CS525: Advanced Database Organization

Notes 2: Hardware

Gerald Balekaki

Department of Computer Science

Illinois Institute of Technology

gbalekaki@iit.edu

01/16/2025

~~June 26th 2023~~

Slides: adapted from a courses taught by Hector Garcia-Molina, Stanford, Shun Yan Cheung, Andy Pavlo, Paris Koutris, & Leonard McMillan

Overview

- In CS425, we already understand what a database looks like at a logical level and how to write queries to read/write data from it.
- In CS525, we will learn how to build software that manages a database.

Outline

- Study of data storage in a database management systems
- We shall learn the basic techniques for managing data within the computer
- There are two issues we must address which are related to how a DBMS deals with very large amounts of data efficiently:
    - How doesa computer system store and manage very large volumes of data? What
    - representations and data structures best support efficient manipulations of this data?

Today

- Hardware: Disks
- Access Times
- Optimizations
- Other Topics:
    - Storage costs
    - Using secondary storage
    - Disk failures

Disk-Oriented Architecture

- The DBMS assumes that the primary storage location of the database is on non-volatile storage (e.g., HDD, SSD).
  - The database is stored in a file as a collection of fixed length blocks called slotted pages on disk.
- The DBMS's components manage the movement of data between non-volatile and volatile storage.
  - The system uses an in-memory (volatile) buffer pool to cache blocks fetched from disk.
  - Its job is to manage the movement of those blocks back and forth between disk and memory.

- How data is stored on non-volatile storage is crucial for understanding how data is accessed to respond to queries and modify data
- *To understand this further, we want to make the distinction between volatile and non volatile storage.*

Typical Storage Hierarchy

- We will focus on a disk-oriented DBMS architecture that assumes that primary storage location of the database is on non-volatile disk.
- At the top of the storage hierarchy, you have the devices that are closest to the CPU.
  - This is the fastest storage but it is also the smallest and most expensive.
- The further you get away from the CPU, the storage devices have larger the capacities but are much slower and farther away from the CPU.
  - These devices also get cheaper per GB.

Classification of Physical Storage Media

- Can differentiate storage into:
  - Volatile Storage Non-
  - volatile Storage
  - Non-volatile Memory[1]
    - devices are designed to be the best of both worlds: almost as fast as DRAM but with the persistence of disk. We will not cover these devices.
  - Factors affecting choice of storage media include
    - Speed with which data can be accessed Cost
    - per unit of data
    - Reliability

[1] Simulations of Ultralow-Power Nonvolatile Cells for Random-Access Memory

Volatile Storage

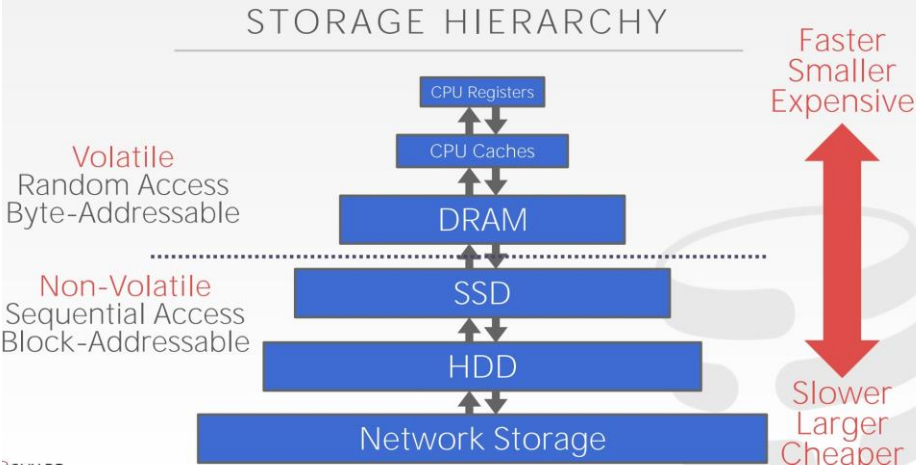
- Volatile means that if you pull the power from the machine, then the data is lost.
  - Loses contents when power is switched off
- Supports fast random access with byte-addressable locations.
  - This means that the program can jump to any byte address and get the data that is there.
- For our purposes, we will always refer to this storage class as memory

Non-Volatile Storage

- Non-volatile means that the storage device does not need to be provided continuous power in order for the device to retain the bits that it is storing
  - Contents persist even when power is switched off.
- Block/page addressable.
  - to read a value at a particular offset, the program first has to load the 4 KB page into memory that holds the value the program wants to read.
- Traditionally better at sequential access
  - reading multiple contiguous chunks of data at the same time
- We will refer to this as disk. We will not make a (major) distinction between solid-state storage (SSD) or spinning hard drives (HDD).

Persistent Memory

- There is also a new class of storage devices that are becoming more popular calledPersistent memory.
- Persistent memory (PMEM)is a solid-state high-performance byte-addressable memory device that resides on the memory bus
- These devices are designed to be the best of both worlds: almost as fast as DRAM with the persistence of disk.
- We will not cover these devices in this course.


STORAGE HIERARCHY
Faster
Smaller
Expensive
Volatile
Random Access
Byte-Addressable
CPU Registers
CPU Caches
DRAM
Non-Volatile
Sequential Access
Block-Addressable
SSD
HDD
Network Storage
Slower
Larger
Cheaper

Disks

- A high-level design goal of the DBMS is to support databases that exceed the amount of memory available.
- DBMS stores information on ("hard") disks. This
- has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM). WRITE:
  - transfer data from RAM to disk.
  - Reading/writingoperations to disk is expensive, relative to in-memory operations, so it must be managed carefully to avoid large stalls and performance degradation

Disks

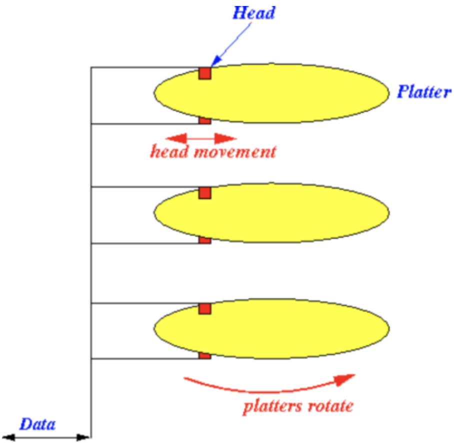
- The use of non-volatile storage is one of the important characteristics of a DBMS.
- To motivate many of the ideas used in DBMS implementation, we must examine the operation of disks in detail

Disks

- Secondary storage device of choice
- random access vs. sequential
  - Sequential: read the data contiguously
  - Random: read the data from anywhere at anytime
- Data is stored and retrieved in units called disk blocks or pages
- Retrieval time depends upon the location of the disk
  - Therefore, relative placement of pages on disk has major impact on DBMS performance! Why?

Components of a Disk

- A disk contains multipleplatters (usually2 surfacesper platter)
  - Platter:circular hard surface on which data is stored by inducing magnetic changes
- Plattersrotates (7200 RPM - 15000 RPM)
  - RPM (Rotations Per Minute)
- Usually, the disk contains read/write heads that allow to read/write from all surfaces simultaneously
- All disk heads move at the same time (in or out)



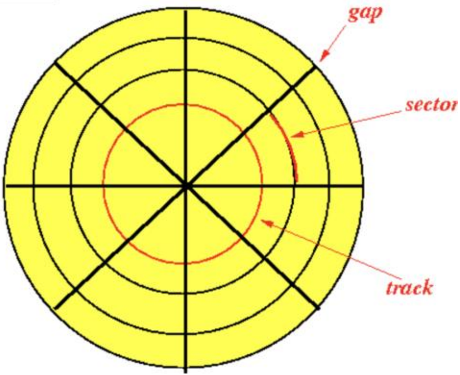
Disks

- Surface of platter divided into circular tracks
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into sectors[1]. Sectors are separated from each other by blank spaces
  - Gaps are non-magnetic and used to identify the **start** of a sector
- A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
- A disk block (disk page)[2] is usually composed of a number of consecutive sectors (determined by the operating system)
  - Data are read/written in units of a disk block (or disk page) A disk
  - block is the same size as a memory block or page.
  - Block size: 4K-64K bytes

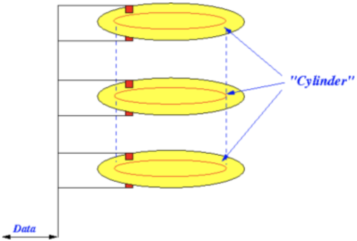
[1] *Sector is a physical unit of the disk*

[2] *Block is a logical unit, a creation of whatever software system (OS or DBMS) is using the disk*

Top View of a Platter



Terminology: cylinder



- One track from each surface will be under the head for that surface and will therefore be readable and writable.
- The tracks that are under the heads at the same time are said to form a cylinder.
  - i.e., the cylinder is the sum total of every track with the same track number on every surface.
  - Cylinder *i* consists of i^{th} track of all the surfaces.
  - Disk head does not need to move when accessing (read/write) data in the same cylinder

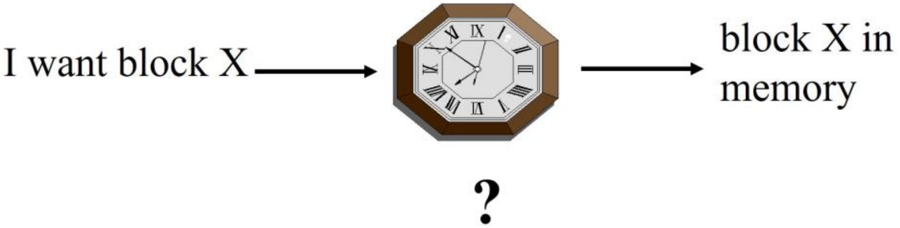
Disk Storage Characteristics

- # Cylinders= # tracks per surface (platter)
  - e.g., 10 tracks ⇒ 10 cylinders and we can refer to them cylinder zero to cylinder nine
- # tracks per cylinder= # of heads or 2 × # platter Average #
- sectors per track
- bytes per sector
- ⇒disk capacity/size

Today

- Hardware: Disks
- Access Times
- Optimizations
- Other Topics:
  - Storage costs
  - Using secondary storage
  - Disk failures

Accessing the Disk



- The time taken between the moment at which the command to read a block is issued and the time that the contents of the block appear in main memory is called thelatency of the disk.
- Theaccess timeis also called thelatency of the disk.

Accessing the Disk

- Basic operations:
  - READ: transfer data from disk to buffer
  - WRITE: transfer data from buffer to disk
- Reading a disk block:
  - Reading a block from disk requires the disk to start spinning Disk
  - arm has to be moved to the correct track of the disk
  - The disk head must wait until the right location on the track is found Then,
  - the disk block can be read from disk and copied to memory

Accessing the Disk

access time = seek time + rotational delay + transfer time +other delay

- Other Delays:
    - CPU time to issue I/O
    - Contention for controller
        - Different programs can be using the disk
    - Contention for bus, memory
        - Different programs can be transferring data
    - These delays are negligible compared to *seek time + rotational delay + transfer time*
    - “Typical” Value: 0

Accessing the Disk

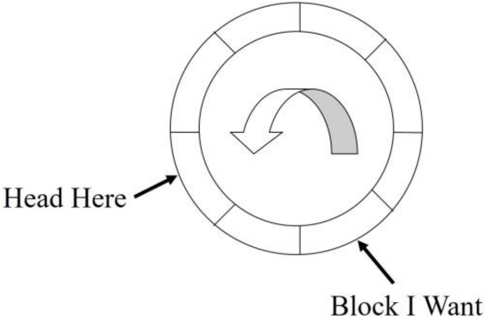
access time = seek time + rotational delay + transfer time

- Seek time: time to move the arm to position disk head on the right track (position the read/write head at the proper cylinder)
- Seek timecan be 0 if the heads happen already to be at the proper cylinder.
  - If not, the heads require some minimum time to start moving and to stop again, plus additional time that is roughly proportional to the distance traveled.
- Theaverage seek timeis often used as a way to characterize the speed of the disk.

Accessing the Disk

access time = seek time + rotational delay + transfer time rotational delay: time

- to wait for sector to rotate under the disk head i.e., wait for the beginning
- of the block
- *how long it takes to get to the correct sector*



Average Rotational Delay

- On the average, the desired sector will be about half way around the circle when the heads arrive at its cylinder.
- Average rotational delay is time for $\frac{1}{2}$ revolution
- Example: Given a total revolution of 7200 RPM
  - One rotation = $\frac{60s \times 1000}{7200} = 8.33$ms
  - Average rotational latency = 4.16 ms

Accessing the Disk

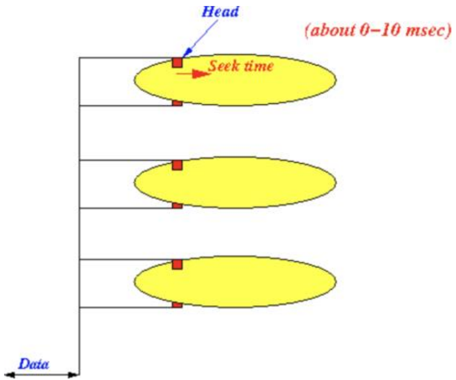
access time = seek time + rotational delay + transfer time

- Data transfer rate: the rate at which data can be retrieved from or stored to the disk.
  - Transfer rate: # bitstransferred/sec
- Transfer timeis the time it takes the sectors of the block and any gaps between them to rotate past the head.
- We can calculate thetransfer timeby dividing the size of a byte sector by the transfer rate.
  - Given a transfer rate, the transfer time = $\frac{\textit{Block size}}{\textit{transfer rate}}$

Steps to access data on a disk

1. Move the disk heads to the desired cylinder

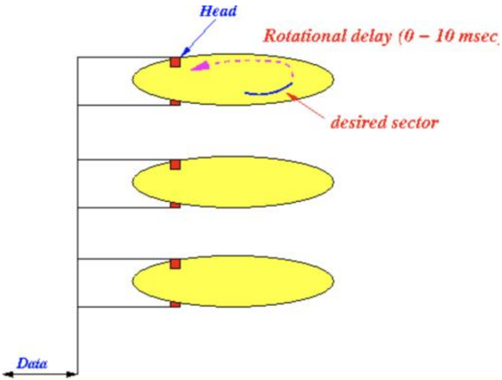
- Time to seek a cylinder = seek time



Steps to access data on a disk

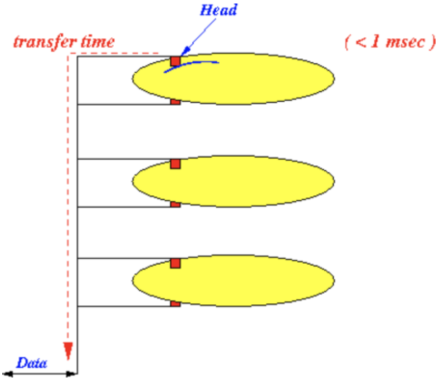
2.Wait for the desired sector to arrive under the disk head

- Time to wait for a sector = rotational delay



Steps to access data on a disk

3. Transfer the data from sector to main memory (through the disk controller)



Accessing the Disk

- Seek time and rotational delay dominate.
- Key to lower I/O cost: reduce seek/rotation delays!

Arranging Blocks on Disk

- So far: One (Random) Block Access
- What about: Reading “Next” block?
- Blocks in a file should be arranged sequentially on disk (by “next”) to minimize seek and rotational delays.
- Next block concept:
  - blocks on same track, followed by blocks
  - on same cylinder, followed by blocks on
  - adjacent cylinder
- For a sequential scan, pre-fetching several blocks at a time is a big win.

If we do things right

- (e.g., Double Buffer, Stagger Blocks...)
- Time to get blocks should be proportional to the size of blocks, and the seek time and rotational latency thus become trivial
- time to get block $= \frac{Block\ size}{transfer\ rate} + Negligible$
- Negligible:
  - skip gap
  - switch track
  - once in a while, next cylinder

Rule of Thumb

- Sequential access pattern
  - Successive requests are for successive disk blocks Disk
  - seek required only for first block
- Random access pattern
  - Successive requests are for blocks that can be anywhere on disk Each
  - access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks

- Random I/O: Expensive
- Sequential I/O: Much less

Cost for Writing similar to Reading

- The process of writing a block is, in its simplest form, quite similar to reading a block
- . . . unless we want to verify!
- need to add (full) rotation + $\frac{Block\ \ size}{transfer\ \ rate}$

To Modify a Block?

It is not possible to modify a block on disk directly. Rather, even if we wish to modify only a few bytes, we must do the following:

1. Read Block into Memory
2. Modify in Memory Write
3. Block
4. [Verify?]

Megatron 747 Disk (old)

Example

- Rotate at 3600 RPM
- Only 1 surface
- 16 MB usable capacity (usable capacity excludes the gaps) 128
- cylinders
- seek time:
  - average = 25 ms.
  - adjacent cylinders = 5 ms.
- 1 KB block = 1 sector
- 10% overhead between blocks
  - gaps represent 10% of the circle and
  - sectors represent the remaining 90%

Megatron 747 Disk (old)

- 1 KB blocks = sectors
- 10% overhead between blocks
- capacity = 16 MB = $(2^{20})16 = 2^{24}$
- # cylinders = 128 = 2^7
- bytes/cylinder = $\frac{total\ capacity}{total\ \#\ cylinders} = \frac{2^{20} \times 16}{128} = \frac{2^{24}}{2^7} = 2^{17}$ =128KB
- #blocks/cylinder = $\frac{capacity\ of\ each\ cylinder}{size\ of\ block} = \frac{128KB}{1KB}$ = 128

Megatron 747 Disk (old)

- 3600 RPM $\rightarrow$ 60 revolutions/sec$\rightarrow$1 rev. = 16.66 msec.

One track:

- Time over useful data= $16.66 \times 0.9 = 14.99$ ms
- Time overgaps=$16.66 \times 0.1 = 1.66$ ms
- Transfer time[3] for 1 block = $\frac{14.99}{128} = 0.117$ms
- Transfer time for 1 block+gap= $\frac{16.66}{128} = 0.13$ms

[3] *Transfer time is the time it takes the sectors of the block and any gaps between them to rotate past the head. Divide the amount of data by the transfer speed to find the transfer time.*

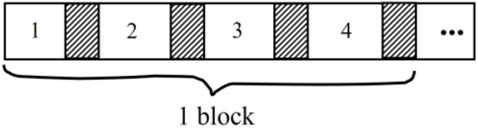
Megatron 747 Disk (old)

- Access time (T_1) = Time to read one random block
- T_1 = seek + rotational delay + transfer time for 1 block
- $T_1 = 25 + \frac{16.66}{2} + 0.117 = 33.45$ ms.

Why we did not use the time it takes to transfer 1 block+gap here?

Megatron 747 Disk (old)

- Suppose OS deals with 4 KB blocks



- Access time = $T_4 = 25 + \frac{16.66}{2} + 0.117 \times 1 + 0.13 \times 3 = 33.83$ ms
- Compare to $T_1 = 33.45$ms
- Q) The time to read a full track is?

Today

- Hardware: Disks
- Access Times
- Optimizations
- Other Topics:
  - Storage costs
  - Using secondary storage
  - Disk failures

Optimizations (in controller or O.S.)

Effective ways to speed up disk accesses:

- Disk Scheduling Algorithms
  - e.g., elevator algorithm
- Track (or larger) Buffer
- Pre-fetch (a.k.a. Double buffering)
- Disk Arrays
- On Disk Cache

Disk Scheduling

- The disk controller can order the requests to minimize seeks Situation: Have
- many read/write requests at any one moment in time
- Question:Service policy: In which order the disk controller process (service) the requests?
  - The order in which you service the disk operations can affect the performance
- Na¨ıveservice (but fair):First Come First Serve
  - Fairness but inefficient (e.g. zig-zag readpattern)
- Commonly used disk scheduling algorithm:the “elevator” algorithm
  - Elevator scheduling for a disk:
    - The disk head sweeps in-and-out (like an elevator) When
    - the disk head is on a cylinder *k*:
      - Disk will service all requests for that cylinder before moving to the next cylinder
  - Efficient but unfair

Pre-fetching (Double Buffer)

- Another suggestion for speeding up some secondary-memory algorithms is called double buffering.
- In some scenarios, we can predict the order in which blocks will be requested from disk by some process.
- Pre-fetching (double buffering) is the method of fetching the necessary blocks into the buffer in advance
- Requires enough buffer space
- Speedup factor up to n, where n is the number of blocks requested by a process

Double Buffering Algorithm

Problem

- Have a File
    - Sequence of Blocks B1, B2, ...
- Have a Program
    - Process B1
    - Process B2
    - Process B3
    - ⋮

Single Buffer Solution (Naïve Solution)

1. Read B1 $\longrightarrow$ Buffer
2. Process Data in Buffer
3. Read B2 $\longrightarrow$ Buffer
4. Process Data in Buffer
5. :

Single Buffer Solution

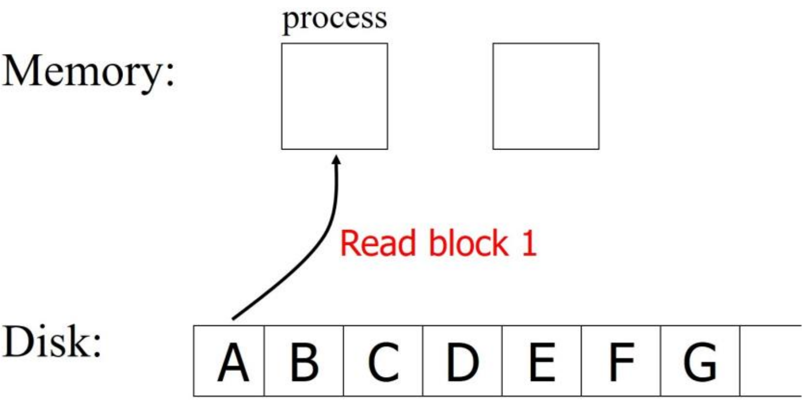
Let:

- P = time to process/block
- R = time to read in 1block
- n = #blocks
  1. Read B1 $\longrightarrow$ Buffer $\Rightarrow$ R
  2. Process Data in Buffer $\Rightarrow$ P
  3. Read B2 $\longrightarrow$ Buffer $\Rightarrow$ R
  4. Process Data in Buffer $\Rightarrow$ P
- Time to process n block=n(P+R)

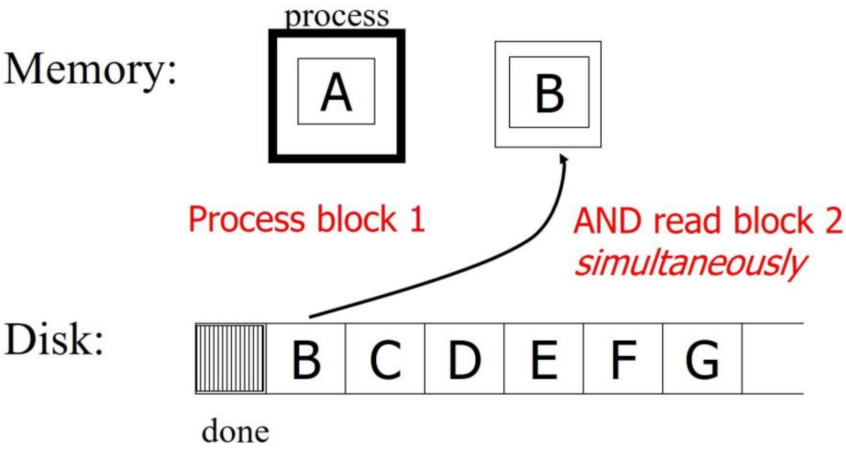
Double Buffering Solution

- The program allocates two buffers to process data from a file
- Data is read in a buffer
- When buffer is full, program processes the data. And at the same time, more data is read in the other buffer
- Rotate buffers when done processing data in buffer

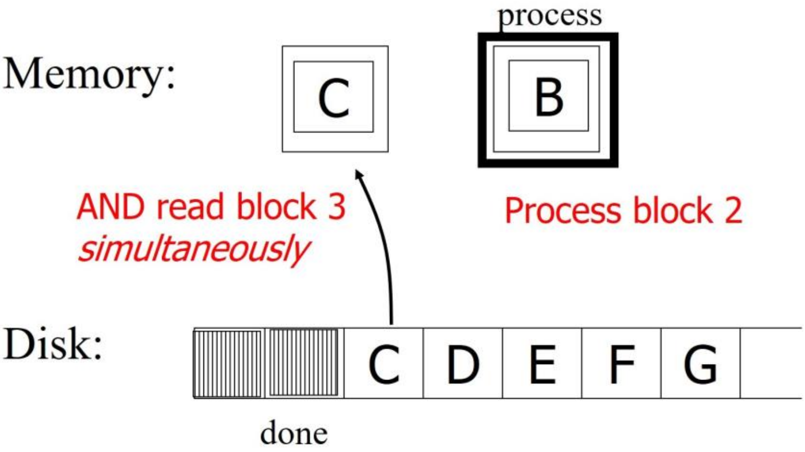
Double Buffering Solution



Double Buffering Solution



Double Buffering Solution



Double Buffer Solution

Let:

- P = time to process/block
- R = time to read in 1block
- n = #blocks
- Say P $\geq$ R

What is processing time?

- Double buffering time = R+nP
- Single buffer time = n(R+P)

Using disk array to accelerate disk access

- Speed of access and reliability of disks can be increased by simply using multiple disks.
- Reliability is the ability of the disk system to accommodate a single- or multi-disk failure and still remain available to the users.
- Why use multiple disks
  - Multiple disks $\longrightarrow$ multiple disk heads
  - Multiple outputs = Increased data rate

Techniques: multiple disks

- Block Striping
  - Store blocks of a file over multiple disks
  - High read and write speed
- Mirror disk
  - Store the same data on multiple disks
  - Mirrored disks contain identical content
  - Read operation: n times as fast
  - Write operation: about the same as 1 disk
- RAID
  - Redundant arrays of inexpensive disks
    - Is a simple theory of using multiple disks to increase both speed of access and reliability of disks.
    - RAID can be implemented using a hardware controller or a software controller.
    - Different levels provide different solutions at different price points.

Disk Failures

We consider ways in which disks can fail and what can be done to mitigate these failures:

- The disk is OK. But: due to electrical fluctuations, a disk read (or disk write) operation failed (a one time event)
  - Intermittent read failure (Cause: power fluctuations/failure)
  - Intermittent write failure (Cause: powerfluctuation/failure)
- Media decay (Disk surface worn out)
  - A sector is worn
  - The sector is part of a block and it can no longer be used
- Permanent failure (Disk crash)
  - The disk head has scratched the platter(s) Data
  - on the whole disk is lost

Coping with Read/Write Failures

- Detection
  - Read (verify) after writing data
  - Better: Use checksum
- Correction
  - Redundancy

Coping with media decay

- Handling media decay: Replacing bad sectors/blocks Disk
- has a number of spare blocks
- When writing a block fails for ntimes
  - Mark block as bad
  - Replace block with one of the spare blocks
- Effect of bad sectors/blocks: The disk capacity isreduced

Coping with Disk Crash

- Only way to recover from a disk crash: Redundancy (e.g., backup copy)
- Different ways to achieve redundancy
  - Exact copy (mirror)
  - RAID

Summary

- Secondary storage, mainly disks
- I/O times
- I/Os should be avoided, especially random ones

Next

- ~~File and System Structure~~ (Database Storage)