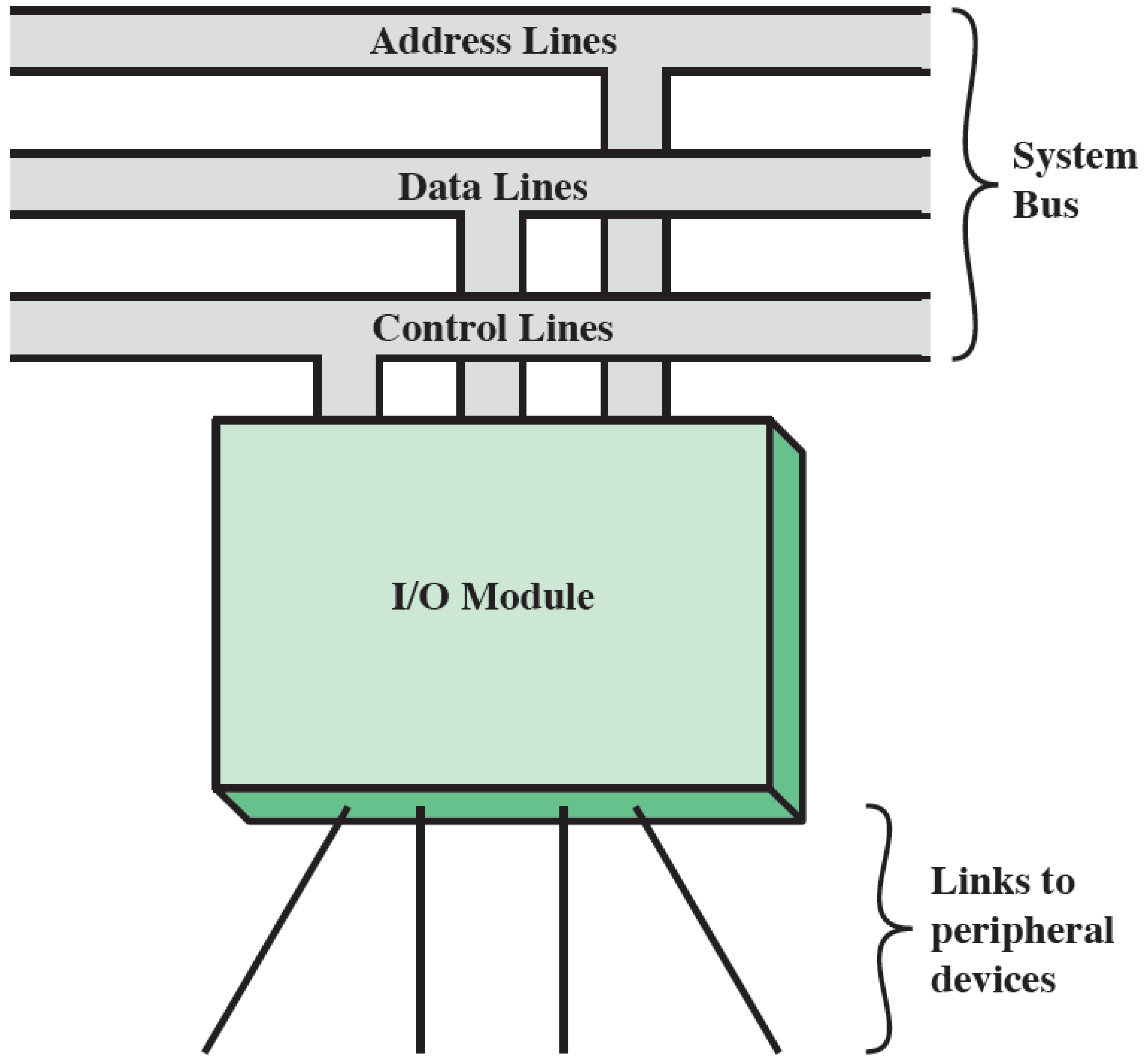
# I/O
## Lecture Content

- Introduction
- I/O Types
  - Programmed I/O
  - Interrupt-Driven
  - DMA
  - I/O Channels
- External Interfaces: USB
- InfiniBand

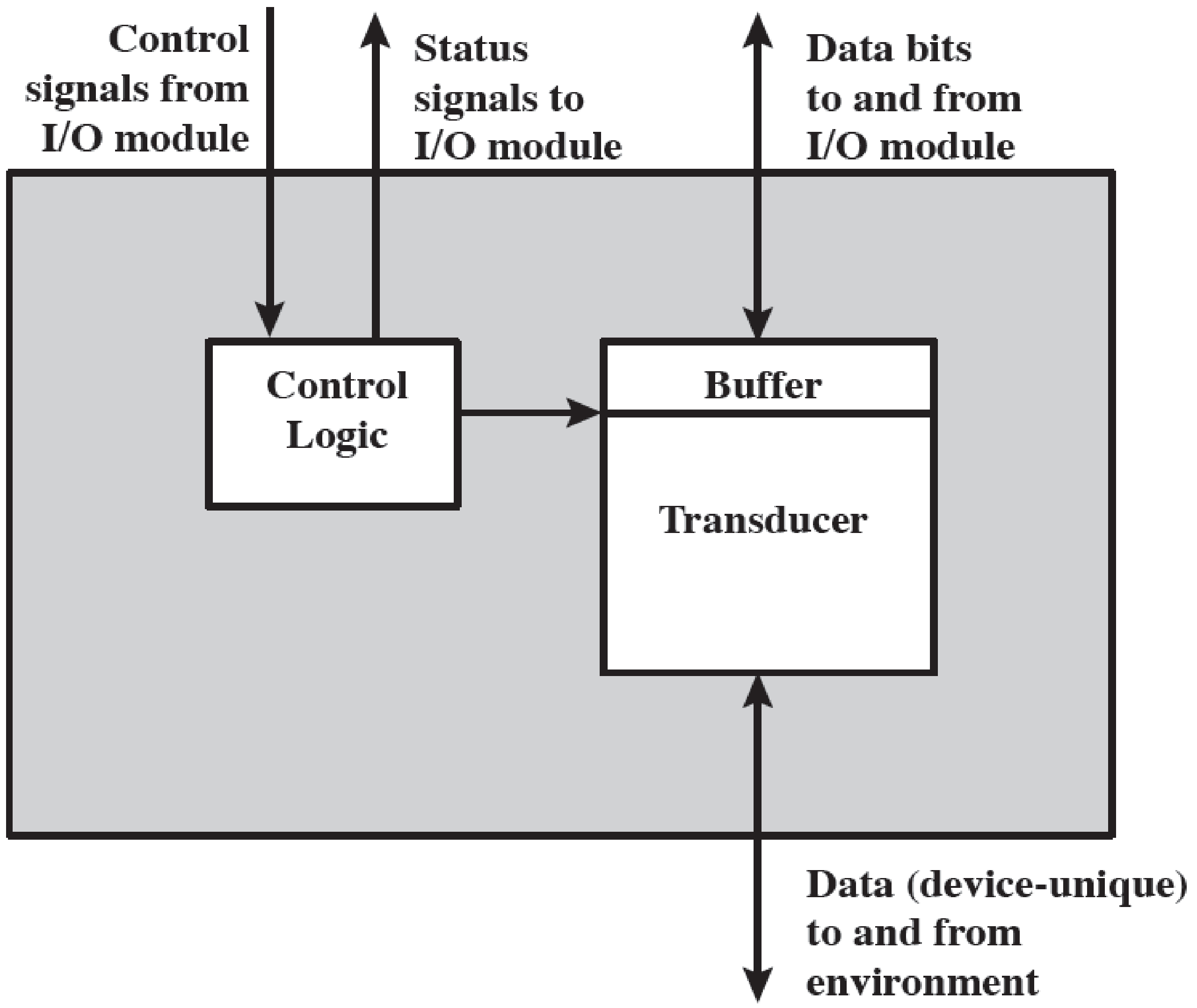
# I/O
## Learning Objectives

- Understand the use and integration of I/O modules
- Understand the differences  programmed and interrupt-driven I/O
- Understand the concepts of DMA
- See two modern examples: USB and Infiniband

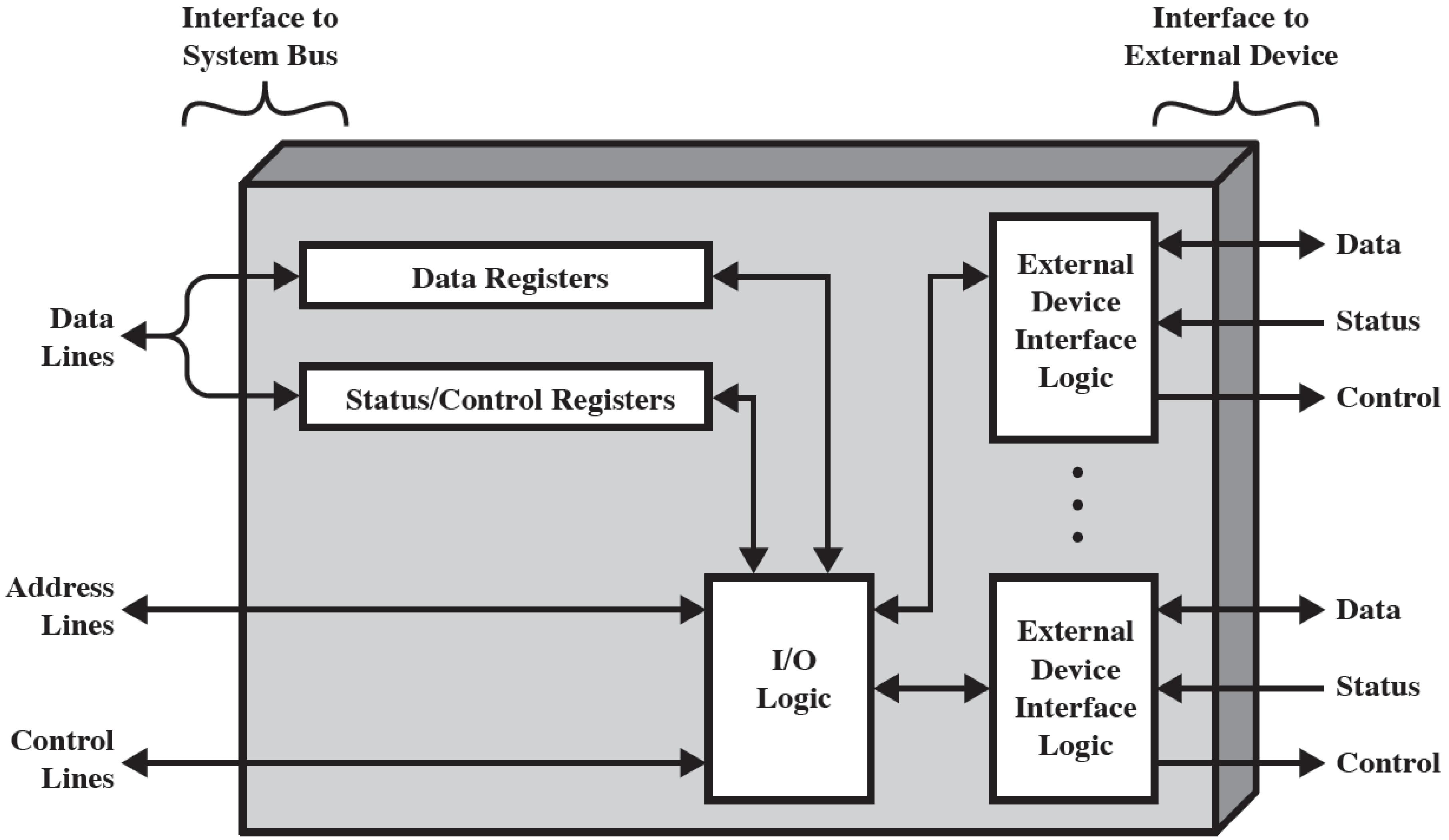
# Recall the General I/O Structure



## External Device


Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique) to and from environment

# I/O Module


Interface to System Bus
Interface to External Device
Data Lines
Address Lines
Control Lines
Data Registers
Status/Control Registers
I/O Logic
External Device Interface Logic
External Device Interface Logic
Data
Status
Control
Data
Status
Control

## Why Using an I/O Module?

- Allows the processor to communicate with a wide range of devices in a simple-minded way
- The I/O module may hide the details of timing, formats, and the electromechanics of an external device
- The processor can function in terms of simple read and write commands
- In its simplest form, the I/O module may still leave much of the work of controlling a device (e.g., rewind a tape) visible to the processor

# Duties of an I/O Module

- **Control and Timing**
  - Coordinates the traffic between internal resources and external devices
- **Processor Communication**
  - Involves command decoding, data, status reporting, address recognition
- **Device Communication**
  - Involves commands, status information, and data
- **Data Buffering**
  - Performs the buffering operation to balance device and memory speeds
- **Error Detection**
  - Detects and reports transmission errors

## Some Naming Confusion

- An I/O module is often categorized as follows:

- **I/O Channel** or **I/O Processor**:
  - takes on most of the detailed processing burden
  - presenting a high-level interface to the processor

- **I/O Controller** or **Device Controller**
  - quite primitive
  - requires detailed control of the CPU

- Controllers are commonly seen on PCs, whereas I/O channels are used on mainframes

# I/O
## Lecture Content

- Introduction
- I/O Types
  - Programmed I/O
  - Interrupt-Driven
  - DMA
  - I/O Channels
- External Interfaces: USB
- InfiniBand

## Different Types of I/O

- **Programmed I/O**
  - Data are exchanged between the processor and the I/O module
  - Processor has direct control of the I/O operation
  - The processor must wait until the I/O operation is complete
  - Wasteful, if CPU is faster than I/O
- **Interrupt-driven I/O**
  - CPU can perform different task, gets notified by Interrupt
- **Direct memory access (DMA)**
  - The I/O module and main memory exchange data directly without processor involvement

# I/O

## Lecture Content

- Introduction
- I/O Types
  - Programmed I/O
  - Interrupt-Driven
  - DMA
  - I/O Channels
- External Interfaces: USB
- InfiniBand

# I/O Commands

- **Control**
  - used to activate a peripheral and tell it what to do
- **Test**
  - used to test various status conditions of an I/O module and its peripherals
- **Read**
  - causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer
- **Write**
  - causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral

## I/O Instructions with Programmed I/O

- Close correspondence between the I/O instruction and the device
- The instruction depends on the way in which external devices are addressed

- Typically, there are several devices
  - Each device has an identifier or address
  - The command contains the address of the desired device
  - Thus each I/O module must interpret the address lines to determine if the command is for itself

- Sharing the bus with CPU, Main Memory and I/O
  - Memory Mapped
  - Isolated

## Memory Mapped

- **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**
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O

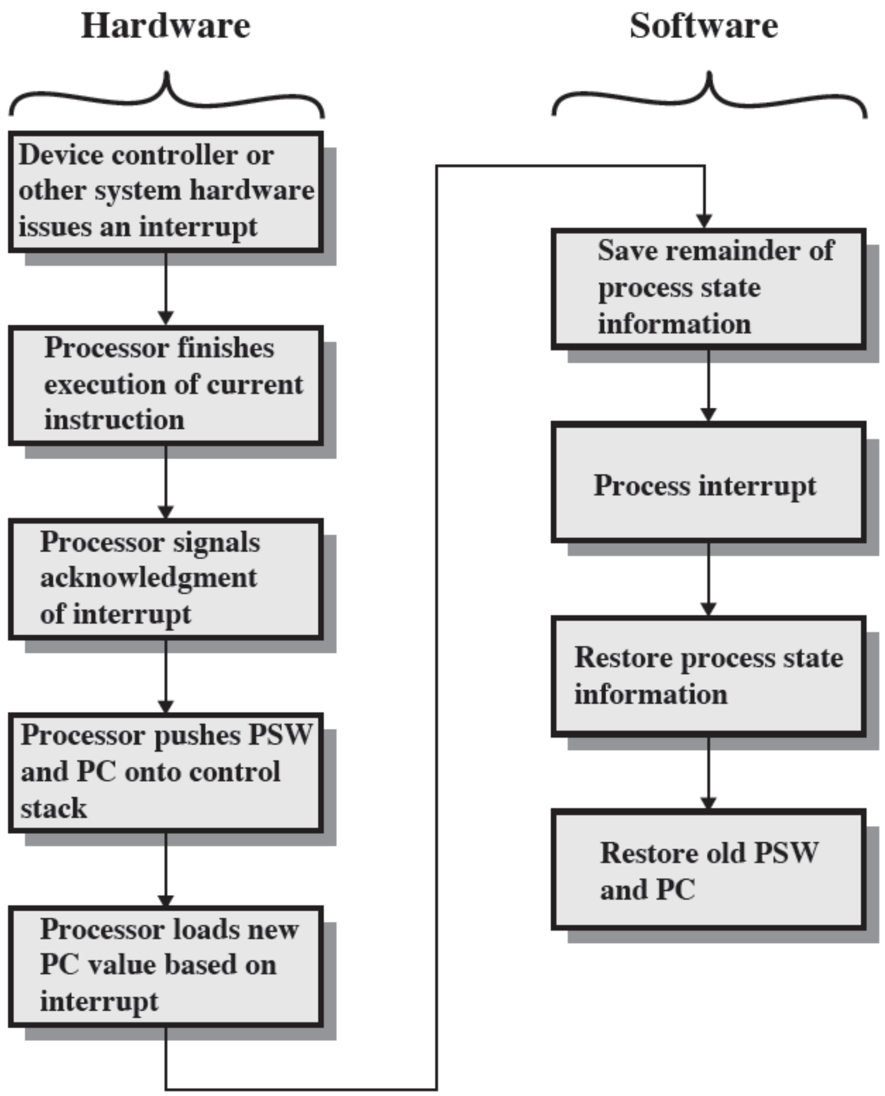
# I/O
## Lecture Content

- Introduction
- I/O Types
  - Programmed I/O
  - Interrupt-Driven
  - DMA
  - I/O Channels
- External Interfaces: USB
- InfiniBand

## Interrupt Driven I/O

- The problem with programmed I/O is that the processor has to wait a long time for the I/O module to be ready for either reception or transmission of data
- An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work
- The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor
- The processor executes the data transfer and resumes its former processing

# Interrupt Processing


Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

## Two Fundamental Implementation Issues

- Because there will be multiple I/O modules how does the processor determine which device issued the interrupt?
- If multiple interrupts have occurred how does the processor decide which one to process? (we already discussed that a couple lectures ago)

## Device Identification

- Multiple Interrupt Lines
- Software Poll
- Daisy Chain (hardware poll, vectored)
- Bus Arbitration (vectored)

## Device Identification

- **Multiple Interrupt Lines**
  - Between the processor and the I/O modules
  - Most straightforward approach to the problem
  - Consequently even if multiple lines are used, it is likely that each line will have multiple I/O modules attached to it
- **Software Poll**
- **Daisy Chain (hardware poll, vectored)**
- **Bus Arbitration (vectored)**

## Device Identification

- **Multiple Interrupt Lines**
- **Software Poll**
  - When processor detects an interrupt it branches to an interrupt-service routine whose job is to poll each I/O module to determine which module caused the interrupt
  - Time consuming
- **Daisy Chain (hardware poll, vectored)**
- **Bus Arbitration (vectored)**

## Device Identification

- **Multiple Interrupt Lines**

- **Software Poll**

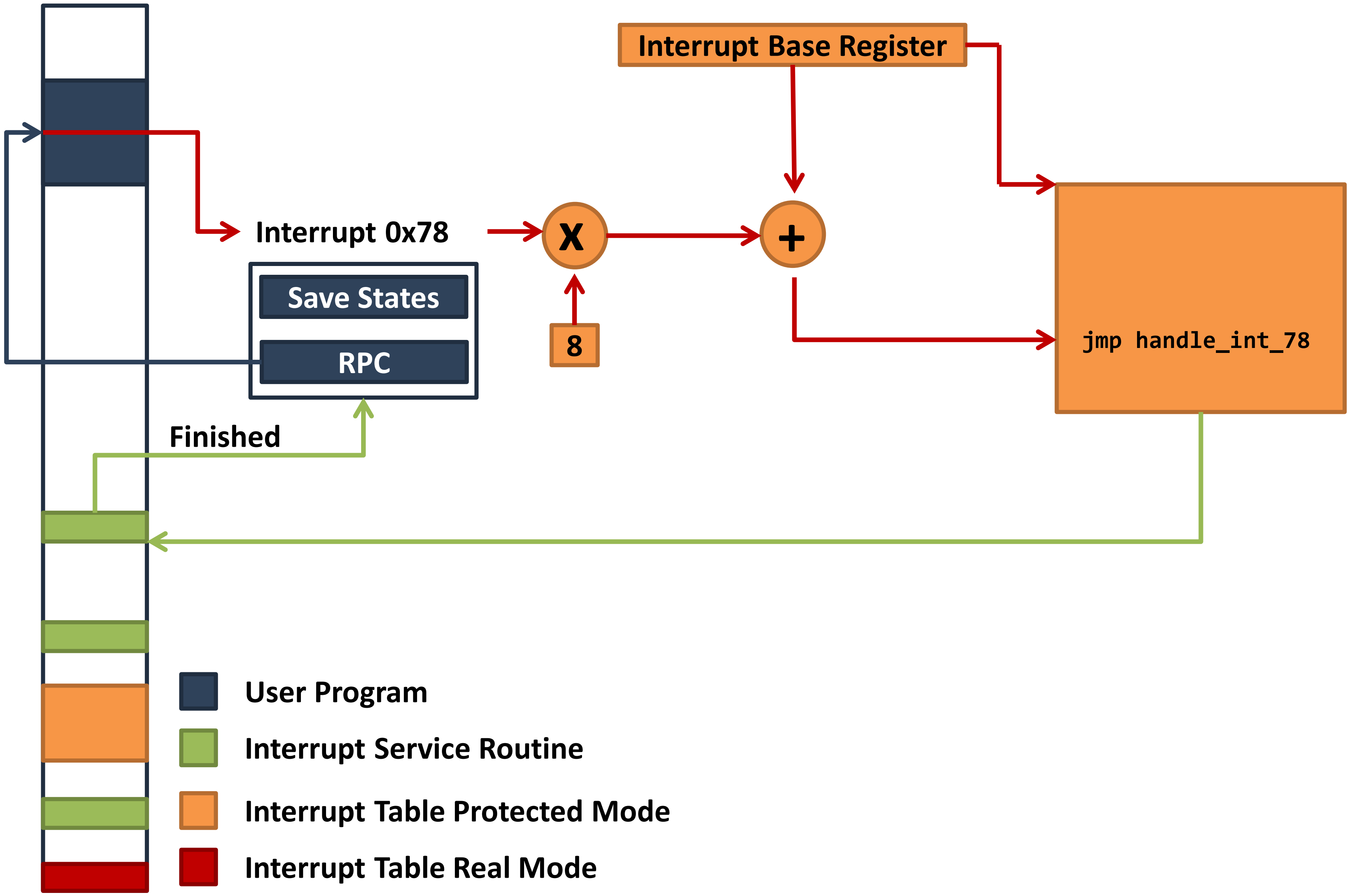
- **Daisy Chain (hardware poll, vectored)**
  - The interrupt acknowledge line is daisy chained through the modules
  - Vector – address of the I/O module or some other unique identifier
  - Vectored interrupt – processor uses the vector as a pointer to the appropriate device-service routine, avoiding the need to execute a general interrupt-service routine first

- **Bus Arbitration (vectored)**

## Device Identification

- **Multiple Interrupt Lines**
- **Software Poll**
- **Daisy Chain (hardware poll, vectored)**
- **Bus Arbitration (vectored)**
  - An I/O module must first gain control of the bus before it can raise the interrupt request line
  - When the processor detects the interrupt it responds on the interrupt acknowledge line
  - Then the requesting module places its vector on the data lines

# Vectored Interrupts



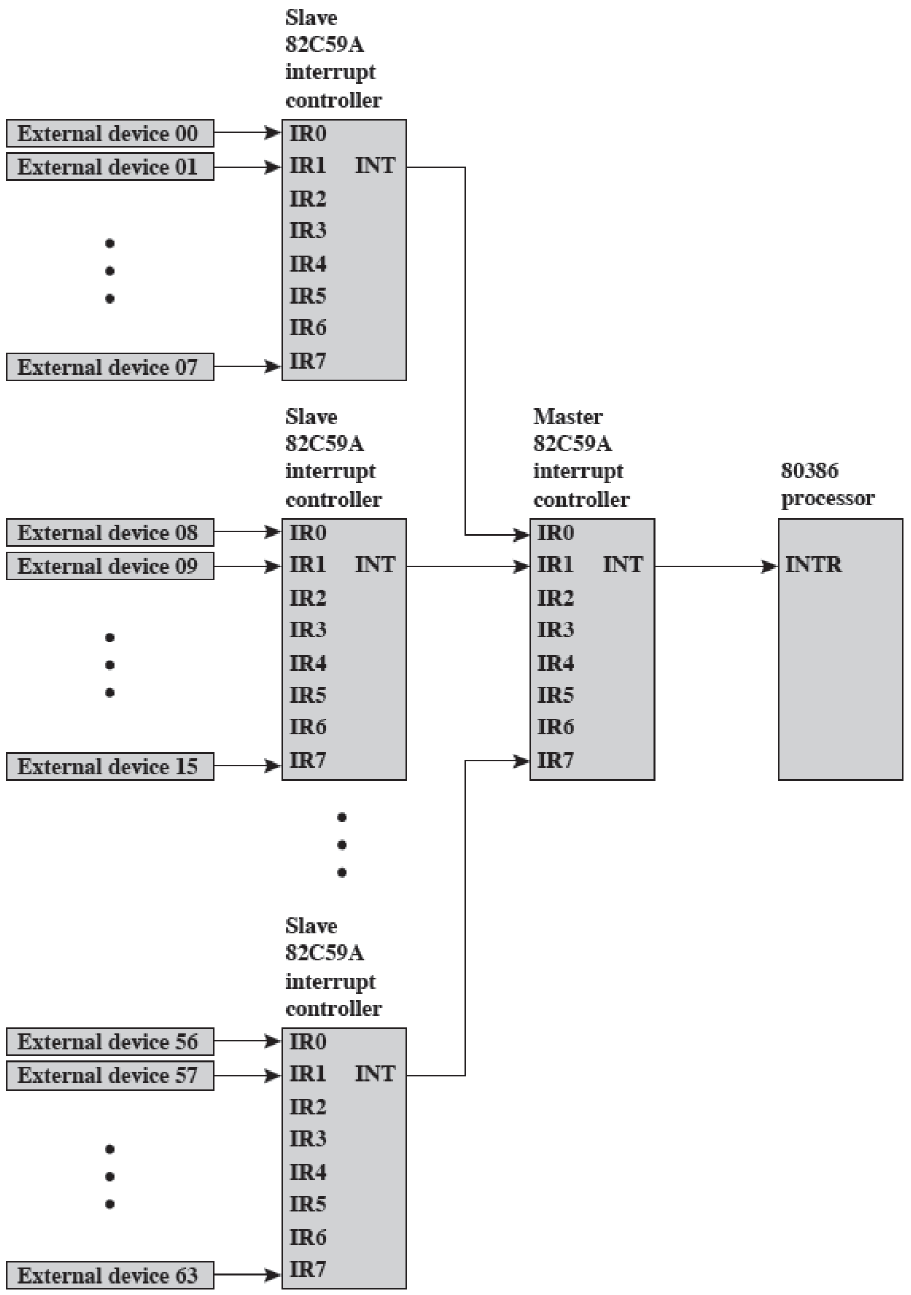
🟦 User Program

🟩 Interrupt Service Routine

🟧 Interrupt Table Protected Mode

🟥 Interrupt Table Real Mode

# Intel 8259 PIC



- Image: wikipedia.org

## The Intel 8259 Programmable Interrupt Controller

- Introduced for the 8086, still in use (only for compatibility reasons)
- The CPU has two interrupt lanes
  - INTR – Interrupt Request
  - INTA – Interrupt Acknowledge
    - After Acknowledge, the 8259 puts the address (for the vector) of the requesting device on the bus

- **Fully Nested**
  - The interrupt requests are ordered in priority from 0 (IR0) through 7 (IR7)
- **Rotating**
  - In some applications a number of interrupting devices are of equal priority
- **Special Mask**
  - This allows the processor to inhibit interrupts from certain devices.

# Drawbacks of Programmed and Interrupt-Driven I/O

- **Programmed I/O**:
  - Most of the time, the CPU idles while waiting for I/O (busy waiting)
  - Using simple programmed I/O, the processor is dedicated to the task of I/O and can move data at a rather high rate, at the cost of doing nothing else.

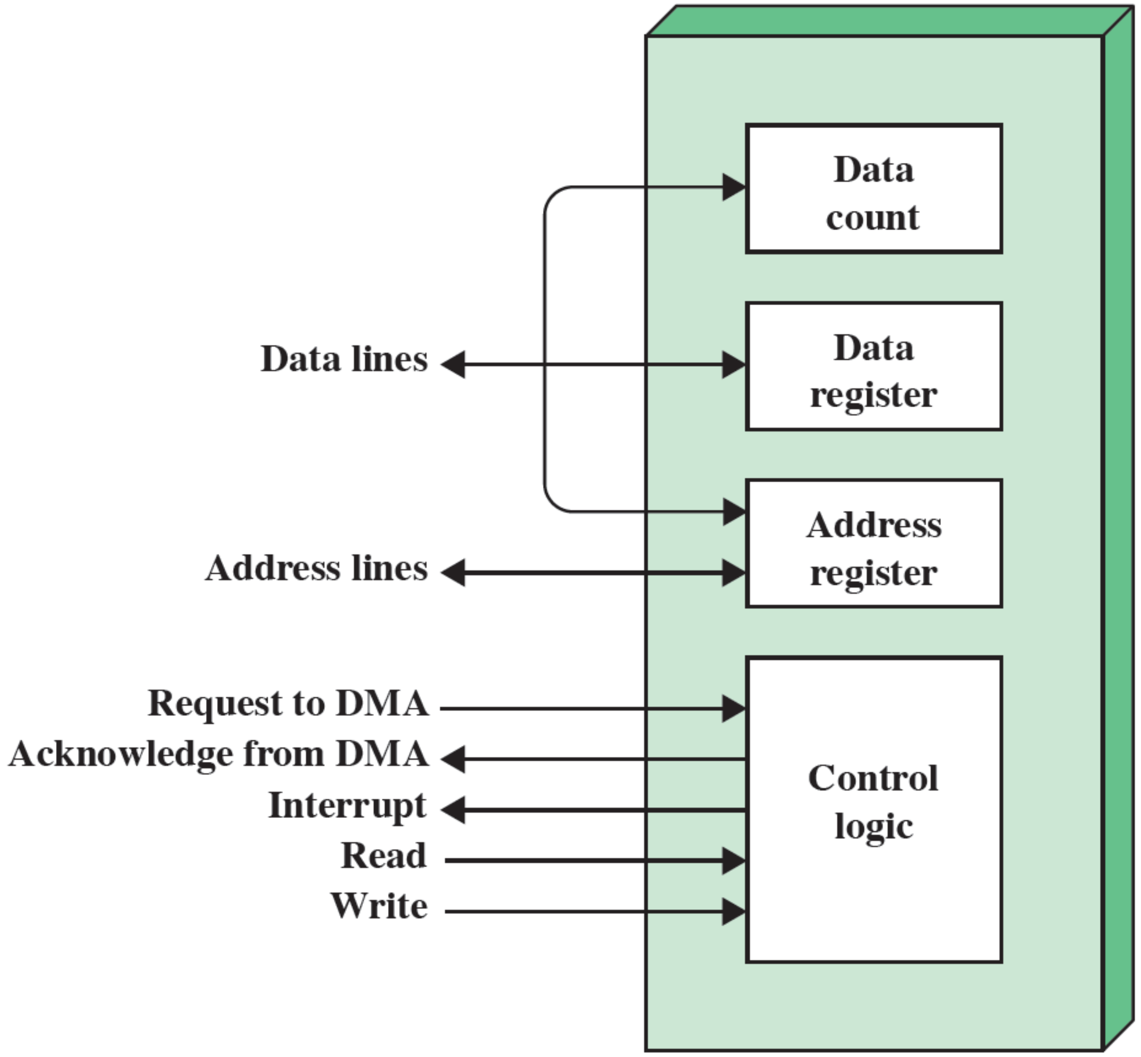
- **Interrupt-driven I/O**:
  - Kind of a trade-off between the above: Interrupt I/O frees up the processor to some extent at the (potential) expense of the I/O transfer rate
  - In reality much faster (overall)
  - But:
    - More complicated to coordinate
    - The CPU is still involved in every data transfer
    - I/O is limited by how fast the CPU can handle interrupts

# I/O

## Lecture Content

- Introduction
- I/O Types
  - Programmed I/O
  - Interrupt-Driven
  - DMA
  - I/O Channels
- External Interfaces: USB
- InfiniBand

# DMA



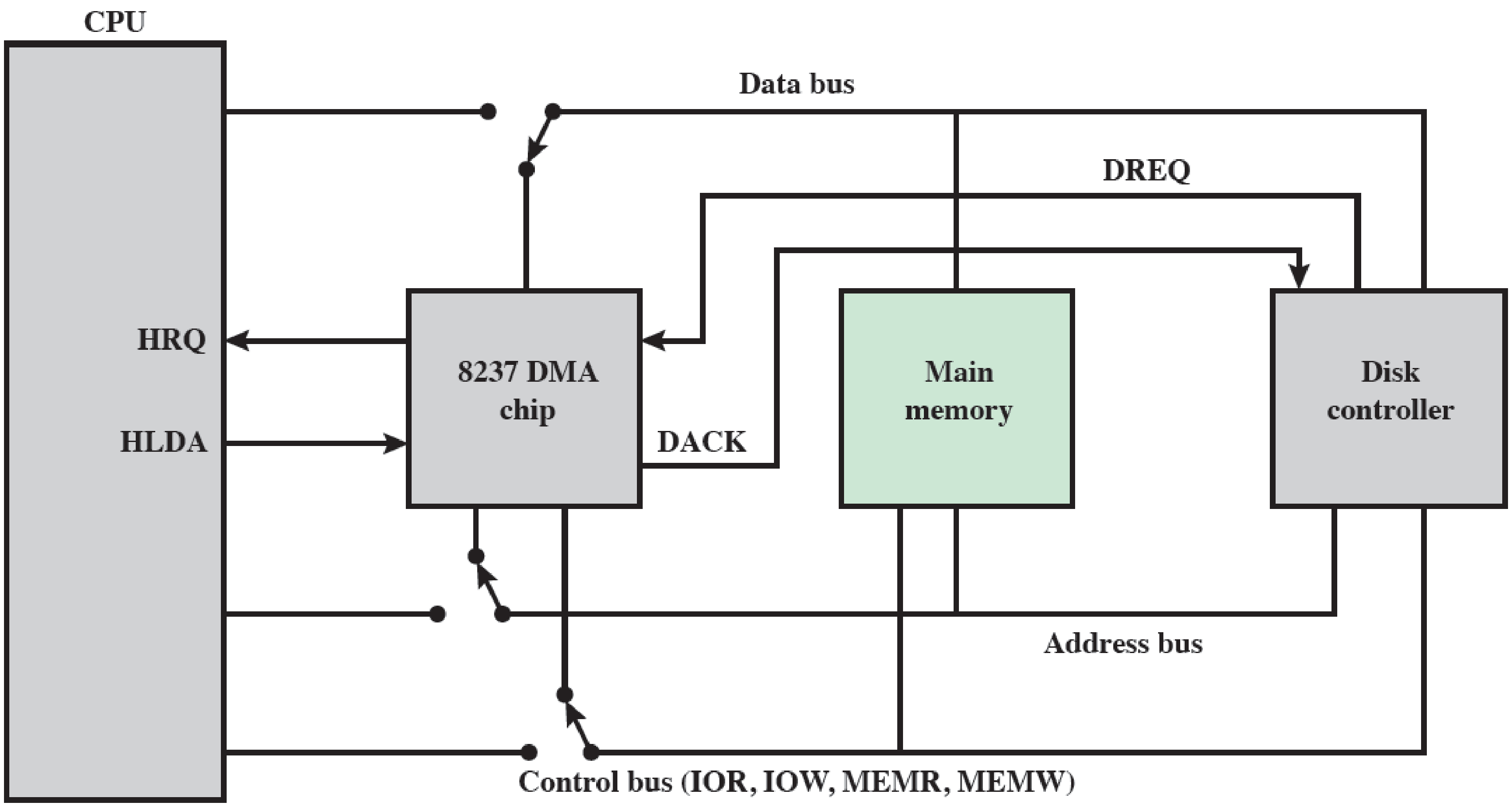
- DMA requires an additional module
- After requesting a DMA transaction, the CPU completely delegated the I/O transfer to the DMA module

## DMA Operation in Detail

- The CPU provides the DMA with the following information
  - Read or Write, using the read or write control line
  - The address of the I/O device involved, communicated on the data lines
  - The starting location in memory to read from or write to
    - communicated on the data lines
    - stored by in the address register
  - The number of words to be read or written, again communicated via the data lines and stored in the data count register

- When the transfer is complete, the DMA module sends an interrupt signal to the processor
  - Thus, the processor is involved only at the beginning and end of the transfer

# Example: Intel 8237 DMA Chip



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

## Example: Write a Block to Disk

1. A device requests DMA by pulling DREQ (DMA request) high
2. The DMA will put a high on its HRQ (hold request)
3. The CPU will finish the present bus cycle and puts high on its HDLA (hold acknowledge)
4. DMA will activate DACK (DMA acknowledge)
5. DMA starts the transfer
   1. DMA puts the address of the first byte of the block on the
   2. Then it activates IOW to write it to the peripheral
6. After the DMA has finished its job it will deactivate HRQ, signaling the CPU that it can regain control over its buses.

# I/O
## Lecture Content

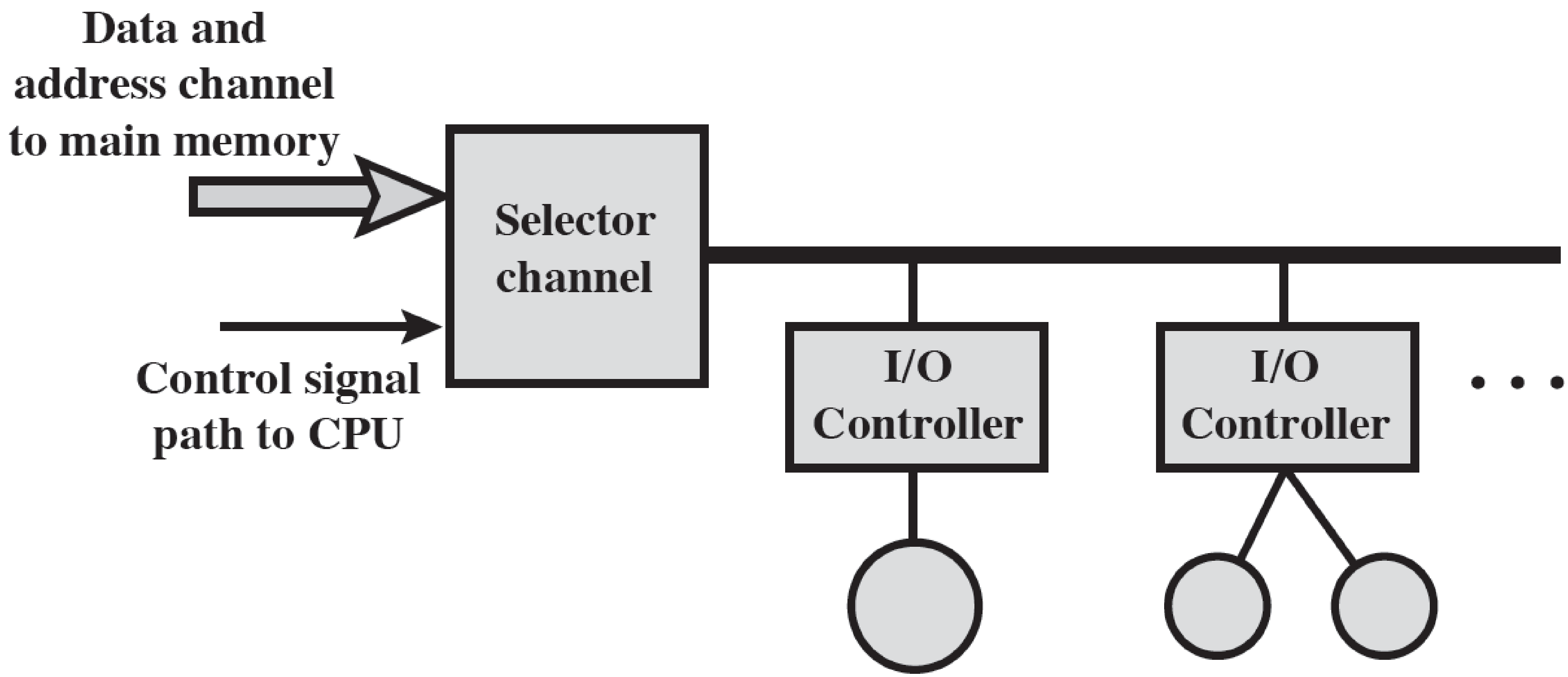
## Evolution of the I/O Function

1. The CPU directly controls a peripheral device
2. An I/O module is added; CPU still uses programmed I/O
   - The Processor benefits from the abstraction
3. The same with interrupts
4. DMA
5. The I/O module becomes an entire processor (specialized I/O ISA)
   - The CPU only gets interrupted after the entire transfer is completed
6. The I/O module has a local memory; becomes an entire computer

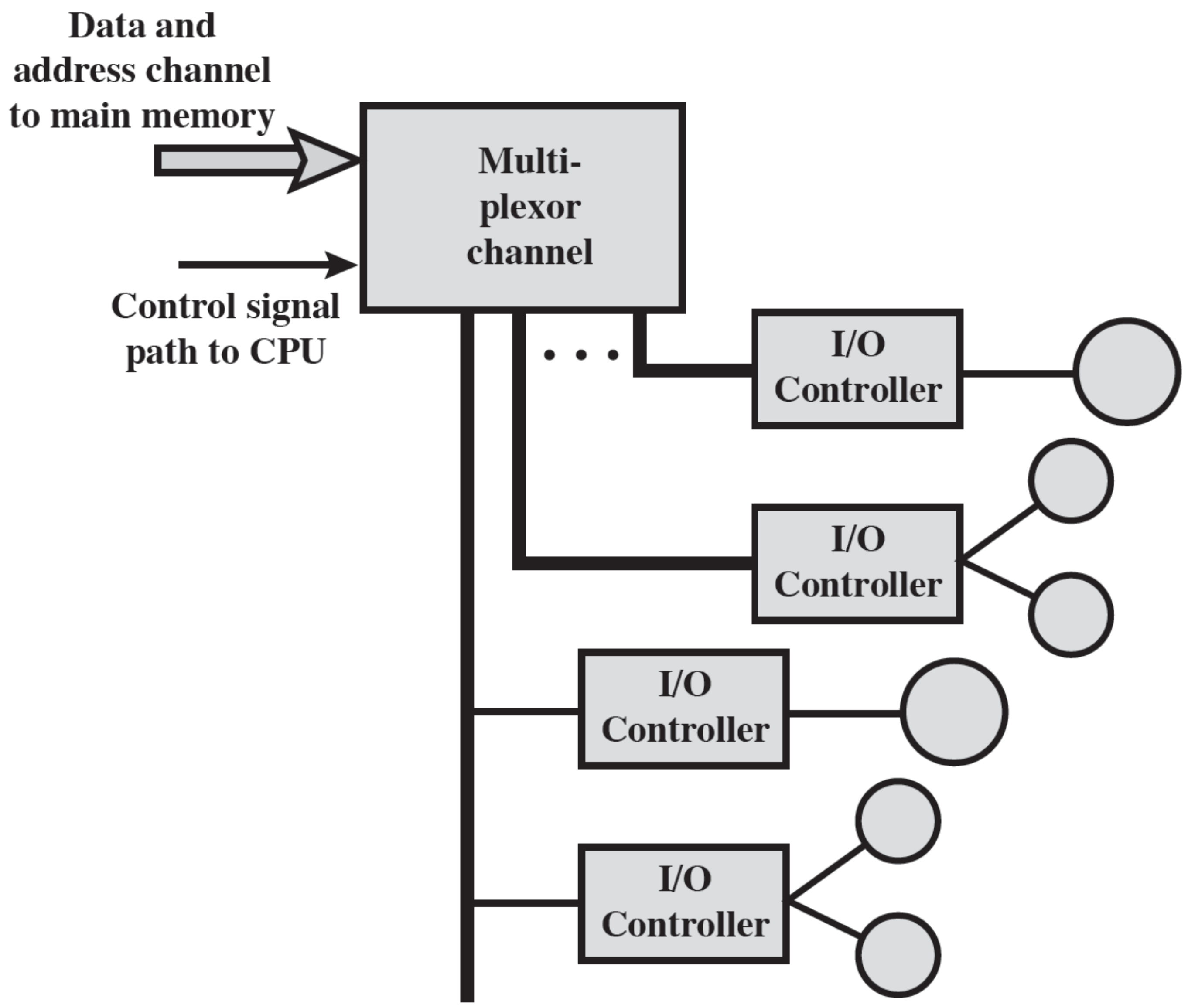
# I/O Channels/Processors

- The I/O channel represents an extension of the DMA concept
- Has the ability to execute I/O instructions
- Has complete control over I/O operations
- The CPU does not execute I/O instructions
- Stored program concept
  - I/O instructions are stored in main memory
  - executed by a special-purpose processor in the I/O channel itself
  - CPU initiates an I/O transfer by instructing the I/O channel to execute a program in memory
  - The program will specify
    - device or devices
    - the area or areas of memory for storage
    - priority
    - …

# I/O Selector Channel


Data and address channel to main memory
Control signal path to CPU
Selector channel
I/O Controller
I/O Controller
• • •

# I/O Multiplexor Channel



## Common Types of I/O Channels

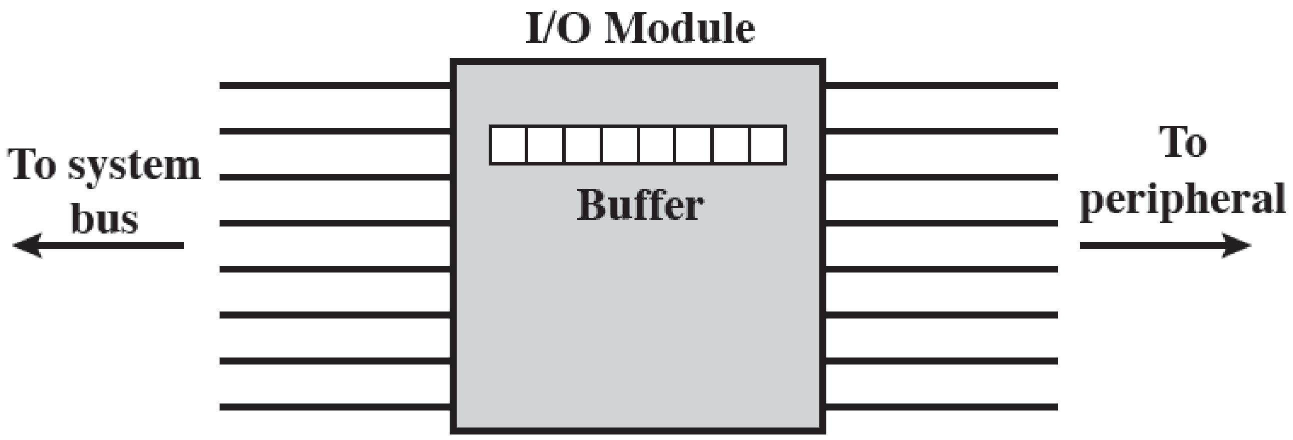
- **Selector Channel**
  - controls multiple high-speed devices
  - Is at any given time dedicated to the transfer of data with one of the devices
  - Each device, or a small set of devices, is handled by a controller
  - Thus, the I/O channel serves in place of the CPU in controlling these I/O controllers

- **Multiplexor Channel**
  - can handle I/O with multiple devices at the same time
  - For low-speed devices, a byte multiplexor accepts or transmits characters as fast as possible to multiple devices
  - Example: the resultant character stream $A_1A_2A_3A_4; B_1B_2B_3B_4; C_1C_2C_3C_4$ of three devices might be $A_1B_1C_1A_2C_2A_3B_2C_3A_4$ ...

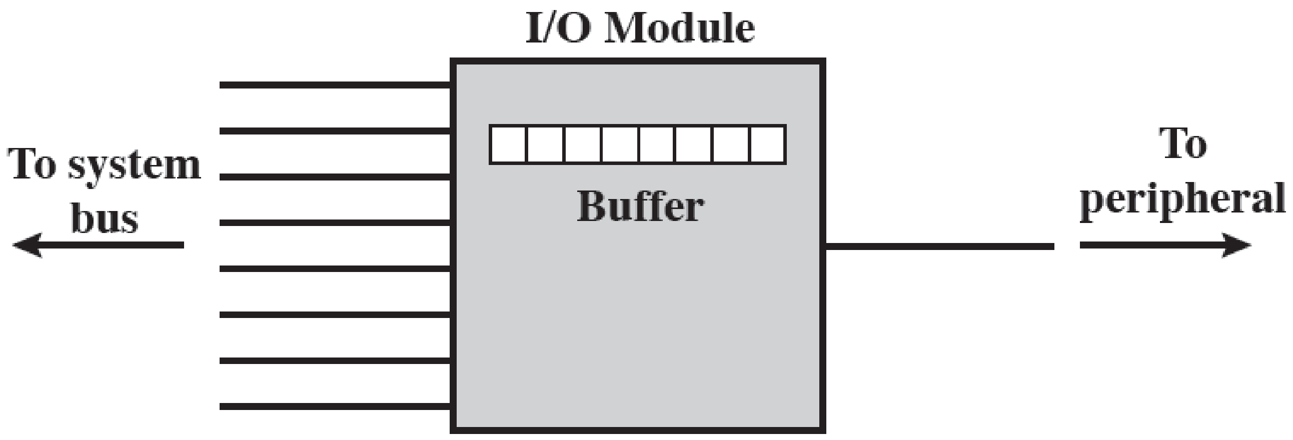
# I/O

## Lecture Content

## Serial vs. Parallel Interfaces



(a) Parallel I/O



(b) Serial I/O

## Universal Serial Bus: Features

- The computer acts as the host
- Up to 127 devices can connect to the host (directly or by hubs)
- Maximal Transfer Rates:
  - Low Speed: 1,5 MBit (USB 1.0)
  - Full Speed: 12 MBit (USB 1.0)
  - High Speed: 450 MBit (USB 2.0)
  - Superspeed: 5000 MBit (USB 3.0)
  - Superspeed+: 10000 MBit (USB 3.1)

- A USB 2.0 cable has two wires for power (+5 volts and ground) and a twisted pair of wires to carry the data.
- USB 3.0 supports bidirectional transfer

## USB Host Responsibilities

- Detecting the attachment and removal of USB devices
- Managing control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Providing power to attached USB devices

# USB Data Transfer

- Uses differential signaling
- Different Transfer Types
  - **Control**
  - **Interrupt**
  - **Bulk**
  - **Isochronous**
- Uses Non-Return-to-Zero-Inverted Encoding
- Transfer is packet based

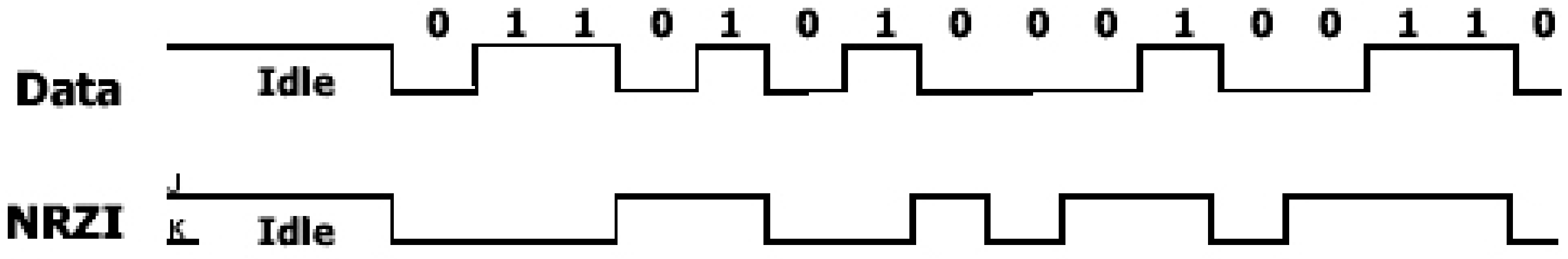
# A USB Packet

- Sync
  - 8 bits for low and full speed 32 bits for high speed
- PID (Packet ID)
  - To identify the type of packet that is being sent
- ADDR
  - The address field specifies which device the packet is designated for
  - 7 bits so allows 127 devices
- ENDP
  - Made up of 4 bits. But only 2 bits for Low speed devices
- CRC
  - 5 bit CRC for token while 16 bit CRC for data
- EOP
  - End of packet

# Non-Return-to-Zero-Inverted Encoding

- Transmitting **0**
  - Flip the value of the differential pair
- Transmitting **1**
  - Leave the value of the differential pair

- Bit Stuffing
  - After 6 consecutive 1 an additional 0 is inserted (gets removed by the receiver)



## USB: Connecting a Device

- After host detects new device on port:
  - Reset
    - Both data lines set to high for 10 ms
  - The device assigns itself the address 0
  - Host sends new address to device
  - As only one port gets activated at a time, no address conflicts

- After Assigning Address
  - Host queries for descriptor
  - Device replies with different configurations and endpoints
  - Each device has to have the endpoint 0

## USB Descriptor

- The Descriptor defines some standard properties of the device:
  - 0x03: Human Interface Device
  - 0x06: Images (E.g., Camera)
  - 0x08: Mass Storage Device
  - …
- This allows the usage of USB standard drivers
- A device can be member of several groups

## USB Endpoints

- USB devices have enumerated Endpoints
- Defined by the device, but each device must have endpoint 0
- These endpoint can be used to define transfers
- Unidirectional
  - Most commonly, a device has IN and OUT endpoints

- A device may have up to 31 endpoints (lowspeed devices have 3):
  - 15 IN and 15 OUT
  - Endpoint 0 for configuration

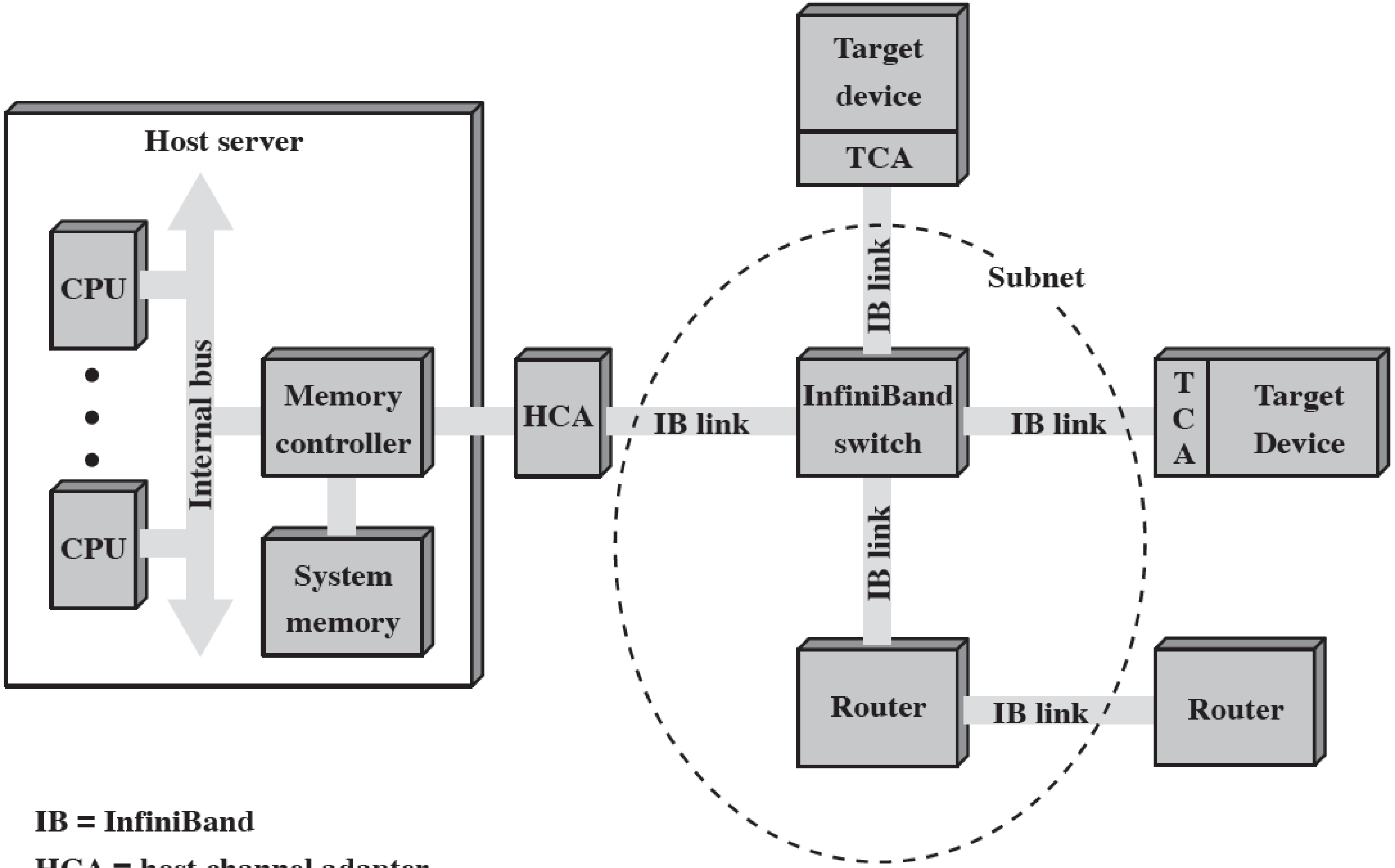
# I/O
## Lecture Content

- Introduction
- I/O Types
  - Programmed I/O
  - Interrupt-Driven
  - DMA
  - I/O Channels
- External Interfaces: USB
- InfiniBand

# InfiniBand

- Recent I/O specification aimed at the high-end server market
- First version was released in early 2001
- Standard describes an architecture and specifications for data flow among processors and intelligent I/O devices
- Enables servers, remote storage, and other network devices to be attached in a central fabric of switches and links
- The switch-based architecture can connect up to 64,000 servers, storage systems, and networking devices

# InfiniBand Switch Fabric



IB = InfiniBand

HCA = host channel adapter

TCA = target channel adapter

# InfiniBand Switch Fabric

- **Host channel adapter (HCA)**
  - Instead of a number of PCI slots, a typical server needs a single interface to an HCA that links the server to an InfiniBand switch
  - The HCA attaches to the server at a memory controller, which has access to the system bus and controls traffic between the processor and memory and between the HCA and memory
  - The HCA uses direct-memory access (DMA) to read and write memory

- **Target channel adapter (TCA)**
- **InfiniBand switch**
- **Subnet**
- **Router**

## InfiniBand Switch Fabric

- **Host channel adapter (HCA)**
- **Target channel adapter (TCA)**
  - A TCA is used to connect storage systems, routers, and other peripheral devices to an InfiniBand switch

- **InfiniBand switch**
- **Subnet**
- **Router**

## InfiniBand Switch Fabric

- **Host channel adapter (HCA)**
- **Target channel adapter (TCA)**
- **InfiniBand switch**
  - A switch provides point-to-point physical connections to a variety of devices and switches traffic from one link to another
  - Servers and devices communicate through their adapters, via the switch. The switch's intelligence manages the linkage without interrupting the servers' operation.
- **Subnet**
- **Router**

## InfiniBand Switch Fabric

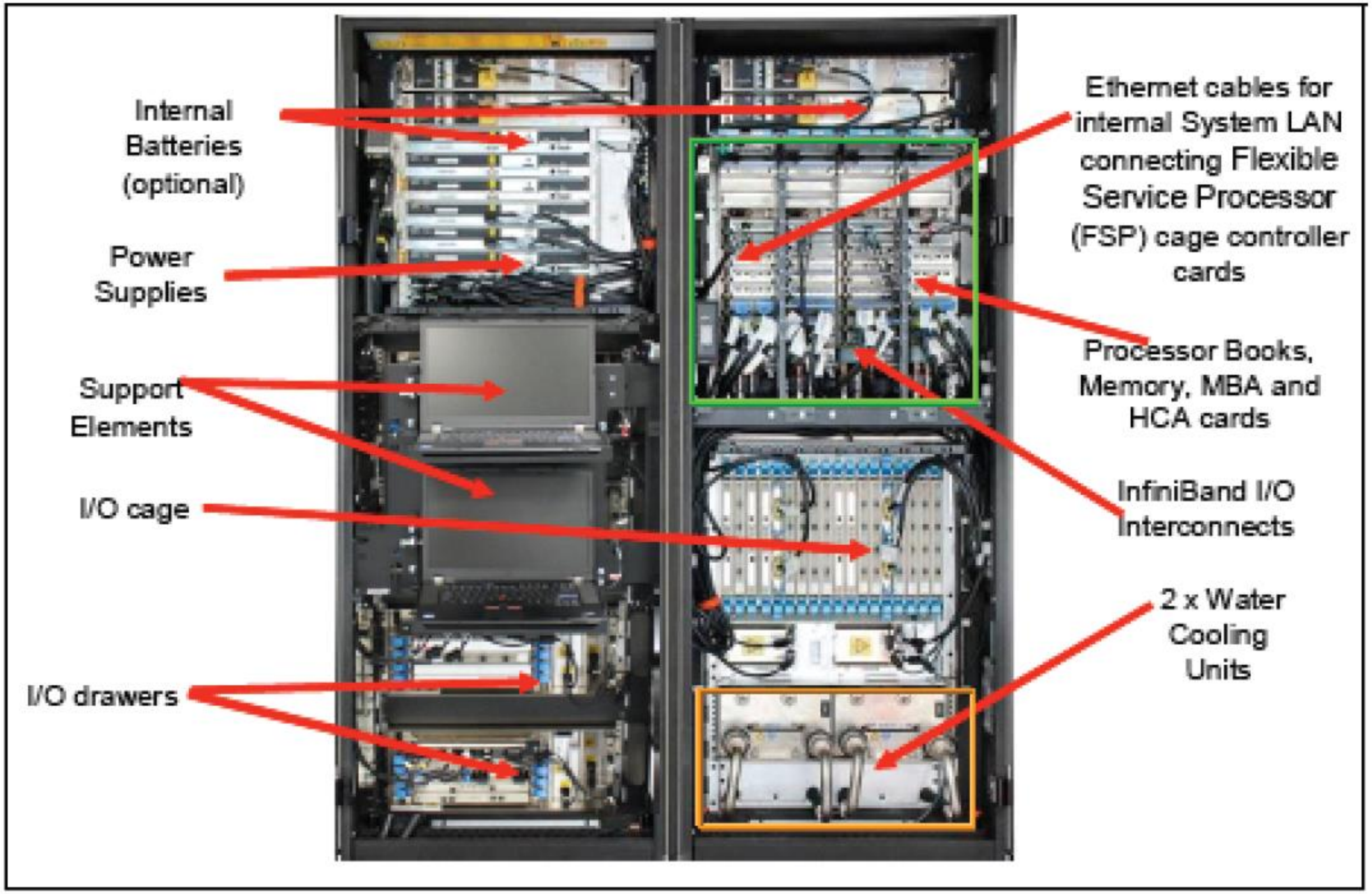
- **Host channel adapter (HCA)**
- **Target channel adapter (TCA)**
- **InfiniBand switch**
- **Subnet**
  - A subnet consists of one or more interconnected switches plus the links that connect other devices to those switches
  - Subnets allow administrators to confine broadcast and multicast transmissions within the subnet.

- **Router**

## InfiniBand Switch Fabric

- **Host channel adapter (HCA)**
- **Target channel adapter (TCA)**
- **InfiniBand switch**
- **Subnet**
- **Router**
  - Connects InfiniBand subnets, or connects an InfiniBand switch to a network, such as a local area network, wide area network, or storage area network.

## IBM zSeries


Internal Batteries (optional)
Power Supplies
Support Elements
I/O cage
I/O drawers
Ethernet cables for internal System LAN connecting Flexible Service Processor (FSP) cage controller cards
Processor Books, Memory, MBA and HCA cards
InfiniBand I/O Interconnects
2 x Water Cooling Units

## Abacus 2.0 at SDU (56 Gbit/s InfiniBand)