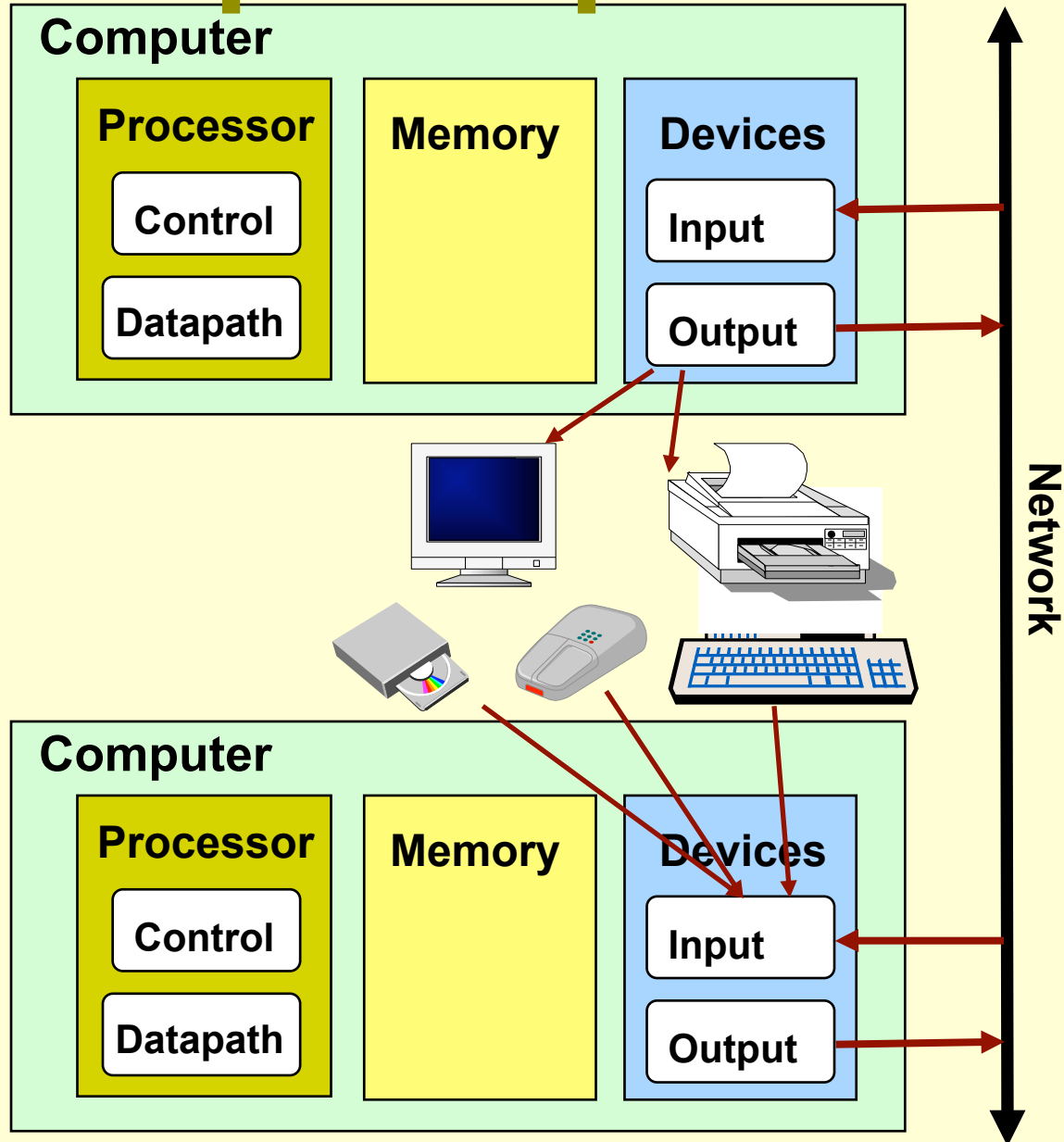


# **CMSC 611: Advanced Computer Architecture**

I/O and Storage

# 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



# Impact of I/O on System Performance

Suppose we have a benchmark that executes in 100 seconds of elapsed time, where 90 seconds is CPU time and the rest is I/O time. If the CPU time improves by 50% per year for the next five years but I/O time does not improve, how much faster will our program run at the end of the five years?

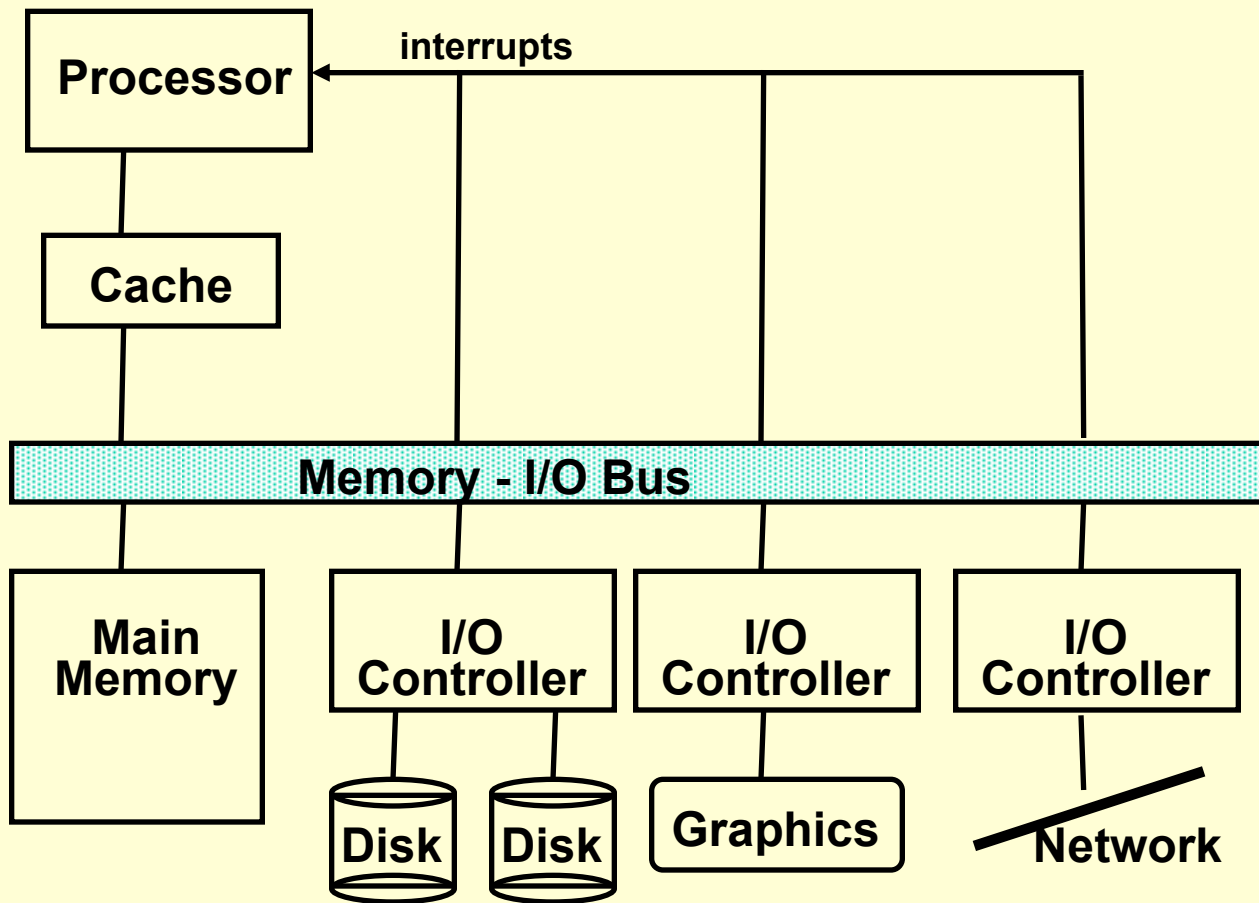
Answer: Elapsed Time = CPU time + I/O time

After n years	CPU time	I/O time	Elapsed time	% I/O time
0	90 Seconds	10 Seconds	100 Seconds	10%
1	$\frac{90}{1.5} = 60$ Seconds	10 Seconds	70 Seconds	14%
2	$\frac{60}{1.5} = 40$ Seconds	10 Seconds	50 Seconds	20%
3	$\frac{40}{1.5} = 27$ Seconds	10 Seconds	37 Seconds	27%
4	$\frac{27}{1.5} = 18$ Seconds	10 Seconds	28 Seconds	36%
5	$\frac{18}{1.5} = 12$ Seconds	10 Seconds	22 Seconds	45%

Over five years:

CPU improvement =  $90/12 = 7.5$  **BUT** System improvement =  $100/22 = 4.5$

# Typical I/O System



# I/O Device Examples

Device	Behavior	Partner	Data Rate (KB/sec)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Line Printer	Output	Human	1.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Optical Disk	Storage	Machine	500.00
Magnetic Disk	Storage	Machine	5,000.00
Network-LAN	Input or Output	Machine	20 – 1,000.00
Graphics Display	Output	Human	30,000.00

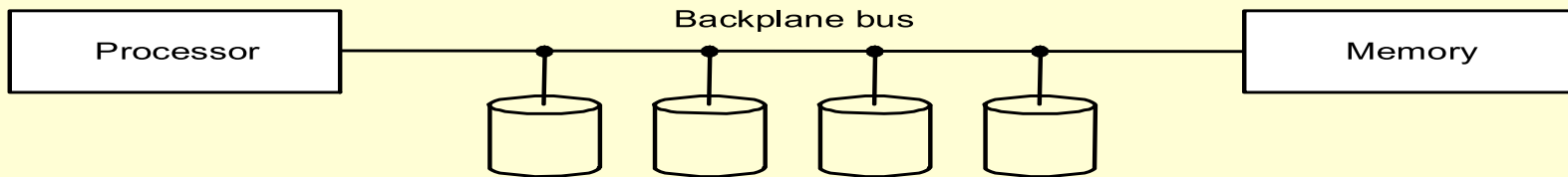
# Connecting I/O Devices

- 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)

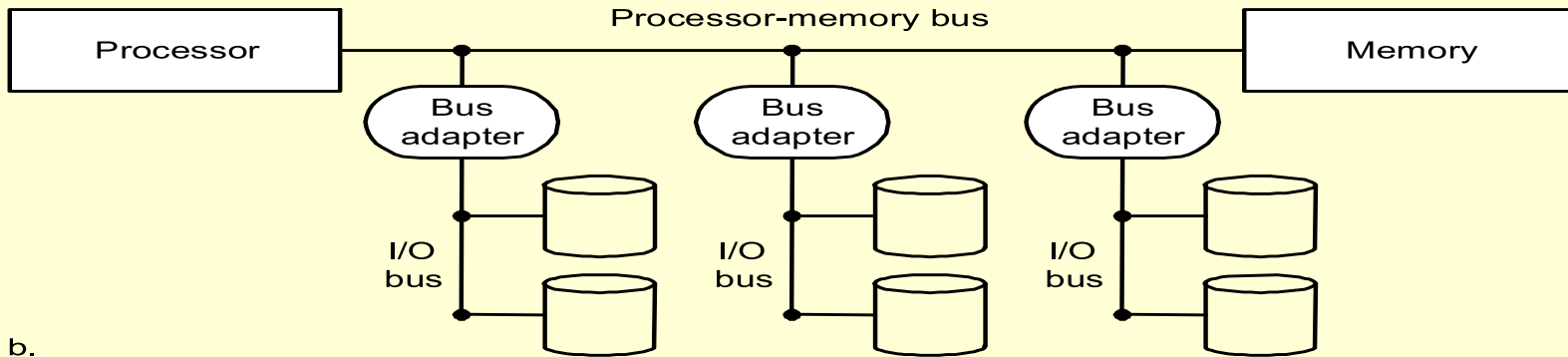
# Types of Buses

1. Processor-memory (local) bus:
    - Generally short and high speed to maximize the bandwidth
  2. I/O Bus
    - Lengthy and supports multiple data rates and devices
    - Must handle wide range of device latency, bandwidth and characteristics
  3. 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 all three

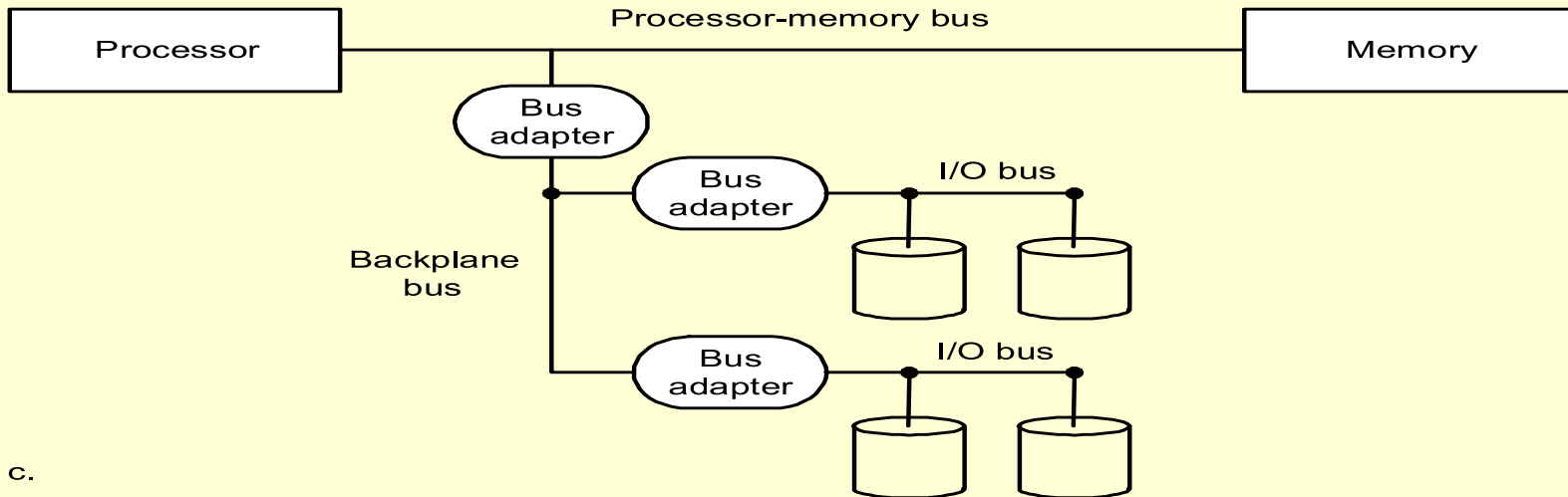
# Bus Configurations



a.



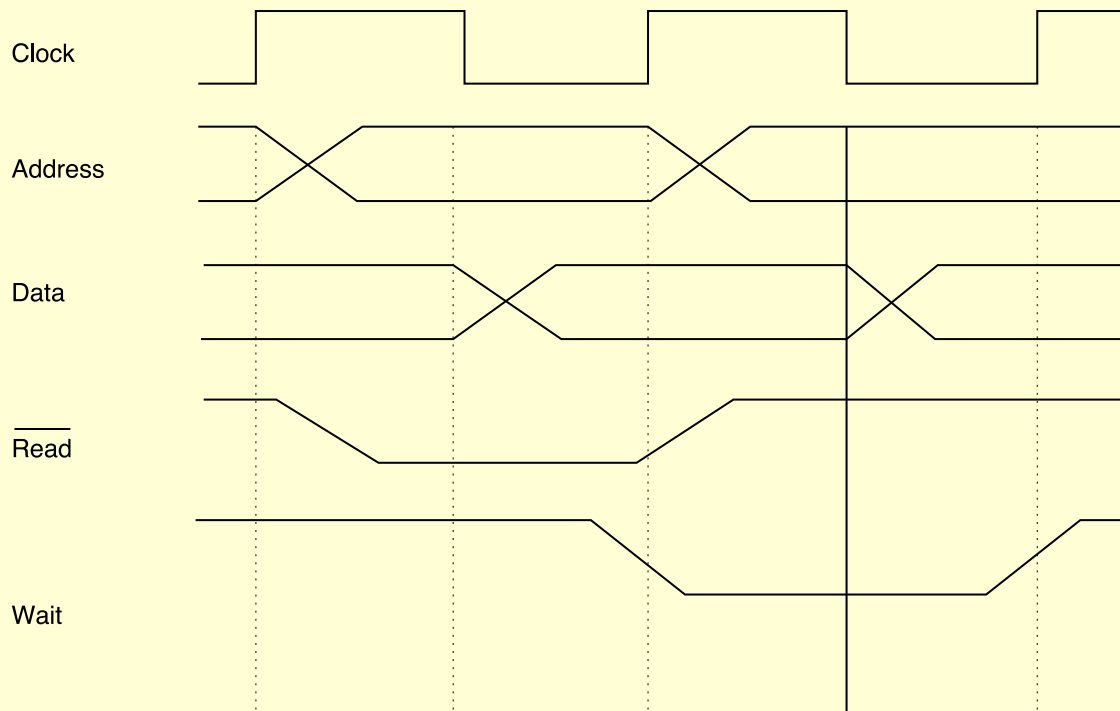
b.



c.

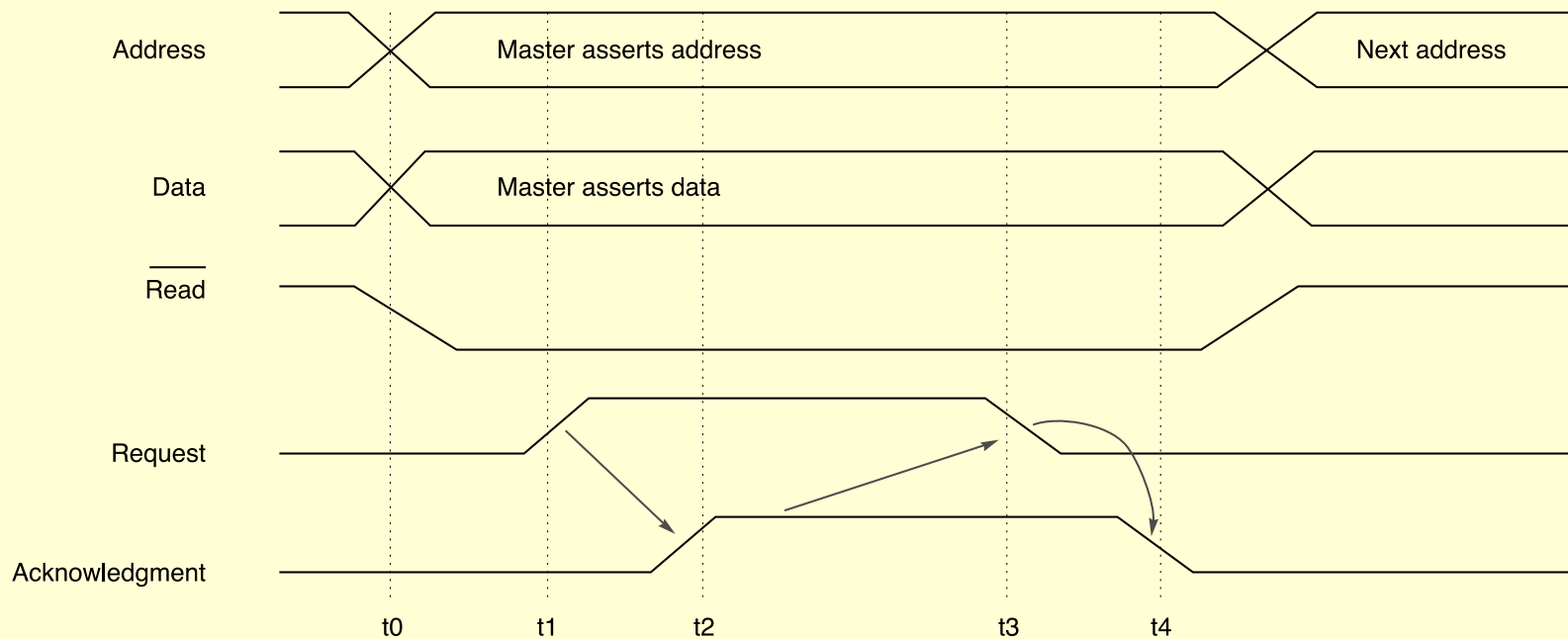
# Bus Control

- 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



# Bus Control

- 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



# Bus Performance 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:

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

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

The step for the asynchronous bus are:

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

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

# 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 (expensive!)
  - 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

# Bus Master

- Single master
  - Bus master (e.g. processor) controls all bus access
  - Bus slave (e.g. device or memory) only responds to requests
- Multiple master
  - Many devices can initiate bus transaction
  - Bus control logic & protocol resolves conflicts

# Bus Arbitration

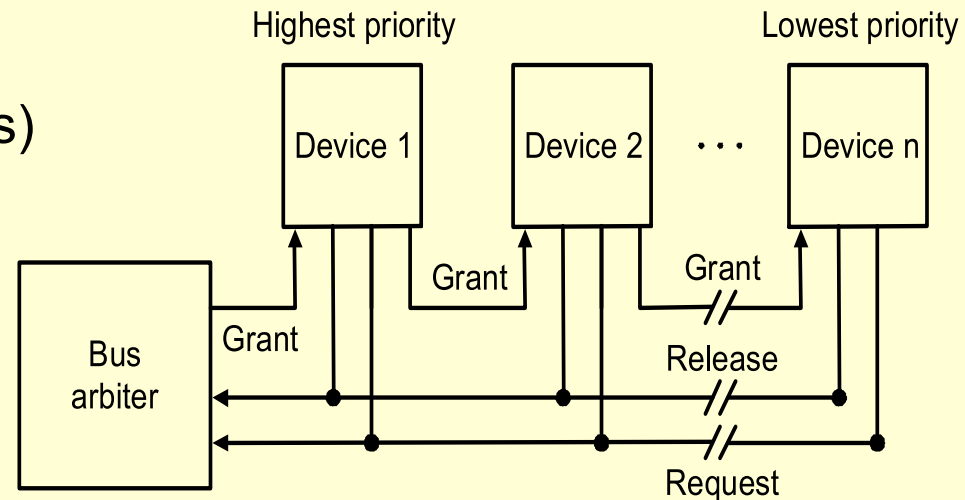
- Bus arbitration coordinates bus usage among multiple devices using request, grant, release mechanism
- Arbitration usually tries to balance two factors in choosing the granted device:
  - Devices with high bus-priority should be served first
  - Maintaining fairness to ensure that no device will be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

# 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
    - Ethernet: back off and try again later

# 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 might lock the bus)



# Arbitration Schemes (Cont.)

- 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

- 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
- Two 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 multiplexed?	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 (asynchronous)
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 Devices' Interface

- Special I/O instructions: (Intel 80X86, IBM 370)
  - Specify both the device number and the command word
    - Device number / address on I/O bus
    - Command word / data on I/O bus
    - Each devices maintain status register to indicate progress
  - Instructions are privileged to prevent user tasks from directly accessing the I/O devices
- Memory-mapped I/O: (Motorola/IBM PowerPC)
  - Portions of the address space are assigned to I/O devices
  - 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

# 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 programs
  - 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

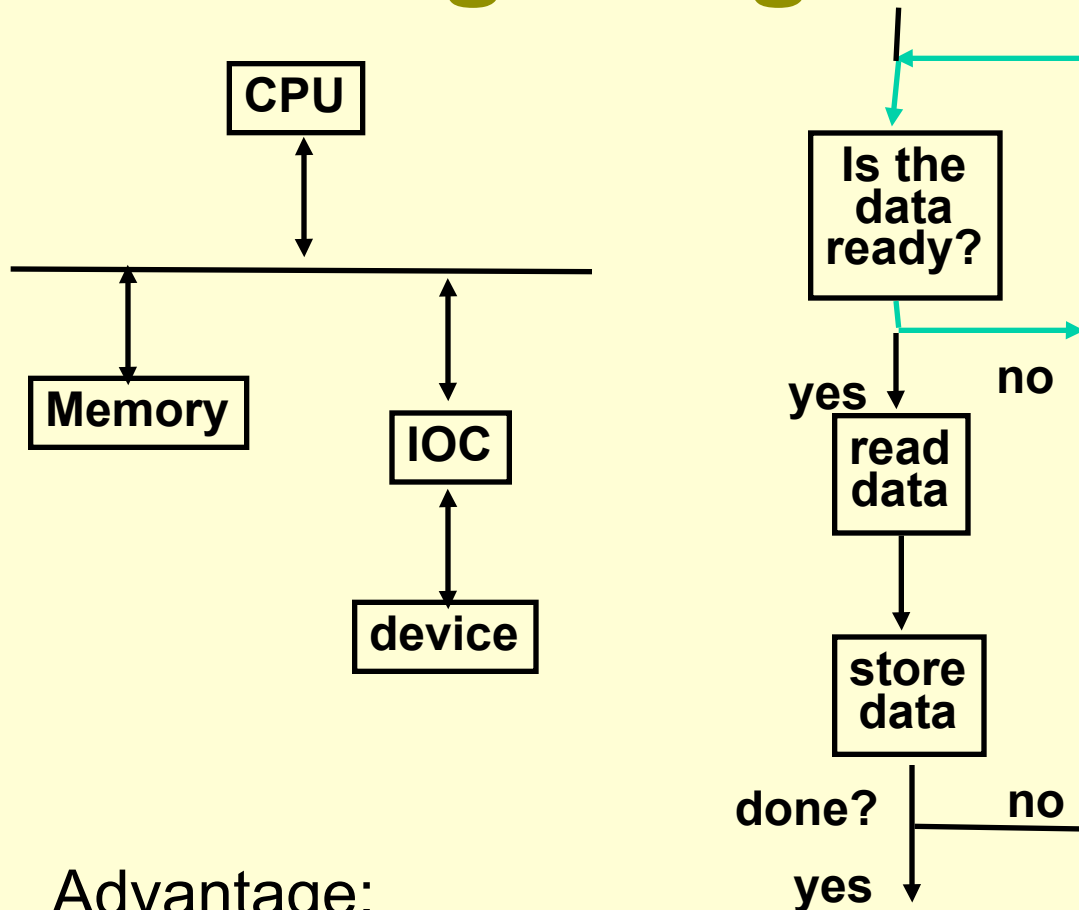
- Provide protection to shared I/O resources
  - Guarantees that a user's program cannot access other processes I/O
- Provides abstraction for accessing devices:
  - Supply routines that handle low-level device operation
  - Handles the interrupts generated by I/O devices
  - Provide equitable access 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

# 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 puts information in a status register
    - The OS periodically check the status register
  - I/O Interrupt:
    - External 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
    - 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

# Polling: Programmed I/O

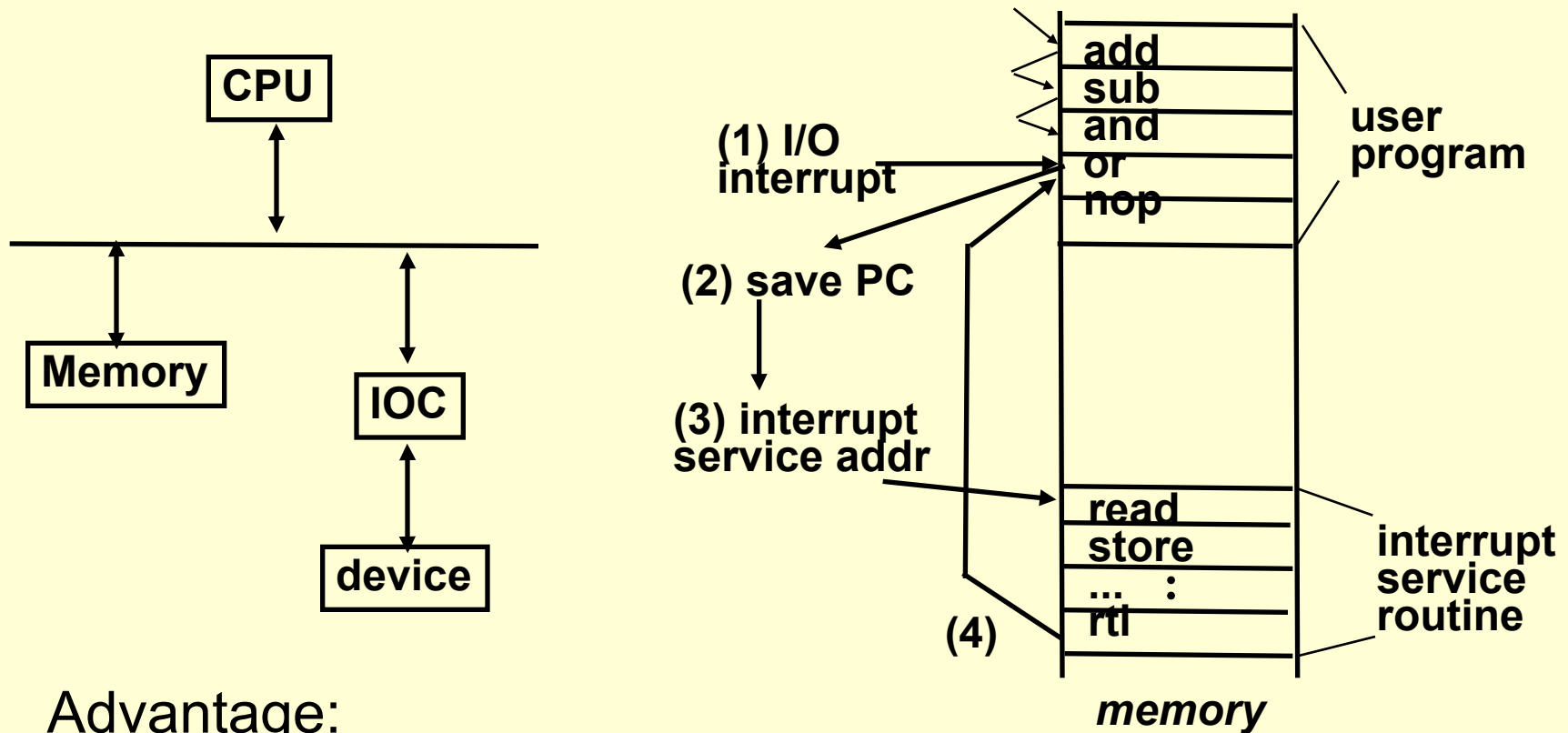


**busy wait loop  
not an efficient  
way to use the CPU  
unless the device  
is very fast!**

**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)

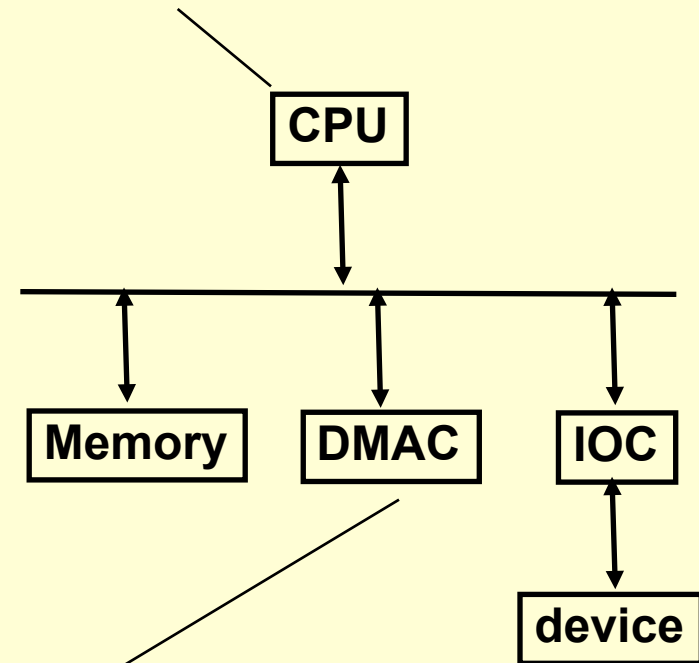
# I/O Interrupt vs. Exception

- 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
- I/O interrupt is asynchronous:
  - 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

# 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

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

# DMA Function

- CPU sets up and supply device id, memory address, number of bytes
- DMA controller (DMAC) starts the access and becomes bus master
- For multiple byte transfer, the DMAC increment the address
- DMAC interrupts the CPU upon completion

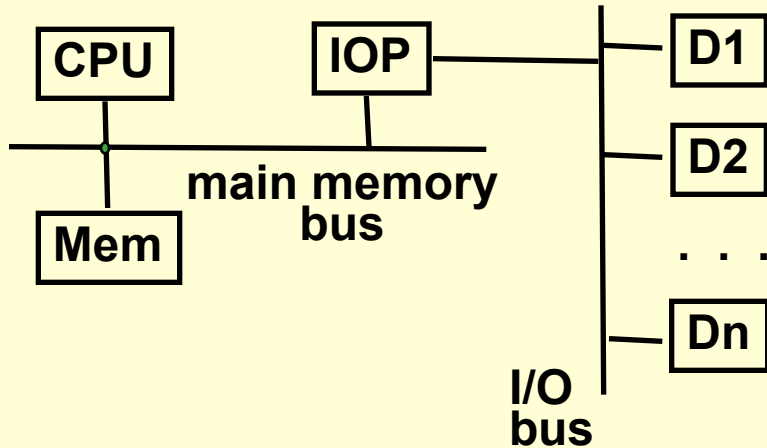
# DMA Problems: VM

- Pages have different 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

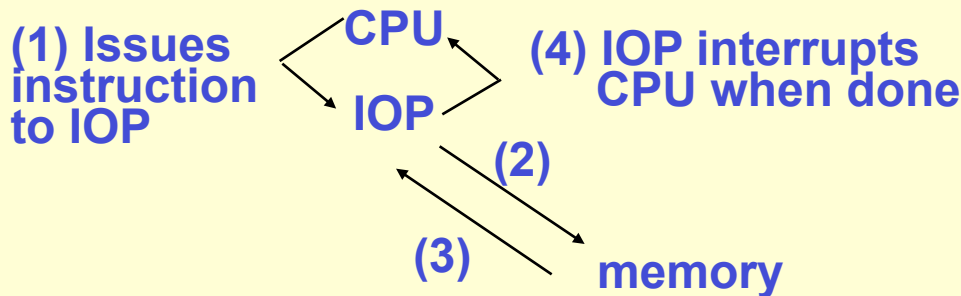
# DMA Problems: Cache

- 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

# I/O Processor

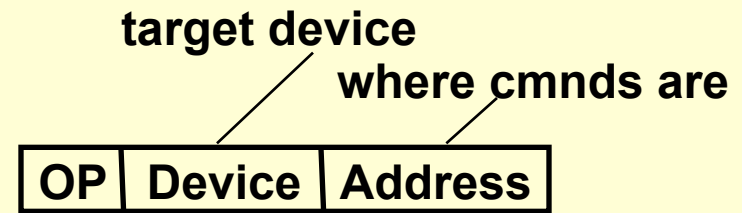


- An I/O processor (IOP) offload the CPU
- Motorola 860 includes IOP for serial communication

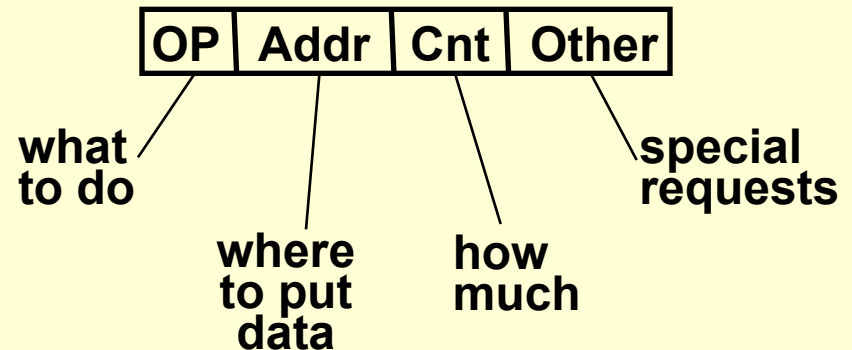


Device to/from memory transfers are controlled by the IOP directly.

IOP steals memory cycles.

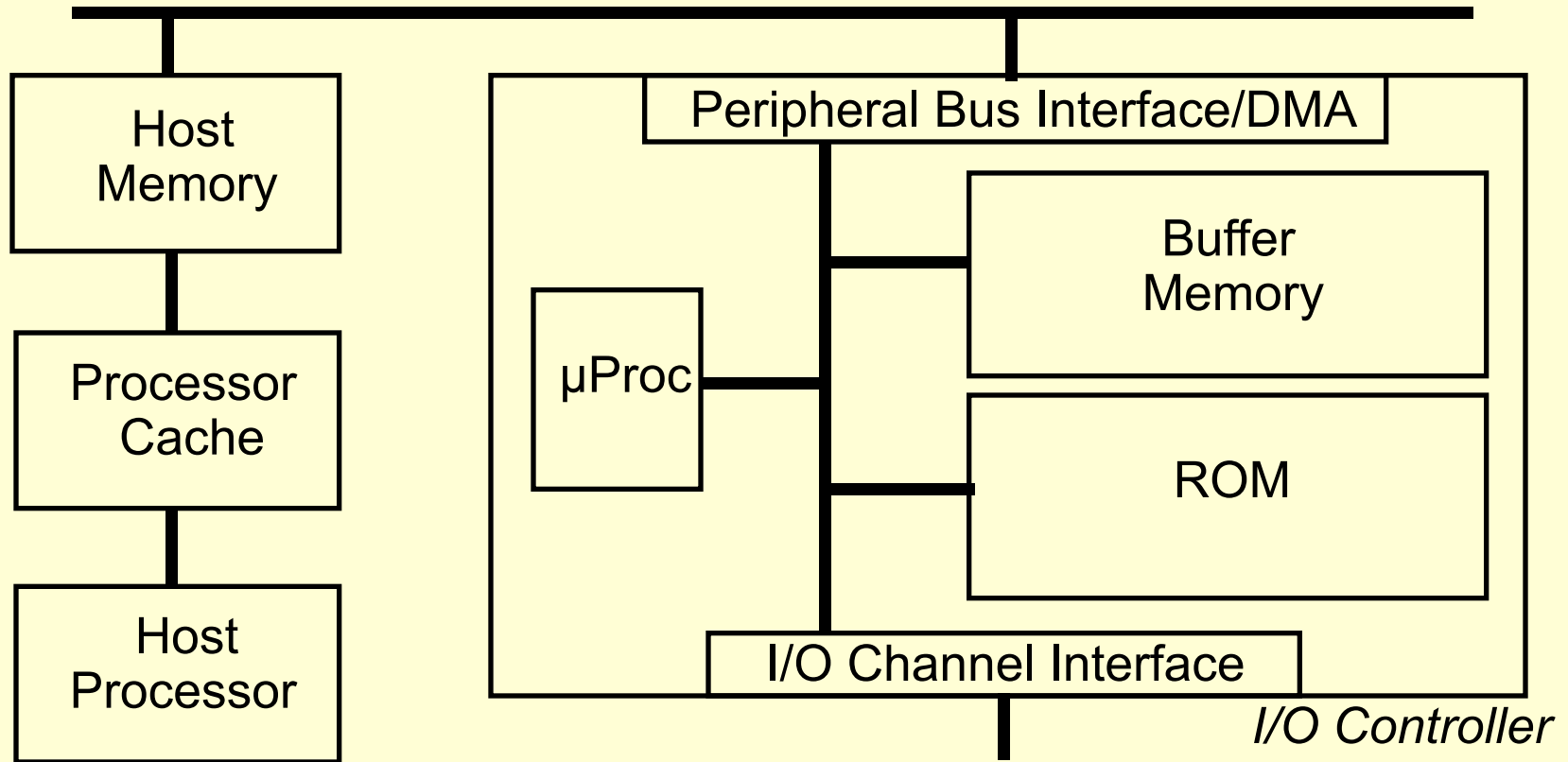


IOP looks in memory for commands



# I/O Controller Architecture

Peripheral Bus (VME, FutureBus, etc.)

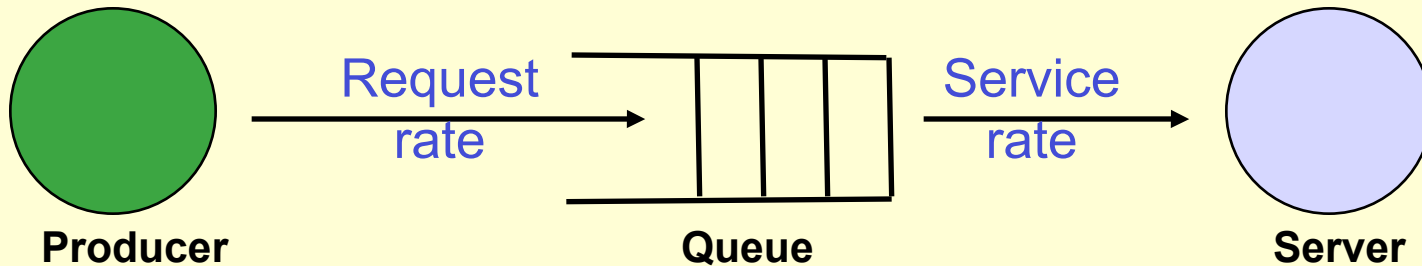


- Request/response block interface
- Backdoor access to host memory

# I/O System Performance

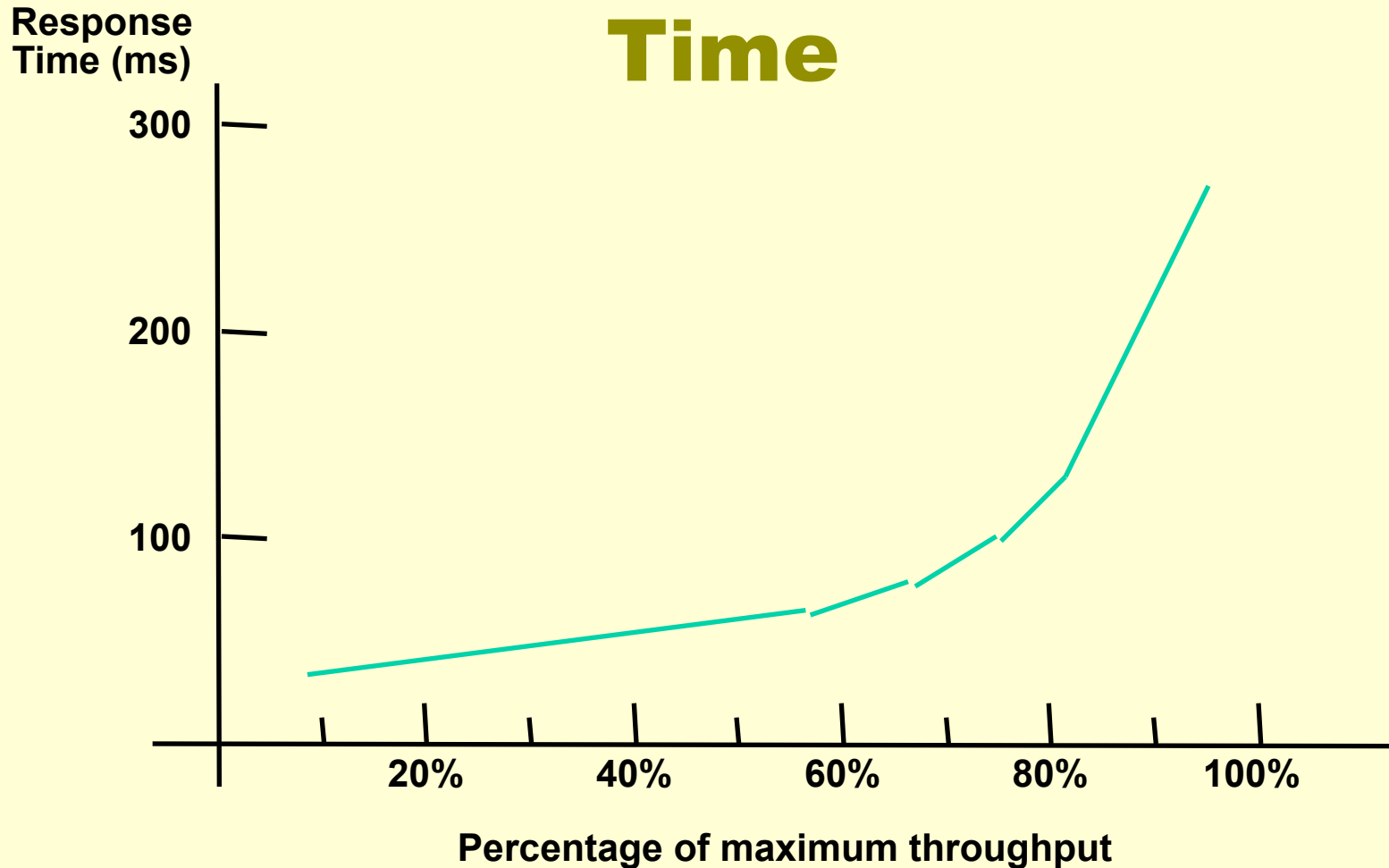
- I/O System performance depends on many aspects of the system (“limited by weakest link in the chain”):
  - The CPU
  - The memory system:
    - Internal and external caches
    - Main Memory
  - The underlying interconnection (buses)
  - The I/O controller
  - The I/O device
  - The speed of the I/O 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

# Simple Producer-Server Model



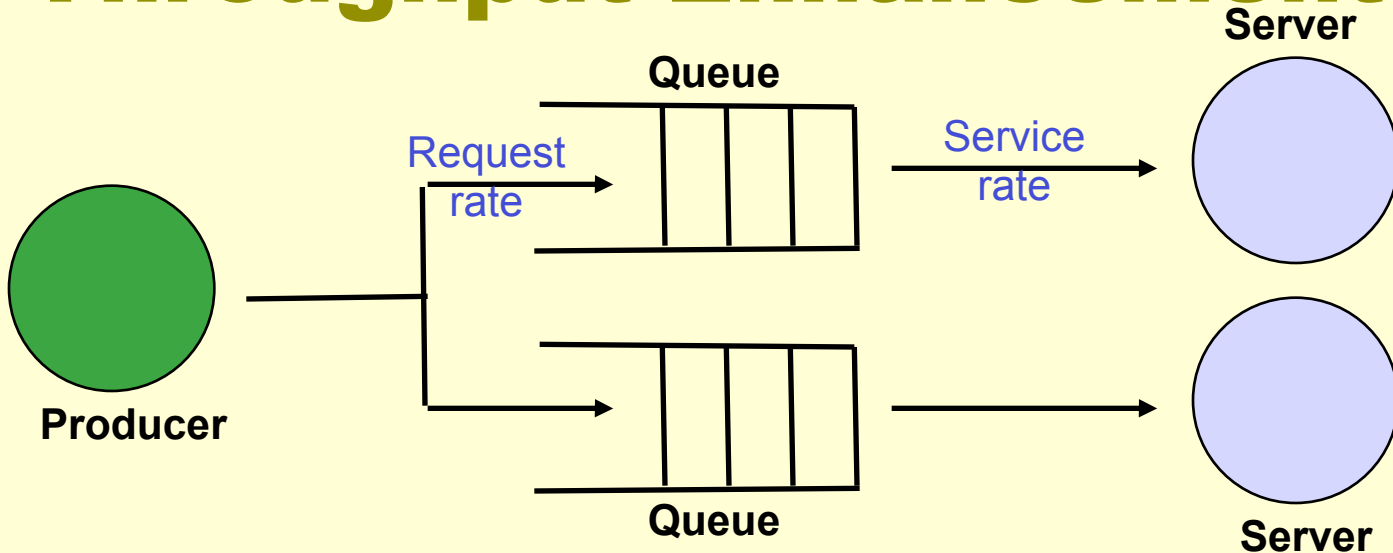
- Throughput:
  - The number of tasks completed by the server in unit time
  - In order to get the highest possible throughput:
    - The server should never be idle
    - The queue should never be empty
- Response time:
  - Begins when a task is placed in the queue
  - Ends when it is completed by the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

# Throughput versus Respond



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

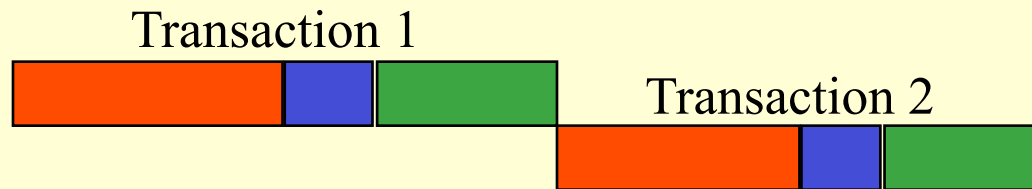
# Throughput Enhancement



- 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 =  $U = \text{Request Rate} / \text{Service Rate}$
  - Mean Queue Length =  $U / (1 - U)$
  - Average queuing time = Mean Queue Length / request rate

# 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

# 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
- 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

# Well Known Storage

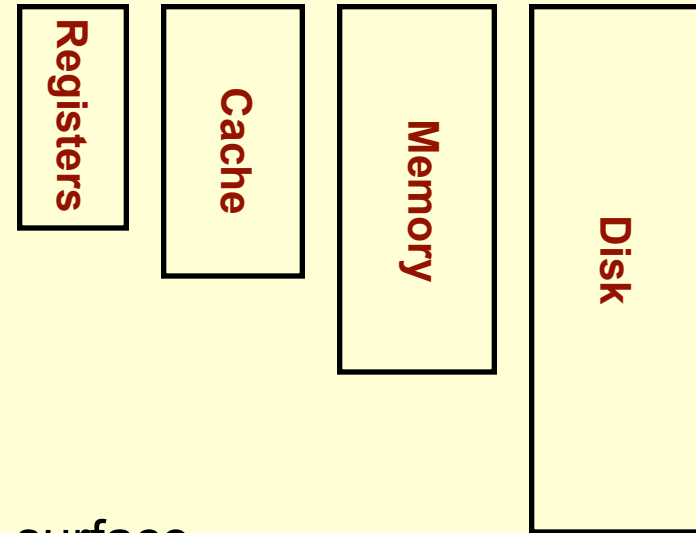
- Hard drive: random access, magnetic based, various density and speed
- Tape: sequential access, huge storage capacity, cheap and replaceable
- Helical scan tapes: diagonal storage of bits to allow high speed tape rotation
  - (used also for VCR and camcorders)
- Flash DRAM: little slower than DRAM, expensive and limited in capacity
- Optical disk: high density, read-only

# Storage Technology Drivers

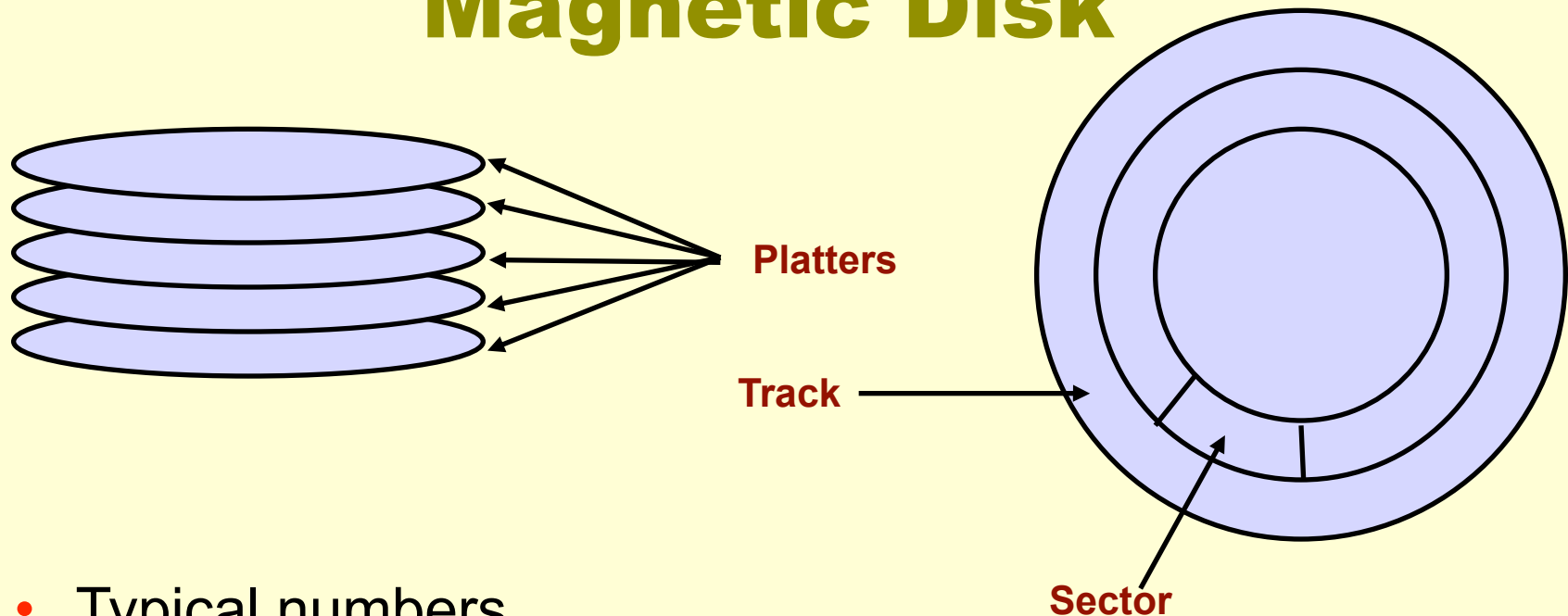
- Driven by computing paradigm:
  - 1950s: migration from batch to on-line processing
  - 1990s: migration to ubiquitous computing
    - Computers in phones, books, cars, video cameras, ...
    - Nationwide fiber optic network with wireless tails
- Effects on storage industry:
  - Embedded storage: smaller, cheaper, more reliable, lower power
  - Data utilities: high capacity, hierarchically managed storage

# Magnetic Disk

- Purpose:
  - Long term, nonvolatile storage
  - Large, inexpensive, and slow
  - Low level in the memory hierarchy
- Two major types:
  - Floppy disk, Hard disk
- Both types of disks:
  - 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



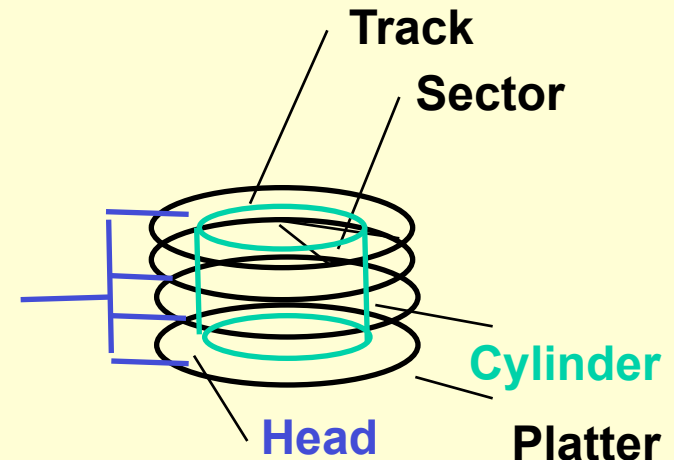
# Organization of a Hard Magnetic Disk



- Typical numbers
  - 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 to
- Traditionally all tracks have the same number of sectors:
  - Constant bit density: record more sectors on the outer tracks
  - Constant bit size, speed varies with track location

# Magnetic Disk Operation

- Cylinder
  - tracks under all heads
- Read/write in three-stages
  - Seek time
    - position the arm over proper cylinder
  - Rotational latency
    - wait for the sector to rotate under the head
  - Transfer time



# Magnetic Disk Characteristic

- Seek time
  - $\sum(\text{possible seek times})/(\# \text{ possible seeks})$
  - Typical average 4–12 ms
    - Locality may reduce by 25% to 33%
- Rotational Latency:
  - Most disks rotate at 3,600 to 7,200 RPM
    - = 16 ms to 8 ms per revolution
  - An average latency 1/2 way around disk
    - 8 ms at 3600 RPM, 4 ms at 7200 RPM

# Magnetic Disk Characteristic

- 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

# Example

Calculate the access time for a disk with 512 byte/sector and 12 ms advertised seek time. The disk rotates at 5400 RPM and transfers data at a rate of 4MB/sec. The controller overhead is 1 ms. Assume that the queue is idle (so no service time)

## Answer:

$$\begin{aligned}\text{Disk Access Time} &= \text{Seek time} + \text{Rotational Latency} + \text{Transfer time} \\ &\quad + \text{Controller Time} + \text{Queuing Delay} \\ &= 12 \text{ ms} + 0.5 / 5400 \text{ RPM} + 0.5 \text{ KB} / 4 \text{ MB/s} + 1 \text{ ms} + 0 \\ &= 12 \text{ ms} + 0.5 / 90 \text{ RPS} + 0.125 / 1024 \text{ s} + 1 \text{ ms} \\ &= 12 \text{ ms} + 5.5 \text{ ms} \quad + 0.1 \text{ ms} \quad + 1 \text{ ms} \\ &= 18.6 \text{ ms}\end{aligned}$$

If real seeks are 1/3 the advertised seeks, disk access time would be 10.6 ms, with rotation delay contributing 50% of the access time!

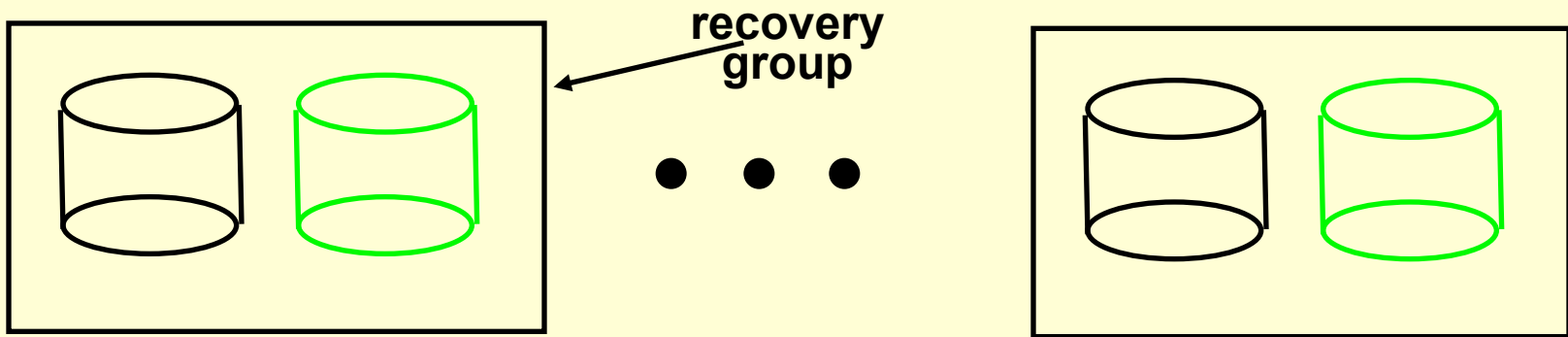
# Reliability and Availability

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

# Disk Arrays (RAID)

- 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:
  - Reliability of N disks = Reliability of 1 Disk  $\div$  N  
(50,000 Hours  $\div$  70 disks = 700 hours)
    - Disk system MTTF: Drops from 6 years to 1 month
  - Arrays (without redundancy) too unreliable to be useful!
  - But availability can be improved by adding redundant disks (RAID)

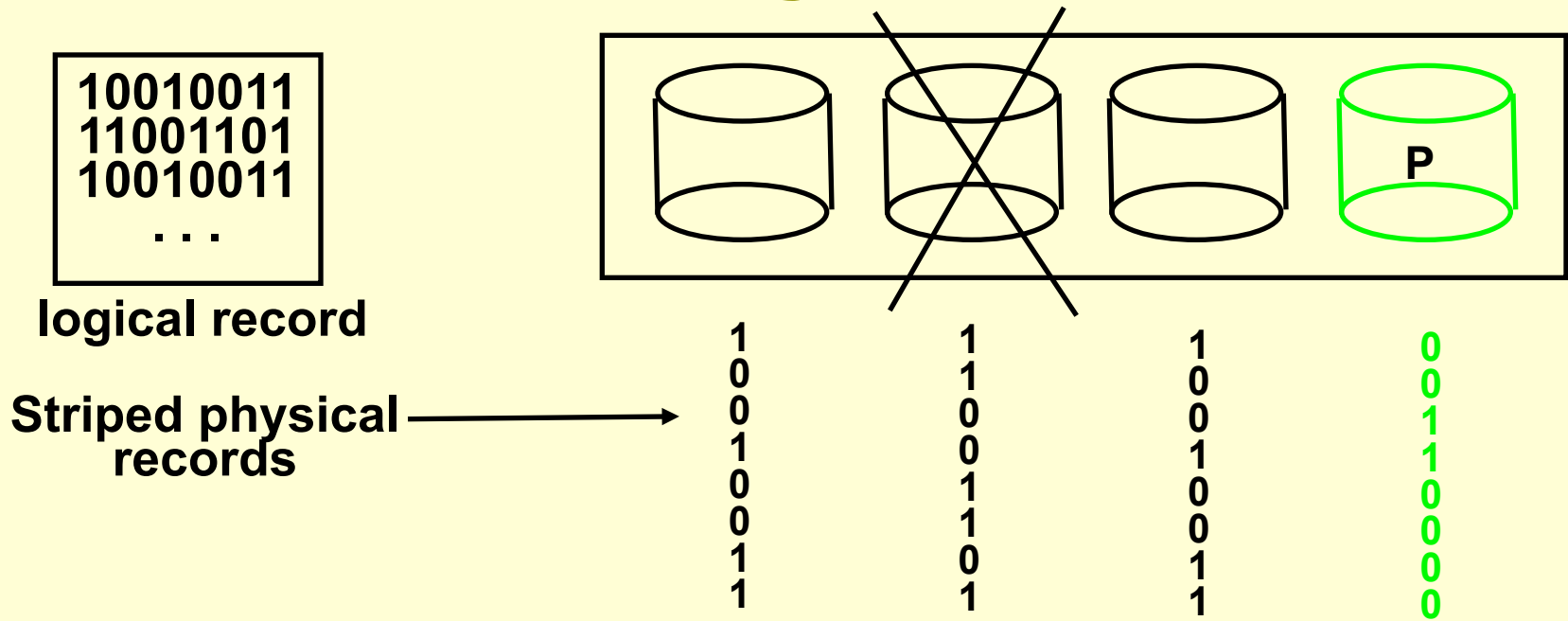
# Disk Mirroring/Shadowing



- 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***

# Parity Disk

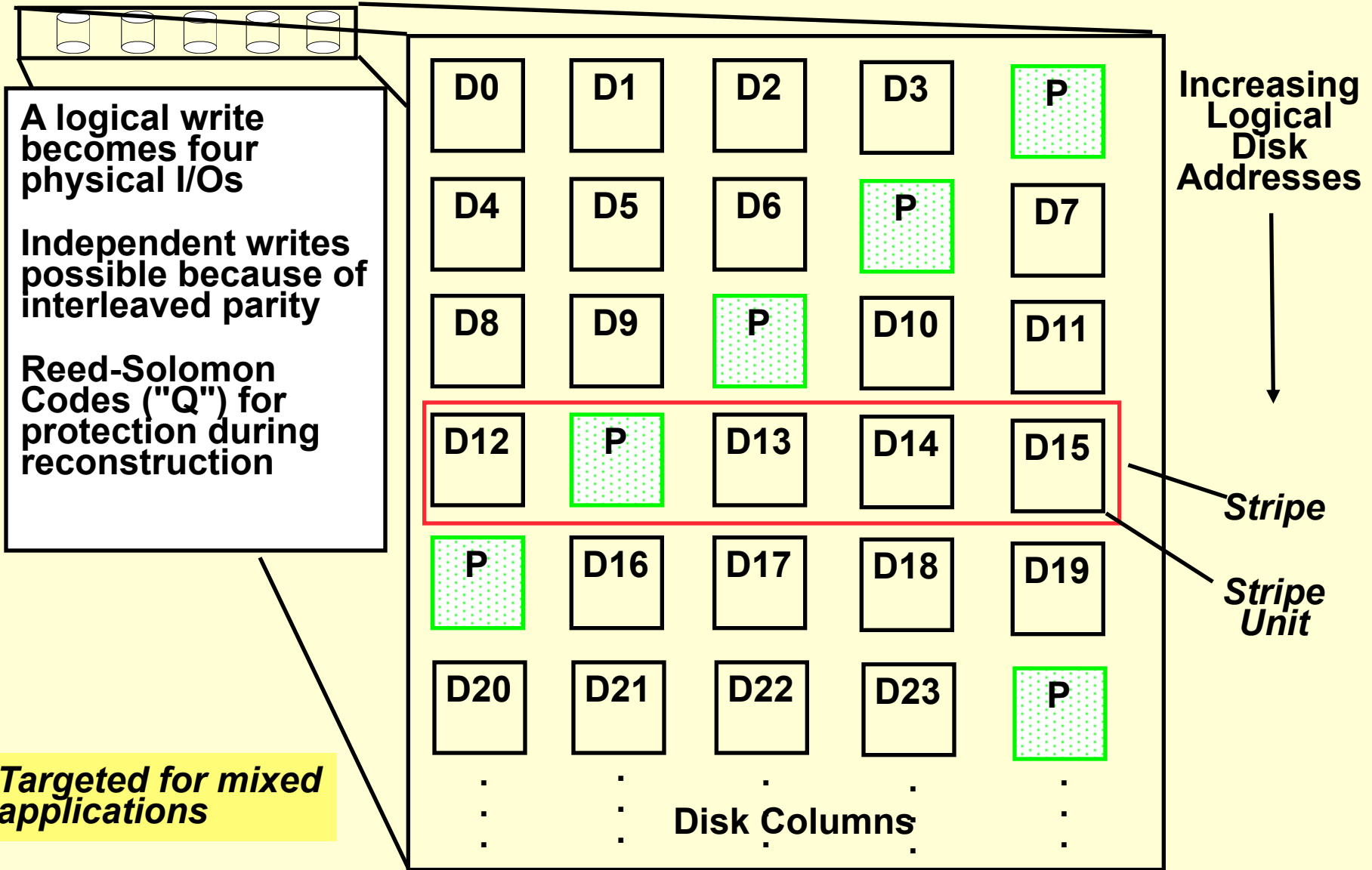


- Parity computed across recovery group to protect against hard disk failures
- 33% capacity cost for parity in this configuration: wider arrays reduce capacity costs, decrease expected availability, increase reconstruction time
- Arms logically synchronized, spindles rotationally synchronized (logically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

*Targeted for high bandwidth applications: Scientific, Image Processing*



# RAID 5+: High I/O Rate Parity

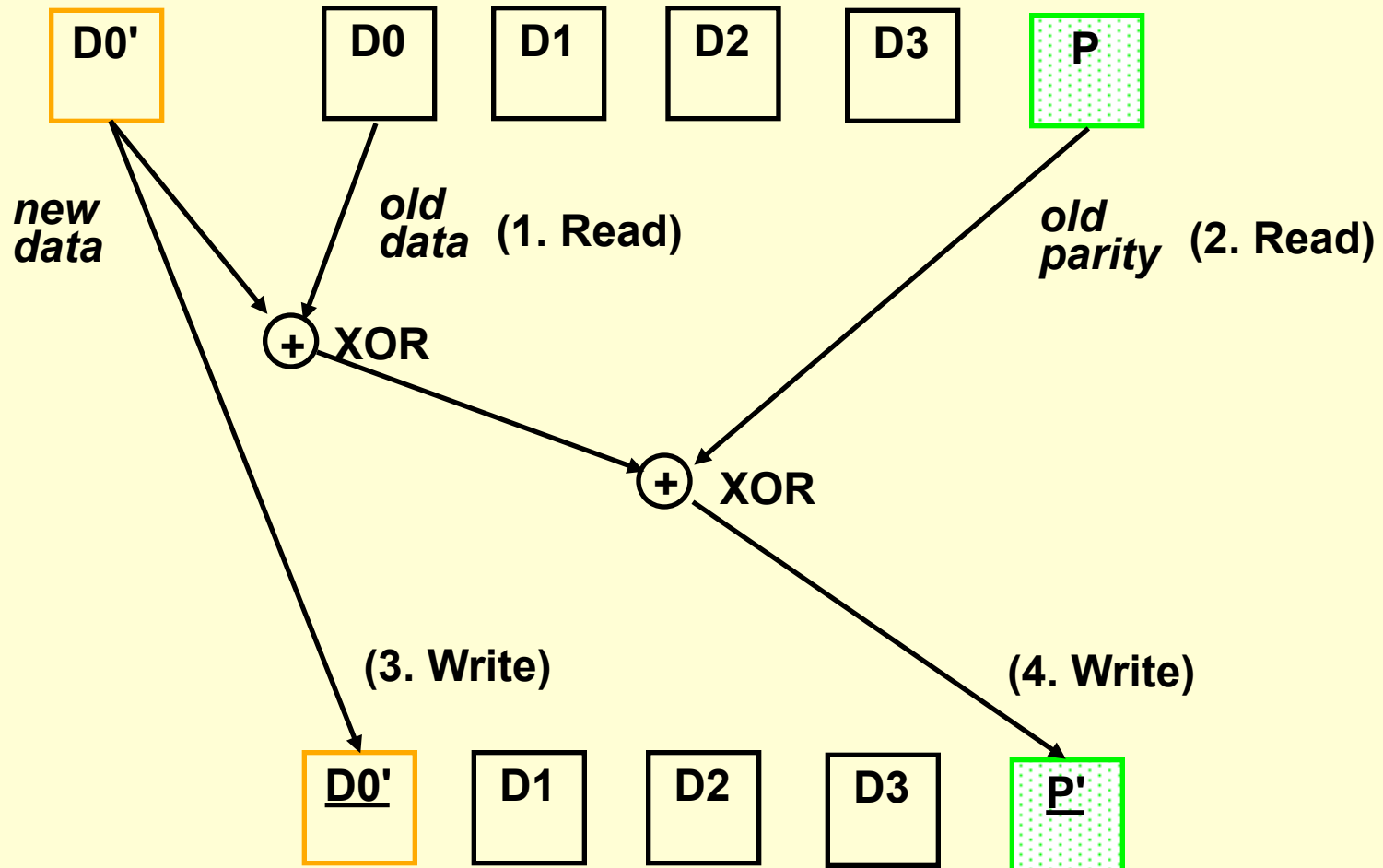


\* Slide is courtesy of Dave Patterson

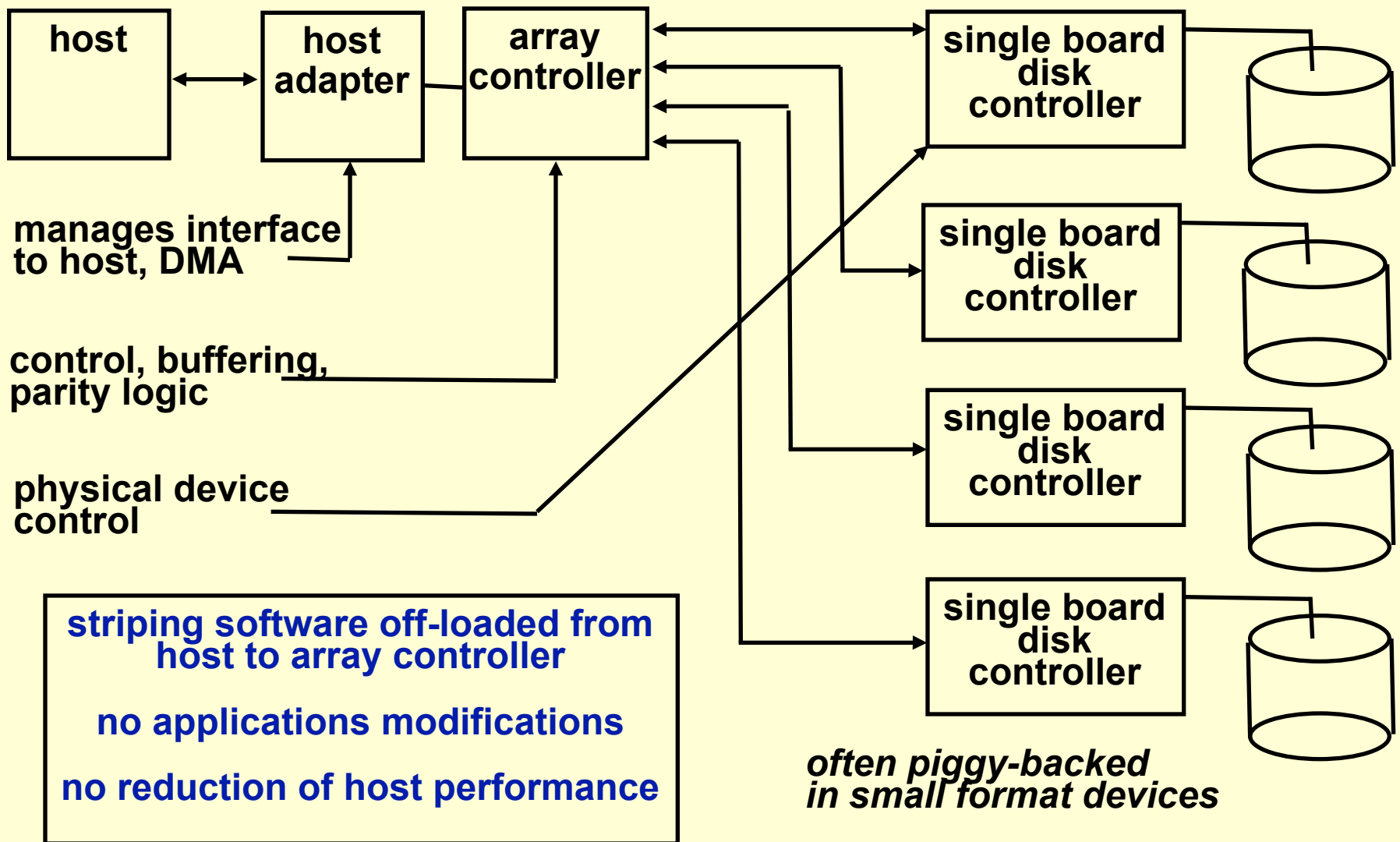
# Problems of Small Writes

## RAID-5: Small Write Algorithm

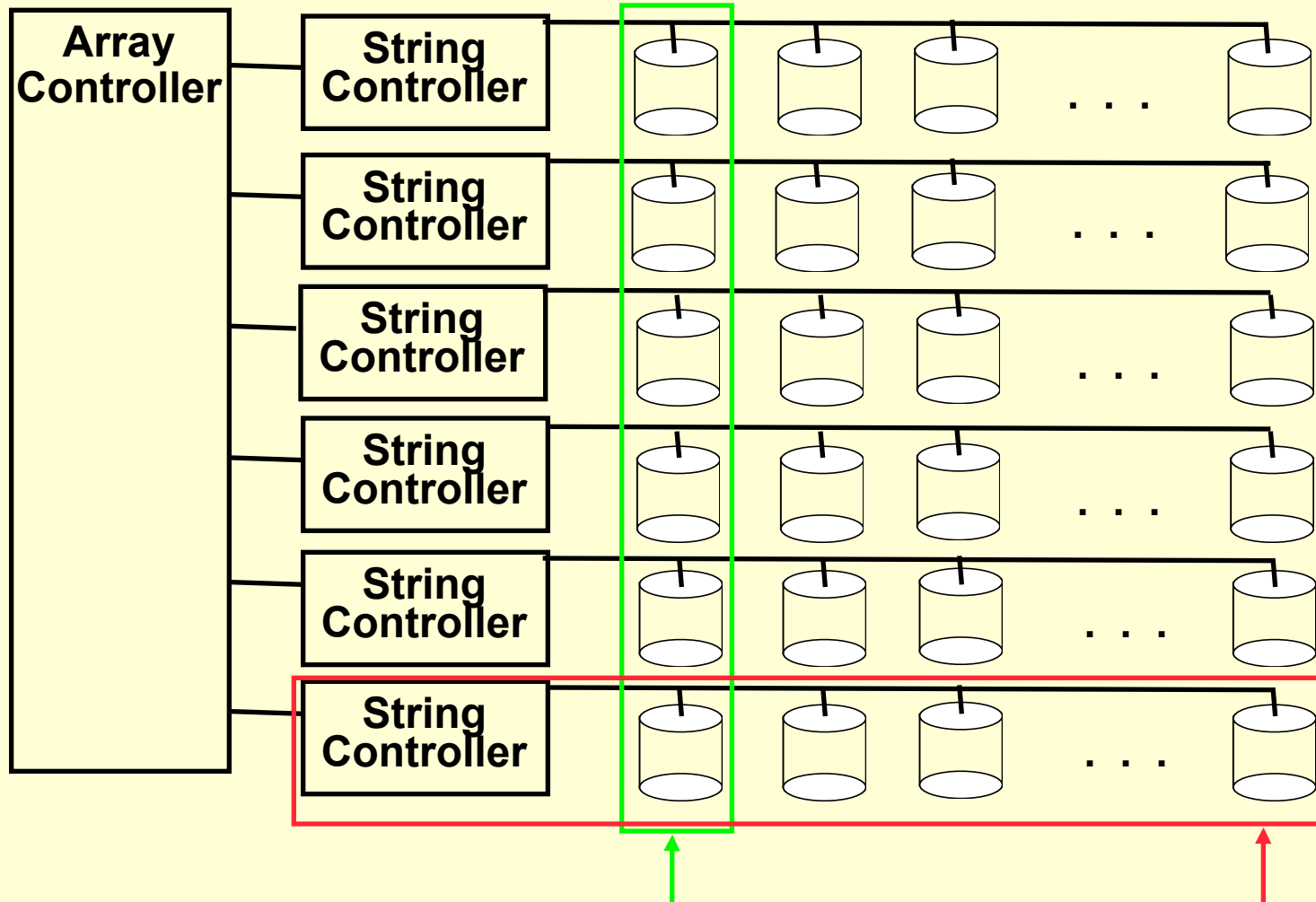
1 Logical Write = 2 Physical Reads + 2 Physical Writes



# Subsystem Organization



# 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

\* Slide is courtesy of Dave Patterson

# System-Level Availability

