Operating Systems Concepts

File Systems

CS 4375, Fall 2025
Instructor: MD Armanuzzaman (*Arman*)
marmanuzzaman@utep.edu
November 17, 2025

Summery

- The Deadlock Problem

  - Characterization of Deadlock

  - Resource Allocation Graph

  - Handling Deadlocks

    - Deadlock Prevention

    - **Deadlock Avoidance**

    - **Deadlock Detection**

    - **Deadlock Recovery**

Agenda

- File Allocation Methods

  - Contiguous

  - Linked

  - Indexed

- Free-Space Management

  - Linked List

  - Bit Vector

  - Grouping

  - Counting

Disk, File, File Systems

- Data is stored on physical devices. E.g., disks
  - Sectors
- A file is a logical view of the data, perceived by the programmer
  - Sequential access
  - Random access
- A file system:
  - Define how the file system should look to the user
  - Creating algorithms and data structures to map the logical file system onto the physical storage devices.

application programs
⬇
logical file system
⬇
file-organization module
⬇
basic file system
⬇
I/O control
⬇
devices
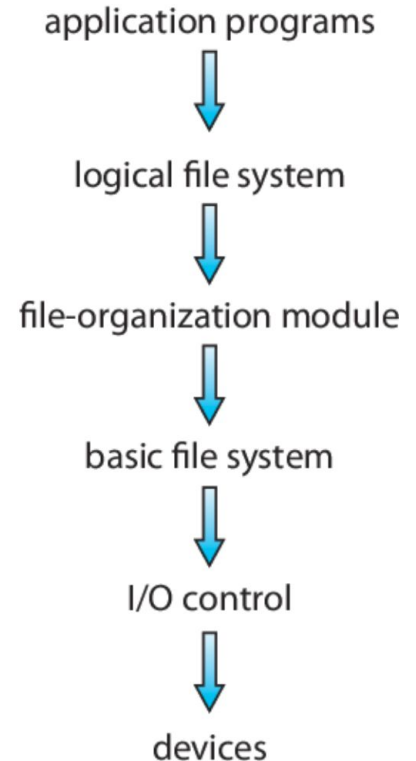
File Allocation Methods

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

File Allocation Methods - Contiguous

- Each file occupies set of contiguous blocks

- Best performance in most cases (both sequential and random access)
- Simple– only starting location (block #) and length (number of blocks) are required
- Finding space for file
- Estimating a file size at creation, or growing it.
- External fragmentation
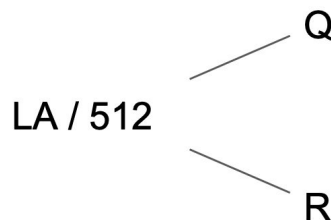 - Compaction

Contiguous Allocation of Disk Space
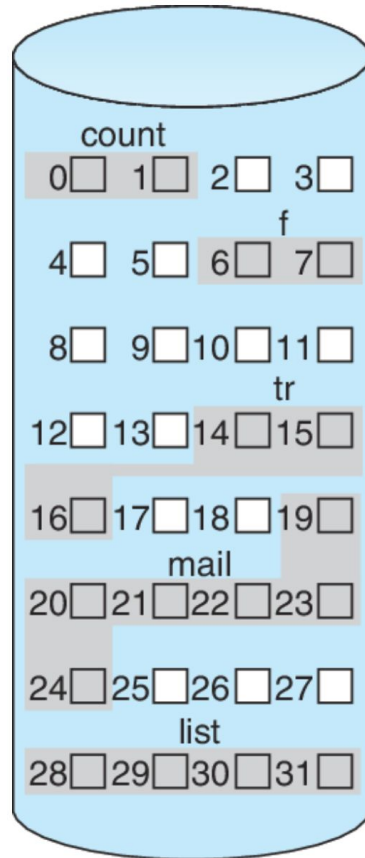
Contiguous Allocation Address Translation

- Mapping from logical to physical disk address:

    - LA: logical address

    - 512: Disk block size (sudo fdisk -l)

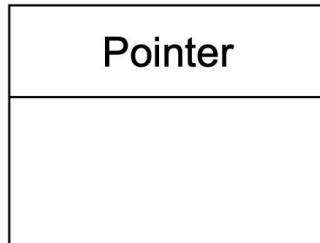   - Q: quotient

    - R: remainder

$$LA / 512 \begin{array}{l} \nearrow \ Q \\ \searrow \ R \end{array}$$

- Block to be accessed = Q + starting address

- Displacement into block = R

File Allocation Methods - Linked Allocation

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

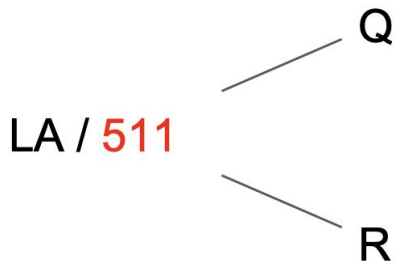Block:

Pointer

- Simple - need only starting address
- Free-space management system - no waste of space
- Defragmentation not necessary for file allocation
- No random access, locating a block can take many disk seek & I/O
- Extra space required for pointers
    - Clusters
- **Reliability**: What if a pointer gets corrupted?

9

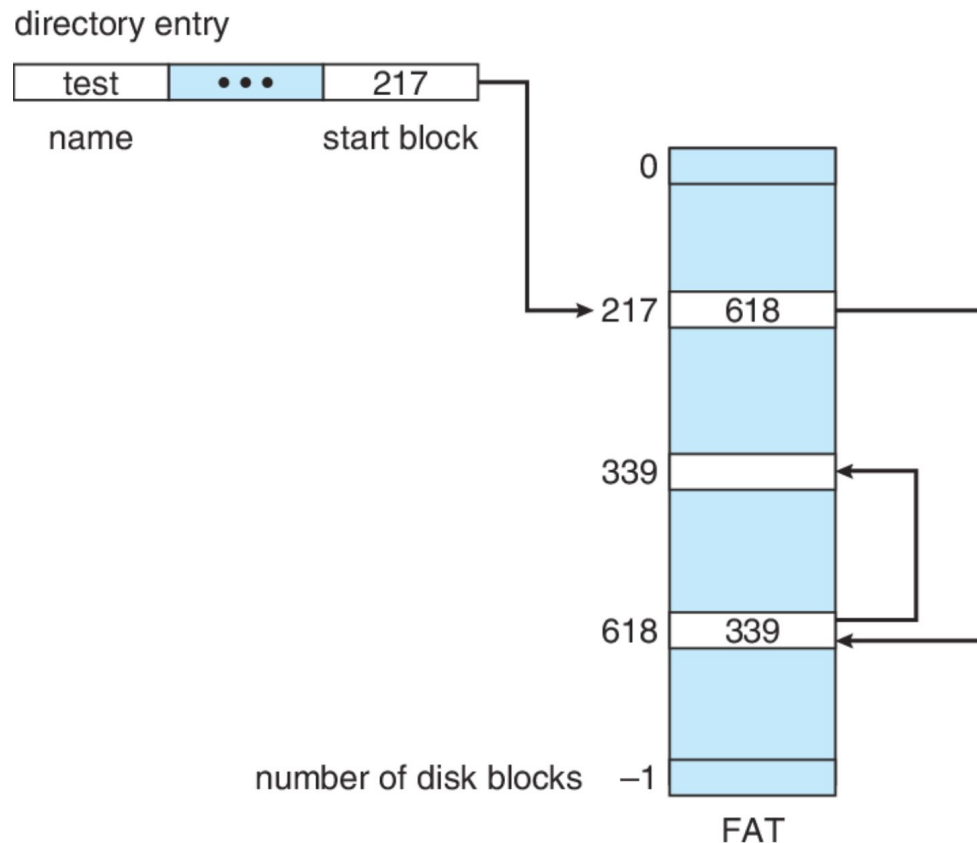# Linked Allocation of Disk Space
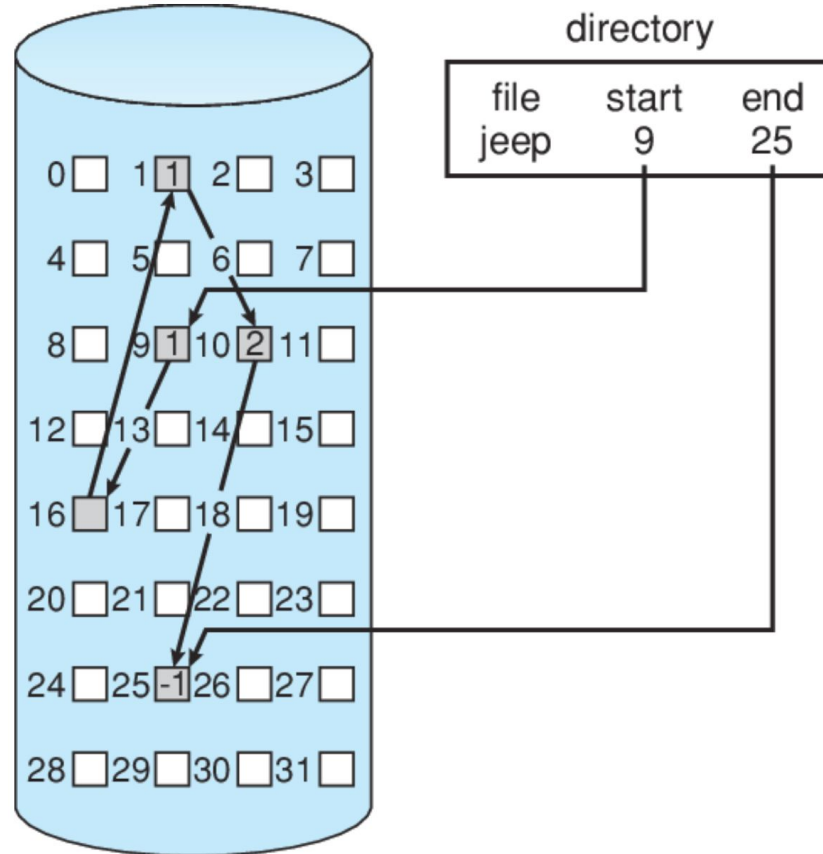
Linked Allocation Address Translation

- Mapping (assuming pointer takes only one byte):
  - LA: logical address
  - 512: Disk block size
  - Q: quotient
  - R: remainder

$$LA / 511 \nearrow Q \searrow R$$

- Block to be accessed = Q^{th} block in the linked chain of blocks representing the file.

- Displacement into block = R + 1

File Allocation Table (FAT)

- Beginning of volume has table indexed by block number
- Much like a linked list but faster on disk and cacheable
  - Smaller; Memory storage; cache
- New block allocation simple

directory entry

| test | • • • | 217 |
| name | | start block |

0

217 | 618

339

618 | 339

number of disk blocks −1

FAT

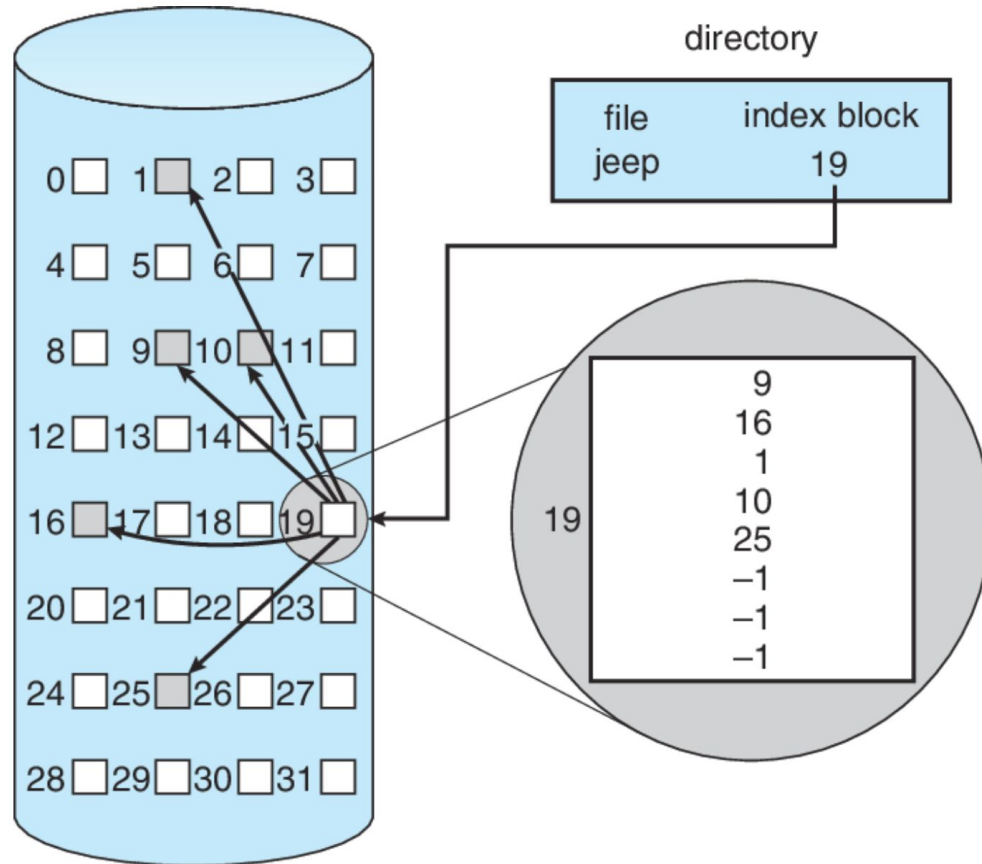# File Allocation Methods - Indexed Allocation

- Brings all pointers together into the *index* block, to allow random access to file blocks.
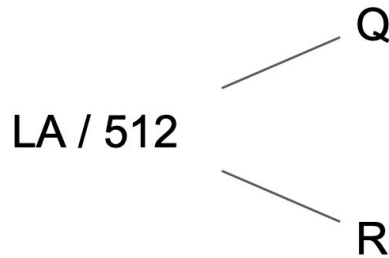
Index Table:

- Supports direct access

- Prevents external fragmentation

- High pointer overhead → Wasted space

Indexed Allocation of Disk Space

Indexed Allocation Address Translation

- 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 of 256K bytes and block size of 512 bytes. We need only 1 block for index table.

$$\text{LA / 512} \quad \begin{cases} Q \\ R \end{cases}$$

- Displacement into index table = Q

- Displacement into block = R

Indexed Allocation Address Translation

- How large should the index block be?
  - Too big → waste of space for small files
  - Too small → the size of file will be limited
- Solutions:
  - Linked scheme (Index blocks linked together)
  - Multilevel index (2 level indexes allow file size of up to 4 GB)
  - Combined scheme
    - First direct blocks
    - Then single indirect blocks
    - Then double indirect blocks
    - Then triple indirect blocks!

Indexed Allocation Address Translation

UNIX inode:
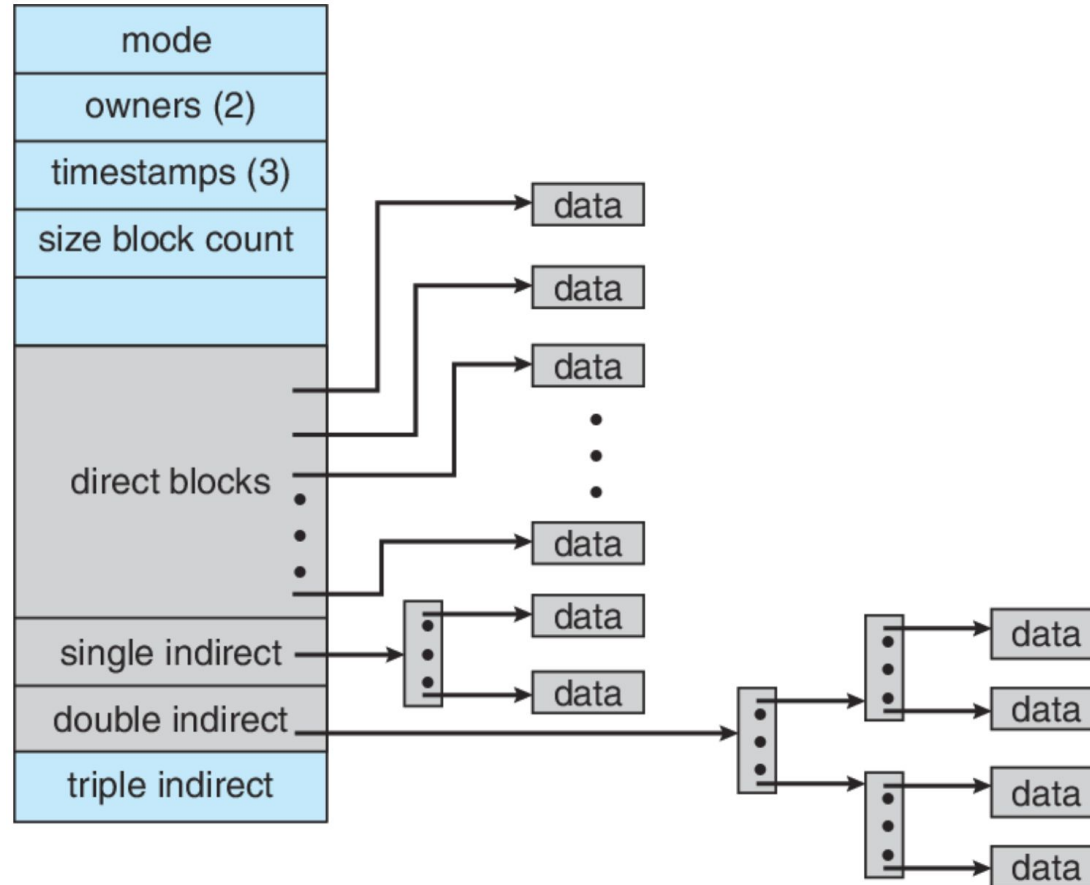
Performance

- Best method depends on file access type

    - **Contiguous**: great for sequential and random

    - **Linked**: good for sequential, not random

        - Declare access type at creation → select either contiguous or linked

    - **Indexed**: more complex!

        - Single block access could require 2 index block reads then data block

            read

Exercise

Consider a file system on a disk that has both logical and physical block sizes of **512 bytes**. Assume that the information about each file is already in memory. For each of the three allocation strategies (**contiguous**, **linked**, and **indexed**), answer these questions. Assume a pointer needs only one byte space.

A. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)

B. If we are currently at the 10th logical block of the file and want to access the 4th logical block. How many physical blocks must be read from the disk?

Exercise

1. **Contiguous**:

   Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.

A. Add X to Z (start address) to obtain the physical block number. Y is the displacement into that block.

B. One

Exercise

1. Linked:

    Divide the logical address by 511 with X and Y the resulting quotient and remainder respectively.

A. Chase down the linked list (getting X + 1 blocks in total). Y + 1 is the displacement into the last physical block.

B. Four

Exercise

1. Indexed:

   Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.

A. Get the index block into memory. Physical block address is contained in the index block at location X. Y is the displacement into the desired physical block.

B. Two

Free-Space Management

- Disk space limited

- Need to re-use the space from deleted files

- To keep track of free disk space, the system maintains a free-space list

  - Records all free disk blocks

- Implemented using

  - Linked Lists

  - Bit Vectors

  - Grouping

  - Counting

Free-Space Management - Linked List
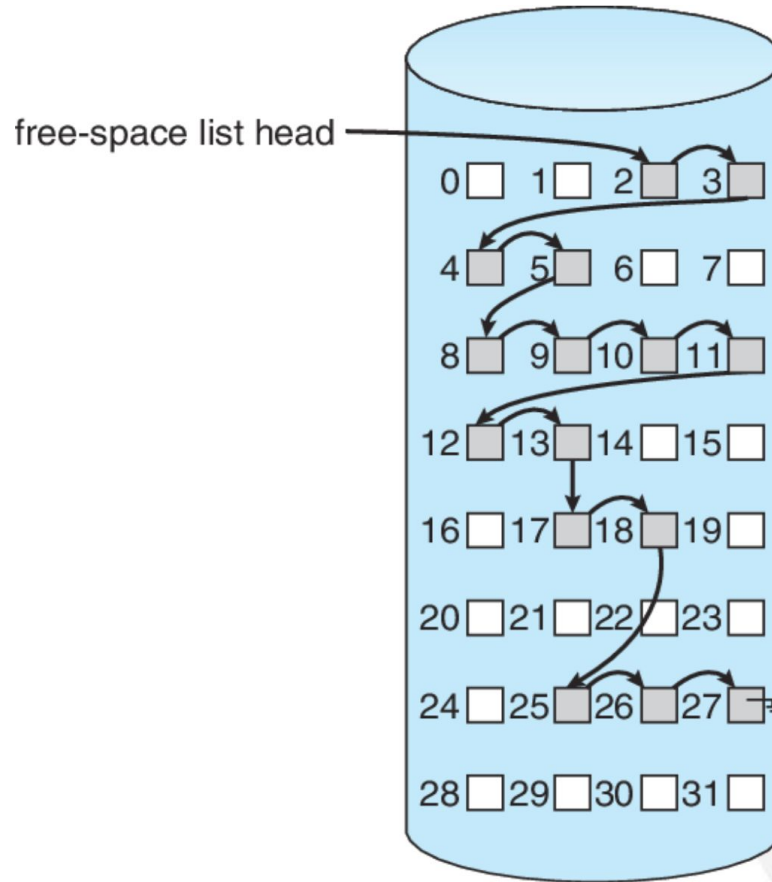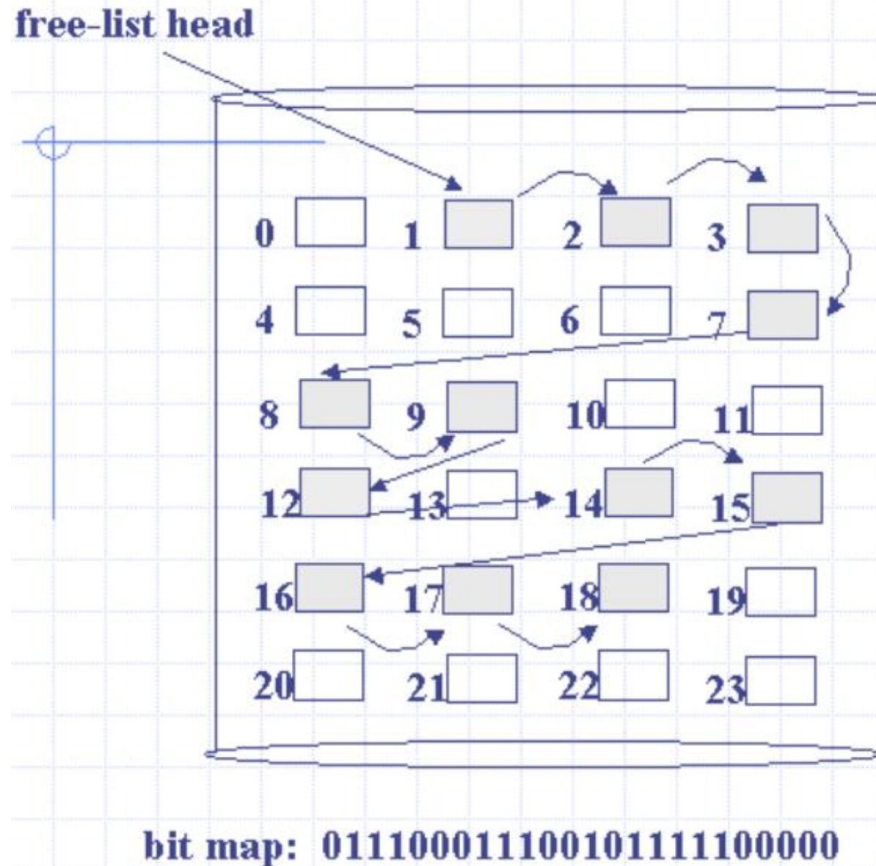
Free-Space Management - Bit Vectors

- Bit vector (n blocks)

  - Each block is represented by 1 bit

  - 1: free   0: allocated

0	1	2					n-1

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

- E.g. 000011111000100010000

Free-Space Management - Bit Vectors

bit map: 011100011100101111100000

Free-Space Management - Grouping

- A modification of the free-list approach

- Stores 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**

- The addresses of a large number of **free blocks** can now be found quickly

Free-Space Management - Counting

- Generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.

- Each entry in the free-space list then consists of a disk address and a count.

Free-Space Management

- **Bit vector** requires extra space

  E.g. Block size = 2^{12} Bytes   disk size = 2^{30} Bytes (1GB) n = 2^{30} / 2^{12} = 2^{18} bits (32KB)
  - Easy to get contiguous files

- **Linked list** (free-list)
  - Cannot get contiguous space easily
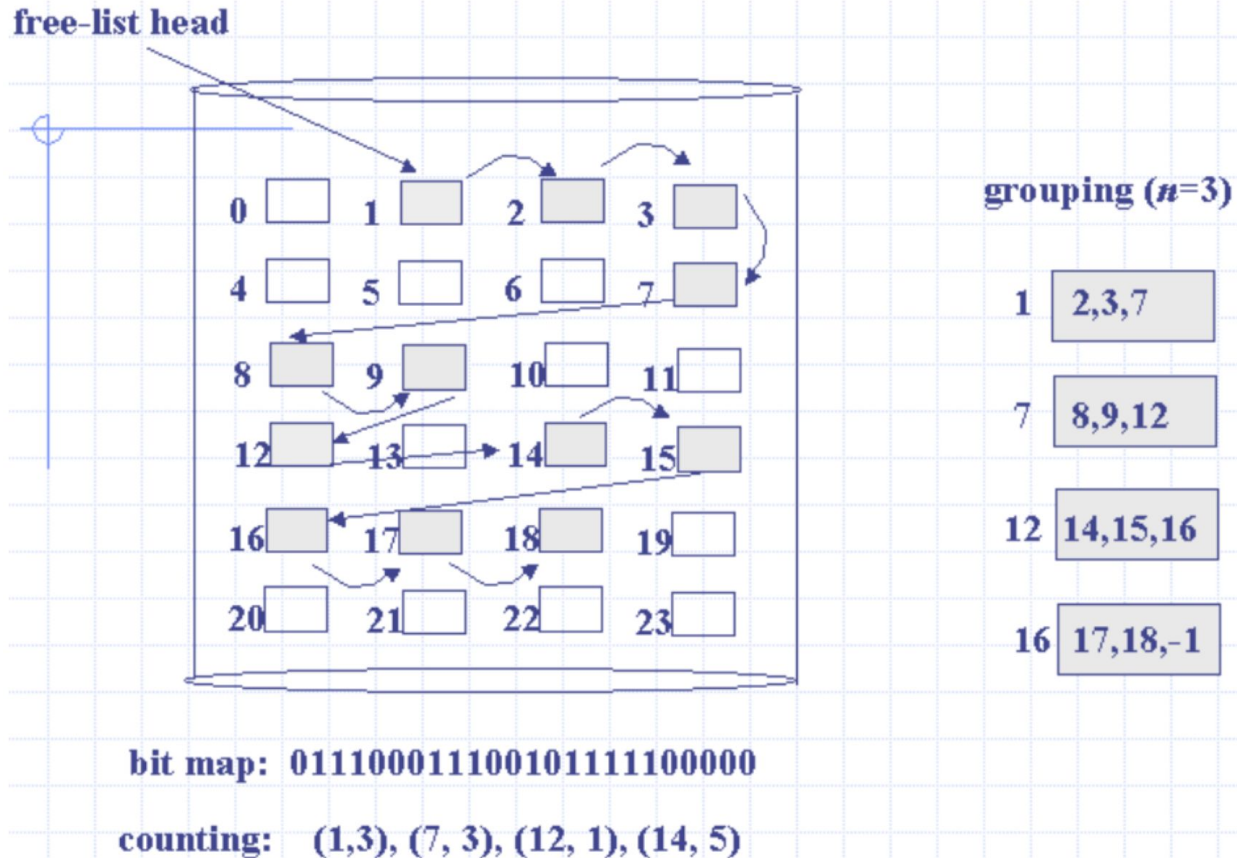 - Requires substantial I/O

- **Grouping**
  - Modification of free-list
  - Store address of n-1 free blocks + next group, in the first free block

- **Counting**
  - Keep the address of the first free block
  - And the number n of free contiguous blocks that follow it

Free-Space Management

free-list head

grouping (n=3)

1	2,3,7
7	8,9,12
12	14,15,16
16	17,18,-1

bit map: 011100011100101111100000

counting: (1,3), (7, 3), (12, 1), (14, 5)

Exercise - 1

In terms of reliability and performance, compare **bit vector** implementation of a free **block list** with keeping a list of free blocks where the first few bytes of each free block provide the logical sector number of the next free block.

Exercise - 1

performance: Bit vector implementation is more efficient since it allows fast and random access to free blocks, linked list approach allows only sequential access, and each access results in a disk read. Bit vector implementation can also find n consecutive free blocks much faster. (Assuming bit vector is kept in memory)

reliability: If an item in a linked list is lost, you cannot access the rest of the list. With a bit vector, only those items are lost. Also, it's possible to have multiple copies of the bit vector since it is a more compact representation. Although keeping the bit vector in memory seem to be unreliable, you can always keep an extra copy on the disk.

Exercise - 2

Given the current file allocation on the disk in this figure, show the linked list, bit-map, counting, and grouping (**n=4**) representations of the free block list.
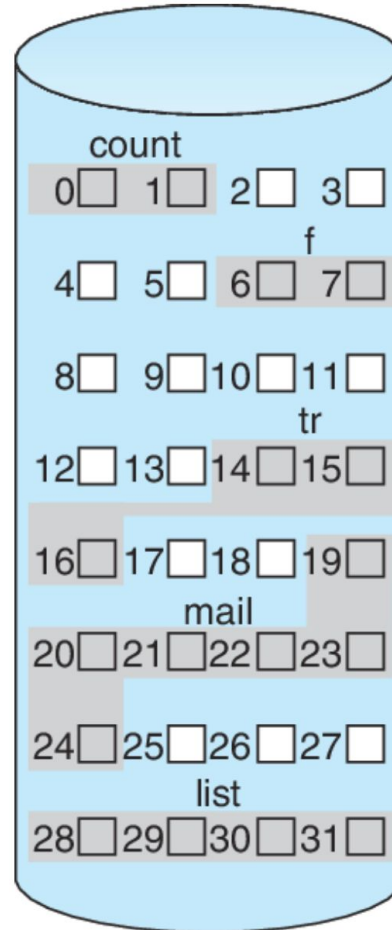
Acknowledgements

- "Operating Systems Concepts" book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne

- "Operating Systems: Internals and Design Principles" book and supplementary material by W. Stallings

- "Modern Operating Systems" book and supplementary material by A. Tanenbaum

- R. Doursat and M. Yuksel from University of Nevada, Reno

- Farshad Ghanei from Illinois Tech

- T. Kosar and K. Dantu from University at Buffalo

Announcement

- Homework 5

  - Released on class website

  - Due on december 8th

- Quiz 6

  - Released on blackboard

  - DUE TOMORROW 11.59 PM