



浙江大学  
ZHEJIANG UNIVERSITY

计算机组成与设计  
**Computer Organization & Design**  
**The Hardware/Software Interface**  
**APPENDIX**  
**Storage, Networks and Other Peripherals**

马德

College of Computer Science and Technology  
Zhejiang University  
[made@zju.edu.cn](mailto:made@zju.edu.cn)

---

# Contents



- ❑ **A.1 Introduction**
- ❑ **A.2 Disk Storage and Dependability**
- ❑ **A.3 Networks (Skim)**
- ❑ **A.4 Buses and Other Connections between Processors Memory, and I/O Devices**
- ❑ **A.5 Interfacing I/O Devices to the Memory, Processor, and Operating System**
- ❑ **A.6 I/O Performance Measures:  
Examples from Disk and File Systems**
- ❑ **A.7 Designing an I/O system**
- ❑ **A.8 Real Stuff: A Typical Desktop I/O System**

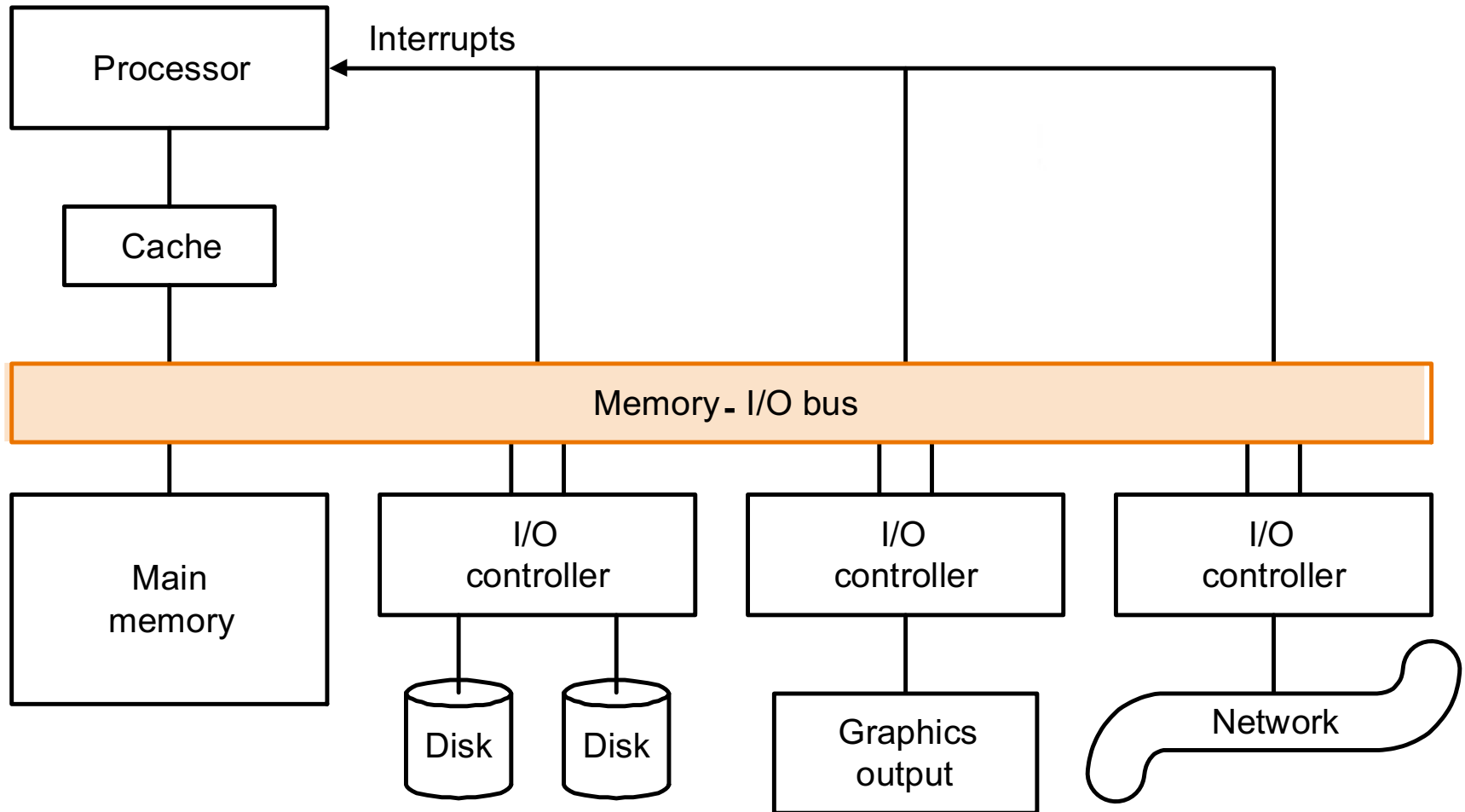


# A.1 Introduction

---

- ❑ **I/O Designers must consider many factors**
  - such as expandability and resilience(resume),as well as performance.
- ❑ **Assessing I/O system performance is very difficult.**
  - In different situations, needs use different measurements.
- ❑ **Performance of I/O system depends on:**
  - connection between devices and the system
  - the memory hierarchy
  - the operating system

# Typical collection of I/O device





# Three characters of IO

---

## □ Behavior

- Input (read once), output (write only, cannot read), or storage (can be reread and usually rewritten)

## □ Partner

- Either a human or a machine is at the other end of the I/O device, either feeding data on input or reading data on output.

## □ Data rate

- The peak rate at which data can be transferred between the I/O device and the main memory or processor.

# I/O performance depends on the application



## □ Throughput:

In these cases, I/O bandwidth will be most important. Even I/O bandwidth can be measured in two different ways according to different situations:

1. How much data can we move through the system in a certain time?

For examples, in many supercomputer applications, most I/O requires are for long streams of data, and transfer bandwidth is the important characteristic.

2. How many I/O operations can we do per unit of time?

For example, National Income Tax Service mainly processes large number of small files.

## □ Response time (e.g., workstation and PC)

## □ both throughput and response time (e.g., ATM)



# The diversity of I/O devices

Device	Behavior	Partner	Data rate (KB/sec)
Keyboard	input	human	0.01
Mouse	input	human	0.02
Voice input	input	human	0.02
Scanner	input	human	400.00
Voice output	output	human	0.60
Line printer	output	human	1.00
Laser printer	output	human	200.00
Graphics display	output	human	60,000.00
Modem	input or output	machine	2.00-8.00
Network/LAN	input or output	machine	500.00-6000.00
Floppy disk	storage	machine	100.00
Optical disk	storage	machine	1000.00
Magnetic tape	storage	machine	2000.00
Magnetic disk	storage	machine	2000.00-10,000.00



# Important but neglected

---

- *“The difficulties in assessing and designing I/O systems have often relegated I/O to second class status”*
- *“courses in every aspect of computing, from programming to computer architecture often ignore I/O or give it scanty coverage”*
- *“textbooks leave the subject to near the end, making it easier for students and instructors to skip it!”*





# Amdahl's (阿姆达尔)law

□ Sequential part can limit speedup

□ Example: 100 processors, 90× speedup?

■  $T_{\text{new}} = T_{\text{parallelizable}}/100 + T_{\text{sequential}}$

■ 
$$\text{Speedup} = \frac{1}{(1 - F_{\text{parallelizable}}) + F_{\text{parallelizable}}/100} = 90$$

■ Solving:  $F_{\text{parallelizable}} = 0.999$

□ Need sequential part to be 0.1% of original time



# Amdahl's (阿姆达尔)law

## □ Remind us that ignoring I/O is dangerous

### ■ Assume:

- a bench mark executes in 100 seconds of elapsed time , where 90 seconds is CPU time and the rest is I/O time.
- CPU time improves by 50% per year, but I/O time doesn't improve.
- After five years, the improvement in CPU performance is 7.5 times.

### ■ The elapsed time is reduced to $90/7.5 + 10 = 12 + 10 = 22$ seconds.

### ■ So, the improvement in elapsed time is only 4.5 times.



# A.2 Disk Storage and Dependability

---

## □ Two major types of magnetic disks

- floppy disks
- hard disks
  - larger
  - higher density
  - higher data rate
  - more than one platter

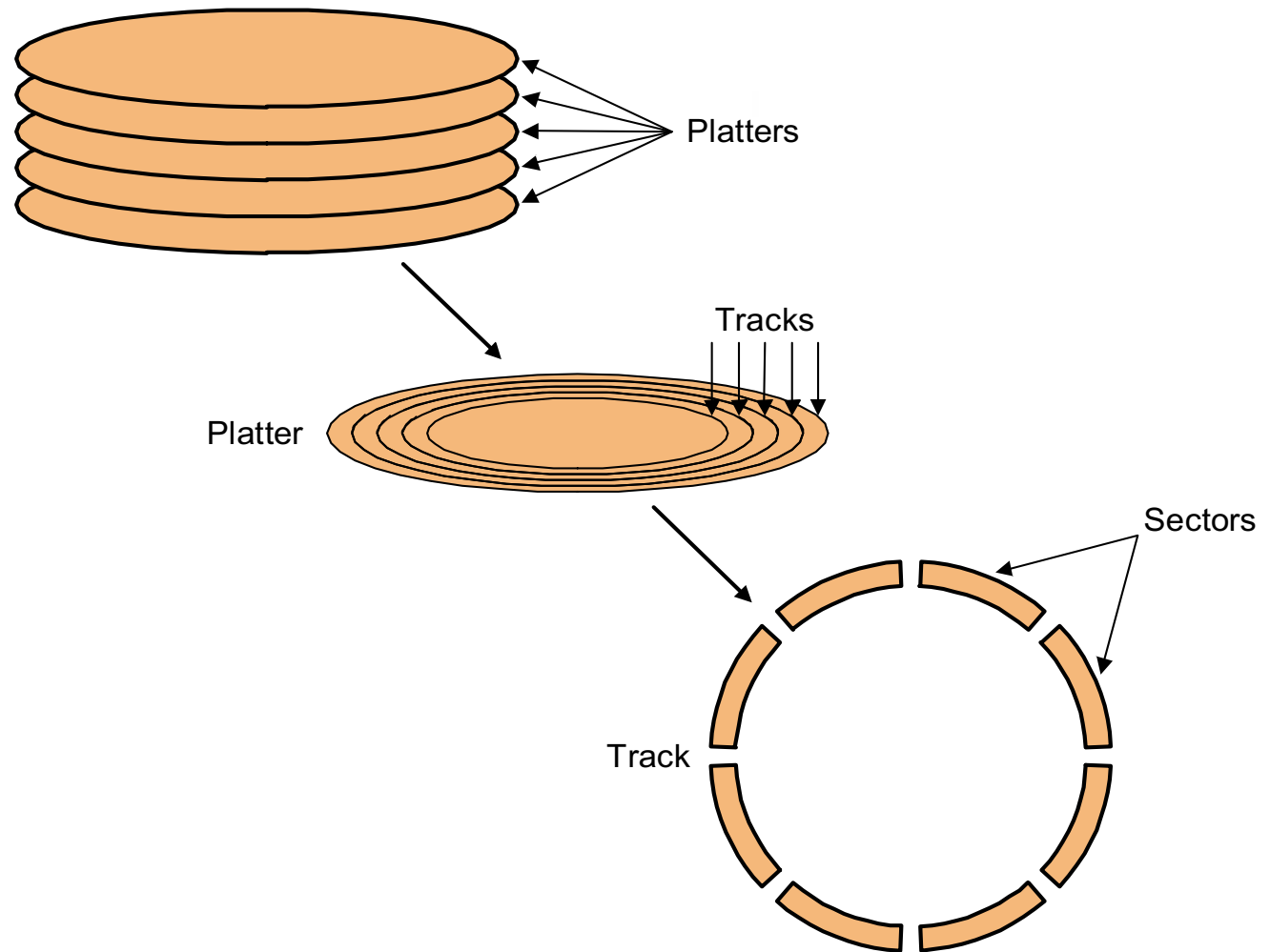


# The organization of hard disk

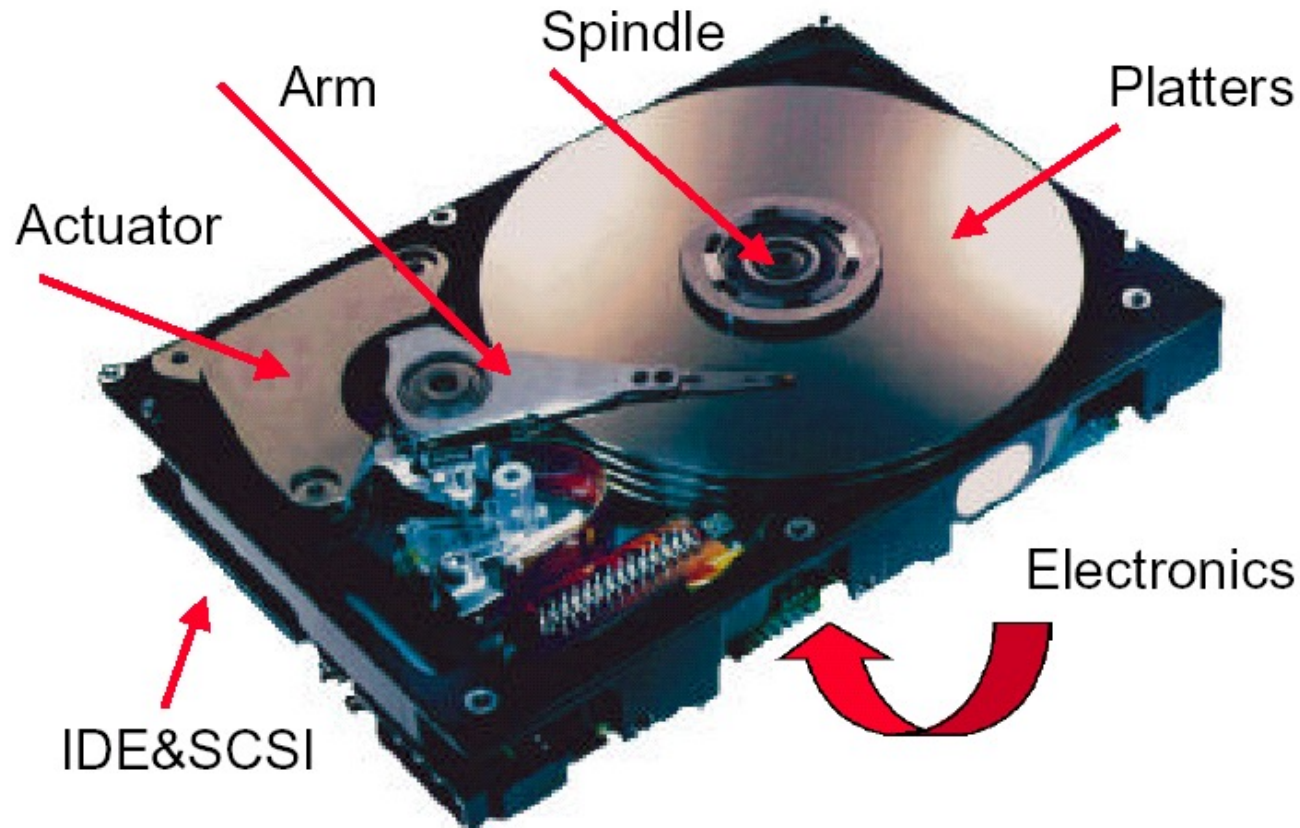
---

- ❑ **platters:** disk consists of a collection of platters, each of which has two recordable disk surfaces
- ❑ **tracks:** each disk surface is divided into concentric circles
- ❑ **sectors:** each track is in turn divided into sectors, which is the smallest unit that can be read or written

## □ Disks are organized into platters, tracks, and sectors



# What's Inside A Disk Drive?





# To access data of disk

- **Seek:** position read/write head over the proper track
  - minimum seek time
  - maximum seek time
  - average seek time (3 to 14 ms)
- **Rotational latency:** wait for desired sector

*For 5400RPM*

$$\begin{aligned}\text{Average rotational latency} &= \frac{0.5 \text{ rotation}}{5400\text{RPM}} = \frac{0.5 \text{ rotation}}{5400\text{RPM} / \left(60 \frac{\text{seconds}}{\text{minute}}\right)} \\ &= 0.0056 \text{ seconds} = 5.6 \text{ ms}\end{aligned}$$

*For 15000RPM*

$$\begin{aligned}\text{Average rotational latency} &= \frac{0.5 \text{ rotation}}{15000\text{RPM}} = \frac{0.5 \text{ rotation}}{15000\text{RPM} / \left(60 \frac{\text{seconds}}{\text{minute}}\right)} \\ &= 0.0020 \text{ seconds} = 2.0 \text{ ms}\end{aligned}$$



# To access data of disk

- ❑ **Transfer:** time to transfer a sector (1 KB/sector) function of rotation speed, Transfer rate today's drives - 30 to 80 MBytes/second
- ❑ **Disk controller:** which control the transfer between the disk and the memory

## Disk Read Time

512B/sector 50MB/S

Access Time = Seek time + Rotational Latency + Transfer time + Controller Time

$$= 6\text{ms} + \frac{0.5}{10,000\text{PRM}} + \frac{0.5\text{KB}}{50\text{MB/sec}} + 0.2\text{ms}$$

$$= 6\text{ms} + 3.0 + 0.01 + 0.2 = 9.2\text{ms}$$

Assuming the measured seek time is 25% of the calculated average

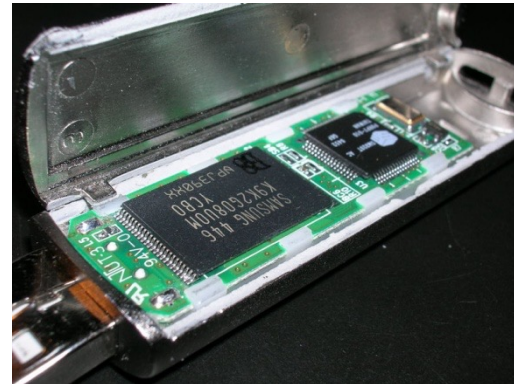
$$\text{Access Time} = 25\% \times 6\text{ms} + 3.0\text{ms} + 0.01\text{ms} + 0.2\text{ms} = 4.7\text{ms}$$



# Flash Storage

## □ Nonvolatile semiconductor storage

- $100\times - 1000\times$  faster than disk
- Smaller, lower power, more robust
- But more \$/GB (between disk and DRAM)





# Flash Types

---

- ❑ **NOR flash: bit cell like a NOR gate**
  - Random read/write access
  - Used for instruction memory in embedded systems
- ❑ **NAND flash: bit cell like a NAND gate**
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, ...
- ❑ **Flash bits wears out after 1000's of accesses**
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks



# Disk Performance Issues

---

## ❑ Manufacturers quote average seek time

- Based on all possible seeks
- Locality and OS scheduling lead to smaller actual average seek times

## ❑ Smart disk controller allocate physical sectors on disk

- Present logical sector interface to host
- SCSI, ATA, SATA

## ❑ Disk drives include caches

- Prefetch sectors in anticipation of access
- Avoid seek and rotational delay



# Dependability, Reliability, Availability

- Computer system **dependability** is the quality of delivered service such that reliance can justifiably be placed on this service.
- Each module also has an ideal specified behavior, where a service specification is an agreed description of the expected behavior. A system failure occurs when the actual behavior deviates from the specified behavior.

**Service accomplishment**, where the service is delivered as specified

**Service interruption**, where the delivered service is different from the specified service



# Measure

- *MTTF* *mean time to failure* 平均无故障时间
- *MTTR* *mean time to repair* 平均修复时间
- *MTBF* (Mean Time Between Failures)  
= *MTTF* + *MTTR* 平均故障间隔时间
- *Availability*

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$



# Three way to improve MTTF

---

## □ ***Fault avoidance:***

*preventing fault occurrence by construction*

## □ ***Fault tolerance:***

*using redundancy to allow the service to comply with the service specification despite faults occurring, which applies primarily to hardware faults*

## □ ***Fault forecasting:***

*predicting the presence and creation of faults, which applies to hardware and software faults*



# Reasons for failures

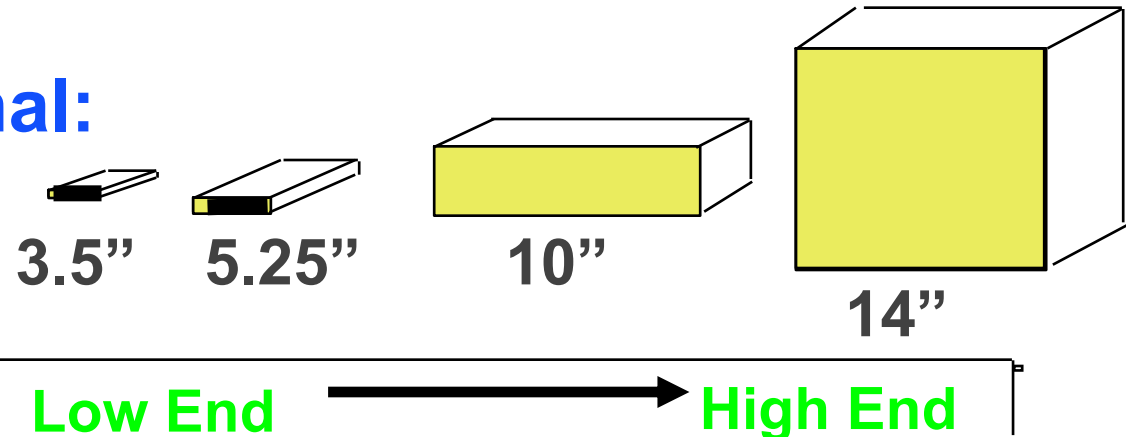
Operator	Software	Hardware	System	Year data collected
42%	25%	18%	Data center (Tandem)	1985
15%	55%	14%	Data center (Tandem)	1989
18%	44%	39%	Data center (DEC VAX)	1985
50%	20%	30%	Data center (DEC VAX)	1993
50%	14%	19%	U.S. public telephone network	1996
54%	7%	30%	U.S. public telephone network	2000
60%	25%	15%	Internet services	2002

# Use Arrays of Small Disks?

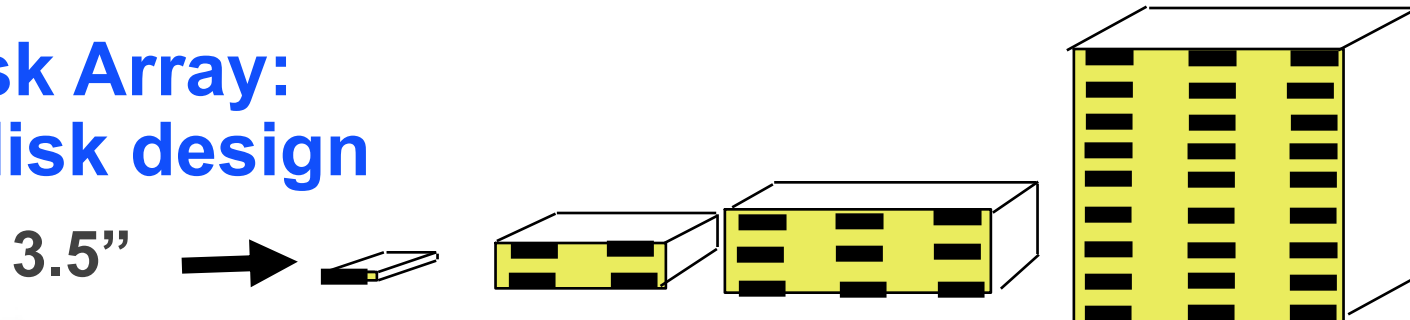
## □ Katz and Patterson asked in 1987:

- Can smaller disks be used to close gap in performance between disks and CPUs?

**Conventional:  
4 disk  
designs**



**Disk Array:  
1 disk design**







# Array Reliability

- Reliability of N disks = Reliability of 1 Disk  $\div$  N

$$50,000 \text{ Hours} \div 70 \text{ disks} = 700 \text{ hours}$$

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 (Inexpensive) Disks

- ❑ Files are "striped" across multiple disks
- ❑ Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- ❑ Disks will still fail
- ❑ Contents reconstructed from data redundantly stored in the array

⇒ Capacity penalty to store redundant info

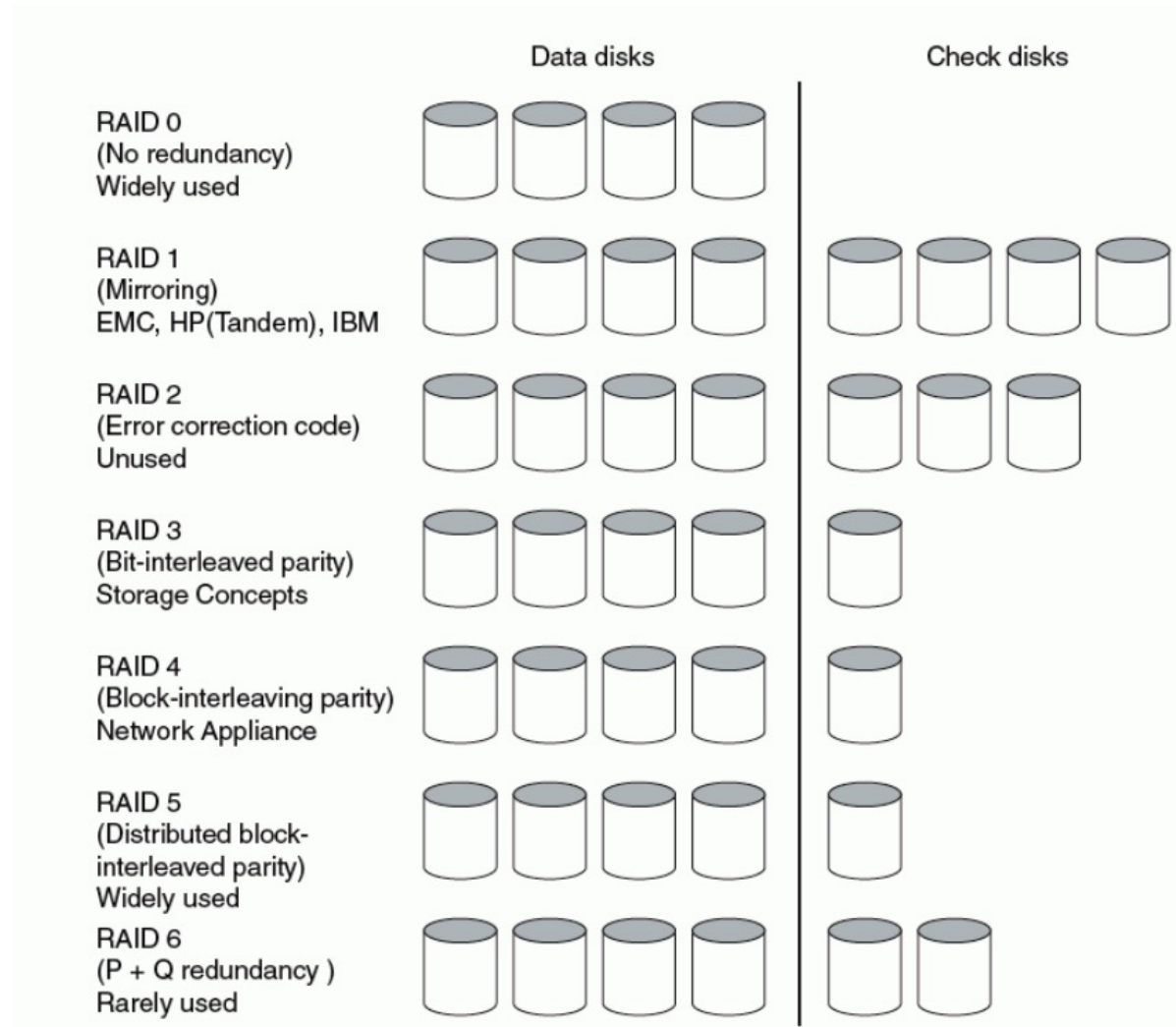
⇒ Bandwidth penalty to update redundant info



# RAID: Redundant Arrays of Inexpensive Disks

## □ A disk arrays replace larger disk

RAID level		Minimum number of Disk faults survived	Example Data disks	Corresponding Check disks	Corporations producing RAID products at this level
0	Non-redundant striped	0	8	0	Widely used
1	Mirrored	1	8	8	EMC, Compaq (Tandem), IBM
2	Memory-style ECC	1	8	4	
3	Bit-interleaved parity	1	8	1	Storage Concepts
4	Block-interleaved parity	1	8	1	Network Appliance
5	Block-interleaved distributed parity	1	8	1	Widely used
6	P+Q redundancy	2	8	2	



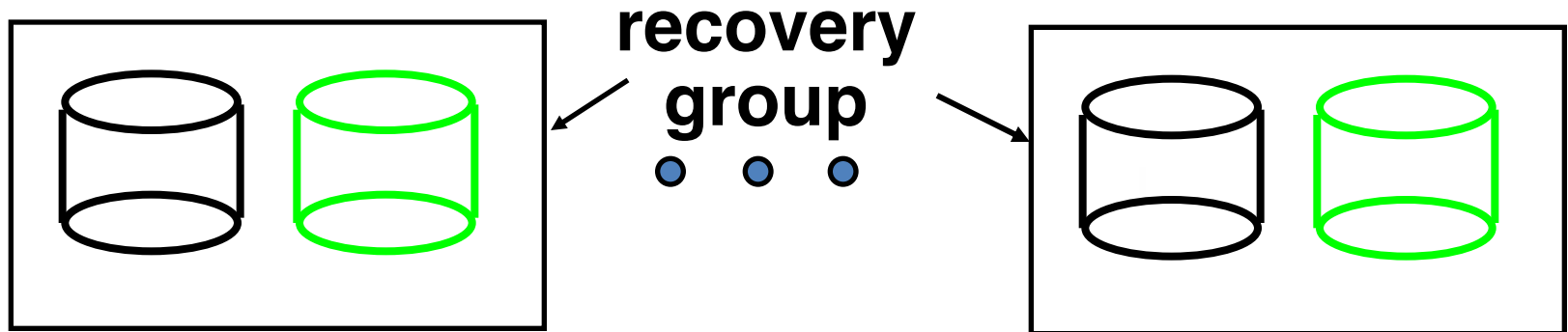


# RAID 0: No Redundancy

---

- ❑ Data is striped across a disk array but there is no redundancy to tolerate disk failure.
- ❑ It also improves performance for large accesses, since many disks can operate at once.
- ❑ RAID 0 something of a misnomer as there is no Redundancy

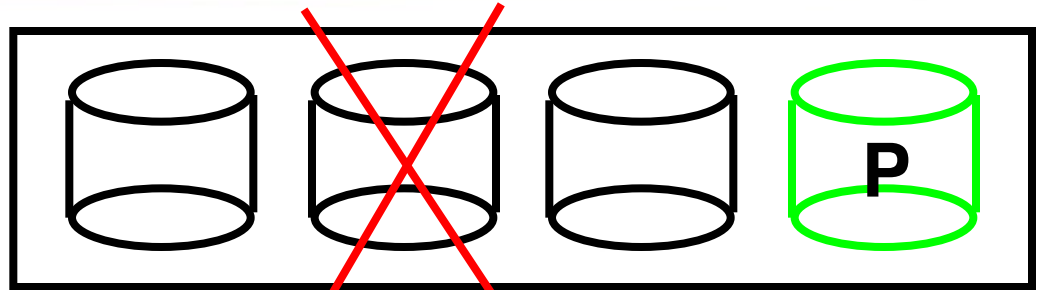
# RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its “**mirror**”  
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
- (RAID 2 not interesting, so skip)

# RAID 3: Bit-Interleaved Parity Disk

```
10010011
11001101
10010011
...
```



logical record

Striped physical records

1	1	1	1
0	1	0	1
0	0	0	0
1	0	1	0
0	1	0	1
0	1	0	1
1	0	1	0
1	1	1	1

P contains sum of other disks per stripe mod 2 (“parity”)

If disk fails, subtract P from sum of other disks to find missing information



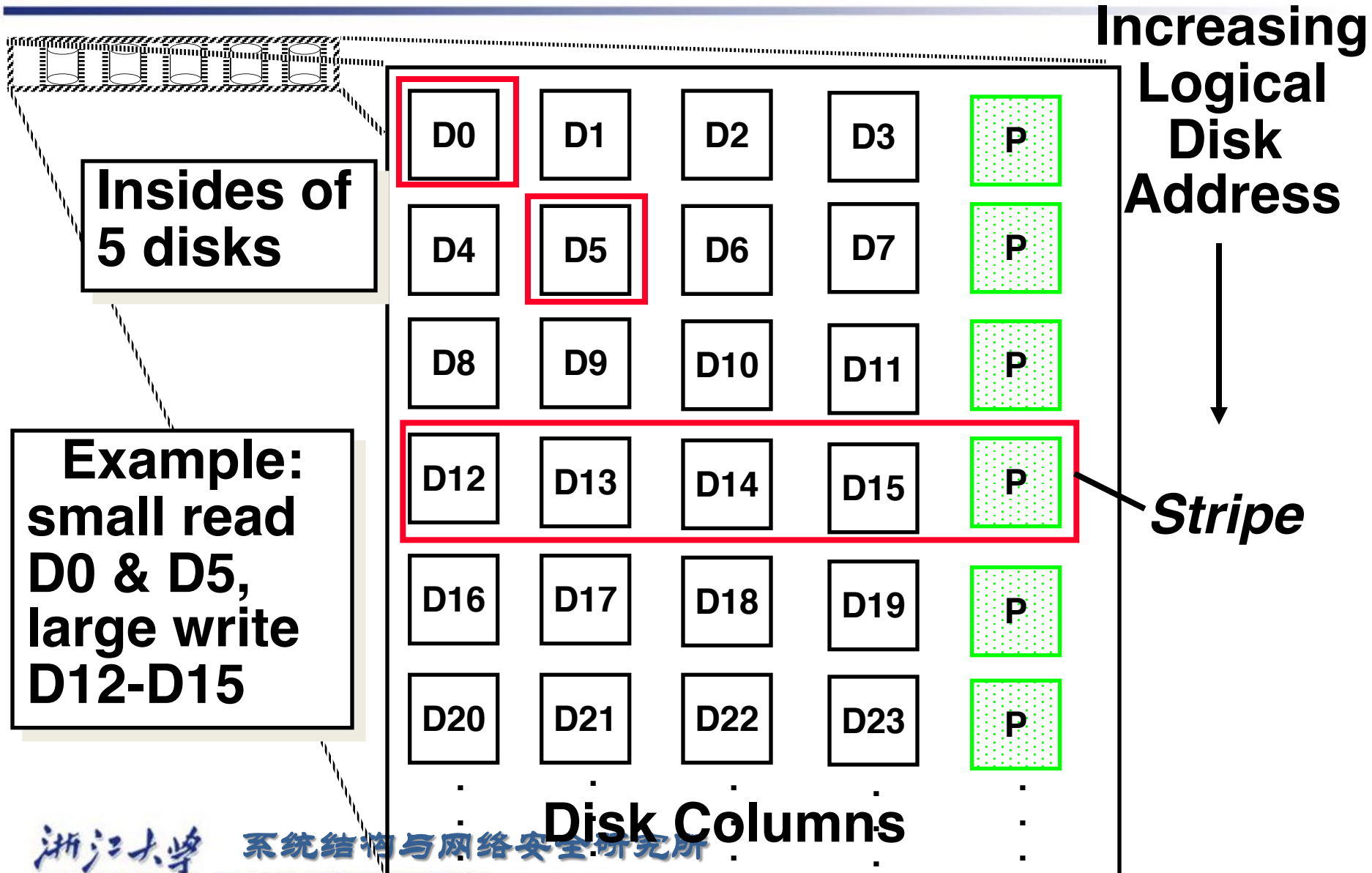
# High I/O Rate Parity

## □ Inspiration for RAID 4

- RAID 3 relies on parity **disk** to discover errors on Read
- But every sector has an error detection field
  - Rely on error detection field to catch errors on read, not on the parity disk
  - Allows independent reads to different disks simultaneously



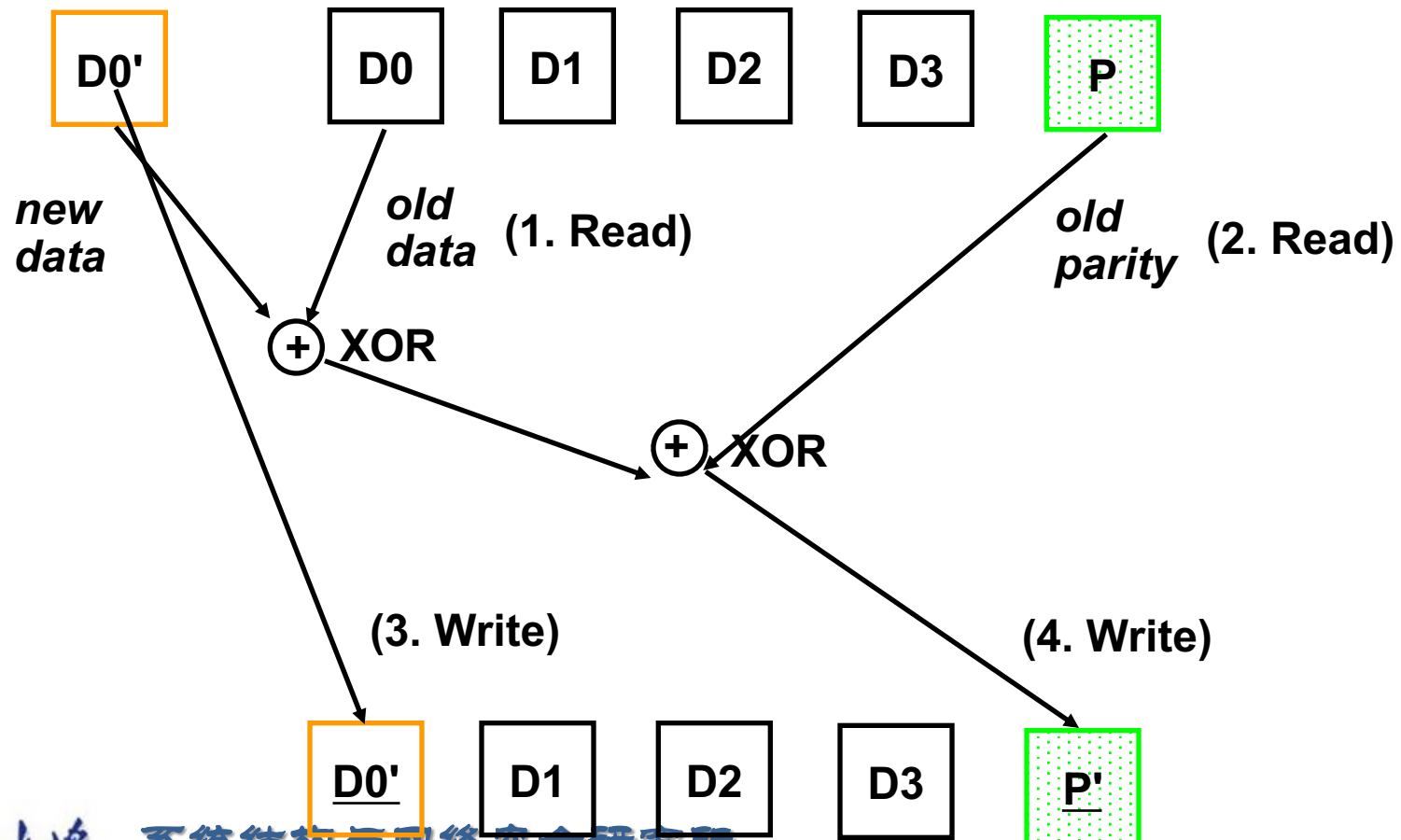
# RAID 4: Block-Interleaved Parity



# Problems of Disk Arrays: Small Writes

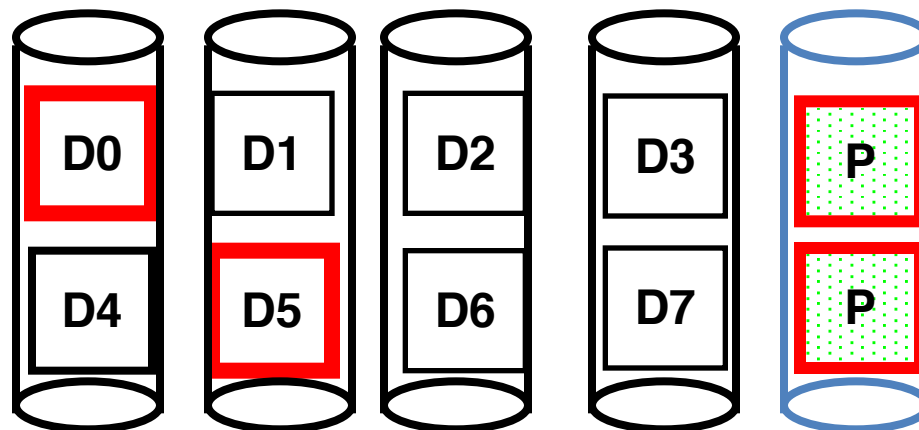
## RAID-4: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



# Inspiration for RAID 5

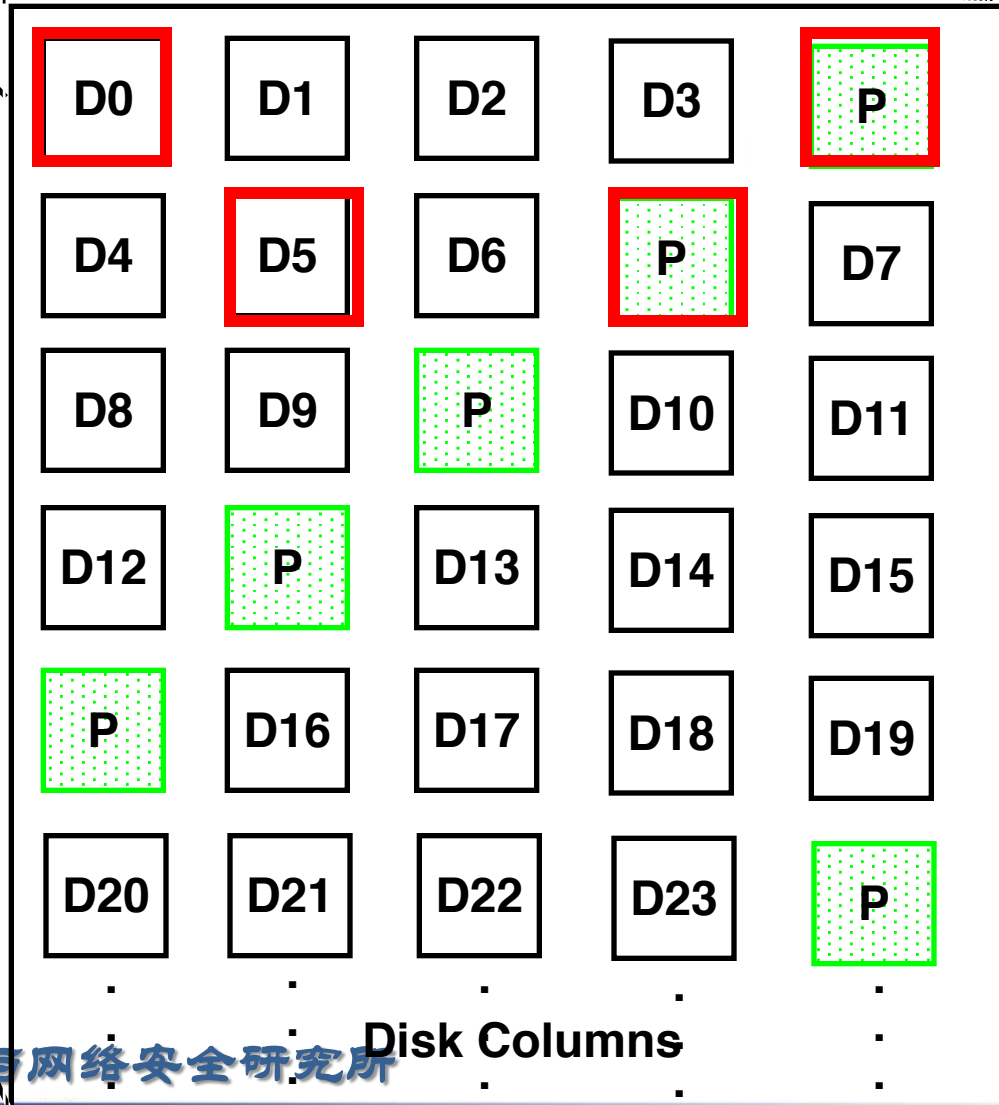
- ❑ RAID 4 works well for small reads
- ❑ Small writes (write to one disk):
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- ❑ Small writes are limited by Parity Disk: Write to D0, D5 both also write to P disk



# RAID 5: High I/O Rate Interleaved Parity

**Independent writes possible because of interleaved parity**

**Example:  
write to  
D0, D5  
uses disks  
0, 1, 3, 4**



Increasing  
Logical  
Disk  
Addresses





# RAID 6: P+Q Redundancy

---

- When a single failure correction is not sufficient, Parity can be generalized to have a second calculation over data and another check disk of information.

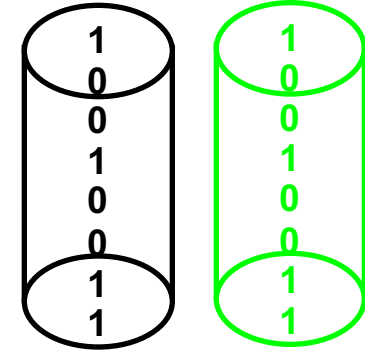
# Summary: RAID Techniques

- *Disk Mirroring, Shadowing (RAID 1)*

Each disk is fully duplicated onto its "shadow"

Logical write = two physical writes

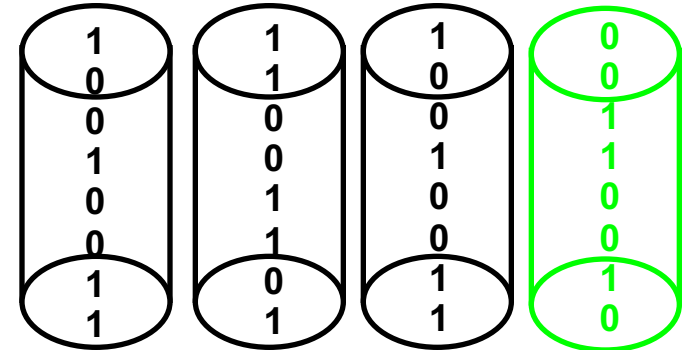
100% capacity overhead



- *Parity Data Bandwidth Array (RAID 3)*

Parity computed horizontally

Logically a single high data bw disk

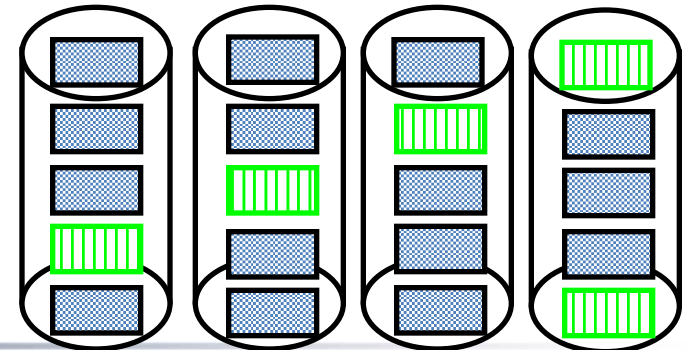


- *High I/O Rate Parity Array (RAID 5)*

Interleaved parity blocks

Independent reads and writes

Logical write = 2 reads + 2 writes

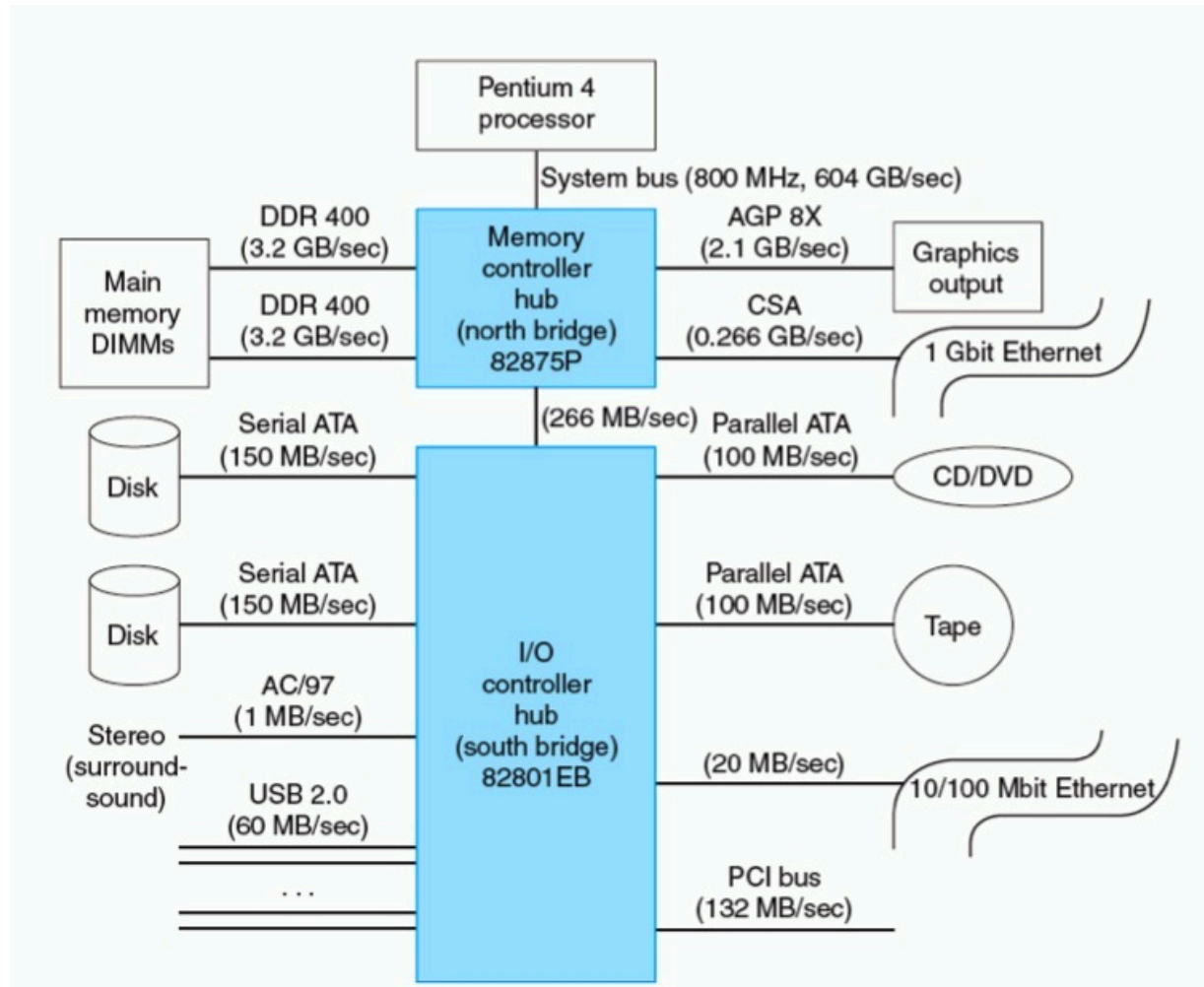




Which of the following are true about RAID levels 1, 3, 4, 5, and 6?

1. RAID systems rely on redundancy to achieve high availability.
2. RAID 1 (mirroring) has the highest check disk overhead.
3. For small writes, RAID 3 (bit-interleaved parity) has the worst throughput.
4. For large writes, RAID 3, 4, and 5 have the same throughput.

## A.4 Buses and Other Connections between Processors Memory, and I/O Devices





# Buses and Other Connections between Processors Memory, and I/O Devices



- **Bus:** Shared communication link (one or more wires)
- **Difficult Design:**
  - may be bottleneck
  - length of the bus
  - number of devices
  - tradeoffs (fast bus accesses and high bandwidth)
  - support for many different devices
  - cost



# Bus Basics

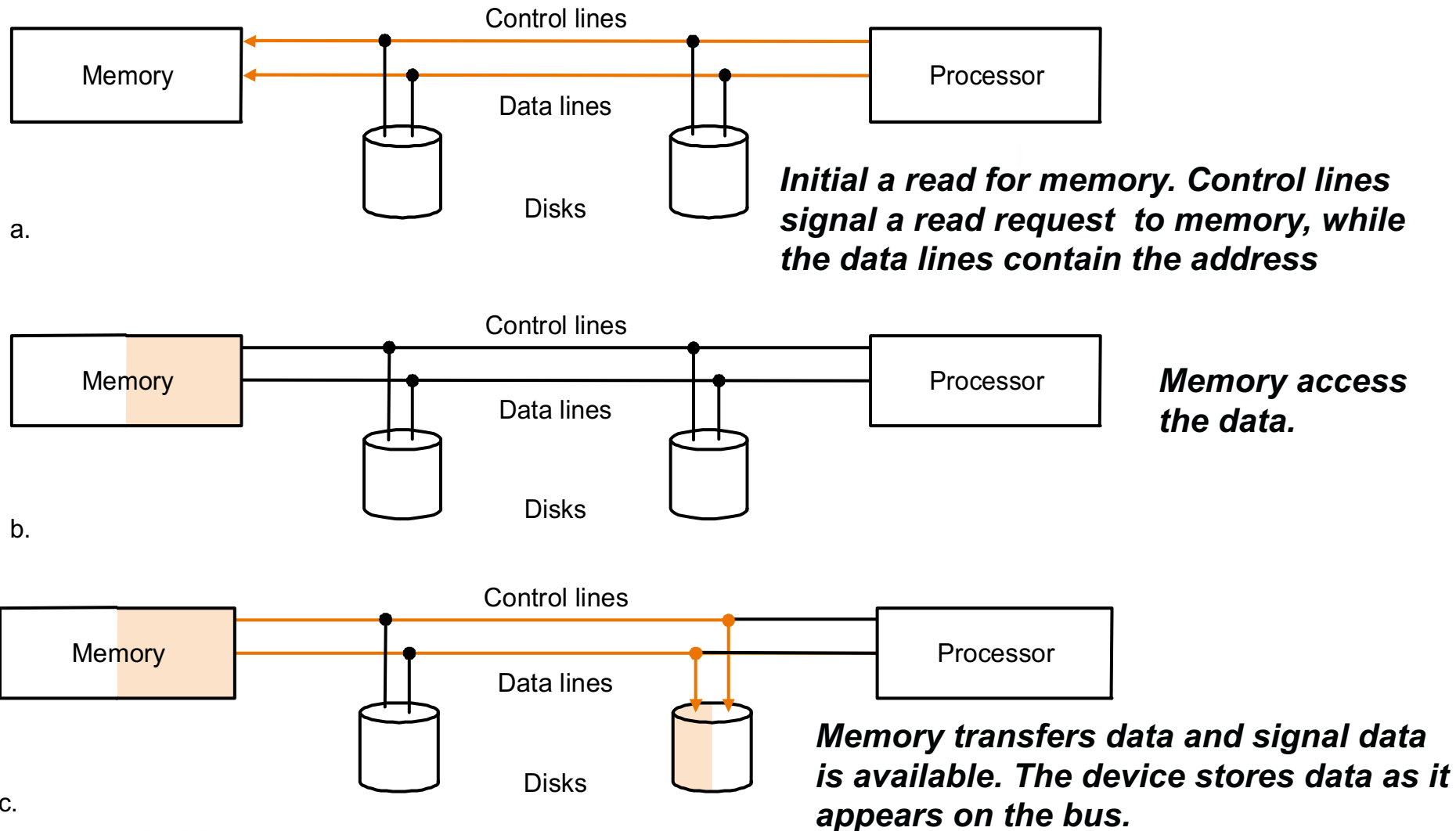
## □ A bus contains two types of lines

- **Control lines:** signal requests and acknowledgments, and to indicate what types of information is on the data lines.
- **Data lines:** carry information (e.g., data, addresses, and complex commands) between the source and the destination.

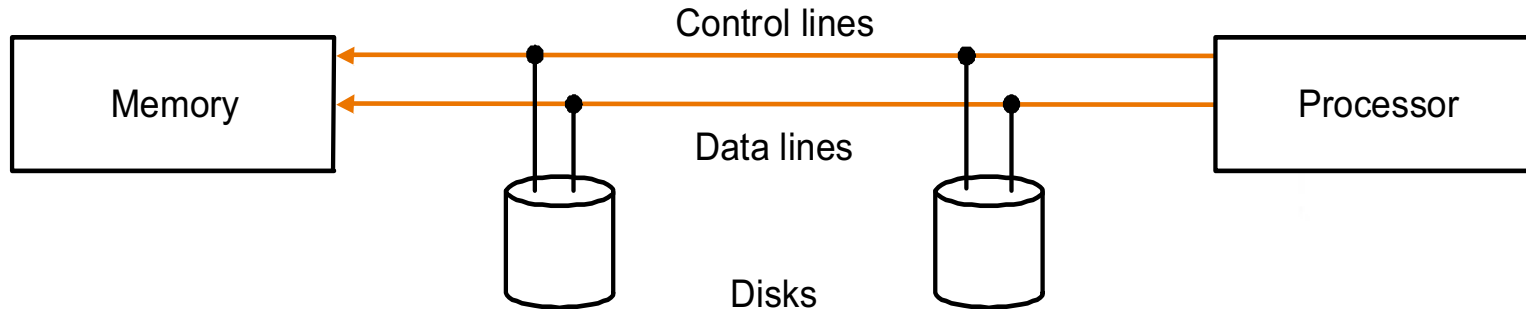
## □ Bus transaction

- include two parts: sending the address and receiving or sending the data
- two operations
  - **input:** inputting data from the device to memory
  - **output:** outputting data to a device from memory

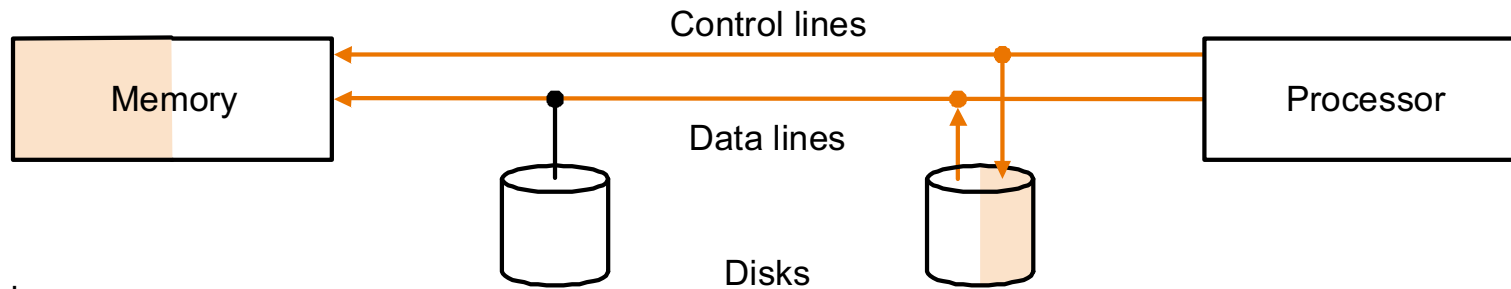
# Output operation



# Input operation



- a. ***Control lines indicate a write request for memory, while the data lines contain the address***

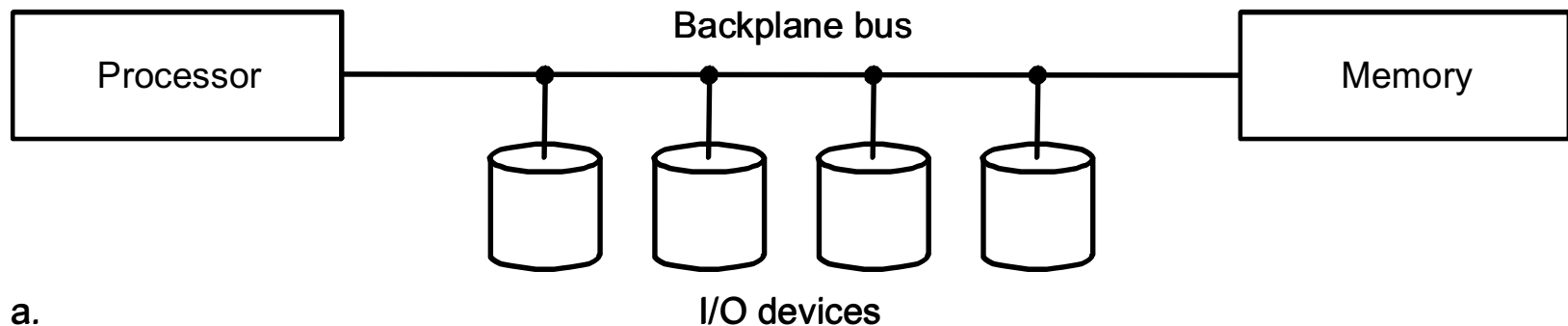


b.

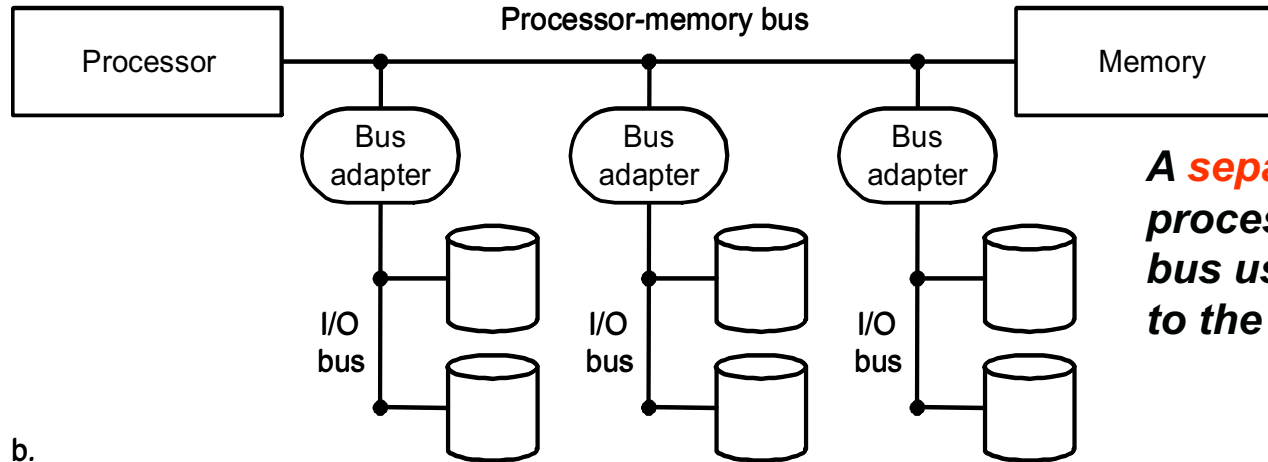
***When the memory is ready, it signals the device, which then transfers the data. The memory will store the data as it receives it. The device need not wait for the store to be completed.***

# Types of buses

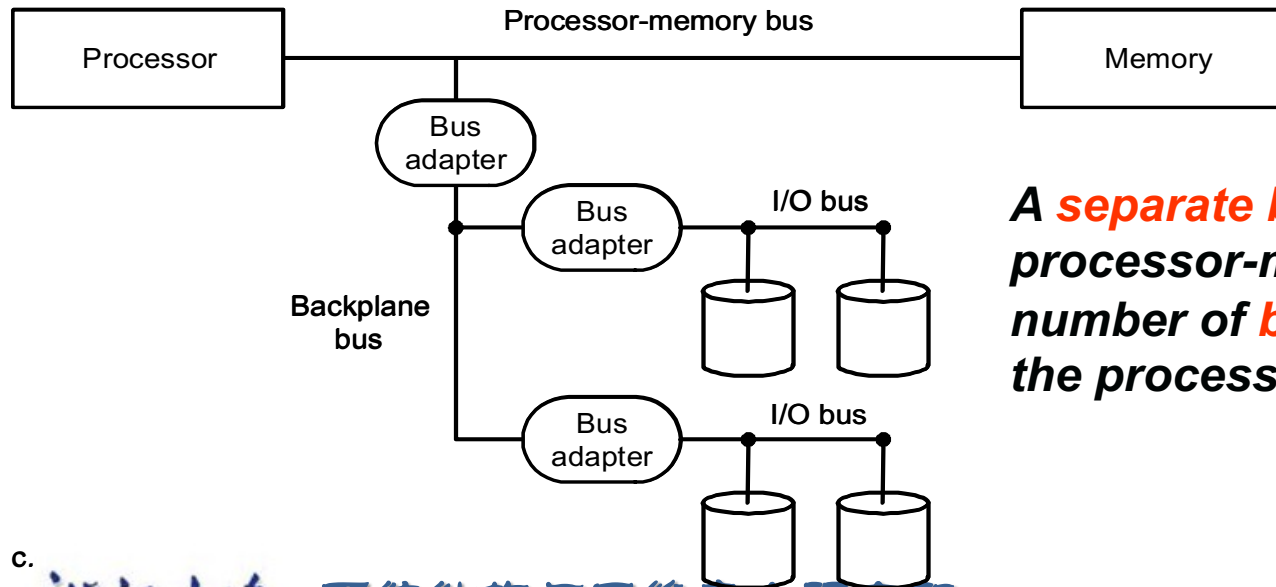
- ❑ **processor-memory** : short high speed, custom design)
- ❑ **backplane** : high speed, often standardized, e.g., PCI)
- ❑ **I/O** : lengthy, different devices, standardized, e.g., SCSI)



*Older PCs often use a **single bus** for processor-to-memory communication, as well as communication between I/O devices and memory.*



*A **separate bus** is used for processor-memory traffic. The I/O bus use a **bus adapter** to interface to the processor-memory bus.*



*A **separate bus** is used for processor-memory traffic. A small number of **backplane buses** tap into the processor-memory bus.*

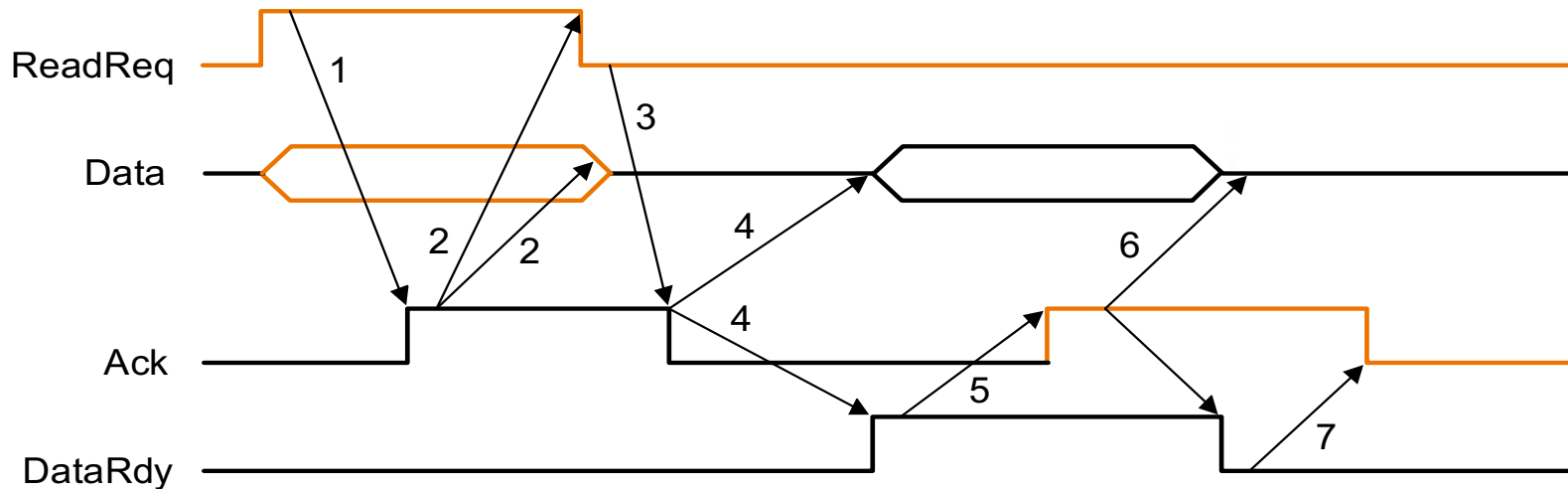


# Synchronous vs. Asynchronous

---

- ❑ **Synchronous bus:** use a clock and a fixed protocol, fast and small but every device must operate at same rate and clock skew requires the bus to be short
- ❑ **Asynchronous bus :** don't use a clock and instead use handshaking
- ❑ **Handshaking protocol:** A serial of steps used to coordinate asynchronous bus transfers.

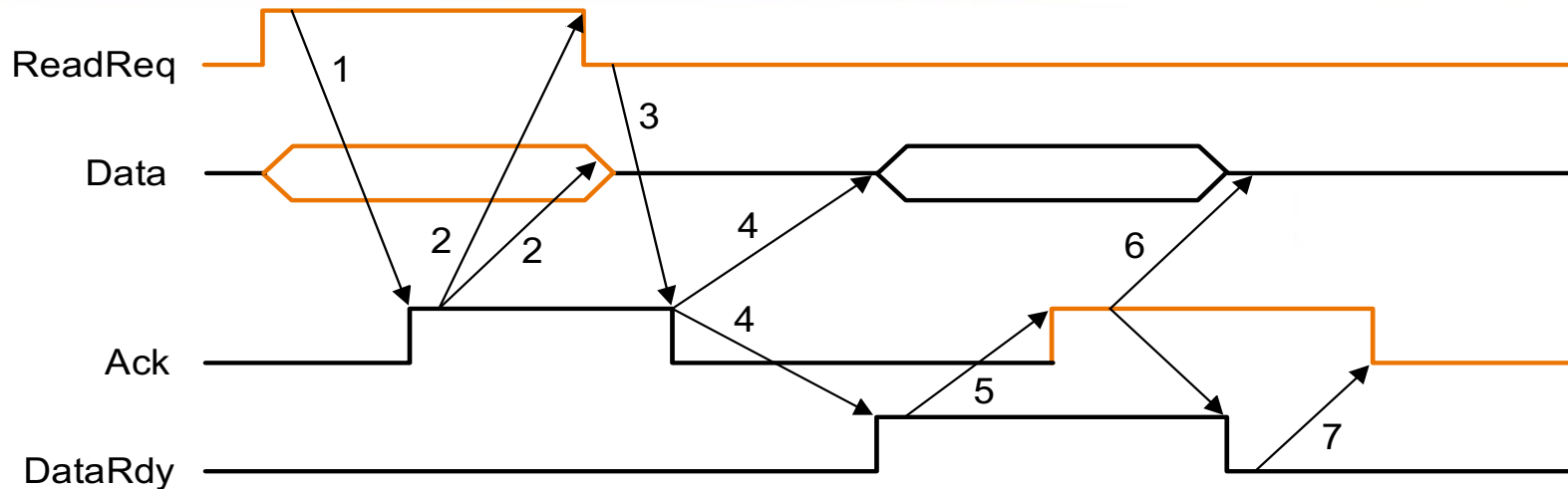
# Asynchronous example



- 1. When memory sees the ReadReq line, it reads the address from the data bus, begin the memory read operation, then raises Ack to tell the device that the ReadReq signal has been seen.**
- 2. I/O device sees the Ack line high and releases the ReadReq data lines.**
- 3. Memory sees that ReadReq is low and drops the Ack line.**



# Asynchronous example



- 4. When the memory has the data ready, it places the data on the data lines and raises DataRdy.**
- 5. The I/O device sees DataRdy, reads the data from the bus, and signals that it has the data by raising ACK.**
- 6. The memory sees Ack signals, drops DataRdy, and releases the data lines.**
- 7. Finally, the I/O device, seeing DataRdy go low, drops the ACK line, which indicates that the transmission is completed.**

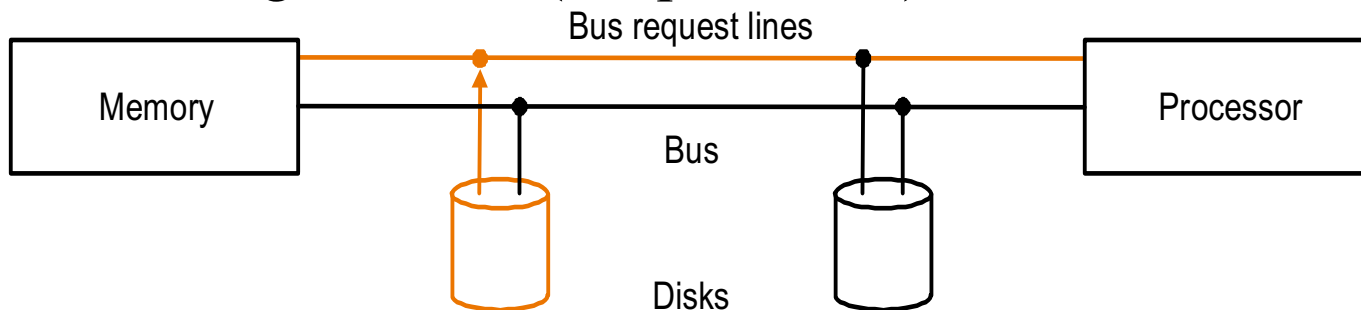
# Bus Arbitration

## □ Obtaining Access to the Bus

- “Without any control, multiple device desiring to communicate could each try to assert the control and data lines for different transfers!”
- So, a **bus master** is needed. Bus masters initiate and control all bus requests.

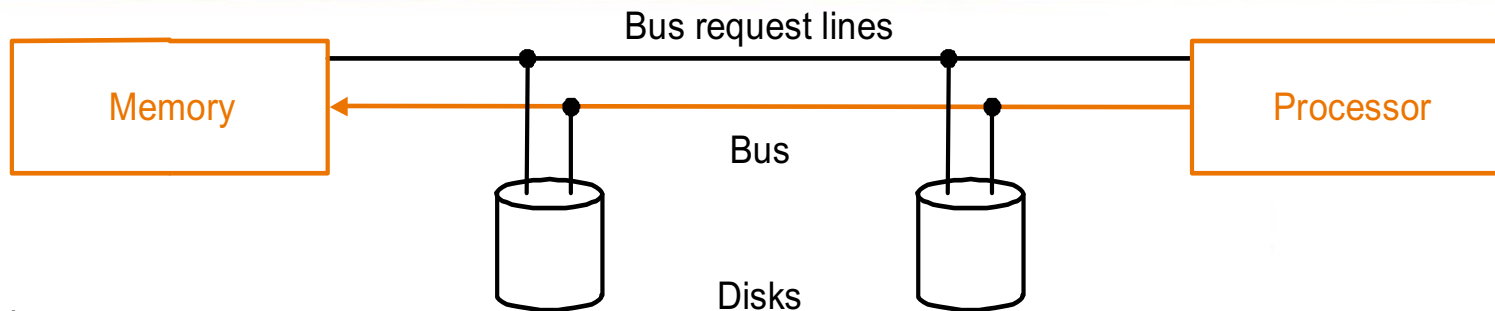
e.g., processor is always a bus master.

- Example: the initial steps in a bus transaction with a single master (the processor).



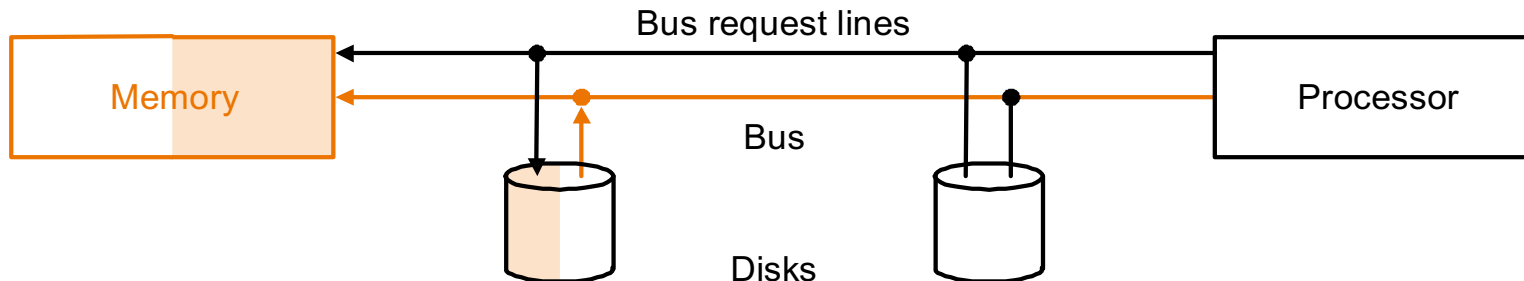
*First, the device generates a bus request to indicate to the processor that it wants to use the bus.*

# Bus Arbitration



b.

***The processor responds and generates appropriate bus control signals. For example, if the device wants to perform output from memory, the processor asserts the read request lines to memory.***



c.

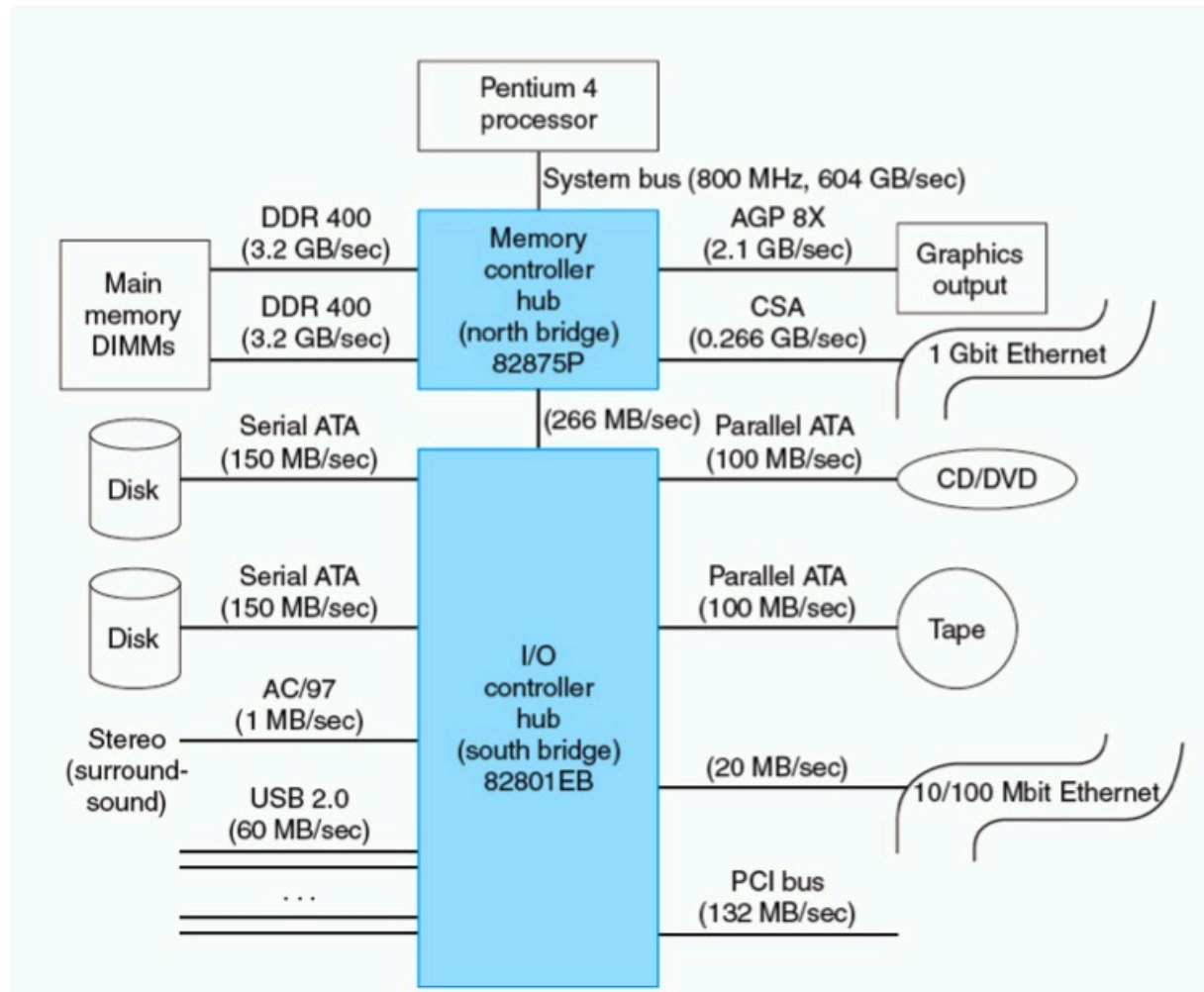
***The processor also notifies the device that its bus request is being processed; as a result, the device knows it can use the bus and places the address for the request on the bus.***



# Bus Arbitration

- ❑ Deciding which bus master gets to use the bus next  
In a bus arbitration scheme, a device wanting to use the bus signals a bus request and is later granted the bus.
- ❑ Four bus arbitration schemes:
  - ❑ *daisy chain arbitration (not very fair)*
  - ❑ *centralized, parallel arbitration (requires an arbiter), e.g., PCI*
  - ❑ *self selection, e.g., NuBus used in Macintosh*
  - ❑ *collision detection, e.g., Ethernet*
- ❑ Two factors in choosing which device to grant the bus:
  - ❑ bus priority
  - ❑ fairness

# The Buses and Networks of Pentium



# Performance analysis of Synchronous versus Asynchronous buses



**Assume:** The synchronous bus has a clock cycle time of 50 ns, and each bus transmission takes 1 clock cycle .

The asynchronous bus requires 40 ns per handshake.

The data portion of both buses is 32 bits wide.

**Question:** Find the bandwidth for each bus when reading one word from a 200-ns memory.

**Answer:** *synchronous bus:*

the bus cycles is 50 ns. The steps and times required for the synchronous bus are follows:

1. *Send the address to memory : 50ns*
2. *Read the memory : 200ns*
3. *Send the data to the device : 50ns*

Thus, the total time is 300 ns. So,

*the bandwidth = 4bytes/300ns = 4MB/0.3seconds*  
*= 13.3MB/second*

# Performance analysis of Synchronous versus Asynchronous buses



**Answer:** *asynchronous bus:*

we realize that several of the steps can be overlapped with the memory access time.

In particular, the memory receives the address at the end of step 1 and does not need to put the data on the bus until the beginning of step 5; steps 2, 3, and 4 can overlap with the memory access time.

This leads to the following timing:

*step1: 40ns*

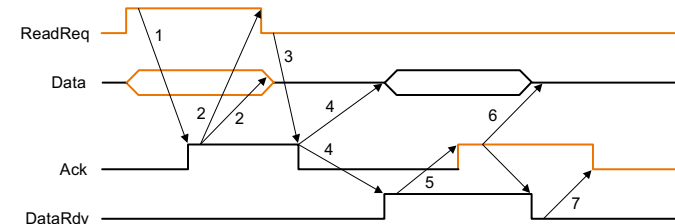
*step2,3,4: maximum( $2 \times 40ns + 40ns$ ,  $200ns$ ) =  $200ns$*

*step5,6,7:  $3 \times 40ns = 120ns$*

Thus, the total time is 360ns, so

*the maximum bandwidth =  $4bytes / 360ns = 4MB / 0.36seconds$   
=  $11.1MB/second$*

Accordingly, the synchronous bus is only about 20% faster.







# Increasing the Bus Bandwidth

**Suppose** we have a system with the following characteristic:

1. A memory and bus system supporting block access of 4 to 16 32-bit words
2. A 64-bit synchronous bus clocked at 200 MHz, with each 64-bit transfer taking 1 clock cycle, and 1 clock cycle required to send an address to memory.
3. Two clock cycles needed between each bus operation.
4. A memory access time for the first four words of 200ns; **each additional set of four words can be read in 20 ns.** Assume that a bus transfer of the most recently read data and a read of the next four words can be overlapped.

**Find** the sustained bandwidth and the latency for a read of 256 words for transfers that use 4-word blocks and for transfers that use 16-word blocks. Also compute effective number of bus transactions per second for each case.





# Answer: the 4-word block transfers

**Answer: the 4-word block transfers:**

**each block takes**

**64-bit synchronous bus**

- 1. 1 clock cycle to send the address to memory*
- 2.  $200\text{ns}/(5\text{ns}/\text{cycle}) = 40$  clock cycles to read memory*
- 3. 2 clock cycles to send the data from the memory*
- 4. Two clock cycles needed between each bus operation.*

***This is a total of 45cycles.***

**There are  $256/4 = 64$  blocks.**

**So the transfer of 256 words takes**

$$45 \times 64 = 2880 \text{ clock cycles}$$

**The latency for the transfer of 256 words is:**

$$2880 \text{ cycles} \times (5\text{ns}/\text{cycle}) = 14,400\text{ns}.$$

**so the number of bus transactions per second is:**

$$64 \text{ transactions} \times \frac{1 \text{ second}}{14,400 \text{ ns}} = 4.44 \text{M transactions/second}$$

**The bus bandwidth is:**

$$(256 \times 4) \text{ bytes} \times \frac{1 \text{ second}}{14,400 \text{ ns}} = 71.11 \text{ MB/sec}$$



# Answer: the 16-word block transfers:

**the first block requires**

- 1. 1 clock cycle to send an address to memory*
- 2. 200ns or 40 cycles to read the first four words in memory.*
- 3. 2 cycles to transfer the data of the set, during which time the read of the next 4-word set is started.*
- 4. It only takes 20ns or 4 cycles to read the next set. After the read is completed, the set will be transferred. Each of the three remaining sets requires repeating only the last two steps.*
- 5. Two clock cycles needed between each bus operation.*

Thus, the total number of cycles for each 16- word block is:

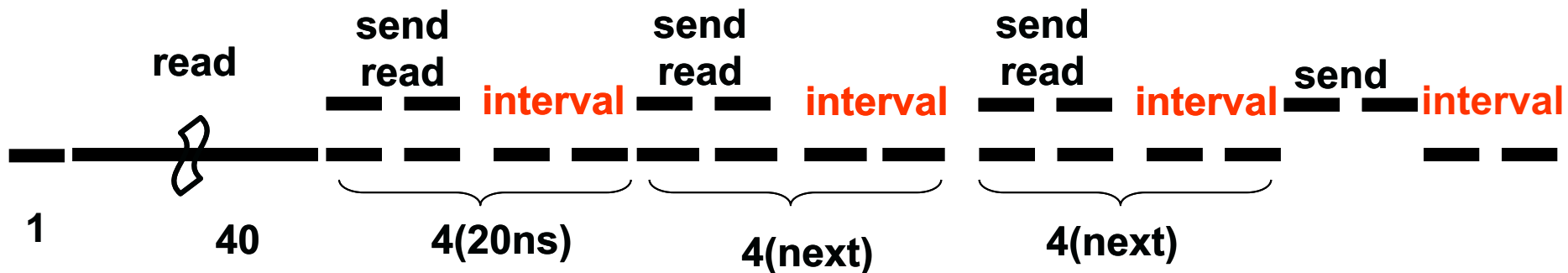
$$1+40+4 \times 3+2+2=57 \text{ cycles.}$$

There are  $256/16=16$  blocks.

so the transfer of 256 words takes  $57 \times 16=912 \text{ cycles.}$

Thus the latency is:

$$912 \text{ cycles} \times 5 \text{ ns/cycles} = 4560 \text{ ns.}$$





**The number of bus transactions per second with 16-word blocks is:**

$$16 \text{ transactions} \times \frac{1 \text{ second}}{4560 \text{ ns}} = 3.51 \text{ M transactions/second}$$

**The bus bandwidth with 16-word blocks is:**

$$(256 \times 4) \text{ bytes} \times \frac{1 \text{ second}}{4560 \text{ ns}} = 224.56 \text{ MB/sec}$$

**Now, let's put two bus bandwidth together:**

**4-word blocks: 71.11 MB/sec**

**16-word blocks: 224.56 MB/sec**

**The bandwidth for the 16-word blocks is 3.16 times higher than for the 4-word blocks.**



# Increasing the Bus Bandwidth

---

- ❑ Increasing data bus width
- ❑ Use separate address and data lines
- ❑ transfer multiple words



## A.5 Interfacing I/O Devices to the Memory, Processor, and Operating System

### □ Three characteristics of I/O systems

- *shared* by multiple programs using the processor.
- often use *interrupts* to communicate information about I/O operations.
- The low-level control of an I/O devices is *complex*

### □ Three types of communication are required:

- The OS must be able to give *commands* to the I/O devices.
- The device must be able to *notify* the OS, when I/O device *completed* an operation or has encountered an *error*.
- Data must be transferred between memory and an I/O device

# Giving Commands to I/O Devices

## Two methods used to address the device



### □ **memory-mapped I/O:**

- portions of the memory address space are assigned to I/O devices, and lw and sw instructions can be used to access the I/O port.

### □ **special I/O instructions**

- exp: in al, port out port, al

### □ **command port ,data port**

- The Status register (a done bit, an error bit.....)
- The Data register, The command register





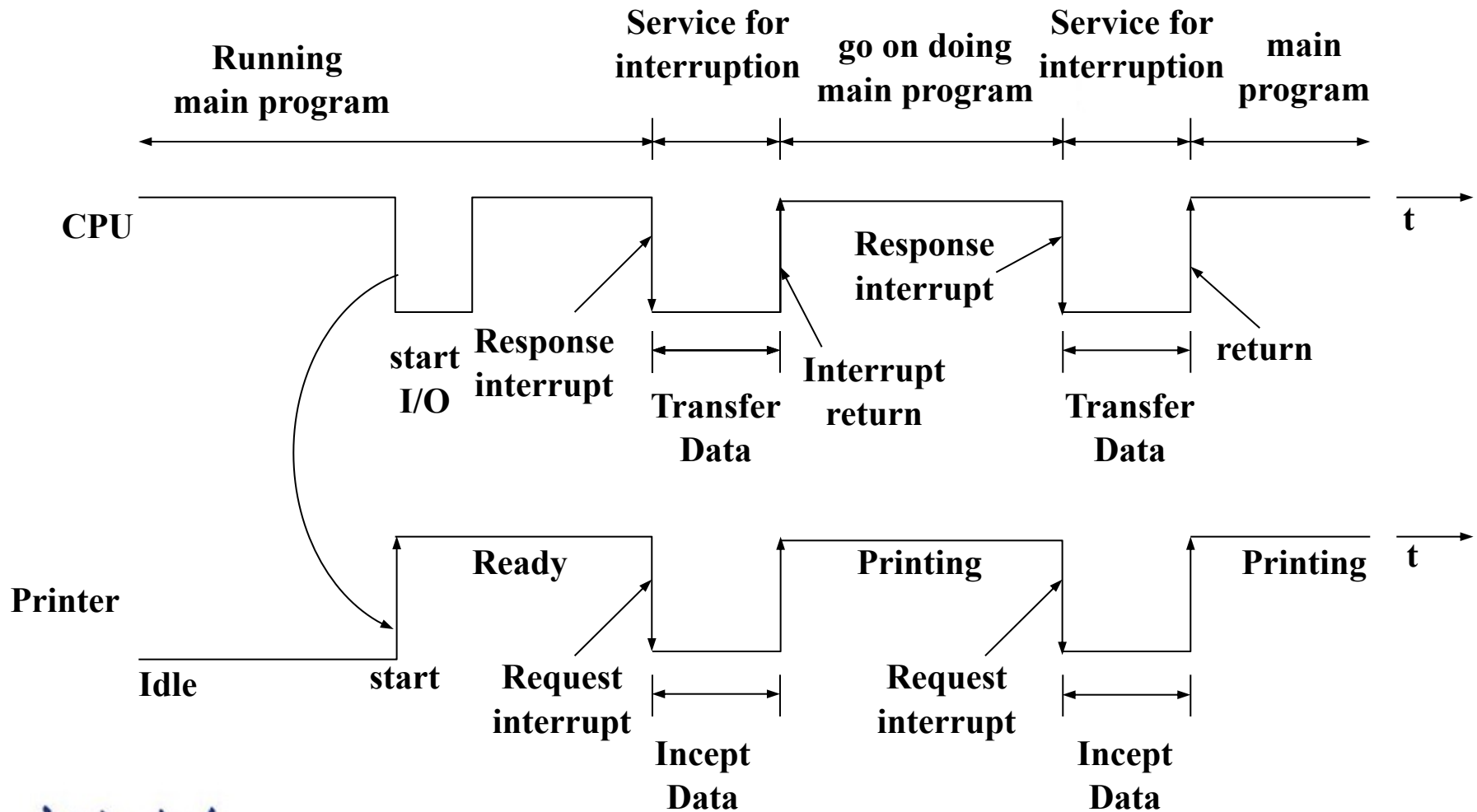
# Communication with the Processor

---

- ❑ **Polling:** The processor periodically checks status bit to see if it is time for the next I/O operation.
- ❑ **Interrupt:** When an I/O device wants to notify processor that it has completed some operation or needs attentions, it causes processor to be interrupted.
- ❑ **DMA** (*direct memory access*): the device controller transfer data directly to or from memory without involving processor.

# Interrupt-Driven I/O mode

## Advantage: concurrent operation





DMA-- I/O---M

# I/O DIRECT ACCESS MEMORY



# DMA transfer mode

## □ A DMA transfer need three steps:

- The processor **sets up** the DMA by supplying some information, including the ***identity of the device, the operation, the memory address that is the source or destination of the data to be transferred, and the number of bytes to transfer.***
- The DMA **starts** the **operation** on the device and arbitrates for the **bus**. If the request requires more than one transfer on the bus, the DMA unit generates the next memory address and initiates the next transfer.
- Once the DMA transfer is complete, the controller **interrupts** the processor, which then examines whether errors occur.



# Compare polling, interrupts, DMA

---

- The disadvantage of polling is that it wastes a lot of processor time. When the CPU polls the I/O devices periodically, the I/O devices maybe have no request or have not get ready.
- If the I/O operations is interrupt driven, the OS can work on other tasks while data is being read from or written to the device.
- Because DMA doesn't need the control of processor, it will not consume much of processor time.



# Overhead of Polling in an I/O System

**Assume:** that the number of clock cycles for a polling operation is 400 and that processor executes with a 500-Mhz clock.

**Determine** the fraction of CPU time consumed for the mouse, floppy disk, and hard disk.

We assuming that you poll often enough so that no data is ever lost and that those devices are potentially always busy.

*We assume again that:*

- 1. The mouse must be polled 30 times per second to ensure that we do not miss any movement made by the user.*
- 2. The floppy disk transfers data to the processor in 16-bit units and has a data rate of 50 KB/sec. No data transfer can be missed.*
- 3. The hard disk transfers data in four-word chunks and can transfer at 4 MB/sec. Again, no transfer can be missed.*

# Answer

*the mouse:*

**clock cycles per second for polling :**

$$30 \times 400 = 12,000 \text{ cycles}$$

**Fraction of the processor clock cycles consumed is**

$$\frac{12 \times 10^3}{500 \times 10^6} = 0.002\%$$

*the floppy disk:*

**the number of polling access per second:**

$$50KB/2B = 25K$$

**clock cycles per second for polling:  $25K \times 400$  cycles**

**Fraction of the processor clock cycles consumed:**

$$\frac{10 \times 10^6}{500 \times 10^6} = 2\%$$

# Answer



## *the hard disk:*

**The number of polling access per second :**

$$4\text{MB}/16\text{B} = 250\text{K}$$

**Clock cycles per second for polling =  $250\text{K} \times 400$**

**Fraction of the processor clock cycles consumed:**

$$\frac{100 \times 10^6}{500 \times 10^6} = 20\%$$

**Now, let's put three fractions together:**

**Mouse: 0.002%**

**Floppy disk: 2%**

**Hard disk: 20%**

**Clearly, polling can be used for the mouse without much performance impact on the processor, but it is unacceptable for a hard disk on this machine.**



# Overhead of Interrupt-Driven I/O



**Suppose** we have the same hard disk and processor we used in the former example, but we used interrupt-driven I/O. The overhead for each transfer, including the interrupt, is 500 clock cycles. Find the fraction of the processor consumed if the hard disk is only transferring data 5% of the time.

**Answer:** First, we assume the disk is **transferring data 100%** of the time. So, the interrupt rate is the same as the polling rate.

*Cycles per second for disk is:*

$$250K \times 500 = 125 \times 10^6 \text{ cycles per second}$$

Fraction of the processor consumed during a transfer is:

$$\frac{125 \times 10^6}{500 \times 10^6} = 25\%$$

**Now, we assume that the disk is only transferring data 5% of the time. The fraction of the processor time consumed on average is:**

$$25\% \times 5\% = 1.25\%$$

**As we can see, no CPU time is needed when an interrupt-driven I/O device is not actually transferring. This is the major advantage of an interrupt-driven interface versus polling.**

# Overhead of I/O Using DMA



**Suppose** we have the same hard disk and processor we used in the former example.

Assume that the initial setup of a DMA transfer takes 1000 clock cycles for the processor, and assume the handling of the interrupt at DMA completion requires 500 clock cycles for the processor.

The hard disk has a transfer rate of 4MB/sec and uses DMA. The average transfer from disk is 8 KB. Assume the disk is actively transferring 100% of the time.

**Please find** what fraction of the processor time is consumed.



# Answer

Time for each 8KB transfer is:

$$8KB/(4MB/second)=2 \times 10^{-3}seconds.$$

It requires the following cycles per second:

$$\frac{(1000+500) \frac{\text{cycles}}{\text{transfer}}}{2 \times 10^{-3} \frac{\text{seconds}}{\text{transfer}}} = 750 \times 10^3 \frac{\text{clock cycles}}{\text{second}}$$

$$\text{Fraction of processor time: } \frac{750 \times 10^3}{500 \times 10^6} = 0.2\%$$

Unlike either polling or interrupt-driven I/O, DMA can be used to interface a hard disk without consuming all the processor cycles for a single I/O.

# I/O Interface & Devices within Experiment

Cpu读译码省略，Why?

```
//+++++RAM & IO decode signals:
always @* begin
    data_ram_we = 0;
    counter_we = 0;
    GPIOf00000000_we = 0;
    GPIOe00000000_we = 0;
    ram_addr = 10'h0;
    ram_data_in = 32'h0;
    Peripheral_in=32'h0;
    Cpu_data4bus = 32'h0;

    case(addr_bus[31:28])
        4'h0: begin//data_ram (00000000-00000ffc, actually lower 4KB RAM)
            data_ram_we = mem_w;
            ram_addr=addr_bus[11:2];
            ram_data_in=Cpu_data2bus;
            Cpu_data4bus=ram_data_out;
        end

        4'he: begin//7 Segement LEDs(e0000000-ffffffff,4 位 7-seg display)
            GPIOe0000000_we = mem_w;
            Peripheral_in = Cpu_data2bus;
            Cpu_data4bus =counter_out;           //read from Counter
        end

        4'hf: begin//LED(f0000000-fffffffff0,8 LEDs&counter,f0000004-
                                                    ffffffff4)

            if(addr_bus[2]) begin                //f00000004
                counter_we = mem_w;
                Peripheral_in = Cpu_data2bus;     //write Counter Value
                Cpu_data4bus = counter_out;      //read from Counter;
            end

            else begin                          //f00000000
                GPIOf00000000_we = mem_w;
                Peripheral_in =Cpu_data2bus;//write Counter set &
                                                    Initialization&light LED
                Cpu_data4bus =
                    {counter0_out,counter1_out,counter2_out,9'h000,led_out,BTN,SW};
            end
        end
    endcase
end

endmodule
```

## A.6 I/O Performance Measures: Examples from Disk and File Systems

---



- ❑ Supercomputer I/O Benchmarks
- ❑ Transaction Processing I/O Benchmarks
  - **I/O rate**: the number of disk access per second, as opposed to **data rate**.
- ❑ File System I/O Benchmarks
  - MakeDir, Copy, ScanDir, ReadAll, Make

# 8.7 Designing an I/O system

---

The general approaches to designing I/O system

- ❑ Find the weakest link in the I/O system, which is the component in the I/O path that will constrain the design. Both the workload and configuration limits may dictate where the weakest link is located.
- ❑ Configure this component to sustain the required bandwidth.
- ❑ Determine the requirements for the rest of the system and configure them to support this bandwidth.

## Examples:

Consider the following computer system:


1. A CPU sustains 3 billion instructions per second and it takes average **100,000** instructions in the operating system per I/O operation.
2. A memory backplane bus is capable of sustaining a transfer rate of 1000 MB/sec.
3. SCSI-Ultra320 controllers with a transfer rate of 320 MB/sec and accommodating up to 7 disks.
4. Disk drives with a read/write bandwidth of 75 MB/sec and an average seek plus rotational latency of 6 ms.



If the workload consists of 64-KB reads (assuming the the data block is sequential on a track), and the user program need **200,000** instructions per I/O operation, **please find the maximum sustainable I/O rate and the number of disks and SCSI controllers required.**

**Answer:**

The two fixed component of the system are the **memory bus** and the **CPU**. Let's first find the I/O rate that these two components can sustain and determine which of these is the **bottleneck**.

$$\begin{aligned}
 \text{Maximum I/O rate of CPU} &= \frac{\text{Instruction execution rate}}{\text{Instruction per I/O}} \\
 &= \frac{3 \times 10^9}{(200 + 100) \times 10^3} = 10000 \frac{\text{I/Os}}{\text{seconds}} \\
 \text{Maximum I/O rate of bus} &= \frac{\text{Bus bandwidth}}{\text{Bytes per I/O}} = \frac{1000 \times 10^6}{64 \times 10^3} = 15625 \frac{\text{I/Os}}{\text{seconds}}
 \end{aligned}$$


The **CPU** is the **bottleneck**, so we can now configure the rest of the system to perform at the level dictated by the bus, **10000** I/Os per second.

Now, let's determine how many disks we need to be able to Accommodate 10000 I/Os per second. To find the number of disks, we first find the time per I/O operation at the disk:

$$\begin{aligned}\textit{Time per I/O at disk} &= \text{Seek/rotational time} + \text{Transfer time} \\ &= 6 \text{ ms} + \frac{64\text{KB}}{75\text{MB/sec}} = 6.9 \text{ ms}\end{aligned}$$

This means each disk can complete 1000ms/6.9ms, or 146 I/Os per second. To saturate the bus,the system need 10000/146≈**69** disks.

To compute the number of SCSI buses, we need to know the average transfer rate per disk, which is given by:

$$\text{Transfer rate} = \frac{\text{Transfer size}}{\text{Transfer time}} = \frac{64\text{KB}}{6.9\text{ms}} \approx 9.56\text{MB/sec}$$

$$69 \times 9.56\text{MB/sec} < 320\text{MB/s}$$

$$69 \div 7 \approx 10$$

Assuming the disk accesses are not clustered so that we can use all the bus bandwidth, we can place 7 disks per bus and controller. This means we will need  $69/7$ , or **10** SCSI buses and controllers.



◎ END