Operating Systems 2023 Spring Term
Week 14-2

Dr. Emrah İnan (emrahinan@iyte.edu.tr)

File System Implementation

June 8, 2023

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

Disk provides in-place rewrite and random access

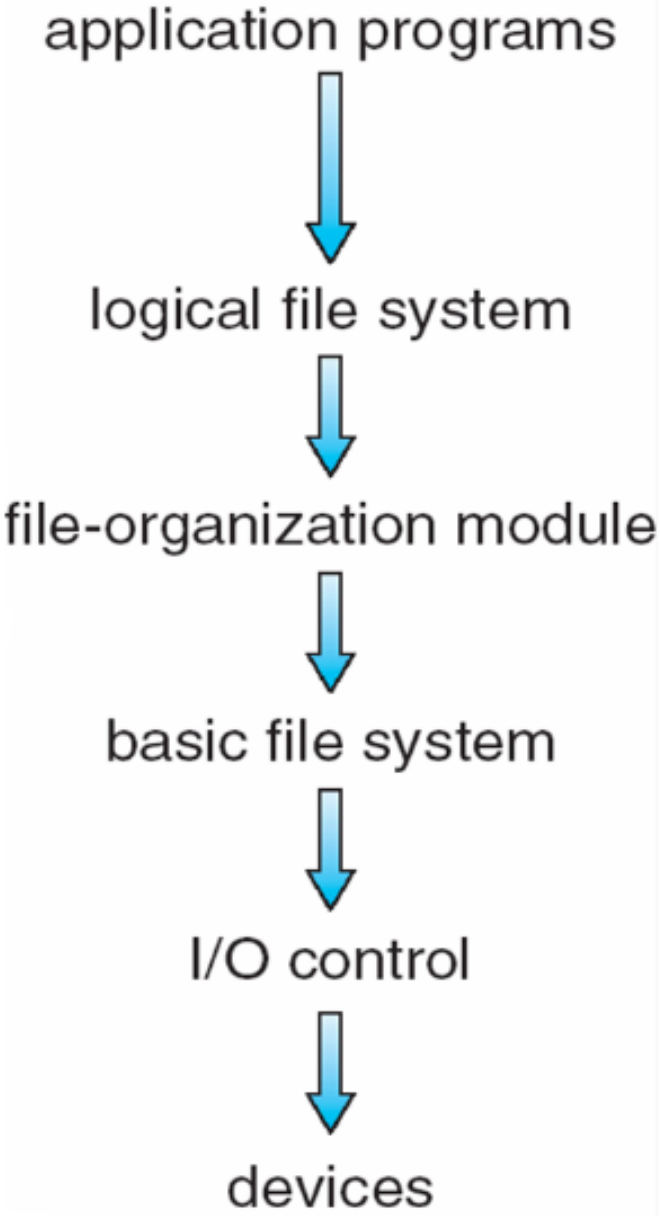
- I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)

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

Device driver controls the physical device

File system organized into layers

Layered File System



File System Layers I

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

Basic file system given command like “retrieve block 123” translates to device driver

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

File System Layers II

- 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
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
  Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)
  Logical layers can be implemented by any coding method according to OS designer

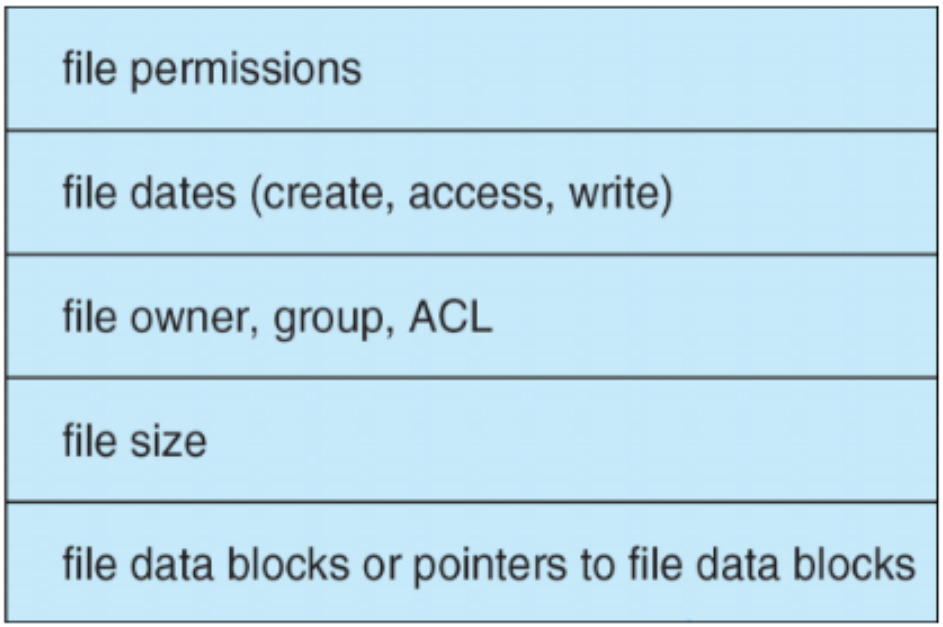
File-System Implementation I

- 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)
- Directory structure organizes the files (Names and inode numbers, master file table)

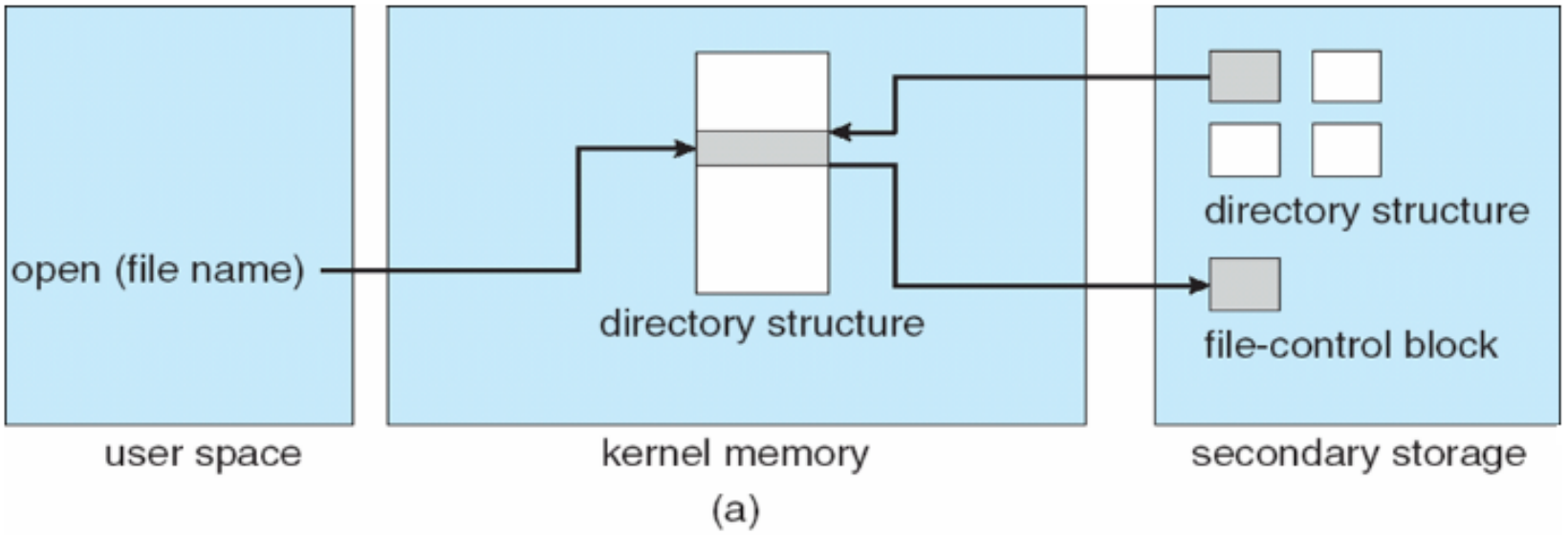
File-System Implementation II

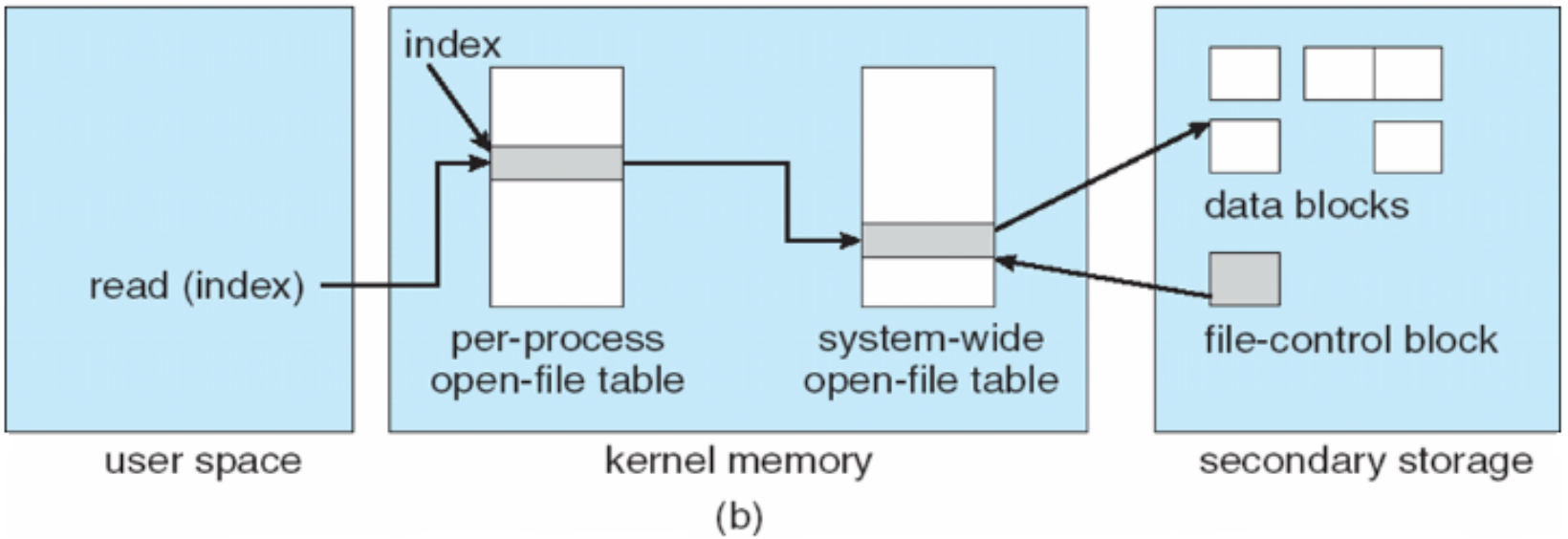
Per-file **File Control Block (FCB)** contains many details about the file

- inode number, permissions, size, dates
- NFTS stores into in master file table using relational DB structures



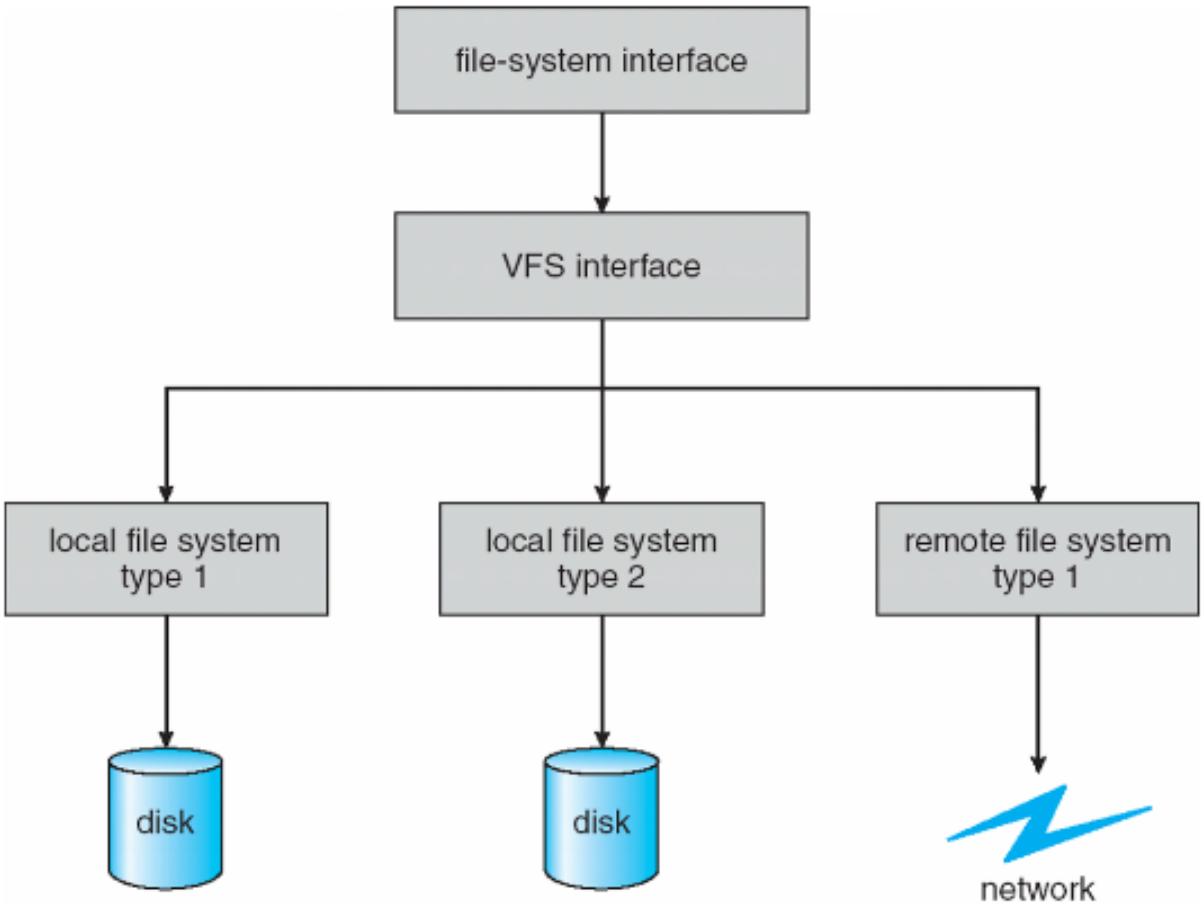
In-Memory File System Structures





Virtual File Systems

The API is to the VFS interface, rather than any specific type of file system



Virtual File System Implementation

For example, Linux has four object types:

inode, file, superblock, dentry

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
- For example:
- • `int open(. . .)`—Open a file
- • `int close(. . .)`—Close an already-open file
- • `ssize t read(. . .)`—Read from a file
- • `ssize t write(. . .)`—Write to a file
- • `int mmap(. . .)`—Memory-map a file

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

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

Allocation Methods - Contiguous

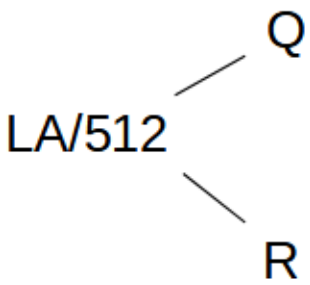
An allocation method refers to how disk blocks are allocated for files:

Contiguous allocation – each file occupies set of contiguous blocks

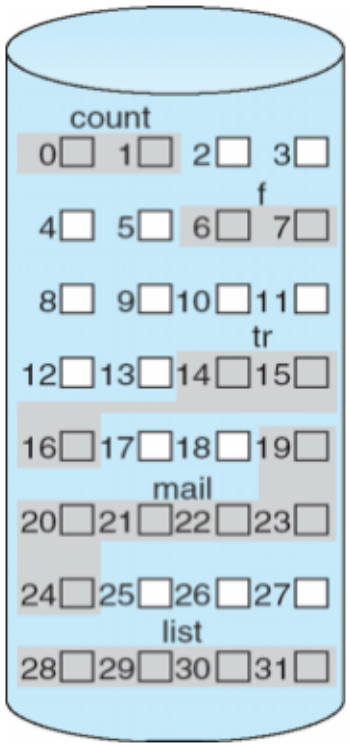
- Best performance in most cases
- Simple – only starting location (block #) and length (number of blocks) are required
- Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

Mapping from logical to physical



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

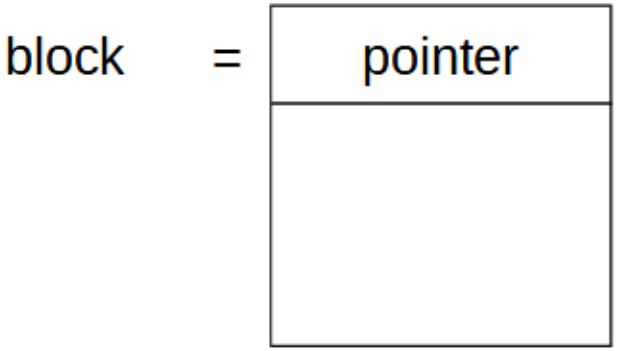


directory

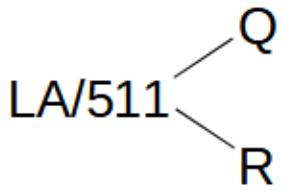
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked Allocation I

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



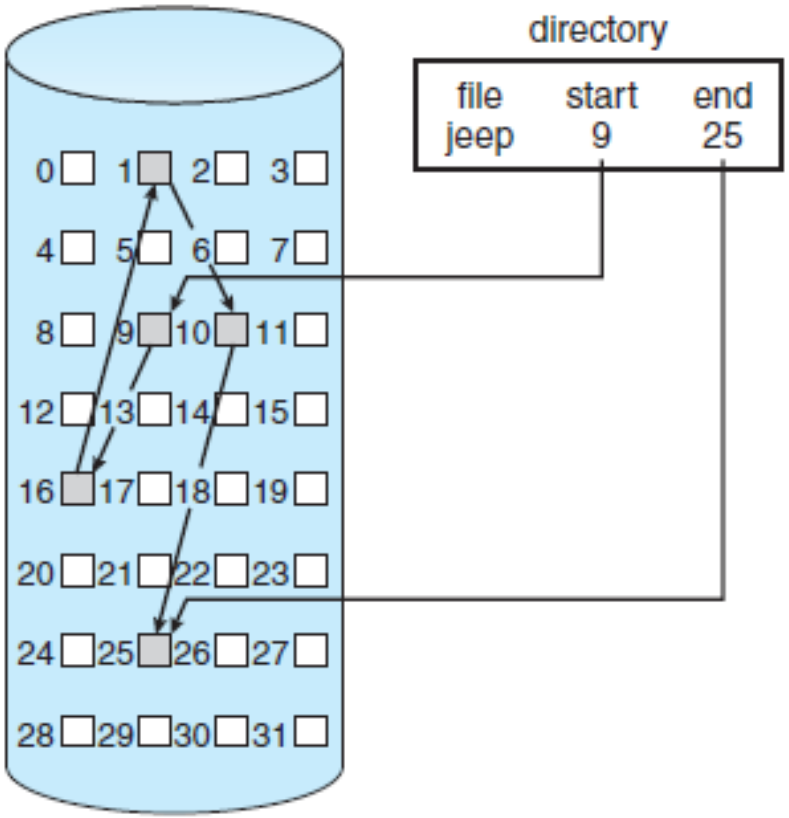
Mapping



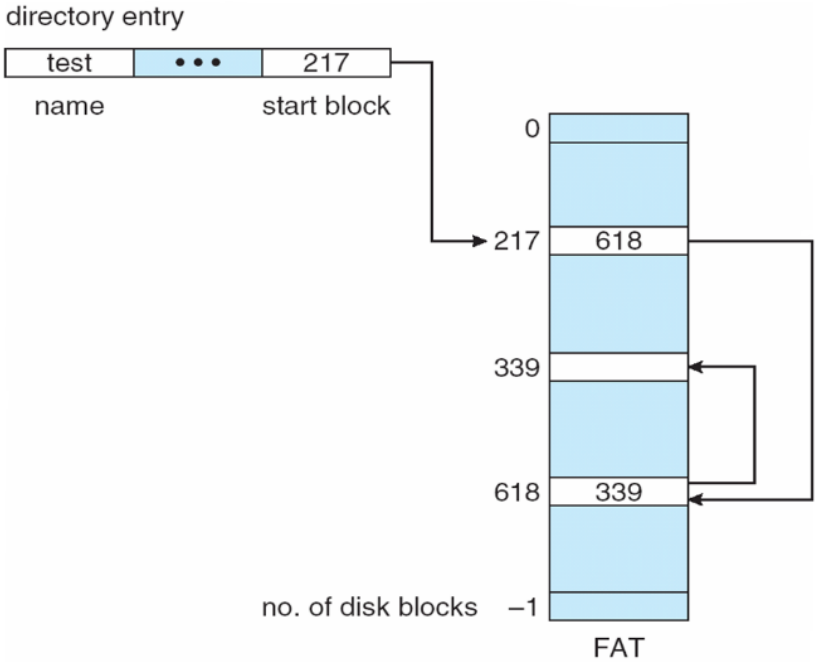
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

Linked Allocation II



File-Allocation Table



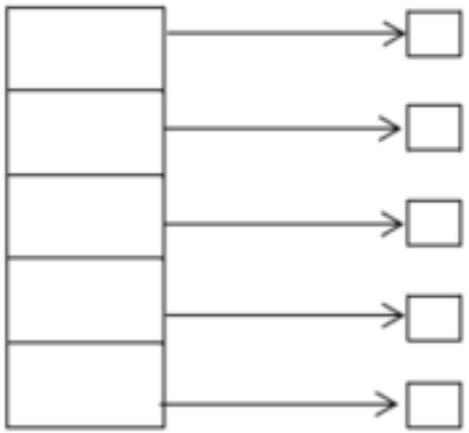
An important variation on linked allocation is the use of a file-allocation table (FAT). This simple but efficient method of disk-space allocation was used by the MS-DOS operating system. A section of storage at the beginning of each volume is set aside to contain the table. The table has one entry for each block and is indexed by block number. The FAT is used in much the same way as a linked list.

Allocation Methods - Indexed

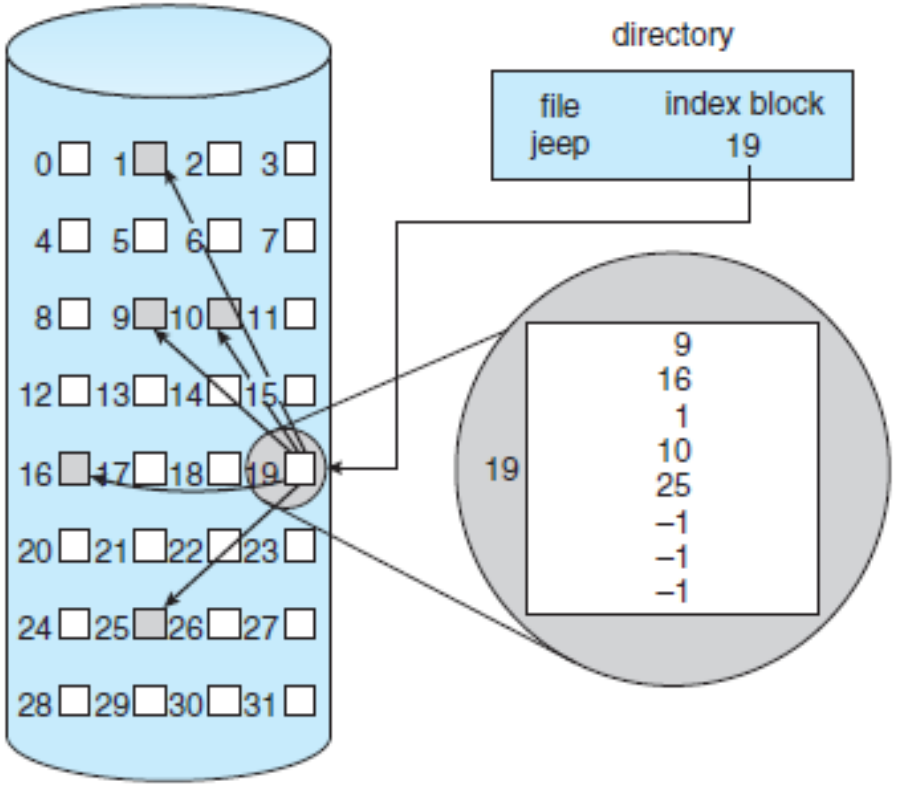
Indexed allocation

Each file has its own **index block**(s) of pointers to its data blocks

Logical view



Example of Indexed Allocation



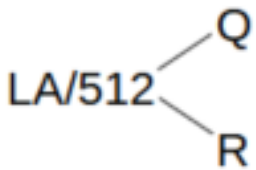
Indexed Allocation

Need index table

Random access

Dynamic access without external fragmentation, but have overhead of index block

Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table

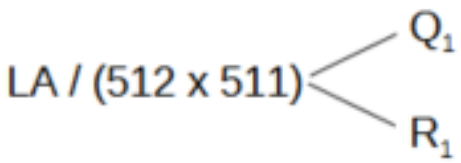


Q = displacement into index table
R = displacement into block

Indexed Allocation – Mapping I

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)



Q_1 = block of index table

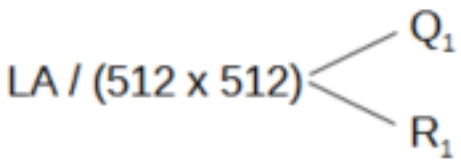
R_1 is used as follows:

Q_2 = displacement into block of index table

R_2 displacement into block of file:

Indexed Allocation – Mapping II

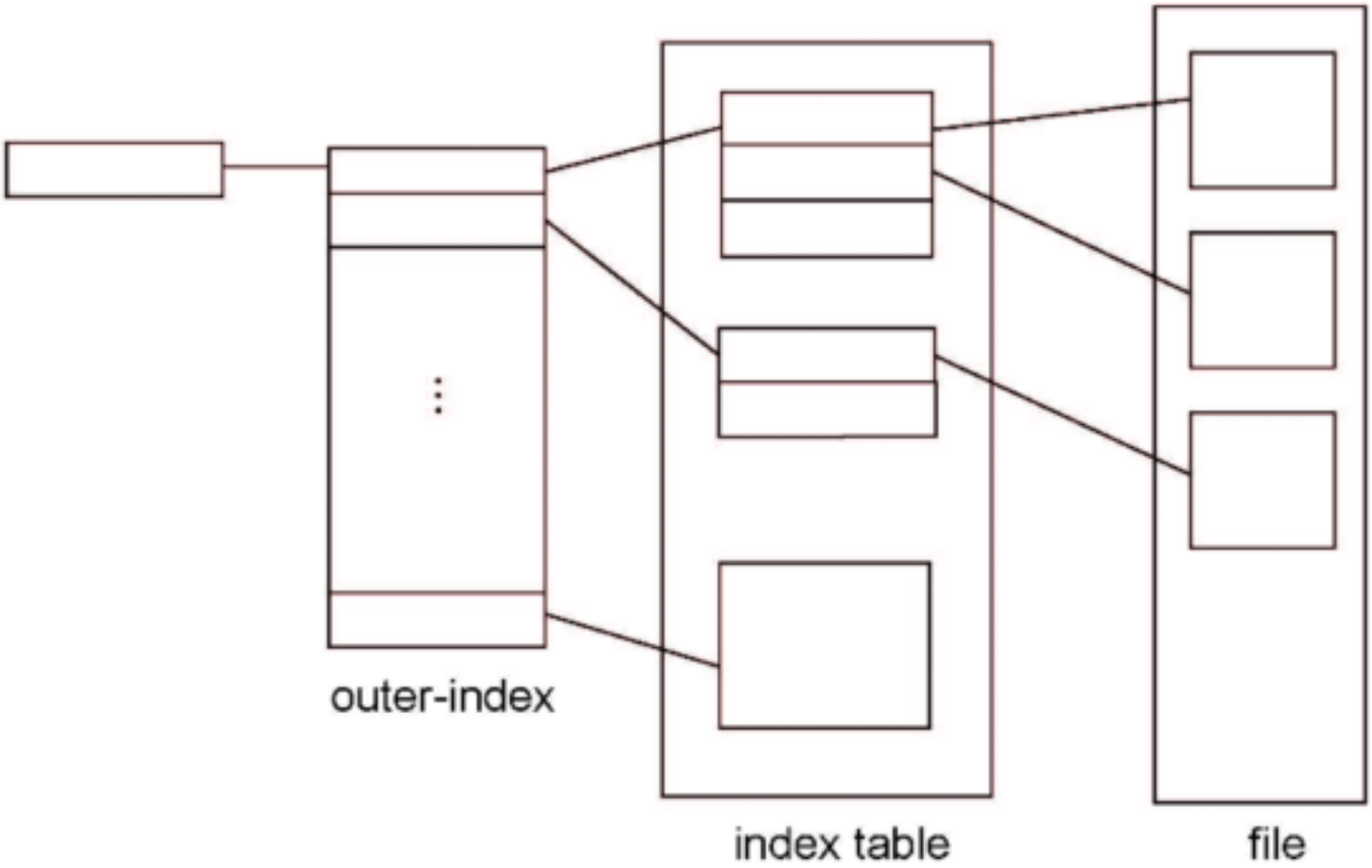
Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)



Q_1 = displacement into outer-index
R_1 is used as follows:

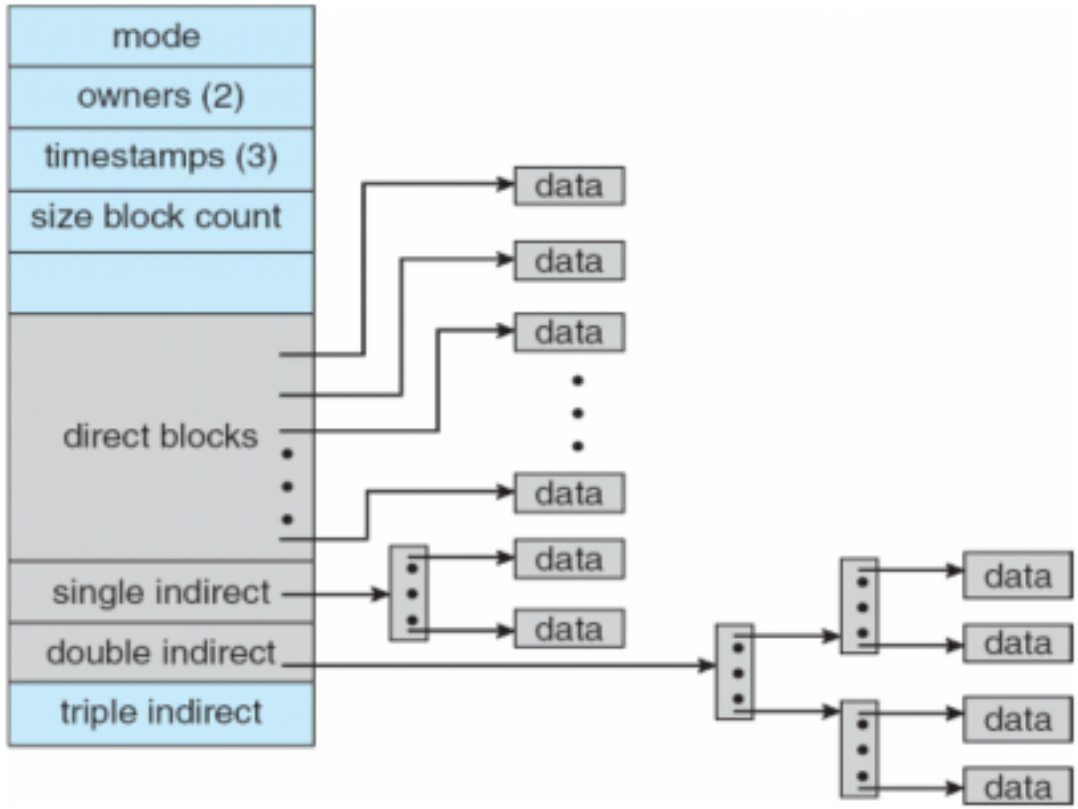
Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping III



Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses

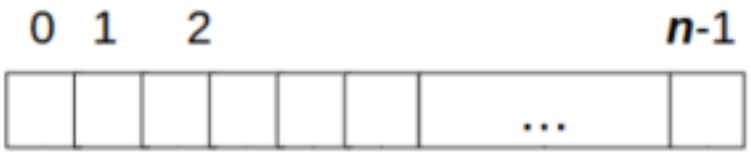


More index blocks than can be addressed with 32-bit file pointer

Performance

- Adding instructions to the execution path to save one disk I/O is reasonable
  Intel Core i7 Extreme Edition 990x (2011) at 3.46Ghz = 159,000 MIPS
  http : *//en.wikipedia.org/wiki/Instructions_per_second*
  Typical disk drive at 250 I/Os per second
  159,000 MIPS / 250 = 630 million instructions during one disk I/O
  Fast SSD drives provide 60,000 IOPS
  159,000 MIPS / 60,000 = 2.65 millions instructions during one disk I/O

Free-Space Management I



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

Block number calculation

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

CPUs have instructions to return offset within word of first “1” bit

Free-Space Management II

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)

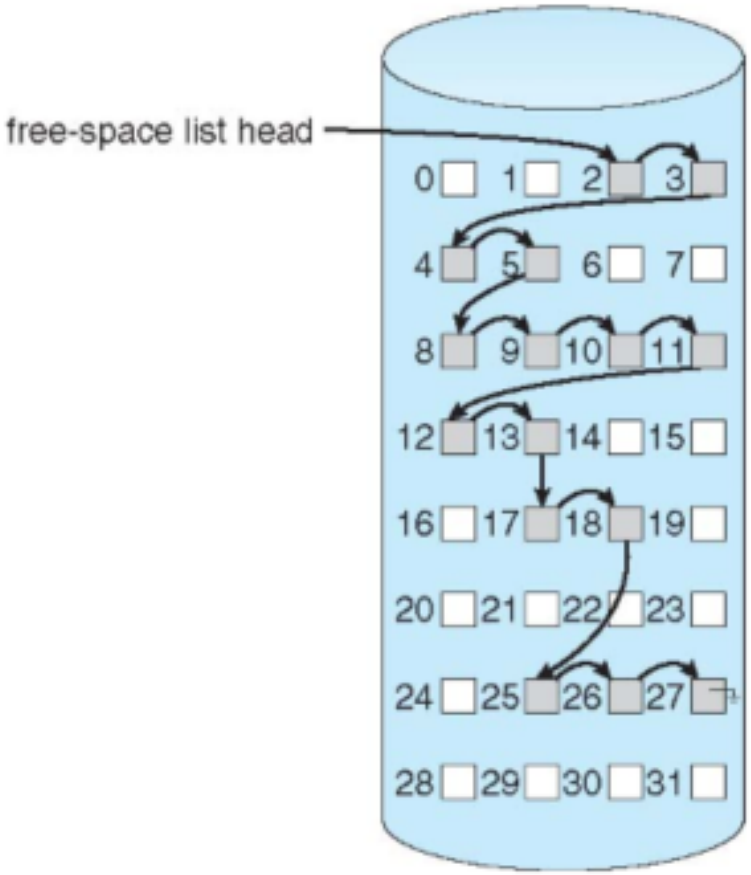
if clusters of 4 blocks -> 8MB of memory

Easy to get contiguous files

Linked Free Space List on Disk

Linked list (free list)

- Cannot get contiguous space easily
- No waste of space
- No need to traverse the entire list (if # free blocks recorded)

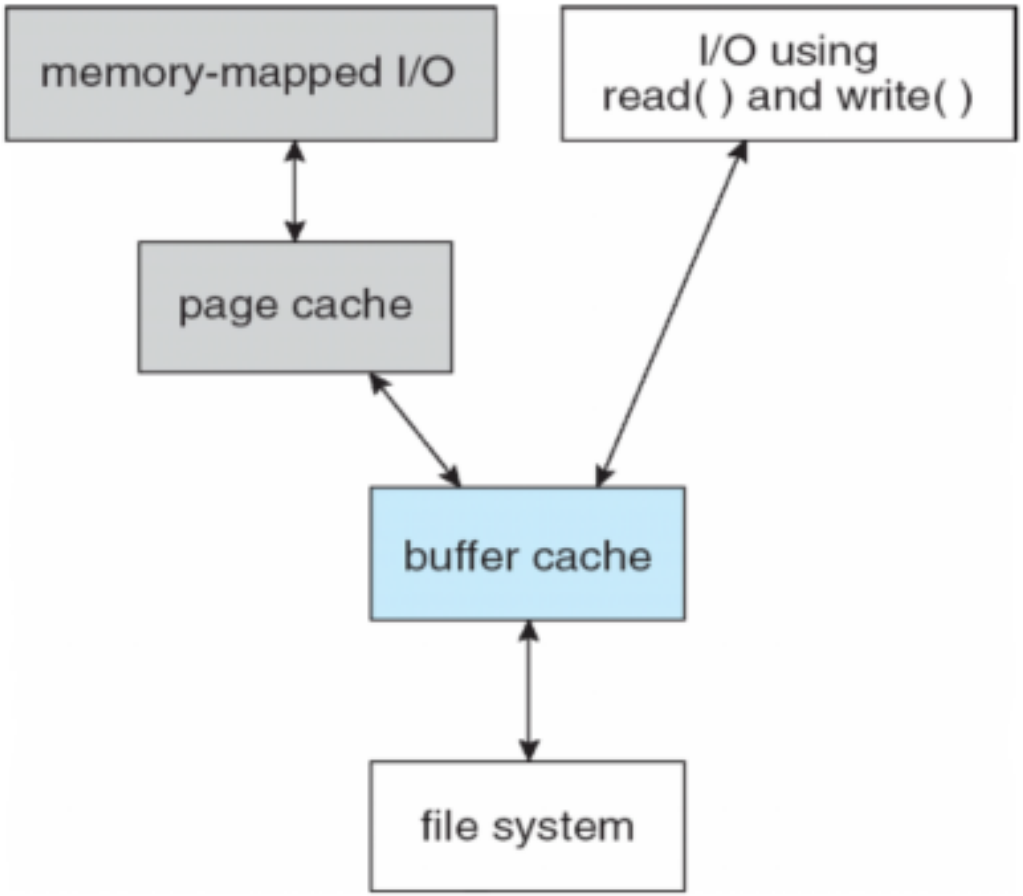


Efficiency and Performance

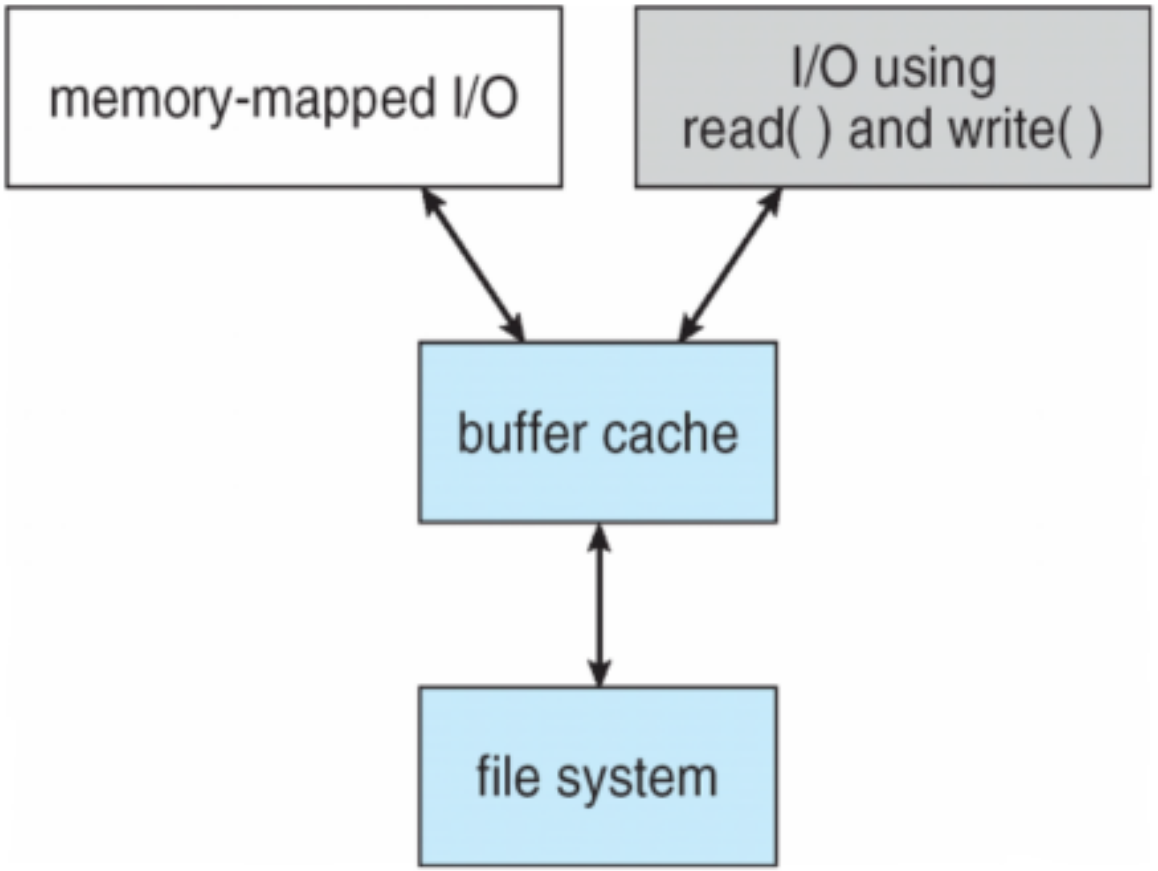
Performance

- 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

I/O Without a Unified Buffer Cache



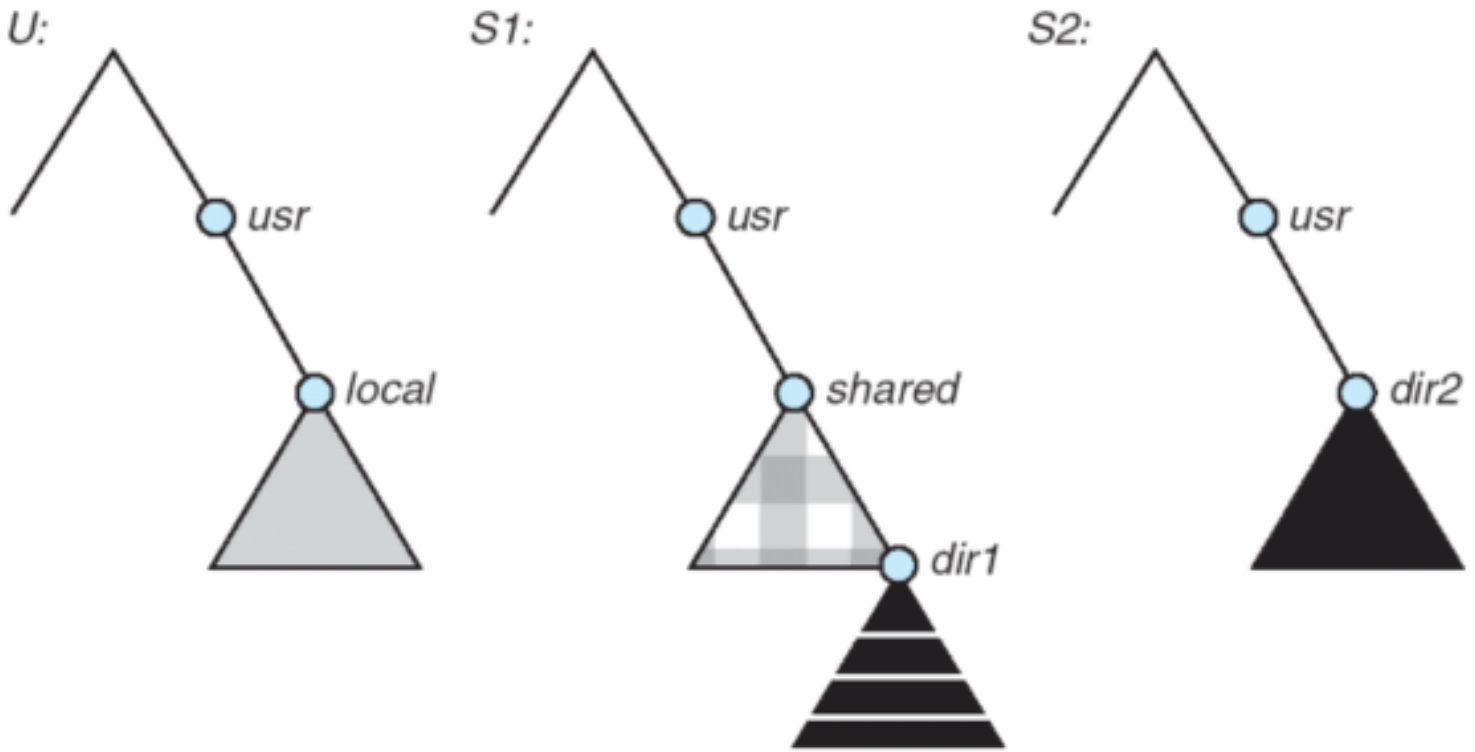
I/O Using a Unified Buffer Cache



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)

Three Independent File Systems



Schematic View of NFS Architecture