File Systems

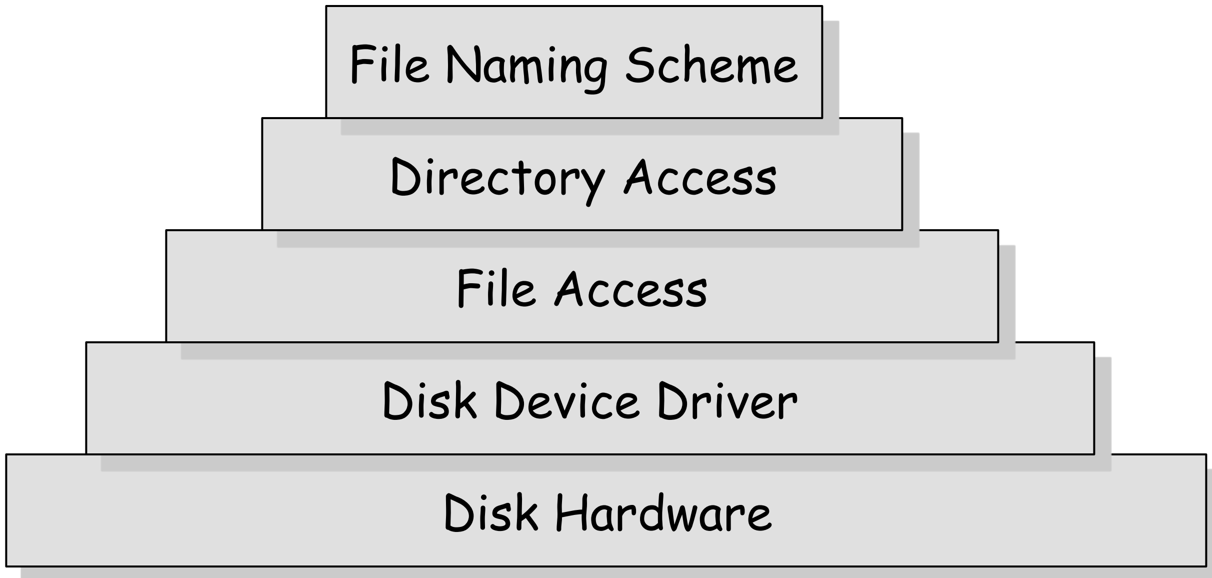
File System

- File system vs. Disk
  - File system builds an abstraction of storage
    - File → Track/sector
- To a user process
  - A file system provides a coherent view of a group of files
    - A file looks like a contiguous block of bytes (Unix)
  - A file system provides protection

File System

- Main Purposes
  - Manages persistent (nonvolatile) storage
  - Allows user to manipulate named objects (files)
  - Provides access to stored information

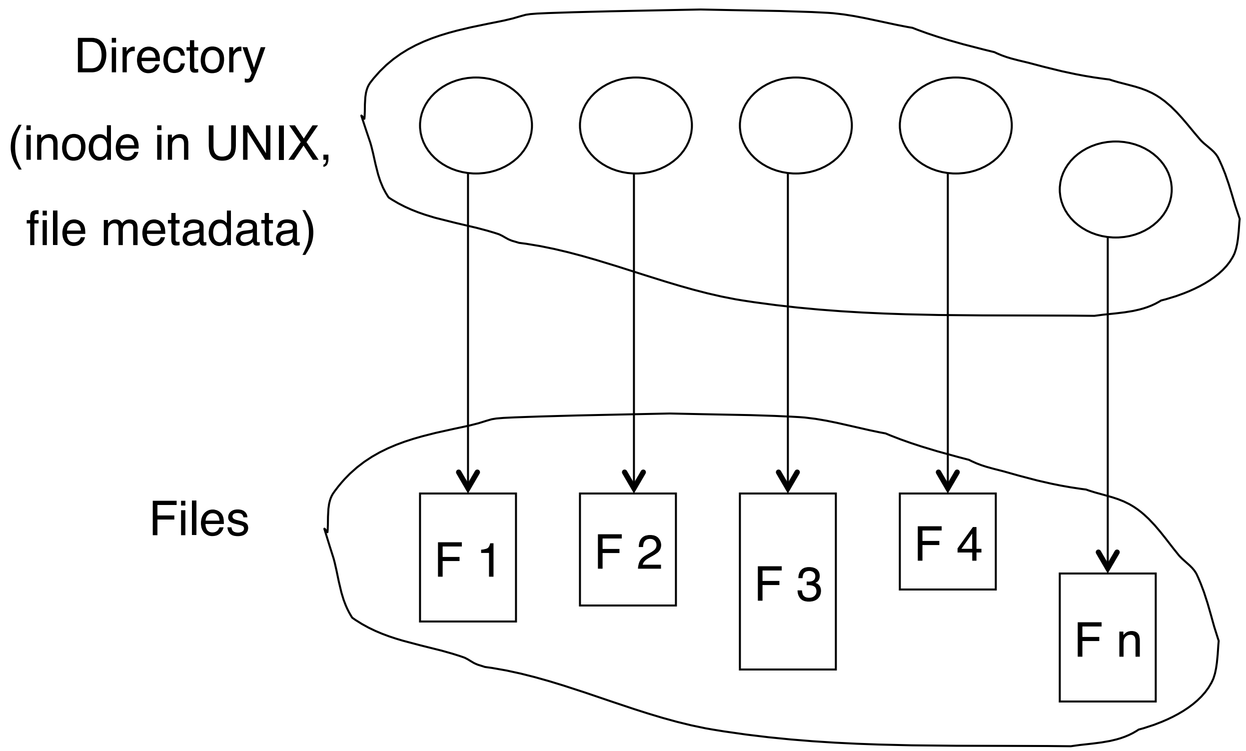
Conceptual Layers of File System



- Layer provides functionality
- Implementation often integrated

Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

Functionality of Each Layer

- Naming Layer
  - Deals with name syntax
  - May understand file location
- Directory Layer
  - Maps name to file data blocks
- File Layer
  - Implements operations on files
    - E.g., create, open, read, write, seek …

Two Fundamental Philosophies

- Typed files
  - System understands file content / format / structure
  - Operations specific to type

- Untyped Files
  - System does not understand contents, format, or structure -- ‘‘sequence of bytes’’ approach
  - Small set of operations apply to all files

Any Example?

Secondary Storage Management

- Space must be allocated to files
  - Static Allocation
    - Allocate file before using, w/ fixed file size
    - Easy to implement, but difficult to use
    - May require compaction
  - Dynamic Allocation
    - Files grow as needed
- Must keep track of the space available for allocation

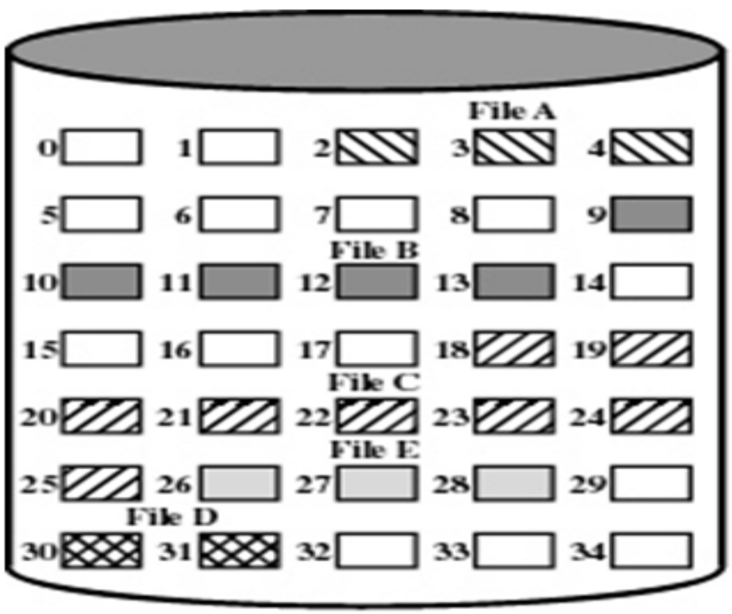
Sounds Familiar?

File Allocation Strategies

- Contiguous allocation
  - Contiguous chunk for whole file
- Chained allocation
  - Pointer to next block allocated to file
- Indexed allocation
  - Index block points to file blocks

Methods of File Allocation

- Contiguous allocation
  - Single set of blocks is allocated to a file at the time of creation
  - Only a single entry in the file allocation table
    - Starting block and length of the file



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

External Fragmentation?

Contiguous Allocation

- Advantages
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Easy random accesses
- Disadvantages
  - External fragmentation -- wasteful of space (dynamic storage-allocation problem)
  - Files cannot grow

Contiguous Allocation

- Mapping from logical address (LA) to physical disk address, block size = 512 bytes

LA/512

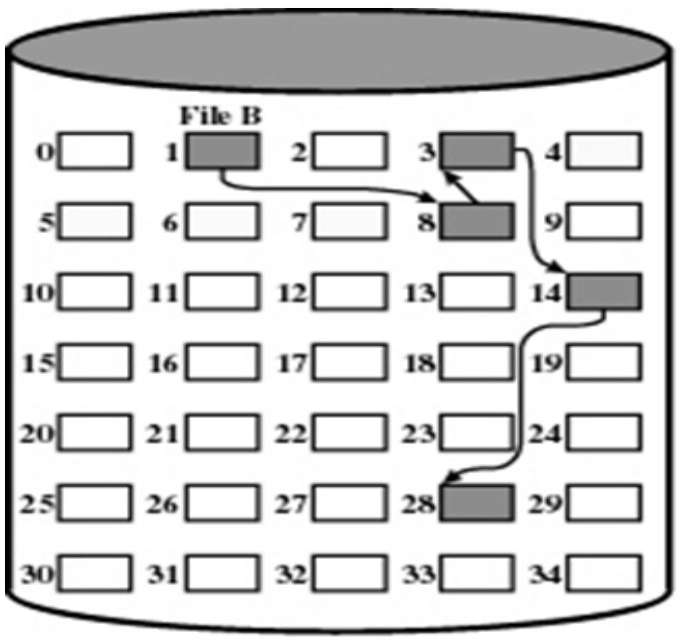
Q ← relative block# to initial block

R ← displacement into block

- Block to be accessed = Q + initial block#
- Displacement into block = R

Methods of File Allocation

- Chained allocation
  - Allocation on basis of individual block
  - Each block contains a pointer to the next block in the chain
  - Only single entry in the file allocation table
    - Starting block and length of file



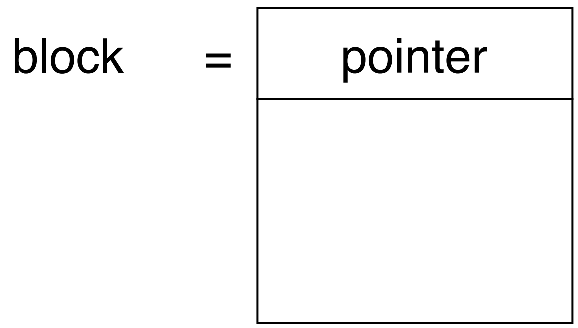
File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

External Fragmentation?

Chained Allocation

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



- Pointer size = 4 bytes, Block size = 512 bytes

Chained Allocation (Cont.)

- (+) Simple – need only starting address
- (+) Free-space management system – no waste of space
- (-) No random access
- Mapping

Q

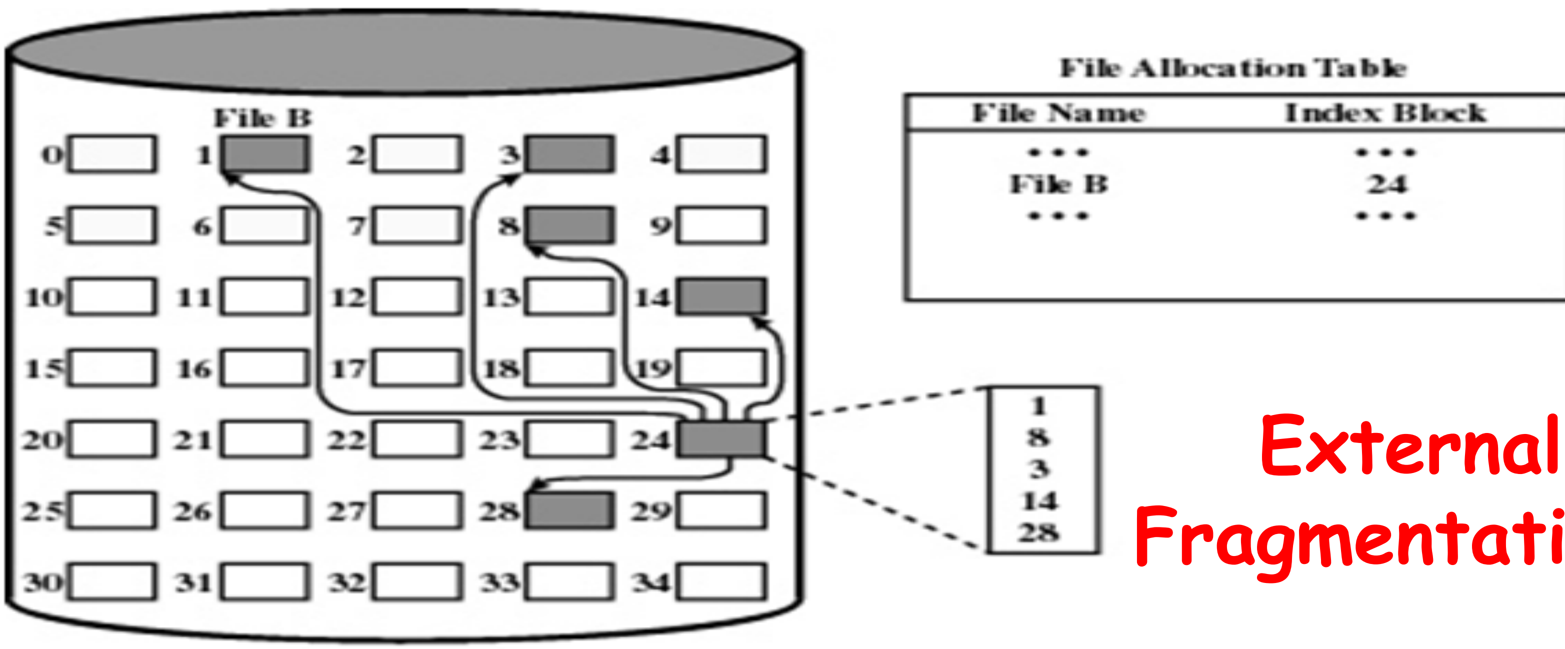
LA/508

R

Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 4

Methods of File Allocation

- Indexed allocation
  - File allocation table (FAT) contains a separate one-level index for each file
  - The index has one entry for each block allocated to the file
  - The FAT contains block number for the index



External Fragmentation?

Indexed Allocation (Cont.)

- (-) Need index table
- (+) Easy random access
- (+) Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a block size of 512 words. We need only 1 block for index table.

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

Q = displacement into index table

R = displacement into block

- Is there a limit on maximum file size given one index block?

Indexed Allocation – Linked Mapping

- Mapping from logical to physical in a file of unbounded length (block size of 512 words, pointer size of 1 word).
- Linked scheme – Link blocks of index table (no limit on size).

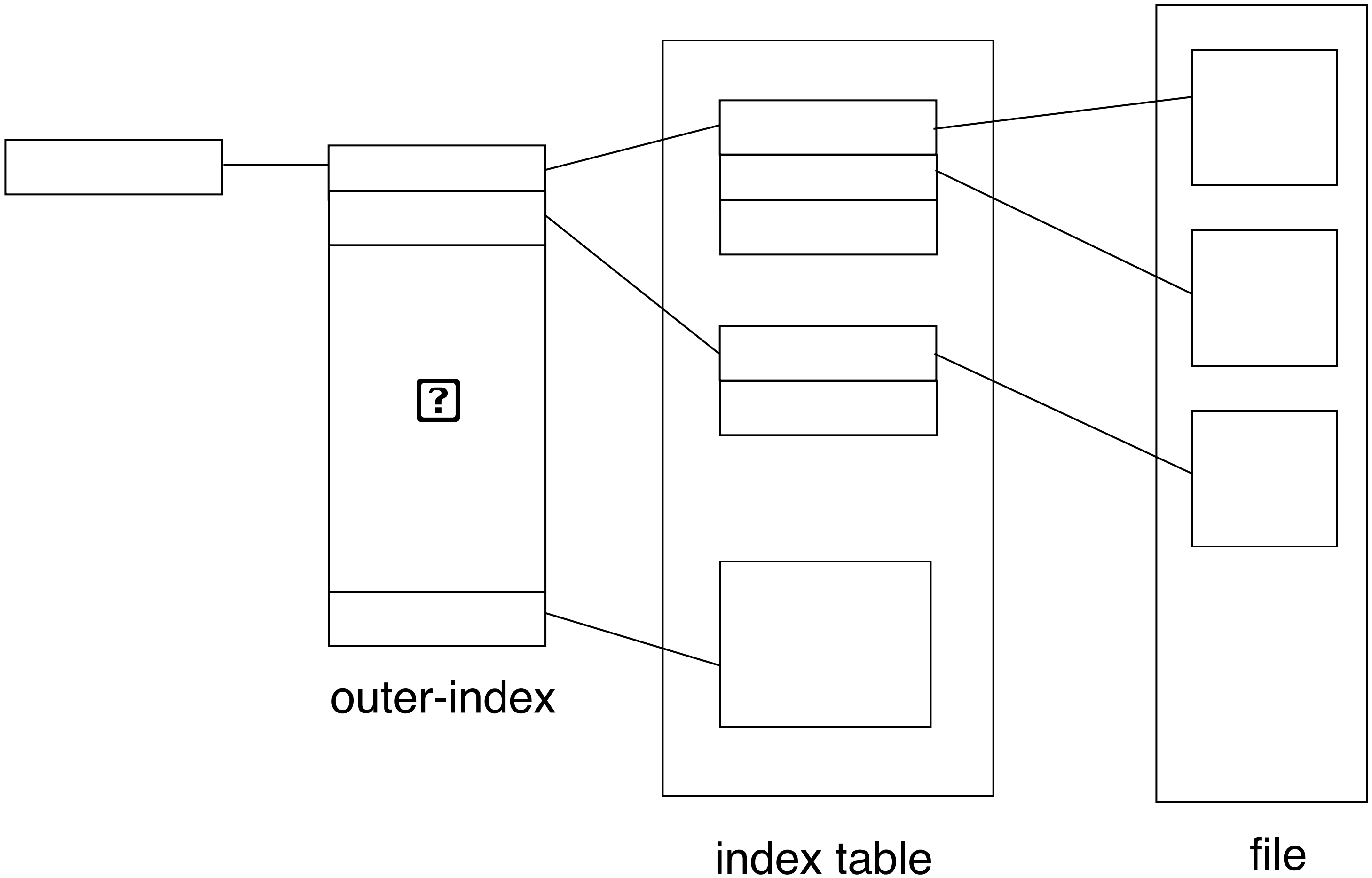
$$LA / (512 \times 511) \begin{cases} Q_1 \\ R_1 \end{cases}$$

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 – Two level index



Free Space Management

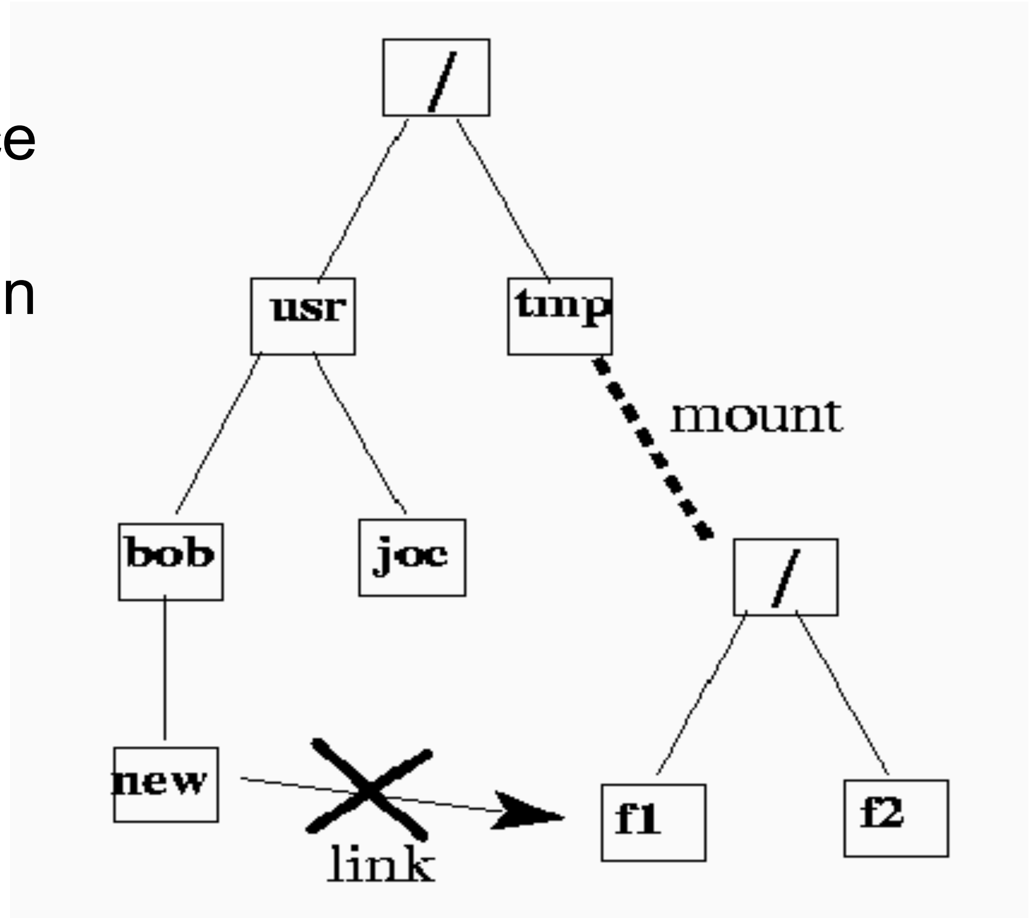
- Bitmap: one bit for each block on the disk
  - Small enough to be kept in memory
  - Requires sequential scan of bits
- Chained free portions: pointer to the next one
- Indexed: treats free space as a file

Case Study w/ Unix File System

- Directories
  - File of files
  - Organized as a rooted tree
  - Pathnames (relative and absolute)
  - Contains links to parent, itself
  - Multiple links to files can exist
    - Link – hard or symbolic

Case Study w/ Unix File System

- Tree-structured file hierarchies
- Mounted on existing space by using mount
- Originally no links between different file systems → later changed



Case Study w/ Unix File System

- File Naming: Each file has a unique name
  - External name (visible to user) must be symbolic
    - In a hierarchical file system, unique external names are given as pathnames (path from the root to the file)
  - Internal names: **i-node** in UNIX
    - An index into an array of file descriptors/headers for a volume
  - Directory: translation from external to internal name
- Information about file is split between the directory and the file descriptor

Case Study w/ Unix File System

- Contents of Unix I-Node
  - File Owner
  - Current size
  - Number of links
  - User ID and group ID
  - Read / write / execute protection bits
  - Access / create / update timestamps
  - Pointers to data

Unix File Allocation Scheme