EECS 211:

Advanced System Software

Lecture: File System Implementation

Prof. Mohammad Al Faruque

The Henry Samueli School of Engineering
Electrical Engineering & Computer Science
University of California Irvine (UCI)

File System Implementation

- **File-System Structure**
- **File-System Implementation**
- **Directory Implementation**
- **Allocation Methods**

File-System Structure

- **File structure**
  - **Logical storage unit**
  - **Collection of related information**
- **File system resides on secondary storage (disks)**
  - **Provided user interface to storage, mapping logical to physical**
  - **Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily**
- **Disk provides in-place rewrite and random access**
  - **I/O transfers performed in blocks of sectors (usually 512 bytes)**
- **File control block – storage structure consisting of information about a file**
- **Device driver controls the physical device**

File system organized into layers

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical

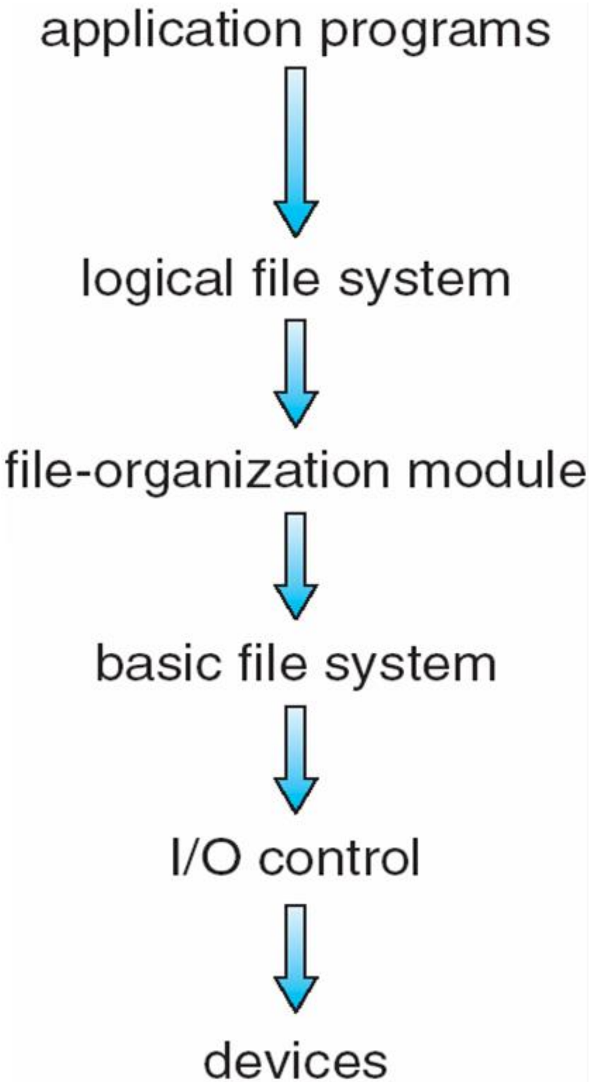
Challenges of a file system

- How the file should look to the user? →previous lecture
- Creating algorithms and data structures to map the logical file system onto the physical secondary-storage device?

- Device driver controls the physical device

File system organized into layers

Layered File System



File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like "retrieve block 123" translates to device driver
  - Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation

File System Layers (Cont.)

- **Logical file system manages metadata information**
  - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in Unix)
  - Directory management
  - Protection
- **Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance**
  - Logical layers can be implemented by any coding method according to OS designer
- **Many file systems, sometimes many within an operating system**
  - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 40 types, with **extended file system** ext2 and ext3 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management

File-System Implementation

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures

On-disk structures

- Boot control block contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- Volume control block (superblock, master file table) contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table
- Per-file File Control Block (FCB) contains many details about the file
  - Inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

A Typical File Control Block

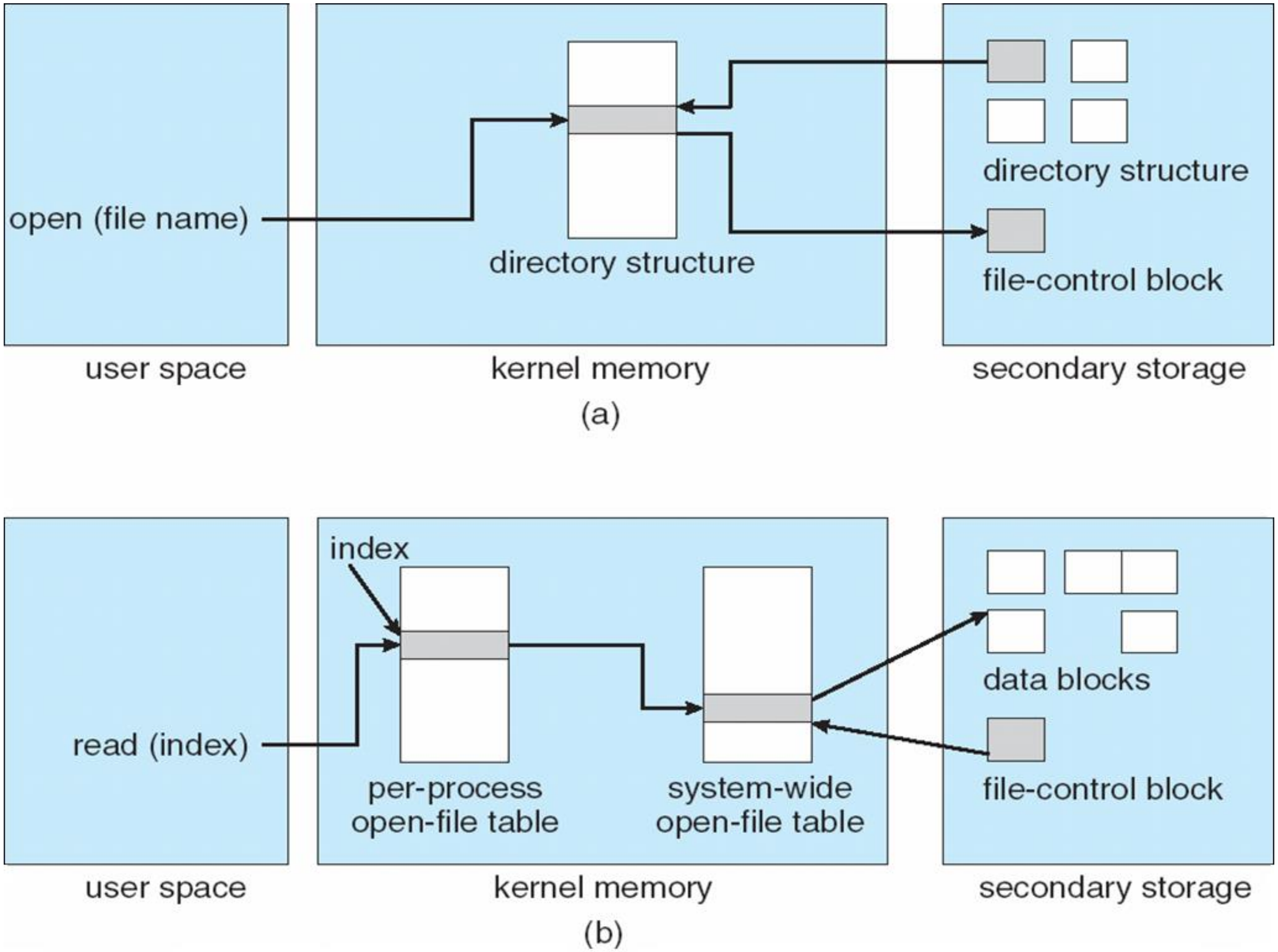
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

In-memory structures

- **Mount table** storing file system mounts, mount points, file system types
- An **in-memory directory-structure** cache holds the directory information of recently accessed directories.
- **System-wide open file table** contains a copy of the FCB of each open file, also other information
- **Per-process Open-file Table** pointer to the appropriate entry in the system-wide open file table, other information
- **Buffers** hold file-system blocks when are being read from disk or write to disk.

In-Memory File System Structures



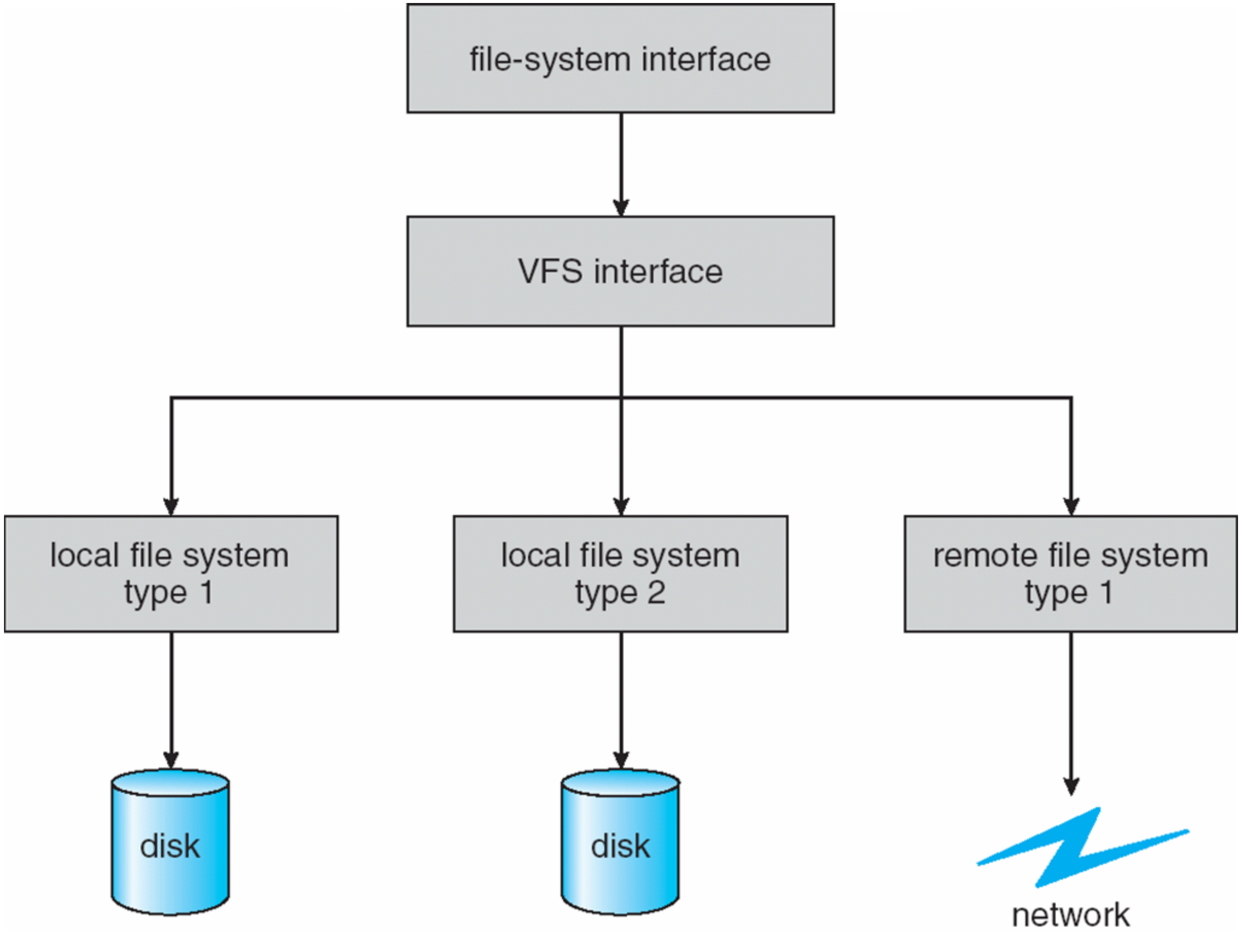
Partitions and Mounting

- **Partition can be a volume:**
  - **Containing a file system ("cooked") or**
  - **Raw – just a sequence of blocks with no file system**
- **Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system**
  - **Or a boot management program for multi-os booting**
- **Root partition contains the OS, other partitions can hold other Oses, other file systems, or be raw**
  - **Mounted at boot time**
  - **Other partitions can mount automatically or manually**
- **At mount time, file system consistency checked**
  - **Is all metadata correct?**
    - If not, fix it, try again
    - If yes, add to mount table, allow access

Virtual File Systems

- Observations:
  - How can we integrate multiple types of file systems into a directory structure?
  - How can users seamlessly move between file-system types as they navigate the file-system space?
- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Major task 1: Separates file-system generic operations from implementation details
  - Major task 2: Implementation can be one of many file systems types, or network file system
    - Implements **vnodes** which hold **inodes** or network file details
  - Then dispatches operation to appropriate file system implementation routines
- The API is to the VFS interface, rather than any specific type of file system

Schematic View of Virtual File System



Virtual File System Implementation

- For example, Linux has four object types:
  - Inode object → an individual file
  - File object → an open file
  - Superblock object → an entire file system
  - Dentry object → an individual directory entry

- VFS defines set of operations on the objects that must be implemented
  - Every object has a pointer to a function table
    - Function table has addresses of routines to implement that function on that object

File System Implementation

- **File-System Structure**
- **File-System Implementation**
- **Directory Implementation**
- **Allocation Methods**
- **Free-Space Management**

Directory Implementation

- **Linear list of file names with pointer to the data blocks**
  - **Simple to program**
  - **Time-consuming to execute**
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree

- **Hash Table – linear list with hash data structure**
  - **Decreases directory search time**
  - **Collisions – situations where two file names hash to the same location**
  - **Only good if entries are fixed size, or use chained-overflow method**

File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management

Allocation Methods

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

- **Challenge:** The major challenge is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly?

- Three methods exist:
  1. Contiguous
  2. Linked
  3. Indexed

Allocation Methods - Contiguous

- **Contiguous allocation – each file occupies 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 file size, external fragmentation, need for compaction off-line (downtime) or on-line**
  - **Second problem → how much space is needed for a file?**
  - **Similar to problems seen in chapter 8 memory allocation → first fit, best fit, worst fit**

Contiguous Allocation

- **Mapping from logical to physical**

Q

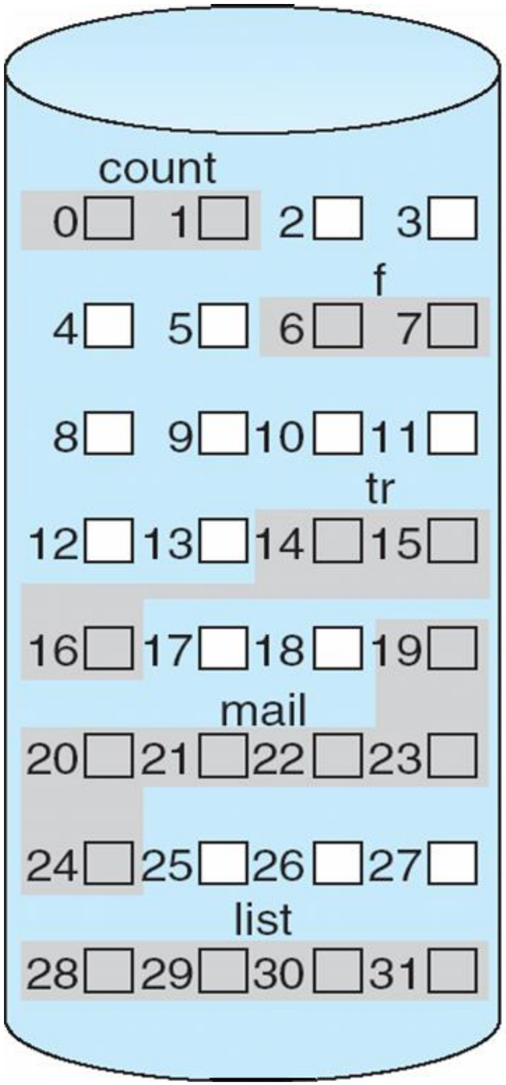
Logical Address LA/512

R

Block to be accessed = Q + starting address

Displacement into block = R

Contiguous Allocation of Disk Space



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

- Extent-based file systems allocate disk blocks in extents

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

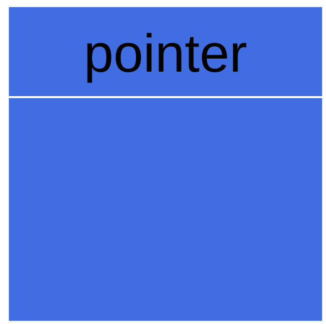
Allocation Methods - Linked

- **Linked allocation – each file a linked list of blocks**
  - **File ends at nil pointer**
  - **No external fragmentation**
  - **Each block contains pointer to next block**
  - **No compaction, external fragmentation**
  - **Free space management system called when new block needed**
  - **Improve efficiency by clustering blocks into groups but increases internal fragmentation**
  - **Reliability can be a problem**
  - **Locating a block can take many I/Os and disk seeks**

Linked Allocation

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

block =



Linked Allocation

- Mapping

Q

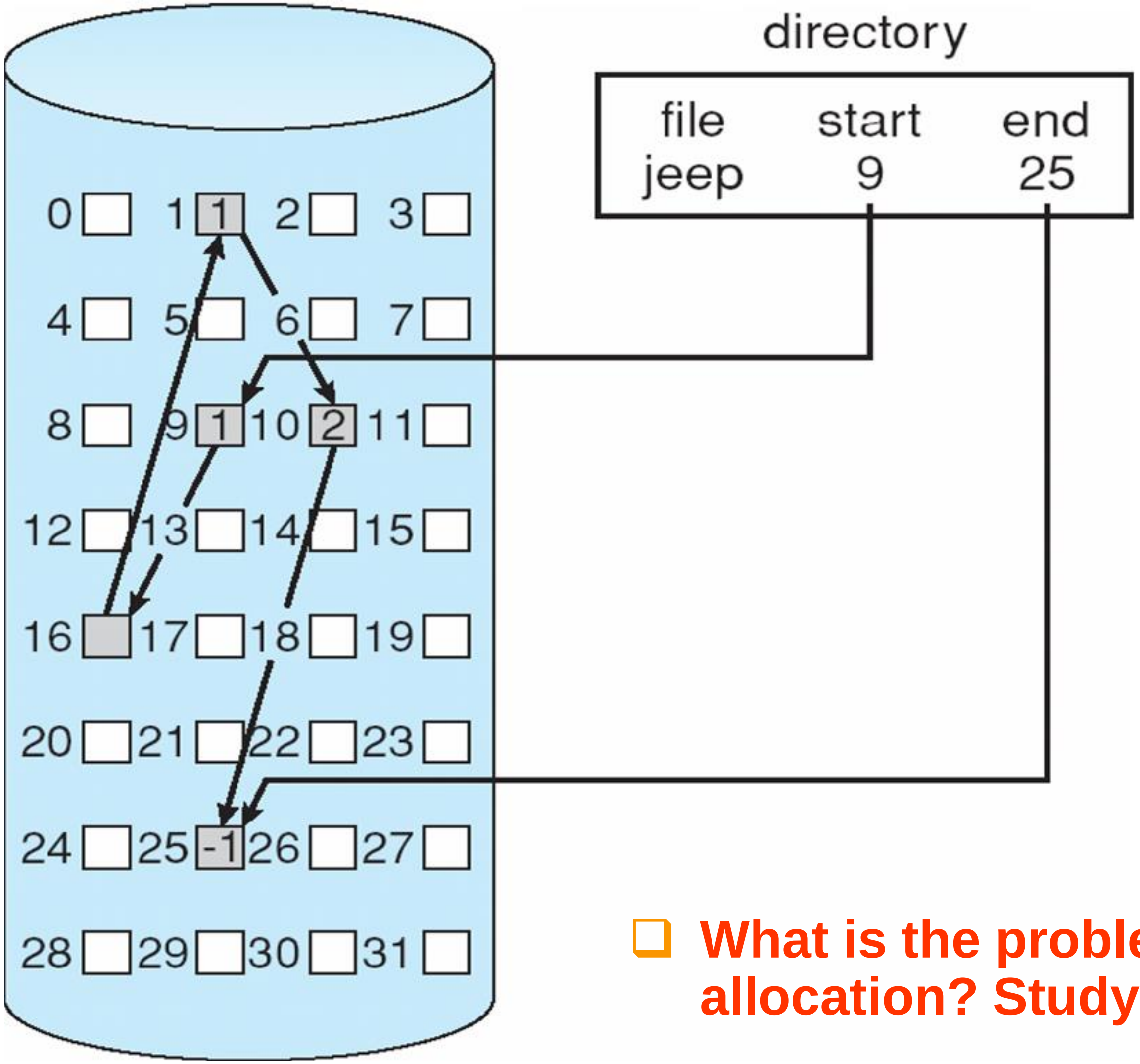
LA/511

R

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

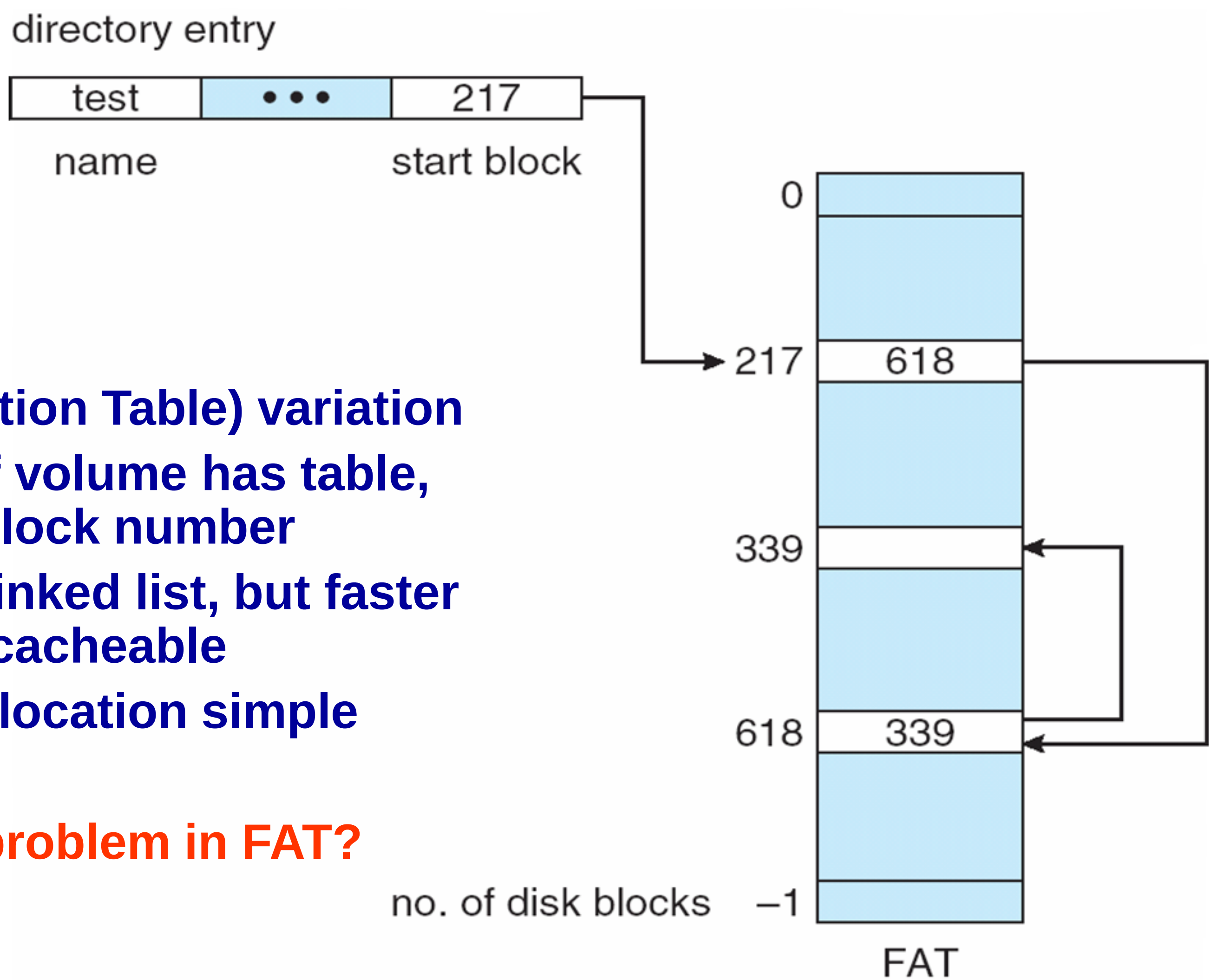
Displacement into block = R + 1

Linked Allocation



- **What is the problem in linked allocation? Study!**

File-Allocation Table

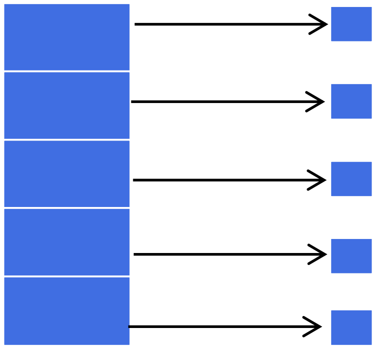


- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple
  - What is the problem in FAT? Study!

Allocation Methods - Indexed

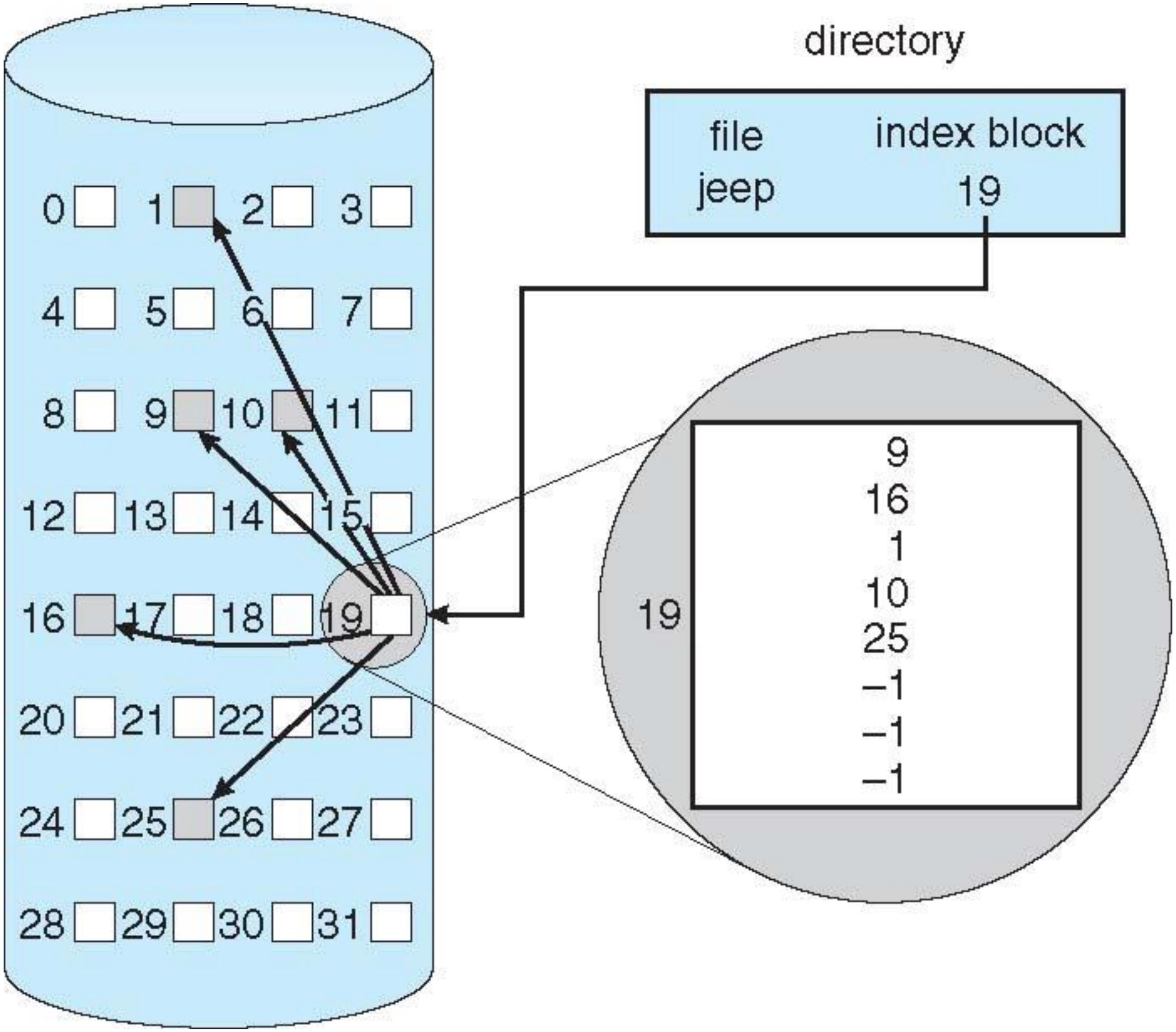
- **Indexed allocation**
  - **Each file has its own index block(s) of pointers to its data blocks**

- **Logical view**



index table

Example of Indexed Allocation



Indexed Allocation (Cont.)

- **Need index table**

Q

LA/512

R

Q = displacement into index table
R = displacement into block

- **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**

Indexed Allocation – Mapping (Cont.)

- **Mapping from logical to physical in a file of unbounded length (block size of 512 words)**
- **Linked scheme – Link blocks of index table (no limit on size)**

$LA\ /\ (512 \times 511)$ → Q_1, R_1

Q_1 = block of index table
R_1 is used as follows:

$R_1\ /\ 512$ → Q_2, R_2

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

Indexed Allocation – Mapping (Cont.)

Q1
LA / (512 x 512)
R1

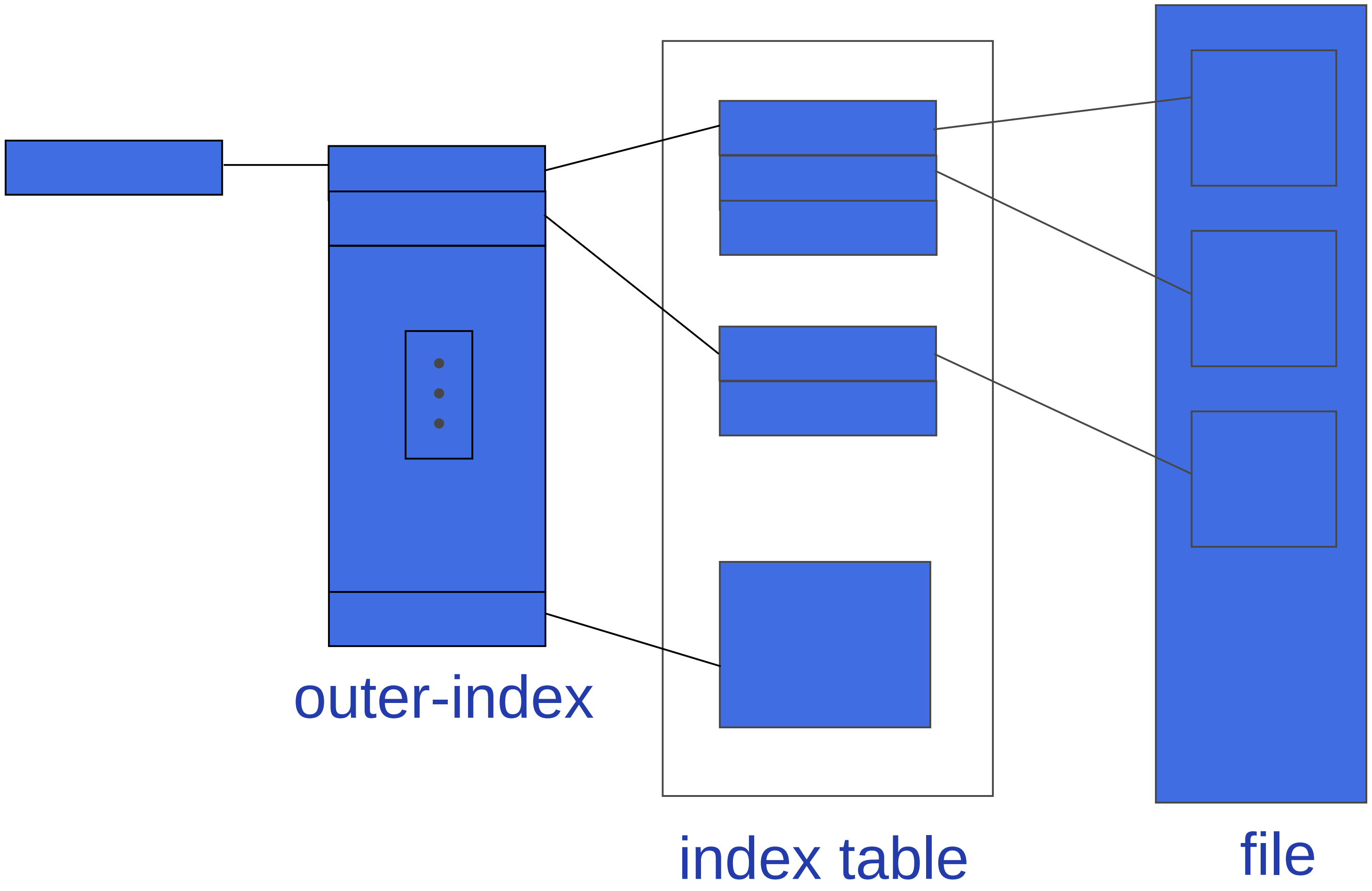
- **Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)**

Q1 = displacement into outer-index
R_1 is used as follows:

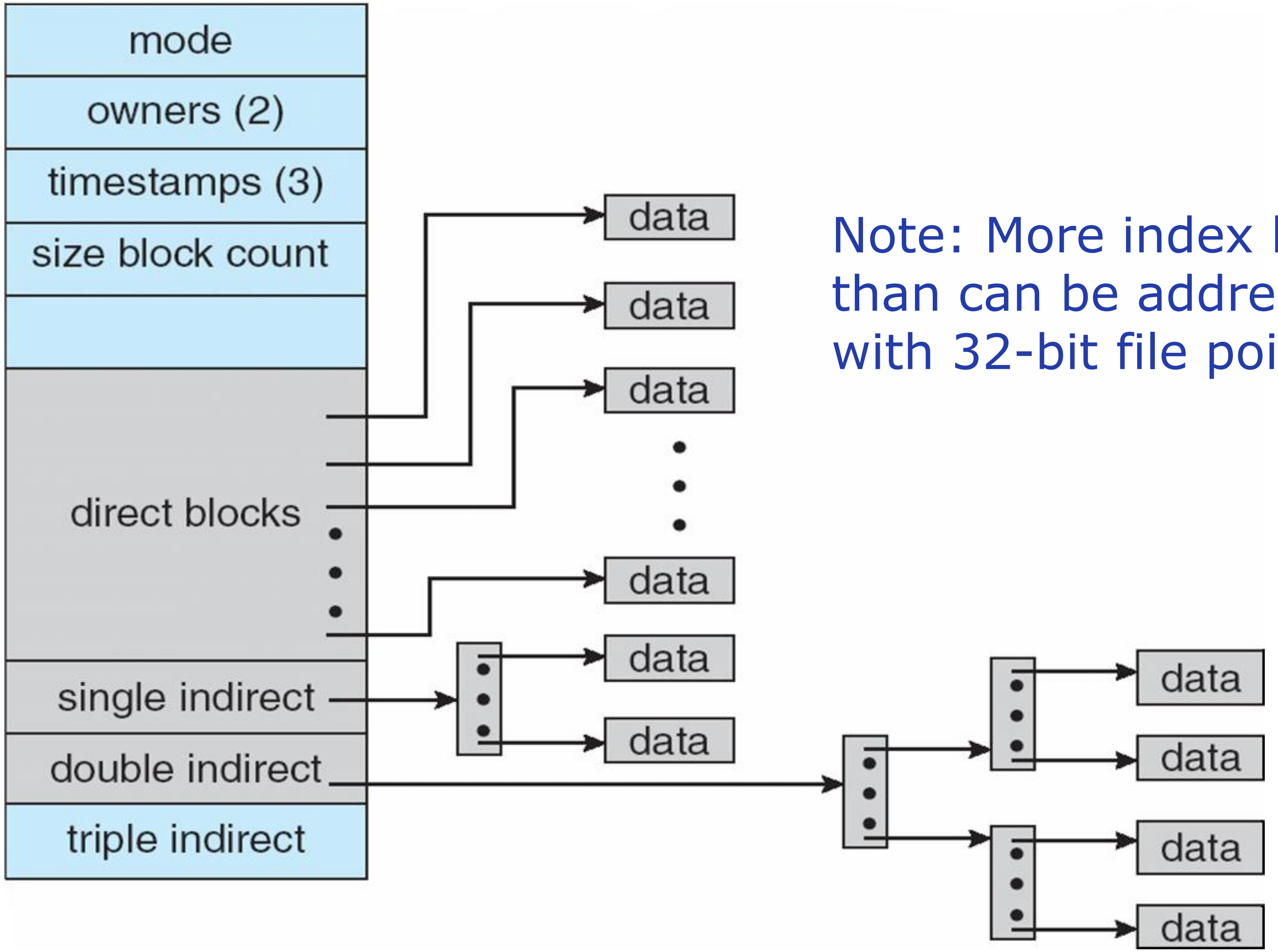
Q2
R1 / 512
R2

Q2 = displacement into block of index table
R2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX UFS (4K bytes per block, 32-bit addresses)



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

Performance

- Needs to consider two KPIs
  1. Storage efficiency
  2. Data-block access time
- 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

Performance (Cont.)

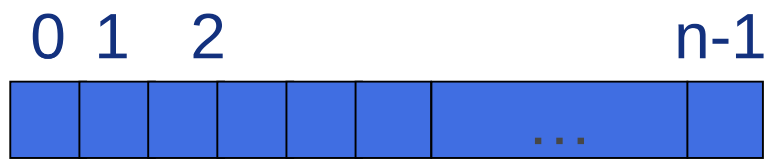
- **Adding instructions to the execution path to save one disk I/O is reasonable**
  - **Intel Core i7 Extreme Edition 990x (2011) at 3.46Ghz = 159,000 MIPS**
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - **Typical disk drive at 250 I/Os per second**
    - **159,000 MIPS / 250 = 630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

File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management

Free-Space Management

- **File system maintains free-space list to track available blocks/clusters**
  - **(Using term "block" for simplicity)**
- **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

(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.)

- **Bit map requires extra space**
  - **Example:**

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

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

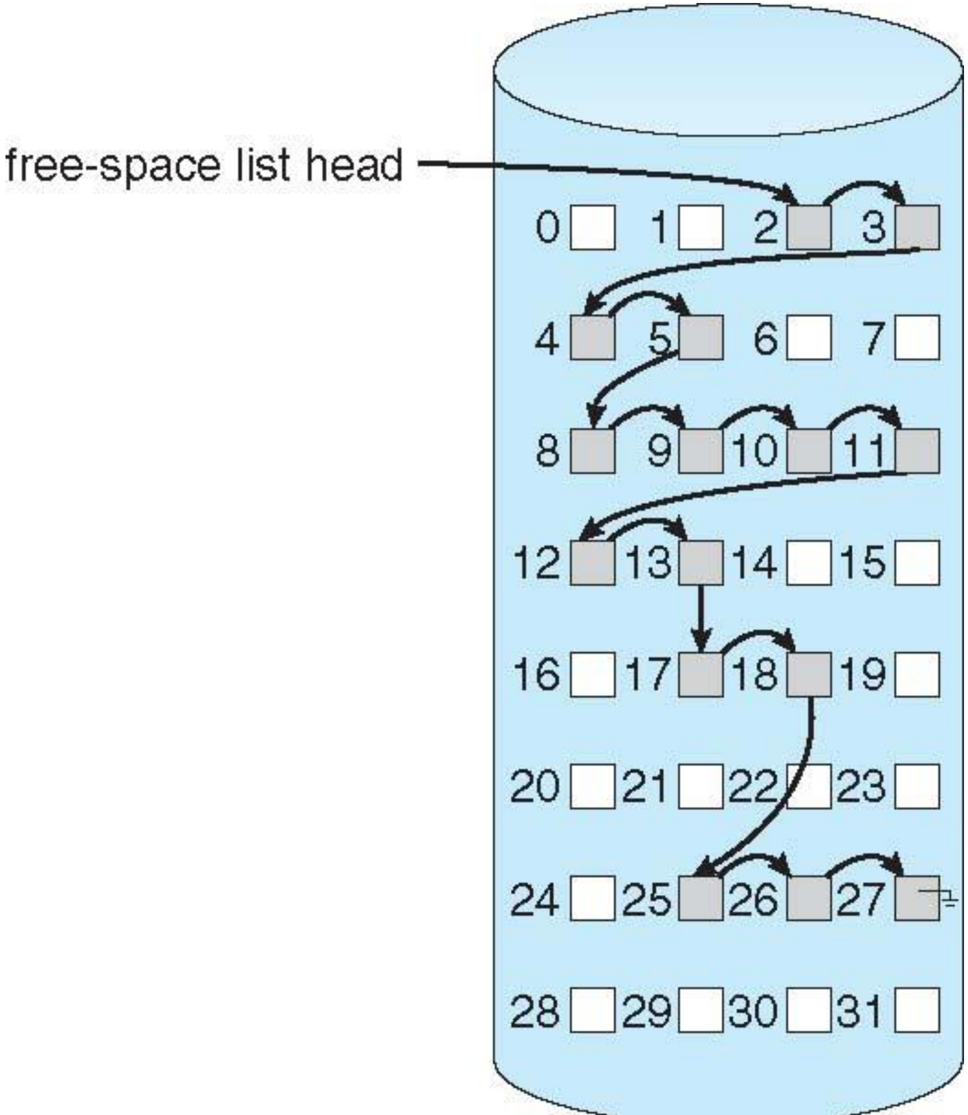
    $n = 2^{40}/2^{12} = 2^{28}$ bits (or 256 MB)

    if clusters of 4 blocks -> 64MB of memory

- **Easy to get contiguous files**

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

Linked Free Space List on Disk



Free-Space Management (Cont.)

- **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

References

Part of the contents of this lecture has been adapted from the book Abraham Silberschatz, Peter B. Galvin, Greg Gagne: "Operating System Concept ", Publisher : Wiley; 9 edition (December 17, 2012), ISBN-13: 978-1118063330

Slides also contain lecture materials from John Kubiatowicz (Berkeley), John Ousterhout (Stanford), Nalini (UCI), Rainer (UCI), and others

Some slides adapted from http://www-inst.eecs.berkeley.edu/~cs162/ Copyright © 2010 UCB

Thank you for your attention