

---

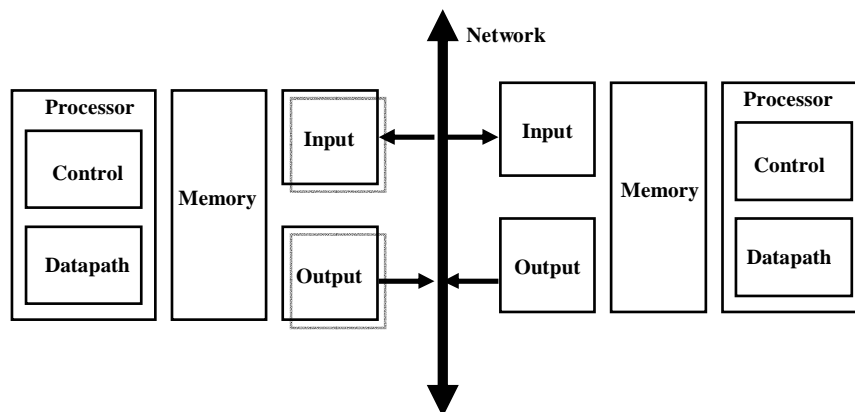
## Chapters 8

### Interfacing Processors and Peripherals

---

#### The Big Picture: Where are We Now?

- Today's Topic: I/O Systems



## Main components of Intel Chipset: Pentium III

Northbridge: a DMA controller, connecting the processor to memory, the AGP graphic bus, and the south bridge chip

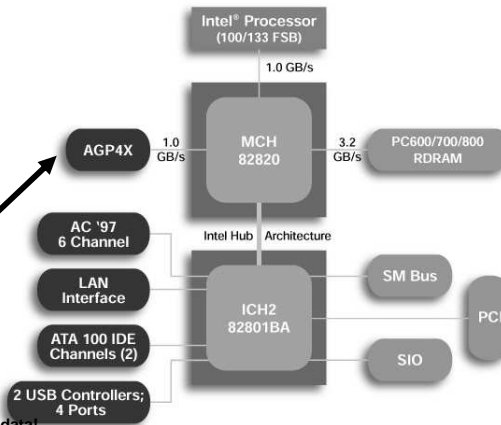
Southbridge: I/O

- PCI bus
- Disk controllers
- USB controllers
- Audio
- Serial I/O
- Interrupt controller
- Timers

DMA gives external device ability to access memory directly, much lower overhead than having processor request one word at a time.

Issue: Cache coherence:

- What if I/O devices write data that is currently in processor cache?
  - The processor may never see new data!
- Solutions:
  - Flush cache on every I/O operation (expensive)
  - Have hardware invalidate cache lines (remember "Coherence" cache misses?)



## Main components of Intel Chipset: Pentium 4

System Bus ("Front Side Bus"): 64 bits x 400, 533, 800 MHz

Gbit Ethernet: 125 MB/s

Hub bus: 8 bits x 266 MHz

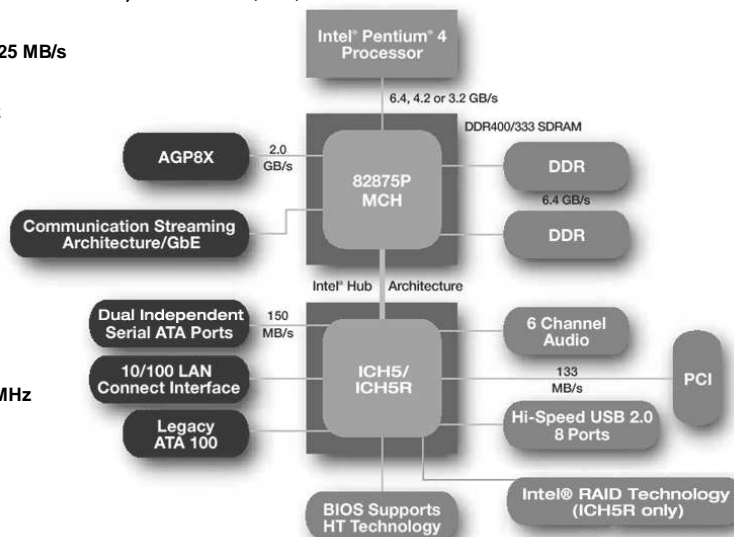
2 Serial ATA: 150 MB/s

10/100 Mbit Ethernet: 1.25 - 12.5 MB/s

Parallel ATA: 100 MB/s

8 USB: 60 MB/s

1 PCI: 32b x 33 MHz



## Table of Contents

---

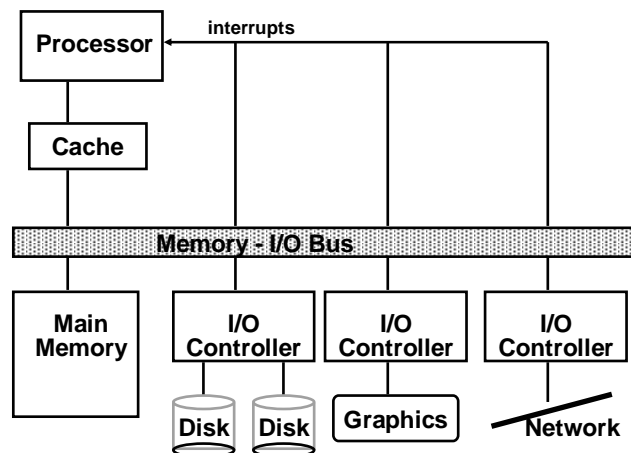
- I/O Performance Measures
- Queueing Theory
- Magnetic Disks – Example
- Other Measures
  - Reliability and Availability
  - RAID (Redundant Array of Inexpensive Disks) - Example
- Bus and Bus Arbitration

## I/O Performance Measures

---

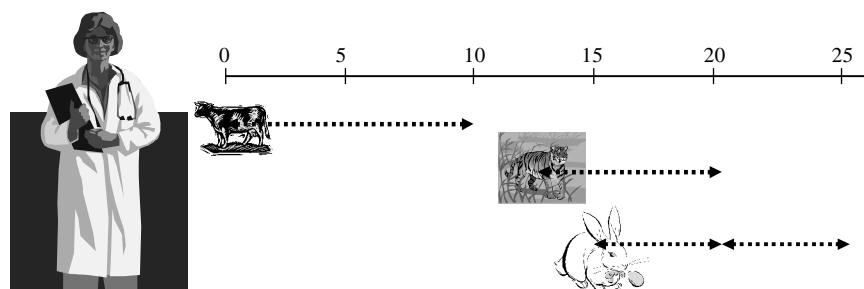
### I/O System Design Issues

- Performance
- Expandability
- Resilience in the face of failure



## 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



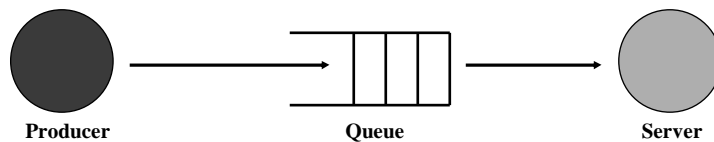
Throughput =  
Response time =

\* Service time, Queueing delay, Utilization

## Table of Contents

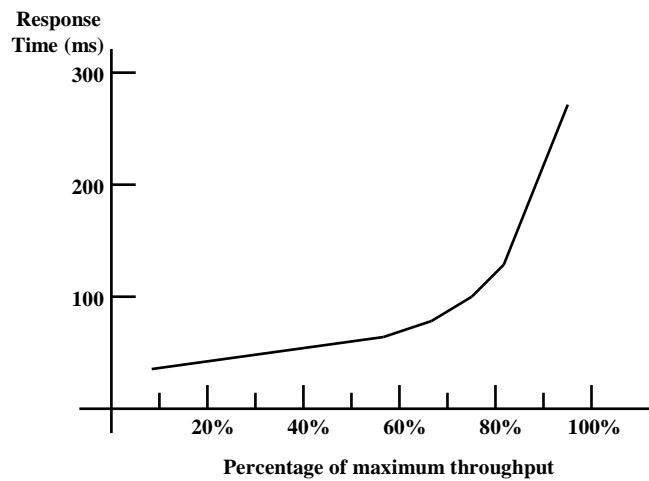
- I/O Performance Measures
- Queueing Theory
- Magnetic Disks – Example
- Other Measures
  - Reliability and Availability
  - RAID (Redundant Array of Inexpensive Disks) - Example
- Bus and Bus Arbitration

## Simple Producer-Server Model

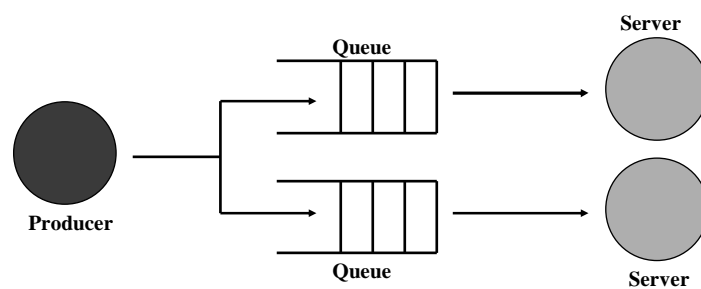


- **Throughput:**
  - The number of tasks completed by the server in unit time
  - In order to get the highest possible throughput:
    - The server should never be idle
    - The queue should never be empty
- **Response time:**
  - Begins when a task is placed in the queue
  - Ends when it is completed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

## Throughput versus Respond Time



## Throughput Enhancement



- In general throughput can be improved by:
  - Throwing more hardware at the problem
  - reduces load-related latency
- Response time is much harder to reduce:
  - Ultimately it is limited by the speed of light (but we're far from it)

## \* Queueing Model

◦ Queue: a useful abstraction of a shared resource

- A cpu in your PC
- The memory bus in your PC
- An Ethernet line in a local area network
- Wireless medium in a wireless LAN
- An output port in a network router
- A disk in a file server
- A semaphore in a multithreaded (multi-process) program

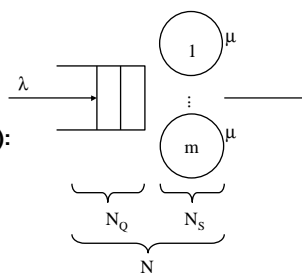
◦ A queueing system consists of

- Customers arrive at unpredictable times
- Each customer has a service requirement
- Specified number of servers and scheduling discipline



## Queue Metrics

- $X$  = throughput (i.e., departure rate)
- $U$  = server utilization (i.e., fraction of time the server is busy)
- $T$  = average time a customer spends in the queue (i.e., average waiting time + average service time)
  - Time in service ( $T_S$ ) vs in queue ( $T_Q$ ):  $T = T_Q + T_S$
- $N$  = average number of customers in the queue
  - Number in service ( $N_S$ ) vs number in queue ( $N_Q$ ):  $N = N_Q + N_S$
- Rules for all queues
  - Stability condition :  $\lambda < \mu$  (for infinite buffer and infinite population system)
  - Number vs Time : Little's Law (1961) if no creation/no lost of jobs
    - mean # in system = arrival rate x mean response time ( $N = \lambda T$  or  $N_Q = \lambda T_Q$ )



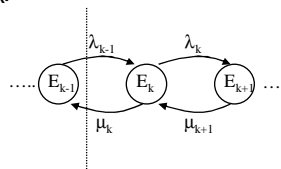
$T$ : response time  
 $T_S$ : service time per job =  $1/\mu$   
 $T_Q$ : waiting time

## Analysis of a Single Queue

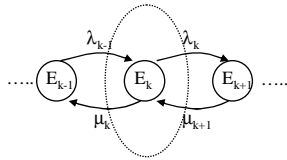
- **Interests in Queueing Systems**
  - Traffic Intensity ( $\lambda/\mu$ )
  - Server Utilization
  - Probability that N customers are in the system at time t
  
- **Relationships (Little's Law)**
  - $N = \lambda T$  (N: avg # in the system)
  - $N_q = \lambda T_q$  ( $N_q$ : avg # in queue)
  - $T = T_q + 1/\mu$  (T: avg waiting time in sys.)  
( $T_q$ : avg waiting time in queue)

## Birth-Death Queueing System

- **Homogeneous continuous-time Markov chain (discrete-state Markov)**
  - $P_k(t) = P[\text{system is in } E_k]$
  
- **In equilibrium, "rate in = rate out principle" : Local Balance Equation**
  - $p_k \triangleq \lim_{t \rightarrow \infty} P_k(t)$  : steady-state probability
  - **conservation of flow** :  $\lambda_{k-1} p_{k-1} = \mu_k p_k$   
 $\Rightarrow p_k = \lambda_{k-1} p_{k-1} / \mu_k$   
 $\Rightarrow p_k = p_0 (\lambda_0 \lambda_1 \dots \lambda_{k-1}) / (\mu_1 \mu_2 \dots \mu_k)$



## Local Balance Equation

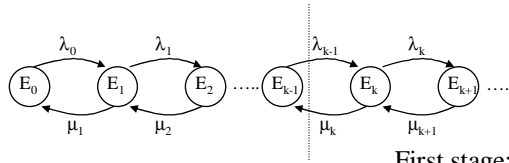


Conservation of flow:

- Flow into  $E_k = \lambda_{k-1}p_{k-1} + \mu_{k+1}p_{k+1}$

- Flow from  $E_k = \lambda_k p_k + \mu_k p_k$

$\Rightarrow \lambda_{k-1}p_{k-1} + \mu_{k+1}p_{k+1} = (\lambda_k + \mu_k)p_k$



First stage:  $\mu_1 p_1 = \lambda_0 p_0$

Second stage:  $\lambda_0 p_0 + \mu_2 p_2 = (\lambda_1 + \mu_1) p_1$

$\Rightarrow \mu_2 p_2 = \lambda_1 p_1$

Therefore,

$\mu_k p_k = \lambda_{k-1} p_{k-1}$  (steady-state)

## M/M/1: Classical Queueing System

◦ Birth-death process with  $\lambda_k = \lambda$ ,  $\mu_k = \mu$  for all  $k$

- $p_k = p_0 (\lambda_0 \lambda_1 \dots \lambda_{k-1}) / (\mu_1 \mu_2 \dots \mu_k) = p_0 (\lambda/\mu)^k$
- $p_0 = 1 / [1 + \sum_{k=1}^{\infty} (\lambda/\mu)^k] = 1 / [1 + (\lambda/\mu) / (1 - \lambda/\mu)] = 1 - \lambda/\mu = 1 - \rho$
- $\Rightarrow p_k = (1 - \rho) \rho^k$  where  $\rho = \lambda/\mu$

• average number of customers

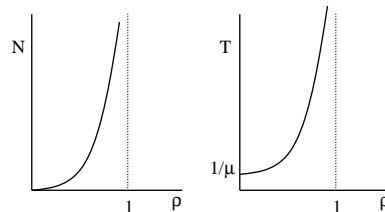
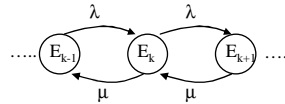
$$N = \sum_{k=1}^{\infty} k p_k = \rho / (1 - \rho)$$

• average time spent in the system

$$T = N / \lambda \text{ (Little's Law)}$$

$$= \rho / (1 - \rho) \lambda$$

$$= 1 / (1 - \rho) \mu$$



## M/M/1: Example

- Engineers use one terminal for serious computation. Arrival pattern is Poisson with a mean of 10 people each day. The distribution of time spent at a terminal is exponential with a mean of 30 minutes. Engineers complain about the terminal service but the manager finds the terminal is in use only 5 hours out of 8-hour working day.
  - $\lambda = 10 \text{ persons/day} \times 1 \text{ day/8hours} \times 1 \text{ hour/60minutes} = 1/48 \text{ persons/min}$
  - $\mu = 1/30 \text{ persons/min}$
  - $\rho = \lambda / \mu = 30/48 = 0.625$  : server utilization
  - $N = \rho / (1 - \rho) = 1.667 \text{ persons}$  : average number of customers in system
  - $N_q = 1.667 - 0.625 = 1.042$  : average number of customers in queue
  - $T = N / \lambda = 80 \text{ minutes}$  : average time spent in the system
  - $T_q = 80 - 30 = 50 \text{ minutes}$  : average time wasted in queue
  
- \*  $T_q = N_q / \lambda$   $\Leftarrow$  Little's Law can be applied in the subsystem

## A Little Queuing Theory: An Example

- processor sends 10 x 8KB disk I/Os per second, requests & service exponentially distrib., avg. disk service = 20 ms
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time spent in the queue?
  - What is the average response time for a disk request?

---

## A Little Queuing Theory: Another Example

- processor sends **20 x 8KB** disk I/Os per sec, requests & service exponentially distrib., avg. disk service = **12 ms**
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time a spent in the queue?
  - What is the average response time for a disk request?

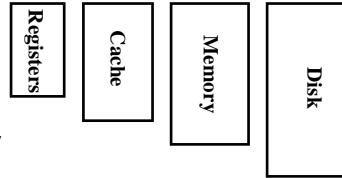
---

## Table of Contents

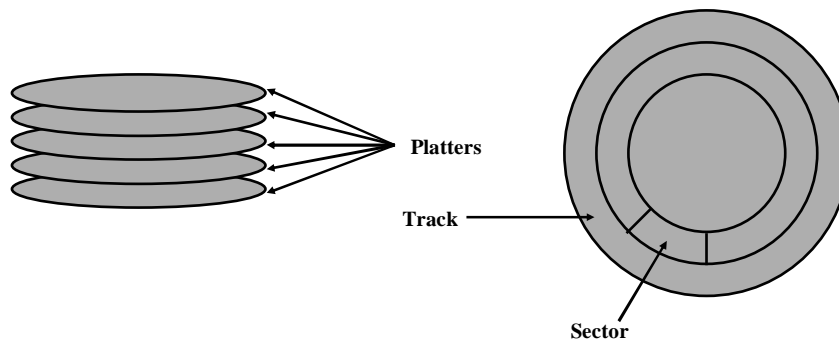
- I/O Performance Measures
- Queuing Theory
- Magnetic Disks – Example
- Other Measures
  - Reliability and Availability
  - RAID (Redundant Array of Inexpensive Disks) - Example
- Bus and Bus Arbitration

## Magnetic Disk

- Purpose:
  - Long term, nonvolatile storage
  - Large, inexpensive, and slow
  - Lowest level in the memory hierarchy
- Two major types:
  - Floppy disk
  - Hard disk
- Both types of disks:
  - Rely on a rotating platter coated with a magnetic surface
  - Use a moveable read/write head to access the disk
- Advantages of hard disks over floppy disks:
  - Platters are more rigid ( metal or glass) so they can be larger
  - Higher density because it can be controlled more precisely
  - Higher data rate because it spins faster
  - Can incorporate more than one platter



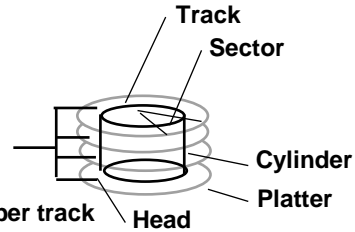
## Organization of a Hard Magnetic Disk



- Typical numbers (depending on the disk size):
  - 500 to 2,000 tracks per surface
  - 32 to 128 sectors per track
    - A sector is the smallest unit that can be read or written
- Traditionally all tracks have the same number of sectors:
  - Constant bit density: record more sectors on the outer tracks
  - Recently relaxed: constant bit size, speed varies with track location

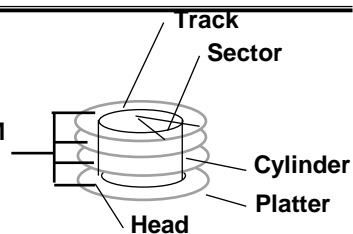
## Magnetic Disk Characteristic

- **Cylinder:** all the tracks under the head at a given point on all surface
- **Read/write data is a three-stage process:**
  - **Seek time:** position the arm over the proper track
  - **Rotational latency:** wait for the desired sector to rotate under the read/write head
  - **Transfer time:** transfer a block of bits (sector) under the read-write head
- **Average seek time as reported by the industry:**
  - Typically in the range of 8 ms to 12 ms
  - (Sum of the time for all possible seek) / (total # of possible seeks)
- **Due to locality of disk reference, actual average seek time may:**
  - Only be 25% to 33% of the advertised number

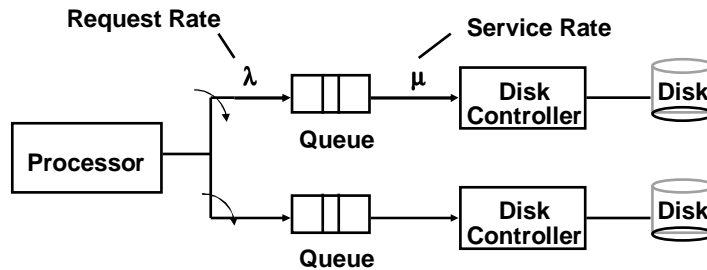


## Typical Numbers of a Magnetic Disk

- **Rotational Latency:**
  - Most disks rotate at 3,600 to 7200 RPM
  - Approximately 16 ms to 8 ms per revolution, respectively
  - An average latency to the desired information is halfway around the disk: 8 ms at 3600 RPM, 4 ms at 7200 RPM
- **Transfer Time is a function of :**
  - Transfer size (usually a sector): 1 KB / sector
  - Rotation speed: 3600 RPM to 7200 RPM
  - Recording density: bits per inch on a track
  - Diameter typical diameter ranges from 2.5 to 5.25 in
  - Typical values: 2 to 12 MB per second



## Disk I/O Performance



- Disk Access Time = Seek time + Rotational Latency + Transfer time + Controller Time + Queueing Delay
- Estimating Queue Length:
  - Utilization =  $U = \text{Request Rate} / \text{Service Rate}$
  - Mean Queue Length =  $U / (1 - U)$
  - As Request Rate  $\rightarrow$  Service Rate
    - Mean Queue Length  $\rightarrow$  Infinity

## Example

- 512 byte sector, rotate at 5400 RPM, advertised seeks is 12 ms, transfer rate is 4 MB/sec, controller overhead is 1 ms, queue idle so no service time
- Disk Access Time = Seek time + Rotational Latency + Transfer time + Controller Time + Queueing Delay
- Disk Access Time = 12 ms + 0.5 / 5400 RPM + 0.5 KB / 4 MB/s + 1 ms + 0
- Disk Access Time = 12 ms + 0.5 / 90 RPS + 0.125 / 1024 s + 1 ms + 0
- Disk Access Time = 12 ms + 5.5 ms + 0.1 ms + 1 ms + 0 ms
- Disk Access Time = 18.6 ms
- If real seeks are 1/3 advertised seeks, then its 10.6 ms, with rotation delay at 50% of the time!

## Magnetic Disk Examples

<b>Characteristics</b>	<b>IBM 3090</b>	<b>IBM UltraStar</b>	<b>Integral 1820</b>
Disk diameter (inches)	10.88	3.50	1.80
Formatted data capacity (MB)	22,700	4,300	21
MTTF (hours)	50,000	1,000,000	100,000
Number of arms/box	12	1	1
Rotation speed (RPM)	3,600	7,200	3,800
Transfer rate (MB/sec)	4.2	9-12	1.9
Power/box (watts)	2,900	13	2
MB/watt	8	102	10.5
Volume (cubic feet)	97	0.13	0.02
MB/cubic feet	234	33000	1050

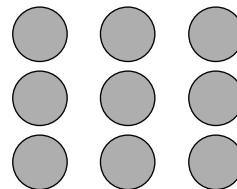
## Table of Contents

- **I/O Performance Measures**
- **Queueing Theory**
- **Magnetic Disks – Example**
- **Other Measures**
  - **Reliability and Availability**
  - **RAID (Redundant Array of Inexpensive Disks) - Example**
- **Bus and Bus Arbitration**

## Reliability and Availability

- Two terms that are often confused:
  - **Reliability:** Is anything broken?
  - **Availability:** Is the system still available to the user?
- **Availability** can be improved by adding hardware:
  - **Example:** adding ECC on memory
- **Reliability** can only be improved by:
  - **Bettering environmental conditions**
  - **Building more reliable components**
  - **Building with fewer components**
    - Improve availability may come at the cost of lower reliability

## Disk Arrays



- **A new organization of disk storage:**
  - **Arrays of small and inexpensive disks**
  - **Increase potential throughput by having many disk drives:**
    - Data is spread over multiple disk
    - Multiple accesses are made to several disks
- **Reliability is lower than a single disk:**
  - **But availability can be improved by adding redundant disks (RAID):**  
Lost information can be reconstructed from redundant information
  - **MTTR:** mean time to repair is in the order of hours
  - **MTTF:** mean time to failure of disks is tens of years

**Replace Small # of Large Disks with Large # of Small Disks! (1988 Disks)**

	IBM 3390 (K)	IBM 3.5" 0061	x70
<i>Data Capacity</i>	20 GBytes	320 MBytes	23 GBytes
<i>Volume</i>	97 cu. ft.	0.1 cu. ft.	11 cu. ft.
<i>Power</i>	3 KW	11 W	1 KW
<i>Data Rate</i>	15 MB/s	1.5 MB/s	120 MB/s
<i>I/O Rate</i>	600 I/Os/s	55 I/Os/s	3900 I/Os/s
<i>MTTF</i>	250 KHrs	50 KHrs	??? Hrs
<i>Cost</i>	\$250K	\$2K	\$150K

*Disk Arrays have potential for*

- large data and I/O rates
- high MB per cu. ft., high MB per KW
- reliability?

**Array Reliability**

- Reliability of N disks = Reliability of 1 Disk ÷ N

50,000 Hours ÷ 70 disks = 700 hours

Disk system MTTF: Drops from 6 years to 1 month!

- Arrays (without redundancy) too unreliable to be useful!

**Hot spares support reconstruction in parallel with access: very high media availability can be achieved**

## Redundant Arrays of Disks

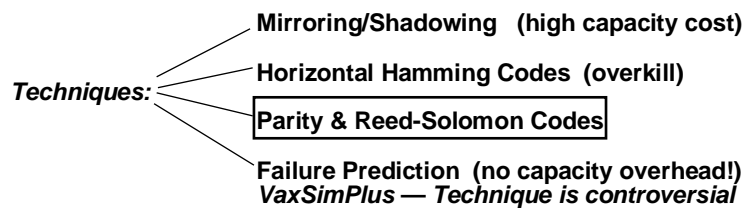
- Files are "striped" across multiple spindles
- Redundancy yields high data availability

Disks will fail

Contents reconstructed from data redundantly stored in the array

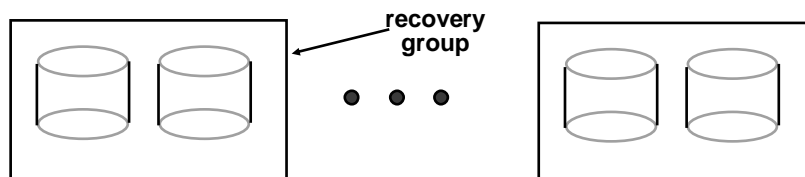
→ Capacity penalty to store it

→ Bandwidth penalty to update



## Redundant Arrays of Disks

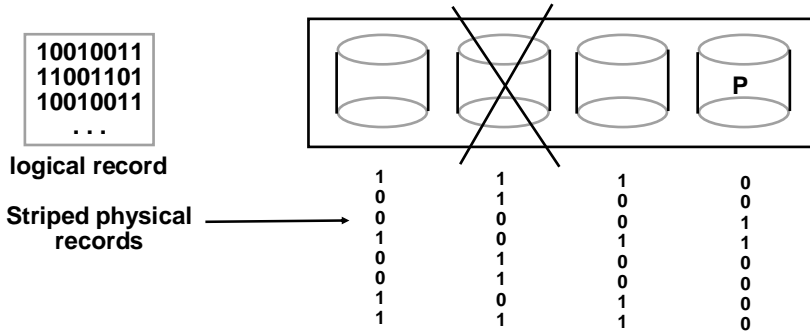
### RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its "shadow"  
Very high availability can be achieved
- Bandwidth sacrifice on write:  
Logical write = two physical writes
- Reads may be optimized
- Most expensive solution: 100% capacity overhead

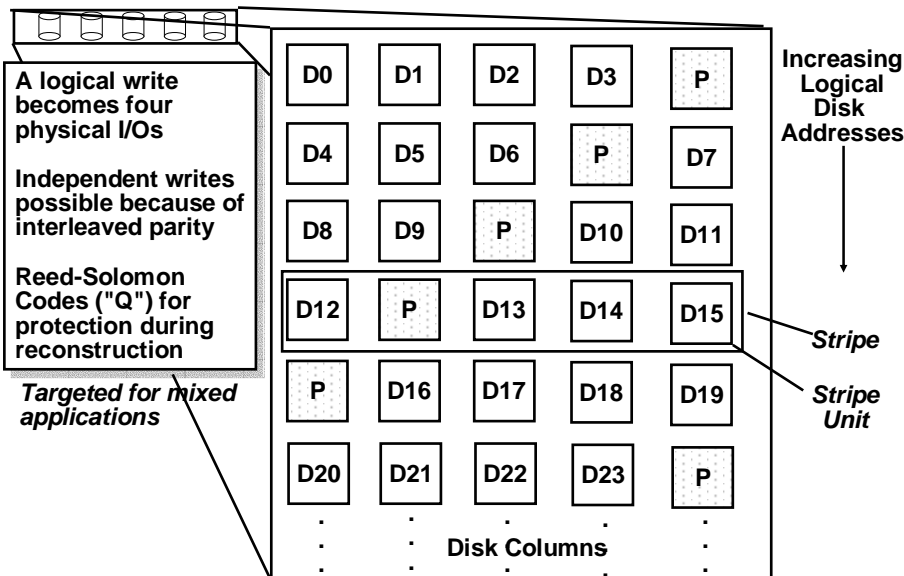
*Targeted for high I/O rate , high availability environments*

## Redundant Arrays of Disks RAID 3: Parity Disk



- Parity computed across recovery group to protect against hard disk failures
    - 33% capacity cost for parity in this configuration
    - wider arrays reduce capacity costs, decrease expected availability, increase reconstruction time
  - Arms logically synchronized, spindles rotationally synchronized
    - logically a single high capacity, high transfer rate disk
- Targeted for high bandwidth applications: Scientific, Image Processing*

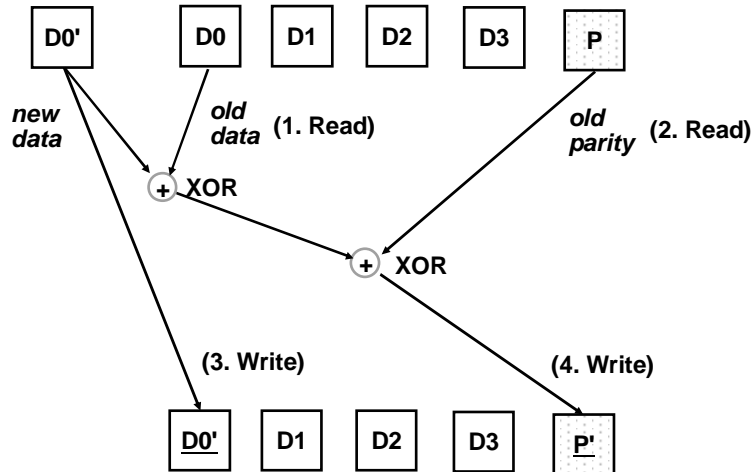
## Redundant Arrays of Disks RAID 5+: High I/O Rate Parity



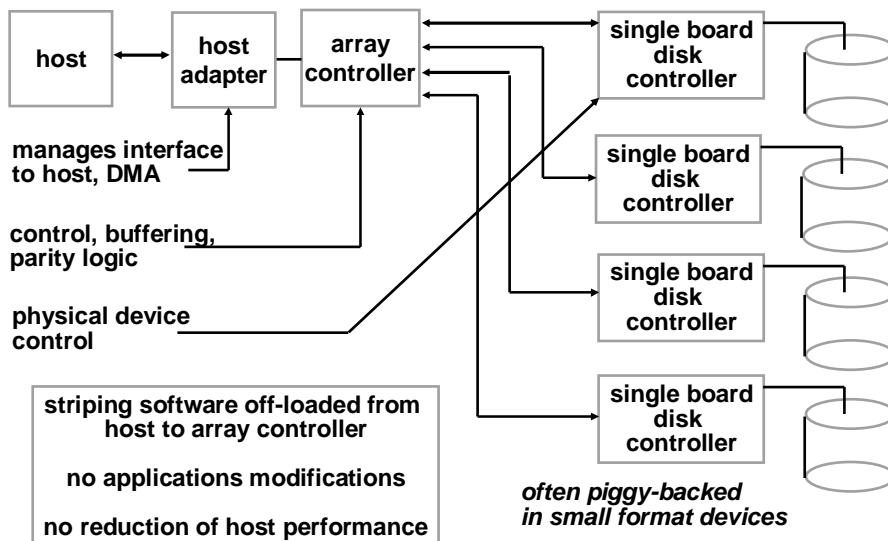
## Problems of Disk Arrays: Small Writes

### RAID-5: Small Write Algorithm

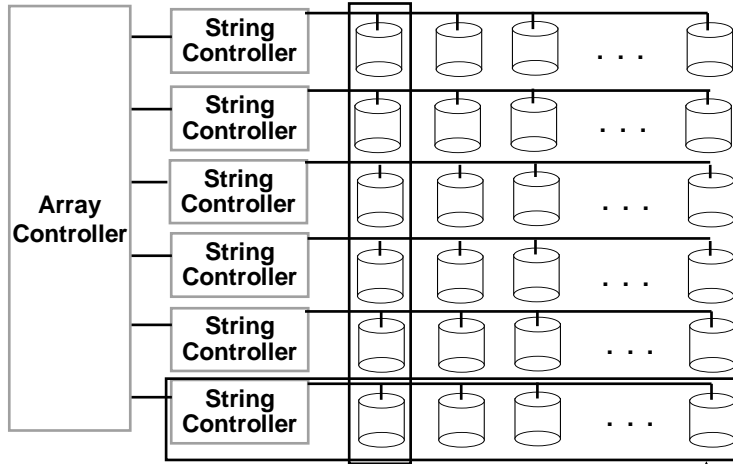
1 Logical Write = 2 Physical Reads + 2 Physical Writes



## Subsystem Organization



## System Availability: Orthogonal RAIDs

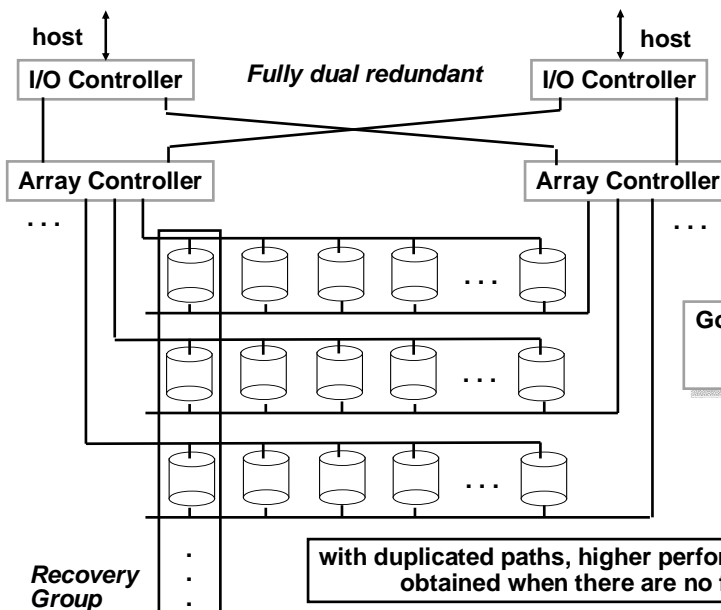


*Data Recovery Group: unit of data redundancy*

*Redundant Support Components: fans, power supplies, controller, cables*

*End to End Data Integrity: internal parity protected data paths*

## System-Level Availability



**Goal: No Single Points of Failure**

*Recovery Group*

**with duplicated paths, higher performance can be obtained when there are no failures**

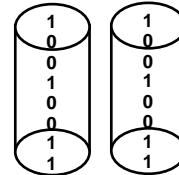
## Summary: Redundant Arrays of Disks (RAID) Techniques

- **Disk Mirroring, Shadowing (RAID 1)**

Each disk is fully duplicated onto its "shadow"

Logical write = two physical writes

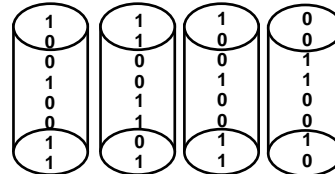
100% capacity overhead



- **Parity Data Bandwidth Array (RAID 3)**

Parity computed horizontally

Logically a single high data bw disk



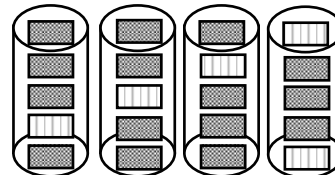
- **High I/O Rate Parity Array (RAID 5)**

Interleaved parity blocks

Independent reads and writes

Logical write = 2 reads + 2 writes

Parity + Reed-Solomon codes



## Table of Contents

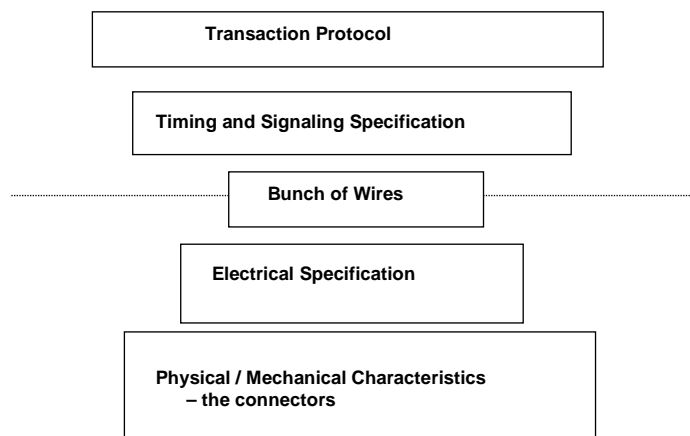
- I/O Performance Measures
- Queueing Theory
- Magnetic Disks – Example
- Other Measures
  - Reliability and Availability
  - RAID (Redundant Array of Inexpensive Disks) - Example
- Bus and Bus Arbitration

## Interconnect Trends

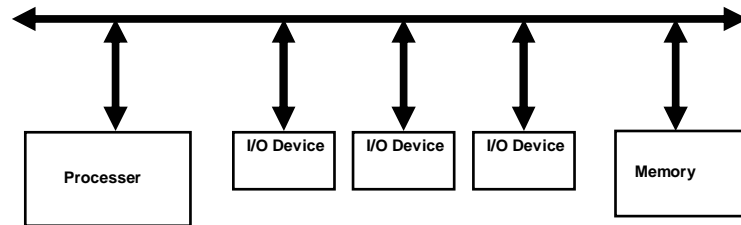
- Interconnect = glue that interfaces computer system components
- High speed hardware interfaces + logical protocols
- Networks, channels, backplanes

	Network	Channel	Backplane
<b>Distance</b>	>1000 m	10 - 100 m	1 m
<b>Bandwidth</b>	10 - 100 Mb/s	40 - 1000 Mb/s	320 - 1000+ Mb/s
<b>Latency</b>	high (>ms)	medium	low (<100ns)
<b>Reliability</b>	low Extensive CRC	medium Byte Parity	high Byte Parity
	message-based narrow pathways distributed arb	←————→	memory-mapped wide pathways centralized arb

## What defines a bus?

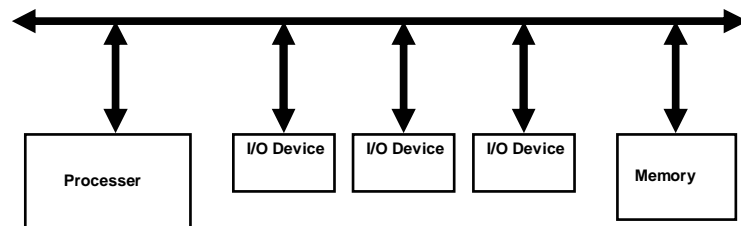


## Advantages of Buses



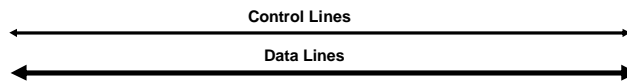
- **Versatility:**
  - New devices can be added easily
  - Peripherals can be moved between computer systems that use the same bus standard
- **Low Cost:**
  - A single set of wires is shared in multiple ways

## Disadvantage of Buses



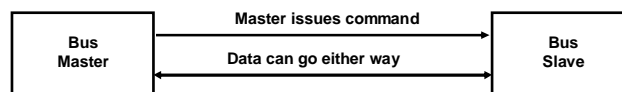
- It creates a communication bottleneck
  - The bandwidth of that bus can limit the maximum I/O throughput
- The maximum bus speed is largely limited by:
  - The length of the bus
  - The number of devices on the bus
  - The need to support a range of devices with:
    - Widely varying latencies
    - Widely varying data transfer rates

## The General Organization of a Bus



- Control lines:
  - Signal requests and acknowledgments
  - Indicate what type of information is on the data lines
- Data lines carry information between the source and the destination:
  - Data and Addresses
  - Complex commands

## Master versus Slave



- A bus transaction includes two parts:
  - Issuing the command (and address) – request
  - Transferring the data – action
- Master is the one who starts the bus transaction by:
  - issuing the command (and address)
- Slave is the one who responds to the address by:
  - Sending data to the master if the master ask for data
  - Receiving data from the master if the master wants to send data

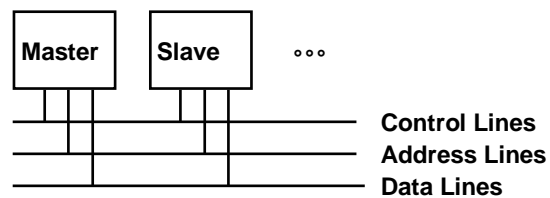
---

## Bus-Based Interconnect

- Two generic types of busses:
  - I/O busses: lengthy, many types of devices connected, wide range in the data bandwidth), and follow a bus standard (sometimes called a *channel*)
  - CPU–memory busses: high speed, matched to the memory system to maximize memory–CPU bandwidth, single device (sometimes called a *backplane*)
  - To lower costs, low cost (older) systems combine together
- Bus transaction
  - Sending address & receiving or sending data

---

## Bus Protocols



**Multibus: 20 address, 16 data, 5 control, 50ns Pause**

**Bus Master:** has ability to control the bus, initiates transaction

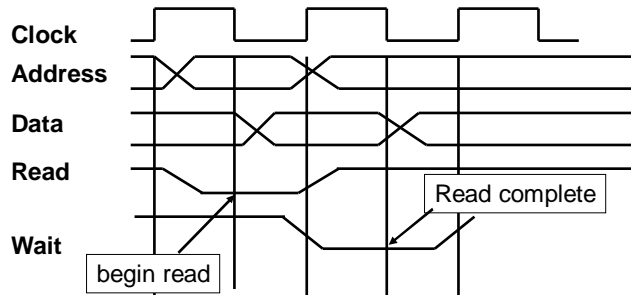
**Bus Slave:** module activated by the transaction

**Bus Communication Protocol:** specification of sequence of events and timing requirements in transferring information.

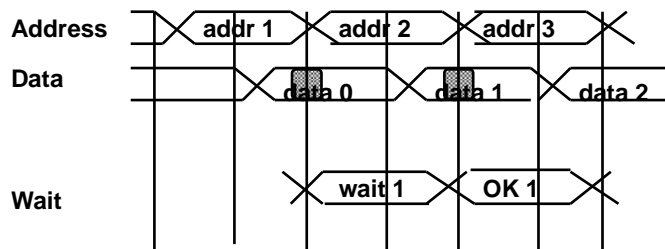
**Asynchronous Bus Transfers:** control lines (req., ack.) serve to orchestrate sequencing

**Synchronous Bus Transfers:** sequence relative to common clock

## Synchronous Bus Protocols

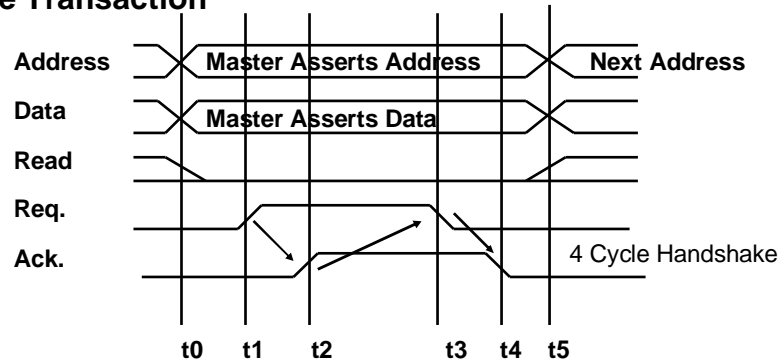


### Pipelined/Split transaction Bus Protocol



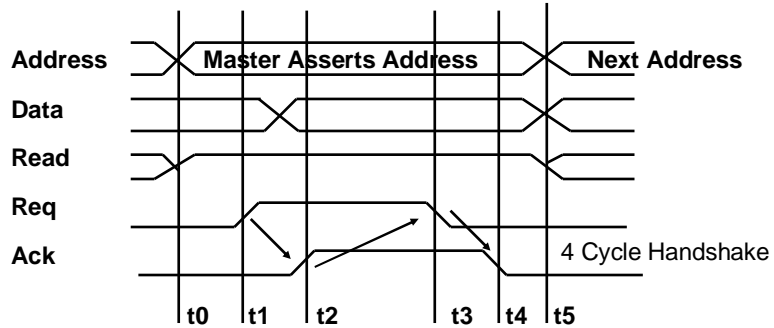
## Asynchronous Handshake

### Write Transaction



- t0 : Master has obtained control and asserts address, direction, data  
Waits a specified amount of time for slaves to decode target\
- t1: Master asserts request line
- t2: Slave asserts ack, indicating data received
- t3: Master releases req
- t4: Slave releases ack

## Read Transaction

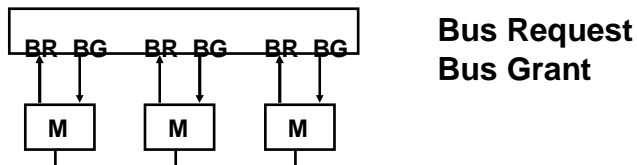


- t0 : Master has obtained control and asserts address, direction, data  
Waits a specified amount of time for slaves to decode target\
- t1: Master asserts request line
- t2: Slave asserts ack, indicating ready to transmit data
- t3: Master releases req, data received
- t4: Slave releases ack

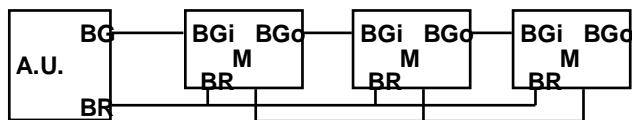
**Time Multiplexed Bus: address and data share lines**

## Bus Arbitration

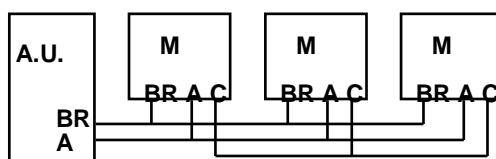
Parallel (Centralized) Arbitration



Serial Arbitration (daisy chaining)



Polling



## Bus Options

<i>Option</i>	<i>High performance</i>	<i>Low cost</i>	
Bus width data lines	Separate address & data lines	Multiplex address	&
Data width 32 bits)	Wider is faster (e.g., 8 bits)	Narrower is cheaper	(e.g.,
Transfer size bus overhead	Multiple words has is simpler	Single-word transfer	less
Bus masters	Multiple (requires arbitration)	Single master (no arbitration)	
Split	Yes—separate Request and Reply packets gets higher bandwidth (needs multiple masters)	No—continuous connection is cheaper and has lower latency	transaction?
Clocking	Synchronous	Asynchronous	

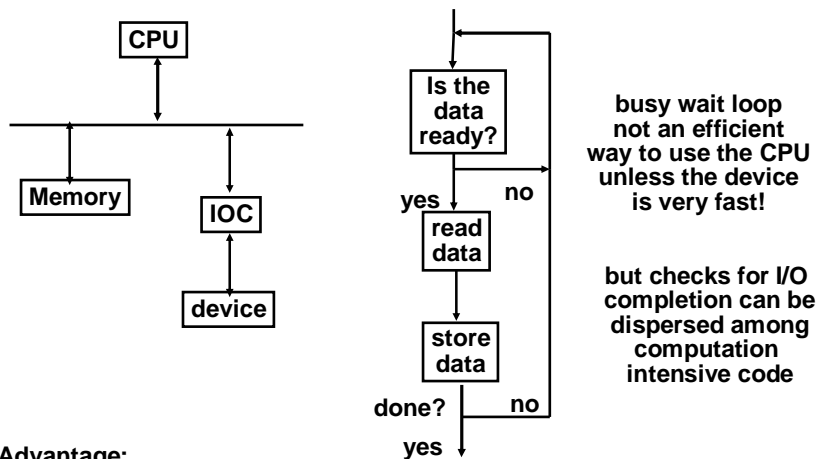
## Giving Commands to I/O Devices

- Two methods are used to address the device:
  - Special I/O instructions
  - Memory-mapped I/O
- Special I/O instructions 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
- Memory-mapped I/O:
  - 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

## I/O Device Notifying the OS

- 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:
    - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing.

## Polling: Programmed I/O

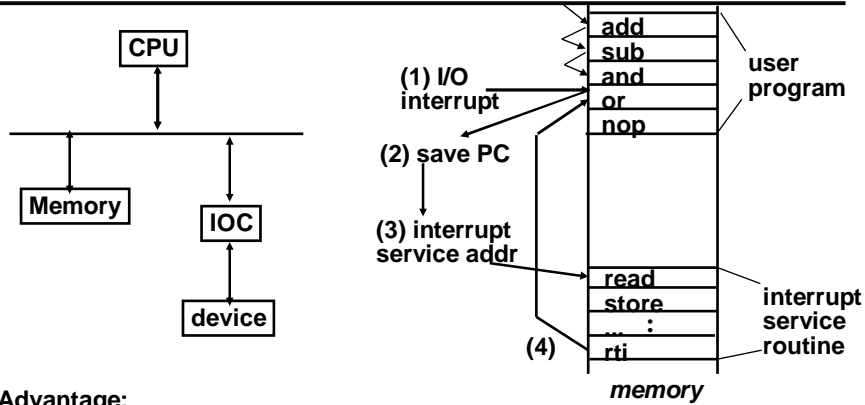


busy wait loop  
not an efficient  
way to use the CPU  
unless the device  
is very fast!

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

## 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)

## I/O Interrupt

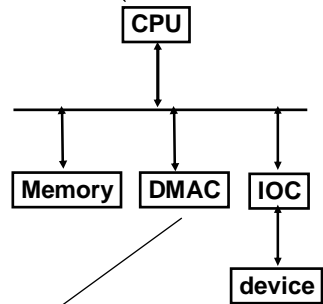
- An I/O interrupt is just like the exceptions except:
  - An I/O interrupt is asynchronous
  - Further information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction
  - I/O interrupt does not prevent 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

## Delegating I/O Responsibility from the CPU: DMA

◦ **Direct Memory Access (DMA):**

- External to the CPU
- Act as a master on the bus
- Transfer blocks of data to or from memory without CPU intervention

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.