

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 39



File-system Implementation

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

File Systems



"MS. GRIMMETT, I SORT OF LIKED THE OLD FILING SYSTEM...IN THE FILE CABINETS."

FAQ

- Can a set of files be mounted to the top of a tree
 - What you mount is a file system on a device, which is attached to a mount point in another file system.
- Can windows mount files from linux and vice versa?
 - Samba, auto mount
- How do you access data when one drive in RAID 0 fails?
 - RAID 0: non-redundant, striping
- What is the advantage of a journaling file system? Records changes not committed. Can resolve inconsistencies due to crashes.
- How much history do journaling file systems keep?

Notes

- Poster Session Dec 9 10-noon
 - See Assignments page, Canvas
- You can get posters printed CNS Lab in
 - Anatomy/Zoology E100 (CNS Undergrads) or
 - Morgan Library (account number from Kim Judith, Accounting, 266 CSB)
 - I recommend getting it printed sometime on Dec 7.

Allocation Methods

Allocation Methods

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

- Contiguous (not common, except for DVDs etc.)
- Linked (e.g. FAT32)
- Indexed (e.g. ex4)

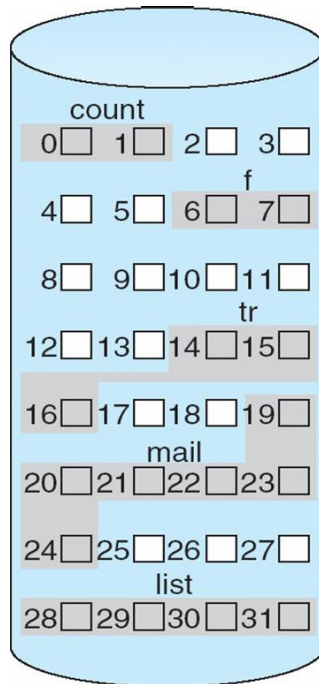
Thoughts about optimization:
- Sequential vs random access
- What is the common file size?

Allocation Methods – i.Contiguous

i. Contiguous allocation – each file occupies set of contiguous blocks

- Simple – only starting location (block #) and length (number of blocks) are required
 - Occupies n block $b, b+1, \dots, b+n-1$
- Minimal disk head movement
- Problems include finding space for file, knowing file size, **external fragmentation**, need for **compaction off-line (downtime)** or **on-line**

Contiguous Allocation



directory

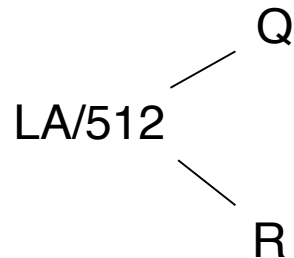
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

File **tr**: 3 blocks
Starting at block 14

Contiguous Allocation

- Mapping logical byte address LA to physical address

Assume block size = 512



Block to be accessed = starting block number (address) + Q

Displacement into block = R

Extent-Based Systems

- Some file systems use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
 - An **extent** is a contiguous block
 - Metadata: beginning block, number of blocks
 - Extents are allocated for file allocation
 - A file consists of one or more extents

Allocation Methods - Linked

ii. **Linked allocation** – each file a linked list of blocks

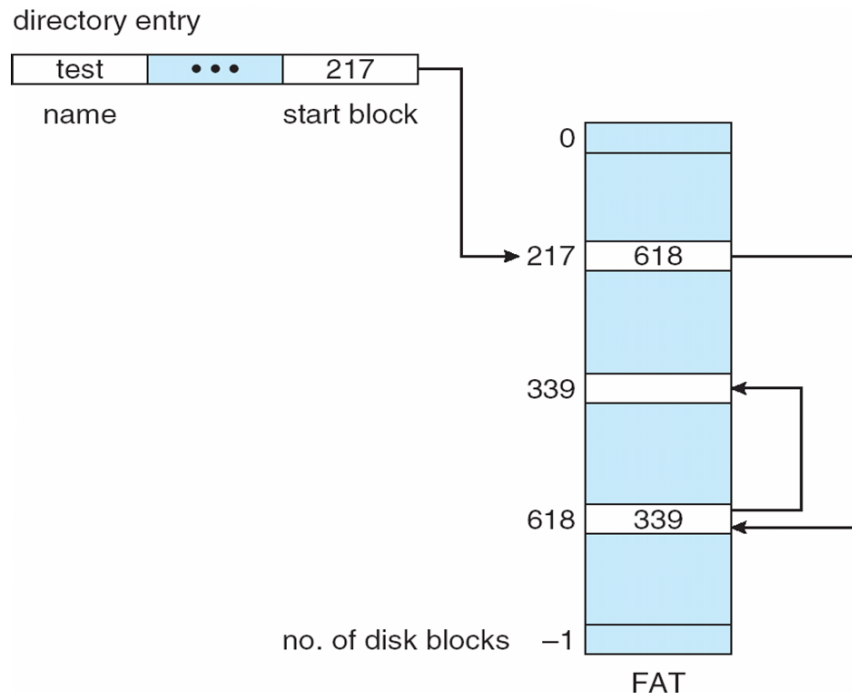
- Each block contains pointer to next block.
- File ends at null pointer
- No external fragmentation, no compaction

Free space management system called when new block needed

- Locating a block can take many I/Os and disk seeks.
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem, since every block in a file is linked

Allocation Methods – Linked (Cont.)

- 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



Each FAT entry corresponds to the corresponding block of Storage.

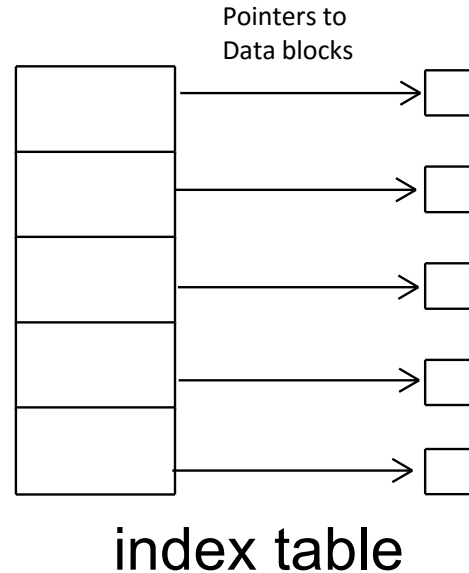
Free block entries are also linked.

Allocation Methods - Indexed

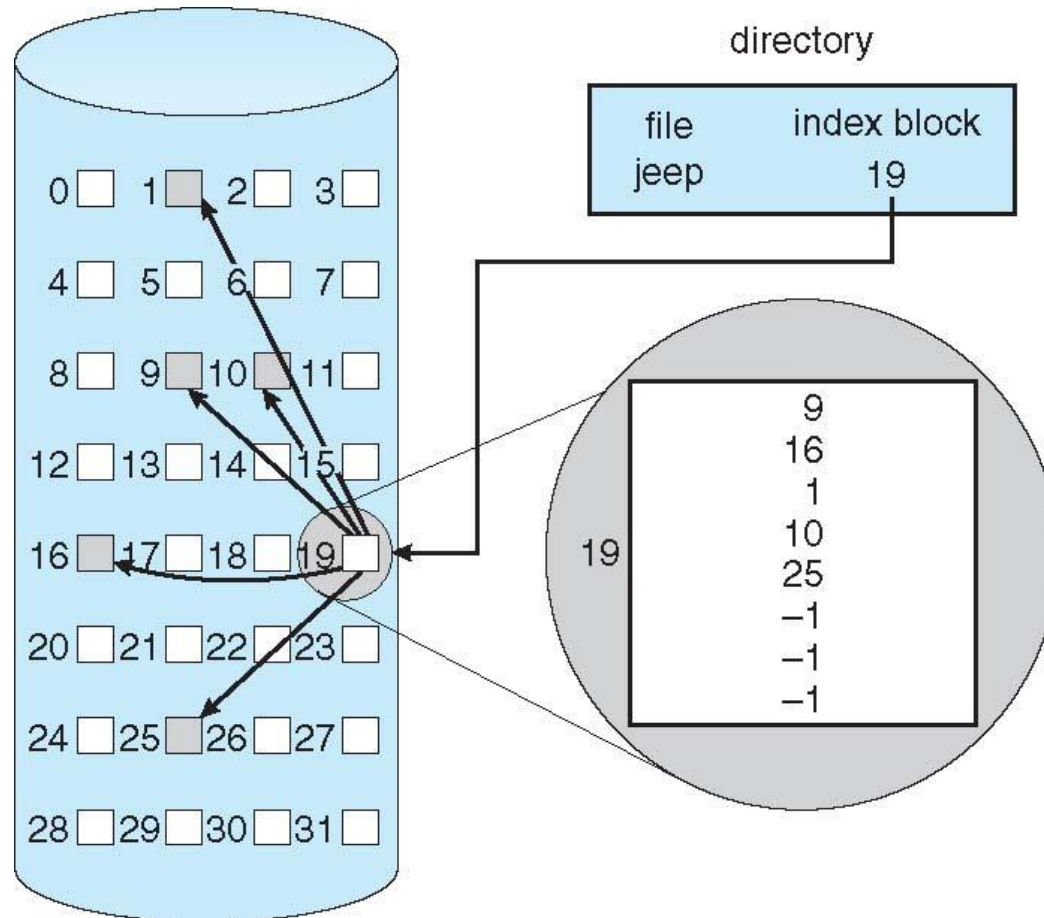
- Indexed allocation

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

- Logical view



Example of Indexed Allocation

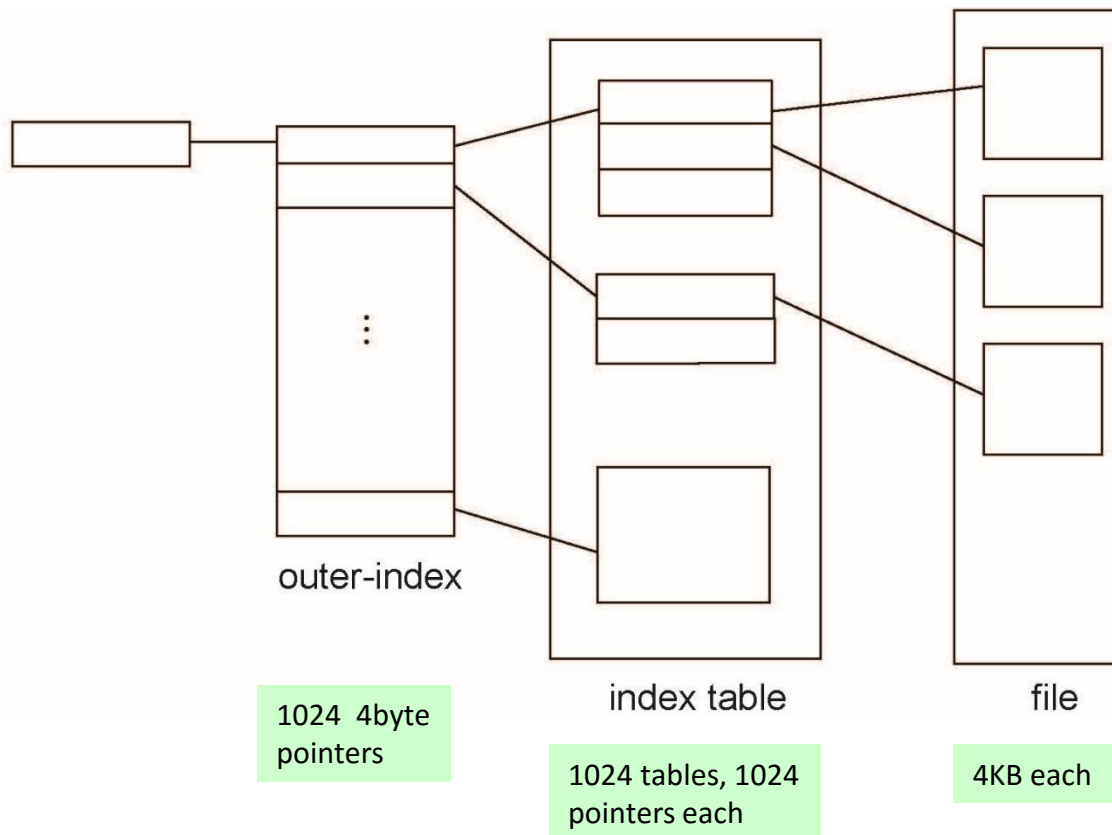


Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block even for a small file
- Assuming pointer size is 1 byte, block is 512 bytes
- 1 block for index table can be used for a file of maximum size of $512 \times 512 = 256\text{K}$ bytes
- Larger files? Multi-block index table?

Indexed Allocation – Two level

- **Two-level index: Ex:** 4K blocks could store 1,024 four-byte pointers in outer index - > 1024x1024 data blocks and file size of up to 4GB



Most files are ..

- Most file are small, but the few very large files* occupy a large fraction of the space
- Optimize for small files, but allow for large files also

* Y. K. Malaiya and J. Denton " Module Size Distribution and Defect Density," Proc. IEEE International Symposium on Software Reliability Engineering, Oct. 2000, pp. 62-71

Inode idea

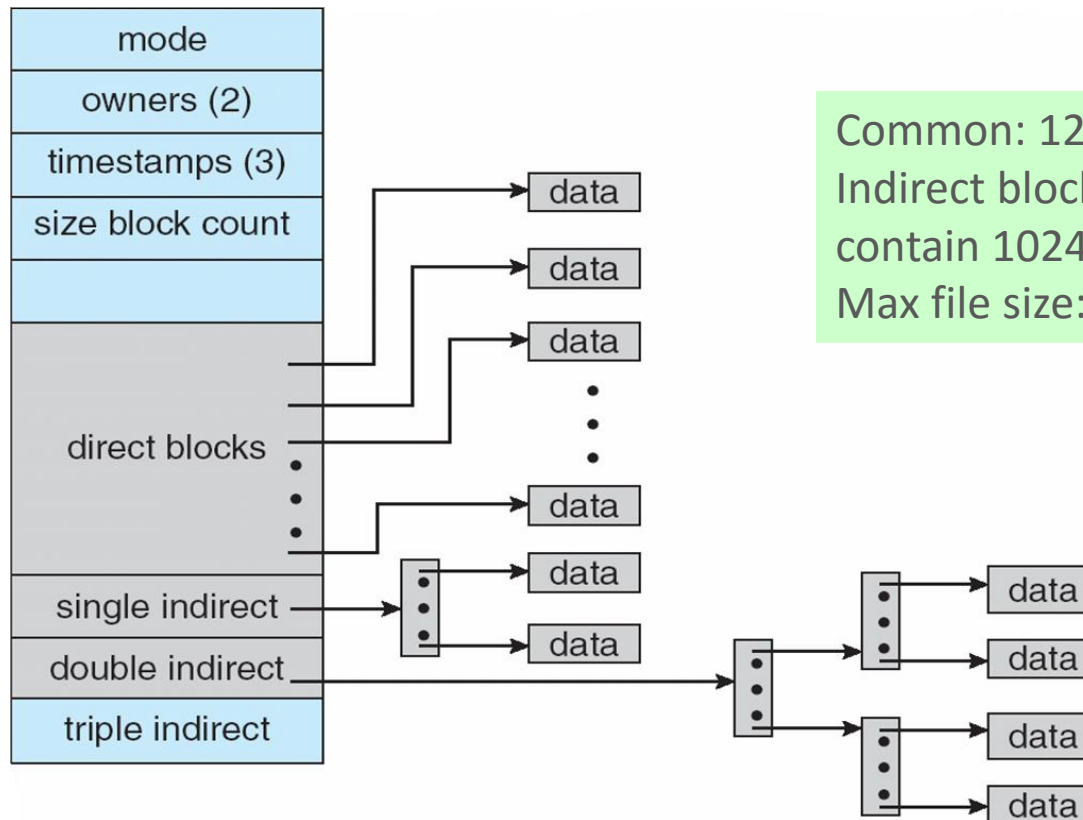
Inode: hold file metadata including pointers to actual data

- Small file stored in the inode itself. Typical inode size 128 bytes
- Medium size: direct pointers to data
- If the file is large: Indirect pointer
 - To a block of pointers that point to additional data blocks
- If the file is larger: Double indirect pointer
 - Pointer to a block of indirect pointers
- If the file is huge: Triple indirect pointer
 - Pointer to a block of double-indirect pointers

Combined Scheme: UNIX inodes

Assume 4K bytes per block, 32-bit addresses

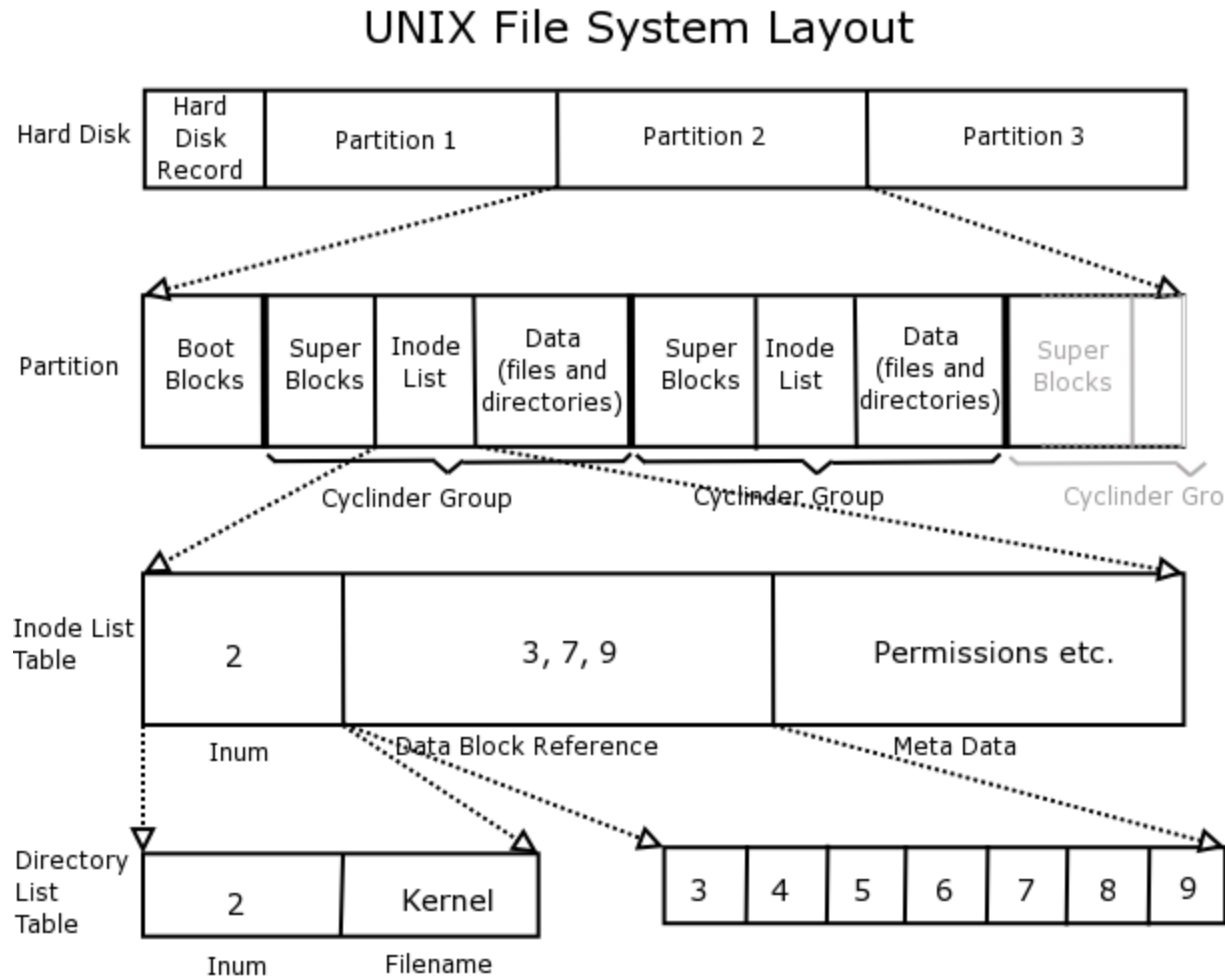
Volume block:
Table with file names
Points to this inode
(file control block)



Common: 12 direct+3.
Indirect block could
contain 1024 pointers.
Max file size: k.k.k.4k+

More index blocks than can be addressed with 32-bit file pointer

On-disk layout of a typical UNIX file system



Performance

- Best method depends on file access type
 - Contiguous great for sequential and random
- Linked good for sequential, not random
- Indexed more complex
 - Single block access could require 0-3 index block reads then data block read
 - Clustering can help improve throughput, reduce CPU overhead

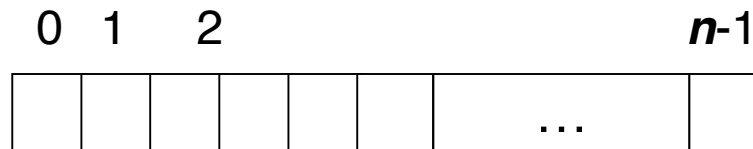
Cluster: set of contiguous sectors

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
 - $159,000 \text{ MIPS} / 250 = 630$ million instructions during one disk I/O
 - Fast SSD drives provide 60,000 IOPS
 - $159,000 \text{ MIPS} / 60,000 = 2.65$ 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)
- **Approaches: i. Bit vector ii. Linked list iii. Grouping iv. Counting**
- **Bit vector** or **bit map** (n blocks)



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

Block number calculation for first free block

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

```
00000000
00000000
00111110
..
```

CPUs may instructions to return offset within word of first “1” bit

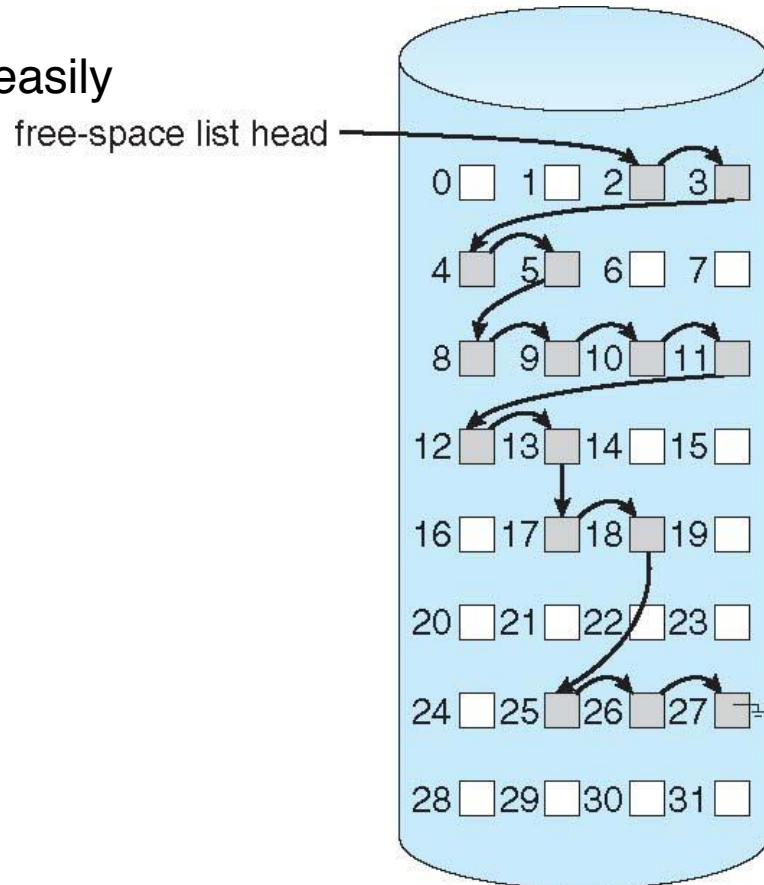
Free-Space Management (Cont.)

- Bit map requires extra space
 - Example:
 - block size = 4KB = 2^{12} bytes
 - disk size = 2^{40} bytes (1 terabyte)
 - blocks: $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB) for map
 - if clusters of 4 blocks -> 8MB of memory
- Easy to get contiguous files if desired

Linked Free Space List on Disk

- ii. Linked list (free list)
 - Cannot get contiguous space easily
 - No waste of space

Superblock Can hold
pointer to head of
linked list



Free-Space Management (Cont.)

- iii. 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
- iv. 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**