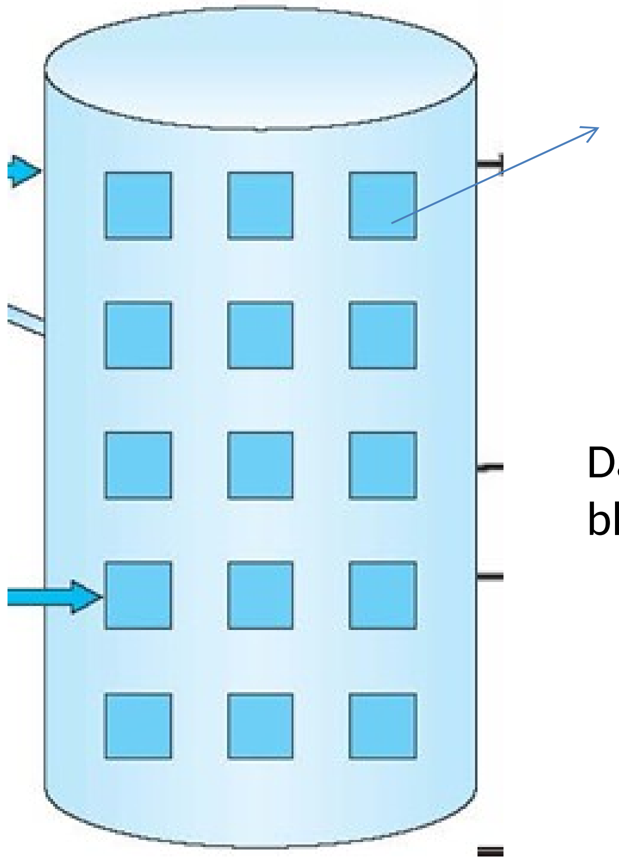
File Management

Objectives

- ***File Systems* : *F***ile system structure, allocation methods (contiguous, linked, indexed), free-space management (bit vector, linked list, grouping), directory implementation (linear list, hash table)

- ***Disk Management* :** disk structure, disk scheduling (FCFS, SSTF, SCAN,C-SCAN) , disk reliability, disk formatting, boot block, bad blocks.



Disk blocks

Data stored within the block

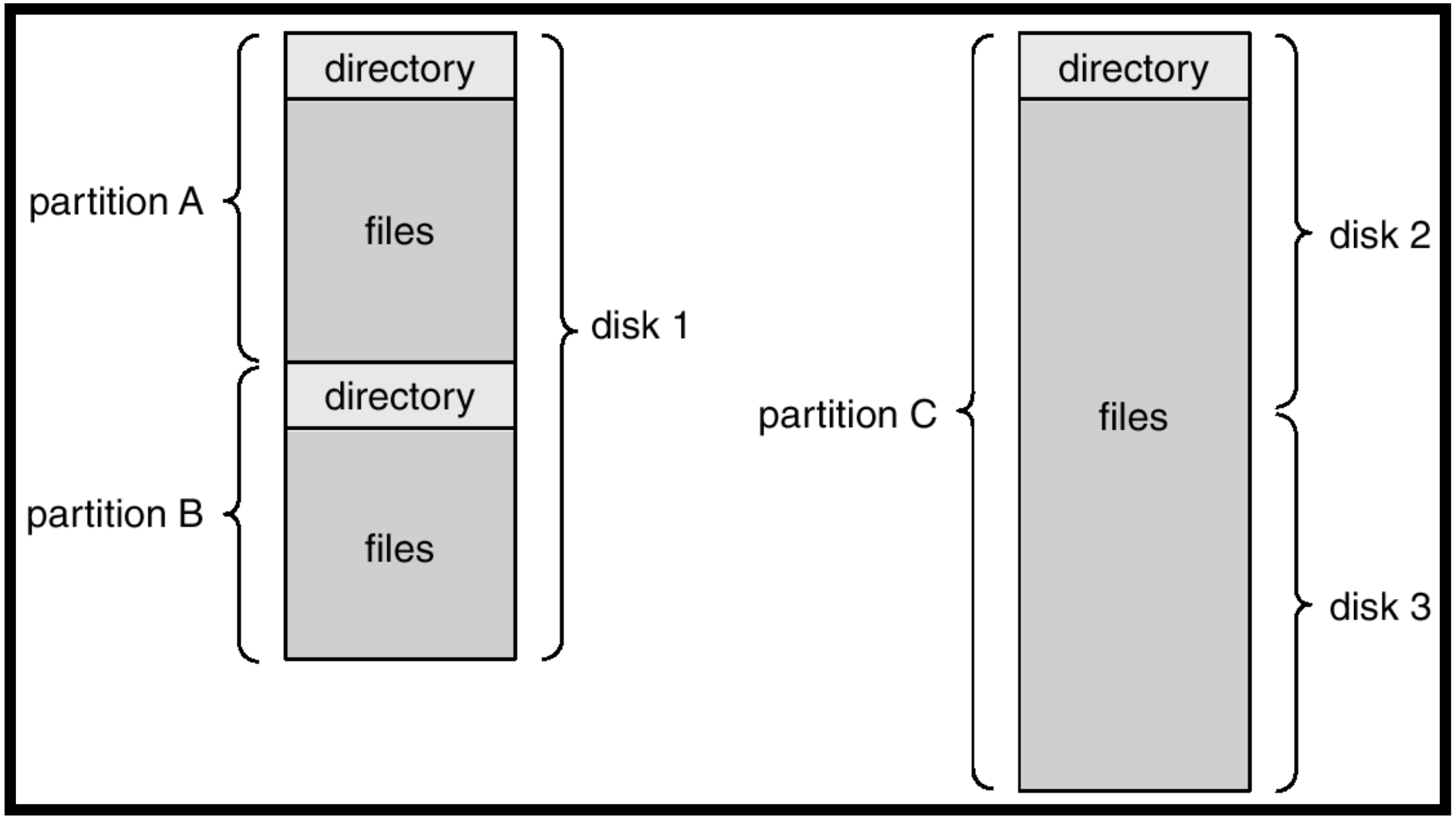
File-System Structure

- File system
  - Provide efficient and convenient access to disk
  - Easy access to the data (store, locate and retrieve)
- Two aspects
  - User’s view
    - Define files/attributes, operations, directory
  - Implementing file system
    - Data structures and algorithms to map logical view to physical one

Disk Layout

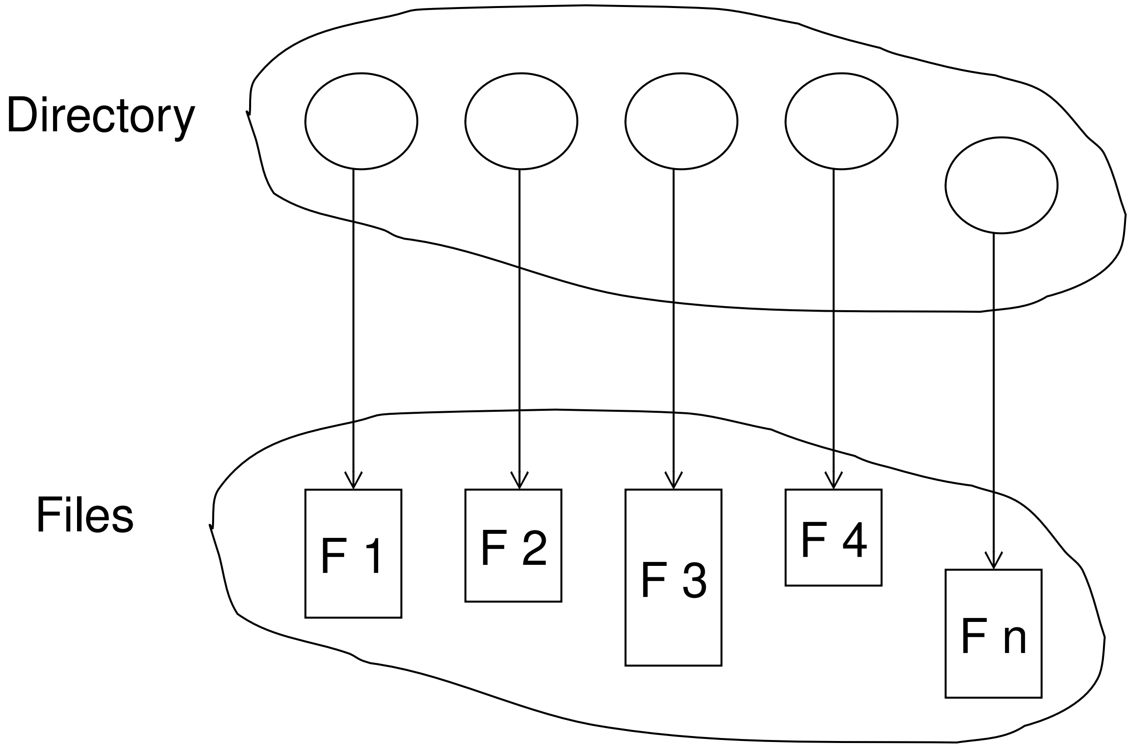
- Files stored on disks.
- Disks broken up into one or more partitions, with separate file system on each partition

A Typical File-system Organization

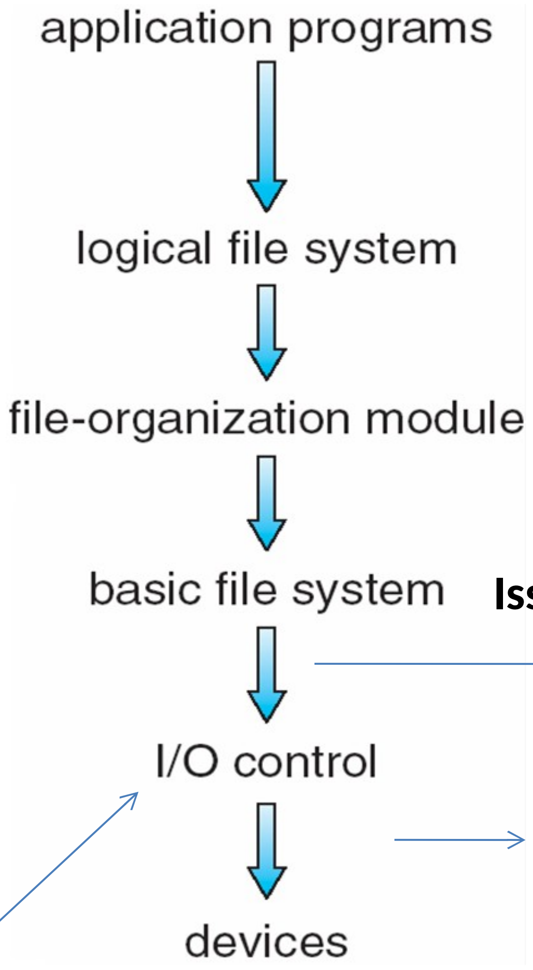


Directory Structure

- A collection of nodes containing information about all files



Layered File System



Manages **FCB**

Translates logical to physical blocks

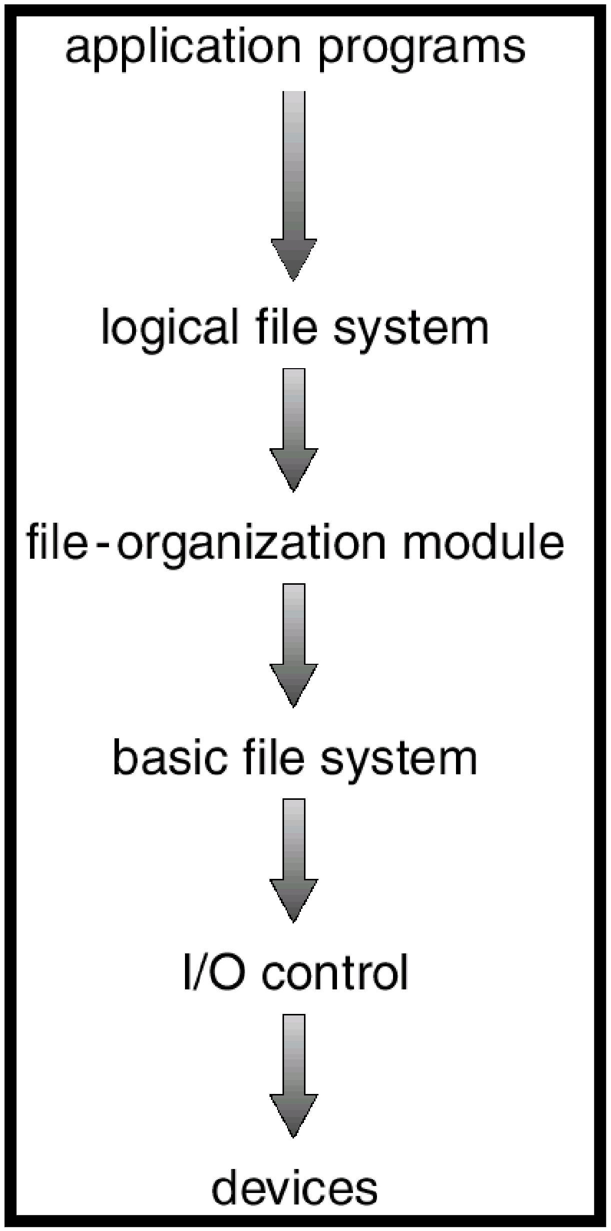
Issue commands

R/W Physical block (cylinder, track, sector)

Hardware specific instruction

- Each level uses the feature of low level
- Create new features for higher level

Device driver, transfer information between memory/disk



Manages meta date about files, file organization, directory structure, file control blocks, etc.

Mapping of logical block# (0..n) to physical block# (sector, track #, etc), free space mgmt

Issues generic commands to device drive to R/W physical blocks on disk

Device drivers, interrupt service routines, etc

File System Layers

I/O control layer consists of 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 Issues commands with physical block address (sector, track)

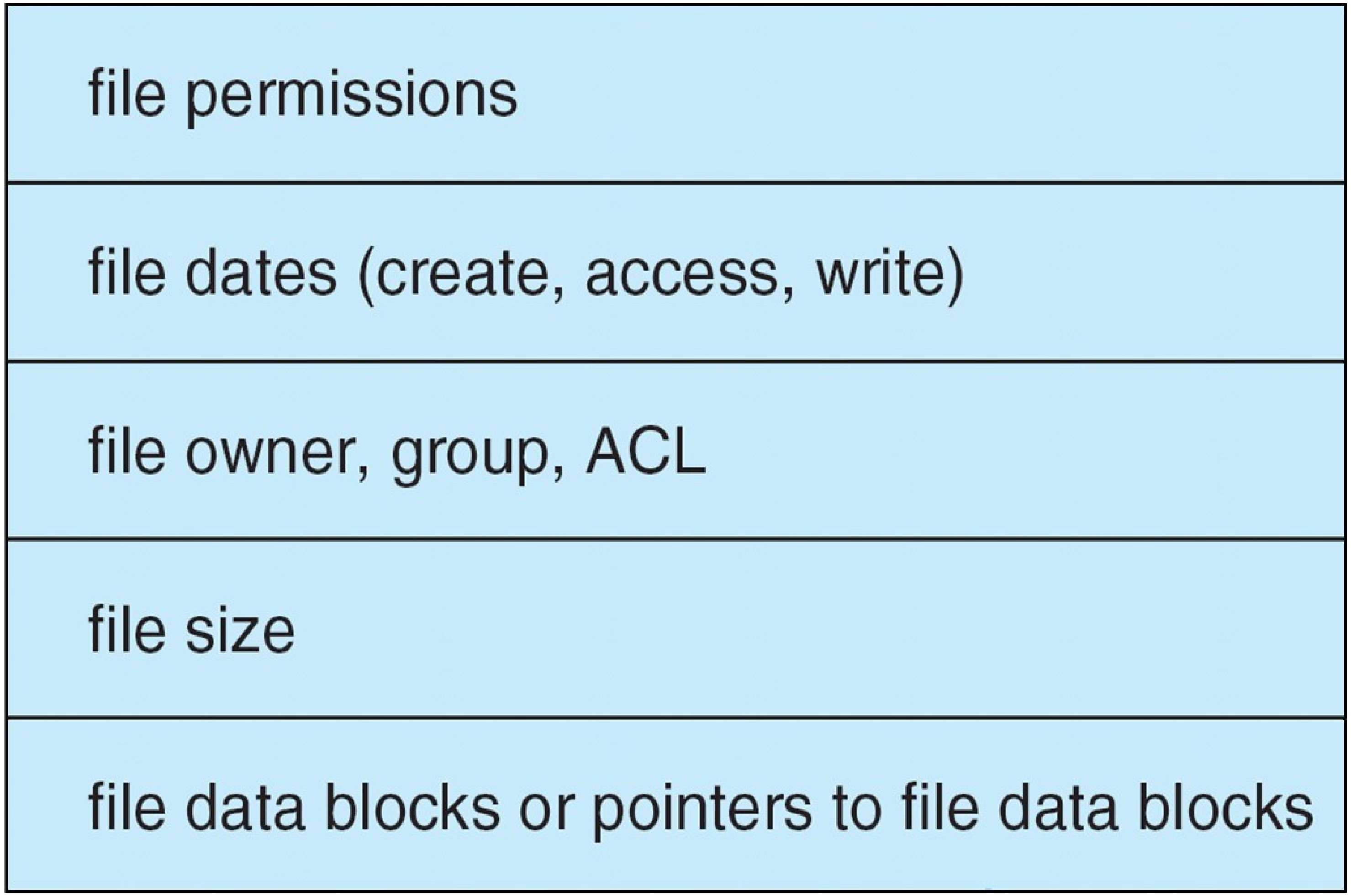
File organization module understands files, logical address, and physical blocks

- 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 number, file handle, location by maintaining file control blocks (**inodes** in Unix)
  - Directory management
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
  - Shares the I/O control and basic FS
- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD, Linux has more than 40 types, with **extended file system** ext2 and ext3 leading; plus distributed file systems, etc)

A Typical File Control Block



file permissions

file dates (create, access, write)

file owner, group, ACL

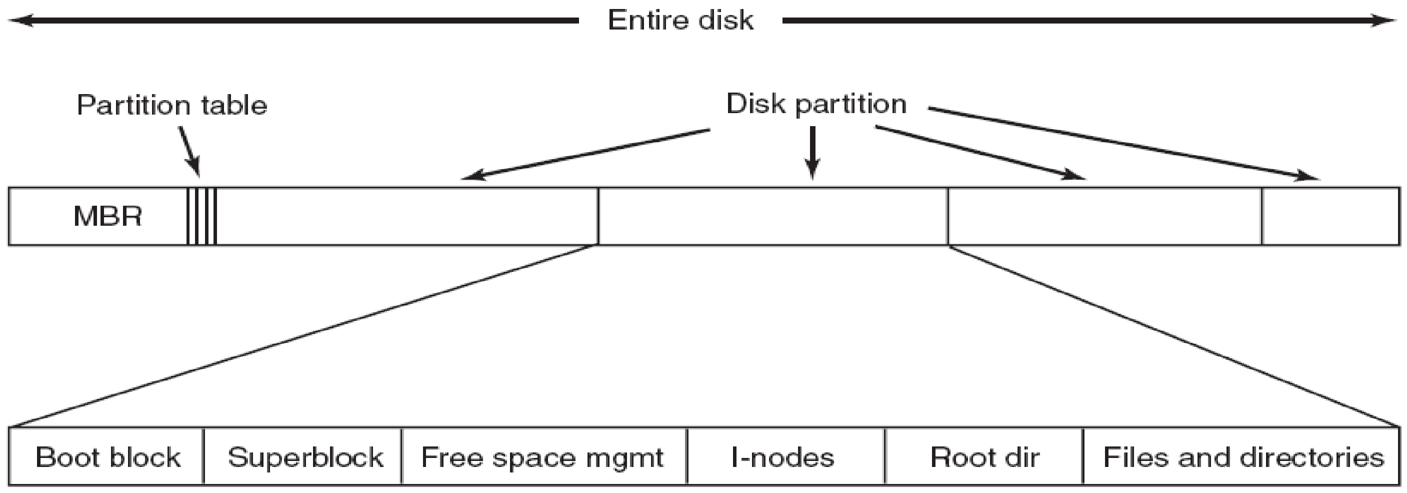
file size

file data blocks or pointers to file data blocks

Disk Layout

- Files stored on disks. Disks broken up into one or more partitions, with separate file system on each partition
- Sector 0 of disk is the Master Boot Record
- Used to boot the computer
- End of MBR has partition table. Has starting and ending addresses of each partition.
- One of the partitions is marked active in the master boot table

Disk Layout


Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories

File system data structures

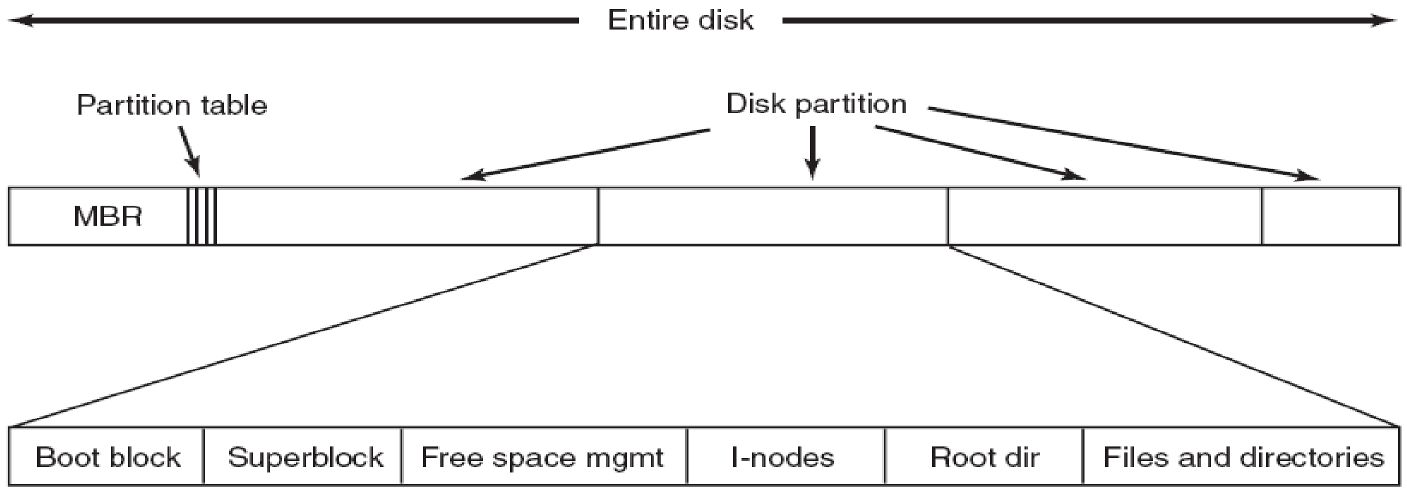
- On disk

- In memory

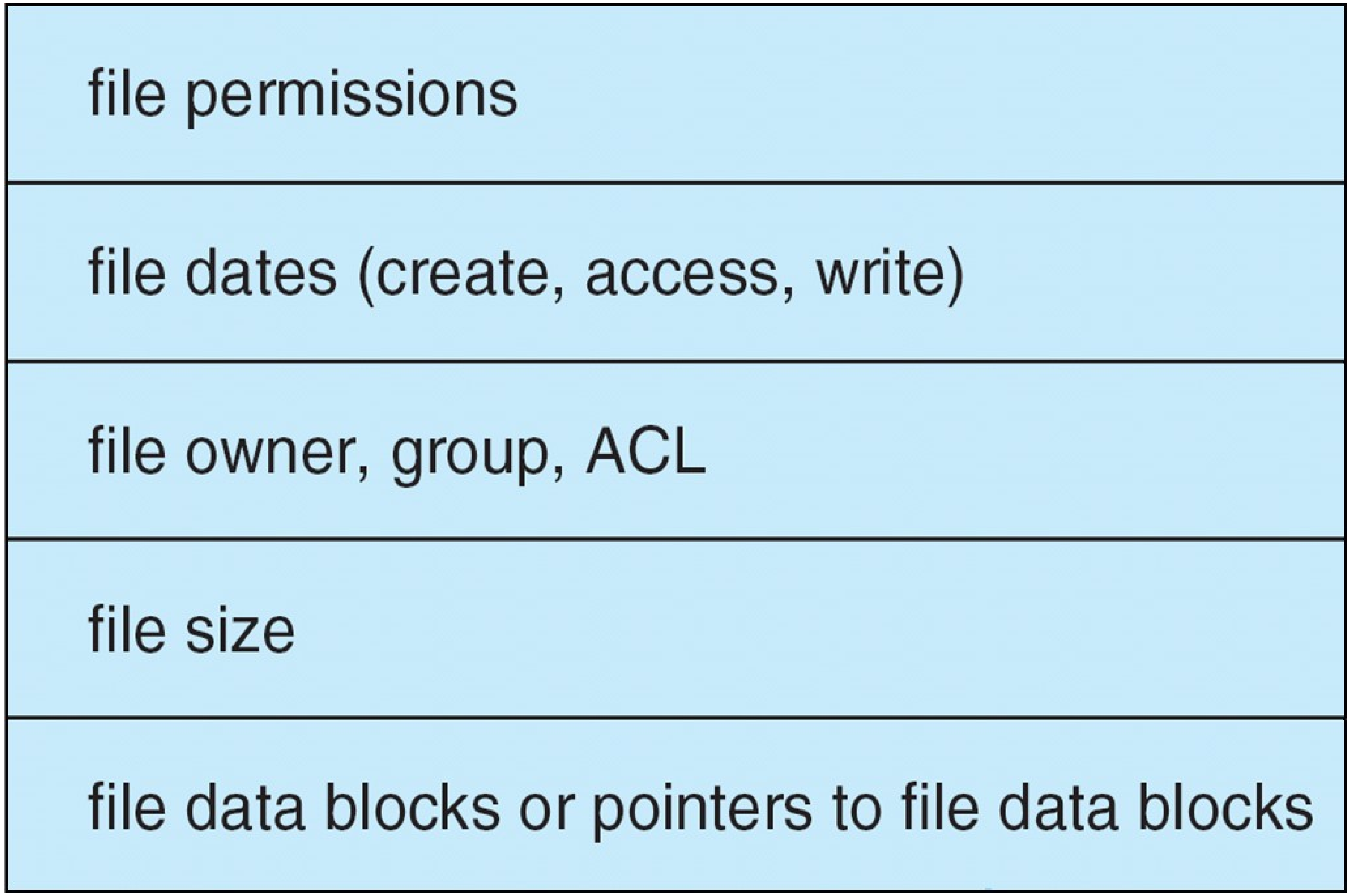
Disk Layout

- **Boot control block** contains info needed by system to boot OS from that partition
  - Needed if partition contains OS, usually first block of partition
- **Partition control block** (**superblock, master file table**) contains partition details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table
- Per-file **File Control Block (FCB)** contains many details about the file
  - Inode number, permissions, size, dates
  - NTFS stores into in master file table using relational DB structures

Disk Layout


Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories

A Typical File Control Block



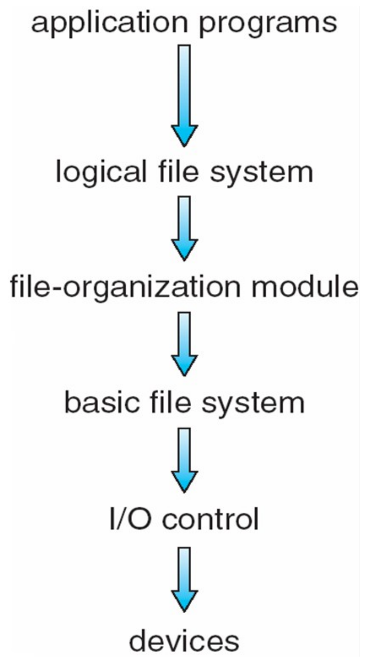
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

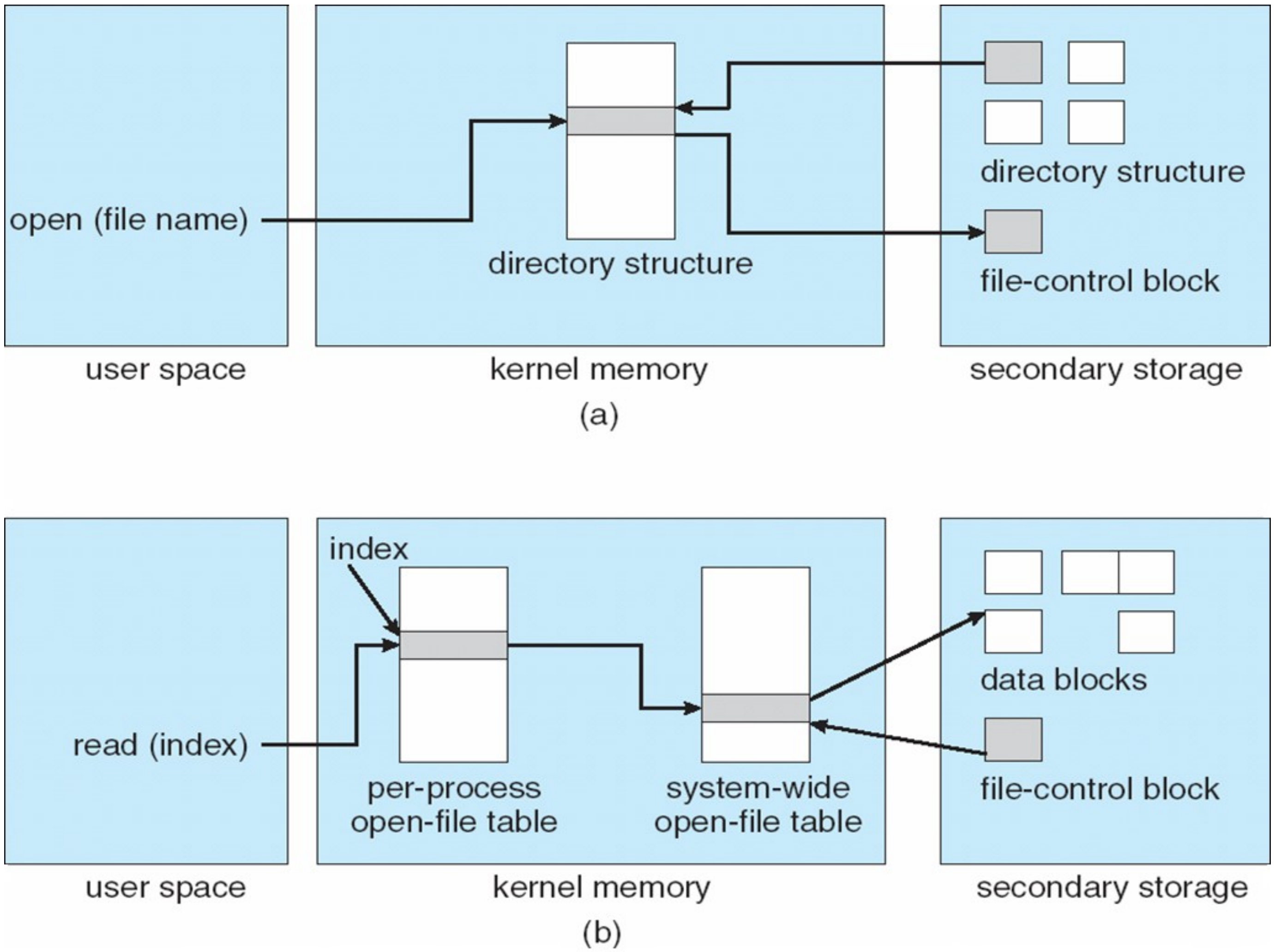
- In memory directory structure holds the directory information of recently accessed directories
- **System-wide-open file** contains a copy of FCB for each file
- **Per-process open file table**: contains pointer to appropriate entry in the **system wide open file table**

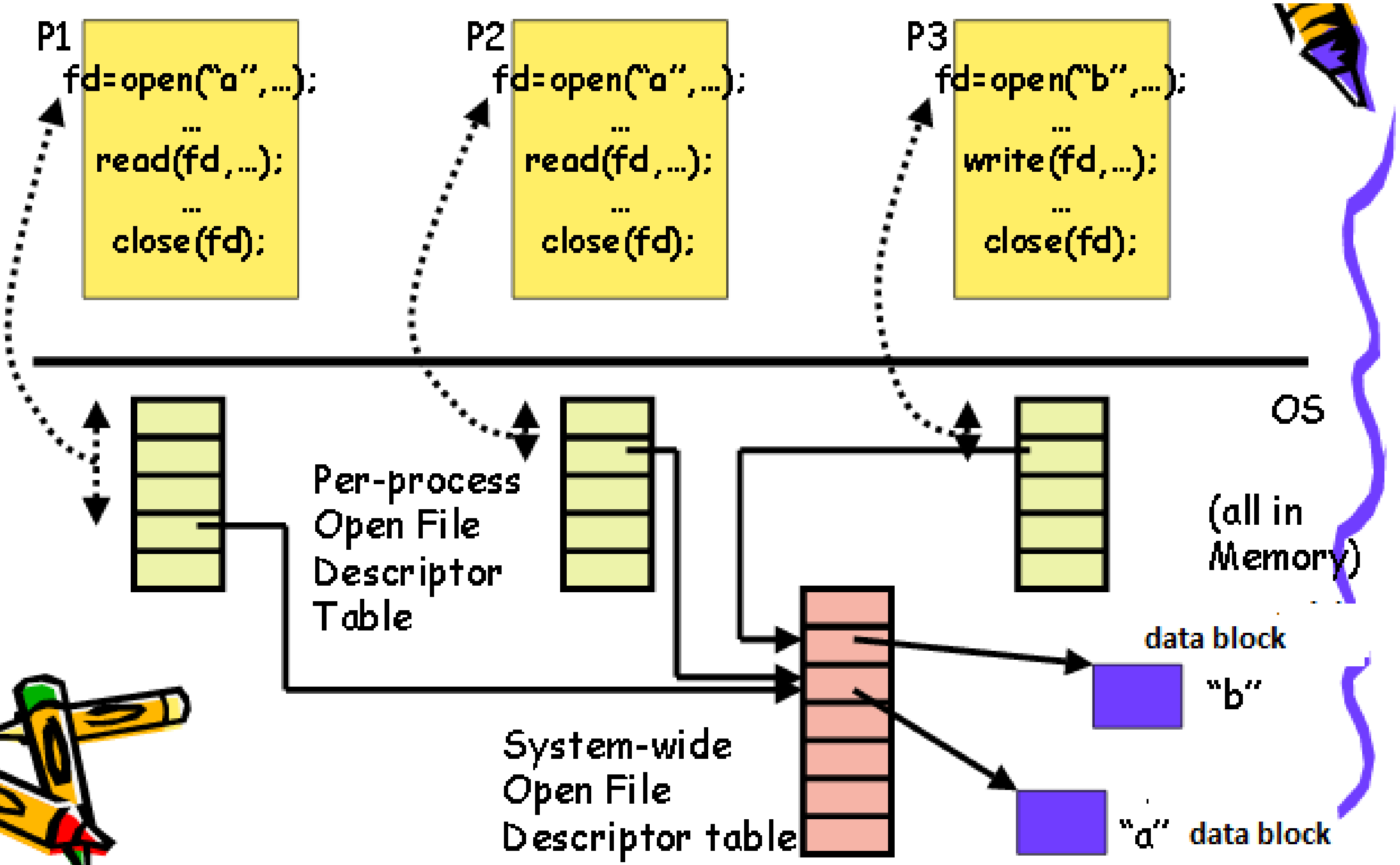
File handling

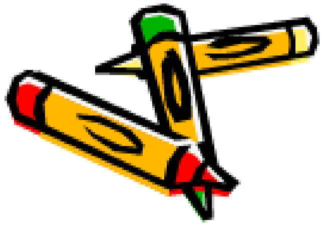
- Create a new file
  - Application program calls the logical file system
- Logical file system
  - Allocates a new FCB
  - Reads the appropriate directory into memory
  - Updates directory with new filename and FCB
  - Write it back to disk
- Using the file (I/O)
  - Open() [filename]
  - Directory is searched
  - FCB is copied into system wide open file table
  - Entry made to Per-process open file table
    - Pointer to the system table entry
    - File descriptor#



In-Memory File System Structures




P1
fd=open("a",...);
...
read(fd,...);
...
close(fd);
P2
fd=open("a",...);
...
read(fd,...);
...
close(fd);
P3
fd=open("b",...);
...
write(fd,...);
...
close(fd);
OS
Per-process Open File Descriptor Table
(all in Memory)
data block
"b"
System-wide Open File Descriptor table
"a" data block



- A process closes a file
  - Per process table entry removed
  - System table count decremented
- All processes closed the file
  - Updated file info is copied back to disk
  - System wide open file table entry removed

Allocating Blocks to files

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

- Most important implementation issue
- Methods
  - Contiguous allocation
  - Linked list allocation
  - Linked list using table
  - I-nodes

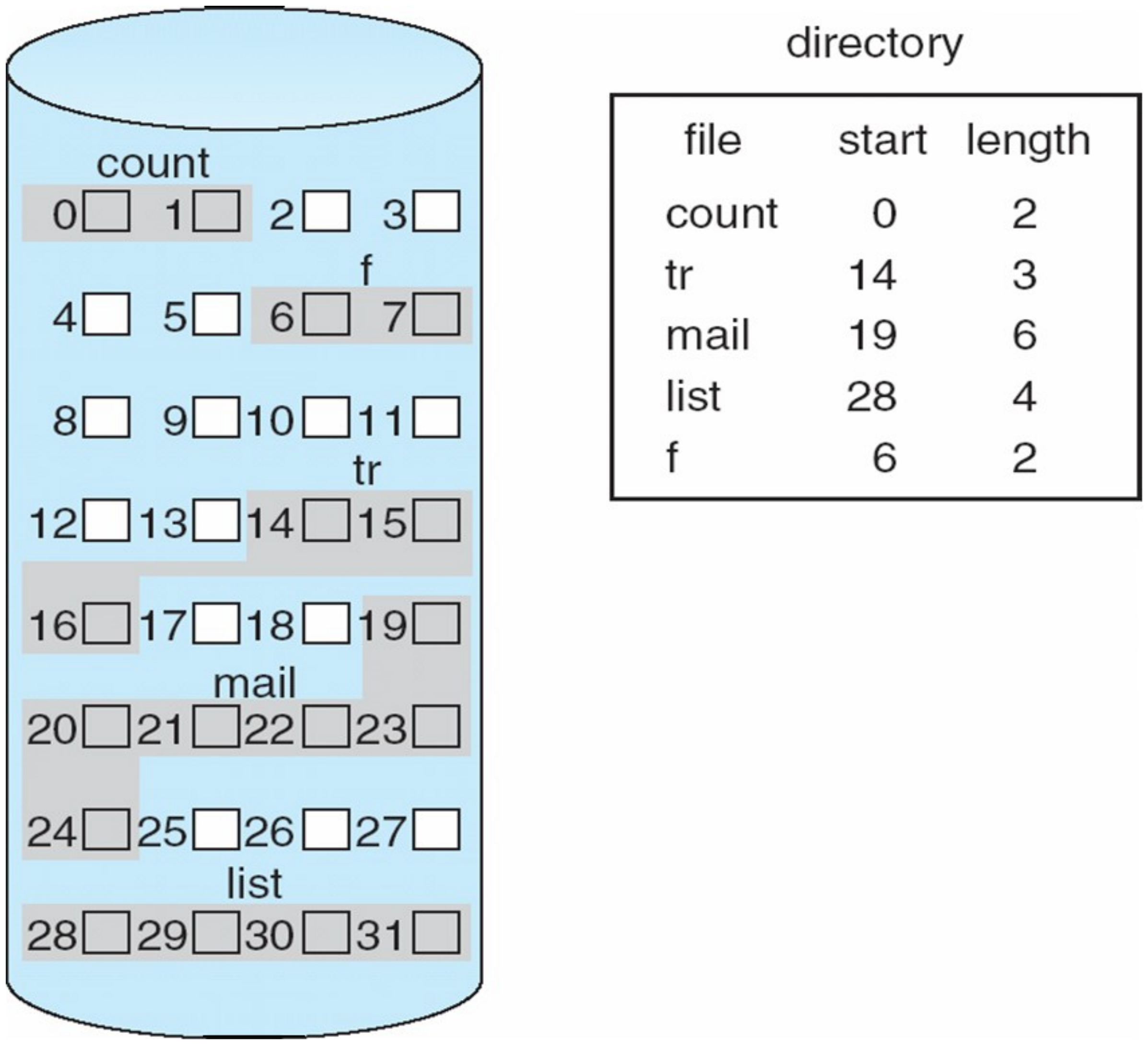
Allocation Methods - Contiguous

- **Contiguous allocation** – each file occupies set of contiguous blocks
- Blocks are allocated b, b+1, b+2,…….
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required (directory)

- Easy to implement
- Read performance is great. Only need one seek to locate the first block in the file. The rest is easy.

- Accessing file is easy
  - Minimum disk head movement
  - Sequential and direct access

Contiguous Allocation of Disk Space



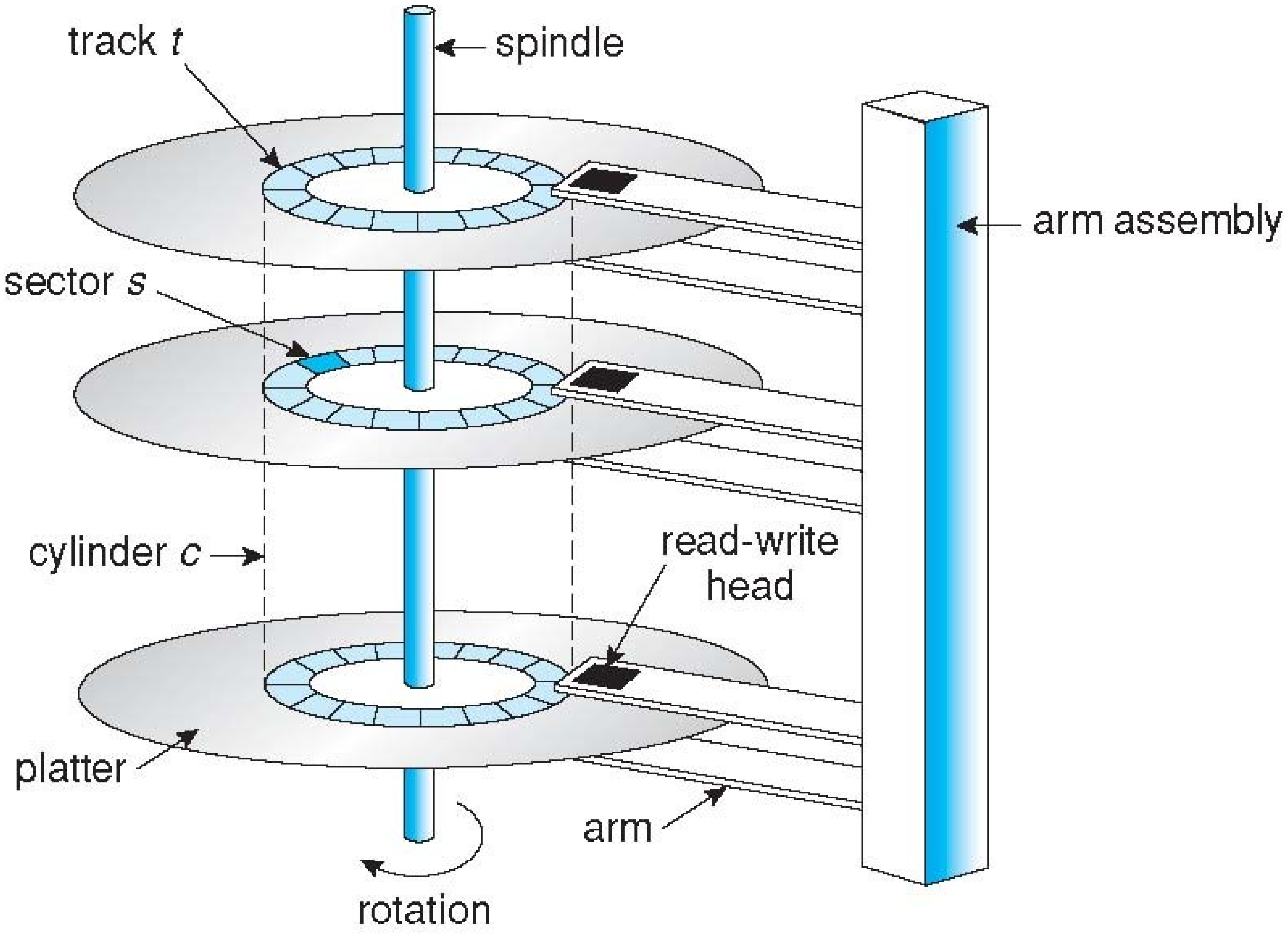
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- Problems
- Finding space for file
    - Satisfy the request of size n from the list of holes
    - External fragmentation
        - Need for **compaction routine**
        - **off-line** (**downtime**) or **on-line**

- Do not know the file size a priori
    - Terminate and restart
    - Overestimate
    - Copy it in a larger hole
    - Allocate new contiguous space (Extent)

Moving-head Disk Mechanism



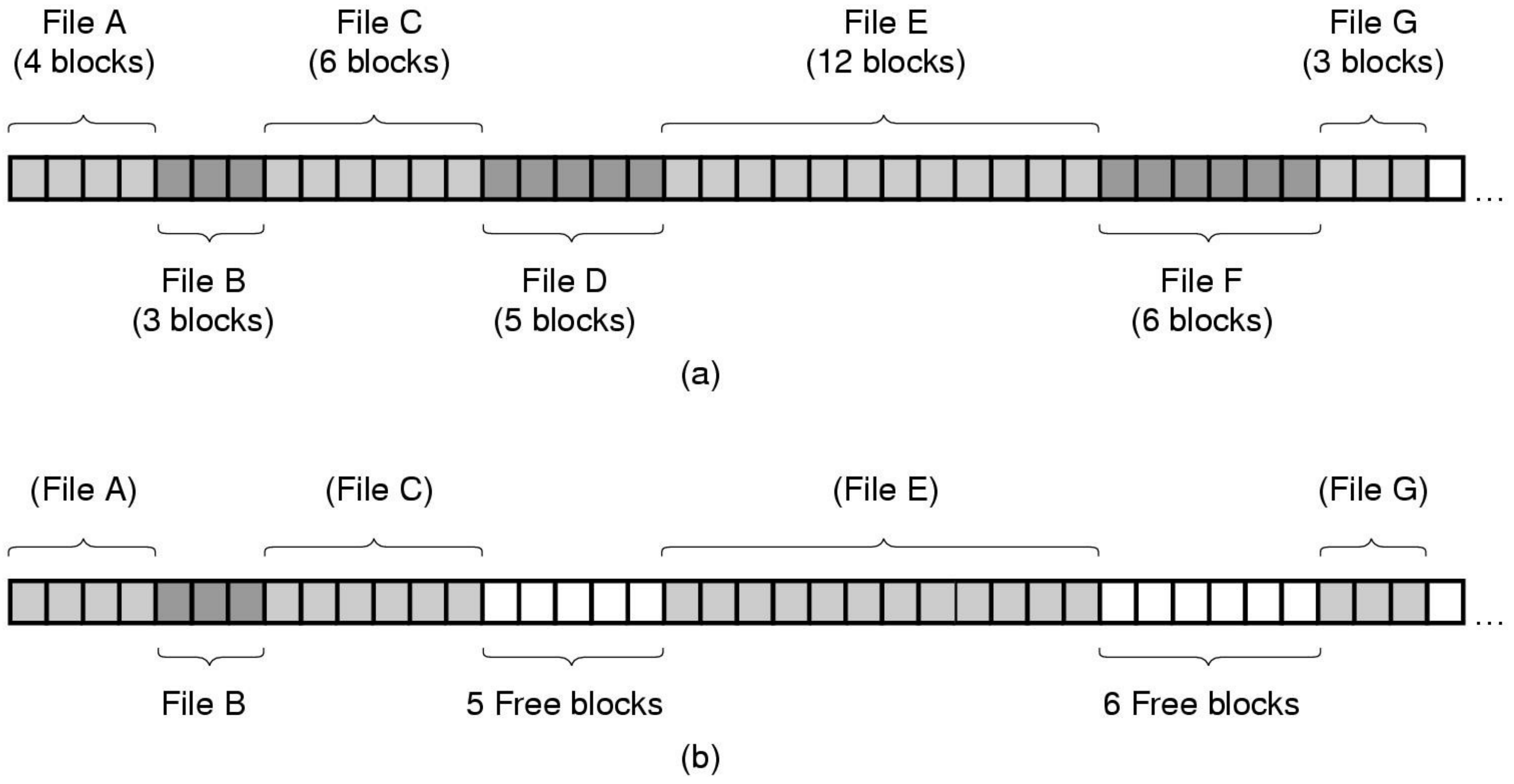
Extent-Based Systems

- Many newer file systems (i.e., 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 allocation
  - A file consists of one or more extents

Contiguous Allocation

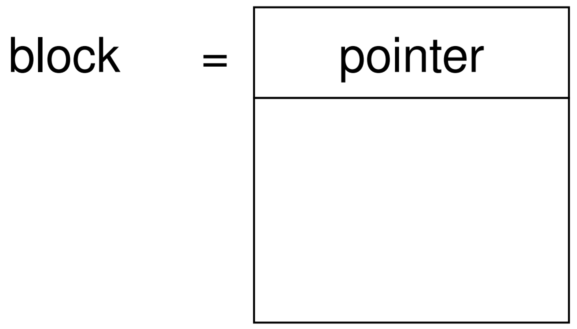


(a) Contiguous allocation of disk space for 7 files.

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

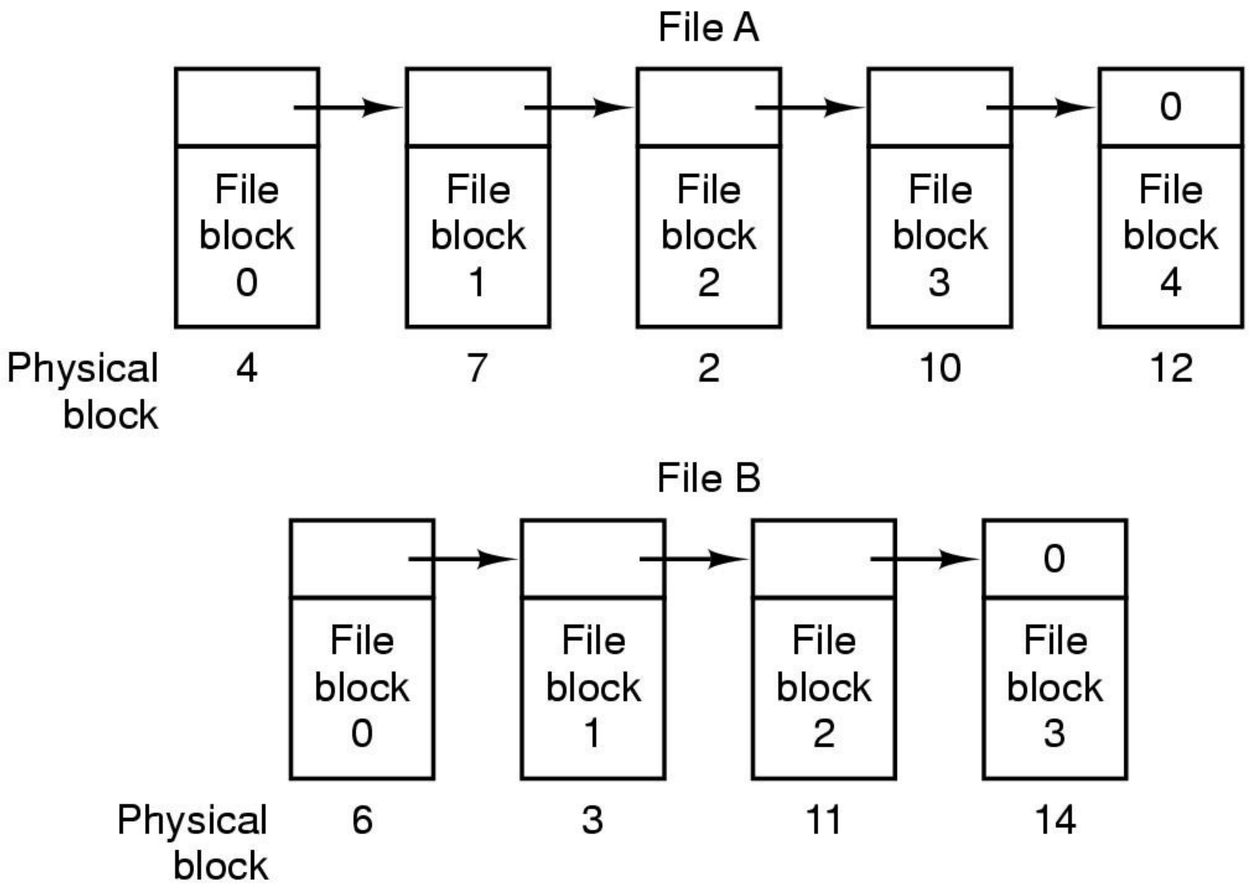
Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk



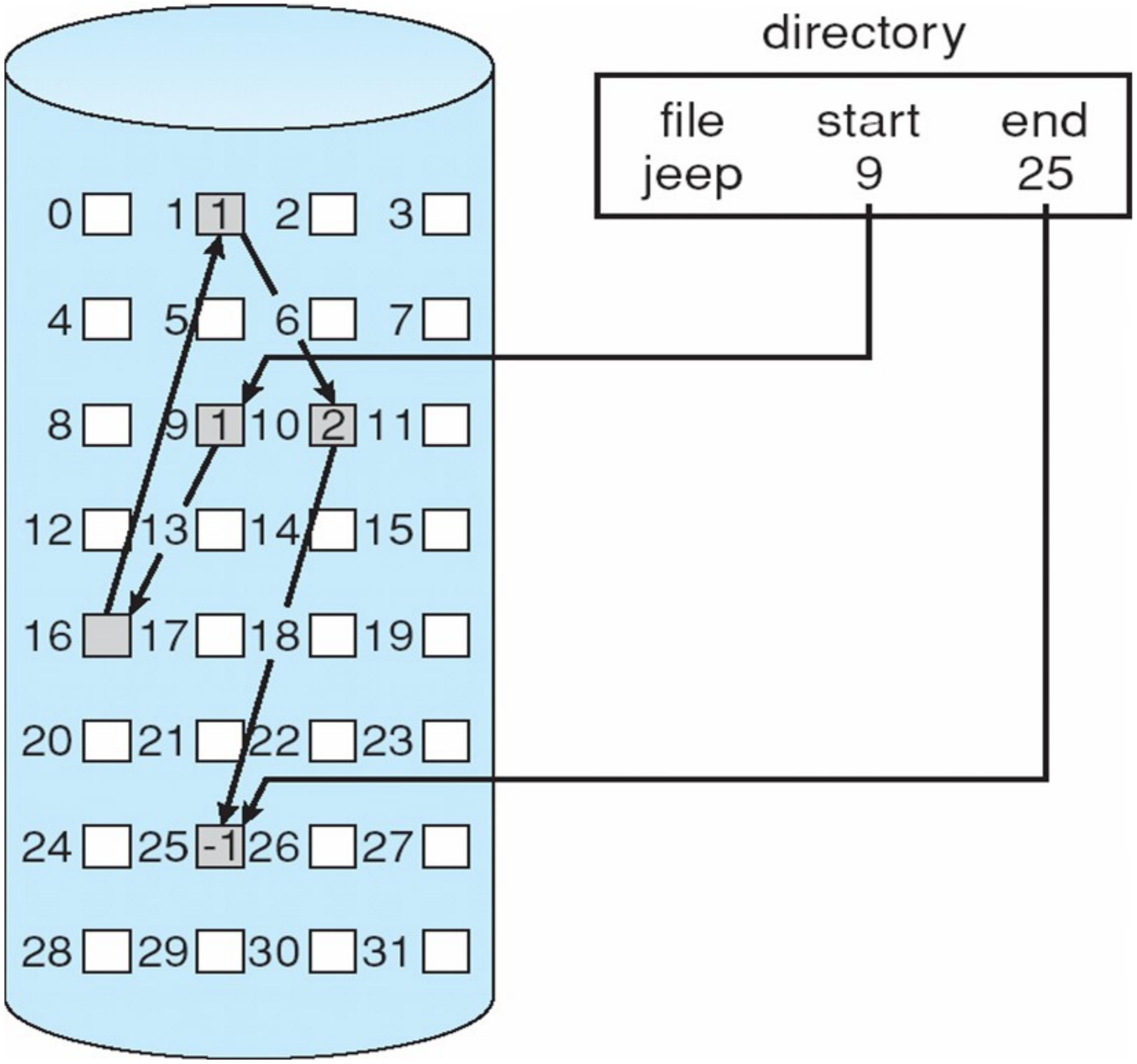
- Each block contains pointer to next block
- File ends at nil pointer

Linked List Allocation



Storing a file as a linked list of disk blocks.

Linked Allocation



Linked Allocation

- Free blocks are arranged from the free space management
- No external fragmentation
- Files can continue to grow

Disadvantage

1. Effective only for sequential access
   Random/direct access (i-th block) is difficult

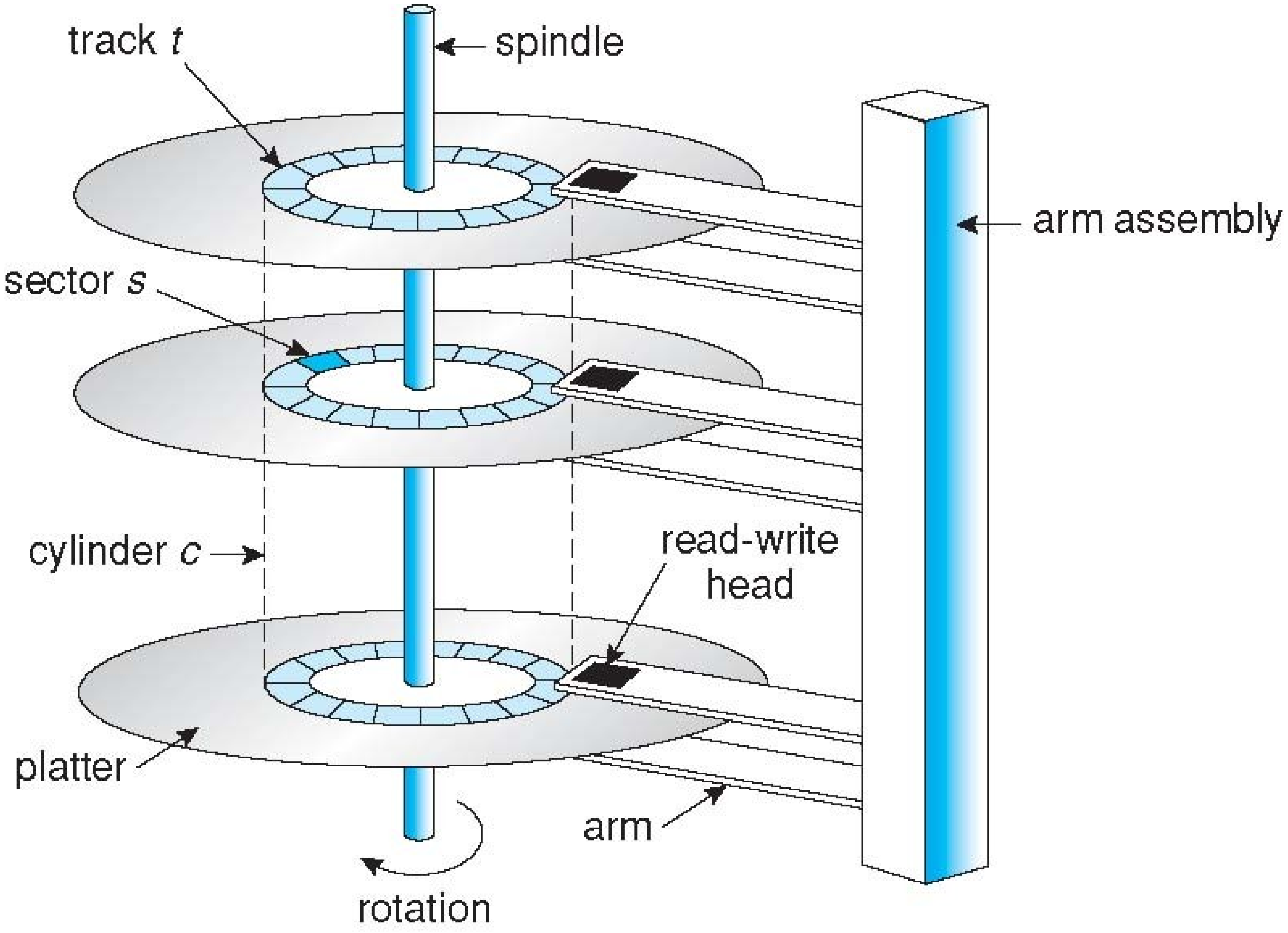
2. Space wastage
If block size 512 B
Disk address 4B
Effective size 508B

3. Reliability
Lost/damaged pointer
Bug in the OS software and disk hardware failure

4. Poor performance

Solution: Clusters
- **Improves disk access time (head movement)**
- **Decreases the link space needed for block**
- **Internal fragmentation**

Moving-head Disk Mechanism



Linked Allocation

FAT (File Allocation Table) variation

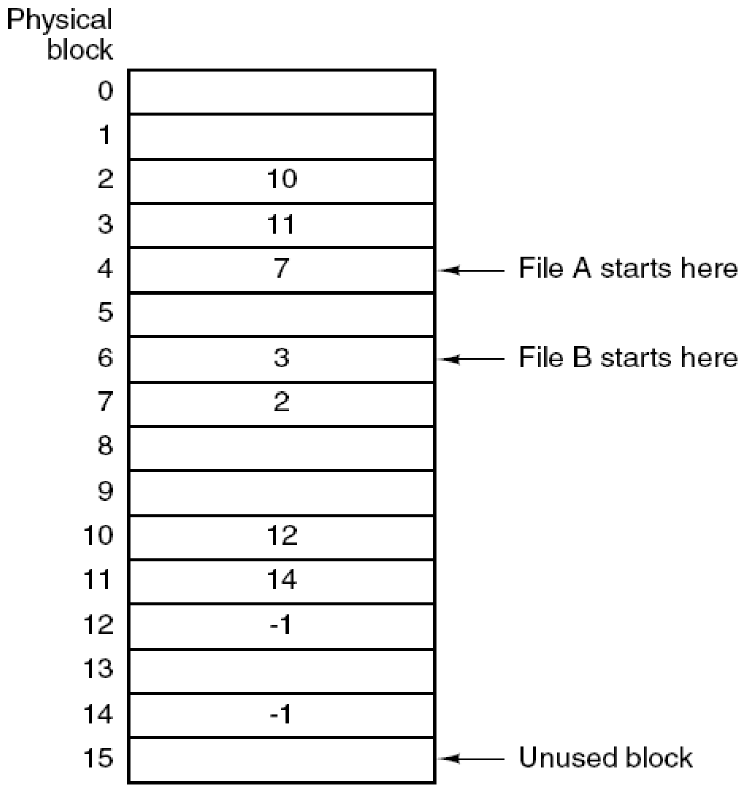
Beginning of partition has table, indexed by block number

Much like a linked list, but faster on disk and cacheable

New block allocation simple

Section of the disk at the beginning of the partition contains FAT table

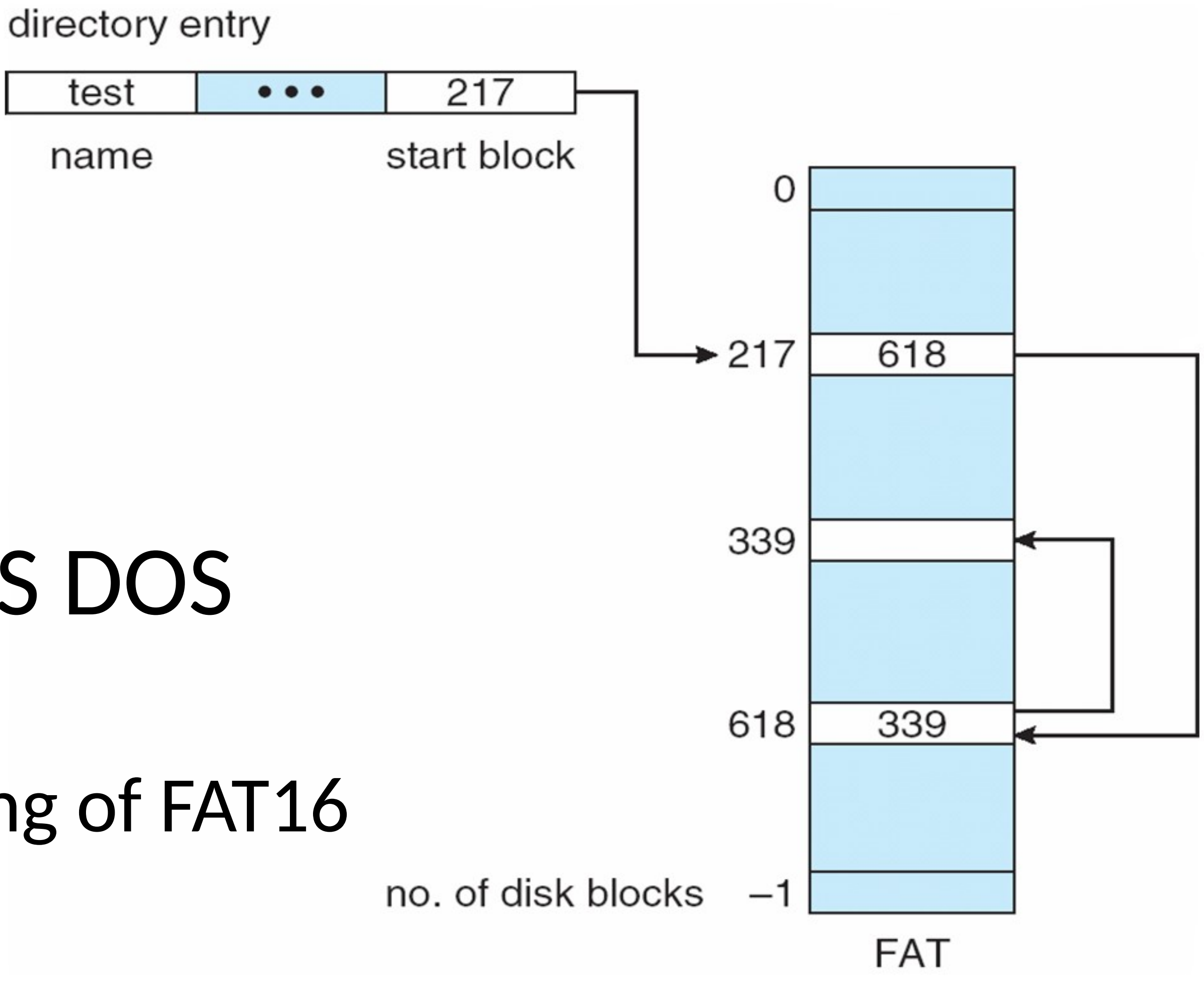
Unused blocks => 0



Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

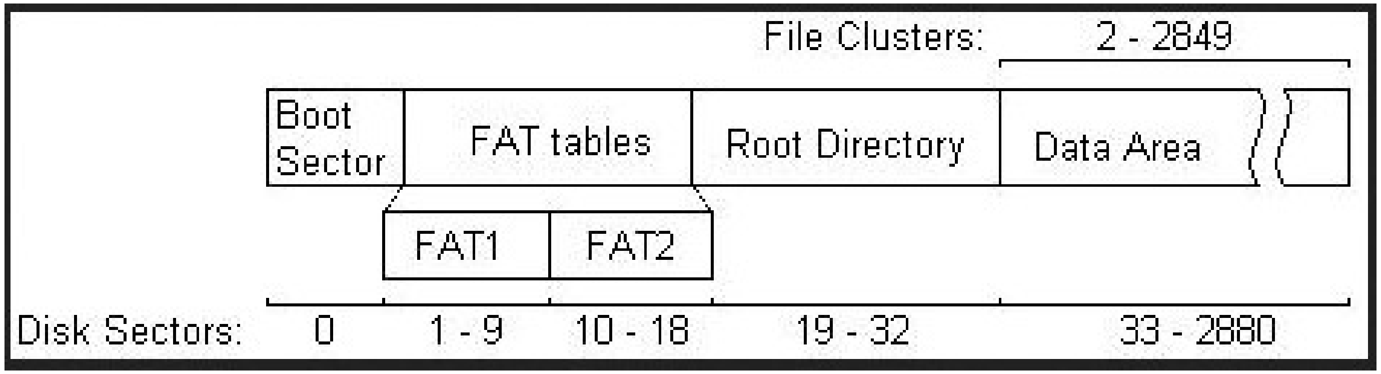
File-Allocation Table



MS DOS

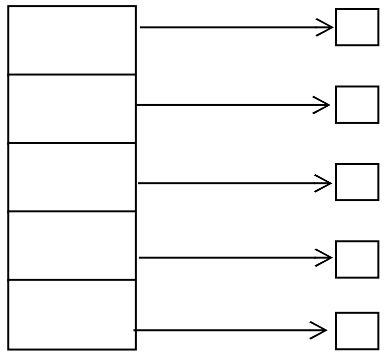
Caching of FAT16

DOS


File Clusters:
2 - 2849
Boot Sector
FAT tables
Root Directory
Data Area
FAT1
FAT2
Disk Sectors:
0
1 - 9
10 - 18
19 - 32
33 - 2880

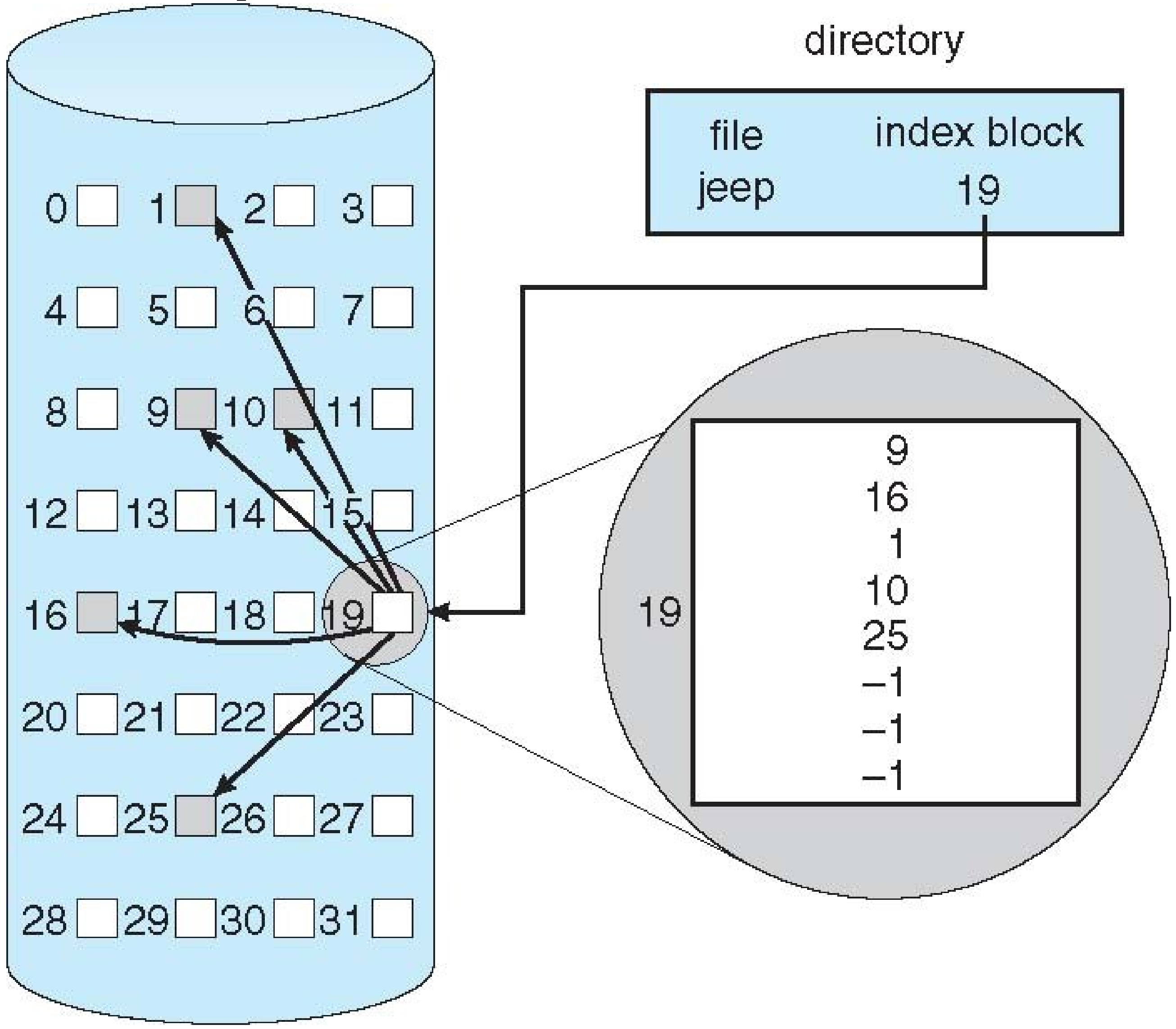
Allocation Methods - Indexed

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



index table

Example of Indexed Allocation



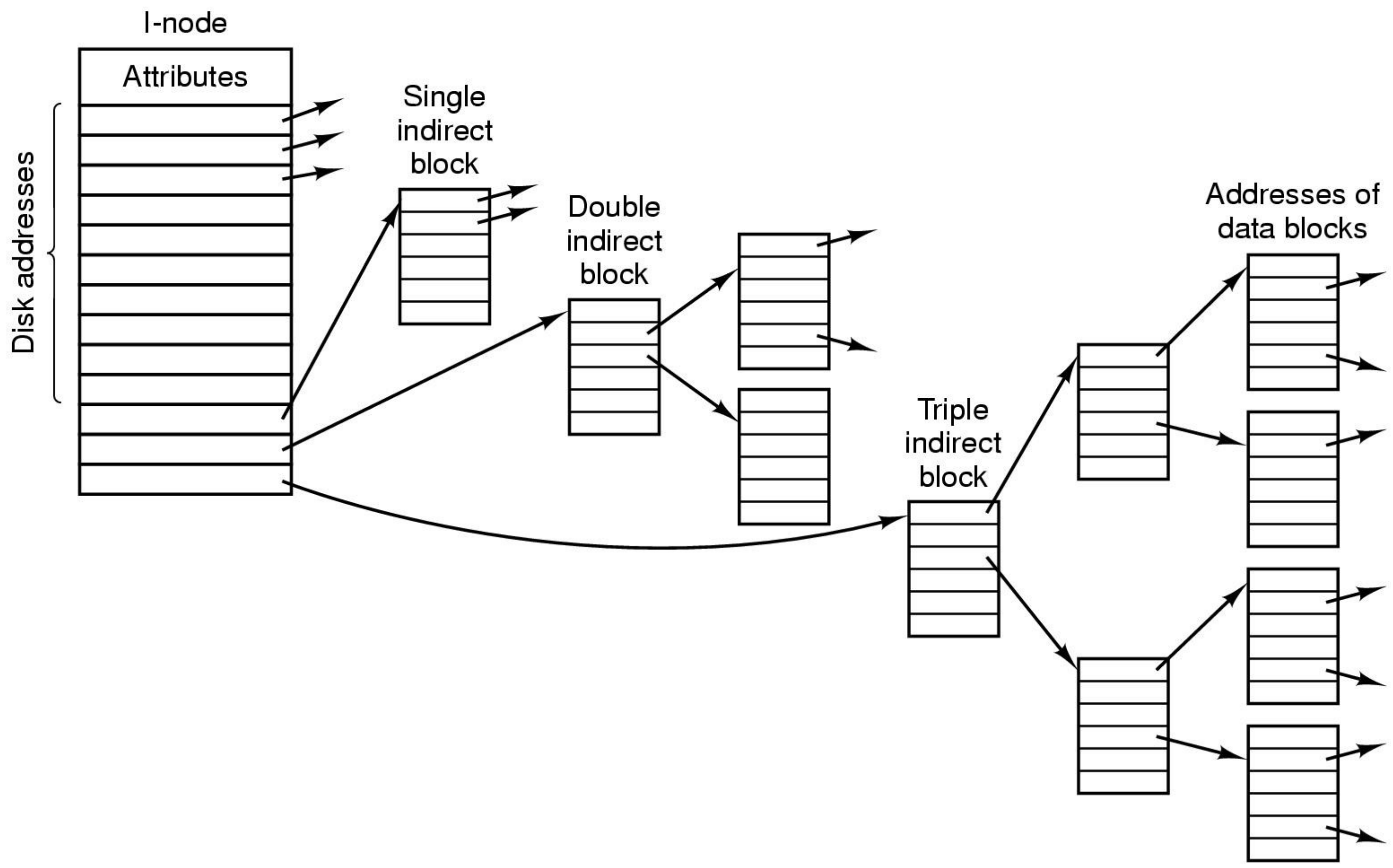
Indexed Allocation

- Efficient random access without external fragmentation,
- Size of index block
  - One data block
- Overhead of index block
  - Wastage of space
  - Small sized files

Linked scheme

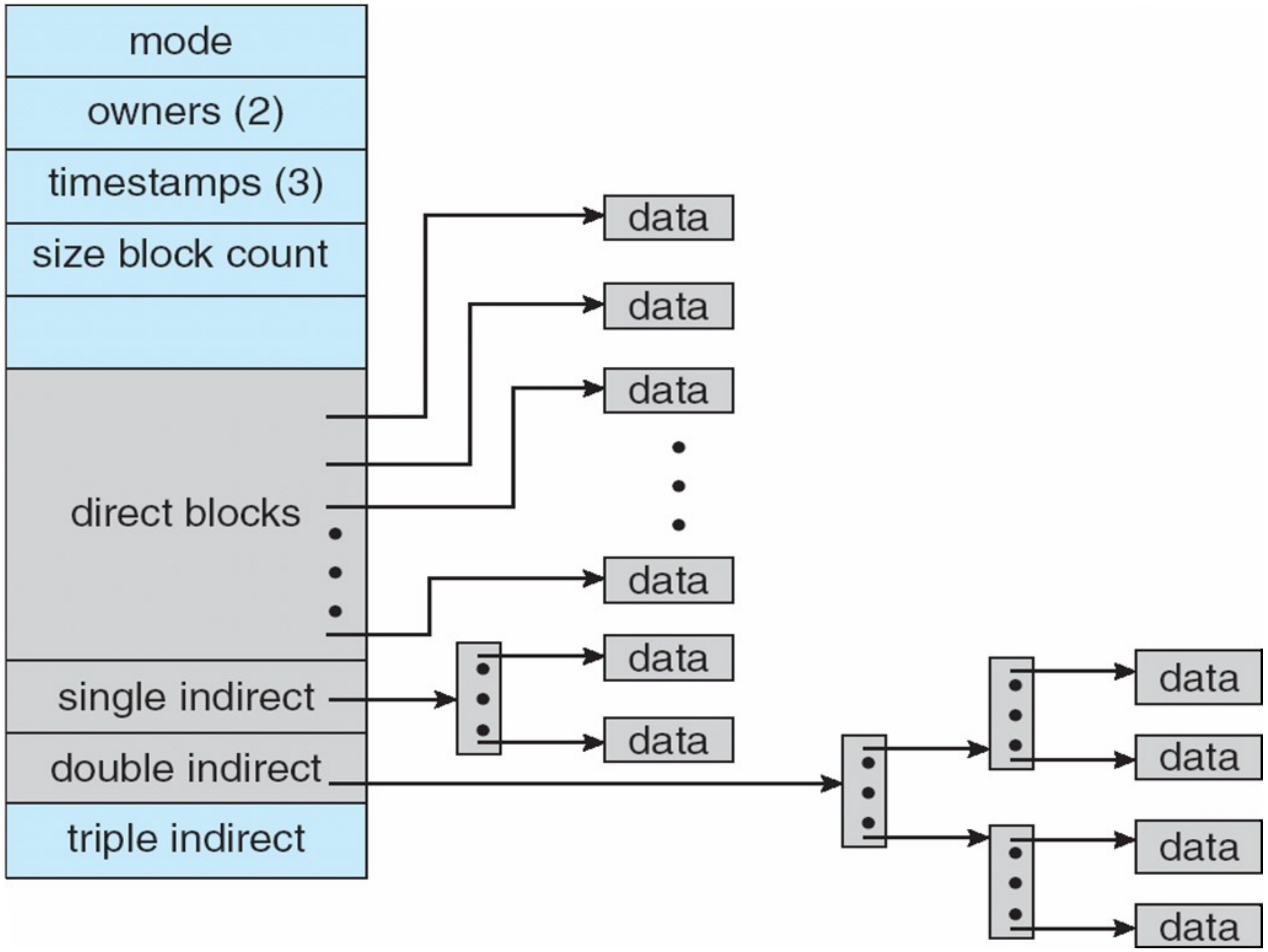
- Linked scheme – Link blocks of index table (no limit on size)
- Multilevel index

The UNIX File System



A UNIX i-node.

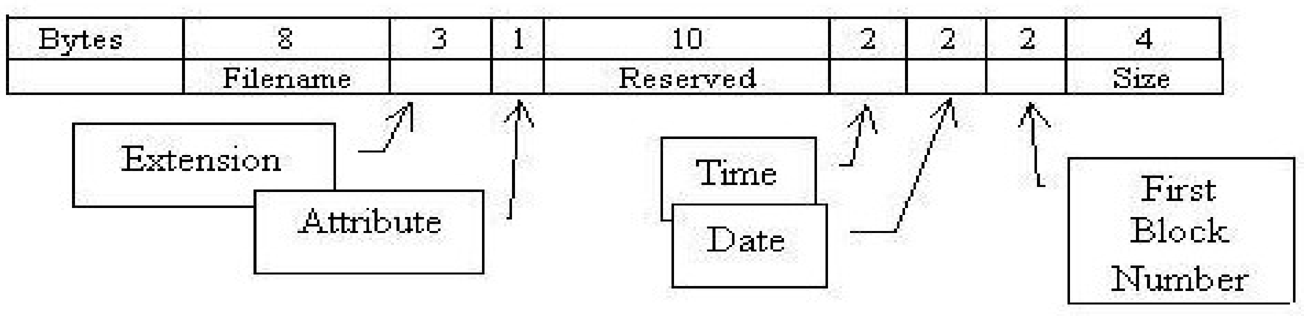
Combined Scheme: UNIX UFS
(4K bytes per block, 32-bit addresses)

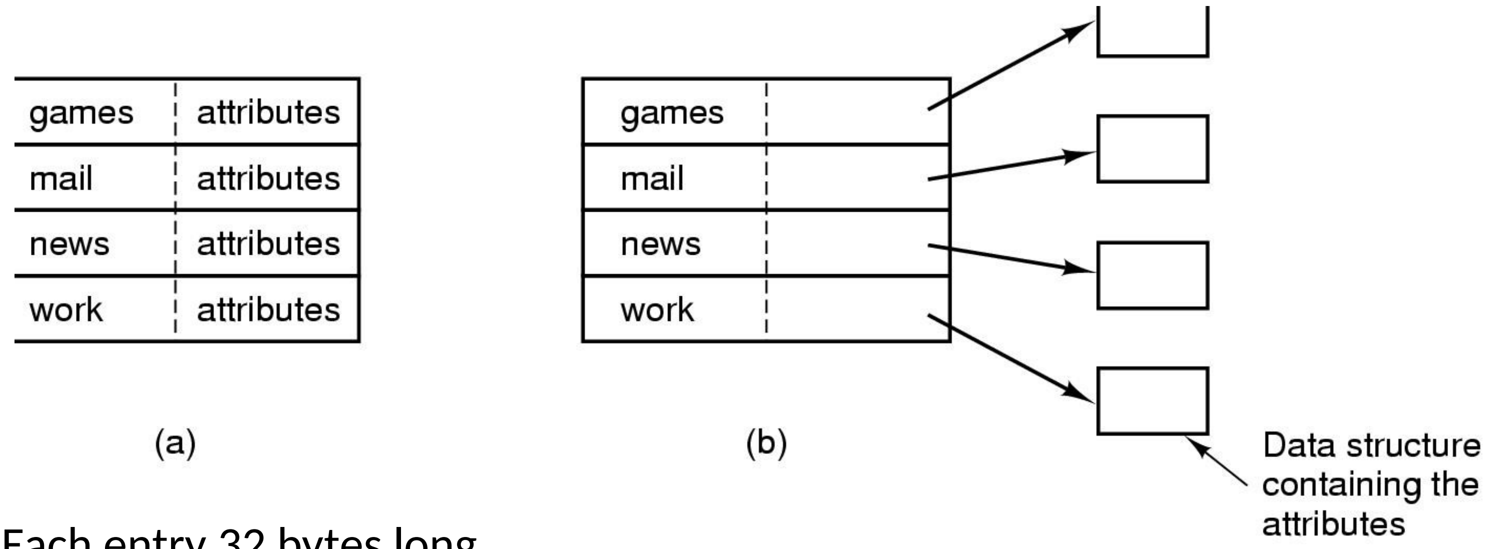


Implementing Directories

- OS uses path name supplied by the user to locate the directory entry
- Stores attributes
- Directory entry specifies block addresses by providing
  - Number of first block (contiguous)
  - Number of first block (linked)
  - Number of i-node

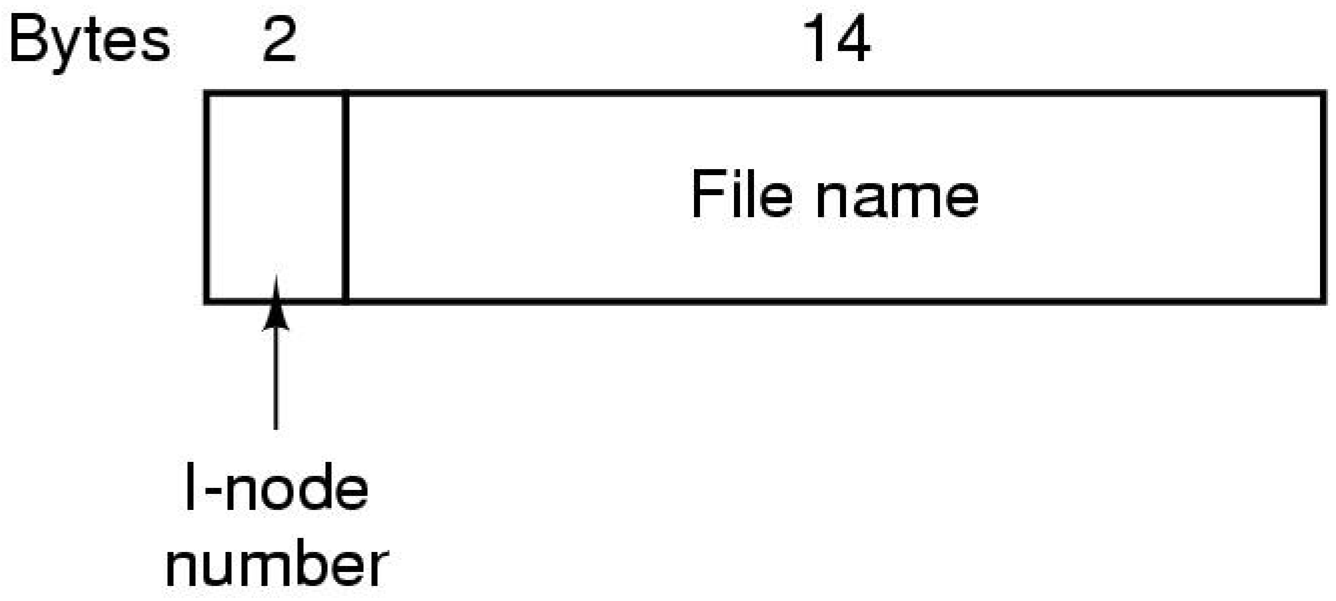
Implementing MS DOS Directories



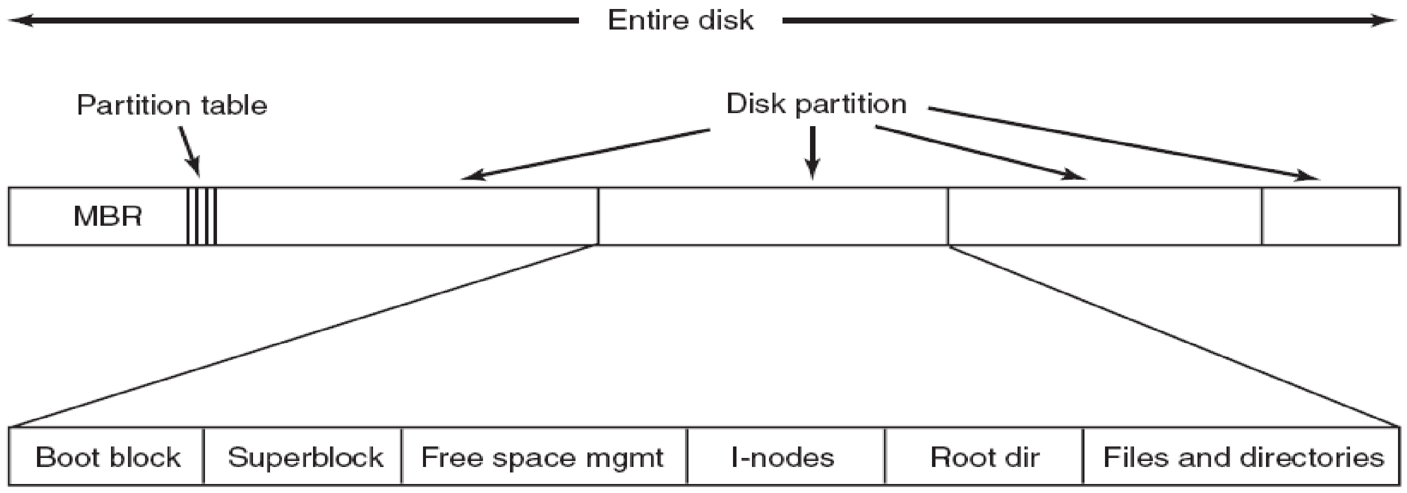


Each entry 32 bytes long

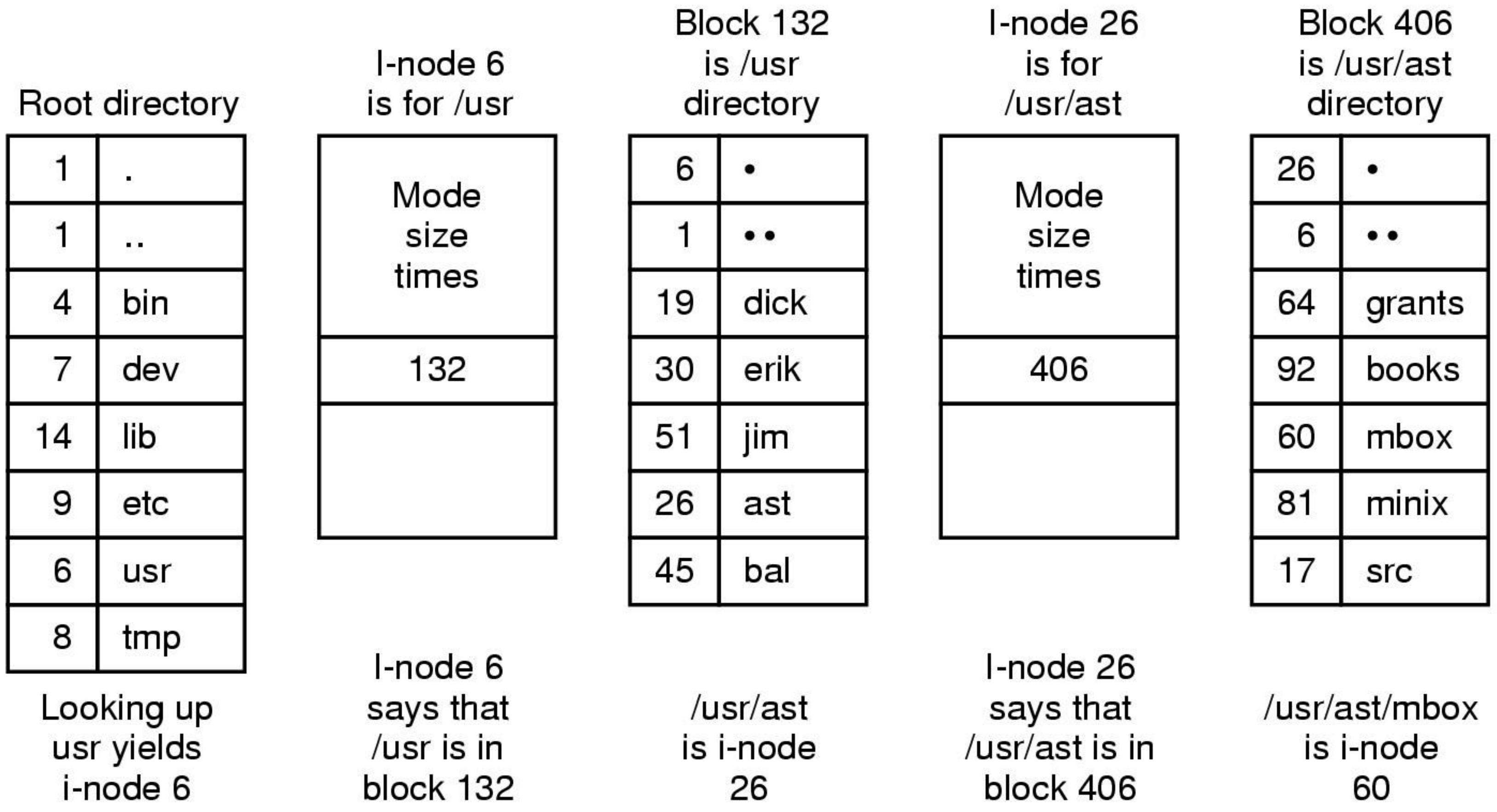
The UNIX File System



Disk Layout


Entire disk
Partition table
Disk partition
MBR
Boot block
Superblock
Free space mgmt
I-nodes
Root dir
Files and directories

The UNIX File System


Root directory
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
Looking up usr yields i-node 6
I-node 6 is for /usr
Mode size times
132
I-node 6 says that /usr is in block 132
Block 132 is /usr directory
6 •
1 ••
19 dick
30 erik
51 jim
26 ast
45 bal
/usr/ast is i-node 26
I-node 26 is for /usr/ast
Mode size times
406
I-node 26 says that /usr/ast is in block 406
Block 406 is /usr/ast directory
26 •
6 ••
64 grants
92 books
60 mbox
81 minix
17 src
/usr/ast/mbox is i-node 60

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

Directory Implementation

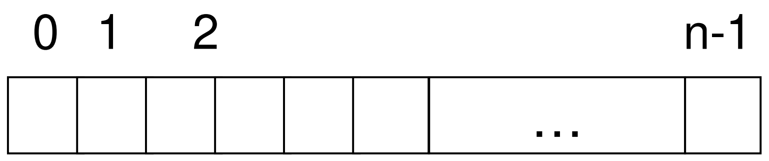
- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
  - New file creation / deletion

- Cache the frequently accessed entry
- Binary search to speedup directory search
  - Could keep ordered alphabetically via linked list
  - or use B+ tree

- **Hash Table** – hash data structure
  - Hash value computed from filename
  - Decreases directory search time
  - Insertion and deletion simple
  - **Collisions** – situations where two file names hash to the same location
    - Chaining
- Hash table of fixed size
  - Only good if entries are fixed size (CD-ROM)
- Performance depends on hash function

Free-Space Management

- File system maintains **free-space list** to track available blocks
- **Bit vector** or **bit map** (*n* blocks)
- Each block is represent by 1 bit

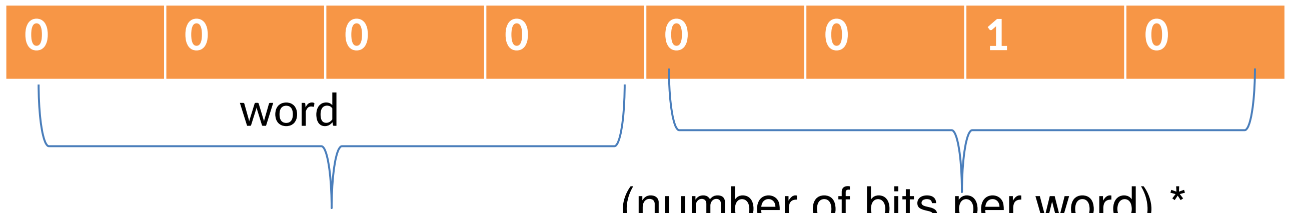


bit[*i*] = { 1 ⇒ block[*i*] free ; 0 ⇒ block[*i*] occupied }

Simple and Efficient to find first free blocks or n consecutive free blocks

Mac

CPUs have instructions to return offset within word of first "1" bit



Bit map

Block number calculation= (number of bits per word) * (number of 0-value words) + offset of first 1 bit

Free-Space Management

- Bit map requires extra space
  - Example:

  block size = 4KB = 2^{12} bytes

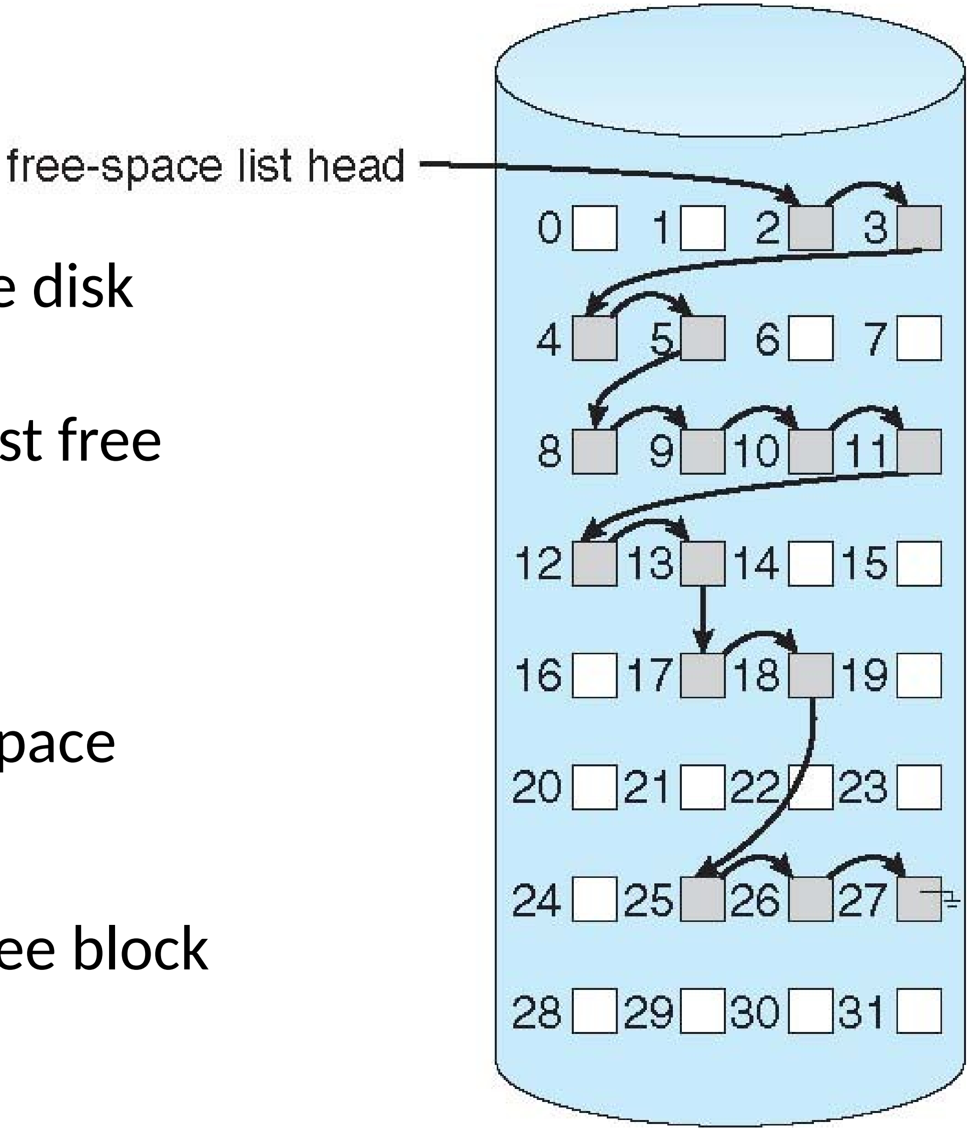
  disk size = 2^{40} bytes (1 terabyte)

  Number of blocks *n* = $2^{40}/2^{12} = 2^{28}$ bits (or 256 MB)

  if clusters of 4 blocks -> 64MB of memory

- Keep the vector in main memory

Linked Free Space List on Disk



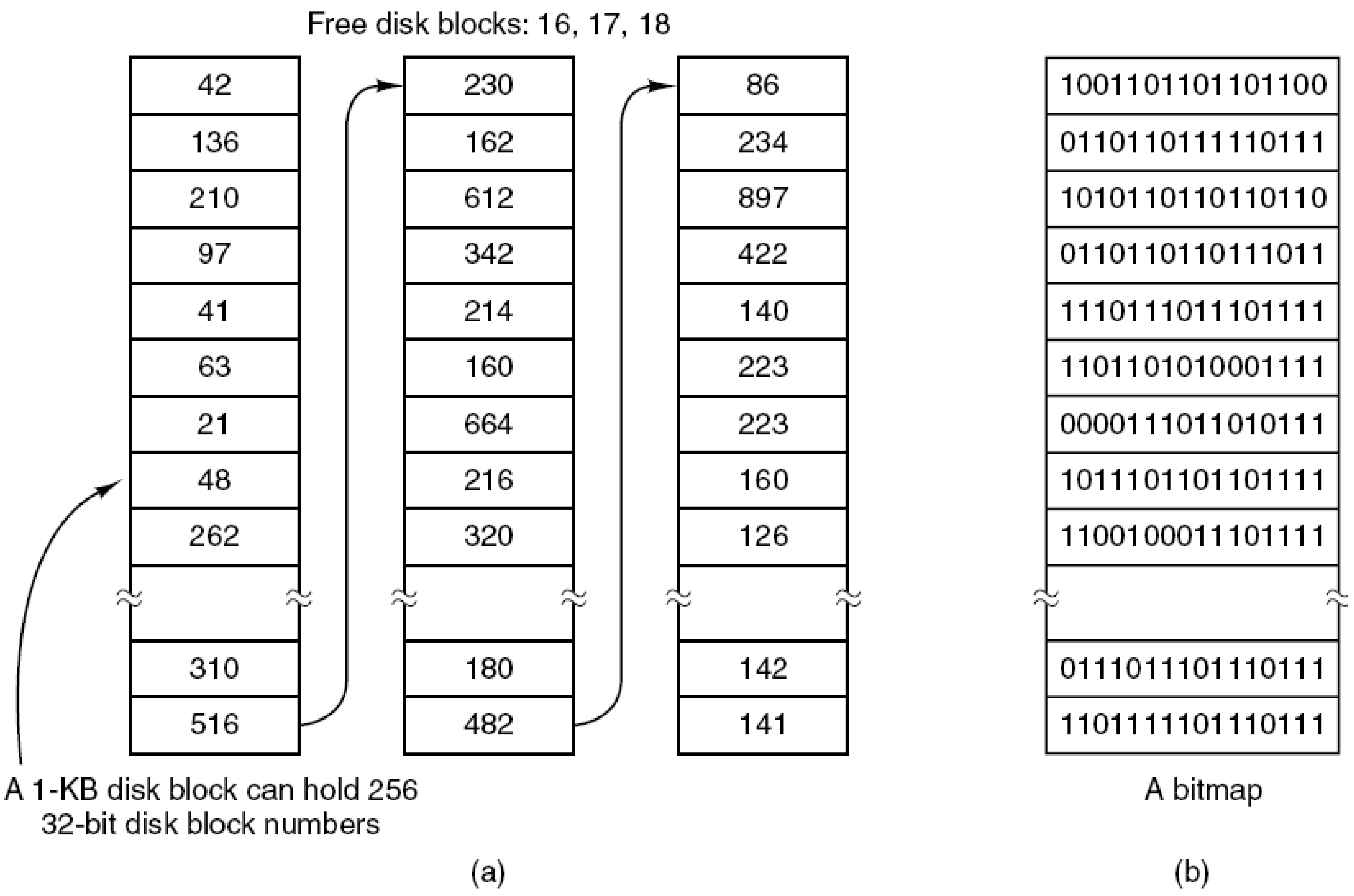
- Link together all the free disk blocks
- Keep a pointer to the first free block

- Cannot get contiguous space easily
  - Traverse the list
- Generally require first free block

Free-Space Management

- Grouping
  - Reserve few disk blocks for management
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers
- Counting

Keeping Track of Free Blocks (1)



Storing the free list using (a) Grouping (b) Bitmap.

1KB block

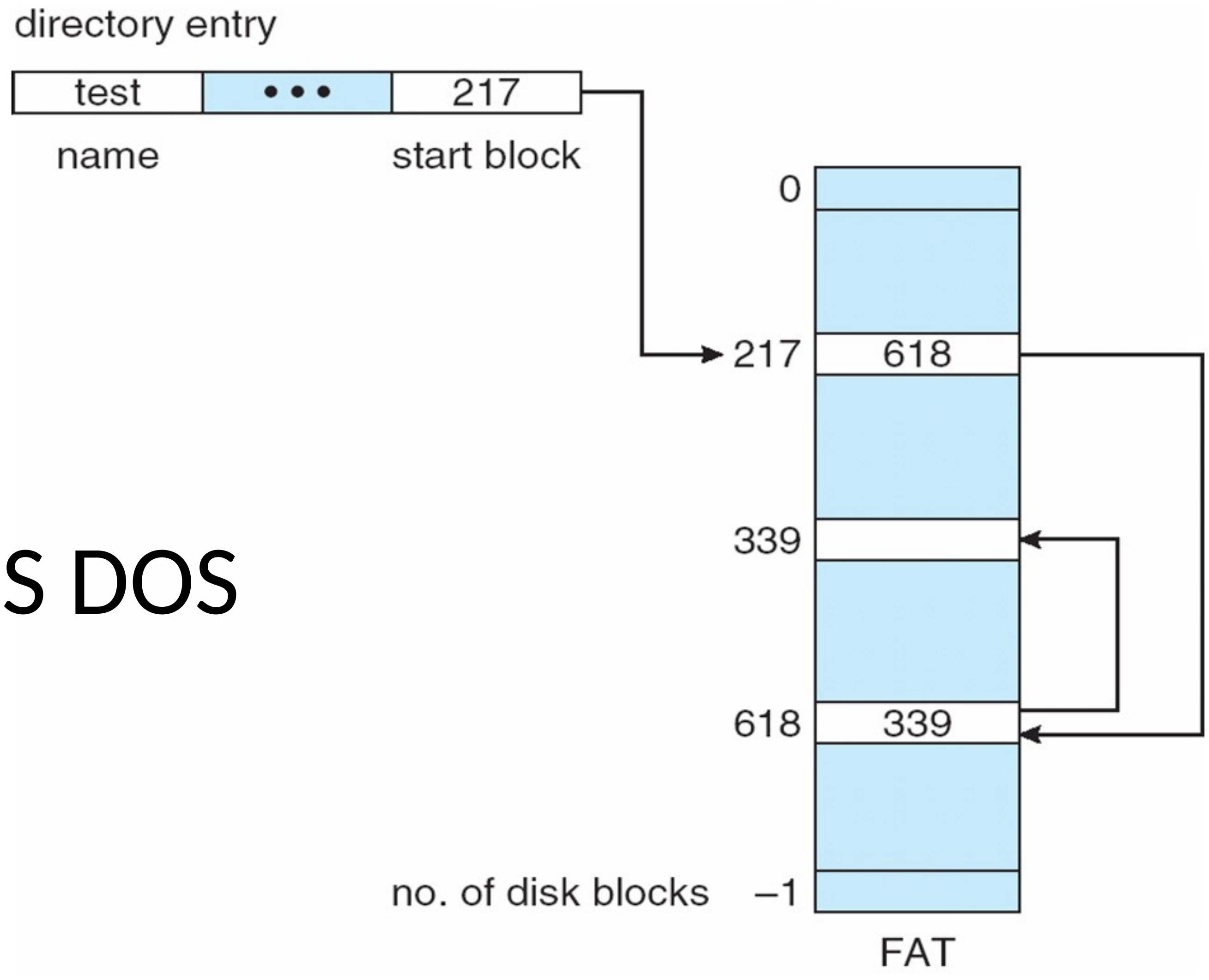
16 bits block number

Each block holds 511 free blocks

20M disk needs 40 blocks for free list

How many for bit map?

Free Space for FAT



MS DOS

Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds

- EAT = (1 – p) x 200 + p (8 milliseconds)

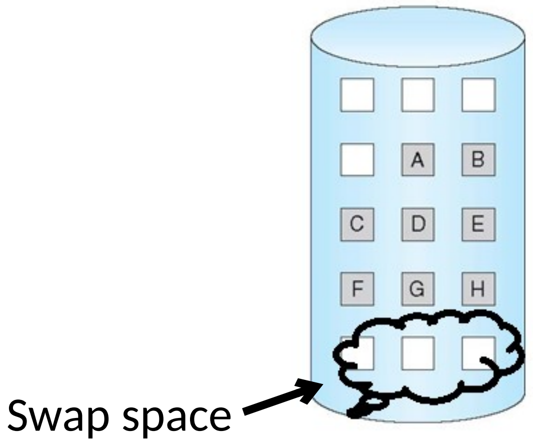
  = (1 – p) x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then

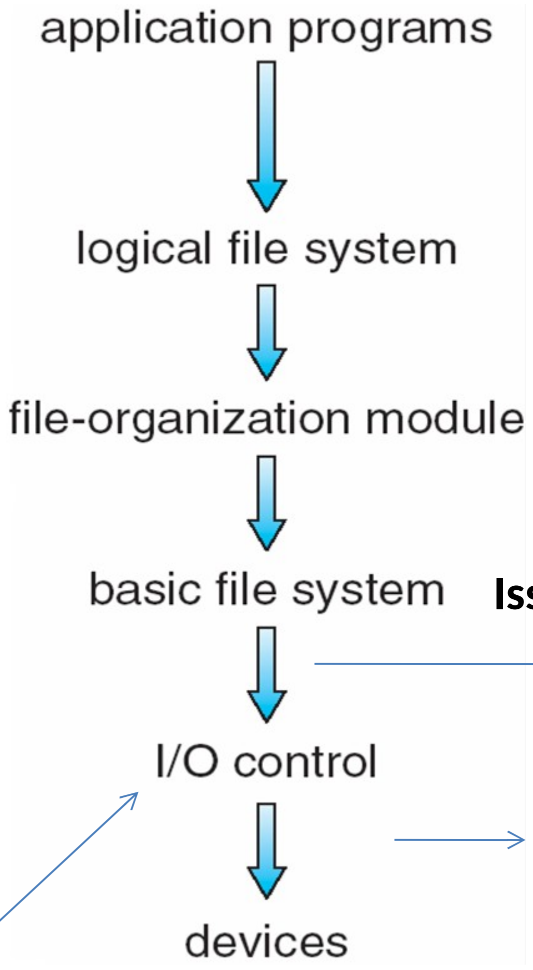
  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40!!
- If want performance degradation < 10 percent
  - 220 > 200 + 7,999,800 x p
    20 > 7,999,800 x p
  - p < .0000025
  - < one page fault in every 400,000 memory accesses

Better utilization of swap space



Layered File System



Manages **FCB**

Translates logical to physical blocks

Issue commands

R/W Physical block (cylinder, track, sector)

Hardware specific instruction

- Each level uses the feature of low level
- Create new features for higher level

Device driver, transfer information between memory/disk