CSE 380
Computer Operating Systems

Instructor: Insup Lee and Dianna Xu

University of Pennsylvania
Fall 2003
Lecture Notes: File Systems

File Systems

- Computer applications need to store and retrieve information:
  - need to store large amount of information
  - need to store information permanently
  - need to share information
- A file is a collection of data records grouped together for purpose of access control and modification
- A file system is software responsible for creating, destroying, organizing, reading, writing, modifying, moving, and controlling access to files; and for management of resources used by files.

User-Level View

- Naming convention
- File structures
- File types
- File access: sequential vs random access
- File operations:
  - system calls (file/directory operations)
- Memory-mapped files
- Directory structure (single-level vs. two-level vs. hierarchical
  - path names
  - directory operations

File Naming

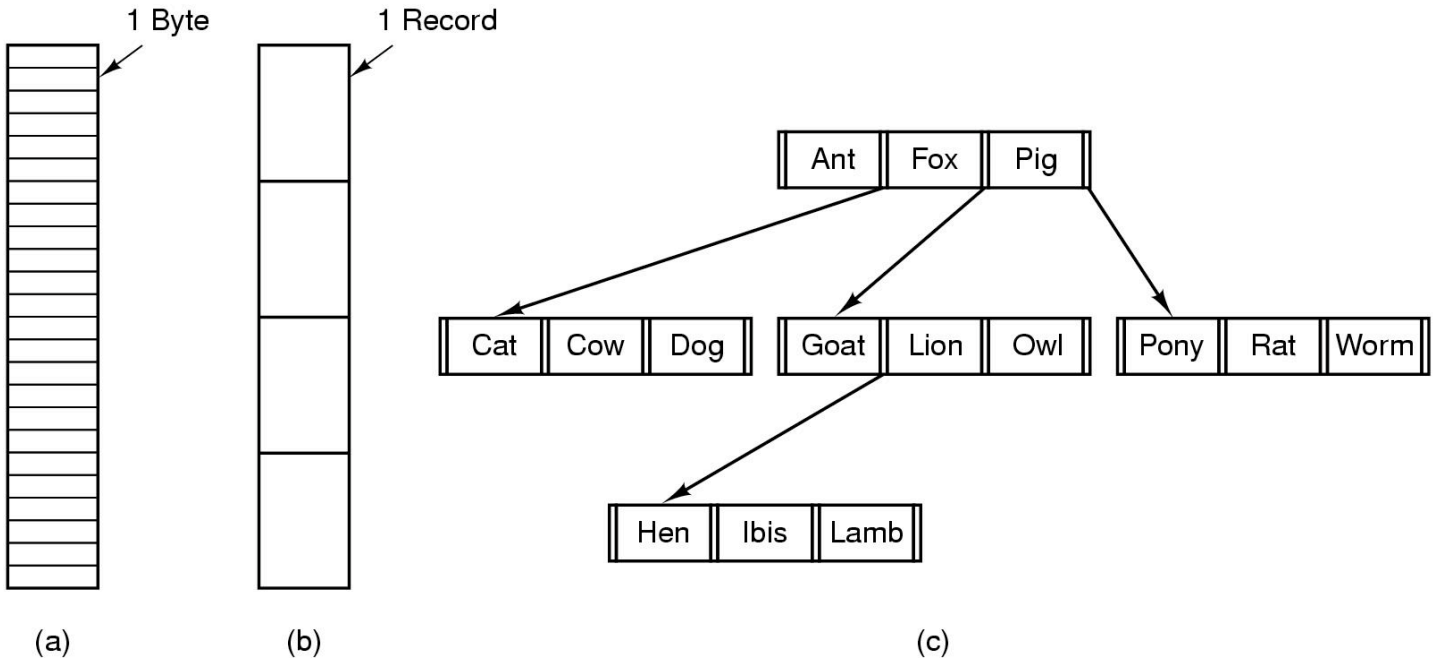
- Naming convention
  - number of characters (e.g. limited to 8+3 in MS-DOS)
  - case sensitive or not, Which chars are allowed
  - special prefixes/extensions (.txt, .ps, .gif, .mpg, …..)
- The family of MS-DOS
  - Win3.1, Win95, Win98
  - NT, Win2000 (supports MS-DOS, but have native file system NTFS)
- In Unix, many extensions are just conventions
  - exceptions are for example compilers
- Windows assigns meaning to extensions

File Naming

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Typical file extensions.

File Structure

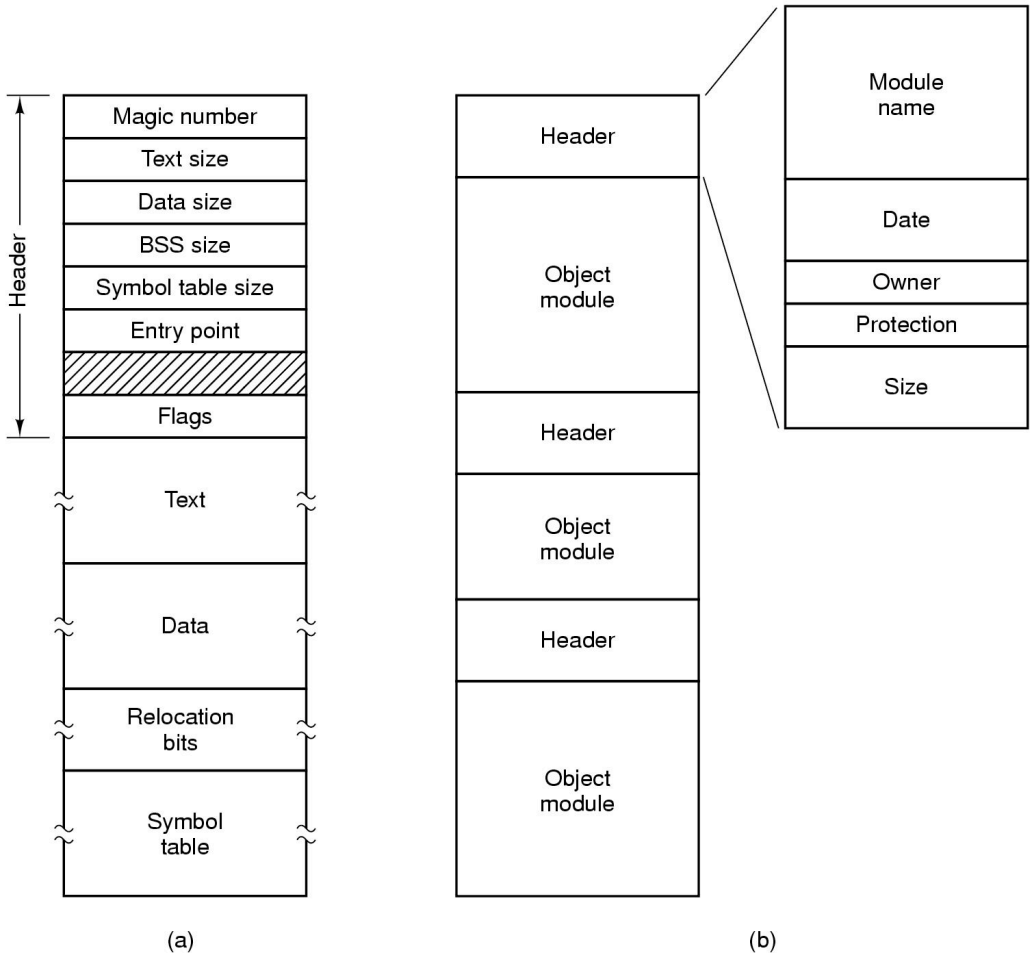


- Three kinds of files
  - byte sequence ⟹ both Unix and Windows
  - record sequence: when 80-column punch cards were king
  - tree: data processing on large mainframe

File Types

- Unix
  - regular files
    - ASCII
    - binary
  - directory files
  - character special files (I/O devices that operate on streams)
  - block special files (disk I/O)
- Every OS must at least recognize its own executables
  - Unix: header, text and data
  - magic numbers

File Types



(a) An executable file (b) An archive

File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was magnetic tape
- Random access
  - bytes/records read in any order
  - essential for many applications
  - read can be …
    - move file pointer (seek), then read or …
    - read and then move file marker
  - all modern OS have all files as random access

File Attributes

- File name
- Size information (current, limit)
- Physical address
- File type
  - ASCII vs binary
  - Temporary vs Permanent
- Access rights: owner, protection (who can access it)
- Access type: Sequential/Random
- History: Creator, time of last access/modification, other usage data
- Info for managing links

File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Possible file attributes

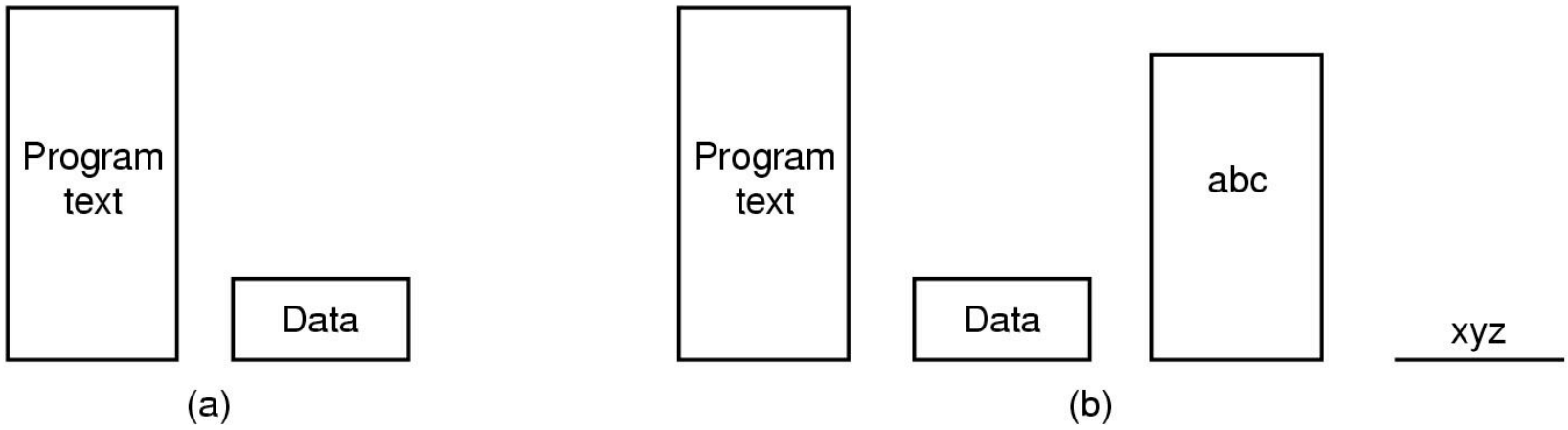
File Operations

1. Create (creat)
2. Delete (unlink)
3. Open
4. Close
5. Read
6. Write
7. Append (write)
8. Seek (lseek)
9. Get attributes (stat, lstat, fstat, fcntl)
10. Set Attributes (fcntl)
11. Rename

Memory Mapped Files

- Instead of making a series of system calls that involve I/O, map files into the address space of a running process
- Just two system calls, map and unmap
- File I/O can be done in simple instructions that address memory as usual
- Problems:
  - files must fit in memory
  - modifications will not be written to disk until unmapped

Memory-Mapped Files



(a) Segmented process before mapping files into its address space

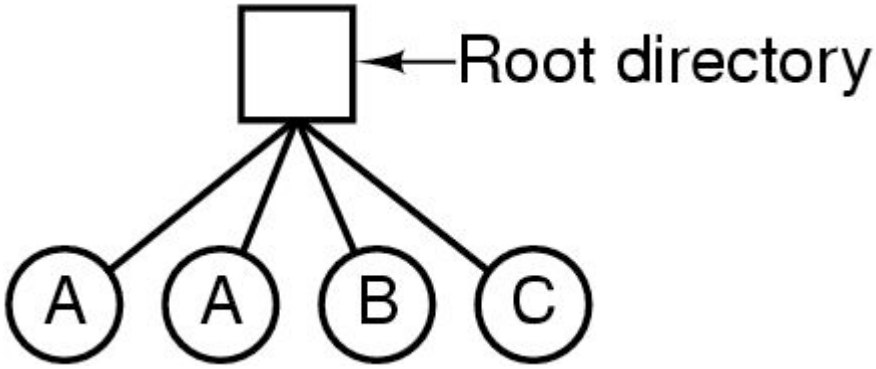
(b) Process after mapping

existing file *abc* into one segment

creating new segment for *xyz*

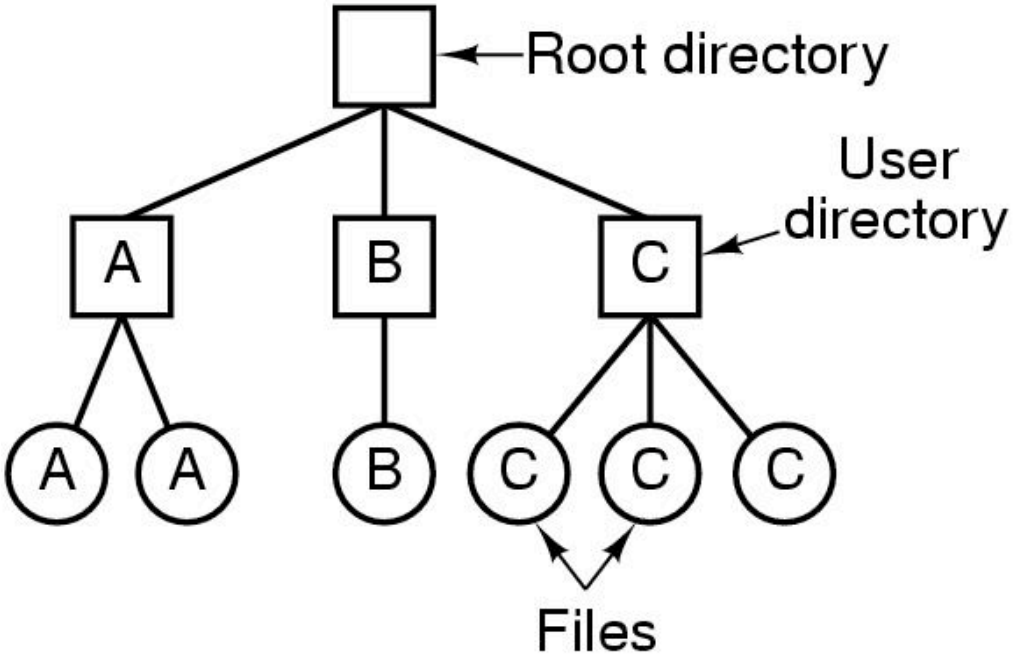
Directories

Single-Level Directory Systems



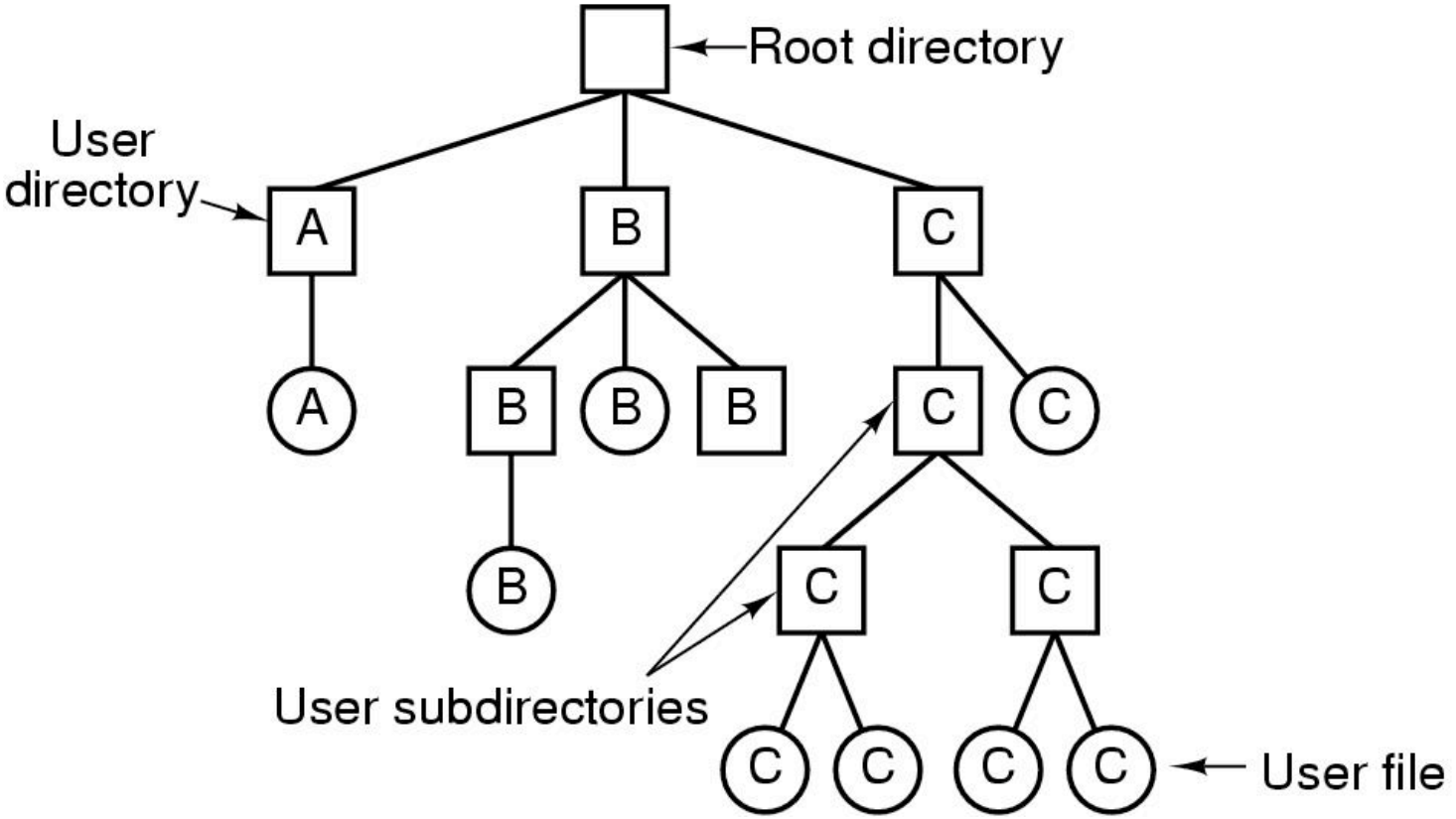
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C
  - ownerships are shown, not file names

Two-level Directory Systems



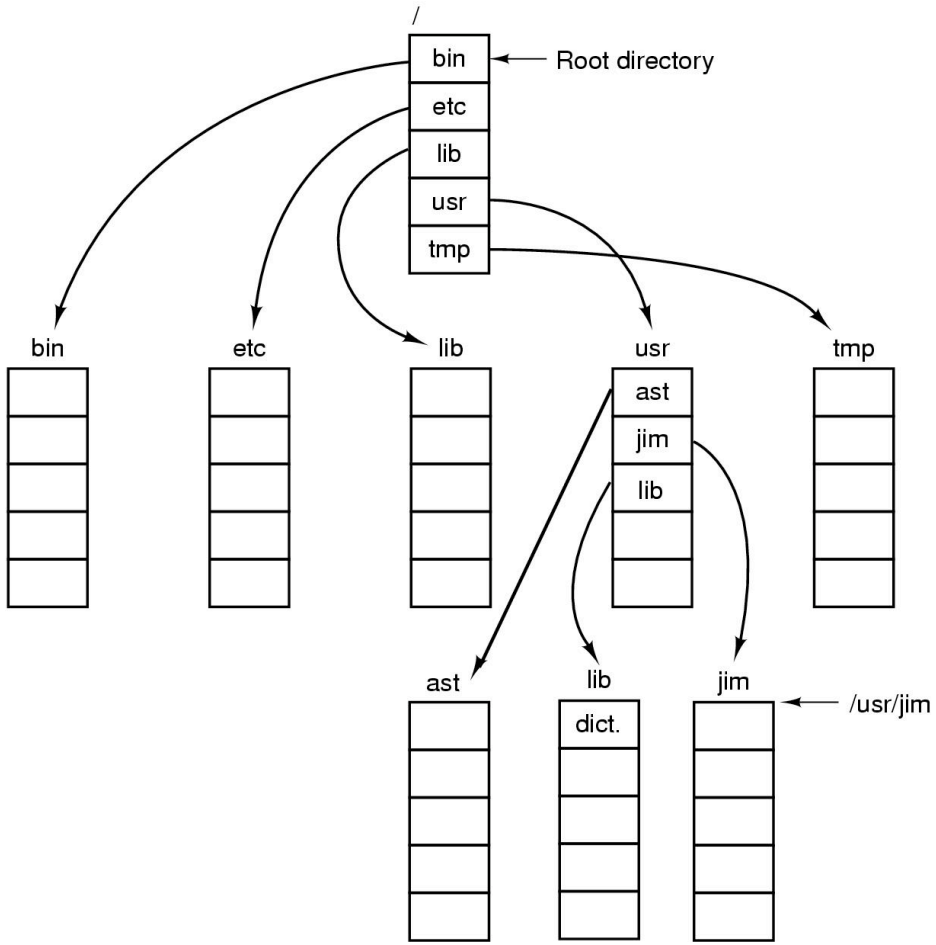
Naming conflicts between different users are eliminated

Hierarchical Directory Systems



A hierarchical directory system

Path Names



A UNIX directory tree

Directory Operations

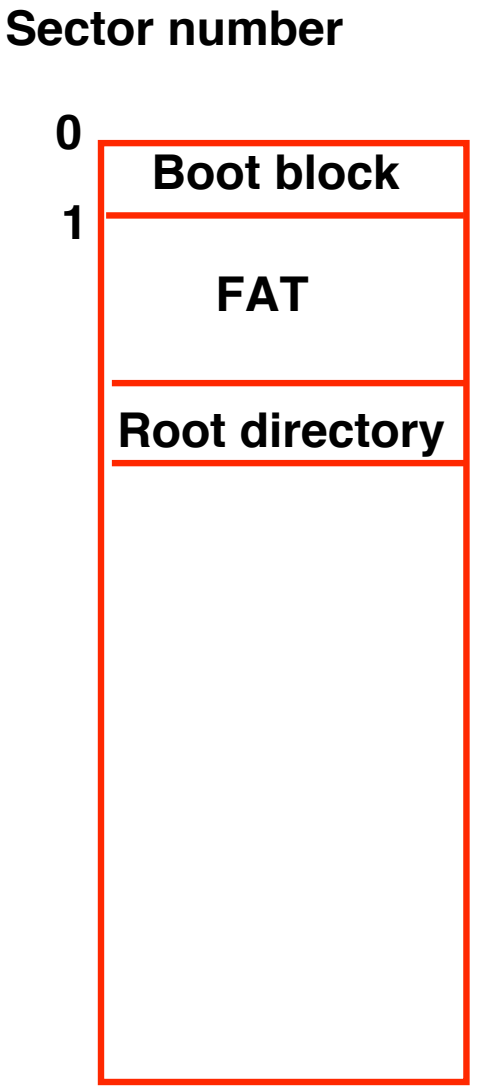
1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

File System Implementation

- Sector 0 is called the Master Boot Record
  - used to boot the computer
  - contains partition table at the end
  - one partition must be marked as active/primary
- BIOS (located on the parentboard) reads and executes MBR (after trying to boot from floppy or CD-ROM)
- MBR locates the active partition and reads in its first block
- Every partitions comes with a boot block

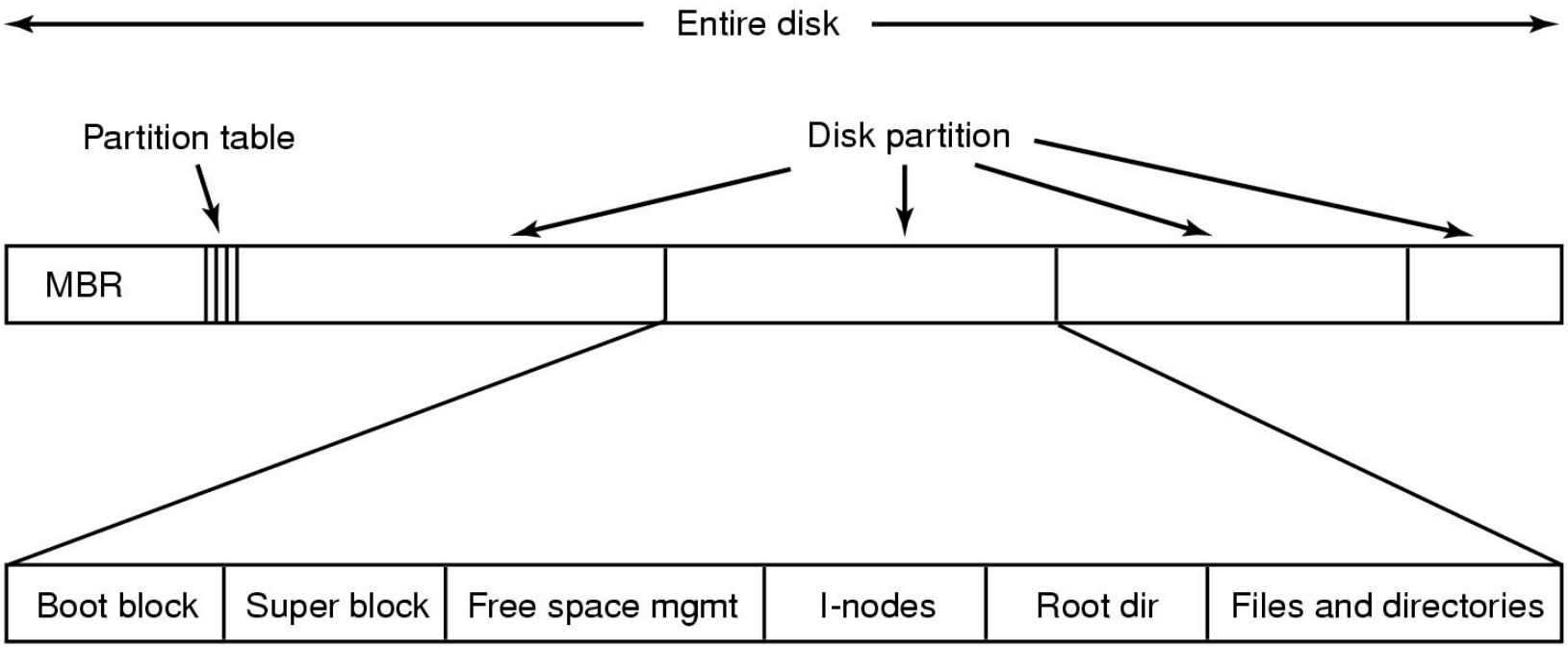
Booting and Disk Layout

- ROM (read-only, persistent) memory contains a small bootstrap loader program
- System disk contains boot block in first block of each partition
- Boot block has bootstrap executable
- Bootstrap loader copies bootstrap program into memory
- Bootstrap program initializes all registers, finds OS kernel on disk, loads OS, and jumps to the initial address of OS
- Why is bootstrap not in ROM?



MS-DOS disk layout

File System Layout



A possible file system layout w*ith a Unix partition*

Disk Space Organization

- Disk can be partitioned
  - Each partition can have a different OS and/or different file system
  - One partition can be swap space for main memory
- First block of disk has master boot record specifying primary partition
- Each partition has
  - Boot block (loads OS in kernel space)
  - Superblock (contains key info about file system which is read into memory at boot time)
  - Free space management
  - List of I-nodes (or other data structure) giving info about all files
  - Directories and Files

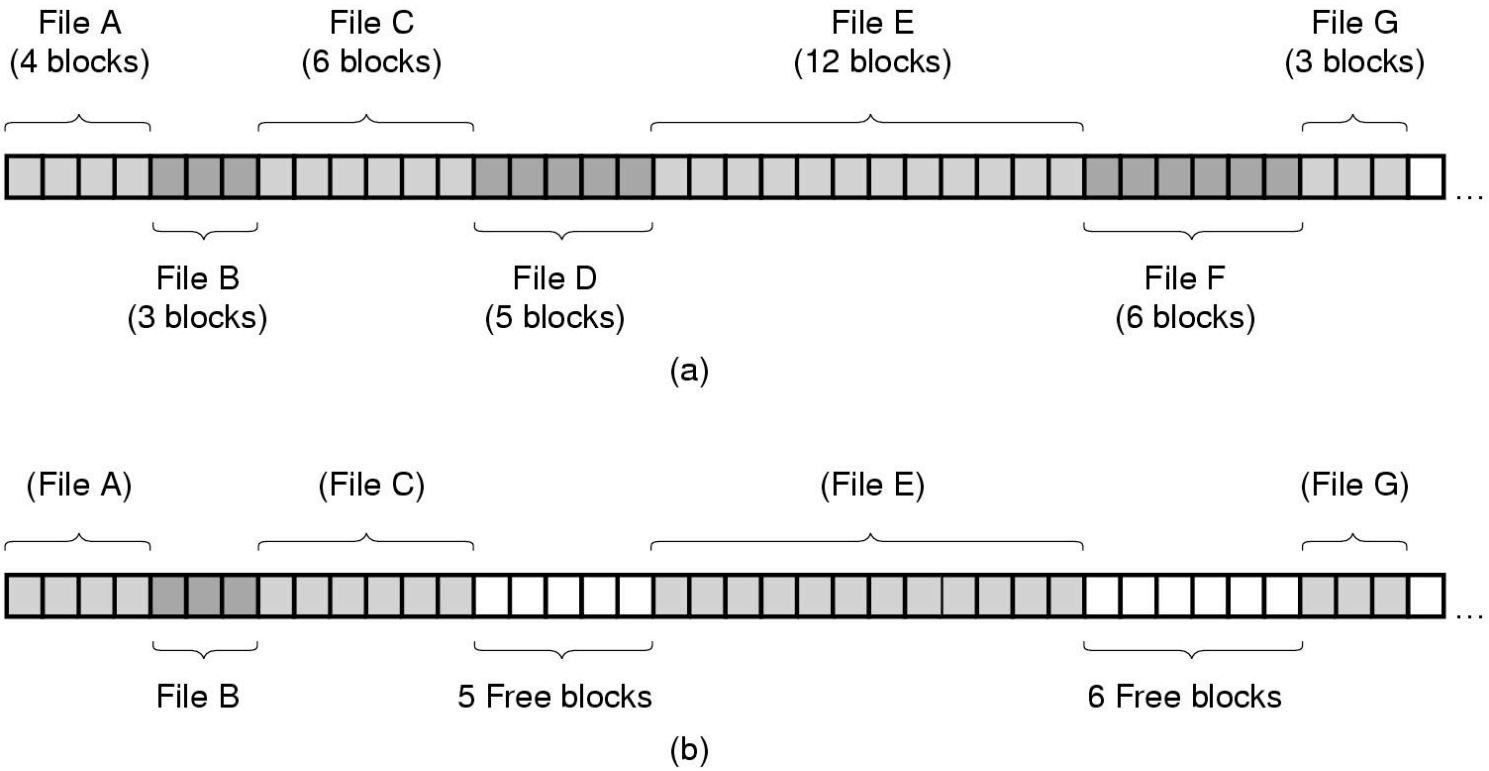
File Space Allocation

- Goals
  - Fast sequential access
  - Fast random access
  - Ability to dynamically grow
  - Minimum fragmentation
- Standard schemes
  - Contiguous allocation (fixed)
  - Linked list allocation
  - Linked list with file allocation table (FAT)
  - Linked list with Indexing (I-nodes)

Contiguous Allocation

- Each file occupies a contiguous region of blocks
- Fast random access (only one seek needed)
- Useful when read-only devices or small devices
  - CD-ROMs, DVD-ROMs and other optical media
  - Embedded/personal devices
- Management is easy
  - Directory entry of a file needs to specify its size and start location
- Fragmentation is a problem if deletes are allowed, or if files grow in size
- After disk is full, new files need to fit into holes → advanced declaration of size at the time of creation

Implementing Files (1)



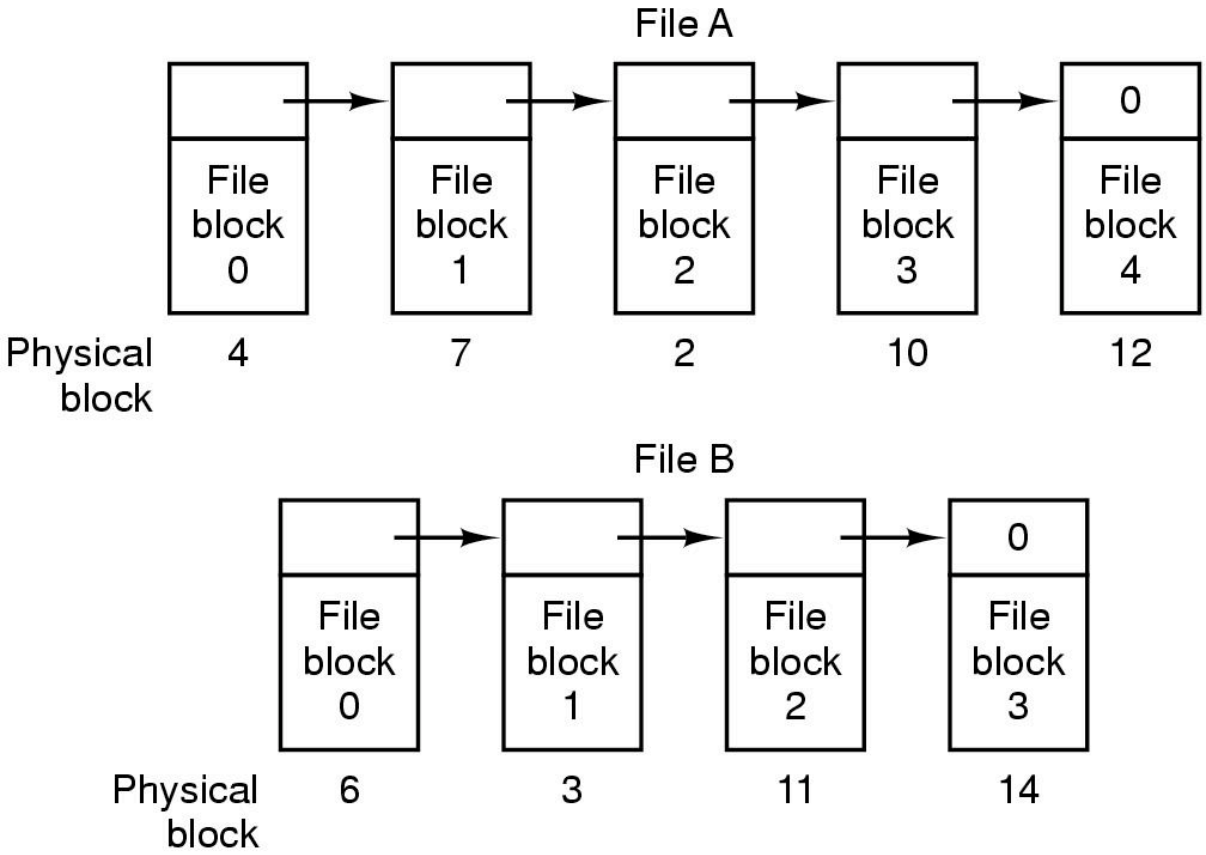
(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *E* have been removed

Linked Lists

- Natural choice: maintain a linked list of blocks that contain each file
- Directory entry of a file points to first block, and each block begins with a pointer to next block
- No external fragmentation
- Speed of random access is slow
  - Accessing ν^{th} block requires ν disk accesses, i.e. ν-1 accesses for pointers
- Data size in a block is no longer power of 2 because of pointer storage

Implementing Files (2)

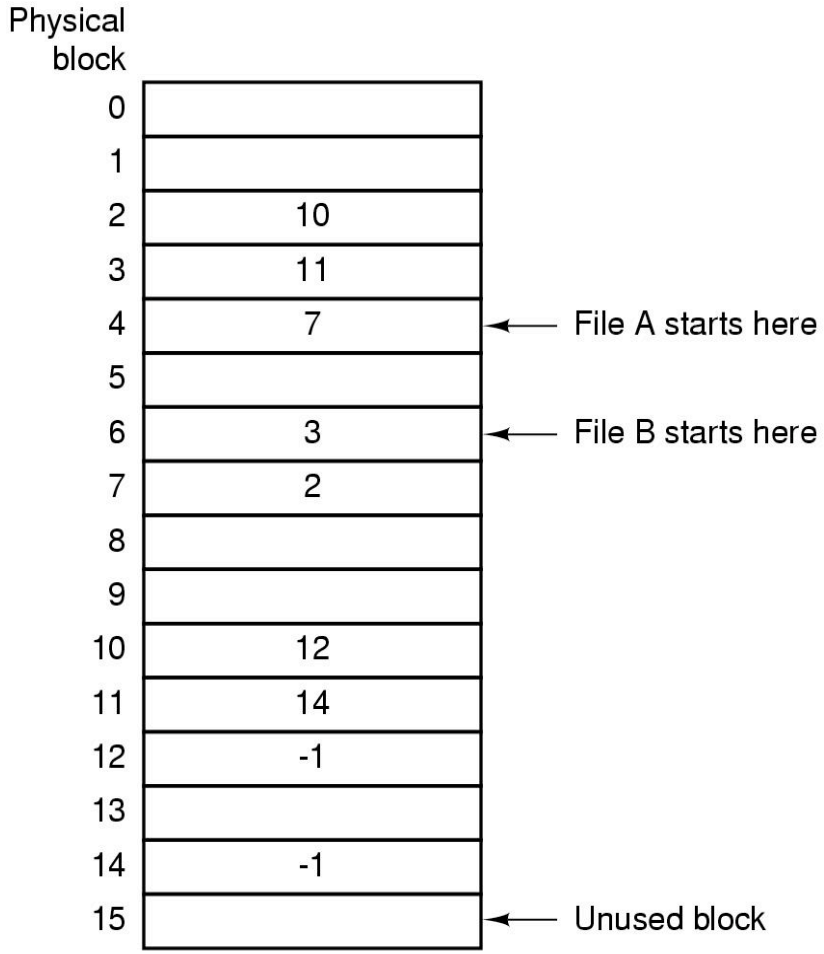


Storing a file as a linked list of disk blocks

Linked List with FAT

- So pointers are trouble, don't store them in blocks!
- Solution: Maintain a File Allocation Table in main memory
  - FAT is an array indexed by blocks
  - Each array entry is a pointer to next block
- Accessing v^{th} block would still require traversing the chain of pointers, however, they are in main memory and no disk I/O is needed
- Problem: the entire FAT must be in memory, and as disk size increases, so does FAT
  - 20GB disk with 1k block size needs 20 million entries, which requires entry sizes of minimum 3 bytes, which is results a FAT of 60MB

Implementing Files (3)

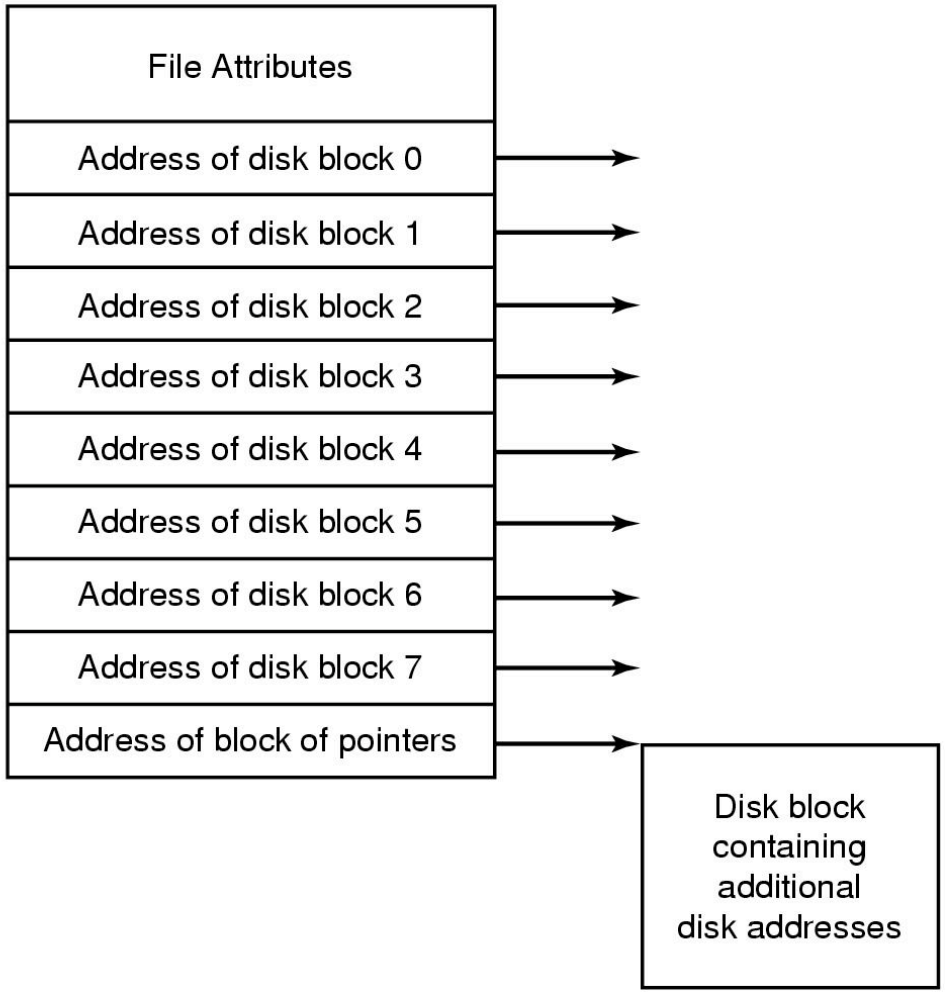


Linked list allocation using FAT in RAM

Indexing with i-nodes

- Each file has an associated fixed length record called an i-node
- i-node maintains all attributes of the file
- i-node also keeps addresses of fixed number of blocks
- Additional levels of indexing possible to accommodate larger files
  - last address reserved for pointer to another block of addresses
- Space required is much less than FAT
  - only i-nodes of open files need to be in memory
  - an array of i-node numbers, whose size is proportional to the max # of open files allowed by the system, not disk size
- Time required to access specific block can vary

Implementing Files (4)

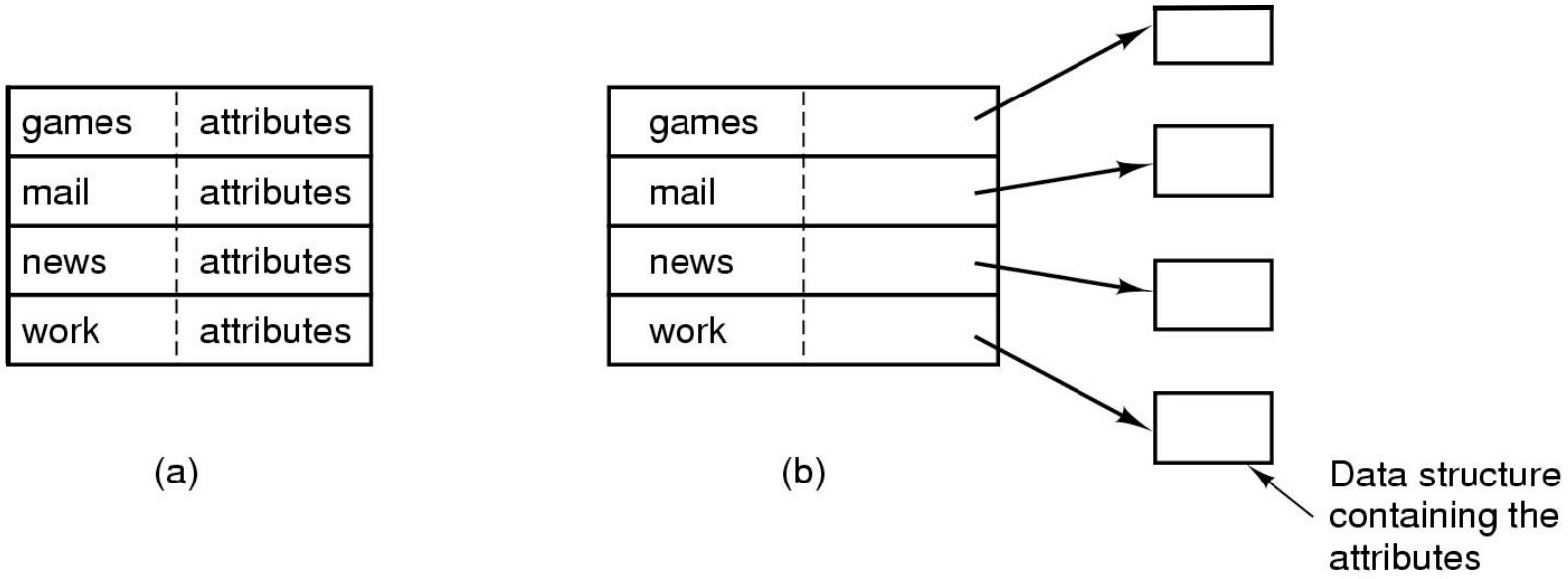


A sample i-node

Directories

- A directory entry provides the info needed to find the disk data blocks of a file
  - disk address of first block and size
  - address of first block
  - number of associated i-node
- File attributes can be stored in the directory entry (Windows) or in its i-node (Unix)
- File name and support of variable length and long file names (255 chars)

Implementing Directories (1)



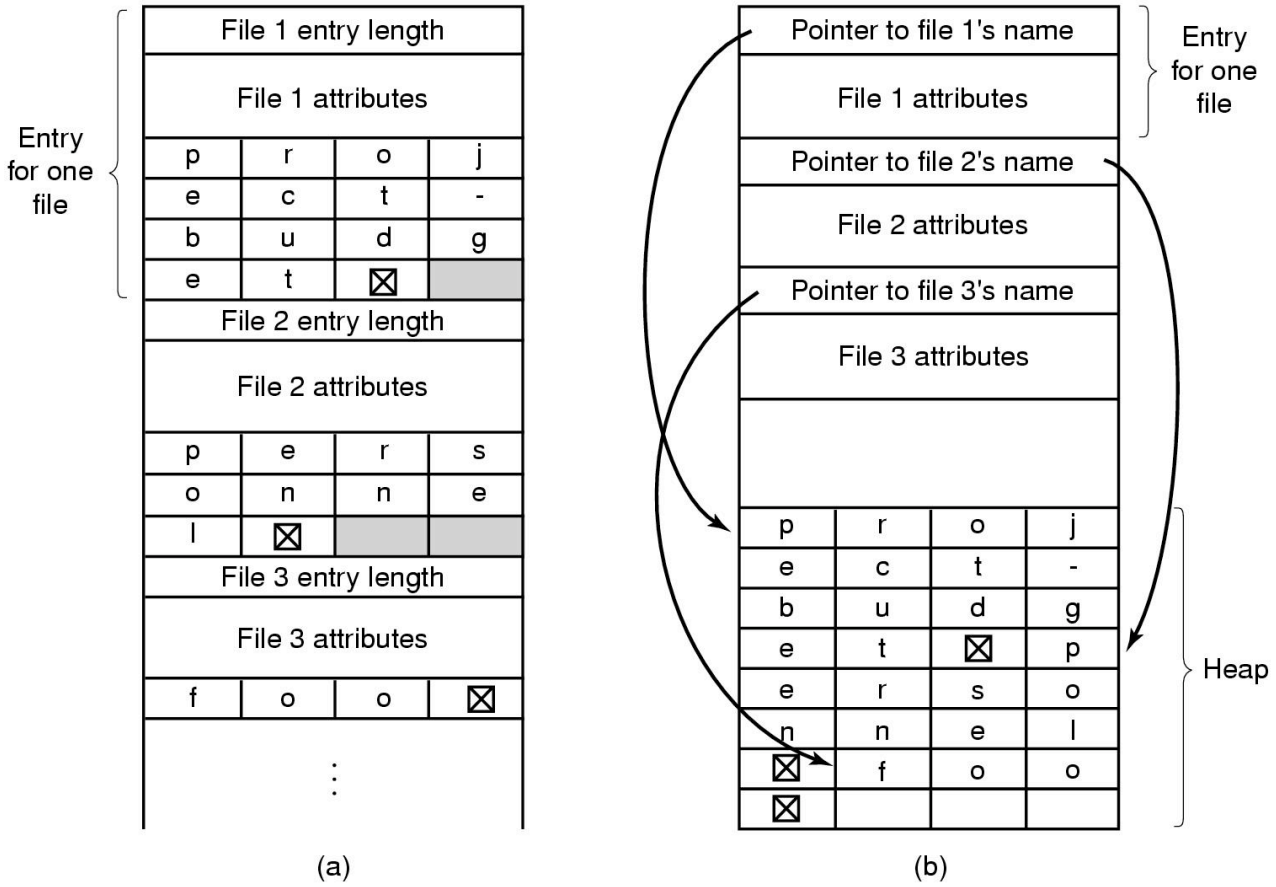
(a) A simple MS-DOS directory

fixed size entries

disk addresses and attributes in directory entry

(b) Unix directory in which each entry just refers to an i-node

Implementing Directories (2)

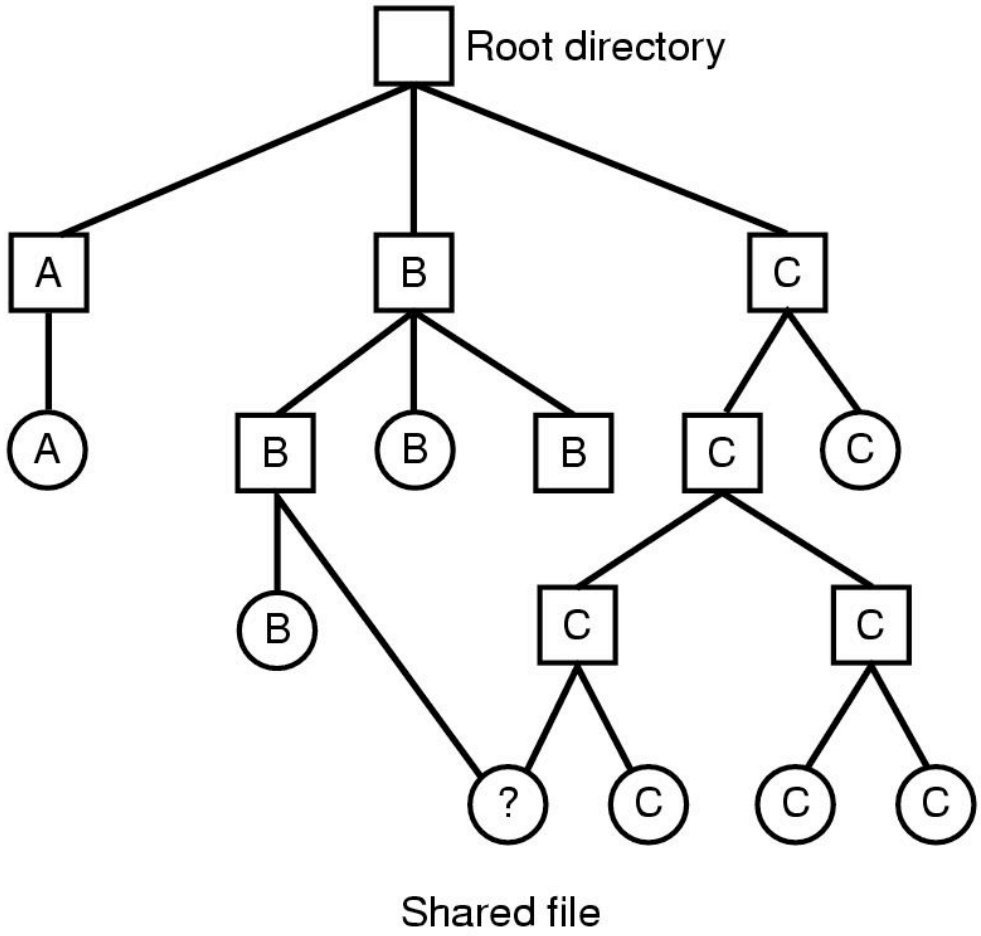


- Two ways of handling long file names in directory
  - (a) In-line
  - (b) In a heap

Links

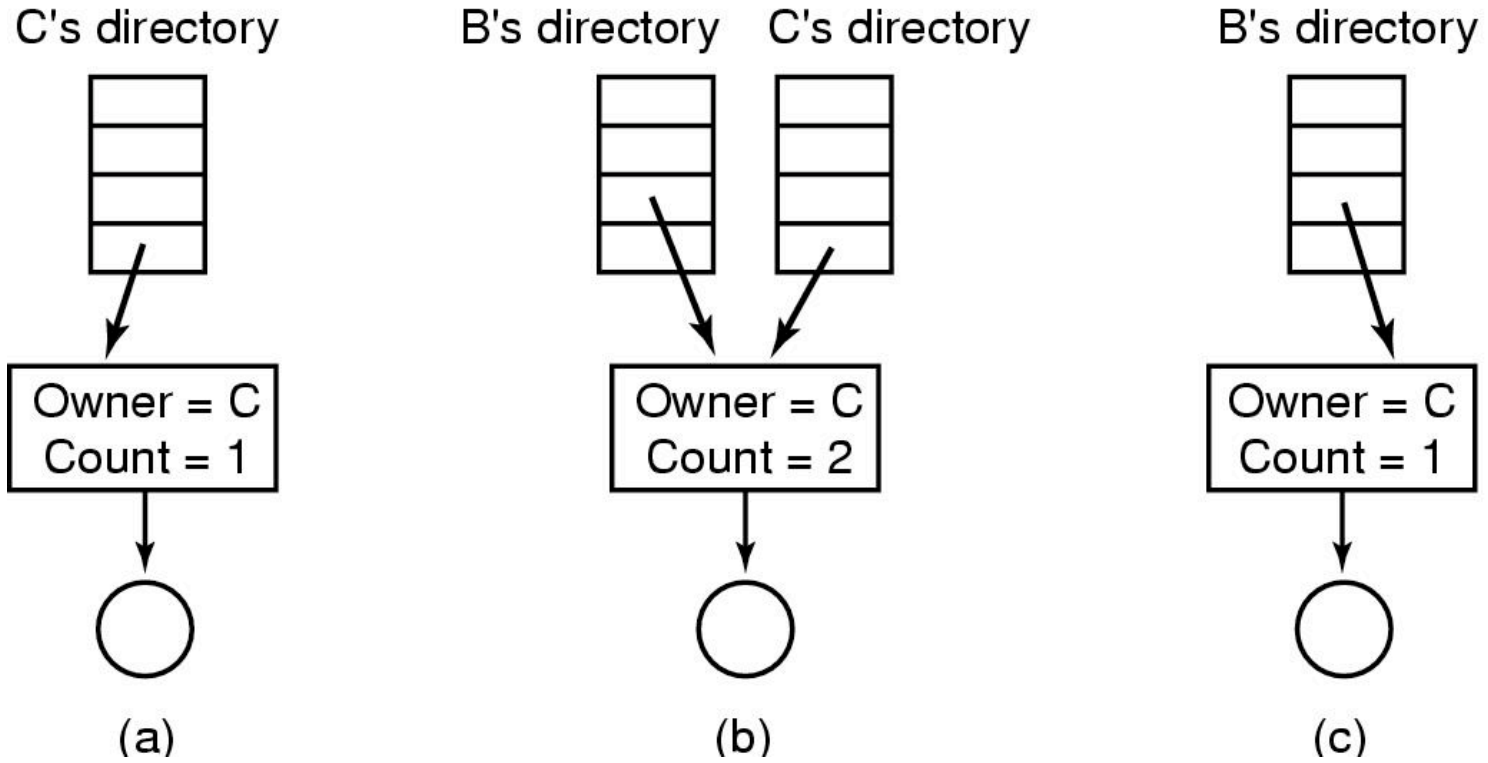
- Convenient solution to file sharing
- Hard link
  - pointer to i-node added to the directory entries
  - link counter in the i-node linked to incremented
  - i-node can only be deleted (data addresses cleared) if counter goes down to 0, why?
  - original owner can not free disk quota unless all hard links are deleted
- Soft link (symbolic)
  - new special file created of type LINK, which contains just the path name
  - new file also has its own i-node created

Shared Files - Link



File system containing a shared file

Shared Files – i-node



(a) Situation prior to linking

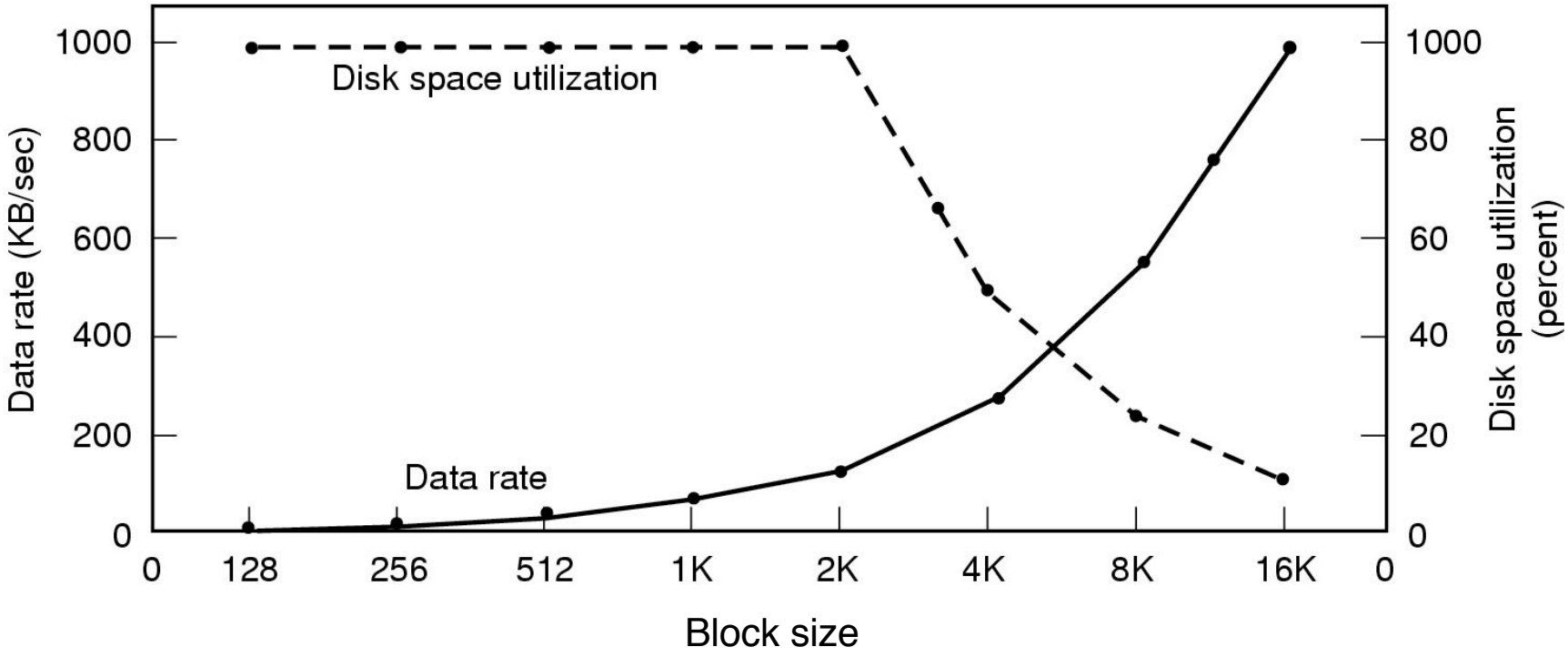
(b) After the link is created

(c)After the original owner removes the file

Disk Space Management

- Many of the same trade-offs or issues in memory management are also present in disk management
  - keeping track of free blocks
  - segmentation
  - caching and paging
- Block size, how big?
  - smaller block size leads to less waste in a block
  - larger block size leads to better data transfer rate (block assess time is dominated by seek time and rotational delay)
  - experiments lead to estimate of median file size at 2KB
  - Unix systems commonly keep 1KB, because its choice was made a long time ago
  - MS-Dos has a 2^{16} limit on number of blocks, making block size proportional to disk size

Block size

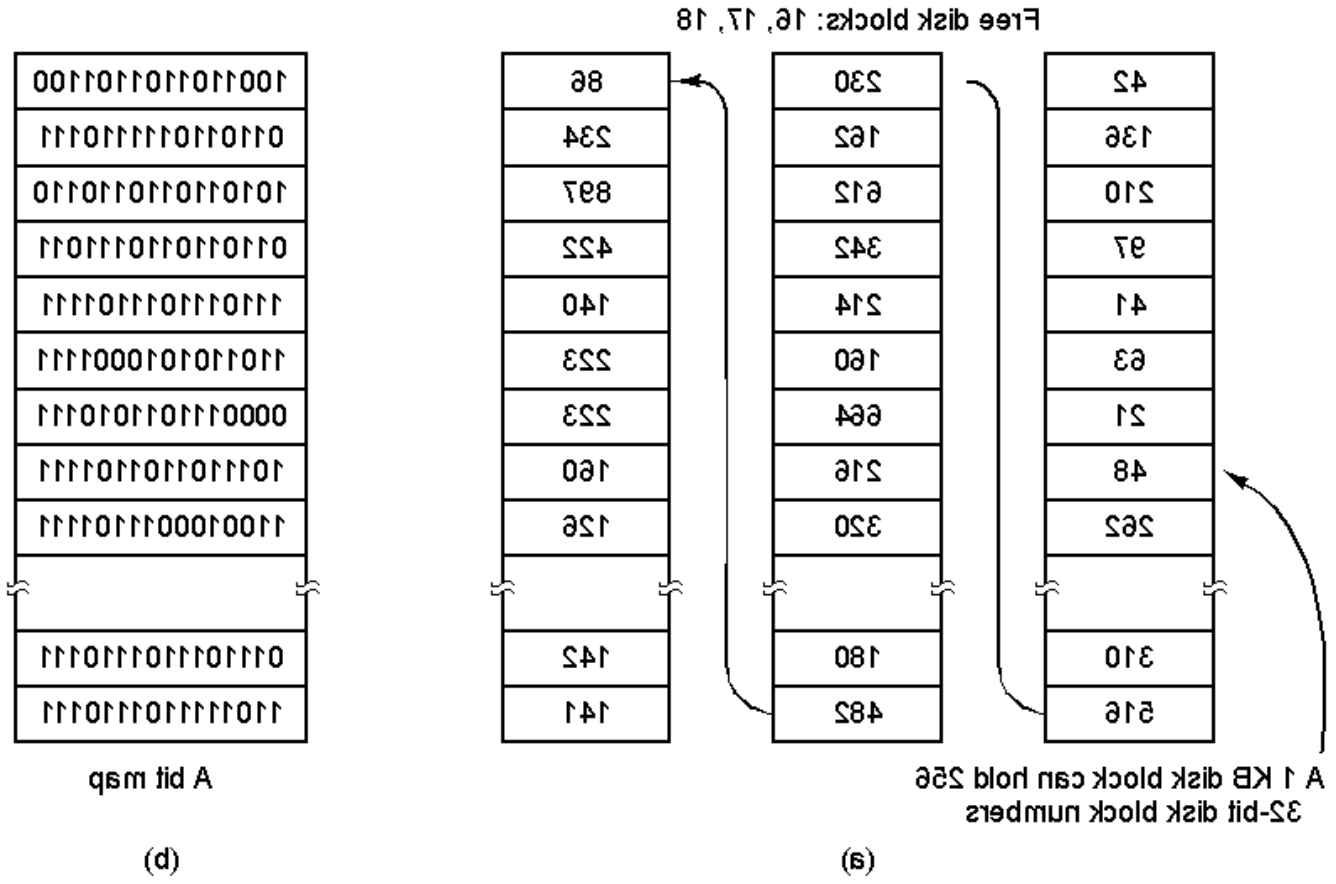


- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency
- All files 2KB

Managing Free Disk Space

- How to keep track of free blocks?
- Linked List Method
  - Maintained as a list of blocks containing addresses of free blocks
  - Each block address is 32 bits or 4 Bytes
  - A 1KB block used can hold addresses of 255 free blocks and an address of the next block in the list for free space management
  - Note: This list is stored in free blocks themselves
- Bitmap method:
  - Keep an array of bits, one per block indicating whether that block is free or not
  - A 16-GB disk has 16 million 1 KB blocks
  - Storing the bitmap requires 16 million bits, or 2048 blocks
- How do the two methods compare in terms of space requirements?

Free Blocks



(a) Storing the free list on a linked list

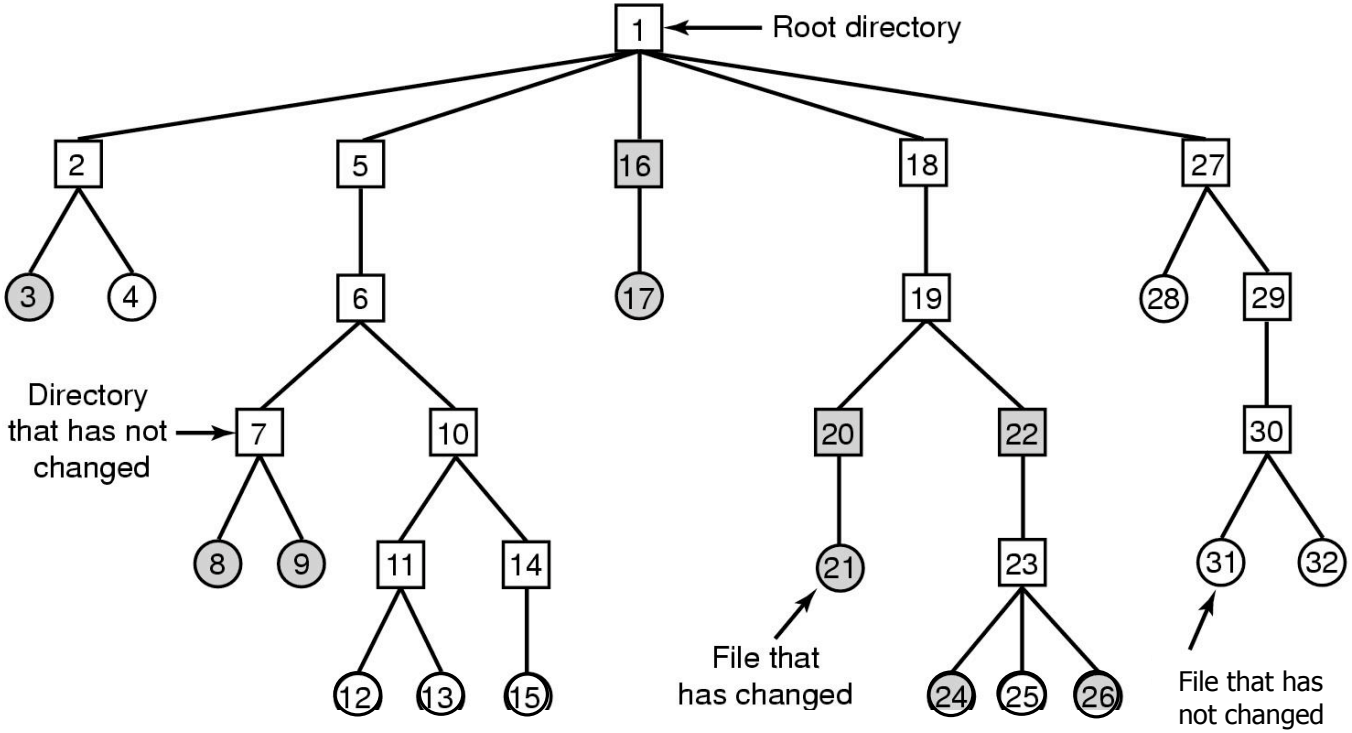
(b) A bit map

System Reliability

- Destruction of a file system is the worst disaster known to a computer

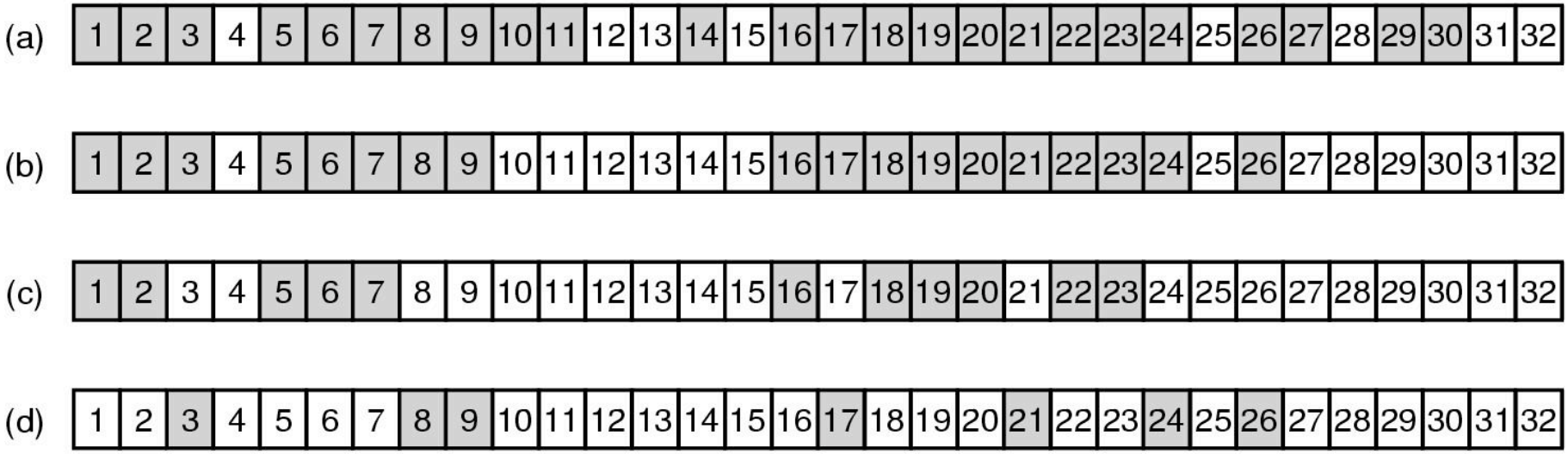
- Backups
  - recover from (natural) disaster
  - recover from stupidity
- Different dumps
  - full dump
  - incremental dump
  - physical dump
  - logical dump

Logical Dump



- A file system to be dumped
  - squares are directories, circles are files
  - shaded items, modified since last dump
  - each directory & file labeled by i-node number

Logical Dump Bitmaps



(a) all directories and modified files

(b) all modified files, their parent directories and modified directories

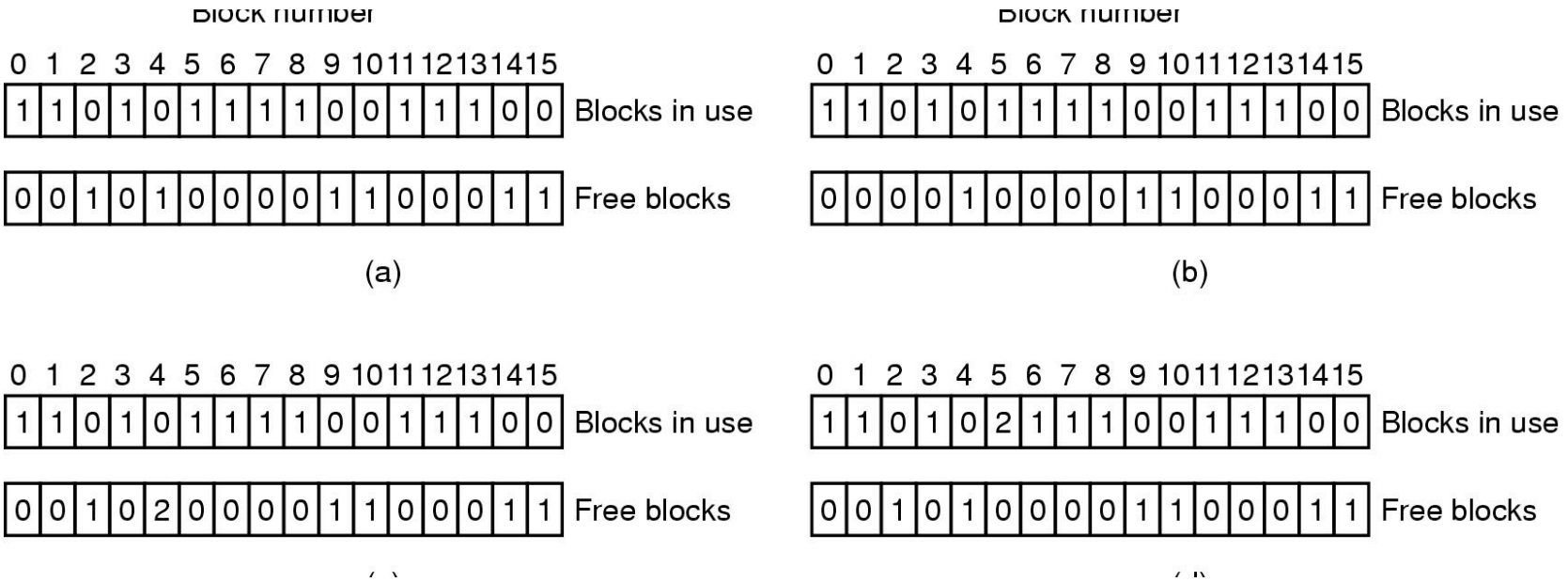
(c) all directories to be dumped

(d) all files to be dumped

File System Consistency

- Disk I/O is buffered
- There may be a crash before the modified blocks in memory are written back to the disk
- File system consistency must be checked after a crash
  - fsck
  - scandisk
  - block consistency – a block is either used (listed in i-node) by only one file or is free (listed in free list)
  - file consistency – a file has the same number of directory entries (i.e. the same number of hard links) as the link count in its i-node

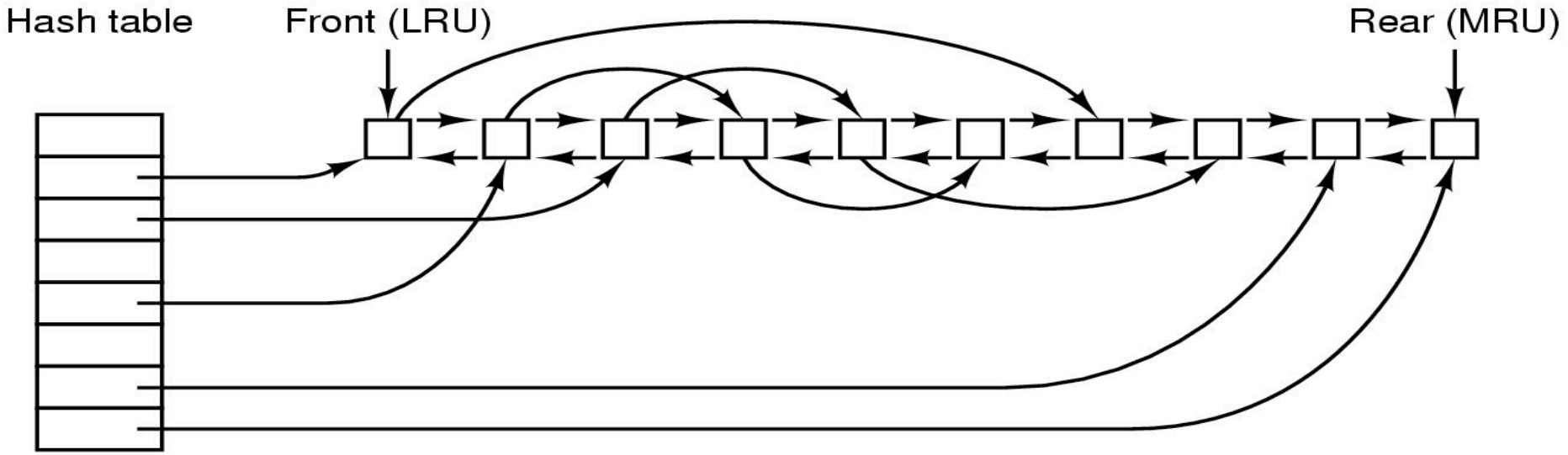
Consistency Check



- File system states
  - (a) consistent
  - (b) missing block
  - (c) duplicate block in free list
  - (d) duplicate data block

Caching

- Since disk I/O is slow, a part of memory is reserved to hold disk blocks that are likely to be accessed repeatedly
- Basic data structure:
  - Maintain a doubly linked list of blocks
  - Use hashing so that a block can be quickly accessed from its address
  - LRU implementation possible: every time a block is accessed move it to the back of the list



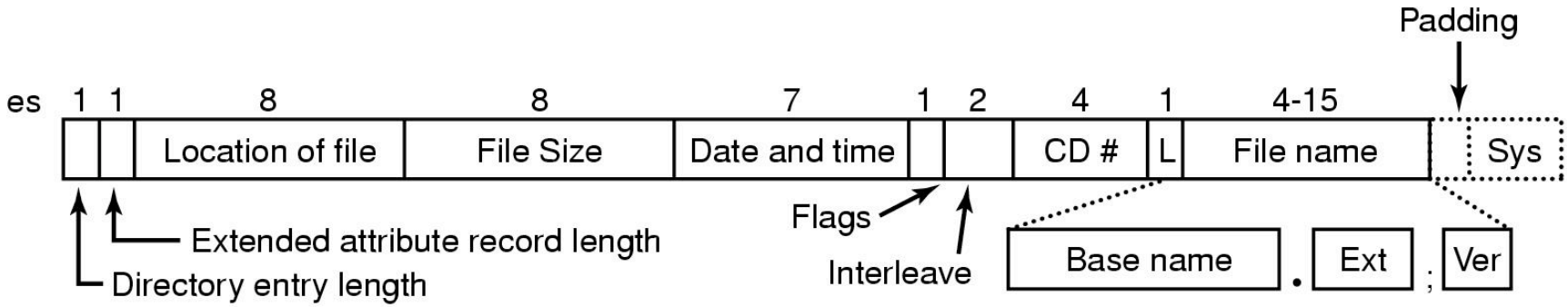
Writing Strategies

- Problem: If there is a crash, modifications to blocks in cache will be lost
  - disastrous for blocks containing i-nodes or FAT
  - LRU guarantees loss of the exactly wrong blocks
- MS-DOS: write-through cache
  - every time a block is modified, a request to write it to disk is issued
  - loop with putchar() generates busy disk I/O
  - reads can still be fast
- Classify blocks and treat them separately (Unix)
  1. i-node blocks
  2. indirect blocks with addresses of file blocks
  3. free space management blocks
  4. data blocks of files
- Write critical blocks (types 1, 2, 3) on disk immediately upon modification
- Transfer data blocks in LRU

The CD-ROM

- ISO 9660
  - adopted in 1988
  - block size 2048
  - begins with 16 blocks whose functions are undefined
    - can be used as boot blocks
  - primary volume descriptor
  - root and directories
- Directory
  - filename 8+3
  - depth of nesting limited to 8

CD-ROM Directory Entry



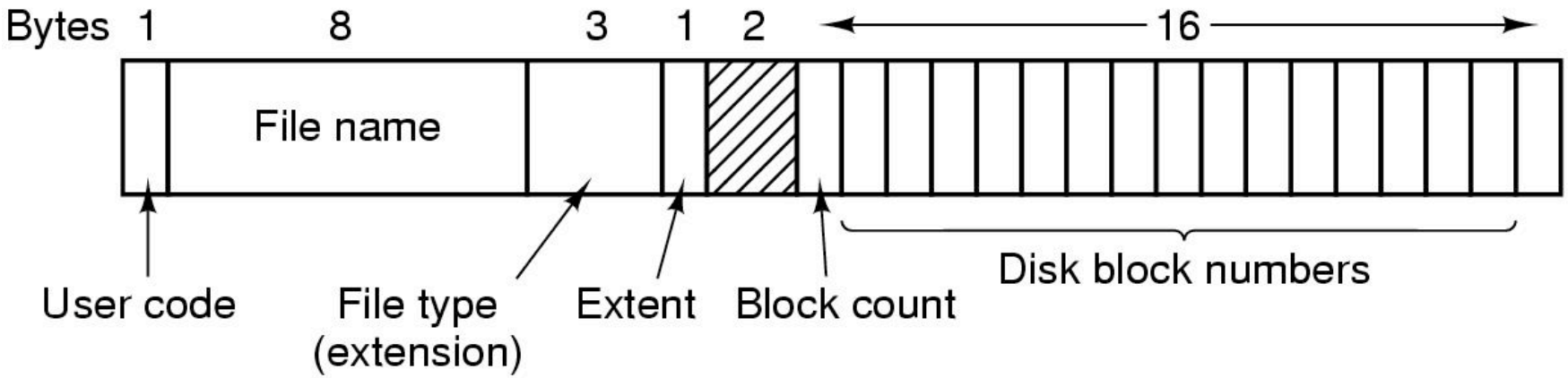
The system use field is undefined, but used by many OS to extend ISO 9660

Unix – Rock Ridge

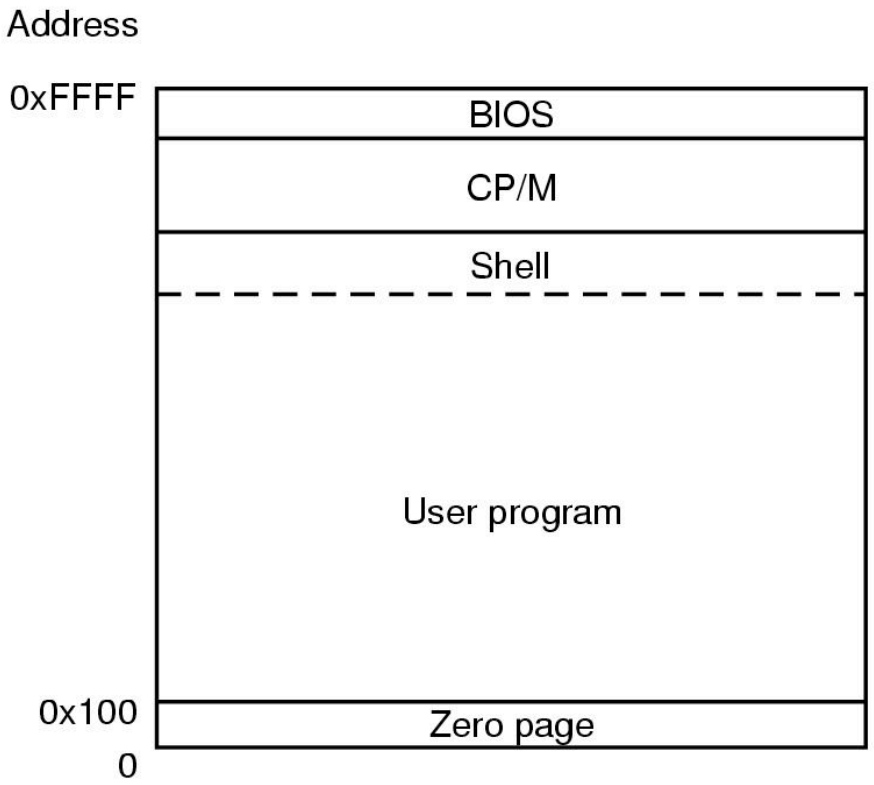
Windows – Joliet

CP/M

- Dominated the microcomputer world in the early 80s
  - Intel 8 bit 8080 CPUs
  - By Gary Kildall, Digital Research
- Ran on 16KB of RAM, well. Entire OS under 4KB
- Direct ancestor of MS-DOS
- Only one directory
- Block size 1KB



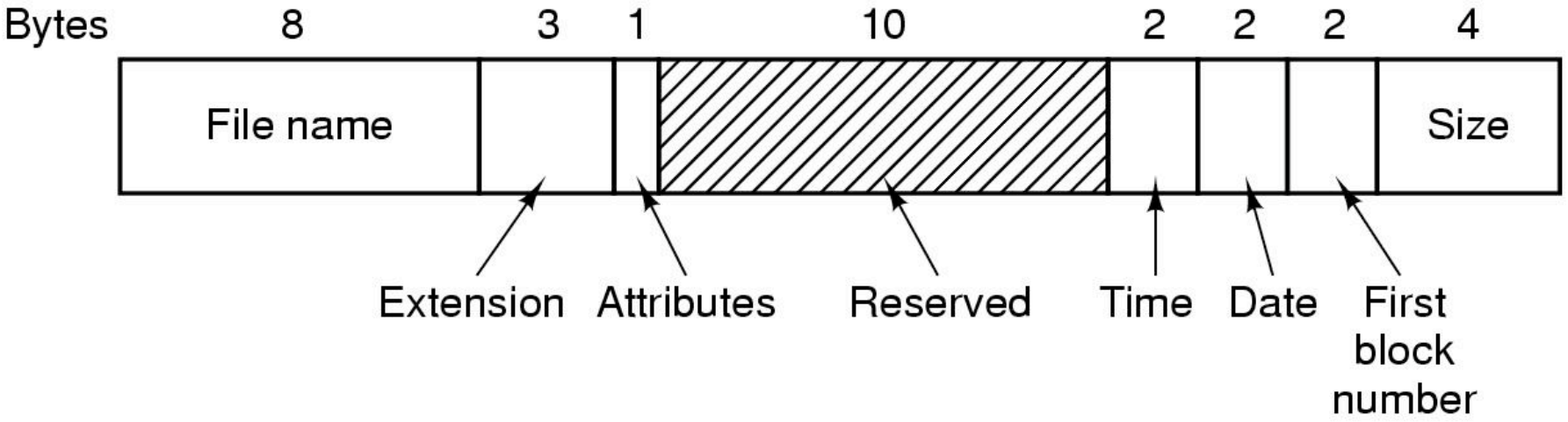
The CP/M File System



Memory layout of CP/M

MS-DOS

- Originally invented as a method for storing data on floppy disks, first version has single directory
- Later used by MS-DOS with supports for a hierarchical directory structure, and fixed disks as well as floppy disks
  - directory entries are fixed 32 bytes
  - different versions support FAT-12, FAT-16 or FAT-32



Block Sizes and FAT Sizes

Disk Size	FAT-12	FAT-16	FAT-32
FAT entry	12 bits	16 bits	28 bits
Max # of blocks	4086 $\sim 2^{12}$	65526 $\sim 2^{16}$	268435456 $\sim 2^{28}$
Block size	0.5KB to 4KB	2KB to 32KB	4KB to 32KB
FAT size	8KB	128KB	1GB

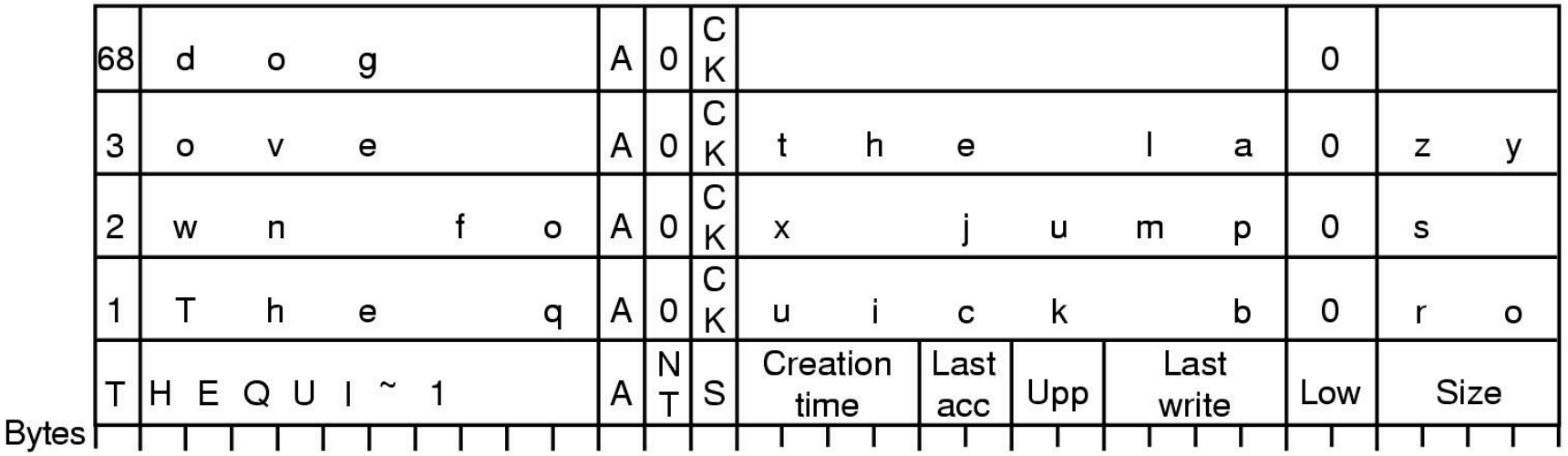
Block Sizes and Max Partition Sizes

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

- Internally sector sizes are 512 bytes
- Empty boxes represent invalid combinations

Windows 95 and 98

- Long file names supported by Win95 V2 and Win98
- In order to be backwards compatible to MS-DOS, decisions were made to assign internal 8 char DOS file names in addition



An example of how a long name is stored in Windows 98

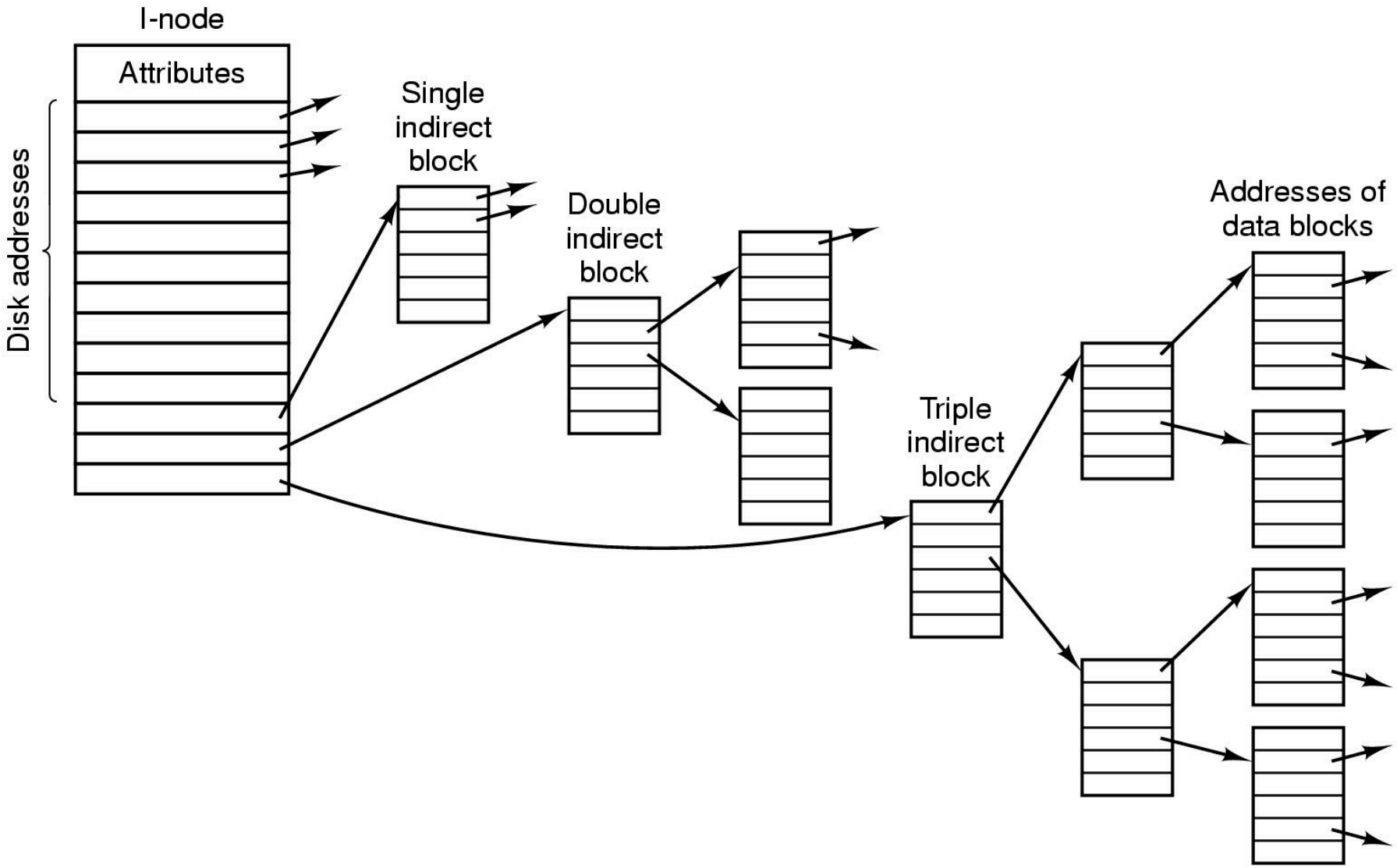
The Classic Unix File System

- Each disk partition has 4 Regions
  - block 0: boot block
  - block 1: super block. Contains the size of the disk and the boundaries of the other regions
  - i-nodes: list of i-nodes, each is a 64-byte structure
  - free storage blocks for the contents of files.
- Each i-node contains owner, protection bits, size, directory/file and 13 disk addresses.
- The first 10 of these addresses point directly at the first 10 blocks of a file. If a file is larger than 10 blocks (5,120 bytes), the 11th points at a block for secondary indexing – single indirect block

Classic Unix File System Cont.

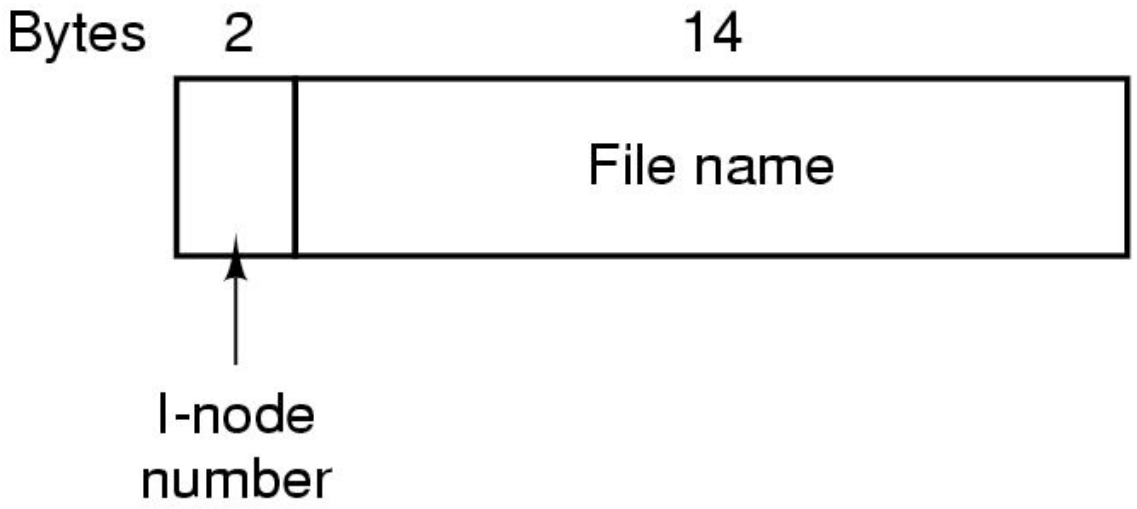
- Second-level index block contains the addresses of the next 128 blocks of the file (70,656 bytes)
- Two levels of indirection:12th entry (double indirect block) points at up to 128 blocks, each pointing to 128 blocks of the file (8,459,264 bytes)
- Three levels of indirection: 13th address (triple indirect block) is for a three layered indexing of 1,082,201,087 bytes.
- A directory is accessed exactly as an ordinary file.
  It contains 16 byte entries consisting of a 14-byte name and an i-number (index or ID of an i-node). The root of the file system hierarchy is at a known i-number (2).

Unix i-node


I-node
Attributes
Disk addresses
Single indirect block
Double indirect block
Triple indirect block
Addresses of data blocks

Directory Entry: Unix V7

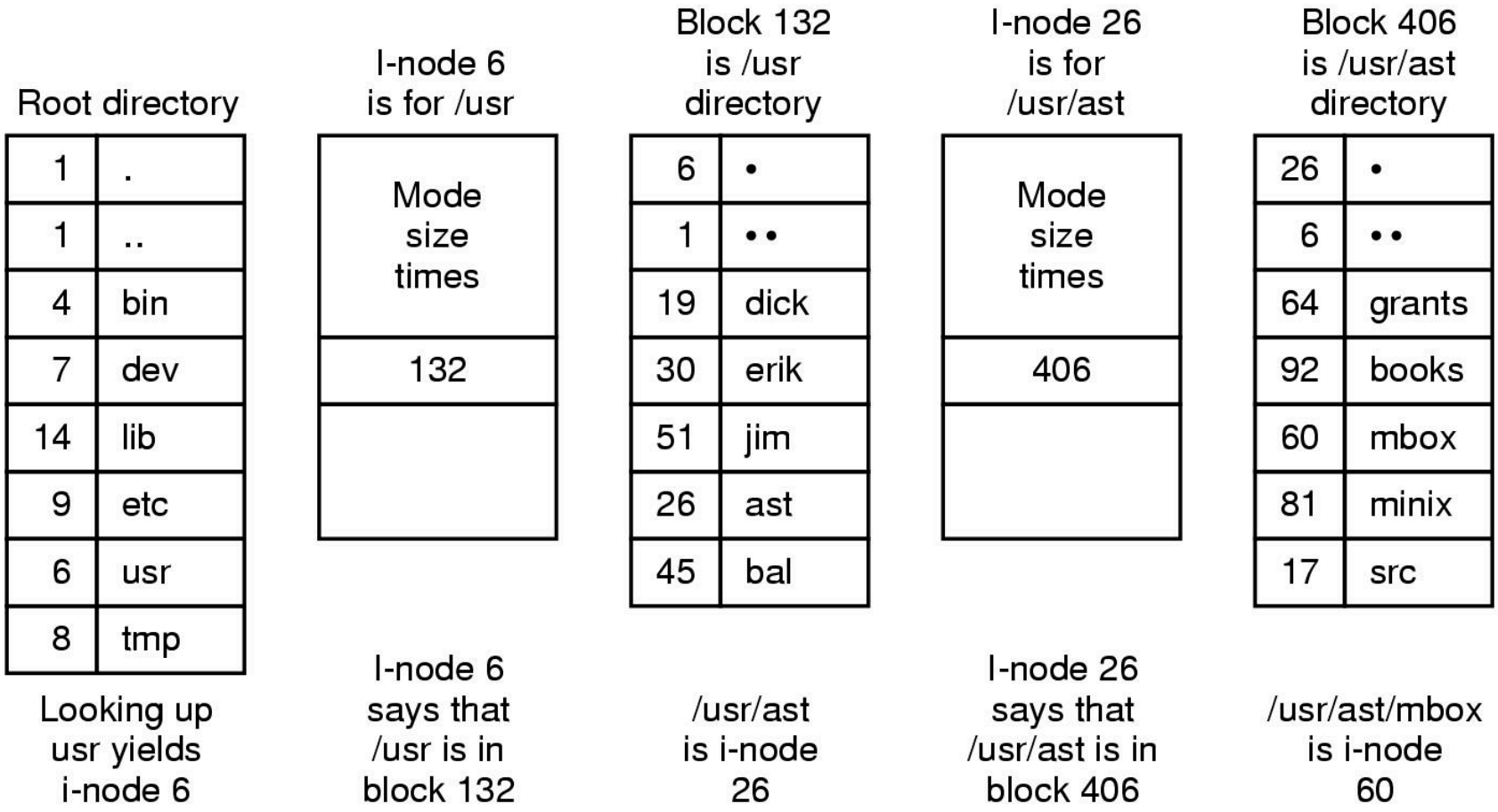
- Each entry has file name and an I-node number
- I-node has all the attributes
- Restriction: File name size is bounded (14 chars)



Traversing Directory Structure

- Suppose we want to read the file /usr/ast/mbox
- Location of a i-node, given i-number, can be computed
- i-number of the root directory known, say, 2
- Read in the i-node 2 from the disk into memory
- Find out the location of the root directory file on disk, and read the directory block in memory
  - If directory spans multiple blocks, then read blocks until usr found
- Find out the i-number of directory usr, which is 6
- Read in the i-node 6 from disk
- Find out the location of the directory file usr on disk, and read in the block containing this directory
- Find out the i-number of directory ast (26), and repeat

The UNIX V7 Directory Lookup



The steps in looking up */usr/ast/mbox*