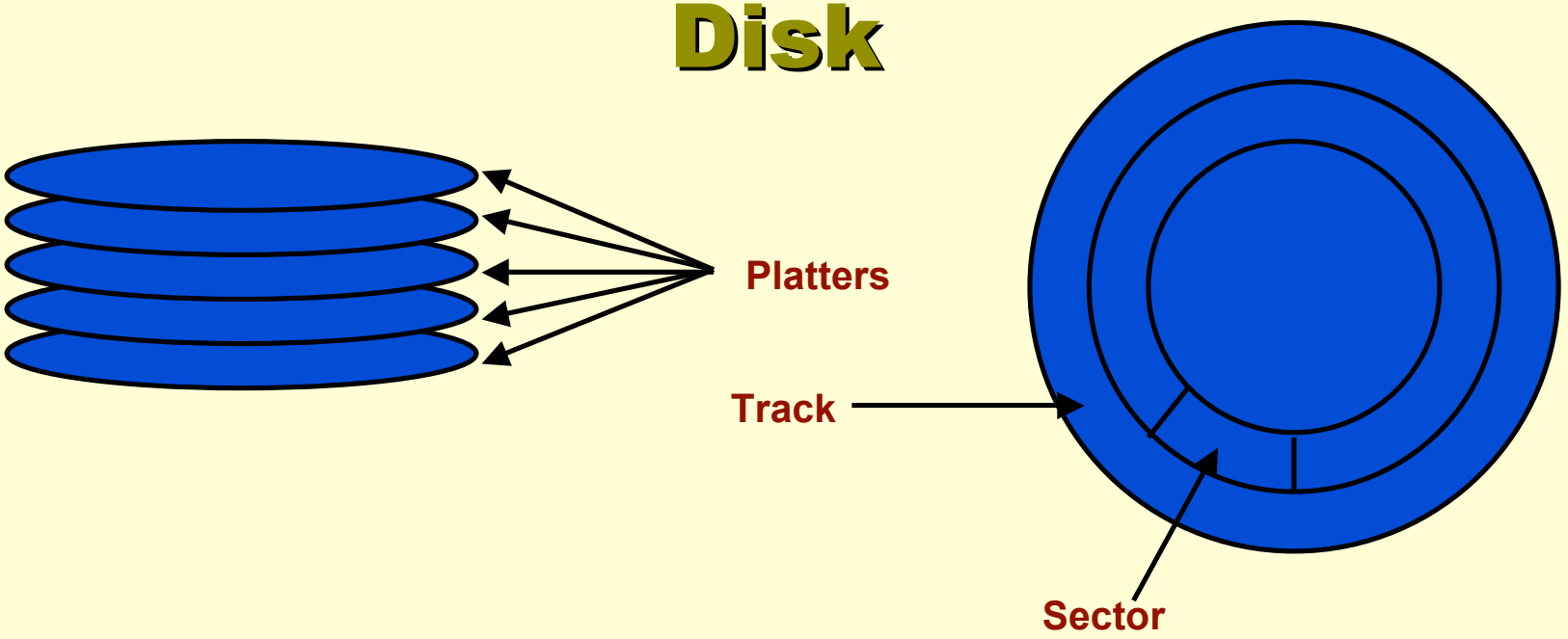
CMSC 611: Advanced Computer Architecture

Storage / I/O

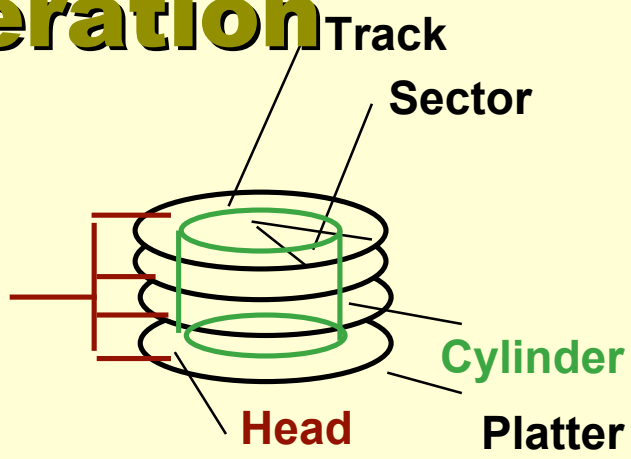
Most slides adapted from David Patterson. Some from Mohomed Younis

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 to
- 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 Operation



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

Magnetic Disk Characteristic

- Rotational Latency:
  - Most disks rotate at 3,600 to 7,200 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”
  - 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:

Disk Access Time = Seek time + Rotational Latency + Transfer time + Controller Time + Queuing Delay

= 12 ms + 0.5 / 5400 RPM + 0.5 KB / 4 MB/s + 1 ms + 0

= 12 ms + 0.5 / 90 RPS + 0.125 / 1024 s + 1 ms + 0

= 12 ms + 5.5 ms + 0.1 ms + 1 ms + 0 ms

= 18.6 ms

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!

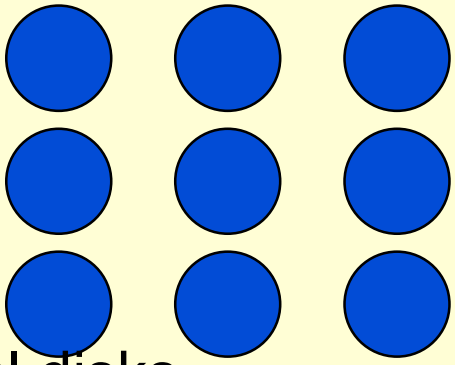
Historical Trend

Characteristics	IBM 3090	IBM UltraStar	Integral 1820
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

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:
  - Enhancing environmental conditions
  - Building more reliable components
  - Building with fewer components
    - Improve availability may come at the cost of lower reliability

Disk Arrays

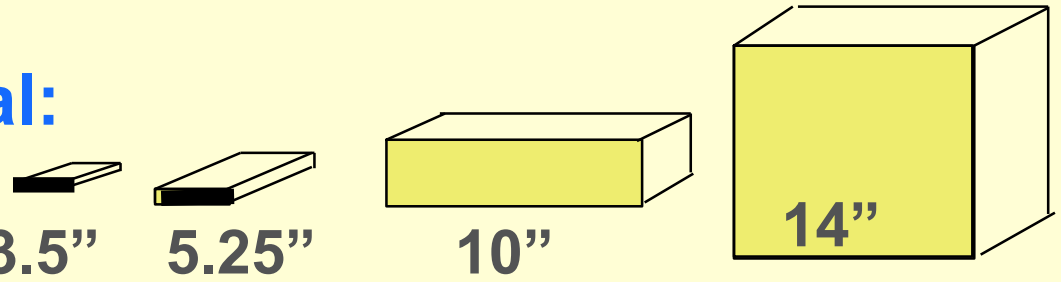


- Increase potential throughput by having many disk drives:
  - Data is spread over multiple disk
  - Multiple accesses are made to several disks
- Reliability is lower than a single disk:
  - Reliability of N disks = Reliability of 1 Disk ÷ N
    - (50,000 Hours ÷ 70 disks = 700 hours)
    - Disk system MTTF: Drops from 6 years to 1 month
  - Arrays (without redundancy) too unreliable to be useful!
  - But availability can be improved by adding redundant disks (RAID):
    - Lost information can be reconstructed from redundant information

Manufacturing Advantages of Disk Arrays

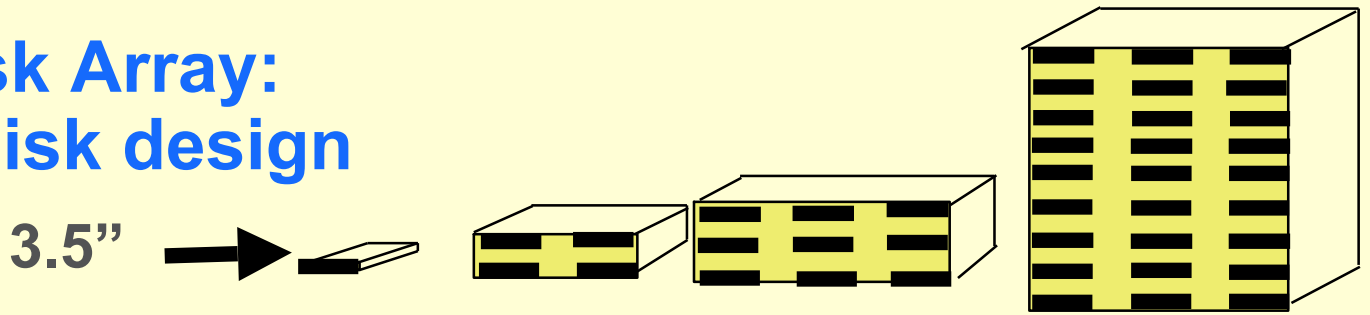
Disk Product Families

Conventional: 4 disk designs



Low End → **High End**

Disk Array: 1 disk design



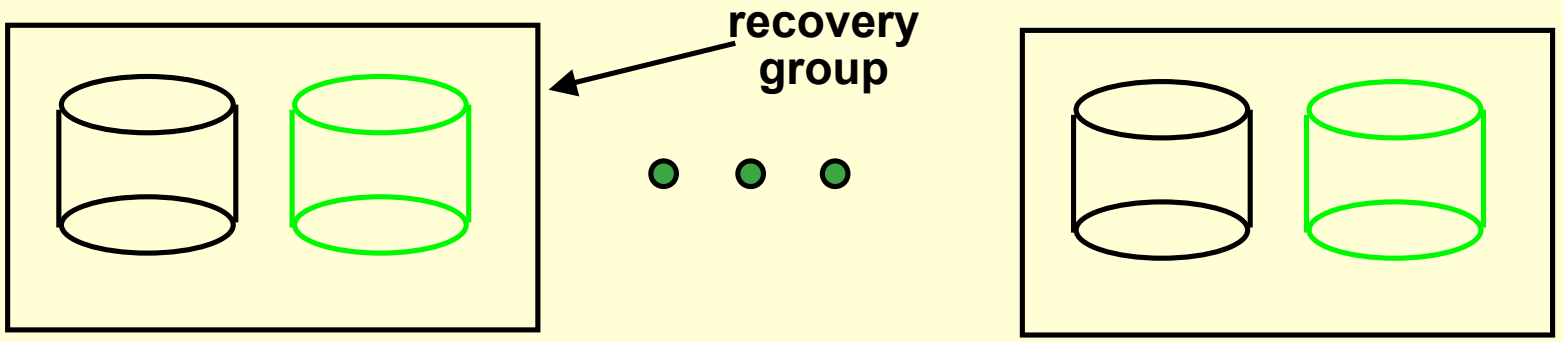
Replace Small # of Large Disks with Large # of Small Disks!

Redundant Arrays of Disks

- Redundant Array of Inexpensive Disks (RIAD)
  - Widely available and used in today's market
  - Files are "striped" across multiple spindles
  - Redundancy yields high data availability despite low reliability
  - Contents of a failed disk is reconstructed from data redundantly stored in the disk array
  - Drawbacks include capacity penalty to store redundant data and bandwidth penalty to update a disk block
  - Different levels based on replication level and recovery techniques

RAID level	Failures survived	Data disks	Check disks
0 Non-redundant	0	8	0
1 Mirrored	1	8	8
2 Memory-style ECC	1	8	4
3 Bit-interleaved parity	1	8	1
4 Block-interleaved	1	8	1
5 Block-interleaved distributed parity	1	8	1

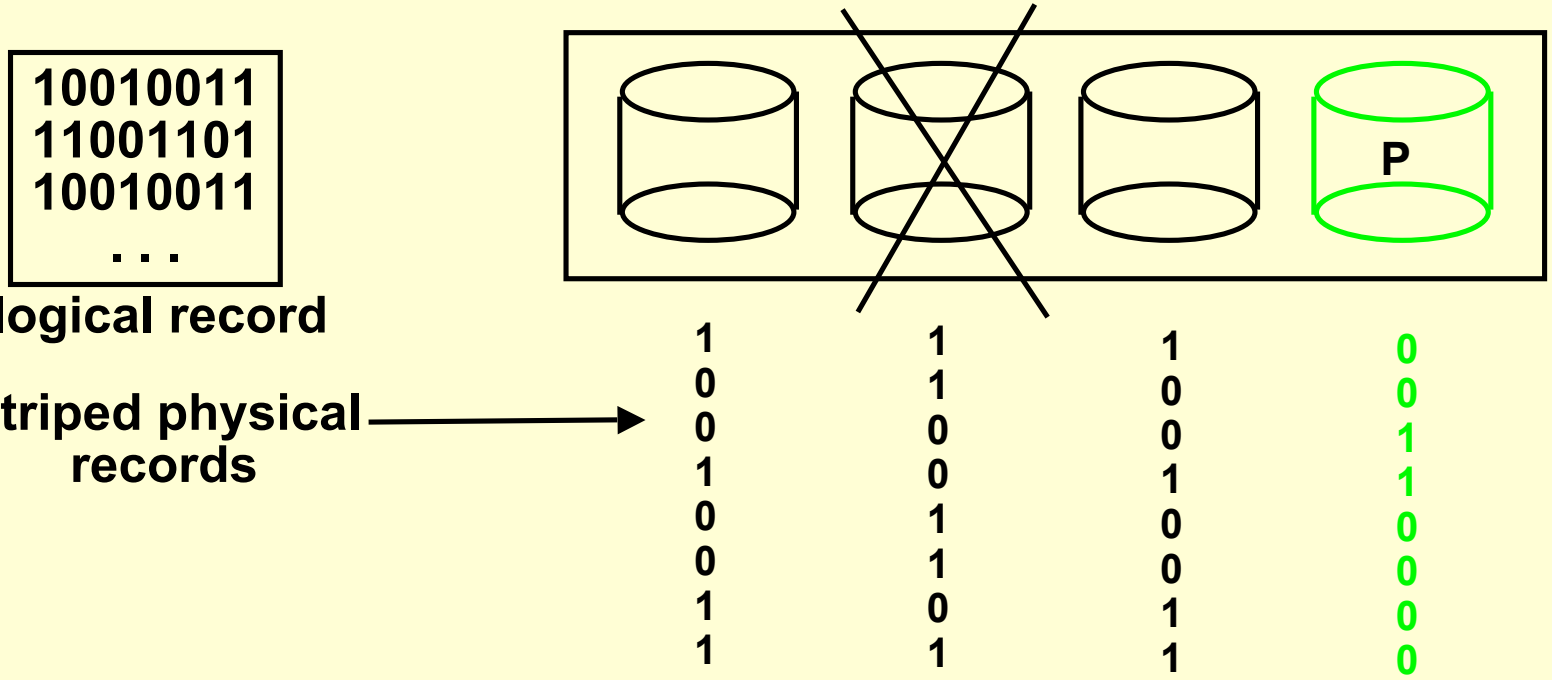
RAID 1: Disk Mirroring/Shadowing



- 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

RAID 3: Parity Disk

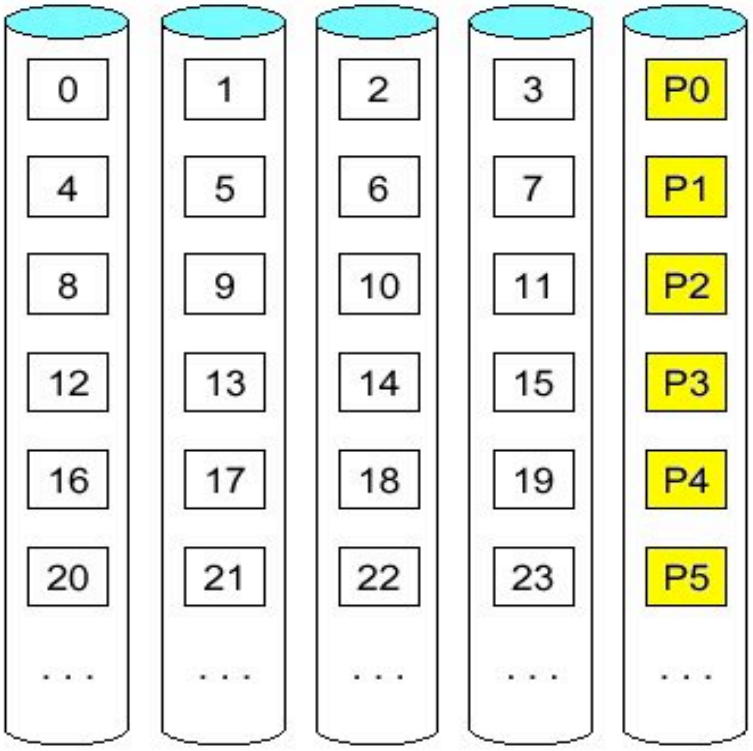


- Parity computed across recovery group to protect against hard disk failures
- 33% capacity cost for parity in this configuration: wider arrays reduce capacity costs, decrease expected availability, increase reconstruction time
- Arms logically synchronized, spindles rotationally synchronized (logically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

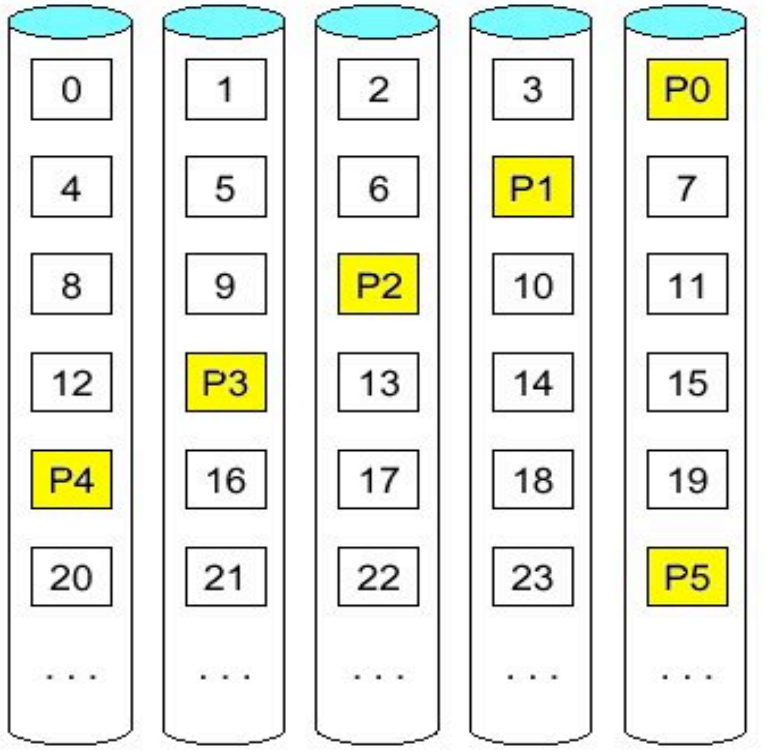
Targeted for high bandwidth applications: Scientific, Image Processing

Block-Based Parity

- Block-based party leads to more efficient read access compared to RAID 3
- Designating a party disk allows recovery but will keep it idle in the absence of a disk failure
- RAID 5 distribute the party block to allow the use of all disk and enhance parallelism of disk access

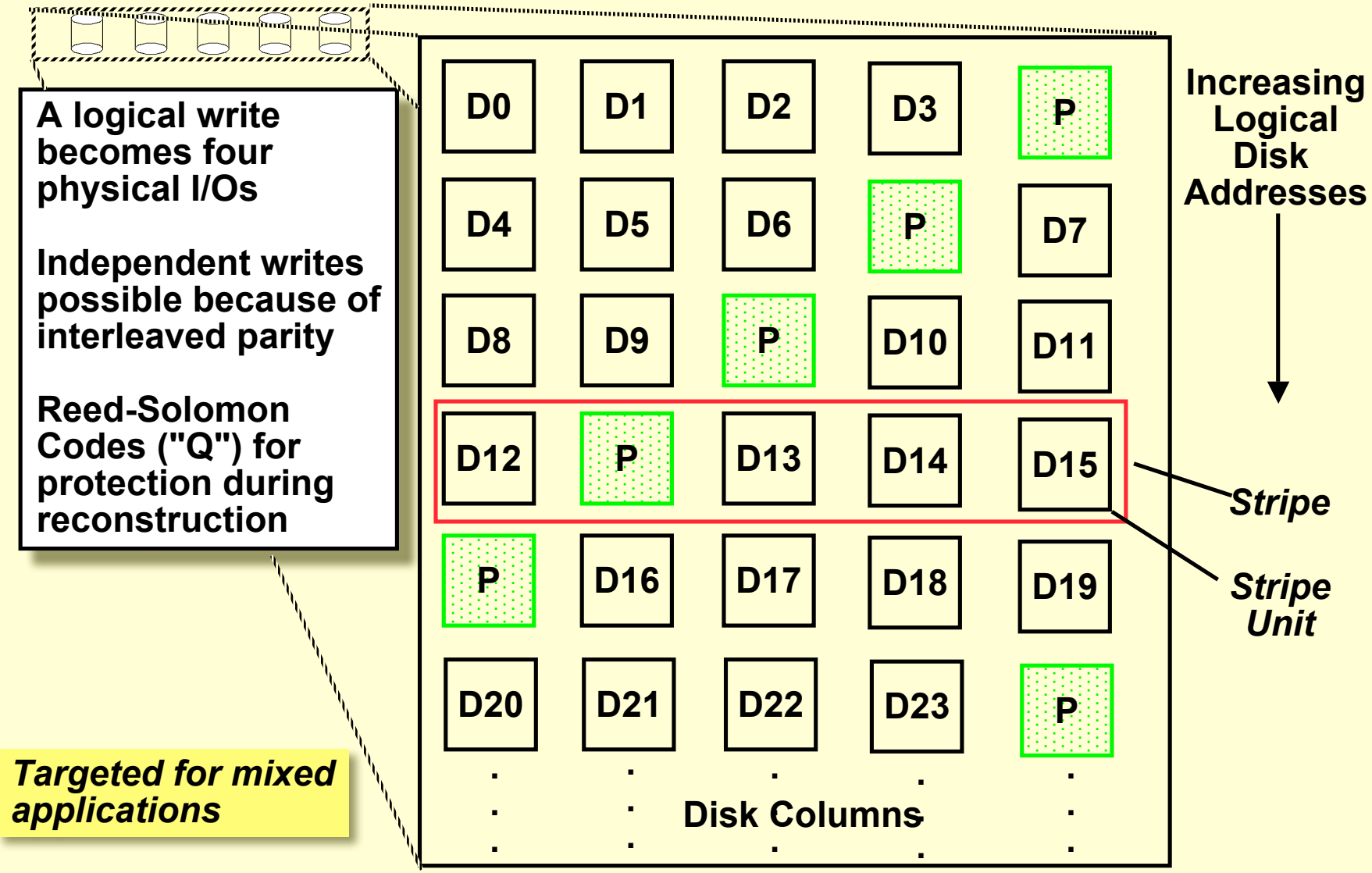


RAID 4



RAID 5

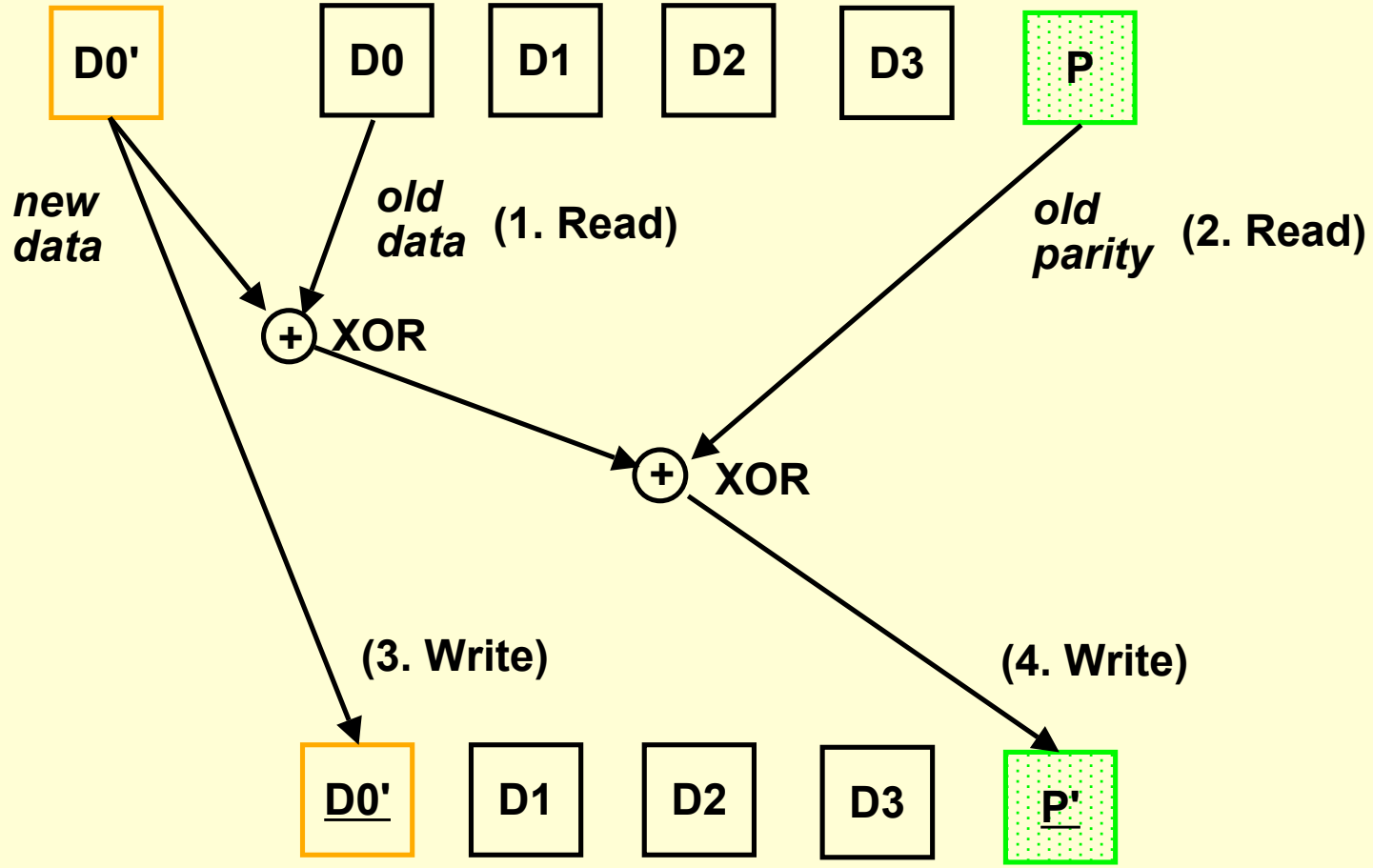
RAID 5+: High I/O Rate Parity



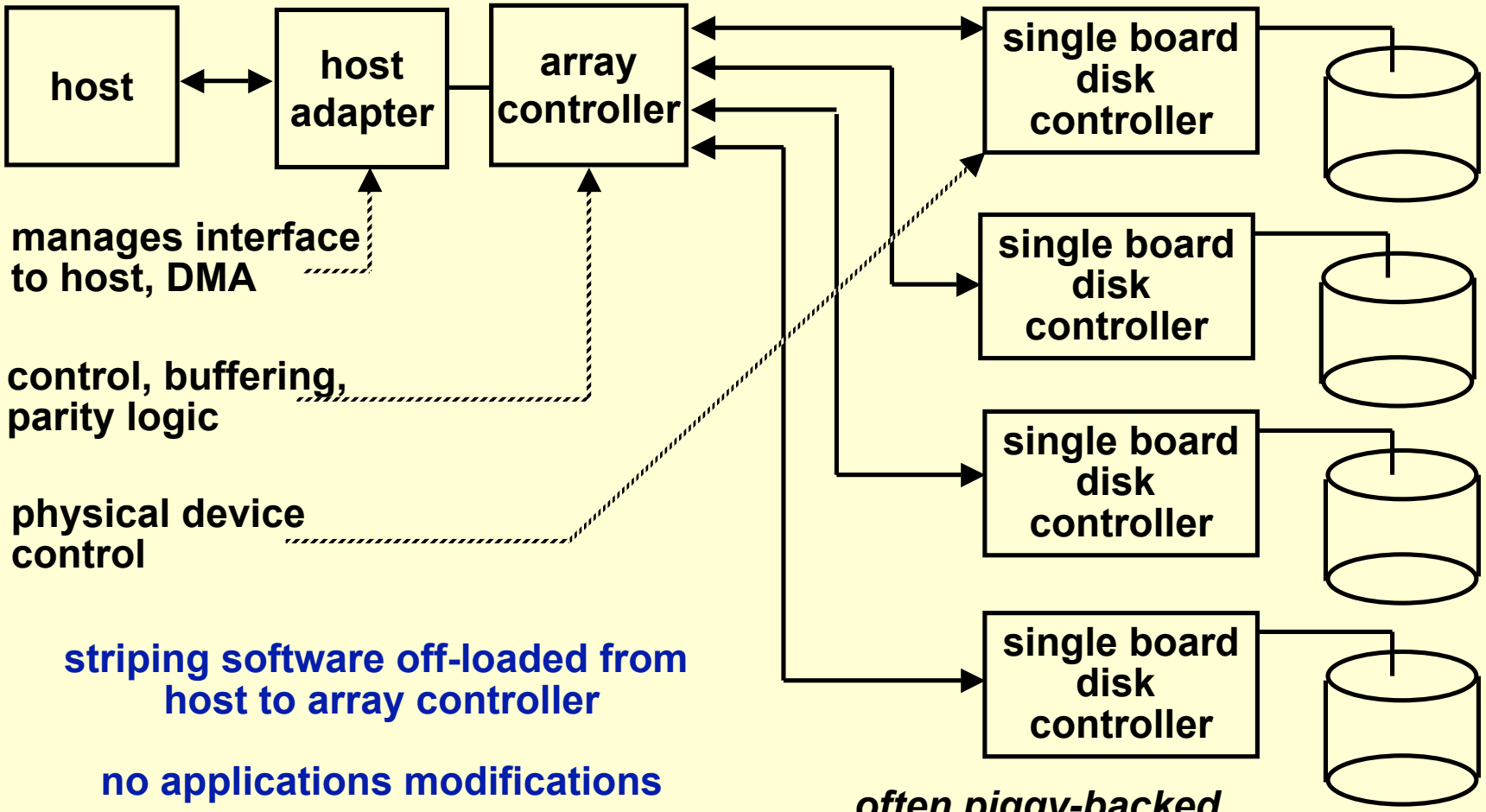
Problems of Small Writes

RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



Subsystem Organization



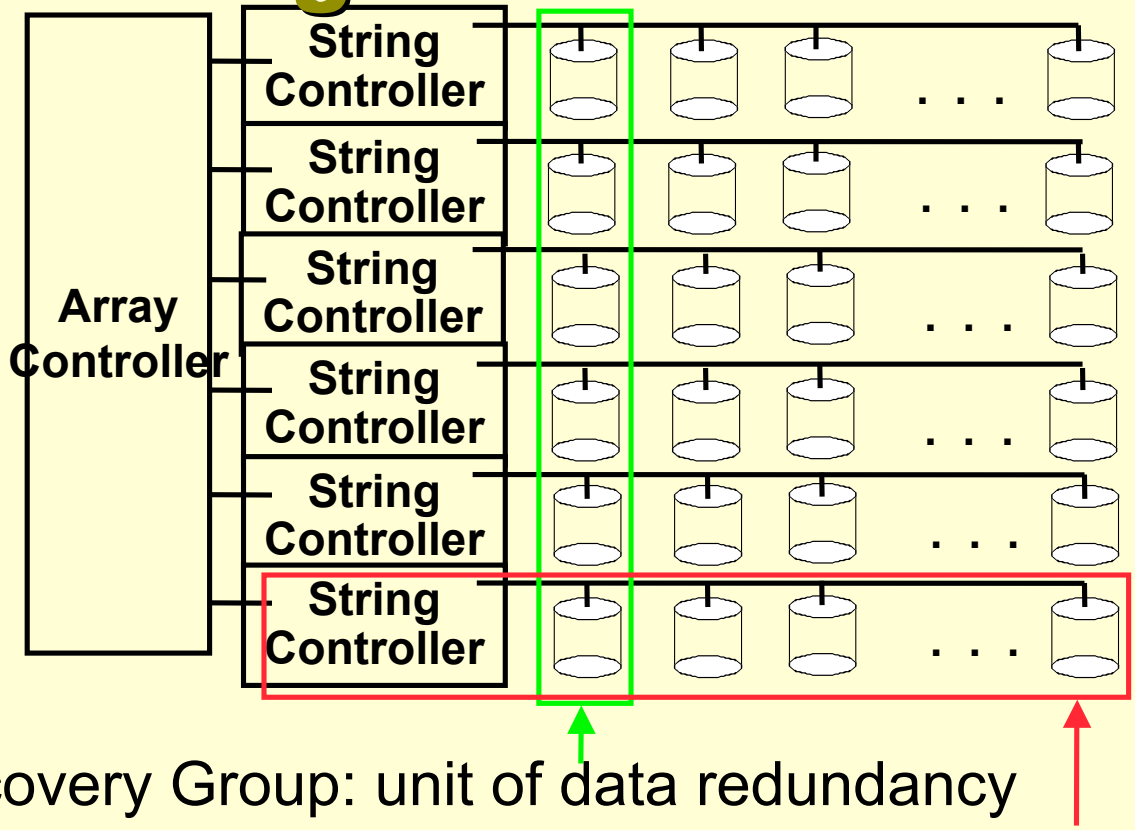
striping software off-loaded from host to array controller

no applications modifications

no reduction of host performance

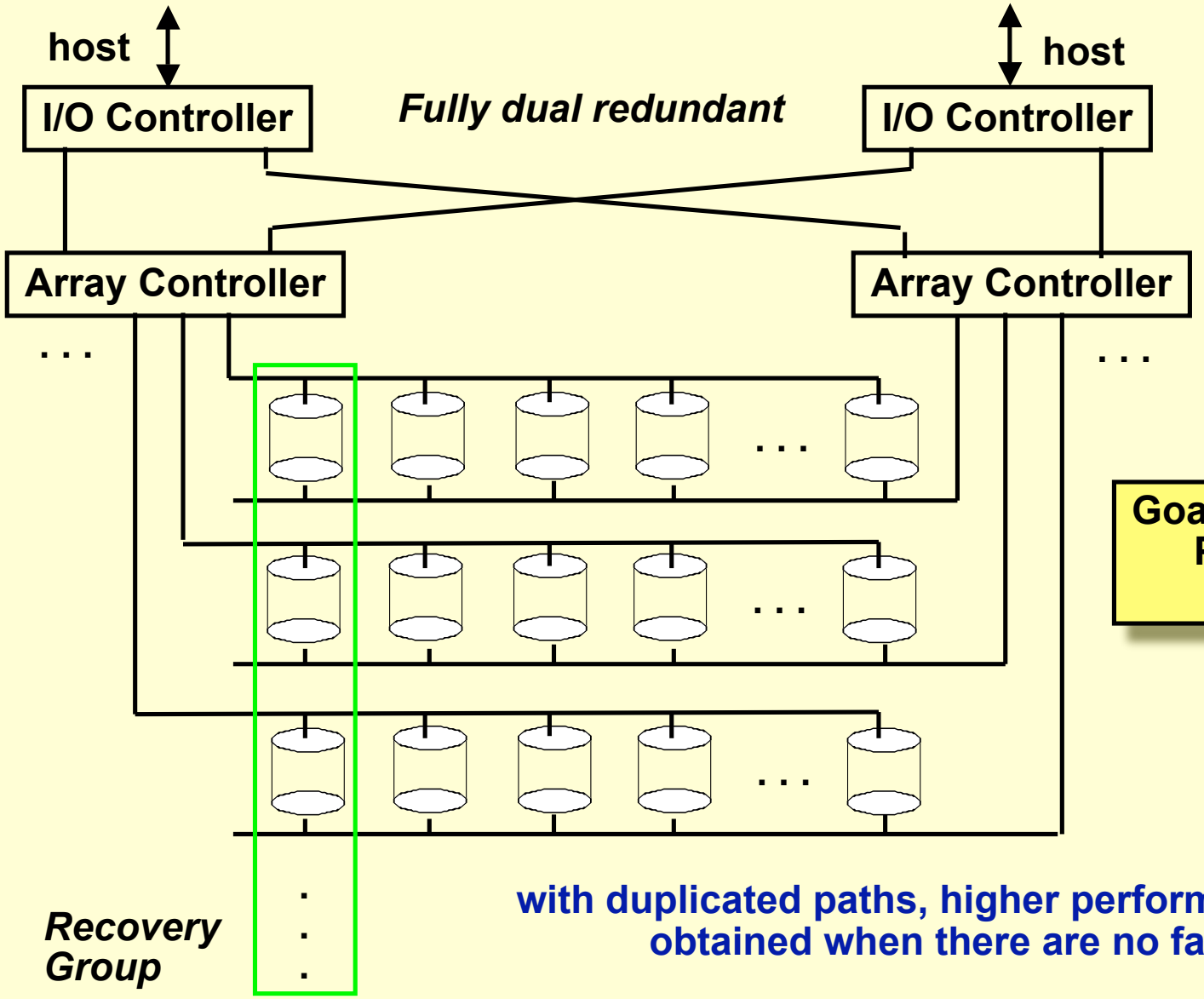
often piggy-backed in small format devices

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

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

I/O Devices' Interface

Two methods are used to address the device:

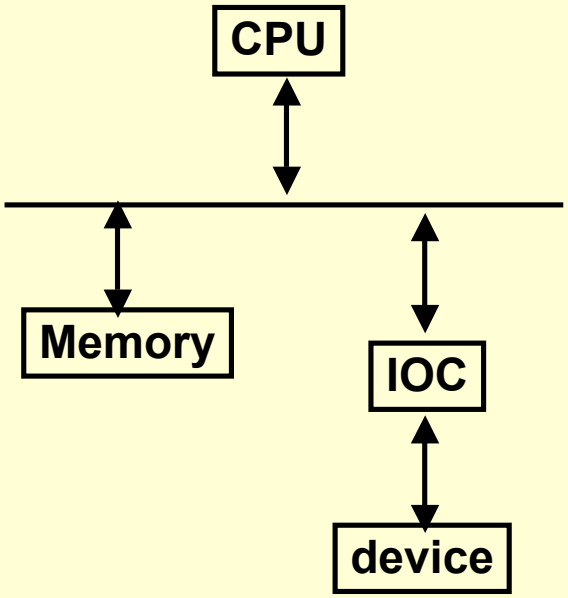
1. *Special I/O instructions*: (Intel 80X86, IBM 370)
   - Specify both the device number and the command word
     - Device number: the processor communicates this via a set of wires normally included as part of the I/O bus
     - Command word: this is usually send on the bus's data lines
     - Each devices maintain status register to indicate progress
   - Instructions are privileged to prevent user tasks from directly accessing the I/O devices
2. *Memory-mapped I/O*: (Motorola/IBM PowerPC)
   - Portions of the address space are assigned to I/O device
   - Read and writes to those addresses are interpreted as commands to the I/O devices
   - User programs are prevented from issuing I/O operations directly:
     - The I/O address space is protected by the address translation

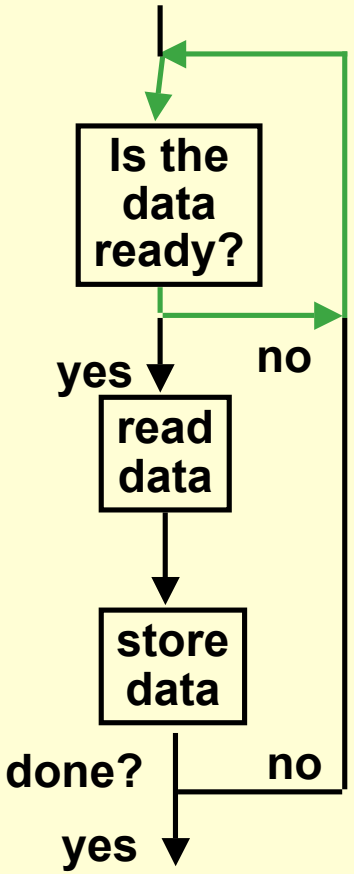
Communicating with I/O Devices

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

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

Polling: Programmed I/O



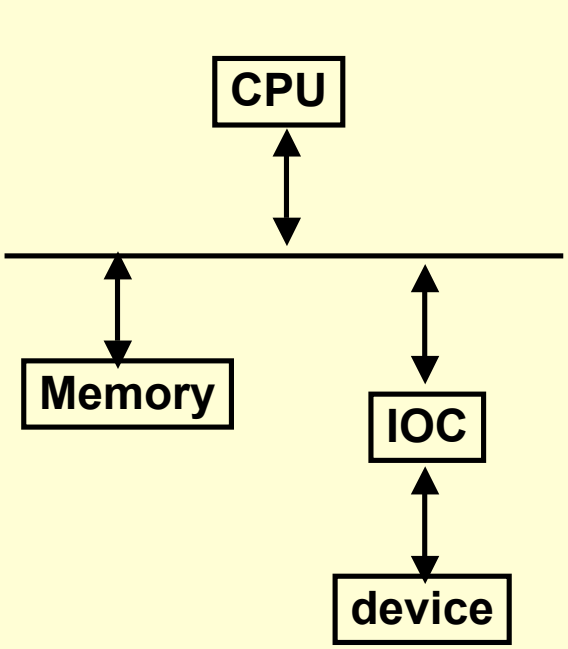


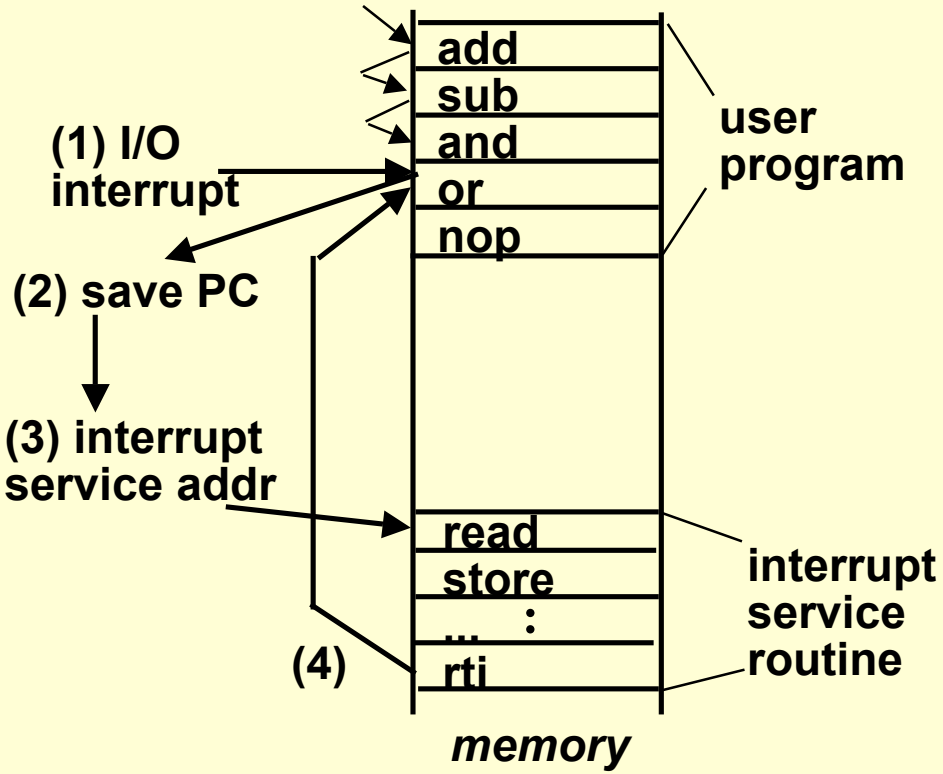
busy wait loop not an efficient way to use the CPU unless the device is very fast!

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)

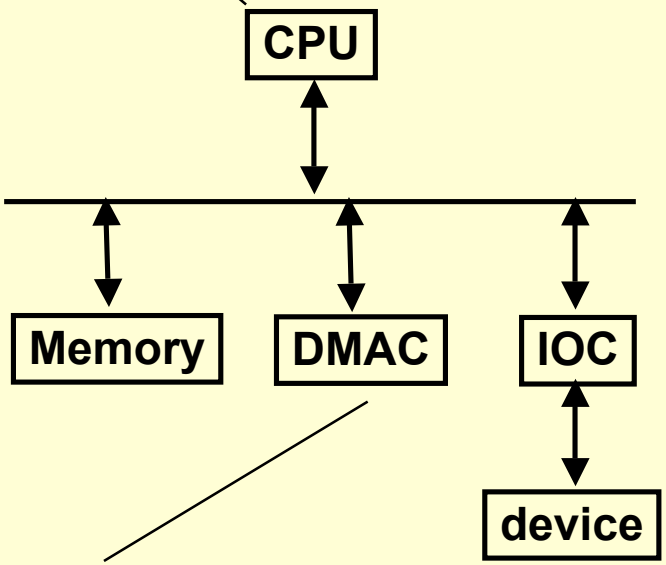
I/O Interrupt vs. Exception

- An I/O interrupt is just like the exceptions except:
  - An I/O interrupt is asynchronous
  - Further information needs to be conveyed
  - Typically exceptions are more urgent than interrupts
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction
  - I/O interrupt does not prevent any instruction from completion
    - You can pick your own convenient point to take an interrupt
- I/O interrupt is more complicated than exception:
  - Needs to convey the identity of the device generating the interrupt
  - Interrupt requests can have different urgencies:
    - Interrupt request needs to be prioritized
    - Priority indicates urgency of dealing with the interrupt
    - high speed devices usually receive highest priority

Direct Memory Access

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

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



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

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

DMA Problems

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

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

Solutions:

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

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

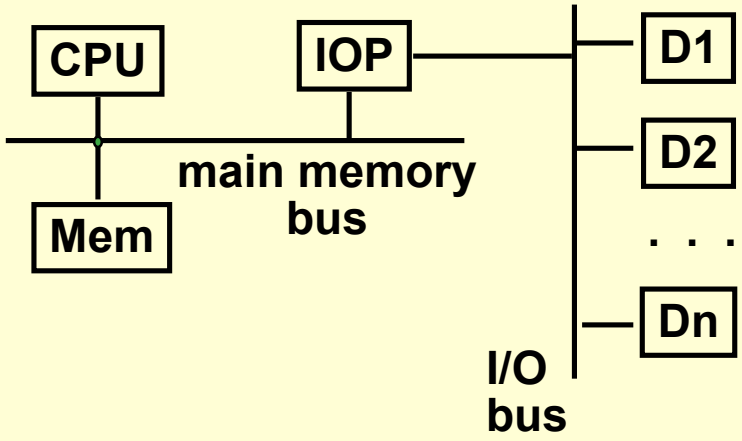
- ➡ Processor might not know that the cache and memory pages are different
- ➡ Write-back caches can overwrite I/O data or makes DMA to read wrong data

Solutions:

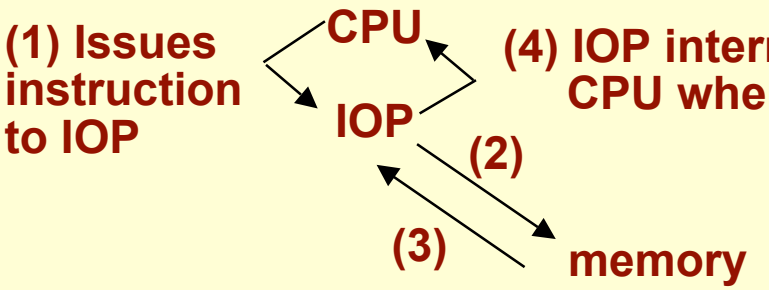
- ➡ Route I/O activities through the cache
  - ⇨ Not efficient since I/O data usually is not demonstrating temporal locality
- ➡ OS selectively invalidates cache blocks before I/O read or force write-back prior to I/O write
  - ⇨ Usually called cache *flushing* and requires hardware support

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

I/O Processor

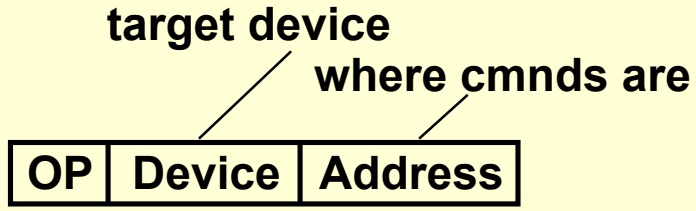


- An I/O processor (IOP) offload the CPU
- Some of the new processors, e.g. Motorola 860, include special purpose IOP for serial communication

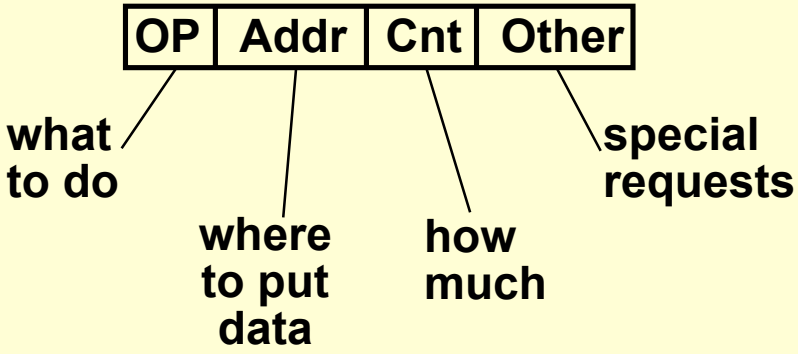


Device to/from memory transfers are controlled by the IOP directly.

IOP steals memory cycles.



IOP looks in memory for commands



Operating System's Role

- Operating system acts as an interface between I/O hardware and programs
- Important characteristics of the I/O systems:
  - The I/O system is shared by multiple program using the processor
  - I/O systems often use interrupts to communicate information about I/O
    - Interrupts must be handled by OS because they cause a transfer to supervisor mode
  - The low-level control of an I/O device is complex:
    - Managing a set of concurrent events
    - The requirements for correct device control are very detailed
- Operating System's Responsibilities
  - Provide protection to shared I/O resources
    - Guarantees that a user's program can only access Provides abstraction for accessing 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