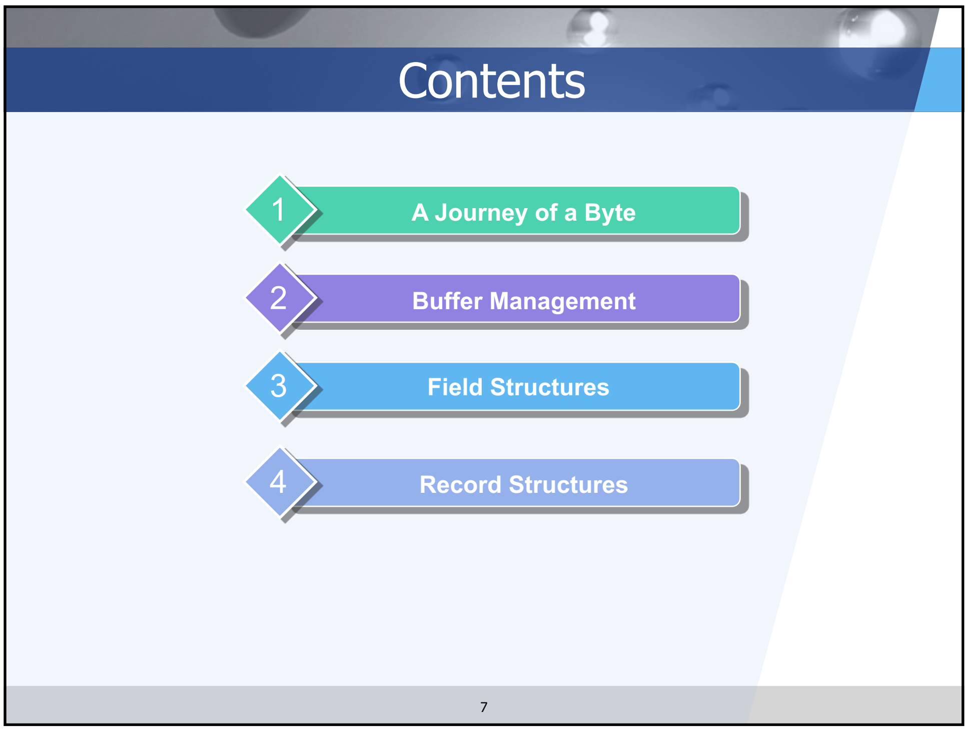
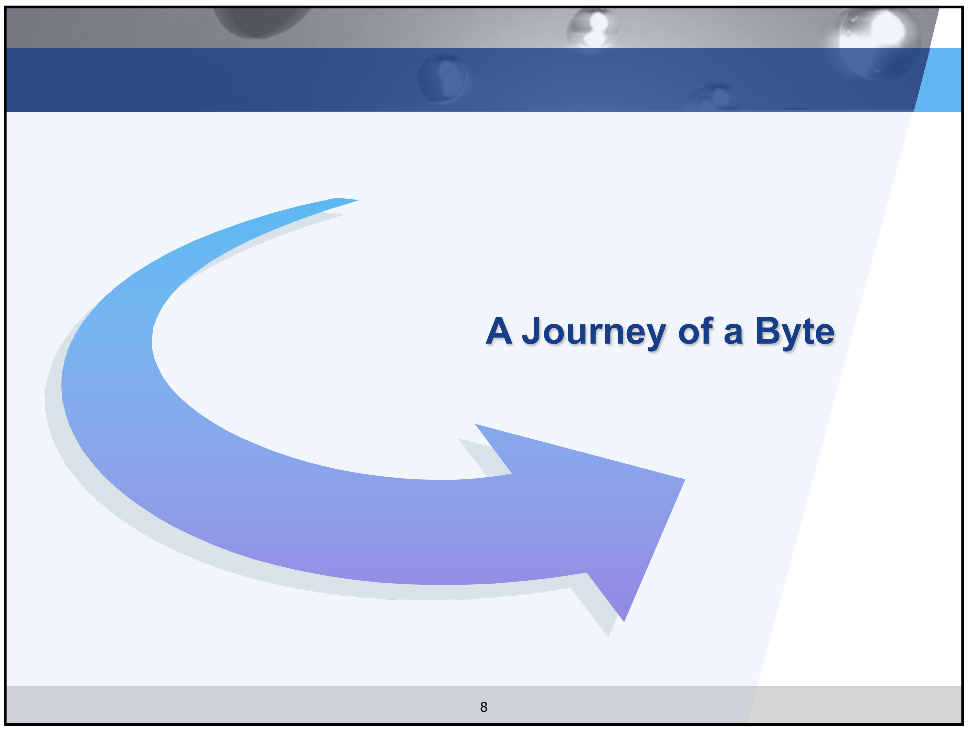


Contents

7

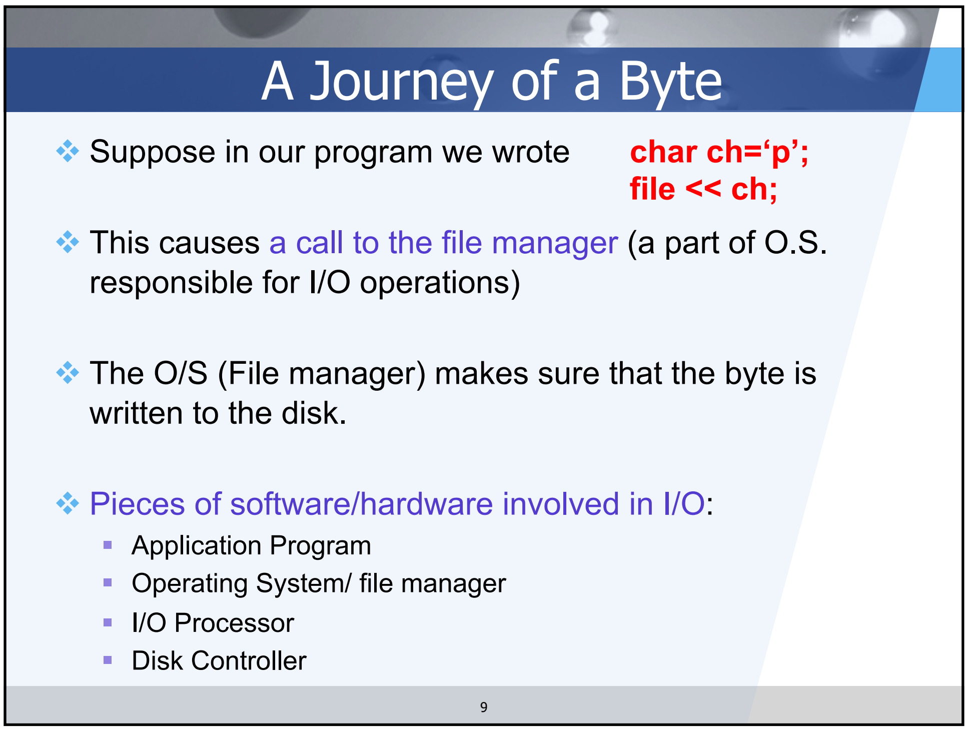
7



A Journey of a Byte

8

8

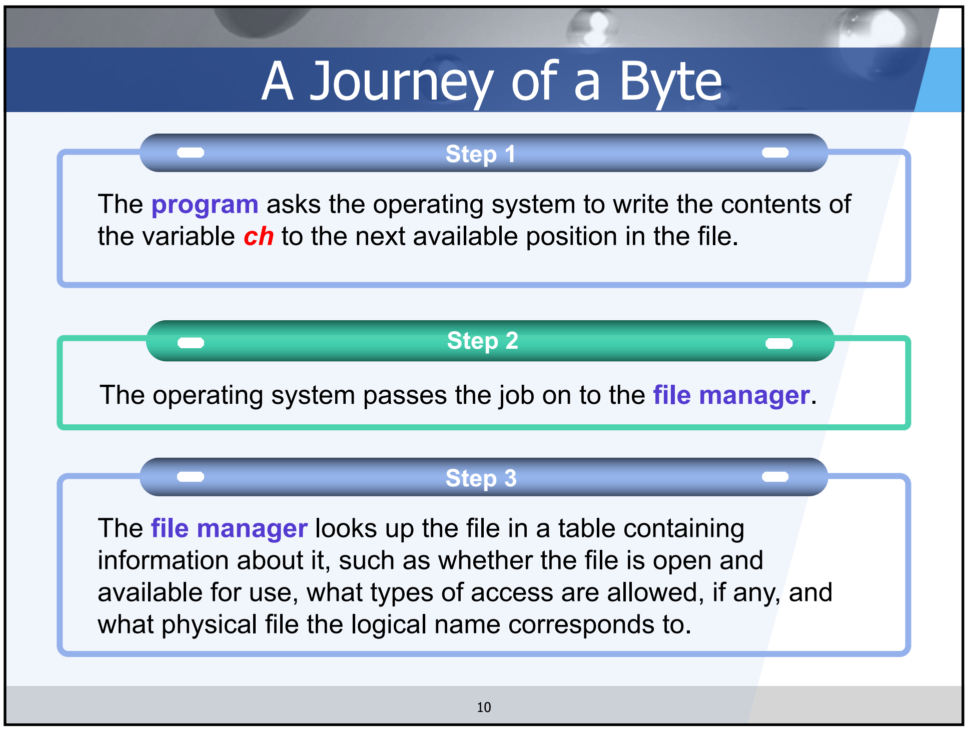


A Journey of a Byte

- Suppose in our program we wrote **char ch='p';** **file << ch;**
- This causes a call to the file manager (a part of O.S. responsible for I/O operations)
- The O/S (File manager) makes sure that the byte is written to the disk.
- Pieces of software/hardware involved in I/O:
  - Application Program
  - Operating System/ file manager
  - I/O Processor
  - Disk Controller

9

9



A Journey of a Byte

Step 1

The **program** asks the operating system to write the contents of the variable ***ch*** to the next available position in the file.

Step 2

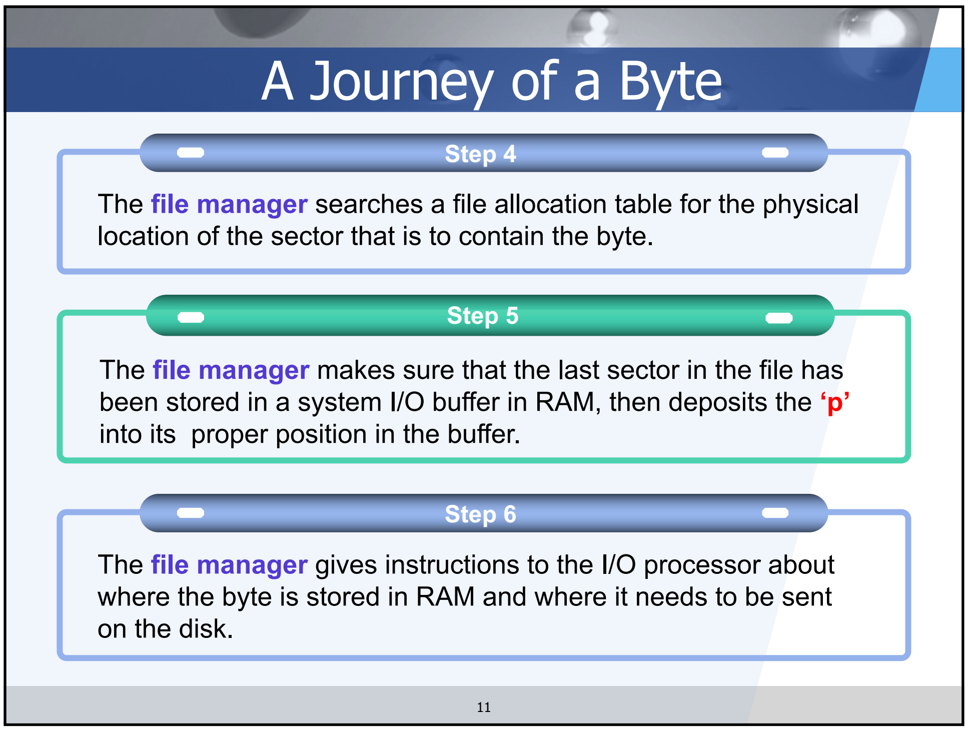
The operating system passes the job on to the **file manager**.

Step 3

The **file manager** looks up the file in a table containing information about it, such as whether the file is open and available for use, what types of access are allowed, if any, and what physical file the logical name corresponds to.

10

10



A Journey of a Byte

Step 4

The **file manager** searches a file allocation table for the physical location of the sector that is to contain the byte.

Step 5

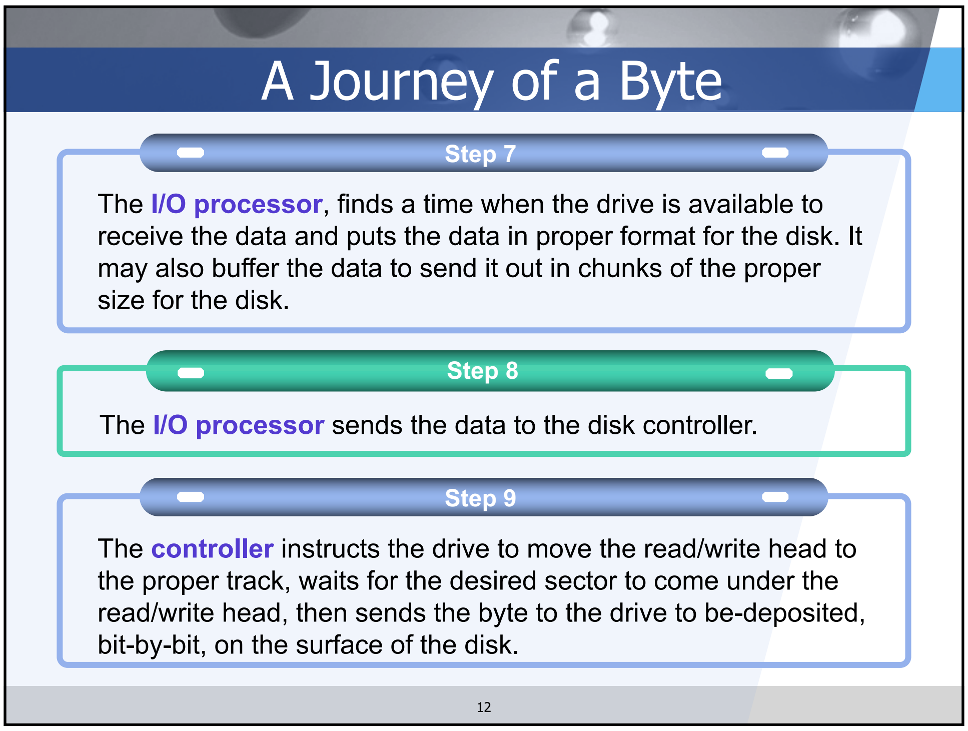
The **file manager** makes sure that the last sector in the file has been stored in a system I/O buffer in RAM, then deposits the **'p'** into its proper position in the buffer.

Step 6

The **file manager** gives instructions to the I/O processor about where the byte is stored in RAM and where it needs to be sent on the disk.

11

11



A Journey of a Byte

Step 7

The **I/O processor**, finds a time when the drive is available to receive the data and puts the data in proper format for the disk. It may also buffer the data to send it out in chunks of the proper size for the disk.

Step 8

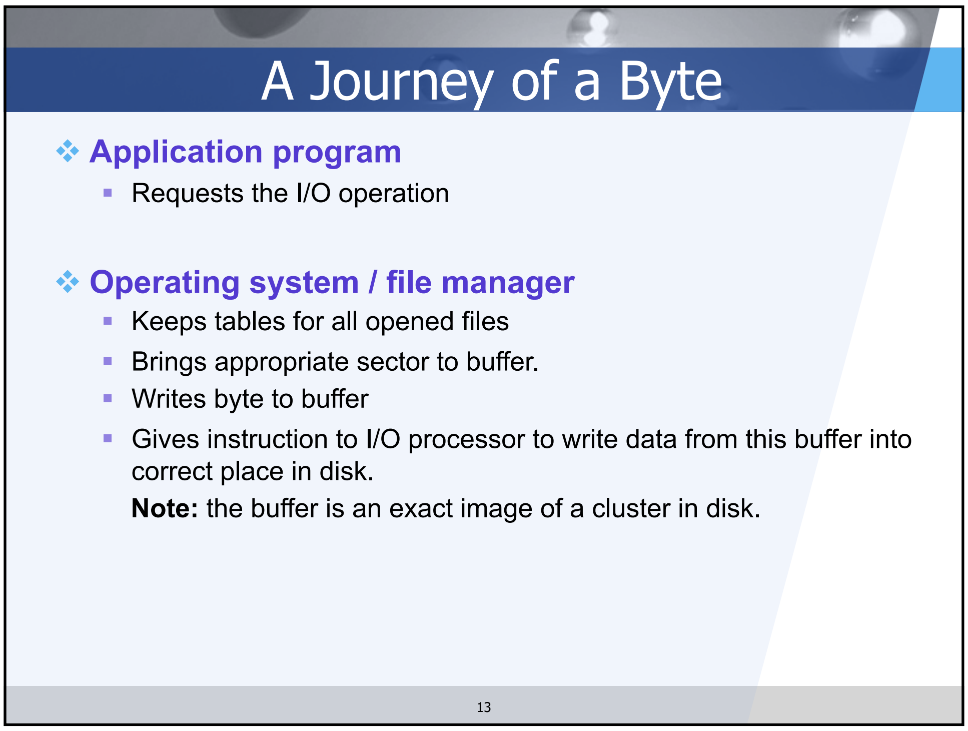
The **I/O processor** sends the data to the disk controller.

Step 9

The **controller** instructs the drive to move the read/write head to the proper track, waits for the desired sector to come under the read/write head, then sends the byte to the drive to be-deposited, bit-by-bit, on the surface of the disk.

12

12



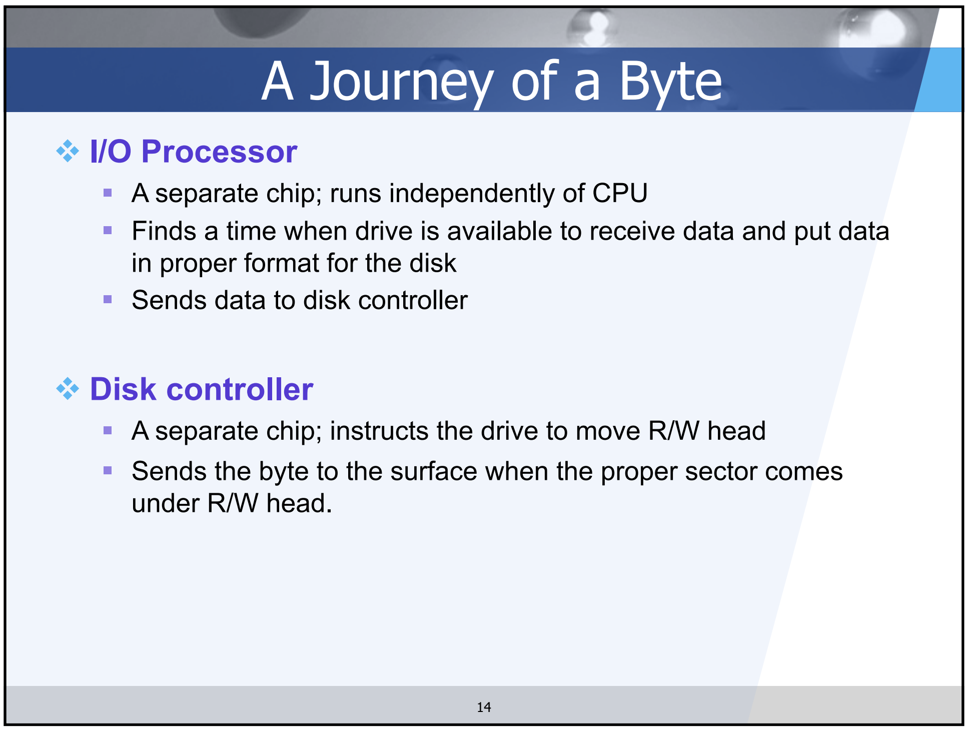
A Journey of a Byte

- **Application program**
  - Requests the I/O operation
- **Operating system / file manager**
  - Keeps tables for all opened files
  - Brings appropriate sector to buffer.
  - Writes byte to buffer
  - Gives instruction to I/O processor to write data from this buffer into correct place in disk.

    Note: the buffer is an exact image of a cluster in disk.

13

13

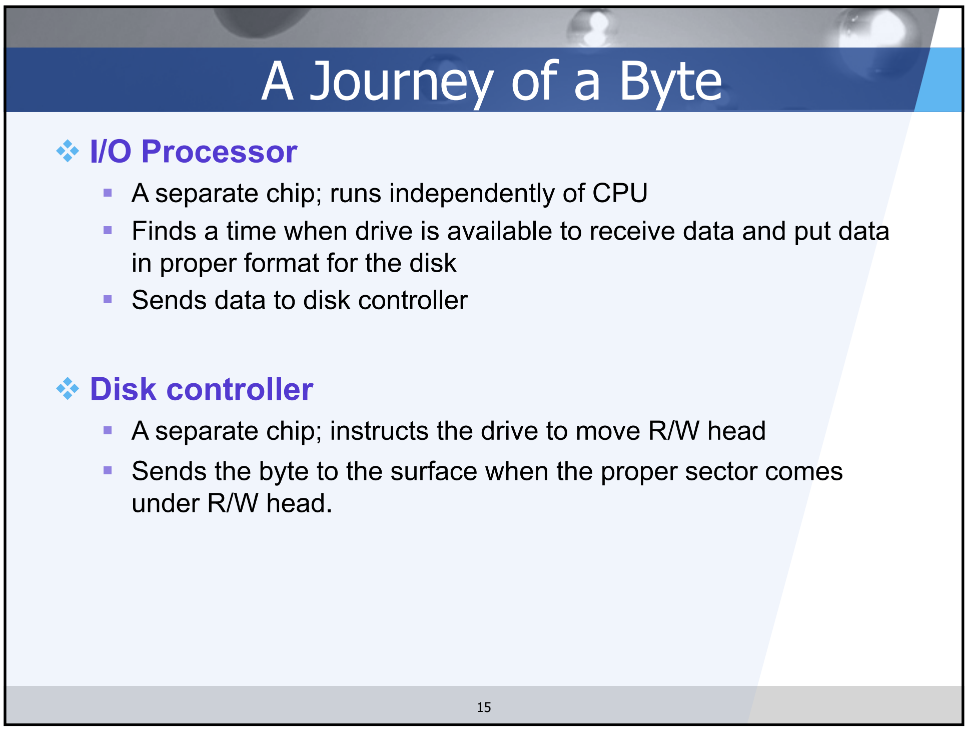


A Journey of a Byte

- **I/O Processor**
  - A separate chip; runs independently of CPU
  - Finds a time when drive is available to receive data and put data in proper format for the disk
  - Sends data to disk controller
- **Disk controller**
  - A separate chip; instructs the drive to move R/W head
  - Sends the byte to the surface when the proper sector comes under R/W head.

14

14

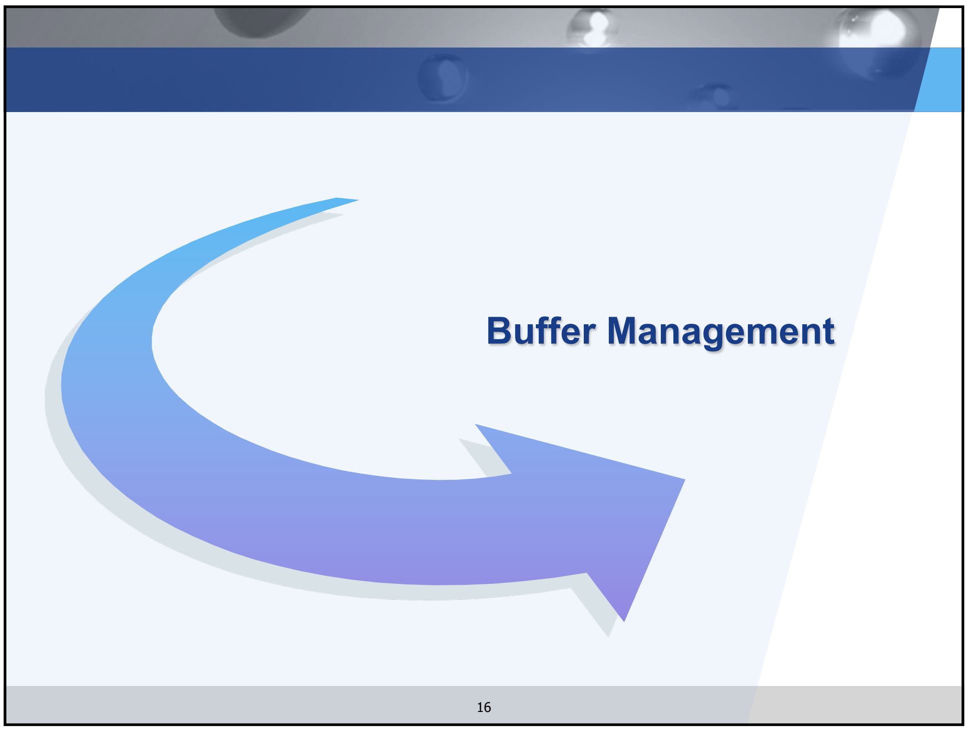


A Journey of a Byte

- **I/O Processor**
  - A separate chip; runs independently of CPU
  - Finds a time when drive is available to receive data and put data in proper format for the disk
  - Sends data to disk controller
- **Disk controller**
  - A separate chip; instructs the drive to move R/W head
  - Sends the byte to the surface when the proper sector comes under R/W head.

15

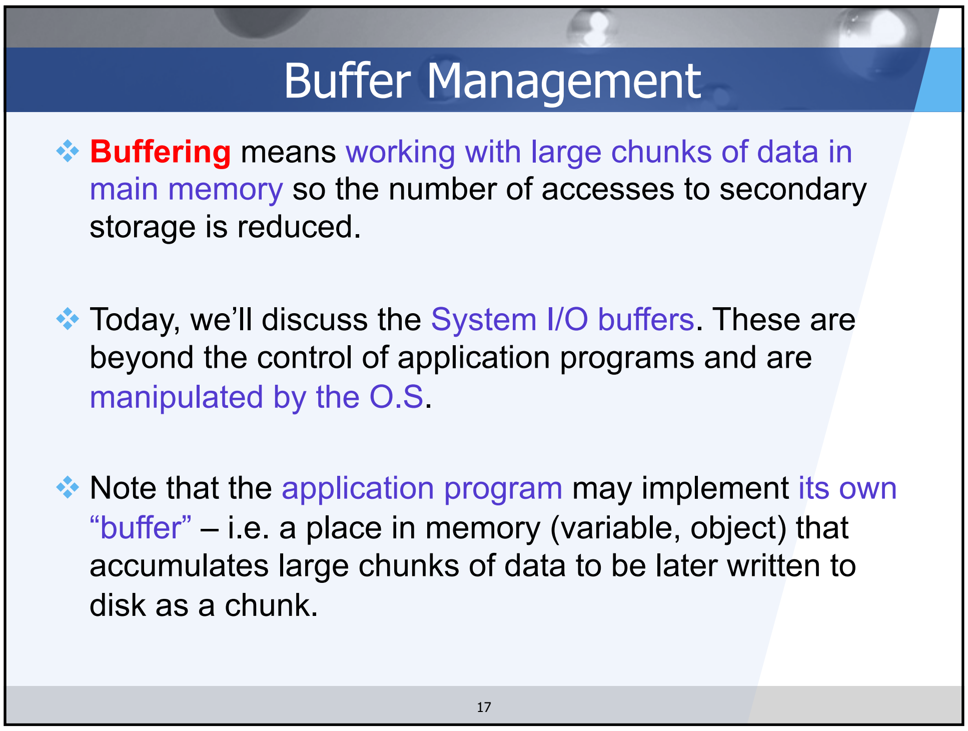
15



Buffer Management

16

16

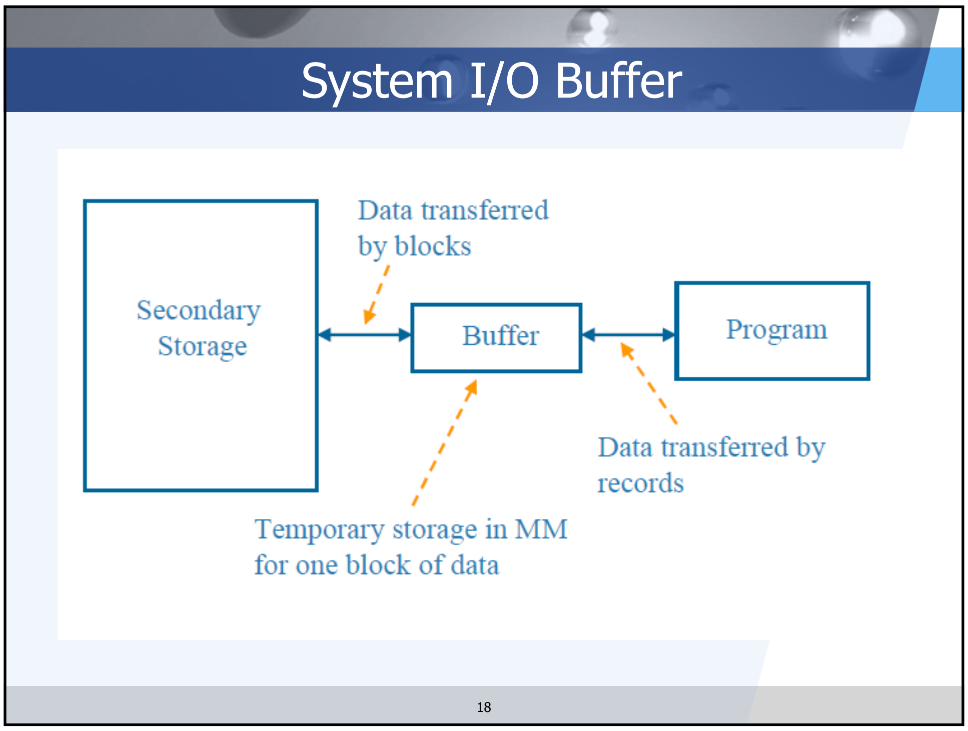


Buffer Management

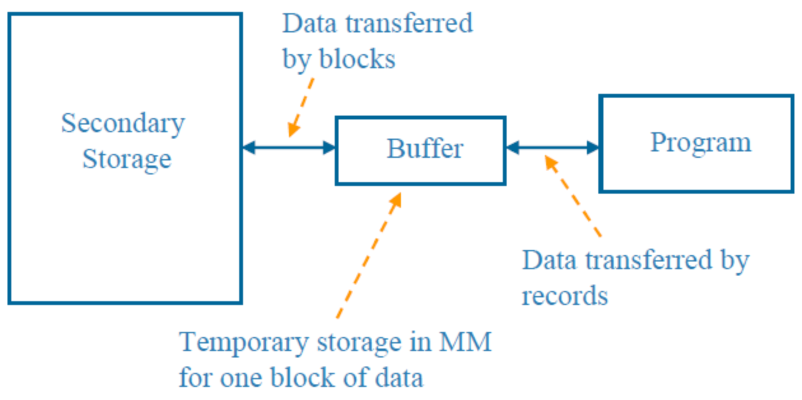
- **Buffering** means working with large chunks of data in main memory so the number of accesses to secondary storage is reduced.
- Today, we'll discuss the System I/O buffers. These are beyond the control of application programs and are manipulated by the O.S.
- Note that the application program may implement its own "buffer" – i.e. a place in memory (variable, object) that accumulates large chunks of data to be later written to disk as a chunk.

17

17

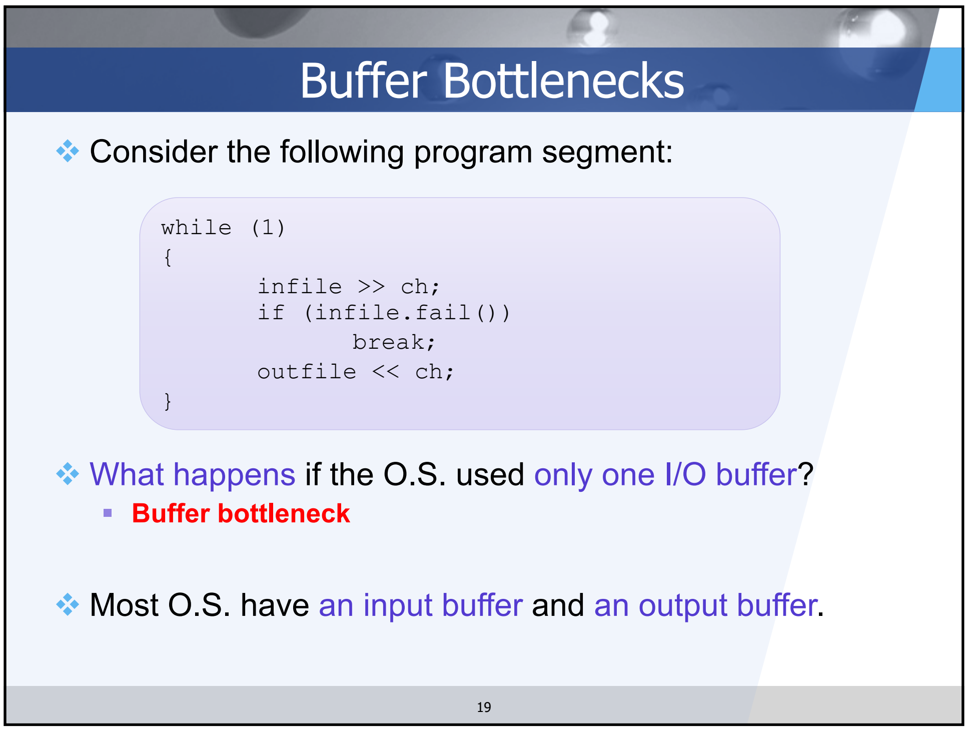


System I/O Buffer


Data transferred by blocks
Secondary Storage
Buffer
Program
Data transferred by records
Temporary storage in MM for one block of data

18

18



Buffer Bottlenecks

- Consider the following program segment:

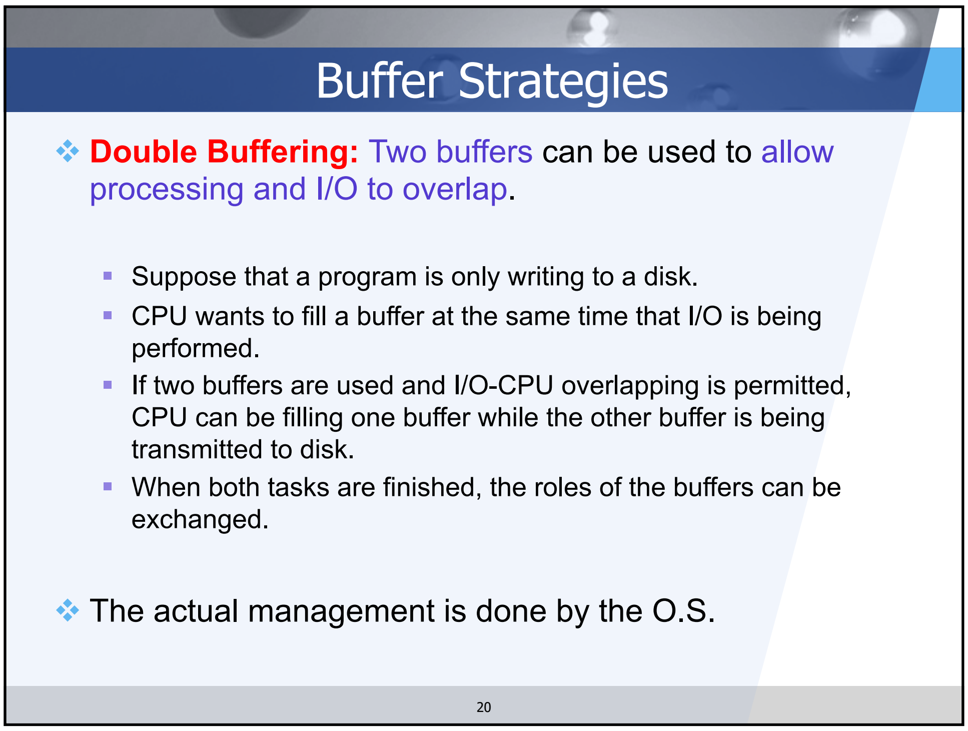
```
while (1)
{
        infile >> ch;
        if (infile.fail())
                break;
        outfile << ch;
}
```

- What happens if the O.S. used only one I/O buffer?
  - **Buffer bottleneck**

- Most O.S. have an input buffer and an output buffer.

19

19



Buffer Strategies

- **Double Buffering:** Two buffers can be used to allow processing and I/O to overlap.

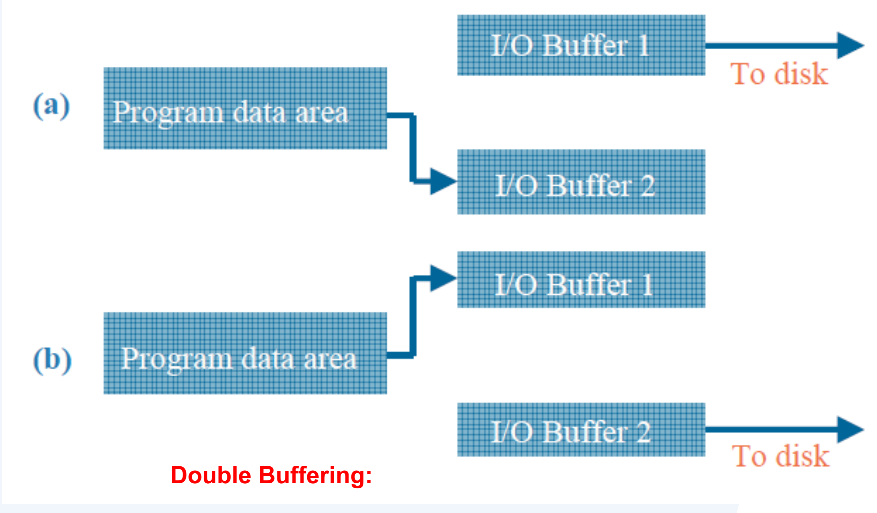
  - Suppose that a program is only writing to a disk.
  - CPU wants to fill a buffer at the same time that I/O is being performed.
  - If two buffers are used and I/O-CPU overlapping is permitted, CPU can be filling one buffer while the other buffer is being transmitted to disk.
  - When both tasks are finished, the roles of the buffers can be exchanged.

- The actual management is done by the O.S.

20

20

Buffer Strategies



Double Buffering:

21

21

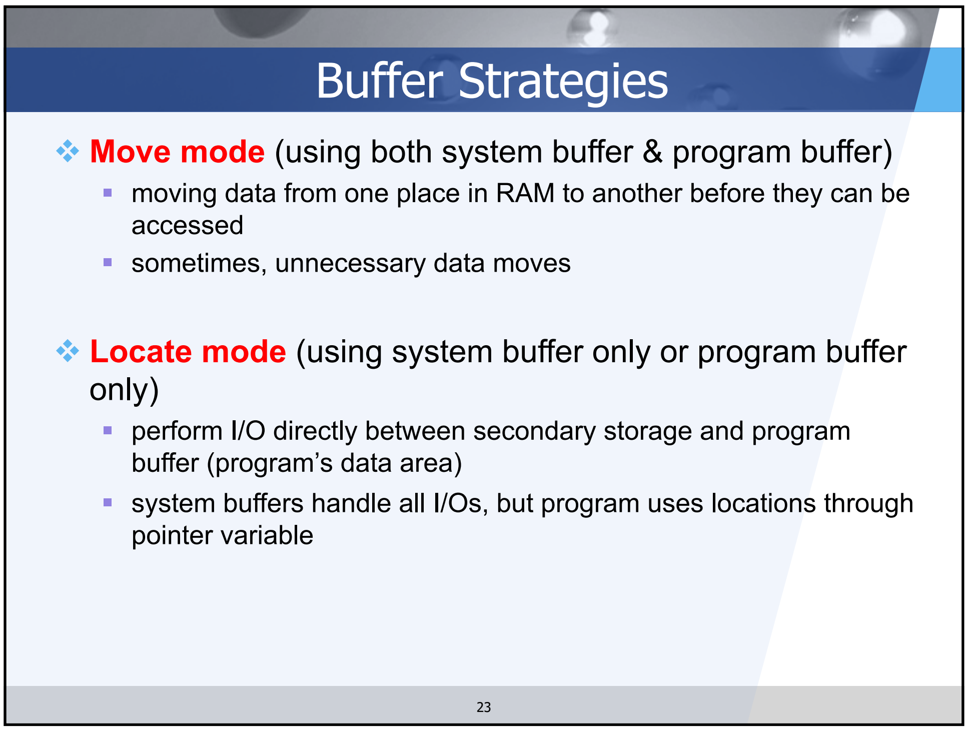
Buffer Strategies

- **Multiple Buffering:** instead of two buffers any number of buffers can be used to allow processing and I/O to overlap.

- **Buffer pooling:**
  - There is a pool of buffers.
  - When a request for a sector is received, O.S. first looks to see that sector is in some buffer.
  - If not there, it brings the sector to some free buffer. If no free buffer exists, it must choose an occupied buffer. Usually, LRU (Least Recently Used) strategy is used.

22

22



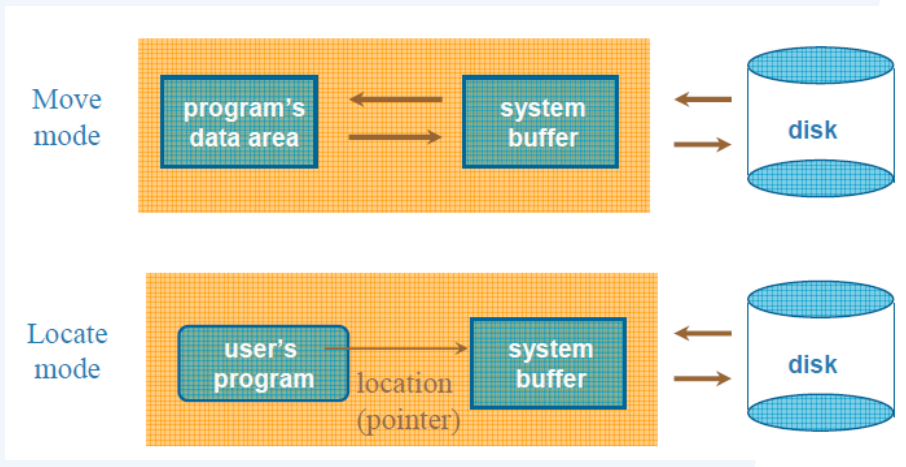
Buffer Strategies

- **Move mode** (using both system buffer & program buffer)
  - moving data from one place in RAM to another before they can be accessed
  - sometimes, unnecessary data moves
- **Locate mode** (using system buffer only or program buffer only)
  - perform I/O directly between secondary storage and program buffer (program's data area)
  - system buffers handle all I/Os, but program uses locations through pointer variable

23

23

Buffer Strategies



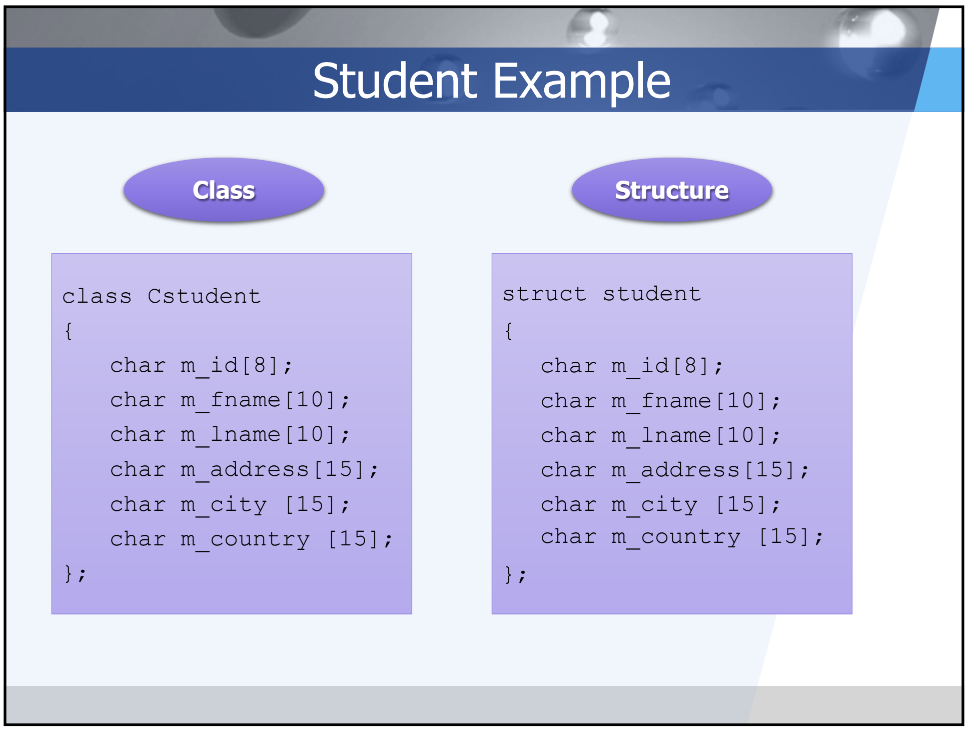
24

24

Fundamental File Structure Concepts

25

25



Student Example

Class

```
class Cstudent
{
    char m_id[8];
    char m_fname[10];
    char m_lname[10];
    char m_address[15];
    char m_city [15];
    char m_country [15];
};
```

Structure

```
struct student
{
   char m_id[8];
   char m_fname[10];
   char m_lname[10];
   char m_address[15];
   char m_city [15];
   char m_country [15];
};
```

26

File Types

- A file can be treated as:

1. a stream of bytes

```
1202345 Ahmed Hassan 1 Safa street Cairo Egypt 2349890 Mona
Hussein 42 El Thawra street Ismailia Egypt 1210322 Tamer Fouad
123 Ramsis street Cairo Egypt
```

2. a collection of records with fields

```
1202345 Ahmed Hassan 1 Safa street Cairo Egypt
2349890 Mona Hussein 42 El Thawra street Ismailia Egypt
1210322 Tamer Fouad 123 Ramsis street Cairo Egypt
```

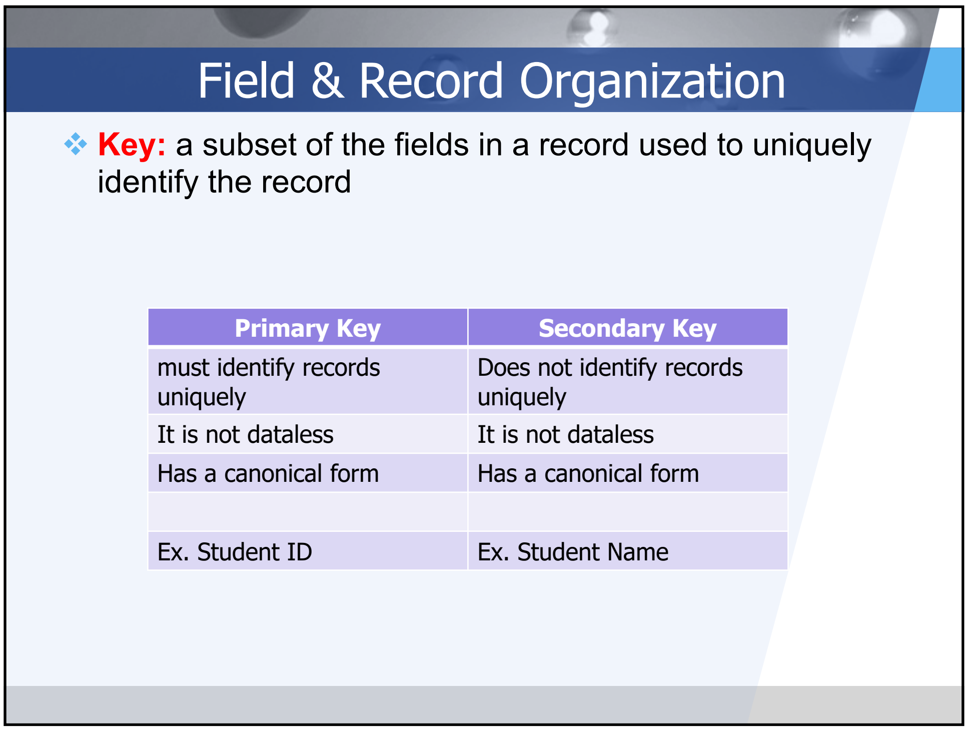
27

Field & Record Organization

- **Field:** a data value, smallest unit of data with logical meaning.
- **Record:** A group of fields that forms a logical unit.

Memory	File
object	record
member	field

28

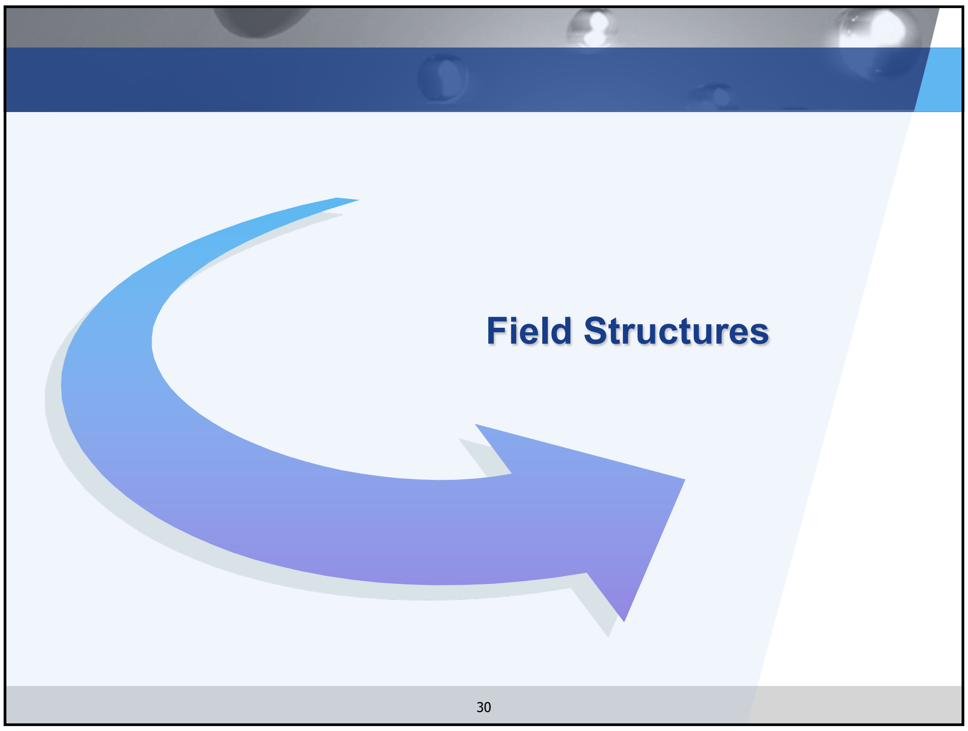


Field & Record Organization

- **Key:** a subset of the fields in a record used to uniquely identify the record

Primary Key	Secondary Key
must identify records uniquely	Does not identify records uniquely
It is not dataless	It is not dataless
Has a canonical form	Has a canonical form
Ex. Student ID	Ex. Student Name

29



Field Structures

30

30

Field structures

- Methods for organizing fields are:
  1. Fix the length of fields
  2. Begin each field with a length indicator
  3. Separate the fields with delimiters
  4. Use a “Keyword=Value” Expression to identify fields

31

1- Fix the Length of Fields

1202345	Ahmed	Hassan	1 safa street	Cairo	Egypt
2349890	Mona	Hussein	42 El Thawra street	Ismailia	Egypt
1210322	Tamer	Fouad	123 Ramsis street	Cairo	Egypt
8	10	10	15	15	15

- **Advantages:**
  - Easy to Read/Store
- **Disadvantages:**
  - Waste space with padding

32

2- Begin Each Field with a Length Indicator

071202345 05Ahmed06Hassan131 Safa street05Cairo05Egypt
072349890 04Mona07Hussein1942 El Thawra street08Ismailia05Egypt
071210322 05Tamer05Fouad17123 Ramsis street05Cairo05Egypt

❖ **Advantages:**
- Easy to jump ahead to the end of the field

❖ **Disadvantages:**
- Long fields require more than 1 byte to store length (Max is 255)

33

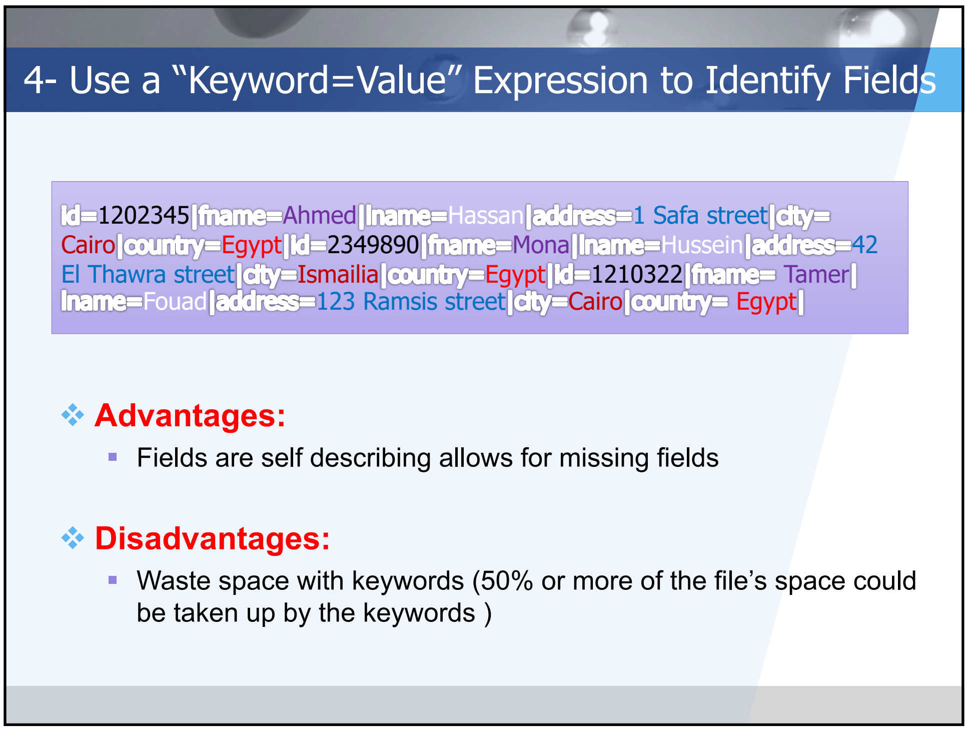
3- Separate the Fields with Delimiters

1202345|Ahmed|Hassan|1 Safa street|Cairo|Egypt|
2349890|Mona|Hussein|42 El Thawra street|Ismailia|Egypt|
1210322|Tamer|Fouad|123 Ramsis street|Cairo|Egypt|

❖ **Advantages:**
- May waste less space than with length-based

❖ **Disadvantages:**
- Have to check every byte of field against the delimiter

34



4- Use a "Keyword=Value" Expression to Identify Fields

Id=1202345|fname=Ahmed|lname=Hassan|address=1 Safa street|city=Cairo|country=Egypt|Id=2349890|fname=Mona|lname=Hussein|address=42 El Thawra street|city=Ismailia|country=Egypt|Id=1210322|fname= Tamer|lname=Fouad|address=123 Ramsis street|city=Cairo|country= Egypt|

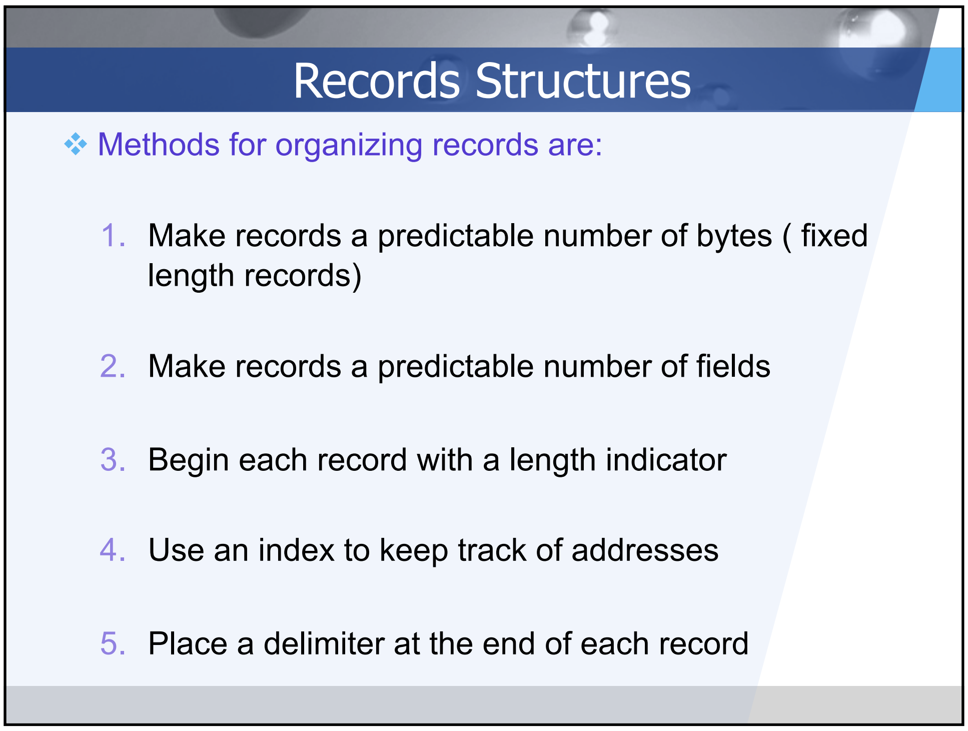
- **Advantages:**
  - Fields are self describing allows for missing fields
- **Disadvantages:**
  - Waste space with keywords (50% or more of the file's space could be taken up by the keywords)

35

Record Structures

36

36

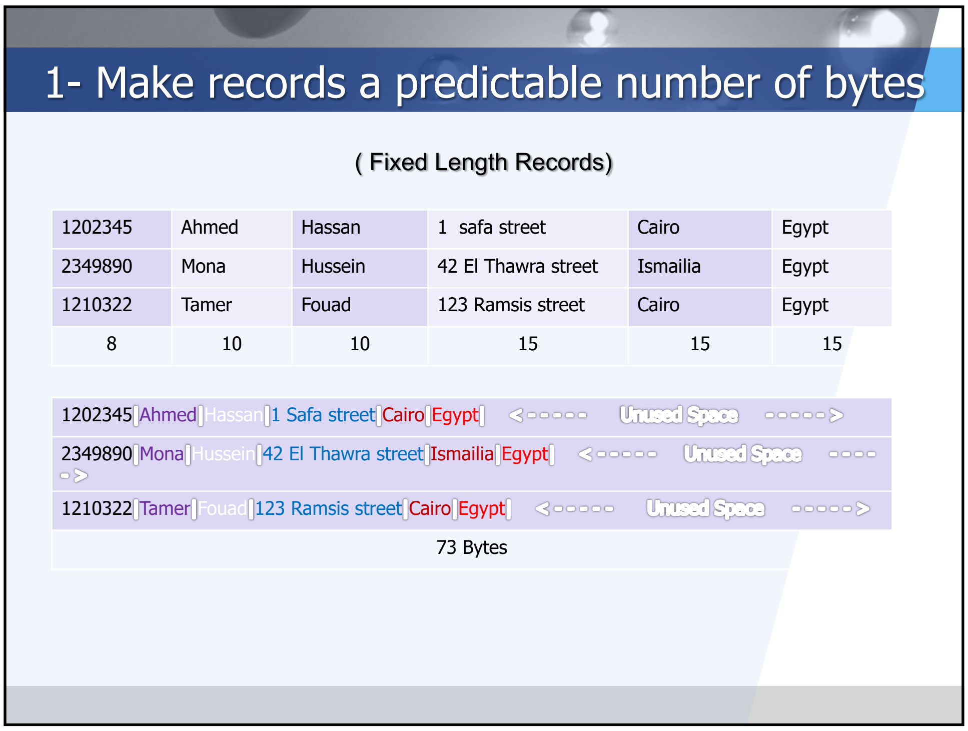


Records Structures

❖ Methods for organizing records are:

1. Make records a predictable number of bytes (fixed length records)
2. Make records a predictable number of fields
3. Begin each record with a length indicator
4. Use an index to keep track of addresses
5. Place a delimiter at the end of each record

37



1- Make records a predictable number of bytes

(Fixed Length Records)

1202345	Ahmed	Hassan	1 safa street	Cairo	Egypt
2349890	Mona	Hussein	42 El Thawra street	Ismailia	Egypt
1210322	Tamer	Fouad	123 Ramsis street	Cairo	Egypt
8	10	10	15	15	15

1202345|Ahmed|Hassan|1 Safa street|Cairo|Egypt| <------ Unused Space ------>

2349890|Mona|Hussein|42 El Thawra street|Ismailia|Egypt| <------ Unused Space ------>

1210322|Tamer|Fouad|123 Ramsis street|Cairo|Egypt| <------ Unused Space ------>

73 Bytes

38

2- Make records a predictable number of fields

1202345|Ahmed|Hassan|1 Safa street|Cairo|Egypt|2349890|Mona|Hussein|42 El Thawra street|Ismailia|Egypt|1210322|Tamer|Fouad|123 Ramsis street|Cairo|Egypt|

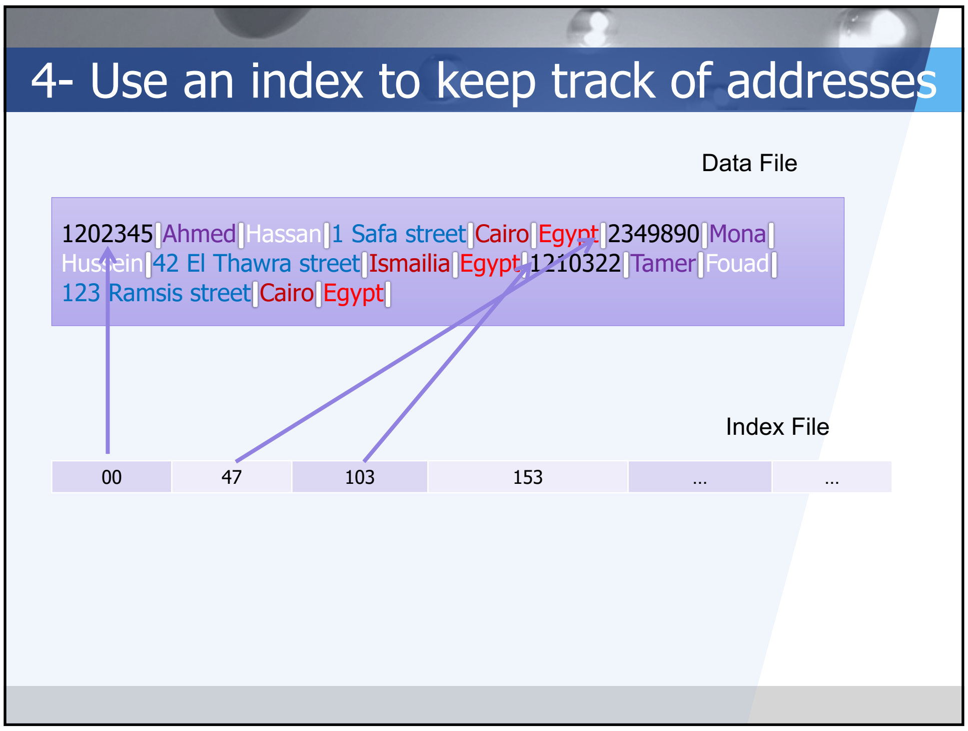
- In this example, each record consists of 6 fields.

39

3- Begin each record with a length indicator

471202345|Ahmed|Hassan|1 Safa street|Cairo|Egypt|562349890|Mona|Hussein|42 El Thawra street|Ismailia|Egypt|501210322|Tamer|Fouad|123 Ramsis street|Cairo|Egypt|

40



4- Use an index to keep track of addresses

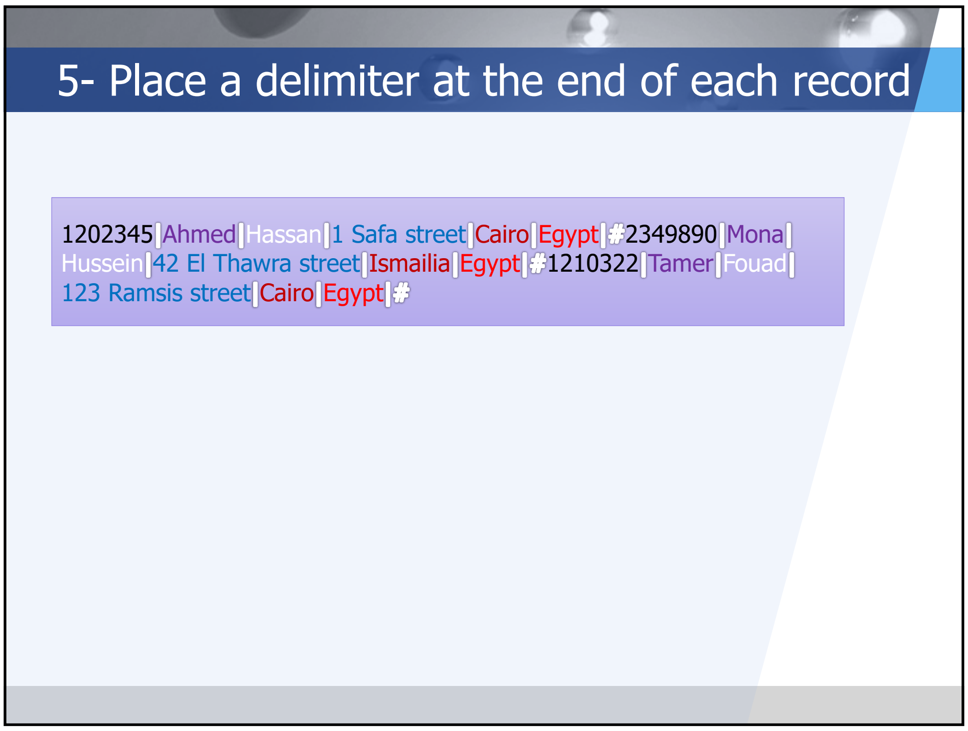
Data File

1202345|Ahmed|Hassan|1 Safa street|Cairo|Egypt|2349890|Mona|
Hussein|42 El Thawra street|Ismailia|Egypt|1210322|Tamer|Fouad|
123 Ramsis street|Cairo|Egypt|

Index File

00	47	103	153	...	...

41



5- Place a delimiter at the end of each record

1202345|Ahmed|Hassan|1 Safa street|Cairo|Egypt|#2349890|Mona|
Hussein|42 El Thawra street|Ismailia|Egypt|#1210322|Tamer|Fouad|
123 Ramsis street|Cairo|Egypt|#

42