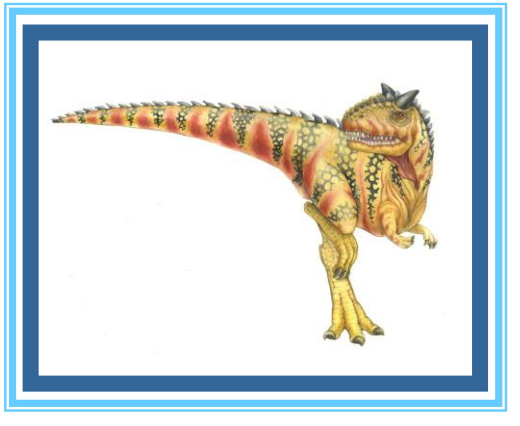
# Chapter 12: File System Implementation



# Chapter 12: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- NFS
- Example: WAFL File System

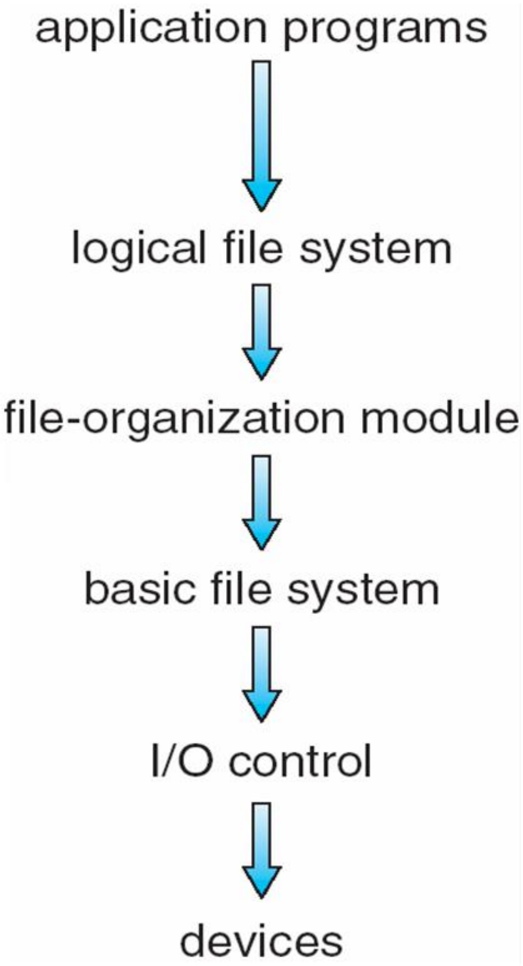
# Objectives

- 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

# 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

# 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 performanceTranslates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Logical layers can be implemented by any coding method according to OS designer

# File System Layers (Cont.)

- 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

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

# File-System Implementation (Cont.)

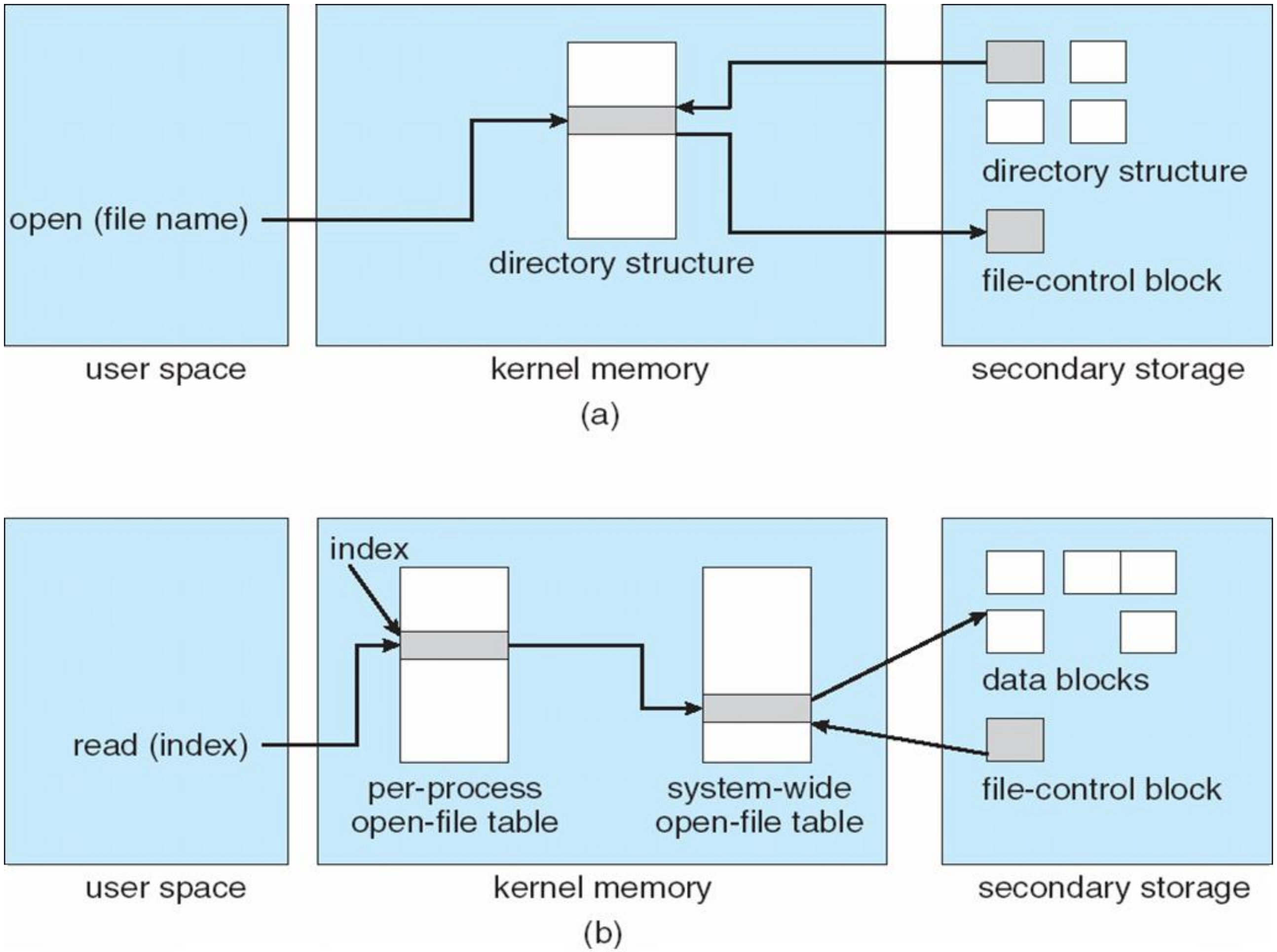
- Per-file **File Control Block** (**FCB**) contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table  using relational DB structures

| |
|---|
| 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

- Mount table storing file system mounts, mount points, file system types
- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

# 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

# Allocation Methods - Contiguous

- 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
  - 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**

# Contiguous Allocation

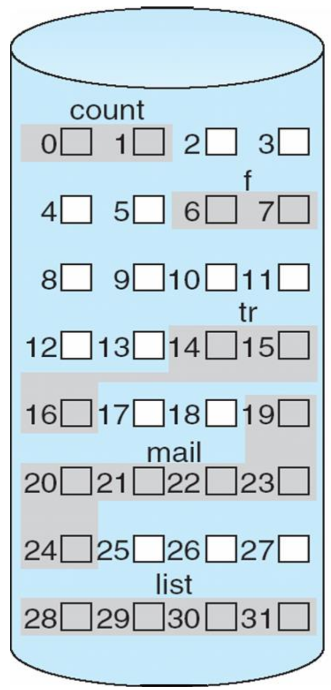
- Mapping from logical to physical

LA/512

Q

R

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




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
  - Reliability issues
  - Locating a block?? many I/Os and disk seeks

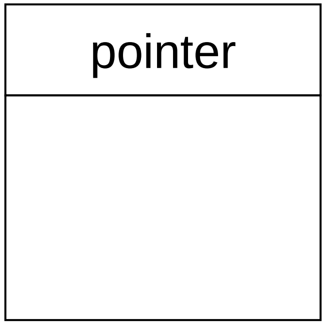
# Allocation Methods – Linked (Cont.)

- 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

# Linked Allocation

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

block =



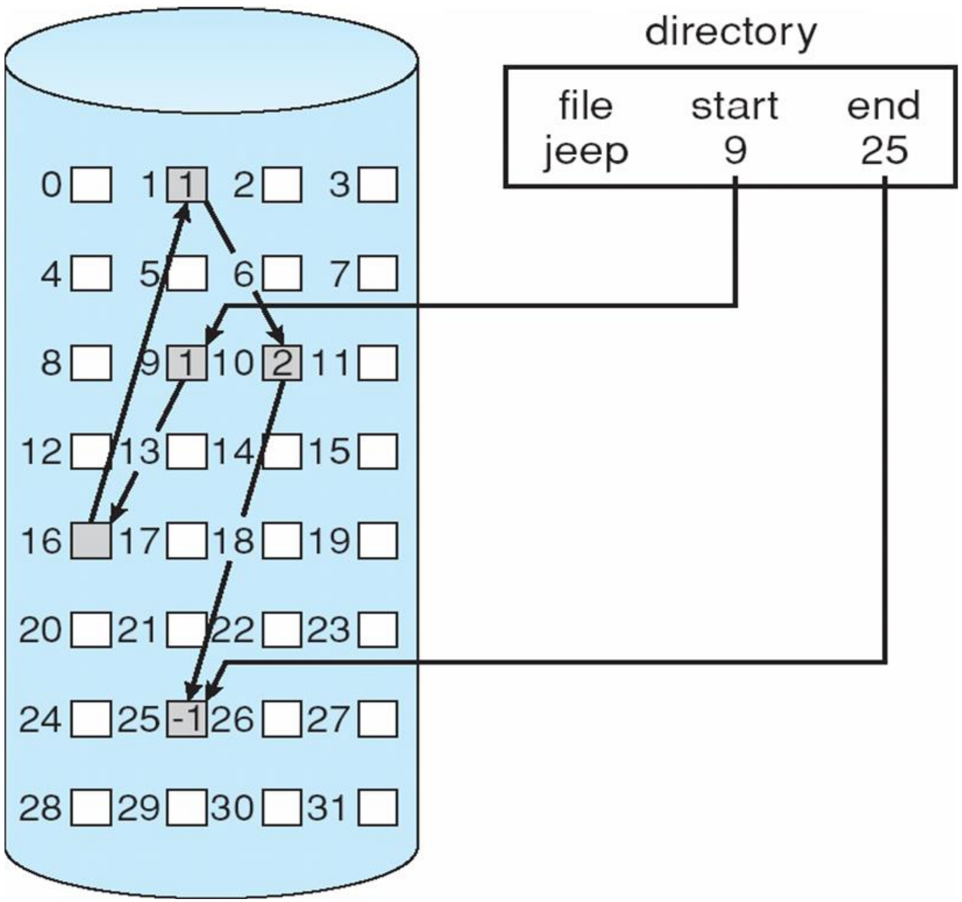

- Mapping

$$\frac{LA}{511} \rightarrow Q, R$$

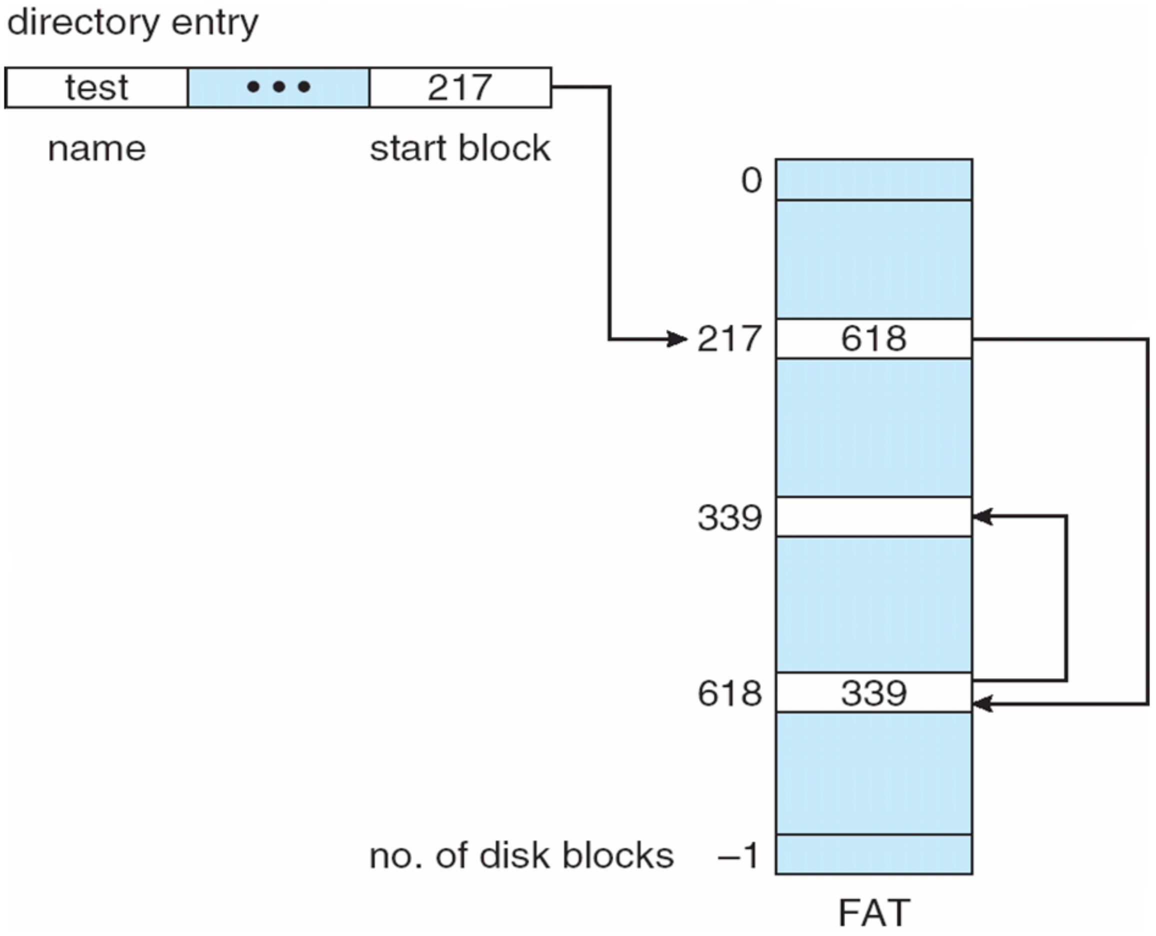
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

# Linked Allocation



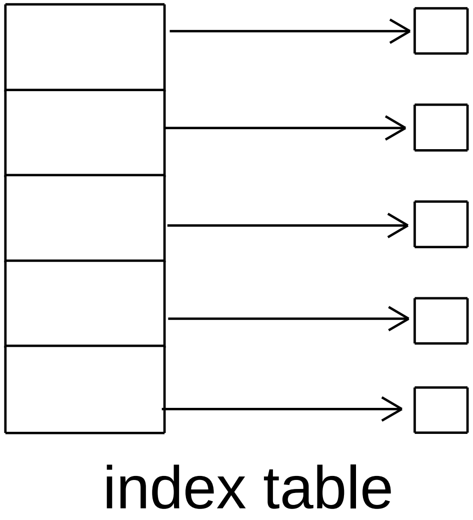
# File-Allocation Table



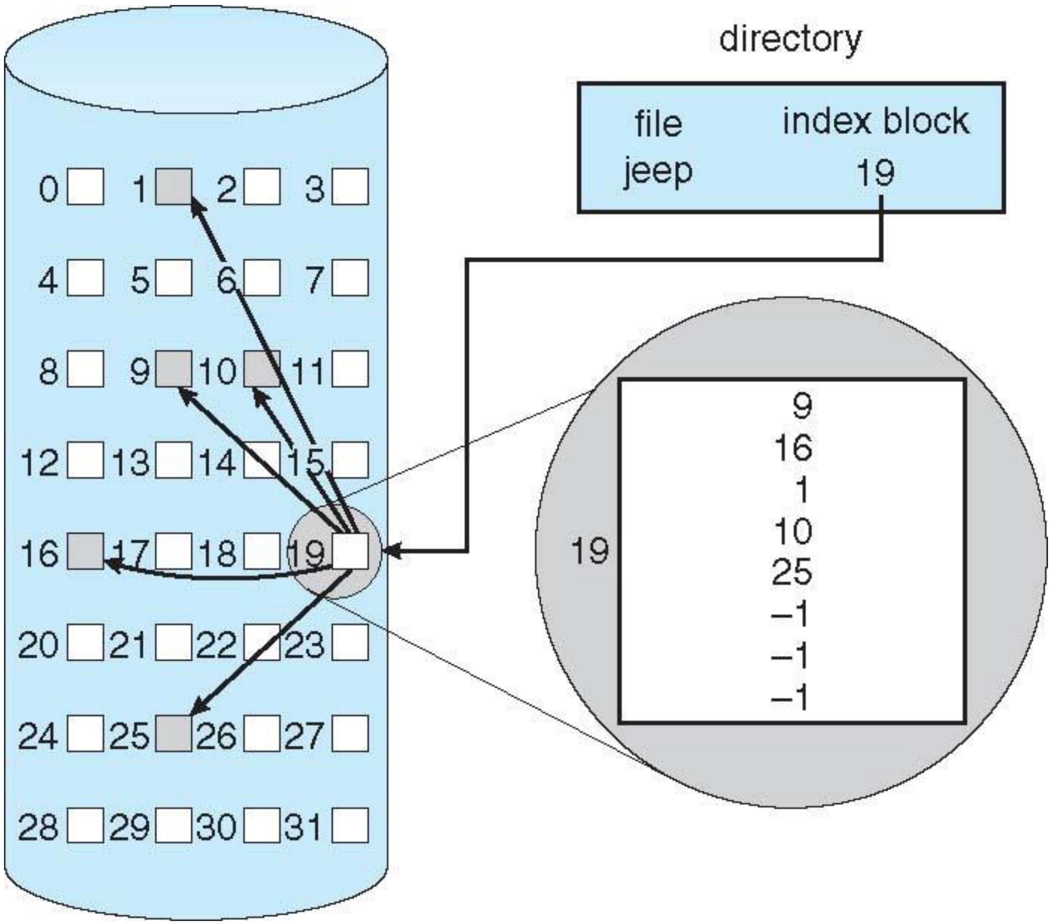
# Allocation Methods - Indexed

- **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, 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

# 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)

$$LA / (512 \times 511) \begin{cases} Q_1 \\ R_1 \end{cases}$$

$Q_1$ = block of index table
$R_1$ 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 – Mapping (Cont.)

- 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)

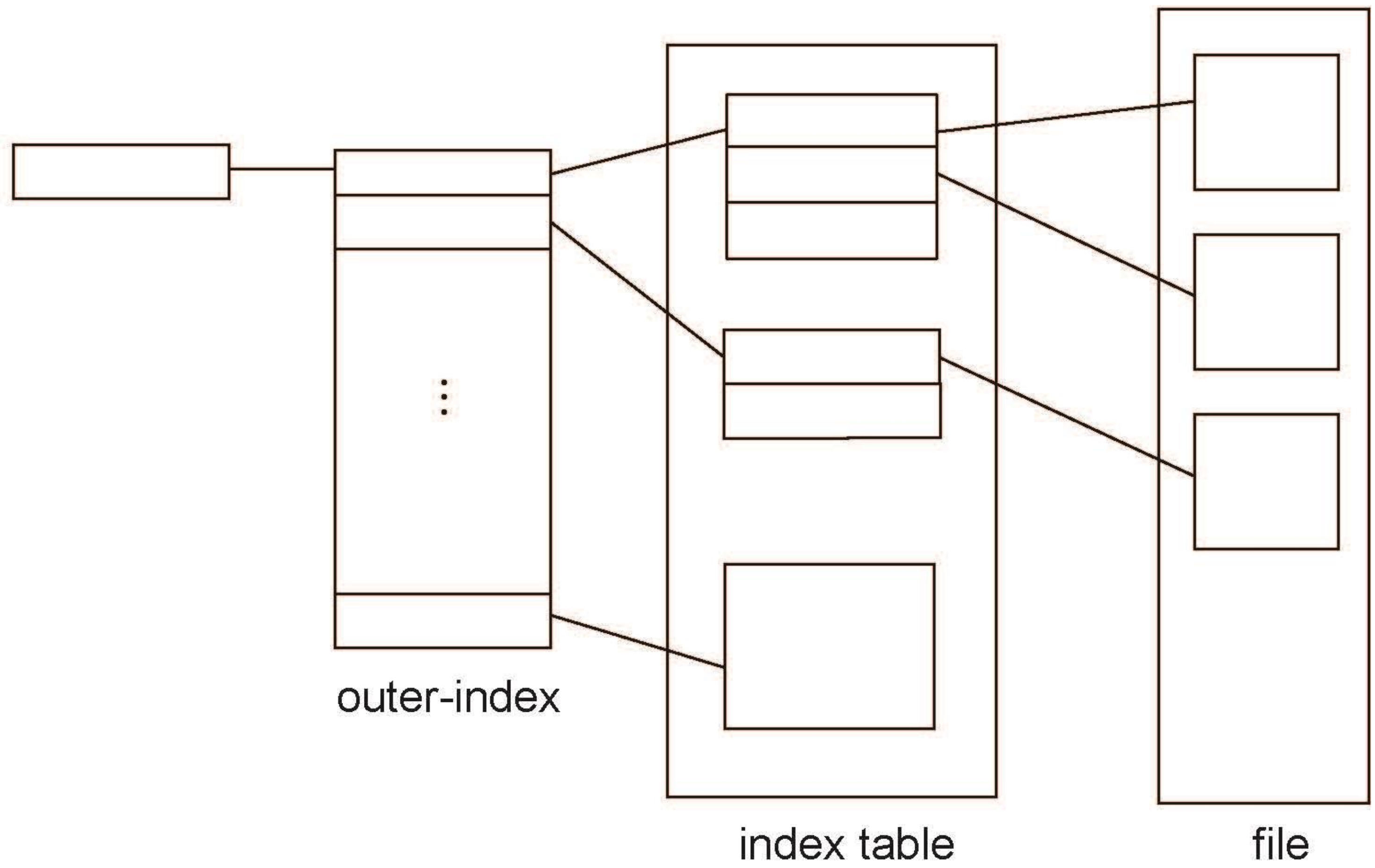
$$\text{LA} / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

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

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

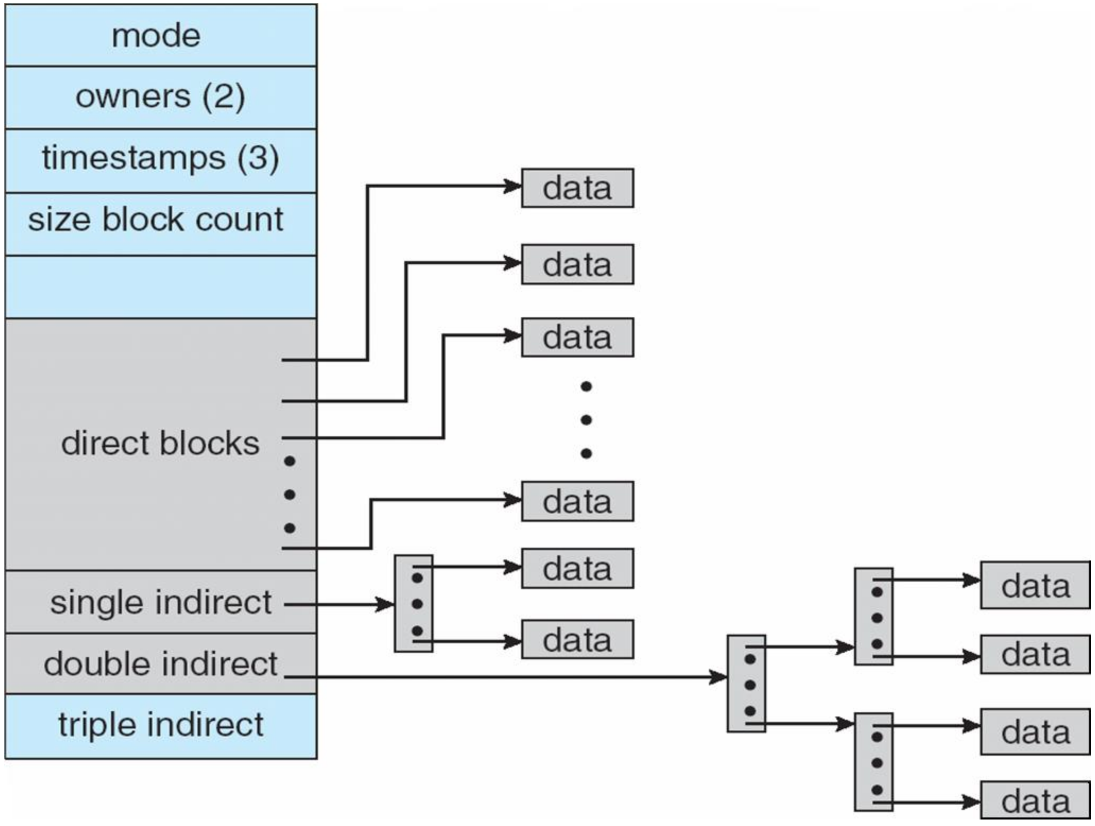
$Q_2$ = displacement into block of index table
$R_2$ displacement into block of file:

# Indexed Allocation – Mapping (Cont.)



# Combined Scheme: UNIX UFS

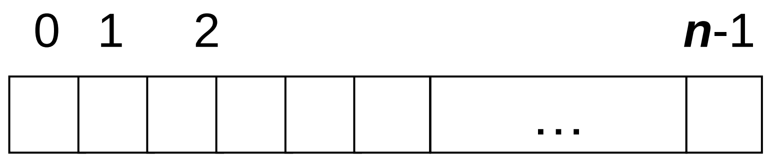
4K bytes per block, 32-bit addresses




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

# 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)




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

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 (Cont.)

- Bit map requires extra space
  - Example:

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

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

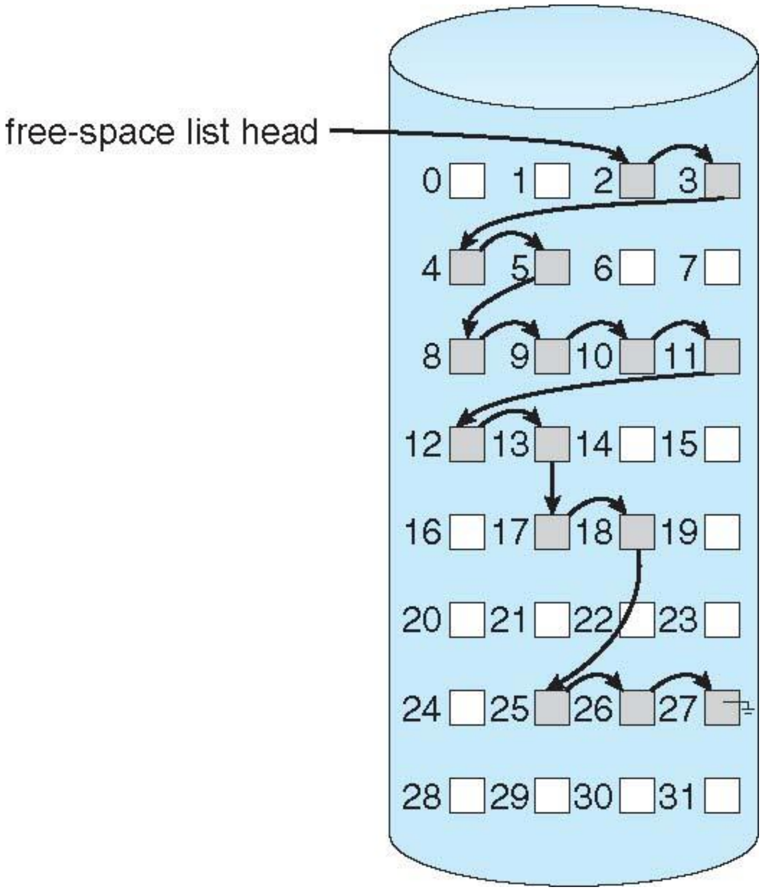
    $\boldsymbol{n} = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files

# Linked Free Space List on Disk

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



# 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

# End of Chapter 12