# File Management

## Chapter 12

# Files

- Used for:
  - input to a program
  - Program output saved for long-term storage

# Terms Used with Files

- ✓ *Field*
  - basic element of data
  - contains a single value
  - characterized by its length and data type
- ✓ *Record*
  - collection of related fields
  - treated as a unit
    - Example: employee record

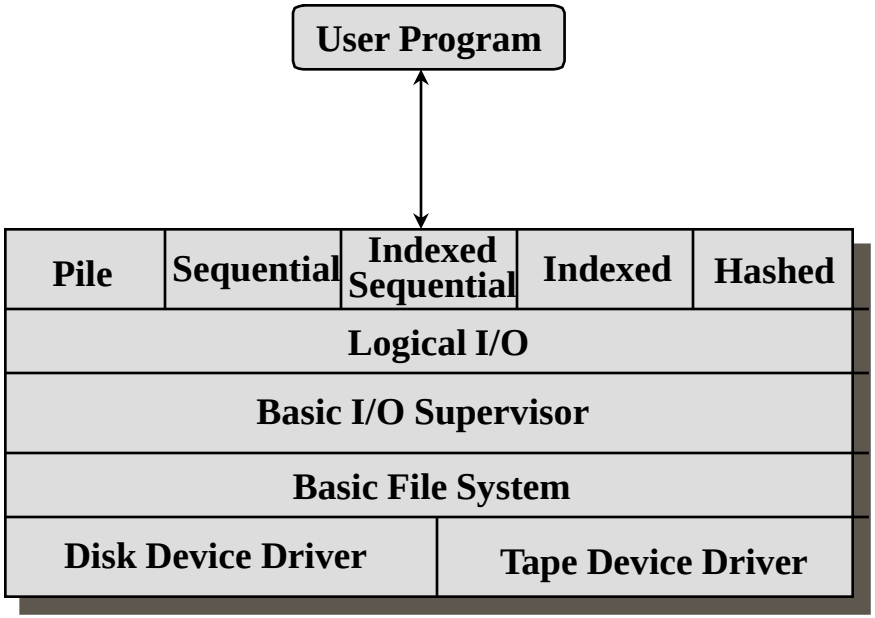
# Terms Used with Files

- ***File***
  - collection of similar records
  - treated as a single entity
  - have unique file names
  - may restrict access
- ***Database***
  - collection of related data
  - relationships exist among elements

# File Management System

- ✓ Controls the way a user of application may access files
- ✓ Controls how data are stored (where, memory allocation, recovery techniques, etc.)
- ✓ Programmer does not need to develop file management software

# File System Software Architecture



# Device Drivers

- ✓ Lowest level in storage management hierarchy
- ✓ Communicates directly with peripheral devices
- ✓ Responsible for starting I/O operations on a device
- ✓ Processes the completion of an I/O request

# Basic File System

- ✓ Physical I/O
- ✓ Deals with exchanging blocks of data
- ✓ Concerned with the placement of blocks
- ✓ Concerned with buffering blocks in main memory

# Basic I/O Supervisor

- Responsible for file I/O initiation and termination (at a higher level than device drivers)
- Maintains control structures
- Concerned with disk access scheduling, to optimize performance
- Part of the operating system

# Logical I/O

- ✓ Allows users and applications to access records
- ✓ Maintains basic data about file
- ✓ Knows about ***access methods:***
  - I.e. the various file structures
  - the different ways to store and process data

# Functions of File Management

- ✓ Identify and locate a selected file
- ✓ Use a directory to describe the location of all files plus their attributes
- ✓ On a shared system do user access control
- ✓ Blocking for access to files (to avoid corruption due to simultaneous access)
- ✓ Allocate files to free blocks
- ✓ Manage free storage for available blocks

# Criteria for File Organization

- ✓ Rapid access
  - needed when accessing a single record
  - not needed for batch mode
- ✓ Ease of update
  - file on CD-ROM will not be updated, so this is not a concern
  - files on hard disks are frequently updated

# Criteria for File Organization

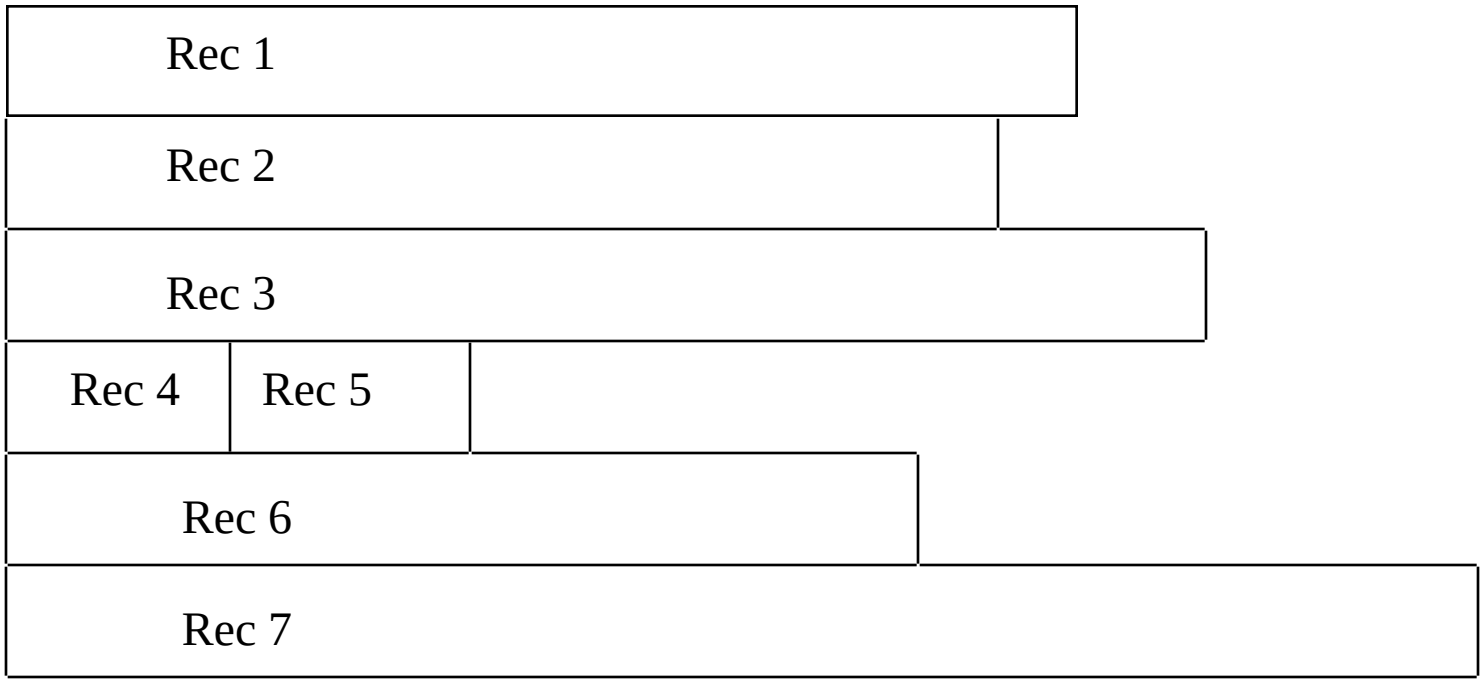
- ✓ Economy of storage
  - should be minimum redundancy in the data
  - redundancy (such as indexing) can be used to speed access
- ✓ Simple maintenance
- ✓ Reliability

# File Organization

- The *Pile*
  - data are collected in the order they arrive
  - purpose is to accumulate a data and save it
  - records may have different # of fields
  - no structure
  - record access is by exhaustive search

# File Organization

The Pile:an example



# File Organization

- The *Sequential* File
  - fixed format used for records
  - records have the same length
  - all fields are the same (order and length)
    - hence: field names & lengths are attributes of the file
  - one field can be the *key* field
    - uniquely identifies the record
    - records can be stored in key sequence

# File Organization

- The Sequential File
  - new records are placed in a log file or transaction file
  - batch update is performed to merge the log file with the master file
  - Why such indirection?

# File Organization

## The Sequential File

Key Field

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# File Organization

- ✓ *Indexed Sequential* File
  - index provides a lookup capability to quickly reach the vicinity of the desired record
    - contains key field and a pointer to the main file
    - index files are **sorted** by the key field value
    - index is searched to find highest key value that is equal or less than the desired key value
    - search continues in the main file at the location indicated by the pointer
    - main file is also sorted on the key field

# File Organization

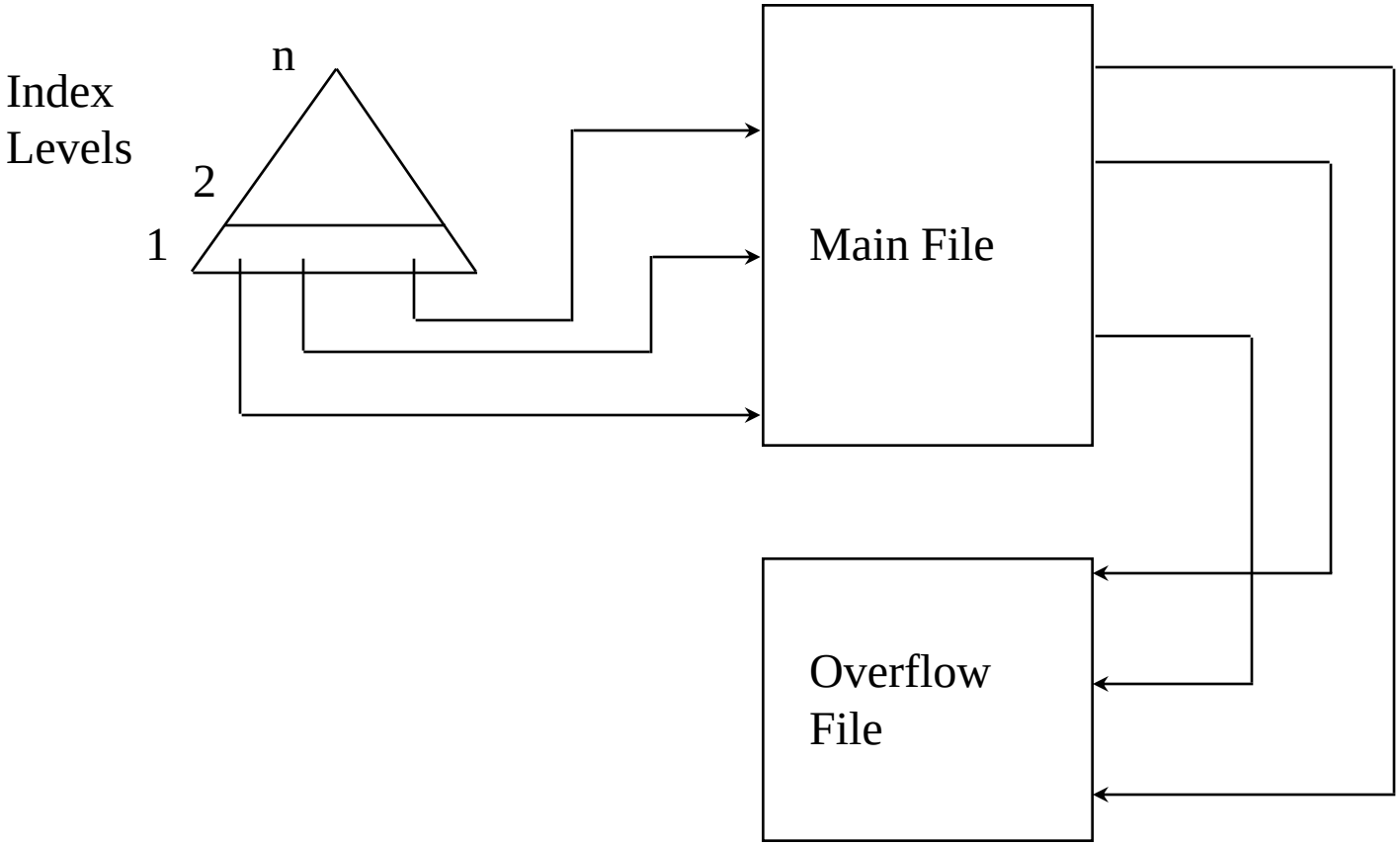
- Comparison of sequential and indexed sequential
  - Example: a file contains 1 million records
  - On average 500,000 accesses are required to find a record in a sequential unsorted file
  - 20 = $\log_2$1,000,000 accesses, if the file is sorted (Why?)
  - If an index contains 50,000 entries (1 entry per 20 records, which fit into a block), it will take on average 16 accesses ($\log_2$50,000) to find the key followed by 1 access in the main file. Now on average it is 17 accesses

# File Organization

- ✓ Indexed Sequential File
  - new records are added to an overflow file
  - record in main file that precedes it is updated to contain a pointer to the new record
  - the overflow is merged with the main file during a batch update
  - multiple indexes for the same key field can be set up to increase efficiency

# File Organization

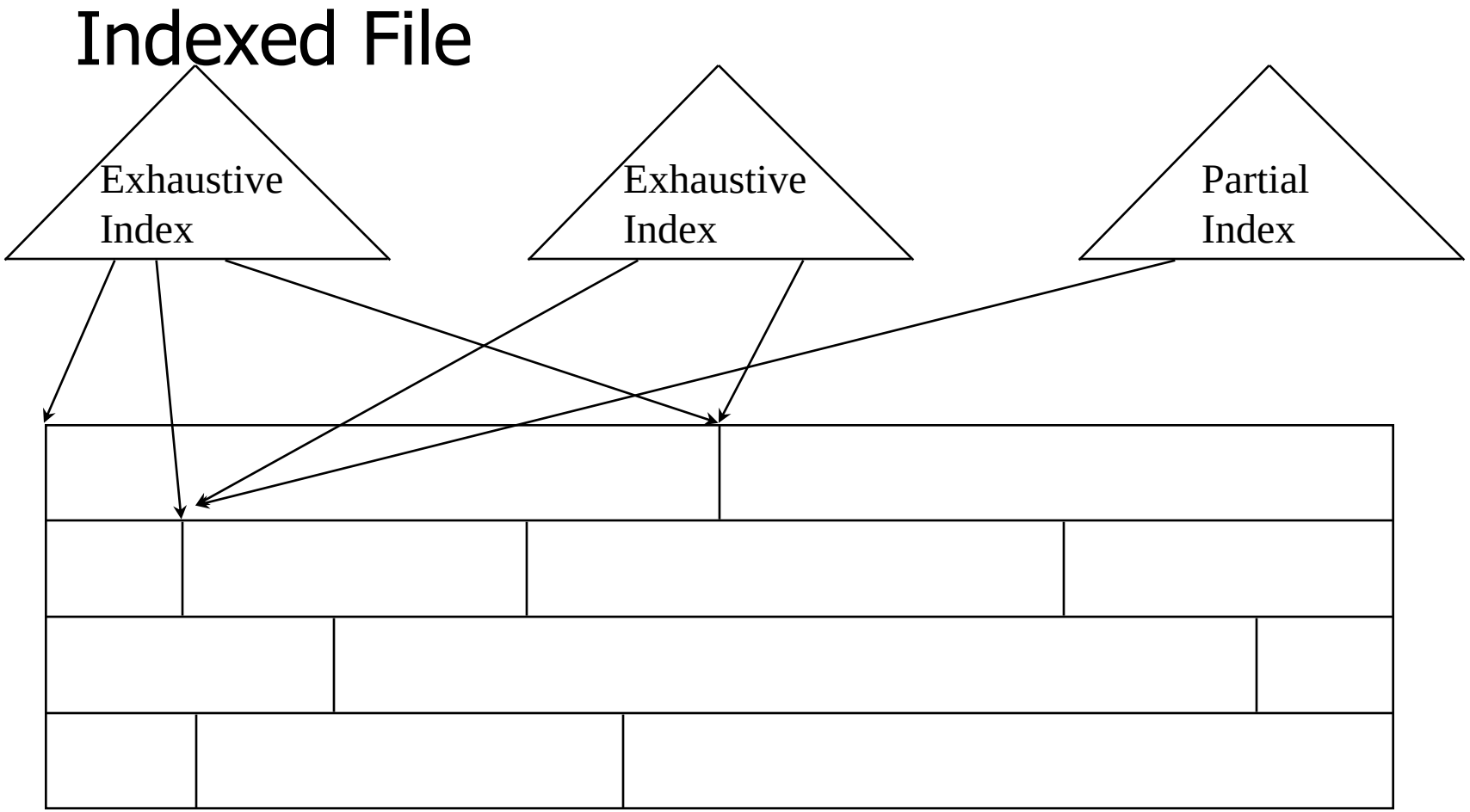
## Indexed Sequential File



# File Organization

- ***Indexed*** File
  - uses multiple indexes for different key fields
  - may contain an exhaustive index that contains one entry for every record in the main file
  - may contain a partial index

# File Organization

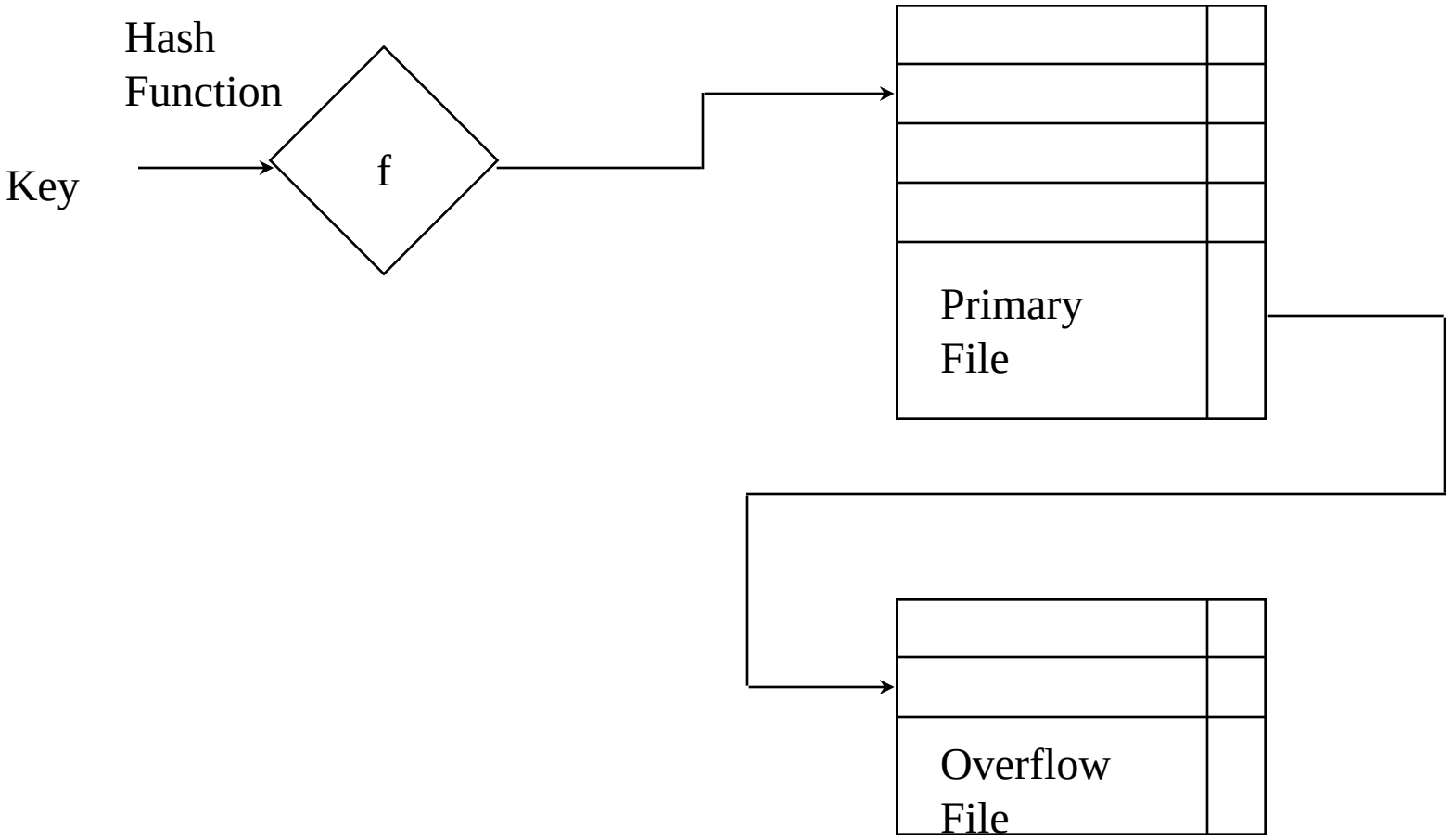


# File Organization

- The *Direct*, or *Hashed*, File
  - directly access a block at a known address
  - key field required for each record

# File Organization

## The Direct, or Hashed, File



# File Directories

- Contains information about files
  - attributes
  - location
  - ownership
- Directory itself is a file manipulated by the operating system
- Provides mapping between file names and the files themselves

# Simple Structure for a Directory

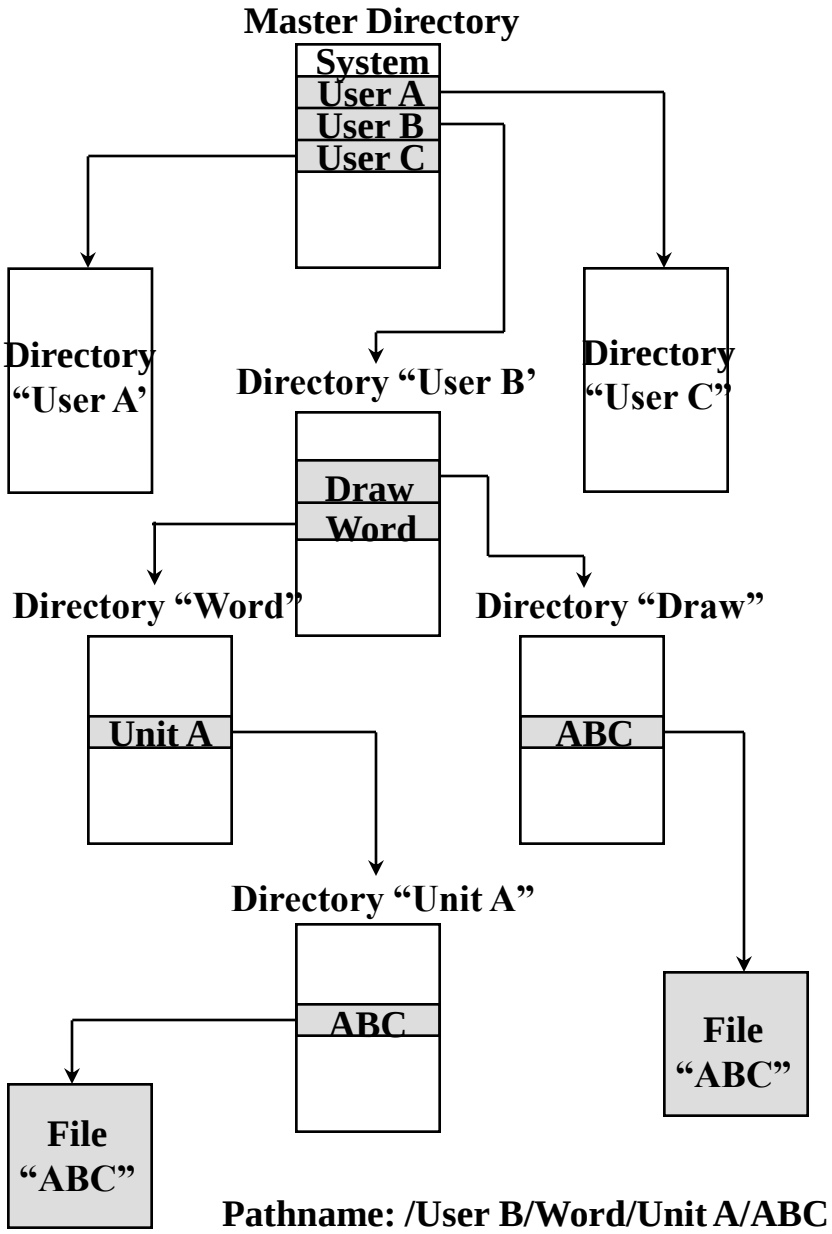
- List of entries, one for each file
- Sequential file with the name of the file serving as the key
- Provides no help in organizing the files
- Forces user to be careful not to use the same name for two different files

# Two-level Directory Scheme

- One directory for each user and a master directory
- Master directory contains entry for each user
  - provides address and access control information
- Each user directory is a simple list of files for that user
- Still provides no help in structuring collections of files

# Hierarchical, or Tree-Structured Directory

- ✓ Master directory with user directories underneath it
- ✓ Each user directory may have subdirectories and files as entries


Master Directory
System
User A
User B
User C
Directory "User A'
Directory "User B'
Directory "User C"
Draw
Word
Directory "Word"
Directory "Draw"
Unit A
ABC
Directory "Unit A"
ABC
File "ABC"
File "ABC"
Pathname: /User B/Word/Unit A/ABC

# Hierarchical, or Tree-Structured Directory

- Files can be located by following a path from the root, or master, directory down various branches
  - this is the *pathname* for the file
- Can have several files with the same file name as long as they have unique path names

# Hierarchical, or Tree-Structured Directory

- ✓ Current directory is the working directory
- ✓ Files are referenced relative to the working directory

# File Sharing

- ✓ Way to control access to a particular file
- ✓ Users or groups of users are granted certain access rights to a file

# Access Rights

- None
  - user may not know of the existence of the file
  - user is not allowed to read the user directory that includes the file
- Knowledge
  - user can only determine that the file exists and who its owner is

# Access Rights

- Execution
  - the user can load and execute a program but cannot copy it
- Reading
  - the user can read the file for any purpose, including copying and execution
- Appending
  - the user can add data to the file but cannot modify or delete any of the file's contents

# Access Rights

- Updating
  - the user can modify, deleted, and add to the file's data.  This includes creating the file, rewriting it, and removing all or part of the data
  - Changing protection
  - user can change access rights granted to other users
- Deletion
  - user can delete the file

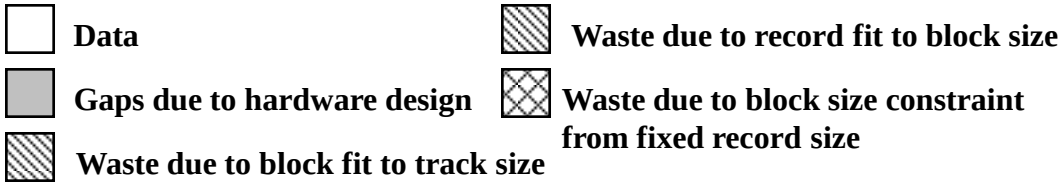
# Access Rights

- Owners
  - has all rights previously listed
  - may grant rights to others using the following classes of users:
    - specific user
    - user groups
    - all users

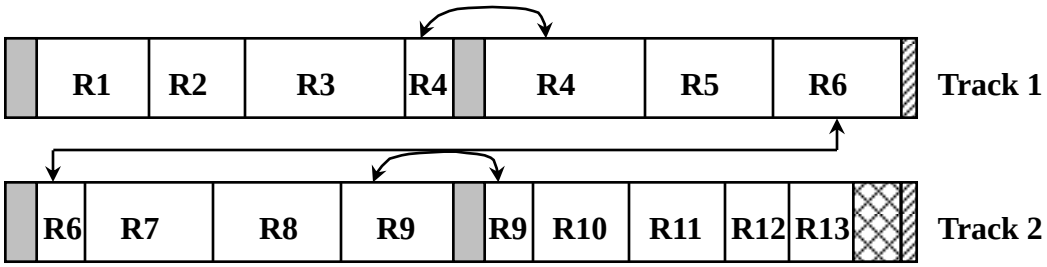
# Simultaneous Access

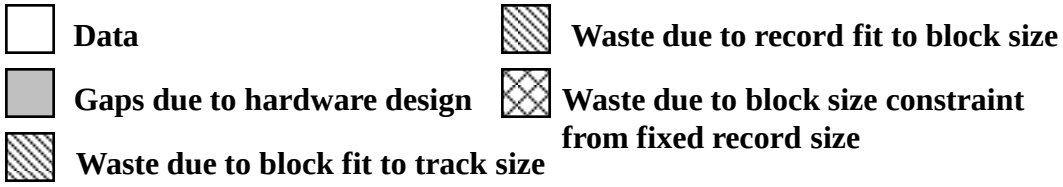
- User may lock entire file when it is to be updated
- User may lock the individual records during the update
- Mutual exclusion and deadlock are issues for shared access

# Record Blocking Methods - Fixed Blocking

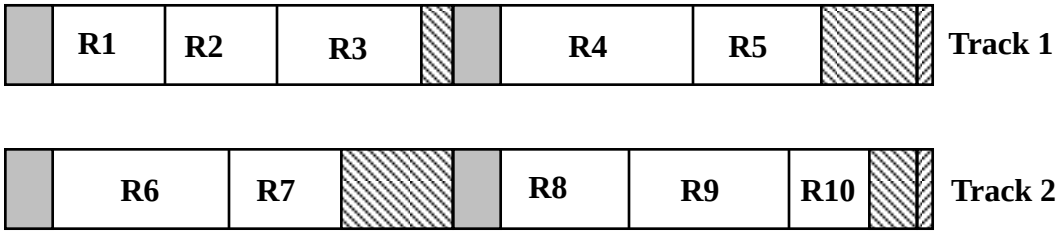



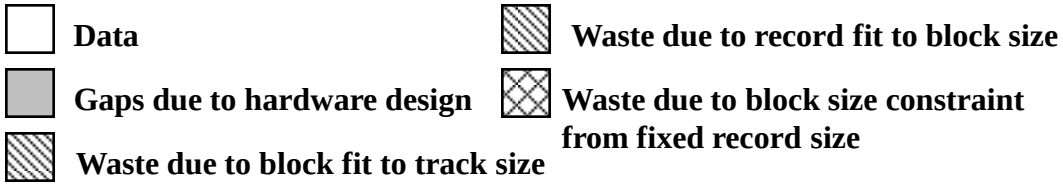
# Record Blocking Methods - Variable Blocking: Spanned



# Record Blocking Methods - Variable Blocking: UnSpanned



# Secondary Storage Management

- Space must be allocated to files
- Must keep track of the space available for allocation
- Space is allocated as one or more contiguous units or portions

# Preallocation Method

- ✓ Disadvantages
  - Needs to know the maximum size for the file at the time of creation
  - Difficult to reliably estimate the maximum potential size of the file
  - Tend to overestimated file size so as not to run out of space, hence waste of space
- ✓ Advantages
  - Used in specialized systems to make sure a file gets contiguous chunk of disk space (to improve performance)

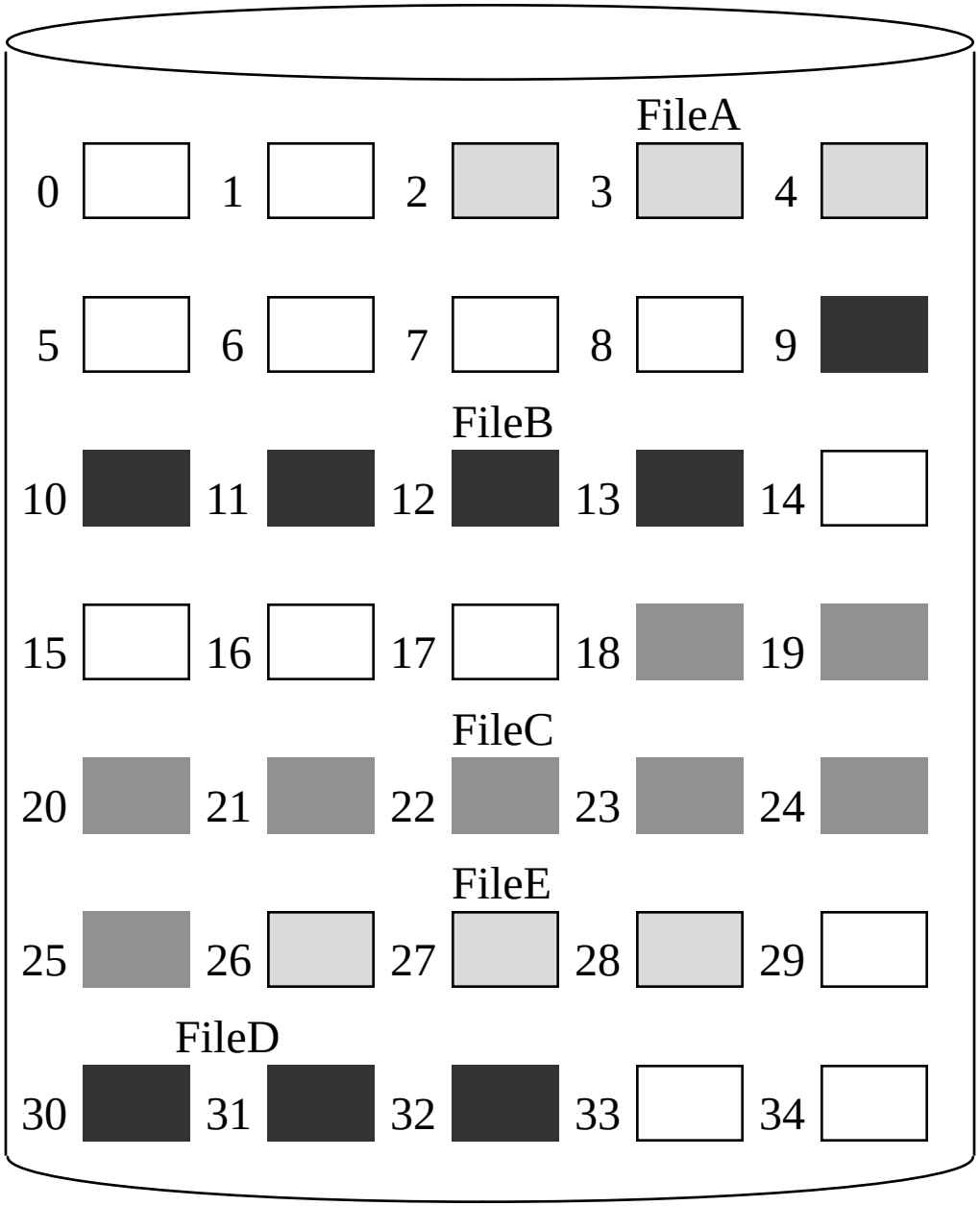
# Portion Size Considerations

- ✓ Contiguity of space increases performance
- ✓ Large number of small portions increases the size of tables needed to keep track of the different portions of files
- ✓ Fixed-size portions simplify the process of reallocation of space
- ✓ Variable-size portions minimize waste of unused storage

# Methods of File Allocation

- ***Contiguous allocation***
  - single set of blocks is allocated to a file at the time of creation
  - only a single entry in the file allocation table
    - starting block and length of the file
- Fragmentation will occur (due to file deletion)
- With time, it'll become difficult to find contiguous blocks of sufficient length for new files

# Contiguous File Allocation



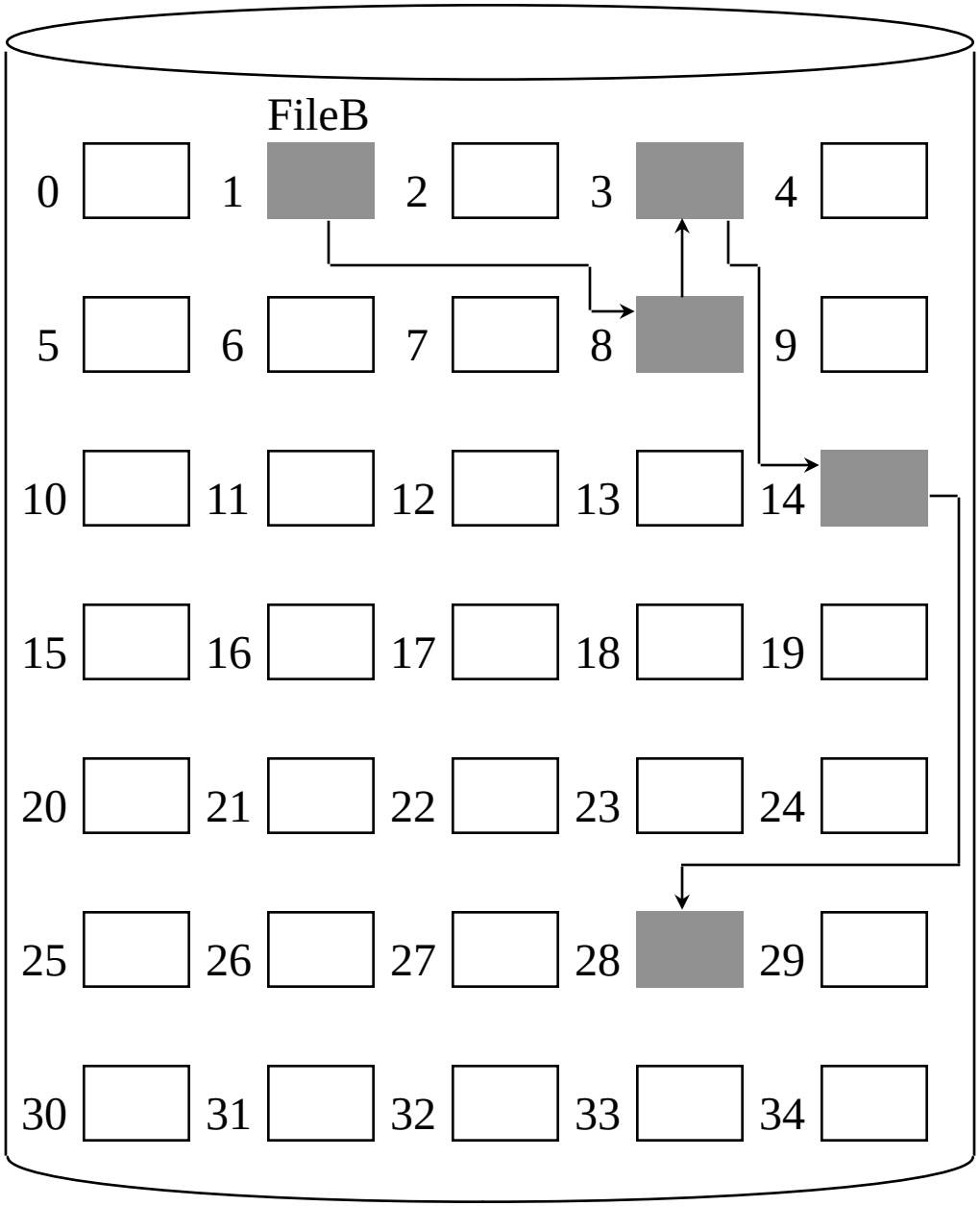

File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| FileA | 2 | 3 |
| FileB | 9 | 5 |
| FileC | 18 | 8 |
| FileD | 30 | 2 |
| FileE | 26 | 3 |

# Methods of File Allocation

- *Chained allocation*
  - allocation on the basis of individual block
  - each block contains a pointer to the next block in the chain
  - only single entry in the file allocation table
    - starting block and length of file
- No fragmentation
- Any free block can be added to the chain
- No easy way to satisfy the principle of locality

# Chained File Allocation



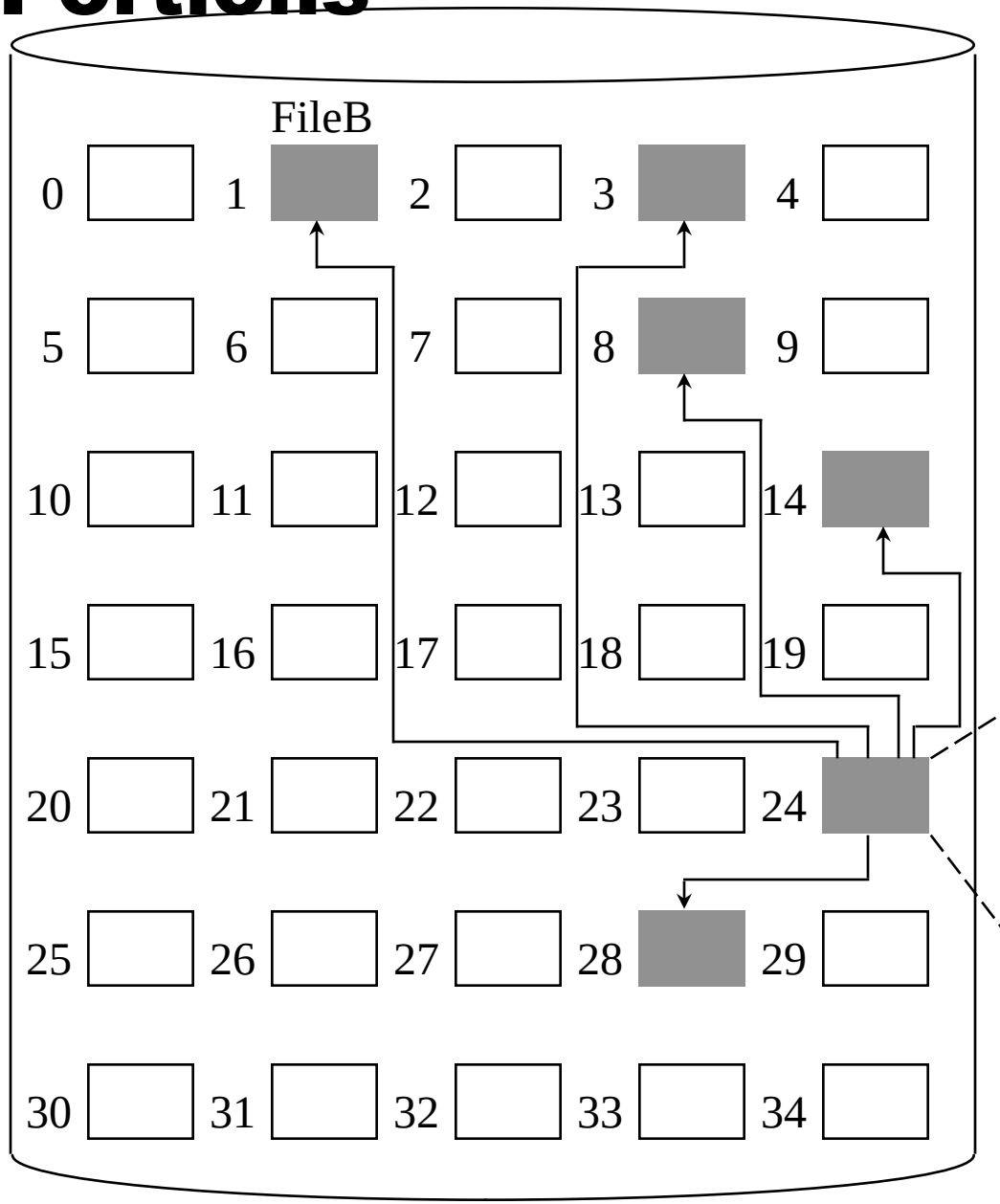

File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| ... | ... | ... |
| FileB | 1 | 5 |
| ... | ... | ... |

# Methods of File Allocation

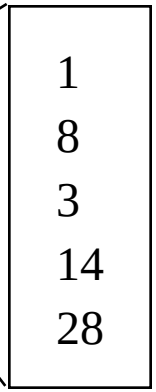
- Indexed allocation
  - file allocation table contains a separate one-level index for each file
  - the index has one entry for each portion allocated to the file
  - the file allocation table contains block number for the index

# Indexed Allocation with Block Portions

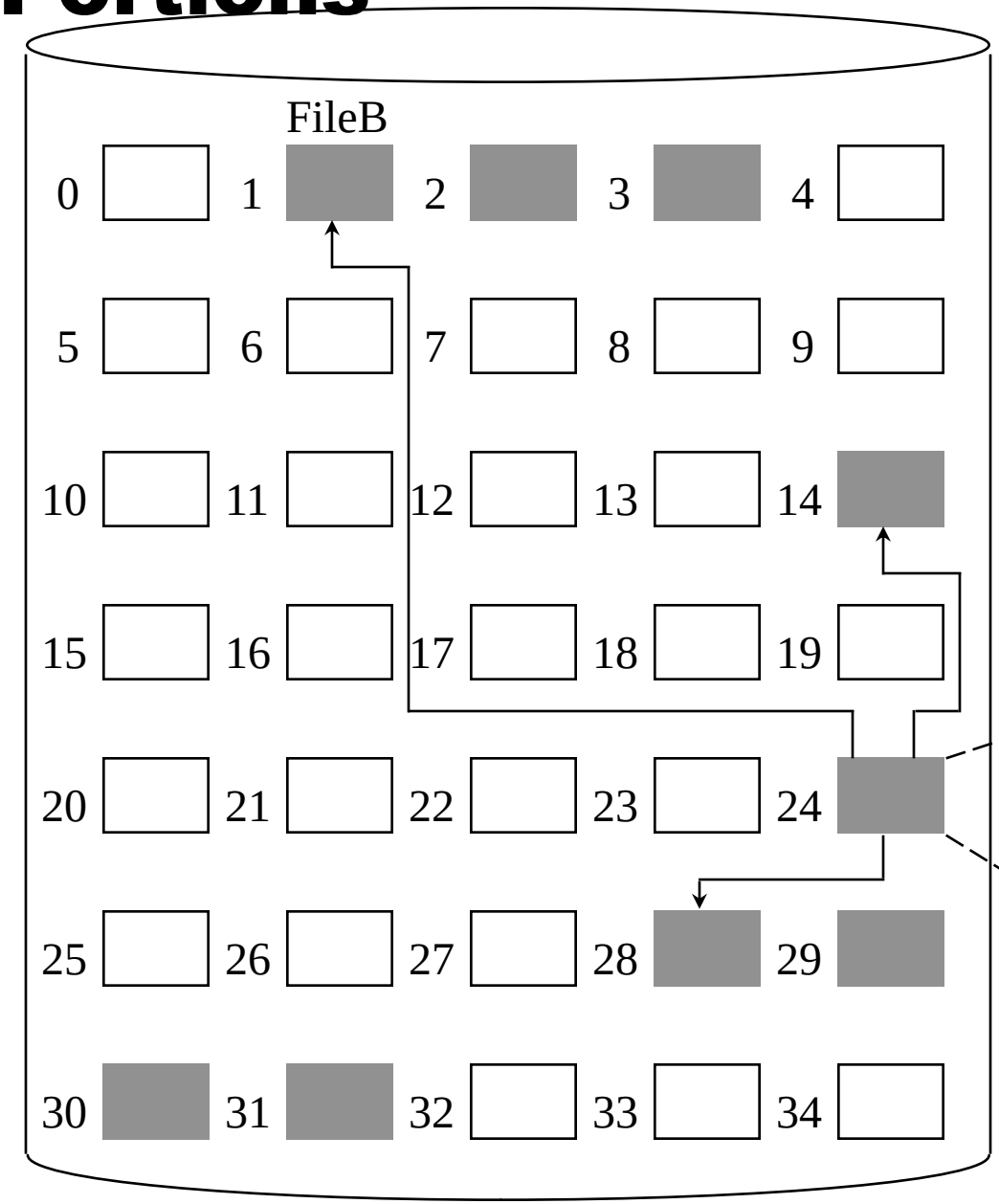



File Allocation Table

| File Name | Index Block |
|---|---|
| ... | ... |
| FileB | 24 |
| ... | ... |



# Indexed Allocation - Var Length Portions




File Allocation Table

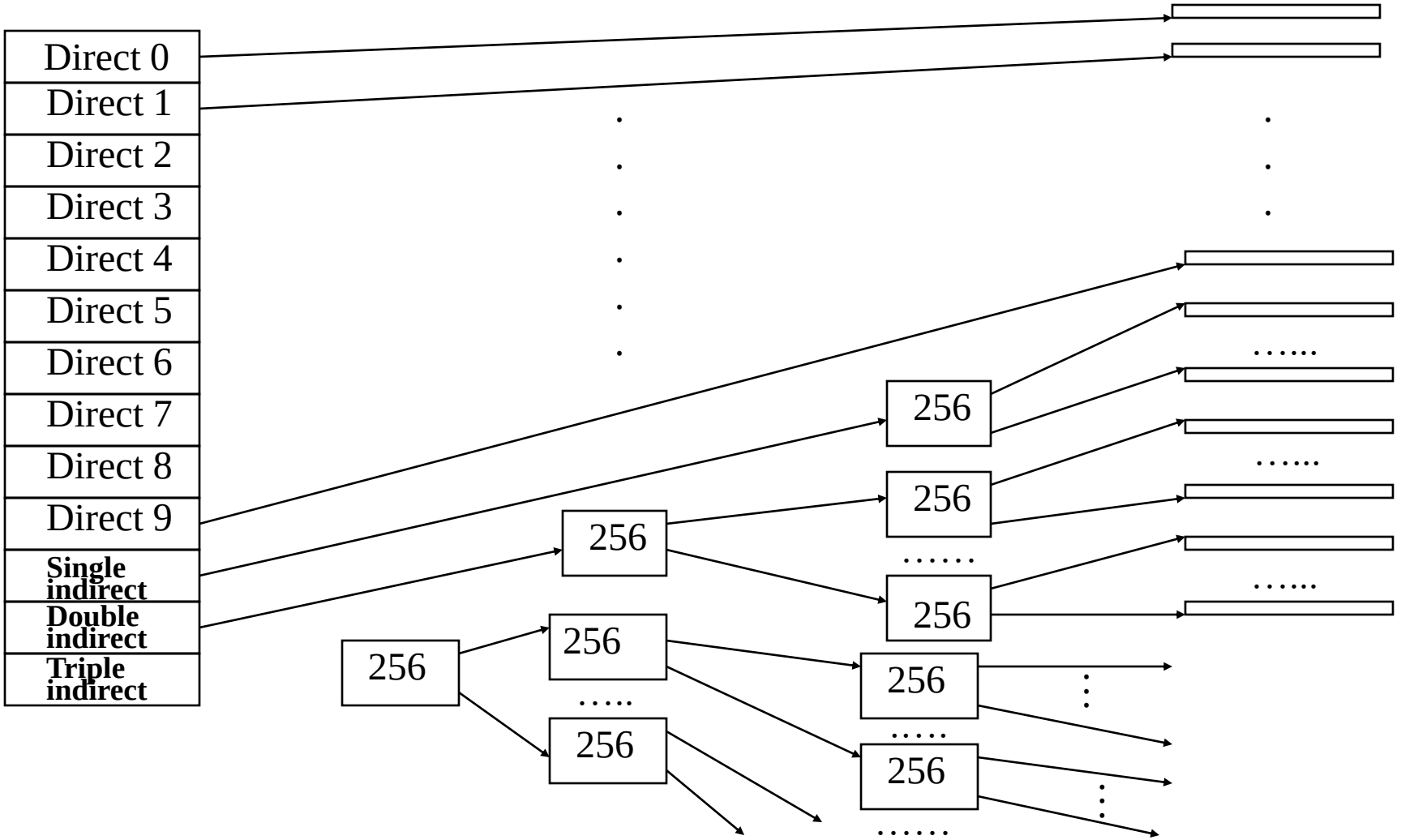
| File Name | Index Block |
|---|---|
| ... | ... |
| FileC | 24 |
| ... | ... |

| Start Block | Length |
|---|---|
| 1 | 3 |
| 28 | 4 |
| 14 | 1 |

# (Traditional) UNIX File Management

- ✓ Files are streams of bytes
- ✓ Types of files
  - *ordinary* - contents entered by user or program
  - *directory* - contains list of file names and pointers to inodes (index nodes)
  - *special* - used to access peripheral devices
  - *named* - named pipes (communication channels)

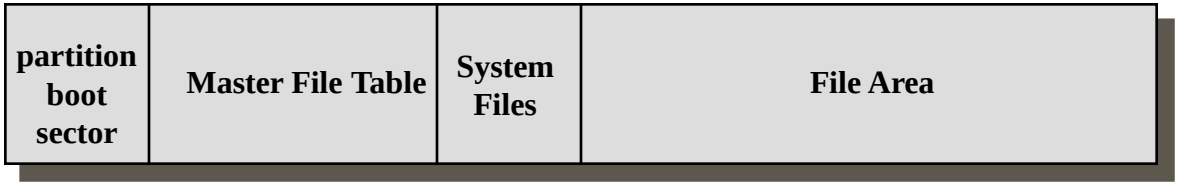
# UNIX Block Addressing Schema



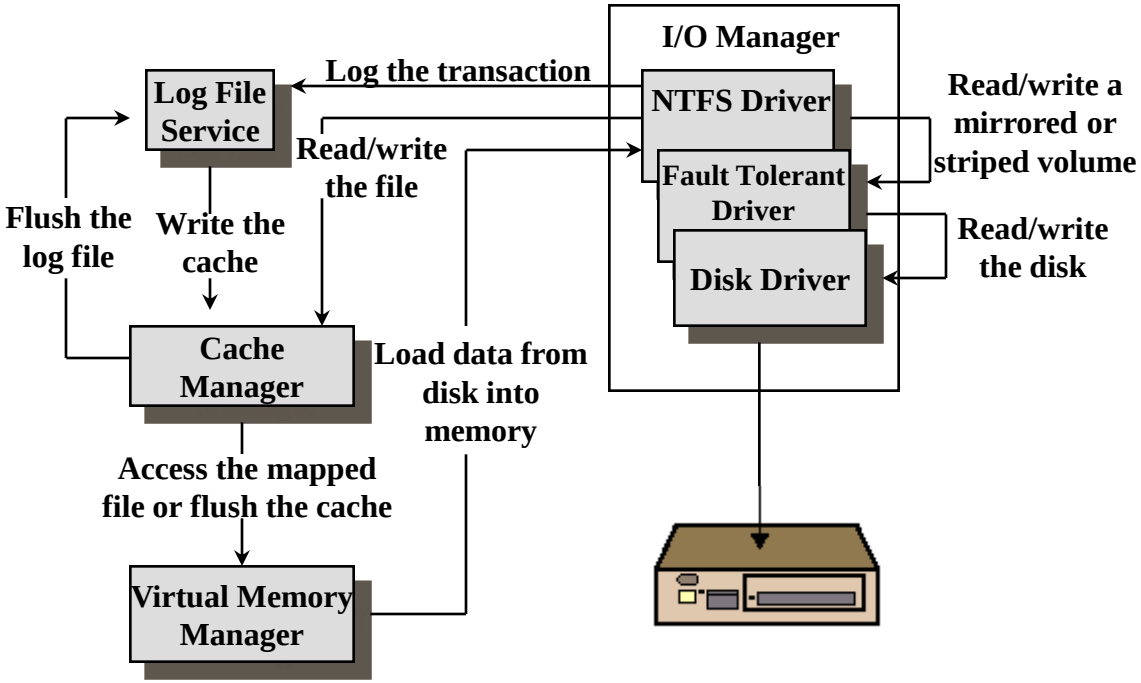
# NTFS: Windows NT/XP File System

- ✓ Sector - smallest unit of storage on a disk
- ✓ Cluster - one or more contiguous sectors
- ✓ Volume - logical partition on a disk

# NTFS Volume Layout



# Windows NTFS Components




The *log file* is used to record all changes to the volume (a technique borrowed from database management) to ensure recoverability after crashes.