CS & IT ENGINEERING

Operating System

File System & Device Management

Lecture No. 5

By- Dr. Khaleel Khan Sir

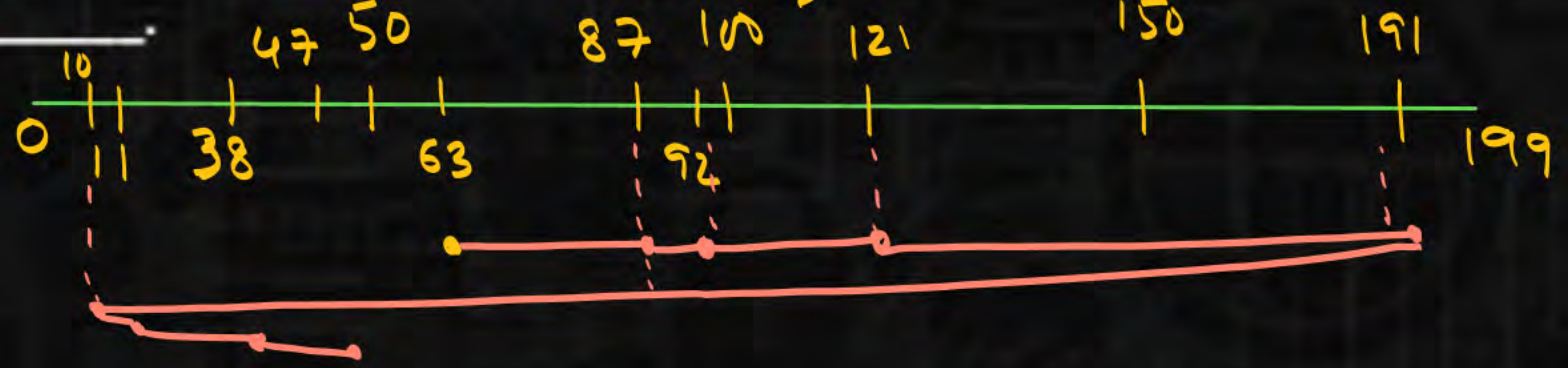
TOPICS TO BE COVERED

- Need for Disk Scheduling
- Disk Scheduling Techniques
- Threads
- System Call Implementation
- Fork System Call

Q.

Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is

346



Q. Consider a Disk with the Following pertinent details:

Track-Track Time = 1 ms

Current Head Position = 65

Direction: moving towards higher tracks with highest Track being 199.

Current Clock time = 160 ms

Consider the Following Data Set:

Calculate the Time of Decision, Pending Requests, Head Position, Selected Request, Seek Time using FCFS, SSTF, SCAN, LOOK, CSCAN, C-LOOK Algorithms.

Serial No.	Track No	Time of Arrival
1	12	65 ms
2	85	80 ms
3	40	110 ms
4	100	120 ms
5	75	175 ms

0 — FCFS — 199

Clk-Time: 160

TTT = 1 ms

R/W : 65

	1	2	
1) Time of decision	160 ms	213 ms	286 ms
2) Pending Req's	1,2,3,4	2,3,4,5	3,4,5
3) Head Position	65	12	65
4) Selected Req	1 <12>	2 <85>	3 <40>
5) Seek Time	53ms	73ms	25ms

Q. Disk requests come to disk driver for cylinders 10,22,20,2,40,06 and 38, in that order at a time when the disk drive is reading from cylinder 20. The seek time is 6 msec per cylinder.

Compute the total seek time if the disk arm scheduling algorithm is:

(a) First come first served. : $146 \times 6 = 876$ ms

(b) Closest cylinder next. : $60 \times 6 = 360$ ms

(S.S.T.F)

Q.

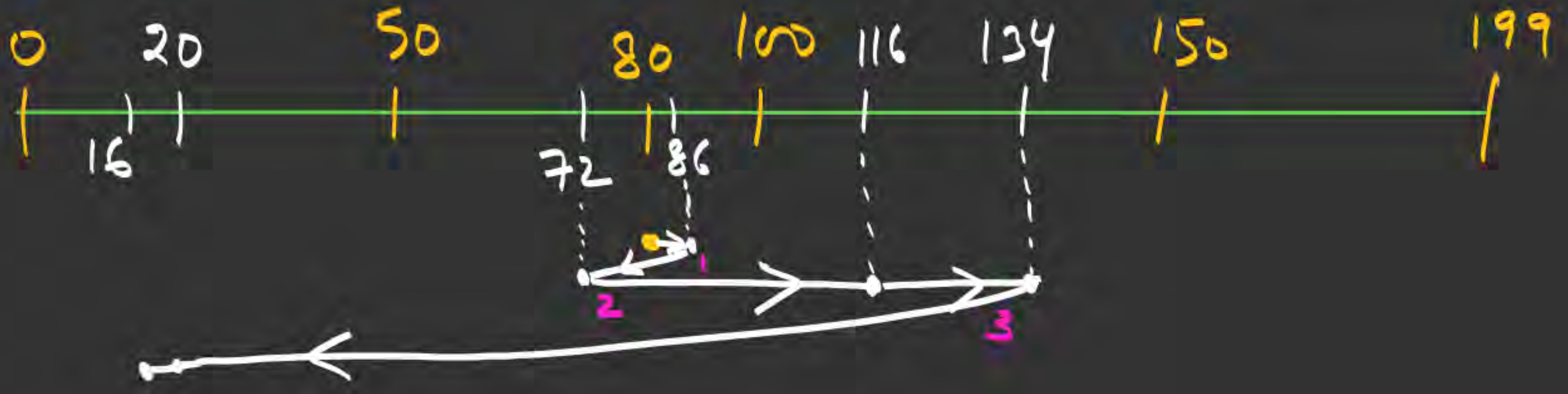
Consider a storage disk with 4 platters (numbered as 0,1,2 and 3), 200 cylinders (numbered as 0, 1, ., 199), and 256 sectors per track (numbered as 0, 1, , 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:

[120, 72, 2], [180, 134, 1], [60, 20, 0] [212, 86, 3], [56, 116, 2], [118, 16, 1]

Currently the head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts.

Power dissipation associated with rotational latency and switching of head between different platters is negligible. The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithms is 85

Req's: $\langle 72, 134, 20, 86, 116, 16 \rangle$ **SSTF**



Total seeks : 200 $\longrightarrow$ $20 \times 2 = 40$ mW

No. of times R/W changes its direc : 3 $\longrightarrow$ $3 \times 15 = 45$ mW

85 mW

Q.

Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing __3__ number of requests.

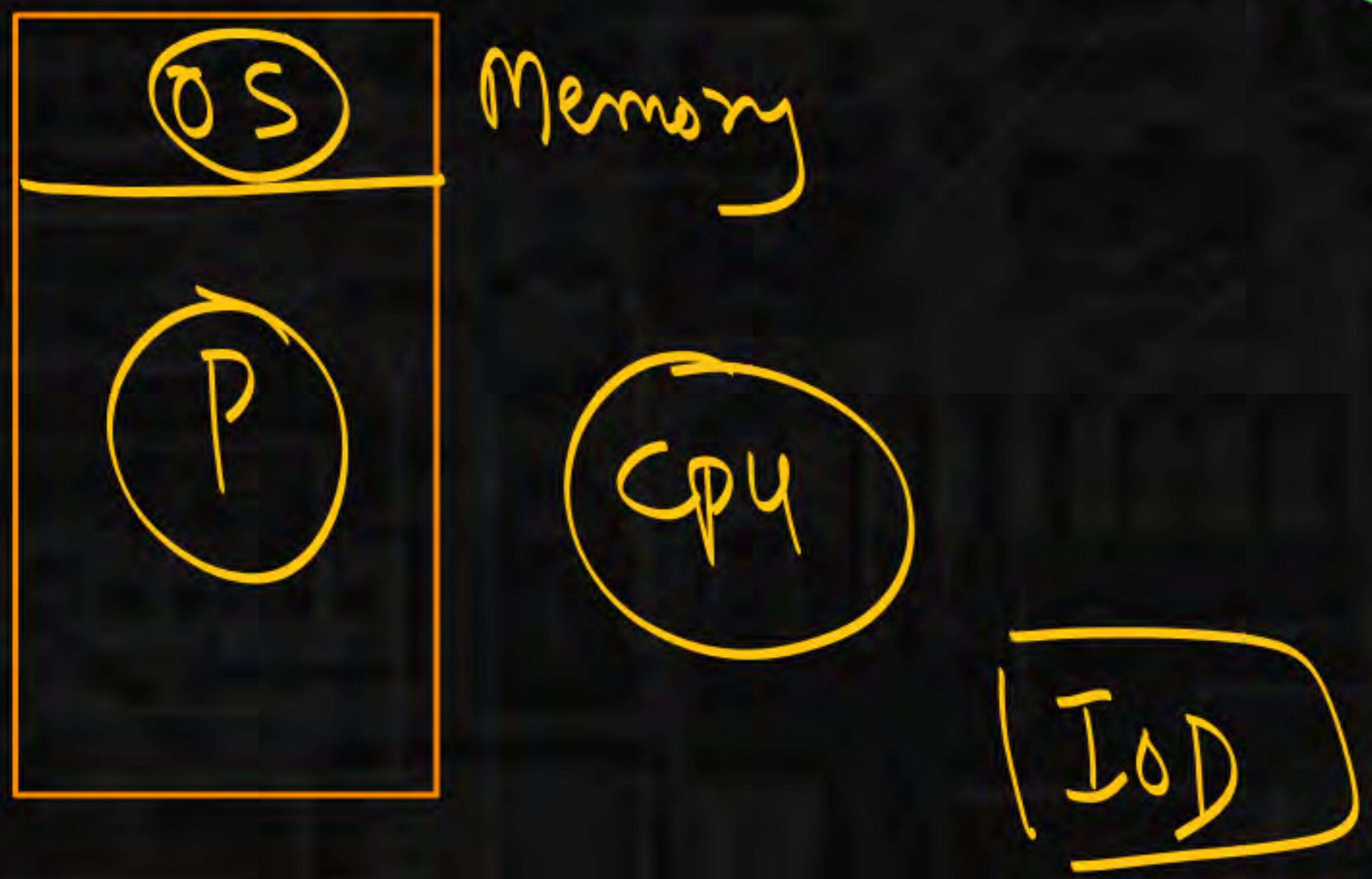
Q.

H/W

Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is ____ tracks.

Q. Consider an operating System capable of loading and executing a single sequential user Process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50 % better benchmark results, what is the expected improvement in the I/O performance of user programs?

A) 100%   B) 50%   C) 25%   D) 0%



Q. The amount of Disk Space that must be available for Page storage is related to Maximum number of Processes 'N' , the number of Bytes in Virtual Address Space 'B' and the number of Bytes in RAM 'R'. Give an expression for the worst case Disk Space required.

Q. Consider the following five disk access requests of the form(request id, cylinder number) that are present in the disk scheduler queue at a given time.

(P, 155), (Q, 85), (R, 110), (S,30), (T,115)

Assume the head is positioned at cylinder 100. The scheduler follows Shortest Seek Time First Scheduling to service the requests. Which one of the following statements is FALSE?

A. R is serviced before P.

B. T is serviced before P.

C. Q is serviced after S, but before T

D. The head reverses its direction of movement between servicing of Q and P.

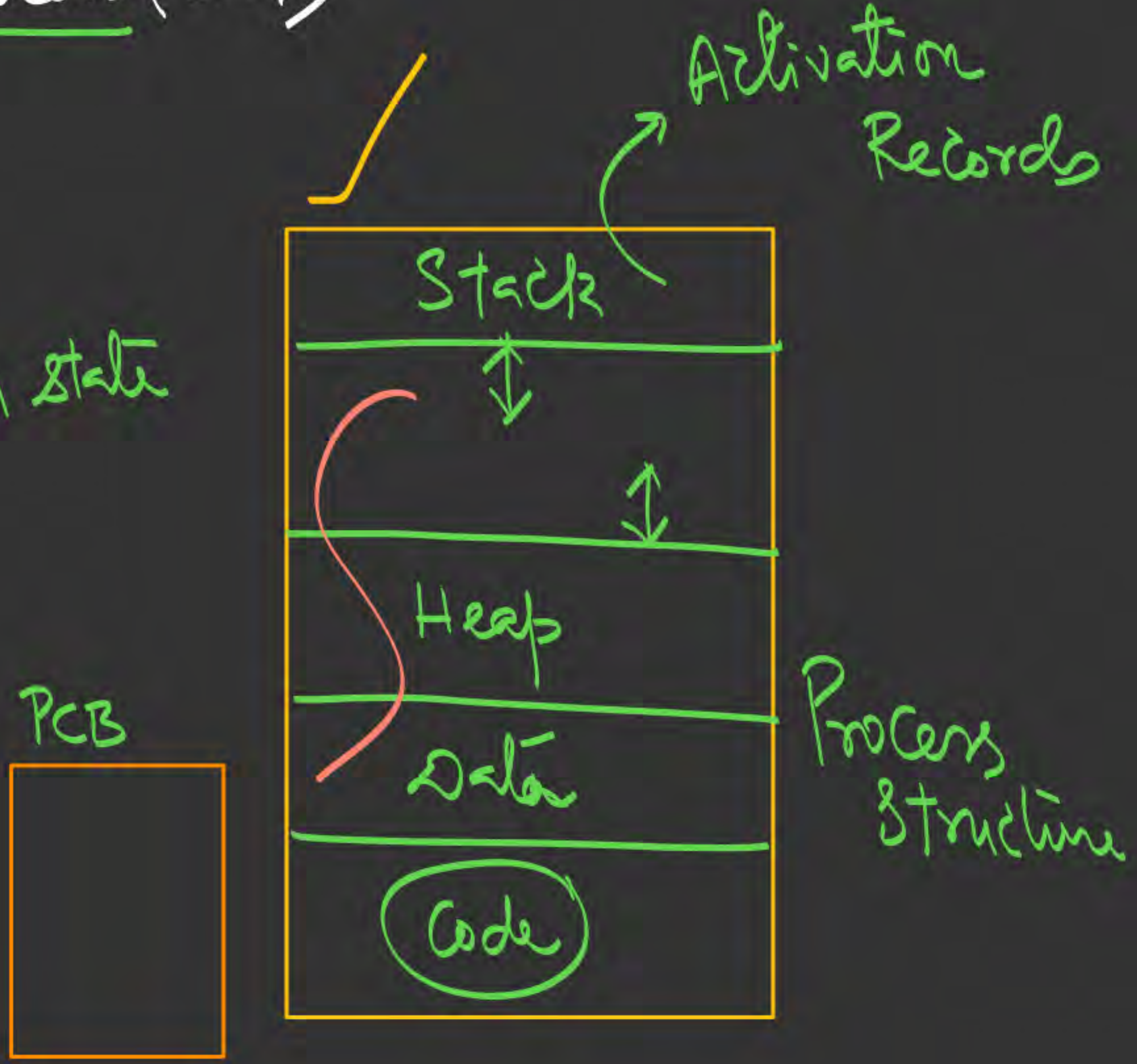
Threads & Multi-Threading

→ Thread is a light weight Process (LWP)

Program vs Process:

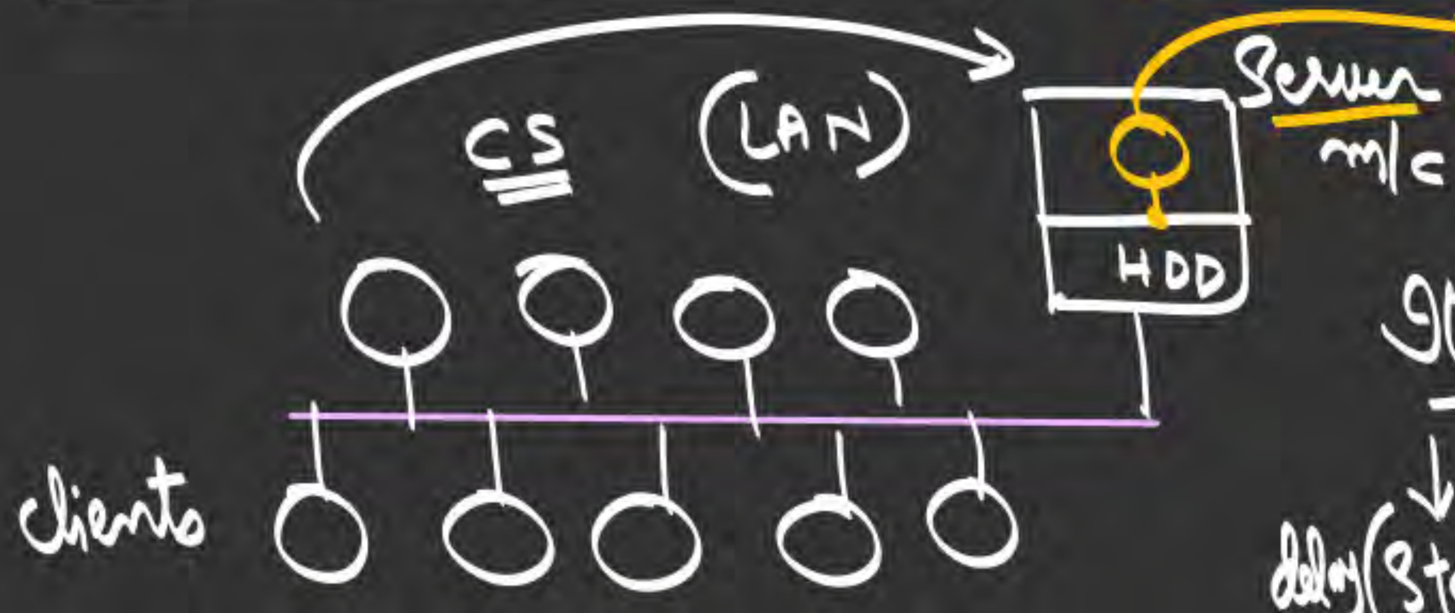
↳ Program in execution state

- Program
  - .exe
    - Instns / Code
    - Data
      - Static
      - Dynamic



Background for Threads
Motivation

Client - Server Environment:



Req - Q
$r_1 \; r_2 \; r_3$

5

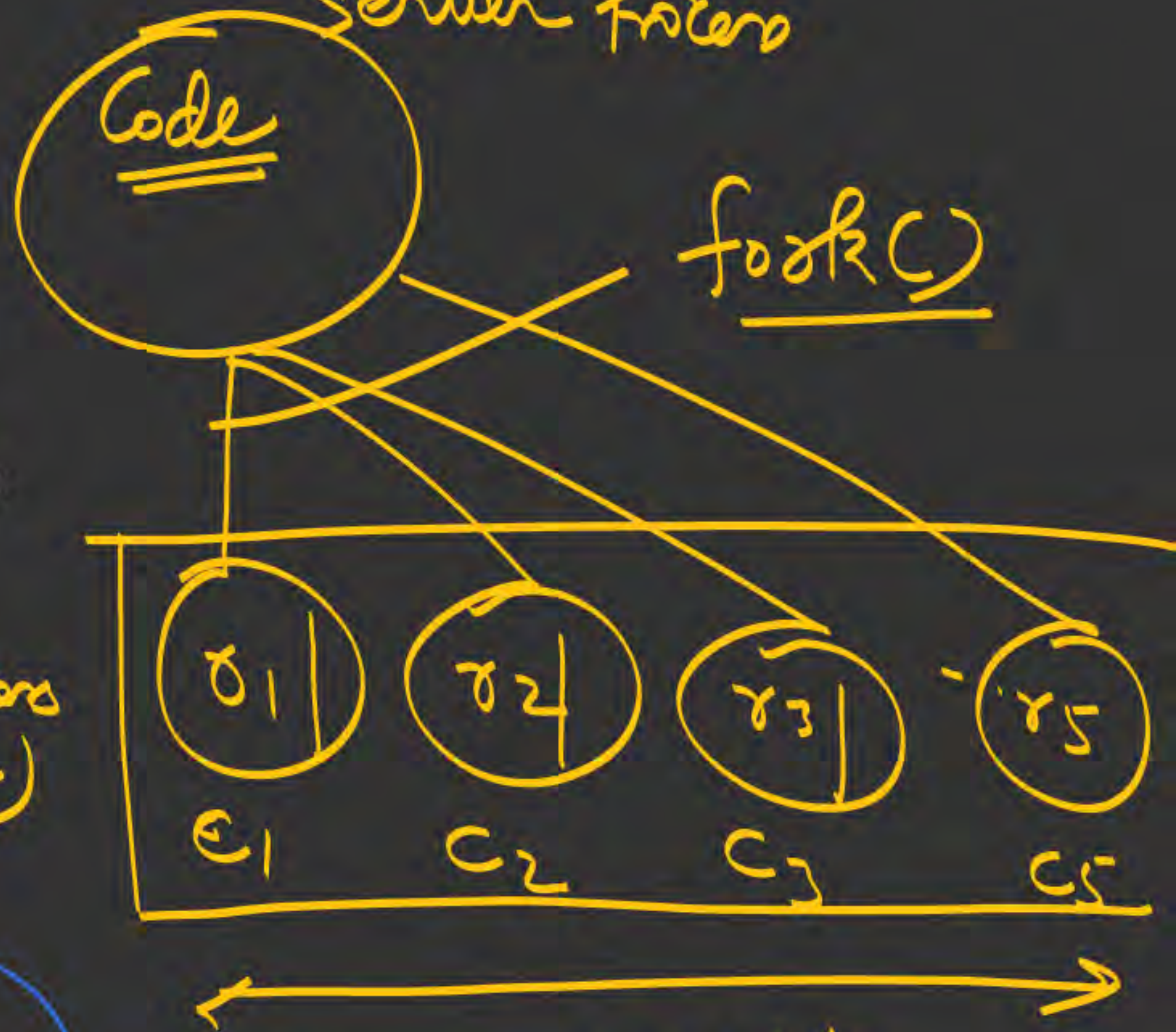
M.T.P

Server (Daemon)

Iterative → delay (Starvation) (Lack of responsive ness)

Concurrent* → (Multi-Process approach)

Server Process



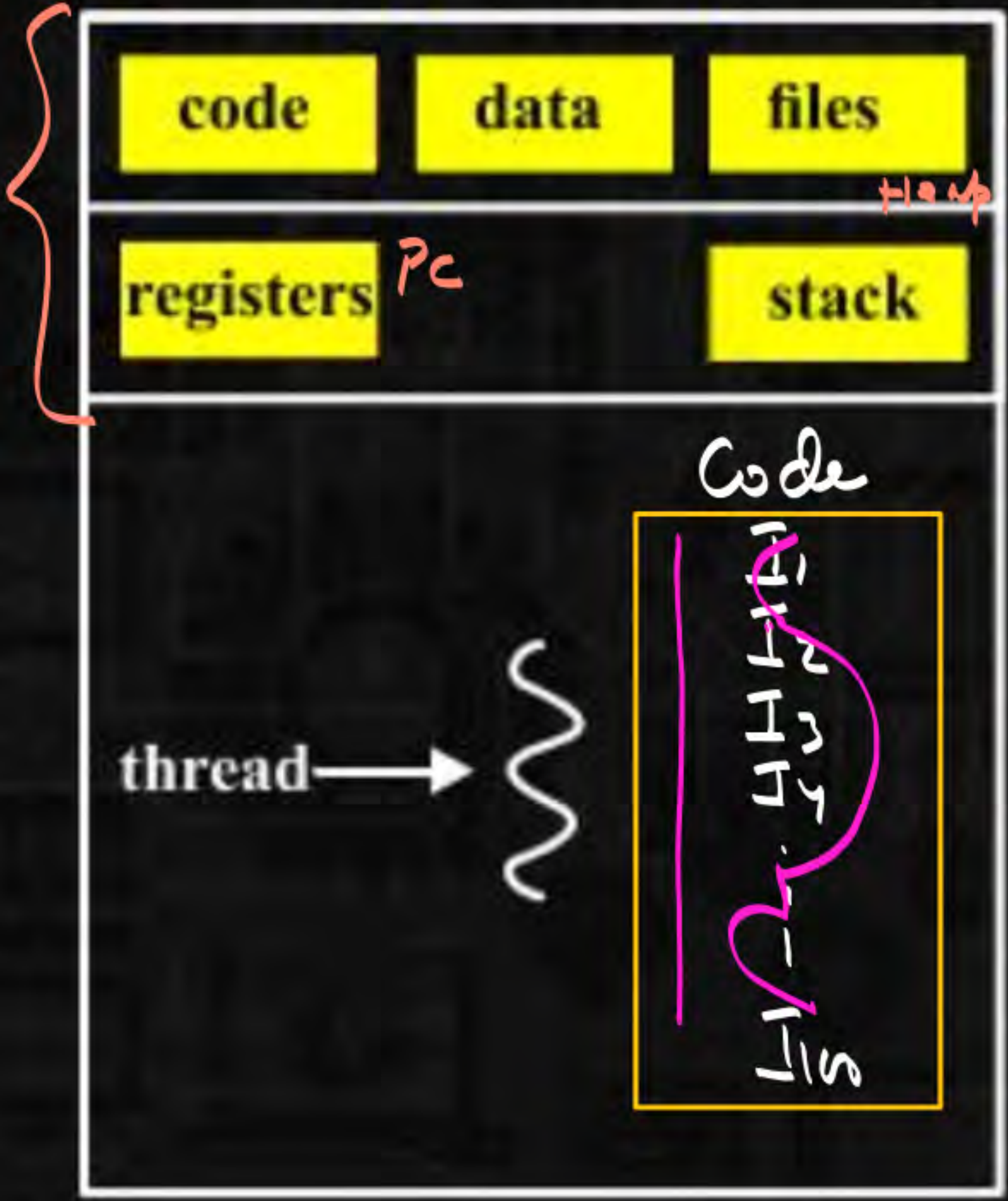
are an exact replica of original process

CPU

R.R
(TQ=5)

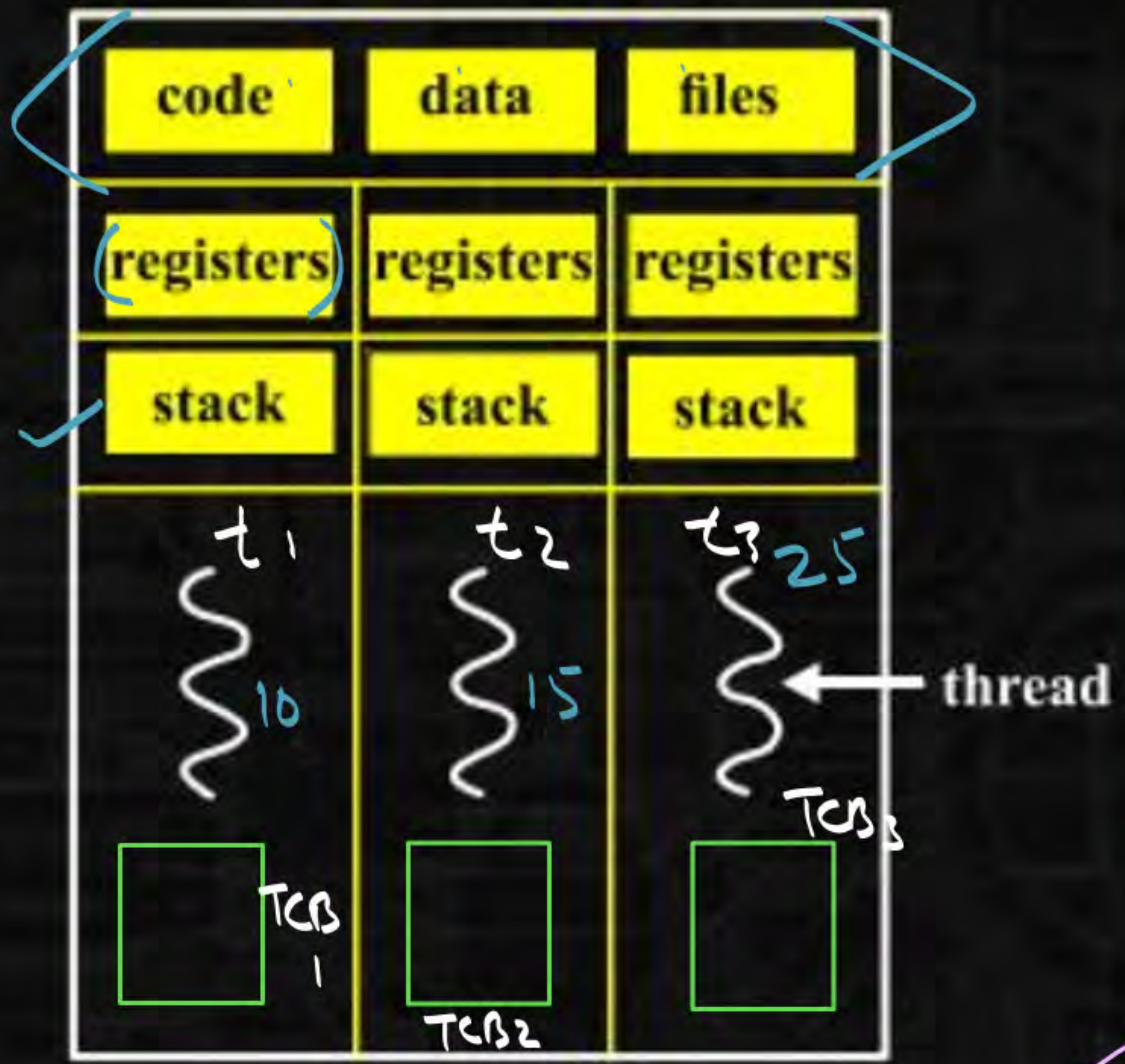
< Resource Sharing >

(Economy)



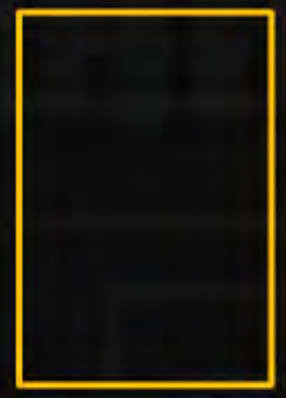
single -threaded process

Traditional Process



multithreaded process

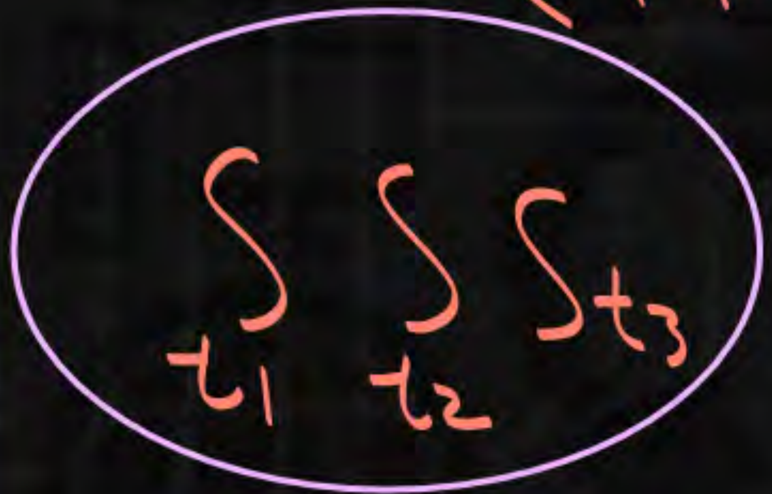
(M.T.P)

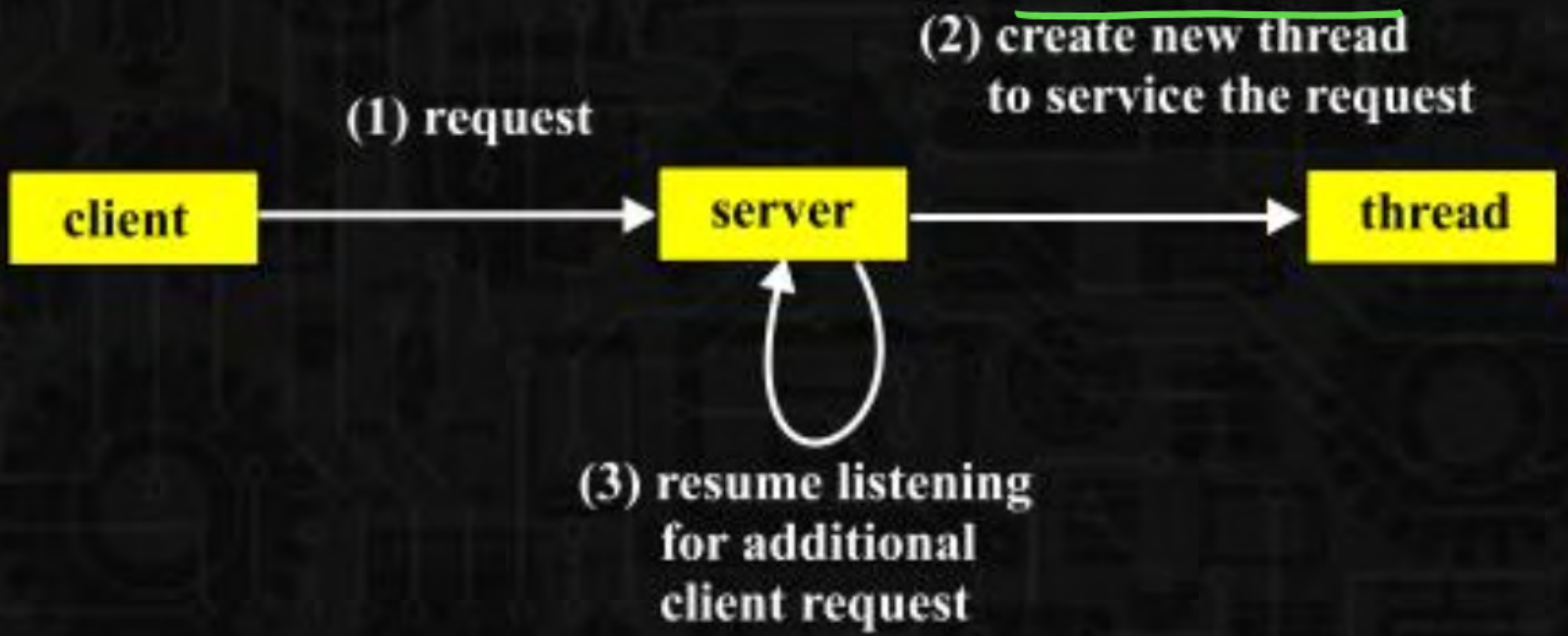


PCB

By default Each Process will have a Single Thread

(M.T.P)




(1) request
(2) create new thread
to service the request
client
server
thread
(3) resume listening
for additional
client request

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

1. 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. This quality is especially useful in designing user interfaces. For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive to the user.

2. Resource sharing: Processes can only share resources through techniques such as. shared memory. and. message) passing.. .Such. techniques, must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. 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.

3. Economy: Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is significantly more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times

4. **Scalability**: The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available. We explore this issue further in the following section.

5. Improved Performance due to Less Content switch overhead

$|TCB| < |PCB|$

Thread Switching is faster than Process switching

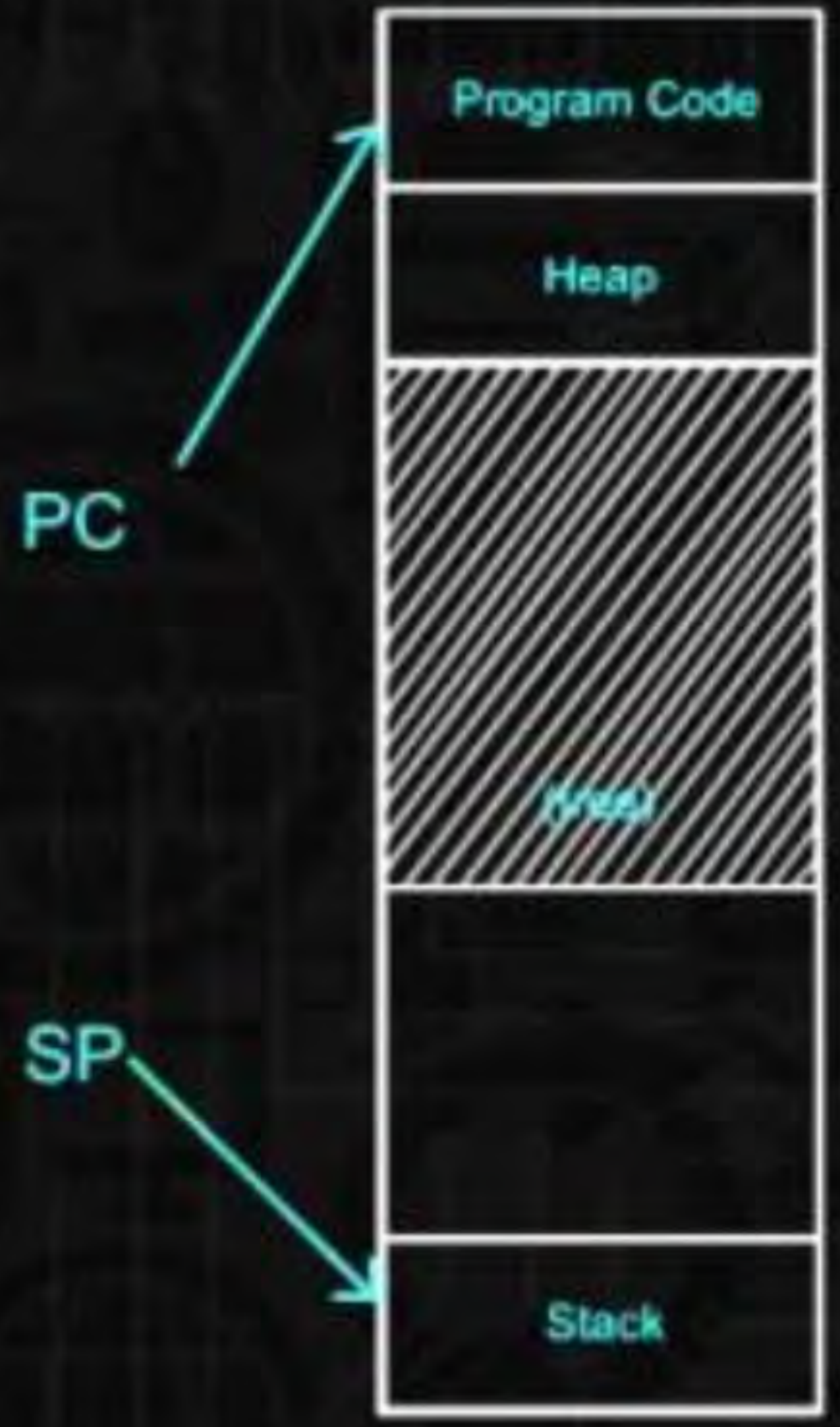
(Less overhead)

6. Can utilise Multi-core Architectures & achieve Parallelism

S S S

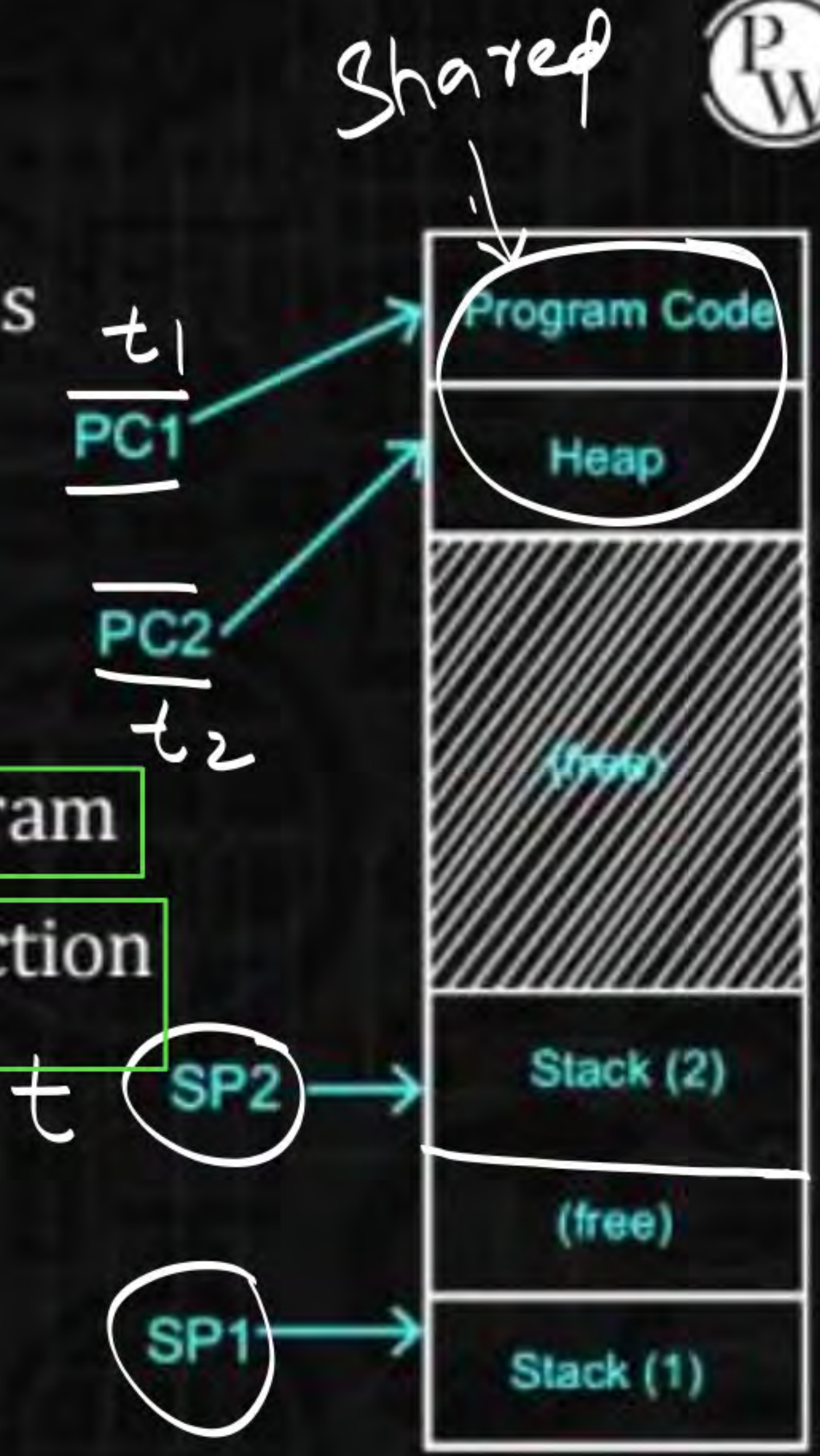
Single Threaded Process

- So,far we have studied single threaded programs
- Recap: process execution
  - Pc points to current instruction being run
  - SP points to stack frame of current function call
- A program can also have multiple threads of execution
- What is a thread



Multi Threaded Process

- A thread is like another copy of a process that executes Independently.
- Threads share the same address space (code heap)
- Each thread has a separate PC
  - Each thread may run over different part of a program
- Each thread has a separate stack for independent function calls



Process Vs. Thread

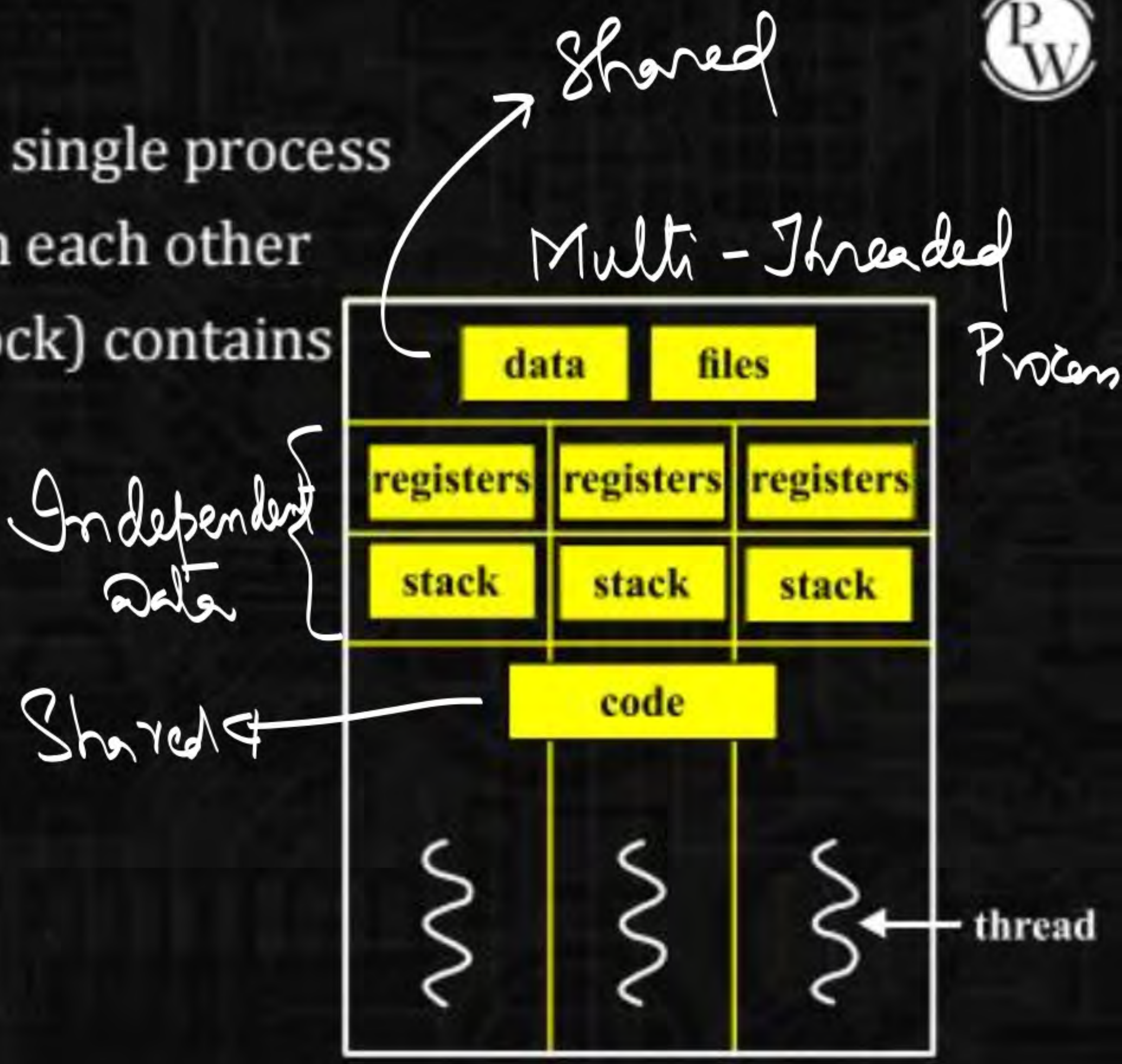
- Parent P forks a child C
  - P and C does not share any memory
  - Need complicated IPC mechanism to communicate
  - Extra copies of code, data in memory
- Parent p execute two threads T1 and T2
  - T1 and T2 share parts of the address space
  - Global Variables can be used for communication
  - Smaller memory footprint
- Threads are like separate processes, except they share the same address space

Why Threads?

- Parallelism: single process can effectively utilise multiple CPU cores
  - Understand the difference between concurrency and the parallelism
  - Concurrency: running multiple threads/ processes at the same time, even on single CPU core, by interleaving their execution
  - Parallelism: running multiple threads/process in parallel over different CPU cores
- Even if no parallelism can concurrency of threads ensure effective use of CPU when one of the threads blocks(e.g.,for l/0)

Threads

- Separate stream of execution within a single process
- Threads in a process not isolated from each other
- Each thread States (thread control block) contains
  - Registers including (EIP, ESP)
  - Stack



Threads Vs Processes

- A thread has no data segment or heap.
- A thread cannot live on its own it need to be attached to a process.
- There can be more than one thread in a process. Each thread has its own stack.
- If a thread dies, its stack is reclaimed.
- A process has code, heap, stack, and other segments
- $\boxed{\text{A process has at-least one thread.}}$
- Heads within a process share the same code, files.
- $\boxed{\text{If a process dies, all thread die.}}$

Types of Threads

- User-Level Threads (ULT)
- Kernel Level Threads (KLT)

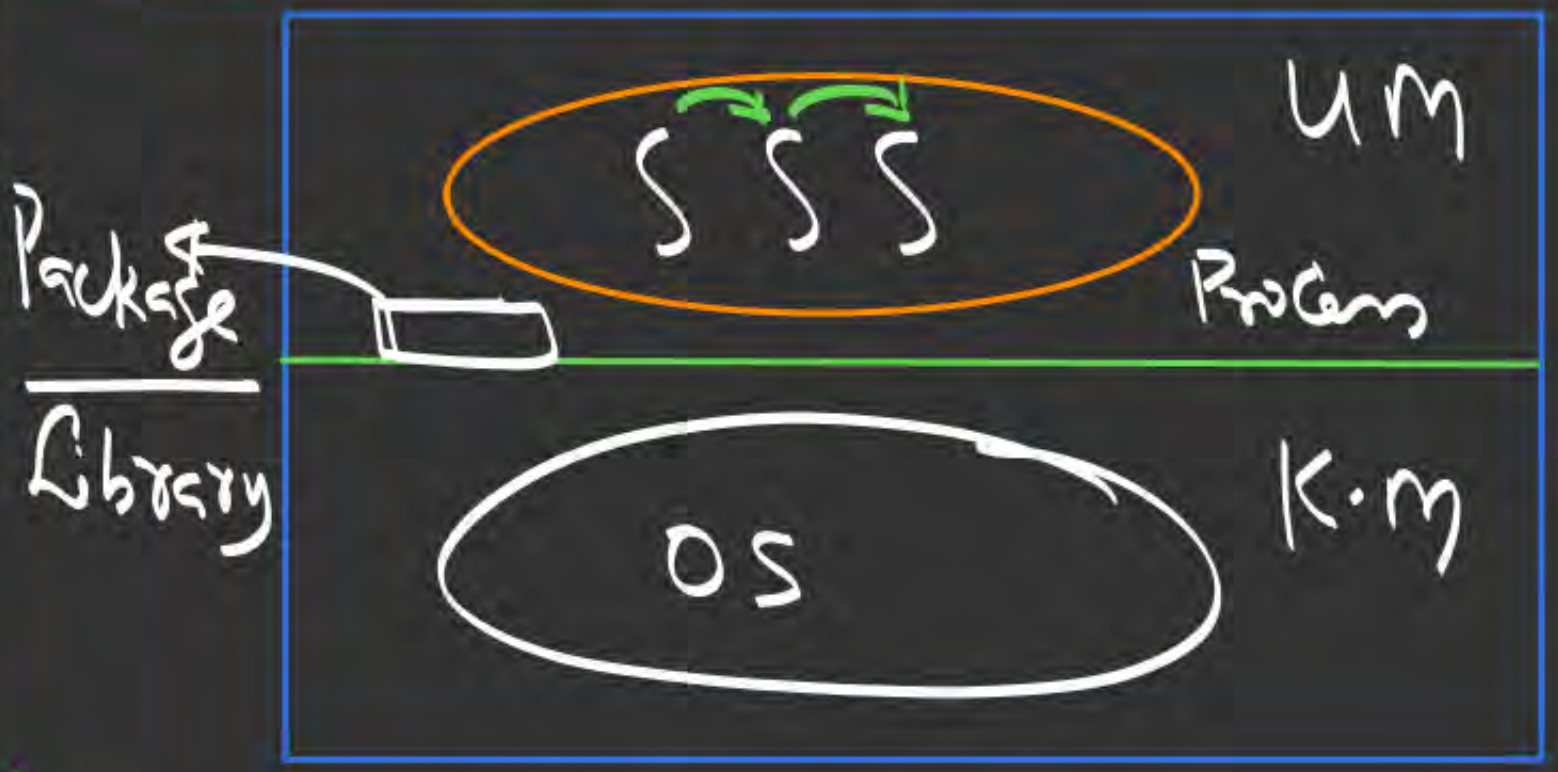
Benefits of ULTs

1) Transparency
2) Flexibility
3) Faster Context-switching (No Need of Mode shifting)

1) User-level Threads

→ Threads that are created & Managed @ the User Level (um) without any support of O.S;

PTHREAD

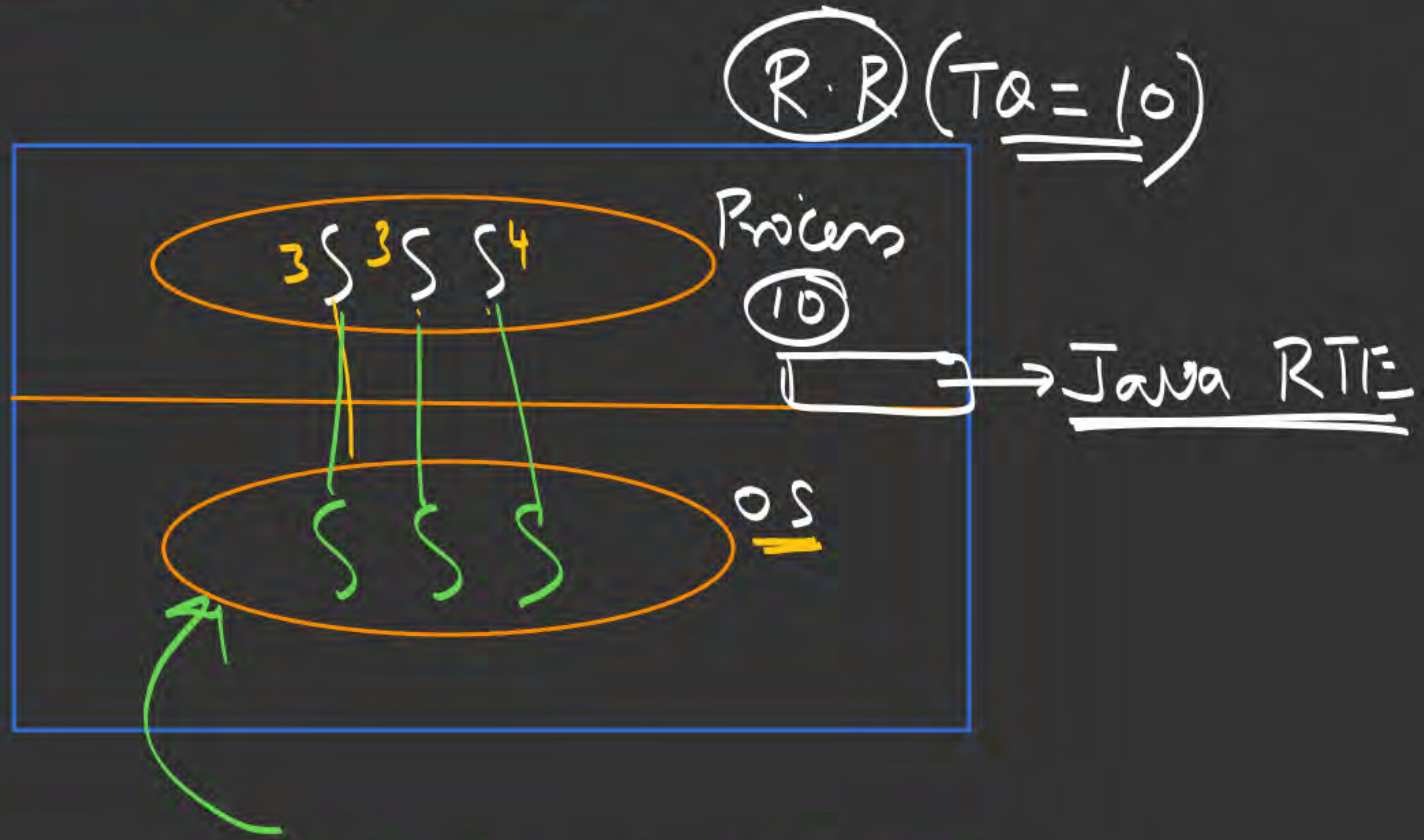


Transparency

Java

Drawback of U.L.T

Requirement of I/O/S.C by one Thread will result in blocking of whole Process)



Multi-Threaded Kernel

Pthread library

- Create a thread in a process

```
Int
pthread_create(pthread_t*thread,
Const pthread_attr_t*attr,
void*(*start_routine) (void*),
void*arg);
```

- Destroying a thread

```
void
pthread_exit(void*retval);
```

Thread identifier (TID) much like

Pointer to a function which starts

execution in a different thread

Arguments to the function

Exit vale of the thread

Pthread library contd.

- Join : wait for a specific thread to complete
  Int pthread_join(pthread_t thread, void**retval);
  TID of the thread to wait for exit status of the thread

Who manages threads?

Two strategies

- User threads
    - Thread management done by user level threads library. Kernel knows nothing about the threads.
- Kernel threads
    - threads directly supported by Kernel.
    - Known as light weight processes.

Use Level threads

Advantages:

- Fast (really lightweight) (no system calls to manage threads. The thread library does everything).
- Can be implemented on an OS that does not support threading.
- Switching is fast. No switch from user to protected mode.

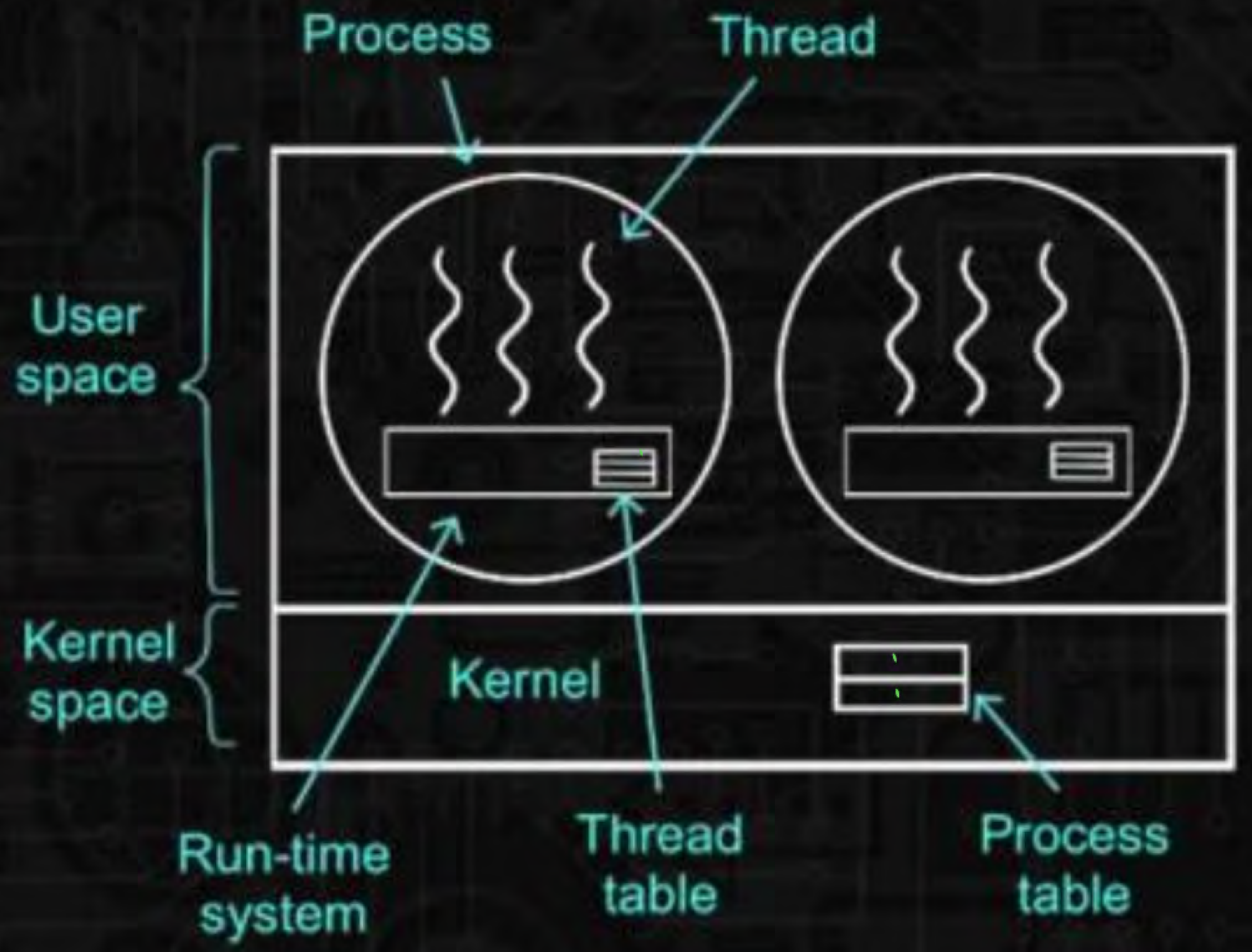
Disadvantages:

Scheduling can be an issue. (Consider, one third that is blocked on an IO and another runnable.)

Lack of coordination between kernel and threads. (A process with 100 threads complete for a time slice with the process having just 1 thread.)

Requires non-blocking system calls. (If one thread invokes a system call, all threads need to wait

Use Level threads



Kernel Level Threads

- Advantages:

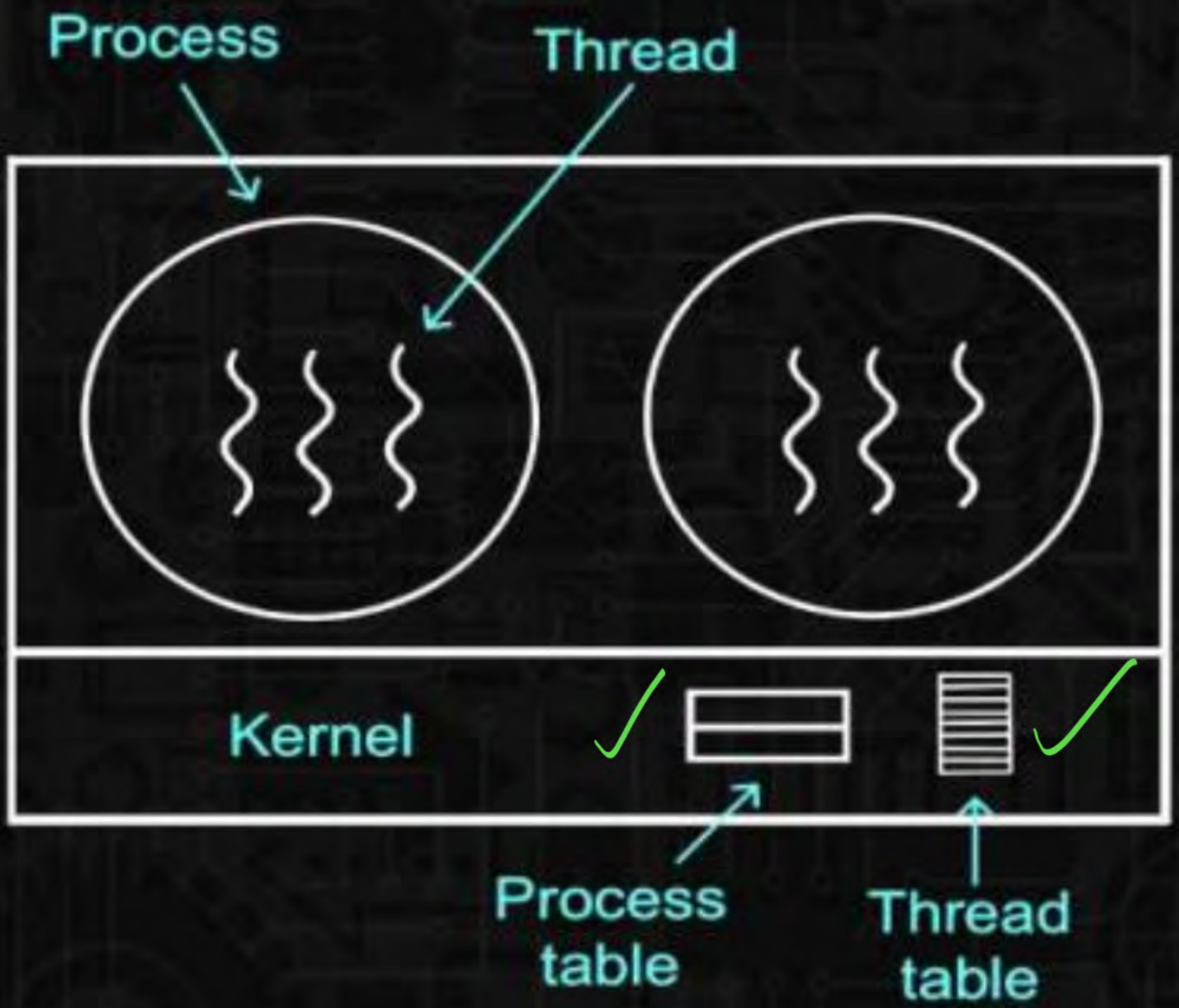
  Scheduler can decide to give more time to a process having small number of threads.

  Kernel-level threads are especially good for applications that frequently block.

- Disadvantages:

  The kernel-level are slow (they involve kernel invocations .

  Overheads in the kernel. (Since kernel must manage and schedule threads as well as processes. It required a full thread control block (TCB) for each thread to maintain information about threads.)


Process
Thread
Kernel
Process table
Thread table

Q. 1

Which of the following is/are shared by all the threads in a process?

I. Program counter

II. Stack

III. Address space

IV. Registers

A. I and II only

B. III only

C. IV only

D. III and IV only

Q. 2 Threads of a process share

A. Global variables but not heap

B. Heap but not global variables

C. Neither global variables nor heap

D. Both heap and global variables

Q. 3

Which one of the following is FALSE?

A. User level threads are not scheduled by the kernel. T

B. When a user level thread is blocked, all other threads of its process are blocked. T

C. Context switching between user level threads is faster than context switching between kernel level threads. T

D. Kernel level threads cannot share the code segment.

Q. 4

A thread is usually defined as a light weight process because an Operating System (OS) maintains smaller data structure for a thread than for a process. In relation to this, which of the following statement is correct?

A. F OS maintains only scheduling and accounting information for each thread.

B. F OS maintains only CPU registers for each thread.

C. OS does not maintain virtual memory state for each thread.

D. F OS does not maintain a separate stack for each thread.

Q. 5

Consider the following statements about user level threads and kernel level threads. Which one of the following statements is FALSE?

A. T Context switch time is longer for kernel level threads than for user level threads.

B. T User level threads do not need any hardware support.

C. T Related kernel level threads can be scheduled on different processor in a multi-processor system.

D. F Blocking one kernel level thread blocks all related threads.

Q. 6

Which one of the following is NOT shared by the threads of the same process?

A. Stack ✓

B. Address Space : Shared

C. File Descriptor Table : Shared

D. Message Queue : Shared

Q. 7

Consider the following statements with respect to user-level threads and kernel-supported threads

I. Context switch is faster with kernel-supported threads.

II. For user-level threads, a system call can block the entire process.

III. Kernel supported threads can be scheduled independently.

IV. User level threads are transparent to the kernel.

Which of the above statements are true?

A. II, III and IV only

B. II and III only

C. I and III only

D. I and II only


THANK YOU GW
SOLDIERS !