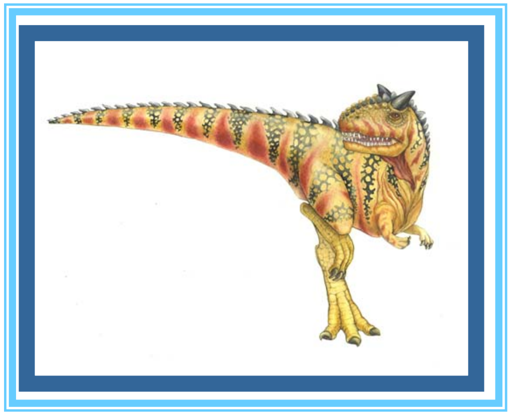
Chapter 11: File System Implementation



Chapter 11: File System Implementation

- Allocation Methods
- Free-Space Management

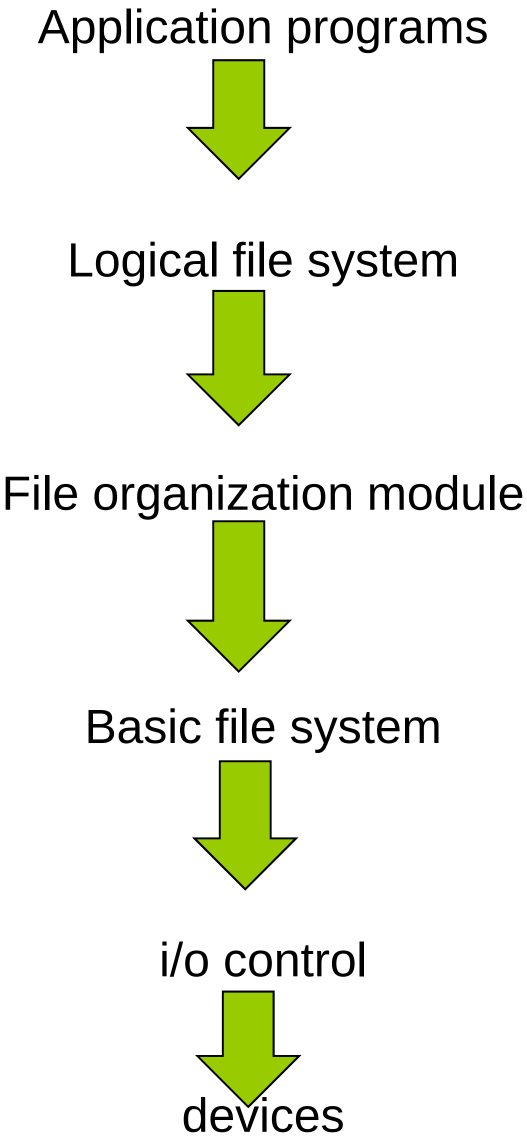
Objectives

- Introduction to file system structure.
- To discuss block allocation and free-block algorithms.

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- The File system is organized into layers (levels).
- File system resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located, and retrieved easily
- File control block – storage structure consisting of information about a file , including ownership, permissions, and location of the file contents
- Device driver controls the physical device

Layered file system



Application programs

Logical file system

File organization module

Basic file system

i/o control

devices

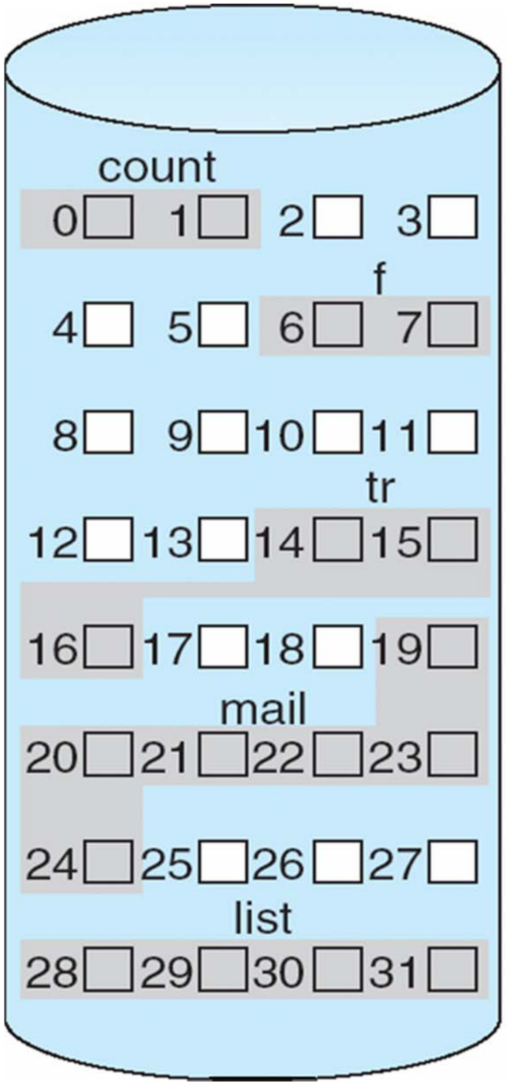
Allocation Methods

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

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file .
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- External fragmentation
- Files cannot grow

Contiguous Allocation of Disk Space



directory

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

Extent-Based Systems

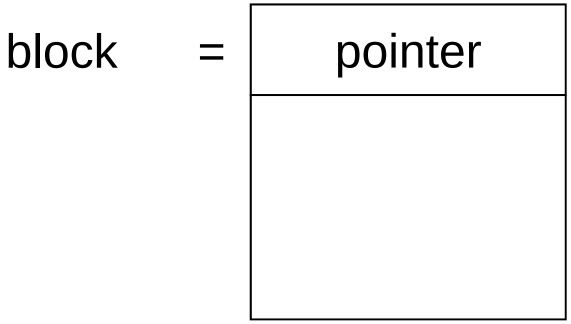
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme

- 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

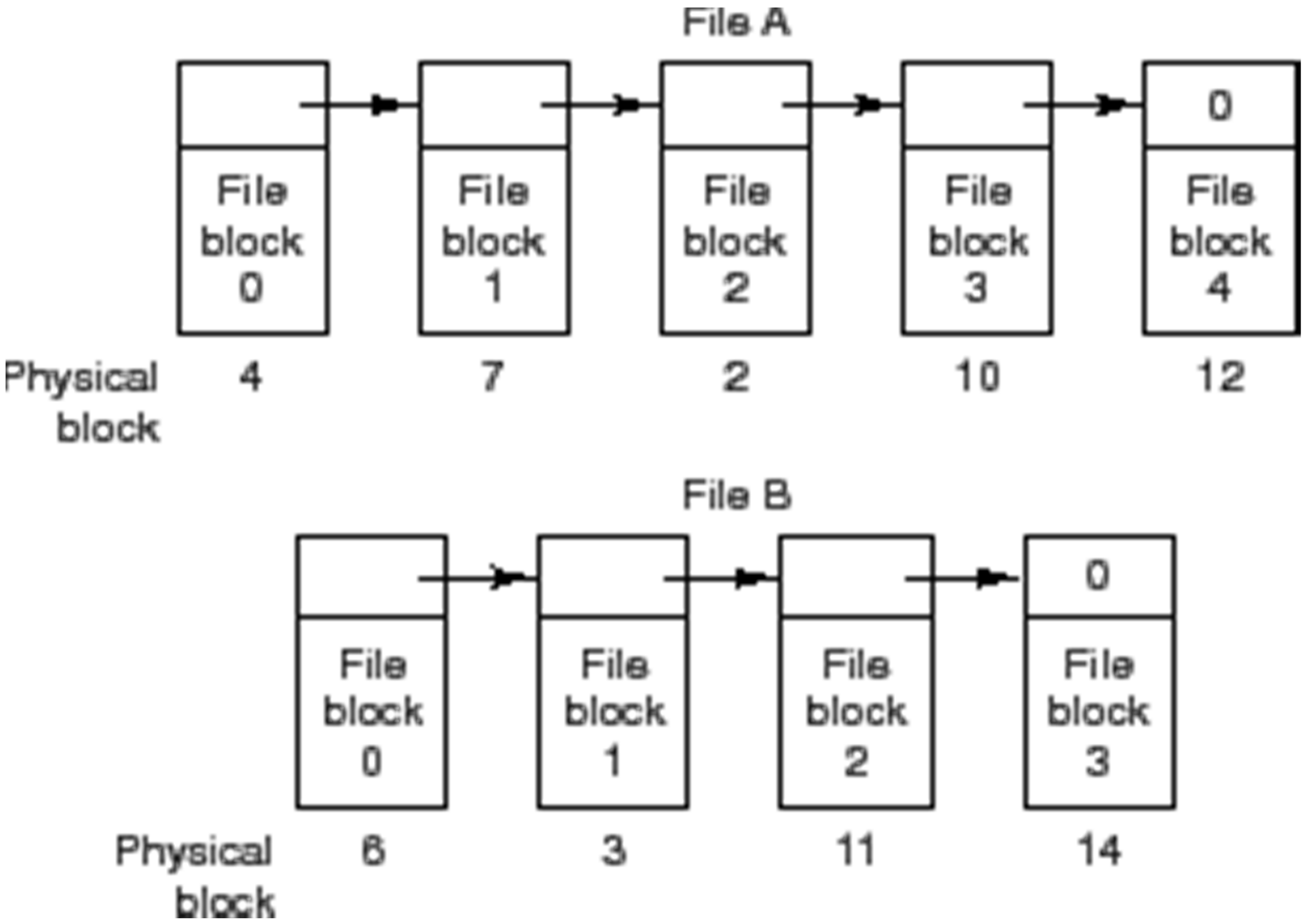
Linked Allocation

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

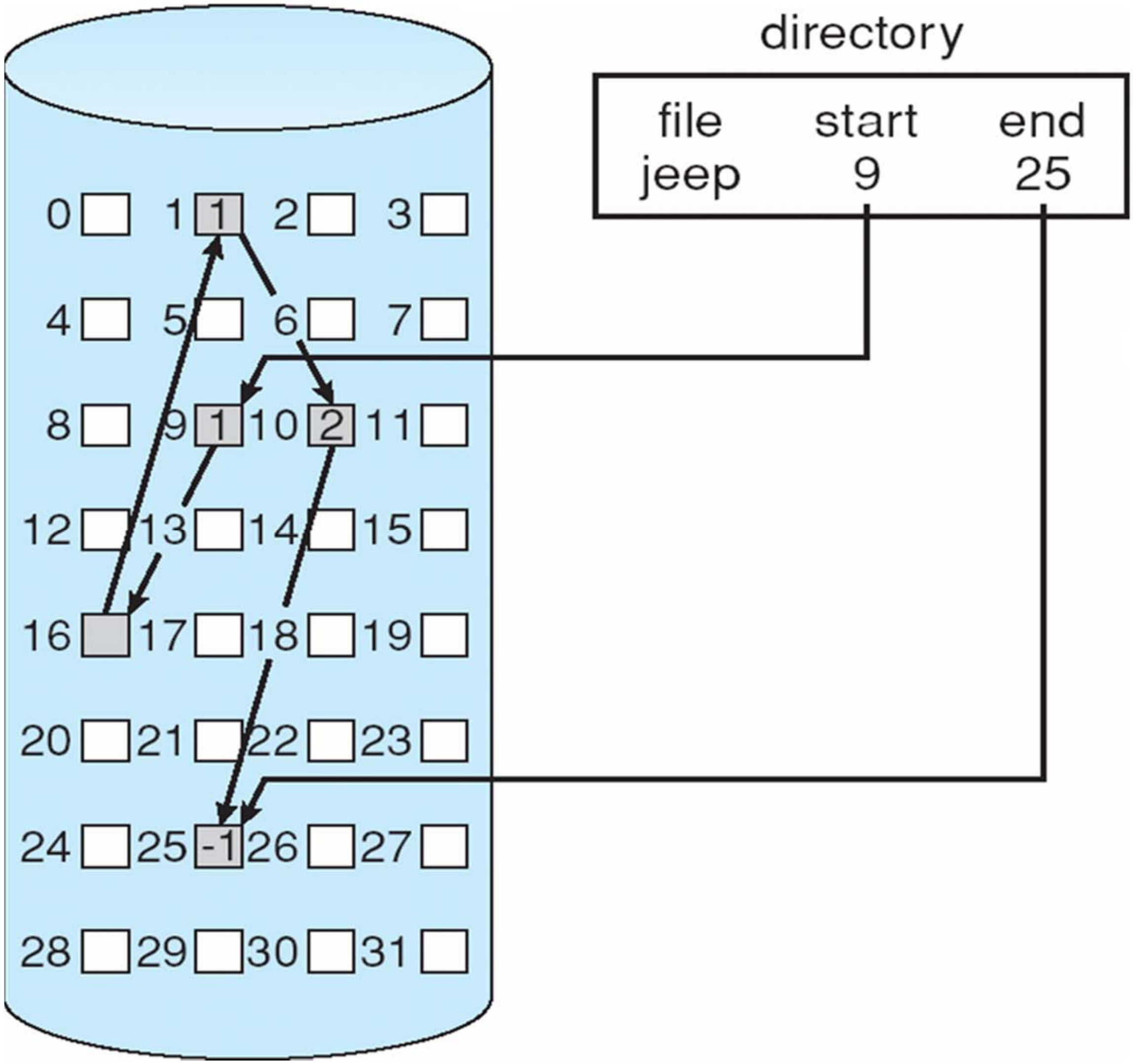


- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- a space is required for the pointers.

Linked Allocation



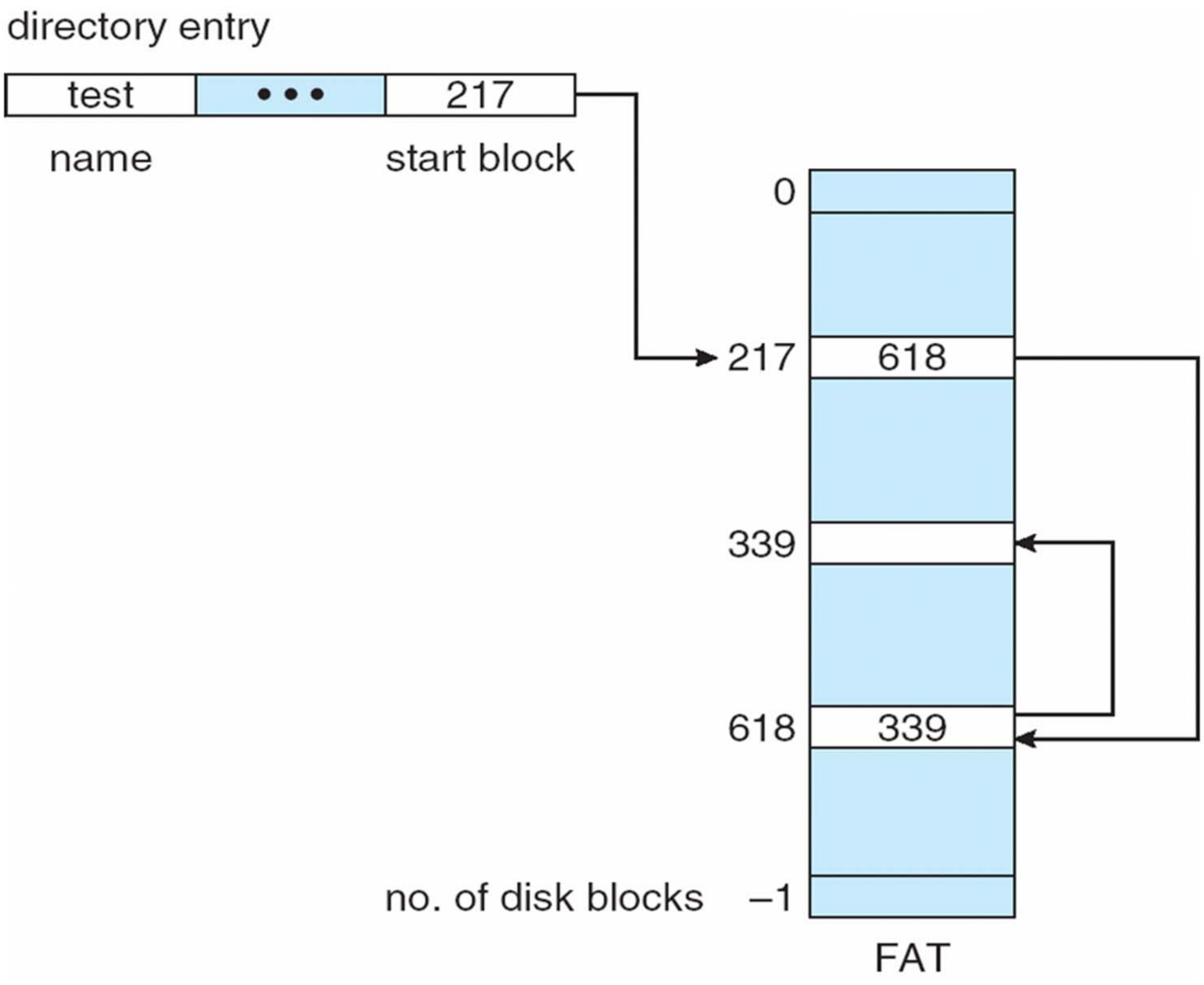
Linked Allocation



File-Allocation Table

- An important variation on linked allocation is the use of a file-allocation table (FAT).
  - A section of disk at the beginning of each volume is set aside to contain the table.
  - The table has one entry for each disk block and is indexed by block number.
  - The FAT is used in much the same way as a linked list. The directory entry contains the block number of the first block of the file.
  - The table entry indexed by that block number contains the block number of the next block in the file.
  - This chain continues until the last block, which has a special end-of-file value as the table entry.
- Unused blocks are indicated by a 0 table value

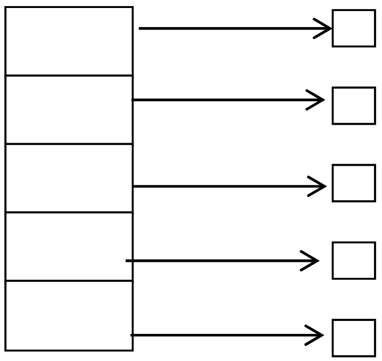
File-Allocation Table



Indexed Allocation

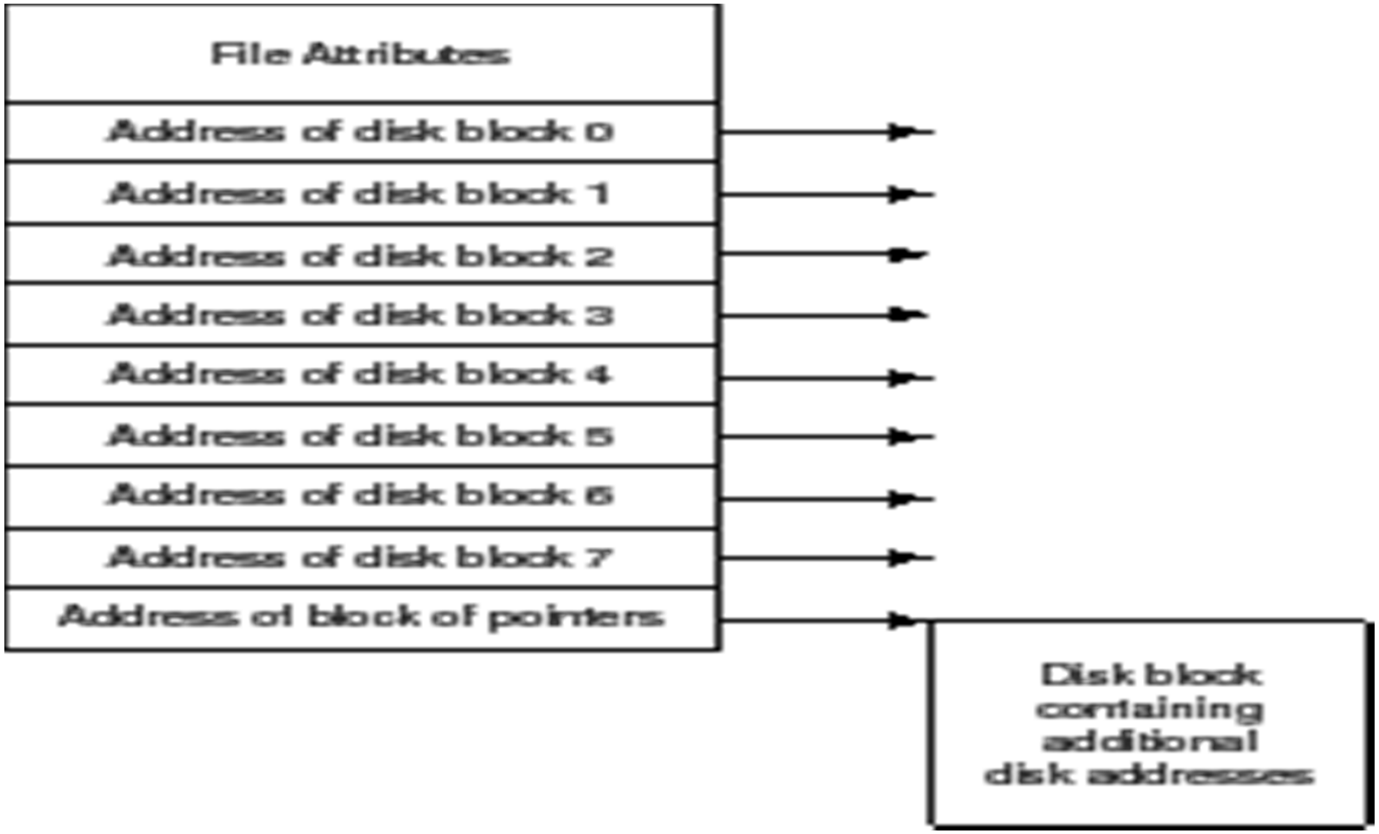
- Brings all pointers together into the index block
- Each file has its own index block, which is an array of disk-block addresses.
- The ith entry in the index block points to the ith block of the file.

- Logical view

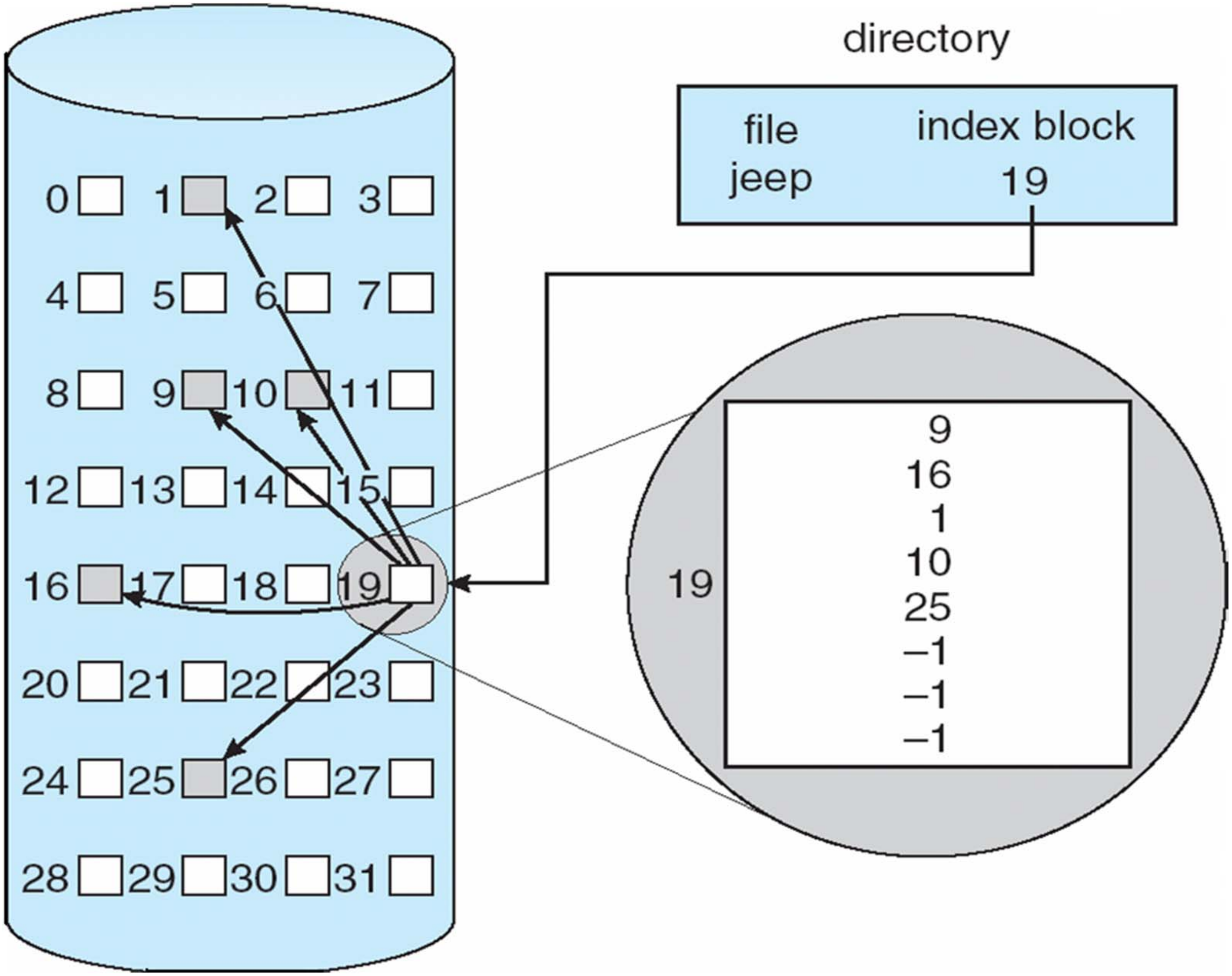


index table

Indexed Allocation



Example of Indexed Allocation



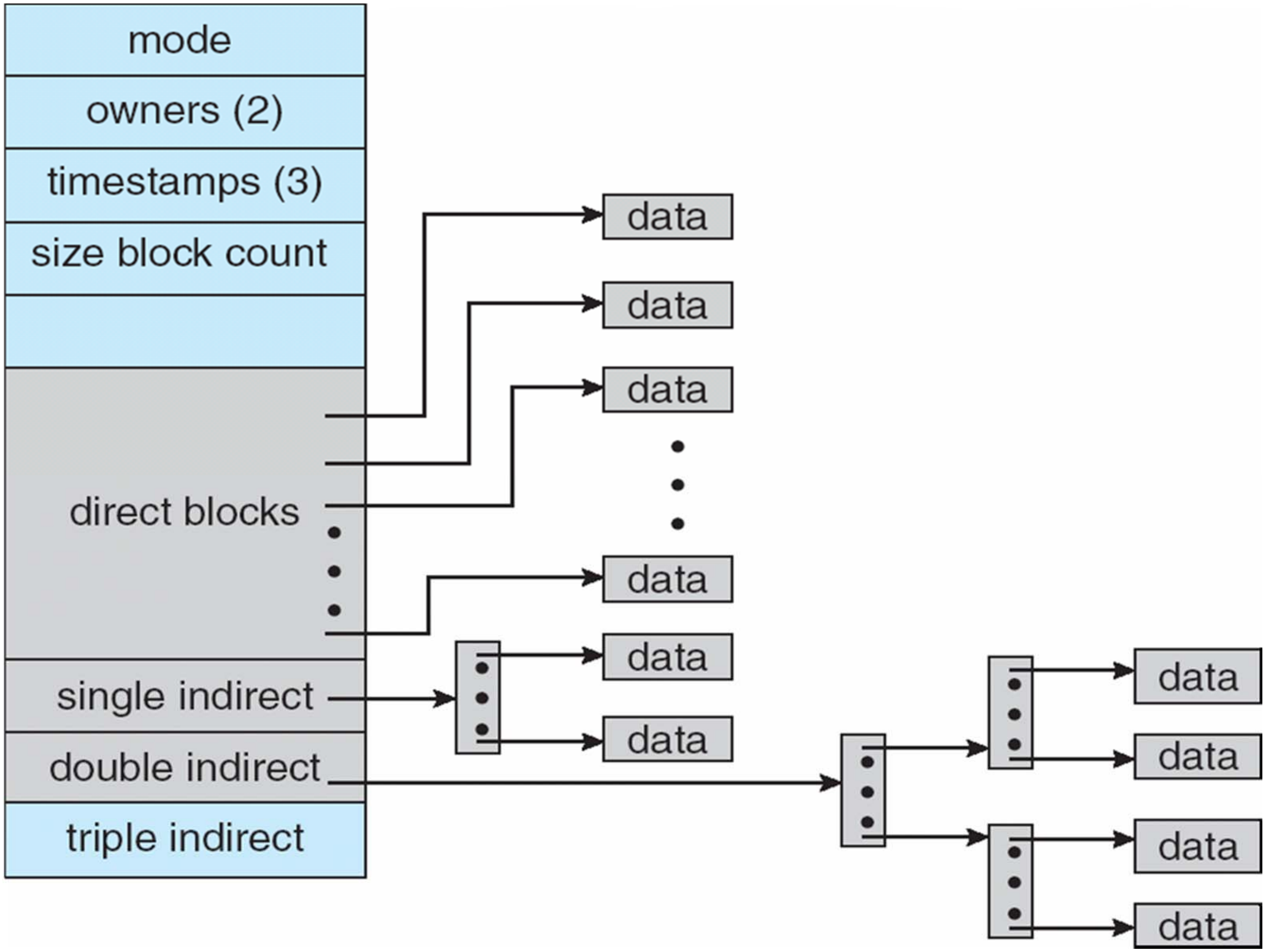
Indexed Allocation (Cont.)

- 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

Linked scheme

- An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we can link together several index blocks.
  - For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses.
  - The next address (the last word in the index block) is nil (for a small file) or is a pointer to another index block (for a large file).

Combined Scheme: UNIX UFS (4K bytes per block)



Free-Space Management

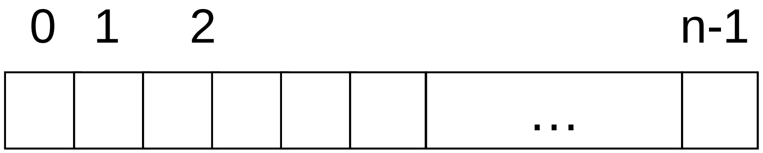
- To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks.
- To create a file, we search the free-space list for the required amount of space and allocate that space to the new file.
- When a file is deleted, its disk space is added to the free-space list.

Bit Vector

- the free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit.
  - If the block is free, the bit is 1;
  - If the block is allocated, the bit is 0.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18,25,26, and 27 are free and the rest of the blocks are allocated.
- The free-space bit map would be 001111001111110001100000011100000
- The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk.

Free-Space Management

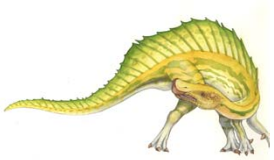
- Bit vector (*n* blocks)



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

Block number calculation

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



Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files

Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.

Linked List

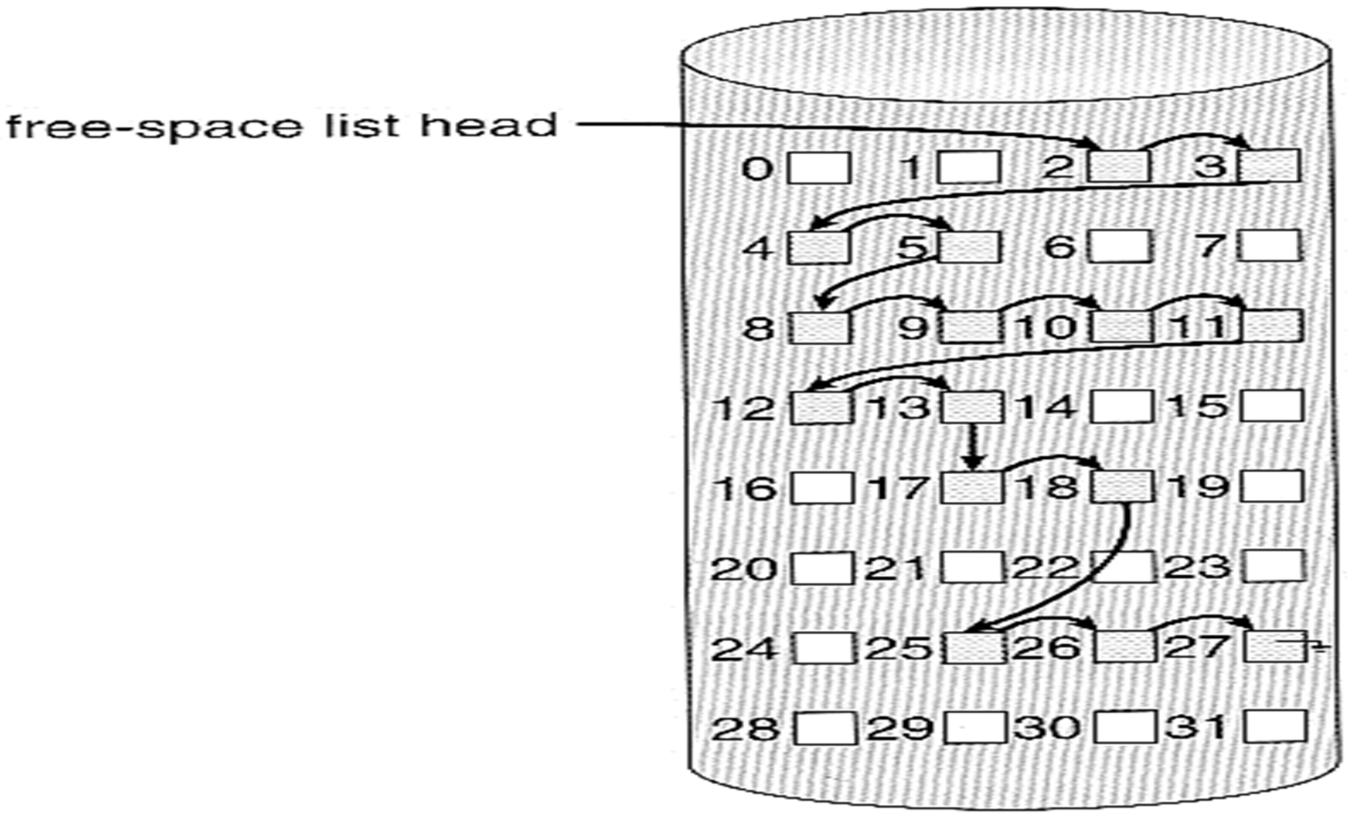
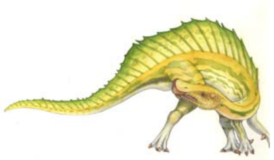


Figure 11.10 Linked free-space list on disk.

Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- The first n-1 of these blocks are actually free. The last block contains the addresses of another n free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.



Counting

- rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.
- Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

End of Chapter 11