Input/Output

Chapter 6

Slides Borrowed from Modern Operating System, Operating System Concepts, OSPP, and OSTEP

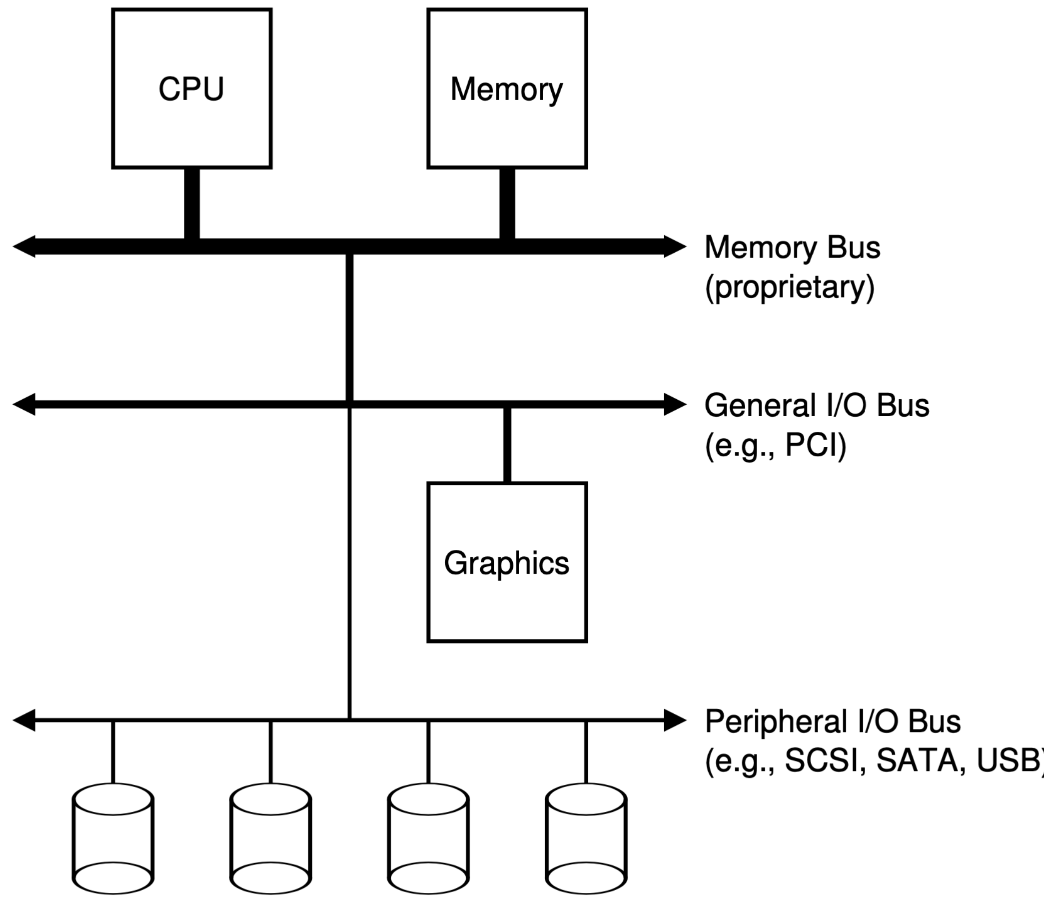
Outline

- Input/Output (I/O) devices
  – How should I/O be integrated into systems?
  – What are the general mechanisms?
- Hard disk drive
  – How is the data actually laid out and accessed?
  – How does disk scheduling improve performance?
- RAID
  – How can we make a large, fast, and reliable storage system?

A Typical System Architecture

A general I/O bus and several peripheral buses to connect devices

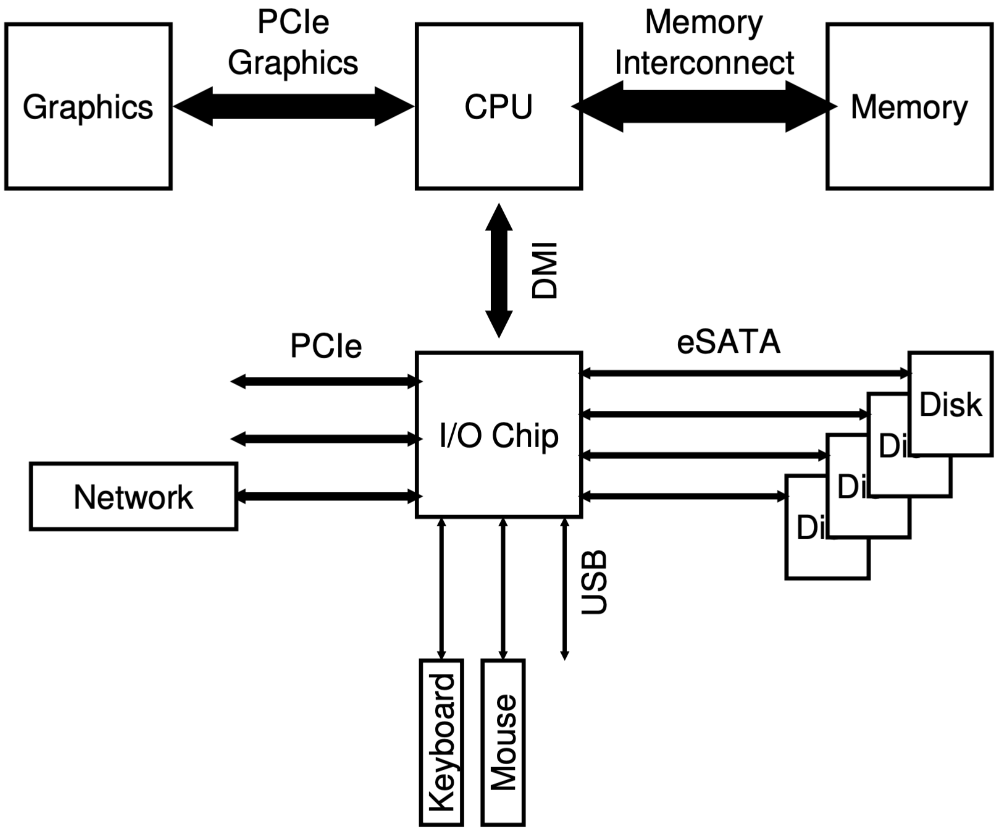
- A hierarchical structure
  - Physics and cost: the faster a bus is, the shorter it must be
  - Devices that demand high performance are nearer the CPU



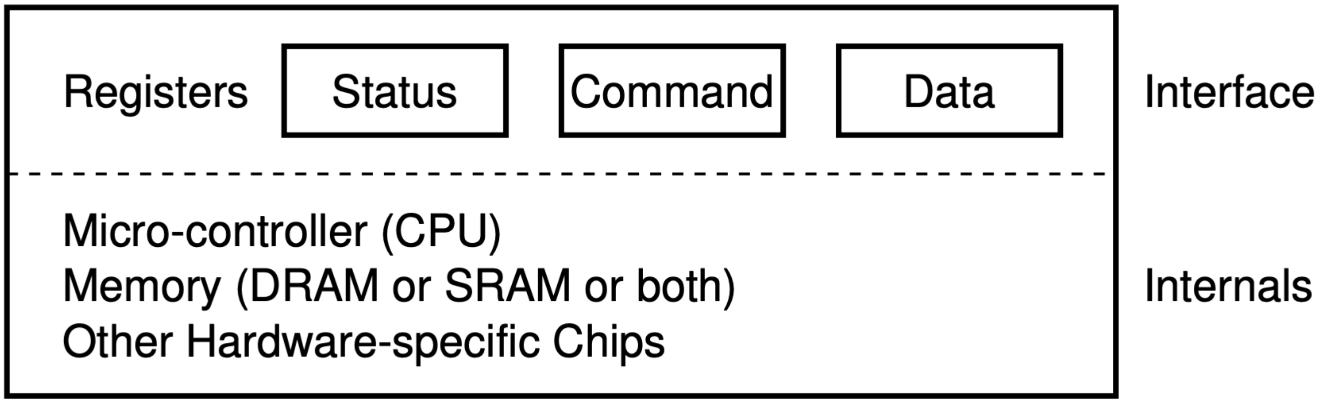
A Typical System Architecture

A general I/O bus and several peripheral buses to connect devices

- Modern systems usually use specialized chipsets to improve performance

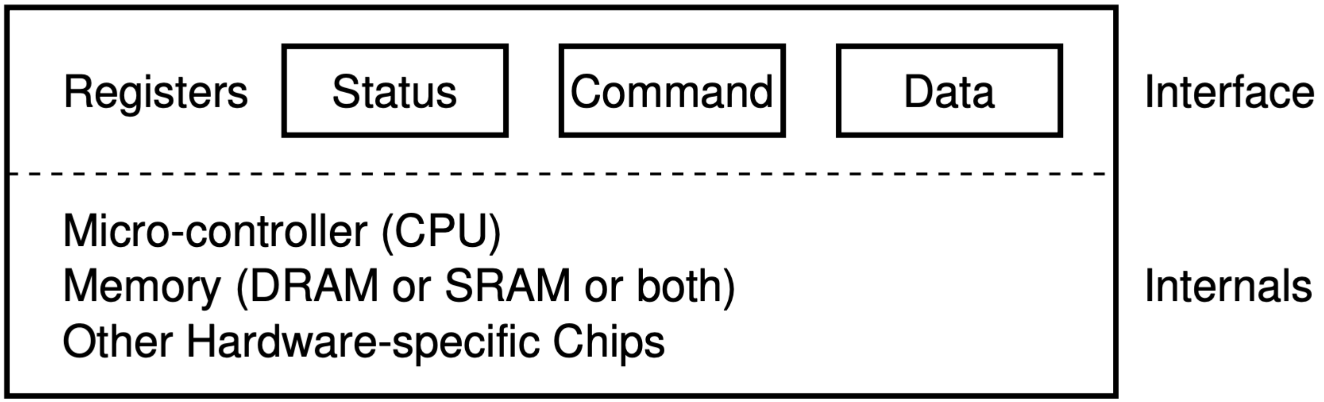


A Canonical Device



- A device has two important components
  - The **hardware interface** it presents to the rest of the system (allow OS to control its operation)
  - Its **internal structure** (implementation specific)

A Canonical Device



- Device interface is comprised of several registers
  - **Status Register**: read to see the current status of the device
  - **Command Register**: tell the device to perform a certain task
  - **Data Register**: pass data to the device, or get data from the device

Device Interaction

How should the CPU communicate with a device (specify a way for the OS to send data to specific device registers)?

- **I/O Instructions**
  - Each control register is assigned an **I/O port** number
  - Use special I/O instructions (on x86, `in` and `out`)
  - Such instructions are usually privileged

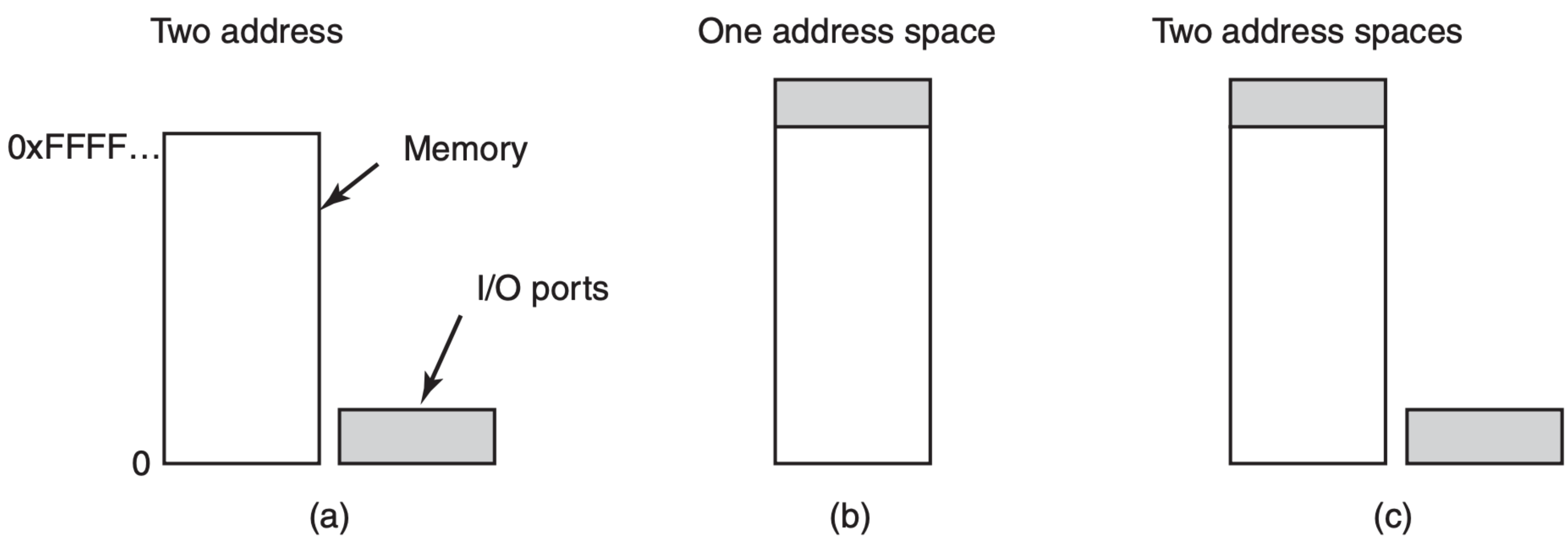
Device Interaction

How should the CPU communicate with a device (specify a way for the OS to send data to specific device registers)?

- **Memory Mapped I/O**: Map all the control registers into the memory space
  - Each control register is assigned a unique memory address
  - To access a particular register, the OS issues a load (to read) or store (to write) the address
  - The hardware then routes the load/store to the device instead of main memory

Device Interaction

How should the CPU communicate with a device (specify a way for the OS to send data to specific device registers)?



Separate I/O and memory space

Memory-mapped I/O

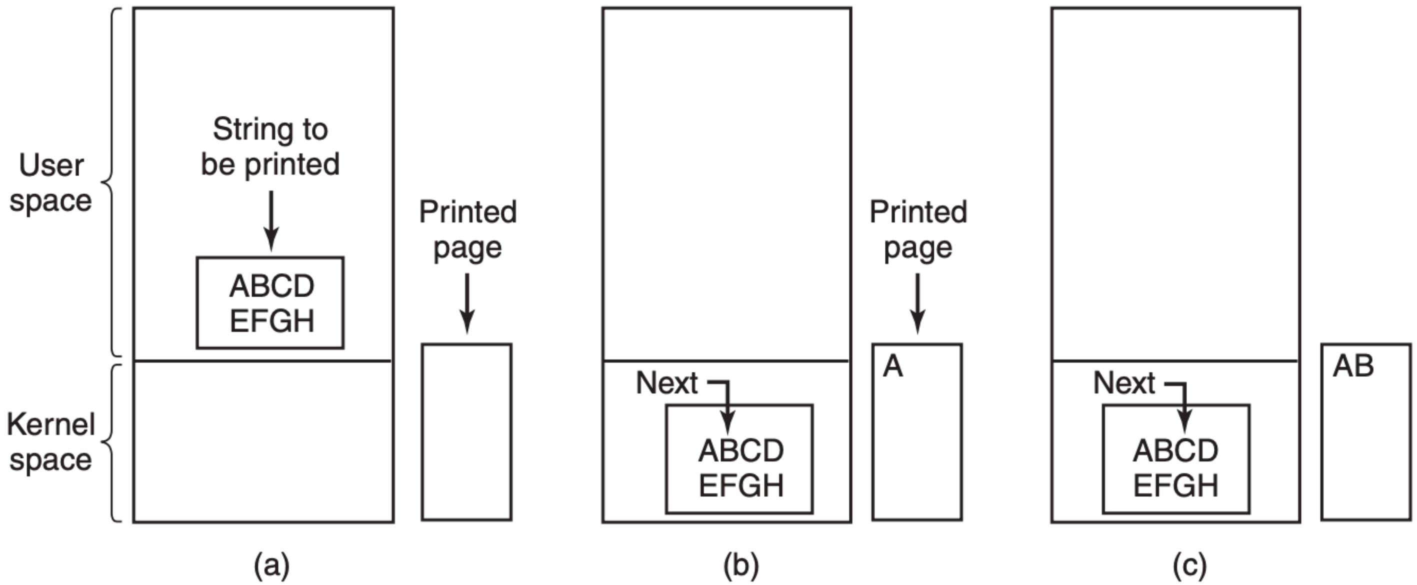
Hybrid

Programmed I/O

A basic interaction: **Polling** the device (busy waiting)

- OS waits until the device is ready to receive a command by repeatedly reading the status register
- OS sends some data down to the data register
- OS writes a command to the command register
- OS waits for the device to finish by again polling it in a loop, waiting to see if it is finished
  - It may then get an error code to indicate success or failure
- The main processor is involved in the data transfer

Programmed I/O

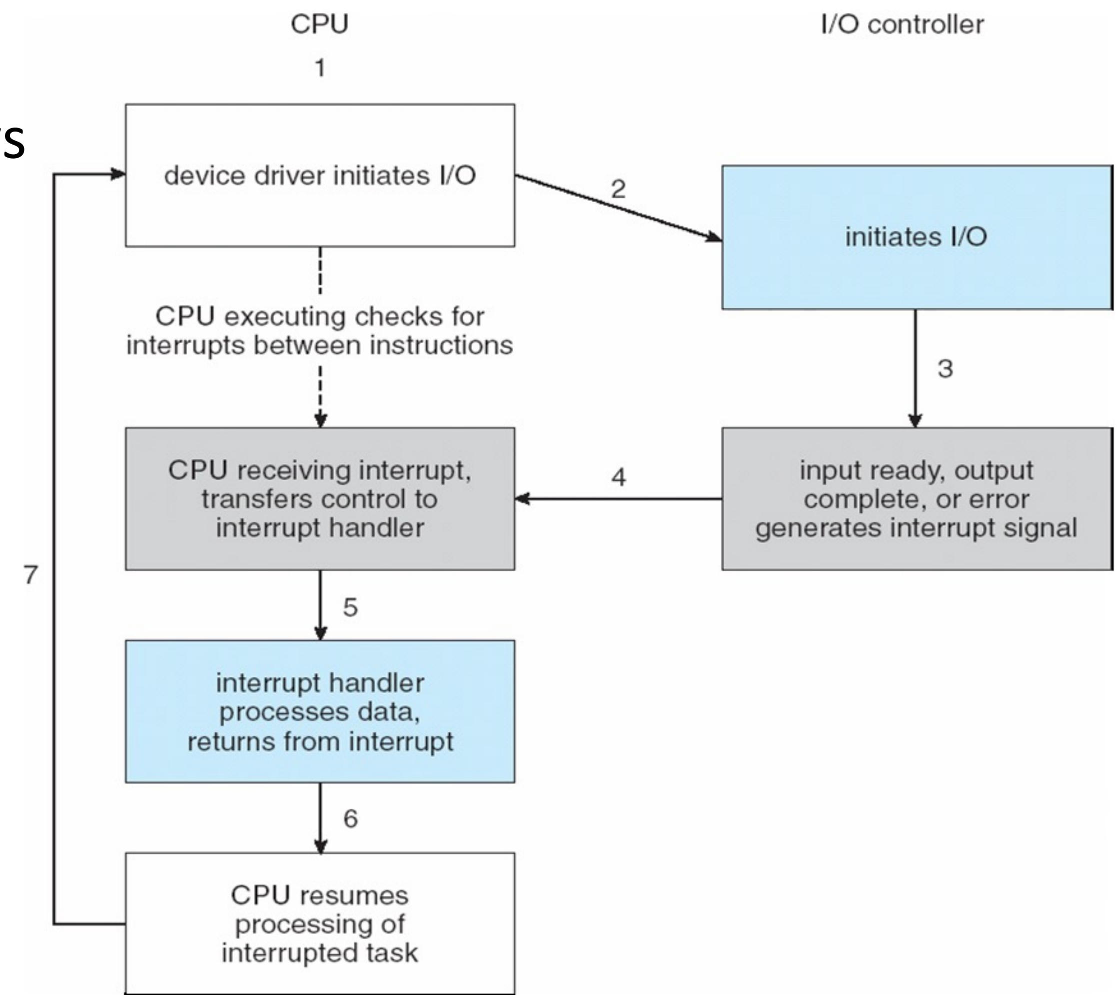



```
copy_from_user(buffer, p, count);              /* p is the kernel buffer */
for (i = 0; i < count; i++) {                  /* loop on every character */
      while (*printer_status_reg != READY) ;   /* loop until ready */
      *printer_data_register = p[i];           /* output one character */
}
return_to_user( );
```

Interrupts

How to lower the CPU overhead required to manage the device?

- **Interrupts:** Hardware mechanism that allows a device to interrupt the processor
- The CPU catches the interrupt and dis-patches it to the **interrupt handler**



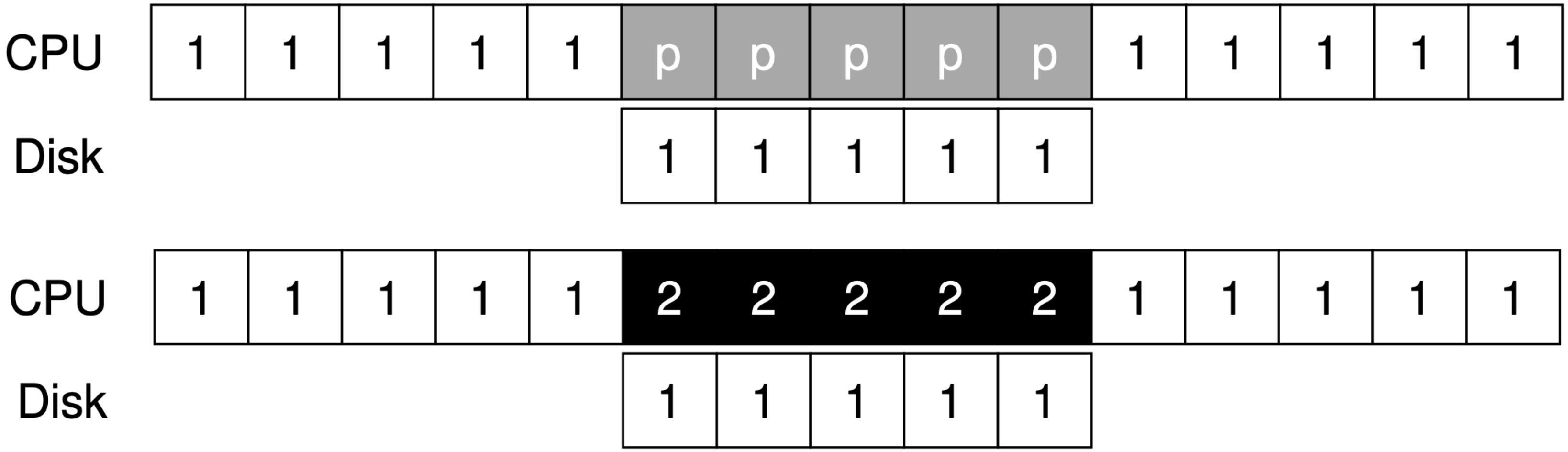
Interrupts

How to lower the CPU overhead required to manage the device?

- OS issues a request, puts the calling process to sleep, and context switch to another task
- When the device is finished with the operation, it will raise a hardware interrupt
- The interrupt causes CPU to jump into the OS at a pre-determined interrupt service routine **(interrupt handler)**
- The handler finishes the request (e.g., by reading data from the device) and wakes the process waiting for the I/O

Interrupts

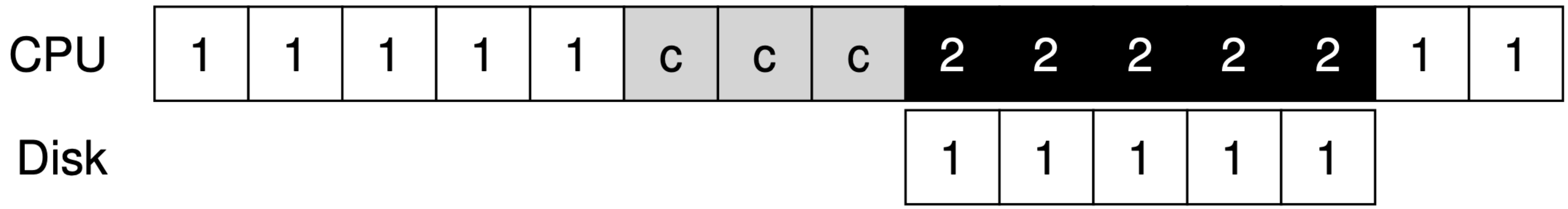
- Interrupts allow for **overlap** of computation and I/O



- The use of interrupts is not always the best solution
  - For example, a device that performs its tasks very quickly
  - Use a **hybrid** that polls for a little while and then, if the device is not yet finished, uses interrupts

Interrupts

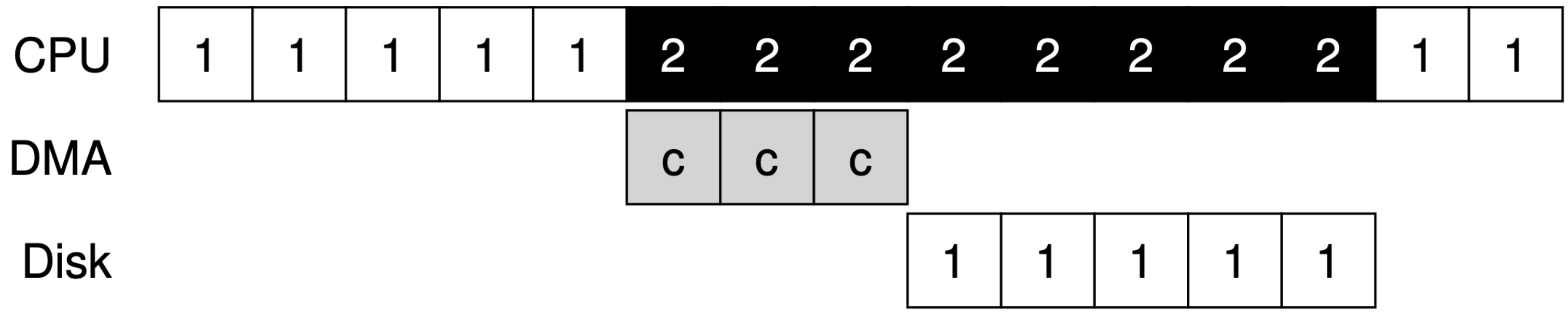
- **Interrupts take time**: To transfer a large chunk of data to a device, the CPU can be overburdened with a rather trivial task
  – When initiating the I/O, CPU must copy the data from memory to the device explicitly, one word at a time
  – Waste a lot of time and effort that could better be spent running other processes



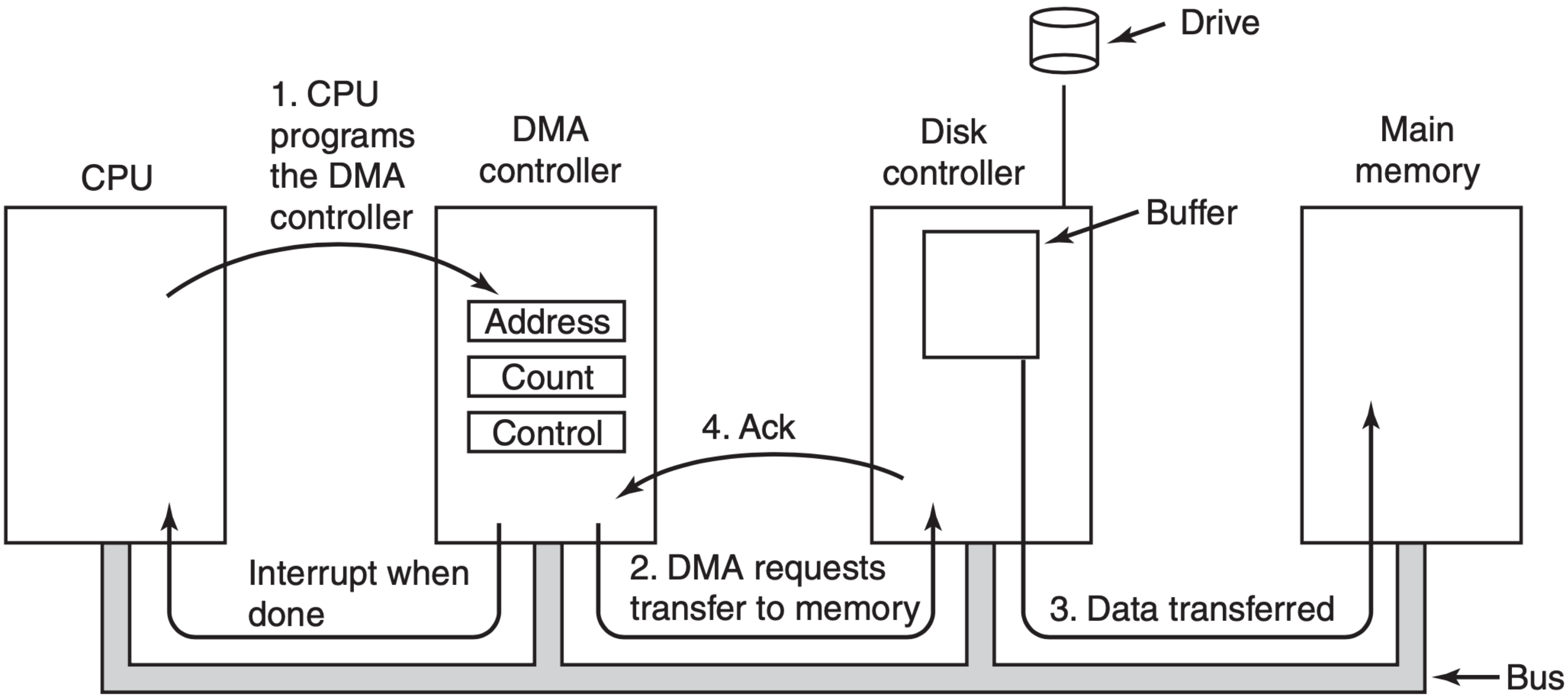
Direct Memory Access (DMA)

Direct Memory Access (DMA): A DMA controller is a specific device within a system that can orchestrate transfers between devices and main memory without much CPU intervention

- OS programs the DMA by telling it where the data lives in, how much data to copy, and which device to send it to
- When the DMA is complete, the DMA controller raises an interrupt



Direct Memory Access (DMA)



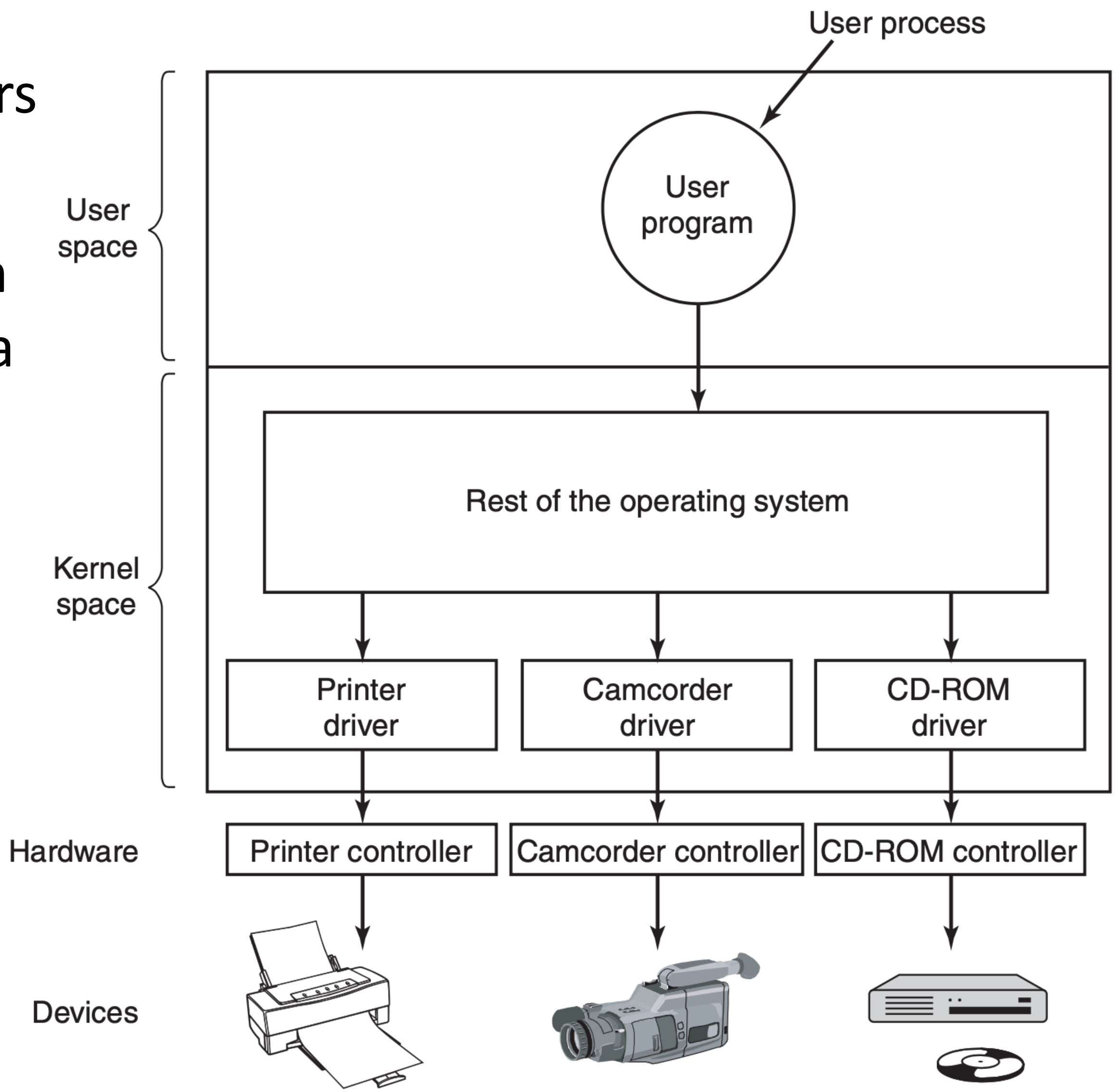
Device Driver

How to fit various devices (with specific interfaces) into the OS (keep as general as possible)?

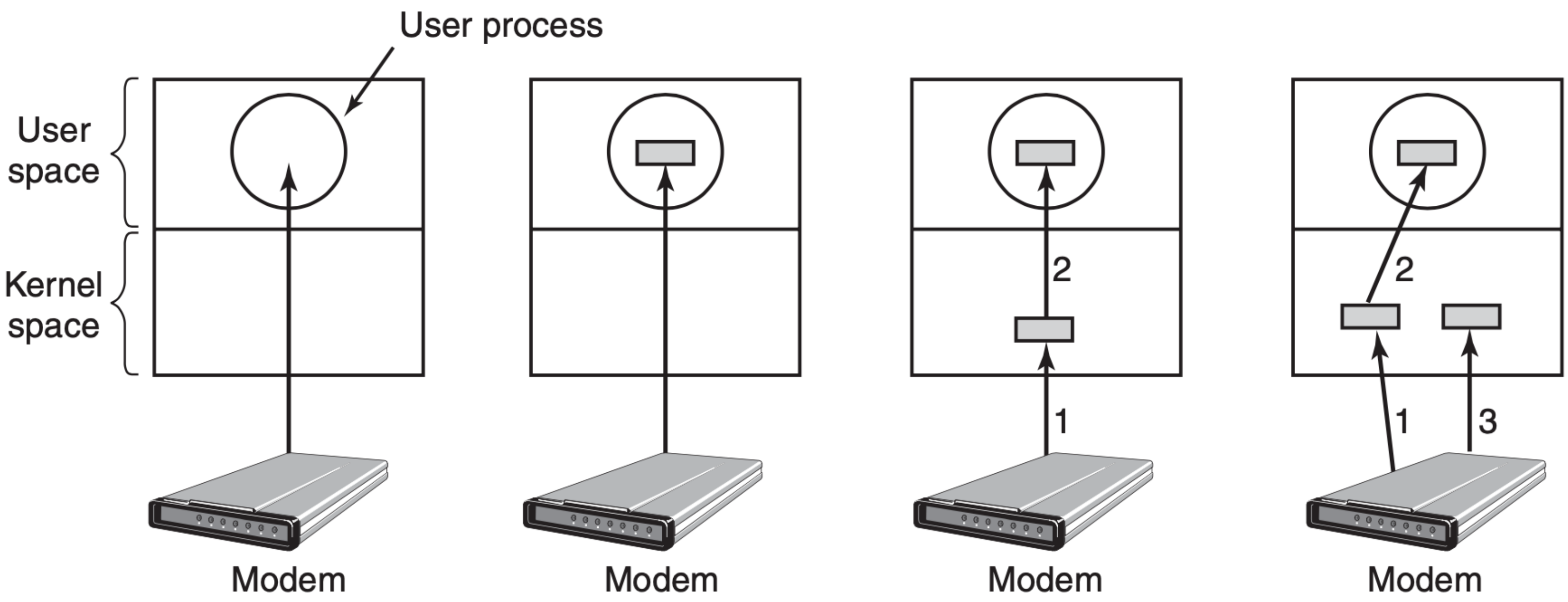
- **Device driver**: A piece of software in OS, which knows in detail how a device works (abstraction)
  - The driver exposes a generic interface to the rest of the OS
    - Any new device should come with a driver that implements (at least part of) the standard I/O interface
  - Device drivers are accounting for 70% of the OS code
    - Device drivers are needed for any device you might plug into your system

Device Driver

- OS often classifies drivers into categories:
  - **Block Devices**, which contain multiple data blocks (e.g., disk)
  - **Character Devices**, which generate or accept a stream of characters (e.g., keyboard)



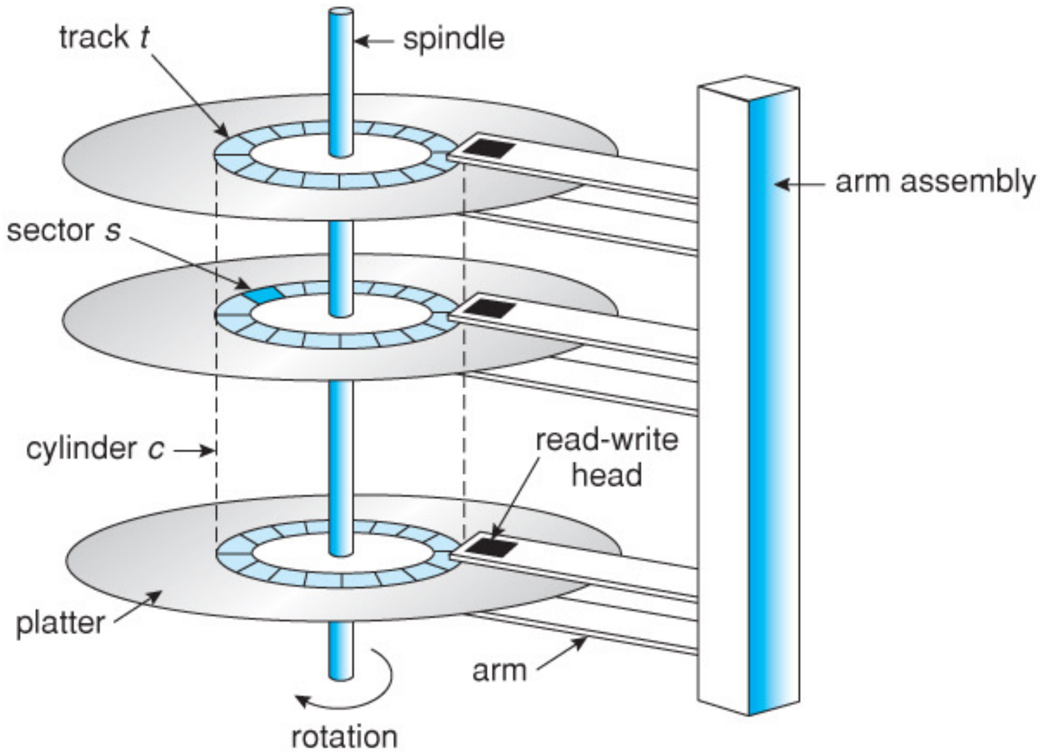
Buffering



- Unbuffered (user process has to be started up each time)
- Buffering in user space (what if the buffer is paged out)
- Buffering in the kernel
- Double buffering in the kernel

Hard Disk

The main form of persistent data storage in computer systems



Hard Disk

Basic geometry of a disk

- **Platter**: a circular hard surface on which data is stored persistently by inducing magnetic changes to it
- **Surface**: One side of a platter, on which data is encoded
- **Tracks**: A surface is divided into concentric tracks
  - Many thousands of tracks on a surface
  - A stack of tracks of fixed radius is a **cylinder**
- **Sector**: A track is divided into 512-byte blocks called sectors
  - A sector is the granularity for atomic operations

Hard Disk

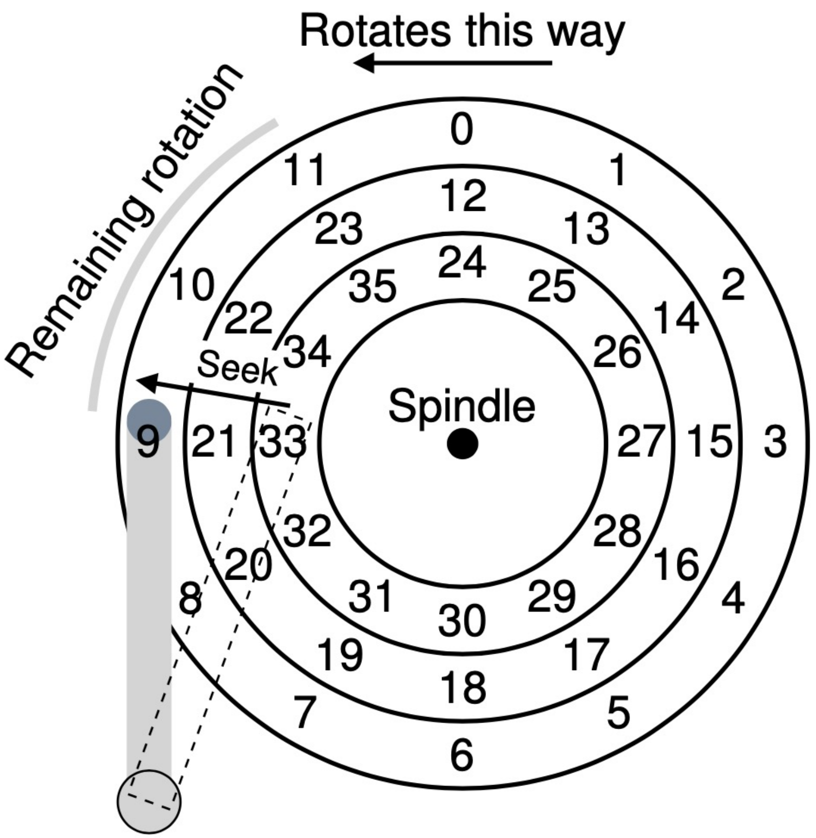
Basic geometry of a disk

- **Spindle:** Platters are bound together around the spindle, which is connected to a motor that spins the platters at a fixed rate
  - The rate of rotation is often measured in **rotations per minute (RPM),** typically ranging from 7,200 to 15,000 RPM
- **Head/Arm:** Reading or writing is accomplished by a disk head attached to a disk arm.
  - One head per surface
  - Heads record and sense data along tracks
  - Generally only one head is active at a time

Hard Disk

To service a request

- **Seek Time**: move the disk arm to the correct track
- **Rotational delay**: wait for the desired sector to rotate under the disk head
- **Transfer:** read or write data
- Seek and rotational times dominate the cost of small accesses



Hard Disk

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} \qquad R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

- To transfer a 4 KB block in a random workload
  - $T_{seek} = 4\ ms$ (average seek)
  - $T_{rotation} = 2\ ms$ (average rotation)
  - $T_{transfer} = 0.03\ ms$
  - $R_{I/O} = 4\ KB / 6.03\ ms$
    $= 0.65\ MB/s$

	Cheetah 15K.5
Capacity	300 GB
RPM	15,000
Average Seek	4 ms
Max Transfer	125 MB/s
Platters	4
Cache	16 MB
Connects via	SCSI

Hard Disk

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} \qquad R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

- To transfer 100 MB data in a sequential workload
  - $T_{seek} = 4\ ms$ (average seek)
  - $T_{rotation} = 2\ ms$ (average rotation)
  - $T_{transfer} = 0.8\ s$
  - $R_{I/O} = 100\ MB / 0.806\ s$
    $= 124\ MB/s$
  - A single seek and rotation before a very long transfer

	Cheetah 15K.5
Capacity	300 GB
RPM	15,000
Average Seek	4 ms
Max Transfer	125 MB/s
Platters	4
Cache	16 MB
Connects via	SCSI

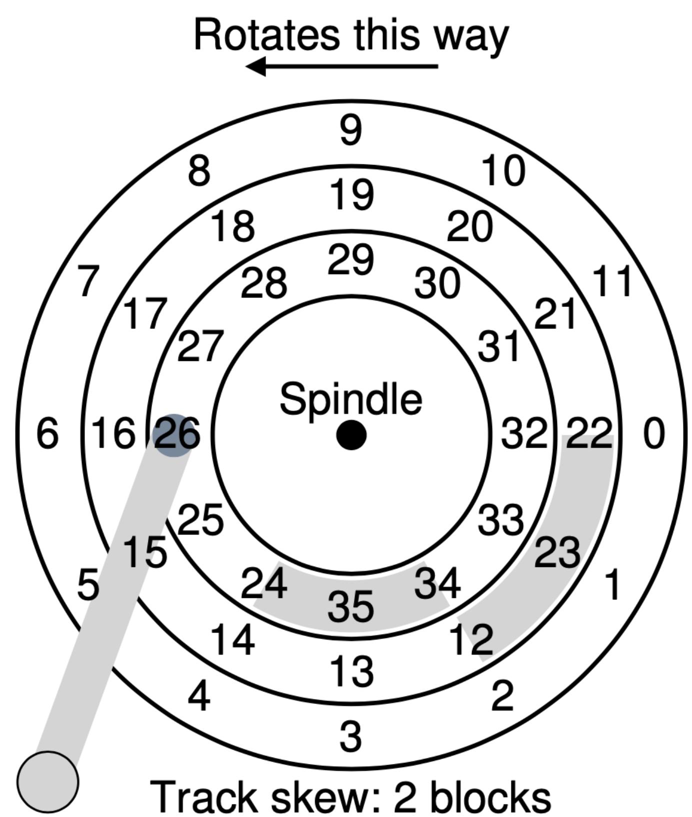
Hard Disk

- Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 3000 HLFS hard disk (after 30 years)

Parameter	IBM 360-KB floppy disk	WD 3000 HLFS hard disk
Number of cylinders	40	36,481
Tracks per cylinder	2	255
Sectors per track	9	63 (avg)
Sectors per disk	720	586,072,368
Bytes per sector	512	512
Disk capacity	360 KB	300 GB
Seek time (adjacent cylinders)	6 msec	0.7 msec
Seek time (average case)	77 msec	4.2 msec
Rotation time	200 msec	6 msec
Time to transfer 1 sector	22 msec	1.4 μsec

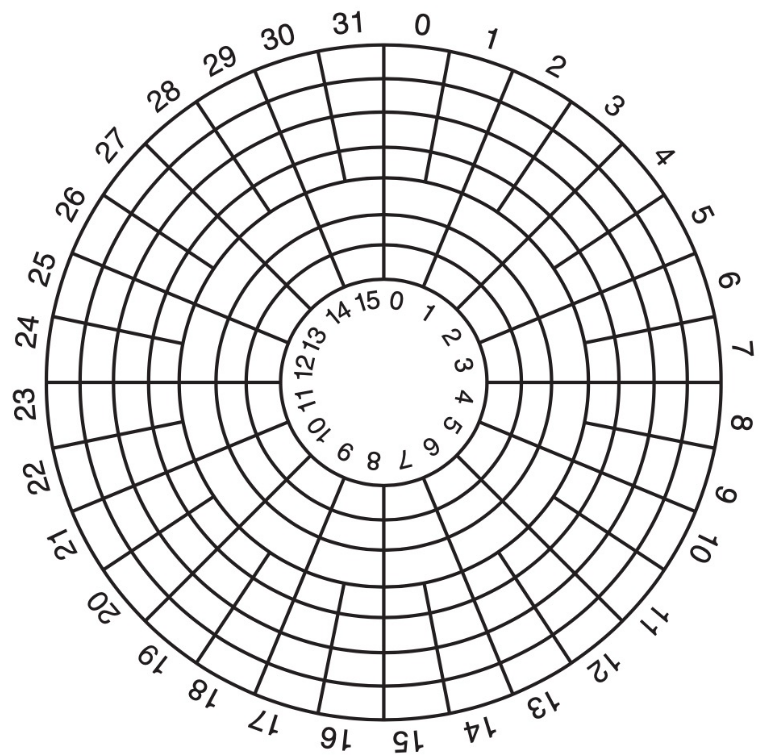
Hard Disk

- **Track (Cylinder) Skew:** make sure that sequential reads can be properly serviced when crossing track boundaries
  - When switching from one track to another, the disk needs time to reposition the head
  - Without such skew, the head would be moved to the next track but the desired next block would have already rotated under the head



Hard Disk

- **Multi-Zoned Disk Drives:** Outer zones have more sectors than inner zones
  - Disk is organized into multiple zones, each of which is a consecutive set of tracks on a surface
  - Each zone has the same number of sectors per track
  - Modern disks support logical block addressing (without regard to the disk geometry)



Hard Disk

- **Cache (Track Buffer):** Some small amount of memory (8 to 16 MB) which the drive can use to hold data read from or written to the disk
  - When reading a sector, reading in all sectors on that track
  - On write, acknowledge the write when it has put the data in its memory (write back), or after the write has actually been written to disk (write through)

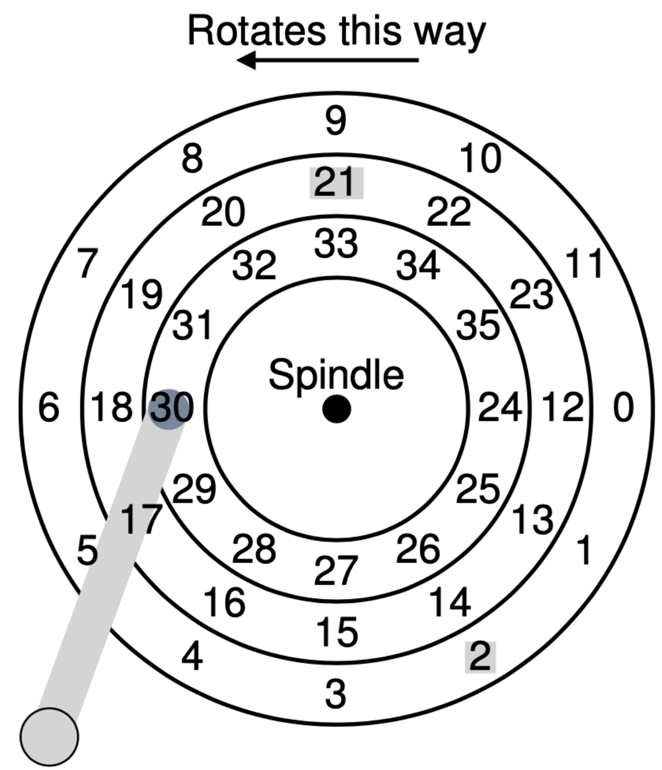
Disk Scheduling

- Because of the high cost of I/O, the OS has historically played a role in deciding the order of I/Os issued to the disk
- Given a set of I/O requests, the **disk scheduler** examines the requests and decides which one to schedule next
  - Simply, **First Come First Service (FCFS)**
    - Process disk requests in the order they are received
  - Instead, try to follow the principle of **shortest job first**
    - Unlike job scheduling, we can make a good guess at how long a disk request will take

Shortest Seek Time First (SSTF)

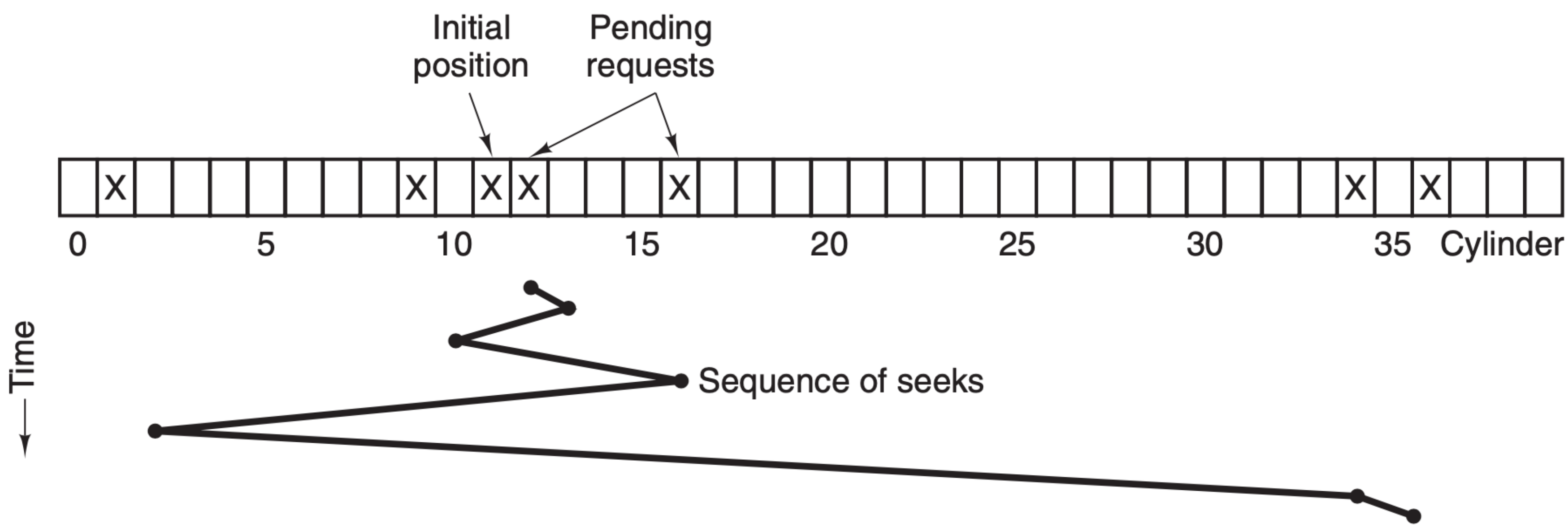
Order the queue of I/O requests by track, picking requests on the **nearest track** to complete first

- **Minimize the seek time**
- Assume the current head position is over the inner track, then issue request 21 (middle) → 2 (outer)



Shortest Seek Time First (SSTF)

- Assume the first request is for track (cylinder) 11, and new requests come in for tracks 1, 36, 16, 34, 9, and 12, in order
  - **First Come First Service**: arm motions of 10 + 35 + 20 + 18 + 25 + 3 = 111 tracks
  - **Shortest Seek Time First**: 1 + 3 + 7 + 15 + 33 + 2 = 61 tracks



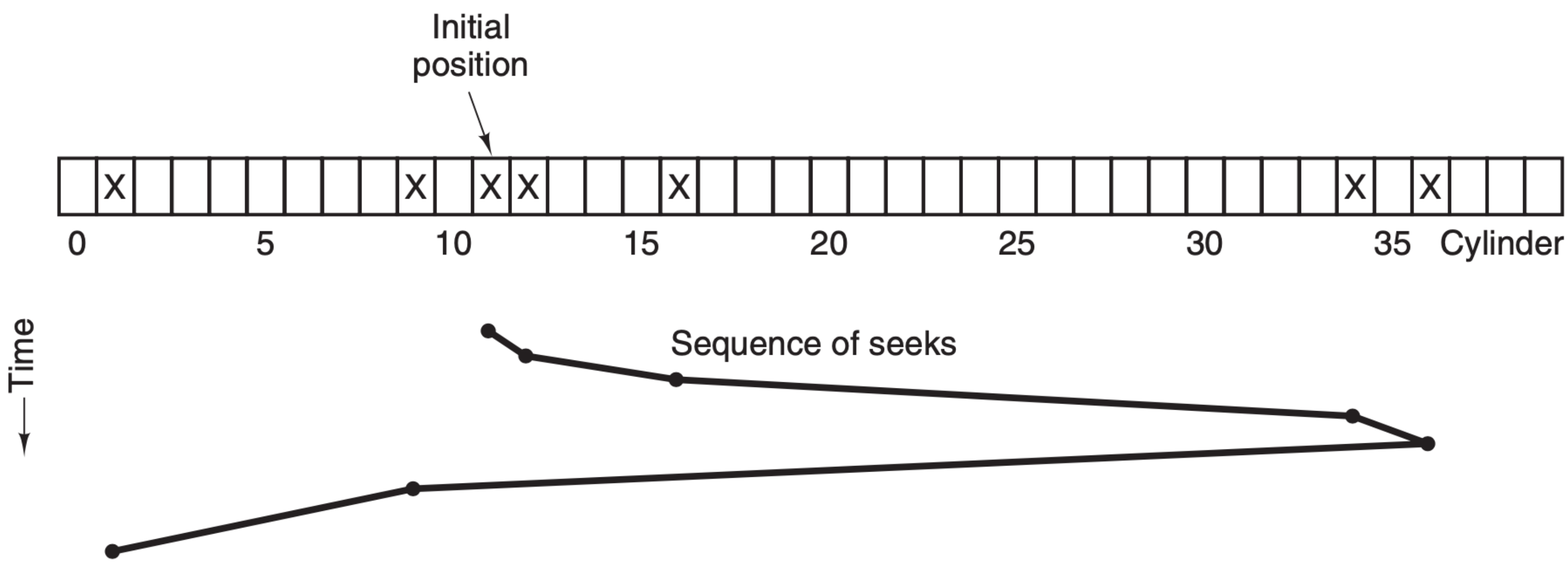
Shortest Seek Time First (SSTF)

- Assume the first request is for track (cylinder) 11, and new requests come in for tracks 1, 36, 16, 34, 9, and 12, in order
  - **First Come First Service**: arm motions of 10 + 35 + 20 + 18 + 25 + 3 = 111 tracks
  - **Shortest Seek Time First**: 1 + 3 + 7 + 15 + 33 + 2 = 61 tracks
- **Starvation**: what if there is a steady stream of requests to the middle tracks?

Elevator (SCAN)

Simply **move back and forth** across the disk servicing requests in order across the tracks (behaves like an elevator)

- A single pass across the disk (from outer to inner tracks, or inner to outer) is called a *sweep*
- Arm motions of 1 + 4 + 18 + 2 + 27 + 8 = 60 tracks



Elevator (SCAN)

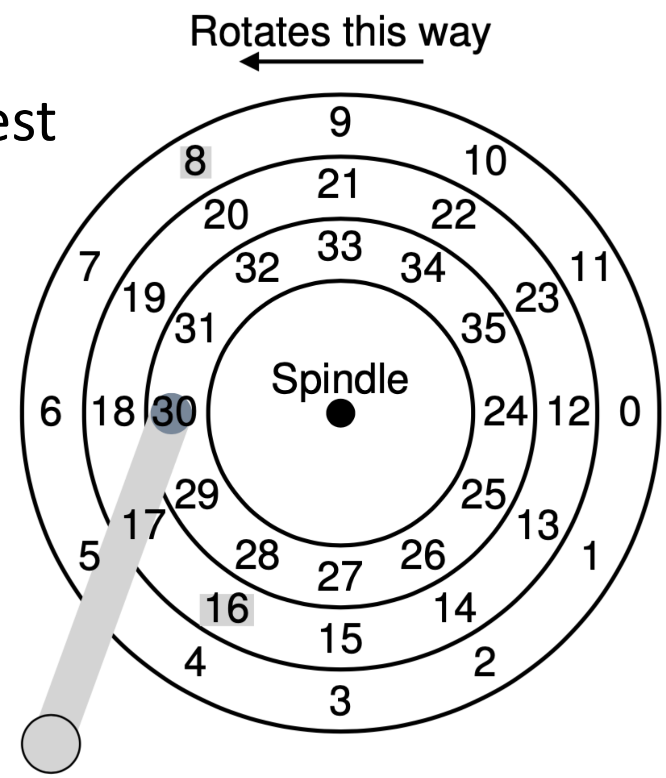
Simply **move back and forth** across the disk servicing requests in order across the tracks (behaves like an elevator)

- A single pass across the disk (from outer to inner tracks, or inner to outer) is called a *sweep*
- A number of variants available
  - *F-SCAN*: freeze the queue to be serviced when it is doing a sweep (avoid starvation of far-away requests)
  - *C-SCAN*: only sweeps from outer-to-inner, and then resets at the outer track to begin again (a bit more fair to inner and outer tracks)

Shortest Positioning Time First (SPTF)

Take both seek and rotation time into account

- Assume the current head position is over sector 30 (inner), then issue request 16 (middle), or 8 (outer) ?
  - If seek time is much higher than rotational delay, then SSF is good
  - If seek is quite a bit faster than rotation, then service 8 first
- Difficult to implement in OS, which generally does not have a good idea where track boundaries are or where the disk head currently is



Disk Scheduling

- Where is disk scheduling performed on modern systems?
  - In older systems, OS did all the scheduling
  - In modern systems, disks accommodate multiple requests, and have sophisticated internal schedulers (like SPTF)
    - OS usually picks what it thinks the best few requests are (say 16) and issues them all to disk
    - Wait for a bit, then a new and "better" request may arrive
    - Some requests can be merged (e.g., merge requests for blocks 33 and 34 into a single two-block request)

RAID

Redundant Array of Inexpensive Disks (RAID): use multiple disks in concert to build a ***bigger***, ***faster*** and ***more reliable*** disk system

- **Capacity**: Large data sets demand large disks
- **Performance**: Greatly speed up I/O times
- **Reliability**: Spread data across multiple disks makes the data vulnerable to the loss of a single disk (redundancy)
- RAIDs provide these advantages transparently to file systems that use them
  - A RAID just looks like a single disk (an array of blocks)

RAID

Redundant Array of Inexpensive Disks (RAID): use multiple disks in concert to build a ***bigger***, ***faster*** and ***more reliable*** disk system

- RAID level 0: Striping
- RAID level 1: Mirroring
- RAID level 4: Saving Space With Parity
- RAID level 5: Rotating Parity

RAID-0: Striping

Spread the blocks across the disks in a round-robin fashion

- Blocks in the same row are referred to as a **stripe**
- Extract the most parallelism when requests are made for contiguous chunks

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

a 4-disk array

RAID-0: Striping

- **Capacity**: Given N disks each of size B blocks, deliver $N * B$ blocks of useful capacity
- **Reliability**: Any disk failure will lead to data loss
- **Performance**
  - For sequential and random workloads, $R_{I/O} = N * X$ MB/s (as all disks are utilized in parallel)

RAID-1: Mirroring

Make more than one copy of each block in the system (each copy should be placed on a separate disk)

- Tolerate disk failures
- Read either copy; On write, must update both copies (can take place in parallel)

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

keep two physical copies of each logical block

RAID-1: Mirroring

- **Capacity**: With mirroring level = 2, useful capacity is $(N * B) / 2$
- **Reliability**: Tolerate 1 disk failure for certain, and up to $N / 2$ failures depending on which disks fail (e.g., disks 1 and 3)
- **Performance:** Fine for reads and poor for writes
  - Each logical write must result in two physical writes, $R_{I/O} = (N / 2) * X$ MB/s
  - Random read is the best case, $R_{I/O} = N * X$ MB/s
  - For sequential read, disk is rotating over skipped blocks, $R_{I/O} = (N / 2) * X$ MB/s

RAID-4: Saving Space With Parity

For each stripe of data, add a single **parity** block that stores the redundant information for that stripe of blocks

- Withstand the loss of any one block from the stripe

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

RAID-4: Saving Space With Parity

- Use the XOR function: the number of 1s in any row, including the parity bit, must be an even number
- To reconstruct the lost value: XOR the data bits and the parity bits together

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1) = 0
0	1	0	0	XOR(0,1,0,0) = 1

Block0	Block1	Block2	Block3	Parity
00	10	11	10	11
10	01	00	01	10

blocks of 4 bits size

RAID-4: Saving Space With Parity

- Use the XOR function: the number of 1s in any row, including the parity bit, must be an even number
- To reconstruct the lost value: XOR the data bits and the parity bits together
- When overwriting a bit, compare the old data and the new data; if they are different, then flip the old parity bit (perform based on each block)

$$P_{new} = (C_{old} \oplus C_{new}) \oplus P_{old}$$

RAID-4: Saving Space With Parity

- **Capacity**: Use 1 disk for parity information, useful capacity is *(N - 1) * B* blocks
- **Reliability**: Tolerate 1 disk failure
- **Performance**
  - Sequential read and write (full-stripe write) can utilize all of the disks except for the parity disk, $R_{I/O} = (N - 1) * X$ MB/s
  - Random read can be spread across the data disks, $R_{I/O} = (N - 1) * X$ MB/s
  - Random write ?

RAID-4: Saving Space With Parity

- Small write problem, $R_{I/O} = X / 2$ MB/s
  - Need to read and write block from disk and parity block
  - Operations at disks 0 and 1 can happen in parallel, while both have to go through the parity disk 4 (the bottleneck)
  - Each write will generate 2 physical I/Os at parity disk

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
*4	5	6	7	+P1
8	9	10	11	P2
12	*13	14	15	+P3

RAID-5: Rotating Parity

Parity and data distributed across all disks

- Remove the parity-disk bottleneck

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

RAID-5: Rotating Parity

- **Capacity**: *(N - 1) * B* blocks (as RAID-4)
- **Reliability**: Tolerate 1 disk failure (as RAID-4)
- **Performance**
  - Sequential read and write, $R_{I/O} = (N - 1) * X$ MB/s (as RAID-4)
  - Random read is little better as all disks can be utilized, $R_{I/O} = N * X$ MB/s
  - Random write is much better, $R_{I/O} = (X/2) * (N/2)$ MB/s
    - As in RAID-4, writes involve two operations at every disk
    - Can issue up to *N/2* writes in parallel (each involving 2 disks)

RAID

The tradeoffs across RAID levels

- Strictly want performance and do not care about reliability
  - RAID-0 (striping)
- Want random I/O performance and reliability
  - RAID-1 (mirroring)
- Capacity and reliability are the main goals
  - RAID-5 (rotating parity)