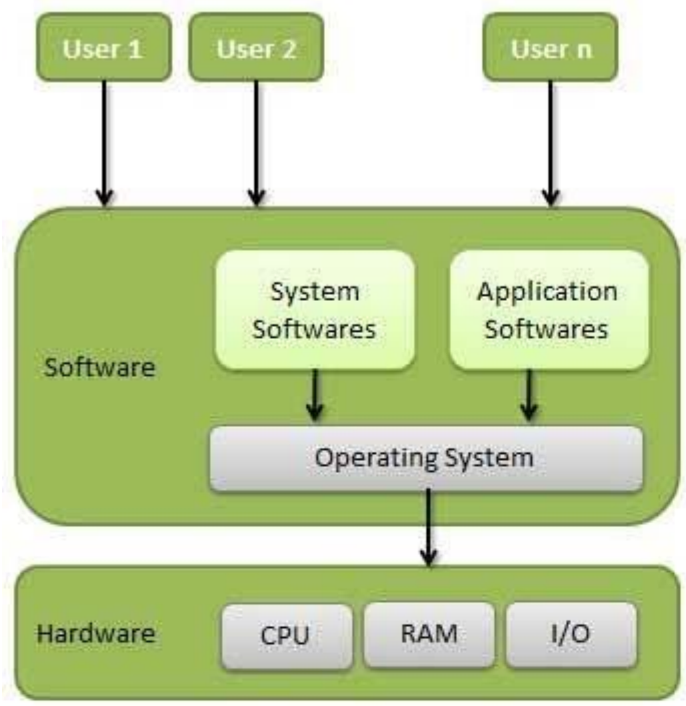
FUNCTION OF OPERATING SYSTEM

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

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.



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

FUNCTION OF OPERATING SYSTEM

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

FUNCTION OF OPERATING SYSTEM

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Other Important Activities

Following are some of the important activities that an Operating System performs −

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other softwares and users** − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

TYPES OF OPERATING SYSTEM

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows −

- Provides the advantage of quick response.
- Avoids duplication of software.

- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.

- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

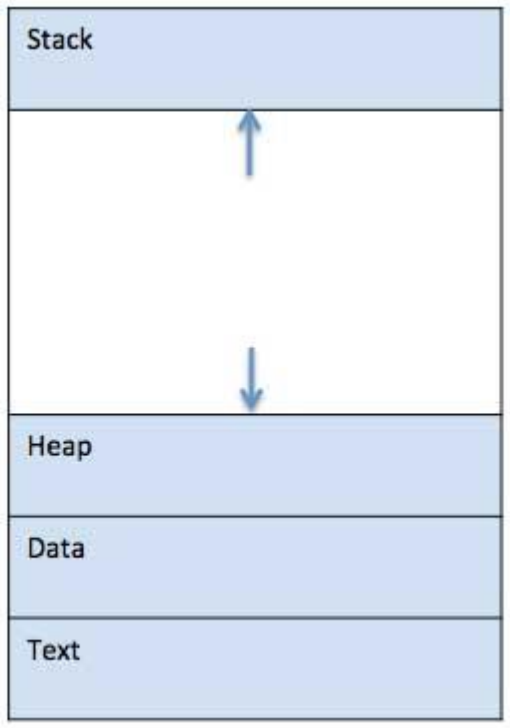
Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap**

	This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include <stdio.h>

int main() {
   printf("Hello, World! \n");
   return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

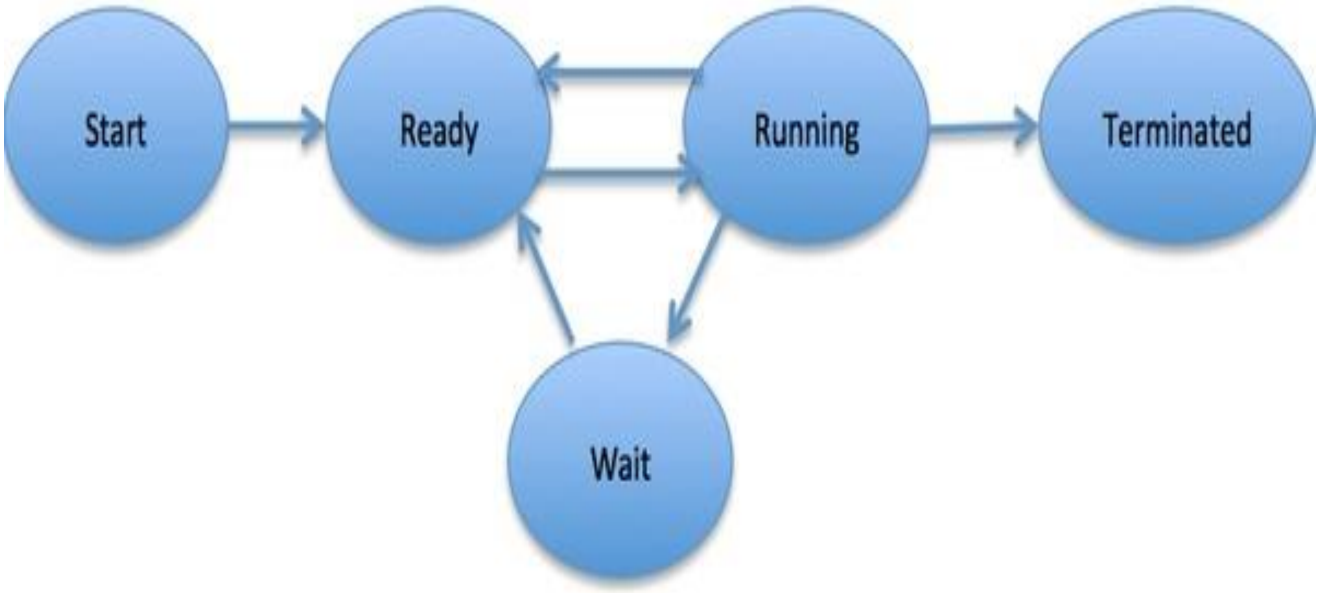
Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start**

	This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



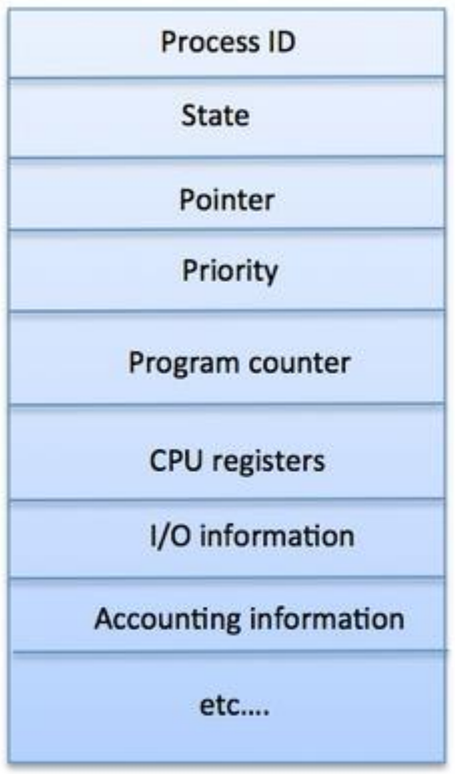
Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table −

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information**

	Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

PROCESS SCHEDULING:

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

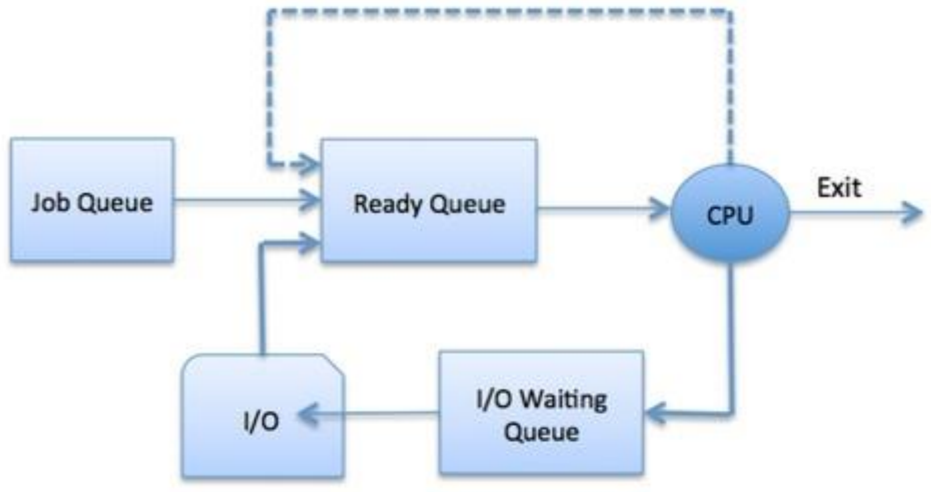
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- **Long-Term Scheduler**
- **Short-Term Scheduler**
- **Medium-Term Scheduler**

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

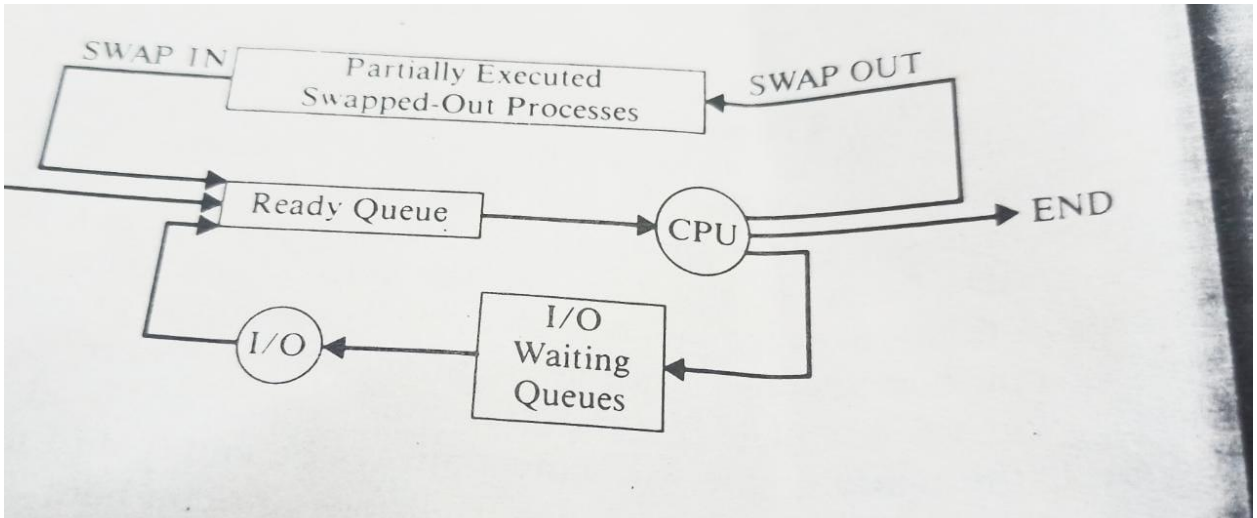
It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.



3 Adding medium-term scheduling to the queueing diagram

Comparison among Scheduler

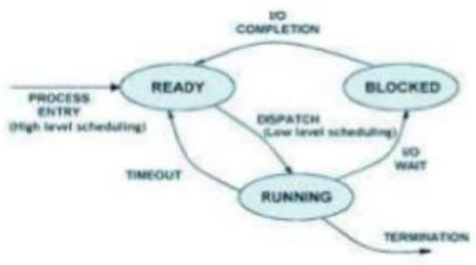
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Dispatcher

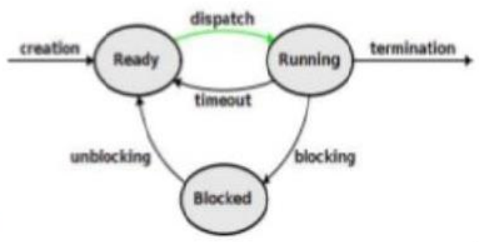
The dispatcher is the module that gives control of the CPU to the process selected by the scheduler. This function involves:

- Switching context.
- Switching to user mode.
- Jumping to the proper location in the newly loaded program.

The dispatcher needs to be as fast as possible, as it is run on every context switch. Dispatch latency is the amount of time required for the scheduler to stop one process and start another.



I/O COMPLETION
READY
BLOCKED
PROCESS ENTRY (High level scheduling)
DISPATCH (Low level scheduling)
I/O WAIT
TIMEOUT
RUNNING
TERMINATION



Dispatcher

When the processes are in the ready state, then the CPU applies some process scheduling algorithm and choose one process from a list of processes that will be executed at a particular instant of time. This is done by a scheduler i.e. selecting one process from a number of processes is done by a scheduler.

Now, the selected process has to be transferred from the current state to the desired or scheduled state. So, it is the duty of the dispatcher to dispatch or transfer a process from one state to another. A dispatcher is responsible for context switching and switching to user mode.

For example, if we have three processes P1, P2, and P3 in the ready state. The arrival time of all these processes is T0, T1, and T2 respectively. If we are using the First Come First Serve approach, then the scheduler will first select the process P1 and the dispatcher will transfer the process P1 from the ready state to the running state. After completion of the execution of the process P1, the scheduler will then select the process P2 and the dispatcher will transfer the process P2 from ready to running state and so on.

Difference between Dispatcher and Scheduler

Till now, we are familiar with the concept of dispatcher and scheduler. Now in this section of the blog, we will see the difference between a dispatcher and a scheduler.

- The scheduler selects a process from a list of processes by applying some process scheduling algorithm. On the other hand, the dispatcher transfers the process selected by the short-term scheduler from one state to another.

- The scheduler works independently, while the dispatcher has to be dependent on the scheduler i.e. the dispatcher transfers only those processes that are selected by the scheduler.
- For selecting a process, the scheduler uses some process scheduling algorithm like FCFS, Round-Robin, SJF, etc. But the dispatcher doesn't use any kind of scheduling algorithms.
- The only duty of a scheduler is to select a process from a list of processes. But apart from transferring a process from one state to another, the dispatcher can also be used for switching to user mode. Also, the dispatcher can be used to jump to a proper location when the process is restarted.

- In the Operating System, there are cases when you have to bring back the process that is in the running state to some other state like ready state or wait/block state.
- If the running process wants to perform some I/O operation, then you have to remove the process from the running state and then put the process in the I/O queue.
- Sometimes, the process might be using a round-robin scheduling algorithm where after every fixed time quantum, the process has to come back to the ready state from the running state.
- So, these process switchings are done with the help of Context Switching. In this blog, we will learn about the concept of Context Switching in the Operating System and we will also learn about the advantages and disadvantages of Context Switching. So, let's get started.

What is Context Switching?

A context switching is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.

It is one of the essential features of the multitasking operating system. The processes are switched so fastly that it gives an illusion to the user that all the processes are being executed at the same time.

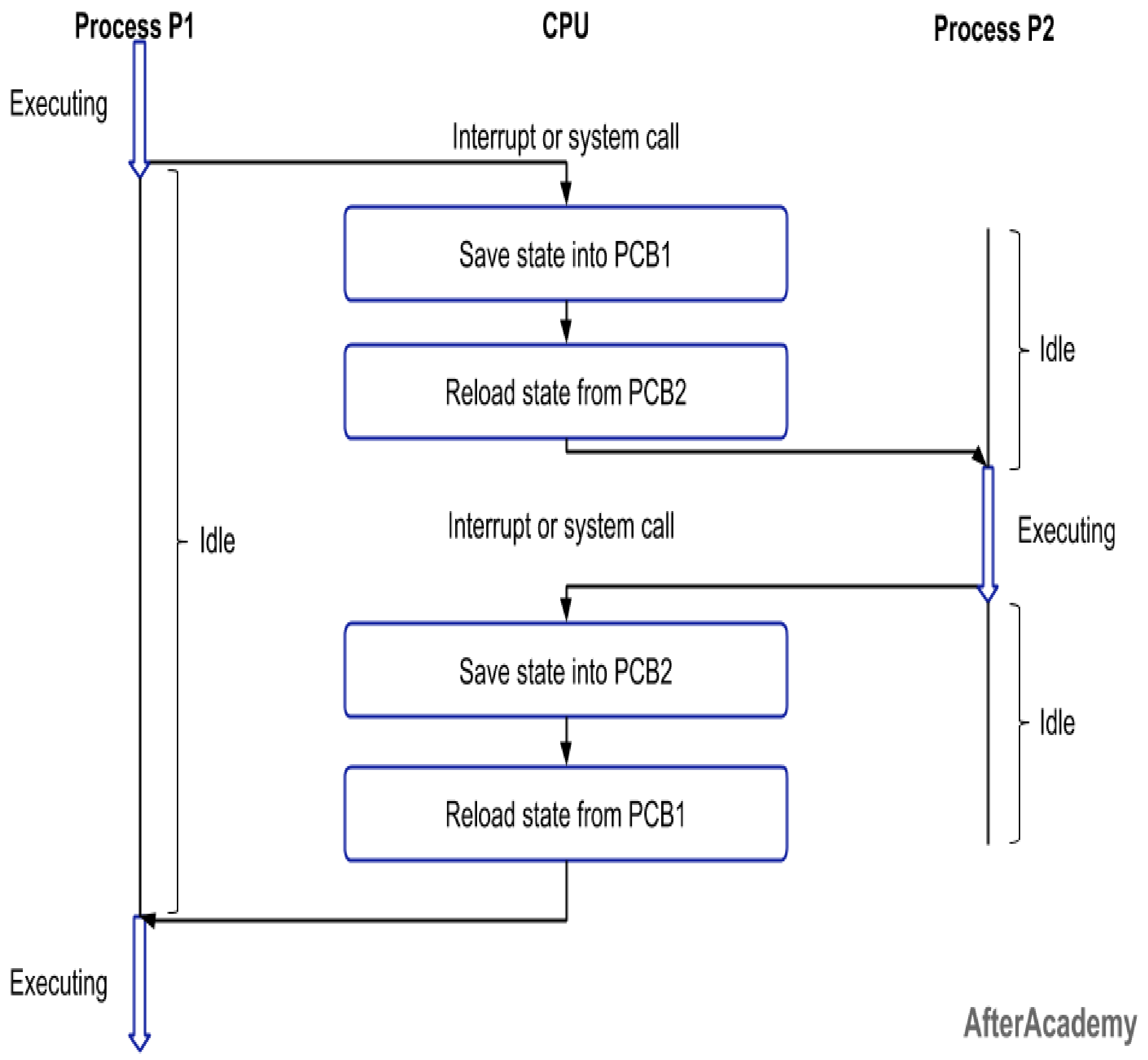
But the context switching process involved a number of steps that need to be followed. You can't directly switch a process from the running state to the ready state. You have to save the context of that process. If you are not saving the context of any process P then after some time, when the process P comes in the CPU for execution again, then the process will start executing from starting. But in reality, it should continue from that point where it left the CPU in its previous execution. So, the context of the process should be saved before putting any other process in the running state.

A context is the contents of a CPU's registers and program counter at any point in time. Context switching can happen due to the following reasons:

- When a process of high priority comes in the ready state. In this case, the execution of the running process should be stopped and the higher priority process should be given the CPU for execution.
- When an interruption occurs then the process in the running state should be stopped and the CPU should handle the interrupt before doing something else.
- When a transition between the user mode and kernel mode is required then you have to perform the context switching.

Steps involved in Context Switching

The process of context switching involves a number of steps. The following diagram depicts the process of context switching between the two processes P1 and P2.



In the above figure, you can see that initially, the process P1 is in the running state and the process P2 is in the ready state. Now, when some interruption occurs then you have to switch the process P1 from running to the ready state after saving the context and the process P2 from ready to running state. The following steps will be performed:

1. Firstly, the context of the process P1 i.e. the process present in the running state will be saved in the Process Control Block of process P1 i.e. PCB1.
2. Now, you have to move the PCB1 to the relevant queue i.e. ready queue, I/O queue, waiting queue, etc.
3. From the ready state, select the new process that is to be executed i.e. the process P2.
4. Now, update the Process Control Block of process P2 i.e. PCB2 by setting the process state to running. If the process P2 was earlier executed by the CPU, then you can get the position of last executed instruction so that you can resume the execution of P2.
5. Similarly, if you want to execute the process P1 again, then you have to follow the same steps as mentioned above(from step 1 to 4).

For context switching to happen, two processes are at least required in general, and in the case of the round-robin algorithm, you can perform context switching with the help of one process only.

The time involved in the context switching of one process by other is called the Context Switching Time.

Advantage of Context Switching

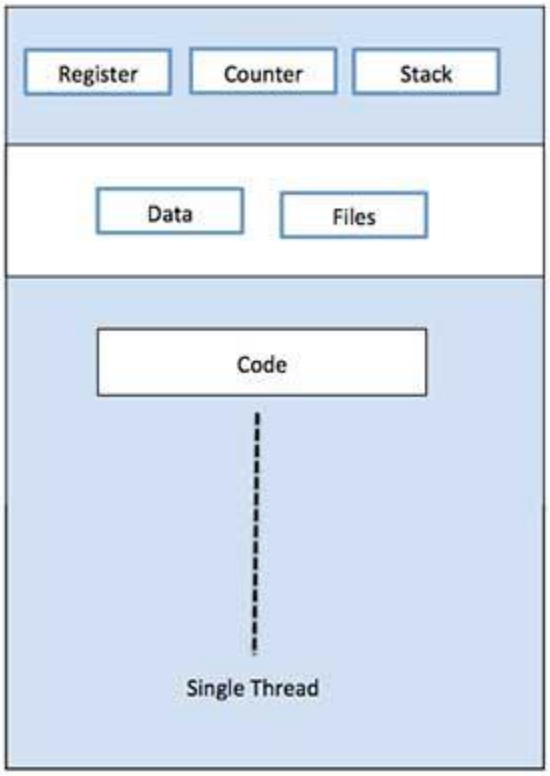
- Context switching is used to achieve multitasking i.e. multiprogramming with time-sharing.
- Multitasking gives an illusion to the users that more than one process are being executed at the same time.
- But in reality, only one task is being executed at a particular instant of time by a processor. Here, the context switching is so fast that the user feels that the CPU is executing more than one task at the same time.

The disadvantage of Context Switching

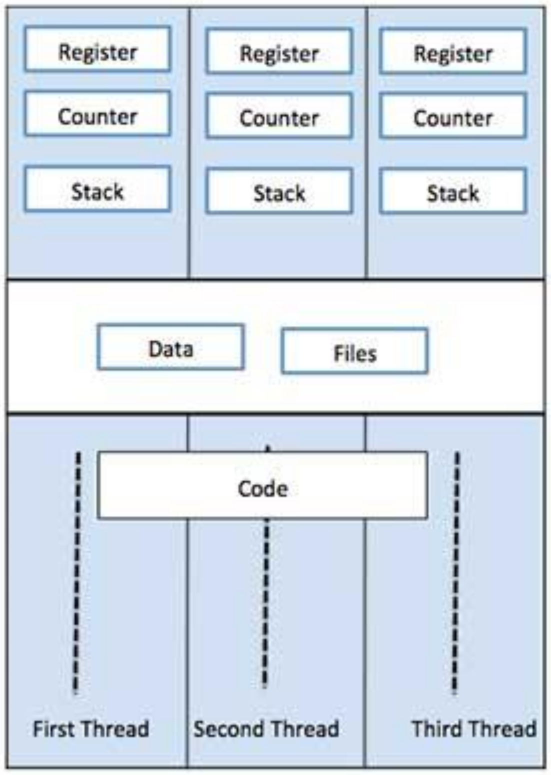
- The disadvantage of context switching is that it requires some time for context switching i.e. the context switching time.
- Time is required to save the context of one process that is in the running state and then getting the context of another process that is about to come in the running state.
- During that time, there is no useful work done by the CPU from the user perspective. So, context switching is pure overhead in this condition.

What is Thread?

- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes

		use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

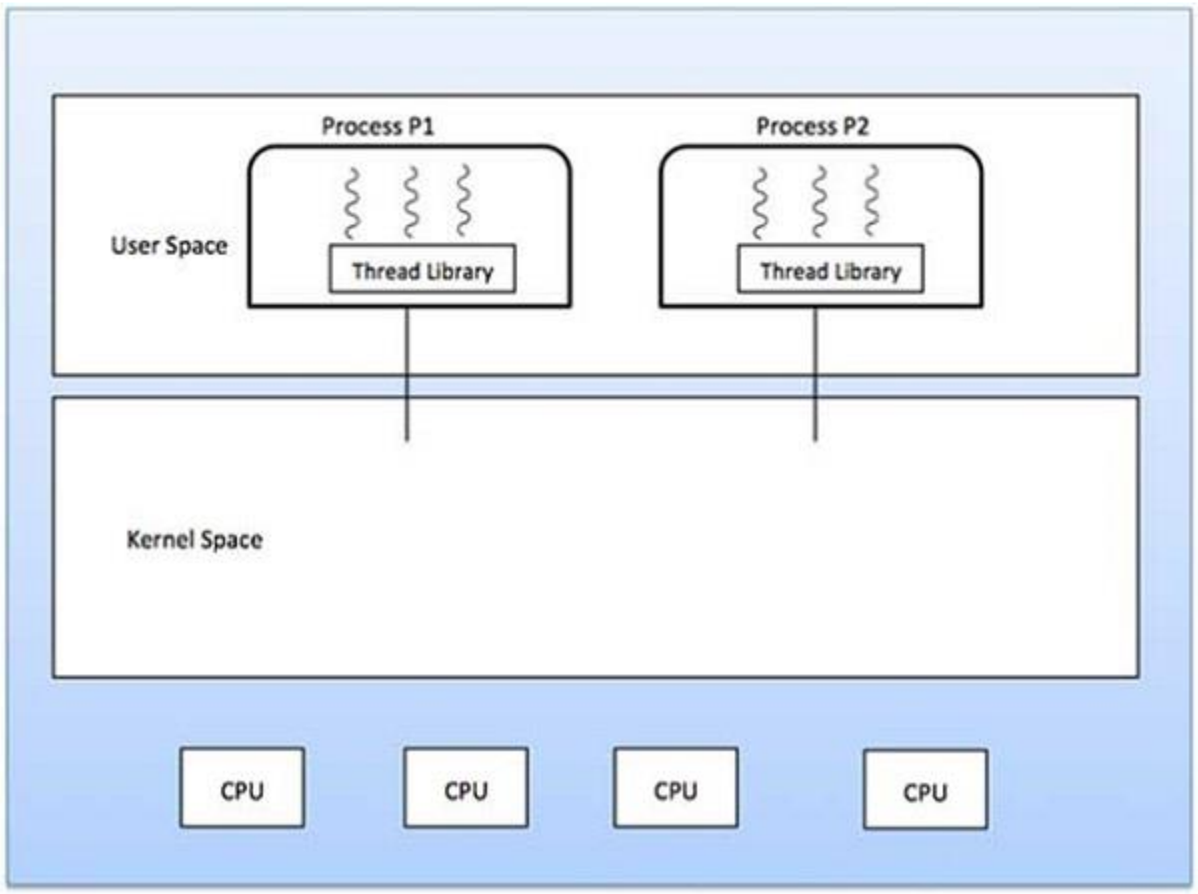
Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.
- Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

- ❖ In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- ❖ The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis.
- ❖ The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

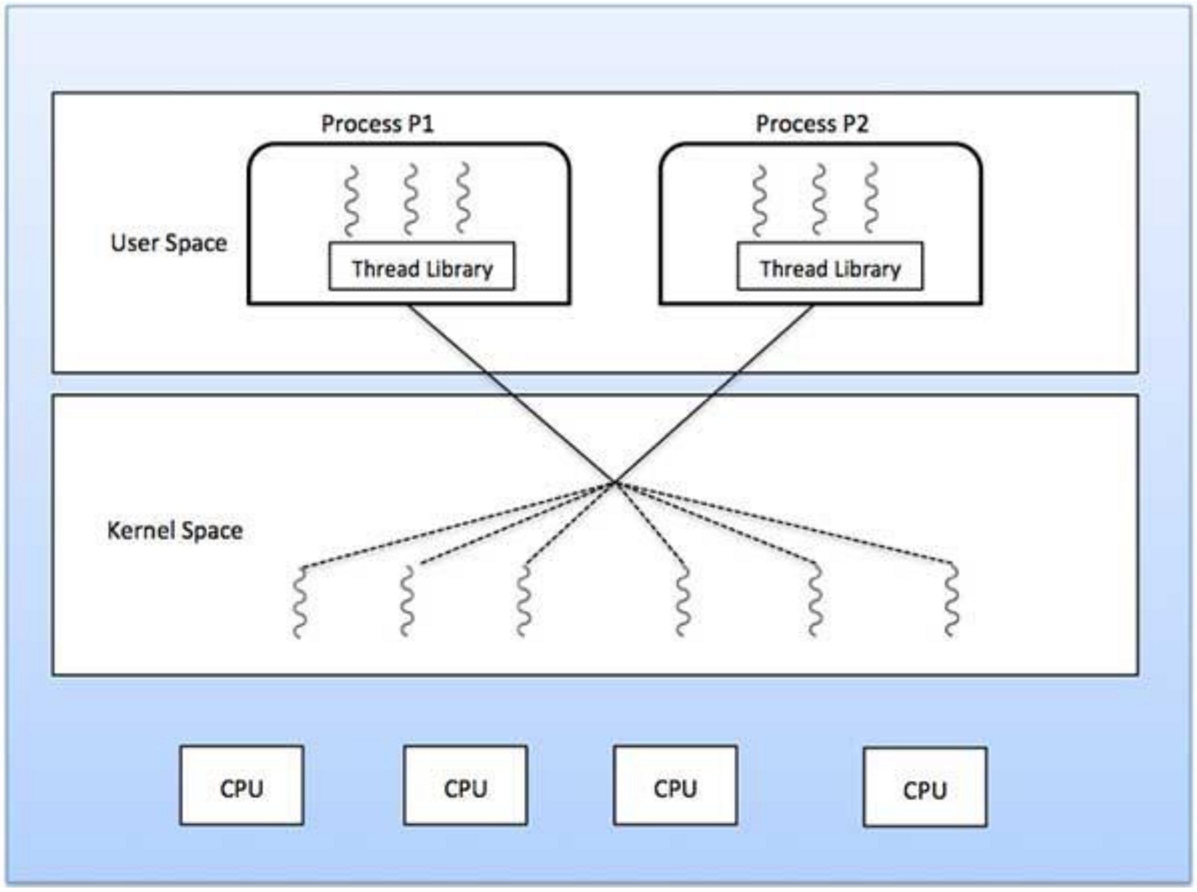
Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

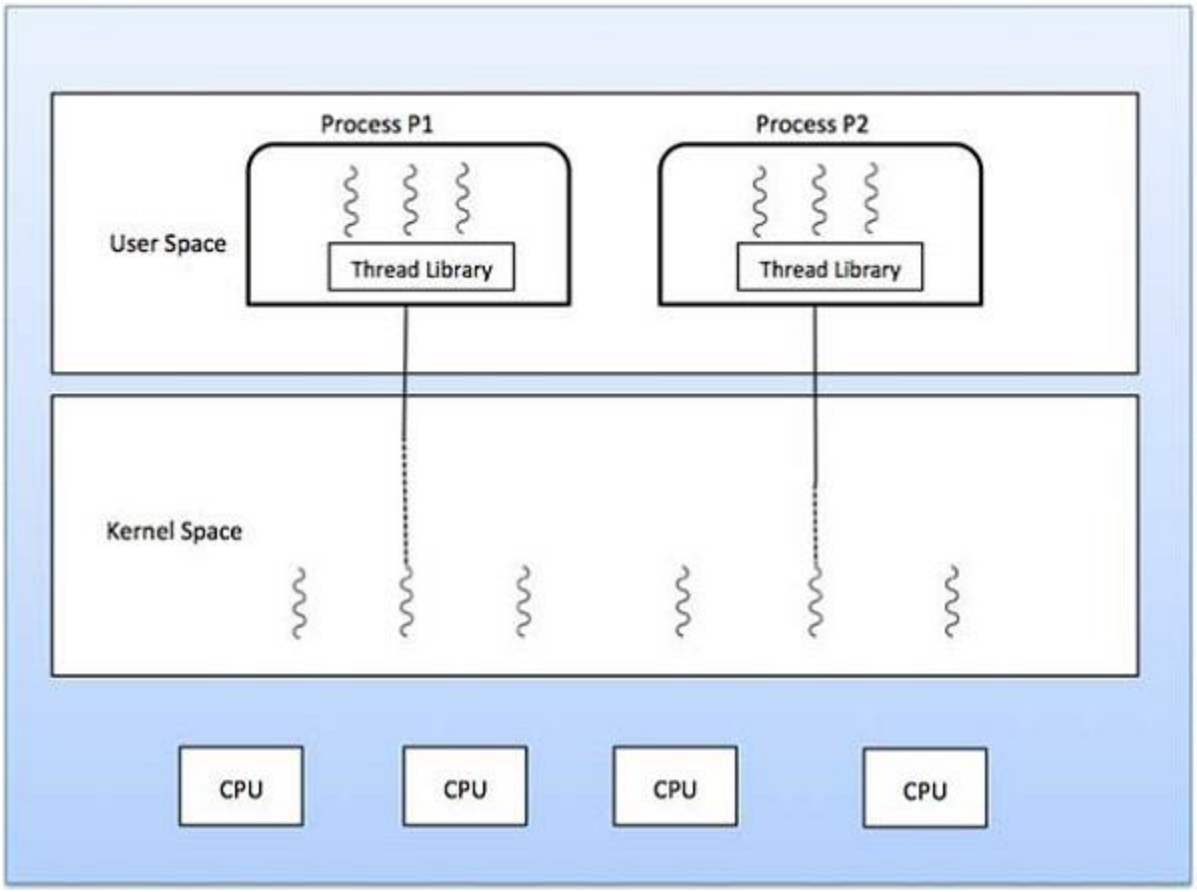
- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

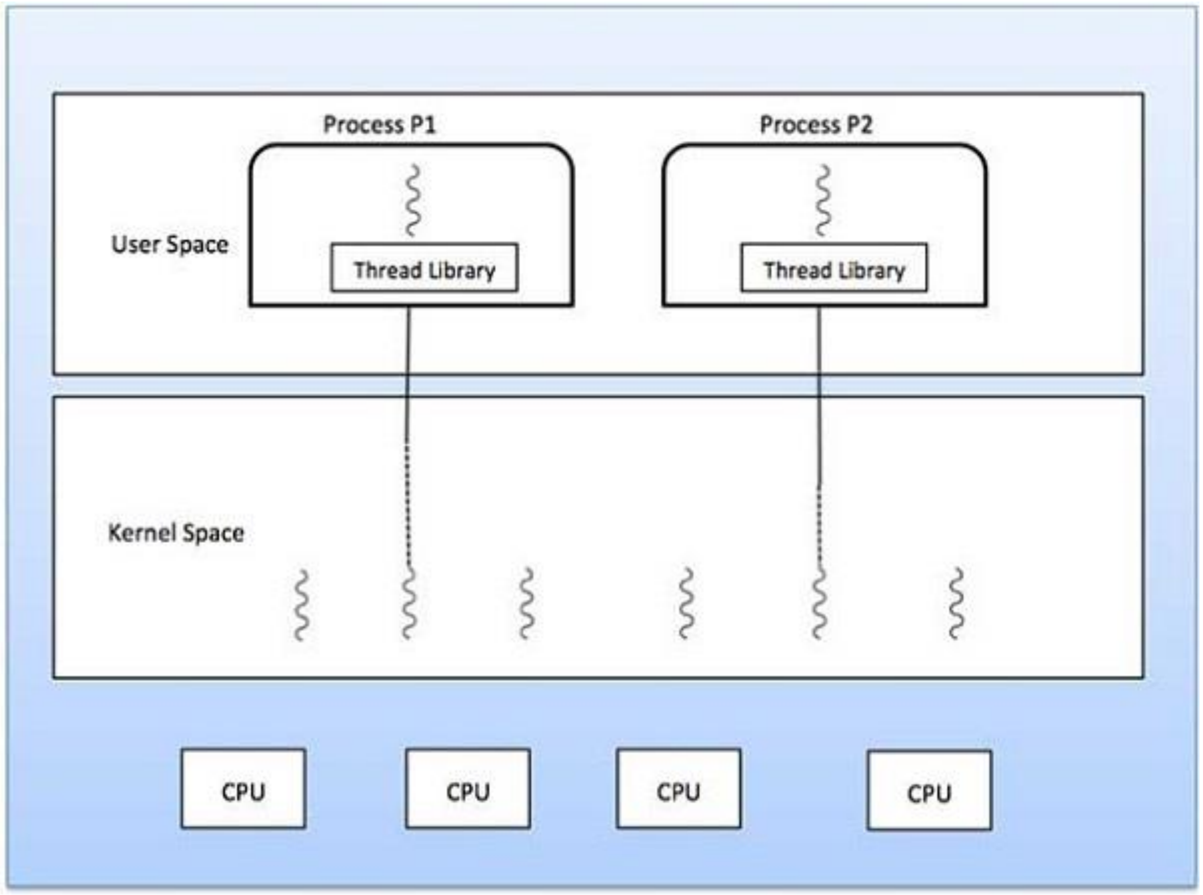
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Process Management

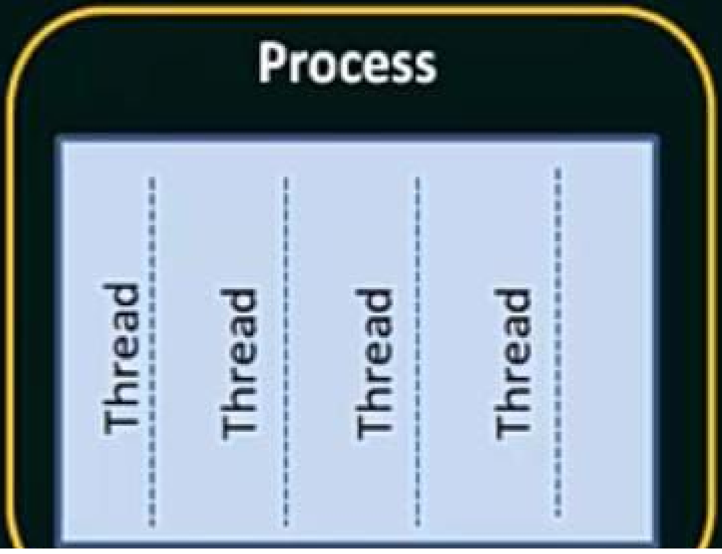
(Processes and Threads)

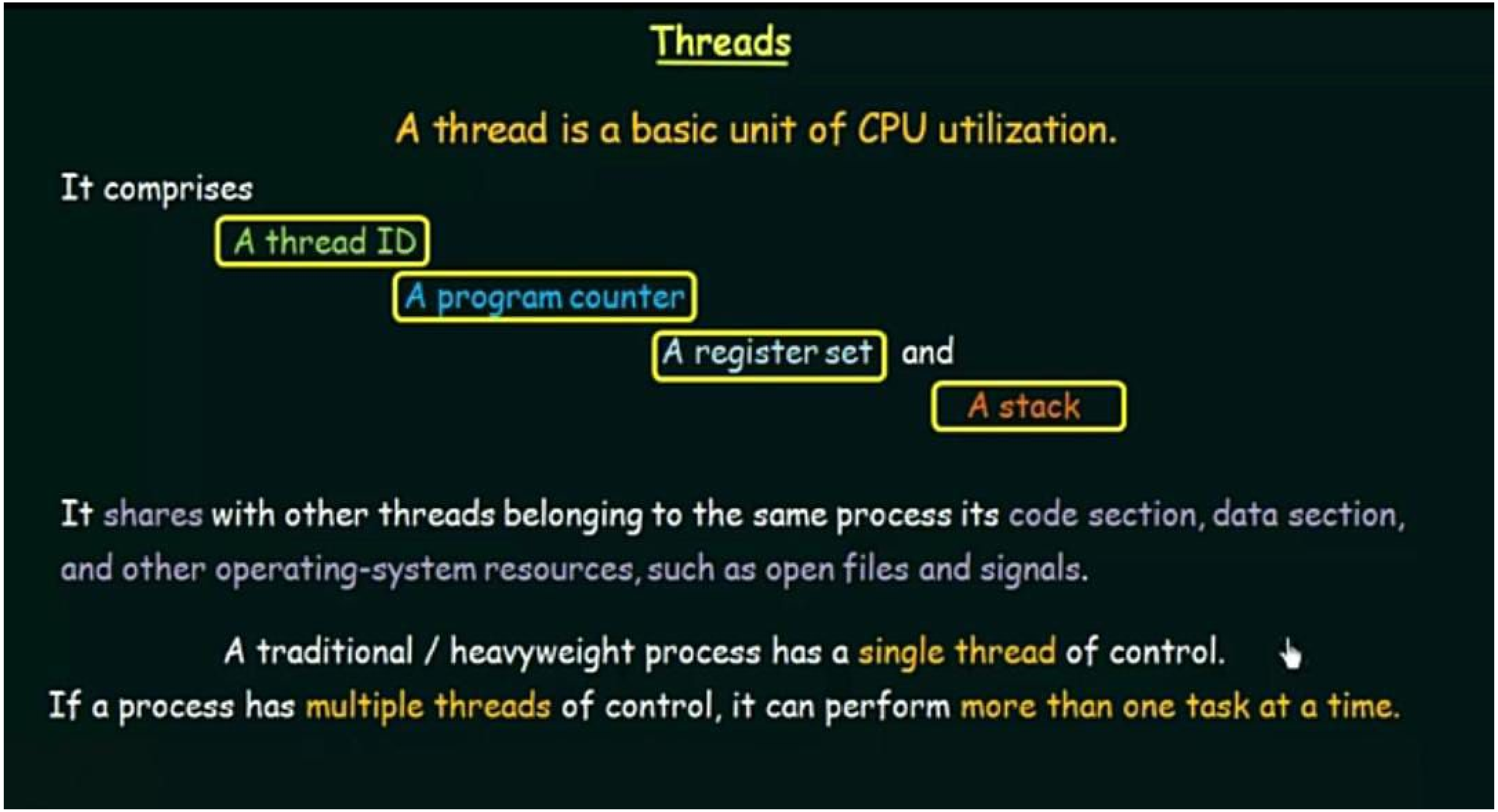
Process:

A process can be thought of as a program in execution.

Thread:

A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads.





Threads

A thread is a basic unit of CPU utilization.

It comprises

- A thread ID
- A program counter
- A register set and
- A stack

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

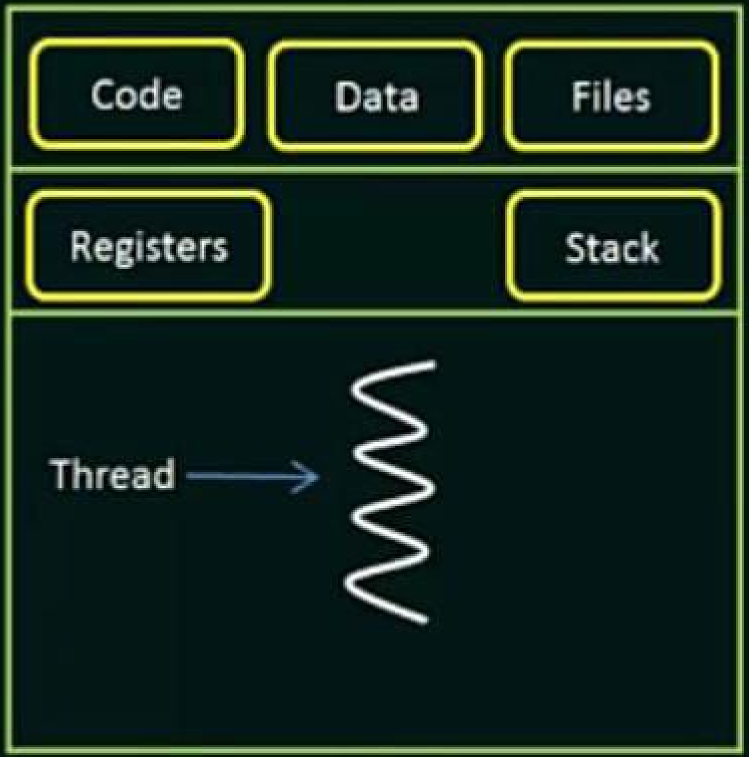
A traditional / heavyweight process has a single thread of control.

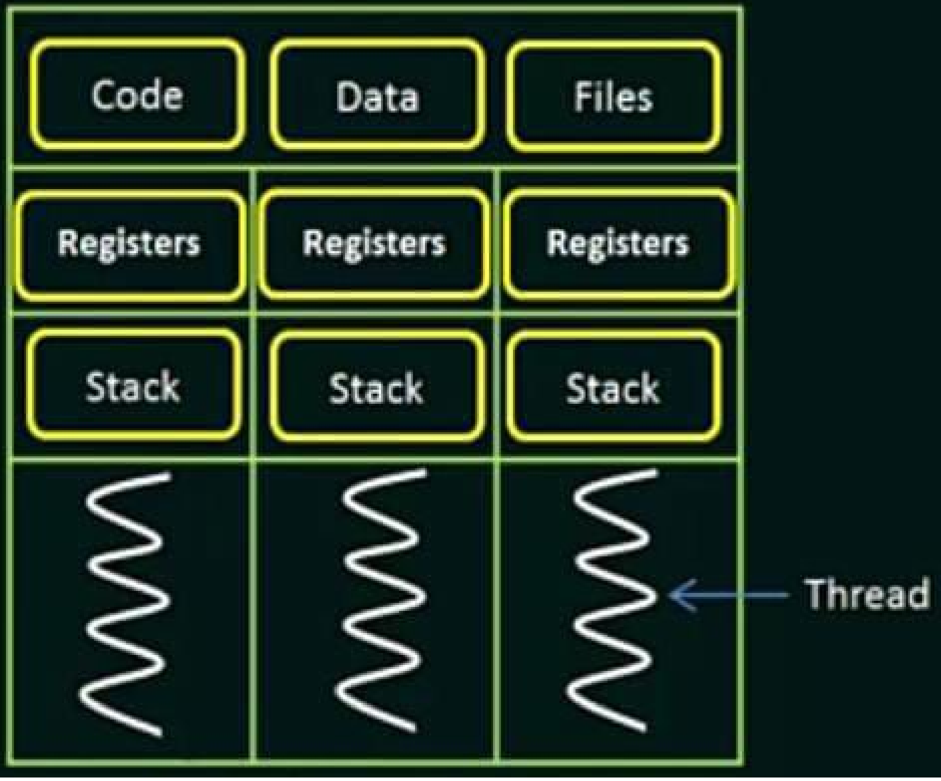
If a process has multiple threads of control, it can perform more than one task at a time.

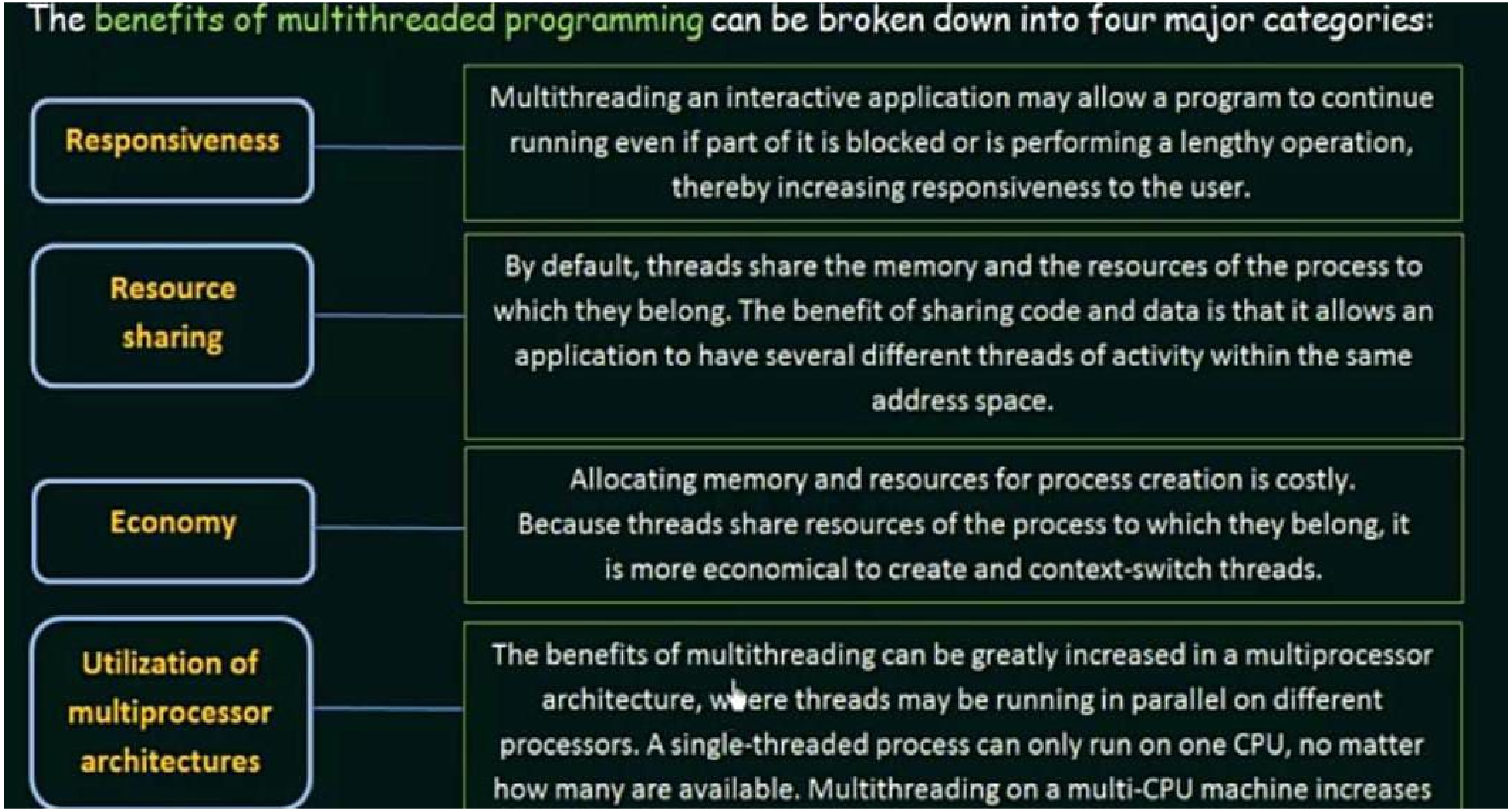
It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

A traditional / heavyweight process has a single thread of control.

If a process has multiple threads of control, it can perform more than one task at a time.

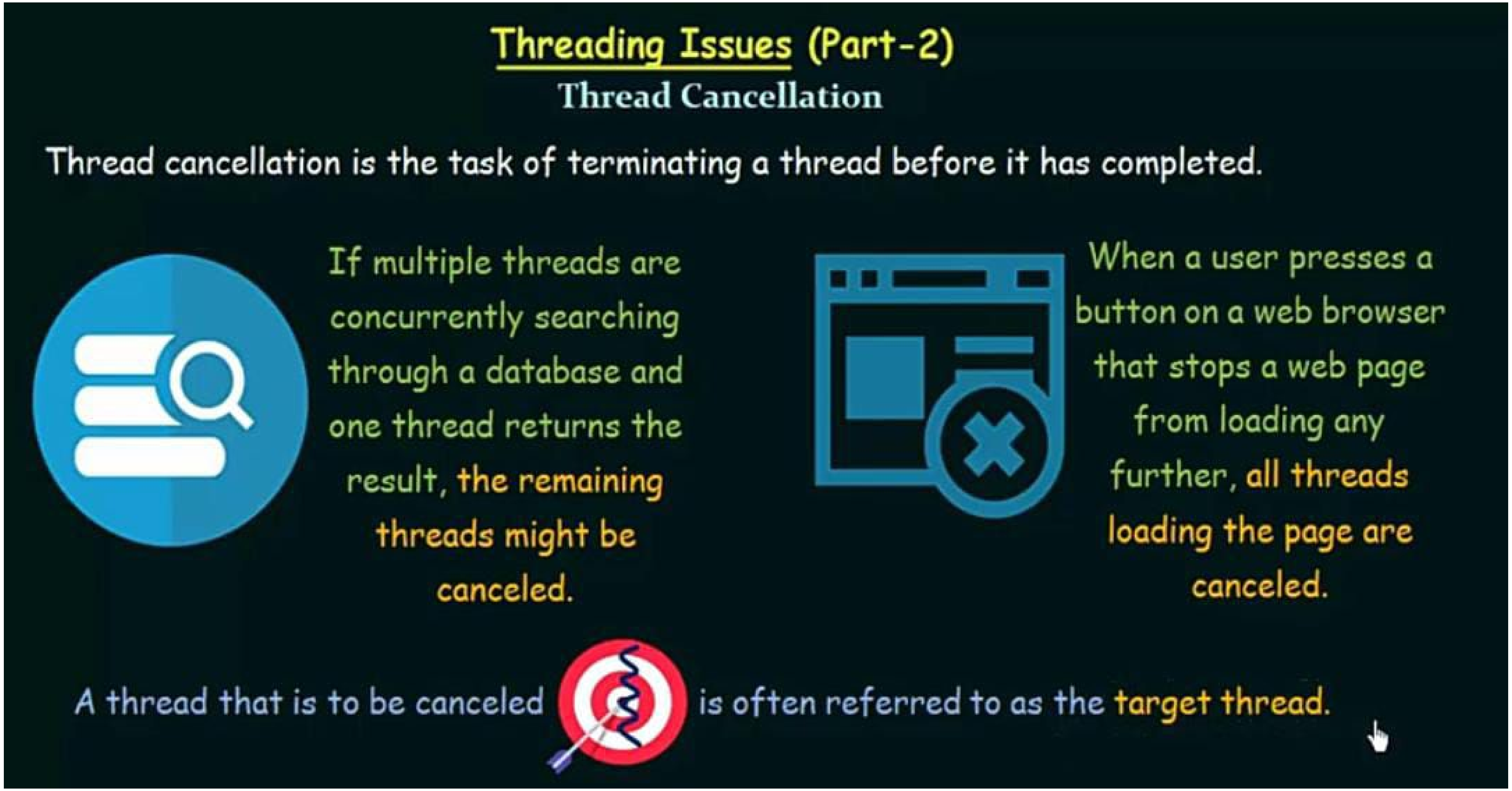






The benefits of multithreaded programming can be broken down into four major categories:

Responsiveness	Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
Resource sharing	By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
Economy	Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
Utilization of multiprocessor architectures	The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases



Threading Issues (Part-2)

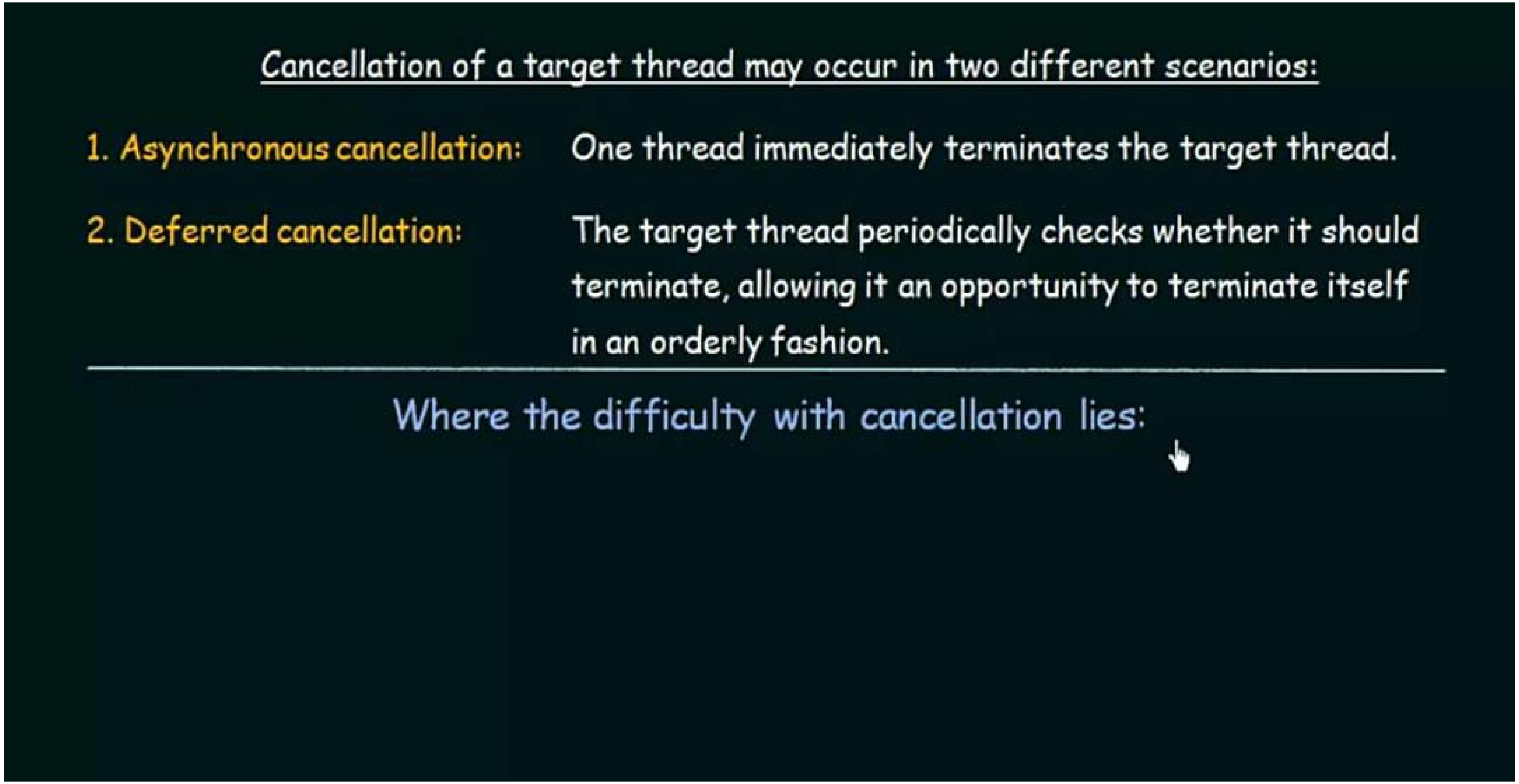
Thread Cancellation

Thread cancellation is the task of terminating a thread before it has completed.

If multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled.

When a user presses a button on a web browser that stops a web page from loading any further, all threads loading the page are canceled.

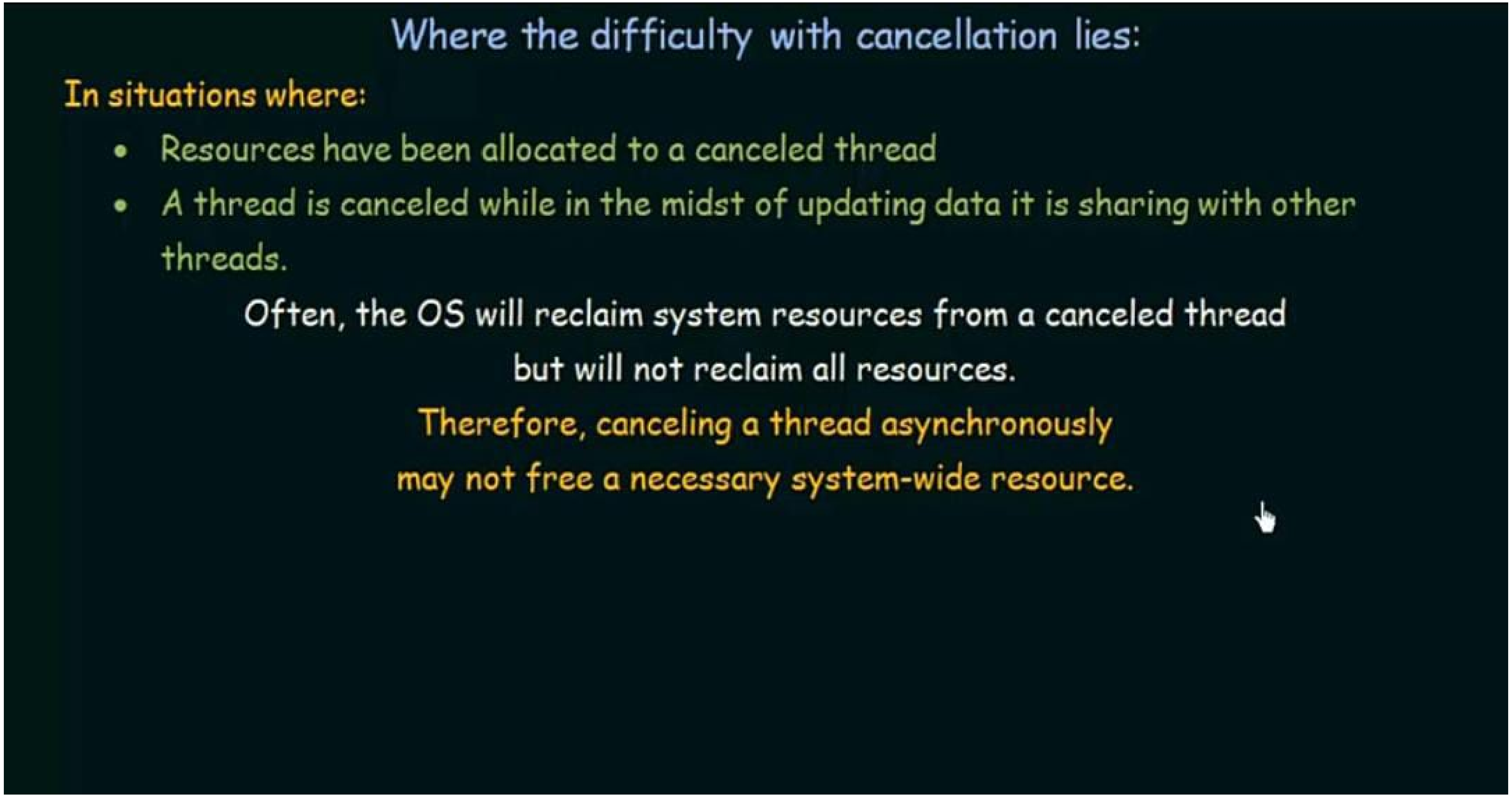
A thread that is to be canceled is often referred to as the target thread.



Cancellation of a target thread may occur in two different scenarios:

1. Asynchronous cancellation:	One thread immediately terminates the target thread.
2. Deferred cancellation:	The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Where the difficulty with cancellation lies:



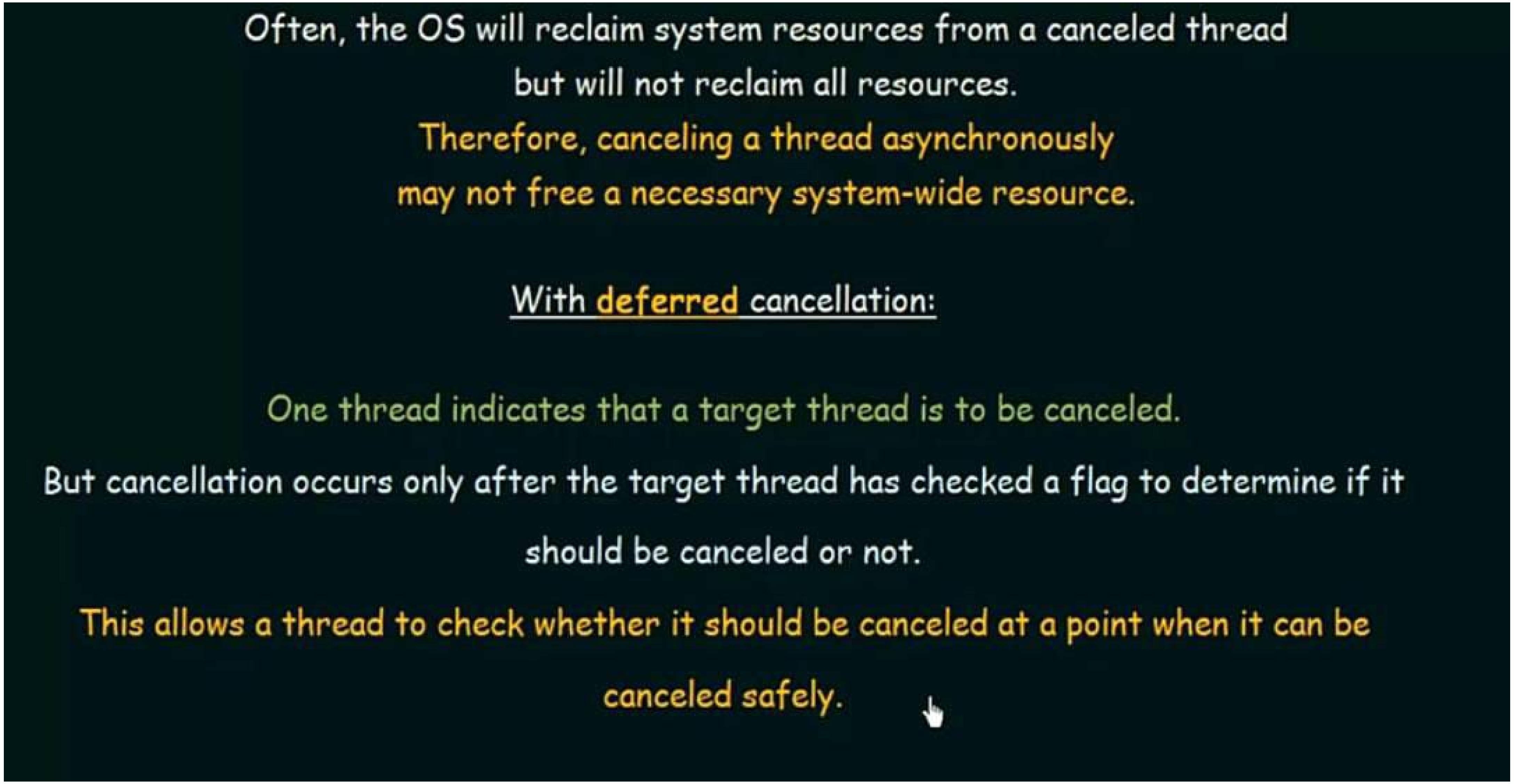
Where the difficulty with cancellation lies:

In situations where:

- Resources have been allocated to a canceled thread
- A thread is canceled while in the midst of updating data it is sharing with other threads.

Often, the OS will reclaim system resources from a canceled thread but will not reclaim all resources.

Therefore, canceling a thread asynchronously may not free a necessary system-wide resource.



Often, the OS will reclaim system resources from a canceled thread
but will not reclaim all resources.
Therefore, canceling a thread asynchronously
may not free a necessary system-wide resource.

With deferred cancellation:

One thread indicates that a target thread is to be canceled.

But cancellation occurs only after the target thread has checked a flag to determine if it should be canceled or not.

This allows a thread to check whether it should be canceled at a point when it can be canceled safely.

```
# include <stdio.h>
# include< conio.h>
 int main ( )
  {
  if (fork ( ) && fork ( ))
      fork ( );
    printf ("engineer");
     return 0;
     }
```

$$\begin{tikzcd}
 & P \arrow[dl] \arrow[dr] & & & \\
\circ\, C_1 \arrow[d] & & P \arrow[dl] \arrow[dr] & & \\
\text{engineer} & \circ\, C_2 & & P \; +\underline{ve} \arrow[dl] \arrow[dr] & \\
 & \text{engineer} & C_3 & & P \\
 & & \text{engineer} & & \text{engineer}
\end{tikzcd}$$