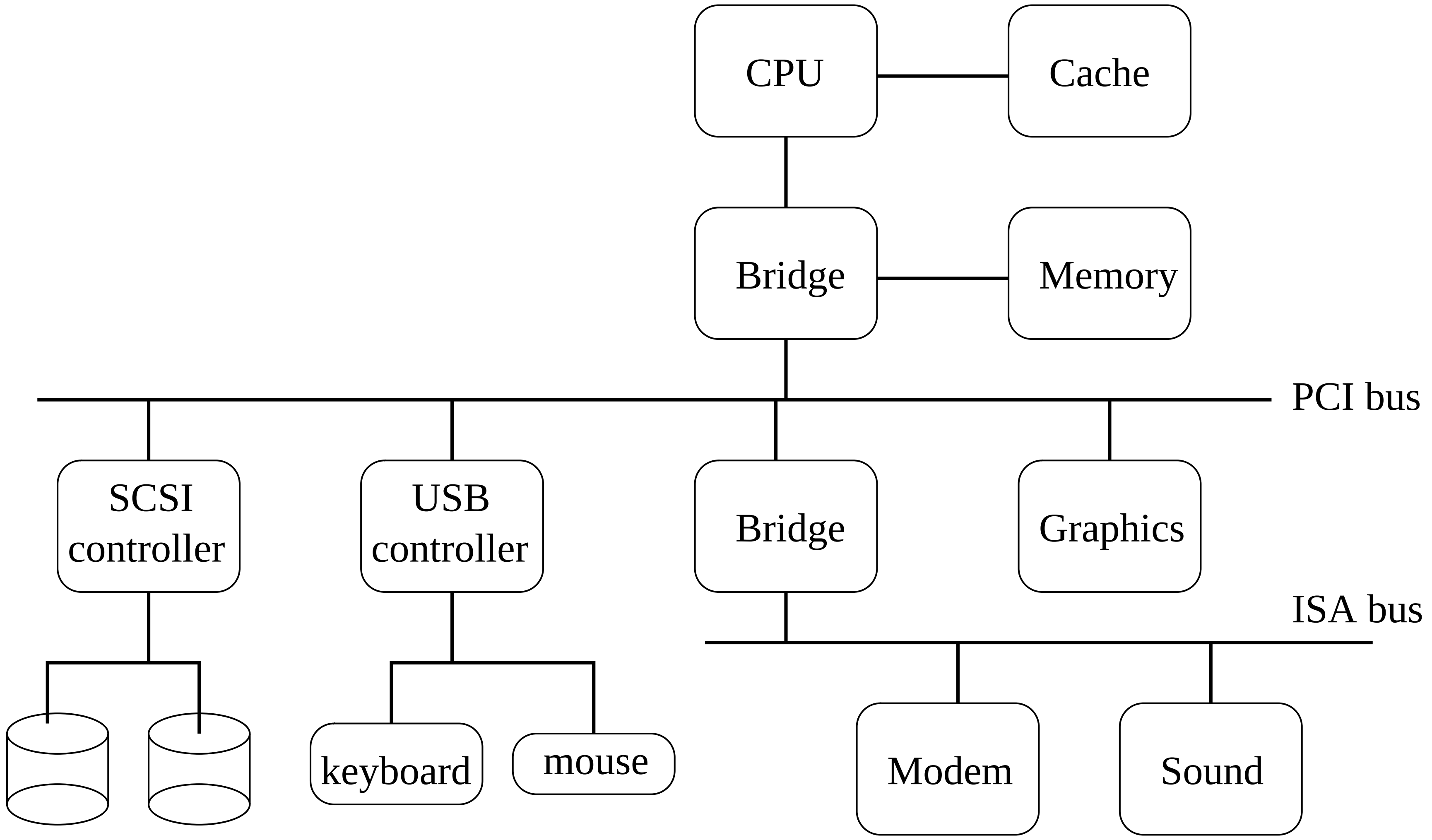
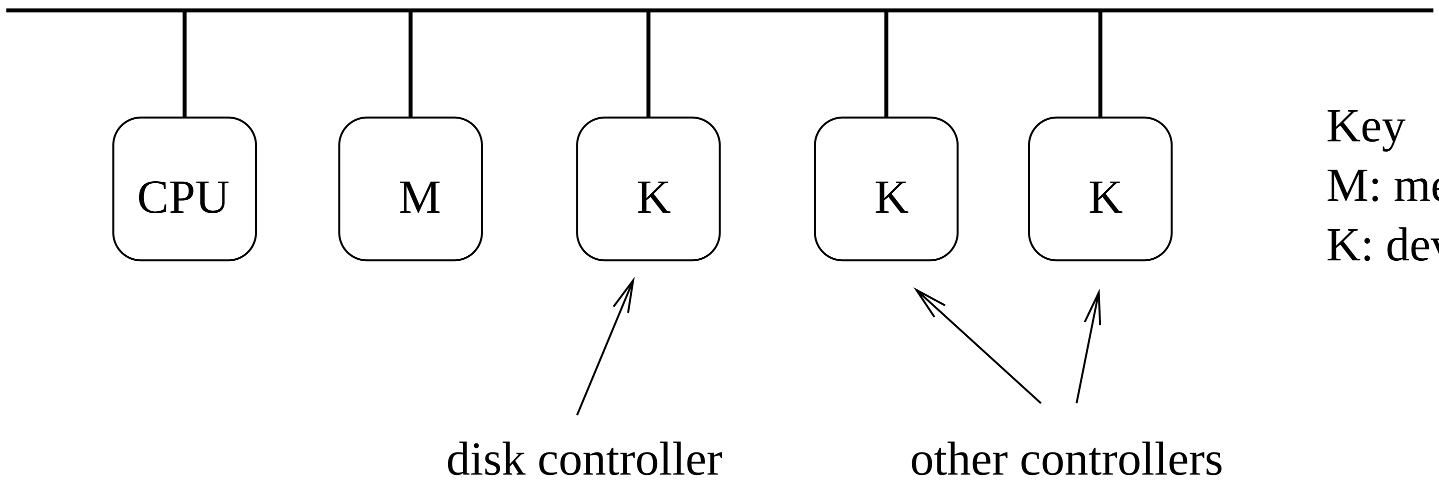
Devices and Device Controllers

- network interface
- graphics adapter
- secondary storage (disks, tape) and storage controllers
- serial (e.g., mouse, keyboard)
- sound
- co-processors
- . . .

Bus Architecture Example



Simplified Bus Architecture



Key
M: memory
K: device controller

Device Interactions

- device registers
  - command, status, and data registers
  - CPU accesses register access via:
    - special I/O instructions
    - memory mapping
- interrupts
  - used by device for asynchronous notification (e.g., of request completion)
  - handled by interrupt handlers in the operating system

Device Interactions: OS/161 Example

```
/* LAMEbus mapping size per slot */
#define LB_SLOT_SIZE         65536
#define MIPS_KSEG1  0xa0000000
#define LB_BASEADDR  (MIPS_KSEG1 + 0x1fe00000)

/* Generate the memory address (in the uncached segment)   *
/* for specified offset into slot’s region of the LAMEbus. *
void *
lamebus_map_area(struct lamebus_softc *bus, int slot, u_int3
{
    u_int32_t address;
    (void)bus;   // not needed

    assert(slot>=0 && slot<LB_NSLOTS);

    address = LB_BASEADDR + slot*LB_SLOT_SIZE + offset;
    return (void *)address;
}
```

Device Interactions: OS/161 Example

```
/* FROM: kern/arch/mips/mips/lamebus_mips.c */
/* Read 32-bit register from a LAMEbus device. */
u_int32_t
lamebus_read_register(struct lamebus_softc *bus,
    int slot, u_int32_t offset)
{
    u_int32_t *ptr = lamebus_map_area(bus, slot, offset);
    return *ptr;
}

/* Write a 32-bit register of a LAMEbus device. */
void
lamebus_write_register(struct lamebus_softc *bus,
    int slot, u_int32_t offset, u_int32_t val)
{
    u_int32_t *ptr = lamebus_map_area(bus, slot, offset);
    *ptr = val;
}
```

Device Interactions: OS/161 Example

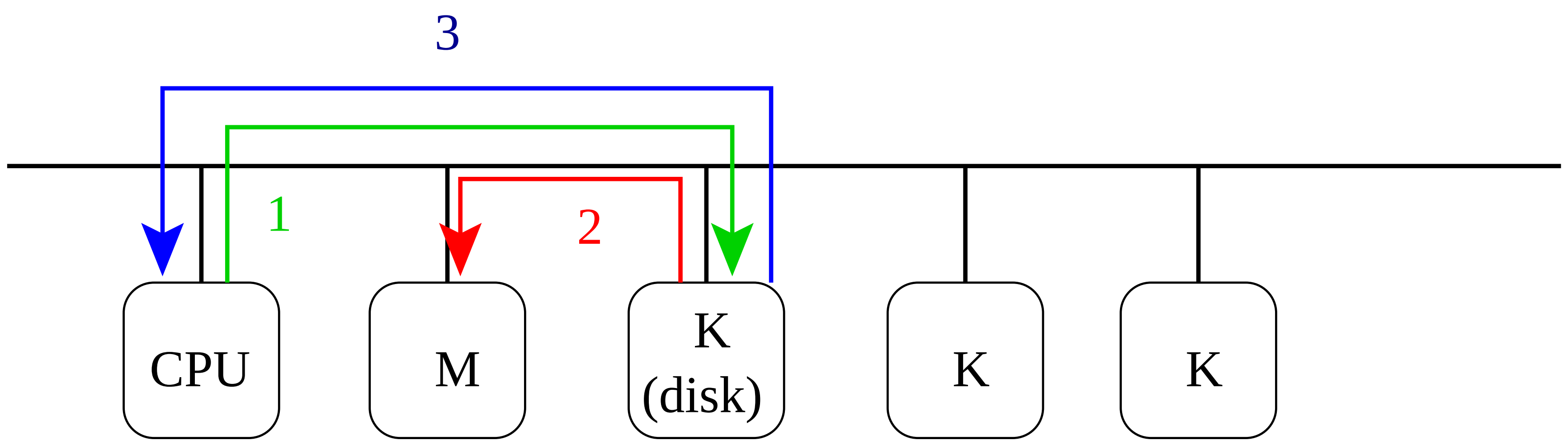
```
/* Registers (offsets within slot) */
#define LT_REG_SEC   0  /* time of day: seconds */
#define LT_REG_NSEC  4  /* time of day: nanoseconds */
#define LT_REG_ROE   8  /* Restart On countdown-timer Expiry flag
#define LT_REG_IRQ   12 /* Interrupt status register */
#define LT_REG_COUNT 16 /* Time for countdown timer (usec) */
#define LT_REG_SPKR  20 /* Beep control */

/* Get the number of seconds from the lamebus timer */
secs = bus_read_register(lt->lt_bus, lt->lt_buspos,
    LT_REG_SEC);

/* Get the timer to beep. Doesn’t matter what value is sent */
bus_write_register(lt->lt_bus, lt->lt_buspos,
    LT_REG_SPKR, 440);
```

Direct Memory Access (DMA)

- used for block data transfers between devices (e.g., disk controller) and memory
- CPU initiates DMA, but can do other work while the transfer occurs



1. CPU issues DMA request to controller
2. controller directs data transfer
3. controller interrupts CPU

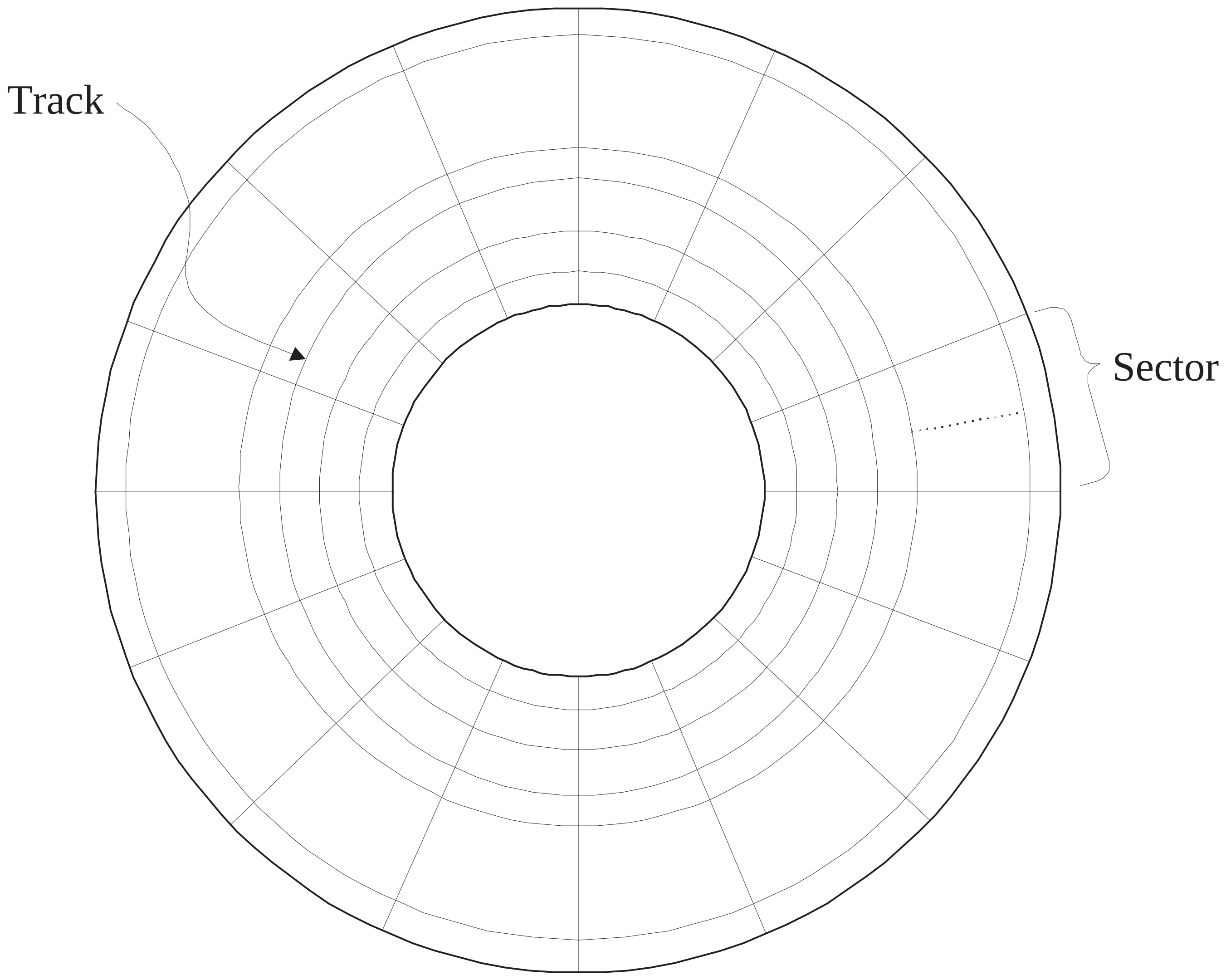
Applications and Devices

- interaction with devices is normally accomplished by device drivers in the OS, so that the OS can control how the devices are used
- applications see a simplified view of devices through a system call interface (e.g., block vs. character devices in Unix)
  - the OS may provide a system call interface that permits low level interaction between application programs and a device
- operating system often *buffers* data that is moving between devices and application programs' address spaces
  - benefits: solve timing, size mismatch problems
  - drawback: performance

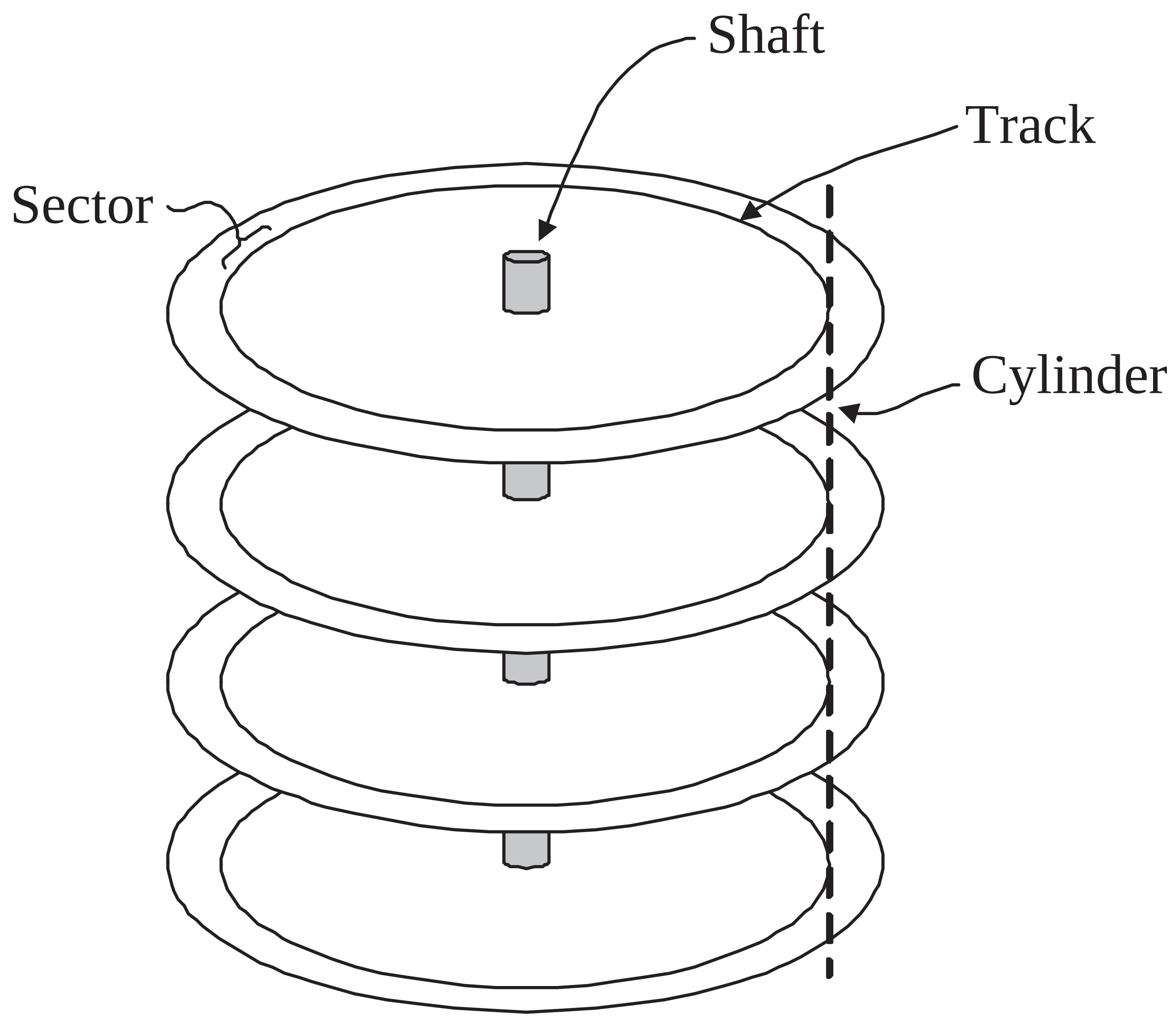
Logical View of a Disk Drive

- disk is an array of numbered blocks (or sectors)
- each block is the same size (e.g., 512 bytes)
- blocks are the unit of transfer between the disk and memory
  - typically, one or more contiguous blocks can be transferred in a single operation
- storage is *non-volatile*, i.e., data persists even when the device is without power

A Disk Platter's Surface



Physical Structure of a Disk Drive



Simplified Cost Model for Disk Block Transfer

- moving data to/from a disk involves:

  seek time: move the read/write heads to the appropriate cylinder

  rotational latency: wait until the desired sectors spin to the read/write heads

  transfer time: wait while the desired sectors spin past the read/write heads

- request service time is the sum of seek time, rotational latency, and transfer time

$$t_{service} = t_{seek} + t_{rot} + t_{transfer}$$

- note that there are other overheads but they are typically small relative to these three

Rotational Latency and Transfer Time

- rotational latency depends on the rotational speed of the disk
- if the disk spins at ω rotations per second:

$$0 \leq t_{rot} \leq \frac{1}{\omega}$$

- expected rotational latency:

$$\bar{t}_{rot} = \frac{1}{2\omega}$$

- transfer time depends on the rotational speed and on the amount of data transferred
- if k sectors are to be transferred and there are T sectors per track:

$$t_{transfer} = \frac{k}{T\omega}$$

Seek Time

- seek time depends on the speed of the arm on which the read/write heads are mounted.
- a simple linear seek time model:
  - $t_{maxseek}$ is the time required to move the read/write heads from the innermost cylinder to the outermost cylinder
  - C is the total number of cylinders
- if k is the required *seek distance* ($k > 0$):

$$t_{seek}(k) = \frac{k}{C} t_{maxseek}$$

Disk Head Scheduling

- goal: reduce seek times by controlling the order in which requests are serviced
- disk head scheduling may be performed by the controller, by the operating system, or both
- for disk head scheduling to be effective, there must be a queue of outstanding disk requests (otherwise there is nothing to reorder)
- an on-line approach is required: the disk request queue is not static

FCFS Disk Head Scheduling

- handle requests in the order in which they arrive

  +: fair, simple

  -: no optimization of seek times

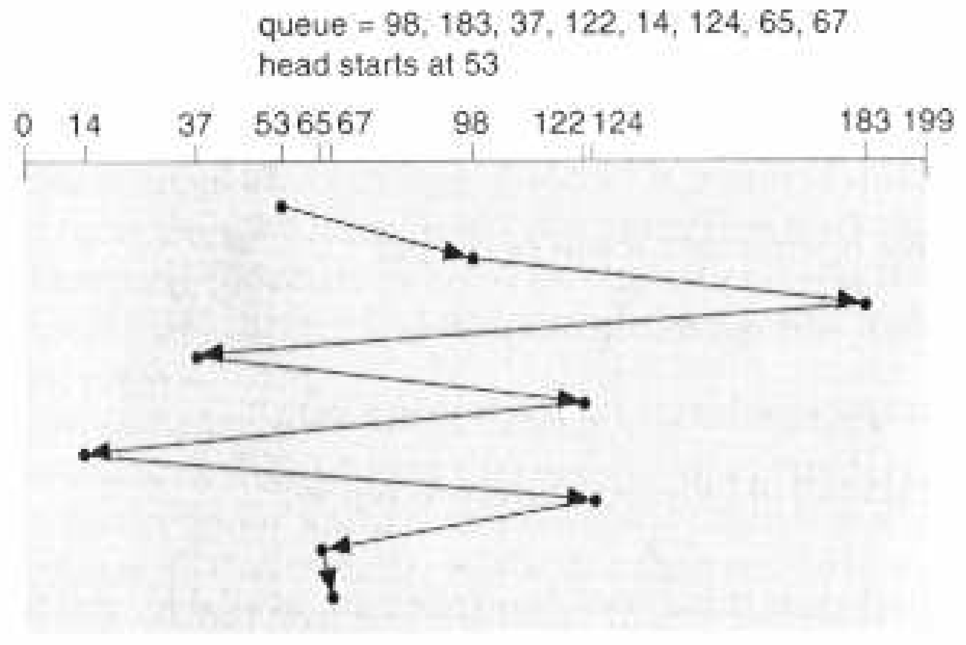


Figure 14.1 FCFS disk scheduling.

Figure from *Operating Systems Concepts, 6th Ed.*, Silberschatz, Galvin, Gagne. Wiley. 2003

Shortest Seek Time First (SSTF)

- choose closest request (a greedy approach)

  +: seek times are reduced

  -: starvation of distant requests

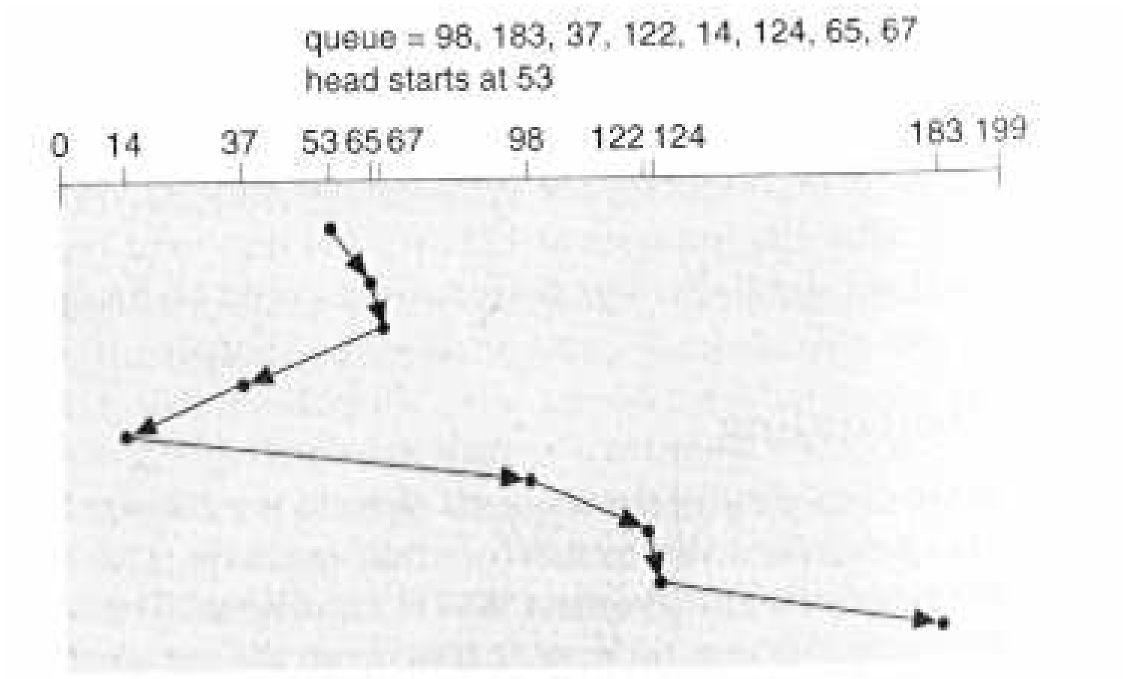


Figure 14.2 SSTF disk scheduling.

Figure from *Operating Systems Concepts, 6th Ed.*, Silberschatz, Galvin, Gagne. Wiley. 2003

SCAN and LOOK

- LOOK is the commonly-implemented variant of SCAN. Also known as the “elevator” algorithm.
- Under LOOK, the disk head moves in one direction until there are no more requests in front of it, then reverses direction.
- seek time reduction without starvation
- SCAN is like LOOK, except the read/write heads always move all the way to the edge of the disk in each direction.

SCAN Example

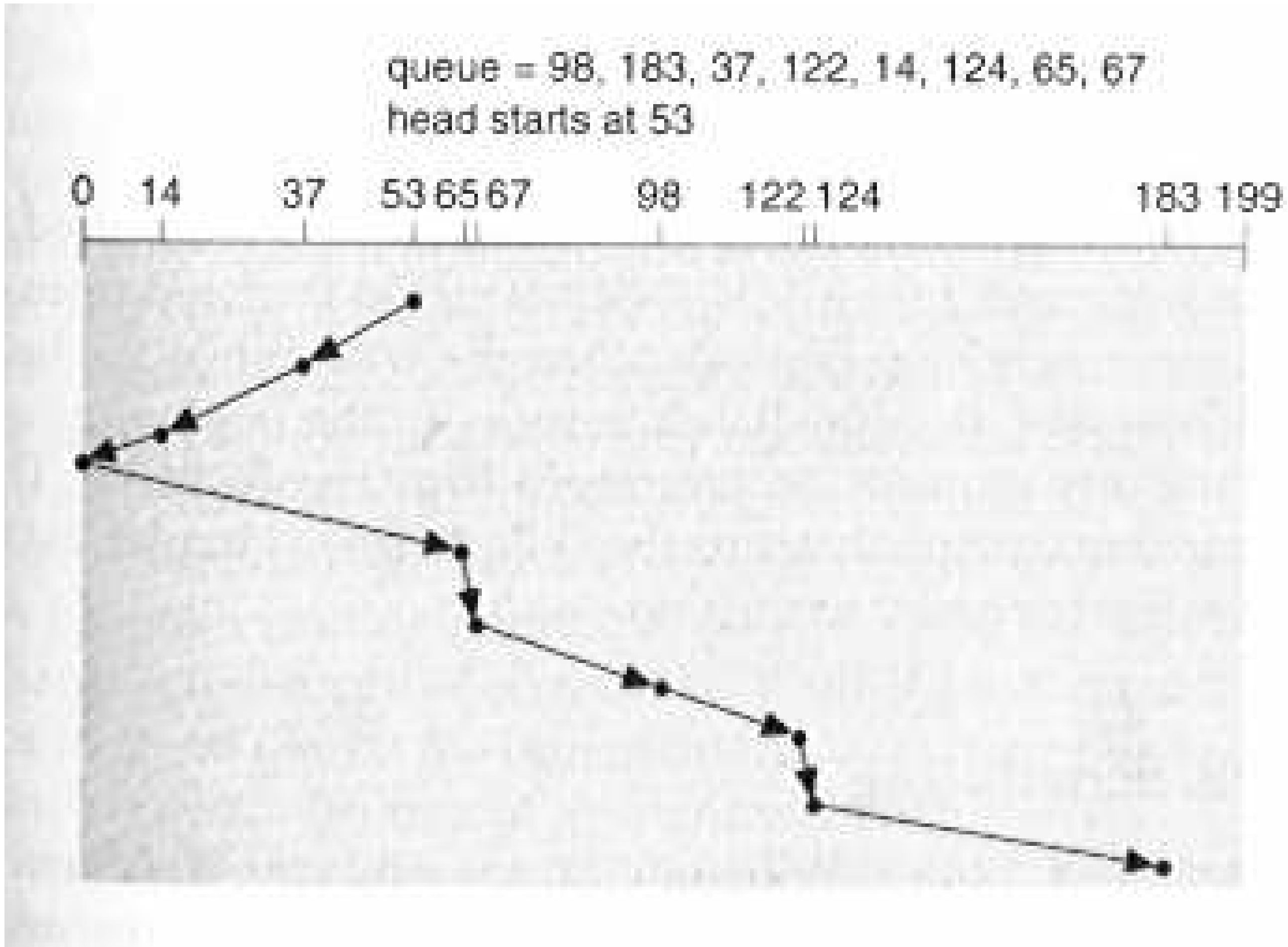


Figure 14.3 SCAN disk scheduling.

Figure from *Operating Systems Concepts*, *6th Ed.*, Silberschatz, Galvin, Gagne. Wiley. 2003

Circular SCAN (C-SCAN) and Circular LOOK (C-LOOK)

- C-LOOK is the commonly-implemented variant of C-SCAN
- Under C-LOOK, the disk head moves in one direction until there are no more requests in front of it, then it jumps back and begins another scan in the same direction as the first.
- C-LOOK avoids bias against “edge” cylinders

C-LOOK Example

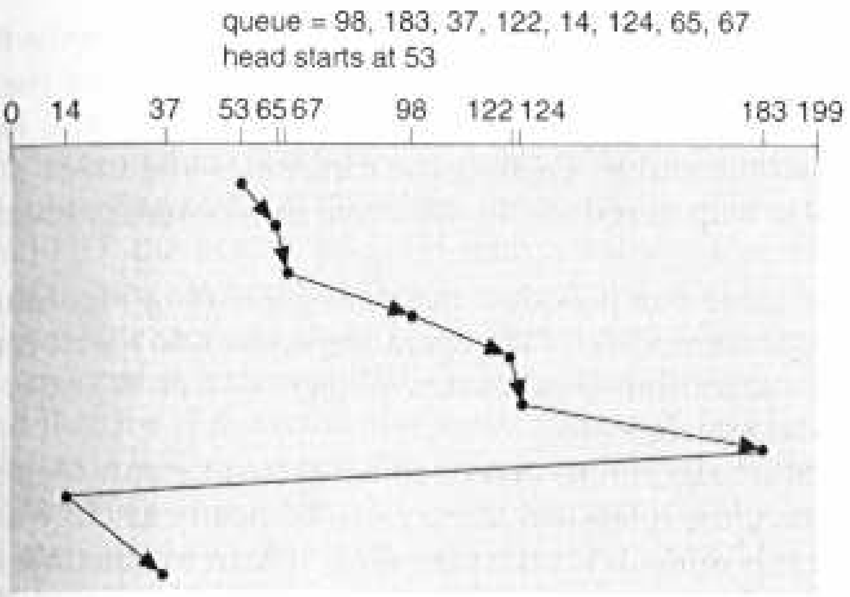


Figure 14.5 C-LOOK disk scheduling.

Figure from *Operating Systems Concepts, 6th Ed.*, Silberschatz, Galvin, Gagne. Wiley. 2003