CS & IT ENGINEERING

Operating Systems

Miscellaneous Topics

Lecture No. 1

By- Dr. Khaleel Khan Sir

TOPICS TO BE COVERED

- Fork() System Call
- Threads
- Monitors

```
main()
{
    int a;

    if (fork() == 0)
    {                        //child//
        a = a+5;
        print(a);      ⟹ "u"
    }
    else
    {                        //Parent//
        a = a-5;
        print(a);      ⟹ "x"
    }
}
```

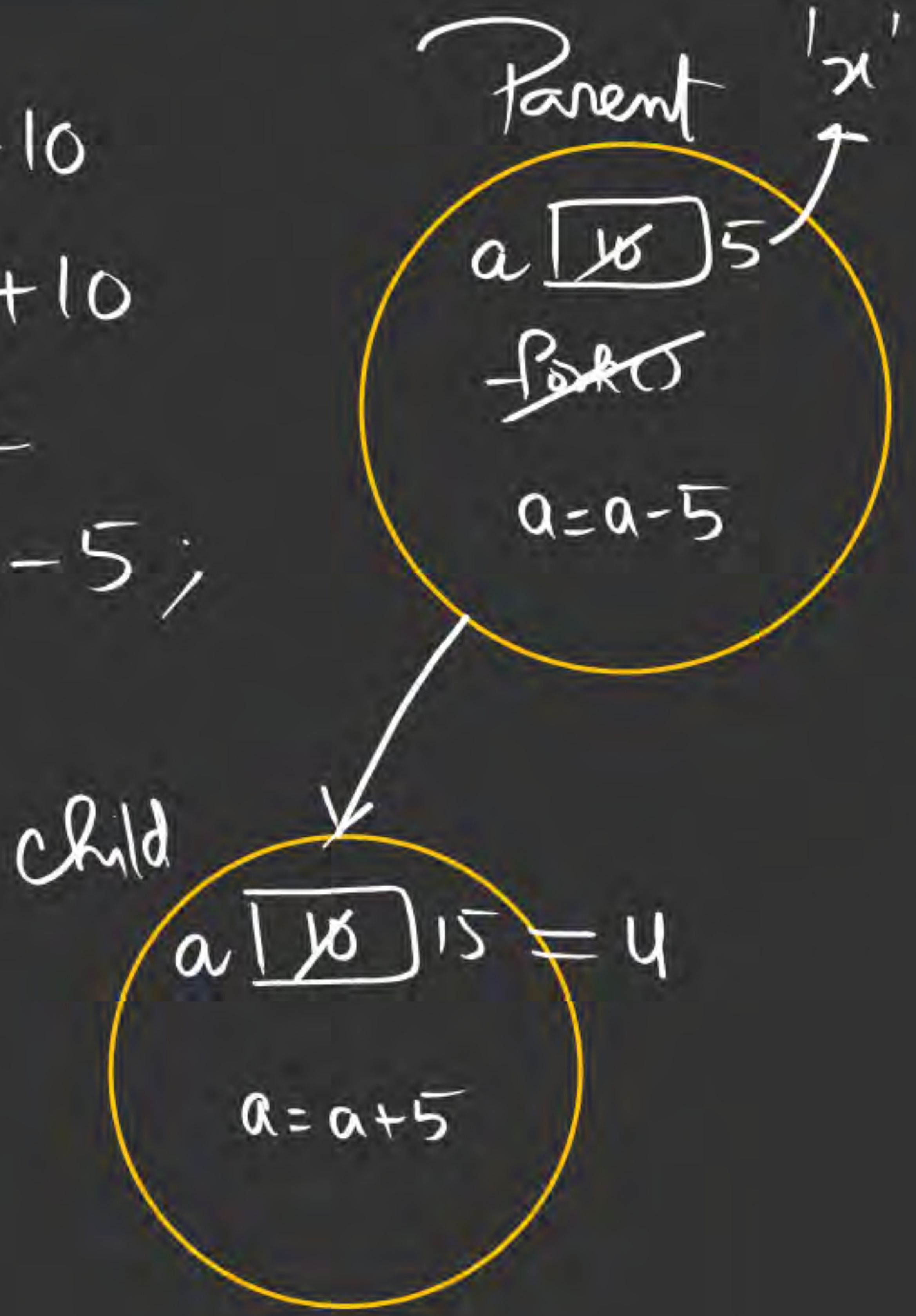
$u \propto x$

a) $u = x + 10$

b) $x = u + 10$

c) $u = x$

d) $u = x - 5$;



Q. 1

```
main ()
{
    int i, n;
    for (i = 1; i < = n; + + i)
    if (fork ( ) = = 0) ;
        print ("*");
}
```

→ outside for-loop

All : child + Parent

No. of Times '*' gets printed (2^n - 1) ✓

2^n ✓

Q. 2

```
main ()
{
    int i, n;
    for (i = 1; i< = n; ++i)
    {
    1. fork ();
    2. print ("*");
    }
}
```

1

**

H/W : No. of Times '*' ______

Q3

```
for i <- 1 to n
{
    print('*');
    fork();
}
```

*

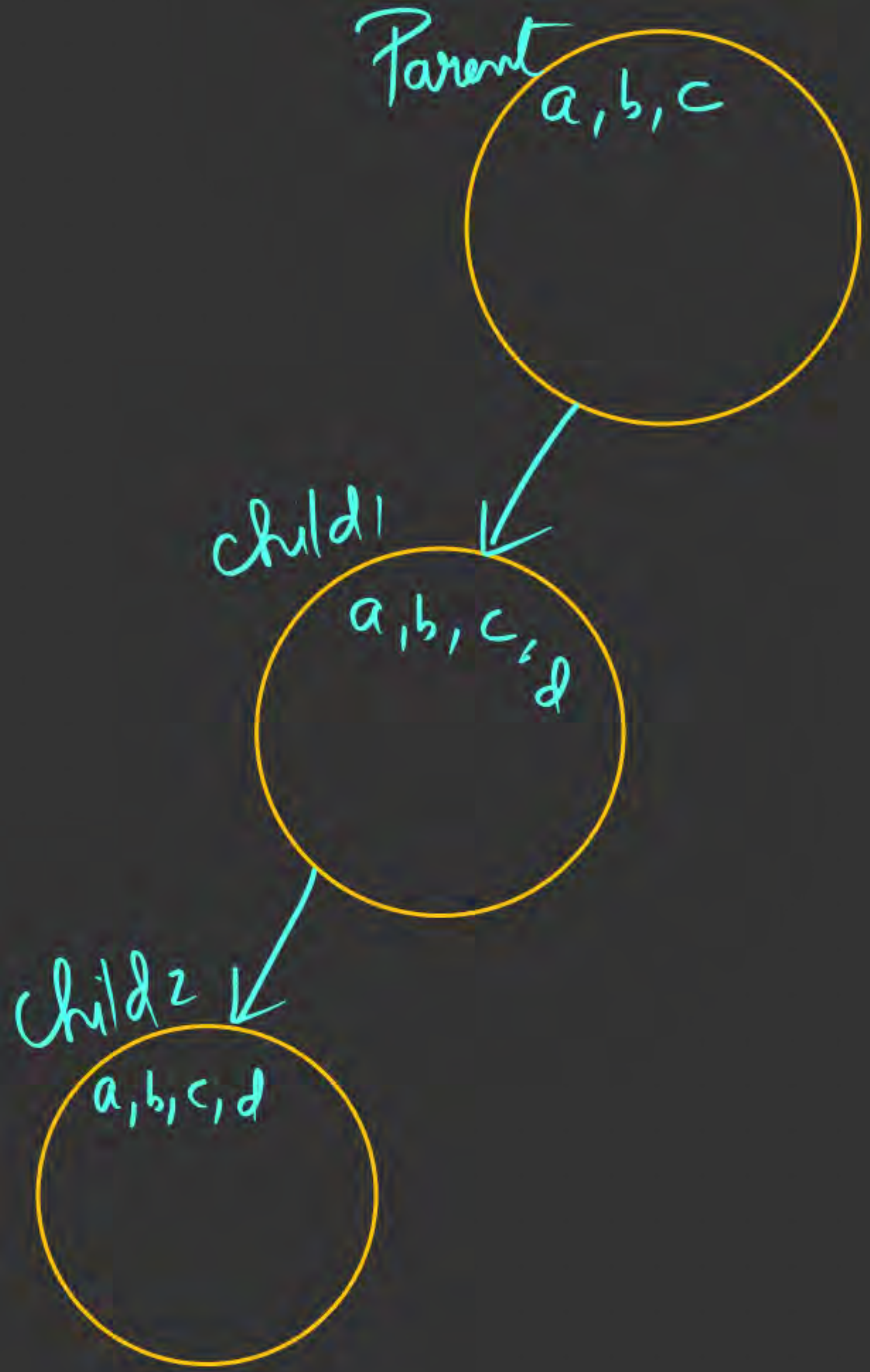
Q. 3

```
main ( )
{
    int i, n;
    for (i = 1; i < = n; ++ i)
    {
        print ("*");
        fork ( );
    }
}
```

Q. 4

```
main ()
{
    int a = 1, b = 2, c = 3; d=4;
    a + = b+ = + + c;
    print (a, b, c);        // Parent //
    if (fork () = = 0)
    {                       // Child 1
     int d;
     + + a; + + b; - - c;
     print (a, b, c);
     if (fork () = = 0)
     {                      // child 2
         d = a + b + c;
         print (a, b, c, d);
     }
     else
     {                      // Child 1 //
         - - a; - - b;
         c = a + b; d = a + b + c;
         print (a, b, c, d);
     }
    }
    else                    // Parent //
    {
     c + = b + = + + a;
     print (a, b, c) ;
    }
    print (d);              All Processes
}
```

Associated with No. of Processes;



'd' is NOT defined in Parent

Q. 5

The following C program is executed on a Unix/Linux system:

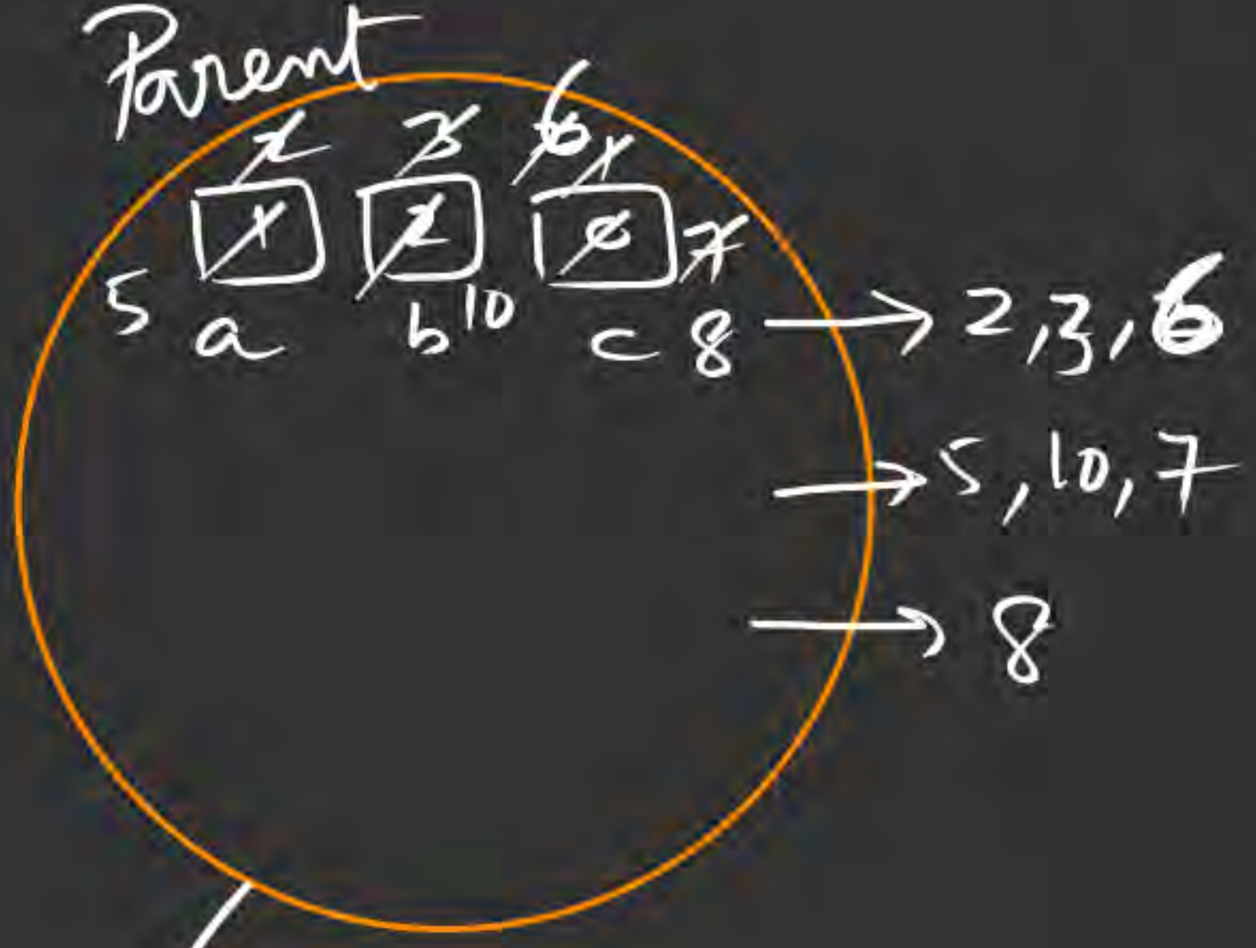
```
#include <unistd. h>
int main( )
{    int i;
     for (i = 0; i < 10; i + +)
        if (i % 2 = = 0) fork( );
}
```

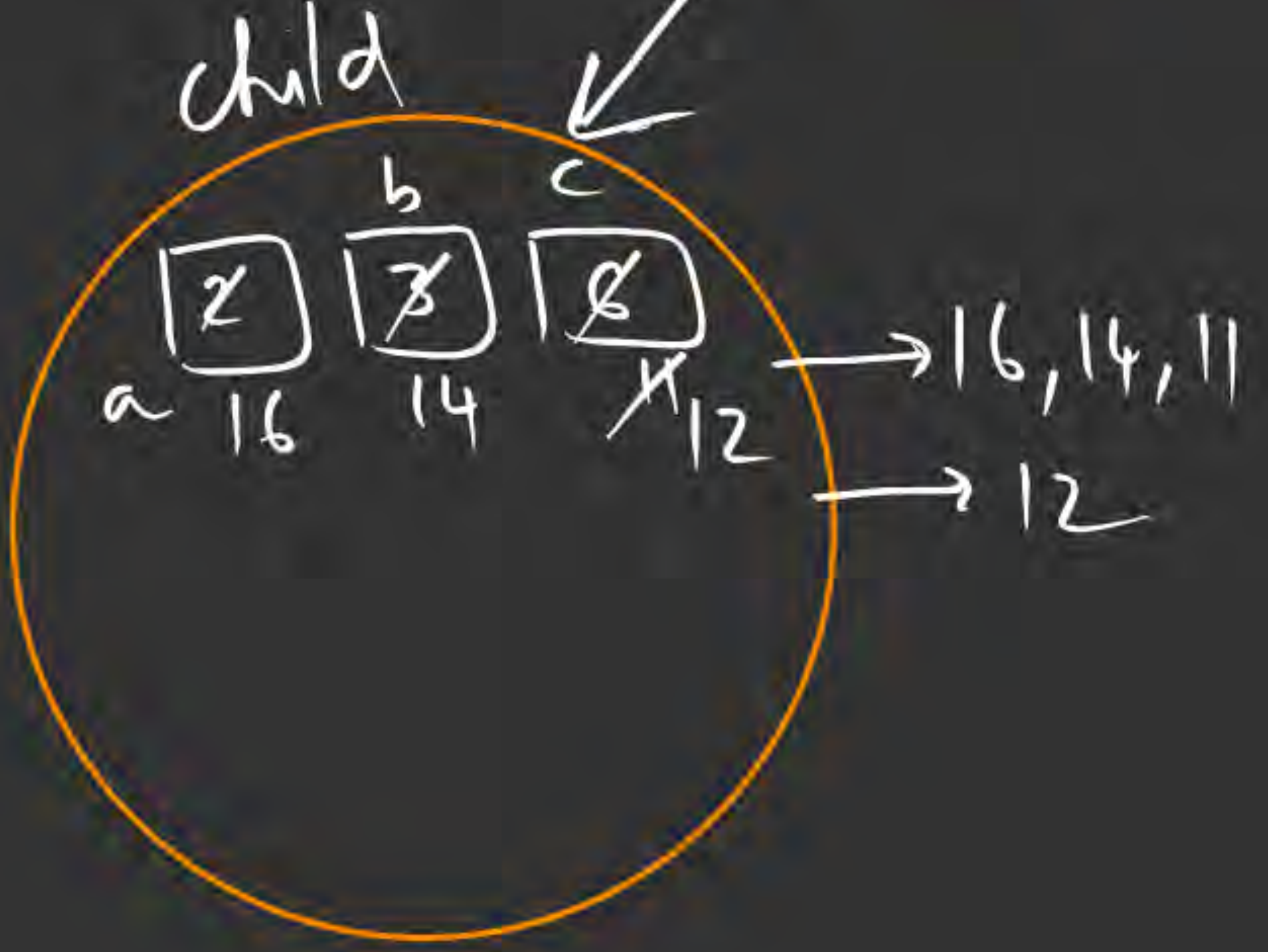
even (5) → '5' Times

The total number of child processes created is ______ $(2^5 - 1) = 31$

```
c = c + (++a) * (++b);

main()
{
    int a=1, b=2, c=0;
    c += ++a * ++b;
    Print(a,b,c);
    if (fork() == 0)
    {
        a += b += c += 5;
        print(a,b,c);
        c++;
    }
    else
    {
        b += c += a += 3;
        print(a,b,c);
        c++;
    }
    print(c);
}
```



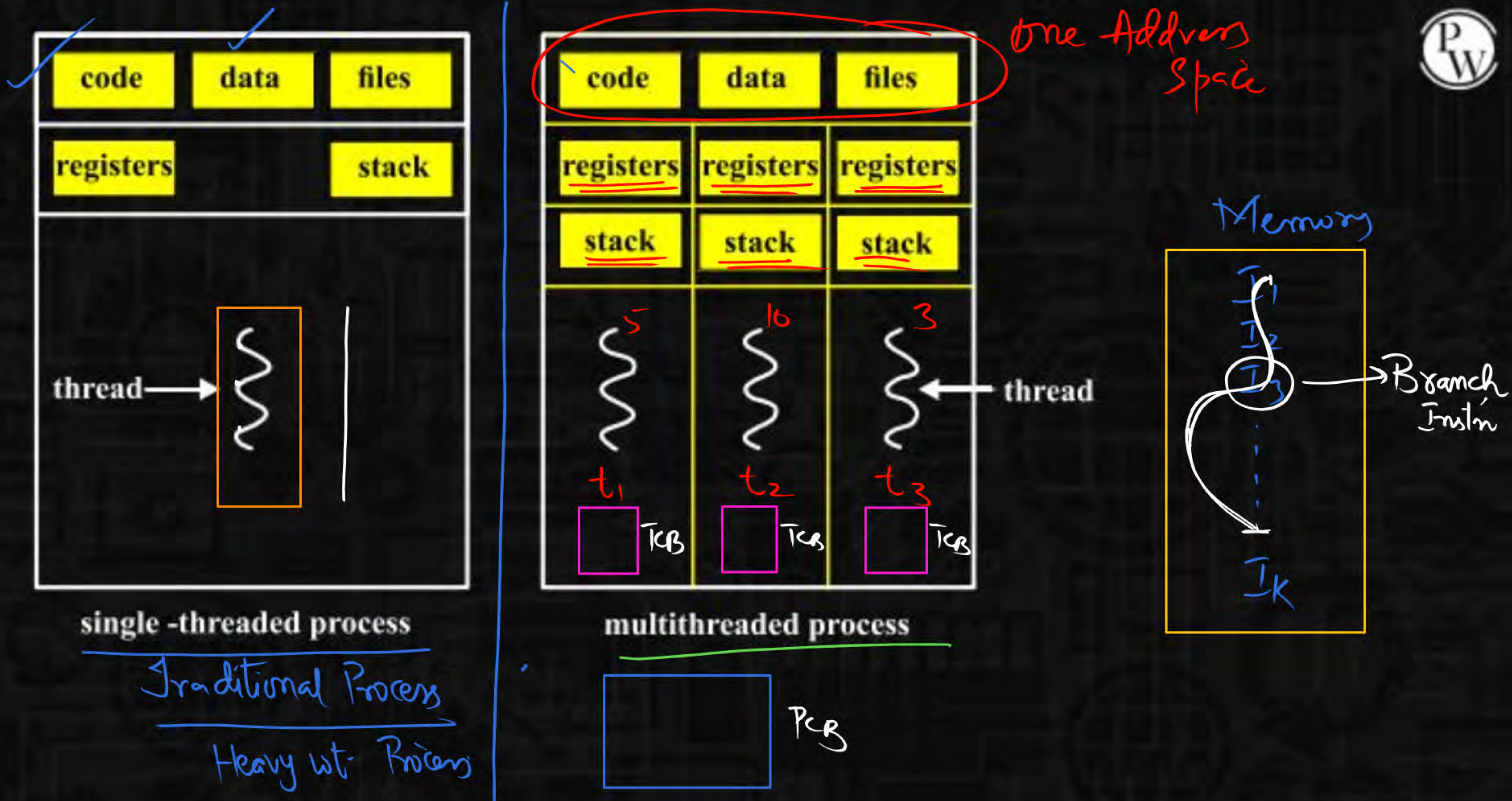


Threads & Multithreading:

(A Light weight Process)

- is an Active entity
- is a Schedulable unit
- is a unit of cpu utilization

- Part of Process
- uses resources of Process
- Context switch ovh'd b/w the Threads is less (Faster Context-switching)


code
data
files
registers
stack
thread
single -threaded process
Traditional Process
Heavy wt. Process
One Address Space
code
data
files
registers
registers
registers
stack
stack
stack
5
10
3
thread
t1
t2
t3
TCB
TCB
TCB
multithreaded process
PCB
Memory
I1
I2
I3
Ik
Branch Instn

Motivation :

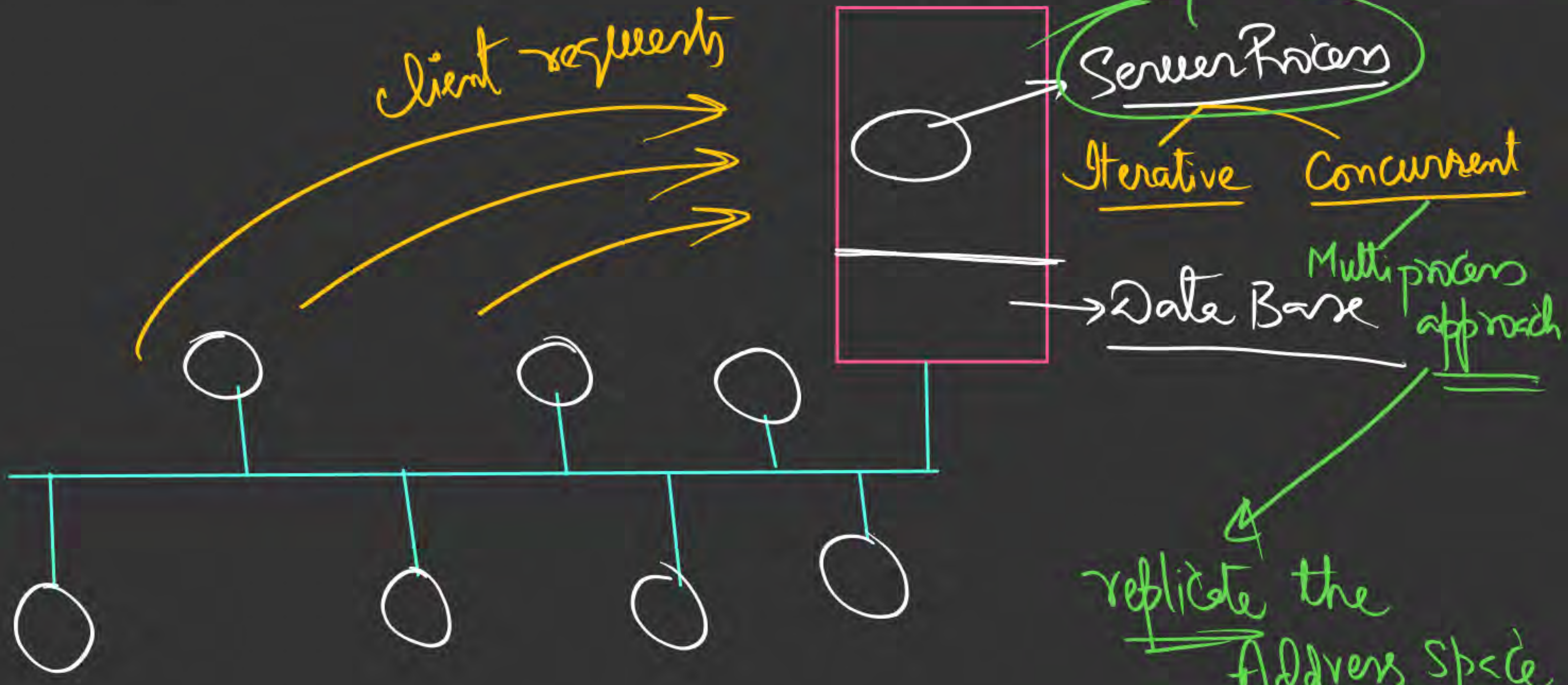
Client-Server Environment

RQ $\boxed{r_1\ r_2\ \ldots\ r_n}$

Round Robin

fork : n child Process

Server



replicate the Address space

Benefits :

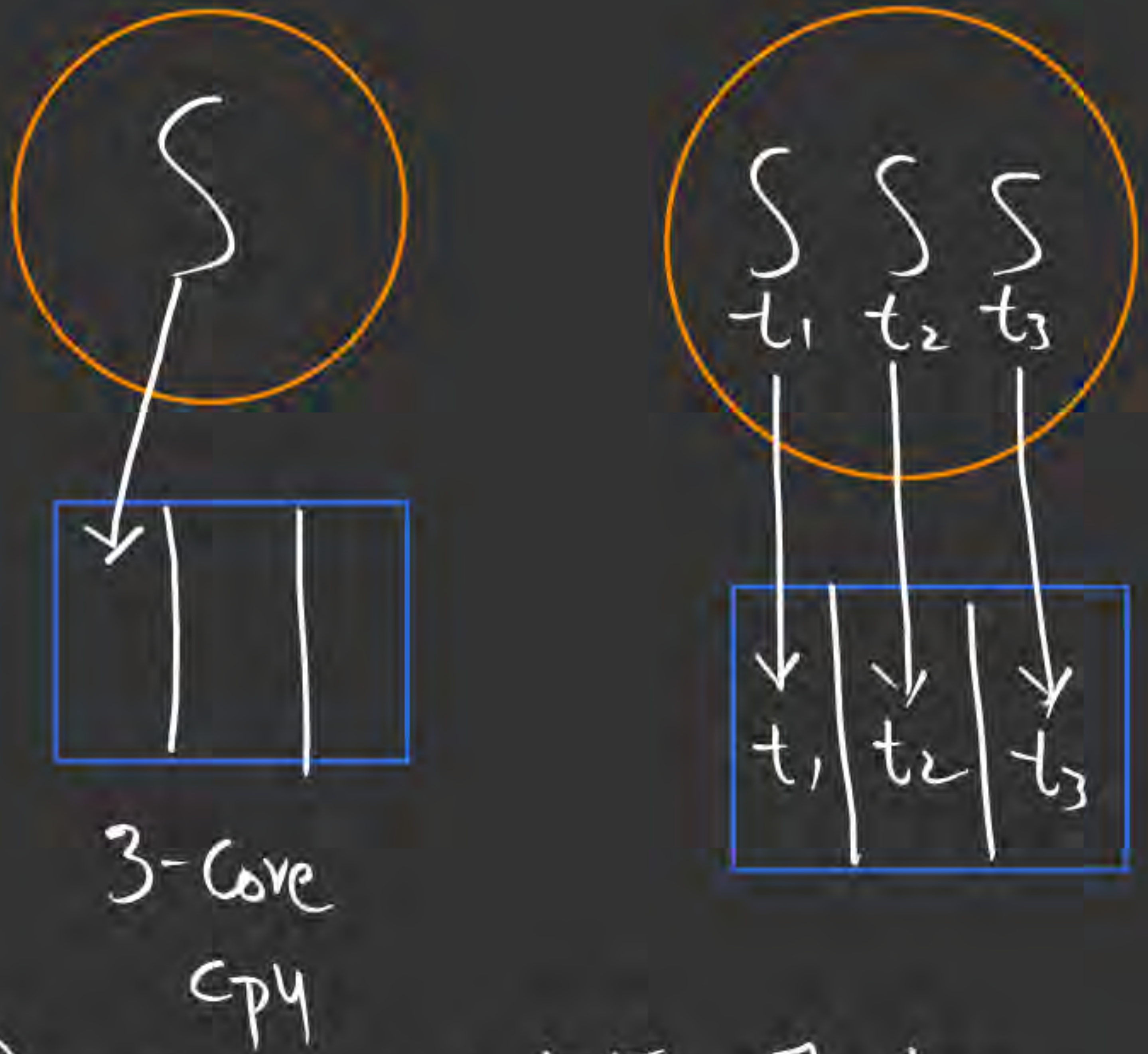
1) Resource Sharing
(Address space)

2) Economical design

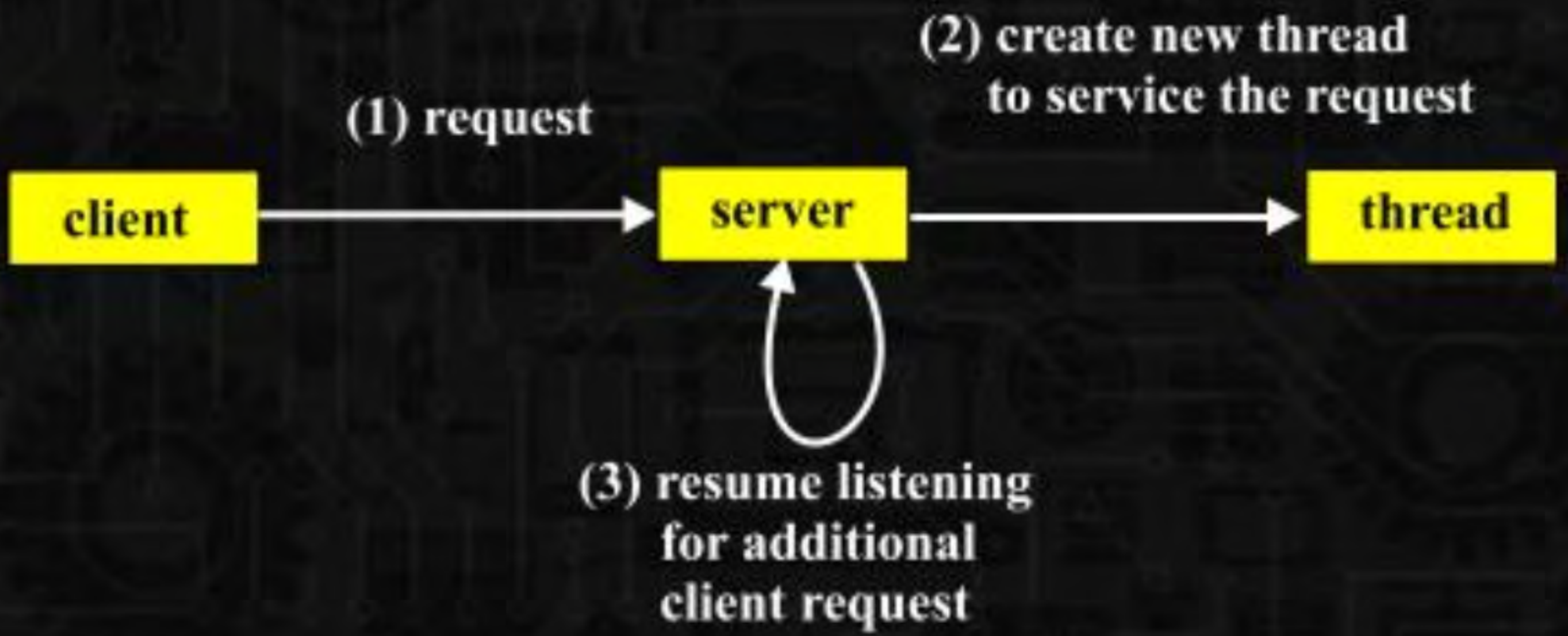
3) Sizeof (TCB) < Sizeof (PCB)

4) less Context-switch ovhd
(Thread-switching is faster)

5) Can exploit
Multi-Core Architecture



6) Make computation Faster


(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.

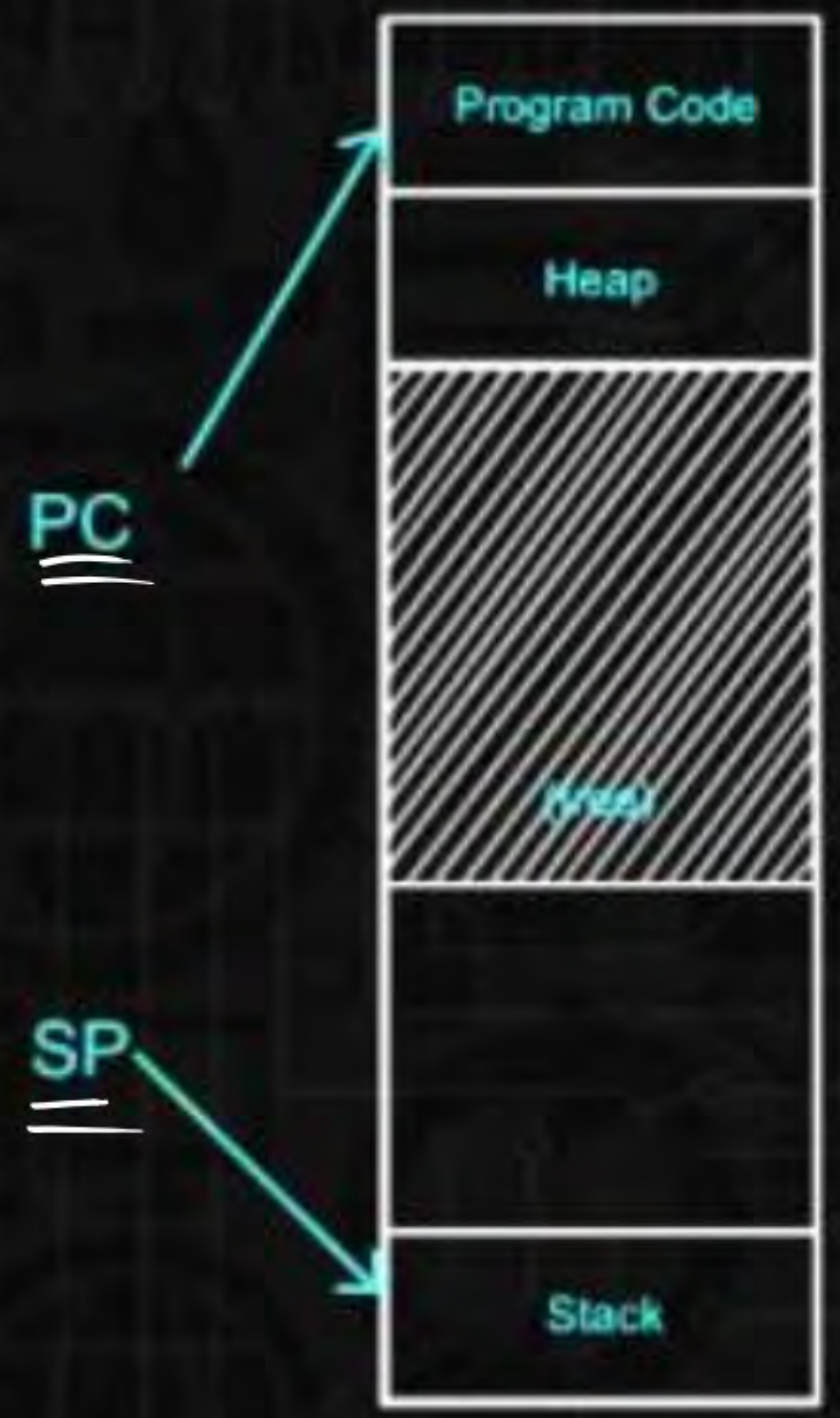
2. Resource sharing: Processes can only share resources through techniques such as. shared memory. and. message) passing.. .Such. techniques, must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. The benefit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

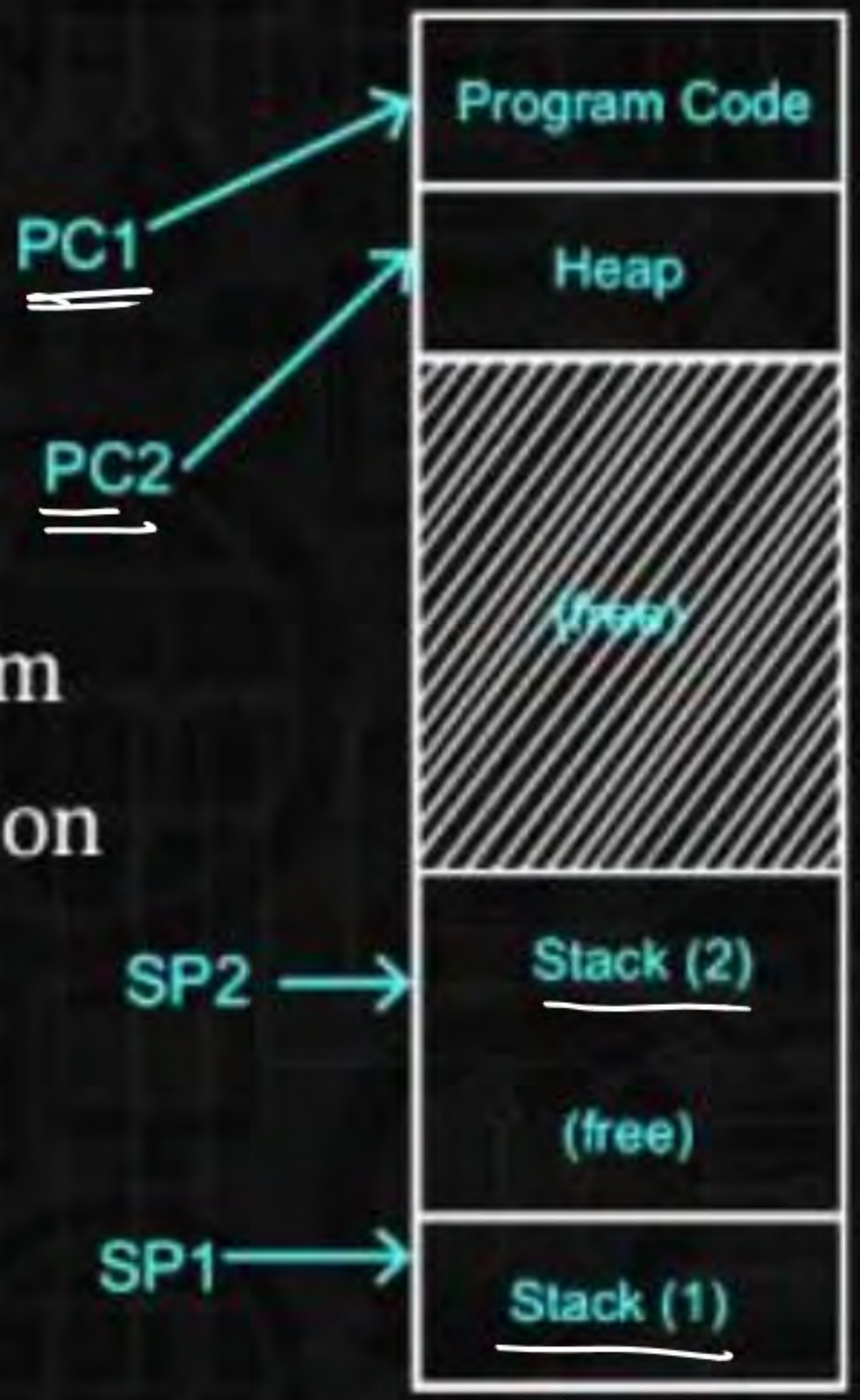
Single Threaded Process

- 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

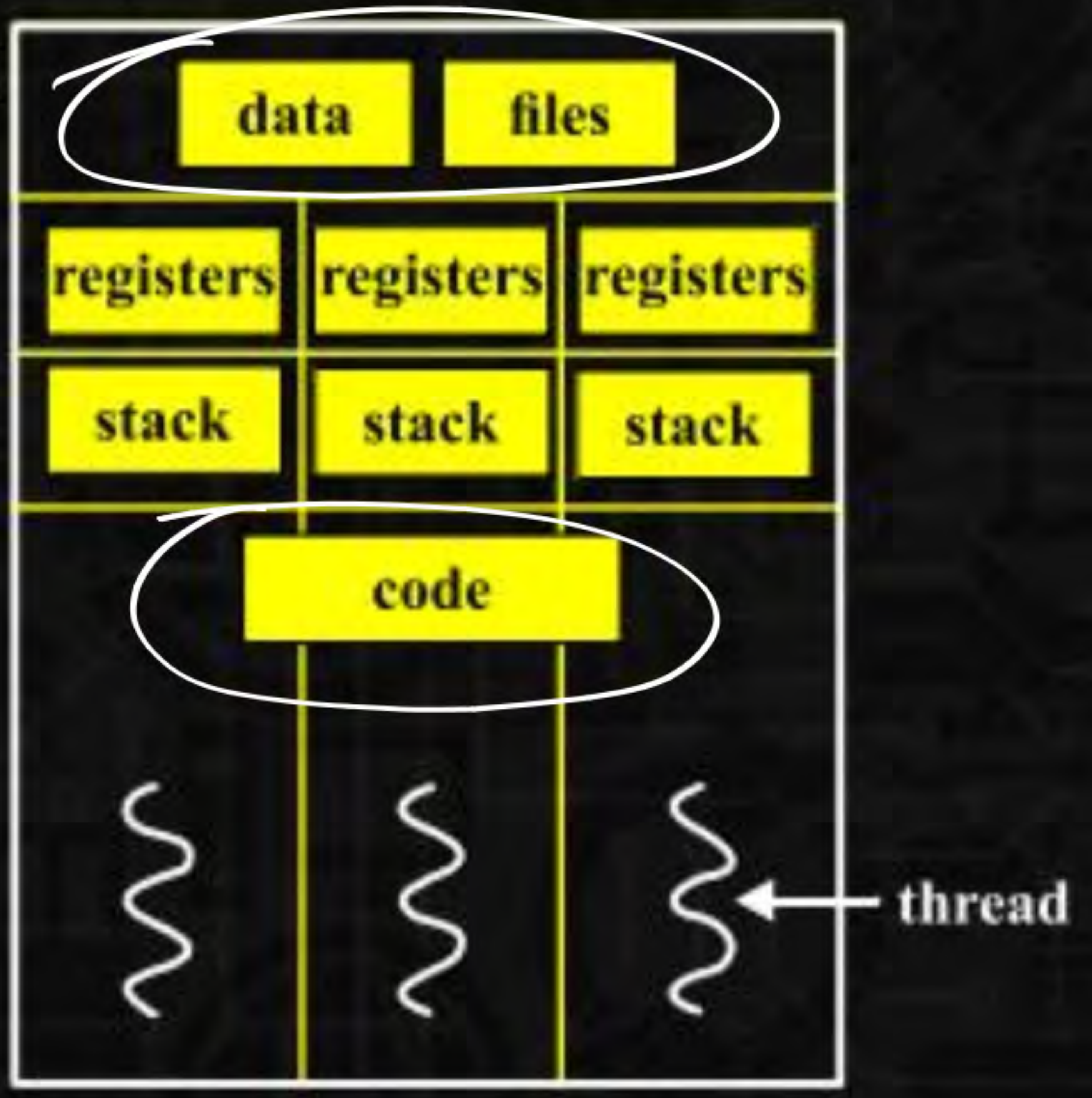


Process Vs. Thread

- 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

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



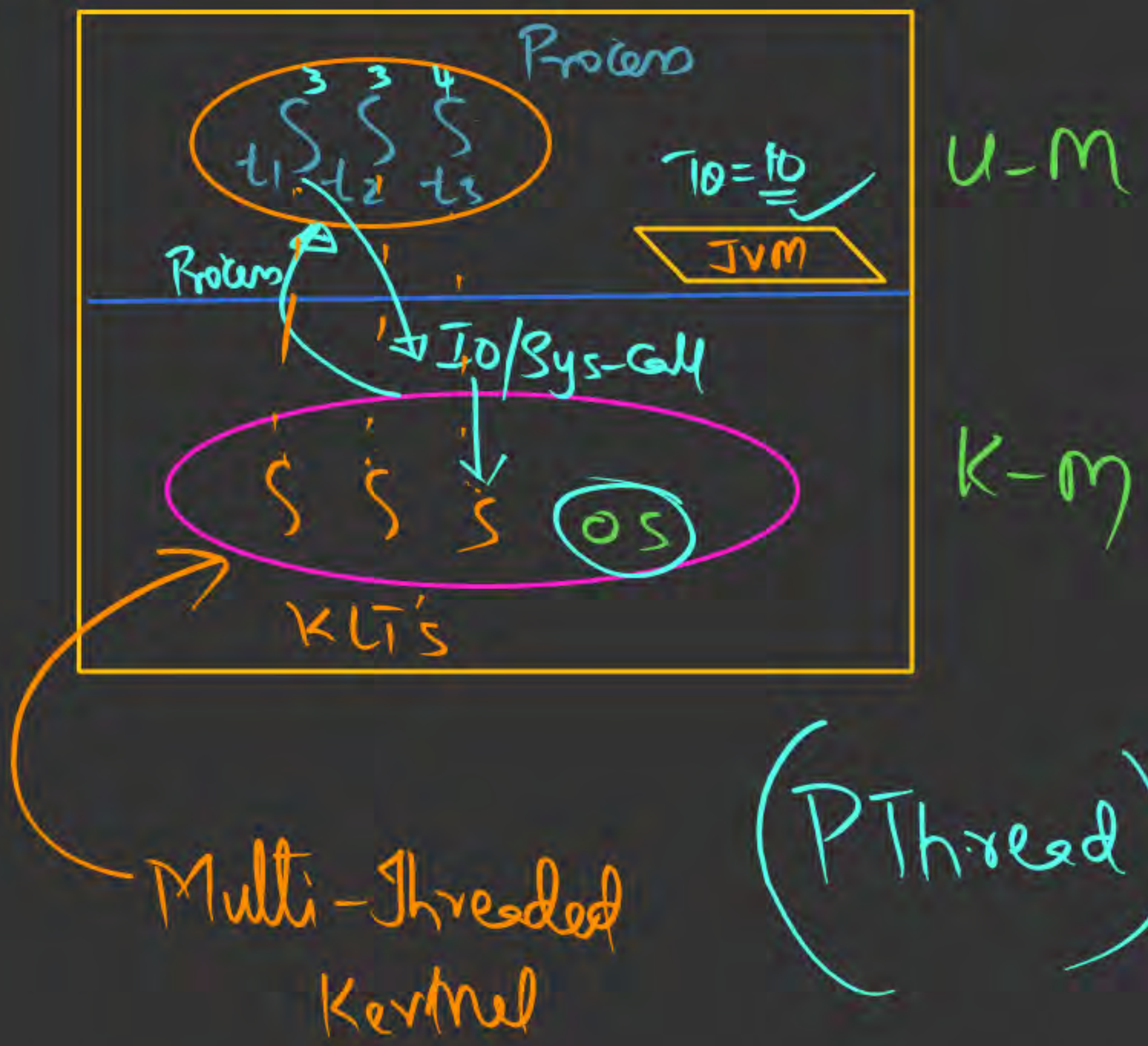
Threads Vs Processes

- 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
- A process has at-least one thread.
- ~~Heads~~ Threads within a process share the same code, files.
- If a process dies, all thread die.

Types of Threads:

User-level Threads (ULTs)

Kernel-level Threads (KLTs)



(PThread)

U.L.T's :

→ Threads that are created & Managed @ user-level without any support of O.S (Knowledge)

→ Flexibility

→ Transparency

→ Fastest Context Switching

(No Need of Mode Shifting)

→ Requirement of IO/Sys-Call exec, in a user-level Thread would result in Blocking of the whole process;

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.

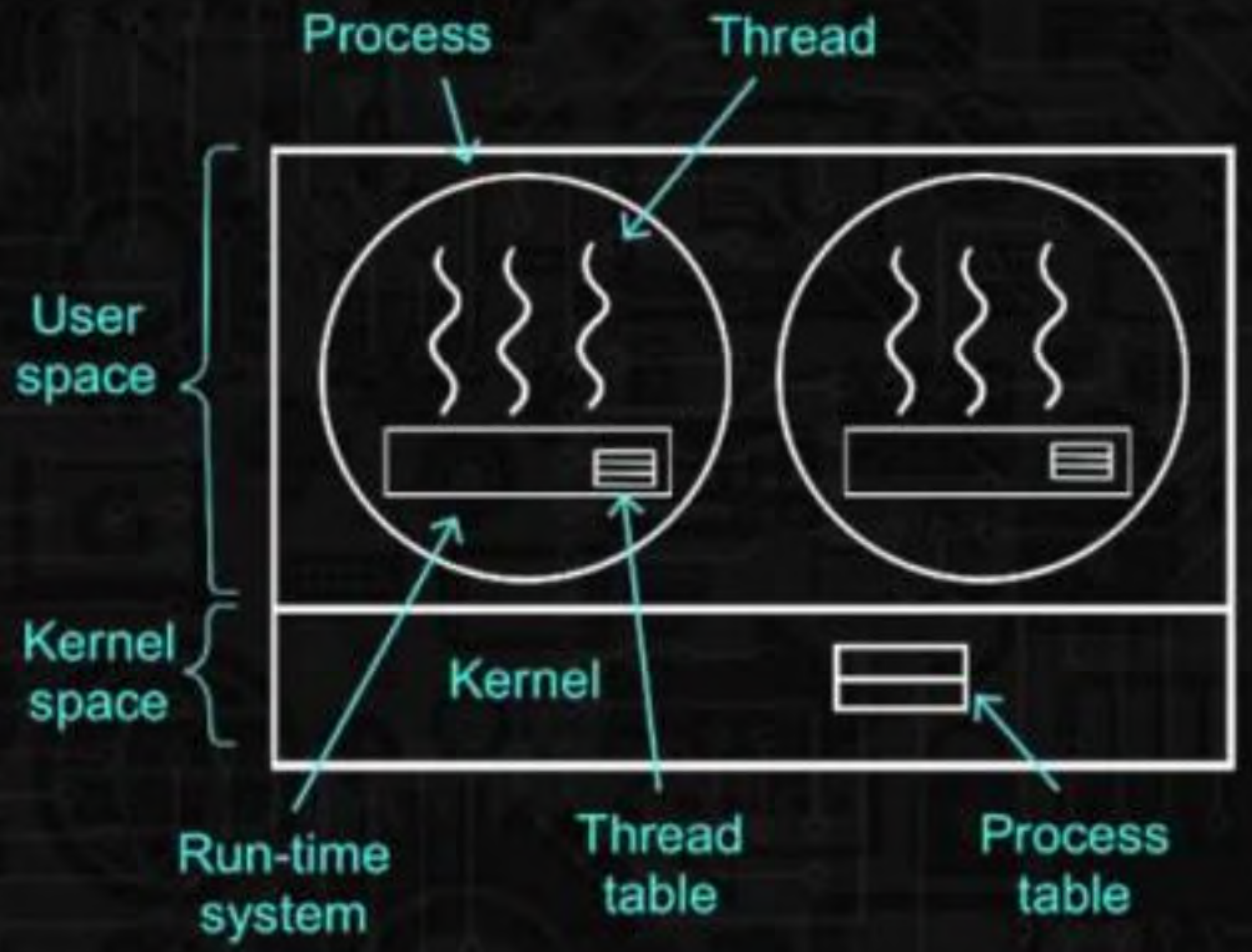
- Join : wait for a specific thread to complete
  Int pthread_join(pthread_t thread, void**retval);
  TID of the thread to wait for exit status of the thread

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



Kernel Level Threads

- Advantages:

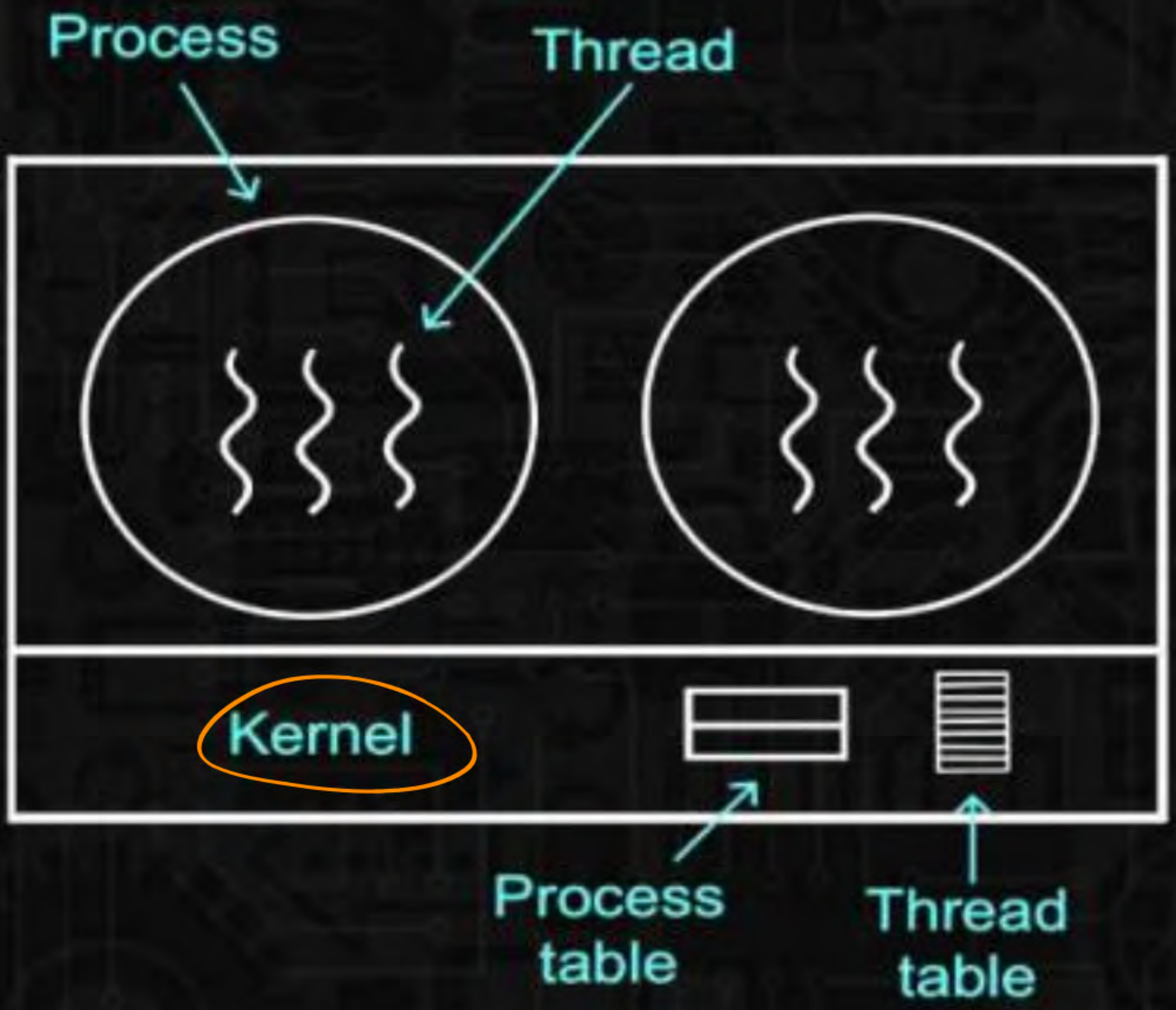
  Scheduler can decide to give more time to a process having small number of threads.

  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. F

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~~? FALSE ?

MSQ

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

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

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

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

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

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

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

Q. 6 Which one of the following is NOT shared by the threads of the same process?

A. Stack

B. Address Space

C. File Descriptor Table (Resource)

D. Message Queue (Resource)

I.P.C Mechanisms

Q. 7

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

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

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

III. T Kernel supported threads can be scheduled independently.

IV. T 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

FCFS

My official Telegram channel

https://t.me/KhaleelSirPw


THANK YOU GW
SOLDIERS !