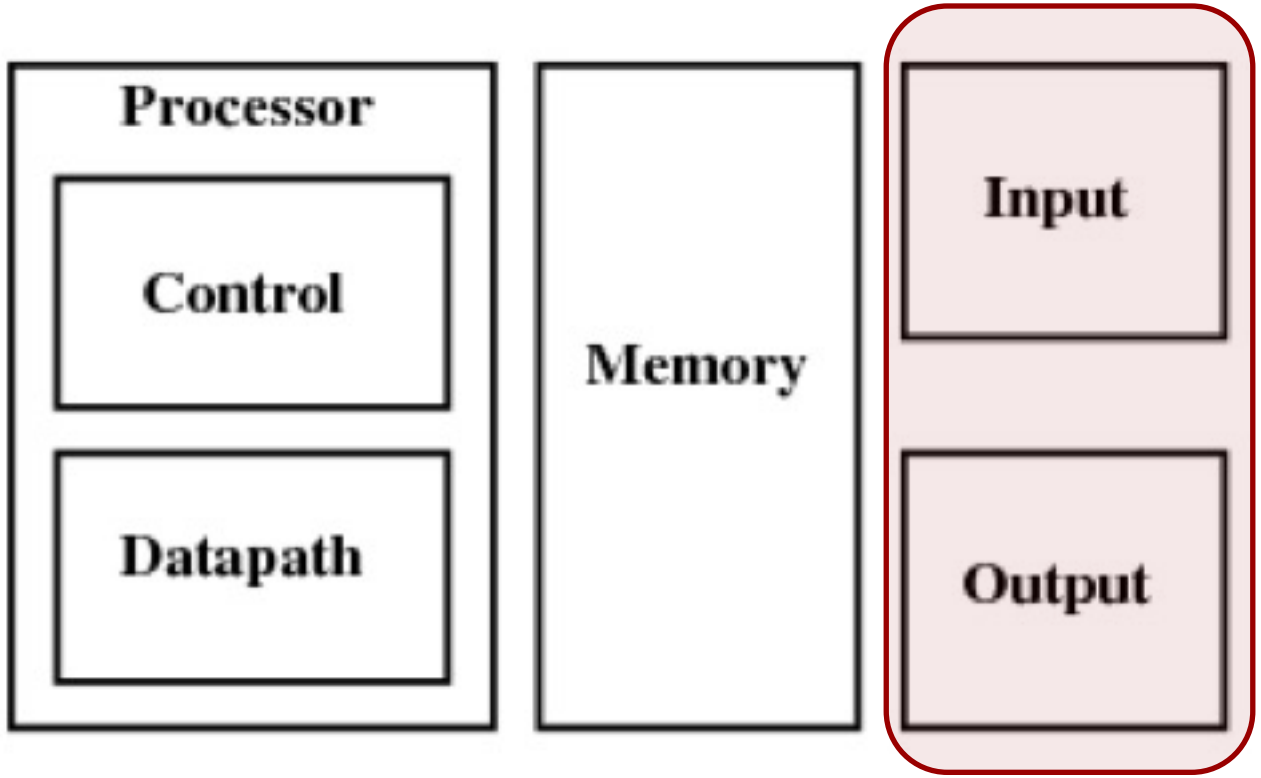
Interfacing I/O Devices

IoT Spring 2024

Suresh Purini, IIIT Hyderabad

Big Picture



I/O Devices

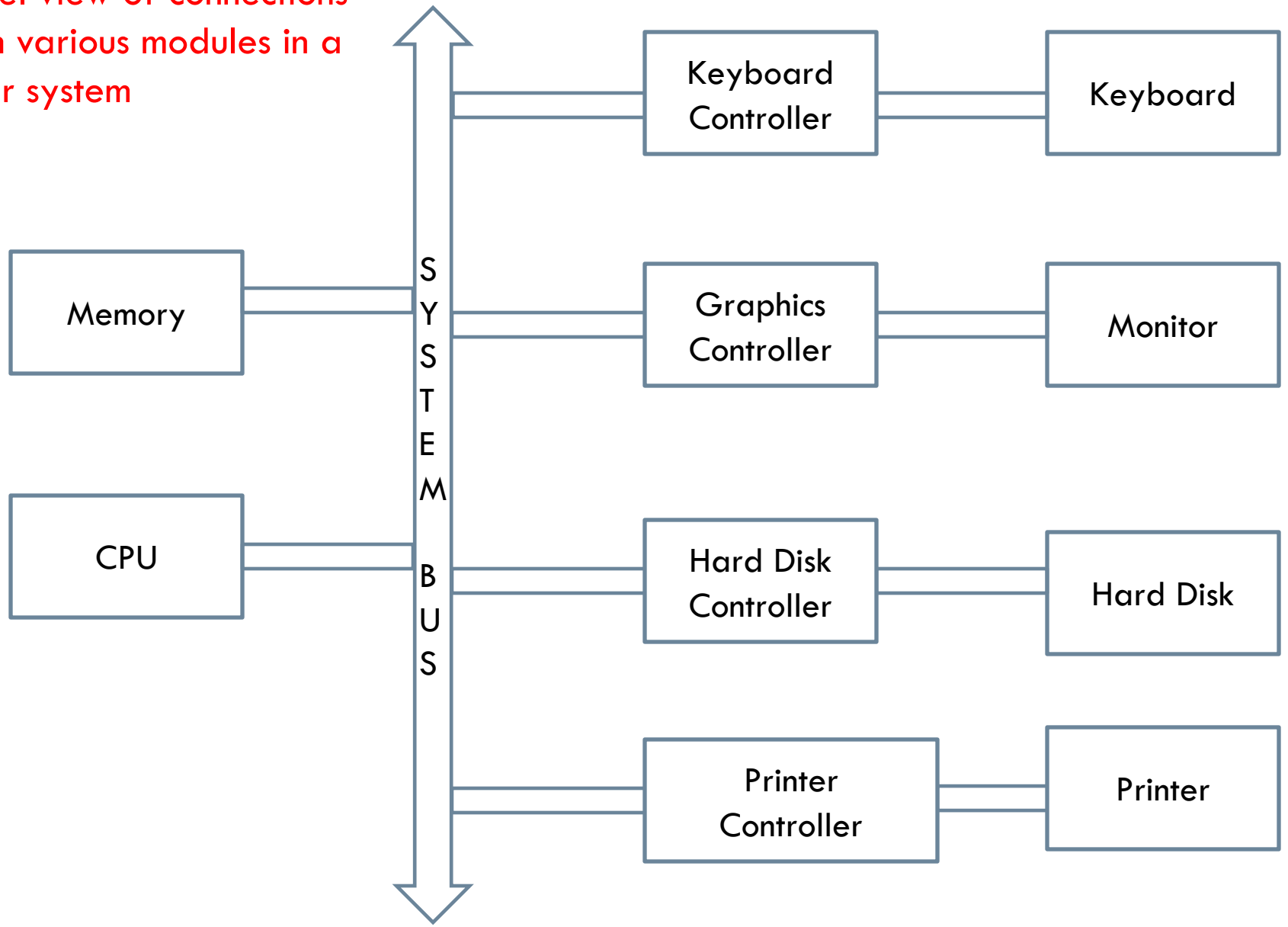
ChatGPT Generated

Device Type	Data Speed	Interface	Use Case
USB Flash Drive	Up to 400 MB/s	USB 2.0/3.0/3.1	Portable storage
HDD (SATA)	Up to 150 MB/s	SATA III	Bulk data storage
SSD (SATA)	Up to 550 MB/s	SATA III	Fast data storage
SSD (NVMe)	Up to 3500 MB/s	PCIe 3.0/4.0	Ultra-fast data storage
Ethernet (Gigabit)	Up to 1 Gbps	RJ45	Network connectivity
Wi-Fi 6	Up to 9.6 Gbps	Wireless	Wireless connectivity
Bluetooth 5	Up to 2 Mbps	Wireless	Short-range wireless connectivity
Keyboard	Low (Not applicable)	USB/Wireless	Input device
Mouse	Low (Not applicable)	USB/Wireless	Input device
Webcam	Up to 5 Gbps (USB 3.0)	USB 2.0/3.0	Video input

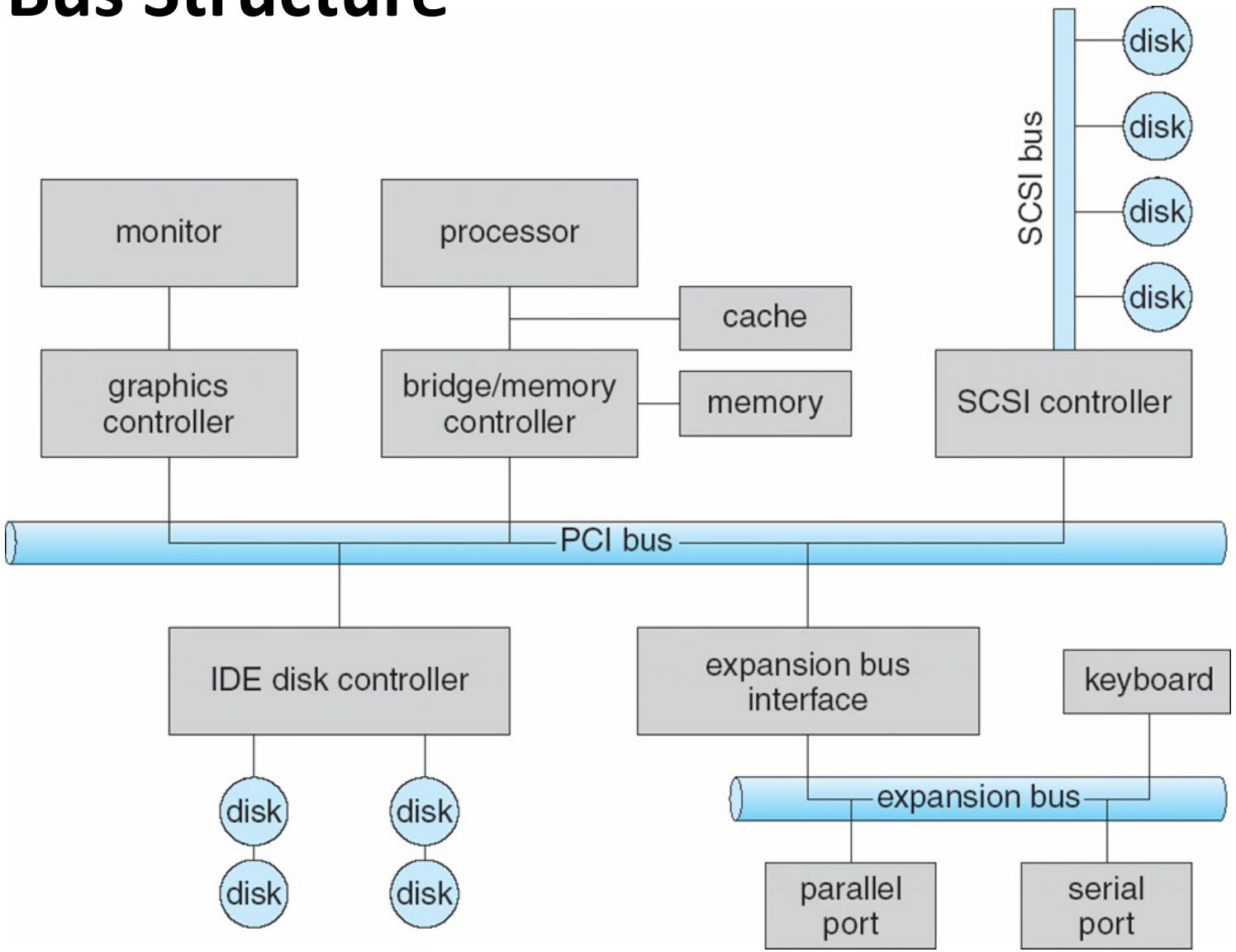
I/O Devices in IoT Nodes

Sensor Type	Interface/Protocol	Data Speed/Rate	Example Use
Analog Temperature Sensor	Analog Voltage	Low (Dependent on reading circuit)	Environmental temperature measurement
Digital Temperature Sensor (I2C)	I2C Digital	Up to 400 Kbps	Precise temperature control systems
Heart Rate Sensor (Analog)	Analog Voltage	Low (Dependent on reading circuit)	Fitness and health tracking
Heart Rate Monitor (Bluetooth Smart)	Bluetooth LE	Up to 1 Mbps	Wireless health monitoring
GPS Module (Serial)	UART Serial	Up to 115200 bps	Location tracking
LIDAR Sensor (USB)	USB	Up to 12 Mbps (USB 2.0 Full Speed)	Distance measurement and mapping

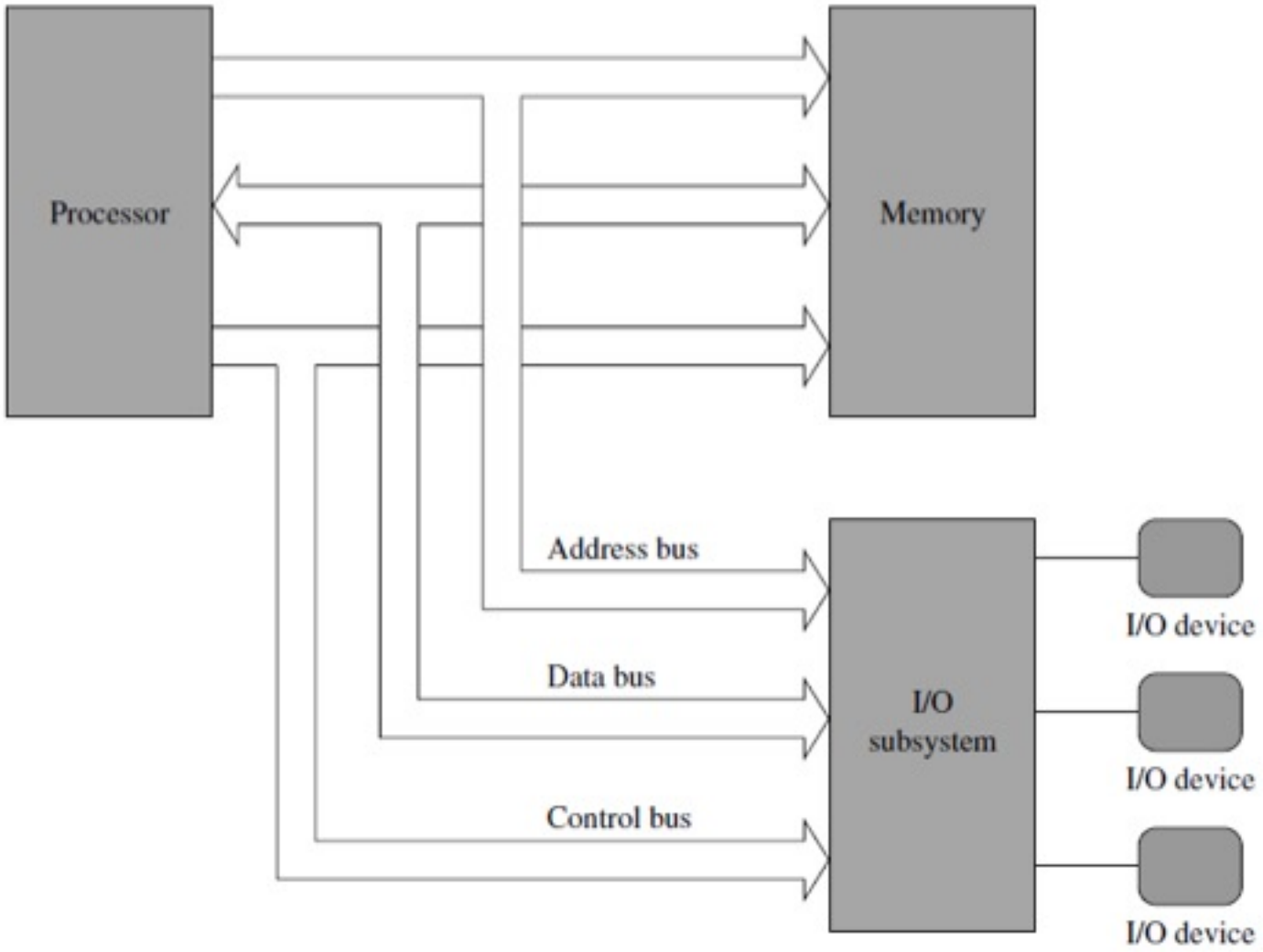
High level view of connections between various modules in a computer system

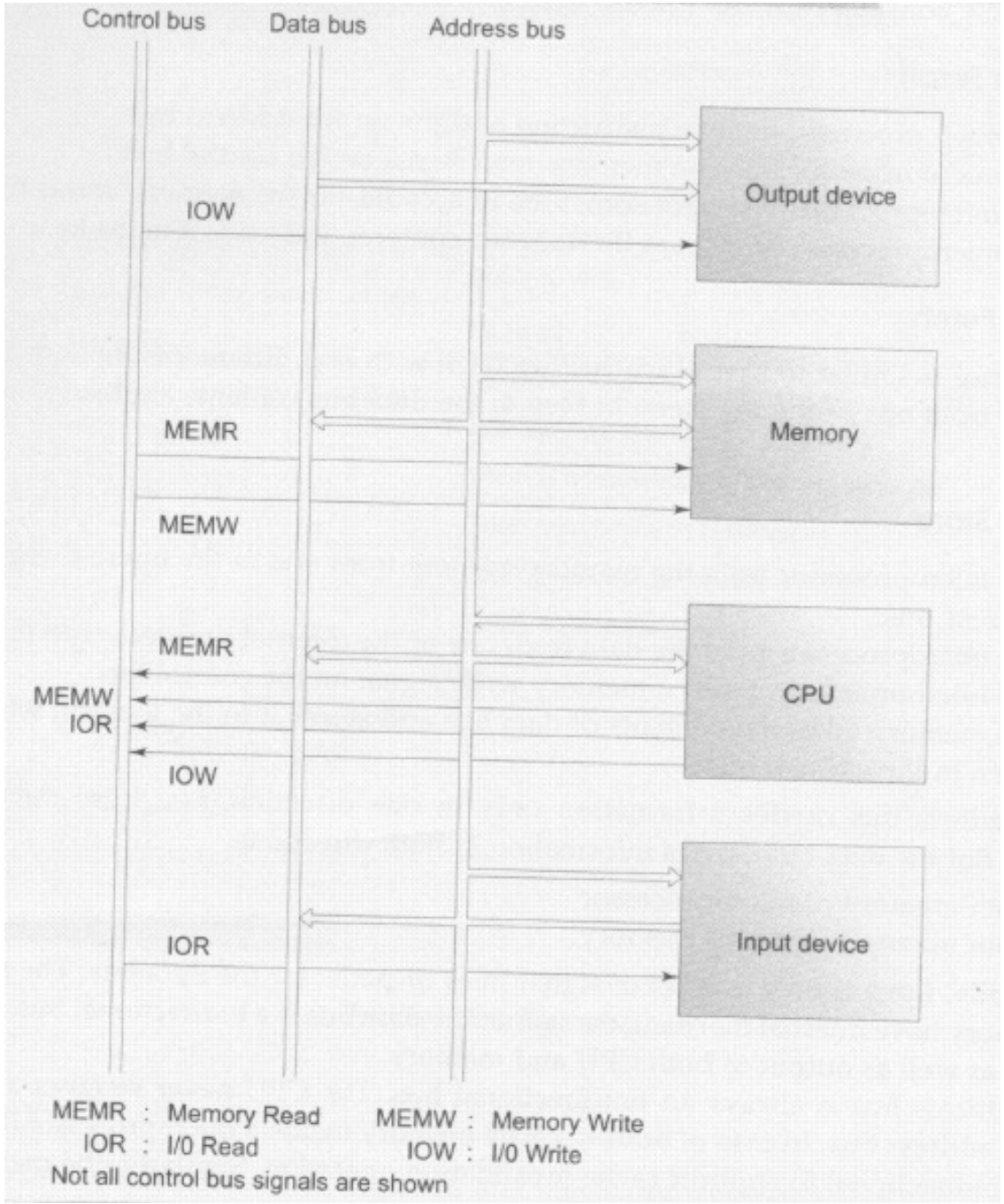


Typical PC Bus Structure



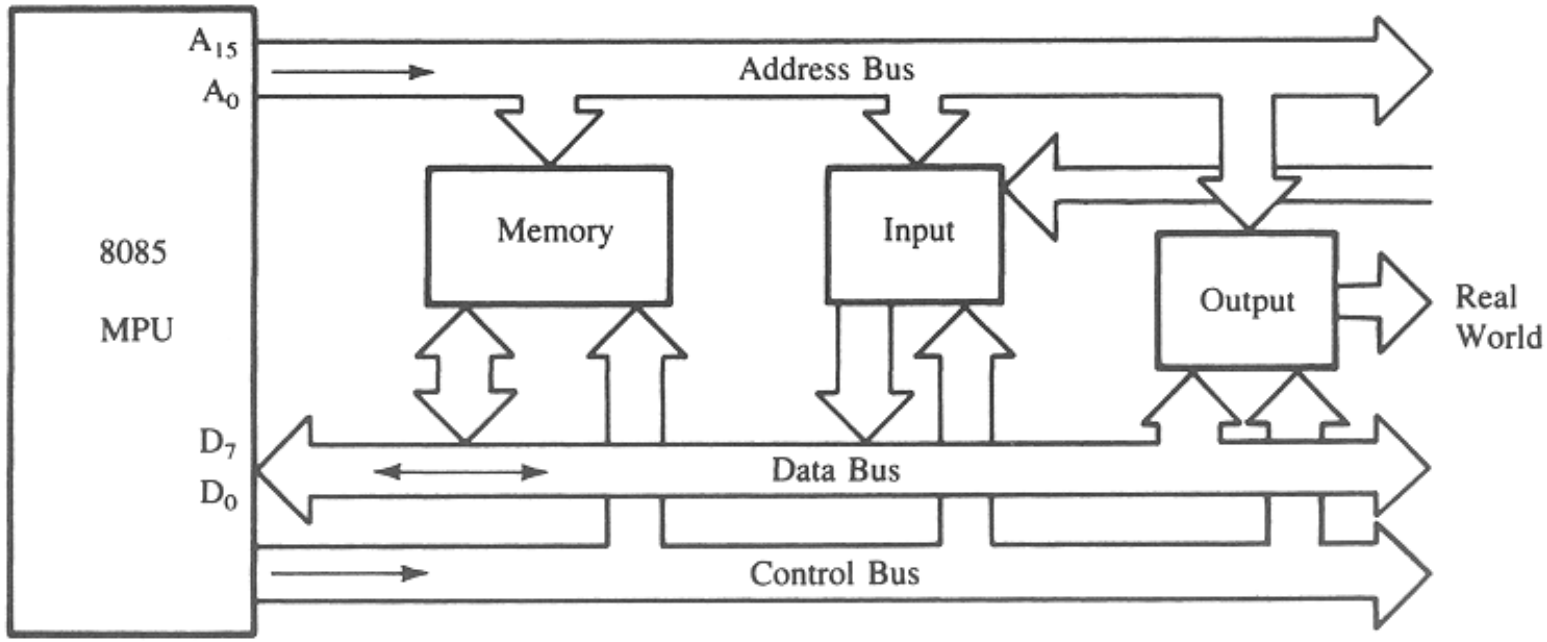
Interconnection: Processor, Memory and I/O Devices



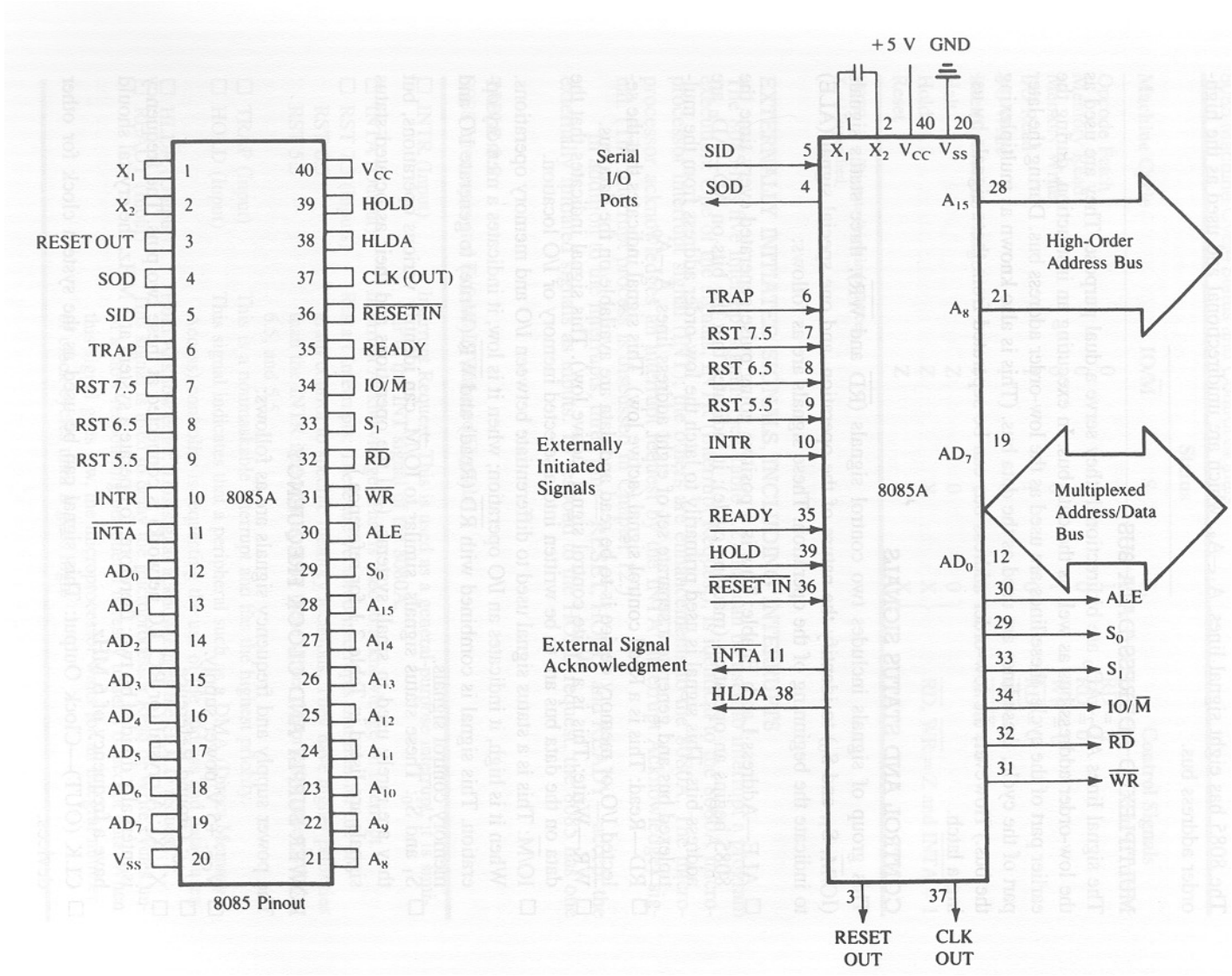

Control bus
Data bus
Address bus
IOW
MEMR
MEMW
MEMR
MEMW
IOR
IOW
IOR
Output device
Memory
CPU
Input device
MEMR : Memory Read
IOR : I/0 Read
MEMW : Memory Write
IOW : I/0 Write
Not all control bus signals are shown

The 8085 Bus Structure

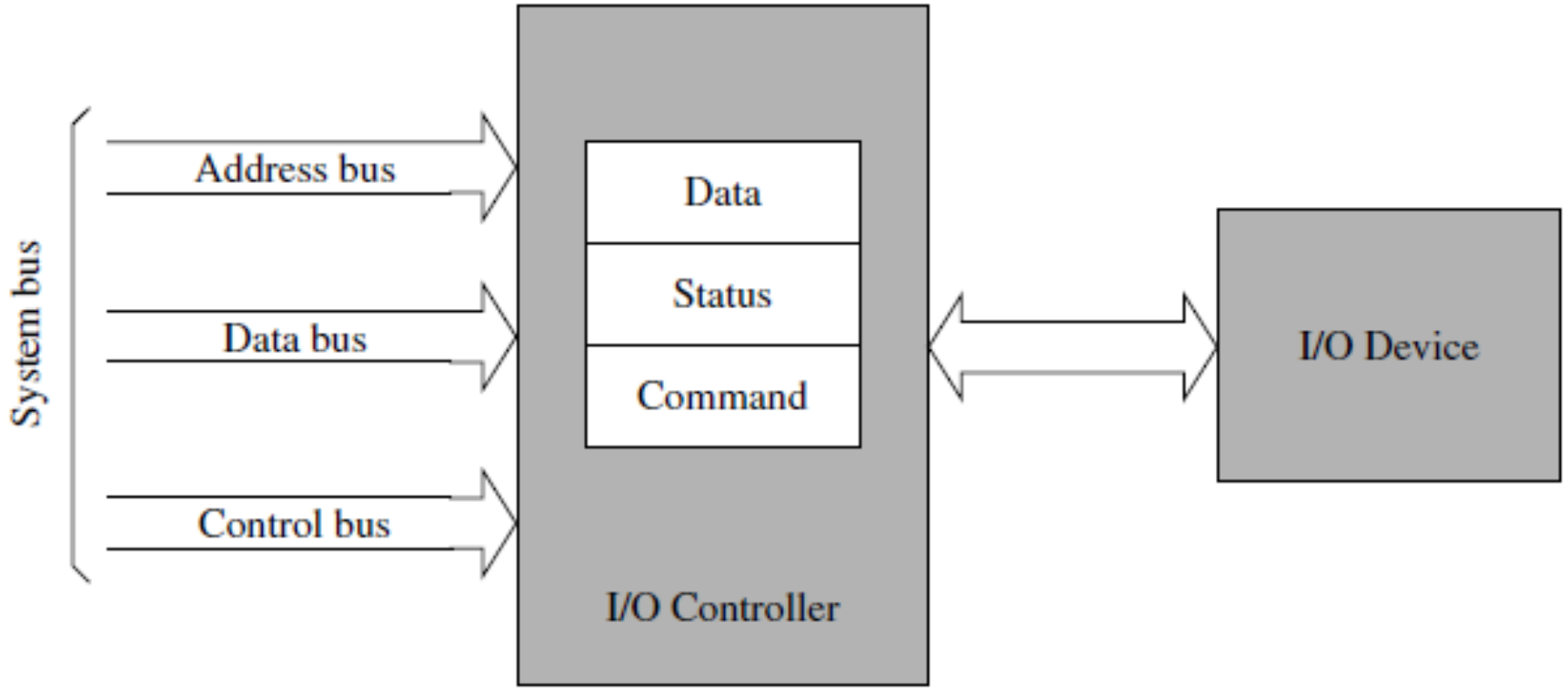
The 8-bit 8085 CPU (or MPU – Micro Processing Unit) communicates with the other units using a 16-bit address bus, an 8-bit data bus and a control bus.



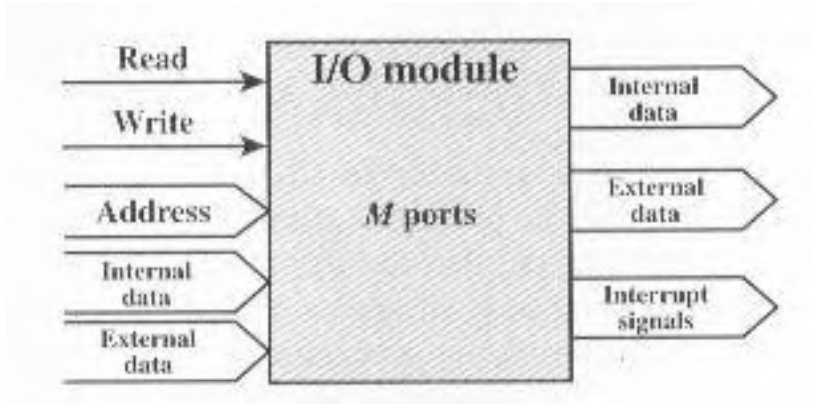
The 8085 Microprocessor



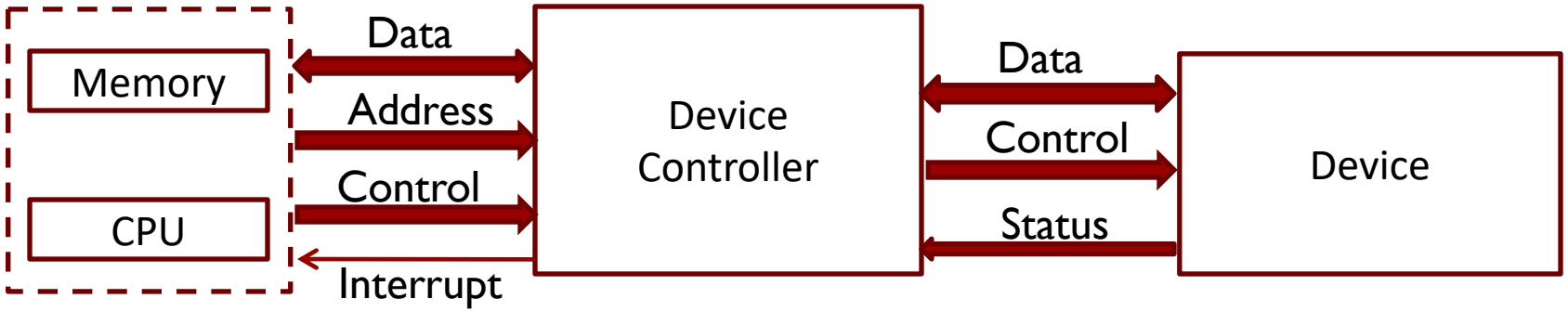
I/O Device Interface



Typical Signal Lines for I/O Modules (Device Controllers)



We need a device driver which knows how to interface with the Device Controller



I/O Mapping

- **Memory mapped I/O**
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
- **Isolated I/O (I/O Mapped I/O)**
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set

I/O Mapped I/O with Polling

C Code to Read Character

```
#define KBDataRegAddr 0x11
#define TTYDataRegAddr 0x12
#define STATUSRegAddr 0x13

unsigned char ch, status;

status = inb(STATUSRegAddr);
while ((status & 0x80) == 0) {
    status = inb(STATUSRegAddr); /* do nothing */
}

ch = inb(KBDataRegAddr);
```

I/O Mapped I/O with Polling

C Code to Write Character

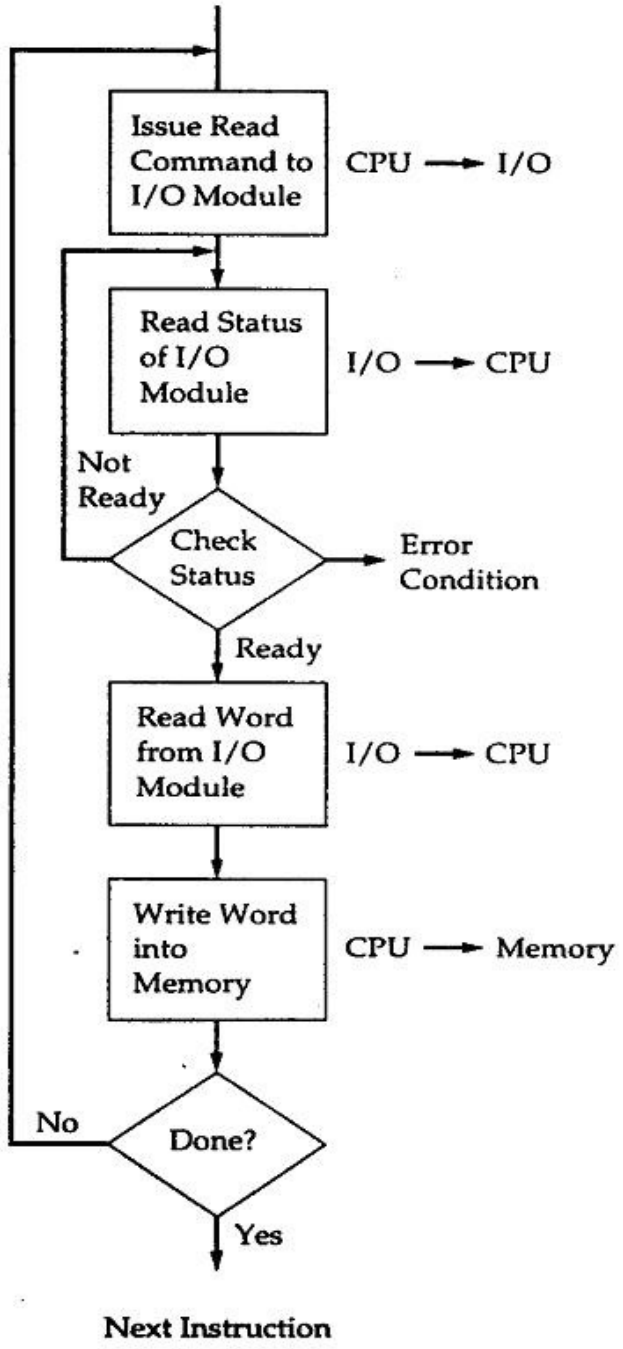
```
#define KBDataRegAddr 0x11
#define TTYDataRegAddr 0x12
#define STATUSRegAddr 0x13

unsigned char ch, status;

status = inb(STATUSRegAddr);
while ((status & 0x40) != 0) {
    status = inb(STATUSRegAddr); /* do nothing */
}

outb(ch, TTYDataRegAddr);
```

Programmed I/O



Device Drivers

Programmed I/O

Polling examples,
But mmap I/O more efficient

```
char read_kbd()
{
do {
    sleep();
    status = inb(0x64);
  } while(!(status & 1));

  return inb(0x60);
}
```

syscall

Memory Mapped I/O

```
struct kbd {
  char status, pad[3];
  char data, pad[3];
};
kbd *k = mmap(...);

char read_kbd()
{
  do {
    sleep();
    status = k->status;
  } while(!(status & 1));
  return k->data;
}
```

syscall

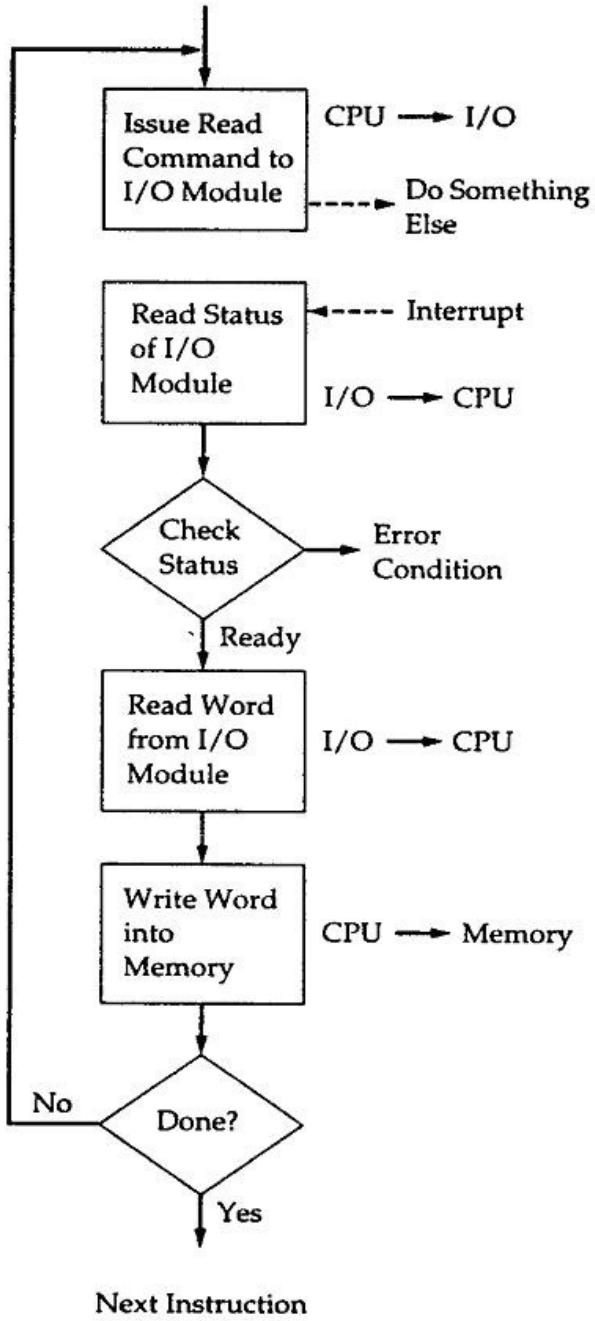
NO syscall

Credit: Prof. Hakim Weatherspoon, Cornell University

Interrupt Driven I/O

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

Interrupt Driven I/O



Interrupt Driven I/O

What does the CPU do after issuing an I/O request?

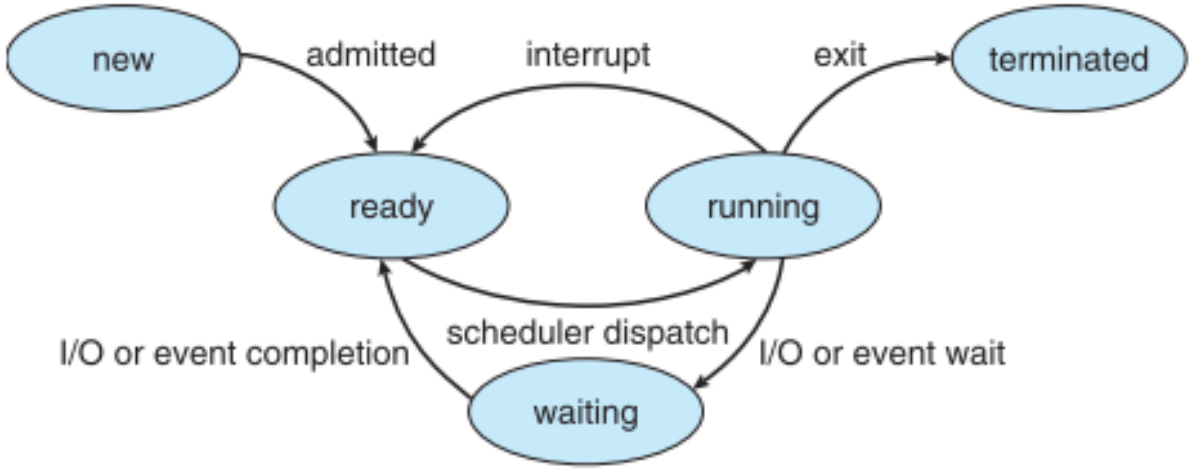
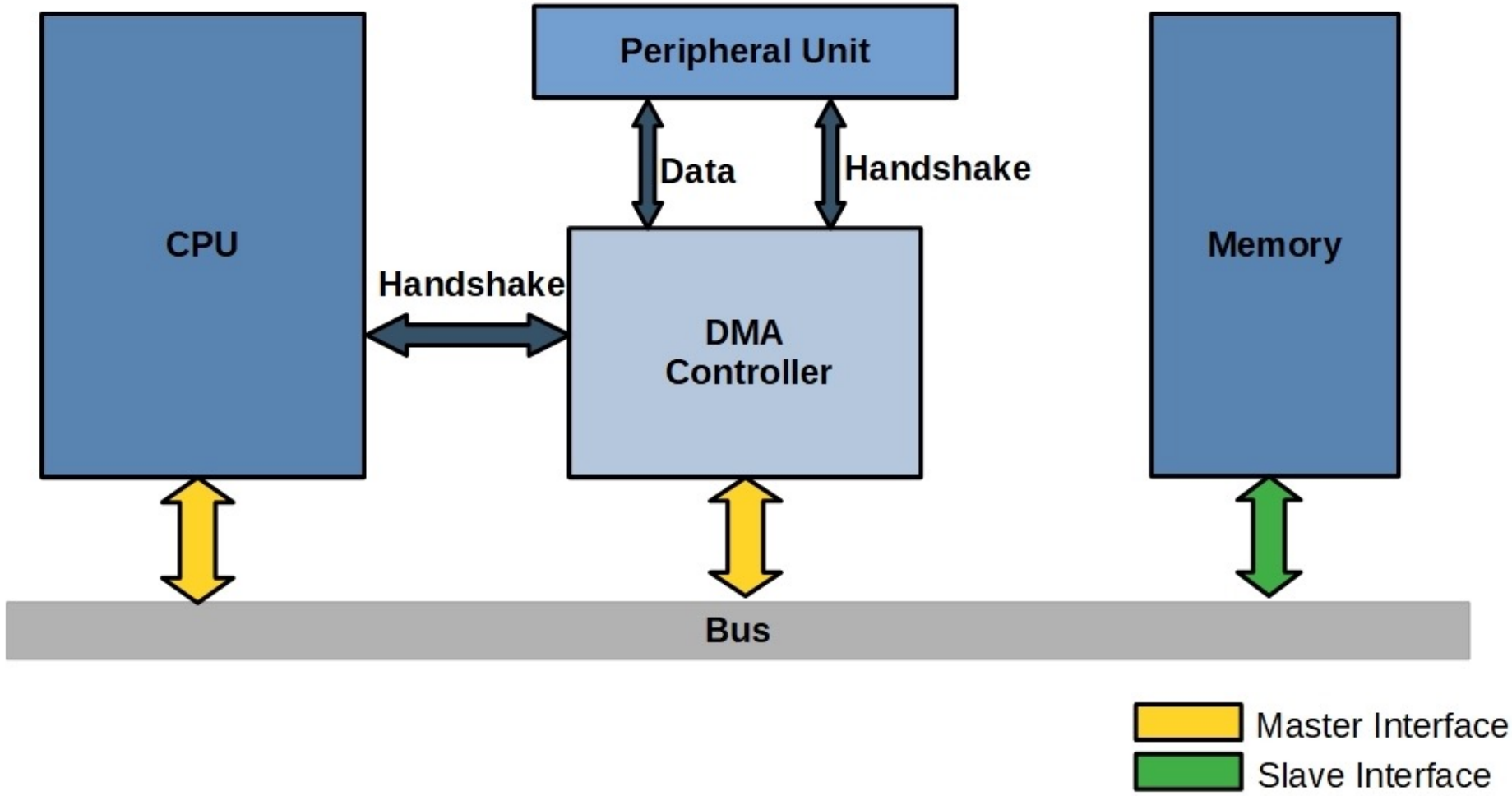
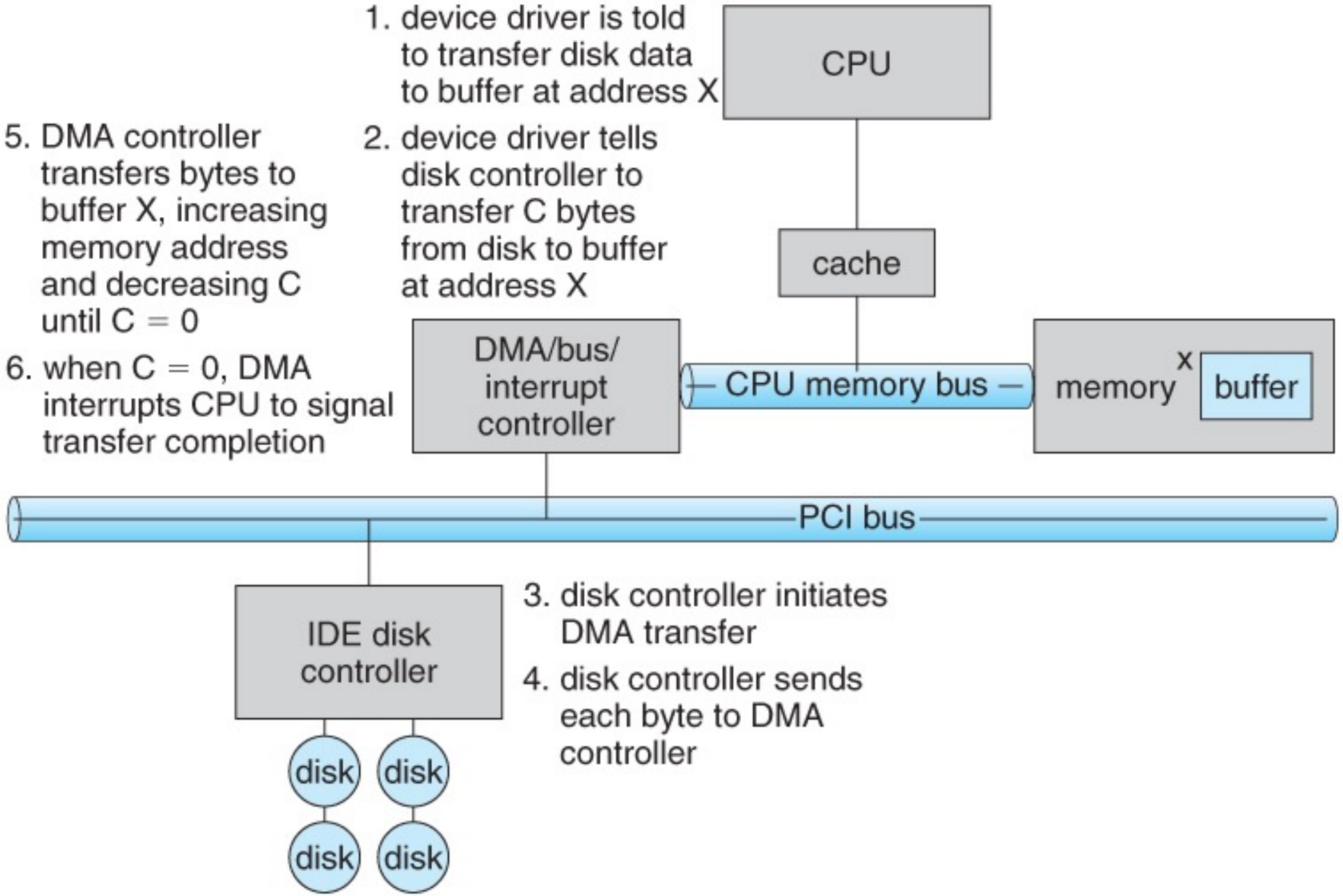


Figure 3.2 Diagram of process state.

What is not discussed? DMA




1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
DMA/bus/ interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

THANK YOU