R

بسم الله الرحمن الرحیم

سیستم عامل

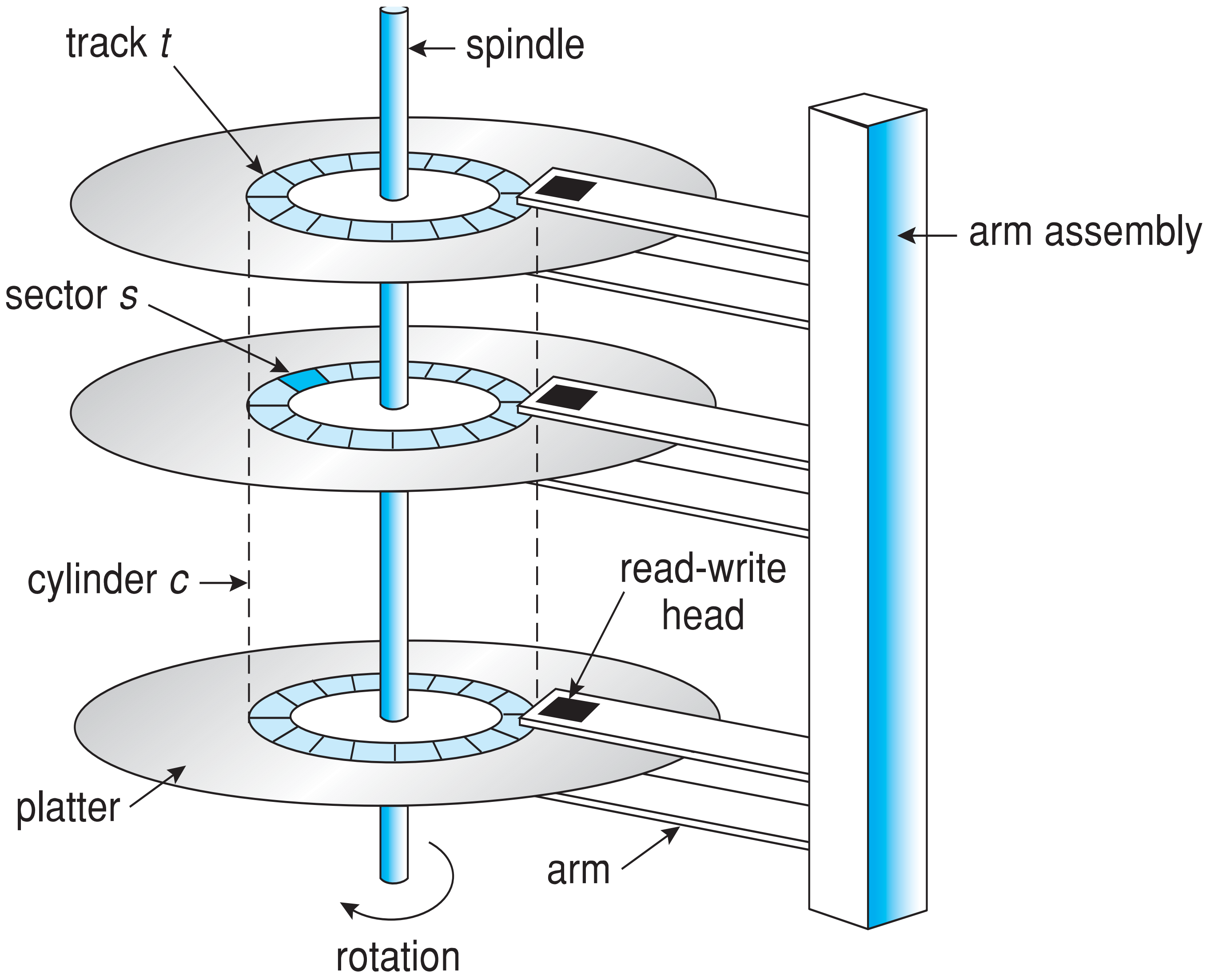
جلسه بیست و دوم – حافظه‌ی جانبی (۲)

جلسه‌ی گذشته

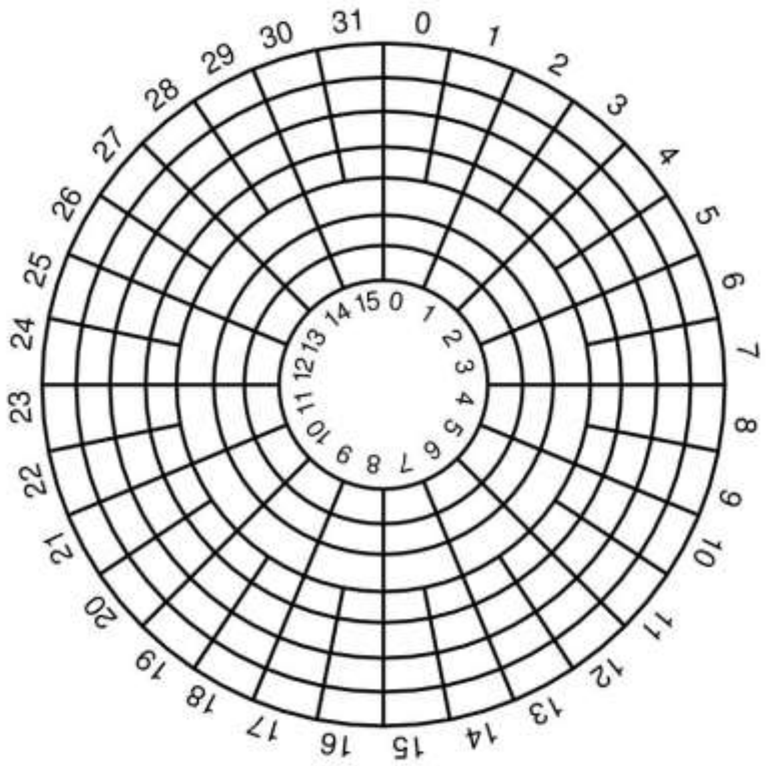
Overview of Mass Storage Structure

- Bulk of secondary storage for modern computers is **hard disk drives** (**HDDs**) and **nonvolatile memory** (**NVM**) devices
- **HDDs** spin platters of magnetically-coated material under moving read-write heads
  - *Drives 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

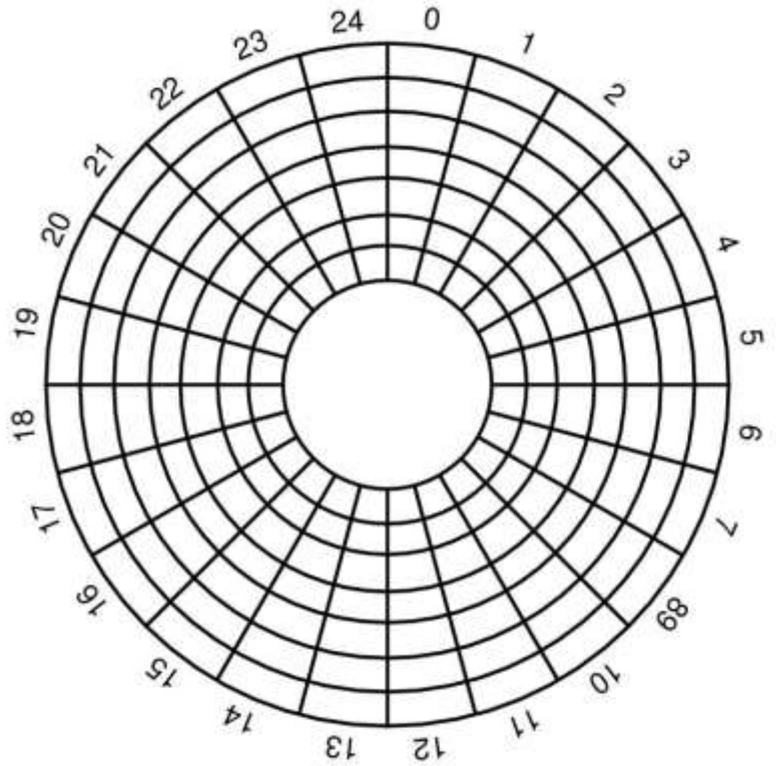
Disk Geometry



Virtual Geometry



physical geometry

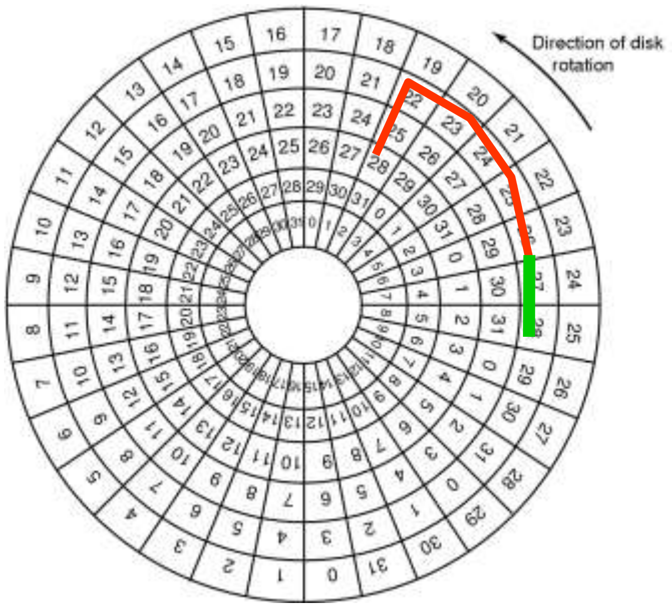


virtual geometry

Disk Scheduling Algorithms

- Time required to read or write a disk block determined by 3 factors
  - *Seek time*
  - *Rotational delay*
  - *Actual transfer time*

- Seek time dominates
  - *Schedule disk heads to minimize it!*

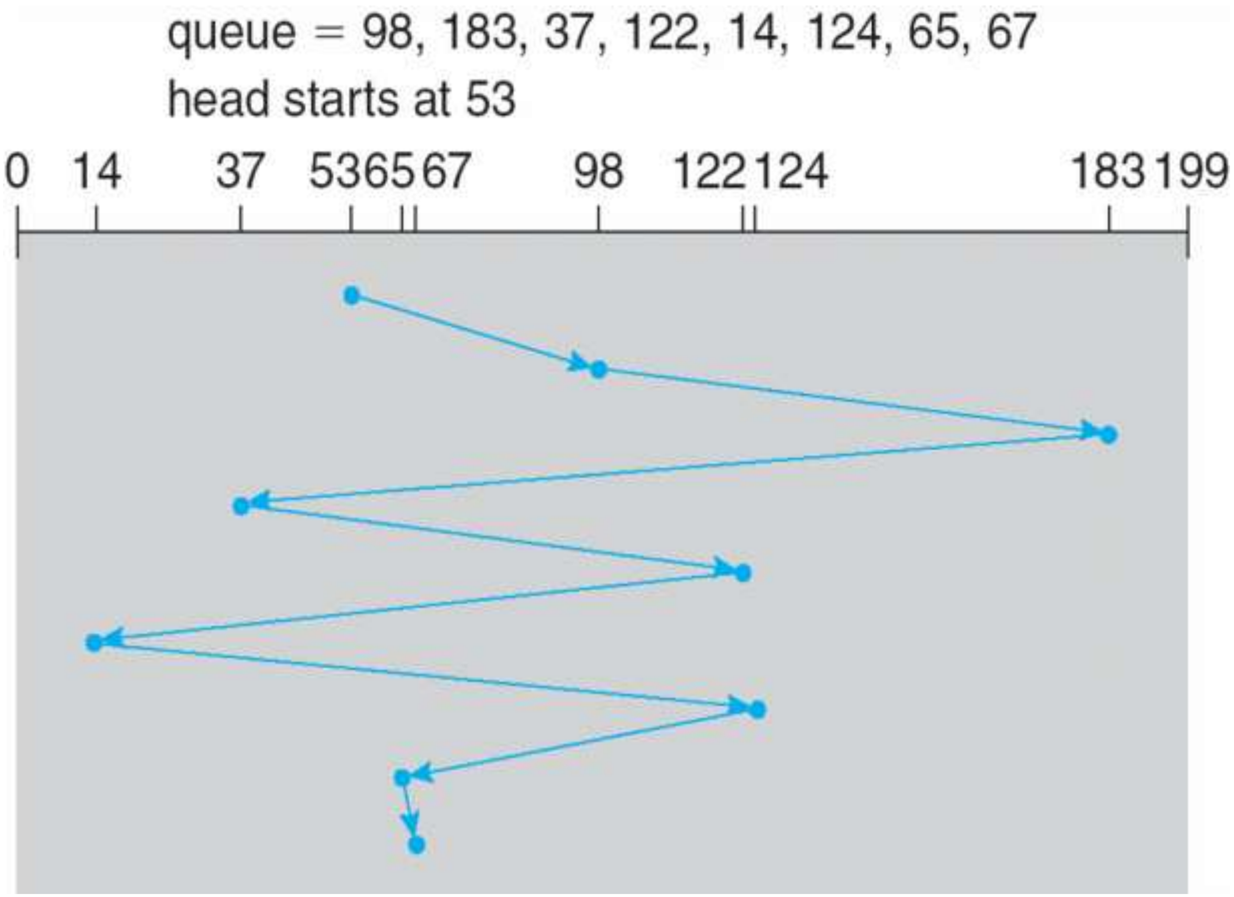


Disk Scheduling Algorithms

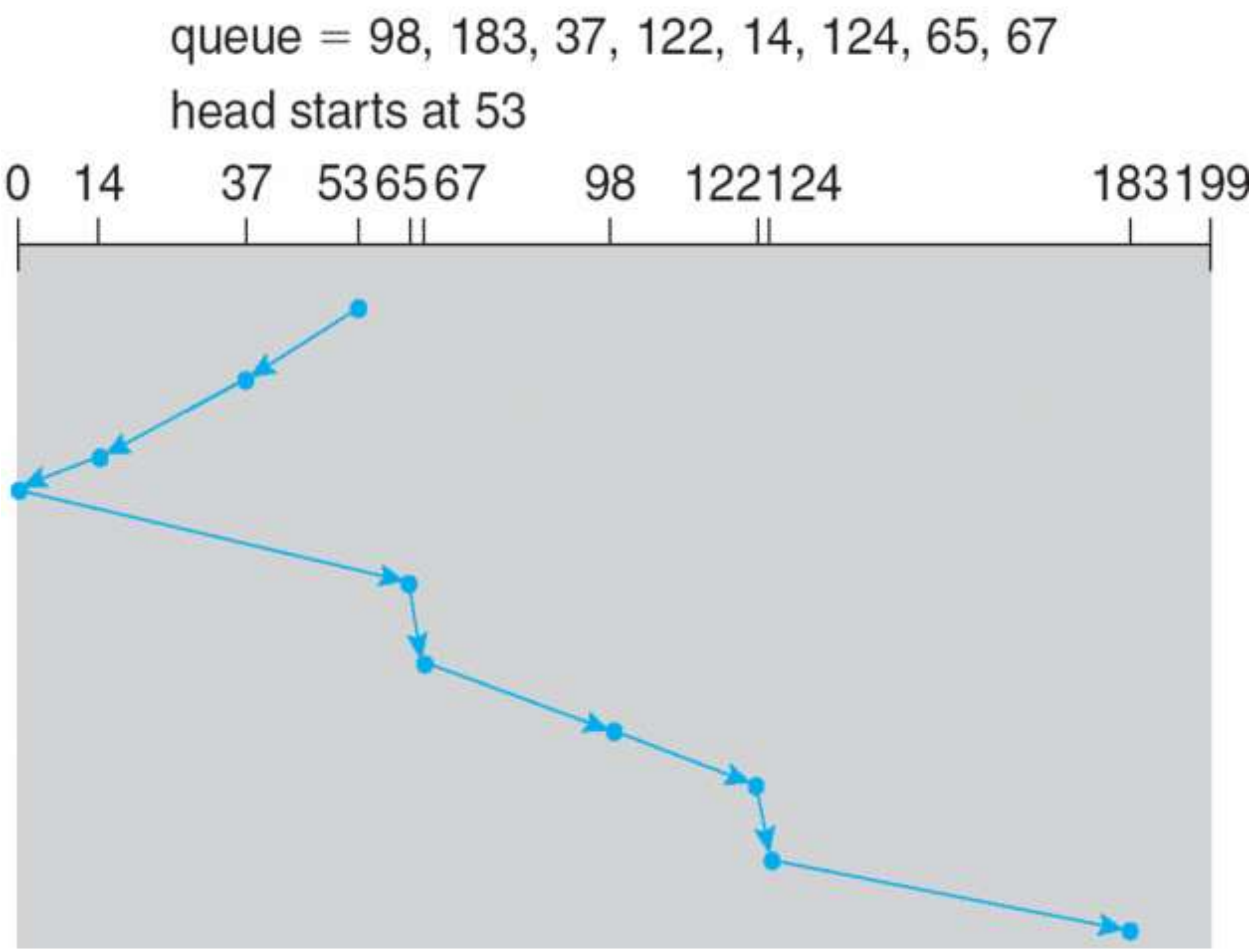
- First-come first serve
- Shortest seek time first
- Scan → back and forth to ends of disk
- C-Scan → only one direction
- Look → back and forth to last request
- C-Look → only one direction

FCFS

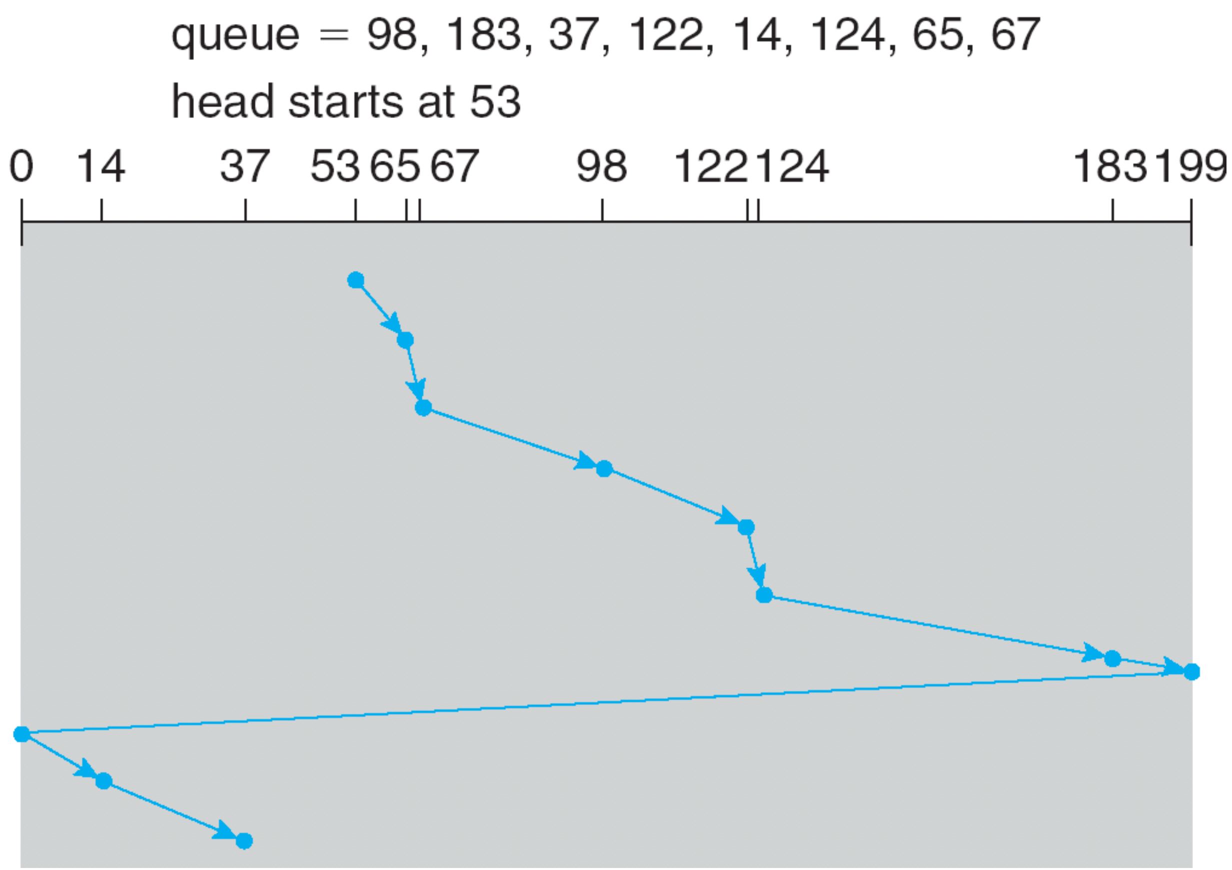
Illustration shows total head movement of 640 cylinders



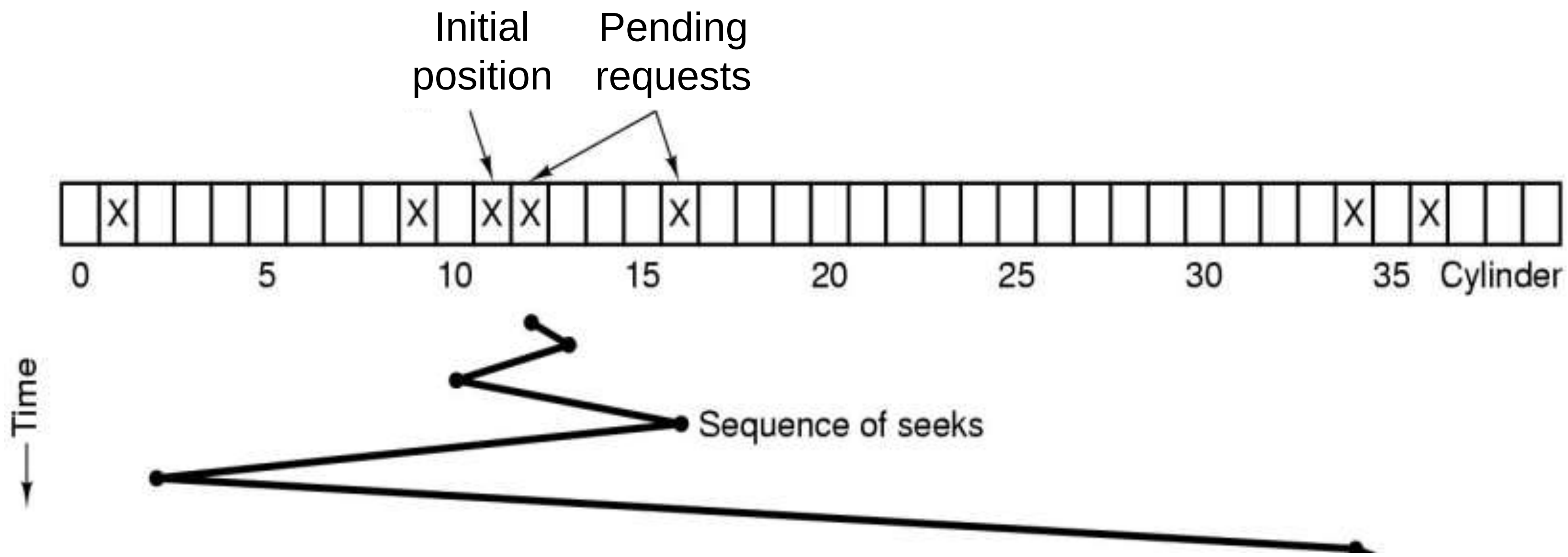
SCAN (Cont.)


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 (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

Shortest Seek First (SSF)



Shortest Seek First (SSF)

- Cuts arm motion in half
- Fatal problem:
  - *Starvation is possible!*

The Elevator Algorithm

- Use one bit to track which direction the arm is moving
  - *Up*
  - *Down*
- Keep moving in that direction
- Service the next pending request in that direction
- When there are no more requests in the current direction, reverse direction

Other Algorithms

- First-come first serve
- Shortest seek time first
- Scan → back and forth to ends of disk
- C-Scan → only one direction
- Look → back and forth to last request
- C-Look → only one direction

جلسه‌ی جدید

SSD

Nonvolatile Memory Devices

- If disk-drive like, then called **solid-state disks** (**SSDs**)
- Other forms include **USB drives** (thumb drive, flash drive), DRAM disk replacements, surface-mounted on motherboards, and main storage in devices like smartphones
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span – need careful management
- Less capacity
- But much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency

Nonvolatile Memory Devices

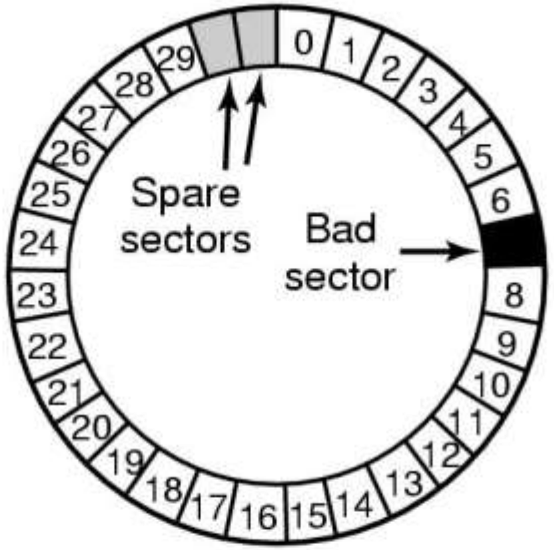
- Have characteristics that present challenges
- Read and written in “page” increments (think sector) but can’t overwrite in place
  - *Must first be erased, and erases happen in larger ”block” increments*
  - *Can only be erased a limited number of times before worn out – ~ 100,000*
  - *Life span measured in **drive writes per day** (**DWPD**)*
    - A 1TB NAND drive with rating of 5DWPD is expected to have 5TB per day written within warrantee period without failing

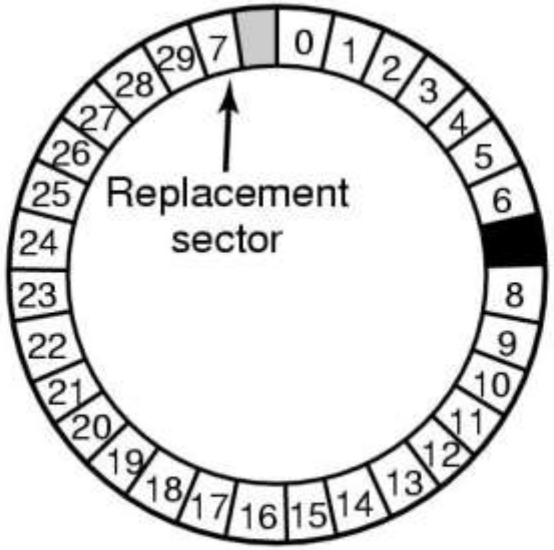
خطاهای دیسک

Disks Errors

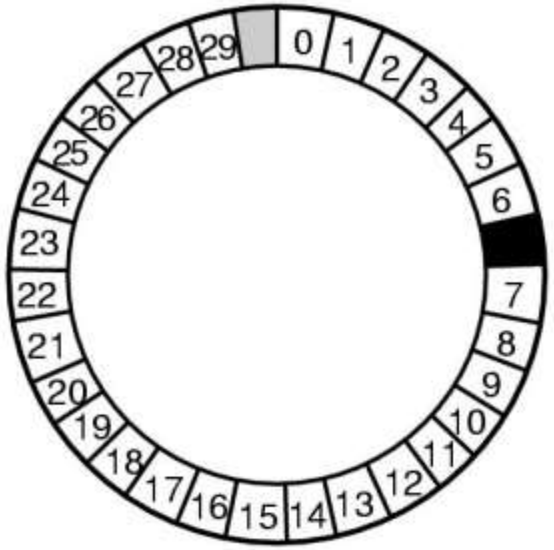
- Transient errors v. hard errors
- Manufacturing defects are unavoidable
  - *Some will be masked with the ECC in each sector*
- Dealing with bad sectors
  - *Allocate several spare sectors per track*
- At the factory, some sectors are remapped to spares
  - *Errors may also occur during the disk lifetime*
- The sector must be remapped to a spare
  - *By the OS*
  - *By the device controller*

Spare Sectors





Substituting
a new sector



Shifting
sectors

Software Handling of Bad Sectors

- Add all bad sectors to a special file
  - *The file is hidden; not in the file system*
  - *Users will never see the bad sectors*

- Backups
  - *Some backup programs copy entire* <u>*tracks*</u> *at a time*
  - *Must be aware of bad sectors!*

STABLE STORAGE

Stable Storage

- The model of possible errors:
  - *Disk writes a block and reads it back for confirmation*
  - *If there is an error during a write it will be detected upon reading the block*
  - *Disk blocks can go bad spontaneously but subsequent reads will detect the error*
  - *CPU can fail but failed writes are detectable errors*
  - *Highly unlikely to loose the same block on two disks (on the same day)*

Stable Storage

- Use two disks for redundancy
- Each write is done twice
  - *Each disk has N blocks*
  - *Each disk contains exactly the same data*
- To read the data ...
  - *you can read from either disk*
- To perform a write ...
  - *you must update the same block on both disks*
- If one disk goes bad ...
  - *You can recover from the other disk*

Stable Storage

- Stable write
  - *Write block on disk # 1*
  - *Read back to verify*
  - *If problems...*
    - Try again several times to get the block written
    - Then declare the sector bad and remap the sector
    - Repeat until the write to disk #1 succeeds
  - *Write same data to corresponding block on disk #2*
    - Read back to verify
    - Retry until it also succeeds

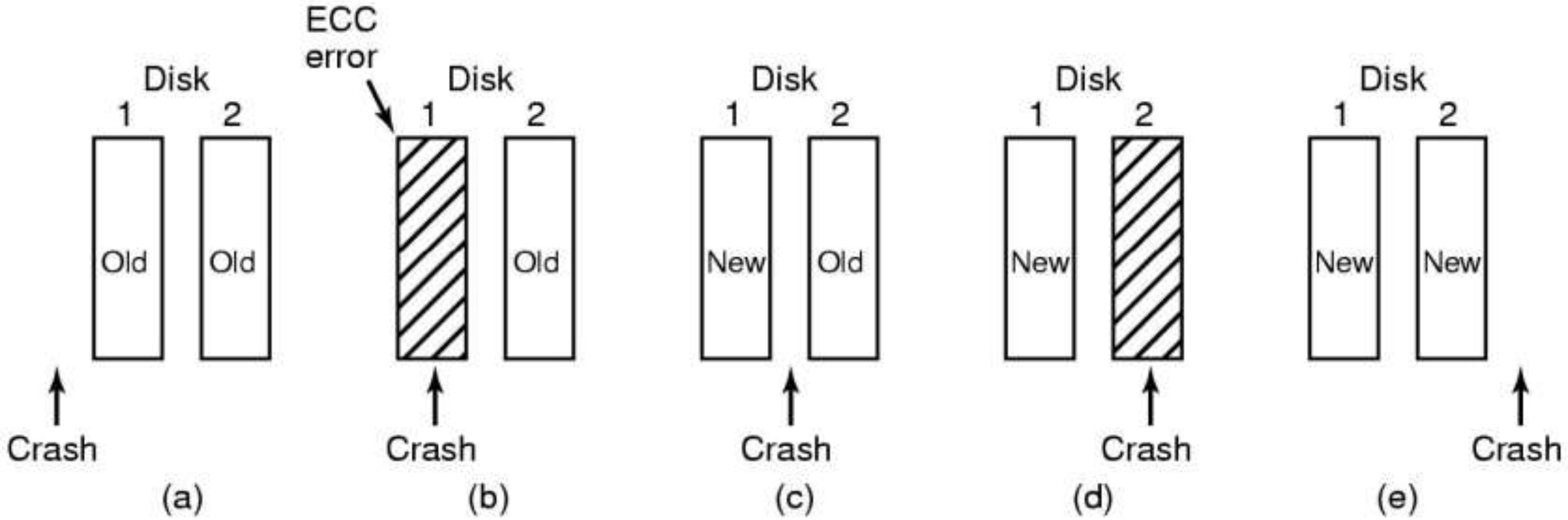
Stable Storage

- Stable Read
  - *Read the block from disk # 1*
  - *If problems...*
    - Try again several times to get the block
  - *If the block can not be read from disk #1...*
    - Read the corresponding block from disk #2
  - *Our Assumption:*
    - The same block will not simultaneously go bad on both disks

Stable Storage

- Crash Recovery
  - *Scan both disks*
  - *Compare corresponding blocks*
  - *For each pair of blocks...*
    - If both are good and have same data...
      - *Do nothing; go on to next pair of blocks*
    - If one is bad (failed ECC)...
      - *Copy the block from the good disk*
    - If both are good, but contain different data...
      - *(CPU must have crashed during a "Stable Write")*
      - *Copy the data from disk #1 to disk #2*

Crashes During a Stable Write



Stable Storage

- Disk blocks can spontaneously decay
- Given enough time...
  - *The same block on both disks may go bad*
    - Data could be lost!
  - *Must scan both disks to watch for bad blocks (e.g., every day)*
- Many variants to improve performance
  - *Goal: avoid scanning entire disk after a crash*
  - *Goal: improve performance*
    - Every stable write requires: 2 writes & 2 reads
    - But we can do better ...

RAID

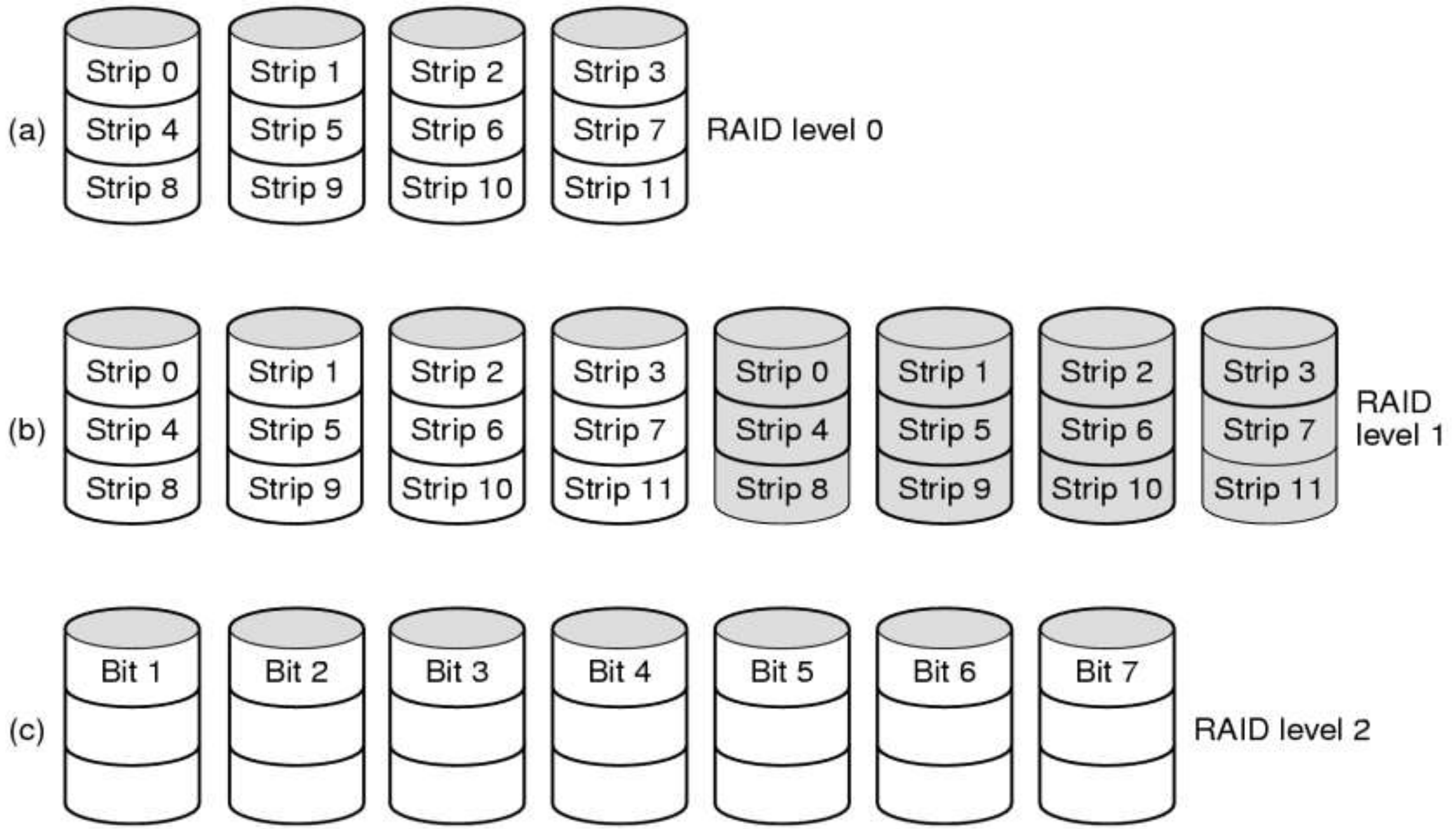
- Redundant Array of Independent Disks
- Redundant Array of Inexpensive Disks

- Two goals:
  - *Increased reliability*
  - *Increased performance*

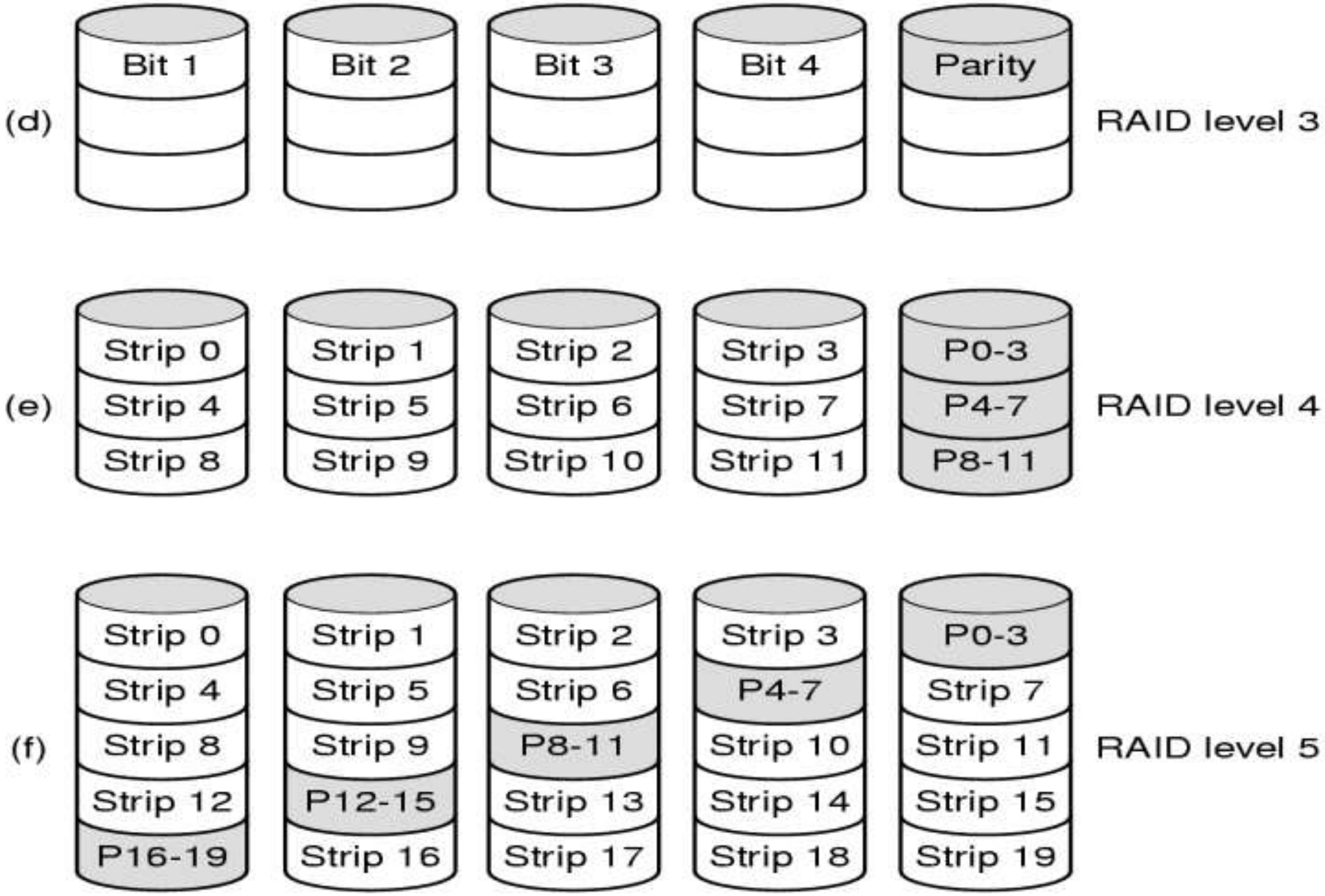
RAID Structure

- **RAID** – **redundant array of inexpensive disks**
  - *multiple disk drives provides reliability via* ***redundancy***
- Increases the **mean time to failure**
- **Mean time to repair –** exposure time when another failure could cause data loss
- **Mean time to data loss** based on above factors
- If mirrored disks fail independently, 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!*

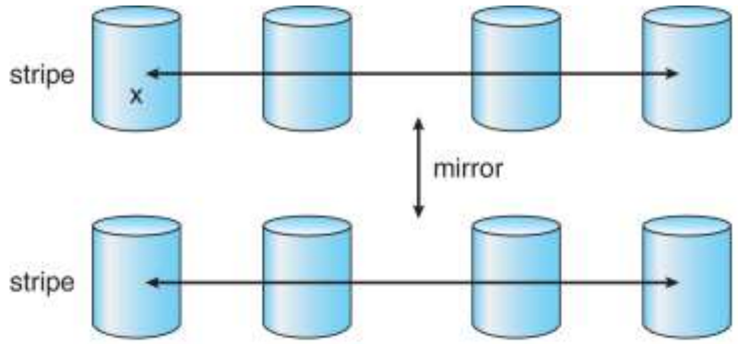
RAID


Strip 0
Strip 1
Strip 2
Strip 3
(a)
Strip 4
Strip 5
Strip 6
Strip 7
RAID level 0
Strip 8
Strip 9
Strip 10
Strip 11
Strip 0
Strip 1
Strip 2
Strip 3
Strip 0
Strip 1
Strip 2
Strip 3
(b)
Strip 4
Strip 5
Strip 6
Strip 7
Strip 4
Strip 5
Strip 6
Strip 7
RAID level 1
Strip 8
Strip 9
Strip 10
Strip 11
Strip 8
Strip 9
Strip 10
Strip 11
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
(c)
RAID level 2

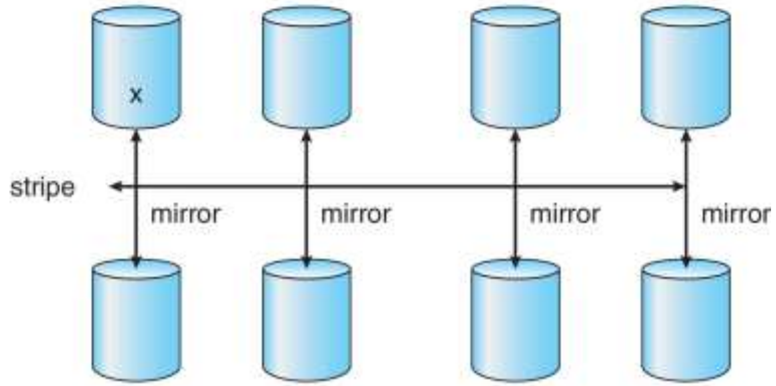
RAID


Bit 1
Bit 2
Bit 3
Bit 4
Parity
(d)
RAID level 3
Strip 0
Strip 1
Strip 2
Strip 3
P0-3
Strip 4
Strip 5
Strip 6
Strip 7
P4-7
Strip 8
Strip 9
Strip 10
Strip 11
P8-11
(e)
RAID level 4
Strip 0
Strip 1
Strip 2
Strip 3
P0-3
Strip 4
Strip 5
Strip 6
P4-7
Strip 7
Strip 8
Strip 9
P8-11
Strip 10
Strip 11
Strip 12
P12-15
Strip 13
Strip 14
Strip 15
P16-19
Strip 16
Strip 17
Strip 18
Strip 19
(f)
RAID level 5

RAID (0 + 1) and (1 + 0)



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

FREE BLOCK

Disk Space Management

- The OS must choose a disk “block” size...
  - *The amount of data written to/from a disk*
  - *Must be some multiple of the disk’s sector size*

- How big should a disk block be?
  - *= Page Size?*
  - *= Sector Size?*
  - *= Track size?*

Disk Space Management

- Large block sizes:
  - *Internal fragmentation*
  - *Last block has (on average) 1/2 wasted space*
  - *Lots of very small files; waste is greater*

- Small block sizes:
  - *More seeks; file access will be slower*

Block Size Tradeoff

- Smaller block size?
  - *Better disk utilization*
  - *Poor performance*

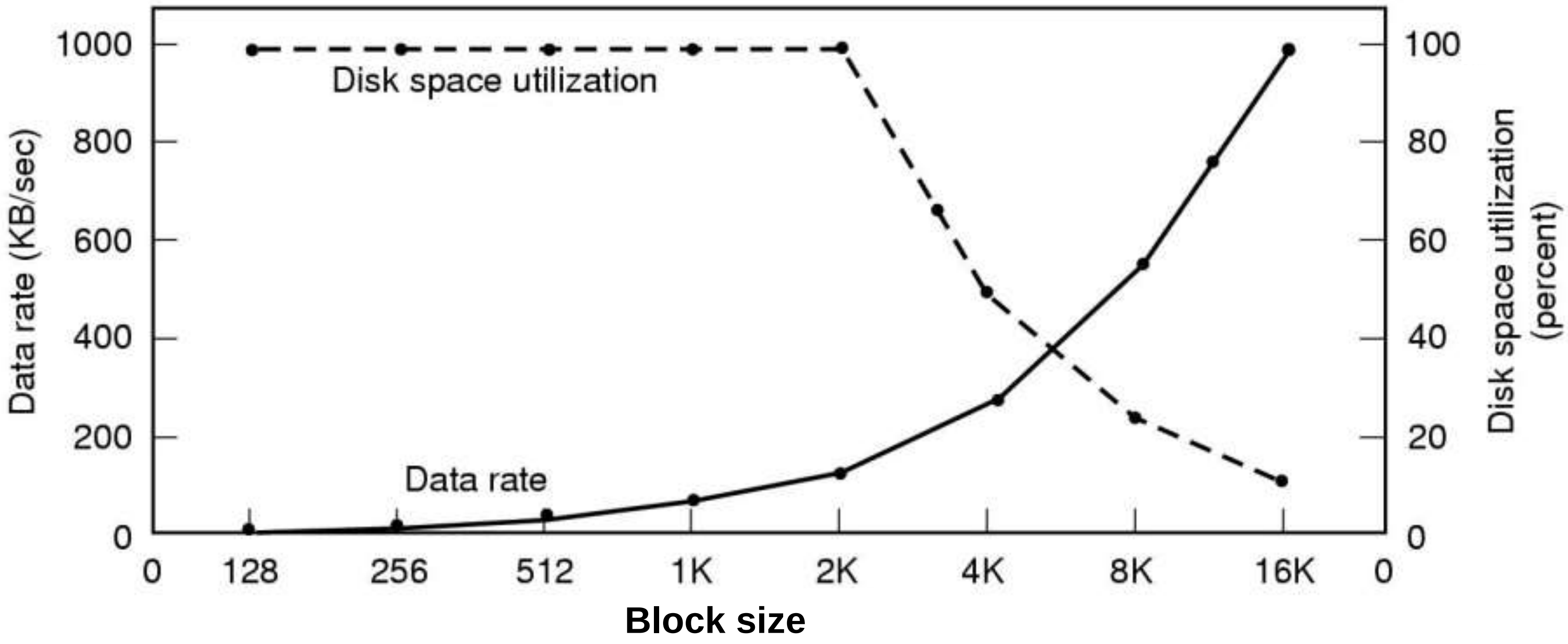
- Larger block size?
  - *Lower disk space utilization*
  - *Better performance*

Simple Example

- A Unix System
  - *1000 users, 1M files*
  - *Median file size = 1,680 bytes*
  - *Mean file size = 10,845 bytes*
  - *Many small files, a few really large files*

- For simplicity, let's assume all files are 2 KB...
  - *What happens with different block sizes?*
  - *The tradeoff will depend on details of disk performance*

Block size tradeoff



Assumption: All files are 2K bytes
Given: Physical disk properties
Seek time=10 msec
Transfer rate=15 Mbytes/sec
Rotational Delay=8.33 msec * 1/2

Managing Free Blocks

- Approach #1:
  - *Keep a bitmap*
  - *1 bit per disk block*

- Approach #2
  - *Keep a free list*

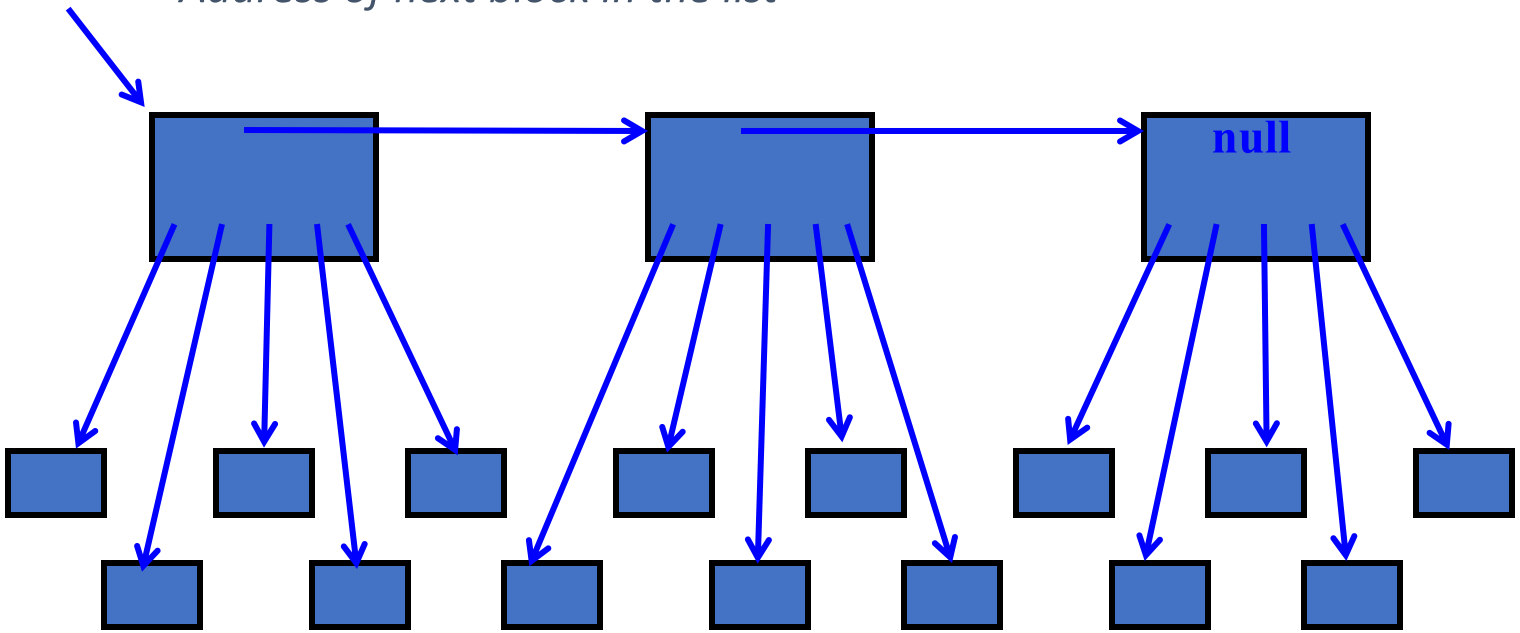
Managing Free Blocks

- Approach #1:
  - *Keep a bitmap*
  - *1 bit per disk block*
    - Example:
      - *1 KB block size*
      - *16 GB Disk $\Rightarrow$ 16M blocks = 2^{24} blocks*
    - Bitmap size = 2^{24} bits $\Rightarrow$ 2K blocks
      - *1/8192 space lost to bitmap*

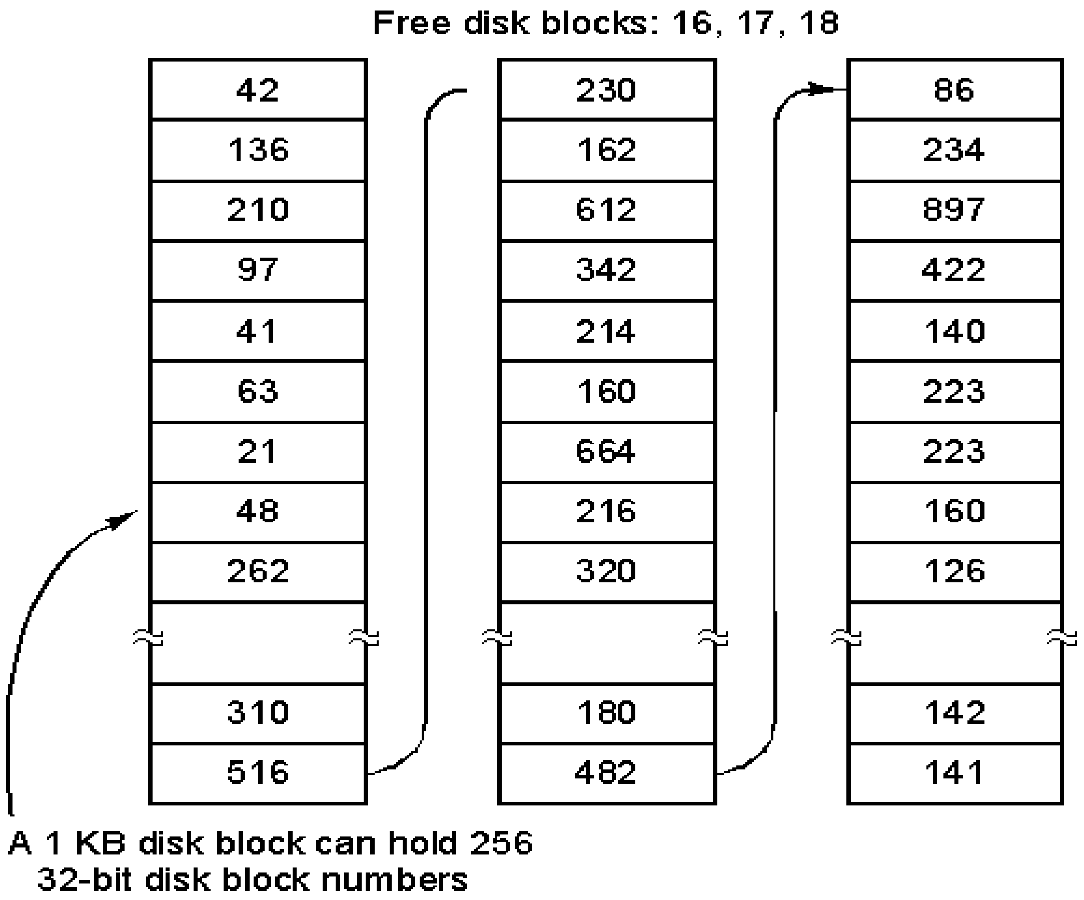
- Approach #2
  - *Keep a free list*

Free List of Disk Blocks

- Linked List of Free Blocks
- Each block on disk holds
  - *A bunch of addresses of free blocks*
  - *Address of next block in the list*



Free list of disk blocks



Assumptions:
- Block size = 1K
- Each block addr = 4bytes
- Each block holds
  - 255 ptrs to free blocks
  - 1 ptr to the next block

This approach takes more space than bitmap…
But "free" blocks are used, so no real loss!

Free List of Disk Blocks

- Two kinds of blocks:
  - *Free Blocks*
  - *Block containing pointers to free blocks*
- Always keep one block of pointers in memory
  - *This block may be partially full*
- Need a free block?
  - *This block gives access to 255 free blocks*
  - *Need more?*
    - Look at the block's "next" pointer
    - Use the pointer block itself
    - Read in the next block of pointers into memory

Free List of Disk Blocks

- To return a block (X) to the free list
  - *If the block of pointers (in memory) is not full, add X to it*

Free List of Disk Blocks

- To return a block (X) to the free list
  - *If the block of pointers (in memory) is not full, add X to it*
  - *If the block of pointers (in memory) is full*
    - Write it to out to the disk
    - Start a new block in memory
    - Use block X itself for a pointer block
      - *All empty pointers except for the next pointer*

Free List of Disk Blocks

- Scenario:
  - *Assume the block of pointers in memory is almost empty*
  - *A few free blocks are needed.*
    - This triggers disk read to get next pointer block
  - *Now the block in memory is almost full*
  - *Next, a few blocks are freed*
  - *The block fills up*
    - This triggers a disk write of the block of pointers
- Problem:
  - *Numerous small allocates and frees, when block of pointers is right at boundary results in lots of disk I/O*

Free list of disk blocks

- Solution
  - *Try to keep the block in memory about 1/2 full*
  - *When the block in memory fills up...*
    - Break it into 2 blocks (each 1/2 full)
    - Write one out to disk
- A similar solution
  - *Keep 2 blocks of pointers in memory at all times*
  - *When both fill up*
    - Write out one
  - *When both become empty*
  - *Read in one new block of pointers*

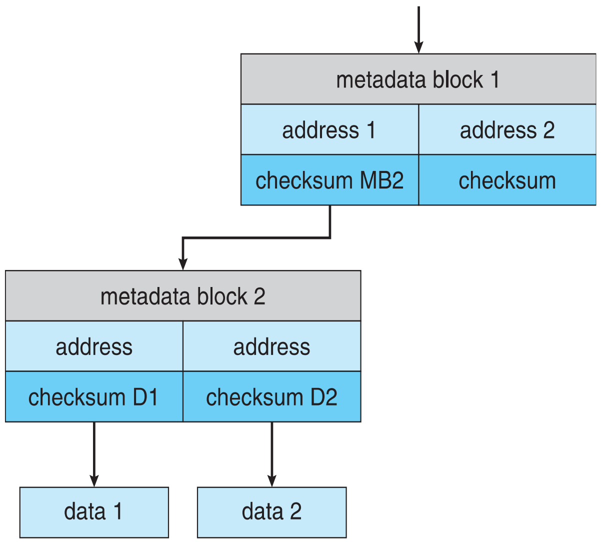
Comparison: Free List vs Bitmap

- Desirable:
  - *Keep all the blocks in one file close together*
- Free Lists:
  - *Free blocks are all over the disk*
  - *Allocation comes from (almost) random location*
- Bitmap:
  - *Much easier to find a free block "close to" a given position*
  - *Bitmap implementation:*
    - Keep 2 MByte bitmap in memory
    - Keep only one block of bitmap in memory at a time

مطالب بیشتر...

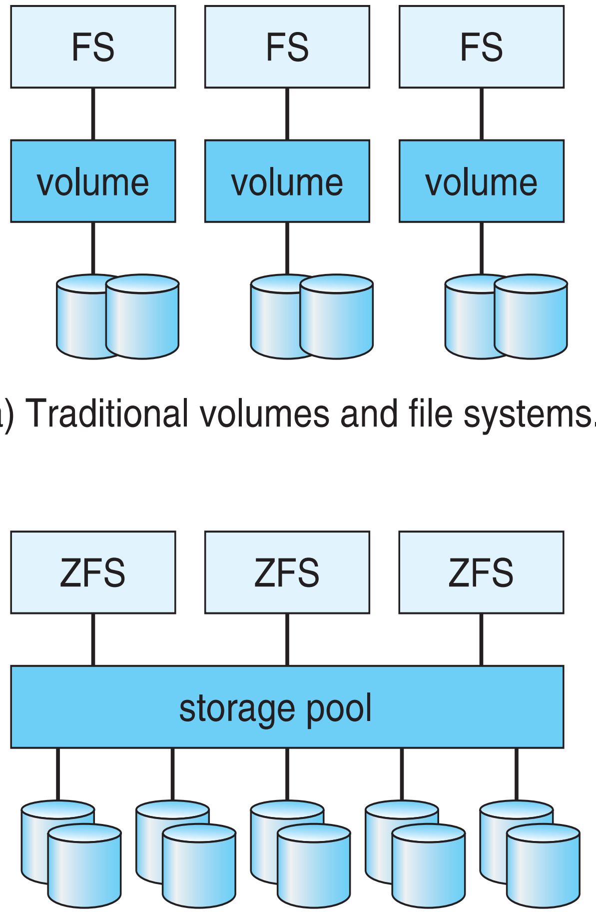
Extensions

- RAID alone does not prevent or detect data corruption or other errors, just disk failures
- Solaris ZFS adds **checksums** of all data and metadata
- Checksums kept with pointer to object, to detect if object is the right one and whether it changed
- Can detect and correct data and metadata corruption
- ZFS also removes volumes, partitions
  - *Disks allocated in* ***pools***
  - *Filesystems with a pool share that pool, use and release space like* **`malloc()`** *and* **`free()`** *memory allocate / release calls*



ZFS checksums all metadata and data

Traditional and Pooled Storage



(a) Traditional volumes and file systems.

(b) ZFS and pooled storage.

Object Storage

- General-purpose computing, file systems not sufficient for very large scale
- Another approach – start with a storage pool and place objects in it
  - *Object just a container of **data***
  - *No way to navigate the pool to find objects (no directory structures, few services*
  - *Computer-oriented, not user-oriented*
- Typical sequence
  - *Create an object within the pool, receive an object ID*
  - *Access object via that ID*
  - *Delete object via that ID*

Object Storage (Cont.)

- Object storage management software like **Hadoop file system** (**HDFS**) and **Ceph** determine where to store objects, manages protection
  - *Typically by storing N copies, across N systems, in the object storage cluster*
  - ***Horizontally scalable***
  - ***Content addressable, unstructured***