# OS Programming

## *Files and File Systems*

Some content for these slides comes from:
A Guide to Unix Using Linux, Fourth Edition

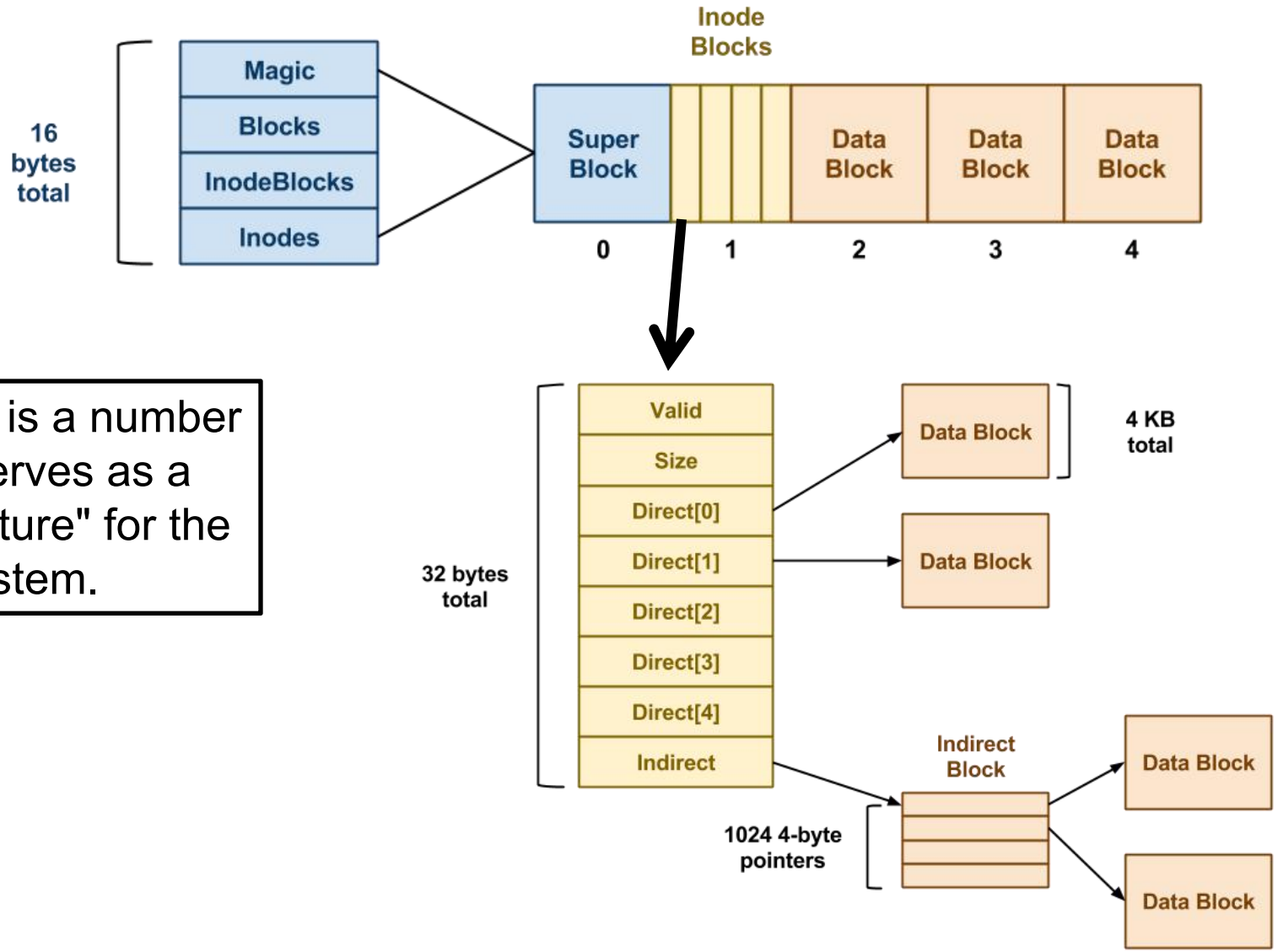
# Understanding UNIX/Linux File Systems

- **File**: basic component for data storage
- **File system**: UNIX/Linux system's way of organizing files on storage devices
  - **Physical file system**: section of the hard disk that has been formatted to hold files
  - Disks are divided into **blocks**. A block is the smallest readable or writable unit which can be addressed. The size of these blocks varies.

# File System Concepts

- **Information nodes**, or **inodes**
  - Each directory/file has an inode and is identified by an inode number
    - Inode 2 contains the root of the directory structure (/)
      - Jumping-off point for all other inodes
      - inode 1 contains "bad blocks"
    - **ls -i** can be used to see the inode of files
  - Contains file/directory name, general information, pointer to the directory/file on a disk partition
- **Superblock** contains information about the layout of blocks on a specific partition

# Basic Idea of a File System




Magic is a number that serves as a "signature" for the file system.

Images from:
https://www3.nd.edu/~pbui/teaching/cse.30341.fa17/project06.html

# Understanding UNIX/Linux File Systems

- UNIX/Linux systems support many file systems
  - Examples: UNIX file system (ufs), extended file system (ext or ext fs)
- **ufs**: original native UNIX file system
  - Expandable, supports large amounts of storage, provides excellent security, reliable
  - Supports **journaling**
    - **The process of keeping chronological records of data or transactions so that if a system crashes without warning, the data or transactions can be reconstructed or backed up to avoid data loss or information that is not properly synchronized.**
  - Supports **hot fixes**
    - **The ability to automatically move data on damaged portions of disks to areas that are not damaged.**

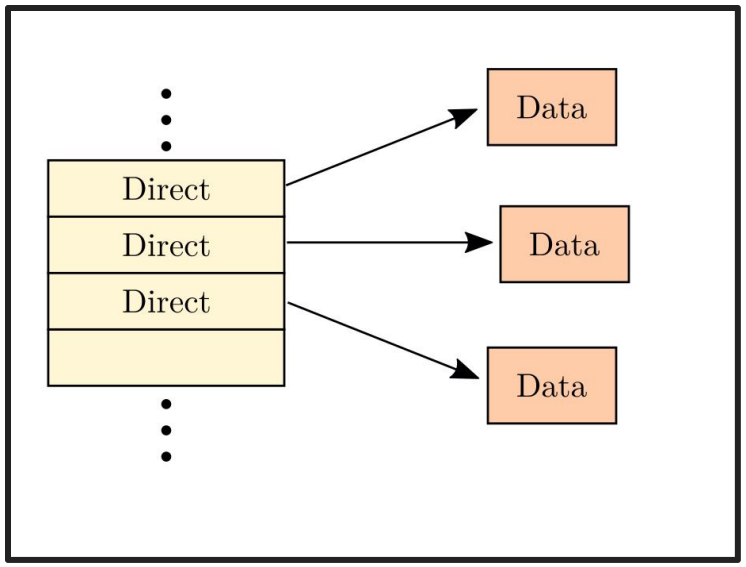
Let's use **df -Th** to look at the filesystems for the online IDE.

# Understanding UNIX/Linux File Systems

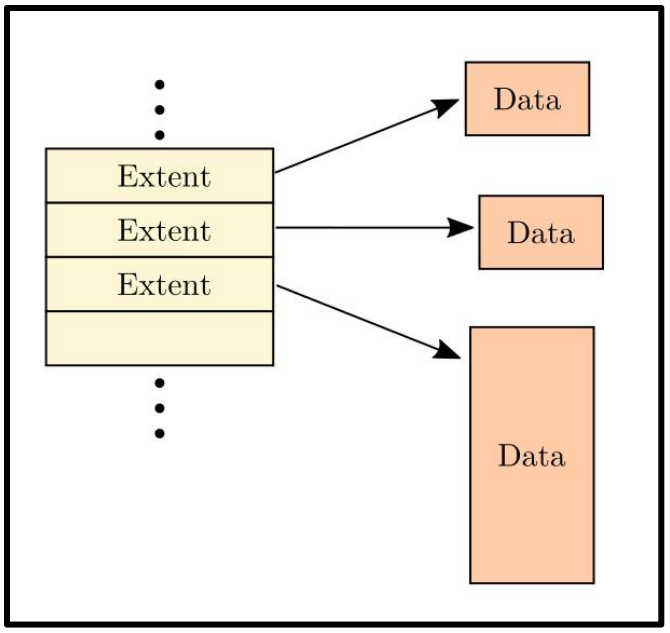
- In Linux, the native file system is **ext**
  - Installed by default
  - Modeled after ufs
    - First version contained some bugs
  - Newer versions of Linux use ext2, ext3, or ext4
    - ext4 enables the use of **extents**
    - An extent is a portion of a disk, such as a block or series of blocks, that is reserved for a file and that represents contiguous space, so that as the file grows, all of it remains in the same location on disk.
      - The use of extents reduces file fragmentation on a disk, which reduces disk wear and the time it takes to retrieve information.

# Understanding UNIX/Linux File Systems

## Extent-based vs. Block-based File Systems




Block-based: Meta-data stores a direct (pointer) to blocks of data.




Extent-based: Meta-data stores an extent (pointer and offset) to ranges of blocks of data.

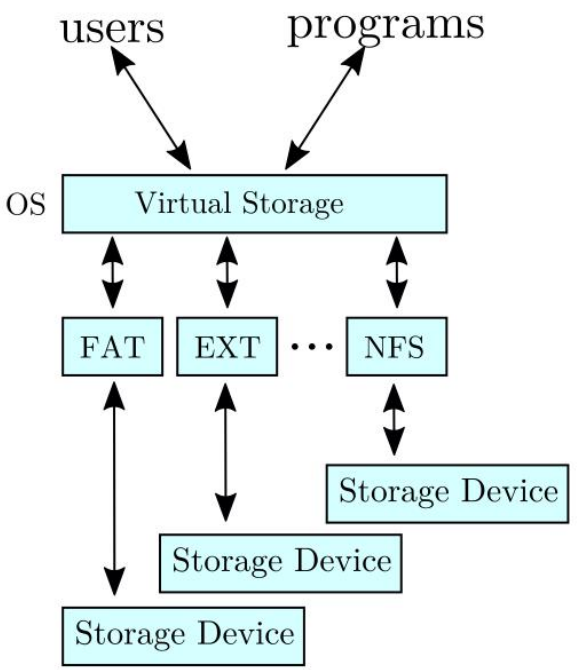
Table 2-2 Comparison of typical file systems supported by UNIX/Linux

| Feature | FAT | NTFS | ext4 | ufs |
|---|---|---|---|---|
| Total volume or partition size | 2 GB to 2 TB | 2 TB | 1 exabyte in Linux depending on the kernel version* | 1 TB in Linux; 4 GB to 2 TB in UNIX depending on the version |
| Maximum file size | 2 GB for FAT16; 4 GB for FAT32 | Potentially 16 TB, but limited by the volume size (up to 2 TB) | 16 GB to 2 TB in Linux depending on the kernel version* | 2 GB in Linux; 2 GB to 16 TB in UNIX depending on the version |
| Security | Limited security based on attributes and shares | Extensive security through permissions, groups, and auditing options | Extensive security through permissions and groups | Extensive security through permissions and groups |
| Reliability through file activity tracking or journaling | None | Journaling | Journaling | Journaling |
| POSIX support | None (FAT16); limited (FAT32) | Yes | Yes | Yes |
| Reliability through hot fix capability | Limited | Supported | Supported | Supported |
| Support for extents | No | Yes, when preallocated via a program | Yes, when enabled | No |
| *These maximums are limited by the kernel version and are based on Linux kernel version 2.6.19. | | | | |

# Understanding UNIX/Linux File Systems

- UNIX/Linux provides a layer of abstraction called **virtual storage**
  - Virtual storage can be allocated using different disks or file systems (or both), but that is transparently accessible as storage to users and programs.

FAT, EXT, and NFS are common filesystems.



# Using UNIX/Linux Partitions

- **Partition**: section of disk that holds a file system
  - UNIX/Linux partitions identified with names
    - Examples: hda1, sda1
      - First two letters tell Linux the device type
      - Third letter indicates if disk is the primary or secondary disk
      - Partitions on a disk are numbered starting with 1

# Setting Up Hard Disk Partitions

- Partition to organize space to contain file systems
- Some UNIX/Linux vendors recommend that:
  - Root partition holds the root file system directory
  - **Swap partition** acts like an extension of memory
    - General rule: same size as RAM
    - A swap partition enables **virtual memory**
  - **/boot partition** to store OS kernel files
- Mount partition to become part of file system

# Using the mount Command

- Use *mount* to connect the file system partitions to the directory tree when the system starts

---

*Syntax* **mount** [-option] [*device-name mount-point*]

---

*Dissection*

- Use the *-t* option to specify a file system to mount.
- *device-name* identifies the device to mount.
- *mount-point* identifies the directory in which you want to mount the file system.

---

- Example:

```
mount -t iso9660 /dev/cdrom /media/cdrom
```

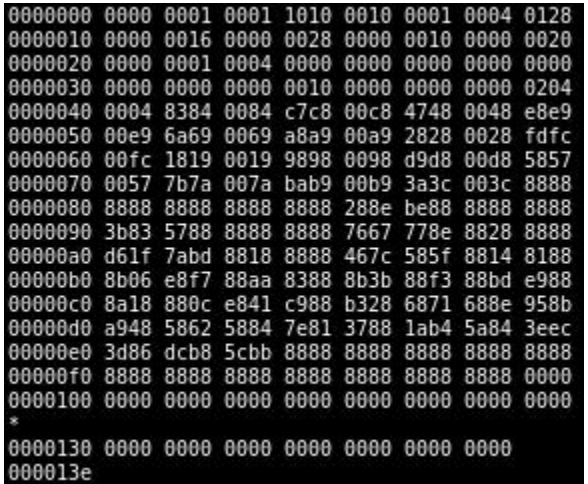
- Use *umount* before removing the storage media

```
umount /media/cdrom
```

Let's end with something hands-on

# The tree, stat, and xxd commands

- **ls -R** and **tree** can be used to view a directory tree.
- The ***stat*** command shows information about a file, including the inode number.
- The ***xxd*** command can make a hexdump of a file or do the reverse.
  - a hexdump is a hexadecimal representation of the binary encoding of a file.
  - example:




Let's try them out!

# Exercises

Create a file called "foo" that contains the word "hello".

Display the contents of the file "foo".

Get the inode number of the file "foo".

View a hexdump of "foo".