立644个文件夹 (36080384-25447168)/644/1024=16k



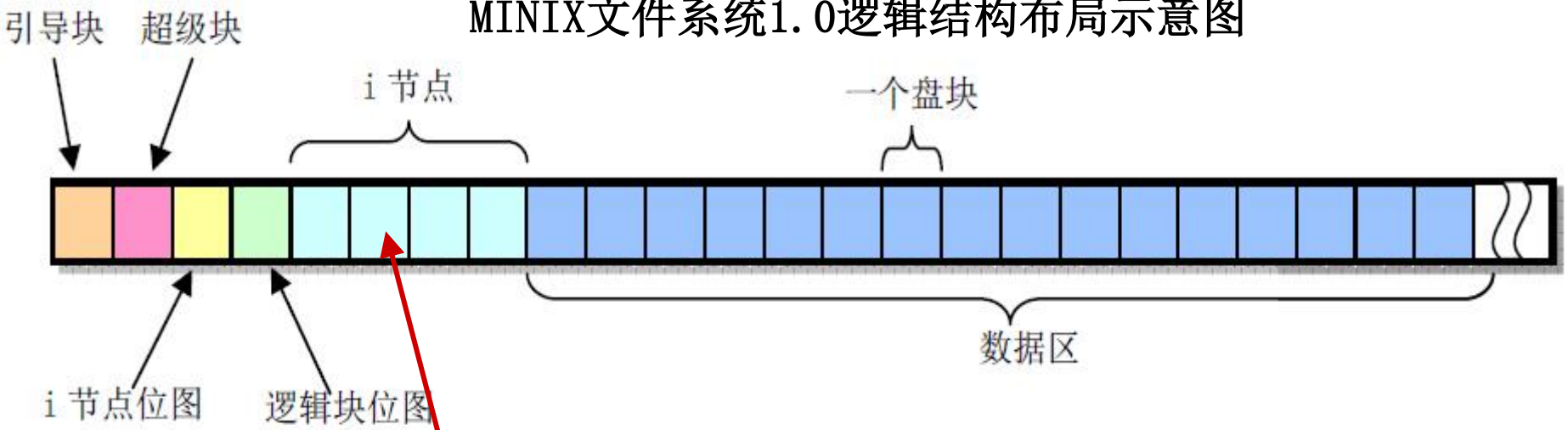
MINIX文件系统V1.0的实现



- 标记磁盘分区中每个数据盘块的使用情况，每bit位代表1个盘块
- 只标记数据区的盘块

MINIX文件系统V1.0的实现

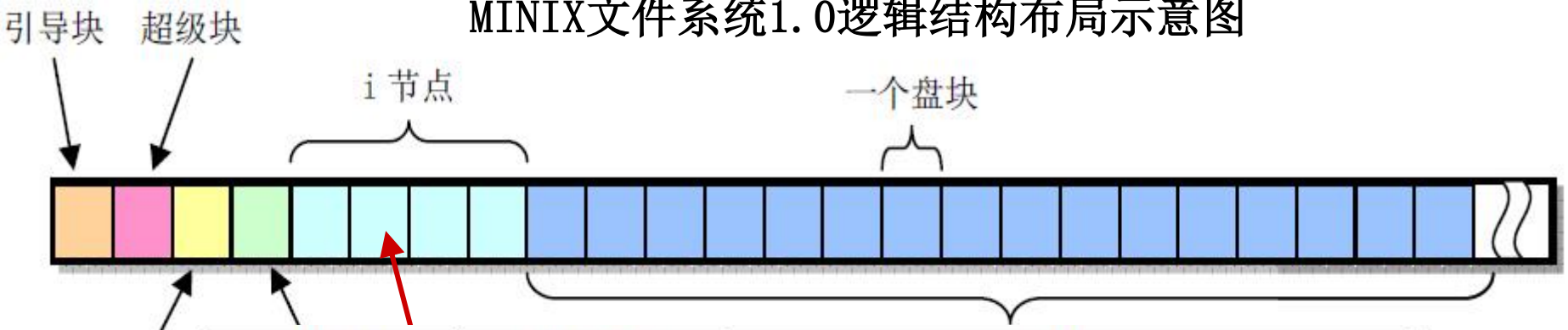
MINIX文件系统1.0逻辑结构布局示意图



- 存放着文件系统中目录文件或普通文件的i节点
- 每个目录文件或普通文件都有1个i节点结构
- 每个i节点结构中存放着对应文件的相关信息
- 第1个i节点为根目录文件的i节点结构

MINIX文件系统V1.0的实现

MINIX文件系统1.0逻辑结构布局示意图



字段名称	数据类型	说明
i_mode	short	文件的类型和属性（rwx 位）
i_uid	short	文件宿主的用户 id
i_size	long	文件长度（字节）
i_mtime	long	修改时间（从 1970.1.1:0 时算起，秒）
i_gid	char	文件宿主的组 id
i_nlinks	char	链接数（有多少个文件目录项指向该 i 节点）
i_zone[9]	short	文件所占用的盘上逻辑块号数组。其中： zone[0]-zone[6]是直接块号； zone[7]是一次间接块号； zone[8]是二次（双重）间接块号。 注：zone 是区的意思，可译成区块或逻辑块。对于设备特殊文件名的 i 节点，其 zone[0]中存放的是该文件名所指设备的设备号。

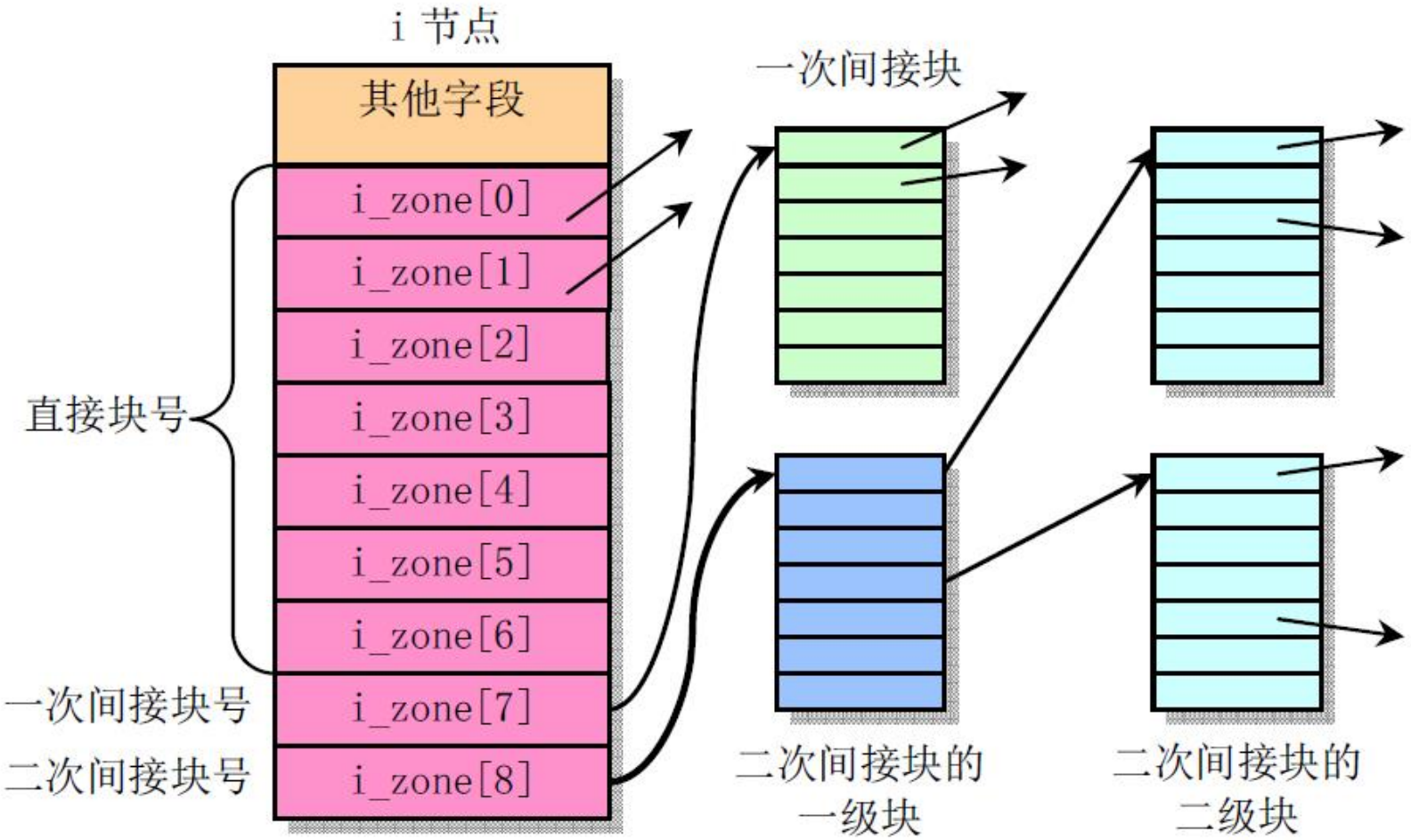
i节点数据结构（32bytes）

MINIX文件系统V1.0的实现

	字段名称	数据类型	说明
在盘上和内存中的字段，共32字节	i_mode	short	文件的类型和属性（rwx位）
	i_uid	short	文件宿主的用户id
	i_size	long	文件长度（字节）
	i_mtime	long	修改时间（从1970.1.1:0时算起，秒）
	i_gid	char	文件宿主的组id
	i_nlinks	char	链接数（有多少个文件目录项指向该i节点）
	i_zone[9]	short	文件所占用的盘上逻辑块号数组。其中： zone[0]-zone[6]是直接块号； zone[7]是一次间接块号； zone[8]是二次（双重）间接块号。 注：zone是区的意思，可译成区块或逻辑块。对于设备特殊文件名的i节点，其zone[0]中存放的是该文件名所指设备的设备号。
仅在内存中使用的字段	i_wait	task_struct *	等待该i节点的进程。
	i_atime	long	最后访问时间。
	i_ctime	long	i节点自身被修改时间。
	i_dev	short	i节点所在的设备号。
	i_num	short	i节点号。
	i_count	short	i节点被引用的次数，0表示空闲。
	i_lock	char	i节点被锁定标志。
	i_dirt	char	i节点已被修改（脏）标志。
	i_pipe	char	i节点用作管道标志。
	i_mount	char	i节点安装了其他文件系统标志。
	i_seek	char	搜索标志（lseek操作时）。
	i_update	char	i节点已更新标志。

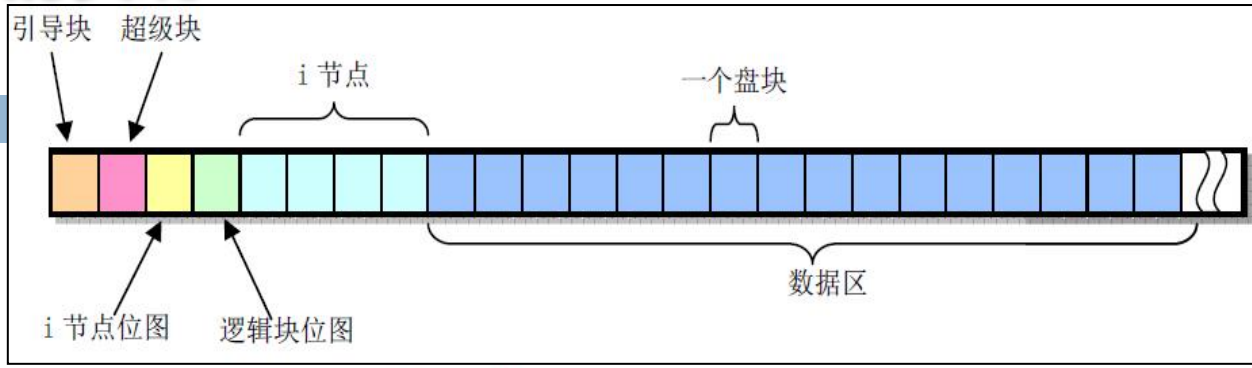
MINIX文件系统1.0的i节点数据结构

MINIX文件系统V1.0的实现

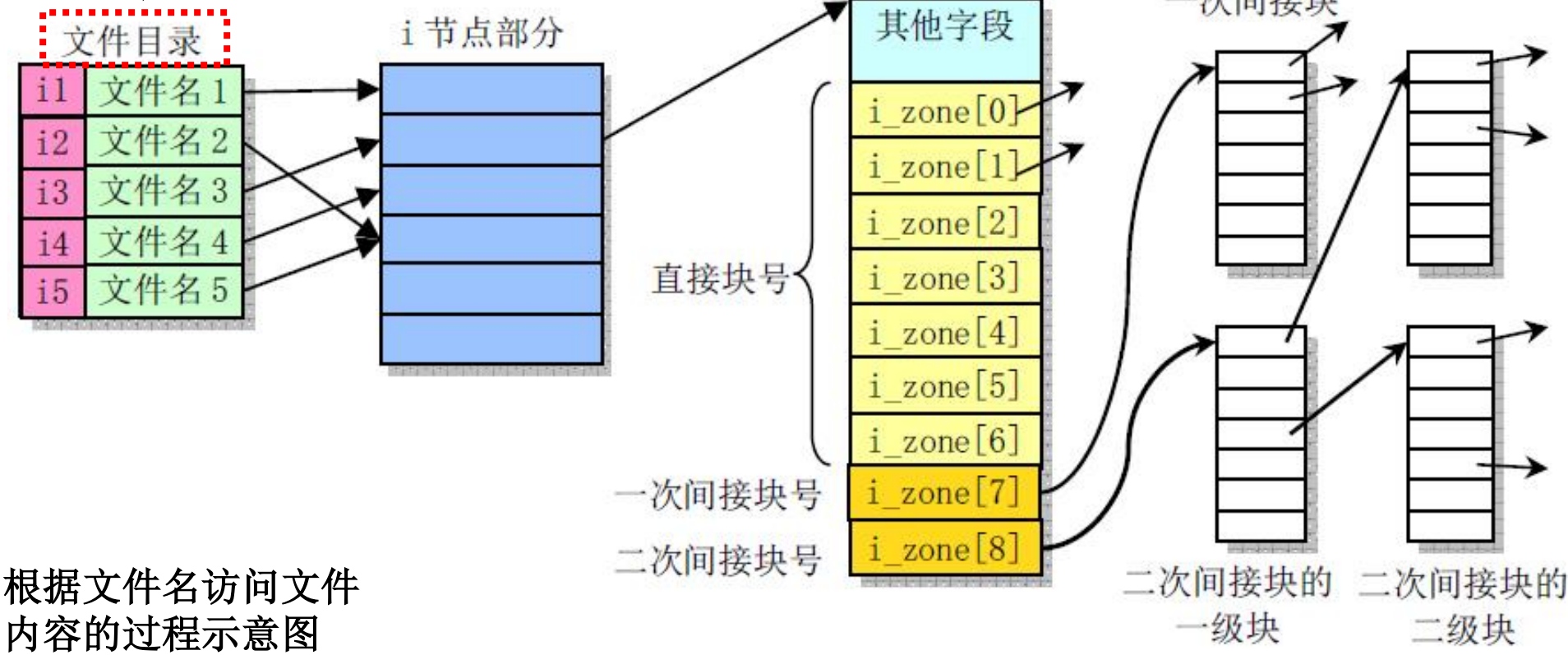


i节点数据结构中逻辑块数组i_zone的功能

MINIX文件系统V1.0的实现



/usr/bin/文件名3



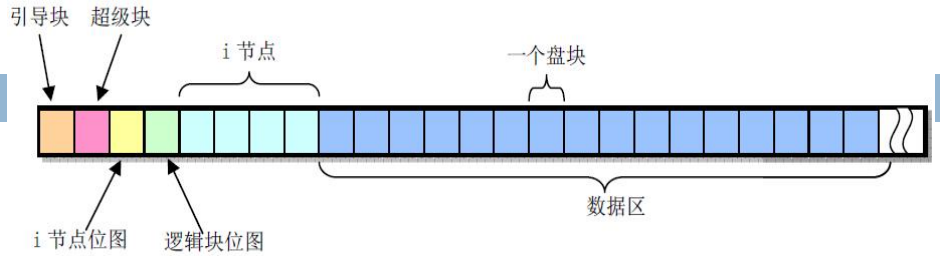
根据文件名访问文件内容的过程示意图

MINIX文件系统V1.0的实现

如何访问文件/usr/bin/文件名3（编辑工具）？

1. 根目录文件的i节点位置是固定的，即第1号i节点
2. 1号i节点的数据块内容为根目录下的目录项列表
3. 通过该目录项列表匹配目录名“usr”
4. 若找到，则可得到文件“/usr”的i节点号i1
5. 根据i1号i节点的数据块，可以取得目录文件“/usr”的内容，即子目录usr的文件目录项列表
6. 通过该目录项列表匹配目录名“bin”
7. 若找到，则可得到文件“/usr/bin”的i节点号i2
8. 根据i2号i节点的数据块，可以取得目录文件“/usr/bin”的内容，即子目录bin的文件目录项列表
9. 通过该目录项列表匹配文件名“文件名3”
10. 若找到，则可得到文件“/usr/bin/文件名3”的i节点号i3
11. 根据i3号i节点的数据块，可以取得文件“文件名3”的内容

MINIX文件系统V1.0的实现



如何删除文件/usr/bin/vi？

1. 根目录文件的i节点位置是固定的，即第1号i节点
2. 1号i节点的数据块内容为根目录下的目录项列表
3. 通过该目录项列表匹配目录名“usr”
4. 若找到，则可得到文件“/usr”的i节点号i1
5. 根据i1号i节点的数据块，可以取得目录文件“/usr”的内容，即子目录usr的文件目录项列表
6. 通过该目录项列表匹配目录名“bin”
7. 若找到，则可得到文件“/usr/bin”的i节点号i2
8. 根据i2号i节点的数据块，可以取得目录文件“/usr/bin”的内容，即子目录bin的文件目录项列表
9. 通过该目录项列表匹配文件名“vi”
10. 若找到，则可得到文件“/usr/bin/vi”的i节点号i3

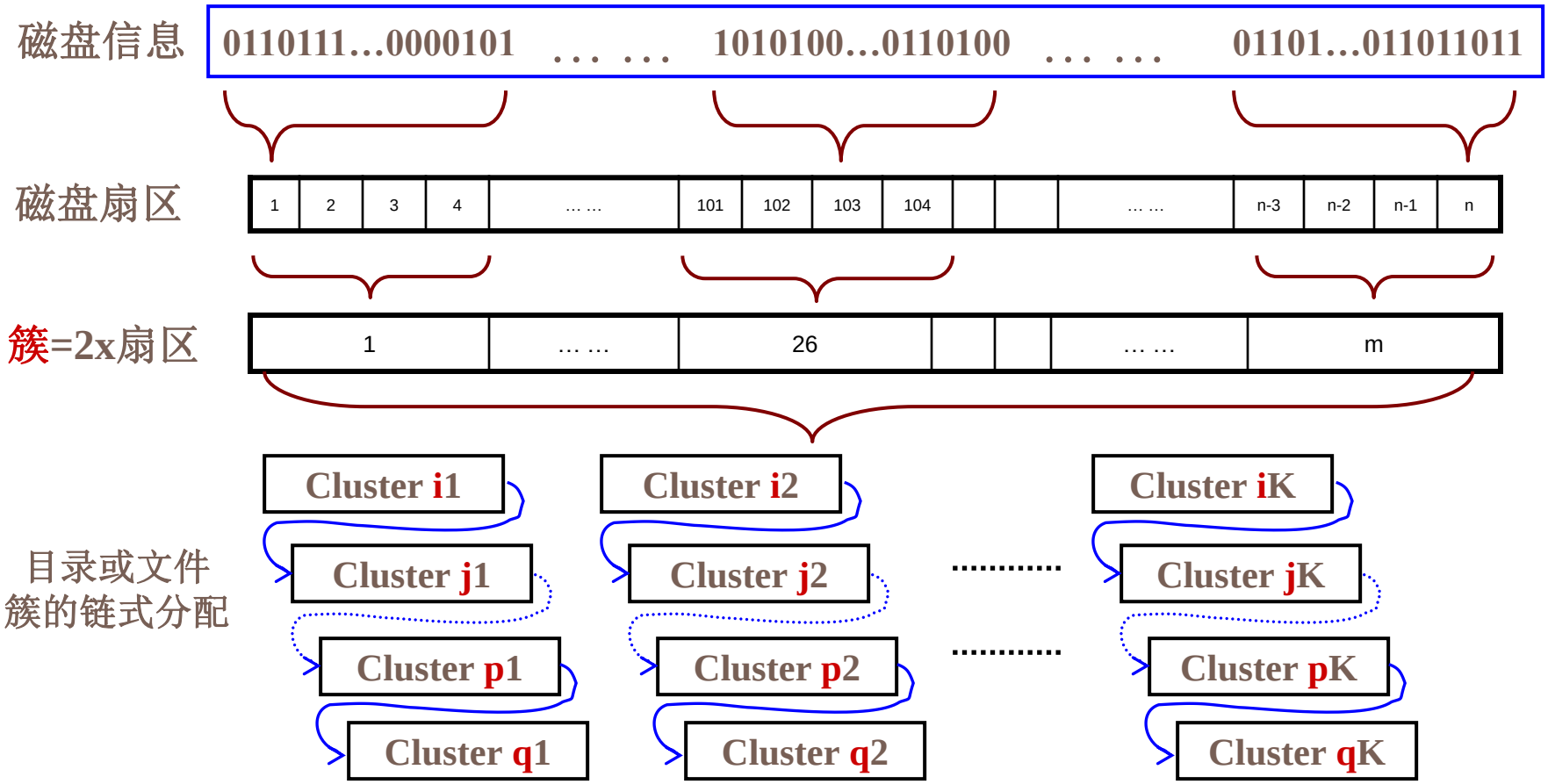
找到i节点，根据文件长度从i节点中找到每个盘块的索引号，将i节点位图和逻辑块位图中相应位置0（释放）。

12.4 Windows的FAT文件系统实现

- Windows的FAT文件系统实现

Windows的FAT文件系统实现

数据组织逻辑结结构



Windows的FAT文件系统实现

FAT卷结构示意图

引导代码

介质标志
磁头数
每磁道扇区数
扇区大小
扇区数/簇

总扇区数
保留扇区数
隐藏扇区数

FAT数目
每个FAT占扇区数
根目录项数

如果是引导分区,则存在该代码,否则空闲

软盘或硬盘等

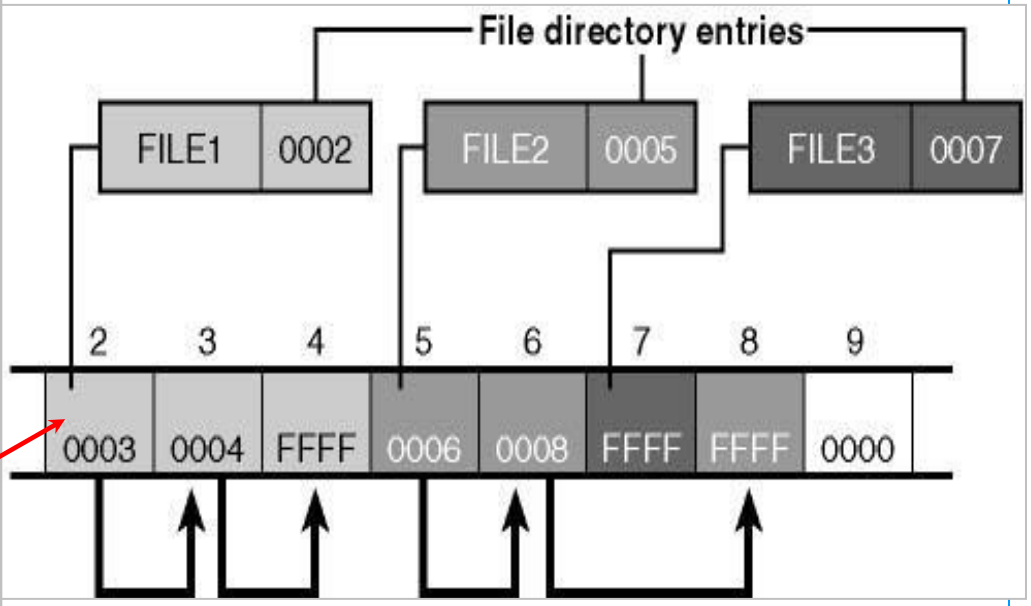
该参数不是固定不变的, 一般卷越大簇越大

Windows的FAT文件系统实现

FAT卷结构示意图

功能： 记录和描述整个卷使用情况

1. FAT文件系统格式信息：
   FAT12/FAT16/FAT32
2. 卷上每一簇对应FAT文件分配表中一项，记录该簇使用情况
   包含：簇地址号和使用标志信息
   使用标志信息＝**0** — 该簇空闲未用
   ≠0 — 该簇被占用
3. 每个目录/文件的文件分配链(簇链)
   链尾标志信息：
   0xFFF/0xFFFF/0xFFFFFFFF



Windows的FAT文件系统实现

FAT卷结构示意图

FAT2是FAT1的“镜像”备份

文件分配表对卷非常重要
它的内容破坏会导致部分文件无法访问，甚至导致整卷瘫痪

对FAT表备份是十分必要的
若FSD（文件系统驱动程序）不能正常访问FAT1，则会访问FAT2

Windows的FAT文件系统实现

FAT卷结构示意图

根目录区保存卷中根目录项内容

FAT12和FAT16卷中预留256个目录项空间(是1个文件)
即指定了根目录可以容纳的文件和目录数上限
FAT32卷中没有预留根目录空间（2号)，文件/目录数更大
FAT目录项大小为32字节(文件名遵循8.3命名规则)，保存：
文件名、文件尺寸、文件属性、起始簇号、
创建日期和时间、最后访问日期、最后修改日期和时间

如果目录/文件名为长文件名（非8.3规则），则通过增加若干个目录项的方法解决

Windows的FAT文件系统实现

FAT卷结构示意图

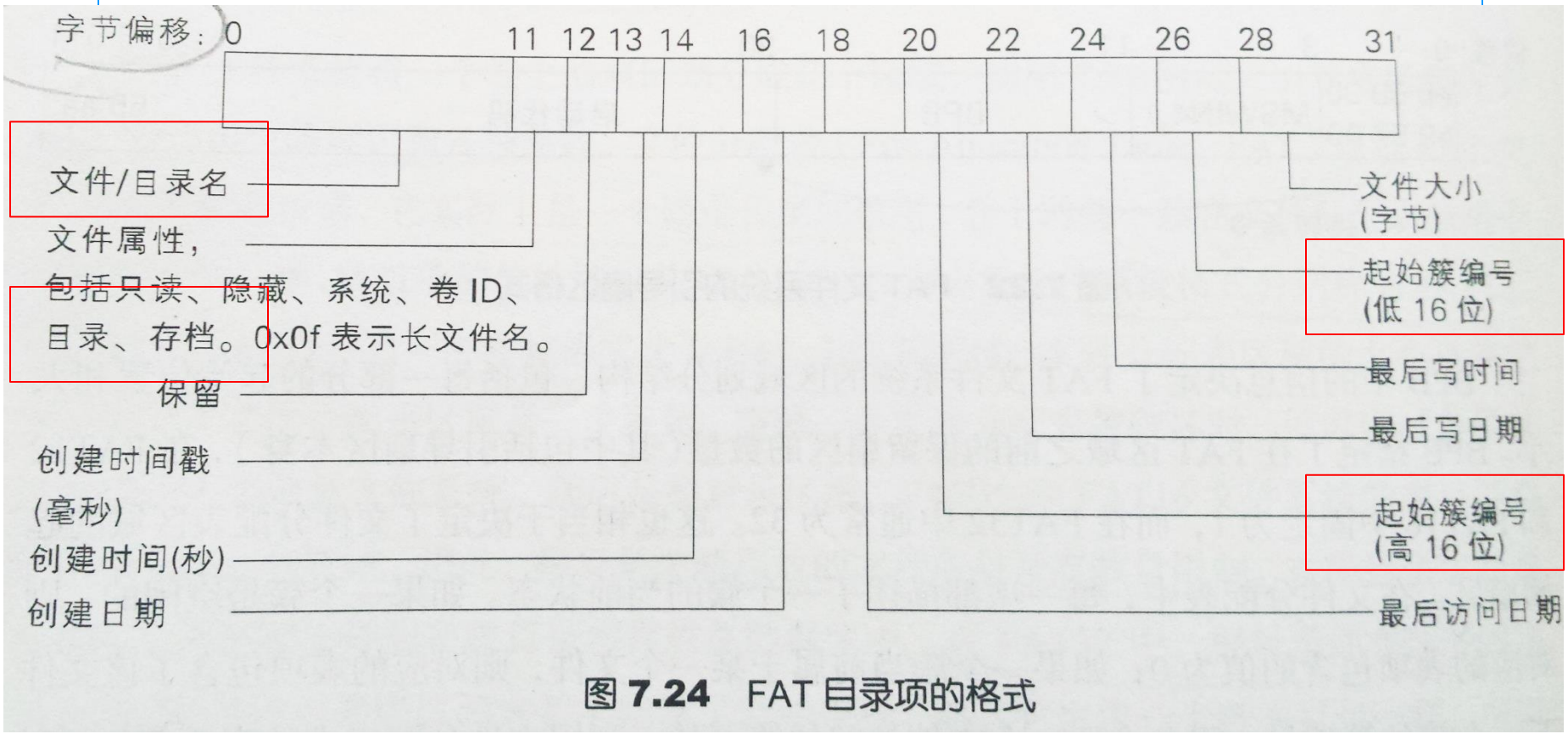
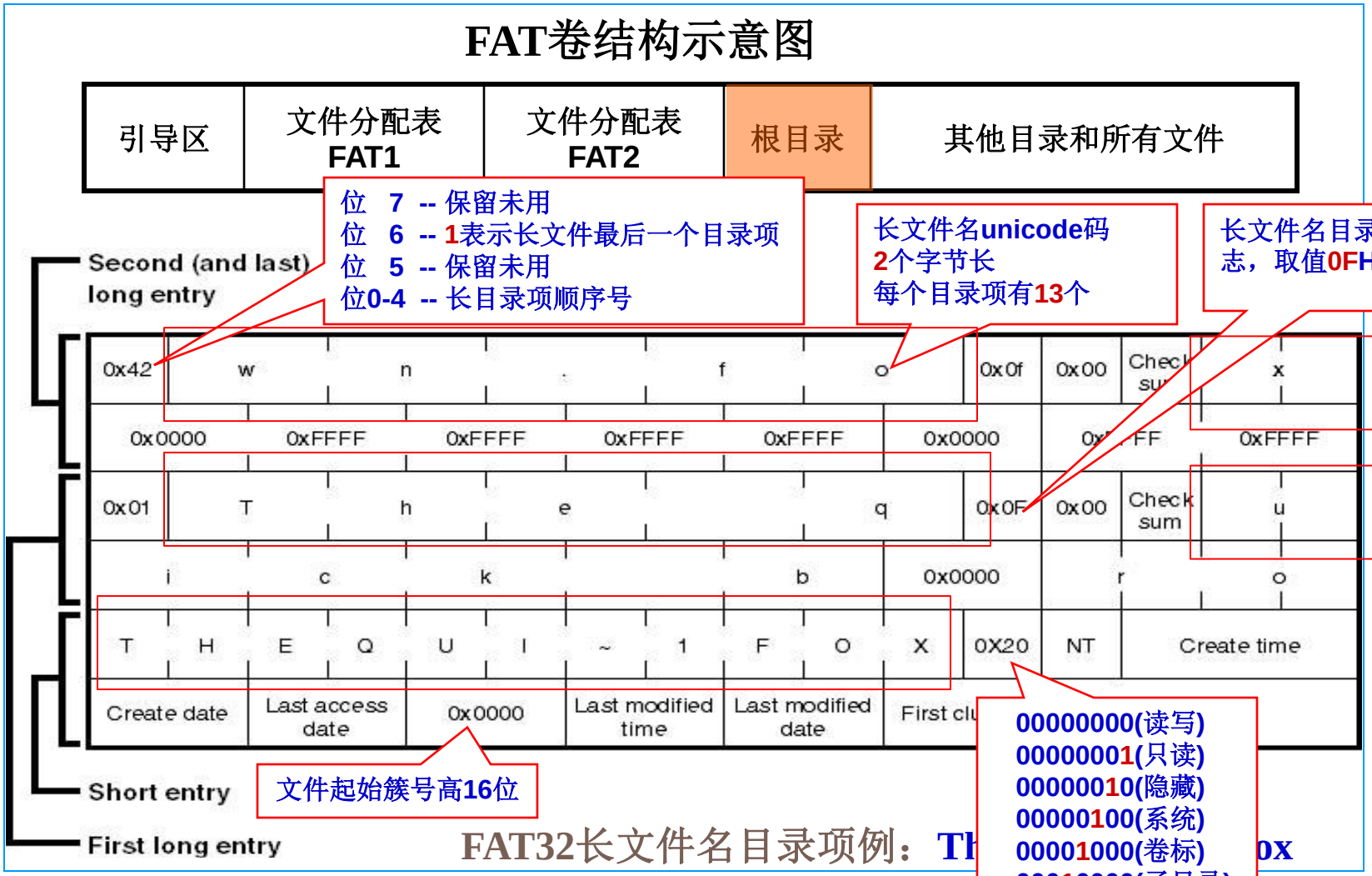
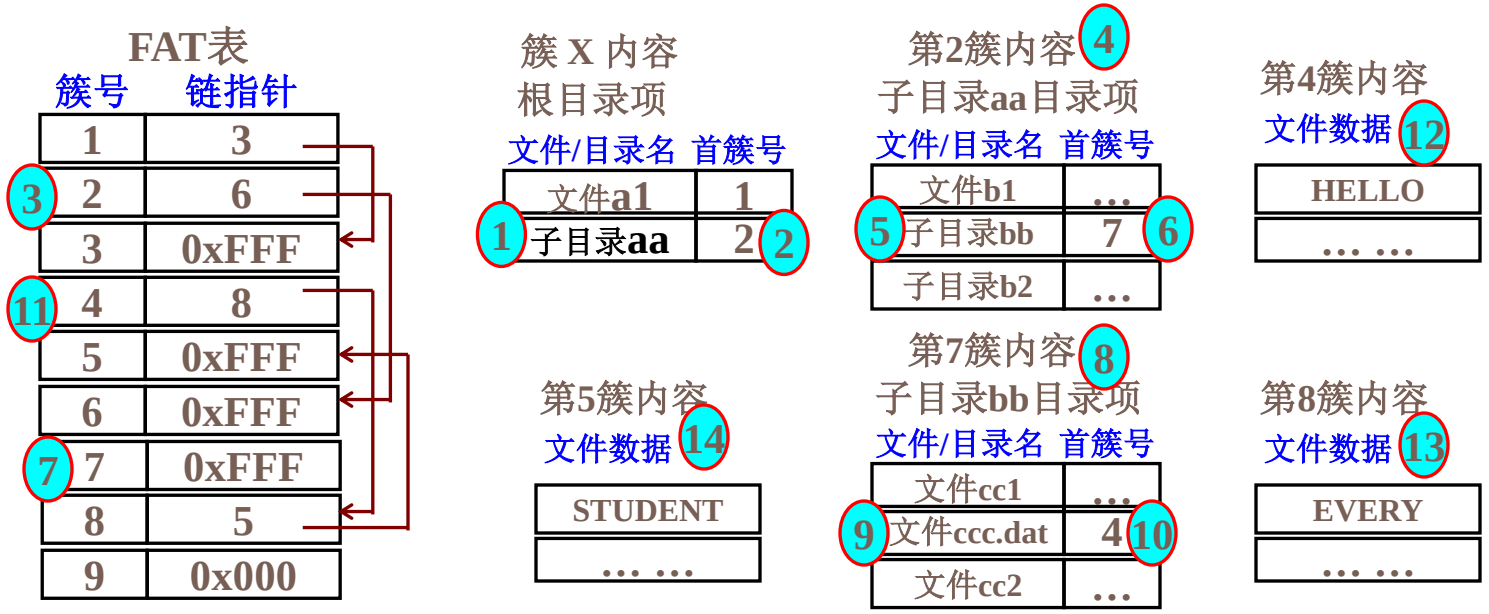


图 7.24　FAT 目录项的格式

Windows的FAT文件系统实现



Windows的FAT文件系统实现



文件访问过程：读出文件\aa\bb\ccc.dat的内容

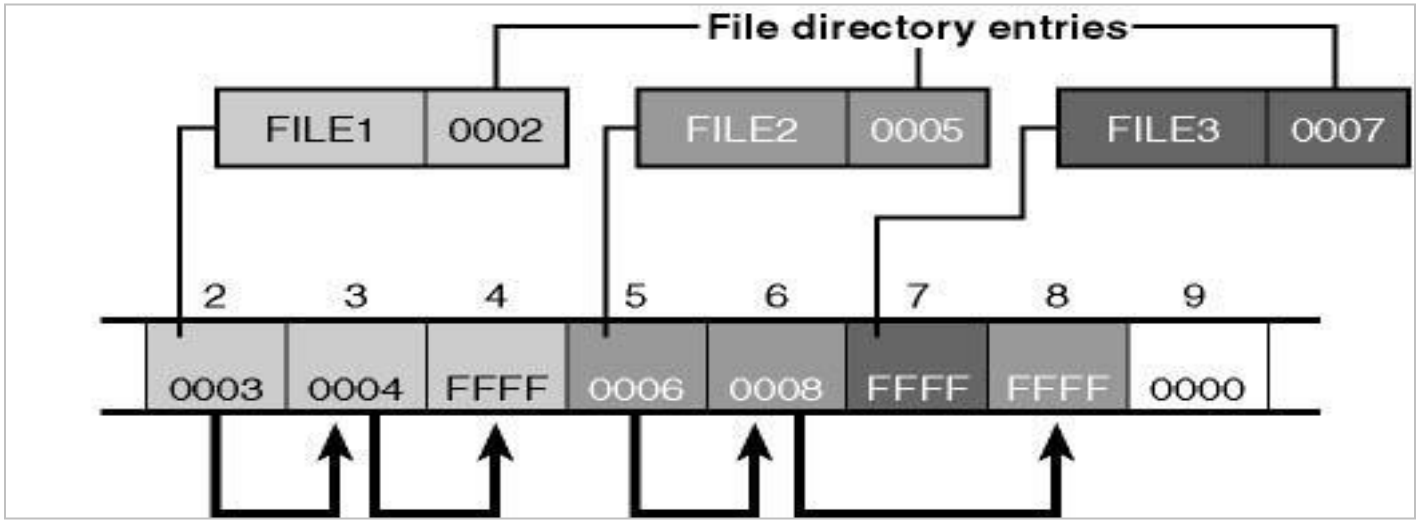
1. 查“根目录”中目录项：找到含目录名=“aa”的目录项
2. 从aa目录项中查出该目录文件的首簇号＝2
3. 查FAT1中以第2簇为头的文件分配簇链，检索相应簇内容找出含目录名=“bb”的目录项
4. 从bb目录项中查出该目录文件的首簇号=7
5. 查FAT1中以第7簇为头的文件分配簇链，检索相应簇内容找出含文件名=“ccc.dat”的目录项
6. 从ccc.dat目录项中查出该文件的首簇号=4
7. 查FAT1中以第4簇为头的文件分配簇链，读出相应簇内容，得到了文件ccc.dat的内容

Windows的FAT文件系统实现

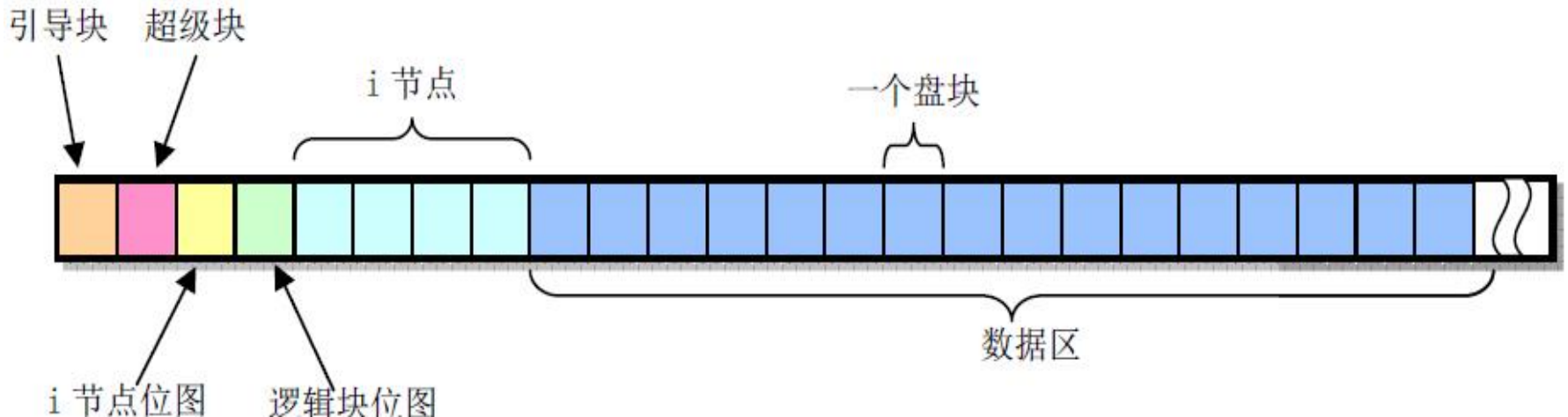
文件删除过程：删除文件\aa\bb\ccc.dat
文件建立过程：建立新文件\aa\fgx.txt

比较两种文件系统

引导区	文件分配表 FAT1	文件分配表 FAT2	根目录	其他目录和所有文件



FAT32中，文件分配表的表项既能记录目录或文件文名信息，同时标识族块是否被占用。



Windows的FAT文件系统实现

FAT卷容量与簇大小的关系

- **FAT12** 最大簇数 = 2^{12}个=4K个
  规定簇大小=0.5KB～8KB
  FAT12卷大小≤32MB
- **FAT16** 最大簇数 = 2^{16}个=64K个
  规定簇大小=0.5KB～64KB
  FAT16卷大小≤4GB
- **FAT32** 最大簇数 = 2^{32}个=4KM个
  限定使用2^{28}个=0.25G个
  规定簇大小=4KB～32KB
  FAT32卷大小≤8KGB

FAT16/FAT32限制单个文件最大为4GB

W2K系统默认FAT16卷缺省簇大小

Volume Size	Cluster Size
0-32MB	512bytes
33MB-64MB	1KB
65MB-128MB	2KB
129MB-256MB	4KB
257MB-511MB	8KB
512MB-1023MB	16KB
1024MB-2047MB	32KB
2048MB-4095MB	64KB

FAT32卷缺省簇大小

Volume Size	Cluster Size
32MB-8GB	4KB
8GB-16GB	8KB
16GB-32GB	16KB
>32GB	32KB

空间大小，簇的大小影响了，文件分配表项数，效率

为什么在同一种文件系统中簇的大小不统一？

Windows的FAT文件系统实现

文件系统总结

- 文件 ⇒ 帮助用户将信息放在磁盘上
- 一个长长的字符序列 ⇒ 盘块集合 ⇒ 文件系统完成映射
- 操作(文件名,偏移) ⇒ 找到文件头(FCB)，找到磁盘盘块
- 系统中会有很多文件 ⇒ 为方便寻找，组织成树状目录
- 目录表明某些文件在一个集合中 ⇒ 目录也是文件，其内容是<文件名，FCB的指针>
- 树状目录 ⇒ 文件路径 ⇒ 路径解析
- 目录+FCB+文件盘块+空闲盘块 = 文件系统
- 从可以运转到良好运转 ⇒ 高效、保护、容错…