

CS 333
Introduction to Operating Systems

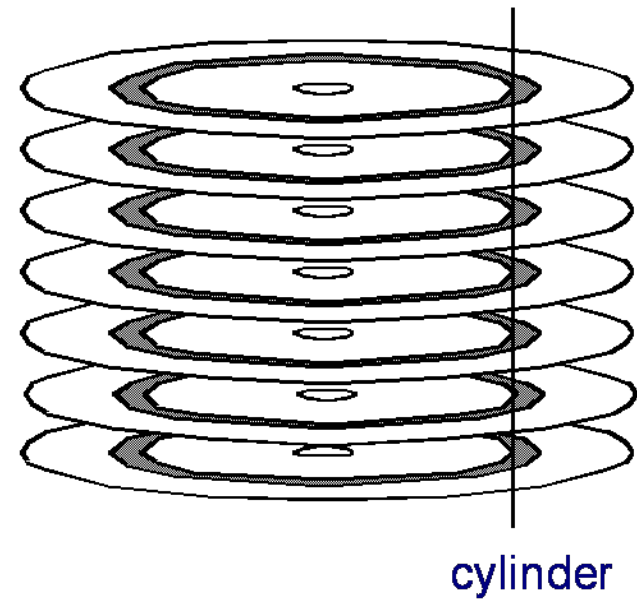
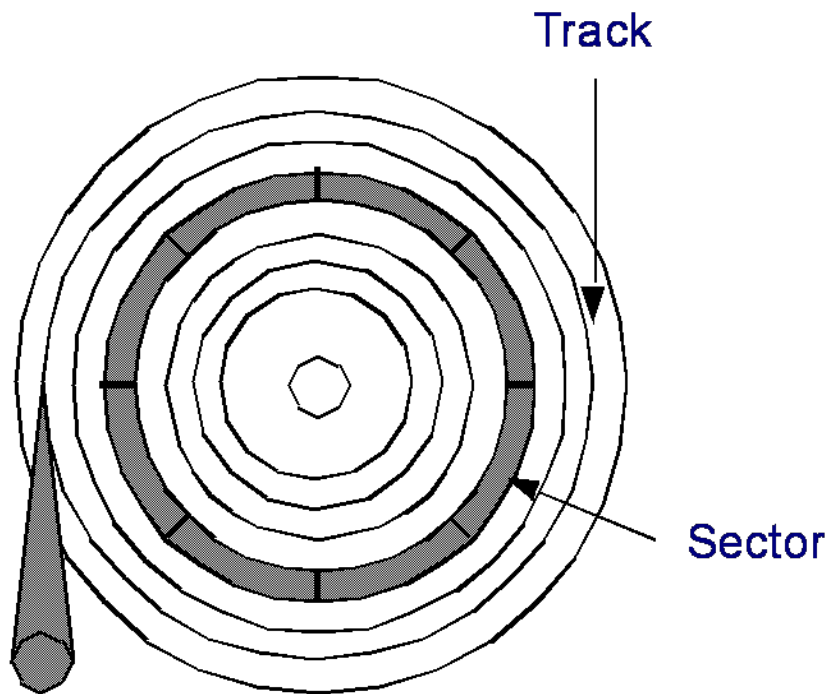
Class 16 - Secondary Storage Management

Jonathan Walpole
Computer Science
Portland State University

Disks

Disk geometry

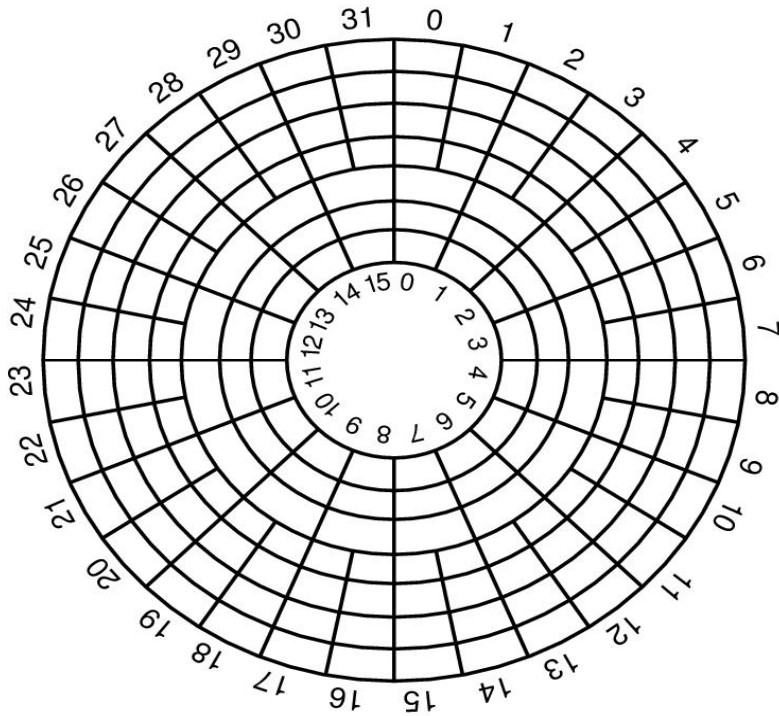
- Disk head, surfaces, tracks, sectors ...



Comparison of (old) disk technology

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μ sec

Disk zones



Constant rotation speed

- Want constant bit density

Inner tracks:

- Fewer sectors per track

Outer tracks:

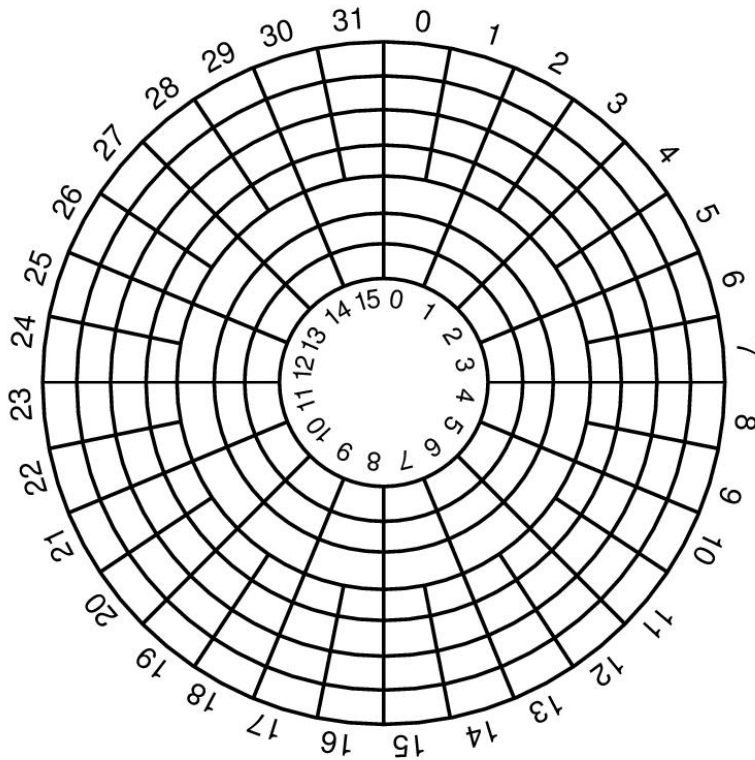
- More sectors per track

Disk geometry

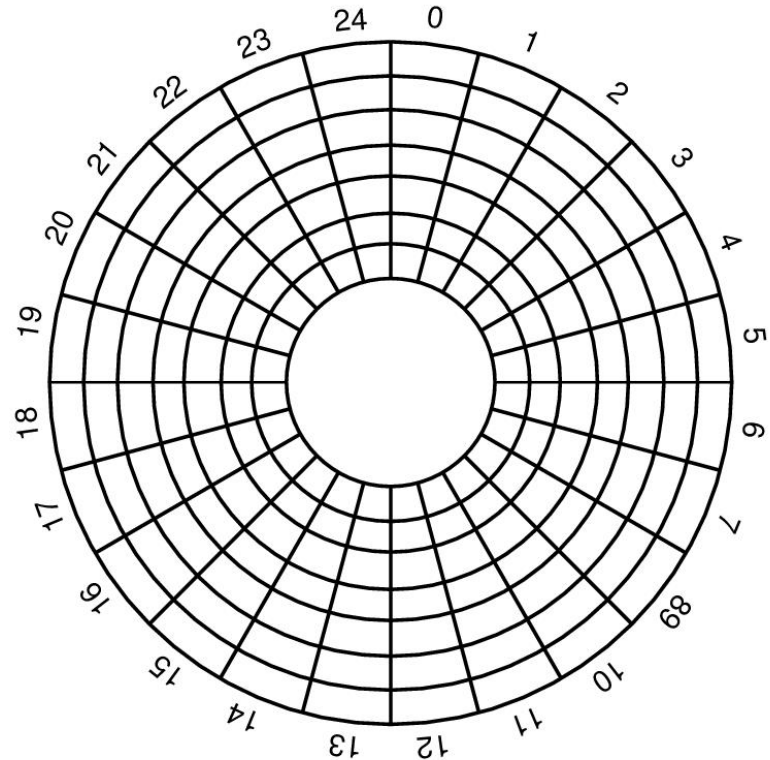
- **Physical Geometry**
 - ❖ The actual layout of sectors on the disk may be complicated
 - ❖ The disk controller does the translation
 - ❖ The CPU sees a “virtual geometry”.

Disk geometry

□



physical geometry



virtual geometry

(192 sectors in each view)

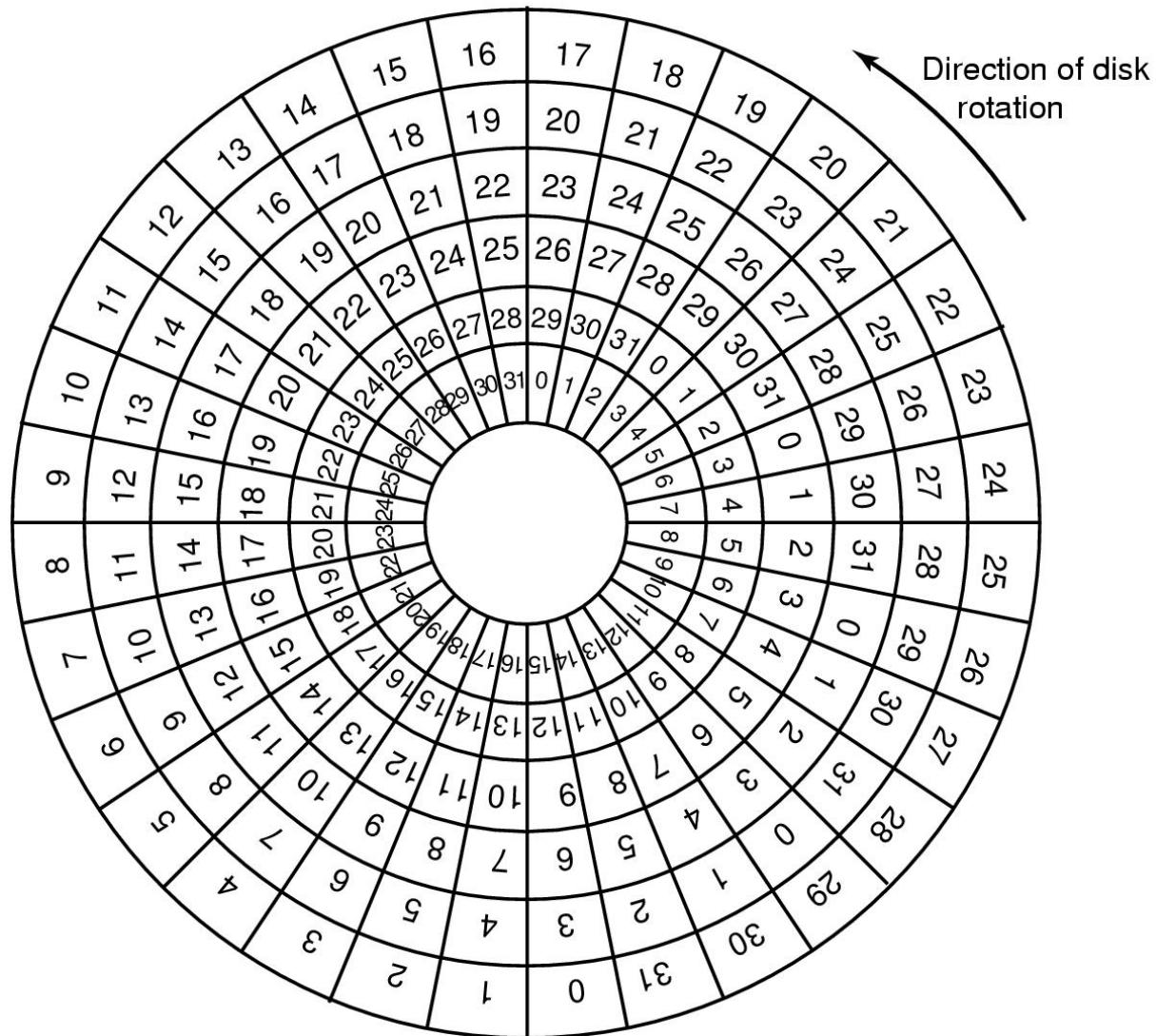
Disk formatting

- A disk sector

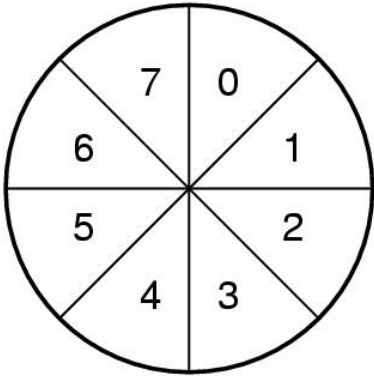


- Typically
 - ❖ 512 bytes / sector
 - ❖ ECC = 16 bytes

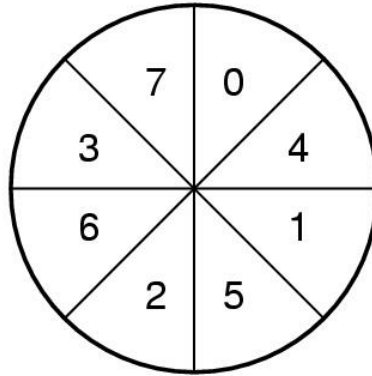
Cylinder skew



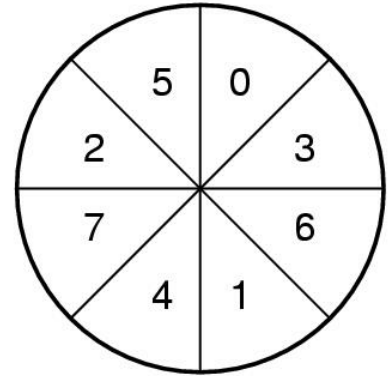
Sector interleaving



**No
Interleaving**



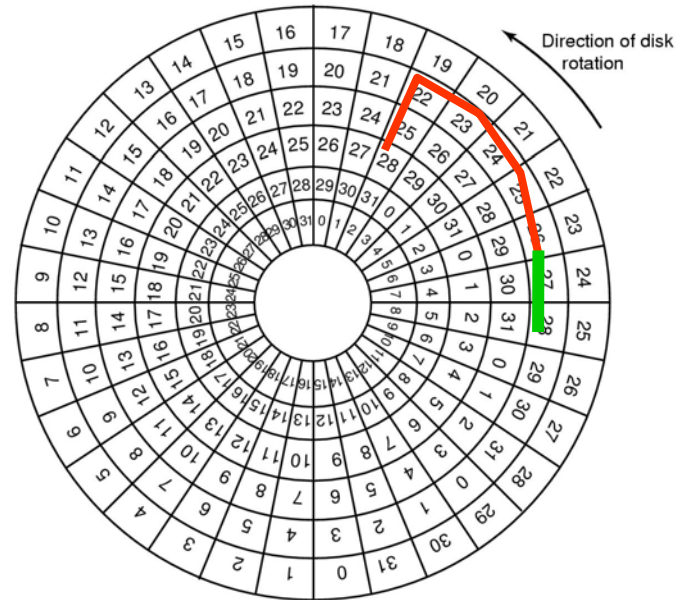
**Single
Interleaving**



**Double
Interleaving**

Disk scheduling algorithms

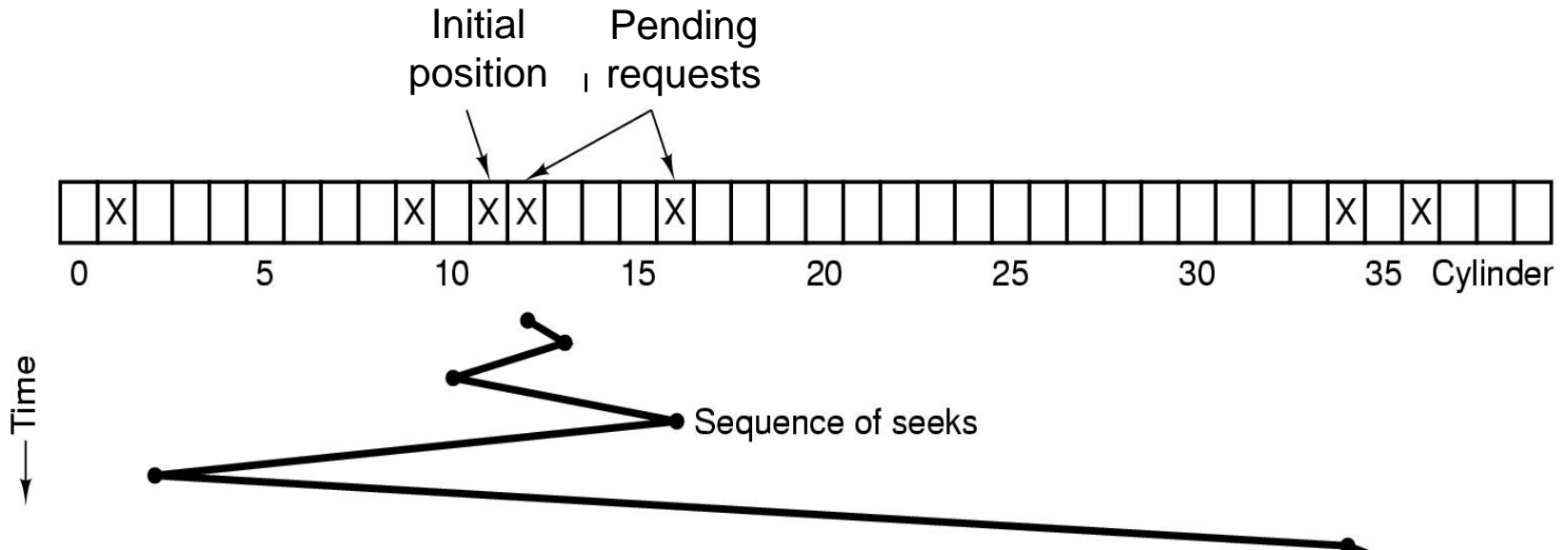
- Time required to read or write a disk block determined by 3 factors
 - ❖ Seek time
 - ❖ Rotational delay
 - ❖ Actual transfer time
- **Seek time dominates**
 - ❖ Schedule disk heads to minimize it



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

Shortest seek first (SSF)



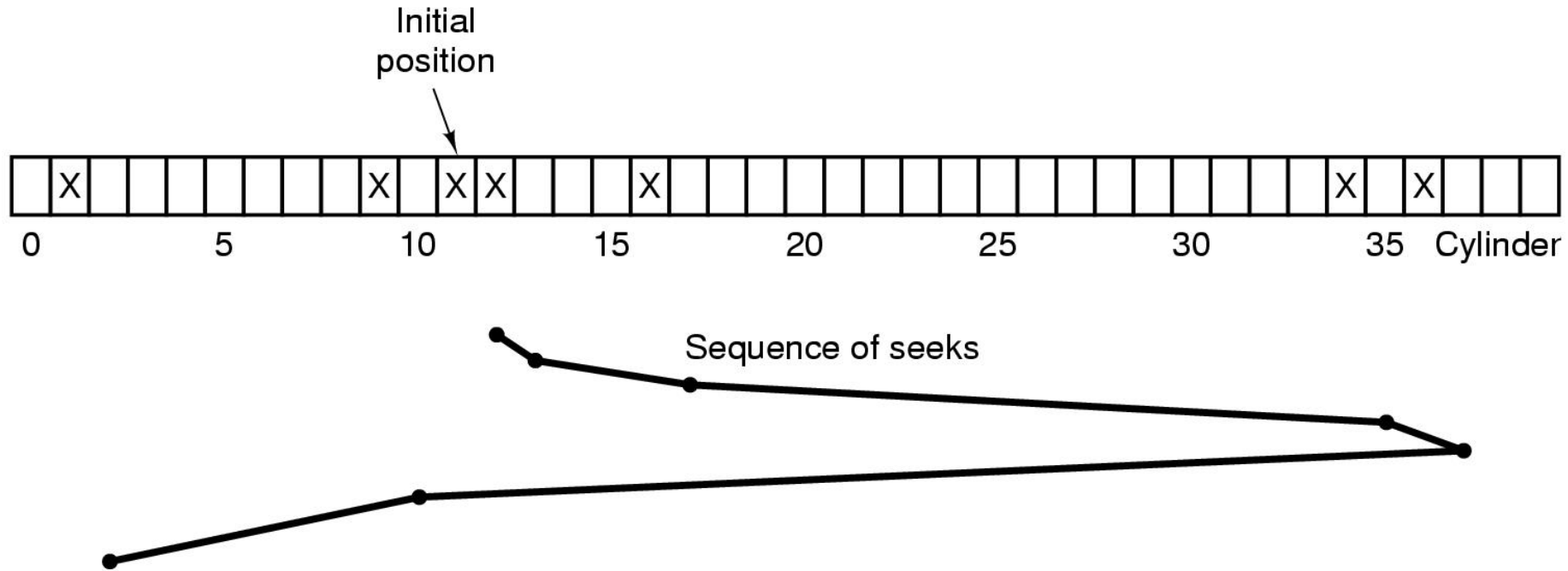
Shortest seek first (SSF)

- ❑ 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

The elevator algorithm



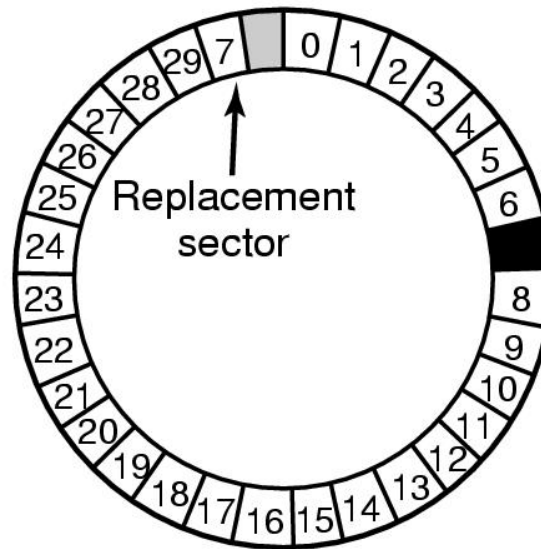
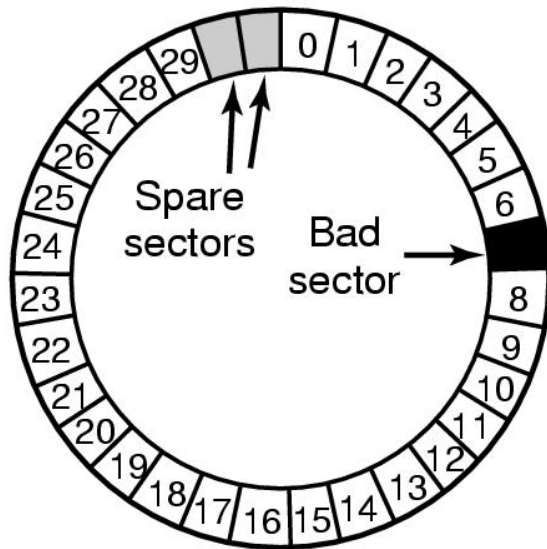
Other disk scheduling algorithms

- ❑ 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

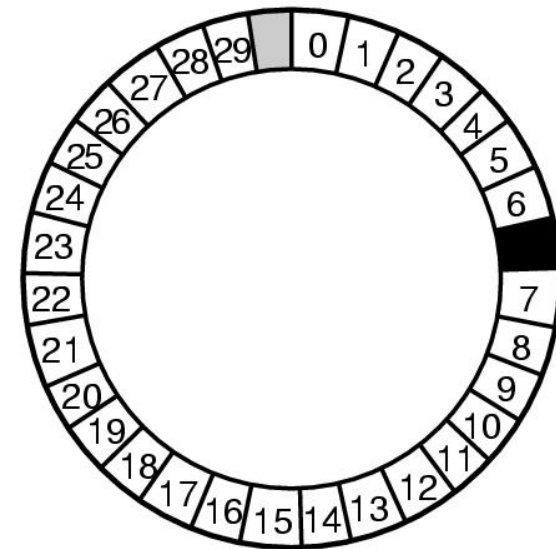
Errors on disks

- ❑ **Transient errors v. hard errors**
- ❑ **Manufacturing defects are unavoidable**
 - ❖ Some will be masked with the ECC (error correcting code) 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

Using spare sectors



**Substituting
a new sector**



**Shifting
sectors**

Handling bad sectors in the OS

- **Add all bad sectors to a special file**
 - ❖ The file is hidden; not in the file system
 - ❖ Users will never see the bad sectors
 - There is never an attempt to access the file
- **Backups**
 - ❖ Some backup programs copy entire tracks at a time
 - Efficient
 - ❖ Problem:
 - May try to copy every sector
 - Must be aware of bad sectors

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 probably be detected upon reading the block**
 - ❖ Disk blocks can go bad spontaneously
 - **But subsequent reads will detect the error**
 - ❖ CPU can fail (just stops)
 - **Disk writes in progress are detectable errors**
 - ❖ Highly unlikely to lose 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

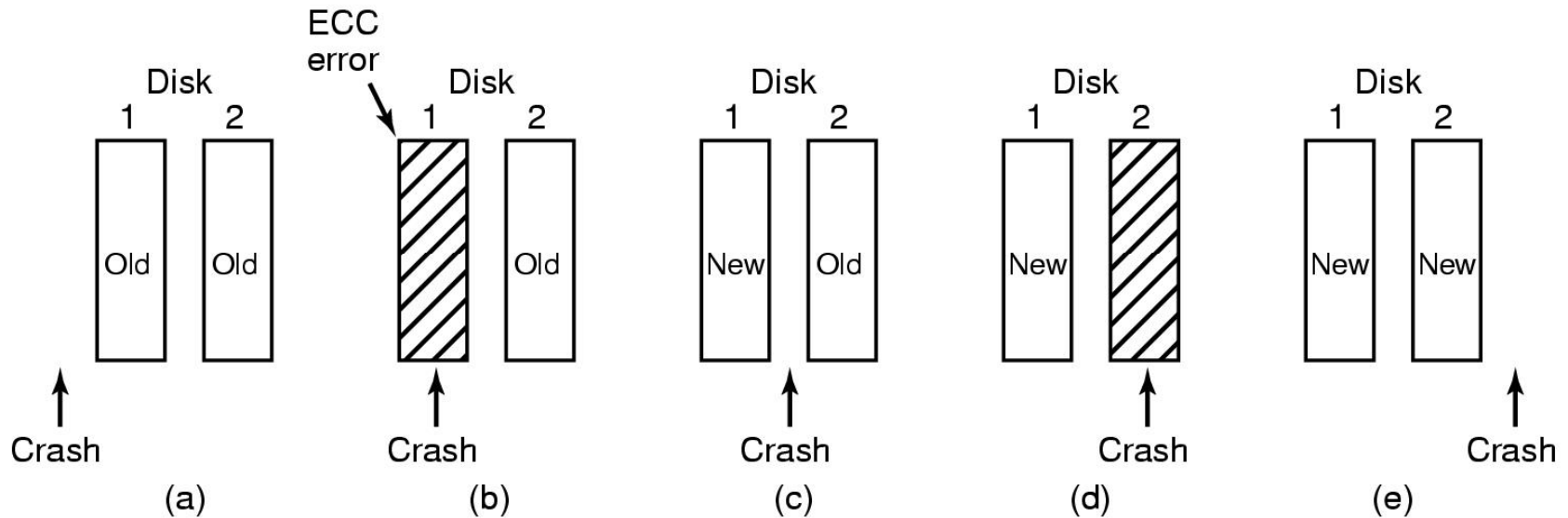
- Stable Read
 - ❖ Read the block from disk # 1
 - ❖ If problems...
 - Try again several times to get the block
 - ❖ If the block can not be read from disk #1...
 - Read the corresponding block from disk #2
 - ❖ Our Assumption:
 - The same block will not simultaneously go bad on both disks

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

□



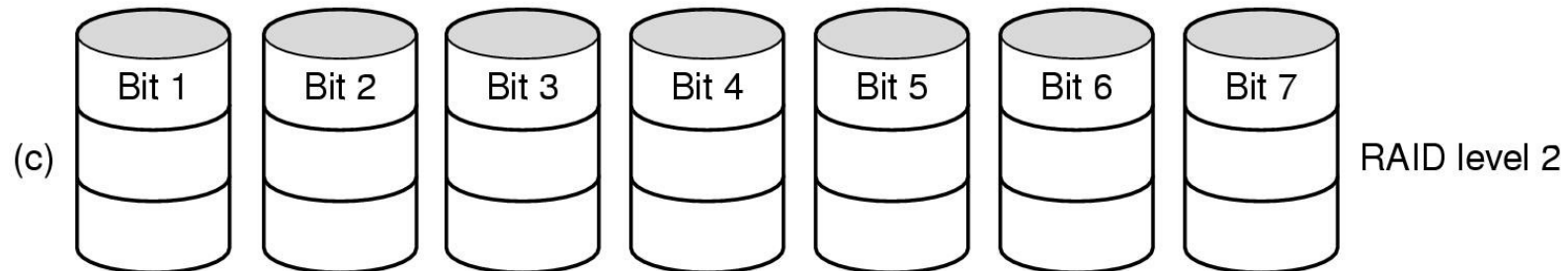
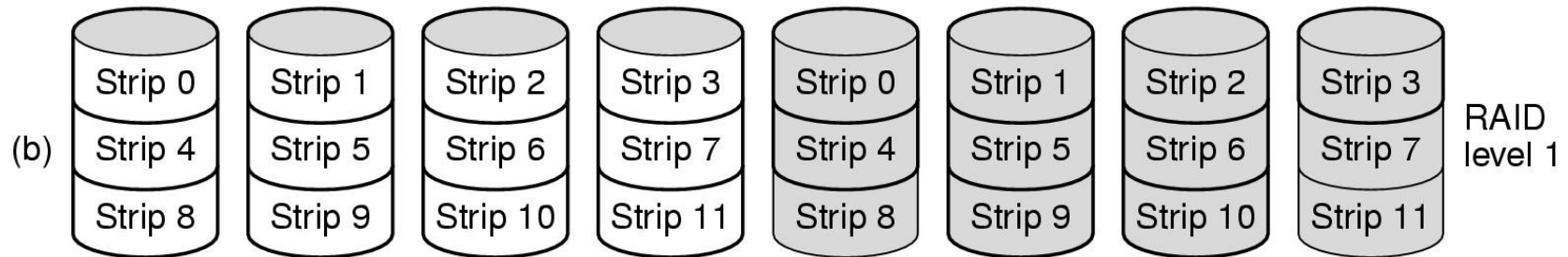
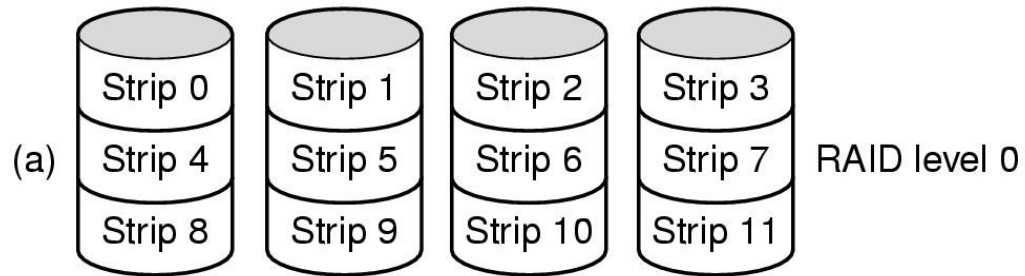
Stable storage

- ❑ 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
 - Can do better...

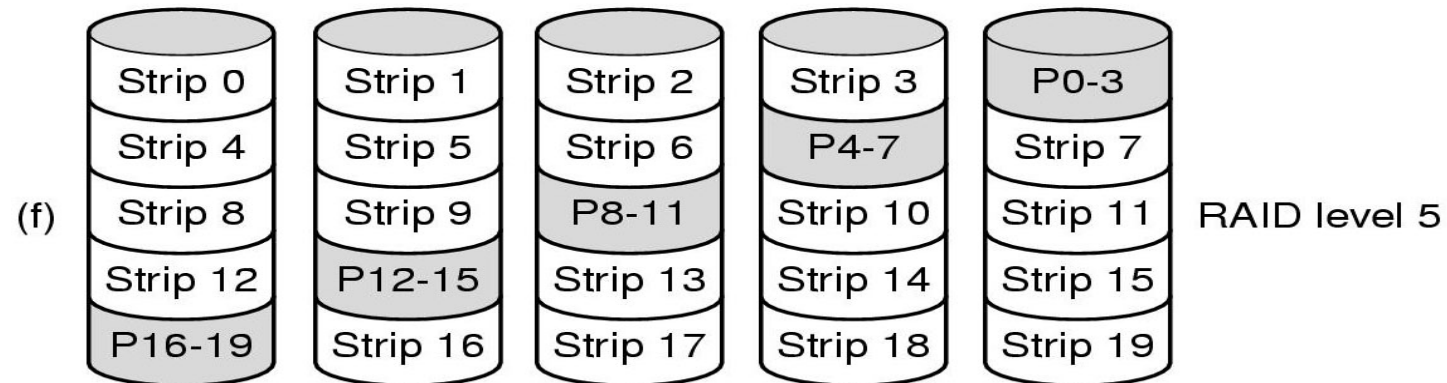
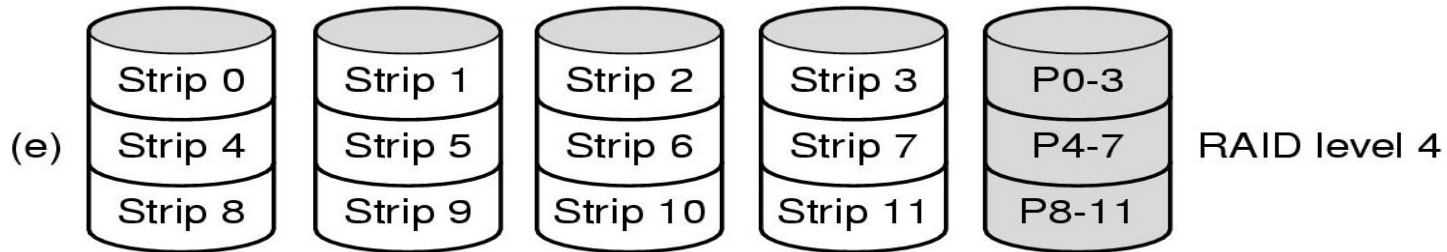
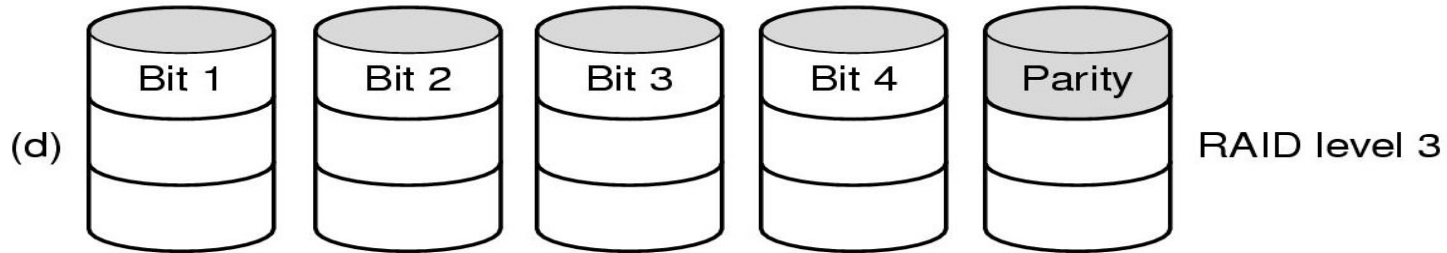
RAID

- ❑ Redundant Array of Independent Disks
- ❑ Redundant Array of Inexpensive Disks
- ❑ Goals:
 - ❖ Increased reliability
 - ❖ Increased performance

RAID



RAID



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

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

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

Disk space management

- ❑ **Must choose a disk block size...**
 - ❖ = Page Size?
 - ❖ = Sector Size?
 - ❖ = Track size?
- ❑ *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

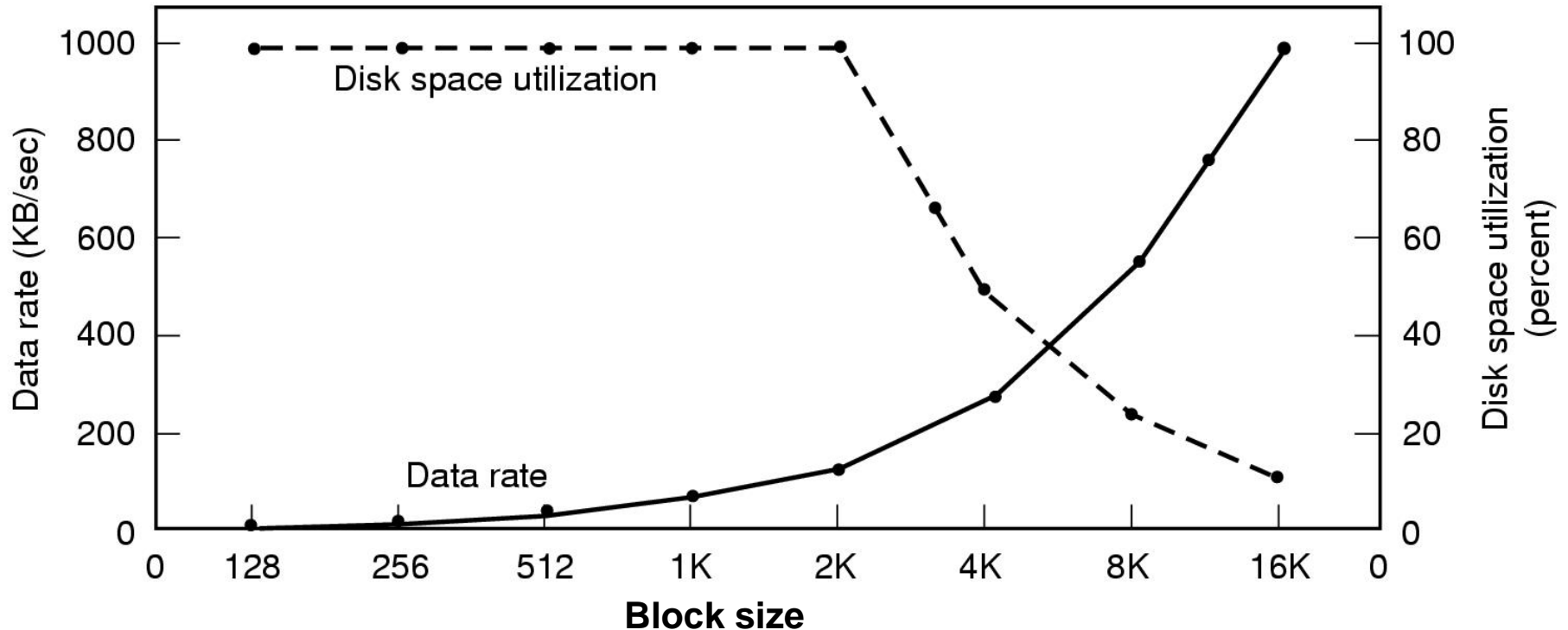
Example

- 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*

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*
- *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

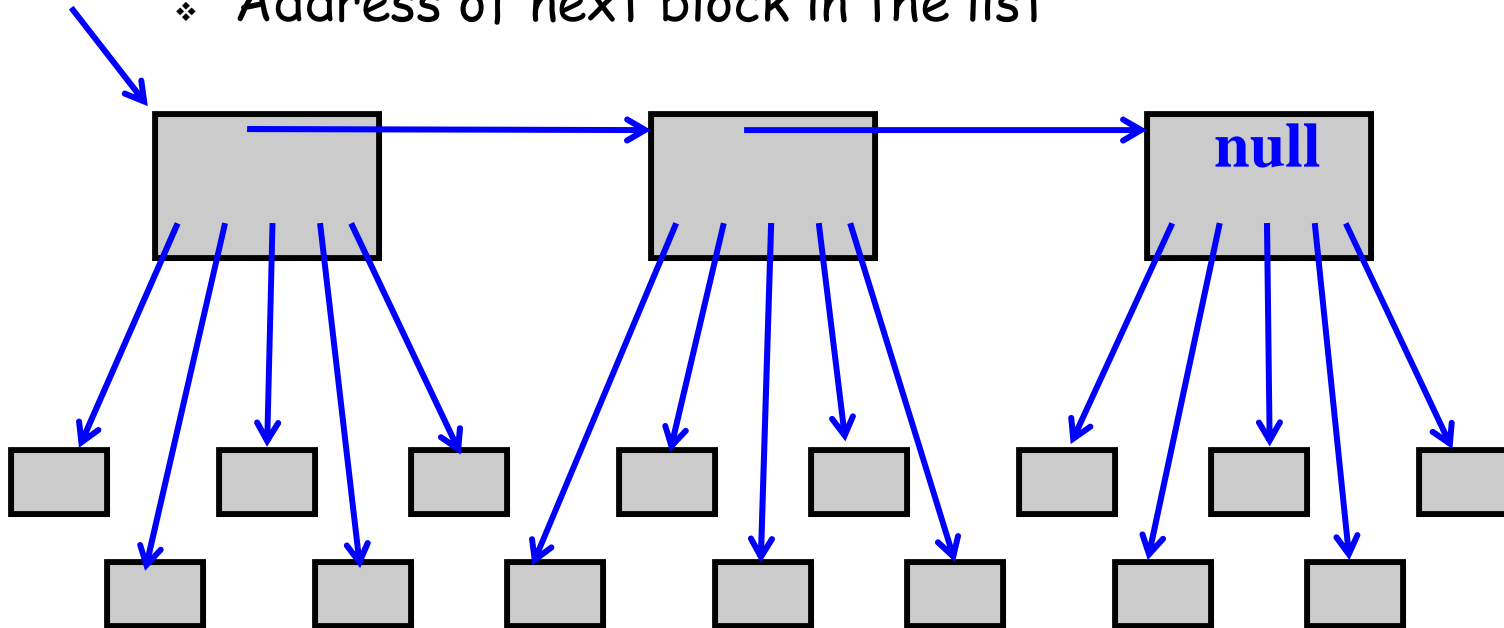
-
- Approach #2
 - ❖ Keep a free list

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

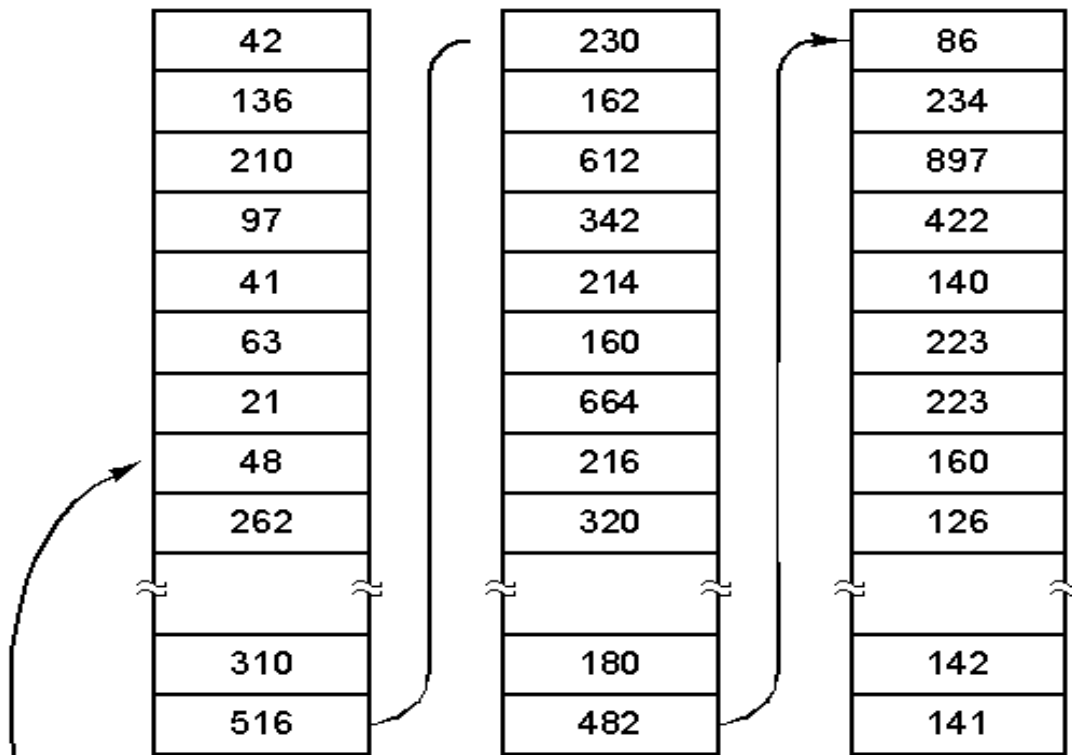
Free list of disk blocks

- ❑ 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

Free disk blocks: 16, 17, 18



A 1 KB disk block can hold 256
32-bit disk block numbers

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 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.

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.

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.

□

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.

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
- ❖ *Lots of disk I/O associated with free block mgmt!*

Free list of disk blocks

- *Solution (in text):*

- ❖ 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

- *Similar Algorithm:*

- ❖ 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.*

Comparison: free list vs bitmap

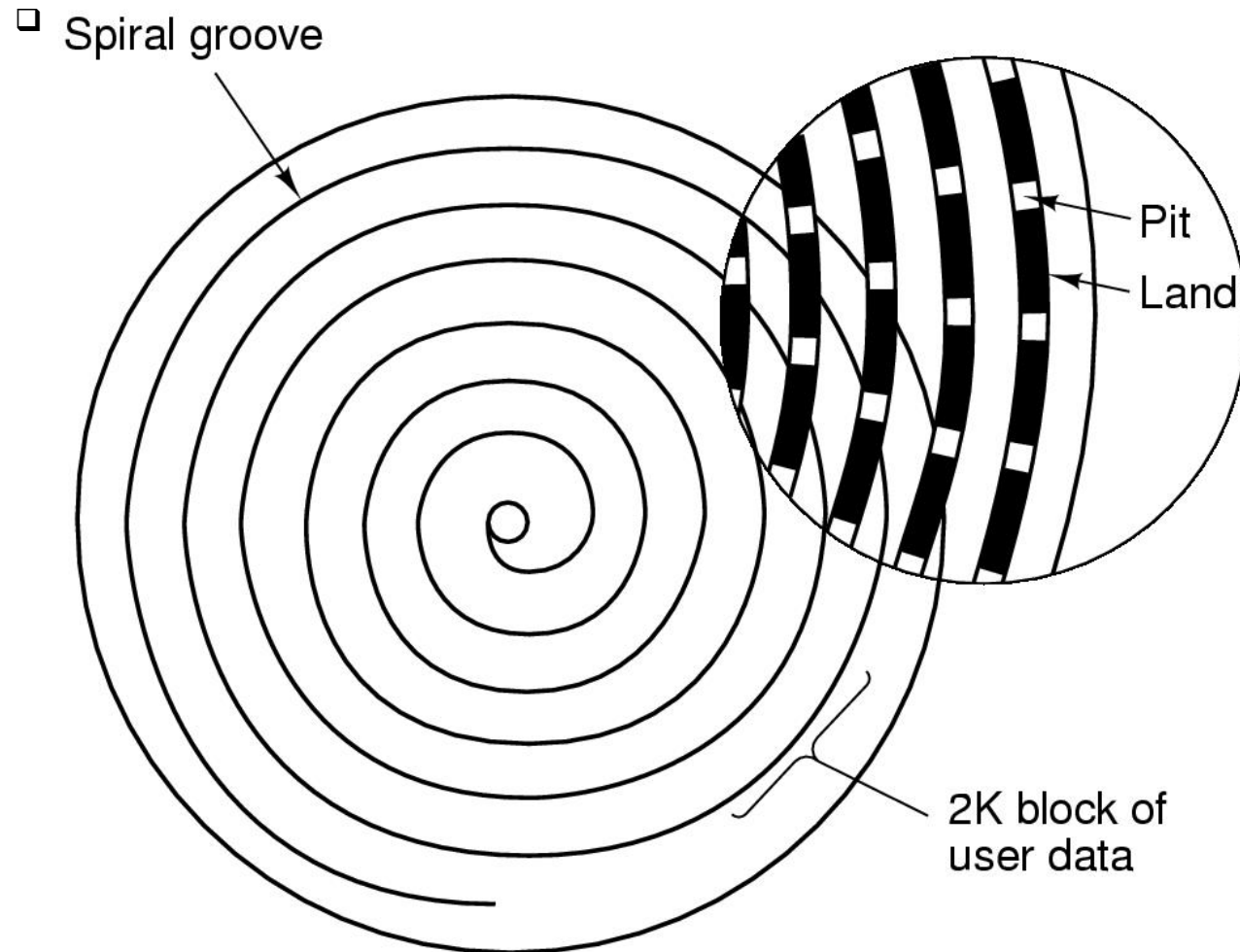
- Desirable:
 - ❖ *Keep all the blocks in one file close together.*
- Free Lists:
 - ❖ Free blocks are all over the disk.
 - ❖ Allocation comes from (almost) random location.

Comparison: free list v. 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

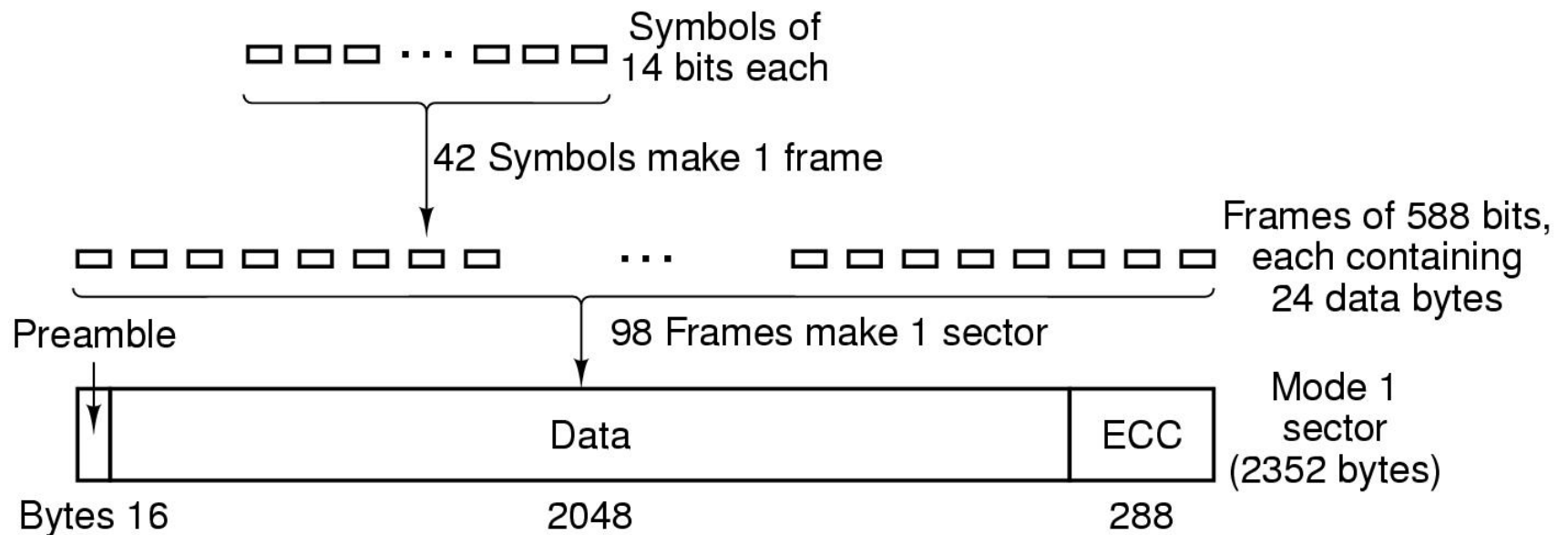
Spare slides

CDs & CD-ROMs

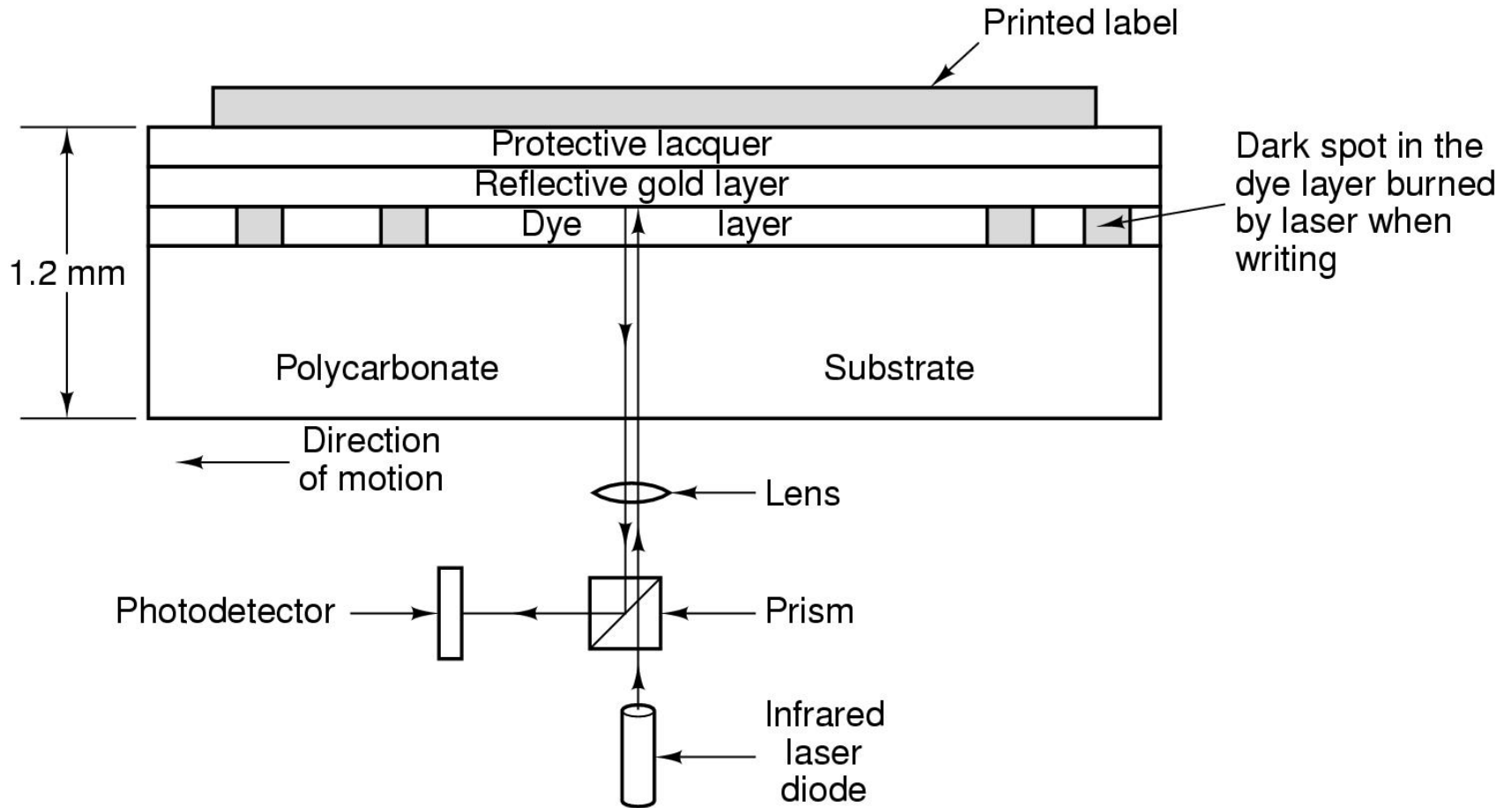


CD-ROMs

- ❑ 32x CD-ROM = 5,000,000 Bytes/Sec
- ❑ SCSI-2 is twice as fast.



CD-R (CD-Recordable)



Updating write-once media

- ❑ VTOC = Volume Table of Contents
- ❑ When writing, an entire track is written at once
- ❑ Each track has its own VTOC

Updating write-once media

- ❑ VTOC = Volume Table of Contents
- ❑ When writing, an entire track is written at once.
- ❑ Each track has its own VTOC.
- ❑ Upon inserting a CD-R,
 - ❖ Find the last track
 - ❖ Obtain the most recent VTOC
 - This can refer to data in earlier tracks
 - ❖ This tells which files are on the disk
 - ❖ Each VTOC supercedes the previous VTOC

Updating write-once media

- ❑ VTOC = Volume Table of Contents
- ❑ When writing, an entire track is written at once.
- ❑ Each track has its own VTOC.
- ❑ Upon inserting a CD-R,
 - ❖ Find the last track
 - ❖ Obtain the most recent VTOC
 - This can refer to data in earlier tracks
 - ❖ This tells which files are on the disk
 - ❖ Each VTOC supercedes the previous VTOC
- ❑ **Deleting files?**

CD-RW

- Uses a special alloy
- Alloy has two states, with different reflectivities
 - ❖ **Crystalline** (highly reflective) - Looks like "land"
 - ❖ **Amorphous** (low reflectivity) - Looks like a "pit"
- Laser has 3 powers
 - ❖ **Low power**: Sense the state without changing it
 - ❖ **High power**: Change to amorphous state
 - ❖ **Medium power**: Change to crystalline state

DVDs

- ❑ **“Digital Versatile Disk”**
 - ❖ Smaller Pits
 - ❖ Tighter Spiral
 - ❖ Laser with different frequency
- ❑ **Transfer speed**
 - ❖ 1X = 1.4MB/sec (about 10 times faster than CD)
- ❑ **Capacity**
 - ❖ 4.7 GB Single-sided, single-layer (7 times a CD-ROM)
 - ❖ 8.5 GB Single-sided, double-layer
 - ❖ 9.4 GB Double-sided, single-layer
 - ❖ 17 GB Double-sided, double-layer

DVDs

