File Management Systems

FAT, EXT, HFS

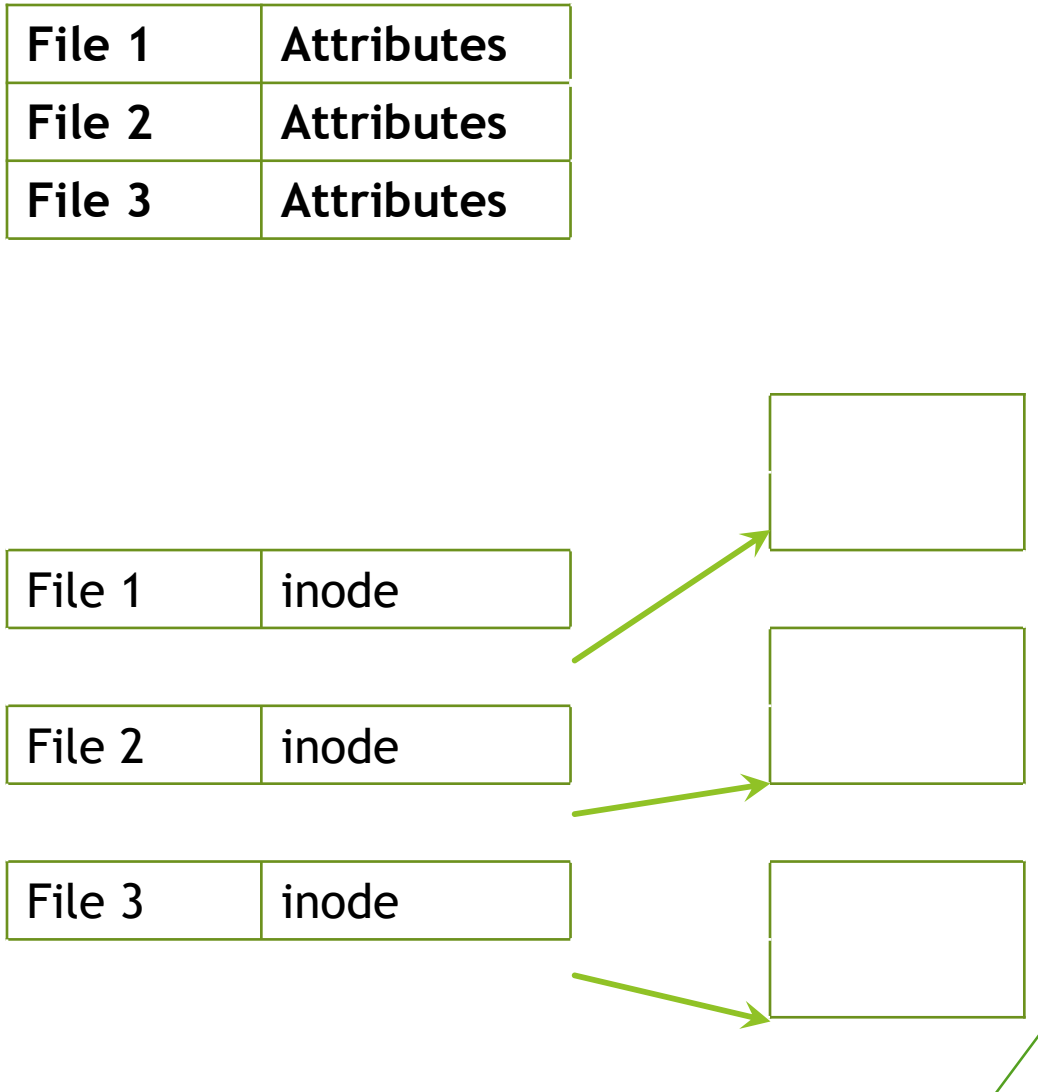
Table of Contents

Directory Implementation

- A directory contains information on file and its attributes.
- Assists in locating file on hard disk.
- Can be implemented as
  - Fixed length records:
  - Variable length records:

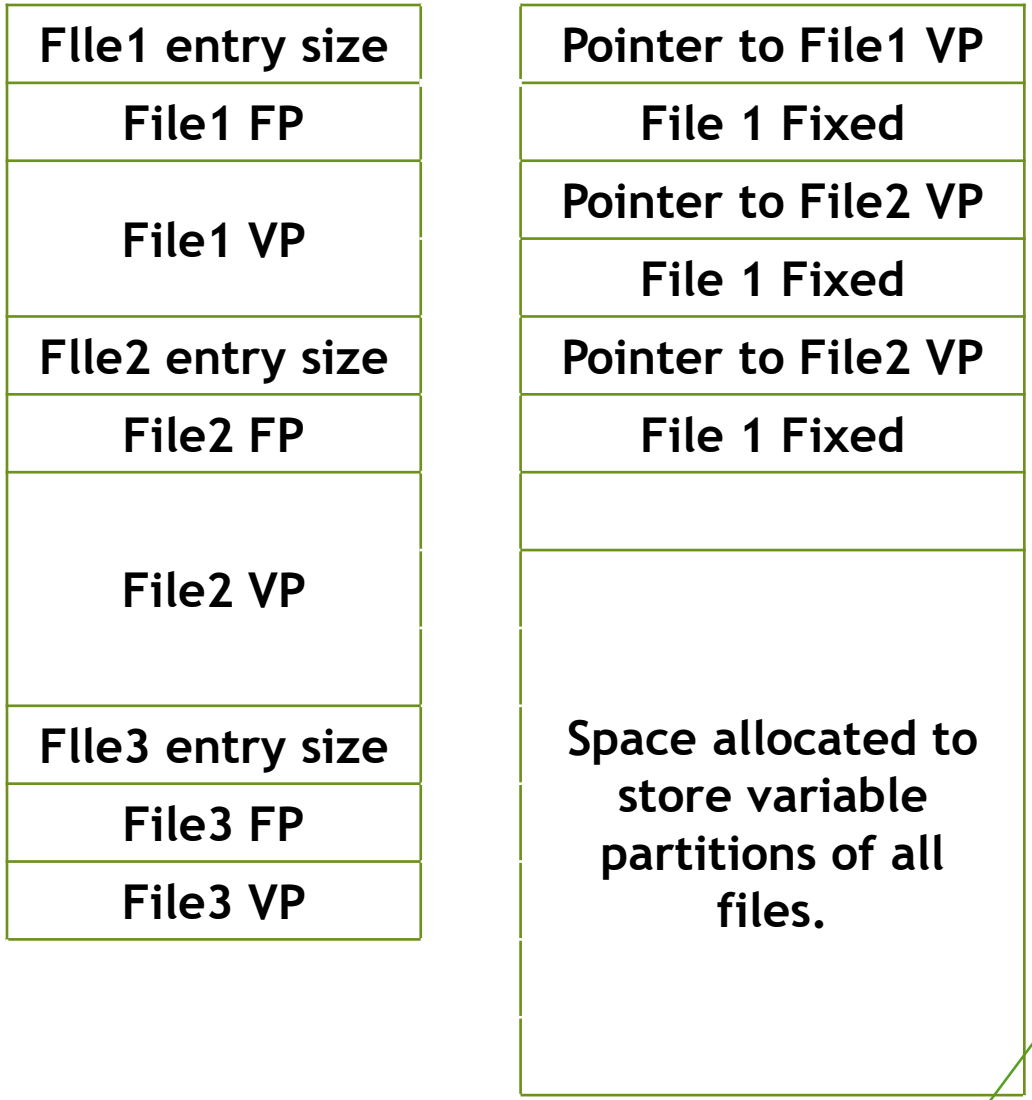
Fixed Length Records

- Each entry contains file name and all attributes; every attribute is allotted a fixed number of bytes. Filename is restricted by number of bytes allowed. Shown in top diagram on RHS.
- In another case, each entry contains filename and a pointer to a block (eg inode) containing attributes. Shown in bottom diagram on RHS.



Directory Implementation

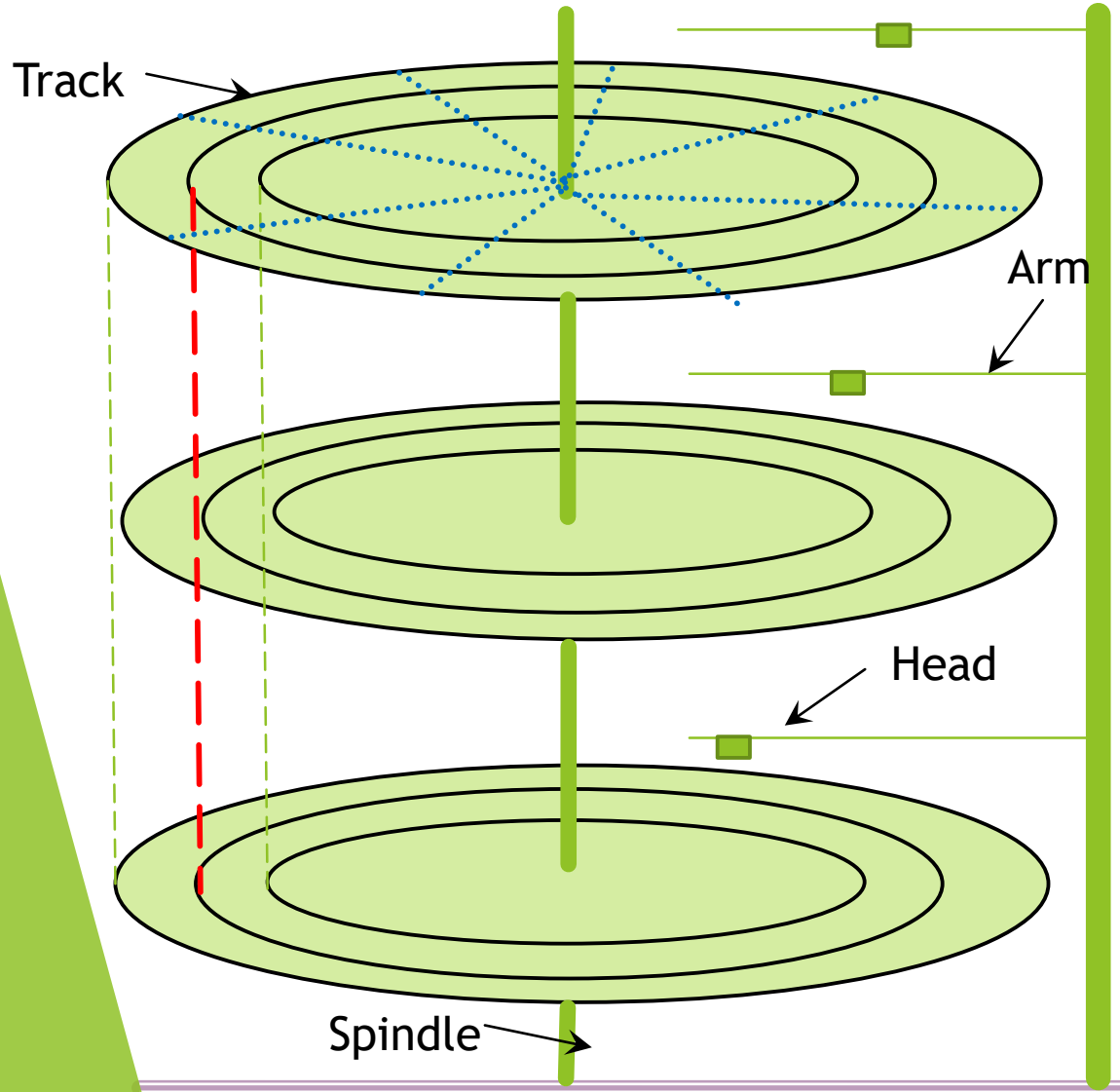
- Each record consists of fixed partition (to store attributes of fixed size such as length) and variable partition (to store variable length attributes such as name). Records. This structure allows file to have longer file names,
- One possible way is to store length of record, fixed partition followed by variable partition. (Left figure)
- Another possible way is to reserve some heap area to store variable partitions of each file. Each fixed partition entry contains pointer to variable partition and fixed size attributed. (Right figure)
- FP: Fixed size partition (used to store fixed length attributed)
- VP: Variable size partition



Search in Directory

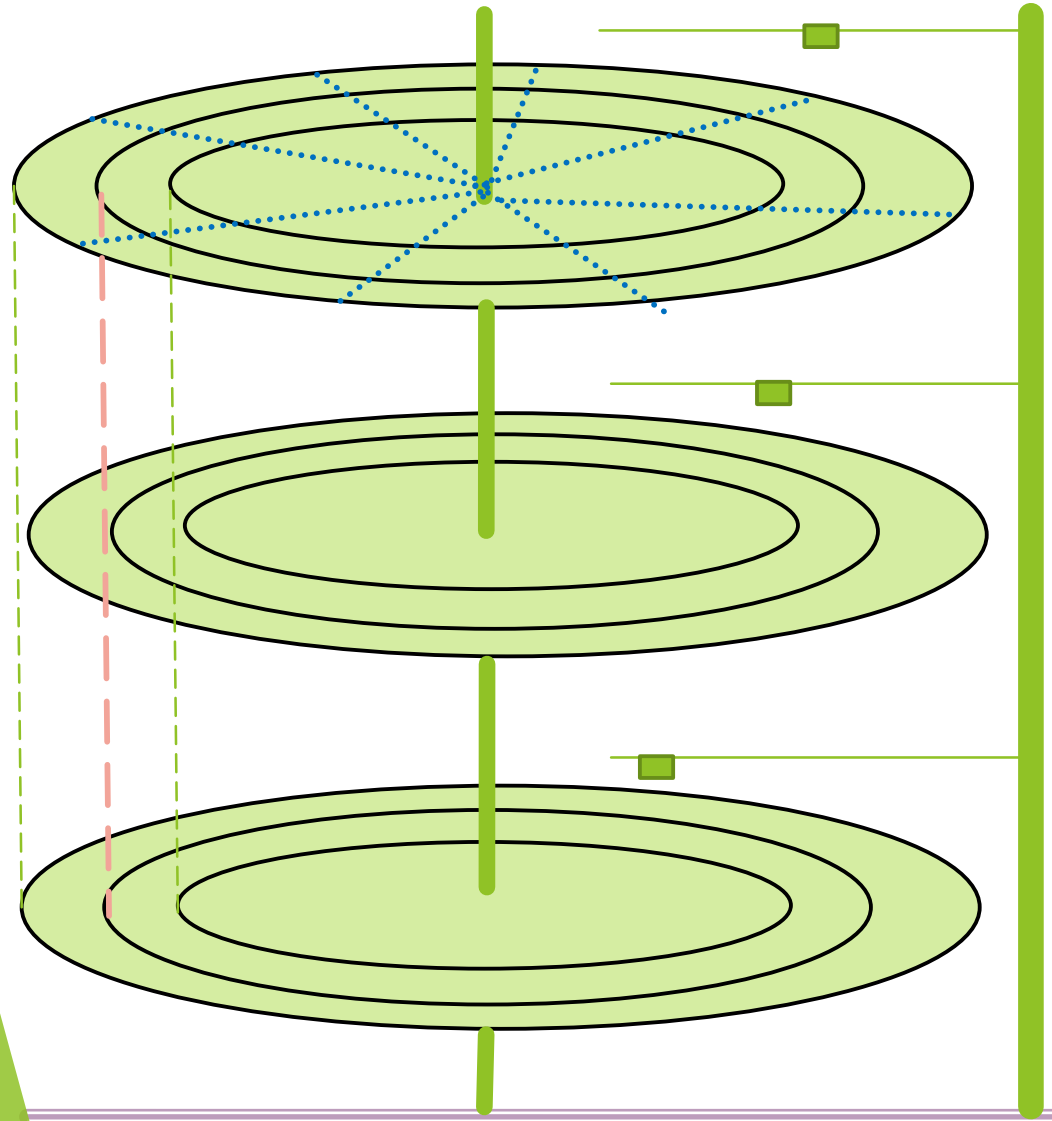
- Searching is used to check for presence of a file
- Filename is used for searching
- Linear search from beginning to end
- Large dictionaries: hash based searching for speed up. Hash of filename is used to identify the index where file entry is stored.

Structure of Hard Disk



- Consists of a stack of disks (coated with magnetic surface), connected to a spindle.
- Each disk has concentric tracks (where data is actually recorded)
- Head moves along an arm to read data from a specific track.
- One head/disk surface
- Corresponding tracks on all surface constitute a cylinder (Red dotted line)
- Each track divided radially into equal number of sectors (blue dotted line)
- Data density (bits per unit length) increases as we move inward from outermost track
- Tracks near center not used as reliable read/write not warranted (owing to physical size of head)

Structure of Hard Disk (contd.)



- Disk rotating at constant angular speed; time to read from/write to a sector is same
- Read/Write
  - Move head along arm to position on track
  - When requisite sector comes under head, start transferring data
  - Identification of a sector is through a format string written at the beginning of a sector
  - Marking sectors through distinguishing sequence of bits is formatting
- Read/write time
  - Seek Time : time to move head to a track
  - Rotational Delay: time to bring a sector under track
  - Data transfer time: time to transfer data to/from sector

Disk Access Time

- Disk access time = seek time + latency time (avg. rotational delay) + data transfer time
- Avg. rotational delay = time to rotate half a track
- Disk rotating at 6000 rpm (revolutions per minute); Number of sectors/track = 8;
- In 60 second, disk is rotated 6000 times
- 6000 tracks transferred in 60 seconds
- Time to transfer one track = 1/100 seconds = 10ms
- Avg. Rotational Delay = 10/2 ms = 5ms
- Data transfer time = 1.25ms per sector
- If sectors being read are non-contiguous, seek time adds to delay for each access.

Disk Addressing

- Track 0 is outermost track
- Consider a large file with contiguous allocation. Two possible ways of storing file on hard disk
  - Surface wise: Requires multiple seek times (moving from track to track) while reading data. Disk address <surface number, track number, sector number>
  - Cylinder wise: Preferred as it requires less number of seeks. <cylinder number, surface number, sector number>
- Most modern drives provide access to logical address space (without any information on how disk address is related to disk geometry). OS may not be able to implement cylinder-wise addressing.

Disk Volume

- Volume is divided into partitions
  - First sector of volume is Master Boot Record (MBR)
  - MBR also contains partition table – information on disk partitions – type of partition, disk addresses for start and end of partition, whether partition is active or not
- Partition
  - The first sector of each partition is a boot record.
  - This is followed by information on file system. Format of this information is specific to file system.

File Allocation Table

- A simple file system designed by Microsoft
- File is organized as clusters; each cluster is a collection of fixed number of sectors.

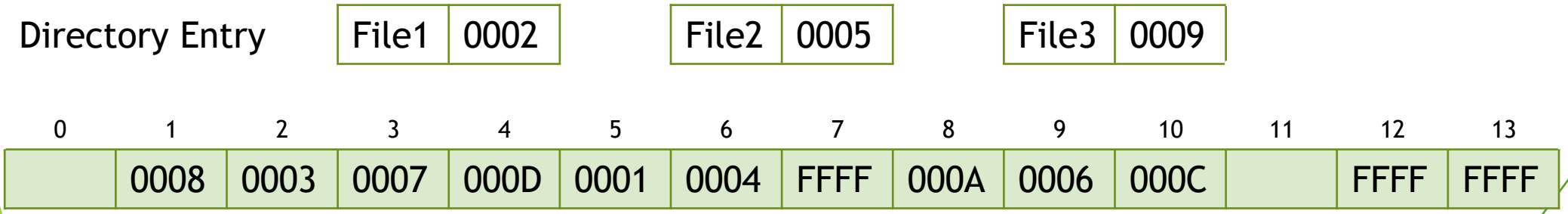
Boot sector	FAT	Copy of FAT	Root directory/folder	Other folders and all files

cluster.

- Maximum number of clusters in FAT16 = 2^{16} and size of FAT shall be 2^{16} x 16 bits = 2^{17} bytes = 128 KB
- Second copy of FAT is kept as a backup. If FAT is corrupted, restoration is possible through this copy.
- Comparing FAT and its copy also enabled OS to detect FAT based virus.

File Allocation Table (contd.)

- A directory entry for a file shall contain number of the first cluster of the file.
- Respective entry for this cluster shall contain number of next cluster.
- Sequence of clusters allocated (cluster number in figure are shown in hexadecimal)
  - File 1: 2 ⟶ 3 ⟶ 7
  - File 2: 5 ⟶ 1 ⟶ 8 ⟶ 10 ⟶ 12
  - File 3: 9 ⟶ 6 ⟶ 4 ⟶ 13



Windows NTFS

- Uses Master File Table (MFT) records to implement file system. MFT is a linear sequence of 1KB records.
- MFT can be placed anywhere (avoiding bad blocks). Boot block contains information on first record of MFT.
- First 16 MFT records reserved by OS to store metadata
- MFT records are numbered 0-15.
- 16^{th} MFT record is about first user file.
- One MFT record for each file or directory
  - Each record keeps information on file name, timestamp etc. and list of disk addresses.
  - For long files, multiple MFT records needed. First MFT (base) points to additional MFT records.

Windows NTFS (contd.)

- Reserved entries
  - First record: information on blocks allocated to MFT file
  - Mirror copy of MFT record
  - Log file for recovery – used for recording addition/removal of a directory; changes to file attributes before these actions are carried out
  - Information on Volume – size
  - Attribute definitions
  - Root directory
  - Bitmap for free space
  - Bootstrap loader file
  - List of bad blocks
  - Security descriptorconversion table (for language
  - Case specific support)
- The locations of the data segments for both the MFT and MFT mirror file are recorded in the boot sector.

Windows NTFS (contd.)

- An MFT record consists of a record header followed by entries stored as <attribute, value> pair. Values can be variable sized.
- Record header contains a magic number used for validity checking, a sequence number updated each time the record is reused for a new file, a count of references to the file, the actual number of bytes in the record used, the identifier of the base record (used only for extension records) etc.
- Record header followed by <attribute, value> pairs.
- Attributes can be
  - resident – stored within MFT record
  - non-resident – stored elsewhere.
- Each attribute stored as a header and value.
- Attribute header length: higher for non-resident attribute is larger as address of disk block where attribute is actually stored is needed.

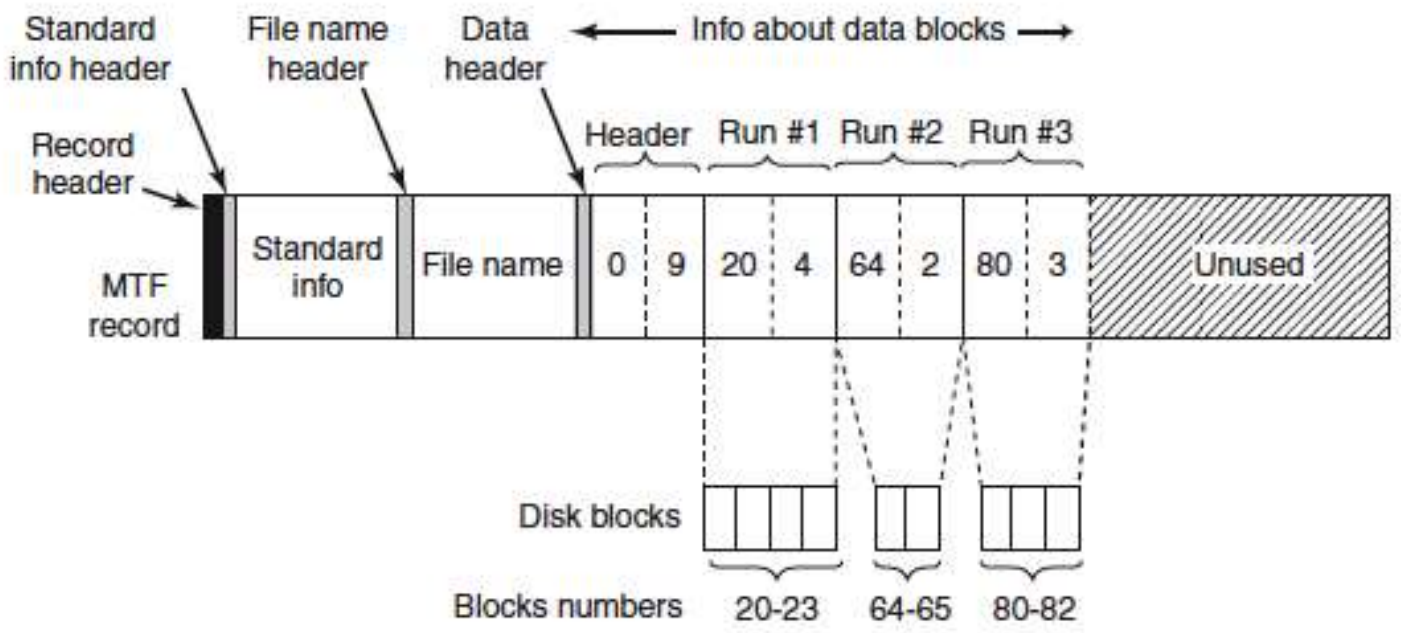
Windows NTFS (contd)

Attribute	Description
Standard information	Owner, permission lag bits, timestamps, etc.
File name	File name in Unicode
Security descriptor	
Attribute list	Location of additional MFT records, if needed
Object ID 64-bit	file identifier (unique to this volume but may be made persistent)
Reparse point	Used for mounting and symbolic links
Volume name	Name of this volume
Volume information	Volume version
Index root	Used for directories
Index allocation	Used for very large directories
Bitmap	Used for very large directories
Logged utility stream	Controls logging to $LogFile
Data	Streamdata; may be repeated

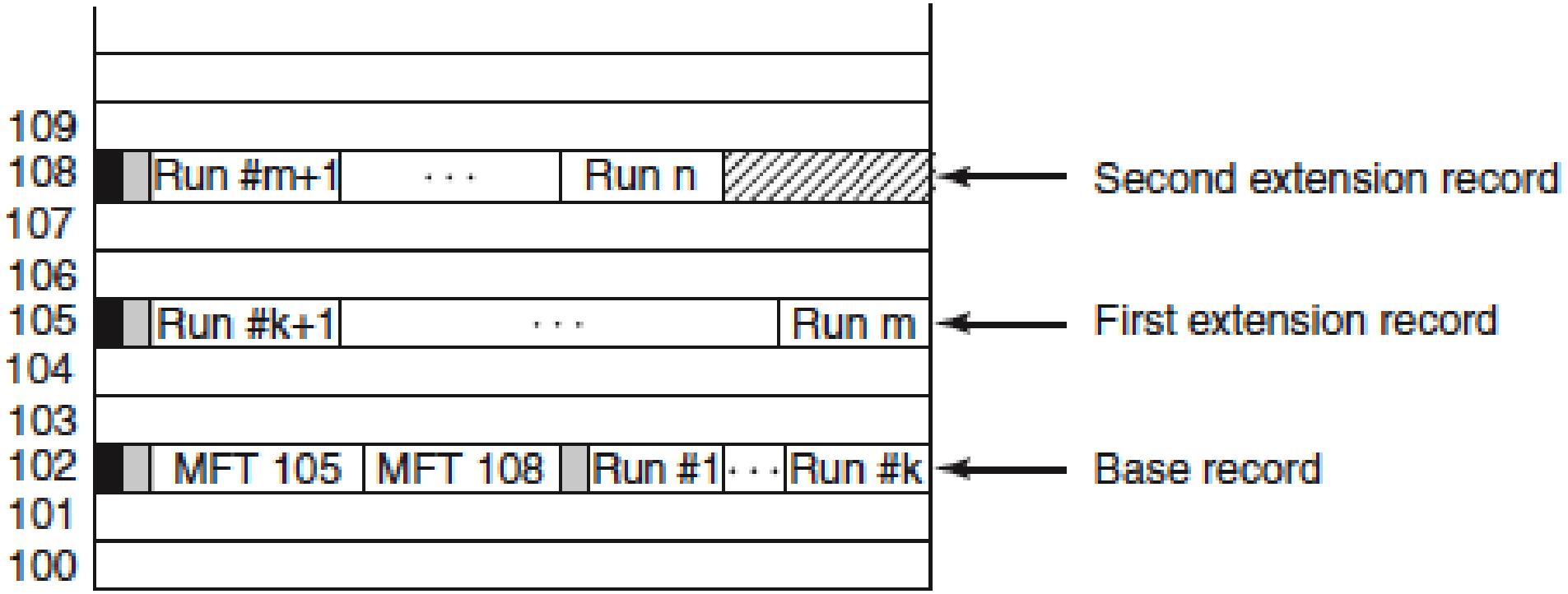
File Allocation

- A file may contain multiple data streams (such as preview as one stream, thumbnail as another etc.)
- Contiguous allocation preferred. Otherwise, multiple runs of contiguous allocations. Information on each run stored in MFT record.

A file with nine data blocks stored as three runs



File Allocation



A large file requiring three MFT records to keep information about all runs

Early Unix File System

Superblock (S)	Inodes	Data Blocks

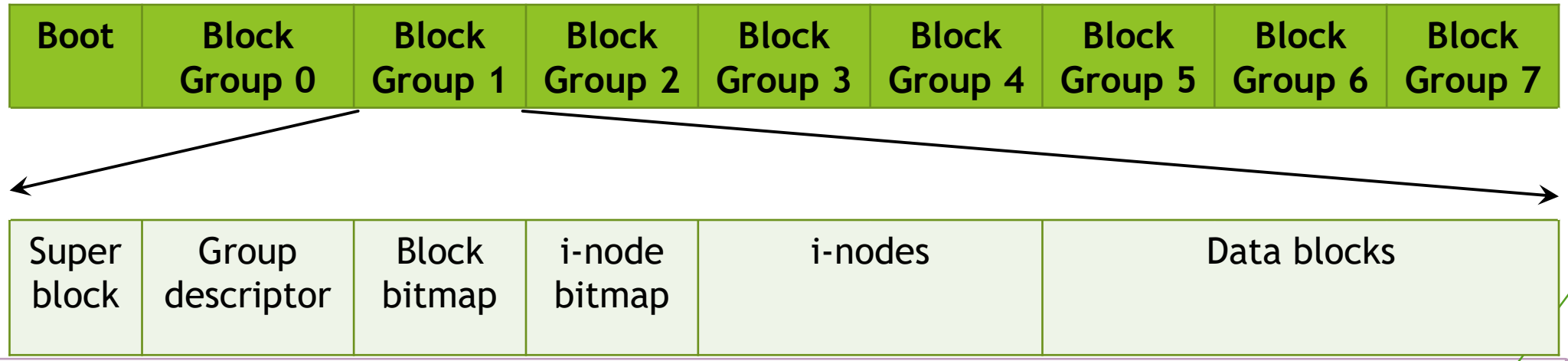
- File System consists of a superblock
  - Volume size
  - Number of inodes
  - Head of list of free disk blocks
- Each inode, assigned to a file, keeps information on file attributes and pointers to data blocks of the file
- To write any file content, disk block is assigned from data blocks.
- Large portion of the disk was reserved for data blocks.

Early Unix File System (contd.)

- Simple to implement
- Worked well for smaller disks
- Limitations: Poor Performance
  - Poor performance as hard disk is not a random access memory;
  - the data blocks of a file were often very far away from its inode (increase in seek time)
  - data blocks assigned to a file began to be non-contiguous over time (due to disk fragmentation); non-contiguous blocks allocated
- Solution: Use block groups.

EXT2 – Linux file system

- Each partition consists of a boot sector followed by group blocks.
- Group block 0 is reserved
- Group block - superblock, group descriptor, block bitmap, inode bitmap, i-nodes, data blocks.



EXT2 – Linux file system (contd.)

- Superblock: no. of i-nodes, no. of disk blocks, the start of the list of free disk blocks
- Group descriptor: location of the bitmaps, no. of free blocks and i-nodes, and no. of directories
- Block bitmap: which data blocks are free
- i-node bitmap: which i-nodes are free
- Both bitmaps are allocated one block (limits number of i-nodes and hence files)
- i-nodes contain information on file and pointers to disk blocks allocated to a file
- Data- blocks are where contents of a file reside

EXT2 – Linux file system (contd.)

- For better performance and reduced fragmentation, EXT2 employs following strategies.
- Collocation (provided that there is sufficient space)
  - Keep ordinary files in the same block group as the parent directory,
  - keep data files in the same group block as the original file i-node,
- Bitmaps: for quick decisions on where to allocate new i-node/ block.
- When new file blocks are allocated, ext2 also pre-allocates a number of additional blocks for that file, so as to minimize the file fragmentation due to future write operations.
- I-nodes corresponding to directories are dispersed throughout the disk block groups.

Linux – Directory Entry

Inode number	Length of entry	Type	Length of file name	Filename (variable sized)	Inode number	Length of entry	Type	Length of file name	Filename (variable sized)

File Systems (CD-ROM)

- Unlike HDD (which have concentric tracks), CD-ROMs have a single continuous spiral track
- Spiral track divided into logical blocks; each block having linear sequence of bits.
- Simple file system (employing contiguous allocation):
  - suffices for CD-ROMs as these are write-once media.
  - No need for keeping track of free blocks as files can neither be added or deleted
- CD-R (CD Recordable)
  - possible to add files after the initial burning,
  - new files are simply appended to the end of the CD-R. files are never removed
  - all the free space is in one contiguous chunk at the end of the CD.
  - as a consequence of this ‘‘append-only’’, CD-ROM file system can be adopted for CD-R
- file systems in DVDs and Blu-ray are based on the one for CD-ROMS.
- Capital case letters are used for name fields for cross-platform compatability.

File Systems (CD-ROM) (contd.)

- Linear sequence of bits along spiral also contains preambles, error correction, and other overhead
- First 16 blocks are reserved
  - May be used by manufacturer for a bootstrap program.
- Next block is the primary volume descriptor
  - system identifier (32 bytes), volume identifier (32 bytes), publisher identifier (128 bytes), and data preparer identifier (128 bytes)
  - names of three files - abstract, copyright notice, and bibliographic information
  - Size of Logical block, the number of blocks on the CD-ROM,
  - creation and expiration dates of the CD-ROM.
  - a directory entry for the root directory, telling where to find it on the CD-ROM (i.e., which block it starts at). From this directory, the rest of the file system can be located.
- CD-ROM may contain a supplementary volume descriptor (similar information to the primary)
- The root directory, and every other directory for that matter, consists of a variable number of entries, the last of which contains a bit marking it as the final one.
- A directory entry is variable-sized; the first field is a byte telling how long the entry is.

File Systems (CD-ROM) (contd.)

- Each directory entry consists of 10 to 12 fields
  - some are in ASCII
  - others are binary (encoded in both little-endian as well as big-endian format)
- Directory entry
  - first byte telling how long the entry is.
  - second byte: length of extended attributes, if any
  - starting block of the file (stored as contiguous runs of blocks)
  - Size of file
  - date and time (separate bytes for the year, month, day, hour, minute, second, and time zone since 1900) when CD-ROM was recorded is stored in the next field,
  - flags field contains a few miscellaneous bits, including one to hide the entry in listings (a feature copied from MS-DOS), one to distinguish an entry that is a file from an entry that is a directory, one to enable the use of the extended attributes, and one to mark the last entry in a directory.
  - Next field is about which CD-ROM (of a set) the file is located on.
  - Next field is length of filename;
  - Filename (base name, dot, extension, semicolon, binary version number)
  - A file name may be present in a directory multiple times, as long as each one has a different version number.
  - Optional: The Padding field is used to force every directory entry to be an even number of bytes
  - Optional: System use field – its usage depends on OS

File Systems (CD-ROM) (contd.)

- Directory entry
  - The first entry is for the directory itself.
  - The second one is for its parent.
  - All other entries in alphabetical order
- The files need not be in same order as their entries appear in directory.
- No explicit limit to the number of entries in a directory.
- Limit to the depth of nesting (maximum depth of directory is eight.
- Files can be stored as three levels
  - Level 1: most restrictive; files are contiguous; file names are limited to 8 + 3 characters; directory names are limited to eight characters with no extensions.
  - Level 2 relaxes the length restriction. It allows files and directories to have names of up to 31 characters.
  - Level 3 uses the same name limits as level 2, but file may consist of several runs of contiguous blocks (called sections or extents).
    - **If same chunk of data occur multiple times in a file or across multiple files ay also occur in two or more files; respective runs can be shared resulting in space optimization.**

Solid State Devices (SSD)

- Solid-state disks (SSD)
  - are more reliable as no moving parts
  - and faster because no seek and rotational delay latency
  - consume less power.
  - more expensive per megabyte and have less capacity
  - shorter life spans than hard disks
- Random accesses are just as fast as sequential ones and many of the problems of traditional disks go away.
- Issues
  - each block can be written only a limited number of times, so writes to a given block needs to be avoided and blocks belonging to frequently updated files may need to be moved
  - Secure deletion of file can not be warranted (Home Assignment: Explain why this should be the case)

Virtual File System (VFS)

- Linux supports: EXT, EXT2, EXT3 and EXT4
- So employs VFS (virtual file system) to integrate all multiple file systems
- VFS provides an abstraction
- The key idea is to
  - abstract out that part of the file system that is common
  - code in a separate layer that calls the underlying concrete file systems to actual file systems
- All system calls are passed to VFS through POSIX interface for user processes
- For individual file system, a low level concrete interface is employed
- When the system is booted, the root file system is registered with the VFS.
- Other file systems are registered with VFS when these are mounted for first time.
- Registration of file system with VFS means a list of the addresses of the functions the VFS requires
- So when a system call is made, VFS maps it to the correct function within the respective file system.

File System - Mount/Unmount

- Mount: Connecting a file system. Files contained in a file system can be accessed only when the file system is mounted
- Unmount: disconnects a file system.
- Mount point: Directory where file system is mounted.
- Effect of mounting temporary – till unmount or next system boot. Does not permanently alter the directory structure.
- Useful:
  - multiple file systems in the OS
  - accessing files located in a remote machine

Mount/Unmount

- Mount point: directory in which a file system can be mounted.
- mount (**<FS_name>**, **<mount_point_name>**), where **<FS_name>**and **<mount_point_name>**are path names respectively of the root of the file system to be mounted and the mount point.
- When the mount operation is performed, the root of the mounted file system assumes the name **<mount_point_name>**.
- Any file with the relative path name **mydir** in the directory **<FS_name>** can be accessed by the path name **<mount_point_name>/ mydir**.
- Any file existing **<mount_point_name>** before mount becomes invisible to the user until the file system is unmounted. Or system may allow only an empty directory as mount-point.

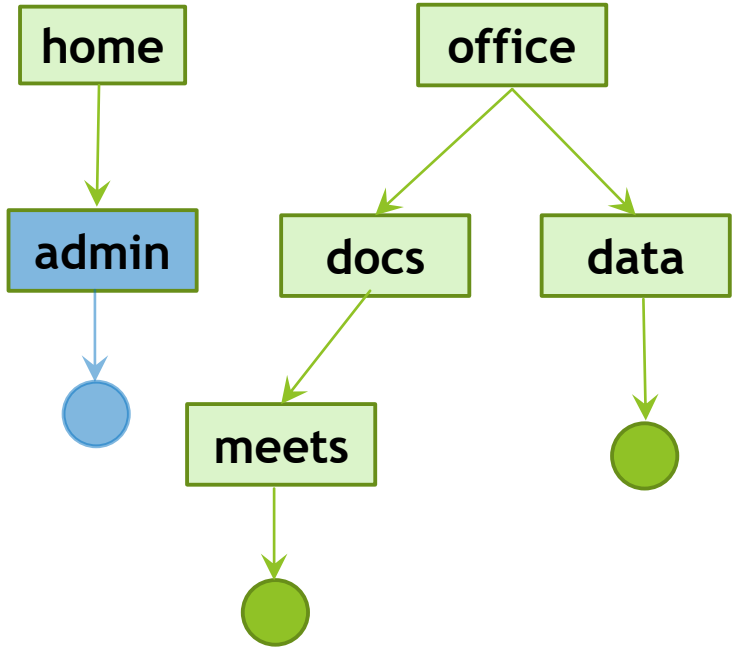
Mounting in a non-empty Directory

- OS may not allow a non-empty directory as a mount-point.
- If OS allows non-empty directory as mount-point
  - obscure the directory's existing files until the file system is unmounted. Only after unmount, access to the original files in that directory is allowed.
- A system may allow the same file system to be mounted repeatedly at different mount points;

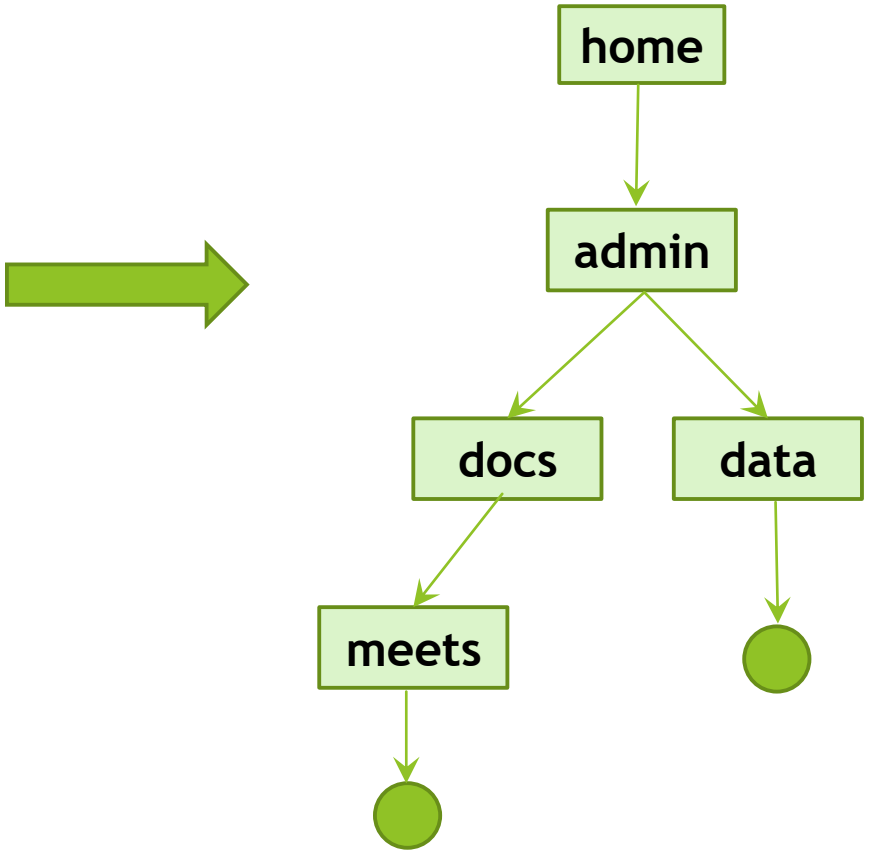
OR

- OS may only allow one mount per file system.

Mount: An example



File system (with root at office) to be mounted at mount-point /home/admin (shown as blue here)



Directory structure After mount operation. Pathname of any file in mounted filesystem shall be prefixed by /home/admin instead of /office

File Sharing

- Consider a file X shared between users B and C.
- If addresses of disk blocks allocated to a file are stored in directories, sharing can pose problems.
  - If B appends to the file, the new blocks will be listed only in B's directory.
  - The changes will not be visible to the other user, defeating the purpose of sharing.
- Solutions
  - Directory entry points to index block (inode in linux) maintaining information about data block addresses. Shared files point to same inode. As a result any changes by one user are reflected to other users too. This is also known as hard link.
  - In the second solution, B links to one of C's files by having the system create a new file, of type LINK, and entering that file in B's directory. The new file contains just the path name of the file to which it is linked. This is akin to soft link in linux.

File Sharing

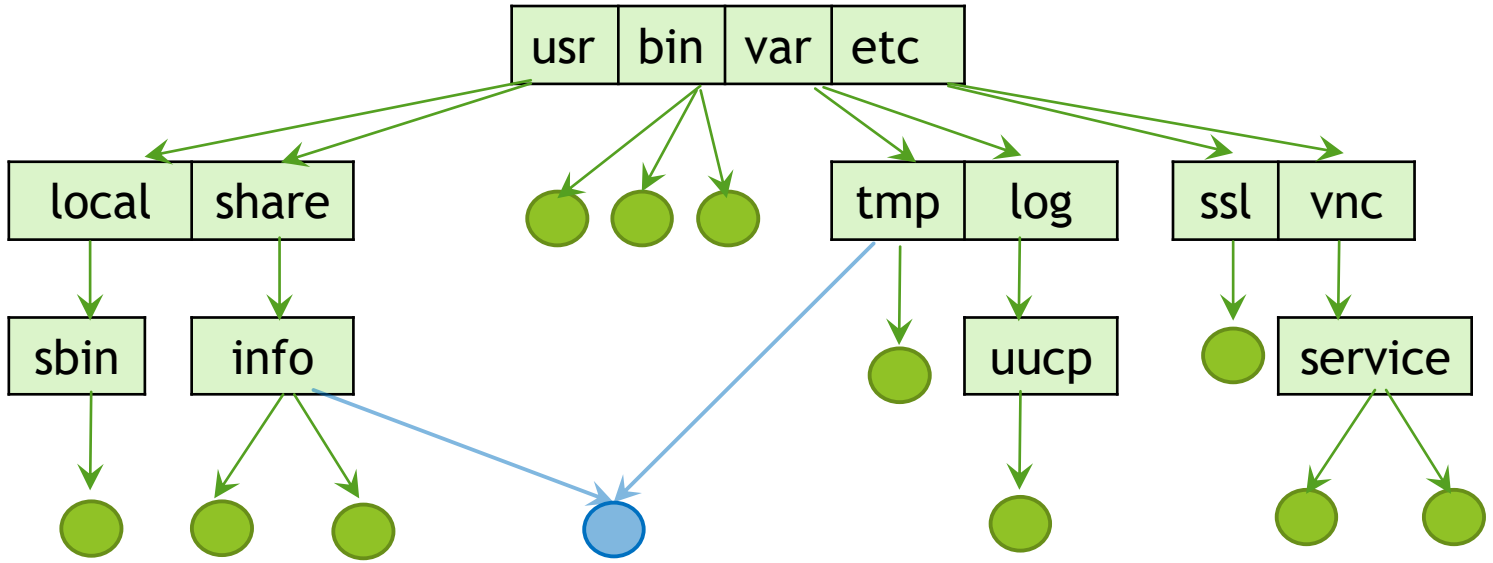
- Hard link:
  - A creates file and is owner. B links to file through hard link. Both A and B share inode. A reference count is incremented each time the file is shared with another user. If the count > 1, the file can not be deleted. So if A deletes the file, A's directory no longer has entry for the file but B's access to file is continued. So now B is pointing to a file owned by A.
- Soft link:
  - As B only keeps pathname, deletion of file by A means B's access is lost. Processing time is increased for B as each read requires path to be parsed.
  - Moving/Renaming a file also results in loss of access.
- Links should ensure that directory structure remain acyclic.

Issues with Linking

- Share file has multiple pathnames
  - traversed multiple times; affect statistics
  - copied multiple times during backup leading to space wastage
- If a shared file is deleted without checking for links
  - dangling pointers to the now-nonexistent file.
  - if disk-blocks released by deleted file are re-used by other files, these dangling pointers may point into the middle of other files
  - Accounting issues

Directory Structure

- Directed Acyclic Graph Blue node is shared file and blue edges are links to the file (In fact, only one is link)



Home Assignment

- Explanatory notes on
  - Network File System
  - ELF (executable linkable format) file format
  - PE (portable executable) file format
- Write a program to compute entropy of
  - Text files
  - Encrypted files
  - Compressed files
  - Word documents

  Files need be opened in binary mode and read byte-by-byte. Read atleast 20 files of each type. Can entropy be used for deducing file extension.

Thank you.