CSCI 360
Introduction to Operating Systems

File Systems

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Outline

- File Abstraction
- File Storage
- File Concepts
- Directory Concepts
- Disk Partitions and File System Layout
- Disk Block Allocation
- Free Disk Block Management
- File System Performance

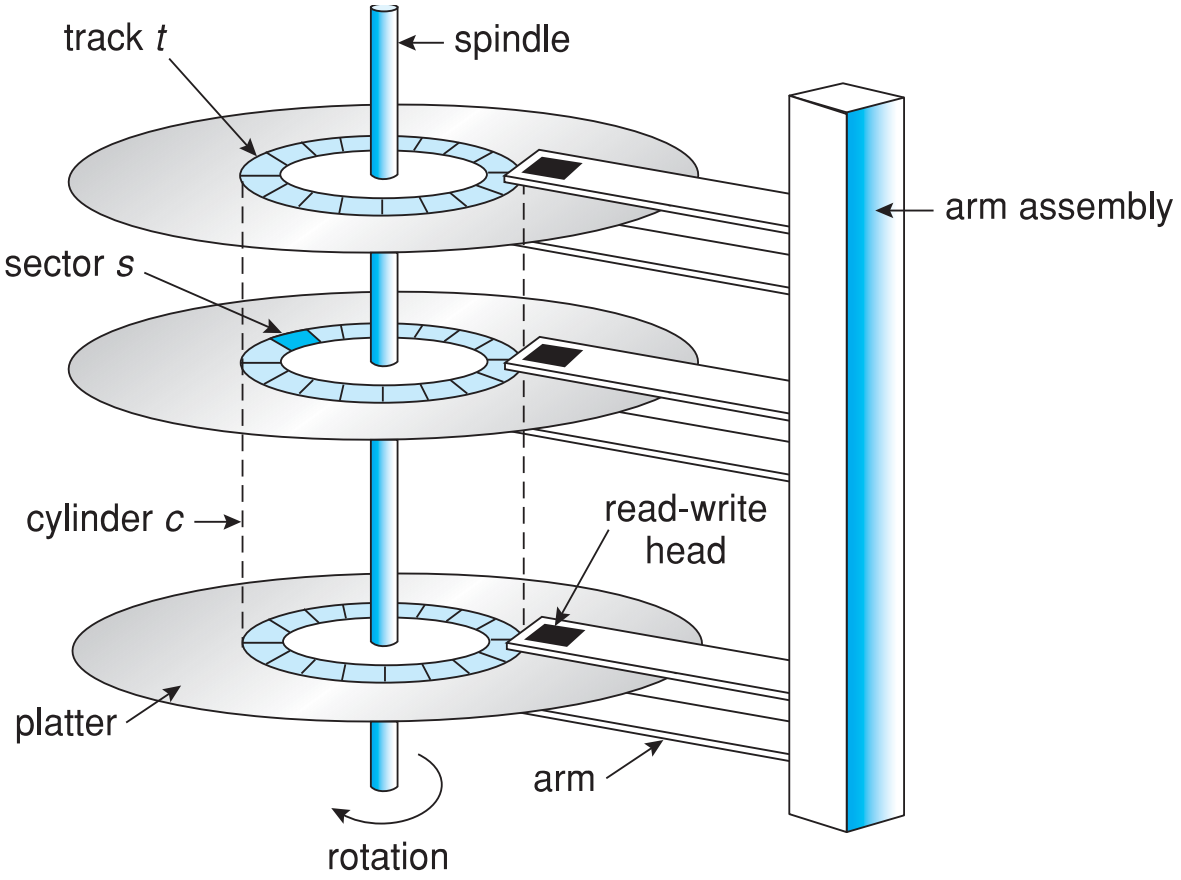
File Abstraction

File abstraction is resulted from the essential requirements for **long-term** information **storage**:

1. System needs to **store** a very large amount of information.
2. Stored information must **survive** termination of process that was using it.
3. If approved, multiple processes must be able to **access** the stored information concurrently.

File Storage: Hard Disk

System stores a large amount of information as a file on a hard disk

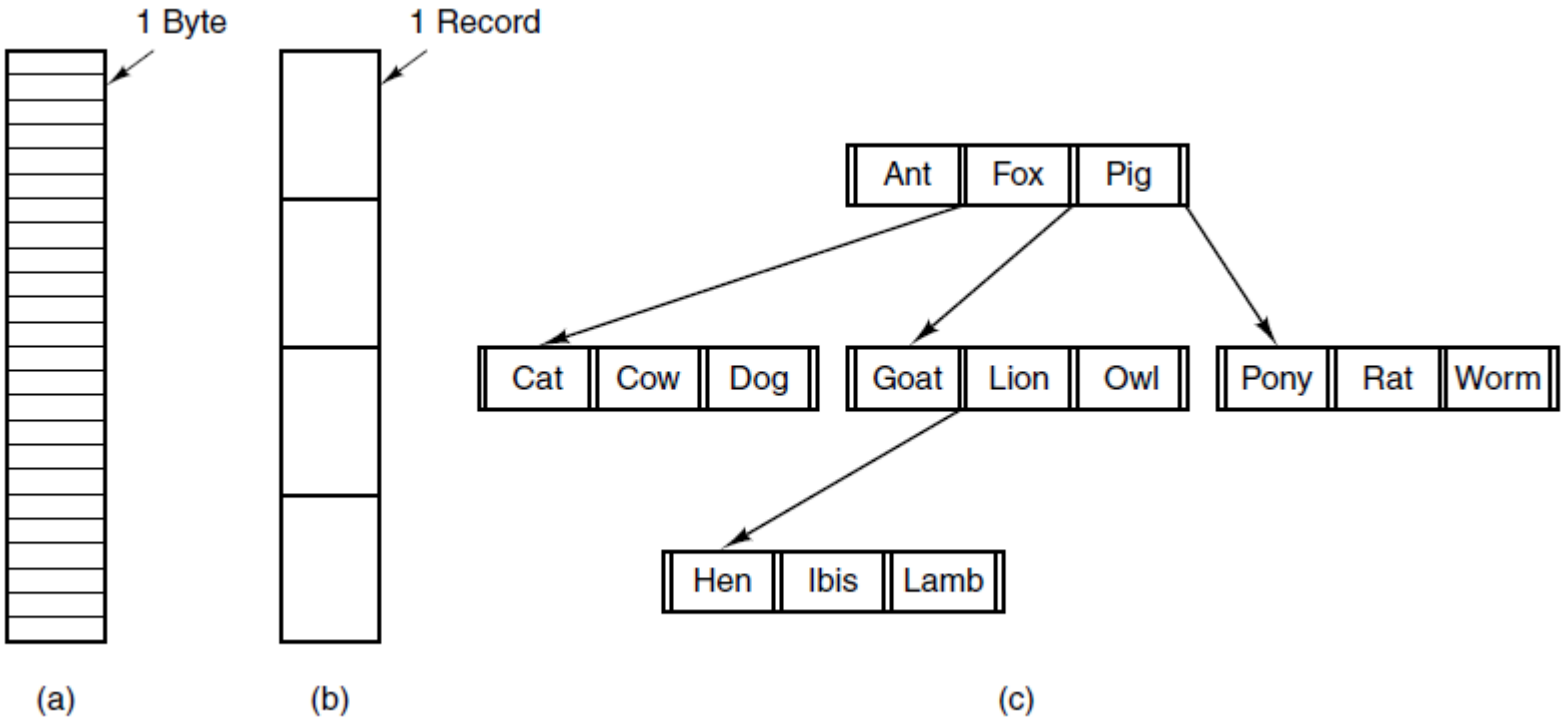


File Abstraction

- A disk is a **linear sequence** of fixed-size sectors or blocks and supporting two operations:
  - ✓ Read block *k*.
  - ✓ Write block *k*
- Information of a file may need **one** or **more** disk blocks.
- System **allocates** free disk blocks to a file and keeps **track** of this allocation.

File Structure

Information may be **structured** in different ways **inside** a file



Byte sequence

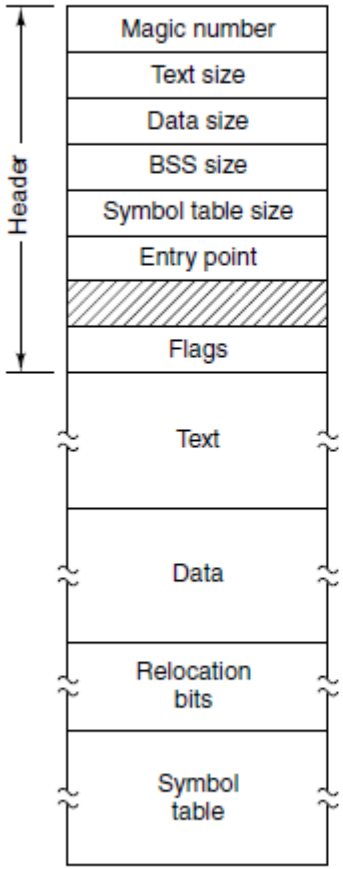
Record sequence

Tree Structure

File Types

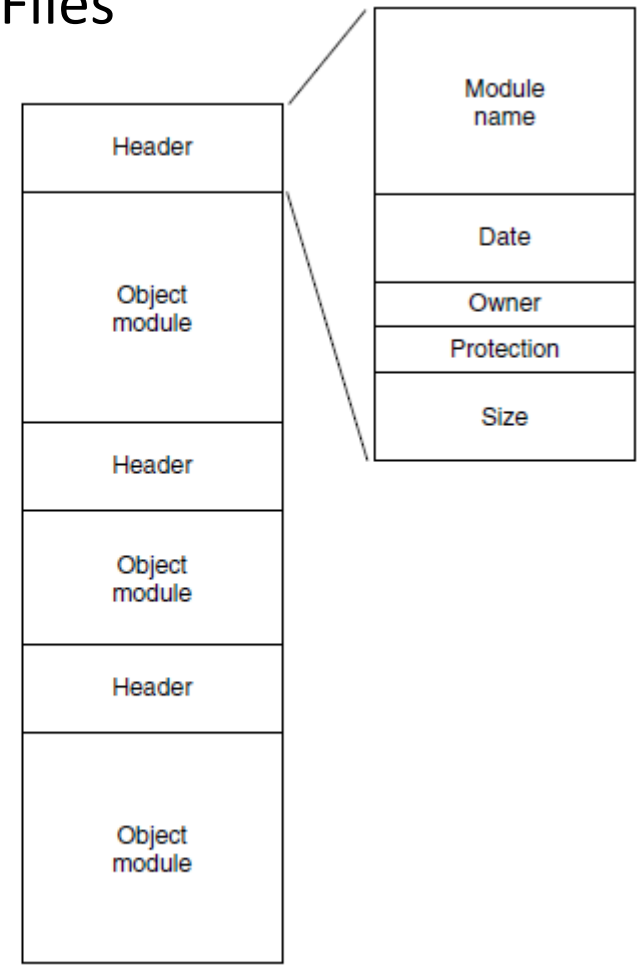
ASCII File

BINARY Files



(a)

Executable file



(b)

Archive file

File Abstraction: Name

- Files are **identified** by their **names**.
- Each file has a **unique** file name.
- File names also have **file extensions** that indicate their types.

File Abstraction: Name Extension

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

Some typical file extensions.

File Abstraction: Attributes

- System **maintains** many metadata or attributes about each file.
- System uses file attributes to **manage** all the files in the system.

File Abstraction: 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 was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Some possible file attributes.

File Operations

System **permits** many operations on a file.

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set attributes
11. Rename
12. Link
13. Unlink

Example Program Using File System Calls

```
#define TRUE 1
#define BUF_SIZE 40960
#define OUTPUT_MODE  0700

int main(int argc, char** argv) {
        int in_fd, out_fd, rd_count, wt_count;
        char buffer[BUF_SIZE];
        if(argc != 3) exit(1);
        in_fd = open(argv[1], O_RDONLY);
        if(in_fd < 0) exit(2);
        out_fd = creat(argv[2], OUTPUT_MODE);
        if(out_fd < 0) exit(3);
        while(TRUE) {
                rd_count = read(in_fd, buffer, BUF_SIZE);
                if(rd_count <= 0) break;
                wt_count = write(out_fd, buffer, rd_count);
                if(wt_count <= 0) exit(4);
         }
         close(in_fd);
         close(out_fd);
}
```

A simple program to **copy** a file

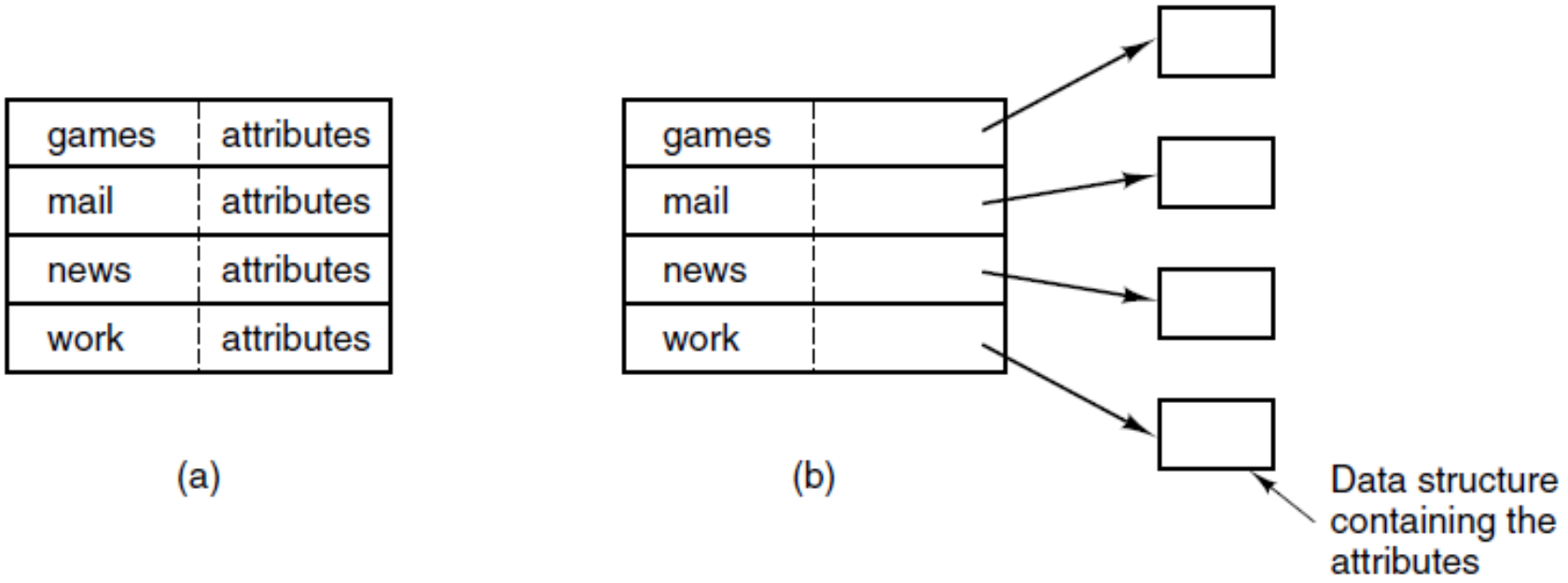
Directory Abstraction

- System uses directories or folders to **organize** hundreds of files in a file system.
- Files are **stored** inside a directory.
- A directory may **contain** one or more subdirectories.
- Each directory name uniquely **identifies** a directory or subdirectory.

Directory Abstraction

- The contents of a directory, **files** and **subdirectories,** commonly **referred to** as directory entries.
- Directory entry **holds** the name and the attributes of a file or subdirectory.
- Disk blocks **allocated** to a file or a subdirectory is one of its attributes.

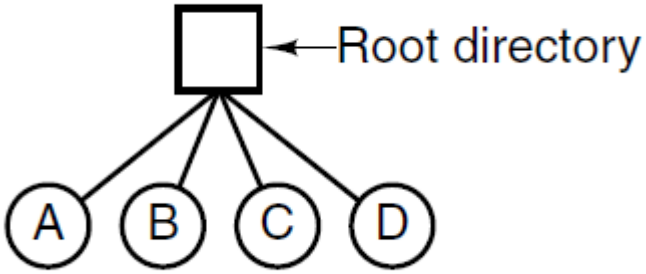
Directory Entries



a) A simple directory in which each entry **contains** the name and attributes.

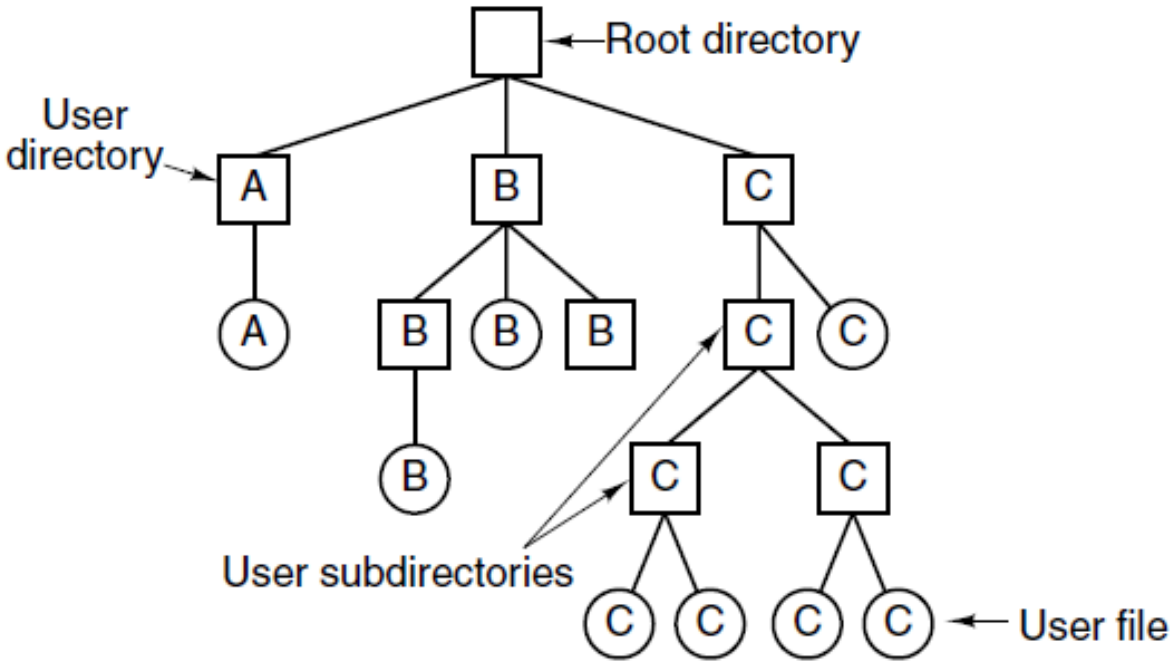
b) A directory in which each entry **refers** to a data structure, which **contains** both name and attributes.

Single-Level Directory Systems



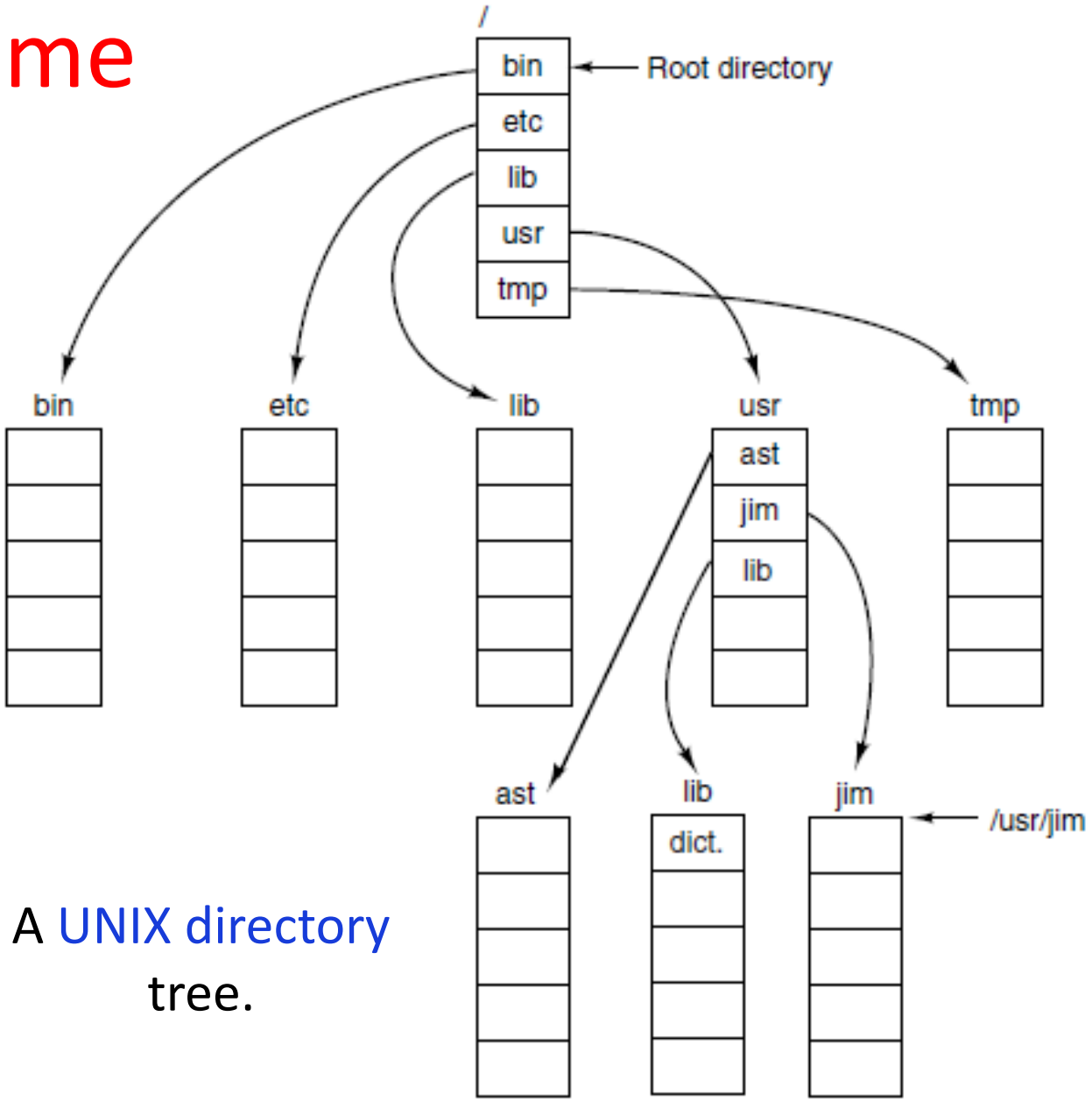
A single-level directory system containing four files.

Hierarchical Directory Systems



A hierarchical directory system with subdirectories and files

Path Name



A UNIX directory tree.

Directory Operations

System **permits** many operations on a directory.

1. Create
2. Delete
3. Opendir
4. Readdir
5. Rename
6. closedir

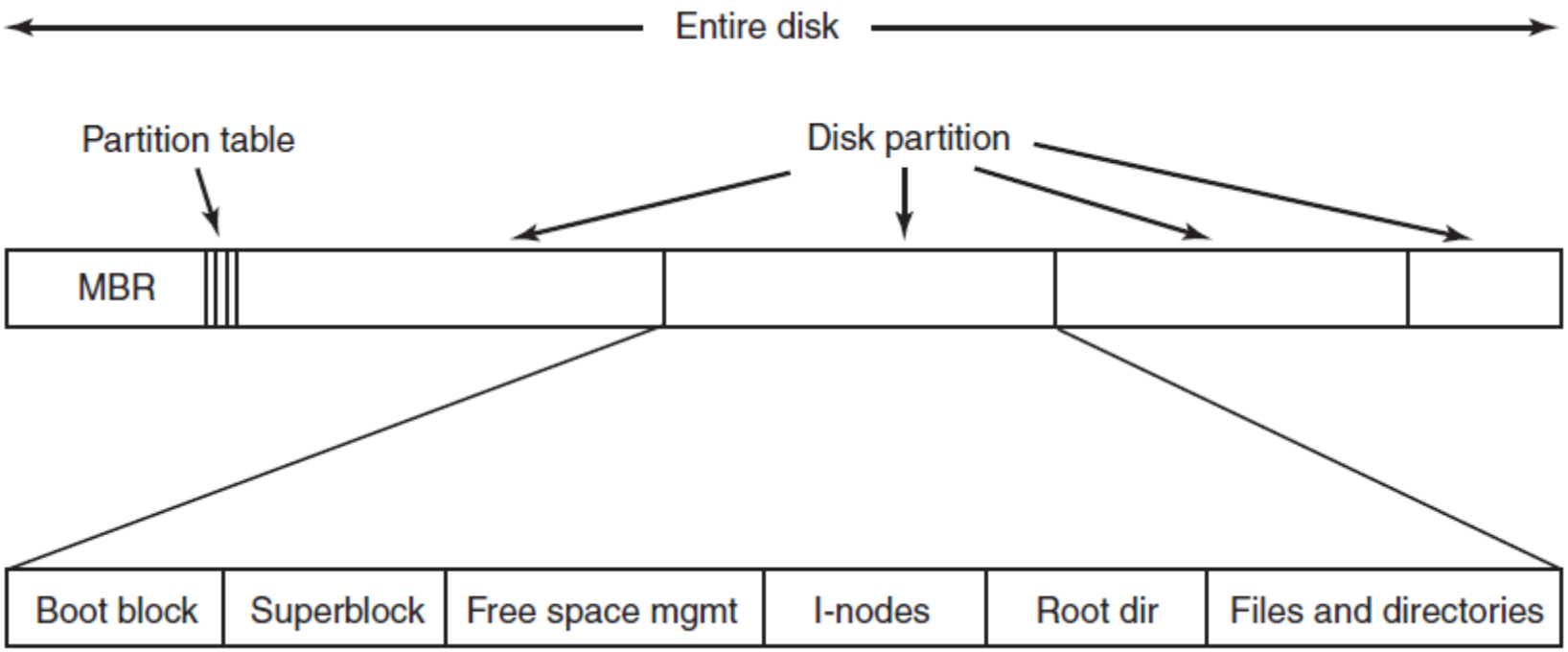
File System Disk Layout

- **First** disk block is the **Master Boot Block** (MBR), which **contains** the code that is executed when the computer starts.
- A disk is **partitioned** into several partitions to **hold** multiple operating systems on the same disk; one in each partition and only one partition is **active**.
- A partition table is kept after MBR on the disk that keeps **track** of the starting addresses of all the partitions and the active partition.

File System Disk Layout

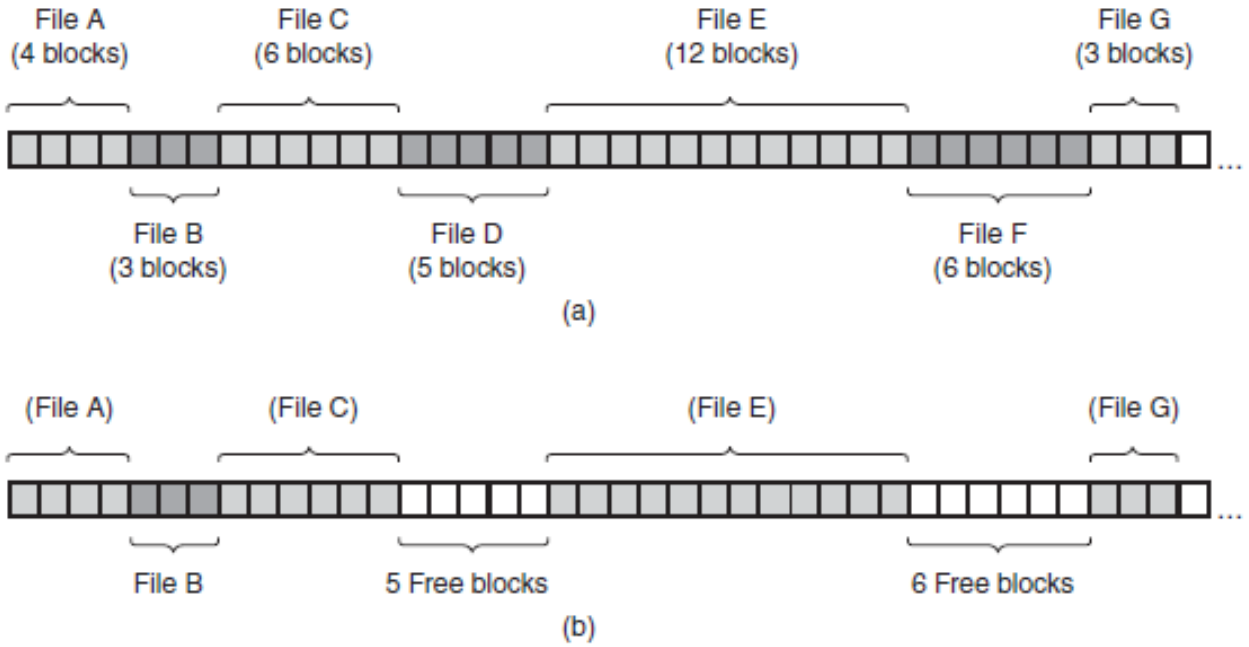
- MBR code **reads** the partition table and **locates** the active partition, then **reads** the first block, called boot block**,** of the active partition and **executes** it.
- Boot block code **loads** the operating system, including filesystem, from the partition into the memory.
- Filesystem information **consists** of *superblock* (filesystem's metadata, e.g., filesystem type, number of blocks etc.), *free-block* information, *allocated-block* information, ***root directory*** information, and the *other directories* and *files* information.

File System Disk Layout



A possible file system layout

Disk Block Allocation: Contiguous

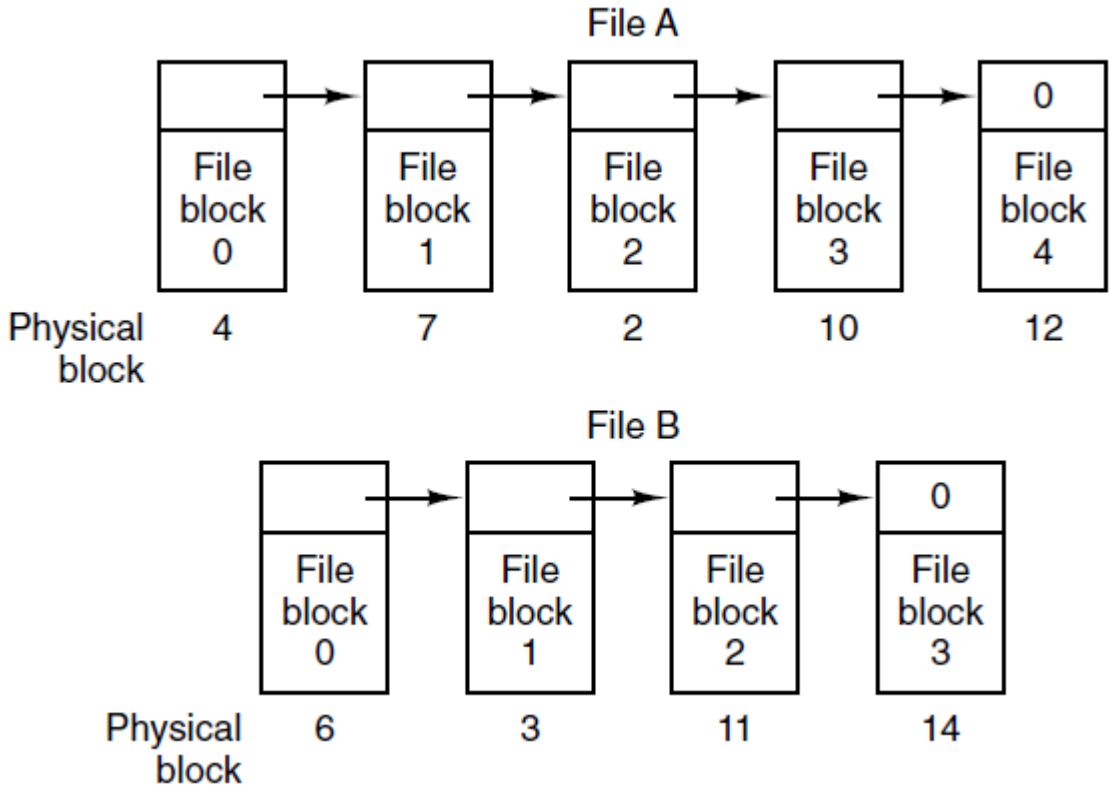


(a) Contiguous allocation of disk space for seven files.

(b) The state of the disk after files *D* and *F* have been **removed**.

Contiguous allocation is not used

Disk Block Allocation: Noncontiguous

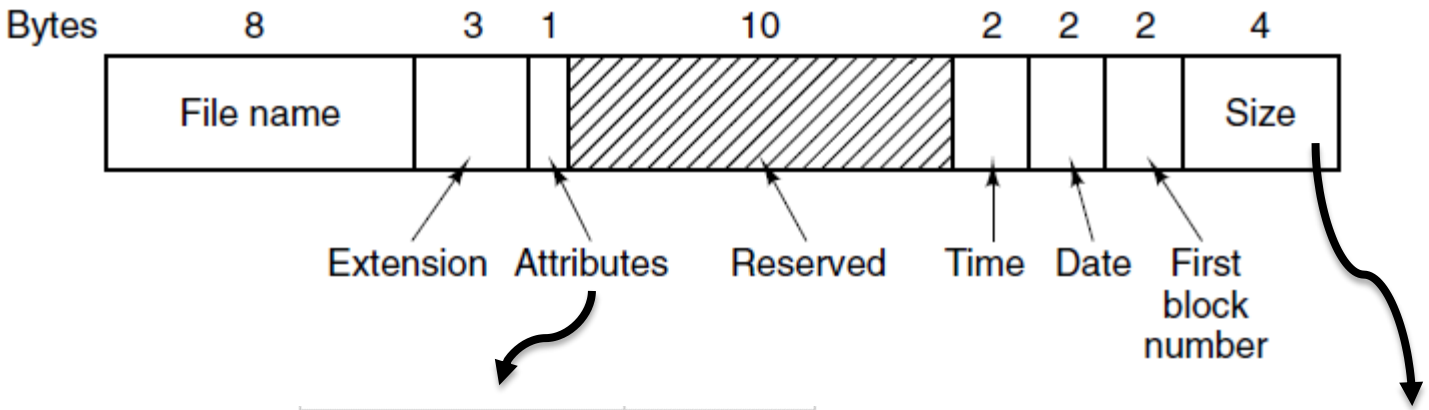


Storing a file as a linked list of disk blocks.

Noncontiguous allocation is **being used** in many operating systems

The MS-DOS File System

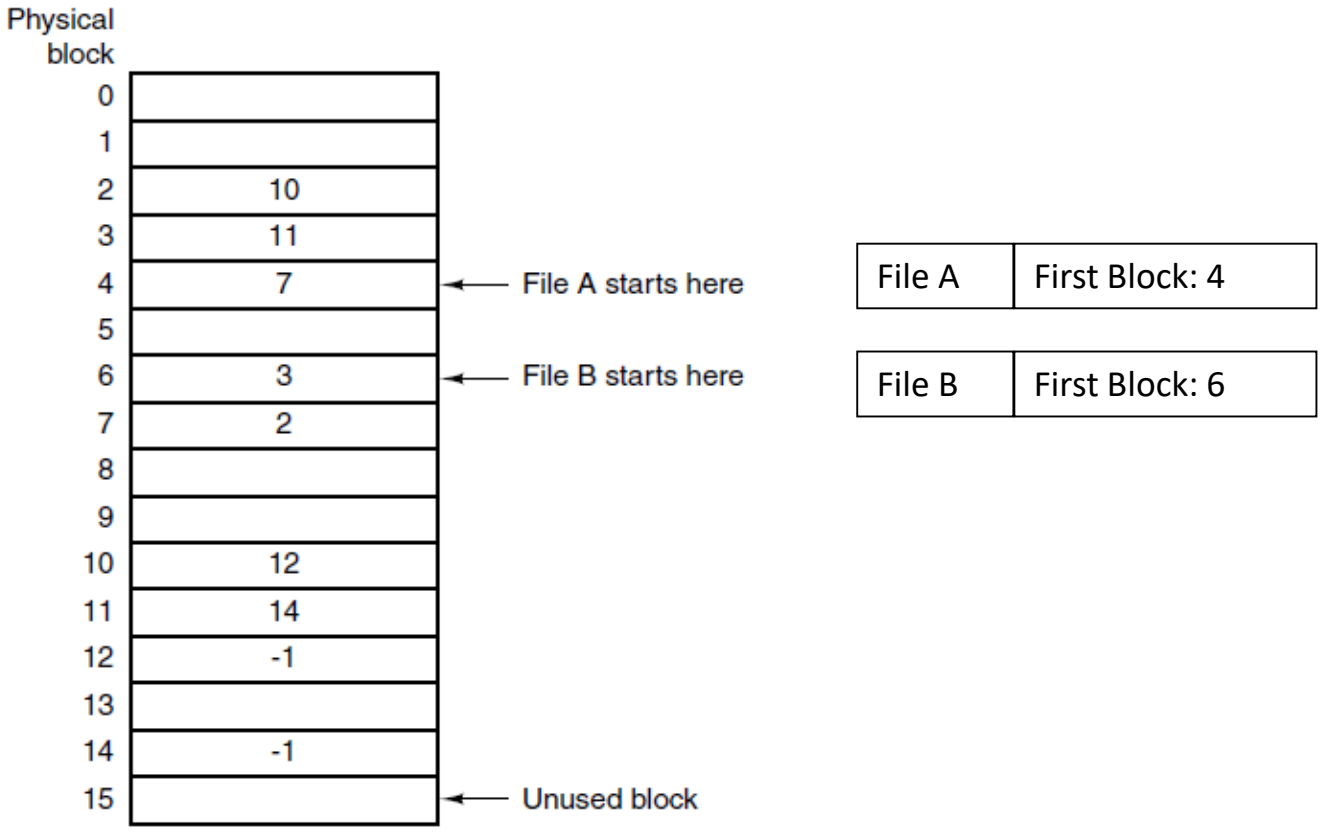
The MS-DOS directory entry **contains** the disk addresses and attributes.



READONLY	0x00000001
HIDDEN	0x00000002
SYSTEM	0x00000004
	0x00000008
DIRECTORY	0x00000010
ARCHIVE	0x00000020

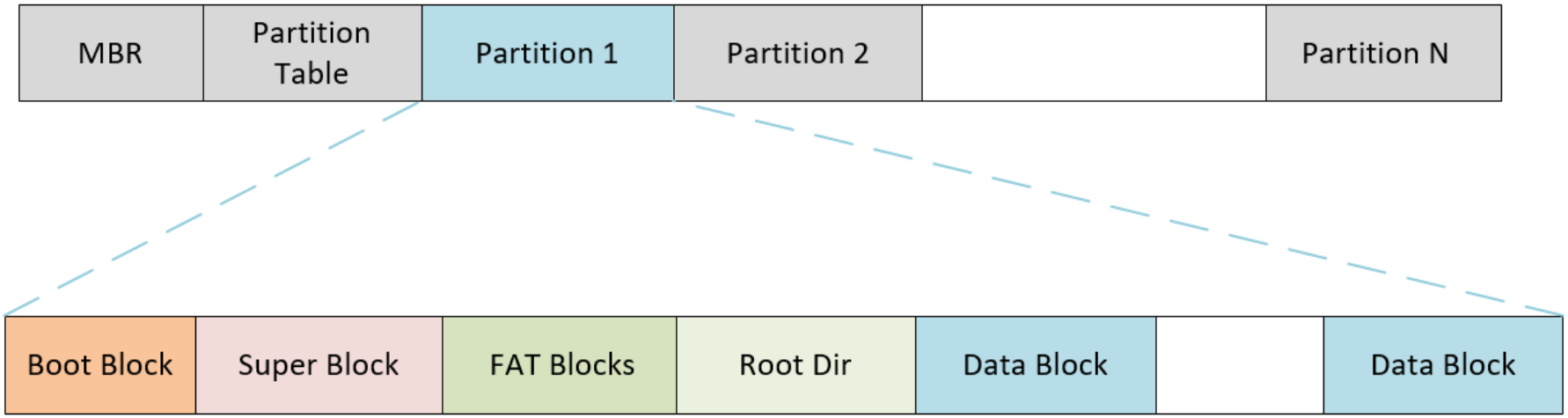
File Size
4GB in theory
2GB in practice

The MS-DOS File System



File Allocation Table (FAT)

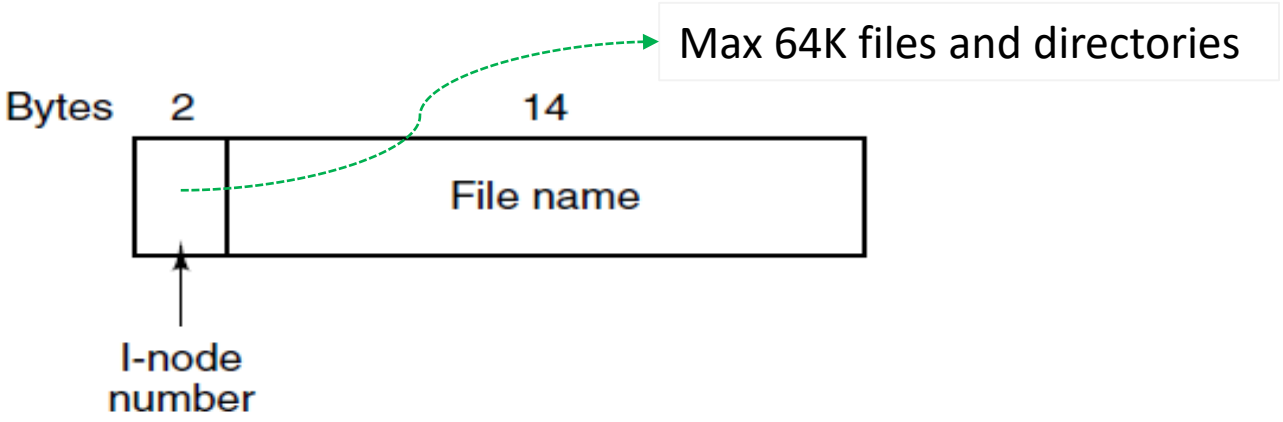
The MS-DOS File System



- FAT **keeps track** of both free and allocated disk blocks.
- Root Dir disk block is **fixed** and right after FAT Blocks.

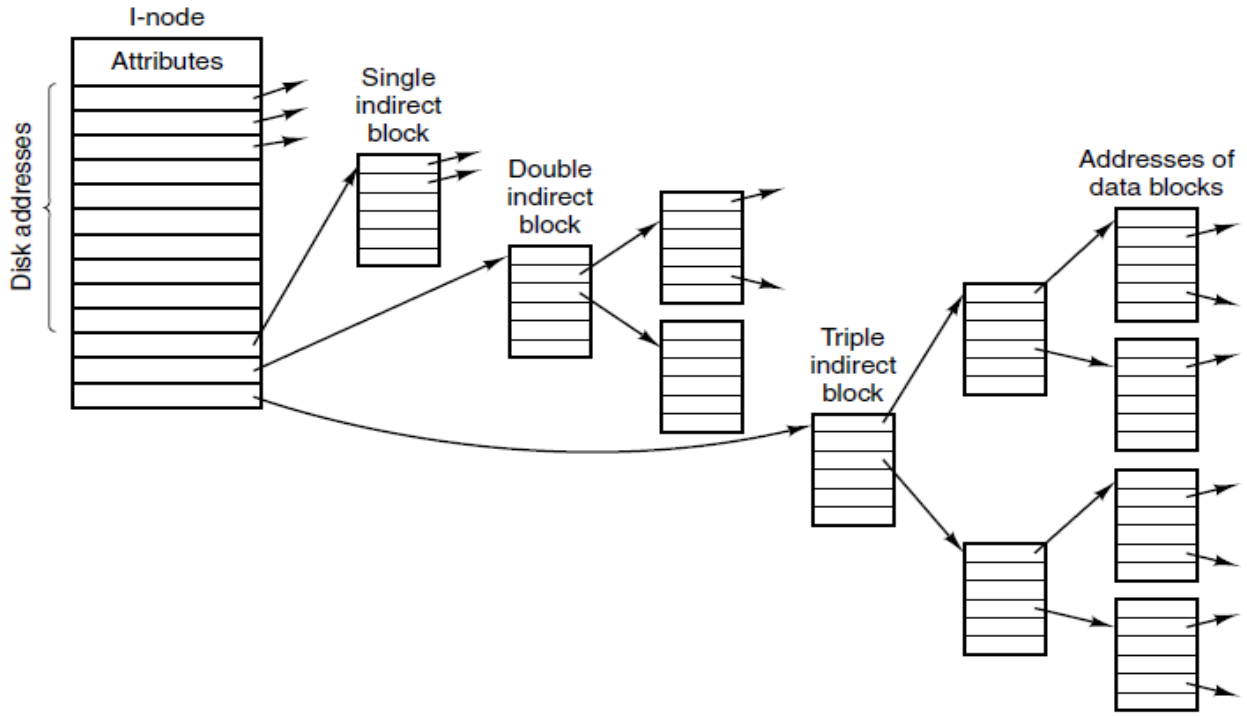
The UNIX V7 File System

- UNIX V7 directory entry **contains** only file name and a pointer to an I-node
- I-node **contains** the disk addresses and file attributes

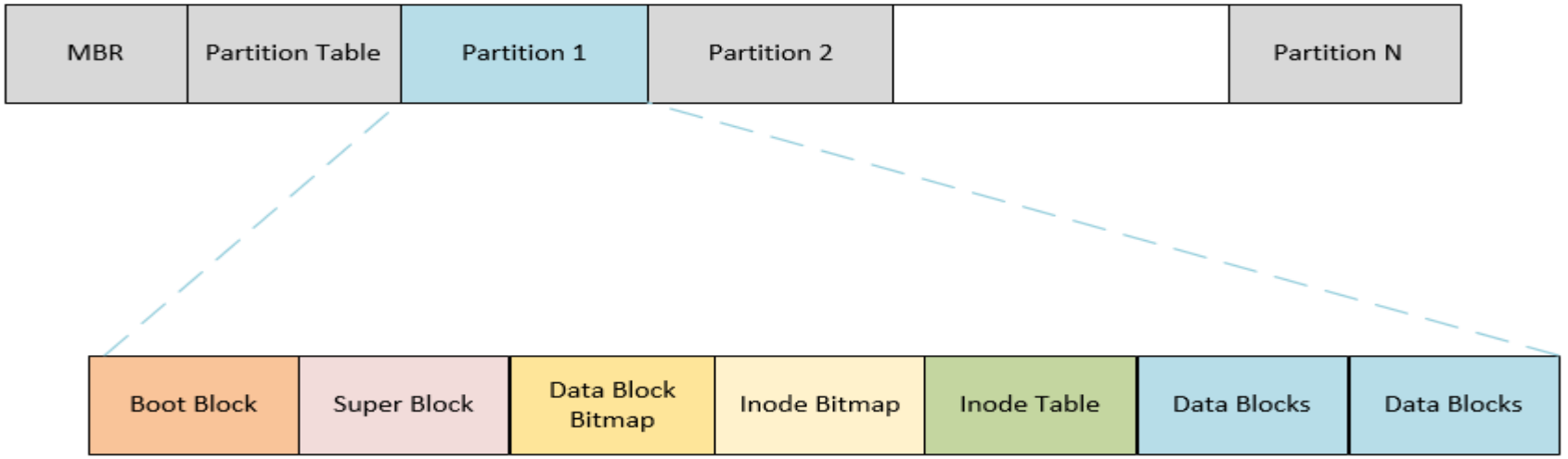


The UNIX V7 File System

- UNIX I-node **contains** file attributes and disk addresses.
- The **first 10** disk block address is directly **kept** in the I-node.
- I-node also **contains** the addresses of *single*, *double*, and *triple indirect* disk block addresses.



The UNIX V7 File System



- Data Block Bitmap **keeps track** of free disk blocks.
- Inode Bitmap **keeps track** of free I-nodes.
- I-node table **contains** i-nodes that **keep track** of disk block allocation**.**
- Root Dir uses a **fixed** i-node.

The UNIX V7 File System

A UNIX i-node Attributes:

- File Size
- Creation Time
- Last Access Time
- Last Modified Time
- Owner
- Group
- Protection
- Link Count

Unix V7 File Systems

Steps to **add** a file in UNIX:

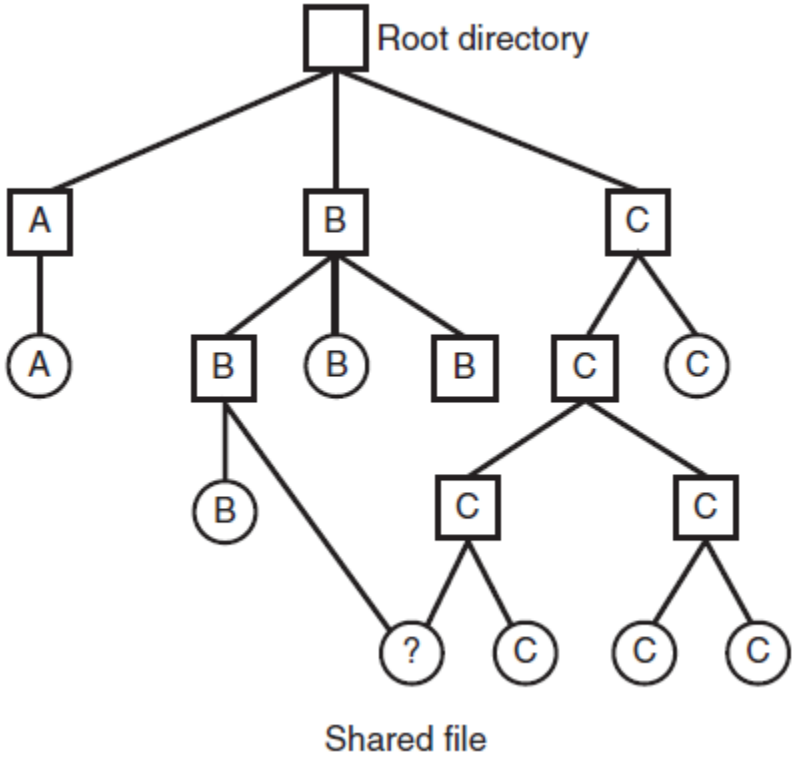
1. **Get** the disk blocks for the file from the pool of free disk blocks.
2. **Get** a free i-node for the file from the pool of free i-nodes.
3. **Record** the disk blocks into i-node and set the link count to 1.
4. **Add** a directory entry for the file in its directory.

Unix V7 File Systems

Steps to **remove** a file in UNIX:

1. **Remove** the directory entry of the file from its directory.
2. **Decrement** link count.
3. **Release** i-node to the pool of free i-nodes if link count reaches zero.
4. **Return** all disk blocks to the pool of free disk blocks.

Unix V7 Shared Files



File system containing a shared file.

Assume User B **shares** User C's file

Unix V7 File Systems

Steps to **add** shared file in UNIX:

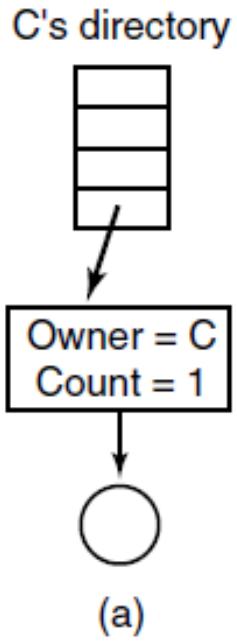
1. **Get** the i-node of the original file.
2. **Increment** the link count.
3. **Add** a directory entry for the shared file in its directory.
4. **Add** the i-node number of the original file in the new directory entry of the shared file.

Unix V7 File Systems

Steps to **remove** a shared file in UNIX:

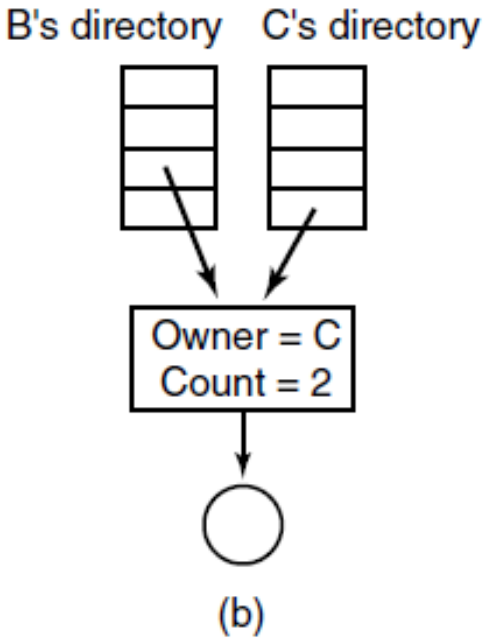
1. **Remove** directory entry of the shared file from its directory.
2. **Decrement** the link count in i-node of the original file.
3. If link count becomes zero **release** i-node to the pool of free i-nodes and **return** all disk blocks to the pool of free disk blocks.

Unix V7 Shared Files



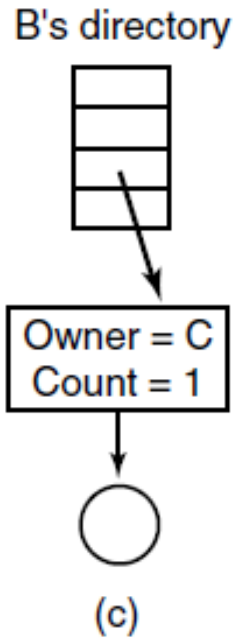
(a)

User C's file before **linking** by user B



(b)

After **linking** by user B

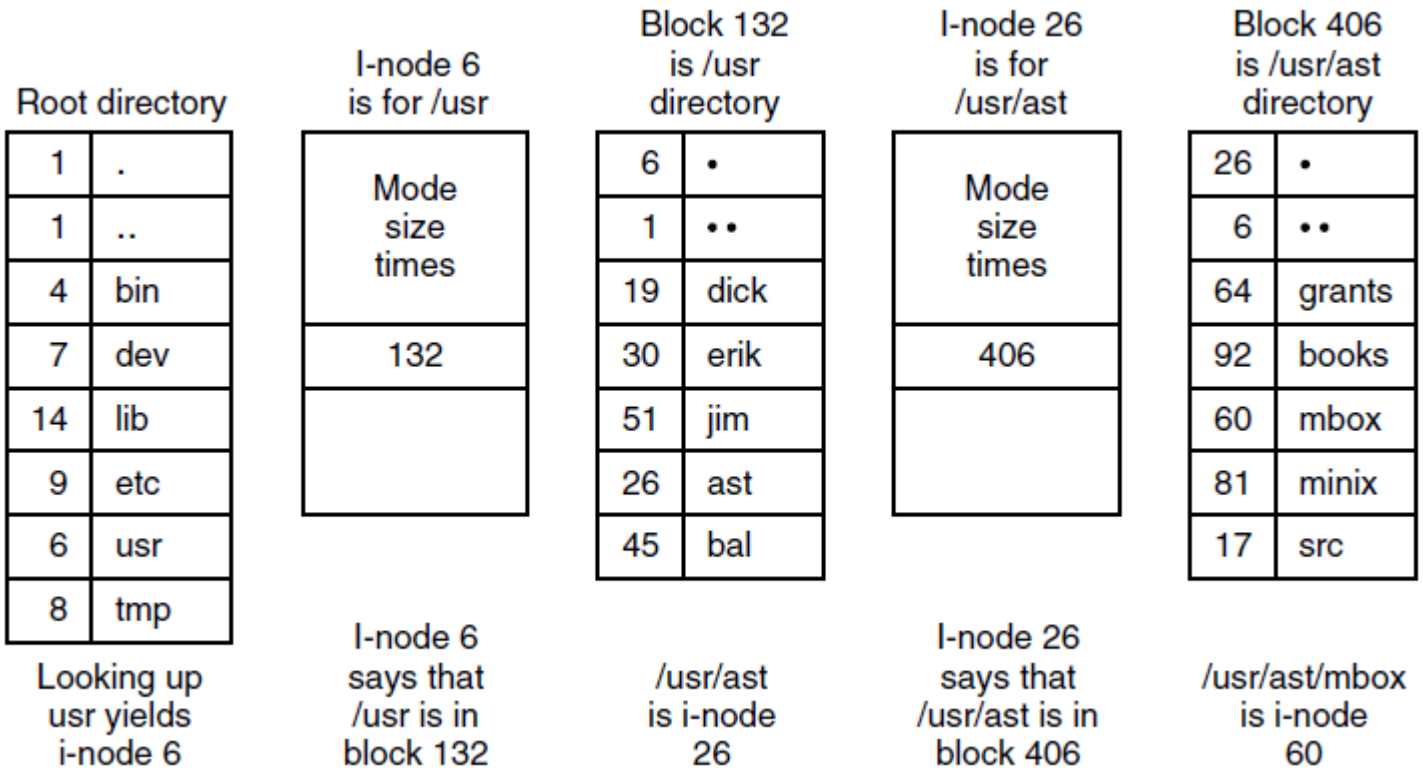


(c)

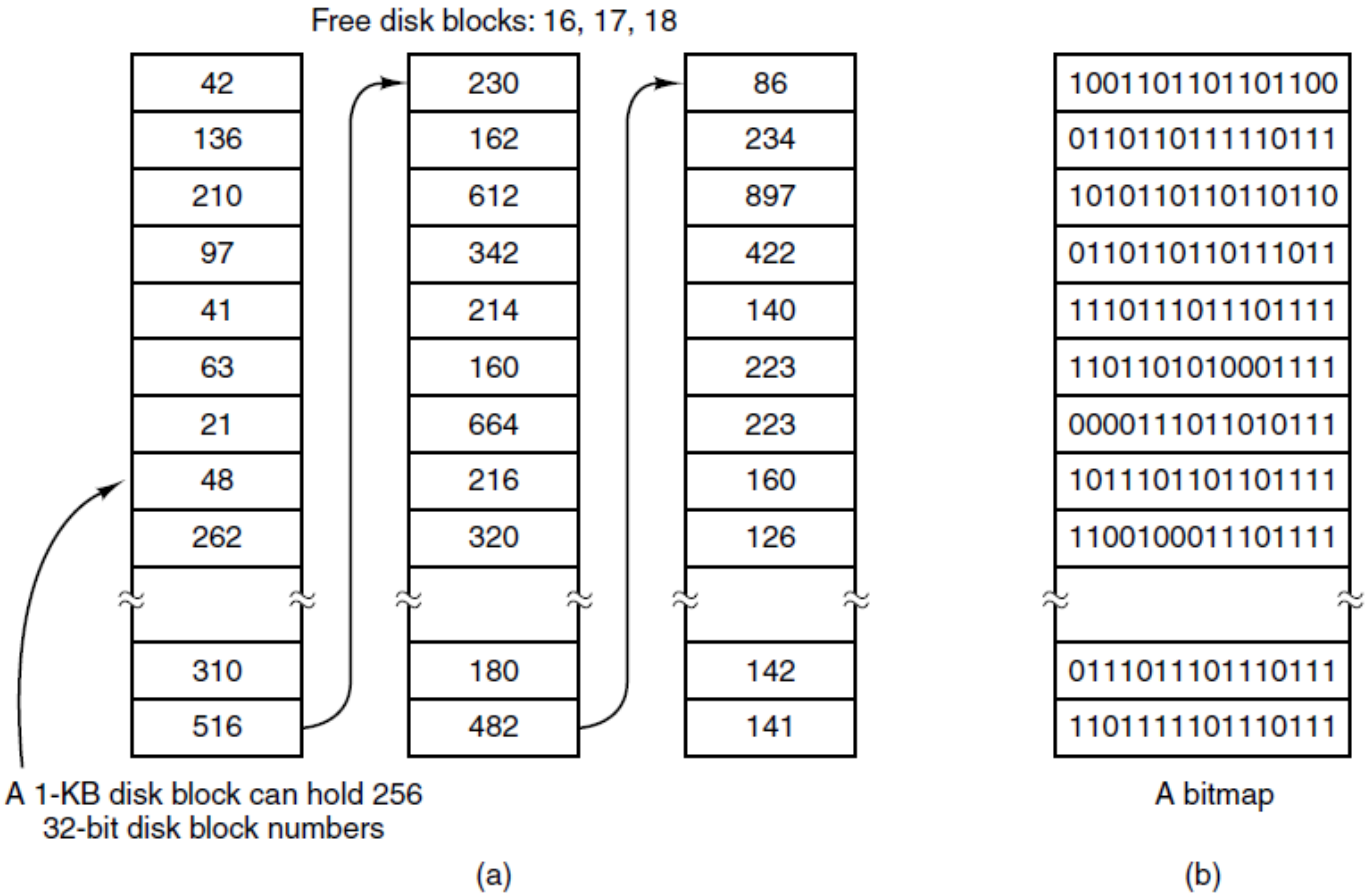
After the original user C **removes** the file from its directory

The UNIX V7 File System

The steps in **looking up** */usr/ast/mbox* using directory entries and i-nodes **starting** from the root directory (assuming i-node 1 is being used for root directory)



Keeping Track of Free Blocks



Storing the free list on a linked list

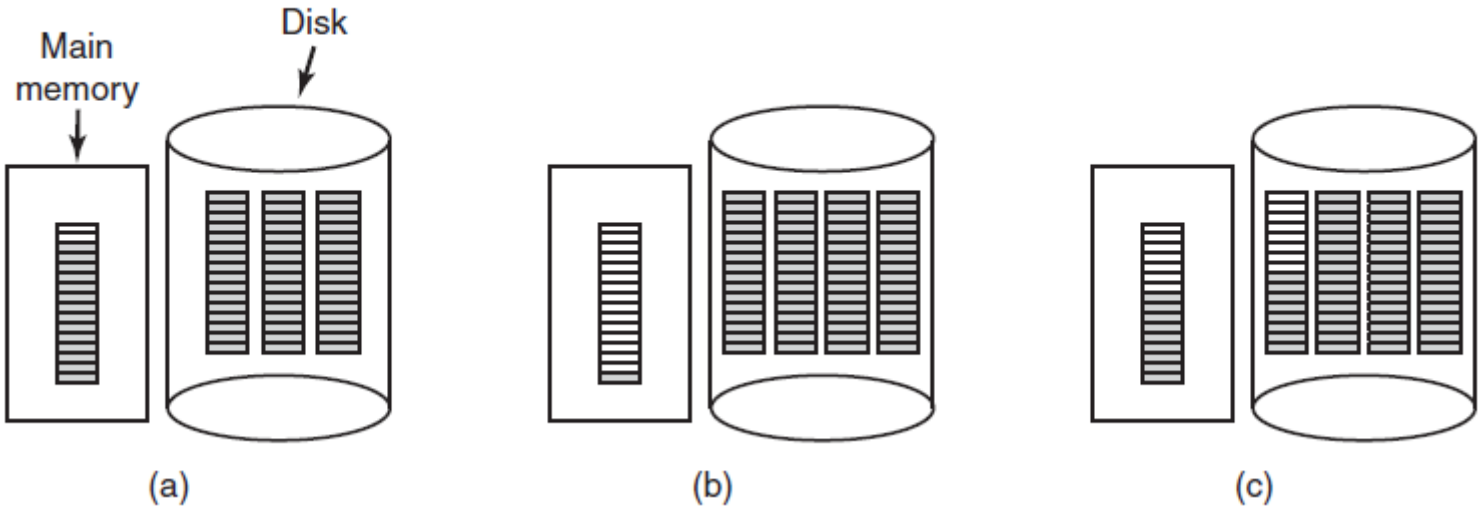
Storing the free list on a bitmap

Keeping Track of Free Blocks

To **enhance** performance of free block list

- One block of free block list is **kept** in memory
- When disk blocks are needed that are **taken** from the in memory list.
- If in memory free block list **becomes** empty a new disk block of free block list is **brought** into memory.
- When disks blocks are **freed** that are **added** to the in memory list.
- If the in memory list becomes **full** it is **written back** to disk.

Keeping Track of Free Blocks



An almost-full (has room for only 2 more pointers) in memory block and 3 blocks on disk.

Result of **freeing** a three-block file. Almost an empty in memory block and 4 disk blocks. Will go back to (a) if a three-block file is written back.

An alternative strategy for handling the three free blocks **avoiding** disk write and read.

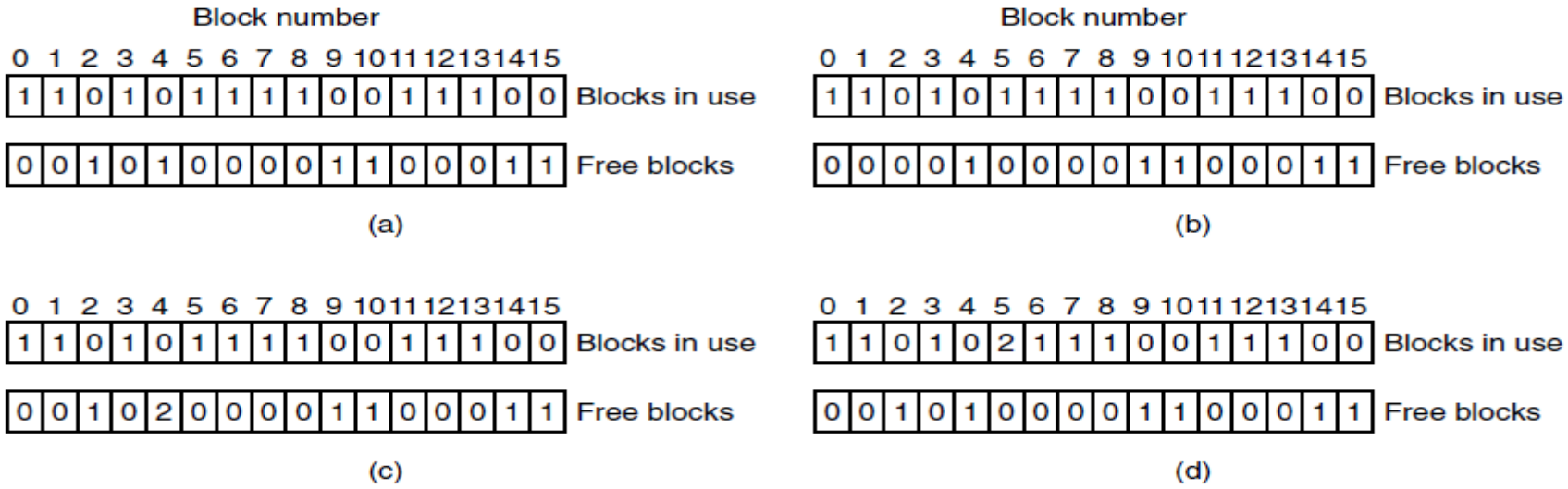
File System Consistency

Two kinds of consistency checks

- Block Consistency
- File Consistency

File System Consistency

In block consistency check, each block is **accounted** for how many times it is in use by the files and how many times it is in free block list.



File system states. (a) Consistent. (b) Missing block **2**. (c) Duplicate block **4** in free list. (d) Duplicate data block **5**.

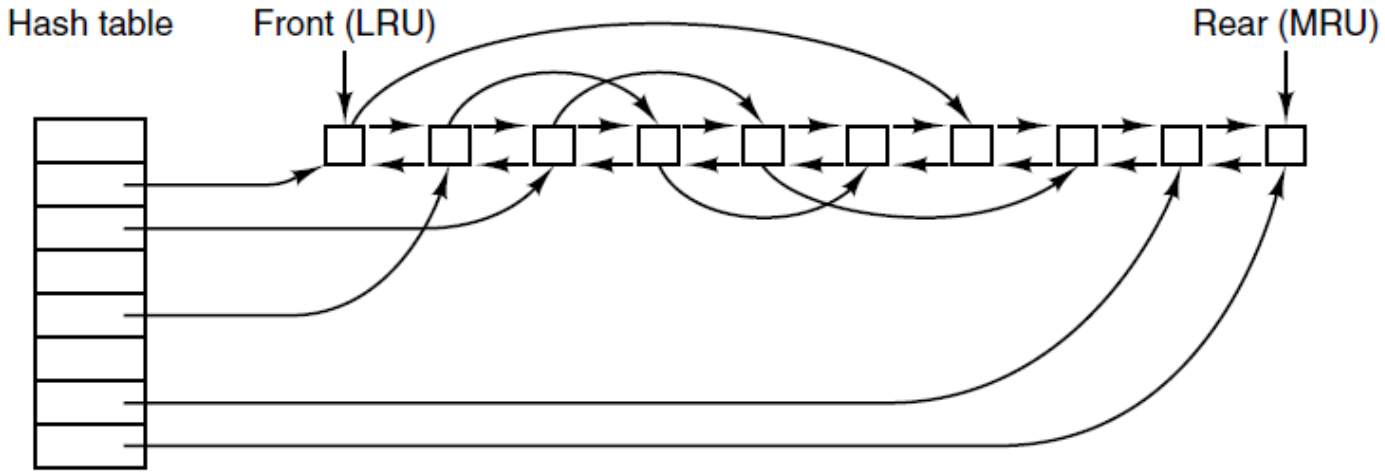
File System Consistency

- In file consistency, **check** all the directories starting from the root directory.
- **Count** the presence of each i-node in the directory entries of the whole file system.
- **Compare** the presence count against the link count of each i-node.
- If the link count is higher or lower **set** it to the presence count.

File System Caching

- A set of disk blocks are **kept** in memory in order to **reduce** the number of disk accesses.
- Against any read request, the cache blocks are **checked** first.
- If the block is **in** the cache, the request is handled **without** a disk access.
- If the block is **not in** the cache, it is **brought** into the cache first and then the request is handled from the cache.

File System Cache Lookup



- **Hash** the device and disk addresses and **lookup** into hash table to find a disk block in the cache.
- All the blocks with the same hash value are **chained** together on a linked list.

File System Cache Replacement: LRU

- In **addition** to the collision chains starting from hash table there is an additional bidirectional list running through all the cache blocks.
- The least recently used block on the front of this list and the most recently used block at the end.
- When a block is **referenced**, it can be **removed** from its position on the bidirectional list and **put** at the end.
- Cache from the block at the front of the list is **reclaimed** when necessary.
- Some blocks are **rarely referenced** two times within a short interval that leads to a **modified** LRU scheme.

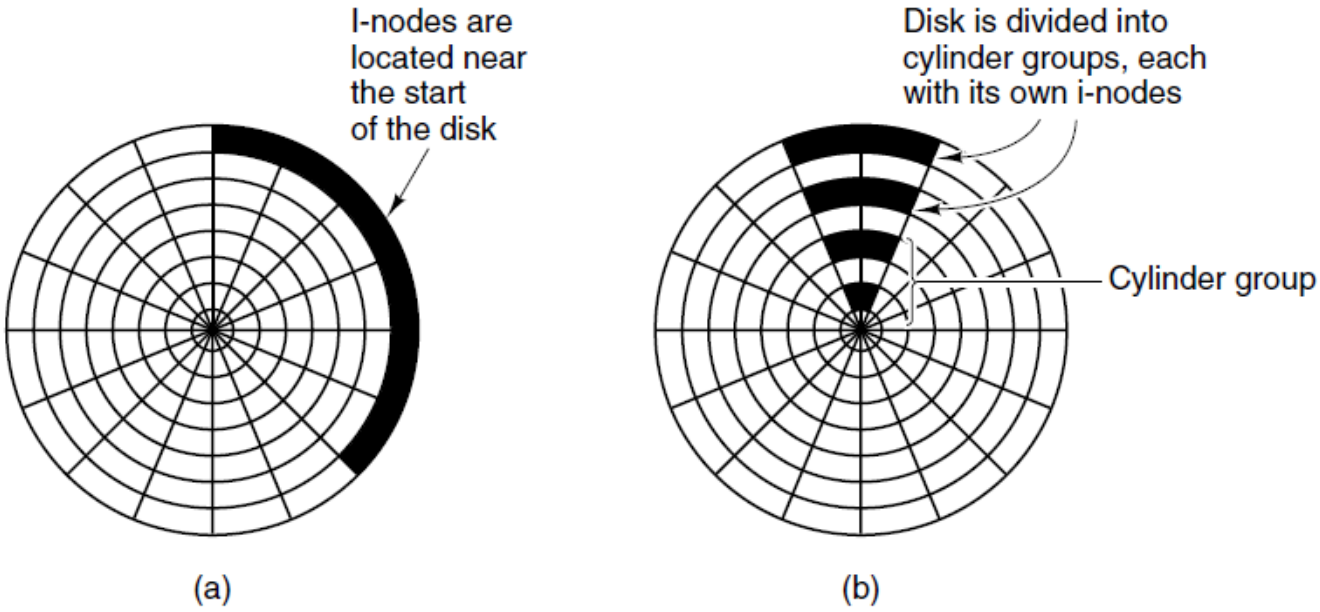
File System Cache Replacement

- **Modified** LRU scheme takes **two factors** into account:
  1. Is the block likely to be **needed again soon**?
  2. Is the block **essential** to the **consistency** of the file system?
- Blocks are **classified** into following classes
  - i-node
  - Indirect
  - Directory
  - Full data
  - Partially full data

File System Caching

- Block that will not be needed soon, e.g., full data block, goes to the front of the list and its cache will be **reclaimed** soon whenever necessary.
- Block that will be needed soon , e.g., partially full data block, goes to the rear of the list its cache will **not** be **reclaimed** soon.
- Block that is **necessary** for the file system consistency, e.g., i-node, indirect, and directory, and have been **modified** should be **written** into the disk immediately irrespective of its position in the list.

Reducing Disk Arm Motion



I-nodes **placed** at the **start** of the disk.

- Disk is **divided** into cylinder groups, each with its **own** blocks and i-nodes.
- When a new file is created it can **choose** any i-node from the free i-node list but **choose** the data blocks from the same cylinder group

Summary

- File Abstraction
- File Concepts
- Directory Concepts
- Disk Partitions and File System Layout
- Disk Block Allocation
- Free Disk Block Management
- File System Performance

Next

I/O Systems

- I/O Concepts
- I/O Software
- I/O Software Layers