上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

L10. 文件系统

宋卓然

上海交通大学计算机系

songzhuoran@sjtu.edu.cn

基本概念

- 文件系统：一种用于持久性存储的系统抽象，映射逻辑文件系统到物理外存设备
  - 在存储器上：组织、控制、导航、访问、和检索数据
  - 大多数计算机系统都包含文件系统
- 文件：文件系统中一个单元的相关数据在操作系统中的抽象
- 从用户的角度来看，文件是逻辑外存的最小分配单元，数据只能通过文件才能写到外存
- 文件被命名以方便人们使用，当被命名后，它就独立于进程、用户、甚至创建它的系统
  - 一个用户创建a.c文件，另一个用户通过邮件发送它，仍称为a.c

文件类别

- 文件名分为两部分：名称+扩展
- 可以通过文件名得知文件类别
  - 文本文件
  - 源文件
  - 可执行文件
  - 音频、视频文件

file type	usual extension	function
executable	exe, com, bin or none	ready-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, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	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, mp3, avi	binary file containing audio or A/V information

文件属性

- 文件的属性因操作系统而异（便于检索），通常包括：
  - 名称：符号文件名是以人类可读形式来保存的唯一信息。
  - 类型：支持不同类型文件的系统需要这种信息。
  - 尺寸：包括文件的当前大小及允许的最大尺寸（以字节为单位）。
  - 位置：该信息为指向设备与设备上文件位置的指针。
  - 保护：访问控制信息确定谁能进行读取、写入、执行等。
  - 时间、日期和用户标识：文件创建、最后修改和最后使用的相关信息可以保存。这些数据用于保护、安全和使用监控

文件操作

- 创建文件：创建文件需要两个步骤。首先，必须在文件系统中为文件找到空间。其次，必须在目录中创建新文件的条目
- 写文件：为了写文件，使用一个系统调用指定文件名称和要写入文件的信息。根据给定的文件名称，系统搜素目录以查找文件位置。系统应保留写指针(write pointer), 用于指向需要进行下次写操作的文件位置。每当发生写操作时，写指针必须被更新
- 读文件:为了读文件，使用一个系统调用， 指明文件名称和需要文件的下一个块应该放在哪里(在内存中)。同样，搜素目录以找到相关条目，系统需要保留一个读指针(read pointer)，指向要进行下一次读取操作的文件位置

文件操作

- 重新定位文件：搜索目录以寻找适当的条目，并且将当前文件位置指针重新定位到给定值
- 删除文件：为了删除文件，在目录中搜索给定名称的文件。找到关联的目录条目后，释放所有文件空间，以便它可以被其他文件重复使用，并删除目录条目
- 截断文件：用户可能想要删除文件的内容，但保留它的属性。不是强制用户删除文件再创建文件，这个功能允许所有属性保持不变，(除了文件长度)，但让文件重置为零，并释放它的文件空间
- 其他操作可以由这些原始操作组合实现

文件打开操作

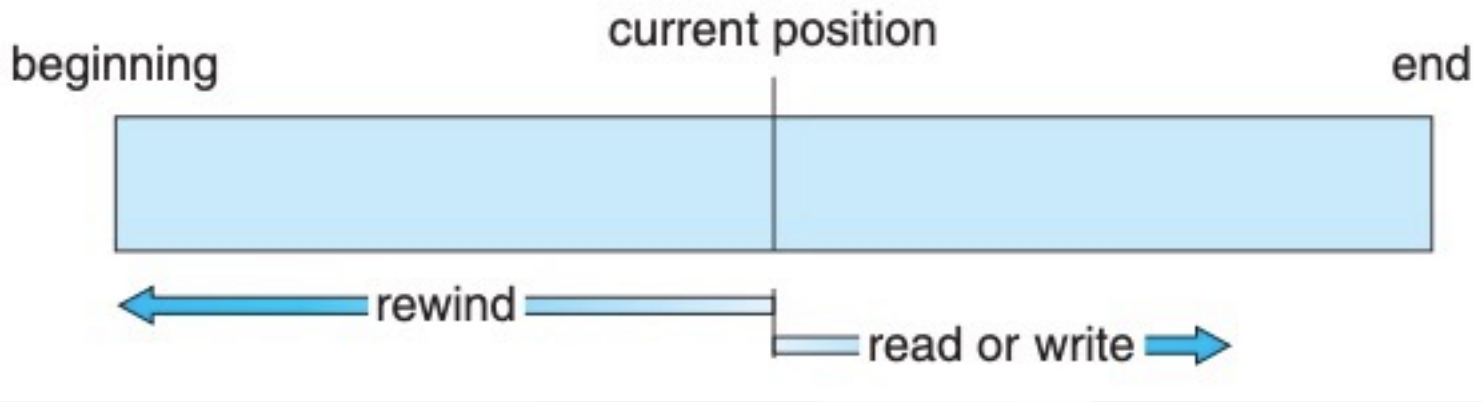
- 大多数文件操作涉及搜索目录，以得到命名文件的相关条目。为了避免不断搜索，系统维护一个**打开文件表**，保存所有打开文件的信息
  - 首次使用文件，调用系统调用open()
- 当请求文件操作时，访问打开文件表，找到指定文件
- 当近期不再使用文件，进程关闭它，操作系统从打开文件表中删除对应条目

文件打开操作

- 每个打开文件具有如下关联信息：
  - 文件指针：系统跟踪上次读写位置
  - 文件打开计数：有多少进程打开了同一文件。在关闭文件时，只有当计数等于0才能关闭，并从系统的打开文件表中删除条目
  - 文件的磁盘位置：将该信息保存在内存中，以便系统不必为每个操作都从磁盘上读取该信息
  - 访问权限：操作系统可以允许或拒绝后续的I/O请求

文件访问方法

- 顺序访问
  - 文件信息按顺序加以处理
  - 读取文件的下一部分 read_next()
  - 写文件，在文件结尾附加内容，并将指针移到新写材料的末尾 write_next()



文件访问方法

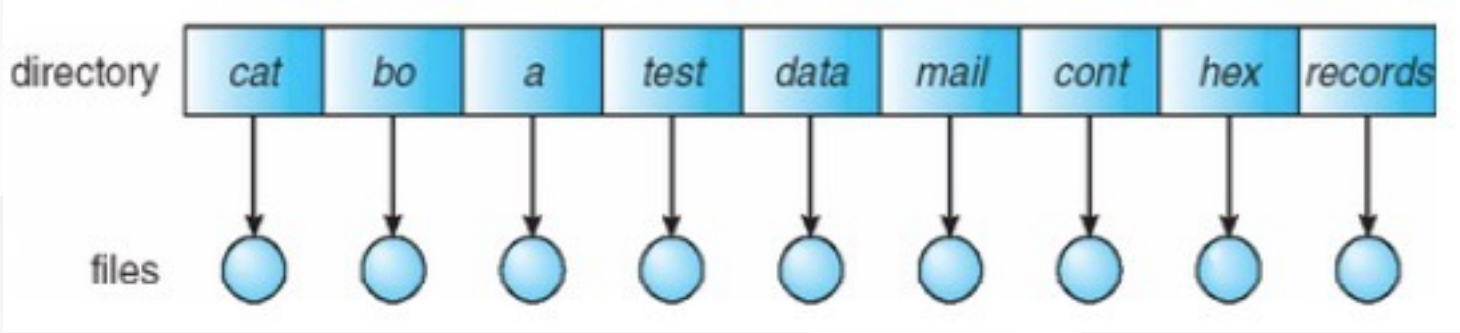
- 直接访问
  - 文件由固定长度的逻辑记录组成，允许程序按任意顺序进行读写
  - 基于文件的磁盘模型，因为磁盘允许对任何文件块的随机访问
  - read(n)：读第n块
  - write(n)：写第n块
  - n：相对位置，相对于文件开头的索引，开头位置由操作系统决定，文件间不能有重叠

目录概述

- 目录可视为符号表
- 可对目录执行的操作：
  - 搜索文件
  - 创建文件
  - 删除文件
  - 遍历目录
  - 重命名文件
  - 遍历文件系统

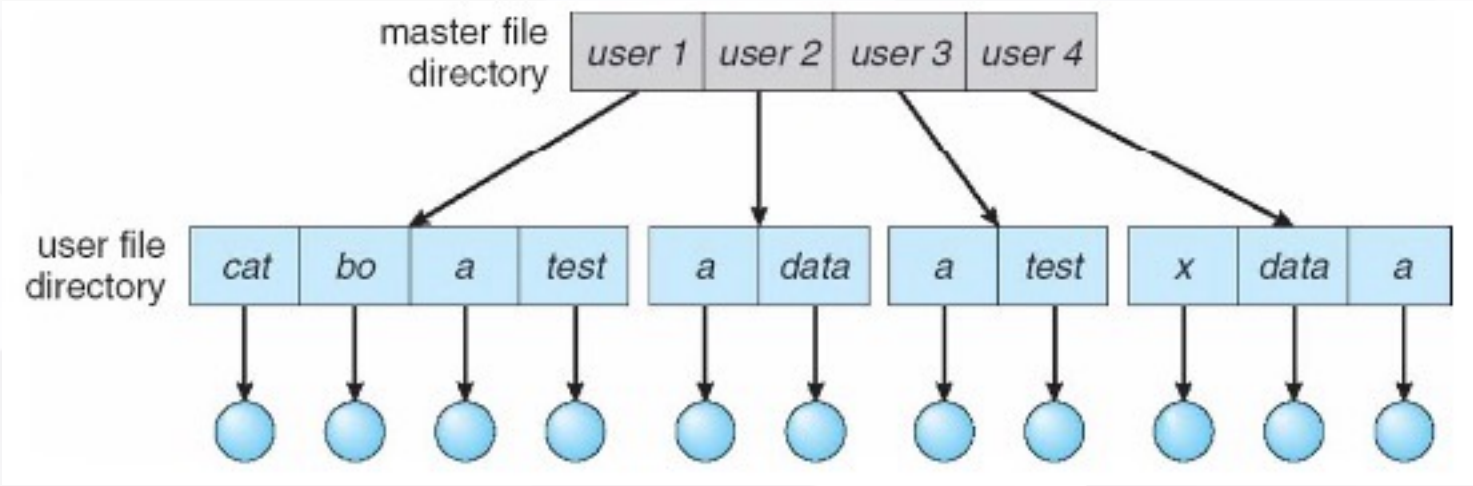
单级目录

- 所有文件都包含在同一目录
- 所有文件必须具有唯一的名称
- 但随着文件数量的增加，即使单级目录的单个用户也会难以记住所有文件的名称



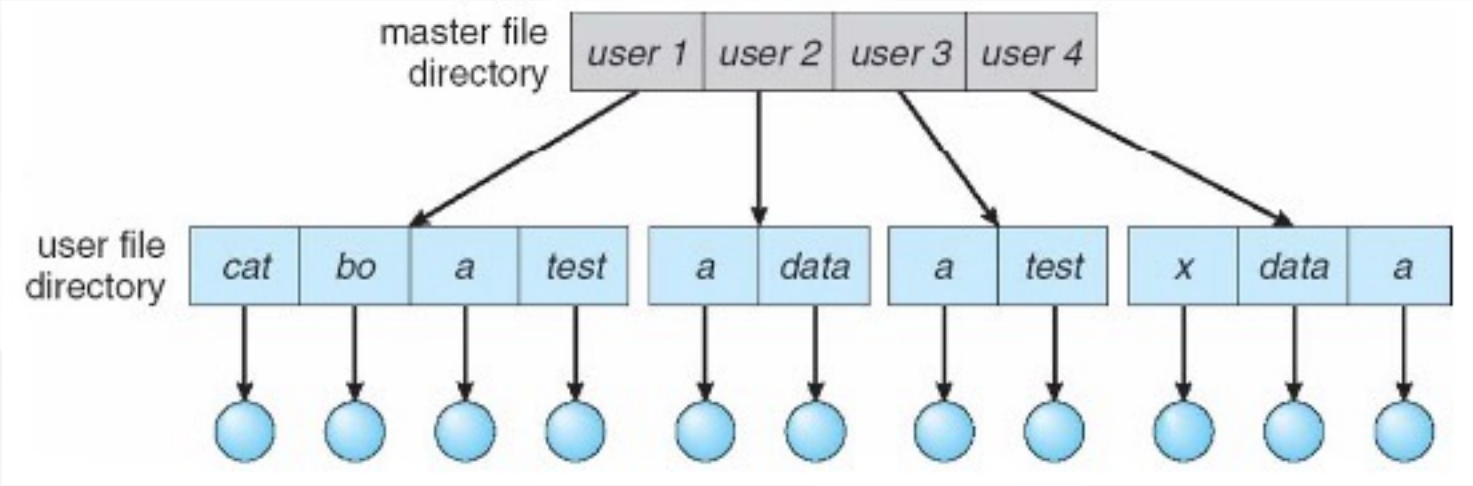
两级目录

- 为每个用户创建一个单独的目录
- 每个用户拥有自己的用户文件目录（user file directory, UFD）
- 不同用户可以拥有相同名称的文件
- 有效地将一个用户与另一个用户隔离
- 相较单级目录会更高效



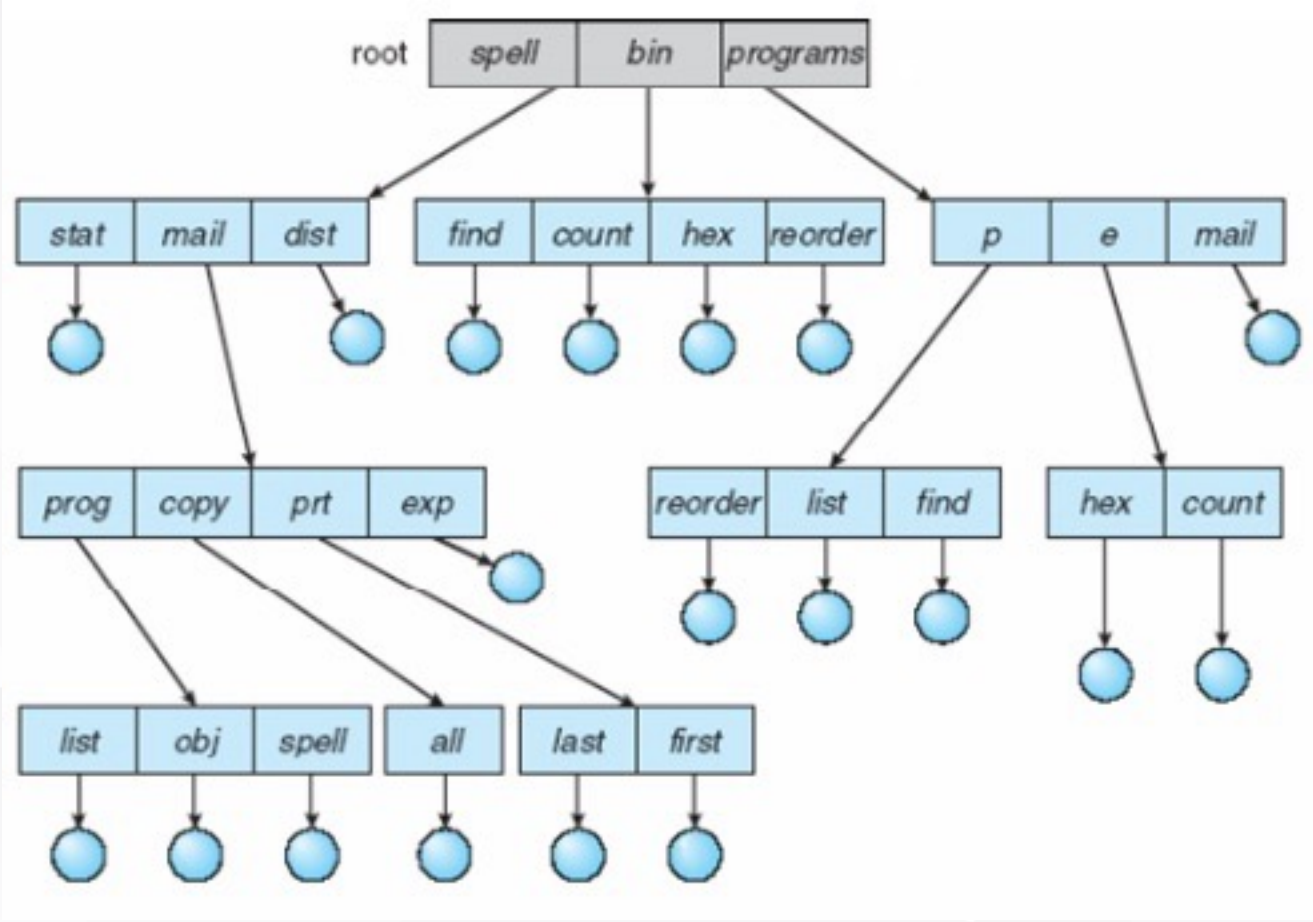
两级目录

- 一个用户想访问另一个用户，可以将系统的主文件目录（master file ditectory，MFD）作为树根；树根的直接后代为用户文件目录
- 因此只需要指定用户名和文件名，就可以访问其他用户
- 即将两级目录视作两级树



树形目录

- 允许用户创建自己的子目录并相应地组织文件
- 具有根目录，系统内的每个文件都有唯一的路径名

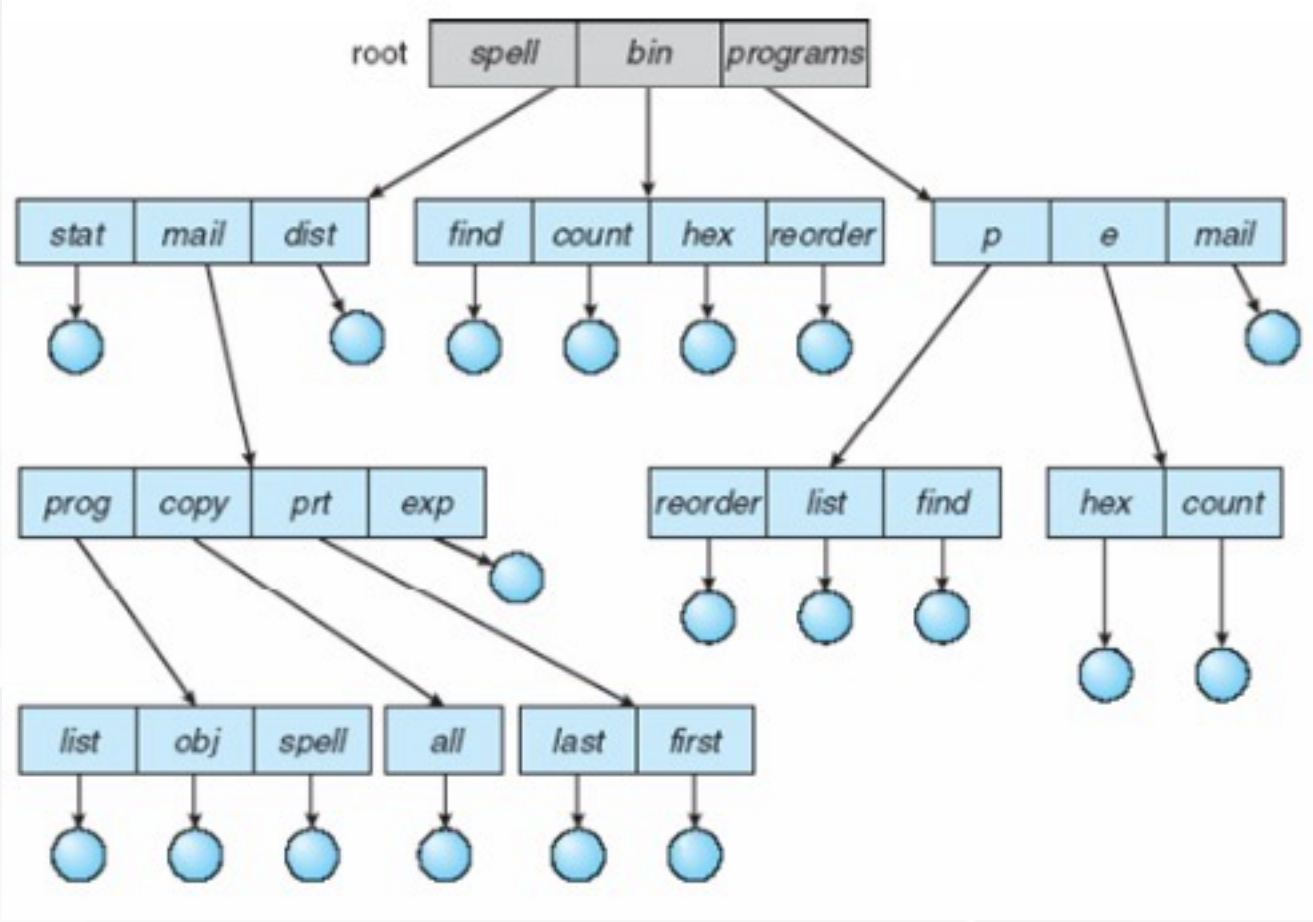


树形目录

- 每个进程都有一个当前目录，当引用一个文件时，优先搜索当前目录
- 若不在当前目录，则需要使用系统调用change_directory()到下一个目录
- 路径名的两种形式：绝对路径名和相对路径名
  - 绝对路径名从树根开始，遵循一个路径到指定文件，并给出这个路径上的目录名
  - 相对路径名从当前目录开始，定义一个路径

树形目录

- 如果当前目录是root/spell/mail，绝对路径名root/spell/mail/prt/first和相对路径名prt/first指向同一文件



保护

- 当信息存储在计算机系统中，需要保护它的安全，以便避免物理损坏（可靠性问题）和非法访问（保护问题）
- 可靠性可通过文件的重复副本来提供
  - 计算机自动地或定期地把可能损坏的文件系统复制到磁盘
- 保护问题
  - 通过控制多个不同的操作类型
  - 根据用户身份控制访问

访问类型

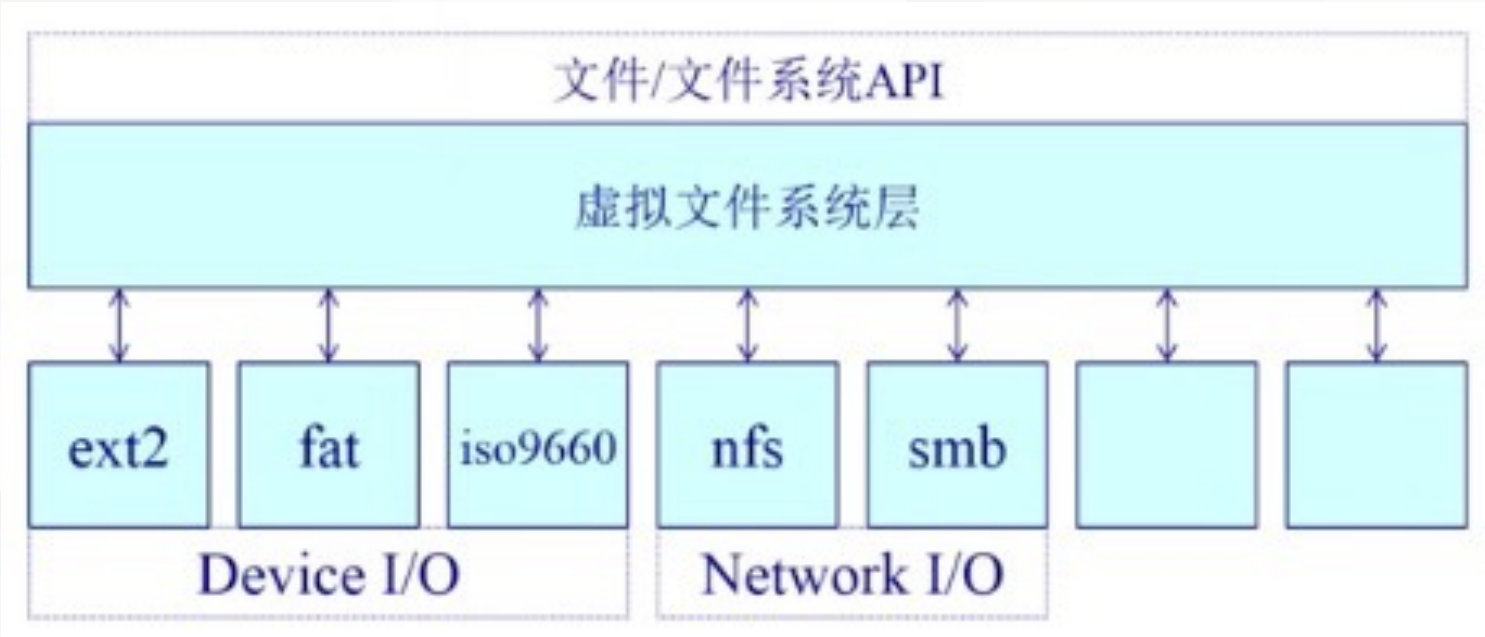
- 控制不同的操作类型
  - 读：从文件中读取。
  - 写：写或重写文件。
  - 执行：加载文件到内存并执行它。
  - 附加：在文件末尾写入新的信息。
  - 删除：删除文件，并释放空间以便重复使用。
  - 列表：列出文件的名称和属性。

访问控制

- 基于身份的访问最普遍的实现方法：为每个文件、目录关联一个访问控制列表，如果该用户是属于可访问的，则允许访问
- 但这项技术有两项不可取的后果：
  - 构造这样的列表是一个冗长乏味的任务
  - 目录条目是可变大小的，从而导致更为复杂的空间管理
- 通过精简的访问列表，定义三种用户类型
  - 所有者：创建文件的用户为所有者
  - 组：共享文件并且需要类似访问的一组用户是组，由所有者设定访问权限
  - 其他：系统内的所有其他用户

虚拟文件系统

- 分层结构
  - 上层：虚拟（逻辑）文件系统，给用户提供的API包括 open,close,read,write
  - 底层：特定文件系统模块



虚拟文件系统

- 目的
  - 对所有不同文件系统的抽象
- 功能
  - 提供相同的文件和文件系统接口
  - 管理所有文件和文件系统相关联的数据结构
  - 高效查询例程，遍历文件系统
  - 与特定文件系统模块的交互

文件分配方法

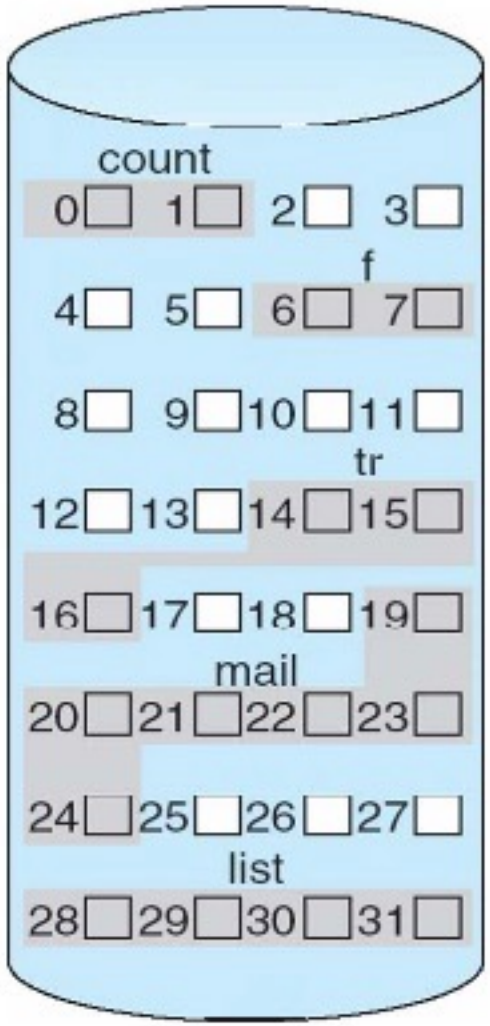
- 很多文件都是存储在同一个磁盘，如何为这些文件分配空间?
- 有些文件很小
  - 需要对小文件提供强力支撑
  - 块空间不能太大
- 一些文件非常大
  - 需要支持大文件
  - 大文件访问需要相当高效

文件分配方法

- 分配方式
  - 连续分配
  - 链式分配
  - 索引分配
- 指标
  - 高效：如存储利用（外部碎片）
  - 表现：如访问速度

连续分配

- 每个文件在磁盘上占有一组连续的块
  - 指定文件的起始块与长度
  - 性能在大多数情况下最优：当需要磁头移动（从一个柱面的最后扇区到下一个柱面的第一扇区）时，只需要移动一个磁道
    - 文件读取表现好
    - 高效的顺序和随机访问

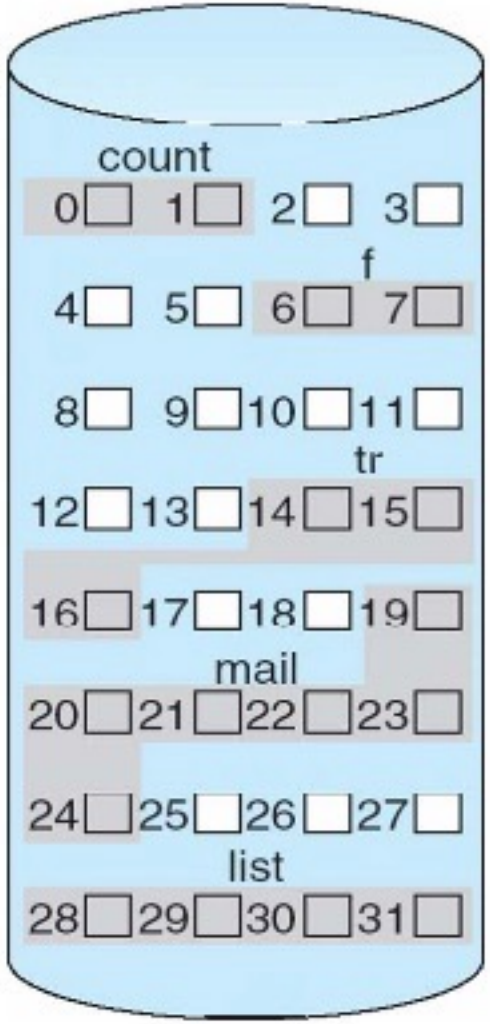


directory

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

连续分配

- 缺点
  - 文件扩展的问题
    - 需要将后续的文件移动，腾出位置
  - 碎片
- 适用于只读文件

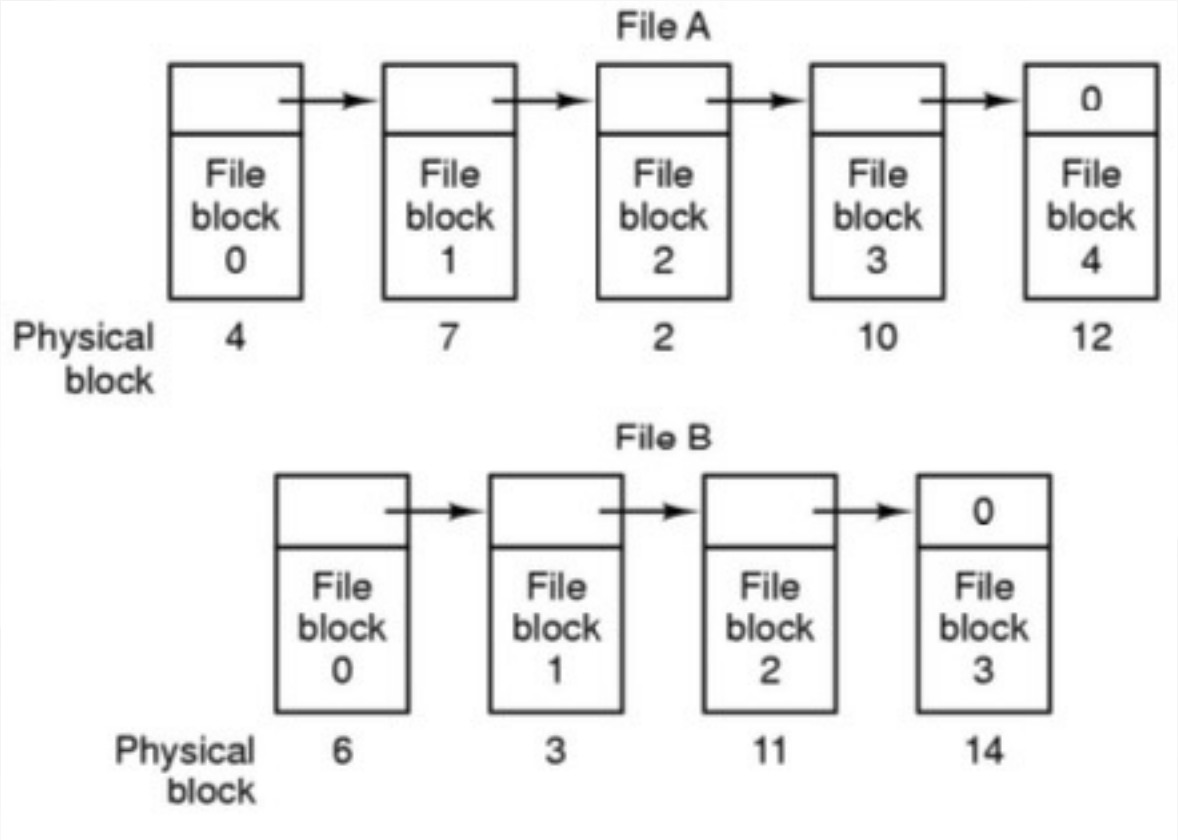


directory

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

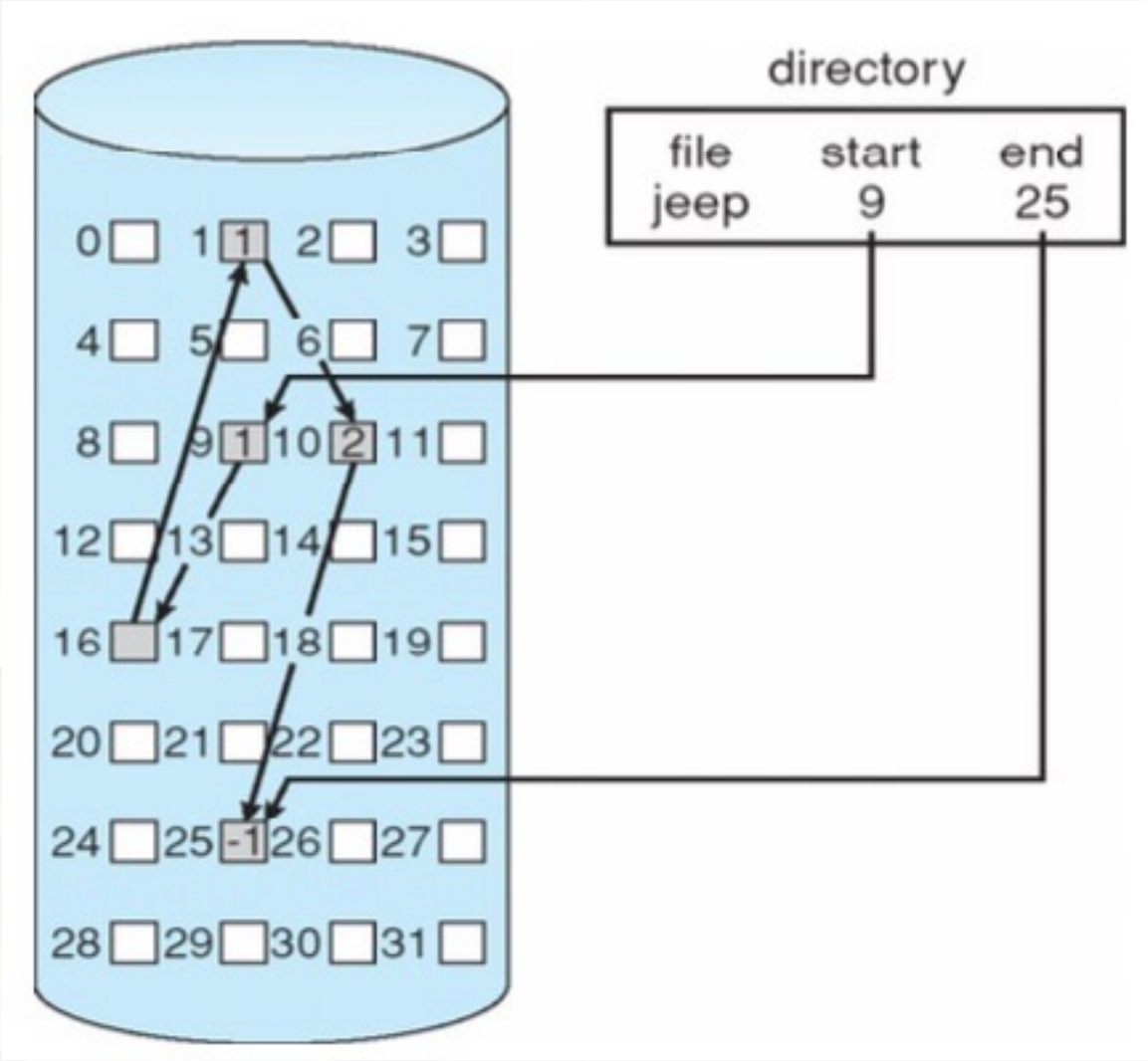
链式分配

- 每个文件时磁盘块的链表，可以散步在磁盘的任何地方



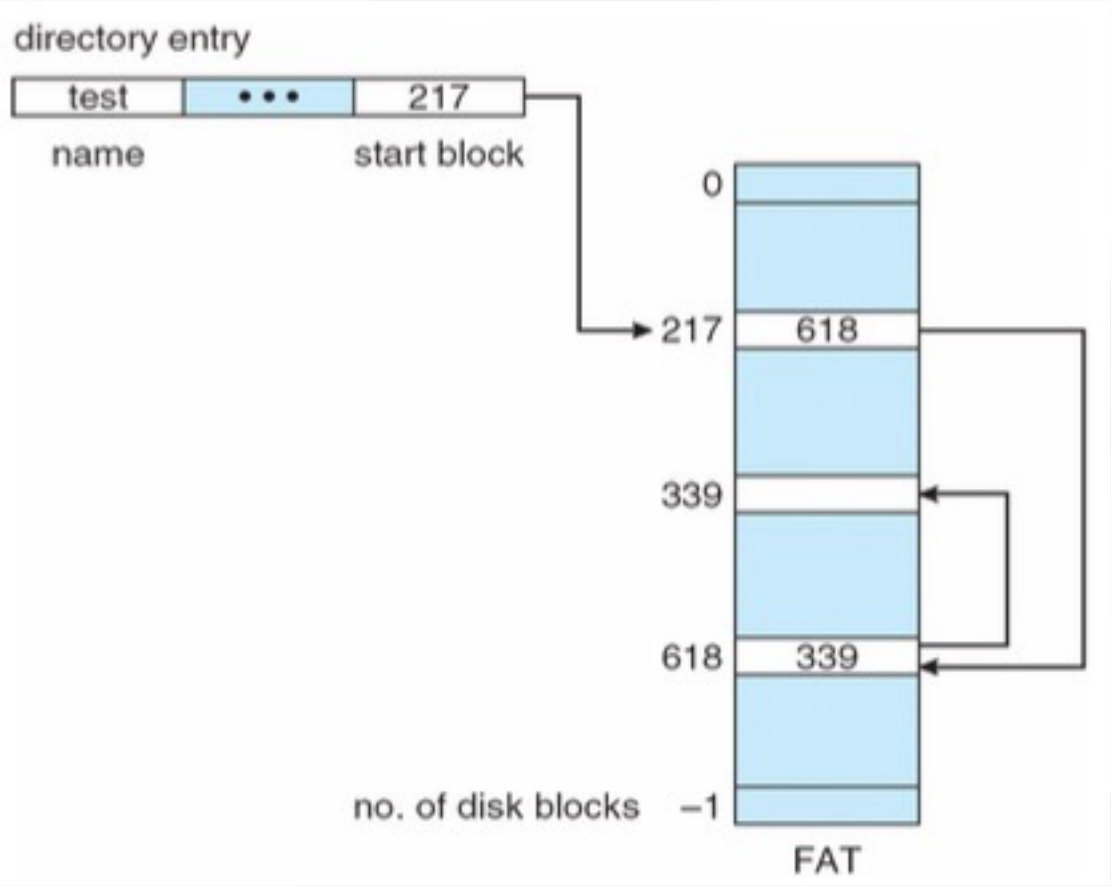
链式分配

- 文件头包含了第一块和最后一块的指针
- 优点
  - 创建、增加、缩小很容易
  - 没有碎片
- 缺点
  - 串行访问
  - 指针所需的空间
  - 可靠性：如果指针丢失，无法正确获取文件信息



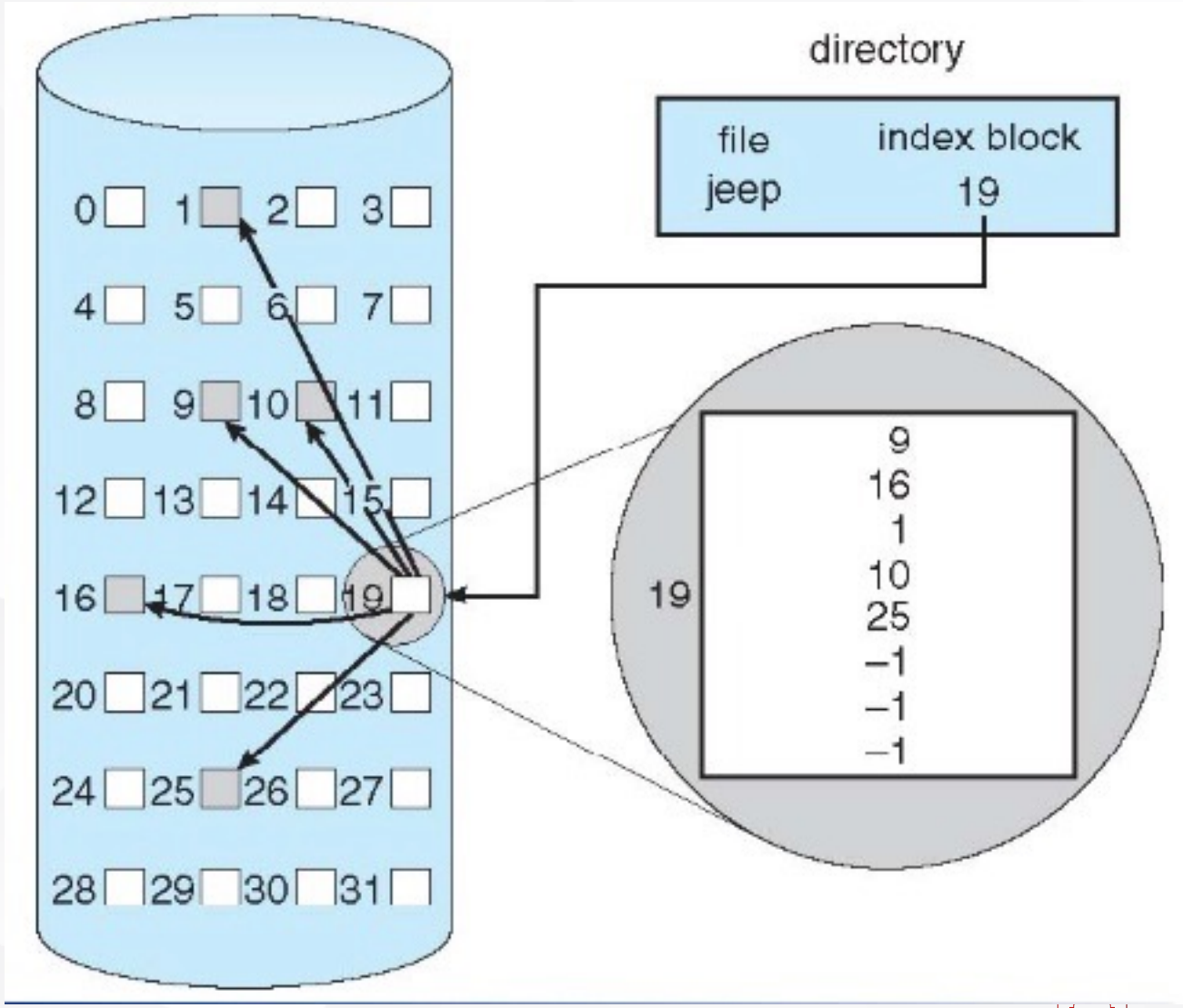
文件分配表（file-allocation table，FAT）

- 链式分配的变种
- 目录条目记录了文件首块的块号
- 维护FAT，在该表中，每个磁盘块有一个条目，并可按块号来索引
- 过程：磁头移动到文件头，读入FAT，找到所需块的位置，再将磁头移到块本身的位置
- 改善了随机访问，因为读入了FAT信息，快速比对，找到相应位置
- 用于MS-DOS操作系统



索引分配

- 每个文件都有自己的**索引块**，是一个磁盘块地址的数组
- 当创建文件时，索引块的所有指针设为NULL
- 当首次写入第i块，先从空闲空间管理器中获得一块，再将其磁盘地址写到索引块的第i个条目



索引分配

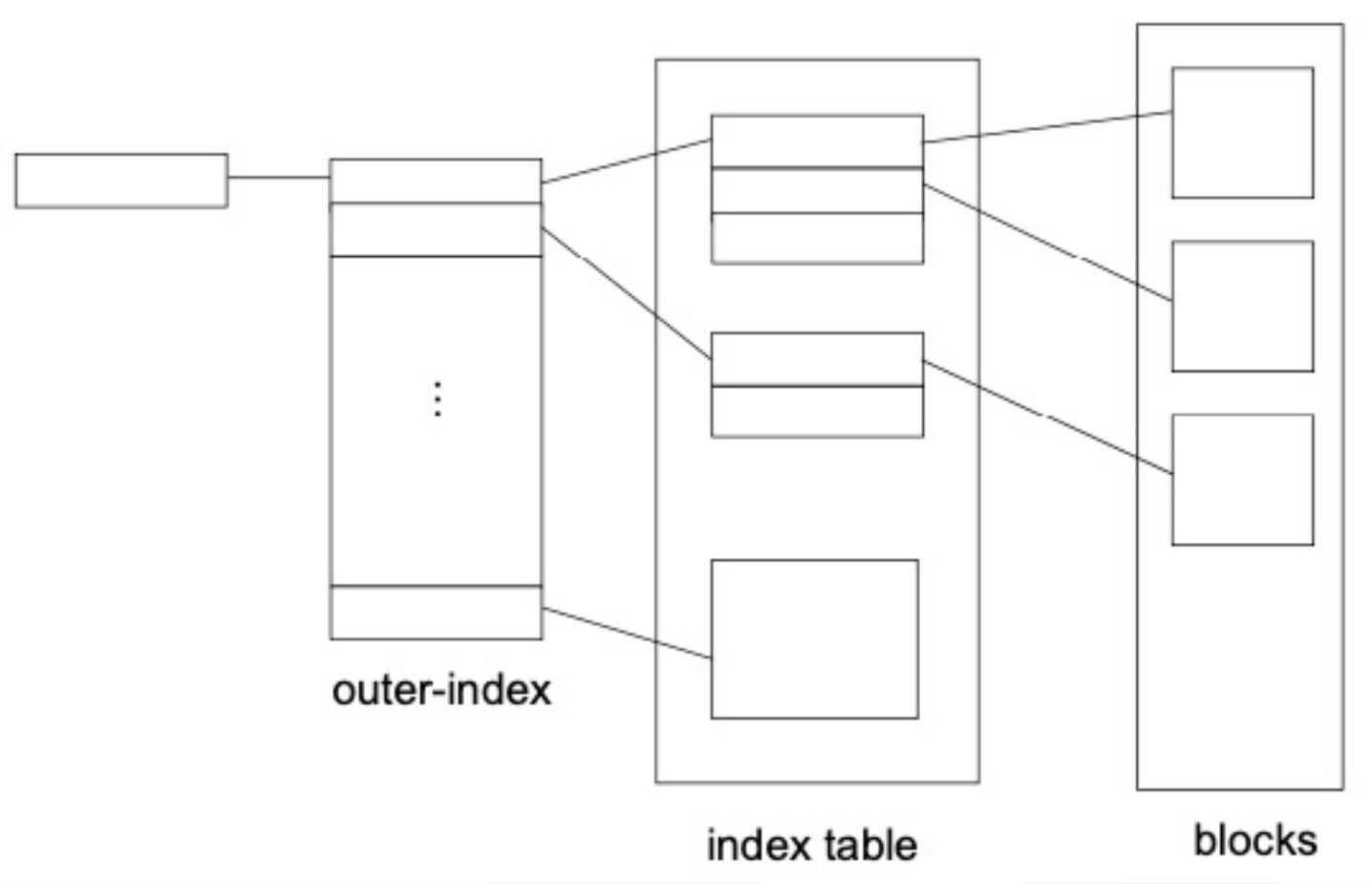
- 支持随机访问
- 没有外部碎片
- 需要存储索引块，浪费空间
  - 即使一个文件只占用了一个磁盘块，依然需要维护一个完整的索引块
- 索引块应该多大?
  - 太小：无法为大文件提供足够多的指针
  - 太大：浪费空间

索引块

- 链式索引块
  - 为了支持大文件，将多个索引块链接起来，索引块需要存储下一个索引块的指针
  - 缺点：
    - 指针开销
    - 可靠性
- 多级索引块（类似于多级页表）
  - 第一级索引块指向第二级索引块
  - 缺点：
    - 访存速度受限，要访问多级索引才能找到相应文件

索引块

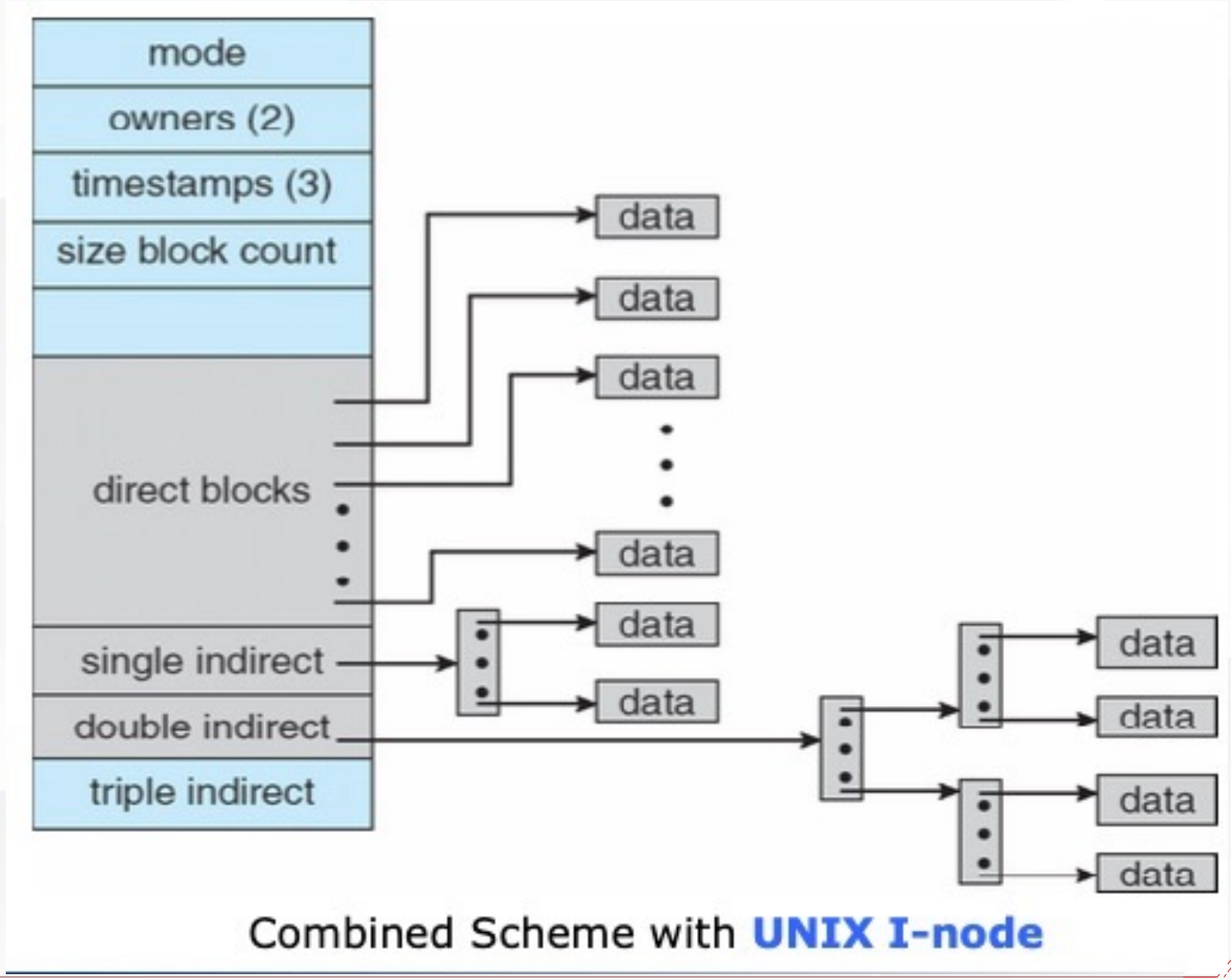
- 多级索引块（类似于多级页表）



索引块

- 组合方案，常见于早期的UNIX文件系统
  - 将索引块的前几个(如15) 指针存在文件的inode中
  - 这些指针的前12个指向直接块，即包含存储文件数据的块的地址
    - 小文件不需要多级索引
  - 剩余3个指针指向间接块
    - 第一个指向一级间接块
    - 第二个指向二级间接块
    - 第二个指向三级间接块

索引块



Combined Scheme with **UNIX I-node**

I-node实例

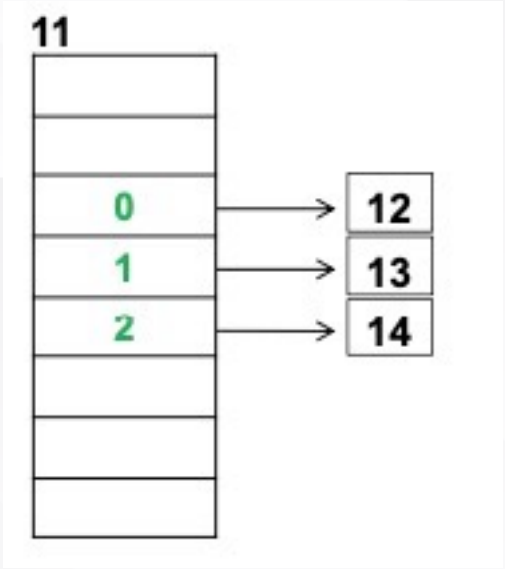
- 假设文件系统是使用32字节的块构建的。指针需要4个字节。I-node结构如下（字，值）：

0	Permission word
1	File Size
2	Direct block
3	Direct block
4	Direct block
5	Direct block
6	Single-indirect
7	Double-indirect

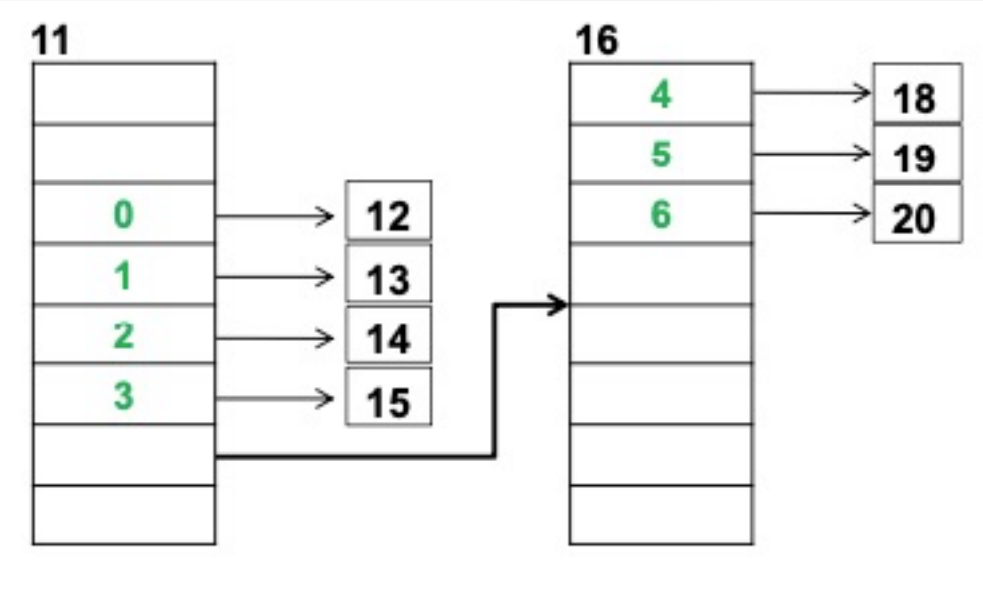
- 假设可用块从块11开始按逻辑顺序分配。此外，已经确定区块17和32是坏的，无法分配。
- 绘制一个框图，显示I-node的结构和分配的块
  - 原始文件有3个块
  - 增加4个块；增加17个块

I-node实例 原始文件有3个块

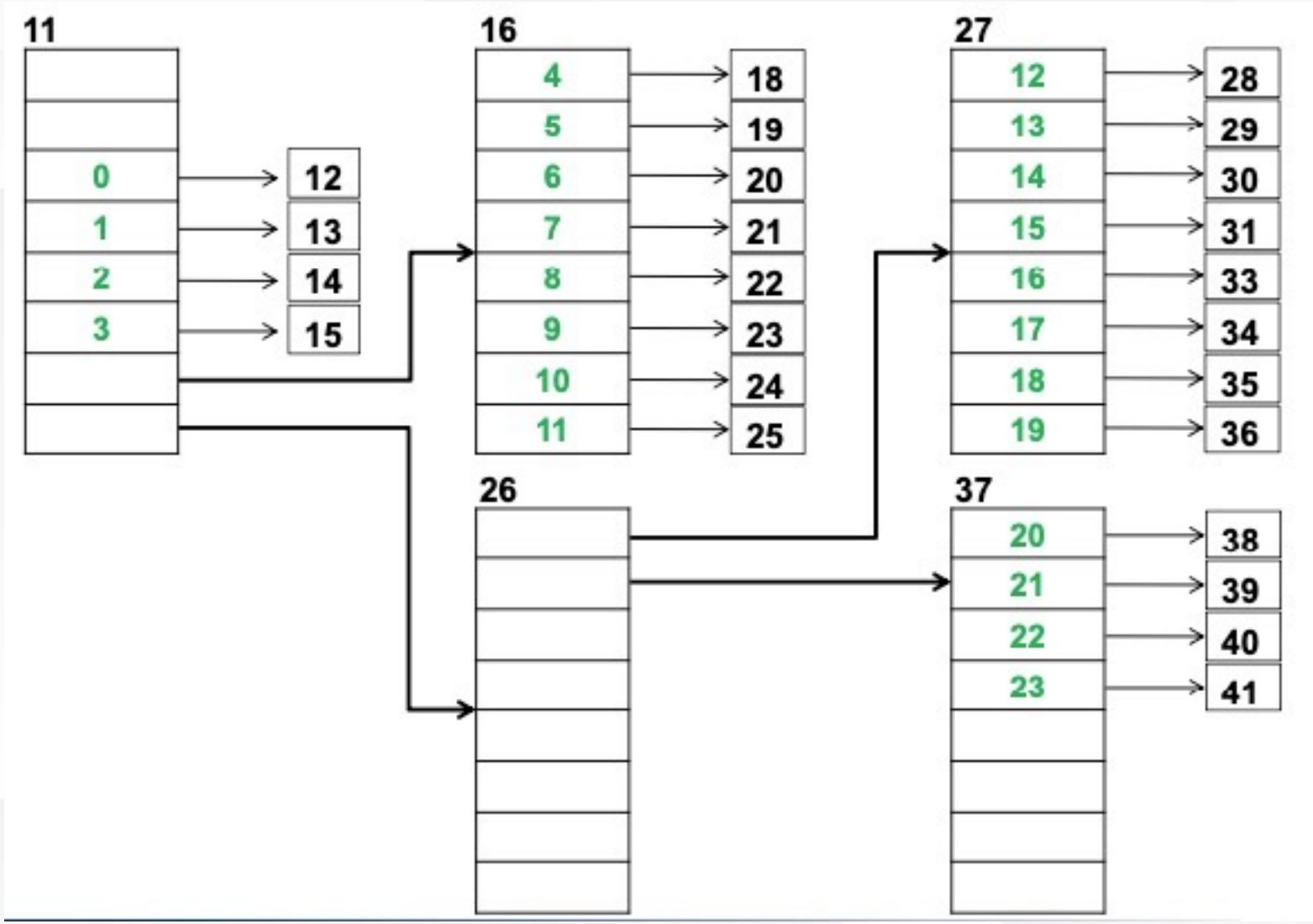
8行=32字节/4字节指针



I-node实例 增加4个块



I-node实例 增加17个块



课堂习题

- 假设文件系统是使用32字节的块构建的。指针需要4个字节。I-node结构如下（字，值）：

0	Permission word
1	File Size
2	Direct block
3	Direct block
4	Direct block
5	Single-indirect
6	Double-indirect
7	Triple-indirect

- 假设可用块从块100开始按逻辑顺序分配。此外，已经确定区块107、108、109、112是坏的，无法分配。
- 绘制一个框图，显示I-node的结构和分配的块：

1.原始文件有3个块；2.增加7个块； 3.增加24个块； 2.增加64个块；