

第12章 文件管理

- 主要内容

- 12.1 概述 ★★☆☆☆
- 12.2 文件组织和访问 ★★☆☆☆
- 12.3 B树 ★★☆☆☆
- 12.4 文件目录 ★★☆☆☆
- 12.5 文件共享 ★★☆☆☆
- 12.6 记录组块 ★★☆☆☆
- 12.7 辅助存储管理 ★★☆☆☆
- 12.8 文件系统安全 ★★☆☆☆
- 12.9 UNIX文件管理 ★★☆☆☆
- 12.10 Linux虚拟文件系统（自学）
- 12.11 Windows文件系统（自学）

12.1 概述 ★★☆☆☆

12.1.1 文件和文件系统

- 文件
 - 由文件名字标识的一组信息的集合。
- 文件系统
 - 文件系统是操作系统中负责存取和管理信息的模块，它用统一的方式管理用户和系统信息的存储、检索、更新、共享和保护，并为用户提供一整套方便有效的文件使用和操作方法。

-
- 文件的理想属性
 - 长期存在
 - 进程间可共享
 - 结构
 - 文件系统提供的对文件的典型操作
 - 创建
 - 删除
 - 打开
 - 关闭
 - 读
 - 写

12.1.2 文件结构 ★★☆☆☆

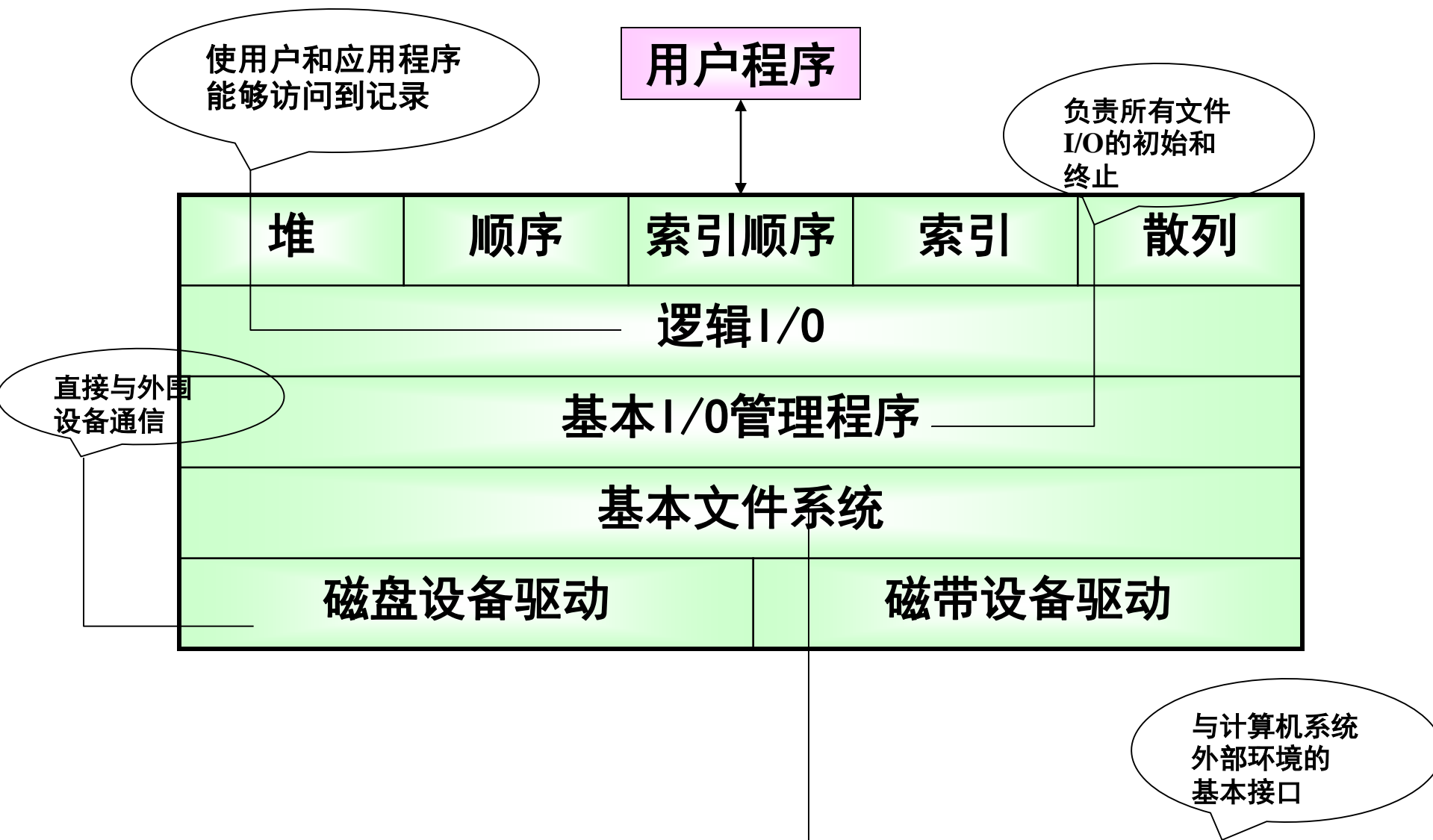
- 域
 - 基本数据单元。
- 记录
 - 一组相关的域的集合。
- 文件
 - 一组相似记录的集合。
- 数据库
 - 一组相关的数据的集合。

12.1.3 文件管理系统 ★★☆☆☆

- 文件管理系统是一组**系统软件**，为使用文件的用户和应用程序提供服务。一个文件管理系统需要满足以下目标：
 - 满足数据管理的要求和用户的需求，包括**存储**数据和执行上述**操作**的能力。
 - 最大限度地保证文件中的**数据有效**。
 - 优化性能，包括总体吞吐量和响应时间。
 - 为各种类型的存储设备提供I/O支持。
 - 减少或消除丢失或破坏数据的可能性。
 - 向用户进程提供标准I/O接口例程集。
 - 在多用户系统中为多个用户提供I/O支持。



文件系统架构



文件系统架构

- 设备驱动

- 程序直接与外围设备(或它们的控制器或通道)通信。设备驱动程序负责启动设备上的I/O操作,
- 处理I/O请求的完成。
- 对于文件操作,典型的控制设备是磁盘和磁带设备。
- 设备驱动程序通常是操作系统的一部分。

文件系统架构

- 基本文件系统(物理I/O层)
 - 与计算机系统外部环境的基本接口。
 - 这一层处理在磁盘间或磁带系统间交换的数据块，因此它关注的是这些块在辅存和内存缓冲区中的位置。
 - 通常是操作系统一部分。

文件系统架构

- 基本I/O管理程序(basic I/O supervisor)
 - 负责所有文件I/O的初始化和终止。
 - 在这一层，需要一定的控制结构来维护设备的输入/输出、调度和文件状态。
 - 根据所选的文件来选择执行文件I/O的设备，为优化性能，它还参与调度对磁盘和磁带的访问。
 - I/O缓冲区的指定和辅存的分配，也是在这一层实现的。
 - 基本I/O管理程序是操作系统的一部分。

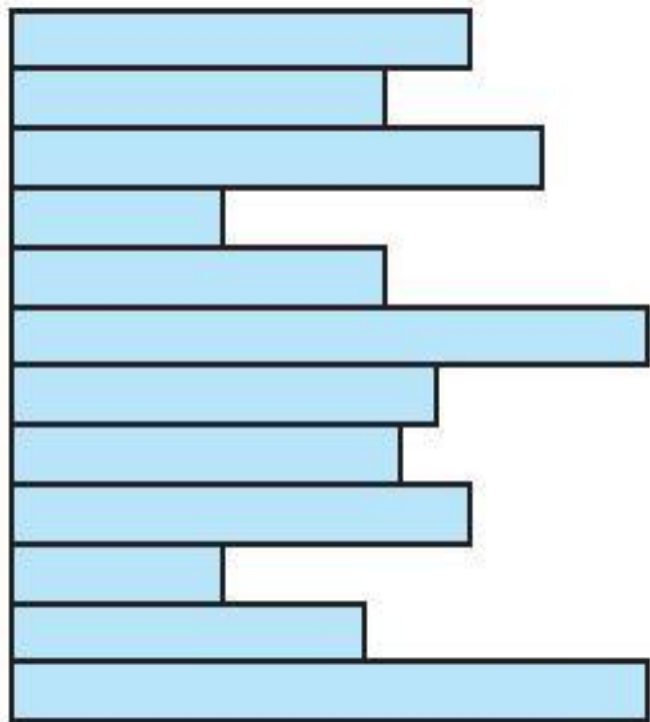
文件系统架构

- 逻辑I/O (logical I/O)
 - 使用户和应用程序能够访问记录。
 - 逻辑I/O提供一种通用的记录I/O能力，并维护关于文件的基本数据。
- 访问方法 (access method)
 - 它在应用程序和文件系统以及保存数据的设备之间提供了一个标准接口。
 - 不同的访问方法反映了不同的文件结构及访问和处理数据的不同方法。

12.2 文件组织和访问 ★★☆☆☆

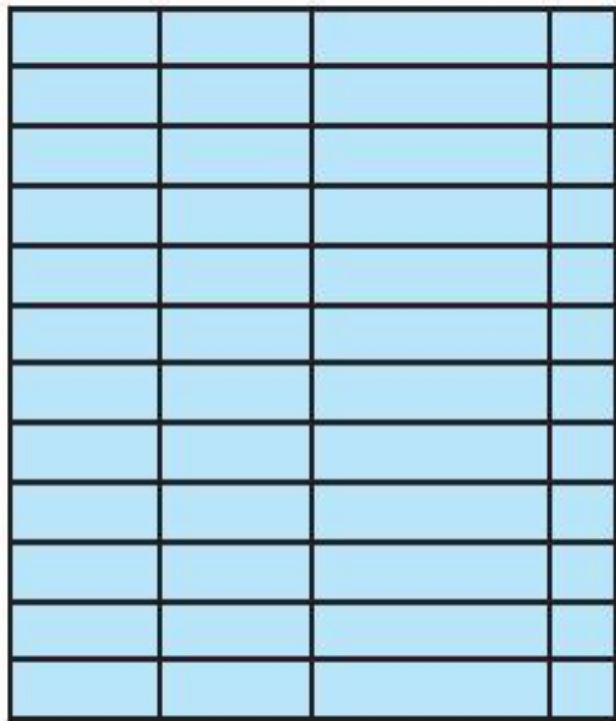
- 文件组织指文件记录中的**逻辑**结构，由用户访问文件的方式确定。
- 文件在辅存中的**物理**组织取决于分块策略和文件分配策略。
- 5种基本文件组织方式：
 - 堆
 - 顺序文件
 - 索引顺序文件
 - 索引文件
 - 直接或散列文件

12.2.1 堆



- 数据按到达的顺序被收集，没有结构，对记录的访问通过穷举查找的方式进行。
- 适用场合：当数据在处理前采集并存储时，或者当数据难以组织时。
 - 当保存的数据大小和结构不同时，堆文件空间使用情况良好，能较好地用于穷举查找，且易于修改。
- 对大多数应用都不适应。

12.2.2 顺序文件

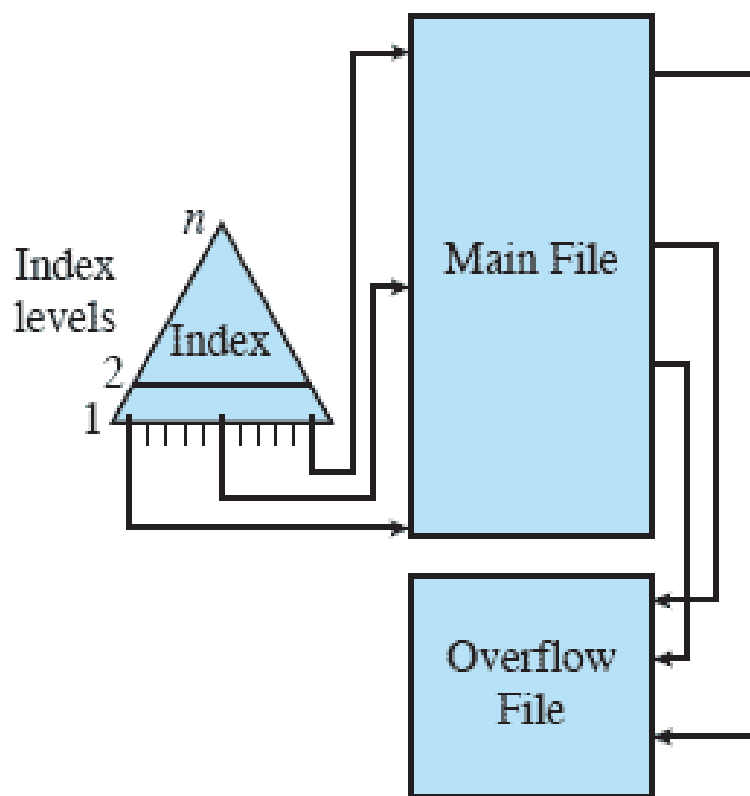


A diagram illustrating a sequential file structure. It consists of a grid with 12 rows and 4 columns. Each row represents a record, and each column represents a field within that record. The grid is empty, showing the uniform structure of sequential files.

- 堆文件（日志文件或事务文件）

- 关键域
- 每条记录都使用一种固定的格式，所有记录都具有相同的长度，并且由相同数目、长度固定的域按特定的顺序组成。
- 通常用于批处理应用中，如果涉及对所有记录的处理，通常是最佳的。唯一可以很容易地存储在磁盘和磁带中的文件组织。
- 对于查询或更新记录的交互式应用，性能很差。

12.2.3 索引顺序文件



(c) Indexed Sequential File

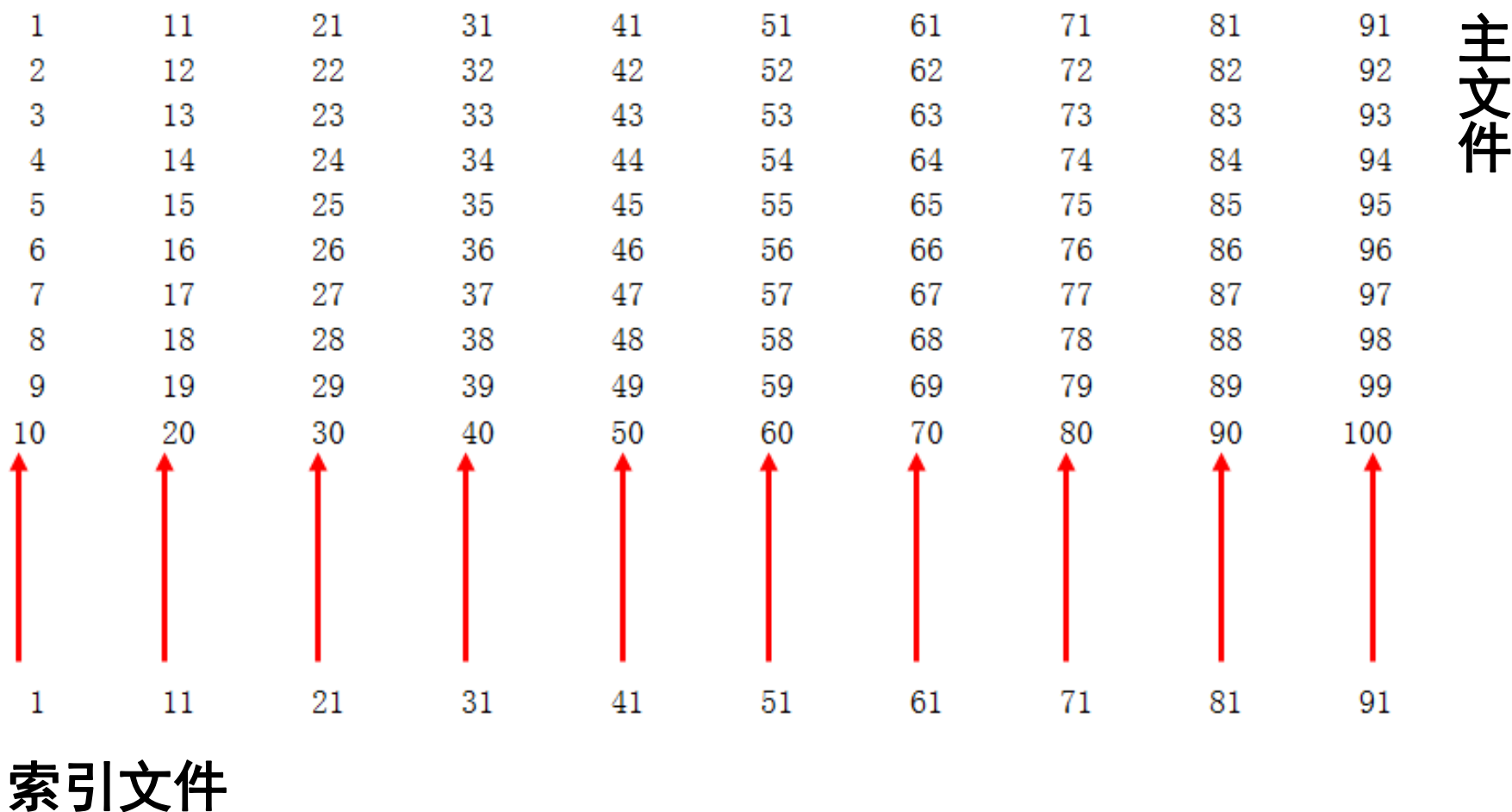
- 保留顺序文件的关键特征：记录按照关键域的顺序组织。增加了两个特征：
 - 文件索引
 - 溢出文件
- 极大地减少了访问单条记录的时间，同时保留了文件的顺序特性。

12.2.3 索引顺序文件

在具有100条记录的顺序文件中查找某个特定的关键域：平均需要访问： $1*1/100+2*(1/100)+\dots+100*(1/100)=50.5$ (50)

$$(1+n) / 2 - n/2$$


12.2.3 索引顺序文件-文件索引



12.2.3 索引顺序文件

在具有100w条记录的顺序文件中查找某个特定的关键域：
平均需要访问：50w次记录

一级索引 访问顺序

	主文件	100w条记录	平均访问次数	
(1)	索引文件	1000项索引	500	在索引文件
				
(2)	每项索引对应	1000条记录	500	在主文件
	总计：		1000	

12.2.3 索引顺序文件

在具有 $100w$ 条记录的顺序文件中查找某个特定的关键域：
平均需要访问： $50w$ 次记录

二级索引

访问顺序

	主文件	100w条记录	平均访问次数	
	低级索引文件	10000项索引		
				
(3)	每项索引对应	100条记录	50	在主文件
	低级索引文件	10000项索引		
(1)	高级索引文件	100项索引	50	在高级索引文件
				
(2)	每项高级索引对应	100项低级索引	50	在低级索引文件

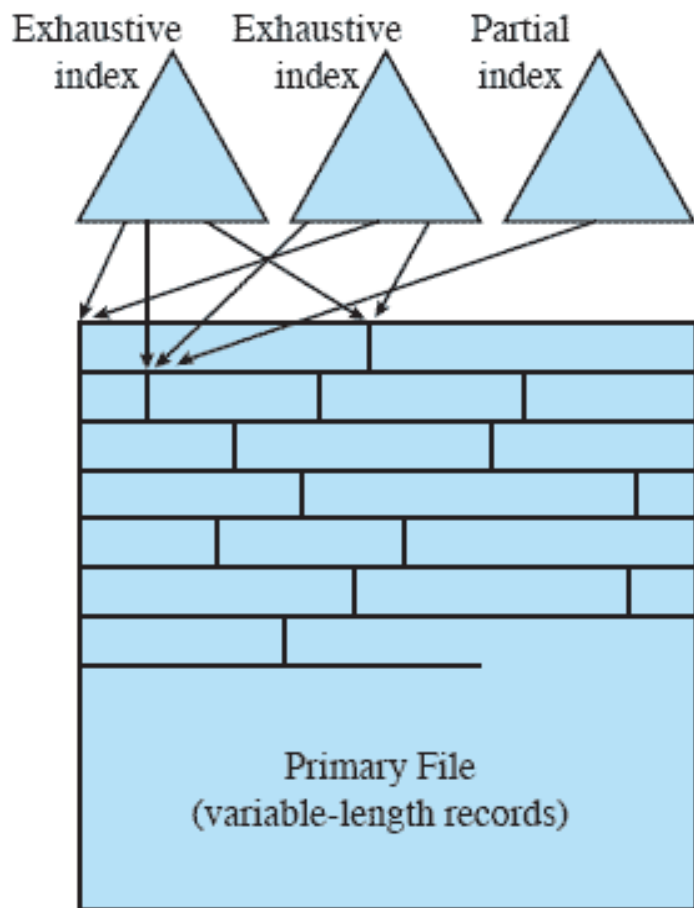
总平均访问次数：150

12.2.3 索引顺序文件-溢出文件

文件可按如下方式处理:主文件中的每条记录都包含一条附加域。附加域对应用程序是不可见的,它是指向溢出文件的一个指针。

- 向文件中插入一条新记录时,它被添加到溢出文件中,然后修改主文件中逻辑顺序位于这条新记录之前的记录,使其包含指向溢出文件中新记录的指针。
 - 如果新记录前面的那条记录也在溢出文件中,那么修改新记录前面的那条记录的指针。
 - 索引顺序文件有时也按批处理的方式合并溢出文件。

12.2.4 索引文件



(d) Indexed File

- 摒弃顺序性和关键字的概念，只能通过索引来访问记录。对记录的放置位置没有限制。
- 大多用于对信息的及时性要求比较严格且很少会对所有数据进行处理的应用程序中。

12.2.5 直接文件或散列文件

- 直接文件：则可根据给定的记录关键值，直接获得对应指定记录的物理地址。
- 散列文件：目前应用最为广泛的一种直接文件。
- 使用基于关键字的散列，能直接访问磁盘中任何一个地址已知的块。
- 通常在要求快速访问时使用，并且记录的长度是固定的，一次只访问一条记录。

12.2.5 直接文件或散列文件

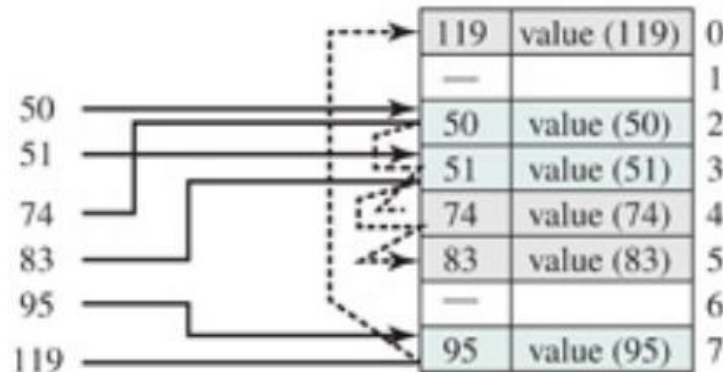
The hashing function can be defined as follows. Assume up to **N items** are to be stored in **a hash table of length M**, with but not much larger than N. **To insert an item** in the table:

(1) Convert **the label** of the item to a near-random **number n** between **0 and M-1**. For example, if the label is numeric, a popular mapping function is to divide the label by M and take the remainder as the value of n. ($n \% M$)

(2) Use **n as the index** into the hash table.

a. If the corresponding entry in the table is empty, store the item (label and value) in that entry.

b. If the entry is already occupied, set **$n = n + 1 \pmod{M}$** and go back to step (2)a.

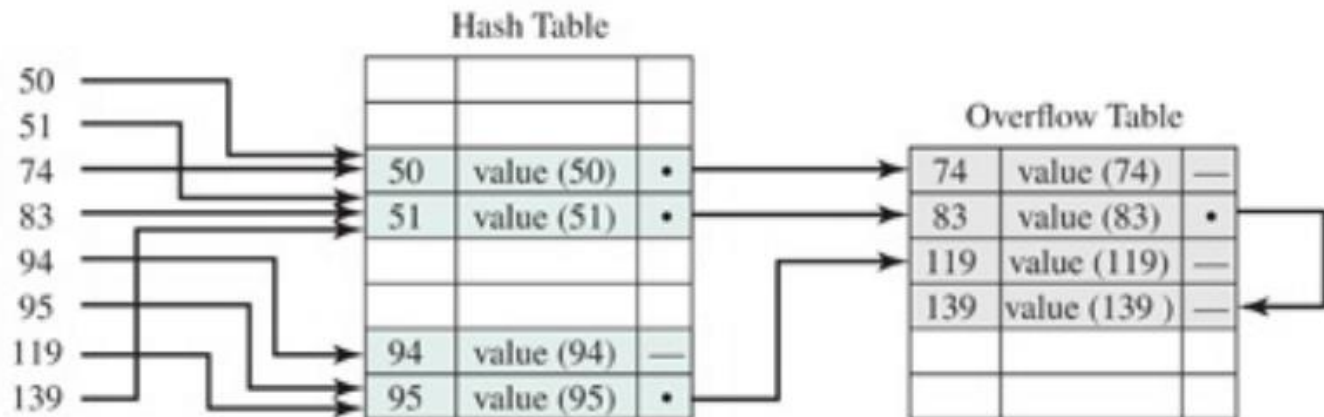


(a) Linear rehashing

12.2.5 直接文件或散列文件

The hashing function can be defined as follows. Assume up to **N items** are to be stored in **a hash table of length M**, with but not much larger than N. **To insert an item** in the table:

- (1) Convert **the label** of the item to a near-random **number n** between **0 and M-1**. For example, if the label is numeric, a popular mapping function is to divide the label by M and take the remainder as the value of n. ($n \% M$)
- (2) Use **n as the index** into the hash table.
 - a. If the corresponding entry in the table is empty, store the item (label and value) in that entry.
 - b. If the entry is already occupied, then store the item in an overflow area.



(b) Overflow with chaining

12.3 B树 ★★☆☆☆

B树是具有以下特征的一种树状结构：

- 一个包含若干**节点和叶子**的树；
- 每个节点至少包含一个用来唯一标识文件记录的关键码，并且包含多于一个的指向子节点或叶子的指针。一个节点包含的关键码和指针的数目是可变的，相关限制如下：
 - 每个节点的关键码数量不能超过一个特定的最大值；
 - 每个节点的关键码按照非减次序来存储。每个关键码对应一个子节点，以该子节点为根的子树包含所有关键码均小于等于当前的关键码并且大于前一个关键码的节点。一个节点包含一个额外的最右子节点，以该节点为根的子树所包含的所有关键码，都大于该节点包含的任意关键码。

最小度数为 d 的B树要满足的性质

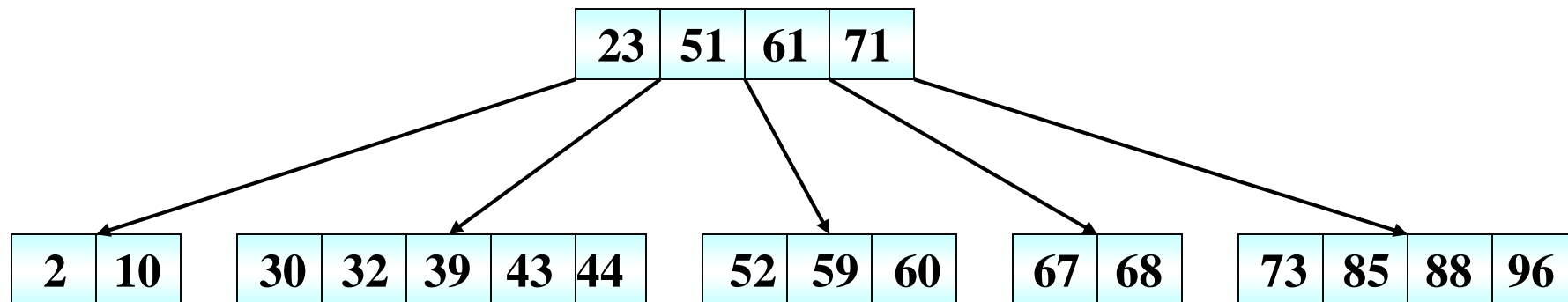
- 每个节点最多有 $2d-1$ 个关键码和 $2d$ 个子女。
- 除根节点外，每个节点都至少有 $d-1$ 个关键码和 d 个指针。
- 根节点最少有1个关键码和2个子女。
- 所有的叶子都在同一层，不包含任何信息。
- 一个包含 k 个指针的非叶子节点有 $k-1$ 个关键码。

插入一个新的关键码

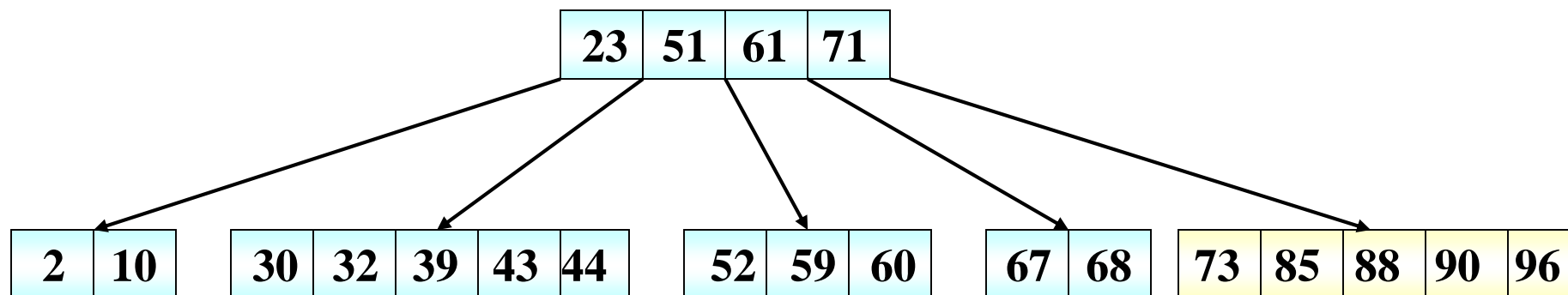
1. 在树中搜索这个关键码。如果该关键码不在树中，那么会到达最底层的一个节点。
2. 如果这个节点的关键码少于 $2d-1$ ，那么把新的关键码按照适当的顺序插入该节点。
3. 如果这个节点是满的（包含 $2d-1$ 个关键码），那么以这个节点的中间的关键码为界把该节点分裂为两个新的节点，每个新节点都包含 $d-1$ 个关键码，并且把中间的关键码提升到上一层，提升过程见步骤4。如果新的关键码的值小于中间的关键码，则把它插入左边的新节点，否则插入右边的新节点。

-
4. 提升的节点按步骤3的规则插入到父节点中。因此，如果父节点已满，它必须被分裂，并且它的中间的关键码被提升到上一层。
 5. 如果提升的过程到达了根节点并且根节点是满的，那么按步骤3的规则插入。这样，中间的关键码变成了一个新的根节点并且树的高度增加1

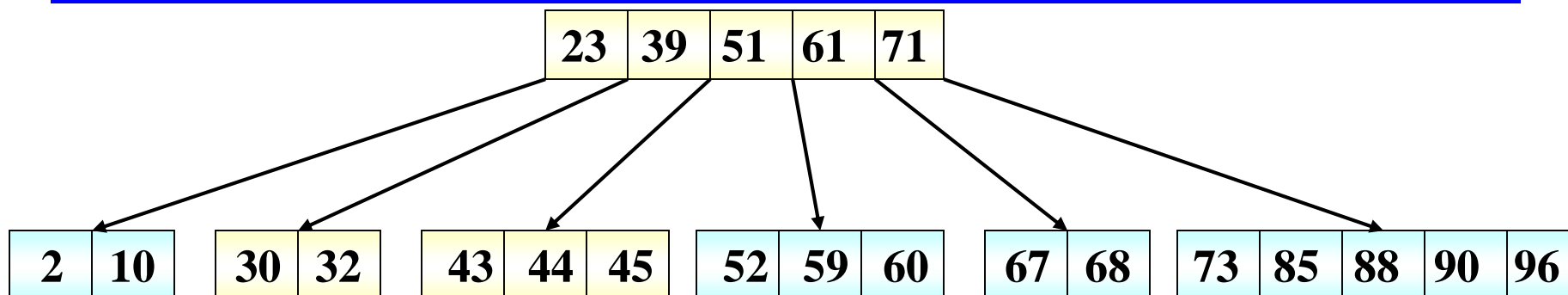
a. 最小度数 $d=3$ 的B树



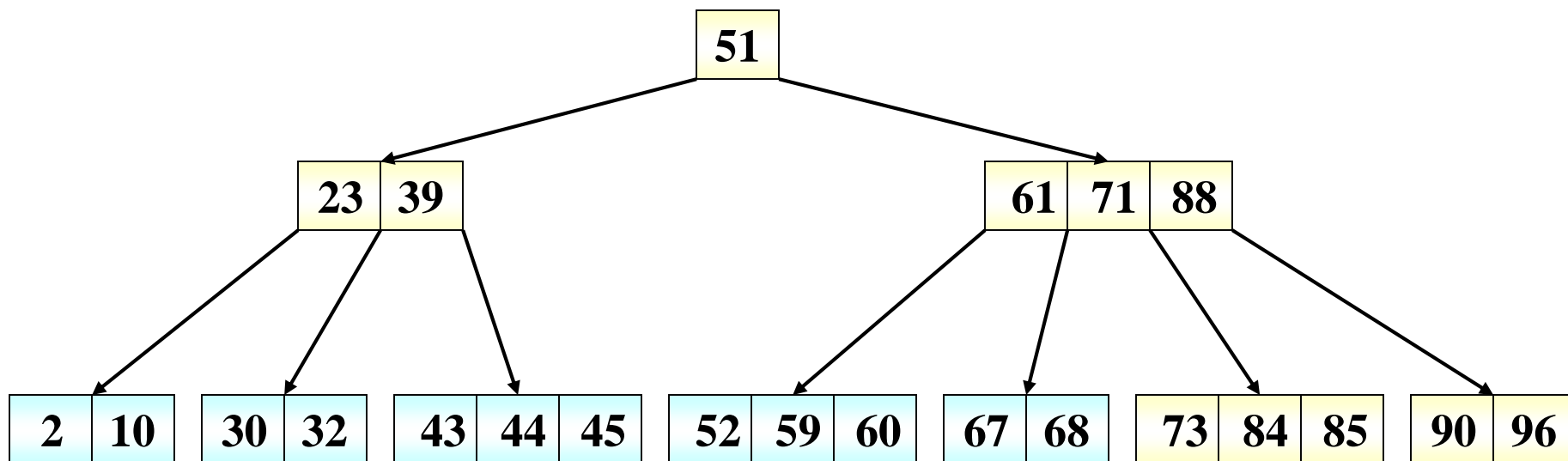
b. 插入关键码 $\text{key}=90$



c. 插入关键码key=45



d. 插入关键码key=84



P316 课本的块的执行时间实际是指：将一个块读到内存，并执行它的时间

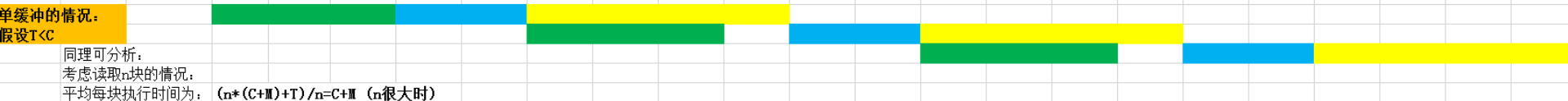
T: 将一个磁带块传到内存的时间
C: 收到磁带块后（此时磁带块在内存）执行该块的时间
M: 将磁带块从内核空间传到用户空间的时间

没有缓冲的情况: 每块的执行时间: $T+C$



考虑读取 n 块的情况:
平均每块执行时间为: $(n * (T+M)+C) / n = T+M$ (n 很大时)

CPU在运行的时候可以将一个块从磁带传到内核空间拷贝 (CPU与IO并行工作)



综上: 平均每块执行时间为: $\max(T, C)+M$

12.4 文件目录 ★★☆☆☆

12.4.1 内容（目录项、文件控制块（FCB））

- 基本信息
 - 文件名
 - 文件类型
 - 文件组织
- 地址信息
 - 卷
 - 起始地址
 - 使用大小
 - 分配大小

-
- 访问控制信息
 - 所有者
 - 访问信息
 - 许可的行为
 - 使用信息
 - 数据创建
 - 创建者身份
 - 最后一次读访问的日期
 - 最后一次读用户的身份
 - 最后一次修改的日期
 - 最后一次修改者的身份
 - 最后一次备份的日期
 - 当前使用

12.4.2 结构 ★★☆☆☆

- 层次或树型结构是普遍采用的一种方法。

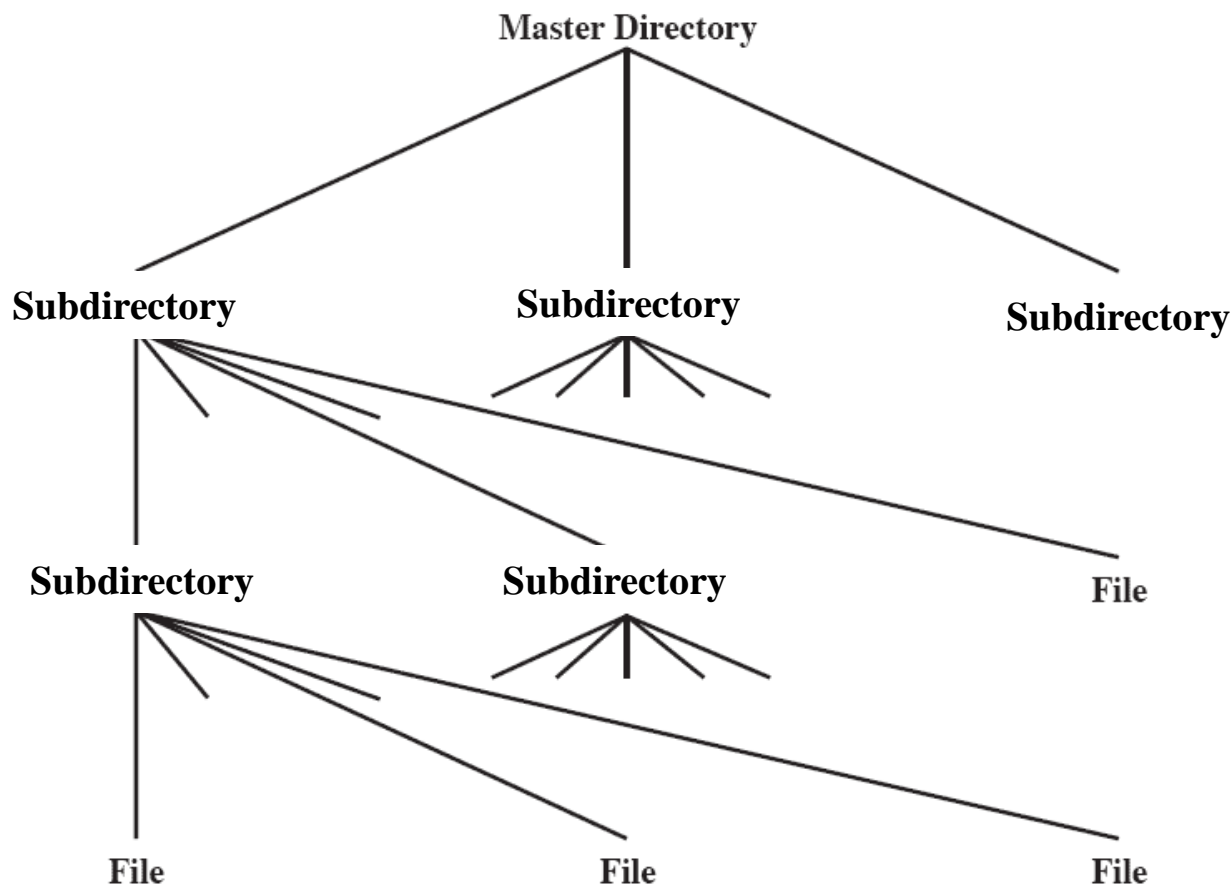


Figure 12.4 Tree-Structured Directory

12.4.3 命名 ★★☆☆☆

- 文件命名唯一性问题

- 路径名：系统中的任何文件都可以按照从根目录或主目录向下到各个分支，最后直到该文件的路径来定位。这一系列目录名和最后到达的文件名组成了该文件的路径名(pathname)。
- 工作目录：必须拼写出完整的路径名仍比较困难。典型情况下，对交互用户或进程而言，总有一个当前路径与之相关联，通常称为工作目录(working directory)。

12.4.3 命名 ★★☆☆☆

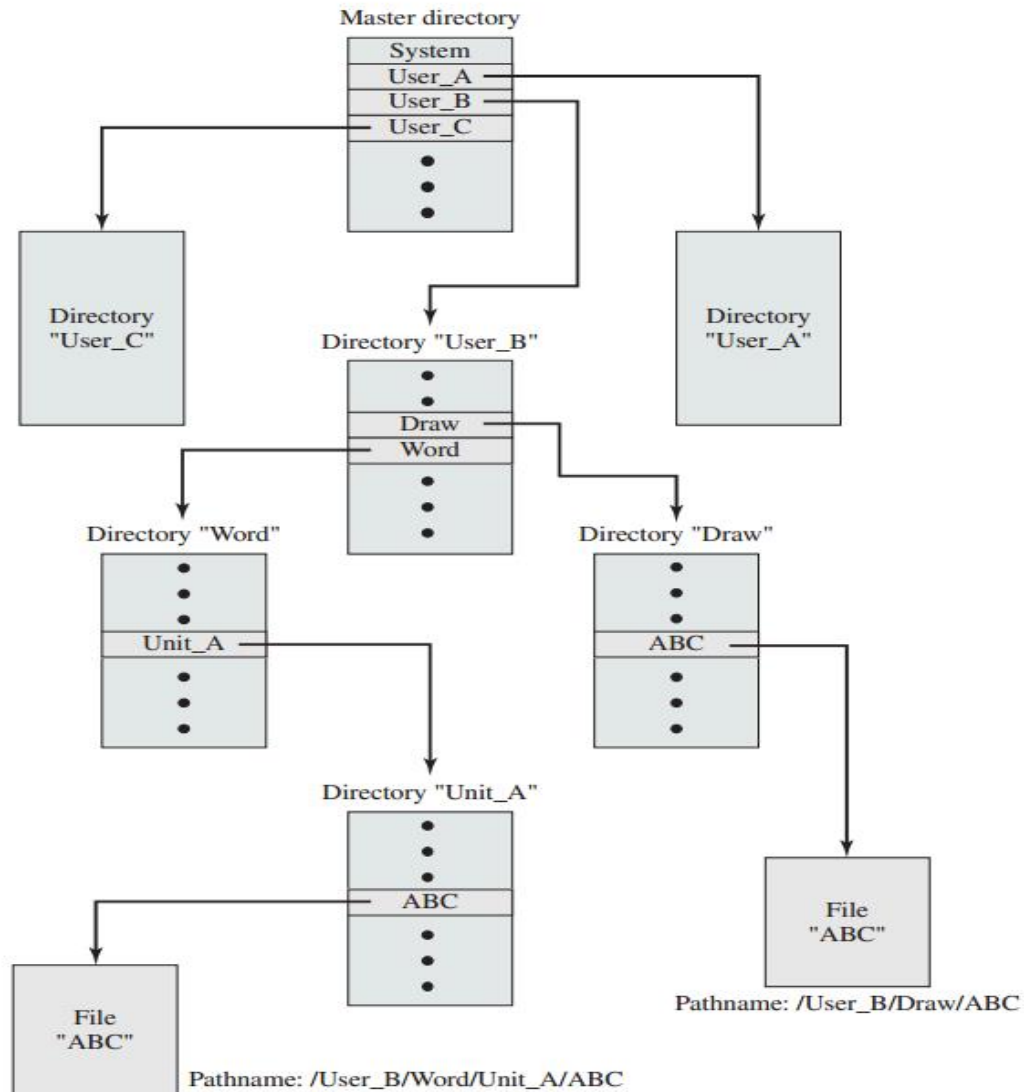


Figure 12.7 Example of Tree-Structured Directory

12.5 文件共享 ★★☆☆☆

- 访问权限

- 无
- 知道
- 执行
- 读
- 追加
- 更新
- 改变保护
- 删除

- 同时访问

- 加锁
- 互斥和死锁问题

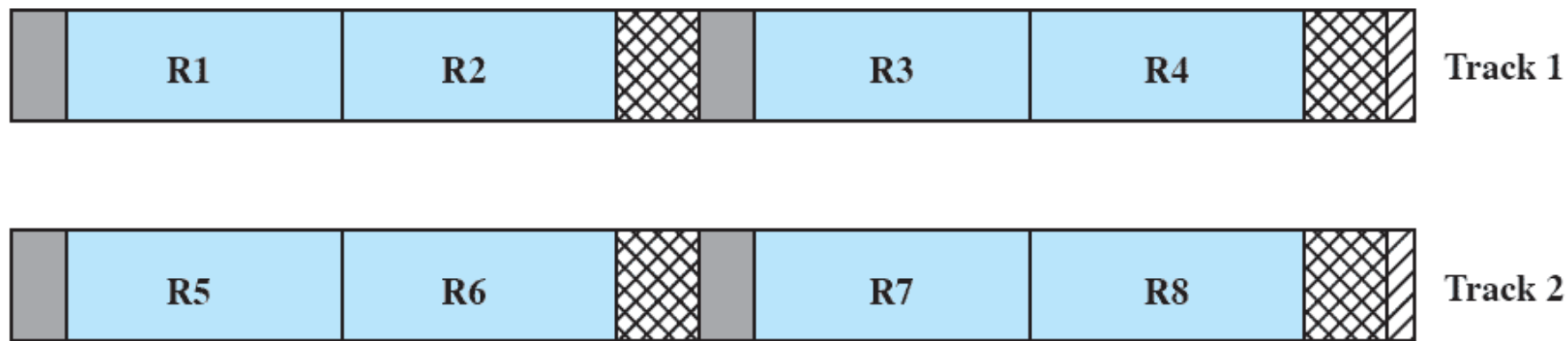
12.6 记录组块 ★★☆☆☆

1、需考虑的问题：

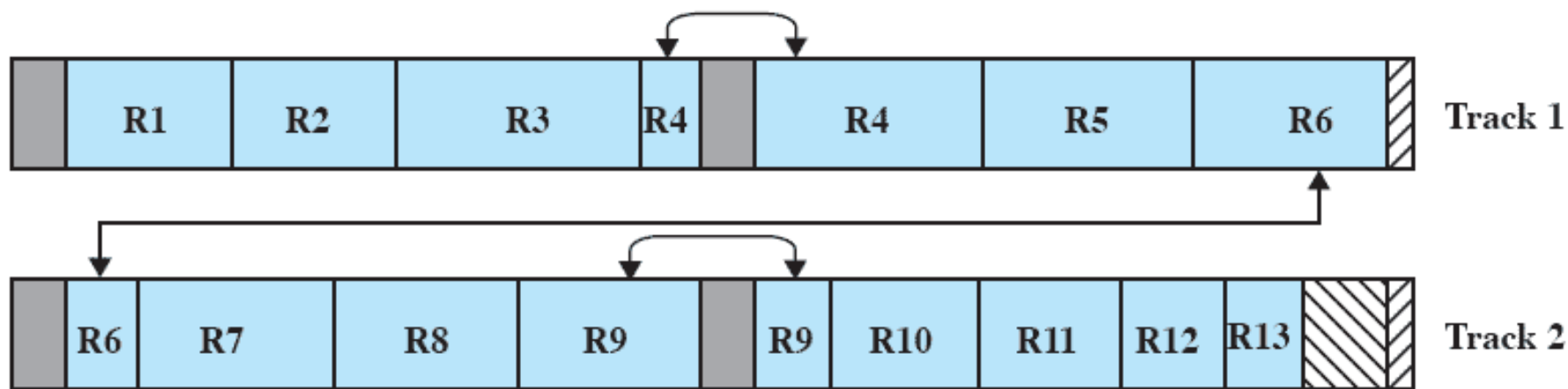
- 块的长度是固定的还是可变的？
 - 大多数系统中，块是固定长度的。
- 与记录的平均大小相比，块的相对大小是多少？
 - 综合考虑顺序访问的频率和访问的局部性潜能，倾向于用大的块，以减少I/O传送时间。

2、记录组块的方法

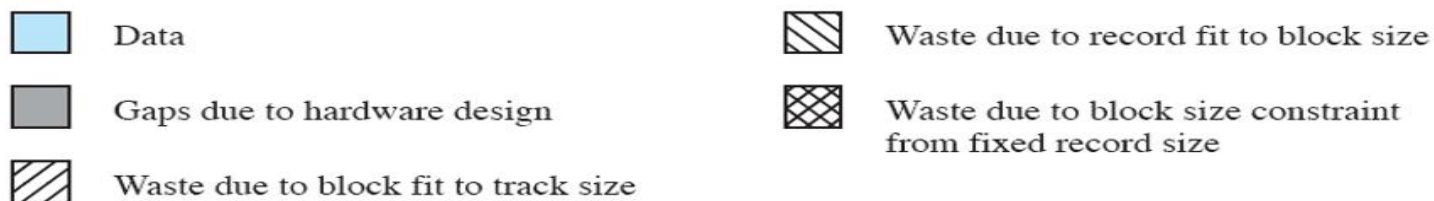
- 定长组块
- 变长度跨越式组块
- 变长度非跨越式组块

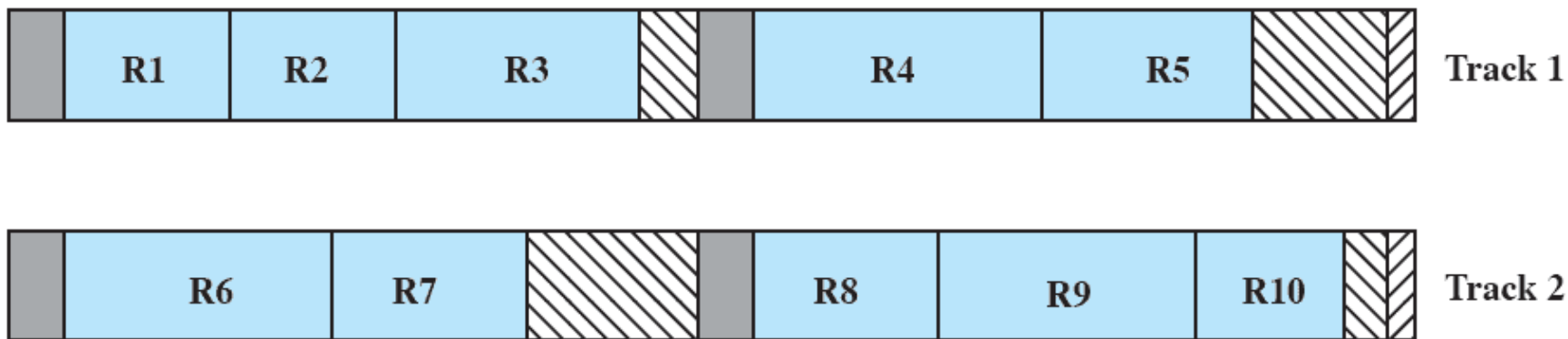


Fixed Blocking

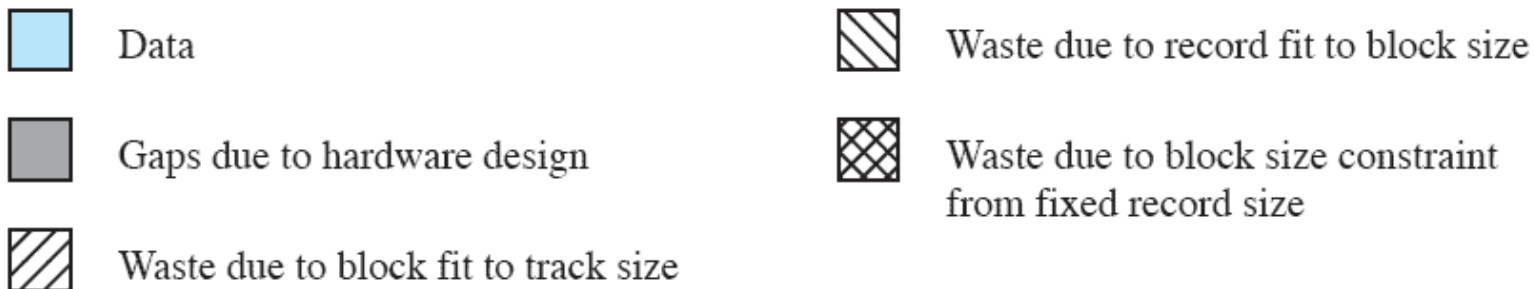


Variable Blocking: Spanned





Variable Blocking: Unspanned



12.7 辅助存储管理 ★★★★★

12.7.1 文件分配

- 文件分配涉及到的问题：
 - 当创建一个新文件时，是否一次性地分配所需要的最大空间？
 - ☒ 预分配与动态分配
 - 在分配文件时，分区的大小应该是多少？
 - ☒ 可变的大规模连续分区
 - ☒ 块
 - 为跟踪分配给文件的分区，应该使用哪种数据结构或表？
 - ☒ 文件分配表

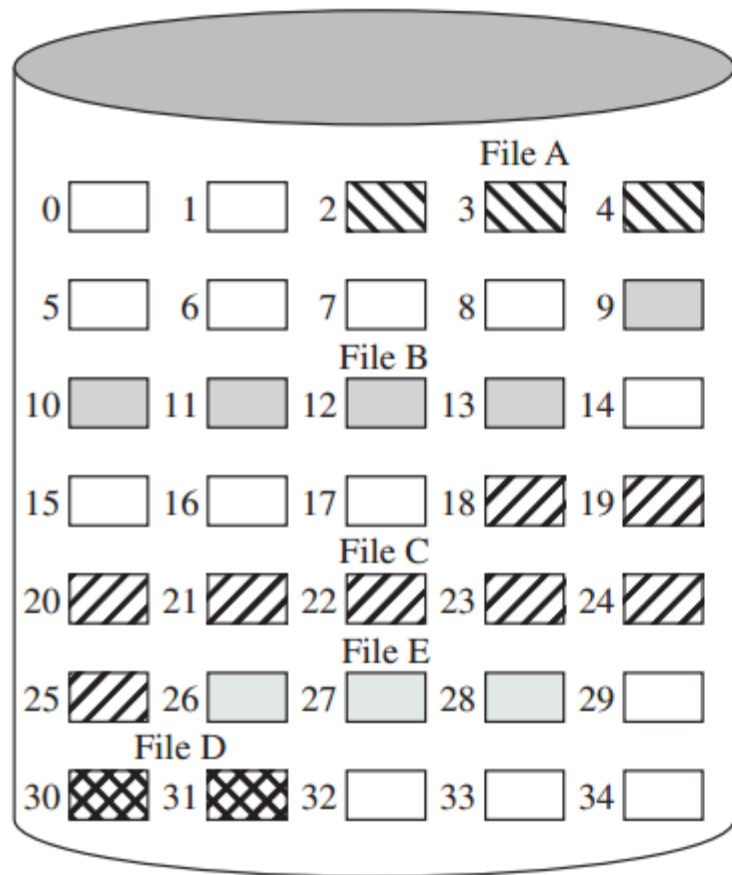
1、预分配与动态分配

- 预分配策略要求在发出创建文件的请求时，声明该文件的最大大小。
 - 若不能可靠地估计文件可能的最大大小，通常会多估计一些，以避免分配的空间不够。——浪费
- 动态分配只有在需要时才给文件分配空间。

2、分区大小

- 选择一个分区大小时，需要折中考虑单个文件的效率和整个系统的效率。
 - 邻近空间可以提高性能；
 - 数目较多的小分区会增加用于管理分配信息表的大小；
 - 使用固定大小的分区可以简化空间的再分配；
 - 使用可变大小的分区或固定大小的小分区，可以减少由于超额分配而产生的未使用存储空间的浪费。
- 综合考虑的两种选择：
 - 可变的大规模连续分区
 - ☒ 大小可变避免了浪费，文件分配表比较小
 - ☒ 空间很难再次利用
 - 块
 - ☒ 小的固定分区提供了更多的灵活性
 - ☒ 可能需要较大的表或更复杂的结构

3、文件的分配方法—（1）连续分配



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

- ✓ 预分配
- ✓ 大小可变分区
- ✓ 对单个顺序文件，连续分配最好
- ✓ 一个表项
- ✓ 外部碎片

Figure 12.9 Contiguous File Allocation

3、文件的分配方法—（1）连续分配

优点

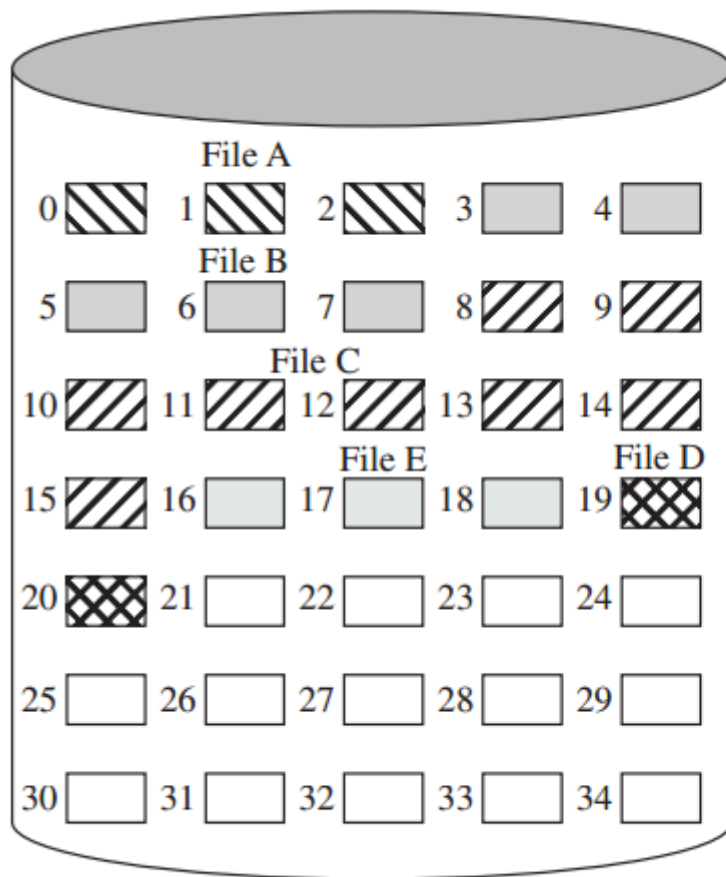
- 简单
- 支持顺序存取和随机存取
- 所需的磁盘寻道次数和寻道时间最少
- 可以同时读入多个块，检索一个块也很容易

缺点

- 文件不能动态增长（预留空间:浪费 或 重新分配和移动）
- 不利于文件插入和删除
- 外部碎片:紧缩技术

紧缩算法

3、文件的分配方法—（1）连续分配



File allocation table		
File name	Start block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.10 Contiguous File Allocation (After Compaction)

3、文件的分配方法—（2）链式分配

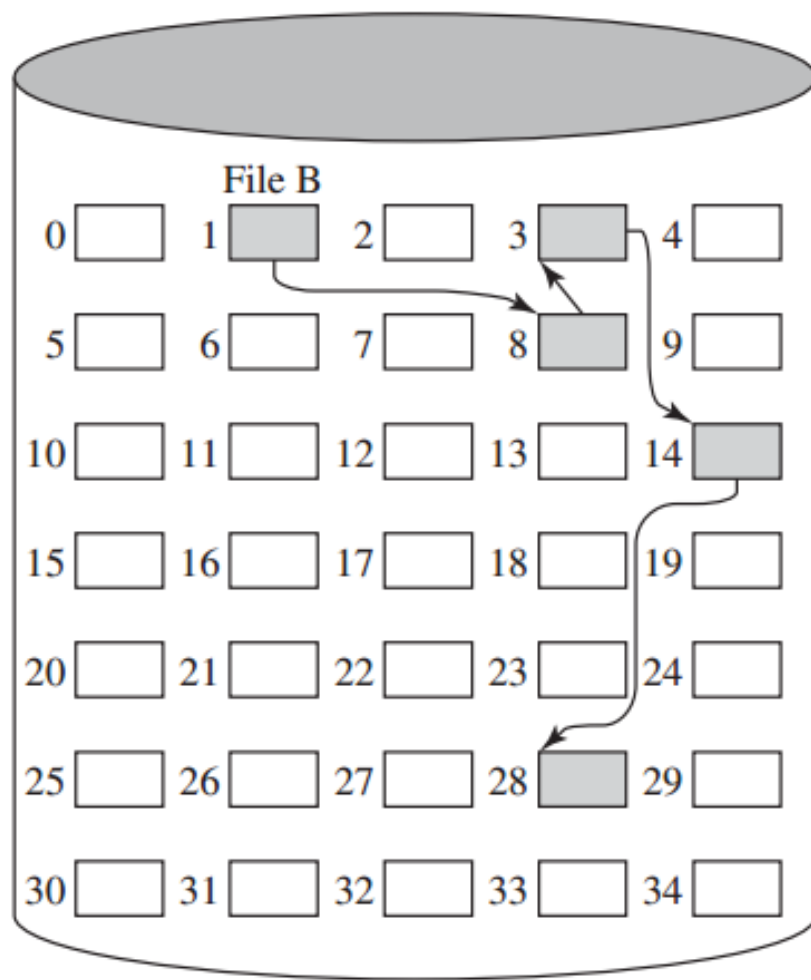


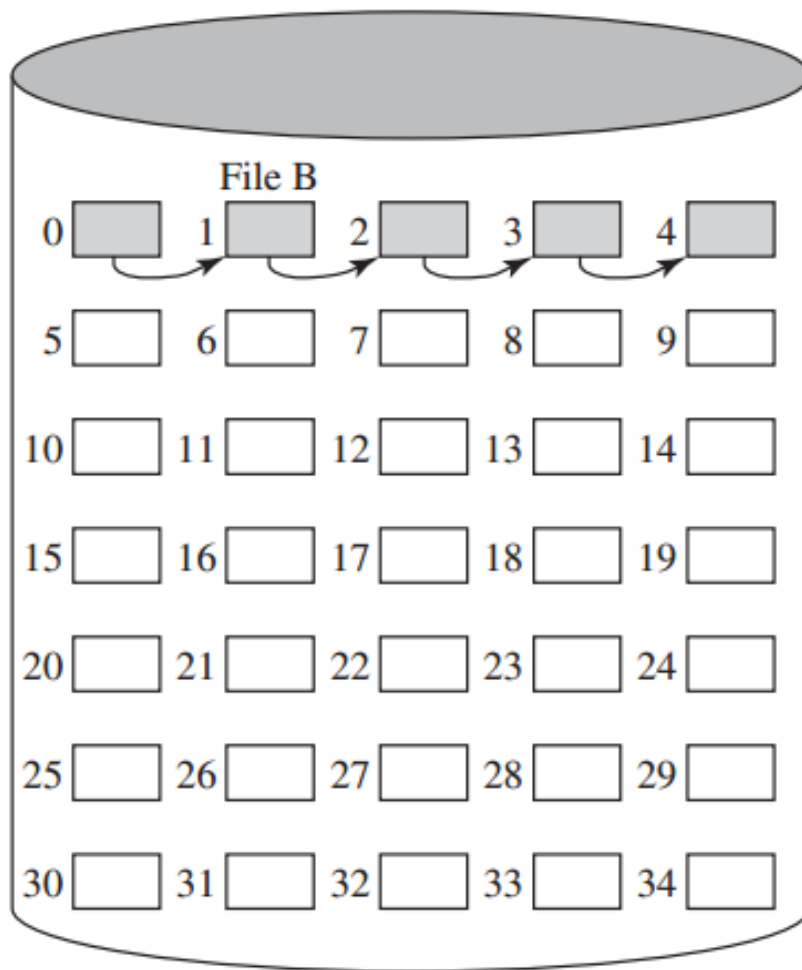
Figure 12.11 Chained Allocation

File allocation table		
File name	Start block	Length
...
File B	1	5
...

- ✓ 预分配或动态分配
- ✓ 没有外部碎片
- ✓ 一个表项
- ✓ 局部性原理不再适用

周期合并文件

3、文件的分配方法—（2）链式分配



File allocation table

File name	Start block	Length
...
File B	0	5
...

Figure 12.12 Chained Allocation (After Consolidation)

3、文件的分配方法—（2）链式分配

优点

- 提高了磁盘空间利用率，不存在外部碎片问
- 有利于文件插入和删除
- 有利于文件动态扩充

缺点

- 存取速度慢，不适于随机存取
- 可靠性问题，如指针出错
- 更多的寻道次数和寻道时间
- 链接指针占用一定的空间

3、文件的分配方法—（3）索引分配

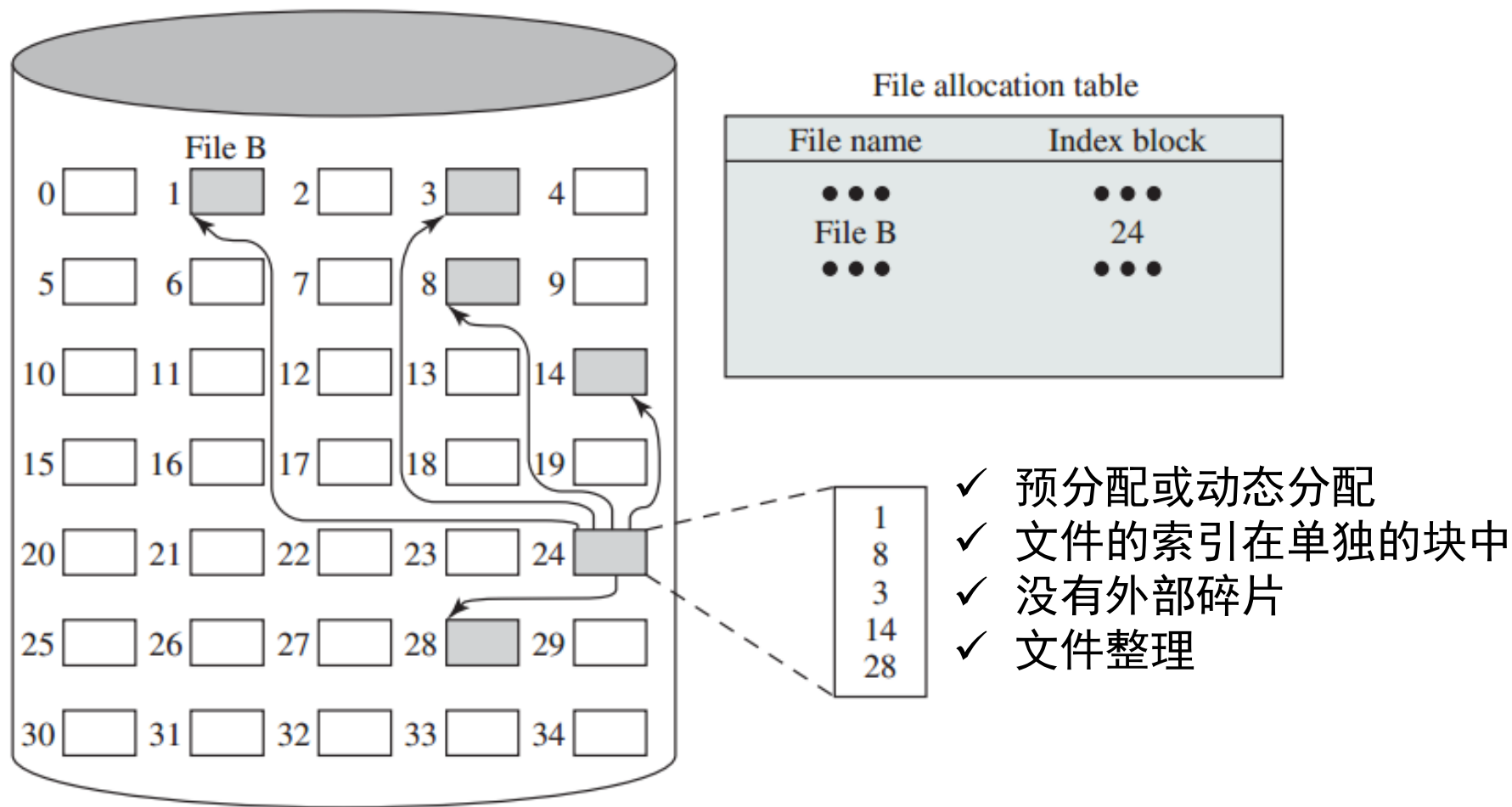


Figure 12.13 Indexed Allocation with Block Portions

基于块的索引分配

3、文件的分配方法—（3）索引分配

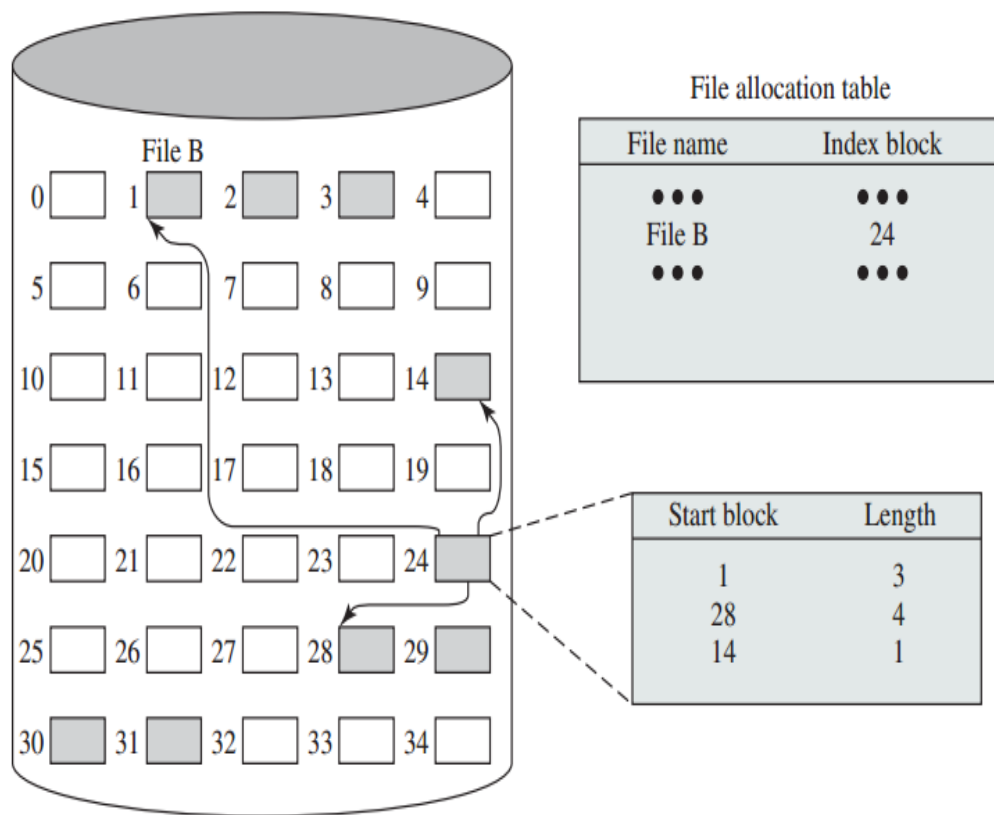


Figure 12.14 Indexed Allocation with Variable-Length Portions

基于长度可变分区的索引分配

- ✓ 预分配或动态分配
- ✓ 文件的索引在单独的块中
- ✓ 提高局部性
- ✓ 文件整理

3、文件的分配方法—（3）索引分配

优点

- 保持了链接结构的优点，又解决了其缺点
- 既能顺序存取，又能随机存取
- 满足了文件动态增长、插入删除的要求
- 能充分利用磁盘空间

缺点

- 较多的寻道次数和寻道时间
- 索引表本身带来了系统开销
- 如：内存、磁盘空间，存取时间

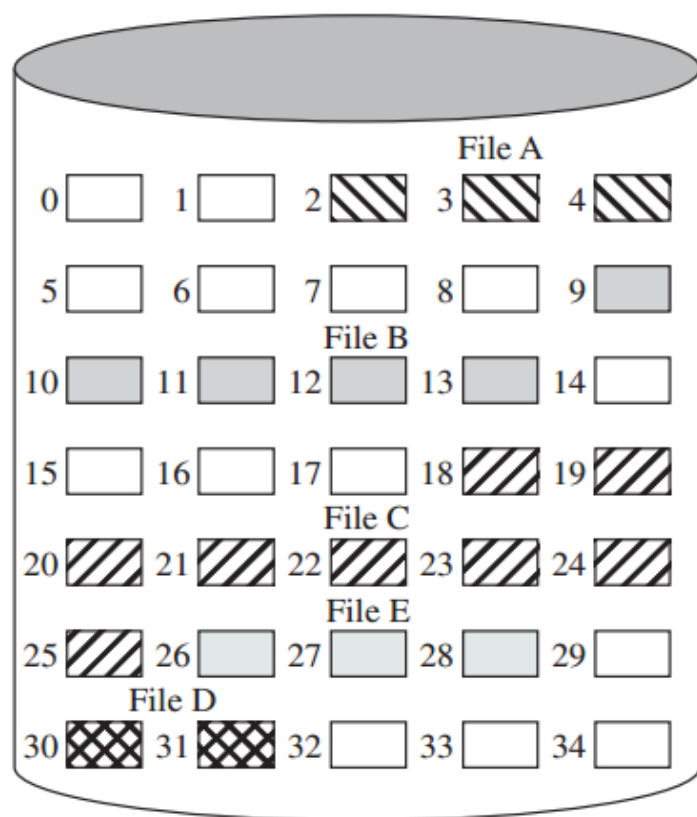
12.7.2 空闲空间的管理

- 磁盘分配表
- 常用的空闲空间管理技术
 - 位表
 - 链接空闲区
 - 索引
 - 空闲块列表
- 卷
 - 一组在辅助存储上可寻址的扇区的集合，操作系统或应用程序用卷来进行数据存储。一个卷中的扇区在物理存储设备上不需要是连续的，只需要对操作系统或应用程序来讲是连续的。一个卷可能是更小的卷合并或组合后的结果。

12.7.2 空闲空间的管理

- 常用的空闲空间管理技术

- 位表**: 这种方法使用一个向量，向量的每一位对应于磁盘中的每一块。0表示空闲块，1表示已使用块。



一个块位图所需的存储容量=
磁盘大小 / (8*块大小)

- 位表的存储：内存？磁盘？
- 划分位表子区域，记于汇总表中

00111000001111100001111111111111011000

Figure 12.9 Contiguous File Allocation

12.7.2 空闲空间的管理

- 常用的空闲空间管理技术

- **链接空闲区**：使用指向每个空闲区的指针和它们的长度值，可将空闲区链接在一起

- 该方法适用于所有的文件分配方法。

- 如果一次只分配一块，那么只要简单地选择链头上的空闲块，并调整第一个指针或长度值即可。

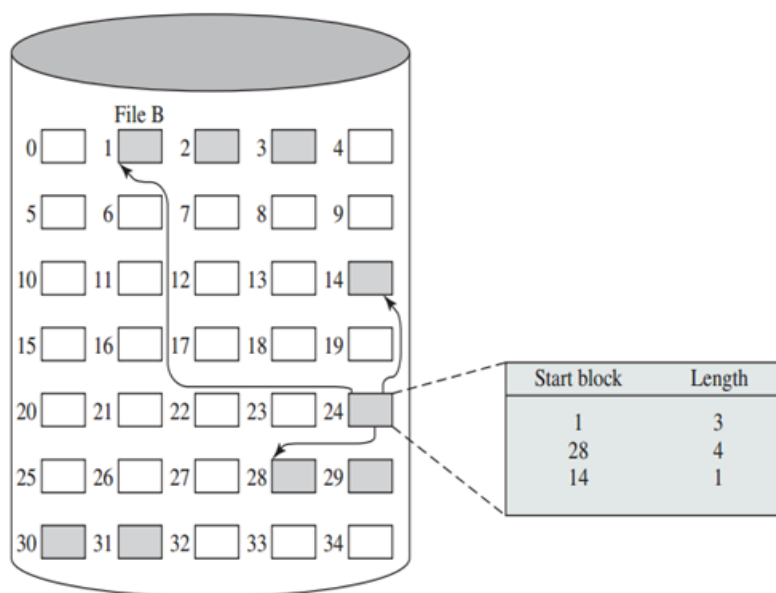
- 如果基于可变分区进行分配，那么可以使用首次适配算法：从头开始取分区，一次取一个，以确定链表中下一个适合的空闲块。这时，同样需要调整指针和长度。

- 使用一段时间后，磁盘会出现很多碎片，许多分区都会变得只有一个块那么长。需要为一个文件操作同时分配许多块时，会大大降低创建文件的速度。与此类似，删除一个由许多碎片组成的文件也非常耗时。

12.7.2 空闲空间的管理

- 常用的空闲空间管理技术

- 索引**：把空闲空间视为一个文件，并使用一个在文件分配时介绍过的索引表。基于效率方面的考虑，索引应该基于可变大小的分区而非块。因此，磁盘中的每个空闲分区在表中都有一个表项。该方法能为所有的文件分配方法提供有效的支持。



12.7.2 空闲空间的管理

- 常用的空闲空间管理技术

- **空闲块列表**：每块都指定一个序号，所有空闲块的序号保存在磁盘的一个保留区中。根据磁盘的大小，存储一个块号需要24位或32位，故空闲块列表的大小是24或32乘以相应的位表大小，因此它须保存在磁盘而非内存中。

- 磁盘上用于空闲块列表的空间小于磁盘空间的1%。
 - 小部分保存在内存：
 - 下推栈
 - FIFO队列

12.8 UNIX文件管理 ★★☆☆☆

- UNIX区分六种类型的文件
 - 普通文件
 - 目录文件
 - 特殊文件
 - 命名管道
 - 链接文件
 - 符号链接

12.8 UNIX文件管理 ★★☆☆☆

- UNIX区分六种类型的文件

- 普通文件

- ☒ 文件中包含的信息是由用户、应用程序或系统实用程序输入的。文件系统在普通文件上不强加任何内部结构，只把它们视为字节流。

- 目录文件

- ☒ 包含文件名列表和指向与之相关联的索引节点(index node)的指针。目录是按层次结构组织的。目录文件实际上是具有特殊写保护权限的普通文件，只有文件系统才能对它进行写操作，但允许所有用户程序对它进行读访问。

12.8 UNIX文件管理 ★★☆☆☆

- UNIX区分六种类型的文件（续）

- 特殊文件

- ☒ 不包含数据，但提供一个将物理设备映射到一个文件名的机制。文件名用于访问外围设备，如终端和打印机。每个I/O设备都有一个特殊文件与之相关联。

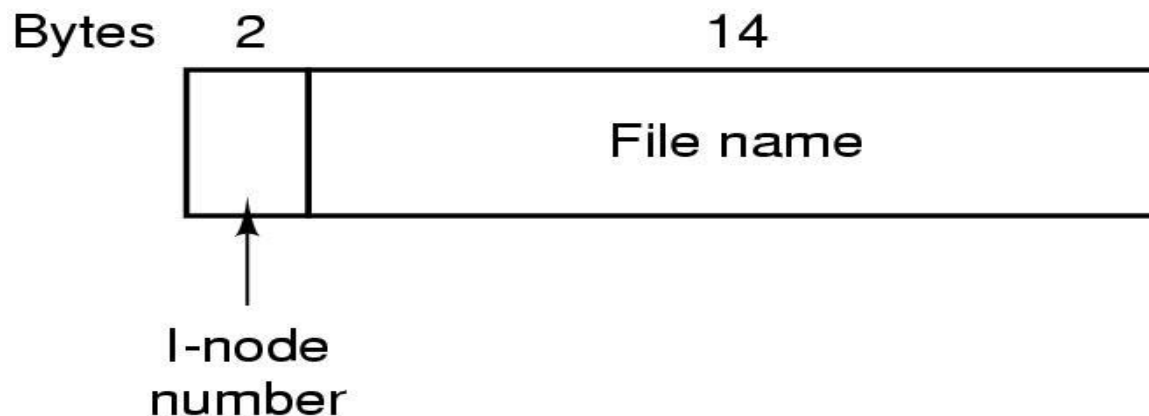
- 命名管道

- ☒ 管道是进程间通信的基础设施。管道缓存输入端接收的数据，以便在管道输出端读数据的进程能以先进先出的方式接收数据。

12.8 UNIX文件管理 ★★☆☆☆

- UNIX区分六种类型的文件（续）
 - 链接文件
 - ☒ 链接是一个已有文件的另一个可选文件名。
 - 符号链接
 - ☒ 是一个数据文件，它包含了其所链接的文件的文件名。

12.8.1 索引节点



目录文件的内容

文件名 14B	i-node # 2B
.....
mytest.c	56
.....

磁盘上的i-node表

i-node #	文件属性
56	存取权限, 大小, 文件主, 建立/修改日期, 磁盘地址 等
.....

64B

12.8.1 索引节点

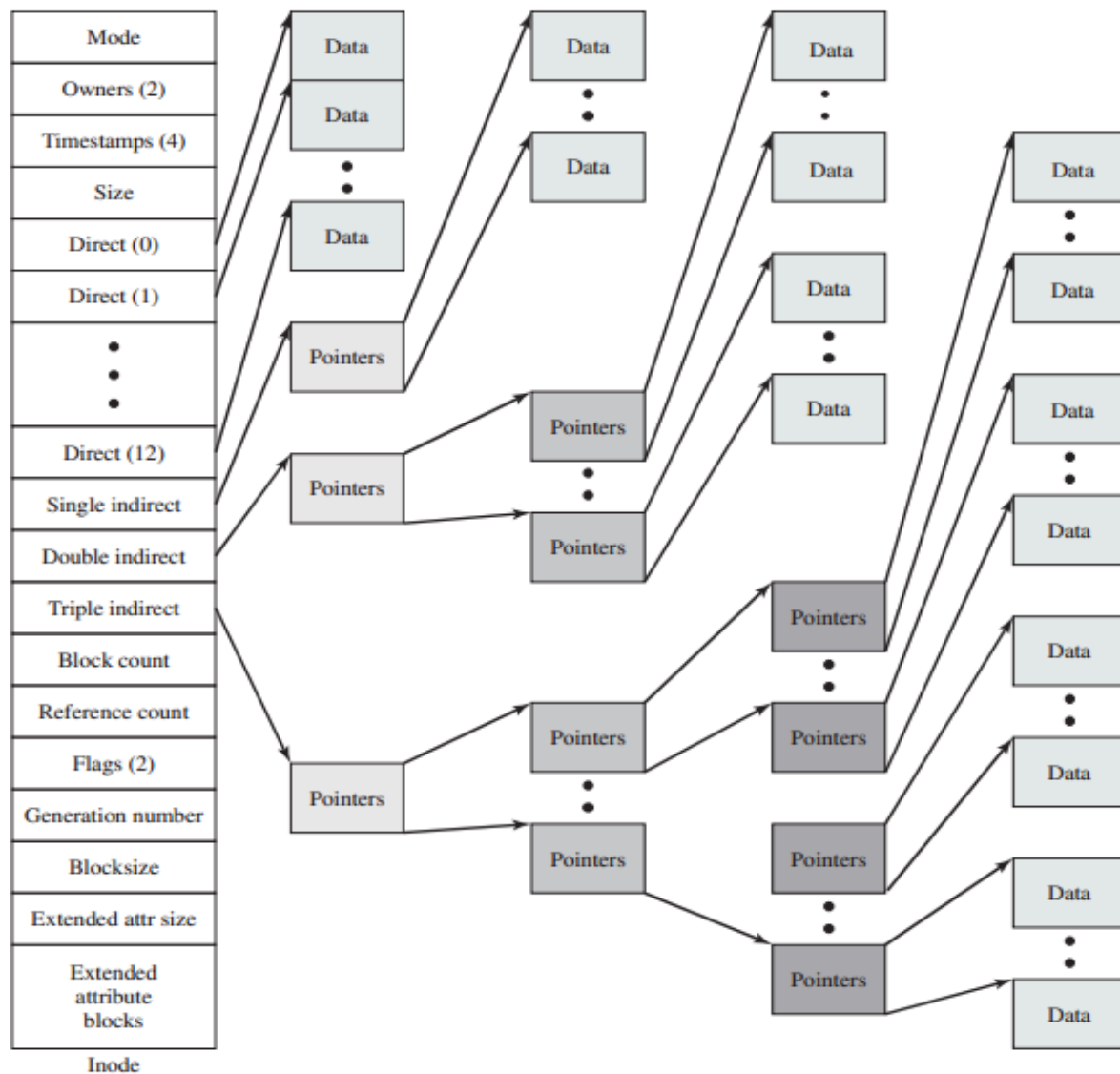


Figure 12.16 Structure of FreeBSD Inode and File

思考：UNIX使用i-node的好处是什么？

- 因为按文件名检索目录文件时，只用到了文件名。当找到该文件名时，才需要它的其它描述信息。所以在把存放该目录文件的盘块从外存调入内存进行比较时，应使一个盘块中包含尽量多的文件名，以**减少启动磁盘次数**，加快按名存取的速度。所以引入索引节点。
- ✓ 例：设物理块大小为512B，某目录下有128个文件。
 - ❑ 原来的FCB占64B，则每物理块能容纳 $512/64=8$ 个FCB，则该目录文件需占 $128/8 = 16$ 块，查找一个文件的平均访盘次数为： $(1+16) / 2 = 8.5$ 次。
 - ❑ 采用i-node后：文件名部分有16B，i-node部分有64B，每物理块能容纳 $512/16=32$ 个文件名部分或 $512/64=8$ 个i-node，则该目录的文件名部分需占 $128/32 = 4$ 块，i-node部分需占 $128/8=16$ 块。查找一个文件的平均访盘次数为： $(1+4) / 2 + 1 = 3.5$ 次。

12.8.2 文件分配

- 文件分配是以块为基础完成的。分配按照需要**动态**地进行，而非预定义分配。因此，文件在磁盘中的块并不需要一定是连续的。
- 系统为了知道每个文件，采用一种索引方法，索引的一部分保存在该文件的索引节点中。在所有的UNIX实现中，索引节点都包含一些直接指针和三个间接指针(一级、二级、三级)。

12.8.2 文件分配

- FreeBSD索引节点包含120字节的地址信息，这些信息通常被组织为15个64位的地址或指针。前12个地址指向文件的前12个数据块，如果文件需要多于12个数据块，那么一级或多级间接寻址。
- 在FreeBSD系统中，最小的块大小是4KB,且每块最多保存512个块地址。
因此在该方案下，文件最大可以超过500GB。

Table 12.4 Capacity of a FreeBSD File with 4-Kbyte Block Size

Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	512	2M
Double Indirect	$512 \times 512 = 256K$	1G
Triple Indirect	$512 \times 256K = 128M$	512G

12.8.3 目录

• 树型目录

- 目录以层次树的形式组织。每个目录都可包含文件和其他目录。位于另一个目录中的目录称为子目录。
- 目录是包含文件名列表和指向相关索引节点的指针的文件。
- 每个目录项都包含一个相关的文件名或目录名和称为索引节点号的整数。
- 访问文件或目录时，其索引节点号会用做索引节点表的索引。

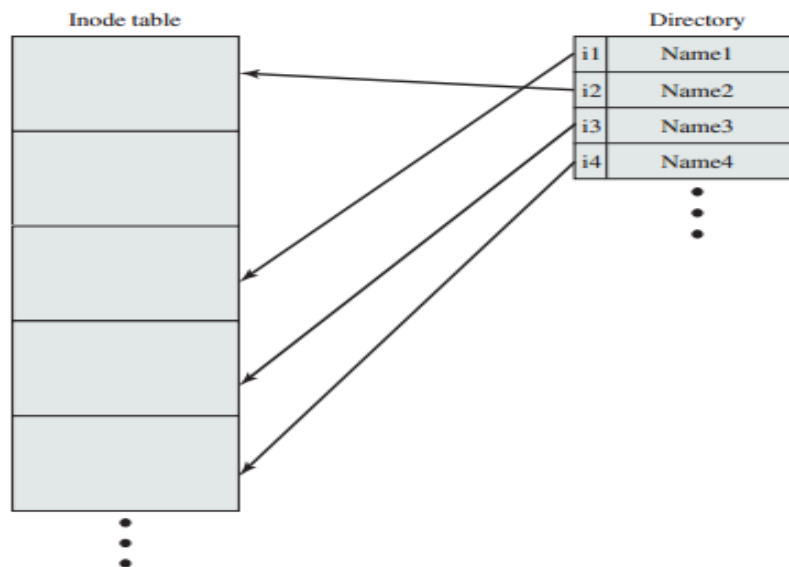


Figure 12.17 UNIX Directories and Inodes

12.8.3 卷结构

- 一个UNIX文件系统驻留在一个单一逻辑磁盘或磁盘分区上，包含以下元素：
 - 引导块: 包含引导操作系统的代码。
 - 超级块: 包含有关文件系统的属性和信息，如分区大小和索引节点表大小。
 - 索引节点表: 系统中所有文件的索引节点集。
 - 数据块: 数据文件和子目录所需的存储空间。

作业

- 复习题 12. 11
- 习题 12. 3