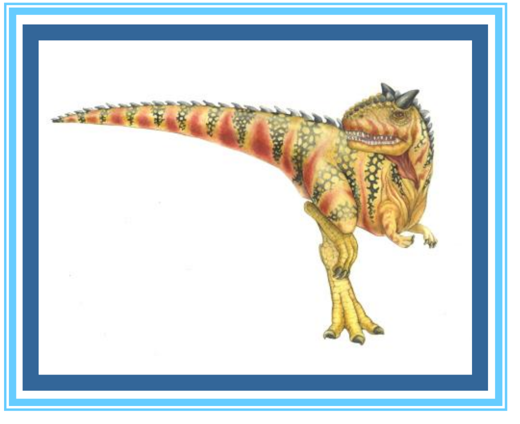
Chapter 14: File System Implementation

Section 14.1,14.4-14.5.1,14.5.2



File-System Structure

- **File system** resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located, and retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in units of **blocks**
  - Each block has one or more **sectors** (usually 512 bytes)
- File system translates file name into file number and location by maintaining directory structure and **file control block** (**FCB**) (**inodes** in UNIX)

File Control Block

- Per-file **File Control Block** (**FCB**) contains many details about the file
  - typically inode number, permissions, size, dates

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

- **system-wide open-file table** contains a copy of the FCB of each file and open count
- **per-process open-file table** contains pointers to appropriate entries in system-wide open-file table as well as other info (e.g., file pointer, access mode)

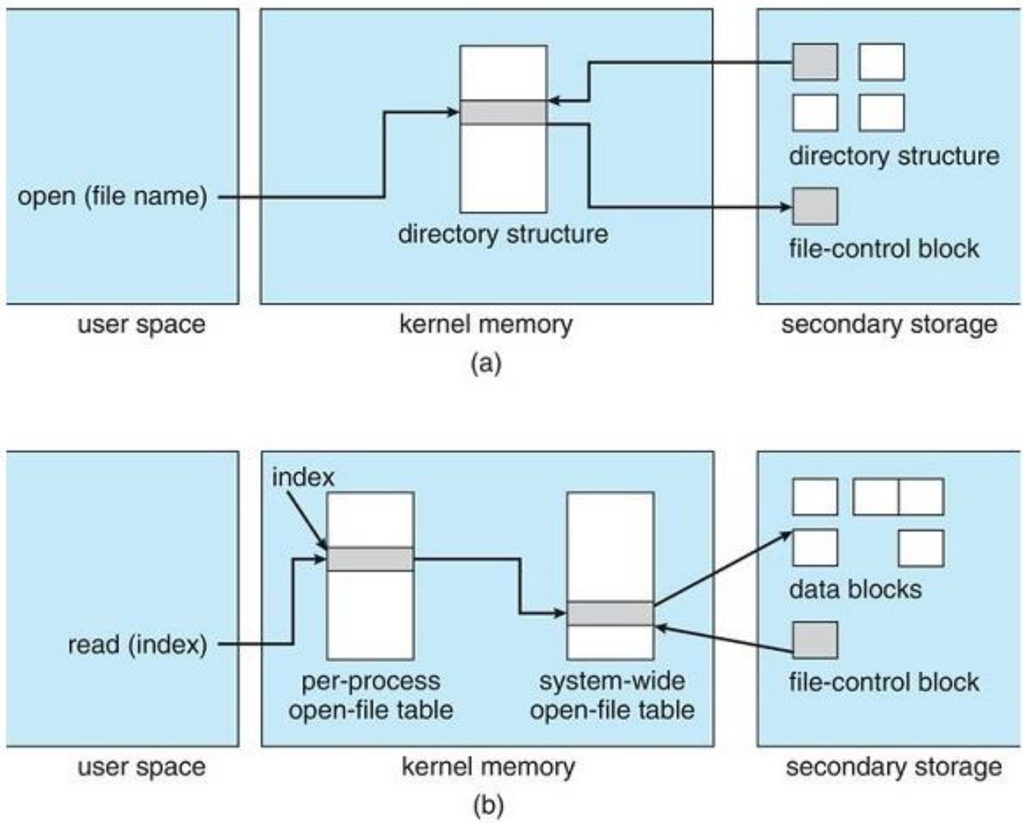


Figure 14.3 In-memory file-system structures. (a) File open. (b) File read.

File Structure

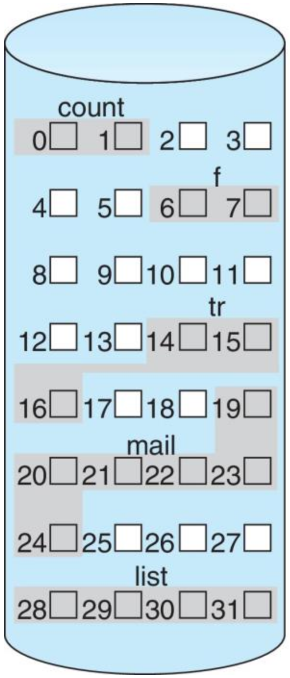
- A file is composed of contiguous **logical blocks**
    - Each logical block has a fixed size
- Secondary storage is composed of **physical blocks**, each having the same size as a logical block
- A file's logical blocks are mapped to physical blocks

Allocation Methods

- An allocation method determines how disk blocks are allocated to files
  - Should utilize disk space efficiently and allow files to be accessed quickly
- Three allocation methods
  - **Contiguous allocation**
  - **Linked allocation**
  - **Indexed allocation**

Contiguous Allocation

- Each file occupies a set of contiguous disk blocks
  - First-fit or best-fit
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Efficient for both sequential access and direct access
- Problems
  - External fragmentation, need for compaction
  - File cannot grow, need to know file size



directory

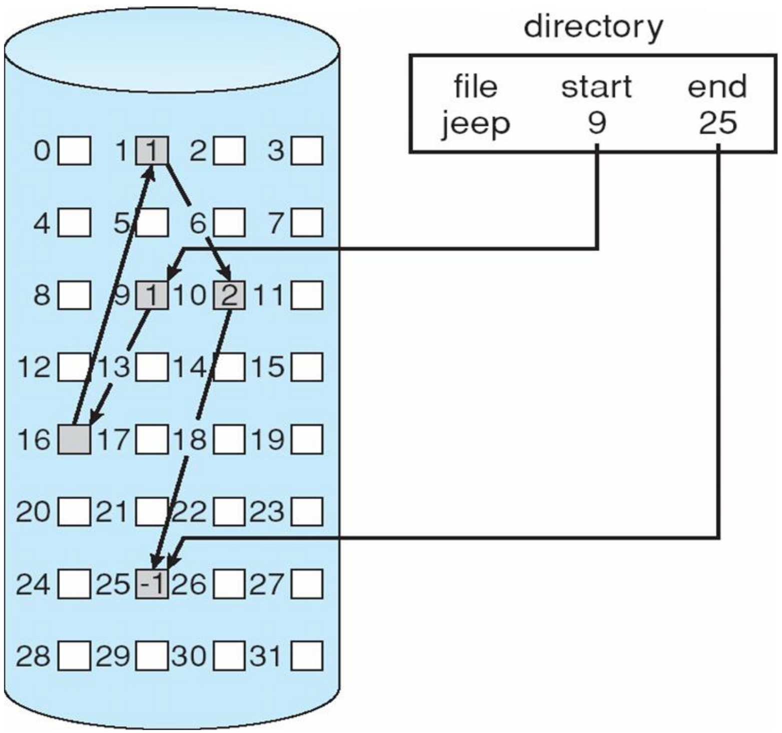
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Extent-Based Systems

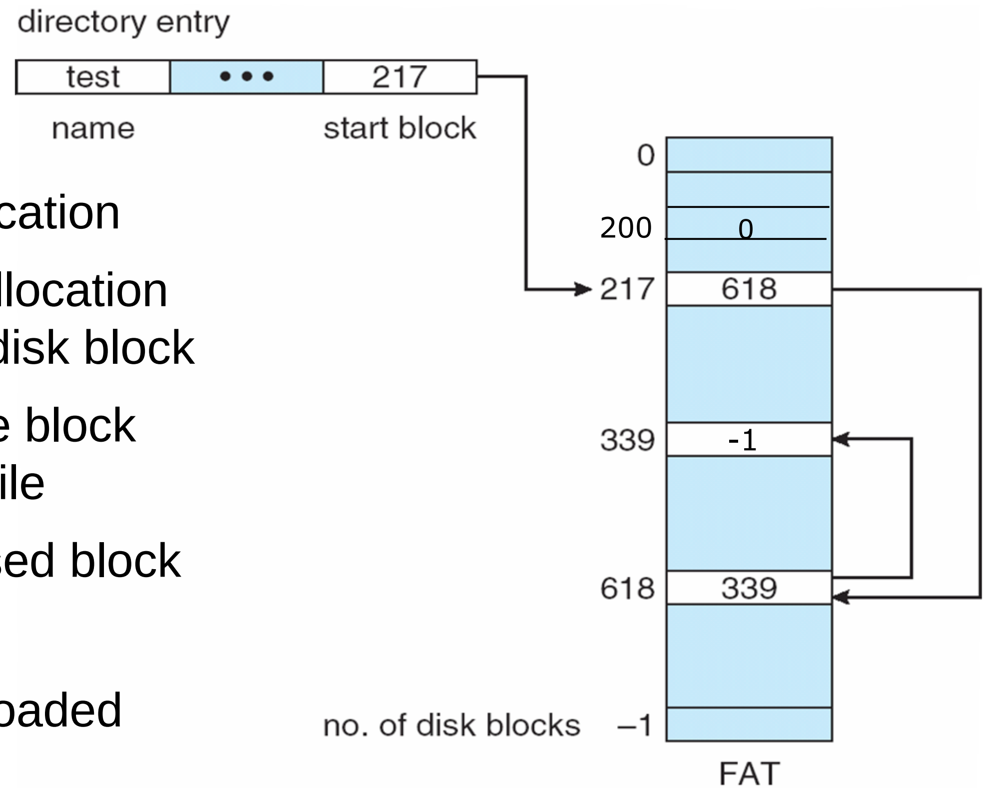
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- An **extent** is a contiguous chunk of space
- An extent is allocated to a file initially
  - If the amount is not large enough, another extent is added
- A file consists of one or more extents
  - For each extend, the location of the first block and the block count are recorded

Linked Allocation

- **Linked allocation** – each file is a linked list of blocks, blocks may be scattered anywhere on the disk
  - Each block contains pointer to next block
  - File ends at nil pointer
- No external fragmentation
- A file can continue to grow as long as free blocks are available
- Efficient for sequential access
- Inefficient for direct access – access to i^{th} block requires i disk reads
- Overhead - space required for pointers
- Reliability problem - what if a pointer is damaged?



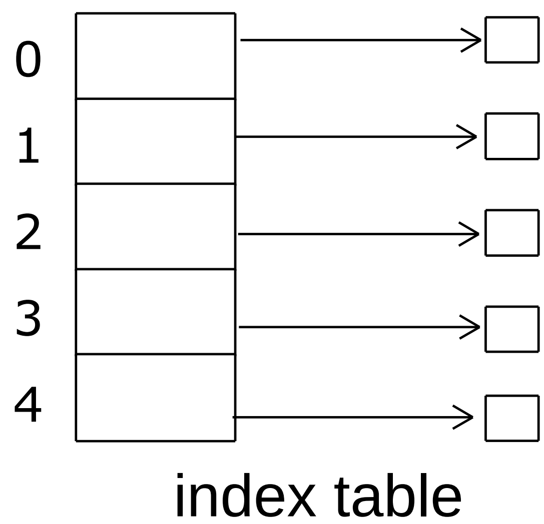
File-Allocation Table (FAT)



- **FAT** is an variation of linked allocation
- Beginning of volume has a file-allocation table having one entry for each disk block
  - Entry for block x contains the block number of the next block in file
  - Table value 0 indicates unused block
- New block allocation is simple
- Efficient direct access if FAT is loaded into the memory

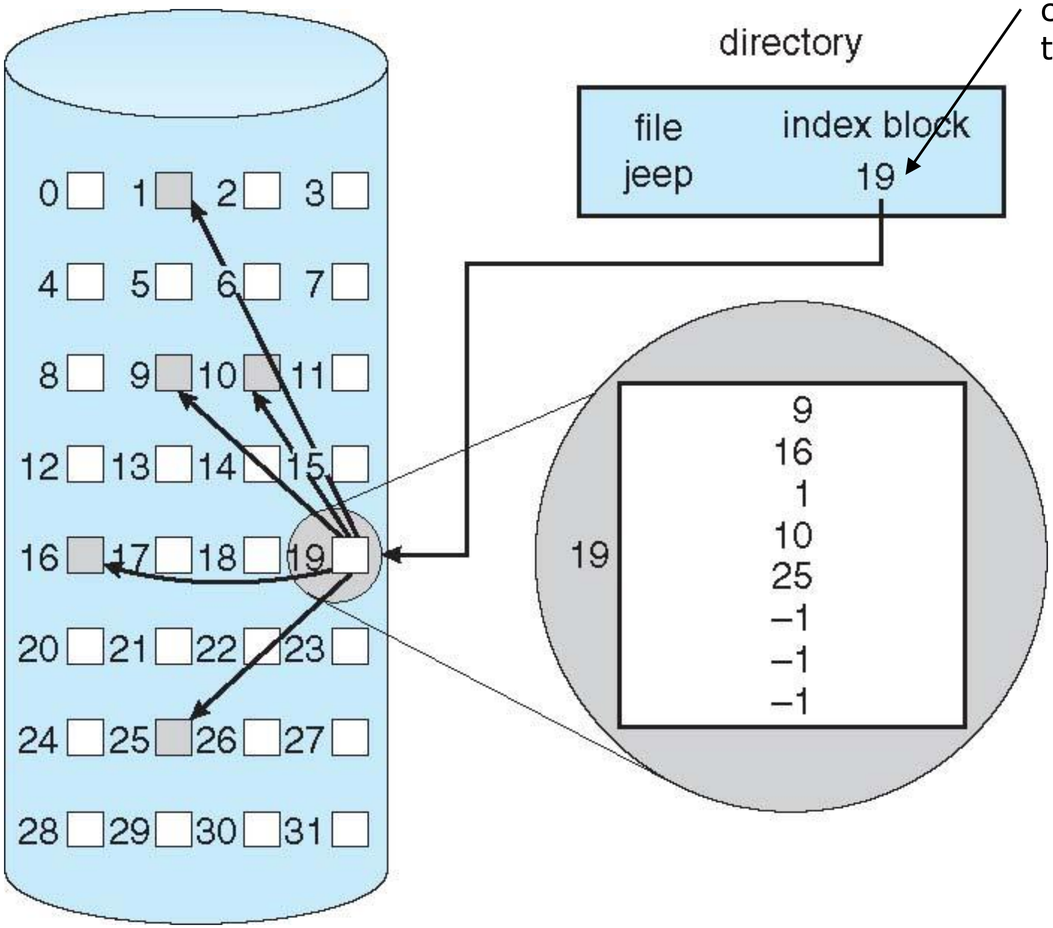
Indexed Allocation

- **Indexed allocation**
  - Each file has its own **index block** of pointers to its data blocks
  - To find the *i*th block of the file, we use the pointer in the *i*th index block entry



Example of Indexed Allocation

Directory entry contains the address of the index block



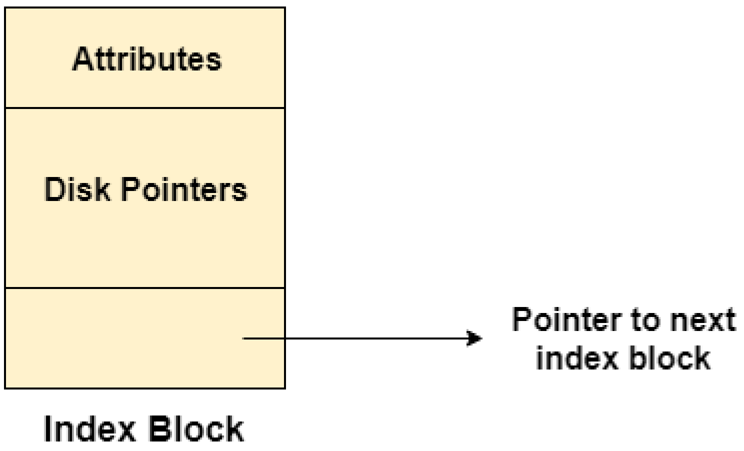
Indexed Allocation (Cont.)

- Efficient direct access if index block is in memory
- File size can change
- No external fragmentation, but have overhead of index block

- If block size is 1KB and pointer size is 4 bytes, how many data blocks can a file have?
  - 256 data blocks → max file size = 256KB
- What if the size of a file exceeds 256KB?
  - Solution 1: Linked index
  - Solution 2: Multilevel index
  - Solution 3: Combined scheme

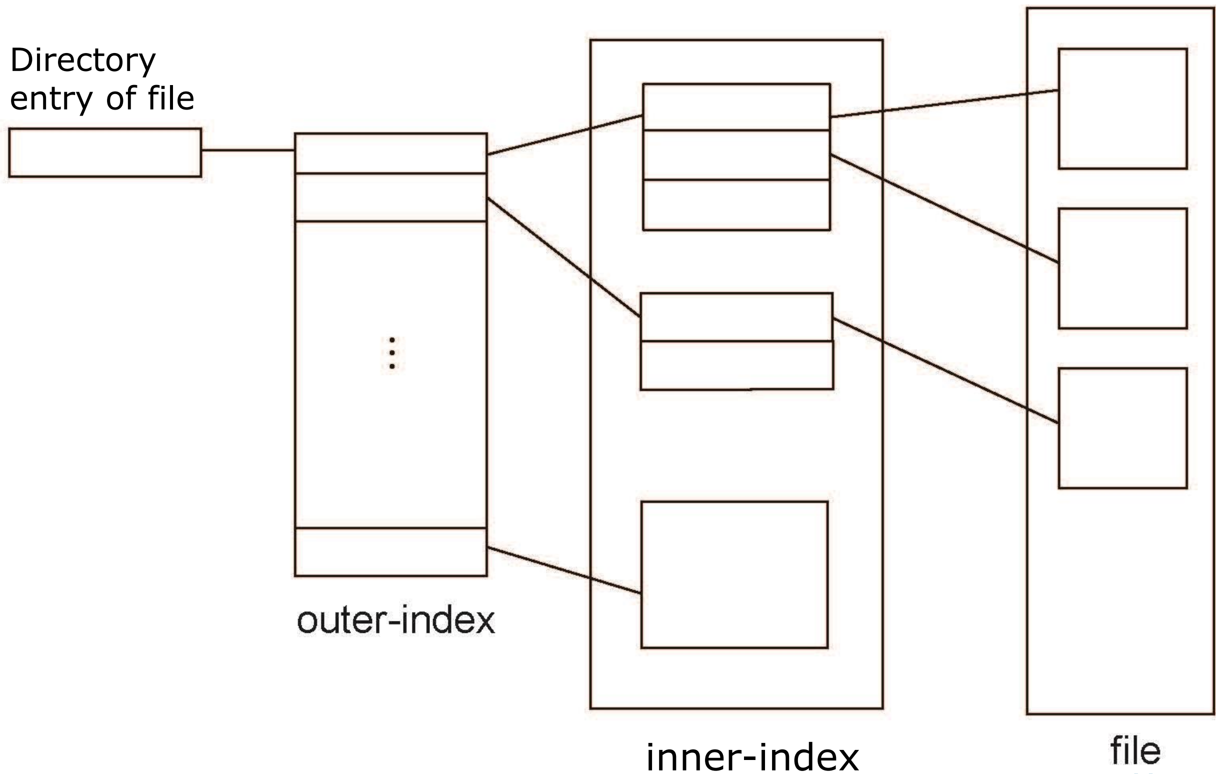
Linked Index Allocation

- To allow large files, we can link several index blocks together
- An index block contains
  - A small header giving name of the file
  - Set of N block addresses
  - Pointer to next index block



Multilevel Index Allocation

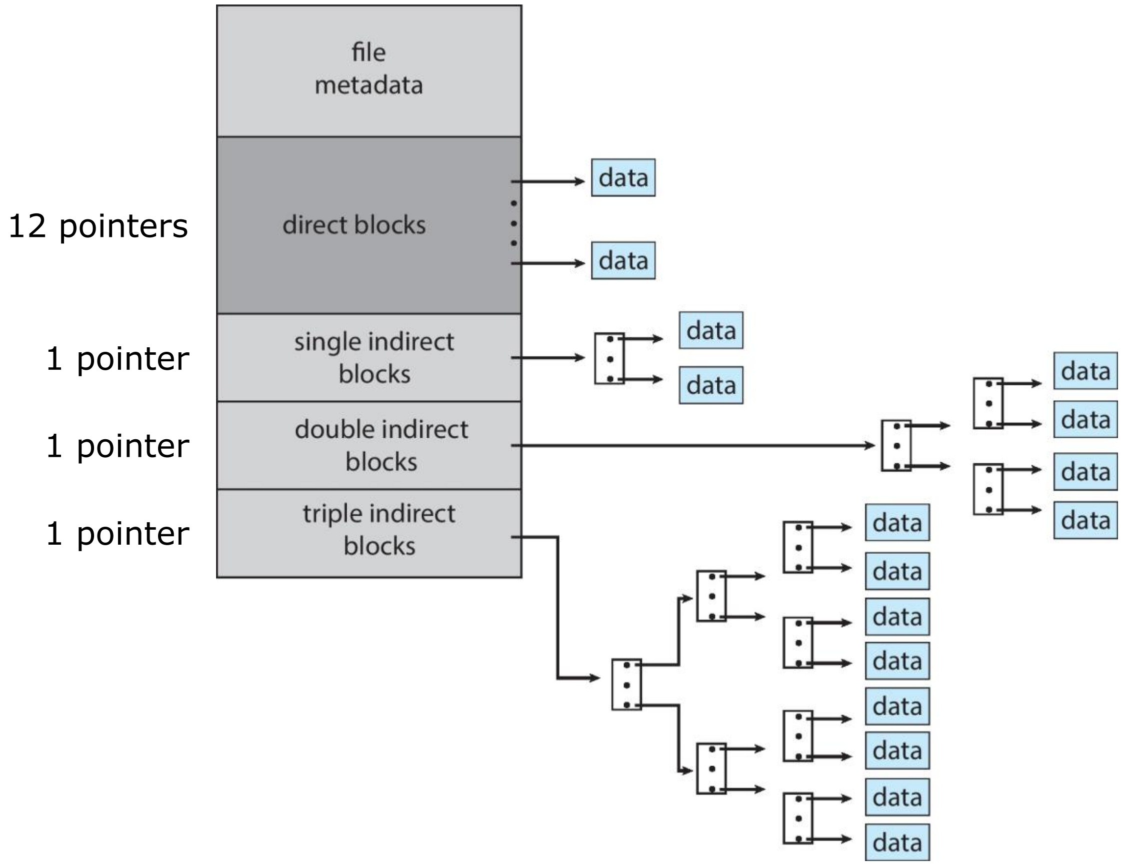
- Two-level index: A first-level index block points to a set of second-level index blocks, which in turn point to file blocks
- If block size is 4KB and pointers are 4 bytes, what is the maximum allowed file size?
  - 1,024 * 1024 data blocks and file size of up to 4GB



Combined Scheme: UNIX

4KB block size, 32-bit pointers

Number of blocks that can be allocated to a file = $12+1024+1024^2+1024^3$



The UNIX inode

Performance

- Best method depends on file access type
- Contiguous allocation great for sequential and direct accesses
  - Only one access is required to get a block
- Linked allocation good for sequential access, not direct access
  - Access to the *ith* block requires *i* block reads
- Some systems support direct-access files by using contiguous allocation and sequential-access files by using linked allocation
  - Declare access type at file creation
- Indexed allocation more complex
  - Single-level index: access requires index block read then data block read
  - Two-level index: access requires 2 index block reads then data block read
- Some systems combine contiguous allocation with indexed allocation
  - Using contiguous allocation for small files (up to 4 blocks) and switching to indexed allocation if the file grows large

Free-Space Management

- File system maintains **free-space list** to track available blocks
- **Bit vector** or **bitmap** (***n*** blocks)

0	1	2	3				n-1
0	0	1	1	1	0	…	1

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

The 1st non-0 word is scanned for the first 1 bit, which is the location of the first free block

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 (Cont.)

- Bitmap requires extra space
  - Example:

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

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

    number of blocks $\boldsymbol{n} = 2^{40}/2^{12} = 2^{28} \rightarrow$ need a bit map

    of 2^{28} bits (or 32MB) to track the free blocks

    if clusters of 4 blocks $\rightarrow$ 8MB of memory
- Can get n consecutive free blocks easily

0	1	2	3				***n***-1
0	0	1	1	1	0	…	1

Linked Free-Space List on Disk

- Link together all the free blocks
  - Allocating a free block to a file is easy
  - Cannot get contiguous space easily