

CMPT 506

Storage and Database File Organization



*Please read
Chapter 17*

Dr. Abdelkarim Erradi

Department of Computer Science and Engineering

QU

Outline

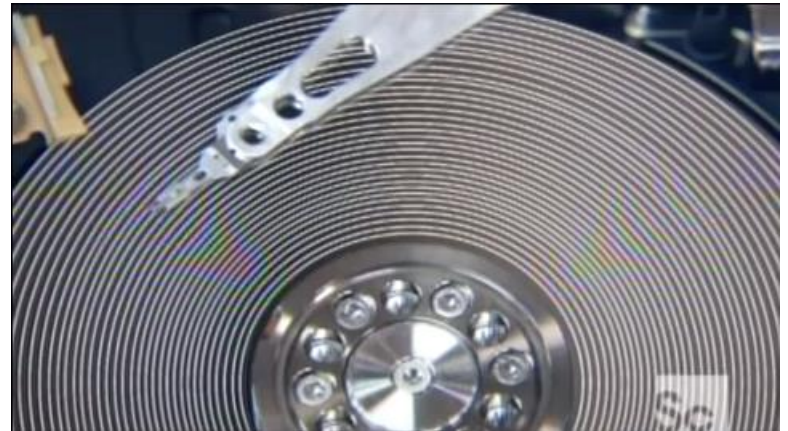
- ① Database Storage Technologies
- ② RAID Technology
- ③ Database File Organization
- ④ Extendible Hashing & Linear Hashing

(4 will be covered in the Indexing Lecture)

Acknowledgment

Some slides are based on textbook slides

Database Storage Technologies



Why study the physical level of DBMS

- Someone has to write the DBMS software and its file manager!
- Some DB systems give the database administrator a range of options for the mapping of the data to physical storage.
 - Intelligent use of these options can make a very significant (and user-noticeable) difference in the way the system performs.
 - To "tune" the system properly, the DBA must understand what is happening at the physical level.
- Some of the techniques and algorithms can be used to solve other problems in other contexts

Key components of DBMS performance

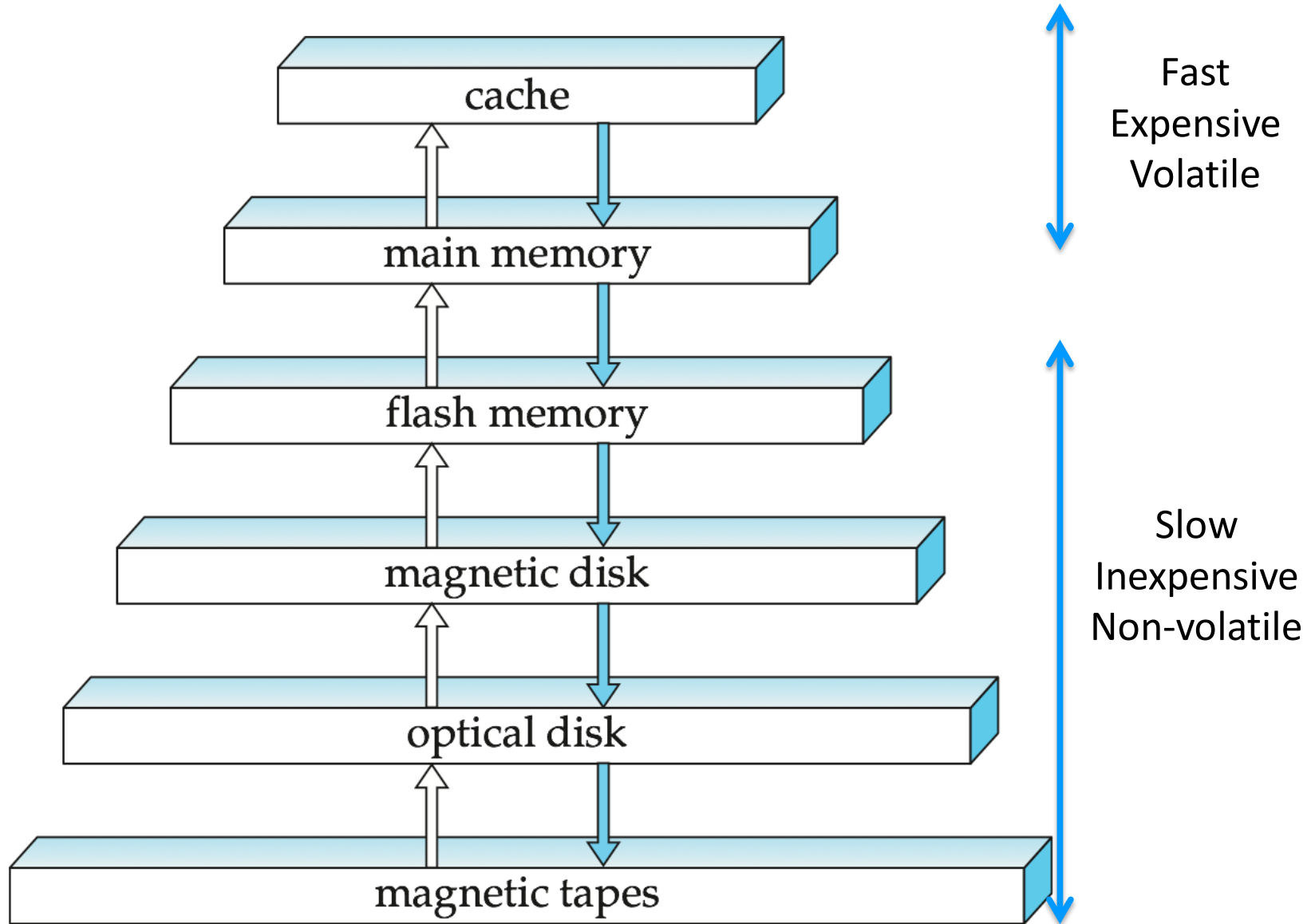
- The **performance of the DBMS file system is often the key component of overall performance**. There are two attributes that can be optimized:
 - 1. Response time** - defined as the time between the issuance of a command and the time that output for the command begins to be available.
 - e.g. if the command is a select statement, the time until the first row of the result appears

=> we want to minimize this
 - 2. Throughput** - the number of operations that can be completed per unit time.

=> we want to maximize this

Storage Hierarchy

Cost decreases and access time increases



The Memory Hierarchy

- **Primary storage:** Fastest media but volatile
 - Volatility = information is lost when an application terminates (normally or due to a power failure or crash)
 - Can hold subset of a database used by current transactions.
- **Cache**
 - Data and instructions in cache when needed by CPU.
 - On-board (L1) cache on same chip as CPU, L2 cache on separate chip.
 - Capacity couple of MBs, **access time a few nanoseconds**
- **Main memory**
 - All active programs and data need to be in main memory.
 - Capacity couple of GBs, access time **10-100 nanoseconds**

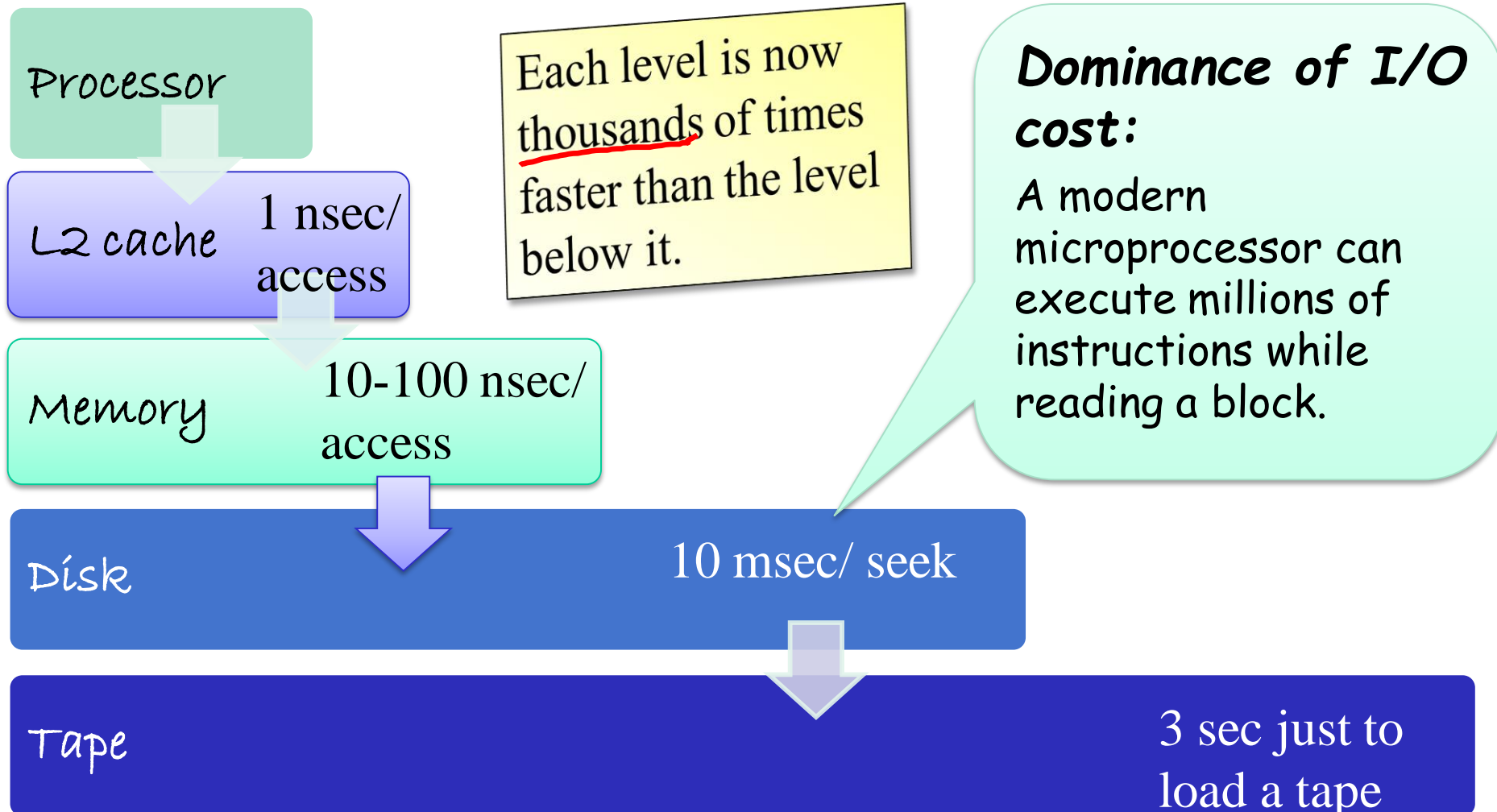
Storage Hierarchy (Cont.)

- **Secondary storage:** non-volatile, moderately fast access time
 - also called **on-line storage** -of current version of entire database-
 - typically used for permanent storage of large amounts of data, typically a magnetic disk.
 - Capacity up to 1 TB, access time ~ 10 milliseconds
- **Tertiary storage:** non-volatile, slow access time
 - also called **off-line storage** – often used for archiving older versions of the database
 - Capacity ~ 1 PB, access time seconds / minutes.
 - E.g. magnetic tape, optical storage

Large speed gap between Memory and Disk

- The large speed gap between primary and secondary storage technologies remains the key issue in DBMS performance.
- Time to access information in secondary storage is the **major determining factor in system performance**.
- The *number of disk I/Os* (block accesses) is a good approximation for the cost of a database operation.

The relative gaps in performance are increasing



Hard Disks

- Secondary storage device of choice.
- Data is stored and retrieved in units called *disk blocks* or *pages* (typically 4 or 16 kilobytes)
- Main advantage over tapes: *random access* vs. *sequential access*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
 - Reading several pages in sequence from a disk takes much less time than reading several random pages
- Therefore, relative placement of pages on disk has major impact on DBMS performance!

What's Inside A Disk Drive?

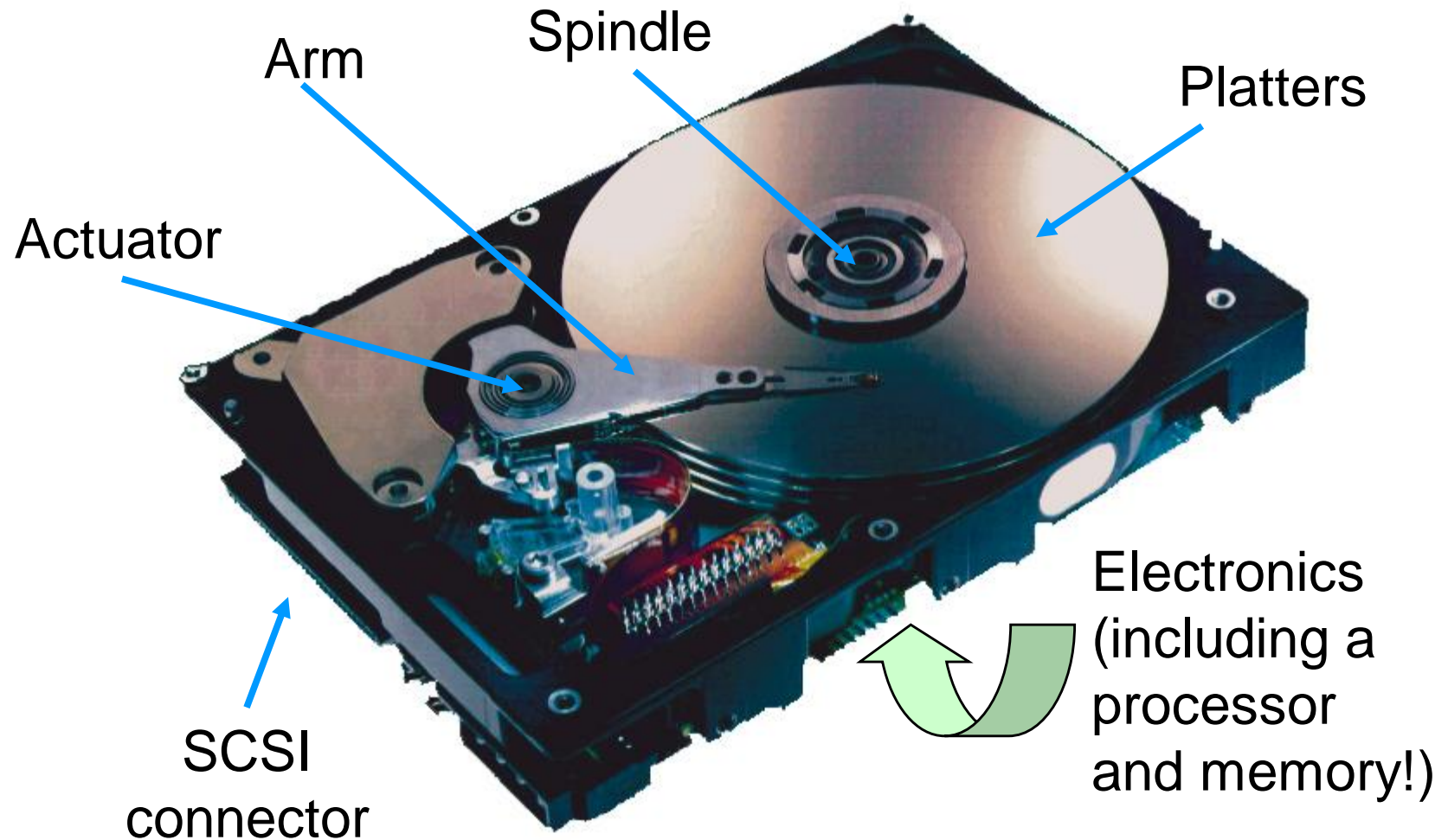
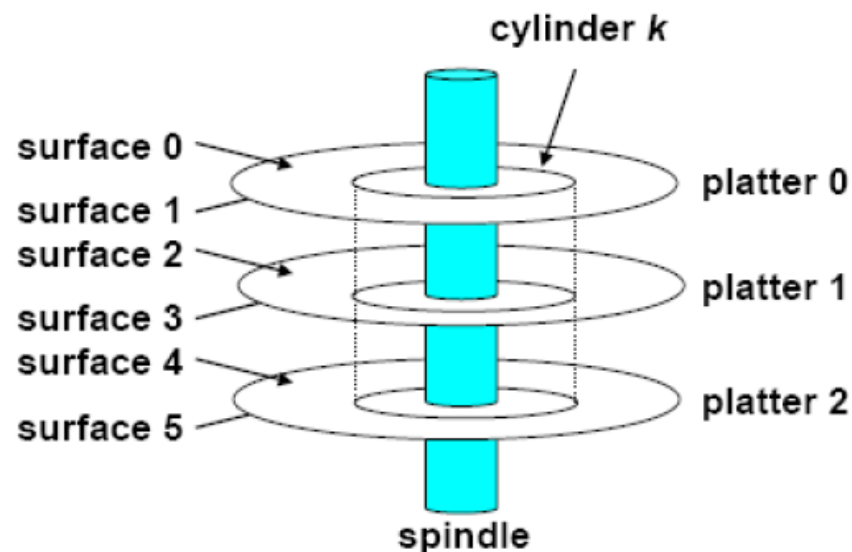
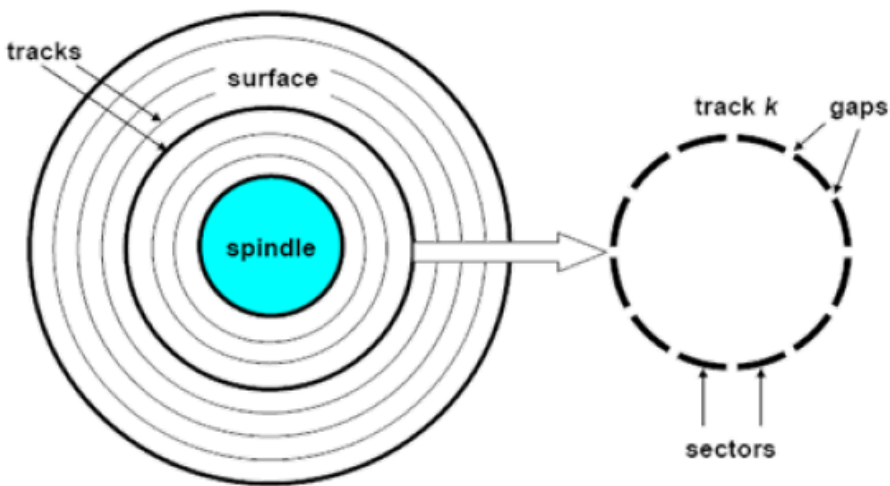


Image courtesy of Seagate Technology

Inside the HD <http://www.youtube.com/watch?v=kdmLv11n82U>

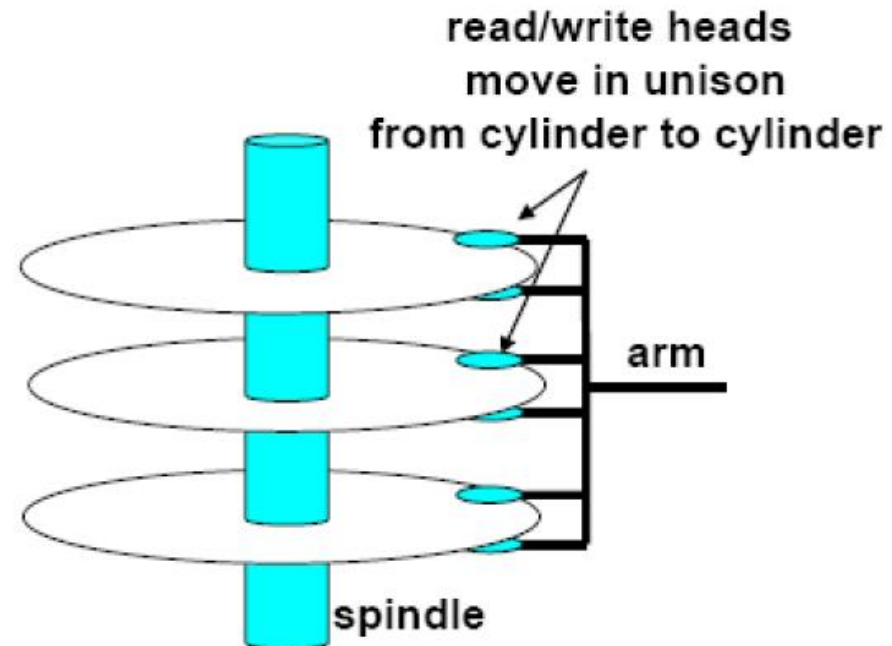
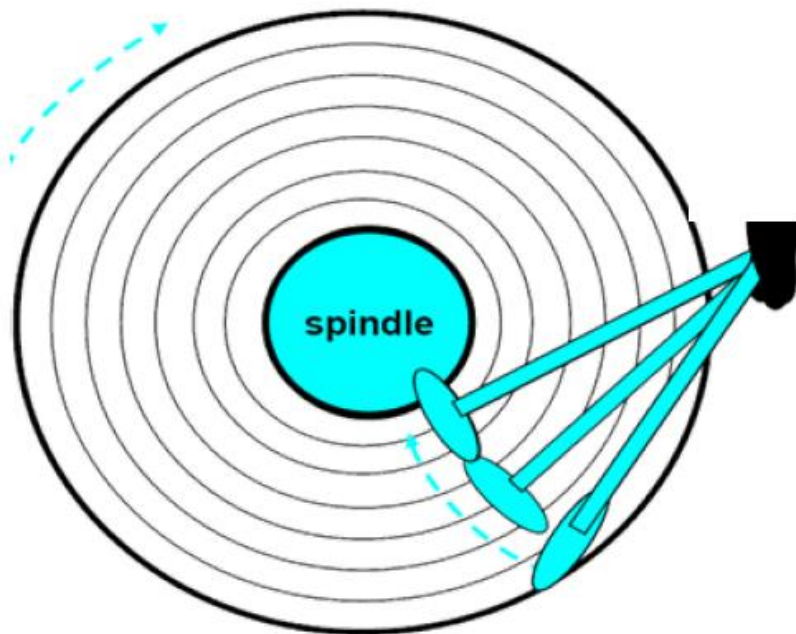
Disk Physical Structure

- Disks consist of **platters**, each with two surfaces
- Each surface consists of concentric rings called **tracks**
- Each track consists of **sectors** separated by gaps
 - Track capacities vary typically from 4 to 50 Kbytes or more
- All tracks under heads at the same time make a **cylinder** (imaginary!).
- Only one head reads/writes at any one time.

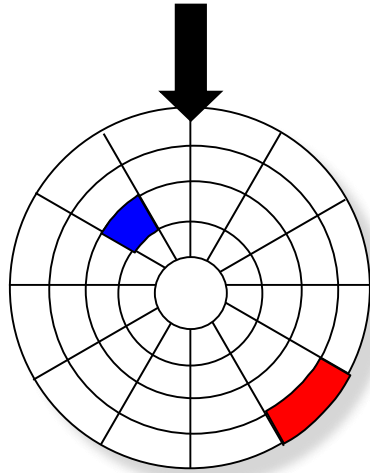


Disk Operation (Single-Platter View)

- The disk surface spins at a fixed rotational rate
- The read/write head is attached to the end of the arm and flies over the disk surface
- By **moving radially**, the arm can position the read/write head over any track

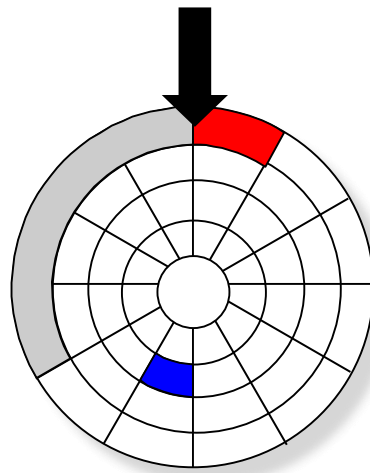


Disk Access – Service Time Components



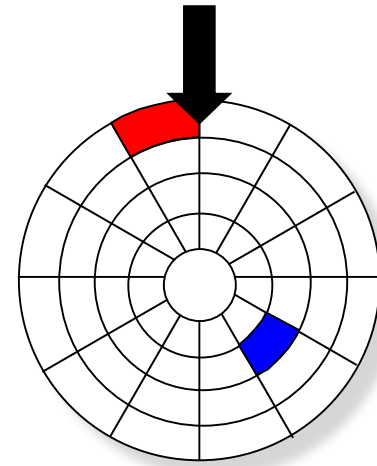
Seek for **RED**

↑
Seek
moving arms to
position disk head
on track



Rotational latency

↑
Rotational latency
waiting for block to
rotate under head



After **RED** read

↑
Data transfer
moving data to/
from disk surface

Typically about 1% of the time is actually spent on data transfer, the rest is access time.

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
 - **Seek time** – time it takes to reposition the arm over the correct track.
 - 4 to 10 milliseconds on typical disks
 - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
 - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
 - **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
 - 25 to 100 MB per second rate, lower for inner tracks
- Access time dominated by seek time and rotational latency.
 - Disk is about 40,000 times slower than RAM

Disk speeds are dominated by access time

- For this reason, information on disk is always organized in Blocks – blocks are **basic units of transfer and storage**
 - relatively large chunks (4 or 16 kilobytes) of contiguous information that is read/written as a unit.
 - it always **reads or writes the whole block** containing a desired piece of information.
 - A system never reads or writes a single disk byte.
 - The block size B is fixed for each system.
 - Typical block sizes range from 4 to 16 kilobytes
- Mapping between logical blocks and actual (physical) sectors
 - Maintained by hardware/firmware device called disk controller.
 - Converts requests for logical blocks into (surface,track,sector) triples.

Optimization of Disk Block Access

A major goal of the design of DBMS file systems is to **minimize the time spent waiting for disk accesses**. 3 ways this is done:

1) Store related information on the same or nearby blocks:

read and write of data on ***contiguous disk blocks*** and eliminates seek time and rotational delay time for all but the first block transfer

- Files may get fragmented over time (if data is inserted to/deleted from the file) => reorganize the database files to speed up access

2) Keeping copies of recently-used information in buffers in memory, so that if the same information is needed again it can be accessed without having to go to the disk again

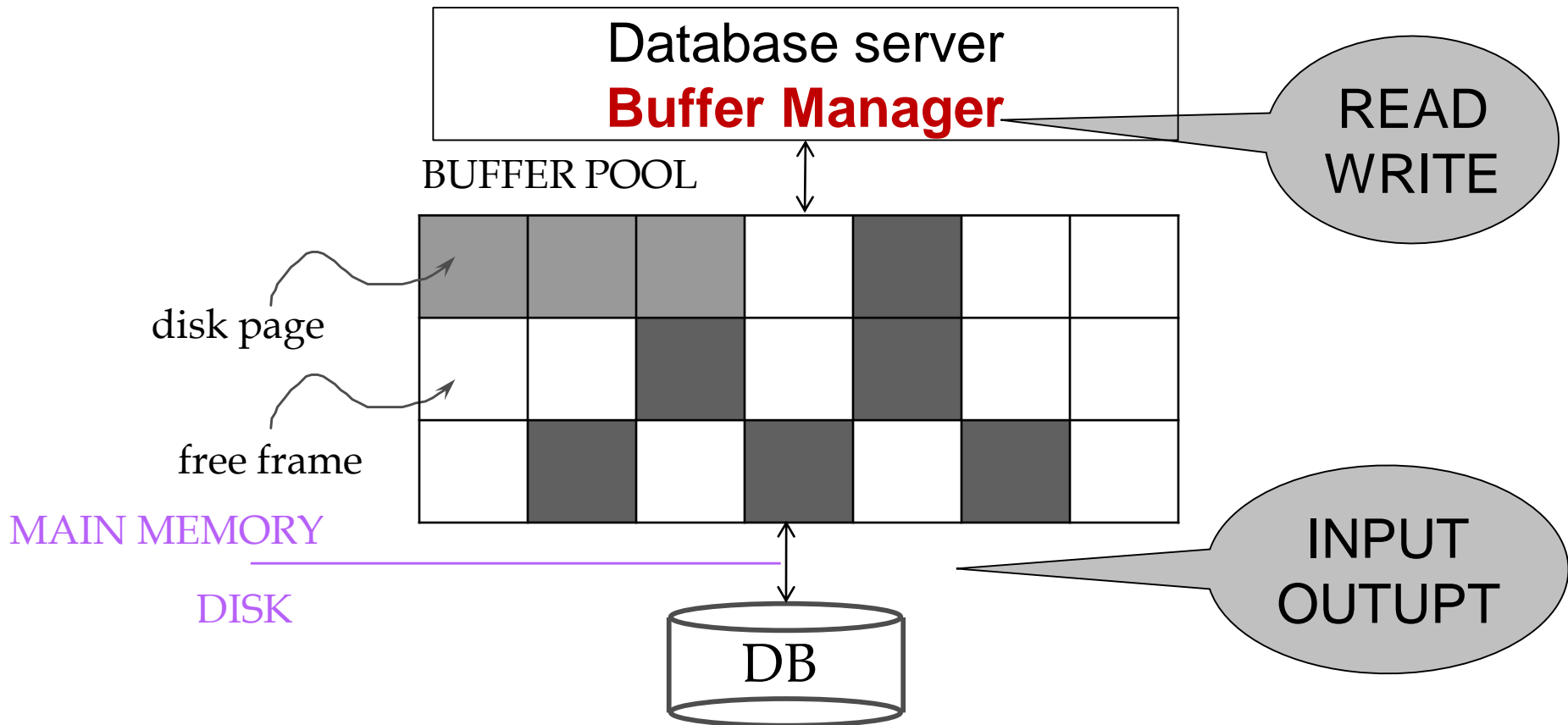
3) Parallelism - spreading information across multiple disks, so that several disks can be going through the physical operations needed to access information at the same time

Example: reading two disk blocks

- Assume
 - average seek time = 10 ms
 - average rotational latency = 3 ms
 - transfer time for 1 block = 0.01875 ms
- Adjacent blocks on same track
 - access time = $10 + 3 + 2 \times (0.01875)$ ms = 13.0375 ms
- Random blocks
 - access time = $2 \times (10 + 3 + 0.01875)$ ms = 26.0375 ms

Buffer Management in a DBMS

Large gap between disk I/O and memory → Buffer pool



- *Data must be in RAM for DBMS to operate on it!*
- *Table of $\langle \text{frame\#}, \text{pageid} \rangle$ pairs is maintained*

Buffer Manager

Manages buffer pool: the pool provides space for a limited number of pages from disk.

Needs to decide on page replacement policy
E.g., Least Recently Used (LRU)

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Why not use the Operating System for the task??

- DBMS may be able to anticipate **access patterns**
- Hence, may also be able to perform **prefetching**
- DBMS needs the ability to **force** pages to disk

When a Page is Requested ...

- Buffer pool information table contains:
<frame#, pageid, pin_count, dirty>
- If requested page is not in pool:
 - Choose a frame for *replacement*.
Only frames with pin_count == 0 are candidates!
 - If frame is “dirty”, write it to disk
 - Read requested page into chosen frame
- *Pin* the page and return its address

*If requests can be predicted (e.g., sequential scans)
pages can be pre-fetched several pages at a time!*

More on Buffer Management

- Requestor of page must eventually unpin it (to indicate it is no longer needed) + indicate whether page has been modified: *dirty* bit is used for this.
- Page in pool may be requested many times,
 - a *pin count* is used.
 - To pin a page, `pin_count++`
 - A page is a candidate for replacement if *pin count* == 0 (*“unpinned”*)
- Frame is chosen for replacement by a *replacement policy*:
 - Least-recently-used (LRU)
 - Most-recently-used (MRU)
 - Other policies

Double Buffering

Double buffering (prefetching) uses two buffers.
While one is being used, the other is being filled.

Motivating Example

We have a File

- » Sequence of Blocks B1, B2, B3, ...

Have a Program

- » Process B1
- » Process B2
- » Process B3

Single Buffer Solution

- (1) Read B1 \rightarrow Buffer
- (2) Process Data in Buffer
- (3) Read B2 \rightarrow Buffer
- (4) Process Data in Buffer ...

Say P = time to process 1 block

R = time to read in 1 block

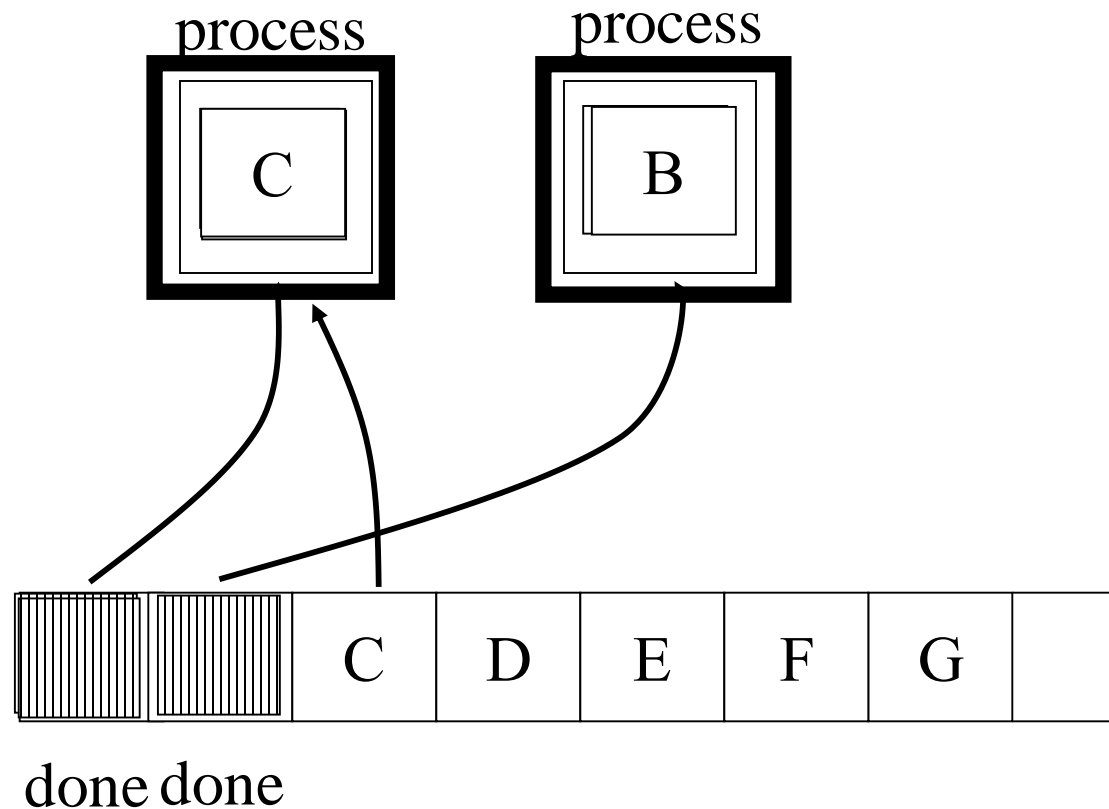
n = # blocks

Single buffer time = $n(P+R)$

Double Buffering

Memory:

Disk:



Say $R \geq P$

P = Processing time/block

R = IO time/block

n = # blocks

What is processing time?

- Double buffering time = nR
- Single buffering time = $n(R+P)$

RAID Technology

Parallelizing Disk Access using RAID Technology

- **RAID: Redundant Arrays of Independent Disks**
 - an array of independent disks **acting as a single higher-performance logical disk**, providing:
 - **high capacity** and **high speed** by using multiple disks in parallel,
 - **high reliability** by storing data redundantly, so that data can be recovered even if a disk fails
- The main goal of RAID is to **reduce the large speed gap between disks and the memory**

RAID goals



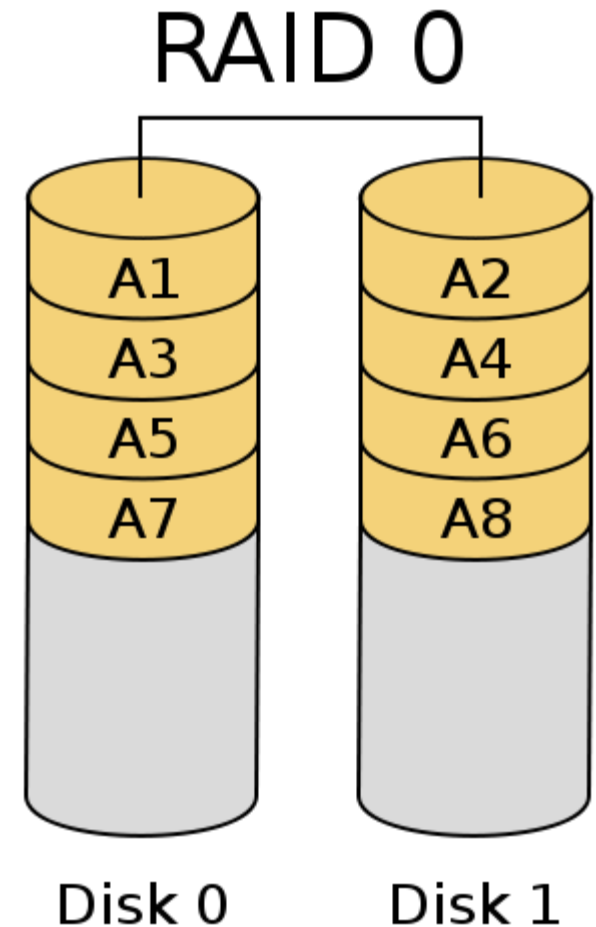
RAID systems seek :

- **to improve throughput** by a technique known as **striping**, in which a single file is spread over multiple disks.
 - **Parallelize** large accesses to reduce response time: multiple accesses to different parts of the same file can often be performed in parallel (assuming that the parts being accessed are on different disks).
- **to improve reliability** by **replication** of data, so that if a disk fails, the data it contained is available somewhere else.
 - => improve throughput for reads -if there are multiple copies of an item, then any copy can be read.
 - but creates an issue on write though - since all copies must be updated.

RAID 0 - a.k.a. Striping

- Requires two or more disks
- No lost drive space due to striping
- Fastest read and write performance.
- Raid level 0 has no redundant data and hence has the best write performance at the risk of data loss
 - Offers no data protection.
- The more disks, the more risk.

Used in high-performance applications where data loss is not critical

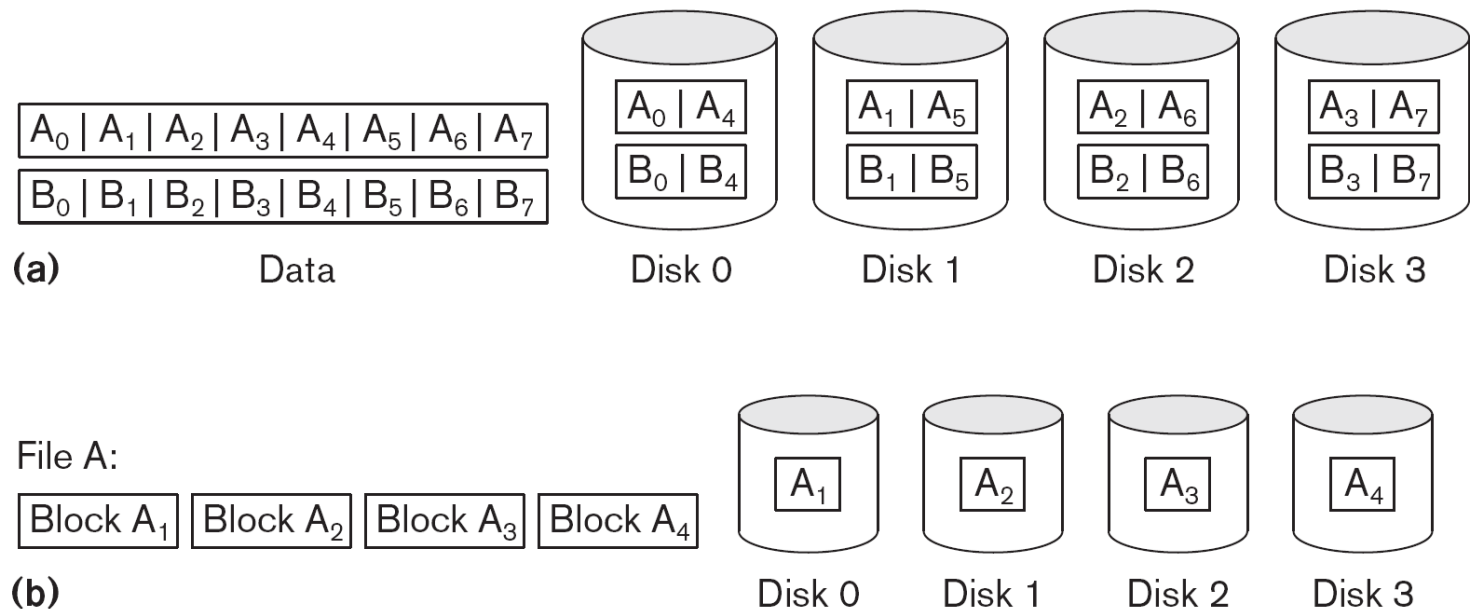


Data striping

- RAID uses a concept called **data striping** = distribute blocks over n disks in a round robin fashion.
 - Make disks appear as a single large, fast disk.
 - Requests for different blocks can run in parallel if the blocks reside on different disks

Figure 17.13

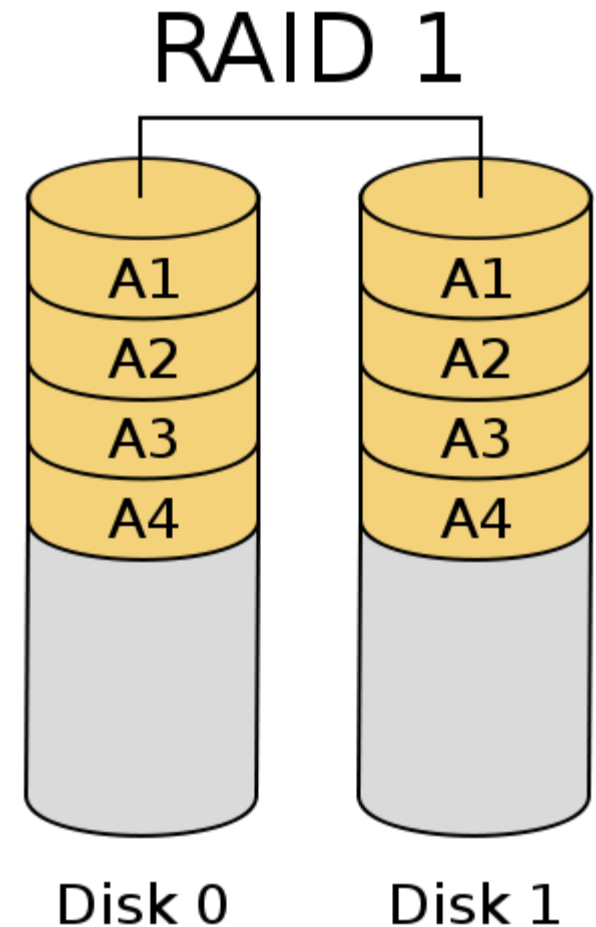
Striping of data across multiple disks.
(a) Bit-level striping across four disks.
(b) Block-level striping across four disks.



RAID 1 - a.k.a. Mirroring

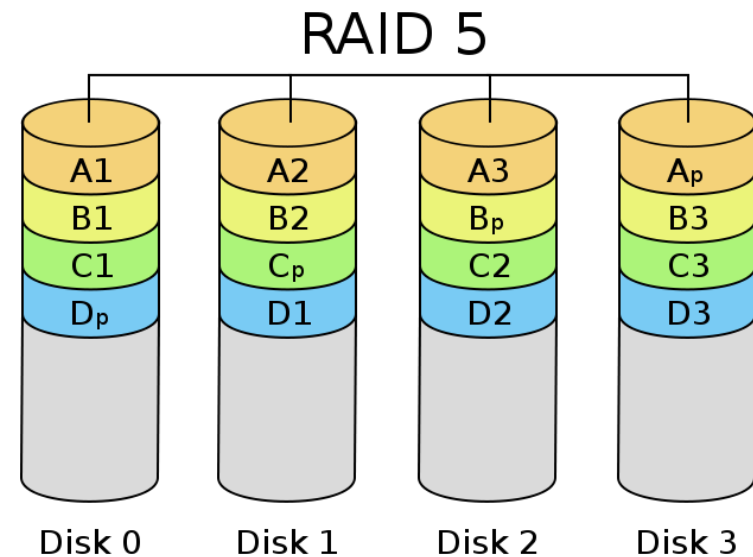
- Raid level 1 uses mirrored disks
- Write speed of one disk
- Read speed of two disks
- Capacity is equal to the size of one

Popular for applications such as storing log files in a database system.



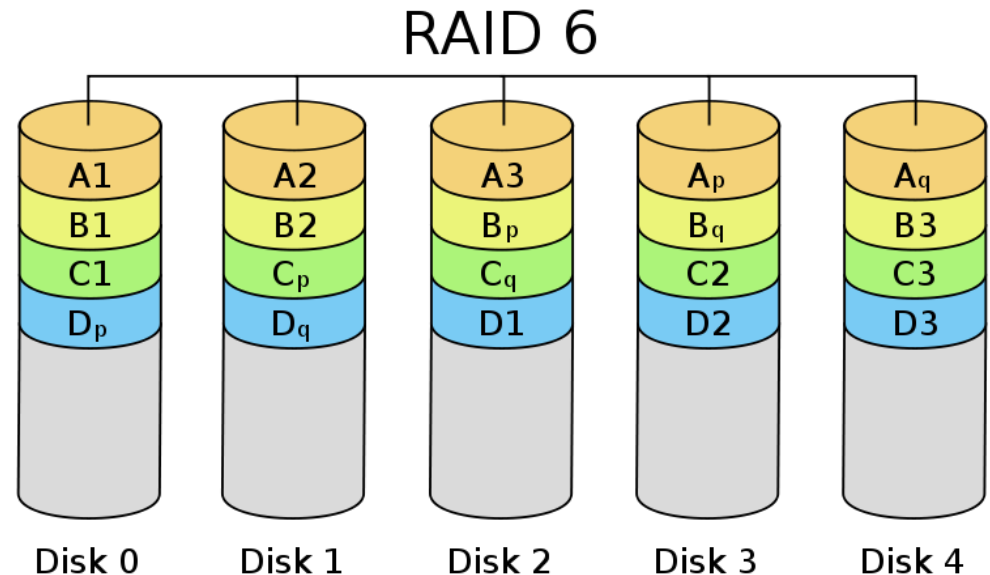
RAID 5 - Striping with Distributed Parity

- Considered best compromise between speed and storage efficiency:
 - Good performance (as blocks are striped) but slower writes due to parity
 - Good redundancy (distributed parity)
- Requires 3 or more drives
- Stripe across all drives with **parity**
- Can loose 1 drive and still function
- Capacity is **$n-1$** where **n** is number of drives in array

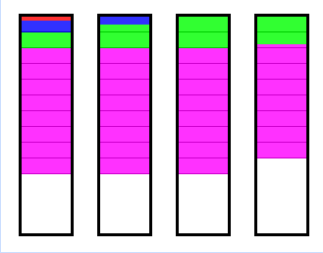
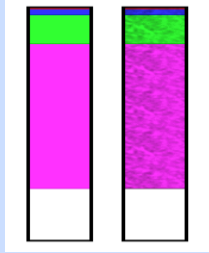
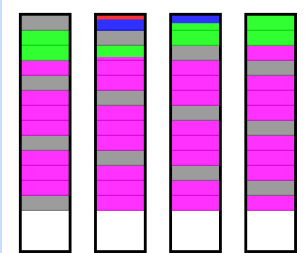
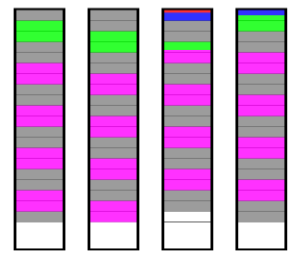


RAID 6 - RAID 5 on Steroids

- 4 or more disk
- Is a stripe with two parity drives
- Can loose two drives and still function
- Capacity is $n-2$ where n is number of drives in array
- Protect against up to two disk failures by using just two redundant disks



Comparison of Single RAID Levels

	RAID 0	RAID 1	RAID 5	RAID 6
Diagram				
Description	Striping	Mirroring	Striping with Parity	Striping with Dual Parity
Minimum Disks	2	2	3	4
Array Capacity	No. of Drives x Drive Capacity	Drive Capacity	(No. of Drives - 1) x Drive Capacity	(No. of Drives - 2) x Drive Capacity

Comparison of Single RAID Levels

	RAID 0	RAID 1	RAID 5	RAID 6
Storage Efficiency	100%	50%	(Num of drives – 1) / Num of drives	(Num of drives – 2) / Num of drives
Fault Tolerance	None	1 Drive failure	1 Drive failure	2 Drive failures
High Availability	None	Good	Good	Very Good
Degradation during <u>rebuild</u>	NA	<ul style="list-style-type: none"> • Slight degradation • Rebuilds very fast 	<ul style="list-style-type: none"> • High degradation • Slow Rebuild (due to write penalty of parity) 	<ul style="list-style-type: none"> • Very High degradation • Very Slow Rebuild (due to write penalty of dual parity)

Understanding the Parity

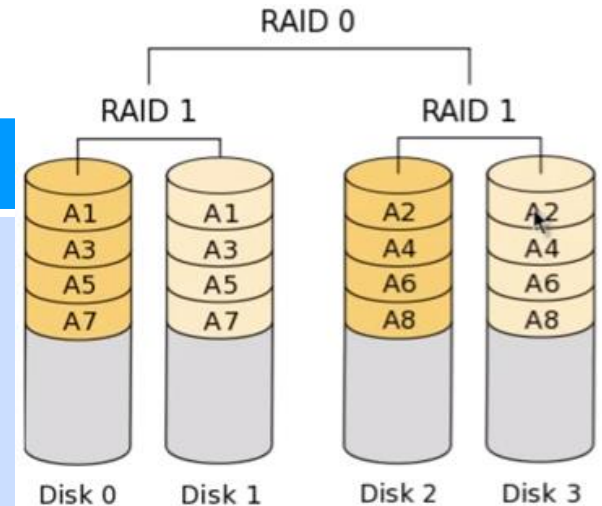
- RAID 5 and RAID 6 store parity information against data for rebuild
- Parity can be calculated using a simple XOR
- eg– “ABCDEFGH IJKL” on a 4 disk RAID 5 array

Disk 1	Disk 2	Disk 3	Disk 4
A (01000001)	B (01000010)	C (01000011)	{P – 01000000}
Parity {P}	D	E	F
G	Parity {P}	H	I
J	K	Parity {P}	L

- If Disk 2 fails then the data “B” can be recalculated as
(01000001 XOR 01000011 XOR 01000000) => 01000010 => B
- More info @ <http://www.youtube.com/watch?v=LTq4pGZtzho>

RAID 10 a.k.a. 1+0

Diagram



Description

Mirroring then Striping

Minimum Disks

Even number > 4

Maximum Disks

Controller Dependant

Array Capacity

$(\text{Size of Drive}) * (\text{Number of Drives}) / 2$

Storage Efficiency

50%

Fault Tolerance

Multiple drive failure as long as 2 drives from same RAID 1 set do not fail

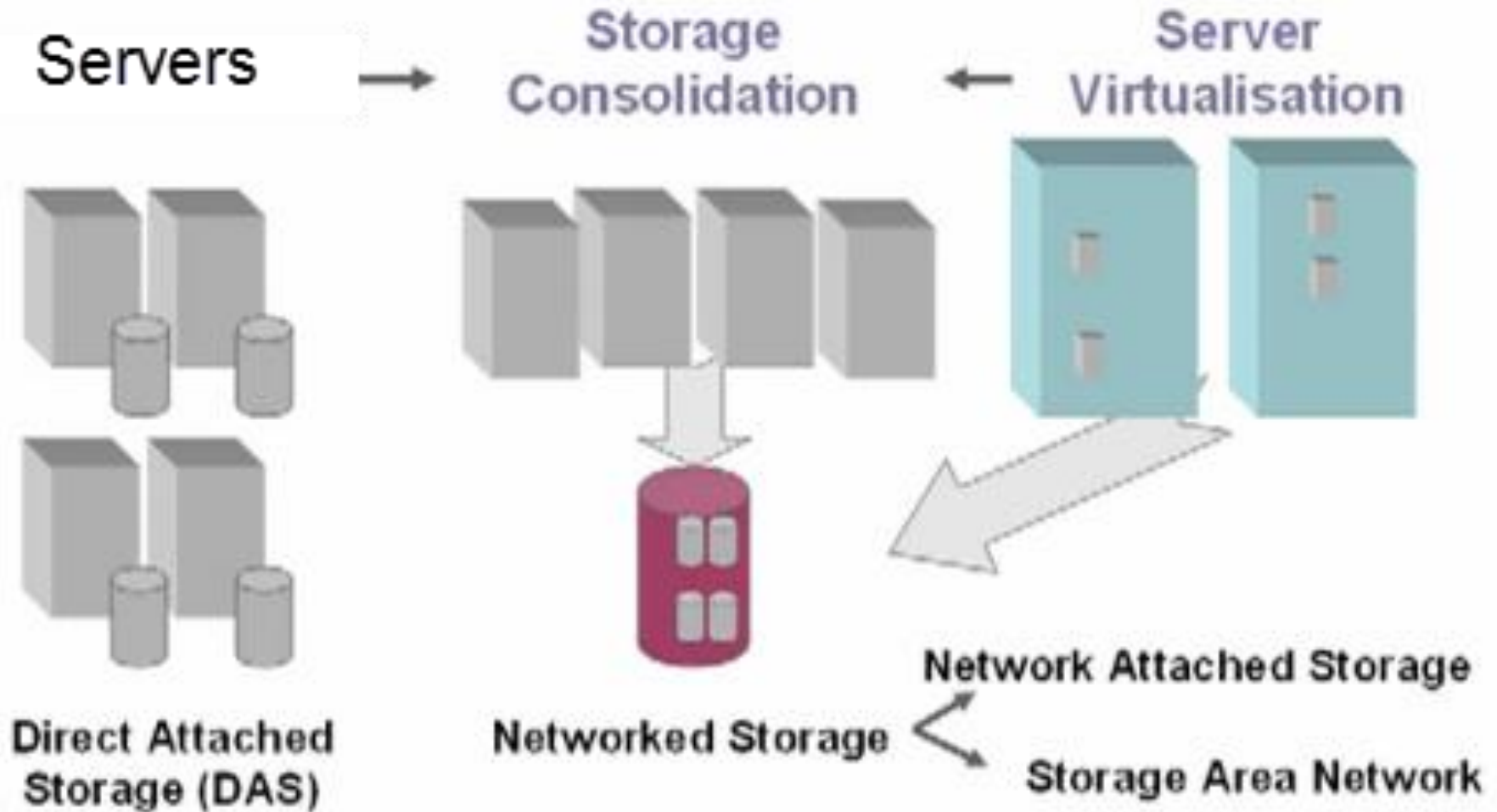
High Availability

Excellent

Choice of RAID Level

- Factors in choosing RAID level
 - Monetary cost
 - Performance: Number of I/O operations per second, and bandwidth during normal operation
 - Performance during failure
 - Performance during rebuild of failed disk
- RAID 0 is used only when data safety is not important
 - E.g. data can be recovered quickly from other sources
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

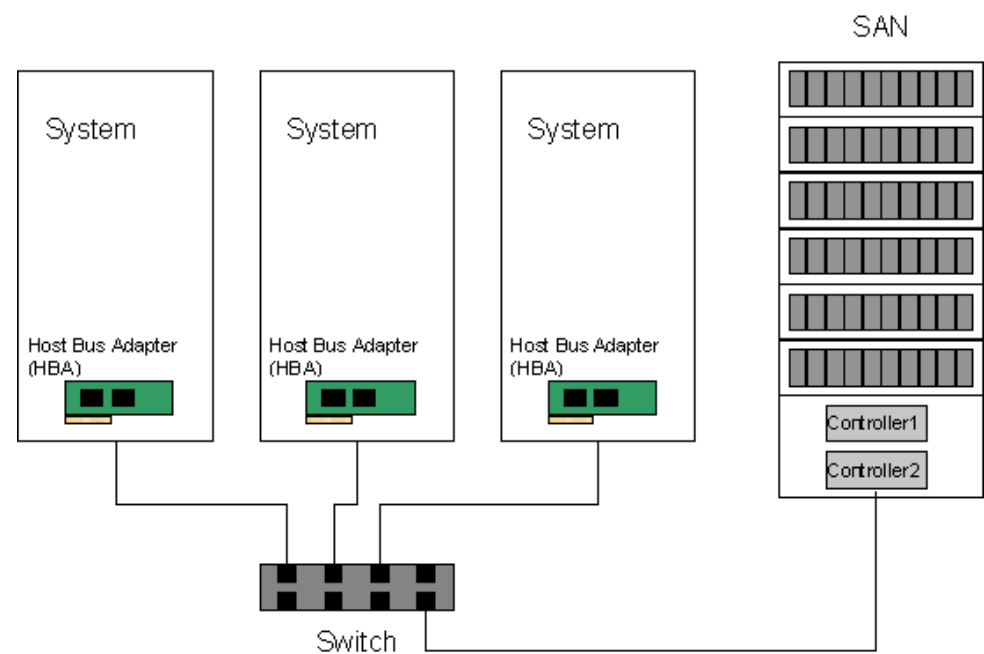
Networked Storage



Source: <http://www.youtube.com/watch?v=2T99tW1KEMc>

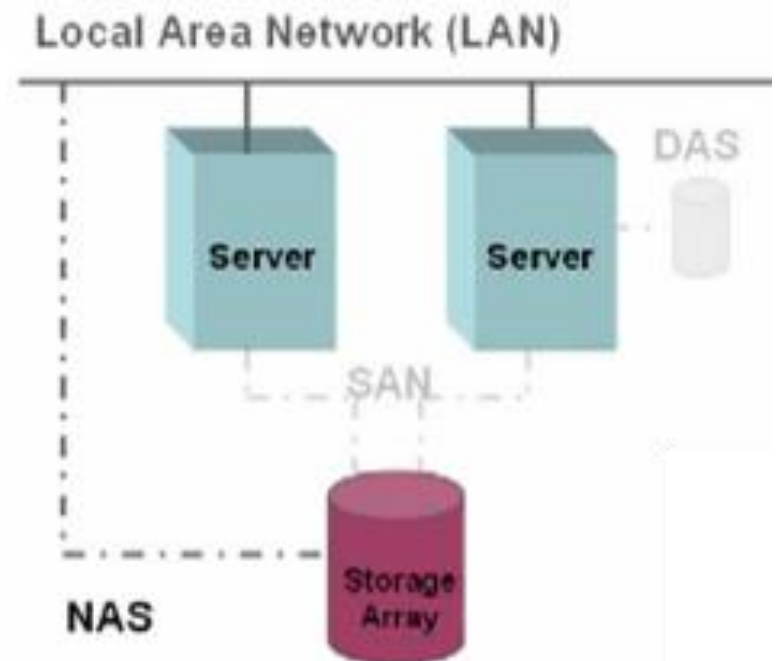
Storage Area Network (SAN)

- Online storage peripherals are configured as nodes on a **high-speed network** and can be attached and detached from servers in a very flexible manner
- Servers see SAN as a virtual drives
- **Dedicated access** - each part of the SAN is dedicated to each server
- **Block based storage**



Network Attached Storage (NAS)

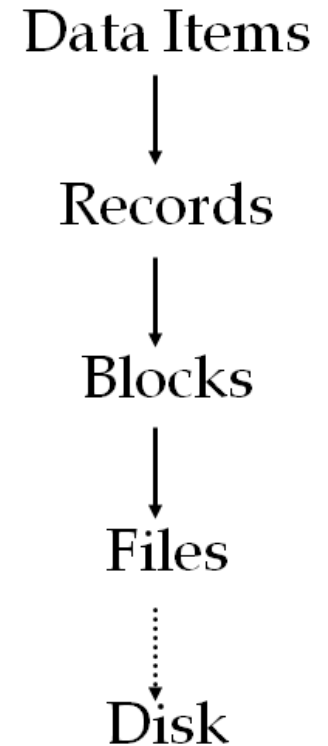
- File Server optimized to serve files over the main LAN (OS dedicated to file system)
- File based storage
- Servers see NAS as a Network Share (need to map it to a drive)
- Suitable for sharing files



Database File Organization

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records*. A record is a sequence of fields.
 - Each file consists of one or more blocks on a disk
- A **file organization** is a method of arranging records in a file
 - File organizations make some operations more efficient, and other operations less efficient



DB Files Organization

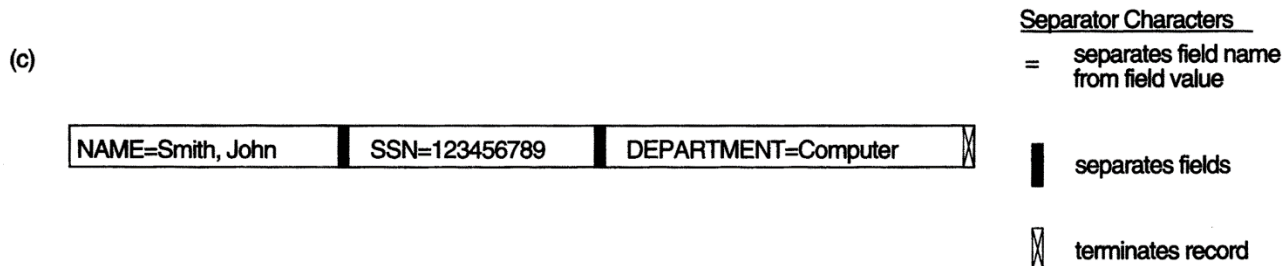
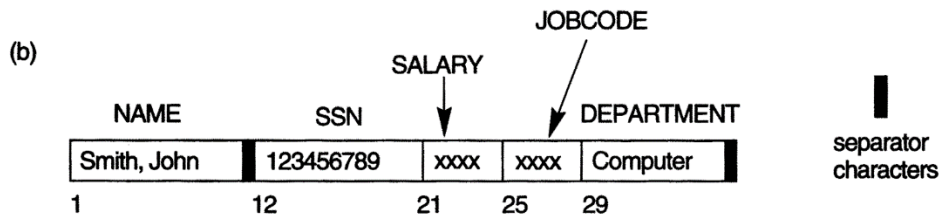
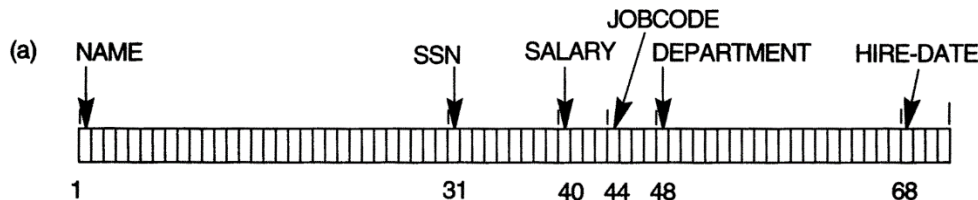
- Some DBMS store their data in a collection of files
 - perhaps one for each higher-level entity plus additional files needed by the implementation.
 - Example: MySQL: a database is represented by a directory, and each table is stored in a file whose name is the same as the name of the table.
- Other DBMS allocate a single, very large file from the host operating system and then build their own file system within it.
 - Example: db2 stores each database within one or more large files on the disk. Each file may contain any number of tables, indices etc.

Placing File Records on Disk

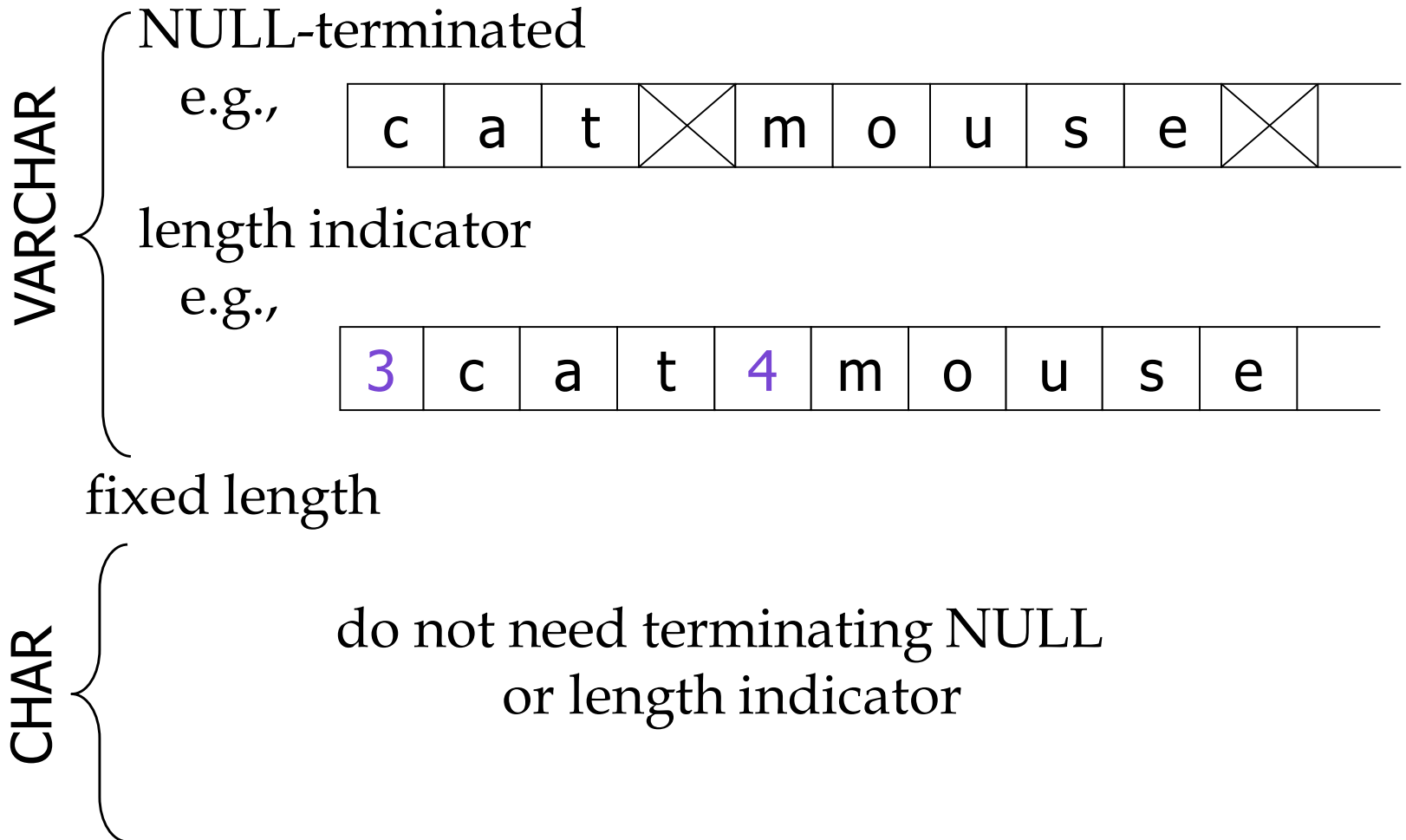
- Records maybe **fixed** and **variable length** records:
 - For fixed-length records, no need to separate
 - For variable-length records use special marker or store record lengths (or offsets) within each record or in block header
- Records contain fields which have values of a particular type (e.g., amount, date, time, age)
- Fields themselves may be fixed length or variable length
- Variable length fields can be mixed into one record
 - separator characters or length fields are needed so that the record can be “parsed”

Placing File Records on Disk

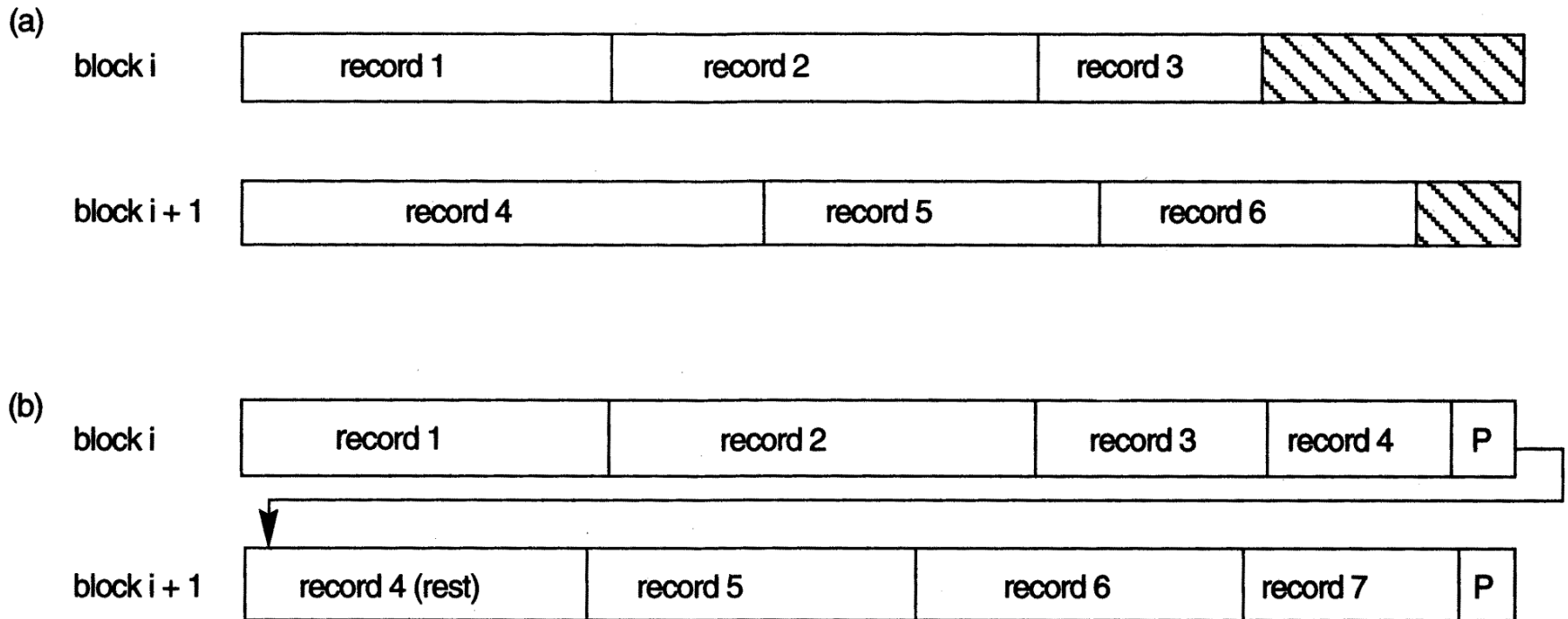
- (a) A fixed-length record with six fields and size of 71 bytes
- (b) A record with variable-length fields and fixed-length fields
- (c) A variable-field record with three types of separator characters



Handling String of characters



Unspanned or spanned records



- Spanned records split into fragments that are distributed over multiple blocks

Unordered Files

- Also called a **heap** or a **pile** file.
- A **linear search** through the file records is necessary to search for a record
 - This requires reading and searching half the file blocks on the average, and is hence quite expensive
- Record insertion is quite efficient.
 - New records are inserted at the end of the file
- Reading the records in order of a particular field requires sorting the file records.
- Deleted rows create gaps in file
 - File must be periodically compacted to recover space

Ordered Files

- Also called a **sequential** file
- File records are **kept sorted** by the values of an *ordering field*
- Insertion is expensive: records must be inserted in the correct order
 - It is common to keep a separate unordered *overflow* file for new records to improve insertion efficiency; this is periodically merged with the main ordered file
- A **binary search** can be used to search for a record on its *ordering field* value.
 - This requires reading and searching \log_2 of the file blocks on the average, an improvement over linear search
- Reading the records in order of the ordering field is quite efficient
- Sorted files may need to be periodically re-ordered
 - Pages should be maintained in sequence to allow for more efficient disk access

Record Insertions

- If record order does not matter, just append record to the end of the file or find a block of that file with empty space.
- If order does matter, may be more complicated.
 - First locate corresponding block
 - If block has enough space left, insert record there and rearrange records in block.
 - If block does not have enough space left either:
 - find space in nearby block - predecessor or successor in sort order
 - create overflow block - and link it into chain of blocks

Record Deletions

- Either move around records within a block or maintain available-space list.
- Another technique is to use a **deletion marker** to flag deleted records
- Periodic DB files reorganization can be used to reclaim the unused space of deleted records.
 - **Blocks should be maintained in sequence to allow for more efficient disk access**



Blocking Factor

- Blocking factor (**bfr**) refers to the **number of records per block**
- Suppose the size of the block is **B** bytes, and a file contains fixed-length records of size **R** bytes
If **$B \geq R$** , then we can allocate **$\text{bfr} = \lfloor (B/R) \rfloor$** records into one block.
 - where $\lfloor (x) \rfloor$ is the floor function which rounds the value x down to the next integer.
- In general, **R** may not divide **B** exactly, so there will be some left-over spaces in each block equal to **$B - (\text{bfr} * R)$** bytes

Blocking Factor (cont.)

- The unspanned organization is useful for fixed-length records with a length $R \leq B$. For variable-length records, either a spanned or unspanned organization can be used.
- For variable-length records using spanned organization, each block may store a different number of records. In this case, the blocking factor **bfr** represents the average number of records per block.
- The number of blocks **b** needed to accommodate a file of **r** records:

$$\mathbf{b} = \lceil (r/bfr) \rceil \text{ blocks}$$

- where $\lceil (x) \rceil$ is the ceiling function which rounds the value x up to the nearest integer
- If the record size R is bigger than the block size B , the spanned organization has to be used

Summary

- I/O times dominate DBMS processing
- Techniques to improve I/O time:
 - Buffering
 - RAID Technology
 - Store related information on contiguous blocks
 - To keep good performance, the DBMS must occasionally rebuild the database files to merge in the overflow pages and reclaim unused blocks