Lecture 11

IO Devices and Mass Data Storage

Previously ..

- Resource allocation graph: Vertices (resource with instant or process) and Edges (request or assignment)
- Deadlock management: Prevention, Avoidance, and Detection with recovery.
  - Dining Philosophers
  - Banker's Algorithm and Avoidance Graph
- So far we have covered two of the main themes of this course – Virtualization and Concurrency

Persistence

- We now move on to the third of the main themes – Persistence

- Computers do two things
  - Processing
  - Input and Output (I/O)
- How should I/O be integrated into a system? – IO Device
- Persistence allows data to be stored safely over the long term, outliving the program that created it – Mass Data Storage

I/O DEVICES

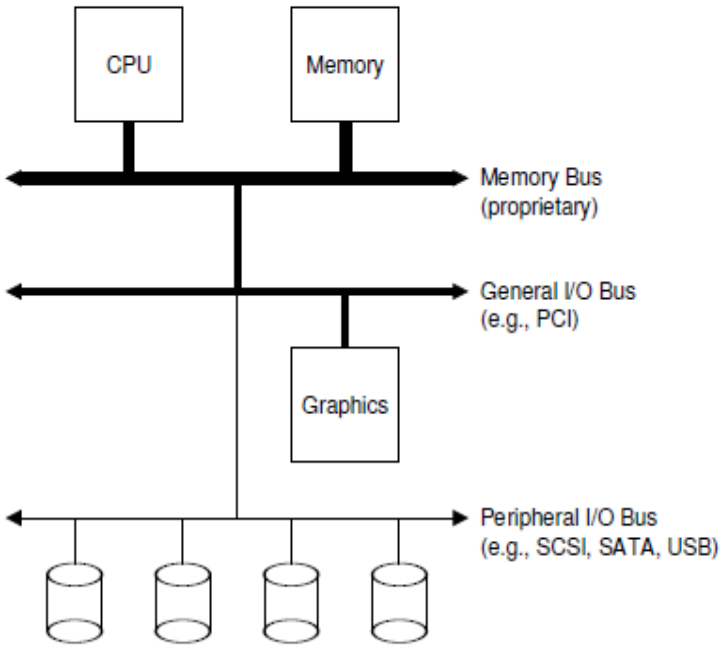
Introduction

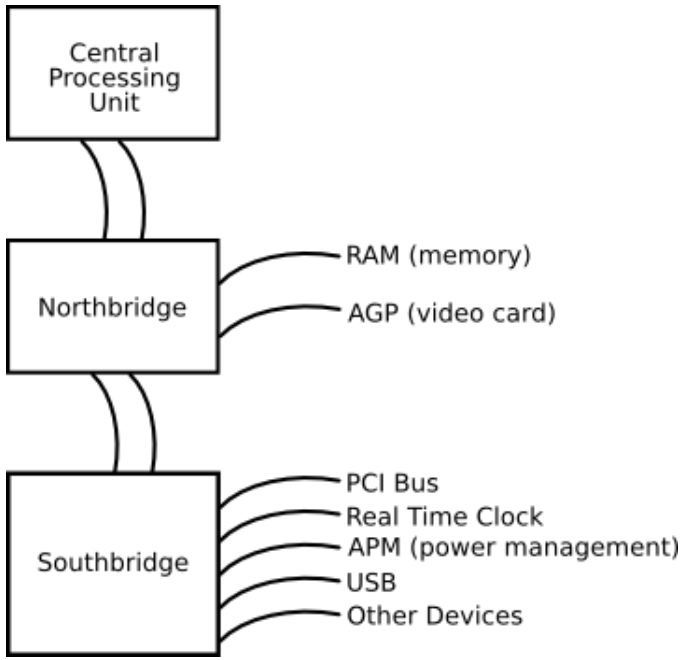
- For many jobs, the main task is I/O
- I/O devices vary widely in function and speed
- Various methods to control them
- New types of devices are often developed
- I/O management is a major part of OS design and operation
  - Manage and control I/O operations and devices
  - Use the hardware efficiently

Devices, Ports, and Buses

- Devices include storage, transmission, HCI devices, etc.
- Ports are the connection point between an I/O device and the machine, e.g. serial port or USB port
- Buses convey data between device and CPU, e.g. PCI bus or SCSI bus
  - Hierarchical Bus Structure – Devices that require high performance are nearer the CPU
    - Memory Bus for fast access to memory
    - General I/O Bus (e.g. PCI) for graphics and some other high-performance devices
    - Peripheral I/O Bus (e.g. SCSI) for slower devices such as disks

Hierarchical Bus Structure

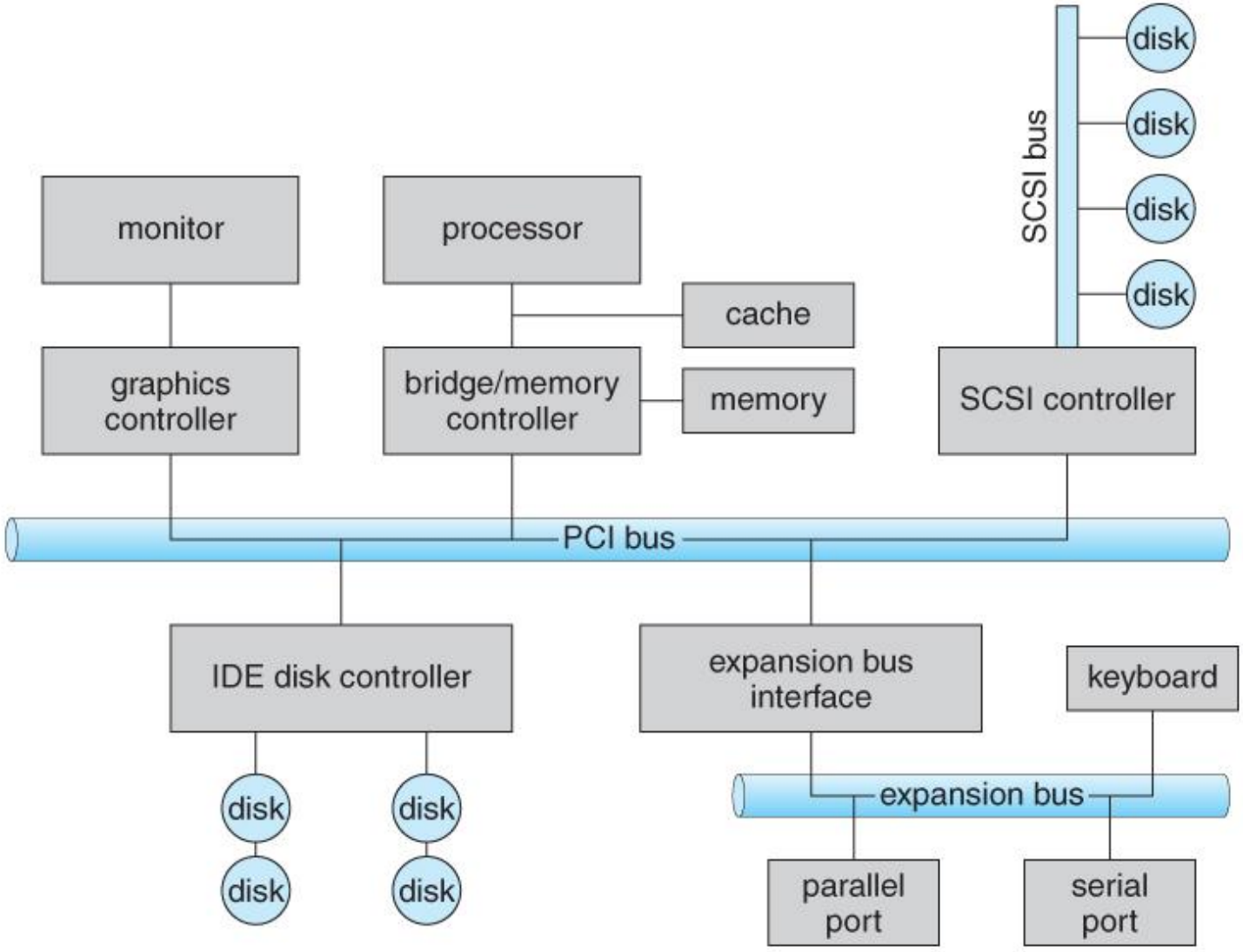




- ATA: Advanced Technology Attachment
- SATA: Serial ATA
- USB: Universal Serial Bus

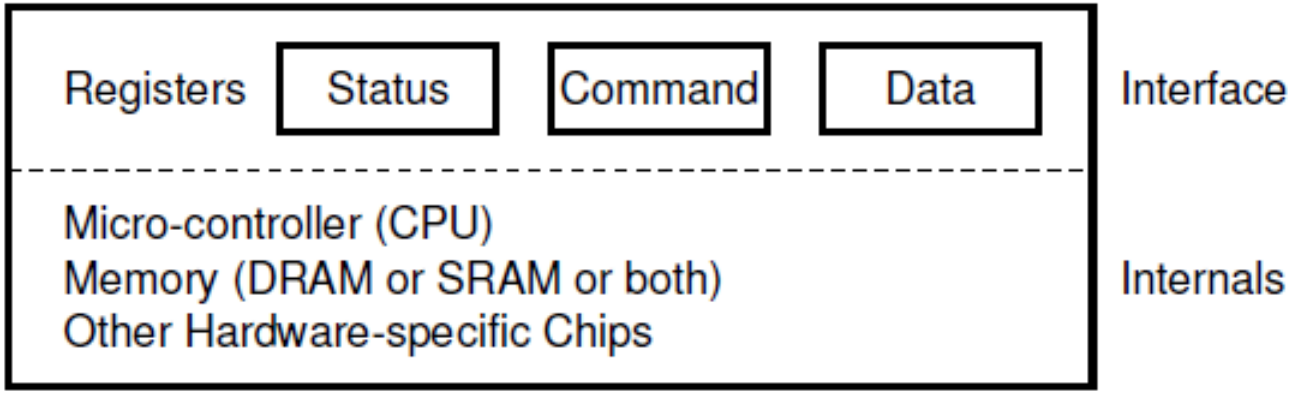
- ISA: Industry Standard Architecture (1981)
- PCI: Peripheral Component Interconnect (1993)
- AGP: Accelerated Graphics Port (1996)

Typical PCI Bus Structure

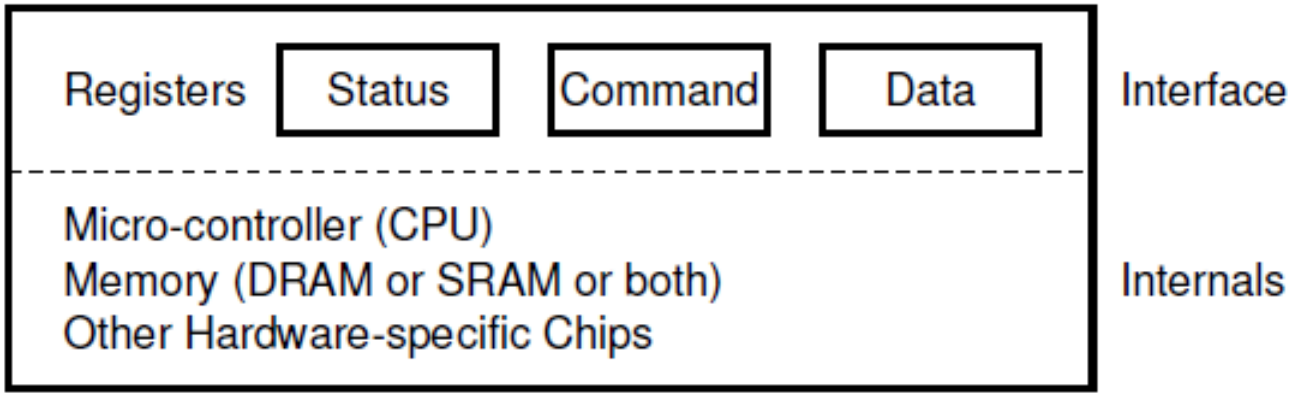


Typical Device

- A Device has
  - An Interface it presents to the system
    - Registers that the OS can use to control the device
  - An Internal Structure that is device specific
    - Simple devices may have just a few chips
    - More complex devices may include a micro-controller CPU, memory and other device specific chips

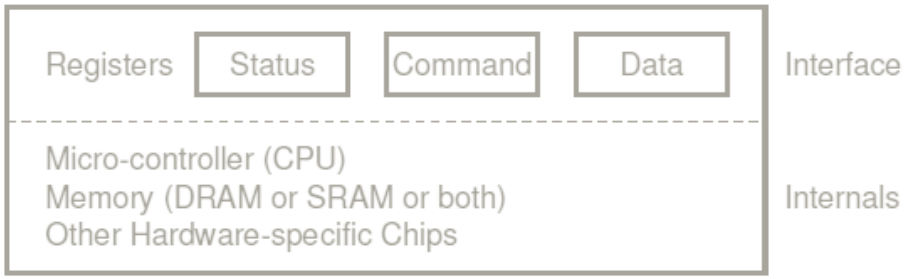


Device Interface



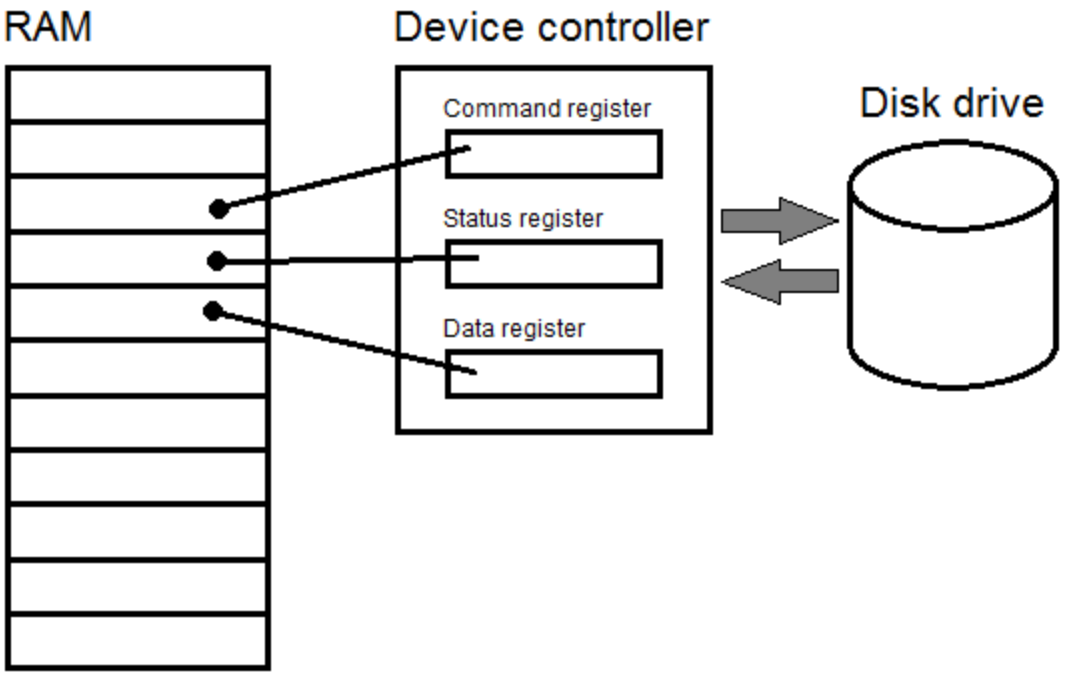
- Device Registers: Status, Command, and Data
  - Status register contains information that can be read by the kernel, e.g. if device busy, command complete, device error
  - Command register can be written by the kernel
  - Data register(s) can be used for passing data in and getting data out

Device Interaction



- There are two techniques that a processor can use to command and transfer data to a device
  - Use special I/O instructions for reading and writing the device registers, called port-based I/O
  - Use Memory-Mapped I/O
    - The device-registers are mapped into the memory space of the kernel
    - The kernel uses standard store and load instructions
  - Some kernels use both techniques

Memory-Mapped I/O



Device I/O Locations

- Example I/O Locations on PC

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Typical I/O Protocol

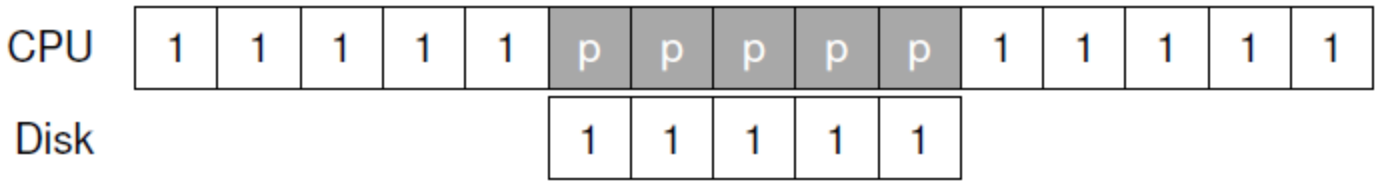
- OS executes four simple steps
  1) Wait until device is not busy (test Status Register)
  2) Move data (to/from Data Register)
  3) Command the device (write command to Command Register)
  4) Wait until device done

Polling Technique

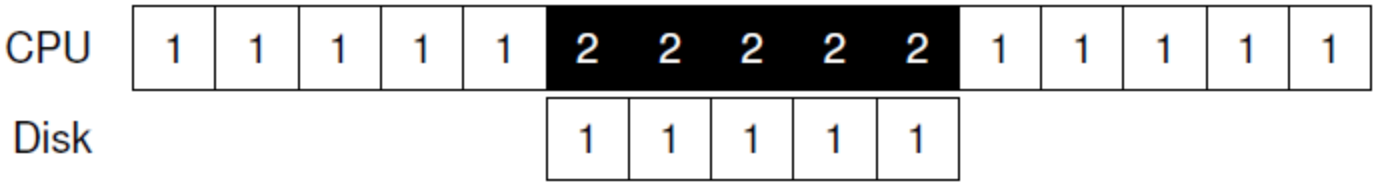
- The I/O protocol described previously uses a polling technique
- In both waiting steps (1 and 4) the kernel is repeatedly reading a status register
- Inefficient and Inconvenient
  - It wastes CPU time just waiting for the (potentially slow) device to complete its activity
  - Why not put the calling process to sleep and switch to another ready process?
  - In this way, we can better utilize the CPU
- This uses a technique called Interrupt

Interrupt Technique

- Without Interrupts



  - While OS is polling for Process 1, no process is doing useful work
- With Interrupts



  - Process 2 is executing while the disk transfers data for Process 1

Interrupt Technique

- Basic Concept
  1. The OS issues a device command on behalf of the calling process
  2. Put the calling process to sleep, and switch to another process
  3. When the device has completed the operation, it will raise a hardware interrupt, causing the CPU to jump into the OS at a pre-determined Interrupt Service Routine (ISR) or more simply an Interrupt Handler
  4. The ISR finishes the operation (e.g. by reading data) and wakes the process waiting for the I/O

Interrupt Controller

- Interrupt Vector Table
  - Contains the vector to ISRs
  - The table is written at boot time or when the device is ready
- Interrupt Priority Levels
  - Allow high priority interrupt to preempt a low priority interrupt
- Maskable Interrupts
  - The kernel can choose to ignore maskable interrupts
  - Non-maskable interrupts cannot be ignored
- Interrupts can also be used for exceptions (e.g. page faults) or software traps (system calls)

Interrupt Vector of Intel Pentium Processor

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Example of Interrupt for VM Paging

- Interrupt raised when page fault occurs (vector 14 in the Intel example)
- ISR is called, puts the process that caused the fault to sleep and executes the page management software
- Context switch another process to execute
- Device Interrupt when the needed frame is completely copied from disk
- ISR for disk is called, completes the operation and wakes the interrupted process

Polling vs. Interrupt

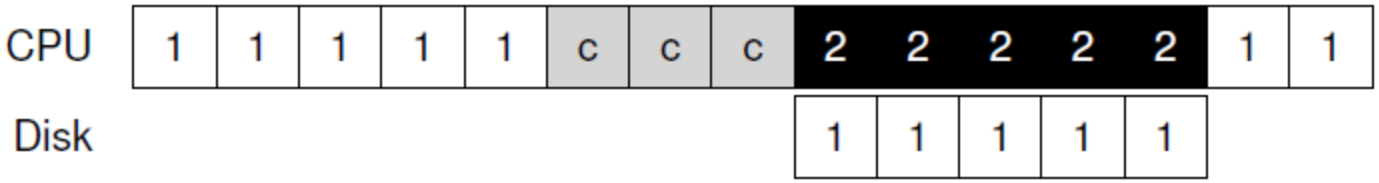
- Interrupts allow for overlap of computation and I/O
- However, for fast devices, the cost of interrupt handling and context switching may be larger than the benefits interrupts provide
- Also a flood of interrupts may overload a system and lead it to “livelock”
  - In networks, a large stream of incoming packets may each generate an interrupt
  - The CPU is so busy handling interrupts that no process can execute

Direct Memory Access (DMA)

- With Programmed I/O as previously described, the CPU writes the data to the device
- Many I/O operations require a lot of data movement from/to main memory
- DMA is a hardware module that it helps the CPU with data movement between main memory and I/O devices, like hard disks
- Bypasses CPU then transfer data directly between memory and I/O device

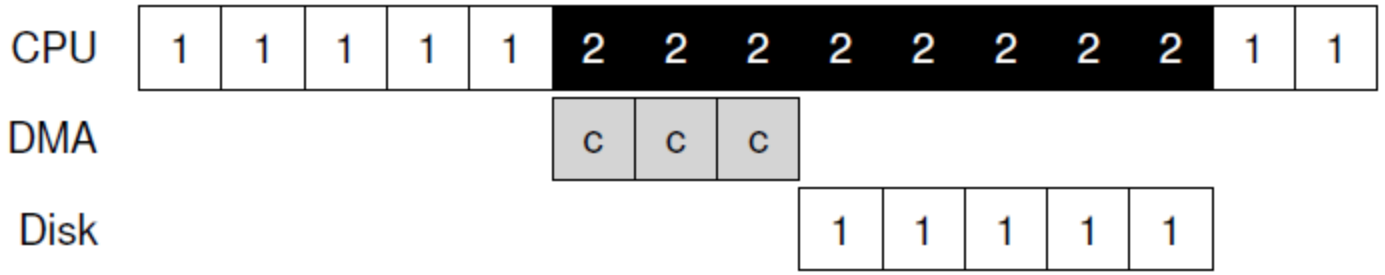
Direct Memory Access

- Without DMA



  - While OS copies data for Process 1, no process is doing useful work

- With DMA

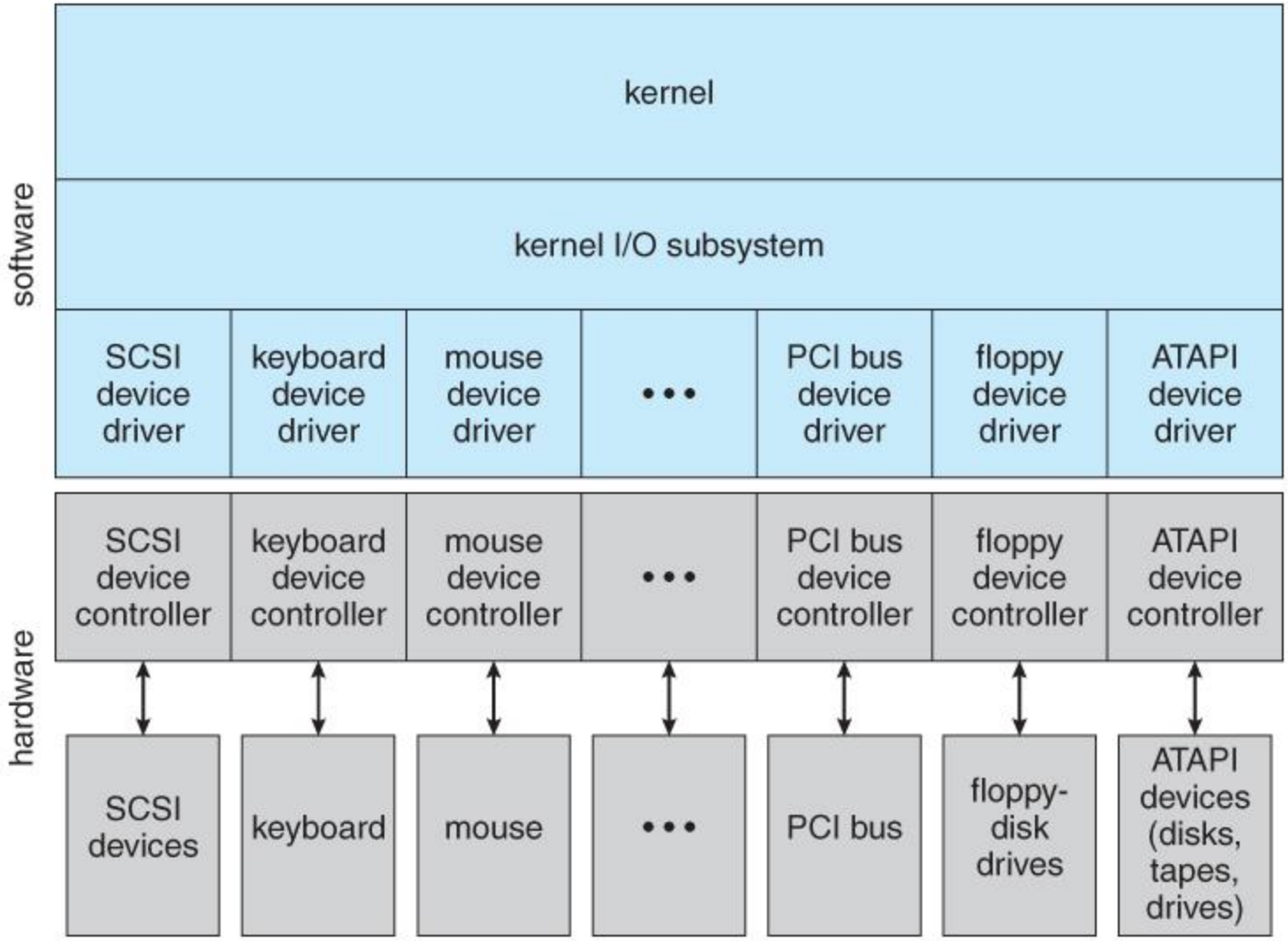


  - Process 2 is executing while the DMA copies data for Process 1

DMA Controller

- To transfer data to the device, the OS sends the command to DMA controller for initiating the data movement (e.g. source and destination address of data and the size)
- The OS is then finished with the transfer and another process may be executed
- When the DMA is complete, the DMA controller raises an interrupt, and the OS thus knows the transfer is complete

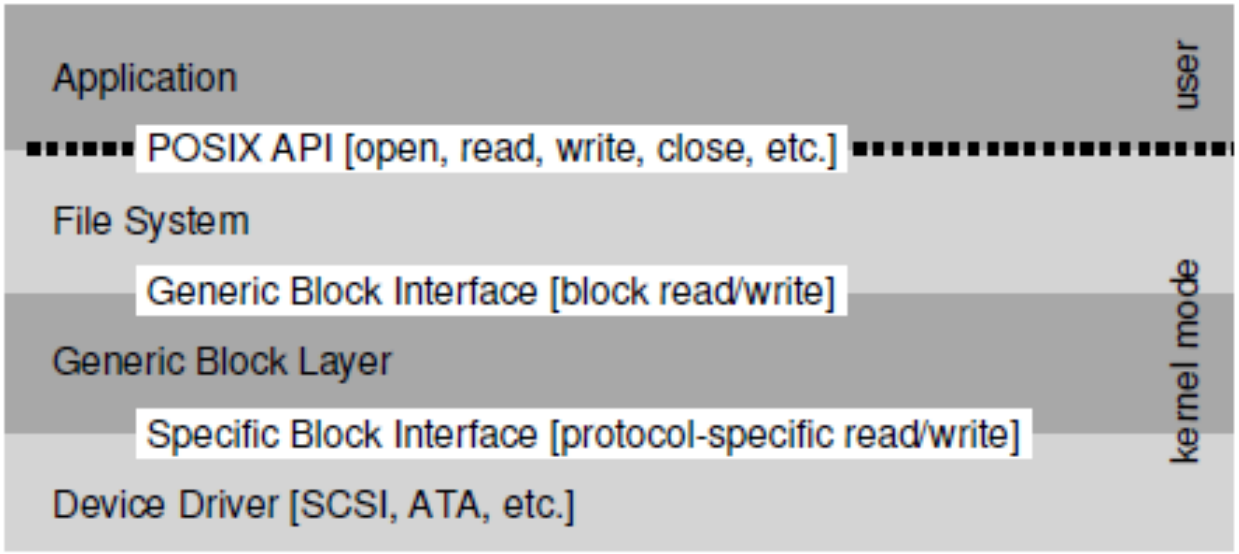
Kernel I/O Structure



Software I/O Interface

- We need software in the OS that knows the details of how a particular device works
  - I/O devices vary widely in function and speed
- Device Driver
  - Make the kernel I/O subsystem look generic as much as possible and independent from hardware
  - No need to recompile the OS every time that the OS has a new device
  - Easier for OS developers and more benefit for hardware manufacturers

Example of File System Stack



- Kernel I/O subsystem provides only basic block read and write in the generic block layer for file system
- How to read and write a specific block is the work of driver

Characteristics of I/O Devices

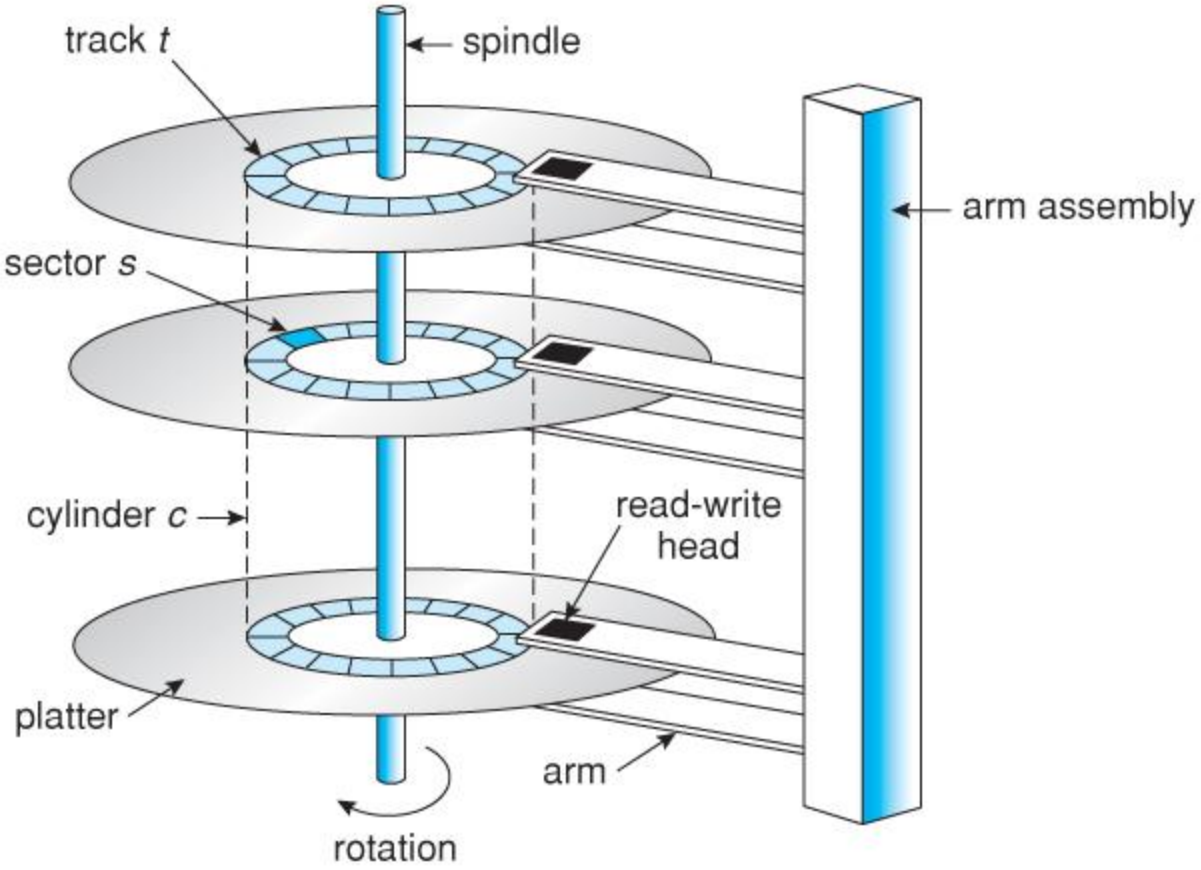
- Data transfer mode (Character-stream or Block)
- Access method (Sequential or Random)
- Transfer schedule (Synchronous or Asynchronous) or (Blocking or Non-blocking)
- Sharing (Dedicated or Sharable)
- Device Speed (transfer rate, latency, or seek time)
- I/O Direction (read only, write only, or read-write)

MASS DATA STORAGE

Introduction

- Hard Disk Drives have been the main form of persistent data storage for decades
- Kernel needs sophisticated algorithms to access the disk
- Storage Structure must be formed under OS management
- Disk Interface
  - Drive consists of a large number of sectors (usually 512-byte blocks), each of which can be read or written
  - Multi-sector operations are possible, but only a single sector operation is *atomic*

Disk Geometry



Disk Geometry

- Disk platter is a flat circular hard surface on which data is stored by magnetic changes on both sides, each of which is called a Surface
- Spindle connects all disk platters together and it is rotated by motor whose rate is measured in rotations per minute (RPM)
- Head is attached with arm above each surface – that arm will move around the surface
- Track is a sub-circle of each surface and each track is divided into sectors – corresponding tracks on each surface form a Cylinder

Disk Speed

- A sector or block is the smallest unit of transfer

$$T_{I/O} = T_{transfer} + T_{seek} + T_{rotation}$$

- Disk speed ($T_{I/O}$) has two parts:
  - The Transfer Rate ($T_{transfer}$)
  - The Positioning Time which consists of two factors
    - The Seek time is the time to move the disk arm to the desired cylinder (T_{seek})
    - The Rotational Delay is the time to rotate the platter to the desired sector ($T_{rotation}$)

Disk Scheduling Algorithms (1)

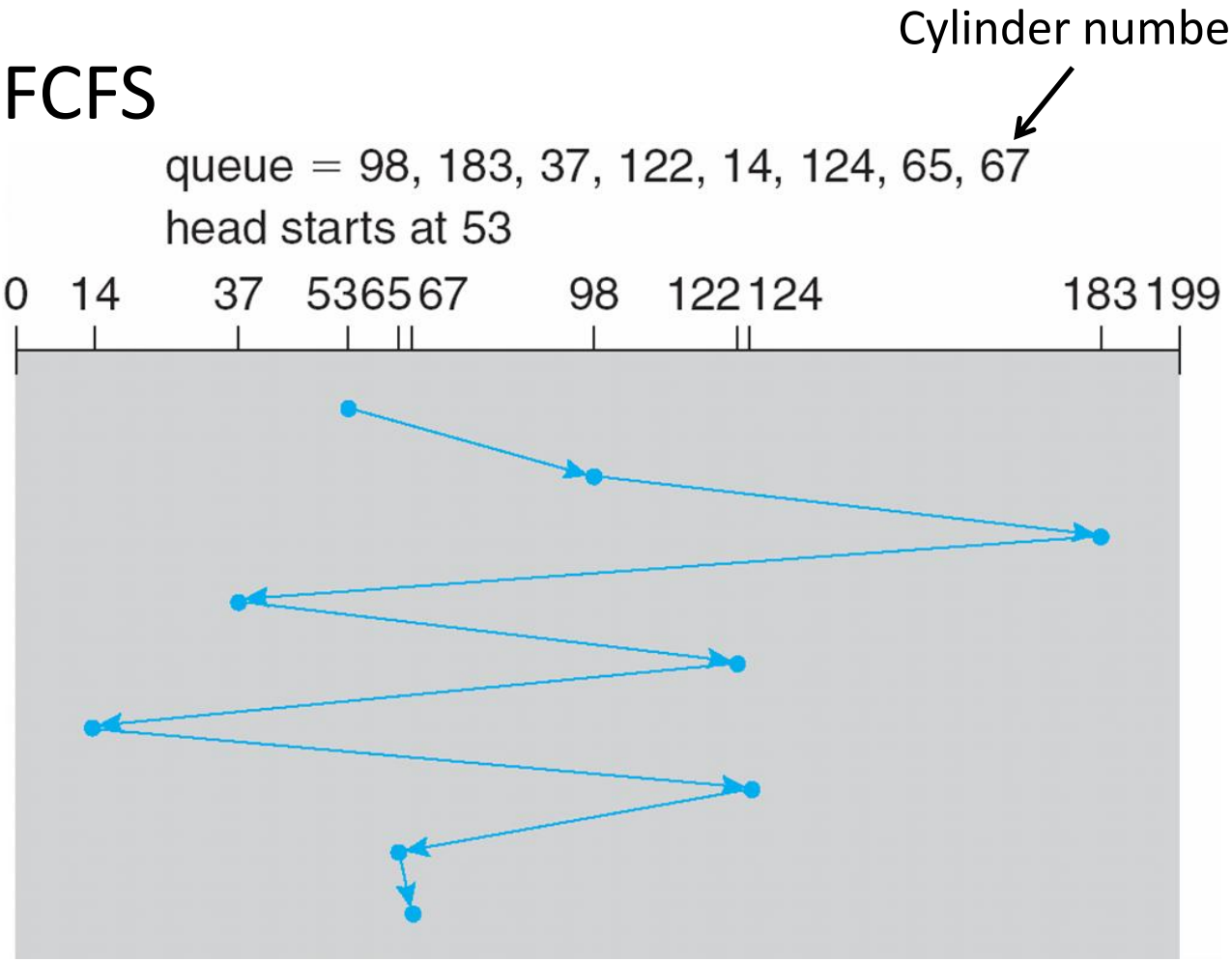
- FCFS (First Come, First Served): It is fair but does not give good performance
- SSTF (Shortest Seek Time First): It is fast but can result in starvation where requests with long seek times are never serviced
- SPTF (Shortest Positioning Time First): On modern disks the rotational latency can be almost as significant as the seek time so SPTF considers both

Disk Scheduling Algorithms (2)

- SCAN: The head sweeps from the outer to inner track and then back to the outer, repeatedly – avoids the starvation problem, but middle tracks get more priority
- C-SCAN (Circle SCAN): It is like the SCAN algorithm, but the head only sweeps in one direction, e.g. outer to inner and then resets to the outer – fairer to inner and outer tracks
- LOOK and C-LOOK: They are like SCAN and C-SCAN, but do not sweep the full width of the disk – the head moves only as far as the final requested block in each direction

FCFS vs. SSTF vs. SCAN

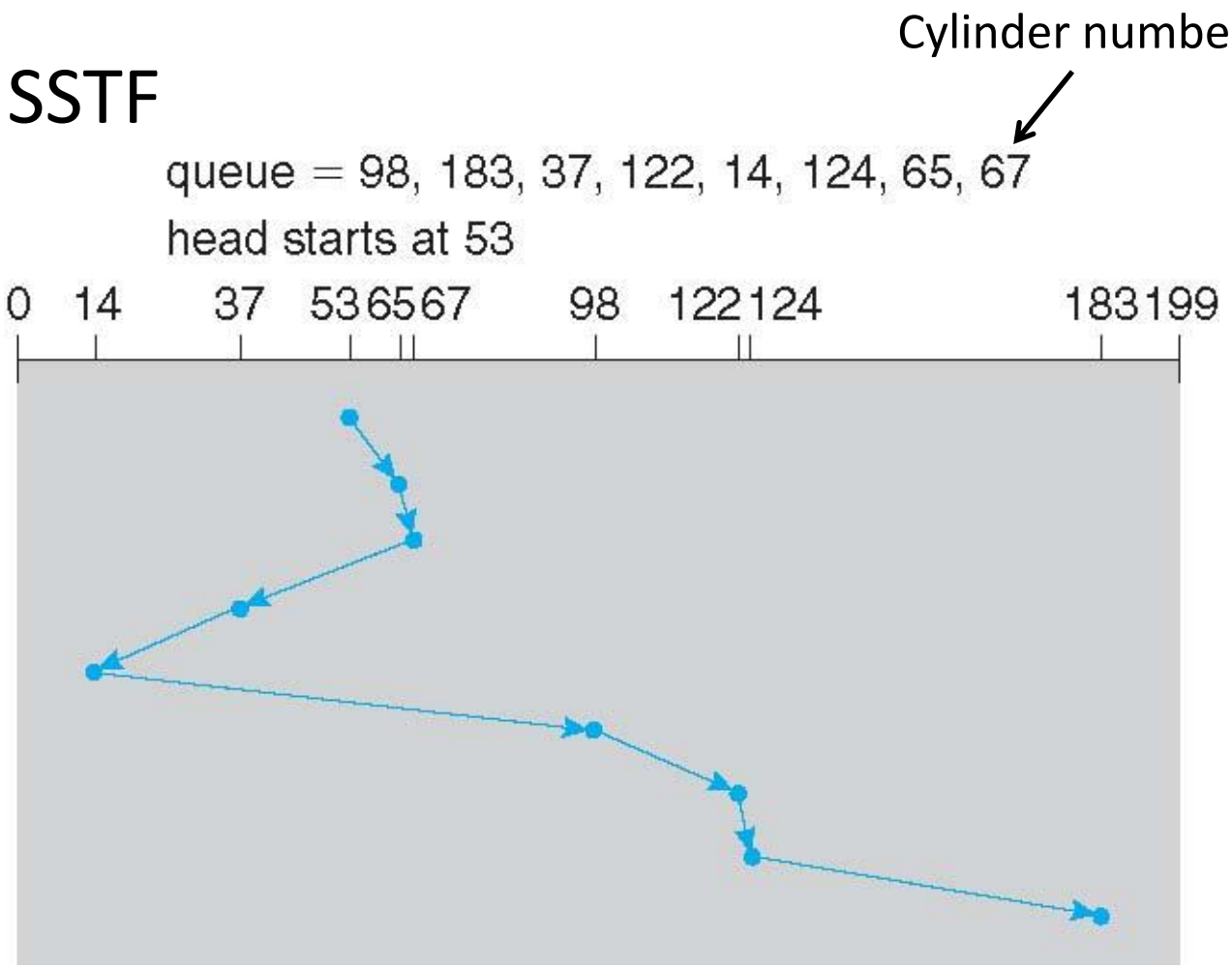
- FCFS



FCFS requires total head movement of 640 cylinders

FCFS vs. SSTF vs. SCAN

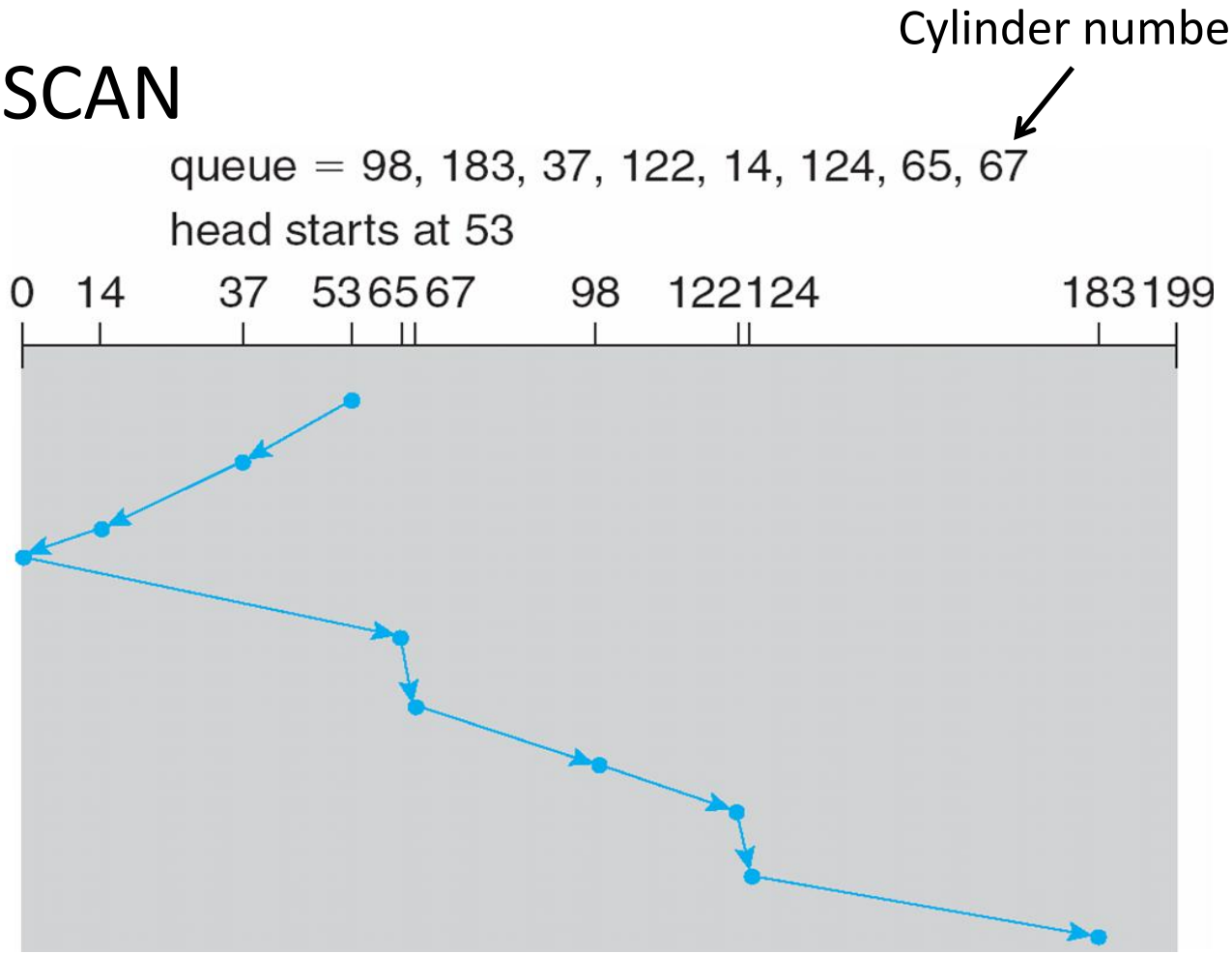
- SSTF



SSTF requires total head movement of 236 cylinders

FCFS vs. SSTF vs. SCAN

- SCAN



SCAN requires total head movement of 208 cylinders

Disk Management (1)

- Disk Formatting
  - Low-level formatting: divide the disk into sectors (usually 512 bytes) each with header and trailer that contain a sector number and Error Correcting Code (ECC)
- Disk Partitioning: before using the disk, the OS needs to group the cylinders into one or more partitions
- Logical Formatting: the OS constructs a file system data structure including maps of free and allocated space

Disk Management (2)

- Boot sector
  - Bootstrap Program: most OS have a tiny bootstrap (boot loader) in the boot ROM that it loads a full bootstrap program into memory
  - It contains the code that understands the disk partitions
- Bad sectors
  - “No disk can be manufactured to 100% perfection”
  - Example of handling bad sectors
    - Manually check bad sectors by using format or check disk command (chkdsk)
    - Using ECC to automatically detect bad sectors by HDD controller