CSE 380
Computer Operating Systems

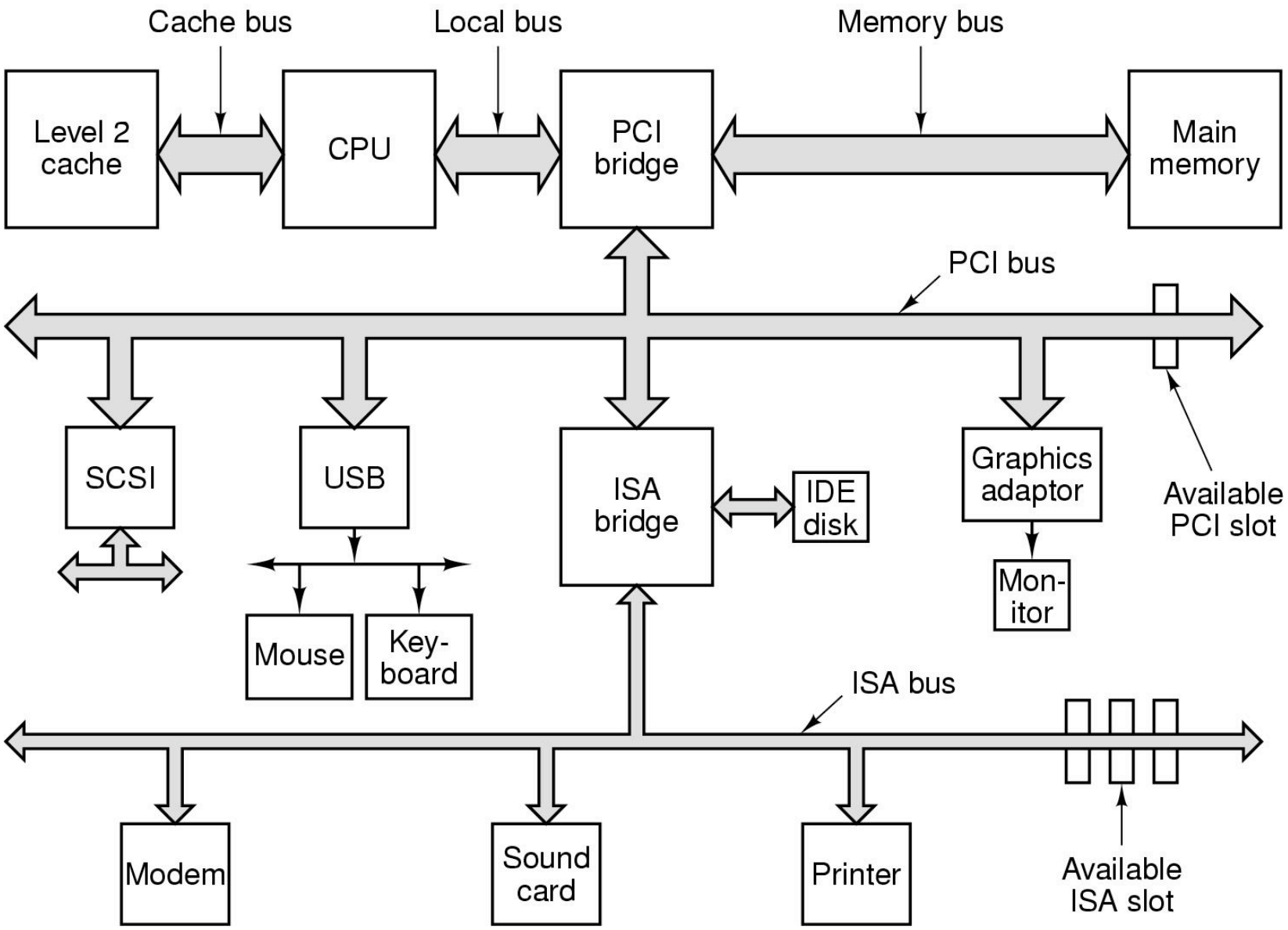
Instructor: Insup Lee

University of Pennsylvania
Fall 2003
Lecture Note on Disk I/O

I/O Devices

- Storage devices
  - Floppy, Magnetic disk, Magnetic tape, CD-ROM, DVD…
- User interaction
  - Keyboard, mouse, monitor, sound card, printer, modem ….
- Significant part of OS code deals with I/O devices
  - Communication with device drivers
  - Error handling
  - Interrupt management
  - Convenient interfaces

Pentium Bus Architecture



Device Controllers

- I/O devices have components:
  - mechanical component
  - electronic component
- The electronic component is the device controller
  - may be able to handle multiple devices
  - Should have a standardized interface
- Controller's tasks
  - convert serial bit stream to block of bytes
  - perform error correction as necessary
- Controller interface
  - Device registers
  - Data buffers for reading/writing data (separate from main memory, or not?)

Accessing the device

- Basically, two choices: I/O instructions that read/write I/O ports, or memory mapped I/O
- I/O instructions can be of the form
  - IN R, P R is a register, P is a port number
- In memory mapped I/O, no special instructions, but some addresses in virtual memory correspond to ports
  - MOV R, L L can be an address which is reserved for a specific device
- Pentium provides both schemes
  - I/O ports numbered 0 to 64K used for control information
  - Memory addresses from 64K to 1M used for device buffers

Memory Mapped I/O

- In an instruction such as MOV R, L, L can be
  - A location in main memory, or
  - A control register, or address in data buffer, of a specific I/O device
- I/O space viewed as an extension of main memory, and addressed in the same way
  - Of course, no overlap between the two address spaces
- Advantages
  - Device drivers are written in C
  - Protection managed by pages accessible in each user space
  - Savings in number of instructions
- Issues
  - Caching must be disabled for I/O pages (why?)
  - How to separate real memory references from I/O ones

I/O Programming

- Uniform naming of devices
  - In Unix, special directories such as /dev/lp, /dev/tty
- Device Independence
- Buffering of data
  - user-space or kernel-space or double buffering
- Driver routines
  - Busy waiting vs. Interrupt driven vs. DMA

I/O Code with Busy Waiting

- Goal: user wants to send N bytes from Buffer to an I/O device (e.g. printer or floppy)
- Device controller has registers for status and for putting data
- Basic approach (too wasteful for CPU):

```
Copy data into kernel buffer B;
for (i=0; i< N; i++) {
    /* wait for device to be free */
    while (*DeviceStatusReg != READY);
    *DeviceDataReg = B[i];
}
```

I/O Code with Interrupts

- Goal: To let CPU schedule other processes while device is busy processing data
- Code for driver:

```
Copy data into kernel buffer B
while (*DeviceStatusReg != ready);
*DeviceDataReg = B[0];    /* Do only the first step */
InvokeScheduler();
```

- Code for interrupt handler:

```
if (N == 0)  Wakeup(); /* unblock the process */
else {
        DeviceDataReg = B[i];
        N--; i++;
        };
ResetInterrupt(); /*Notify device as an ack */
```

I/O with Direct Memory Access

- Goal: Device should interrupt CPU only after the entire transfer
- Solution: Provide direct access between memory and device so that data can be copied into memory
- DMA Controller: separate unit that manages this
- Driver routine
  - Copy data into kernel space
  - Set up DMA controller registers
  - Schedule another process
- Interrupt handler
  - Ack the interrupt
  - Wakeup User

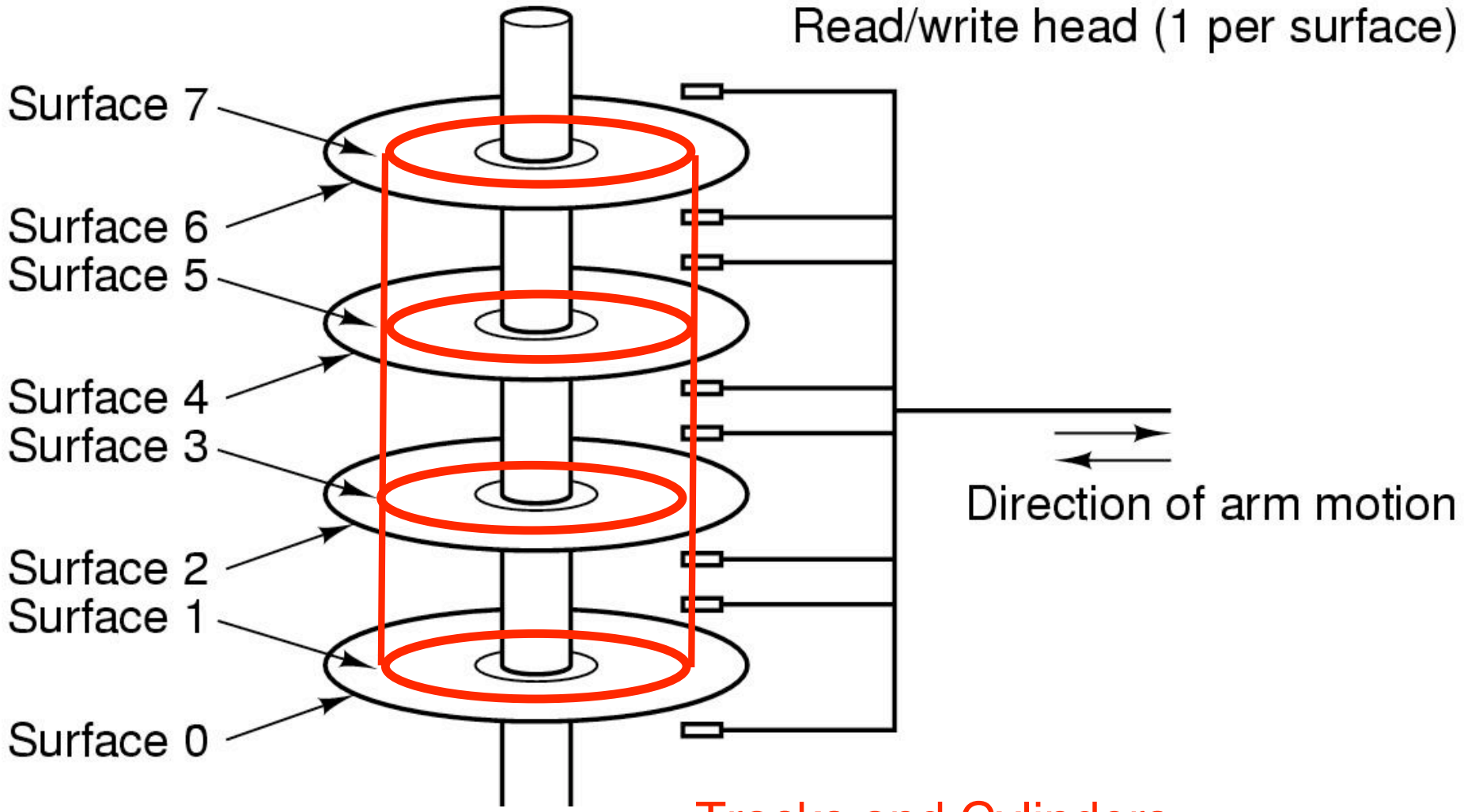
Mechanism of DMA Transfer

- A program running on CPU sets up the registers in DMA controller (e.g. write N bytes into memory starting at location XYZ)
  - Is the address XYZ virtual or physical ?
- DMA controller sets up the registers in device controller to request the transfer
  - Original request may be divided into chunks
- Disk controller transfers the data directly into memory
  - Bus architecture should resolve contention for memory
- Disk controller notifies DMA controller
- When the original request is entirely processed, DMA controller interrupts the CPU

Secondary Storage Management

- Why secondary storage?
  1. main memory too small
  2. main memory volatile
- Devices: Disks, Tapes, Drums
- Disks are critical components
  - For virtual memory management
  - Storing files
- Disk technology hasn't changed much compared to processor technology (esp., speed)

Disk Drives



Tracks and Cylinders

Typical Disk Parameters

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 µsec

Compare Seek time, rotation time, transfer time

Disk Formatting

- Typical sector is 512 bytes
  - Preamble identifying start code and sector address
  - Data
  - Error correction code (16 bits). At least detecting errors possible with probability almost 1

- Cylinder skew:

After reading an entire track, as head is moved across a cylinder, seeking next sector shouldn't cause waiting for full rotation

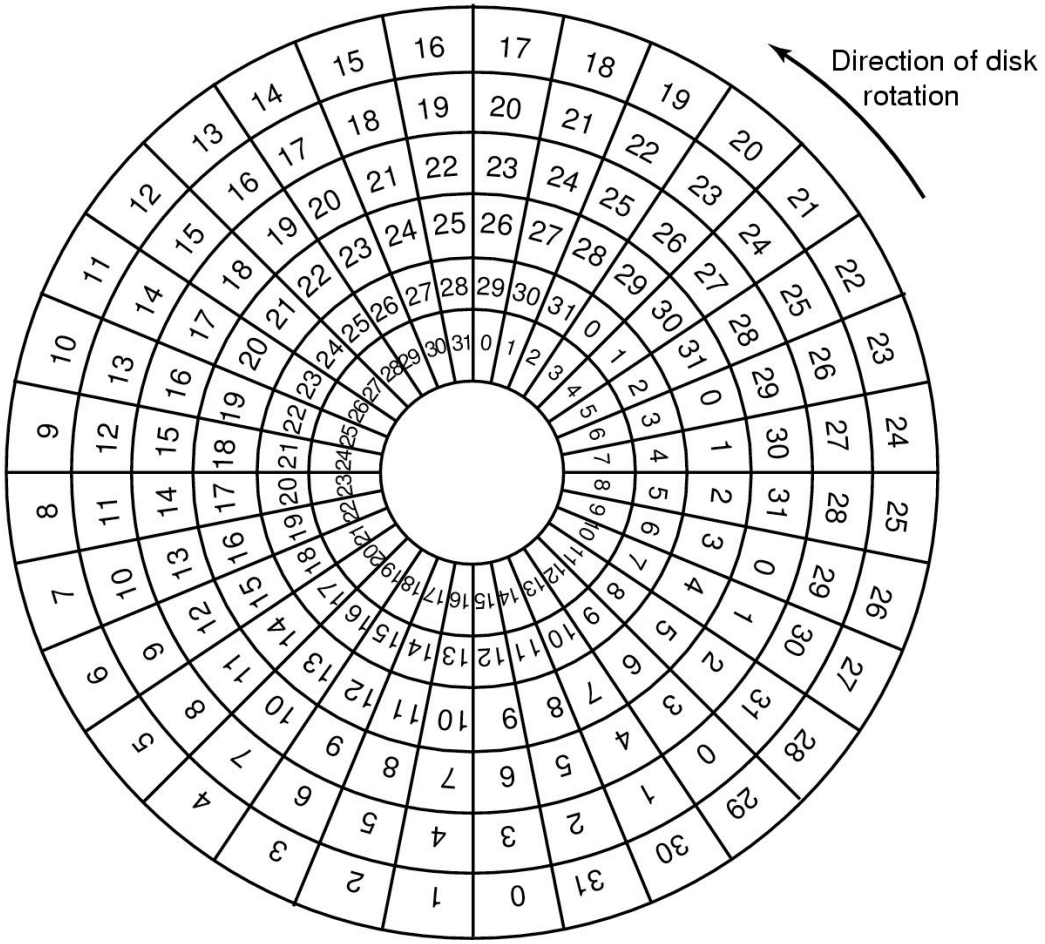
- Interleaving sectors:

As a sector is copied into controller buffer, it needs to be copied into main memory. So next sector should not be adjacent if we wish to avoid waiting for one full rotation

Disk Formatting (1)

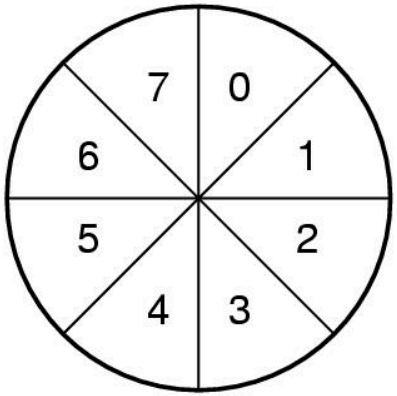
A disk sector

Disk Formatting (2)

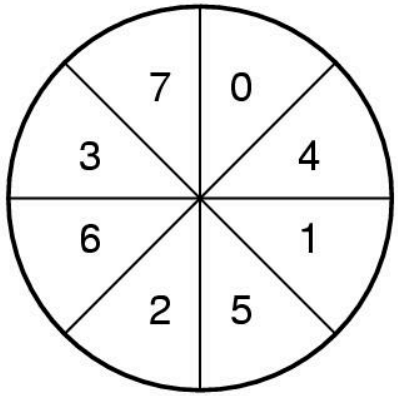


An illustration of cylinder skew

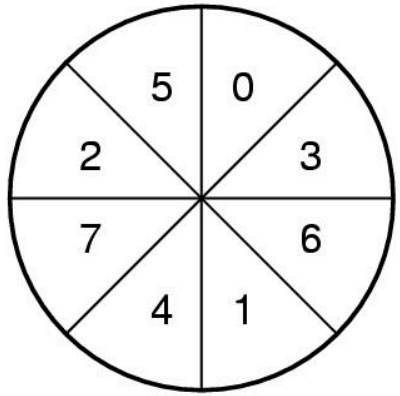
Disk Formatting (3)



(a)



(b)



(c)

- No interleaving
- Single interleaving
- Double interleaving

Disk Access

- Physical address on a disk

  (cylinder number, head number, sector number)

- Sectors can be given logical numbers that get decoded by Disk Controller

- Design considerations
  - Seek time (moving head to correct cylinder) and rotational latency time (waiting for correct sector) are greater than data transfer time (time to read)
  - Lots of errors possible (bad sectors, bad bits)
  - Caching blocks that happen to pass under disk head commonly used

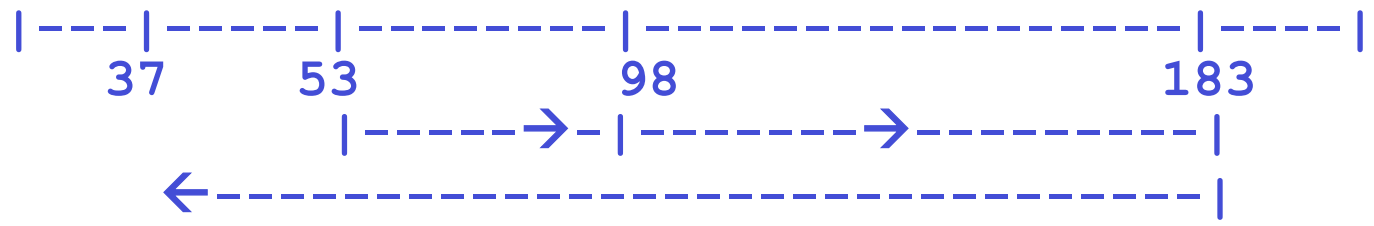
Disk-Head Scheduling

- Time required to read or write a disk block determined by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time
- Seek time dominates
- Error checking is done by controllers

Disk-Arm Scheduling: FCFS

Basic strategy for moving disk arm across cylinders: First-come-first-served queue

disk queue: 98, 183, 37, 122, 14, 124, 65, 67
initial r/w head position: cylinder 53



total head movement
= 45 + 75 + 146 + 85 + 108 + 110 + 59 + 2
= 640 tracks

OK for small load, low variance of waiting time

Disk-Arm Scheduling:SSF

Shortest Seek-Time First (SSF): Pick the closest one next

Consider example from last slide: ordering will be

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

total head movement (starting at 53)

= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236 tracks

High variance (starvation possible)

Disk-Arm Scheduling: Scan

Maintain a direction bit

Pick the request with shortest seek time in current direction first, and change direction only at edges.

Previous example:

Assume head was moving toward 0.
Ordering: 37, 14, 0, 65, 67, 98, 122, 124, 183
total head movement
= 16+23+14+65+2+31+24+2+59
= 236 tracks

Almost as good as SSF in mean wait time, lower variance.

Disk-Arm Scheduling Algorithms

- Look (elevator):
  - Variation of Scan algorithm
  - Change direction if no more requests farther down in current direction
  - Preferable to Scan (total head movement 208 in our case)
- Circular Look
  - Always scan in the same direction servicing cylinders in a circular manner (i.e. go to lowest numbered pending request if no requests are higher than current position)

Fault Tolerance for Disks

- Disks can have lots of errors
  - Corrupted bits
  - Corrupted sectors
  - Manufacturing defects
  - Disk crashes

- Many levels of protection implemented
  - Error correction within sectors
  - Extra sectors within a disk
  - Mirroring
  - RAID (redundancy and parallelism)

Error Correction Code in Sectors

- 16 bytes within 512 byte sector used for ECC
- Basic idea for k-bit ECC for n-bit data
  - Define a function f from n-bit integers to k-bit integers
  - n-bit data I is tagged with the ECC f(n)
  - If two n-bit data items I and J differ only in small number of bits then f(I) and f(J) are guaranteed to be different
  - To store a value I, write <I,f(I)>
  - While reading, read n+k bits, and check that last k-bits correspond to the ECC of first n-bits
- Simplest scheme: Parity
  - k=1, f(I)=1 if number of 1's in binary encoding of I is even
  - Error detection possible if just one bit changes, but can't correct

Hamming Codes (1950's)

- Let's see how 1 error can be corrected in n-bits of data by adding (log n) extra bits
- Suppose 11-bits of data D1 .. D11, we are going compute correction bits C1 .. C4
- Let's number the total 15 bits together so that correction bits occupy positions of powers of 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C1	C2	D1	C3	D2	D3	D4	C4	D5	D6	D7	D8	D9	D10	D11

- Consider a sequence of 11 bits, say, 10101000100, map it to above format, and find out which bit positions are 1's

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C1	C2	1	C3	0	1	0	C4	1	0	0	0	1	0	0

- Chosen positions are: 3, 6, 9, 13. Take their binary representations, and XOR all the words

  (0011) XOR (0110) XOR (1001) XOR (1101) = (0001)
- So the error correction bits are: C1 = 1, C2=0, C3=0, C4=0

Error Correction in Hamming Codes

- What happens if a bit, say D11, gets flipped in storage/transmission
- D11 is in position 15, so decoder will compute the correction code as (0001) XOR (1111) = (1110)
- Correction code in stored data = 0001; computed correction code = 1110.
  - Compute XOR of these two values; this gives 1111, which is the position number of the corrupted bit
- This correction scheme works for all bits (data as well as correction)
- Why does this work?
  - C4 gives number of 1's among bits D5 – D11
  - C3 gives number of 1's among bits D2 – D4 and D8 – D11
  - C2 gives number of 1's among bits D1, D3, D4, D6, D7, D10, D11
  - C1 gives number of 1's among bits D1, D2, D4, D5, D7, D9, D11
- If C4 is decoded incorrectly, decoder knows that error must be in either C4 or D5 – D11.
- Now if C3 is correct, then D8 – D11 must be correct, and error must be in C4 or D5—D7.
- What happens if C2 is incorrect? Wrong bit must be D7.

Error Correction Continued

- In general, in Hamming code, k-bits of error correction are used to augment $2^k - 1 - k$ bits of data giving a encoded string of $2^k - 1$ bits
- If there is a single error in storing or transmitting the encoded string, the decoder can detect the error, as well as correct it
- Reed-Solomon codes (1960's: Polynomial codes over certain finite fields)
  - Original codes can correct upto 2 errors in 251 bit/byte long messages by adding 5 extra bits/bytes
  - Works well for bursty errors
- Error correction technology is central all data transmission, data storage for disks, CDs etc

Bad Sectors

- Sectors can get defective for a variety of reasons
- Detected by the format command (in MS-DOS)
- Every track has extra (spare) sectors that are not part of logical address space
- Disk controller keeps track of bad sectors and does remapping of sector numbers
- Two choices for renumbering
  - Substitute a spare sector for defective
  - Renumber all by shifting
- Obvious tradeoff
  - Efficiency of reading consecutive sectors
  - Initial effort of rewriting preambles in each sector

Read Errors

- After reading a sector, what if ECC indicates an error?

1 ECC may allow error correction

2 Read again, and error may go away

3 If read error seems to happen repeatedly, then the sector can be marked as bad, and replaced by a spare sector

- Instead of the driver, OS may need to keep track of bad sectors

- Backups have to worry about bad sectors

Reliability by Redundancy

- Mirroring (or Shadowing): duplicate disks
- Write every block to both the disks
- Read can be performed from any of the two disks
  - This can be exploited to allow parallelism and speeding up of reads
  - Also for error correction: if first read fails, try the other
- Crash: If one disk crashes, then the other one can be used to restore data
- If failures are independent, this dramatically increases mean-time to failure
- This can be managed by device driver (if it can handle multiple disks) or by OS
- Duplicating a disk is expensive.

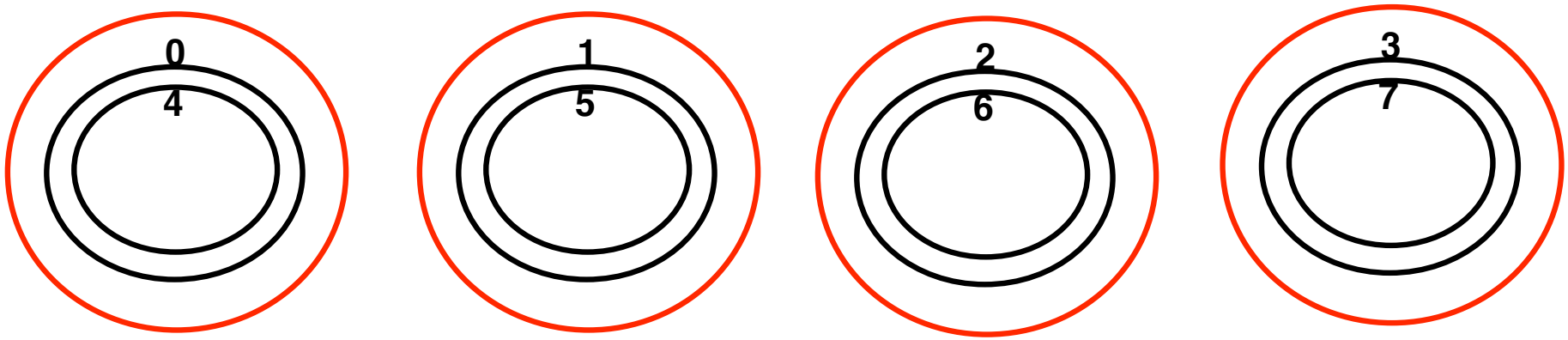
RAID
Redundant Array of Independent Disks

- Disk access times are still slow compared to CPU/memory
- Natural solution: Read concurrently from multiple disks
- RAID Controller talks to OS just like a device driver and manages a suite of disks to improve reliability and performance
- Striping:
  - Organize logically consecutive chunks of data across different disks to improve performance
  - Striping can be at bit-level, byte-level, or sector-level
  - Incoming request is split into all relevant disks by RAID controller, and their outputs are put together
- Redundancy through duplication
  - Multiple requests can be load-balanced
  - Error recovery on faults

RAID Level 0

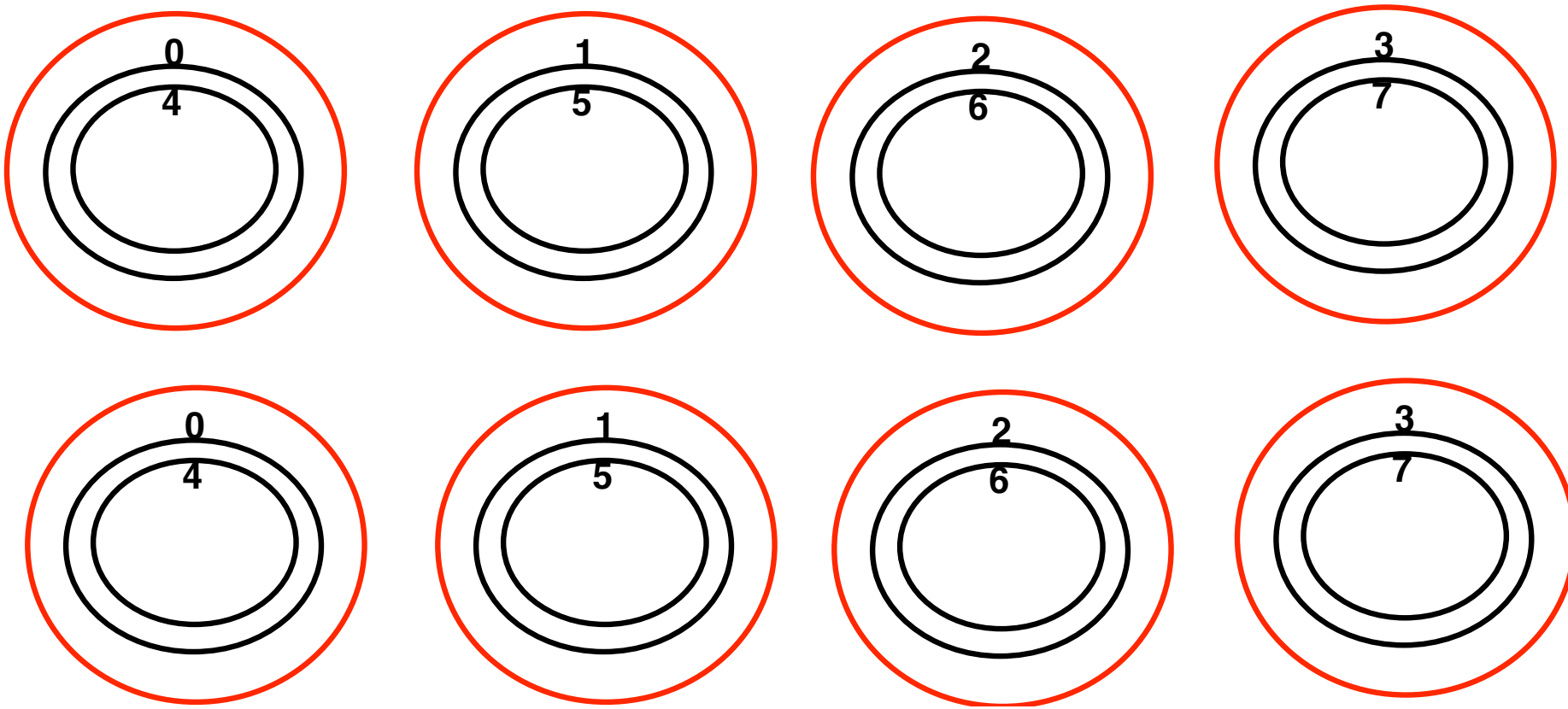
- Array of disks with block-level striping
- Strip = k sectors (for suitably chosen k)
- First k sectors are on first strip of first disk, next k sectors are on first strip of second disk, and so on
- Advantages:
  - A large request spanning multiple strips can be parallelized easily
  - Not much overhead
  - Used in high-performance applications
- Disadvantages
  - No redundancy
  - Mean-time to failure is, in fact, less than standard disks (why?)
  - No parallelism for small requests

RAID Level 0



RAID Level 1

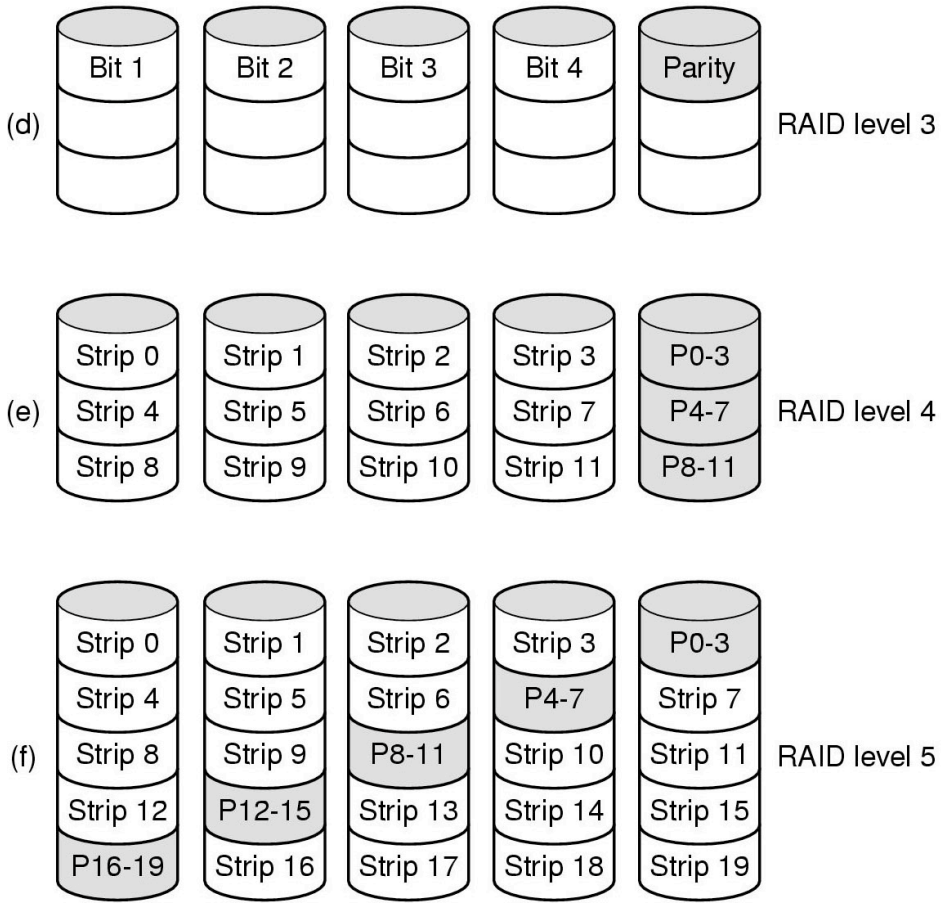
- Like mirroring, but with striping of level 0
- Use same number of backup disks with same striping
- Same benefits/issues as in mirroring



RAID Level 2

- Called memory-style error correcting code organization
- Uses bit-level striping
- Some of the bits used for parity or other error correction
- Sample: 4-bits of data encoded by additional 3-bits of ECC, and resulting 7 bits stored on 7 different disks
- CM-2 machine: 32-bits of data with 7-bits of ECC spanning over array of 39 disks
- Advantages
  - Great parallelism for read as well as write
  - Even if one disk crashes, all data can be recovered (due to ECC)
- Disadvantages
  - Drives have to synchronized
  - Overhead in putting together reads from different disks
  - Crash recovery much slower (compared to RAID-1)

RAID Levels 3-5



- Raid levels 3 through 5
- Backup and parity drives are shaded