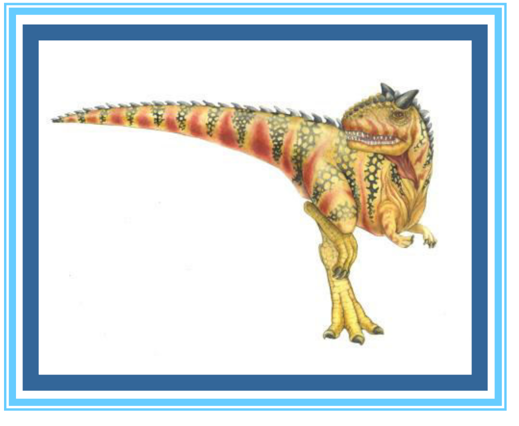
Chapter 14: File-System Implementation



Chapter 14: File-System Implementation

- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery

Objectives

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

Disk Space Allocation Methods

- A disk is a direct-access device and thus gives us flexibility in the implementation of files.
- The main issue is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating disk space are in wide
  - Contiguous
  - Linked
  - Indexed

Contiguous Allocation

Each file occupies a set of contiguous blocks

- Best performance in most cases
- Simple – only starting location (block #) and length (number of blocks) are required
- Problems include:
  - Finding space for file,
  - Knowing the max file size,
  - External fragmentation,
    - Need for compaction
      - Off-line (downtime) or
      - On-line

Contiguous Allocation (Cont.)

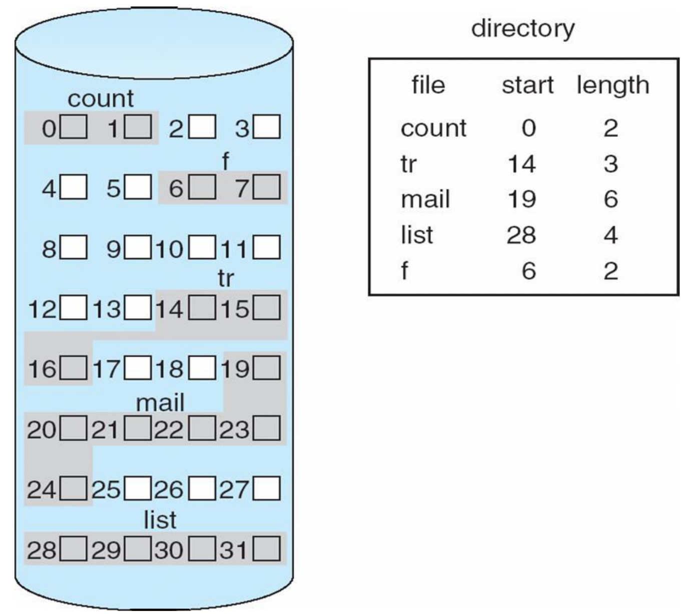
- Mapping from logical to physical (assume block size if 512)

Q

LA/512

R

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



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 chunks called extents.
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

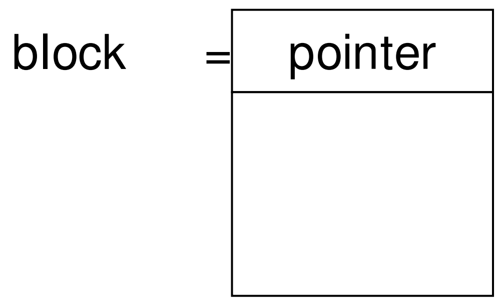
Linked Allocation

Each file is a linked list of blocks

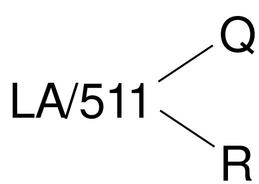
- No external fragmentation
- Each block contains pointer to next block
- Free space management system called when new block needed
- Locating a block can take many I/Os and disk seeks
- Reliability can be a problem (what if one of the pointers get corrupted?)
- Improve efficiency by clustering blocks into groups but increases internal fragmentation

Linked Allocation (Cont.)

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

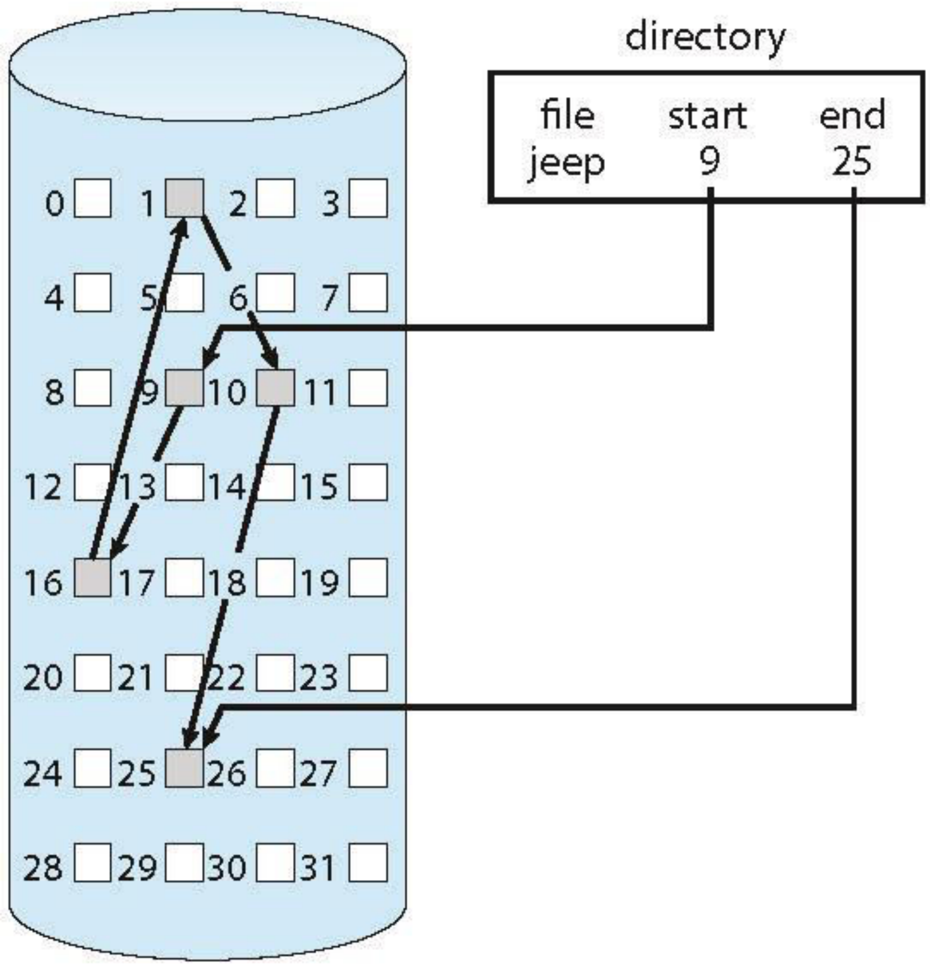


- Mapping:



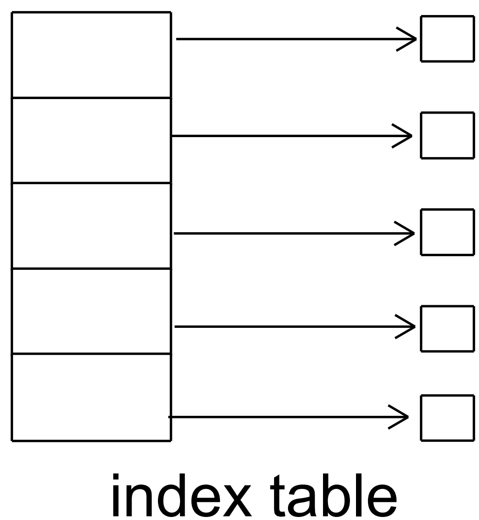
  - Block to be accessed is the Qth block in the linked chain of blocks representing the file
  - Displacement into block = R + 1

Linked Allocation (Cont.)

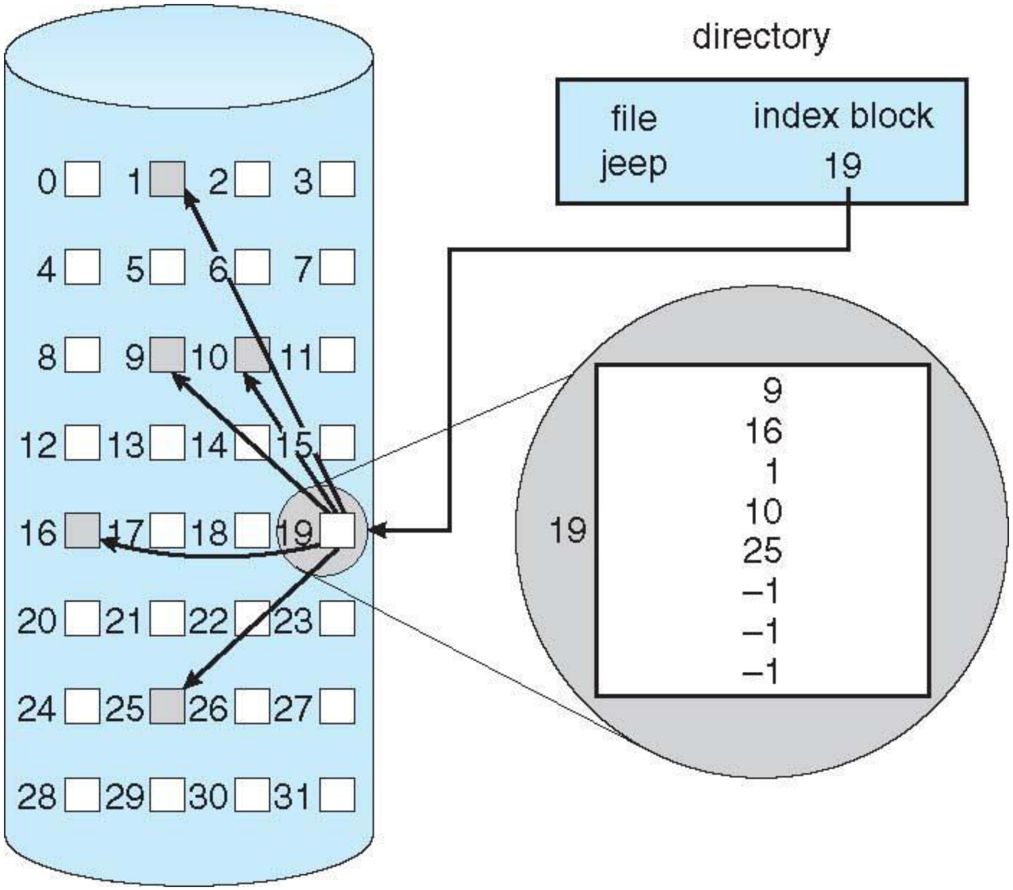


Indexed Allocation

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



Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need an 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

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

  - Q = displacement into index table
  - R = displacement into block

Indexed Allocation – Large Files

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
- Can be accomplished with 3 different methods:
  - Linked scheme.
  - Multilevel index
  - Combined scheme.

Indexed Allocation – Linked Scheme

Link the blocks of an index table

Q_1

$LA / (512 \times 511)$

R_1

Q_1 = block of index table

R_1 is used as follows:

Q_2

$R_1 / 512$

R_2

Q_2 = displacement into block of index table

R_2 displacement into block of file:

Indexed Allocation – Multilevel Index

Two-level index (4K blocks could store 1,024 four-byte pointers in outer index → 1,048,567 data blocks and file size of up to 4GB)

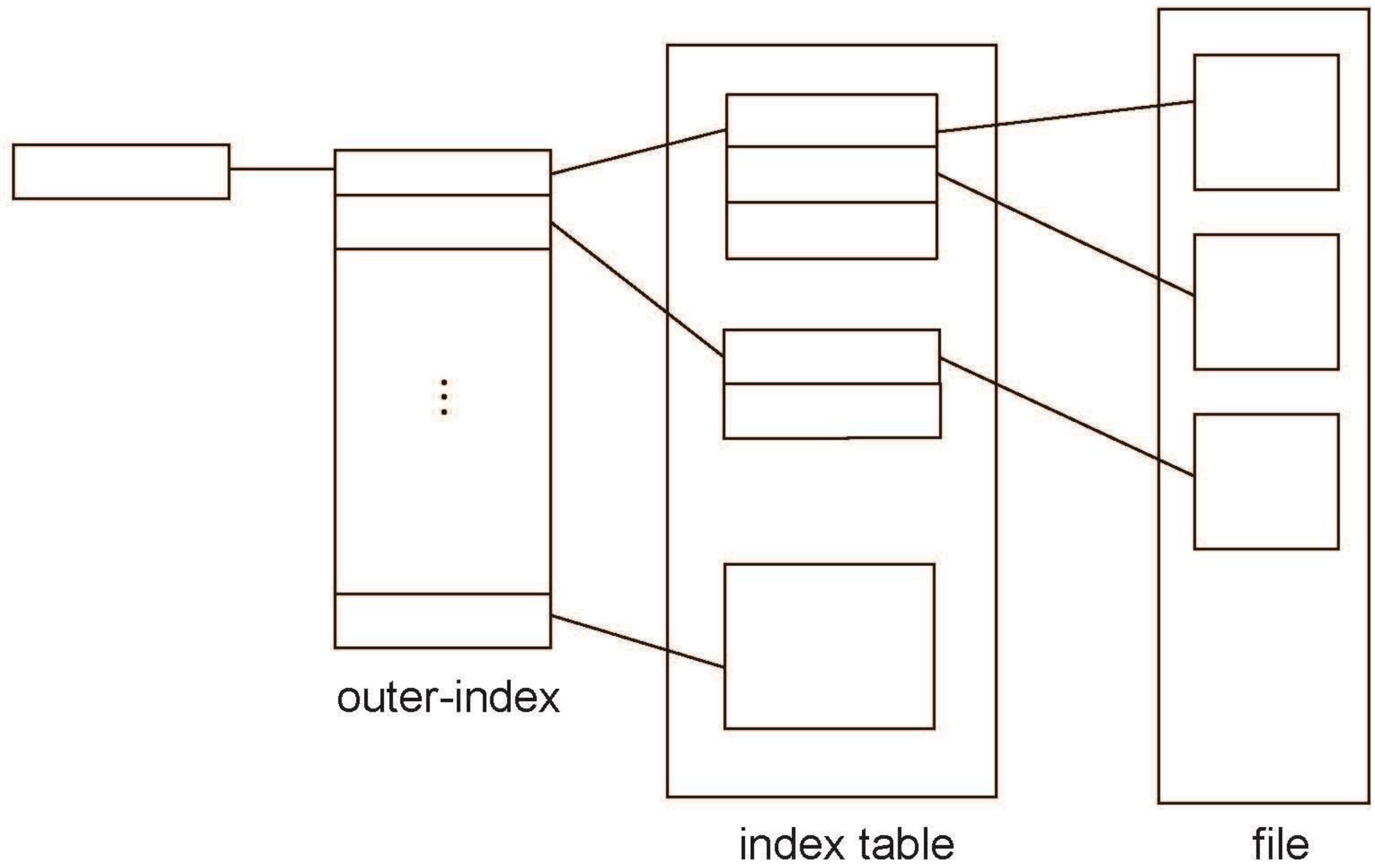
$$LA / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

Q_1 = displacement into outer-index
R_1 is used as follows:

$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

Q_2 = displacement into block of index table
R_2 displacement into block of file:

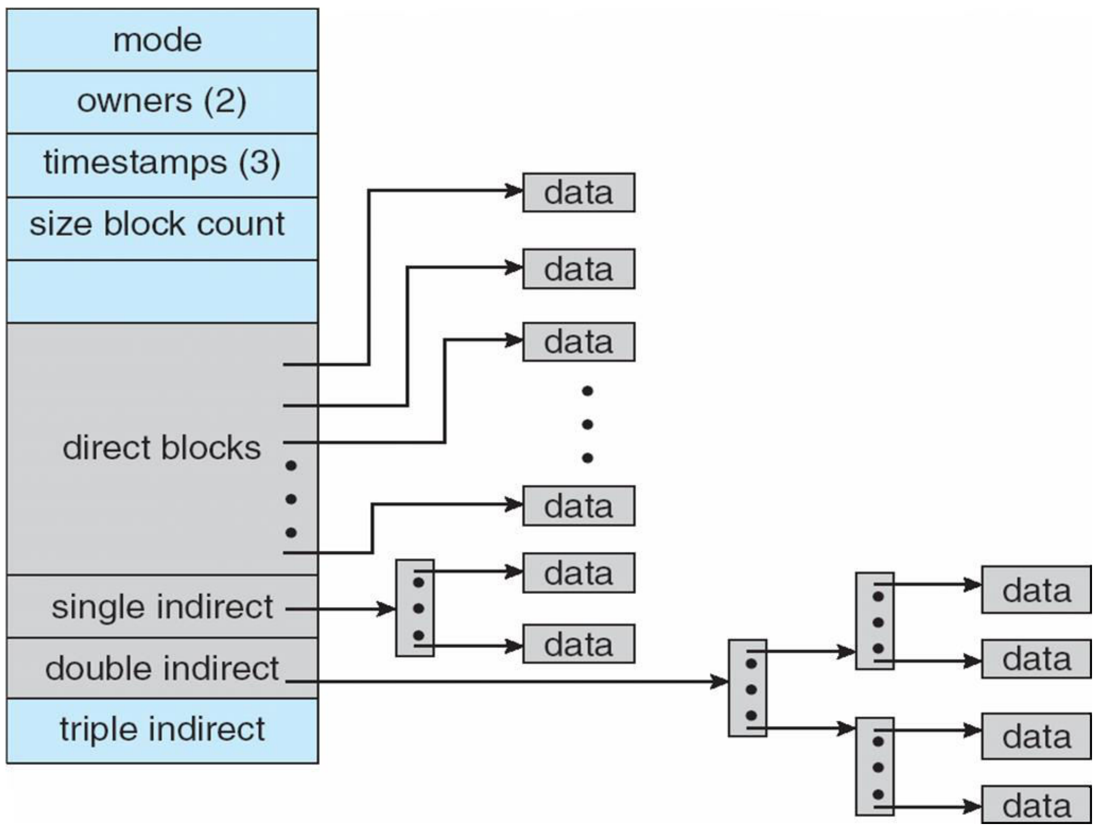
Indexed Allocation – Multilevel (Cont.)



Combined Scheme: UNIX UFS

- 4K bytes per block, 32-bit addresses
- Keep the first 15 pointers of the index block in the file's inode
- The first 12 of these point to direct blocks; i.e., they contain addresses of the first 12 data blocks of the file. No need for a separate index block small files (of no more than 12 blocks). Up to 48 KB of data can be accessed directly.
- The next three pointers point to indirect blocks.
  - The first points to a single indirect block, which is an index block containing not data but the addresses of blocks that do contain data.
  - The second points to a double indirect block, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks.
  - The last pointer contains the address of a triple indirect block.

Combined Scheme: UNIX UFS Example



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

Performance (Cont.)

- Adding instructions to the execution path to save one disk I/O is reasonable
  - Intel Core i7 Extreme Edition 990x (2011) at 3.46Ghz = 159,000 MIPS
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - Typical disk drive at 250 I/Os per second
    - 159,000 MIPS / 250 = 630 million instructions during one disk I/O
  - Fast SSD drives provide 60,000 IOPS
    - 159,000 MIPS / 60,000 = 2.65 millions instructions during one disk I/O

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term “block” for simplicity)
- Four different methods:
  - Bit map
  - Free list
  - Grouping
  - Counting

Free-Space Management – bit map

- **Bit vector** (or **bit map**) (***n*** blocks)

0	1	2					***n***-1
						…	

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

Block number calculation

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

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

Free-Space Management – bit map

- **Bit vector** (or **bit map**) (***n*** blocks)

0	1	2					***n***-1
						…	

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

Block number calculation

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

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

Bit Map (Cont.)

- Bit map requires extra space
  - Example:

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

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

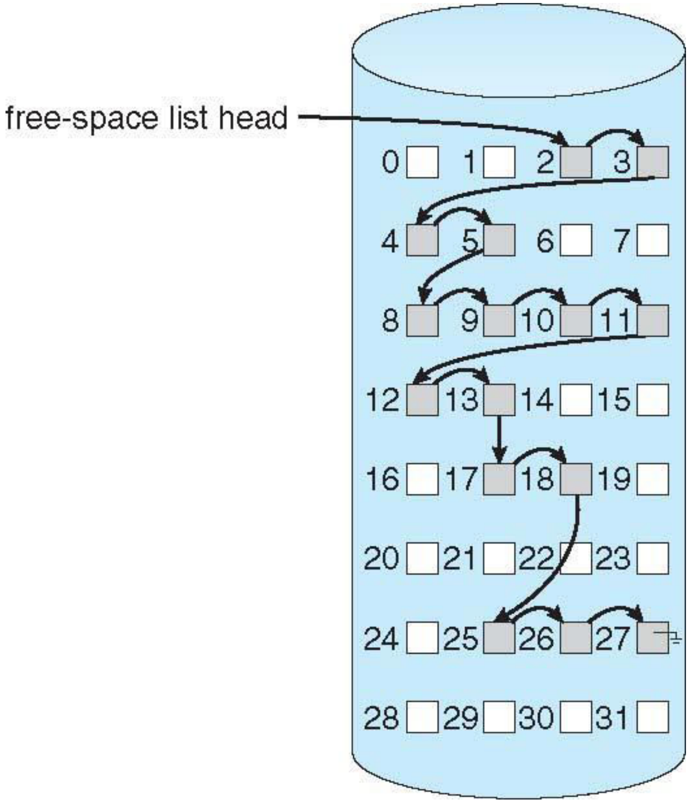
    $\boldsymbol{n} = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- But -- easy to get contiguous files

Free List

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Grouping and Counting

- Grouping
  - 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 (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

End of Chapter 14