BITS Pilani
Pilani Campus

Advance Computer Networks (CS G525)

Virendra SHEKHAWAT
Department of Computer Science and Information Systems

Agenda

- File System Interface
  - Files and File System
  - Access Methods
  - Disk and Directory Structure
  - File System Mounting
  - File Sharing
  - Protection
- UNIX File System as an example

What is a File?

- Contiguous logical address space or collection of data with some properties
  - Contents, size, owner, last read/write time, protection, etc…
- Files may also have types
  - Understood by file system
    - device, directory, symbolic link
  - Understood by other parts of OS or by runtime libraries
    - executable, dll, source code, object code, text file, …
- Type can be encoded in the file’s name or contents
  - Windows encodes type in name
    - .com, .exe, .bat, .dll, .jpg, .mov, .mp3, …

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides?
  - Operating system
  - Program

File System

- The concept of a file system is simple
  - The implementation of the abstraction for secondary storage
    - abstraction = files
  - Logical organization of files into directories
    - the directory hierarchy
  - Sharing of data between processes, people and machines
    - access control, consistency, …

File Operations

Unix

- create(name)
- open(name, mode)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)
- rename(old, new)

Windows

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle, …)
- FlushFileBuffers(handle, …)
- SetFilePointer(handle, …)
- CloseHandle(handle, …)
- DeleteFile(name)
- CopyFile(name)
- MoveFile(name)

Access Methods

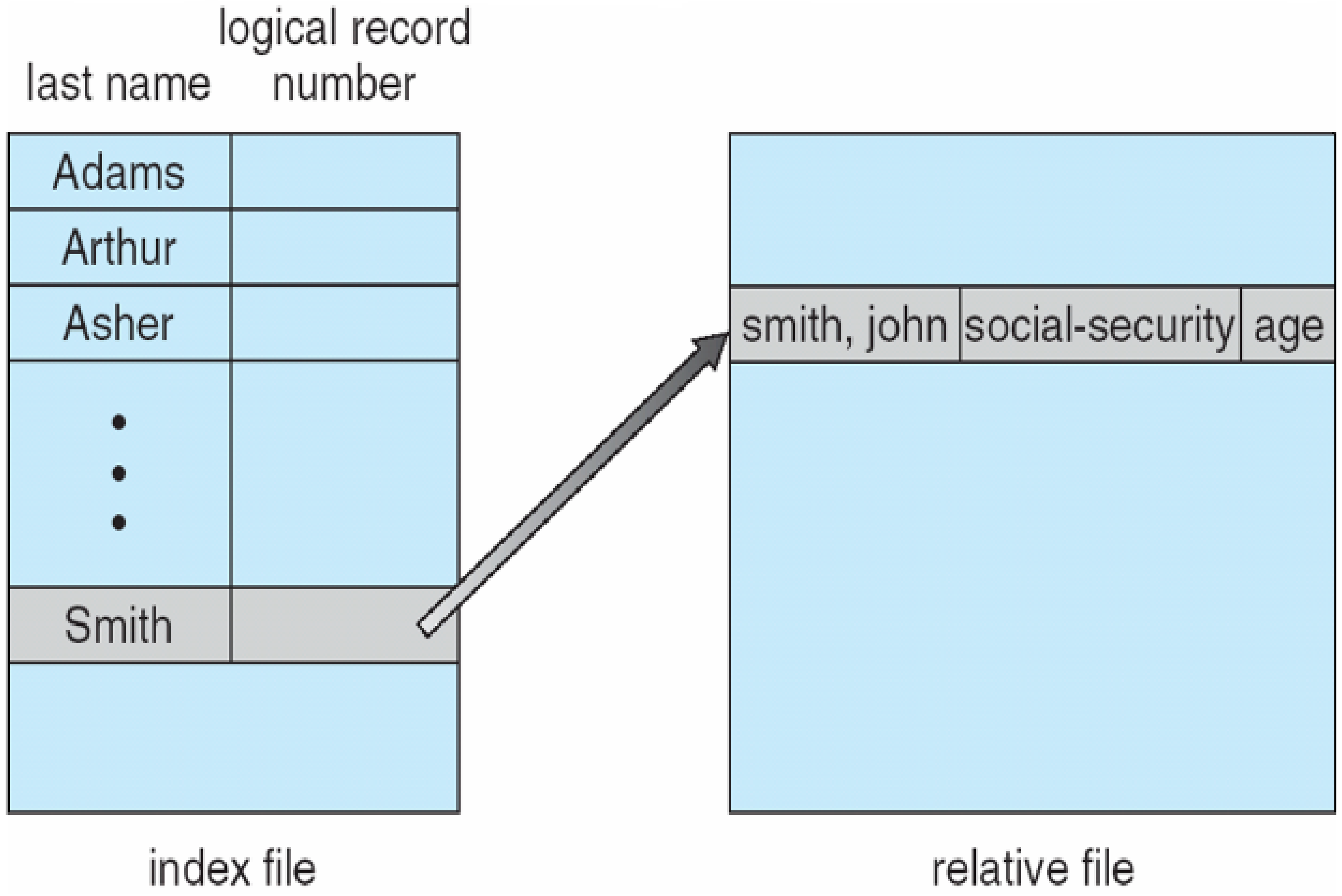
- **Sequential Access**

  ```
  read next
  write next
  reset
  ```
  no read after last write
  (rewrite)
- **Direct Access** – file is fixed length logical records

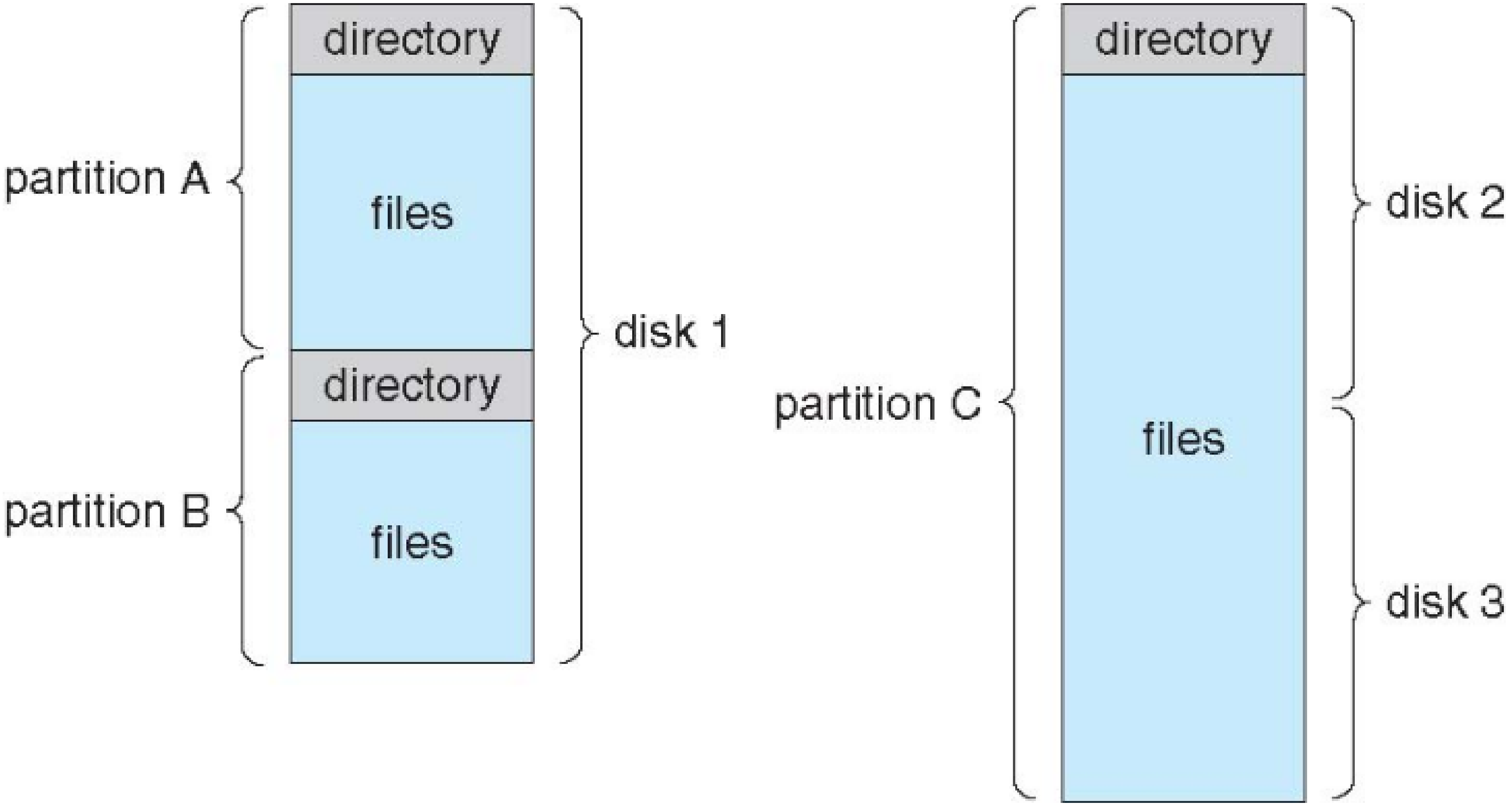
  ```
  read n
  write n
  position to n
      read next
      write next
  rewrite n
  ```

n = relative block number

Other Access Methods- Index



File System Organization



Directory Structure

- Directories provide:
  - A way for users to organize their files
  - e.g., logical grouping of files by properties
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local, /usr/local/bin, …)
- Most file systems support the notion of current directory
  - Absolute names: fully-qualified starting from root of FS

```
$ cd /usr/local
```

  - Relative names: specified with respect to current directory

```
$ cd /usr/local      (absolute)
$ cd bin             (relative, equivalent to cd /usr/local/bin)
```

Directory Internals

- A directory is typically just a file that happens to contain special metadata
  - Directory = list of (name of file, file attributes)
  - Attributes include such things as:
    - Size, protection, location on disk, creation time, access time, …
  - The directory list is usually unordered (effectively random)
    - When you type “ls”, the “ls” command sorts the results for you
- Let’s say you want to open “/one/two/three”

```
fd = open(“/one/two/three”, O_RDWR);
```

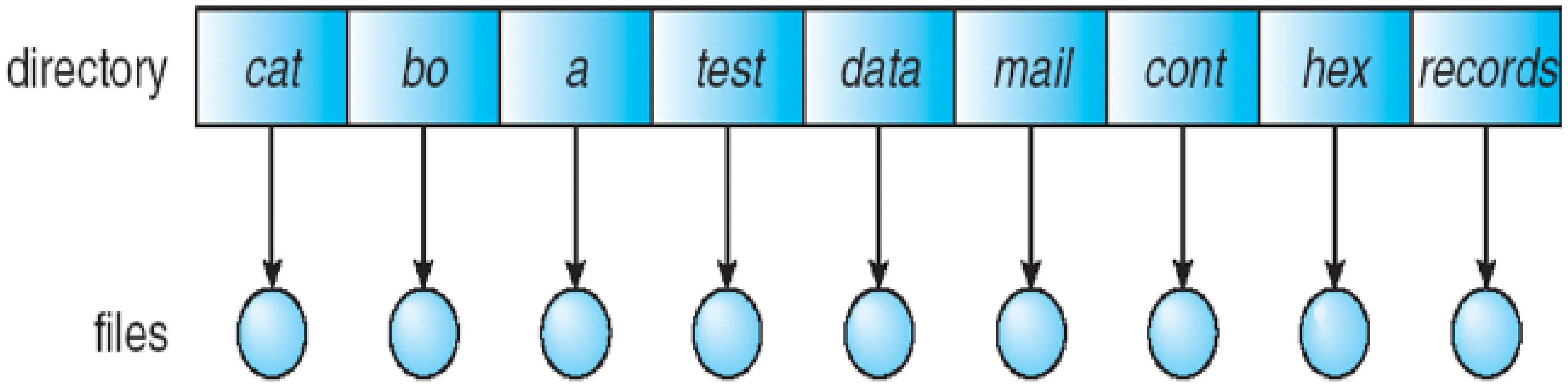
- What goes on inside the file system?

Directory Operations

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

Single Level Directory

- A single directory for all users

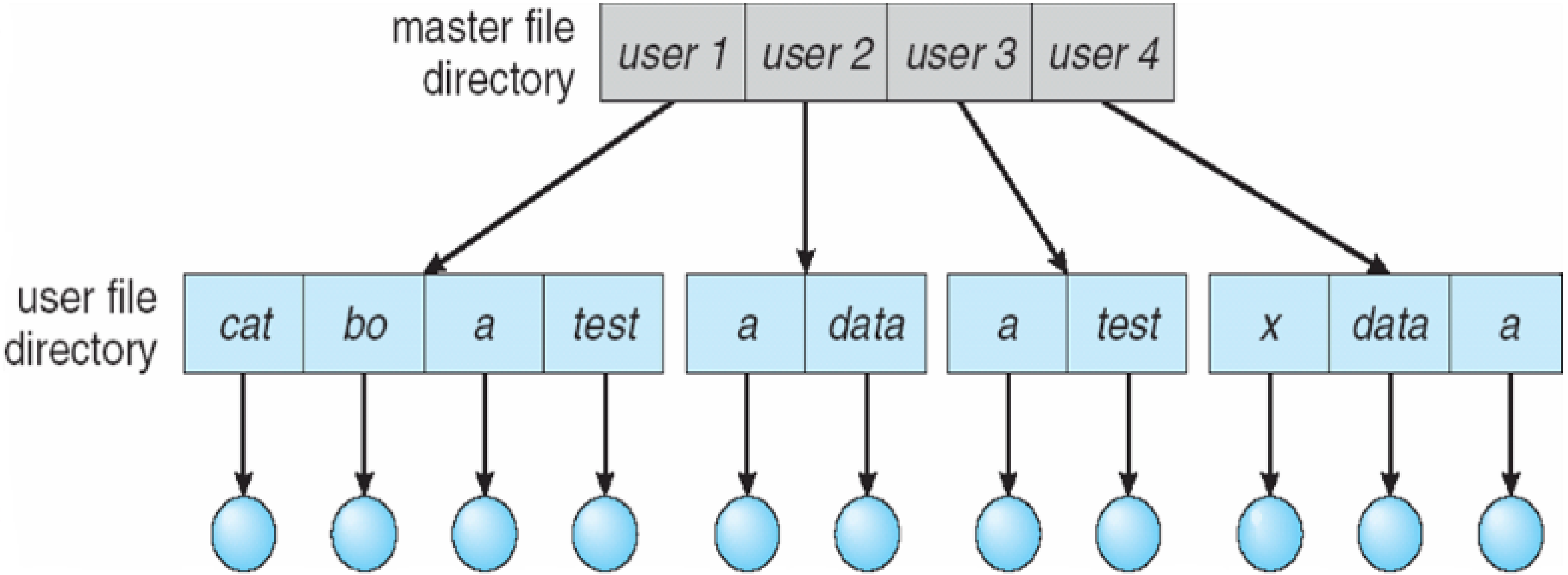


- Naming Problem?
  - Two users can have same name for different files.
  - The same file can have several different names.

- Grouping Problem?

Two Level Directory

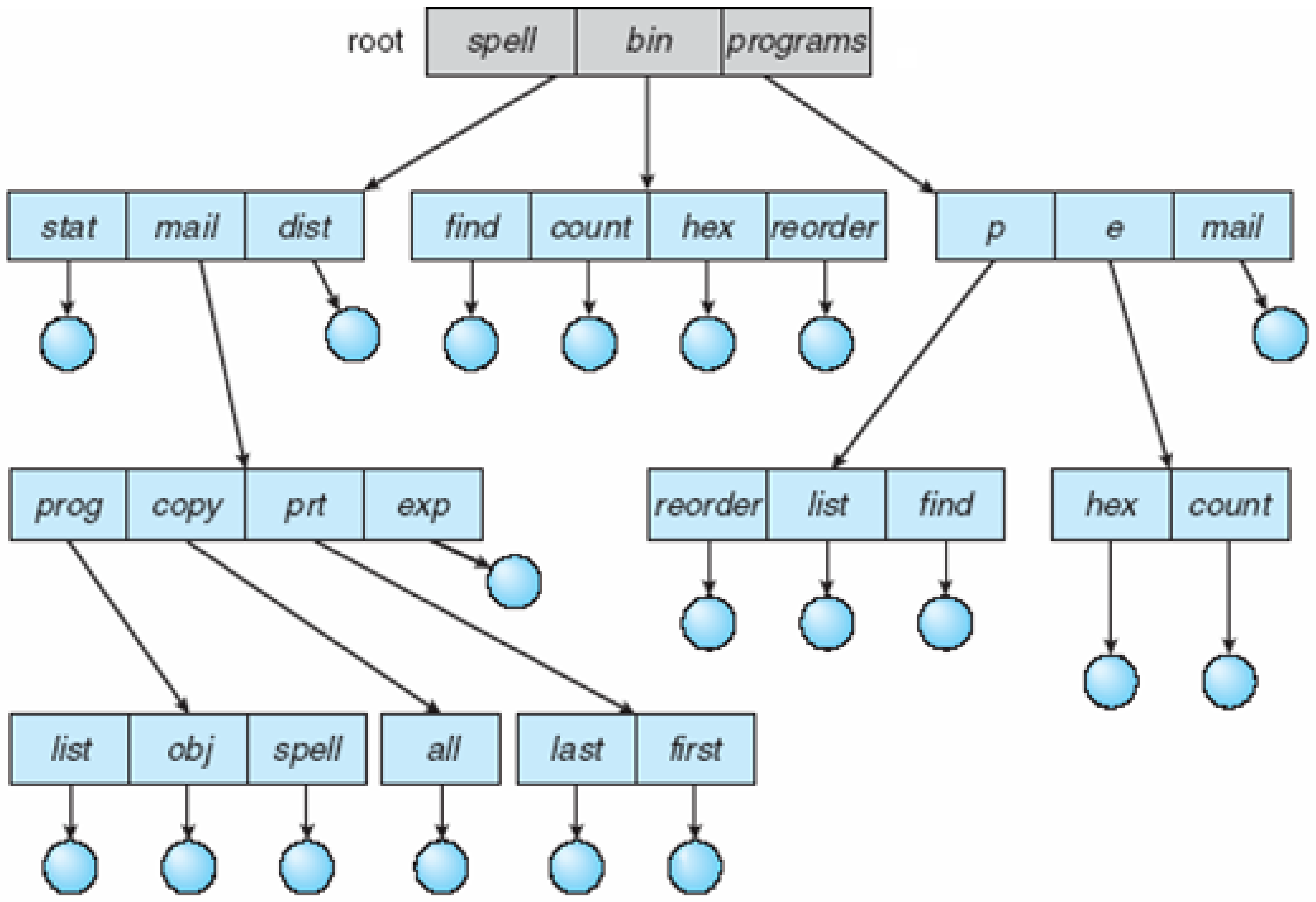
- Separate directory for each user



- Can have the same file name for different user
- Efficient searching
- No grouping capability

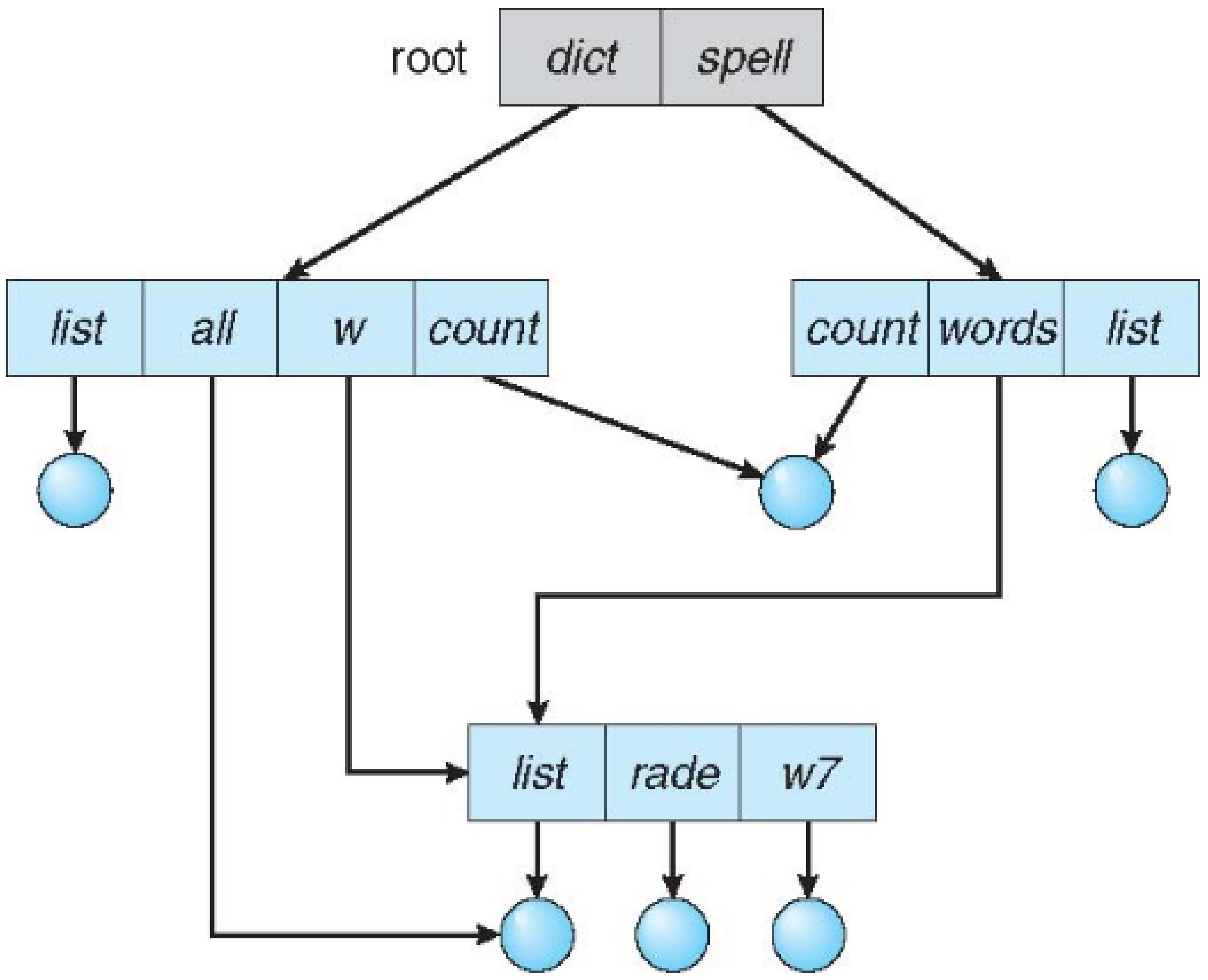
Tree Structured Directories

- Generalization of two-level tree
- Two types of path names
  – Relative and Absolute
- Allows user to have his own subdirectories.
- Policy for deleting a directory
  – Directory must be empty
  – Recursive deletion by UNIX **r m**
- A user can access files of other users.



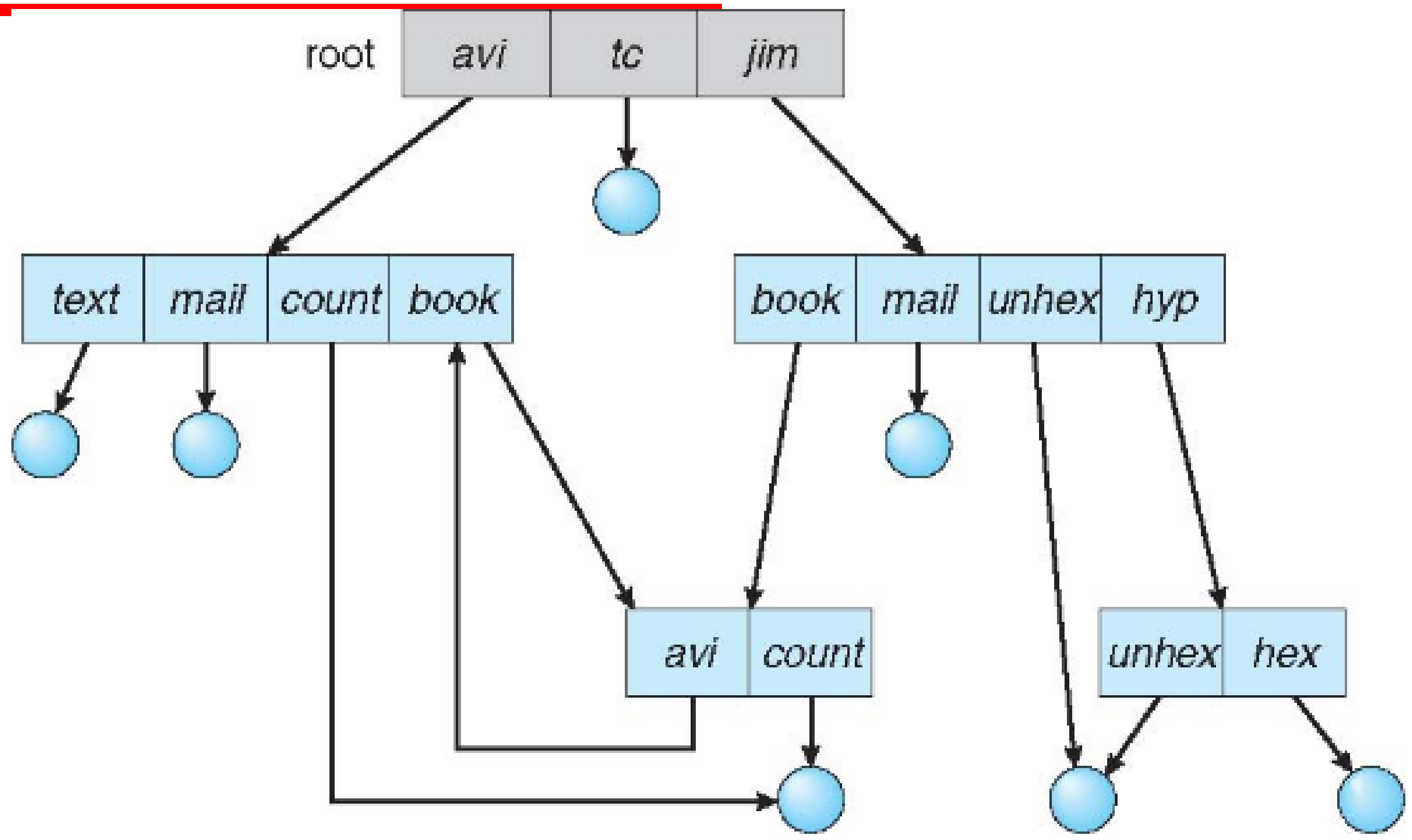
Acyclic Graph Directories

- A tree structure prohibits the sharing of files or directories
  - We can achieve this in acyclic graph
  - **Remember sharing is not copying!!!**
- UNIX Implementation
  - Pointer to another file or directory (link)
  - OS ignores these links when traversing directory trees to preserve acyclic structure
- Acyclic graph directory structure is more flexible but complex. Why?
  - Multiple absolute path names for a file or dir.
  - Think about deletion of a shared file.



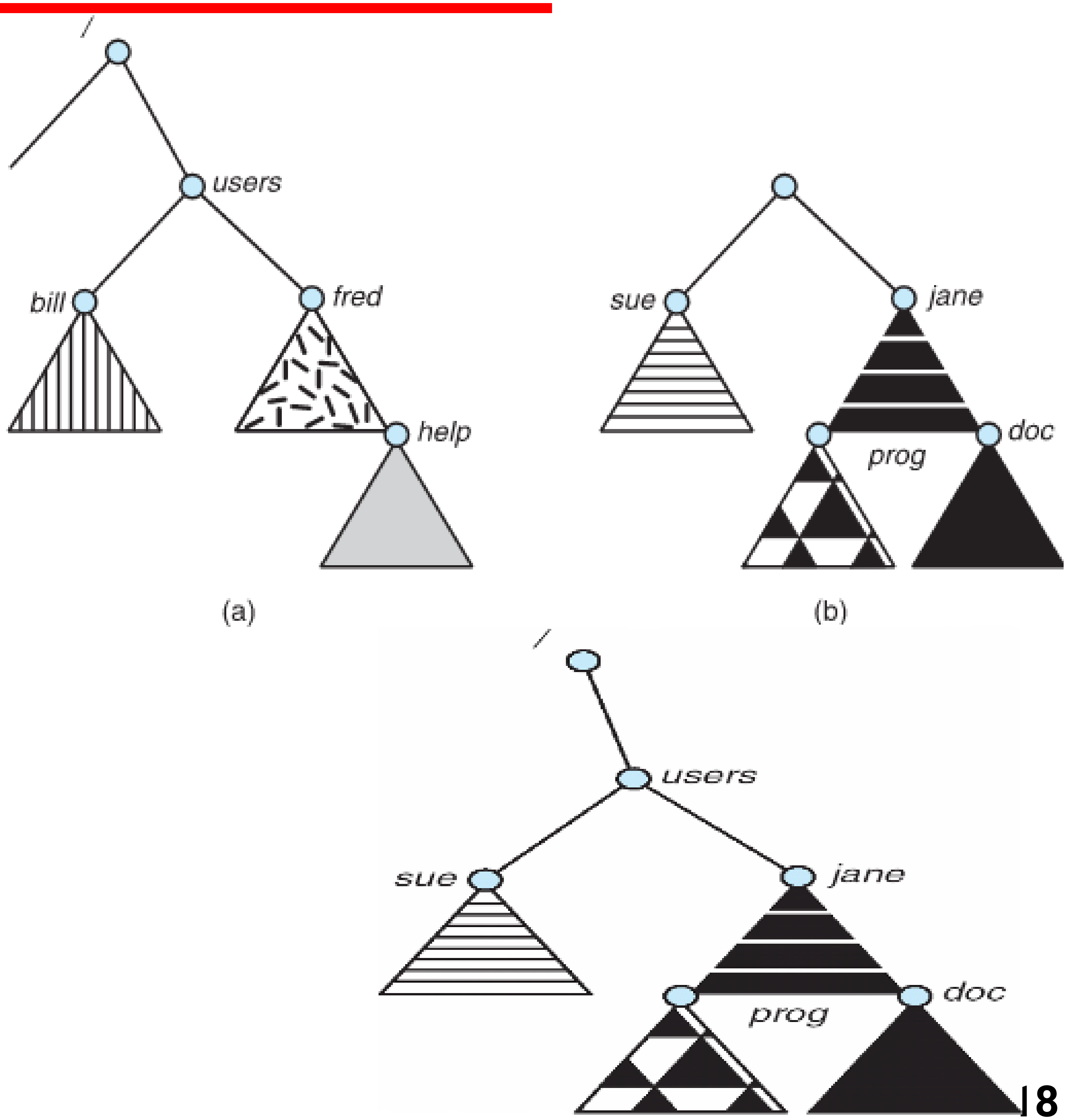
General Graph Directory

- If cycles are allowed to exists in the directory, we want to avoid searching any component twice.
  – For performance and correctness
- How do we guarantee no cycles in graph directory structure?
  – 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
  – **Garbage collection**



File System Mounting

- Just like a file must be opened before it is used, a file system must be mounted before it can be available to access by the processes.
- Mount procedure
  – OS given name of the device and the **mount point**



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

Protection: Ability to Access files

- File owner/creator should be able to control:
  - What can be done and by whom?
- Types of access
  - **Read, Write, Execute, Append, Delete, List**
- Higher level functions like renaming, copying, and editing the file may be implemented by making use of low-level system calls.
  - Copying can be implemented by a sequence of read requests.

Access Control Implementation

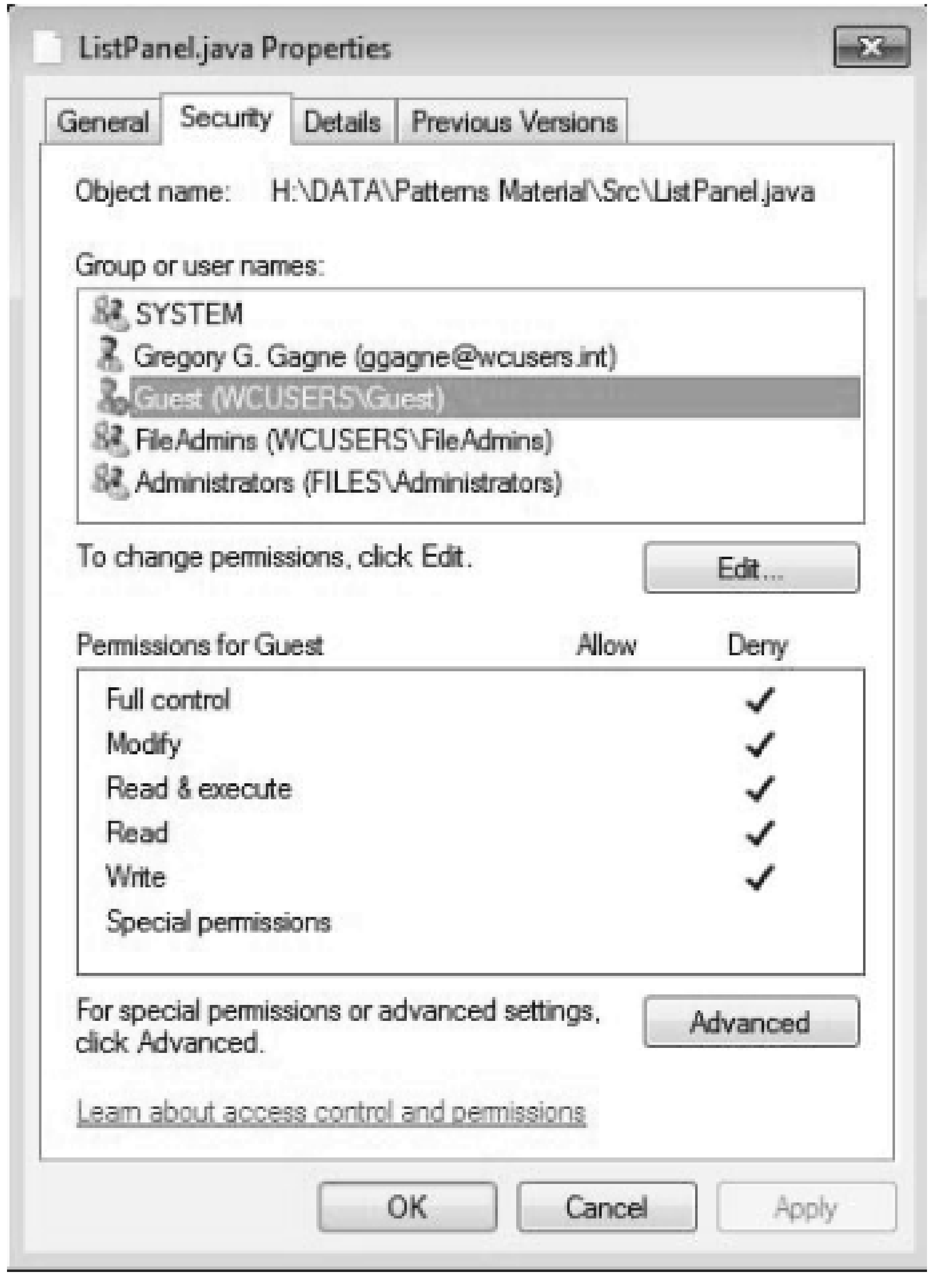
- Different users may need different types of access to a file or directory
  - Maintain an identity dependent access control list (ACL) for each file
    - Enabling complex access methodologies but not practical.
- To condense the length of the ACL, many systems recognize three classifications of users for each file
  - Owner
  - Group
  - Others (Universe)

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	$\Rightarrow$	1 1 1
			RWX
b) **group access**	6	$\Rightarrow$	1 1 0
			RWX
c) **public access**	1	$\Rightarrow$	0 0 1

Windows Access Control List Management



UNIX System Directory Listing

-rw-rw-r--	1	pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5	pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2	pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2	pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1	pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1	pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4	pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3	pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3	pbg	staff	512	Jul 8 09:35	test/

Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, Active Directory implement unified access to information needed for remote computing.

Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously
  - Similar to process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency for remote file systems
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes only visible to sessions starting after the file is closed

Thank You!

File System Implementation

- Details of implementing local file systems and directory structures
- Implementation of remote file systems
- Block allocation and free-block algorithms and trade-offs

Layered File System

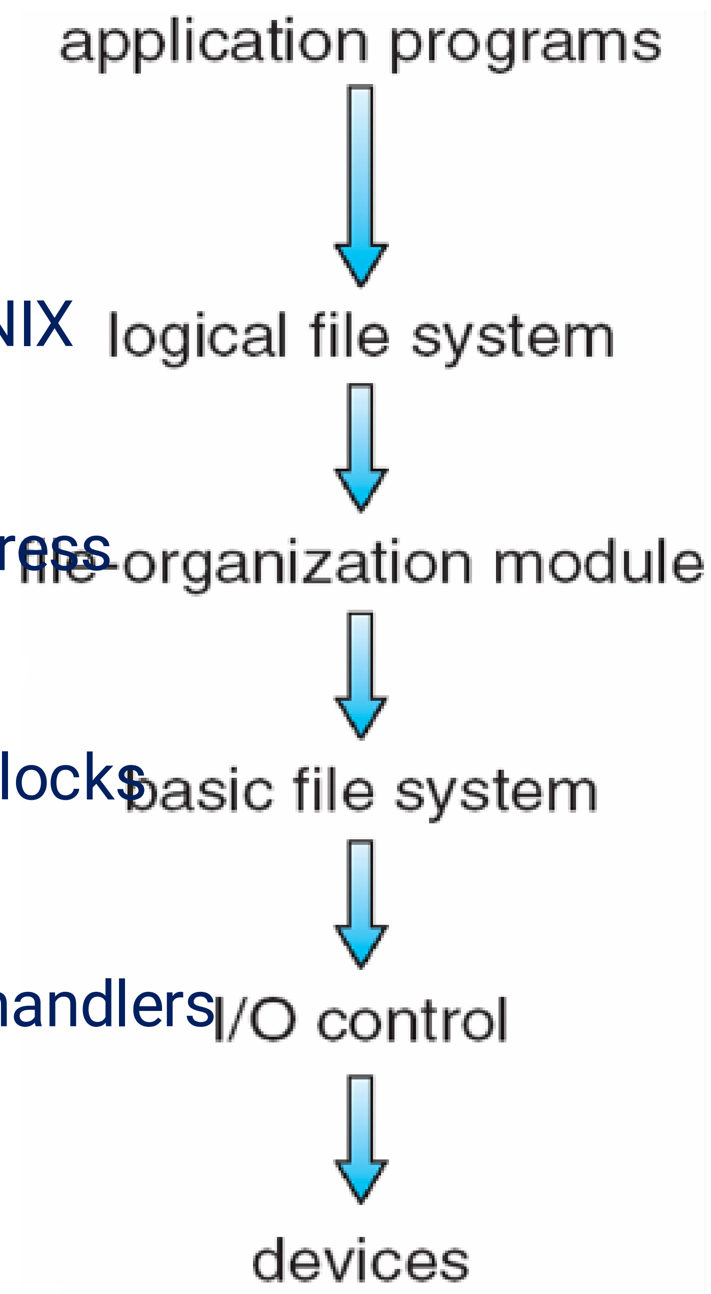
- File System organized into layers

Maintains metadata in the form of FCB or i-node in UNIX

Translates logical block address to physical block address

Issue generic commands to device driver to r/w blocks

Consists of device drivers and interrupt handlers



File-System Implementation

- Several **On-disk** and **in-memory structures** are used to implement a FS
- **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 i-node numbers, master file table

File-System Implementation

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

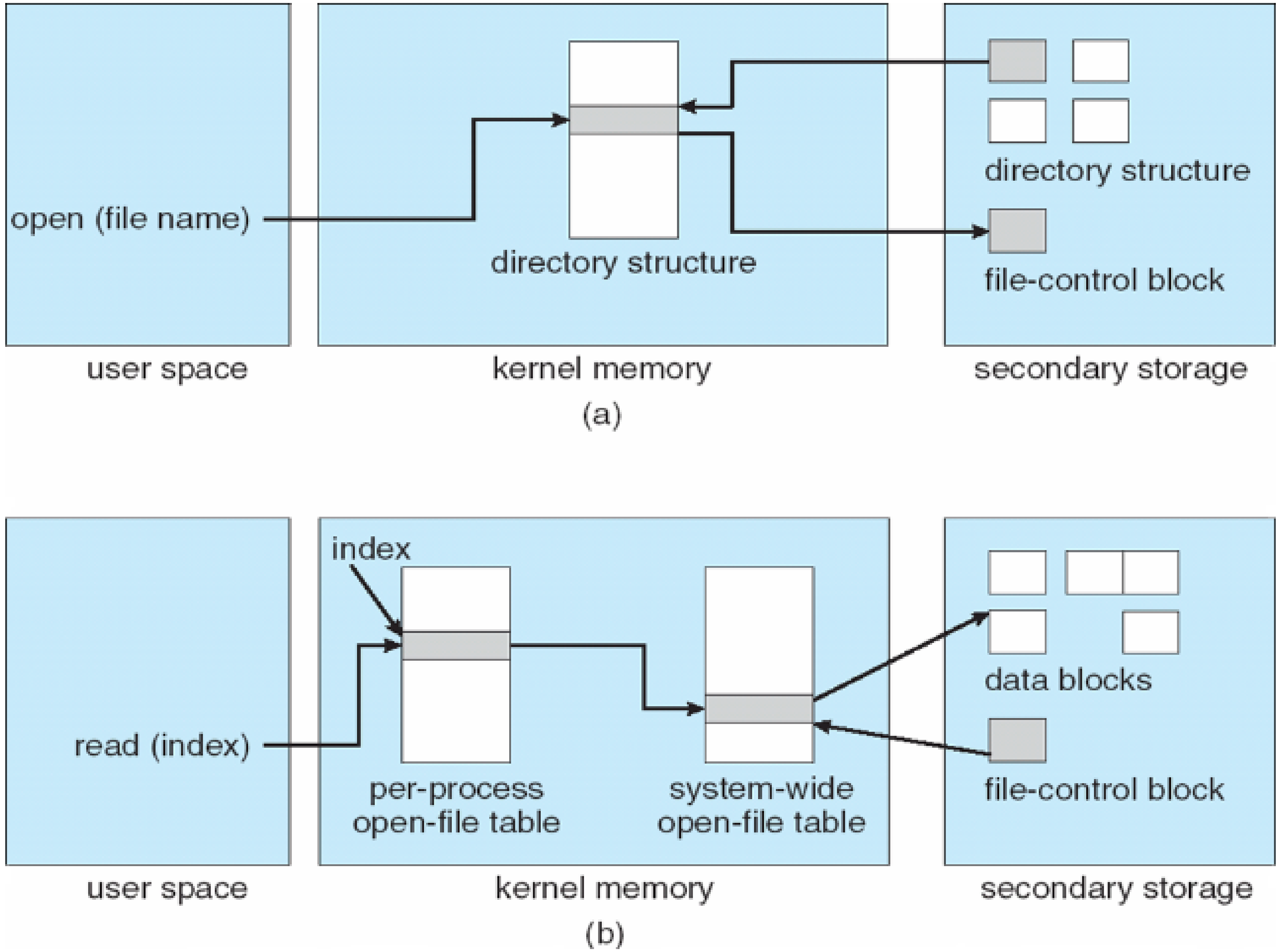
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

(Old) Unix disks are divided into five parts …

- Boot block
  - can boot the system by loading from this block
- Superblock
  - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- i-node area
  - contains descriptors (i-nodes) for each file on the disk; all i-nodes are the same size; head of free list is in the superblock
- File contents area
  - fixed-size blocks; head of free list is in the superblock
- Swap area
  - holds processes that have been swapped out of memory

In-Memory File System Structures

- Mount table storing file system mounts, mount points, file system types
- 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



Directory Implementation

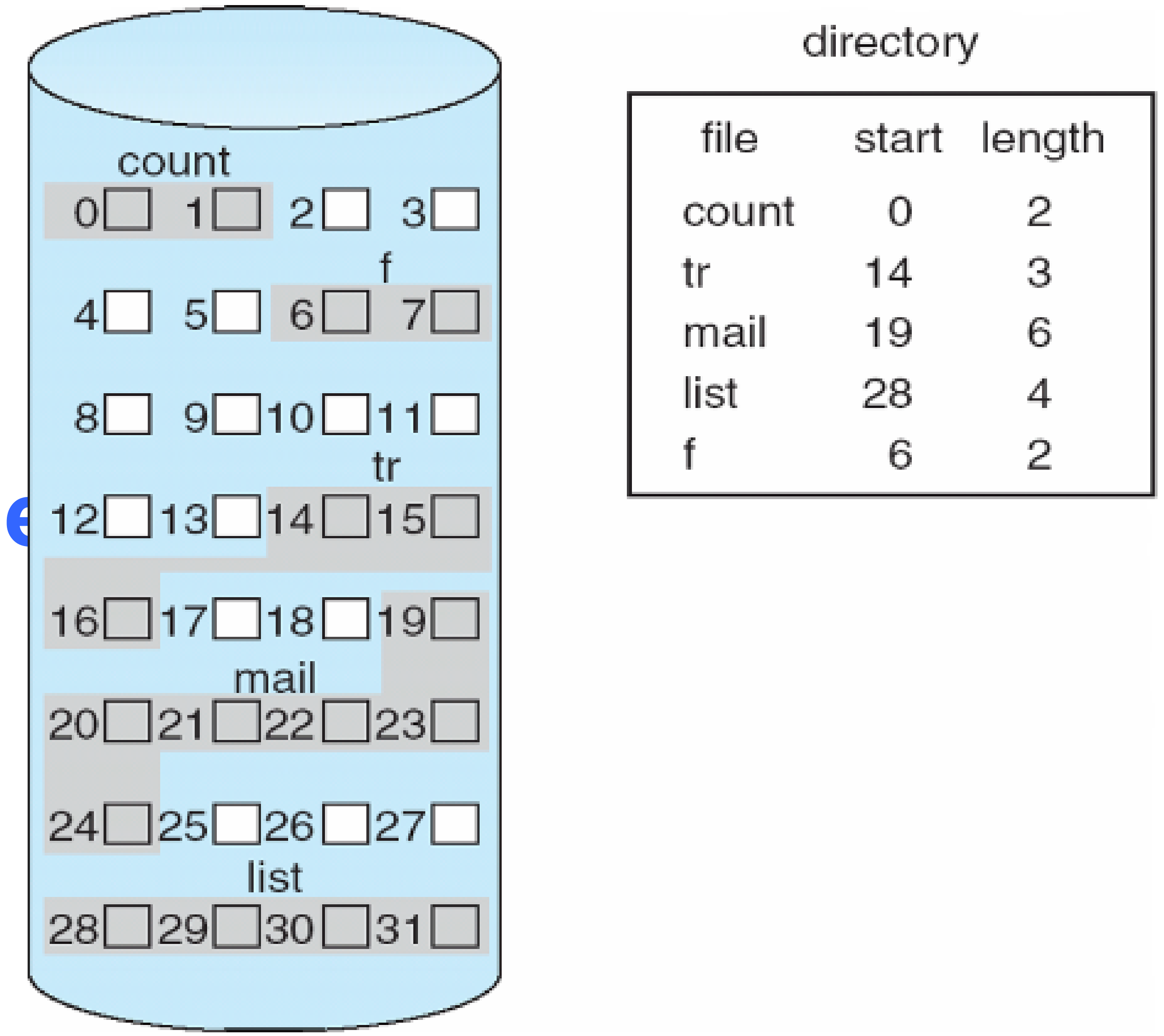
- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods

- 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

Contiguous Allocation

- Problems include
  - Finding space for file,
  - Knowing file size,
  - External fragmentation,
  - Need for **compaction off-line (downtime**
  - **On-line**

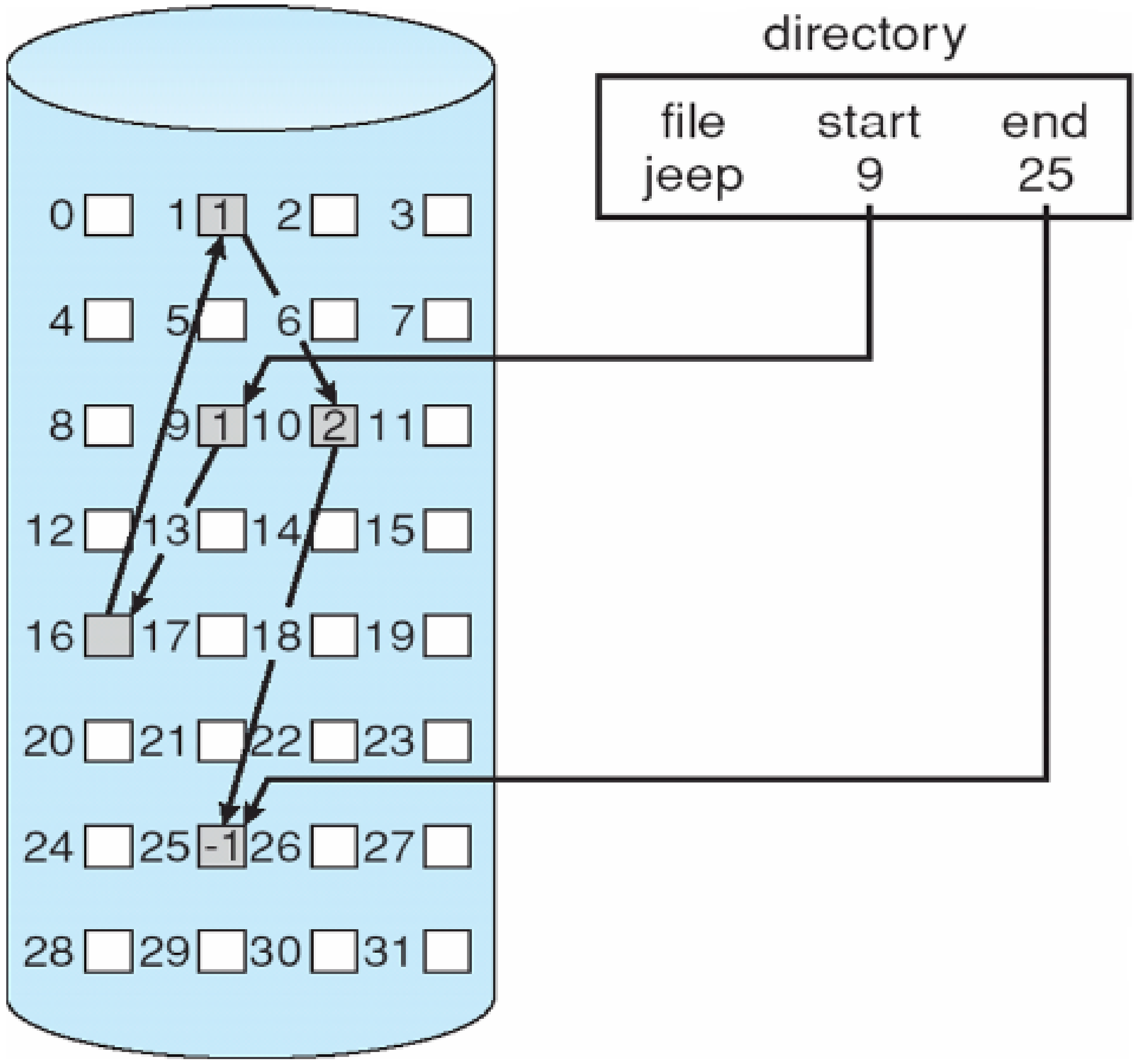


Extent Based Systems

- 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
- Internal (extents are too large) and external fragmentations (varying size extents are allocated and deallocated) can still be a problem

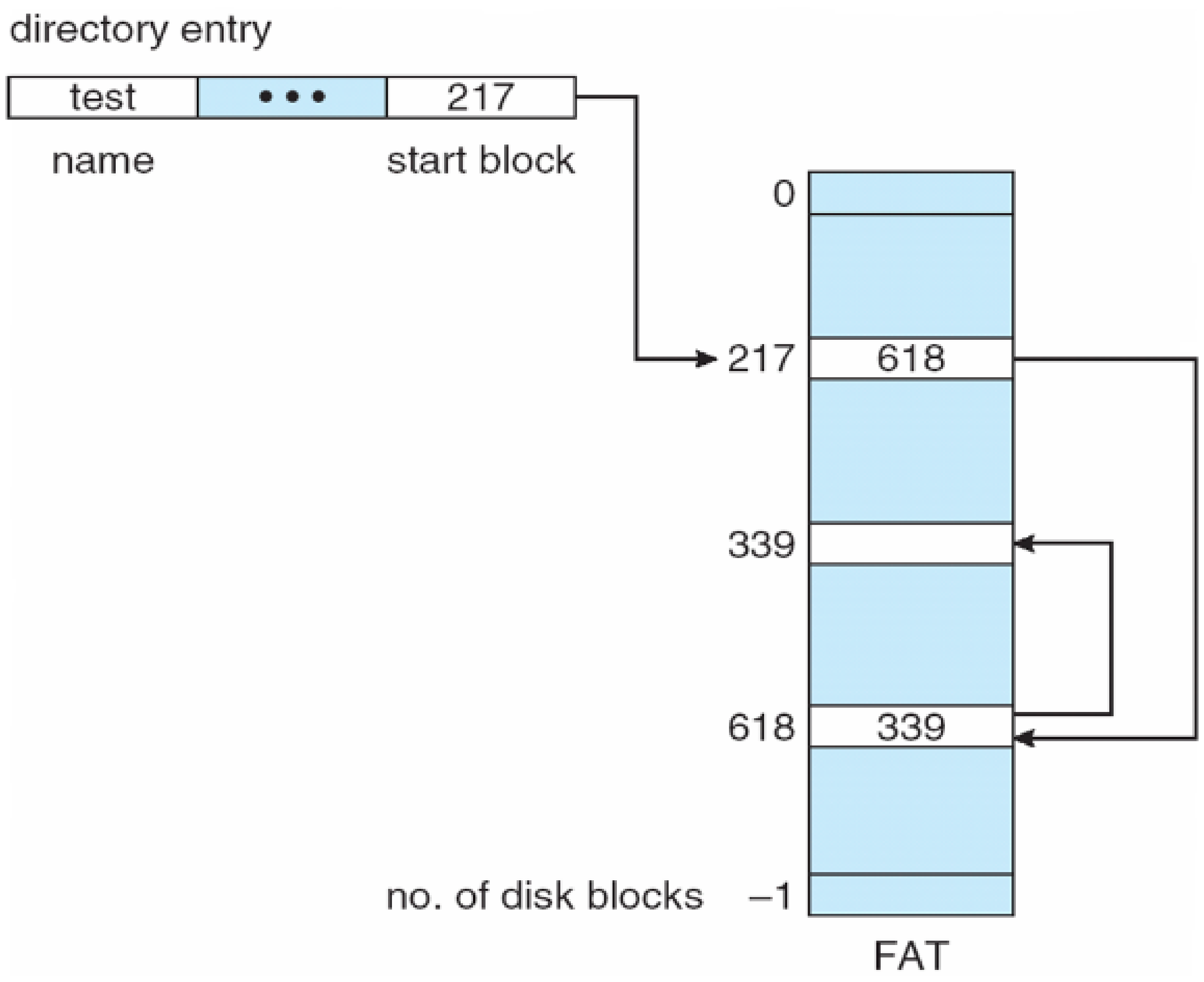
Linked Allocation

- **Linked allocation** – each file a linked list of blocks
  - File ends at NULL pointer
  - Each block contains pointer to next block
  - No compaction, and 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
  - **Effective only for sequential access of**



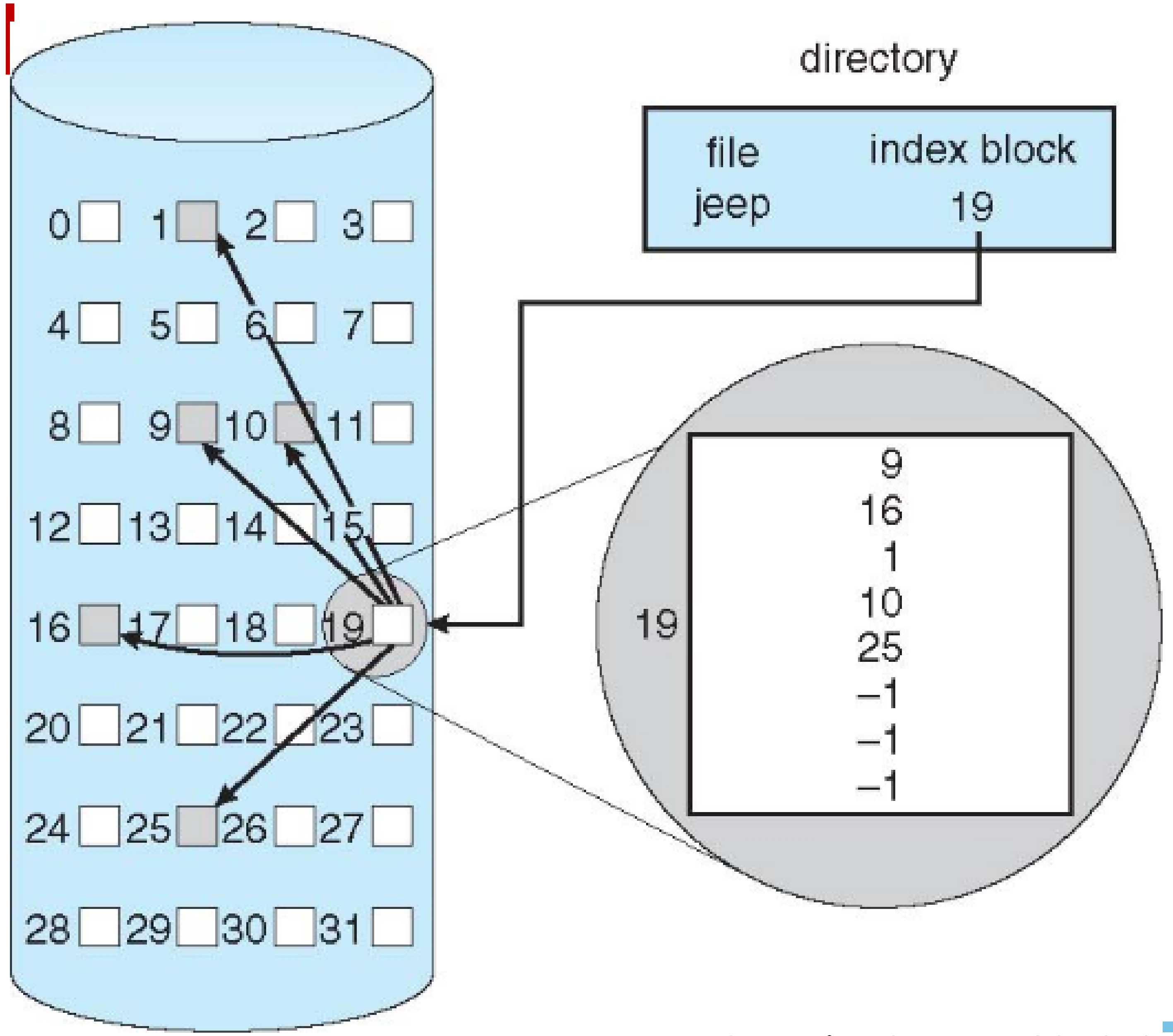
Indexed Allocation: File Allocation Table

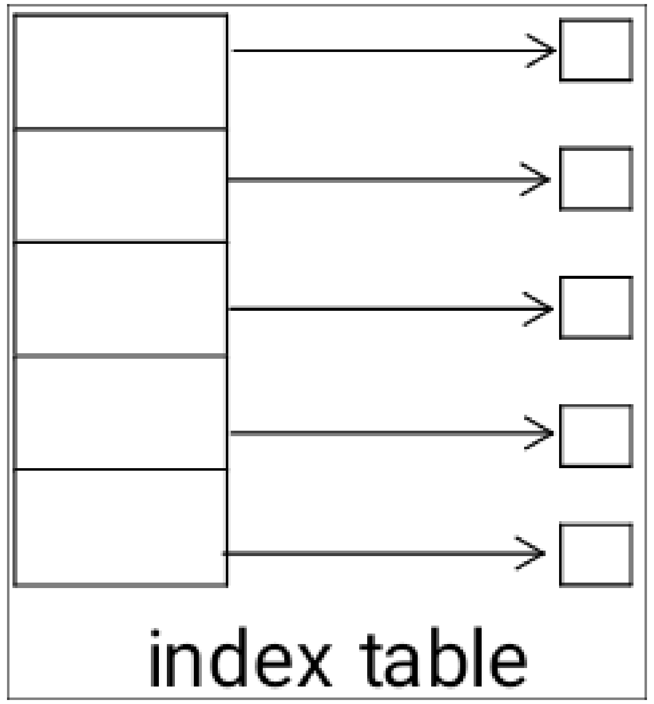
- A section of disk at the beginning of each volume is set aside to contain the table.
- Table has one entry for each disk block and is indexed by block number.
- The directory entry contains the block number of the first block in the file.



Indexed Allocation

- Each file has its own index block of pointers to its data blocks



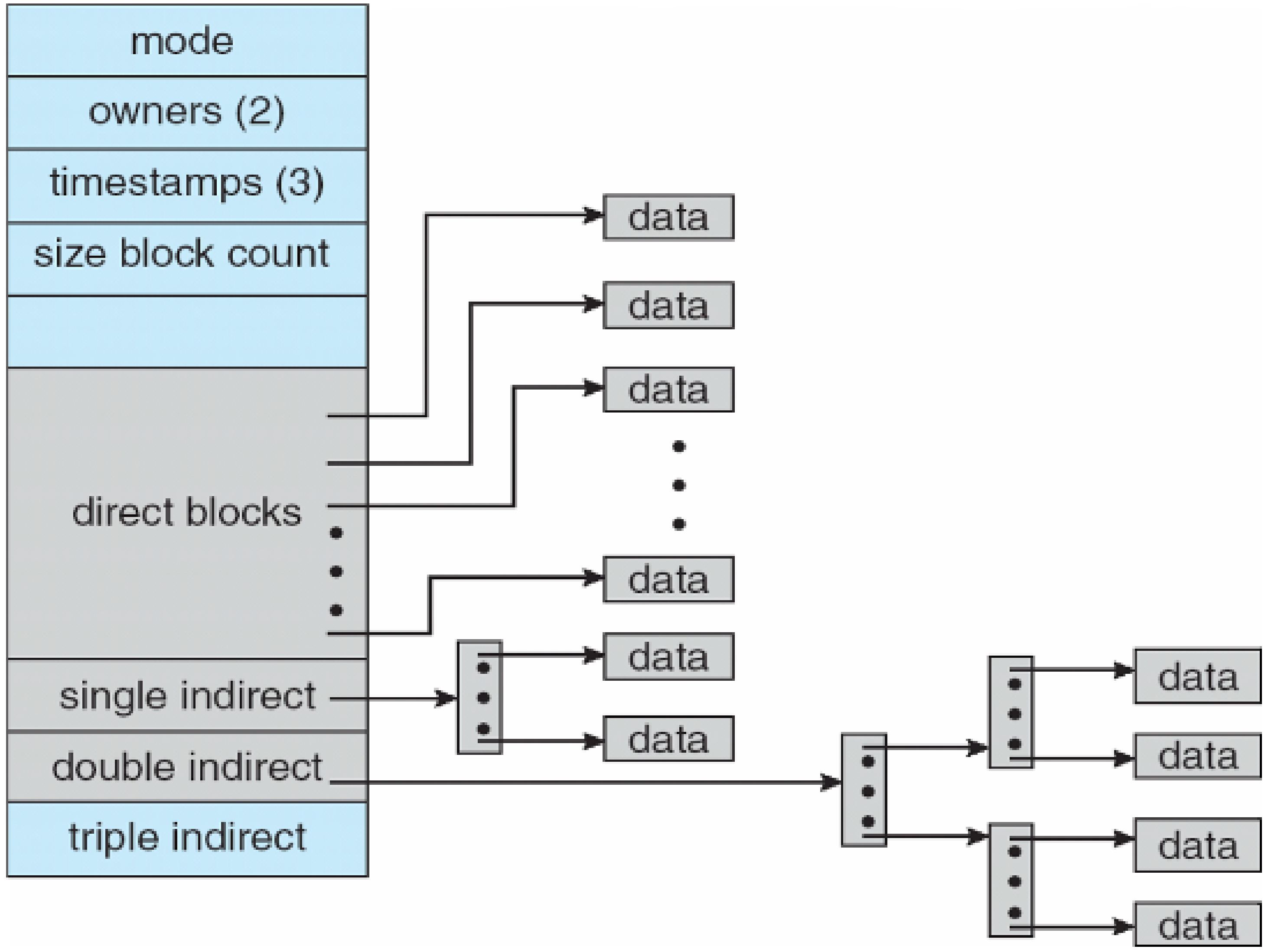


Indexed Allocation

- Minimum one index block is required for a file
  - Wastage of space for a small file
  - For a large file, several index blocked can be linked together
- Multilevel Index
  - First level index block point to a set of second level index blocks which in turn point to the file blocks
  - We can generalize it to any level for very large files
- UNIX-based File systems
  - Uses combined approach

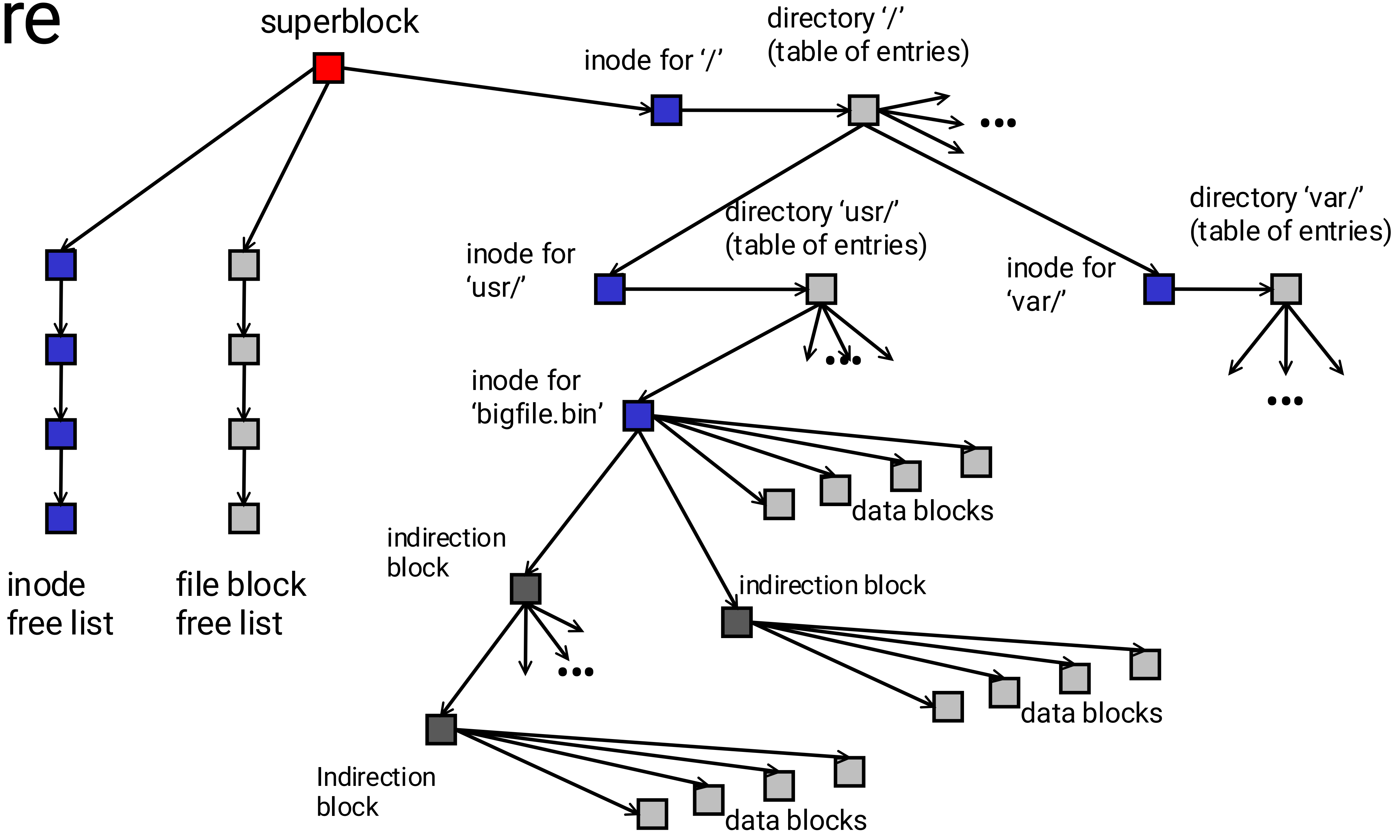
UNIX File Systems

- 4K bytes per block, 32-bit addresses



Example: UNIX File System

- The file system is just a huge data structure

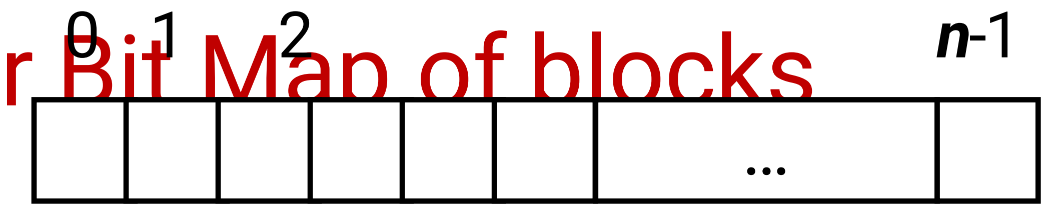


Allocation Methods Summary

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Some systems support direct-access files by using contiguous allocation and sequential access files by using linked allocation
  - 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

Free Space Management

- File system maintains **free-space list** to track available blocks/clusters.
- Bit Vector or Bit Map of 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

Bit Vector Performance

- Bit vectors are inefficient unless the entire vector is kept in main memory (written to disk occasionally)
- For large disks too much space required
- Example:

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

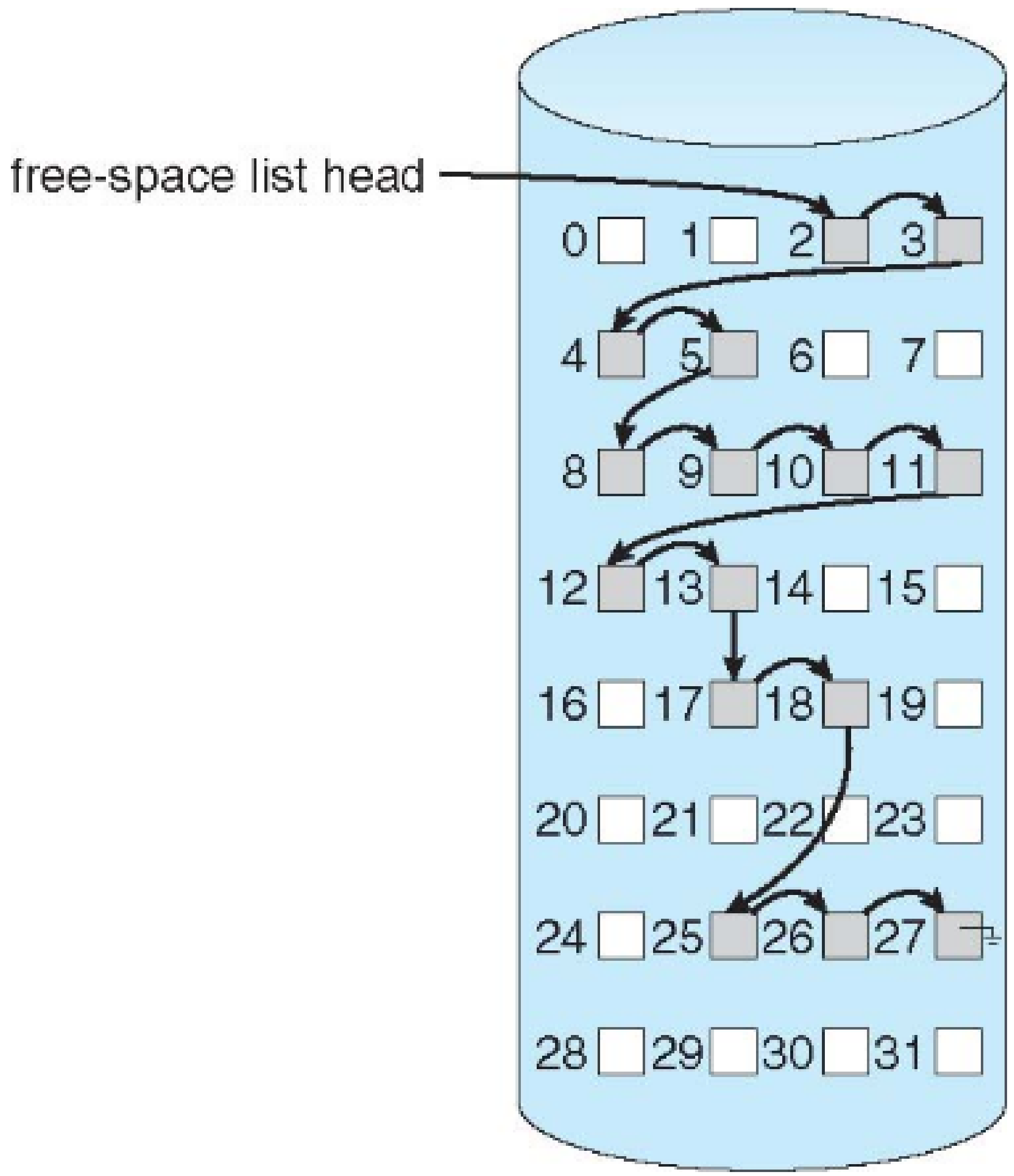
  disk size = 2^{40} bytes (1 terabyte)

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

  if clusters of 4 blocks -> 8MB of memory

Linked List

- Keeping a pointer to the first free block on the disk and caching in the memory.
- Not a good scheme for traversing the list but do we need to traverse it?
- FAT incorporates free blocks accounting in the allocation data structure



Other Methods For Free Space Management

- 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.
  - The address of a large number of free blocks can be can now be found quickly.
- 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

Efficiency and Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms
  - Types of data kept in file’s directory entry
  - Pre-allocation or as-needed allocation of metadata structures
  - Fixed-size or varying-size data structures

Recovery

- A system crash can cause inconsistencies among on-disk file-system data structures.
- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
  - A file can be reconstructed from the data blocks in linked allocation but a loss of directory entry on an indexed allocation system can be irrepairable.
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

Log Structured File System

- **Log structured** (or **journaling**) file systems record each metadata update to the file system as a **transaction**
- All transactions are written to a log
  – A transaction is considered committed once it is written to the log (sequentially)
  – Sometimes to a separate device or section of disk
  – However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system structures
  – When the file system structures are modified, the transaction is removed from the log

Log Structured File System cont.

- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata
- If a transaction was not committed (or aborted) before system crashed
  - Any changes from such a transaction that were applied to the file system must be undone
- Logging of disk metadata updates has performance benefit as well. How???

File Sharing in UNIX

- Each user has a “**channel table**” (or “per-user open file table”)
- Each entry in the channel table is a pointer to an entry in the system-wide “**open file table**”
- Each entry in the **open file table** contains a file offset (file pointer) and a pointer to an entry in the “memory-resident i-node table”
- If a process opens an already-open file, a new **open file table** entry is created (with a new file offset), pointing to the same entry in the memory-resident i-node table
- If a process forks, the child gets a copy of the channel table (and thus the same file offset)

File Sharing