Working Sets and Page Fault Rates

- Direct relationship between working set of a process and its page-fault rate
- Working set changes over time
- Peaks and valleys over time

Other Considerations

- Prepaging
- Page size
- TLB reach
- Inverted page tables
- Program structure
- I/O interlock and page locking

Other Considerations -- Prepaging

- Prepaging used to reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory was wasted
- Assume that:
  - *s* pages are prepaged and
  - A fraction α of these *s* pages is actually used (α between 0 and 1)
  - Question -- is the cost of $s * \alpha$ saved pages faults greater or less than the cost of prepaging $s * (1- \alpha)$ unnecessary pages?
    - If α near zero $\Rightarrow$ prepaging loses
    - If α near one $\Rightarrow$ prepaging wins


Other Issues – Page Size

- Sometimes OS designers have a choice
  - Especially if running on custom-built CPU
- Page size selection must take into consideration:
  - Fragmentation
  - Page table size
  - Resolution
  - I/O overhead
  - Number of page faults
  - Locality
  - TLB size and effectiveness
- Always power of 2, usually in the range 2^{12} (4,096 bytes) to 2^{22} (4,194,304 bytes)
- On average, growing over time


Other Issues – TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  - Otherwise there is a high degree of page faults
- Increase the Page Size
  - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation


Other Issues – Program Structure

- Program structure
  - `int[128,128] data;`
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

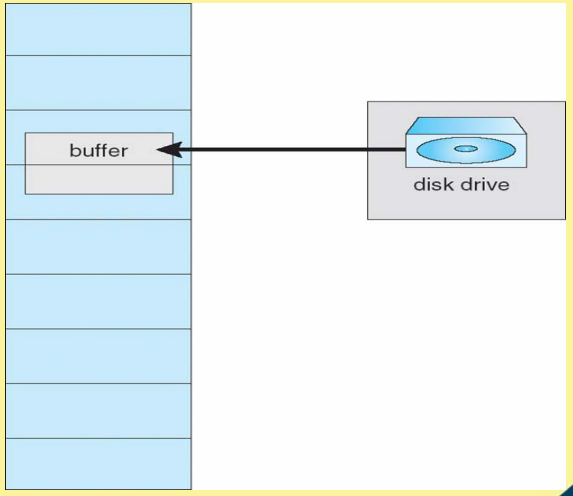
  - Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 page faults


Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- **Pinning** of pages to lock into memory



Conclusion

Conclusion:

- Virtual memory makes memory available to a process very large
- Page replacement is an issue
- Program locality helps in the success of virtual memory


NPTEL ONLINE CERTIFICATION COURSES

Thank you

Operating System Fundamentals

Santanu Chattopadhyay
Electronics and Electrical Communication Engg.

File Systems and Mass Storage

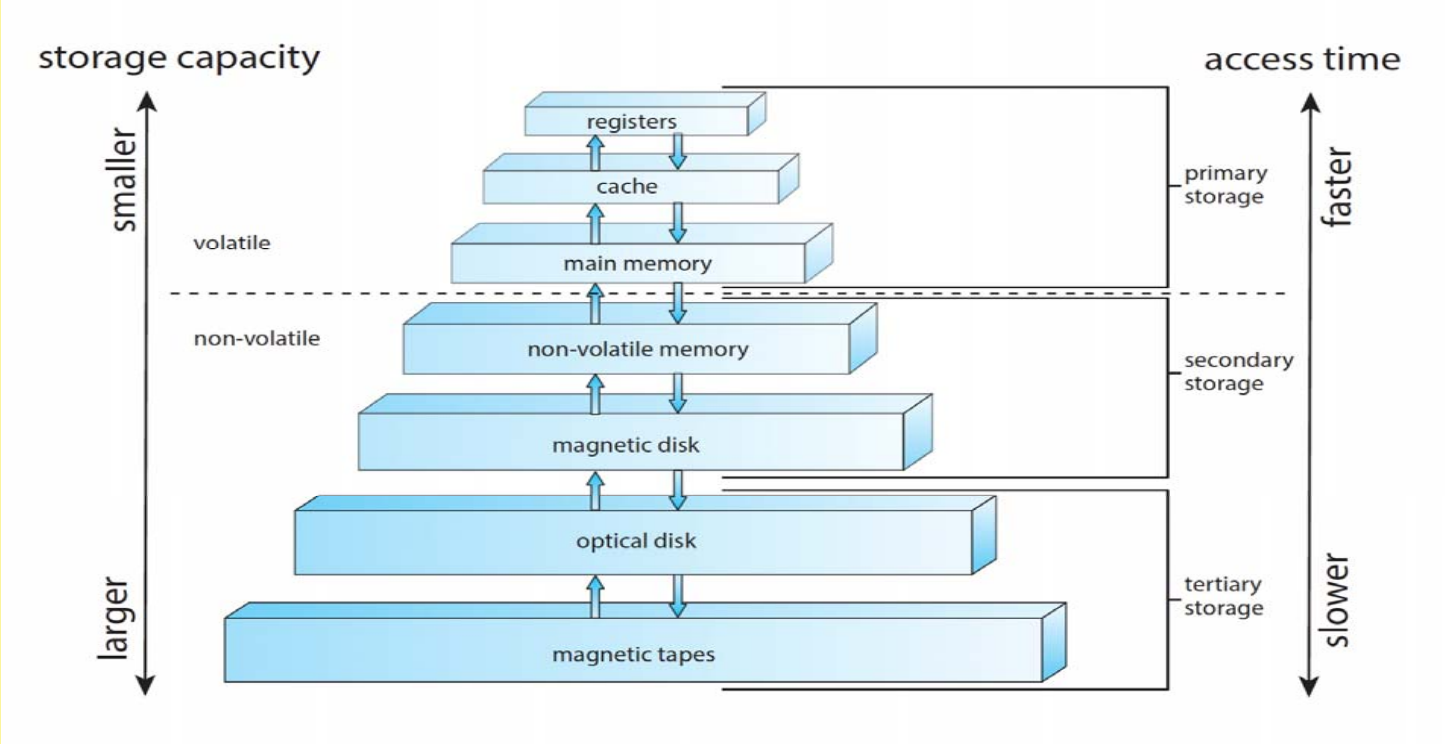
Concepts Covered:

- Secondary storage system
- Storage hierarchy
- Disk structure
- Disk scheduling
- File organization
- File access and protection

Mass-Storage Systems

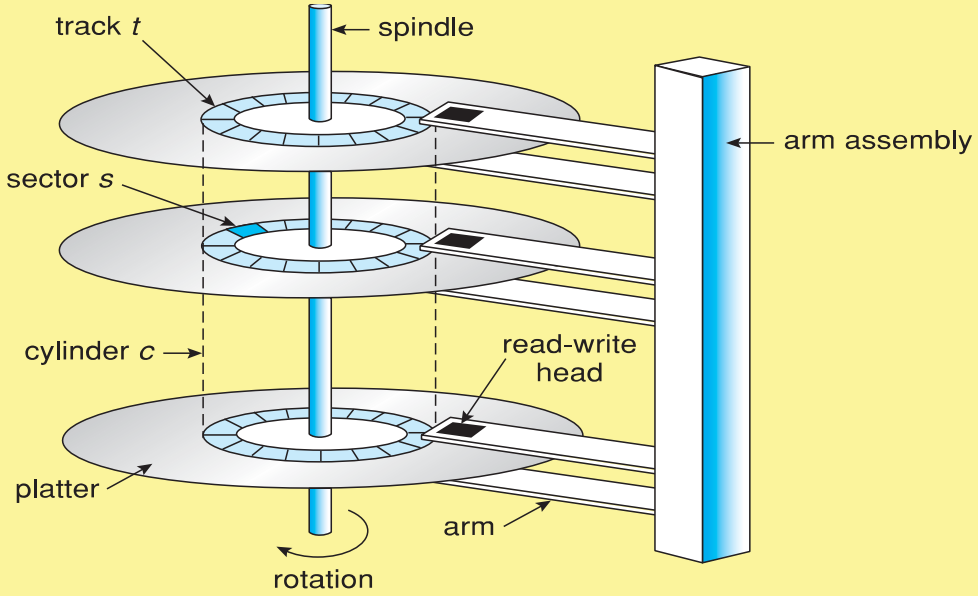
- Overview of Mass Storage Structure
- Storage Attachment
- Secondary Storage I/O Scheduling
- Storage Device Management
- Swap-Space Management
- RAID Structure


Storage-Device Hierarchy



Hard Disk Drive Moving-head Mechanism

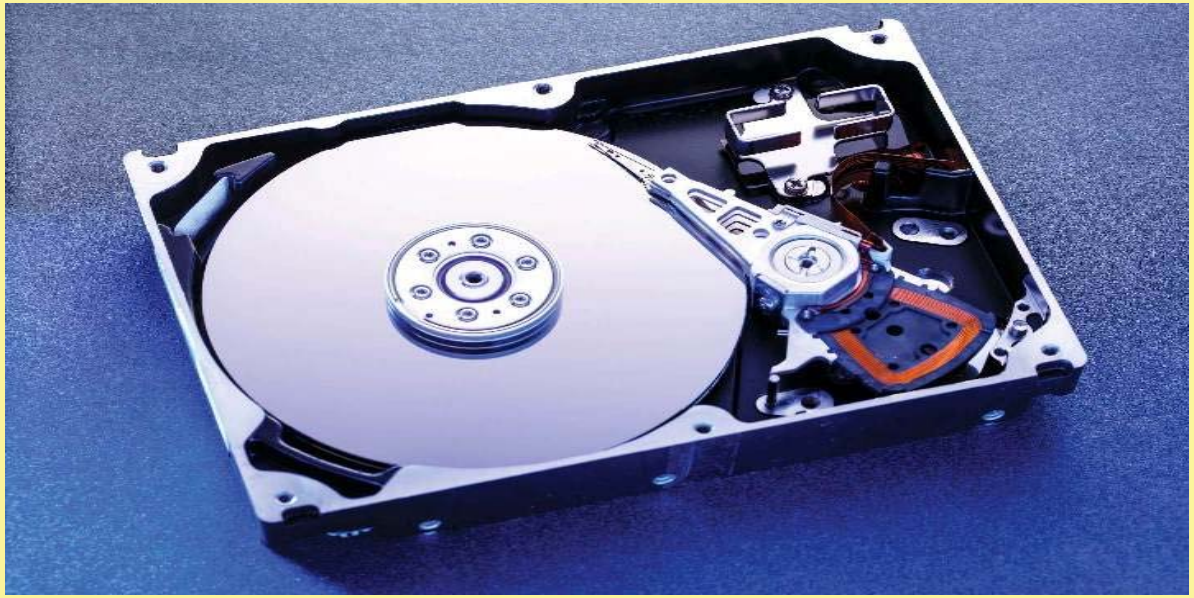
- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8”
- Range from gigabytes through terabytes per drive



Hard Disk Drives

- HDDs rotate at 60 to 250 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** results from disk head making contact with the disk surface
  - That's bad
- Disks can be removable
- Other types of storage media include CDs, DVDs, Blu-ray discs. magnetic tape


A 3.5" HDD with Cover Removed.



The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5 million (7 bit) characters
50 x 24” platters
Access time = < 1 second

Performance of Magnetic Disks (Cont.)

- **Access Latency** = **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead


Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
- Low-level formatting creates **logical blocks** on physical media
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - Logical to physical address should be easy
    - Except for bad sectors
    - Non-constant # of sectors per track via constant angular velocity
  - Each sector 512B or 4KB – smallest I/O the drive can do


Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ 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


Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queued.
  - Optimization algorithms only make sense when a queue exists


Disk Scheduling (Cont.)

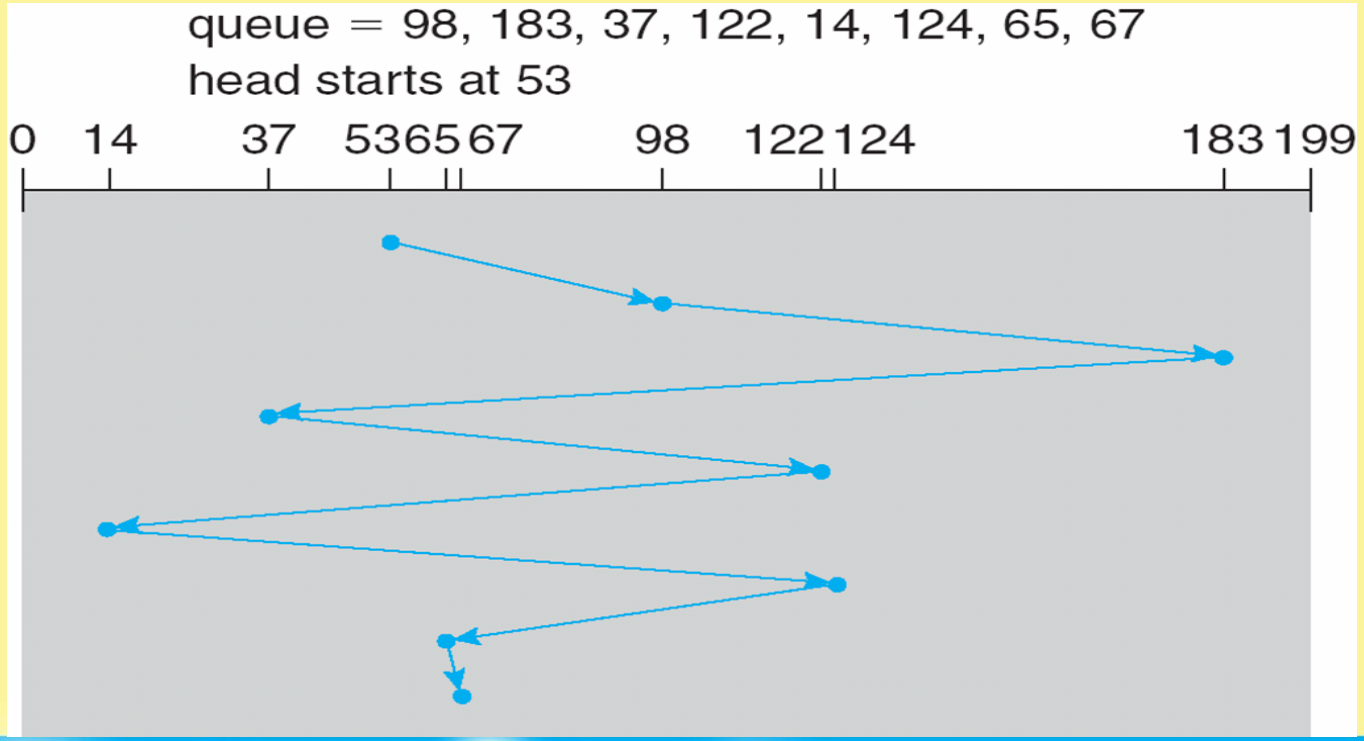
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

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

Head pointer 53

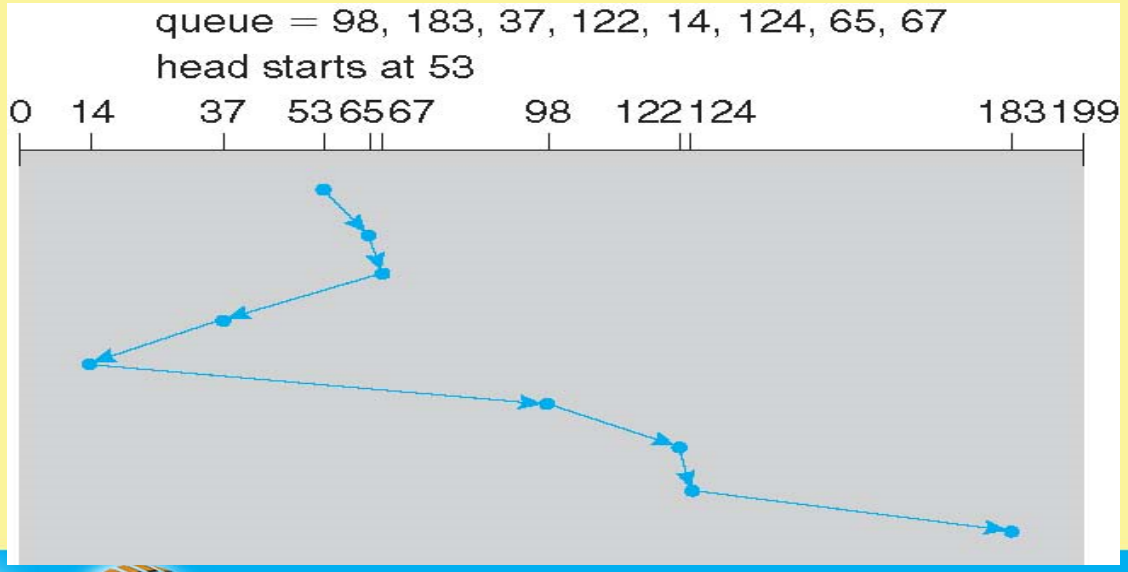

FCFS

Illustration shows total head movement of 640 cylinders



SSTF

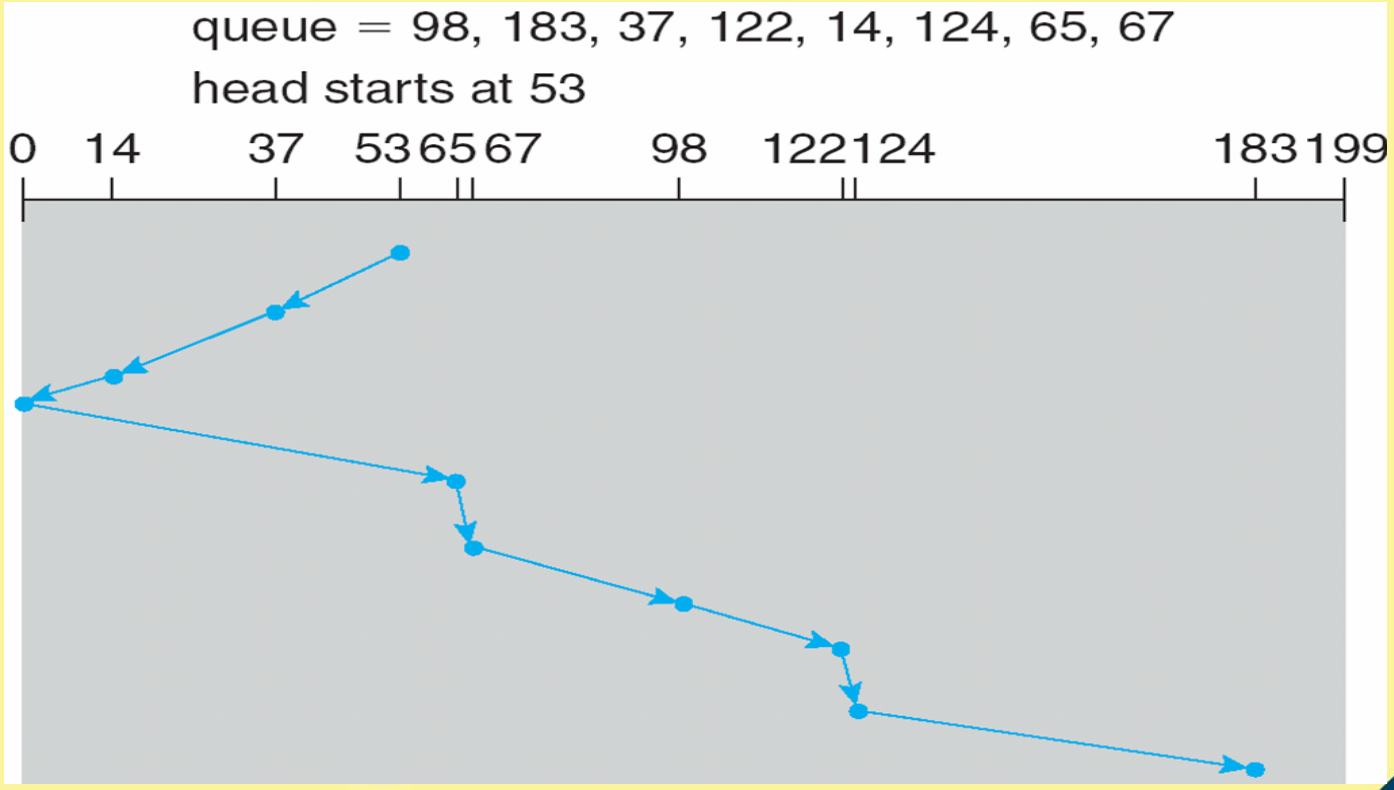
- Shortest Seek Time First -- selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders



SCAN

- **SCAN algorithm**. 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 it is called the **elevator algorithm**
- Illustration shows total head movement of 208 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

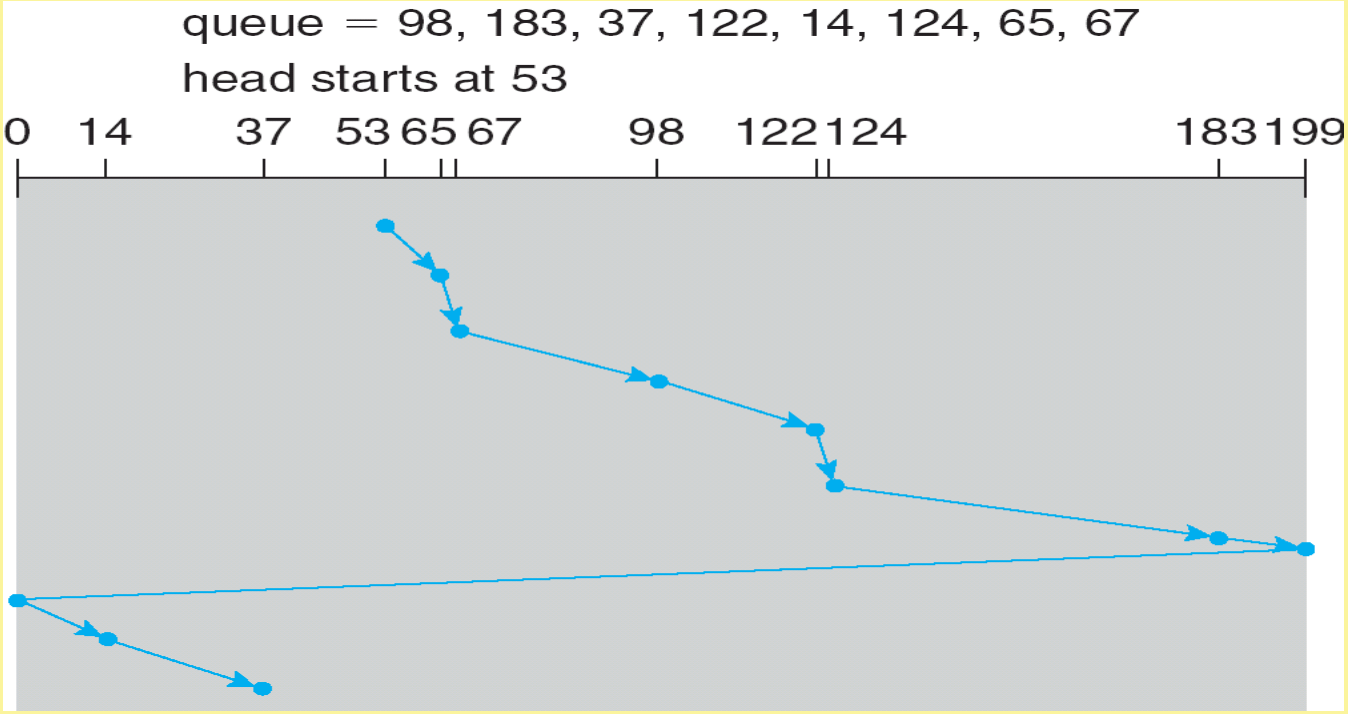

SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0 14 37 53 65 67 98 122 124 183 199

C-SCAN

- 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

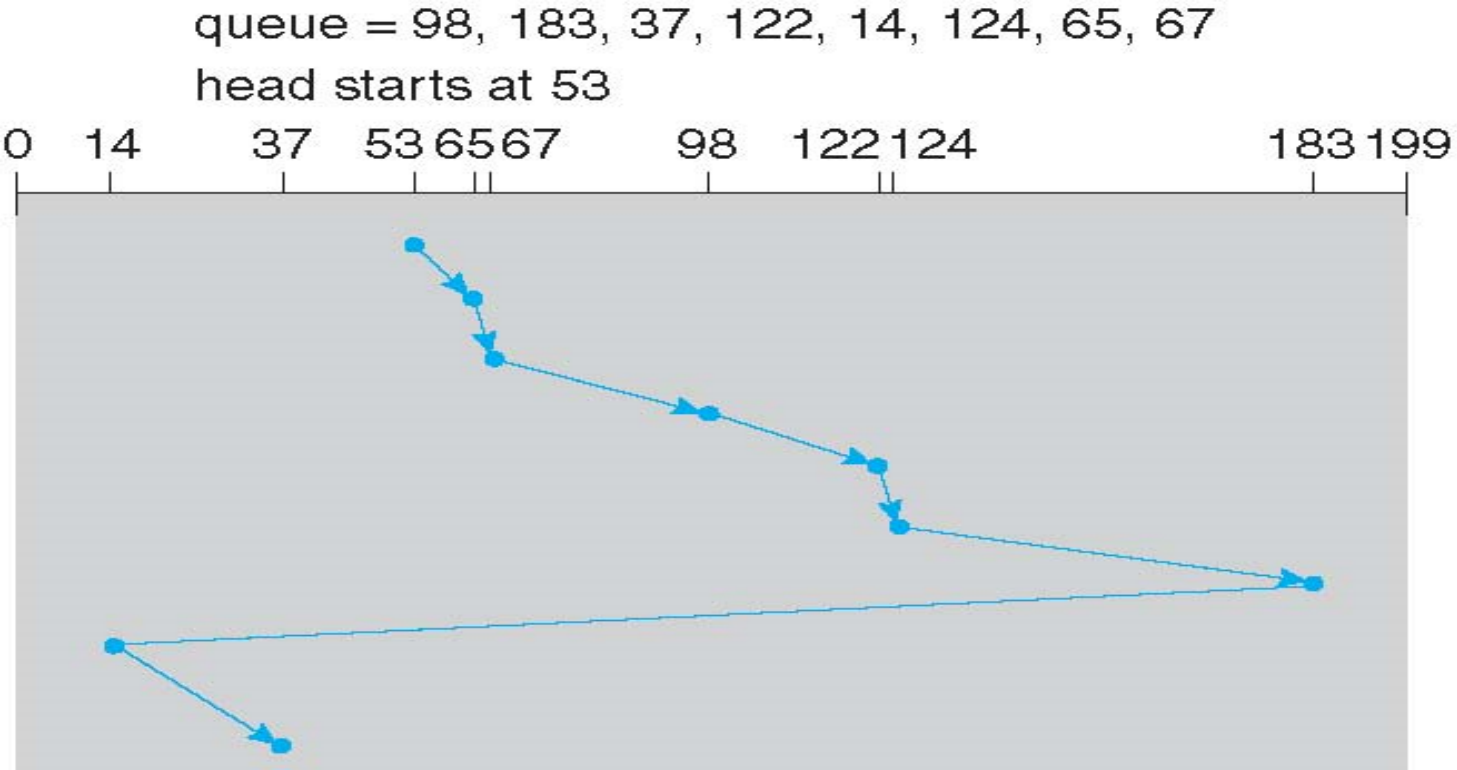

C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

LOOK and C-LOOK

- LOOK is a version of 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
- C-LOOK a version of C-SCAN. Arm only goes as far as the last request in one direction, then reverses direction immediately, without first going all the way to the end of the disk


C-LOOK (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout


Disk-Scheduling Algorithm (Cont.)

- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queuing effect OS queue ordering efforts?


Reliability and Redundancy

- **Mean time to failure**. The average time it takes a disk to fail.
- **Mean time to repair.** The time it takes (on average) to replace a failed disk and restore the data on it.
- **Mirroring.** Copy of a disk is duplicated on another disk.
  - Consider disk with 100,000 mean time to failure and 10 hour mean time to repair
    - Mean time to data loss is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years If mirrored disks fail independently,
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively


RAID Structure

- RAID – redundant array of inexpensive disks
- Use of many disks Increases the **mean time to failure.**
  - 100 disks with MTF of 100,000 hours.
  - 100,000/100 = 1,000 hours or 41.66 days.
- Solution is to have data redundancy over the 100 disks.
- **Disk striping.** Splitting the bits (or blocks) across multiple disks
  - **bit-level striping**. The bits of a byte are split across multiple disks.
  - **block-level striping**. The blocks of a file are split across multiple disks.
- For example, if we have an array of eight disks, we write bit "*i*" of each byte to disk "*i*". The array of eight disks can be treated as a single disk with sectors that are eight times the normal size and, more important, that have eight times the access rate.


RAID Structure (Cont.)

- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
- RAID within a storage array can still fail if the array fails:
  - Replication of the data between arrays is common -- automatic duplication of writes between separate sites
    - For redundancy and disaster recovery
    - Can be synchronous or asynchronous
  - Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them
  - Hot spare disk is not used for storing data. It is configured to be used as a replacement in case of a disk failure.


RAID Levels

(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.