1DT057
DISTRIBUTED INFORMATION SYSTEM

DISTRIBUTED FILE SYSTEM

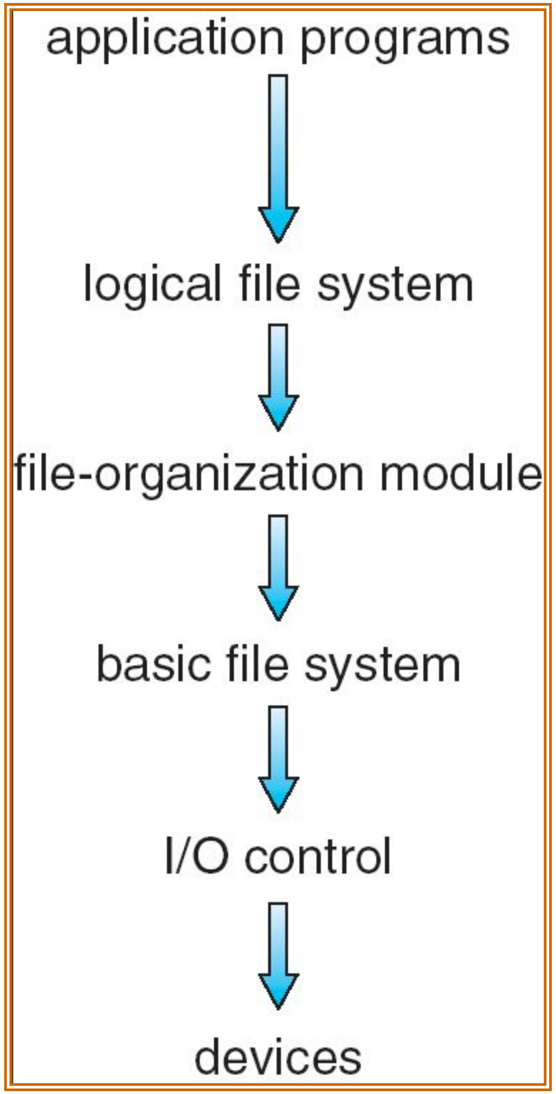
CHAPTER 8: DISTRIBUTED FILE SYSTEM

- Introduction to File System
  - File-System Structure
  - Directory Implementation
  - Allocation Methods
- Distributed File System
- Example: Sun NFS
- Example: AFS

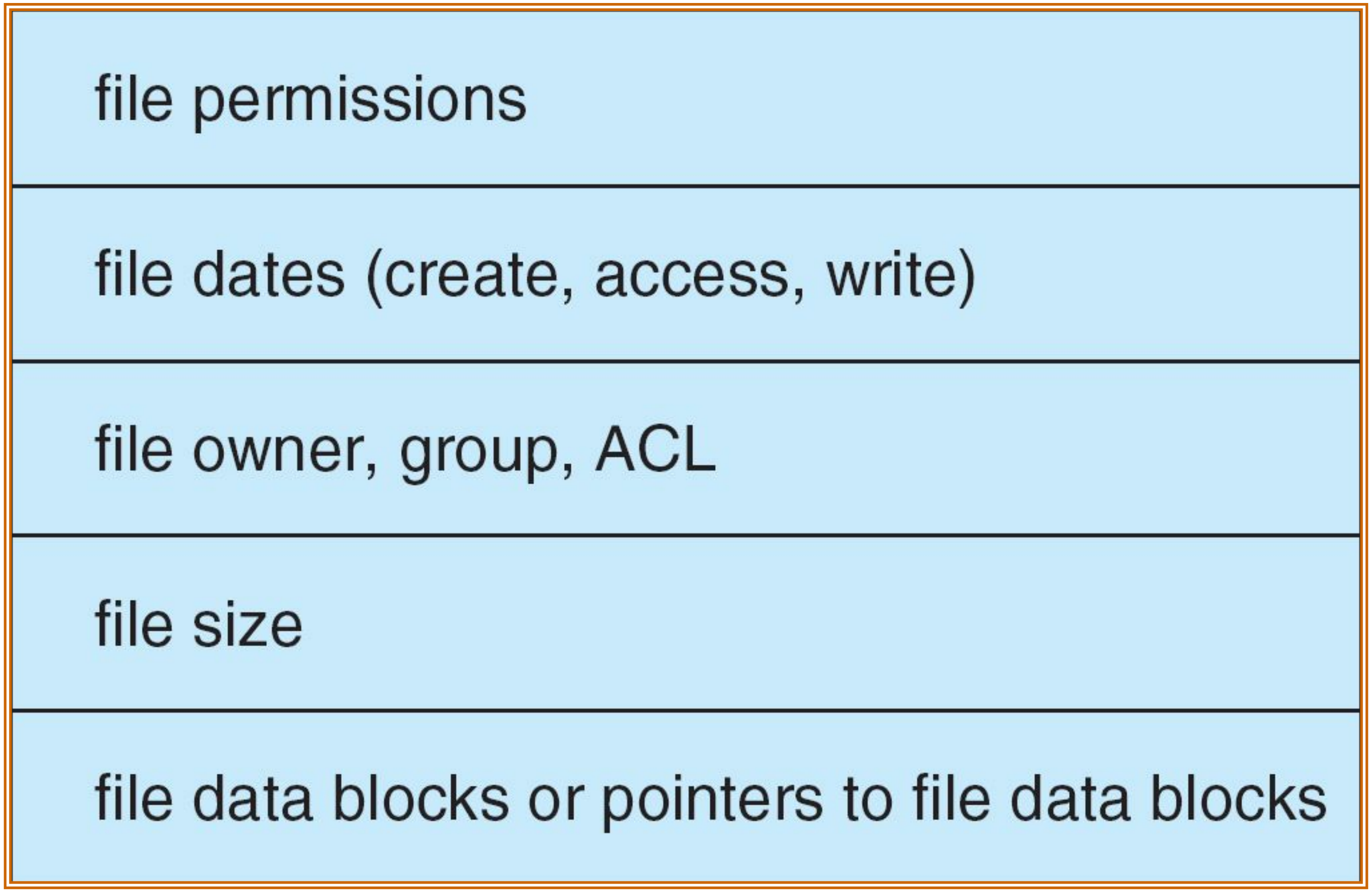
FILE-SYSTEM STRUCTURE

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File system organized into layers
- **File control block** – storage structure consisting of information about a file

LAYERED FILE SYSTEM



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

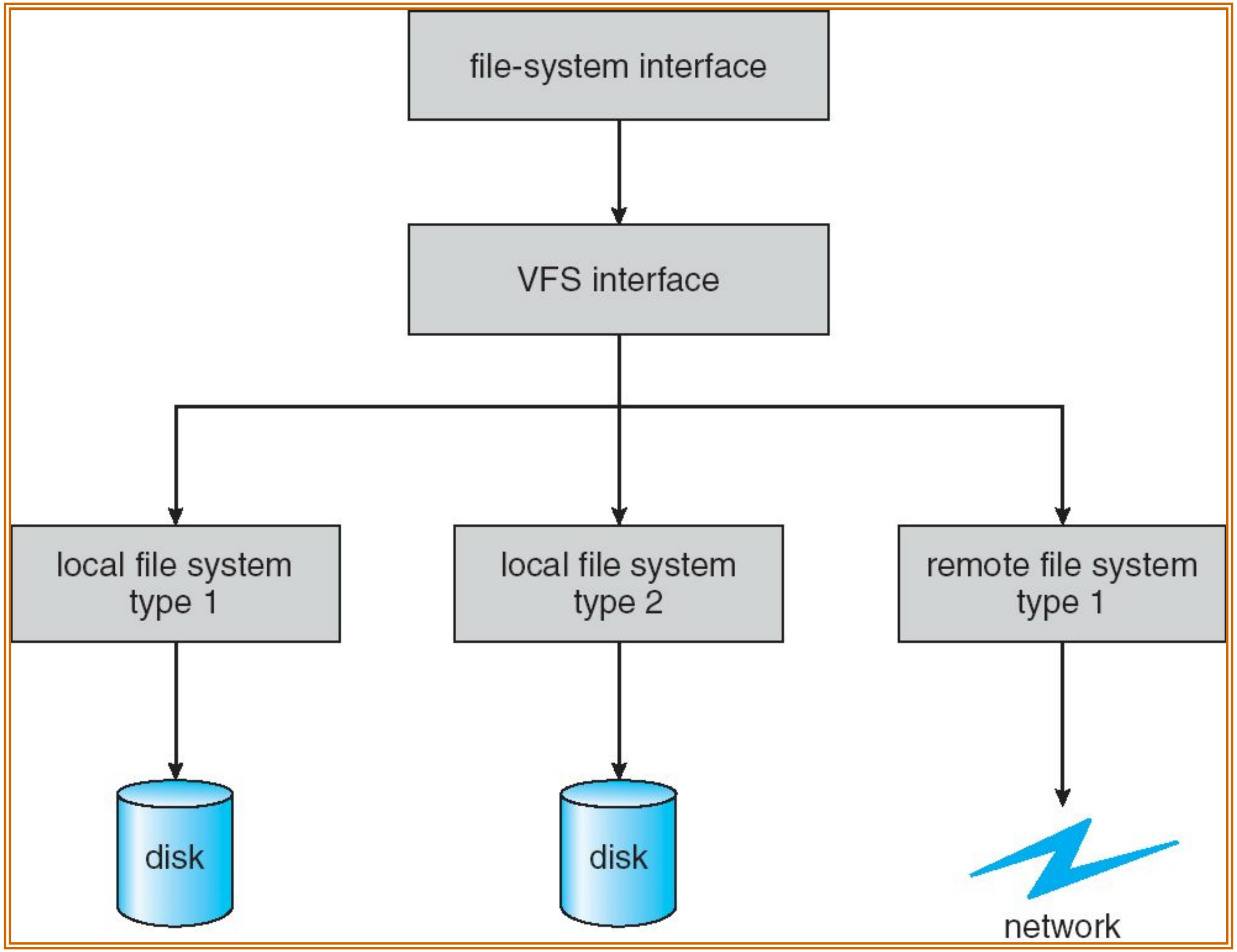
VIRTUAL FILE SYSTEMS

- 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



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

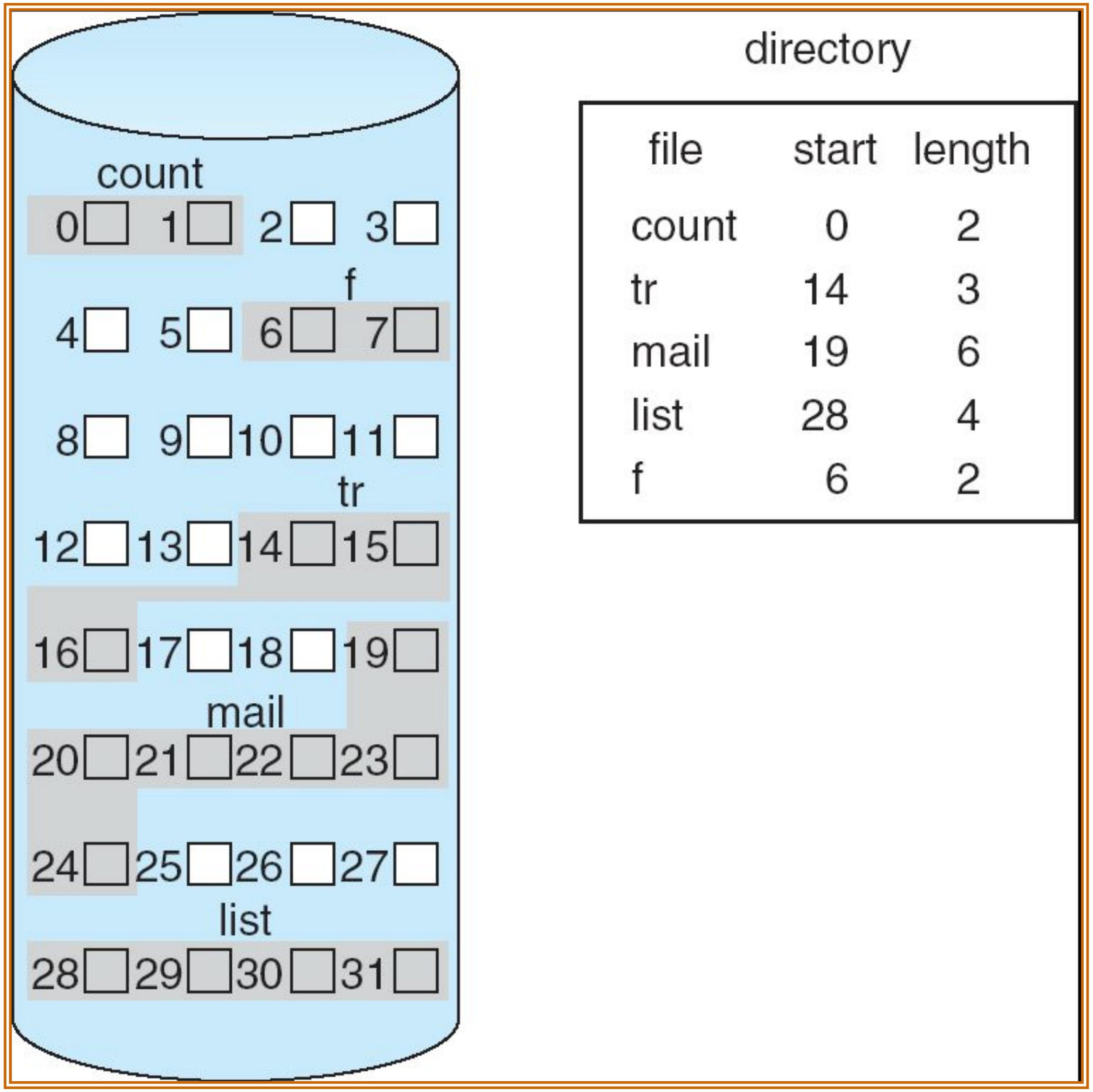
- Each file occupies a set of contiguous blocks on the disk

- Simple – only starting location (block #) and length (number of blocks) are required

- Wasteful of space (dynamic storage-allocation problem)

- Files cannot grow

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

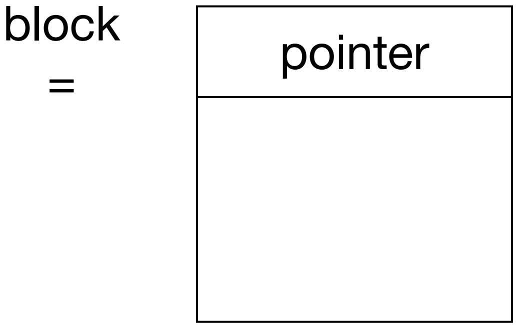
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme

- Extent-based file systems allocate disk blocks in **extents**

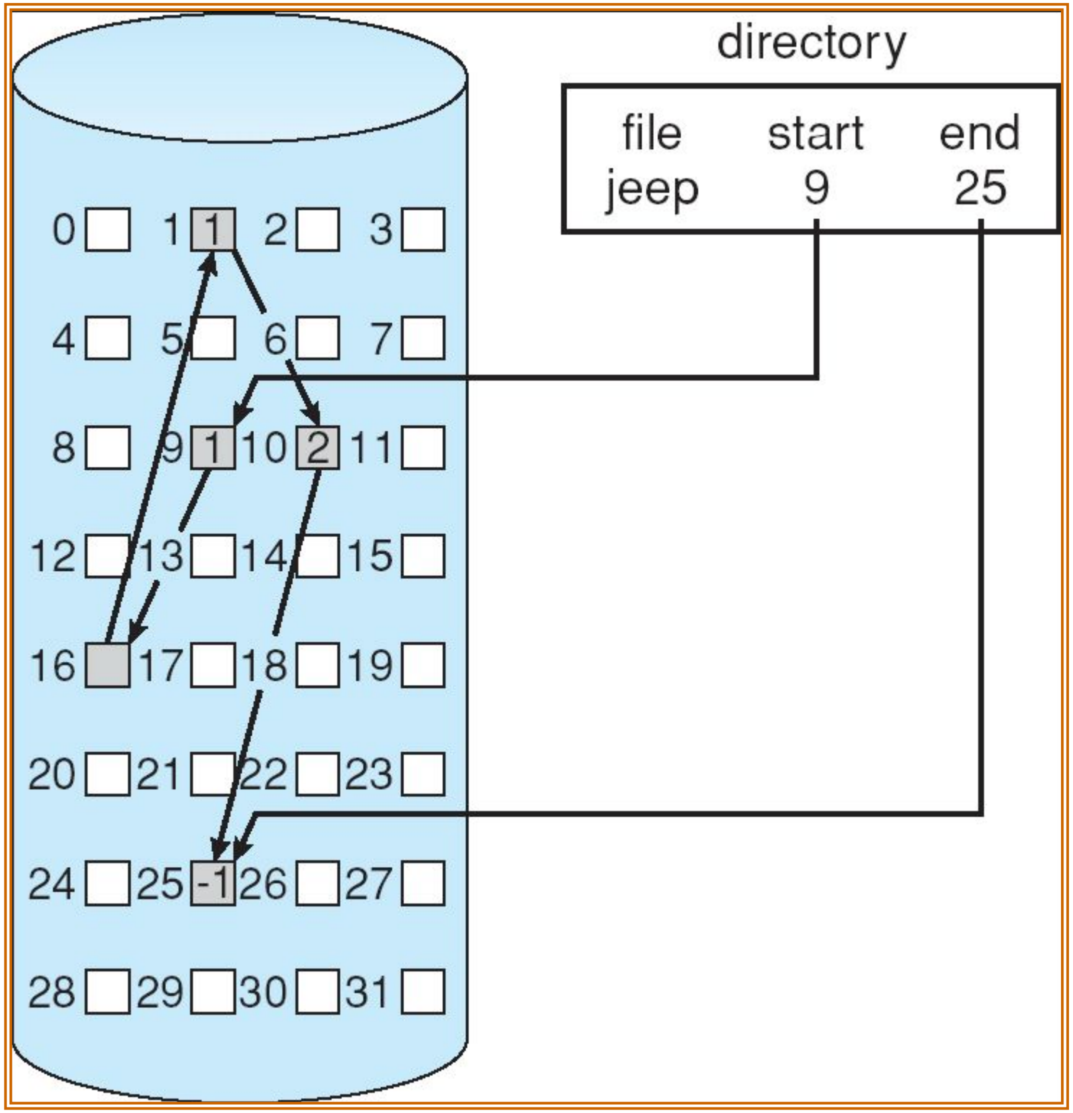
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents.

LINKED ALLOCATION

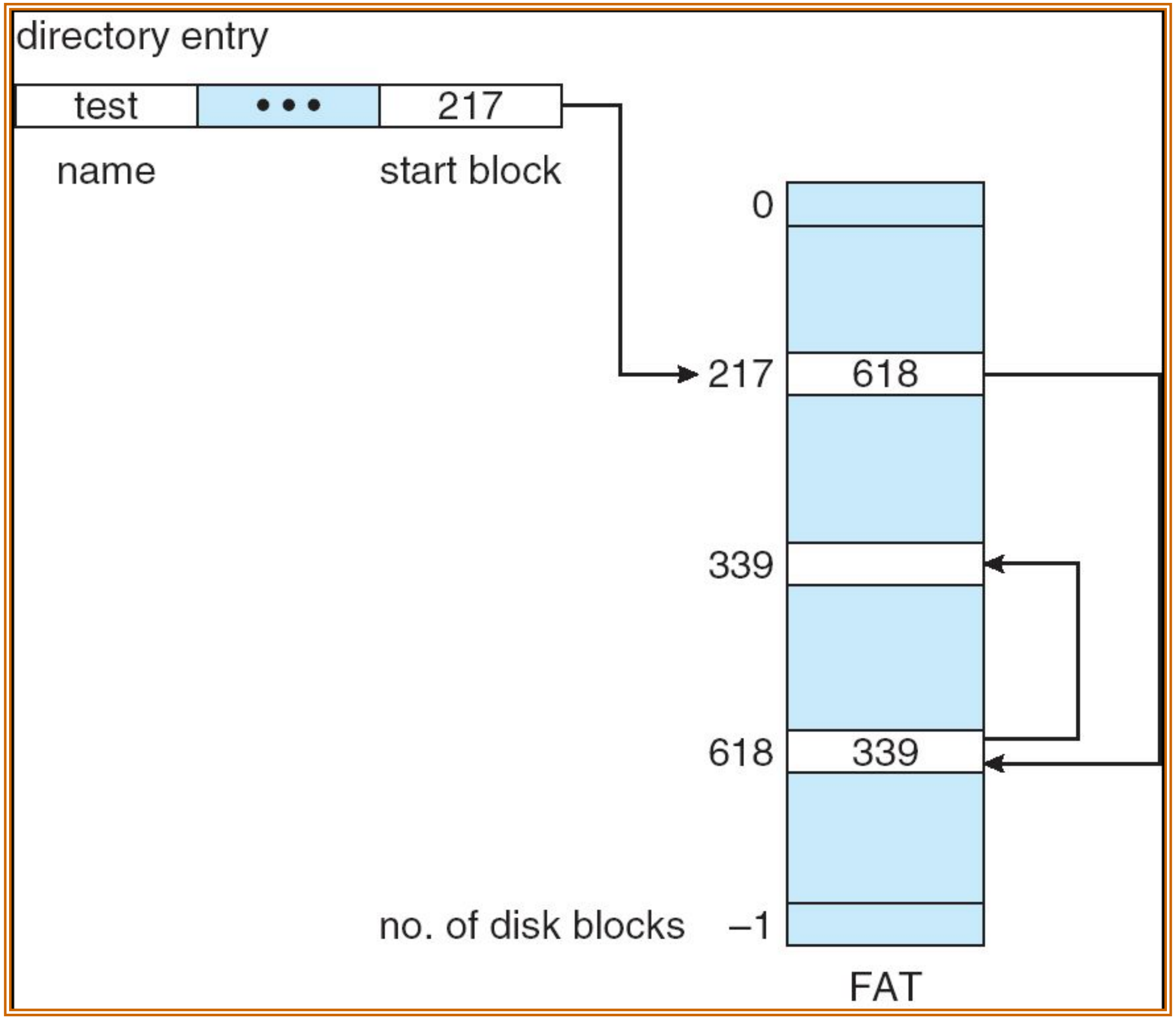
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



LINKED ALLOCATION

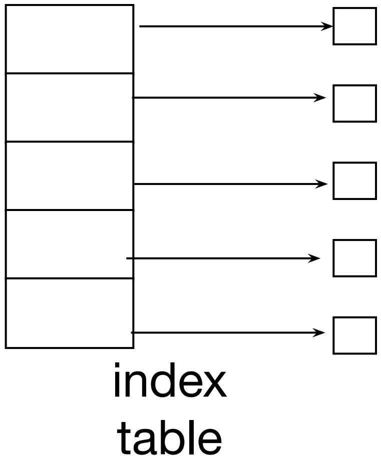


FILE-ALLOCATION TABLE

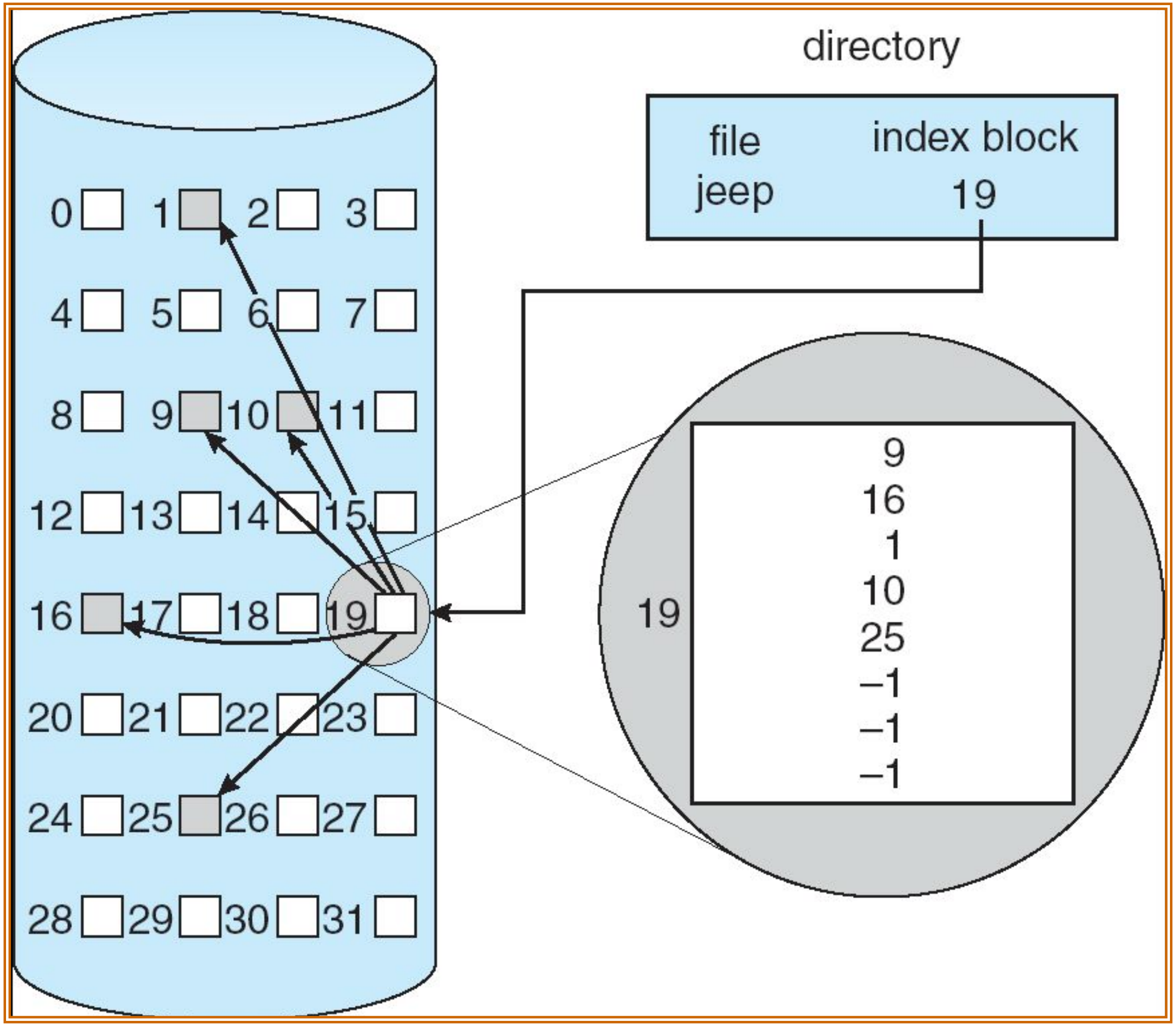


INDEXED ALLOCATION

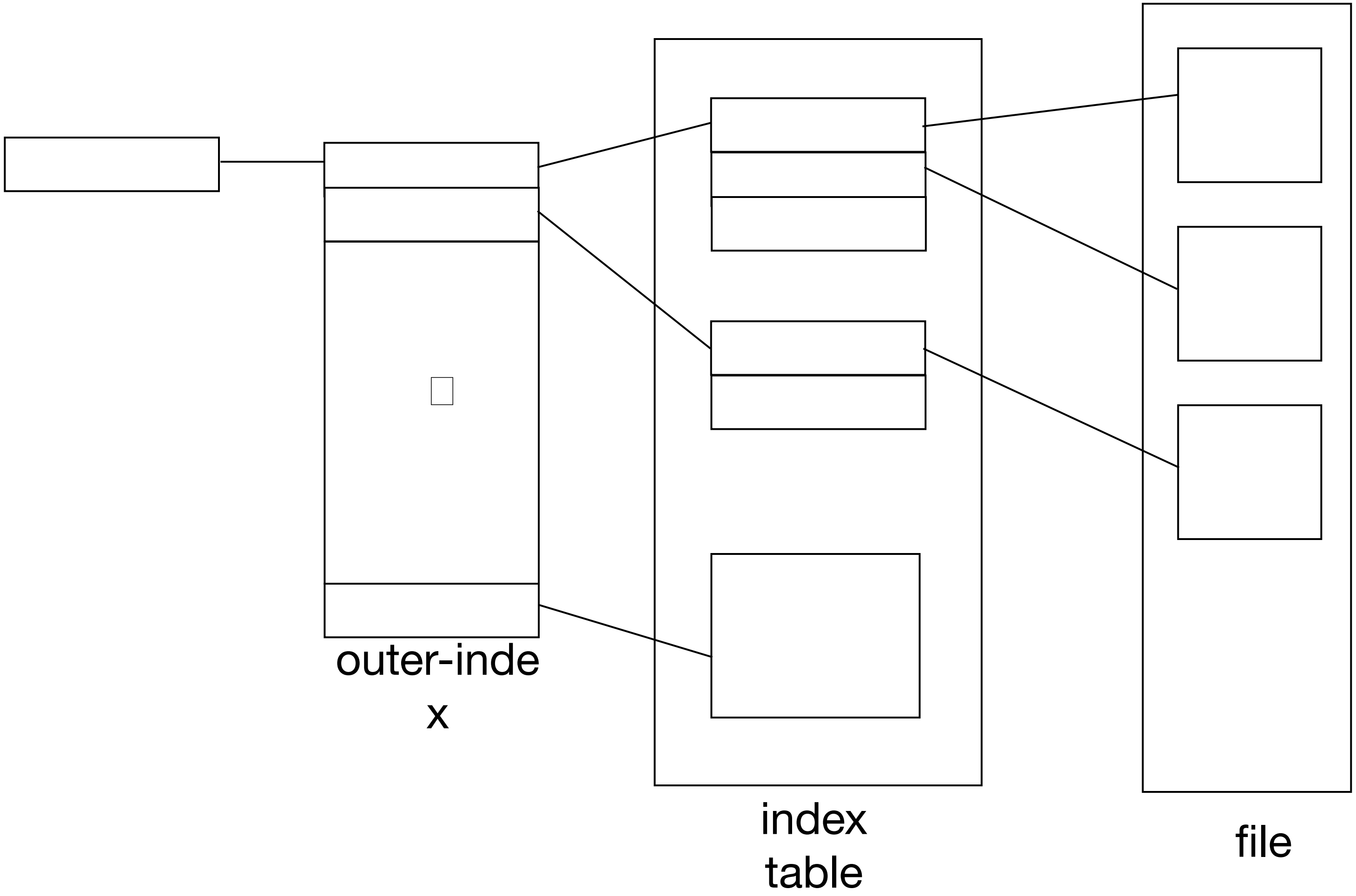
- Brings all pointers together into the *index block*.
- Logical view.



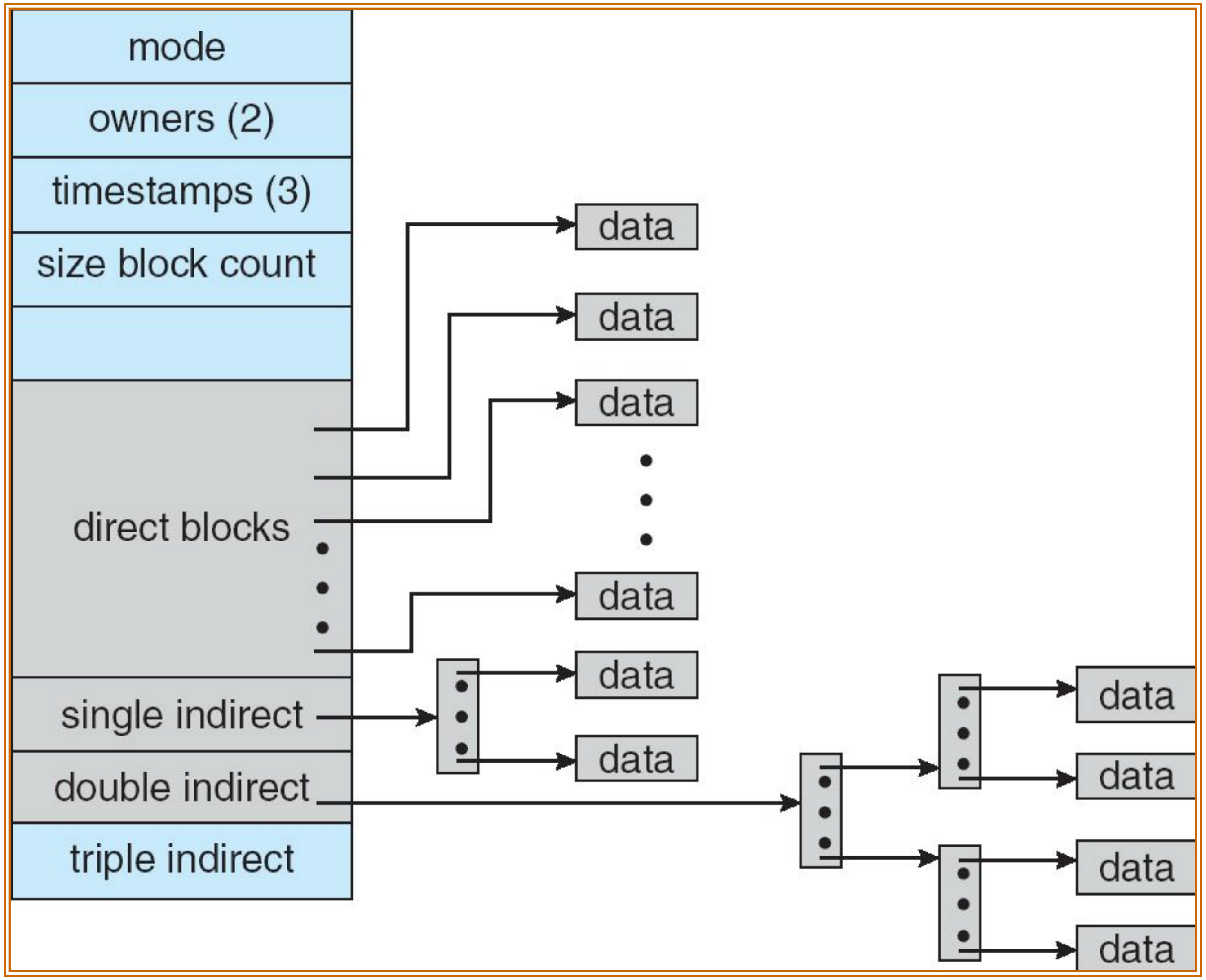
EXAMPLE OF INDEXED ALLOCATION



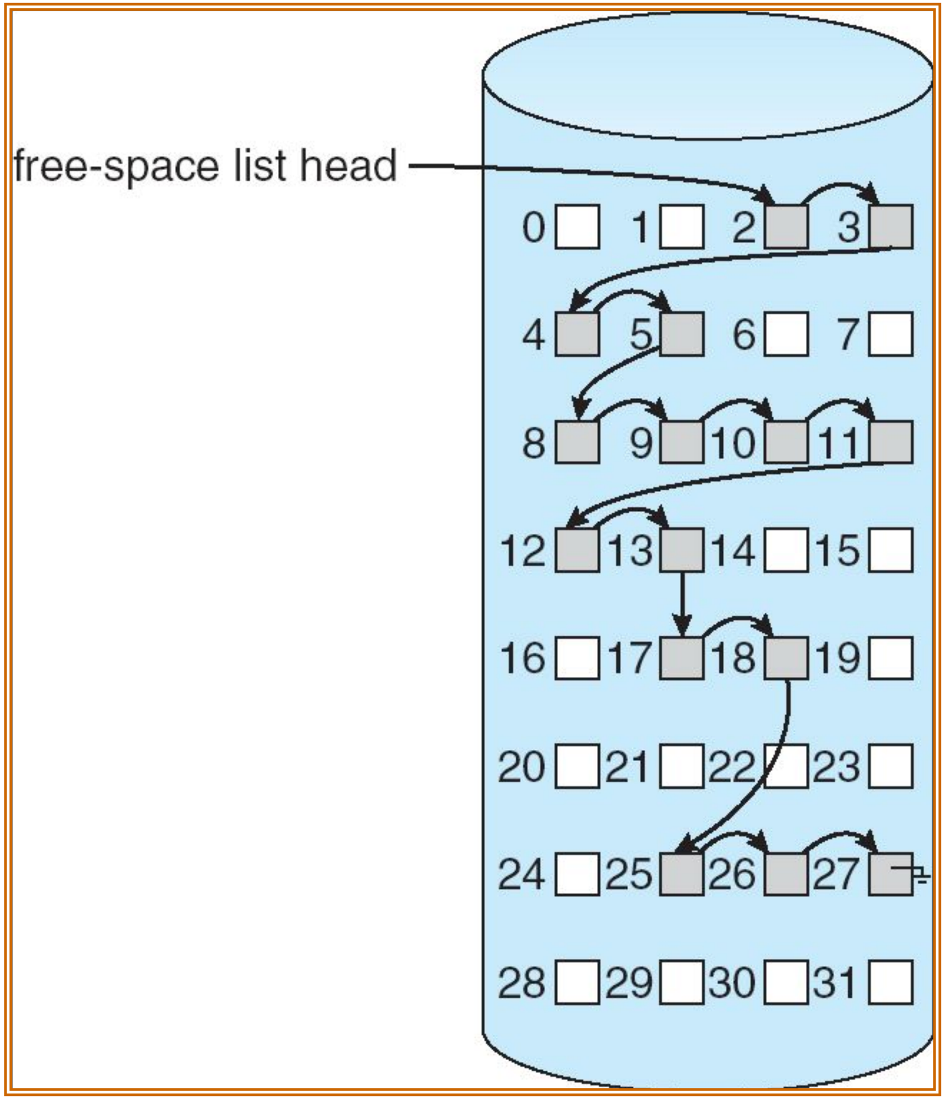
INDEXED ALLOCATION – MAPPING (CONT.)



COMBINED SCHEME: UNIX (4K BYTES PER BLOCK)



LINKED FREE SPACE LIST ON DISK



DISTRIBUTED FILE SYSTEM

DISTRIBUTED FILE SYSTEMS

- A special case of distributed system
- Allows multi-computer systems to share files
- Examples:
  - NFS (Sun's Network File System)
  - Windows NT, 2000, XP
  - Andrew File System (AFS) & others …

DISTRIBUTED FILE SYSTEMS (CONTINUED)

- One of most common uses of distributed computing

- *Goal:* provide common view of centralized file system, but distributed implementation.
  - Ability to open & update *any* file on any machine on network
  - All of synchronization issues and capabilities of shared local files

NAMING OF DISTRIBUTED FILES

- *Naming* – mapping between logical and physical objects.
- A *transparent* DFS hides the location where in the network the file is stored.
- **Location transparency** – file name does not reveal the file's physical storage location.
  - File name denotes a specific, hidden, set of physical disk blocks.
  - Convenient way to share data.
  - Could expose correspondence between component units and machines.
- **Location independence** – file name does not need to be changed when the file's physical storage location changes.
  - Better file abstraction.
  - Promotes sharing the storage space itself.
  - Separates the naming hierarchy from the storage-devices hierarchy.

DFS – THREE NAMING SCHEMES

1. *Mount* remote directories to local directories, giving the appearance of a coherent local directory tree
   - *Mounted* remote directories can be accessed transparently.
   - Unix/Linux with NFS; Windows with mapped drives
2. Files named by combination of *host name* and *local name*;
   - Guarantees a unique system wide name
   - Windows *Network Places*, Apollo Domain
3. Total integration of component file systems.
   - A single global name structure spans all the files in the system.
   - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable.

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

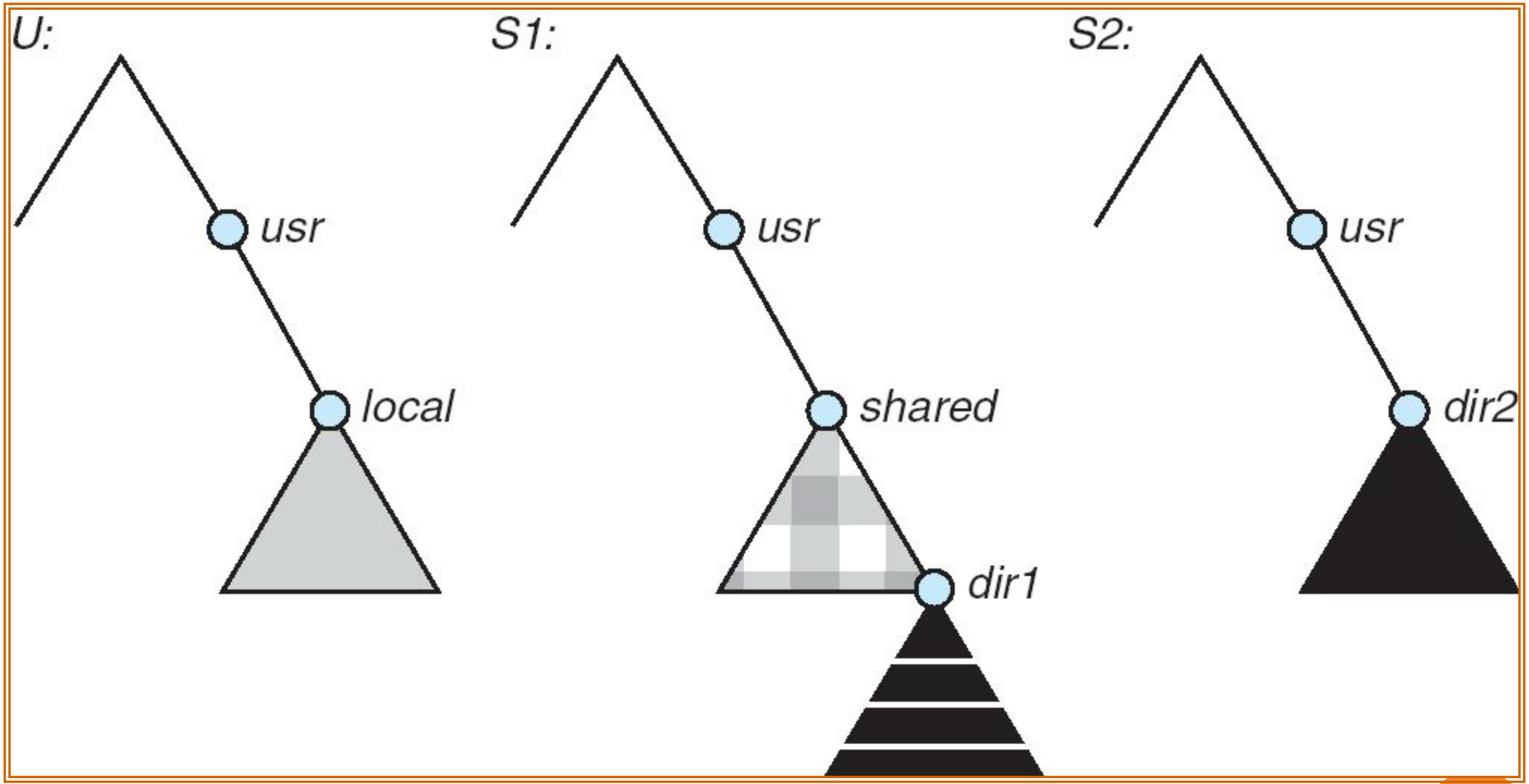
NFS (CONT.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media

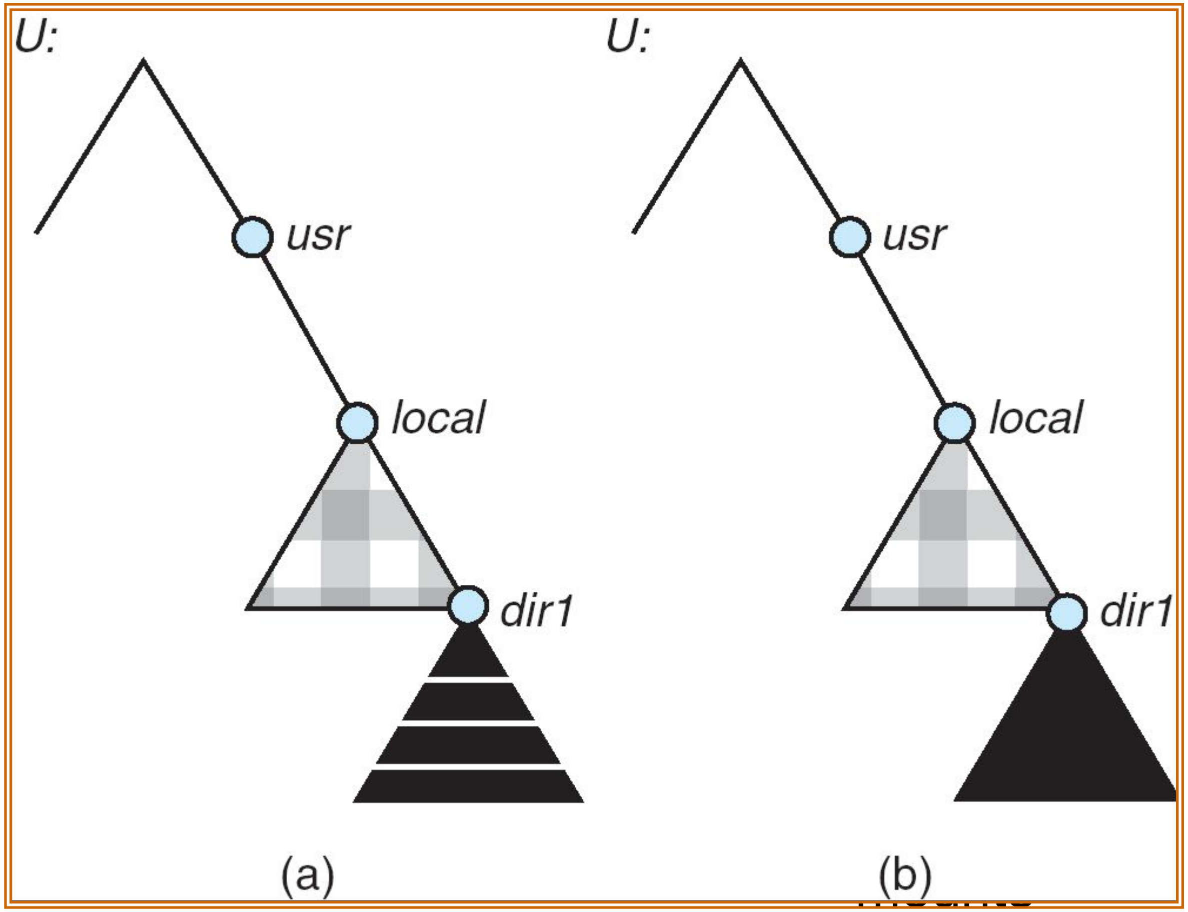
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces

- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

THREE INDEPENDENT FILE SYSTEMS



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
- 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

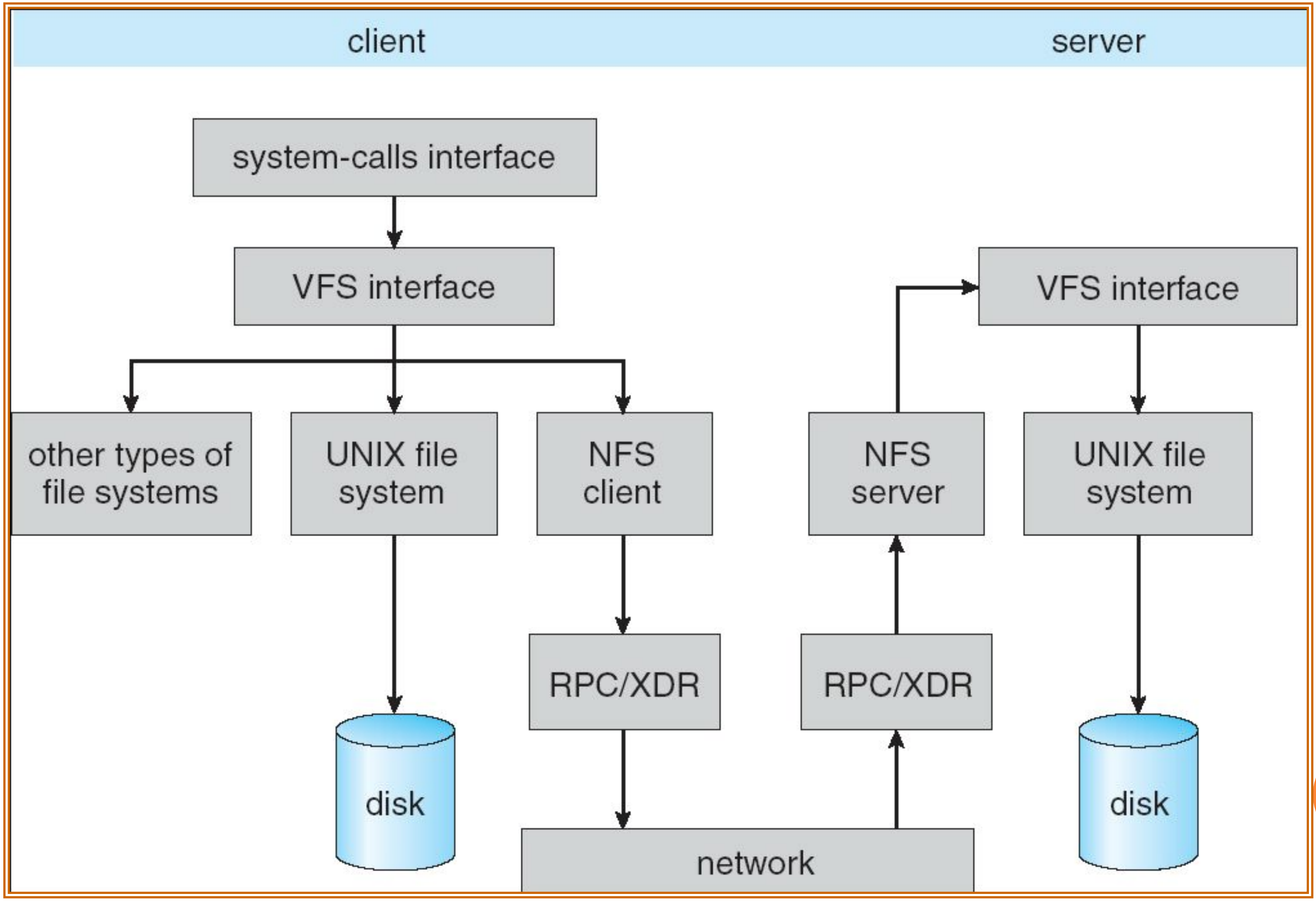
THREE MAJOR LAYERS OF NFS ARCHITECTURE

- 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

ANDREW FILE SYSTEM (AFS)

- Completely different kind of file system

- Developed at CMU to support all student computing.
- Consists of workstation clients and dedicated file server machines.

ANDREW FILE SYSTEM (AFS)

- Stateful
- Single name space
  - File has the same names everywhere in the world.
- Lots of local file caching
  - On workstation disks
  - For long periods of time
  - Originally whole files, now 64K file chunks.
- Good for distant operation because of local disk caching

AFS

- Need for scaling led to reduction of client-server message traffic.
  - Once a file is cached, all operations are performed locally.
  - On close, if the file is modified, it is replaced on the server.
- The client assumes that its cache is up to date!
- Server knows about all cached copies of file
  - *Callback* messages from the server saying otherwise.
- …

AFS

- On file *open*()
  - If client has received a callback for file, it must fetch new copy
  - Otherwise it uses its locally-cached copy.
- Server crashes
  - Transparent to client if file is locally cached
  - Server must contact clients to find state of files

DISTRIBUTED FILE SYSTEMS REQUIREMENTS

- *Performance* is always an issue
  - Tradeoff between performance and the semantics of file operations (especially for shared files).
- *Caching* of file blocks is crucial in any file system, distributed or otherwise.
  - As memories get larger, most read requests can be serviced out of file buffer cache (local memory).
  - Maintaining coherency of those caches is a crucial design issue.
- Current research addressing disconnected file operation for mobile computers.

SUMMERY

- Introduction to file system
- Characteristics of distributed file system
- Case study: Sun Network File System
- Case study: The Andrew File system

- Read chapter 8 [Coulouris et al.] after the lecture…