File systems

Long-term Information Storage

- Must store large amounts of data
- Information stored must survive the termination of the process using it
- Multiple processes must be able to access the information concurrently

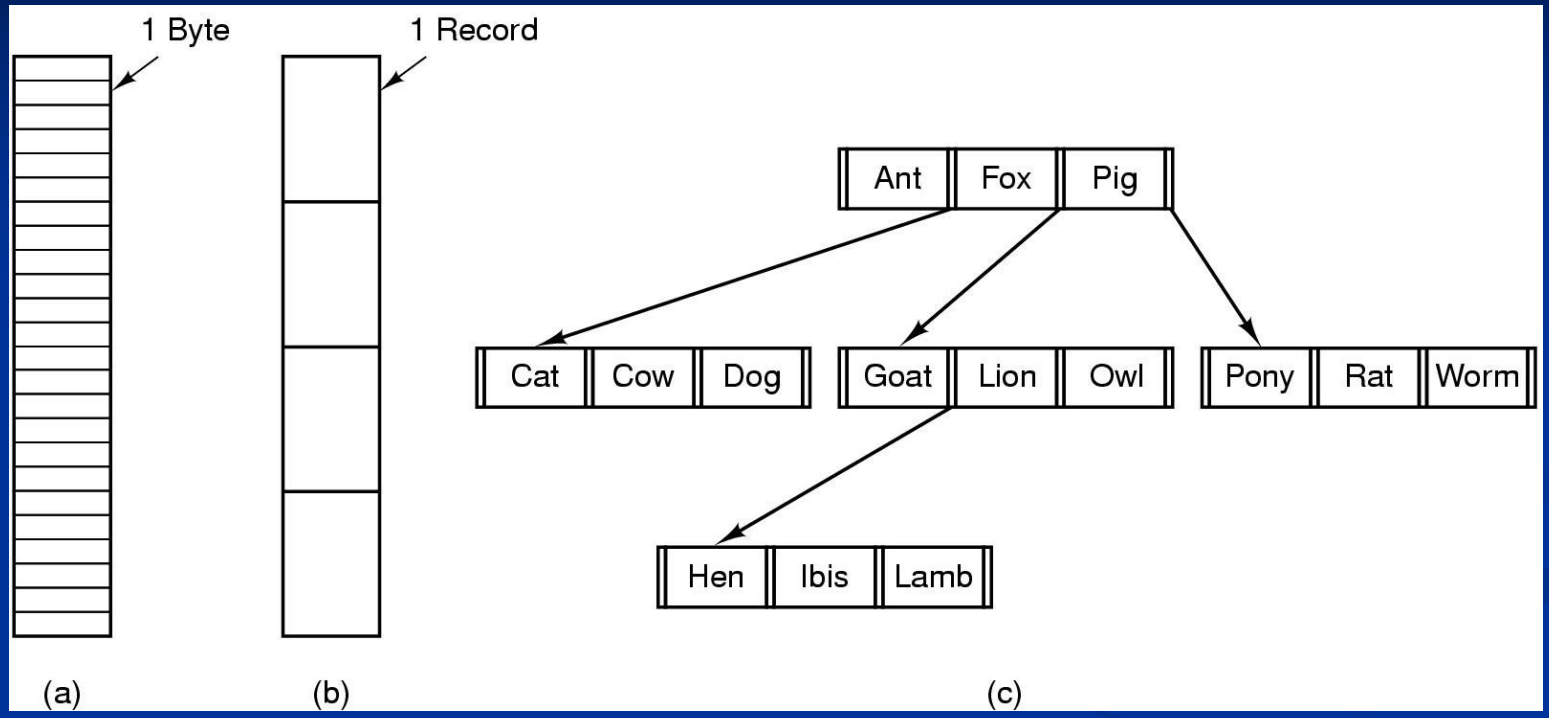
Typical file extentions

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

These are just conventions.

Nothing forces you to follow this naming.

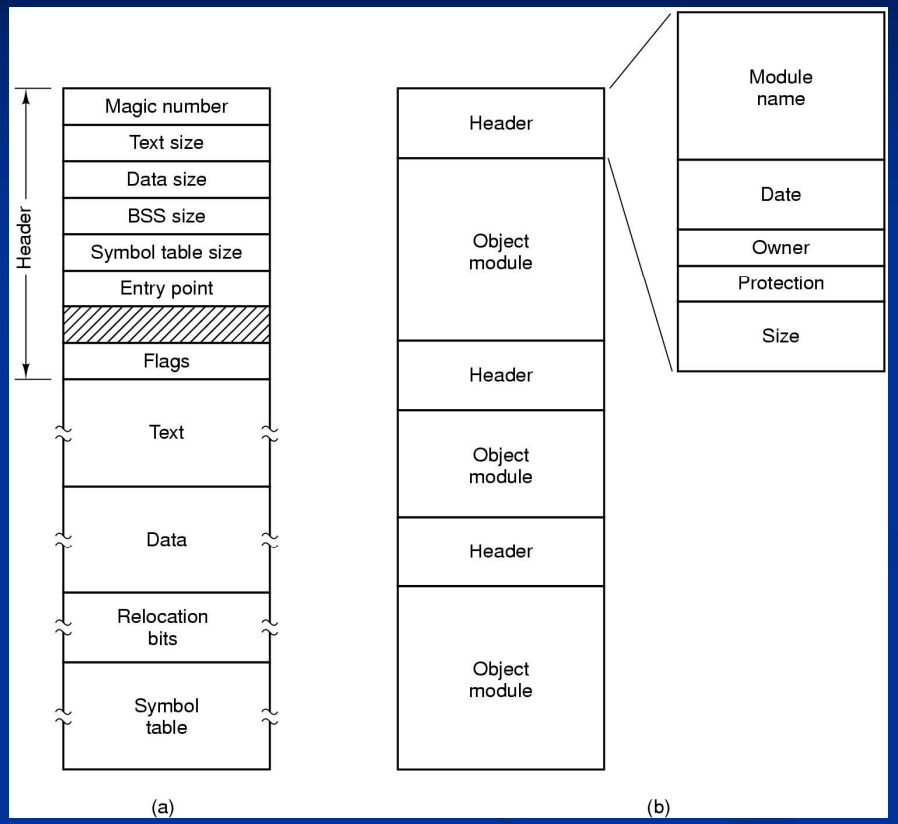
File Structure



- Three kinds of files
  - byte sequence
  - record sequence
  - tree

File Types and 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 database systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker



(a) An executable file
(b) An archive

File Attributes (or metadata)

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

File Operations

- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get attributes
- Set Attributes
- Rename

An Example Program Using File System Calls (1/2)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
        int in_fd, out_fd, rd_count, wt_count;
        char buffer[BUF_SIZE];

        if (argc != 3) exit(1);         /* syntax error if argc is not 3 */
```

An Example Program Using File System Calls (2/2)

```
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);     /* open the source file */
if (in_fd < 0) exit(2);              /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
if (out_fd < 0) exit(3);             /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;            /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
        if (wt_count <= 0) exit(4);  /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                   /* no error on last read */
        exit(0);
else
        exit(5);                     /* error on last read */
}
```

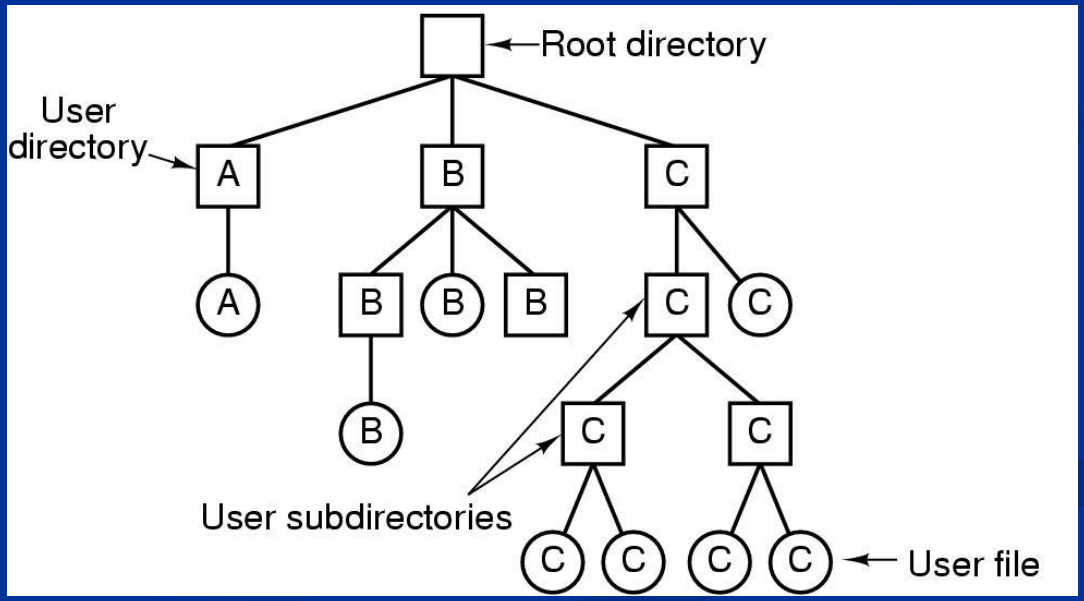
Directories

Directory

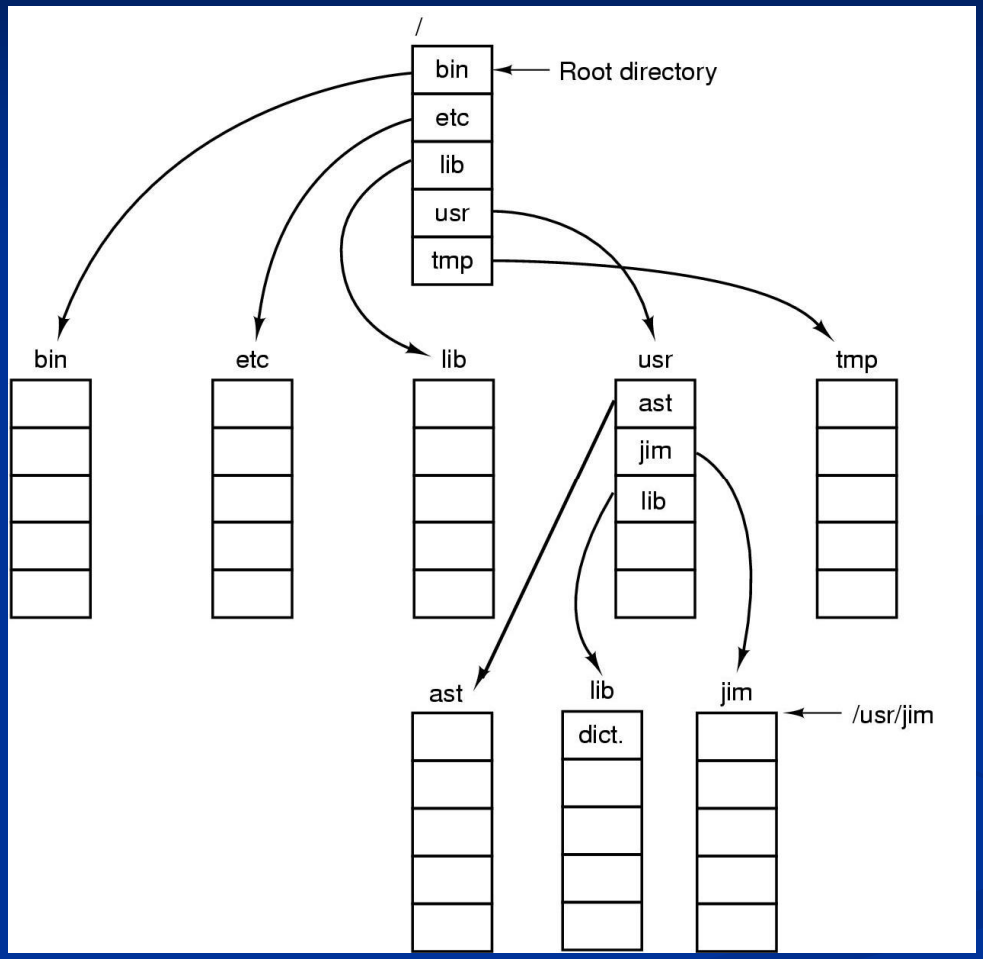
= File, that contains information about other files

- Only OS is allowed directly to access these files
  - All changes through system calls only

- Root directory, home directories
- Processes can create subdirectories
- Fixed location for root directory on disk

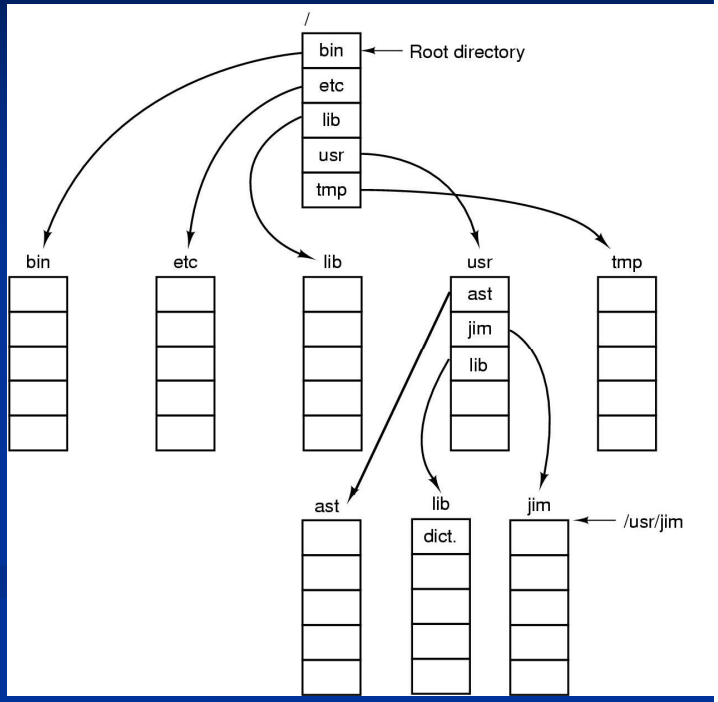


Path Names



- A UNIX directory tree

Path names

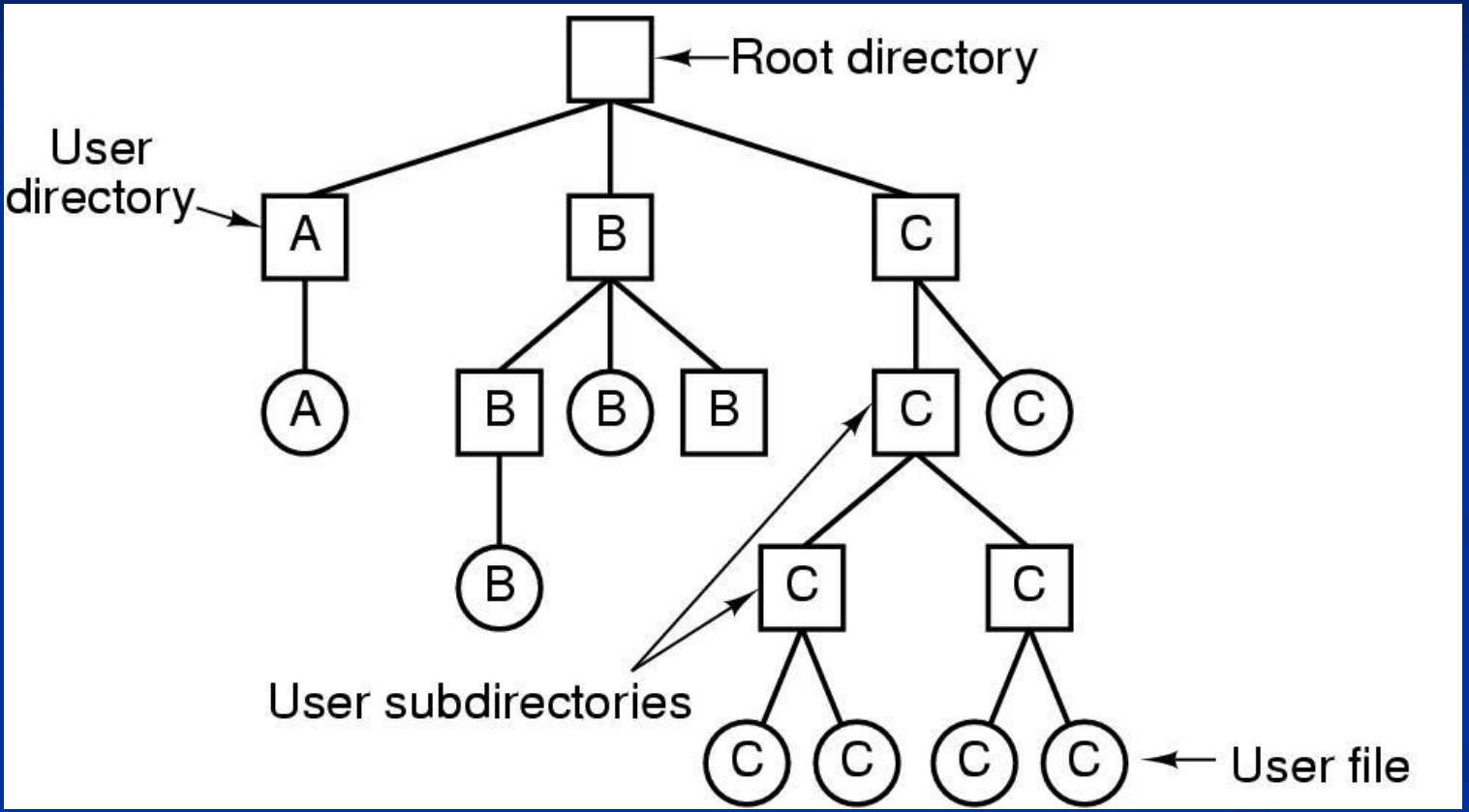


- Process has a working directory
  - Current working dir in proc. table
  - At start it is normally user's home dir
  - User can change it
- Files identified with combining path and file name
  - Absolute path name: /dirname/dirname/filename
  - Relative path name: dirname/filename or just filename
  - Directory names: . and ..

Directory Operations

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

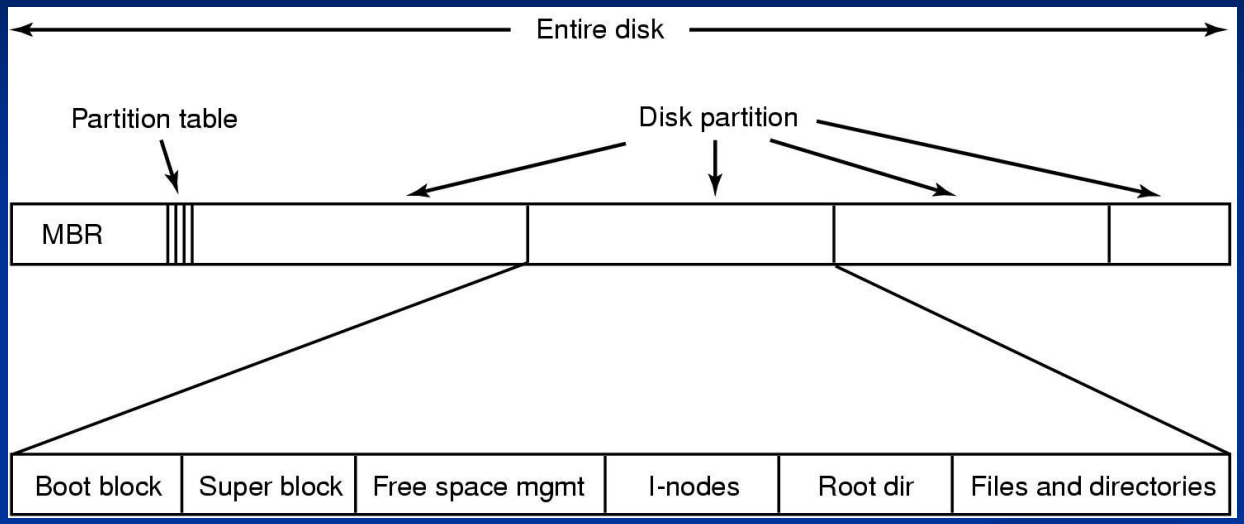
Hierarchical Directory Systems



- A hierarchical directory system

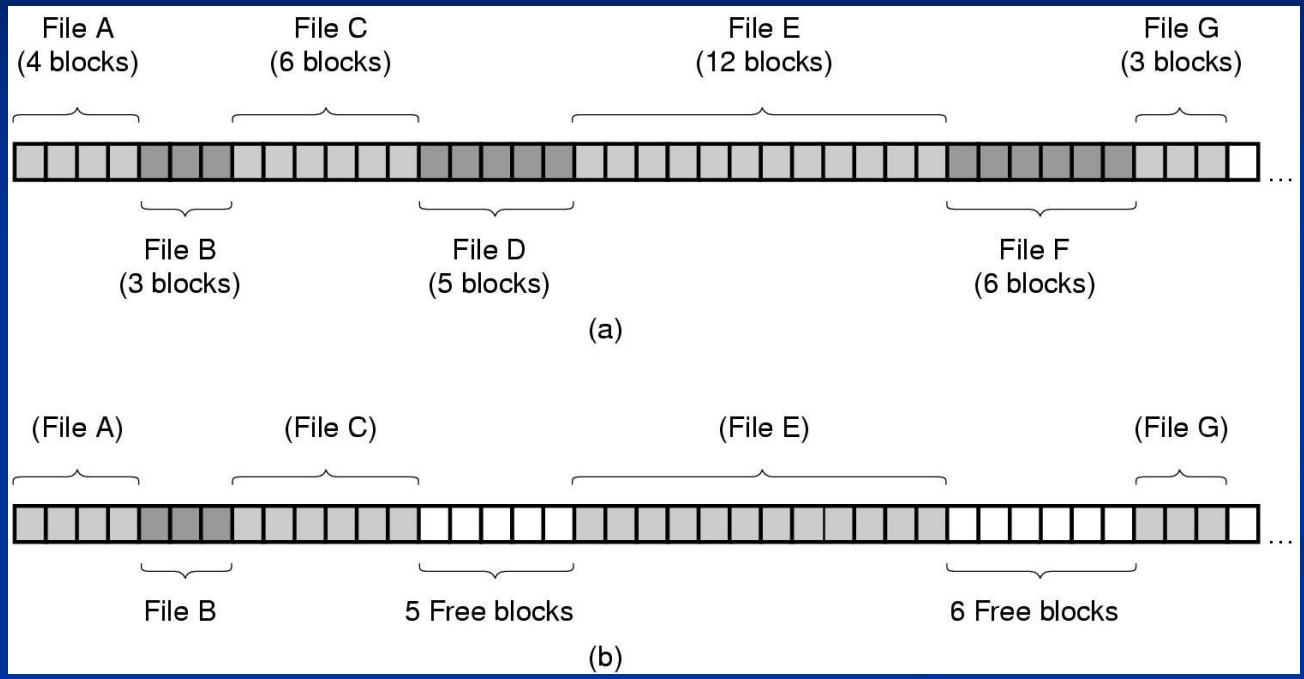
File system implementation

File System Layout



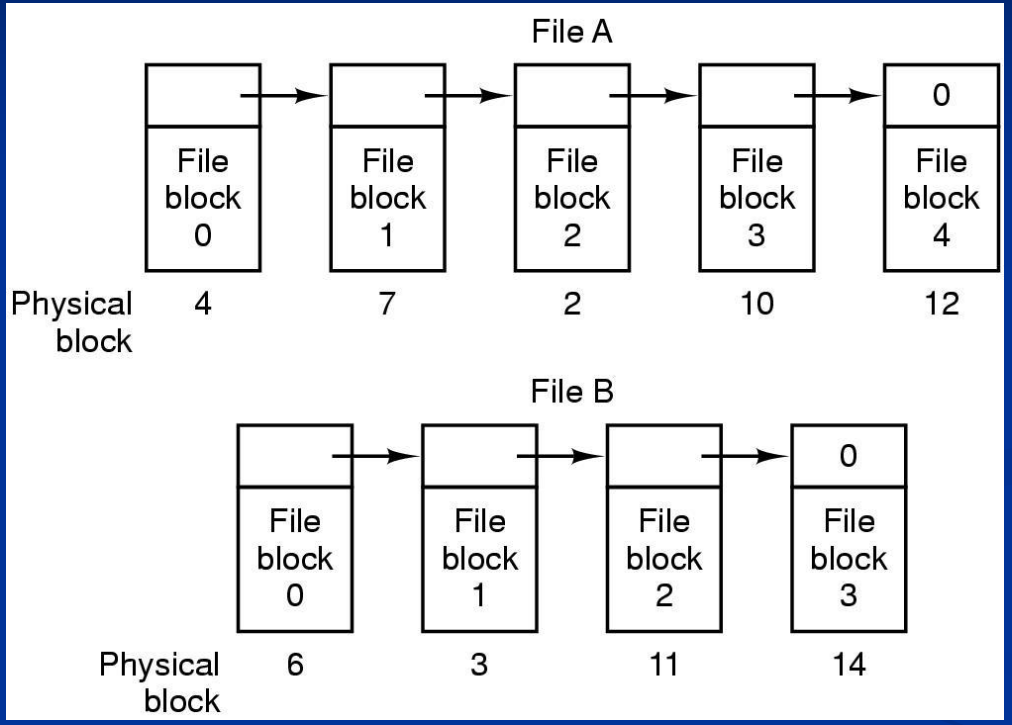
- Master Boot Record (MBR) in fixed position
- Disk partition information in partition table (fixed loc)
- Each partition usually has boot block
- Everything else on partition depends on the file system
  - Superblock contains key parameters: id, block struct

Allocating blocks for files: contiguous



- File location info: start block, number of blocks
- Good read performance: only one seek for the whole file
- Difficult (or impossible) to increment file
- Fragmentation: Holes from deleted files (see (b))
- Used on CD-ROMs

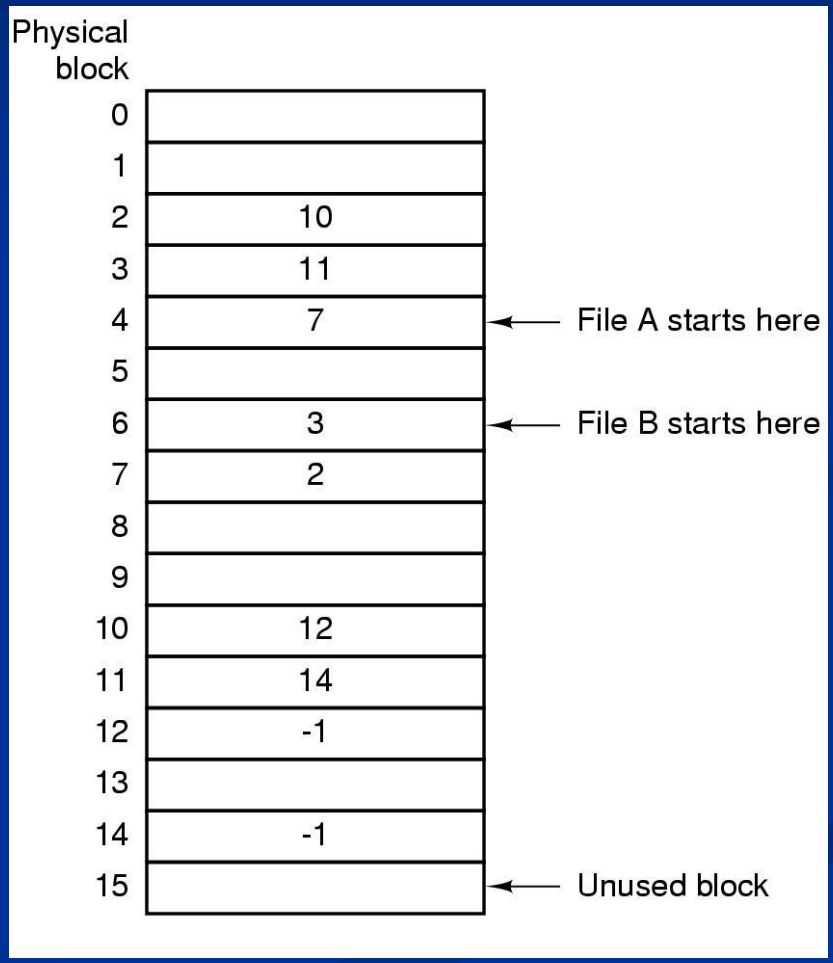
Allocating blocks for files: linked list of disk blocks



- File location info: Start block
- No fragmentation: Each block has a pointer to next block
- Reading performance: sequential fine, random access difficult

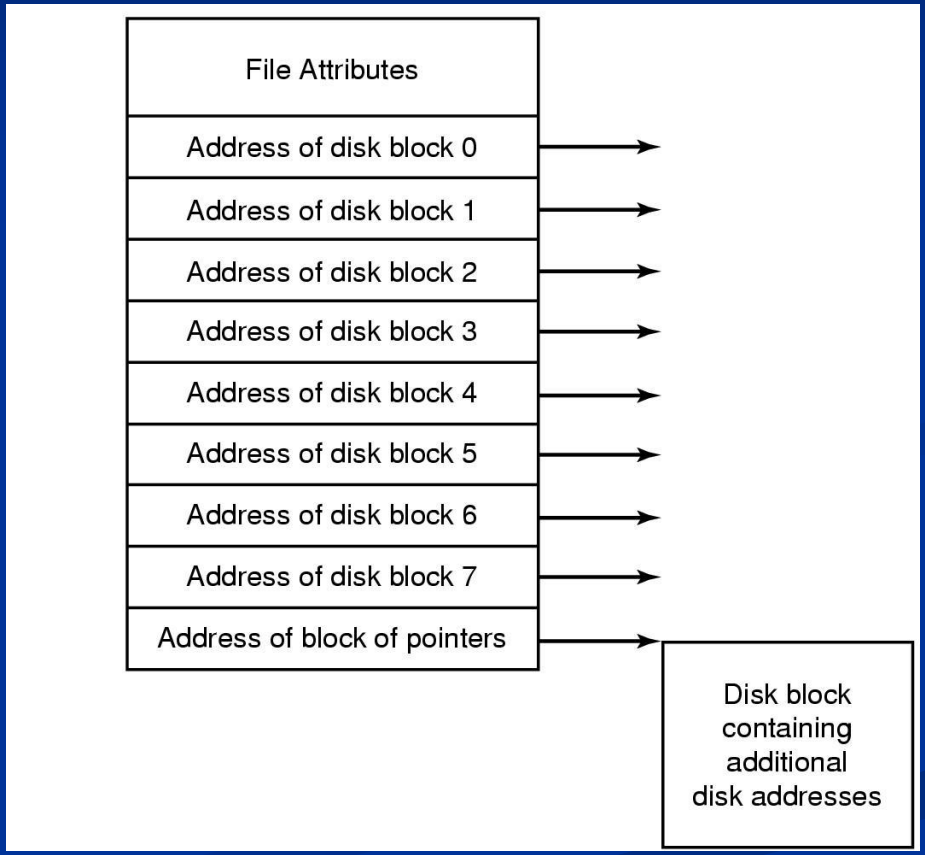
Implementing Files: File Allocation Table (FAT)

- Link information in FAT
- Read performance:
  - Blocks of file in multiple locations, more seeks
  - Random access possible
- On disk when power-down
- In memory when running
  - Does not scale for large disks
- To improve read performance:
  - Consolidation, defragmentation

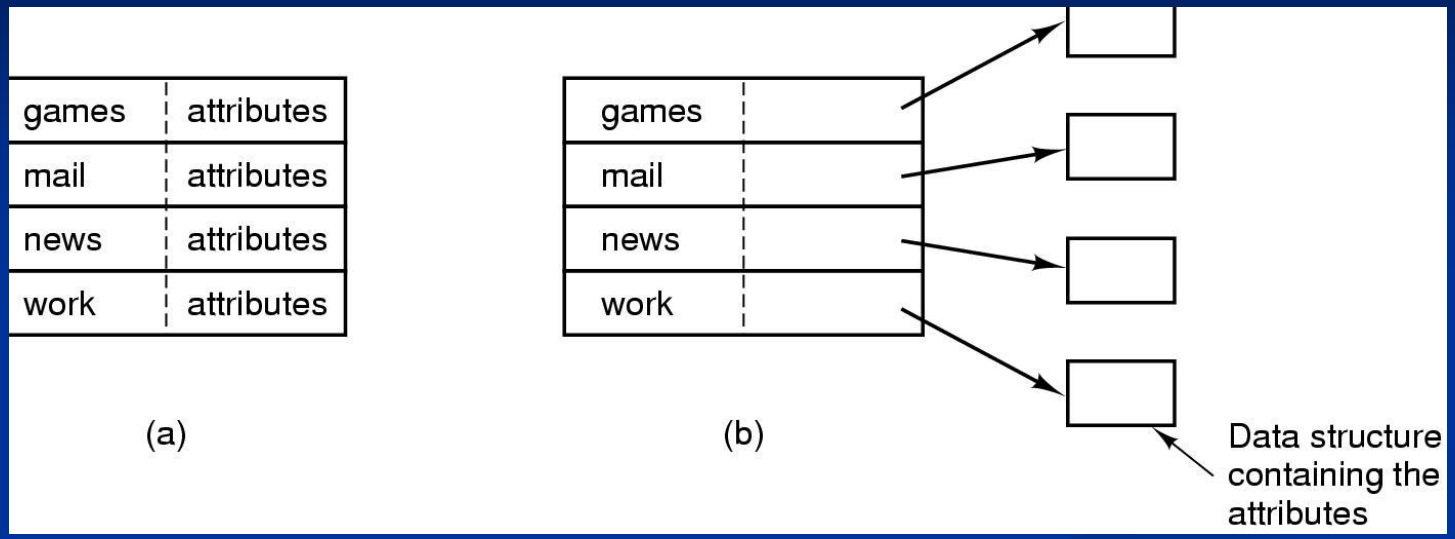


Implementing Files: i-node

- Special data structure for each file separately
- In memory only when this file accessed
  - Independent of disk size
- I-node has fixed number of block locations
  - Reserve last for address to block which contains more block addresses

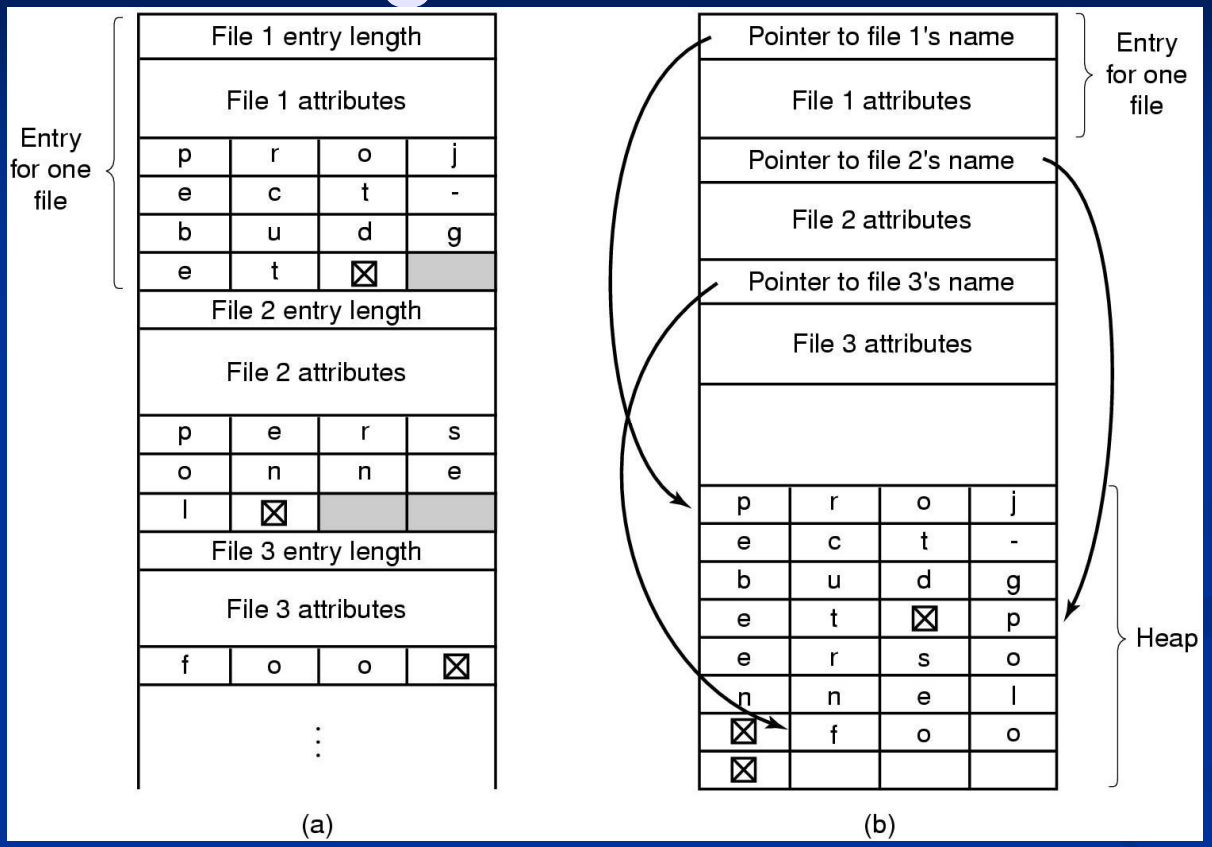


Directory structure



- Directory entry contains
  - File name
  - File location information: block number, i-node number
  - File attributes (unless stored in i-node)
- Specific structure depends on the file system

Long file names



- Two ways of handling long file names in directory
  - (a) In-line – entry size varies
  - (b) In a heap – fixed enry and names in heap

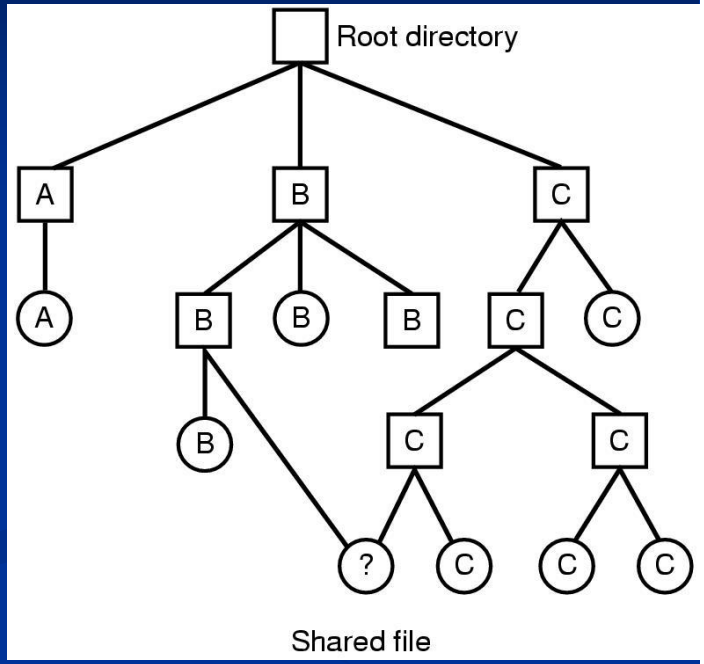
Shared files

Shared files: rights to access

- Usage rights collected to file attributes
- User grouping in (UNIX) u,g,o
- Rights (UNIX) r,w,x, -
- Directory rights (UNIX)
  - r right to list the directory content (read the directory element)
  - w right to remove a file from directory (write the directory element)
  - x right to use the directory name as part of the path name
- Some systems use different methods like access control / capability lists
- Access right is checked by OS only at the opening of file
  - Must have right to all parts of the path name

Shared file

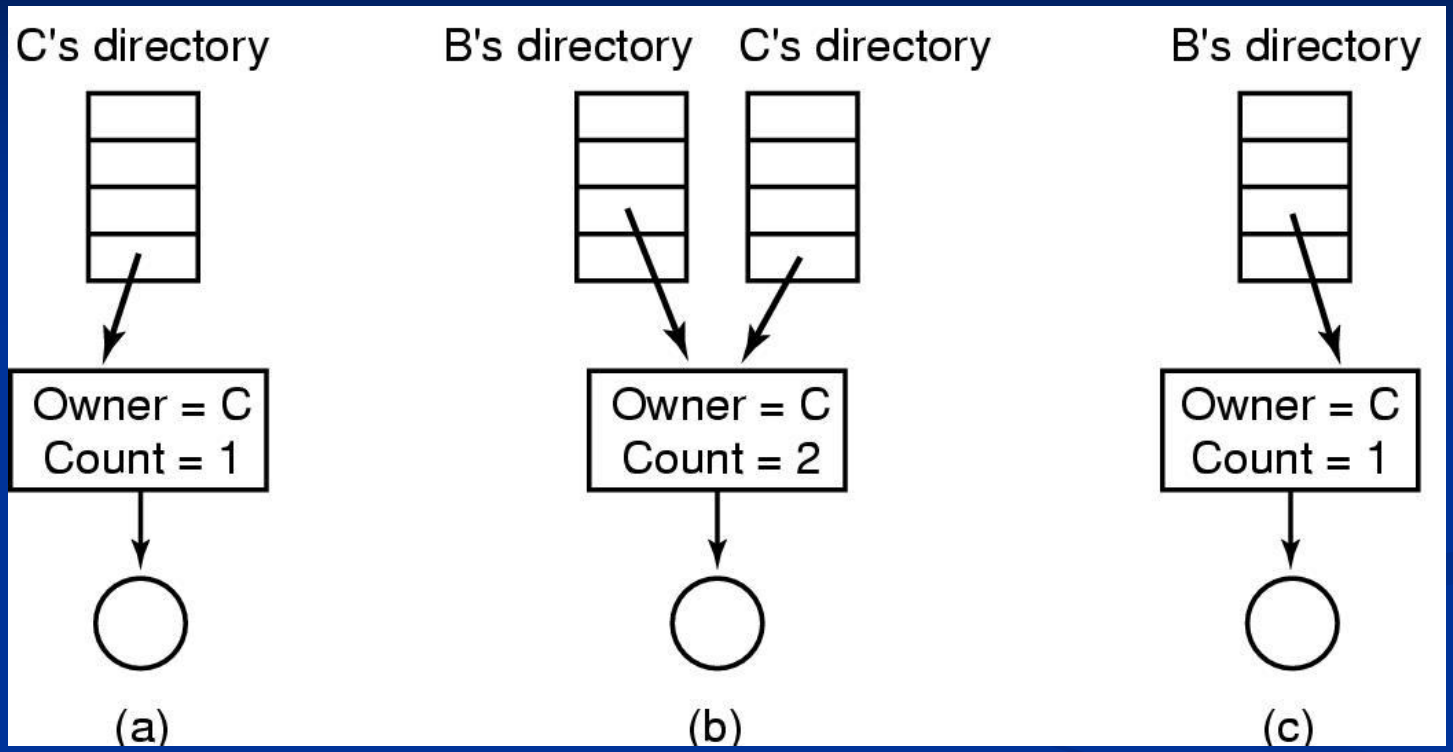
- Hard link
  - Direct link from multiple directories
  - Multiple owners, all with same rights
    - removed from one directory ⟶ available still from other locations
- Soft link (symbolic link)
  - New file, file type: symbolic link
    - Content of the new file: path to the actual file
  - Original file exists only in one directory (-> just one owner)
    - owner removes ⟶ the others cannot access, the link is invalid



Shared file

Directory structure - Directed Acyclic Graph

Shared Files: example



(a) Situation prior to linking

(b) After the link is created

(c)After the original owner removes the file

Logs and journaling

Log-Structured File Systems

- With CPUs faster, memory larger
  - disk caches can also be larger
  - increasing number of read requests can come from cache
  - thus, most disk accesses will be writes
- LFS Strategy structures entire disk as a log
  - have all writes initially buffered in memory
  - periodically write these to the end of the disk log
  - when file opened, locate i-node, then find blocks
  - To free no longer user blocks, cleaner thread compacts the content

Journaling File Systems: NTFS, ext3fs, ReiserFS

- Created to improve robustness and speed up crash recovery
- Copies the log idea from the logging FS
- First write to log intentions, then do operations
- Log elements:
  - Idempotent – can be repeated several times
  - Contain all structural changes – during recovery processing the log is enough

CS Dept use ext3fs in all file servers

Virtual file systems

Virtual File System

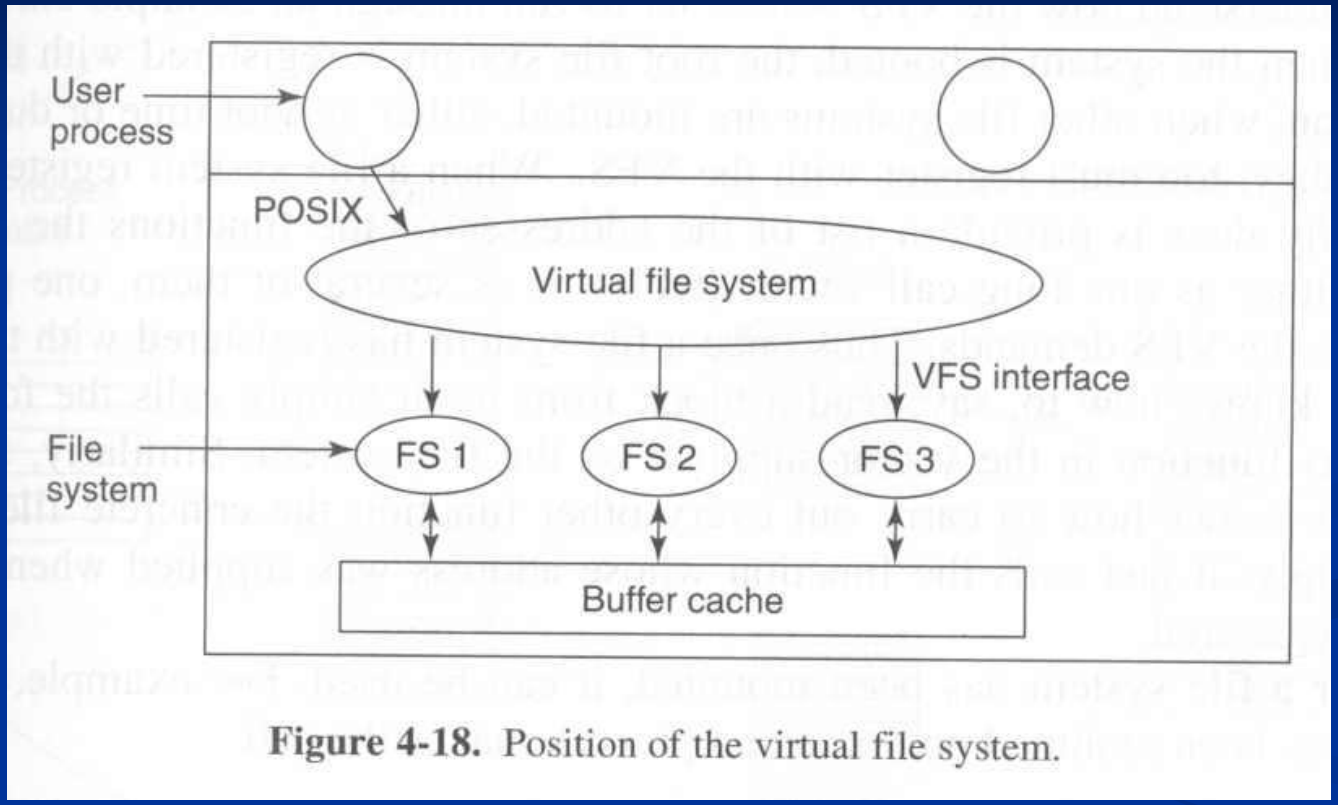


Figure 4-18. Position of the virtual file system.

- Abstract common part to a separate (VFS) layer
- This layer uses VFS interface to call the actual file system services
- Original motivation NFS

LINUX VFS

- Identical interface from the applications
- Supports several actual different file systems
- All requests via VFS

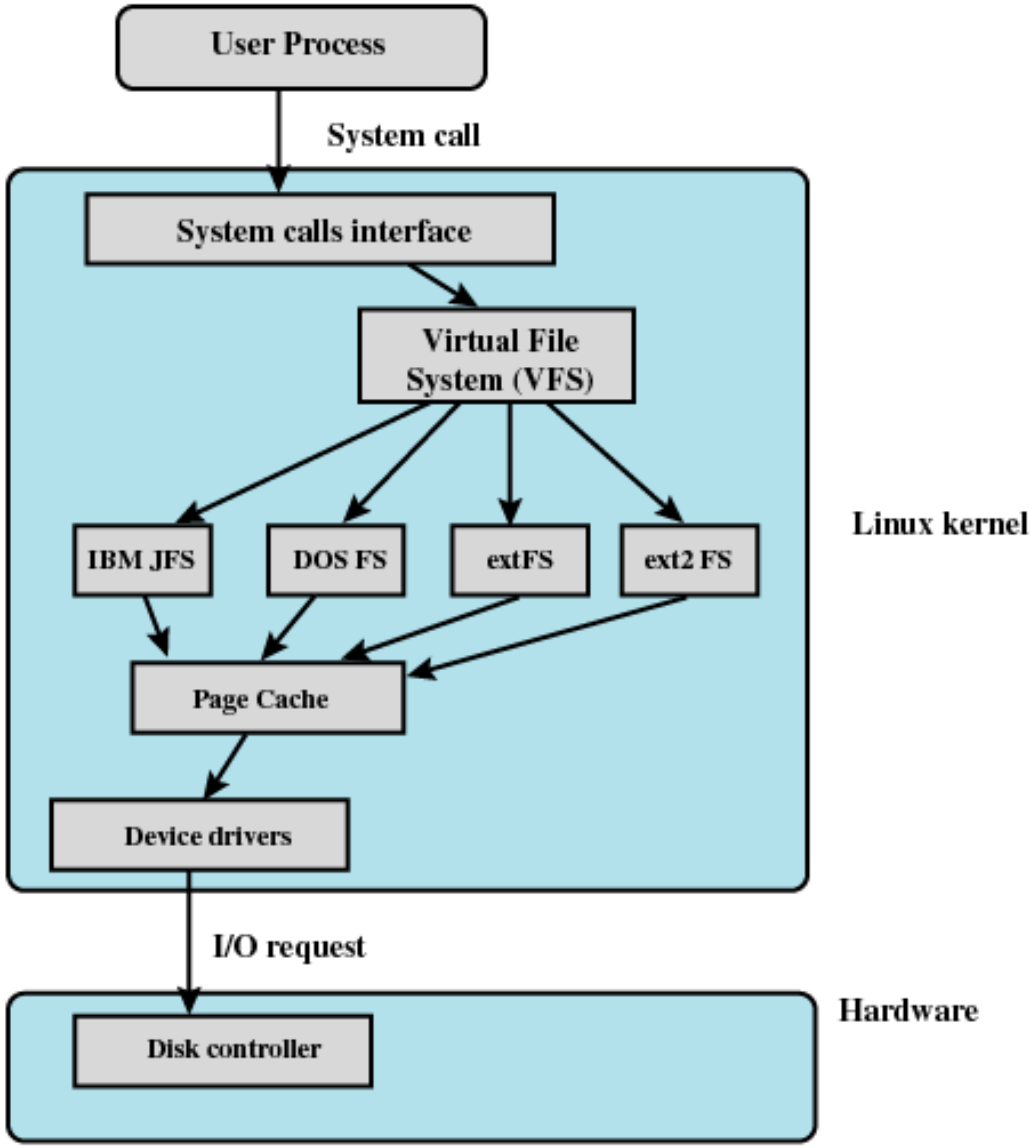


Figure 12.15 Linux Virtual File System Context

VFS: Example

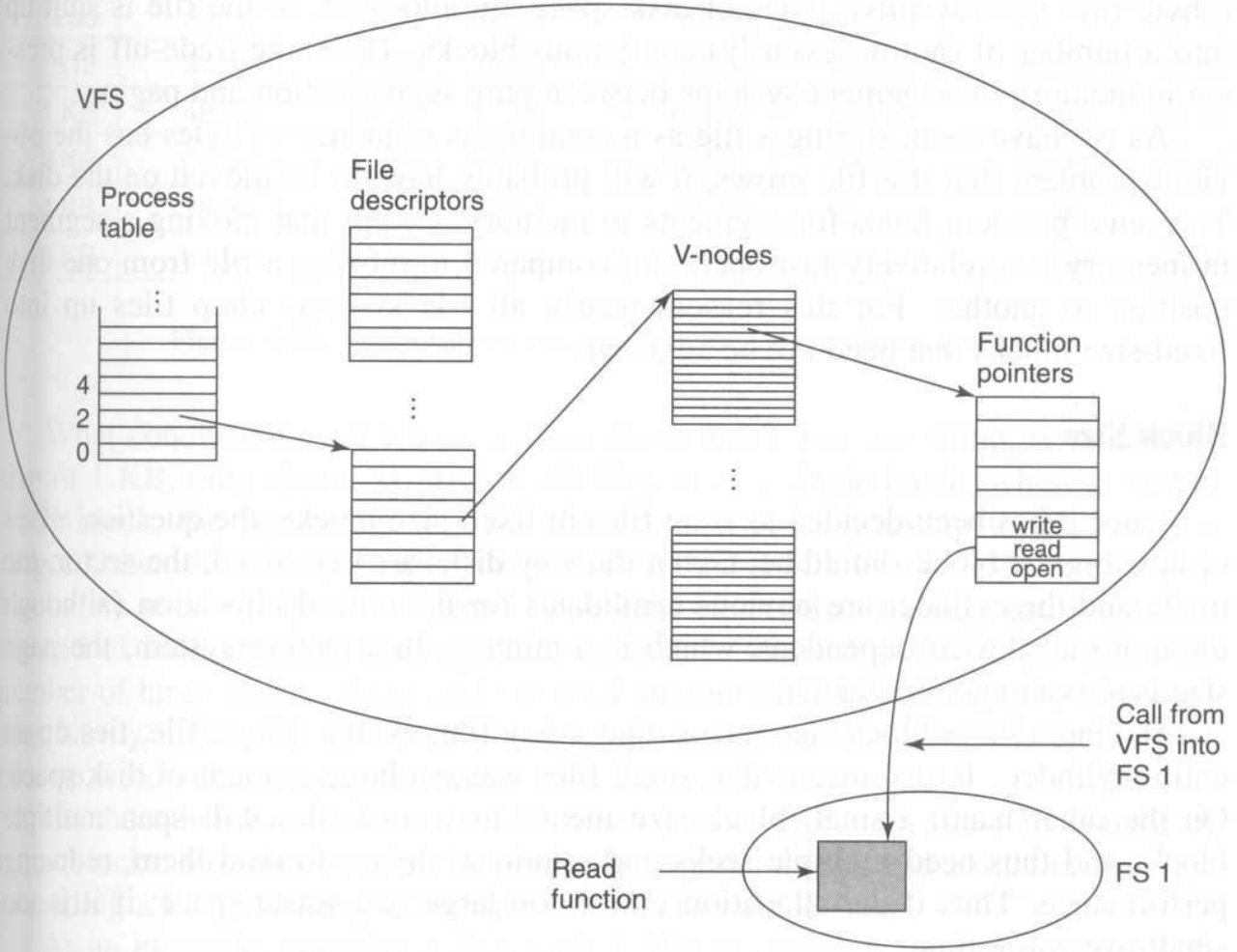


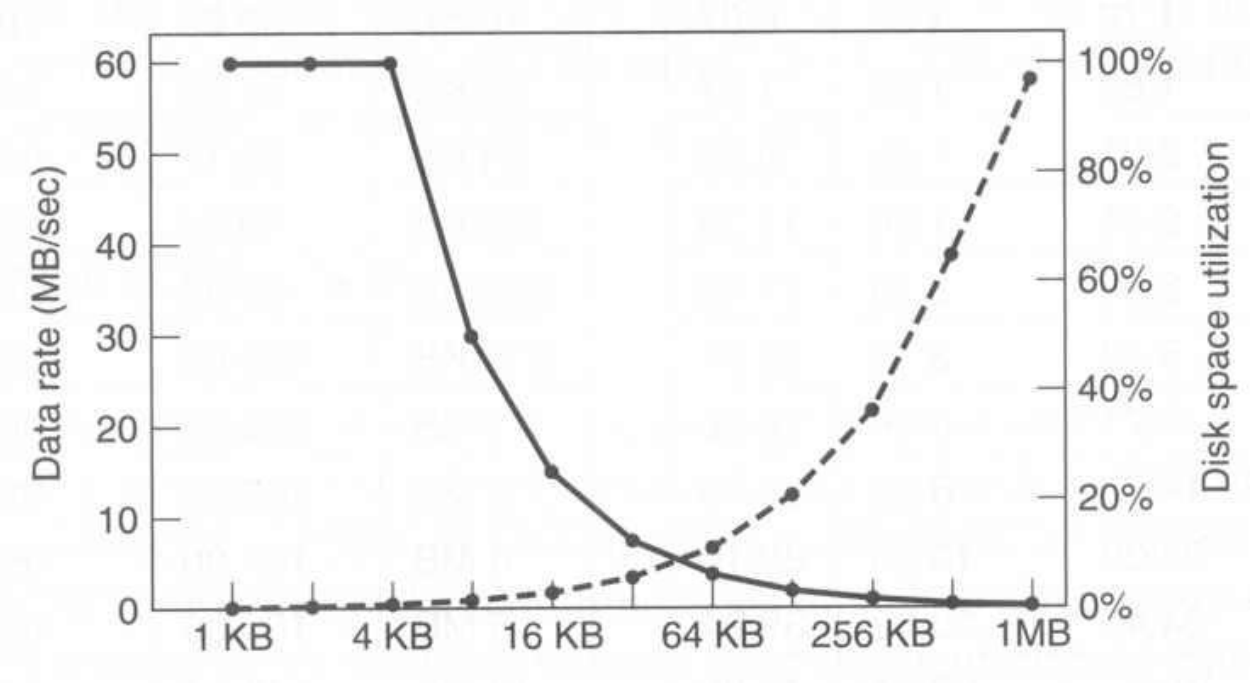
Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a read.

Space management, block allocation

Disk space management

- Two alternatives for storing an n byte file
  - N consecutive bytes of disk space allocated for the file
  - File split up to number of (not necessarily) contiguous blocks
- File size grows?
- Block size?
  - Sector, track, cylinder size - device dependent
  - Page size
  - Large block – small file wastes large amount of space
  - Small block – files span on several blocks (read separately)
  - Study at VU: 60% smaller than 4KB, 90% smaller than 64KB

Effect of block size



X: block size

Y: *Dotted* line (left hand scale) gives data rate of a disk

Dark line (right hand scale) gives disk space efficiency

Disk: 1 MB per track

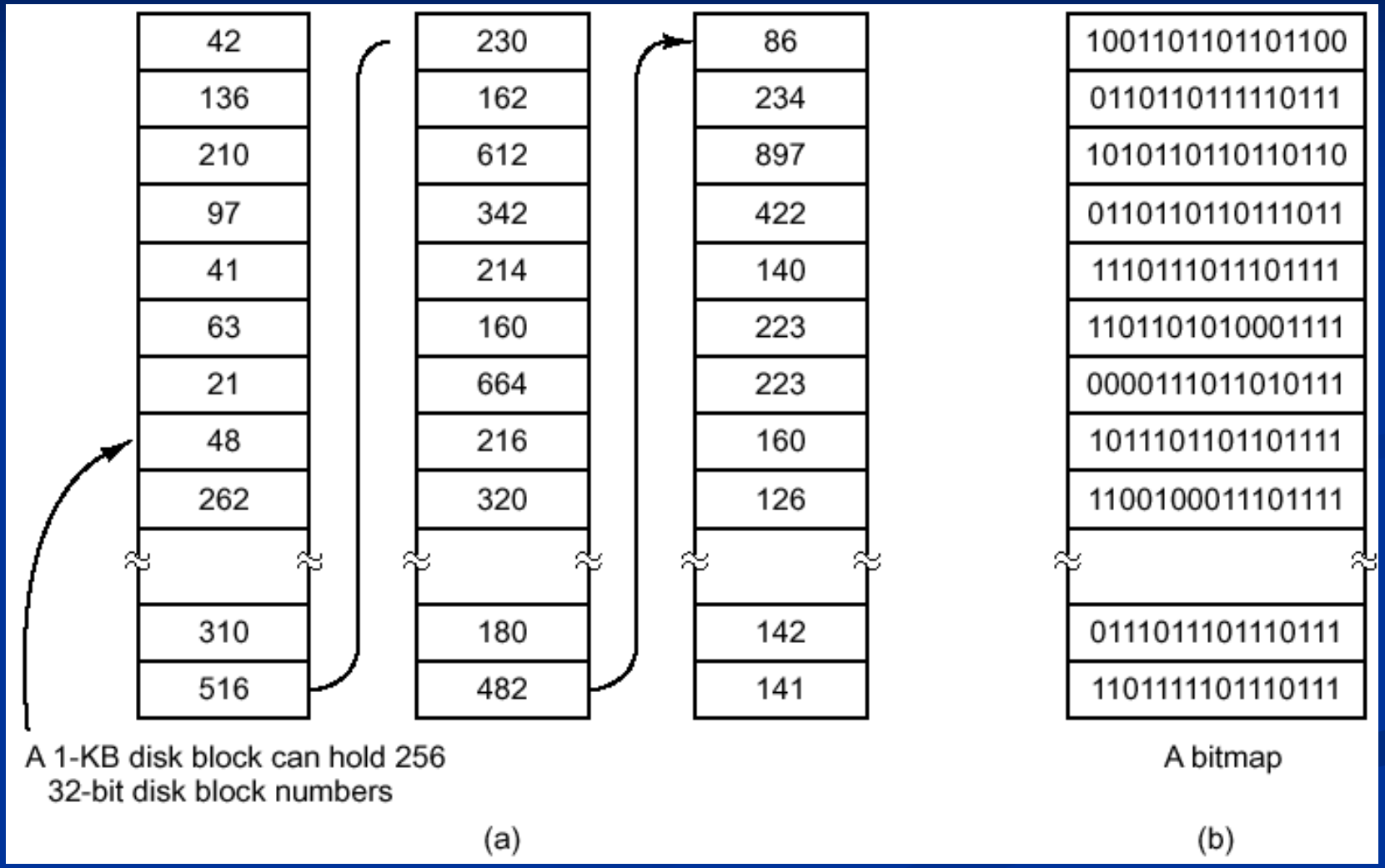
rot. time 8.33 msec

seek time 5 msec

All files 4KB

- The access time for the block is dominated by the seek time and rotational delay
- With small files, the space efficiency drop fast with large block size

Free blocks Tan08 4-22



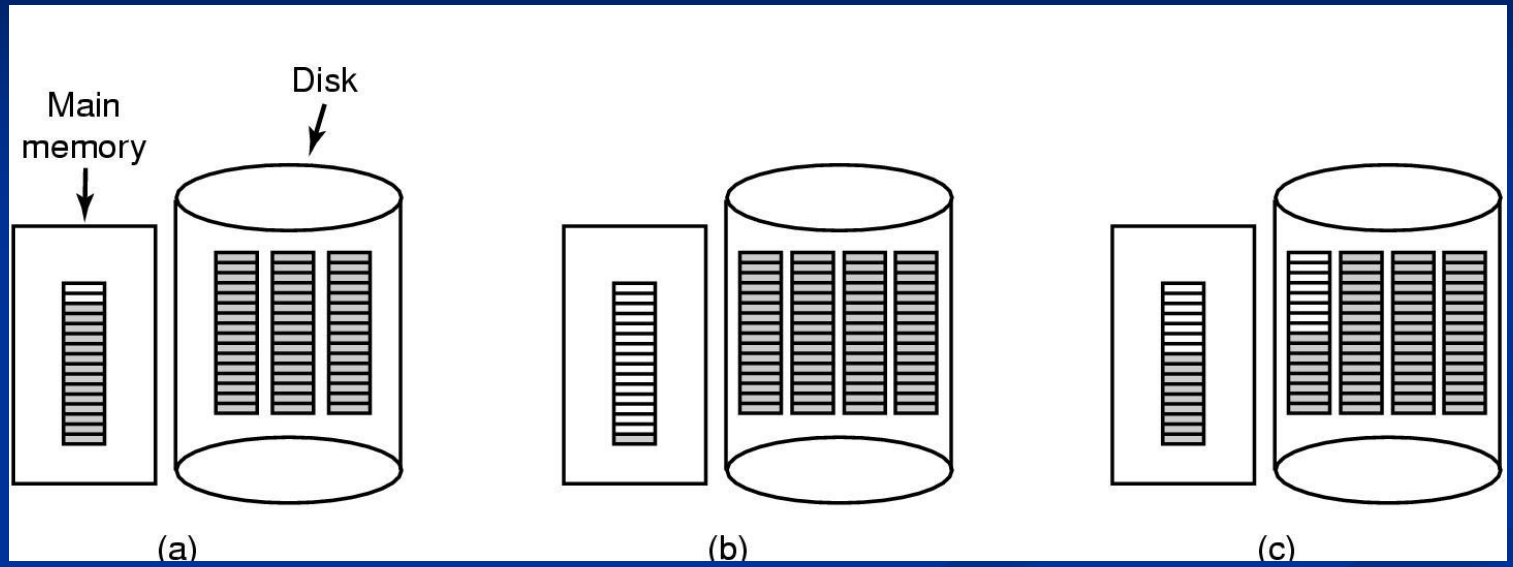
Linked list of blocks

Bit map

Free blocks

- Space on disk:
  - Bit map needs less space, unless disk very full
- Space in memory with linked list:
  - Only one block of the linked list needs to be in memory
  - (See next page for problem in this)
- Space in memory with bit map:
  - Only part of the bit map in memory allocations from it
    - Keeps file close to each other
  - Freeing may require reading some other part to memory
  - Using virtual memory here might ease implementation

One block from free list in memory



(a) Almost-full block of pointers to free disk blocks in RAM

- three blocks of pointers on disk

(b) Result of freeing a 3-block file

- empty block in memory (previous written to disk, need to read new soon)

(c) Alternative strategy for handling 3 free blocks

- shaded entries are pointers to free disk blocks