

# 文件系统

# 阅读任务

- 第39节 “File and Directory”
- 第42节 “Crash Consistency: FSCK and Journaling”

# 本章教学目标

- 理解文件的概念
- 理解文件目录的概念和组织方式
- 理解文件的逻辑结构和组织方式
- 掌握文件的物理结构和组织方式
- 理解文件空间管理的方法
- 理解Linux文件系统中文件系统调用

# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件系统

- 文件系统的功能：
  - 文件的按名存取
  - 文件目录的建立和维护
  - 文件的查找和定位
  - 文件存储空间的分配和管理
  - 提供文件的存取方法和文件存储结构
  - 实现文件的共享、保护和保密
  - 提供一组易用的文件操作和命令
  - 提供与设备管理交互的统一接口

# 文件的概念

- 文件是操作系统对所存储的信息对外提供的统一逻辑视图
- 文件是由文件名标识的一组存储在二级存储设备上的信息的集合
- 从用户的角度，文件是二级存储设备最小的逻辑分配单元
- 用户按名存取文件，操作系统负责文件名到物理存储位置的映射
- 文件是由位串、连续的字节、行、或记录构成的集合，文件内容的含义取决于文件创建者

# 文件类型

- 按用途分
  - 系统文件
  - 库文件
  - 用户文件
- 按保护级别分
  - 只读文件
  - 读写文件
  - 不保护文件
- 按数据类型分
  - 源程序文件
  - 目标文件
  - 可执行文件

# UNIX/Linux的文件类型

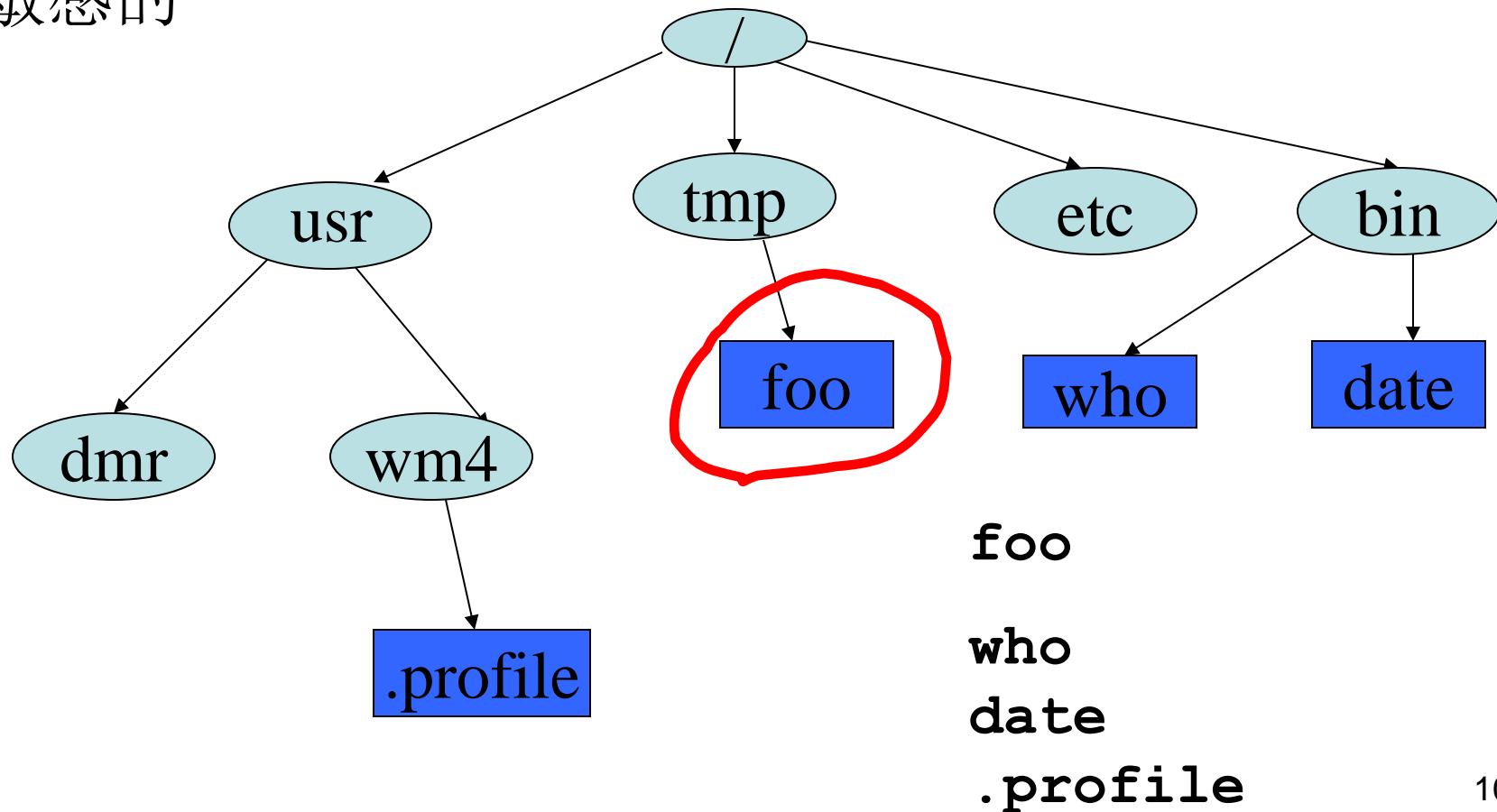
- 普通文件
  - ASCII文件
  - 二进制文件
- 目录文件
  - 由文件目录所构成的用来维护文件系统结构的系统文件
- 特别文件
  - 指各类外部设备文件
  - 将所有I/O设备统一在文件系统下

# 文件名与文件类型

- 各个操作系统的文件命名规则略有不同
- 文件名一般由文件名称和扩展名称组成，两者中间以“.”分割，如“myfile.doc, command.exe”
  - 文件名用于识别文件
  - 扩展名用于区分文件类型

# 文件名

由字符（除了/）构成的字符串，有可能是大小写敏感的



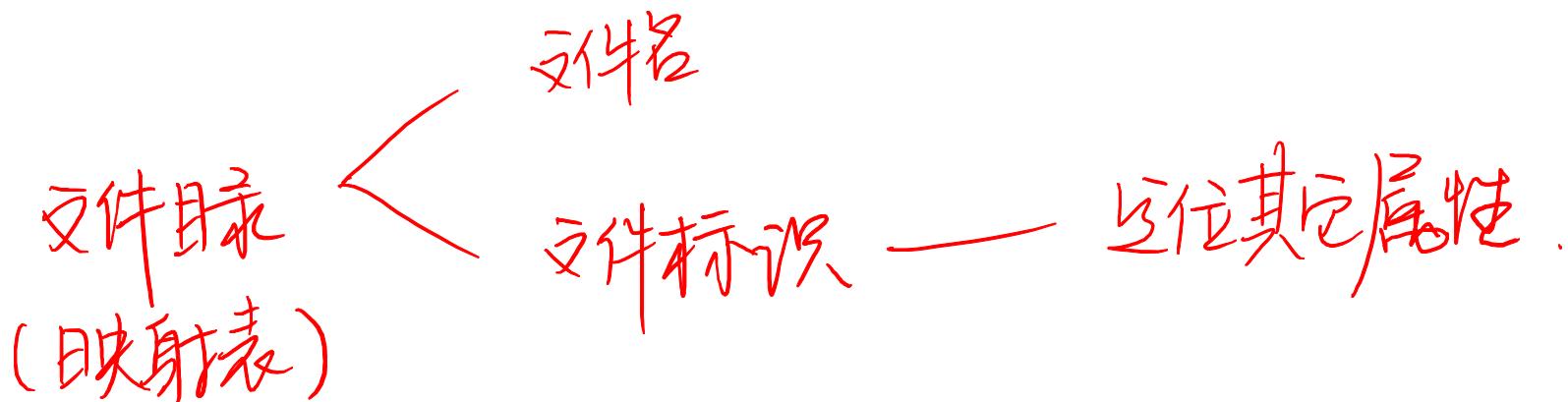
file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

# 文件属性

- 文件名
  - 唯一用户可读的信息
- 文件标识
  - 文件系统内的唯一标识
- 类型
  - 普通文件、目录文件、设备文件
- 位置
  - 指向文件的位置，包括设备以及设备上的位置
- 大小
  - 当前长度和允许的最大长度
- 权限
  - 访问控制信息，例如谁可以读、写、执行文件等
- 时间
  - 文件的创建时间，最后修改时间，最后使用时间等

# 文件属性

- 与文件相关的所有信息保存在文件目录结构中
- 文件目录通常包含文件名和文件标识两部分
- 文件标识用于定位其它属性



# 文件操作

- **创建文件(create)**
  - 从文件系统中找到存放文件的空间（空间分配问题）
  - 在目录中为该文件添加新目录项
- **写文件(write)**
  - 给出文件名和需要写入文件的信息
  - 文件系统查找目录，从文件目录项中找到文件的位置
  - 文件系统保存了一个写指针，指出下一个写操作开始的位置
- **读文件(read)**
  - 给出文件名以及内存地址，用于存放读出的下一个文件块
  - 文件系统查找目录，找到对应的文件目录项
  - 系统保存一个读指针，指出下一个读操作开始的位置
    - 读指针每个进程都不一样

# 文件操作(cont)

- 重定位(repositioning, seek)
  - 查找文件目录，找到对应的目录项，将当前文件位置指针修改为给定的值
  - 文件重定位并不涉及真正的I/O操作
- 删除文件(delete) *(判断是否共享)*
  - 查找文件目录，找到给定文件名对应的目录项
  - 释放文件所占用的所有空间
  - 删除对应的目录项
- 清空文件
  - 删除文件内容，但保留其属性（长度除外）

# 文件操作(cont)

- 追加(appending)
- 重命名(renameing)
- 拷贝(copy)
- 获取和修改文件属性(get/set attribute)

# 文件存取方法

- 顺序存取
  - 存取操作在上次的基础上进行
  - 系统设置读写指针，指向要读出或写入的字节位置或记录位置。
- 直接存取
  - 快速地以任意次序直接读写某条记录，对文件读/写的次序没有任何限制
- 索引存取
  - 基于索引文件的存取方法
  - 用户提供记录名或记录键，按名搜索，找到所需要的记录  
*key — value*

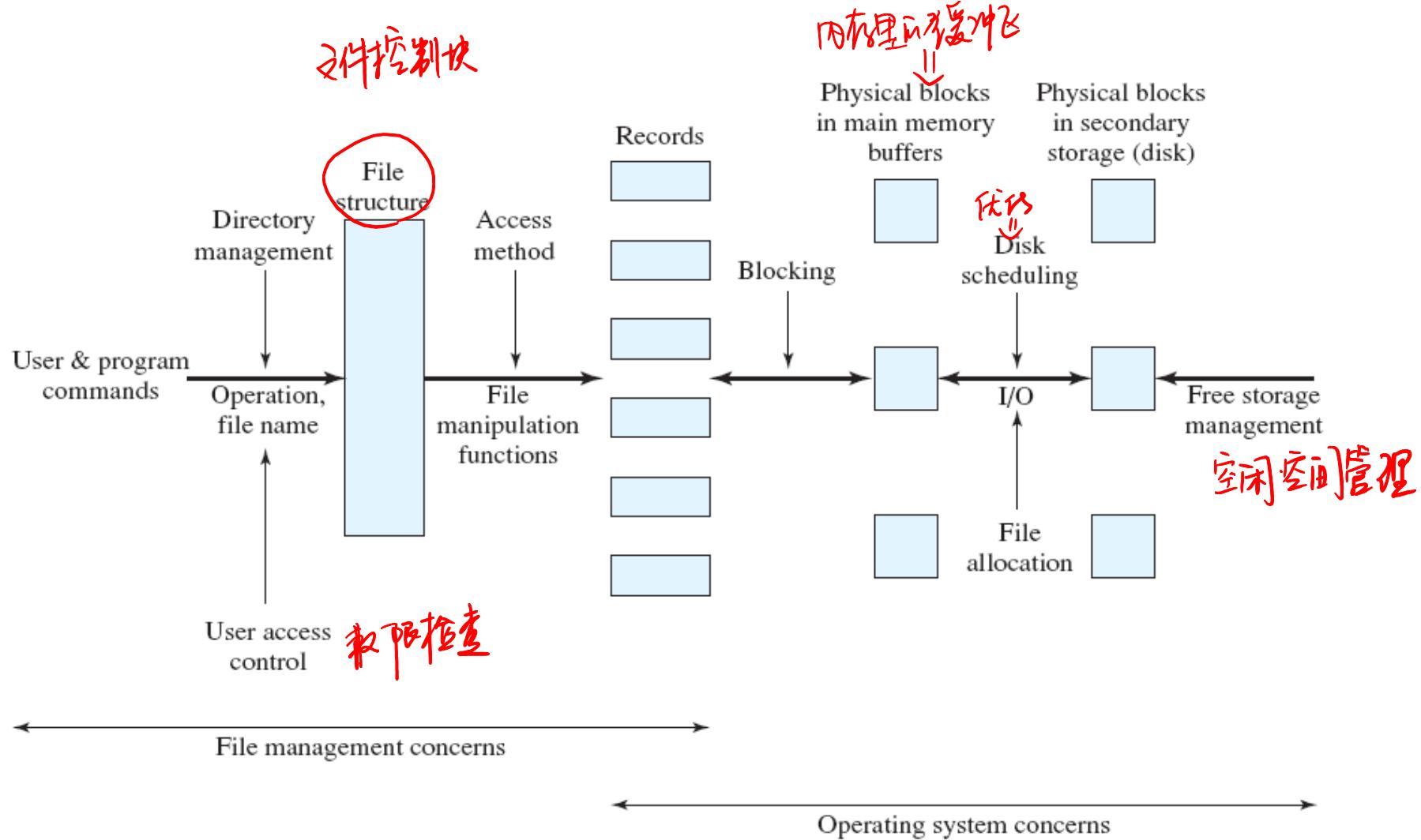


Figure 12.2 Elements of File Management

# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件控制块(FCB)

- 文件系统为每个文件建立的唯一的管理数据结构，一般包括：
  - 文件标识和控制信息
    - 文件名、用户名、文件存取权限、访问控制权限、文件类型等；
  - 文件逻辑结构信息（仅仅对记录型的）
    - 记录类型、记录个数、记录长度、成组因子等；
  - 文件物理结构信息
    - 文件所在设备名、文件物理结构类型、记录存放在辅存中的块号或文件信息首块盘块号，文件索引的位置等；
  - 文件使用信息 有多个用户 Map
    - 共享文件的进程数，文件修改情况，文件最大长度和当前大小
  - 文件管理信息
    - 文件建立日期，最近修改日期，最近访问日期等；

# FCB

- 基于FCB可以方便地实现文件的按名存取
  - 创建文件时，为其建立一个FCB，用来记录文件的属性信息
  - 存取此文件时，先找到其FCB，再找到文件信息盘块号或首块物理位置
- 为了加快文件的查找速度，通常将FCB集中起来进行管理，组成文件目录 树型查找

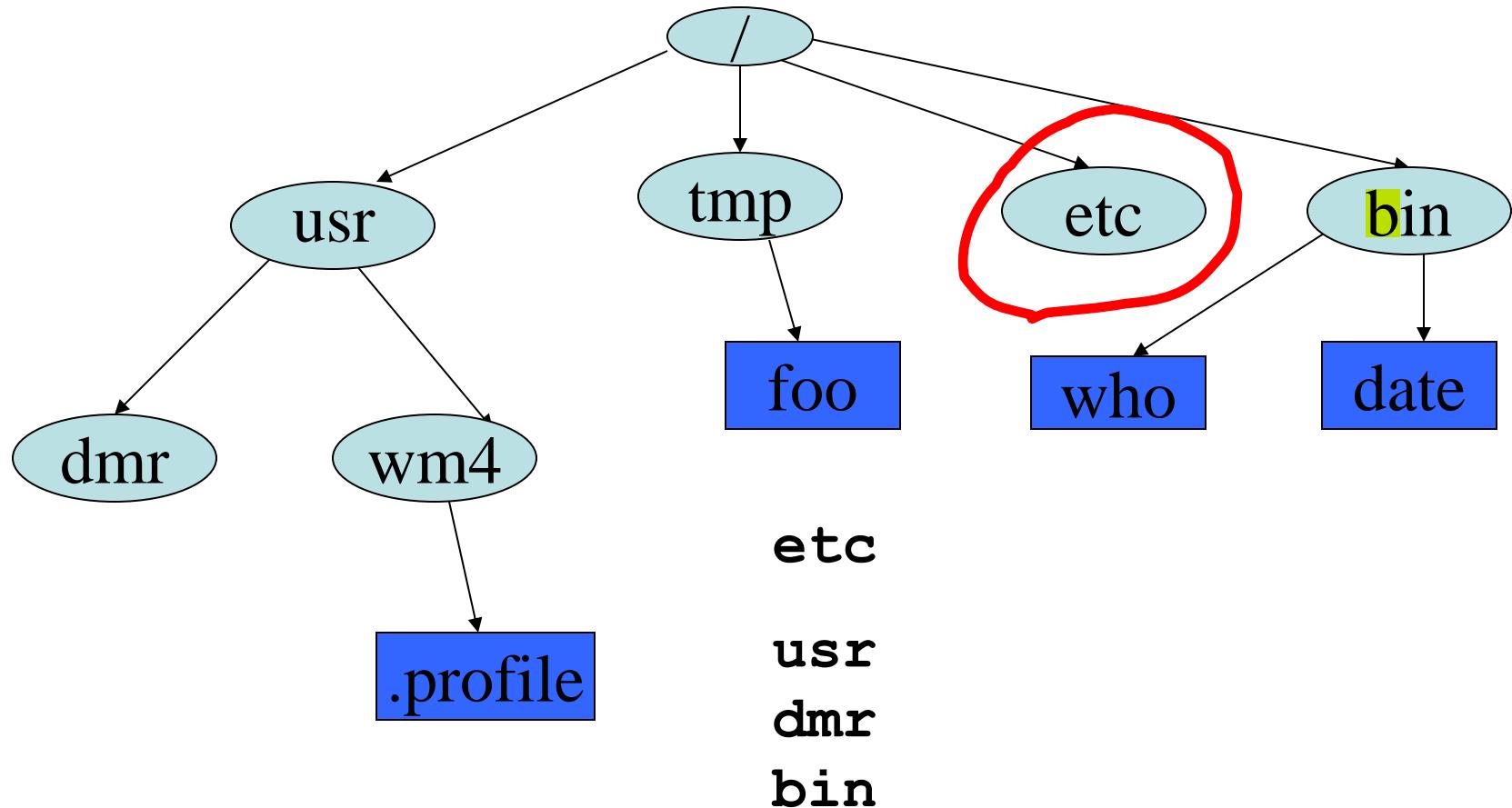
# 文件目录

文件名 - PCB = id

- 文件目录包含两种目录项：
  - 文件的FCB (普通文件)
  - 描述子目录的目录文件的FCB (子目录)
- 全部由目录项构成的文件称为目录文件
  - 目录项的格式按统一标准定义
- 目录文件至少包含两个目录项
  - 当前目录项 “.”
  - 父目录项 “..”
- 文件目录的基本功能是将文件名转换成此文件信息在磁盘上的物理位置

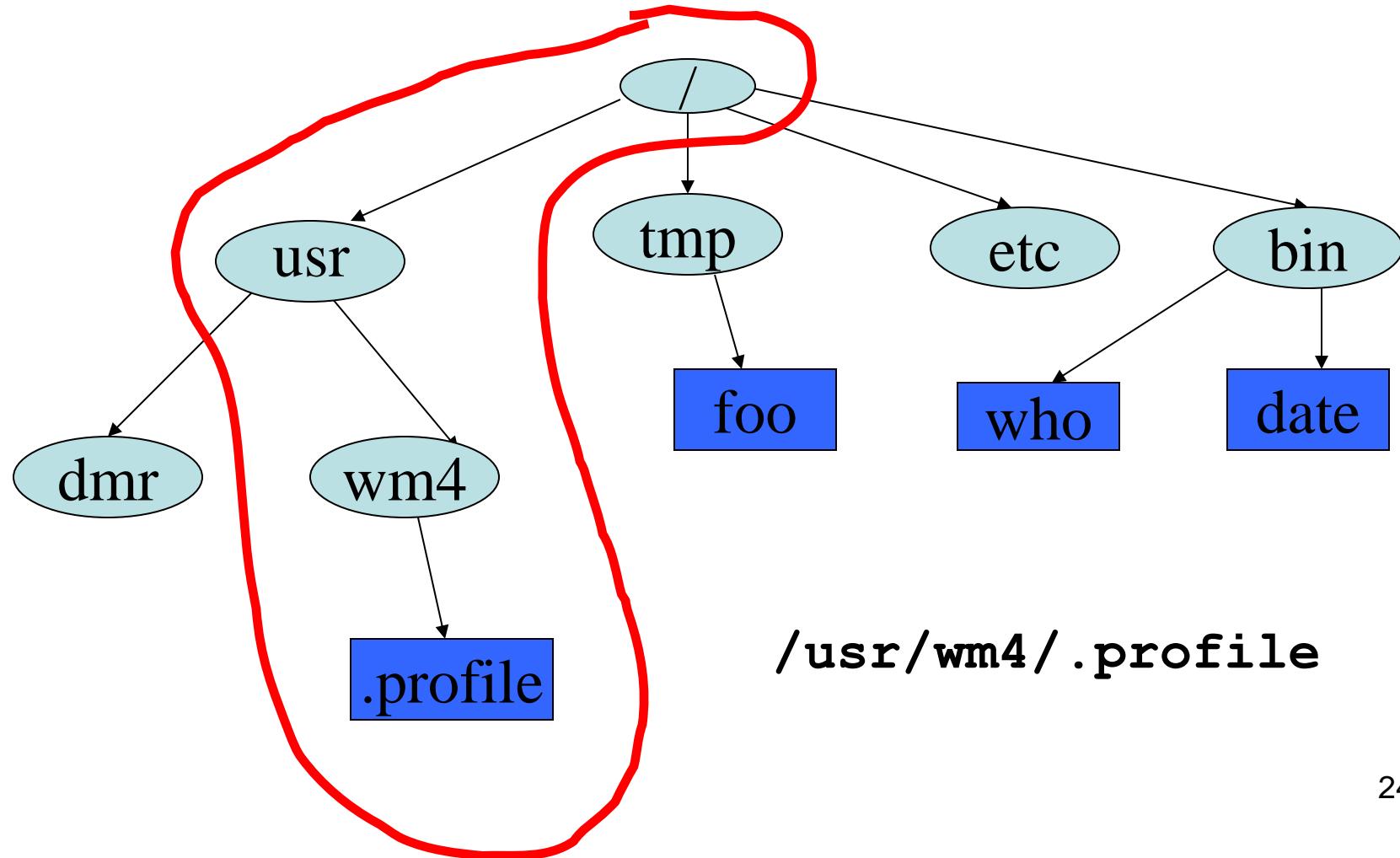
# 目录

○ 目录  
□ 文件



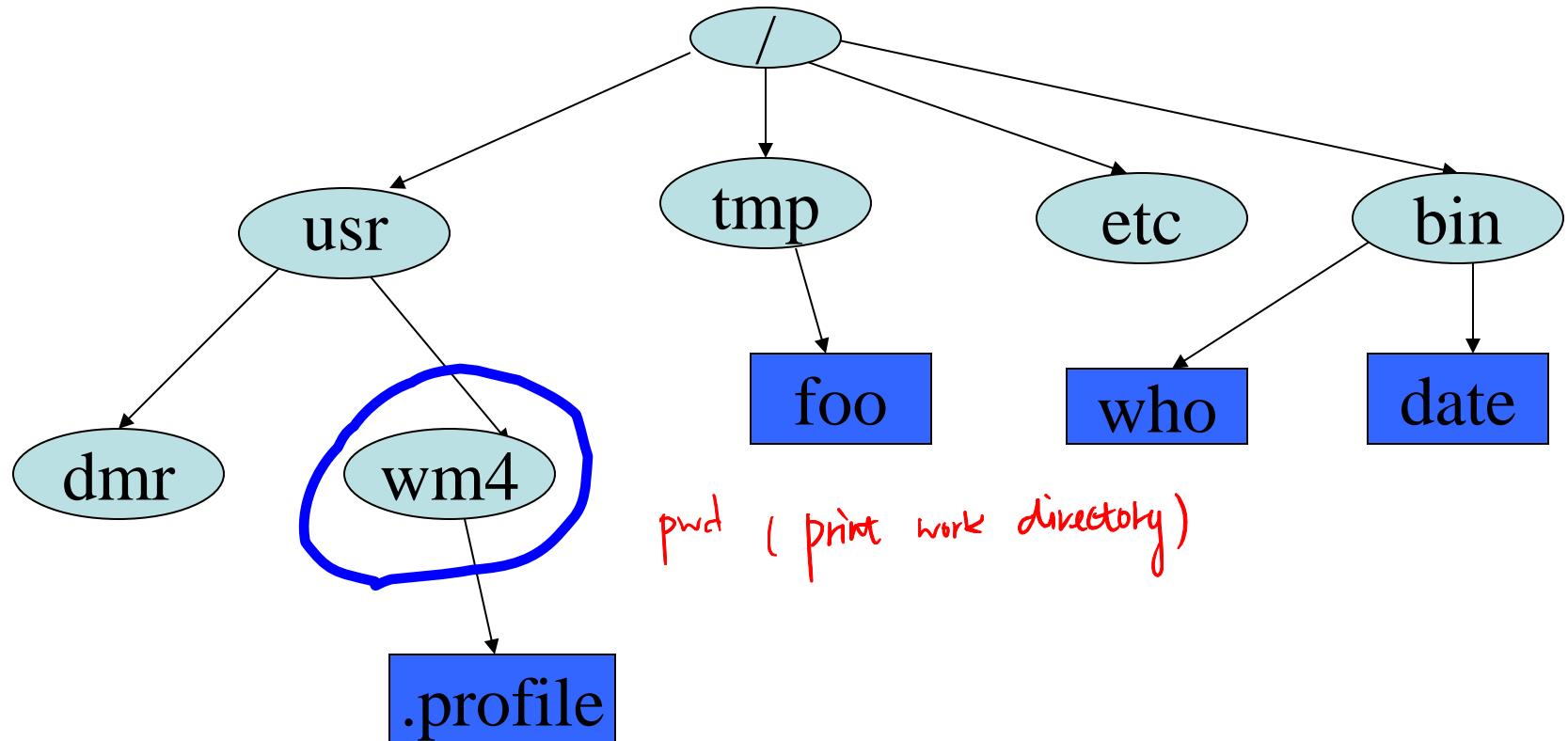
# 文件路径

由从根目录开始沿着目录结构一直到文件的目录项和文件名顺序组合而成



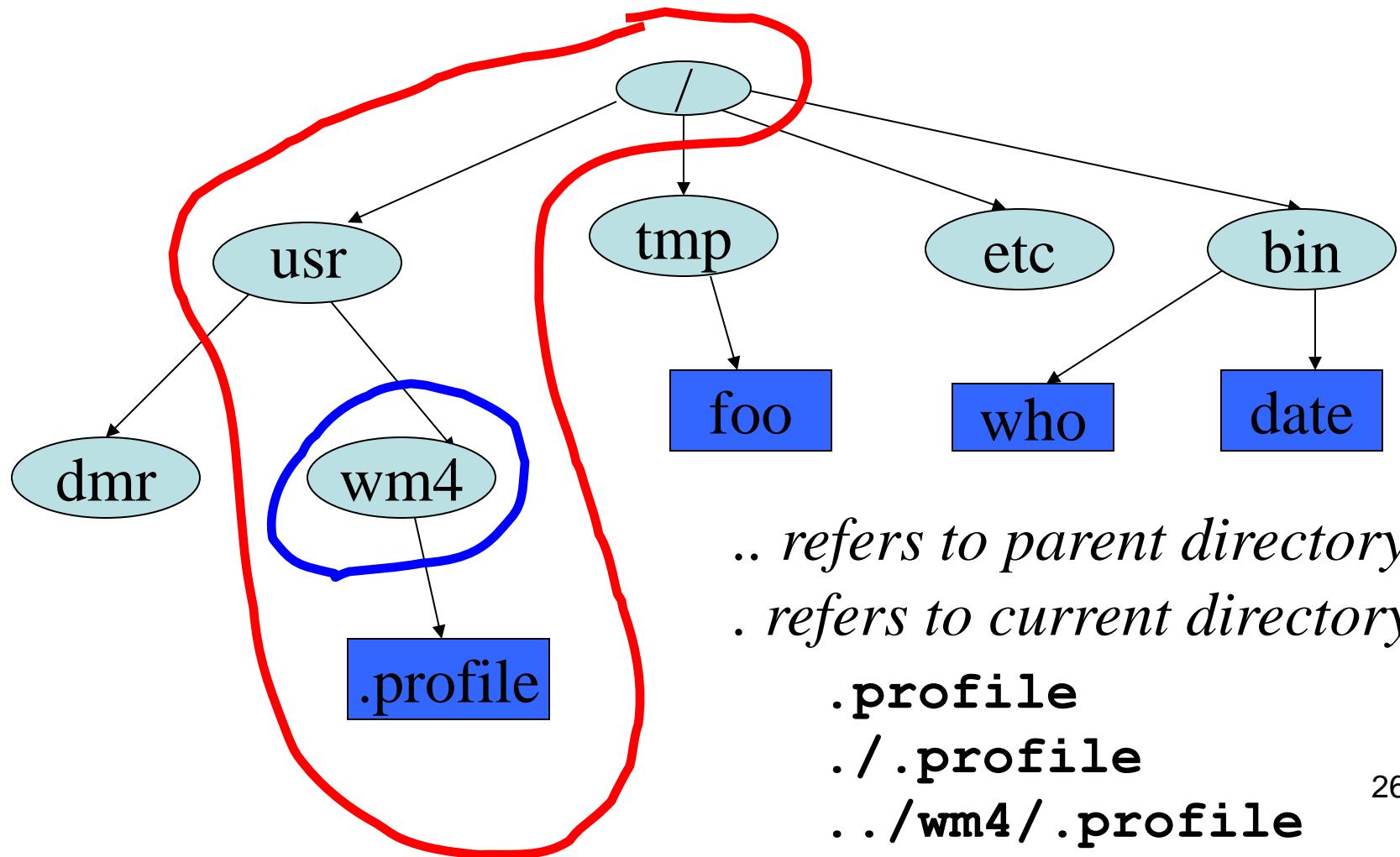
# 工作目录

- 默认情况下文件名所处的目录



# 相对路径

相对于当前工作目录的路径

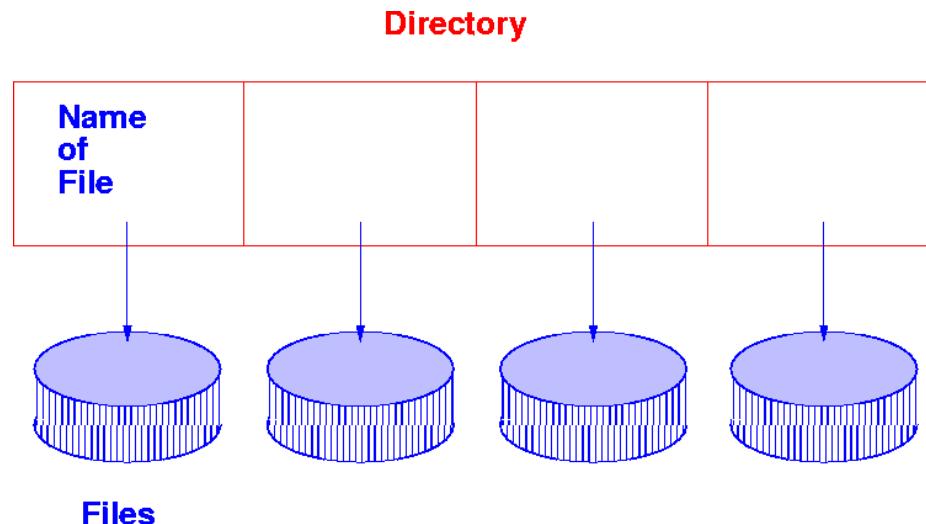


# 目录结构

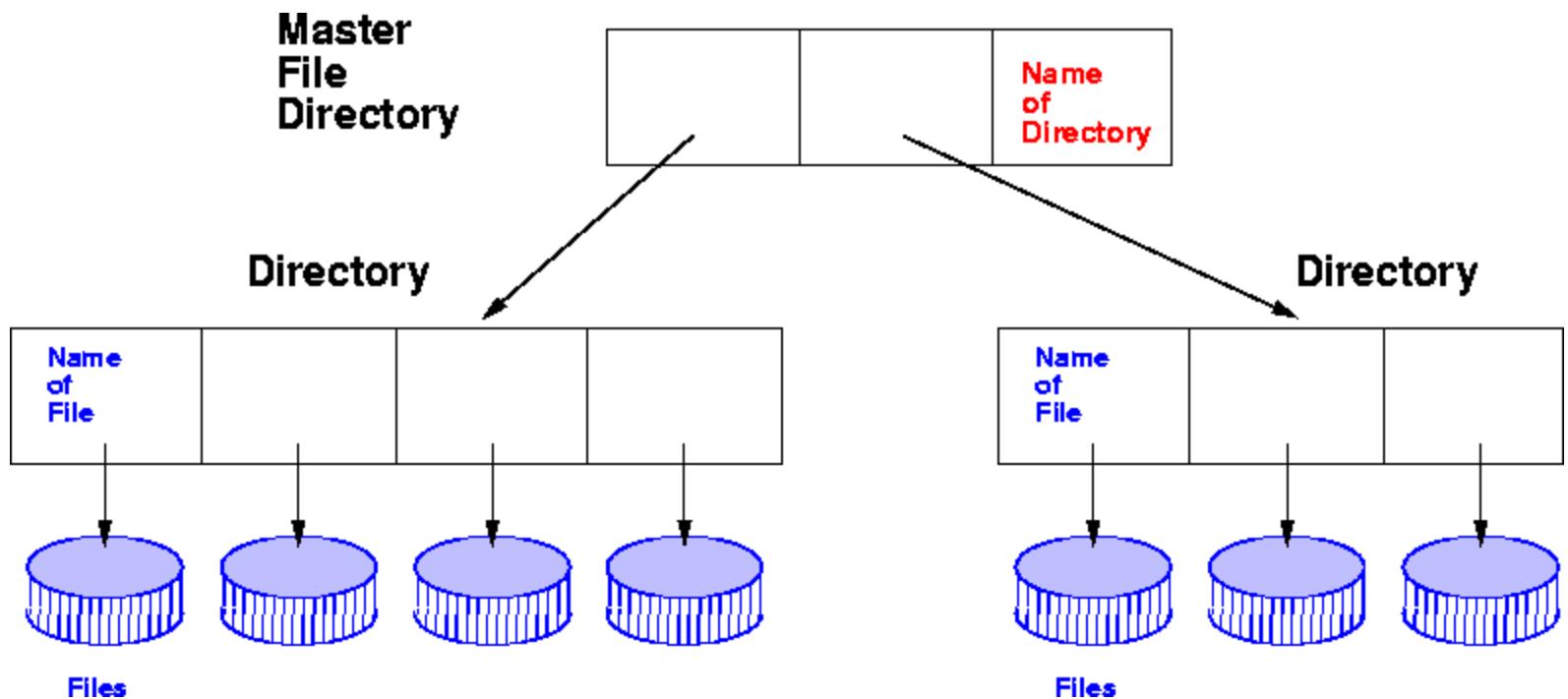
- 单级目录结构
- 两级目录结构
- 树形目录结构
- DAG结构 有向无环图 *Directed Acyclic Graph*.
- 图结构

# 单级目录结构

- 所有用户都共用一个目录
- 缺点：
  - 名字易于重复
  - 用户没有独立的目录，不利于共享和保护

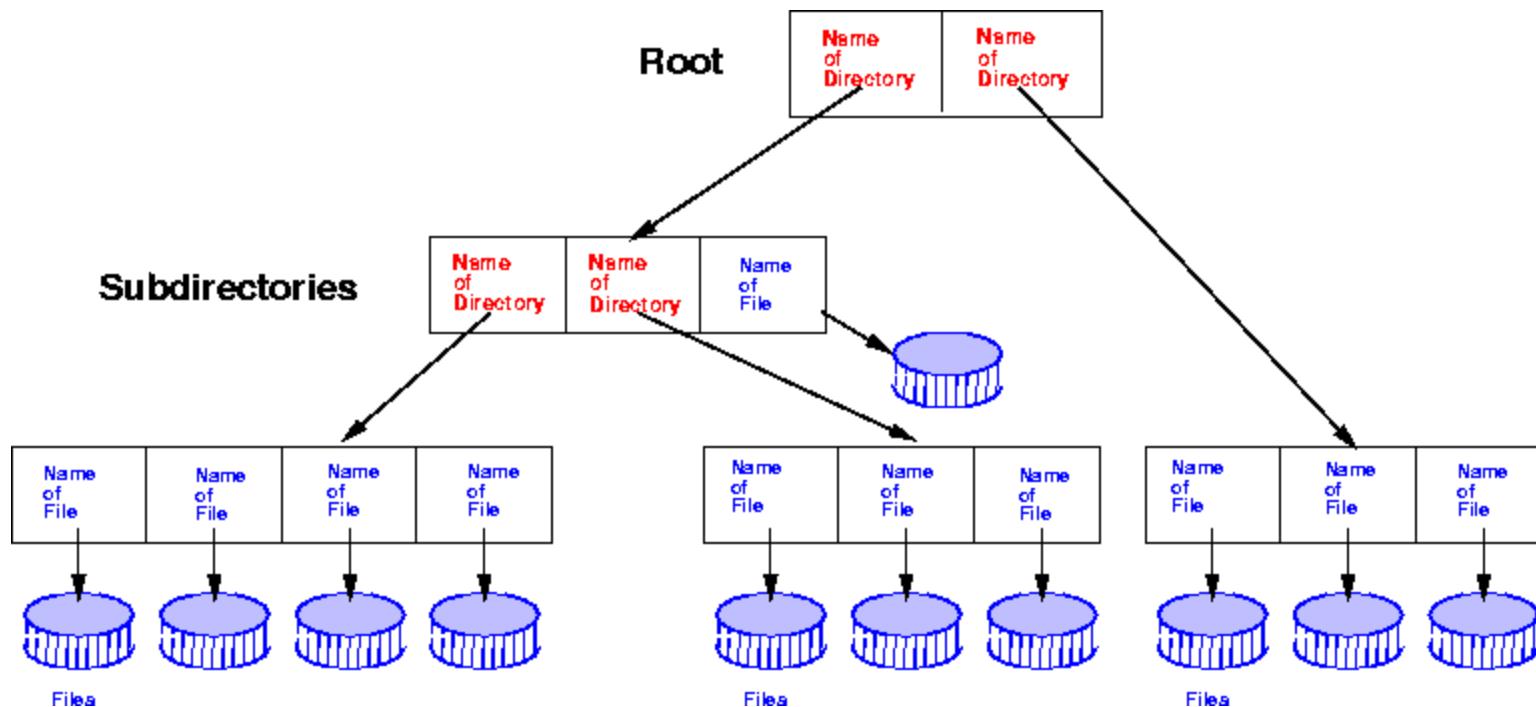


# 两级目录结构



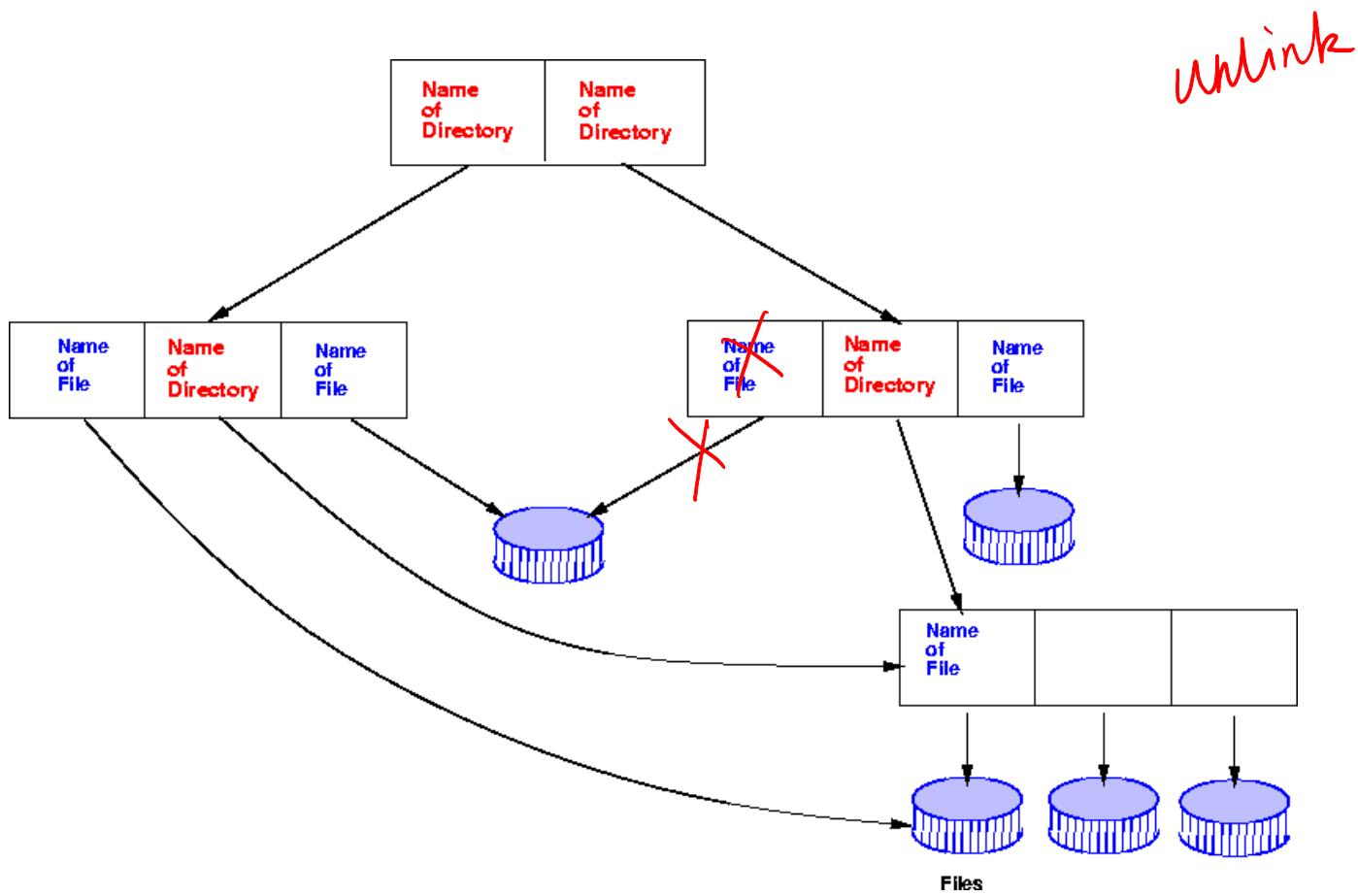
# 纯树形目录结构

- 目录可以有任意多的层次
- 目录可以包含子目录，也可以包含文件
- 每个文件~~只有一个父目录~~
- ~~文件共享较为困难~~



# DAG 目录结构

- 一个文件可以有多个父目录
- 能较方便地实现文件的共享
- 目录结构的维护较复杂
- 需要为每个文件维护一个引用计数，以记录文件的父目录个数，仅当引用计数值为1时，删除操作才真正删除文件

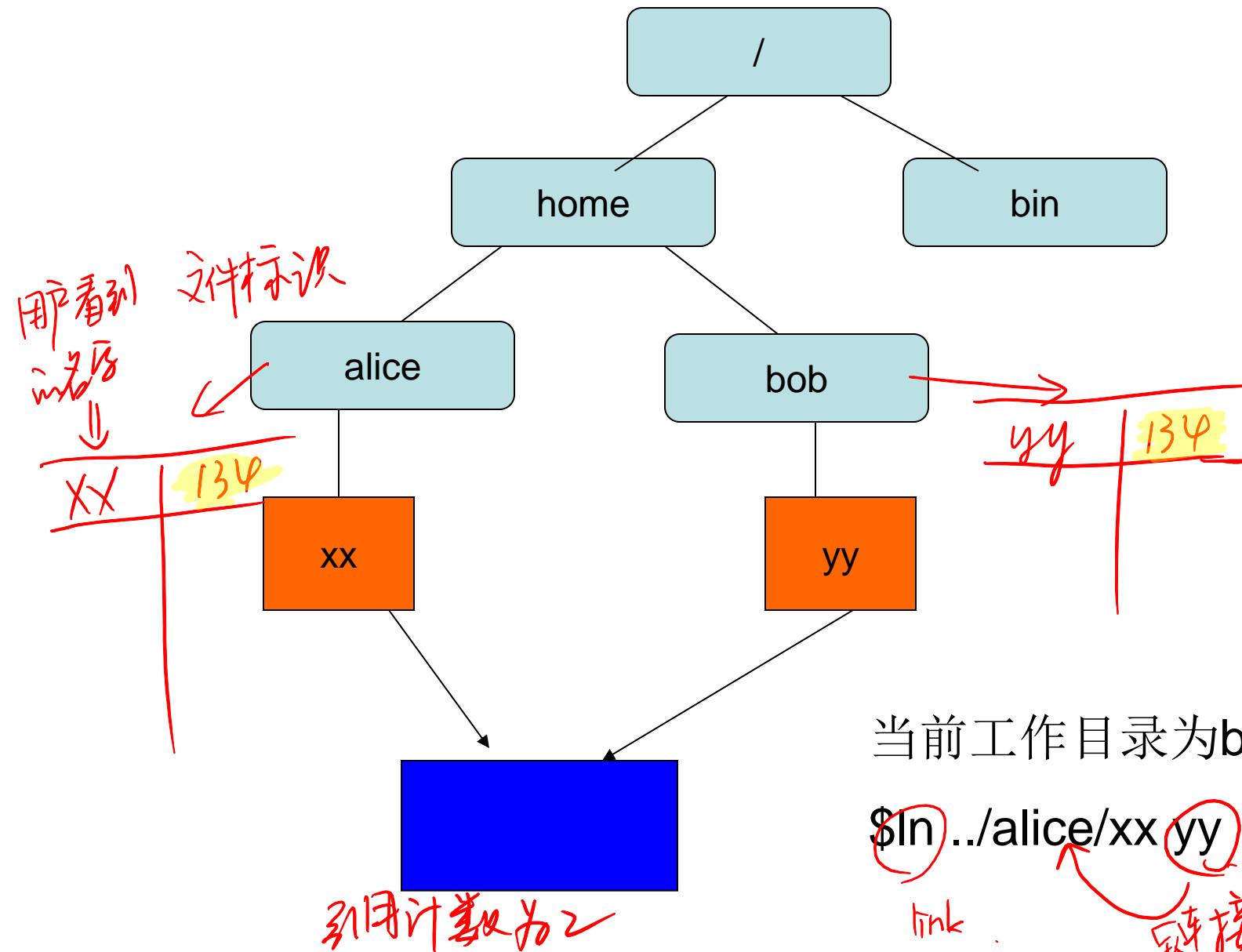


DAG目录结构示意图

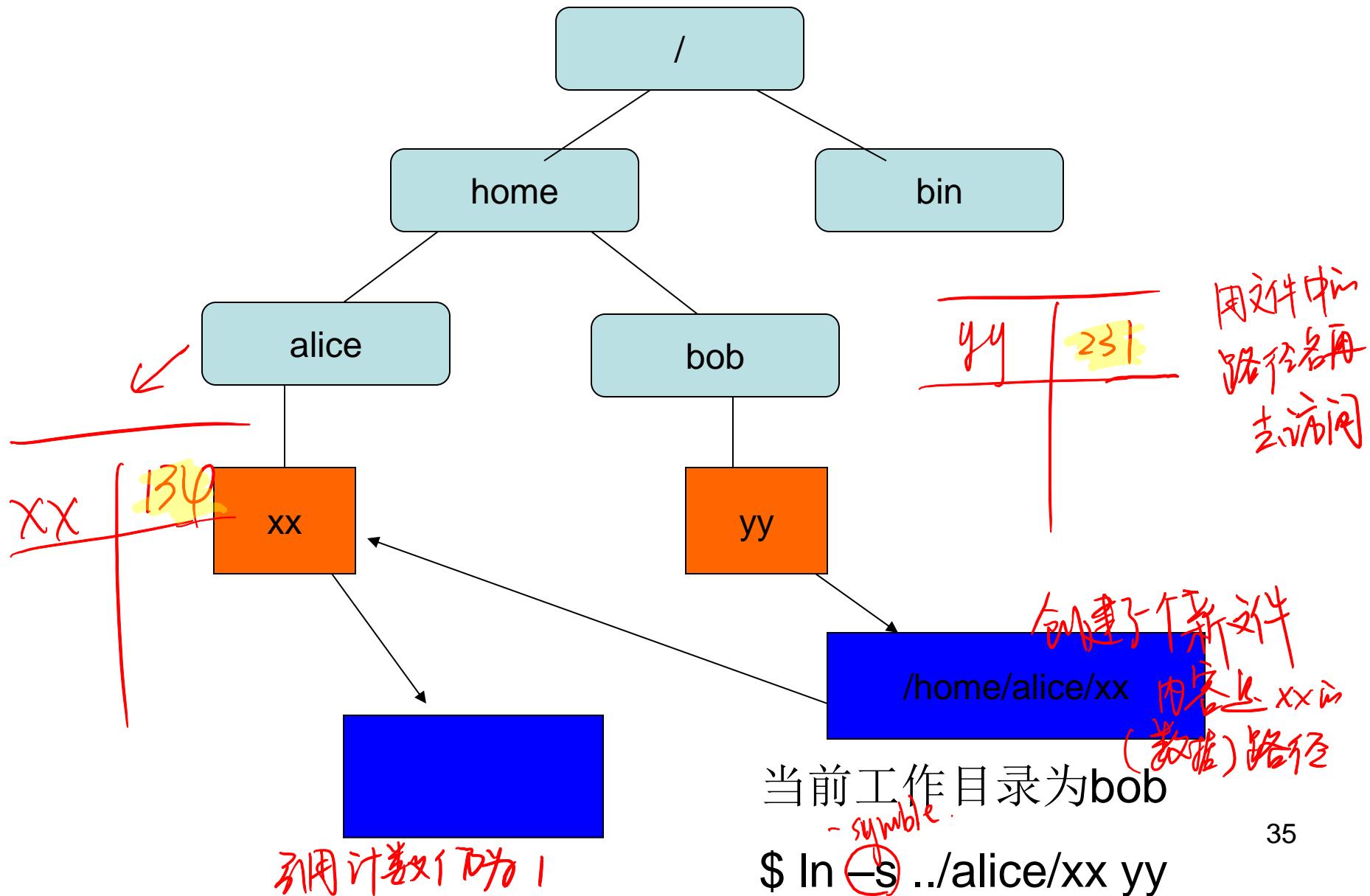
# 文件共享的实现方式

- LINUX
  - Hard link
  - Symbolic link *符号连接*
- Windows
  - 快捷方式

# Hard Link

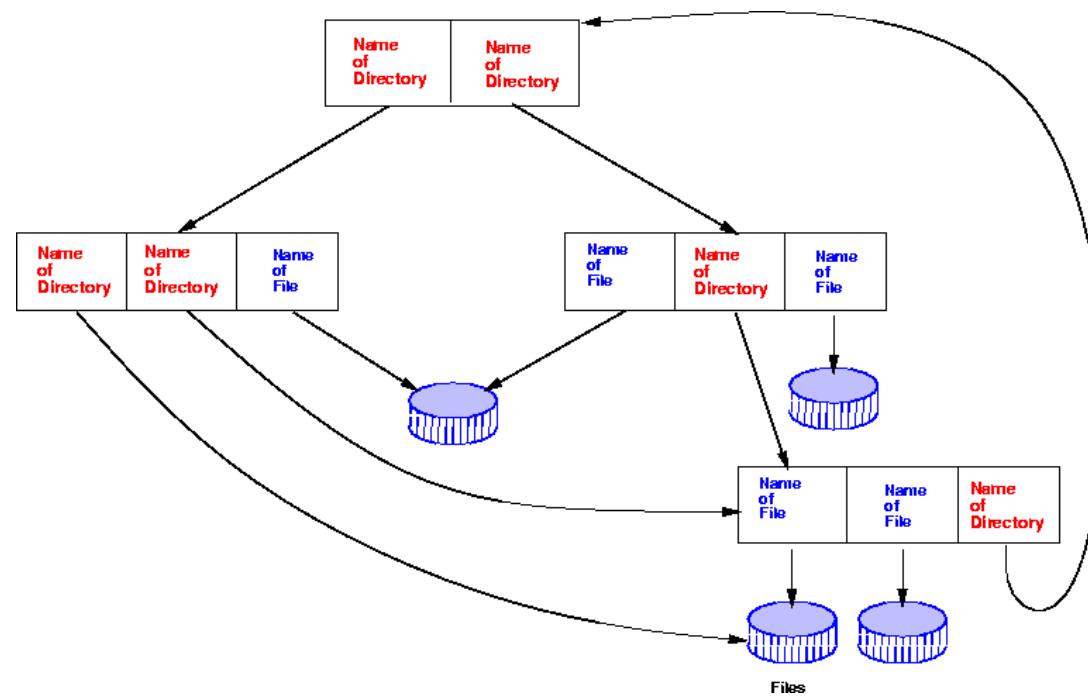


# Symbolic Link



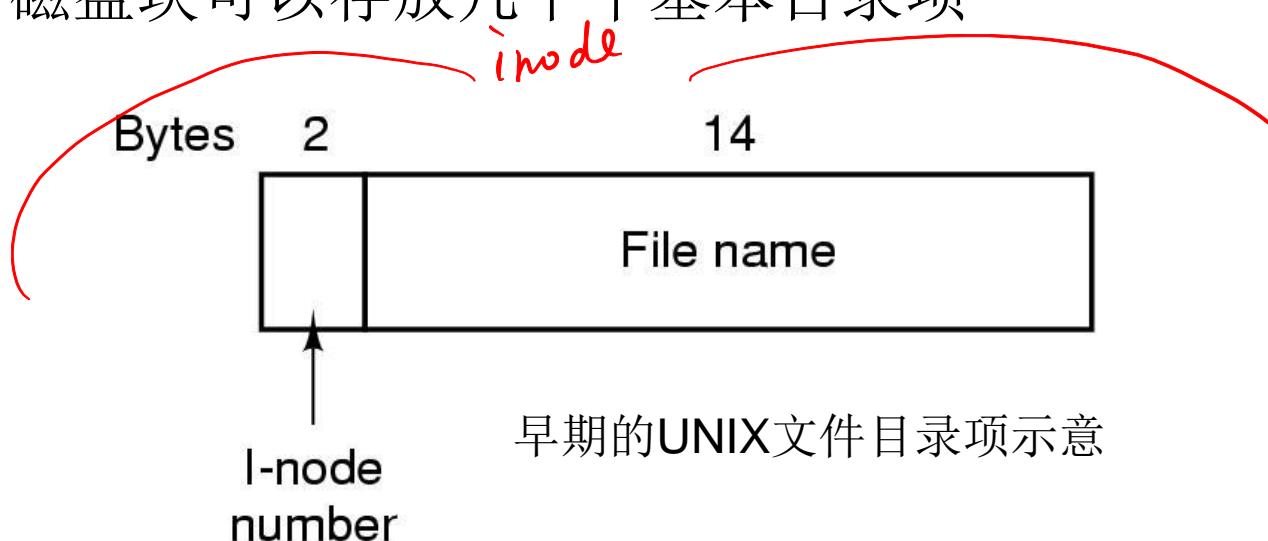
# 任意图结构

- 目录可能包含环
- 查找比较困难

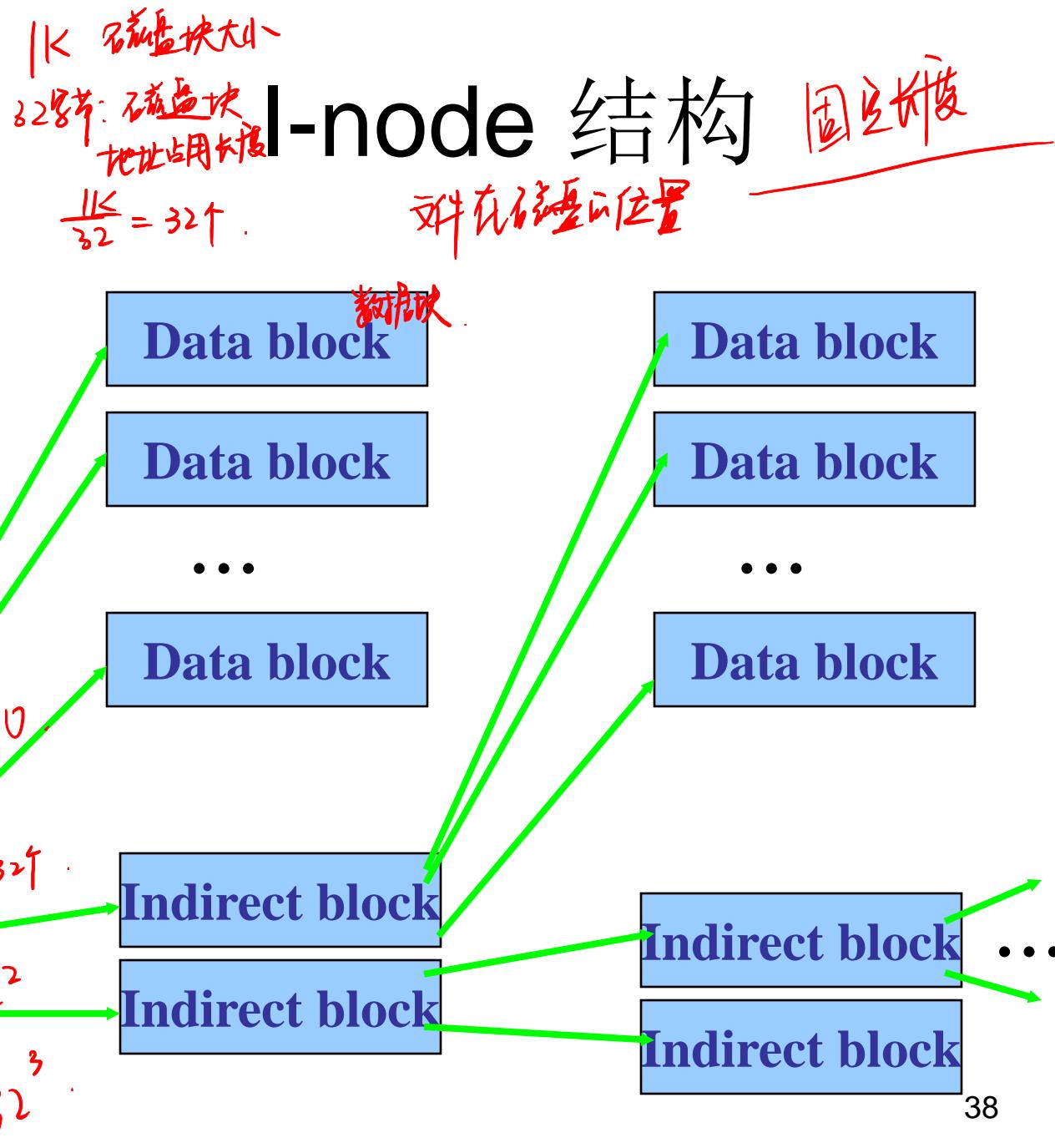


# Linux/Unix的目录项结构

- 将文件名和文件属性分离，其它信息单独组成一个数据结构，称为索引节点inode，其位置由inode号标识
- 每个文件或子目录都在父目录文件中有一条目录项
- 每个目录项包含两个字段
  - 文件名
  - i-node号：指出存放文件属性信息的inode节点号
- 每个磁盘块可以存放几十个基本目录项

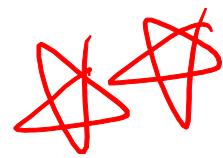


mode	
owner	
timestamp	
Size	
<u>Reference count</u>	引用计数
Block count	$\geq 10$
Direct blocks 0-9	第0-9块
Single indirect (10 ~ 41)	第10-41块
Double indirect	第32-111块
Triple indirect	第4096-11111块



# inode

- Unix/Linux操作系统对由文件目录项组成的目录文件和普通文件同等对待，均存放在磁盘中
- 文件系统中的每个文件都有一个磁盘inode与之对应，这些inode存放在磁盘的inode区
- 找到文件的inode就能找到文件在物理磁盘中的存放位置

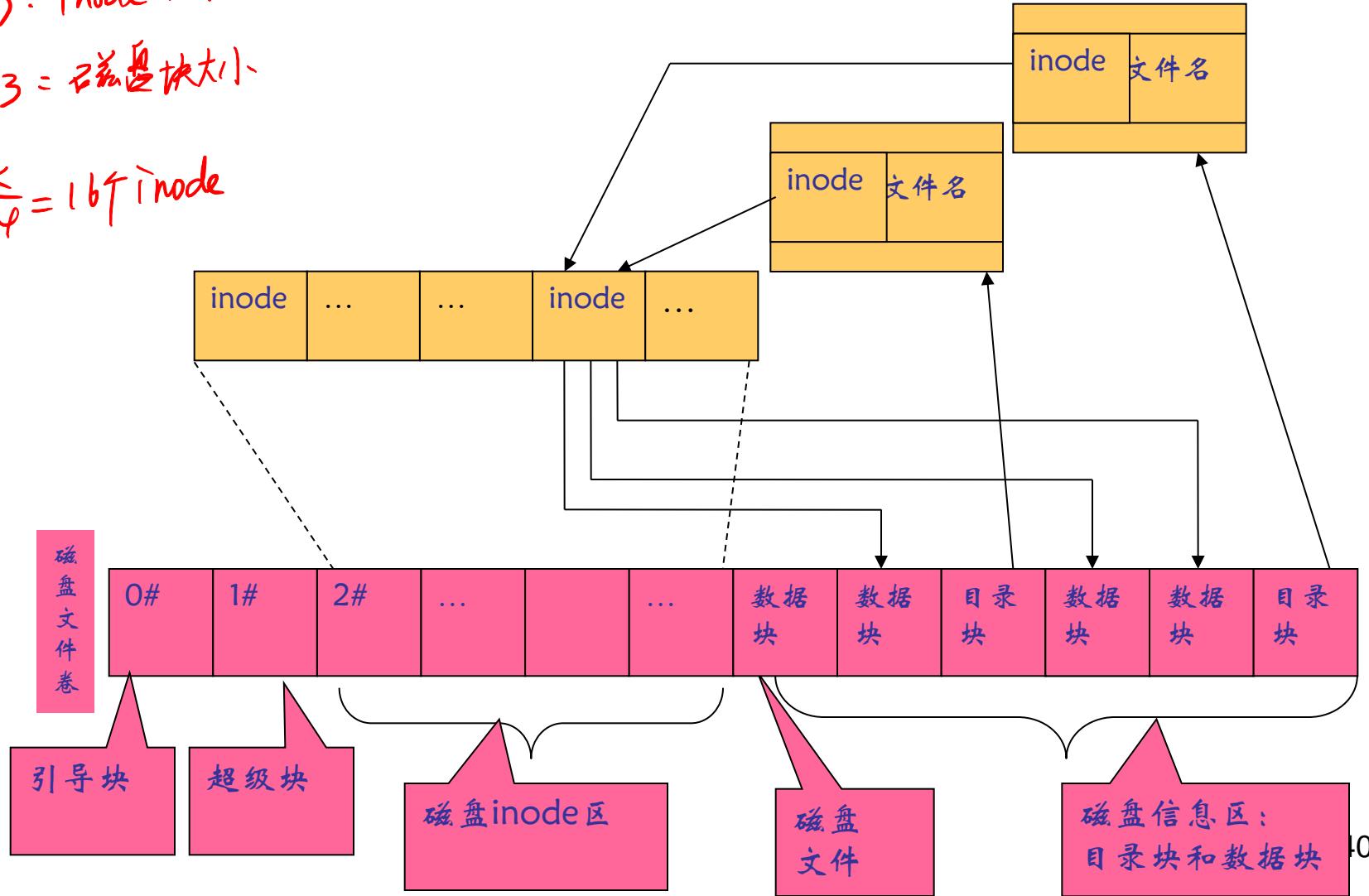


# 目录项、inode和数据块的关系

64B: inode 大小

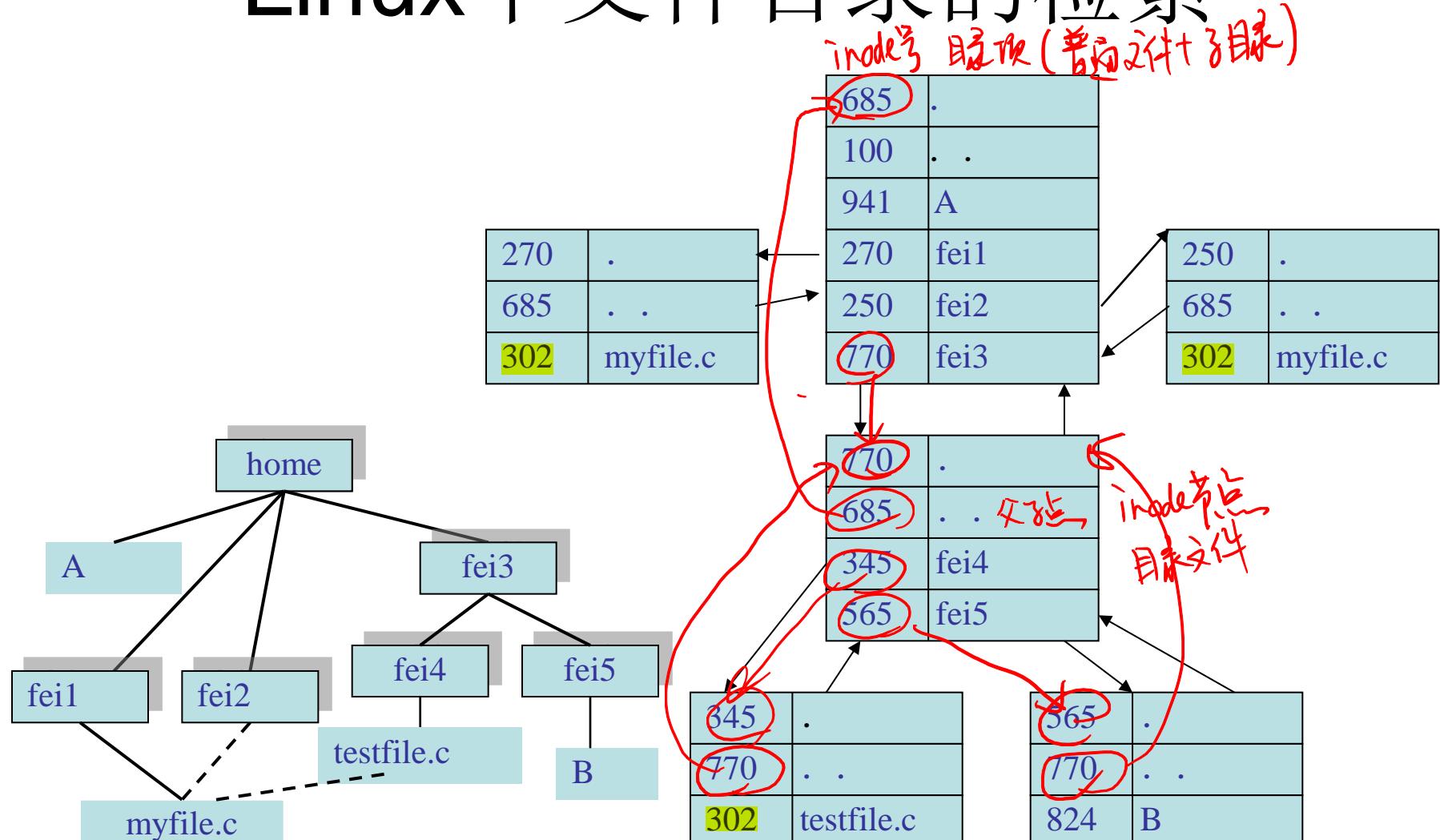
1KB = 磁盘块大小

$\frac{1K}{64} = 16$  个 inode



- 引导块
  - 位于文件卷最开始的第一扇区，该512字节是文件系统的引导代码，为根文件系统所特有，其他文件系统这512字节为空
- 超级块
  - 位于文件系统第二扇区，紧跟引导块之后，用于描述本文件系统的结构和管理信息。如inode节点所占盘块数、文件数据所占的盘块数等
- 磁盘inode区
  - 位于超级块之后，长度由超级块中的inode所占盘块数决定
  - 每个inode用于描述文件属性中除文件名之外的属性，包括文件的长度、属主、物理数据块号等；
- 数据块
  - 分为目录文件数据块和普通文件数据块
  - 目录文件数据块中存放的是目录项的集合
  - 普通文件数据块存放的是文件数据

# Linux中文件目录的检索



(a) 用户角度目录结构

(b) 系统角度目录链接

不同角度的目录结构

Root directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

inode 123  
found

Looking up  
usr yields  
i-node 6

i-node 6  
is for /usr

Mode	*
size	132
times	

自底向上  
Block 132  
is /usr  
directory

6	*
1	..
19	dick
30	erik
51	jim
26	ast
45	bal

i-node 26  
is for  
/usr/ast

26	*
6	..
64	grants
92	books
60	mbox
81	minix
17	src

Block 406  
is /usr/ast  
directory

26	*
6	..
64	grants
92	books
60	mbox
81	minix
17	src

i-node 6  
says that  
/usr is in  
block 132

/usr/ast  
is i-node  
26

i-node 26  
says that  
/usr/ast is in  
block 406

/usr/ast/mbox  
is i-node  
60

# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件组织与数据存储

- 逻辑结构
  - 从用户的观点出发，研究用户概念中的抽象的信息组织方式
  - 用户可以观察到的、可加以处理的数据集合
  - 相关数据的集合称为逻辑文件
- 物理结构
  - 逻辑文件在物理存储空间中的存放方法和组织关系
  - 物理文件被看成是相关物理块的集合
  - 主存与物理存储器进行信息交换的物理单位是块

# 文件的逻辑结构

## ① 流式文件

看做字节流

- 又称为无结构文件
- 文件内的数据不再分记录，而是看成字节流
- 也可以看作是记录就为1个字节的记录式文件
- 大多数现代操作系统如**WINDOWS, UNIX, LINUX**只提供流式文件
- 由应用程序自行根据字节重构对应用有意义的记录

# 文件的逻辑结构

## ②记录式文件 (不适用)

- 文件是一组记录的集合
  - 例如职工的工资记录构成工资文件
- 从操作系统的角度看, 逻辑记录是文件内独立的最小信息单位, 每次总是为使用者存储、检索或更新一条逻辑记录
- 记录式文件的记录组织和使用方法
  - 记录式顺序文件
    - 记录顺序编号, 并被顺序访问
  - 记录式索引文件 (key, value)
    - 用索引表根据记录键快速查找到相应的记录在文件中的位置

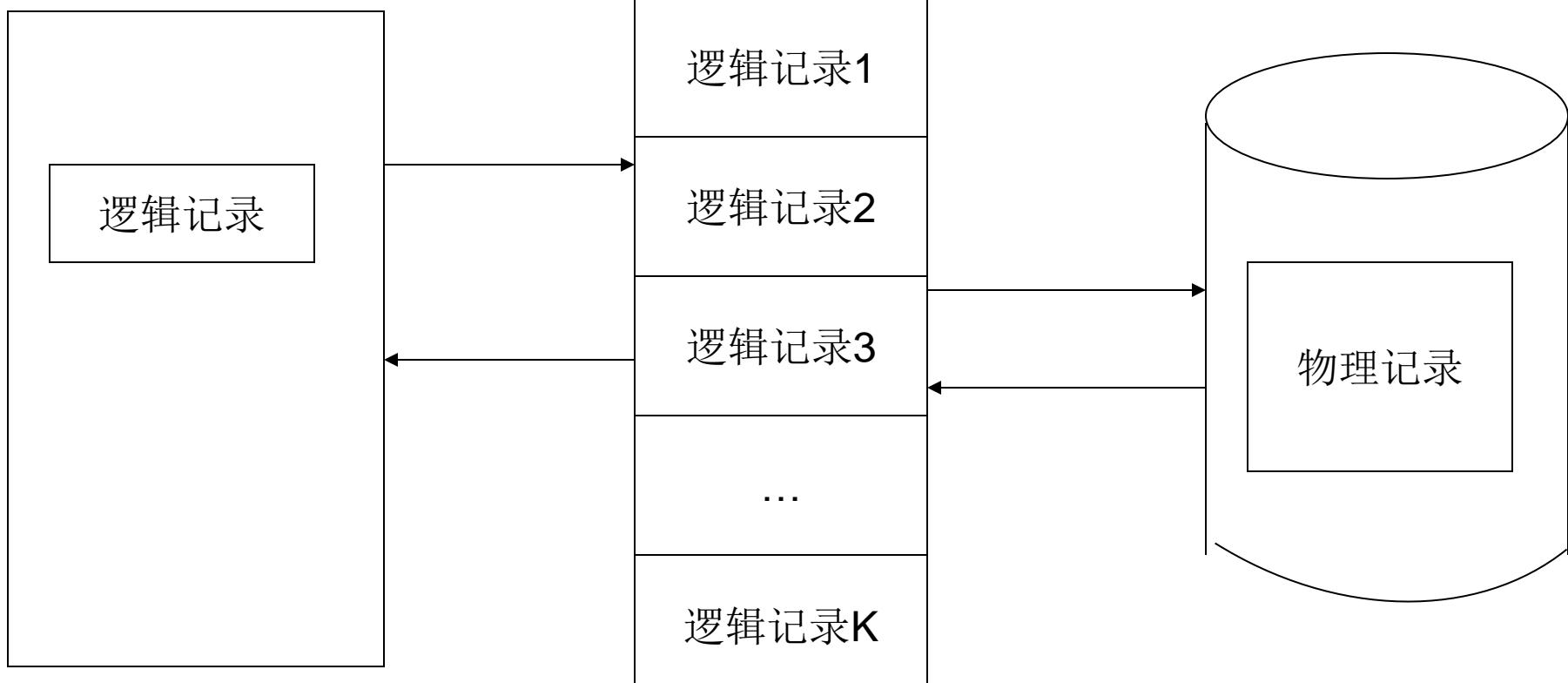
# 成组与分解

- 逻辑记录与物理块之间的对应关系
- 一条逻辑记录被存放到物理存储器时，可能会占用一块或多块，或者一个物理块可以包含多条逻辑记录
- 成组：若干逻辑记录合并成一组，写入一块，每块中的逻辑记录的个数称为成块因子
- 分解：从读进内存缓冲区的物理块中分解出逻辑记录的过程

用户工作区

系统缓冲区

物理存储区



记录成组与分解的处理过程

TLV

Length, Value

# 记录格式

- 记录格式就是记录内数据的排列方式
- 记录格式分为：
  - i) 定长记录
    - 所有逻辑记录具有相同的长度 如定位
    - 记录中的所有数据项的相对位置固定
  - ii) 变长记录
    - 逻辑记录长度不等
    - 每条逻辑记录的长度在处理之前能预先确定

# 记录键

- 能用于区别同一文件中其它逻辑记录的数据项，也称为关键字或键
- 能唯一标识某条逻辑记录的键称为主键

# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式 *流式*
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件的物理结构

- 文件系统需要提供将逻辑文件存储到物理存储设备上去的组织方式
  - 逻辑上连续的文件是否需要连续存放在物理存储介质上?
  - 文件的存取速度? 读写. 随机.
- 两类方法 逻辑块号  $\xrightarrow{\text{ }} \text{物理块号?}$ 
  - ① 计算法
    - 通过对记录键进行计算, 从而转换成对应的物理地址
  - ② 指针法
    - 设置专门的指针, 指明相应记录的物理地址或表达各个记录之间的关联

# 文件的物理结构

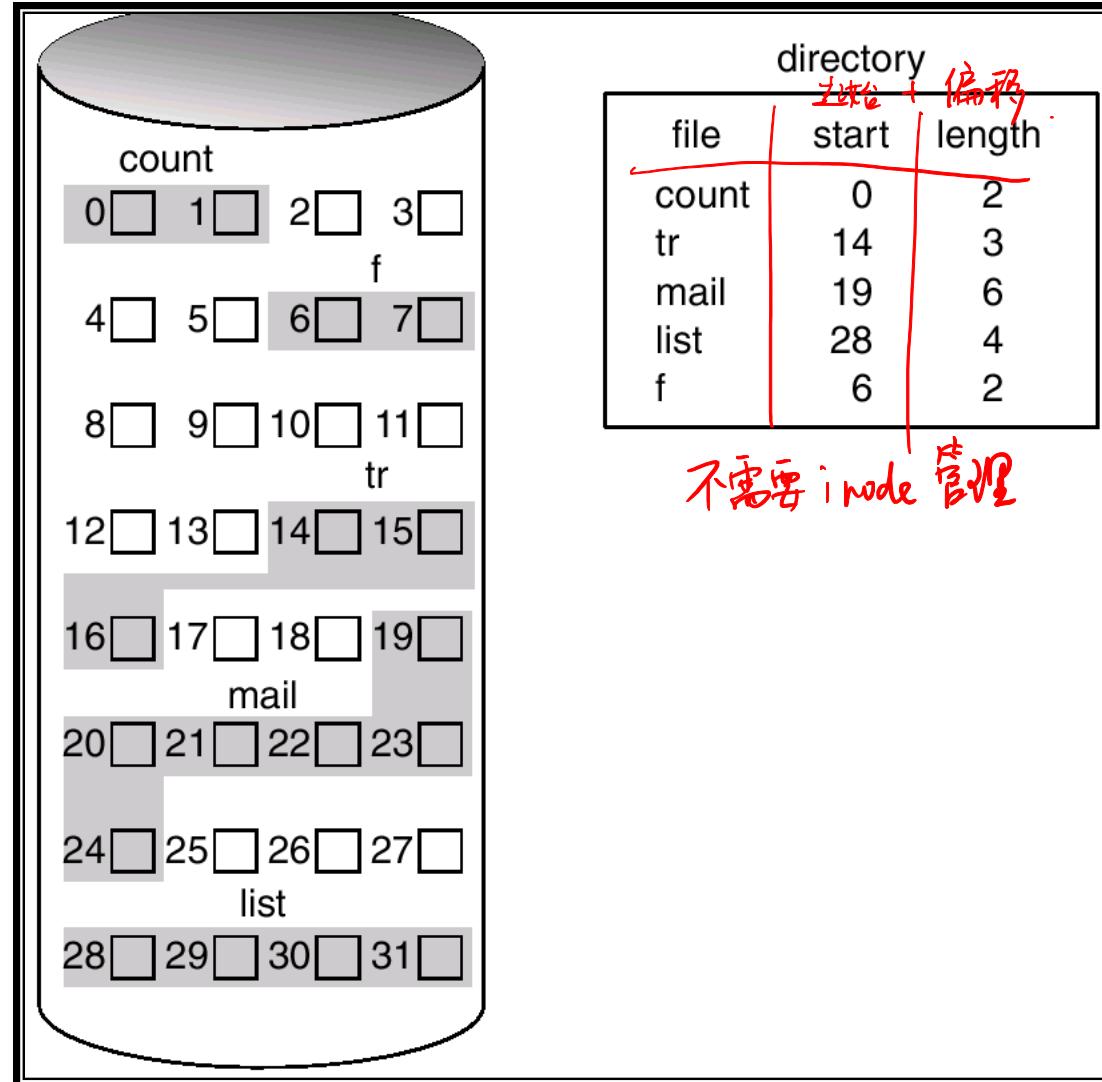
- 具体的文件物理结构和组织方法
  - ①连续文件
    - 类似数组
  - ②连接文件
    - 类似链表      *逐块*、  
*next → 下一个*
    - FAT: 连接文件的改进
  - ③索引文件
    - 类似Map

# 顺序文件/连续文件

31位，12位  
<柱面号, 磁头号, 扇区号>  
磁盘编号  
盘面

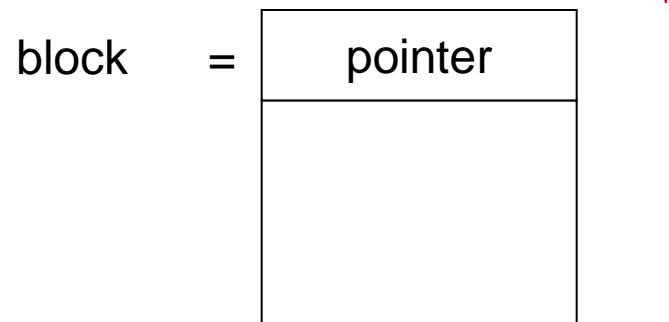
- 文件中逻辑上连续的信息存放到物理介质的相邻物理块上形成顺序结构
- 文件控制块FCB中保存第一个物理块的地址和占用的总物理块数  
*首字节*
- 可以对顺序文件按记录进行排序，成为有序的顺序文件
- 优点：
  - 顺序存取和随机存取时速度较快
- 缺点
  - 建立文件之前需要确定文件的长度，以分配存储空间
  - 修改、插入和添加文件记录较为困难
  - 对于变长记录的处理很困难  
*难扩充*
  - 会产生外部碎片

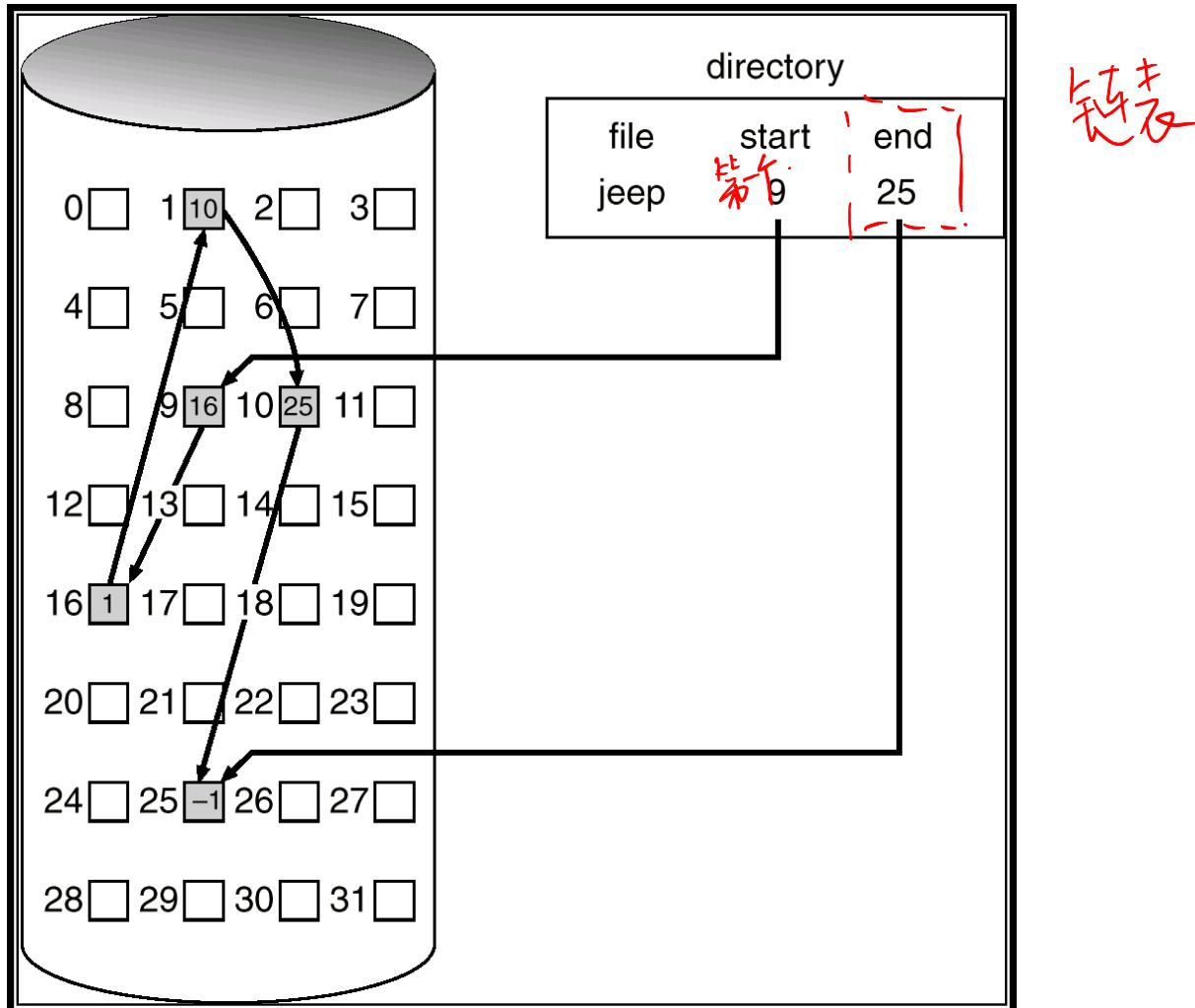
当数据块



# 连接文件

- 文件在物理上被组织成**物理块的链表**，分配给一个文件的物理块在空间上可能是分散的
- 文件控制块FCB给出**第一个**物理块的地址
- 每个块的**连接字**指出文件的下一个物理块位置
- 当连接字内容为某个特殊值时，表示文件至本块结束





$$\begin{aligned}
 \text{P}[rs[1]] &= 10 & [16] &= 1 & [25] &= -1 \\
 [9] &= 16 & [10] &= 25
 \end{aligned}$$

# 连接文件(cont)

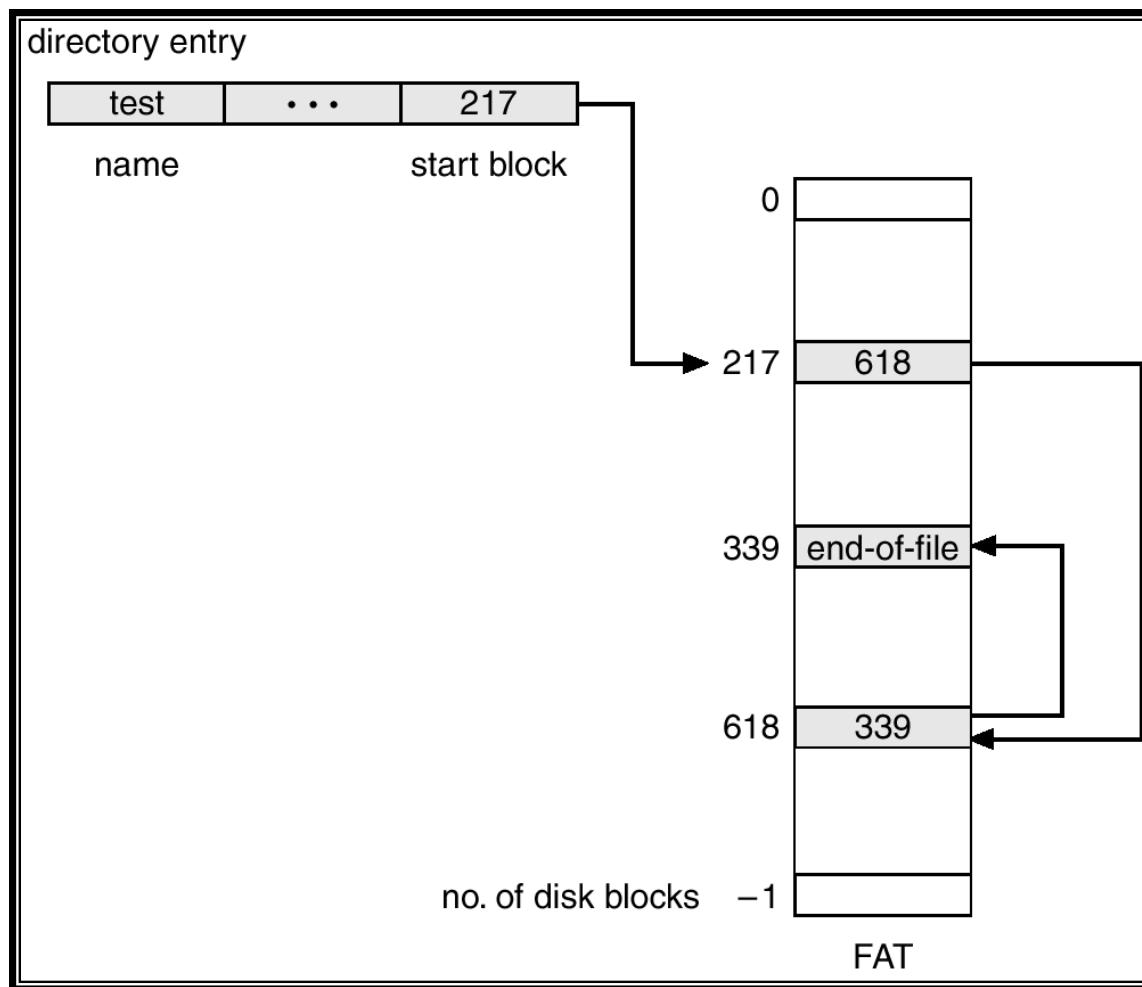
- 优点：
  - 文件的逻辑顺序独立于存储空间的物理块顺序
  - 易于记录的增、删、改
  - 易于文件扩充
- 缺点
  - 仅适用于顺序存取
  - 随机存取速度慢，需要从头开始查找
  - 连接字与数据混放，破坏了数据块的完整性
  - 连接字需要占用额外的空间

# FAT

- 连接文件的一个变种，用于克服连接字和数据混放的缺点，提高随机存取的速度
- 把连接指针从数据块中分离出来，单独建立一个指针数组 PTRS[n], n为组成磁盘连接文件物理块的总块数
- 每个PTRS[i]对应于一个物理块i, 如果物理块j在文件中紧跟物理块i之后，则PTRS[i]=j
- 文件控制块FCB存放文件第一个物理块的块号
- 指针数组保存在磁盘的一个专门区域，如0磁道的前k个块
- 为了缩短定位文件信息块所需要的时间，可以把这些指针连续装进主存或高速缓存。在内存里，找到磁盘块号后，直接读取
- 假设4<sup>2</sup>字节记录磁盘块号，块大小为1K<sub>2</sub><sup>10</sup>则当k=100时，可以记录 $256 \times 100 = 25600$ 个数据块的磁盘块号，即25<sub>16</sub><sup>10</sup>不需要600, 000字节大小的存储空间

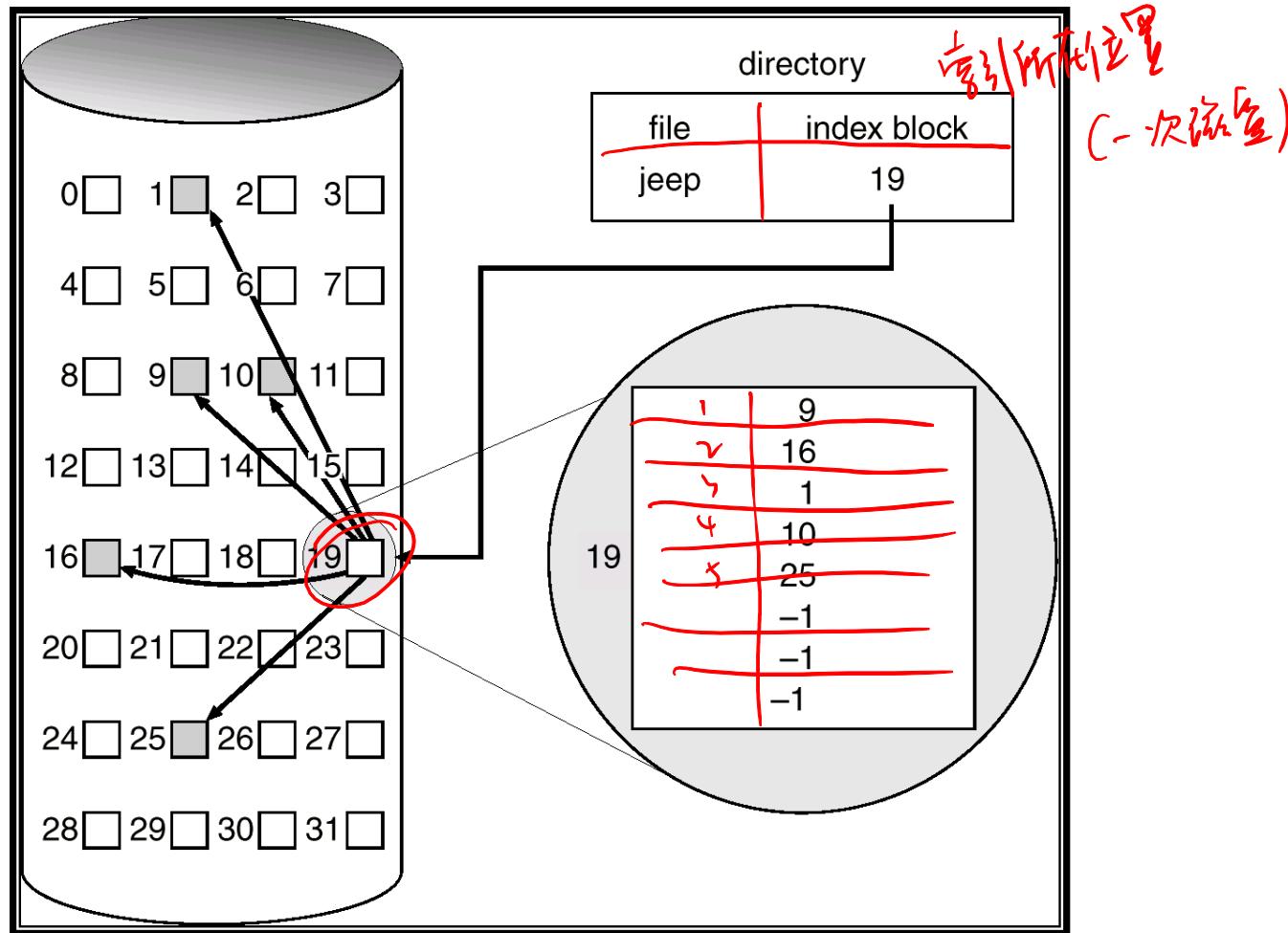
$$2^{10} - 2^2 = 2^8$$

1K



# 索引文件

- 为每个文件建立索引，给出逻辑记录或逻辑块号到物理块号的映射
- 索引表可以存放在文件控制块中，也可以让索引表作为物理块单独驻留在磁盘的任意位置，而**FCB**中仅包含索引表的地址
- 优点
  - 随机存储速度快
  - 便于信息的增、删、改
- 缺点
  - 索引表的空间开销



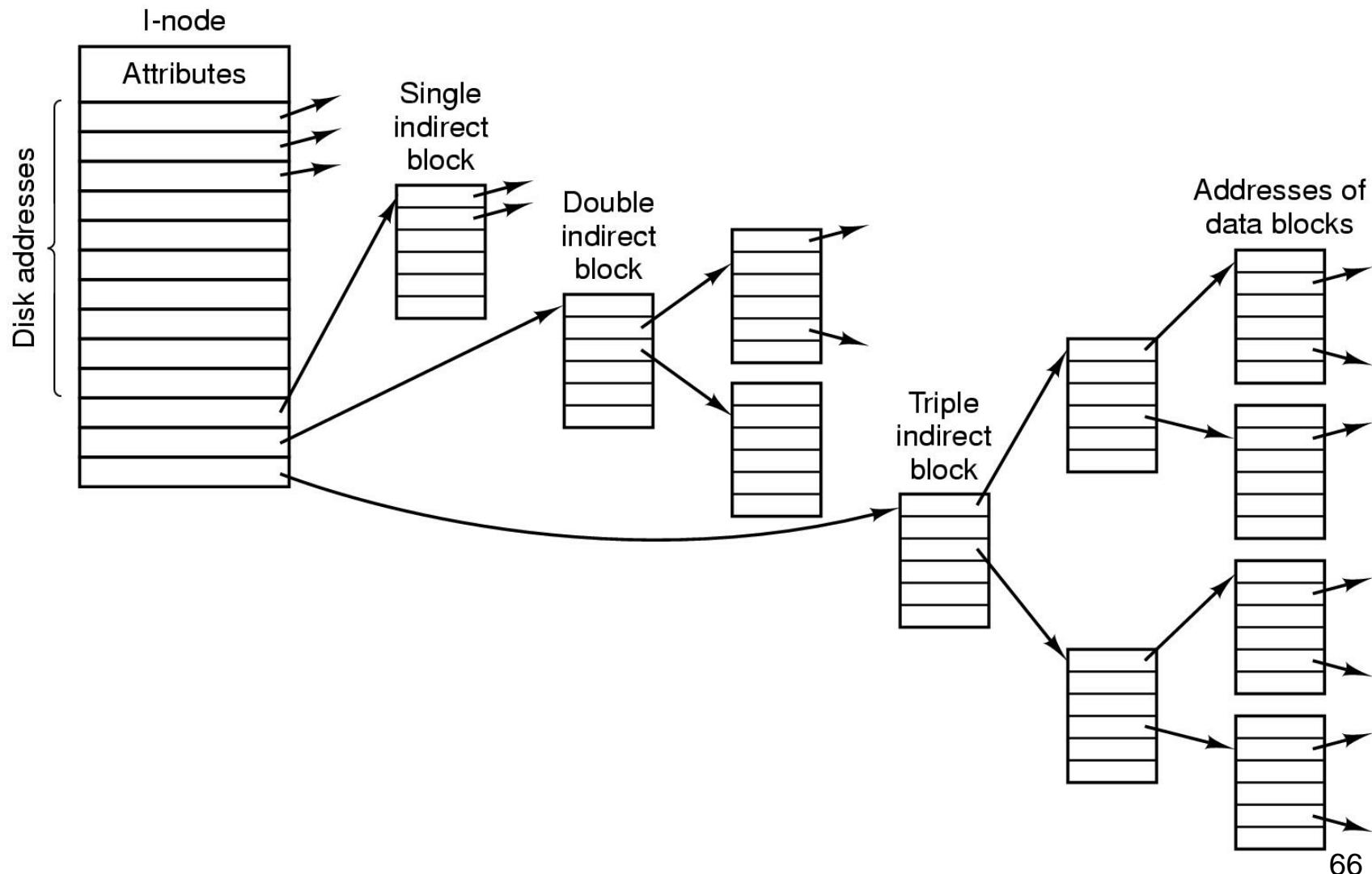
# 索引文件(cont)

- 当记录数目很多/文件很大时，索引表本身要占用很多物理块，则查找某个记录键所对应的索引项时，可能需要查找多个物理块
  - 若索引表占用 $n$ 块，则平均要载入 $n/2$ 个物理块，才能找到所需记录的物理地址
- 可以建立索引的索引，称为二级索引；或索引的索引的索引，即三级索引

# UNIX/LINUX中的多级索引结构

- 每个FCB中规定了13个索引项
  - 前10项为直接索引
  - 11项为一次间接寻址
  - 12项为二次间接寻址
  - 13项为三次间接寻址
- 小文件无需二次索引，超大文件可以用3级索引，文件最大可以到11亿字节

# UNIX/LINUX中的多重索引结构



# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件空间管理的作用

- 辅助存储空间的有效分配和释放
  - 创建和扩充文件时，**决定分配哪些磁盘块**是很重要的，这会影响磁盘访问次数
  - 删除文件和缩短文件时，需要**回收磁盘块**
  - 随着分配和回收，**可能会出现碎片**

# 常用磁盘空间管理方法

- 位示图
- 空闲区表
- 空闲块链
- 成组空闲块链

# 位示图法

- 磁盘空间由固定大小的块组成，可方便地使用位示图管理。一个bit
- 每一字位对应于一个物理块，字位值为1表示被占用，0表示空闲。
- 优点：
  - 每个盘块仅需一比特来标识
  - 若盘块长为1KB，则位示图开销为0.012%

# 空闲区表法

- 常常用于连续文件，将空闲区存储块的位置及其连续空闲的块数构成一张表
- 分配时，依次扫描空闲区表，寻找合适的空闲块并修改登记项；
- 删除文件并释放空闲区时，把空闲位置及连续空闲区长度填入空闲区表，出现邻接的空闲区时，还需执行合并操作
- 搜索算法：
  - 最先适应
  - 最佳适应
  - 最坏适应等

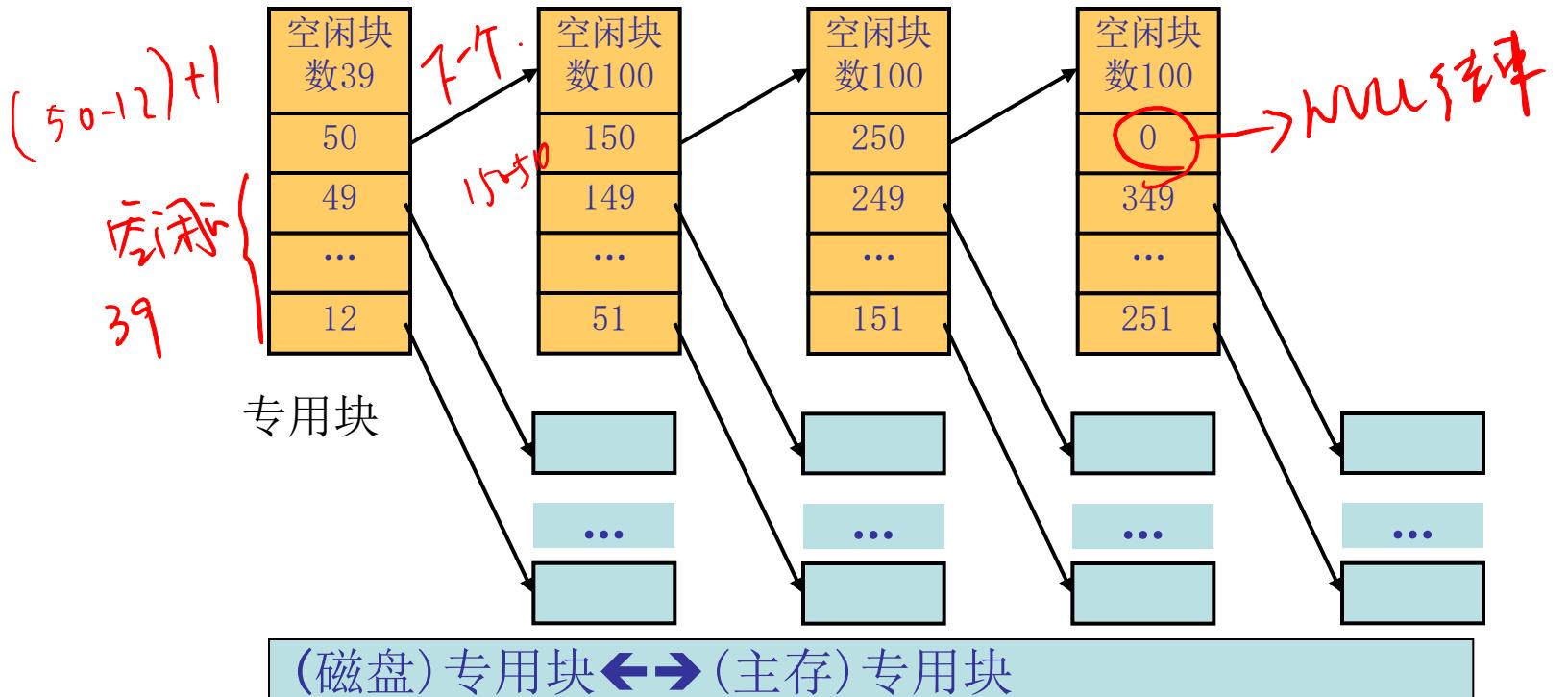
# 空闲块链

- 把所有空闲块连接在一起，系统保持指针指向第一个空闲块，每一空闲块中包含指向下一个空闲块的指针
- 申请一个空闲块时，从链头取一块并修改系统指针；
- 删除时释放占用块，使其成为空闲块并挂到空闲链上

# UNIX/Linux空闲块成组连接法(1)

- 每100块划分一组，每组第一块登记下  
一组空闲块的物理盘块号和空闲块总数。
- 余下不足100块的那部分空闲块的块号及块数登记在一个专用块中。
- 若一个组的第一个空闲块号为0，则表示该组为最后一组

# UNIX/Linux空闲块成组连接法(2)



(磁盘)专用块  $\leftrightarrow$  (主存)专用块

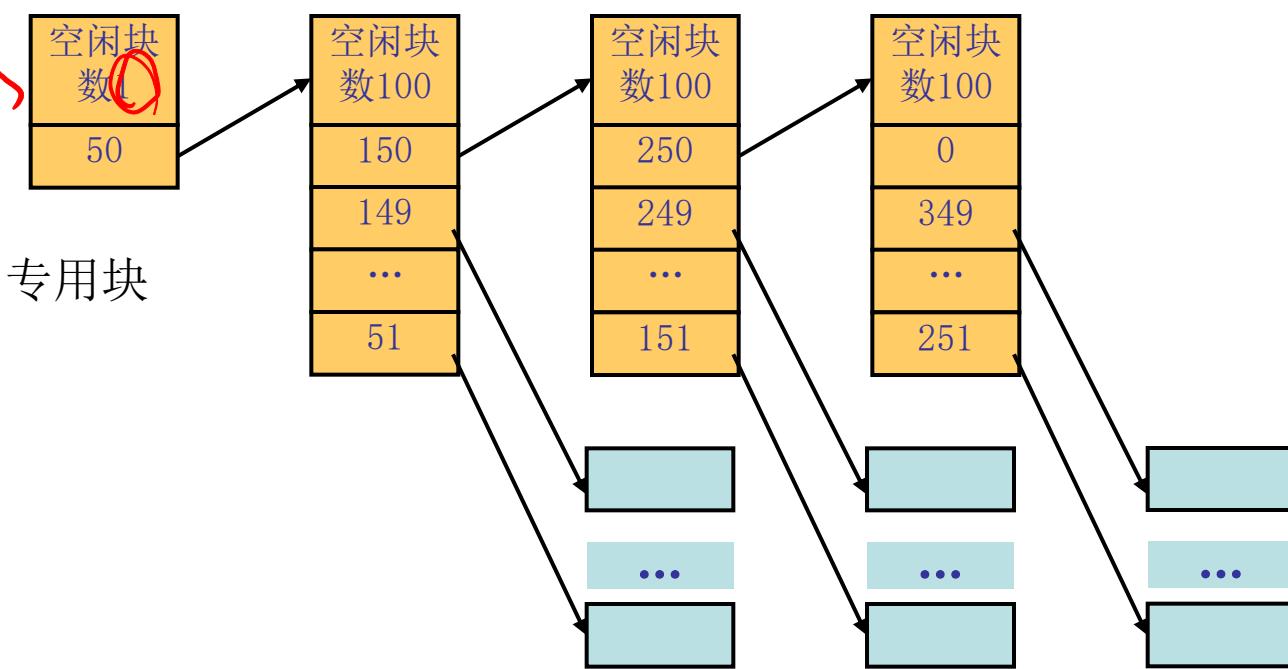
分配算法

IF 空闲块数=1 THEN  
  IF 第一个单元=0 THEN 等待  
    ELSE 复制第一个单元对应块到专用块，并分配之  
  ELSE 分配第(空闲块数)个单元对应块，空闲块数减1

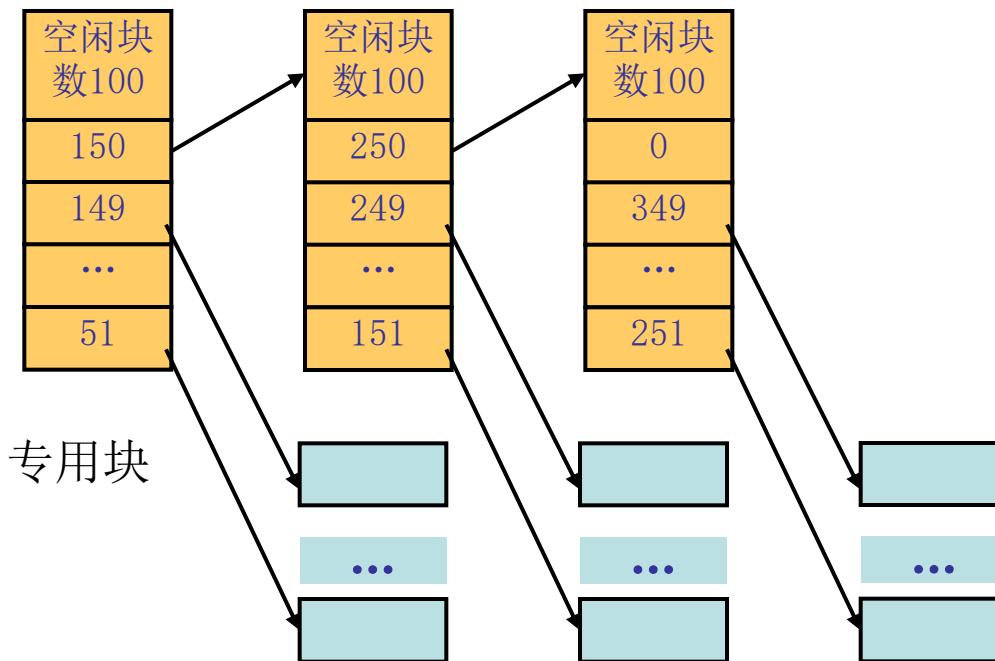
归还算法

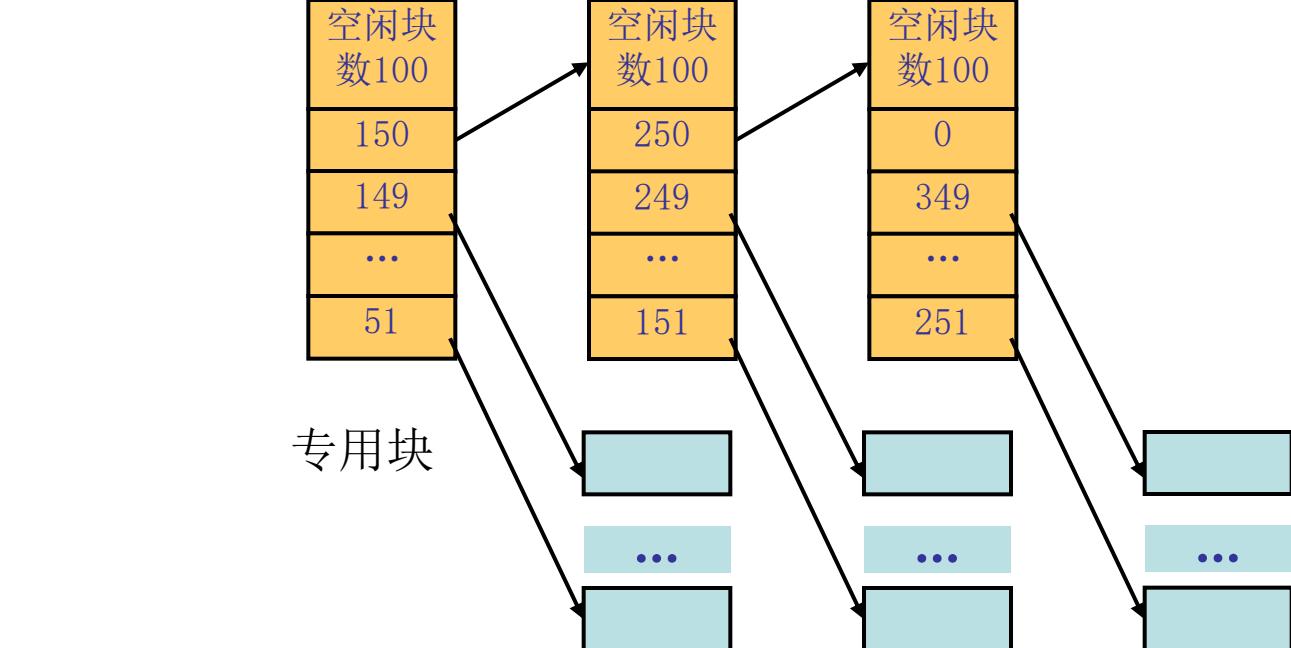
IF 空闲块数<100 直接归还  
THEN 专用块的空闲块数加一，第(空闲块数)个单元置归还块号  
ELSE 复制专用块到归还块，专用块的空闲块数置一，第一单元置归还块号

下个页

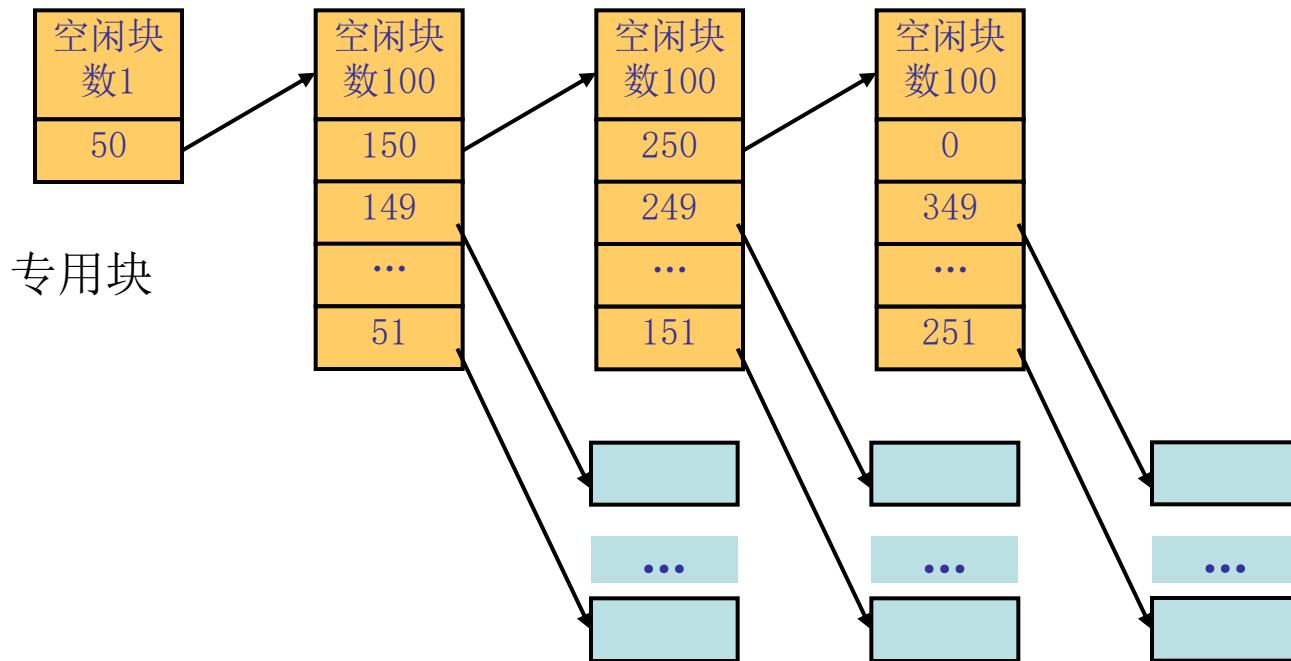


分配第50块





归还第50块



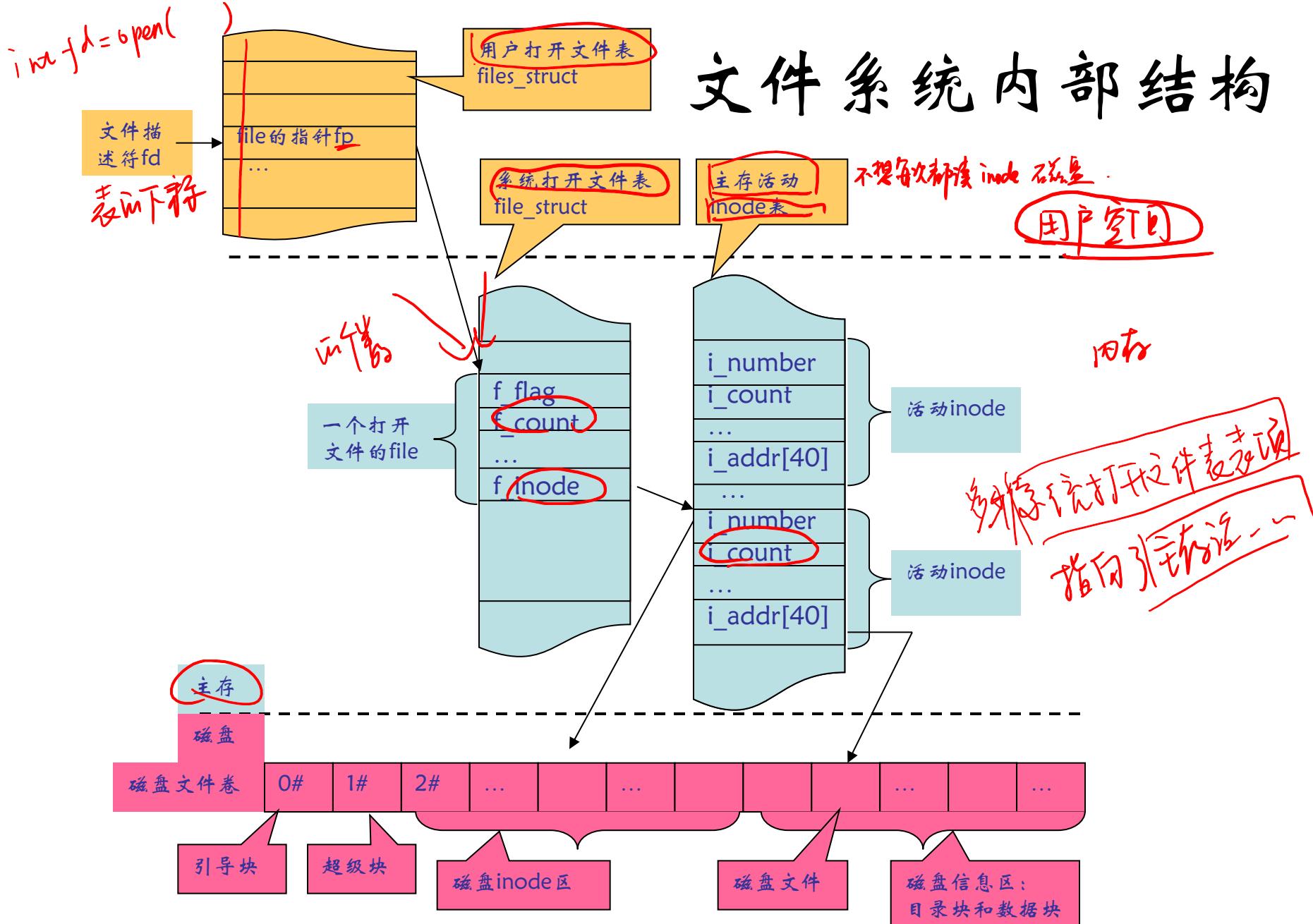
# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件系统调用的实现

- 创建和删除文件
- 打开和关闭文件
- 读/写文件
- 文件共享
- 主存映射文件

# 文件系统内部结构



# 主存活动inode表

- 磁盘inode记录文件的属性和相关信息，文件访问过程会频繁地用到它，因此频繁地访问辅存极不经济
- Linux/Unix在内核开辟一张专用表，用于存储经常访问的磁盘inode，以及一些动态信息，称为主存inode表/活跃inode表

# 主存活动inode表

- 主存inode表包含：
  - 磁盘inode的所有信息，如物理块地址
  - 主存inode的状态
    - Inode是否上锁
    - Inode数据是否改变
    - 文件数据是否改变
  - 文件系统的逻辑设备号.
  - inode号
  - 引用数:i\_count
    - 多少个系统打开文件表项指向该主存inode

# 系统打开文件表

- 内核有一个系统打开文件表，用于存放文件访问信息
- 系统打开文件表中的每一个表项包括：
  - 指向主存inode表的指针  $t\_inode$
  - 下一次读/写文件的位移
  - 打开进程的访问权限
  - 引用计数  $f\_count$ 
    - 多少个用户打开文件表项指向该系统打开文件表项

# 用户打开文件表

- 每个进程有一个用户文件描述符表, 用于描述所有打开的文件, 称为用户打开文件表
- 每个用户打开文件表中的表项都指向一个内核系统打开文件表中的表项
  - Entry 0: 标准输入
  - Entry 1: 标准输出
  - Entry 2: 标准错误输出

# 文件系统调用

--文件的创建

系统调用C语言格式为：

```
int fd, mode;
```

```
char *filenamep;
```

```
fd = create (filenamep, mode);
```

创建兼有打开功能

# 文件系统调用

## --文件的创建

- ① 为新文件分配索引节点和活动索引节点，并把索引节点编号与文件分量名组成新目录项，记到目录中。

1513 | a.txt
- ② 在新文件所对应的活动索引节点中置初值，如置存取权限i\_mode，连接计数i\_nlink等。
- ③ 分配用户打开文件表项和系统打开文件表项，置系统打开文件表项初值，如读写位移f\_offset清“0”。
- ④ 把各表项及文件对应的活动索引节点用指针连接起来，把文件描述字fd返回给调用者。

# 文件系统调用

## --文件的删除

- 删除的任务：
  - 把指定文件从所在的目录文件中去除。  
    - 如果没有连接用户(i\_nlink 为“1” )，还要把文  
        件占用的存储空间释放。     $\rightarrow$  | 别的目录文件也指向它.
- 系统调用形式为： unlink (filenamep)。解除连接
- 在执行删除时，必须要求用户对该文件具有  
“写”操作权。

# 文件系统调用

## --文件的打开

调用方式为：

```
int fd, mode;  
char * filenamep;  
fd = open (filenamep, mode);
```

# 文件系统调用

## --文件的打开

- ① 检索目录，把它的外存索引节点复制到活动索引节点表。  
    外存索引节点  $\triangle\triangle$  复制到 活动索引节点表  $\triangle\triangle$
- ② 根据参数mode核对权限，如果非法，则这次打开失败。
- ③ 当“打开”合法时，为文件分配用户打开文件表项和系统打开文件表项，并为系统打开文件表的表项设置初值。通过指针建立这些表项与活动索引节点间的联系。把文件描述字，即用户打开文件表中相应文件表项的序号返回给调用者。

# 文件系统调用

## --文件的打开

- 如果执行open()时，其它用户已经打开了同一文件，则活动inode表中已经有此文件的inode，则无需执行第①步的inode复制工作
- 但需要将活动inode中的*i\_count*加1
  - *i\_count*反映通过不同的系统打开文件表项来共享同一个活动inode的进程数目
  - 是执行文件关闭操作时，能否释放活动inode节点的依据。

```
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>

main()
{
    int fd1, fd2, fd3;
    printf("Before open ...\n");
    fd1 = open( "/etc/passwd", O_RDONLY);
    fd2 = open("./openEx1.c", O_WRONLY);
    fd3 = open( "/etc/passwd", O_RDONLY);
    printf("fd1=%d fd2=%d fd3=%d \n", fd1, fd2, fd3);
}
```

```
$ cc openEx1.c -o openEx1
$ openEx1
Before open ...
fd1=3 fd2=4 fd3=5
$
```

0 1 2  
打开 打开 错误  
写入

同文件

## 用户区



## 用户打开文件表

0
1
2
3
4
5
6
7
⋮
⋮
⋮

## 系统打开文件表

...
CNT=1 R
...
CNT=1 W
...
CNT=1 R
...

不同

+从先读5B  
+后读5B >读出结果一致

## 主存活动inode表

指向同一个inode文件
CNT=2
/etc/passwd
...
CNT=1
/openEx2.c
...

# 文件系统调用

## --文件的关闭

调用方式为：

```
int fd;  
close (fd);
```

# 文件系统调用

## --文件的关闭

用户  
系统  $f\_count = 0$   
 $\Rightarrow$  inode  $i\_count = 0$

- ~~① 根据fd找到用户打开文件表项，再找到系统  
打开文件表项，释放用户打开文件表项。~~
- ~~② 把对应系统打开文件表项中的 $f\_count$ 减1，  
如果非0，说明进程族中还有其它进程共享这一表项，不用释放直接返回；否则释放表项，  
并找到与之连接的主动活动inode。~~
- ~~③ 把活动inode中的 $i\_count$ 减1，若不为0，表明还有用户进程正在使用该文件，不用释放  
而直接返回；否则在把该活动inode中的内容  
复制回磁盘上的相应inode后，释放该活动  
inode。~~

dirty  
放过

# 文件系统调用

- `f_count`和`i_count`反映了进程动态共享一个文件的两种方式
  - 系统打开文件表中的`f_count`反映不同进程通过同一个系统打开文件表项共享一个文件的情况
    - 进程使用相同的位移指针`f_offset`共享文件
  - 主存活动inode中的`i_count`反映不同进程通过不同系统打开文件表项共享一个文件的情况
    - 进程使用不同的位移指针`f_offset`共享文件

# 文件系统调用

--读文件

调用的形式为：

```
int nr, fd, count;
```

```
char buf [];
```

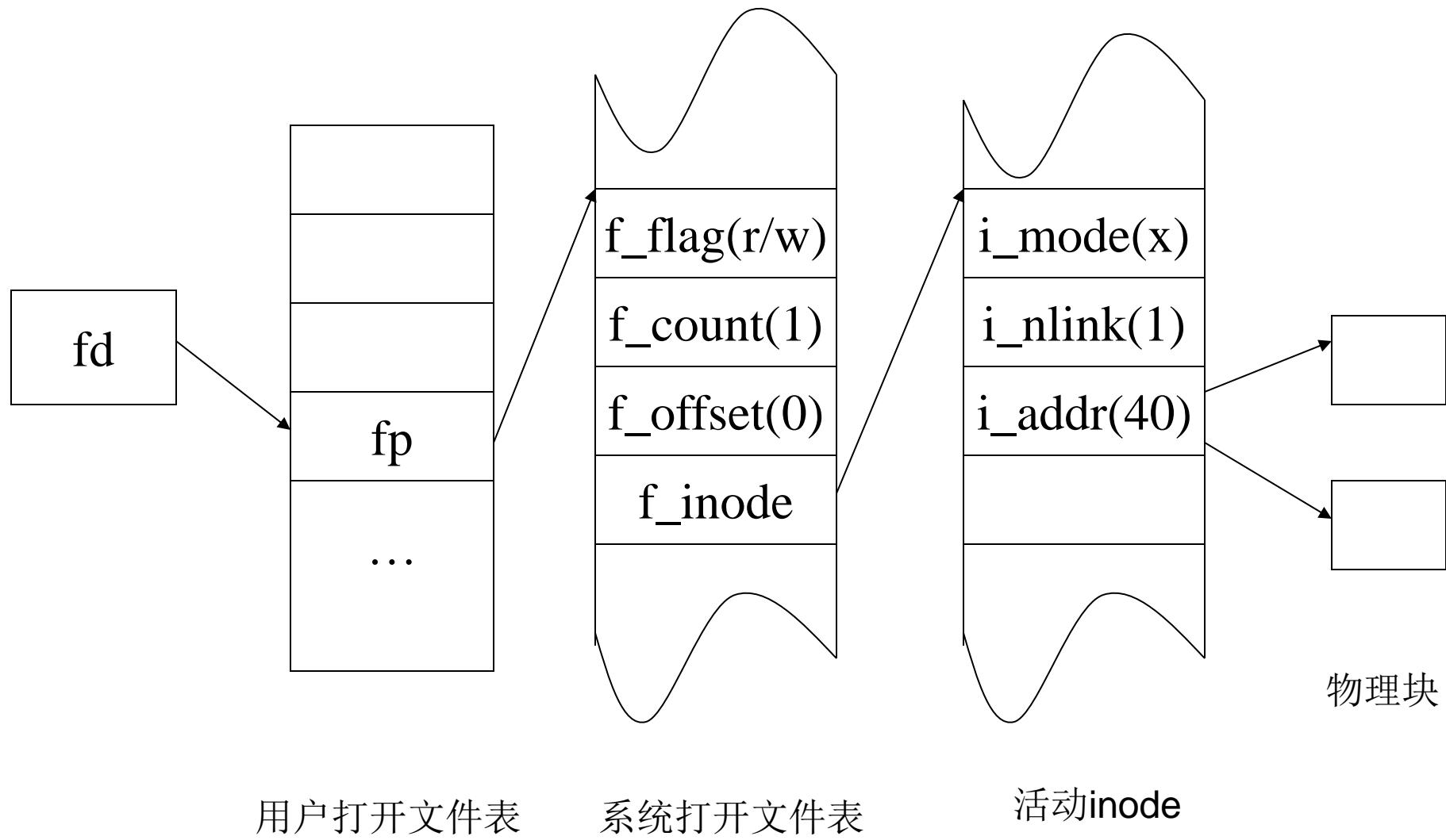
```
nr = read (fd, buf, count);
```

# 文件系统调用

## --读文件

- 系统根据f\_flag中的信息，检查读操作合法性；
- 若合法，按活动i\_node中i\_addr指出的文件物理块存放地址，从文件当前的位移量f\_offset处开始，读出所要求的count个字节，存放到系统缓冲区中，然后再送到buf指向的用户主存区中。

$$f\_offset \sim + count .$$



```
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>

main()
{
    int fd1, fd2, fd3;
    char buf1[20], buf2[20];
    buf1[19]='\0';
    buf2[19]='\0';
    printf("=====\\n");
    fd1 = open("/etc/passwd", O_RDONLY); //一个文件
    read(fd1, buf1, 19); //从文件读取19个字节
    printf("fd1=%d buf1=%s \\n",fd1,buf1);
    read(fd1, buf2, 19);
    printf("fd1=%d buf2=%s \\n",fd1,buf2);
    printf("=====\\n");
}
```

```
$ cc openEx2.c -o openEx2
$ openEx2
=====
fd1=3 buf1=root:x:0:1:Super-Us
fd1=3 buf2=er:/sbin/sh
daemo
=====
$
```

```

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
main()
{
    int fd1, fd2, fd3;
    char buf1[20], buf2[20];
    buf1[19]='\0';
    buf2[19]='\0';
    printf("=====\\n");
    fd1 = open("/etc/passwd", O_RDONLY);
    fd2 = open("/etc/passwd", O_RDONLY);
    read(fd1, buf1, 19);
    printf("fd1=%d  buf1=%s \\n",fd1, buf1);
    read(fd2, buf2, 19);
    printf("fd2=%d  buf2=%s \\n",fd2, buf2);
    printf("=====\\n");
}

```

```

$ cc openEx3.c -o openEx3
$ openEx3
=====
fd1=3  buf1=root:x:0:1:Super-Us
fd2=4  buf2=root:x:0:1:Super-Us
=====
$ 

```

两次系统调用，创建了两个系统打开文件表项，每个都有自己  
的读写位移指针

*2 ffd\_offset*

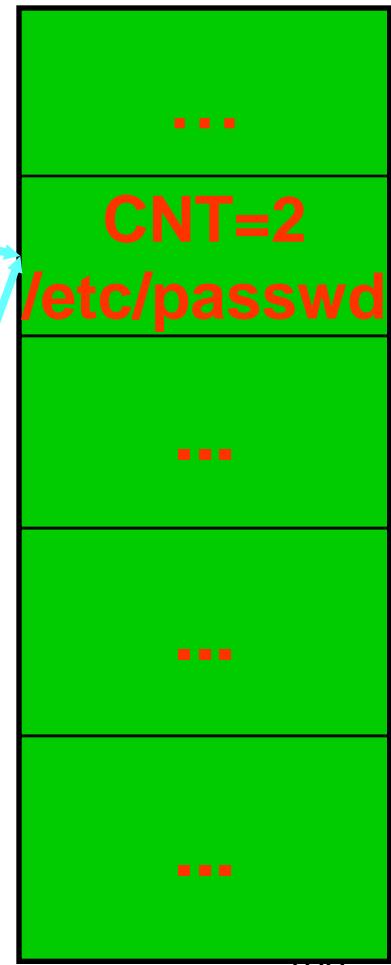
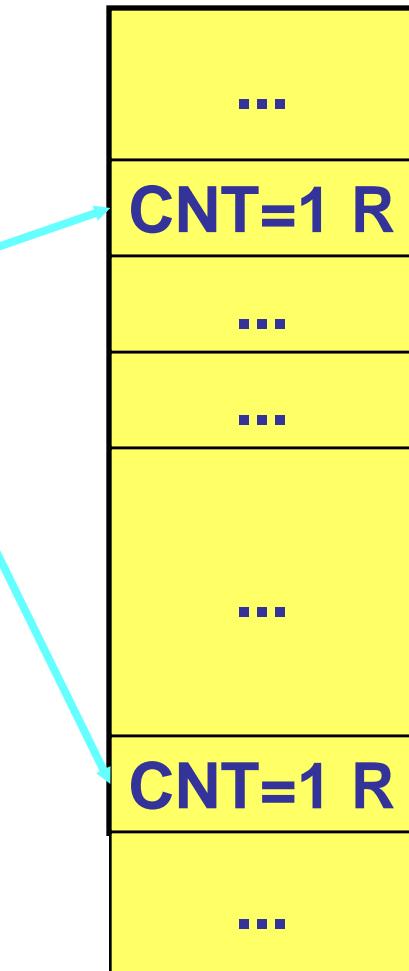
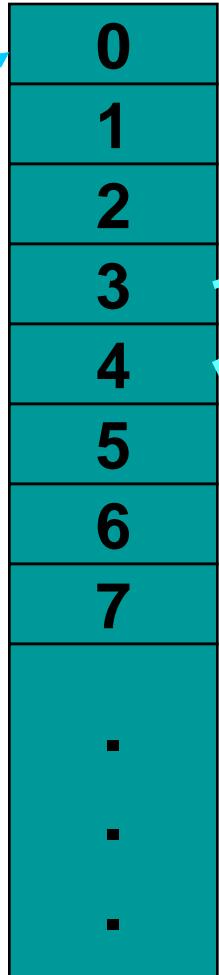
用户区

用户打开  
文件表指针

用户打开文件表

系统打开文件表

主存活动  
**inode**表



# 文件系统调用

## --写文件

<sup>实际写入</sup>  
<sup>参数</sup> 调用的形式为：

`nw = write (fd, buf, count);`

buf是信息传送的源地址，即把buf所指向的用户主存区中的信息，写入到文件中。

# 文件系统调用

## --文件的随机存取

系统调用的形式为：

```
long offset;
```

```
int whence, fd;
```

```
long lseek (fd, offset, whence);
```

whence

绝对位移  
或 相对位移

# 文件系统调用

## --文件的随机存取

- fd:
  - 文件描述字fd必须指向一个用读或写方式打开的文件
- offset:
  - 当whence是0时，则f\_offset被置为offset，即设置**绝对位移**
  - 当whence是1时，则f\_offset被置为文件当前位置加上offset，即设置**相对位移**

负数向后

正数向前

# 大纲

- 文件与文件系统
- 文件目录的组织方式
- 文件的逻辑组织方式
- 文件的物理组织方式
- 文件空间管理方法
- 文件系统调用的实现
- 文件共享实现

# 文件共享

- 文件的静态共享
- 文件的动态共享
- 文件的符号链接共享

# 文件的静态共享

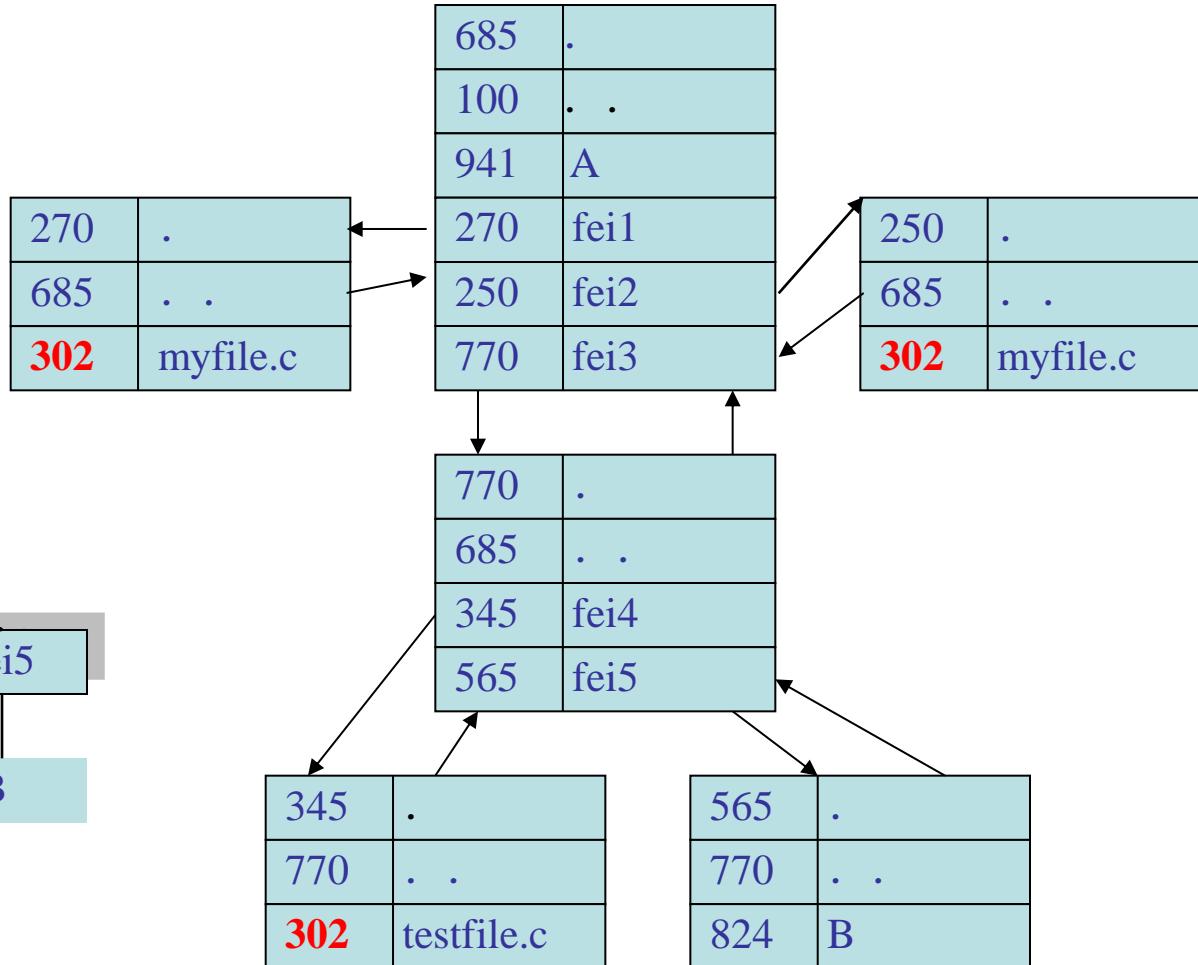
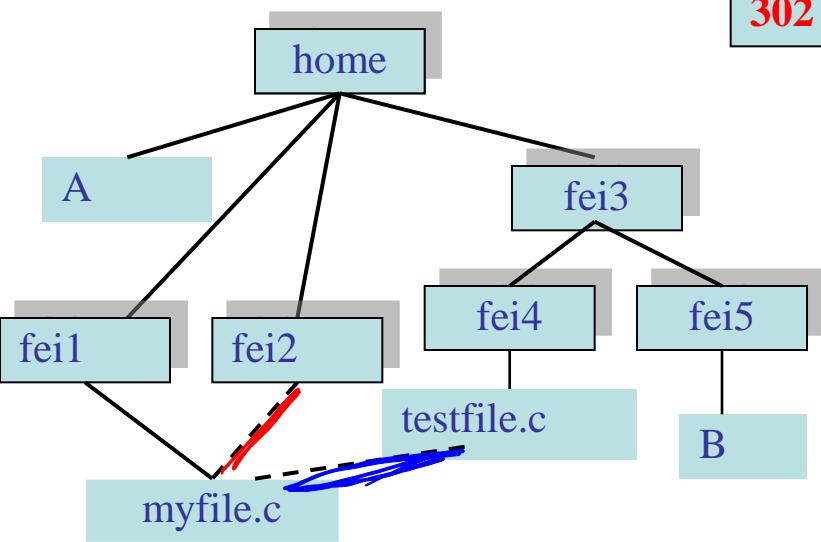
- 一个文件同时属于多个目录，但实际上文件仅有一处物理存储
  - 文件链接 破链接
- 优点：
  - 节省空间
  - 文件的修改对任何用户都可见，保证文件的一致性

# 文件的静态共享

系统调用形式为：

```
char * oldnamep, * newnamep;  
link (oldnamep, newnamep);
```

- ① 检索目录找到oldnamep所指向文件的索引节点  
i\_node编号。
- ② 再次检索目录找到newnamep所指文件的父目录文件，并把已存在文件的索引节点*i\_node*编号与  
别名构成一个目录项，记入到该目录中去。
- ③ 把已存在文件索引节点*i\_node*的连接计数  
i\_nlink加“1”。  
    ++



```
link("/home/fei1/myfile.c", "/home/fei2/myfile.c");
```

```
link("/home/fei1/myfile.c", "/home/fei3/fei4/testfile.c");
```

# 文件的静态共享

- 文件解除链接调用形式为:  
`unlink (namep)`
- 解除链接与文件删除执行的是同一系统调用代码。
- 删除文件是从文件主角度讲的，解除文件连接是从共享文件的其他用户角度讲的。
- 都要删去目录项，把`i_nlink`减1，不过，只有当`i_nlink`减为0时，才真正删除文件。

# 文件的动态共享

- 文件动态共享是系统中不同的用户进程或同一用户的不同进程并发访问同一文件。
- 这种共享关系只有当用户进程存在时才可能出现，一旦用户的进程消亡，其共享关系也就自动消失。
- 文件的每次读写由一个读/写位移指针指出要读写的位置。现在的问题是：**应让多个进程共用同一个读/写位移，还是各个进程具有各自的读写位移呢？**

独立读写 — 各自上各自下

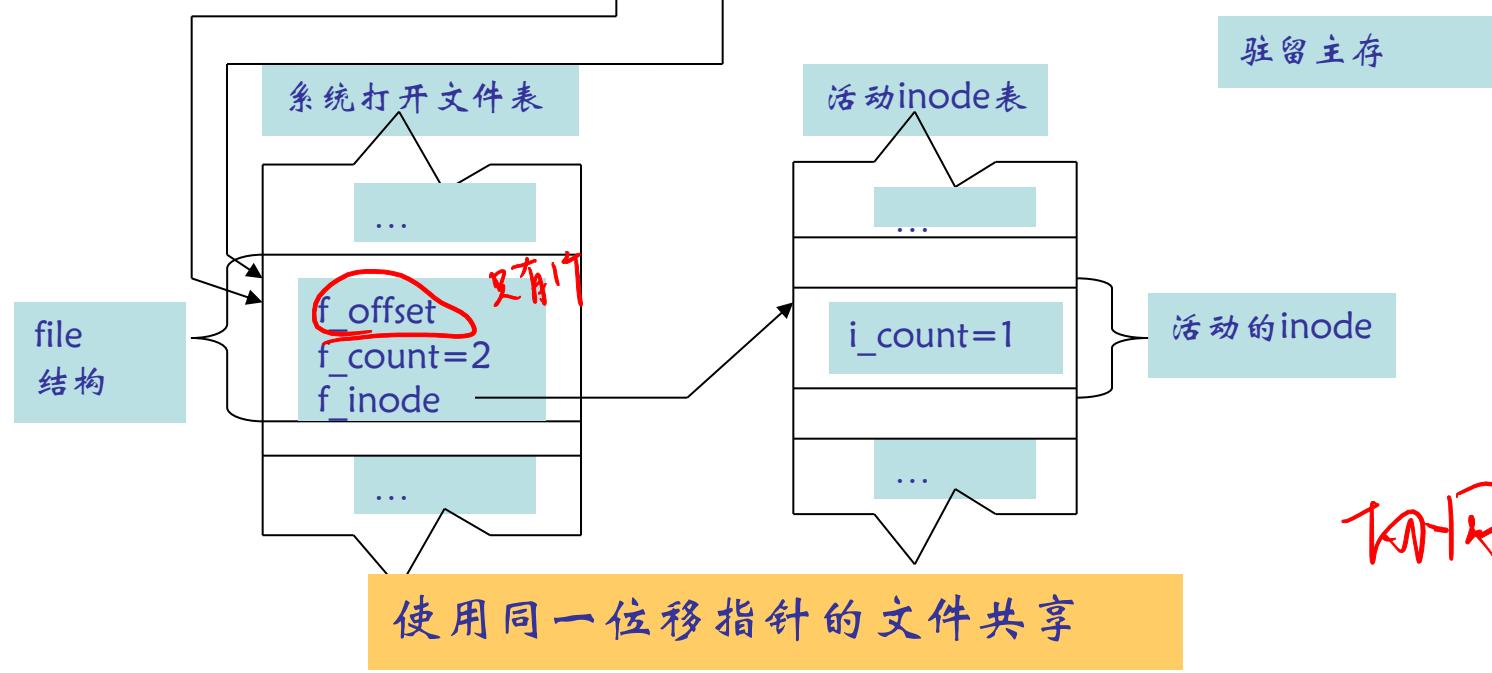
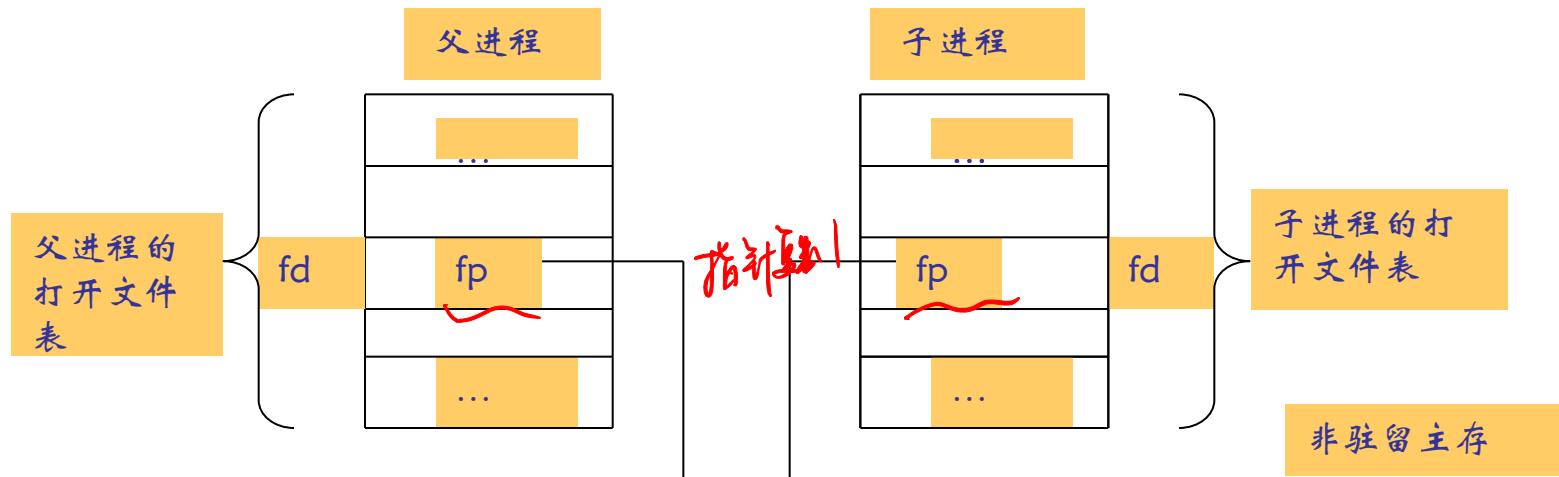
协同

# 文件的动态共享

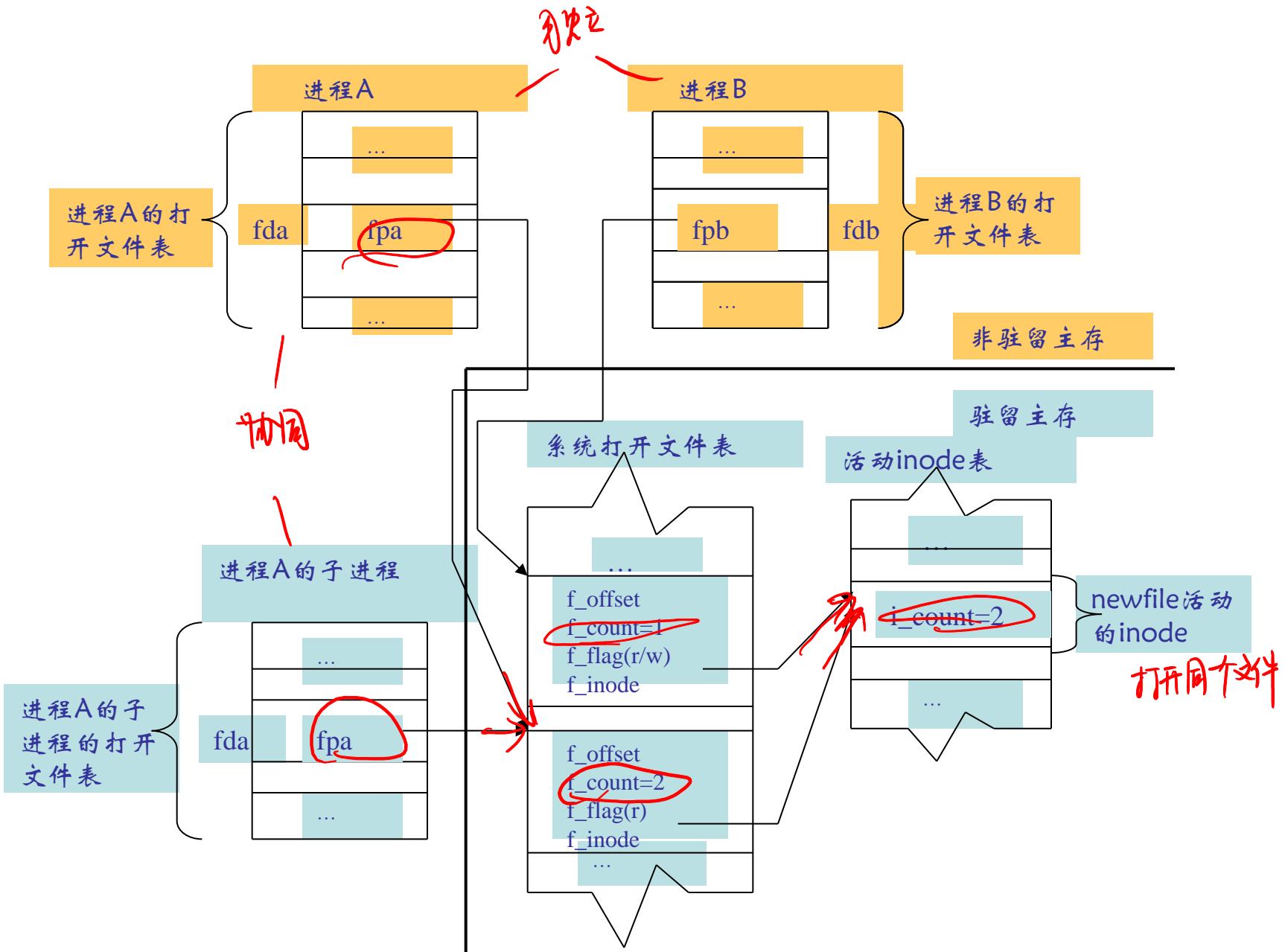
活动inode表(共享)

不能放在用户打开表项中。(独立)

- 同一用户父、子进程协同完成任务，使用同一读/写位移，同步地对文件进行操作。该位移指针似乎适合放在相应文件的活动索引节点中。
- 多用户共享文件，每个希望独立地读、写文件，这时不能只设置一个读写位移指针，须为每个用户进程分别设置一个读、写位移指针，该位移指针似乎放在用户打开文件表中合适。
- 为了解决上述矛盾，系统建立系统打开文件表来解决上述矛盾



子进程复制父进程的资源，包括用户打开文件表



使用不同位移指针的文件共享

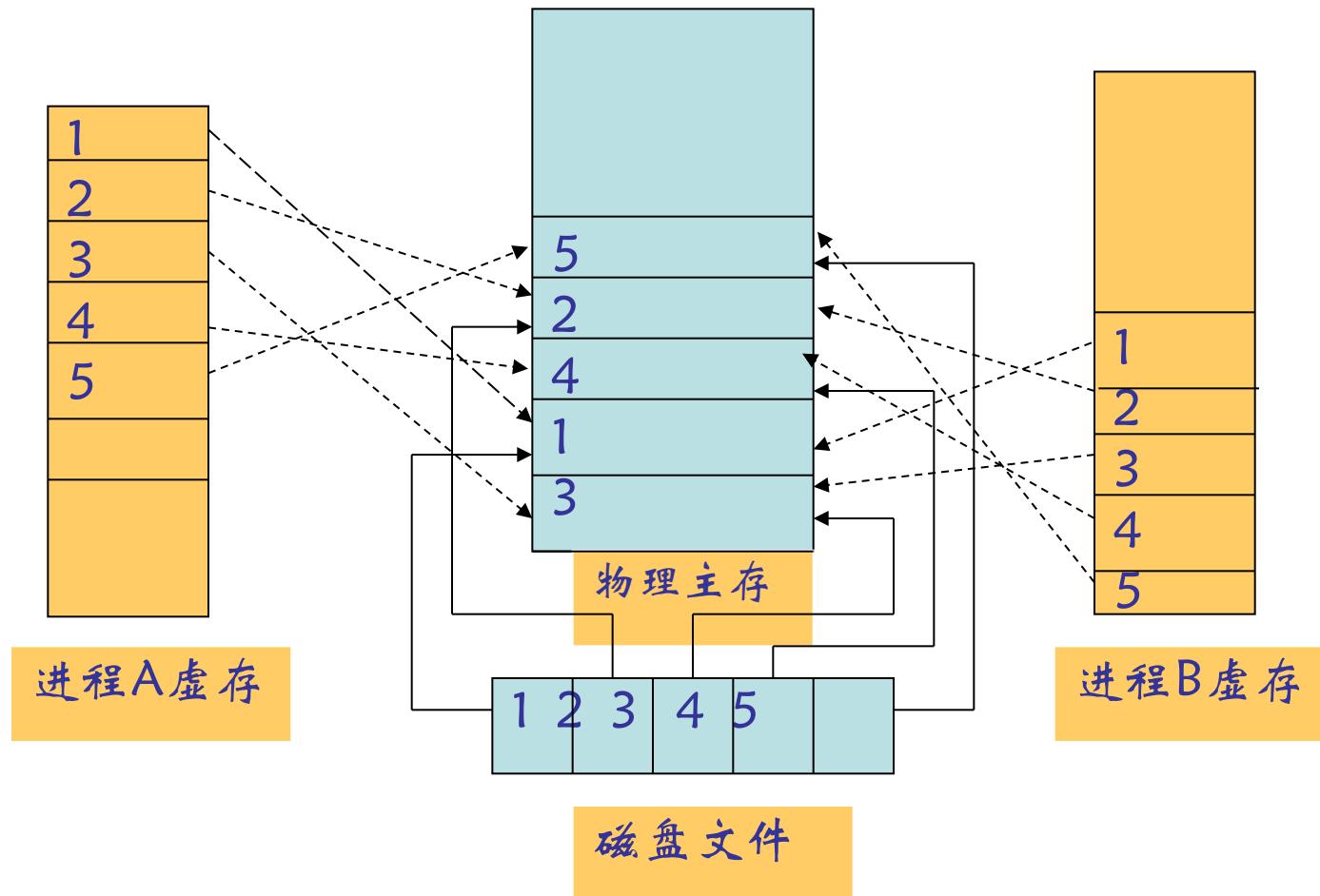
# 文件的符号链接共享

- 又称软链接， 符号链接是一种只有文件名，不指向inode的文件
- 符号链接共享文件的实现思想：
  - 用户A目录中形式： afile→bfile， 实现A的目录与B的文件的链接。 其中只包含被链接文件bfile的路径名而不是它的inode号

# 主存映射文件

- 主存映射文件 *直接访问内存*
  - 使用读写主存的操作来操作文件，简化编程
  - 基于虚拟存储管理机制
- 系统提供两个新的系统调用，
  - 映射文件mmap
    - 有两个参数：一个文件名和一个虚拟地址，把一个文件映射到进程地址空间。
  - 移去映射文件munmap
    - 让文件与进程虚拟地址空间断开，并把映射文件的数据写回磁盘文件。

# 主存映射文件



# mmap

- 头文件：
  - #include <unistd.h>
  - #include <sys/mman.h>
- 函数声明：
  - void \*mmap(void \*start, size\_t length, int prot,  
int flags, int fd, off\_t offsize);  
保护
- 函数说明：
  - mmap()用来将某个文件内容映射到内存中，对该内存区域的存取即是直接对该文件内容的读写。

参数	说明
start	指向欲对应的内存起始地址，通常设为NULL，代表让系统自动选定地址，对应成功后该地址会返回。 页面设置 倍
length	代表将文件中多大的部分对应到内存。
prot	代表映射区域的保护方式，有下列组合： <ul style="list-style-type: none"> <li>• PROT_EXEC 映射区域可被执行；</li> <li>• PROT_READ 映射区域可被读取；</li> <li>• PROT_WRITE 映射区域可被写入；</li> <li>• PROT_NONE 映射区域不能存取。</li> </ul>
flags	会影响映射区域的各种特性： <ul style="list-style-type: none"> <li>• MAP_FIXED 如果参数 start 所指的地址无法成功建立映射时，则放弃映射，不对地址做修正。通常不鼓励用此旗标。</li> <li>• MAP_SHARED 对应射区域的写入数据会复制回文件内，而且允许其他映射该文件的进程共享。 共享</li> <li>• MAP_PRIVATE 对应射区域的写入操作会产生一个映射文件的复制，即私人的“写入时复制”(copy on write)对此区域作的任何修改都不会写回原来的文件内容。</li> <li>• MAP_ANONYMOUS 建立匿名映射，此时会忽略参数fd，不涉及文件，而且映射区域无法和其他进程共享。</li> <li>• MAP_DENYWRITE 只允许对应射区域的写入操作，其他对文件直接写入的操作将会被拒绝。</li> <li>• MAP_LOCKED 将映射区域锁定住，这表示该区域不会被置换(swap)。</li> </ul> <p>在调用mmap()时必须要指定MAP_SHARED 或MAP_PRIVATE。</p>
fd	open()返回的文件描述词，代表欲映射到内存的文件。
offset	文件映射的偏移量，通常设置为0，代表从文件最前方开始对应，offset必须是分页大小的整数倍。

- 返回值：若映射成功则返回映射区的内存起始地址，否则返回**MAP\_FAILED(-1)**，错误原因存于**errno** 中。
- 错误代码：
  - **EBADF** 参数**fd** 不是有效的文件描述词。
  - **EACCES** 存取权限有误。如果是**MAP\_PRIVATE** 情况下文件必须可读，使用**MAP\_SHARED** 则要有**PROT\_WRITE** 以及该文件要能写入。
  - **EINVAL** 参数**start**、**length** 或**offset** 有一个不合法
  - **EAGAIN** 文件被锁住，或是有太多内存被锁住
  - **ENOMEM** 内存不足。

```

#include<sys/mman.h>
#include<sys/types.h>
#include<fcntl.h>
#include<stdio.h>
#include<unistd.h>
typedef struct{
    char name[4];
    int age;
}people;
void main(int argc, char **argv) //map a normal file as shared mem:
{
    int fd, i;
    people *p_map;
    fd=open(argv[1], O_CREAT|O_RDWR, 00777);
    p_map=(people*)mmap(NULL, sizeof(people)*10, PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, 0); start length PROT
for(i=0;i<10;i++)
{
    printf("name:%sage%d;\n", (*(p_map+i)).name, (*(p_map+i)).age);
}
munmap(p_map, sizeof(people)*10);
//释放文件

```

写入

读写

Flag

释放

# 虚拟文件系统

- 虚拟文件系统要实现以下目标
  - 同时支持多种文件系统；
  - 多个文件系统应与传统的单一文件系统没有区别，在用户面前表现为一致的接口；
  - 提供通过网络共享文件的支持，访问远程结点上的文件系统应与访问本地结点的文件系统一致；
  - 可以开发出新的文件系统，以模块方式加入到操作系统中。

# 虚拟文件系统

Linux VFS  
屏蔽对底层的实现

- 虚拟文件系统设计思想：  
  - 应用层：
    - 直接基于标准的UNIX文件系统调用来操作文件，无需考虑具体文件系统的特性和物理存储介质
  - 虚拟层：
    - 对具体文件系统的共同特性进行抽象，形成与具体文件系统的实现无关的虚拟层，定义与用户的一致性接口
  - 实现层：
    - 使用类似于开关表的技术进行具体文件系统的转接，实现各种文件系统的细节。
    - 自包含，包含文件系统的各种实施设施，如超级块、节点区、数据区以及各种数据结构和文件类的操作函数

# 本章小结

- 文件可以分为普通文件和目录文件，其中目录文件是完全由目录项组成的文件
- 文件目录的主要作用之一是实现文件的按名存取，即将文件名转换成物理块
- Linux/Unix将文件名和文件的其它属性进行分离，而将其它属性单独用一个数据结构inode表示，文件目录的表项仅包括文件名和inode号
- 文件的物理组织方式包括顺序/连续文件、连接文件、**FAT**和索引文件等，其中连接文件不适合随机存取

# 本章小结

- 文件空间的管理方法可以用位示图、空闲区表、空闲块链和空闲块成组管理法等
- 文件系统通常包含**用户打开文件表**和**系统打开文件表**两个数据结构，返回给用户的**文件描述符即为在用户打开文件表中的下标**，可以通过这两个表来实现文件的共享
- 文件的动态共享通过引入系统打开文件表来实现，父子进程可以通过共享同一个读写指针来协同工作。