UNIX PROGRAMMING

Course Code:	CSE-506	Credits:	03
		CIE Marks:	90
Exam Hours:	03	SEE Marks:	60

Course Learning Outcome (CLOs): After Completing this course successfully, the student will be able to…

CLO	Course Learning Outcome
CLO1	Understand the fundamentals of the Unix operating system, including its history, features, architecture, and file system.
CLO2	Demonstrate proficiency in using basic and advanced Unix commands for file handling, directory management, permissions, filters, and piping.
CLO3	Develop and debug shell scripts using variables, loops, conditional statements, and advanced scripting techniques for automation and text processing tasks.
CLO4	Apply process management and inter-process communication (IPC) techniques, including system calls, to manage processes, signals, and shared resources in Unix.
CLO5	Implement networking and security features in Unix, including client-server programming with sockets, user management, and access control, and perform basic system administration tasks.

SUMMARY OF COURSE CONTENT:

Serial No.	SUMMARY OF COURSE CONTENT	Hours	CLOs
1	Introduction to Unix: History, features, architecture, and basic Unix commands for navigation, file, and directory management.	4	CLO1, CLO2
2	File System Management: File permissions, ownership, directory hierarchy, and advanced commands like find, grep, and awk.	4	CLO1, CLO2
3	Shell Scripting Basics: Writing and executing shell scripts, variables, data types, input/output redirection, and basic loops.	5	CLO3
4	Advanced Shell Scripting: Conditional statements, functions, arrays, debugging, and text processing utilities like sed and cut.	5	CLO3
5	Process Management: Unix process architecture, system calls (fork, exec), process states, job control, signals, and scheduling.	4	CLO4
6	Inter-Process Communication (IPC): Pipes, shared memory, semaphores, and message queues for coordination between processes.	4	CLO4
7	Networking in Unix: Basics of TCP/IP, socket programming, creating client-server applications, and network troubleshooting commands like netstat and tcpdump.	5	CLO5
8	Unix System Administration: User and group management, file system mounting, backups, system monitoring, and securing a Unix environment.	5	CLO5

RECOMMENDED BOOKS:

1. **"The UNIX Programming Environment"**
   Authors: *Brian W. Kernighan, Rob Pike*
   Publisher: Prentice Hall
   Description: A classic book that introduces UNIX concepts, shell programming, and tools.
2. **"Advanced Programming in the UNIX Environment"**
   Author: *W. Richard Stevens*
   Publisher: Addison-Wesley
   Description: A detailed resource for UNIX systems programming with a focus on APIs.
3. **"UNIX Systems Programming: Communication, Concurrency and Threads"**
   Authors: *Kay A. Robbins, Steven Robbins*
   Publisher: Prentice Hall
   Description: Provides an in-depth explanation of interprocess communication and thread programming in UNIX.

CIE- Continuous Internal Evaluation (90 Marks)

ASSESSMENT PATTERN

Bloom's Category Marks (out of 90)	Tests (45)	Assignments (15)	Quizzes (15)	Attendance (15)
Remember	5	03		
Understand	5	04	05	
Apply	15	05	05	
Analyze	10			
Evaluate	5	03	05	
Create	5			

SEE- Semester End Examination (60 Marks)

Bloom's Category	Test
Remember	7
Understand	7
Apply	20
Analyze	15
Evaluate	6
Create	5

COURSE PLAN

Week No	Topics	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
1	Introduction to Unix Operating System: History, Features, Architecture	Lecture, PowerPoint presentation, Q&A	Quiz, participation in discussions	CLO1
2	Unix File System: File Types, Structure, and Navigation	Interactive lectures, hands-on practice using Unix commands	Short assignment, practical exercises	CLO1, CLO2
3	Basic Unix Commands: File Handling, Directory Management, and Permissions	Hands-on lab sessions, group discussions	Practical lab test, quiz	CLO2, CLO3
4	Advanced Unix Commands: Filters, Piping, and Redirection	Problem-solving sessions, case studies	Problem-solving tasks, in-class assignment	CLO2, CLO3
5	Shell Scripting Basics: Writing and Executing Scripts	Practical demonstrations, step-by-step scripting tutorials	Hands-on scripting assignment, quiz	CLO3
6	Variables, Loops, and Conditional Statements in Shell Scripts	Lecture with coding examples, lab practice	Lab assessment, quiz	CLO3, CLO4
7	Process Management in Unix: Forking, Background/Foreground Processes, and Signals	Hands-on exercises, group problem-solving sessions	Practical lab task, participation in group activities	CLO4
8	Inter-Process Communication: Pipes, FIFOs, and Shared Memory	Lecture, practical coding tasks, and real-world case studies	Assignment, lab test	CLO4, CLO5
9	System Calls in Unix: File Manipulation, Process Control, and I/O	Lecture, practical examples, and individual hands-on practice	Lab test, problem-solving tasks	CLO4
10	Networking in Unix: Sockets, Client-Server Programming	Lecture, practical demonstrations, group discussions	Group project, practical assessment	CLO4, CLO5

COURSE PLAN

Week No	Topics	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
11	Unix Text Processing Tools: awk, sed, grep	Practical exercises, coding examples, collaborative problem-solving	Assignment, short practical quiz	CLO3
12	Unix Development Tools: Make, gcc, and Debugging	Lecture, practical demonstrations	Practical lab assessment, assignment	CLO4
13	Security in Unix: Permissions, Access Control, and File Encryption	Lecture, hands-on lab activities	Lab report, quiz	CLO5
14	Unix System Administration Basics: User Management, Backup, and Recovery	Lecture, practical lab sessions	Short test, lab task	CLO5
15	Advanced Shell Scripting: Automation and Real-World Applications	Lecture, collaborative problem-solving sessions, hands-on scripting exercises	Group project, practical exam	CLO3, CLO4, CLO5
16	Revision and Comprehensive Review	Group discussions, Q&A sessions	Participation in group activities, review task	CLO1–CLO5
17	Final Exam and Project Presentation	Assessment of final project and examination	Final exam, project-based evaluation	CLO1–CLO5

Govt. & UGC Approved

UNIVERSITY OF GLOBAL VILLAGE (UGV), BARISHAL

THE UNIVERSITY FOR HI-TECH AND HUMANITY

Dept. of Computer Science and Engineering

Course Code: CSE-506

Course Name: Unix Programming

Instructor: **Md. Tariqul Islam**

Textbook:

UNIX: The Textbook (3rdth edition)

WEEK: 01
INTRODUCTION TO UNIX

HISTORY OF UNIX

- **UNIX** is a computer operating system.
- It was first developed in 1969 at Bell Labs.
- Ken Thompson, Dennis Ritchie, Douglas McIlroy, and others created it.
- They used assembly language to write it.
- In 1972, the Unix code was rewritten with the new C programming language.

HISTORY OF UNIX

- The Unix operating system is a multiuser and multiprocessing system.
- This means it can run several application programs at the same time, for more than one user at the same time.
- It also is able to operate well in a network of computers.
- Computer security is also important in Unix, because many people can have access to it, both by using a computer directly or over a network.

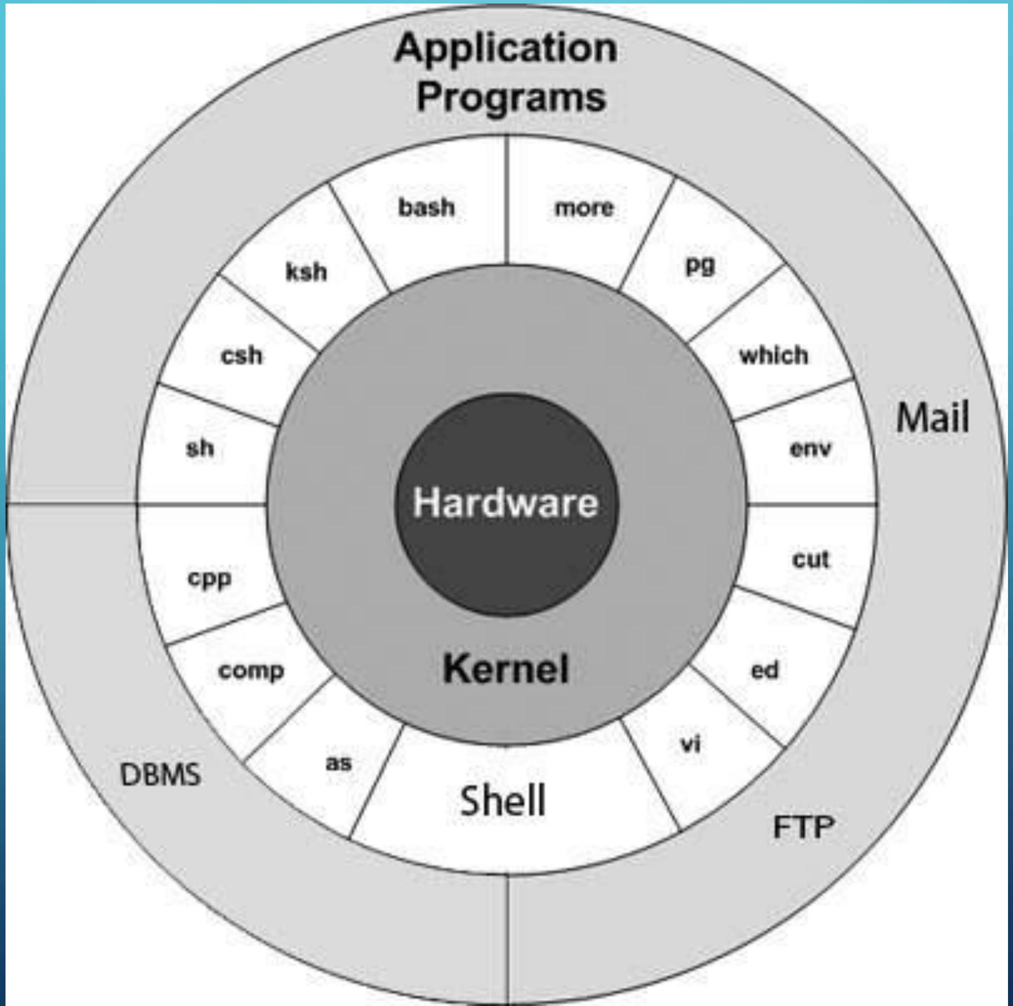
UNIX / LINUX SCRIPTING- GETTING STARTED

What is Unix ?

- The Unix **operating system** is a set of programs that plays a role as a connection between the computer and the user.
- Users communicates with the kernel through a program known as the shell.
- The shell is a command line interpreter.

UNIX ARCHITECTURE

- Here is a basic block diagram of a Unix system −



UNIX ARCHITECTURE

The main concept that unites all the versions of Unix is the following four basics –

- **Kernel** – The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
- **Shell** – The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.

UNIX ARCHITECTURE

Commands and Utilities – There are various commands and utilities which you can make use of in your day to day activities.

- **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities.
- There are over 250 standard commands plus numerous others provided through 3rd party software.
- All the commands come along with various options.
- **Files and Directories** – All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

PLEASE FIND THE STEP BY STEP PROCEDURE TO INSTALL CYGWIN-

- 1. Download the file setup.exe from site www.cygwin.com.
- 2. Run the application from your local hard drive.
- 3. Choose the next button on first screen.
- 4. Select "Install from Internet" option and click next button.
- 5. Give a preferred installation directory and click next button.
- 6. Give a temporary installation directory and click next button.
- 7. Select "Direct Connection" and click next button.
- 8. Give a temporary directory and click next button.
- 9. Select "Direct Connection" and click next.
- 10. Select a download site and click next.
- 11. Select the packages you want to install and click next.
- 12. Simply select another server and continue the installation process.
- 13. Once installation is completed, click Finish and continue with the Setup section.

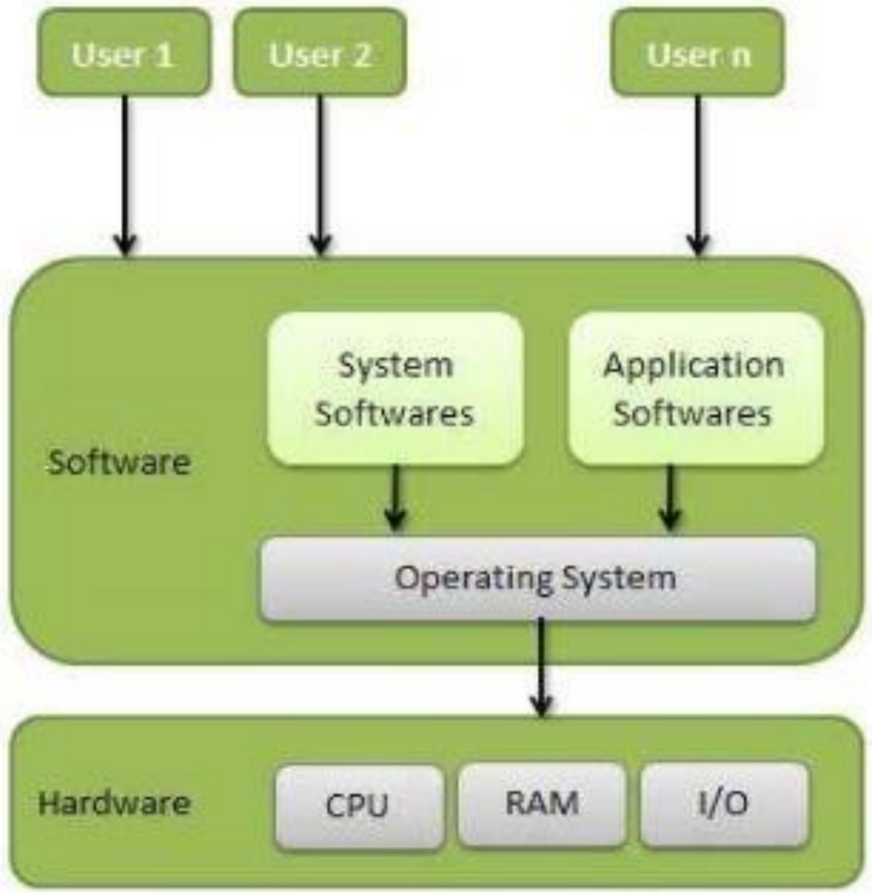
SOME BASI UNIX

- `Cal - Calendar`
- `whoami`

WEEK: 02
INTRODUCTION TO OPERATING SYSTEM

What is Operating System?

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



WHAT IS UNIX?

- UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

- UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment. However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no windows interface available, for example, in a telnet session.

TYPES OF UNIX

- There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.
- Here in the School, we use Solaris on our servers and workstations, and Fedora Linux on the servers and desktop PCs.

UNIX Introduction

What is UNIX?

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

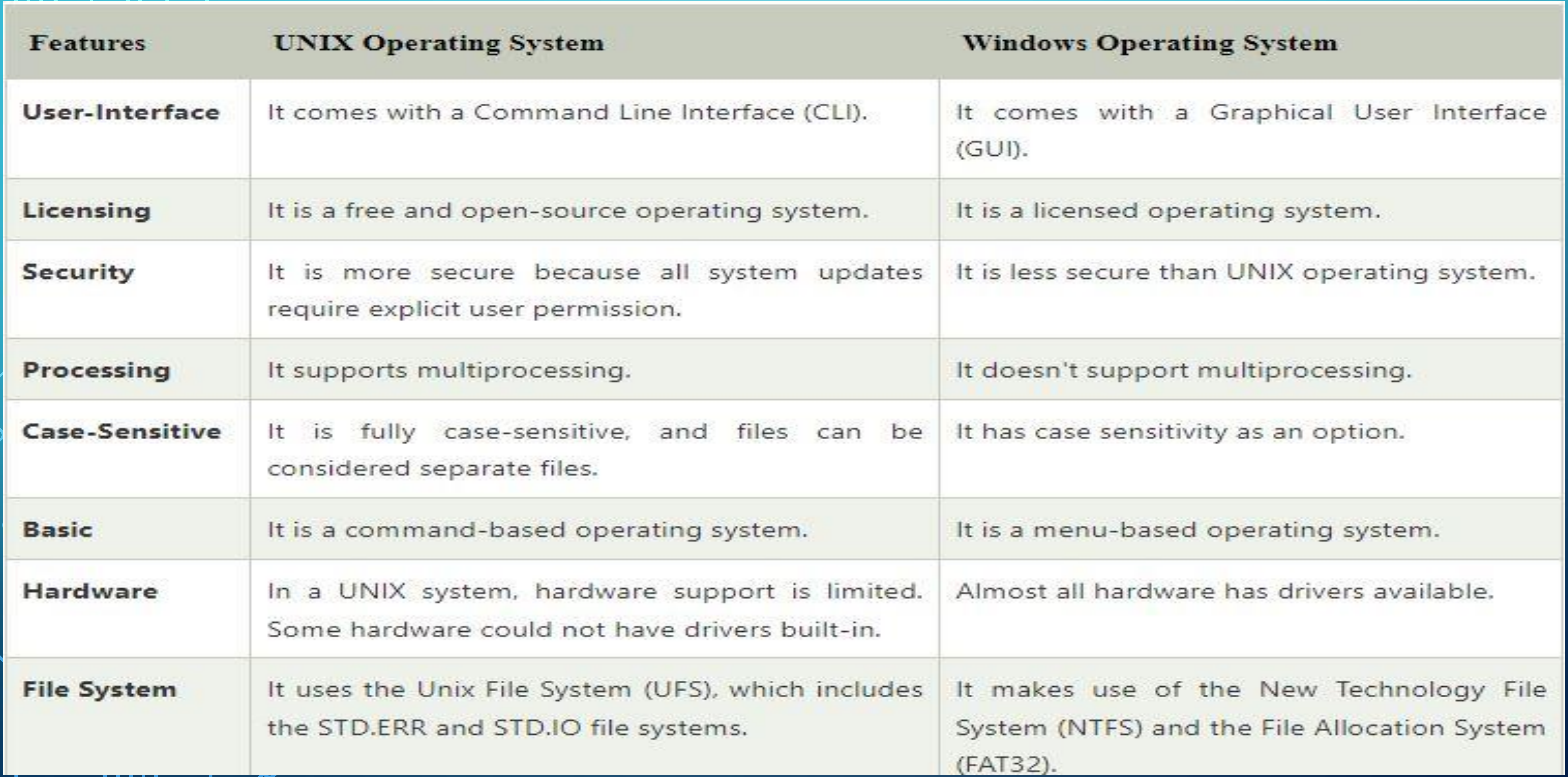
UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment. However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no windows interface available, for example, in a telnet session.

Types of UNIX

There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.

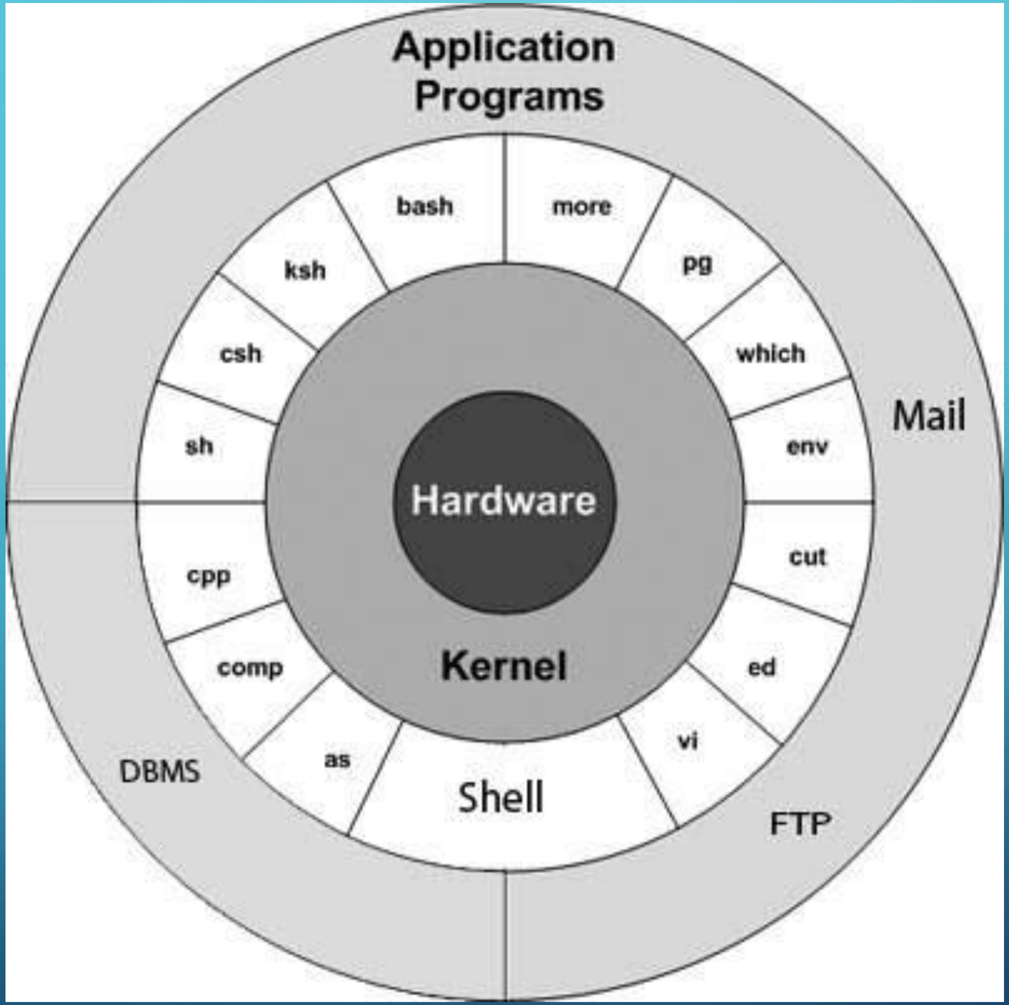
Here in the School, we use Solaris on our servers and workstations, and Fedora Linux on the servers and desktop PCs.

Comparison between the UNIX operating system and Windows operating system



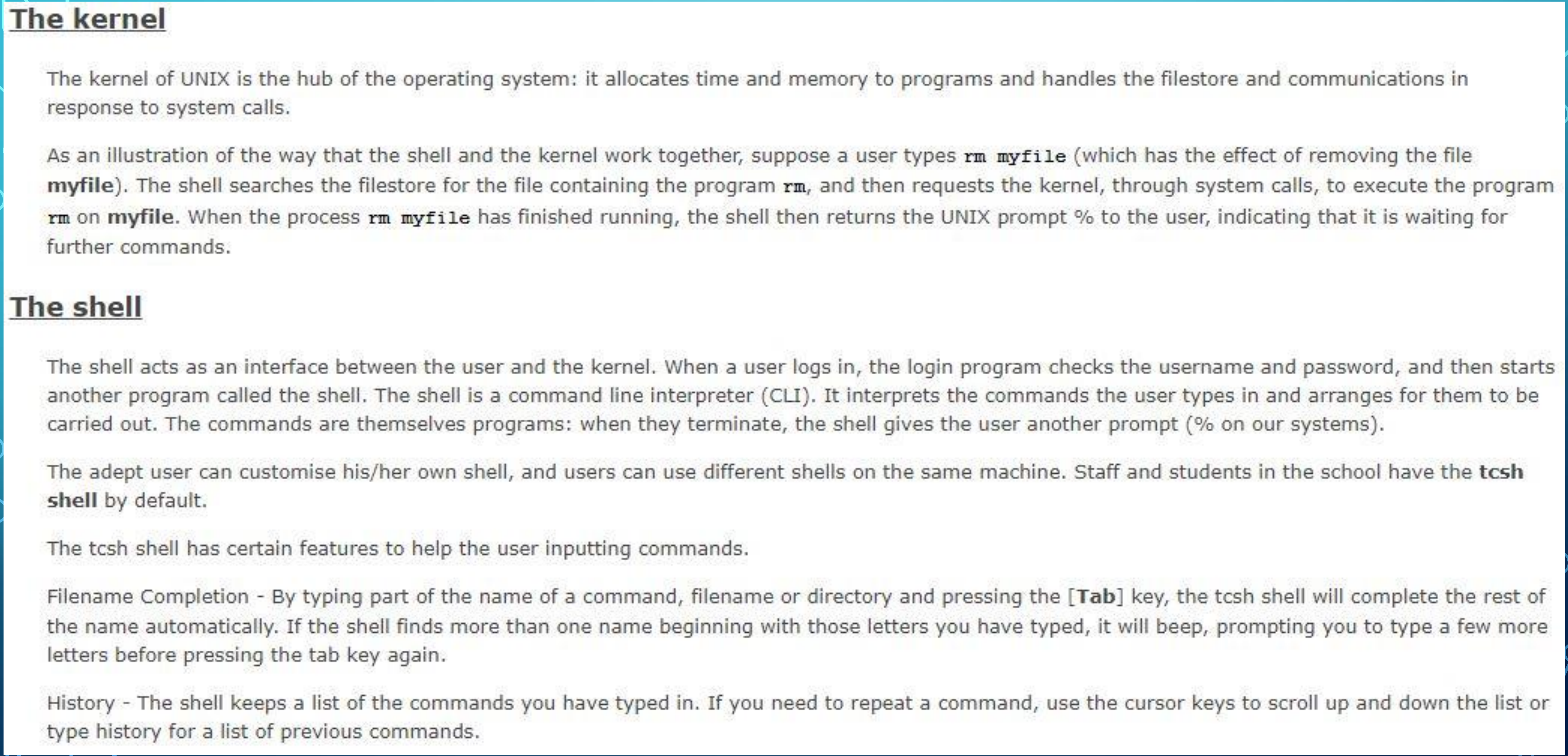
Features	UNIX Operating System	Windows Operating System
User-Interface	It comes with a Command Line Interface (CLI).	It comes with a Graphical User Interface (GUI).
Licensing	It is a free and open-source operating system.	It is a licensed operating system.
Security	It is more secure because all system updates require explicit user permission.	It is less secure than UNIX operating system.
Processing	It supports multiprocessing.	It doesn't support multiprocessing.
Case-Sensitive	It is fully case-sensitive, and files can be considered separate files.	It has case sensitivity as an option.
Basic	It is a command-based operating system.	It is a menu-based operating system.
Hardware	In a UNIX system, hardware support is limited. Some hardware could not have drivers built-in.	Almost all hardware has drivers available.
File System	It uses the Unix File System (UFS), which includes the STD.ERR and STD.IO file systems.	It makes use of the New Technology File System (NTFS) and the File Allocation System (FAT32).

- The UNIX operating system is made up of three parts; the kernel, the shell and the programs.



WEEK: 03
INTRODUCTION TO UNIX OPERATING SYSTEM

The UNIX operating system



The kernel

The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the filestore and communications in response to system calls.

As an illustration of the way that the shell and the kernel work together, suppose a user types **`rm myfile`** (which has the effect of removing the file **myfile**). The shell searches the filestore for the file containing the program **`rm`**, and then requests the kernel, through system calls, to execute the program **`rm`** on **myfile**. When the process **`rm myfile`** has finished running, the shell then returns the UNIX prompt % to the user, indicating that it is waiting for further commands.

The shell

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt (% on our systems).

The adept user can customise his/her own shell, and users can use different shells on the same machine. Staff and students in the school have the **tcsh shell** by default.

The tcsh shell has certain features to help the user inputting commands.

Filename Completion - By typing part of the name of a command, filename or directory and pressing the [**Tab**] key, the tcsh shell will complete the rest of the name automatically. If the shell finds more than one name beginning with those letters you have typed, it will beep, prompting you to type a few more letters before pressing the tab key again.

History - The shell keeps a list of the commands you have typed in. If you need to repeat a command, use the cursor keys to scroll up and down the list or type history for a list of previous commands.

Files and processes

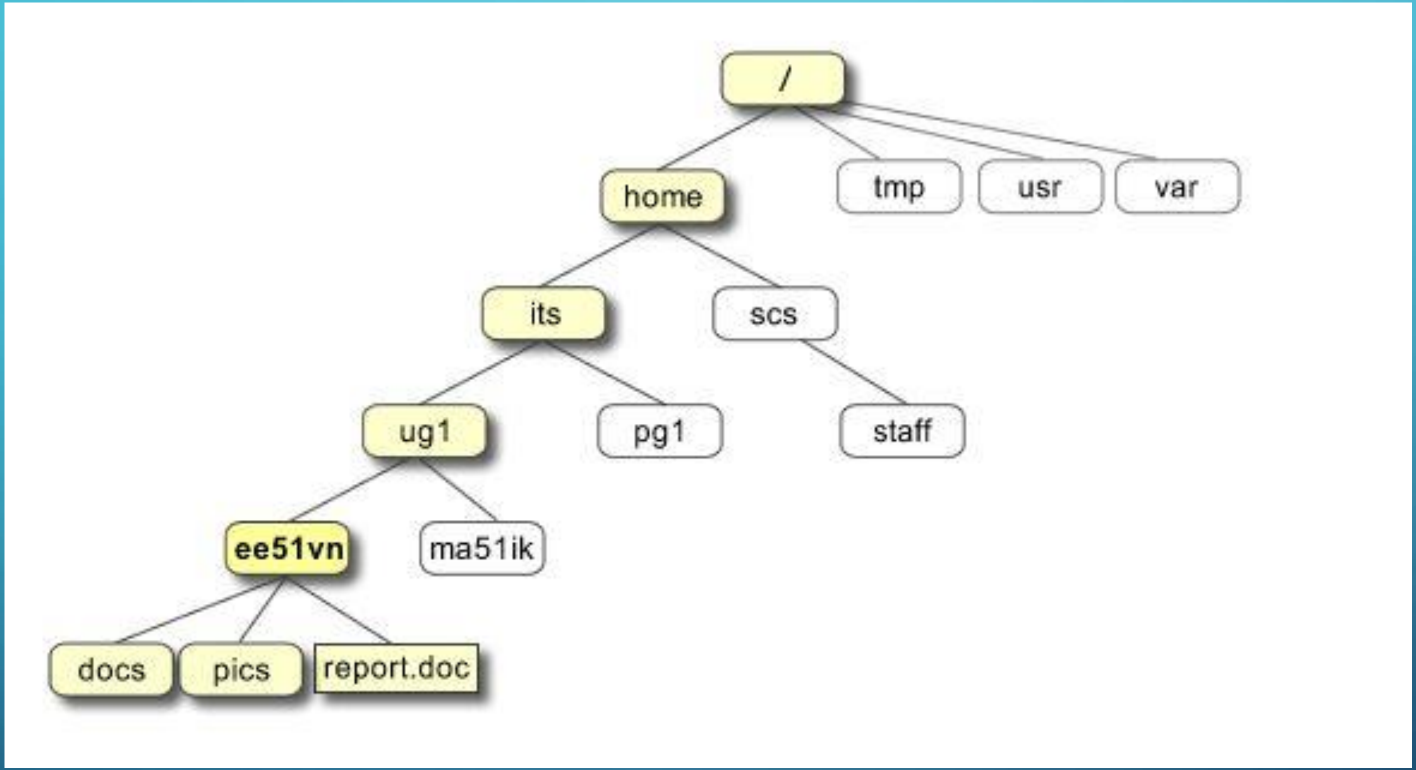
- ❑ Everything in UNIX is either a file or a process.
- ❑ A process is an executing program identified by a unique PID (process identifier).
- ❑ A file is a collection of data. They are created by users using text editors, running compilers etc.

UNIX FILE SYSTEM

- Unix file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system. Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called “root” which is represented by a “/”. All other files are “descendants” of root.

The Directory Structure

All the files are grouped together in the directory structure. The file-system is arranged in a hierarchical structure, like an inverted tree. The top of the hierarchy is traditionally called **root** (written as a slash /)



- In the diagram above, we see that the home directory of the undergraduate student **"ee51vn"** contains two sub-directories (**docs** and **pics**) and a file called **report.doc**.

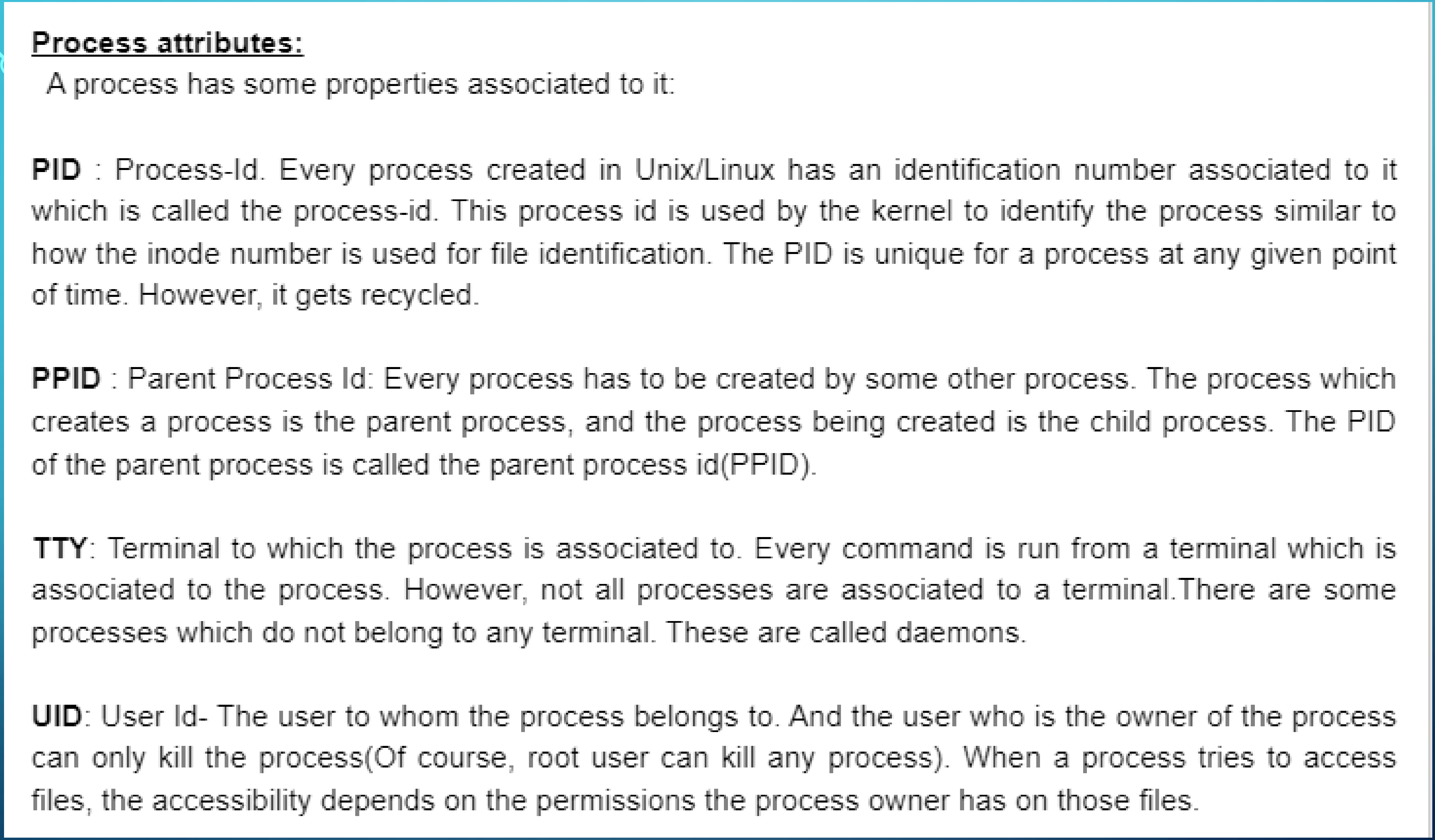
- The full path to the file **report.doc** is **"/home/its/ug1/ee51vn/report.doc"**

BRIEFLY EXPLAIN THE DIRECTORY STRUCTURE OF UNIX OS ?

https://www.geeksforgeeks.org/unix-file-system/

WHAT IS A PROCESS IN UNIX / LINUX?

- A **process** is a program in execution in memory or in other words, an instance of a program in memory. Any program executed creates a process. A program can be a command, a shell script, or any binary executable or any application. However, not all commands end up in creating process, there are some exceptions. Similar to how a file created has properties associated with it, a process also has lots of properties associated to it.



Process attributes:

A process has some properties associated to it:

PID : Process-Id. Every process created in Unix/Linux has an identification number associated to it which is called the process-id. This process id is used by the kernel to identify the process similar to how the inode number is used for file identification. The PID is unique for a process at any given point of time. However, it gets recycled.

PPID : Parent Process Id: Every process has to be created by some other process. The process which creates a process is the parent process, and the process being created is the child process. The PID of the parent process is called the parent process id(PPID).

TTY: Terminal to which the process is associated to. Every command is run from a terminal which is associated to the process. However, not all processes are associated to a terminal.There are some processes which do not belong to any terminal. These are called daemons.

UID: User Id- The user to whom the process belongs to. And the user who is the owner of the process can only kill the process(Of course, root user can kill any process). When a process tries to access files, the accessibility depends on the permissions the process owner has on those files.

File Descriptors: File descriptors related to the process: input, output and error file descriptors.

List the processes:

```
$ ps
     PID    TTY   TIME CMD
 1315012  pts/1   0:00 -ksh
 2490430  pts/1   0:00 ps
```

ps is the Unix / Linux command which lists the active processes and its status. By default, it lists the processes belonging to the current user being run from the current terminal.

The ps command output shows 4 things:
PID : The unique id of the process
TTY: The terminal from which the process or command is executed.
TIME: The amount of CPU time the process has taken
CMD: The command which is executed.

VIRTUAL MEMORY

- In computing, virtual memory, or virtual storage is a memory management technique that provides an "idealized abstraction of the storage resources that are actually available on a given machine" which "creates the illusion to users of a very large (main) memory".

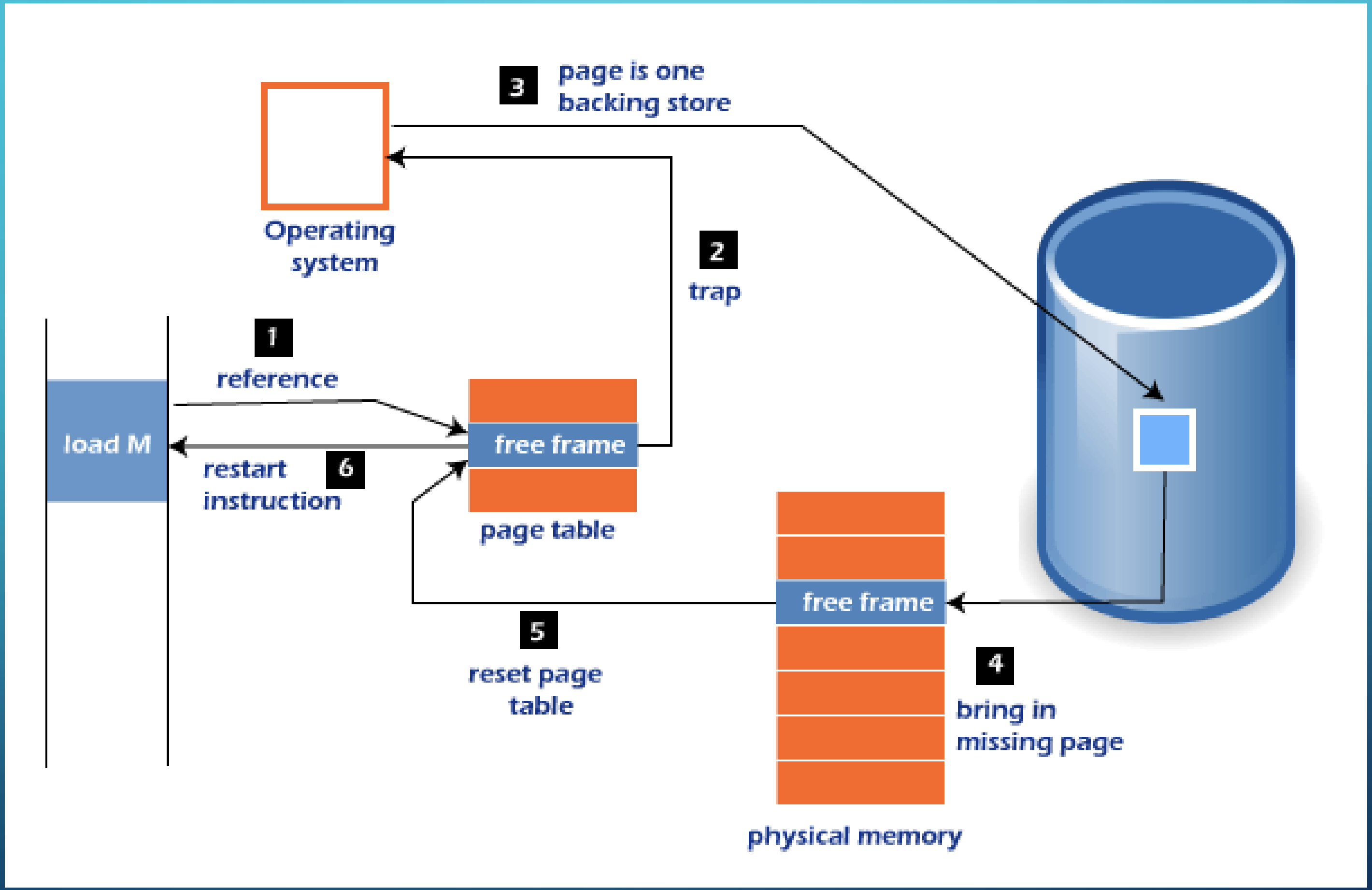
WHAT IS PAGE FAULT?

- A page fault is an interruption that occurs when a software program attempts to access a memory block not currently stored in the system's RAM.
- This exception tells the operating system to find the block in virtual memory so it can be sent from a device's storage (SSD or HD) to RAM.

WHAT IS PAGE FAULT IN OPERATING SYSTEM?

- Page faults dominate more like an error. A page fault will happen if a program tries to access a piece of memory that does not exist in physical memory (main memory). The fault specifies the operating system to trace all data into virtual memory management and then relocate it from secondary memory to its primary memory, such as a hard disk.

WHAT IS PAGE FAULT IN OPERATING SYSTEM?



PAGE FAULT

- A page fault trap occurs if the requested page is not loaded into memory. The page fault primarily causes an exception, which is used to notify the operating system to retrieve the "pages" from virtual memory to continue operation. Once all of the data has been placed into physical memory, the program resumes normal operation. The Page fault process occurs in the background, and thus the user is unaware of it.

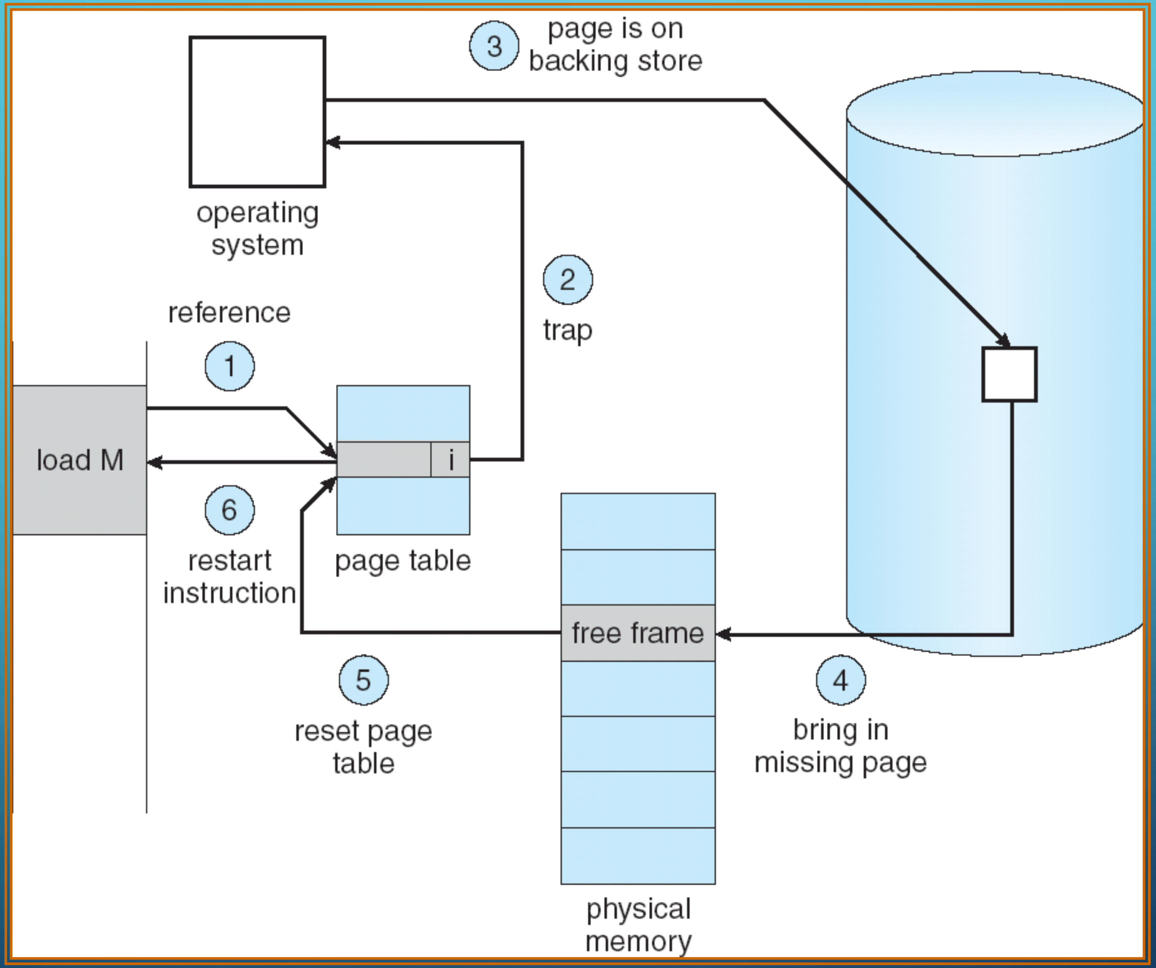
- The computer's hardware track to the kernel and the program counter is often saved on the stack. The CPU registers hold information about the current state of instruction.
- An assembly program is started, which saves the general registers and other volatile data to prevent the Operating system from destroying it.

DESCRIBE THE PROCESS OF HANDLING A PAGE FAULT WITH DIAGRAM.

- A Page Fault happens when you access a page that has been marked as invalid. The paging hardware would notice that the invalid bit is set while translating the address across the page table, which will cause an operating system trap. The trap is caused primarily by the OS's failure to load the needed page into memory.
- Now, let's understand the procedure of page fault handling in the OS:

1. Firstly, an internal table for this process to assess whether the reference was valid or invalid memory access.
2. If the reference becomes invalid, the system process would be terminated. Otherwise, the page will be paged in.
3. After that, the free-frame list finds the free frame in the system.
4. Now, the disk operation would be scheduled to get the required page from the disk.
5. When the I/O operation is completed, the process's page table will be updated with a new frame number, and the invalid bit will be changed. Now, it is a valid page reference.
6. If any page fault is found, restart these steps from starting.

STEPS IN HANDLING A PAGE FAULT



BACKGROUND AND FOREGROUND JOBS

- **A process that connects to the terminal is called a foreground job.** A job is said to be in the foreground because it can communicate with the user via the screen and the keyboard.
- **On the other hand, a process that disconnects from the terminal and cannot communicate with the user is called a background job.** If the background job requires interaction with the user, it will stop and wait until establishing a connection to the terminal.
- We can place the jobs that do not require interaction from the user as they run (like sorting a large file) in the background. This allows the user to access the terminal and continue to work, instead of waiting for a long job to finish:

FOREGROUND PROCESS AND BACKGROUND PROCESS WITH APPROPRIATE COMMAND?

https://www.linuxshelltips.com/foreground-and-background-process-in-linux/

- What is page fault?
- What is process?
- Describe the process of handling a page fault with diagram.
- Explain foreground process and background process with appropriate command.
- What is UNIX?
- Describe why UNIX OS is superior than Windows OS.
- Briefly explain the directory structure of UNIX OS
- What is swapping?
- Describe why swap is necessary in UNIX OS?
- Describe swapping in and swapping out with diagram.

EXPLAIN THE FOLLOWING COMMANDS IN UNIX OS?

- pwd
- cat
- mkdir
- grep
- touch

PWD COMMAND IN LINUX WITH EXAMPLES

https://www.geeksforgeeks.org/pwd-command-in-linux-with-examples/

MKDIR COMMAND IN LINUX WITH EXAMPLES

https://www.geeksforgeeks.org/mkdir-command-in-linux-with-examples/

CAT COMMAND IN LINUX WITH EXAMPLES

https://www.geeksforgeeks.org/cat-command-in-linux-with-examples/

TOUCH COMMAND IN LINUX WITH EXAMPLES

https://www.geeksforgeeks.org/touch-command-in-linux-with-examples/

GREP COMMAND IN LINUX WITH EXAMPLES

https://www.geeksforgeeks.org/grep-command-in-unixlinux/

a) What is swapping?

b) Describe why swap is necessary in UNIX OS?

c) Describe swapping in and swapping out with diagram.

For Details Find the check the link bellow:

https://www.scaler.com/topics/swapping-in-os/

https://www.javatpoint.com/swapping-in-operating-system

WHAT IS SWAPPING?

- Swapping in OS is one of those schemes which fulfill the goal of maximum utilization of CPU and memory management by swapping in and swapping out processes from the main memory. Swap in removes the process from hard drive(secondary memory) and swap out removes the process from RAM(main memory).
- **Example:** Suppose the user process's size is 2048KB and is a standard hard disk where swapping has a data transfer rate of 1Mbps. Now we will calculate how long it will take to transfer from main memory to secondary memory.

DESCRIBE WHY SWAP IS NECESSARY IN UNIX OS?

- Advantages of Swapping

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

DESCRIBE WHY SWAP IS NECESSARY IN UNIX OS?

- Disadvantages of Swapping

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

DESCRIBE SWAPPING IN AND SWAPPING OUT WITH DIAGRAM.

- There are two important concepts in the process of swapping which are as follows:

1. Swap In
2. Swap Out

- Refer to the 'Swap In and Swap out in OS' section for a detailed explanation.

DESCRIBE SWAPPING IN AND SWAPPING OUT WITH DIAGRAM.

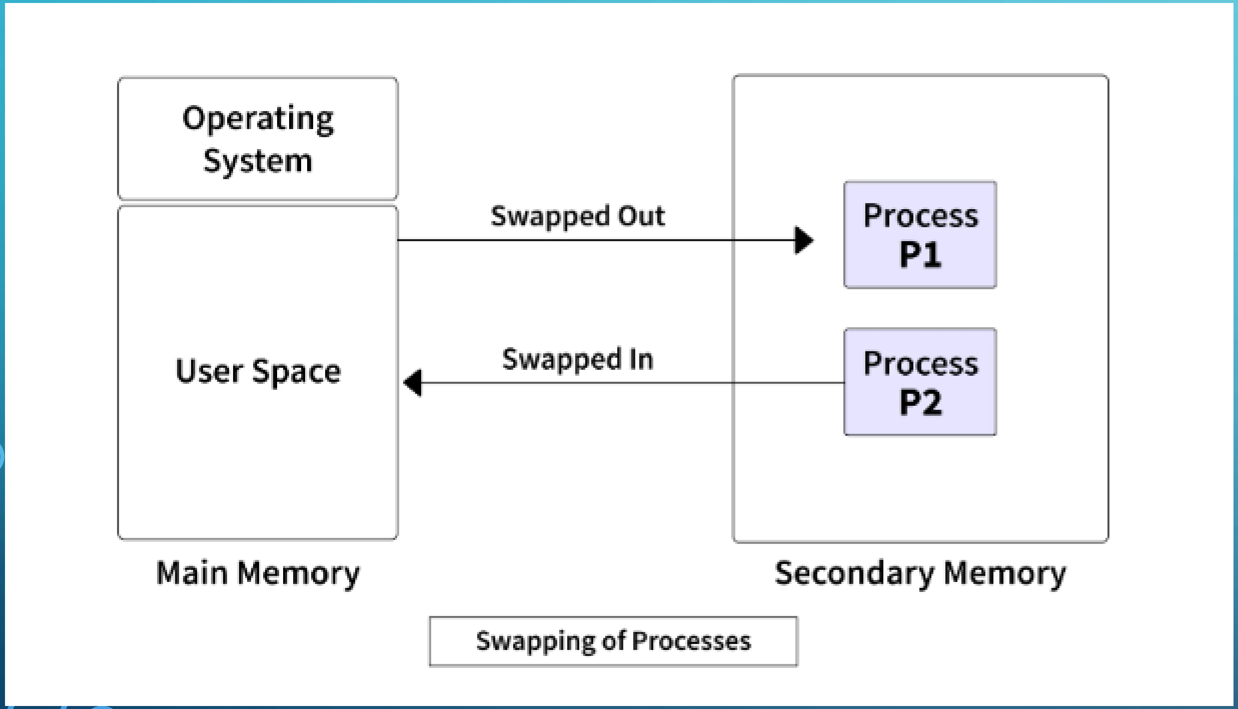
Swap In:

- The method of removing a process from secondary memory (**Hard Drive**) main memory (**RAM**) for execution is known as the Swap In method.

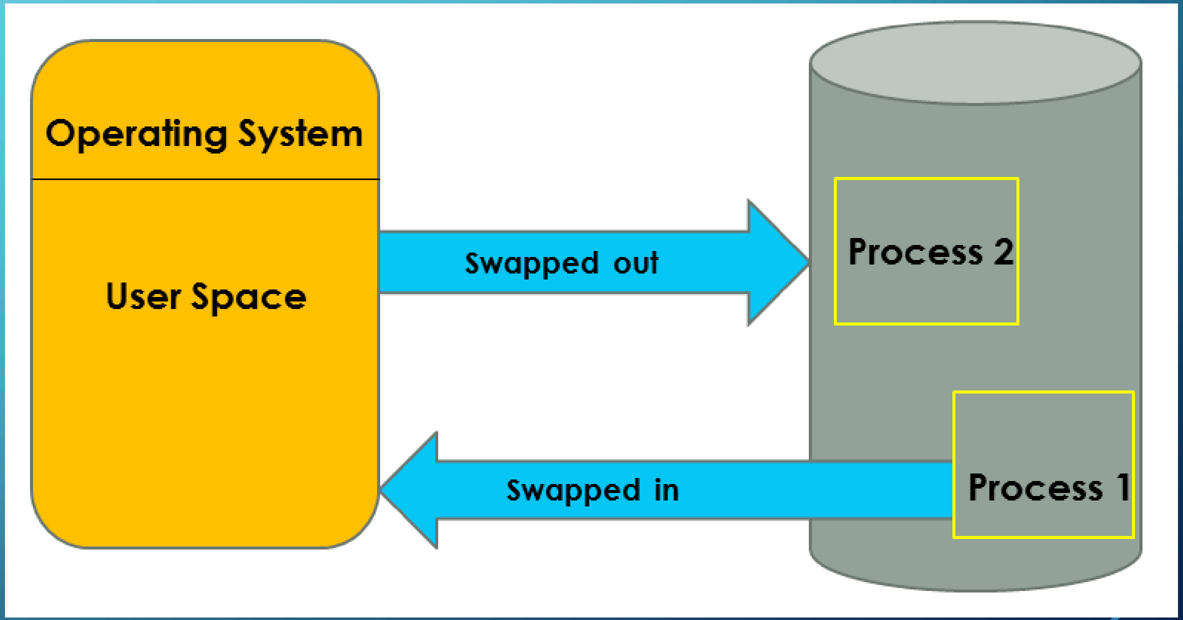
Swap Out:

- It is a method of bringing out a process from the main memory(**RAM**) and secondary memory(**hard drive**) so that the processes with higher priority consumption will be executed is known as the Swap Out method.

DESCRIBE SWAPPING IN AND SWAPPING OUT WITH DIAGRAM.

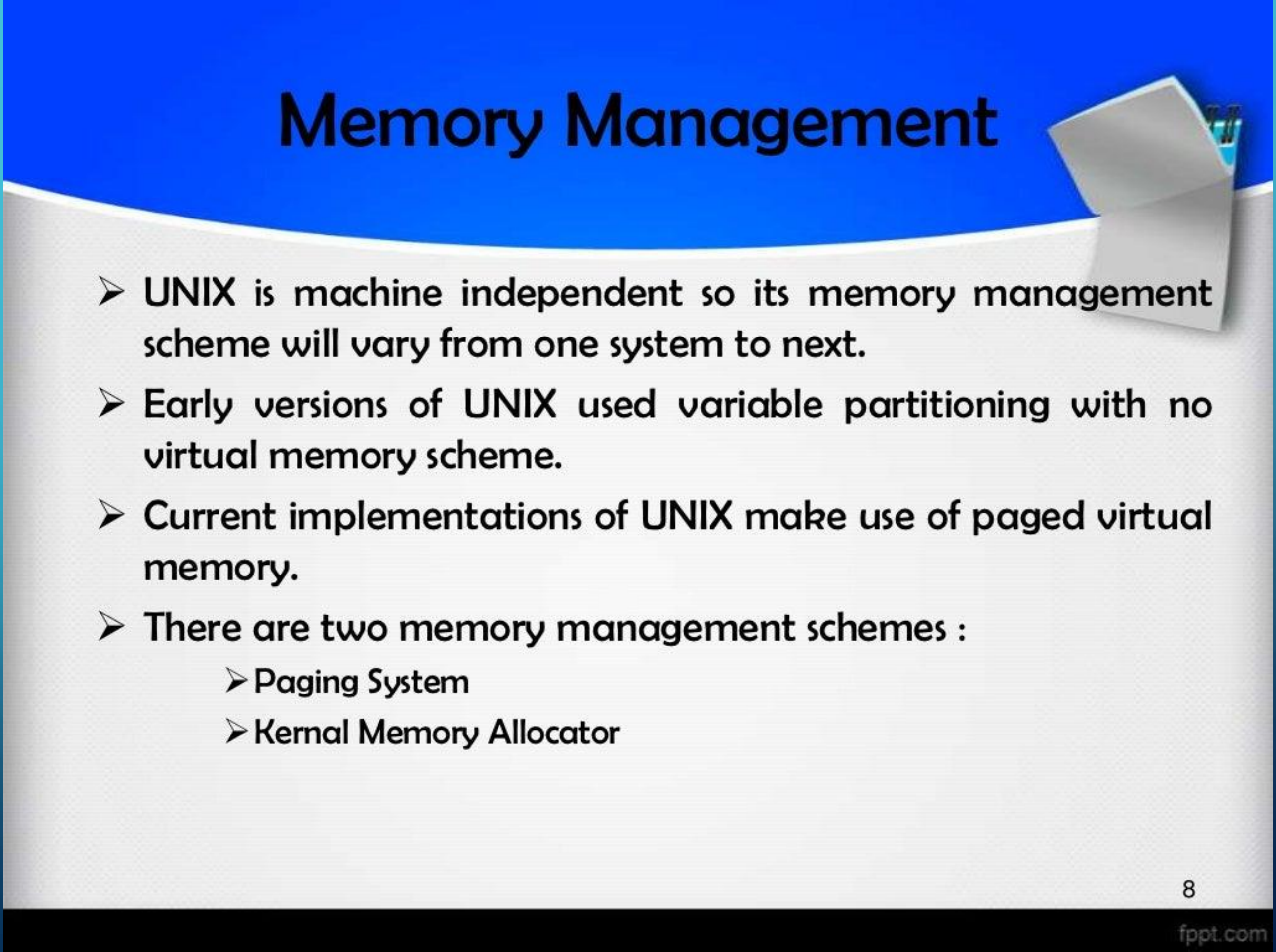


Swapping of Processes



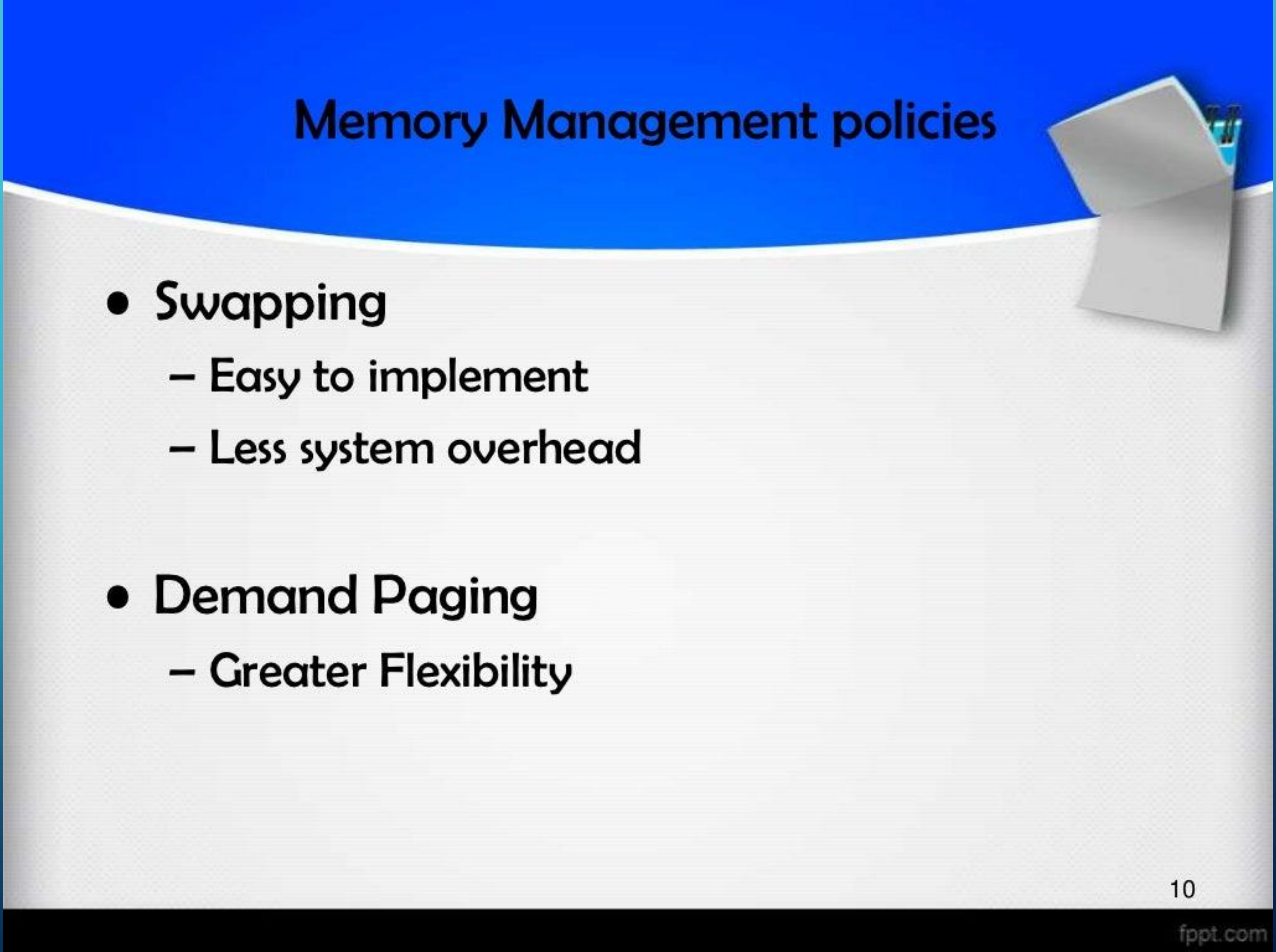
WEEK: 04
UNIX MEMORY MANAGEMENT 01

UNIX MEMORY MANAGEMENT



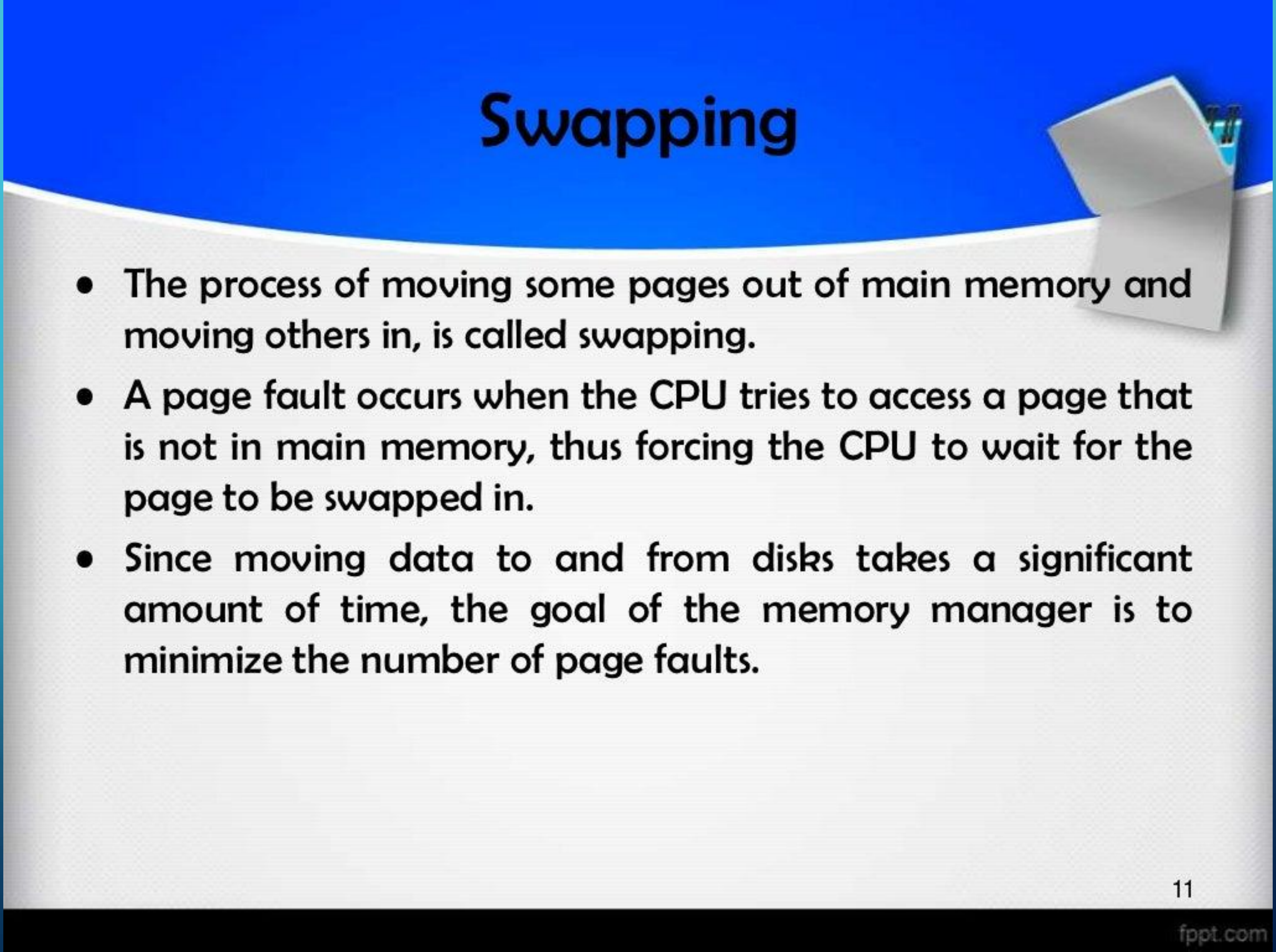
Memory Management

- UNIX is machine independent so its memory management scheme will vary from one system to next.
- Early versions of UNIX used variable partitioning with no virtual memory scheme.
- Current implementations of UNIX make use of paged virtual memory.
- There are two memory management schemes :
  - Paging System
  - Kernal Memory Allocator



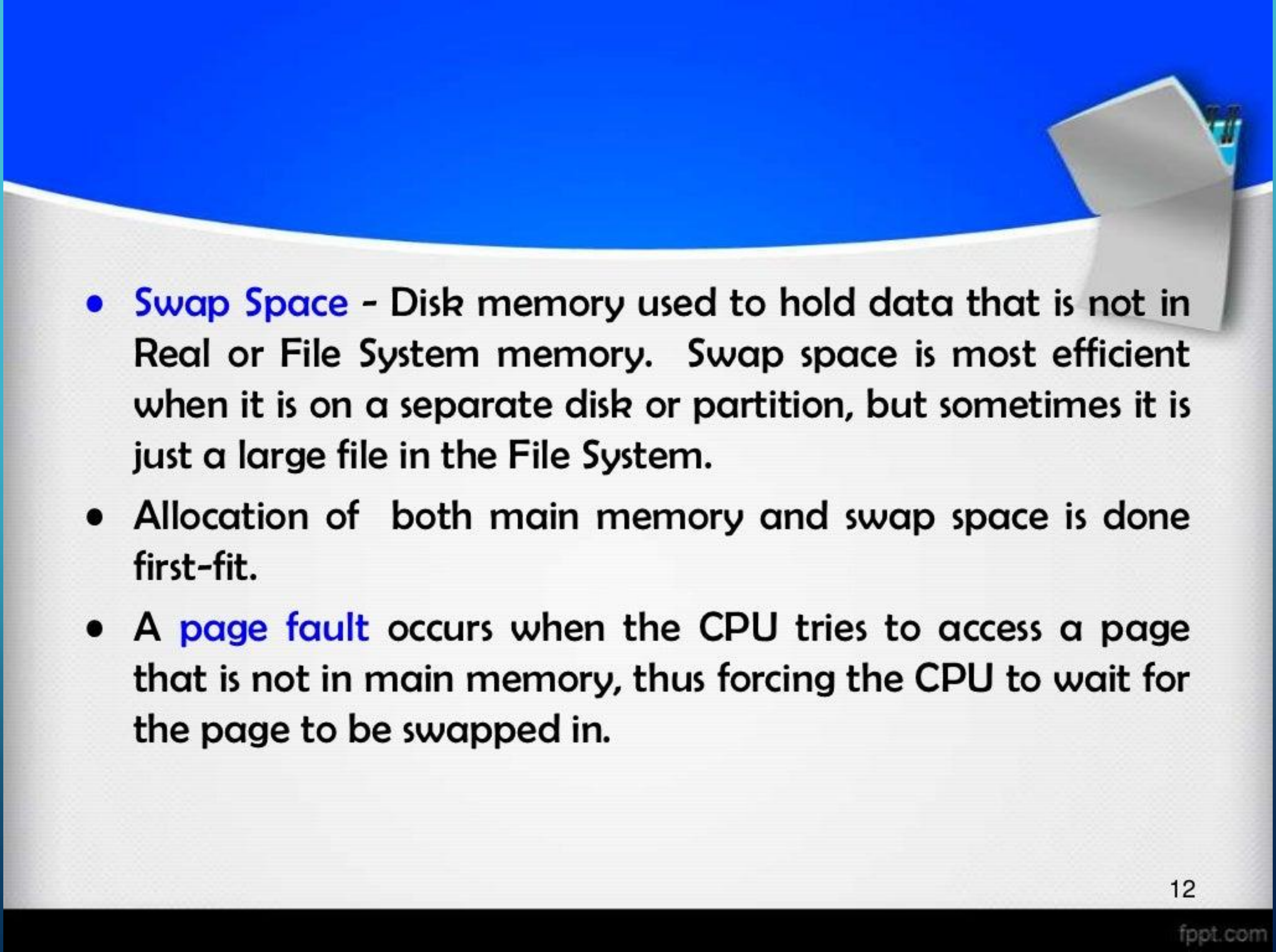
Memory Management policies

- Swapping
  - Easy to implement
  - Less system overhead
- Demand Paging
  - Greater Flexibility

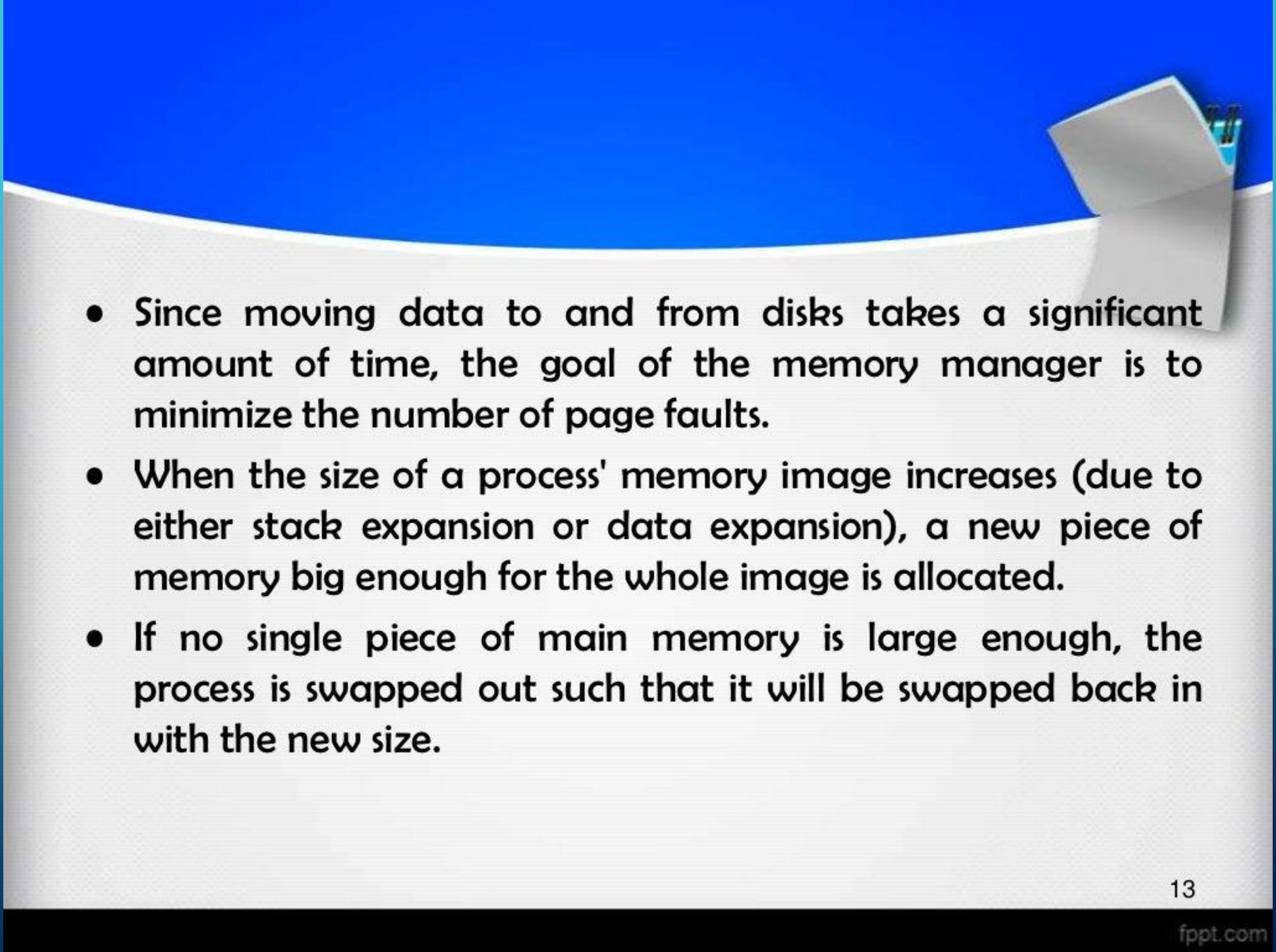


Swapping

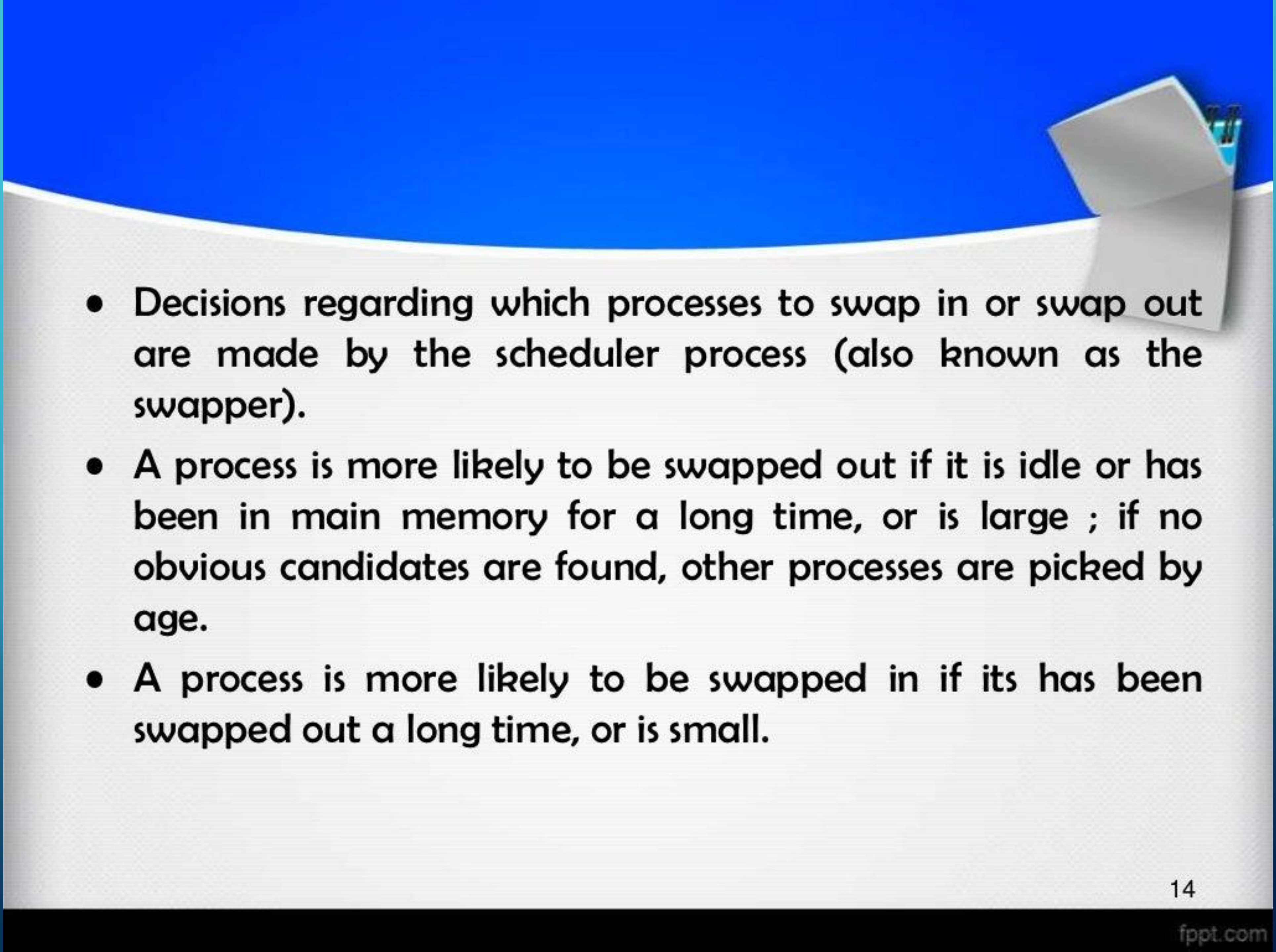
- The process of moving some pages out of main memory and moving others in, is called swapping.
- A page fault occurs when the CPU tries to access a page that is not in main memory, thus forcing the CPU to wait for the page to be swapped in.
- Since moving data to and from disks takes a significant amount of time, the goal of the memory manager is to minimize the number of page faults.



- Swap Space - Disk memory used to hold data that is not in Real or File System memory. Swap space is most efficient when it is on a separate disk or partition, but sometimes it is just a large file in the File System.
- Allocation of both main memory and swap space is done first-fit.
- A page fault occurs when the CPU tries to access a page that is not in main memory, thus forcing the CPU to wait for the page to be swapped in.



- Since moving data to and from disks takes a significant amount of time, the goal of the memory manager is to minimize the number of page faults.
- When the size of a process' memory image increases (due to either stack expansion or data expansion), a new piece of memory big enough for the whole image is allocated.
- If no single piece of main memory is large enough, the process is swapped out such that it will be swapped back in with the new size.



- Decisions regarding which processes to swap in or swap out are made by the scheduler process (also known as the swapper).
- A process is more likely to be swapped out if it is idle or has been in main memory for a long time, or is large ; if no obvious candidates are found, other processes are picked by age.
- A process is more likely to be swapped in if its has been swapped out a long time, or is small.

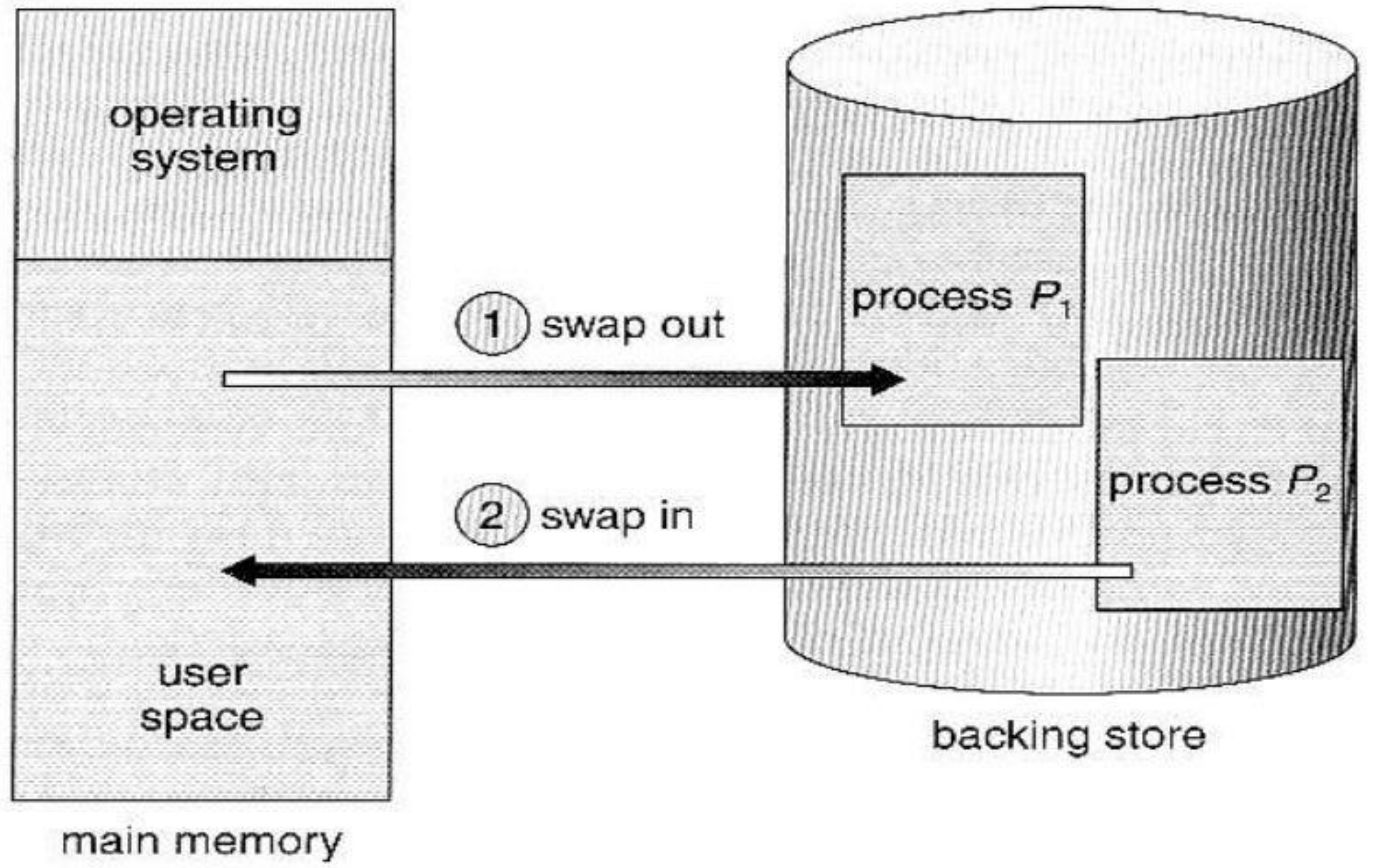
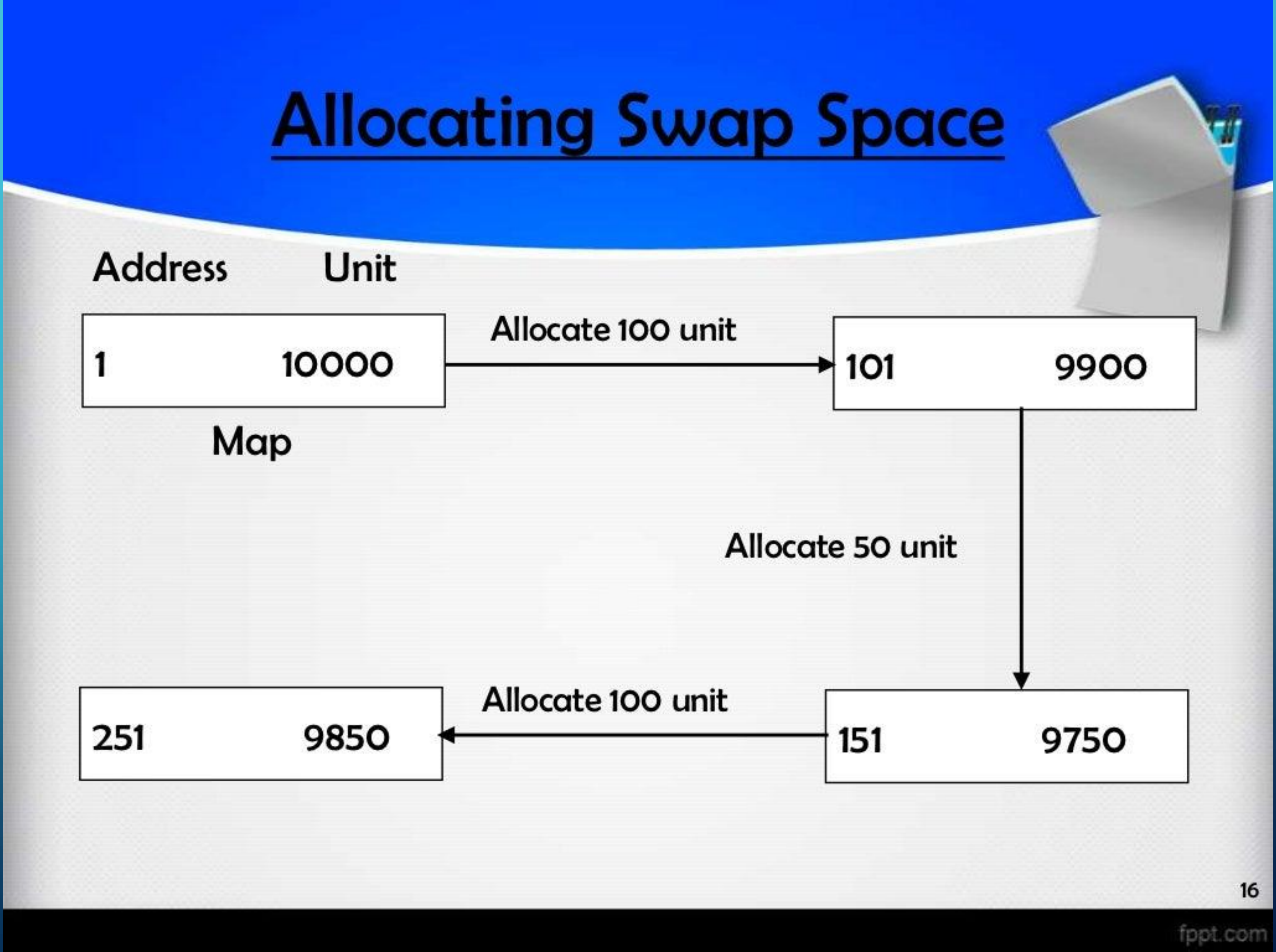
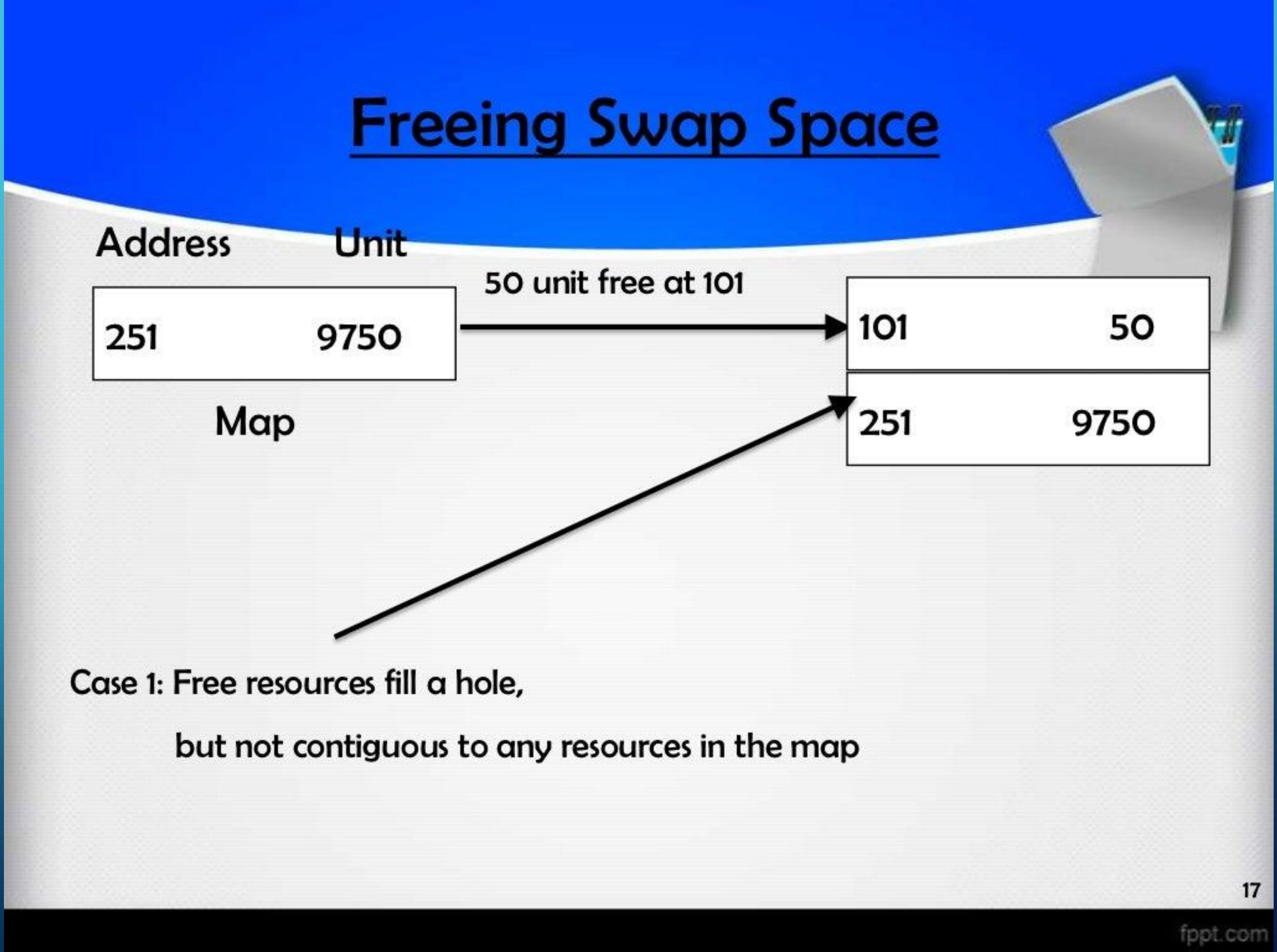


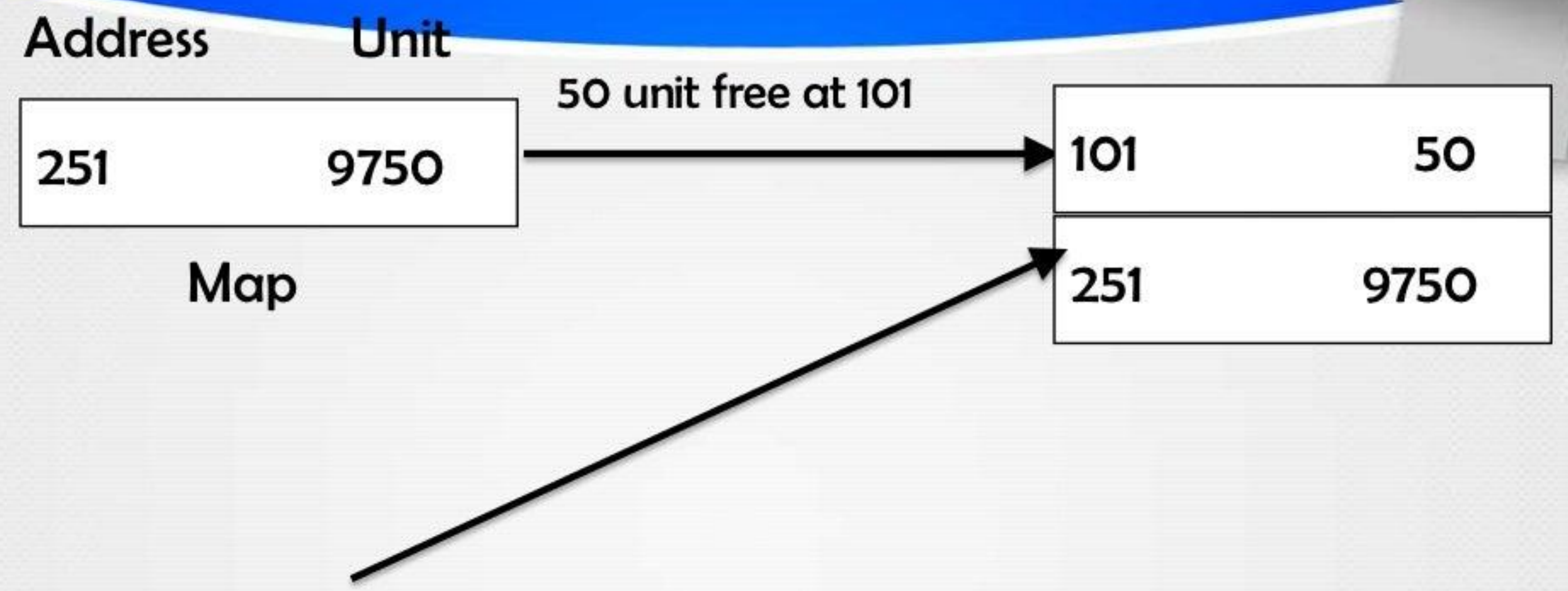
Figure 8.5 Swapping of two processes using a disk as a backing store.



Allocating Swap Space



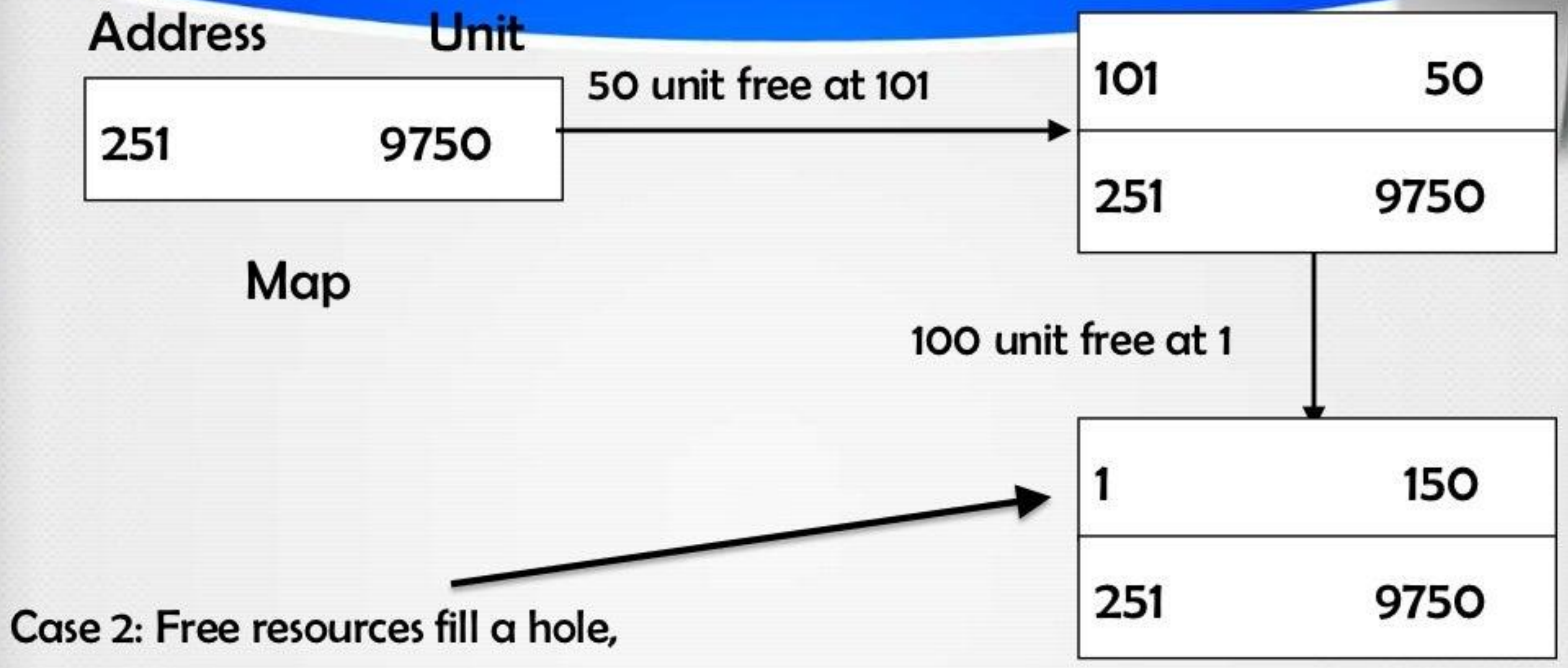
Freeing Swap Space



Case 1: Free resources fill a hole,

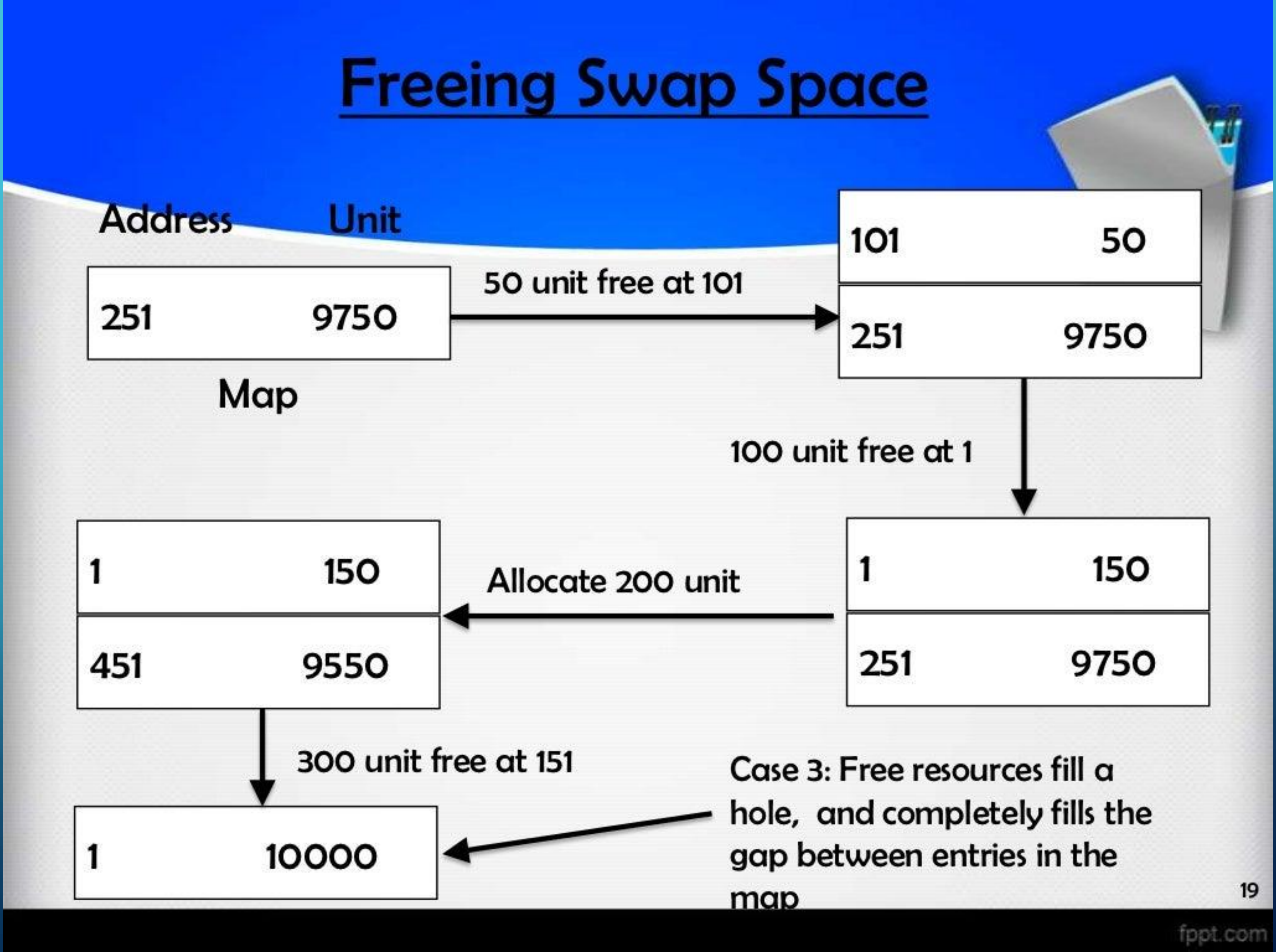
but not contiguous to any resources in the map

Freeing Swap Space



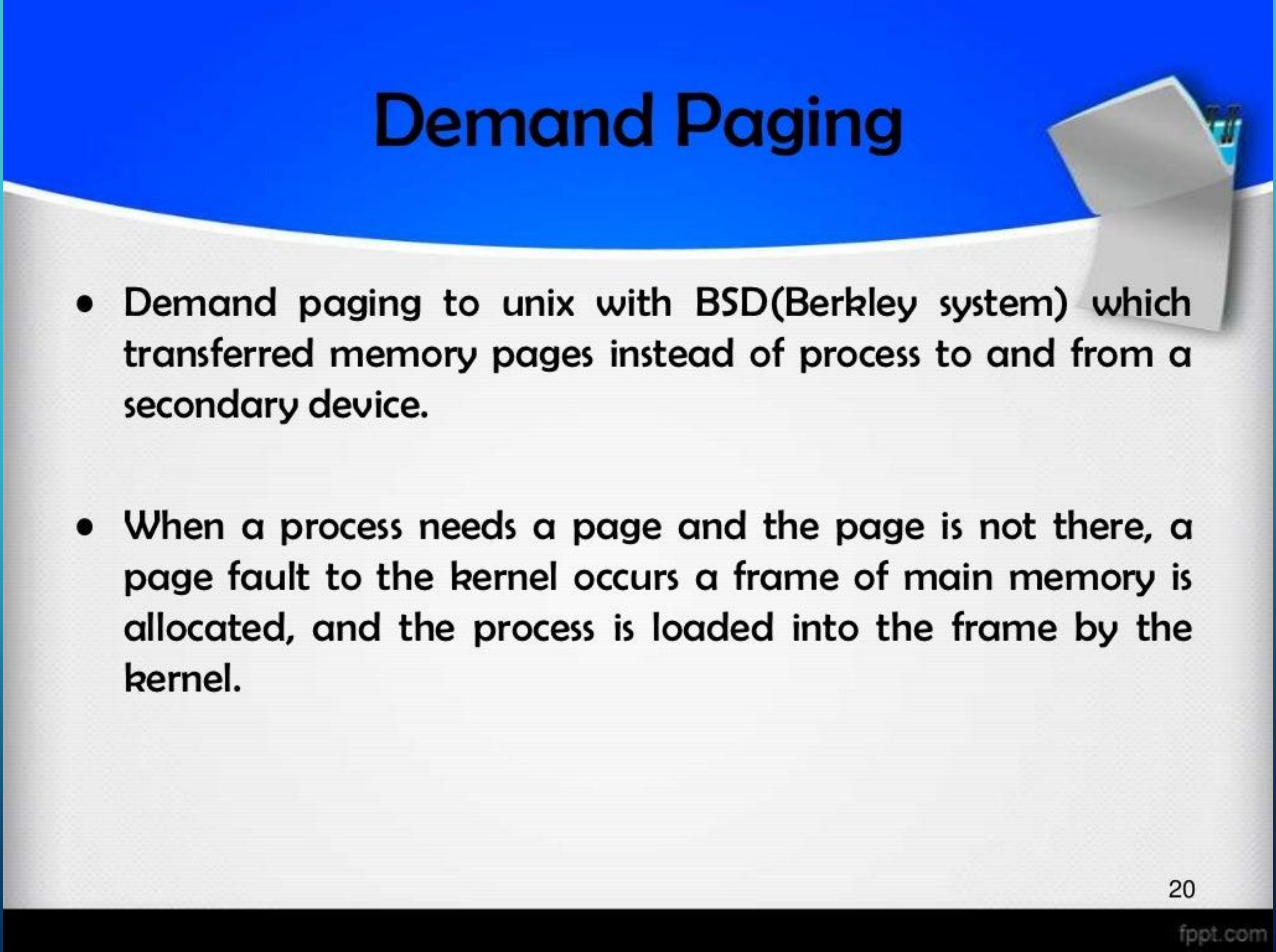
Case 2: Free resources fill a hole,

and immediately precedes an entry in the map



Freeing Swap Space

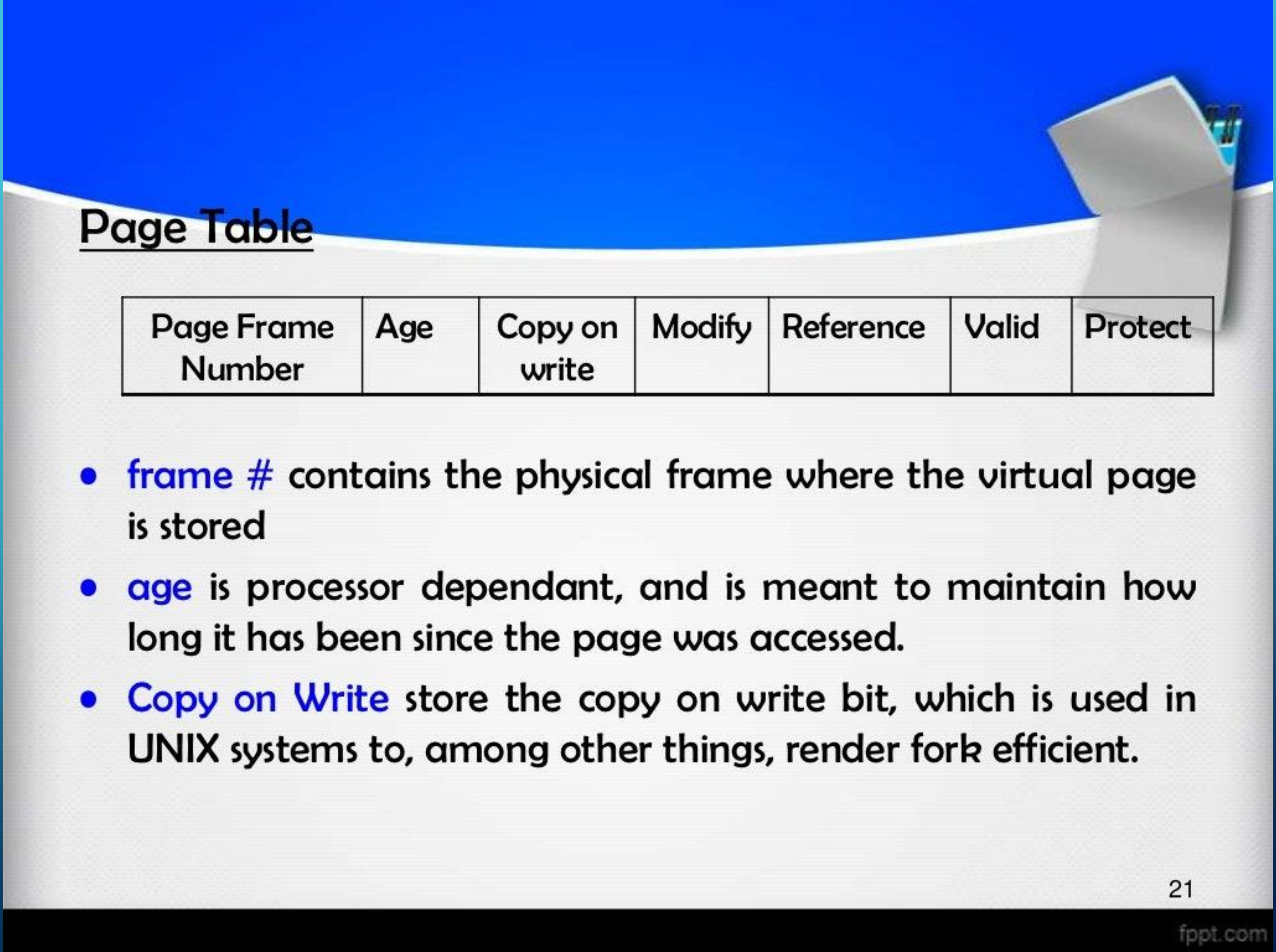
Case 3: Free resources fill a hole, and completely fills the gap between entries in the map



Demand Paging

- Demand paging to unix with BSD(Berkley system) which transferred memory pages instead of process to and from a secondary device.

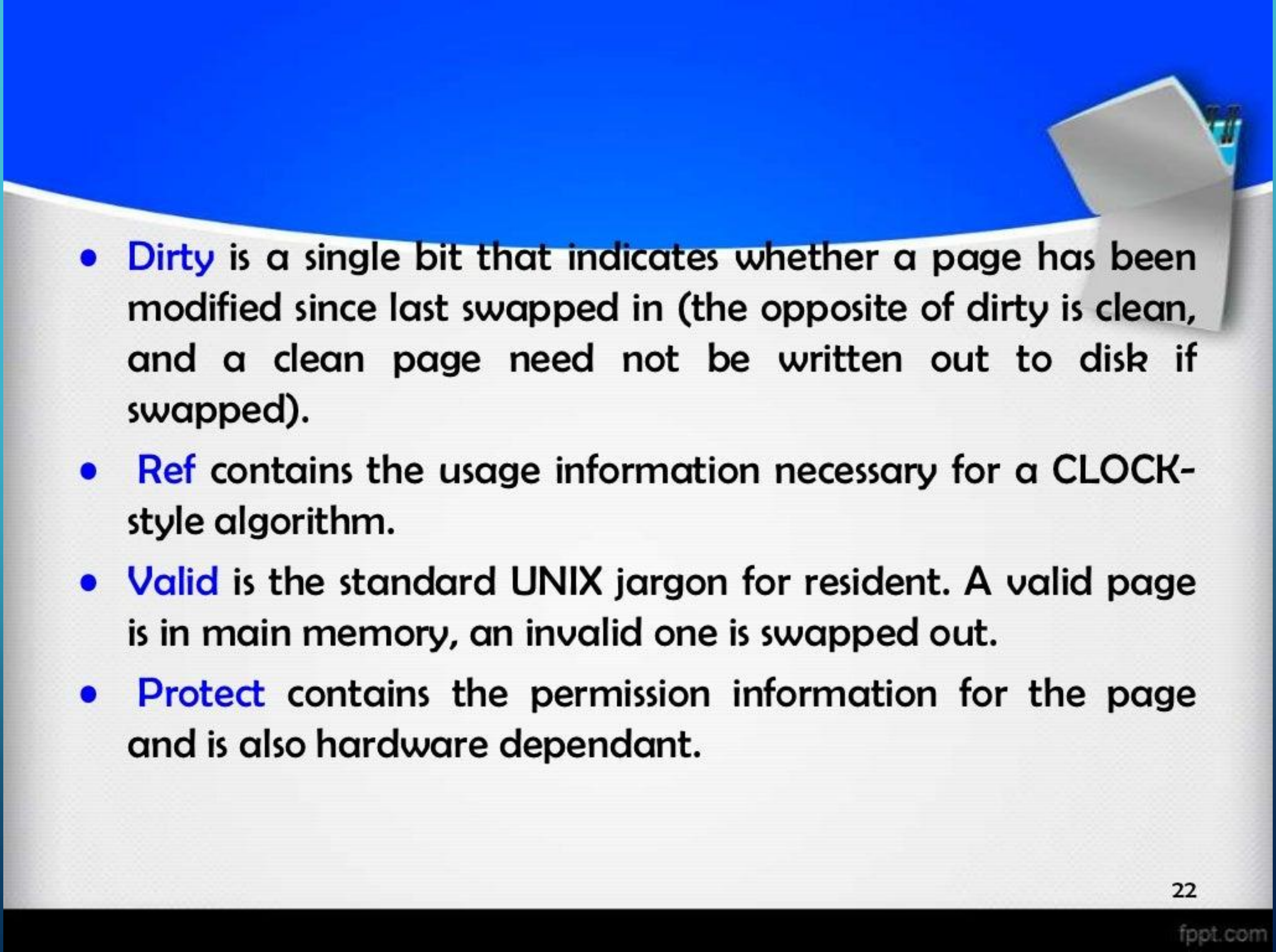
- When a process needs a page and the page is not there, a page fault to the kernel occurs a frame of main memory is allocated, and the process is loaded into the frame by the kernel.



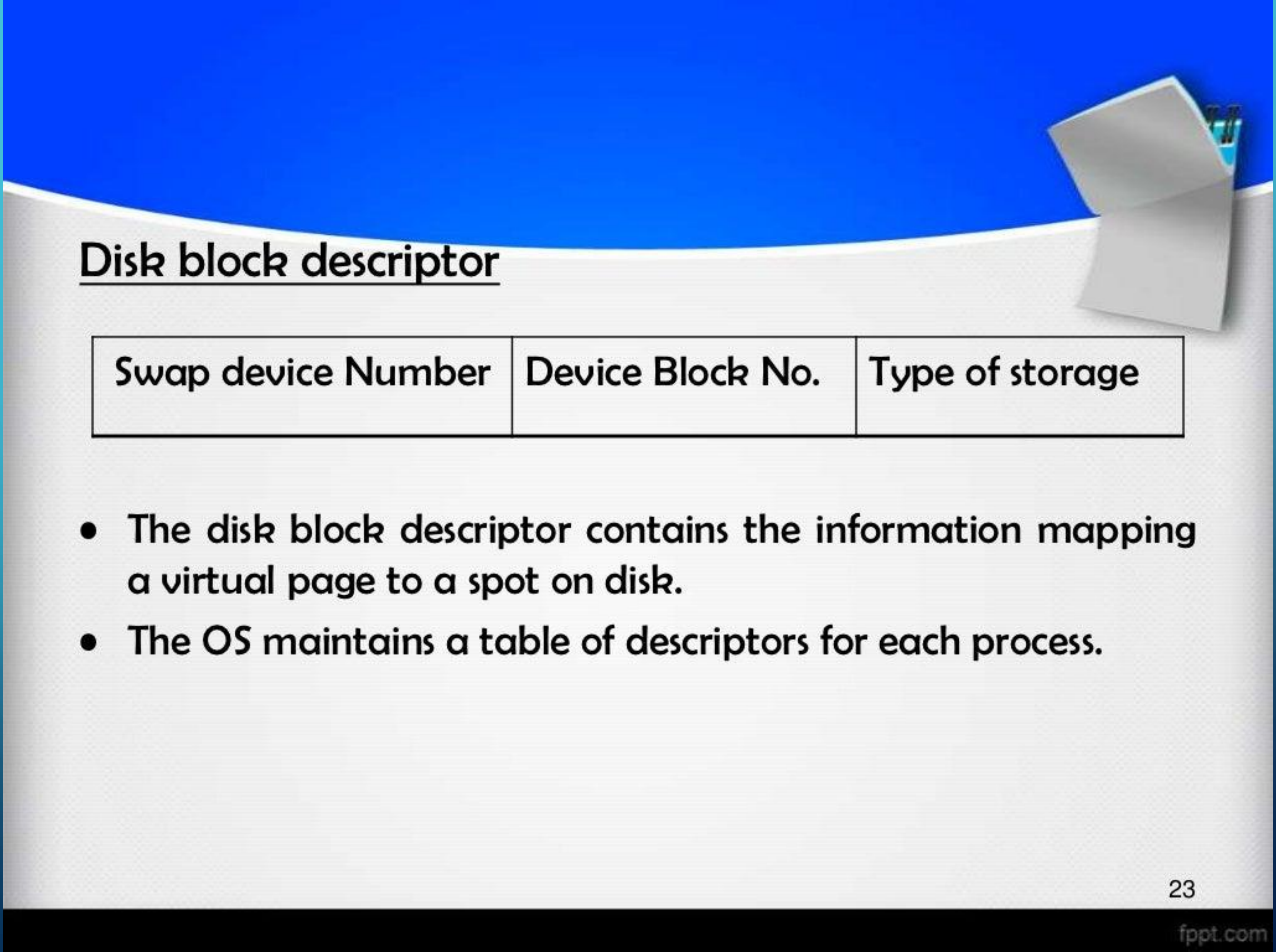
Page Table

Page Frame Number	Age	Copy on write	Modify	Reference	Valid	Protect

- **frame #** contains the physical frame where the virtual page is stored
- **age** is processor dependant, and is meant to maintain how long it has been since the page was accessed.
- **Copy on Write** store the copy on write bit, which is used in UNIX systems to, among other things, render fork efficient.



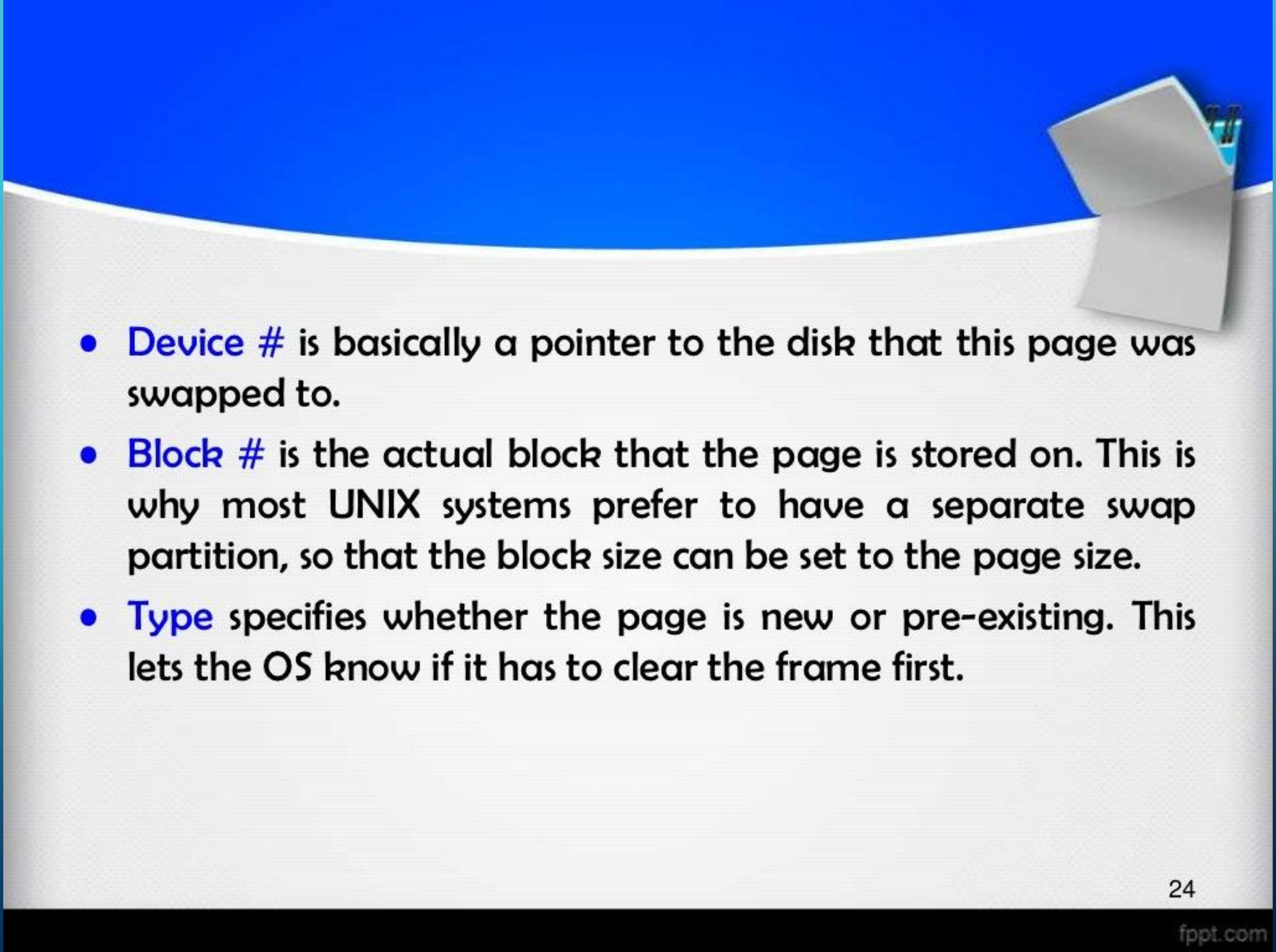
- Dirty is a single bit that indicates whether a page has been modified since last swapped in (the opposite of dirty is clean, and a clean page need not be written out to disk if swapped).
- Ref contains the usage information necessary for a CLOCK-style algorithm.
- Valid is the standard UNIX jargon for resident. A valid page is in main memory, an invalid one is swapped out.
- Protect contains the permission information for the page and is also hardware dependant.



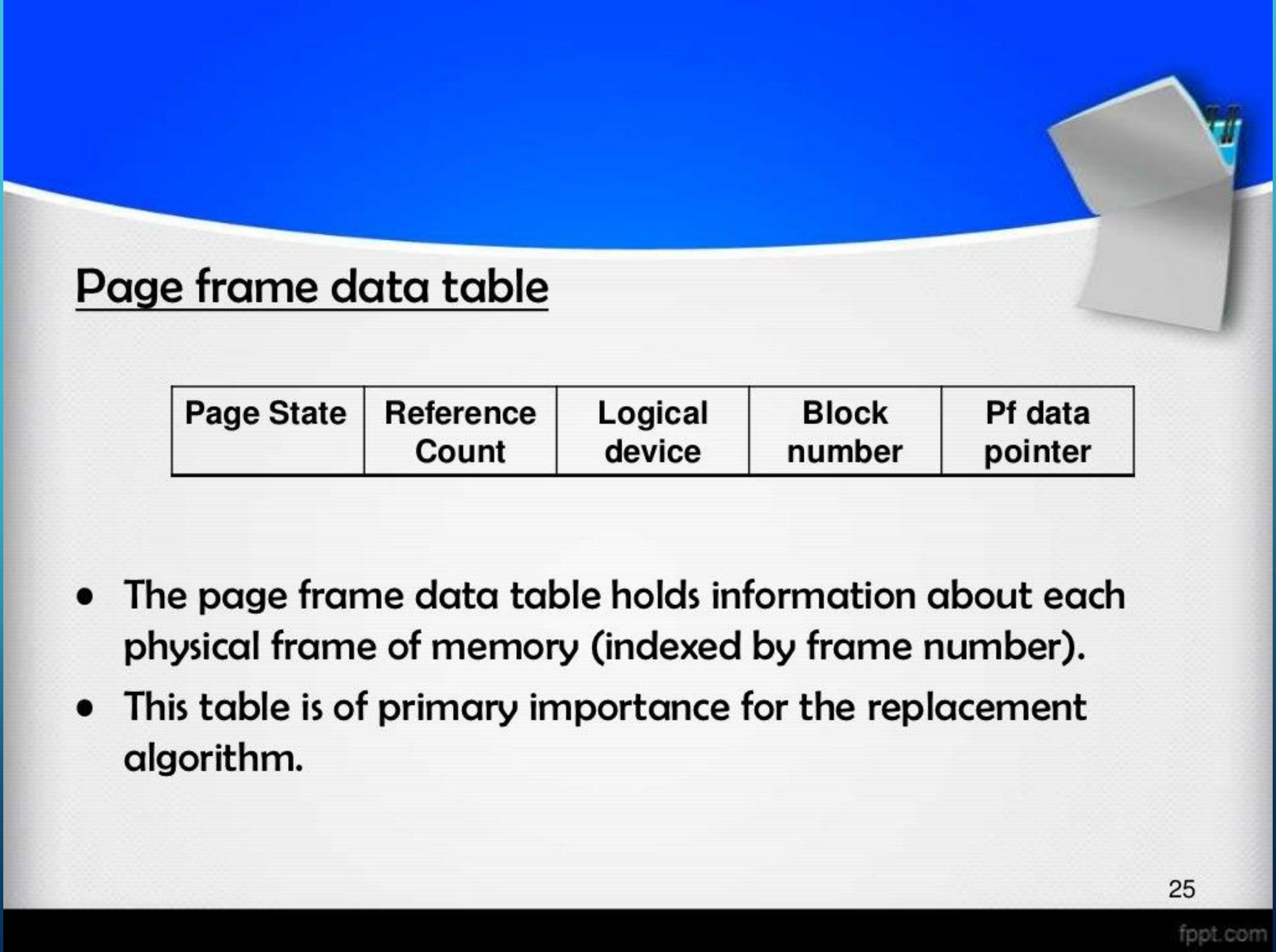
Disk block descriptor

Swap device Number	Device Block No.	Type of storage

- The disk block descriptor contains the information mapping a virtual page to a spot on disk.
- The OS maintains a table of descriptors for each process.



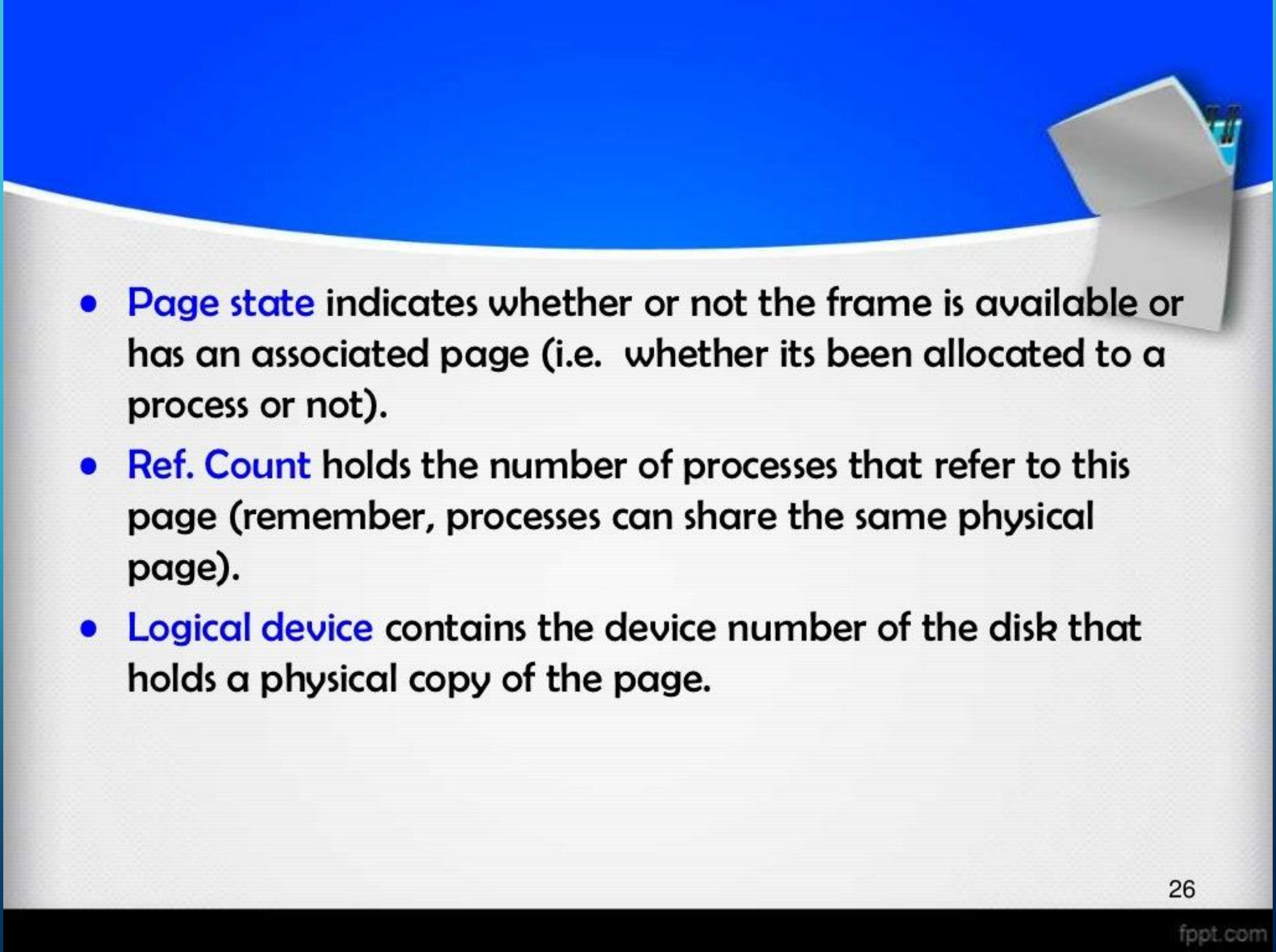
- **Device #** is basically a pointer to the disk that this page was swapped to.
- **Block #** is the actual block that the page is stored on. This is why most UNIX systems prefer to have a separate swap partition, so that the block size can be set to the page size.
- **Type** specifies whether the page is new or pre-existing. This lets the OS know if it has to clear the frame first.



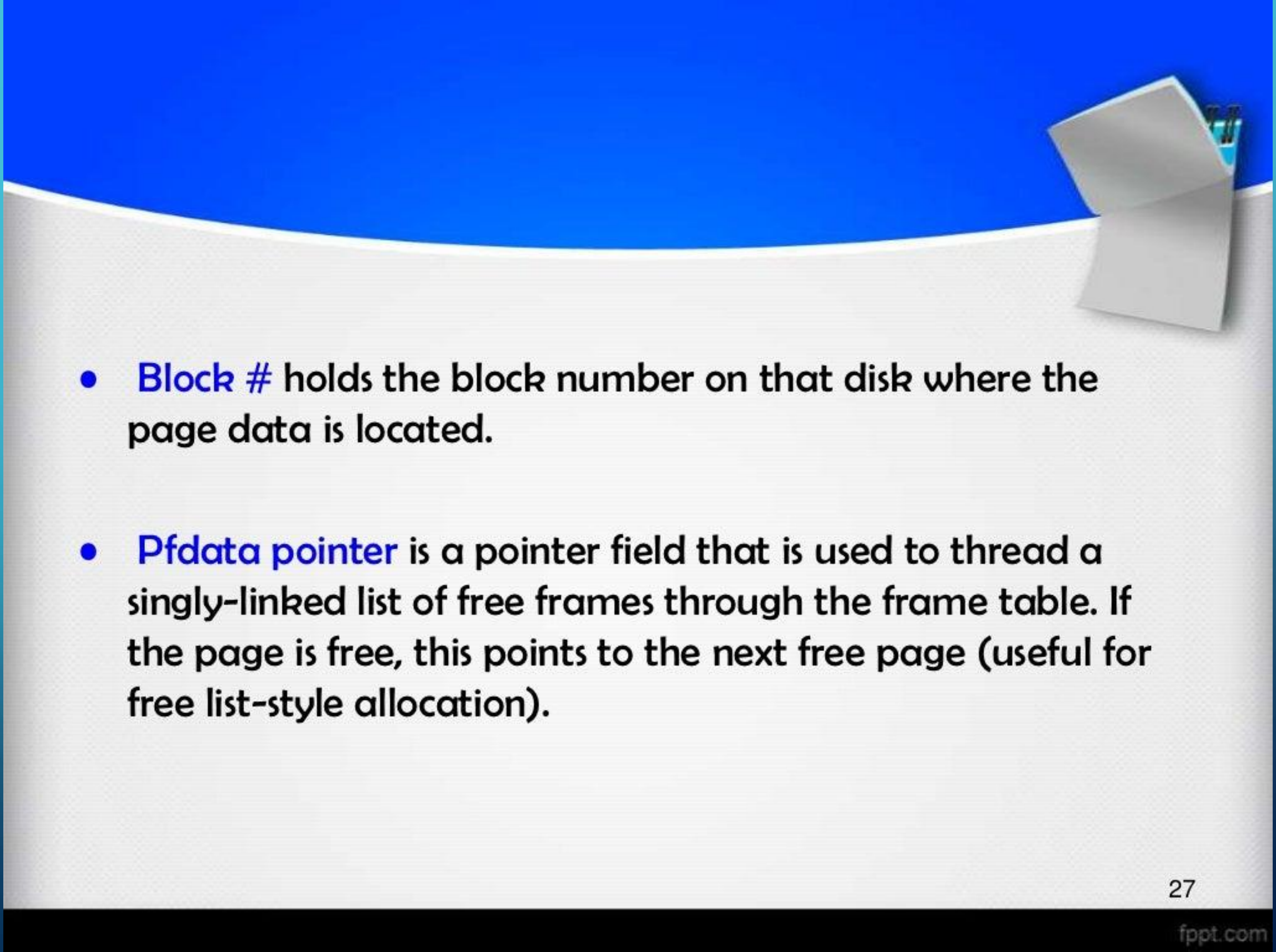
Page frame data table

Page State	Reference Count	Logical device	Block number	Pf data pointer

- The page frame data table holds information about each physical frame of memory (indexed by frame number).
- This table is of primary importance for the replacement algorithm.

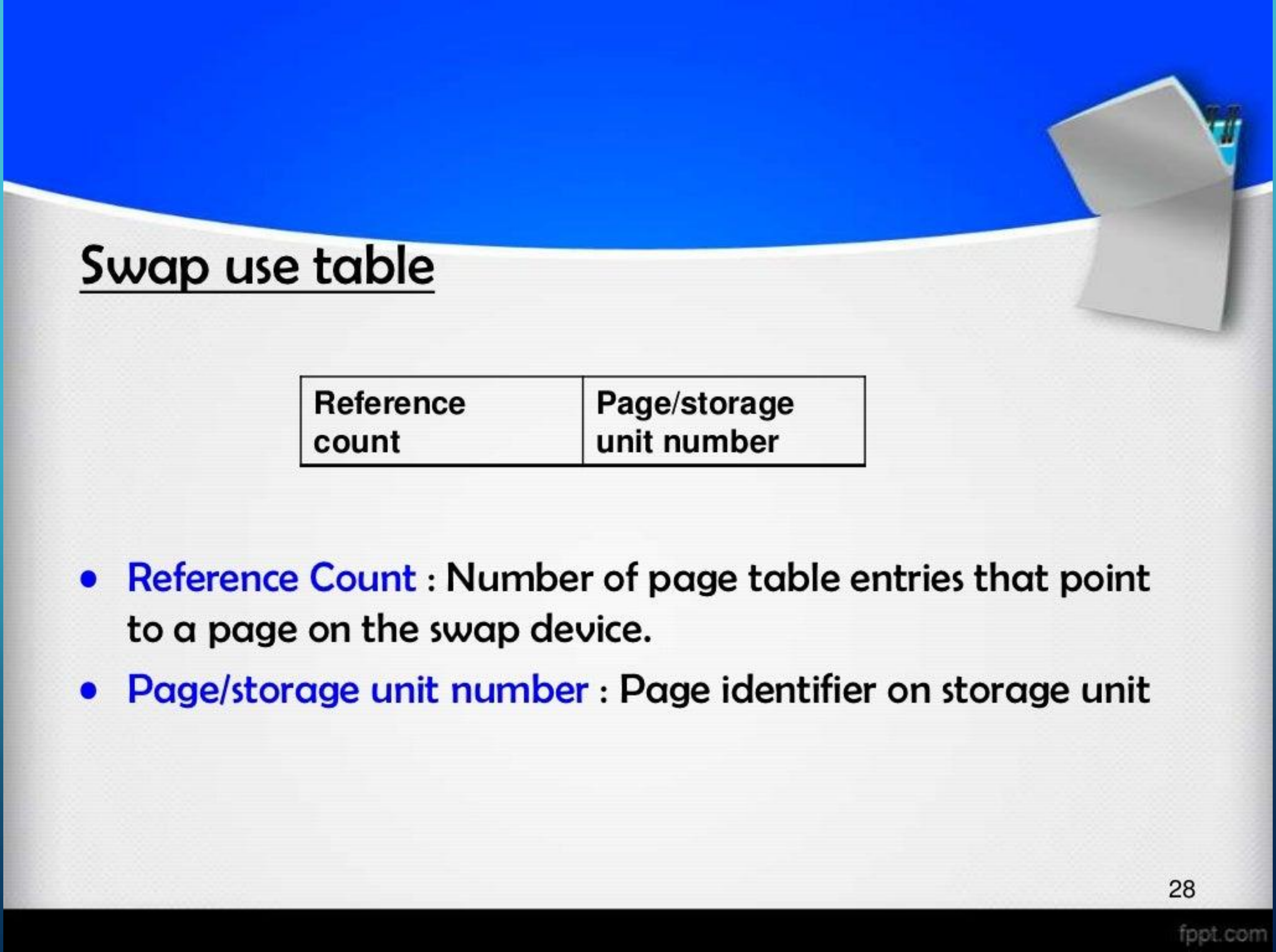


- **Page state** indicates whether or not the frame is available or has an associated page (i.e. whether its been allocated to a process or not).
- **Ref. Count** holds the number of processes that refer to this page (remember, processes can share the same physical page).
- **Logical device** contains the device number of the disk that holds a physical copy of the page.



- Block # holds the block number on that disk where the page data is located.

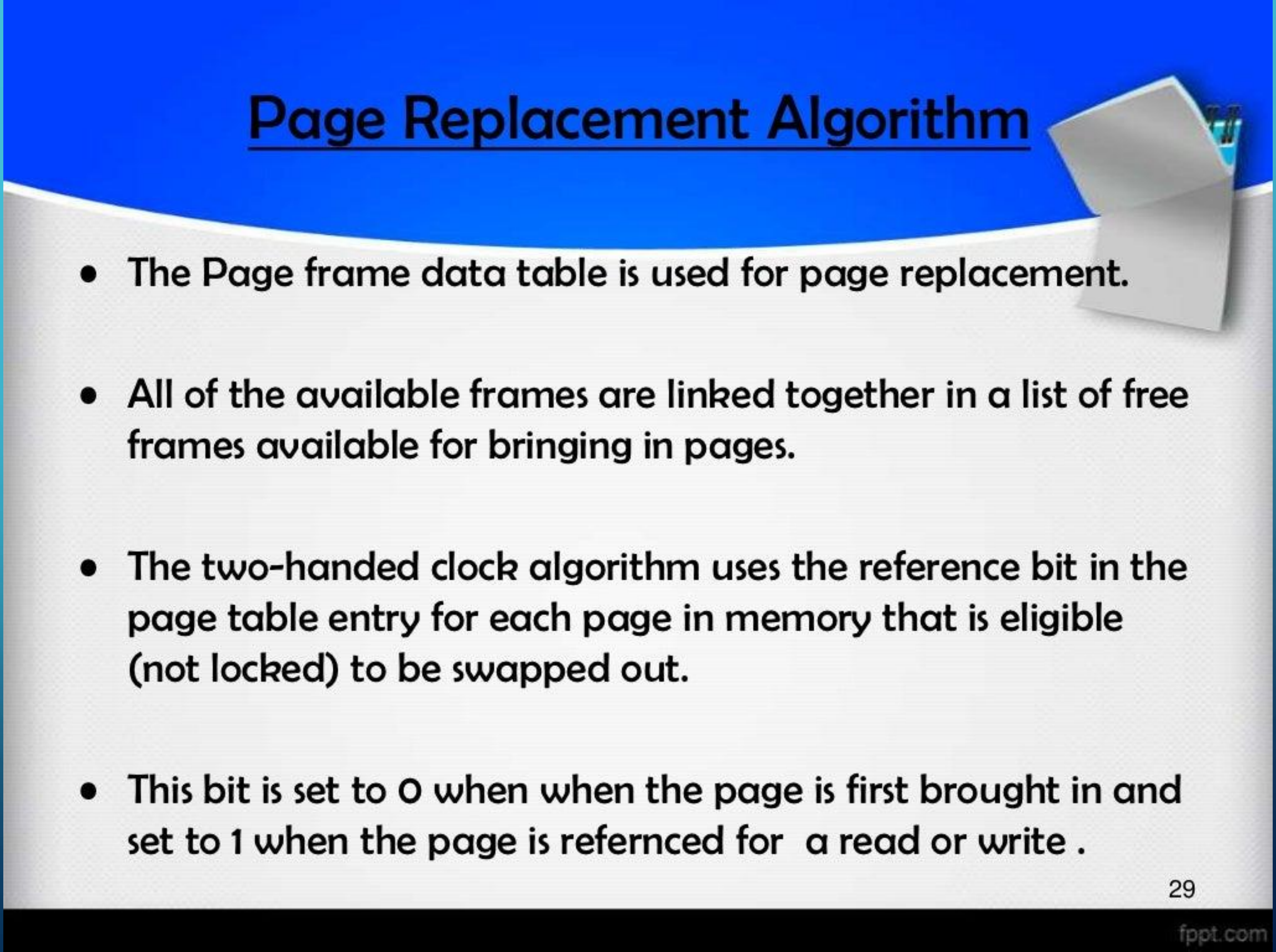
- Pfdata pointer is a pointer field that is used to thread a singly-linked list of free frames through the frame table. If the page is free, this points to the next free page (useful for free list-style allocation).



Swap use table

Reference count	Page/storage unit number

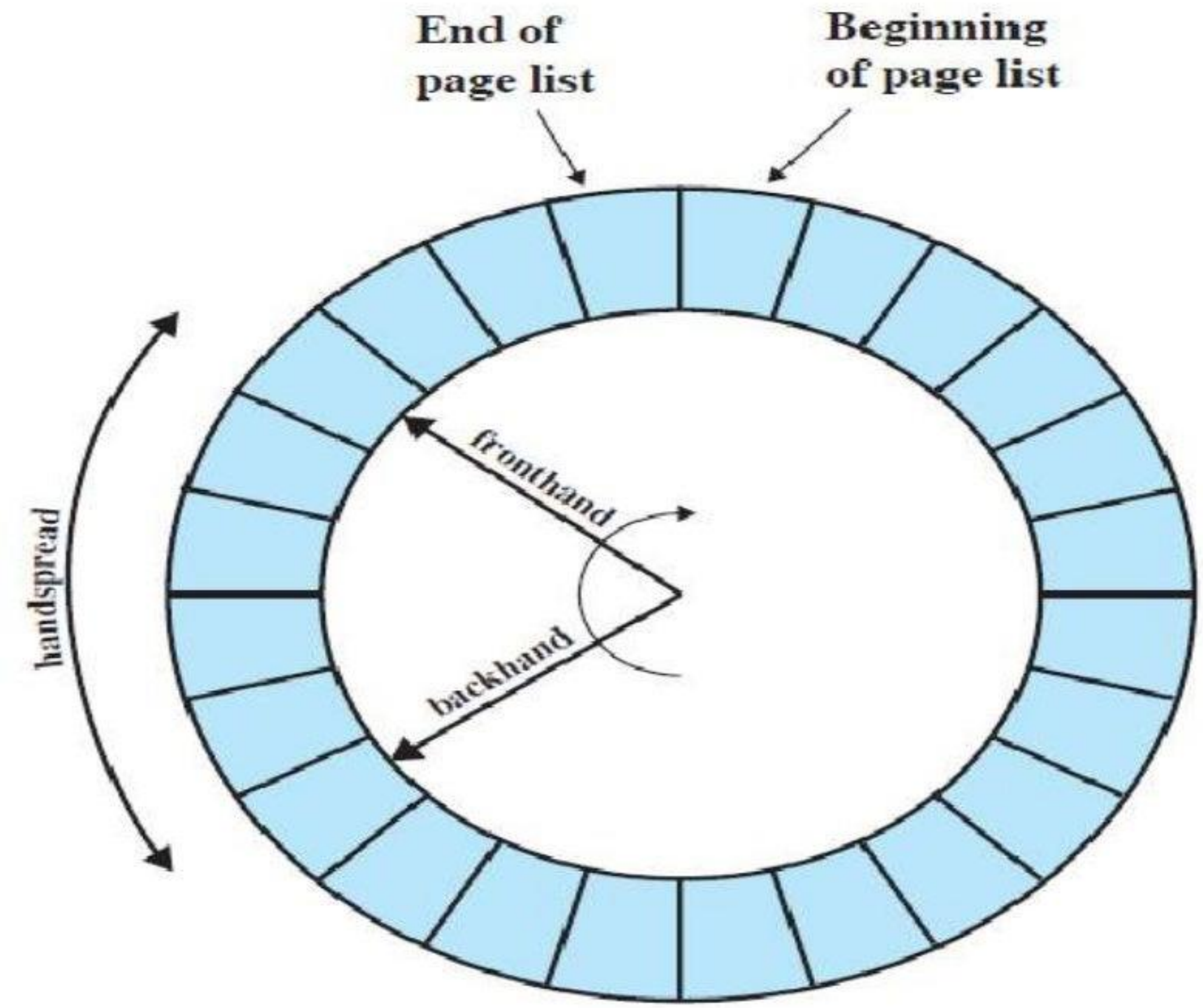
- Reference Count : Number of page table entries that point to a page on the swap device.
- Page/storage unit number : Page identifier on storage unit

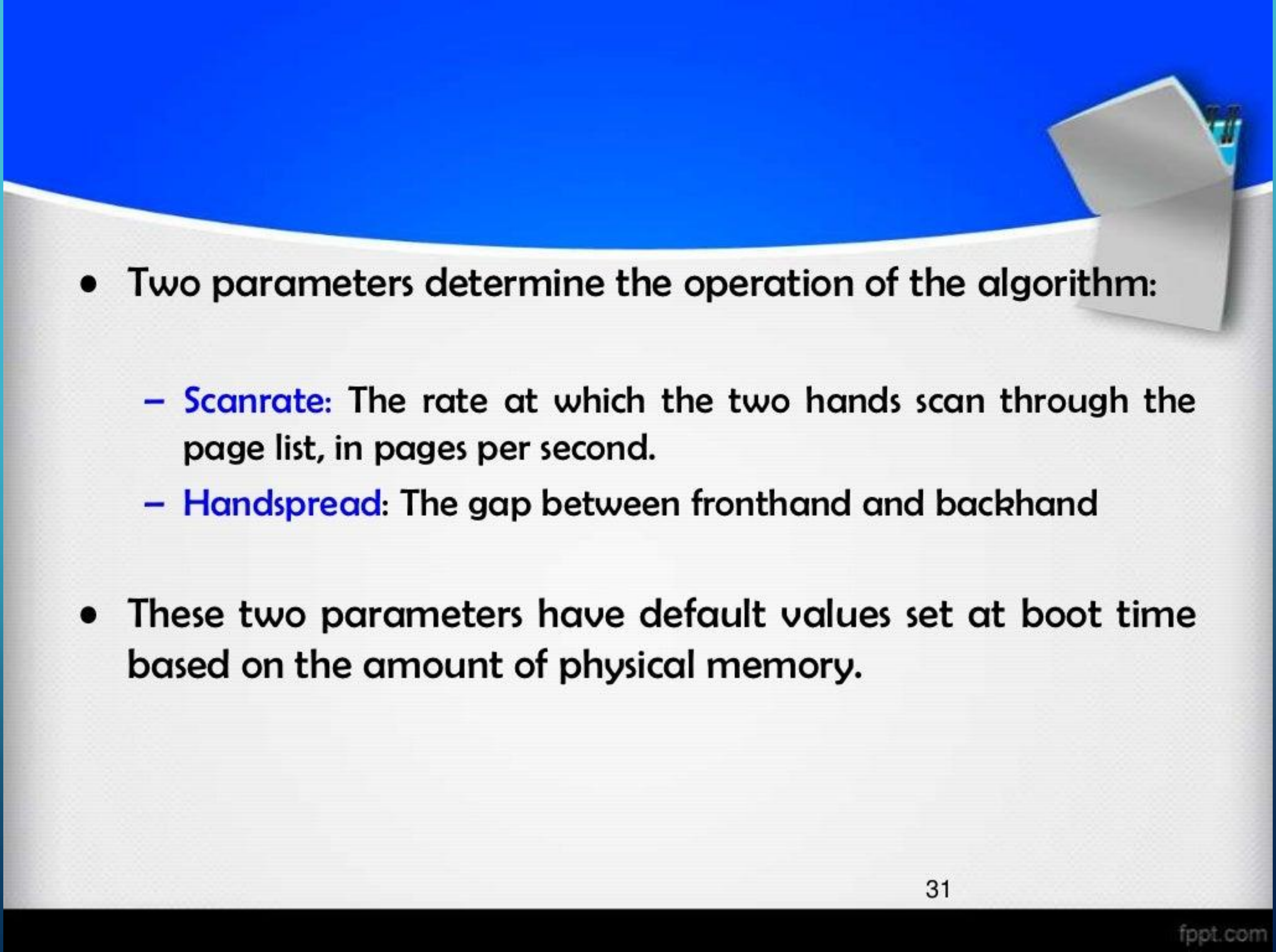


Page Replacement Algorithm

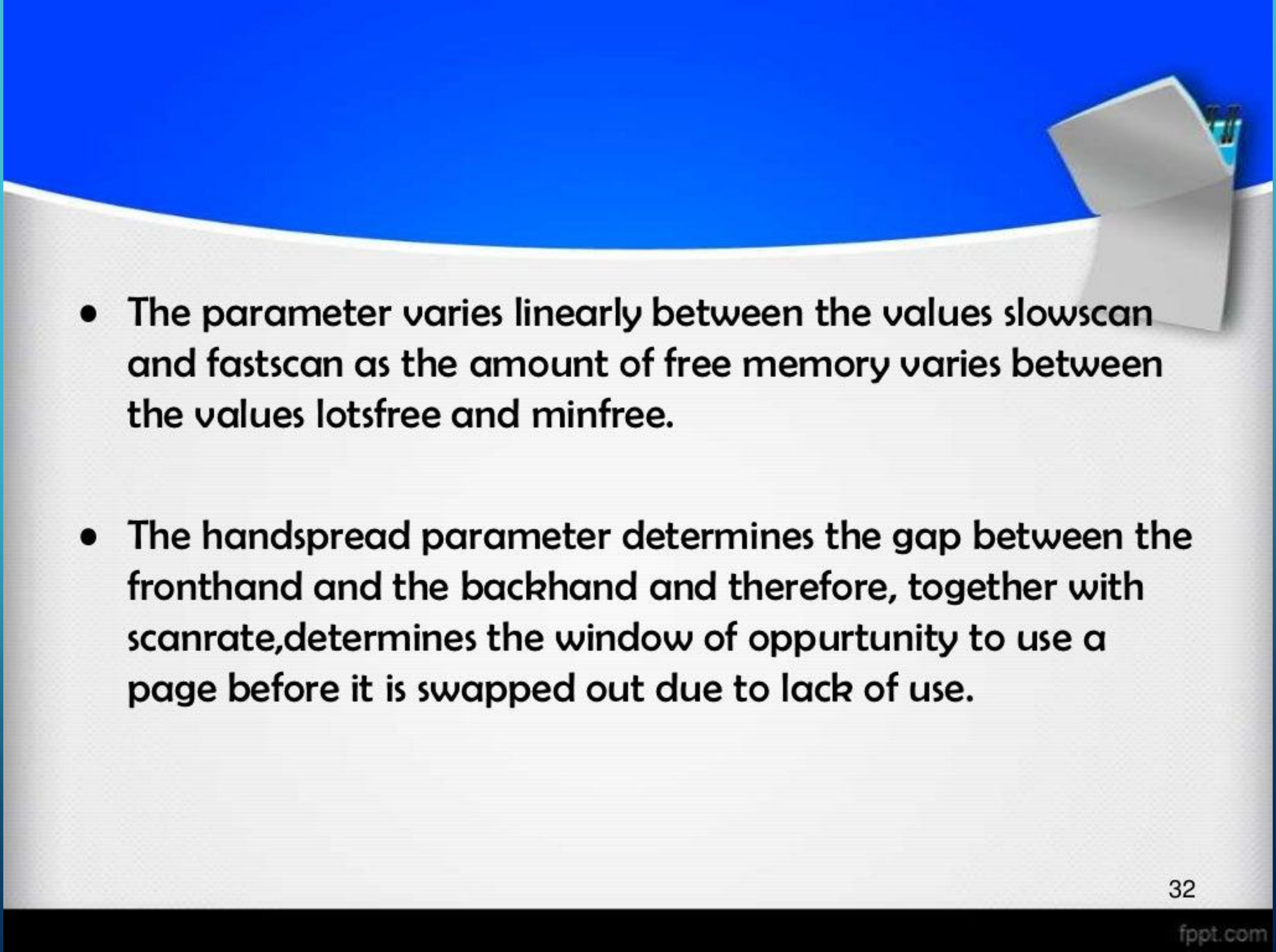
- The Page frame data table is used for page replacement.
- All of the available frames are linked together in a list of free frames available for bringing in pages.
- The two-handed clock algorithm uses the reference bit in the page table entry for each page in memory that is eligible (not locked) to be swapped out.
- This bit is set to 0 when when the page is first brought in and set to 1 when the page is refernced for a read or write .

Two-Handed clock Page Replacement Algorithm



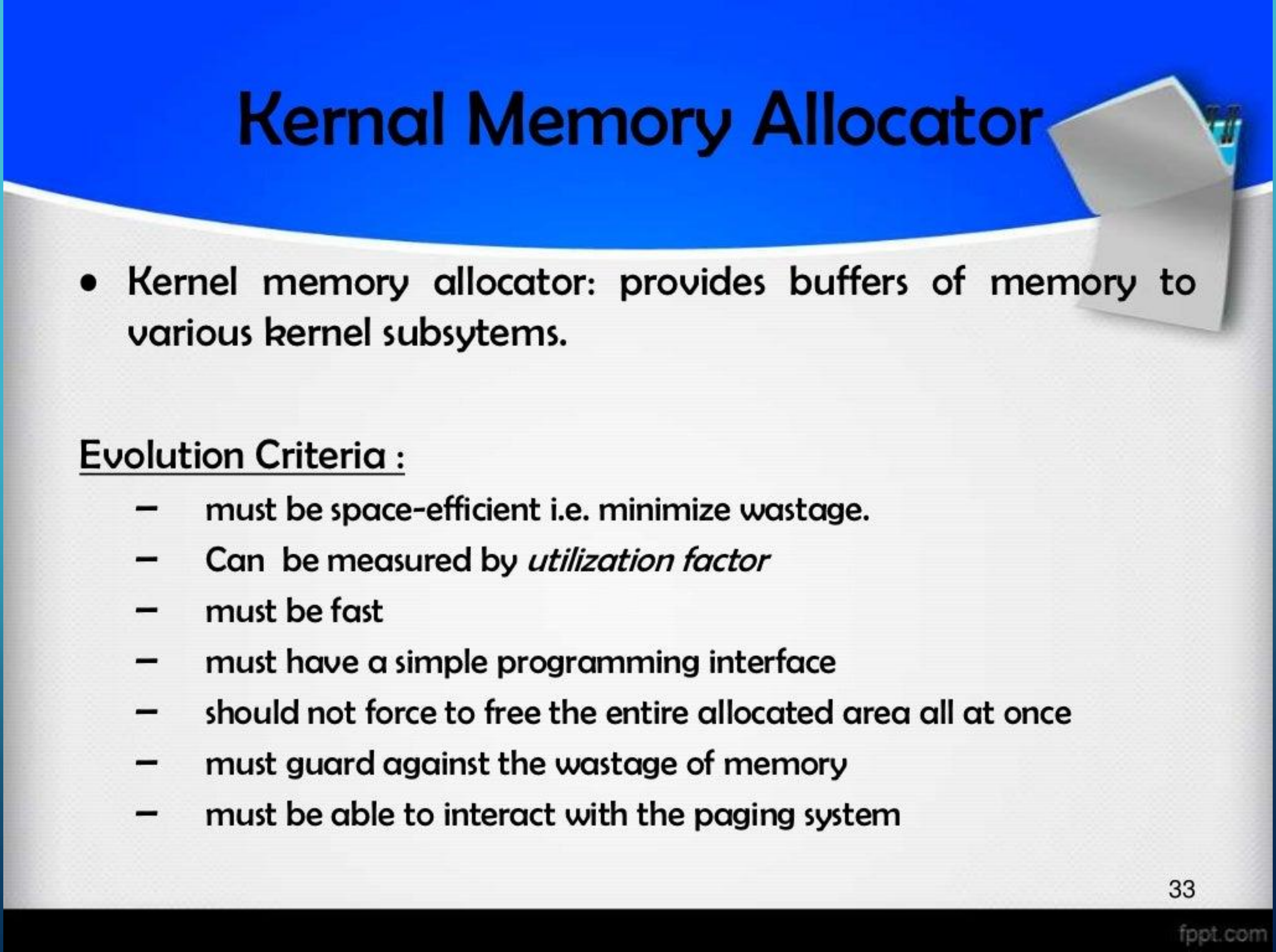


- Two parameters determine the operation of the algorithm:
  - **Scanrate:** The rate at which the two hands scan through the page list, in pages per second.
  - **Handspread**: The gap between fronthand and backhand
- These two parameters have default values set at boot time based on the amount of physical memory.



- The parameter varies linearly between the values slowscan and fastscan as the amount of free memory varies between the values lotsfree and minfree.

- The handspread parameter determines the gap between the fronthand and the backhand and therefore, together with scanrate,determines the window of oppurtunity to use a page before it is swapped out due to lack of use.

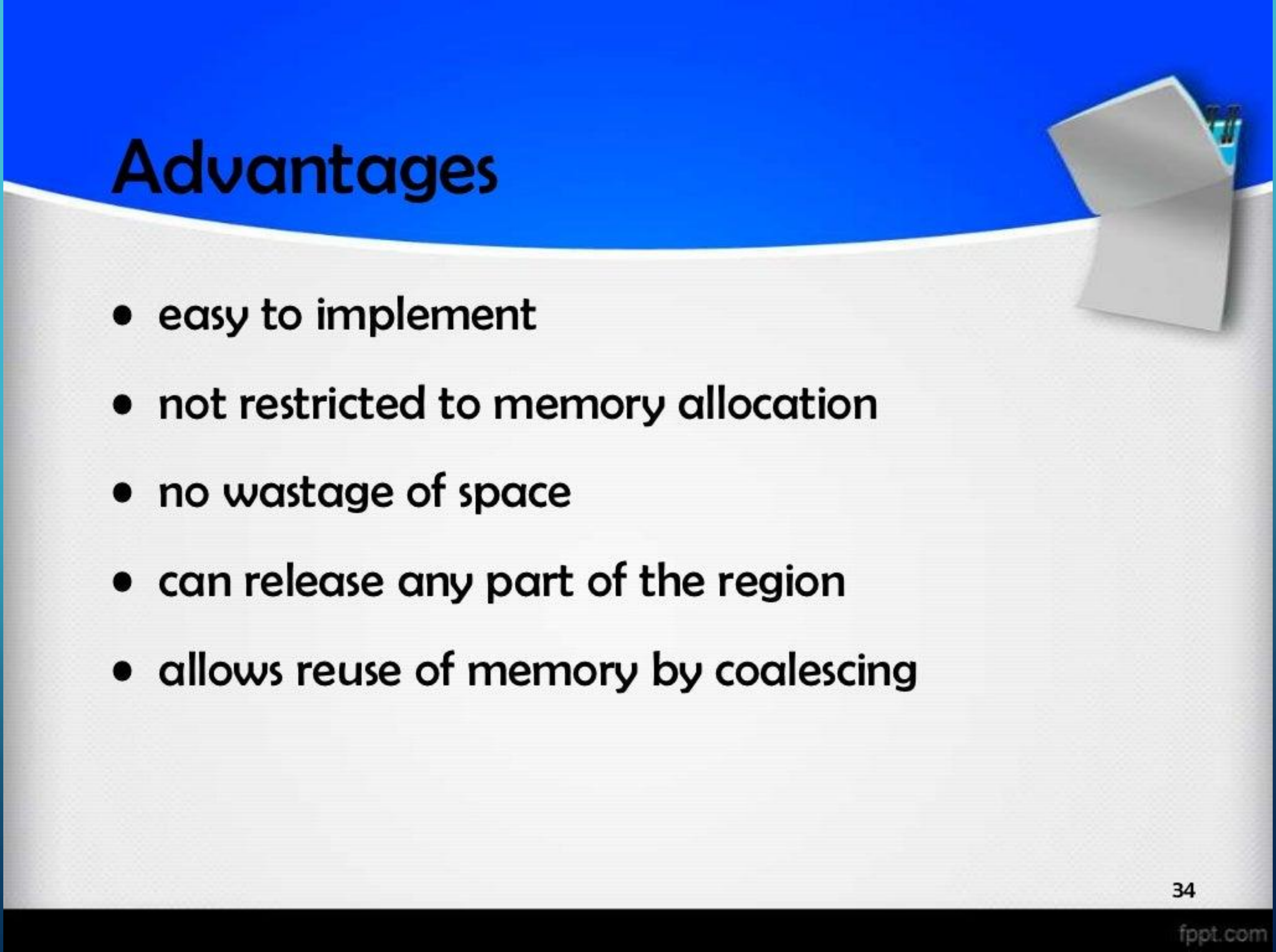


Kernal Memory Allocator

- Kernel memory allocator: provides buffers of memory to various kernel subsytems.

<u>Evolution Criteria :</u>

- must be space-efficient i.e. minimize wastage.
- Can be measured by *utilization factor*
- must be fast
- must have a simple programming interface
- should not force to free the entire allocated area all at once
- must guard against the wastage of memory
- must be able to interact with the paging system



Advantages

- easy to implement
- not restricted to memory allocation
- no wastage of space
- can release any part of the region
- allows reuse of memory by coalescing

WEEK: 05
UNIX MEMORY MANAGEMENT 02

MEMORY MANAGEMENT IN LINUX

Unix Memory Management

Memory is an important resource in computer. Memory management is the process of managing the computer memory which consists of primary memory and secondary memory.

- The goal for memory management is to keep track of which parts of memory are in use and which parts are not in use, to allocate memory to processes when they need it and de-allocate it when they are done. UNIX memory management scheme includes swapping and demand paging.

Memory Partitioning
The simplest form of memory management is splitting up the main memory into multiple logical spaces called partition. Each partition is used for separate program.

There are 2 types of memory partitioning:-

- Single Partition Allocation
- Multiple Partition Allocation

Unix Memory Management

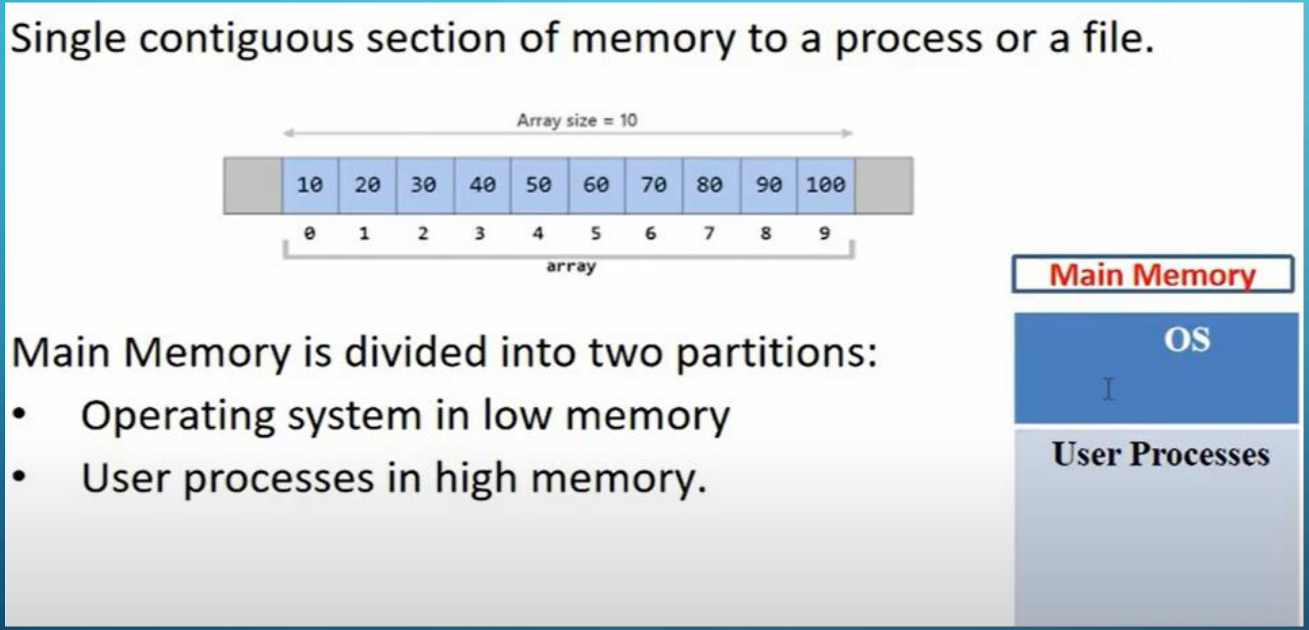
- UNIX is machine independent so it's memory management scheme will vary from one system to next
- Early versions of UNIX used variable partitioning with no virtual memory scheme.
- Current implementations of UNIX make use of paged virtual memory.

There are two memory management schemes:

- Paging system
- Kernal Memory Allocator

Single Partition Allocation

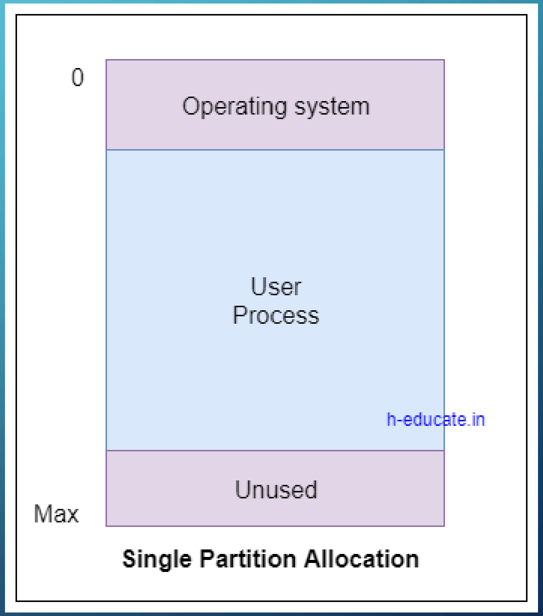
Single partition allocation only separates the main memory into operating system and one user process area. Operating system will not able to have virtual memory using single partition. Using single partition is very ineffective because it only allows one process to run in the memory at one time.



Single contiguous section of memory to a process or a file.

Main Memory is divided into two partitions:

- Operating system in low memory
- User processes in high memory.



Single Partition Allocation

Multiple Partition Allocation

- Most of the operating system nowadays is using multiple partitions because it is more flexible. Multiple partition allocation enabled multiple programs run in the main memory at once. Each partition is used for one process.

- There are two different forms of multiple partition allocation
  - Fixed partitioning
  - Variable partitioning.

- Fixed partitioning divides memory up into many fixed partitions which cannot be change.

- However, variable partitioning is more flexible because the partitions vary dynamically in the later as processes come and go. Variable partitioning (Variable memory) has been used in UNIX.

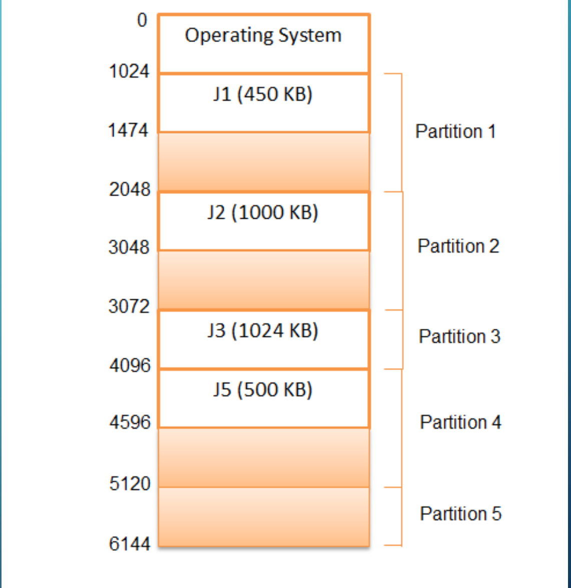
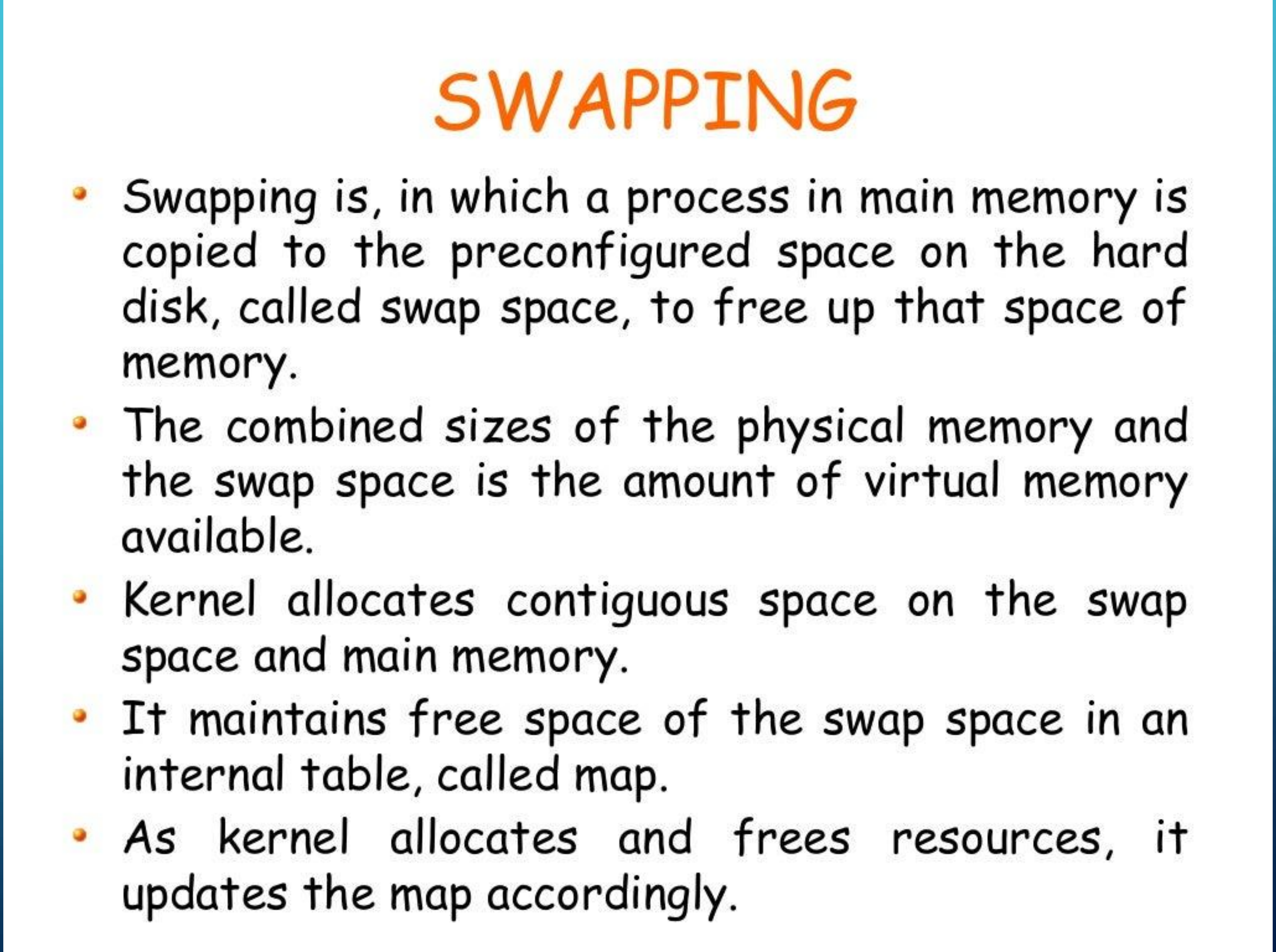


Fig: Memory Allocation for Multiple Partitioned

Memory Management Policies

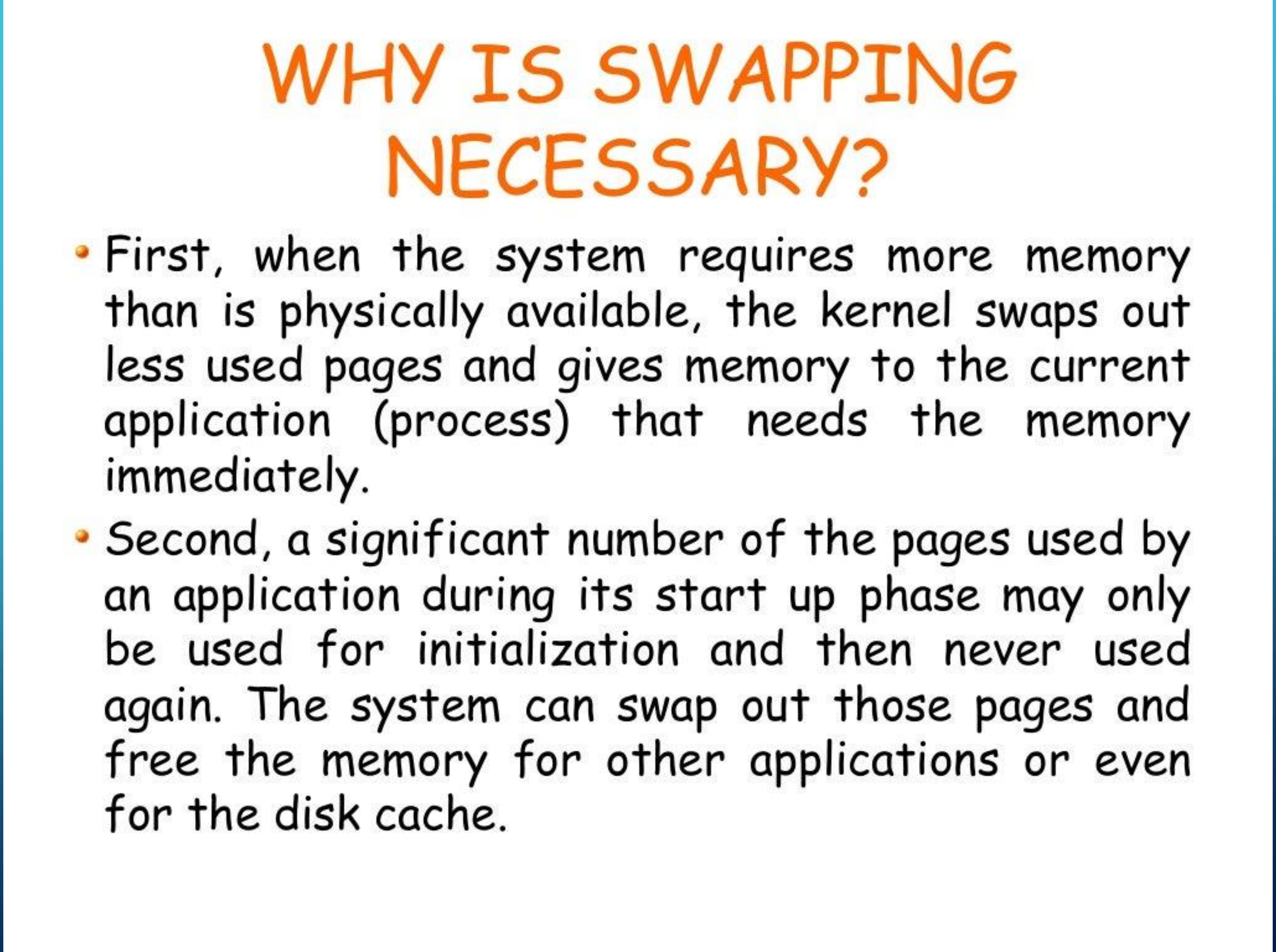
- ❑ **Swapping**
  - Easy to implement
  - Less system overhead

- ❑ **Demand paging**
  -Greater Flexibility



SWAPPING

- Swapping is, in which a process in main memory is copied to the preconfigured space on the hard disk, called swap space, to free up that space of memory.
- The combined sizes of the physical memory and the swap space is the amount of virtual memory available.
- Kernel allocates contiguous space on the swap space and main memory.
- It maintains free space of the swap space in an internal table, called map.
- As kernel allocates and frees resources, it updates the map accordingly.



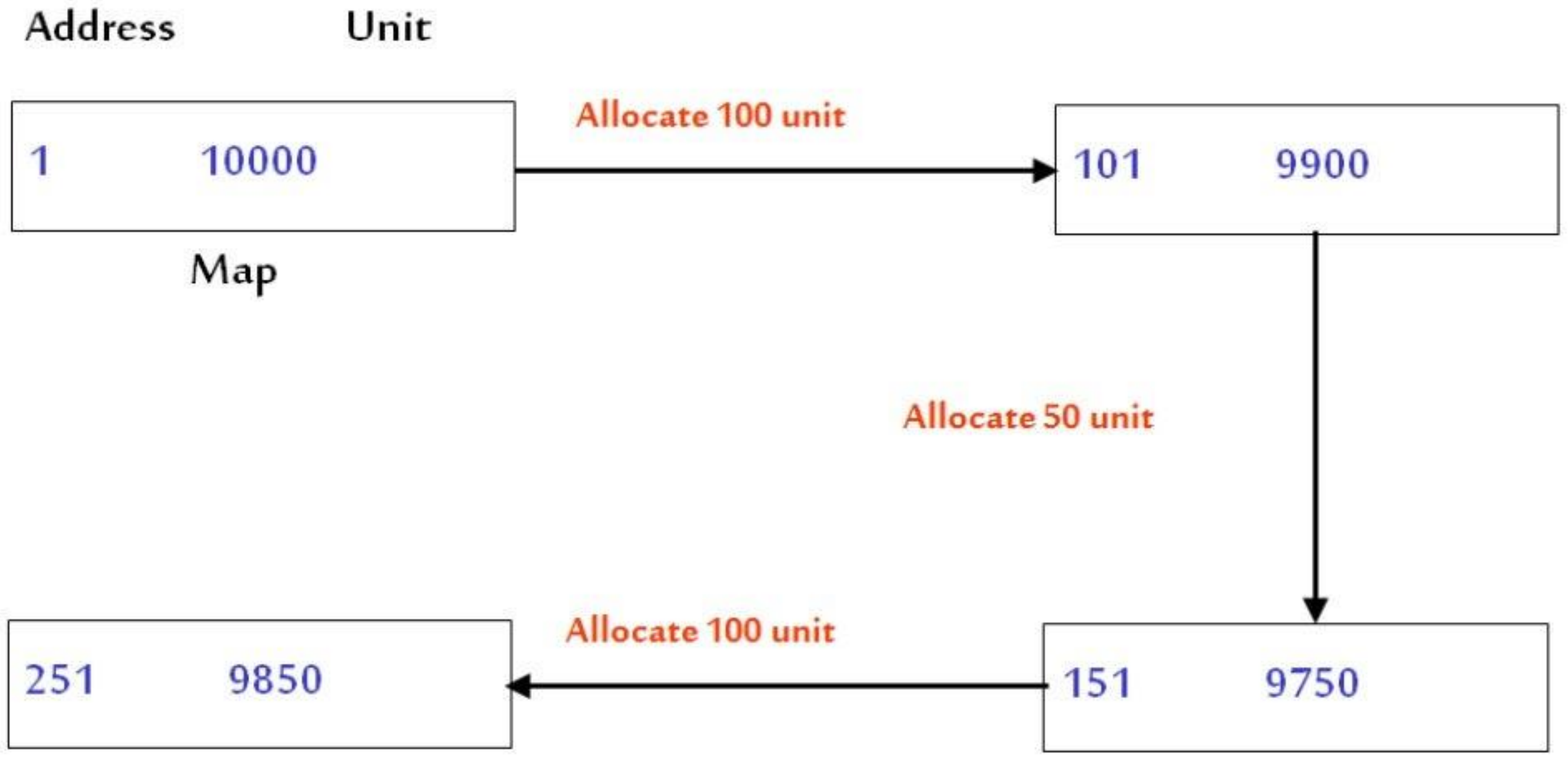
WHY IS SWAPPING NECESSARY?

- First, when the system requires more memory than is physically available, the kernel swaps out less used pages and gives memory to the current application (process) that needs the memory immediately.
- Second, a significant number of the pages used by an application during its start up phase may only be used for initialization and then never used again. The system can swap out those pages and free the memory for other applications or even for the disk cache.

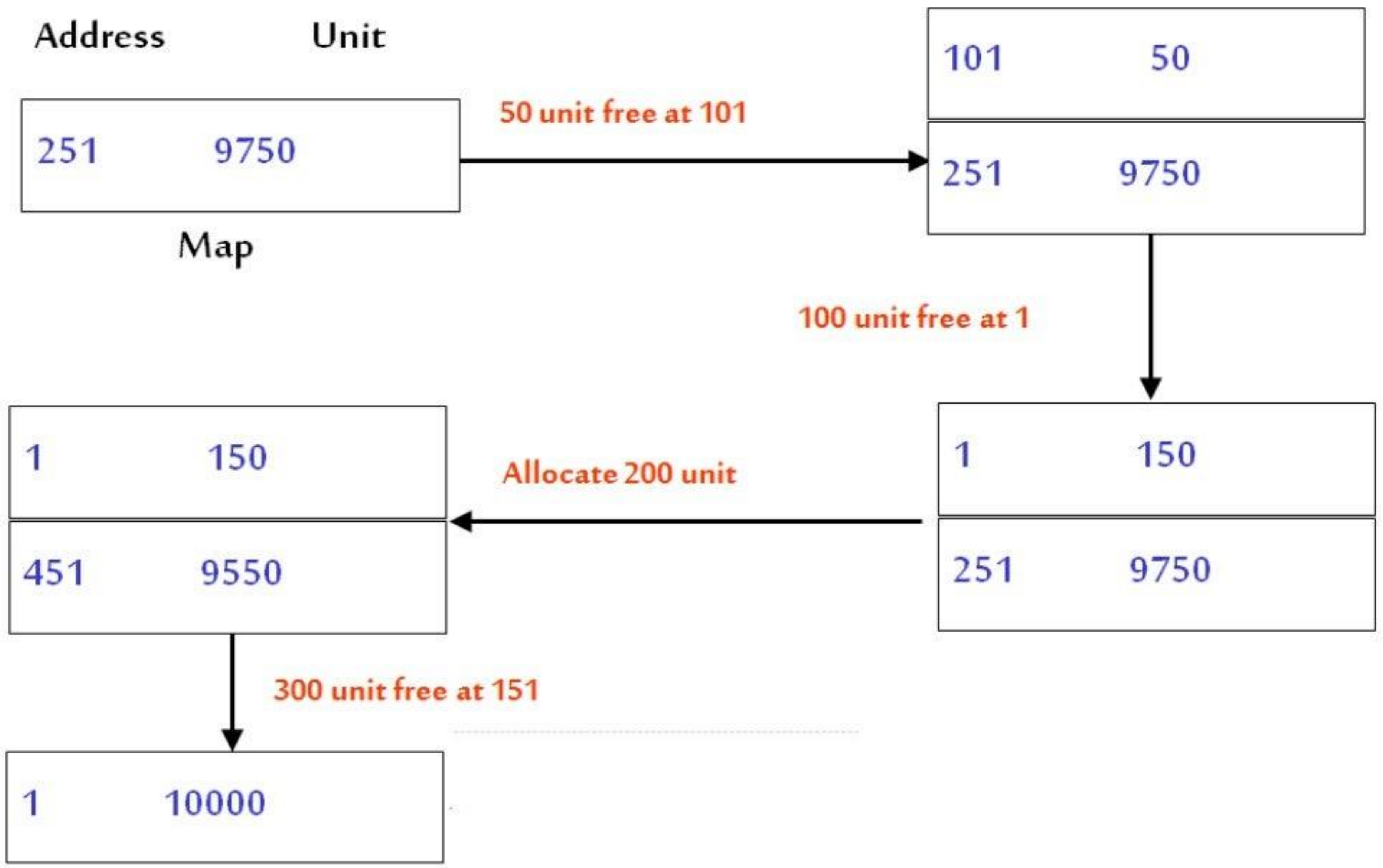
DOWNSIDE OF SWAPPING

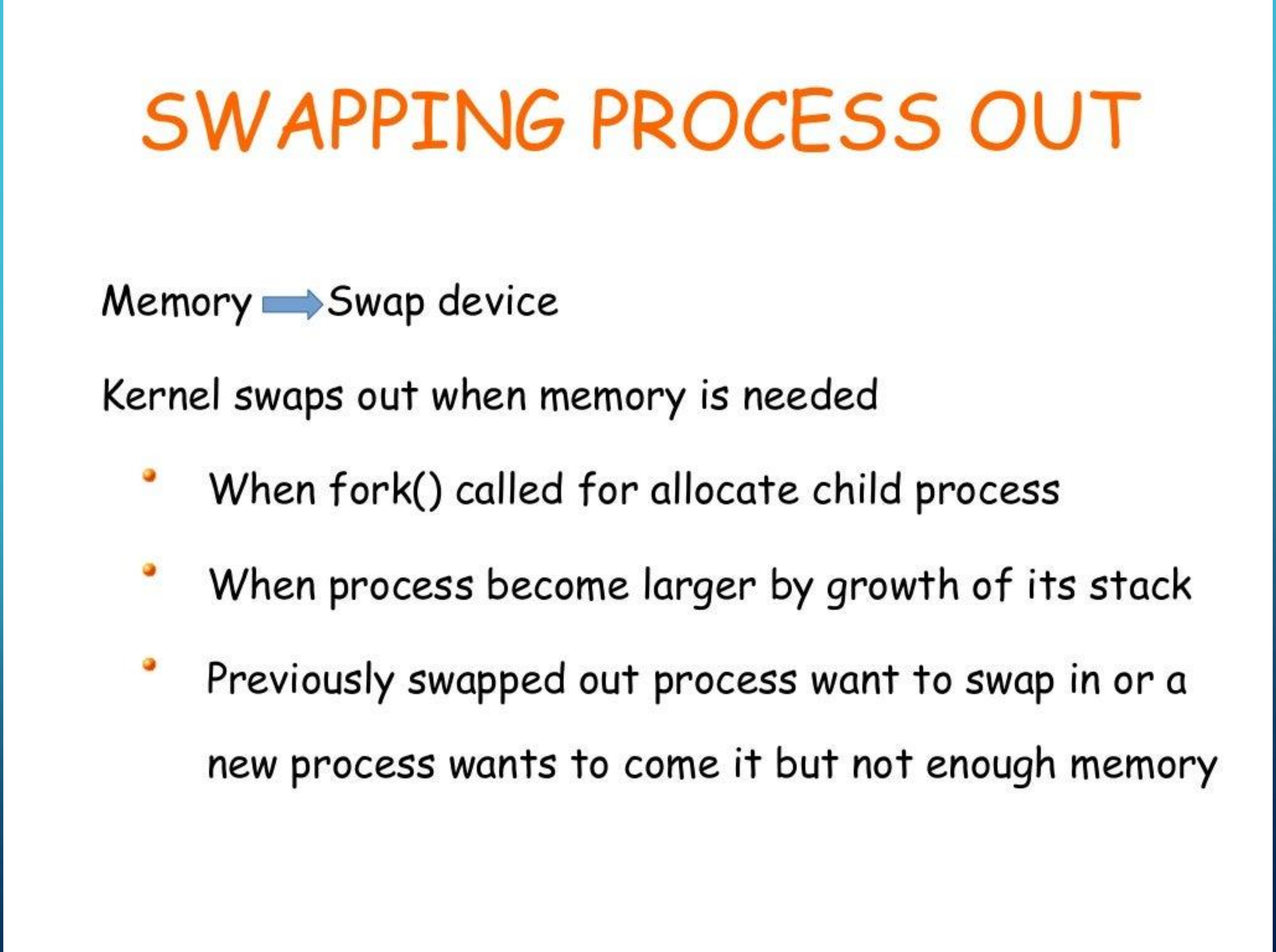
- Compared to memory, disks are very slow. So accessing the disk can be tens or thousands times slower than accessing physical memory. The more swapping that occurs, the slower your system will be.

ALLOCATING SWAP SPACE



FREEING SWAP SPACE





SWAPPING PROCESS OUT

Memory $\rightarrow$ Swap device

Kernel swaps out when memory is needed

- When fork() called for allocate child process
- When process become larger by growth of its stack
- Previously swapped out process want to swap in or a new process wants to come it but not enough memory

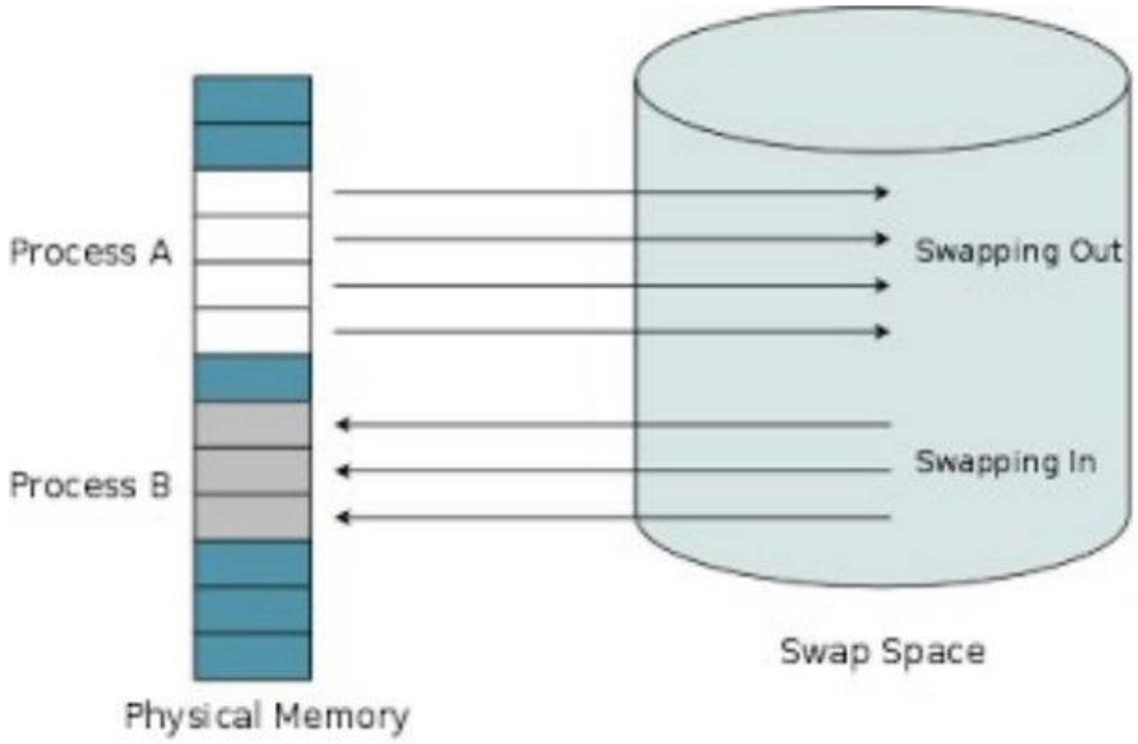
SWAPPING PROCESS IN

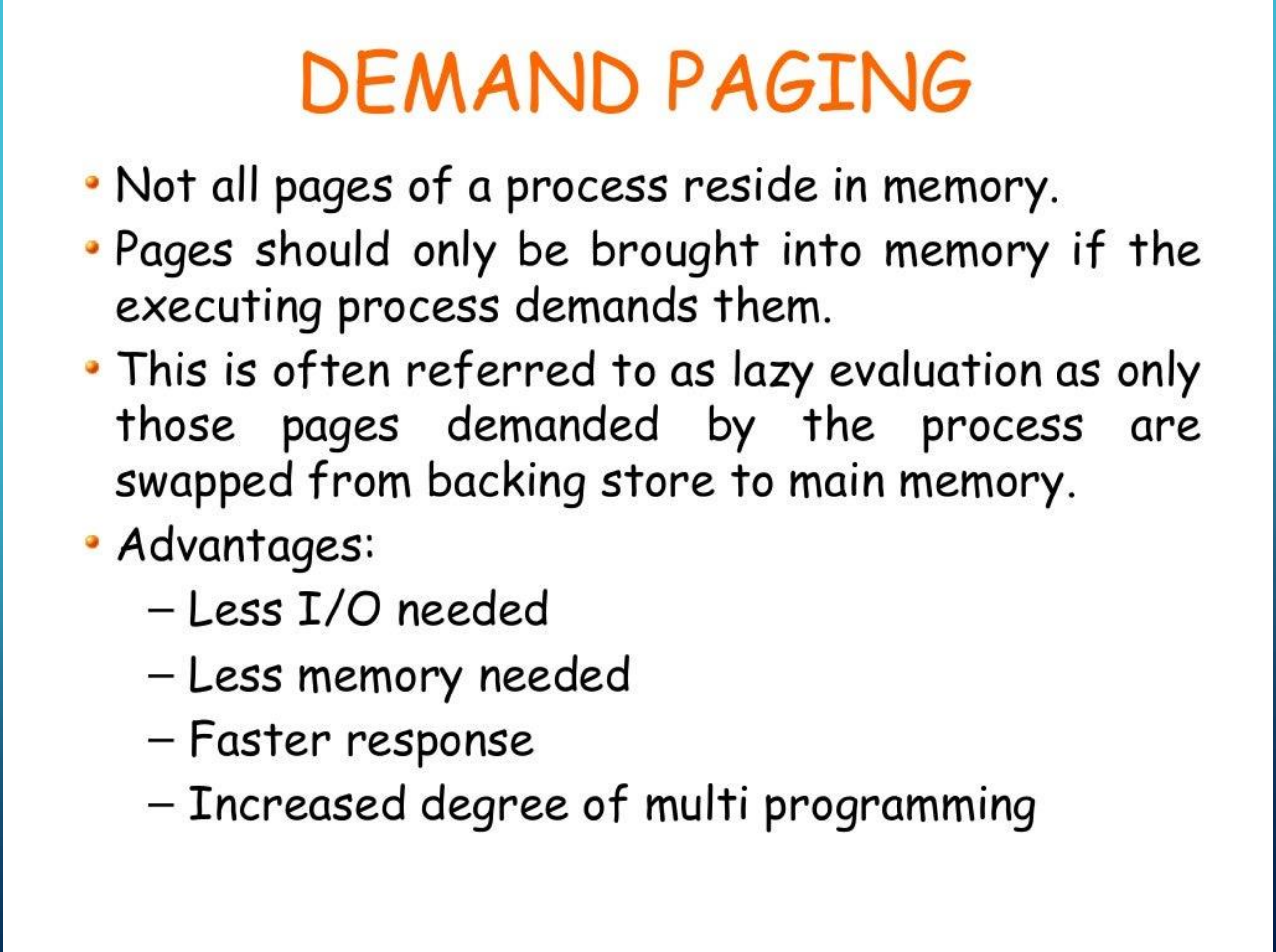
Swap Device $\rightarrow$ Memory

Kernel swaps in

- When a new process needs to be executed
- When a previous process which was swapped out wants to continue its execution

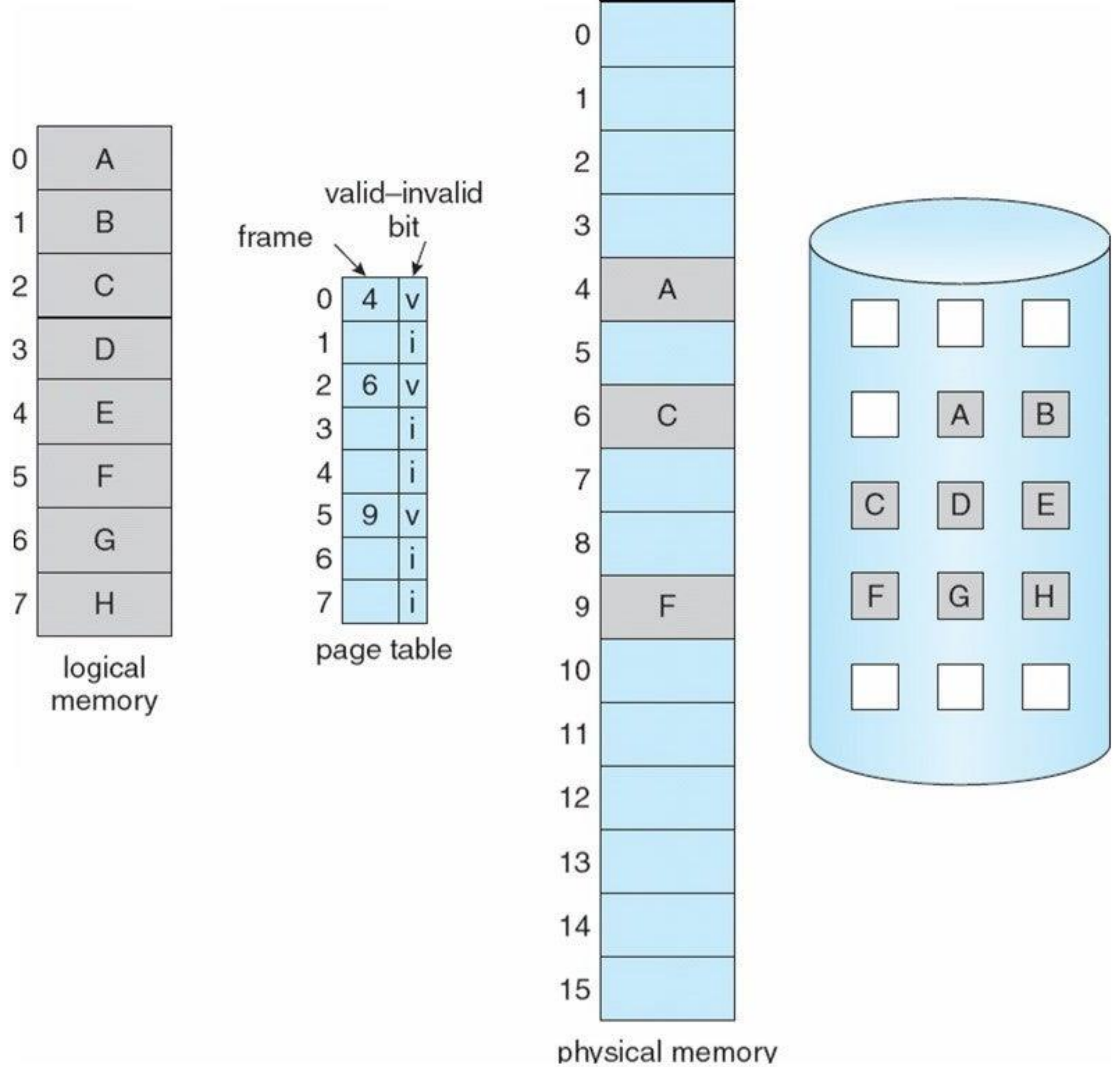
SWAPPING IN SWAPPING OUT

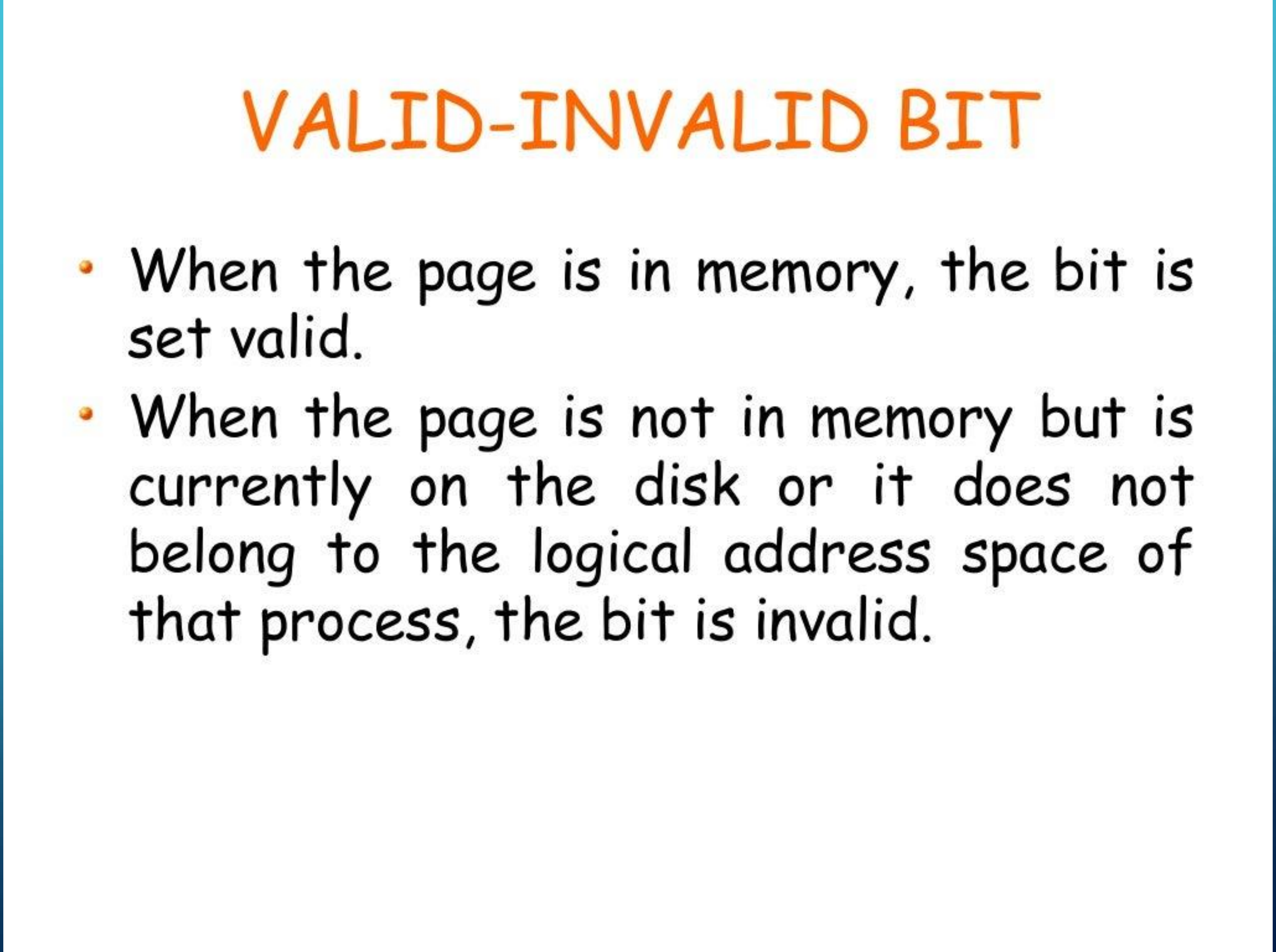




DEMAND PAGING

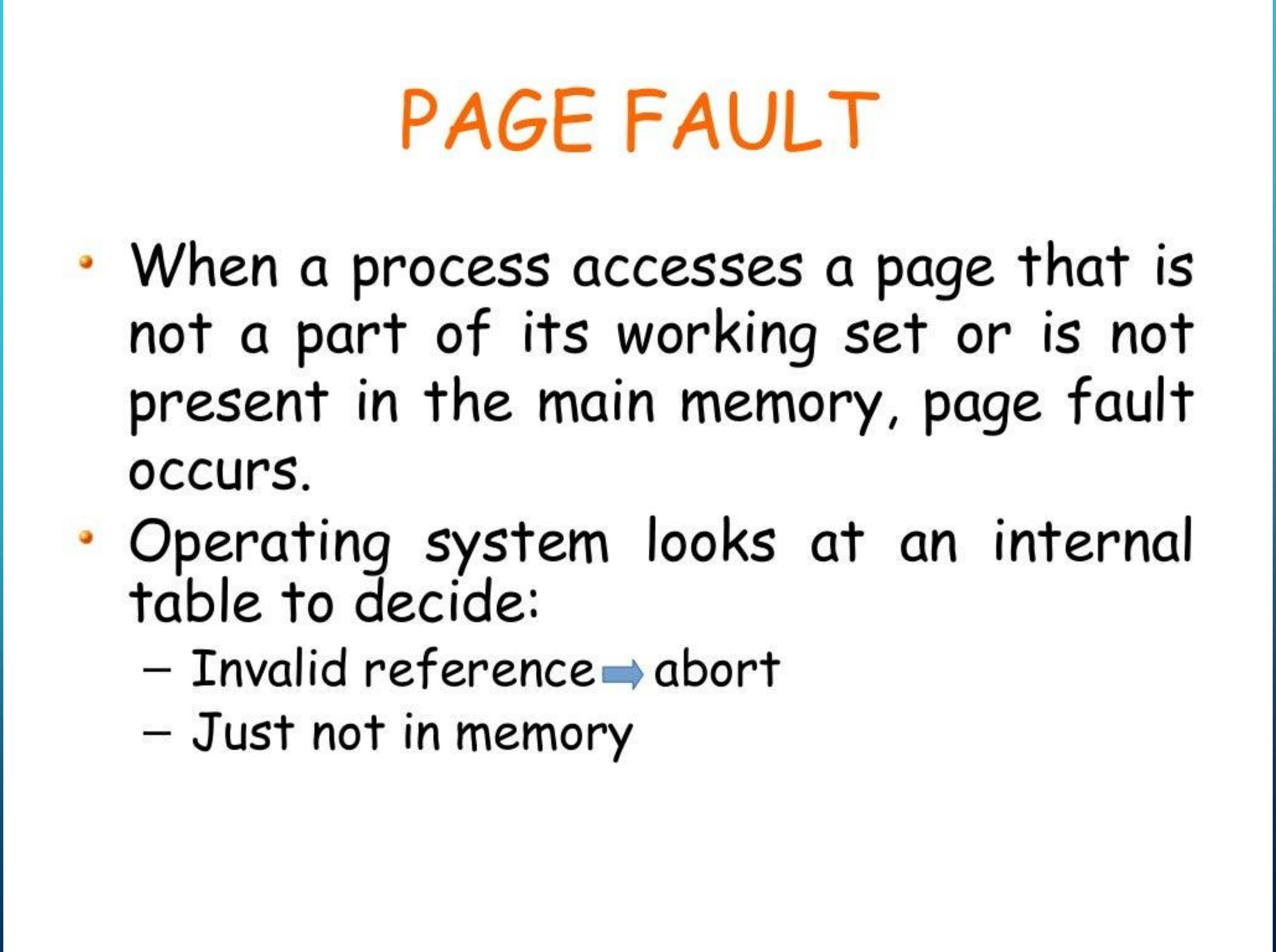
- Not all pages of a process reside in memory.
- Pages should only be brought into memory if the executing process demands them.
- This is often referred to as lazy evaluation as only those pages demanded by the process are swapped from backing store to main memory.
- Advantages:
  - Less I/O needed
  - Less memory needed
  - Faster response
  - Increased degree of multi programming


0
1
2
3
4
5
6
7
A
B
C
D
E
F
G
H
logical memory
valid–invalid bit
frame
0
1
2
3
4
5
6
7
4
v
i
6
v
i
i
9
v
i
i
page table
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A
C
F
physical memory
A
B
C
D
E
F
G
H



VALID-INVALID BIT

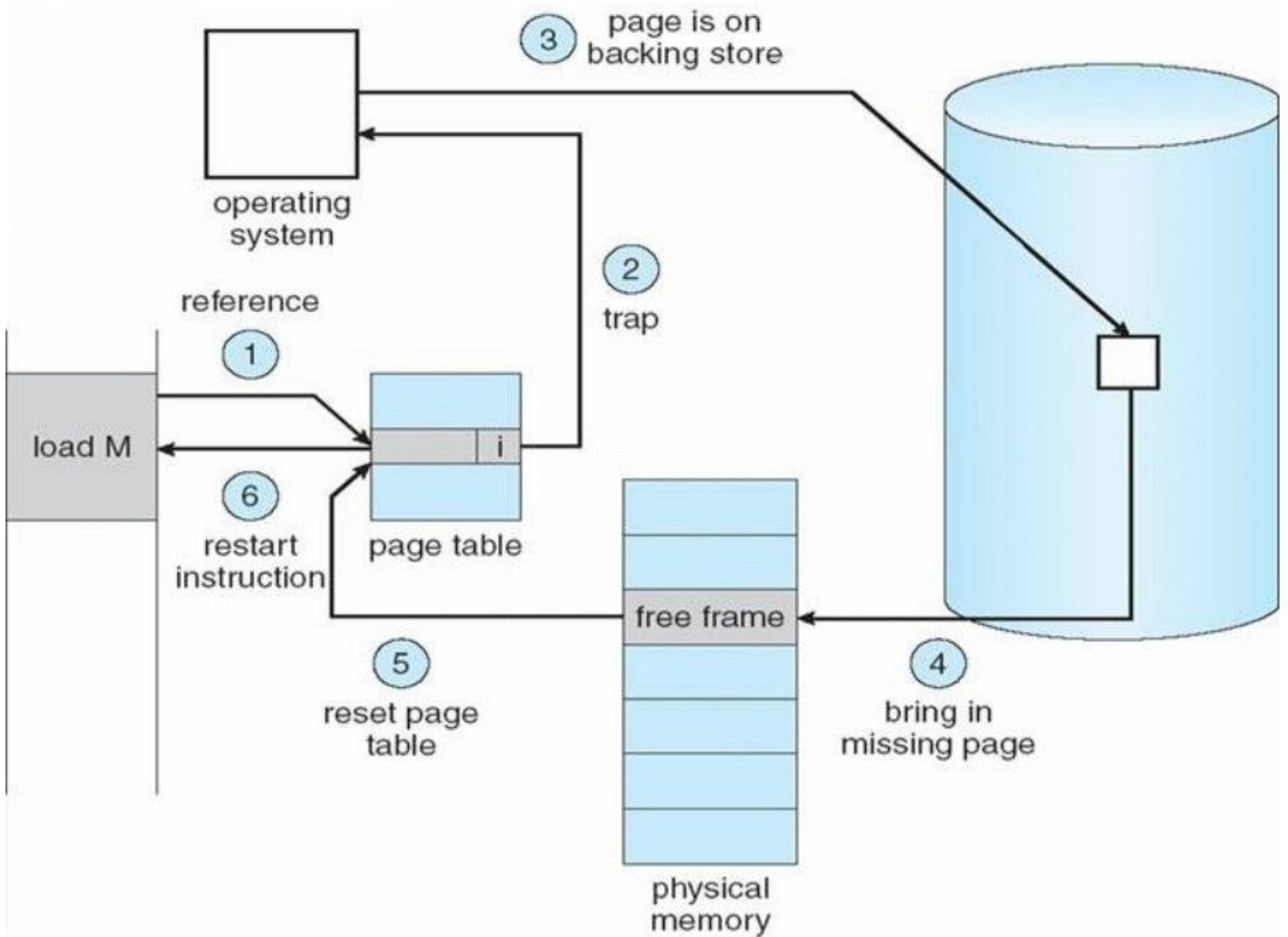
- When the page is in memory, the bit is set valid.
- When the page is not in memory but is currently on the disk or it does not belong to the logical address space of that process, the bit is invalid.



PAGE FAULT

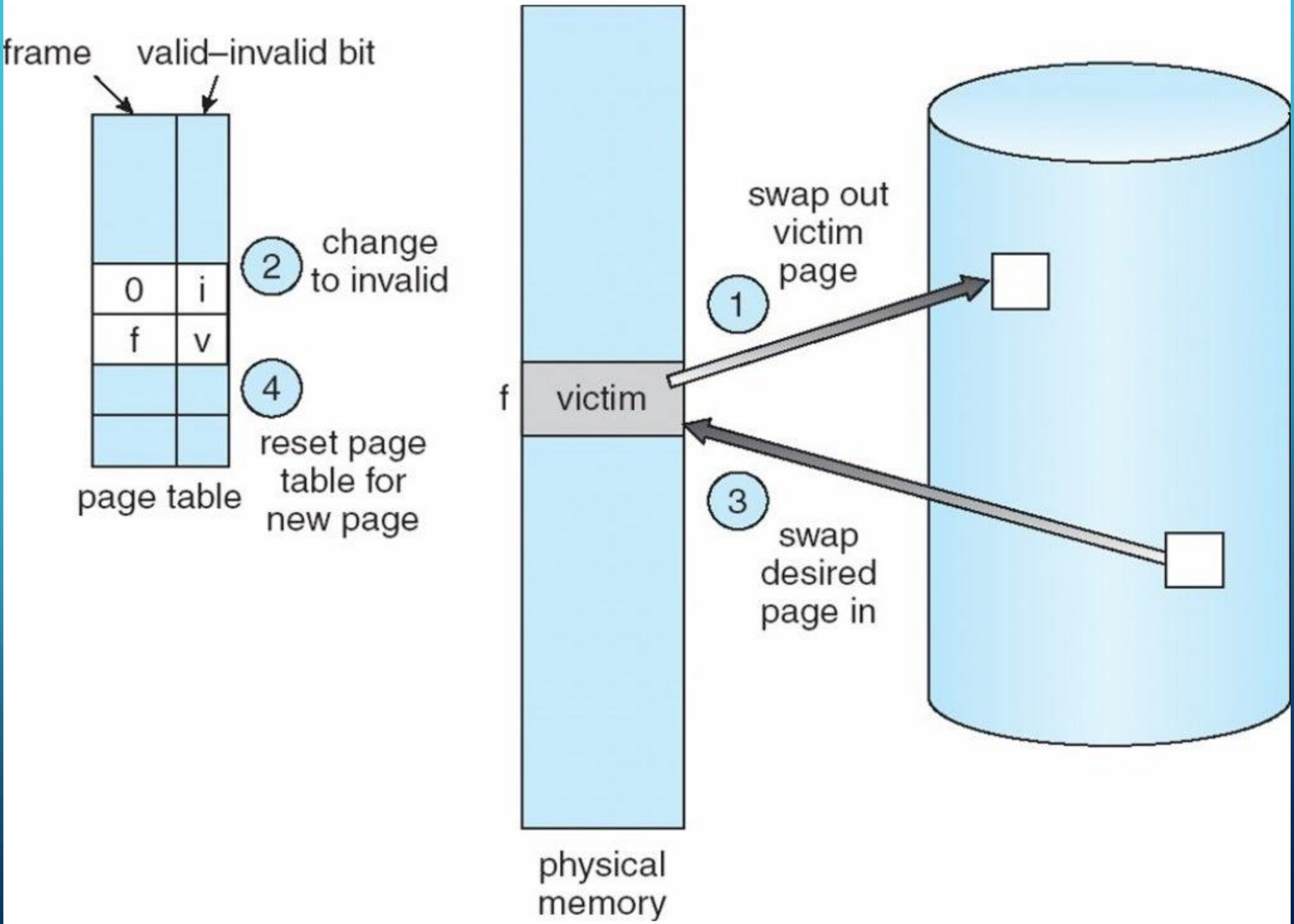
- When a process accesses a page that is not a part of its working set or is not present in the main memory, page fault occurs.
- Operating system looks at an internal table to decide:
  - Invalid reference → abort
  - Just not in memory

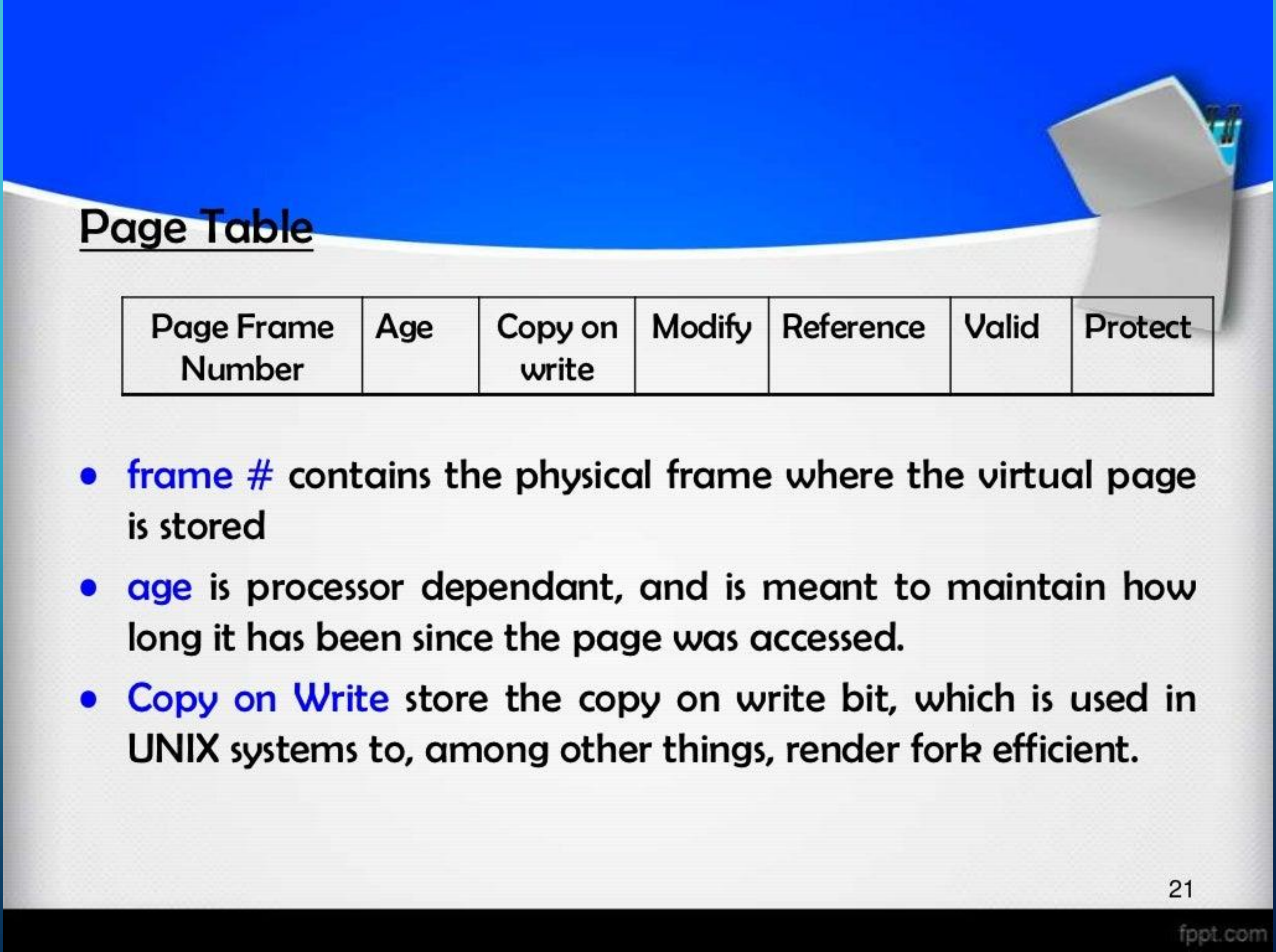
STEPS IN HANDLING A PAGE FAULT



WHAT HAPPENS WHEN NO FREE FRAME EXISTS?

- When no free frame exists in the main memory we find one, using certain algorithm, that is not currently being used and page it out.
- In UNIX, Not Recently Used(NRU) algorithm is used to select the victim page.

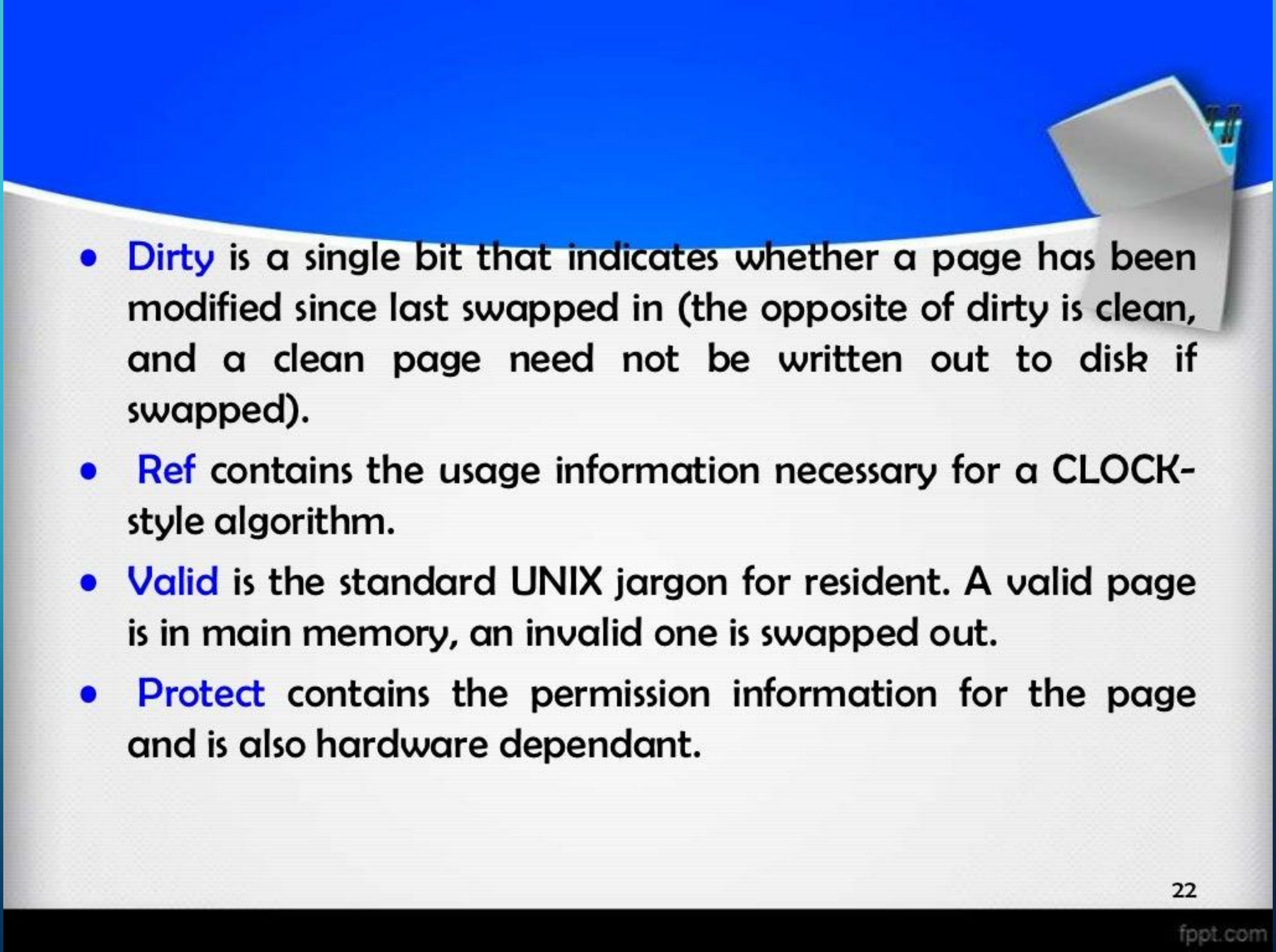

frame
valid–invalid bit
0
i
f
v
2 change to invalid
4 reset page table for new page
page table
f
victim
physical memory
1 swap out victim page
3 swap desired page in



Page Table

Page Frame Number	Age	Copy on write	Modify	Reference	Valid	Protect

- frame # contains the physical frame where the virtual page is stored
- age is processor dependant, and is meant to maintain how long it has been since the page was accessed.
- Copy on Write store the copy on write bit, which is used in UNIX systems to, among other things, render fork efficient.



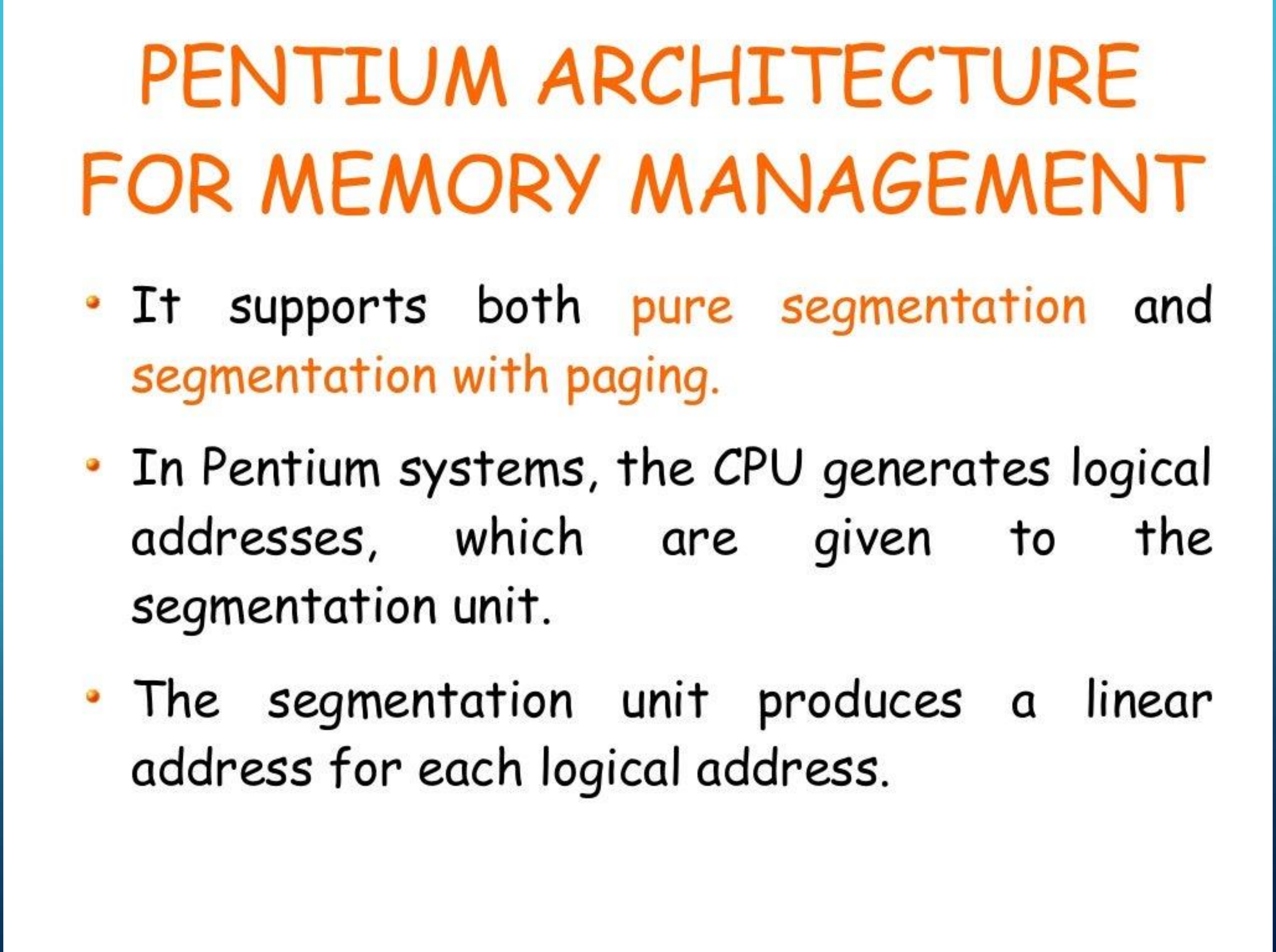
- **Dirty** is a single bit that indicates whether a page has been modified since last swapped in (the opposite of dirty is clean, and a clean page need not be written out to disk if swapped).
- **Ref** contains the usage information necessary for a CLOCK-style algorithm.
- **Valid** is the standard UNIX jargon for resident. A valid page is in main memory, an invalid one is swapped out.
- **Protect** contains the permission information for the page and is also hardware dependant.

Page frame data table

Page State	Reference Count	Logical device	Block number	Pf data pointer

- The page frame data table holds information about each physical frame of memory (indexed by frame number).
- This table is of primary importance for the replacement algorithm.

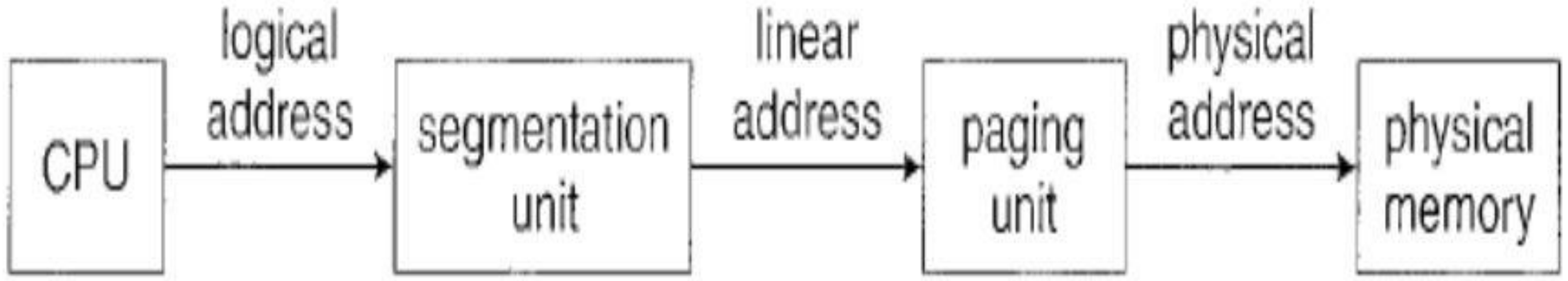
WEEK: 06
UNIX MEMORY MANAGEMENT 03



PENTIUM ARCHITECTURE FOR MEMORY MANAGEMENT

- It supports both pure segmentation and segmentation with paging.
- In Pentium systems, the CPU generates logical addresses, which are given to the segmentation unit.
- The segmentation unit produces a linear address for each logical address.

- The linear address is then given to the paging unit, which in turn generates the physical address in main memory.



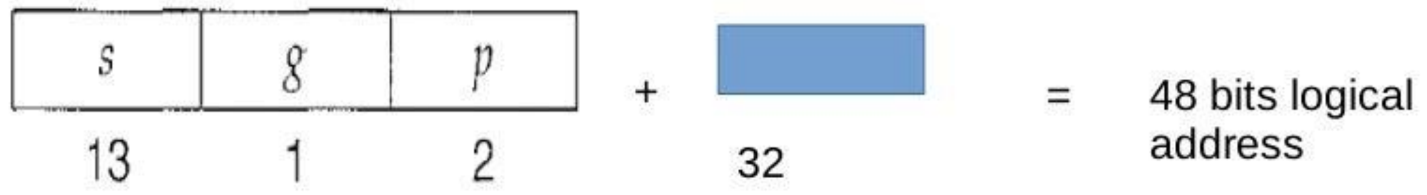
PENTIUM PURE SEGMENTATION

- The Pentium architecture allows a segment to be as large as 4 GB, and the maximum number of segments per process is 16 K.
- The logical-address space of a process is divided into two partitions :-

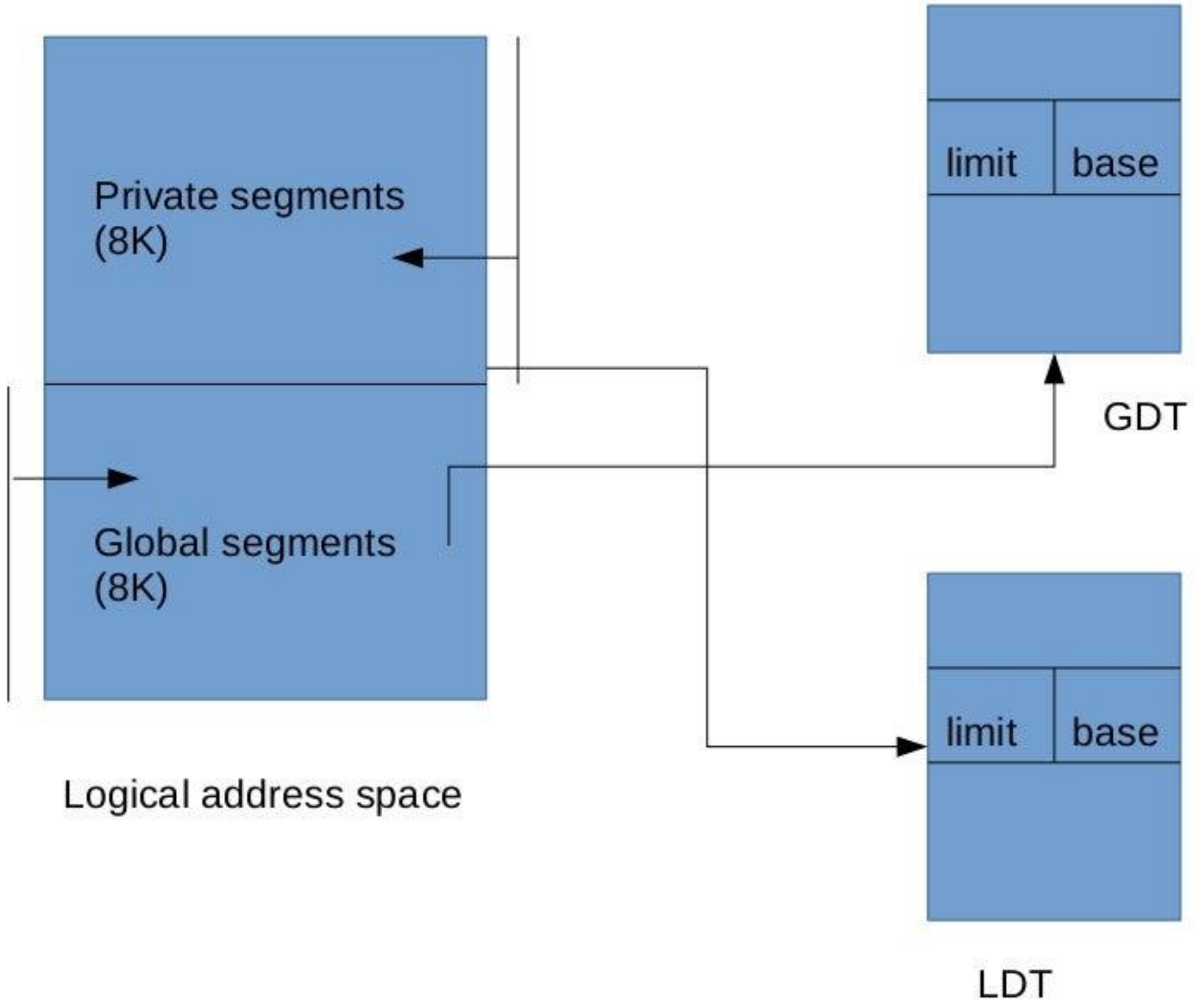
  -The first partition consists of up to 8 K segments that are private to that process.

  -The second partition consists of up to 8 K segments that are shared all the processes.

- Information about the first partition is kept in LOCAL DESCRIPTOR TABLE (LDT).
- The information about the second Partition is kept in the GLOBAL DESCRIPTOR TABLE(GDT).
- Each entry in the LDT and GDT consists of an 8-byte segment descriptor with detailed information about a particular segment, including the base location and limit of that segment.
- The logical address is a pair, where the selector is a 16 -bit number and the offset is of 32- bits.

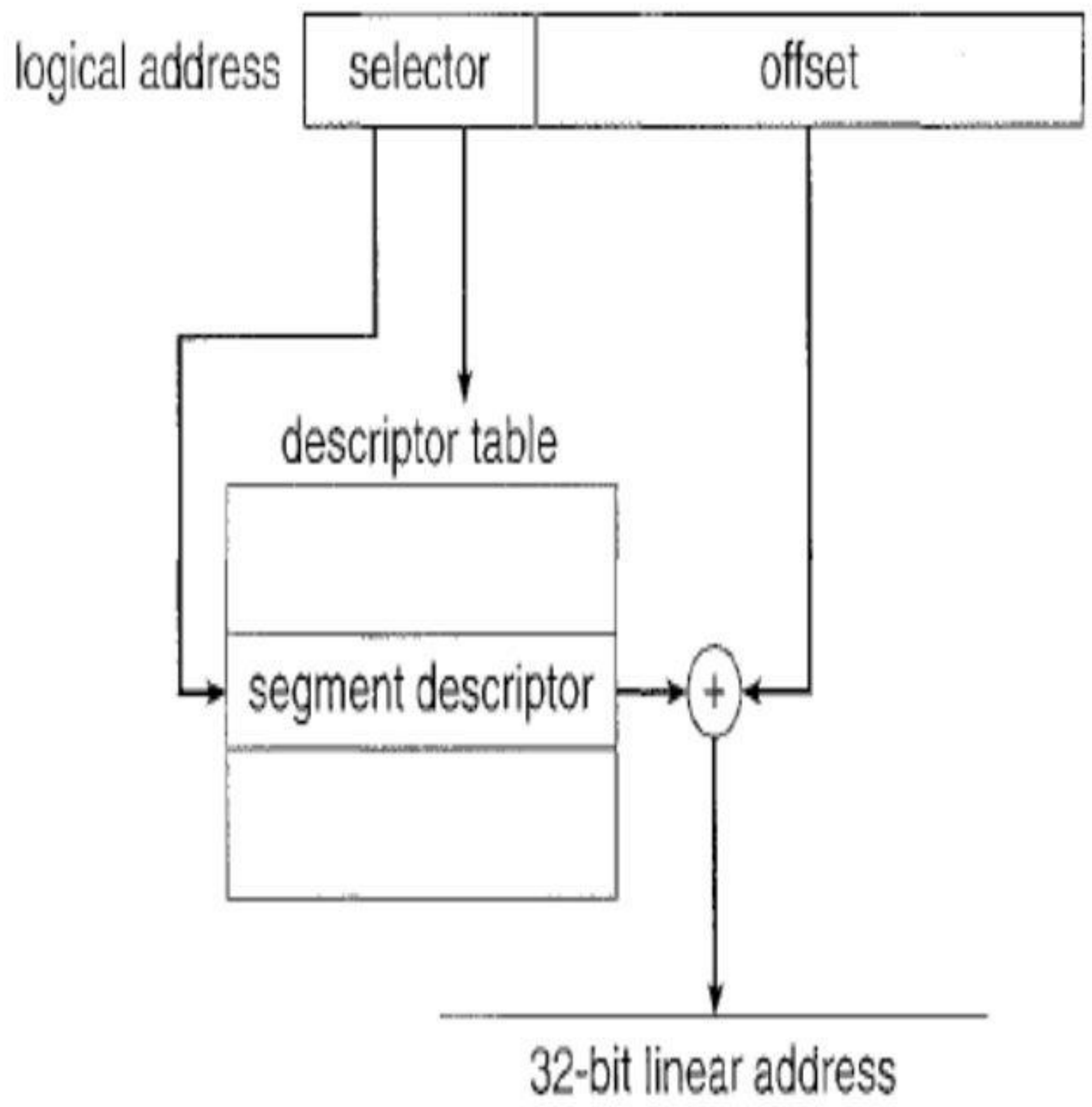


s designates the segment number, *g indicate whether the segment is* in the GDT or LDT, and *p deals with protection.*



- *The offset is a 32-bit number* specifying the location of the byte (or word) within the segment in question.
- The machine has six segment registers, allowing six segments to be addressed at any one time by a process. It also has six 8-byte micro-program registers to hold the corresponding descriptors from either the LDT or GDT.
- This cache lets the Pentium avoid having to read the descriptor from memory for every memory reference.

- The linear address on the Pentium is 32 bits long and is formed as follows.
- The segment register points to the appropriate entry in the LDT or GDT. The base and limit information about the segment in question is used to generate a linear address .
- First, the limit is used to check for address validity.
- If the address is not valid, a memory fault is generated, resulting in a trap to the operating system.
- If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address.


logical address
selector
offset
descriptor table
segment descriptor
+
32-bit linear address

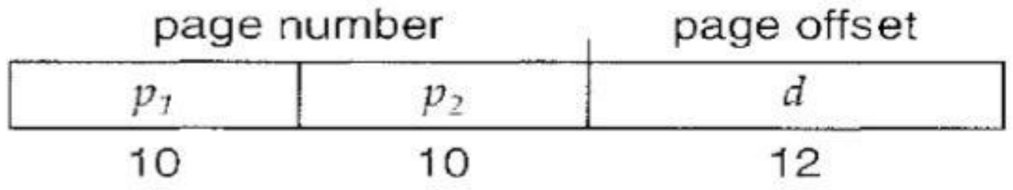
PENTIUM SEGMENTATION WITH PAGING

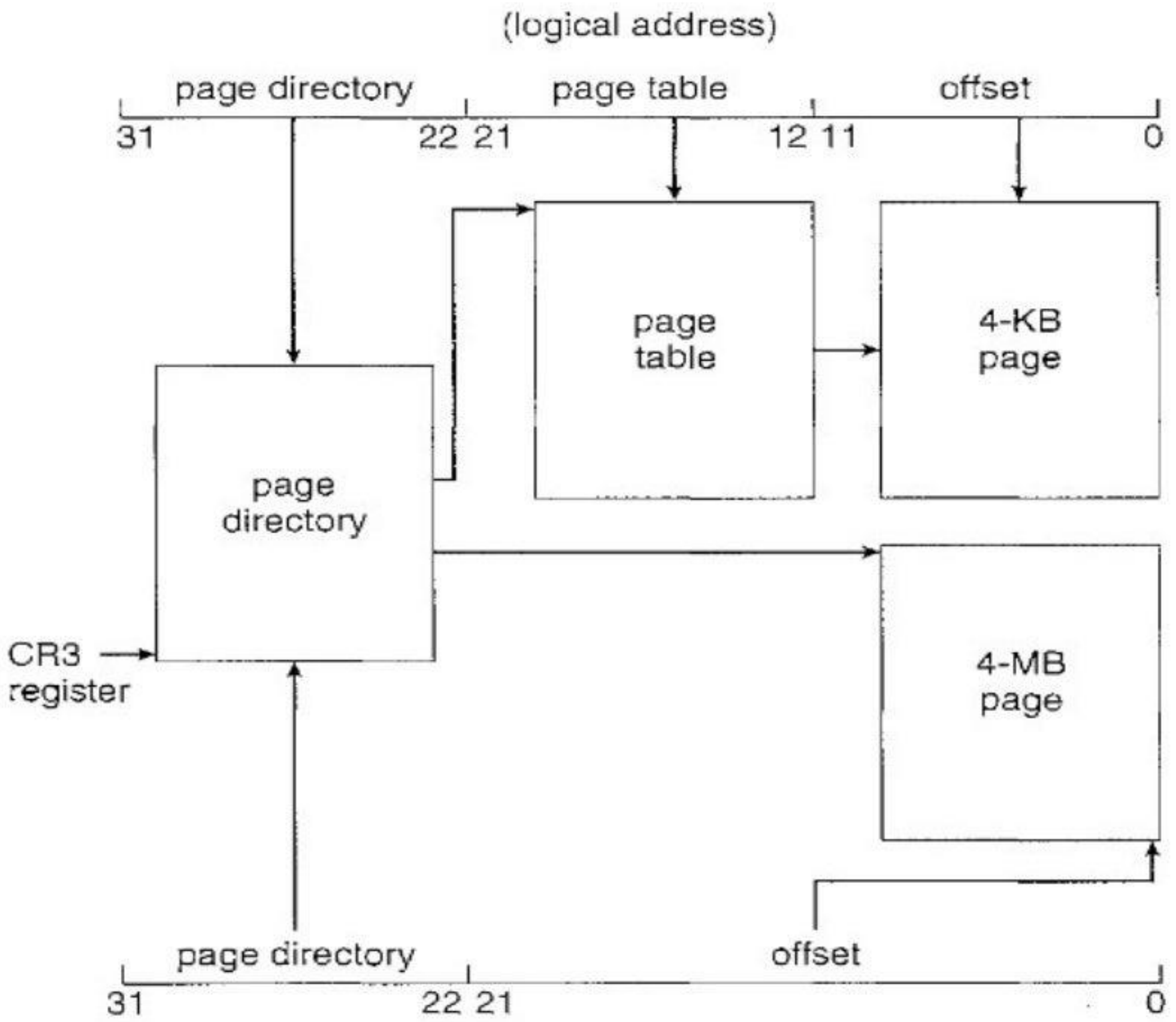
- The Pentium architecture allows a page size of either 4 KB or 4 MB.
- It uses a 32 bit linear address which is divided as follows:

  -The 10 high-order bits referes to an entry in the **page directory**.

  -The page directory entry points to an inner page table that is indexed by the contents of the innermost 10 bits in the linear address.

  -Finally, the low-order bits 0-11 refer to the offset in the 4-KB page pointed to in the page table.




(logical address)
page directory
page table
offset
31
22 21
12 11
0
page table
4-KB page
page directory
CR3 register
4-MB page
page directory
offset
31
22 21
0

- One entry in the page directory is the Page Size flag, which-if set indicates that the size of the page frame is 4 MB and not the standard 4 KB.
- If this flag is set, the page directory points directly to the 4-MB page frame, and the 22 low-order bits in the linear address refer to the offset in the 4-MB page frame.

- To improve the efficiency of physical memory use, Intel Pentium page tables can be swapped to disk.
- In this case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is in memory or on disk.
- If the table is on disk, the operating system can use the other 31 bits to specify the disk location of the table; the table then can be brought into memory on demand.

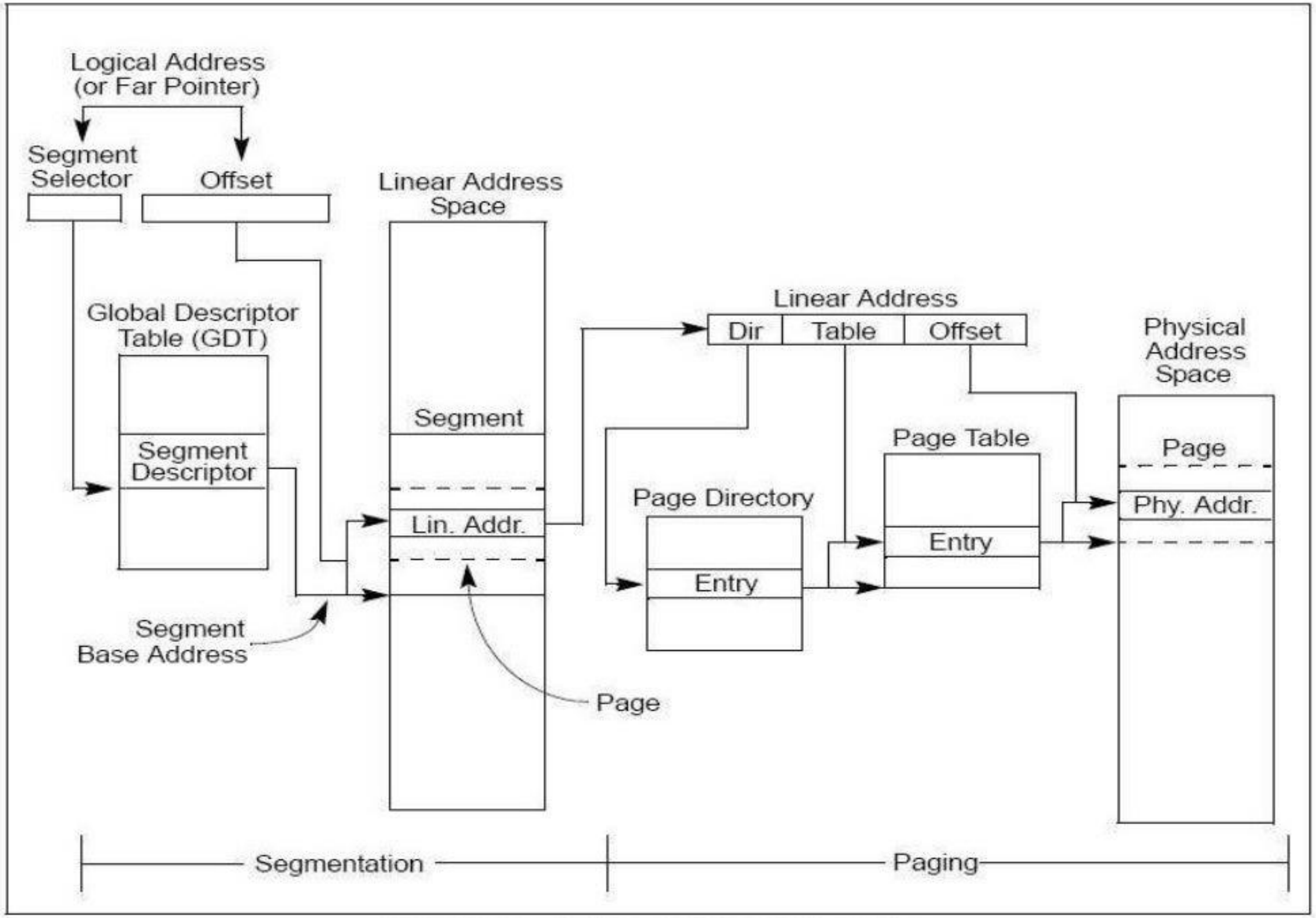
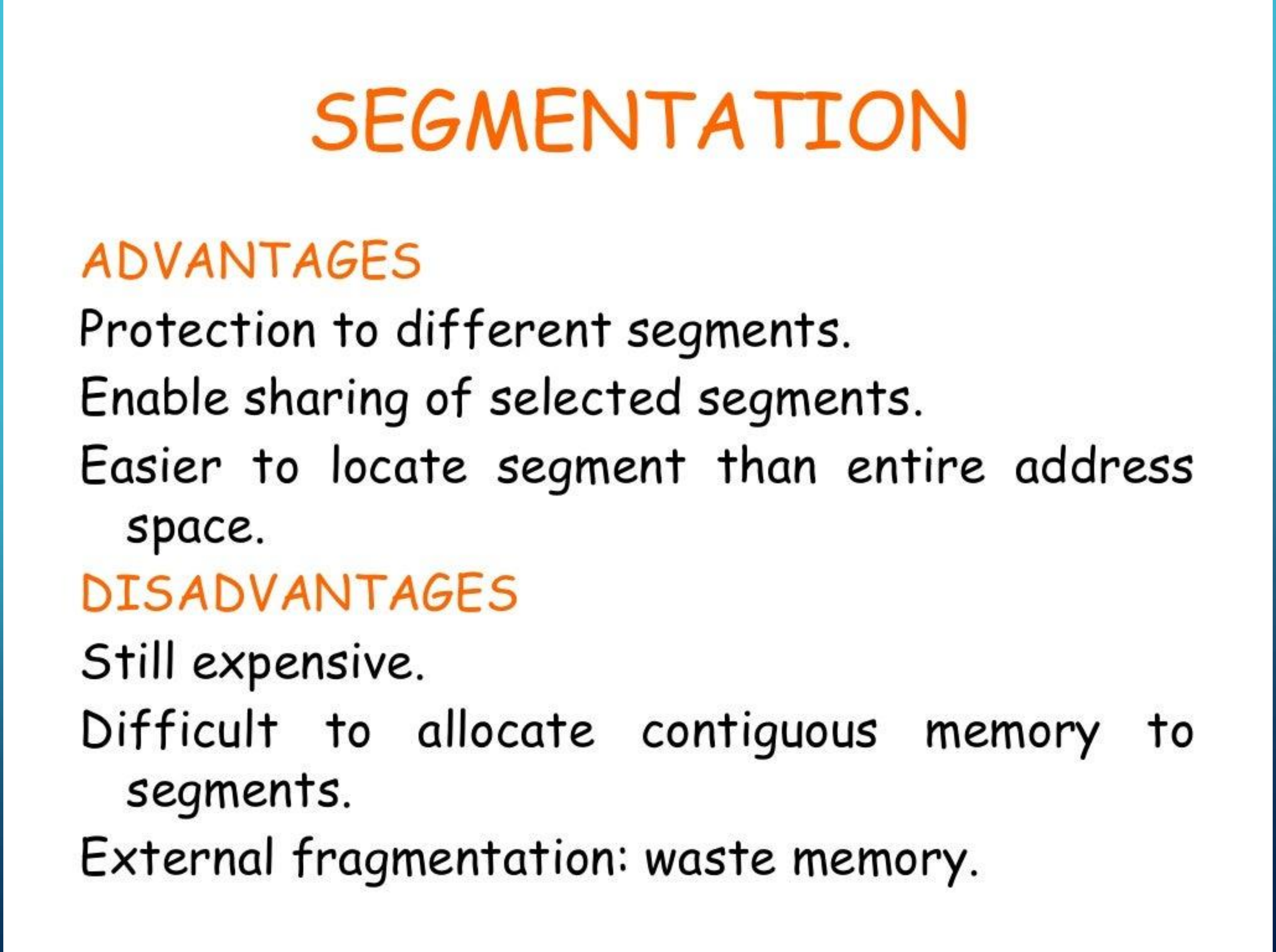


Figure 3-1. Segmentation and Paging



SEGMENTATION

ADVANTAGES

Protection to different segments.

Enable sharing of selected segments.

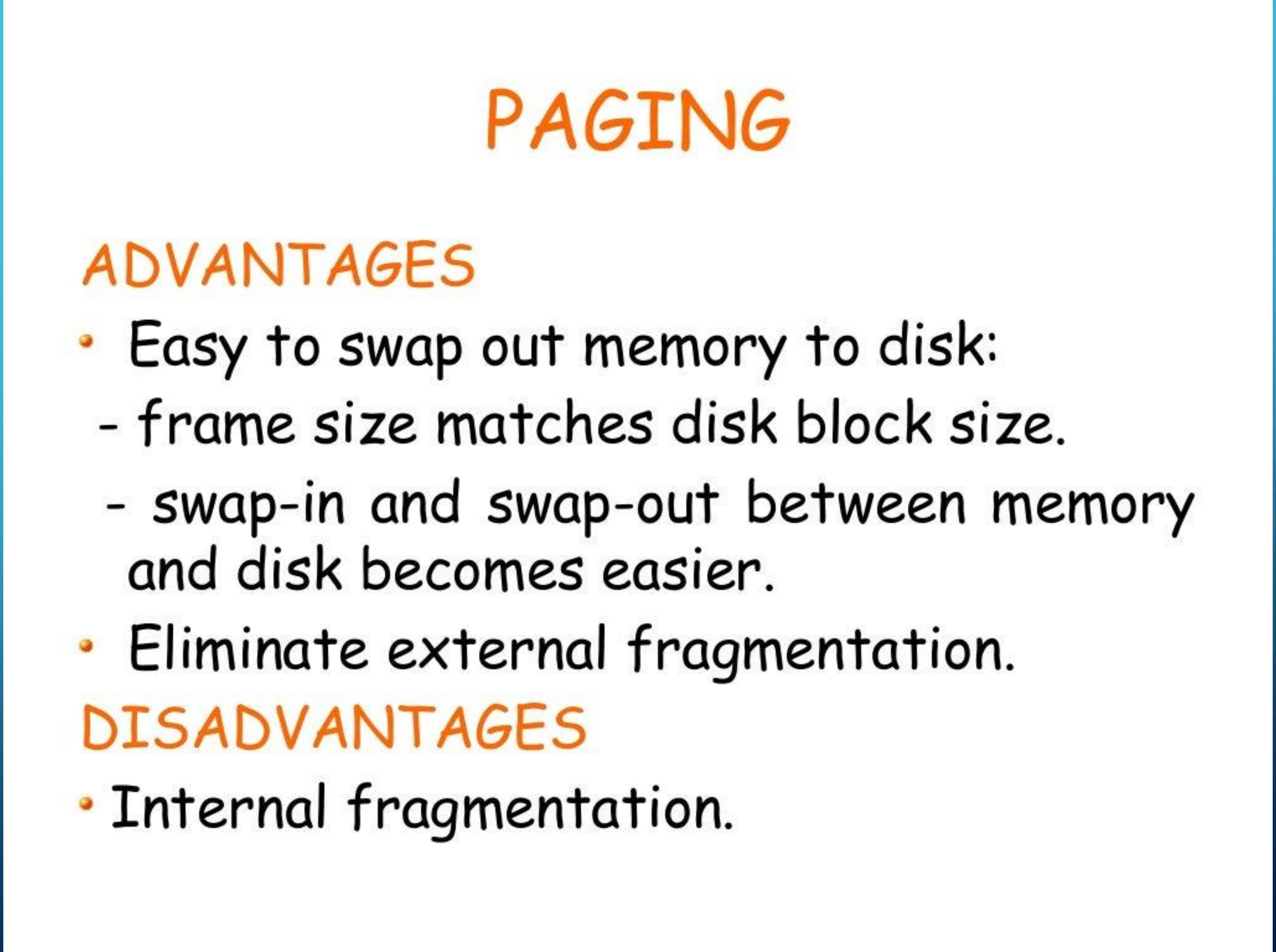
Easier to locate segment than entire address space.

DISADVANTAGES

Still expensive.

Difficult to allocate contiguous memory to segments.

External fragmentation: waste memory.



PAGING

ADVANTAGES

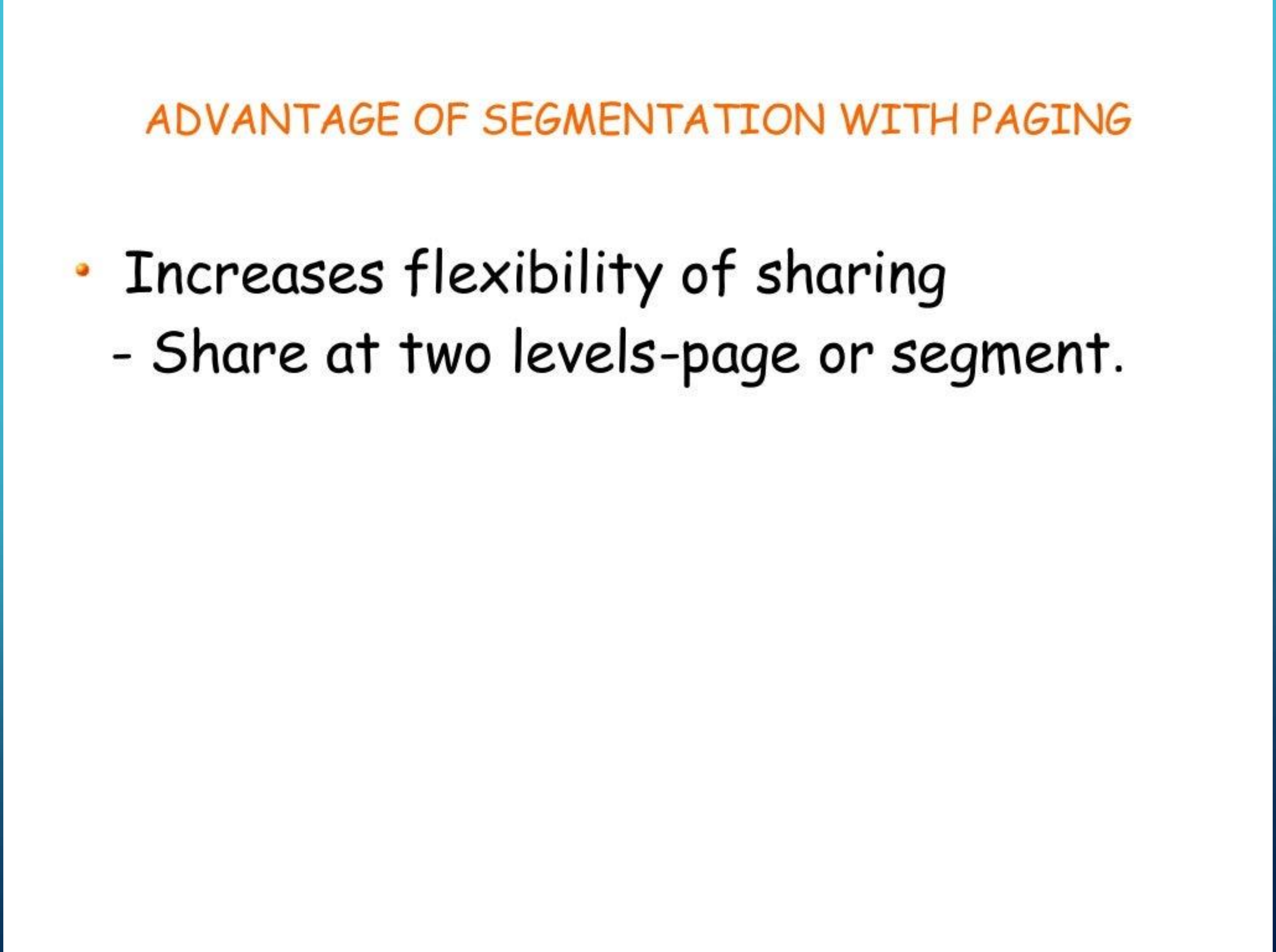
- Easy to swap out memory to disk:

  - frame size matches disk block size.
  - swap-in and swap-out between memory and disk becomes easier.

- Eliminate external fragmentation.

DISADVANTAGES

- Internal fragmentation.



ADVANTAGE OF SEGMENTATION WITH PAGING

- Increases flexibility of sharing

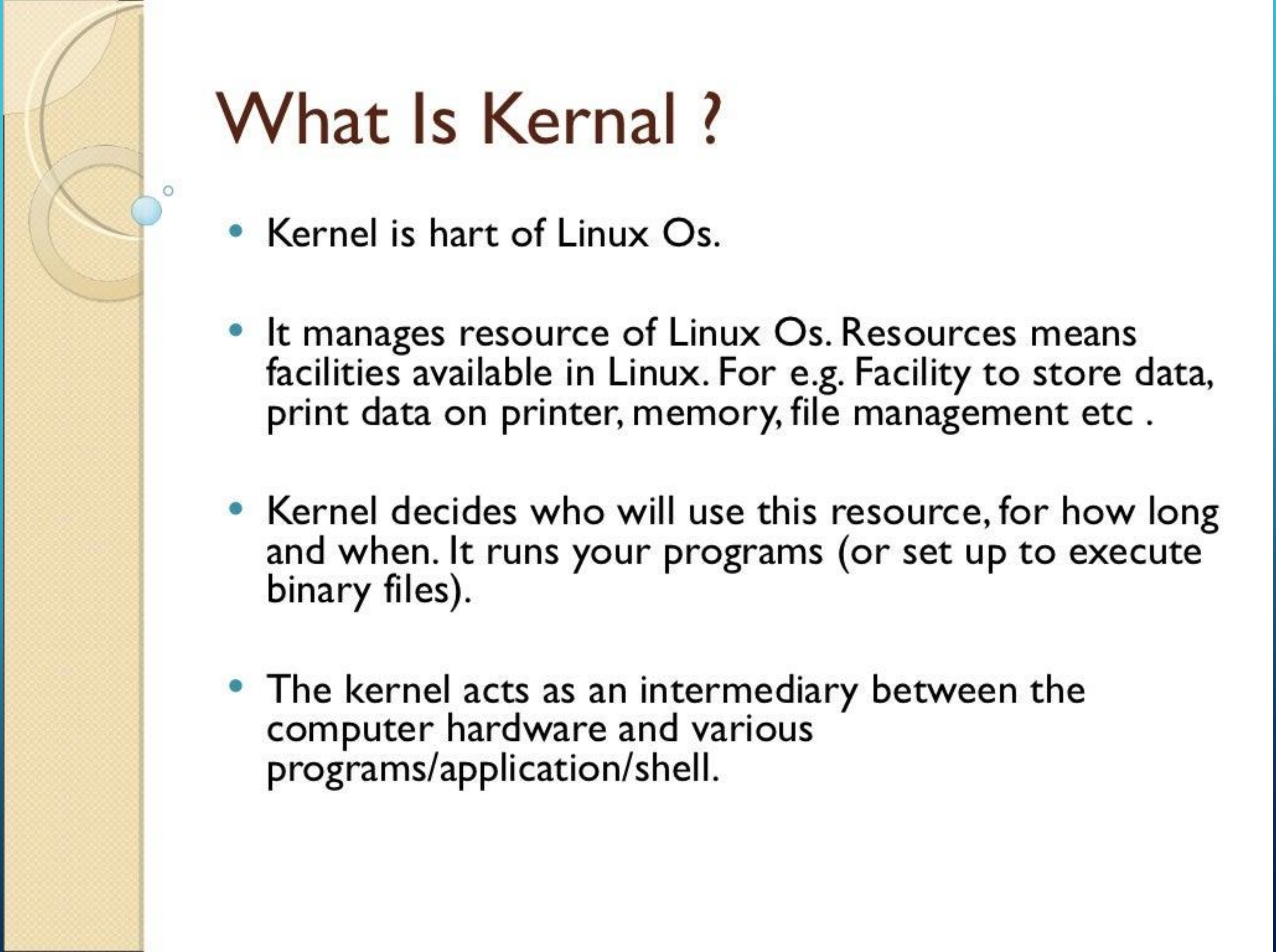
  - Share at two levels-page or segment.

BIBLIOGRAPHY

- Operating System Concepts by Abraham Silberschatz, Greg Gagne, Peter Baer Galvin
- www.linux.com
- www.tutorialspoint.com

WEEK: 07
SHELL PROGRAMMING

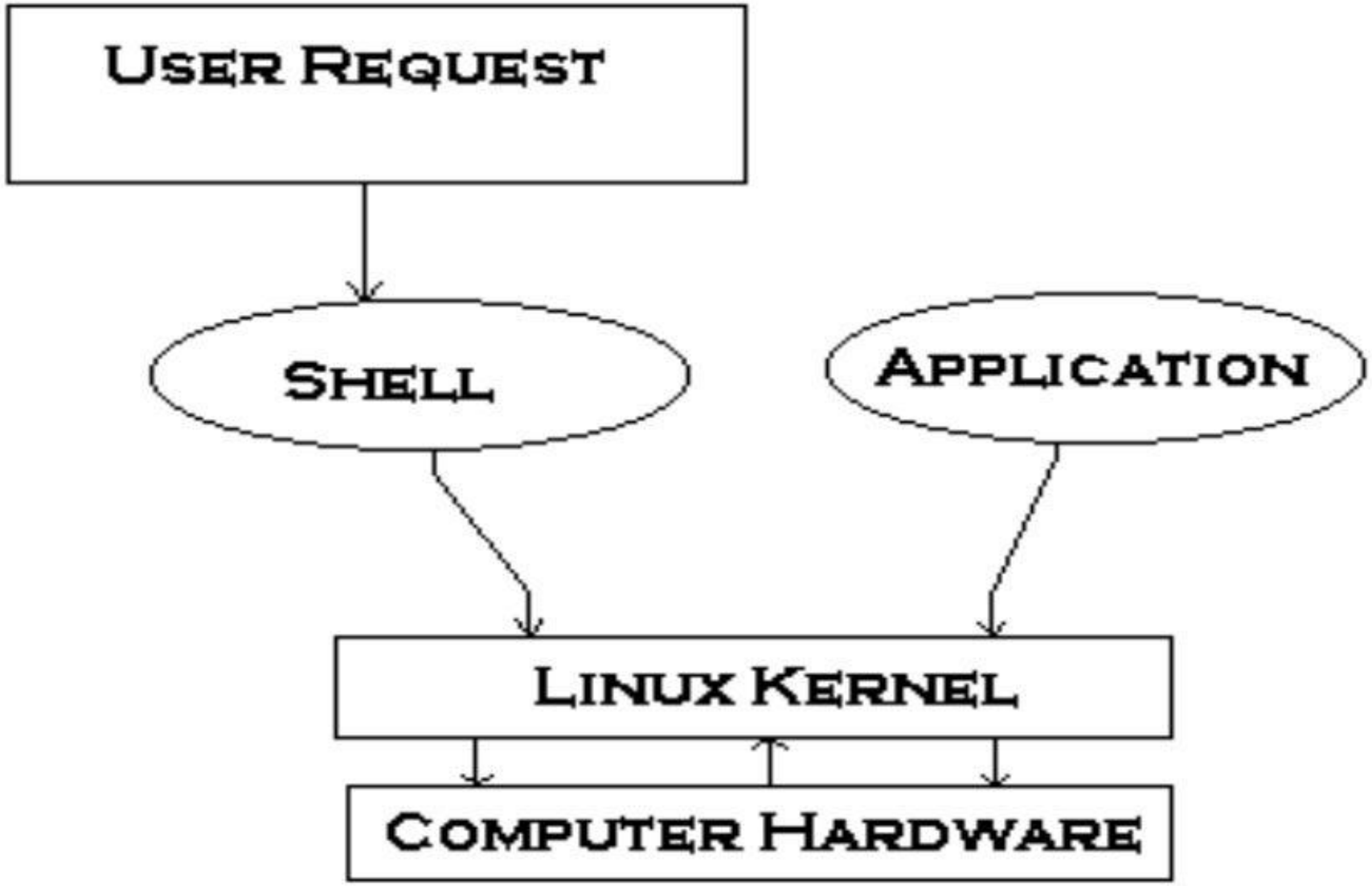
Shell Programming

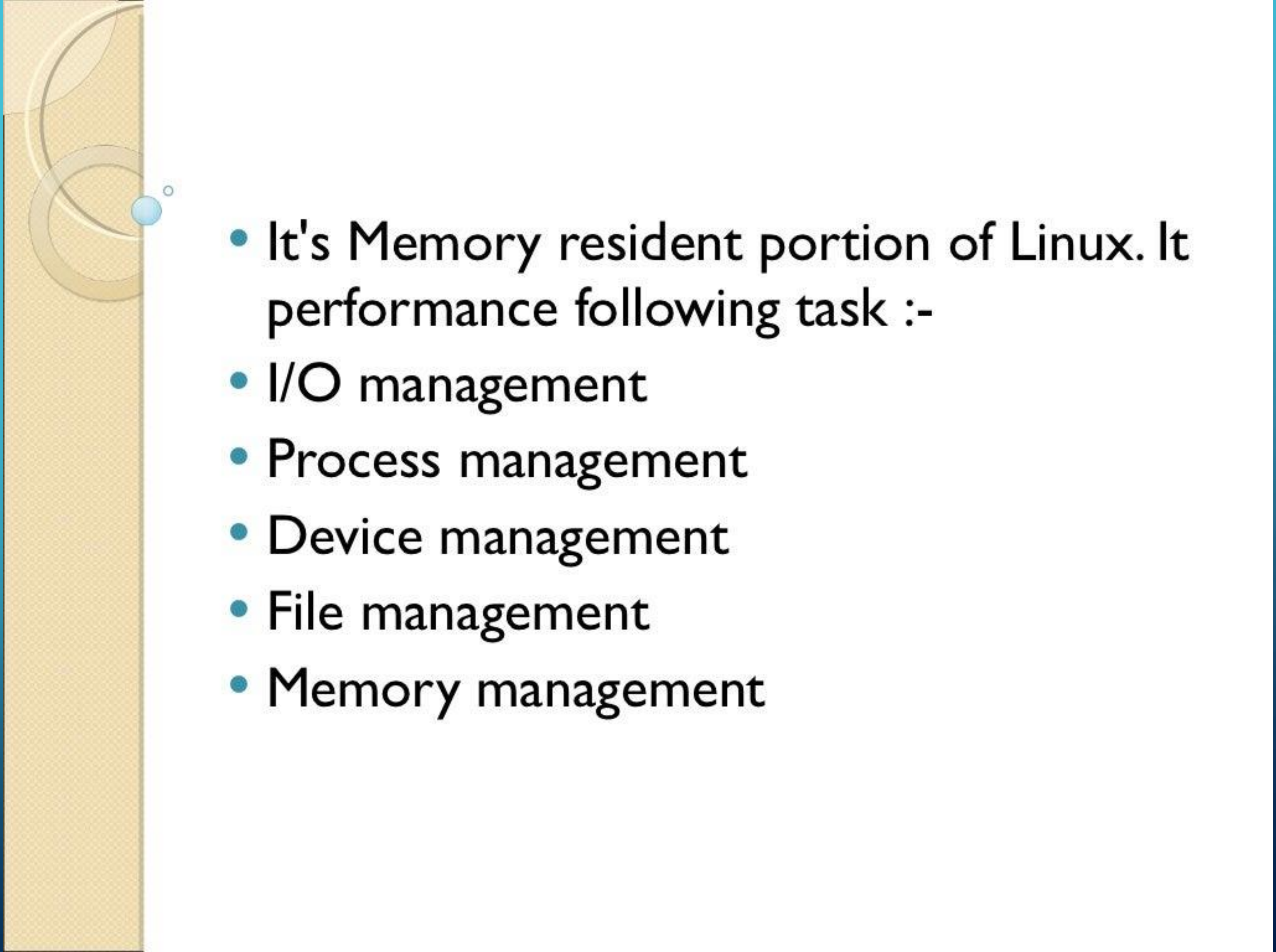


What Is Kernal ?

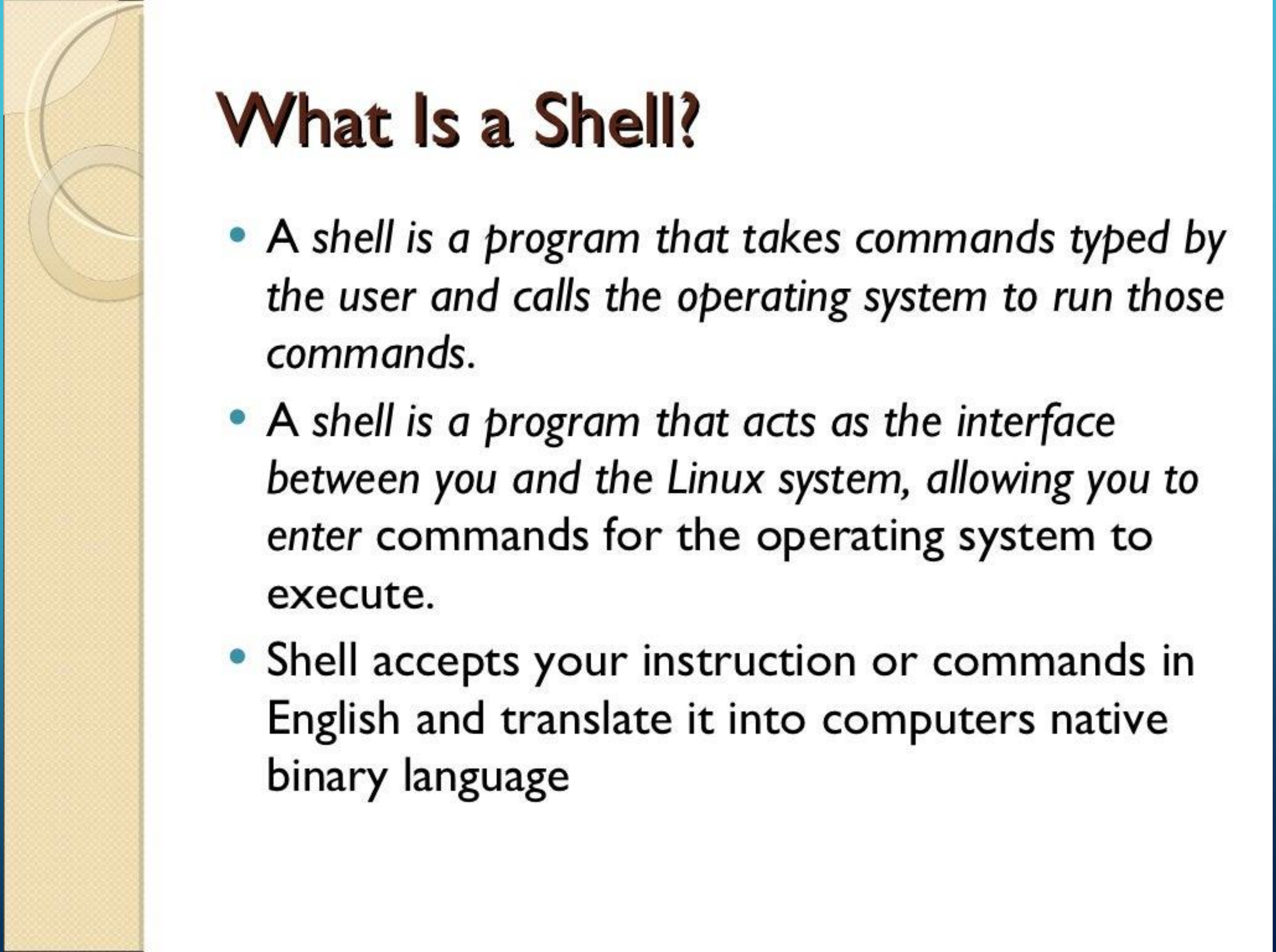
- Kernel is hart of Linux Os.
- It manages resource of Linux Os. Resources means facilities available in Linux. For e.g. Facility to store data, print data on printer, memory, file management etc .
- Kernel decides who will use this resource, for how long and when. It runs your programs (or set up to execute binary files).
- The kernel acts as an intermediary between the computer hardware and various programs/application/shell.

Kernal con.


User Request
Shell
Application
Linux Kernel
Computer Hardware

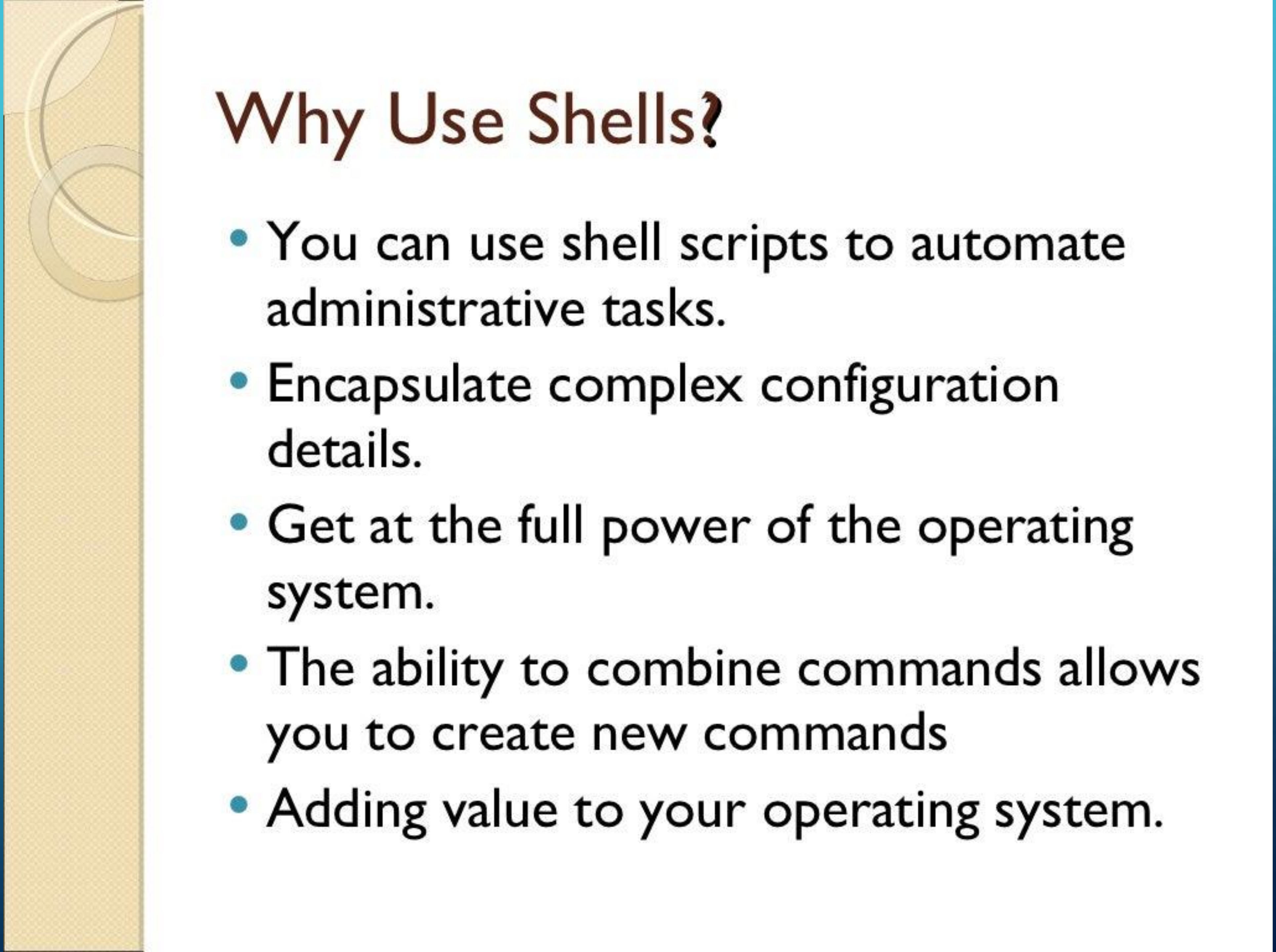


- It's Memory resident portion of Linux. It performance following task :-
- I/O management
- Process management
- Device management
- File management
- Memory management



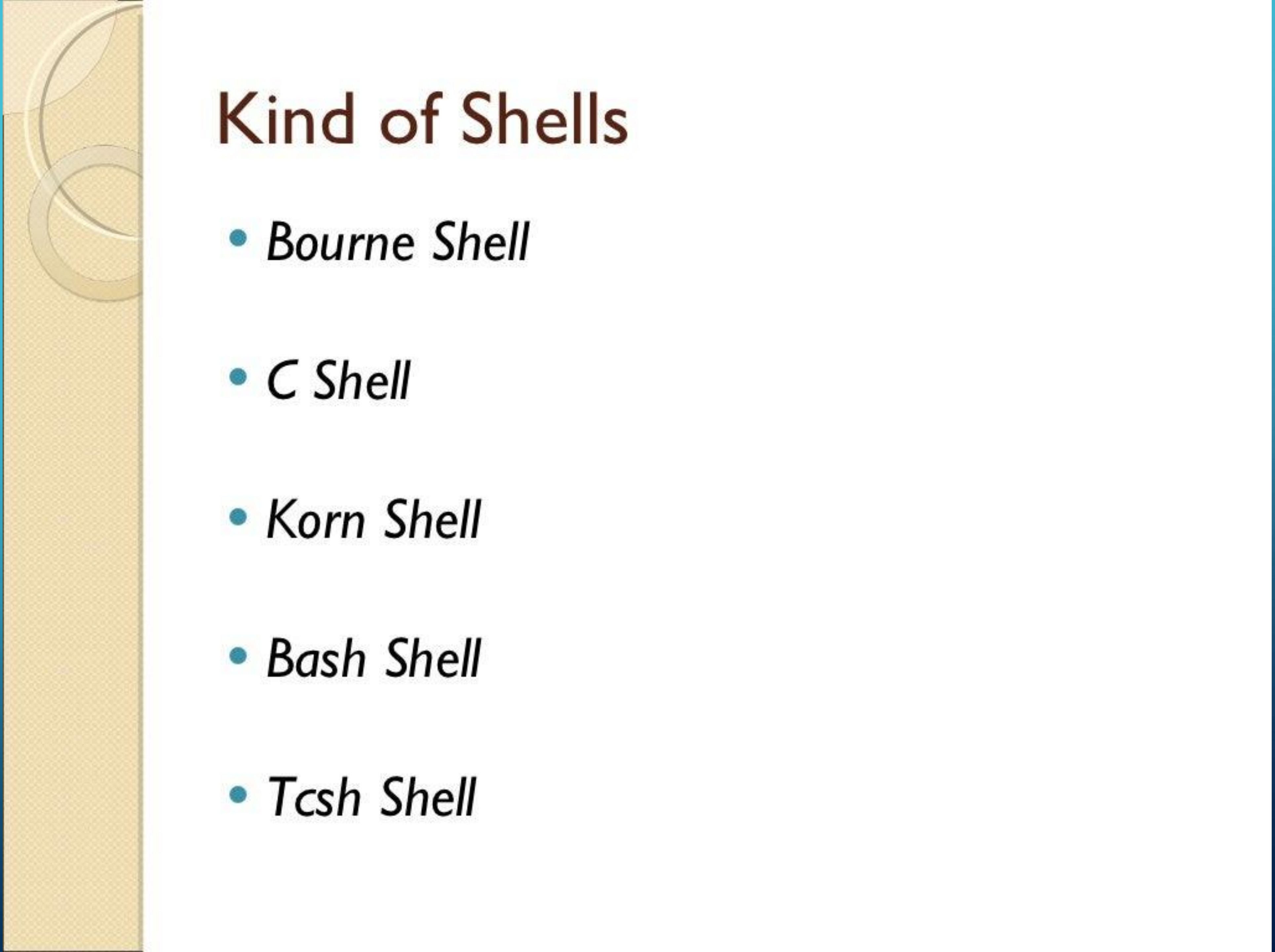
What Is a Shell?

- A *shell is a program that takes commands typed by the user and calls the operating system to run those commands.*
- A *shell is a program that acts as the interface between you and the Linux system, allowing you to enter* commands for the operating system to execute.
- Shell accepts your instruction or commands in English and translate it into computers native binary language



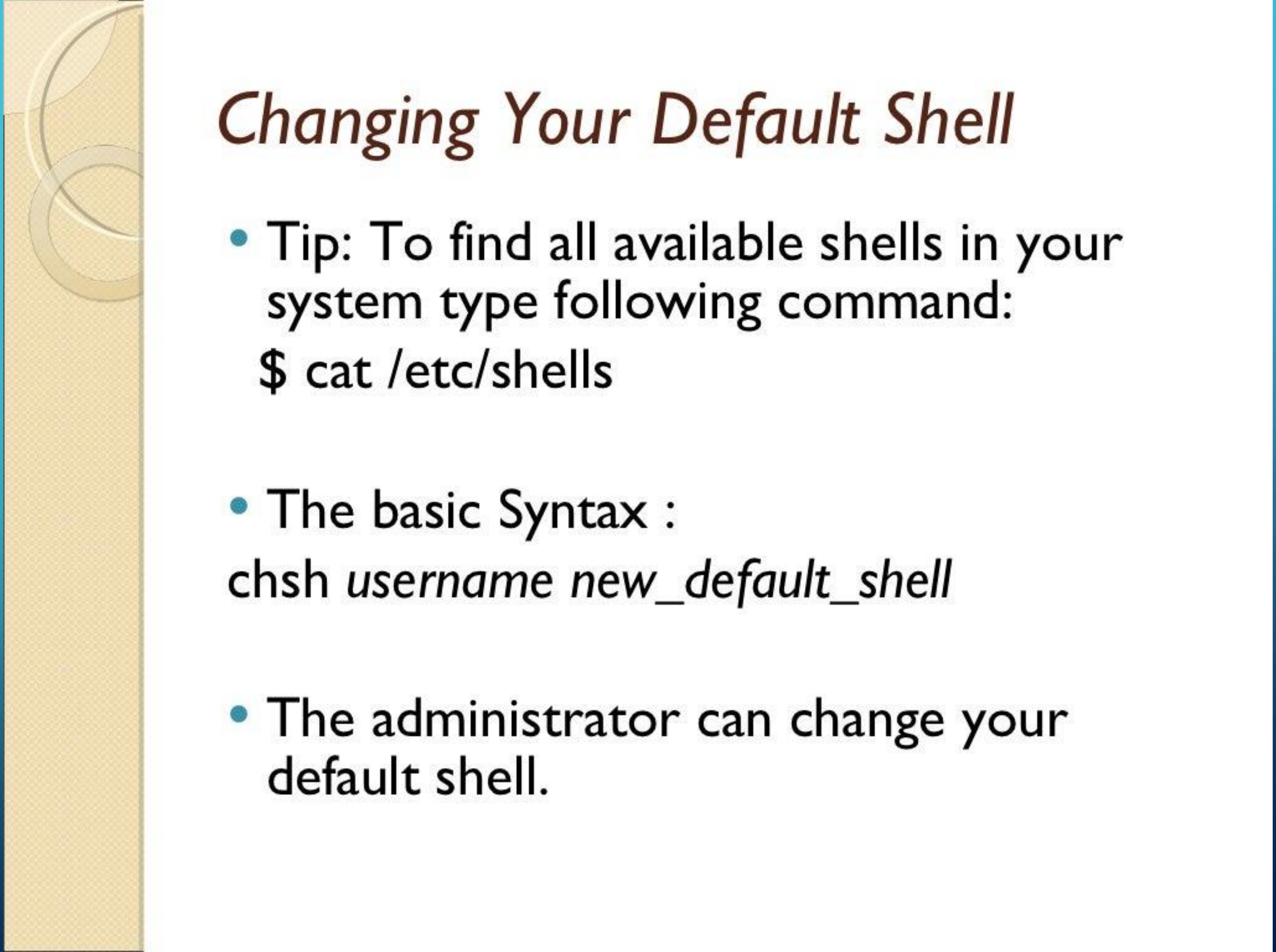
Why Use Shells?

- You can use shell scripts to automate administrative tasks.
- Encapsulate complex configuration details.
- Get at the full power of the operating system.
- The ability to combine commands allows you to create new commands
- Adding value to your operating system.



Kind of Shells

- *Bourne Shell*
- *C Shell*
- *Korn Shell*
- *Bash Shell*
- *Tcsh Shell*



Changing Your Default Shell

- Tip: To find all available shells in your system type following command:

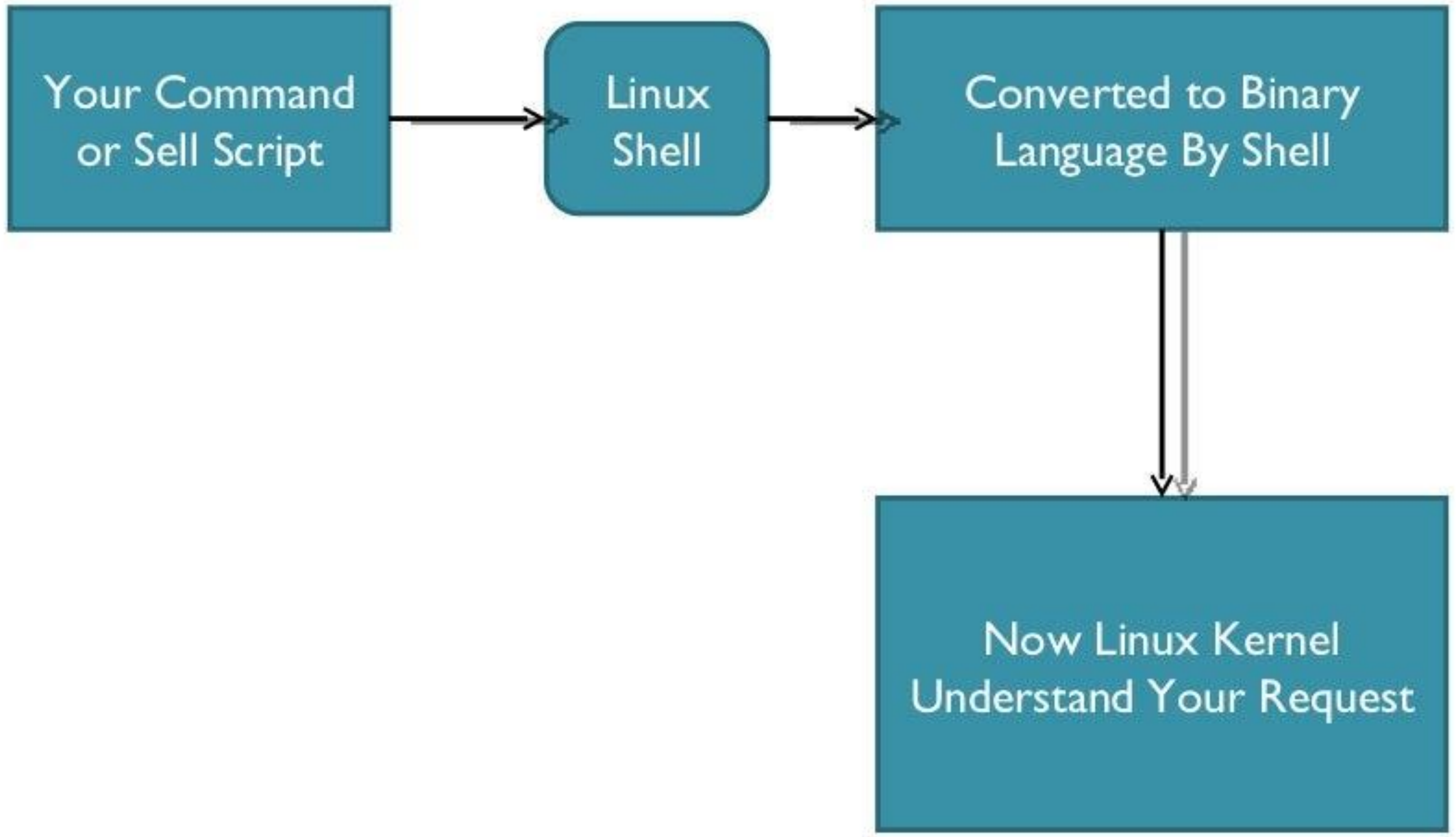
```
$ cat /etc/shells
```

- The basic Syntax :

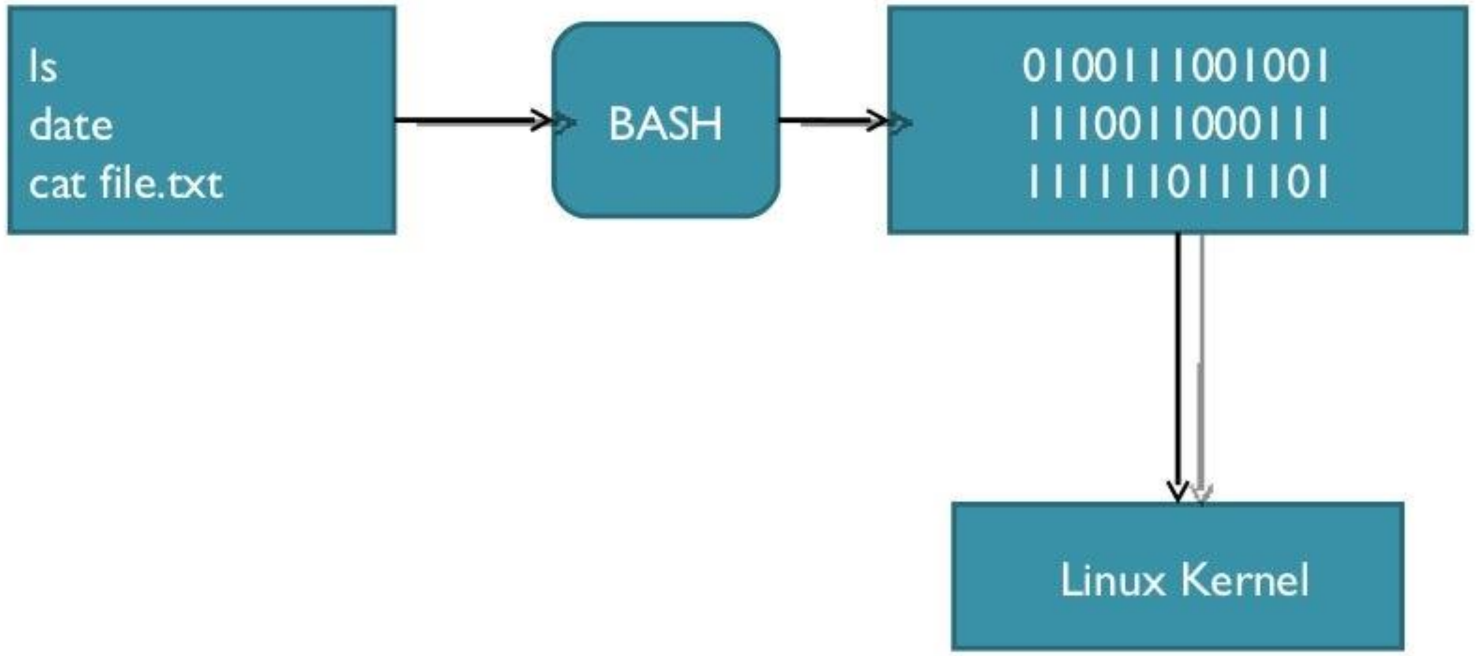
chsh *username new_default_shell*

- The administrator can change your default shell.

This is what Shell Does for US


Your Command or Sell Script
Linux Shell
Converted to Binary Language By Shell
Now Linux Kernel Understand Your Request

Example

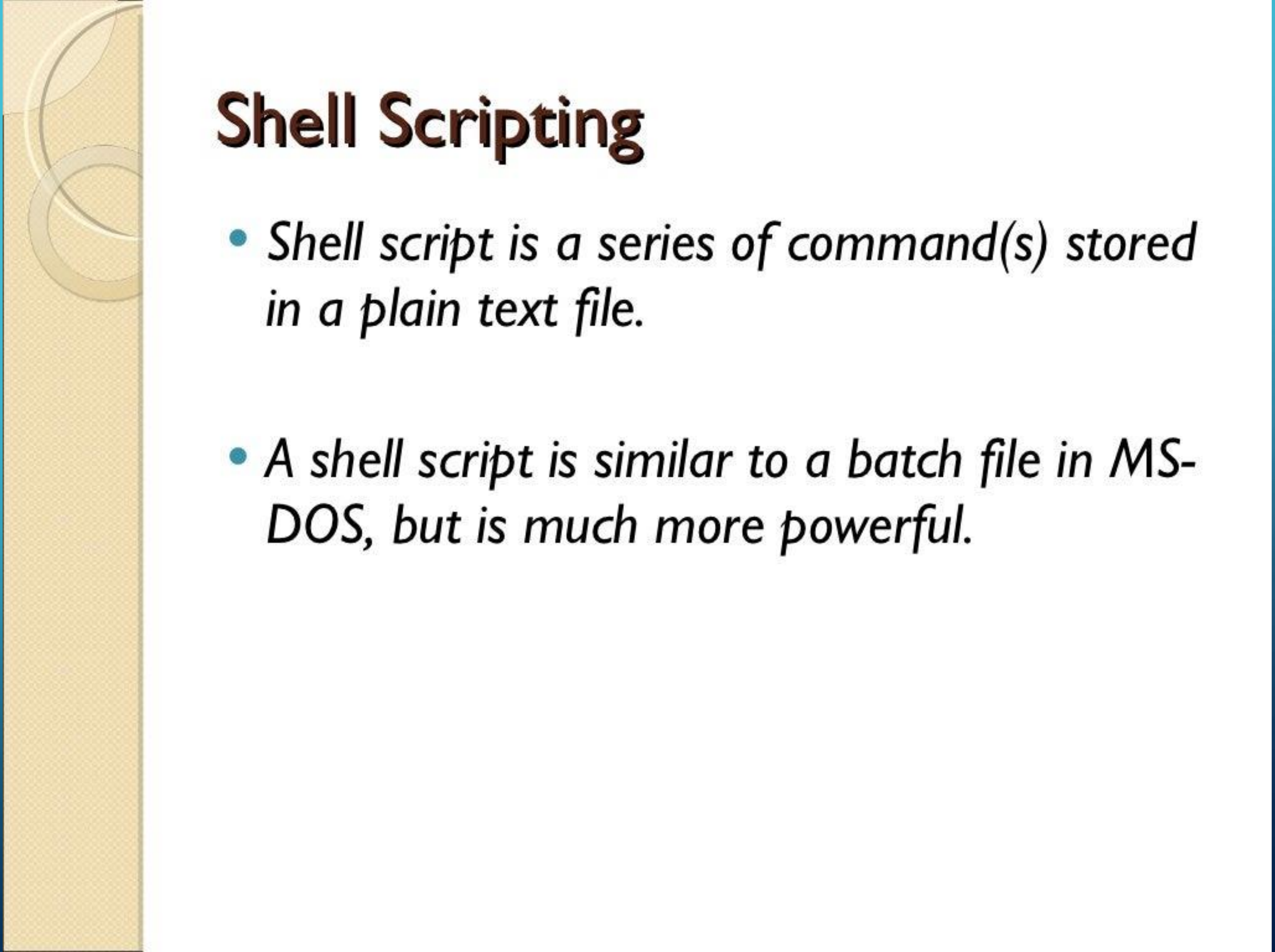


Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

WEEK: 08
SHELL PROGRAMMING 2

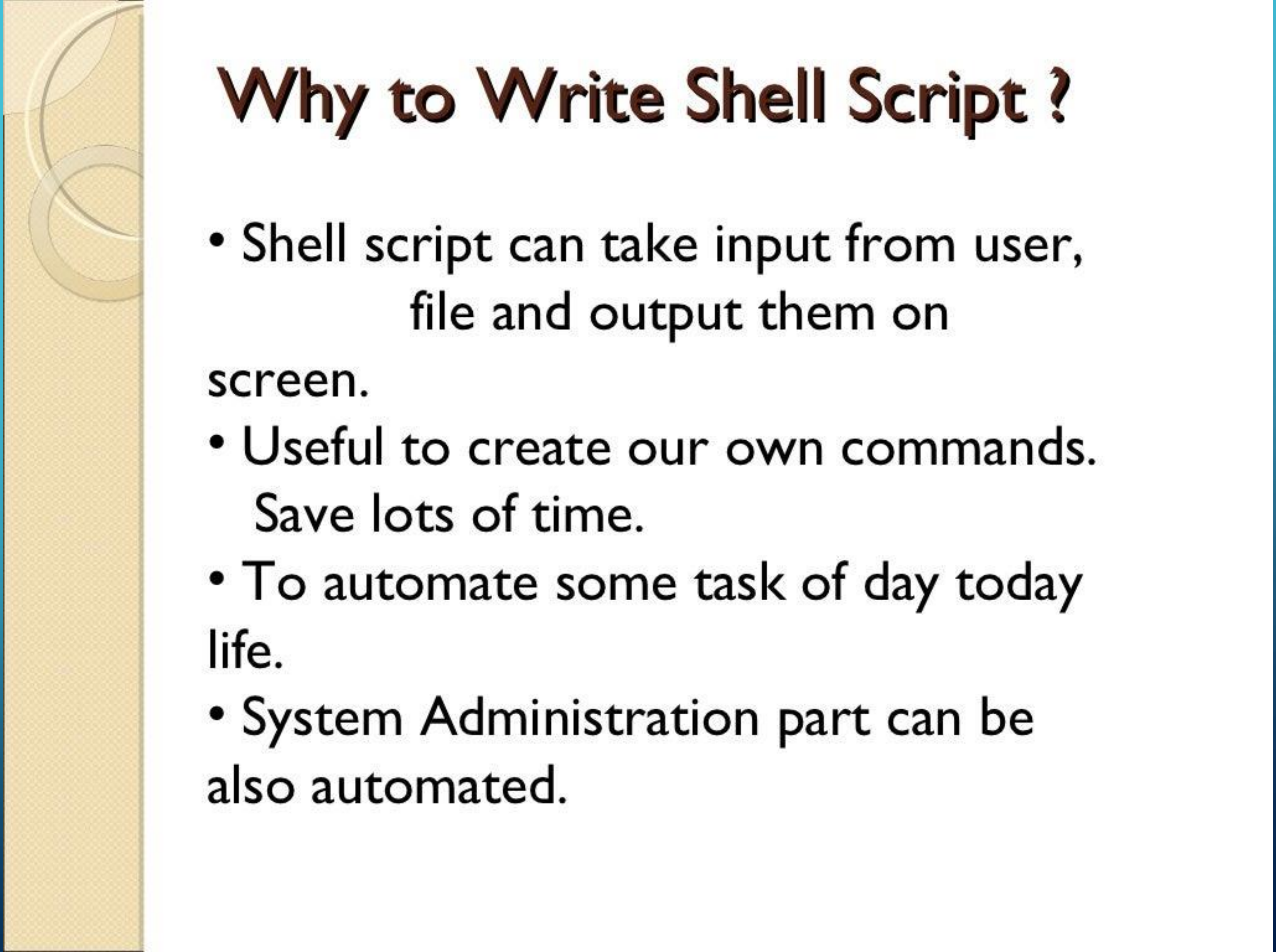
The Shell as a Programming Language

- Now that we've seen some basic shell operations, it's time to move on to scripts.
- There are two ways of writing shell programs.
  1. You can type a sequence of commands and allow the shell to execute them interactively.
  2. You can store those commands in a file that you can then invoke as a program(shell script).



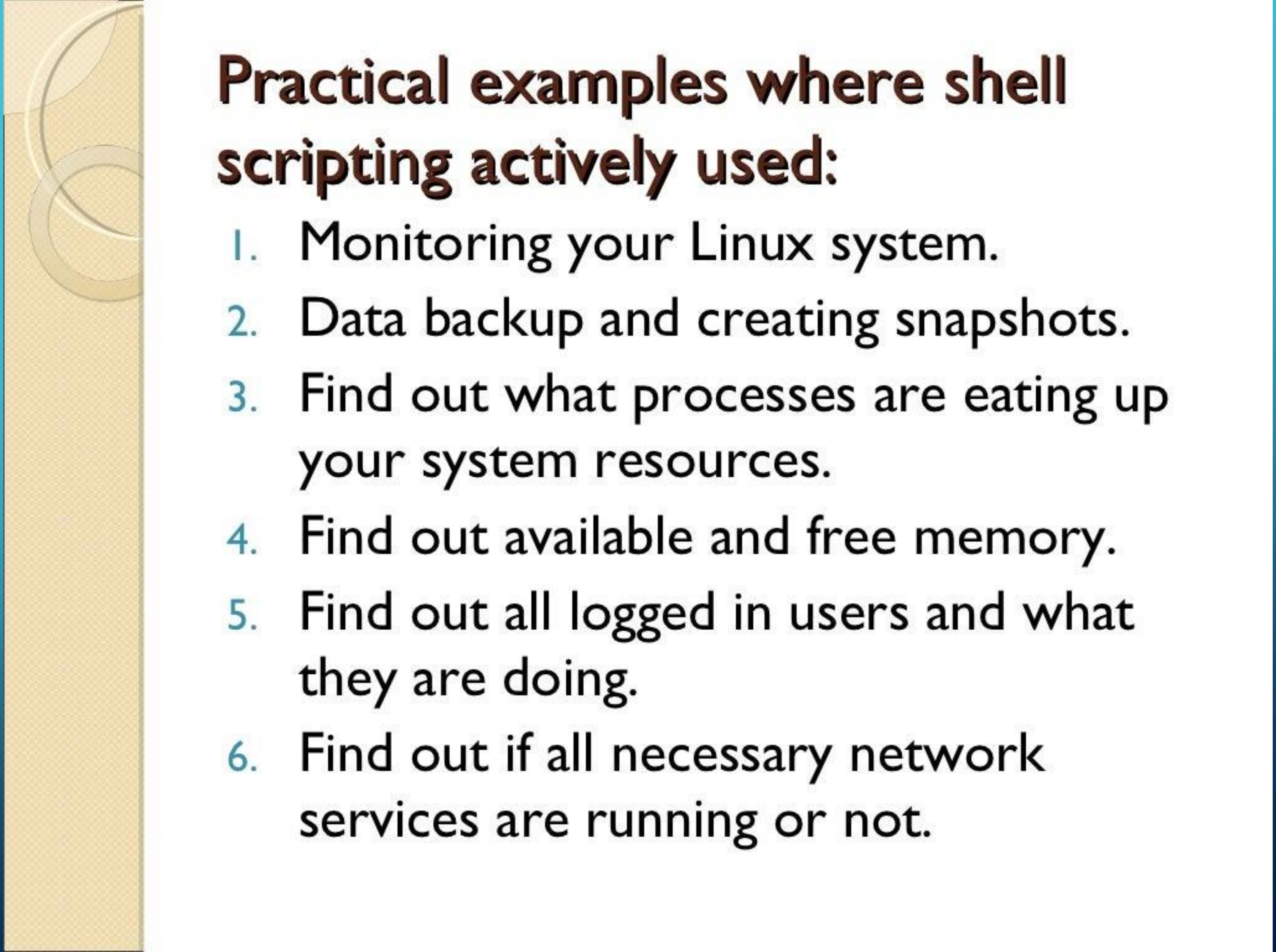
Shell Scripting

- *Shell script is a series of command(s) stored in a plain text file.*
- *A shell script is similar to a batch file in MS-DOS, but is much more powerful.*



Why to Write Shell Script ?

- Shell script can take input from user,
  file and output them on
  screen.
- Useful to create our own commands.
  Save lots of time.
- To automate some task of day today
  life.
- System Administration part can be
  also automated.



Practical examples where shell scripting actively used:

1. Monitoring your Linux system.
2. Data backup and creating snapshots.
3. Find out what processes are eating up your system resources.
4. Find out available and free memory.
5. Find out all logged in users and what they are doing.
6. Find out if all necessary network services are running or not.

When to Use Shell Script

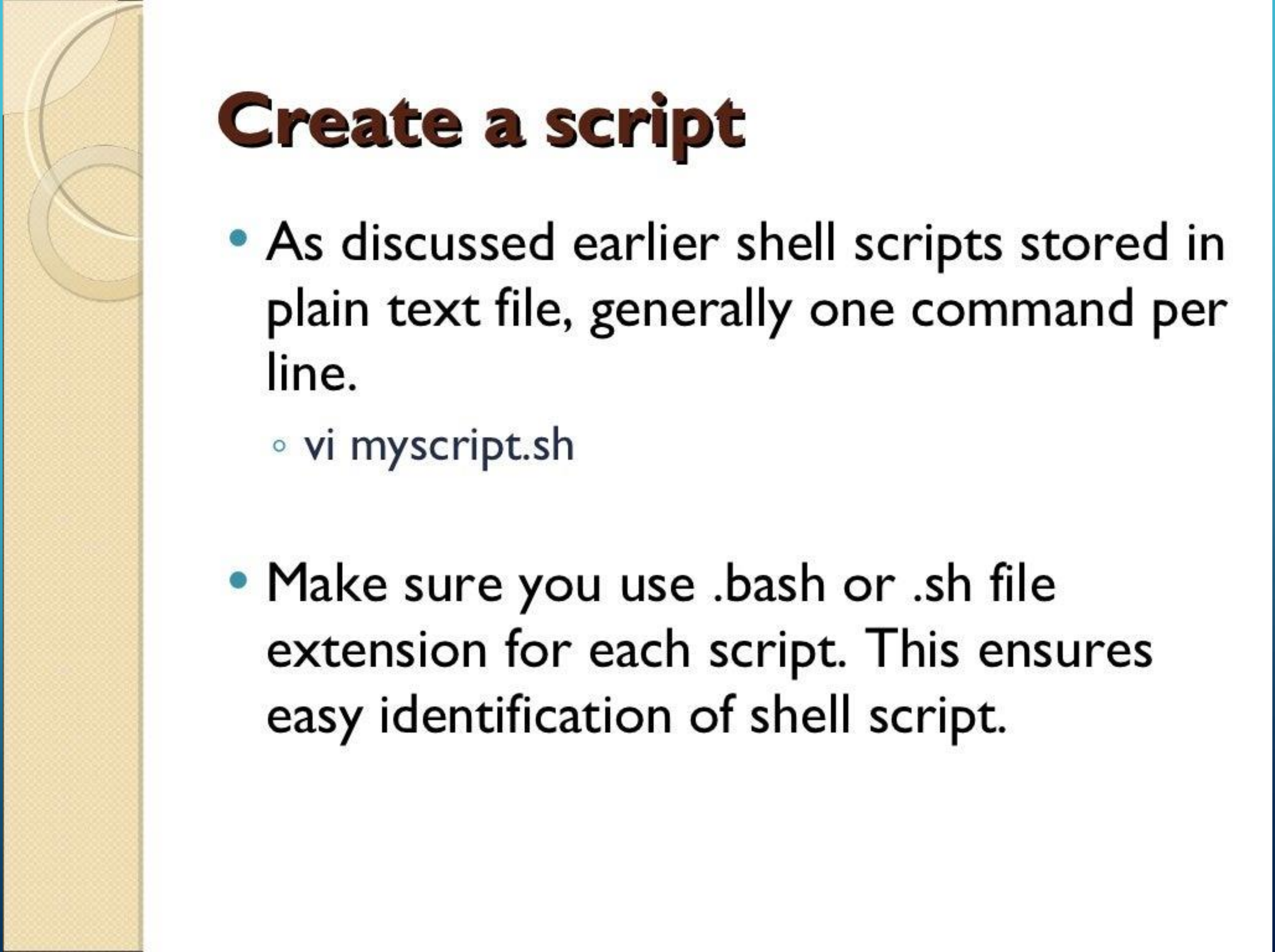
Shell scripts can be used for a variety of tasks

- Customizing your work environment
    - Every time login to see current date, welcome message etc
- Automating your daily tasks
    - To take backup all your programs at the end of the day
- Automating repetitive tasks
    - Producing sales report of every month etc
- Executing important system procedures
    - Shutdown, formatting a disk, creating a file system on it, mounting and un mounting the disk etc
- Performing same operation on many files
    - Replacing printf with myprintf in all C programs present in a dir etc

When Not to Use Shell Scripting

When the task :

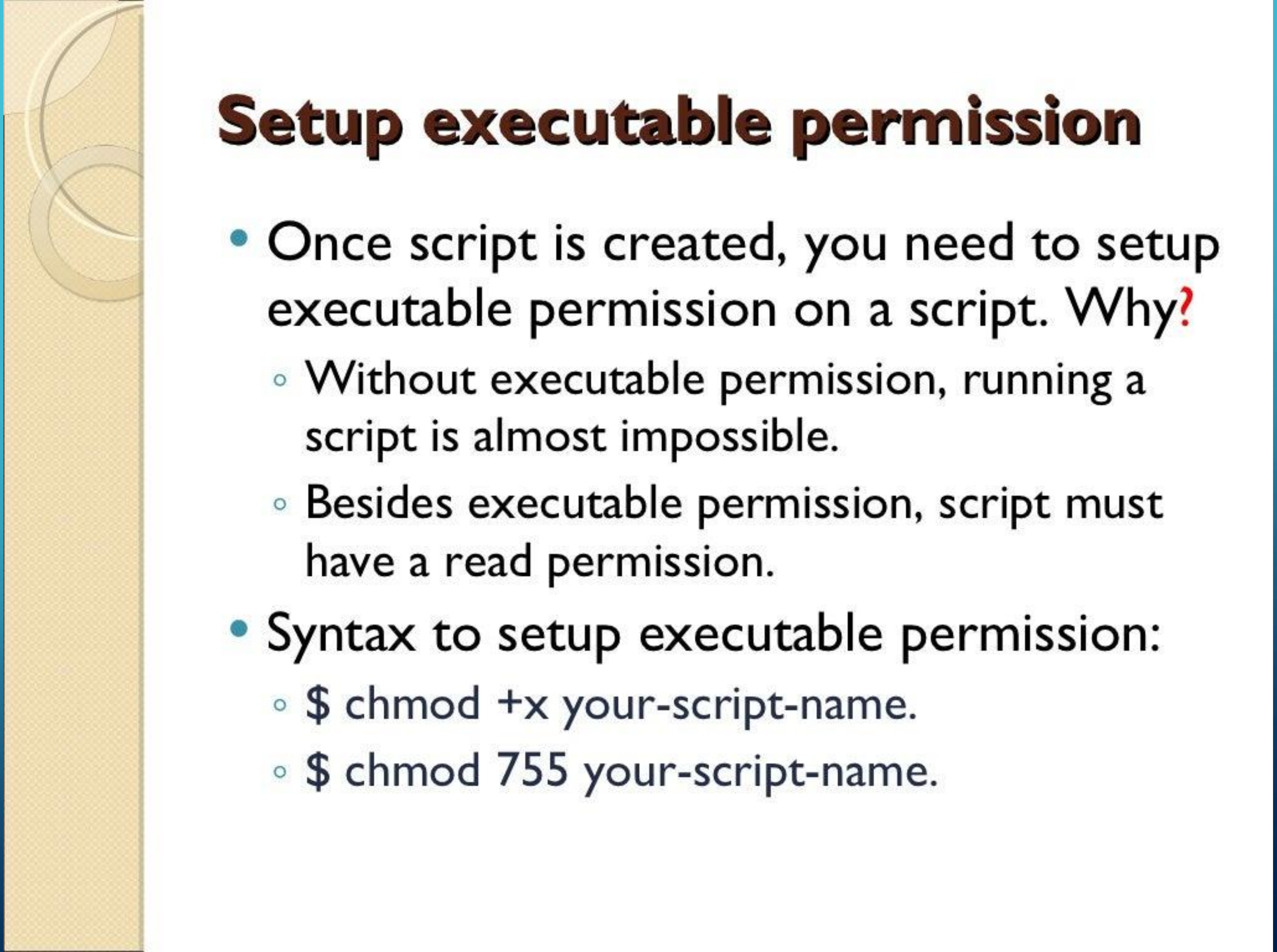
- is too complex such as writing entire billing system
- Require a high degree of efficiency
- Requires a variety of software tools



Create a script

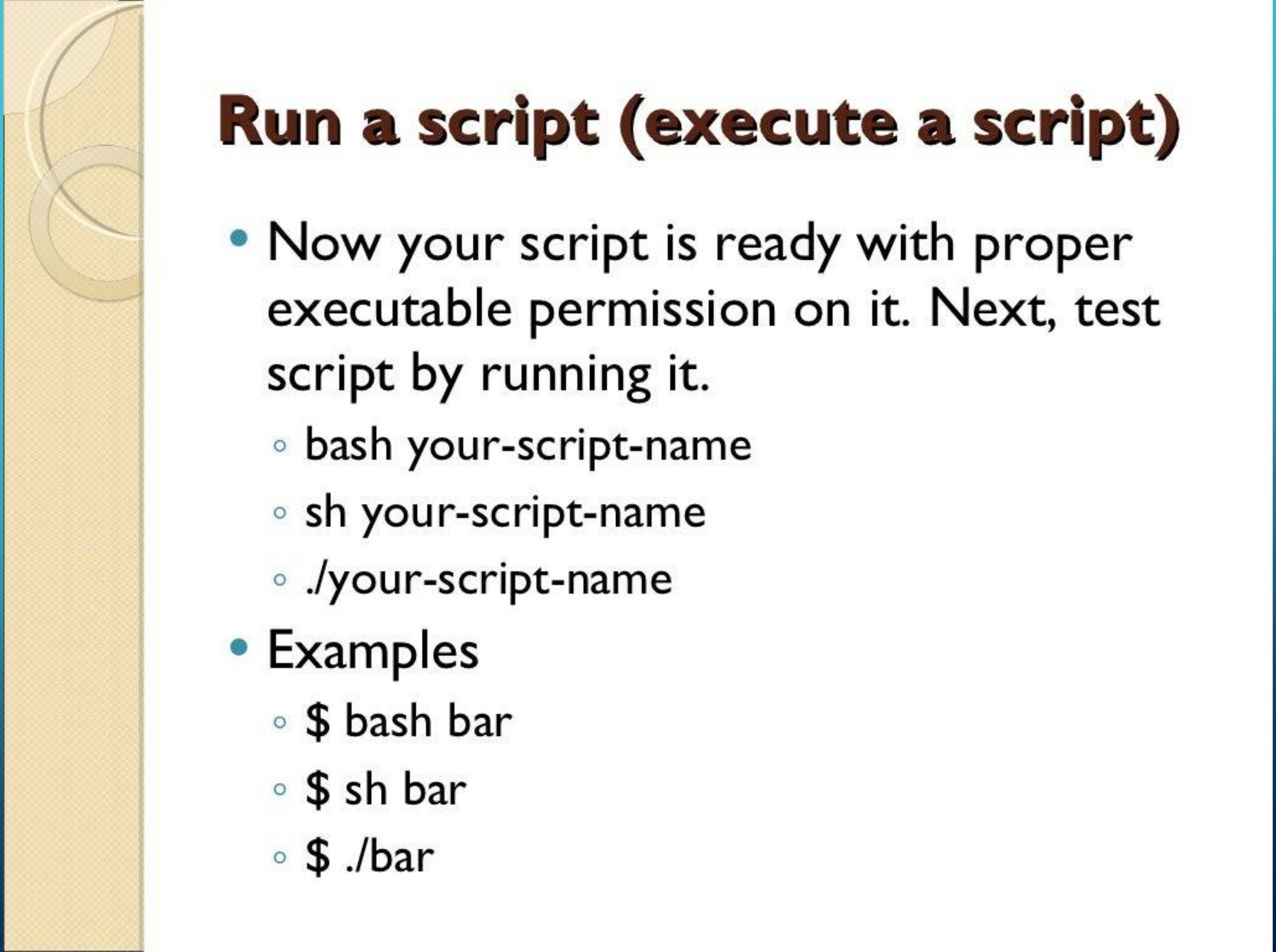
- As discussed earlier shell scripts stored in plain text file, generally one command per line.
  - vi myscript.sh

- Make sure you use .bash or .sh file extension for each script. This ensures easy identification of shell script.



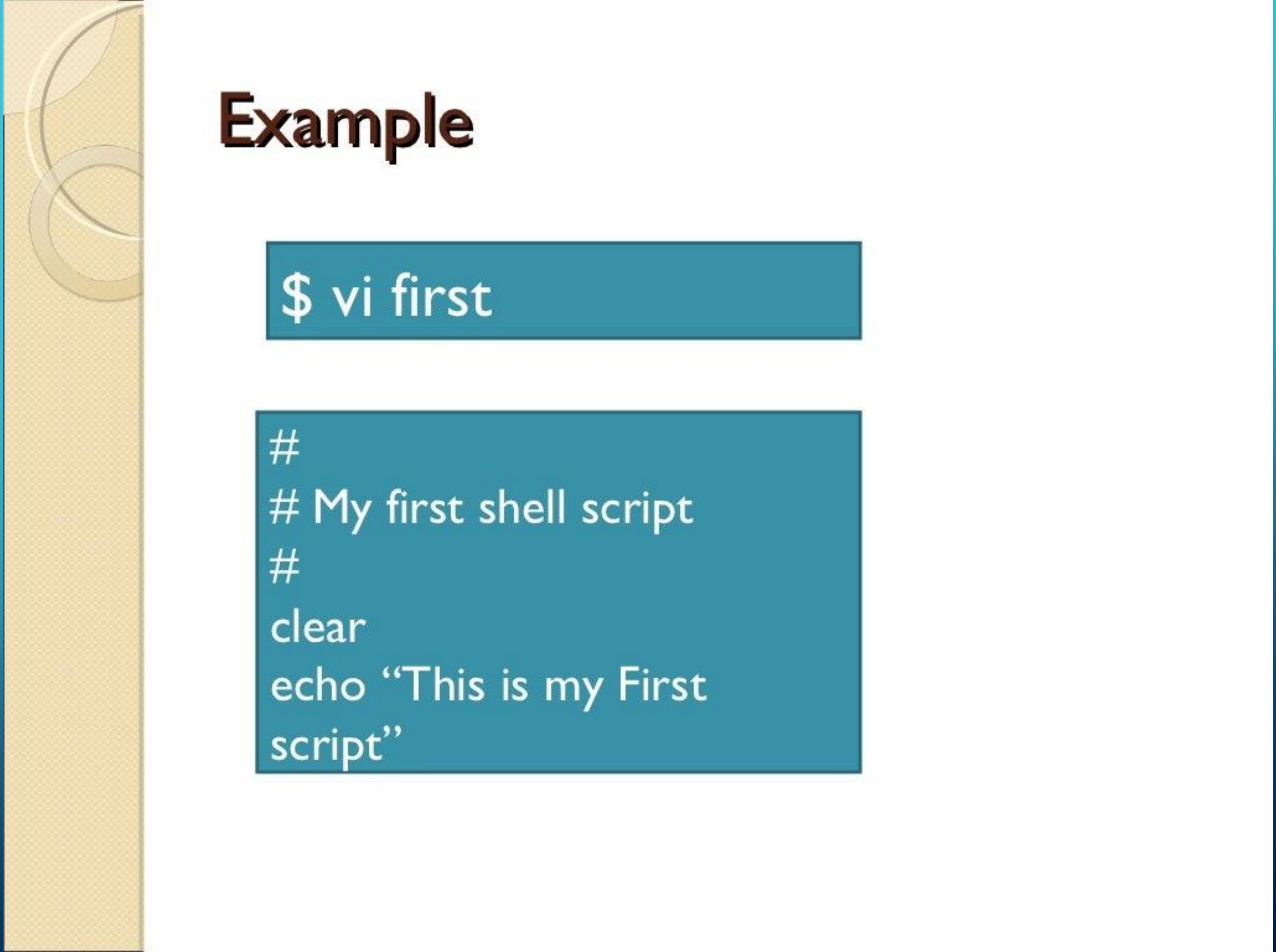
Setup executable permission

- Once script is created, you need to setup executable permission on a script. Why?
  - Without executable permission, running a script is almost impossible.
  - Besides executable permission, script must have a read permission.
- Syntax to setup executable permission:
  - $ chmod +x your-script-name.
  - $ chmod 755 your-script-name.



Run a script (execute a script)

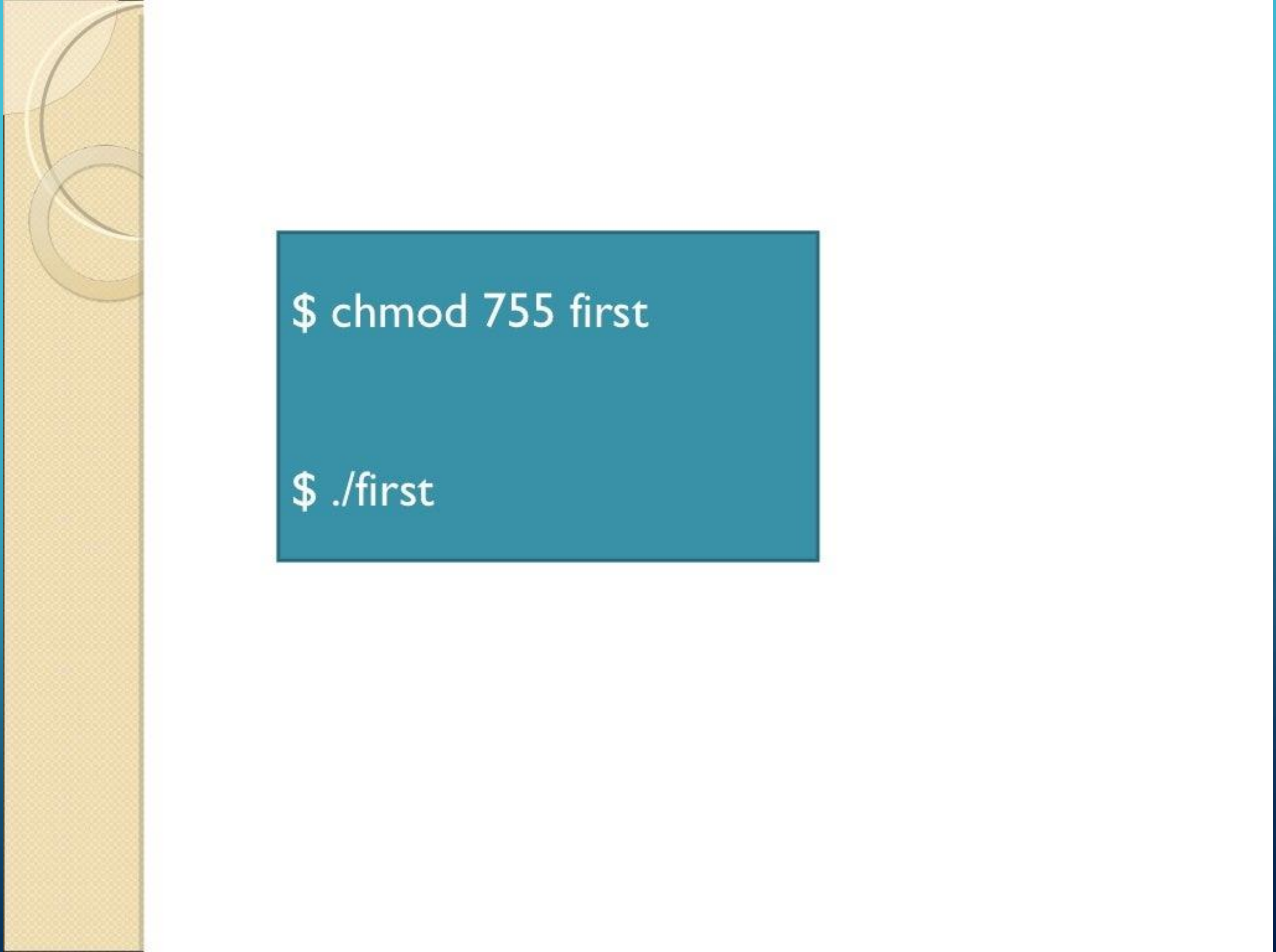
- Now your script is ready with proper executable permission on it. Next, test script by running it.
  - bash your-script-name
  - sh your-script-name
  - ./your-script-name
- Examples
  - $ bash bar
  - $ sh bar
  - $ ./bar



Example

```
$ vi first
```

```
#
# My first shell script
#
clear
echo "This is my First
script"
```



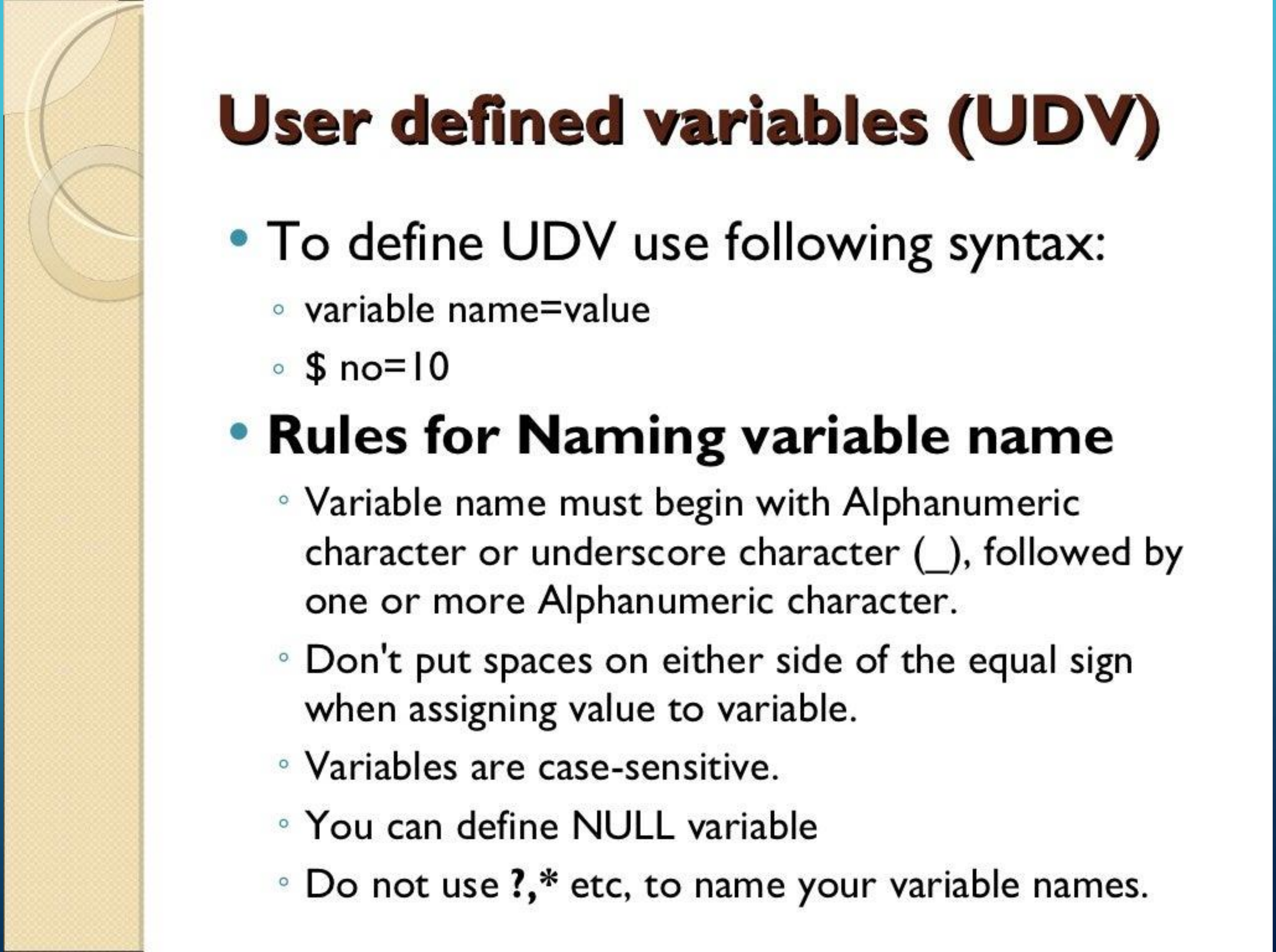
```
$ chmod 755 first

$ ./first
```

WEEK: 09
SHELL PROGRAMMING 3

Variables in Shell

- In Linux (Shell), there are two types of variable:
  - **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
  - **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.



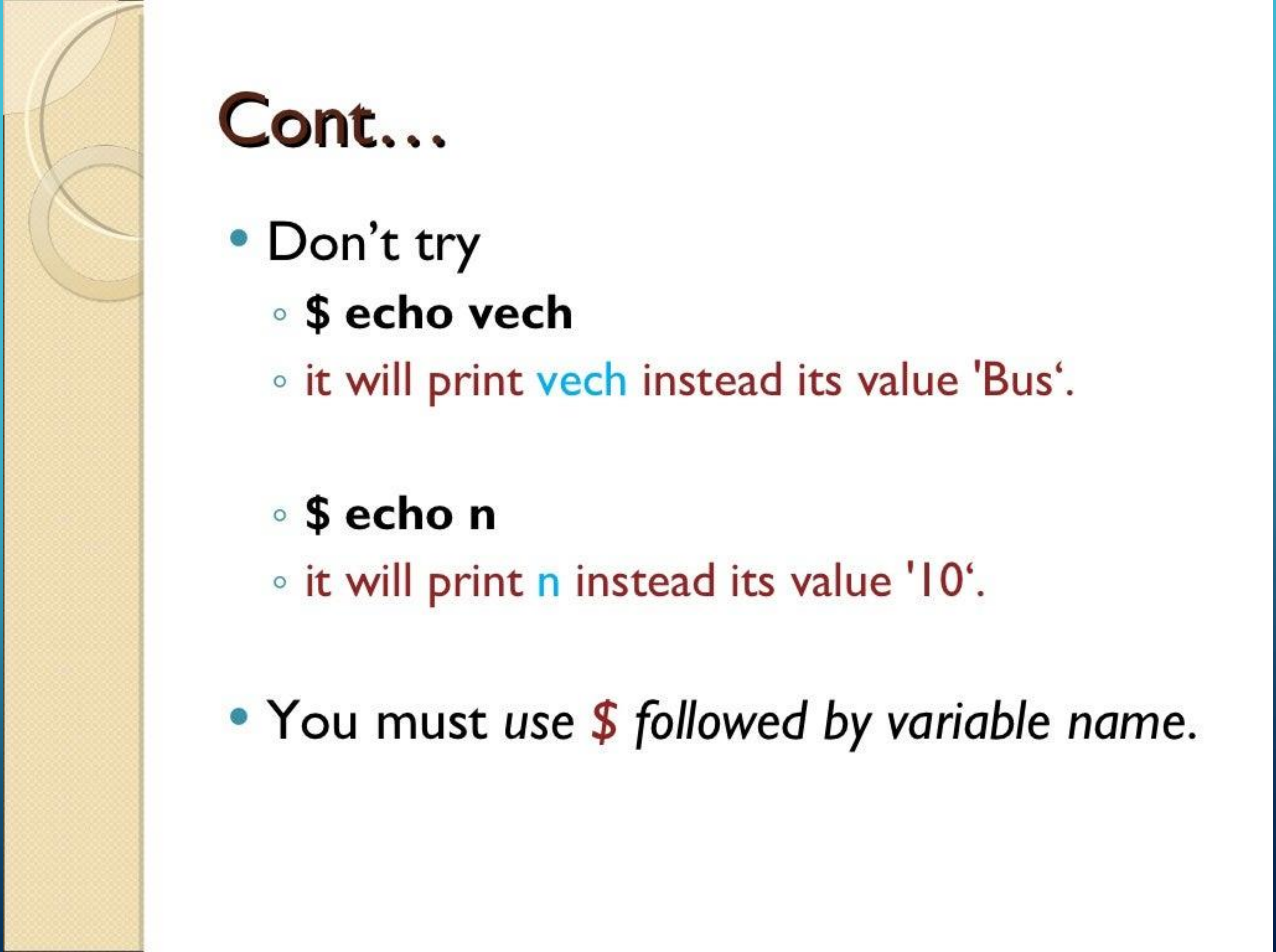
User defined variables (UDV)

- To define UDV use following syntax:
  - variable name=value
  - $ no=10
- **Rules for Naming variable name**
  - Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character.
  - Don't put spaces on either side of the equal sign when assigning value to variable.
  - Variables are case-sensitive.
  - You can define NULL variable
  - Do not use **?,*** etc, to name your variable names.

Print or access value of UDV

- To print or access UDV use following syntax :
  - $variablename.
- Examples:
  - $vech=Bus
  - $ n=10

  - $ echo $vech
  - $ echo $n



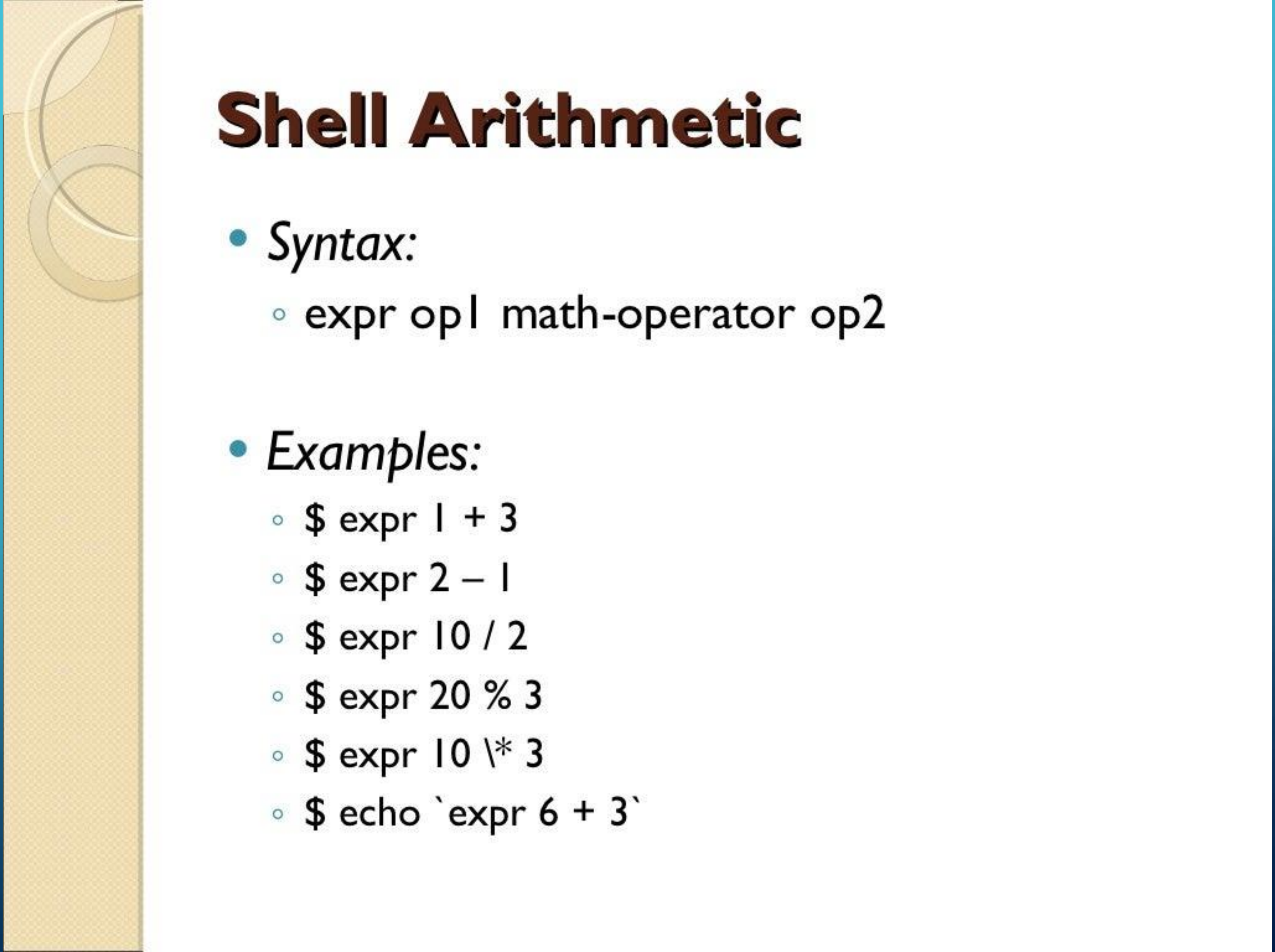
Cont…

- Don’t try
  - **$ echo vech**
  - it will print vech instead its value 'Bus‘.

  - **$ echo n**
  - it will print n instead its value '10‘.
- You must *use* $ *followed by variable name.*

Class work

1. Define variable x with value 10 and print it on screen.
2. Define variable xn with value SUST and print it on screen.
3. print sum of two numbers, let's say 6 and 3 .

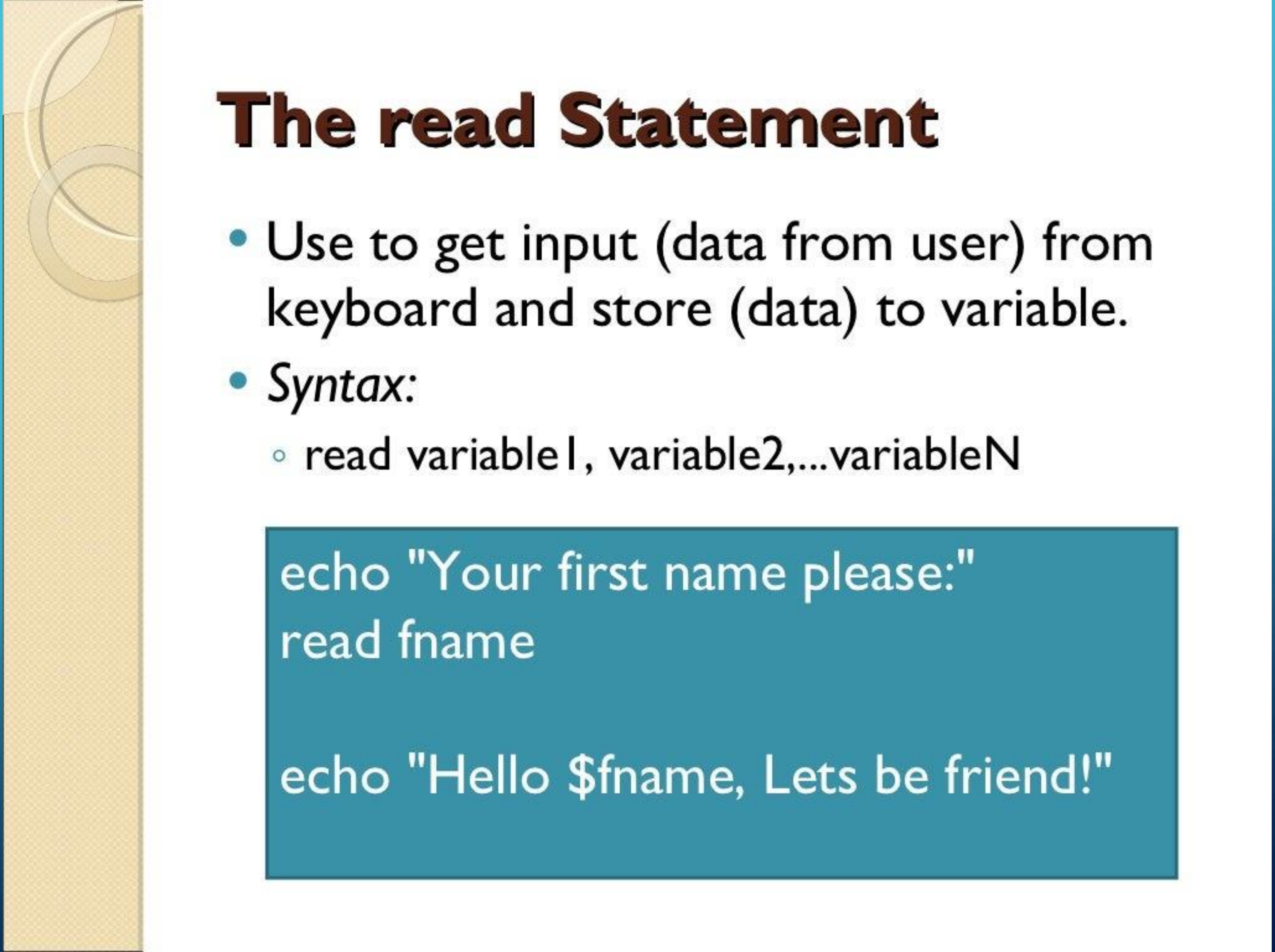


Shell Arithmetic

- *Syntax:*
  - expr op1 math-operator op2

- *Examples:*

```
$ expr 1 + 3
$ expr 2 – 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 \* 3
$ echo `expr 6 + 3`
```

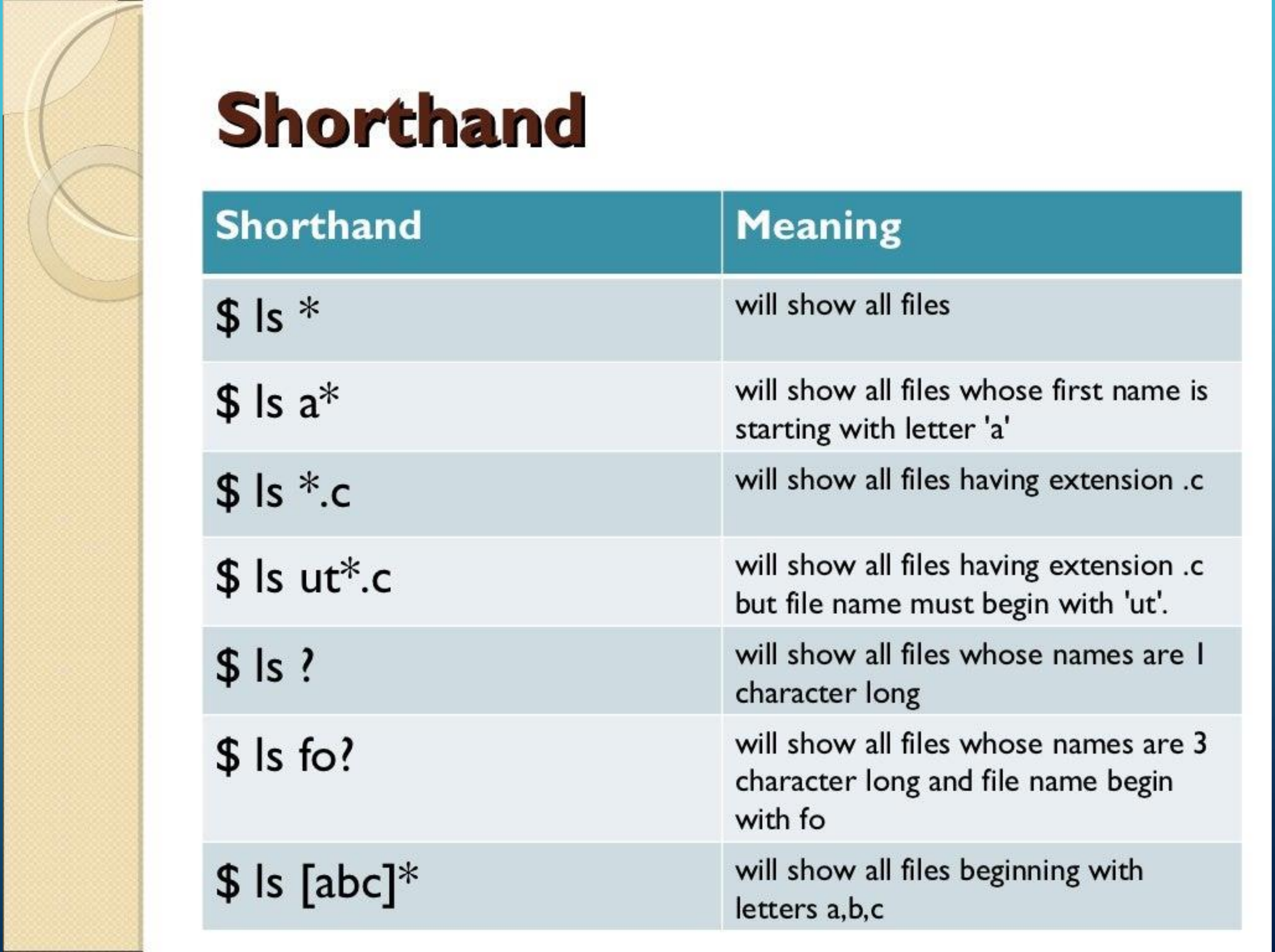


The read Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
- *Syntax:*
  - read variable1, variable2,...variableN

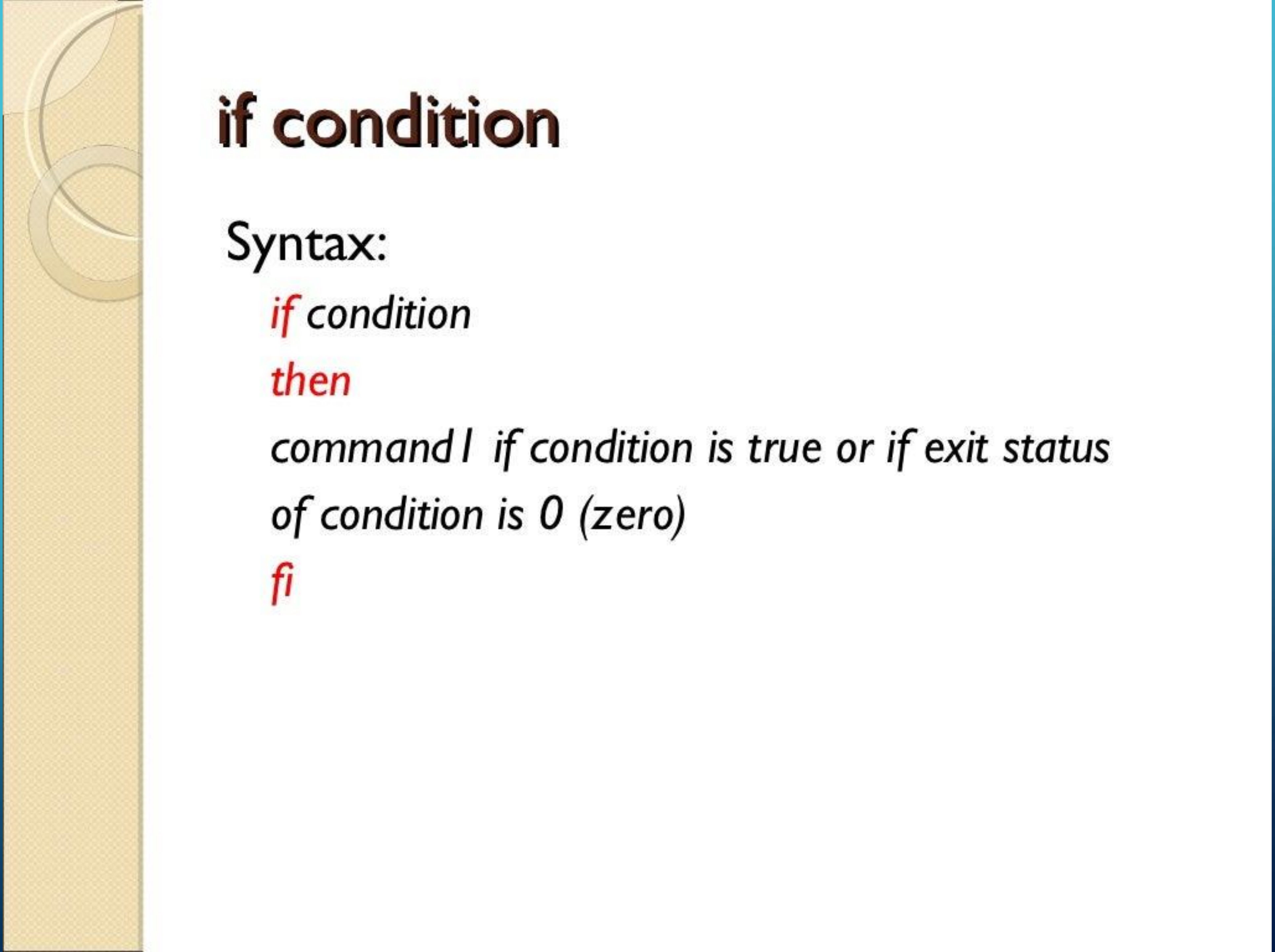
```
echo "Your first name please:"
read fname

echo "Hello $fname, Lets be friend!"
```



Shorthand

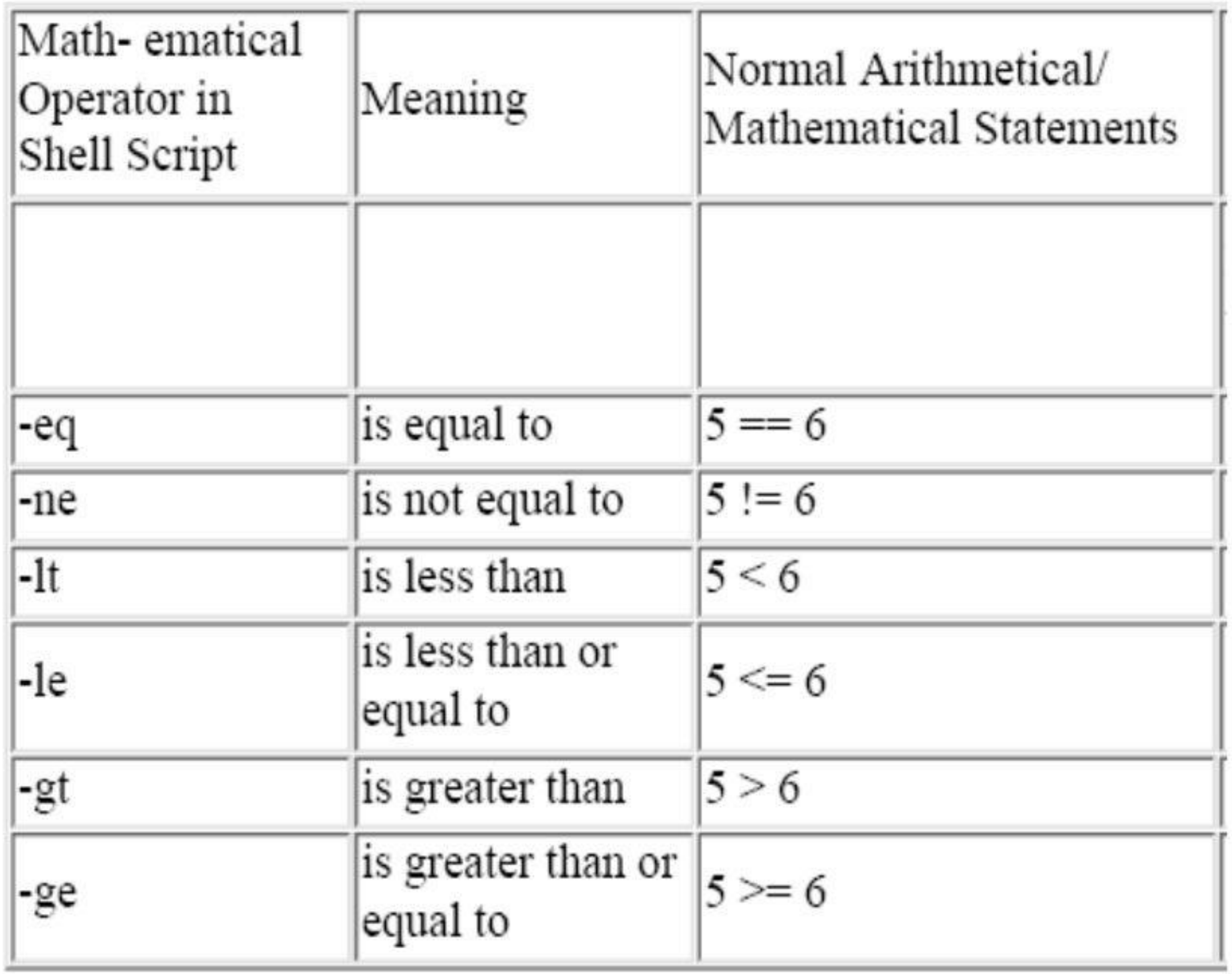
Shorthand	Meaning
$ ls *	will show all files
$ ls a*	will show all files whose first name is starting with letter 'a'
$ ls *.c	will show all files having extension .c
$ ls ut*.c	will show all files having extension .c but file name must begin with 'ut'.
$ ls ?	will show all files whose names are 1 character long
$ ls fo?	will show all files whose names are 3 character long and file name begin with fo
$ ls [abc]*	will show all files beginning with letters a,b,c



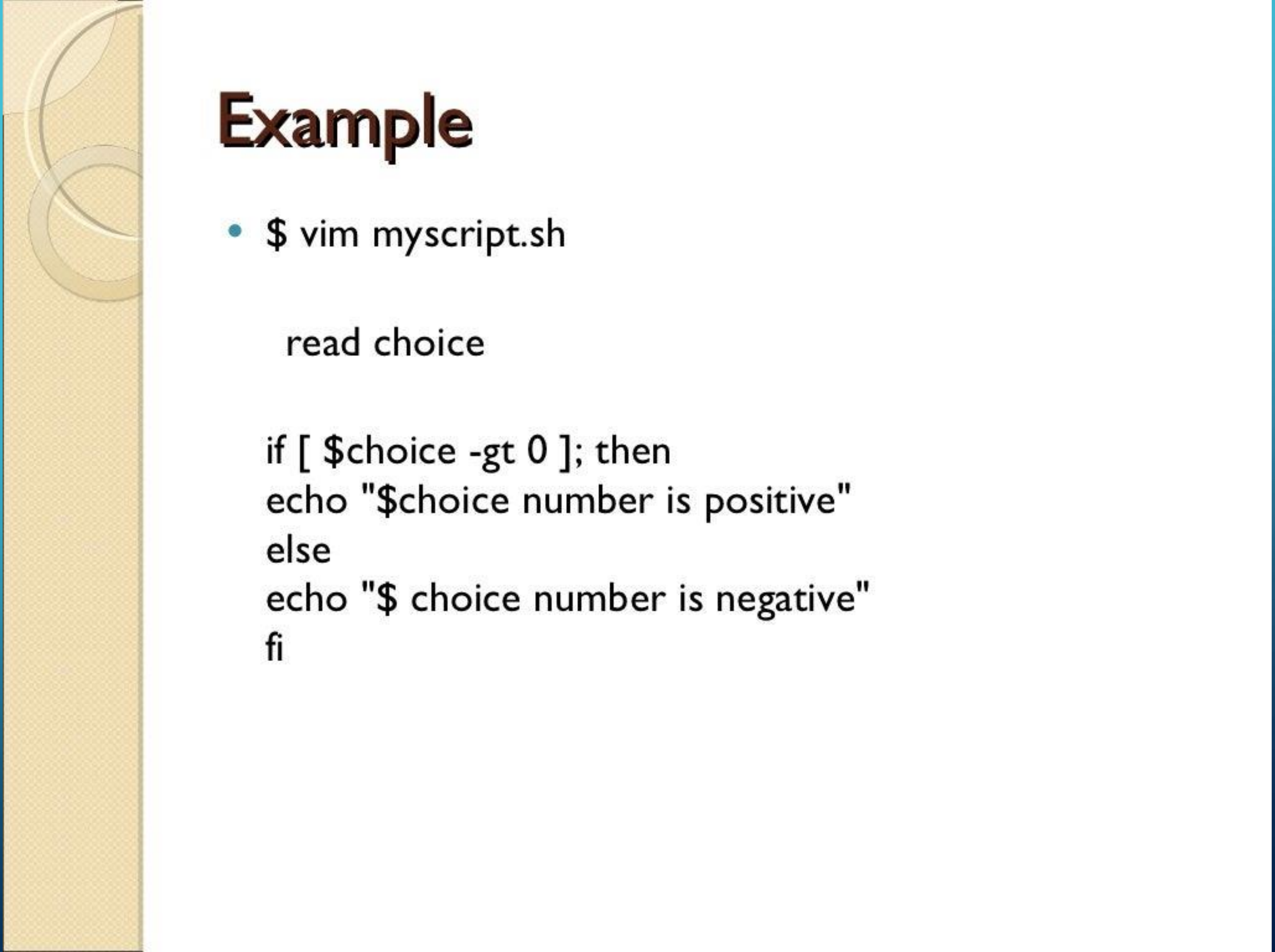
if condition

Syntax:

```
if condition
then
command1  if condition is true or if exit status
of condition is 0 (zero)
fi
```



Math- ematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements
-eq	is equal to	5 == 6
-ne	is not equal to	5 != 6
-lt	is less than	5 < 6
-le	is less than or equal to	5 <= 6
-gt	is greater than	5 > 6
-ge	is greater than or equal to	5 >= 6



Example

- $ vim myscript.sh

```
  read choice

if [ $choice -gt 0 ]; then
echo "$choice number is positive"
else
echo "$ choice number is negative"
fi
```

Nested if-else-fi

- $ vi nestedif.sh

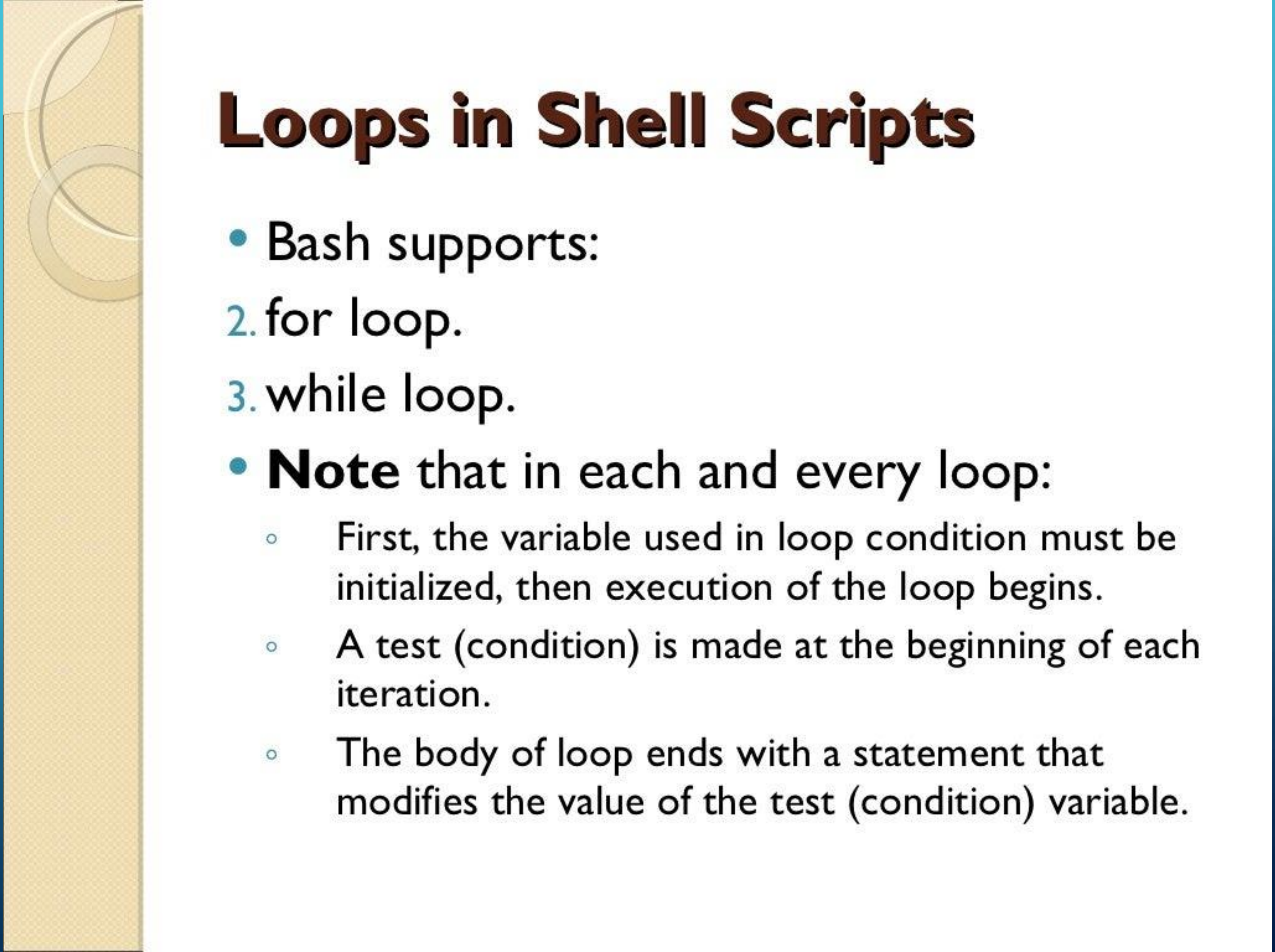
```
echo "1. Unix (Sun Os)"
echo "2. Linux (Red Hat)"
echo -n "Select your os choice [1 or 2]? "
read osch

if [ $osch -eq 1 ] ; then

    echo "You Pick up Unix (Sun Os)"

else
      if [ $osch -eq 2 ] ; then
            echo "You Pick up Linux (Red Hat)"
      else
            echo "What you don't like Unix/Linux OS."
     fi
fi
```

WEEK: 10
SHELL PROGRAMMING 4

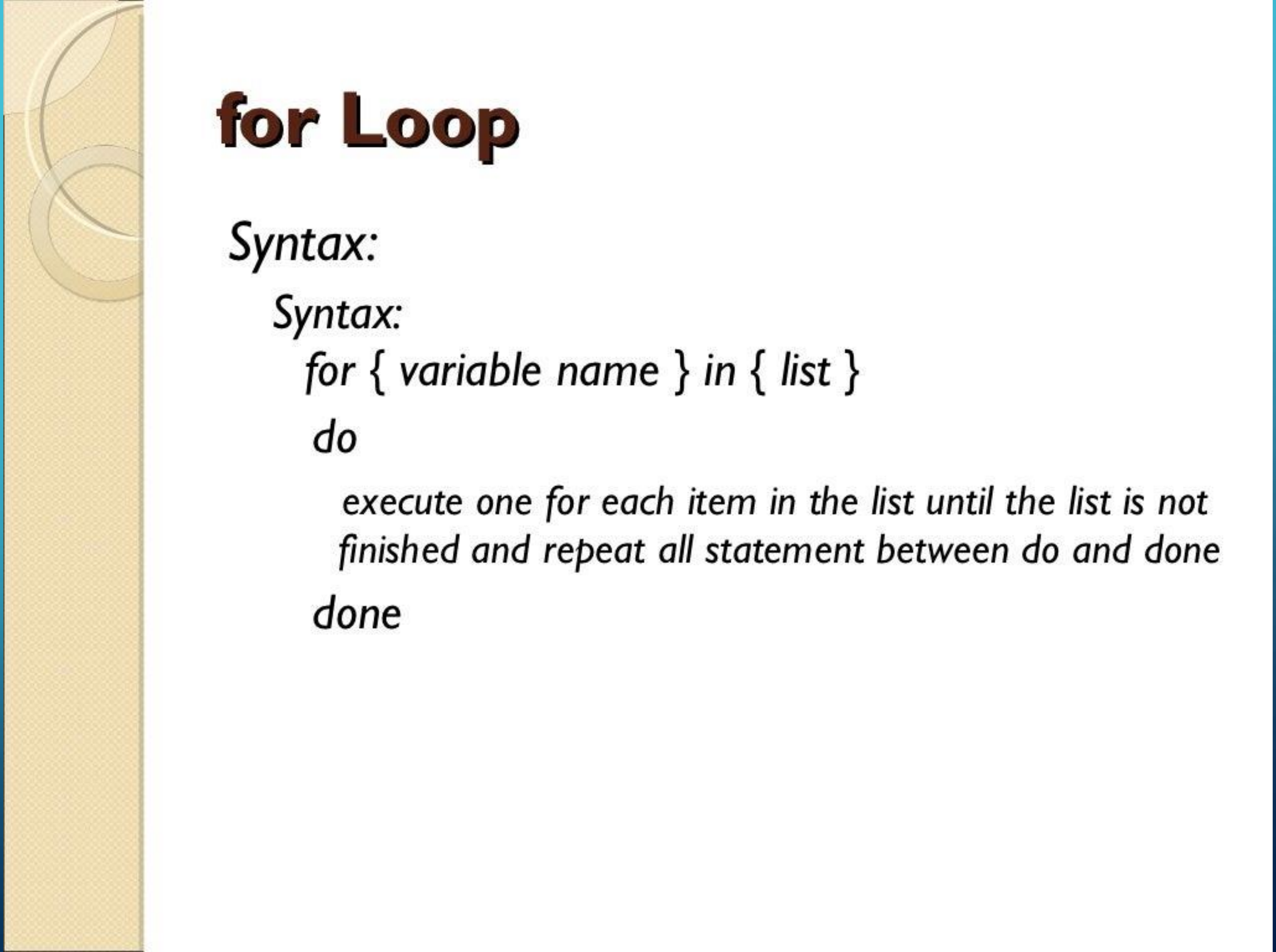


Loops in Shell Scripts

- Bash supports:

2. for loop.
3. while loop.

- **Note** that in each and every loop:
  - First, the variable used in loop condition must be initialized, then execution of the loop begins.
  - A test (condition) is made at the beginning of each iteration.
  - The body of loop ends with a statement that modifies the value of the test (condition) variable.

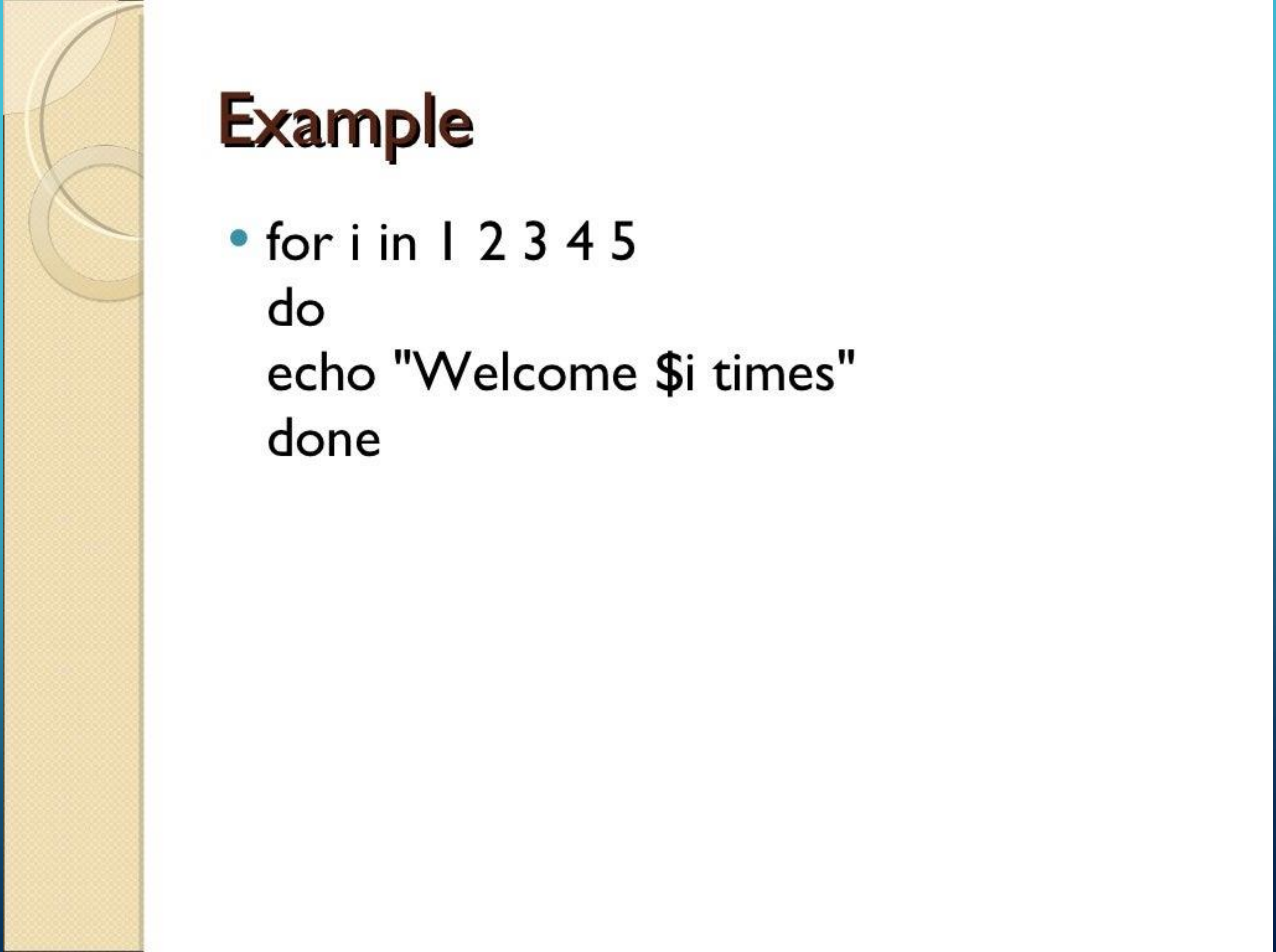


for Loop

Syntax:

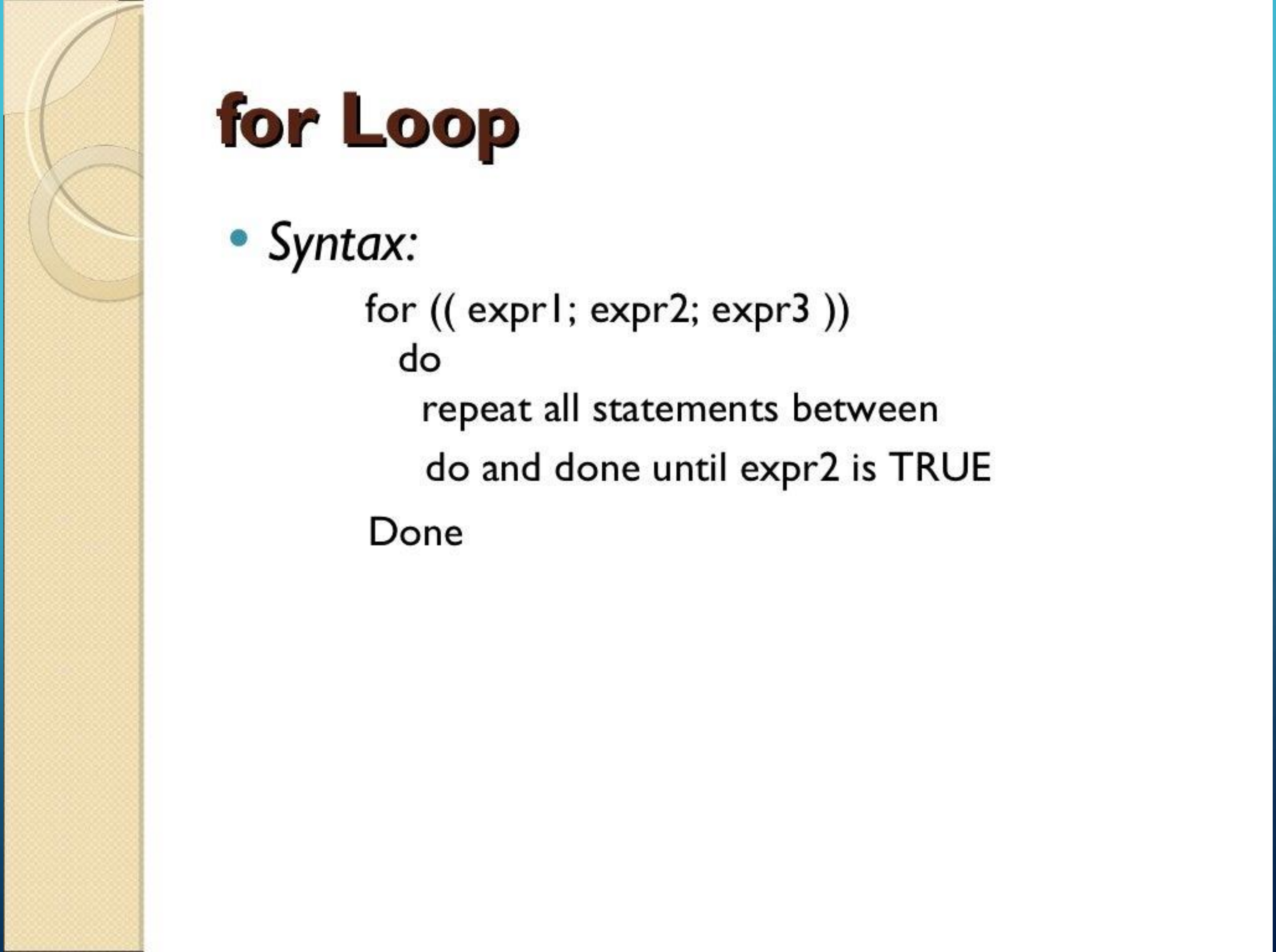
Syntax:

```
for { variable name } in { list }
do
   execute one for each item in the list until the list is not
   finished and repeat all statement between do and done
done
```



Example

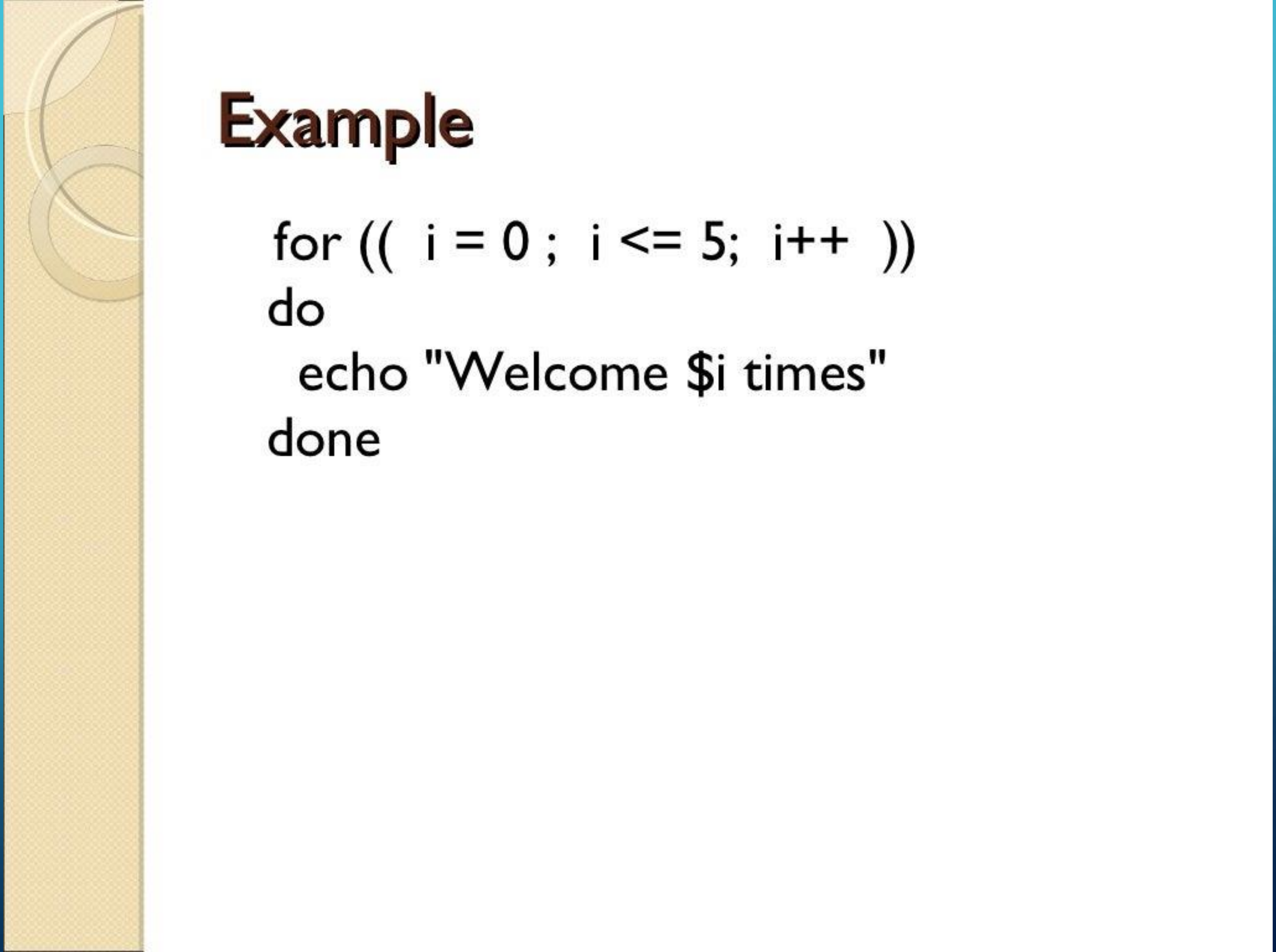
-
  ```
  for i in 1 2 3 4 5
  do
  echo "Welcome $i times"
  done
  ```



for Loop

- *Syntax:*

```
for (( expr1; expr2; expr3 ))
  do
    repeat all statements between
    do and done until expr2 is TRUE
Done
```



Example

```
for ((  i = 0 ;  i <= 5;  i++  ))
do
  echo "Welcome $i times"
done
```

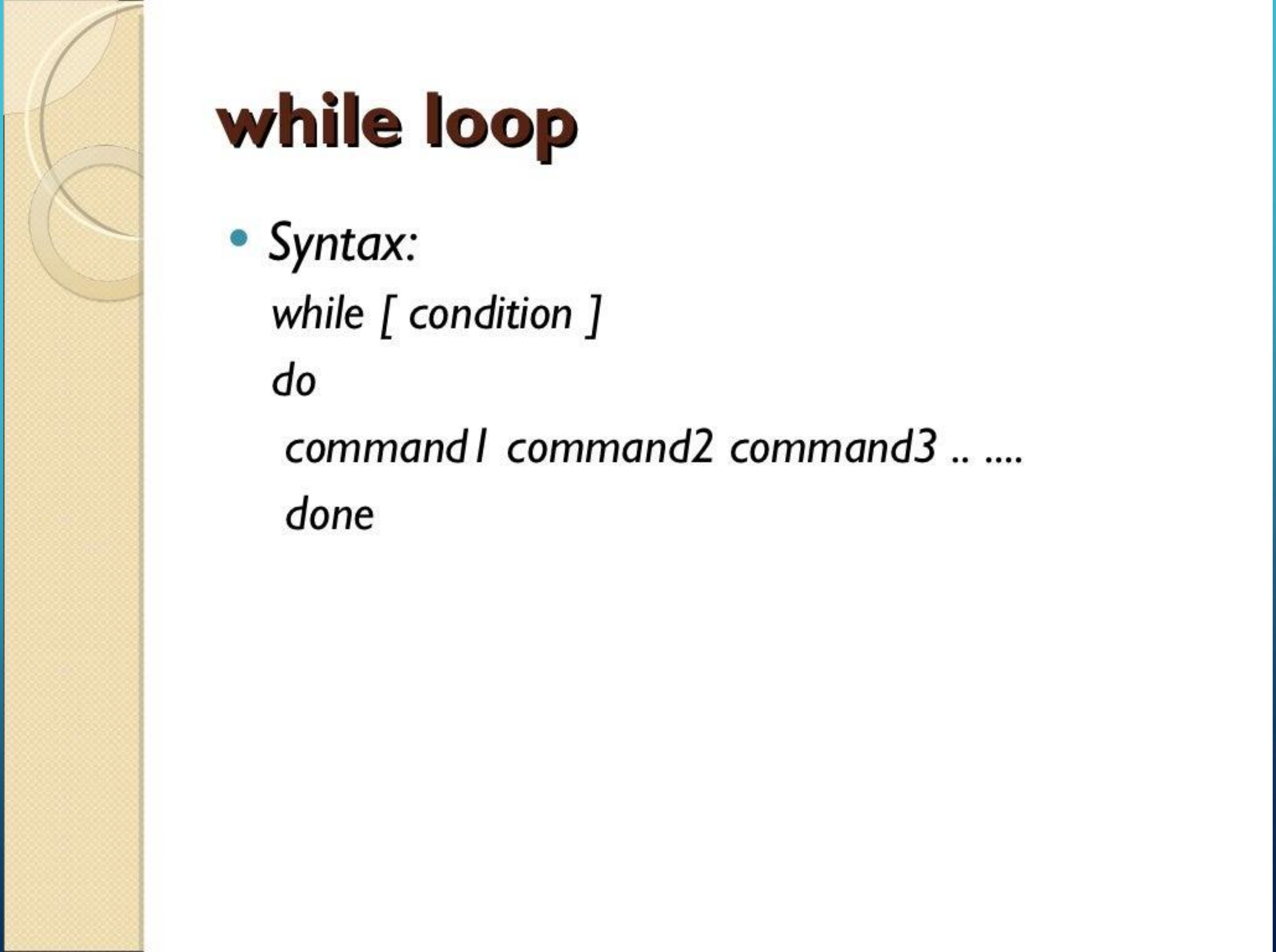
Nesting of for Loop

- $ vi nestedfor.sh

```
for (( i = 1; i <= 5; i++ ))
do

    for (( j = 1 ; j <= 5; j++ ))
    do
          echo -n "$i "
    done

  echo “”
don
```



while loop

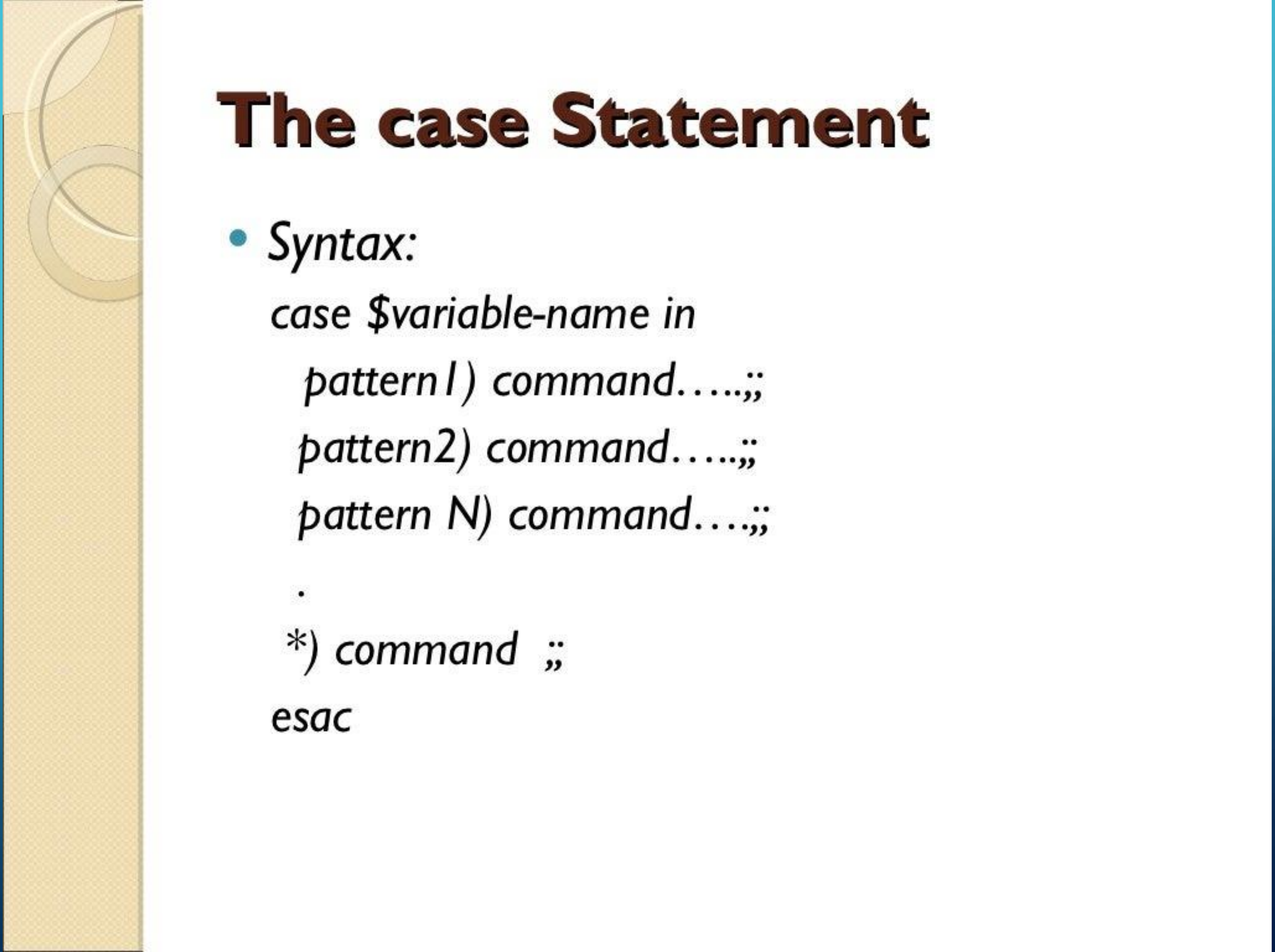
- *Syntax:*

```
while [ condition ]
do
 command1 command2 command3 .. ....
 done
```

Example

```
i=1
while [ $i -le 10 ]
 do
   echo "$n * $i = `expr $i \* $n`"
   i=`expr $i + 1`
 done
```

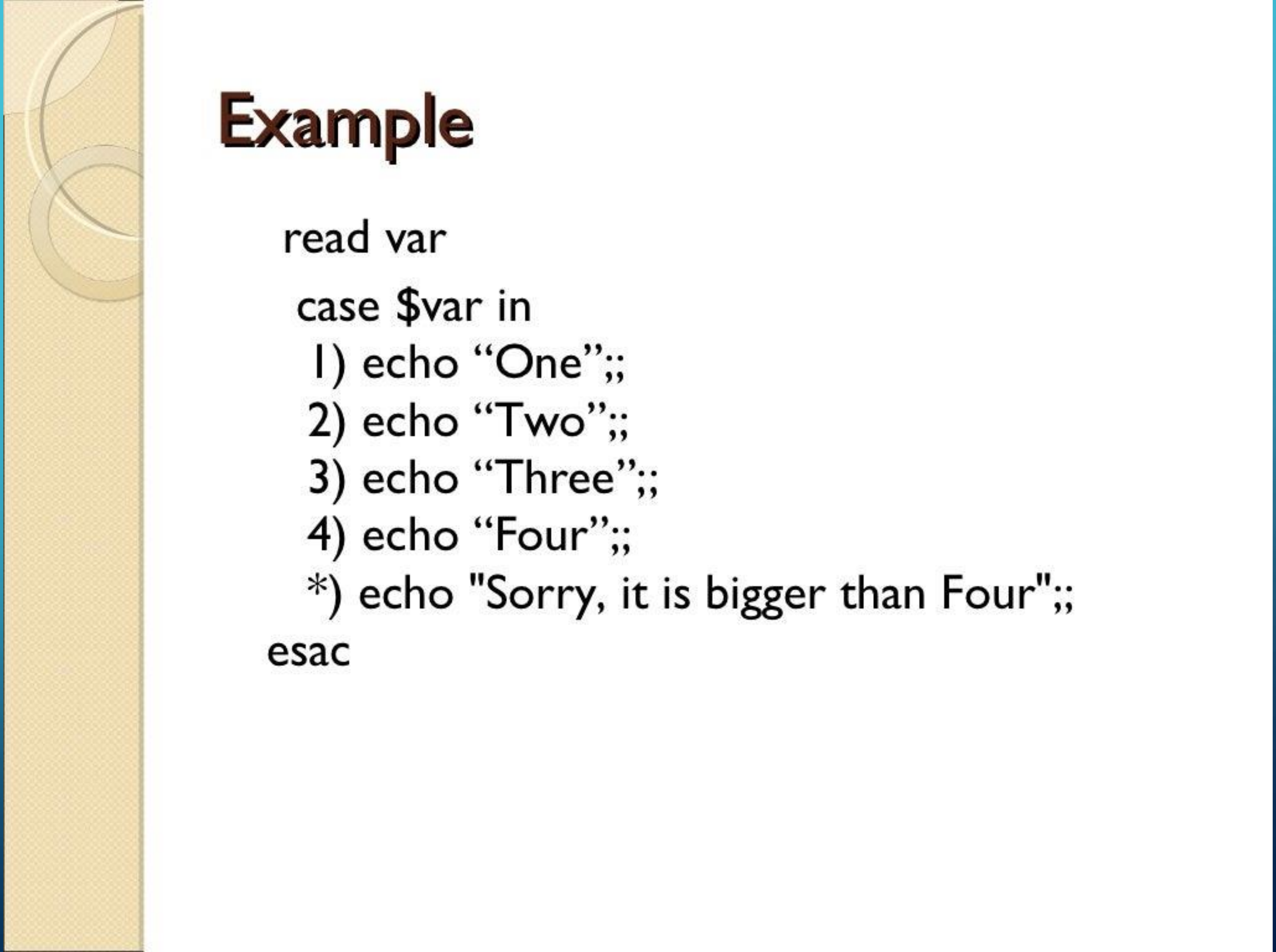
WEEK: 11
SHELL PROGRAMMING 5



The case Statement

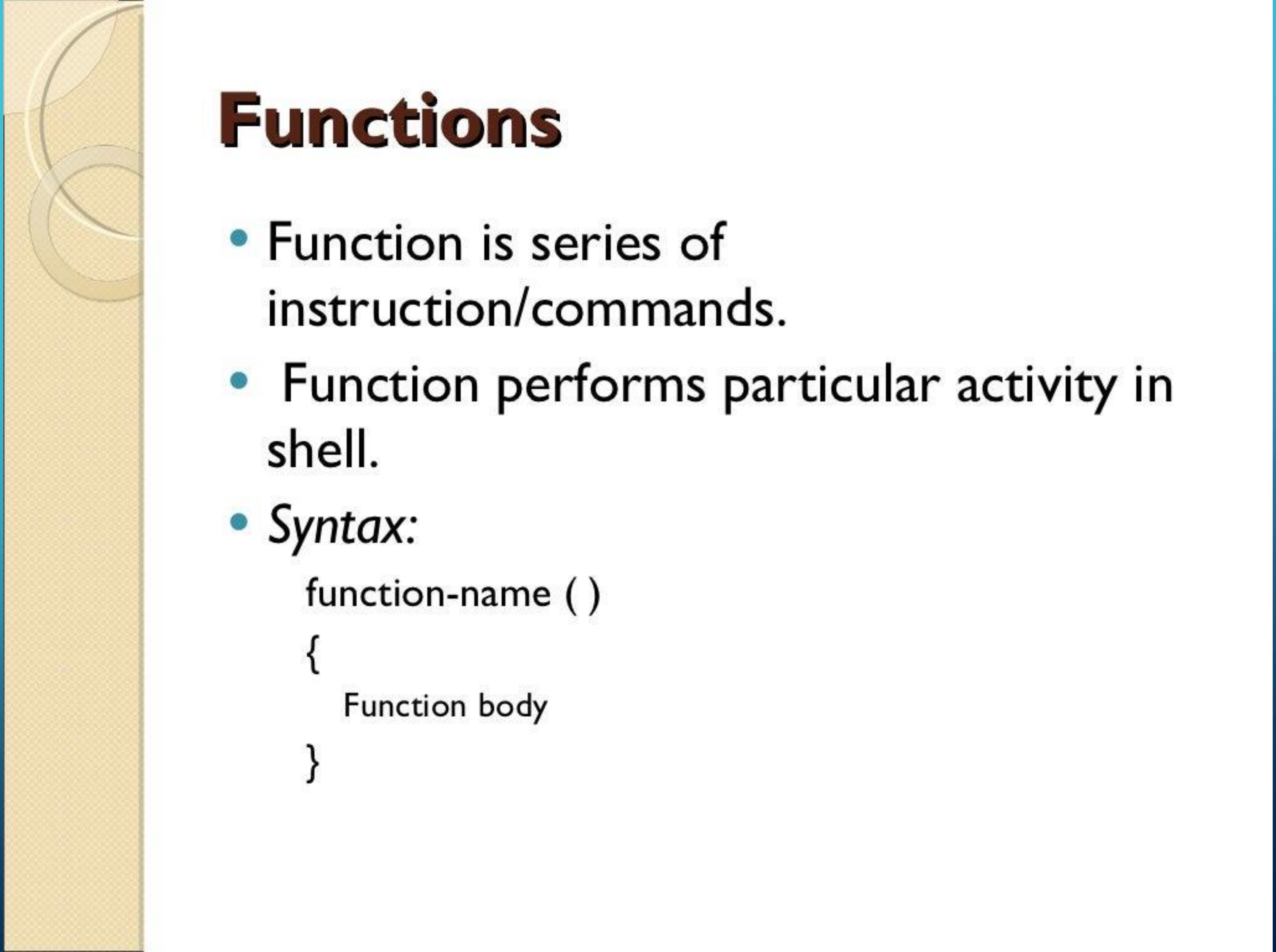
- *Syntax:*

```
case $variable-name in
   pattern1) command…..;;
  pattern2) command…..;;
  pattern N) command….;;
  .
 *) command  ;;
esac
```



Example

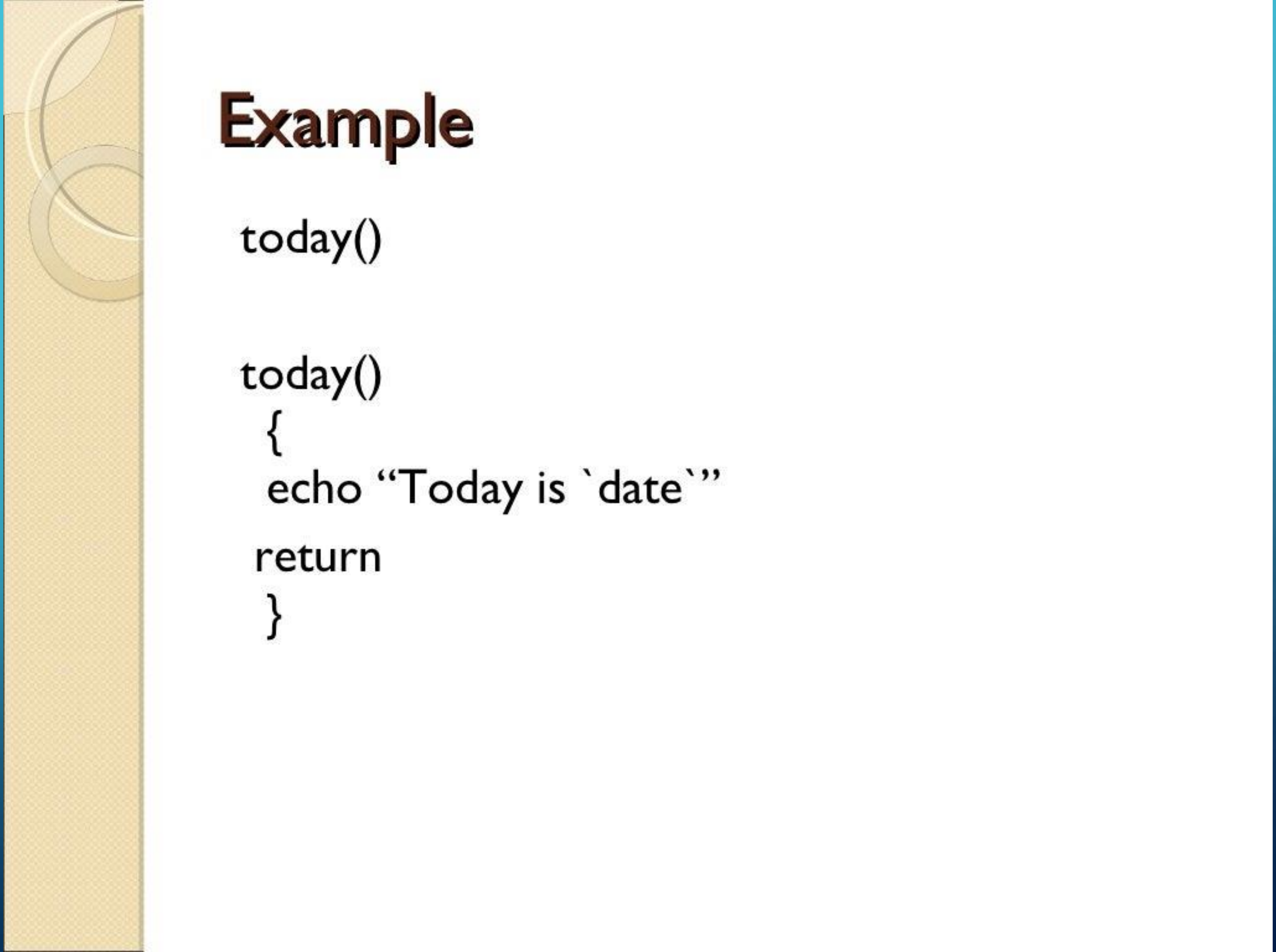
```
 read var
  case $var in
   1) echo “One”;;
   2) echo “Two”;;
   3) echo “Three”;;
   4) echo “Four”;;
   *) echo "Sorry, it is bigger than Four";;
esac
```



Functions

- Function is series of instruction/commands.
- Function performs particular activity in shell.
- *Syntax:*

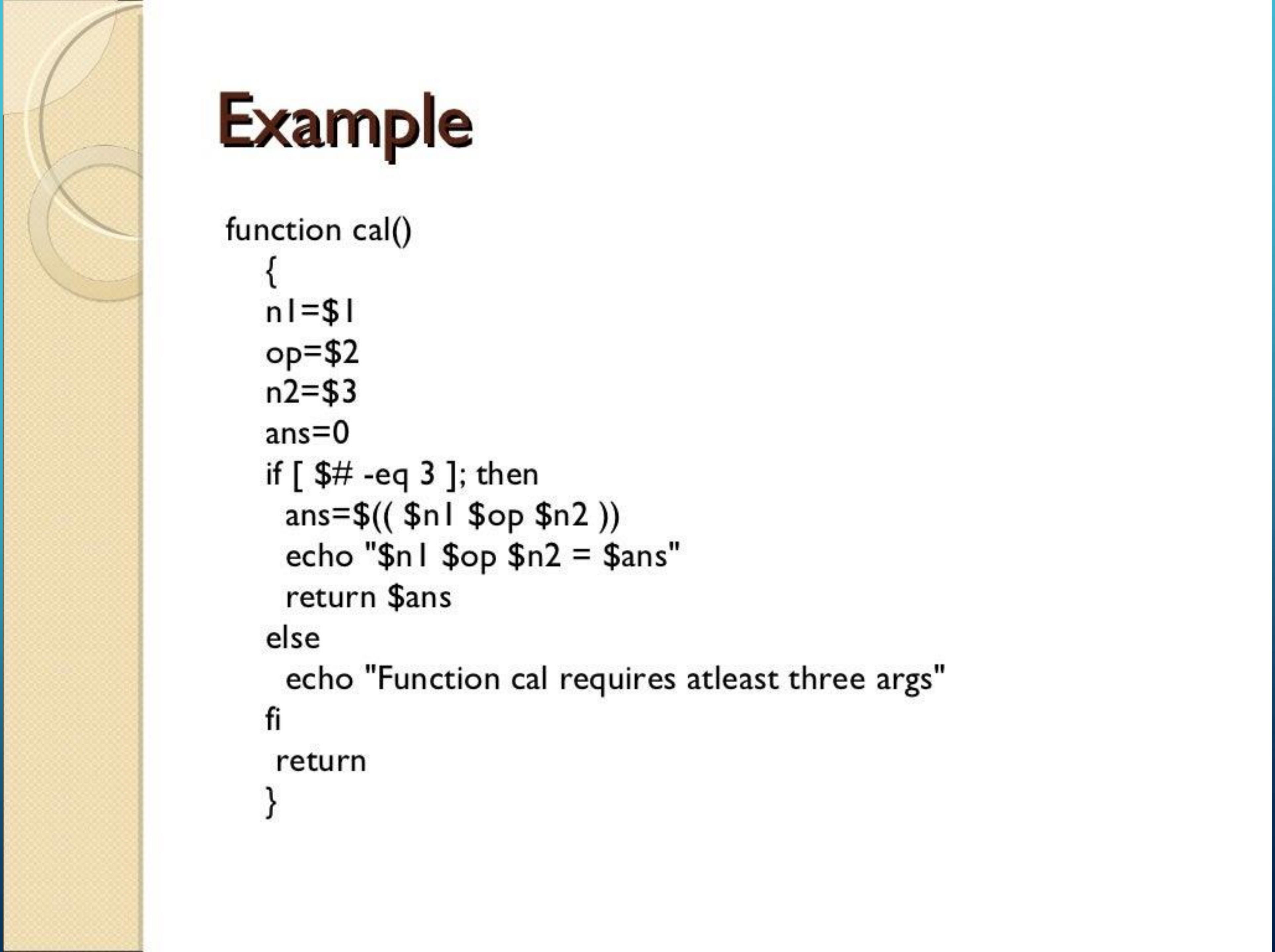
```
function-name ( )
{
   Function body
}
```



Example

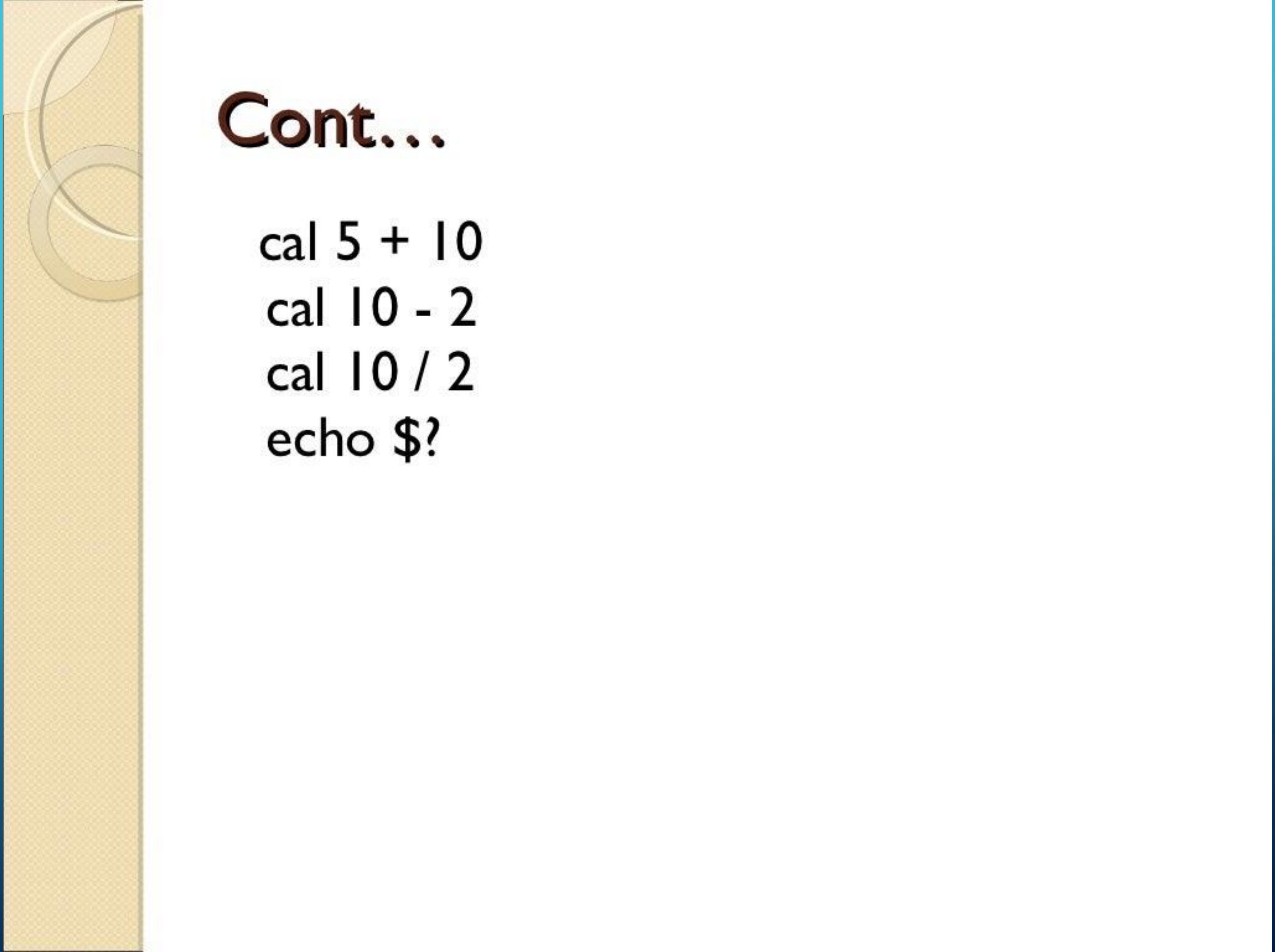
```
today()


today()
  {
  echo “Today is `date`”
 return
  }
```



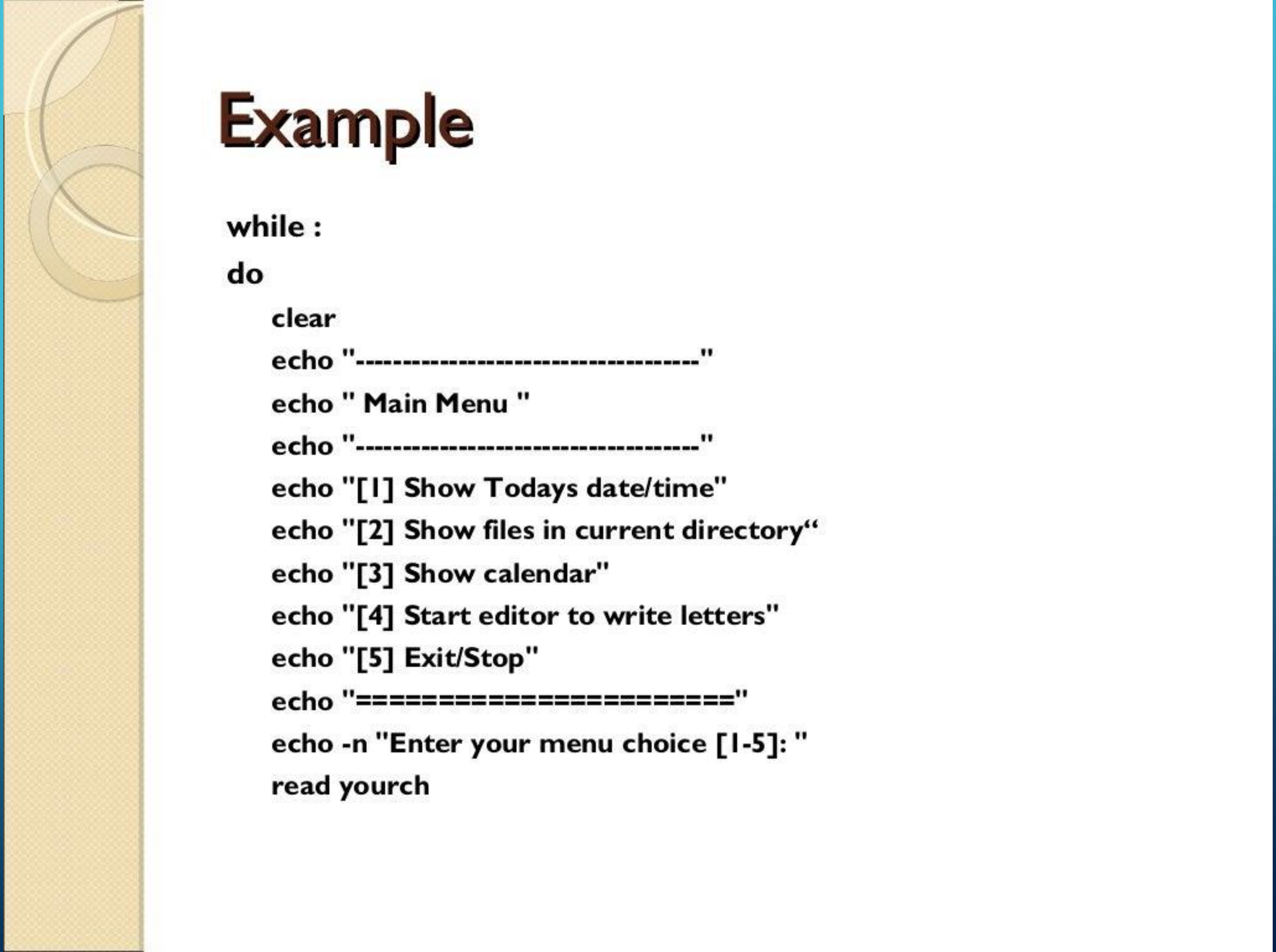
Example

```
function cal()
    {
    n1=$1
    op=$2
    n2=$3
    ans=0
    if [ $# -eq 3 ]; then
      ans=$(( $n1 $op $n2 ))
      echo "$n1 $op $n2 = $ans"
      return $ans
    else
      echo "Function cal requires atleast three args"
    fi
     return
    }
```



Cont…

```
cal 5 + 10
cal 10 - 2
cal 10 / 2
echo $?
```



Example

```
while :
do
     clear
     echo "----------------------------------------"
     echo " Main Menu "
     echo "----------------------------------------"
     echo "[1] Show Todays date/time"
     echo "[2] Show files in current directory“
     echo "[3] Show calendar"
     echo "[4] Start editor to write letters"
     echo "[5] Exit/Stop"
     echo "======================="
     echo -n "Enter your menu choice [1-5]: "
     read yourch
```

Cont…

```
case $yourch in
     1) echo "Today is `date` , press a key. . ." ; read ;;
     2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. . ." ; read ;;
     3) cal ; echo "Press a key. . ." ; read ;;
     4) vi ;;
     5) exit 0 ;;
     *) echo "Opps!!! Please select choice 1,2,3,4, or 5";
     echo "Press a key. . ." ; read ;;
esca
done
```

Class work#2

--------menu---------

WEEK: 12
SHELL PROGRAMMING 6

Working with Files

- "On a UNIX system, everything is a file; if something is not a file, it is a process."
- *Directories*: files that are lists of other files.
- *Special files*: the mechanism used for input and output. Most special files are in /dev, we will discuss them later.

- *Links*: a system to make a file or directory visible in multiple parts of the system's file tree. We will talk about links in detail.
- *(Domain) sockets*: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.

- *Named pipes*: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
S	Socket
P	Named pipe
C	character (unbuffered) device file special
b	block (buffered) device file special

WEEK: 13
UNIX COMMANDS

UNIX Commands

UNIX has a large family of powerful commands.

Some major commands are:

date, mkdir, cd, pwd, rmdir, ls, cp, mv, rm, cat, cal, who, tput clear, man, passwd, path, ps, chmod, chown, more etc.

→date: displays the date and time.

```
ex:        $ date          [enter]
           wed march 31   16: 22:40 IST 2005
```

→cal: displays the calendar of any specific month or year.

```
ex:        $ cal 2004      [enter]
```

→who: displays the list of users that are currently logged into system.

```
ex:        $ who           [enter]
           Deepanshu   console    may 9    09:31
           John    pts/4      may 9    09:31
```

→cat: creates a file.

```
ex:        $ cat>filename          [enter]
```

→ls: displays the list of files and directories.

ex:
```
$ date  [enter]
wed march 31   16: 22:40 IST 2005
```

→mkdir: creates directories.

ex:
```
$ mkdir  abc    [enter]
```

→rmdir: removes directories.

ex:
```
$ rmdir  abc    [enter]
```

→mv: perform two functions.

1. Renames a file or directory.
2. Moves a group of files to different directory.

ex:
```
$ mv c1 b1       [enter]
renames c1 to b1
$ mv c1 c2 abc [enter]
moves c1, c2 to directory abc
```

→cp: for copying one or more files.

```
ex:        $ cp sourcefilename destinationfilename    [enter]
```

→rm: deletes one or more files.

```
ex:        $ rm c1 c2 c3   [enter]
```

→cat: display contents of file on screen.

```
ex:        $ cat filename          [enter]
```

→tput clear: clears the screen.

```
ex:        $ tput clear    [enter]
```

→pwd: display path of your present working directory.

```
ex:        $ pwd   [enter]
           /home/sharma
```

→cd: change the current working directory to another.

```
ex:        $ cd ram        [enter]
           $pwd
           /home/sharma/ram
```

WEEK: 14
UNIX COMMANDS

General Purpose Utilities

Unix provides various general purpose utilities having diverse functionality . Some of them are:

cal, date, echo, bc, printf, passwd, man, whatis, mailx etc.

→bc: is the text based calculator.

```
    ex:       $ bc    [enter]
              12+5    [enter]
              17      [ctrl+d]
```

→echo: is used to print messages.

```
    ex:       $ echo Deepanshu          [enter]
```

→printf: is an alternative to echo.

```
    ex:       $ printf "Deepanshu"      [enter]
```

→mailx: is the universal mailer to send or receive mails.

```
Ex.     $mailx  Deepanshu        [enter]
        subject: Attention
        call me now          [ctrl-d]
        Will send the mail to Deepanshu.
```

→passwd: for changing password.

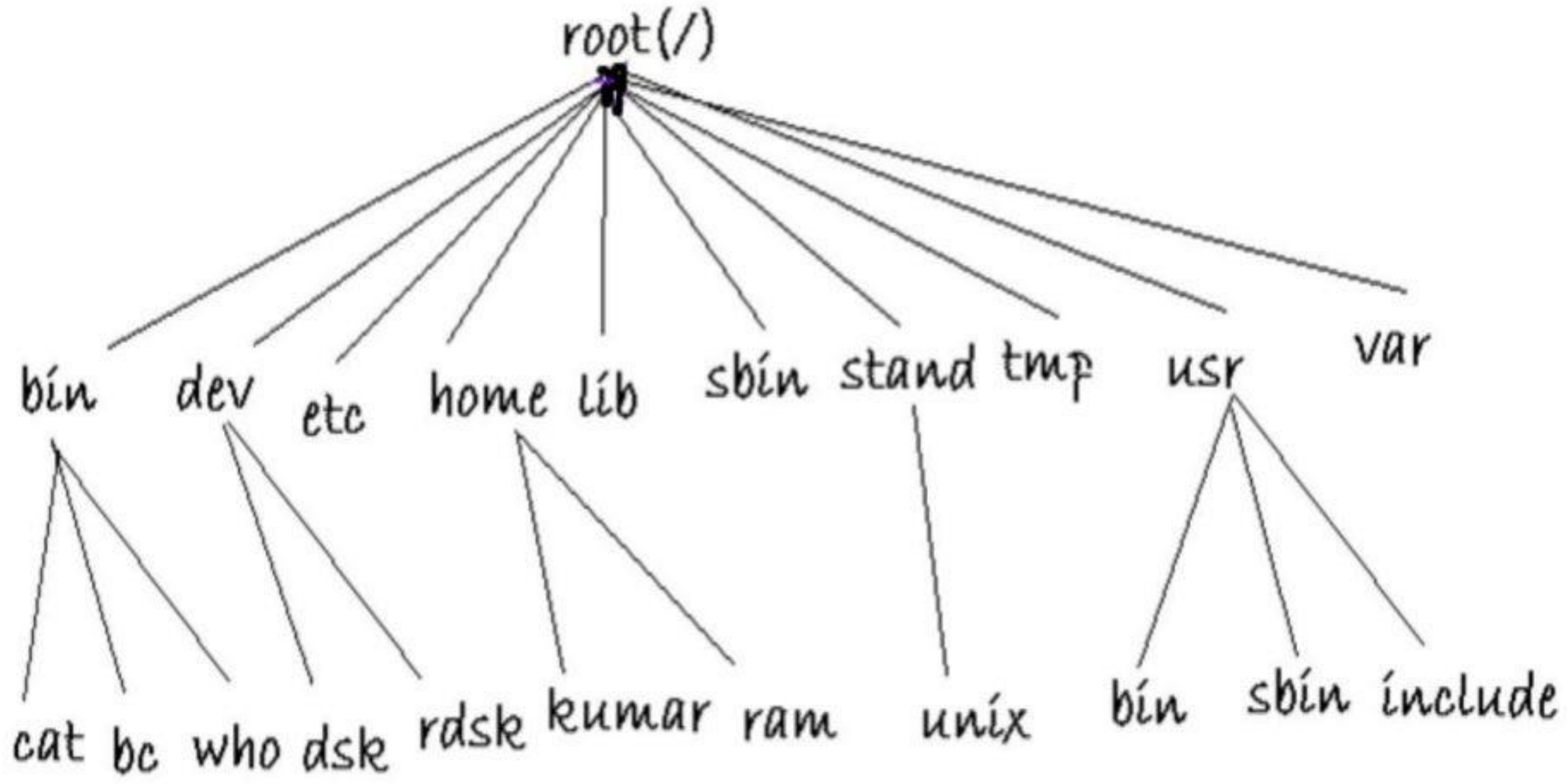
```
ex:      $ passwd      [enter]
         (Current) UNIX password: *****   (12345)
         New password: ***** (56789)
         Re-enter: ***** (56789)
   passwd(SYSTEM): password successfully changed.
```

→man: is the help facility.

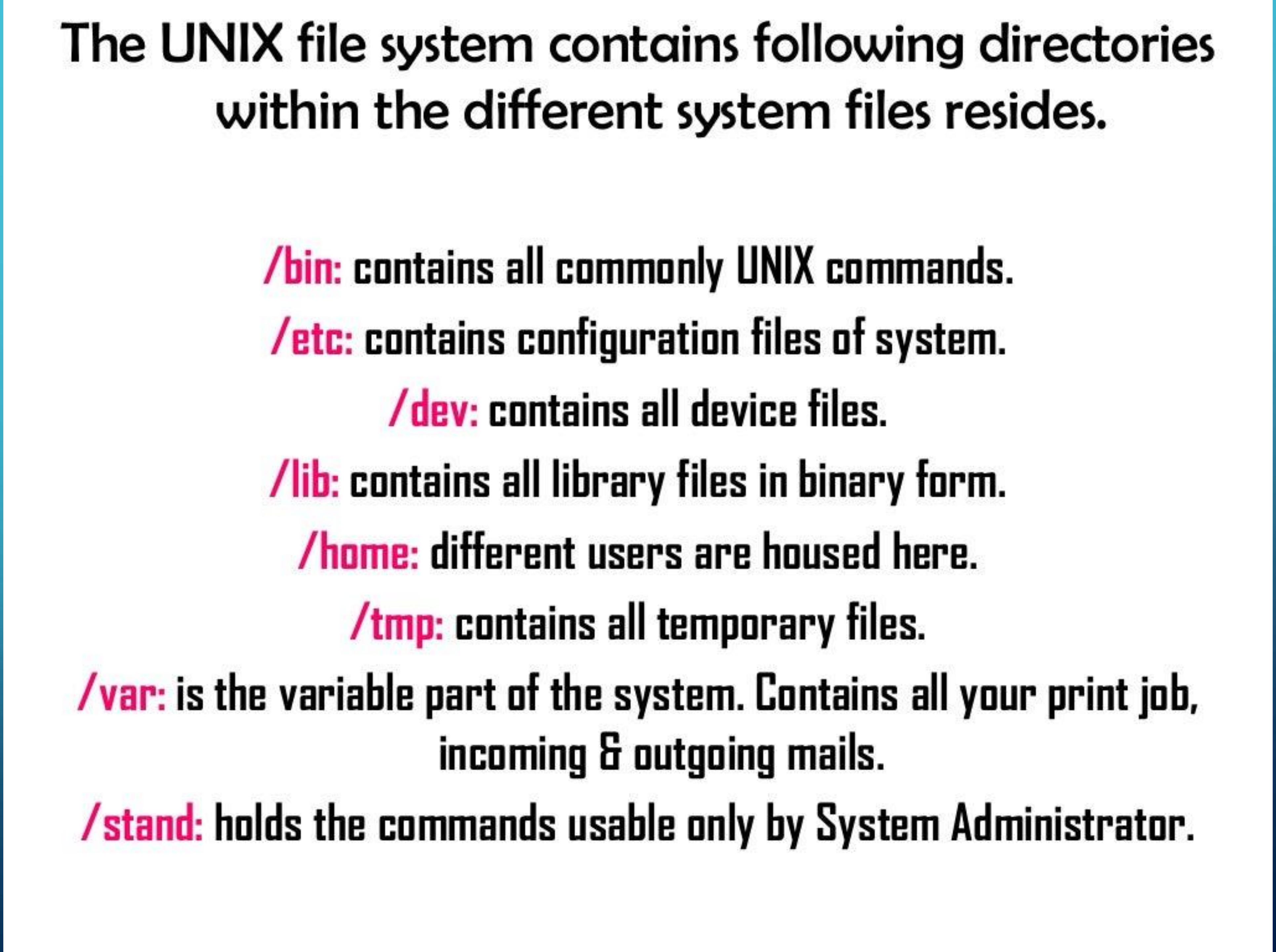
```
ex:      $ man bc     [enter]
```

Will give a page having complete documentation about bc command including options, descriptions, usage etc.

UNIX File System



The UNIX File System Tree



The UNIX file system contains following directories within the different system files resides.

/bin: contains all commonly UNIX commands.

/etc: contains configuration files of system.

/dev: contains all device files.

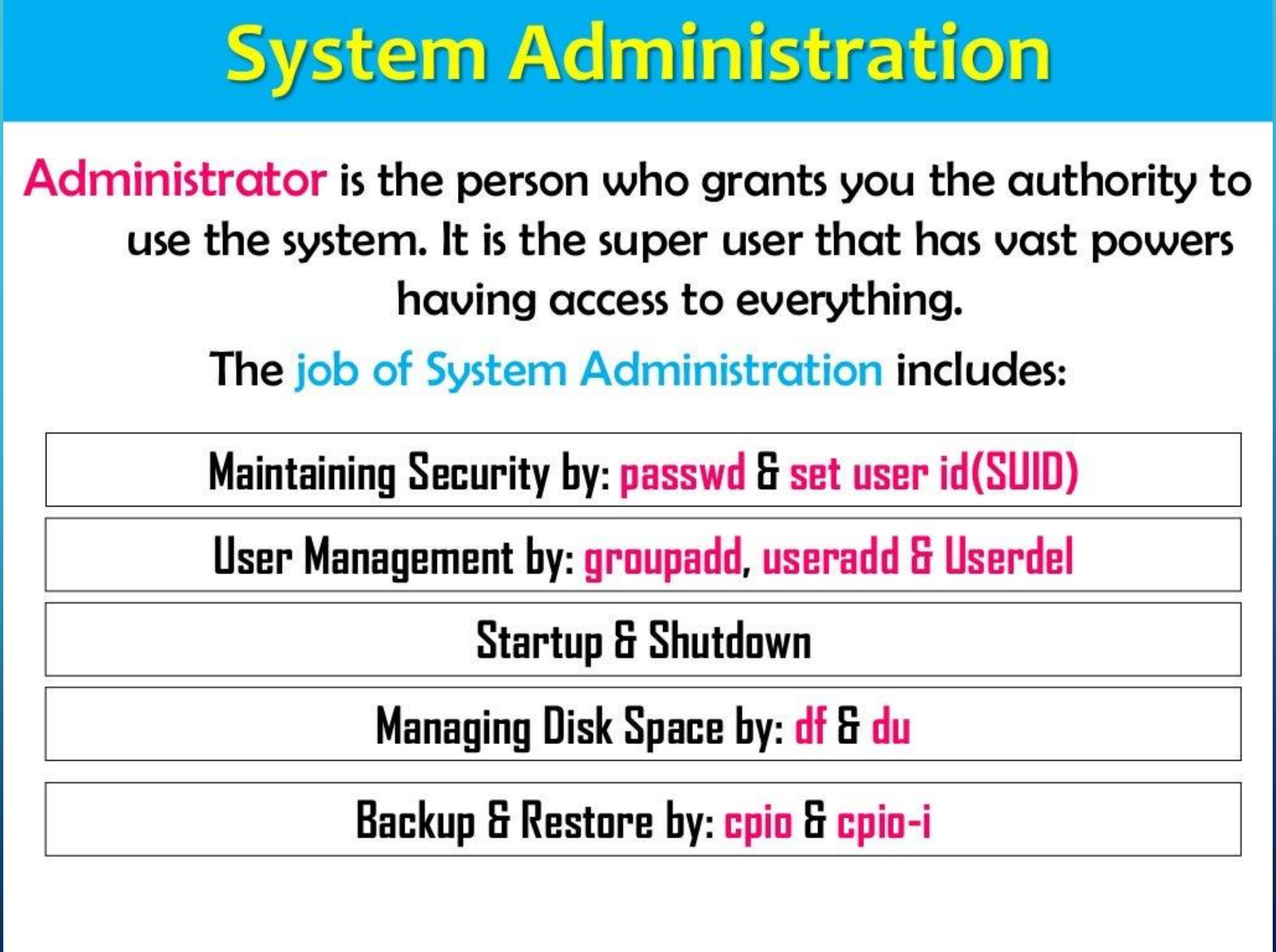
/lib: contains all library files in binary form.

/home: different users are housed here.

/tmp: contains all temporary files.

/var: is the variable part of the system. Contains all your print job, incoming & outgoing mails.

/stand: holds the commands usable only by System Administrator.



System Administration

Administrator is the person who grants you the authority to use the system. It is the super user that has vast powers having access to everything.

The job of System Administration includes:

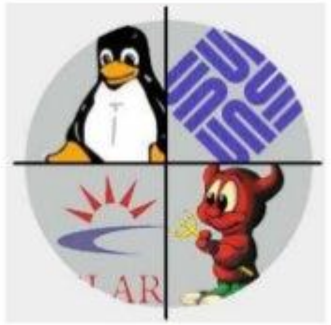
Maintaining Security by: passwd & set user id(SUID)
User Management by: groupadd, useradd & Userdel
Startup & Shutdown
Managing Disk Space by: df & du
Backup & Restore by: cpio & cpio-i

Linux's Root in UNIX

Linux is a full UNIX clone created by Linus Torvalds when when he was a student at the university of Helsinki in 1991.It is the free operating system based on UNIX. It is written in C language.

Primary Advantages of Linux:

1. Its initial price is free.
2. Help is always available on internet.
3. It is portable to any Hardware platform.
4. It is scalable and secure.



Popular Linux Distribution :

Red Hat Enterprise Edition, Fedora Core, Debian, SuSE Linux