CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 38+

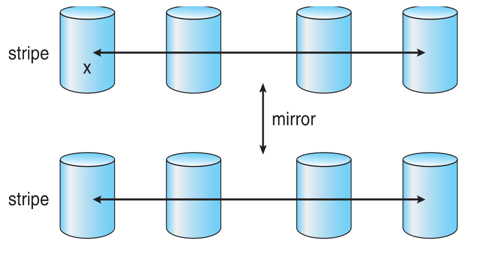
File-system Implementation

Slides based on

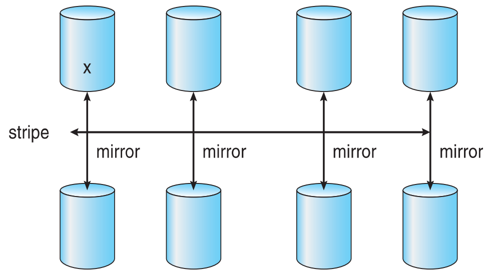
- Text by Silberschatz, Galvin, Gagne
- Various sources

FAQ

- Why RAID 2,3,4 are not used? RAID 2,3 Bit/byte level operations are inefficient for disks. RAID 5 provides more even wear compared to RAID 4.
- RAID 5: Block parity: How does it rebuild bad disk?
  - Bad disk identified by bad CRC.
  - Guess value of corrupted bit to obtain same parity
  - Assume even parity. 1011x0 must have been 1011_0
- Difference between RAID 01 and 10?
  - RAID 10 mirrored at a lower level, will tolerate multiple failures.



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

Colorado State University

RAID Techniques

- **Striping** uses multiple disks in parallel by splitting data: higher performance, no redundancy (ex. RAID 0)
- **Mirroring** keeps duplicate of each disk: higher reliability (ex. RAID 1)
- **Block parity:** **One Disk hold** parity block for other disks. A failed disk can be rebuilt using parity. Wear leveling if interleaved (RAID 5, double parity RAID 6).
- Ideas that did not work: Bit or byte level level striping (RAID 2, 3) Bit level Coding theory (RAID 2), dedicated parity disk (RAID 4).
- Nested Combinations:
  - RAID 01: Mirror RAID 0
  - RAID 10: Multiple RAID 1, striping
  - RAID 50: Multiple RAID 5, striping
  - others

Notes

- Upcoming deadlines:
- Q12 Nov 29
- HW2 Dec 1
- Poster Session Dec 9 (Design, Print earlier Dec 6?)
- Peer reviews Dec 12
- Final project Dec 13
- Final test: Dec 16

Outline

- File Concept, types
- Attributes, Access Methods, operations, Protection
- Directory Structure, namespace, File-System Mounting, File Sharing
- Storage abstraction: File system metadata (size, freelists), File metadata(attributes, disk block maps), data blocks
- Allocation of blocks to files: contiguous, sequential, linked list allocation, indexed
- In memory: Mount table, directory structure cache, open file table, buffers
- Unix: inodes numbers for directories and files

Data and Metadata

Storage abstraction:

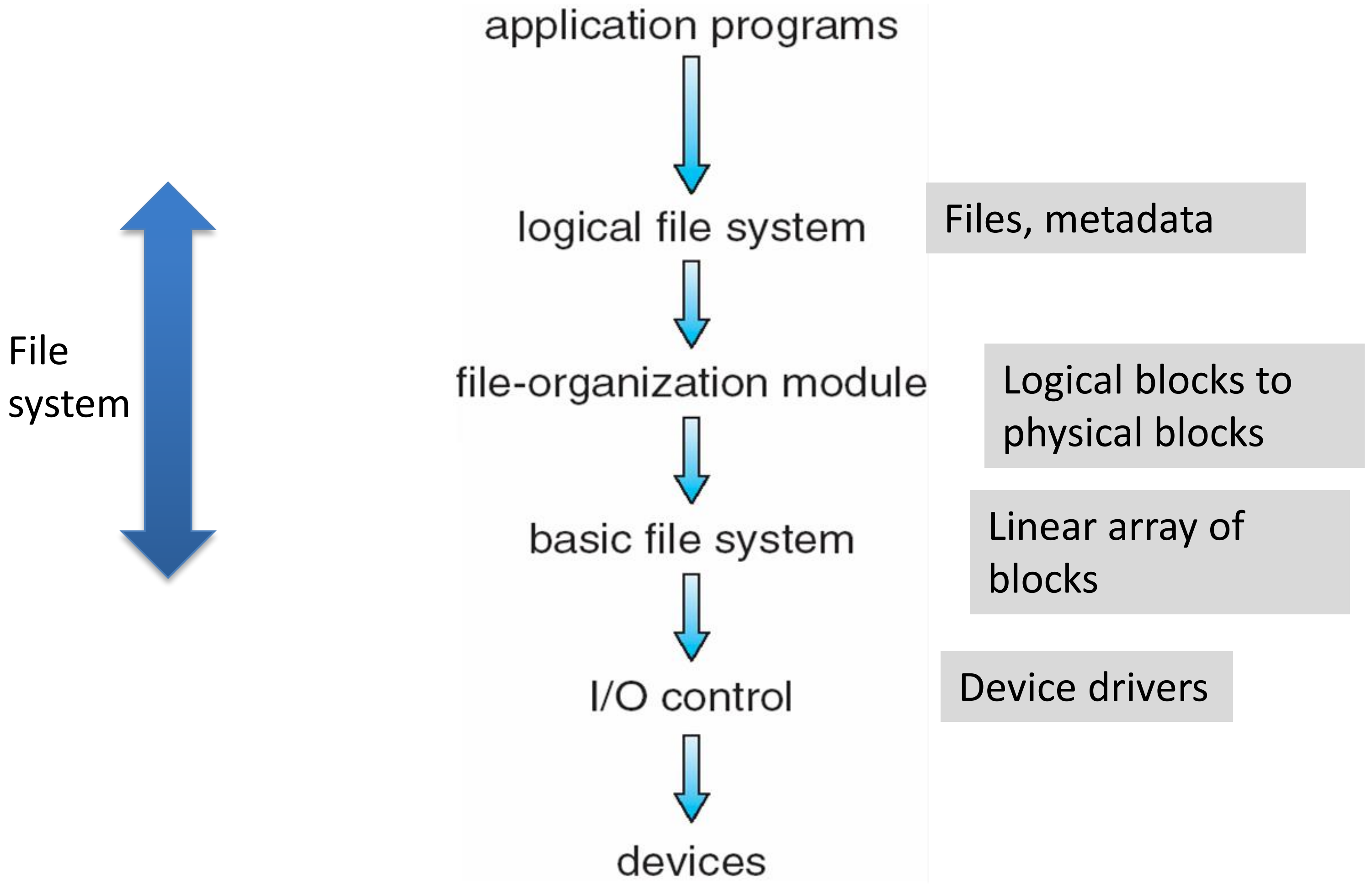
- File system metadata (size, free lists),
- File metadata (attributes, disk block maps),
- Data blocks

OS File Data Structures

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

Layered File System



Layered File System

Processes

fd = open (afilename, ..)
read (fd, buf, size);
write (fd, buf, size);
close (fd)

Logical File System Layer

Search dir, find file location, determine which file blocks will be used

Logical block numbers(s);
file_start block on disk

File Organization Layer

Map file blocks (logical blocks) to disk blocks (physical blocks), disk allocation

Physical block numbers

Basic File System Layer

Commands to device driver, Buffering of disk data, caching of disk blocks

In cache? If not, get block

Device Driver

Cylinder, track, sector, R/W

Device Controller

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** gives command like “retrieve block 123” 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

File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file handle FD, 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)
    - Linux has more than 40 types, with **extended file system** (1992) ext2 (1993), ext3 (2001), ext4 (2008);
    - plus distributed file systems
    - floppy, CD, DVD Blu-ray
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE, xFAT

Common File Systems

Journaling: 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

- 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

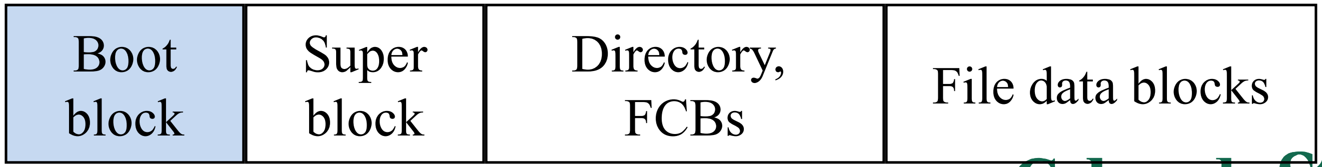
Volume: logical disk drive, perhaps a partition

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
- master file table using relational DB structures NTFS

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Colorado State University

Create a file

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

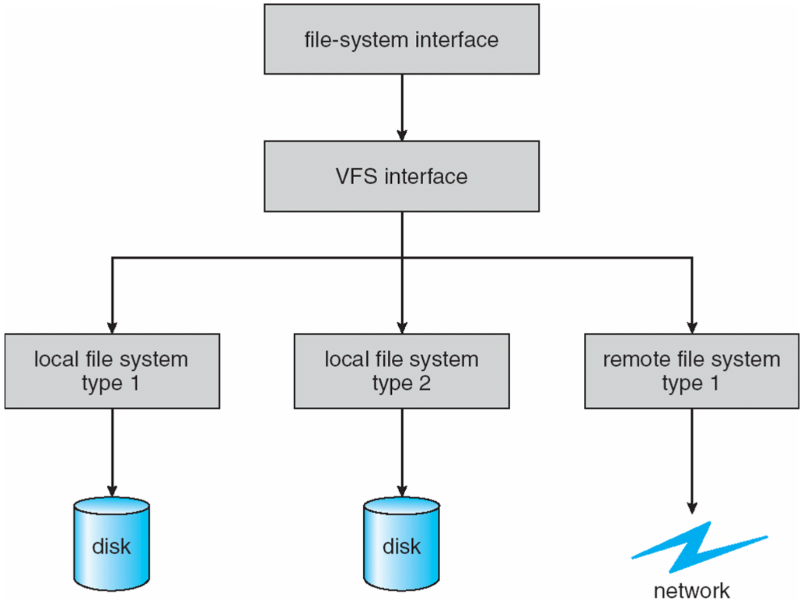
Colorado State University

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**) on Unix provide an object-oriented way of implementing 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

Colorado State University

Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location.
    - use chained-overflow method
    - Each hash entry can be a linked list instead of an individual value.

Colorado State University

Allocation Methods

Allocation Methods – i.Contiguous

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

- Contiguous (not common now)
- Linked (e.g. FAT32)
- Indexed (e.g. ex3)

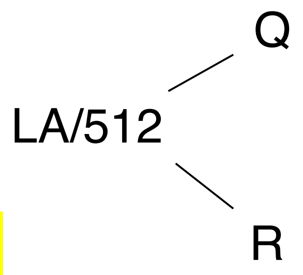
Allocation Methods – i.Contiguous

i. Contiguous allocation – each file occupies set of contiguous blocks

- Simple – only starting location (block #) and length (number of blocks) are required
  - Occupies n block   b, b+1, …b+n-1
- Minimal disk head movement
- Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

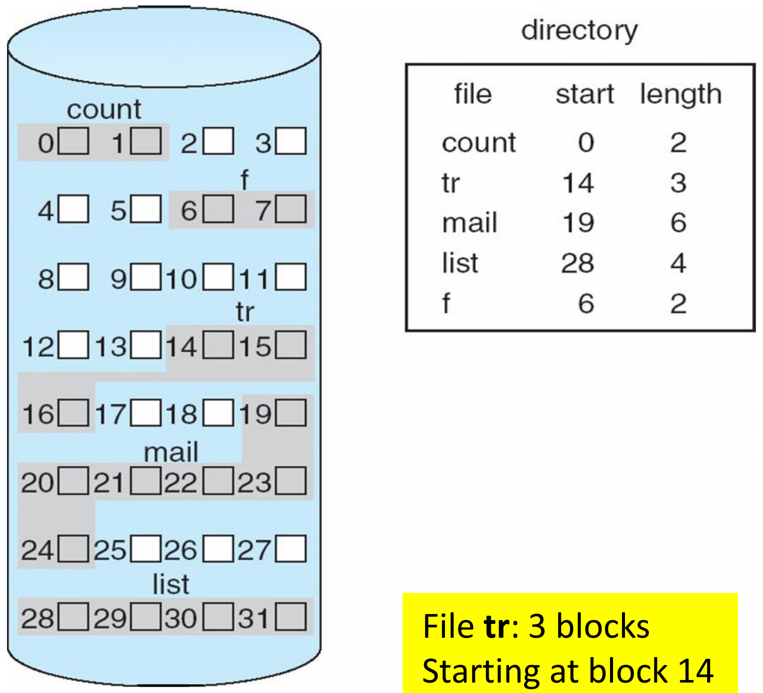
Contiguous Allocation

- Mapping logical byte address LA to physical



Assume block size =512

Block to be accessed = Q + starting block number (address)
Displacement into block = R



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

Colorado State University

Extent-Based Systems

- Some file systems use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
  - An **extent** is a contiguous block
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Allocation Methods - Linked

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

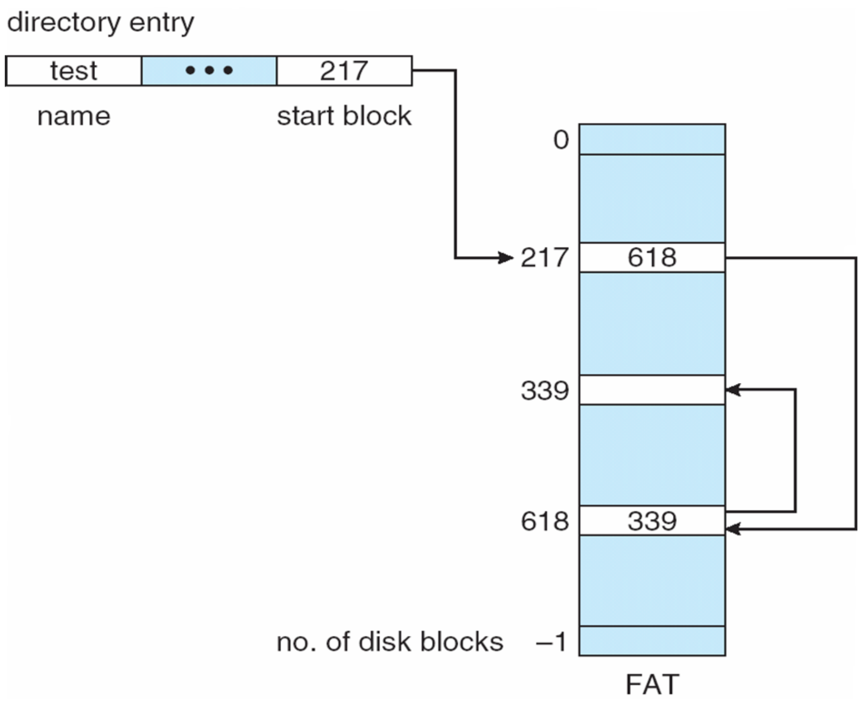
- Each block contains pointer to next block.
- File ends at null pointer
- No external fragmentation, no compaction

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

Allocation Methods – Linked (Cont.)

- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

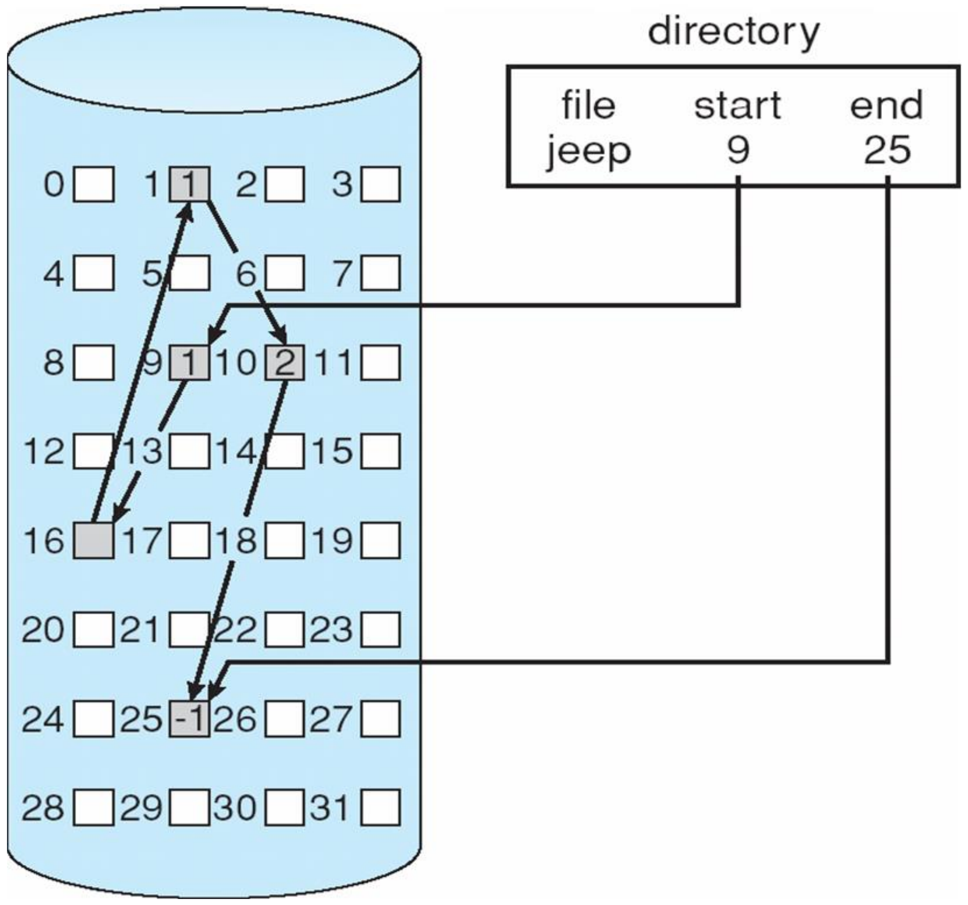


Each FAT entry corresponds to the corresponding block of Storage.

Free block entries are also linked.

Colorado State University

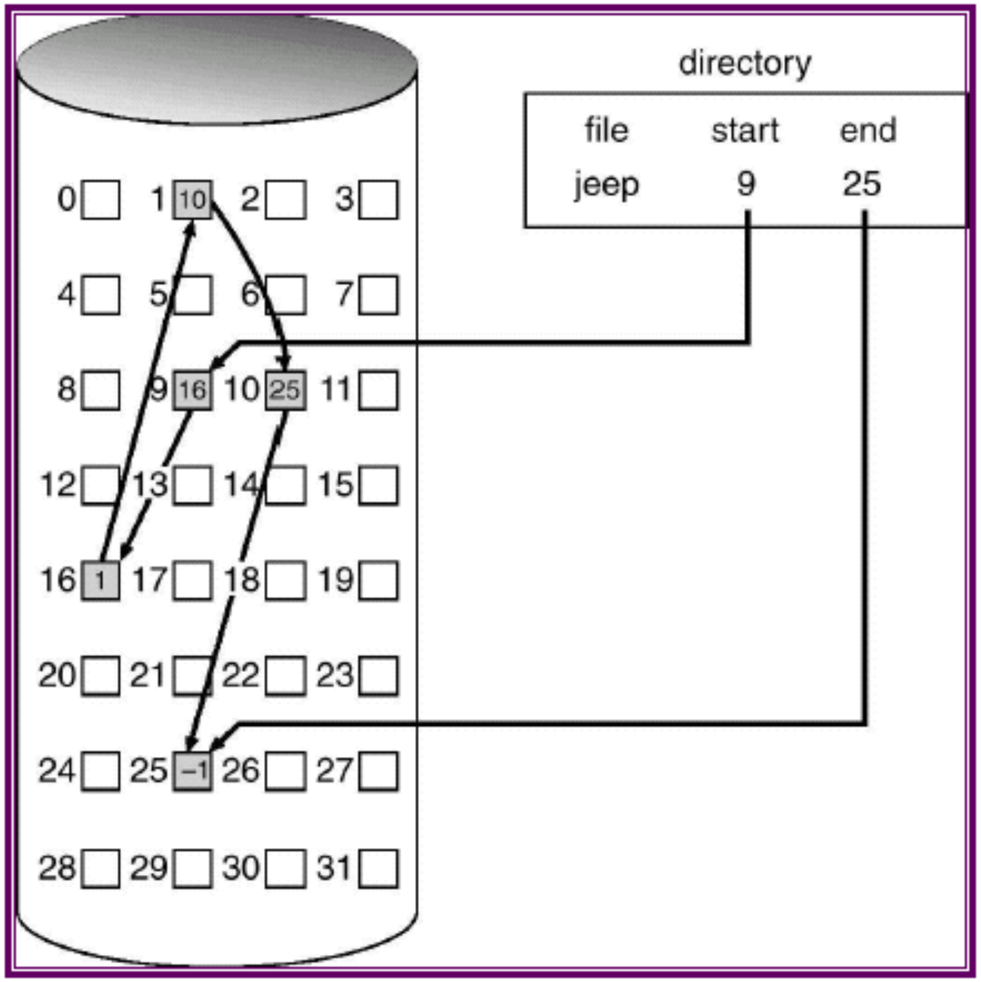
Linked Allocation



9[16]-16[1]-1[10]-10[25]-25[-1]
bad diagram!

Colorado State University

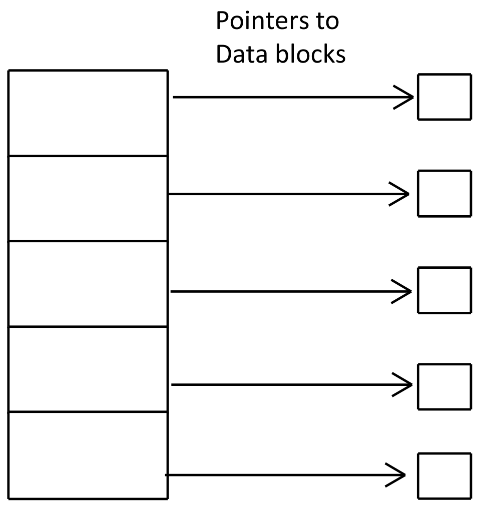
Linked Allocation



Correct version!

Allocation Methods - Indexed

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks
- Logical view



index table

Colorado State University