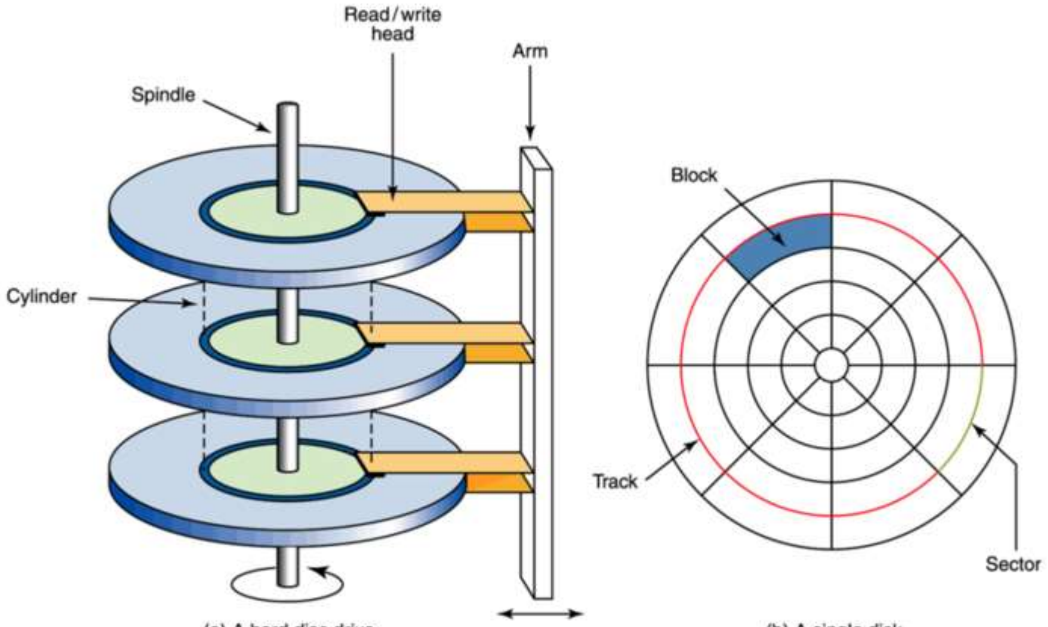
7th Seminar

File System

- **File System Manager**: set of OS Services that provides Files and Directories for user applications.
- **File**: named, ordered collection of information
- **File System**: set of files and directories contained on a single drive. The raw data on the drive is translated to this abstract view of files and directories by the file system manager according to the specification of the file system standard.
- The file manager administers the file system by:
    - Storing the information on a device
    - Mapping the block storage to a logical view
    - Allocating/deallocating storage
    - Providing directories

File System



(a) A hard disc drive (b) A single disk

- **sector**: a physical space on a disk that holds information. When a disk is formatted, tracks are defined. Each track is divided into slices, which are sectors. On hard drives each sector can hold 512 bytes of data.
- **block**: a group of sectors that the operating system can address (point to). It may be one sector, or several sectors (2,4,8, or even 16).

File Systems

- **file**: stored on a storage medium. Each storage is a linear space for reading or both reading and writing digital information. Each byte of information on it has its offset from the storage start address and is referenced by this address. A storage can be presented as a grid with a set of numbered cells (each cell is a single byte). Any file saved to the storage gets its own cells.
- **file system**: a process that manages how and where data is stored, accessed and managed on a storage disk. It is a logical disk component that manages a disk's internal operations as it relates to a computer and is abstract to a human user.

File Systems

- There are several common approaches to storing information on disk. However, in comparison to memory management, there are some key differences in managing disk media:
  - Data can only be written in fixed size chunks.
  - Access times are different for different locations on the disk. Seeking is usually a costly operation.
  - Data throughput is very small compared to RAM
  - Data commonly has to be maintained

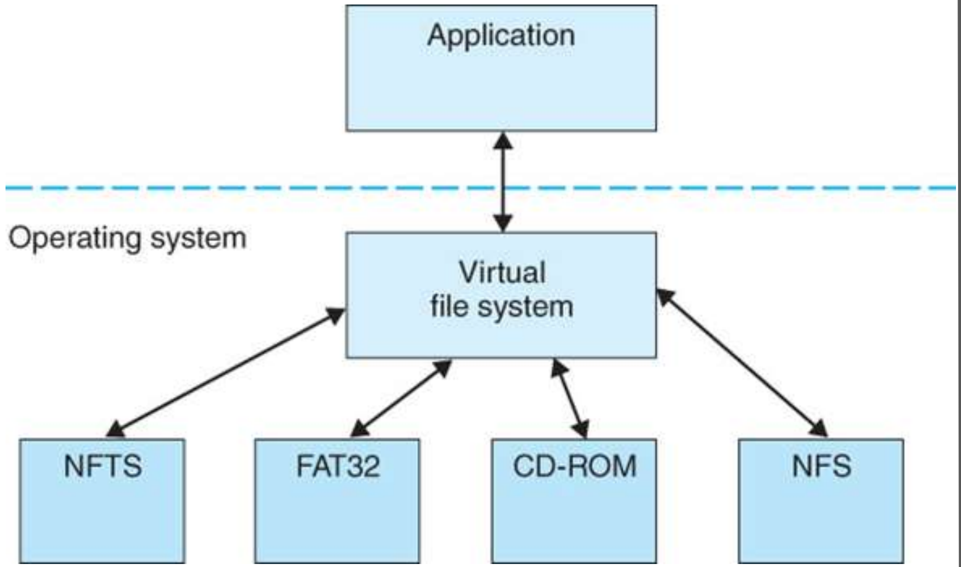
File System

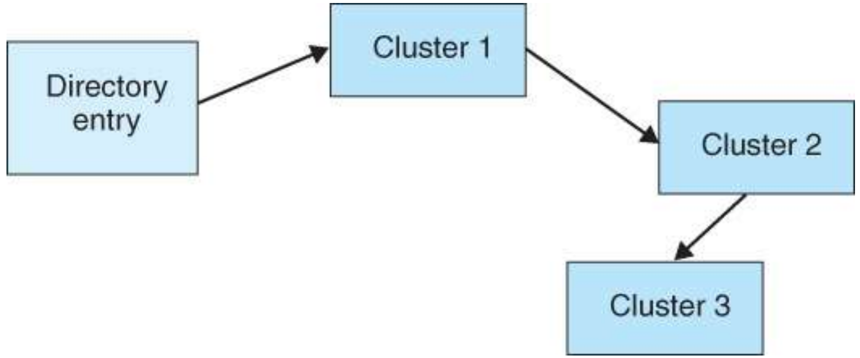
- the goal of a filesystem is to allow logical groups of data to be organized into files, which can be manipulated as a unit. In order to do this, the filesystem must provide some sort of index of the locations of files in the actual secondary storage. The fundamental operations of any file system are:
    - Tracking the available storage space
    - Tracking which block or blocks of data belong to which files
    - Creating new files
    - Reading data from existing files into memory
    - Updating the data in the files
    - Deleting existing files

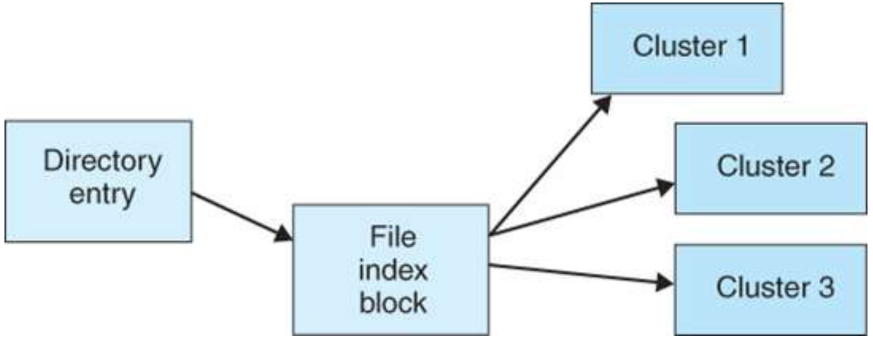
File System

- there are other features which go along with a practical filesystem:
    - Assigning human-readable names to files, and renaming files after creation
    - Allowing files to be divided among non-contiguous blocks in storage, and tracking the parts of files even when they are fragmented across the medium
    - Providing some form of hierarchical structure, allowing the files to be divided into directories or folders
    - Buffering reading and writing to reduce the number of actual operation on the physical medium
    - Caching frequently accessed files or parts of files to speed up access
    - Allowing files to be marked as 'read-only' to prevent unintentional corruption of critical data
    - Providing a mechanism for preventing unauthorized access to a user's files

File System Standards







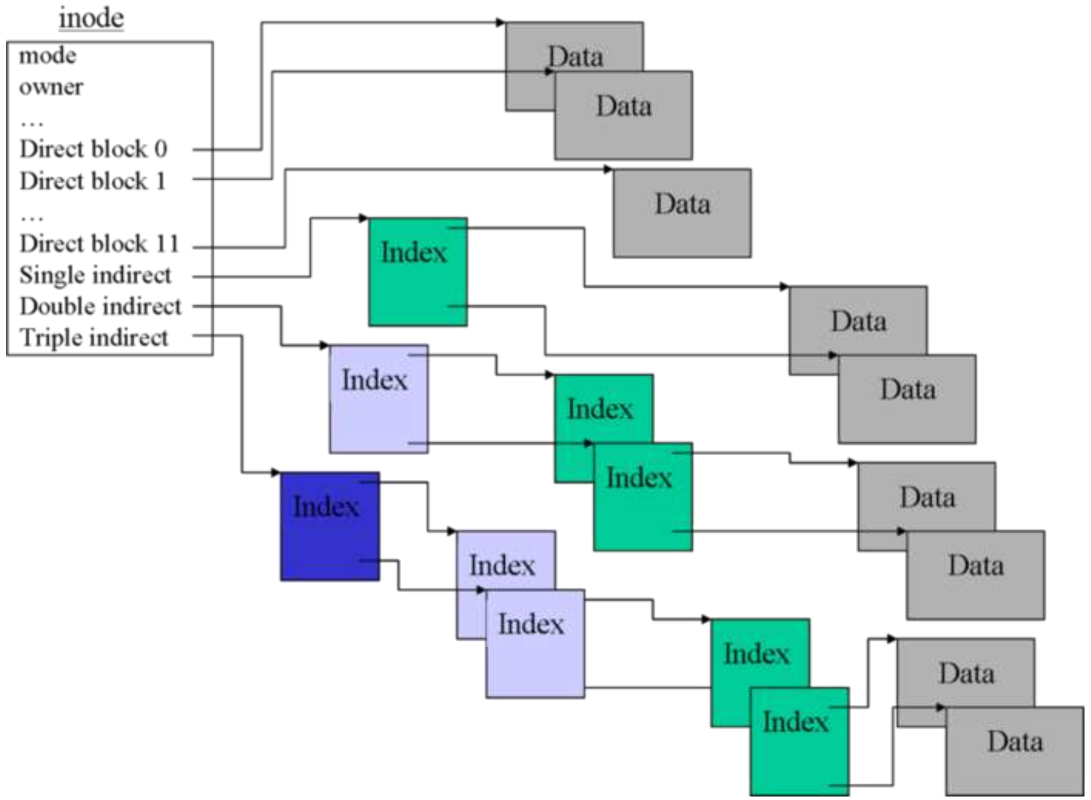
FS Indexing

- There are several methods of indexing the contents of files, with the most commonly used being i-nodes and File Allocation Tables.
    - **inodes** (information nodes/index nodes) are a design element in most Unix file systems: Each file is made of data blocks (the sectors that contains your raw data bits), index blocks (containing pointers to data blocks so that you know which sector is the nth in the sequence), and one inode block.The inode is the root of the index blocks, and can also be the sole index block if the file is small enough. Moreover, as Unix file systems support hard links (the same file may appear several times in the directory tree), inodes are a natural place to store Metadata such as file size, owner, creation/access/modification times, locks, etc.
    - The **File Allocation Table** (FAT) is the primary indexing mechanism for MS-DOS and it's descendants. There are several variants on FAT, but the general design is to have a table (actually a pair of tables, one serving as a backup for the first in case it is corrupted) which holds a list of blocks of a given size, which map to the whole capacity of the disk

I-node

- In Unix based operating system each file is indexed by an Inode. Inode are special disk blocks that are created when the file system is created. The number of Inode limits the total number of files/directories that can be stored in the file system. The Inode contains the following information:
  - Administrative information (file type, userId, groupId, size, permissions, timestamps, etc).
  - A number of direct blocks (typically 12) that contains the first 12 blocks of the files.
  - A single indirect pointer that points to a disk block which in turn is used as an index block, if the file is too big to be indexed entirely by the direct blocks.
  - A double indirect pointer that points to a disk block which is a collection of pointers to disk blocks which are index blocks, used if the file is too big to be indexed by the direct and single indirect blocks.
  - A triple indirect pointer that points to an index block of index blocks of index blocks.

I-node


inode
mode
owner
…
Direct block 0
Direct block 1
…
Direct block 11
Single indirect
Double indirect
Triple indirect
Data
Data
Data
Index
Data
Data
Index
Index
Index
Data
Data
Index
Index
Index
Index
Index
Data
Data

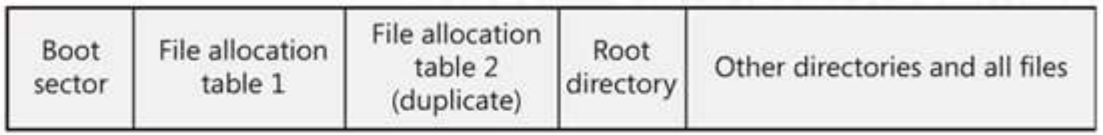
FAT

- a table maintained on a hard disk by MS-DOS and Microsoft Windows operating systems that acts as a table of contents, showing where directories and files are stored on the disk.
- resides at the beginning of the volume. To protect the volume, two copies of the table are kept, in case one becomes damaged. In addition, the file allocation tables and the root folder must be stored in a fixed location so that the files needed to start the system can be correctly located.
- the major versions of the FAT format are named after the number of table element bits: 12 (FAT12), 16 (FAT16), and 32 (FAT32); each of these variants is still in use.

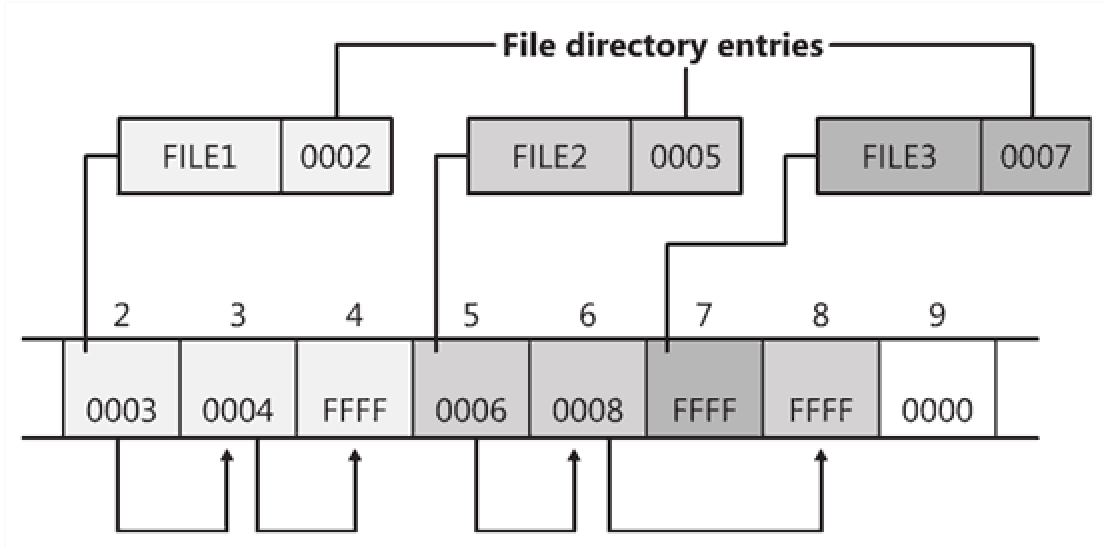
FAT

- a structure that maps the locations of the clusters in which files and folders are stored on the disk. The FAT records the location of each cluster that makes up a given file and the sequence in which it is stored. This is necessary because files are usually not stored in a contiguous location on a hard disk because of the presence of disk fragmentation caused by the creation and deletion of files on the disk.
- For each file on a FAT volume, the FAT contains the entry point for the allocation unit in which the first segment of the file is stored, followed by a series of links called the allocation chain. The allocation chain indicates where succeeding segments of the file are located and is then terminated by an end-of-file (EOF) marker.

FAT


Boot sector
File allocation table 1
File allocation table 2 (duplicate)
Root directory
Other directories and all files

- 0x0000 - unused
- 0xFFF8-0xFFFF last cluster in a file for FAT16
- files are given the first available location on the volume



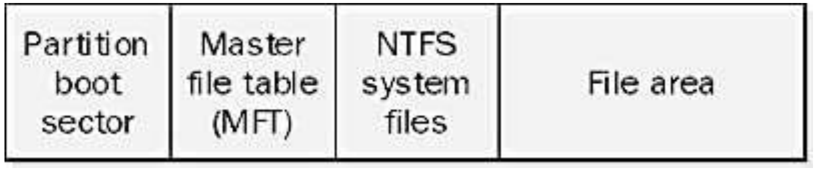
NTFS

- the primary file system for recent versions of Windows and Windows Server—provides a full set of features including security descriptors, encryption, disk quotas, and rich metadata, and can be used with Cluster Shared Volumes (CSV) to provide continuously available volumes that can be accessed simultaneously from multiple nodes of a failover cluster
- uses its log file and checkpoint information to restore the consistency of the file system when the computer is restarted after a system failure. After a bad-sector error, NTFS dynamically remaps the cluster that contains the bad sector, allocates a new cluster for the data, marks the original cluster as bad, and no longer uses the old cluster. For example, after a server crash, NTFS can recover data by replaying its log files

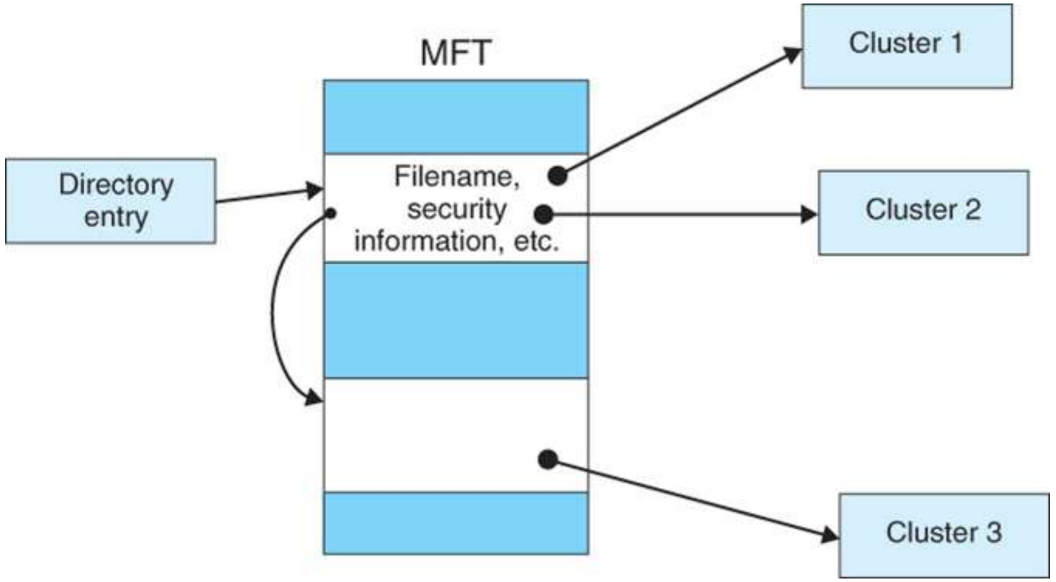
NTFS

- Access Control List (ACL)-based security for files and folders—NTFS allows to set permissions on a file or folder, specify the groups and users whose access you want to restrict or allow, and select access type.
- Support for BitLocker Drive Encryption — provides additional security for critical system information and other data stored on NTFS volumes. BitLocker provides support for device encryption on x86 and x64-based computers with a Trusted Platform Module (TPM) that supports connected stand-by
- Support for large volumes — can support volumes up to 256 terabytes. Supported volume sizes are affected by the cluster size and the number of clusters

NTFS



- Each record is exactly 1 KB in size. The first 42 bytes in the header have a fixed structure (12 fields), while the rest of the record is used to store attributes such as the file name or system attributes. The number of attributes as well as the size of each attribute can vary



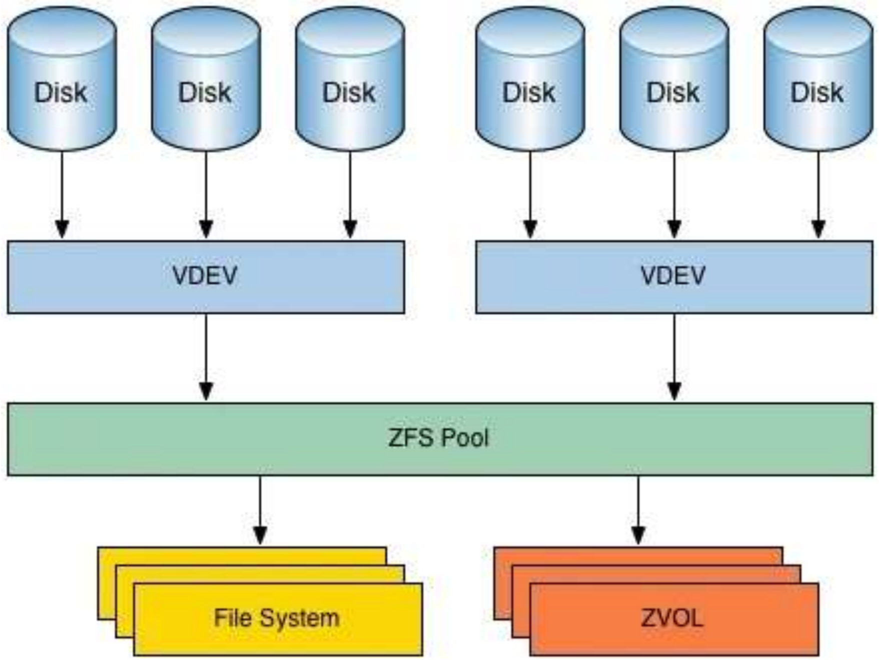
ZFS

- The Z File System is an advanced file system designed to overcome many of the major problems found in previous designs. Has 3 major design goals:
  - Data integrity: All data includes a checksum of the data. When data is written, the checksum is calculated and written along with it. When that data is later read back, the checksum is calculated again. If the checksums do not match, a data error has been detected
  - Pooled storage: physical storage devices are added to a pool, and storage space is allocated from that shared pool. Space is available to all file systems, and can be increased by adding new storage devices to the pool
  - Performance: multiple caching mechanisms provide increased performance. ARC (advanced memory-based read cache), a second level of disk-based read cache (added with L2ARC), and disk-based synchronous write cache (with ZIL)

ZFS

- The major building blocks to understand are zpools, vdevs, and devices.
  - The zpool is the uppermost ZFS structure. A zpool contains one or more vdevs, each of which in turn contains one or more devices. Zpools are self-contained units—one physical computer may have two or more separate zpools on it, but each is entirely independent of any others. Zpools cannot share vdevs with one another
  - ZFS redundancy is at the vdev level, not the zpool level. There is absolutely no redundancy at the zpool level—if any storage vdev or SPECIAL vdev is lost, the entire zpool is lost with it
  - Each zpool consists of one or more vdevs(short for virtual device). Each vdev, in turn, consists of one or more real devices. Most vdevs are used for plain storage, but several special support classes of vdev exist as well—including CACHE, LOG, and SPECIAL. Each of these vdev types can offer one of five topologies—single-device, RAIDz1, RAIDz2, RAIDz3, or mirror

ZFS


Disk
Disk
Disk
Disk
Disk
Disk
VDEV
VDEV
ZFS Pool
File System
ZVOL

Crash Resistance

- Crashes can happen anywhere, even in the middle of critical sections:
    - Lost data: information cached in main memory may not have been written to disk yet.
        - in original Unix: up to 30 seconds worth of changes
    - Inconsistency:
        - If a modification modifies multiple blocks, a crash could occur when some of the blocks have been written to disk but not the others.
        - Examples:
            - Adding block to file: free list was updated to indicate block in use, but file descriptor wasn't yet written to point to block.
            - Creating link to a file: new directory entry refers to file descriptor, but reference count wasn't updated in file descriptor.
- Ideally, we'd like something like an atomic operation where multi-block operations happen either in their entirety or not at all.

Crash Resistance - fsck

- check consistency during reboot, repair problems
  - Unix fsck ("file system check")
    - During every system boot fsck is executed.
    - Checks to see if disk was shut down cleanly; if so, no more work to do.
    - If disk didn't shut down cleanly (e.g., system crash, power failure, etc.), then scan disk contents, identify inconsistencies, repair them.
    - Ex: block in file and also in free list
    - Ex: reference count for a file descriptor doesn't match the number of links in directories
    - Ex: block in two different files
    - Ex: file descriptor has a reference count > 0 but is not referenced in any directory.
  - Limitations of fsck:
    - Restores disk to consistency, but doesn't prevent loss of information
    - Security issues: a block could migrate from the password file to some random file.
    - Can take a long time: 1.5 hours to read every block in a medium-size disk today. Can't restart system until fsck completes. As disks get larger, recovery time increases.

Crash Resistance - ordered writes

- ordered writes
  - Prevent certain kinds of inconsistencies by making updates in a particular order.
  - For example, when adding a block to a file, first write back the free list so that it no longer contains the file's new block.
  - Then write the file descriptor, referring to the new block.
  - What can we say about the system state after a crash?
  - In general:
    - Never write a pointer before initializing the block it points to (e.g., indirect block).
    - Never reuse a resource (inode, disk block, etc.) before nullifying all existing pointers to it.
    - Never clear last pointer to a live resource before setting new pointer (e.g. mv).
  - Result: no need to wait for fsck when rebooting
  - Problems:
    - Can leak resources (run fsck in background to reclaim leaked resources).
    - Requires lots of synchronous metadata writes, which slows down file operations.

Crash Resistance - journaling

- write-ahead logging (also called journaling file systems)
    - Implemented in Linux ext3 and NTFS (Windows).
    - Similar in function to logs in database systems; allows inconsistencies to be corrected quickly during reboots
    - Before performing an operation, record information about the operation in a special append-only log file; flush this info to disk before modifying any other blocks.
        - Example: adding a block to a file
        - Log entry: "I'm about to add block 99421 to file descriptor 862 at block index 93"
        - Then the actual block updates can be carried out later.
        - If a crash occurs, replay the log to make sure all updates are completed on disk.
        - Guarantees that once an operation is started, it will eventually complete.
    - Problem: log grows over time, so recovery could be slow.

Crash Resistance - journaling

- Solution #1: checkpoint
  - Occasionally stop and flush all dirty blocks to disk.
  - Once this is done, the log can be cleared.
- Solution #2: keep track of which parts of the log correspond to which unwritten blocks; as blocks get written to disk, can gradually delete old portions of the log that are no longer needed.
- Typically the log is used only for metadata (free list, file descriptors, indirect blocks), not for actual file data.

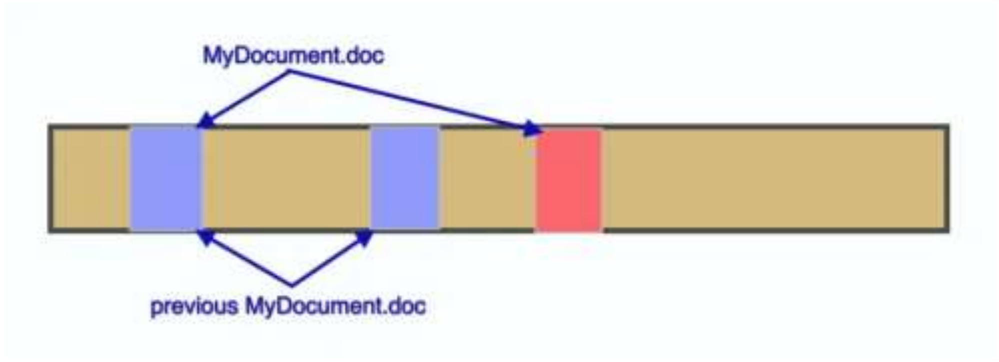
- Logging advantages:
  - Recovery much faster.
  - Eliminate inconsistencies such as blocks confused between files.
  - Log can be localized in one area of disk, so writes are faster (no seeks).
  - Metadata writes can be delayed a long time, for better performance.
- Logging disadvantages:
  - Synchronous disk write before every metadata operation.

Crash Resistance copy-on-write

- The majority of writes to disk occur when saving modified versions of existing files. Usually, if possible, these changes are written to the same storage blocks already occupied by the files. This minimises the changes required to the disk.
- If anything goes wrong in overwriting the revised version of the file, normally both the original and revised versions are corrupted or lost. If keeping the original version we end up with two complete copies of the file, which wastes space if they only differ in a few words. Various solutions have been proposed to this, of which copy on write is one.

Crash Resistance copy-on-write

- In copy on write, those changes are written out to a newly-allocated block on the disk, not the original. At that time, there are effectively two copies of the file on the disk: the two original blue blocks, which together make up the first version, and the first blue block and the pink one, making up the revised file.


MyDocument.doc
previous MyDocument.doc