CSE 421/521 - Operating Systems
Fall 2018

Lecture - XVIII

File Systems - I

Tevfik Koşar

University at Buffalo

November 1st, 2018

Roadmap

- File Allocation Methods
  - Contiguous
  - Linked
  - Indexed
- Free Space Management
  - Bit vector
  - Linked List
  - Grouping
  - Counting

File Allocation Methods

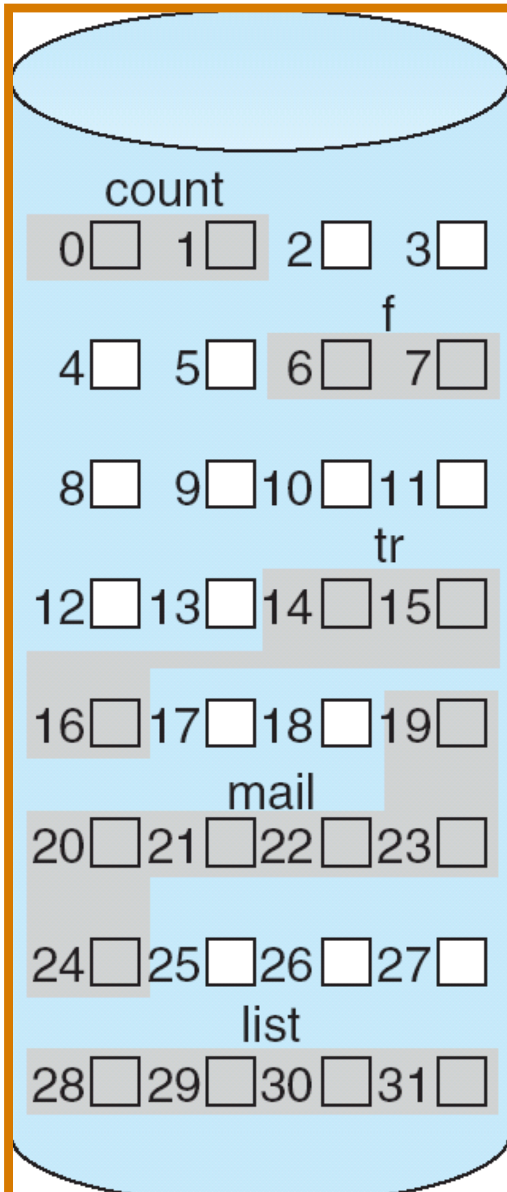
- An allocation method refers to **how disk blocks are allocated & organized for files on the disk:**
- Contiguous allocation
- Linked allocation
- Indexed allocation

1. Contiguous Allocation

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

- Contiguous allocation – 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
- Problems include finding space for file, knowing file size,

Contiguous Allocation of Disk Space



directory

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

Contiguous Allocation Address Translation

- Mapping from logical to physical disk address, given:
    - LA: logical address
    - 512: disk block size
    - Q: quotient
    - R: remainder

Q

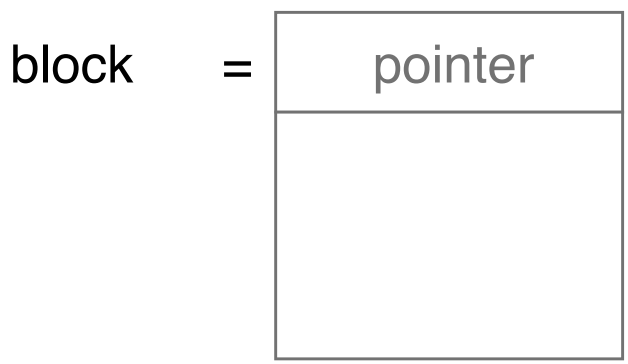
LA/512

R

Block to be accessed = Q + starting address
Displacement into block = R

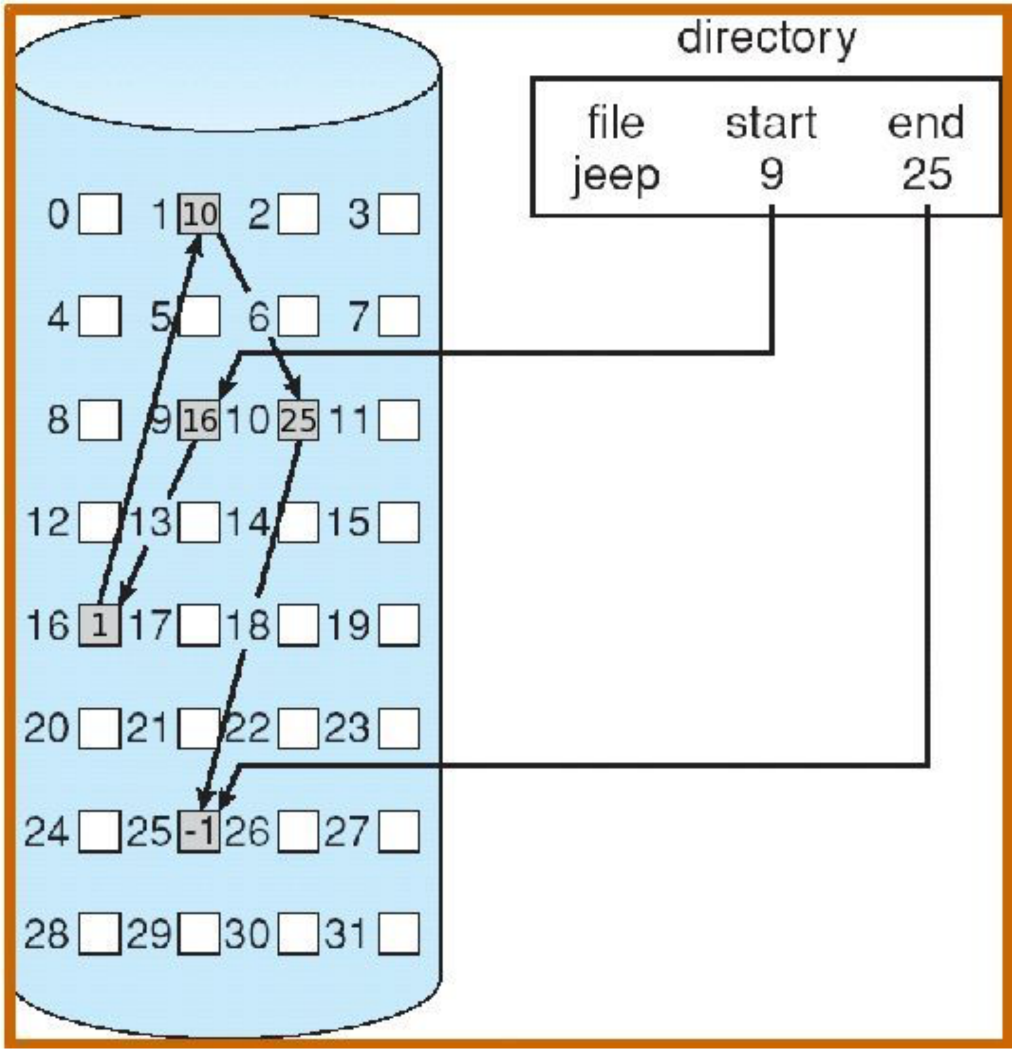
2. Linked Allocation

- 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
+ Defragmentation not necessary for file allocation
- No random access, locating a block can take many disk seek & I/O
- Extra space required for pointers
- Reliability: what if a pointer gets corrupted?

Linked Allocation



Linked Allocation

- Mapping (Assuming pointer takes only one byte)

Q

LA/511

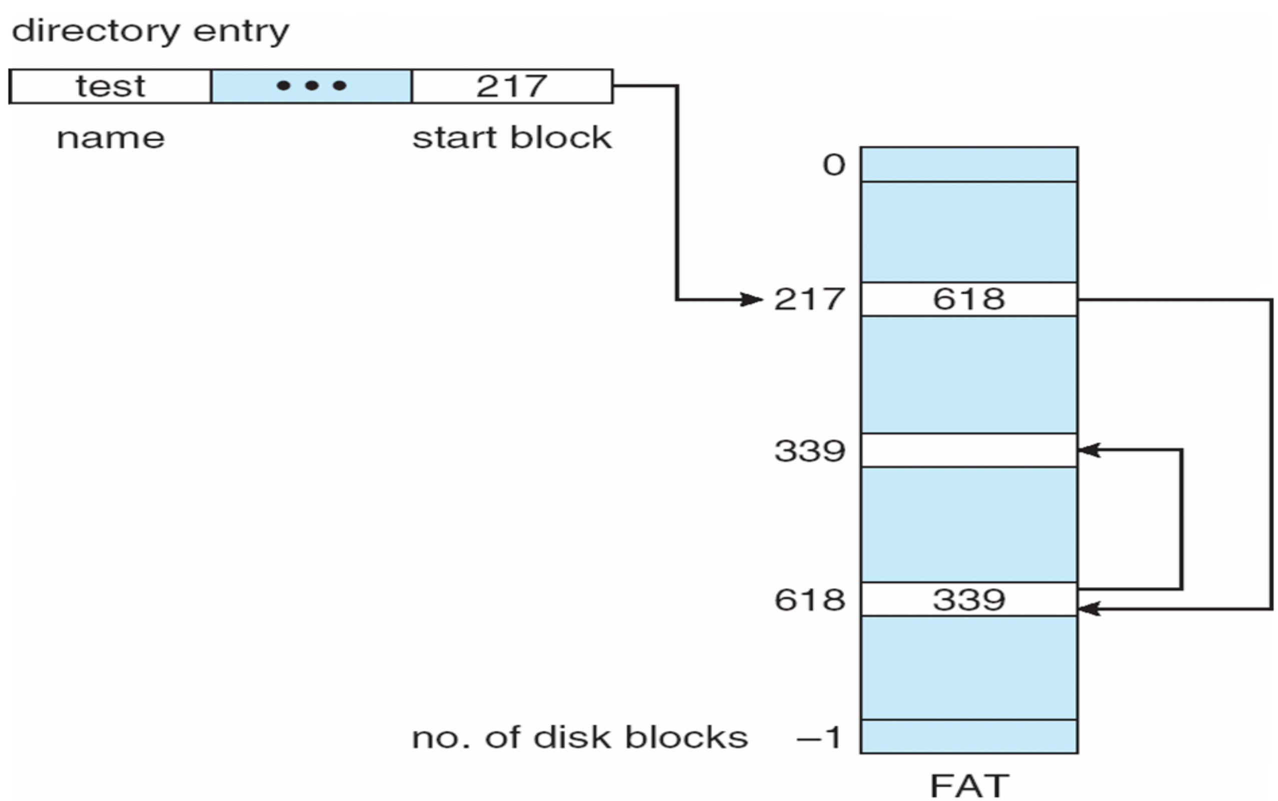
R

Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

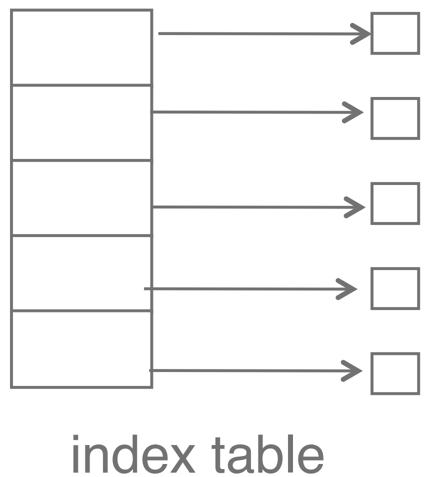
File-Allocation Table

- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple



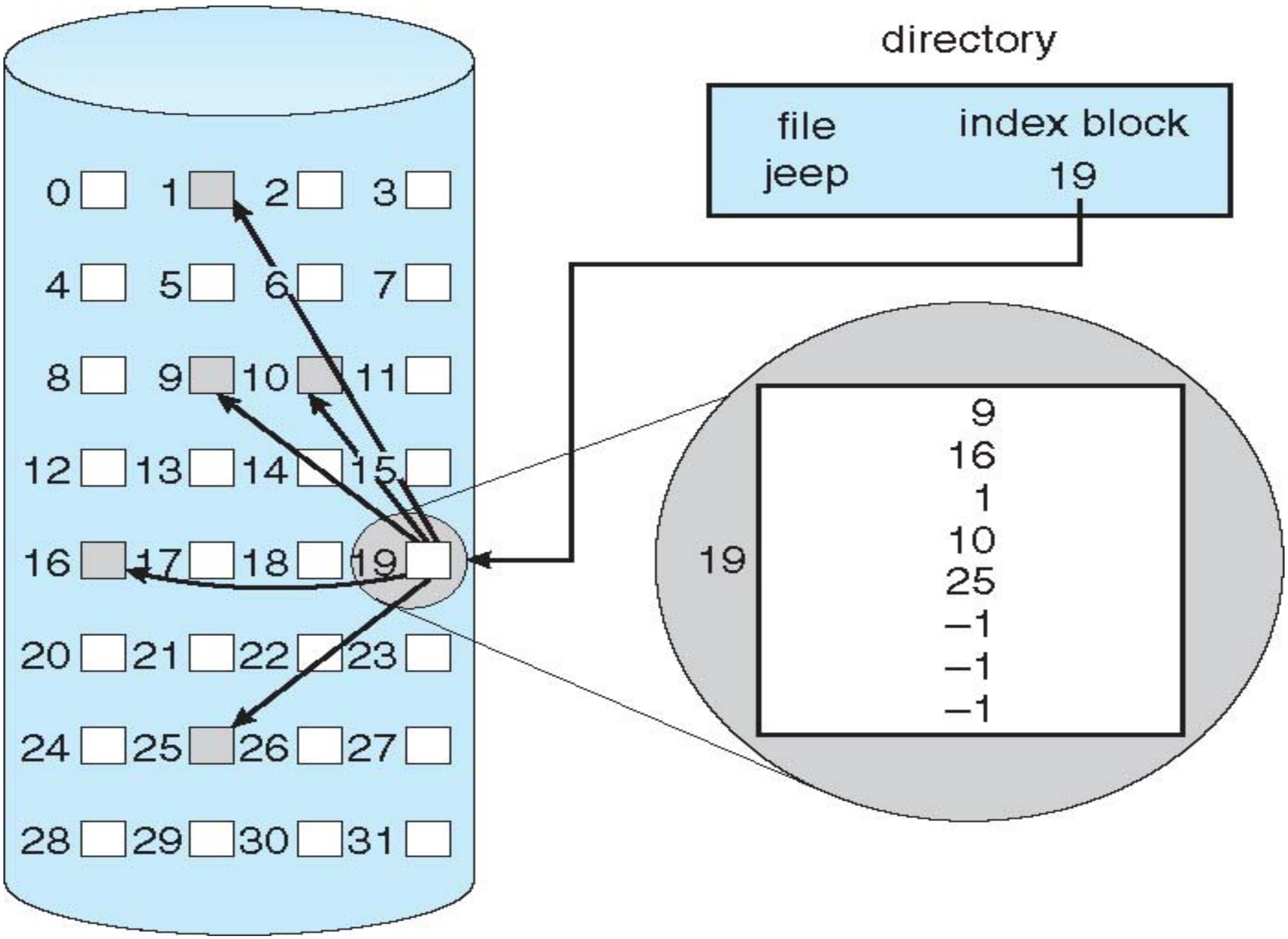
3. Indexed Allocation

- Brings all pointers together into the *index block*, to allow random access to file blocks.
- Logical view.



+ Supports direct access
+ Prevents external fragmentation
- High pointer overhead --> wasted space

Example of Indexed Allocation



Indexed Allocation

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

Q

LA/512

R

Q = displacement into index table
R = displacement into block

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
  - Clustering can help improve throughput, reduce CPU overhead

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:

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?

PS: Assume a pointer needs only one byte space.

Exercise - Solution

a. Contiguous:

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

1. Add *X* to *Z* to obtain the physical block number. *Y* is the displacement into that block.

2. One.

Exercise - Solution

b. Linked:

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

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

2. Four.

Exercise - Solution

c. Indexed:

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

1. 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.

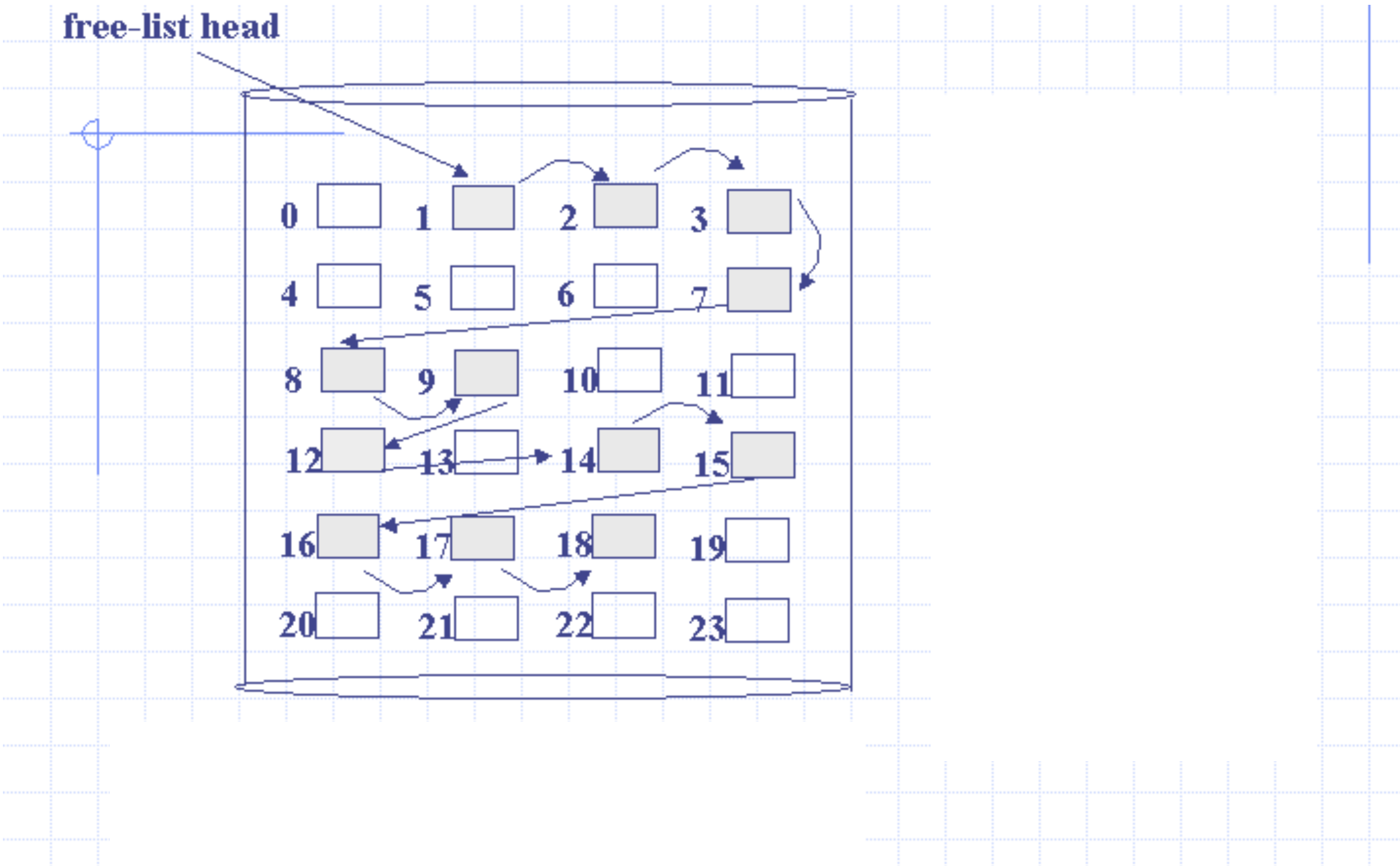
2. Two

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
  - Bit vectors
  - Linked lists

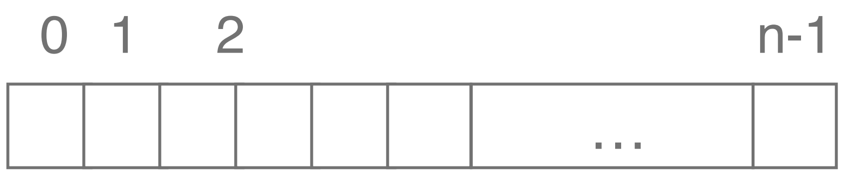
Free-Space Management (Cont.)

- Linked List Approach



Free-Space Management (Cont.)

- Bit vector   (*n* blocks)
  - Each block is represented by 1 bit
  - 1: free, 0: allocated

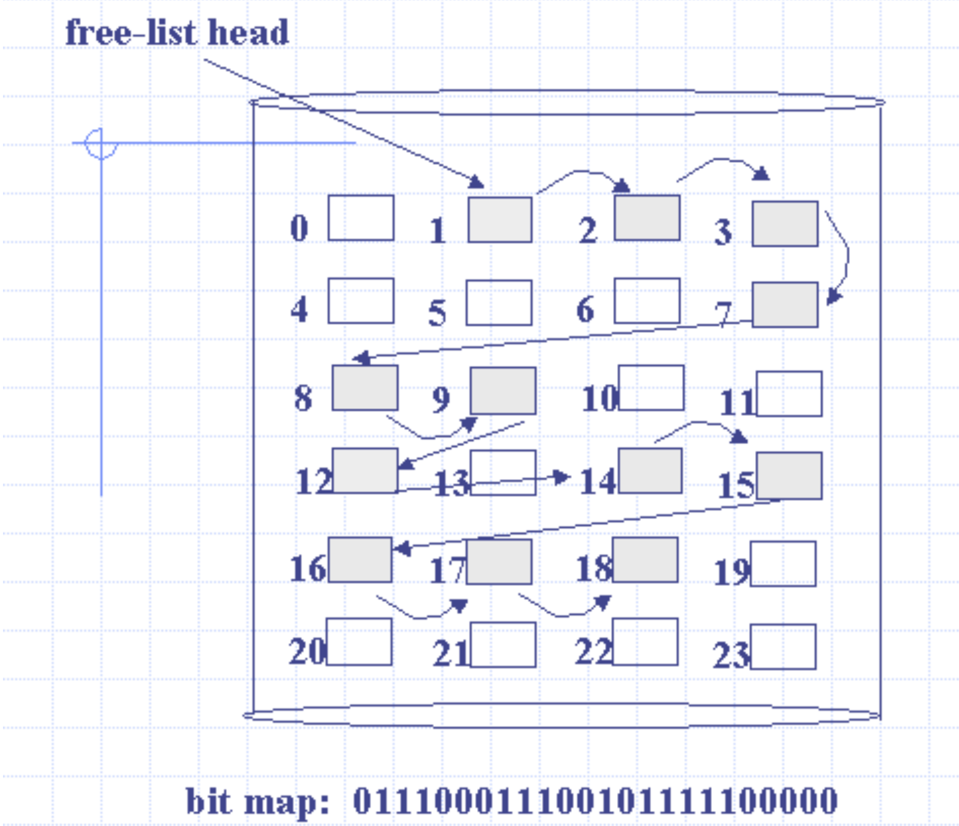


$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- e.g. 00001111100010001000010000

Free-Space Management (Cont.)

- Bit Vector Approach



Free-Space Management (Cont.)

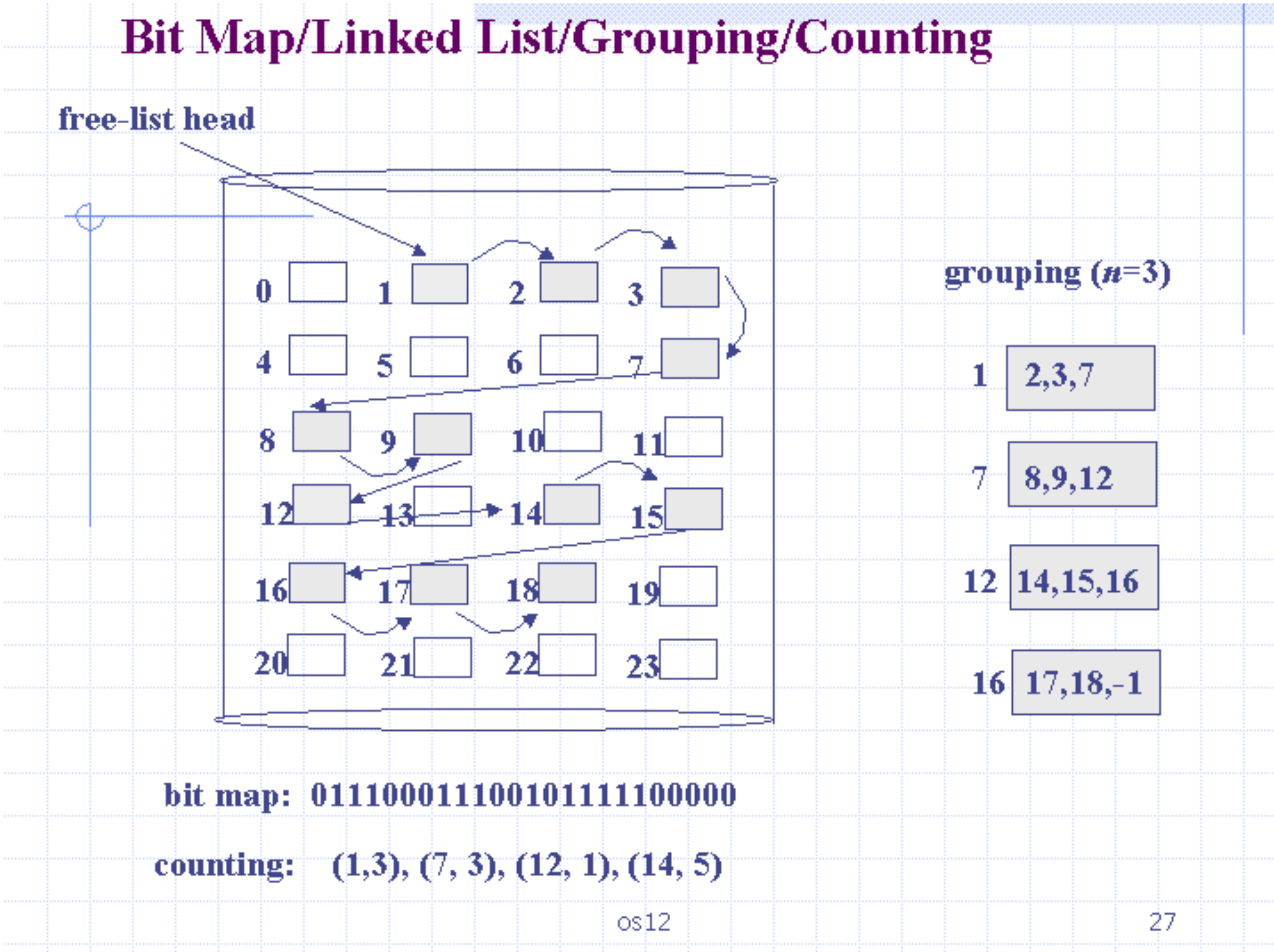
- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

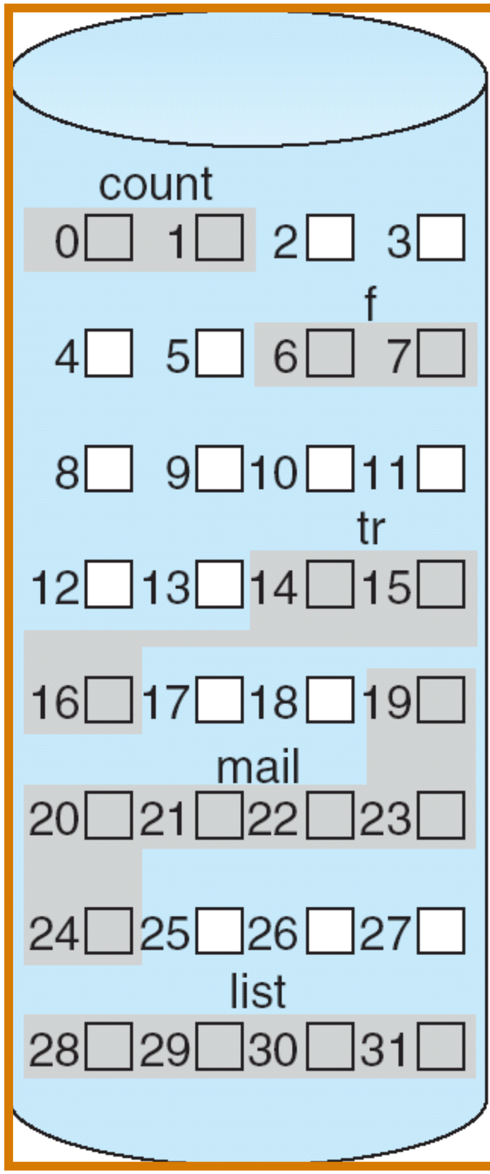
    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - requires substantial I/O
- Grouping
  - Modification of free-list
  - Store addresses of n free blocks in the first free block
- Counting
  - Rather than keeping list of n free addresses:
    - Keep the address of the first free block
    - And the number n of free contiguous blocks that follow it

Free-Space Management (Cont.)



Exercise - 1



directory

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

Exercise - 2

- 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.

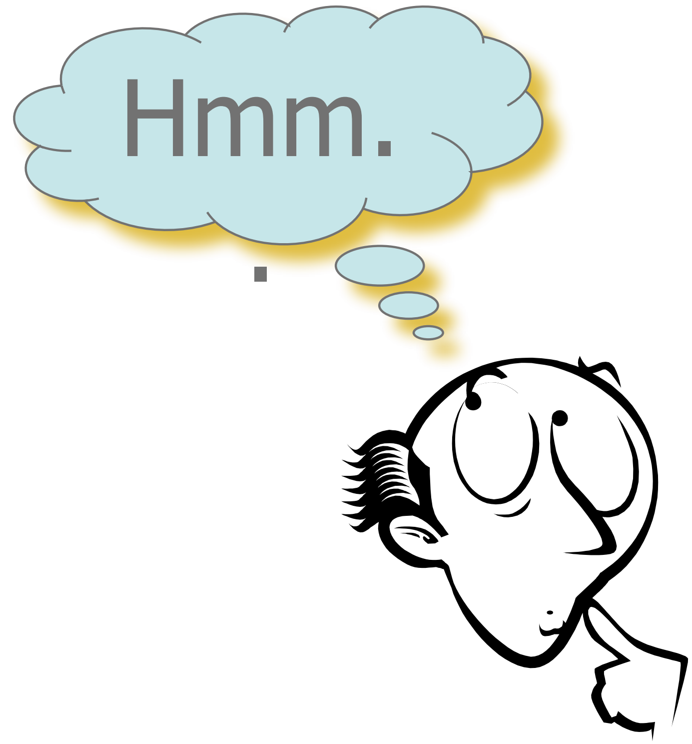
Exercise - 2 Solution

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, its 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.

Any Questions?

- File Allocation Methods
  - Contiguous
  - Linked
  - Indexed
- Free Space Management
  - Bit vector
  - Linked List
  - Grouping
  - Counting



Reading assignment: Chapter 12 from Silberschatz

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 UNR