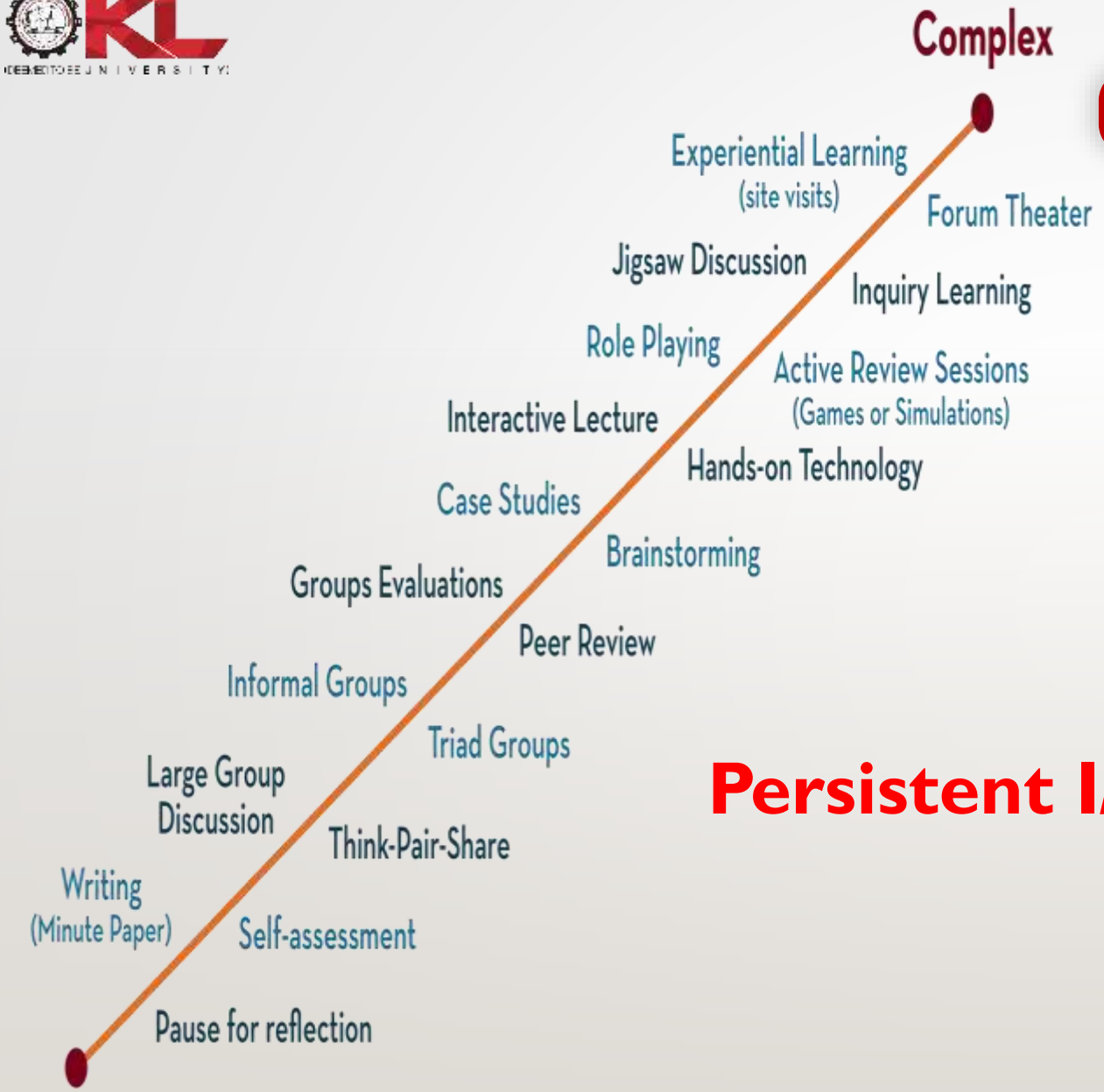


CO4

Session - 1

COURSE NAME: OPERATING SYSTEMS

COURSE CODE: 23CS2104R/A

Persistent I/O Devices, Device Management Techniques

AIM OF THE SESSION

To familiarize students with the basic concept of Threads.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate what is meant by I/O Devices.
2. Demonstrate what is meant by Interrupts and Polling.
3. Describe the types of Device Interaction.
4. Describe the Cananicol Devices

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Defines what Interrupts and Polling.
2. Describe Device Interaction.
3. Summarize the Role of DMA.

TECHNIQUES FOR DEVICE MANAGEMENT

Three major techniques are used for managing and allocating devices:

- **Dedicated**
- **Shared**
- **Virtual**.

DEDICATED DEVICES

- Such devices in the operating system's device management are dedicated or assigned to only one job at a time until that job releases them.
- Devices like printers, tape drivers, plotters etc. demand such allocation scheme since it would be awkward if several users share them at the same point of time.
- The disadvantage is the inefficiency resulting from the allocation of the device to a single user for the entire duration of job execution even though the device is not used 100% of the time.

SHARED DEVICES

- Some devices such as disks, drums, and most other Direct Access Storage Devices (DASD) may be shared concurrently by several processes.
- Essentially, several processes can read from a single disk at the same time.
- Process request is to be satisfied first based on (1) a priority list or (2) the objective of achieving improved system output.

VIRTUAL DEVICES

- These devices are a combination of the first two types and they are dedicated devices which are transformed into shared devices.
- For example, a printer converted into a shareable device via a spooling program which re-routes all the print requests to a disk. A print job is not sent straight to the printer, instead, it goes to the disk(spool)until it is fully prepared with all the necessary sequences and formatting, and then it goes to the printers. This technique can transform one printer into several virtual printers which leads to better performance and use.

DEVICE CHARACTERISTICS

Hardware considerations

- Input or output devices
- Storage devices

I/O DEVICES

The three main jobs of a computer are Input, Output, and Processing.

- In most of the cases, the most important job is Input / Output, and the processing is simply incidental.
- For an example, when we browse a web page or edit any file, our immediate attention is to read or enter some information, not for computing an answer.
- The fundamental role of the operating system in computer Input / Output is to manage and organize I/O operations and all I/O devices.
- The various devices that are connected to the computer need to be controlled and it is a key concern of operating-system designers.

I/O HARDWARE

- Computers operate many kinds of devices. General categories of devices are input or output devices and storage devices.
- A device that communicates with the operating system of a computer by transferring signals over cable or even through the air.
- The Peripheral devices that communicate with the machine through a connection point also called ports- (one example is a serial port).
- A bus is a collection of wires and a firmly defined protocol that describes a set of messages that can be sent on the wires.
- An I/O port usually contains four different registers −

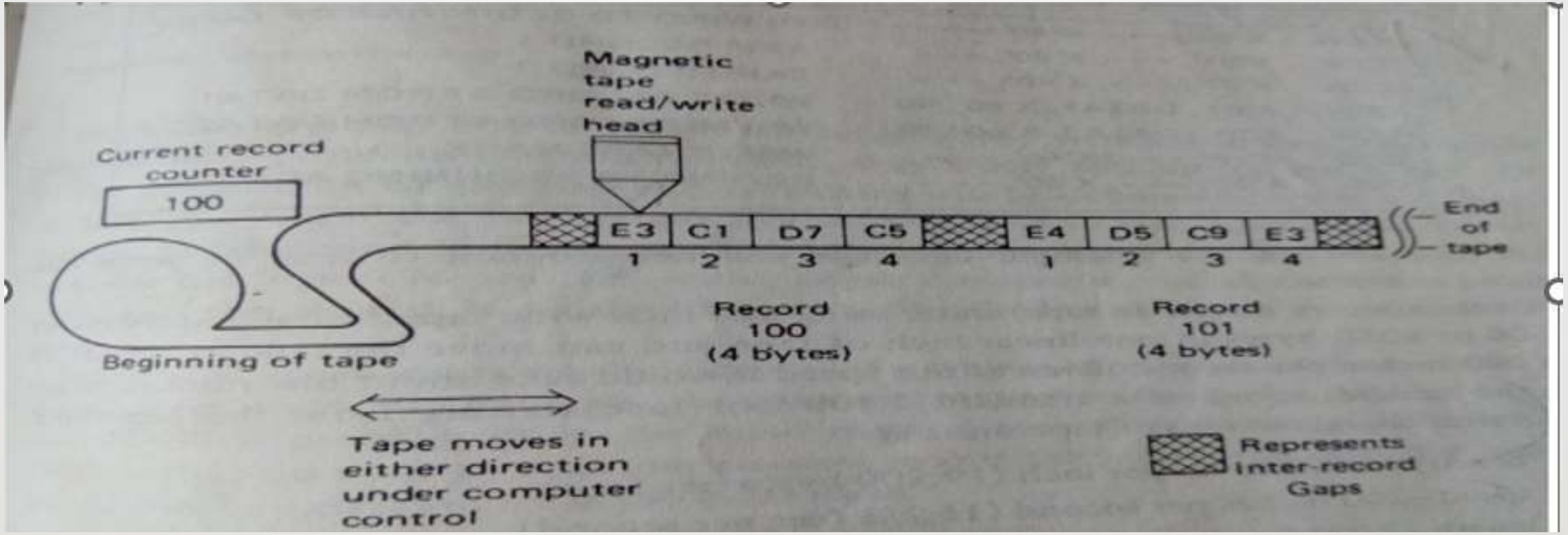
  (1) control, (2) status, (3) data-in, and (4) data-out registers

STORAGE DEVICES

- Magnetic Disks / Serial Access Devices
- Magnetic Drums / Completely Or Random Access Devices
- Optical Disks / DASD – Direct Access Storage Devices

MAGNETIC DISKS / SERIAL ACCESS DEVICES

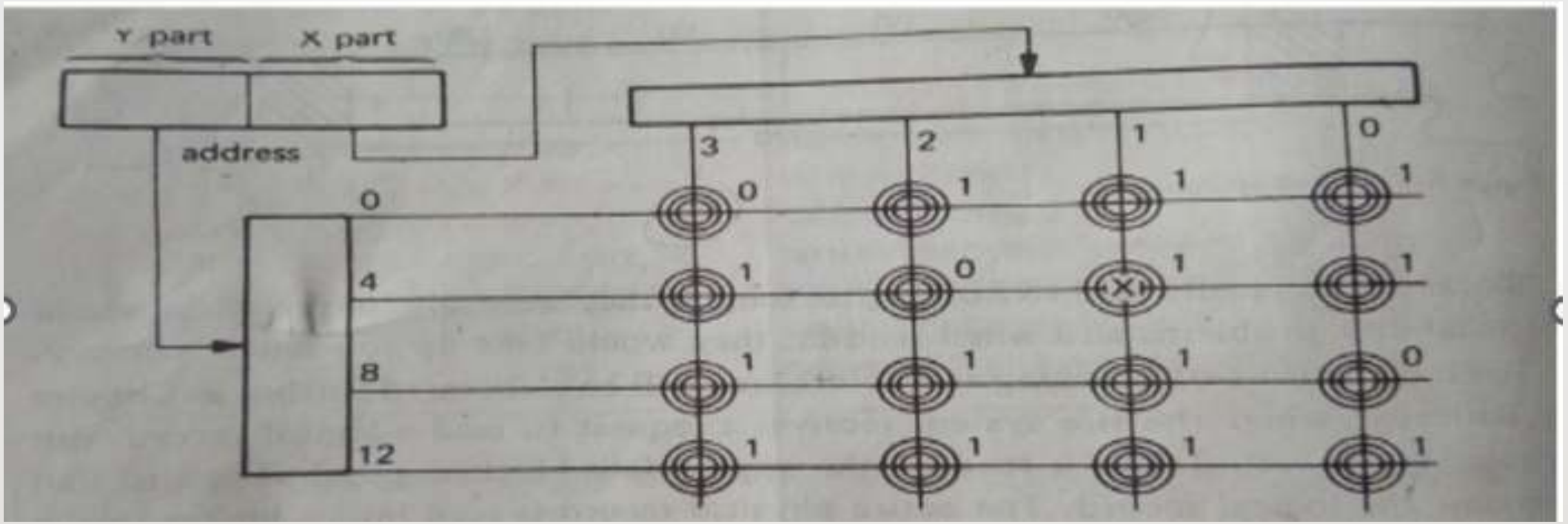
- It is a sequential access storage device.
- Magnetic disks provides a bulk of secondary storage in modern computer systems.
- Each disk platter has a flat circular shape like a CD.
- Common platter size ranges from 1.8 to 3.5 inches.
- The two surfaces of a platter are covered with a magnetic material.
- We store information by recording it magnetically on the platters.



Several kinds of buses are available like
ATA - Advanced Technology Attachment
SATA – Serial ATA
USB – Universal Serial Bus
FC – Fiber Channel

DIRECT ACCESS DEVICES

- It is a Direct access, or random-access, storage device.
- A magnetic drum, also referred to as drum, is a metal cylinder coated with magnetic iron-oxide material on which data and programs can be stored.
- Magnetic drums were once used as a primary storage device but have since been implemented as auxiliary storage devices.
- The tracks on a magnetic drum are assigned to channels located around the circumference of the drum, forming adjacent circular bands that wind around the drum.
- A single drum can have up to 200 tracks.



In Figure the circles represent magnetic each core having two selection wires through it.One property of a magnetic core is that when it changes magnetic state it induces a current. A third sensing wire is passed through all the cores.

DIRECT ACCESS STORAGE DEVICES

- A device access device is one that is characterized by small variances in the access time Tij.
- These have been called Direct Access Storage Devices. Although they do not offer completely direct access, the name persists in the field.
- Two examples of DASD are fixed-head and moving-head devices.

Fixed-Head Drums and Disks

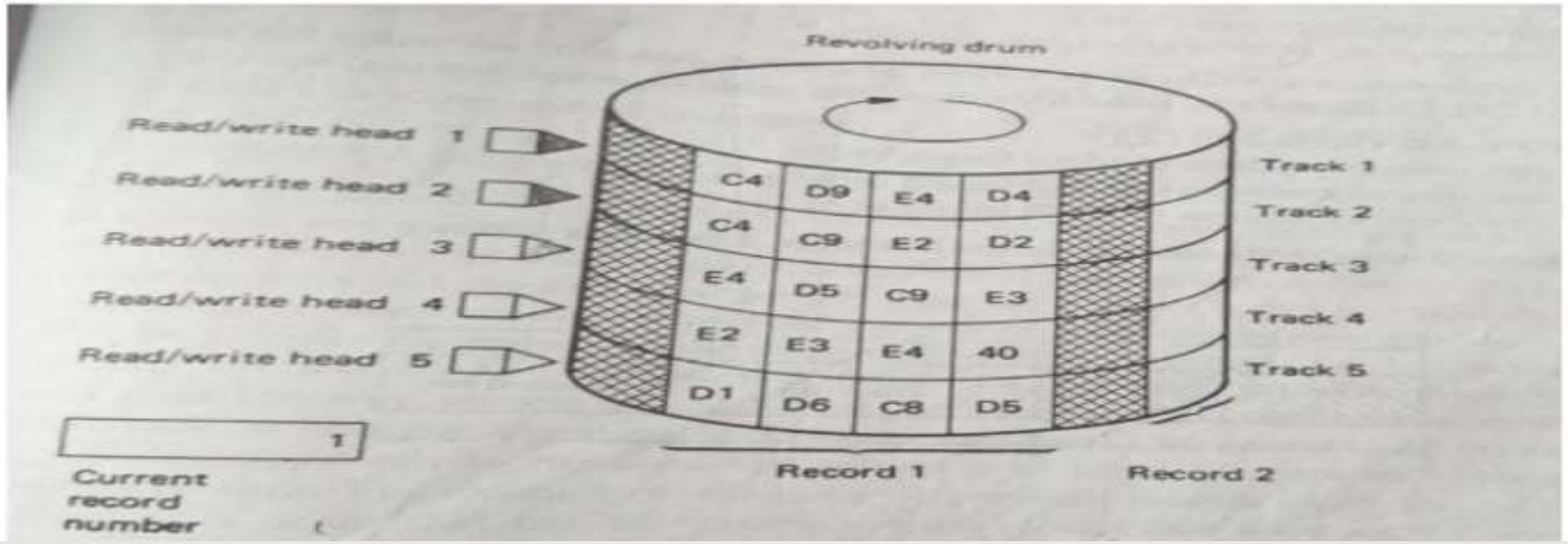
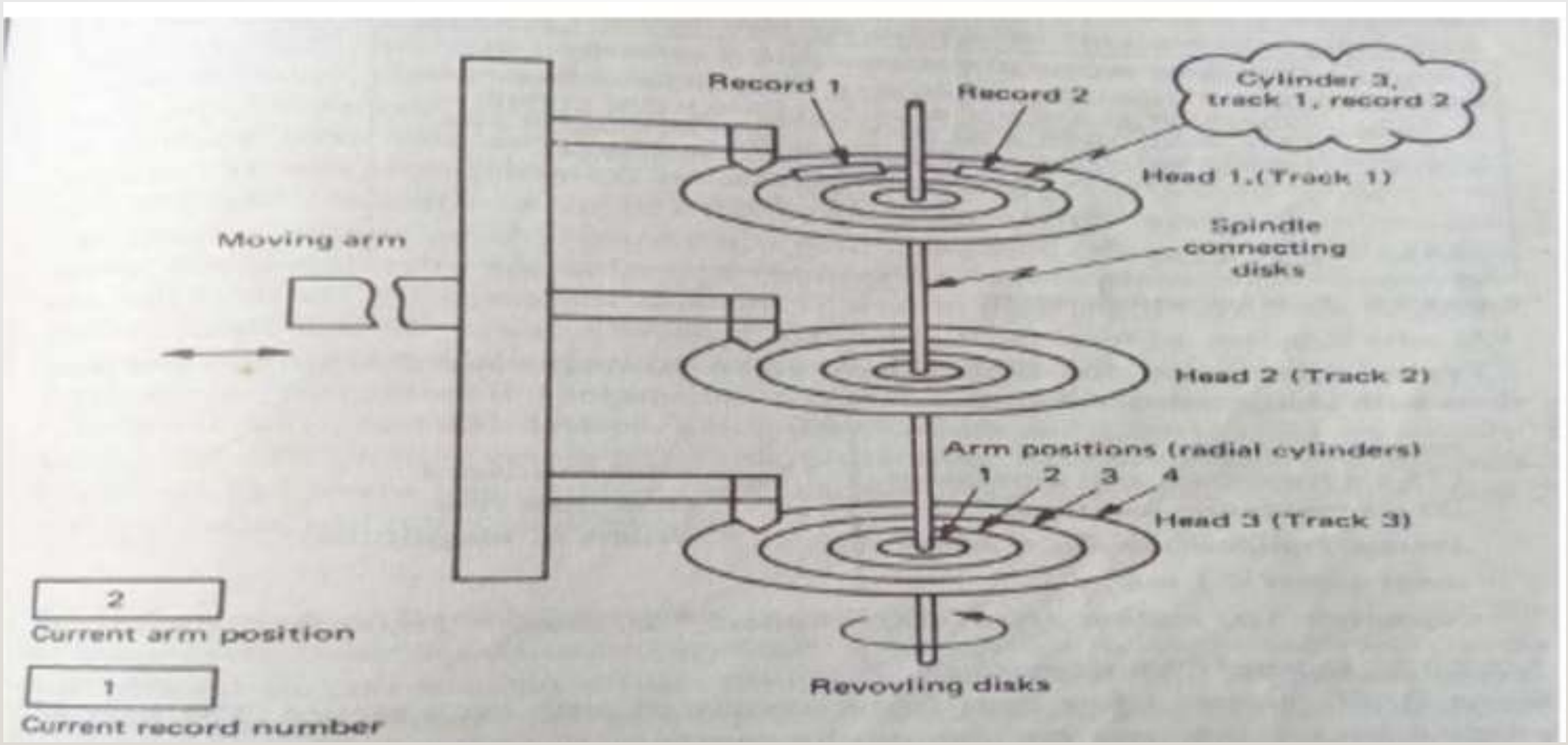


Figure depicts a DASD storage device similar to a magnetic drum. A magnetic drum can be simplistically viewed as several adjacent strips of magnetic tape wrapped around a drum so that the ends of each tape strip join. Each tape strip, called a track, has a separate head/write head.

MOVING-HEAD DISKS AND DRUMS

- Both drums and fixed head disks are expensive because of the requirement of one head (and associated electronics) per recording track.
- Moving head disks vary considerably in performance, from low-speed, low-capacity floppy disks, with only one or two recording surfaces, to high speed units with multiple disk platters all mounted on a common recording surface.
- The original moving head disk was developed under the direction of Reynold B. Johnson at IBM's lab.


Record 1
Record 2
Cylinder 3, track 1, record 2
Head 1 (Track 1)
Moving arm
Spindle connecting disks
Head 2 (Track 2)
Arm positions (radial cylinders)
1 2 3 4
Head 3 (Track 3)
2
Current arm position
1
Current record number
Revovling disks

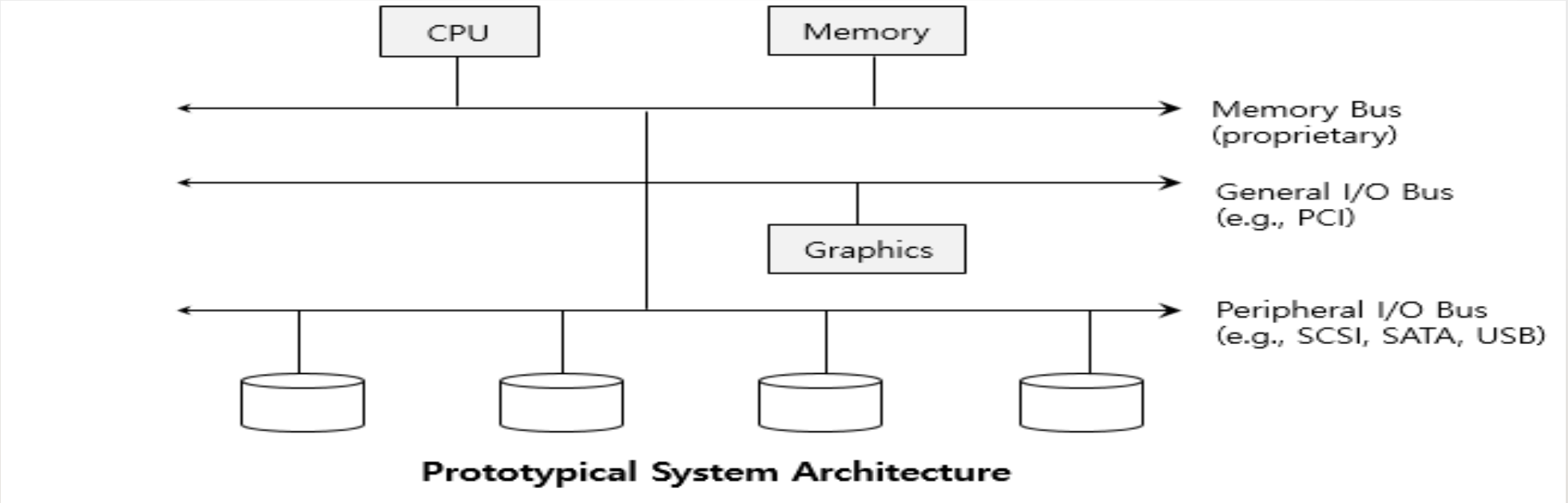
OPTICAL DISKS

- Optical disk is an electronic storage medium that can be written to or read from a low powered laser beam.
- It was developed by James T. Russel in the year 1960.
- It stores data as micron wide dots of light and dark.
- First, optical storage system uses a powerful backlight to read the dots through a transparent sheet of material on which the dots were encoded.
- In later optical systems, the laser read the dots and data was converted to an electrical signal.
- An optical disk could store much more data than magnetic disk.

I/O DEVICES

- I/O is **critical** to computer systems to **interact with systems.**
- I/O is quite critical to computer systems. imagine a program without any input (it produces the same result each time); now imagine a program with no output (what was the purpose of it running?). For computer systems to be interesting, both input and output are required.
- Issue :
  - How should I/O be integrated into systems?
  - What are the general mechanisms?
  - How can we make it efficient?

STRUCTURE OF INPUT/OUTPUT (I/O) DEVICE



Prototypical System Architecture

CPU is attached to the system's main memory via some kind of memory bus. Some devices(higher-performance I/O devices are connected to the system via a general I/O bus and a peripheral bus connects the slowest devices to the system, including disks, and mice.

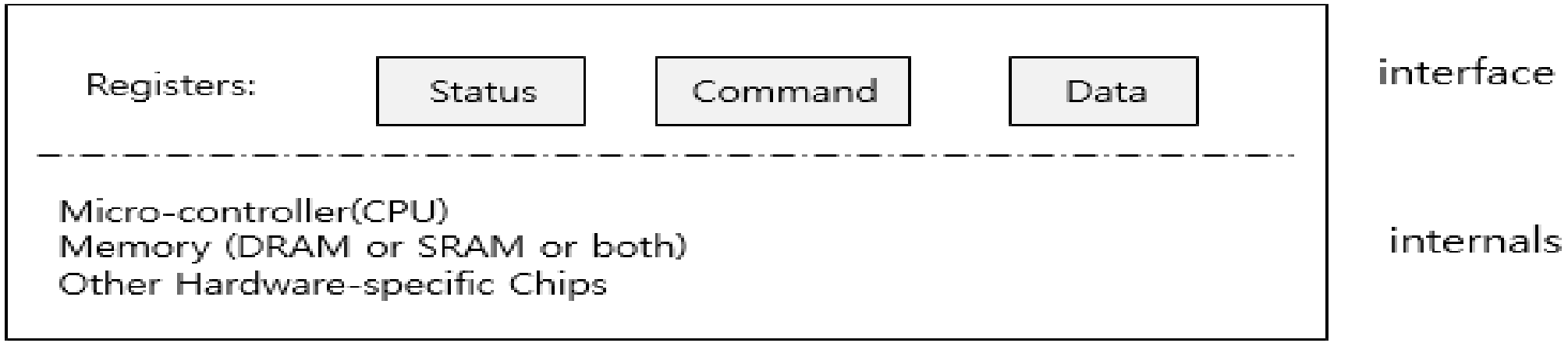
I/O ARCHITECTURE

- Buses
  - Data paths that provided to enable information between CPU(s), RAM, and I/O devices.

- I/O bus
  - Data path that connects a CPU to an I/O device.
  - I/O bus is connected to I/O device by three hardware components: I/O ports, interfaces and device controllers.

CANONICAL DEVICE

- canonical device (not a real one), and use this device to drive our understanding of some of the machinery required to make device interaction efficient.
- Canonical Devices have two important components.
  - **Hardware interface** allows the system software to control its operation.
  - **Internals** which is implementation-specific.



Canonical Device

HARDWARE INTERFACE OF CANONICAL DEVICE

- **status register**
  - 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
- By reading and writing these registers, the operating system can control device behavior.

HARDWARE INTERFACE OF CANONICAL DEVICE (CONT.)

Typical interaction example that OS might have with the device in order to get the device to do something on its behalf. The protocol is as follows:

```
while ( STATUS == BUSY) ; //wait until the device is not busy

Write data to the data register

Write a command to command register

    Doing so starts the device and executes the command

While ( STATUS == BUSY);//wait until the device is done with your
request
```

PROTOCOL OF CANONICAL DEVICE

- **The protocol has four steps.**
- In the first, the OS waits until the device is ready to receive a command by repeatedly reading the status register; we call this **polling the device.**
- Second, the OS sends some data down to the data register; When the main CPU is involved with the data movement we refer to it as **programmed I/O (PIO).**
- Third, the OS writes a command to the command register; doing so implicitly lets the device know that both the data is present and that it should begin working on the command.
- Finally, the OS waits for the device to finish by again polling it in a loop, waiting to see if it is finished

POLLING

- The operating system waits until the device is ready by **repeatedly** reading the status register.
  - Positive aspect is simple and working.
  - **However, it wastes CPU time just waiting for the device**.
    - Switching to another ready process is better utilizing the CPU.

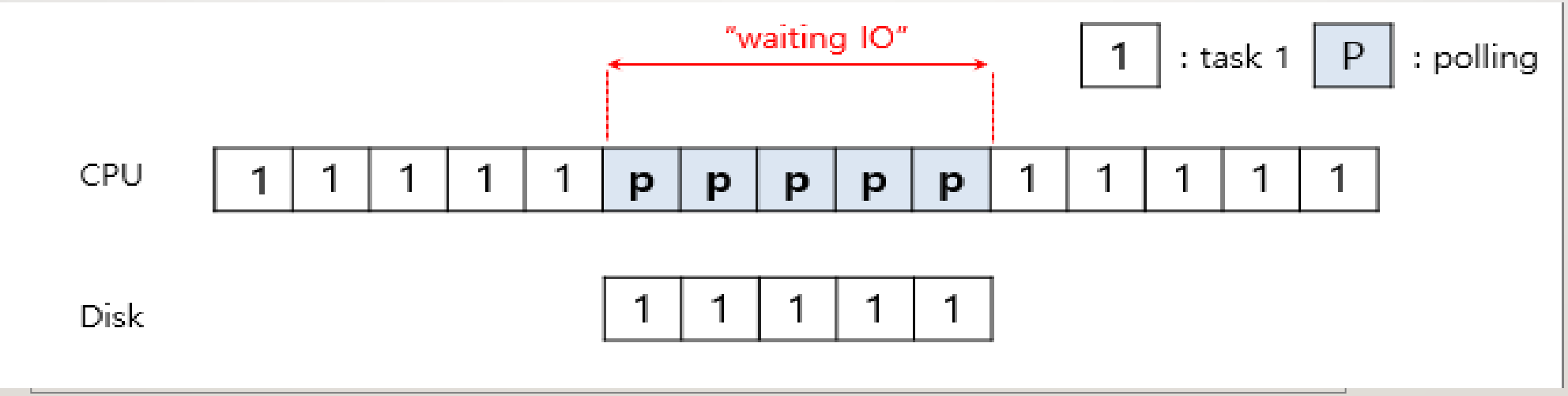


Diagram of CPU utilization by polling

INTERRUPTS

- **Put the I/O request process to sleep** and context switch to another.
- When the device is finished, wake the process waiting for the I/O by **interrupt**.
  - The positive aspect is allowed to the **CPU and the disk is properly utilized.**

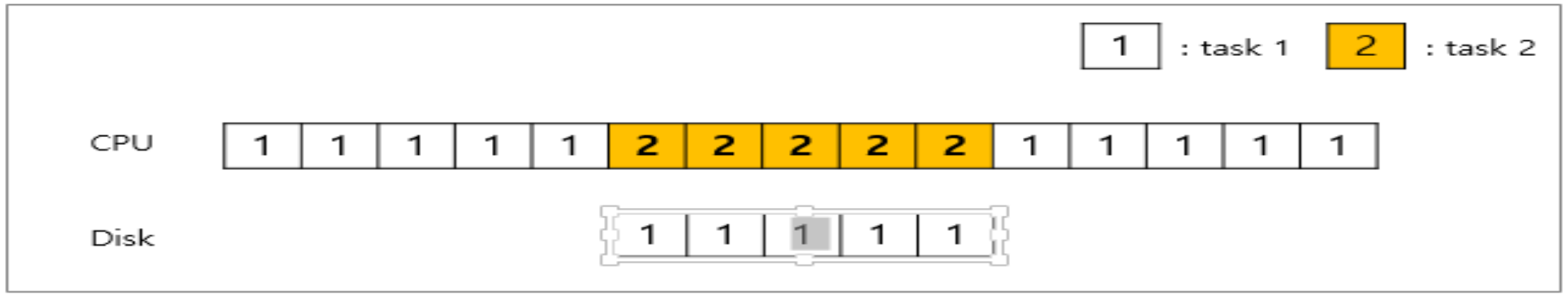


Diagram of CPU utilization by interrupt

POLLING VS INTERRUPTS

- *However,* **"interrupts is not always the best solution"**
  - If the device performs very quickly, an interrupt will "slow down" the system.
  - Because **context switch is expensive (switching to another process)**

If a device is fast → poll is best.
If it is slow → interrupts is better.

CPU IS ONCE AGAIN OVER-BURDENED

- CPU **wastes a lot of time** to copy the *a large chunk of data* from memory to the device.

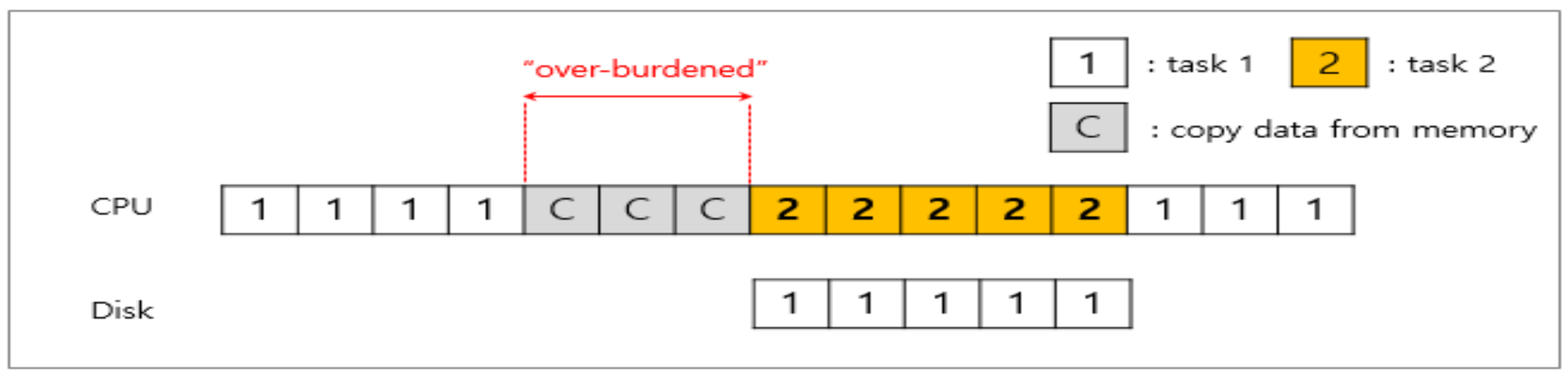


Diagram of CPU utilization

DMA (DIRECT MEMORY ACCESS)

- OS would program the data engine by telling it "where the data lives in memory, how much data to copy and which device to send it to."
- When completed, DMA raises an interrupt, I/O begins on Disk.

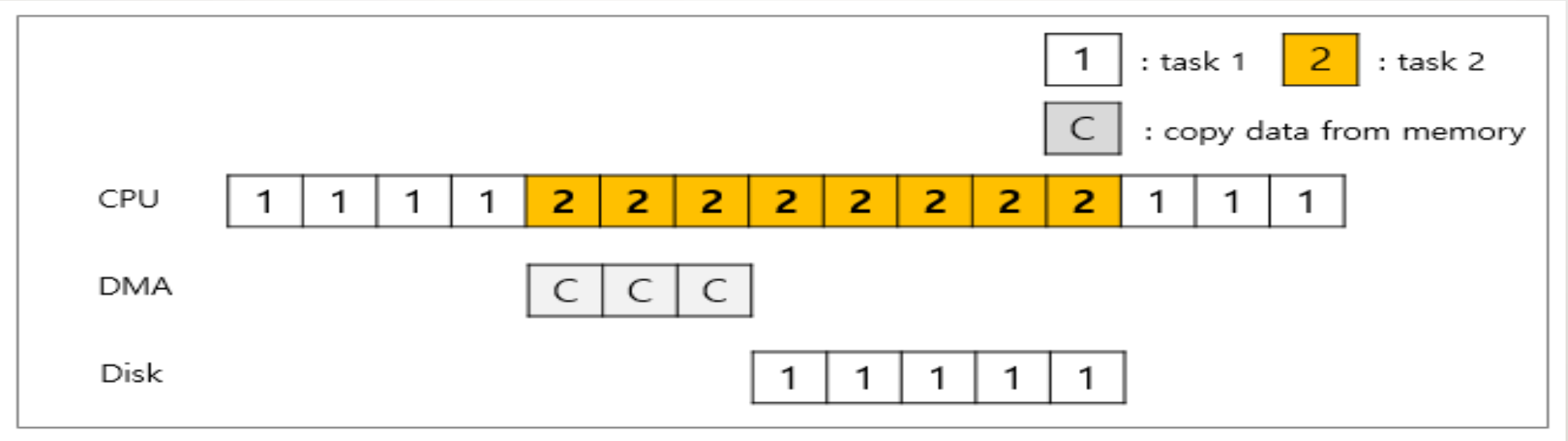


Diagram of CPU utilization by DMA

METHODS OF DEVICE INTERACTION

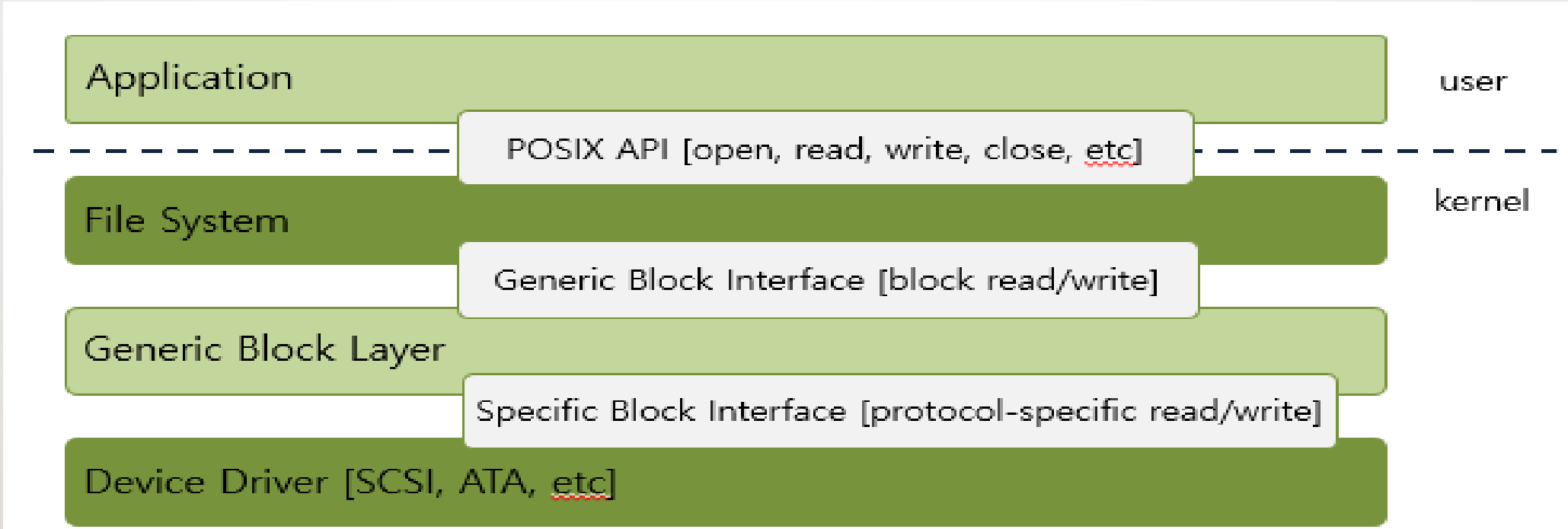
- How the OS communicate with the **device**?
- Solutions
  - I/O instructions: a way for the OS to send data to specific device registers.
    - Ex) in and out instructions on x86
  - memory-mapped I/O
    - Hardware makes Device registers available as if they were memory locations.
    - To access 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 (CONT.)

- How the OS interact with **different specific interfaces**?
  - Ex) We'd like to build a file system that worked on top of SCSI disks, IDE disks, USB keychain drivers, and so on.
- Solutions: Abstraction
  - Abstraction encapsulates any specifics of device interaction.

FILE SYSTEM ABSTRACTION

- File system specifics of which disk class it is using.
  - Ex) It issues **block read** and **write** request to the generic block layer.



The File System Stack

PROBLEM OF FILE SYSTEM ABSTRACTION

- If there is a device having many special capabilities, these capabilities will go unused in the generic interface layer.

- Over 70% of OS code is found in device drivers.
  - Any device drivers are needed because you might plug it to your system.
  - They are primary contributor to **kernel crashes**, making **more bugs**.

A SIMPLE IDE DISK DRIVER

- Four types of register
  - Control, command block, status, and error

  Memory-mapped IO
    - in and out I/O instruction

A SIMPLE IDE DISK DRIVER

- **Control Register:**

Address 0x3F6 = 0x80 (0000 1RE0): R=reset, E=0 means "enable interrupt"

- **Command Block Registers:**

Address 0x1F0 = Data Port

Address 0x1F1 = Error

Address 0x1F2 = Sector Count

Address 0x1F3 = LBA low byte

Address 0x1F4 = LBA mid byte

Address 0x1F5 = LBA hi byte

Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive

Address 0x1F7 = Command/status

A SIMPLE IDE DISK DRIVER

- **Status Register (Address 0x1F7):**

7	6	5	4	3	2	1	0
BUSY	READY	FAULT	SEEK	DRQ	CORR	IDDEX	ERROR

- **Error Register (Address 0x1F1): (check when Status ERROR==1)**

7	6	5	4	3	2	1	0
BBK	UNC	MC	IDNF	MCR	ABRT	T0NF	AMNF

  - BBK = Bad Block
  - UNC = Uncorrectable data error
  - MC = Media Changed
  - IDNF = ID mark Not Found
  - MCR = Media Change Requested
  - ABRT = Command aborted
  - T0NF = Track 0 Not Found
  - AMNF = Address Mark Not Found

THE BASIC PROTOCOL TO INTERACT WITH THE DEVICE IS AS FOLLOWS

assuming it has already been initialized:

- **Wait for the drive to be ready**. Read Status Register (0x1F7) until the drive is not busy and READY.
- **Write parameters to command registers**. Write the sector count, logical block address (LBA) of the sectors to be accessed, and drive number (master=0x00 or slave=0x10, as IDE permits just two drives) to command registers (0x1F2-0x1F6).
- **Start the I/O**. by issuing read/write to the command register. Write READ—WRITE command to command register (0x1F7).
- **Data transfer (for writes)**: Wait until the drive status is READY and DRQ (drive request for data); write data to the data port.
- **Handle interrupts**. In the simplest case, handle an interrupt for each sector transferred; more complex approaches allow batching and thus one final interrupt when the entire transfer is complete.
- Error handling. After each operation, read the status register. If the ERROR bit is on, read the error register for details.

THE XV6 IDE DISK DRIVER (SIMPLIFIED)

```
static int ide_wait_ready() {
        while (((int r =
inb(0x1f7)) & IDE_BSY) ||
        !(r & IDE_DRDY))
        ; // loop until the
drive isn't busy
        }
```

```
void ide_intr() {
  struct buf *b;
  acquire(&ide_lock);
  if (!(b->flags & B_DIRTY) && ide_wait_ready(1) >= 0)
    insl(0x1f0, b->data, 512/4); // if READ: get data
  b->flags |= B_VALID;
  b->flags &= ~B_DIRTY;
  wakeup(b); // wake waiting process
  if ((ide_queue = b->qnext) != 0) // start next request
    ide_start_request(ide_queue); // (if one exists)
  release(&ide_lock);
}
```

```
void ide_rw(struct buf *b) {
  acquire(&ide_lock);
  for (struct buf **pp = &ide_queue; *pp;
pp=&(*pp)->qnext)
  ; // walk queue
  *pp = b; // add request to end
  if (ide_queue == b) // if q is empty
    ide_start_request(b); // send req to disk
  while ((b->flags & (B_VALID|B_DIRTY)) !=
B_VALID)
    sleep(b, &ide_lock); // wait for completion
  release(&ide_lock);
}
```

```
void ide_intr() {
  struct buf *b;
  acquire(&ide_lock);
  if (!(b->flags & B_DIRTY) && ide_wait_ready(1) >= 0)
    insl(0x1f0, b->data, 512/4); // if READ: get data
  b->flags |= B_VALID;
  b->flags &= ~B_DIRTY;
  wakeup(b); // wake waiting process
  if ((ide_queue = b->qnext) != 0) // start next request
    ide_start_request(ide_queue); // (if one exists)
  release(&ide_lock);
}
```

THANK YOU

Team - Operating System