Amirkabir University of Technology
(Tehran Polytechnic)
Department of Computer Engineering

File System Implementation

Hamid R. Zarandi
h_zarandi@aut.ac.ir

File system structure

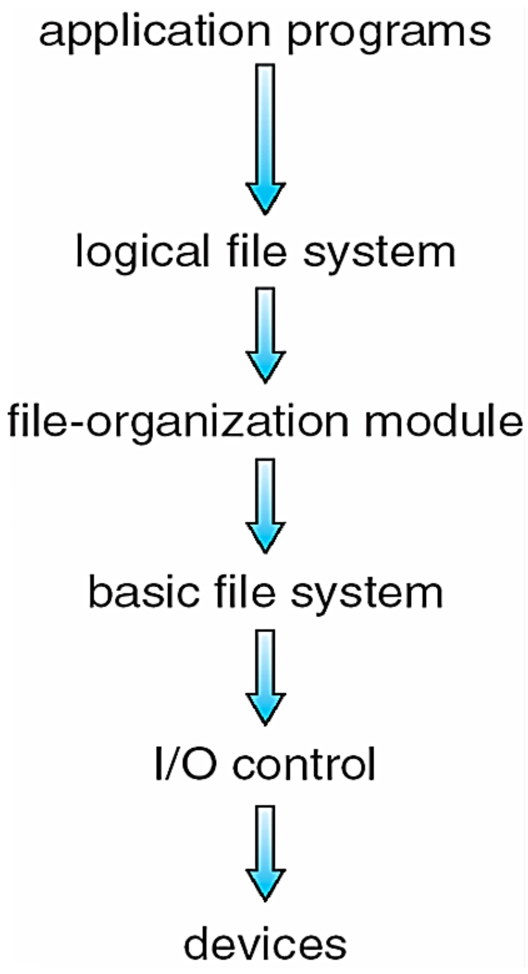
- **Why file system?**
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located, retrieved easily

- **File control block**
  - Storage structure consisting of information about a file

- **Device driver**
  - Controls the physical device

Layered file system

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX), FCB and metadata
  - Directory management
  - Protection
- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation
- **Basic file system** given command like "retrieve block 123" translates to device driver
- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
  - Interrupt Service Routines
- Layered design
  - Reduction of complexity and redundancy
  - Overheads and performance penalty translation of file to disk



File system layers

- Each with its own format
  - CD-ROM: ISO 9660
  - Unix: UFS, FFS
  - Windows: FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray file systems
  - Linux: more than 40 types (extended file system ext2 and ext3)
    - +distributed file systems
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

- File systems should have
  - 1) How to boot?
  - 2) Total # of blocks?
  - 3) # and location of free blocks?
  - 4) Directory structures?
  - 5) Files?

File system implementation

➢ Boot control block (Boot sector)

- 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

➢ Per-file File Control Block (FCB)

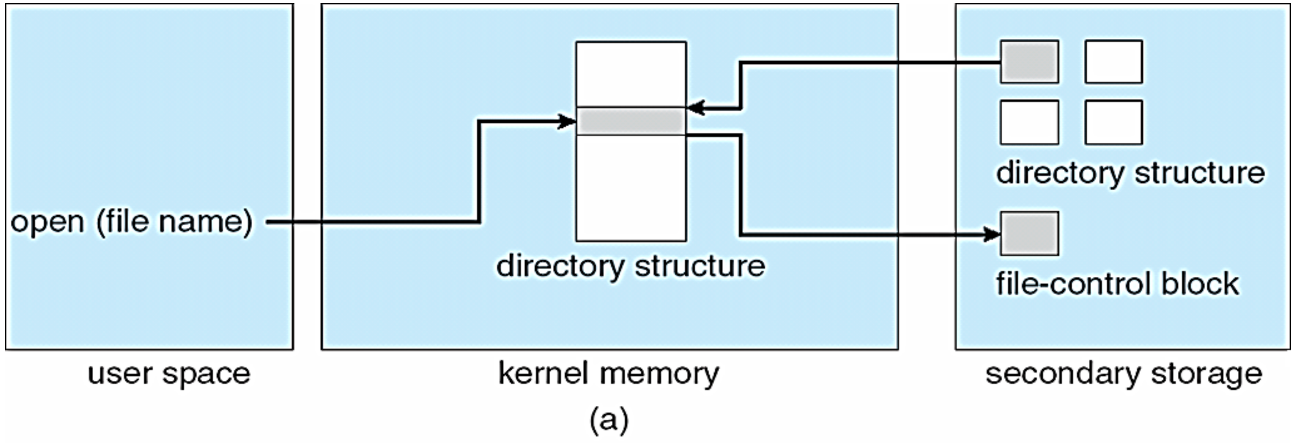
- ➢ Contains many details about the file

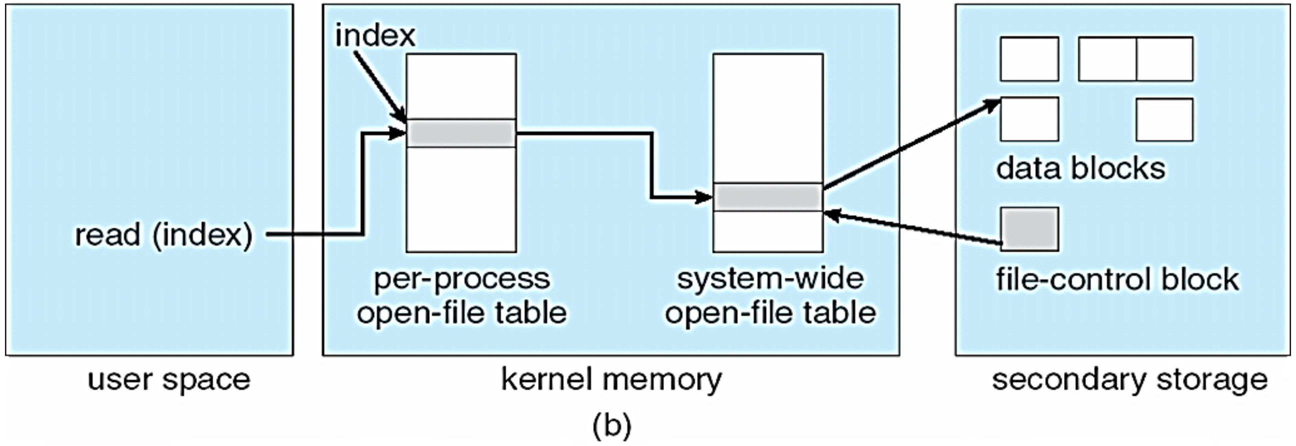
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

File Control Block

In-memory file system structures

- Mount table
- System wide open file table
- per-process open file table



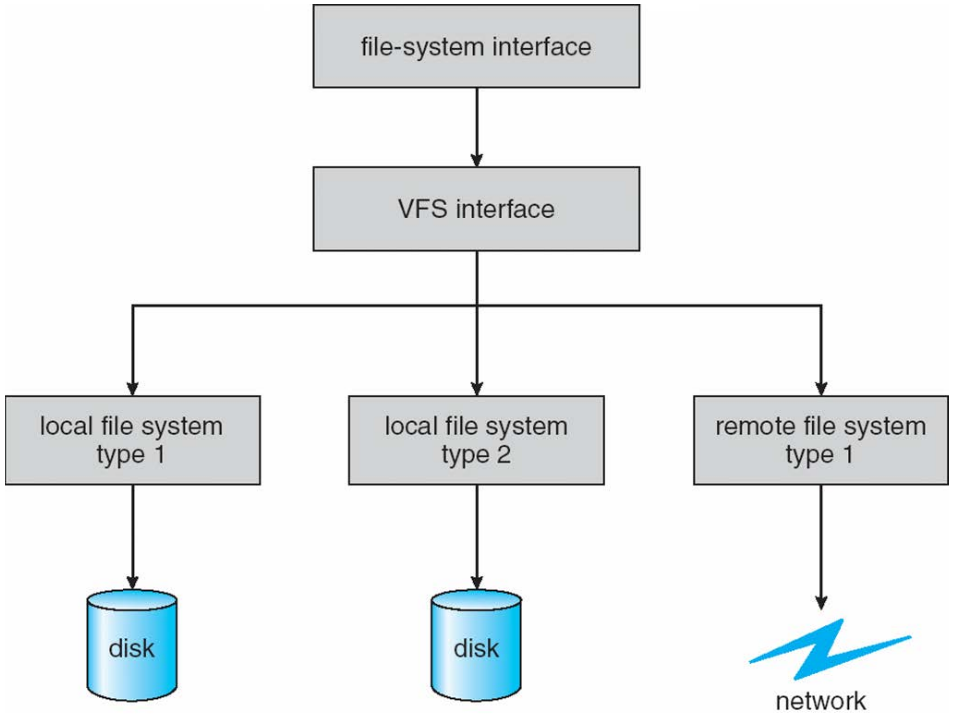


Partitions and mounting

- Boot loader
- 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

Virtual file systems (VFS)

- Virtual File Systems (VFS) on Unix
  - Provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (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
  - Then dispatches operation to appropriate file system implementation routines



Directory Implementation

Directory implementation

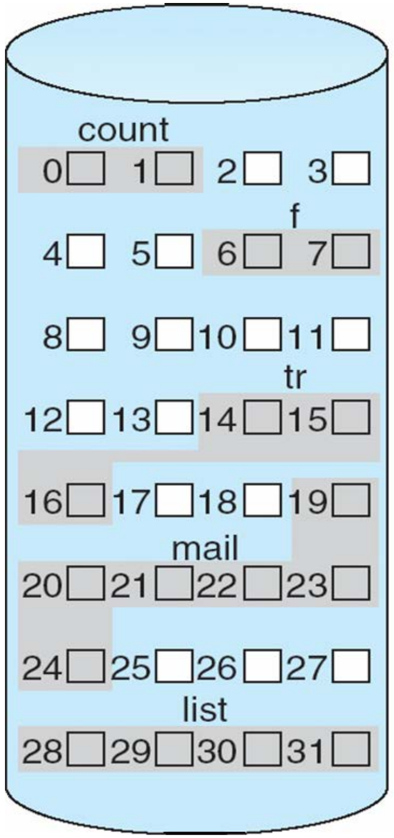
- 1) Linear list of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute (Why?)
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
  - What would happen if a file is deleted?
    - Problem with fragmentation
  - Problem of limited size

- 2) Hash Table
  - Linear list with hash data structure
  - Decreases directory search time
  - Hash is used to map file name to a number
  - Collisions – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods

1) Contiguous allocation

- ➢ Contiguous allocation
  - o Each file occupies set of contiguous blocks
  - o Best performance in most cases
  - o Simple – only starting location (block #) and length (number of blocks) are required
  - o Problems include
    - ▪ Finding space for file
    - ▪ Knowing file size
    - ▪ External fragmentation
    - ▪ Need for compaction off-line (downtime) or on-line

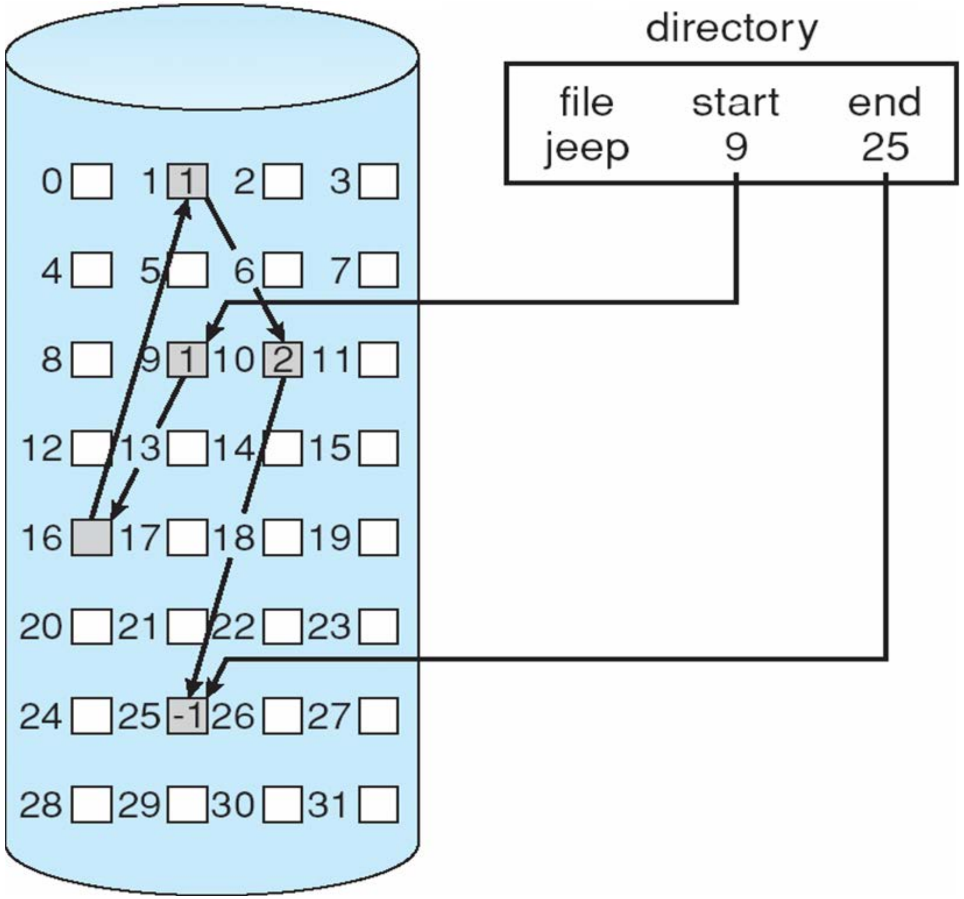


directory

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

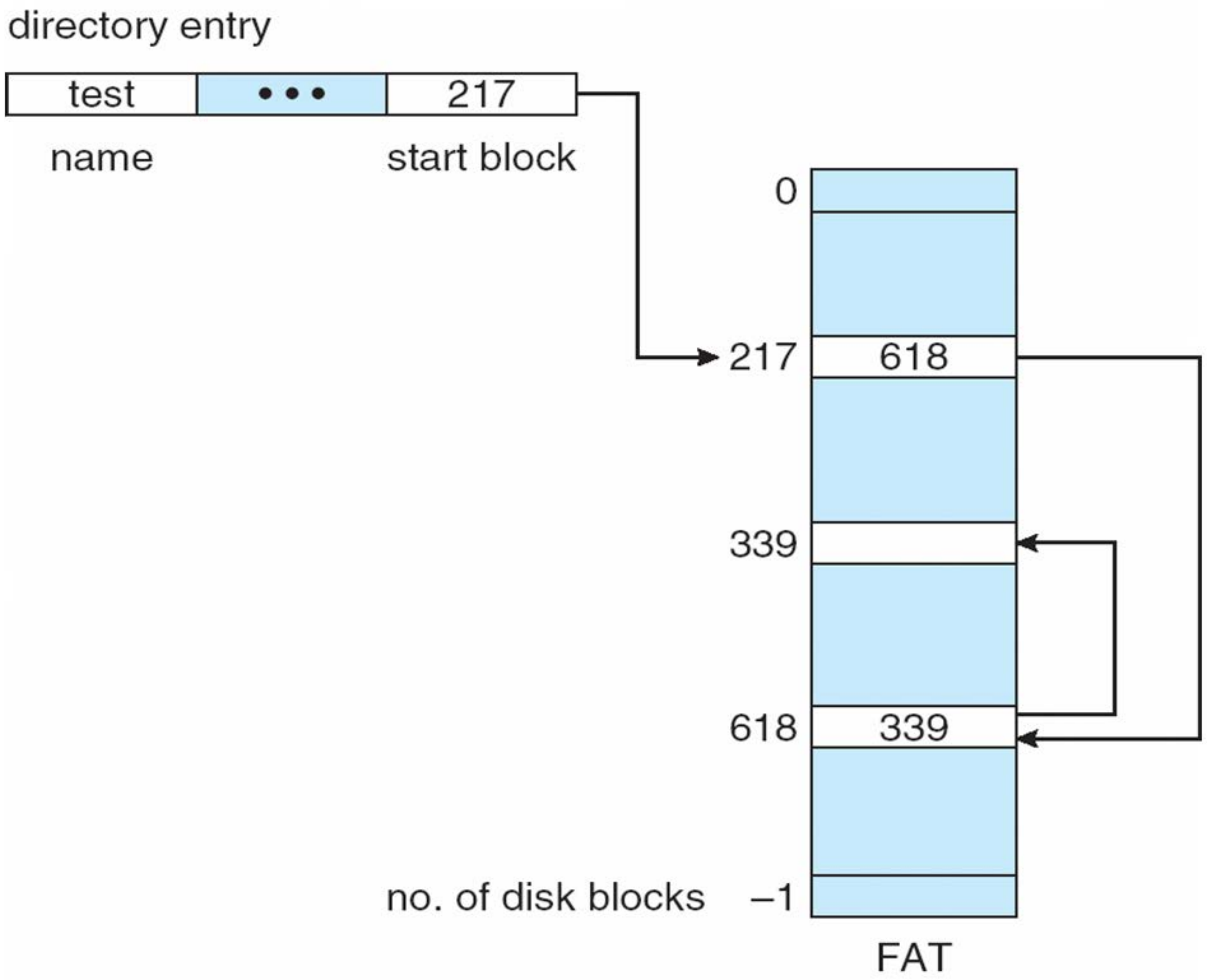
2) Linked allocation

- Linked allocation
  - Each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem (How?)
  - Locating a block can take many I/Os and disk seeks
  - 4 byte pointer in 512 byte block = 0.78% overhead
    - Use of cluster
    - Internal fragmentation



FAT in linked allocation

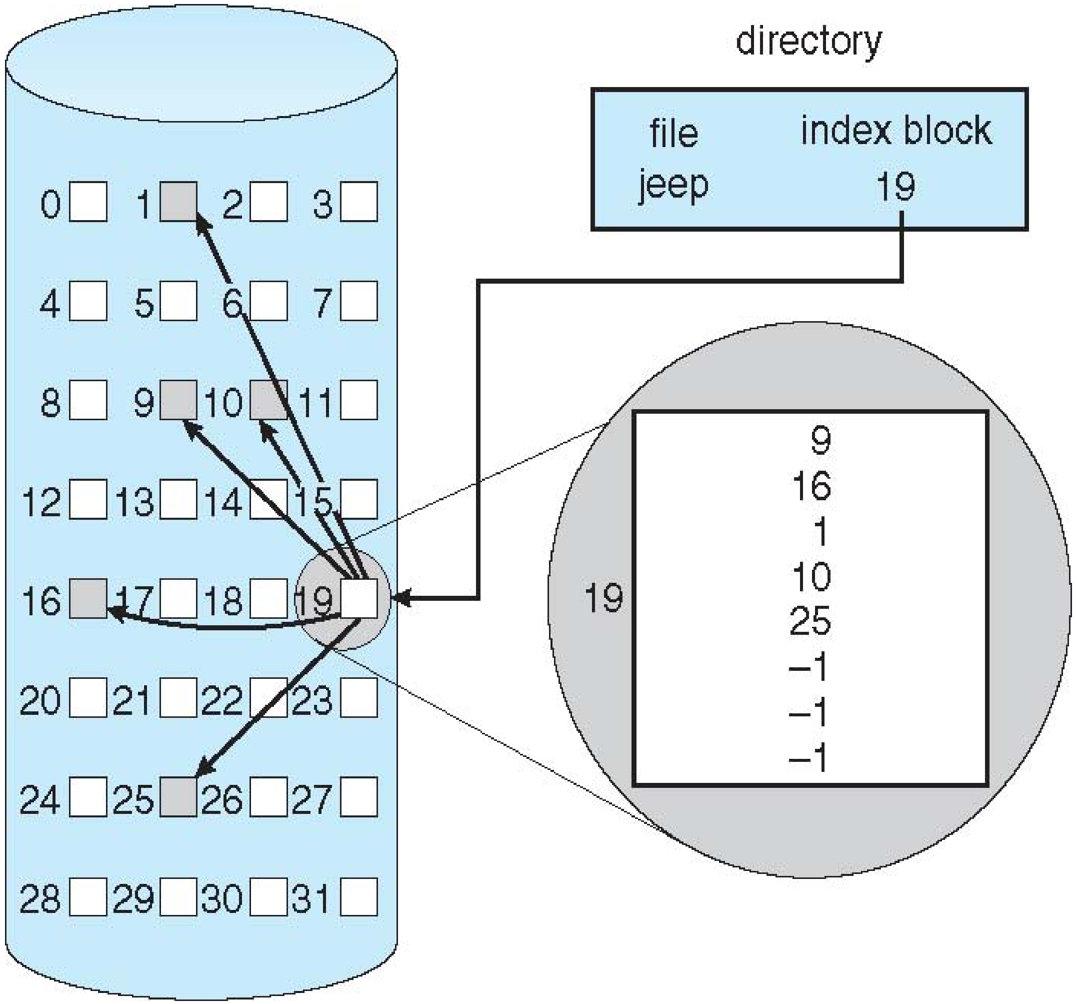
- **Windows FAT**
  - **File Allocation Table**



3) Indexed allocation

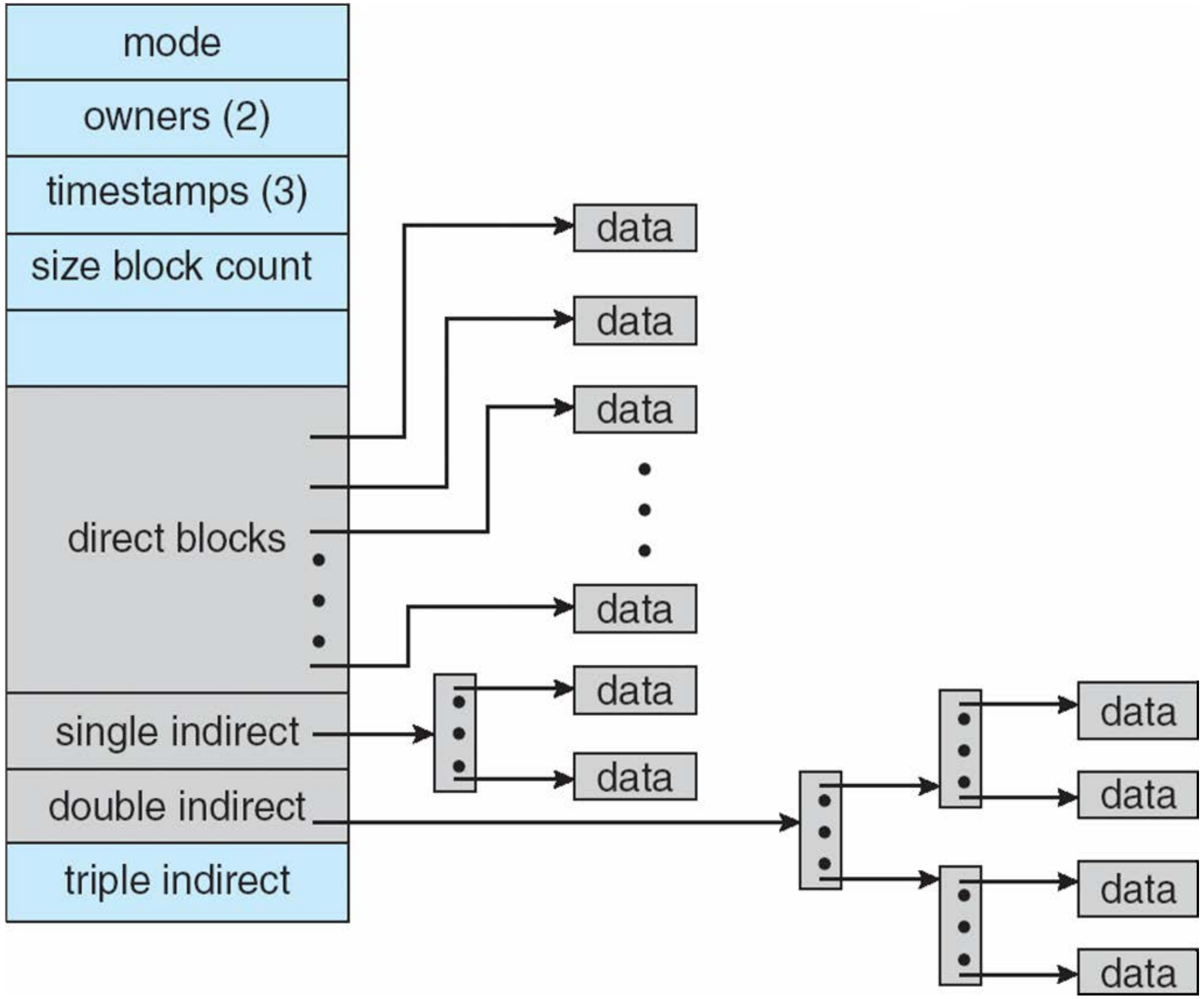
➢ Indexed allocation

- Each file has its own index block(s) of pointers to its data blocks
- 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 unbounded length (block size of 512 words)
- Linked scheme
  - Link blocks of index table (no limit on size)



4) Combined allocation

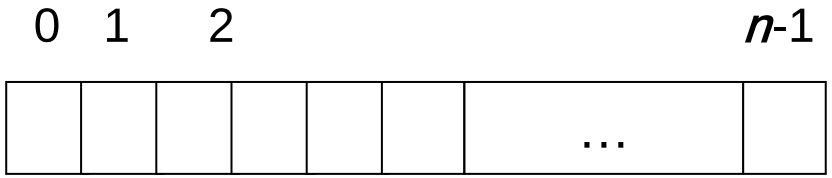
- Example:
  - UNIX UFS
- 4K bytes per block, 32-bit addresses
- More index blocks than can be addressed with 32-bit file pointer



Free Space Management

1) Bit vector (bit map)

- File system maintains free-space list to track available blocks/clusters
  - (Using term "block" for simplicity)
- Bit vector or bit map (*n* blocks)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- Bit map requires extra space
  - Example:

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

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

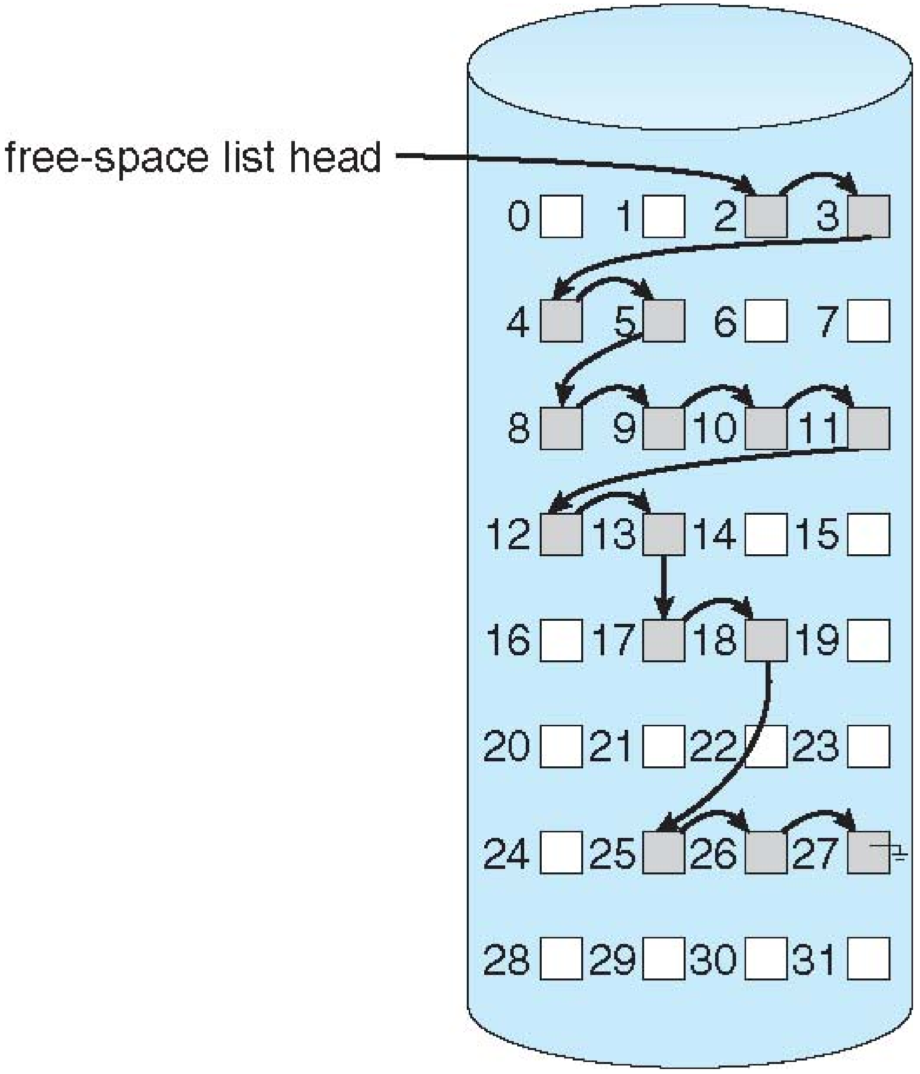
    $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files

2) Linked 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)
  - FAT is better to contain free blocks



3) Others: grouping, counting

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)
  (Similar to index block)
- The last free block contains address of another *m* free blocks

Counting

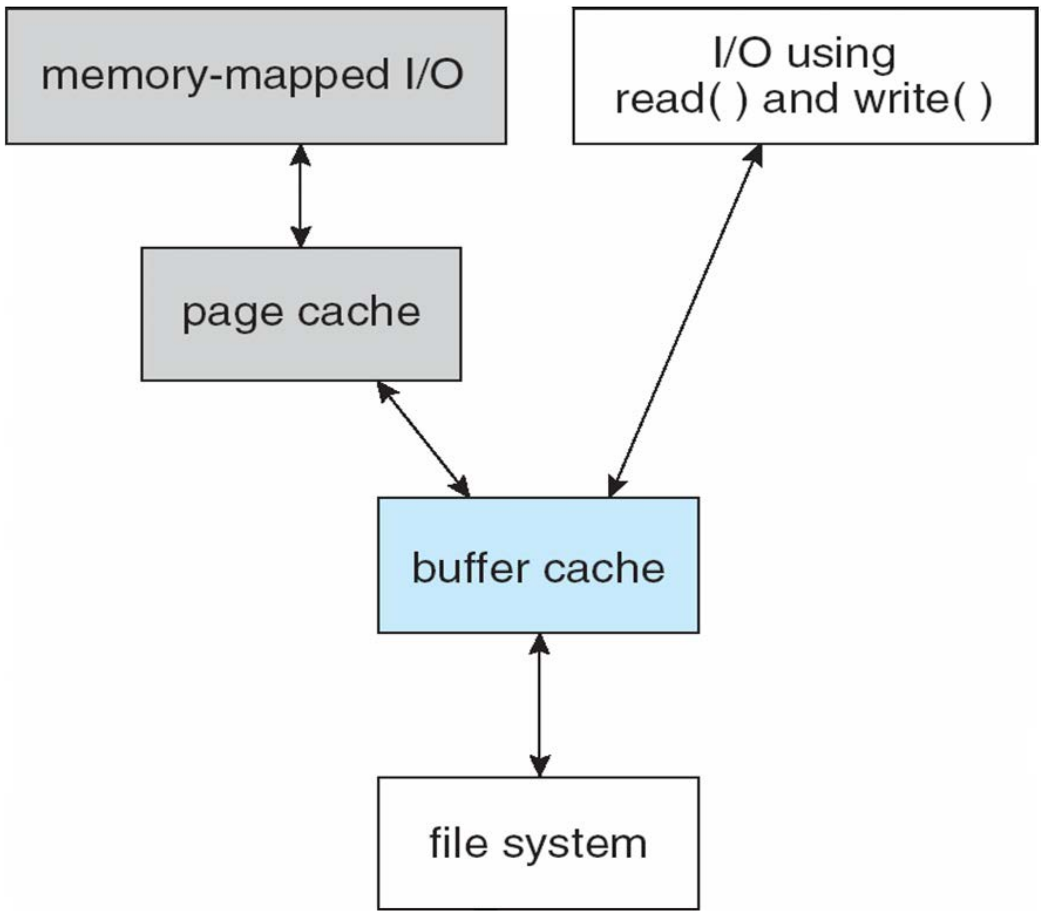
- Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
  - Keep address of first free block and count of following free blocks
  - Free space list then has entries containing addresses and counts

Performance

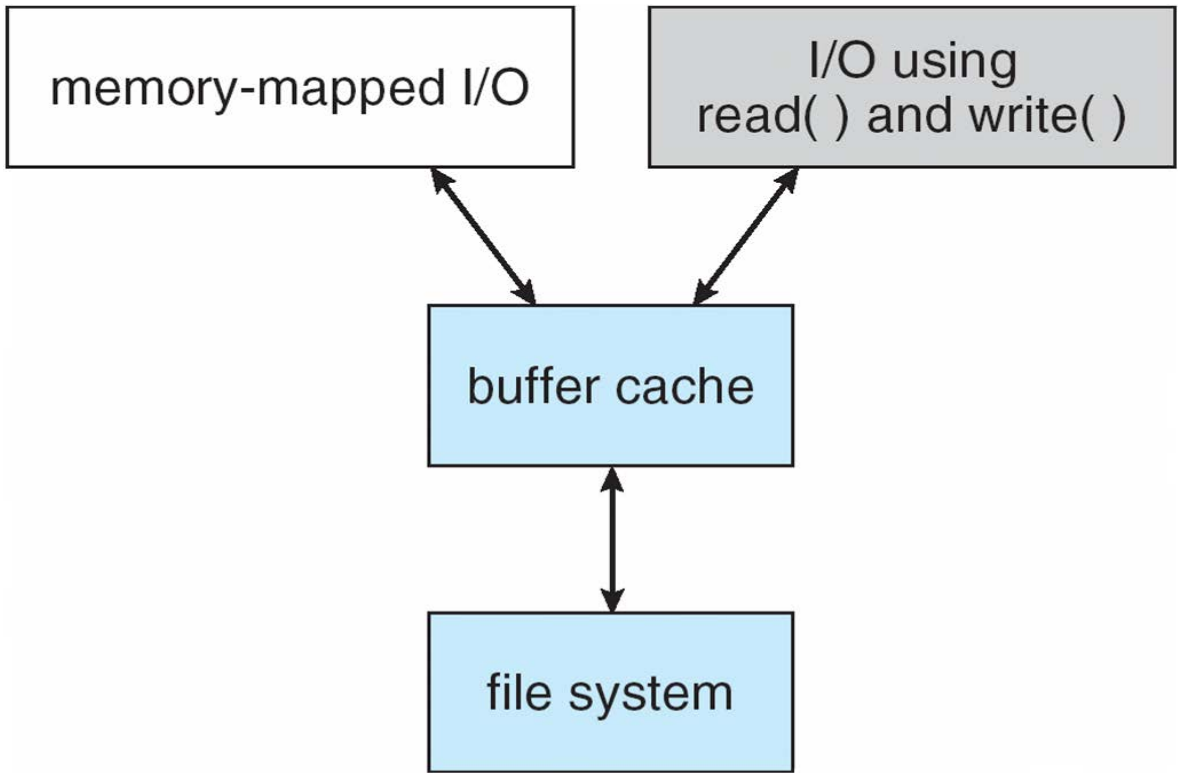
➢ Performance

- Keeping data and metadata close together
- Buffer cache
  - Separate section of main memory for frequently used blocks
- Synchronous writes sometimes requested by apps or needed by OS
  - No buffering / caching – writes must hit disk before acknowledgement
  - Asynchronous writes more common, buffer-able, faster
- Free-behind and read-ahead
  - Techniques to optimize sequential access
  - Reads frequently slower than writes

Unified vs. non-unified buffer cache



I/O Without a Unified Buffer Cache
Wastes mem+CPU+IO



I/O Using a Unified Buffer Cache
Solaris, Linux, Windows

Failure Recovery

1) Consistency checking

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

- 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

2) Log structured file systems

- Similar to database log-based recovery algorithm
- Log structured (or journaling) file systems record each metadata update to the file system as a transaction
- All transactions are written to a log
- Transactions can be
  - Committed
  - Rolled back
- If the file system crashes, all remaining transactions in the log must still be performed
- Where logs should be stored?

3) Backup and restore

- Back up data from disk to another storage device, such as a magnetic tape or other hard disk

- Recovery from the loss of an individual file, or of an entire disk, may then be a matter of restoring the data from backup

- A typical backup schedule:
  - Day 1. full backup: copy all files from the disk to a backup medium.
  - Day 2. incremental backup: copy all files changed since day 1 to another medium.
  - Day 3. incremental backup: copy all files changed since day 2 to another medium.
  - ...
  - Day N. incremental backup: copy all files changed since day N-1 to another medium. Then go back to day 1

Network File System

The Sun Network File System (NFS)

- An implementation & 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

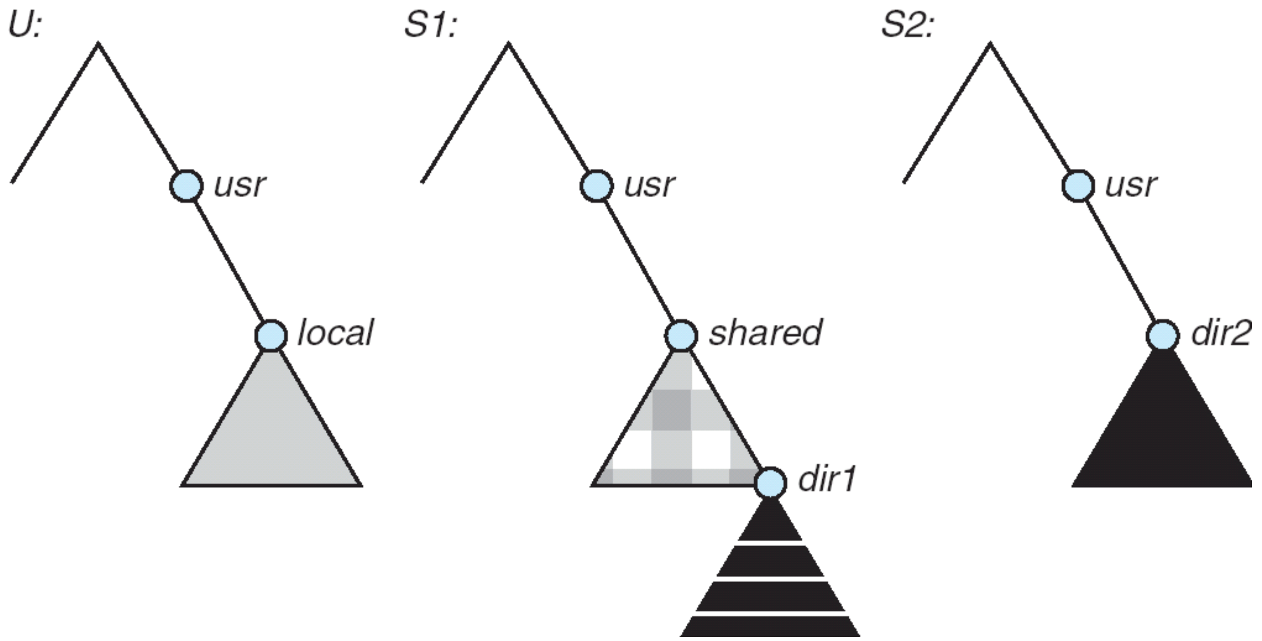
- A remote directory is mounted over a local file system directory
  - The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory

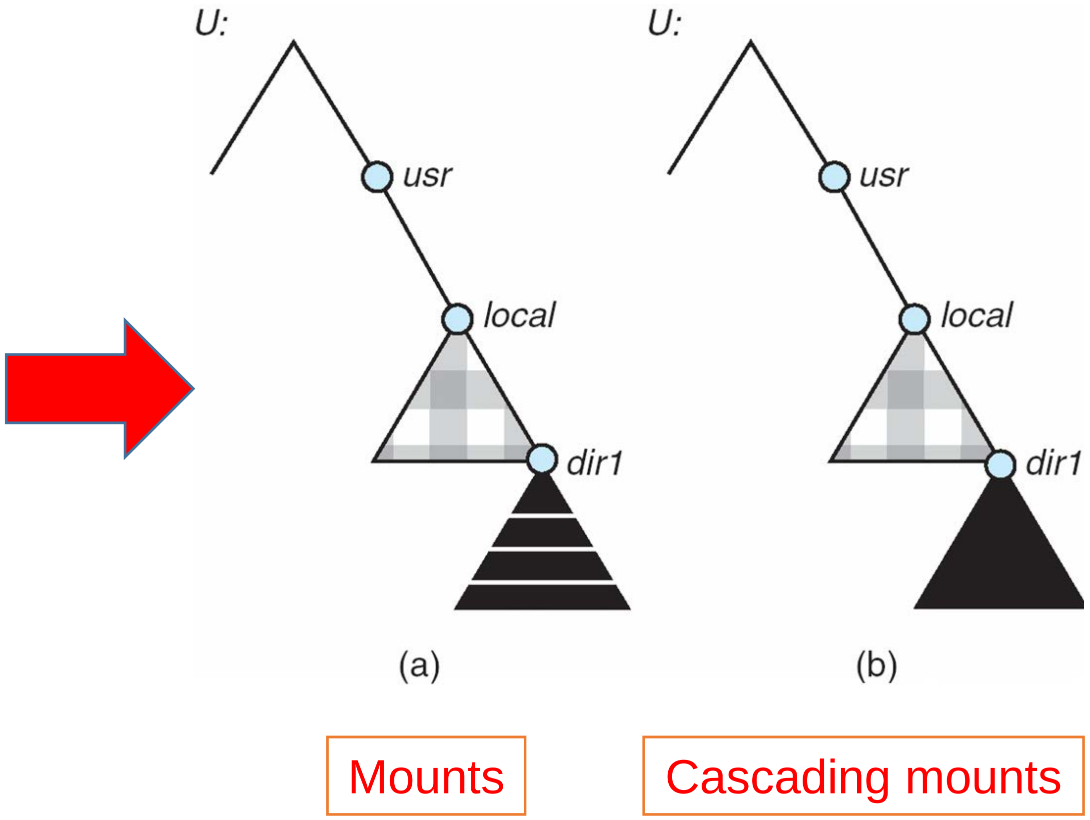
- Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
  - Files in the remote directory can then be accessed in a transparent manner

- 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

- 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

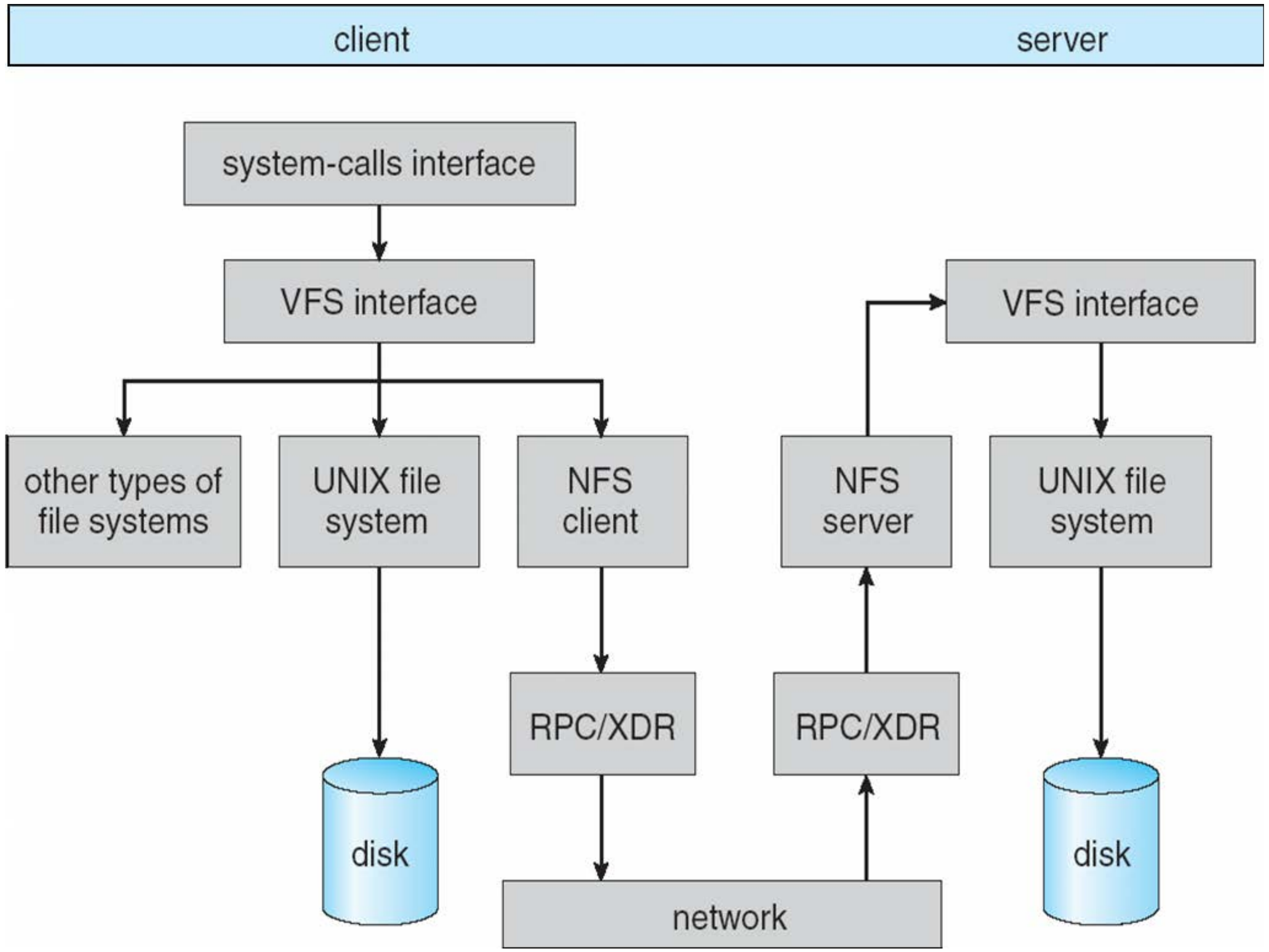
Three independent file systems & mounting





Schematic view of NFS architecture

- NFS write procedure call is atomic
- NFS servers are state-less (why?)
- No open() and close() routines
- All routines are RPC for robustness
- Writes are synchronous (NVM used)



Questions?