



DSTN Session #2

Operating System Basics

Agenda

2

- Quick recap of Session #1
- File Systems
- Storage Subsystems
- Device Drivers

Recap of Session #1

3

Learnt about:

Basics of Networking Layers (Encapsulation)

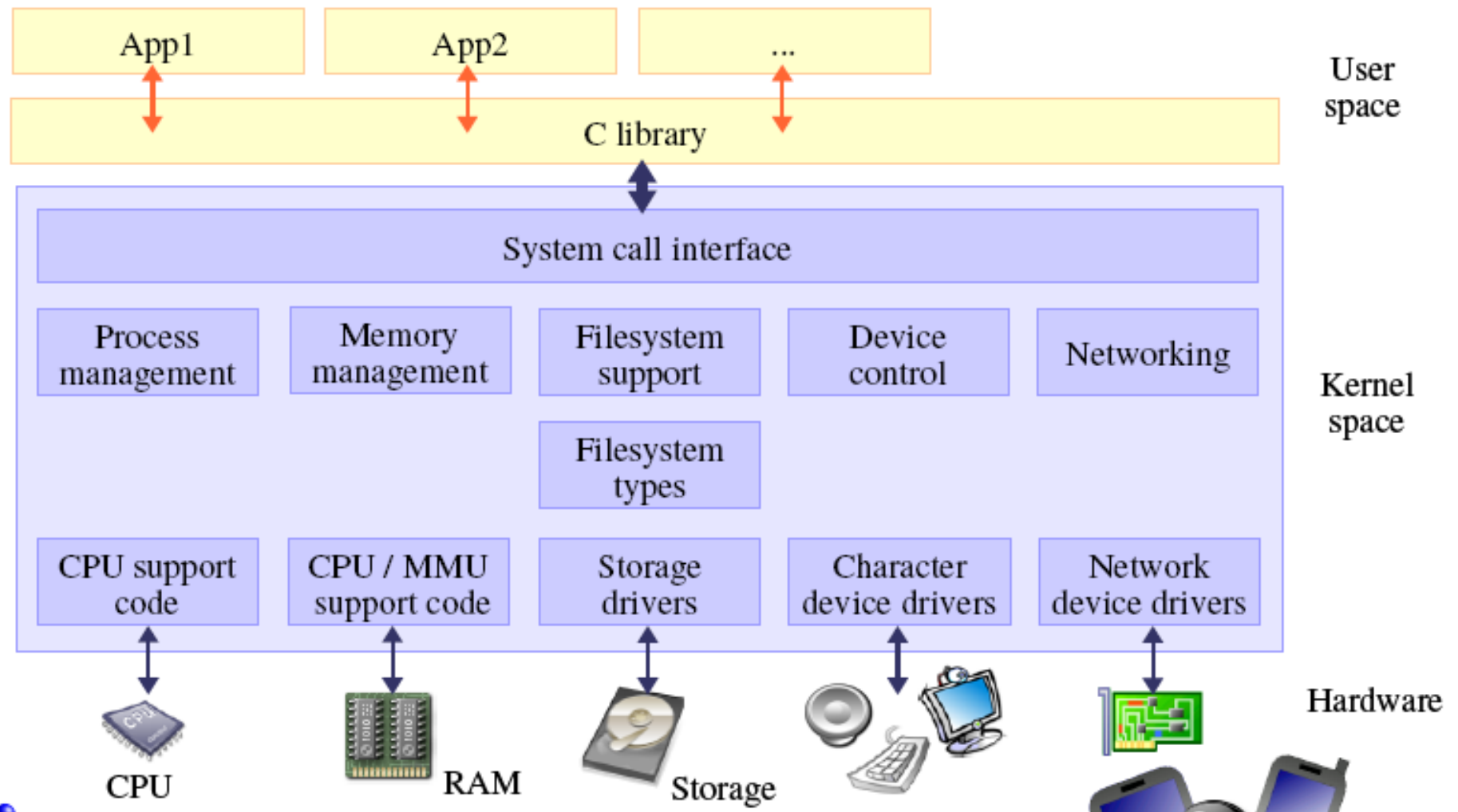
Transport Layer Basics (TCP and UDP Headers)

Network Layer Basics (IP Header)

Data Link Layer Basics (Ethernet & CSMA/CD)

Operating Systems

UNIX Kernel Architecture



File Systems

Concept and Structure of a File

77

- Contiguous logical address space
- Types:
 - Data
 - numeric
 - character
 - binary
 - Program
- None - sequence of words, bytes
- Simple record structure
 - Lines
 - Fixed length
 - Variable length
- Complex Structures
 - Formatted document
 - Re-locatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
 - Operating system
 - Program

Attributes of a File

- **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 Operation

- File is an **abstract data type**. Typical operations are:
 - **Create**
 - **Write**
 - **Read**
 - **Reposition within file**
 - **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

What is needed to 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

What is File System?

- The file system is a mechanism for storing data and programs.
- Main characteristics of Secondary storage
 - Large storage capacity
 - Non-volatile nature.
- Consequence?
 - Give raise to the convenience of storing data and programs on secondary storage.

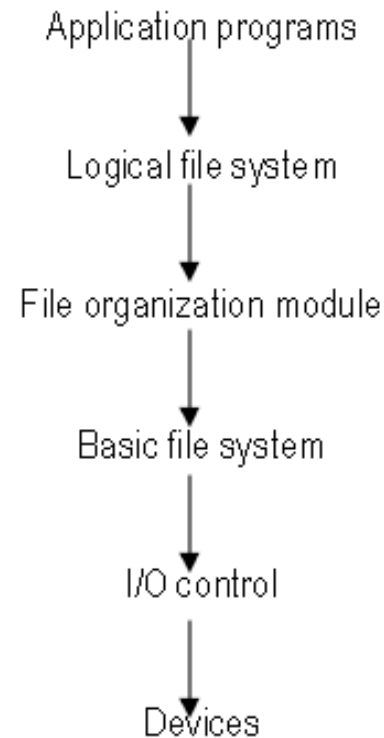
What do you Require of File Systems?

- Naming
 - Should be flexible, e.g., allow multiple names for same files
 - Support hierarchy for easy of use
- Persistence
 - Want to be sure data has been written to disk in case crash occurs
- Sharing/Protection
 - Want to restrict who has access to files
 - Want to share files with other users
- Speed & Efficiency for different access patterns
 - Sequential access
 - Random access
 - Sequential is most common & Random next
 - Other pattern is Keyed access (not usually provided by OS)

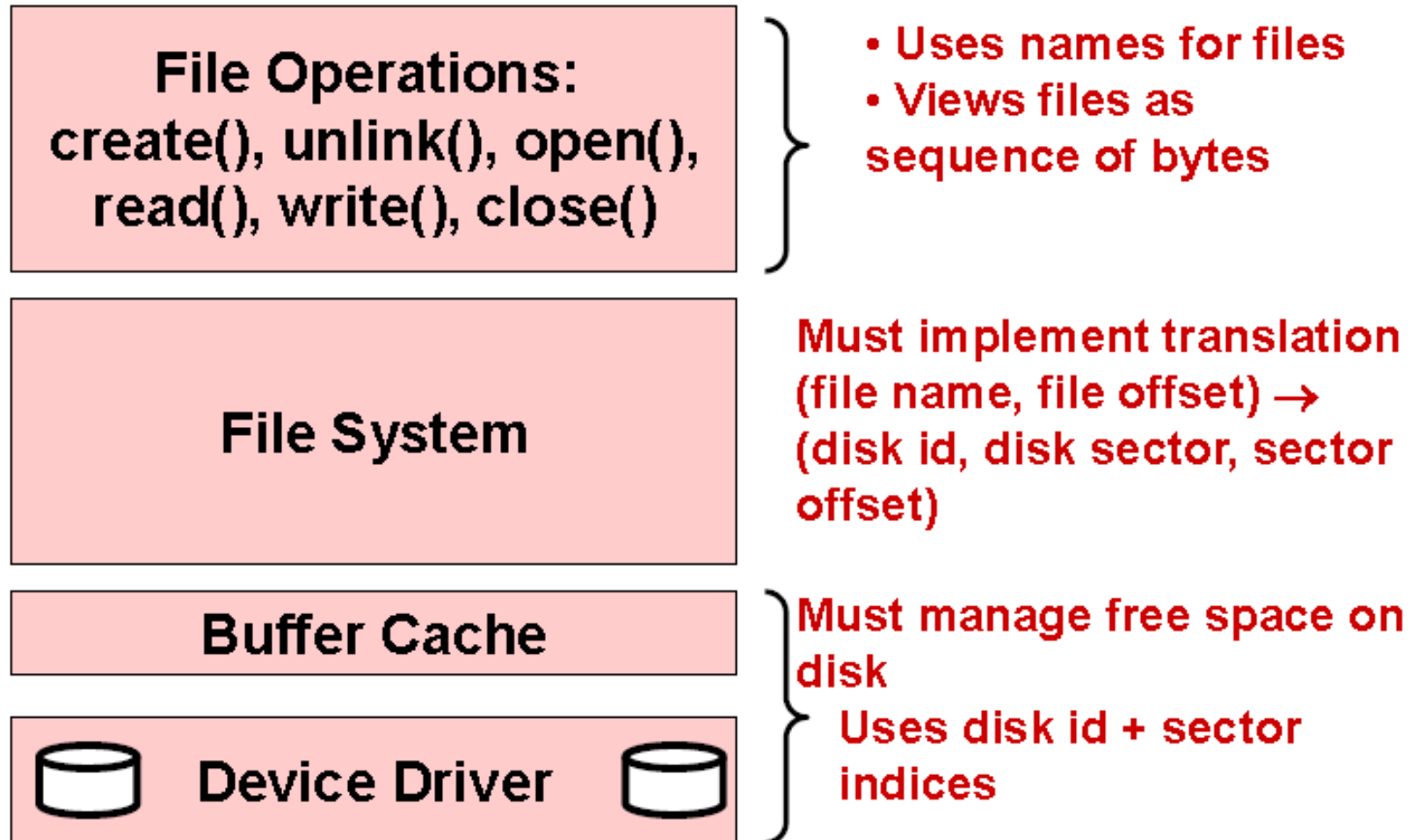
What do you Require of File Systems?

- Minimum Space Overhead
 - Disk space needed to store metadata is lost for user data
- *Twist: all metadata that is required to do translation must be stored on disk*
 - Translation scheme should minimize number of additional accesses for a given access pattern
 - Harder than, say page tables where we assumed page tables themselves are not subject to paging!

File System Organization

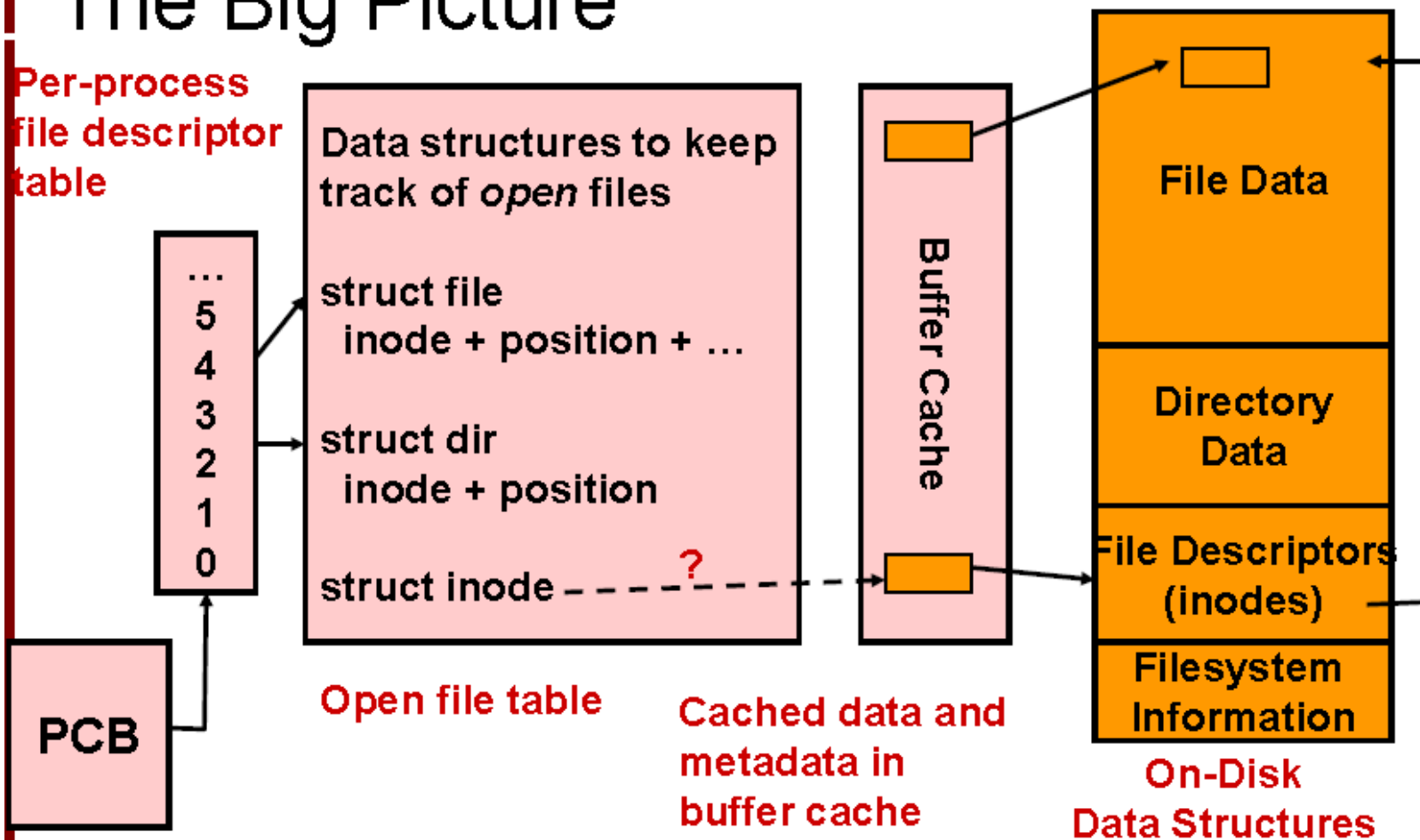


Over-View



The Big Picture

Per-process
file descriptor
table



Steps in Opening and Reading a File

- Look-up (via-directory)
 - Find on disk's file descriptors block number
- Find entry in open file table.
 - Create one if none, else increment ref count
- Find where file data is located
 - By reading on-disk file descriptor
- Read data & return to user

Open File Table

- inode –represents file
 - at most 1 in-memory instance per unique file
 - #number of openers & other properties
- file–represents one or more processes using an file
 - With separate offsets for byte-stream
- dir–represents an open directory file
- Generally:
 - None of data in OFT is persistent
 - Reflects how processes are currently using files
 - Lifetime of objects determined by open/close
 - Reference counting is used

File Descriptors (“inodes”)

- Term “inode” can refer to 3 things:
 - 1.in-memory inode
 - -Store information about an open file, such as how many openers, corresponds to on-disk file descriptor
 - 2.on-disk inode
 - -Region on disk, entry in file descriptor table, that stores persistent information about a file -who owns it, where to find its data blocks, etc.
 - 3.on-disk inode, when cached in buffer cache
 - -A bitwise copy of 2. in memory

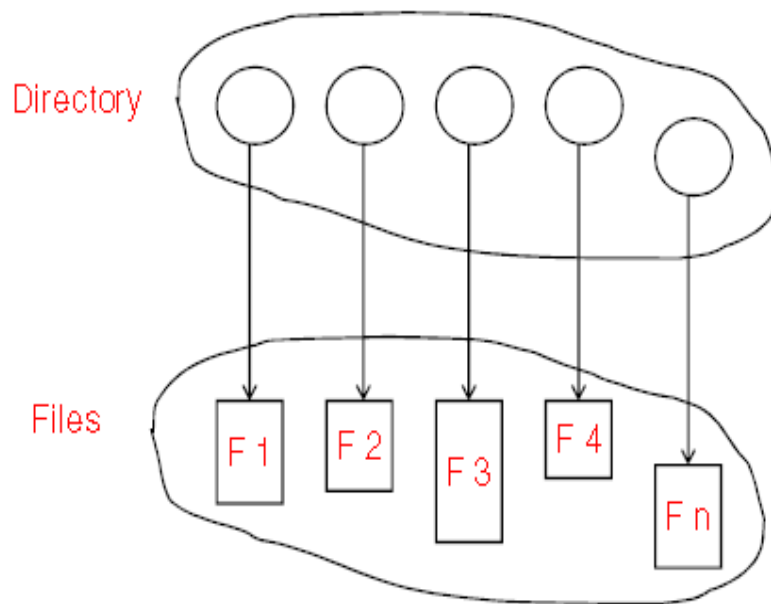
What information does File System contain?

- Contains “super block” stores information such as size of entire file system, etc.
 - Location of file descriptor table & free map
- Free Block Map
 - Bitmap used to find free blocks
 - Typically cached in memory
- Superblock & free map often replicated in different positions on disk



Directory Structure

- Directory is collection of nodes containing information about files.

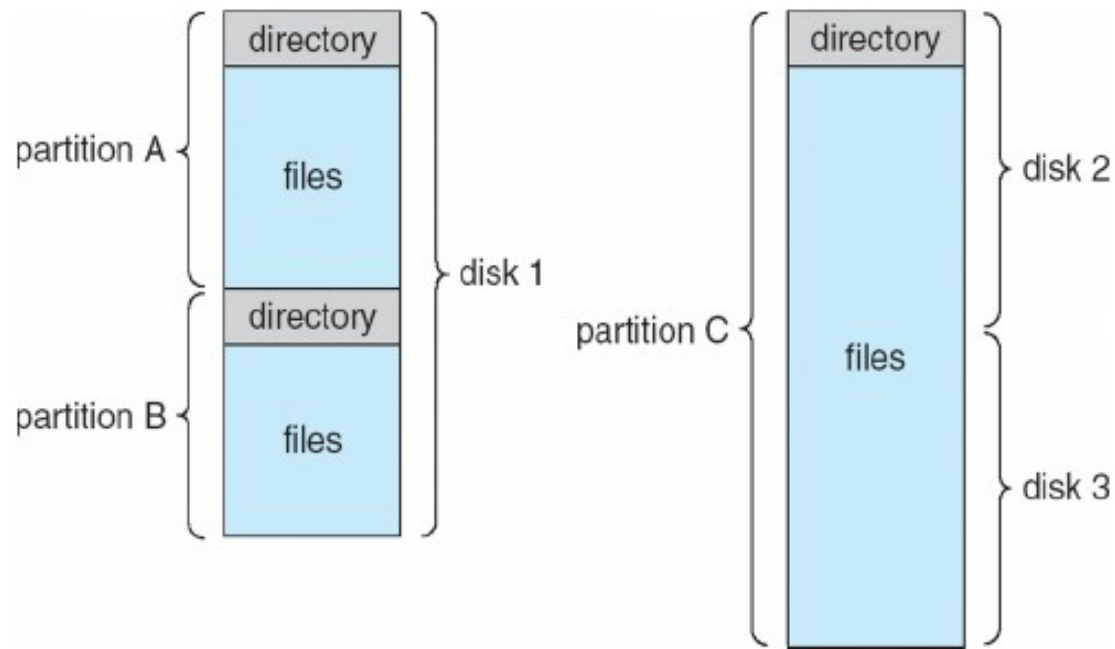


- Both the directory structure and the files reside on disk
- Backups of these two structures are kept on tapes

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, ...)

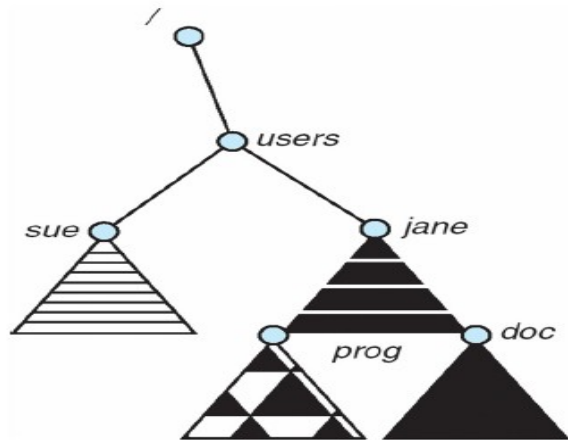
Typical File System Organization



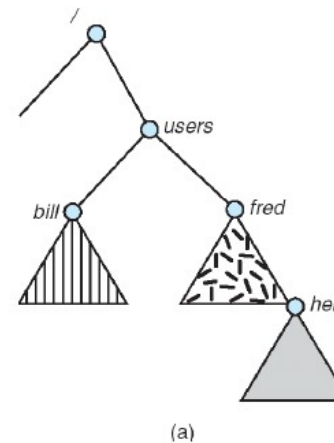
How is File System Mounted?

235

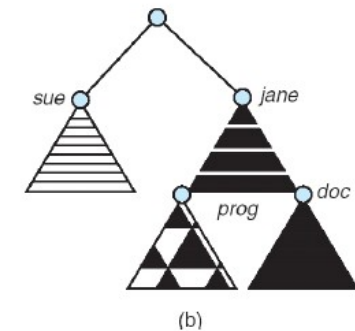
- A file system must be **mounted before it can be accessed**
- A un-mounted file system is mounted at a **mount point**



Mount point



(a) Existing.



(b) Unmounted Partition

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method.
- Multiple Users
 - **User IDs** identify users, allowing permissions and protections to be per-user
 - **Group IDs** allow users to be in groups, permitting group access rights

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
 - Manually via programs like FTP
 - Automatically, seamlessly using **distributed file systems**
 - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
 - Server can serve multiple clients
 - Client and user-on-client identification is insecure or complicated
 - **NFS** is standard UNIX client-server file sharing protocol
 - **CIFS** is standard Windows protocol
 - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as **LDAP, DNS, NIS**, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
 - Similar to process synchronization algorithms
 - Tend to be less complex due to disk I/O and network latency (for remote file systems)
 - Andrew File System (AFS) implemented complex remote file sharing semantics
 - Unix file system (UFS) implements:
 - Writes to an open file visible immediately to other users of the same open file
 - Sharing file pointer to allow multiple users to read and write concurrently
 - AFS has session semantics
 - Writes only visible to sessions starting after the file is closed

Protection

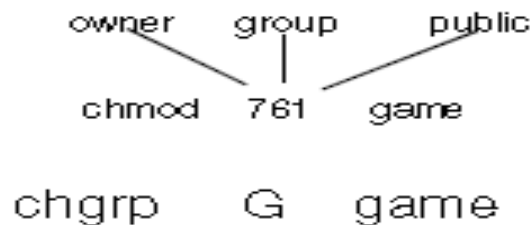
- File owner/creator should be able to control:
 - what can be done
 - by whom
- Types of access
 - **Read**
 - **Write**
 - **Execute**
 - **Append**
 - **Delete**
 - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

a) owner access	7	⇒	RWX 1 1 1 RWX
b) group access	6	⇒	1 1 0 RWX
c) public access	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5 pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx-----	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

File-System Structure

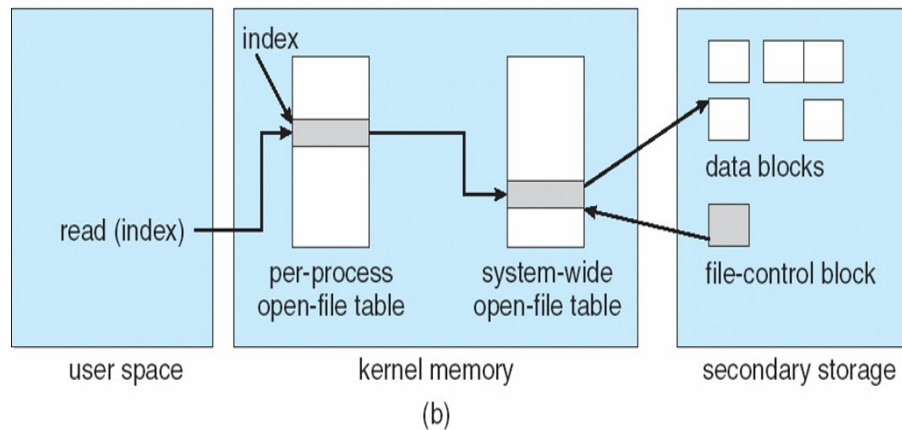
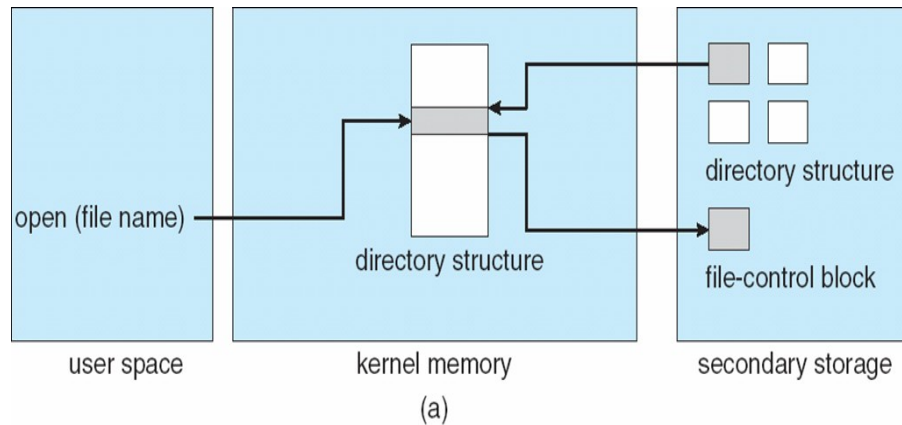
- File structure
 - Logical storage unit
 - Collection of related information
- File system organized into layers
- File system resides on secondary storage (disks)
 - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- File control block – storage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

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

355



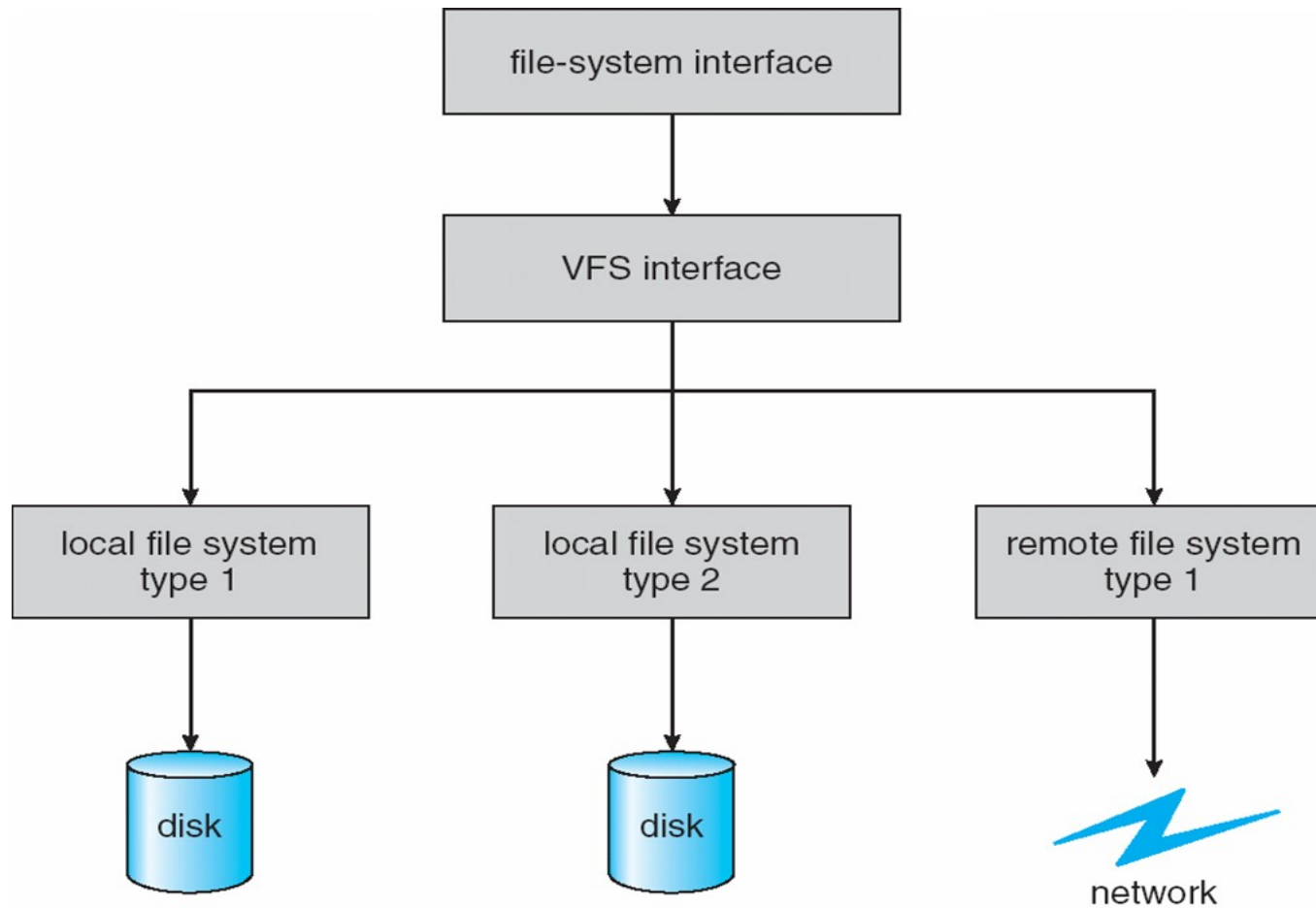
The following figure illustrates the necessary file system structures provided by the operating systems.

Figure (a) refers to opening a file.
Figure (b) refers to reading a file.

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

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

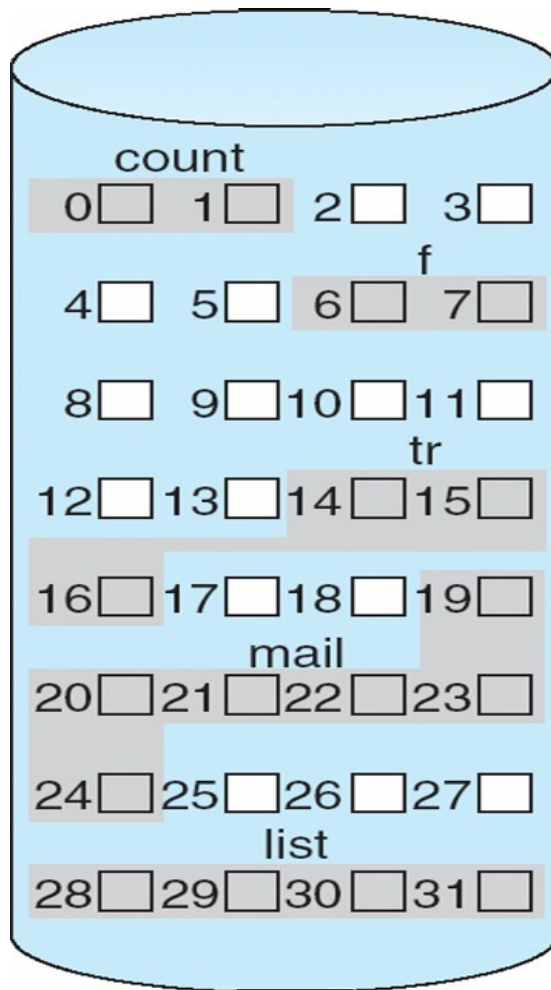
Contiguous Allocation

- Mapping from logical to physical



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

Contiguous Allocation of Disk Space



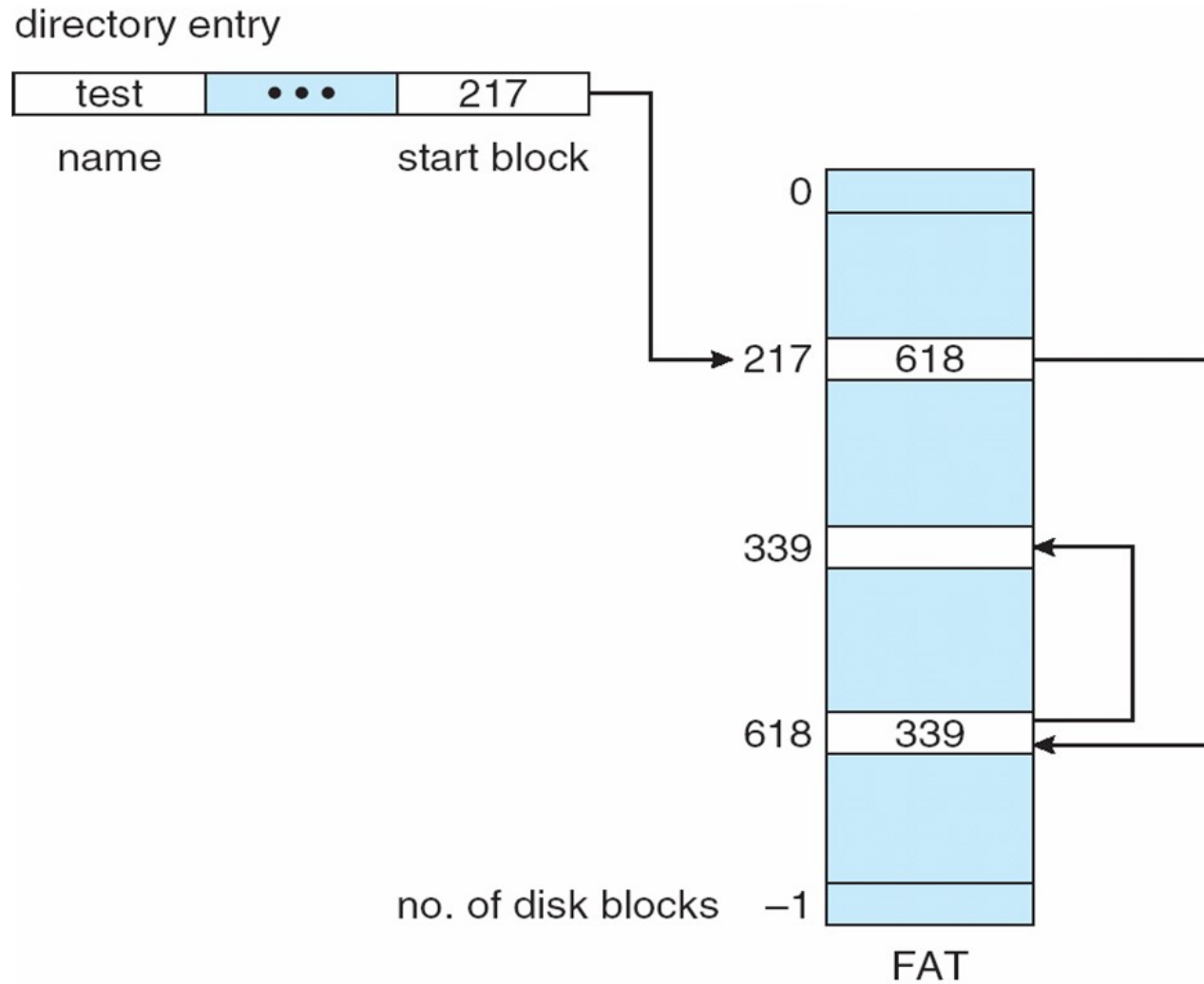
directory

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

Extent-Based Systems

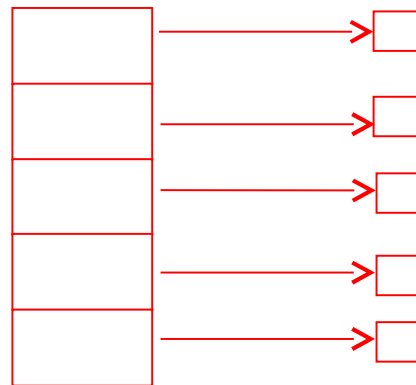
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
 - Extents are allocated for file allocation
 - A file consists of one or more extents

File-Allocation Table



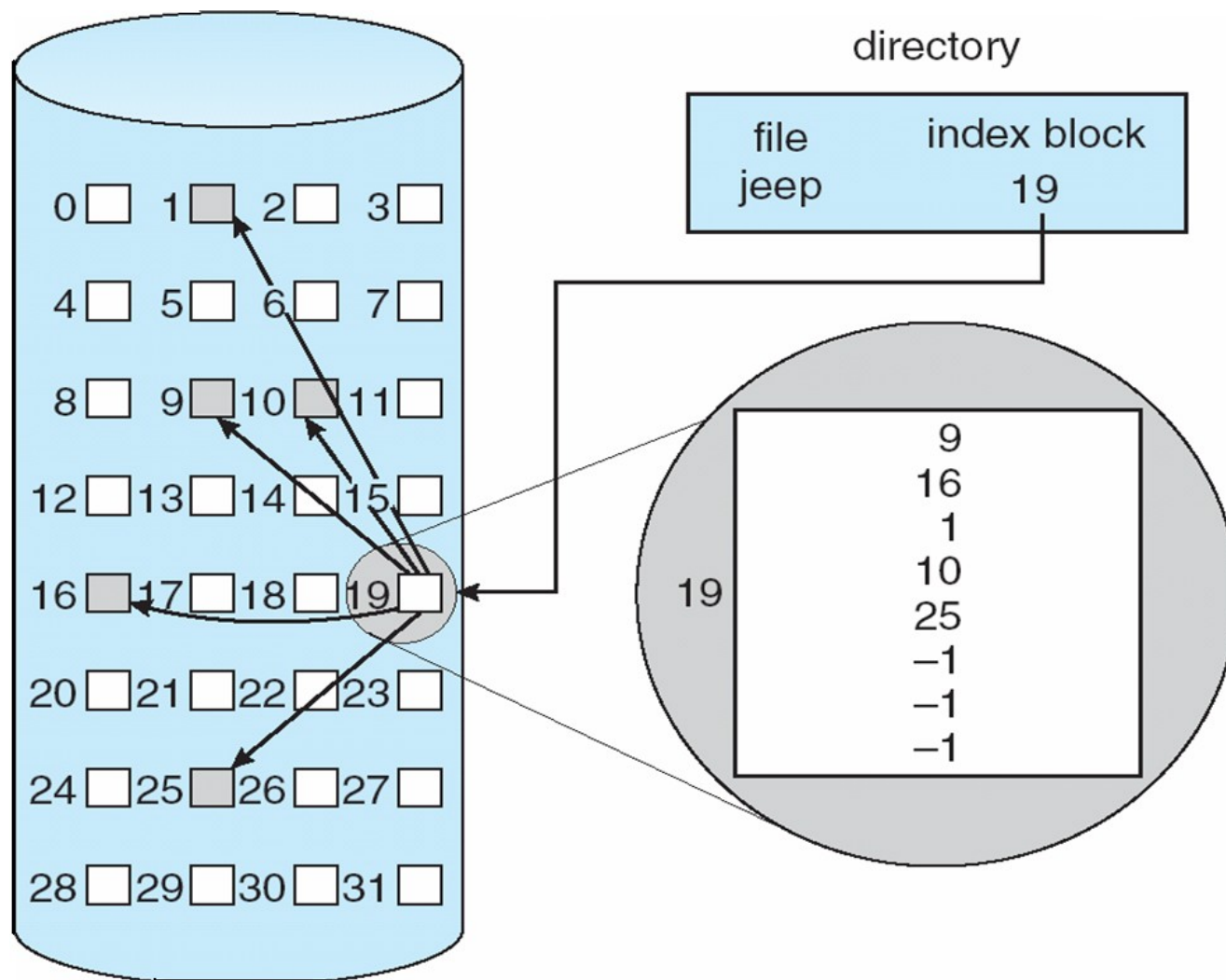
Indexed Allocation

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

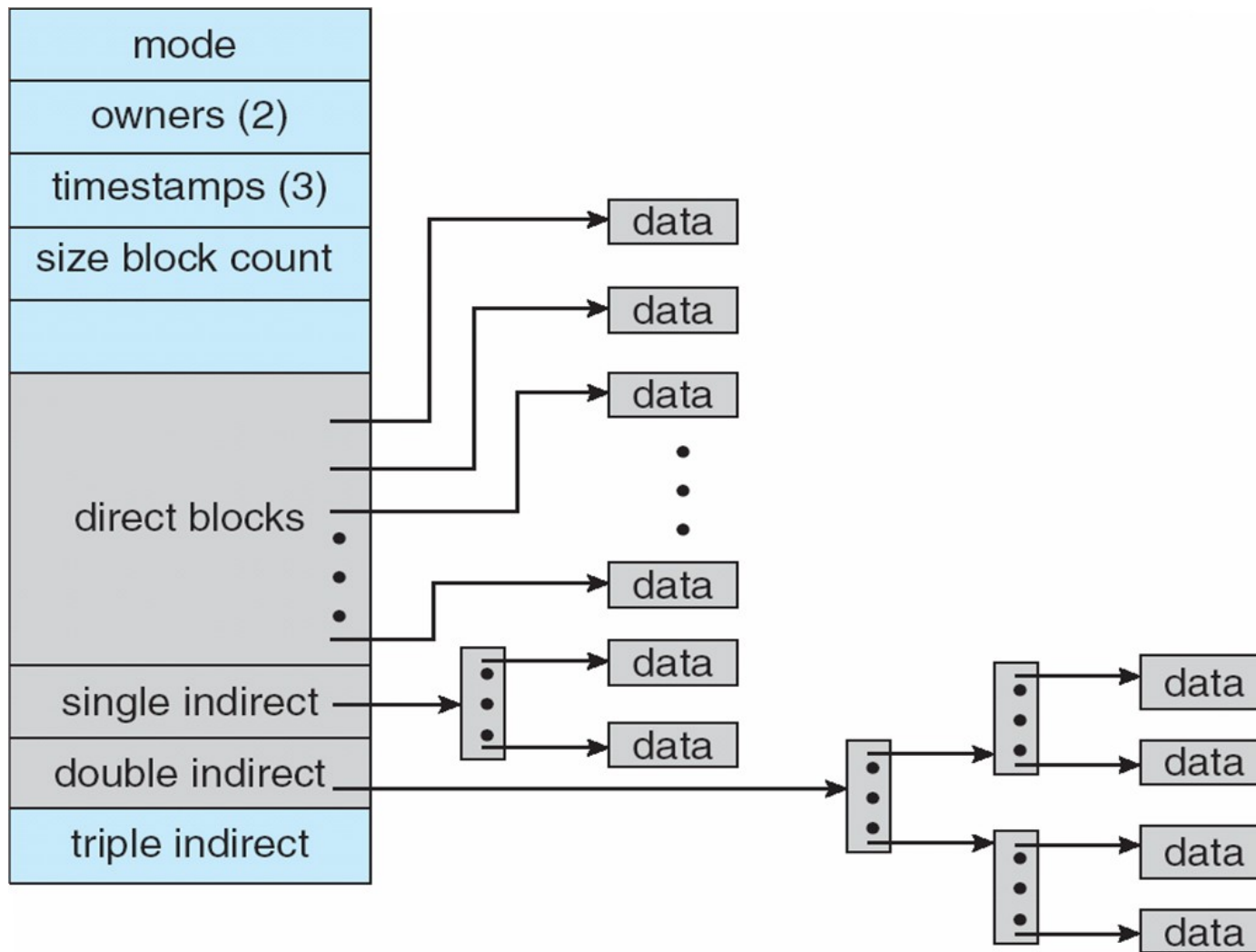


index table

Example of Indexed Allocation

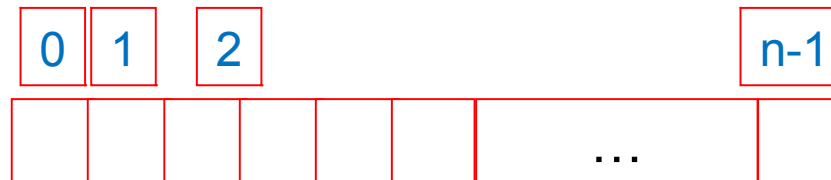



Combined Scheme: UNIX UFS (4K bytes per block)



Free-Space Management

- Bit vector (n blocks)



bit[i] =  $0 \Rightarrow$ block[i] free
 $1 \Rightarrow$ block[i] occupied

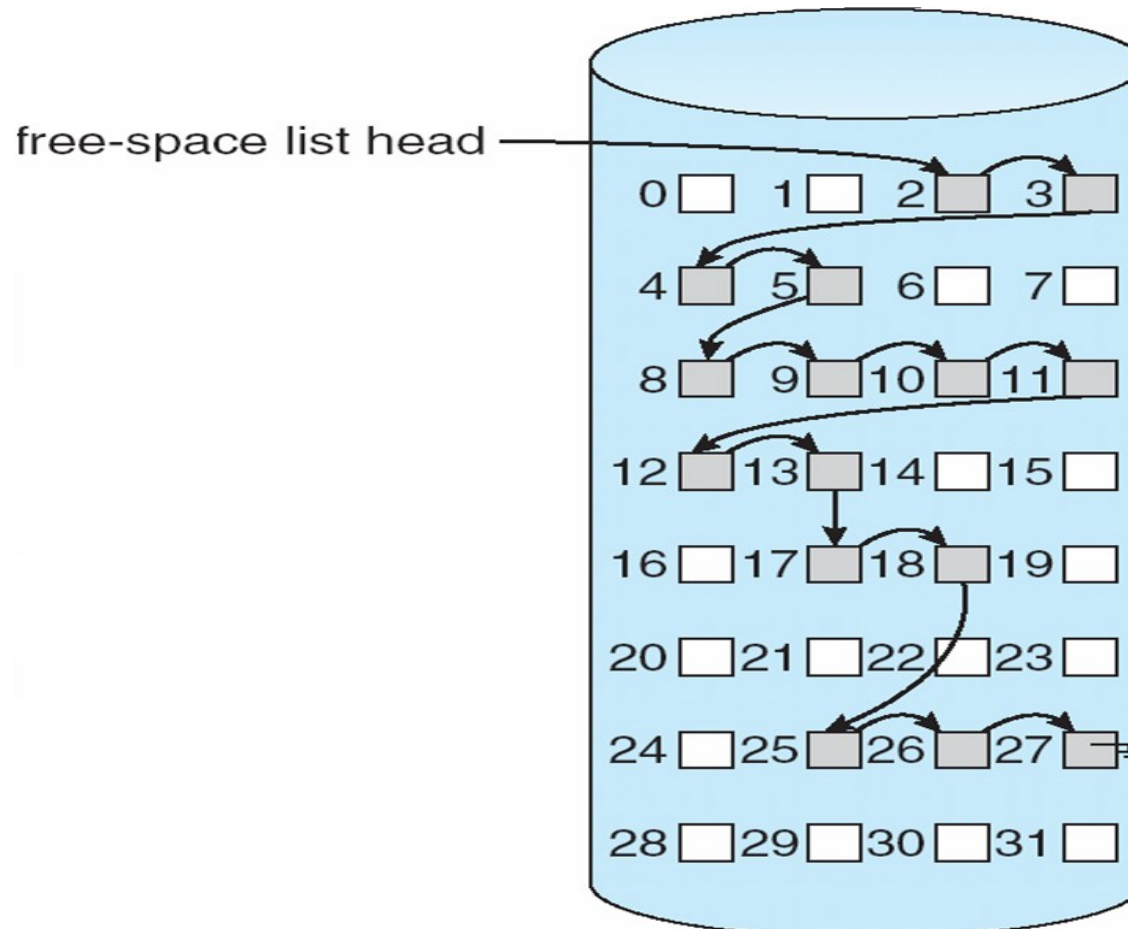
Block number calculation

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

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

Linked Free Space List on Disk



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

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.

File Sharing – Remote File Systems

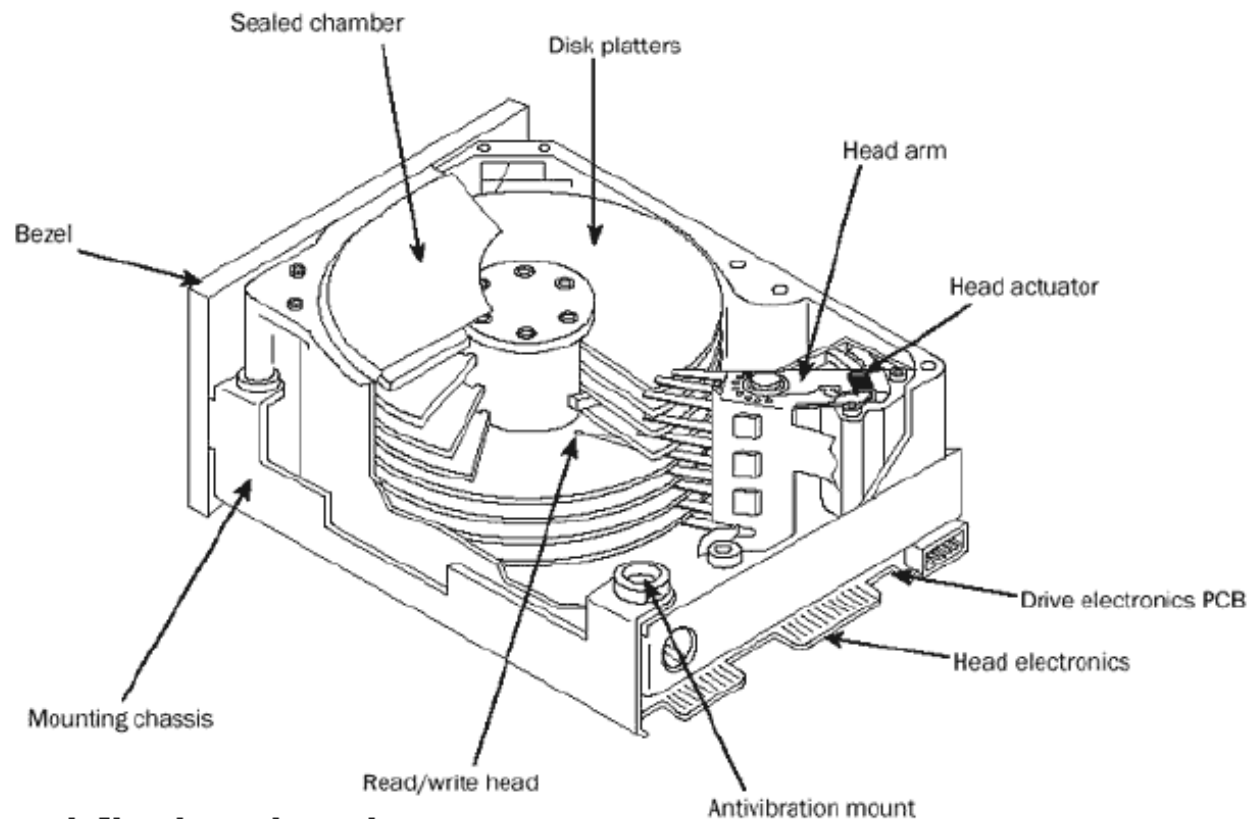
- Uses networking to allow file system access between systems
 - Manually via programs like FTP
 - Automatically, seamlessly using **distributed file systems**
 - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
 - Server can serve multiple clients
 - Client and user-on-client identification is insecure or complicated
 - **NFS** is standard UNIX client-server file sharing protocol
 - **CIFS** is standard Windows protocol
 - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as **LDAP, DNS, NIS**, Active Directory implement unified access to information needed for remote computing

Storage Subsystems

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.

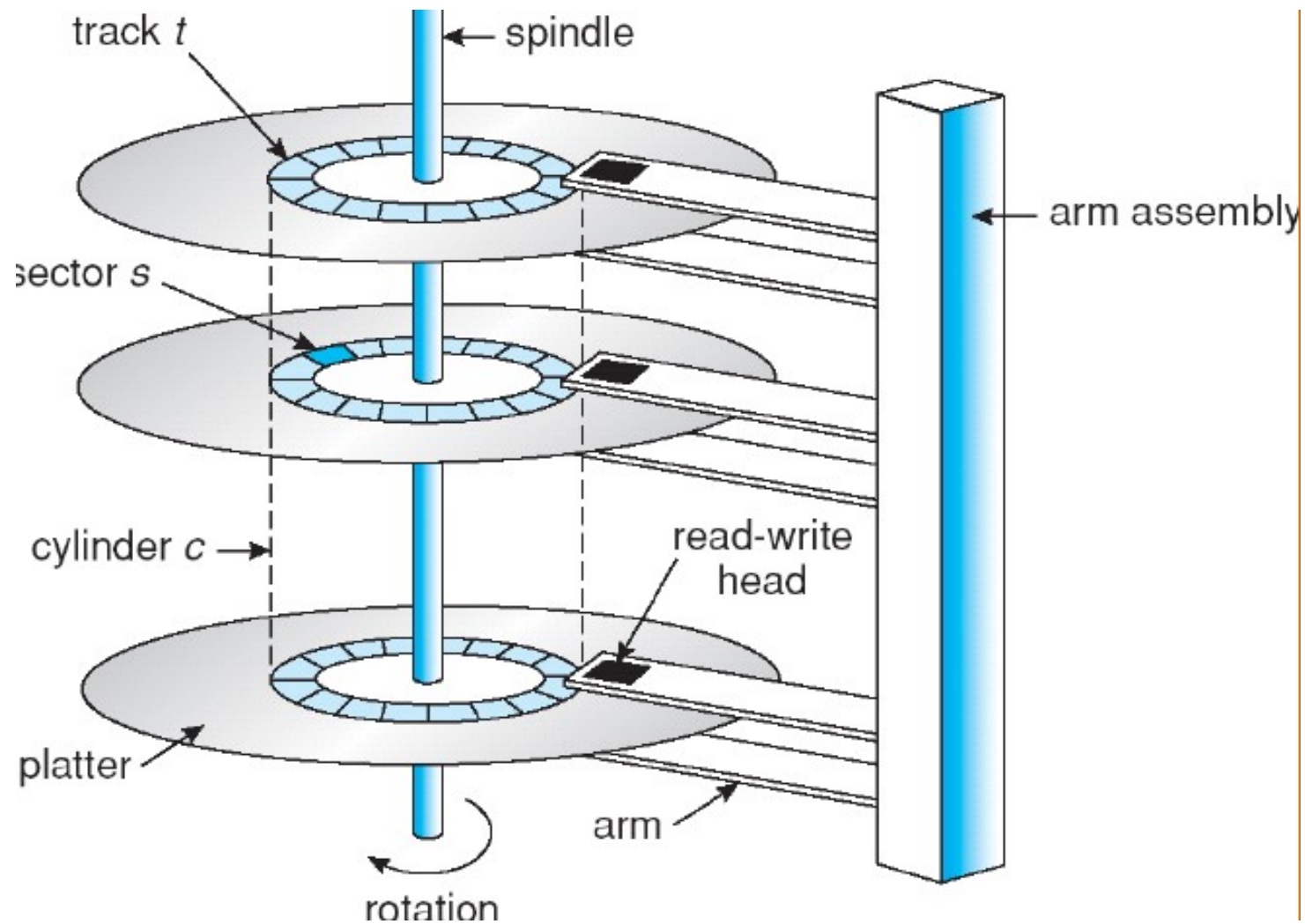
Disk Schematics



See narrated flash animation at
<http://cis.poly.edu/cs2214rvs/disk.swf>

*Source: Micro House PC
Hardware Library Volume I:
Hard Drives*

Tracks Sectors and Cylinders



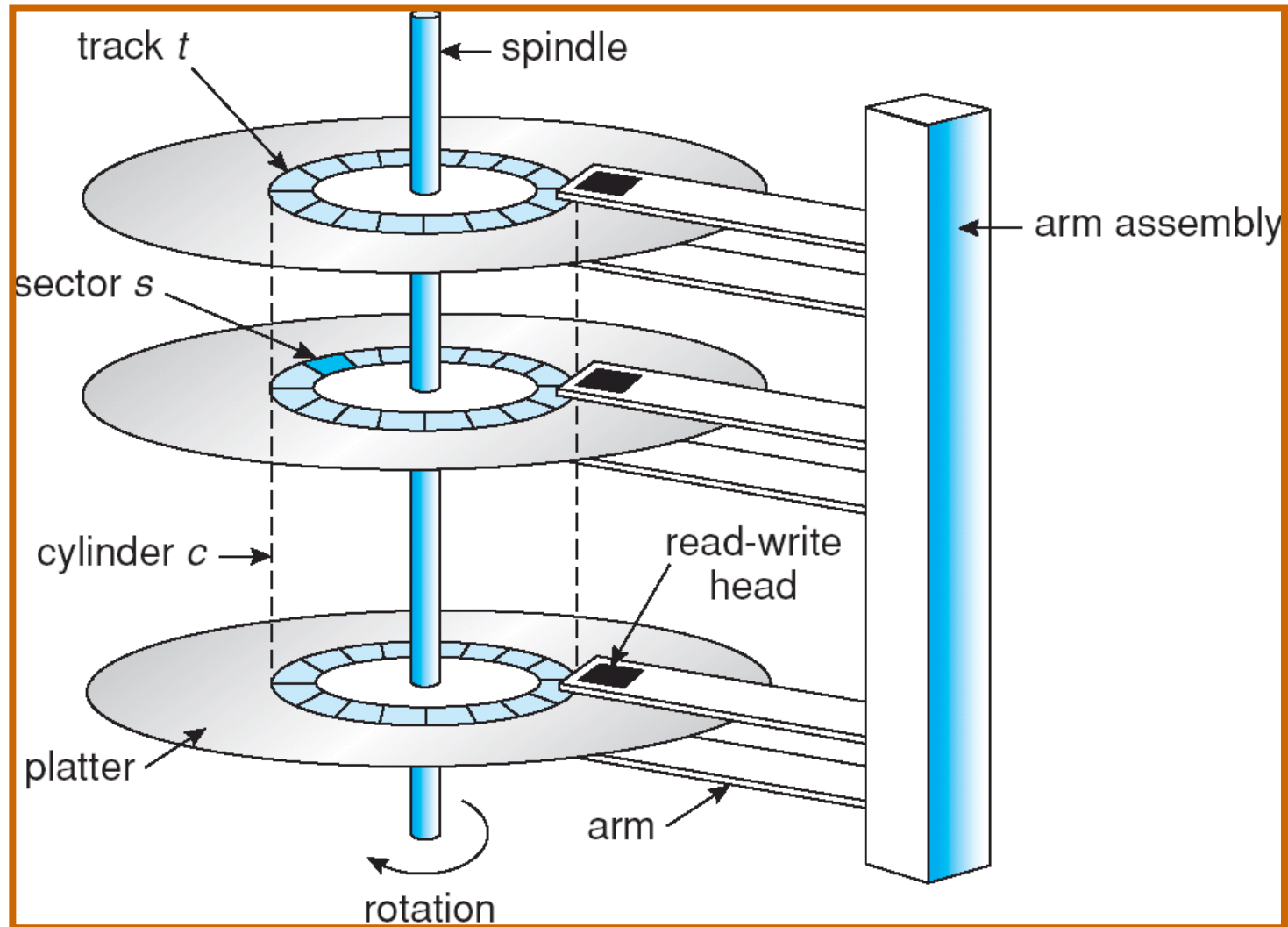
From O.S Perspective

- Disks are big & slow -compared to RAM
- Access to disk requires
 - Seek (move arm to track)
 - Rotational delay (wait for sector to appear under track)
 - Transfer time
- Seek+Rot Delay dominates
- Random Access is expensive
 - And unlikely to get better
- Consequence:
 - Avoid seeks
 - Seek to short distances
 - Amortize seeks by doing bulk transfers

Mass Storage Structure

- Magnetic disks provide bulk of secondary storage of modern computers
 - Drives rotate at 60 to 200 times per second
 - **Transfer rate** is rate at which data flow between drive and computer
 - **Positioning time (random-access time)** is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
 - **Head crash (is bad)** results from disk head making contact with the disk surface
- Disks can be removable
- Drive attached to computer via **I/O bus**
 - Buses vary, including **EIDE, ATA, SATA, USB, Fibre Channel, SCSI**
 - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Moving-head Disk Mechanism



Typical Disk Parameters

- 2-30 heads (2 per platter)
- Modern disks: no more than 4 platters
- Diameter: 2.5" –14"
- Capacity: 20MB-500GB
- Sector size: 64 bytes to 8K bytes
- Most PC disks: 512 byte sectors
- 700-20480 tracks per surface
- 16-1600 sectors per track

From OS Point Of View

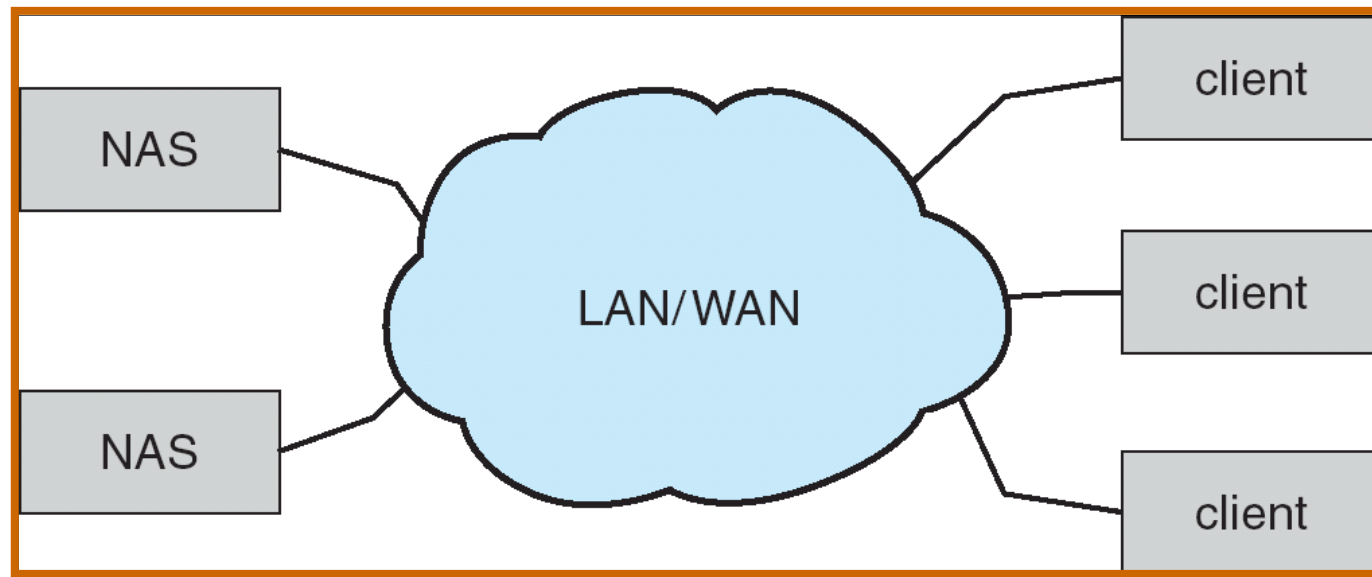
- ❑ Disks are big & slow -compared to RAM
- ❑ Access to disk requires
- ❑ Seek (move arm to track) –to cross all tracks anywhere from 20-50ms, on average takes 1/3.
- ❑ Rotational delay (wait for sector to appear under track) 7,200rpm is 8.3ms per rotation, on average takes $\frac{1}{2}$: 4.15ms rot delay
- ❑ Transfer time (fast: 512 bytes at 998 Mbit/s is about 3.91us)
- ❑ Seek + Rot Delay dominates
- ❑ Random Access is expensive
 - and unlikely to get better
- ❑ Consequence:
 - avoid seeks
 - seek to short distances
 - amortize seeks by doing bulk transfers

Disk Attachment

- Host-attached storage accessed through I/O ports talking to I/O buses
- **SCSI** itself is a bus, up to **16** devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
 - Each target can have up to **8 logical units** (disks attached to device controller)
- **FC** is high-speed serial architecture
 - Can be switched fabric with 24-bit address space – the basis of **storage area networks (SANs)** in which many hosts attach to many storage units
 - Can be **arbitrated loop (FC-AL)** of 126 devices

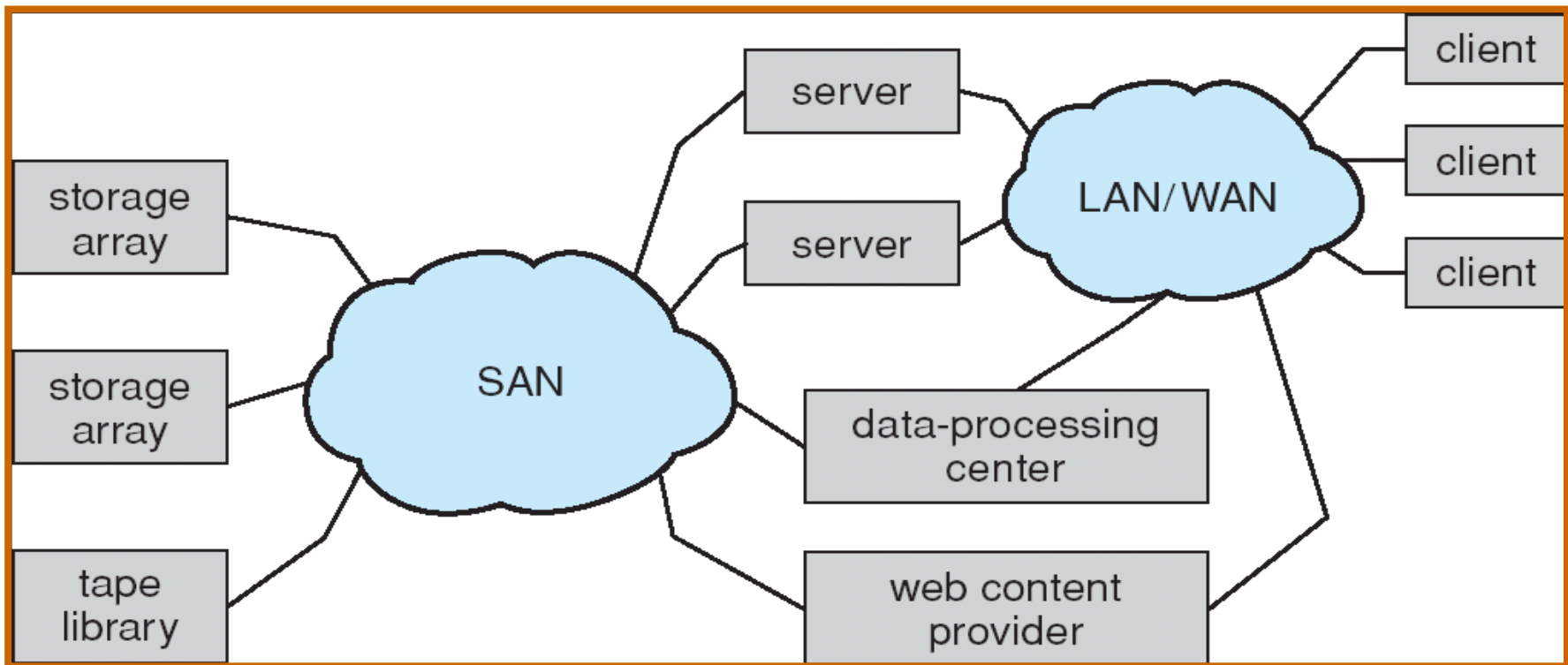
Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
- **NFS** and **CIFS** are common protocols
- Implemented via remote procedure calls (**RPCs**) between host and storage
- New **iSCSI** protocol uses IP network to carry the **SCSI** protocol



Storage Area Network

- Common in large storage environments (and becoming more common)
- Multiple hosts attached to multiple storage arrays - flexible



Disk Scheduling

- Can use priority scheme
- Can reduce average access time by sending requests to disk controller in certain order
 - Or, more commonly, have disk itself reorder requests
- **SSTF: shortest seek time first**
 - Like **SJF** in CPU scheduling, guarantees minimum average seek time, but can lead to starvation. (*How?*)
- **SCAN: “elevator algorithm”**
 - Process requests with increasing track numbers until highest reached, then decreasing etc. –repeat
- **Variations:**
 - **LOOK** –don’t go all the way to the top without passengers
 - **C-SCAN**: -only take passengers when going up

More on Disk Scheduling

- The OS is responsible for using hardware efficiently — a fast access time and disk bandwidth for disk drives
 - **Access time** has two major components
 - **Seek time**: the time for the disk arm to move the heads to the cylinder containing the desired sector
 - **Rotational latency**: the additional time waiting for the disk to rotate the desired sector to the disk head
 - **Minimize seek time**
 - **Seek time** \approx **seek distance**
 - **Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

More on Disk Scheduling (Cont.)

- Several algorithms exist to schedule the servicing of disk I/O requests
- Following is an illustration with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53(The seek head is pointing to sector 53)

FCFS

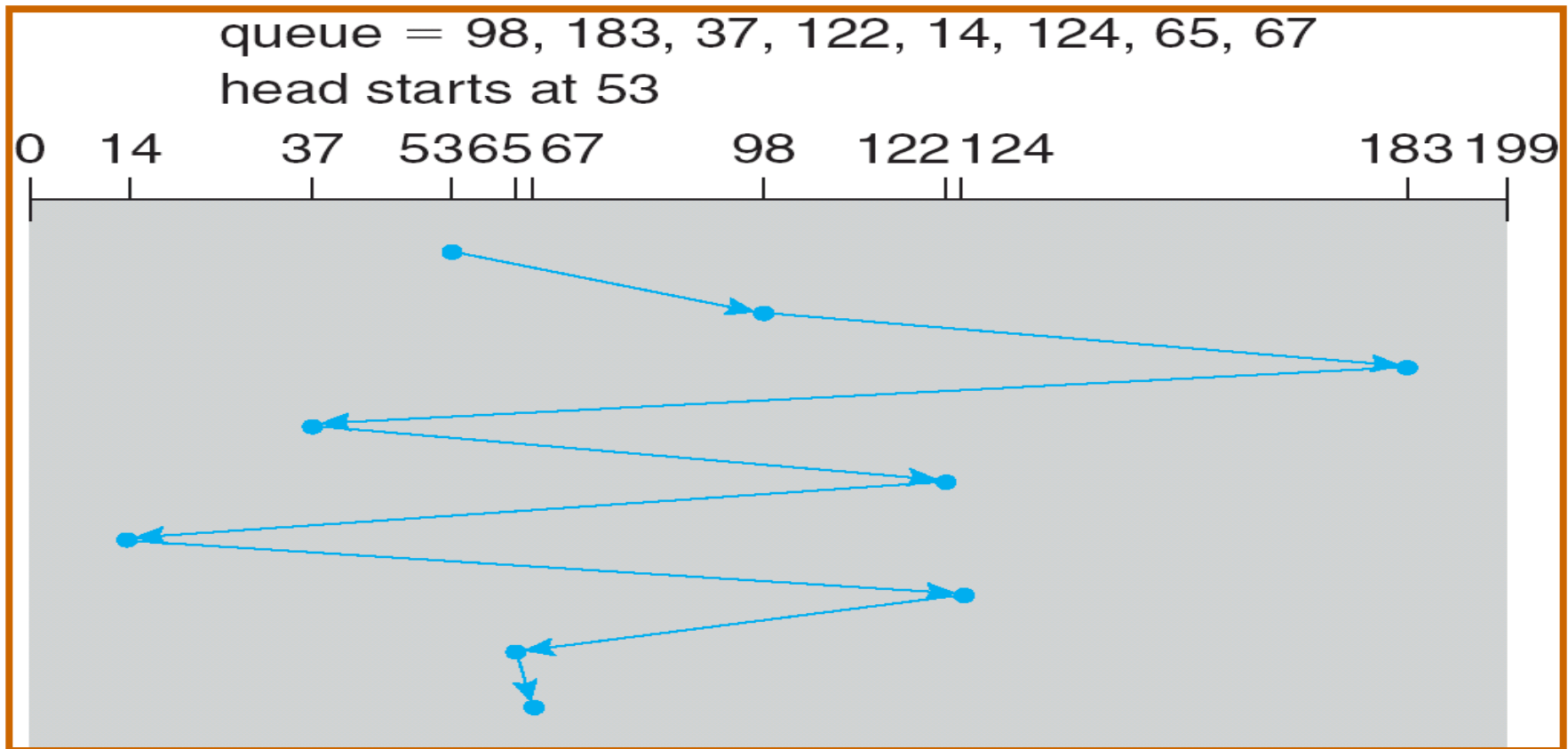


Illustration shows total head movement of 640 cylinders

SSTF

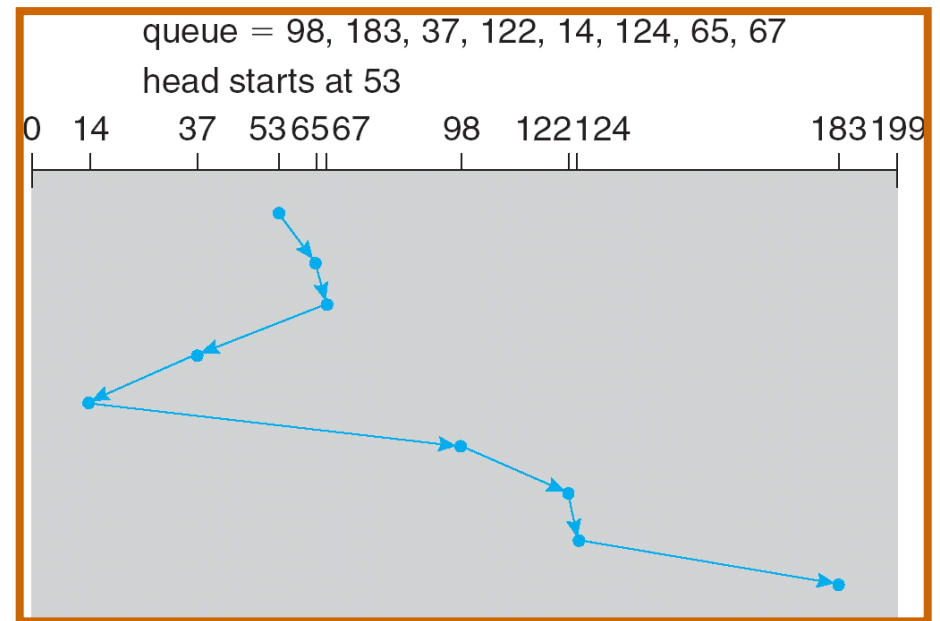
700

Selects the request with the **minimum seek time** from the current head position

SSTF scheduling is a form of SJF scheduling

- may cause starvation of some requests

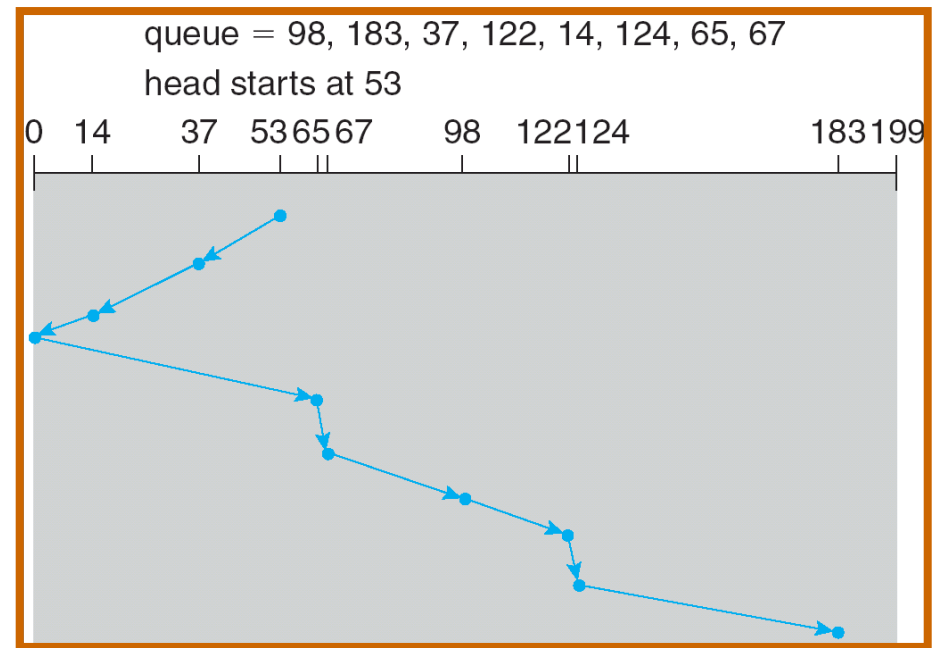
Illustration shows total head movement of 236 cylinders



SCAN

771

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues
 - Sometimes called the *elevator algorithm*
- Illustration shows total head movement of 208 cylinders



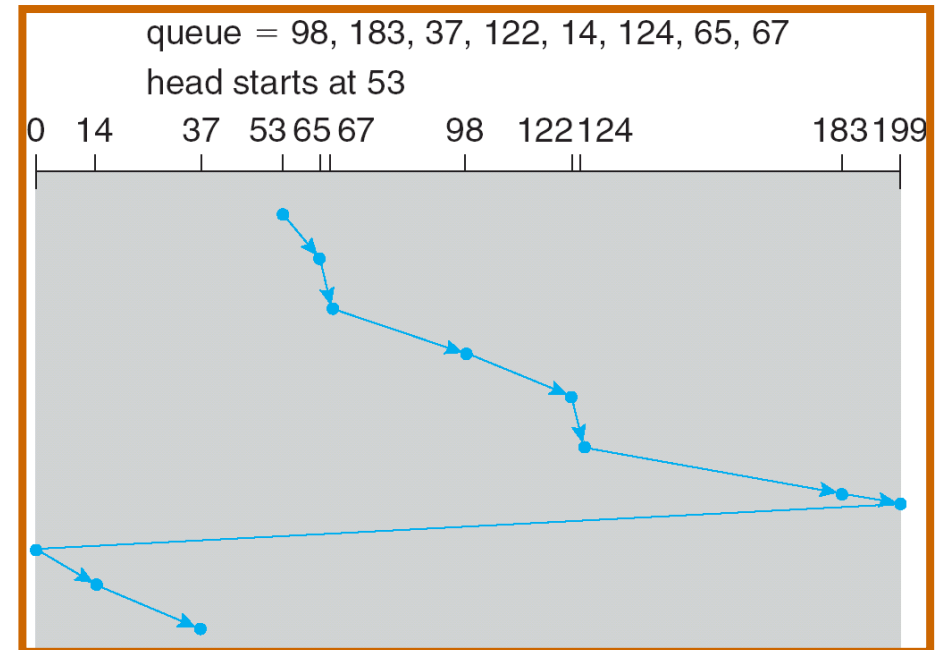
C-SCAN

722

Provides a more uniform wait time than SCAN
The head moves from one end of the disk to the other, servicing requests as it goes

- When it reaches the other end, however, it **immediately returns to the beginning of the disk**, without servicing any requests on the return trip

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

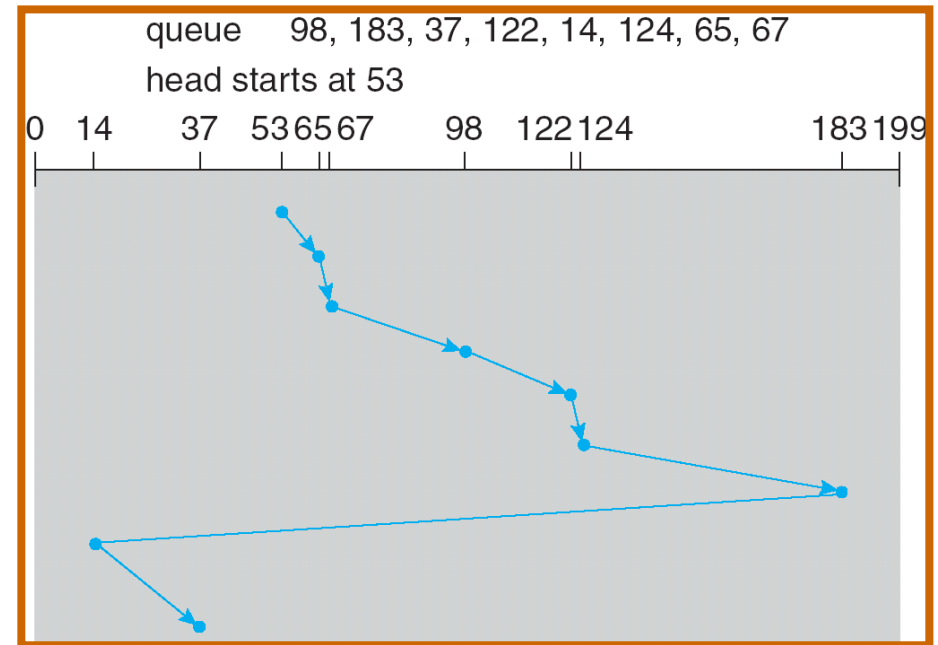


C-LOOK

733

Version of C-SCAN

Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk



How to select a Disk-Scheduling Algorithm?

- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
- The disk-scheduling algorithm should be written as a separate module of the OS, allowing it to be replaced with a different algorithm if necessary
- **SSTF** is common and has a natural appeal
- **SCAN** and **C-SCAN** perform better for systems that place a heavy load on the disk
- Either **SSTF** or **LOOK** is a reasonable choice for the default algorithm

How to Manage Disk?

- *Low-level formatting, or physical formatting* — Dividing a disk into sectors that the disk controller can read and write
- To use a disk to hold files, the OS still needs to record its own data structures on the disk
 - *Partition* the disk into one or more groups of cylinders
 - *Logical formatting* or “making a file system”
- Boot block initializes system
 - The bootstrap is stored in ROM
 - *Bootstrap loader* program
- Methods such as *sector sparing* used to handle bad blocks

Device Drivers

Introduction

- What Is A Kernel Module?
 - Modules are pieces of code that can be loaded and unloaded into the kernel upon demand
 - Difference Monolithic kernel and driver module are available
- How Do Modules Get Into The Kernel?
 - Dependency - `/lib/modules/version/modules.dep`
 - `Insmod /rmmod`
 - `Modprobe`

Modversioning

- A module compiled for one kernel won't load if you boot a different kernel
- Role of CONFIG_MODVERSIONS
- If you're having trouble loading the modules because of versioning errors, compile a kernel with modversioning turned off.

Compiling Issues and Kernel Version

- A more common problem is that some Linux versions distribute incomplete kernel headers.
- For novice it is highly recommend to download, compile and boot into a fresh Linux kernel
- Default GCC include directories, can use -I option

Hello, World (part 1): The Simplest Module

```
#include <linux/module.h>      /* Needed by all modules */
#include <linux/kernel.h>      /* Needed for KERN_INFO */
int init_module(void)
{
    printk(KERN_INFO "Hello world 1.\n");
    /* A non 0 return means init_module failed; module can't be loaded.
    */
    return 0;
}
void cleanup_module(void)
{
    printk(KERN_INFO "Goodbye Hello world 1.\n");
}
```


Introducing printk()

- Used to log information or give warnings
- There are 8 priorities and the kernel has macros for them linux/kernel.h
 - The default priority, DEFAULT_MESSAGE_LOGLEVEL
- if console_loglevel > printk
- If syslogd and klogd are running then print to /var/log/messages

Compiling Kernel Modules

Kernel modules compilation is different from regular user space applications

Makefile

```
obj-m += hello-1.o
```

all:

```
make -C <kernel source> M=$(PWD) modules
```

clean:

```
make -C <source> M=$(PWD) clean
```

Difference obj-m and obj-y

Hello World (part 2)

```
/* hello-2.c - Demonstrating the module_init() and module_exit() macros.
 * This is preferred over using init_module() and cleanup_module().
 */
#include <linux/module.h>    /* Needed by all modules */
#include <linux/kernel.h>    /* Needed for KERN_INFO */
#include <linux/init.h>     /* Needed for the macros */
static int __init hello_2_init(void)
{
    printk(KERN_INFO "Hello, world 2\n");
    return 0;
}
static void __exit hello_2_exit(void)
{
    printk(KERN_INFO "Goodbye, world 2\n");
}
module_init(hello_2_init);
module_exit(hello_2_exit);
```

*Hello World (part 3): The *init* and *exit* Macros*

Available as a feature of kernel 2.2 and later.

- The *init* macro discard init memory
- Only for the built-in modules and not for external modules
- Introducing *initdata*
- The *exit* macro causes omission of the function
- Boot time “Freeing unused kernel memory: 236k freed” message

Hello World (part 3): The `__init` and `__exit` Macros

```
/* hello-3.c - Illustrating the __init, __initdata and __exit macros.
 */
#include <linux/module.h>    /* Needed by all modules */
#include <linux/kernel.h>    /* Needed for KERN_INFO */
#include <linux/init.h>      /* Needed for the macros */
static int hello3_data __initdata = 3;
static int __init hello_3_init(void)
{
    printk(KERN_INFO "Hello, world %d\n", hello3_data);
    return 0;
}
static void __exit hello_3_exit(void)
{
    printk(KERN_INFO "Goodbye, world 3\n");
}
module_init(hello_3_init);
module_exit(hello_3_exit);
```

Hello World (part 4): Licensing and Module Documentation

proprietary modules and kernel versions 2.4 above

- “Warning: loading xxxxxx.ko will taint the kernel: no license”
- MODULE_LICENSE
- MODULE_AUTHOR
- MODULE_DESCRIPTION
- MODULE_SUPPORTED_DEVICE (Used only for the info. Having different future use)

Modules with Multiple Files

```
hello-init.c
#include <linux/kernel.h>    /* We're doing kernel work */
#include <linux/module.h>    /* Specifically, a module */
int init_module(void)
{
    printk(KERN_INFO "Hello, world - this is the kernel speaking\n");
    return 0;
}
```

```
hello-clean.c
#include <linux/kernel.h>    /* We're doing kernel work */
#include <linux/module.h>    /* Specifically, a module */
void cleanup_module()
{
    printk(KERN_INFO "Short is the life of a kernel module\n");
}
```

Makefile

```
obj-m += hello.o
```

```
hello-objs := hello-init.o hello-clean.o
```

```
all:
```

```
    make -C <kernel source location> M=$(PWD) modules
```

```
clean:
```

```
    make -C <kernel source location> M=$(PWD) clean
```

Modules vs Programs

- Entry and exit

Library functions

Difference printf and printk

printf is in glibc where printk is a system call

How printk is available to a module with out library ?

/proc/kallsyms

Even printf uses “write” system call

- Exercise : Trace small program using printf function using strace command

User Space to Kernel Space

- process and kernel system call
- normal and protected mode
- segment
- User space to kernel space communication through device files.

Device Drivers

- One class of module is the device driver, which provides functionality for hardware like Ethernet card, video card etc.
- Device file and /dev folder
- Unix and device files

Major and Minor Numbers

What is major and minor number ?

```
brw-rw---- 1 root disk 3, 1 Jul 5 2000 /dev/hda1
```

```
brw-rw---- 1 root disk 3, 2 Jul 5 2000 /dev/hda2
```

```
brw-rw---- 1 root disk 3, 3 Jul 5 2000 /dev/hda3
```

mknod command

e.g

```
$ls -l /dev/fd0 /dev/fd0u1680
```

```
brwxrwxrwx 1 root floppy 2, 0 Jul 5 2000 /dev/fd0
```

```
brw-rw---- 1 root floppy 2, 44 Jul 5 2000 /dev/fd0u1680
```

Block and character device

- The block devices have a buffer for requests
- E.g. storage devices,
- The character devices use as many bytes as they like
- Most of the devices are character devices in Unix

Character Device Drivers

- Each device is represented as files so they need file operations.
- This is available in linux/fs.h
- Registering a Device
- Unregistering a Device

File Operations

```
struct file_operations {  
    struct module *owner;  
    loff_t(*llseek) (struct file *, loff_t, int);  
    ssize_t(*read) (struct file *, char __user *, size_t, loff_t *);  
    ssize_t(*write) (struct file *, const char __user *, size_t, loff_t  
*);  
    int (*ioctl) (struct inode *, struct file *, unsigned int,  
                unsigned long);  
    int (*open) (struct inode *, struct file *);  
    int (*release) (struct inode *, struct file *);  
    ...  
};
```

File Operations

file is not a FILE

```
struct file_operations fops = {  
    .read = device_read,  
    .write = device_write,  
    .open = device_open,  
    .release = device_release  
};
```

Registering a Device

```
int register_chrdev(unsigned int major, const char *name, struct  
file_operations *fops);
```

Where

- The major number tells you which driver handles which device file
- name is the name of the device file in /dev/
- file_operation is the operations associated with the device file.
- If Major number is zero then kernel assigns a dynamic number.
- problems with the dynamic major number and the mknode

3 way to solve this

- Using printks and create device file
- Statically assigned Major number – should assign carefully
- Using mknode system call

Protecting module while in use

What is usage count?

- Why do we need this ?

In 2.6 kernels

- `try_module_get(THIS_MODULE)`: Increment the use count
- `try_module_put(THIS_MODULE)`: Decrement the use count.

In 2.4 kernels

- `MOD_INC_USE_COUNT` Increment the use count
- `MOD_DEC_USE_COUNT` Decrement the use count

The /proc File System

- The *proc* file system was originally designed to allow easy access to information about processes. It is now used by every bit of the kernel which has something interesting to report
- e.g /proc/modules
- The method to use the proc file system is very similar to the one used with device files
- struct proc_dir_entry
- create_proc_entry

Talking to Device Files

IOCTLs (short for Input Output ConTroL)

Difference Read/write with IOCTL

The ioctl function is called with three parameters:

- The file descriptor of the appropriate device file
- The ioctl number
- Parameter

where the ioctl number encodes the major device number, the type of the ioctl, the command, and the type of the parameter

Created by a macro call (`_IO`, `_IOR`, `_IOW` or `_IOWR`)

Eg `_IOR(10,0,int)`

Source: Operating Systems Principles
by
A. Silberschatz, and P. Galvin

And

Internet Sources