# Principles of Operating Systems

Lecture 8 - File-System Interface and Implementation

Ardalan Amiri Sani (ardalan@uci.edu)

[*lecture slides contains some content adapted from previous slides by Prof. Nalini Venkatasubramanian, and course text slides © Silberschatz]*

# File Concept

- Contiguous logical address space for storage
  - OS creates the abstraction of a logical storage unit called file.
  - (Typically) persistent
  - OS maps files to physical devices.
- Types
  - Data
    - numeric, character, binary
  - Program
    - Source file, object file
  - Documents

# File Structure

- None - sequence of words/bytes
- Simple record structure
  - Lines
  - Fixed Length
  - Variable Length
- Complex Structures
  - Formatted document
  - Relocatable executable file
- Can achieve last two with first method by inserting appropriate control characters
- Who decides
  - Operating System
  - Program

# File Attributes

- Name
  - symbolic file-name; human-readable name
- Identifier
  - Unique tag that identifies file within file-system; non-human readable name
- Type
  - for systems that support multiple types
- Location
  - pointer to a storage device and to file location on device
- Size
  - current file size, maximum possible size
- Protection
  - controls who can read, write, execute
- Time, date, and user (owner) identification
  - data for protection, security and usage monitoring

Where is this information stored?

# File Attributes

- Name
  - symbolic file-name; human-readable name
- Identifier
  - Unique tag that identifies file within file-system; non-human readable name
- Type
  - for systems that support multiple types
- Location
  - pointer to a storage device and to file location on device
- Size
  - current file size, maximum possible size
- Protection
  - controls who can read, write, execute
- Time, date, and user (owner) identification
  - data for protection, security and usage monitoring

Where is this information stored?

Information about files are kept in the directory structure, maintained on disk

# File types - name.extension

| *File Type* | *Possible extension* | *Function* |
|---|---|---|
| Executable | Exe,com,bin | Machine language program |
| Object | Obj, o | Compiled machine lang., not linked |
| Source code | c, CC, p, java, asm… | Source code in various languages |
| Batch | Bat, sh | Commands to command interpreter |
| text | Txt, doc | Textual data, documents |
| Print, view | ps, dvi, gif | ASCII or binary file |
| archive | Arc, zip, tar | Group of files, sometimes compressed |
| Library | Lib, a | Libraries of routines |

# File Operations

- A file is an abstract data type. It can be defined by operations:
  - Create a file
  - Write a file
  - Read a file
  - Reposition within file - file seek
  - Delete a file
  - Truncate a file
  - Open(Fi)
    - search the directory structure on disk for entry Fi, and move the content of entry to memory.
  - Close(Fi)
    - move the content of entry Fi in memory to directory structure on disk.

# Directory Structure

- Number of files on a system can be large
  - Break file systems into partitions (treated as a separate storage device)
  - Hold information about files within partitions.
- Device Directory: A collection of nodes containing information about all the files on a partition.
- Both the directory structure and files reside on disk.

# Operations Performed on Directory

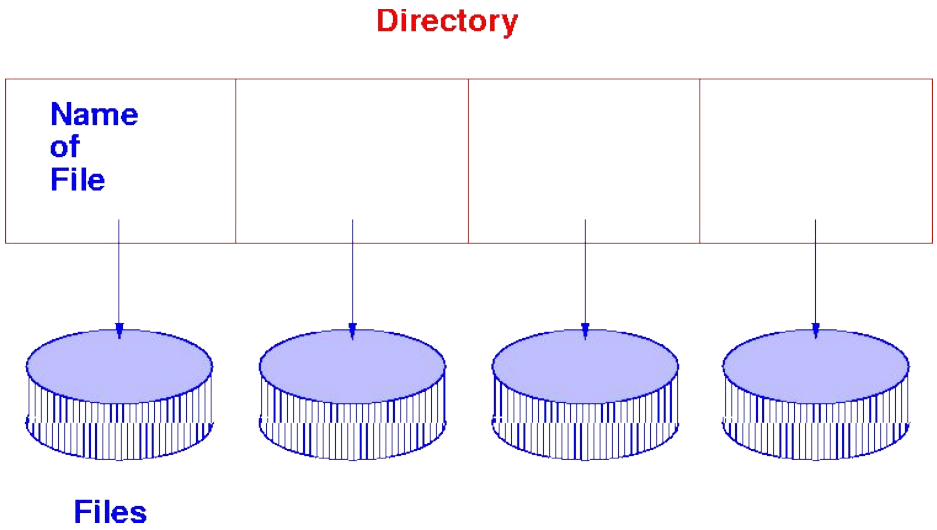
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file-system

# Logical Directory Organization -- Goals

- Efficiency - locating a file quickly
- Naming - convenient to users
  - Two users can have the same name for different files.
  - The same file can have several different names.
- Grouping
  - Logical grouping of files by properties (e.g., all Python programs, all games, all pictures…)

# Single Level Directory

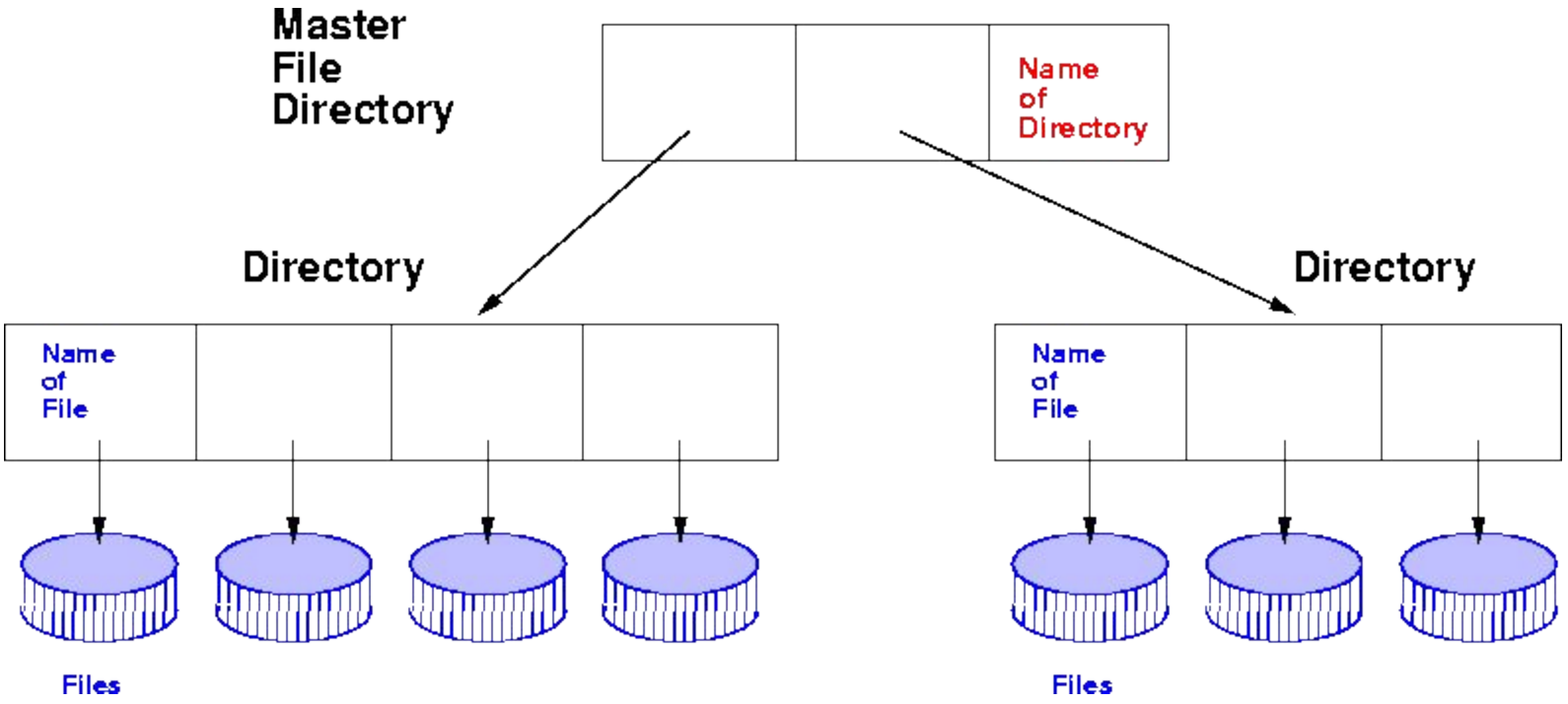
- A single directory for all users
- Naming faces problems as file names must be unique
  - As the number of files increases, difficult to remember unique names
  - As the number of users increase, users must have unique names.
- Grouping not possible



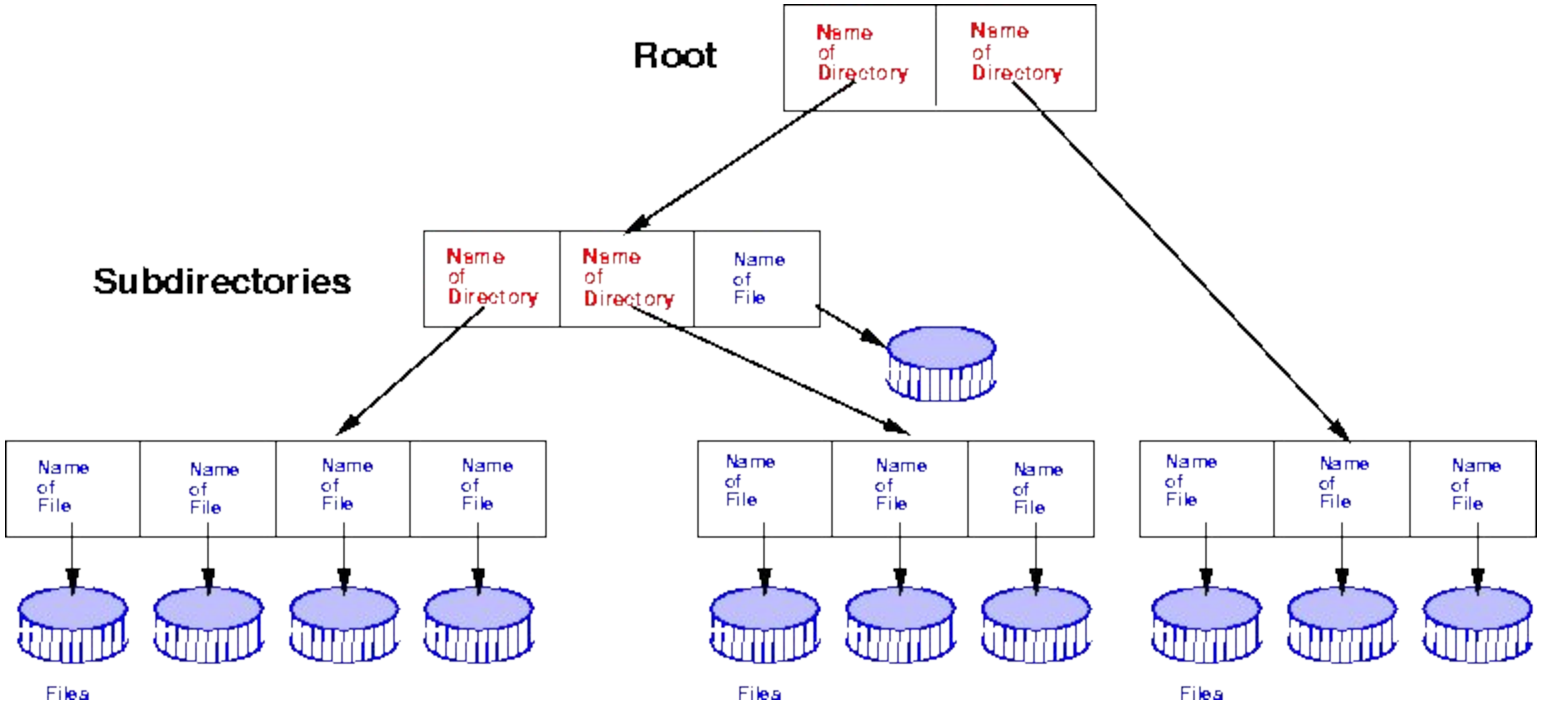
# Two Level Directory

- Introduced to remove naming problem between users
  - First Level contains list of user directories
  - Second Level contains user files
  - Need to specify Path name
  - Can have same file names for different users. But each user must still use unique names among her own files.
  - System files kept in separate directory or Level 1.

# Two Level Directory



# Tree structured Directories



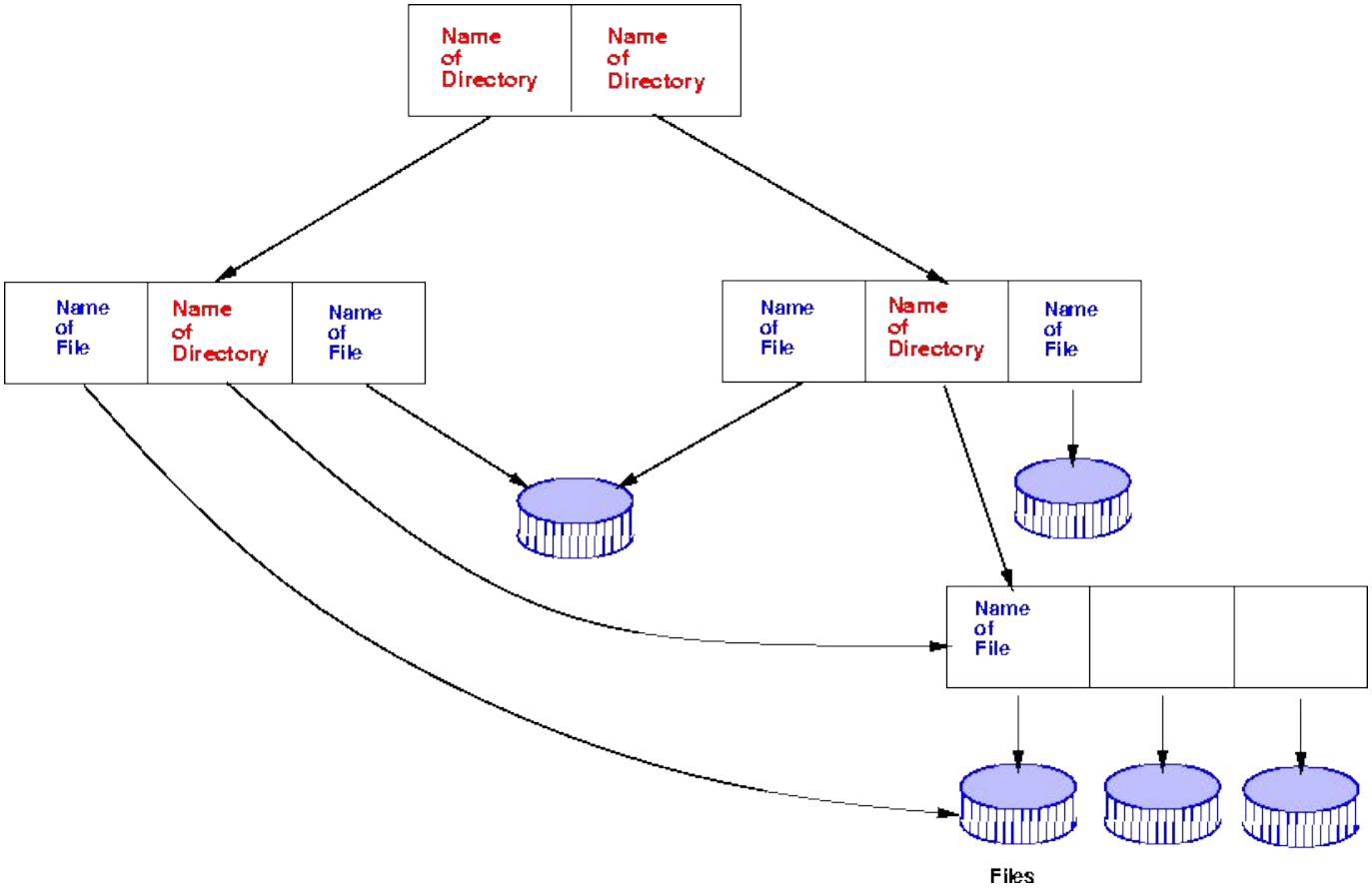
# Tree Structured Directories

- Arbitrary depth of directories
  - Leaf nodes are files, interior nodes are directories.
- Efficient Searching
- Grouping Capability

# Tree Structured Directories

- Absolute or relative path name
  - Absolute from root
  - Relative paths from current working directory pointer.
- Creating a new file or subdirectory (or deleting them) is done in current directory unless path is provided)

# Acyclic Graph Directories



# Acyclic Graph Directories

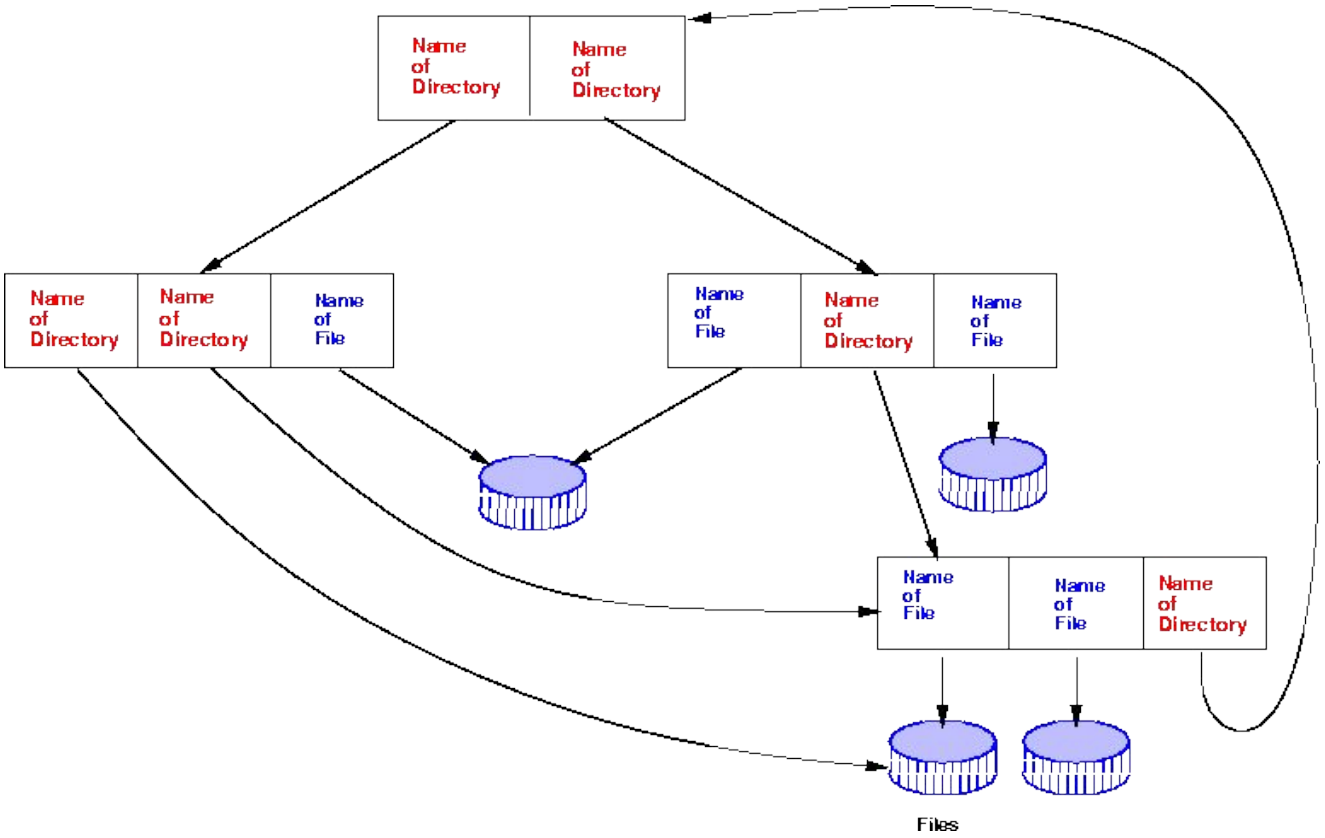
- Acyclic graphs allow sharing
- Implementation by links (symbolic link)
  - Links are pointers to other files or subdirectories
  - Need to resolve link to locate file.

- Implementation by shared files (hard link)
  - Duplicate information in sharing directories
  - Original and copy indistinguishable.
  - Need to maintain consistency if one of them is modified.

# Acyclic Graph Directories

- Naming : File may have multiple absolute path names
- Traversal
  - ensures that shared data structures are traversed only once.
- Deletion
  - Removing file when someone deletes it may leave dangling pointers.
  - Preserve file until all references to it are deleted
    - Keep a list of all references to a file or
    - Keep a count of the number of references - *reference count.*
    - When count = 0, file can be deleted.

# General Graph Directories



# General Graph Directories (cont.)

- How do we guarantee no cycles in a tree structured directory?
  - Allow only links to file not subdirectories.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is ok.
- If links to directories are allowed, we have a simple graph structure
  - Need to ensure that components are not traversed twice both for correctness and for performance, e.g. search can be non-terminating.

# Access Methods

- Sequential Access

  read next
  write next
  reset

- Direct Access ( n = relative block number)

  read n
  write n

  or

  position to n
  read next
  write next

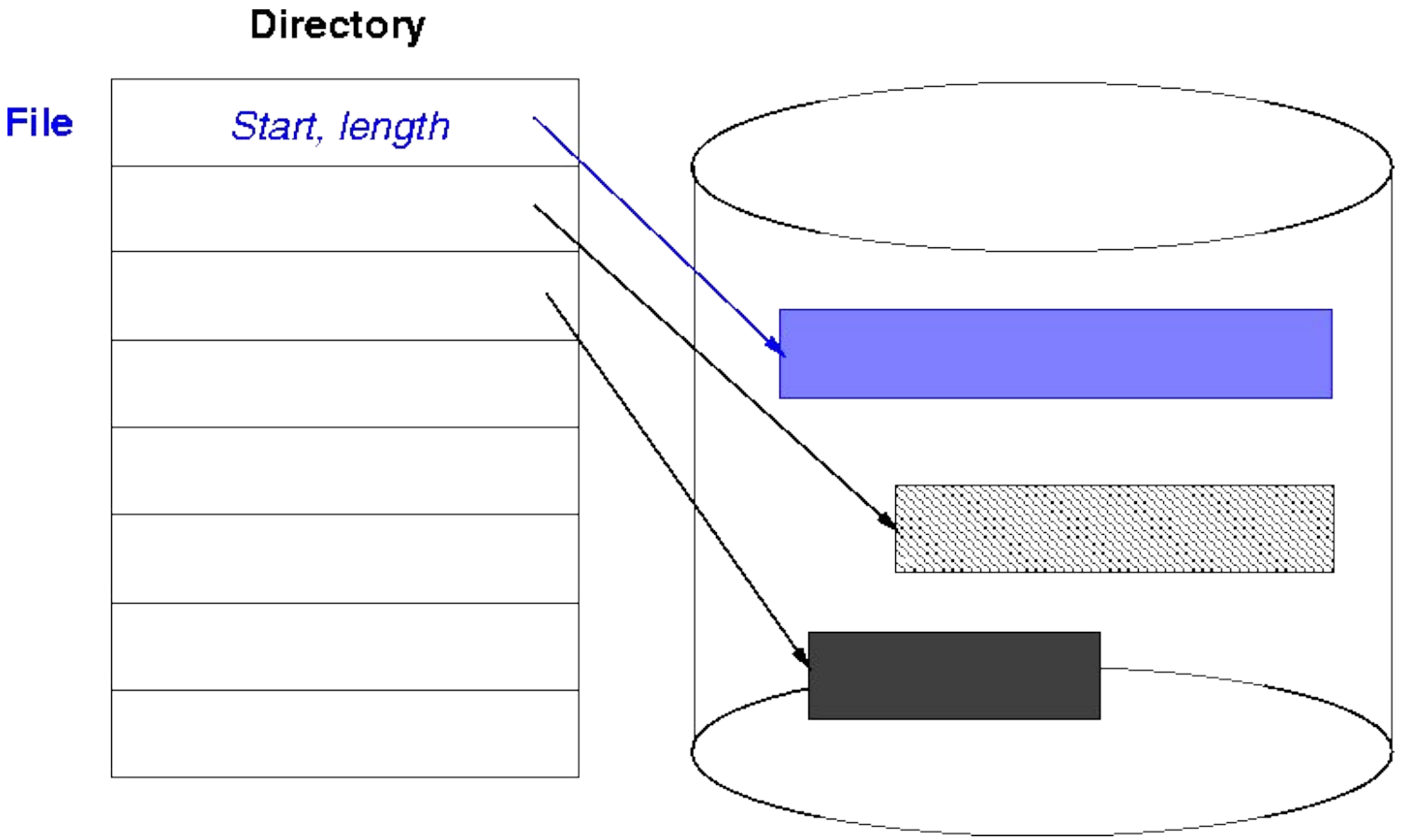
# Protection

- File owner/creator should be able to control
  - what can be done
  - by whom
- Types of access
    - read
    - write
    - execute
    - append
    - delete
    - list
- In UNIX, 3 fields of length 3 bits are used.
    - Fields are user, group, others (u,g,o),
    - Bits are read, write, execute (r,w,x).

# Allocation of Disk Space

- Low level access methods depend upon the disk allocation scheme used to store file data
  - Contiguous Allocation
  - Linked Allocation
  - Indexed Allocation

# Contiguous Allocation

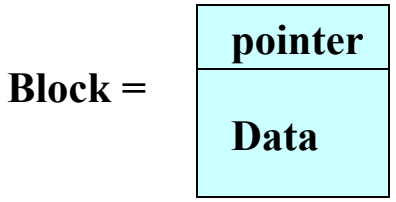


# Contiguous Allocation

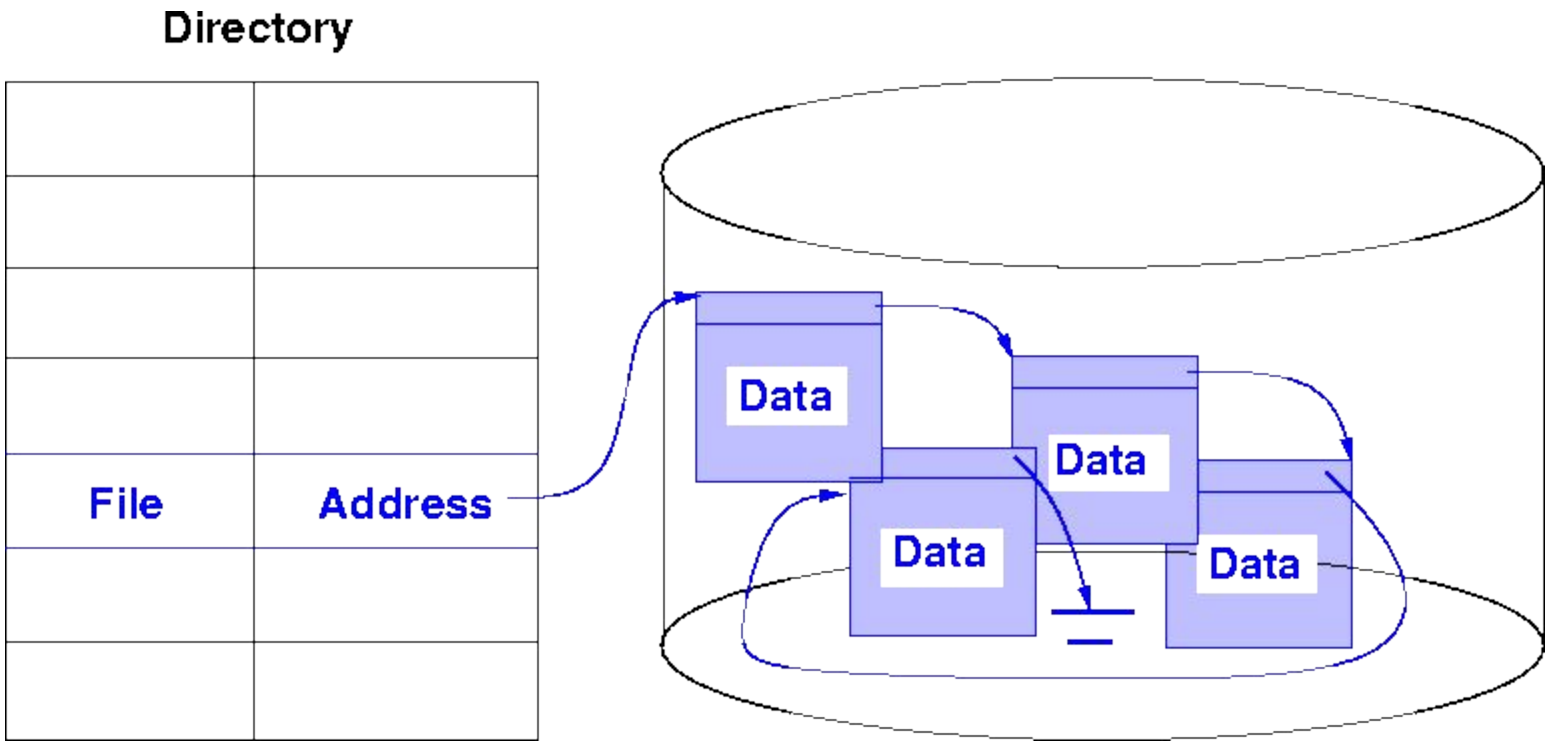
- Each file occupies a set of contiguous blocks on the disk.
  - Simple - only starting location (block #) and length (number of blocks) are required.
  - Suits sequential or direct access.
  - Fast (very little disk head movement)
  - Easy to recover in the event of disk data corruption.
  - Problems
    - Wasteful of space (dynamic storage-allocation problem). Use first fit, best fit, etc. Leads to external fragmentation on disk.
    - Files cannot grow if there's no space - in this case, expanding file requires copying
    - Users tend to overestimate space - internal fragmentation.
- Mapping from logical to physical - <Q,R>
  - File block number = Q -> block number on the disk = Q + file starting block
  - Displacement into block = R

# Linked Allocation

- Each file is a linked list of disk blocks
  - Blocks may be scattered anywhere on the disk.
  - Each node in list can be a fixed size physical block (or a contiguous collection of blocks called clusters).
  - Allocate as needed and then link together via pointers.
    - Disk space used to store pointers: if disk block is 512 bytes, and pointer (disk address) requires 4 bytes, user sees 508 bytes of data per block.
  - Pointers in list not accessible to user.



# Linked Allocation



# Linked Allocation

- Easy to add to the file
  - Can grow in middle and at ends. No estimation of size necessary.
- Suited for sequential access but not random access.
- Simple directory structure - need only starting address.

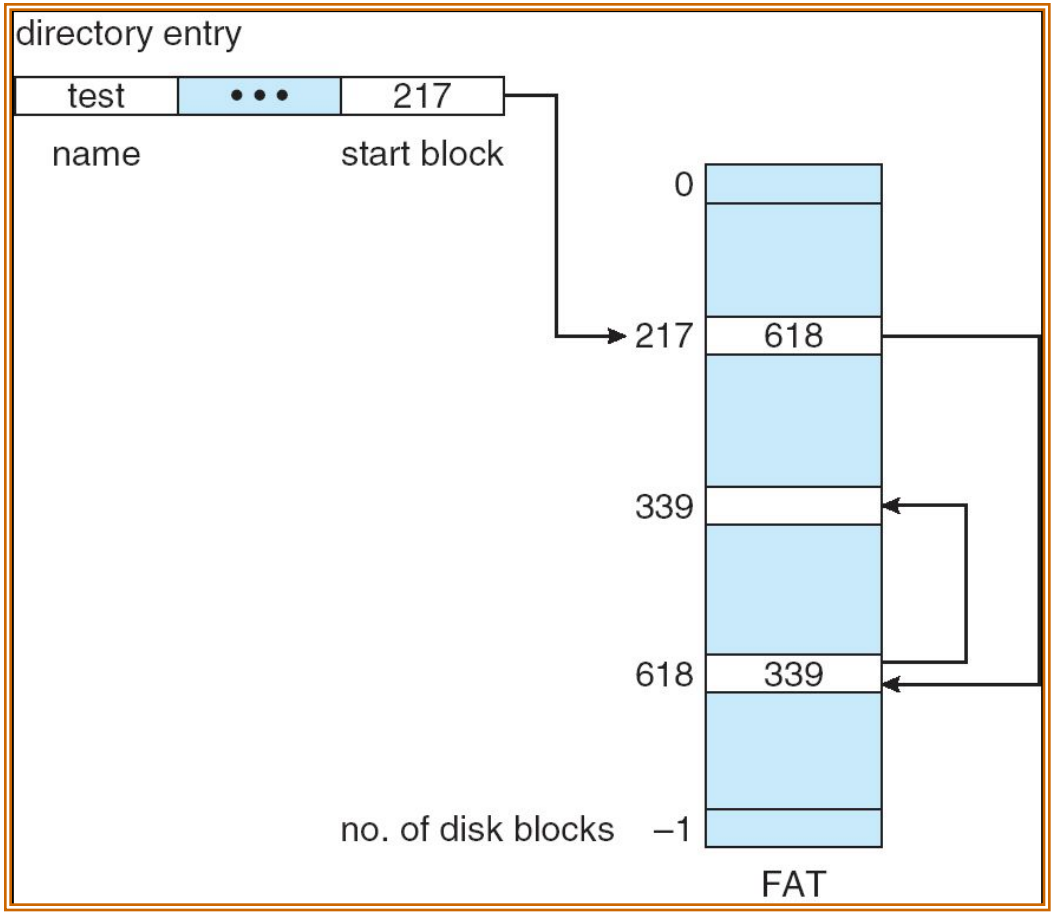
- Mapping - <Q, R>
  - File block number = Qth block in the linked chain of blocks representing the file.
  - Displacement into block = R (must consider the pointer space in the block)

# Linked Allocation (cont.)

- Slow
  - Need to read through linked list nodes sequentially to find the record of interest.
- Not very reliable
  - Disk data corruption can make it very hard to recover the files.
- Important variation on linked allocation method
  - File-allocation table (FAT) - disk-space allocation used by MS-DOS.

# File Allocation Table (*FAT)*

- Instead of link on each block, put all links in one table
  - the *F*ile *A*llocation *T*able — i.e., *FAT*
- One entry per physical block in disk
  - Directory points to first block of file
  - Each block points to next block (or *End Of File (EOF)*)



# FAT File Systems

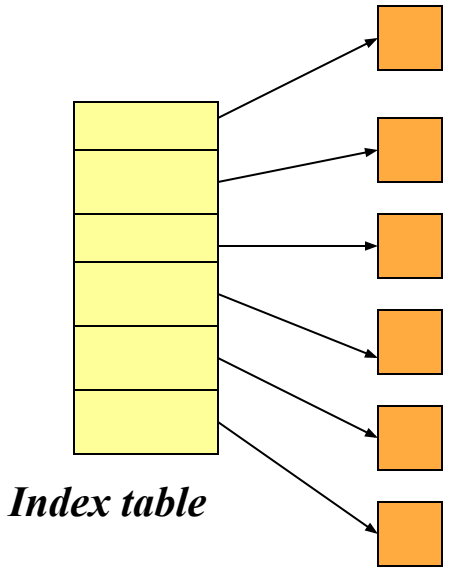
- Advantages
  - Advantages of Linked File System
  - FAT can be *cached* in memory
  - Fast search, pseudo-random access
- Disadvantages
  - Not suitable for very large disks
    - FAT cache describes *entire* disk, not just open files!
  - Not fast enough for large databases

# Disk Defragmentation

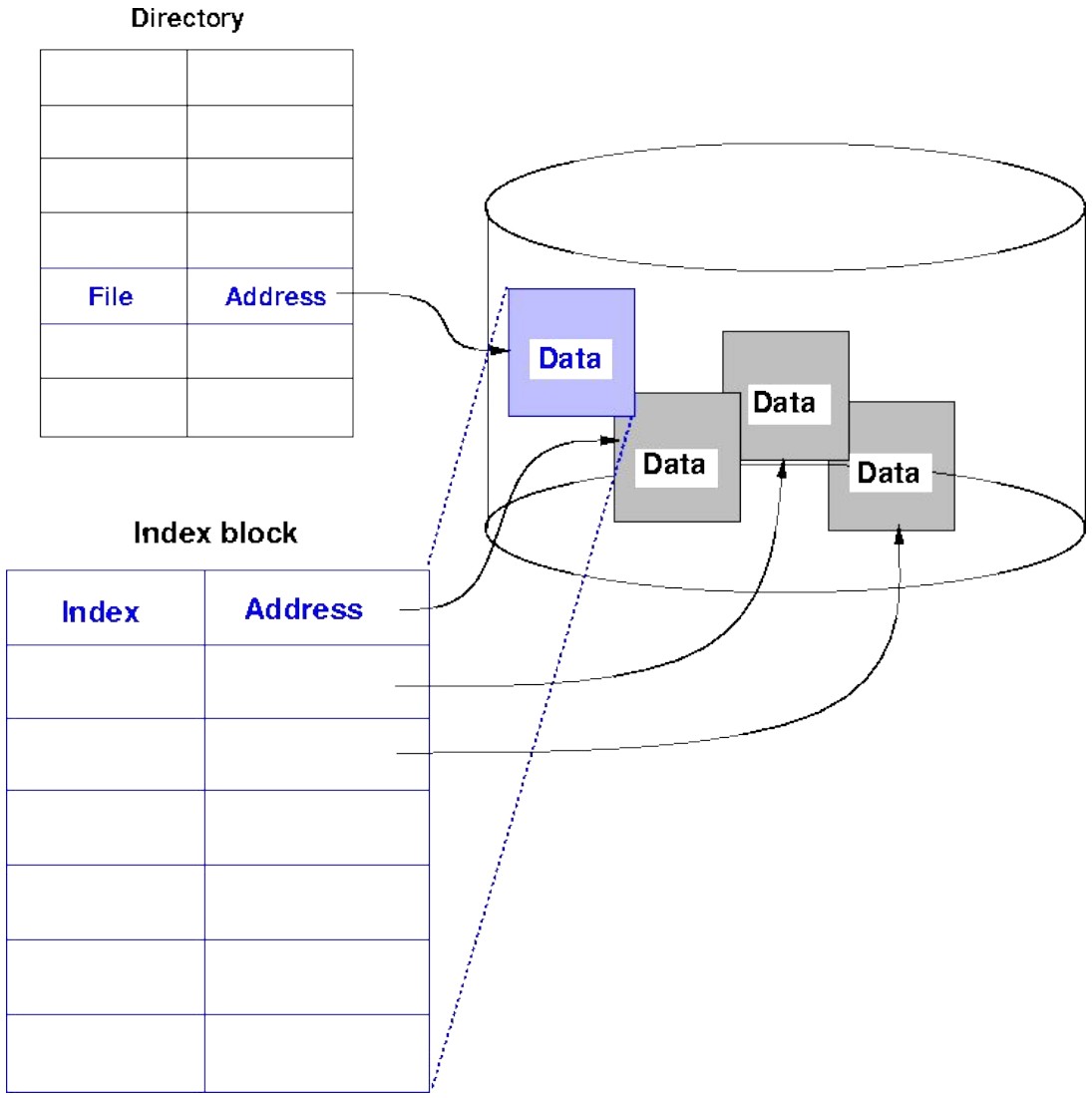
- Re-organize blocks in disk so that file is (mostly) contiguous
- Link or FAT organization preserved
- Purpose:
  - To reduce disk arm movement during sequential accesses

# Indexed Allocation

- Brings all pointers together into the index block.
- Logical view



# Indexed Allocation





35

# Indexed Allocation (cont.)

- Need index table (space overhead).
- Supports sequential and direct access.
- No external fragmentation.
- Mapping - <Q,R>
  - Q - file block number = displacement into index table
  - R - displacement into block

# Indexed Allocation - Mapping

- Index table size limits the size of the file.
  - Example: assume we only allow 1 block for the index table. Also, assume that block size is 512 words and every pointer is one word. In this case, the max file size is **?** words.

# Indexed Allocation - Mapping

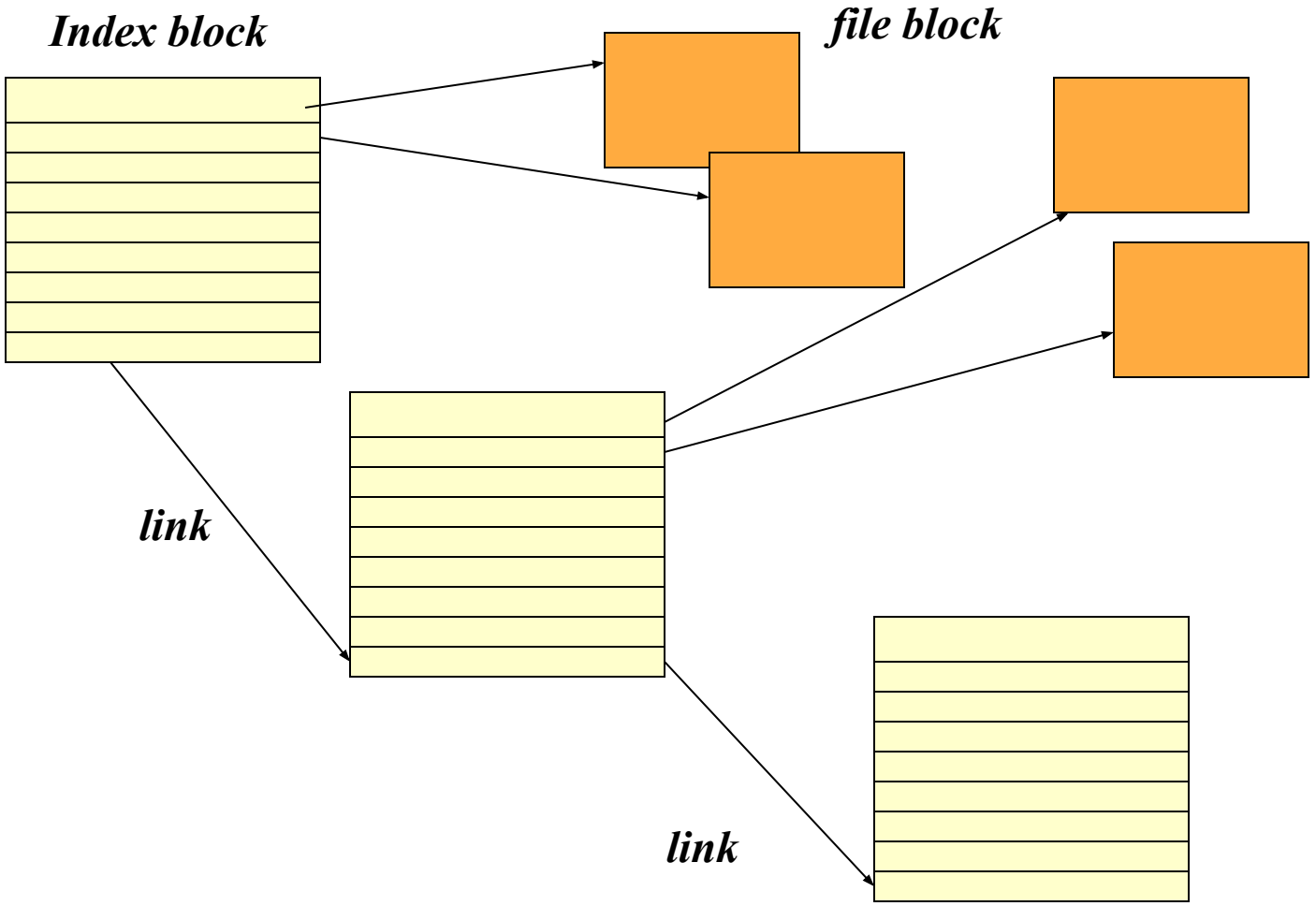
- Index table size limits the size of the file.
  - Example: assume we only allow 1 block for the index table. Also, assume that block size is 512 words and every pointer is one word. In this case, the max file size is 256k words.

# Indexed Allocation - Mapping

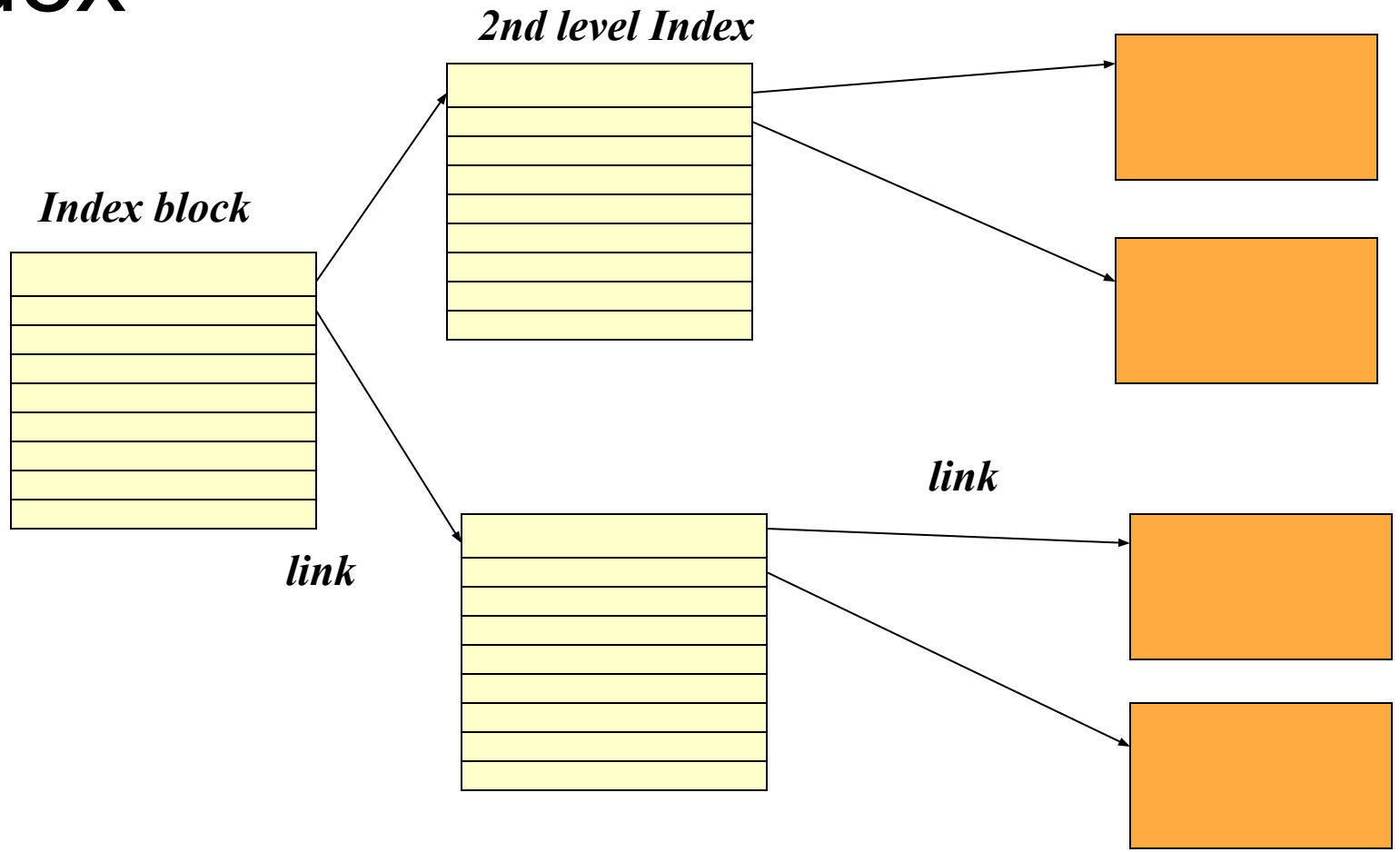
Mapping from logical to physical in a file of large/unbounded length:

- Linked scheme -
  - Link blocks of index tables (no limit on size)
- Multilevel Index
  - E.g., two Level Index - first level index block points to a set of second level index blocks, which in turn point to file blocks.
  - Increase number of levels based on maximum file size desired.
  - Maximum size of file is bounded.

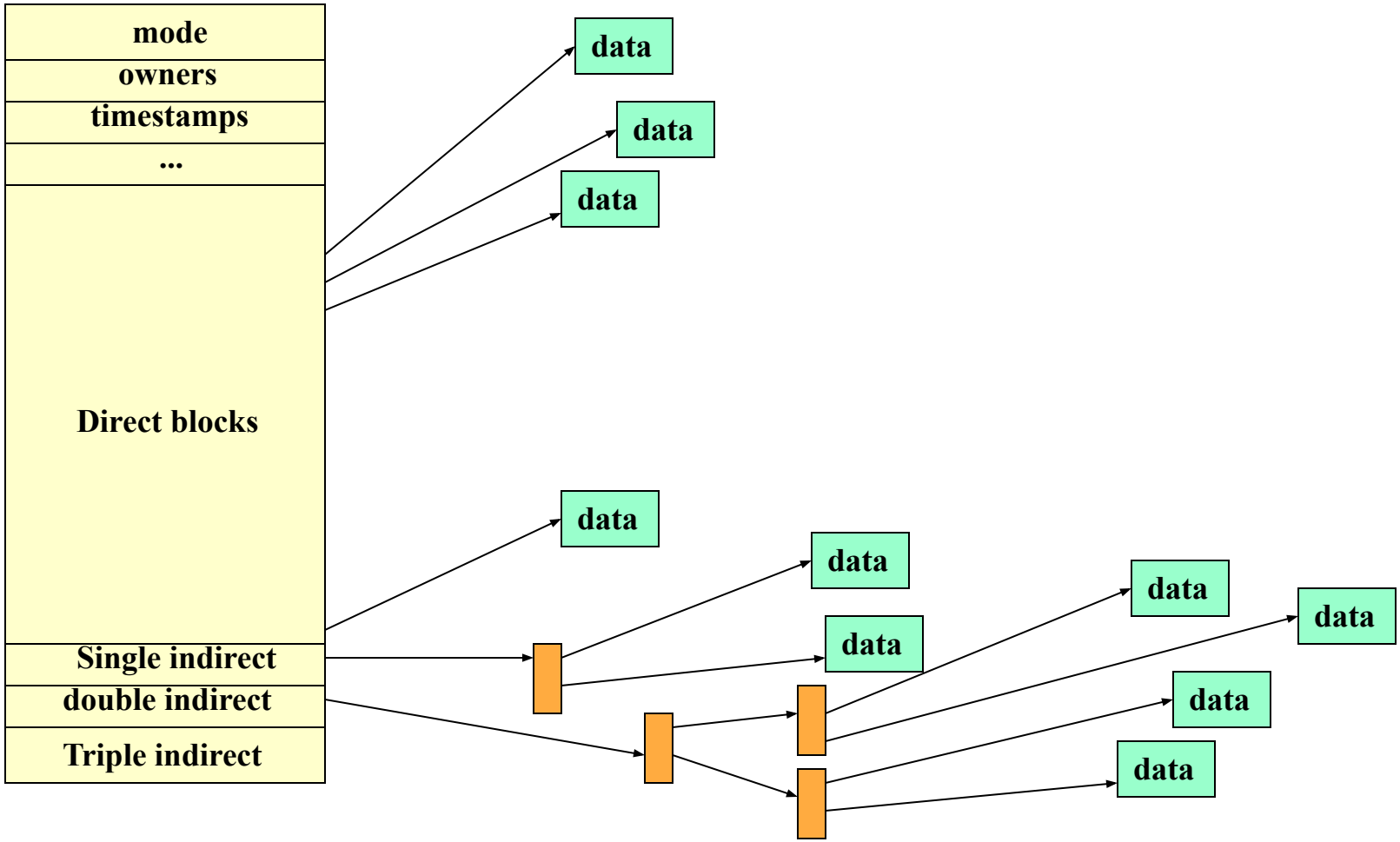
# Indexed File - Linked Scheme



# Indexed Allocation - Multilevel index



# Combined Scheme: UNIX Inode

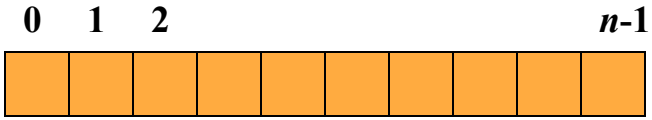


# What is an inode?

- An inode (index node) is a control structure that contains key information needed by the OS to access a particular file. Several file names may be associated with a single inode, but each file is controlled by exactly ONE inode.

# Free Space Management

- Bit Vector (*n* blocks) - bitmap of free blocks




bit[i] = { 1 implies block[i] free
0 implies block[i] occupied

  - Calculating the first free block number:
    (number of bits per word) *
    (number of 0-value words) +
    offset of 1st 1-value bit
  - Bitmap requires extra space.
    - E.g., Block size = 2^12 bytes, Disk size = 2^30 bytes
      n = 2^30/2^12 = 2^18 bits (or 32 kbytes)
  - Easy to find contiguously free blocks

# Free Space Management

- Linked list
  - Keep a linked list of free blocks
  - Cannot get contiguous space easily, not very efficient because linked list needs traversal.
  - No waste of space
- Linked list of indices
  - Keep a linked list of index blocks. Each index block contains addresses of free blocks and a pointer to the next index block.
  - Can find a large number of free blocks quickly.
- Linked list of contiguous blocks
  - Linked list of contiguous blocks that are free
  - Free list node contains pointer and number of free blocks starting from that address.

# Recovery

- Tries to ensure that system failure does not result in loss of data or data inconsistency.
- Consistency checker
  - E.g., compares data in directory structure with data blocks on disk and tries to fix inconsistencies, for example:
    - A block cannot be marked as free and belonging to a file.
    - A block cannot be marked as belonging to two files.
- Backup and restore
  - Use system programs to back up data from disk to another storage device (e.g., magnetic tape).
  - Recover lost file or disk by restoring data from backup.