CMSC 611-101
Advanced Computer Architecture

Lecture 25

Performance and Interfacing I/O Devices

December 3, 2009

www.csee.umbc.edu/~younis/CMSC611/CMSC611.htm

Lecture’s Overview

❑ ***Previous Lecture:***

➔ I/O systems architecture

- I/O design issues
- I/O devices

➔ Magnetic Disk

- Access time and performance characteristics
- Theory of operation and historical trend
- Disk non-functional attributes (reliability and availability)
- Redundant array of inexpensive disks

❑ ***This Lecture***

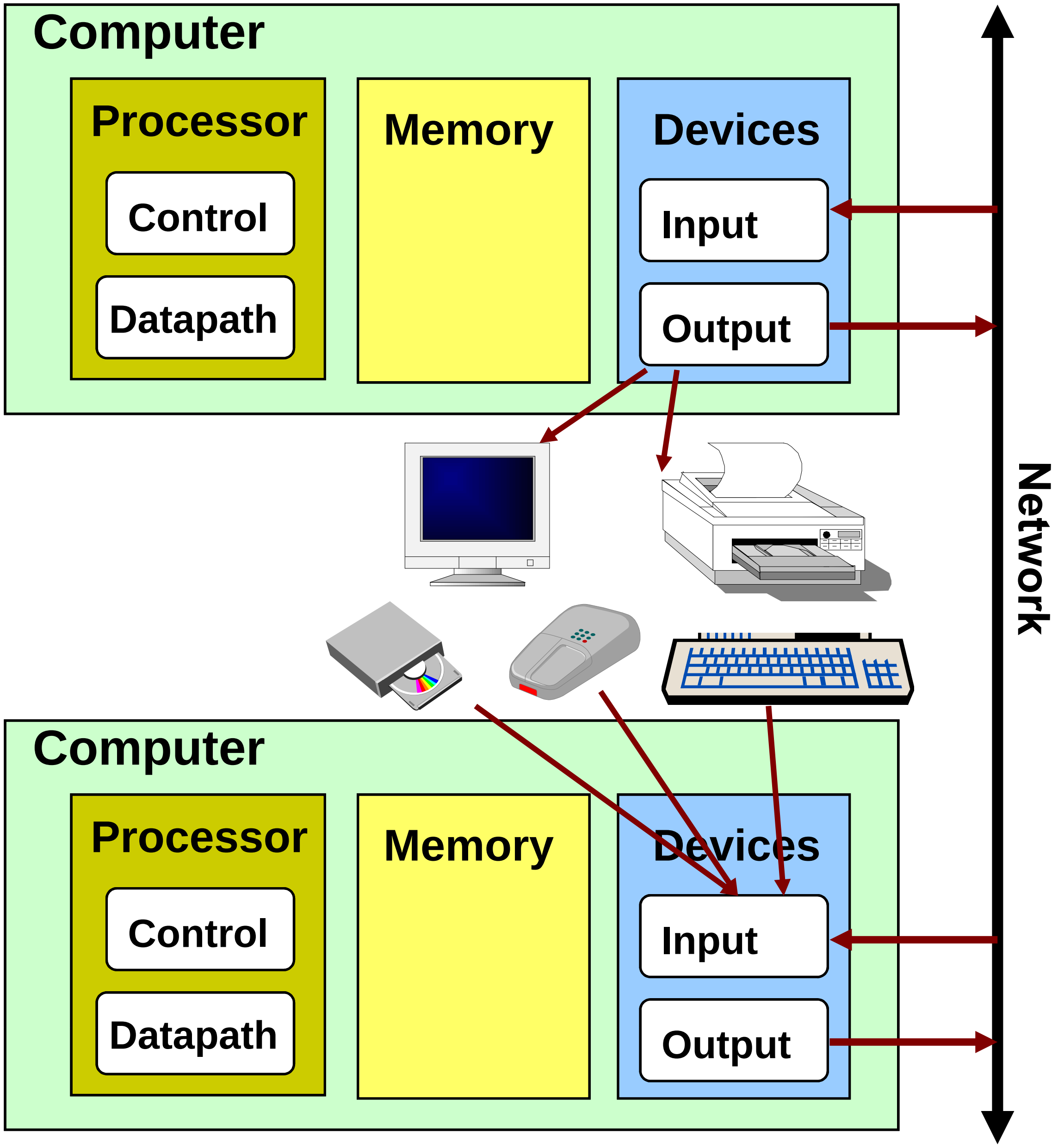
➔ Memory to processor interconnect

➔ Performance of I/O systems

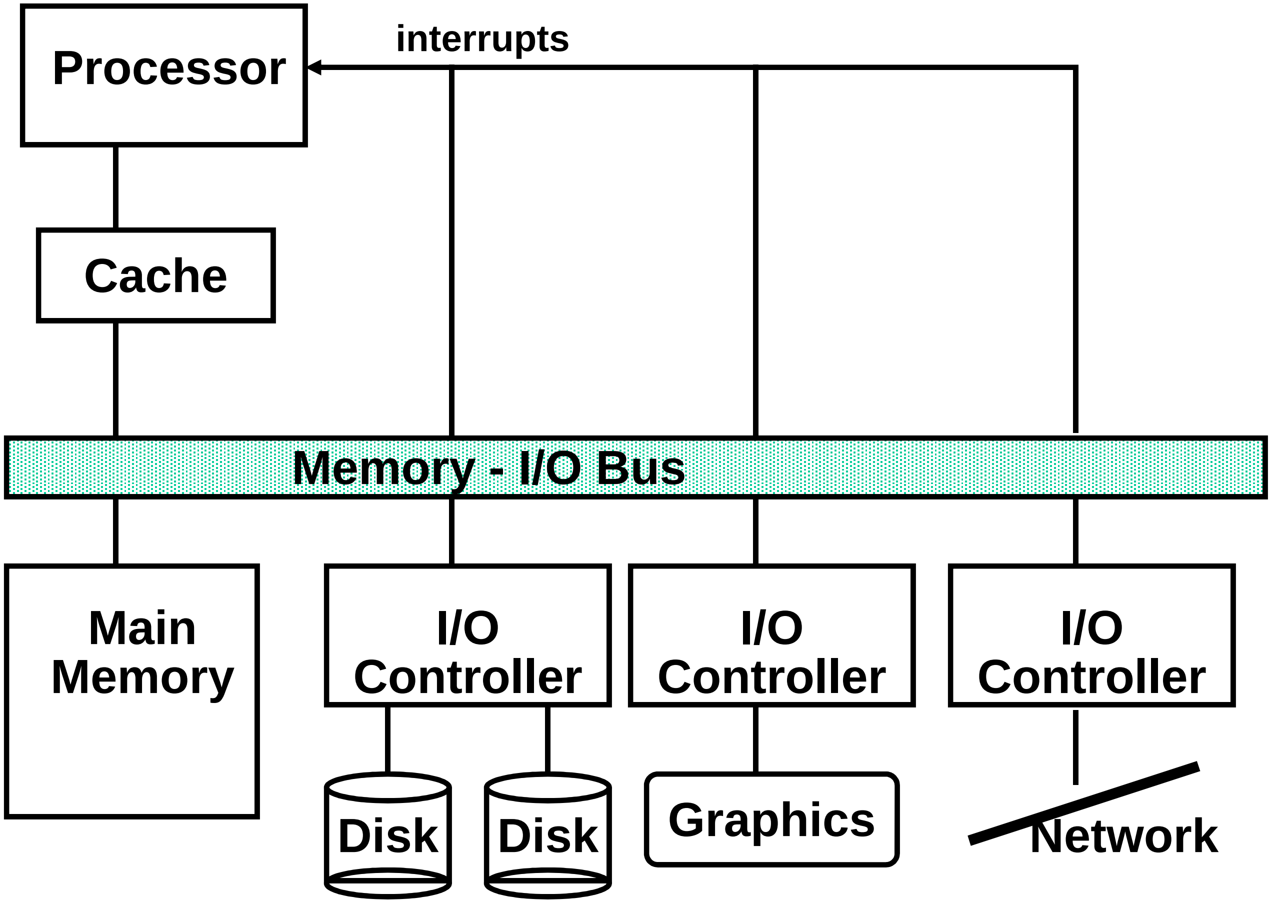
➔ Interfacing I/O devices

Computer Input/Output

- ❑ I/O Interface
  - ➔ Device drivers
  - ➔ Device controller
  - ➔ Service queues
  - ➔ Interrupt handling
- ❑ Design Issues
  - ➔ Performance
  - ➔ Expandability
  - ➔ Standardization
  - ➔ Resilience to failure
- ❑ Impact on Tasks
  - ➔ Blocking conditions
  - ➔ Priority inversion
  - ➔ Access ordering



Typical I/O System



- ❑ The connection between the I/O devices, processor, and memory are usually called (local or internal) *bus*
- ❑ Communication among the devices and the processor use both protocols on the bus and interrupts

I/O Devices' Interface

Two methods are used to address the device:

❶ *Special I/O instructions*: (Intel 80X86, IBM 370)

➔ Specify both the device number and the command word

- Device number: the processor communicates this via a set of wires normally included as part of the I/O bus
- Command word: this is usually send on the bus's data lines
- Each devices maintain status register to indicate progress

➔ Instructions are privileged to prevent user tasks from directly accessing the I/O devices

❷ *Memory-mapped I/O*: (Motorola/IBM PowerPC)

➔ Portions of the address space are assigned to I/O device

➔ Read and writes to those addresses are interpreted as commands to the I/O devices

➔ User programs are prevented from issuing I/O operations directly:

- The I/O address space is protected by the address translation

* Slide is courtesy of Dave Patterson

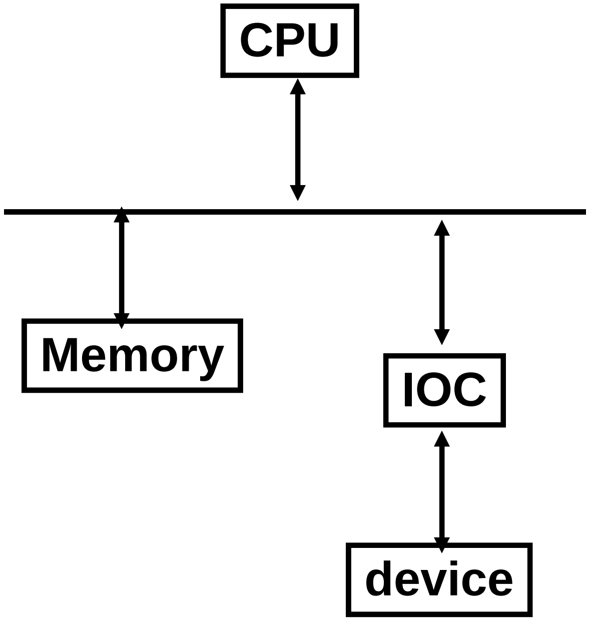
Communicating with I/O Devices

- ❑ The OS needs to know when:
  - ➔ The I/O device has completed an operation
  - ➔ The I/O operation has encountered an error
- ❑ This can be accomplished in two different ways:
  - ➔ Polling:
    - The I/O device put information in a status register
    - The OS periodically check the status register
  - ➔ I/O Interrupt:
    - An I/O interrupt is an externally stimulated event, asynchronous to instruction execution but does NOT prevent instruction completion
    - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing
    - Some processors deals with interrupt as special exceptions

These schemes requires heavy processor's involvement and suitable only for low bandwidth devices such as the keyboard

* Slide is partially a courtesy of Dave Patterson

Polling: Programmed I/O

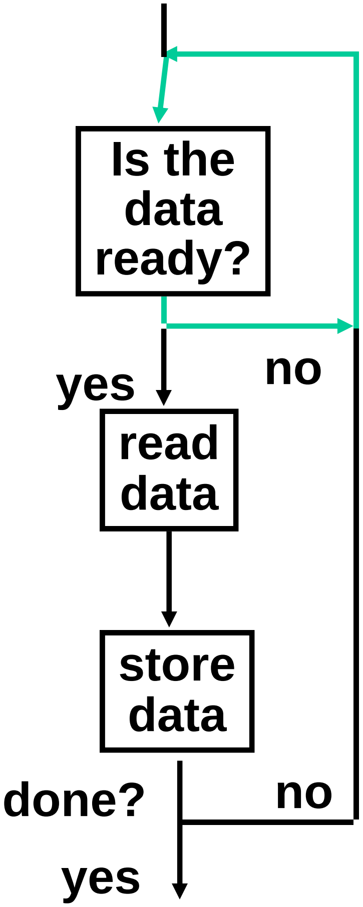


CPU

Memory

IOC

device



Is the data ready?

yes

no

read data

store data

done?

no

yes

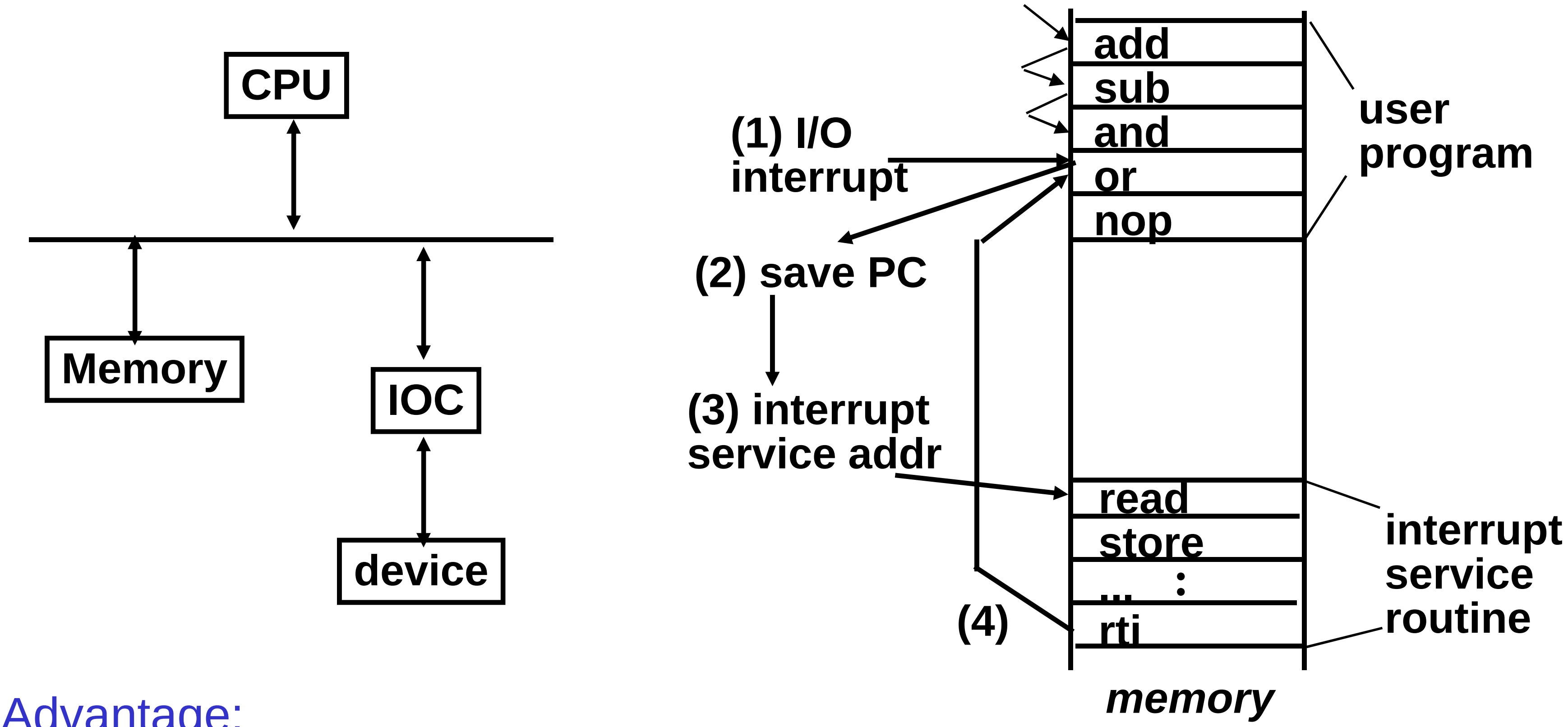
busy wait loop
not an efficient
way to use the CPU
unless the device
is very fast!

but checks for I/O
completion can be
dispersed among
computation
intensive code

- ❑ Advantage:
  - ➔ Simple: the processor is totally in control and does all the work
- ❑ Disadvantage:
  - ➔ Polling overhead can consume a lot of CPU time

* Slide is courtesy of Dave Patterson

Interrupt Driven Data Transfer



- ❑ Advantage:
  - ➔ User program progress is only halted during actual transfer
- ❑ Disadvantage: special hardware is needed to:
  - ➔ Cause an interrupt (I/O device)
  - ➔ Detect an interrupt (processor)
  - ➔ Save the proper states to resume after the interrupt (processor)

* Slide is courtesy of Dave Patterson

I/O Interrupt vs. Exception

- ❑ An I/O interrupt is just like the exceptions except:
  - ➔ An I/O interrupt is asynchronous
  - ➔ Further information needs to be conveyed
  - ➔ Typically exceptions are more urgent than interrupts

- ❑ An I/O interrupt is asynchronous with respect to instruction execution:
  - ➔ I/O interrupt is not associated with any instruction
  - ➔ I/O interrupt does not prevent any instruction from completion
    - You can pick your own convenient point to take an interrupt

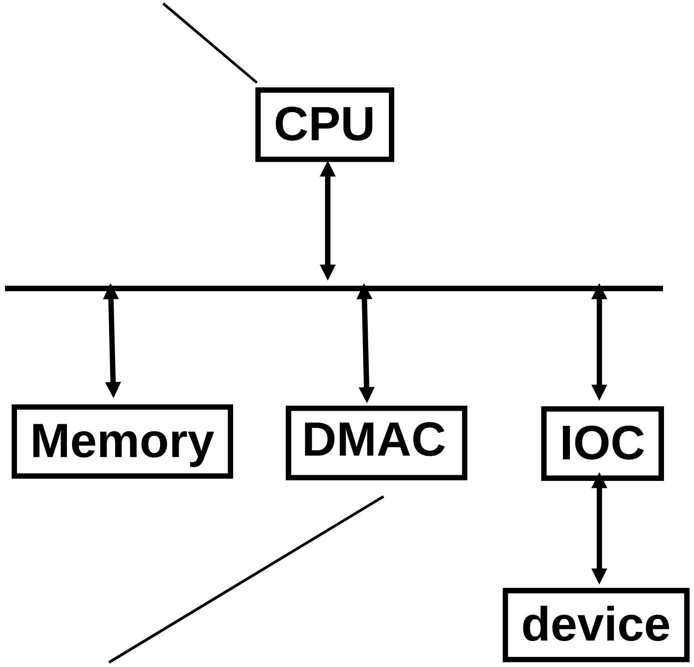
- ❑ I/O interrupt is more complicated than exception:
  - ➔ Needs to convey the identity of the device generating the interrupt
  - ➔ Interrupt requests can have different urgencies:
    - Interrupt request needs to be prioritized
    - Priority indicates urgency of dealing with the interrupt
    - high speed devices usually receive highest priority

* Slide is courtesy of Dave Patterson

Direct Memory Access

- Direct Memory Access (DMA):
  - External to the CPU
  - Use idle bus cycles (*cycle stealing*)
  - Act as a master on the bus
  - Transfer blocks of data to or from memory without CPU intervention
  - Efficient for large data transfer, e.g. from disk
  - ✉ Cache usage allows the processor to leave enough memory bandwidth for DMA
- How does DMA work?:
  - CPU sets up and supply device id, memory address, number of bytes
  - DMA controller (DMAC) starts the access and becomes the bus master
  - For multiple byte transfer, the DMAC increment the address
  - DMAC interrupts the CPU upon completion

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".



DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.

For multiple bus system, each bus controller often contains DMA control logic

* Figure is courtesy of Dave Patterson

DMA Problems

❶ **With virtual memory systems**: (pages would have physical and virtual addresses)

- ➔ Physical pages re-mapping to different virtual pages during DMA operations
- ➔ Multi-page DMA cannot assume consecutive addresses

Solutions:

- ➔ Allow virtual addressing based DMA
  - ⇧ Add translation logic to DMA controller
  - ⇧ OS allocated virtual pages to DMA prevent re-mapping until DMA completes
- ➔ Partitioned DMA
  - ⇧ Break DMA transfer into multi-DMA operations, each is single page
  - ⇧ OS chains the pages for the requester

❷ **In cache-based systems**: (there can be two copies of data items)

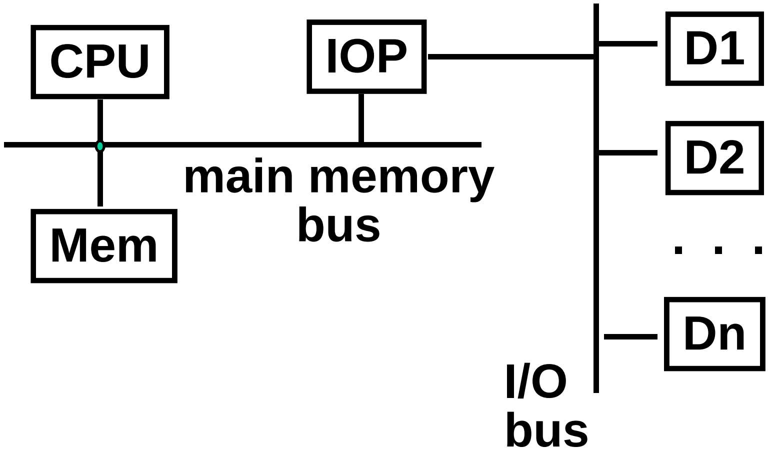
- ➔ Processor might not know that the cache and memory pages are different
- ➔ Write-back caches can overwrite I/O data or makes DMA to read wrong data

Solutions:

- ➔ Route I/O activities through the cache
  - ⇧ Not efficient since I/O data usually is not demonstrating temporal locality
- ➔ OS selectively invalidates cache blocks before I/O read or force write-back prior to I/O write
  - ⇧ Usually called cache *flushing* and requires hardware support

DMA allows another path to main memory with no cache and address translation

I/O Processor



CPU

IOP

D1

D2

. . .

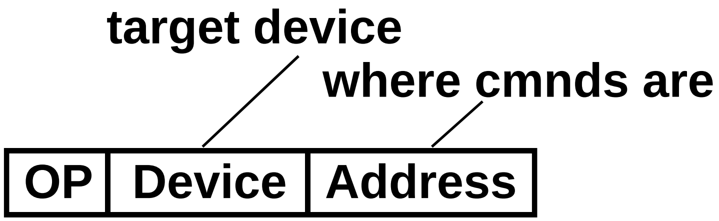
Dn

Mem

main memory bus

I/O bus

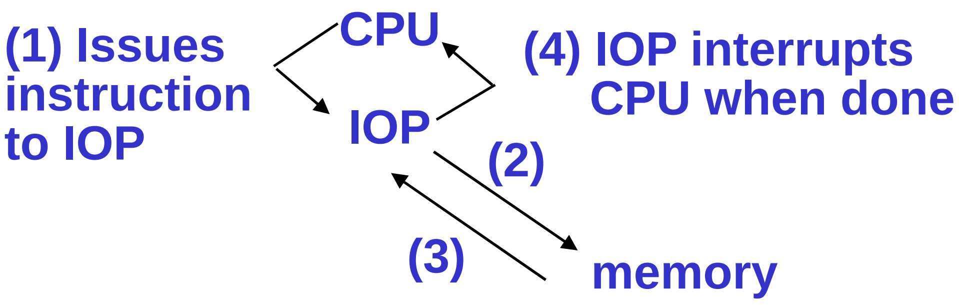
- ➔ An I/O processor (IOP) offload the CPU
- ➔ Some of the new processors, e.g. Motorola 860, include special purpose IOP for serial communication



target device

where cmnds are

OP	Device	Address



(1) Issues instruction to IOP

CPU

IOP

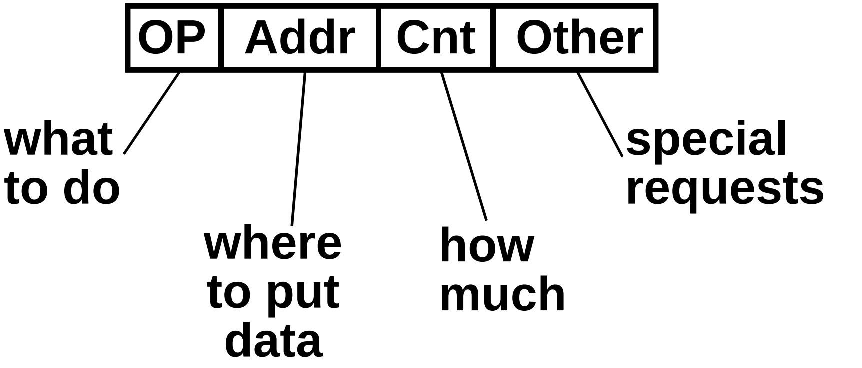
(4) IOP interrupts CPU when done

(2)

(3)

memory

IOP looks in memory for commands



OP	Addr	Cnt	Other

what to do

where to put data

how much

special requests

Device to/from memory transfers are controlled by the IOP directly.

IOP steals memory cycles.

* Slide is courtesy of Dave Patterson

Operating System’s Role

❑ Operating system acts as an interface between I/O hardware and programs

❑ Important characteristics of the I/O systems:

➔ The I/O system is shared by multiple program using the processor

➔ I/O systems often use interrupts to communicate information about I/O

- Interrupts must be handled by OS because they cause a transfer to supervisor mode

➔ The low-level control of an I/O device is complex:

- Managing a set of concurrent events
- The requirements for correct device control are very detailed

❑ Operating System’s Responsibilities

➔ Provide protection to shared I/O resources

- Guarantees that a user’s program can only access Provides abstraction for accessing I/O devices
- Supply routines that handle low-level device operation

➔ Handles the interrupts generated by I/O devices

➔ Provide equitable access to the shared I/O resources

- All user programs must have equal access to the I/O resources

➔ Schedule accesses in order to enhance system throughput allowed set of I/O services

I/O System Performance

❑ I/O System performance depends on many aspects of the system ("limited by weakest link in the chain"):

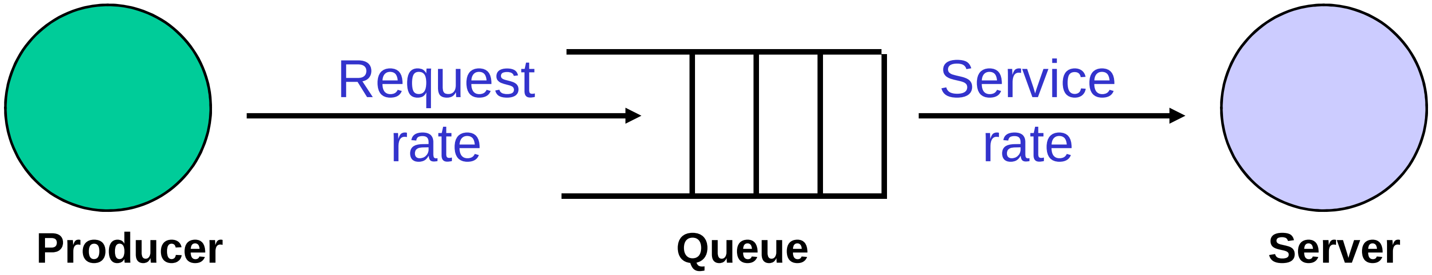
- ➔ The CPU
- ➔ The memory system:
  - Internal and external caches
  - Main Memory
- ➔ The underlying interconnection (buses)
- ➔ The I/O controller
- ➔ The I/O device
- ➔ The speed of the I/O software (Operating System)
- ➔ The efficiency of the software's use of the I/O devices

❑ Two common performance metrics:

- ➔ *Throughput*: I/O bandwidth
- ➔ *Response time*: Latency

* Slide is courtesy of Dave Patterson

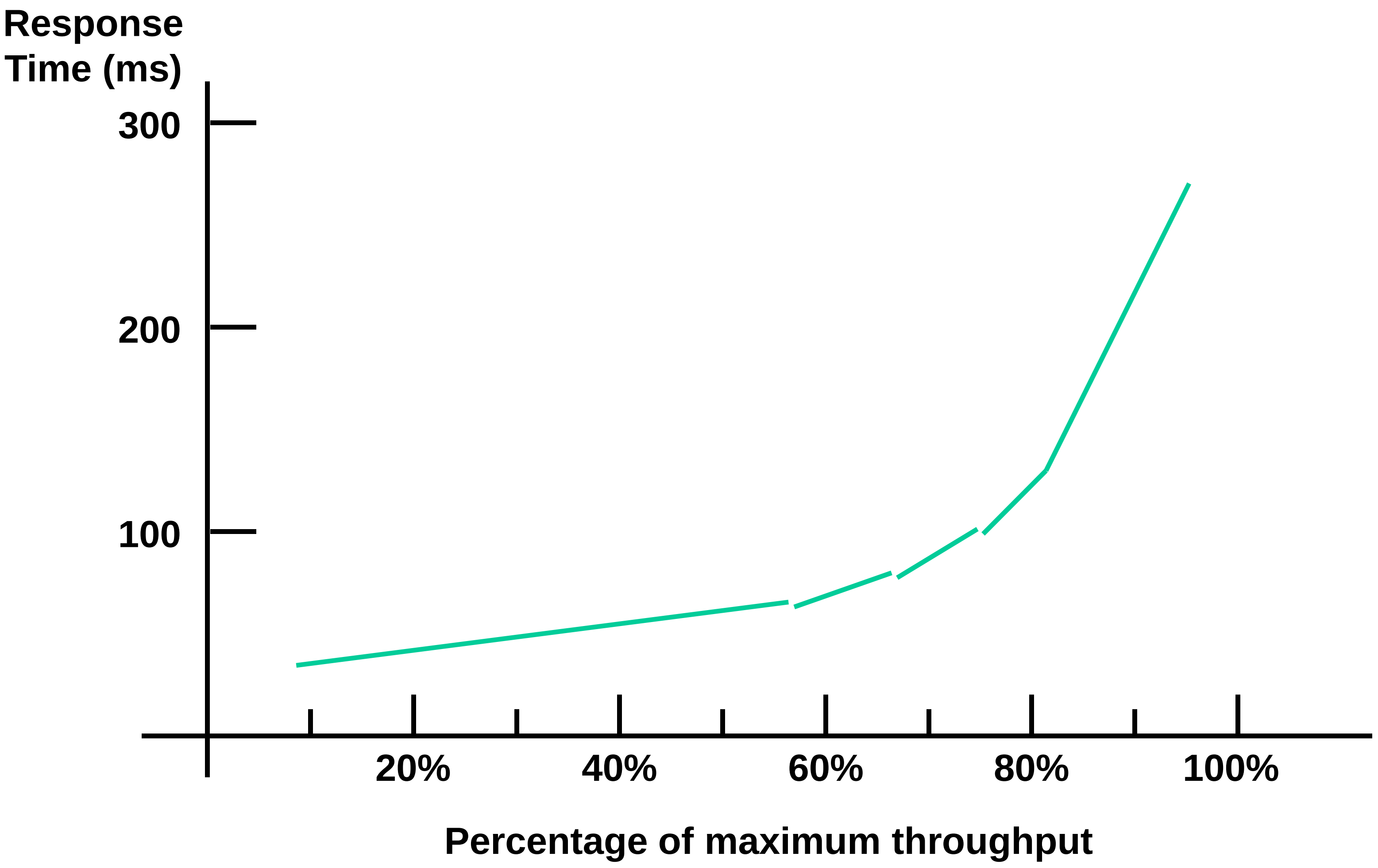
Simple Producer-Server Model



- ❑ *Throughput*:
  - ➔ The number of tasks completed by the server in unit time
  - ➔ In order to get the highest possible throughput:
    - The server should never be idle
    - The queue should never be empty
- ❑ *Response time*:
  - ➔ Begins when a task is placed in the queue
  - ➔ Ends when it is completed by the server
  - ➔ In order to minimize the response time:
    - The queue should be empty
    - The server will be idle

* Slide is courtesy of Dave Patterson

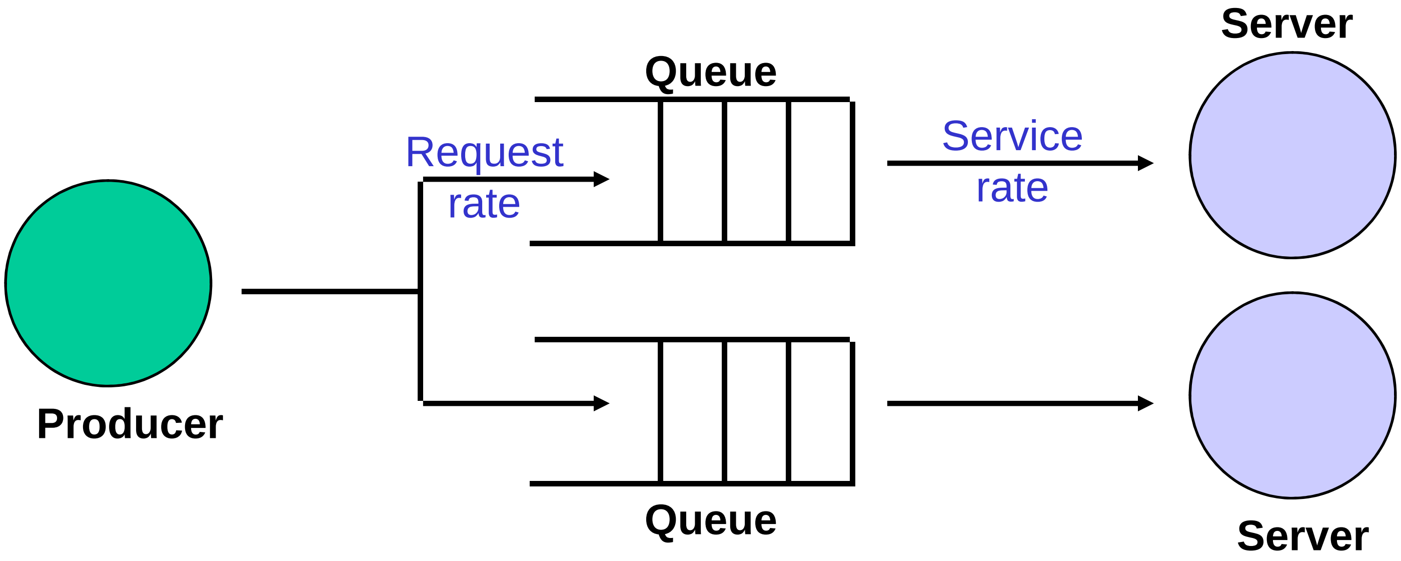
Throughput versus Respond Time



Low response time is user-desirable but leads to low throughput that is system-*Un*desirable (low device utilization)

* Slide is courtesy of Dave Patterson

Throughput Enhancement

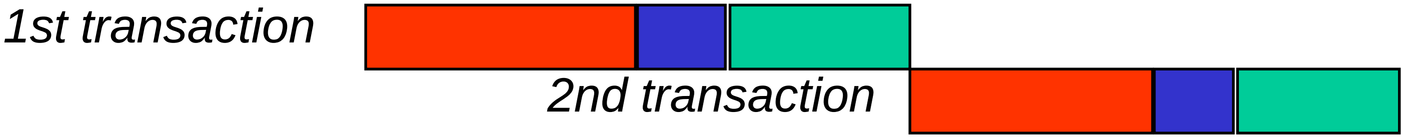


- ❑ In general throughput can be improved by throwing more hardware at the problem
- ❑ Response time is much harder to reduce:
  - ➔ Average response time = average queuing time + average service time
- ❑ Estimating Queue Length:
  - ➔ Utilization = U = Request Rate / Service Rate
  - ➔ Mean Queue Length = U / (1 - U)
  - ➔ Average queuing time = Mean Queue Length / request rate

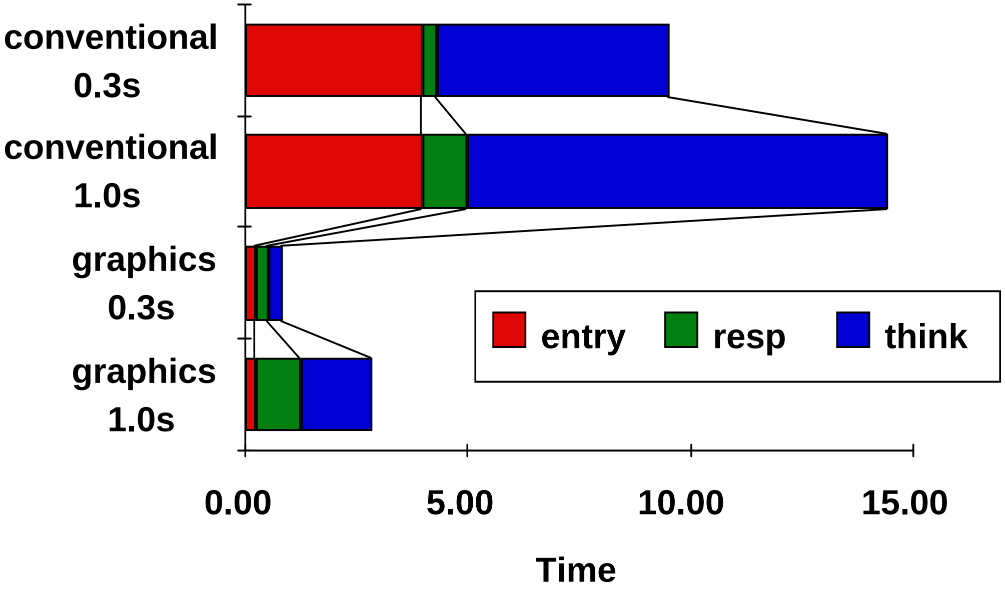
* Slide is courtesy of Dave Patterson

Response Time vs. Productivity

- Interactive environments: assume each interaction (*transaction*) has 3 parts:
  - ➔ *Entry Time*: time for user to enter command
  - ➔ *System Response Time*: time between user entry & system replies
  - ➔ *Think Time*: Time from response until user begins next command



- An empirical study was conducted to capture the effect of response time on transaction time



- ➢ 0.7sec off response saves 4.9 sec (34%) and 2.0 sec (70%) total time per transaction => greater productivity
- ➢ Another study: everyone gets more done with faster response, but novice with fast response = expert with slow

* Slide is courtesy of Dave Patterson

I/O Benchmarks

- ❑ Processor benchmarks have aimed at response time for fixed sized problem
- ❑ I/O benchmarks typically measure throughput, possibly with upper limit on response times (or 90% of response times)

Benchmark	*Size of Data*	*% Time I/O*	*Year*
I/OStones	1 MB	26%	1990
Andrew	4.5 MB	4%	1988

- ➔Not much time in I/O
- ➔Not considering main memory

Self-Scaling Benchmark

- ❑ Automatically and dynamically increase aspects of workload to match characteristics of system measured
- ❑ Suitable for wide range of current & future applications
- ❑ Famous self-scaling benchmarks include:
  - ➔Transaction Processing: TPC-A, TPC-B, TPC-C, TPC-D with different workload for type of application
  - ➔NFS: SPEC SFS offers a mix of read, read & other file-related operations

Connecting I/O Devices

- ❑ A bus is a shared communication link, which uses one set of wires to connect multiple subsystems
- ❑ The two major advantages of the bus organization are:
  - ➔ *Versatility*: adding new devices and moving current ones among computers
  - ➔ *Low cost*: single set of wires are shared in multiple ways
- ❑ The major disadvantage of a bus is that it creates a communication bottleneck possibly limiting throughput
- ❑ The maximum bus speed is largely limited by physical factors: the length of the bus and the number of devices
- ❑ Increasing bus bandwidth (*throughput*) can be increased using buffering which slow down bus access (*response time*)
- ❑ A bus generally contains:
  - ➔ *control lines*: requests and acknowledge signals
  - ➔ *data lines*: data, addresses, commands
- ❑ A bus protocol is enforced to define the semantics of the bus transaction and arbitrate bus usage

Bus Transactions

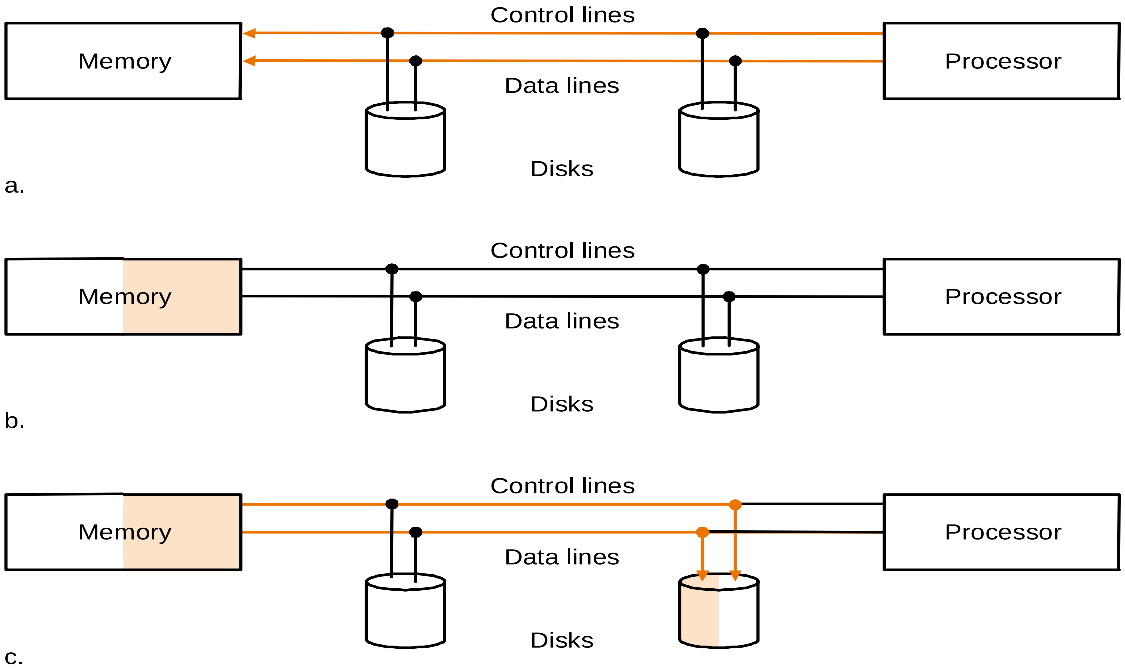
A bus transaction includes two parts:

❶ sending the address

❷ receiving or sending data

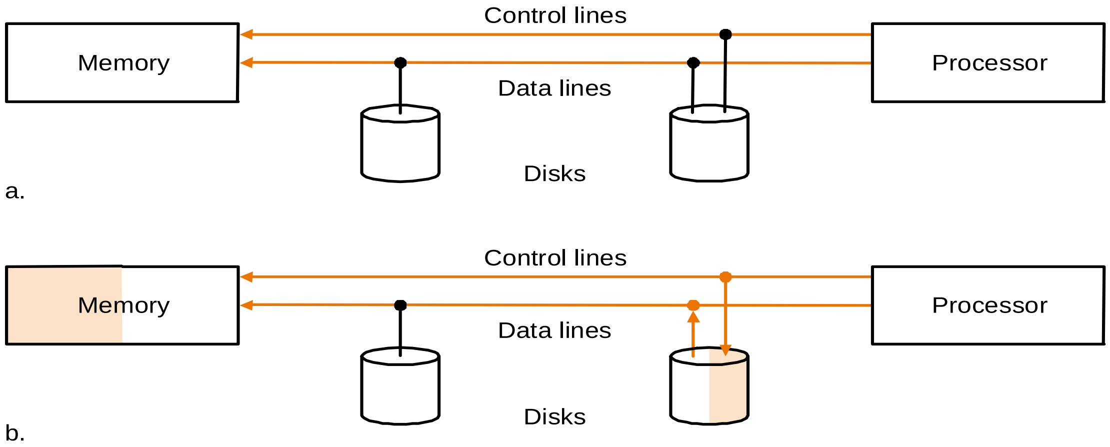
__Read transaction:__

transfers data from memory to processor or output device



__Write transaction:__

transfers data from processor or an input device to memory



Types of Buses

➊ ***Processor-memory (local) bus:***

- ❑ Generally short and high speed to maximize the bandwidth

➋ ***I/O Bus***

- ❑ Lengthy and supportive to multiple data rates and devices
- ❑ Do not interface directly to memory but use local or backplane bus
- ❑ Must handle wide range of device latency, bandwidth and characteristics

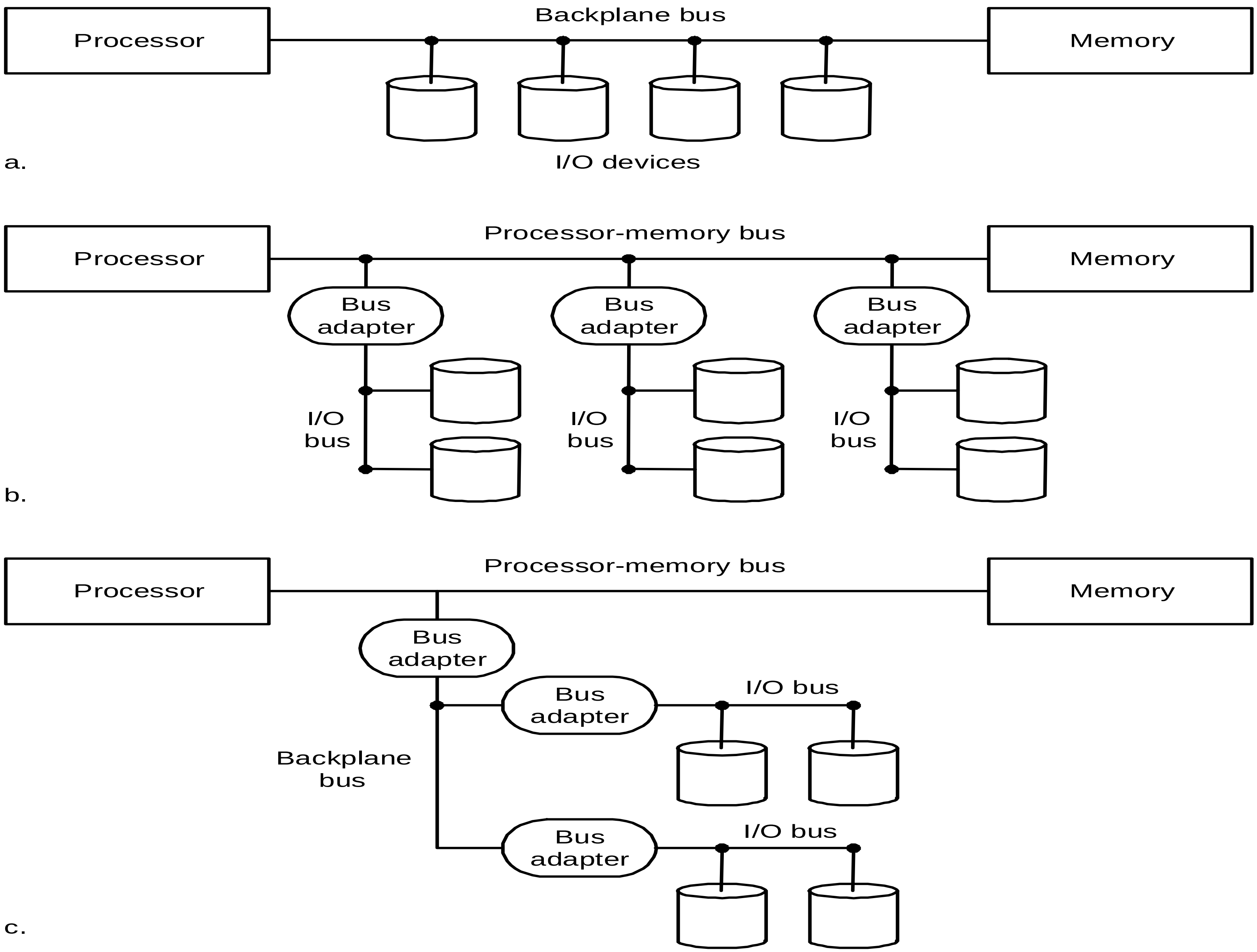
➌ ***Backplane Bus***

- ❑ Received that name because they lay in the back of the chassis structure
- ❑ Allows memory, processor and I/O devices to be connected
- ❑ Requires additional logic to interface to local and I/O buses

❑ Local buses are usually design-specific while I/O and backplane buses are portable and often follow industry-recognized standard

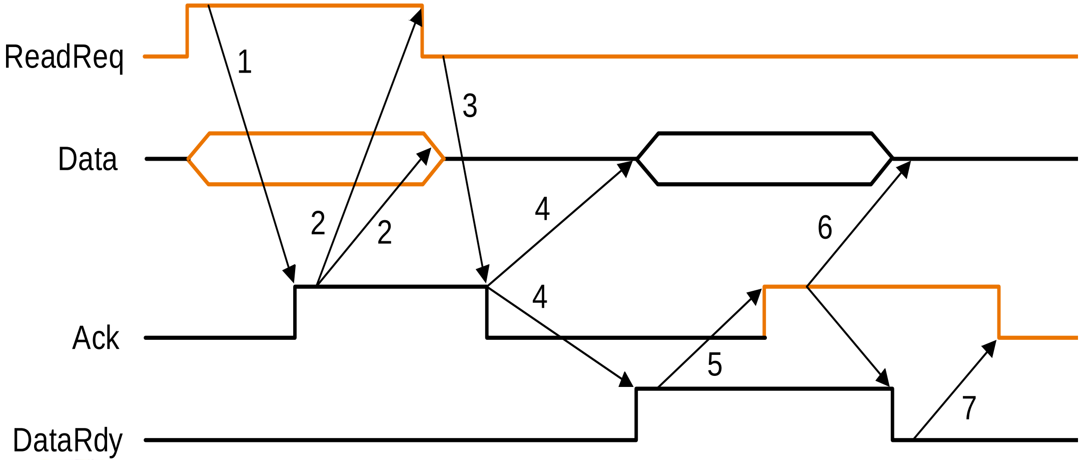
❑ A System can use one backplane or a combination of three buses to link memory, processor and the various I/O devices

Bus Configurations



Synchronous vs. Asynchronous Buses

- **Synchronous Bus:**
  - ➔ Clock based protocol and time-based control lines
  - ➔ Simple interface logic and fast bus operations
  - ➔ Every device on the bus must run at the same clock rate
  - ➔ Because clock skew, synchronous busses cannot be long
  - ➔ Local buses are often synchronous
- **Asynchronous Bus:**
  - ➔ Not clock-based → can accommodate a wide variety of devices
  - ➔ Not limited in length because of clock skew
  - ➔ Uses a handshaking protocol to ensure coordination among communicating parties
  - ➔ Requires additional control lines and logic to manage bus transactions
- **Example:**



1. Memory read address when it sees ReadReq
2. I/O device sees Ack and release ReadReq
3. Memory drops Ack when ReadReq is low
4. Memory sets DataRdy when data is available
5. I/O device reads data and set Ack
6. Memory drops DataRdy when sees Ack.
7. I/O device drops Ack When DataRdy is low

Bus Performance (An Example)

One synchronous bus has a clock cycle time of 50 ns with each bus transmission taking 1 clock cycle. Another asynchronous bus requires 40 ns per handshake. The data portion of both is 32-bit wide. Find the bandwidth of each bus for one-word reads from 200-ns memory.

Answer:

The step for the synchronous bus are:

$$\left.\begin{array}{l}\text{1. Send the address to memory: 50 ns}\\ \text{2. Read the memory: 200 ns}\\ \text{3. Send the data to the device: 50 ns}\end{array}\right\}\ \text{300 ns}$$

The maximum bandwidth is 4 bytes every 300 ns $\Rightarrow \dfrac{4\text{ bytes}}{300\text{ ns}} = \dfrac{4\text{ MB}}{0.3\text{ sec}} = 13.3\text{ MB/Sec}$

The step for the asynchronous bus are:

$$\left.\begin{array}{l}\text{1. Memory read address when seeing ReadReq : 40 ns}\\ \text{2,3,4. Data ready \& handshake: max(3}\times\text{40 ns, 200 ns)= 200 ns}\\ \text{5,6,7. Read \& Ack. : 3 }\times\text{ 40 ns = 120 ns}\end{array}\right\}\ \text{360 ns}$$

The maximum bandwidth is 4 bytes every 360 ns $\Rightarrow \dfrac{4\text{ bytes}}{360\text{ ns}} = \dfrac{4\text{ MB}}{0.36\text{ sec}} = 11.1\text{MB/Sec}$

Increasing Bus Bandwidth

❑ Much of bus bandwidth is decided by the protocol and timing characteristics

❑ The following are other factors for increasing the bandwidth:

Data bus width:

➔ Transfers multiple words requires fewer bus cycles

➔ Increases the number of bus lines

Multiplexing Address & data line:

➔ Uses separate lines for data and address speeds up transactions

➔ Simplifies the bus control logic

➔ Increases the number of bus lines

Block transfer:

➔ Transfers multiple words in back-to-back bus cycles without releasing the bus or sending new address

➔ Increases response time since transactions will be longer

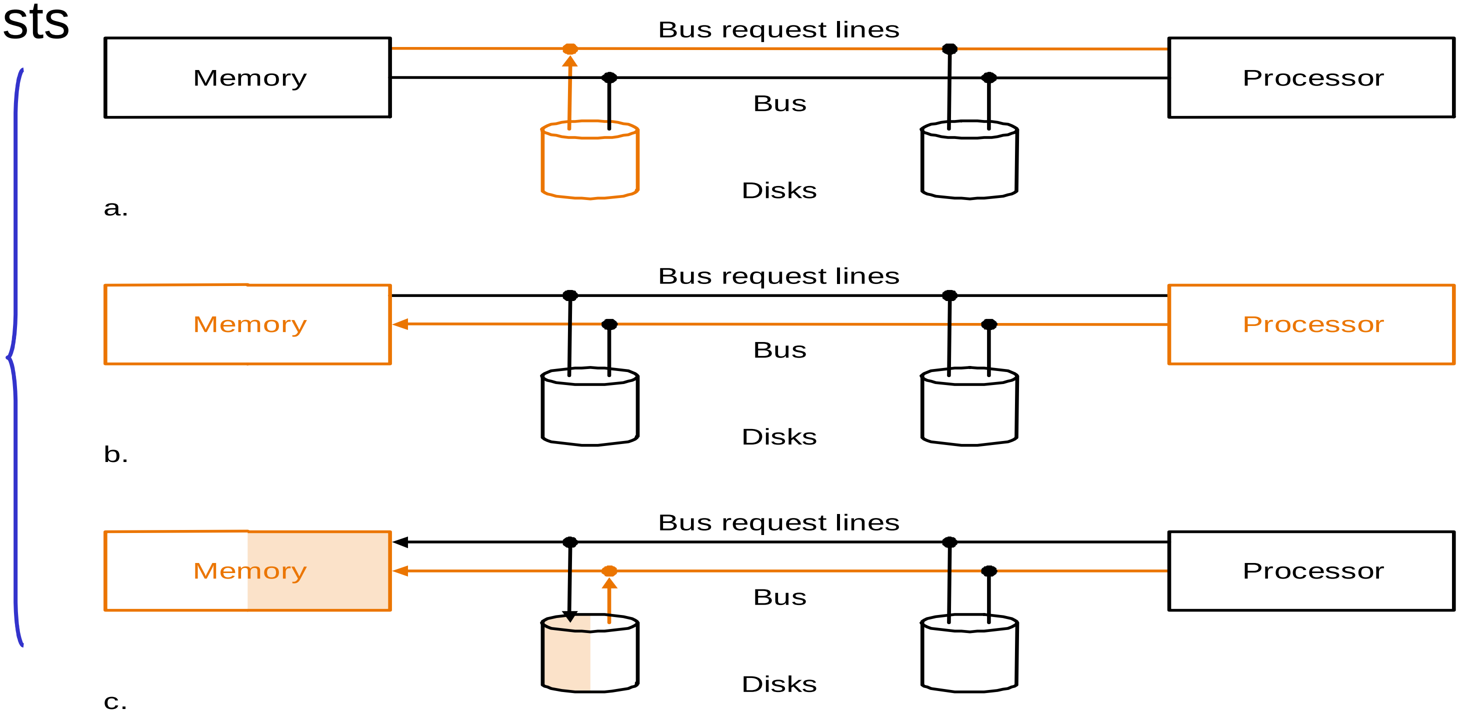
➔ Increases complexity of the bus control logic

Increasing the number of bus lines has very significant impact on the bus cost

Bus Access

- I/O should occur without processor's continuous and low-level involvement
- A bus master, e.g. processor, controls access to the bus and initiates and manage all bus requests
- A bus slave, e.g. memory, only responds to access requests but never initiate its own requests

Single master bus transaction



- A single bus master is very simple to implement but requires the involvement of the processor in every transaction
- Multi-master buses requires arbitration to coordinate bus usage among potential access requesters

Bus Arbitration

- ❑ Bus arbitration coordinates bus usage among multiple devices using *request*, *grant*, *release* mechanism
- ❑ Arbitration usually tries to balance two factors in choosing the granted device:
  - ➔ Devices with high *bus-priority* should be served first
  - ➔ Maintaining *fairness* to ensure that no device will be locked out from the bus
- ❑ Arbitration time is an overhead

  ⇒ should be minimized to expedite bus access

Arbitration Schemes

Distributed arbitration by self-selection: (e.g. NuBus used on Apple Macintosh)

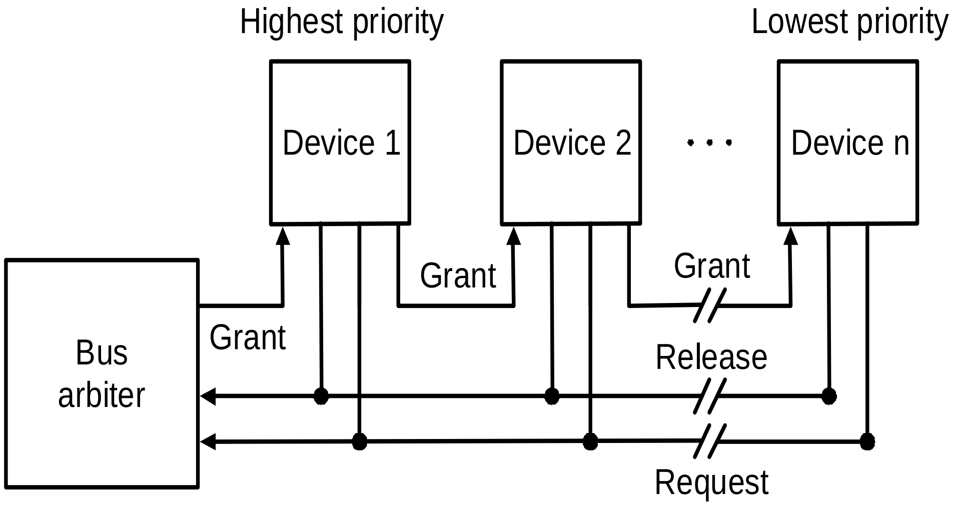
- ➔ Uses multiple request lines for devices
- ➔ Devices requesting the bus determine who will be granted access
- ➔ Devices associate a code with their request indicating their priority
- ➔ Low priority devices leave the bus if they notice a high priority requester

Distributed arbitration by collision detection: (e.g. Ethernet)

- ➔ Devices independently request the bus and assume access
- ➔ Simultaneous requests results in a collision
- ➔ A scheme for selecting among colliding parties is used

Arbitration Schemes (Cont.)

Daisy chain arbitration: (e.g. VME bus)



- ➔ The bus grant line runs through devices from highest priority to lowest
- ➔ High priority devices simply intercept the grant signal and prevent the low priority device from seeing the signal
- ➔ Simple to implement (use edge triggered bus granting)
- ➔ Low priority devices can starve (some designs prevent bus usage in two successive clock cycles)
- ➔ Limits the bus speed (grant signals take long to reach the last device in the chain)
- ➔ Allow fault propagation (failure of a high priority device might lock the bus)

Centralized, parallel arbitration: (e.g. PCI bus)

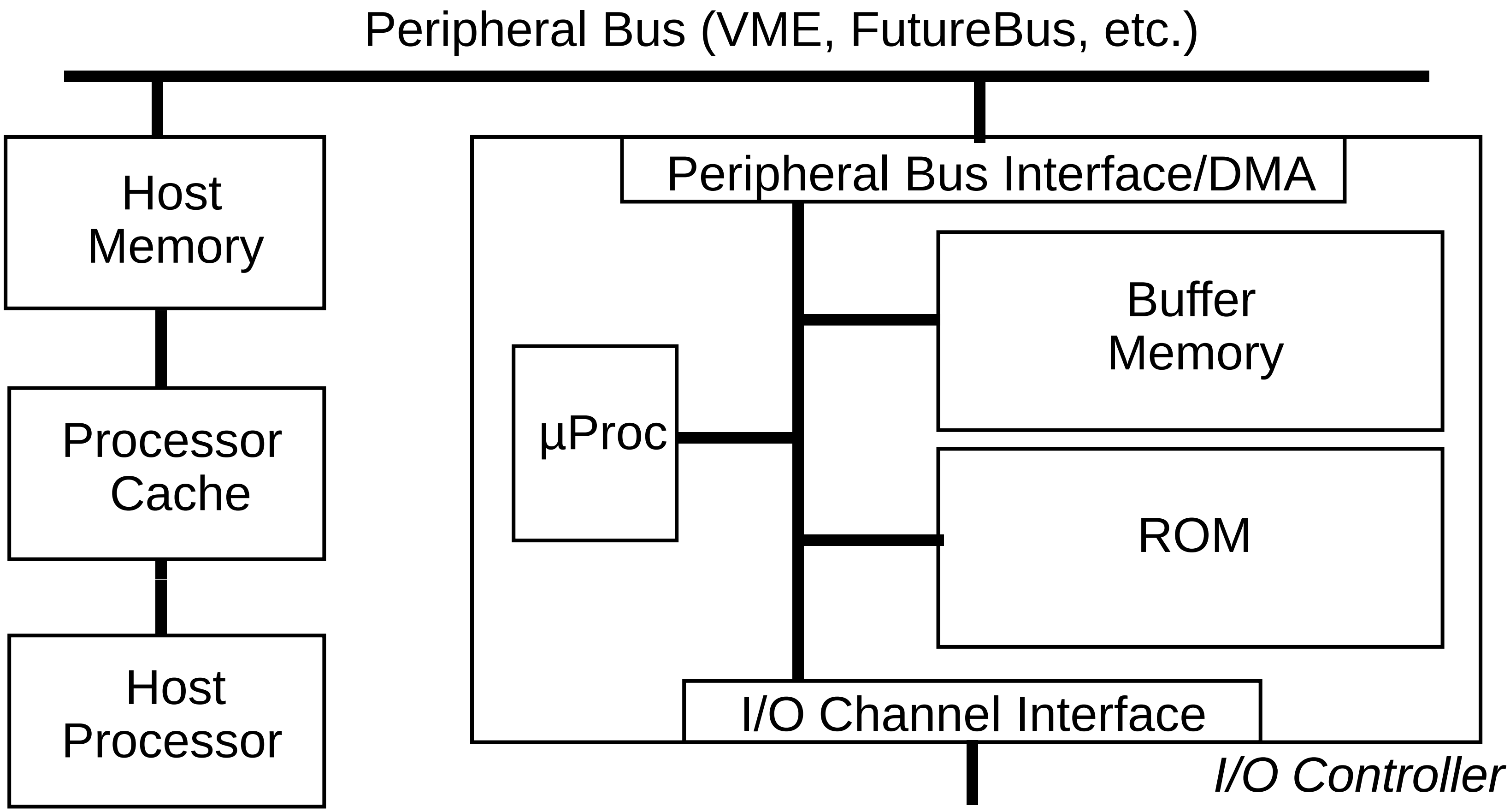
- ➔ Uses multiple request-lines and devices independently request bus (one line / device)
- ➔ A centralized arbiter selects a device and notifies it to be the bus master
- ➔ The arbiter can be a bottleneck for bus usage

Bus Standards

- I/O bus serves as a way of expanding the machine and connecting new peripherals
- Standardizing the bus specifications ensure compatibility and portability of peripherals among different computers
- Popularity of a machine can make its I/O bus a de facto standard, e.g. IBM PC-AT bus
- Examples of widely known bus standards are Small Computer Systems Interface (SCSI), and Peripheral Computer Interface (PCI)

Characteristics	PCI	SCSI
Bus type	Backplane	I/O
Basic data bus width	32-64	8-32
Address/data multiplexied?	Yes	Yes
Single / Multiple bus masters	Multiple	Multiple
Arbitration	Centralized. Parallel arbitration	Self-select
Clocking	Synchronous 33-66 MHz	Asynchronous or Synchronous (5-10 MHz)
Theoretical peak bandwidth	133-512 MB/sec	5-40 MB/sec
Estimating typical achievable bandwidth of basic bus	80 MB/sec	2.5-4.0 MB/sec (synchronous) or 1.5 MB/sec (asychronous)
Maximum number of devices	1024 (for multiple bus segments, with 32 device / bus segment)	7-31 (bus width -1)
Maximum bus length	0.5 meter	25 meters
Standard name	PCI (Intel)	ANSI X3.131

I/O Controller Architecture



- ➢ **Request/response block interface**
- ➢ **Backdoor access to host memory**

* Slide is courtesy of Dave Patterson

Conclusion

❑ ***Summary***

- ➔ Interfacing I/O devices
  - Interfacing with I/O devices
  - Communication with I/O devices
  - Operating System's role
- ➔ Memory to processor interconnect
  - Definition of bus structure
  - Bus transactions
  - Types of buses
  - Bus Standards
- ➔ Bus Performance and Protocol
  - Synchronous versus Asynchronous buses
  - Bandwidth optimization factors
  - Single versus multiple master bus
  - Bus arbitration approaches

❑ ***Next Lecture***

- ➔ Multiprocessor Systems

Reading assignment includes sections 6.3--6.6 in textbook (3rd Ed.)