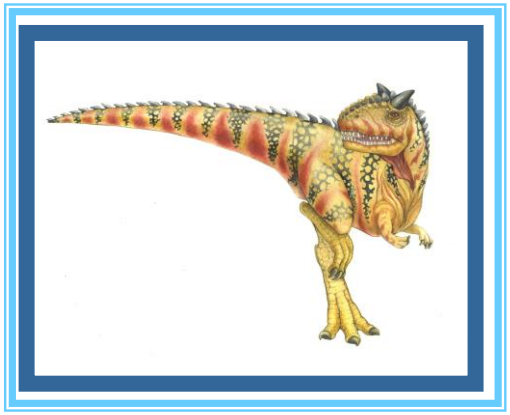
Chapter 11
Implementing File-Systems



Chapter 11: Implementing File Systems

File-System Structure

File-System Implementation

Directory Implementation

Allocation Methods

Free-Space Management

Efficiency and Performance

Recovery

Log-Structured File Systems

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

File-System Structure

File structure

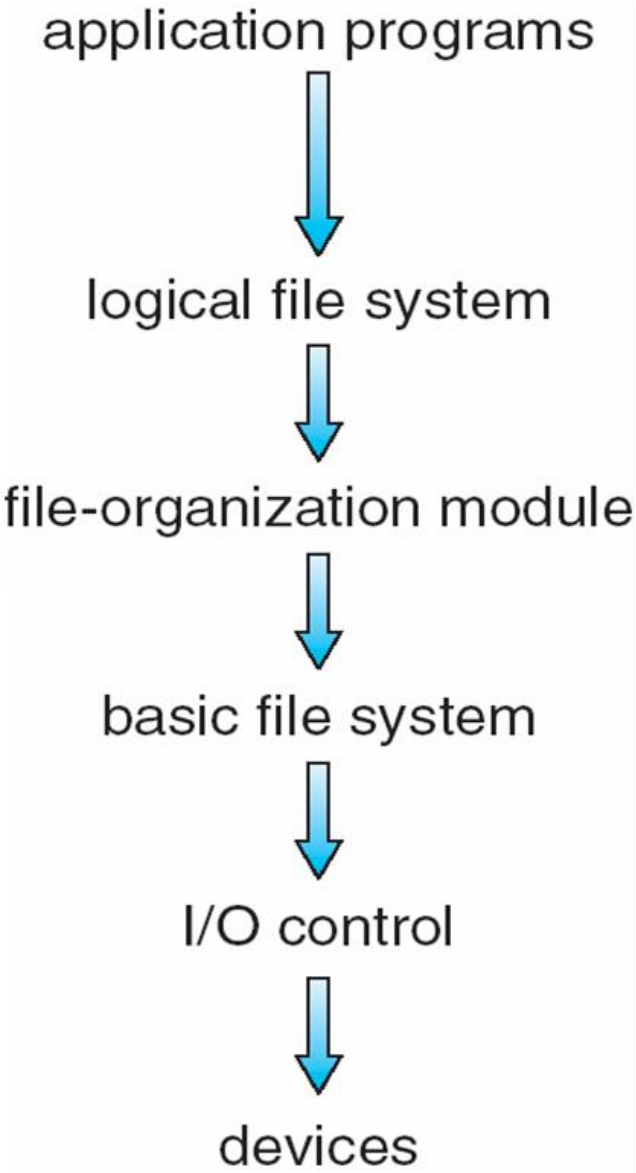
- Logical storage unit
- Collection of related information

File system resides on secondary storage (disks)

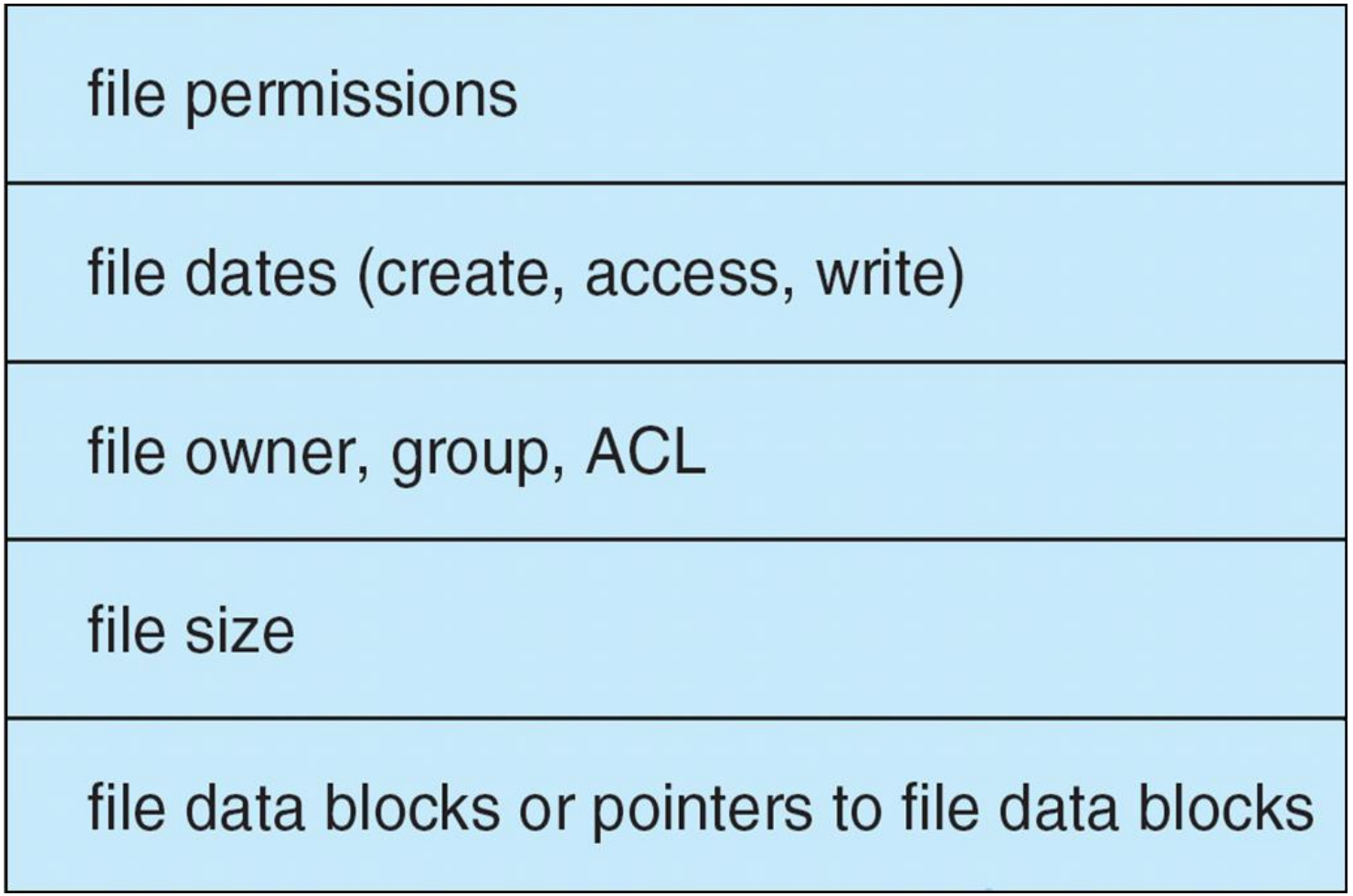
File system organized into layers

File control block – storage structure consisting of information about a file

Layered File System

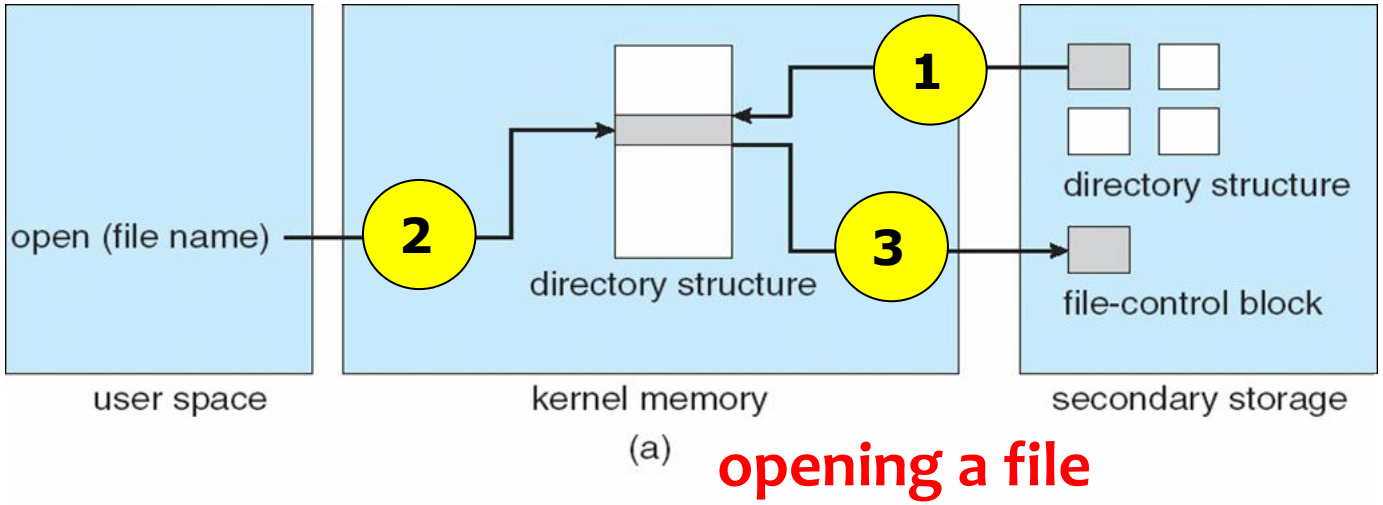


A Typical File Control Block

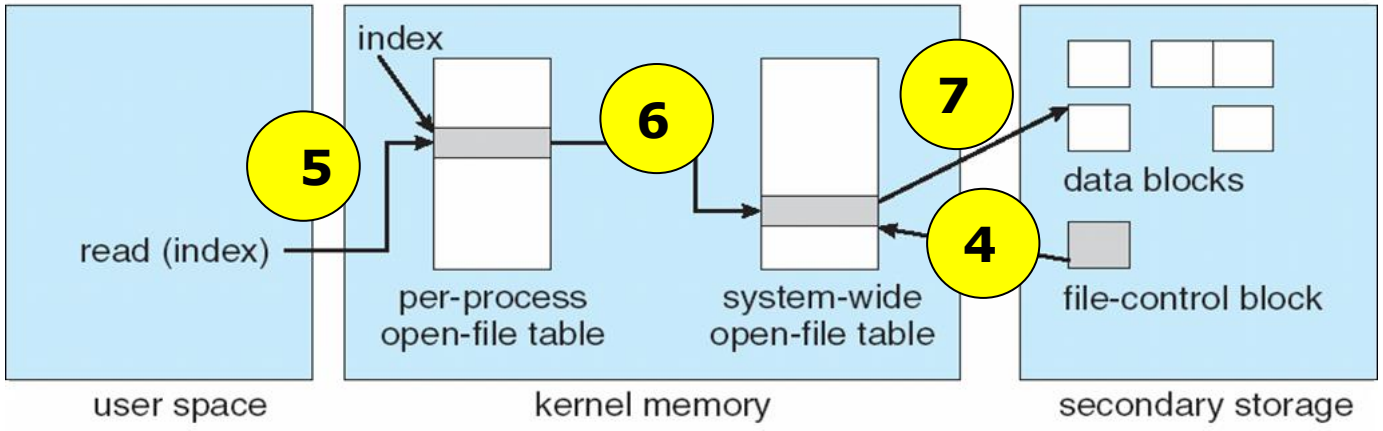


In-Memory File System Structures

The necessary file system structures provided by the OS.



(a) opening a file



(b) reading a file

Virtual File Systems

Modern OS must concurrently support multiple types of file systems.

How does an OS allow multiple types of file systems to be integrated into a directory structure ?

To write directory and file routines for each type.

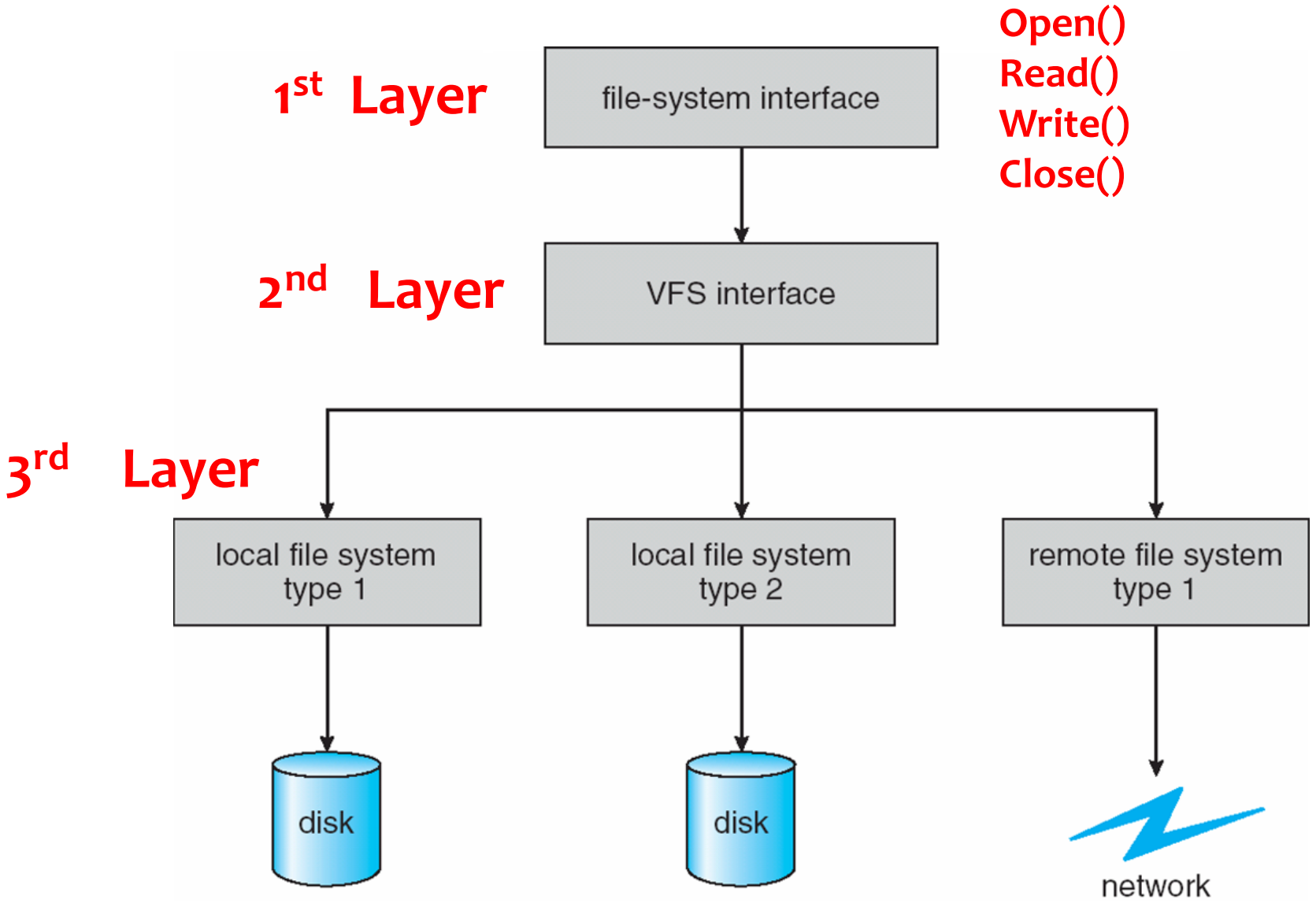
Most Operating systems provide an object-oriented way of implementing file systems, including UNIX.

The file-system implementation consists of three major layers (Figure 11.4)

The first layer is the file-system interface, based on open(), read(), write(), and close() calls

The 2nd layer is called the virtual file system (VFS) layer

Schematic View of a Virtual File System



Virtual File Systems

The VFS serves two important functions

It separates file-system-generic operations from their implementation by defining a clean VFS interface.

It provides a mechanism for uniquely representing a file throughout a network.

- The VFS is based on a file-representation structure, called a vnode, that contains numerical designator for a network-wild unique file.

Virtual File Systems

- Unix inodes are unique only within a single file system
- The kernel maintains one vnode structure for each active node (file or directory)

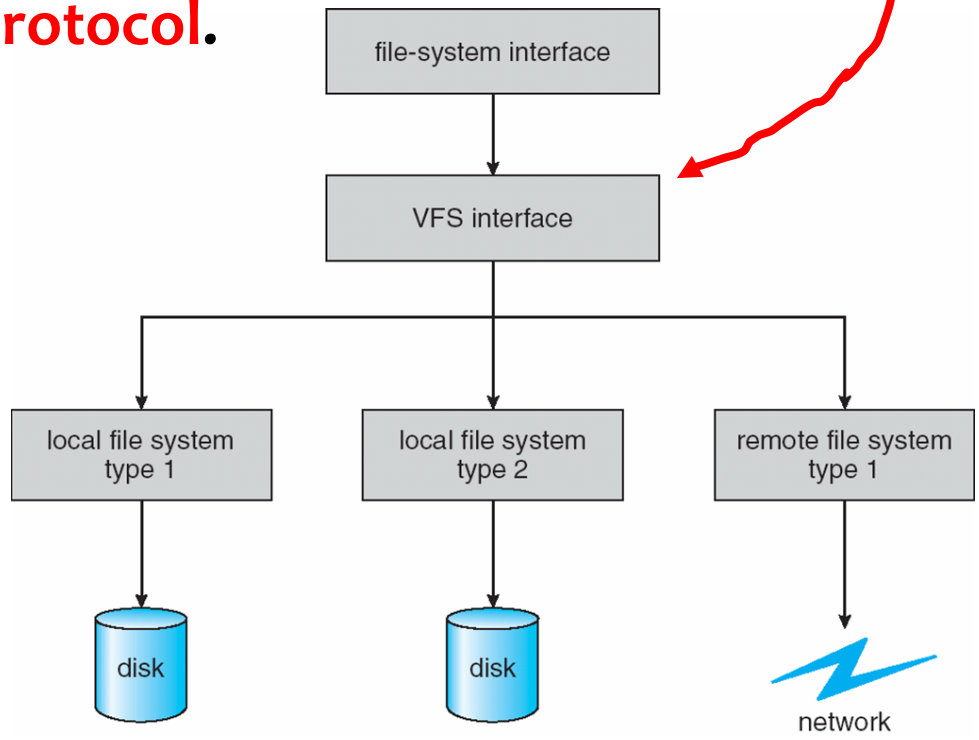
The VFS distinguishes local files from remote files, and local files are further distinguished according to their file-system types.

Virtual 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.

The 3rd layer implements the file-system type or the remote-file-system protocol.



Virtual File Systems

The VFS architecture in Linux defines four major object types

- The inode object, represents an individual file
- The file object, represents an open file
- The Superblock object, represents an entire file system
- The dentry object, represents an individual directory entry.

Directory Implementation

The selection of directory-allocation and directory-management algorithms affects the efficiency, performance, and reliability of the file system

Linear list of file names with pointer to the data blocks.

- simple to program
- time-consuming to execute

Hash Table – linear list stores the directory entries, but a hash data structure is also used.

- decreases directory search time
- collisions – situations where two file names hash to the same location
- fixed size

Allocation Methods

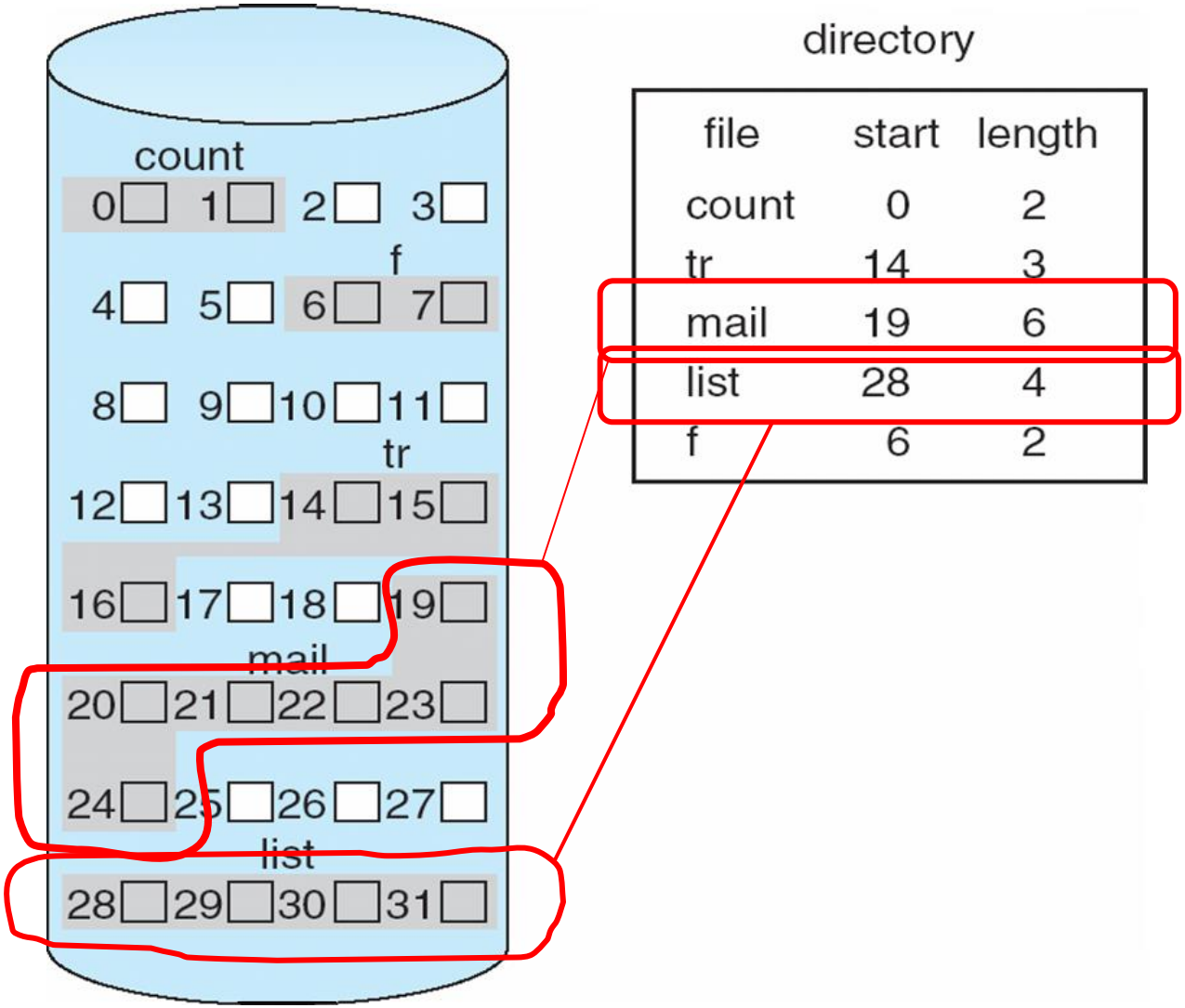
An allocation method refers to how disk blocks are allocated for files:

- 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), external fragmentation
- Files cannot grow

Contiguous Allocation of Disk Space



Extent-Based Systems

Many newer file systems use a modified contiguous allocation scheme

Extent-based file systems allocate disk blocks in extents

An extent is a contiguous block of disks

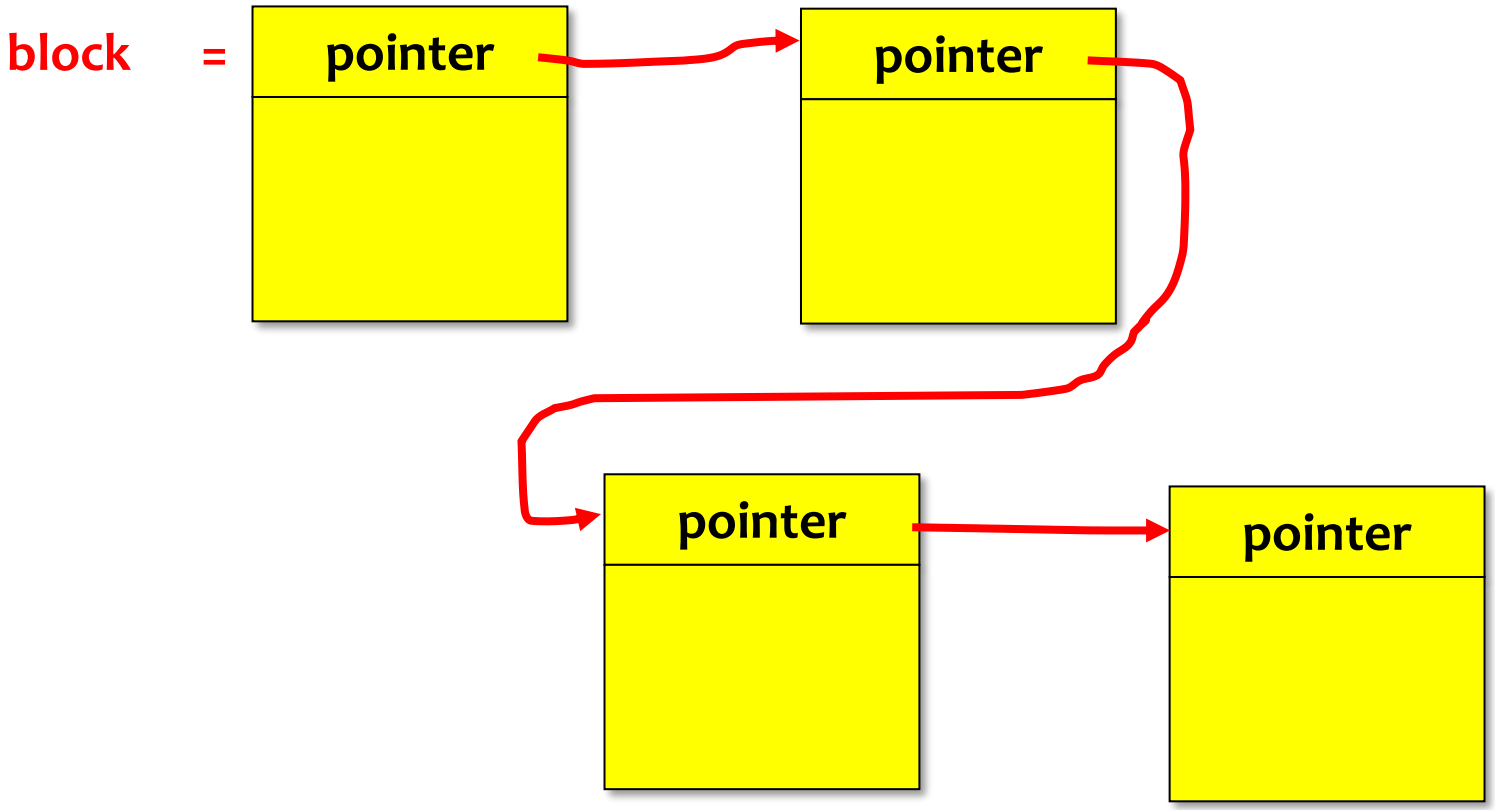
- Extents are allocated for file allocation
- A file consists of one or more extents.

The location of a file's blocks is then recorded as a location and a block count, plus a link to the first block of the next extent.

The commercial Veritas File System uses extents to optimize performance. It is a high-performance replacement for standard UNIX UFS.

Linked Allocation

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



Linked Allocation (Cont.)

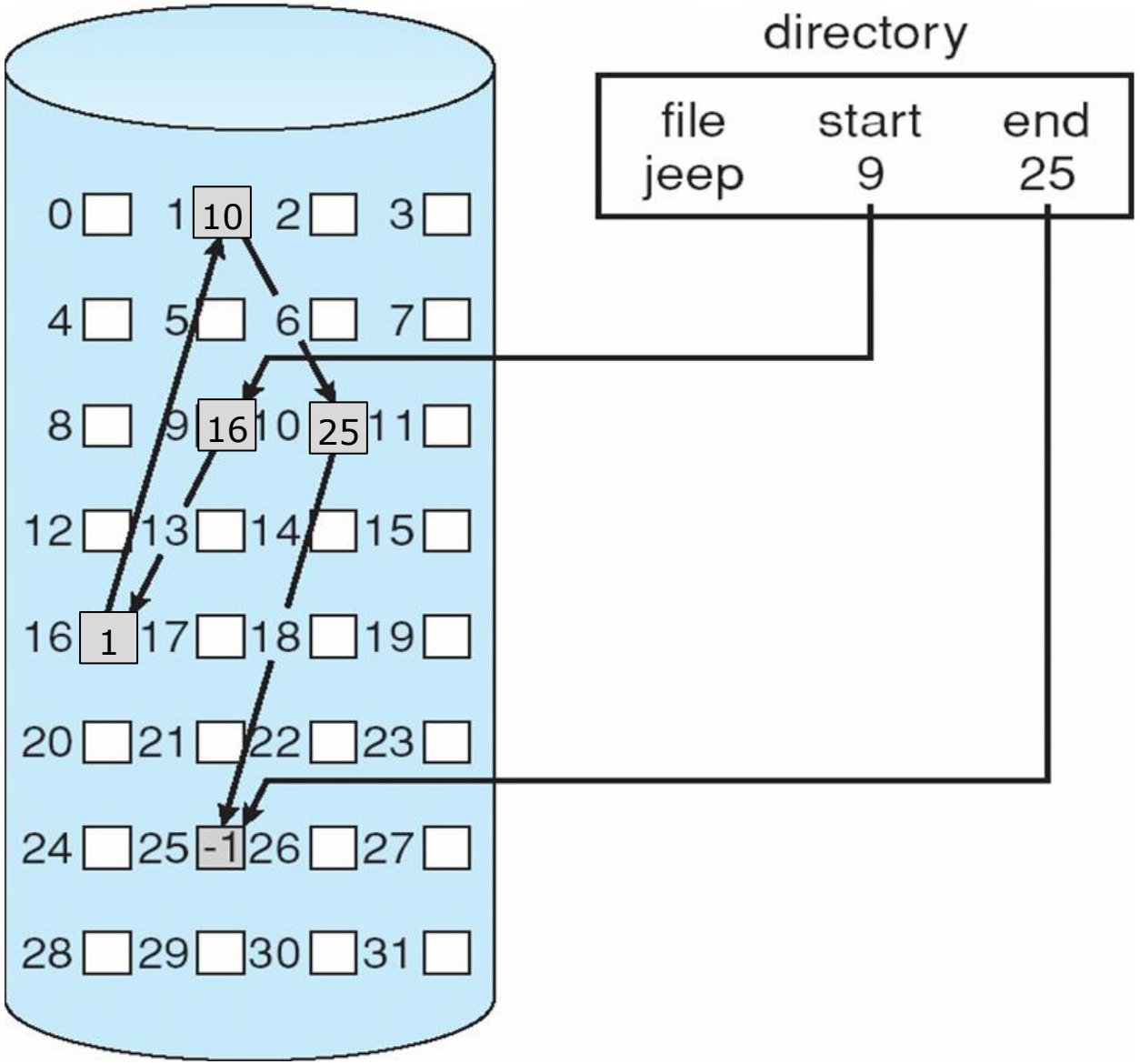
Simple – need only starting address

Free-space management system – no waste of space

Some disadvantages

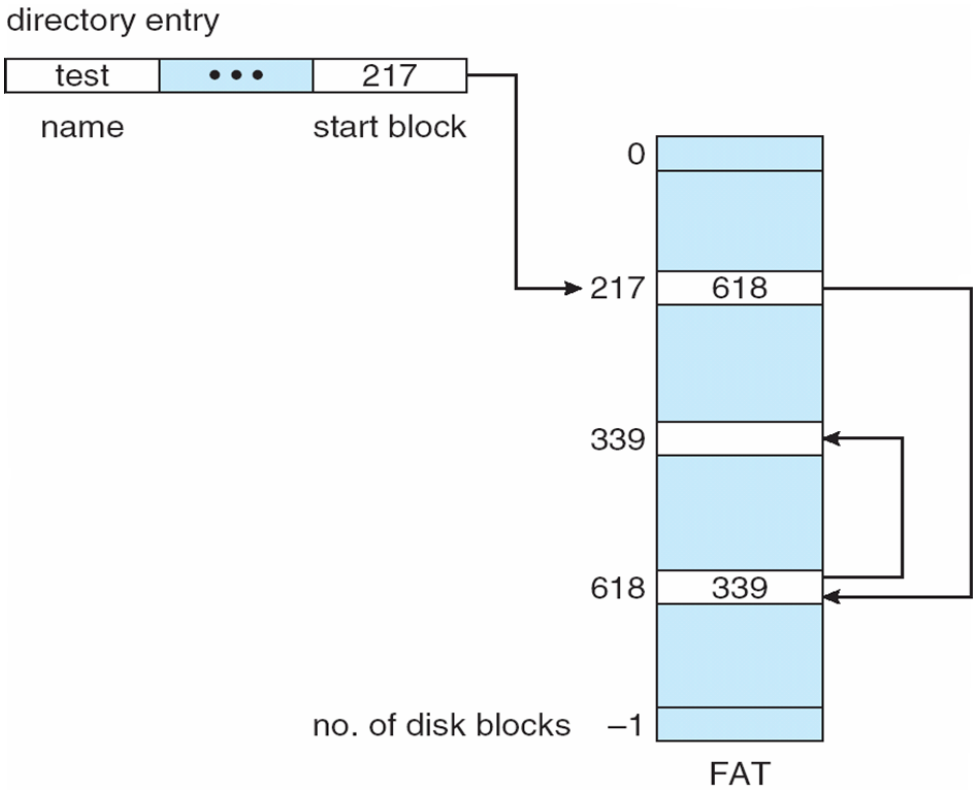
- No random access (only for sequential access files)
- Space required for pointers, If a pointer requires 4 bytes out of a 512 bytes block, 0.78 percent is used for pointers
  - Clusters (k blocks)
- Reliability Issue

Linked Allocation



File-Allocation Table

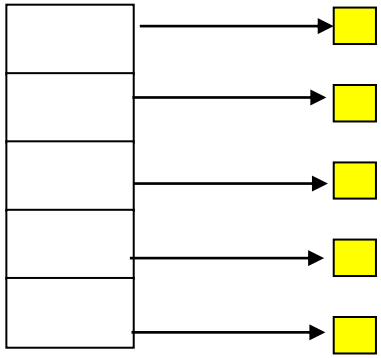
- An important variation on linked allocation is the use of a file-allocation table (FAT)
- The FAT allocation scheme can result in a significant number of disk head seeks, unless the FAT is cached.



Indexed Allocation

Brings all pointers together into the *index block.*

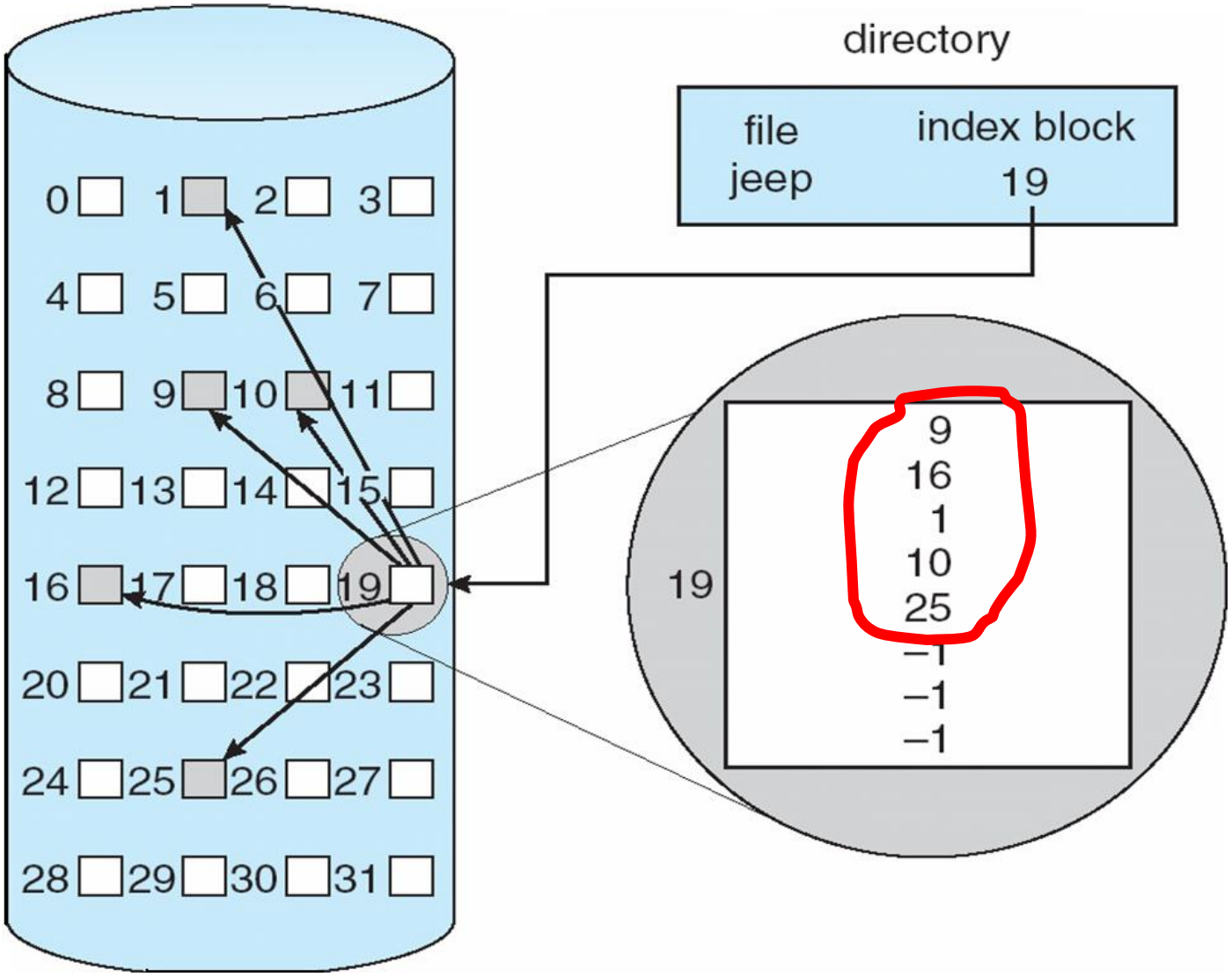
Logical view.



blocks

index table

Example of Indexed Allocation



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.

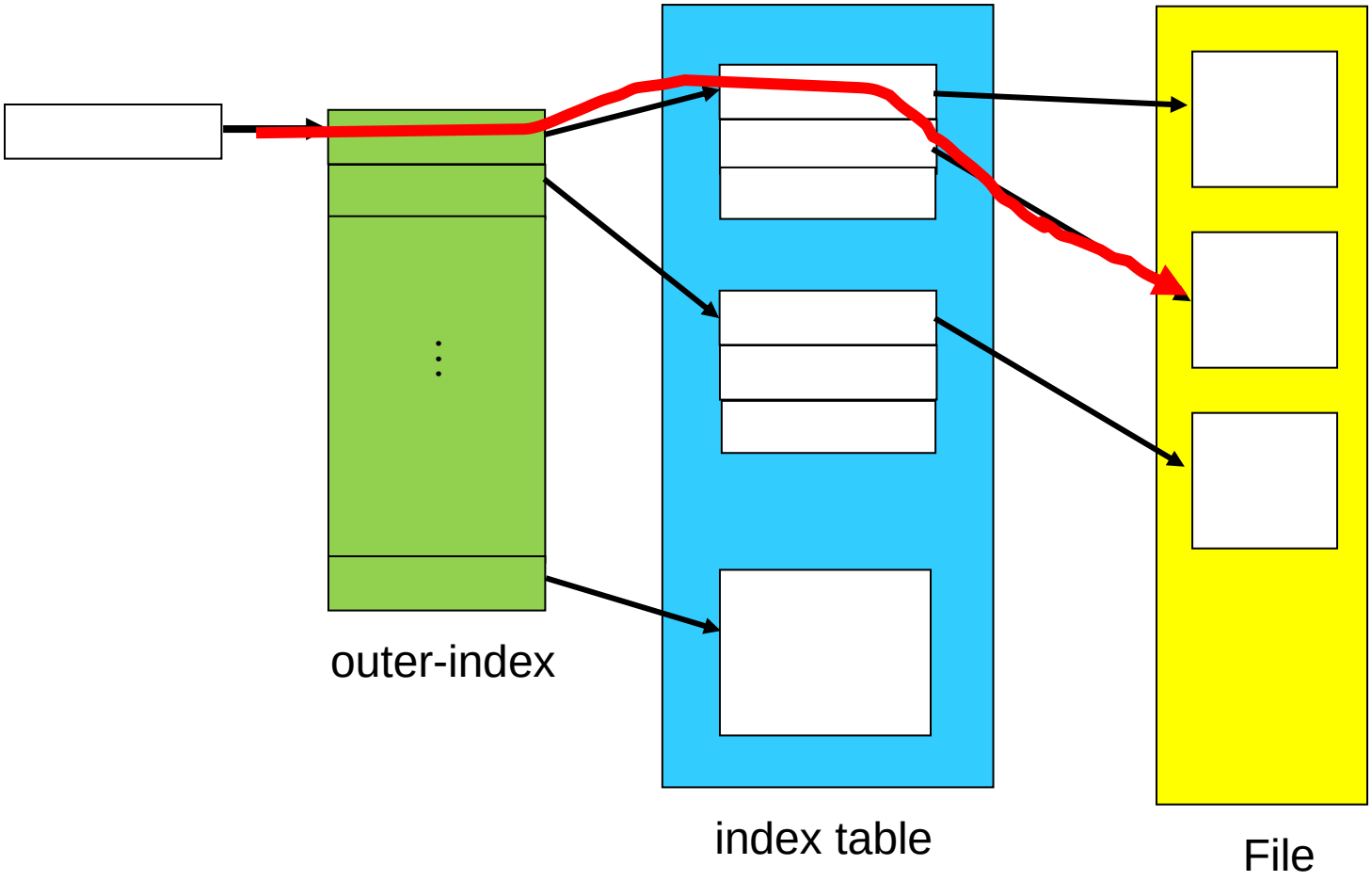
Only 1 block is needed for index table.
256K/0.5K = 512 blocks

Indexed Allocation – Mapping (Cont.)

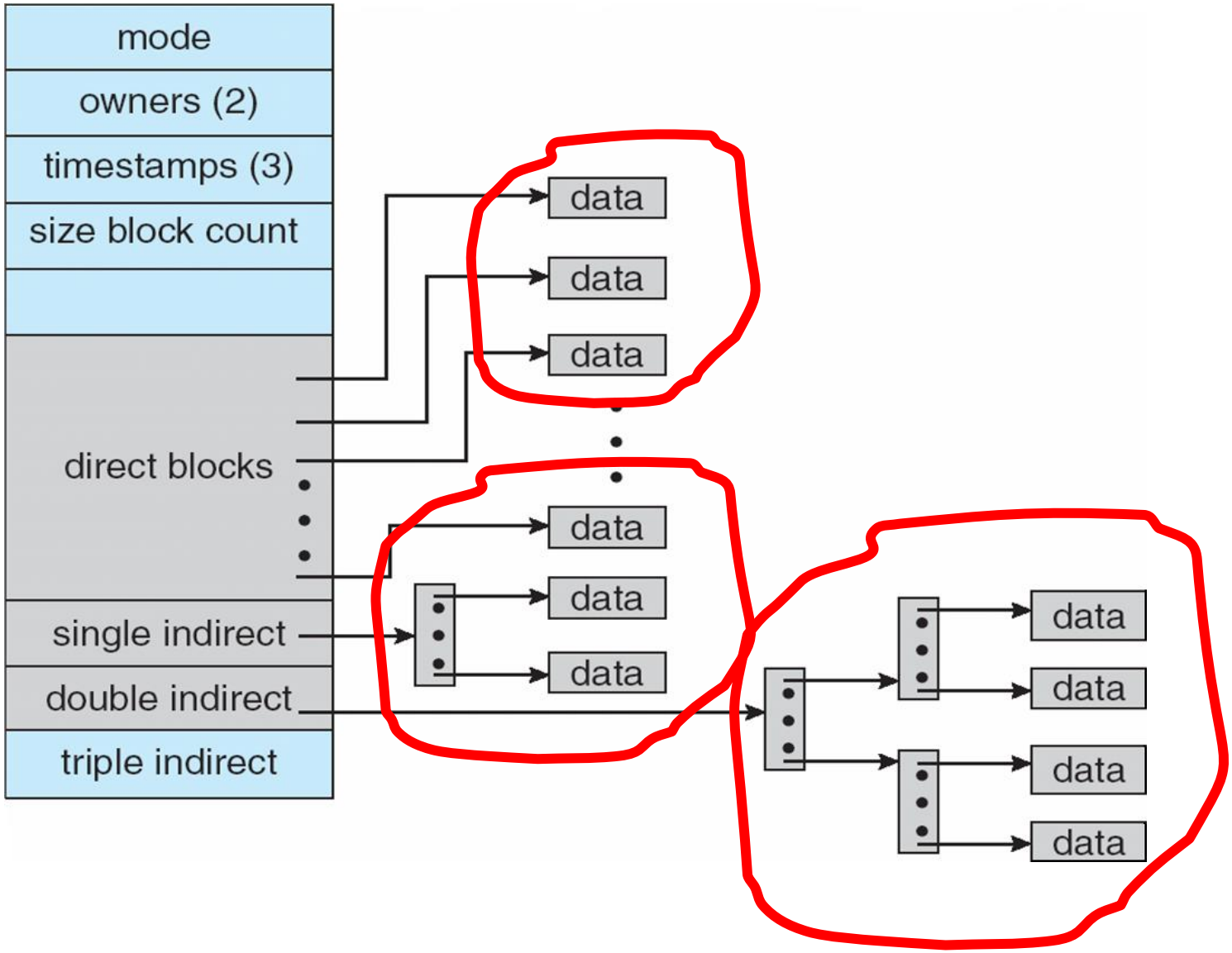
Mapping from logical to physical in a file of unbounded length (block size of 512 words).

Linked scheme – Link blocks of index table (no limit on size).

Indexed Allocation – Mapping (Cont.)

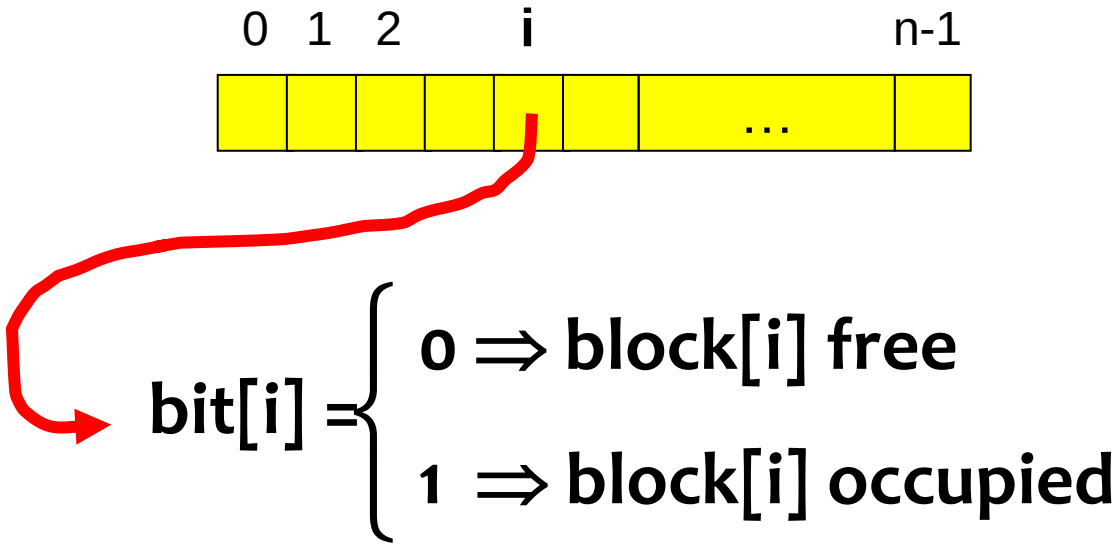


Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

Bit vector (*n* blocks)



Free-Space Management (Cont.)

Bit map requires extra space

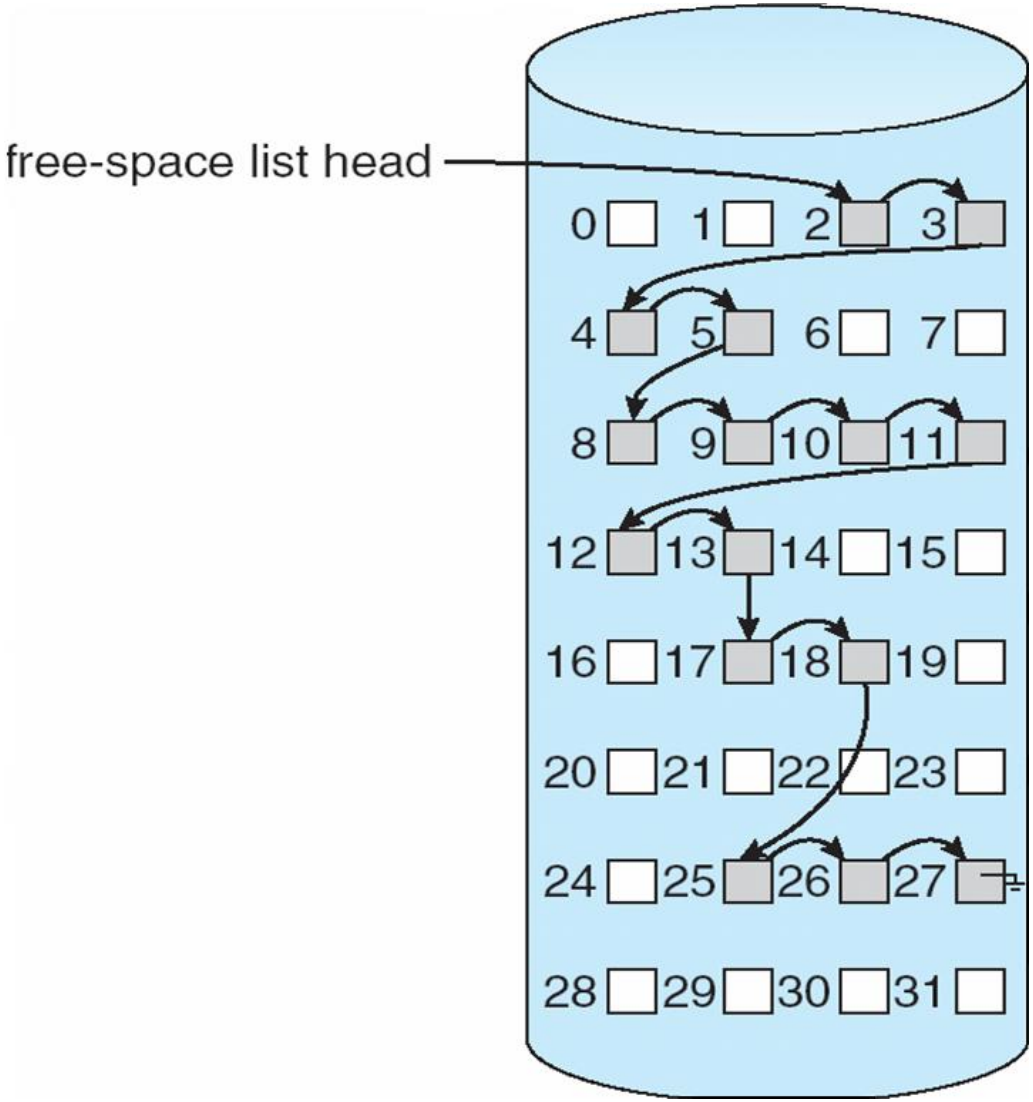
- Example:
  - block size = 2^{12} bytes
  - disk size = 2^{30} bytes (1 gigabyte)
  - $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

Easy to get contiguous files …. 000**11111111**000….

Linked list (free list)

- Cannot get contiguous space easily
- No waste of space

Linked Free Space List on Disk



Free-Space Management (Cont.)

Grouping: stores the **addresses of n free blocks** in the first free block. A block has **n** entries.

The first n-1 of these blocks are actually free.

The last block contains the addresses of another n free blocks, and so on.

Counting: keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block.

Address 1, n1 (2,3)

Address 2, n2, (8,5)

Address 3, n3, (17,2) …

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 (occupied) in memory and bit[*i*] = 0 (free) on disk
- Solution:
  - Set bit[*i*] = 1 in disk
  - Allocate block[*i*]
  - Set bit[*i*] = 1 in memory

Efficiency and Performance

Efficiency

The Efficient use of disk space depends heavily on the disk allocation and directory algorithms

- UNIX inodes are preallocated on a volume. Even a "empty" disk has a percentage of it space lost to inodes.
- By preallocating, the inodes and spreading them across the volume, we improve the file system's performance. --- Try to keep a file's data block near that file's inode block to reduce the seek time.

The type of data kept in file's directory (or inode) entry also require consideration

- "Last write date"
- "Last access date"

Efficiency and Performance

Performance

disk cache – separate section of main memory for frequently used blocks

free-behind and read-ahead – techniques to optimize sequential access

- Free-behind removes a page from the buffer as soon as the next page is requested.
- With read-ahead, a requested page and several subsequent pages are read and cached.

improve PC performance by dedicating section of memory as virtual disk, or RAM disk

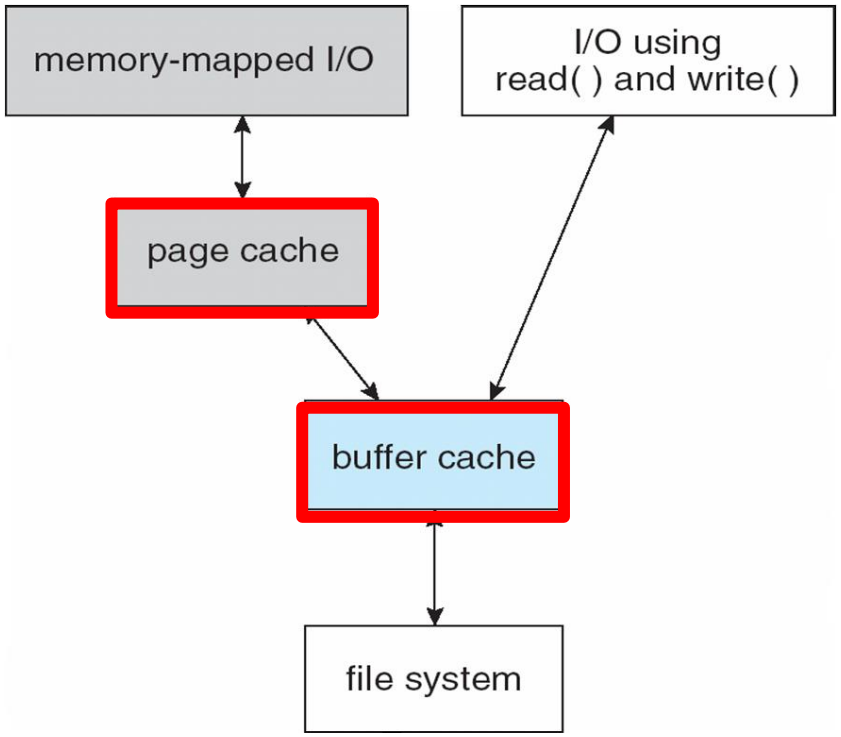
Page Cache

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

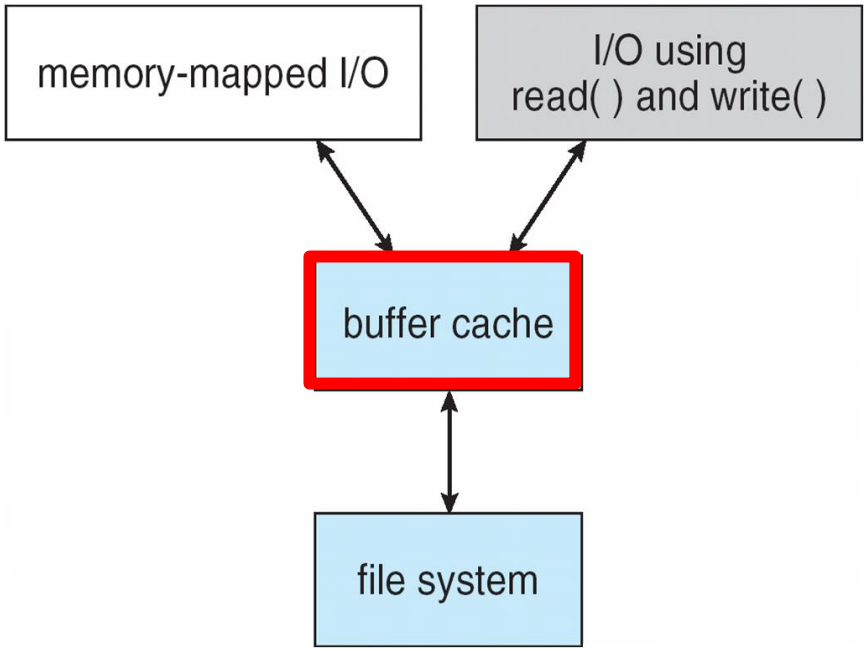
Double caching



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 (floppy disk, magnetic tape, other magnetic disk, optical)

Recover lost file or disk by restoring data from backup

Log Structured File Systems

Log structured 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)

Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner

NFS (Cont.)

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

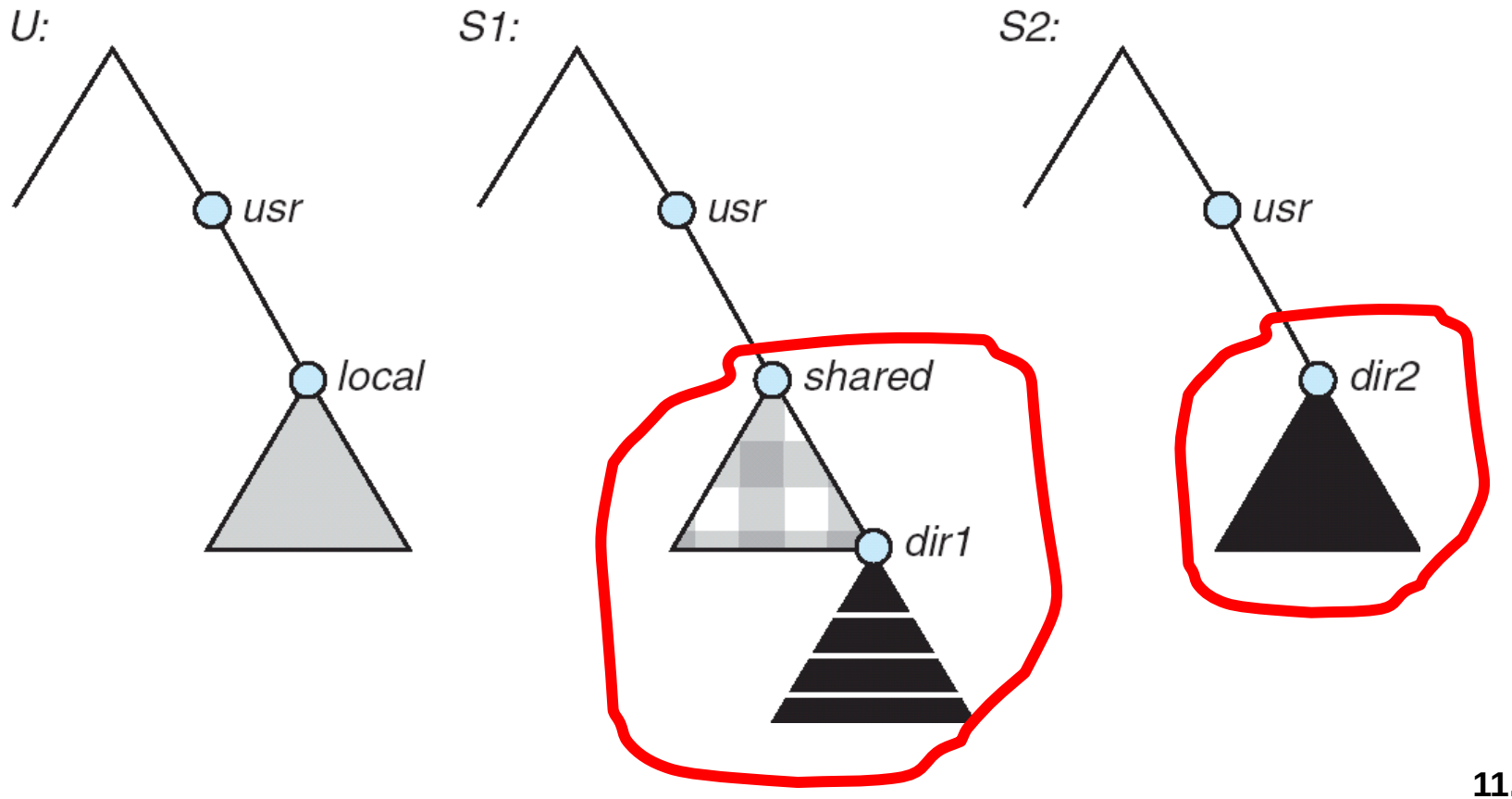
- Files in the remote directory can then be accessed in a transparent manner

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.)

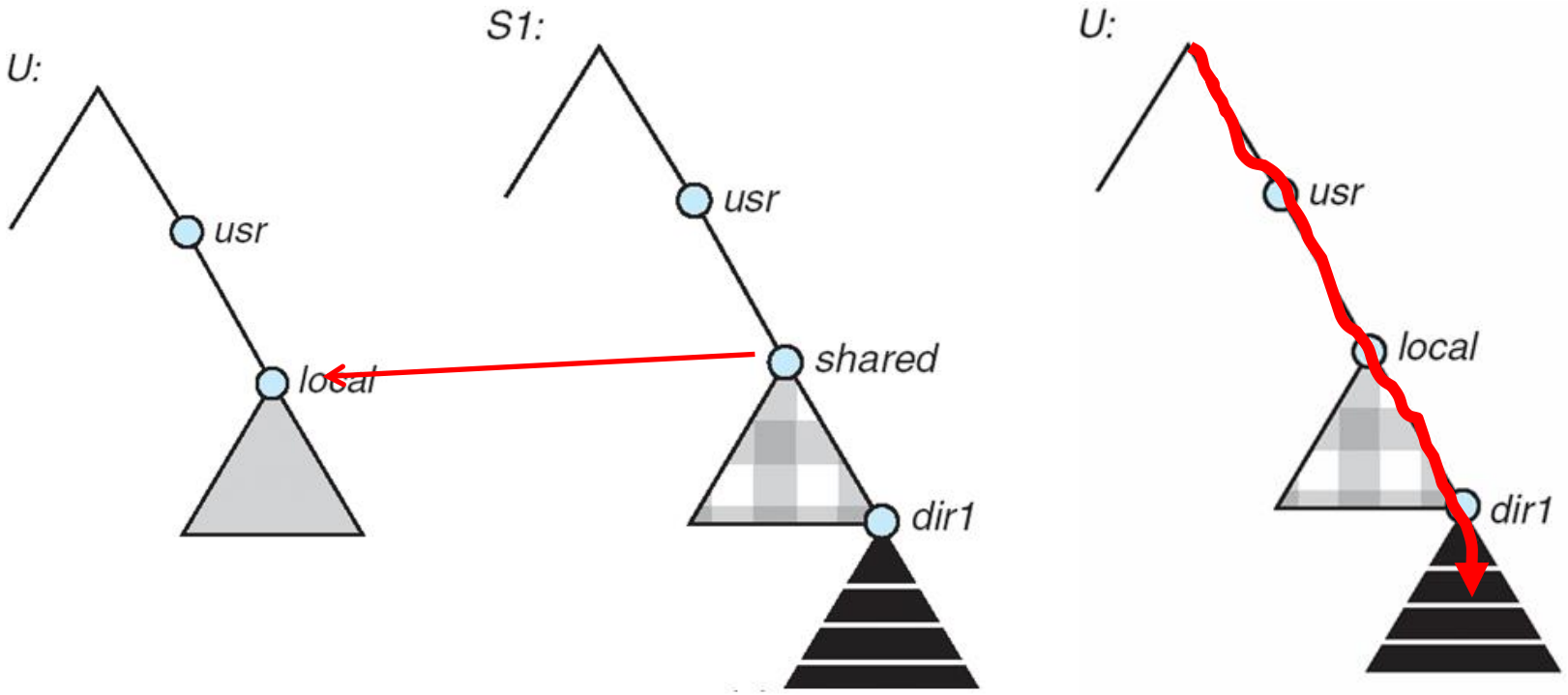
Consider the following file system where the triangles represent subtrees of directories that are of interest.

Three independent file systems of machines: U, S1, and S2



NFS (Cont.)

The effect of mounting S1:/user/shared over U:/user/local



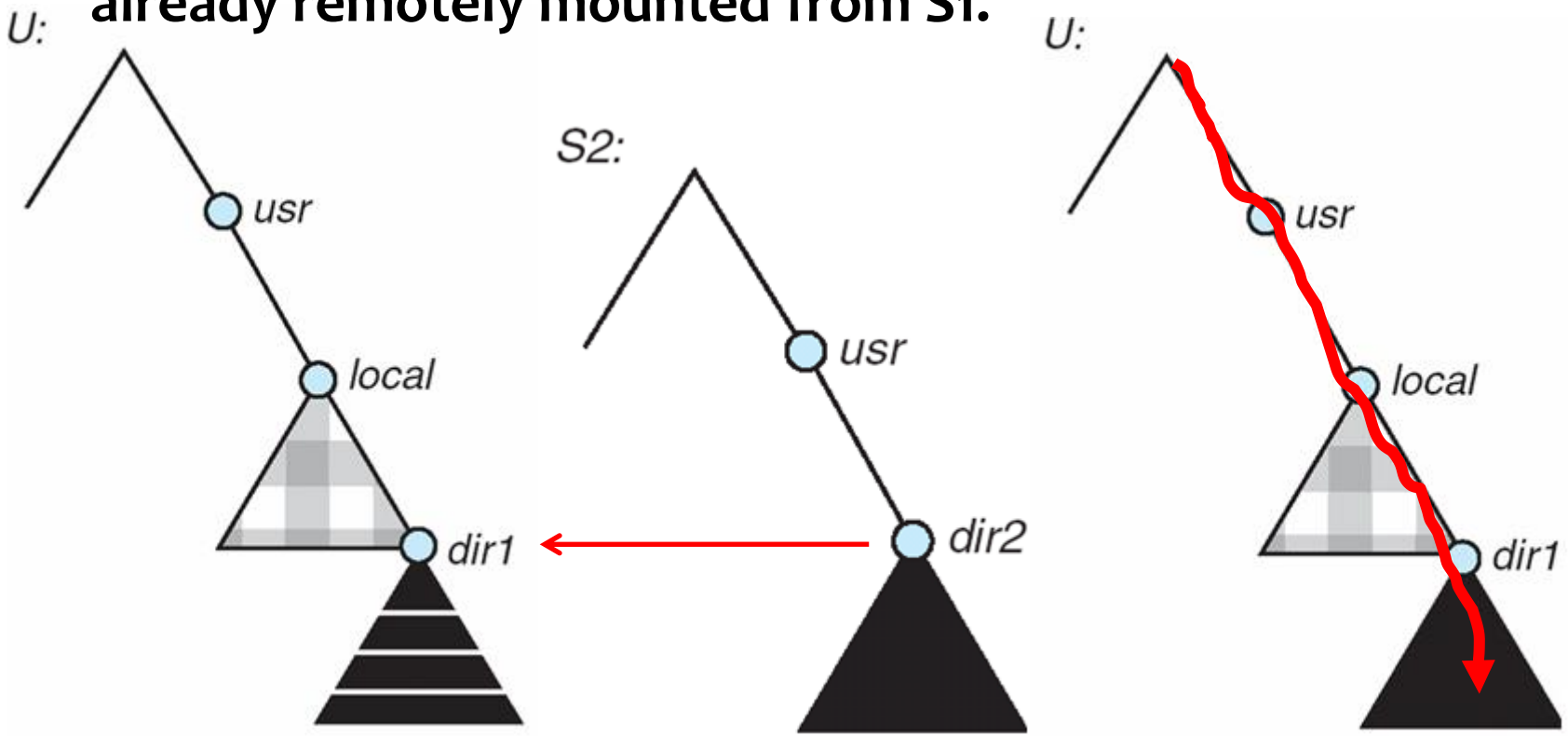
The machine U can access any file within the dir1 directory using prefix /usr/local/dir1.

The original /usr/local on that machine is no longer visible.

NFS (Cont.)

Cascading mounts.

mounting S2:/user/dir2 over U:/user/local/dir1 which is already remotely mounted from S1.



Users can access files within the dir2 on U using prefix /usr/local/dir1.

NFS (Cont.)

User mobility

If a shared file system is mounted over a user's home directories on all machines in a network, the user can log into any workstation and get his home environment.

NFS (Cont.)

One of the design goals of NFS was 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

NFS (Cont.)

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

Accordingly, two separate protocols are specified for these services:

- a mount protocol and
- an NFS protocol for remote file accesses.

The protocols are specified as sets of RPCs.

- These RPCs are the building blocks used to implement transparent remote file access.

NFS Mount Protocol

The mount protocol establishes initial logical connection between a server and a client.

A 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

The server maintains an 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

In Solaris, this list is the /etc/dfs/dfstab, which can be edited only by a supervisor.

Any directory within an exported file system can be mounted by an accredited machine.

A component unit is such a directory.

Following a mount request that conforms to its export list, the server returns a file handle — a key for further accesses

In UNIX terms, the file handle consists of 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

The NFS protocol provides a set of RPCs 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

These procedures can be invoked only after a file handle for the remotely mounted directory has been established.

NFS Protocol

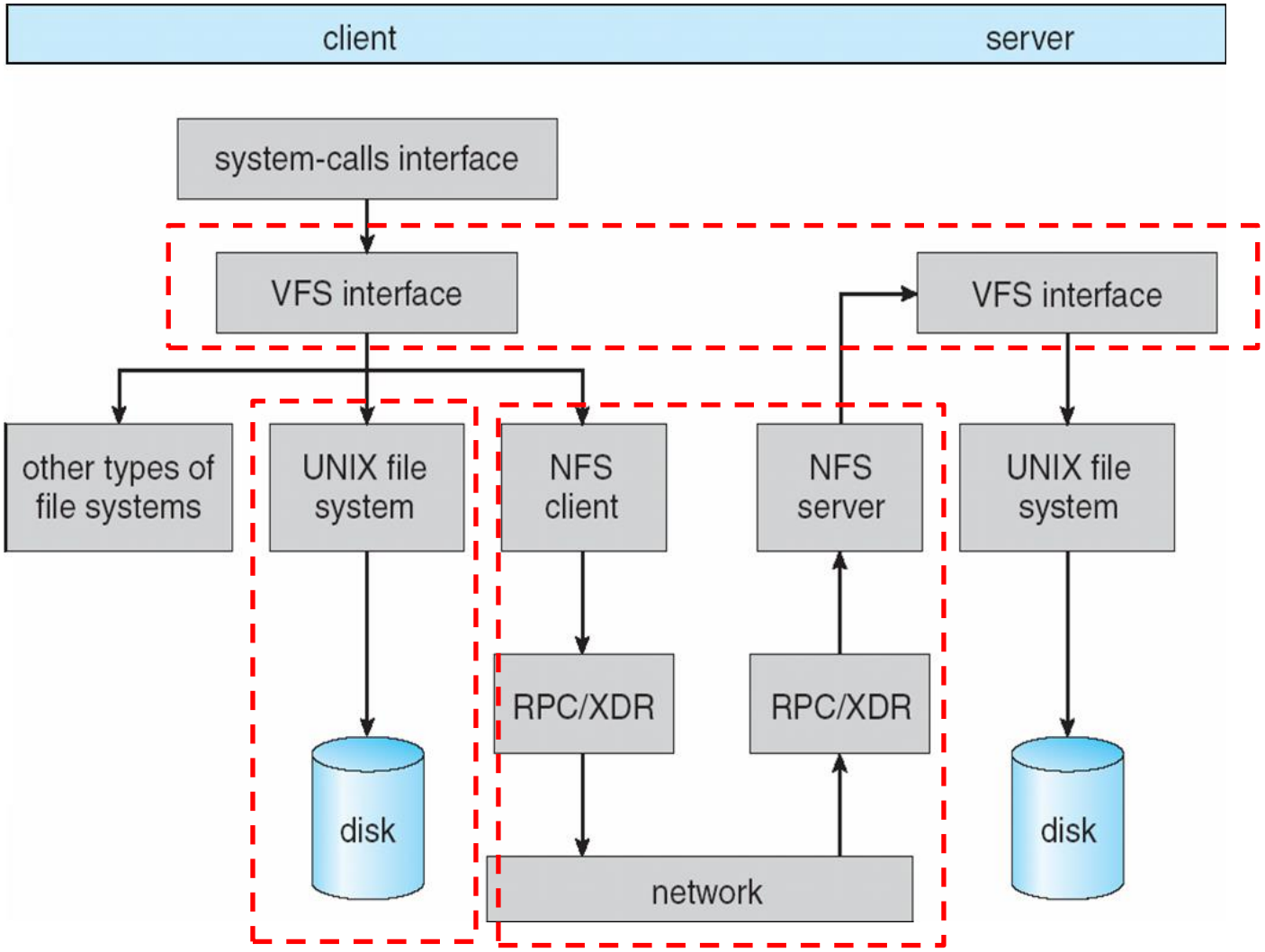
NFS servers are stateless; Servers do not maintain information about their clients from one access to another. Each request has to provide a full set of arguments.

Every NFS request has a sequence number, allowing the server to determine if a request is duplicated or missed.

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

Schematic View of NFS Architecture



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

An illustration of NFS Architecture

NFS is integrated into the OS via a VFS.

Let's trace how the operation on an already open remote file is handled.

The client initiates the operation with a regular system call.

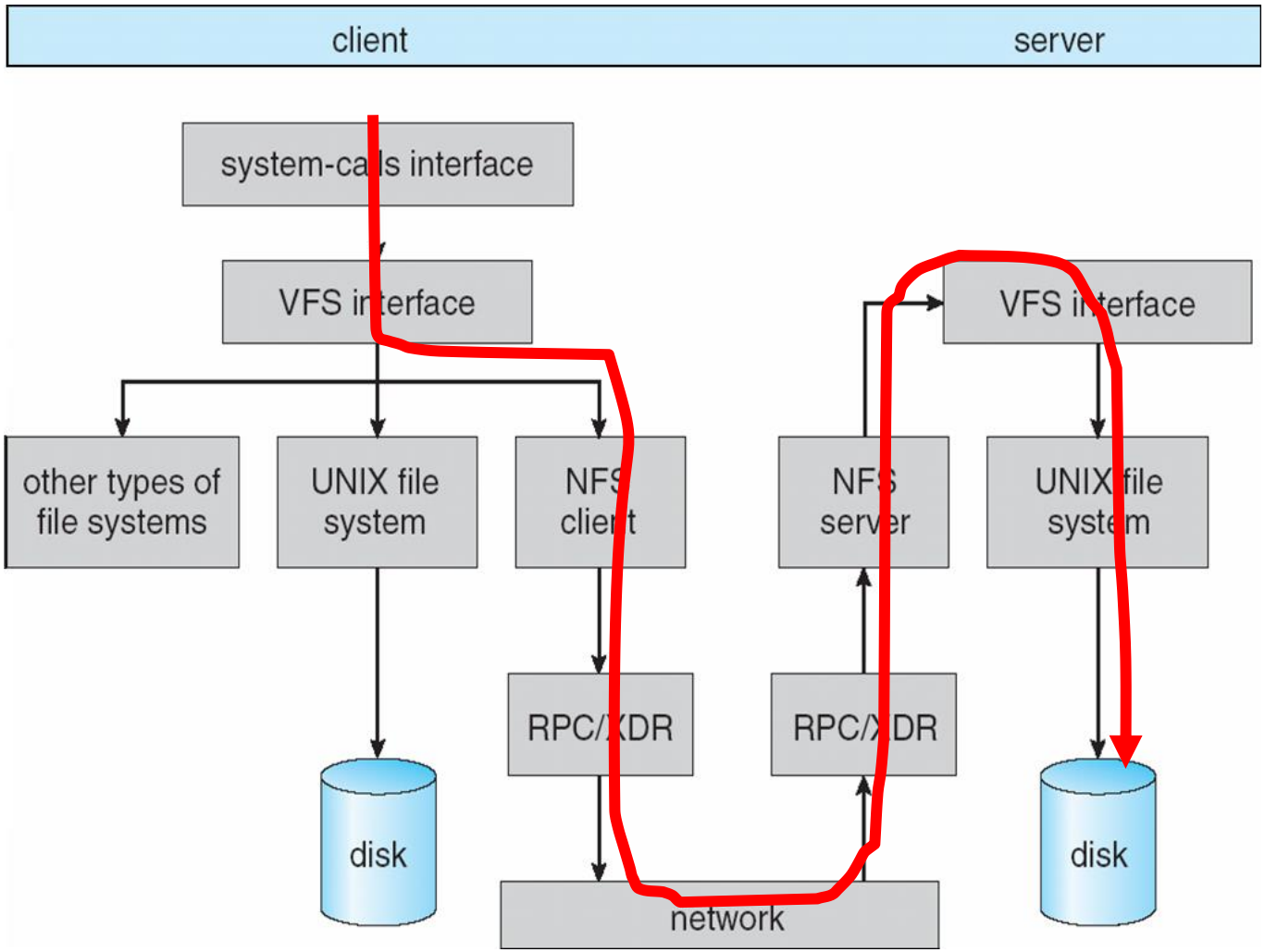
The OS layer maps this call to a VFS operation on the appropriate vnode.

The VFS layer identifies the file as a remote one and invokes the appropriate NFS procedure.

An RPC call is made to the NFS service layer at the remote server. This call is reinjected to the VFS layer on the remote system, which finds that it is local and invokes the appropriate file system operation.

This path is retraced to return the results.

Schematic View of NFS Architecture



NFS Path-Name Translation

Path-name translation in NFS involves the parsing of a path name such as /usr/local/dir1/file.txt into separate directory entries, or components: (1) usr, (2) local, and (3) dir1.

Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode.

Once a mount point is crossed, every component lookup causes a separate RPC to the server.

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 for file operations and the NFS protocol RPCs (except opening and closing files).

A remote file operation can be translated directly to the corresponding RPC.

NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance.

There are two caches: the file-attribute cache and the file-blocks cache.

NFS Remote Operations

When a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes.

The cached file blocks are used only if the corresponding cached attributes are up to date.

The attribute cache is updated whenever new attributes arrive from the server.

Cached attributes are discarded after 60 seconds.

Both read-ahead and delayed-write techniques are used between the server and the client.

Clients do not free delayed-write blocks until the server confirms that the data have been written to disk.

Example: WAFL File System

Disk I/O has a huge impact on system performance.

Some file systems are general purpose, others are optimized for specific tasks,

The WAFL ("Write-anywhere file layout") file system from Network Appliance, is a powerful file system optimized for random writes.

Used on Network Appliance "Files" – distributed file system appliances

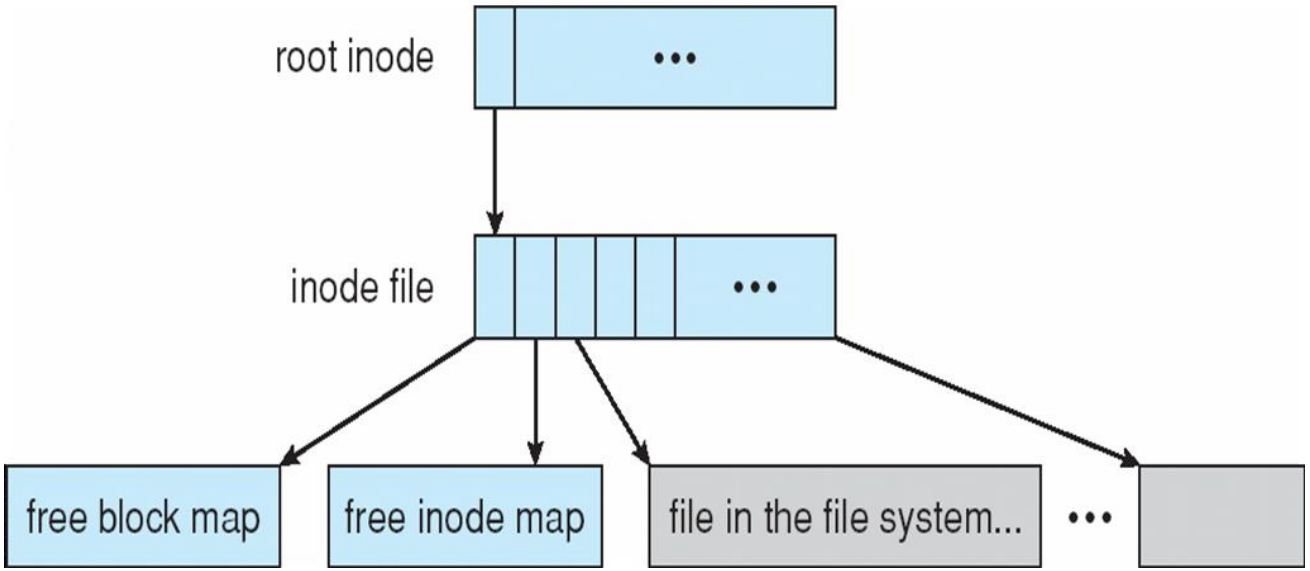
Serves up NFS, CIFS, http, ftp protocols

Random I/O optimized, write optimized

Example: WAFL File System

Similar to Berkeley Fast File System, with extensive modifications.

It is block-based and uses inodes to describe files. Each inode contains 16 pointers to blocks (or indirect blocks) belonging to the file described by the inode.



The WAFL File Layout

Example: WAFL File System

Each file system has a root inode.

A WAFL file system is a tree of blocks with the root inode as its base.

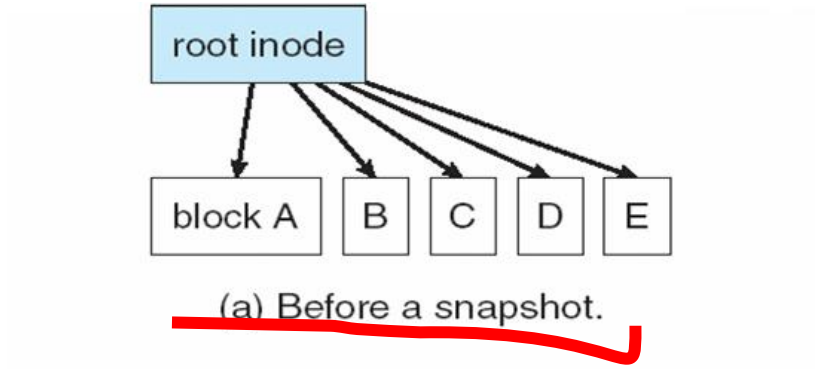
To take a snapshot, WAFL creates a copy of the root inode.

Any file or metadata updates after that go to new blocks rather than overwriting their existing blocks.

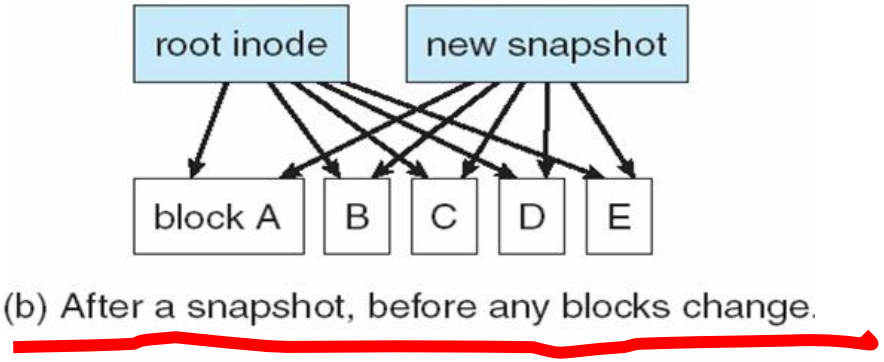
Used blocks are never overwritten, so writes are very fast, because a write can occur at the free block nearest the current head location.

The snapshot facility is also useful for backups, testing, versioning, and so on.

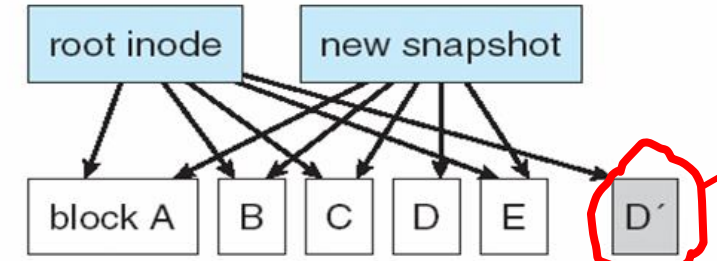
Snapshots in WAFL



(a) Before a snapshot.



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



(c) After block D has changed to D´.

A write can occur at the free block nearest the current head location

End of Chapter 11