Input/Output

Chapter 5: Sections 5.3.3-5.4.3

Device Independent I/O

» While some I/O is device specific, other parts are device independent.

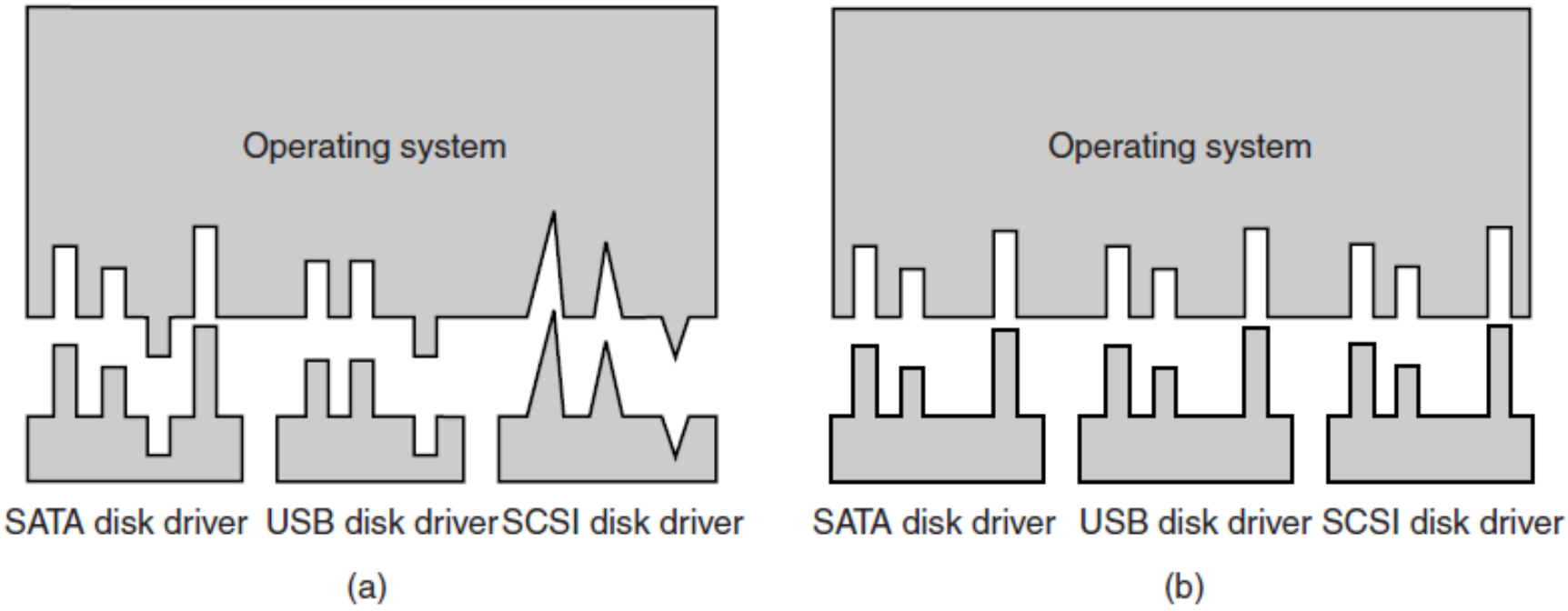
  » Lines varies

    » Efficiencies

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Uniform Interfacing

» How do we make all the drivers and I/O devices look the same?



(a) Without a standard driver interface.
(b) With a standard driver interface.

Uniform Interfacing

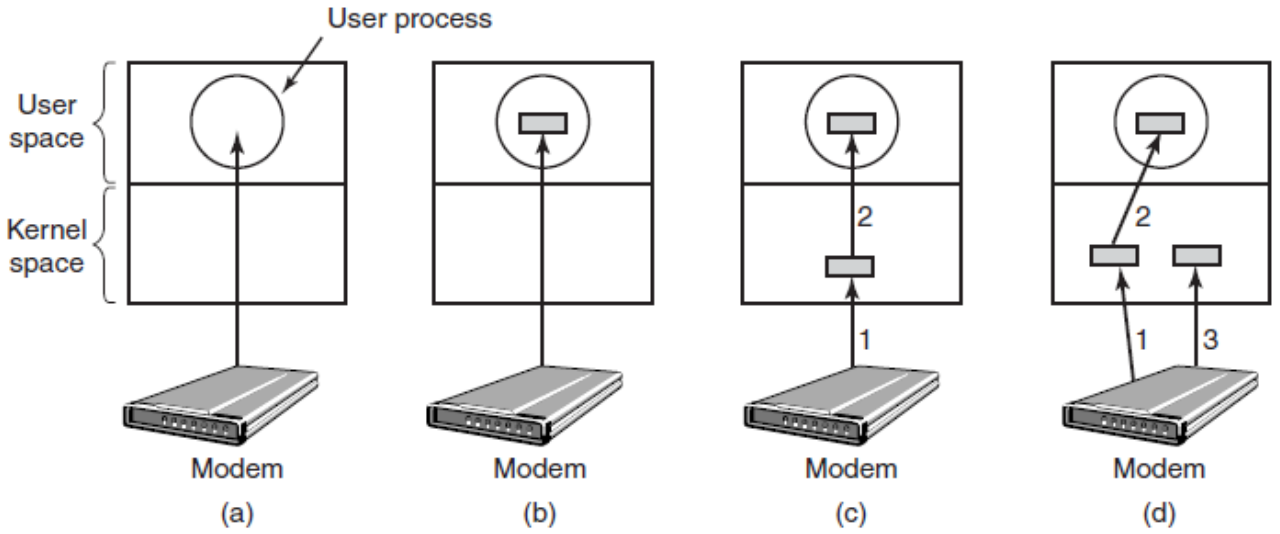
- Device driver writers know what is expected in their drivers
- While all I/O devices are not identical usually there are a small number of device types
  - Each class defines a set of operations a driver must supply

Uniform Interfacing

- Uniform naming.
  - /dev/sda
  - /dev/hda
- Uniform protections and permissions

Buffering

» Both block and character devices must deal with buffering.



(a) Unbuffered input. (b) Buffering in user space. (c) Buffering in the kernel followed by copying to user space. (d) Double buffering in the kernel.

Buffering

- (a) Unbuffered input.
  - read, block, interrupt, repeat
  - inefficient
- (b) Buffering in user space.
  - Process provides n-byte buffer
  - When buffer full the process is awakened
    - What if buffer is paged out?

Buffering

- (c) Buffering in the kernel followed by copying to user space.
  - Copy in one operation
- (d) Double buffering in the kernel.
  - Copy one while accumulating in another
- Circular buffer

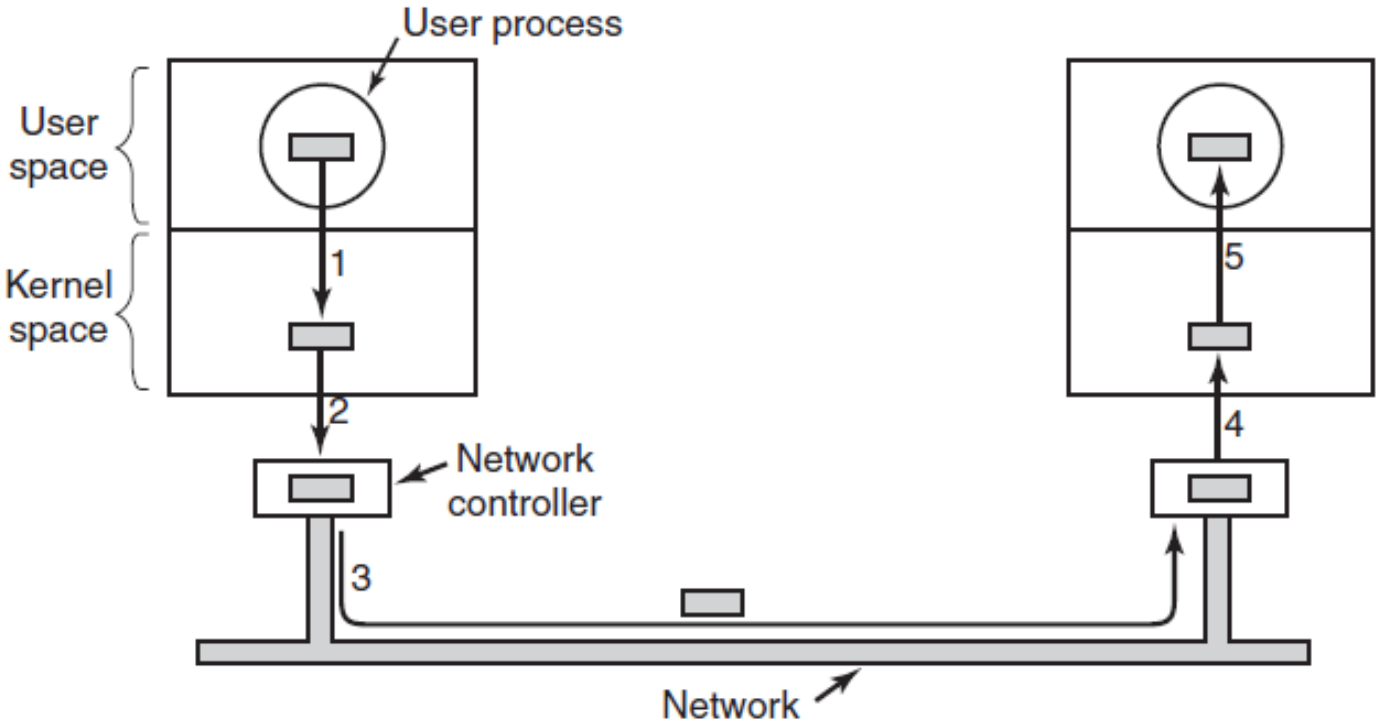
Buffering

- Also important on output
- On write:
  - Block the user until all characters written.
    - Slow
  - Release the user and process I/O while user process continues
    - How do you know when it's finished?

Buffering

- System could generate signal or interrupts
  - Prone to race conditions
- Better to copy to kernel buffer and in (c)
- Too many buffers on the path cause performance problems.

Buffering


User process
User space
Kernel space
1
2
3
4
5
Network controller
Network

Error Reporting

- Errors are far more common in I/O context than in any other.
  - Device specific and handled by driver
  - Framework for error handling is device independent

Classes of Errors

1. Programmer error
   - Writing to an input
   - Just report error
2. I/O error
   - Writing to bad disk block
   - Driver needs to determine what to do
   - May pass error up to device independent software

Dedicated Devices

- Some device, such as printers, can only be used by a single process at a time
  - Operating system adjudicates requests
    - Require processes to call `open()`
  - Special mechanism for requesting and releasing devices
    - Processes block and wait in queue

Device Independent Block Size

» Different disks may have different block sizes.

  – Device independent software hides this and provides a uniform block size to higher layers.

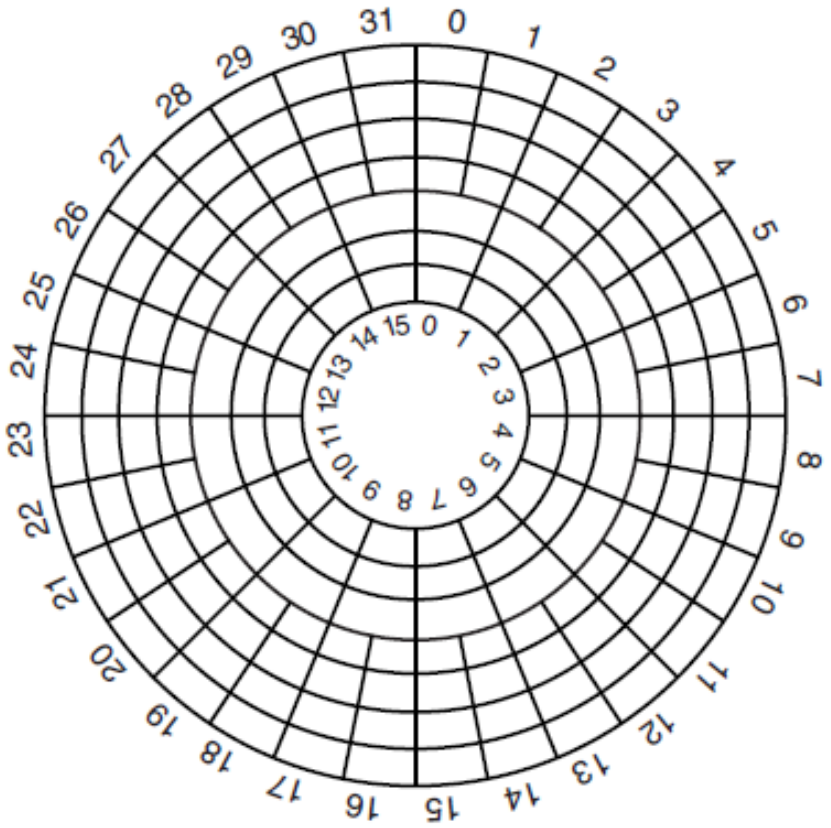
    • Treat several sectors as a singe logical block.

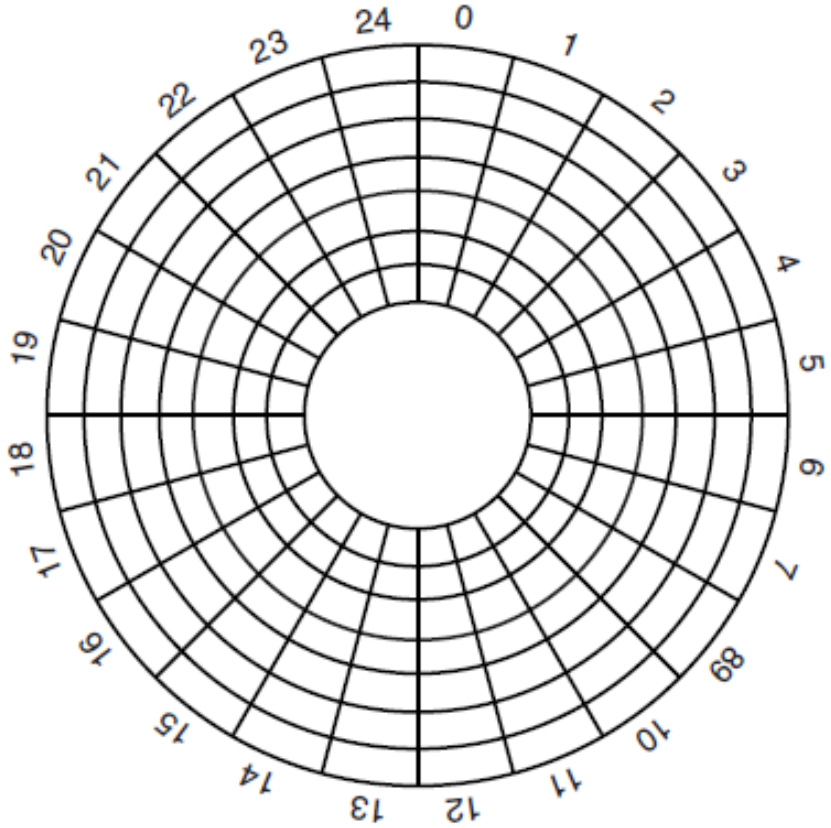
Disks

- Magnetic disks are organized into cylinders, each having as many tracks as there are heads stack vertically.
  - Tracks are divided into sectors
- Early dusks delivered serial bit stream
- IDE and SATA have a microcontroller that does considerable work and allows the real controller to issue higher commands.
  - Overlapped seeks

Disks

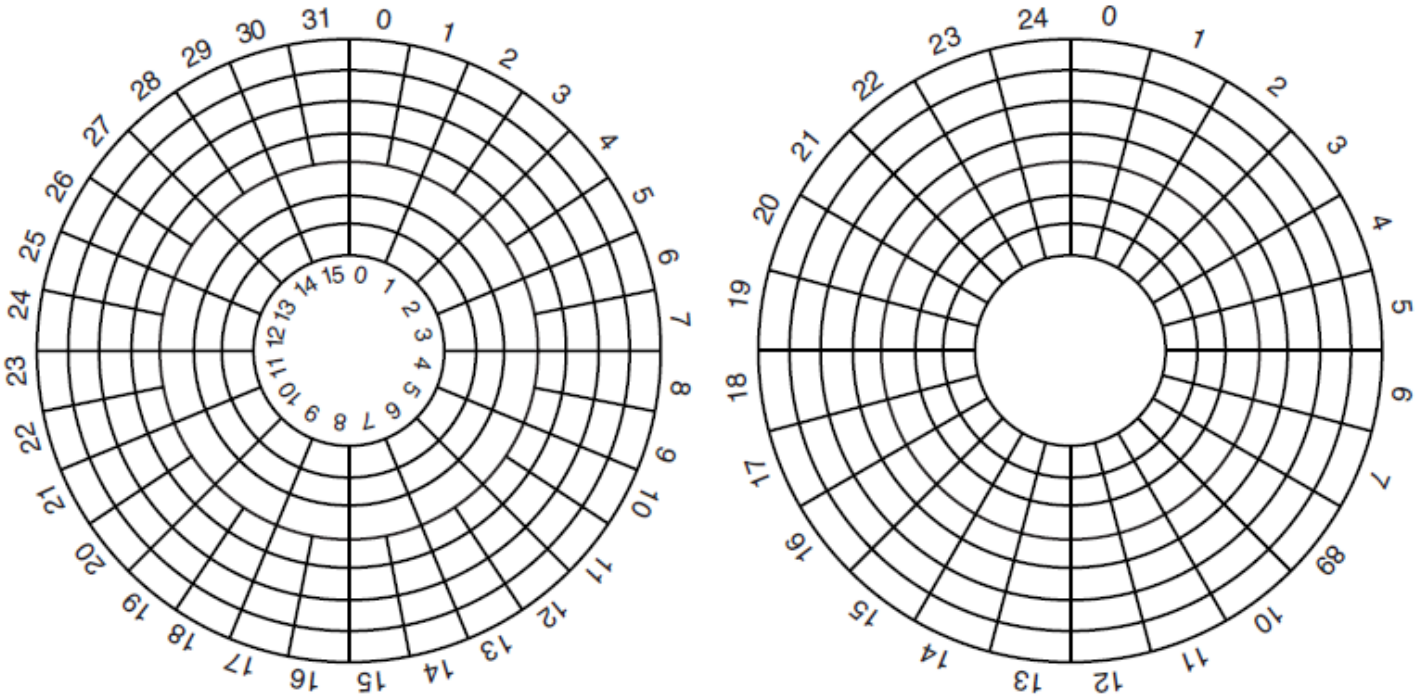
» Geometry specified and used by the driver software is almost always different form the physical format.





Disks

» Logical Block Addressing - disk sectors are numbered consecutively starting at 0 with no regard for the disk geometry,



RAID Structure

- Reliability measured in mean time between failure
- RAID - redundant array of inexpensive disks
- RAID – multiple disk drives provides reliability via redundancy
- Frequently combined with NVRAM to improve write performance
- RAID is arranged into six different levels

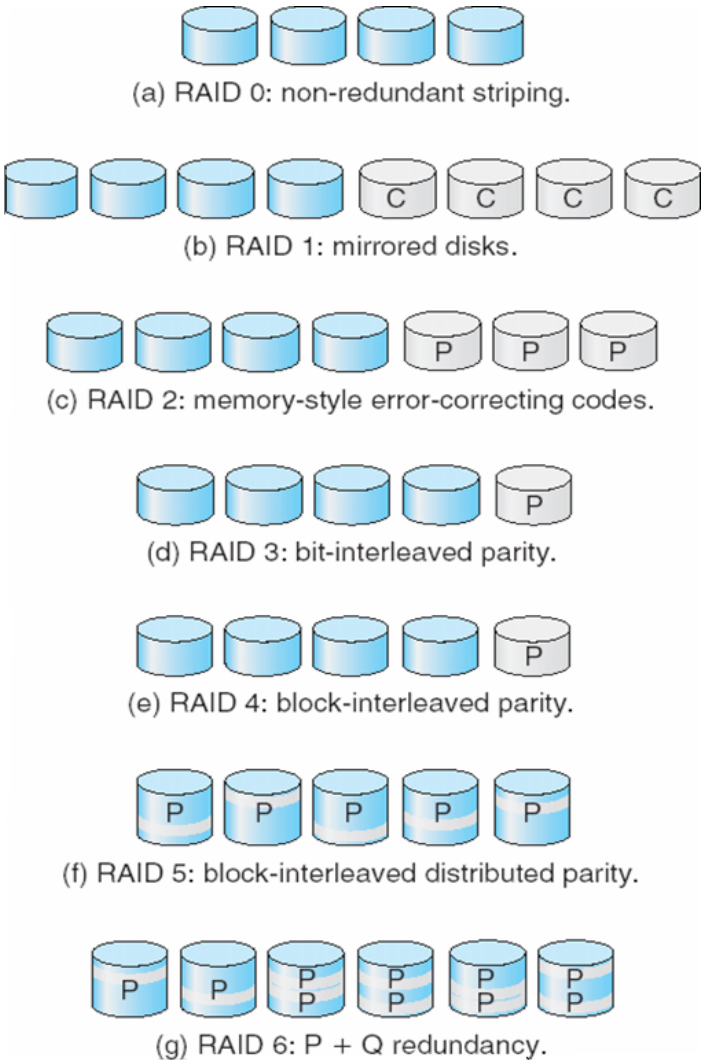
RAID (Cont.)

- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk striping uses a group of disks as one storage unit
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - Mirroring or shadowing (RAID 1) keeps duplicate of each disk
  - Striped mirrors (RAID 1+0) or mirrored stripes (RAID 0+1) provides high performance and high reliability
  - Block interleaved parity (RAID 4, 5, 6) uses much less redundancy

RAID (Cont.)

- RAID within a storage array can still fail if the array fails, so automatic replication of the data between arrays is common
- Frequently, a small number of hot-spare disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

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.

RAID 0

- RAID 0 - Striped disk array without parity.
  - Data spread across multiple disk drives but no data redundancy
  - Improves performance because multiple reads and writes can be carried out at the same time.
  - Works best with large requests.
  - Works worst with OS that ask for data one sector at a time.
    - No parallelism

RAID 0

- RAID 0 - Striped disk array without parity.
  - Does not increase fault tolerance.
  - Actually decreases reliability since if one drive fails the all data is lost since the OS treats the array as a single drive.
    - N drives means configuration is N times as likely to fail.

RAID 1

- RAID 1 - Mirroring
  - Duplicate set of drives
  - When data is written to one it's also written to its duplicate.
    - When one fails, swap in new and copy the data over.

RAID 1

- RAID 1 - Mirroring
  - Assuming drive and its mirror can be read at the same time it provides twice the read transaction rate
  - Write transaction is unaffected.
    - No performance gain
  - Most expensive raid configuration

RAID 2 and RAID 3

- RAID 2 - Error-correcting coding
- RAID 3 - Bit interleaved parity
- Prohibitively expensive and inferior to other RAID levels.
- Drive spinning must be synchronized

RAID 4

- RAID 4 - Dedicated parity drive
- Block level striping like RAID 0 with a parity disk
- If a data disk fails the parity data is used to create a replacement disk.
- Every time a block is written the parity block must also be read, recalculated and rewritten. I/O bottleneck.

RAID 4

- Same reliability as RAID 1 but if a drive fails the performance hit is worse.
- Heavy load on the parity drive
  - Bottleneck

RAID 5

- RAID 5 - Block interleaved distribution parity
- Like RAID 4 but instead of parity on the same drive the parity block is assigned to the drives in a round robin fashion.
- Removes excessive use of the parity drive.

RAID 6

- RAID 6 - Independent data disks with double parity
- Block-level striping with parity across the disks like RAID 5 but instead of a simple parity scheme it calculates parity using two different algorithms.
- Requires an extra disk drive of RAID 5 but will tolerate the loss of two drives at the same time.

Disk Formatting

- Hard disk consists of a stack of aluminum or glass platters.
  - On each is deposited a thin magnetizable metal oxide.
  - No information
- Each platter must receive a low-level format.
  - Series of sectors with short gaps between

Disk Formatting

- Preamble contains bit pattern hardware can recognize the sector.
  - Cylinder number
  - Sector number
- Usually 512 -byte sectors.
- ECC size varies by manufacturers but usually 16-bytes.

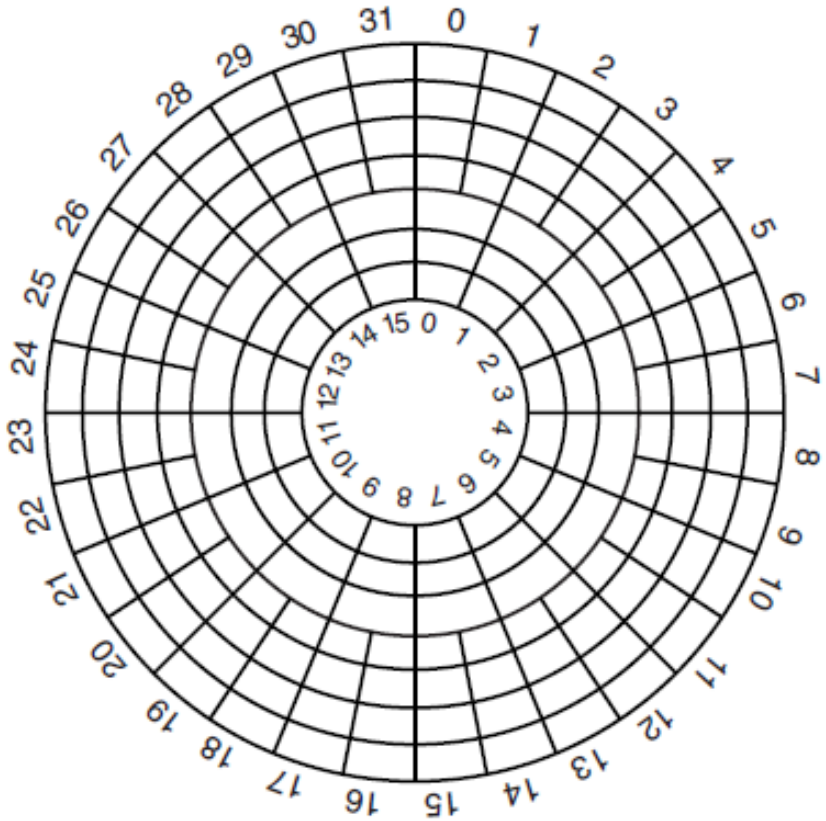
Transfer Time

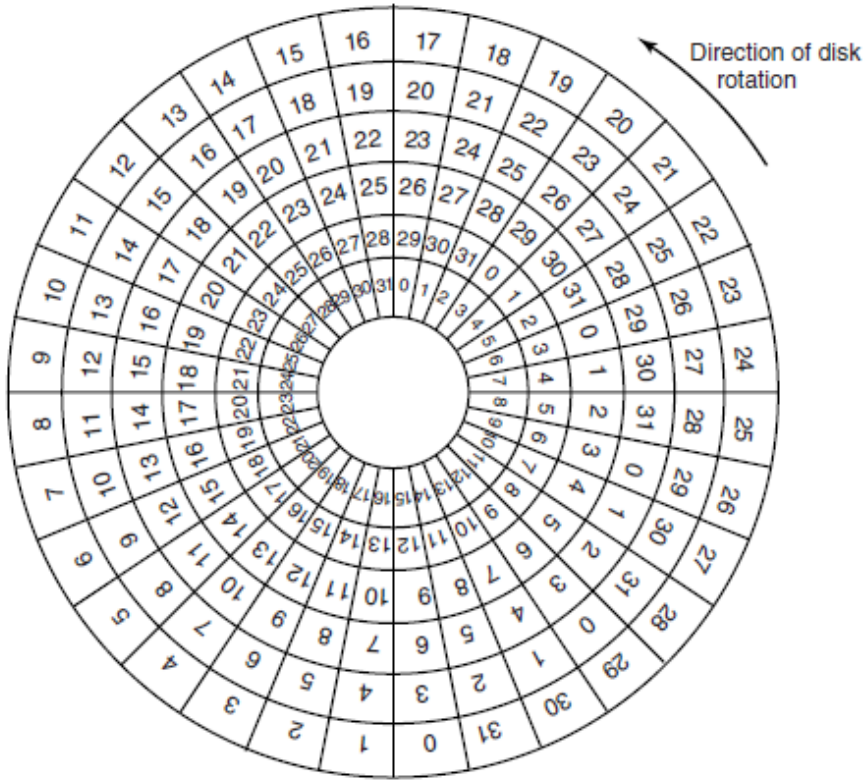
- Consider a disk with 1MB per track, a rotation time of 8.33 ms and a seek time of 5 ms.
- The time in ms to read a block of k bytes is the sum of the seek, rotational delay and transfer times:

$$5 + 4.165 + (k / 1000000) * 8.33$$

Disk Formatting

» Cylinder skew - position of sector 0 is offset form previous track





Skew

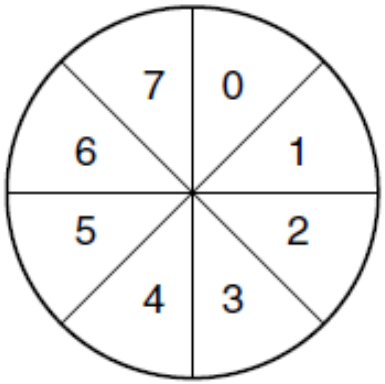
- 10,000 RPM drive rotates in 5 msec. If a track has 300 sectors a new sector passes under the head every 20 usec.
- If the track-to-track seek time is 800 usec, 40 sectors will pass by during the seek so skew must be at least 40 sectors.
- Head skew - much smaller than cylinder skew, less than one sector of time.
- Result means less disk space for storage
  - Up to 20% wasted

Disk Formatting

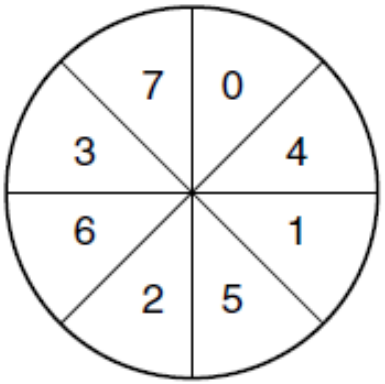
» Transferring from controller to memory also causes delays that must be accounted for:

  – Single interleaving

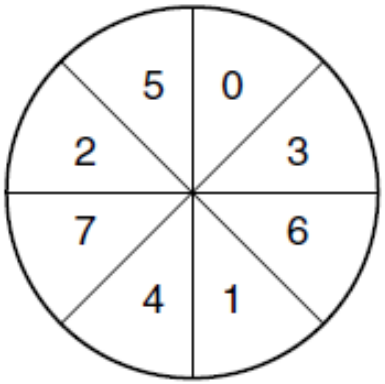
  – Double interleaving



(a)



(b)



(c)

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
- Access time has two major components
  - **Seek time** is the time for the disk are to move the heads to the cylinder containing the desired sector
  - **Rotational latency** is 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

Disk Scheduling (Cont.)

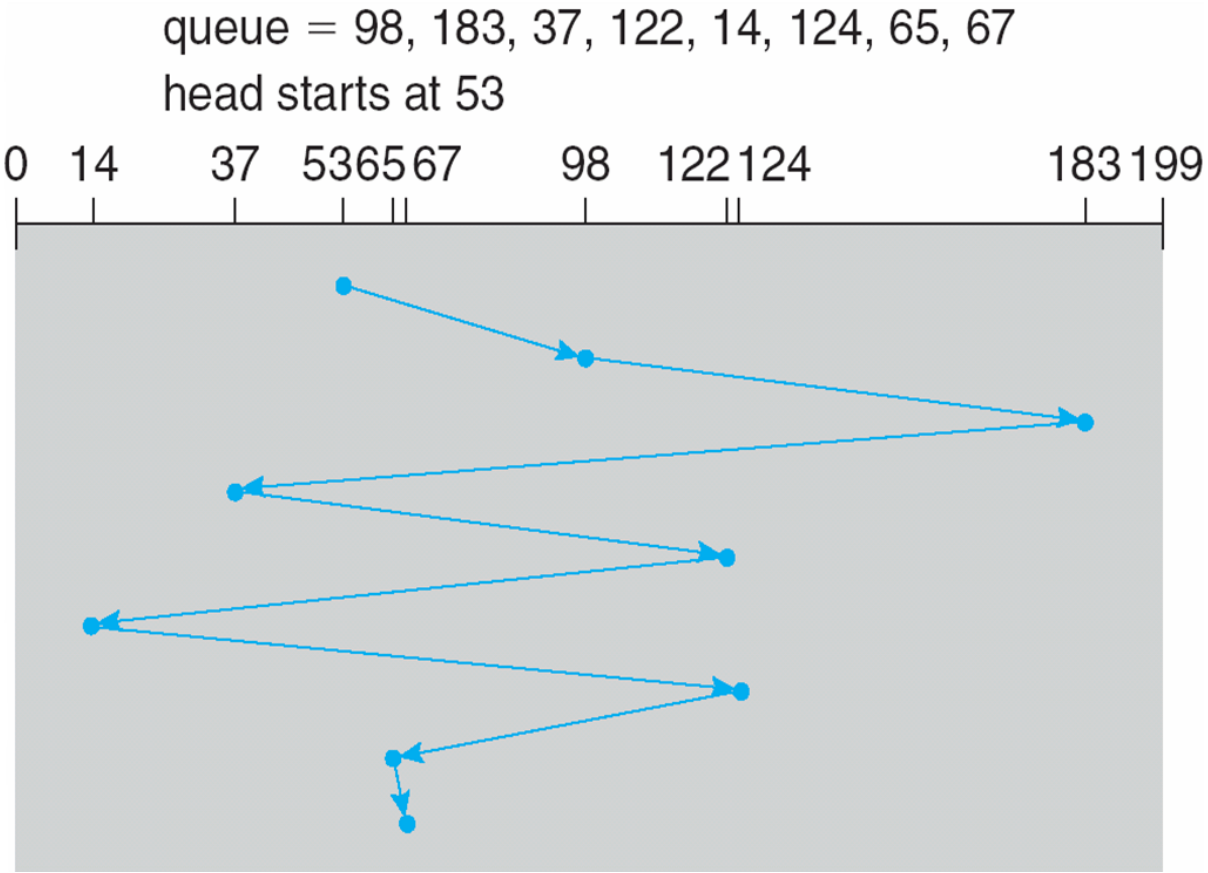
- Several algorithms exist to schedule the servicing of disk I/O requests
- We illustrate them with a request queue (0-199)

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

Head pointer 53

FCFS

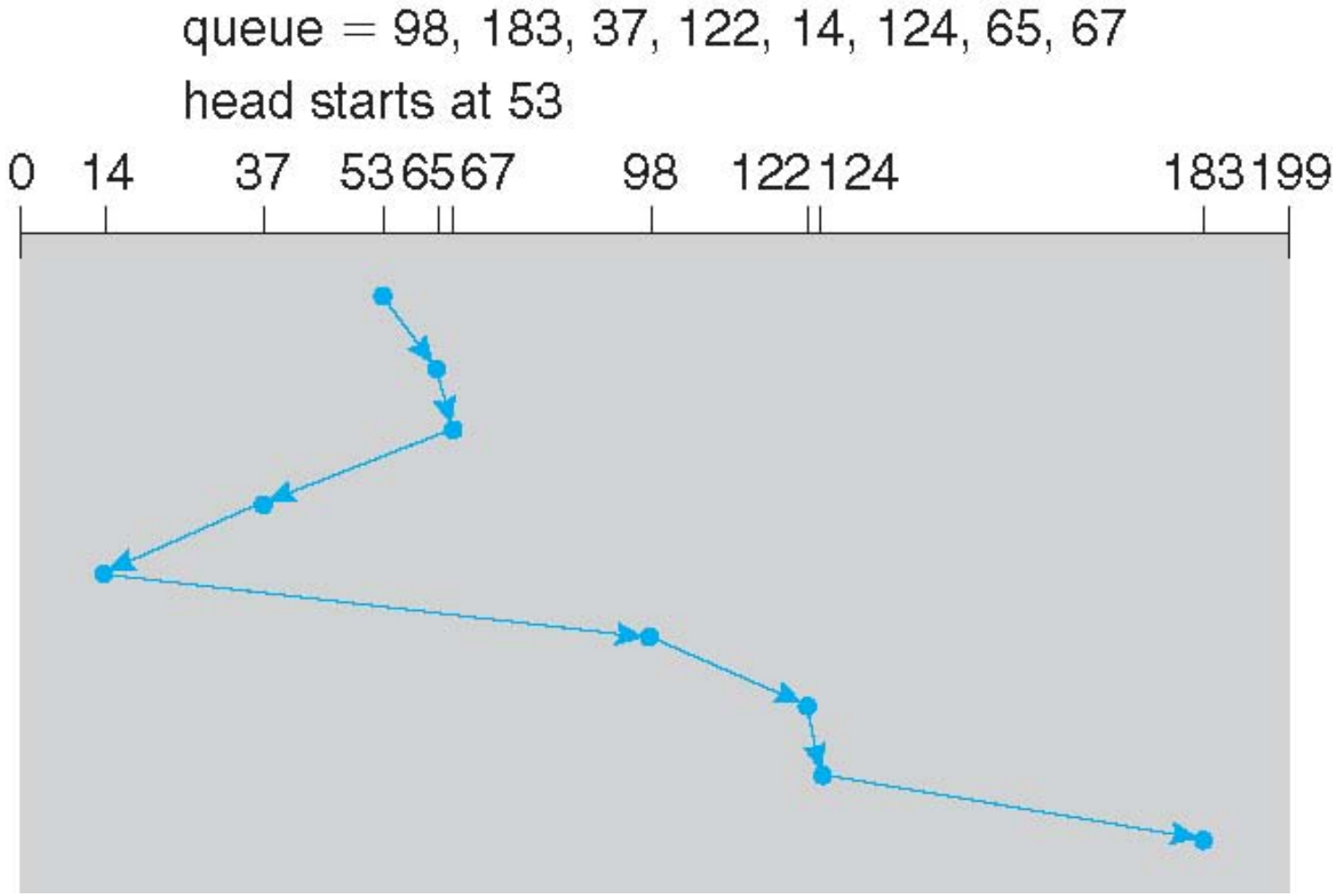
Illustration shows total head movement of 640 cylinders



SSF

- Selects the request with the minimum seek time from the current head position
- SSF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders

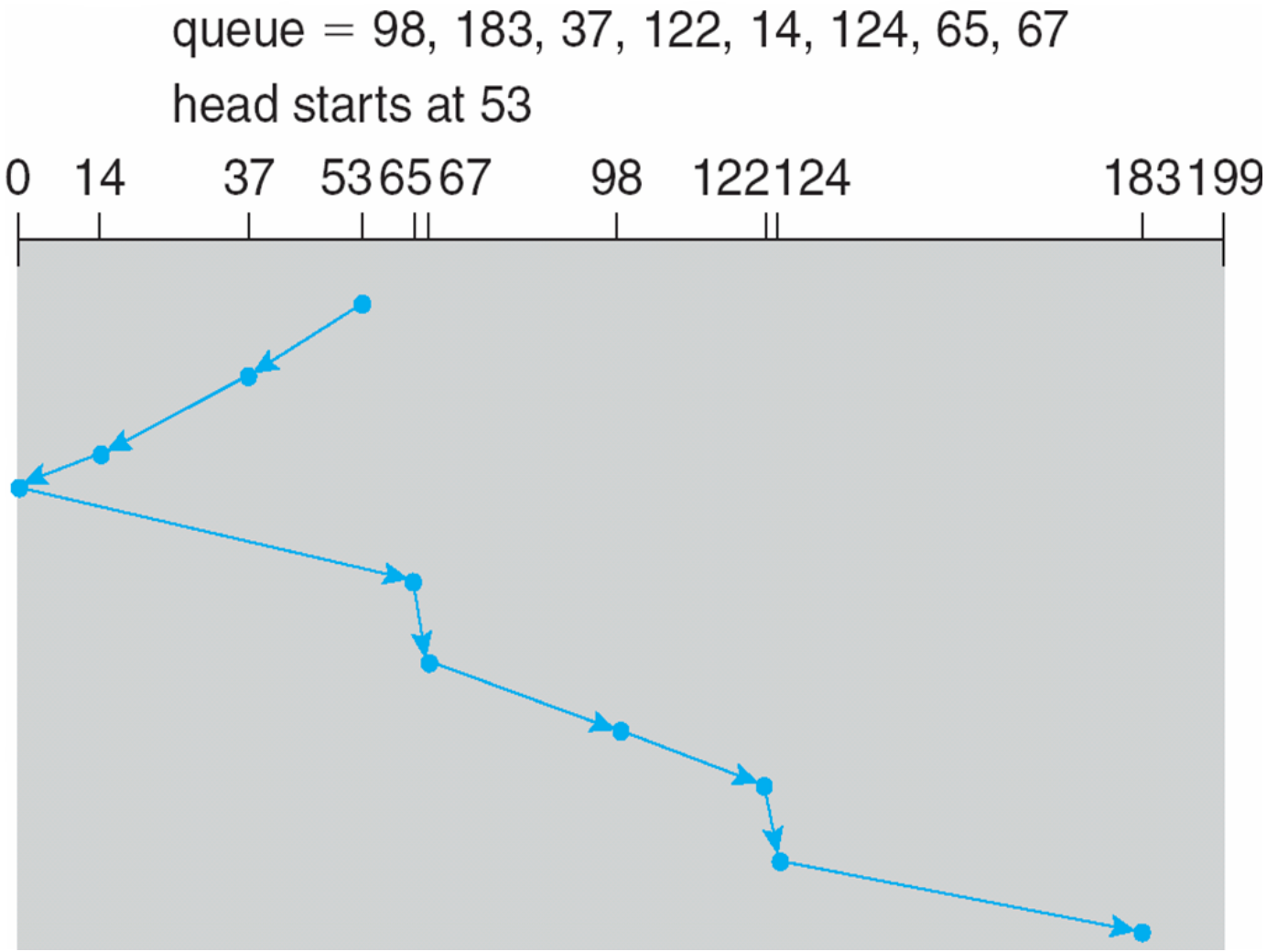
SSF (Cont.)


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

SCAN/Elevator

- 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.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 236 cylinders

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

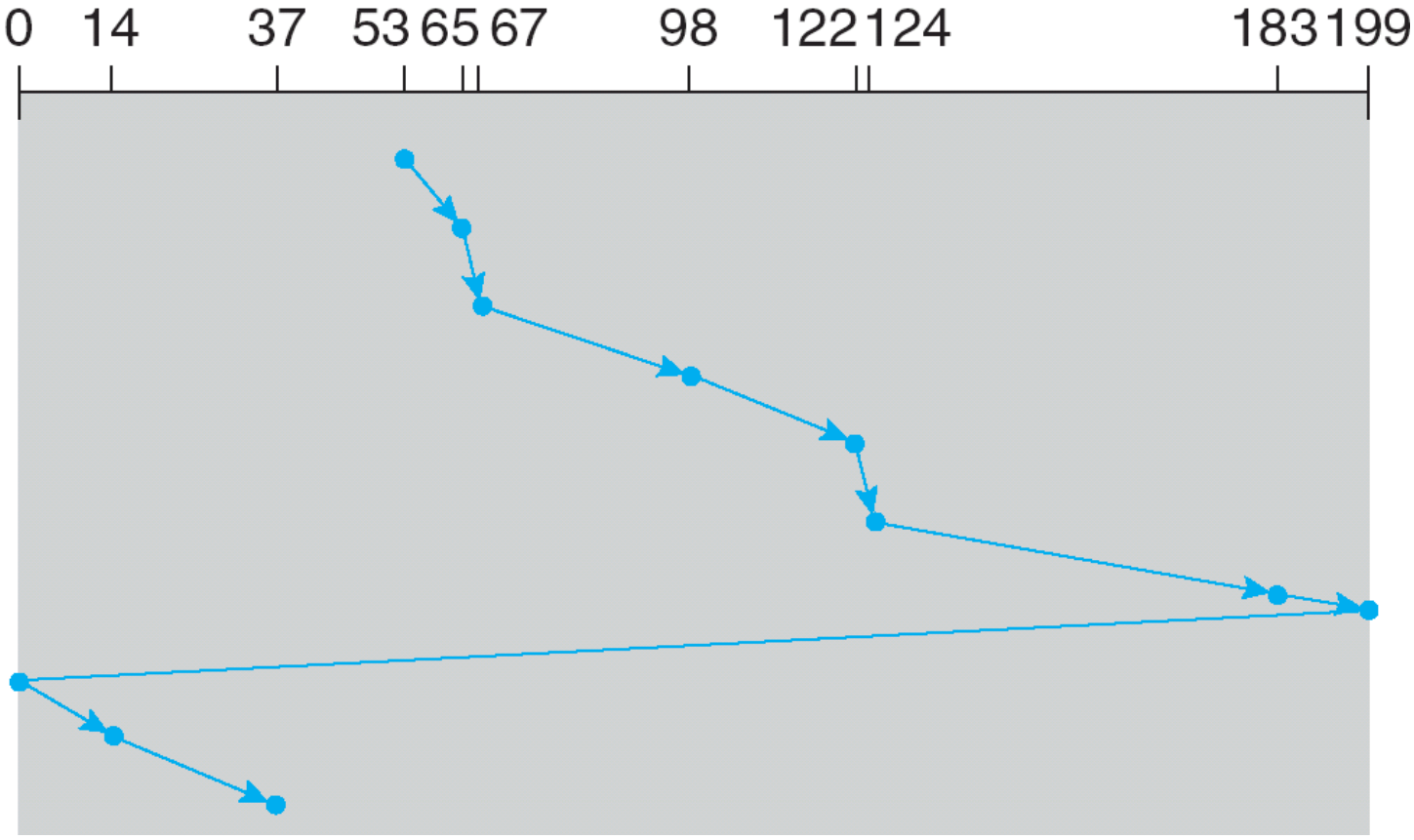
C-SCAN

- 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



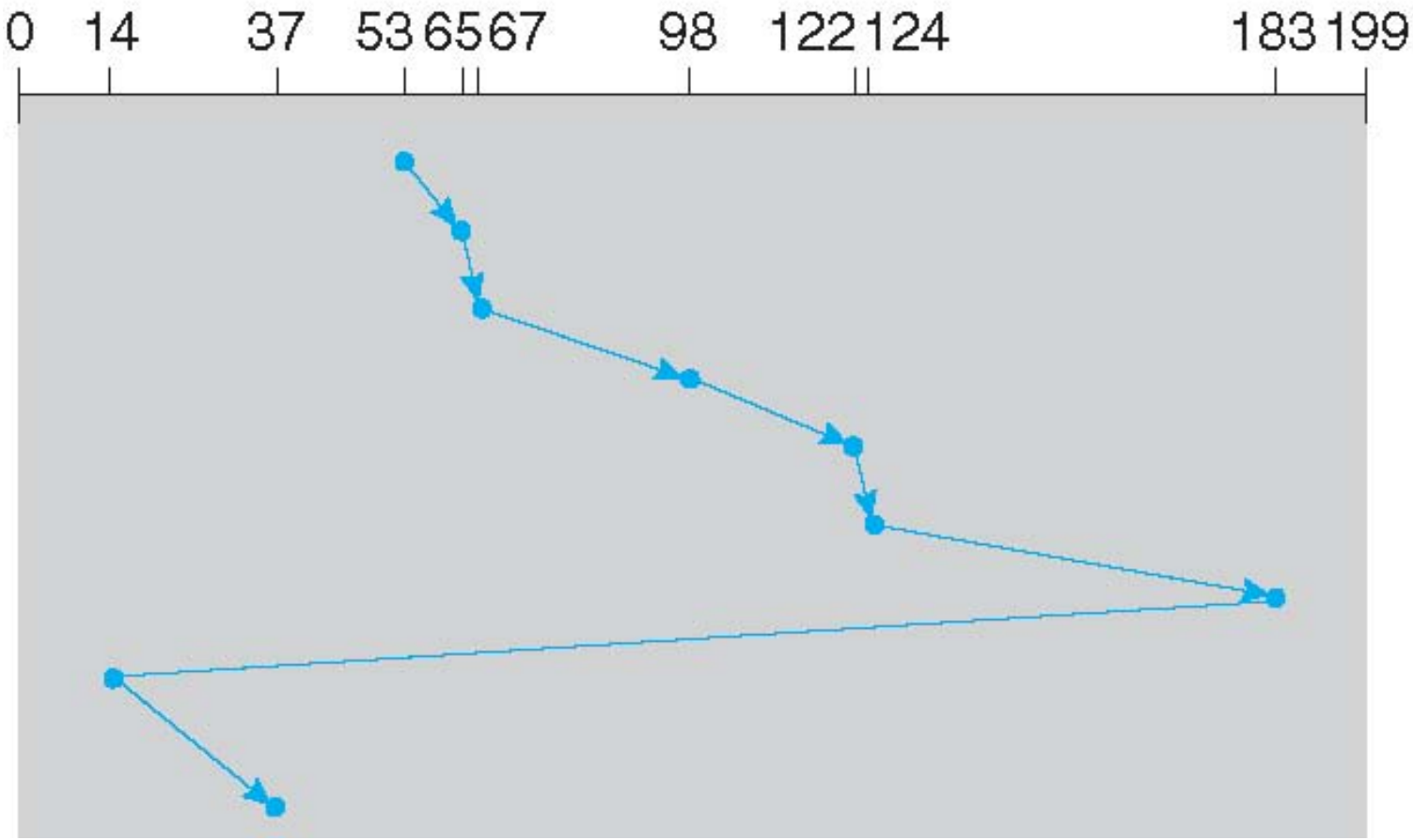
C-LOOK

- 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

C-LOOK (Cont.)

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

head starts at 53



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
- 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 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

Disk Error Handling

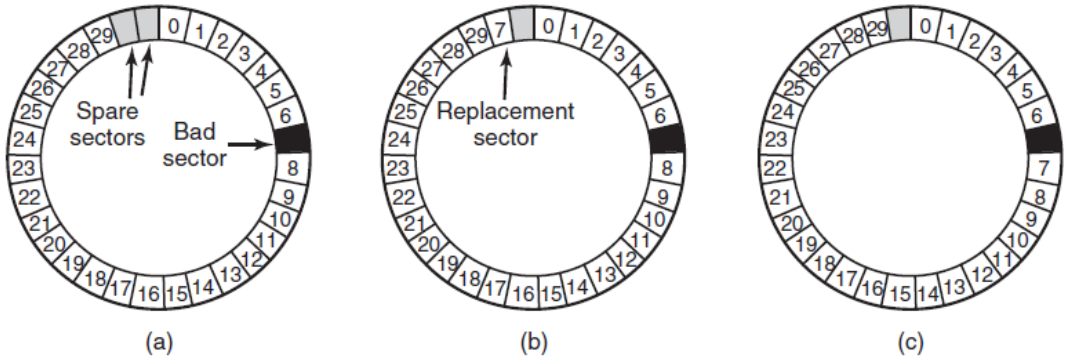
- Increasing linear bit densities constant drive by manufacturers
  - Defects introduced
- Bad sector - sector that does not correctly read back the value written to them.
  - A few bits can be corrected by ECC
  - Larger defects can not be masked.
  - Deal with them in the controller or in the OS

Disk Error Handling

» Before a drive is shipped it is tested and bad sectors are written onto the disk

– Spares substituted

• Substitute or shift



(a) A disk track with a bad sector. (b) Substituting a spare for the bad sector. (c) Shifting all the sectors to bypass the bad one.

Disk Error Handling

» What about errors during runtime?

– The first line of defense if ECC fails, try reading it again

• Some read errors are transient: dust

– Controller can switch to a spare before the sector dies completely

Disk Error Handling

» Option 2: Let the OS handle it
  – OS must acquire a list of bad sectors
    • Read them from the disk
    • Test entire disk
  – Can't let bad sectors reside in free block list or bitmap
    • Create a secret file
  – Backups can't backup bad block file
    • Sector by sector backups an issue

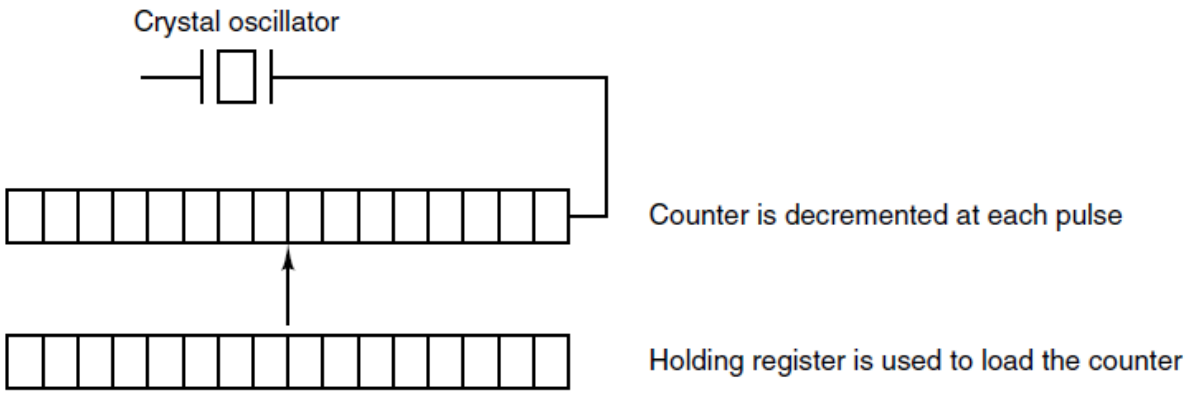
Disk Error Handling

» Seek errors also an issue.

- Mechanical problem with the arm
- Modern controllers handle arm issues automatically
- 80's and 90's floppy drives set error bit and let the driver handle it
  - Driver issued a recalibrate command to move arm as far out as it would go and reset the current cylinder to 0

Clocks

- Essential to the operation of a system
- Original clocks tied to 110 or 220 volt power and caused an interrupt every voltage cycle at 50 or 60 Hz
- More recent built out of crystal oscillator, counter and holding register.



Clocks

- Two modes
  - One shot
  - square-wave mode
- Interrupts called clock ticks
- 500 Mhz crystal will pulse every 2 nsec
  - 32 bit register means interval from 2 sec to 8.6 sec
- Battery backup
- UNIX / Windows Epoch

Clocks

- Typical duties of a clock driver:
- Maintaining the time of day.
- Preventing processes from running longer than allowed.
- Accounting for CPU usage.
- Handling alarm system call from user processes.
- Providing watchdog timers for parts of system itself.
- Profiling, monitoring, statistics gathering.

Clocks

- 32 bit at 60Hz will overflow in a bit over 2 years
  - How can we store ticks since Jan 1, 1970?
  - 1. 64 bit counter
  - 2. Store time in seconds rather than ticks
    - Use secondary counter for ticks this second
    - 136 years
  - 3. Calculate ticks since boot
    - On boot store current time in memory
    - Use ticks as offset.

Clocks

- Processes can request clock notifications
- If driver has enough clocks it can assign one to each process
  - Never enough clocks
  - Driver instead maintains a table with pending events
    - Store evens as linked list indexed by time
- Watchdog timers
  - Timer and procedure must be in the same address space

Soft Timers

» Interrupts have significant overhead.
  – Context switches
» Polling has high latency
» Soft timers avoid interrupts
  – Check real time mode every time before leaving kernel mode.
    • If timer expires then perform the event.
    • Already in kernel mode, no overhead

Soft Timers

» Soft timers stand or fall with the rate at which kernel entries are made for other reasons. These reasons include:
  – System calls.
  – TLB misses.
  – Page faults.
  – I/O interrupts.
  – The CPU going idle.

» Average varies from 2 to 18 usec. 10 usec