Chapter 12: File System Basics

Chapter 12: File-System Basics

- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection 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
- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

File Concept

- Contiguous logical address space
- Types:
    - Data
        - numeric
        - character
        - binary
    - Program (executable)

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 Attributes

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

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

File Operations

- File is an abstract data type
- Create
- Write
- Read
- 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

Based on Slides by Silberschatz, Galvin and Gagne ©2009

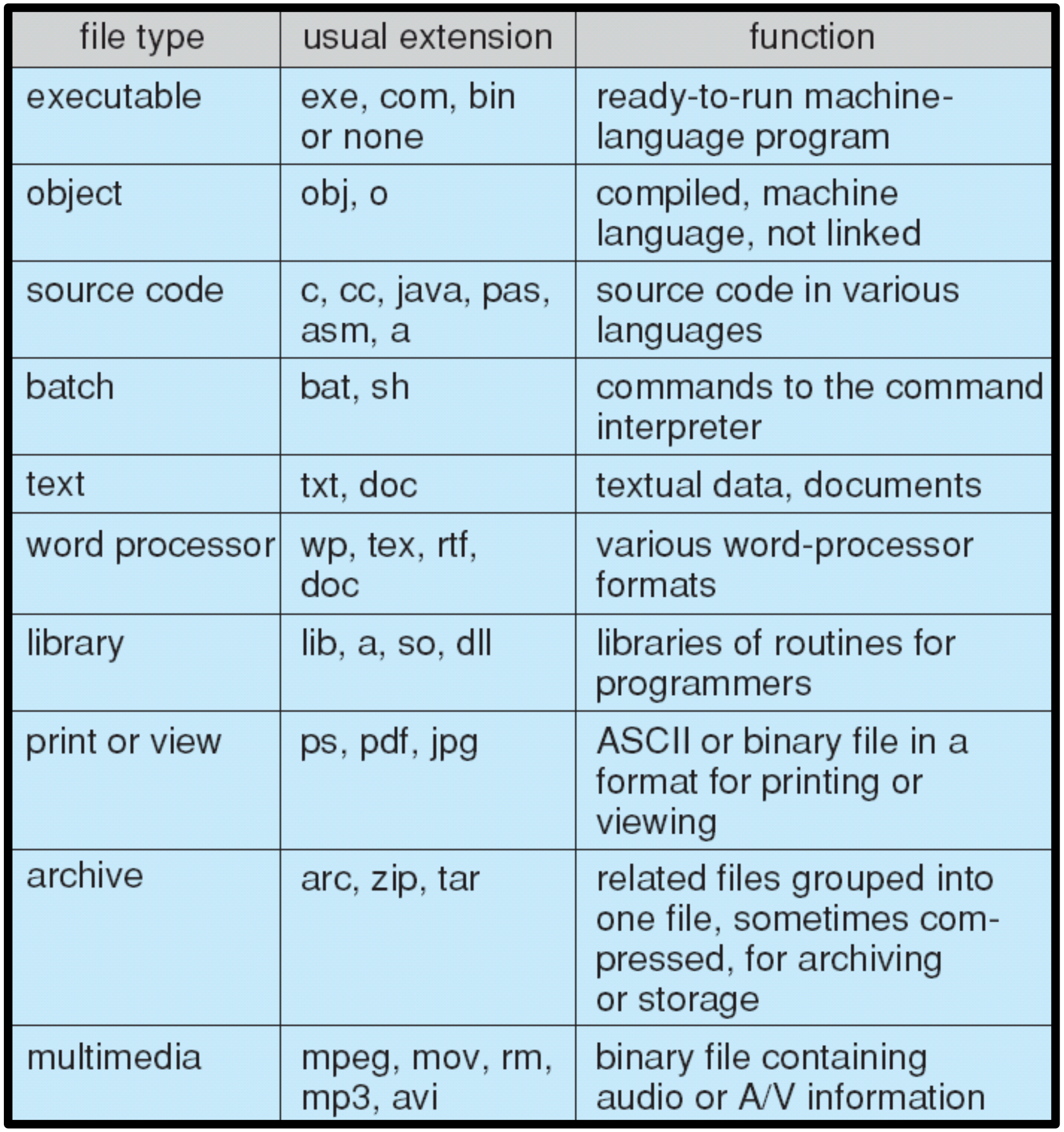
Open Files

- Several pieces of data are needed to manage open files:
  - File pointer: pointer to last read/write location, per process that has the file open
  - File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - 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
- Mediates access to a file
- 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, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Based on Slides by Silberschatz, Galvin and Gagne ©2009

Access Methods

- Sequential Access

  read next
  write next
  reset
  no read after last write
  (rewrite)

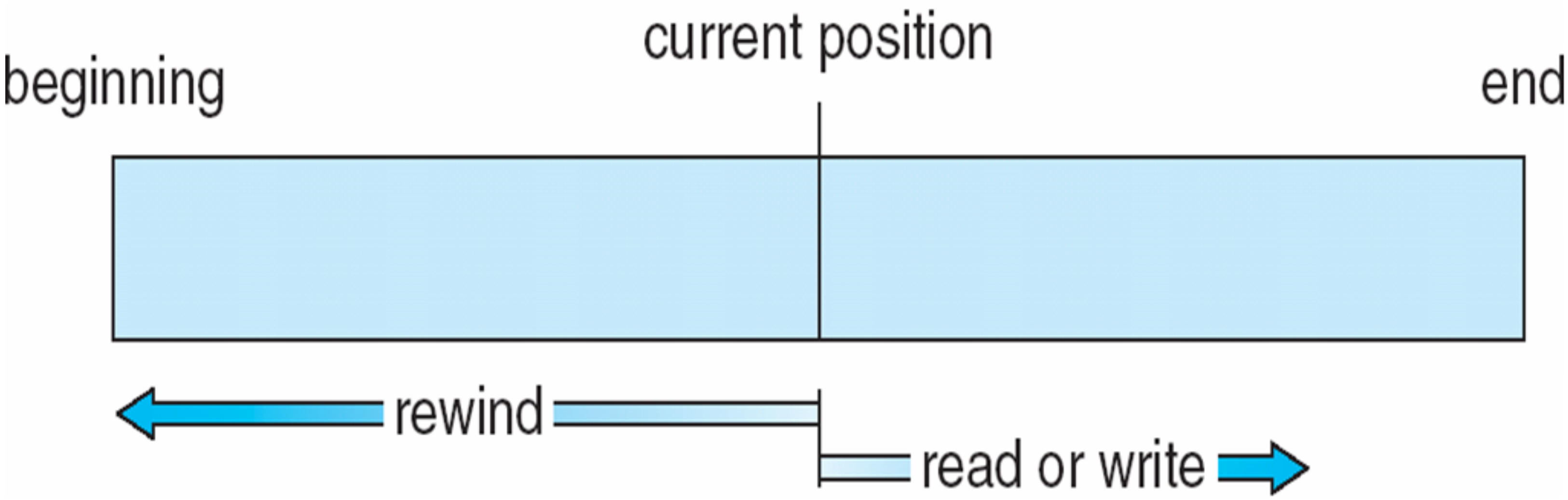
- Direct Access

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

n = relative block number

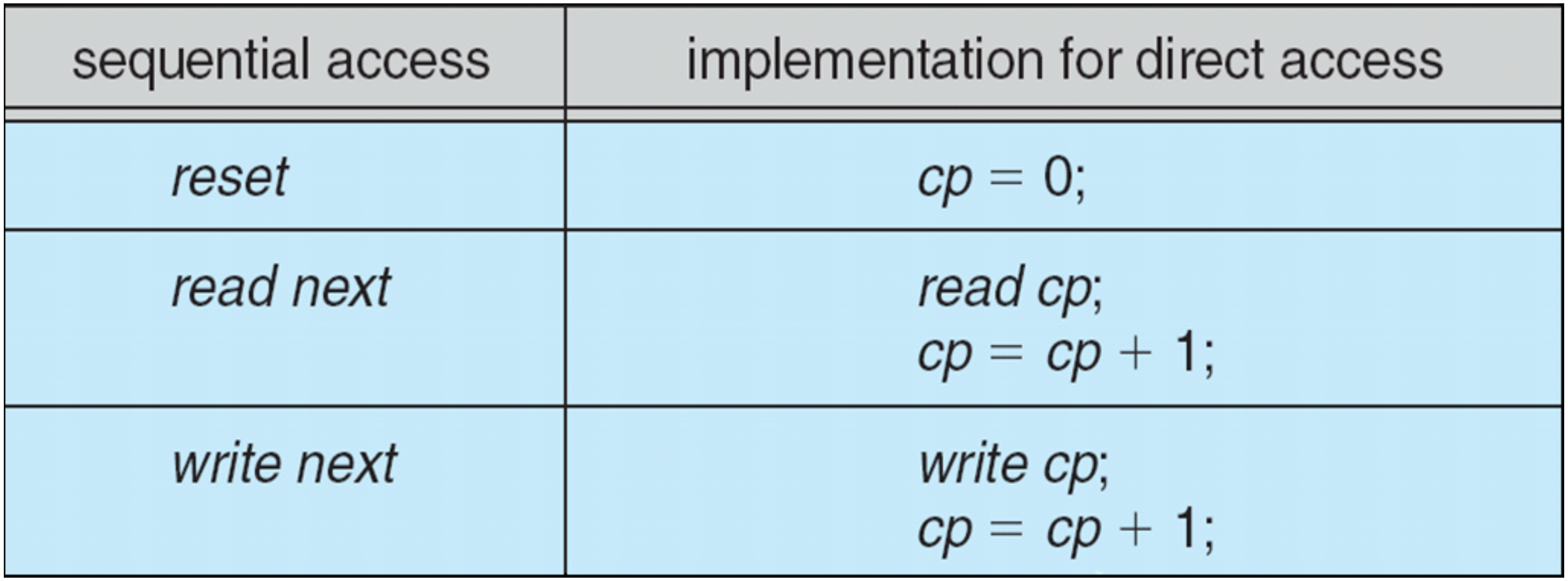
Based on Slides by Silberschatz, Galvin and Gagne ©2009

Sequential-access File



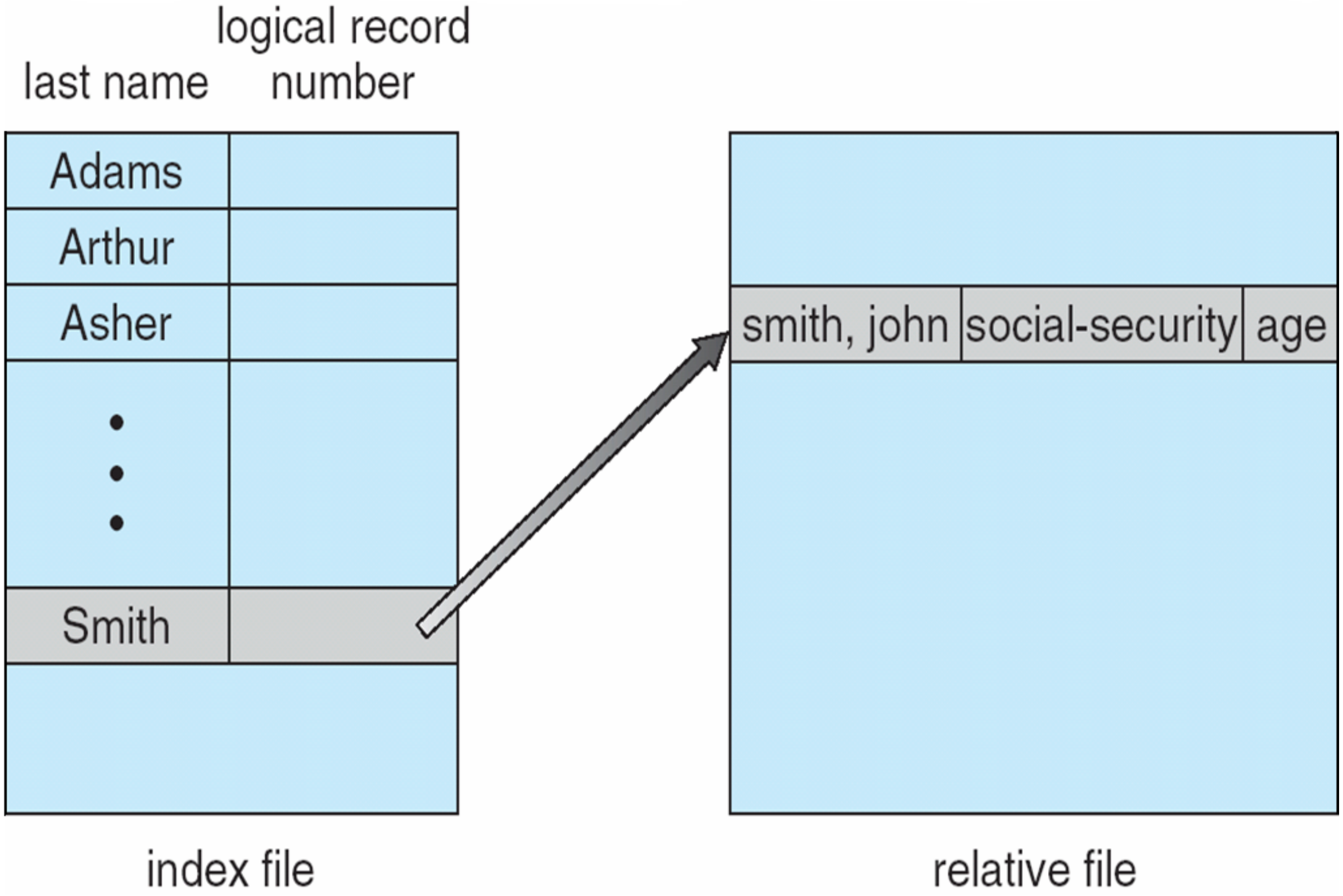
Based on Slides by Silberschatz, Galvin and Gagne ©2009

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;

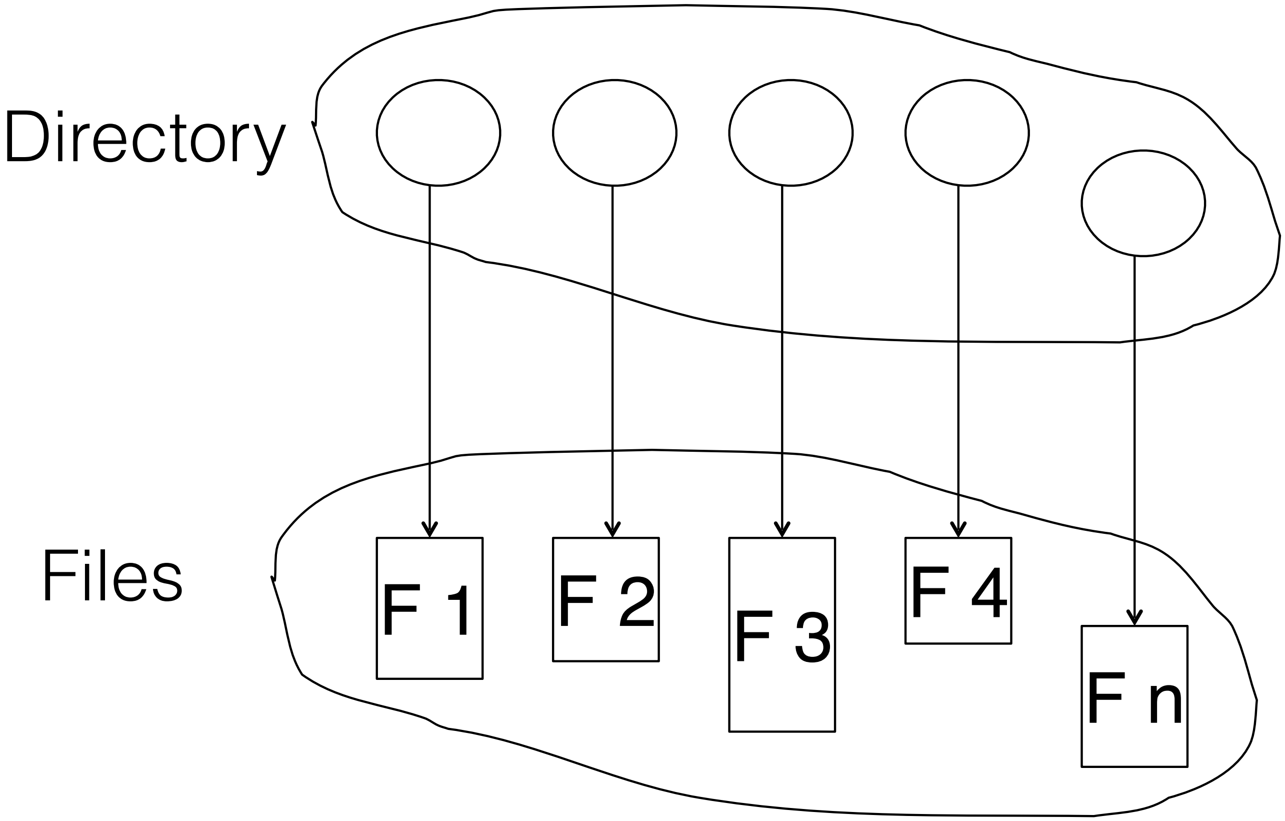
Example of Index and Relative Files



Based on Slides by Silberschatz, Galvin and Gagne ©2009

Directory Structure

A collection of nodes containing information about all files



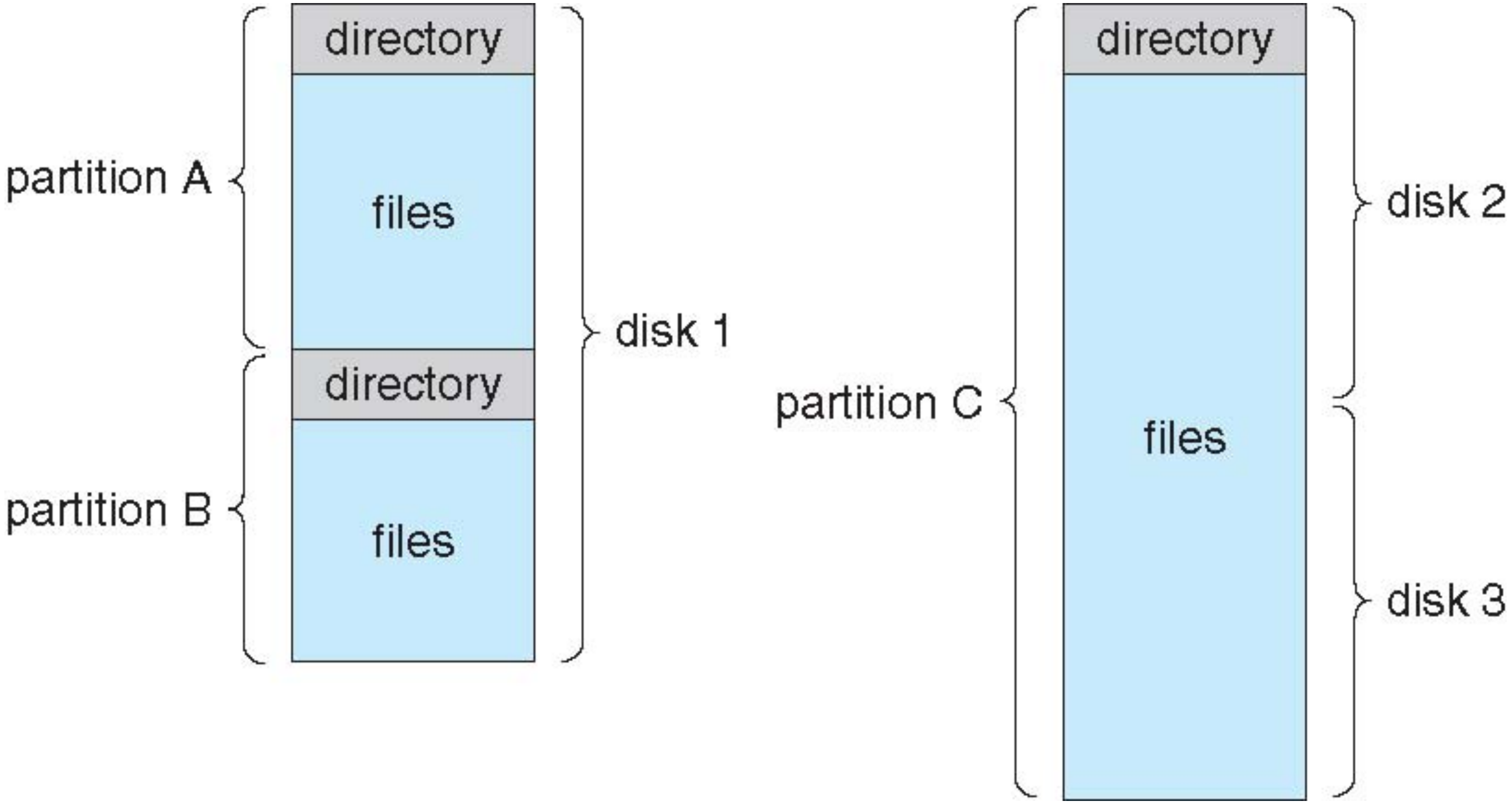
Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

Based on Slides by Silberschatz, Galvin and Gagne ©2009

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
- 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

A Typical File-system Organization



Based on Slides by Silberschatz, Galvin and Gagne ©2009

Operations Performed on Directory

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

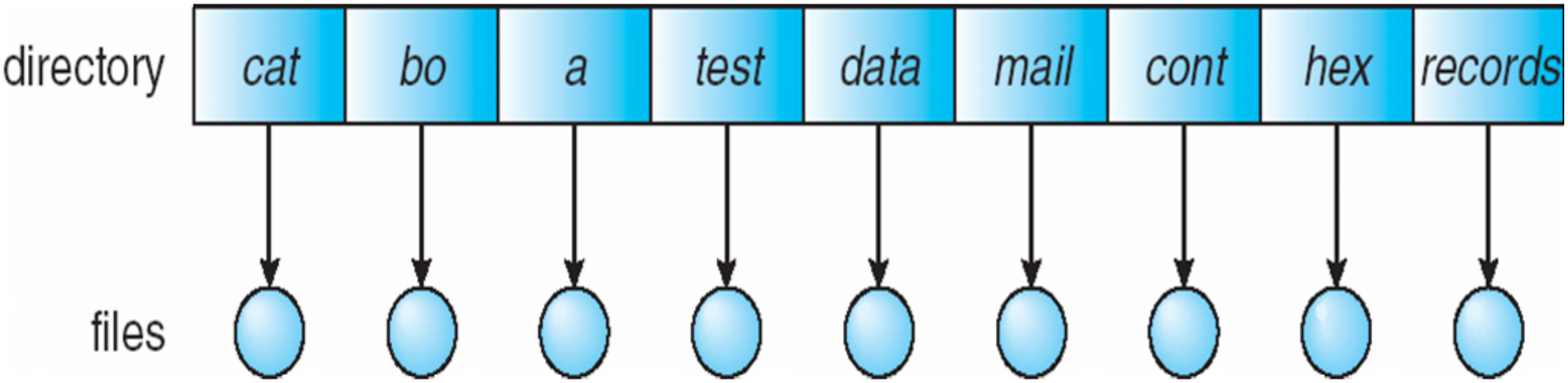
Organize the Directory (Logically) to Obtain

- 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, …)

Based on Slides by Silberschatz, Galvin and Gagne ©2009

Single-Level Directory

- A single directory for all users



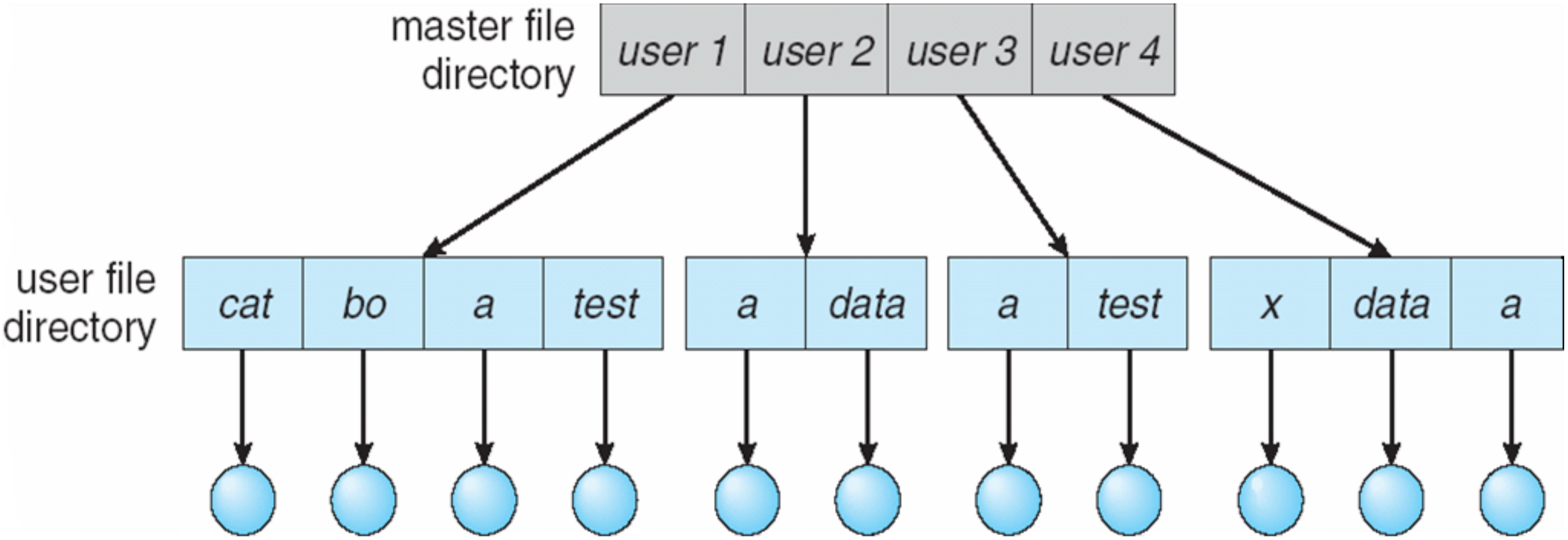
Naming problem

Grouping problem

Based on Slides by Silberschatz, Galvin and Gagne ©2009

Two-Level Directory

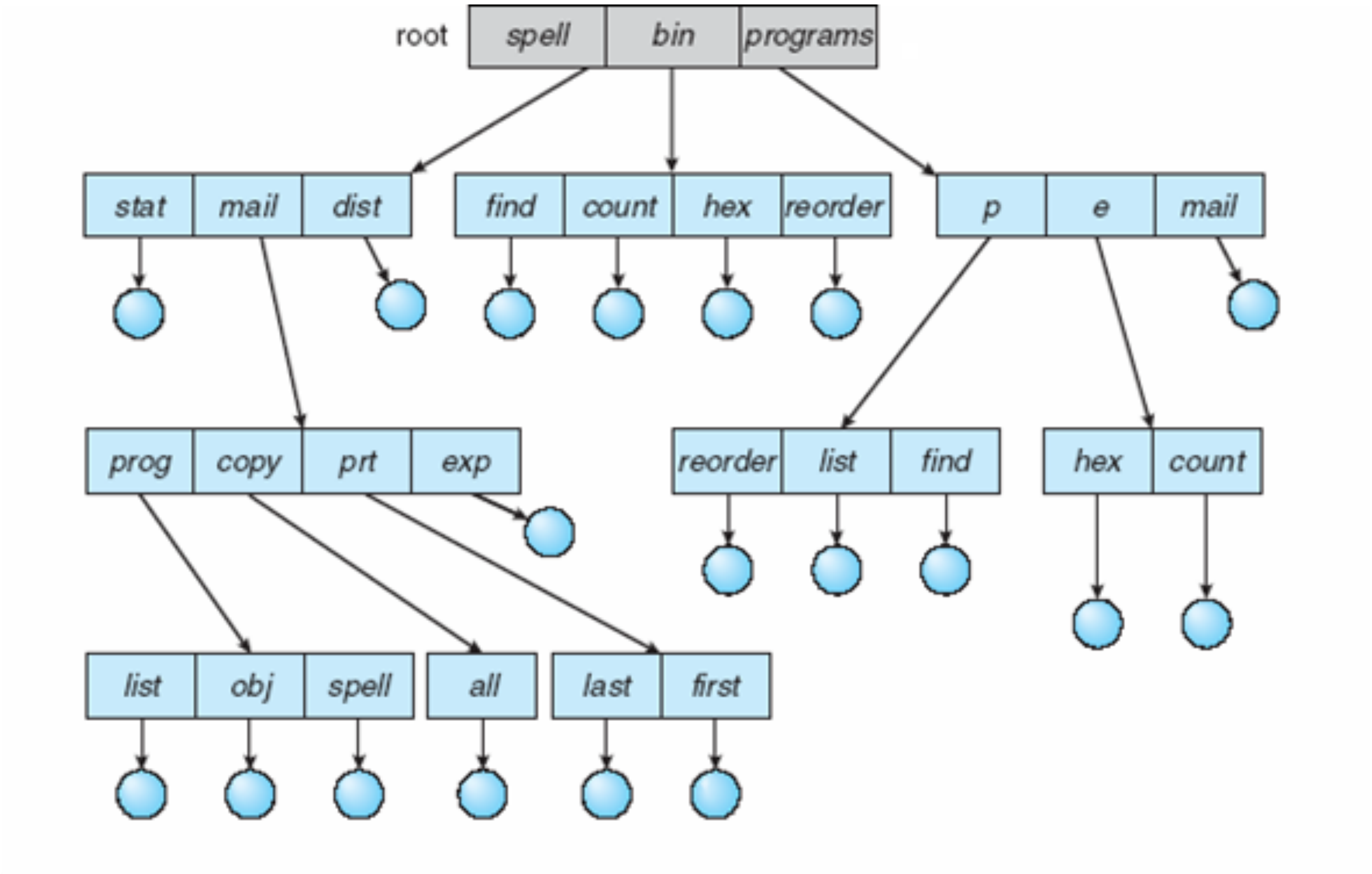
- Separate directory for each user



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

Based on Slides by Silberschatz, Galvin and Gagne ©2009

Tree-Structured Directories



Based on Slides by Silberschatz, Galvin and Gagne ©2009

Tree-Structured Directories (Cont.)

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

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

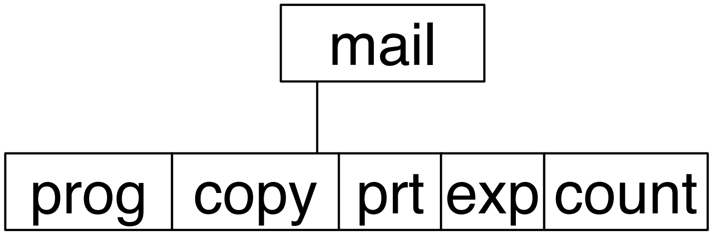
  rm <file-name>

- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory **/mail**

  mkdir count



mail

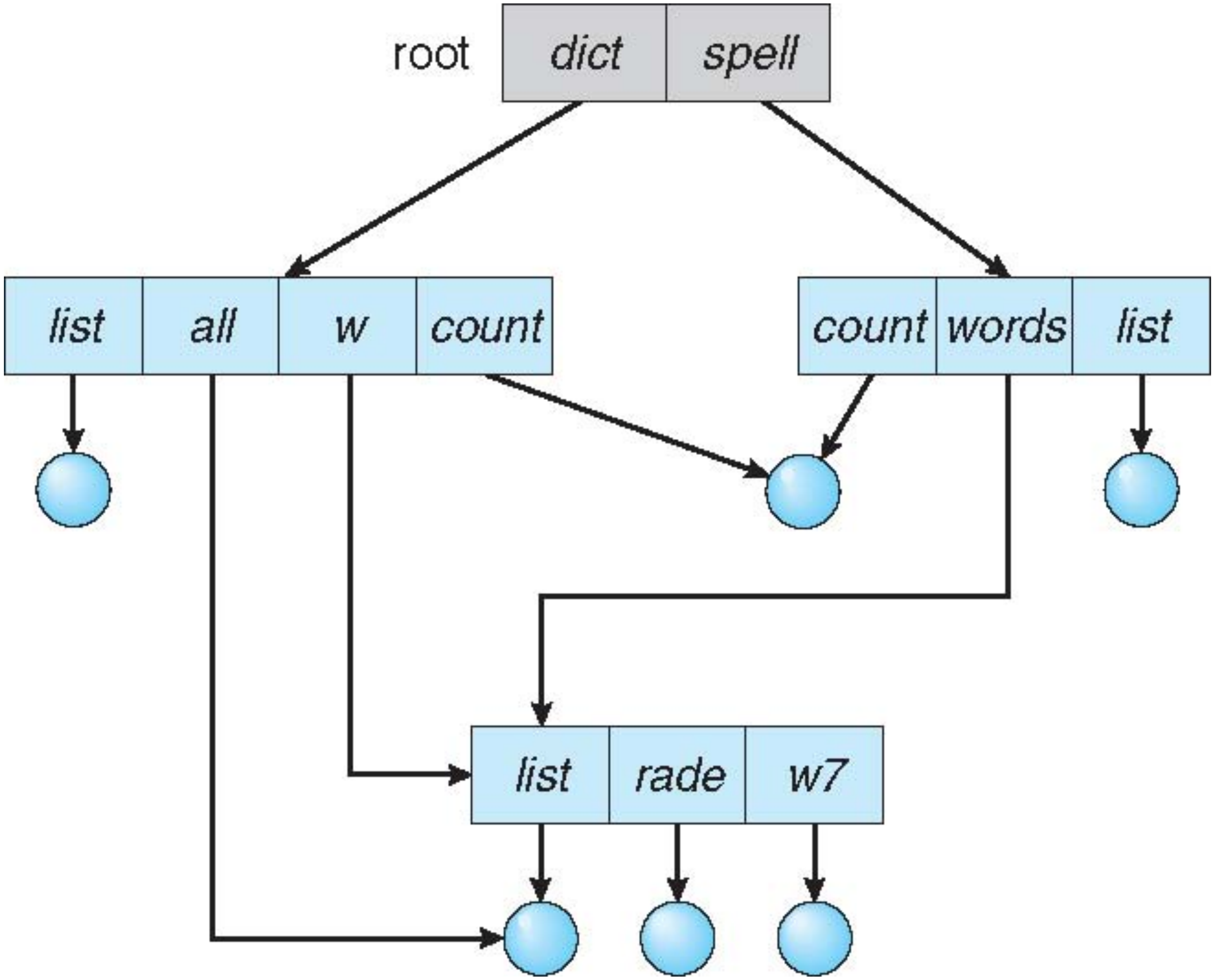
prog copy prt exp count

Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Based on Slides by Silberschatz, Galvin and Gagne ©2009

Directed Acyclic-Graph Directories

- Have shared subdirectories and files



Based on Slides by Silberschatz, Galvin and Gagne ©2009

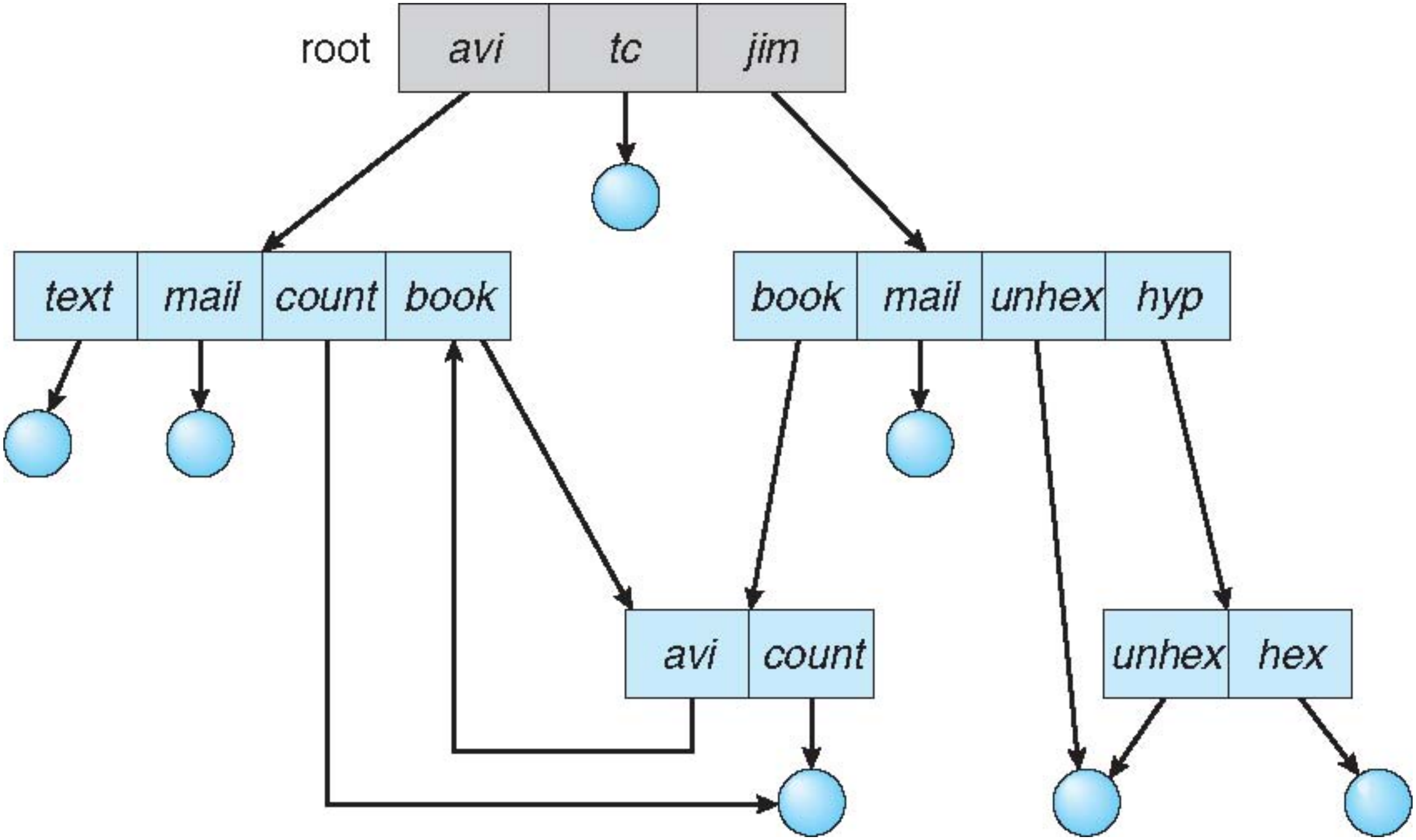
Acyclic-Graph Directories (Cont.)

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

General Graph Directory



Based on Slides by Silberschatz, Galvin and Gagne ©2009

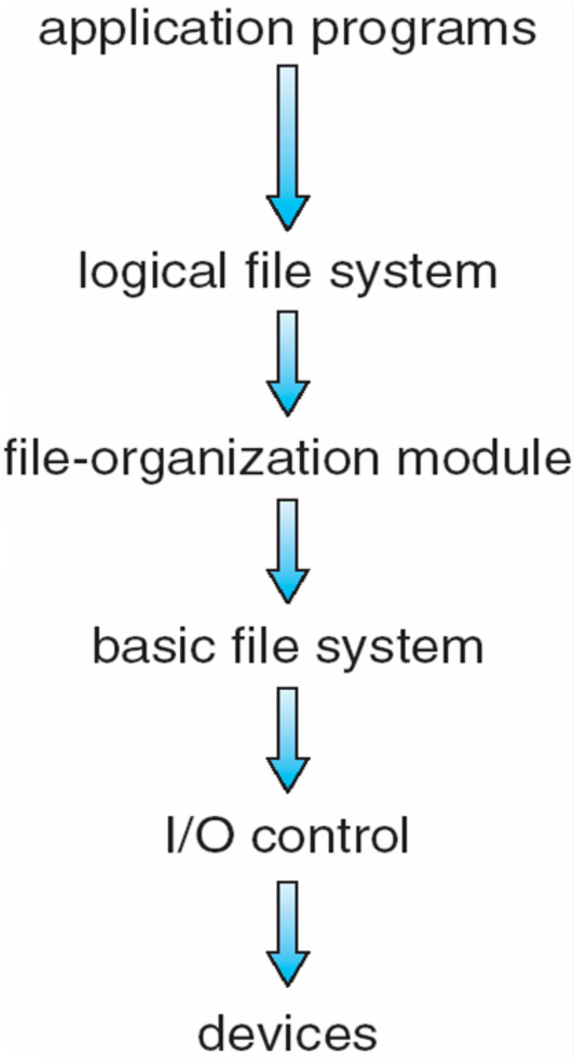
General Graph Directory (Cont.)

- 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

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system organized into layers
- **File system** resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device

Layered File System



application programs

logical file system

file-organization module

basic file system

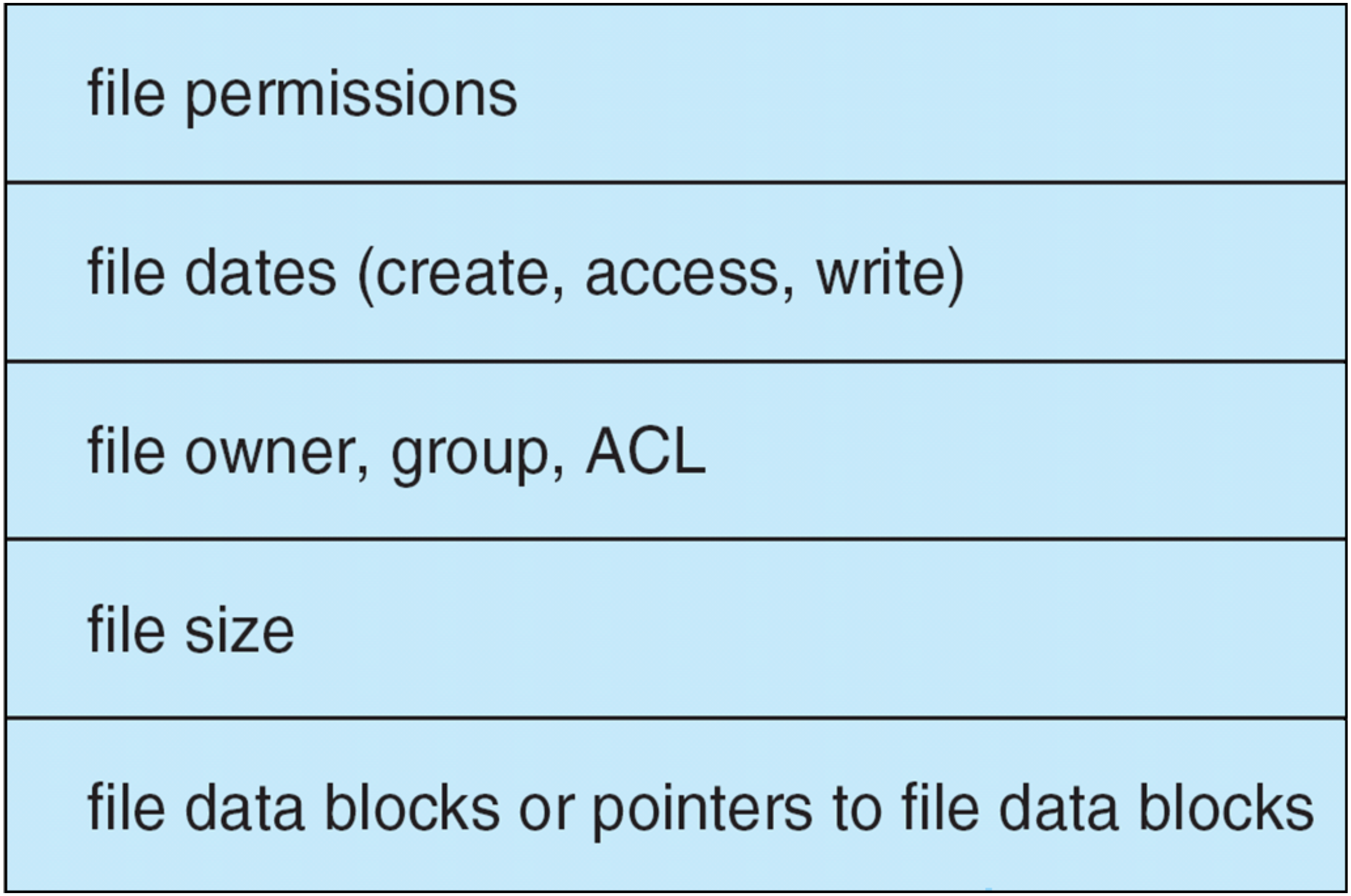
I/O control

devices

File-System Implementation

- **Boot control block** contains info needed by system to boot OS from that volume
- **Volume control block** contains volume details
- Directory structure organizes the files
- Per-file **File Control Block (FCB)** contains many details about the file

A Typical File Control Block



file permissions

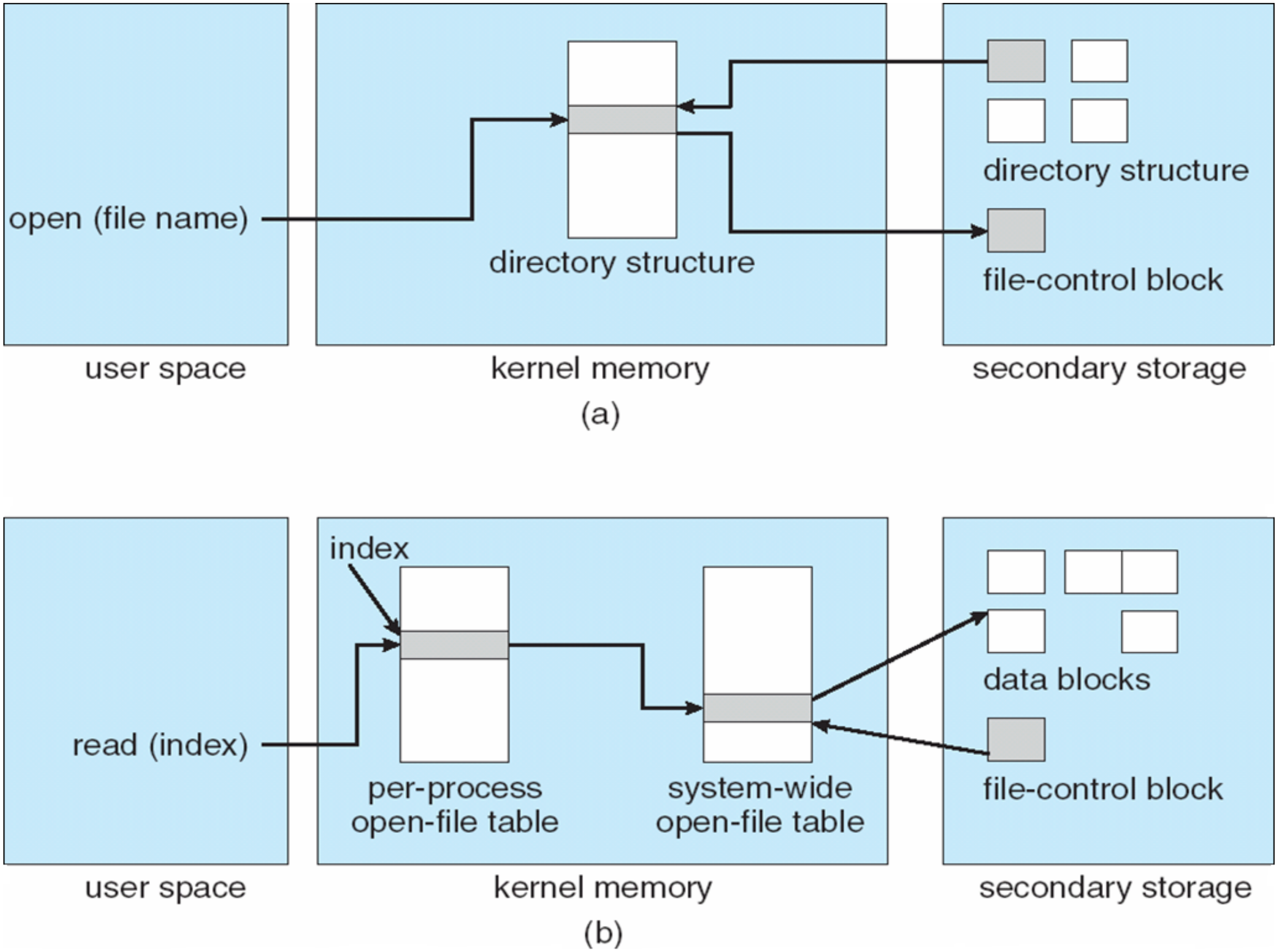
file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks

In-Memory File System Structures

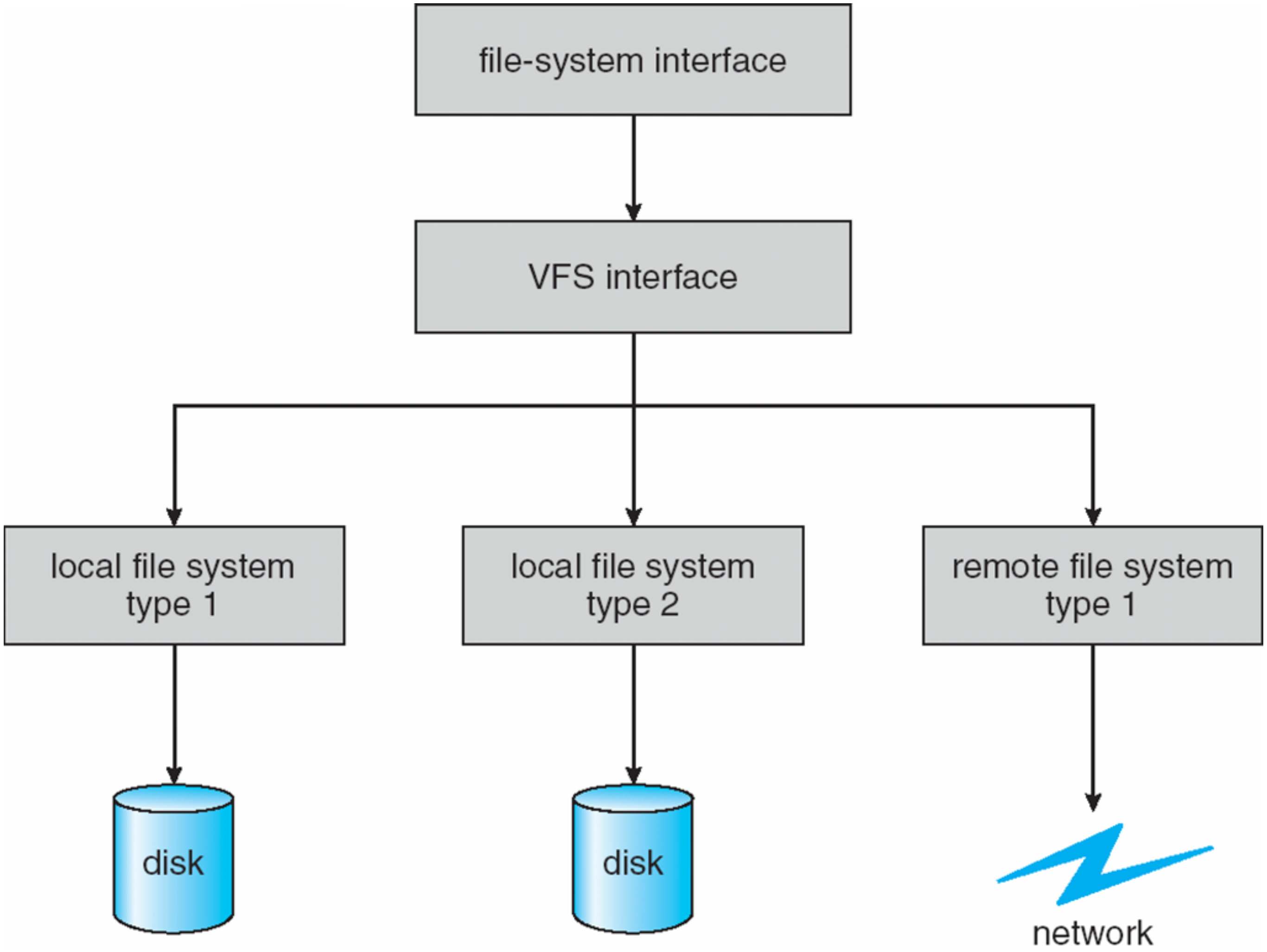


Silberschatz, Galvin and Gagne ©2009

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



Silberschatz, Galvin and Gagne ©2009

Directory Implementation

- Linear list of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure.
  - decreases directory search time
  - collisions – situations where two file names hash to the same location
  - fixed size

Allocation Methods

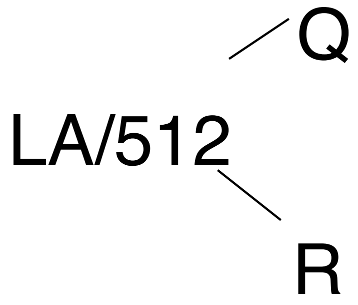
- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

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
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

Contiguous Allocation

- Mapping from logical to physical



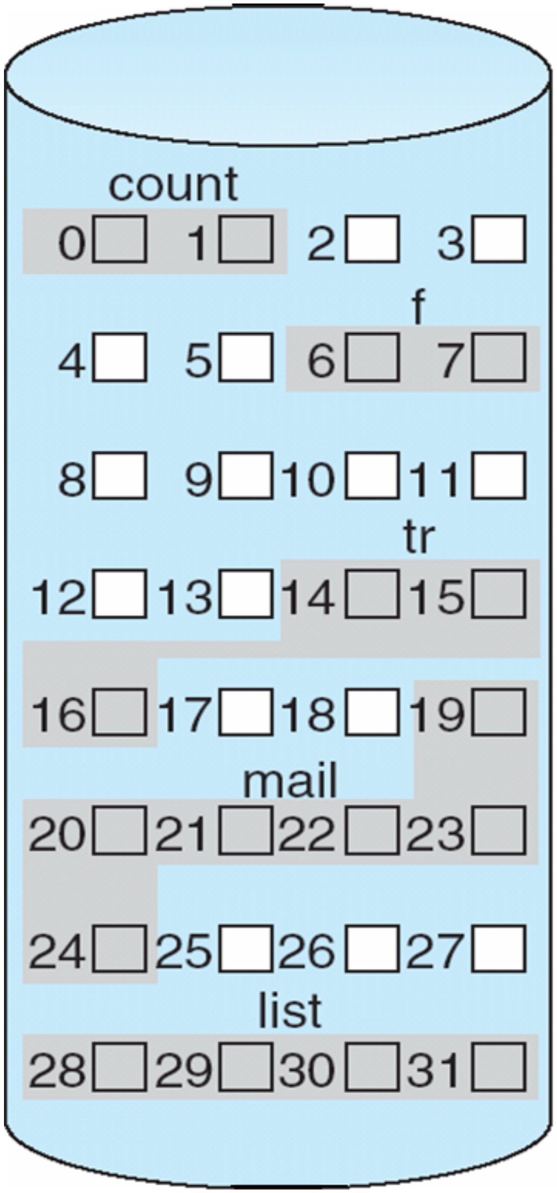
Q

LA/512

R

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

Contiguous Allocation of Disk Space



directory

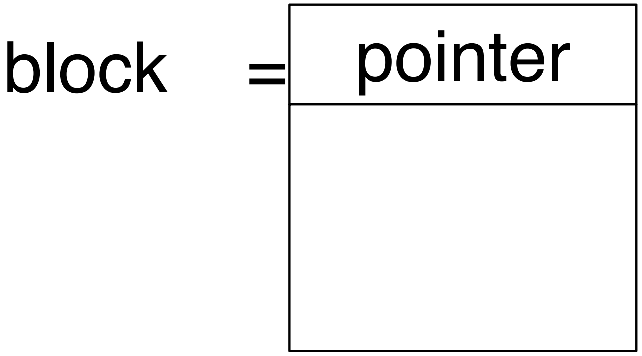
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

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

Linked Allocation

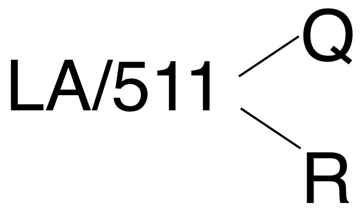
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



block = pointer

Linked Allocation (Cont.)

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



LA/511

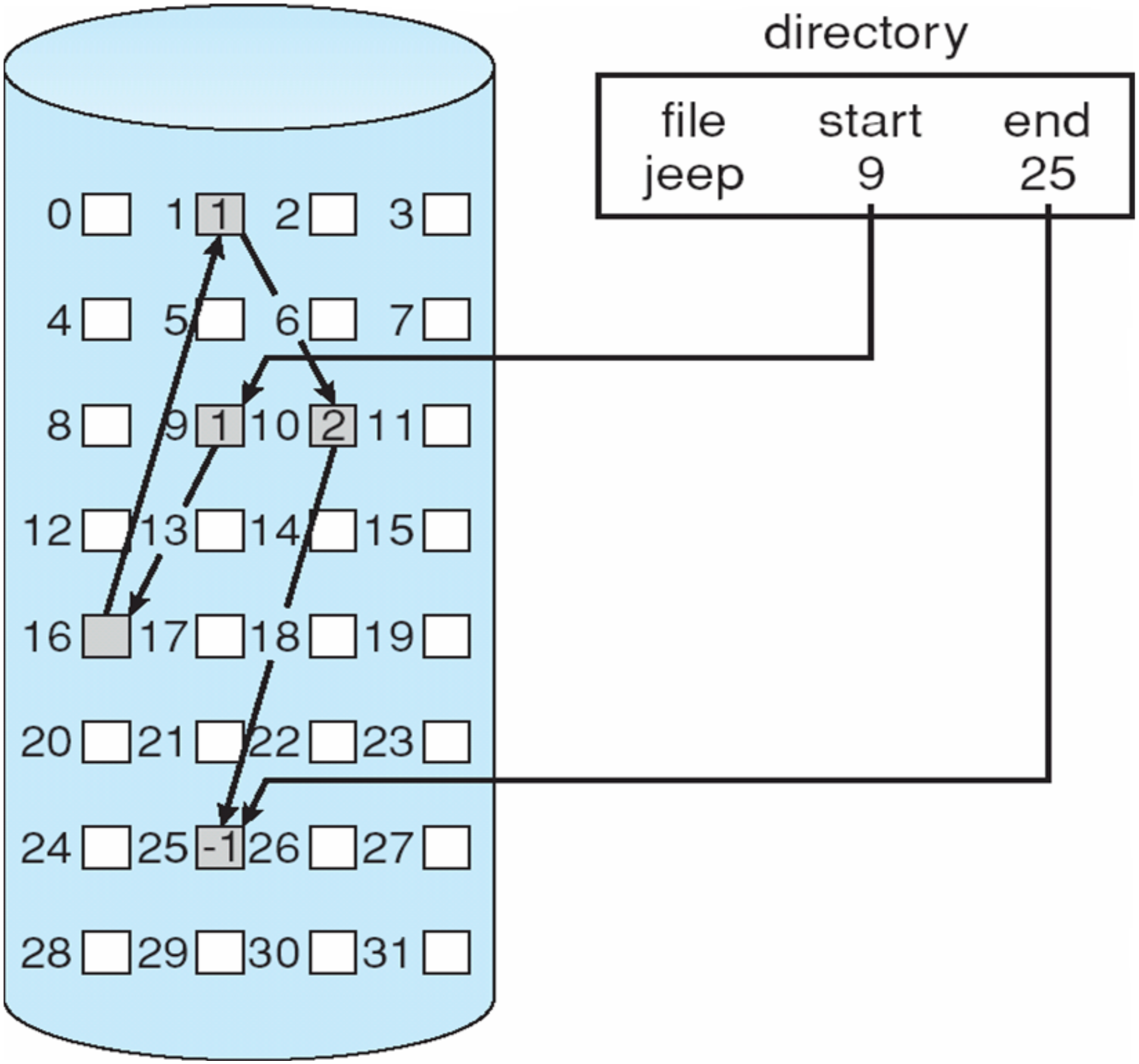
Q

R

Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 1

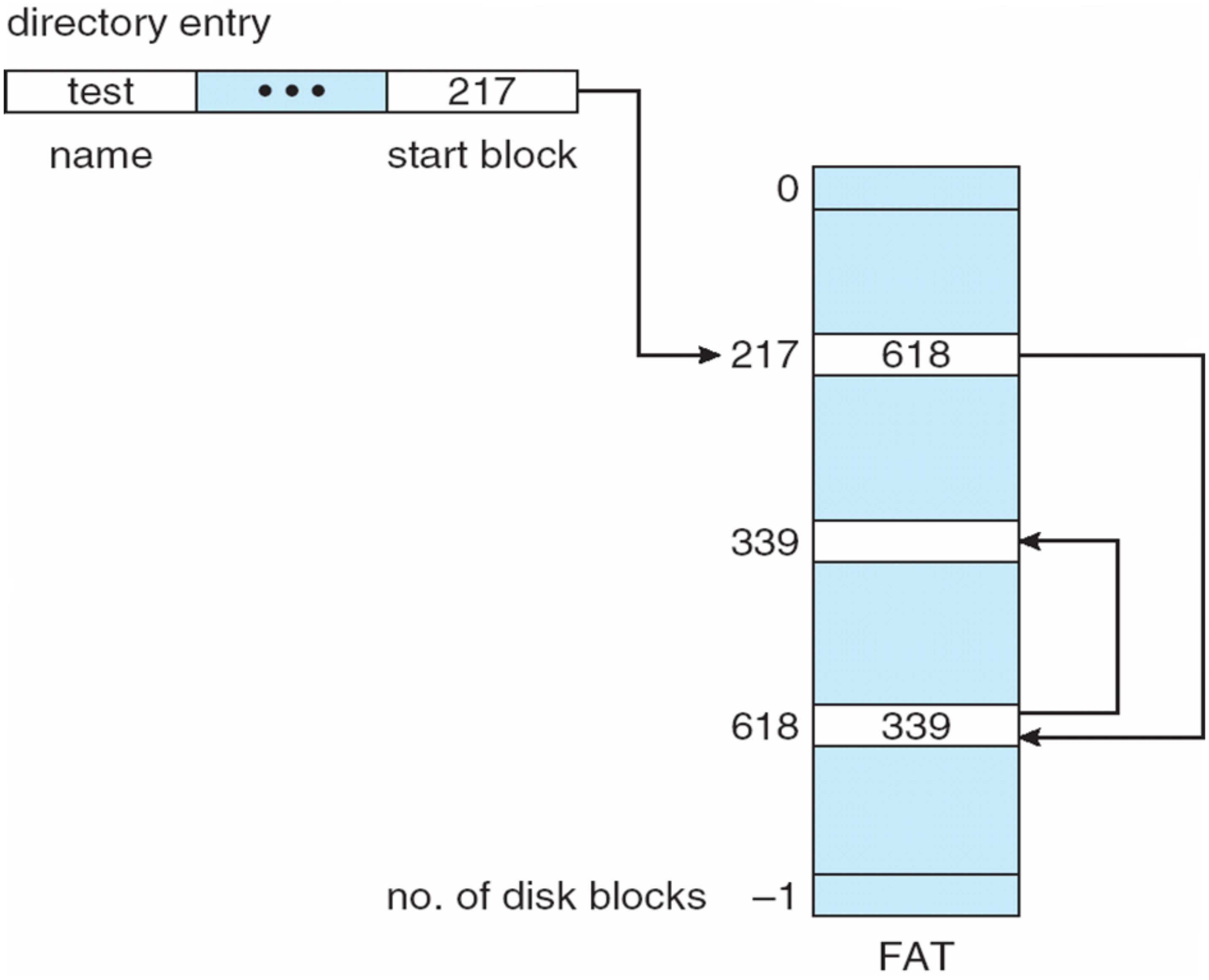
File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Linked Allocation



Silberschatz, Galvin and Gagne ©2009

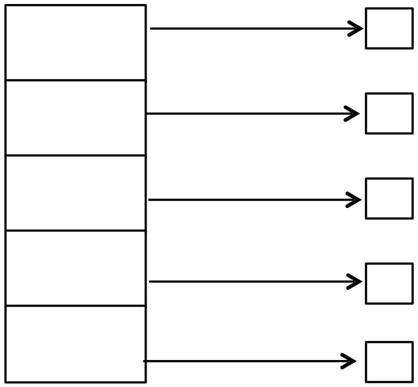
File-Allocation Table



Silberschatz, Galvin and Gagne ©2009

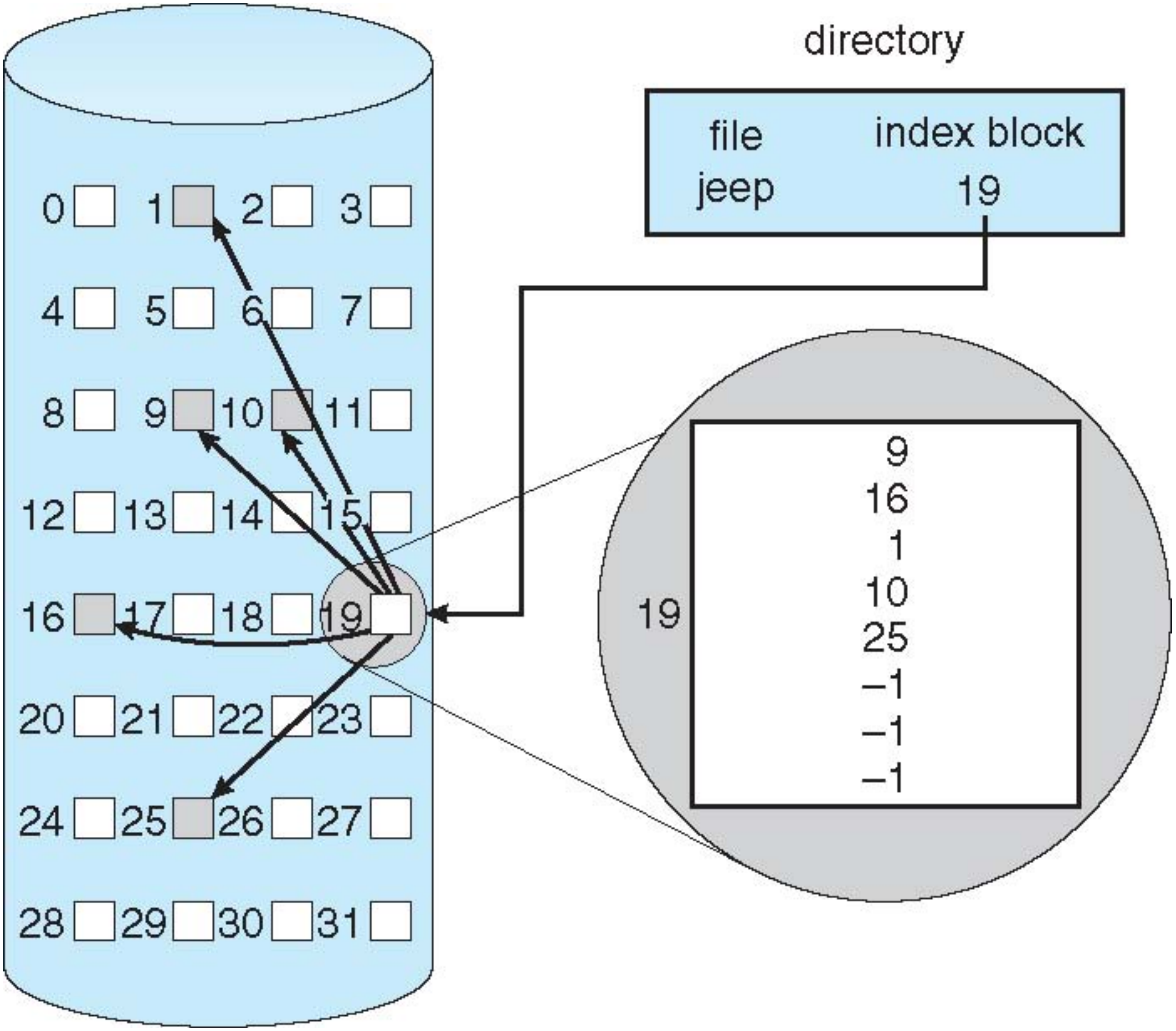
Indexed Allocation

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



index table

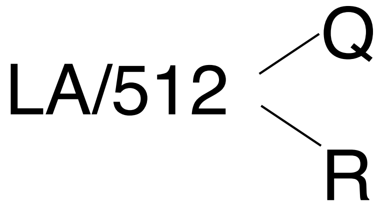
Example of Indexed Allocation



Silberschatz, Galvin and Gagne ©2009

Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table



LA/512

Q

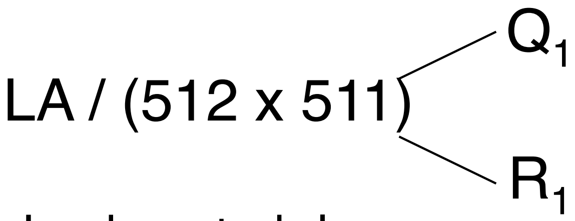
R

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)



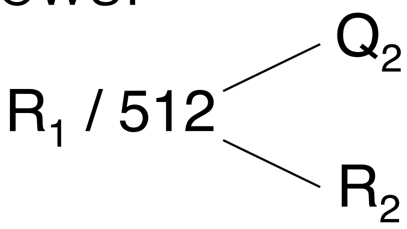
LA / (512 x 511)

Q_1

R_1

Q_1 = block of index table

R_1 is used as follows:



R_1 / 512

Q_2

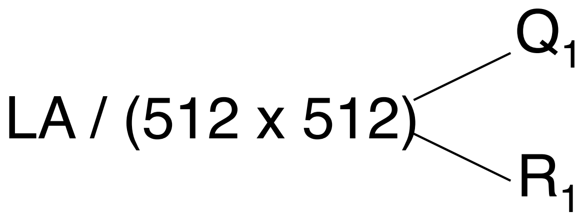
R_2

Q_2 = displacement into block of index table

R_2 displacement into block of file

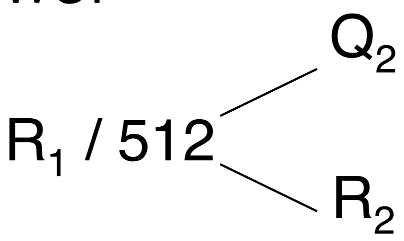
Indexed Allocation – Mapping (Cont.)

- Two-level index (maximum file size is 512^3)



Q_1 = displacement into outer-index

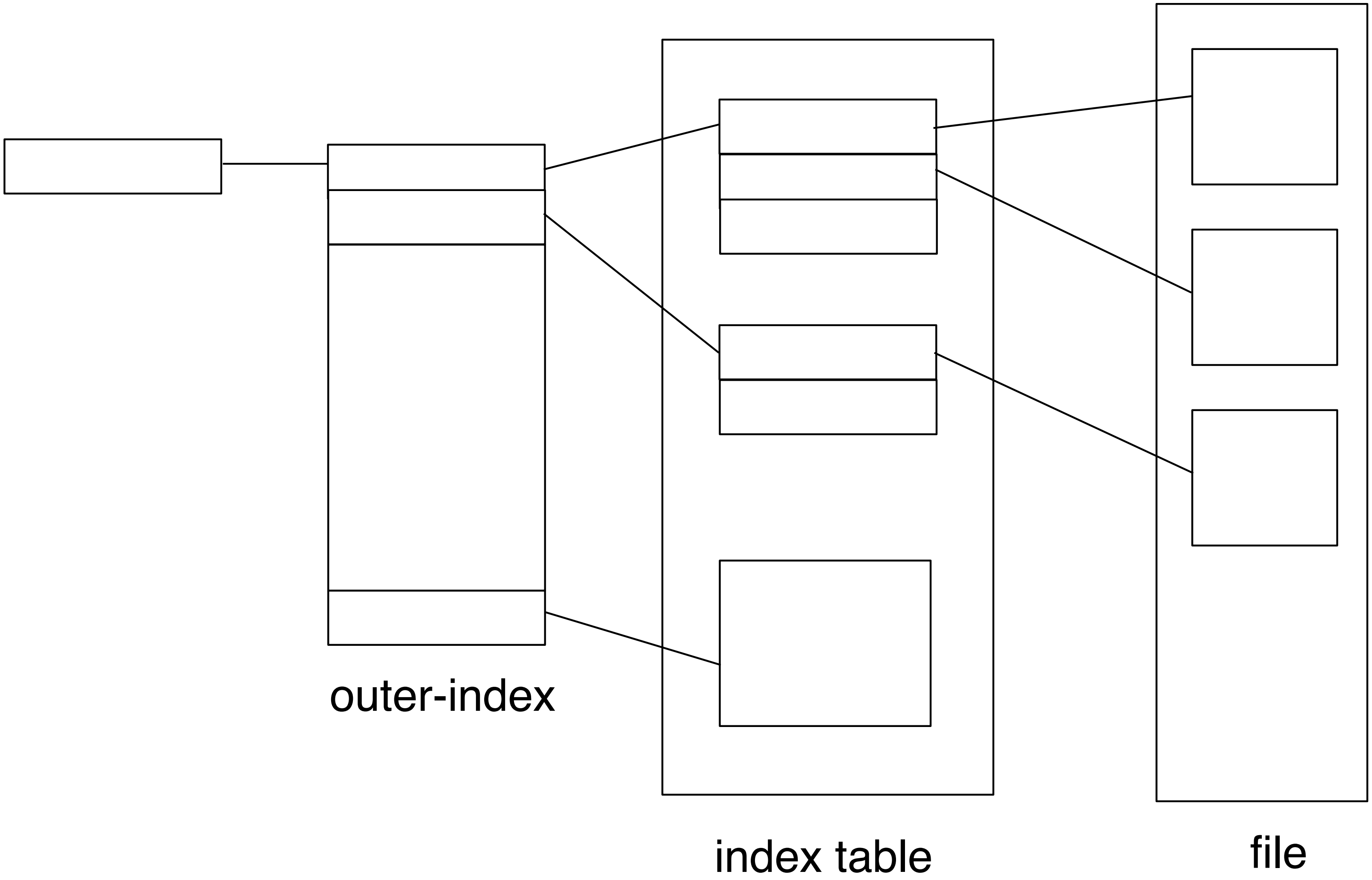
R_1 is used as follows:



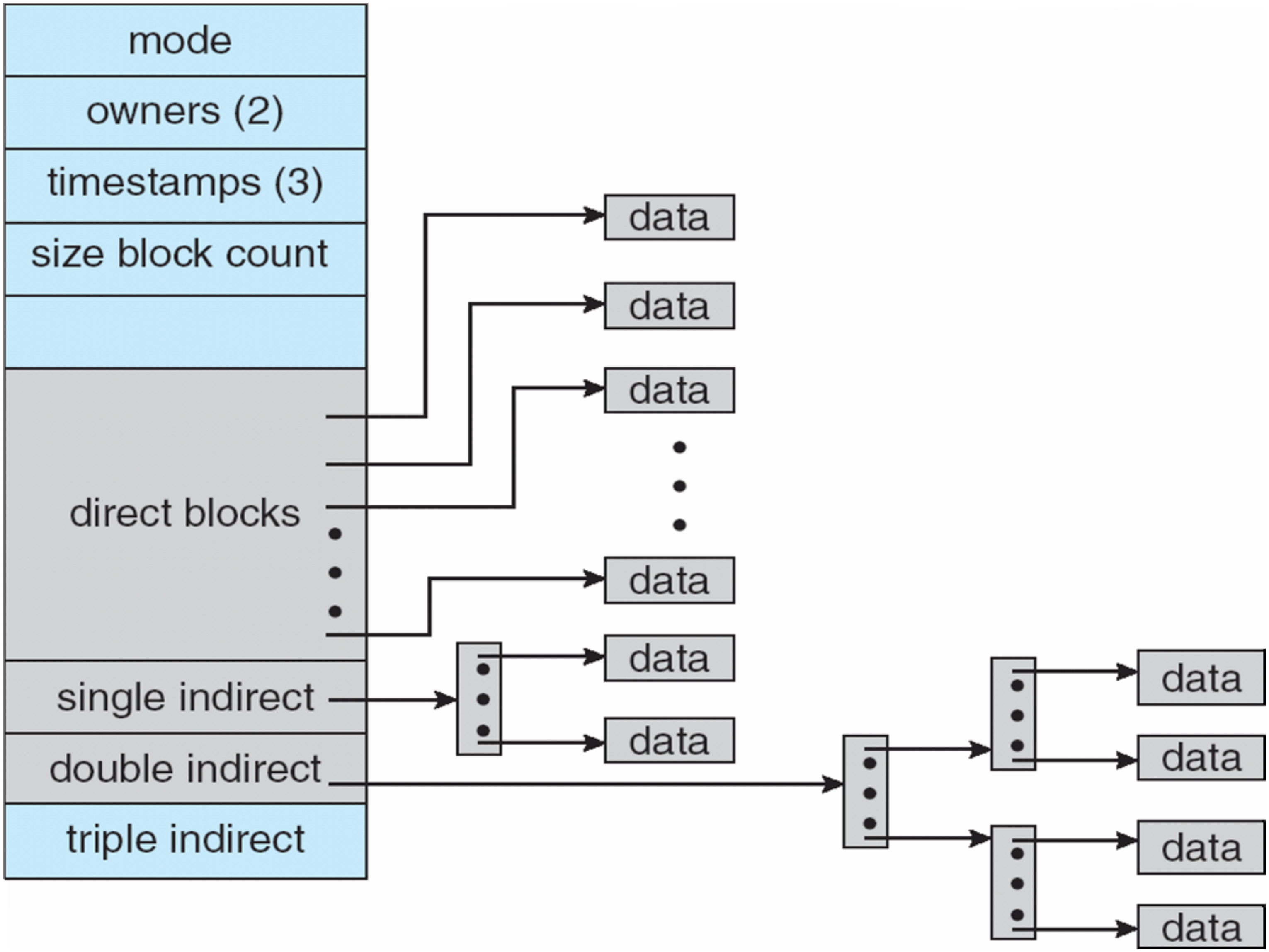
Q_2 = displacement into block of index table

R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)



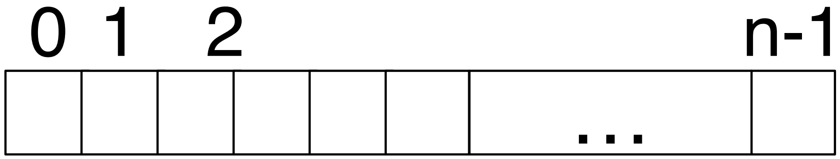
Combined Scheme: UNIX UFS (4K bytes per block)



Silberschatz, Galvin and Gagne ©2009

Free-Space Management

- Bit vector (*n* blocks)



0 1 2 n-1

…

bit[*i*] =

0 ⇒ block[*i*] free

1 ⇒ block[*i*] occupied

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

Free-Space Management (Cont.)

- Bit map requires extra space
    - Example:
- block size = 2^{12} bytes
- disk size = 2^{30} bytes (1 gigabyte)
- $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
    - Cannot get contiguous space easily
    - No waste of space
- Grouping
- Counting

Free-Space Management (Cont.)

- Need to protect:
    - Pointer to free list
    - Bit map
        - Must be kept on disk
        - Copy in memory and disk may differ
        - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
    - Solution:
        - Set bit[*i*] = 1 in disk
        - Allocate block[*i*]
        - Set bit[*i*] = 1 in memory

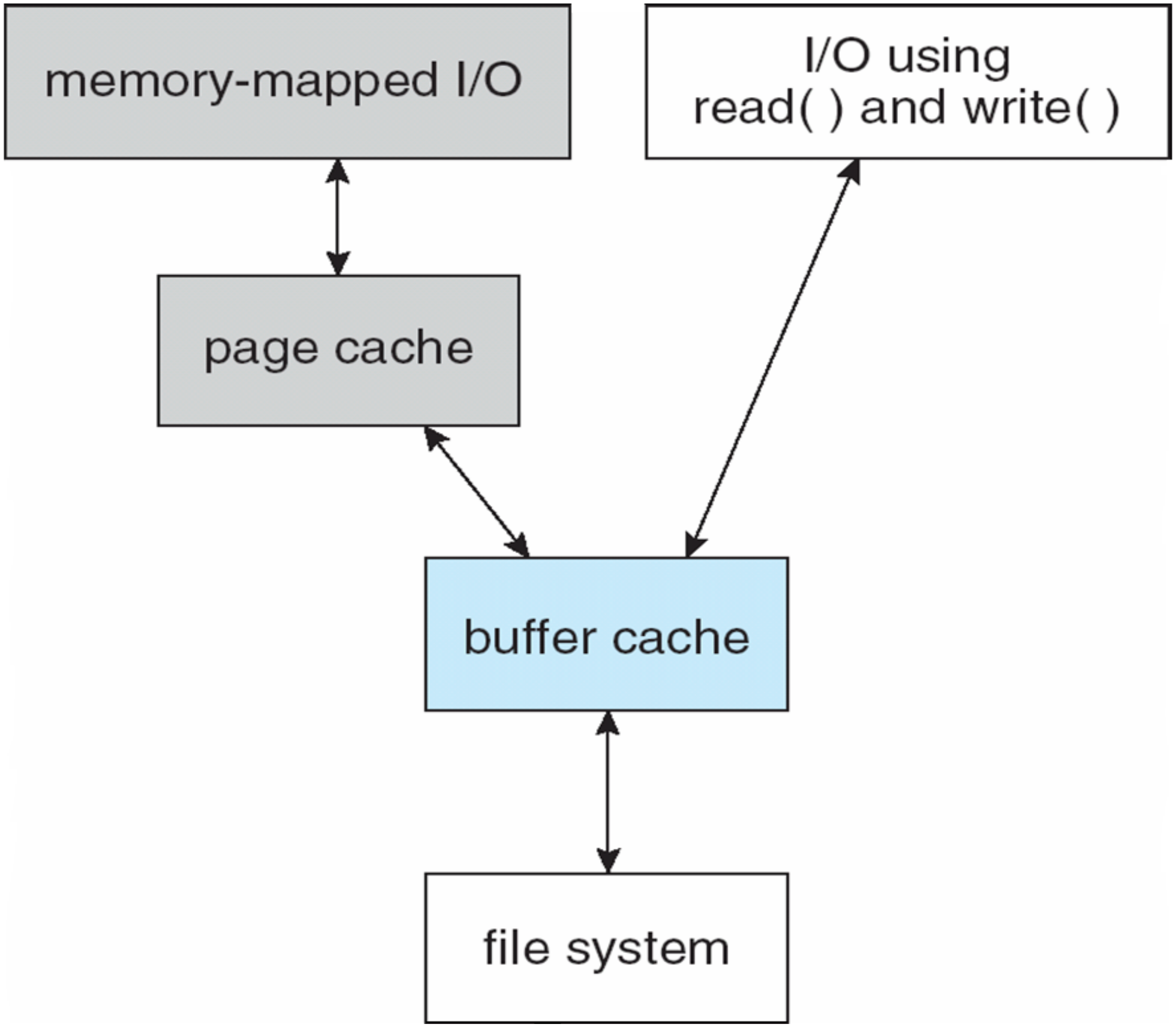
Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

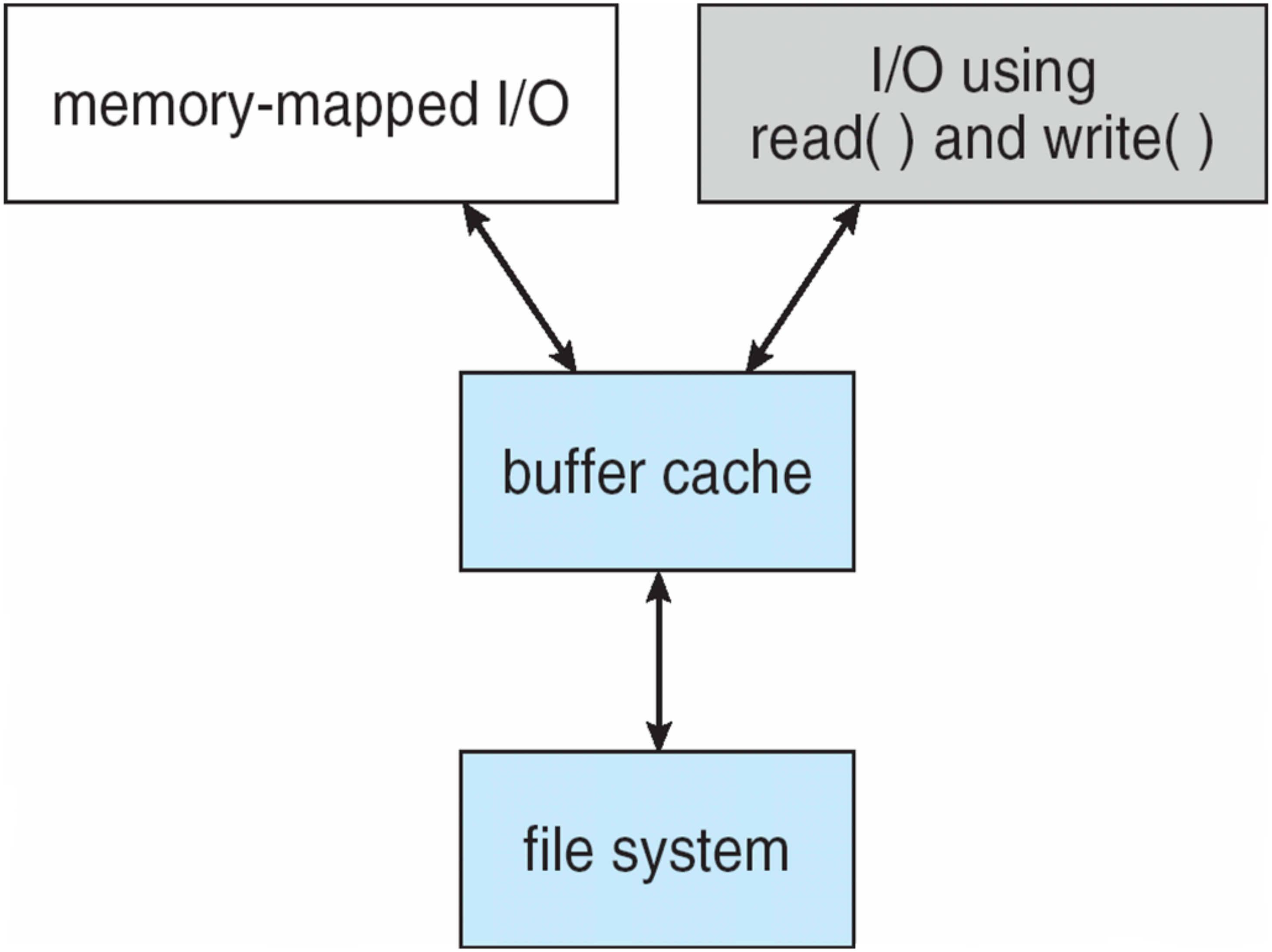
I/O Without a Unified Buffer Cache



Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O

I/O Using a Unified Buffer Cache



Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- 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 (full vs. incremental backups)

Log Structured File Systems

- **Log structured** (or **journaling**) file systems record each 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
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed

The Sun Network File System (NFS)

- 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

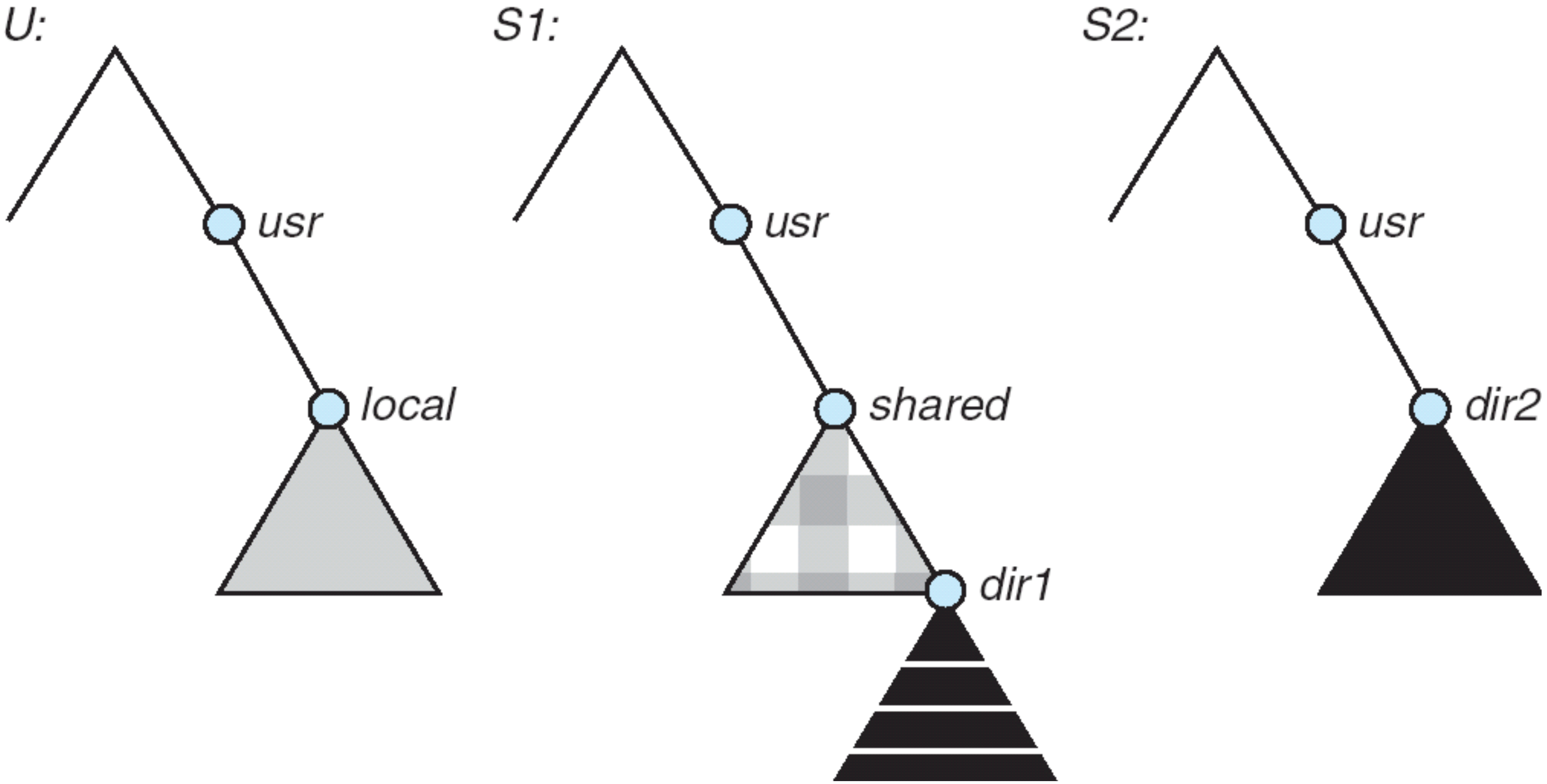
NFS (Cont.)

- 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
  - 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 (Cont.)

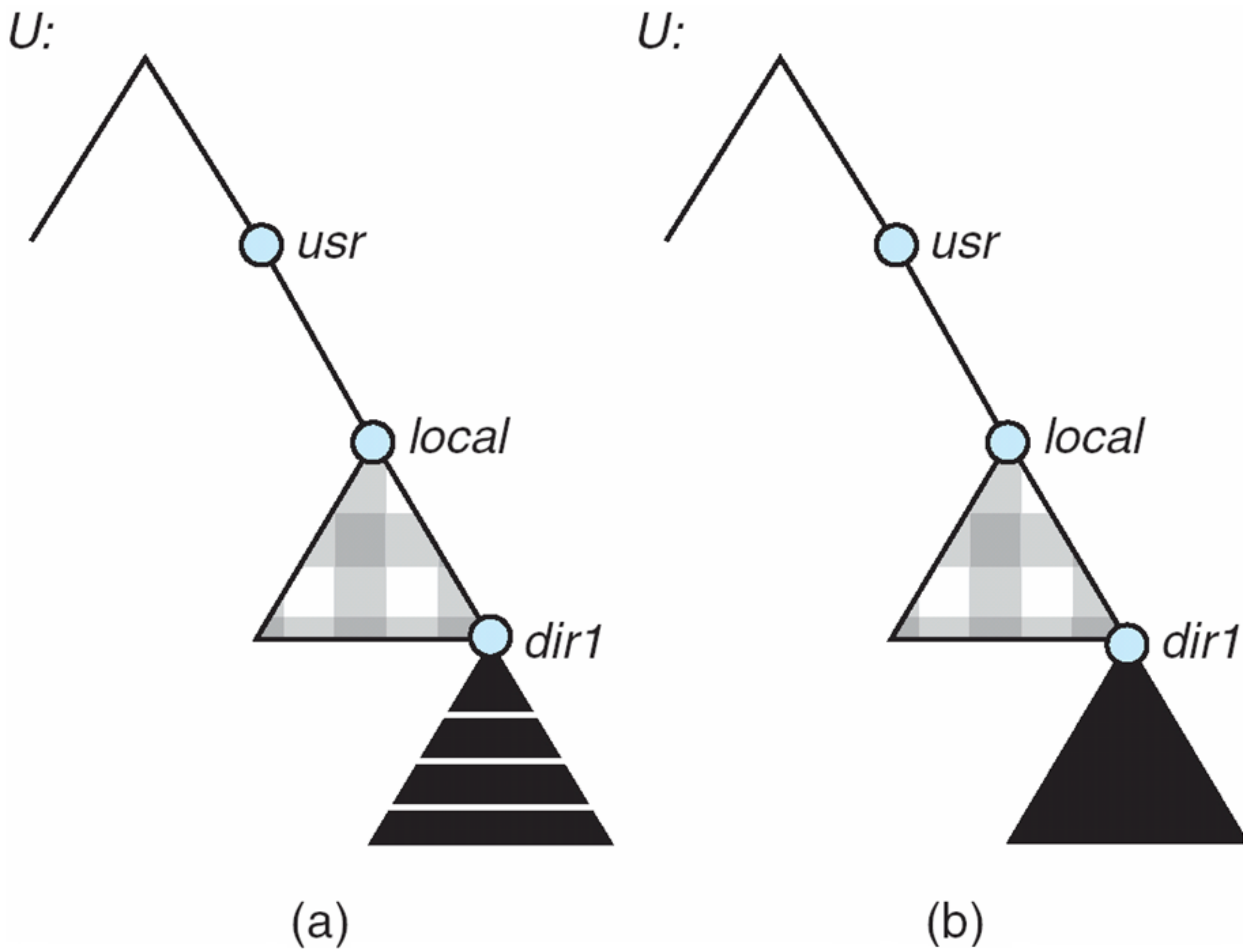
- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol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

Three Independent File Systems



Silberschatz, Galvin and Gagne ©2009

Mounting in NFS



Mounts

Cascading mounts

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
- 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

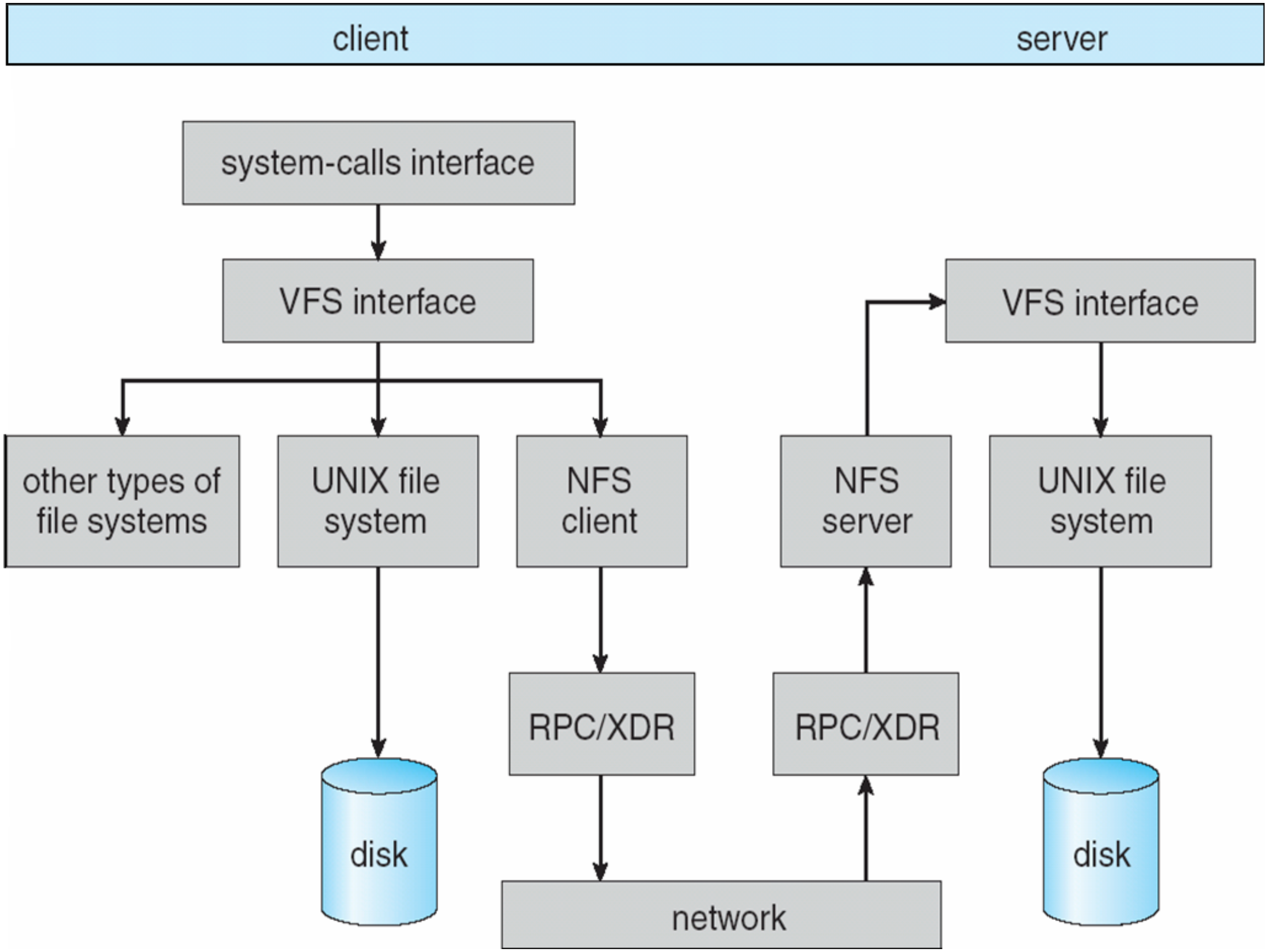
NFS Protocol

- 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 servers are **stateless**; each request has to provide a full set of arguments (NFS V4 is just coming available – very different, 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

- UNIX file-system interface (based on the **open, read, write**, and **close** calls, and **file descriptors**)
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- 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

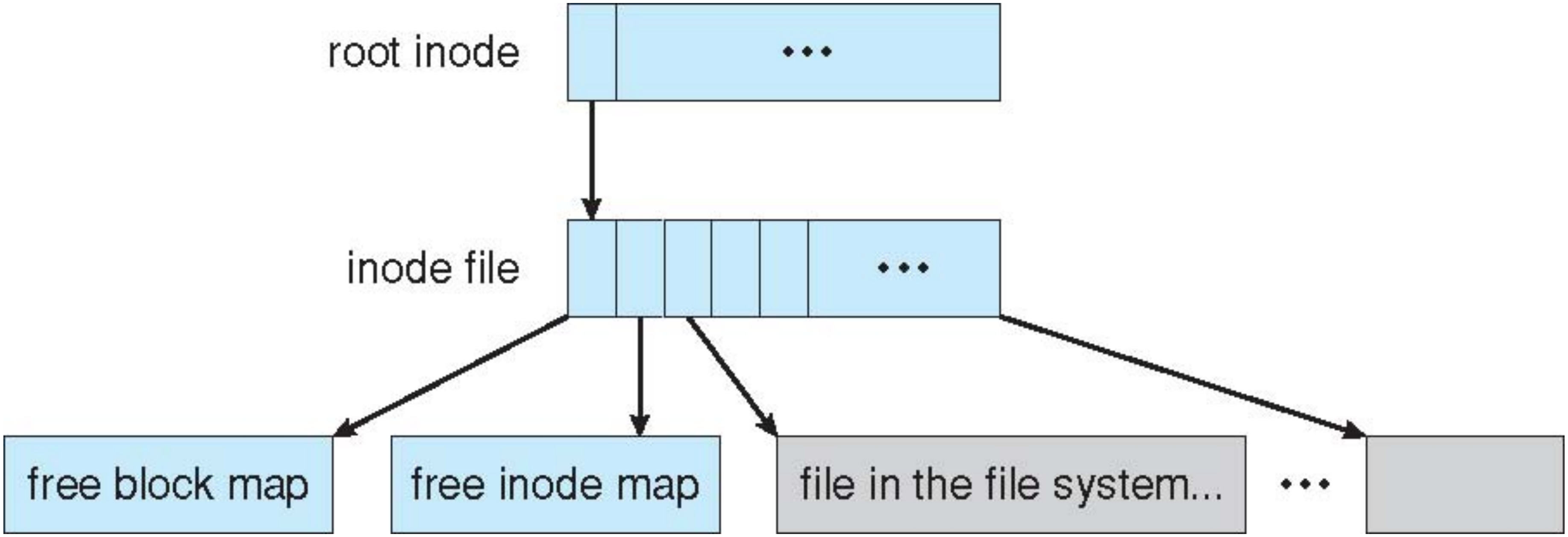
NFS Remote Operations

- 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
- 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

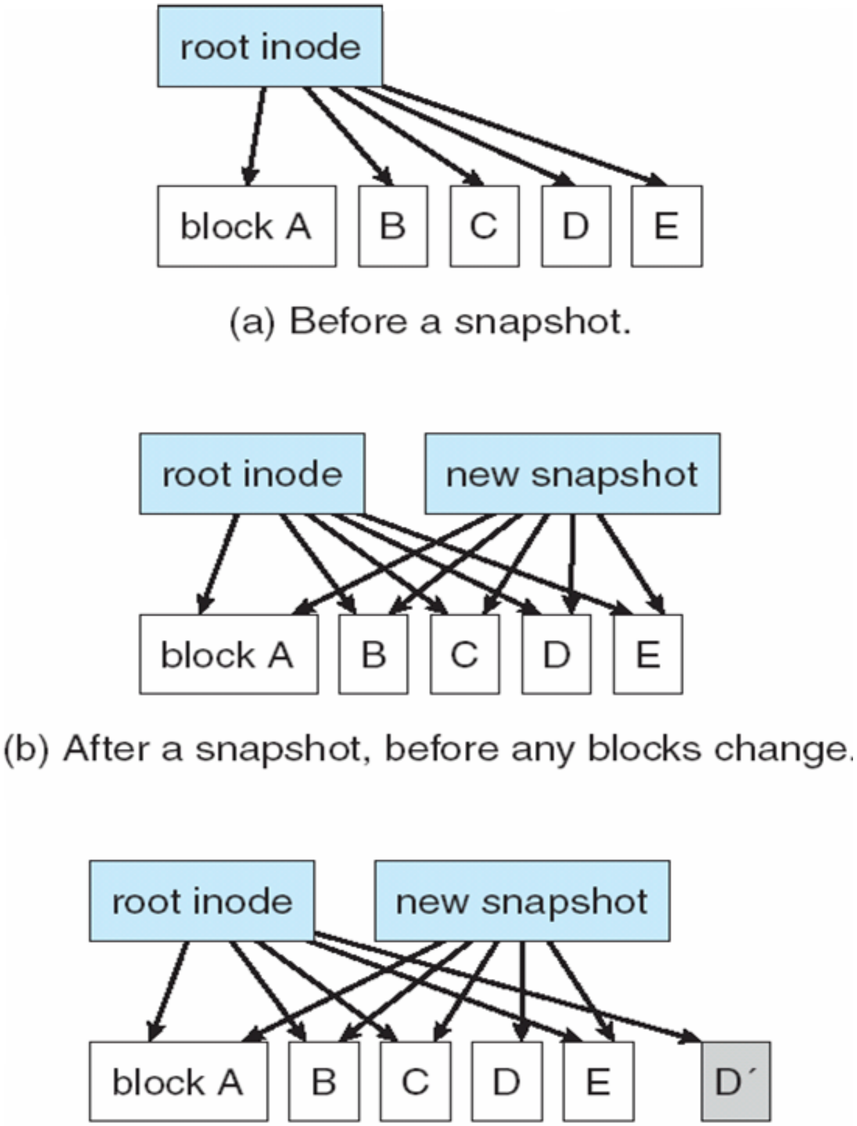
Example: WAFL File System

- 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
- Similar to Berkeley Fast File System, with extensive modifications

The WAFL File Layout



Snapshots in WAFL



(a) Before a snapshot.

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

(c) After block D has changed to D´.