1 Files and Systems

1.1 How to Virtualize Persistent Storage

1.2 File Systems

- Provides long term storage
- Roles
  - Implements an abstraction (files) for secondary storage
  - Organize files logically (directories)
  - Permit sharing of data between processes, people and machine
  - Protect data from unwanted access (security)
- Requirements
  - Store a large amount of information
  - Information must survive after termination of using it
  - Multiple process must be able to access it concurrently
- Has two views
  1. User View
     - Convinent organization of information
  2. OS View
     - Managing physical storage
     - Enforcing access restrictions

1.3 File Operations

- Creation
- Writing
- Reading
- Repositioning within a file
- Deleting a file
- Truncation and appending
  - File attributes are kept
  - May erase/add parts or entire contents of a file

1.4 File Access Methods

- Sequential Access
  - Read bytes one at a time, in order
- Direct Access
  - Random access given block/byte number

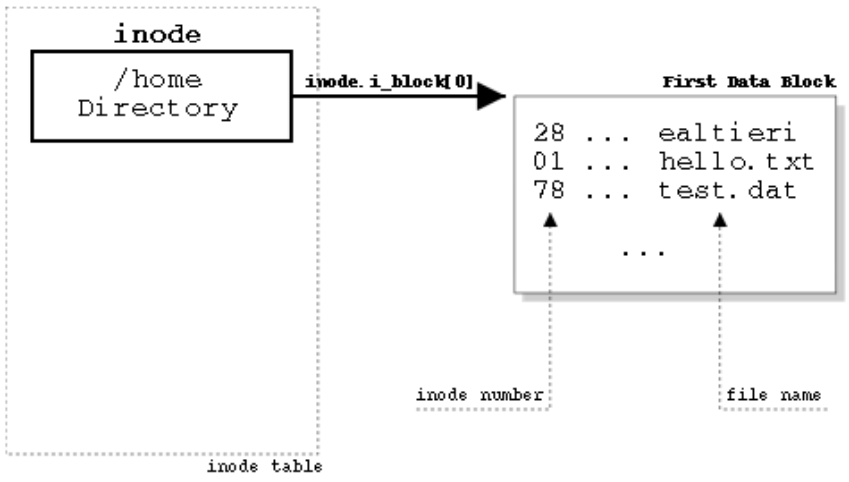
1.5 Directories

- Purpose
  - Provides logical structure to file system
    - Users → A way to organize file
    - File system → Convinent naming interface (programmers dont't get lost)

1.6 Directory Structure

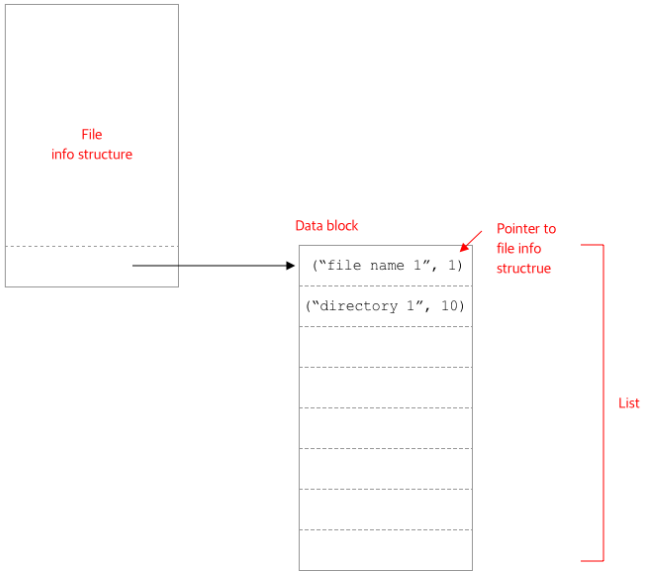
- Is a list of entries - names and associated metadata
  - List usually unordered
  - Metadata is like a label to a box
  - e.g size, protection, location, i-node number

Example



1.7 Directory Implementations

- Acyclic graph directories
- List
  - Simple list of file names and pointers to file info structure



- Hash table
  - Creates a list of file info structure
  - Hash file name to get a pointer to the file info structrue

    Example

    ```
    directory["file-name"]
    ```

  - Takes more space (hash table)

1.8 Hard Links

- Is a directory entry that associates a name with a file on a file system.
- Can't create new hard links to directories
  - May create cycles (Not good)
  - Breaks directory tree (Tree → no cycle!!)
  - Breaks unambiguity of parent directories
  - They multiply files
- Can't create hard links across partitions
- Removing a hard link only removes the file if it is the last link to file

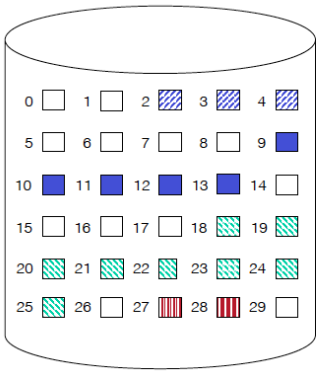
1.9 Symbolic Link

- Is a directory entry that associates a name with another directory entry on a file system.
- Can create a link to diretory
- Can create a link across partitions
- File system needs to look up a link to follow it
- Removing a file → Dangling link

1.10 Associating Data Blocks to Files

1.11 Disk Layout Strategies

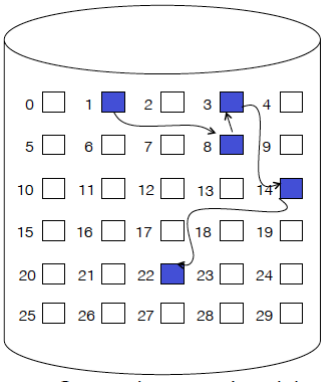
- Contiguous Allocation



File Name	Start Blk	Length
File A	2	3
File B	9	5
File C	18	8
File D	27	2

  - Each file is given one (or more) set of contiguous set of blocks
  - In info structure, stores
    - Starting Block
    - Length
  - Pros and Cons
    - Fast, simplifies directory acccess
    - Allows indexing
    - Inflexible
      - Need to know size of data blocks beforehand (when only one extent is allowed)
    - Causes external fragmentation
  - Example
    - Microsoft NTFS
    - Linux's ext4

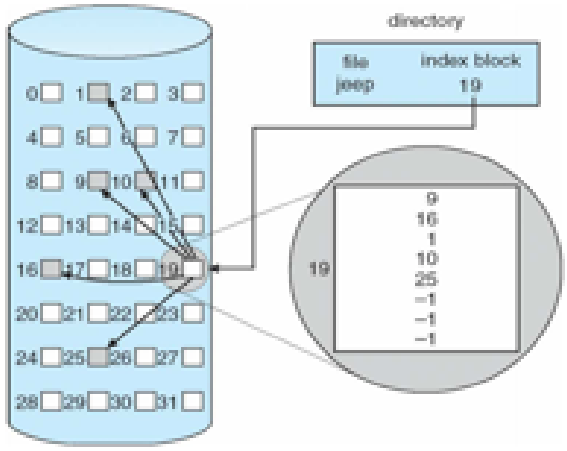
- Linked Allocation



File Name	Start Blk	Last Blk
...	...	...
File B	1	22
...	...	...

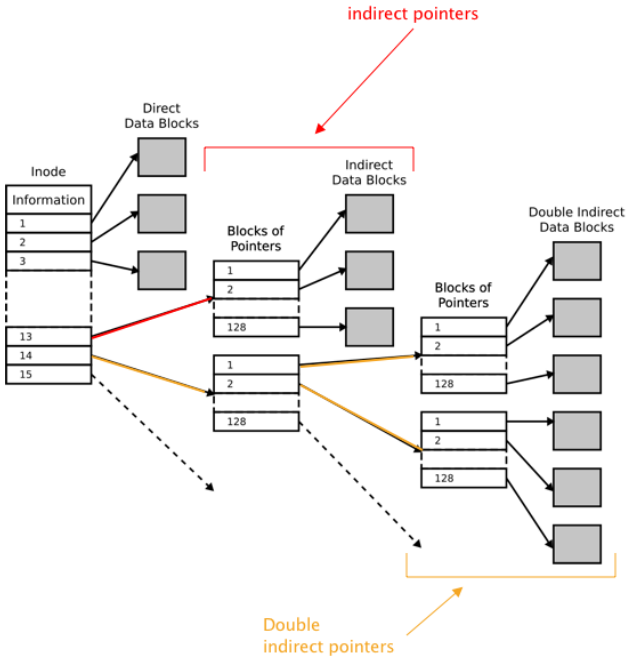
  - Each block in file contains a pointer to next block
  - In info structure, stores
    - Starting Block
    - Length
  - Pros and Cons
    - Good for sequential access
    - Bad for others
  - Example
    - Microsoft FAT

- Indexed Allocation



  - All file metadata is stored in an inode
  - Each inode contains 15 pointers

    * 12 direct pointers
    * 1 single indirect pointer
        · Address of block containing address of data blocks
    * 1 double indirect pointer
    * 1 triple indirect pointer



– Pros and Cons
    * No external fragmentation
    * Handles random access better
    * Files can be easily grown
    * Requires many intermediary files
– Example
    * Linux's ext2