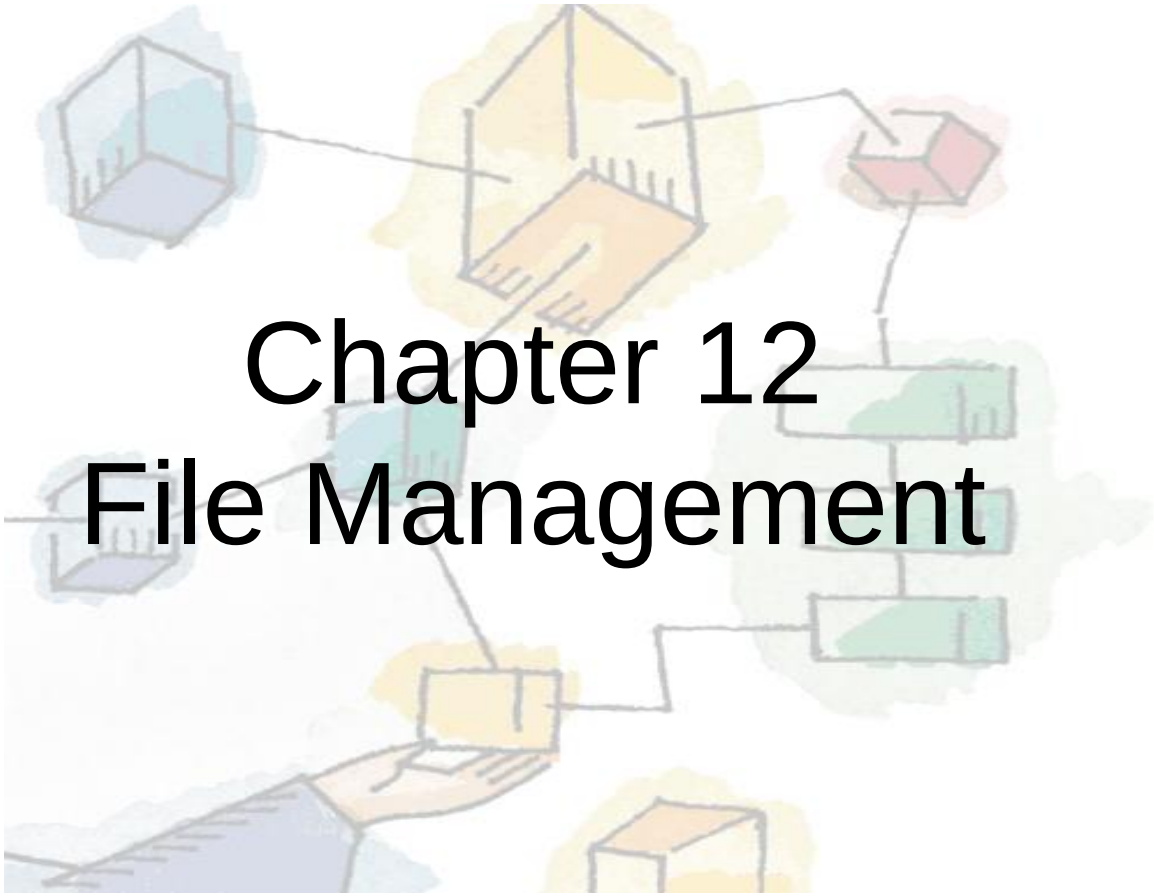
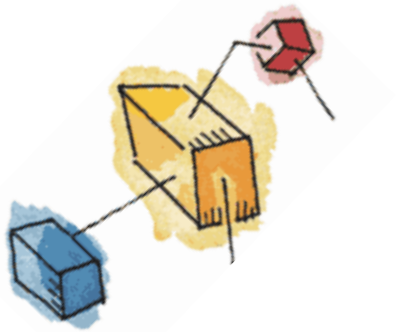


Chapter 12
File Management



Files

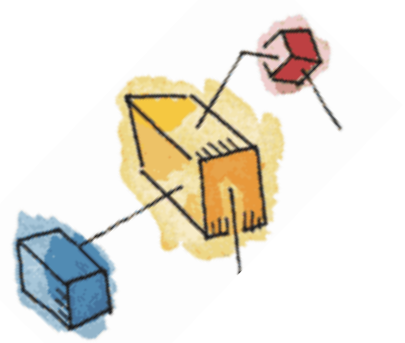
- Files are the central element to most applications
- The File System is one of the most important part of the OS to a user

- **Desirable properties of files:**
  - Long-term existence
  - Sharable between processes
  - Structure

File Management System

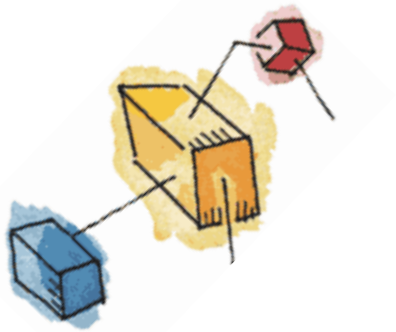
- File management system consists of utility programs that run as privileged applications

- Concerned with secondary storage



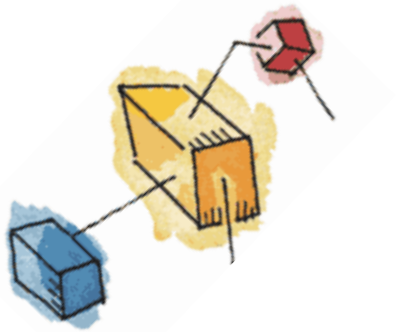
Typical Operations

- File systems also provide functions which can be performed on files, typically:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write



Terms

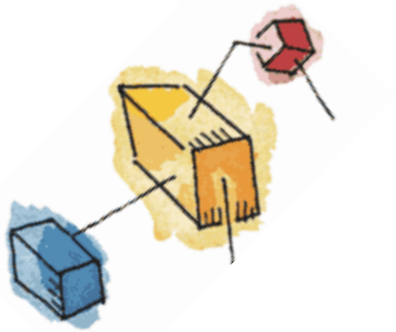
- Four terms are in common use when discussing files:
  - Field
  - Record
  - File
  - Database



Fields and Records

- **Fields**
  - Basic element of data
  - Contains a single value
  - Characterized by its length and data type

- **Records**
  - Collection of related fields
  - Treated as a unit

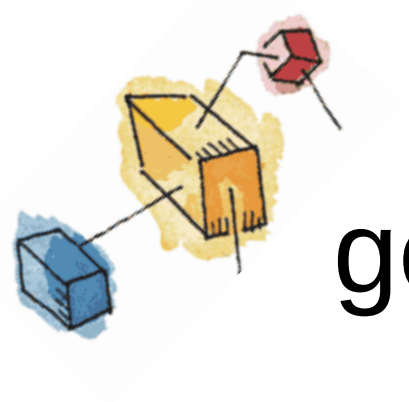


File and Database

- **File**
  - Have file names
  - Is a collection of similar records
  - Treated as a single entity
  - May implement access control mechanisms
- **Database**
  - Collection of related data
  - Relationships exist among elements
  - Consists of one or more files

Objectives for a File Management System

- Meet the data management needs of the user
- Guarantee that the data in the file are valid
- Optimize performance
- Provide I/O support for a variety of storage device types
- Provide a standardized set of I/O interface routines to user processes
- Provide I/O support for multiple users (if needed)
- Minimize lost or destroyed data



Requirements for a general purpose system

1. Each user should be able to create, delete, read, write and modify files
2. Each user may have controlled access to other users' files
3. Each user may control what type of accesses are allowed to the users' files
4. Each user should be able to restructure the user's files in a appropriate form

Requirements cont.

5. Each user should be able to move data between files
6. Each user should be able to back up and recover the user's files in case of damage
7. Each user should be able to access the user's files by using symbolic names

File System Architecture

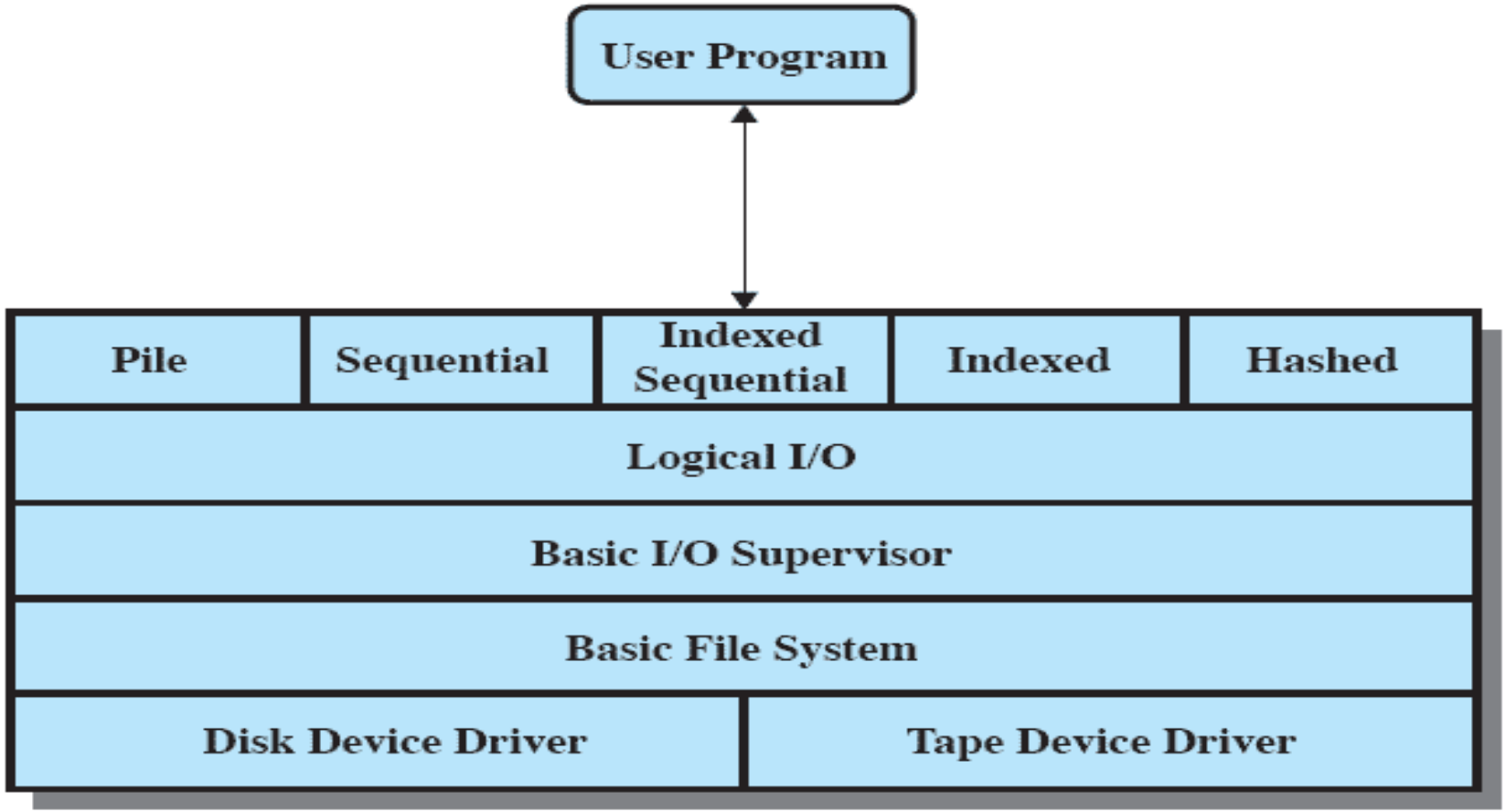
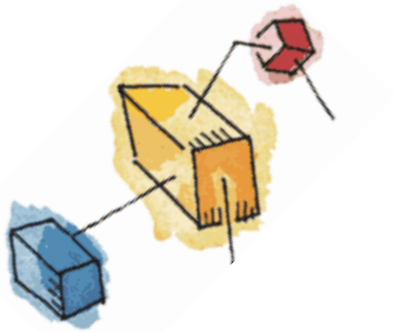
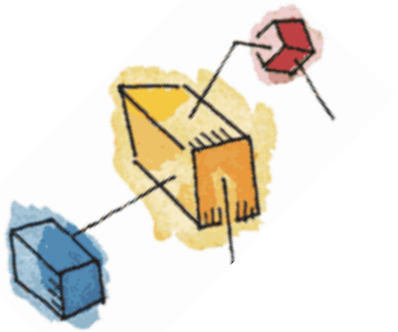


Figure 12.1 File System Software Architecture



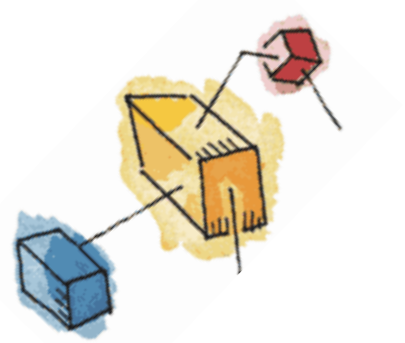
Device Drivers

- Lowest level
- 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

- Also known as Physical I/O
- Primary interface with the environment outside the computer system
- Deals with exchanging blocks of data
- Concerned with the placement of blocks
- Concerned with buffering blocks in main memory

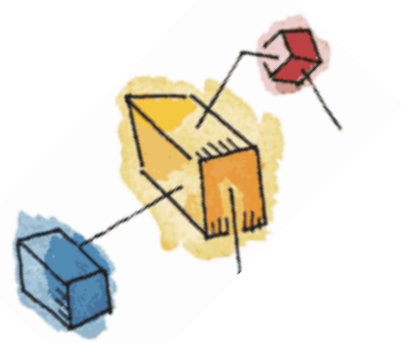


Basic I/O Supervisor

- Responsible for all file I/O initiation and termination.
- Control structures deal with
  - Device I/O,
  - Scheduling,
  - File status.
- Selects and schedules I/O with the device

Logical I/O

- Interface between program's logical commands and physical details of disk
- Physical I/O deals with blocks while logical I/O deals with records
- Enables users and applications to access records
- Maintains basic data about file



Access Method

- Closest to the user
- Reflect different file structures
- Provides a standard interface between applications and the file systems and devices that hold the data
- Access method varies depending on the ways to access and process data for the device.

Elements of File Management

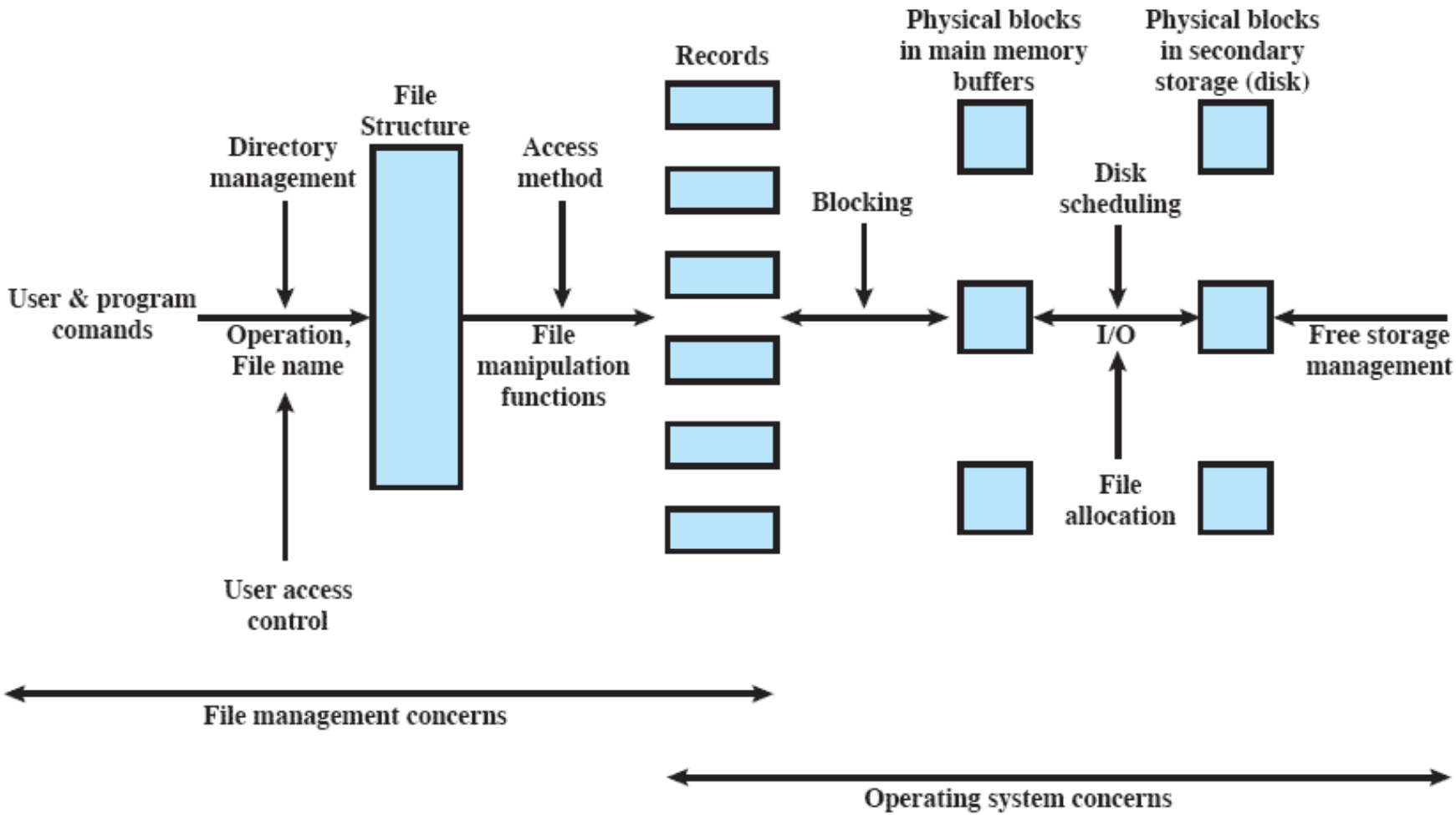
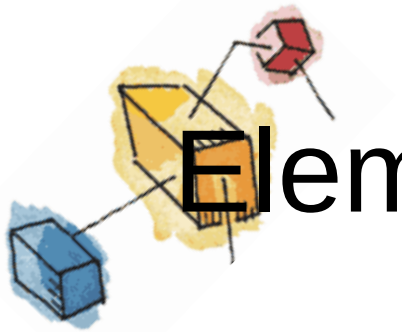
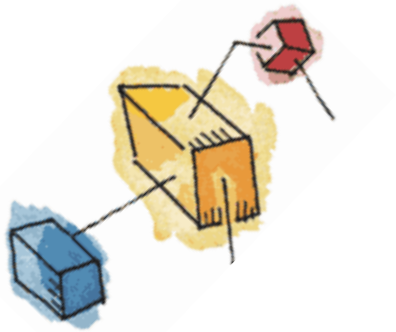


Figure 12.2 Elements of File Management



Elements of File Management

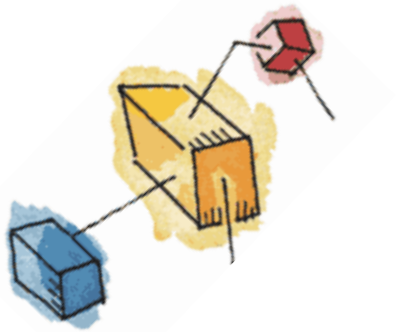
- Directory Management:
  - Load file from directory to perform any operation
- Access Control
  - Verify User access rights
- Access method
  - translates user commands to file manipulation commands
- Blocking
  - I/O is performed on block by block basis
- Scheduling
  - Needed for optimum performance
- Free Space Management



File Organization

- File Management Referring to the logical structure of records
  - Physical organization discussed later

- Determined by the ***way*** in which files are accessed



Criteria for File Organization

- Important criteria include:
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

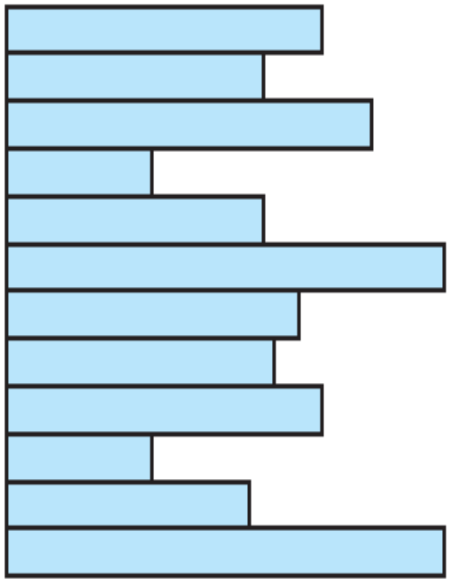
- Priority will differ depending on the use (e.g. read-only CD vs Hard Drive)

File Organisation Types

- Many exist, but usually variations of:
  - Pile
  - Sequential file
  - Indexed sequential file
  - Indexed file
  - Direct, or hashed, file

The Pile

- Simplest form of organization
- Data are collected in the order they arrive
  - No structure
- Purpose is to accumulate a mass of data and save it
- Variable length records, variable number of fields
- Record access is by exhaustive search

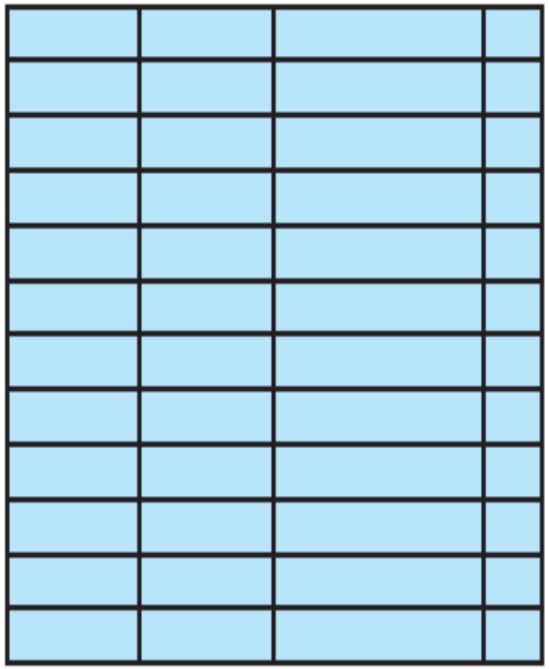


Variable-length records
Variable set of fields
Chronological order

(a) Pile File

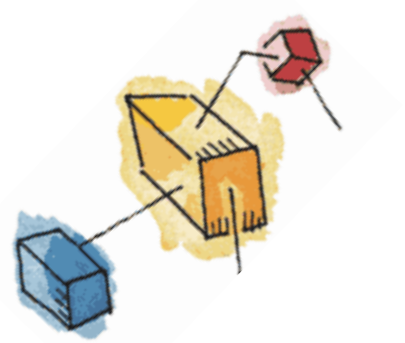
The Sequential File

- Most common form
- Fixed format used for records with fixed length
- All fields the same (order and length)
- Field names and lengths are attributes of the file
- Key field is used
  - Uniquely identifies the record
  - Records are stored in key sequence



Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

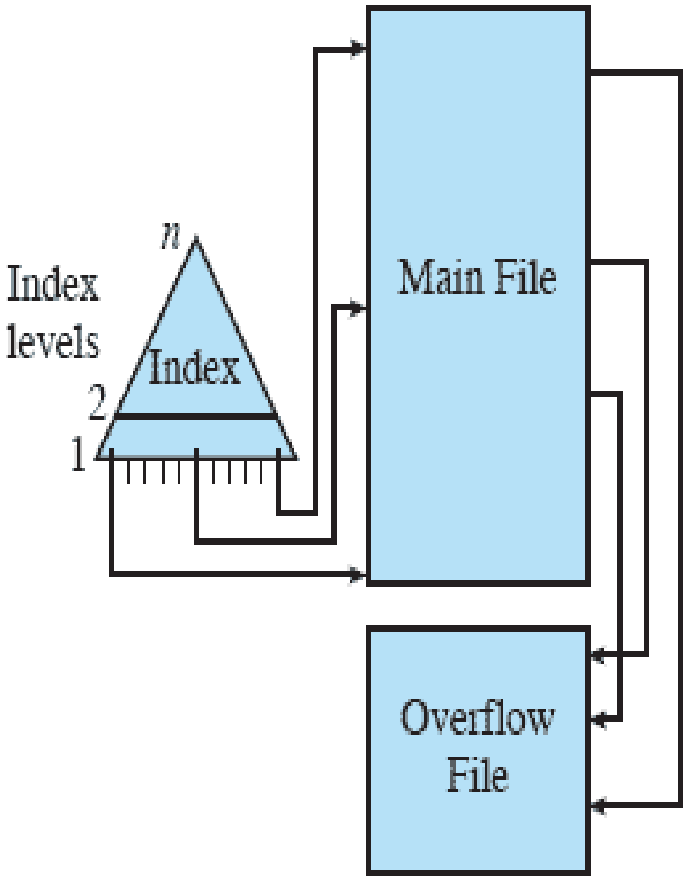


The Sequential File

- Works poorly when queries are made for individual records
  - For very large sequential file
- Additions/Insertions are also difficult
  - New records are placed in a separate pile file called log file
    - And merged later in batch mode
- Solution?
  - Using linked list

Indexed Sequential File

- Maintains the key characteristic of the sequential file:
  - Records are organized in sequence based on a key field.
- **Two features** are added:
  - **An index** to the file to support random access,
  - and an **overflow file**.



(c) Indexed Sequential File

Indexed Sequential File

- **Index file:**
  - Simple sequential file where each record contains two fields
    - Key – same as key in main file
    - Pointer to main file
- **Overflow file:**
  - Similar to log file but it is integrated with the main file
    - A Record in overflow file can be located by following its predecessor

Indexed Sequential File

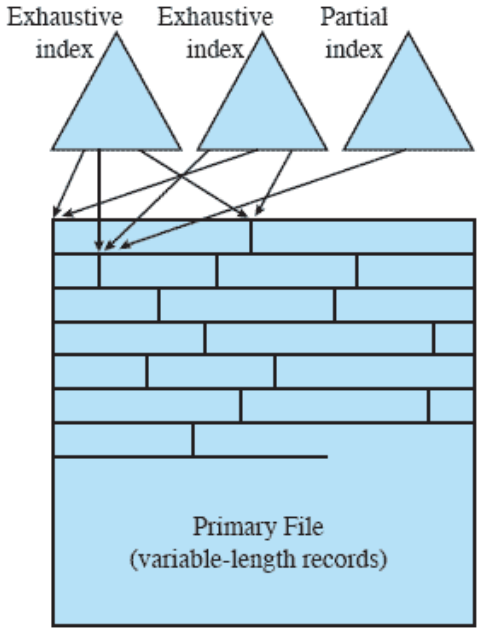
- **Addition Operation:**
  - New record is added to overflow file
  - Record in the main file that precedes new record is updated to contain a pointer to the new record
  - If the preceding record is in the overflow file itself, then its pointer is updated
- Main file is merged with overflow file in batch mode

Indexed Sequential File

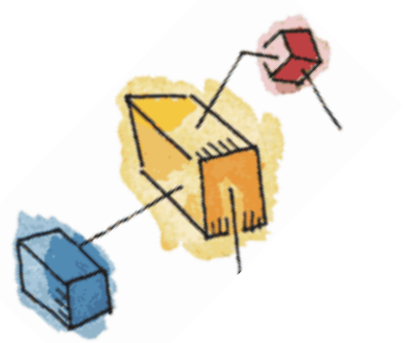
- Advantages:
  - Reduced access time
  - Can be improved further by using multiple levels of indexing
- Issues:
  - When the file is searched using a criteria other than the key, this approach won't work

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 that contains records based on field of interest
- When a new record is added to the main file, all of the index files must be updated.

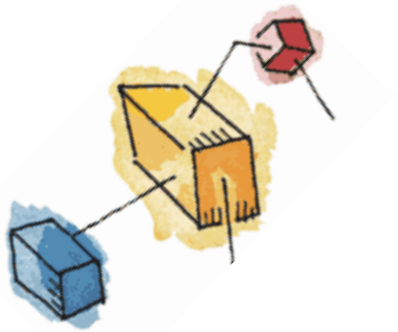


(d) **Indexed File**



Indexed File

- This structure can be helpful when the information is required on time
  - E.g. Booking information systems



Direct/Hashed File

- Directly access a block at a known address
- Key field required for each record

Directory: Contents

- Contains information about files
  - Attributes
  - Location
  - Ownership
- Directory itself is a file owned by the operating system
- Provides mapping between file names and the files themselves

Directory Elements: Basic Information

- File Name
  - Name as chosen by creator (user or program).
  - Must be unique within a specific directory.
- File type
- File Organisation
  - For systems that support different organizations

Directory Elements: Address Information

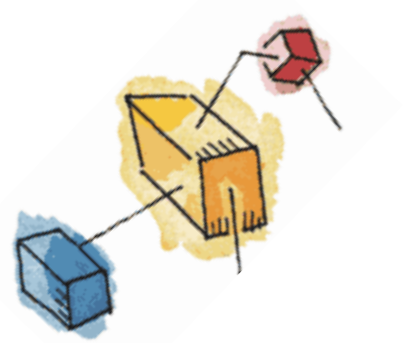
- Volume
  - Indicates device on which file is stored
- Starting Address
- Size Used
  - Current size of the file in bytes, words, or blocks
- Size Allocated
  - The maximum size of the file

Directory Elements: Access Control Information

- Owner
  - The owner may be able to grant/deny access to other users and to change these privileges.
- Access Information
  - May include the user's name and password for each authorized user.
- Permitted Actions
  - Controls reading, writing, executing, transmitting over a network

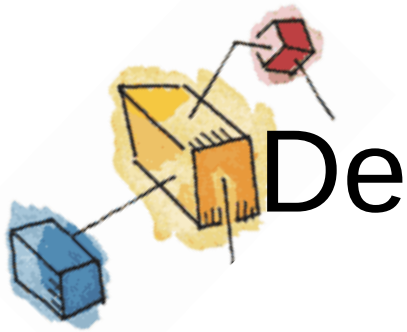
Directory Elements: Usage Information

- Date Created
- Identity of Creator
- Date Last Read Access
- Identity of Last Reader
- Date Last Modified
- Identity of Last Modifier
- Date of Last Backup
- Current Usage
  - Current activity, locks, etc



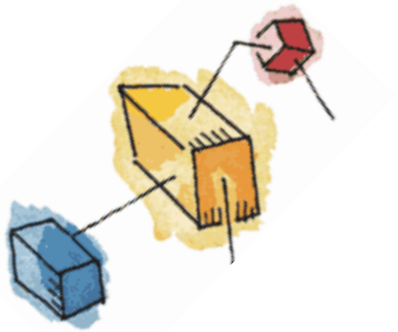
Blocks and records

- Records are the logical unit of access of a structured file
  - But blocks are the unit for I/O with secondary storage
- Records must be organized as blocks to perform I/O operation
- **Three approaches** are common
  - Fixed length blocking
  - Variable length spanned blocking
  - Variable-length unspanned blocking



Deciding factors for Blocks

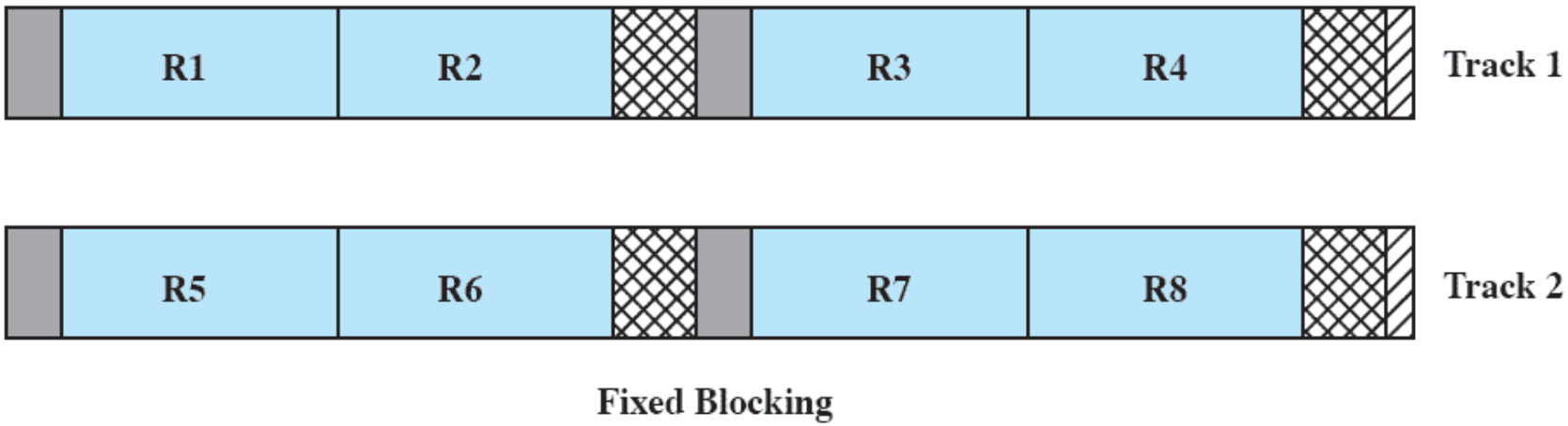
- Fixed length or Variable Length?
  - Fixed length simplifies I/O and organization
- Size of block Large or Small?
  - Large size will result in more records passed in one I/O operation (Good for sequential access)
  - Small size will be useful in absence of locality of reference (For random access)



Fixed Blocking

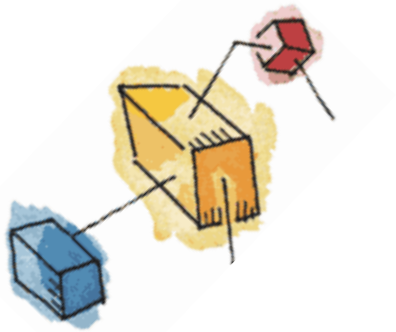
- Fixed-length records are used
- Similar to paging
- An integral number of records are stored in a block.
- Unused space at the end of a block is ***internal fragmentation***

Fixed Blocking



Fixed Blocking

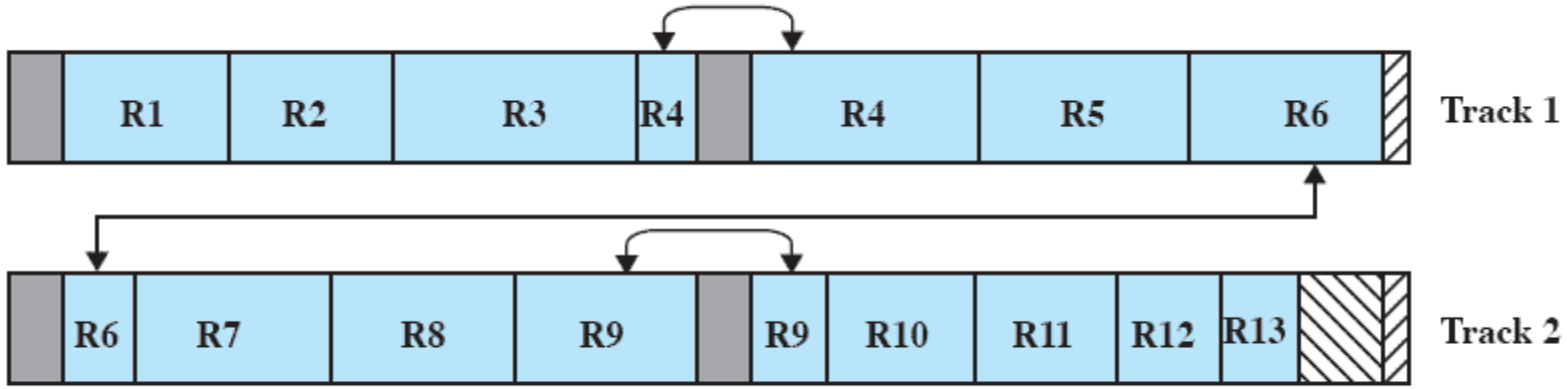
- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size



Variable Length Spanned Blocking

- Variable-length records are used
- Blocks are packed together with no unused space
- Some records may span multiple blocks
  - Continuation is indicated by a pointer to the successor block
- Record size not limited to block size

Variable Blocking: Spanned



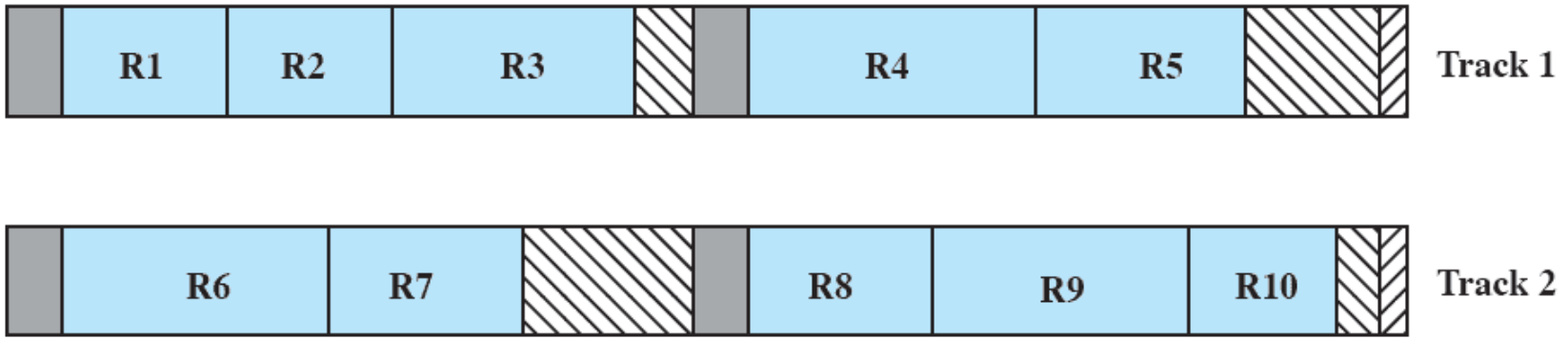
Variable Blocking: Spanned

- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

Variable-length unspanned blocking

- Uses variable length records without spanning
- Wasted space in most blocks
  - because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.

Variable Blocking: Unspanned



Variable Blocking: Unspanned

- Data
- Gaps due to hardware design
- Waste due to block fit to track size
- Waste due to record fit to block size
- Waste due to block size constraint from fixed record size

Summary: Record Blocking

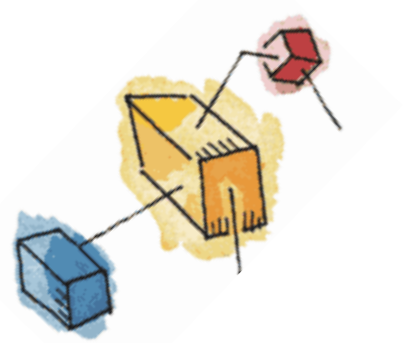
- Fixed blocking is common for sequential files with fixed length records
- Variable length spanned blocking is efficient for storage, does not limit record size
  - But difficult in implementation
- Variable length unspanned blocking results in wasted space and limits record size to size of block

Secondary Storage Management

- Files are stored on secondary storage as a collection of blocks
- The Operating System is responsible for allocating blocks to files
- **Two related issues**
  - Space must be allocated to files (Allocation)
  - Must keep track of the space available for allocation (Free Space Management)

File allocation issues

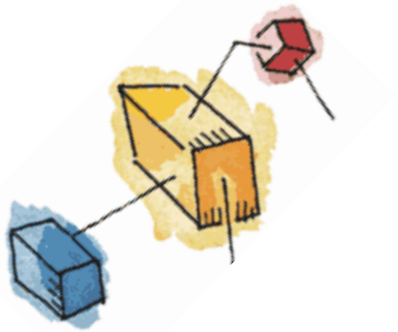
1. When a file is created – is the maximum space allocated at once?
2. What size should be the 'portion' size allocated to a file?
   - Contiguous set of blocks or
   - One block at a time
3. Which data structure should be used to keep track of the file portions?



Preallocation vs Dynamic Allocation

- Declare 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 dynamic allocation should be preferred



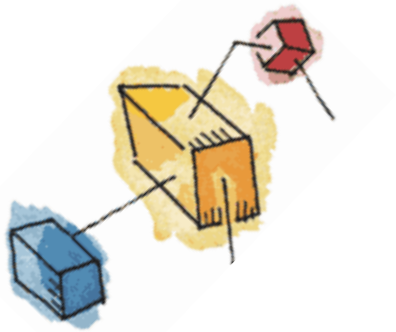
Portion size

- **Two extremes:**
  - Portion large enough to hold entire file is allocated
  - Allocate space one block at a time

- Trade-off between efficiency from the point of view of a single file, or the overall system efficiency

Portion size

- Variable, large contiguous portions allocation
  - Gives better performance
  - Avoids wastage due to variable size
  - FAT small in size
- Block by block basis allocation
  - Good flexibility
  - No contiguity
  - FAT large in size



Combinations

- Pre allocation and variable, large contiguous portions
  - File is allocated one contiguous group of blocks at creation time
  - No need of FAT
    - Only the pointer to first block and total number of blocks allocated need to be stored
- Pre allocation on block by block basis
  - Allocate blocks on block by block basis at creation time
  - Fixed size FAT
    - Due to pre allocation

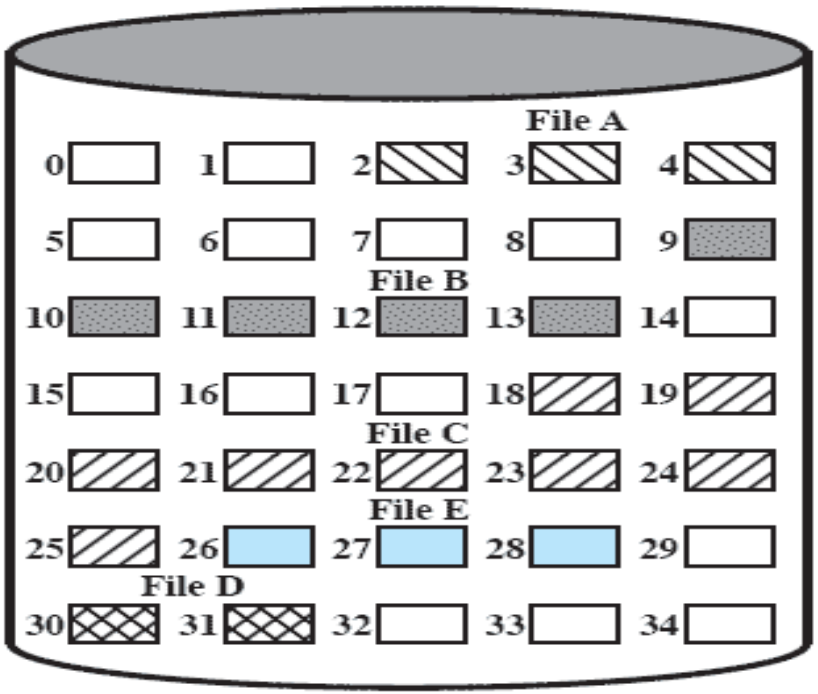
File Allocation Method

- Three methods are in common use:
  - contiguous,
  - chained, and
  - indexed.

Contiguous Allocation

- Single contiguous set of blocks is allocated to a file at the time of creation
- Pre allocation with variable, large portions
- Only a single entry in the file allocation table
  - Starting block and length of the file
- **Benefit:**
  - Good for sequential files
- **Issues:**
  - External fragmentation will occur, compaction needed
  - File size is predefined

Contiguous File Allocation

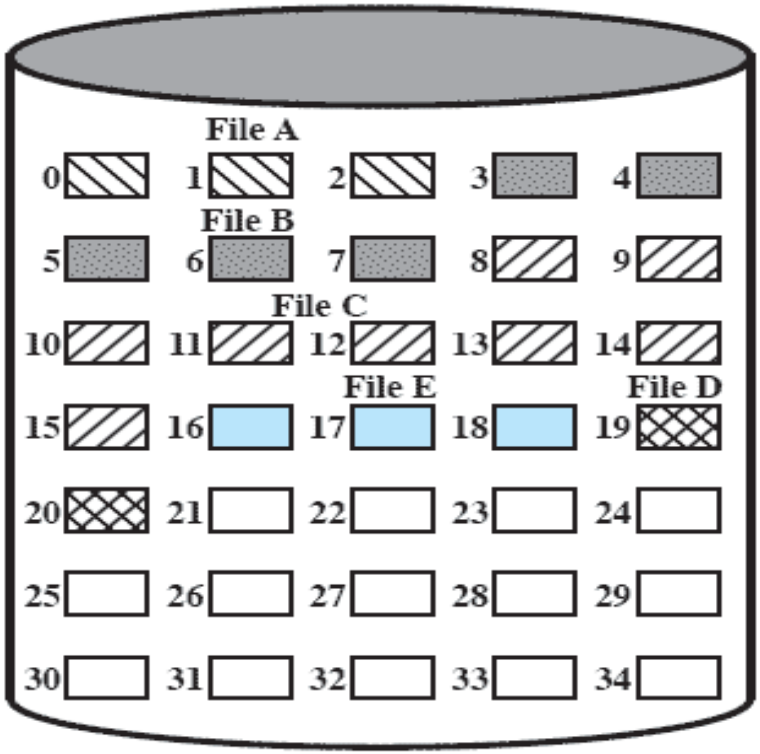


File Allocation Table

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

Figure 12.7 Contiguous File Allocation

External fragmentation



File Allocation Table

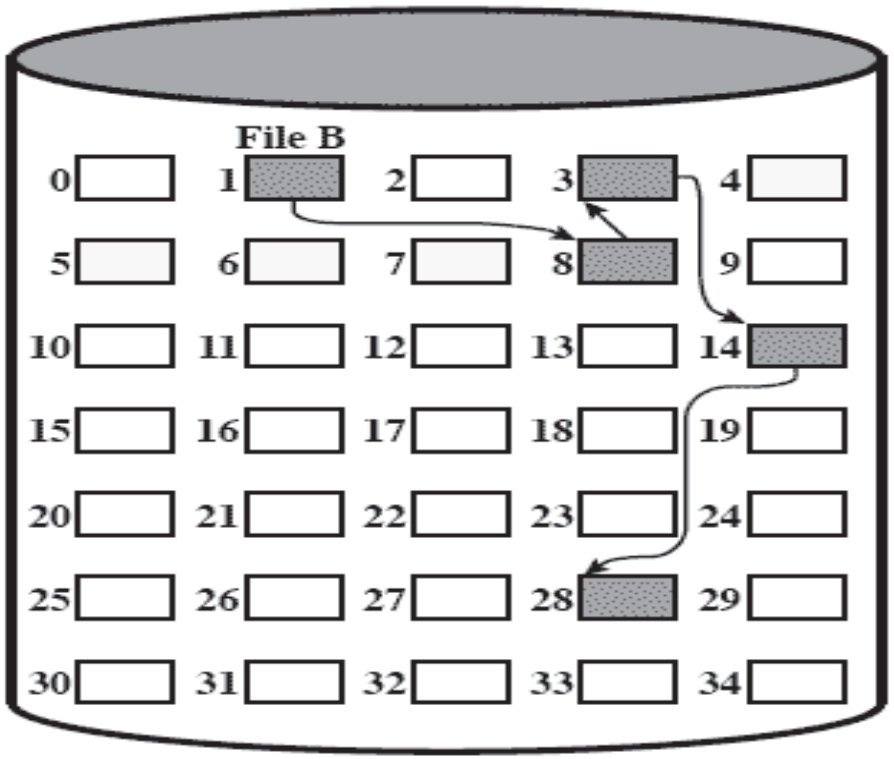
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.8 Contiguous File Allocation (After Compaction)

Chained Allocation

- Allocation on 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
- **Benefits:**
  - Any free block can be utilized
  - No external fragmentation
- **Issues:**
  - No accommodation of POL – Consolidation needed to solve this issue

Chained Allocation

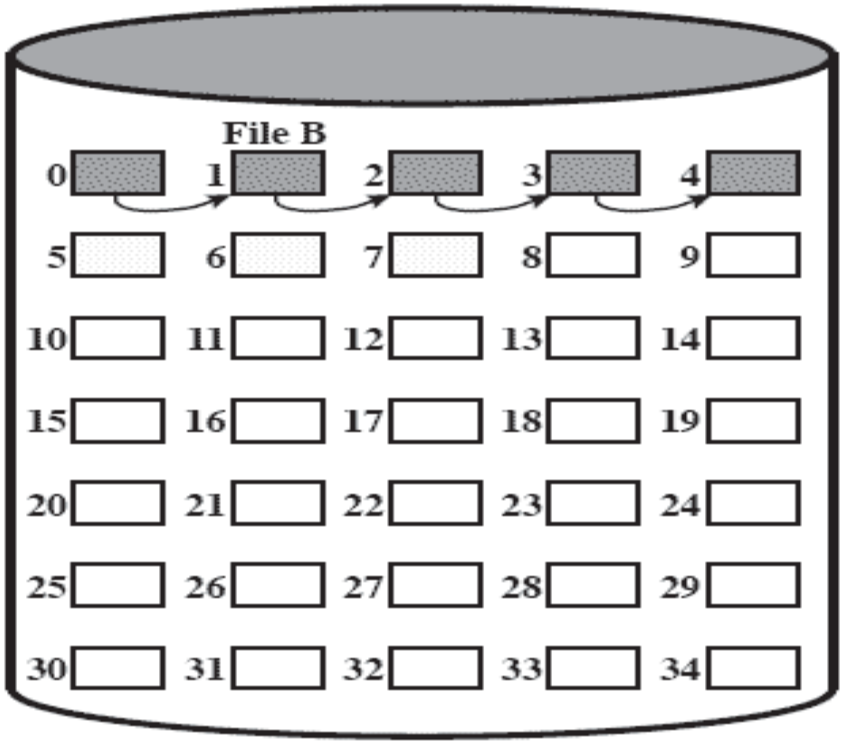


File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.9 Chained Allocation

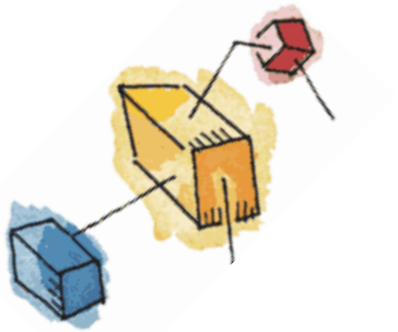
Chained Allocation Consolidation



File Allocation Table

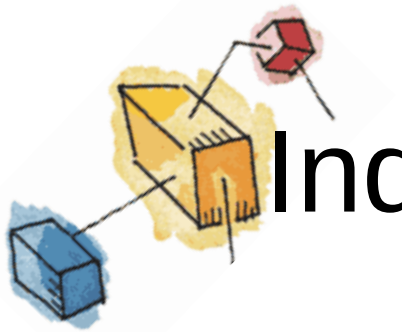
File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.10 Chained Allocation (After Consolidation)



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
- Index is kept in a separate block
  - The file allocation table contains block number for the index



Indexed Allocation Method

- **Allocation** may be either
  - Fixed size blocks or
  - Variable sized blocks

- Allocating by blocks eliminates external fragmentation
- Variable sized blocks improves locality
- Both cases require occasional consolidation

Indexed allocation with Block Portions

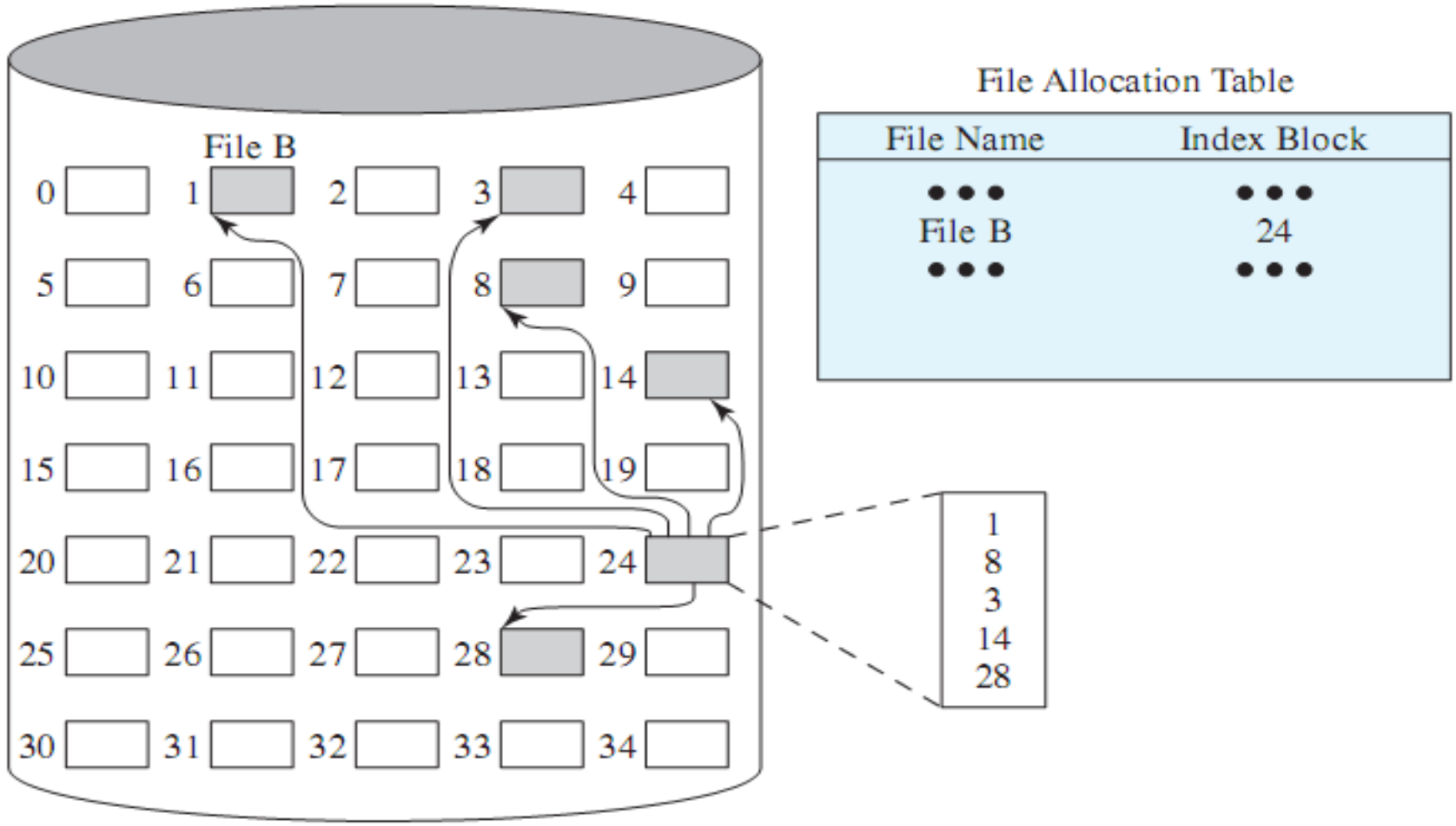


Figure 12.11 **Indexed Allocation with Block Portions**

Indexed Allocation with Variable Length Portions

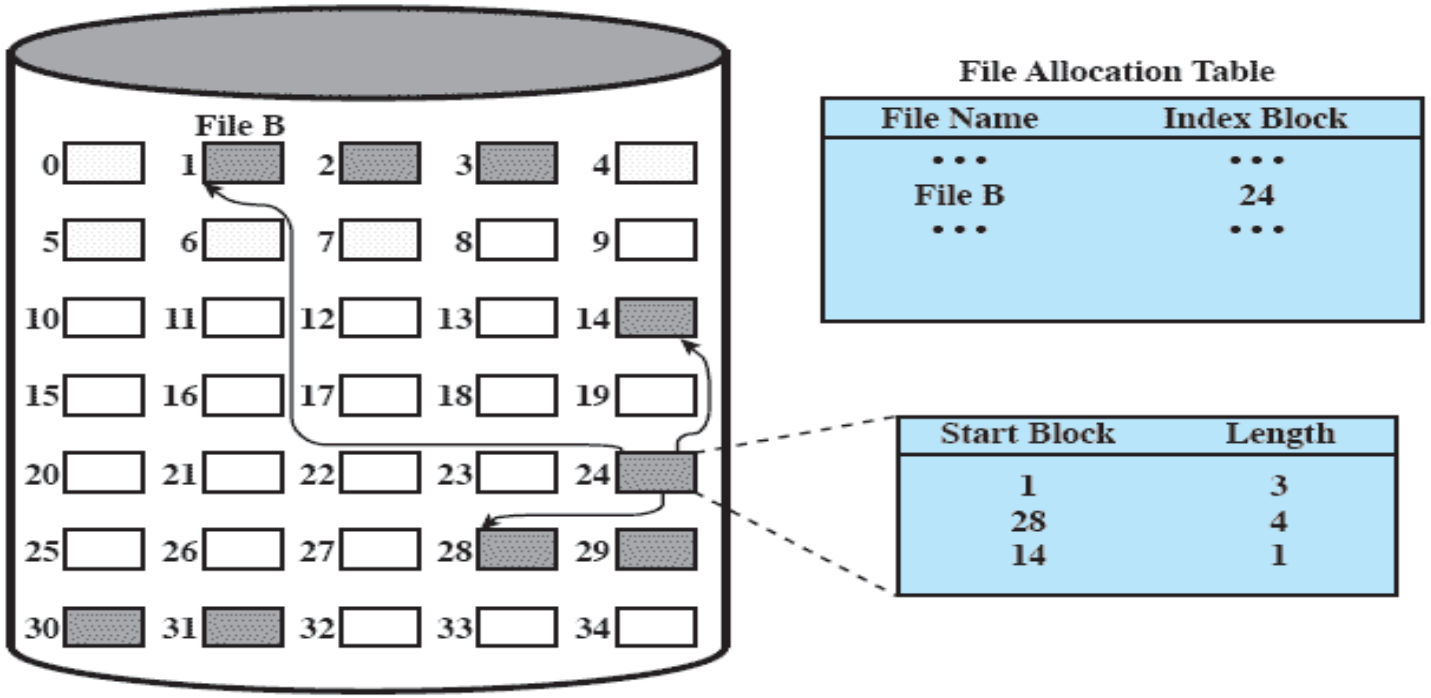
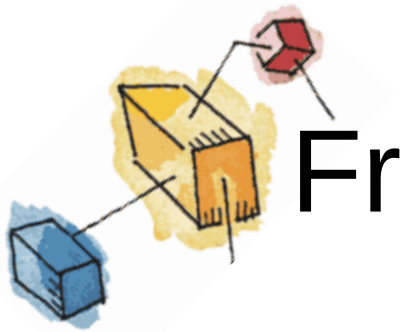
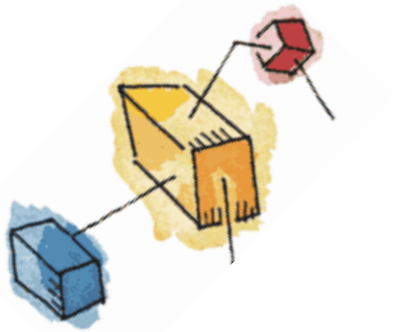


Figure 12.12 Indexed Allocation with Variable-Length Portions



Free Space Management

- Just as allocated space must be managed, so must the unallocated space
- To perform file allocation, we need to know which blocks are available.
- We need a disk allocation table in addition to a file allocation table



Bit Tables

- This method uses a vector containing one bit for each block on the disk.
- Each entry of a 0 corresponds to a free block,
  - and each 1 corresponds to a block in use.
- Advantages:
  - Works well with any file allocation method
  - Small as possible
- Issue:
  - Requires exhaustive search

Chained Free Portions

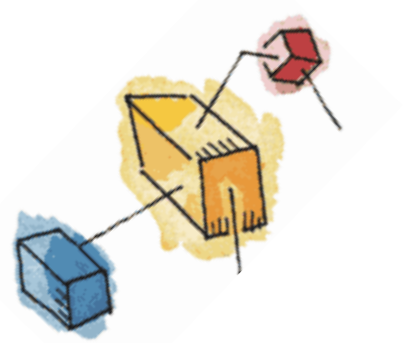
- The free portions may be chained together by using a pointer and length value in each free portion.
- Advantages:
  - Negligible space overhead
  - Suited to all file allocation methods
- Issue:
  - Leads to fragmentation

Indexing

- Treats free space as a file and uses one level index
- There is one entry in the index for every free portion on the disk
- This approach provides efficient support for all of the file allocation methods.

Free Block List

- Each block is assigned a number sequentially
  - the list of the numbers of all free blocks is maintained in a reserved portion of the disk.
- Size of free block list may be an issue
  - Still this is a better method
    - The space needed is usually less than 1% of the total disk space
    - List can be treated as a push down stack and keep only first few thousand elements in main memory
    - List can be treated as a FIFO queue and few thousand entries from head + tail in main memory



Volumes

- A collection of addressable sectors in secondary memory that an OS or application can use for data storage.

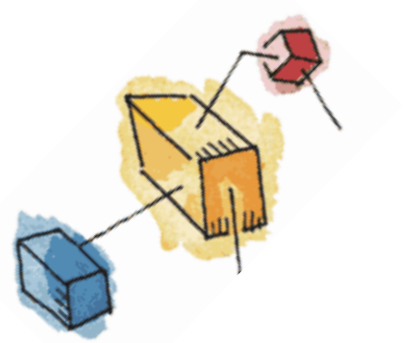
- A single disk is one volume
- Disk can be divided in partitions, where each partition works as a volume

Reliability

- Consider a scenario as below:
  - User A requests for updation of file
  - The request is granted, DAT and FAT updated in main memory
  - System crashes and restarts
  - User B requests file allocation and given the space overlapping with last allocation to A
  - User A accesses the overlapped portion

PROBLEM!!

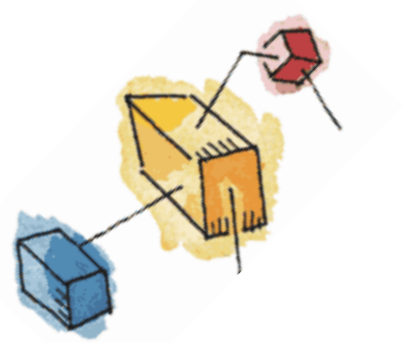
Copy of FAT and DAT in Main Memory



Reliability

- Solution:
  - Lock DAT on disk
  - Bring DAT copy to MM and look for space
  - Allocate space, update DAT, Update Disk
  - Update FAT, Update Disk
  - Unlock DAT

  - Overhead involved in this method
    - Reduced by using batch storage allocation



Access Control

- By successfully logging on to a system, the user is identified
- The OS can then enforce rules
  - Granting access to files and applications (or denying)
- The OS needs a rule-set to enforce

Access Matrix

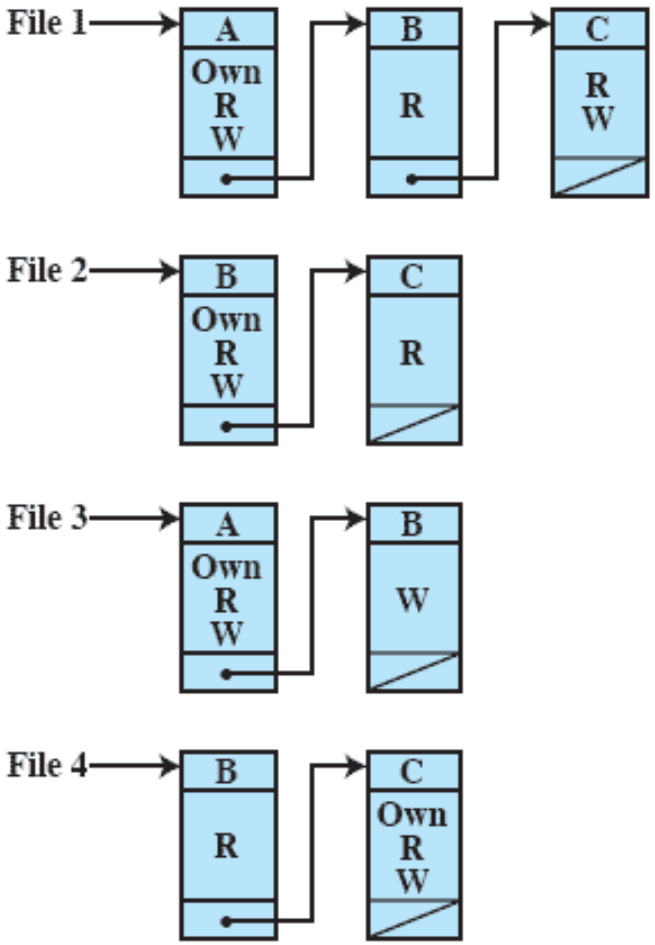
- One such rule set is an Access Matrix

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

(a) Access matrix

Access Control Lists

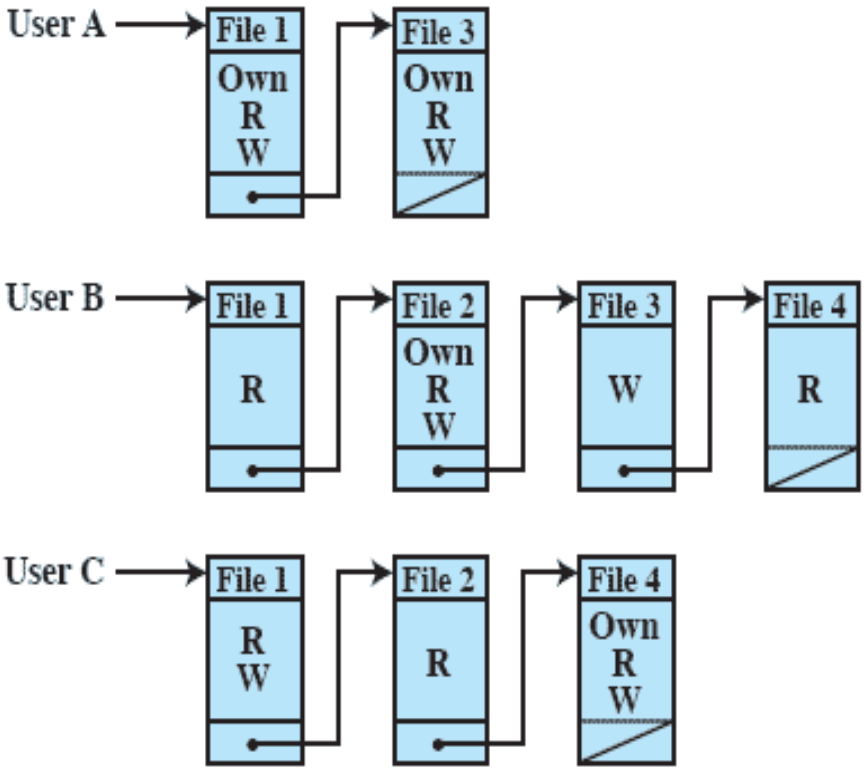
- A matrix may be decomposed by columns
- Giving an Access Control List (ACL) for each file.



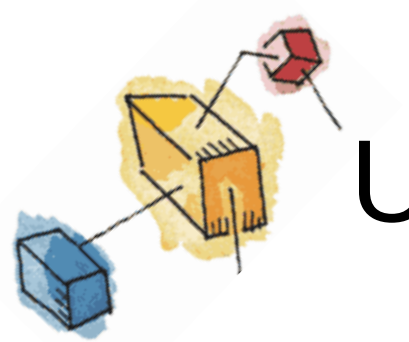
(b) Access control lists for files of part (a)

Capability Lists

- Decomposition by rows yields capability lists (or ticket)
  - specifies authorized objects and operations for a user.

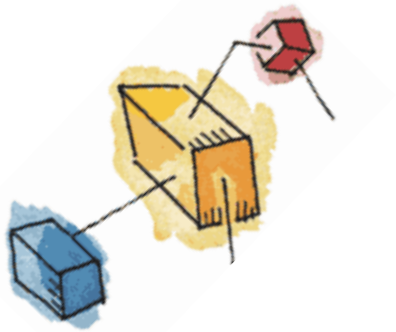


(c) Capability lists for files of part (a)



UNIX File Management

- Six types of files
  - Regular, or ordinary
  - Directory
  - Special
  - Named pipes
  - Links
  - Symbolic links

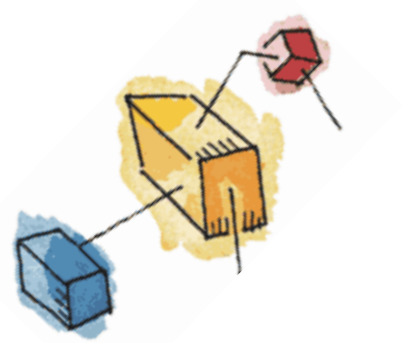


Inodes

- Index node
- Control structure that contains key information for a particular file.
- Each file is controlled by one inode
- Disk contains an inode table which in turn contains inodes of all files of the file system
- When a file is opened, its inode is loaded in main memory and stored in main memory resident inode table

Free BSD Inodes include:

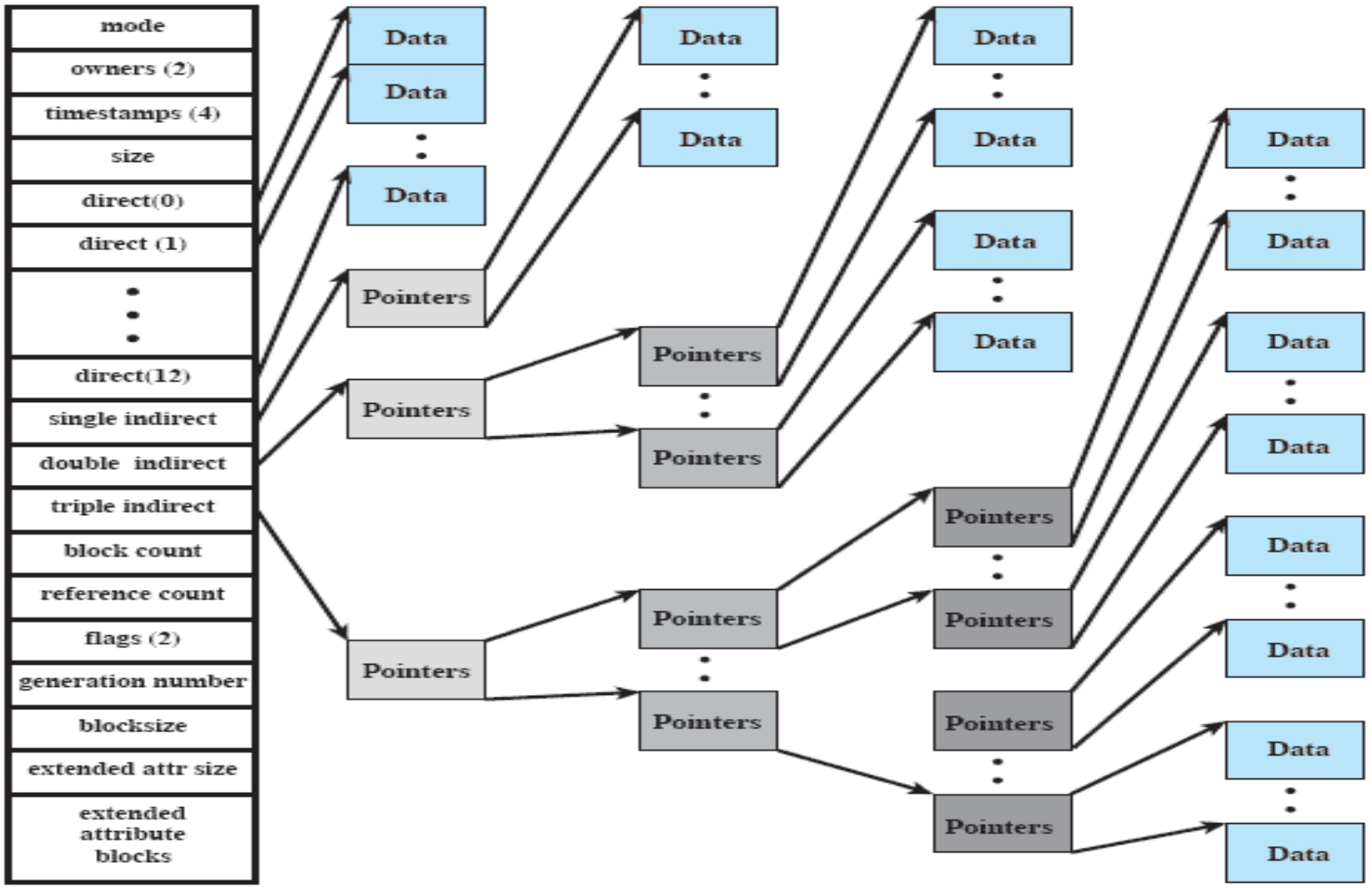
- The type and access mode of the file
- The file’s owner and group-access identifiers
- Creation time, last read/write time
- File size
- Sequence of block pointers
- Number of blocks and Number of directory entries
- Blocksize of the data blocks
- Kernel and user setable flags
- Generation number for the file
- Size of Extended attribute information
- Zero or more extended attribute entries



Inodes

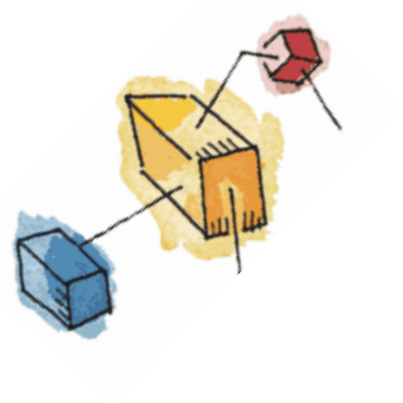
- Uses dynamic allocation on block basis
  - Blocks need not to be contiguous
  - Hence indexing method is used to keep track
- All UNIX implementations include a number of direct pointers to blocks and three indirect pointers
  - First 12 pointers point to 12 data blocks of file
  - IF the file needs more data blocks, then indirect pointers are used

FreeBSD Inode and File Structure



Advantages

- Fixed and relatively small size
  - Can be kept in main memory
- Small files can be accessed quickly



Thank You!!