CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2024 L21
File Systems

Slides based on

- **Text by Silberschatz, Galvin, Gagne**
- **Various sources**

Disk Structure

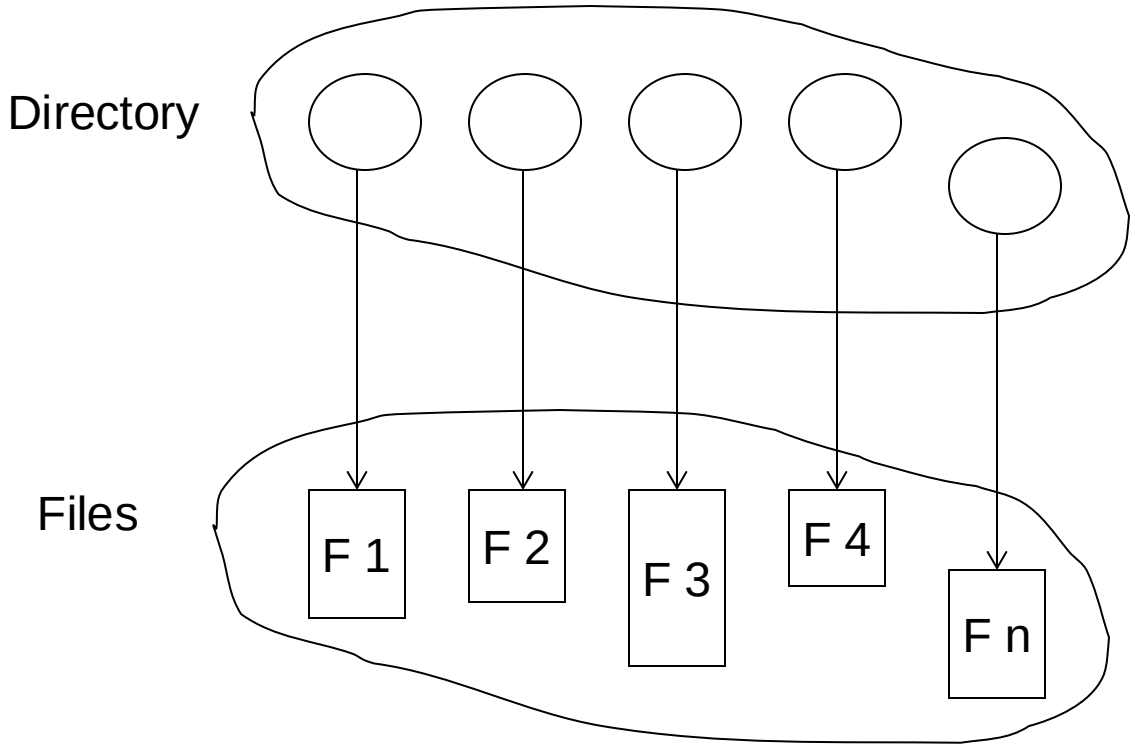
Disk can be subdivided into **partitions**

- Disks or partitions can be **RAID** protected against failure
- Partition can be **formatted** with a file system. Different partitions can host different file systems.
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**

As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

Directory Structure

Directory: A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

Course notes

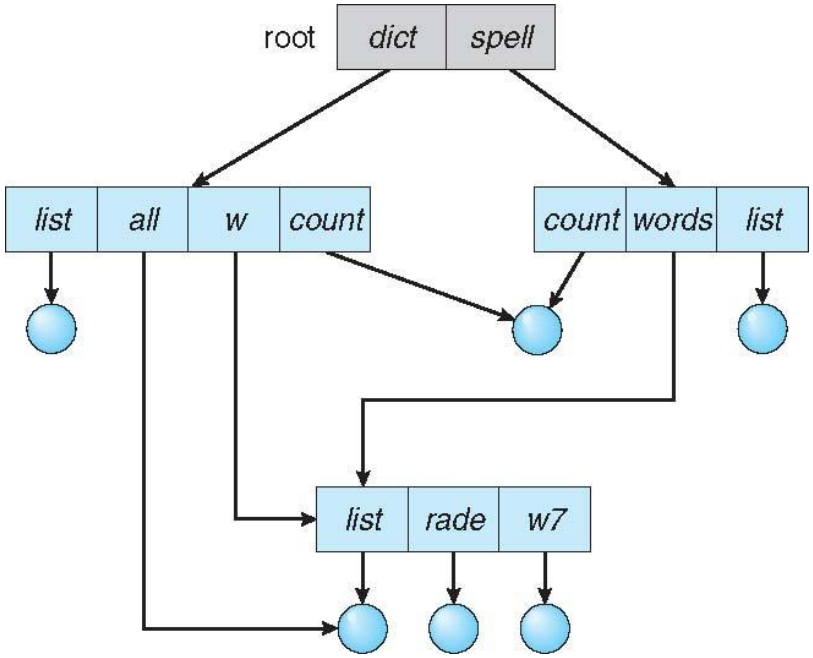
- Help Session for HW5 today (Thursday Oct 31) 5 PM in CSB 130.
- Multithreaded Virtual Network Simulation with producer consumer interaction (Java)
- D2 due today.

Directory Organization

- All files within a directory must have a unique name. But ..

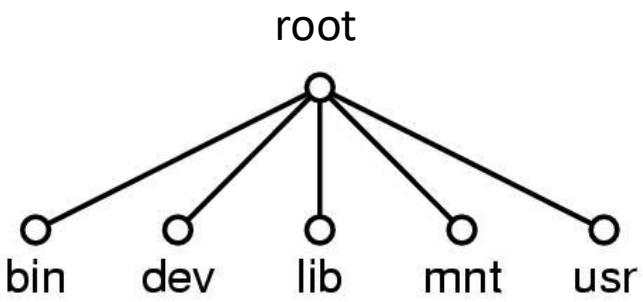
Evolution of directory structure

- Single level directory
- Two-level directory
- Tree-structured directories:
  - efficient grouping, searching,
  - absolute or relative path names
- Acyclic graph directories
  - Shared sub-directory, files

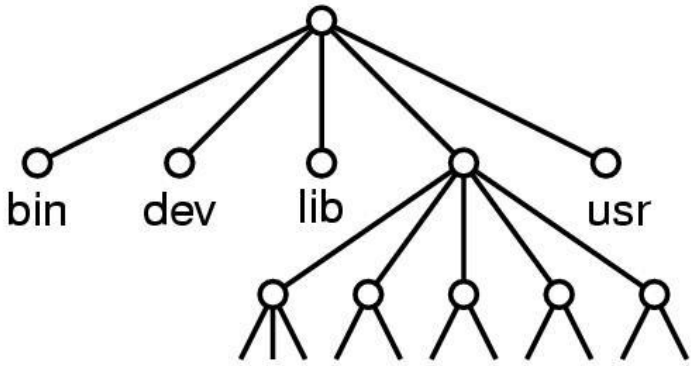


File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**
- **Merges the file system**



(a)



(b)

File Sharing

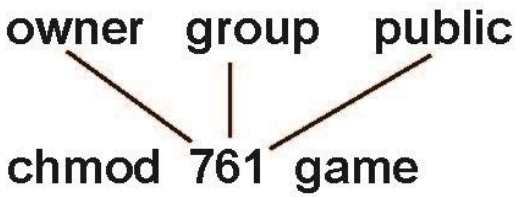
- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

Protection: Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

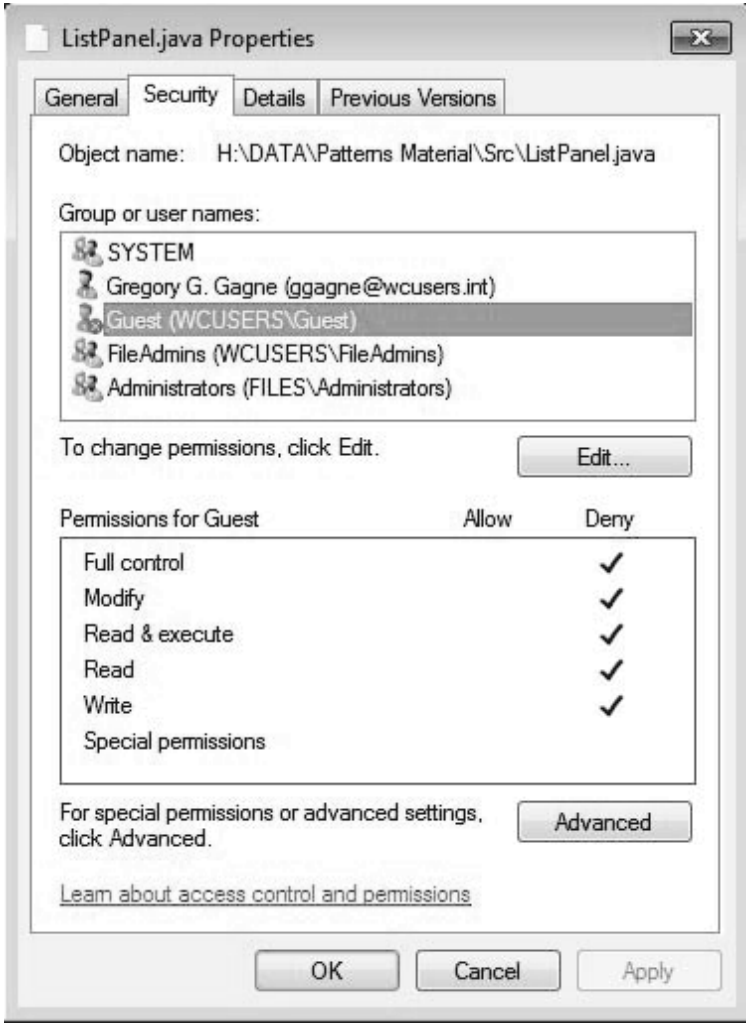
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



- Attach a group to a file

```
chgrp      G      game
```

Windows 7 Access-Control List Management



Colorado State University

A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

dir, access, links, owner, group owner, size, last modification time, name

CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

File-system Implementation

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

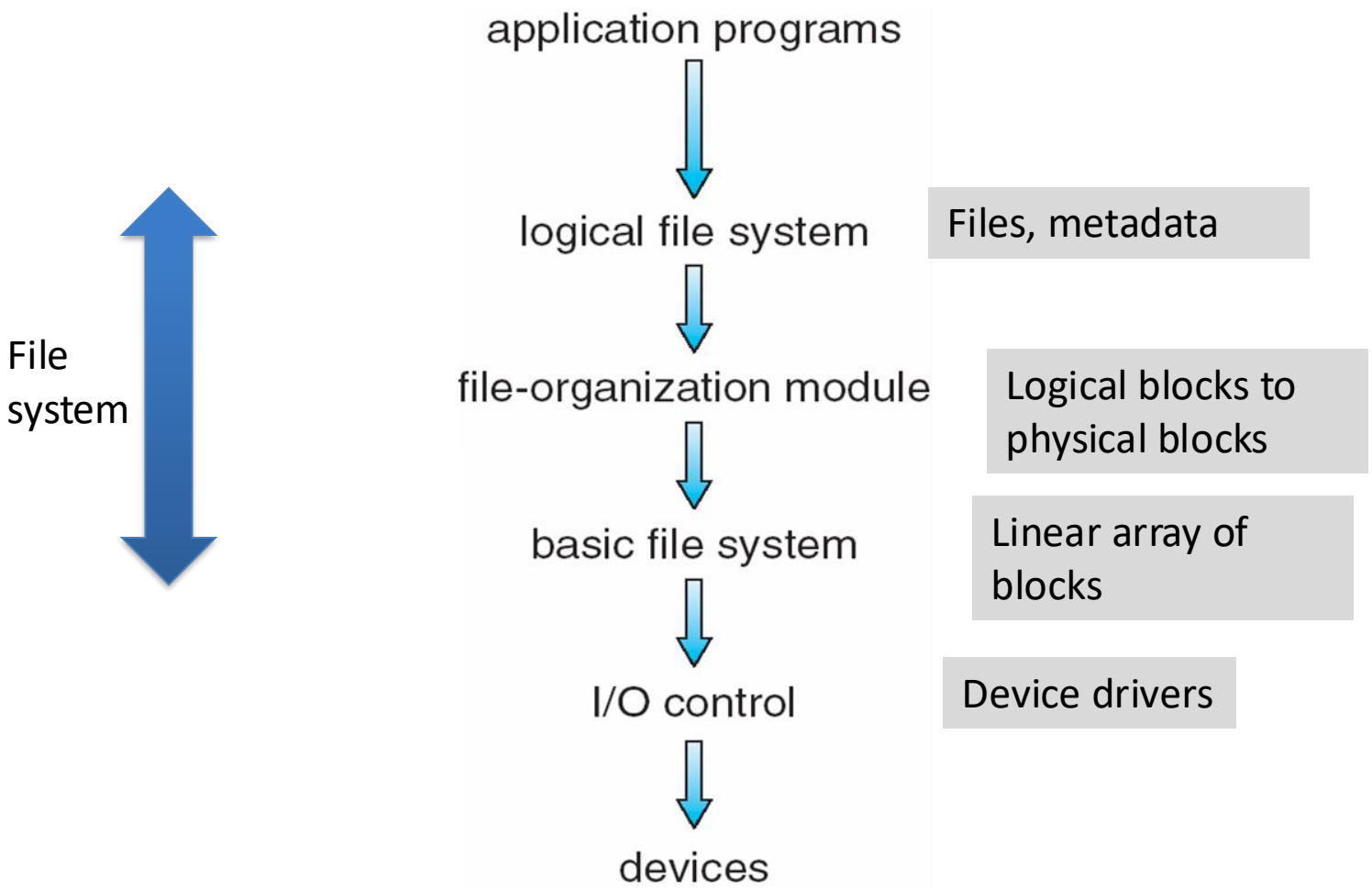
Chap 14/15: File System Implementation/internals

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

File-System Structure

- **File** structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks/SSD)
  - Provides user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
  - Can be on other media (flash etc), with different file system
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure -information about a file ("inode" in Linux) inc location of data
- **Device driver** controls the physical device

Layered File System



File System Layers (from bottom)

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **"Basic file system"** given command like "retrieve block 123" translates to device driver
  - Also manages memory buffers and caches (allocation, freeing, replacement)
    - Buffers hold *data in transit*
    - Caches hold *frequently used data*
- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation
- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining *file control blocks* (**inodes** in UNIX)
  - Directory management
  - Protection

File Systems

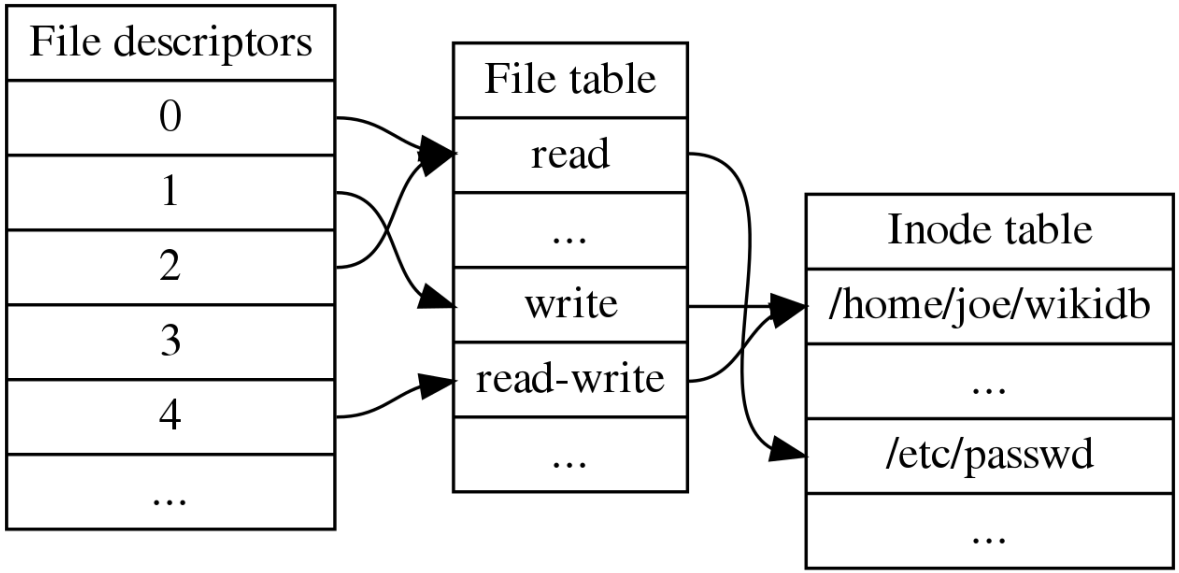
- Many file systems, sometimes several within an operating system
  - Each with its own format
    - Windows has FAT (1977), FAT32 (1996), NTFS (1993), xFAT (USB/SD cards 2006), ReFS (2012)
    - Linux has more than 40 types, with **extended file system** (1992) ext2 (1993), ext3 (2001), ext4 (2008);
    - distributed file systems, GoogleFS (2003), HDFS (2006)
    - floppy, CD, DVD Blu-ray ..
  - New ones still arriving..

Data and Metadata

Storage abstraction:

- File system metadata (size, free lists),
  - File metadata (attributes, disk block maps),
    - Data blocks

Process, System, Files



- **File descriptor table for a process:** File descriptor, pointer
- **System wide open File Table:** r/w status, offset, inode number
- **Inode table for all files/dirs:** indexed by inode numbers **(unix: ls –ia)**
  - **Inode for a file:** file/dir metadata, pointers to blocks

OS File Data Structures

- **Per-process file descriptor table** - for each file,
  - pointer to entry in the open file table
  - current position in file (offset)
  - mode in which the process will access the file (r, w, rw)
  - pointers to file buffer

FD: int

- **Open file table** - shared by all processes with an open file.
  - open count
  - Inode number
- **Inode table** – an inode contains
  - file attributes, including ownership, protection information, access times, ...
  - pointers to location(s) of file in memory

Common File Systems

Journaling: keeps track of changes not yet committed: allows recovery

File System	Max File Size	Max Partition Size	Journaling	Notes
Fat32	4 GiB	8 TiB	No	Commonly supported
ExFAT	128 PiB	128 PiB	No	Optimized for flash
NTFS	2 TiB	256 TiB	Yes	For Windows Compatibility
ext2	2 TiB	32 TiB	No	Legacy
ext3	2 TiB	32 TiB	Yes	Standard linux filesystem for many years.
ext4	16 TiB	1 EiB	Yes	Modern iteration of ext3.

File-System Implementation: Outline

- In memory/On disk structures
- Partitions, mounting
- Disk Block allocation approaches

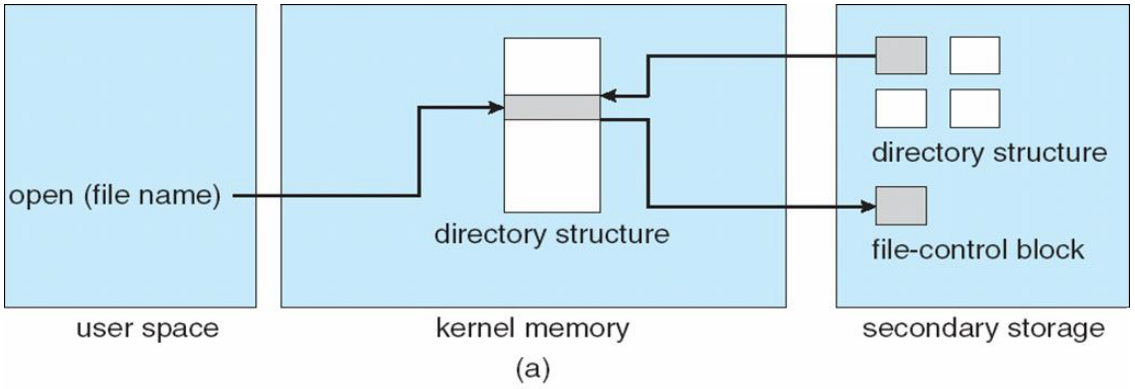
File-System Implementation

Based on several on-disk and in-memory structures.

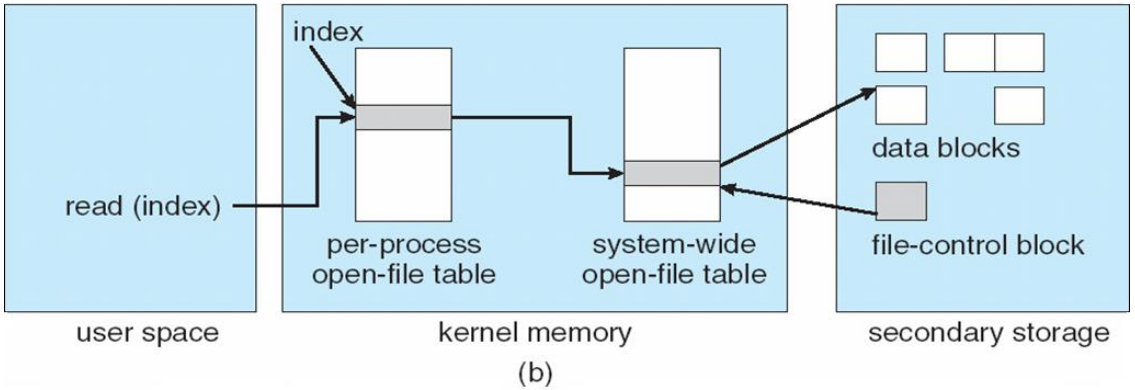
- On-disk
  - Boot control block (per volume) boot block in unix
  - Volume control block (per volume) master file table in UNIX
  - Directory structure (per file system) file names and pointers to corresponding FCBs
  - File control block (per file) inode in unix
- In-memory
  - Mount table about mounted volumes
  - The open-file tables (system-wide and per process)
  - Directory structure cache
  - Buffers of the file-system blocks

Volume: logical disk drive, perhaps a partition

In-Memory File System Structures



Opening a file
fopen( ) returns fid



Reading a file
Inode refers to an individual file

On-disk File-System Structures

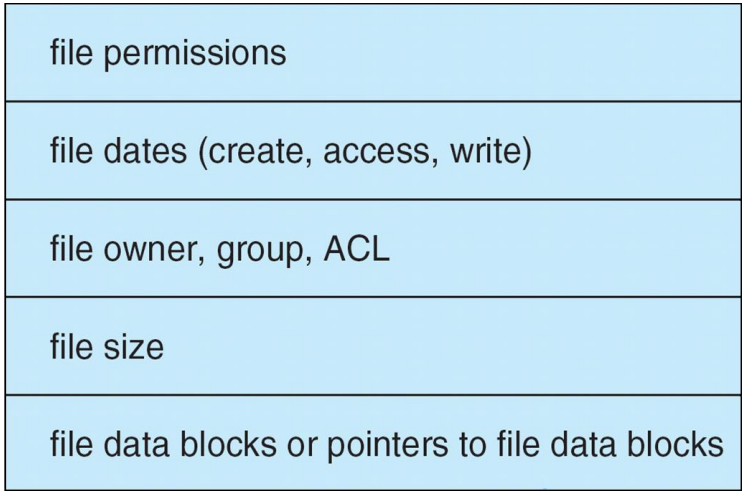
1. **Boot control block** contains info needed by system to boot OS from that volume
   - Needed if volume contains OS, usually first block of volume

Volume: logical disk drive, perhaps a partition

2. **Volume control block (superblock ext or master file table NTFS)** contains volume details
   - Total # of blocks, # of free blocks, block size, free block pointers or array
3. Directory structure organizes the files
   - File Names and inode numbers UFS, master file table NTFS

File-System Implementation (Cont.)

4. Per-file **File Control Block (FCB or "inode")** contains many details about the file
   - Indexed using inode number; permissions, size, dates UFS (unix file system)
   - master file table using relational DB structures NTFS



When a file is created

The OS

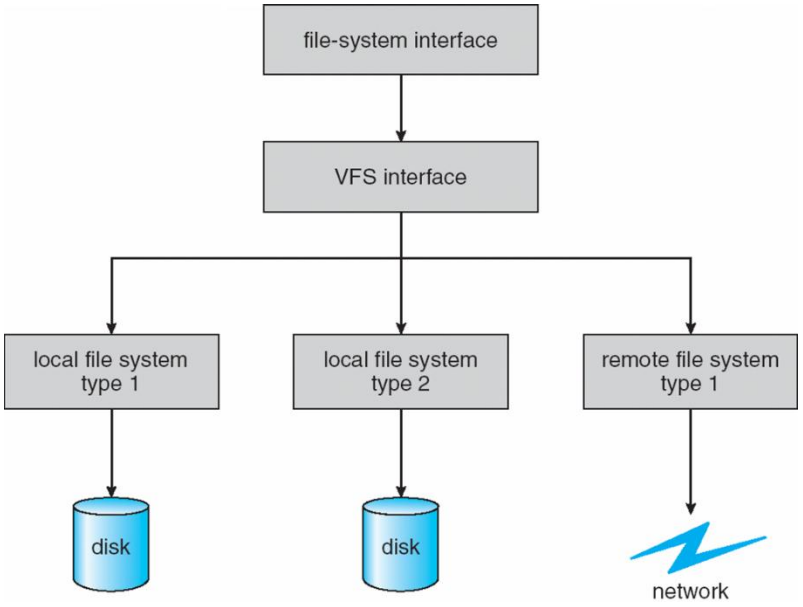
- Allocates a new FCB.
- Update directory
  - Reads the appropriate directory into memory, in unix a directory is a file with special type field
  - updates it with the new file name and FCB,
  - writes it back to the disk.

Partitions and Mounting

- Partition can be a volume containing a file system (*cooked*) or **raw** – just a sequence of blocks with no file system perhaps for swap space
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
- **Root partition** contains the OS, Mounted at boot time
  - other partitions can hold other OSes, other file systems, or be raw
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked

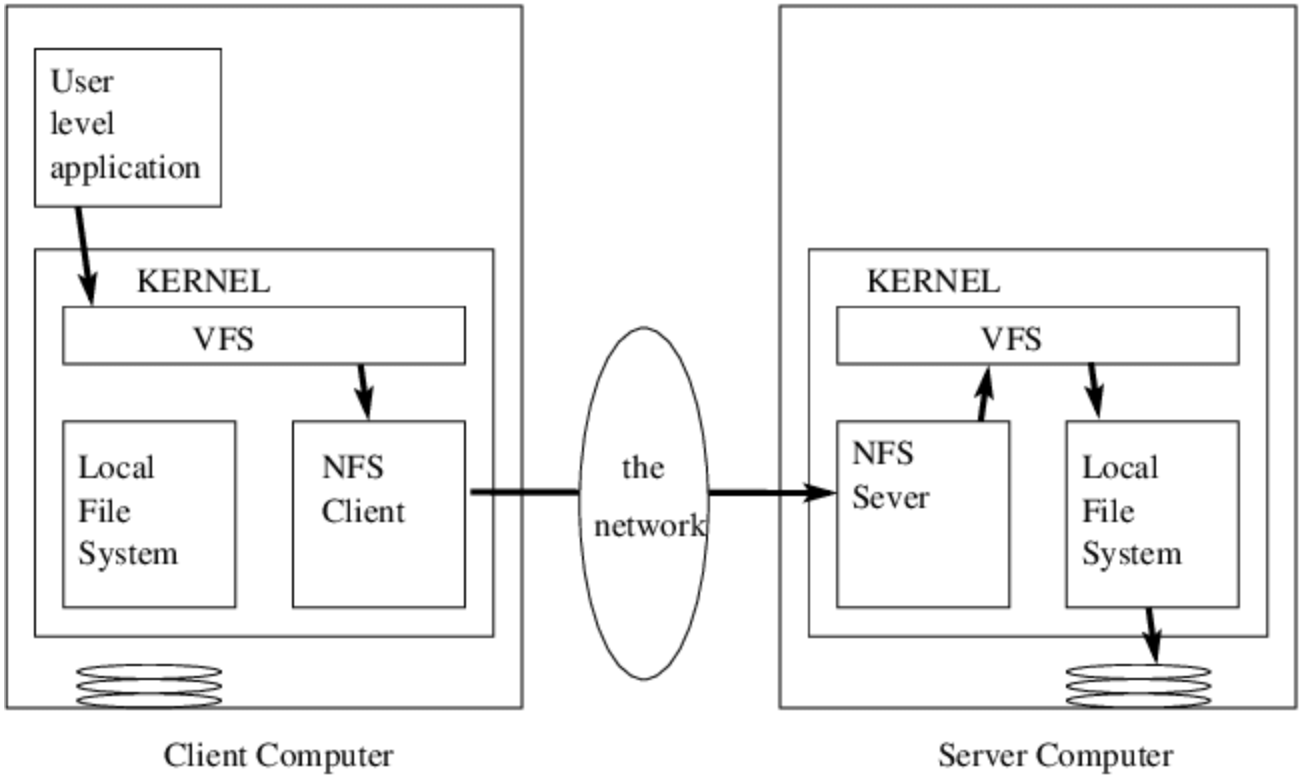
Virtual File Systems

- **Virtual File Systems** (**VFS**) in Unix kernel is an abstraction layer on top of specific file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems
- The API (POSIX system calls) is to the VFS interface, rather than any specific type of file system



Virtual to specific FS interface

NFS (Network File System)



Source

A distributed file system protocol uses the Open Network Computing Remote Procedure Call (ONC RPC) system (1984).

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP/SFTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls

Block Allocation Methods

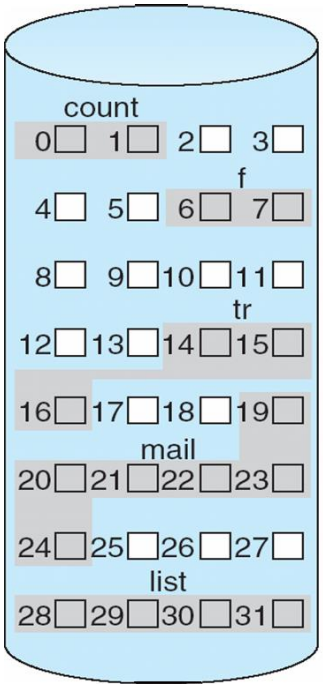
An allocation method refers to how disk blocks are allocated for files:

- Contiguous (not common, except for DVDs etc.)
- Linked blocks, Linked guide (e.g., FAT32)
- Indexed (e.g., ex4)

A disk block can be a physical sector. They ae numbered using a linear sequence.

Actual implementations are more complex than the simple ones examined here.
Contrast these with allocation for processes in memory

Contiguous Allocation

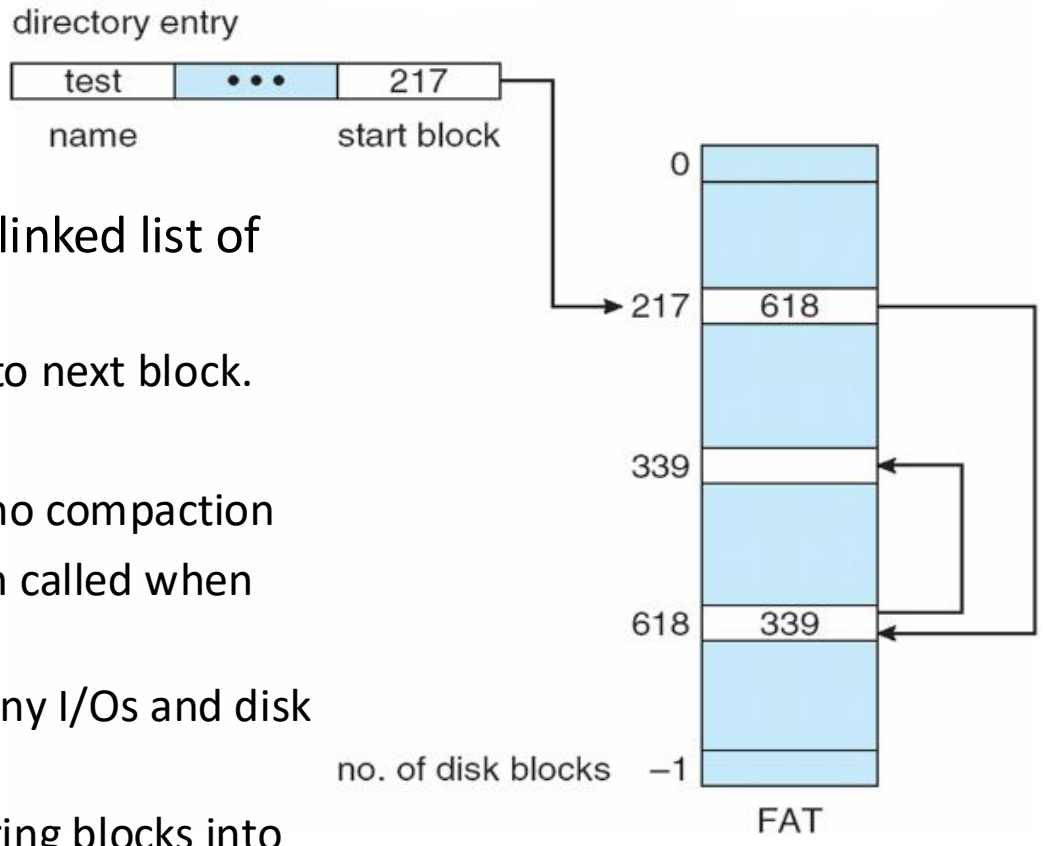


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

File **tr**: 3 blocks
Starting at block 14

Allocation Methods - Linked



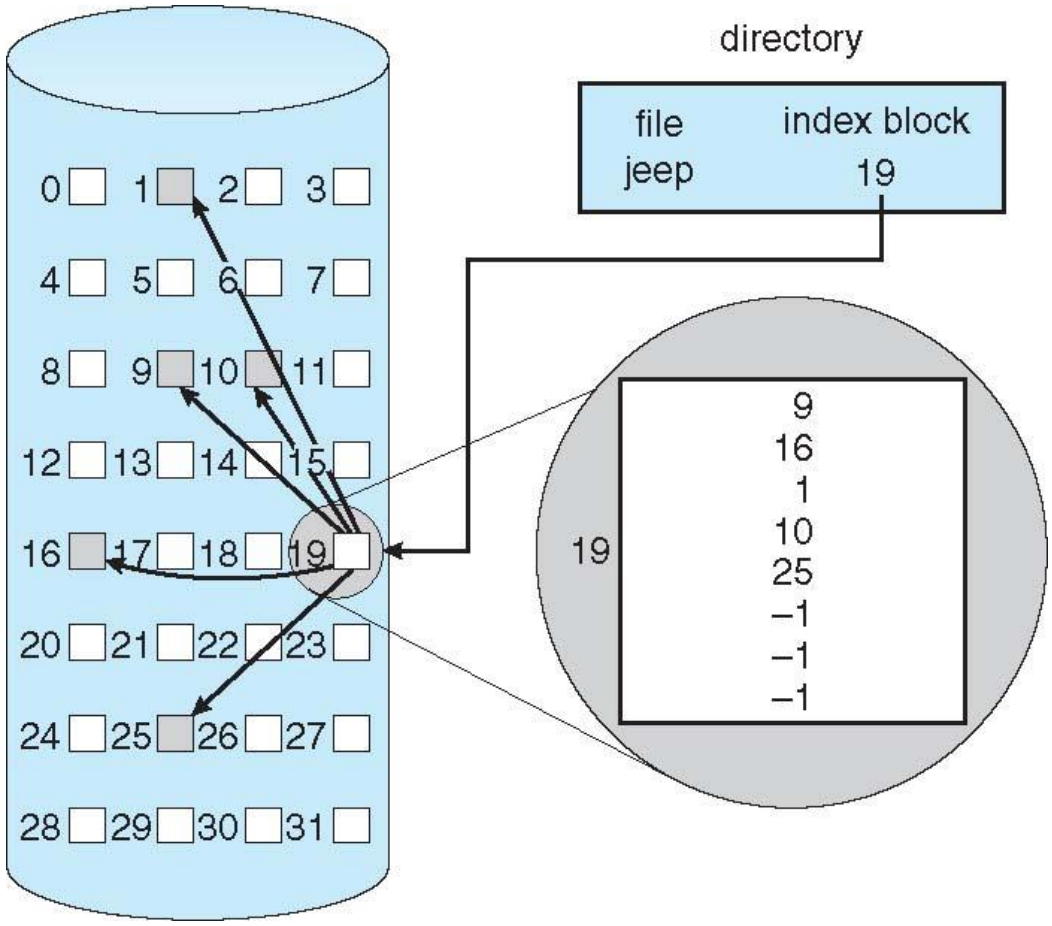
ii. Linked allocation – each file a linked list of blocks

- Each block contains pointer to next block.
- File ends at null pointer
- No external fragmentation, no compaction

Free space management system called when new block needed

- Locating a block can take many I/Os and disk seeks.
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem, since every block in a file is linked.

Example of Indexed Allocation

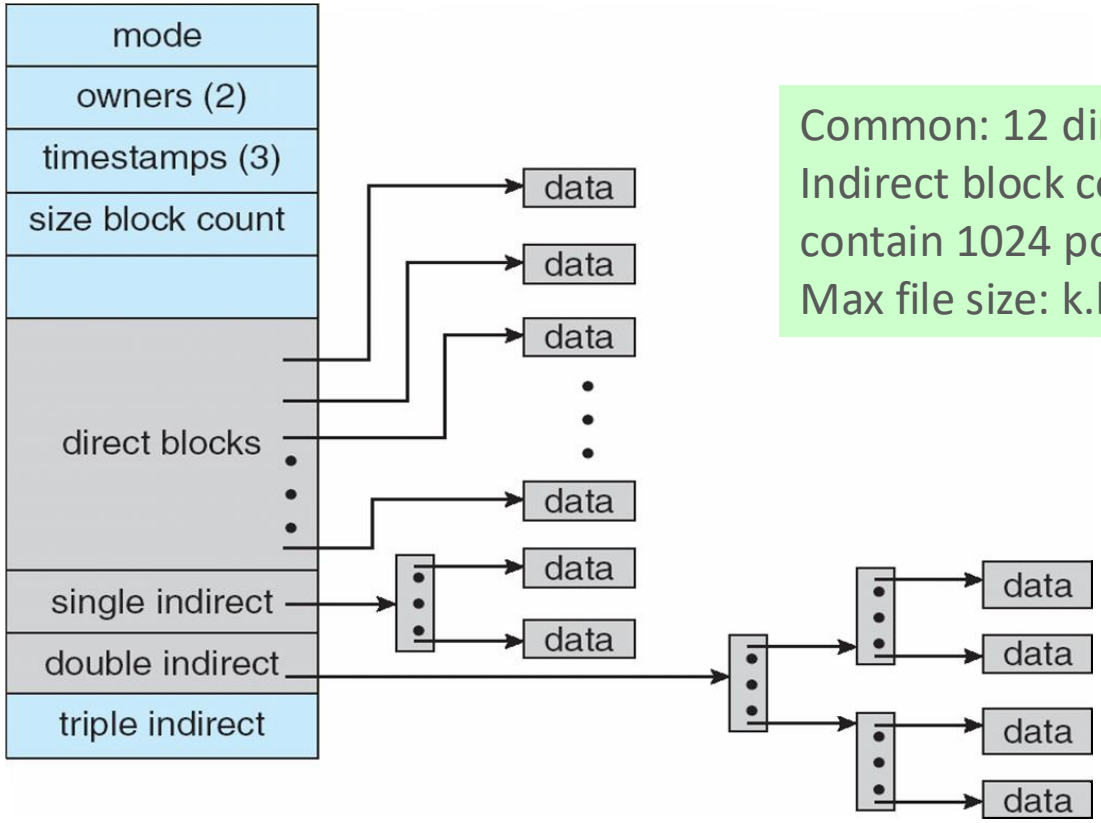


Uses Index blocks.
Index block has pointers to data blocks for a file.

Indexed Scheme: UNIX inodes

Assume 4K bytes per block, 32-bit addresses. Ext3 example.

Volume block:
Table with file names
Points to this inode
(file control block)



Common: 12 direct+3.
Indirect block could
contain 1024 pointers.
Max file size: k.k.k.4k (triple)+

Ext4: uses extents
(pointer+ length)

More index blocks than can be addressed with 32-bit file pointer

Performance

- Best method depends on file access type
  - Contiguous: OK when files don't change much
  - Linked: used for smaller file systems of the past: FAT, FAT32
  - Indexed more complex, modern
    - Single block access could require 0-3 index block reads then data block read
    - Clustering or disk caching can help improve throughput, reduce CPU overhead
    - Ex: Ext3, Ext4

Cluster: set of contiguous sectors

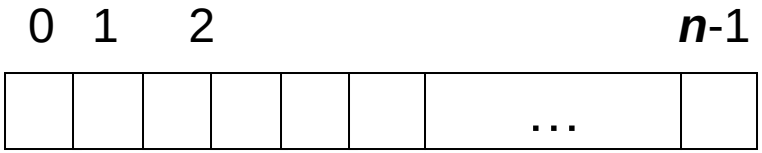
Performance (Cont.)

- Is adding instructions to the execution path to save one disk I/O is reasonable?
  - AMD Ryzen Threadripper 3990X (2020) 2,356,230 MIPS
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - Typical disk drive at 250 I/Os per second
    - 2,356,230 MIPS / 250 = 9425 million instructions during one disk I/O
  - Fast SSD drives provide 60,000 IOPS
    - 2,356,230 MIPS / 60,000 = 39.3 millions instructions during one disk I/O

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
  - **Approaches: i. Bit vector ii. Linked list iii. Grouping iv. Counting**

i. Bit vector or **bit map** (***n*** blocks)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Block number calculation for first free block

(number of bits per word) *(number of 0-value words) + offset of first 1 bit

```
00000000
00000000
00111110
..
```

CPUs may have instructions to return offset within word of first "1" bit

Free-Space Management (Cont.)

Bit map requires extra space

- Example:

  block size = 4KB = 2^{12} bytes

  disk size = 2^{40} bytes (1 terabyte)

  blocks: $n = 2^{40}/2^{12} = 2^{28}$

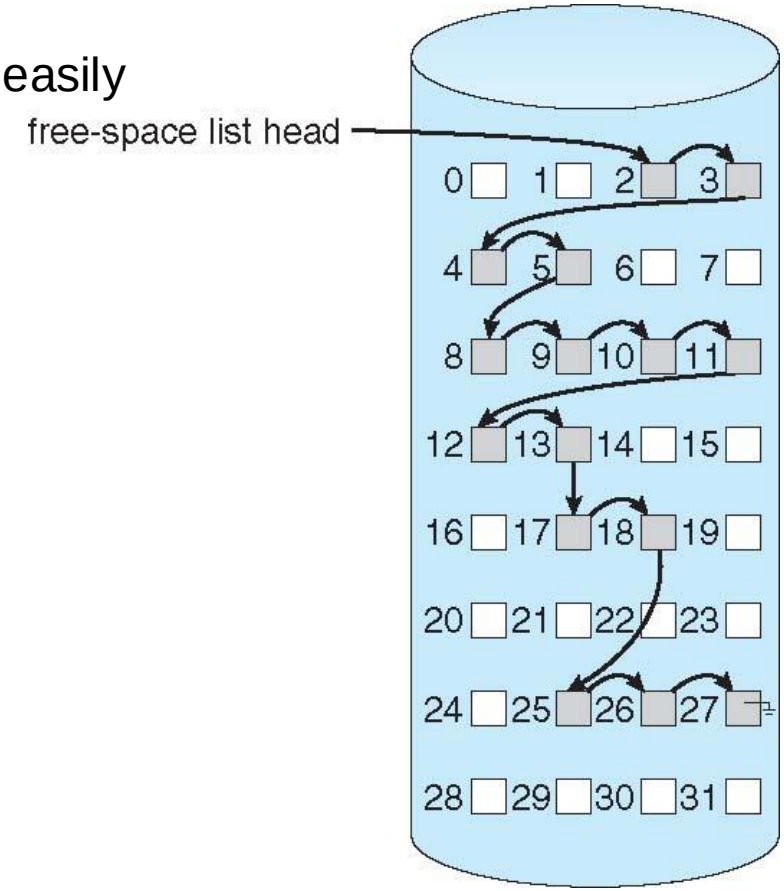
  Need 2^{28} bits or 32MB for map

  if clusters of 4 blocks -> 8MB of memory

Bit map makes it easy to get contiguous files if desired

Linked Free Space List on Disk

- ii. Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space



Superblock Can hold pointer to head of linked list

Free-Space Management (Cont.)

- iii. Grouping
  - Modify **linked list** to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers free block pointer blocks in a linked list.
- iv. Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free contiguous blocks
    - Free space list then has entries containing **addresses and counts**

UNIX directory structure

- Contains only file names and the corresponding inode numbers an inode uniquely identifies a file
- Use **ls – i** to retrieve inode numbers of the files in the directory
- Looking up path names in UNIX

  Example: /usr/tom/mbox

  – Lookup inode for /, then for usr, then for tom, then for mbox

Advantages of directory entries that have name and inode information

- Changing filename only requires changing the directory entry
- Only 1 physical copy of file needs to be on disk
  - File may have several names (or the same name) in different directories
- Directory entries are small
  - Most file info is kept in the inode

Hard and symbolic links

Hard Links:

- Both file names refer to the same inode (and hence same file)
  - Directory entry in /dirA

    ..[12345 filename1]..
  - Directory entry in /dirB

    ..[12345 filename2]..
- To create a hard link

  ln /dirA/filename1 /dirB/filename2
- Symbolic link *shortcut* in windows
  - To create a symbolic link

    ln -s /dirA/filenmame1 /dirB/filename3

    File filename3 just contains a pointer

File system based on inodes

Limitations

- File **must fit** in a single disk partition
- Partition size and number of files are **fixed** when system is set up

inode preallocation and distribution

- inodes are **preallocated** on a volume
  - Even on empty disks % of space lost to inodes
- Preallocating inodes
  - Improves performance
- Keep file's data block **close** to its inode
  - Reduce seek times

Checking up on the inodes

Command: df -i (df is for disk filesystem)

Gives inode statistics for the file systems: total, free and used nodes

```
Filesystem              Inodes   IUsed    IFree IUse% Mounted on
devtmpfs               2045460     484  2044976    1% /dev
tmpfs                  2053722       1  2053721    1% /dev/shm
tmpfs                  2053722     695  2053027    1% /run
tmpfs                  2053722      16  2053706    1% /sys/fs/cgroup
```

Command: ls –i (lists inodes of the files in current directory_

```
13320302              diskusage.txt
2408538               Documents/
680003                downloads/
```