Lecture 7: Storage Management
File System Management

Contents

- Non-volatile memory
- Tape, HDD, SSD
- Files & File System Interface
- Directories & their Organization
- File System Implementation
- Disk Space Allocation
- File System Efficiency & Reliability

Non-volatile memory

- Non-volatile memory can get back stored information even when not powered.
- Non-volatile memory is typically used for the task of secondary storage, or long-term persistent storage.
- Examples of non-volatile memory from history:
  - paper tape and punched cards.
  - read-only memory, flash memory, ferroelectric RAM (F-RAM)
  - magnetic computer storage devices (e.g. hard disks, floppy disks, and magnetic tape)
  - optical discs (CD, DVD, BlueRay)

Magnetic tape data storage

- First used in 1951 to record data from UNIVAC I
- Development from 10.5 inch open reel to small closed cartridge
- Natural sequential reading and writing
- Suitable for backup of data
- Tape has the benefit of a comparatively long duration of the data stored on the media
- Capacity similar to HDD (5TB in 2011)

Hard Disk Drive

- History development from 8”, to 3.5” and 2.5”, to minimal 0.85” Toshiba in 2004 4GB and 8GB versions
- The head or heads on arm store information on magnetic medium on Platters

Hard Disk Drive

- Reading and writing time is similar
- Latency depends on:
  - Seek time – move arm to correct cylinder (2-10 ms)
  - Rotational latency – wait for correct head position on platter, depends on rotation speed (4.200 RPM – avg. 7.14ms, 7.200RPM – avg. 4.17ms, 15.000RPM – avg. 2ms)
  - Transfer time – time for reading the data from disk (0.2ms)
- Random reading 100KB/sec – need to make seek, wait for correct rotation and read data
- Random sector on the same cylinder – 200KB/sec – need only rotational latency and read time
- Next sector on the same cylinder – 4MB/sec – new disks 600MB/sec

Solid-State Drive

- SSD has no moving mechanical components
- More resistant to physical shock, run silently
- Most SSD's use NAND-based flash memory, which retains data without power
- From construction side it is RAM – random access memory
- No difference for sequential vs. random reading
- Big difference between reading and writing
- Reading 200 μsec
- Write can be only on erased pages and erasing need aprox .1.5 ms
- If SSD has free already erased page the write takes only 200 μsec. Otherwise, the write costs 1.7ms

Solid-State Drive

- Erase use “high” voltage - limited life time
- The cell can be erased 1k-100k times, depending on structure, SLC, MLC, TLC
- The firmware is responsible for uniform using of cells
- TRIM command – the OS can say to SSD, that this page is not used
- The firmware is the most important part of SSD
- The firmware makes
  - Mapping of linear space to SSD memory
  - Uniform usage of cells
  - Keep erased pages for fast writing

File Systems Interface

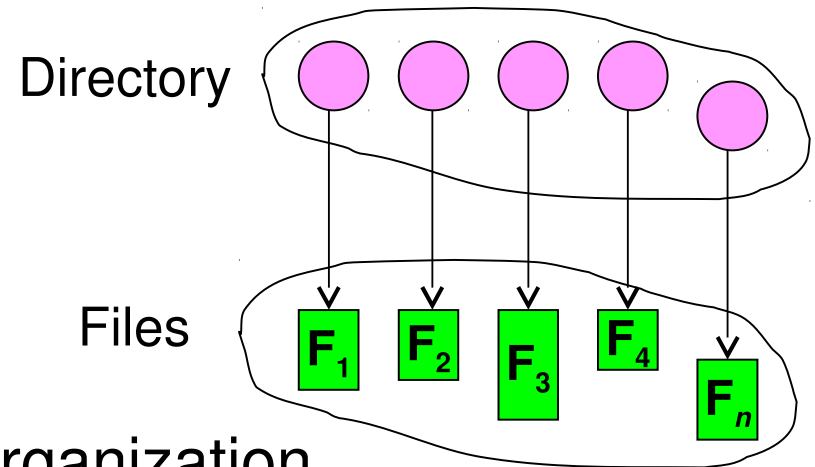
- Concept of the file
  - Contiguous logical address space
  - Types:
    - Data – numeric, character, binary
    - Program
- File Structure
  - None - sequence of words, bytes
  - Simple record structure – lines, fixed length records, variable length records
  - Complex Structures
    - Formatted documents, relocatable load files
  - Complex Structures can be simulated
    - by simple record structures through inserting appropriate control characters
    - by having special control blocks in the file (e.g., section table at the file beginning)

File Systems Interface (2)

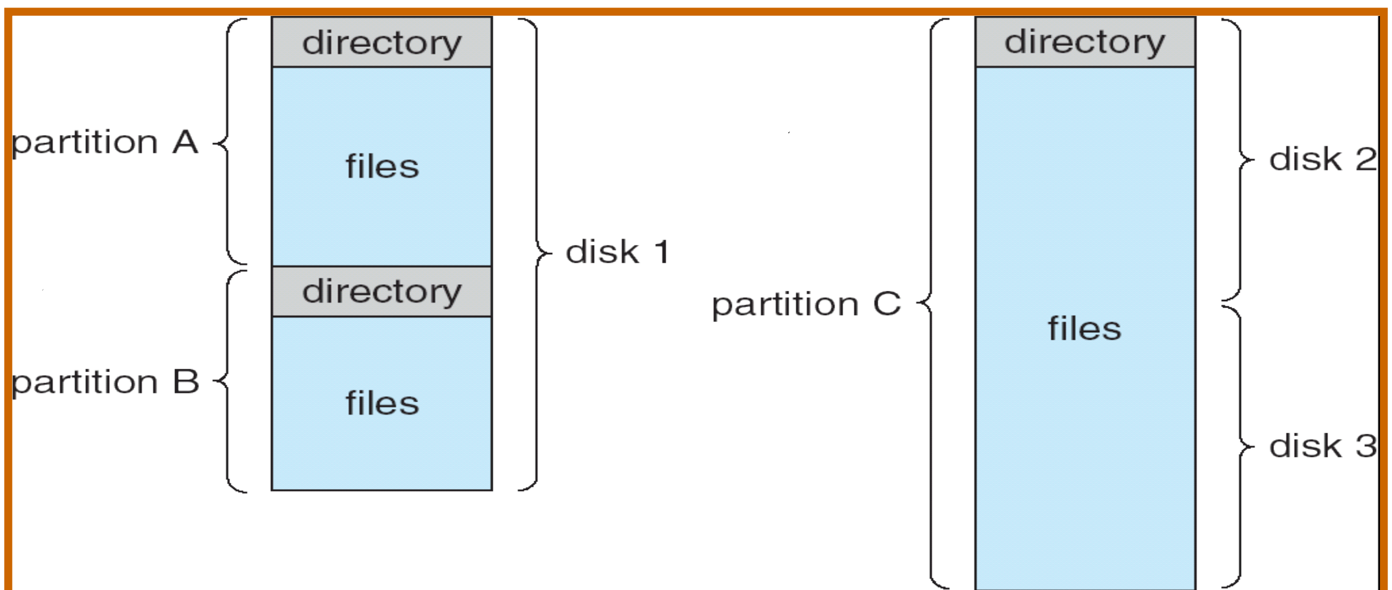
- File Attributes
  - *Name* – the only information kept in human-readable form
  - *Identifier* – unique tag (number) identifies file within file system
  - *Type* – needed for systems that support different types
  - *Location* – information on file location on a device
  - *Size* – current file size
  - *Protection* – for control who can do reading, writing, executing
  - *Time, date,* and *user identification* – data for protection, security, and usage monitoring
  - Information about files is kept in the file-system structures, which are stored and maintained on the disk
- File Operations – exported by the OS API (cf. e.g., POSIX)
  - *Open*(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory
  - *Write, Read, Reposition within file*
  - *Close*(F_i) – move the content of entry F_i in memory to directory structure on disk
  - *Delete, Truncate*
  - etc.

Directory Structure

- Directory is a collection of nodes containing information about files
  - Both the directory structure and the files reside on disk



- A Typical File-system Organization

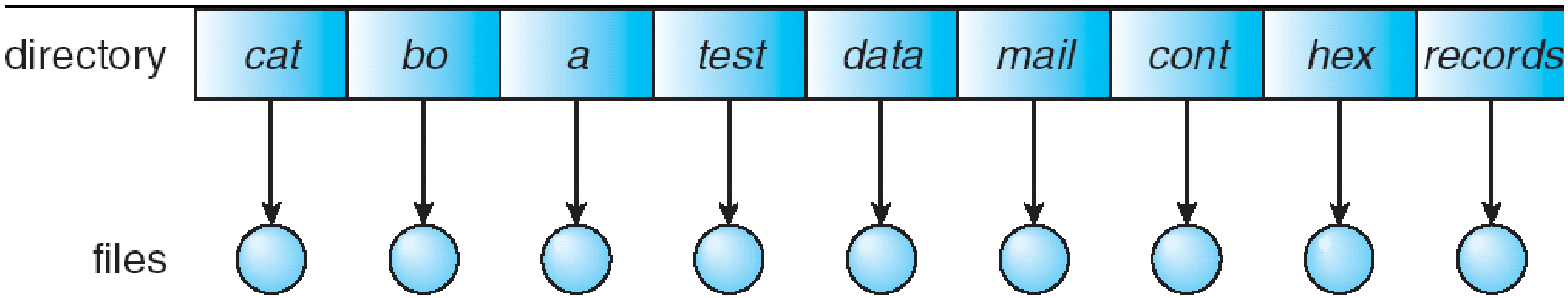


Logical Organization the Directories

- Operations Performed on Directory
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system
- Organize directories to get
  - **Efficiency** – locating a file quickly
    - The same file can have several different names
  - **Naming** – convenient to users
    - Two users can have same name for different files
  - **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

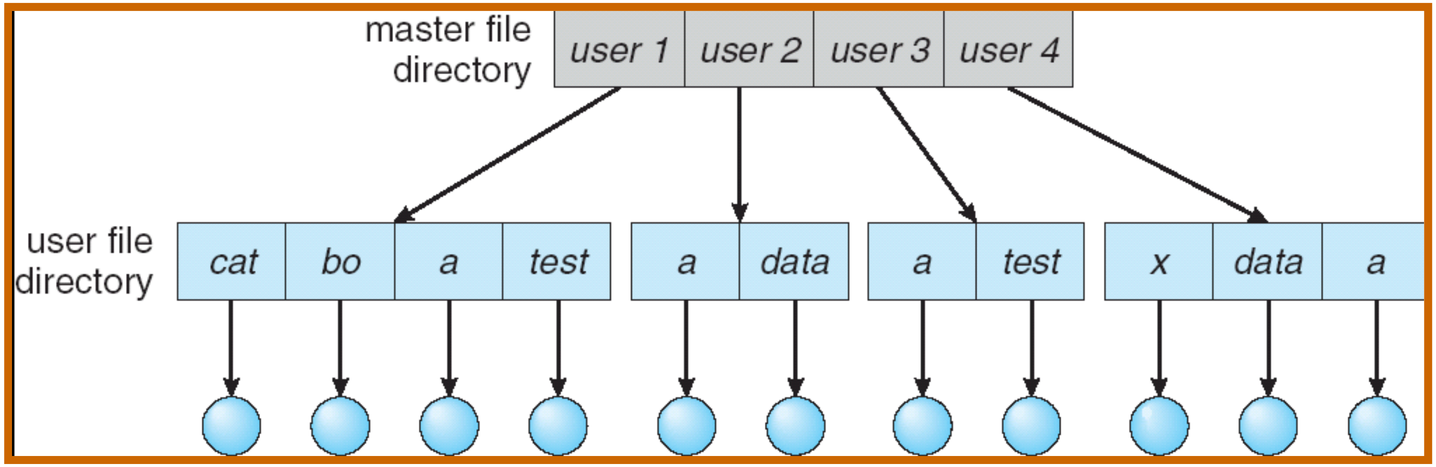
- A single directory for all users



- Easy but
  - Naming problem
  - Grouping problem
  - Sharing problem

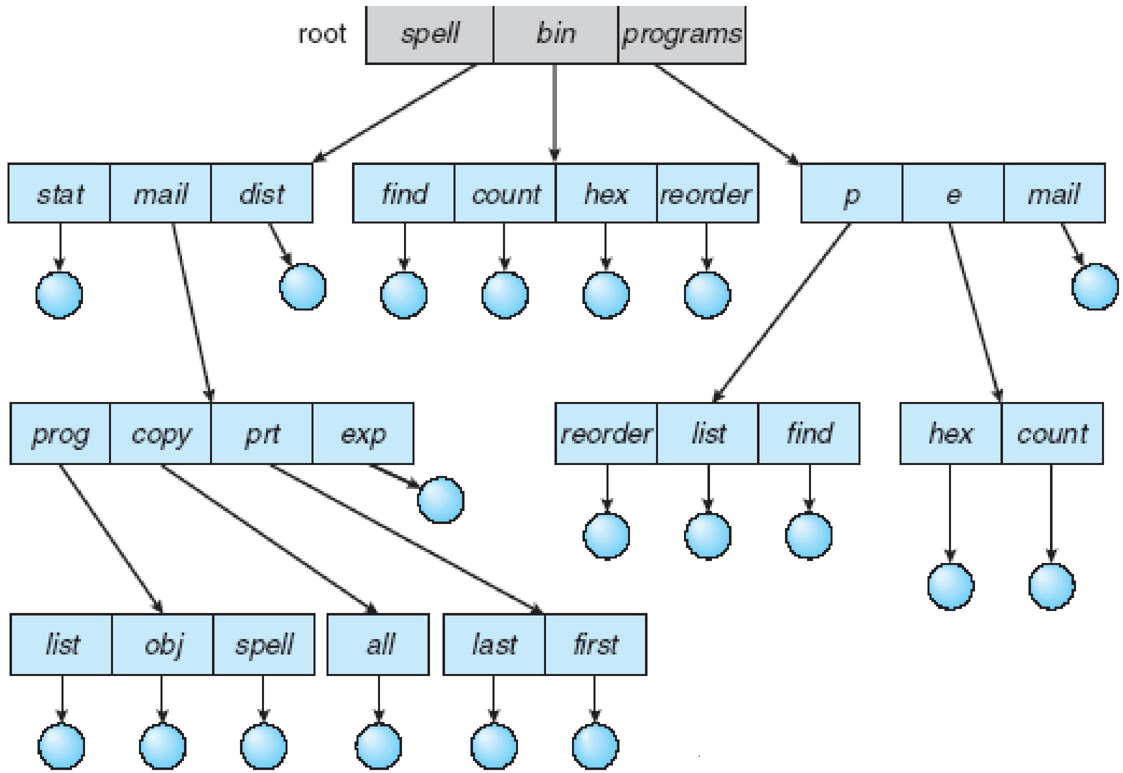
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

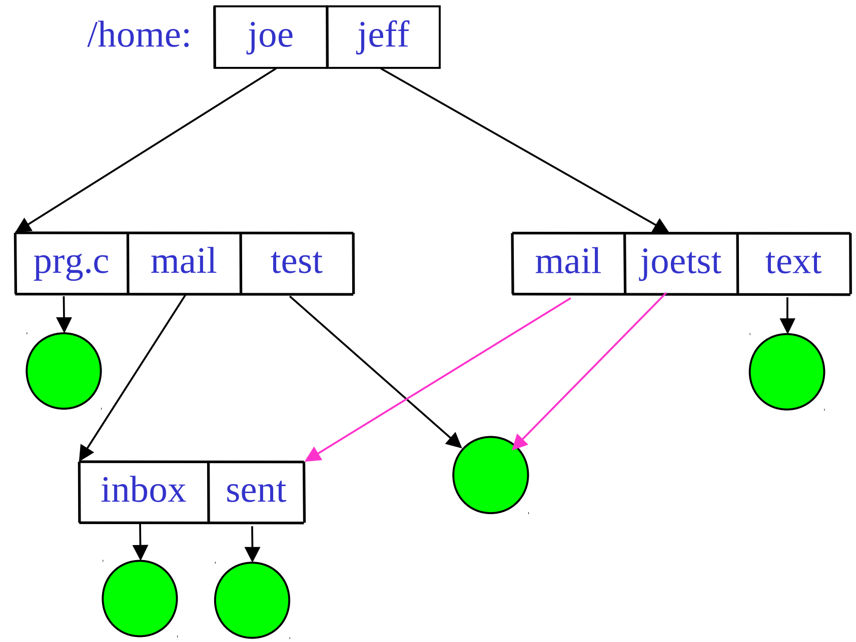
Tree-Structured Directories



- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - cd /spell/mail/prog
  - type list

Acyclic-Graph Directories

- Have shared subdirectories and files
  - *aliasing* – an object can have different names

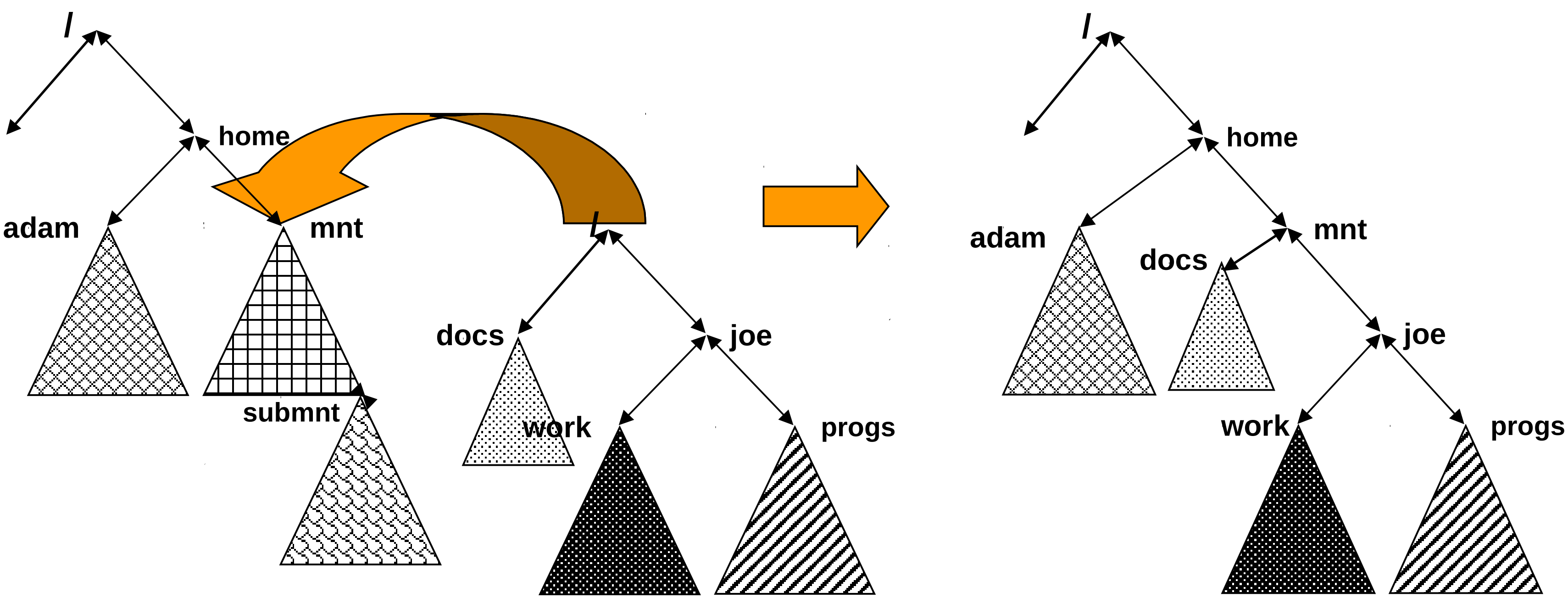


- Problem:
  - When 'joe' deletes file 'test', the directory item 'joetst' points wrong
  - Solution:
    - Each object has a counter containing a count of references.

      The counter increments when a new reference is created and decrements when a reference is deleted. The object is erased when the counter drops to zero

File System Mounting

- A file system must be **mounted** before it can be accessed
  - E.g., file system on a removable media must be 'announced' to the OS, i.e. must be mounted
  - Have prepared a **mount point** – a directory
    - Anything referenced from the mount-point before mounting will be hidden after mounting

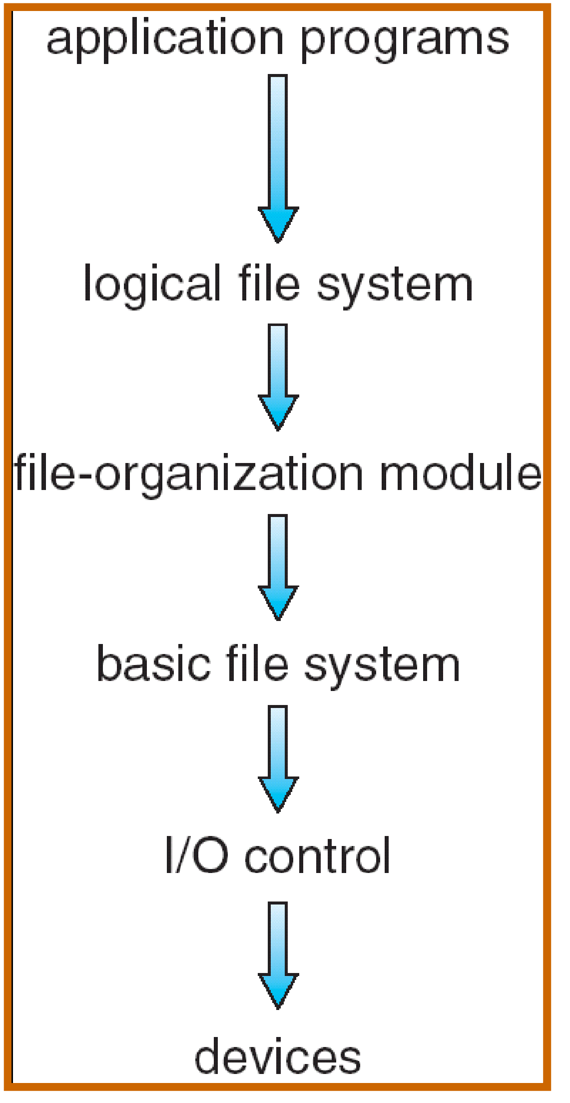


File Sharing

- 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
- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights
  - POSIX **`rwx|rwx|rwx`** scheme
    `U   G   O`
  - ACL – Access Control Lists (Windows, some UNIXes)

File System Implementation Objectives

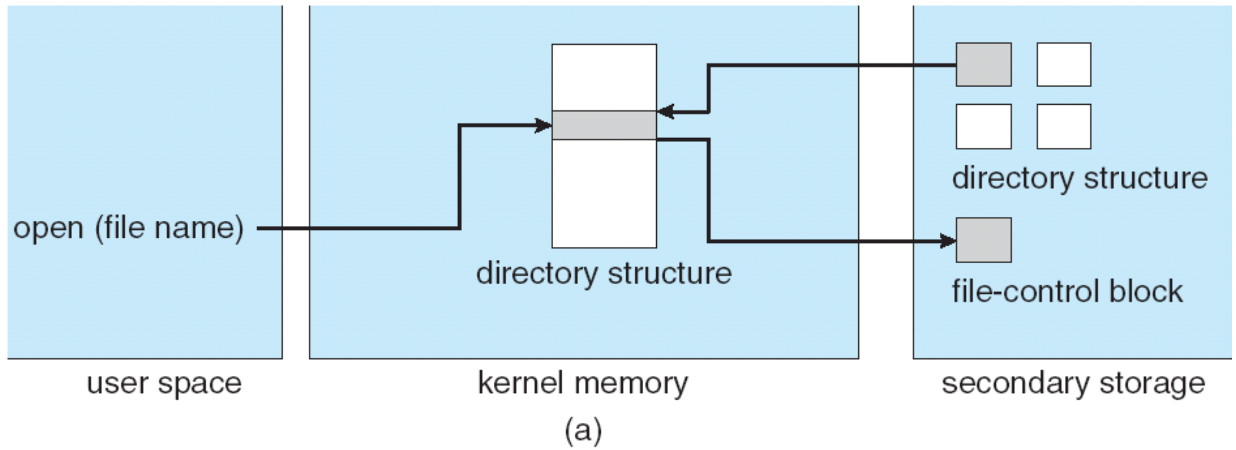
- Implementation possibilities of local file systems and directory structures
- File block allocation and free-block strategies, algorithms and trade-offs
- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system is organized into layers
- **File control block** – storage structure consisting of information about a file
  - Size, ownership, allocation info, time stamps, ...



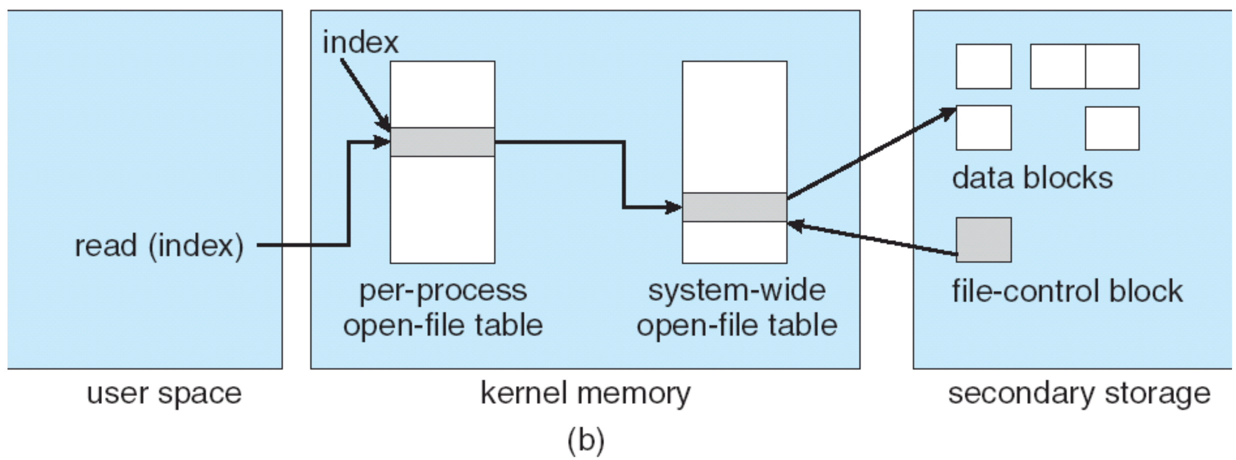
In-Memory File System Structures

- The following figure illustrates the necessary file system structures provided by the operating systems.

opening a file

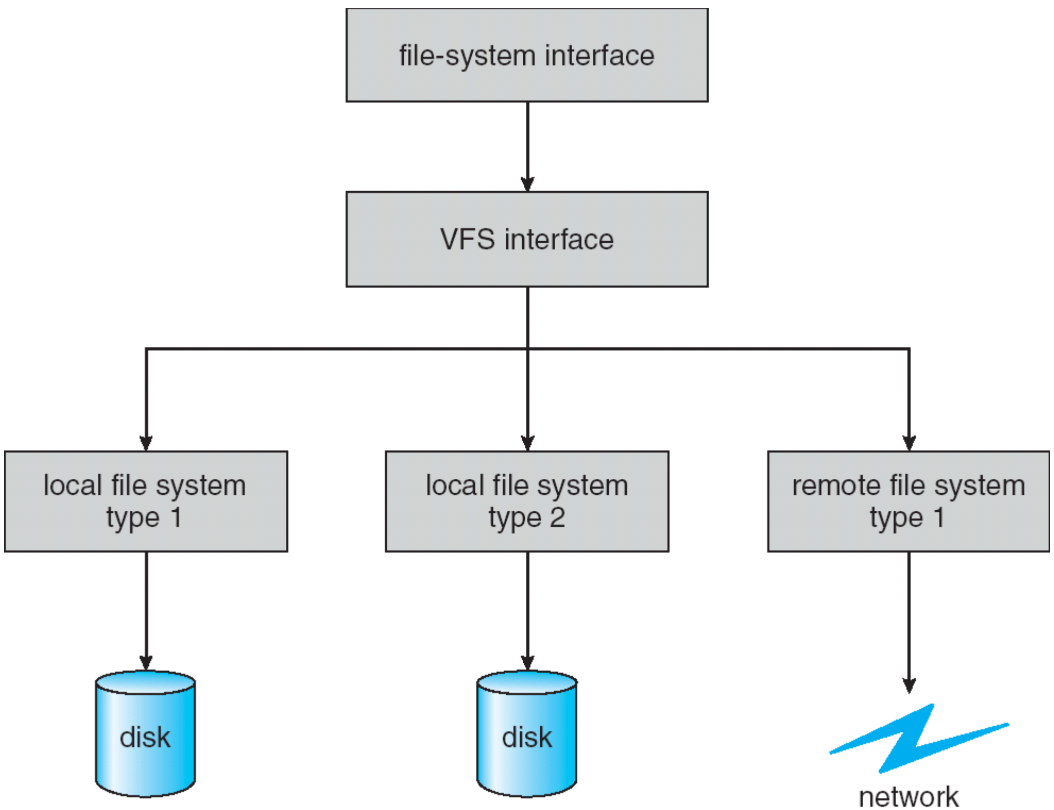


reading a file



Virtual File Systems

- Virtual File Systems (VFS) 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 is to the VFS interface, rather than any specific type of file system.



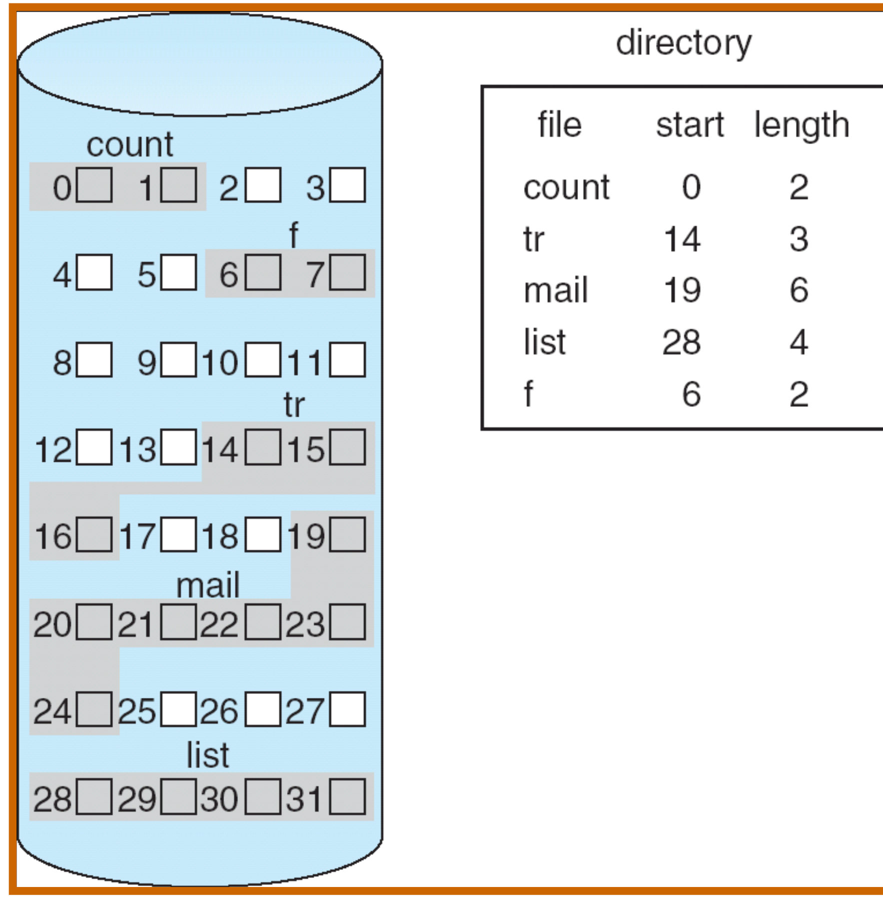
Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size
- **Complex data structure** – e.g., B+ tree
  - NTFS in MS Windows

Allocation Methods for Files

- An allocation method refers to how disk blocks are allocated for files:
  - **Contiguous allocation**
  - **Linked allocation**
  - **Indexed allocation**

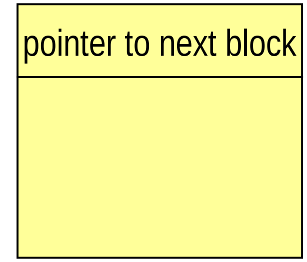
- **Contiguous allocation** – simple to implement
  - Each file occupies a set of contiguous blocks on the disk
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Random access
  - Wasteful of space (dynamic storage-allocation problem)
  - Files cannot grow

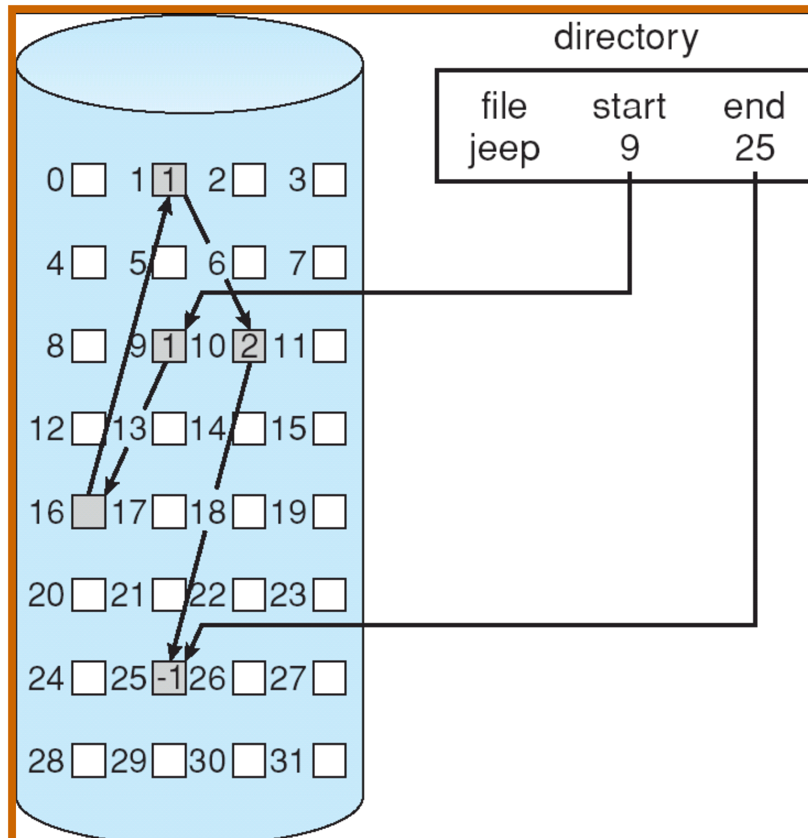


Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Simple – need only starting address
- Free-space management system – no waste of space
- Difficult random access
  - must go through the whole chain

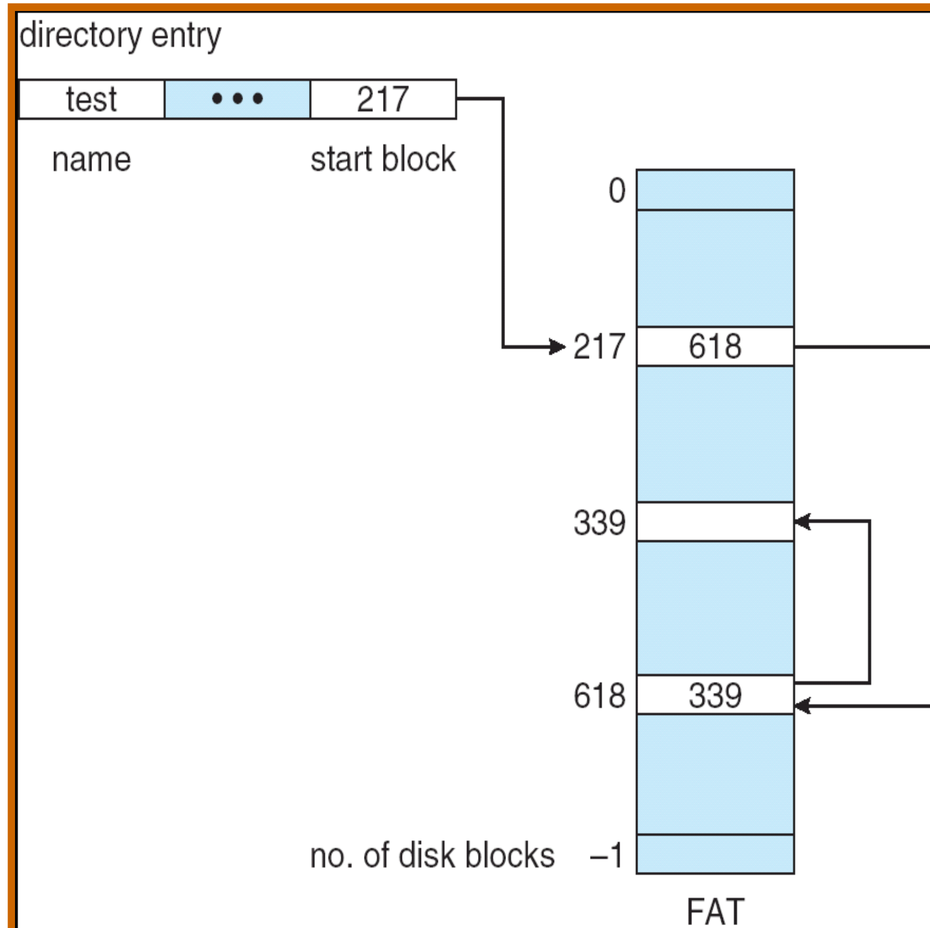
block =





Linked Allocation with FAT

- Allocation chains stored separately
- File-allocation table (FAT)
  - Disk-space allocation used by MS-DOS and OS/2.
- Problems:
  - Size of the table
  - Access speed
  - Reliability
    - All file info is concentrated in one place
    - FAT duplicates



Allocation block size with FAT

- Allocation block, cluster
  - group of adjacent disk sectors
- Fixed size of FAT on disk
- Different FAT types
  - FAT item has 12, 16 or 32 bits
  - Directory entry (MSDOS):

FAT-16	8 bytes	3	1	10	4	2	4
	Name	Extension	Attrs	Reserved	Date and time	1st block	File size

- Addressing capability of different FAT types

Block size	FAT-12	FAT-16	FAT-32
0.5 KB = 1 sector	2 MB	a)	
1 KB = 2 sectors	4 MB		
2 KB = 4 sectors	8 MB	128 MB	
4 KB = 8 sectors	16 MB	256 MB	1 TB
8 KB = 16 sectors	b)	512 MB	2 TB
16 KB = 32 sectors		1 GB	2 TB
32 KB = 64 sectors		2 GB	2 TB

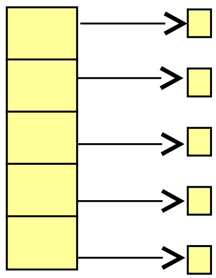
Empty entries in the table are unused because:

a) FAT is too large compared to the disk capacity

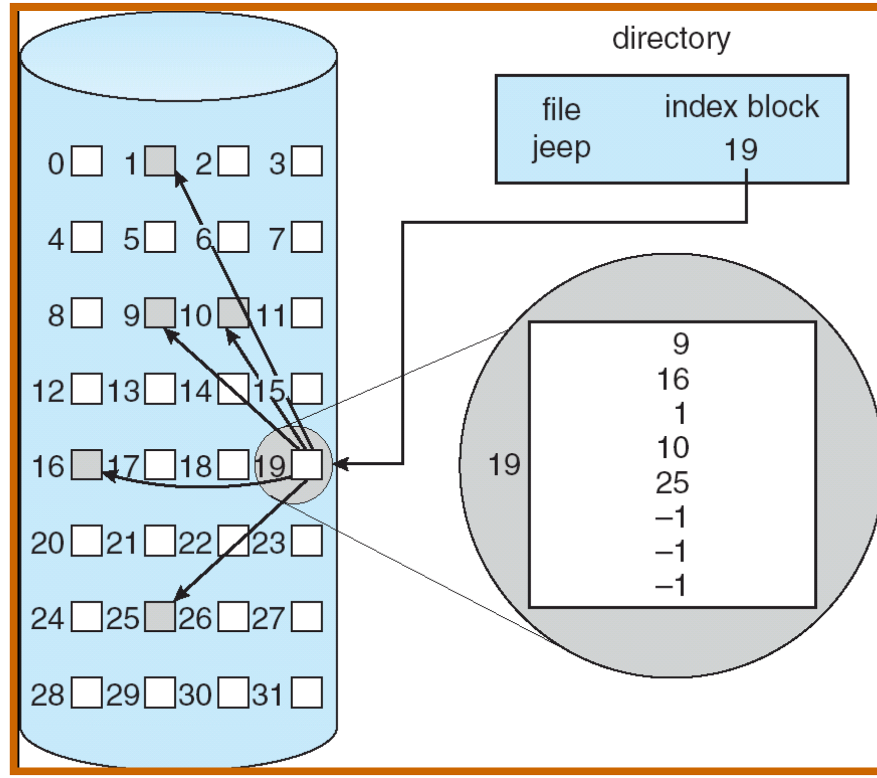
b) losses due to internal fragmentation are to high

Indexed Allocation

- Brings all pointers for one file together into an *index block.*
- Logical view



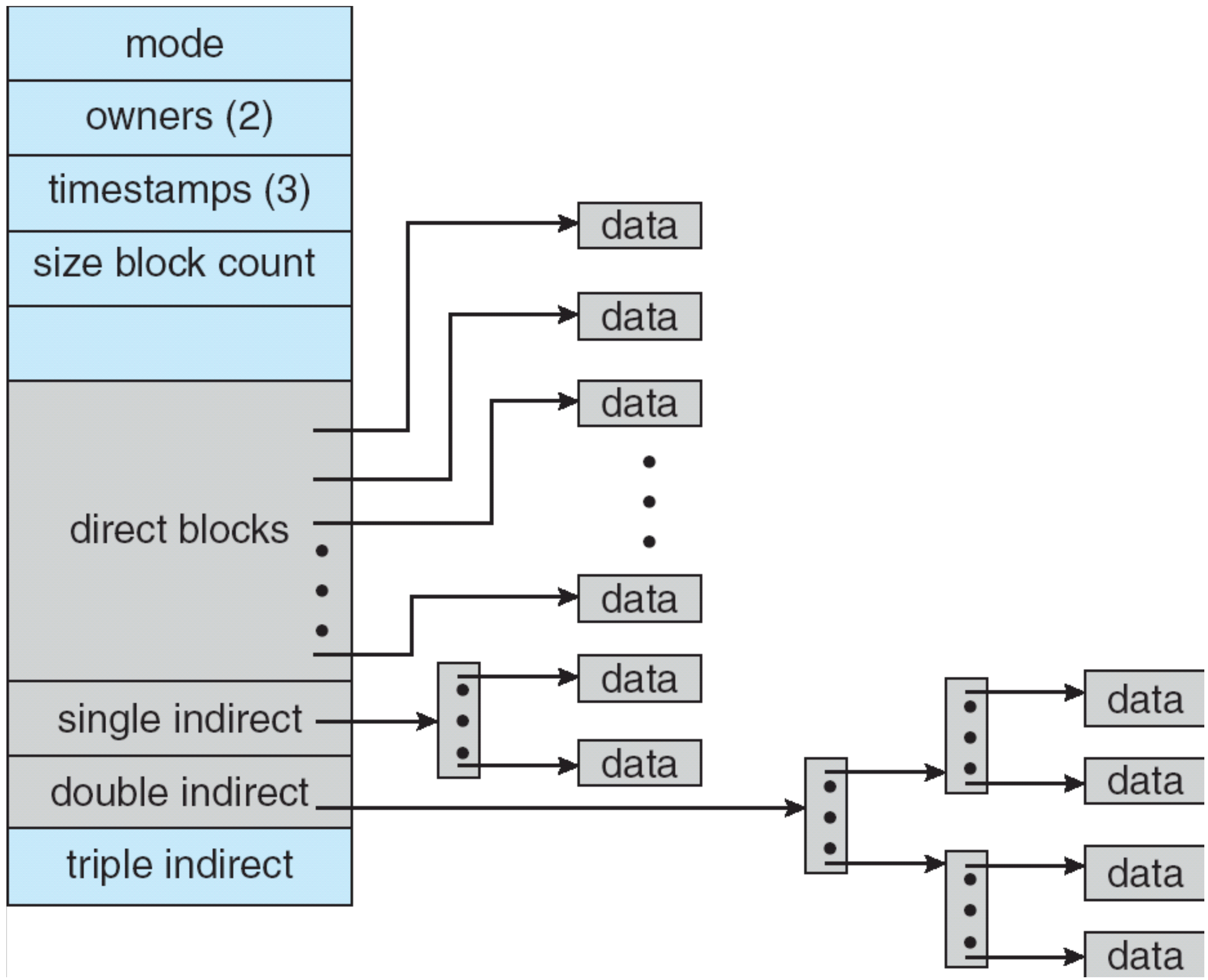
index table



- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
  - Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table
  - Only “small” files

Combined Scheme: UNIX FS

- Disk i-node
  - 4K bytes per block



NTFS

- Database structure
- File has attributes – name, time modification, data stream
- Everything is file
- Master File Table contain information about all files
- Master File Table is file too.
- MFT contains information about itself
- Resident attributes are stored in MFT
- Non-resident are on disk according allocation map
- If the file is too much fragmented and the allocation map cannot be in MFT so allocation map becomes non-resident and is moved on disk and mapped by another allocation map

NTFS vs. FAT

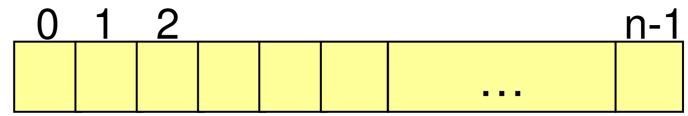
- NTFS is for large disk >500MB
- FAT has less memory overhead
- FAT is more simple and operation are more effective
- NTFS has save file descriptor
- NTFS has transaction recovery
- NTFS has B+tree for directory structure – fast for big directories

Extent-Based Systems

- Many newer file systems (e.g., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**
- An **extent** is a contiguous block of disks
  - Extents are allocated for file growth
  - A file consists of one or more extents

Free-Space Management

- Bit vector (*n* blocks) – one bit per block
  - Bit map requires extra space
  - Easy to get contiguous files



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

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

- Need to protect:
  - Pointer to free list
  - Bit map
    - Must be kept on disk
    - Copy in memory and disk may differ
    - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
  - Solution:
    - Set bit[*i*] = 1 in disk
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

Directory Implementation

- Linear list of file names with pointer to the data blocks
  - simple to implement
  - time-consuming to execute
  - directory can grow and shrink
- Hash Table – linear list with hash data structure
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

File System Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Recovery from a Crash

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to **back up** data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

Log Structured File Systems

- **Log structured** (or journaling) file systems record each update to the file system as a **transaction**
  - similar to database systems
- All transactions are written to a **log**
  - A transaction is considered **committed** once it is written to the log
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed
- Used by NTFS file system