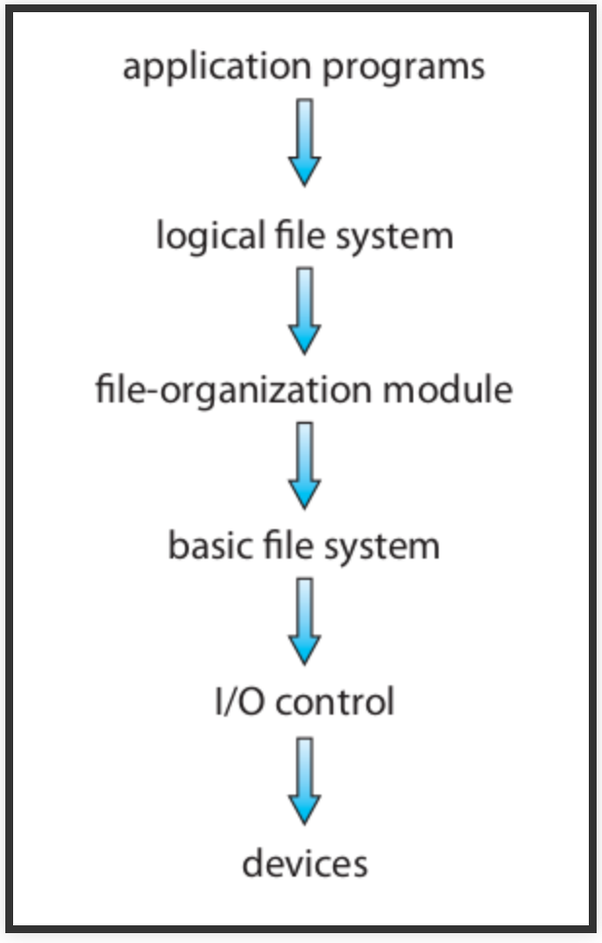
CHAPTER 13, 14 AND 15 - FILE-SYSTEM INTERFACE, IMPLEMENTATION AND INTERNALS

OBJECTIVES

- Discuss file system features (files and access methods, directories, protection)
- Discuss implementation features
  - File control block (inode)
  - Allocation methods
  - Directory implementation
  - Free space management
  - Recovery, efficience and performance
- File system types (remote file systems, NFS, WAFL)

INTRO

LAYERED FILE SYSTEM



DEVICES

- Transfer are done in blocks
  - To improve efficiency
  - Blocks have sectors (one or more)
    - 32 bytes to 4096 bytes. Usualy 512

I/O CONTROL

- Device drivers manage I/O devices at the I/O control layer
- Device drivers
- Interrupt handlers

Think of **translators**

BASIC FILE SYSTEM

- Issues generic commands
- 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

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

FILE SYSTEM LAYERS

- 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

FILE SYSTEMS

- 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

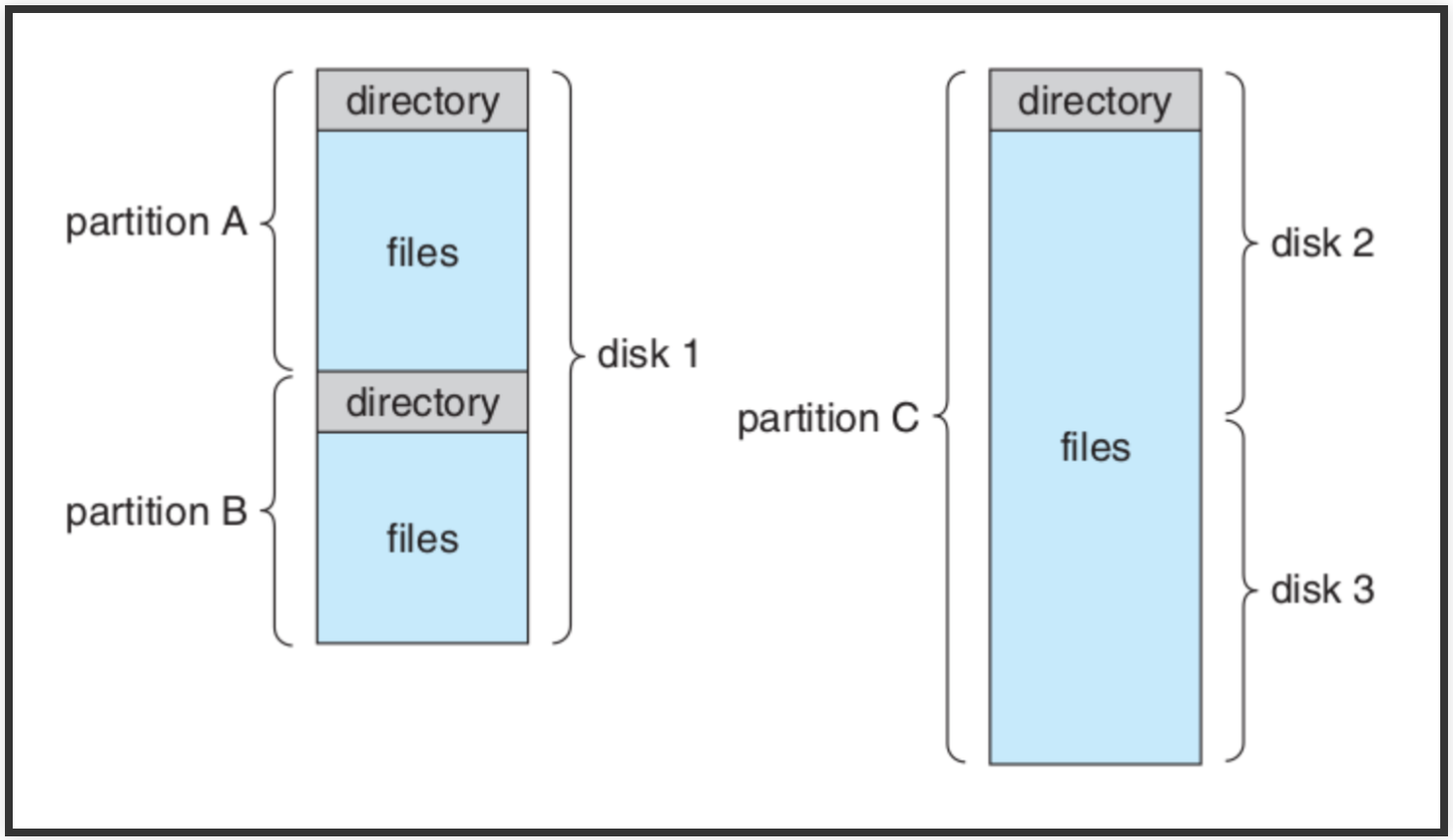
DISK STRUCTURE

- Disk can be subdivided into partitions
- Disks or partitions can be RAID protected against failure
- Disk or partition can be used raw – without a file system, or formatted with a file system
- Partitions also known as minidisks, slices

DISK STRUCTURE

- Entity containing file system known as a volume
- Each volume containing file system also tracks that file system's info in device directory or volume table of contents
- As well as general-purpose file systems there are many special-purpose file systems, frequently all within the same operating system or computer

TYPICAL FILE-SYSTEM ORGANIZATION



TYPES OF FILE SYSTEMS

- We mostly talk of general-purpose file systems
- But systems frequently have many file systems, some general- and some special- purpose

SOLARIS FILE SYSTEMS

- tmpfs – memory-based volatile FS for fast, temporary I/O
- objfs – interface into kernel memory to get kernel symbols for debugging
- ctfs – contract file system for managing daemons
- lofs – loopback file system allows one FS to be accessed in place of another
- procfs – kernel interface to process structures
- ufs, zfs – general purpose file systems

FILE-SYSTEM MOUNTING

- A file system must be mounted before it can be accessed
- A unmounted file system is mounted at a mount point

FILE SYSTEM FEATURES

- Files and access methods
- Directories
- Protection

FILE CONCEPT

- Contiguous logical address space
- Types:
  - Data → numeric - character - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider text file, source file, executable file

FILE ATTRIBUTES 1

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing

FILE ATTRIBUTES 2

- Time, date, and user identification – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

FILE OPERATIONS

- Create
- Write – at write pointer location
- Read – at read pointer location
- Reposition within file - seek
- Delete + Truncate
- Open(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory
- Close (F_i) – move the content of entry F_i in memory to directory structure on disk

OPEN FILES

- **Open-file table**: tracks open files
- **File pointer**: pointer to last read/write location, per process with file open
- **File-open count**: counter of number of times a file is open
- Disk location of the file: cache of data access information
- Access rights: per-process access mode information

OPEN FILE LOCKING

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - Shared lock similar to reader lock – several processes can acquire concurrently
  - Exclusive lock similar to writer lock
- Mediates access to a file

OPEN FILE LOCKING

- Mandatory or advisory:
  - Mandatory – access is denied depending on locks held and requested
  - Advisory – processes can find status of locks and decide what to do

FILE TYPES – NAME, EXTENSION

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

FILE STRUCTURE

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file

FILE STRUCTURE

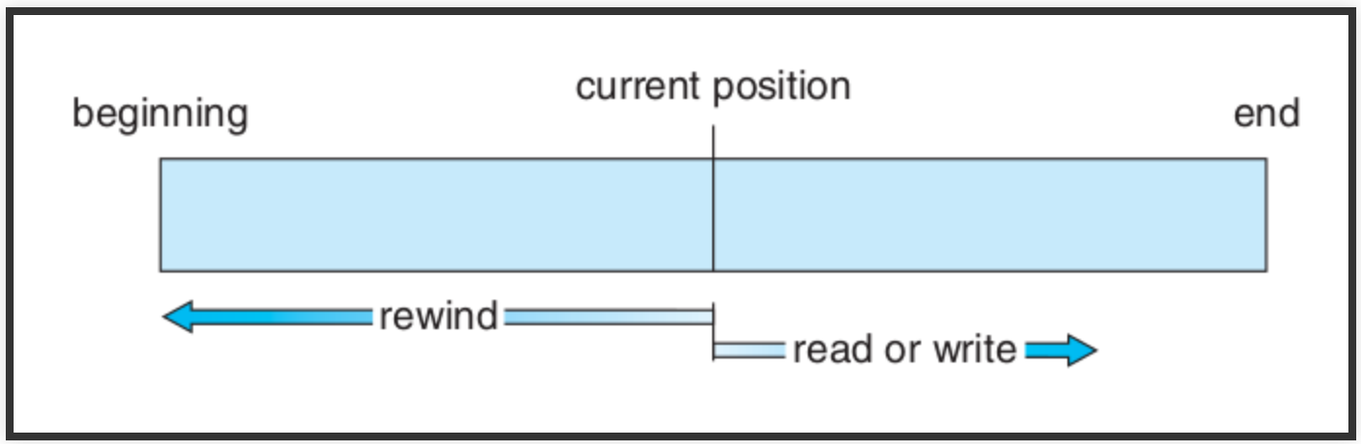
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

ACCESS METHODS

ACCESS METHODS

- Sequential Access
  - `read next`
  - `write next`
  - `reset`
  - `no read after last write`
  - `(rewrite)`

SEQUENTIAL-ACCESS FILE



ACCESS METHODS

- 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*
- Relative block numbers allow OS to decide where file should be placed

SIMULATION OF SEQUENTIAL ACCESS ON DIRECT-ACCESS FILE

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;
write_next	write cp; cp = cp + 1;

OTHER ACCESS METHODS

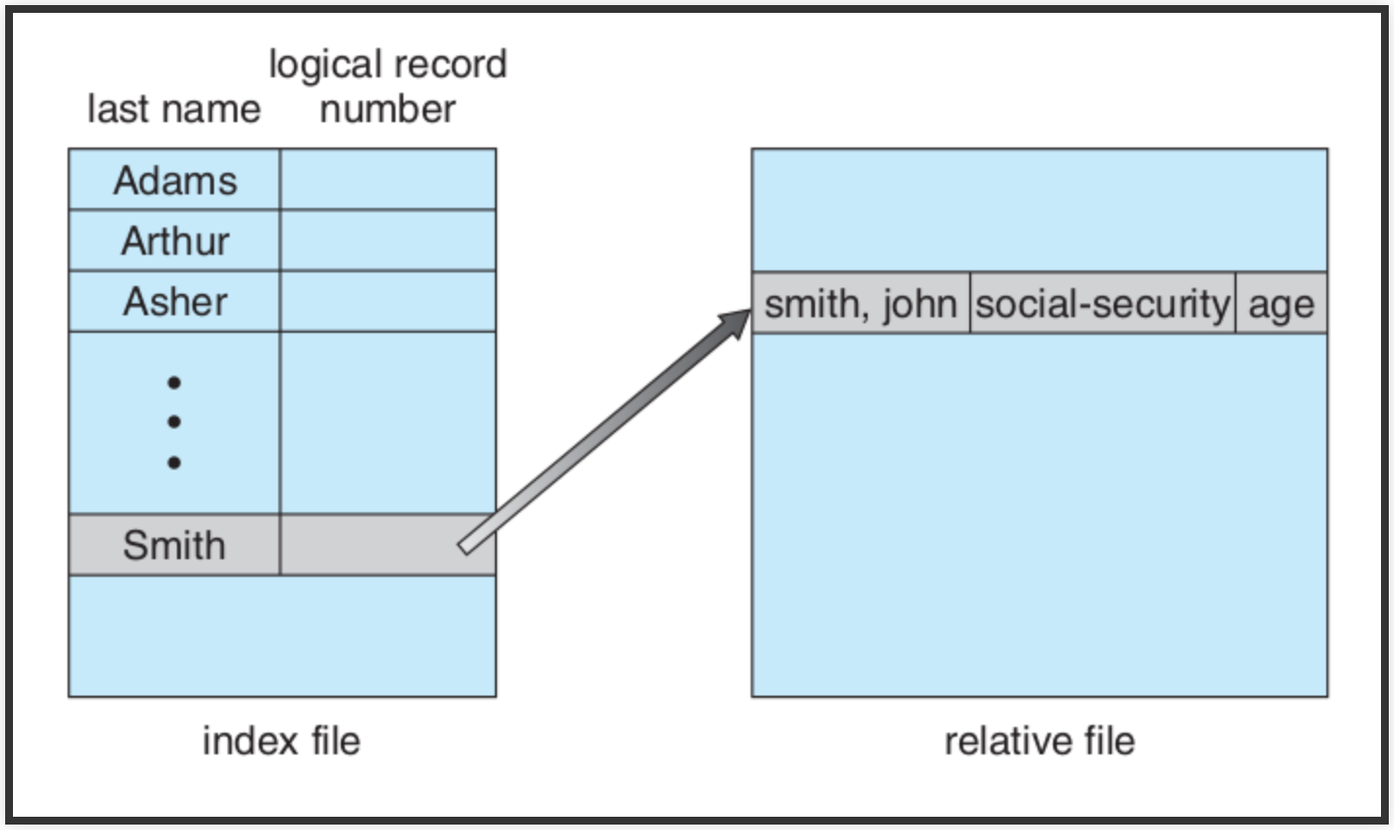
- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)

ISAM

- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS

VMS operating system provides index and relative files as another example (see next slide)

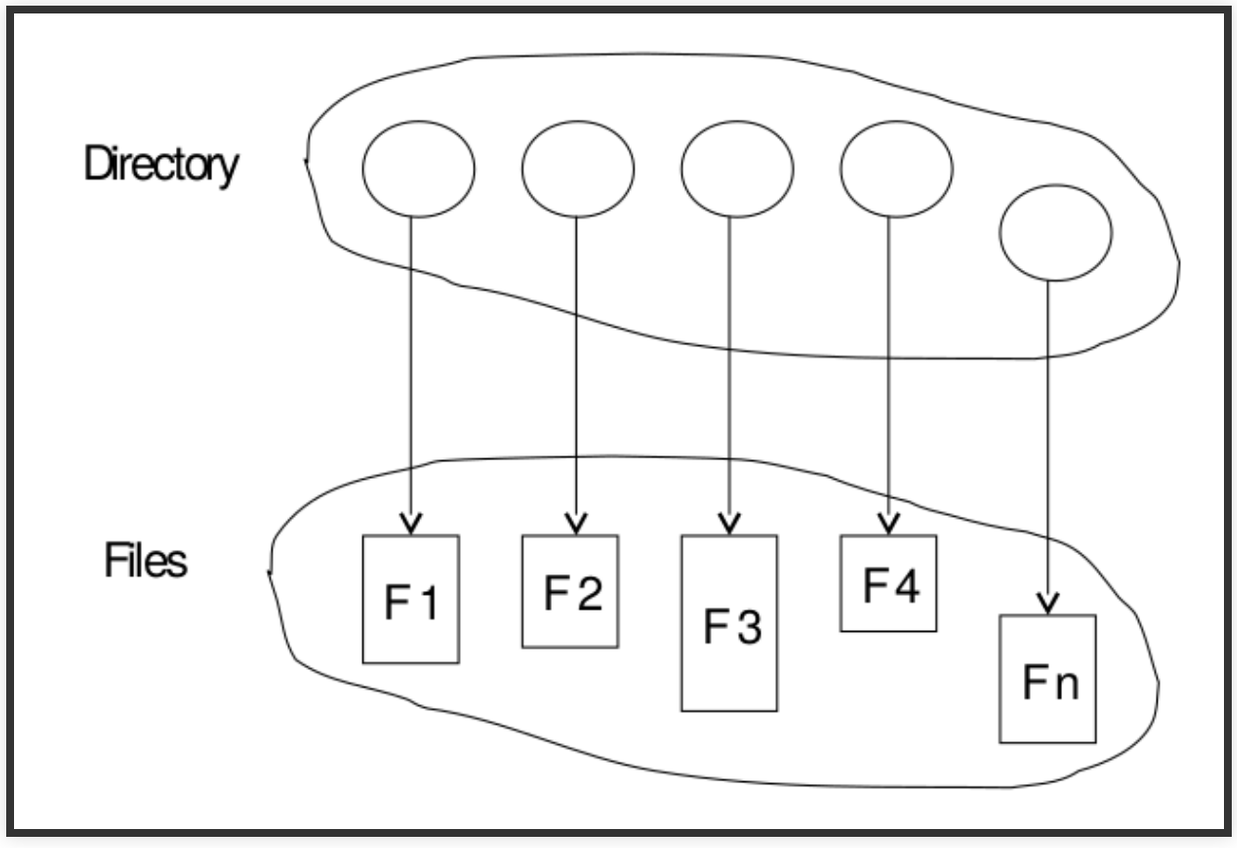
EXAMPLE OF INDEX AND RELATIVE FILES



DIRECTORIES

DIRECTORY STRUCTURE

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

OPERATIONS PERFORMED ON DIRECTORY

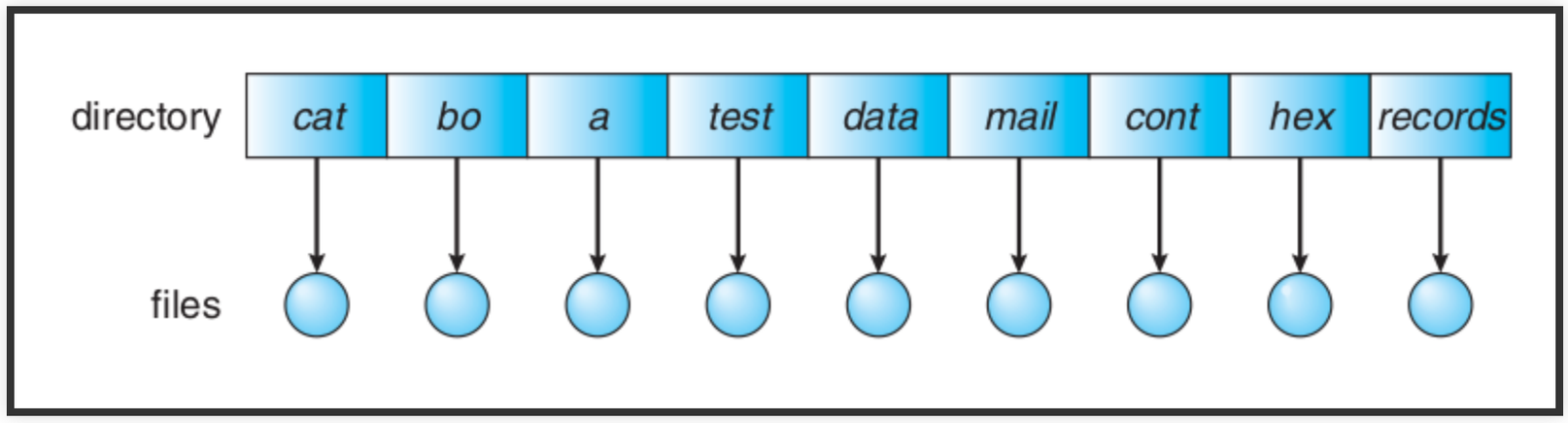
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

ORGANIZE THE DIRECTORY (LOGICALLY) TO OBTAIN

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

SINGLE-LEVEL DIRECTORY

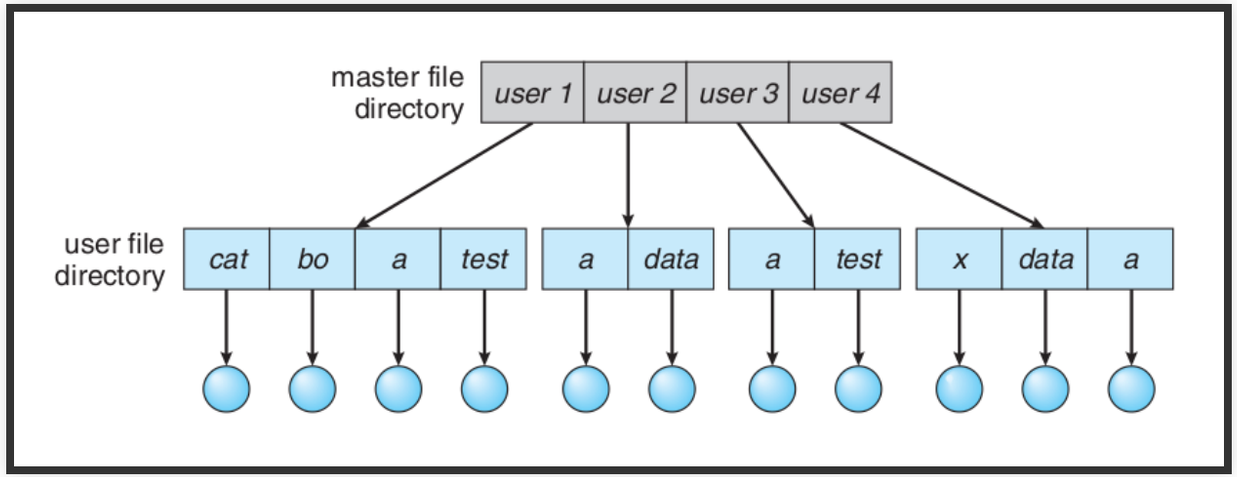
- A single directory for all users



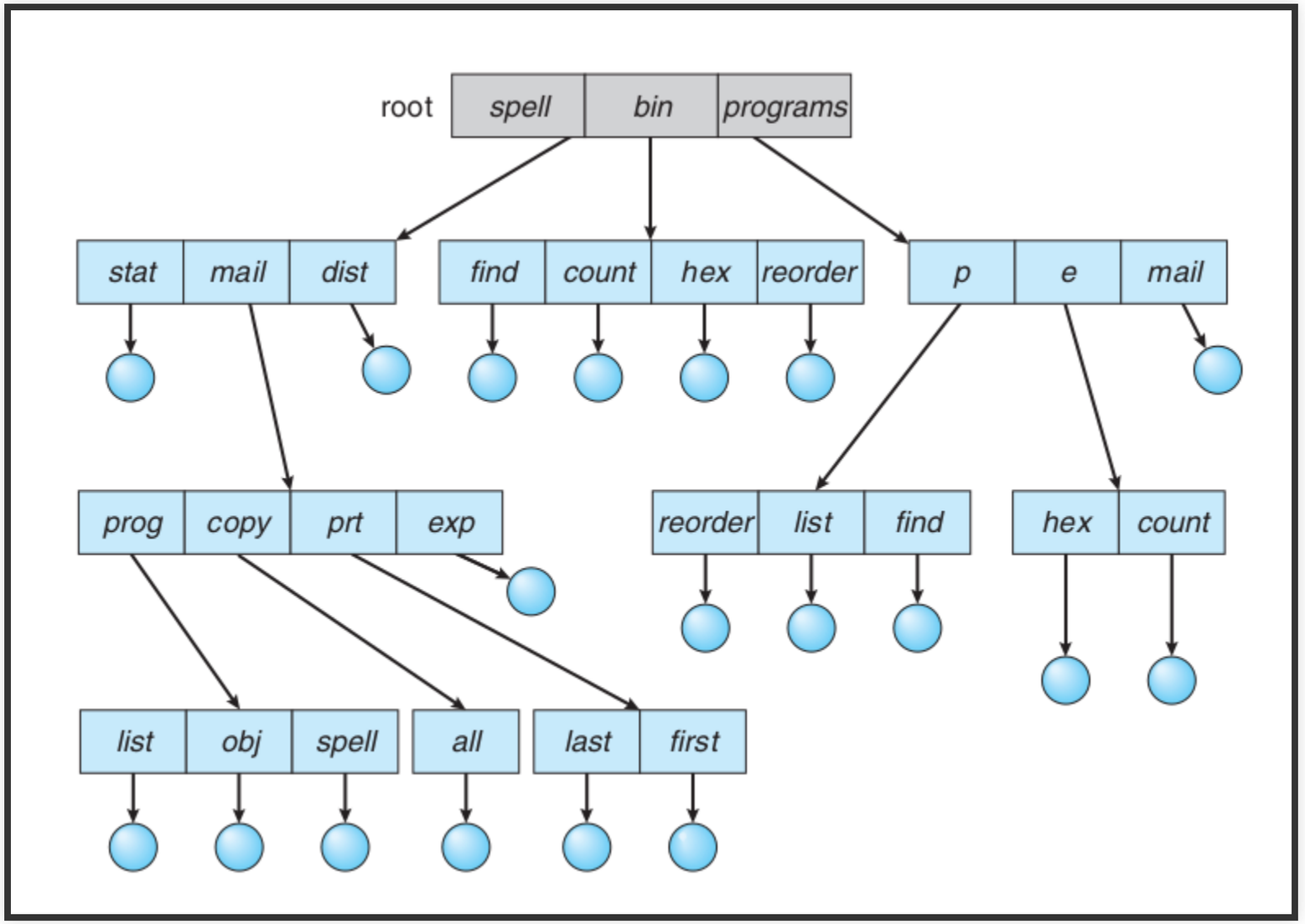
- Naming problem
- Grouping problem

TWO-LEVEL DIRECTORY

- Separate directory for each user



TREE-STRUCTURED DIRECTORIES

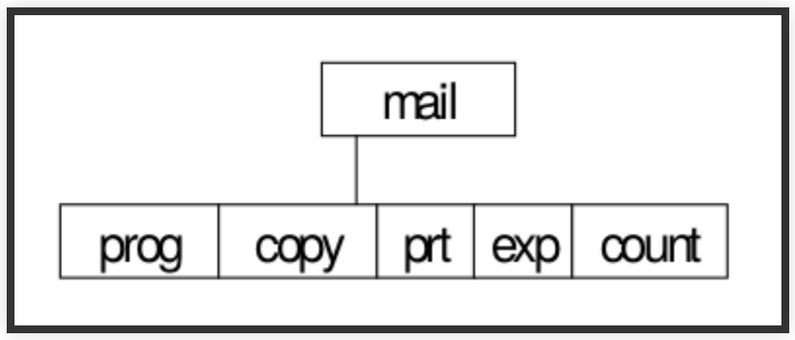


TREE-STRUCTURED DIRECTORIES

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - `cd /spell/mail/prog`
  - `type list`

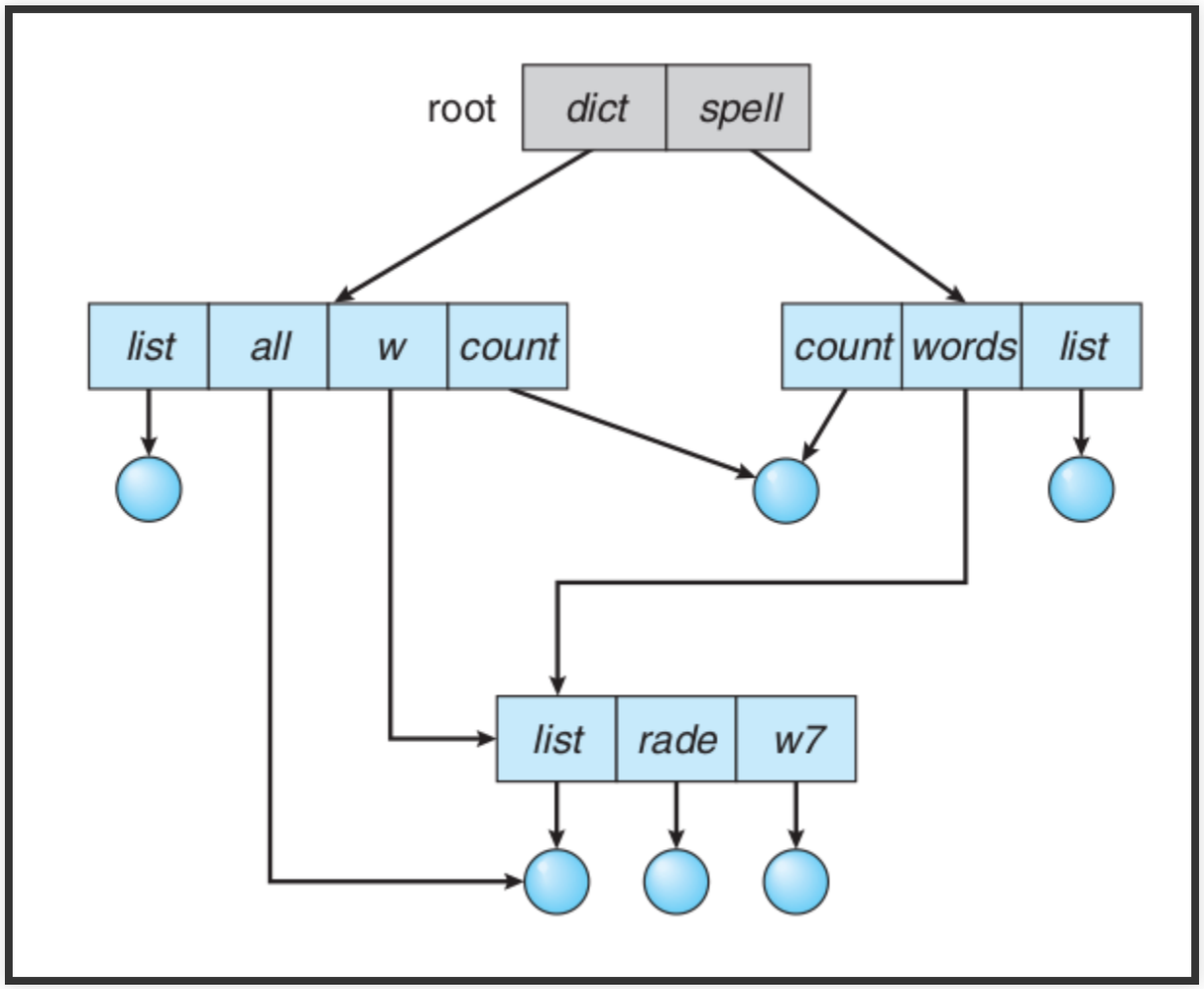
TREE-STRUCTURED DIRECTORIES

- Absolute or relative path name
- Creating a new file is done in current directory
- Delete a file: `rm <file-name>`
- Creating a new subdirectory is done in current directory: `mkdir <dir-name>` Example: if in current directory */mail* : `mkdir count`



ACYCLIC-GRAPH DIRECTORIES

- Have shared subdirectories and files



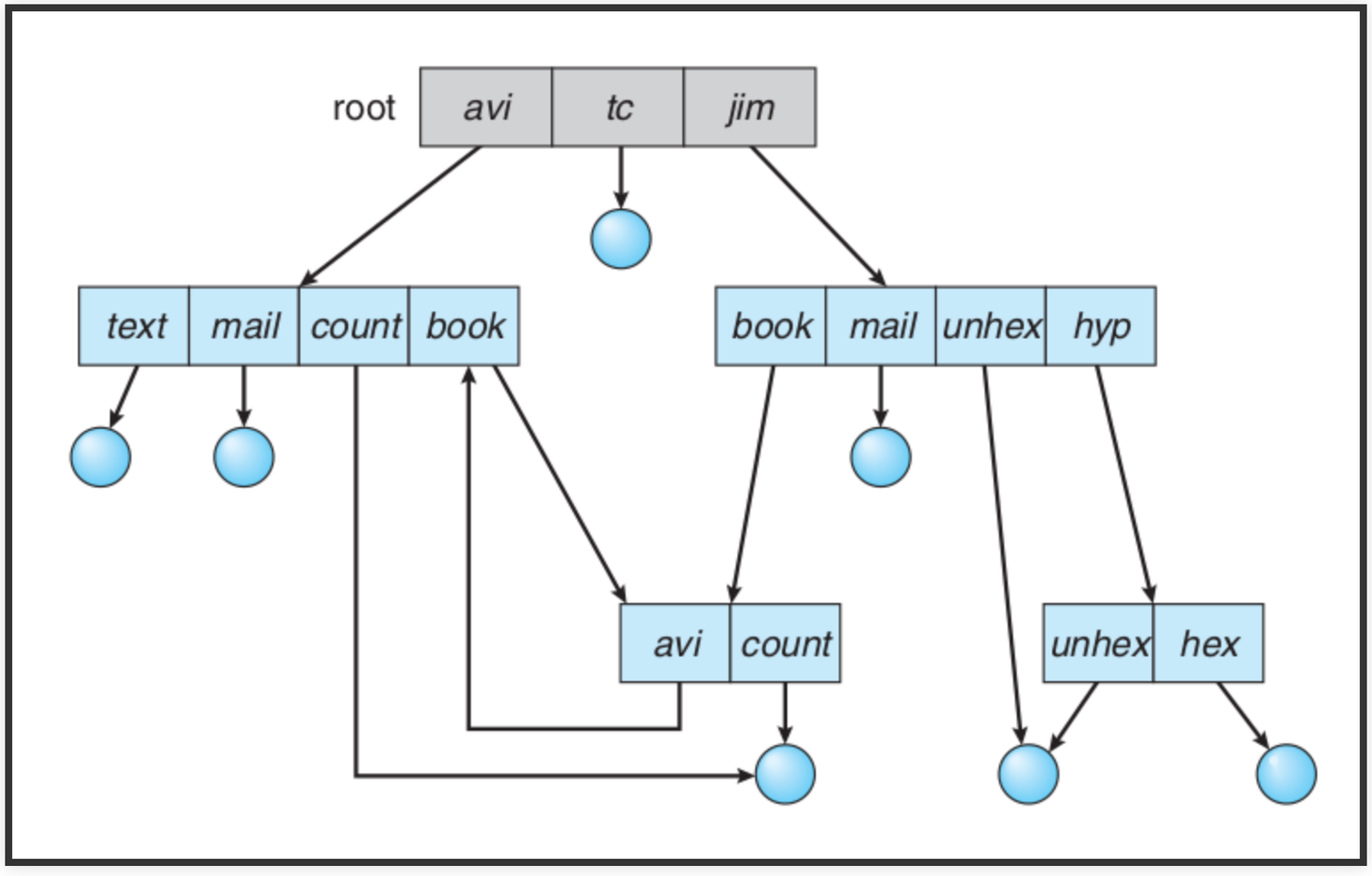
ACYCLIC-GRAPH DIRECTORIES

- Two different names (aliasing)
- If dict deletes list ⇒ dangling pointer Solutions:
  - Backpointers, so we can delete all pointers Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution

ACYCLIC-GRAPH DIRECTORIES

- New directory entry type
  - Link – another name (pointer) to an existing file
  - Resolve the link – follow pointer to locate the file

GENERAL GRAPH DIRECTORY



GENERAL GRAPH DIRECTORY

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Garbage collection
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

PROTECTION

PROTECTION

- File owner/creator should be able to control:
  - what can be done
  - by whom

PROTECTION

- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

ACCESS LISTS AND GROUPS

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

```
                     R W X
a) owner access  7 ⇒ 1 1 1
                     R W X
b) group access  6 ⇒ 1 1 0
                     R W X
c) public access 1 ⇒ 0 0 1
```

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say game) or subdirectory, define an appropriate access.

A SAMPLE UNIX 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 jwg   student      512  Aug 3 14:13    student-proj/
-rw-r--r--    1 pbg   staff       9423  Feb 24 2012    program.c
-rwxr-xr-x    1 pbg   staff      20471  Feb 24 2012    program
drwx--x--x    4 tag   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/
```

IMPLEMENTATION ISSUES

FILE-SYSTEM IMPLEMENTATION

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

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

FILE-SYSTEM STRUCTURE

- 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

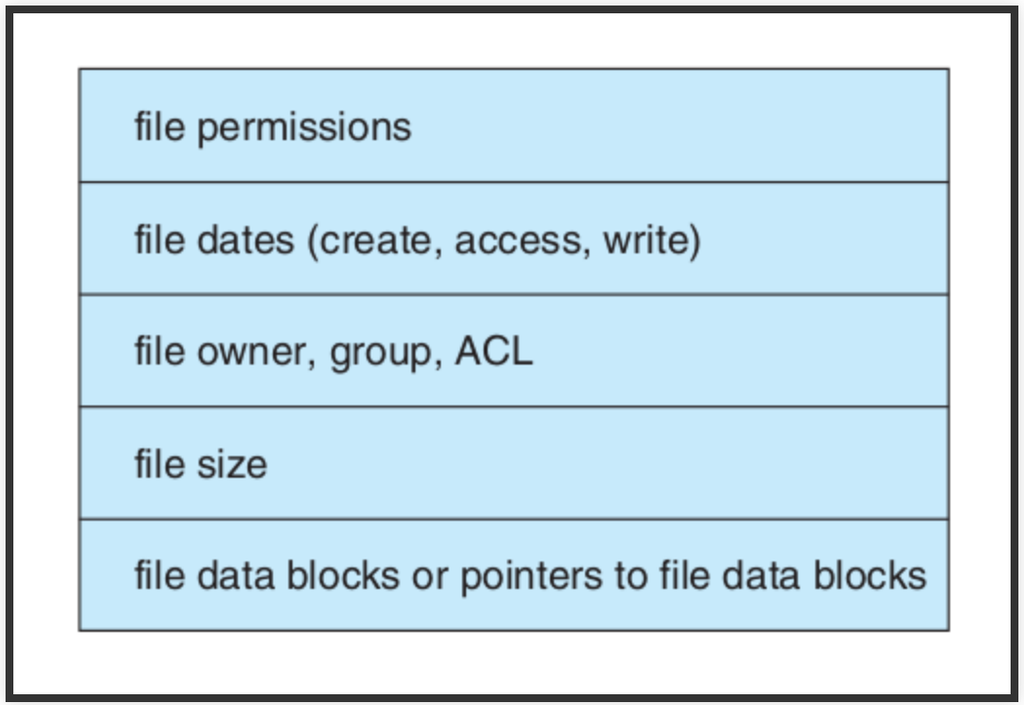
ON-DISK STRUCTURES

- **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

ON-DISK STRUCTURES

- 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

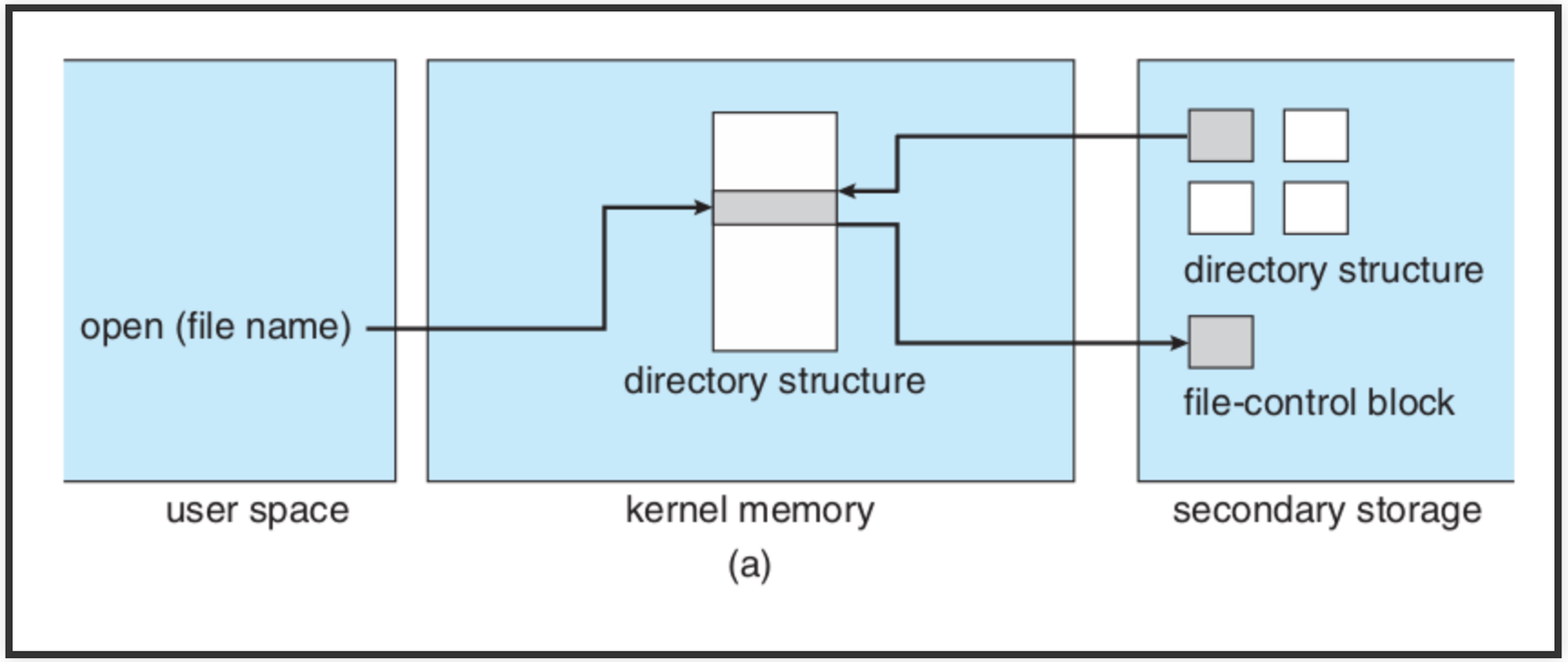
FILE CONTROL BLOCK (INODE)



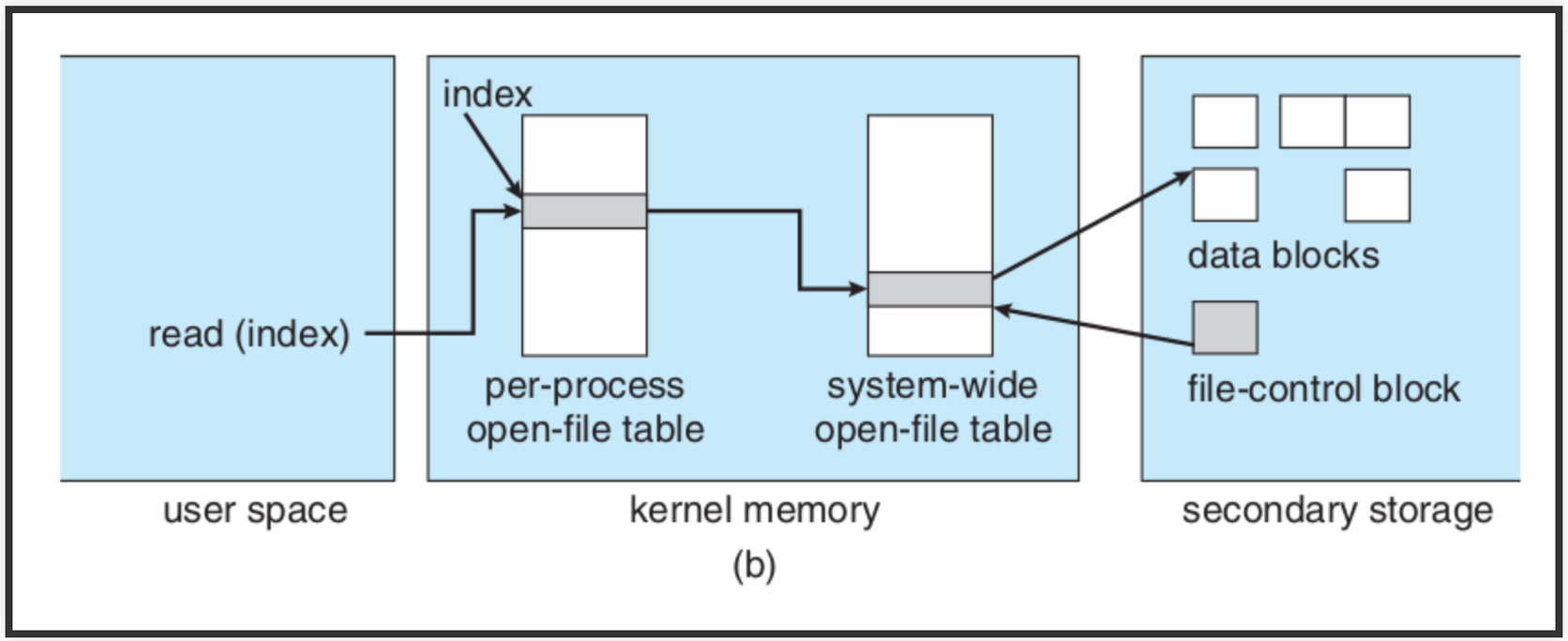
IN-MEMORY STRUCTURES

- **Mount table** storing file system mounts, mount points, file system types
- Directory-structure cache of recently accessed directories
- **System wide open-file table** with copy of each files FCB
- **Per-process open-file table** pointers to system wide table
- Buffers hold data blocks while they are read/written

IN-MEMORY FILE SYSTEM STRUCTURES



IN-MEMORY FILE SYSTEM STRUCTURES



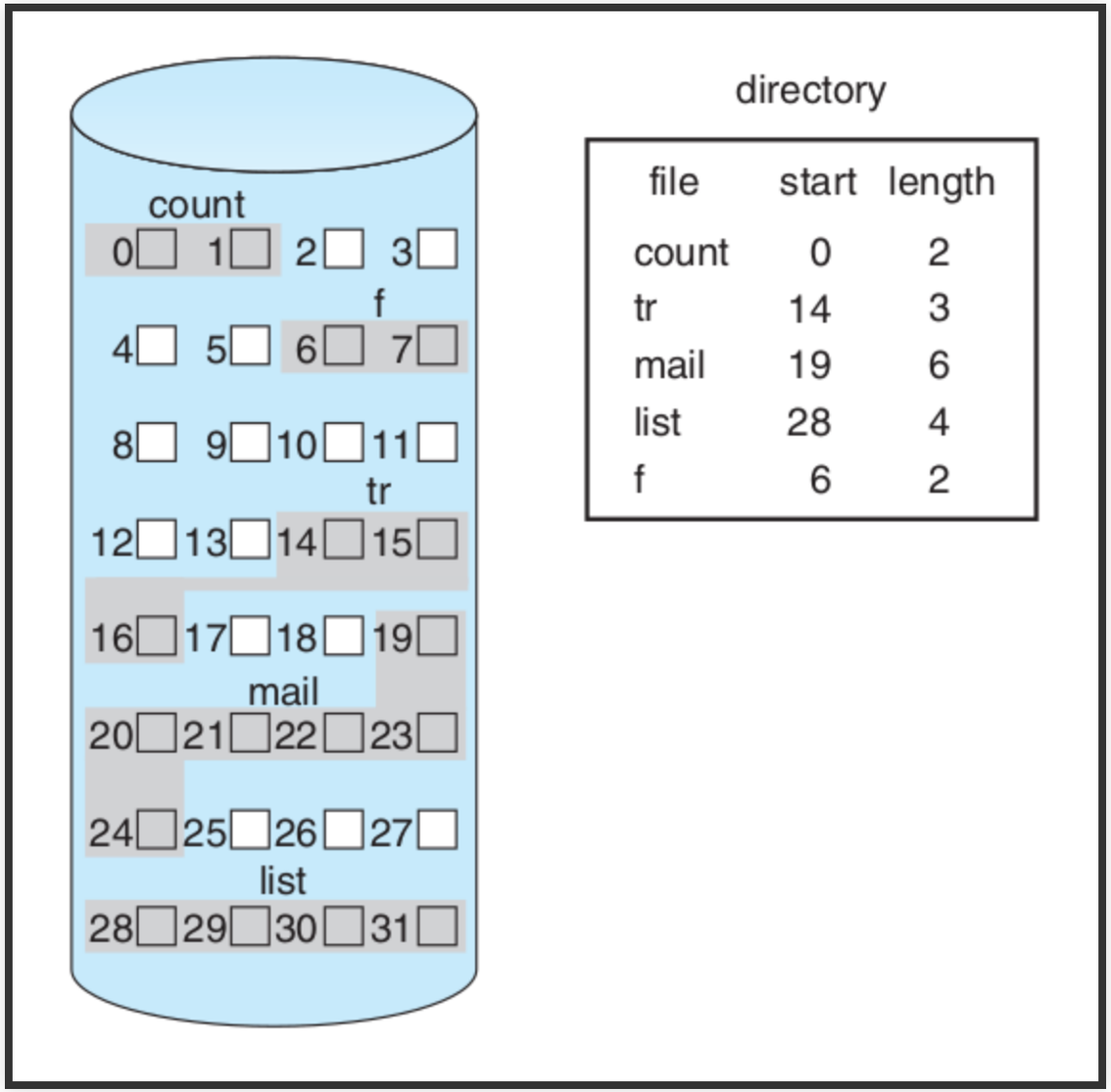
(b)

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

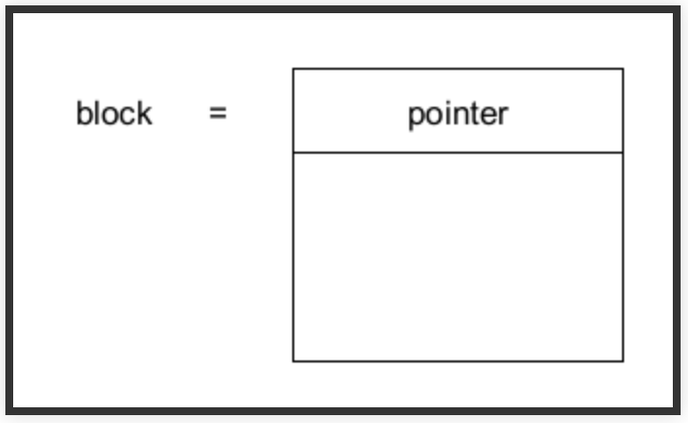


EXTENT-BASED SYSTEMS

- 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

LINKED ALLOCATION

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



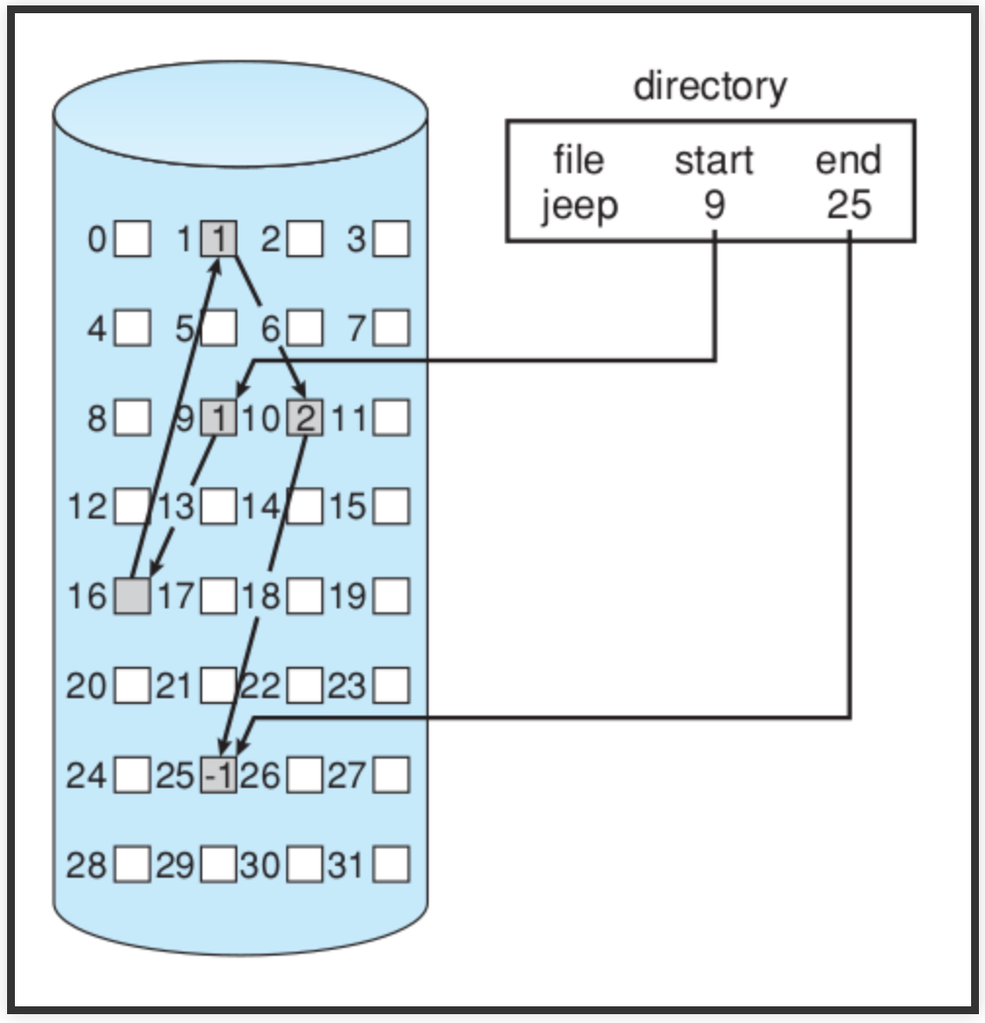
LINKED

- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system when new block needed

LINKED

- 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

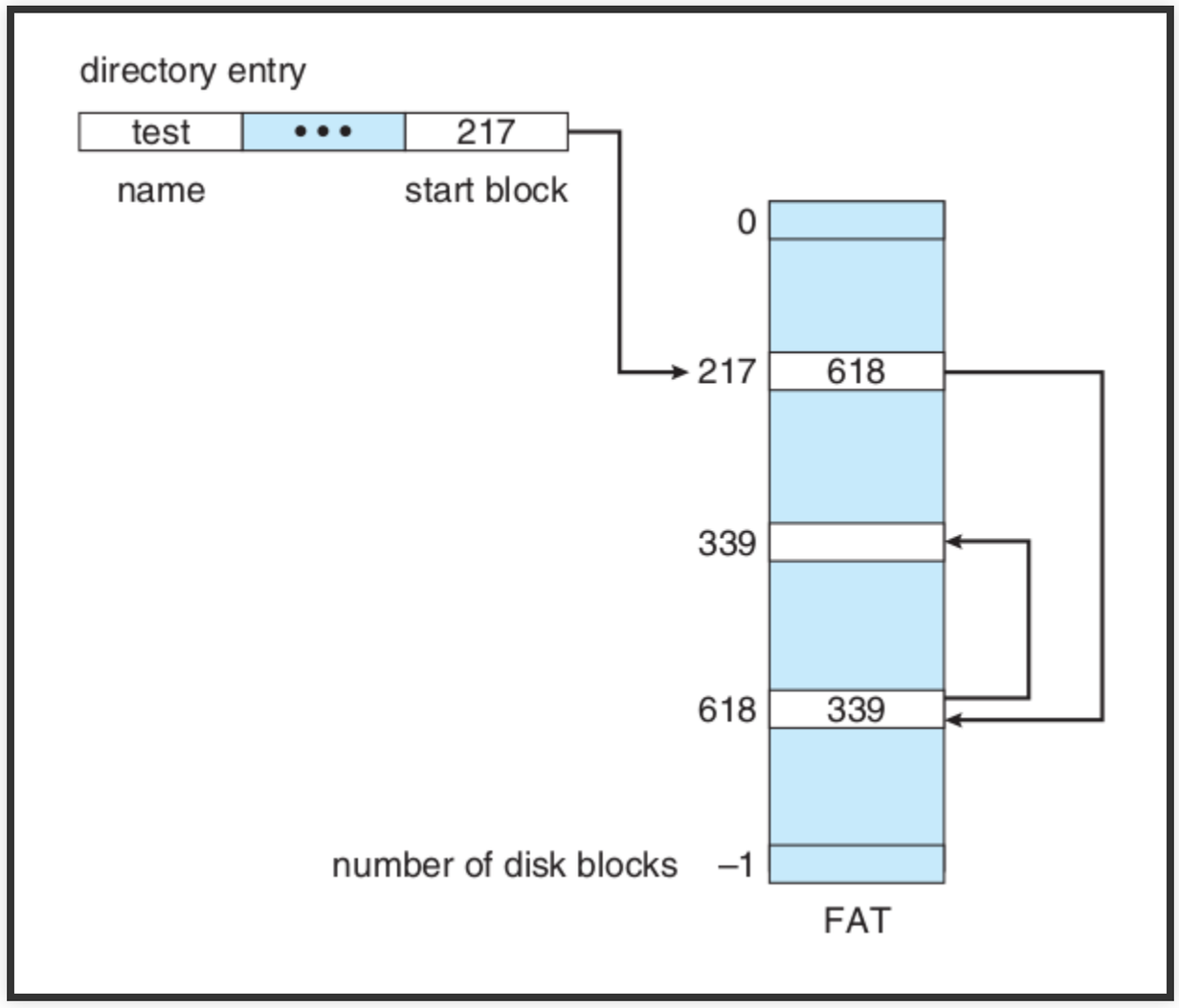
LINKED ALLOCATION



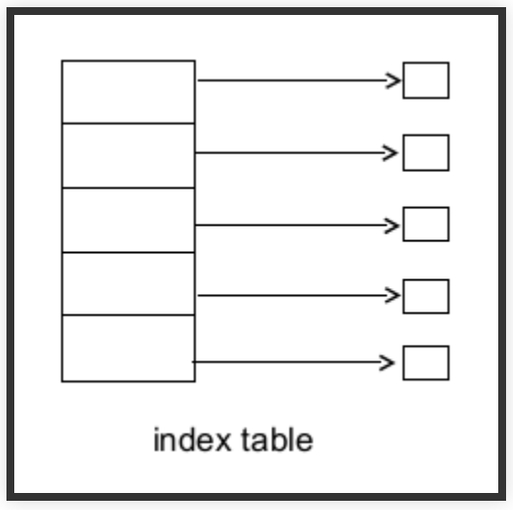
FAT

- 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

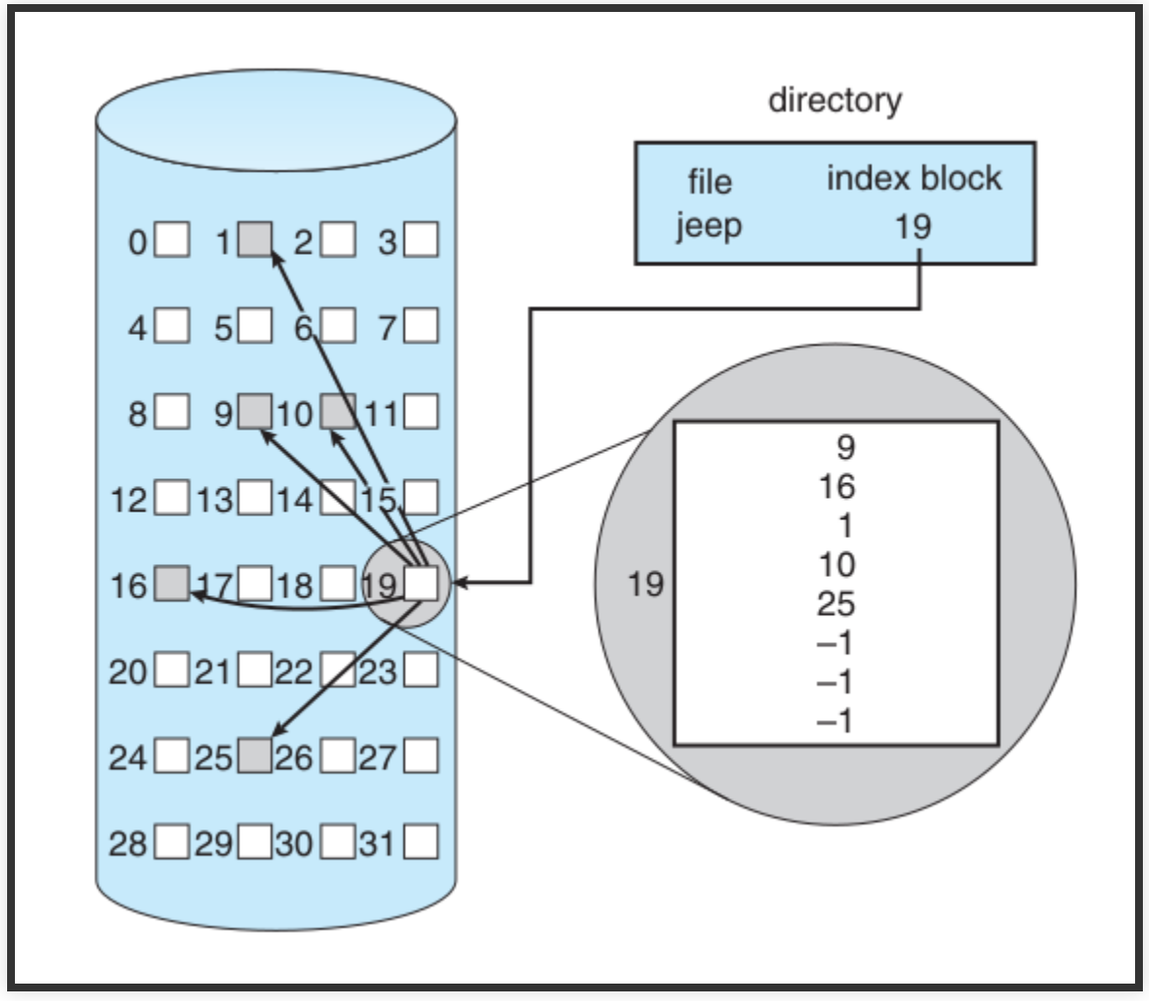
FILE-ALLOCATION TABLE



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

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

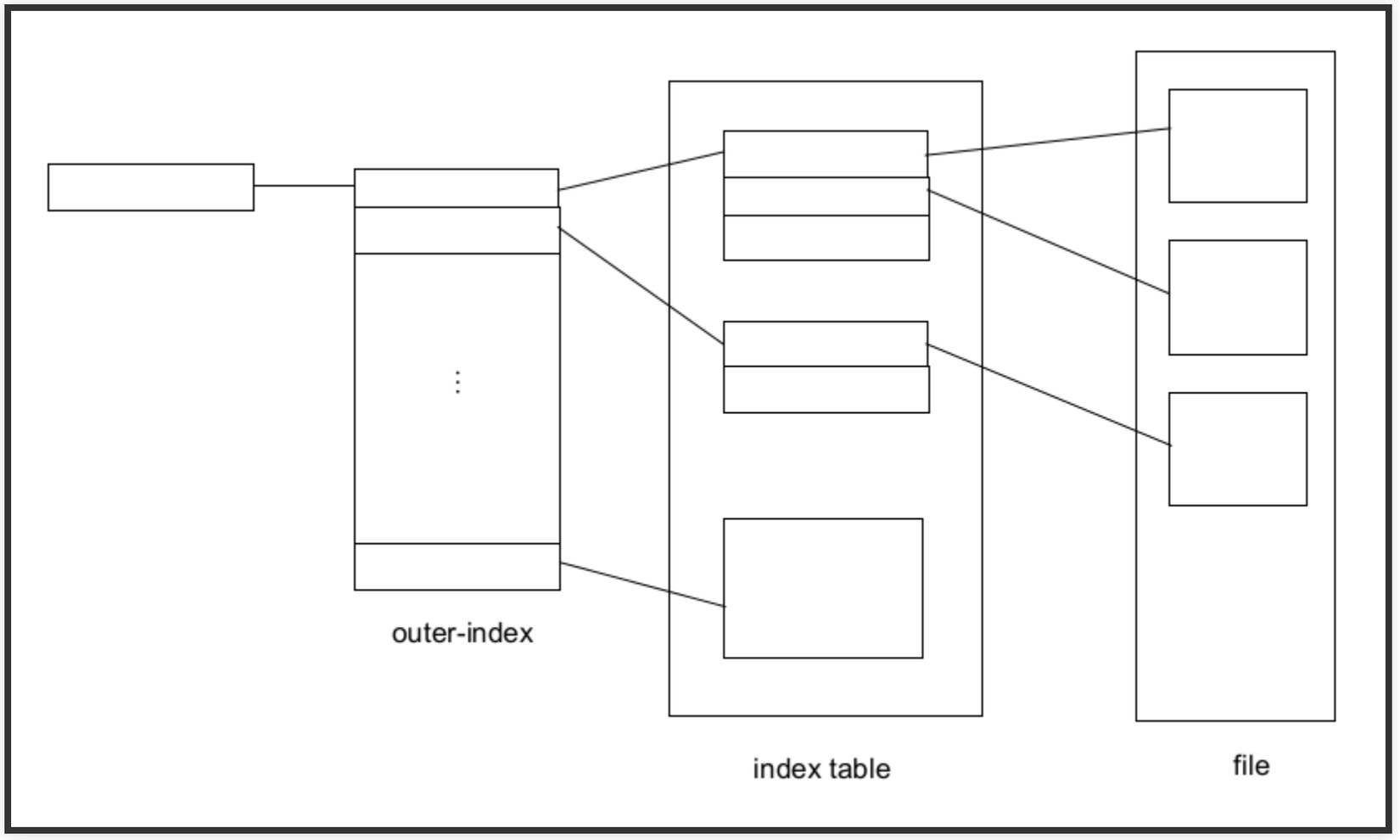
LINKED SCHEME

- Index block normally one disk block
- To allow large files - link them togeather with last pointer in block.
  - no limit on size

MULTILEVEL INDEX

- First-level index block points to second-level index blocks
- Second level index blocks points to file blocks with data
- Can be extended to more layers
- 4K blocks could store 1,024 four-byte pointers in outer index
  - 1,048,567 data blocks and file size of up to 4GB

MULTILEVEL INDEX



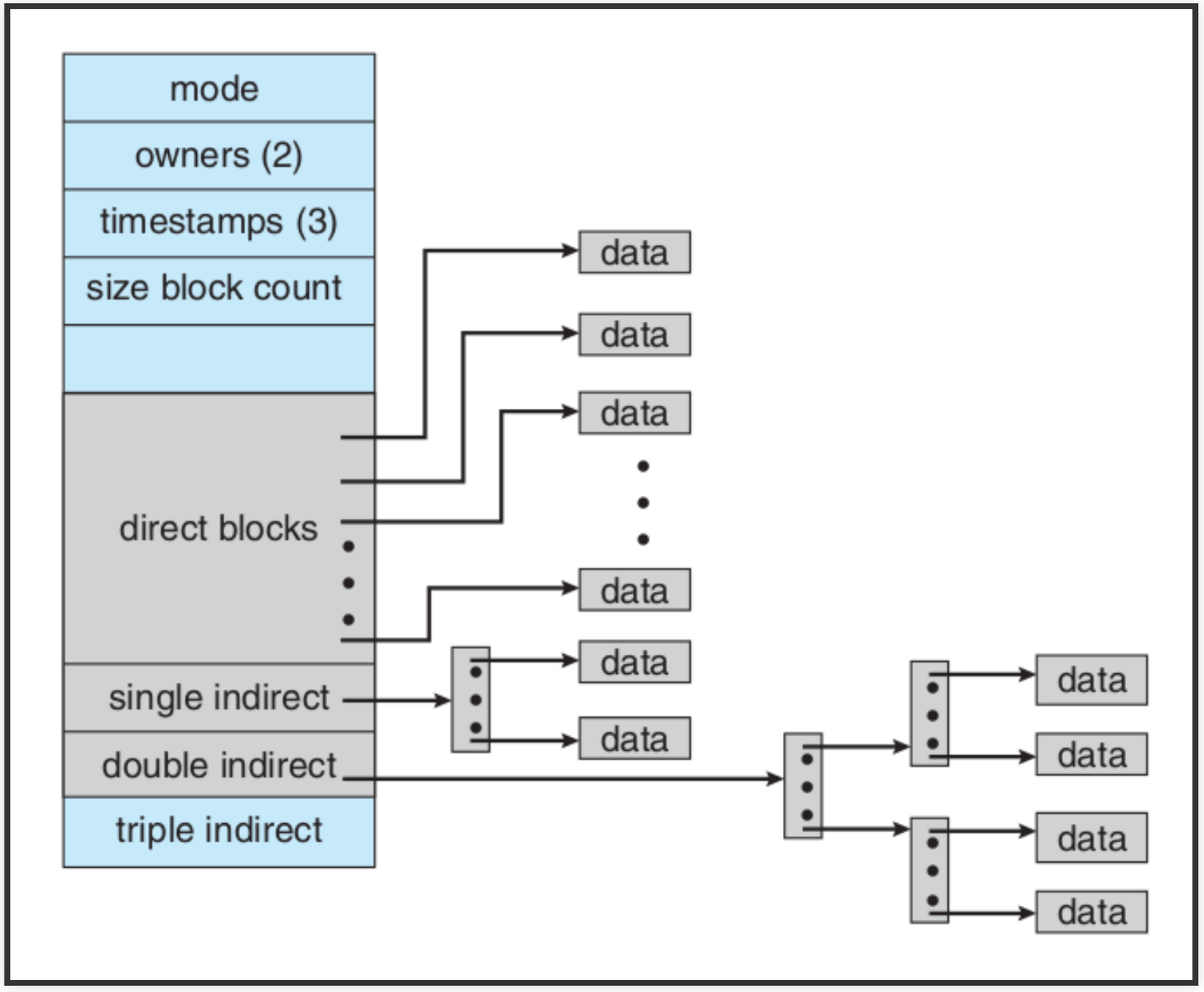
COMBINED SCHEME

- First 15 pointers to point to data
- Next points to single level index block
- Next to 2 level index block
- Last to Tripple indirect block

On 32 bit systems, this is larger file than can be addressed

COMBINED SCHEME: UNIX UFS

(4K bytes per block, 32-bit addresses)


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data

DIRECTORY IMPLEMENTATION

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

DIRECTORY IMPLEMENTATION

- **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

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)
- Example: 2,3,4,5,8,9,10 are free, rest allocated
  - Bitvector: `00111100111`

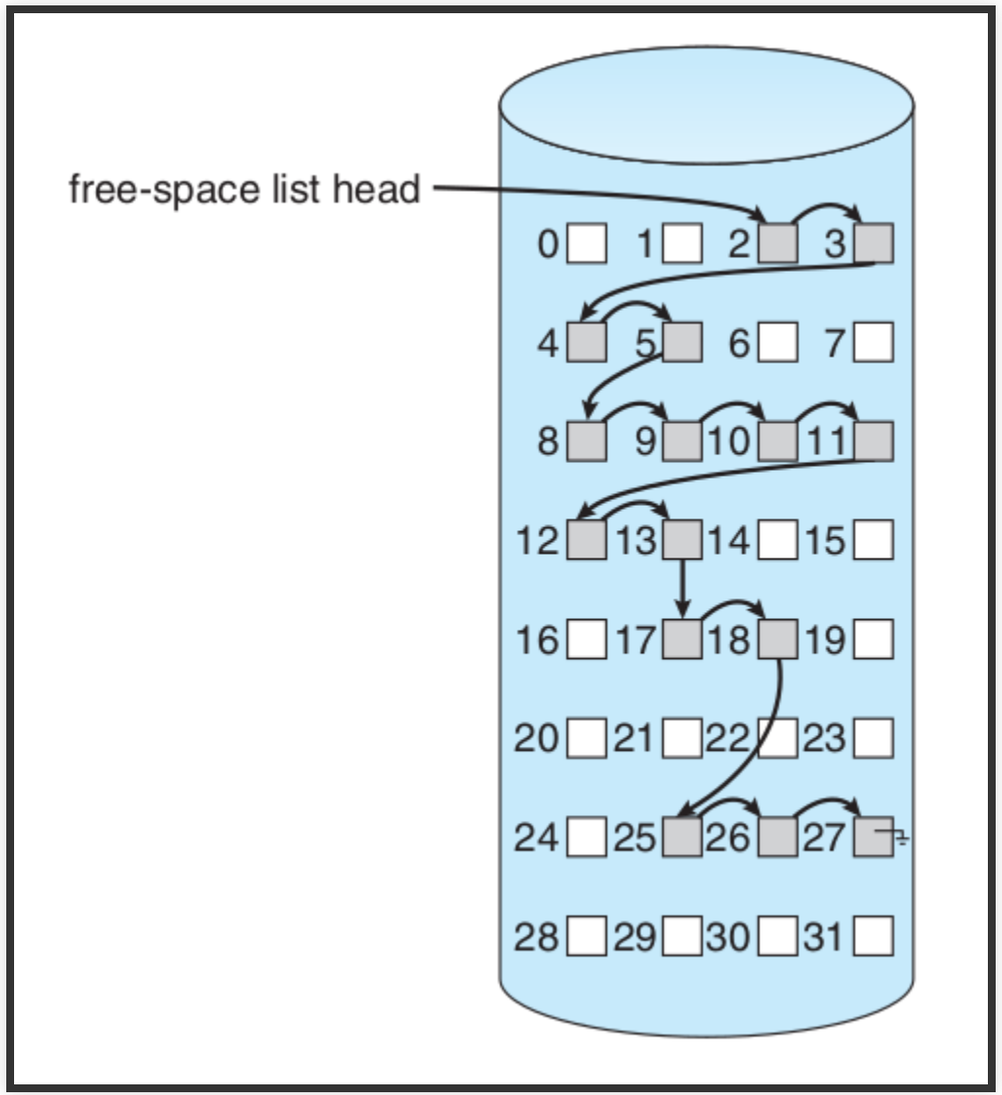
BITMAP

- 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

FREE LIST

- 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



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

SPACE MAPS

- Used in ZFS
- Consider meta-data I/O on very large file systems
  - Full data structures like bit maps couldn’t fit in memory → thousands of I/Os
- Divides device space into metaslab units and manages metaslabs
  - Given volume can contain hundreds of metaslabs

SPACE MAPS

- Each metaslab has associated space map
  - Uses counting algorithm
- But records to log file rather than file system □** Log of all block activity, in time order, in counting format
- Metaslab activity → load space map into memory in balanced-tree structure, indexed by offset
  - Replay log into that structure
  - Combine contiguous free blocks into single entry

RECOVERY, EFFICIENCE AND PERFORMANCE

EFFICIENCY

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

PERFORMANCE

PERFORMANCE

- 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 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

RECOVERY

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- 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 SYSTEMS

- File system as a circular buffer
- 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

LOG STRUCTURED FILE SYSTEMS

- 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
- 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

SPECIAL TYPES OF FILESYSTEMS

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

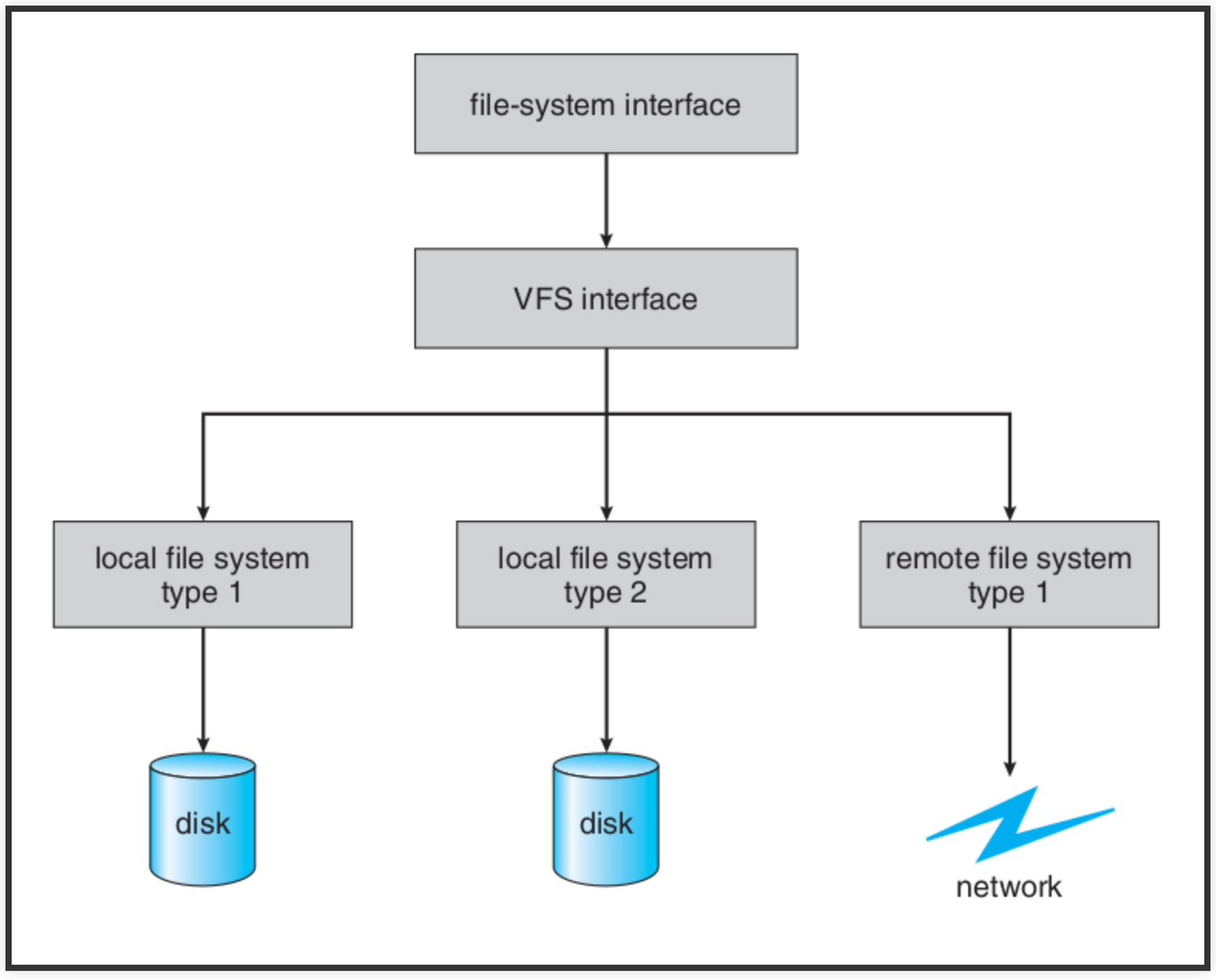
PARTITIONS AND MOUNTING

- 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

- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems
- The API is to the VFS interface, rather than any specific type of file system

SCHEMATIC VIEW OF VFS



VIRTUAL FILE SYSTEMS

- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - 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

VFS IMPLEMENTATION

- For example, Linux has four object types:
  - inode (individual file)
  - file (open file)
  - superblock (entire filesystem)
  - dentry (individual directory entry)

VFS IMPLEMENTATION

- 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

NFS - THE SUN NETWORK FILE SYSTEM

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet)

NFS

- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
- A remote directory is mounted over a local file system directory
  - The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory

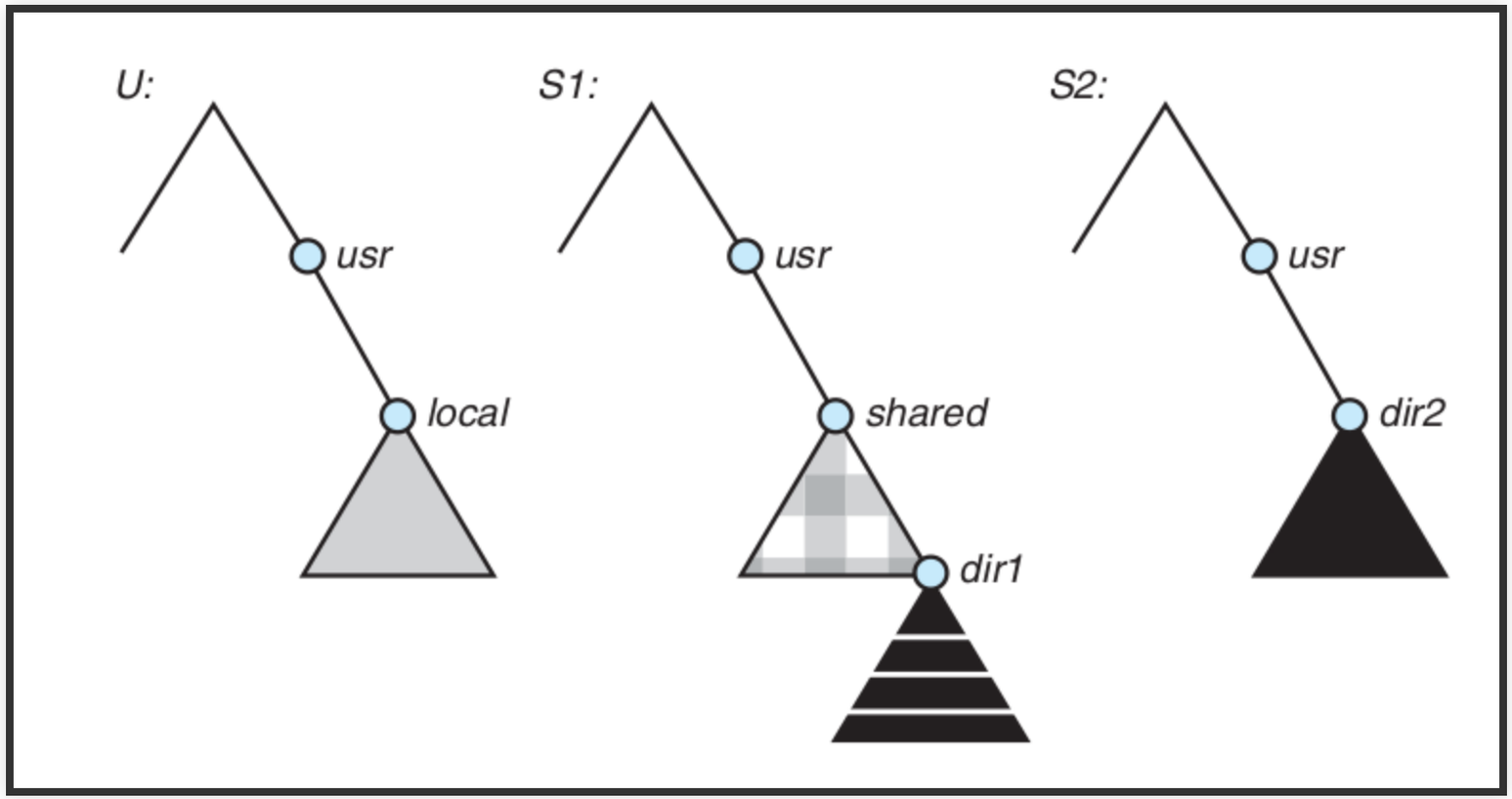
NFS

- Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
  - Files in the remote directory can then be accessed in a transparent manner
- Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

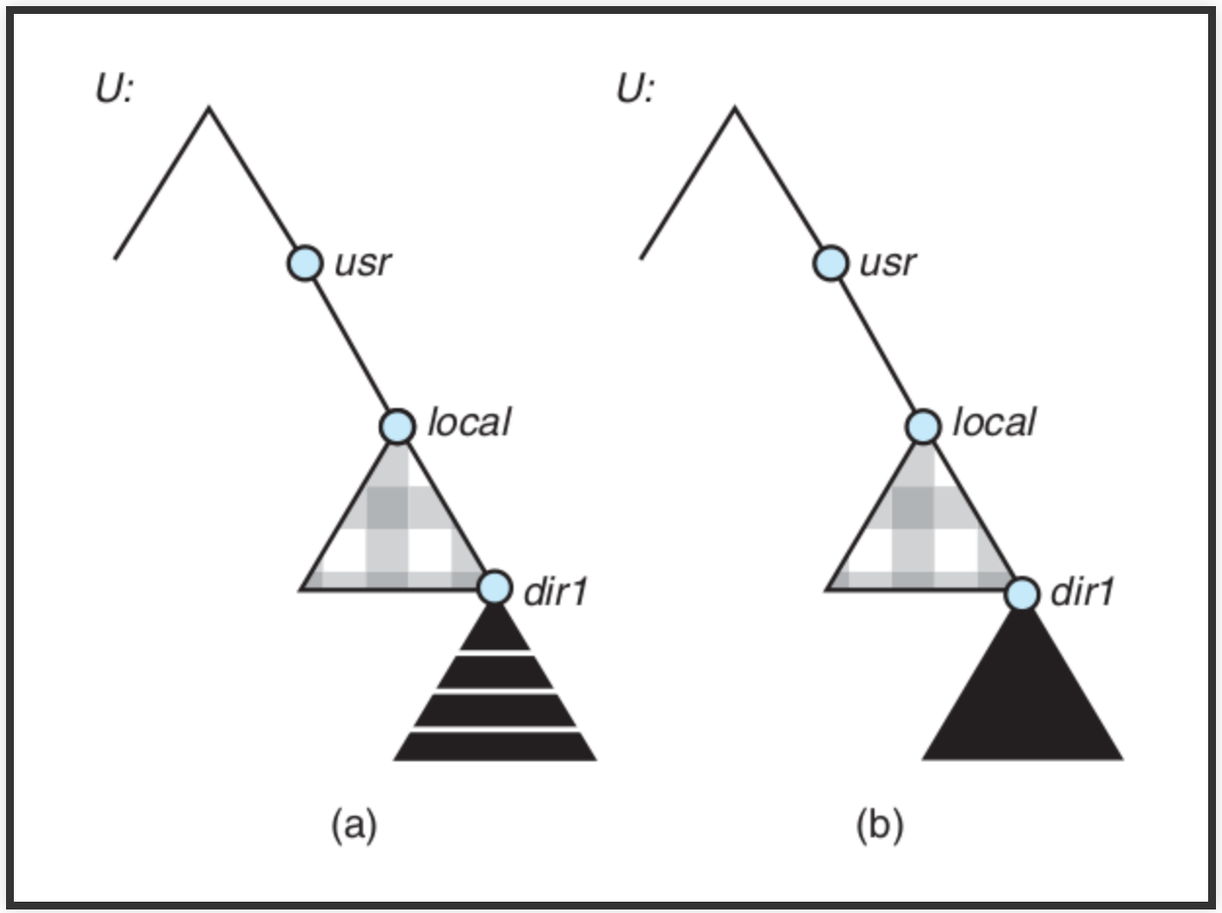
NFS

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces

THREE INDEPENDENT FILE SYSTEMS



MOUNTING IN NFS



NFS MOUNT PROTOCOL

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them

NFS MOUNT PROTOCOL

- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

NFS PROTOCOL

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
  - searching for a file within a directory
  - reading a set of directory entries
  - manipulating links and directories
  - accessing file attributes
  - reading and writing files

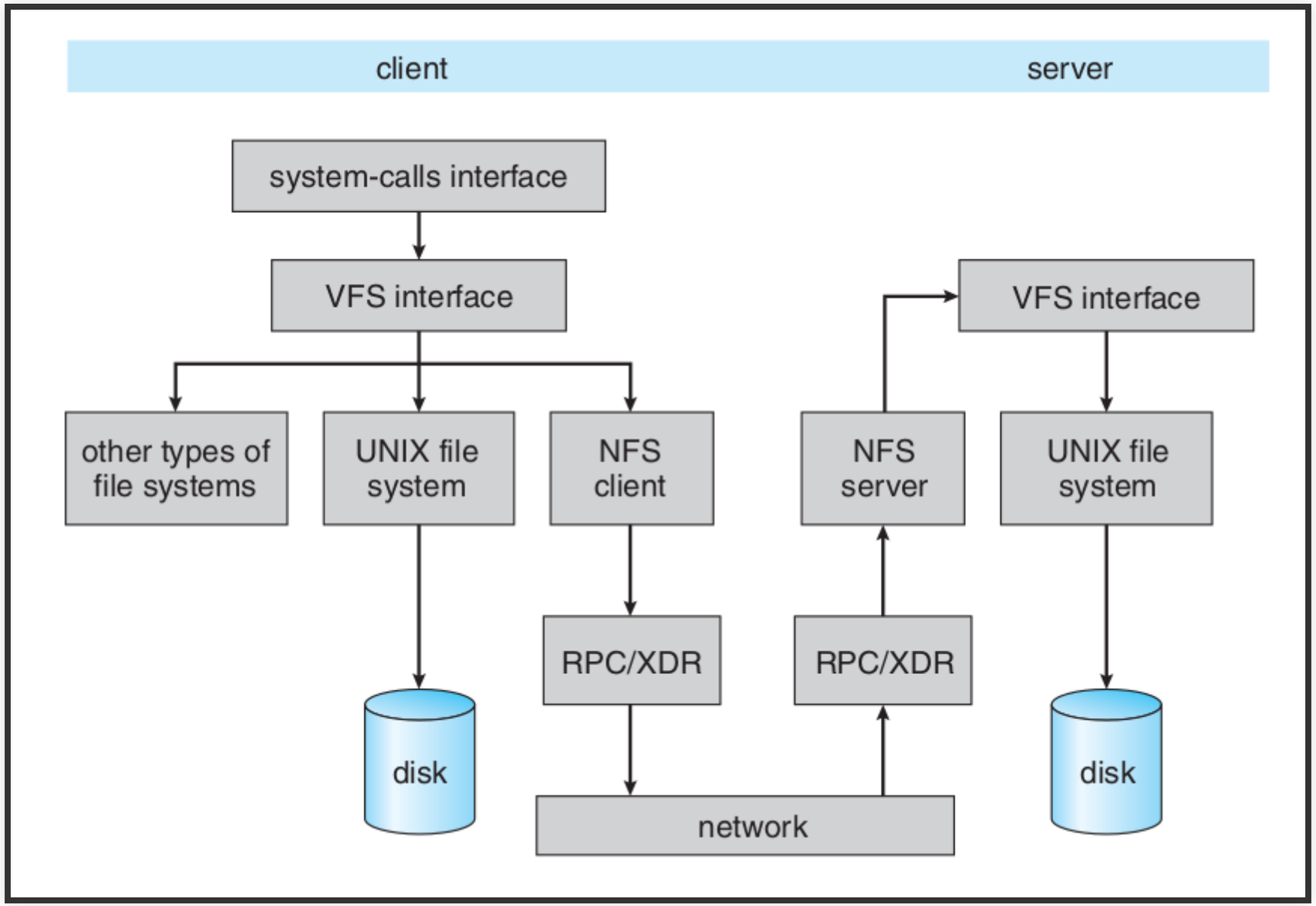
NFS PROTOCOL

- NFS servers are stateless; each request has to provide a full set of arguments (NFS V4 is available –> stateful)
- Modified data must be committed to the server’s disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms

THREE MAJOR LAYERS OF NFS ARCHITECTURE

1. UNIX file-system interface (based on the open, read, write, and close calls, and file descriptors)
2. Virtual File System (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
3. NFS service layer – bottom layer of the architecture
    - Implements the NFS protocol

SCHEMATIC VIEW OF NFS ARCHITECTURE



NFS PATH-NAME TRANSLATION

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

NFS REMOTE OPERATIONS

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance

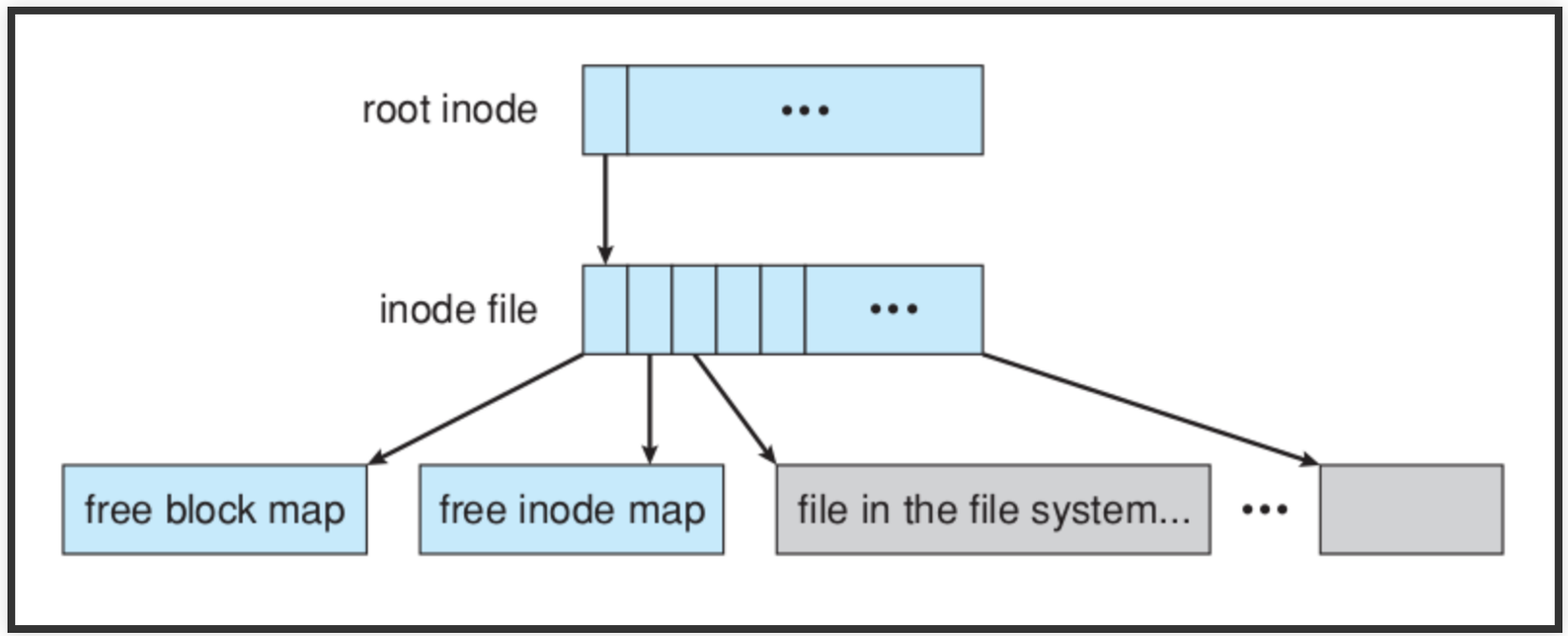
NFS REMOTE OPERATIONS

- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk

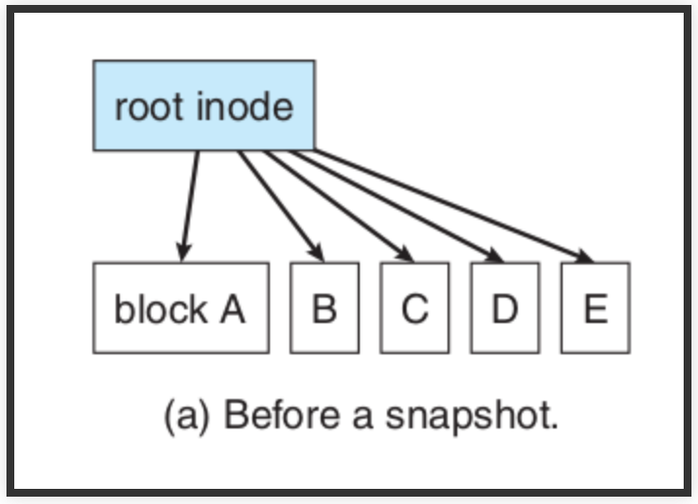
EXAMPLE: THE WAFL FILE SYSTEM

- Used on Network Appliance “Filers” – distributed file system appliances
- “Write-anywhere file layout”
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
  - NVRAM for write caching

THE WAFL FILE LAYOUT

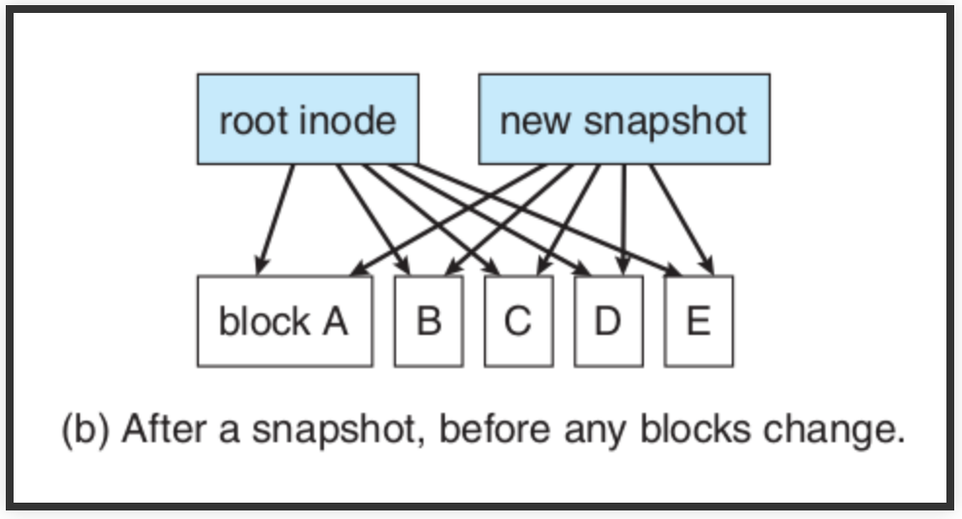


SNAPSHOTS IN WAFL



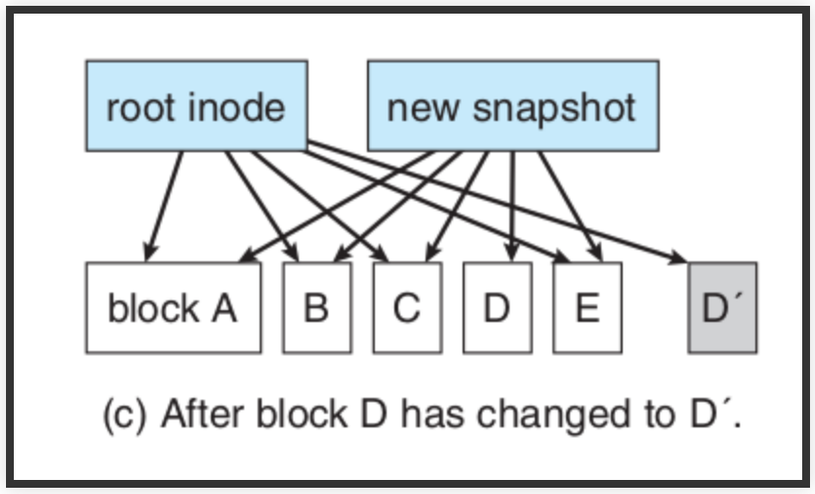
(a) Before a snapshot.

SNAPSHOTS IN WAFL



(b) After a snapshot, before any blocks change.

SNAPSHOTS IN WAFL



(c) After block D has changed to D´.

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

FILE SHARING

- 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

FILE SHARING – REMOTE FS

- 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

FILE SHARING – REMOTE FS

- 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

FILE SHARING – REMOTE FS

- Distributed Information Systems (distributed naming services) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

FILE SHARING – FAILURE MODES

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called metadata
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

FILE SHARING – CONSISTENCY SEMANTICS

- Specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 6 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

FILE SHARING – CONSISTENCY 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

QUESTIONS

BONUS

Exam question number 8: **File-System and Implementing File-Systems**