Input-Output

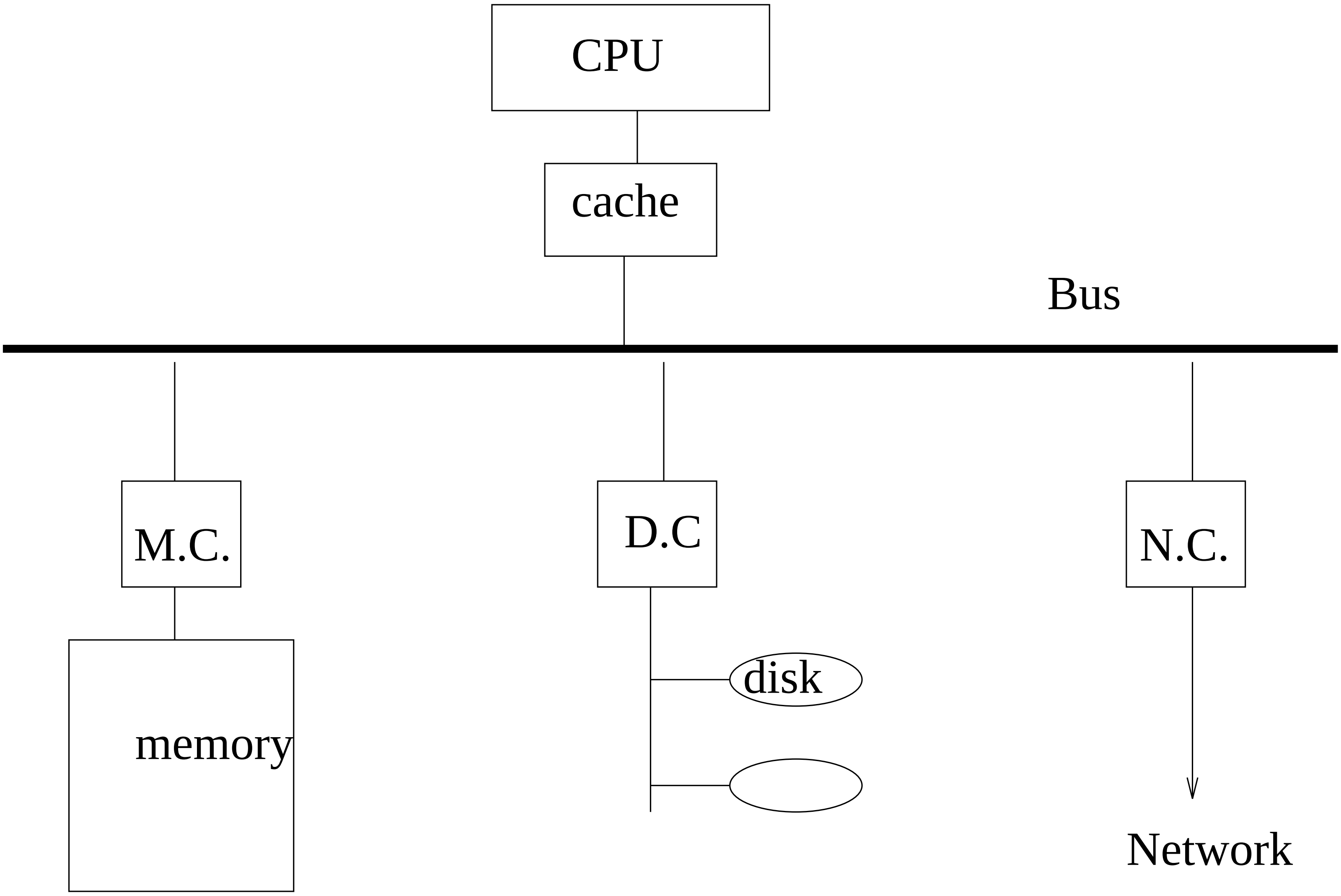
Input-Output is very much architecture dependent.

I/O requires cooperation between

- Processor that issues the I/O command
- Buses that provide the interconnection between processor, memory and I/O devices
- I/O controllers that handle the specifics of control of each device and interfacing
- Devices that store data or signal events

Basic Input-Output Architecture

This is a slight oversimplification



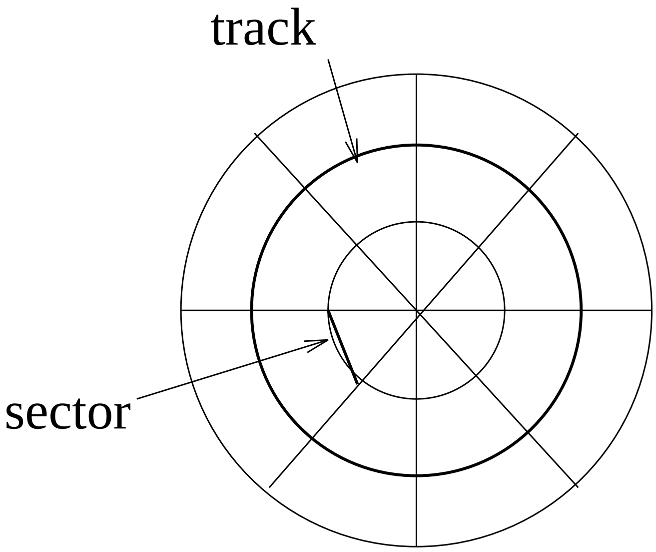
Types of I/O devices

Input devices: keyboard, mouse

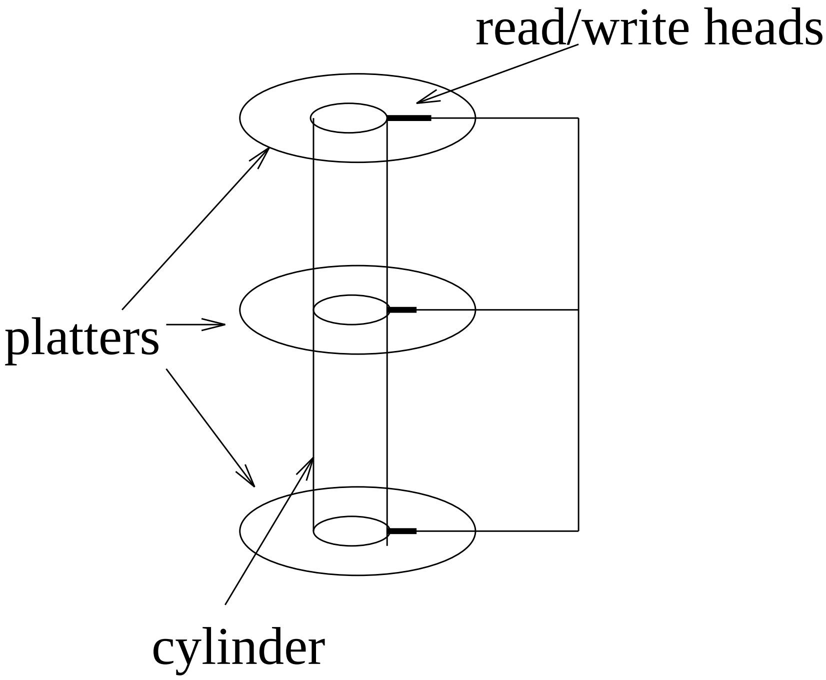
Output devices: screen, line printer

Input/output devices: disks, tapes, connections to networks

An important I/O device: the disk



Disk surface



Disk components

A disk is composed of one or more *platters* (today 10 to 20). The platters rotate (3600 to 5400 rpm and even 7200 rpm)

Each platter (diameter of 1 to 10 inches) is composed of concentric *tracks* (500 to 3000 tracks per surface; 2 surfaces/platter)

Tracks are divided into *sectors* (32 to 128 sectors/track). The sector is the smallest unit that can be read/written (by read/write heads)

A *cylinder* is the set of tracks under the read/write heads.

Some typical numbers for a small disk (2.5 inch diameter): 10 disk surfaces, 2 GB capacity

A larger disk (5.25 in diameter) 28 disk surfaces, 23 GB capacity

Disk operation and timing

To perform a disk transfer (read or write):

- The heads must be placed over the right track: *seek* operation
- Then wait until right sector is under the head: *rotational latency*
- Transfer the number of sectors dictated by the operation: *transfer time*

The overall time to perform a disk operation:

$$seek\ time + rotation\ time + transfer\ time$$

Each of these three "times" takes a few milliseconds

- Average seek time 10-20 ms
- Average rotation time (1/2 disk rotation. At 3600 rpm this yields $0.5/60 = 8.3$ ms)
- External transfer rates are of the order of 20 Mbytes/second. Thus the transfer time for a 4KB page is 0.2 ms

Also one must add the O.S. and disk controller overhead to initiate the I/O operation (about 1 ms).

Buses

Bus: shared communication link used to exchange data and control information between devices.

Example of buses:

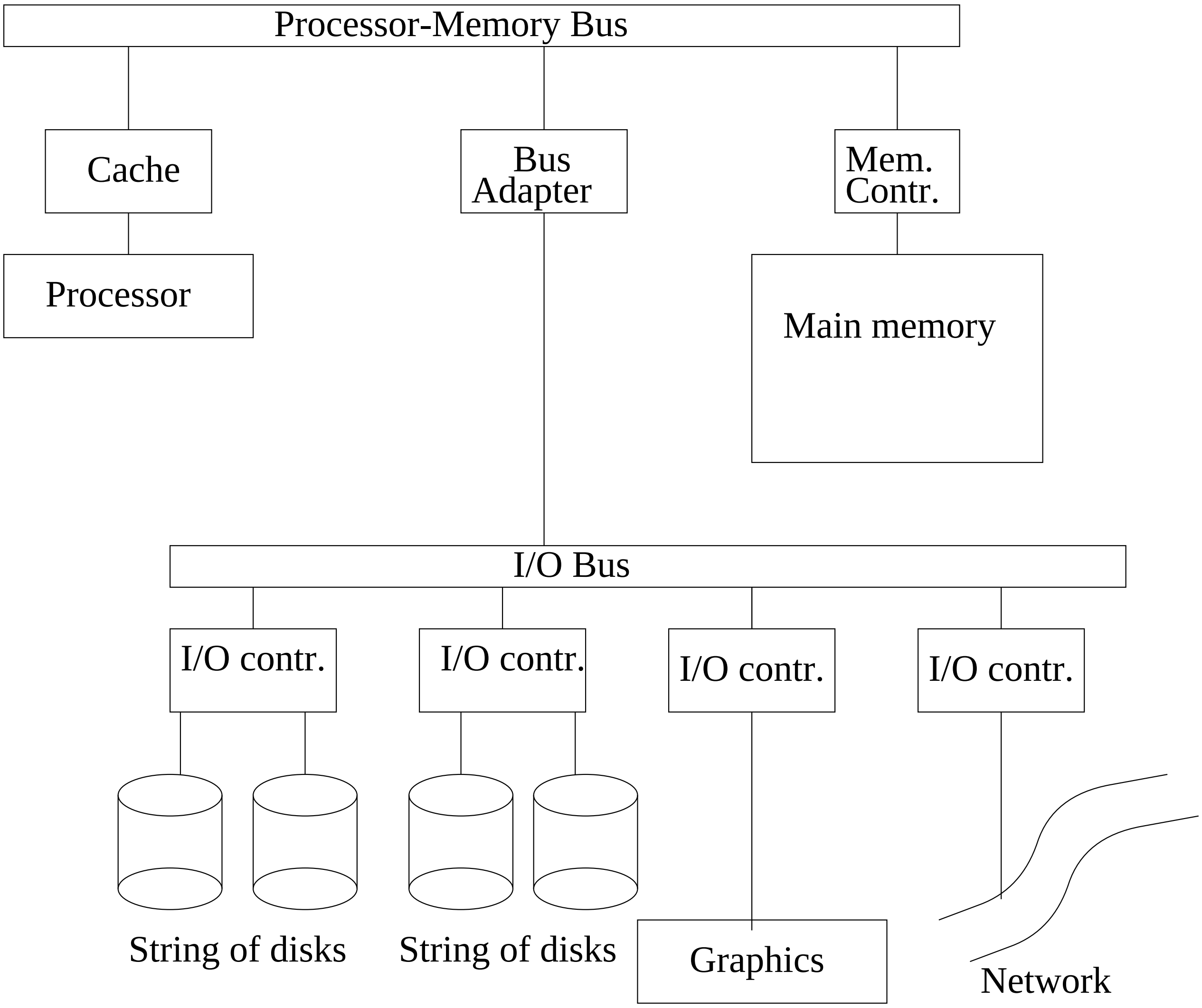
- Processor-memory (in general specific to an architecture)
- I/O buses (standardized so that several I/O devices of various manufacturers can be connected to the bus; example SCSI bus)
- Backplane buses to allow processor, memory and I/O devices to coexist on a single bus (example PCI bus)

Standard buses (I/O and backplane) must be governed by a protocol.

The protocol can be *synchronous* (governed by a clock). Only good for short buses and when an answer is expected in a short number of cycles.

Or the protocol is *asynchronous* requiring *handshaking*, i.e., agreement between sender and receiver (cf. CSE 370)

Typical Processor-memory and I/O bus configuration



Bus operation

A bus is set of wires (lines)

There are:

- address lines
- data lines
- control lines

(address and data lines can be multiplexed)

A bus transaction consists of arbitration (who is getting control of the bus) and commands (e.g., read request, acknowledgment, transfer of data)

Bus arbitration

Since several devices are connected to the bus, there might be concurrent requests to use the bus. Requesting devices are called *masters*. The processor is always a master.

If more than one master (e.g., some I/O controllers on the I/O bus) there is a need for *arbitration*.

Arbitration can be by priority (each device has a predetermined priority) or round-robin (each device in turn has highest priority).

Arbitration can be hardwired (e.g., daisy-chain), centralized (a central entity decides who wins; the problem is that the arbiter might become the bottleneck) or decentralized (arbitration is done in parallel; each device monitors the requests and determines its own priority).

Asynchronous bus protocols

Hand-shaking Protocol

For example: Master requests data from Slave. When Master has control of the bus (after arbitration)

1. Master transmits a read request (control lines) and address (on data/address lines)
2. Slave recognizes the request. Grabs address and raises the Ack control line.
3. Master sees the Ack line high. Releases the Read request and data lines.
4. Slave sees Read request is low. Releases the Ack line.
5. Slave is ready to transmit data. Places data on data lines and raises Data ready (control line)
6. Master sees data ready high. Grabs data and raises Ack.
7. Slave sees Ack high. Releases Data ready and data lines
8. Master sees Data ready low. Releases Ack.

A new transaction can begin.

In *split-transaction* buses, the Slave acts as Master after step 4 and the bus does not need to be held during the whole transaction.

I/O. Hardware-software interface

I/O is best left to the O.S. (for protection and scheduling in particular)

O.S. provides routines that handles devices (or device controllers).

But since O.S. is a program there must be instructions to generate I/O commands!

Thus the CPU must be able to:

- tell a device what it wants done (e.g., read, write, rewind)
- start the operation on the device
- find out when the operation is completed
- find out if the operation was completed without error

There is no unique way to do all these things. Depends on the ISA and the I/O architecture.

CPU-I/O Device interaction

There are two basic ways for the CPU to control I/O devices:

- specific I/O instructions
- memory mapped I/O

For specific I/O instructions, an I/O instruction specifies both the device number and a command (or an address where the I/O device can find a series of commands).

For memory mapped, I/O device controllers appear to be (reserved) memory addresses. When data is sent to a memory address corresponding to a device controller, it is ignored by the real memory. The data that is sent is interpreted by the I/O controller as a command.

In both cases, only the O.S. can execute I/O instructions or write data to memory mapped I/O.

I/O termination

There are two techniques to know when an I/O operation is over:

- Polling
- Interrupts

In polling, the CPU repeatedly checks whether a device has completed (used often for the mouse, an input-only device and rarely for most other devices in todays processors). Time consuming if the device has to be polled often.

With interrupts, the CPU initiates the I/O operation and then context-switches. When the operation is completed, the I/O device *interrupts* the CPU.

Upon termination, there is always a check for possible errors. Each device controller has a status register to indicate potential errors.

I/O interrupts

An interrupt is like an exception (e.g., page fault, divide by zero) except that the stimulus for the interrupt is outside the program currently executing.

Like for an exception, upon an interrupt the O.S. takes over (implies context-switch, saving of the state etc.)

There are two schemes for handling interrupts:

- *vector interrupts* where the O.S. is told where to go for a specific interrupt
- *cause register* where the O.S. (interrupt handler) can examine the cause of the interrupt and transfer to the appropriate routine from there.

Data transfer

Data can be transferred one of two ways:

- Use the processor to transfer data from the device to (from) memory.

  Drawback: slow operation. Can be done with either polling (*programmed I/O operations*) or interrupts.

- Direct-memory access (DMA)

In DMA, operations are initiated by the CPU but data is transferred directly to/from memory. Upon completion, CPU is interrupted.

For a DMA transfer, e.g., from memory to disk, the CPU must give the address in memory of where the data comes from, the number of bytes, and the address on the output device.