CS & IT ENGINEERING

Operating Systems-Revision

Process Concepts

Lecture No.- 02

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

Topic Introduction & Backgrouond

Topics to be Covered

Topic Process Concepts

Topic Threads & Multithreading

Program / Job

(.exe)

- Instrn's
  - Ex: Load AC, R_1
  - Add R1, R_2
- Data
  - Static
    - → Fixed & Known Size
    - → @ Load time b/f R.T
  - Dynamic
    - → Unknown / Variable Size
    - → @ R.T alloc.

Process / Task / App

- → Prog in Execution
- → Instance of a Prog
- → Active entity
- → Locus of control
- → unit of cpu utiliz
- → Animated Spirit

Topic : Process Concept

- An operating system executes a variety of programs that run as a process.
- Process – a program in execution; process execution must progress in sequential fashion. No parallel execution of instructions of a single process
- Multiple parts
  - The program code, also called text section
  - Current activity including program counter, processor registers
  - Stack containing temporary data
    - Function parameters, return addresses, local variables
  - Data section containing global variables
  - Heap containing memory dynamically allocated during run time

Topic : Process Concept (Cont.)

- Program is passive entity stored on disk (executable file); process is active
  - Program becomes process when an executable file is loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc.
- One program can be several processes (fork() system call)
  - Consider multiple users executing the same program

Topic : Process in Memory

Developer's view of Process

↓

A.D.T

<Defn, Repr Structure ; operations, Attributes>

(Activation Records of function calls)

Dynamic Alloc.

Code | Instr's

Topic : Memory Layout of a C Program


```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

Process Attributes:

→ Pid, PPid, gid,
PC; G.P.R
mem. limits
Priority; State,

Process Context
Environment

OS D.S

K.M by OS processes

P.C.B (Process Control Block)
Process Descriptor

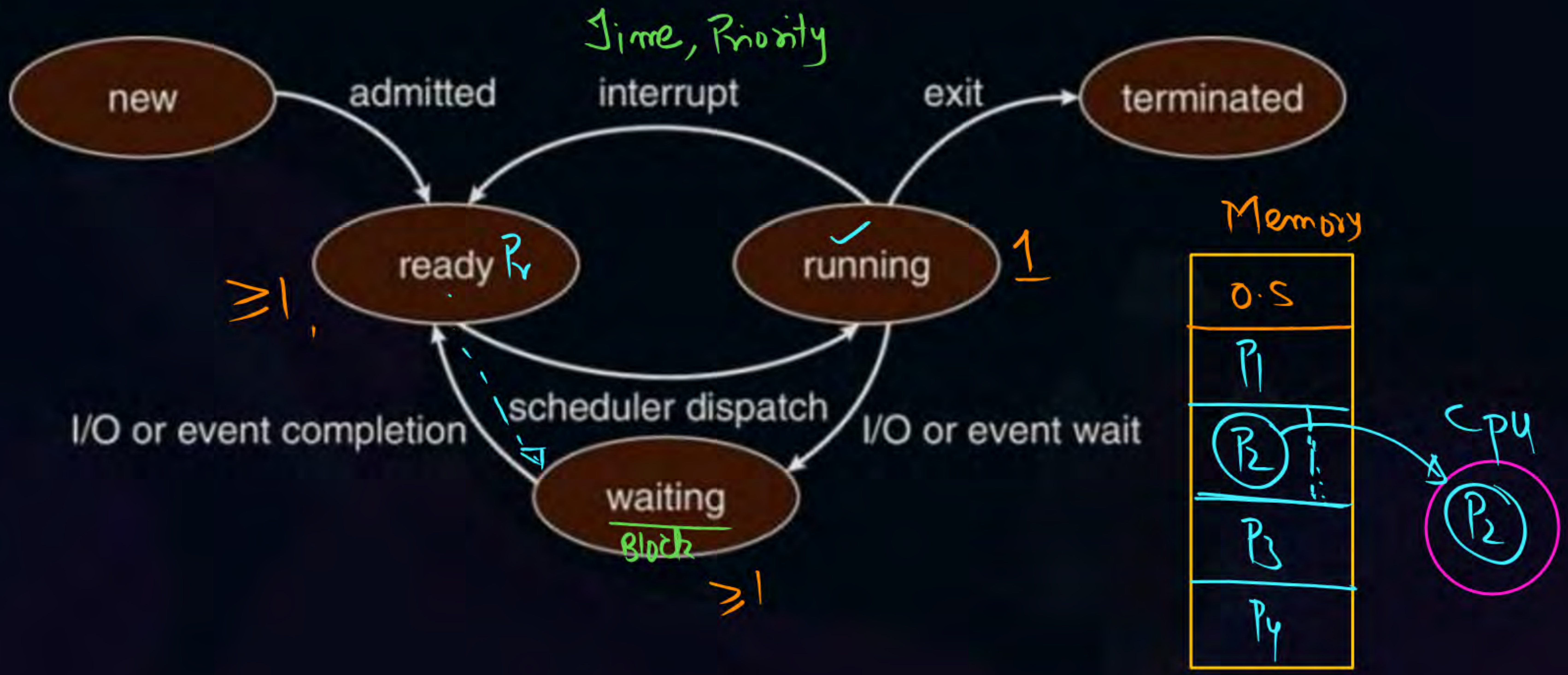
Topic : Process State

- As a process executes, it changes state
  - New: The process is being created
  - Running: Instructions are being executed on cpu
  - Waiting: The process is waiting for some event to occur (IO, Interrup (event))
  - Ready: The process is waiting to be assigned to a processor (Ready Q)
  - Terminated: The process has finished execution

Block

Topic : Diagram of Process State

M.Pr < N.Pr, Pr



Topic : Process Control Block (PCB)

Information associated with each process(also called task control block)

- Process state – running, waiting, etc.
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

Topic : Process Scheduling

- Process scheduler selects among available processes for next execution on CPU core
- Goal -- Maximize CPU use, quickly switch processes onto CPU core
- Maintains scheduling queues of processes
- Ready queue – set of all processes residing in main memory, ready and waiting to execute
- Wait queues – set of processes waiting for an event (i.e., I/O)
- Processes migrate among the various queues

Scheduling Q's & Queueing diagram

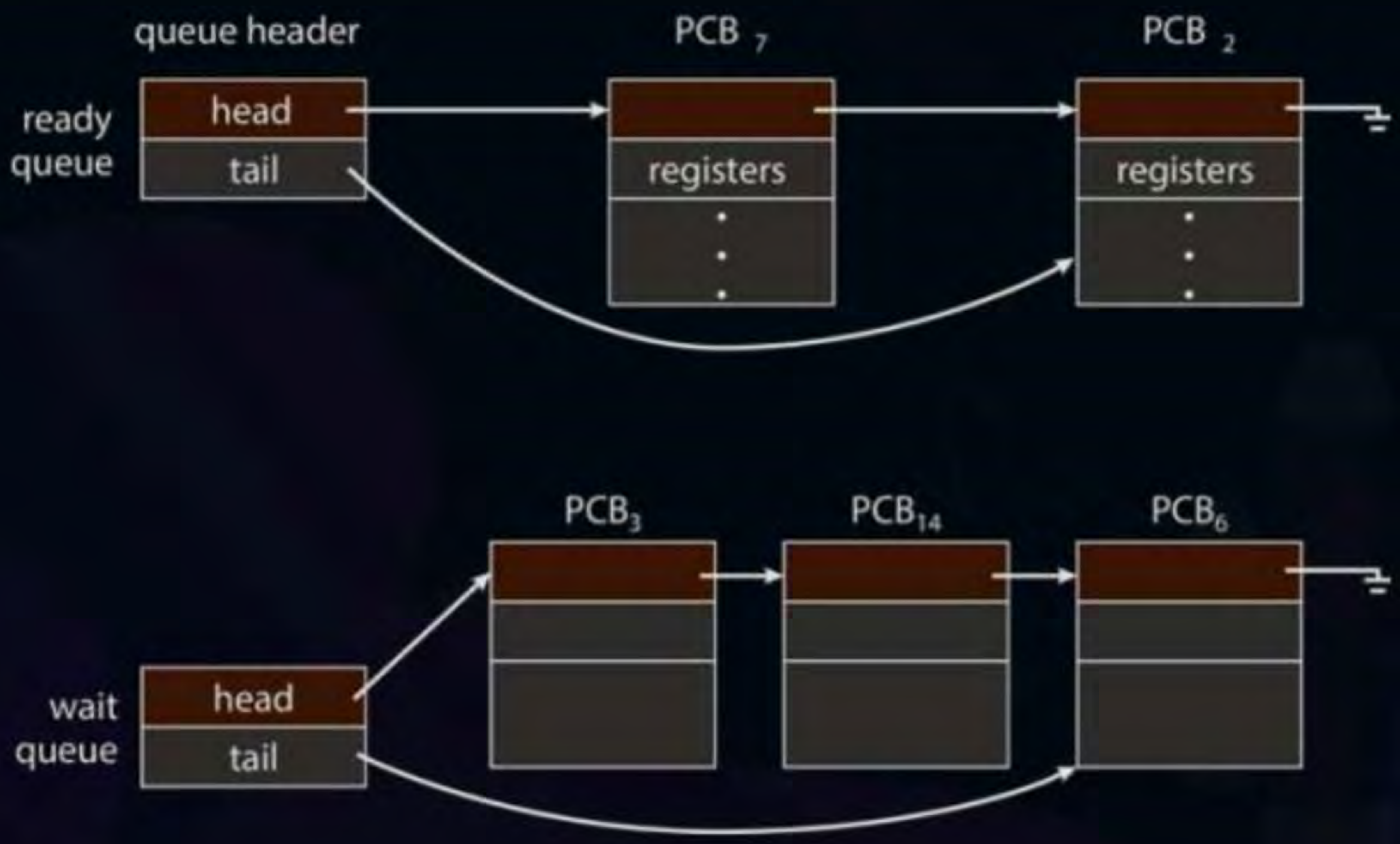
- In Memory
  - Ready Q
  - Block Q
    <IO devices>
- on disk
  - Job-Q / IP-Q
  - Suspend-Q

$\rightarrow \boxed{P_i \; P_j \; P_k \cdots P_n}$

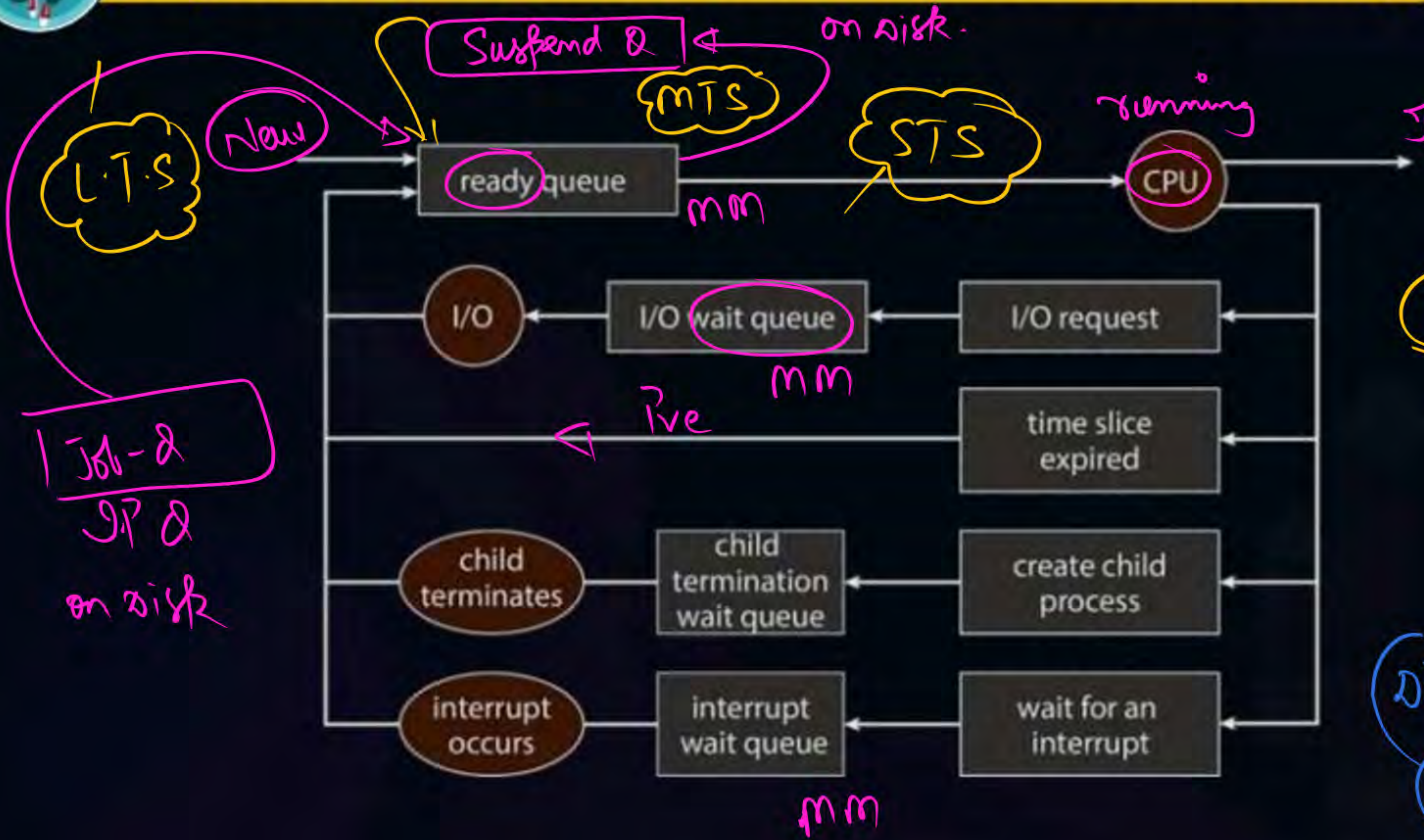
RQ

Linked list

Topic : Ready and Wait Queues



Topic : Representation of Process Scheduling



Terminate

Control degree of M.Pr.

L.T.S

S.T.S (CPU Schd.)

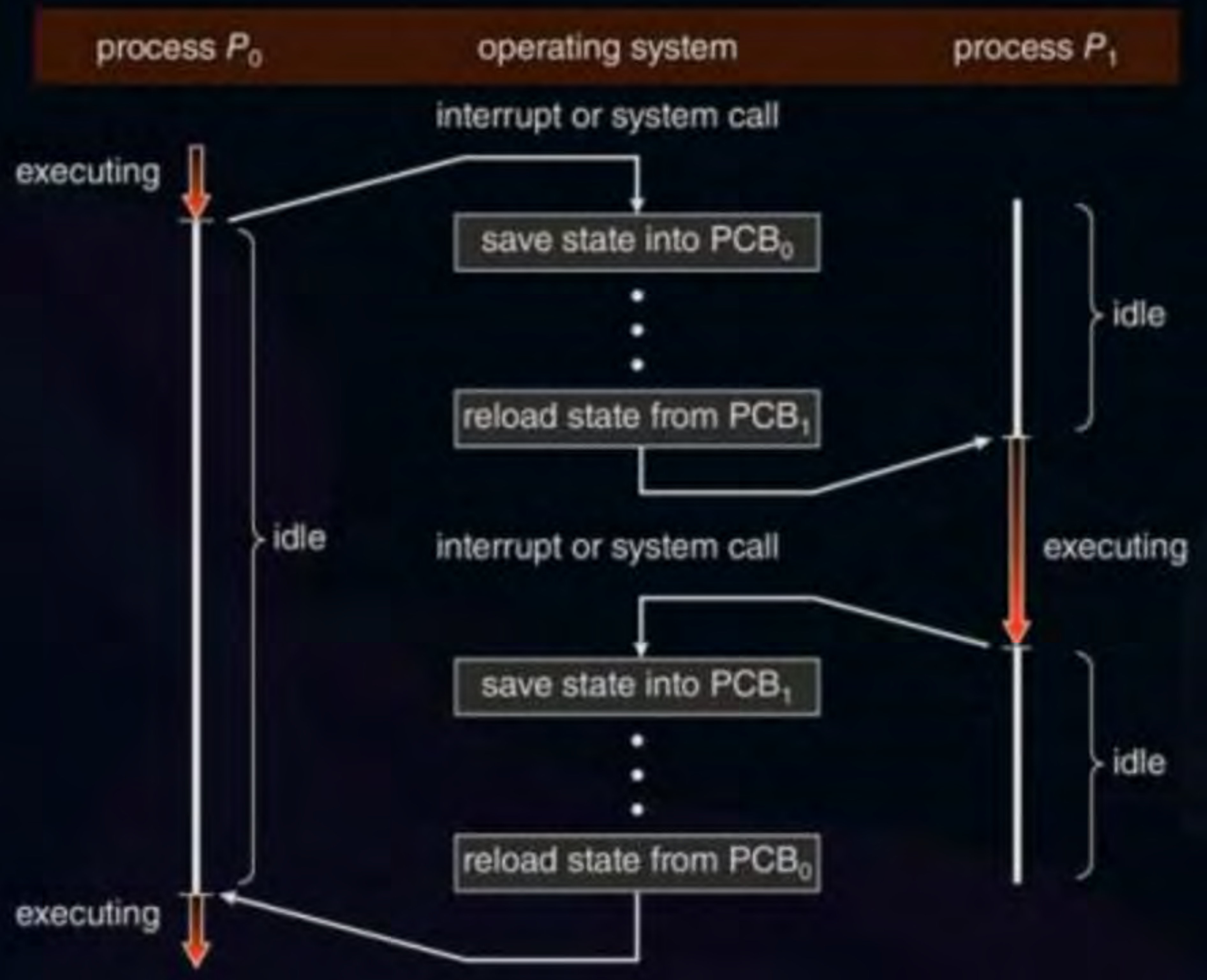
M.T.S (Swapper)

Dispatcher

Context switching

Topic : CPU Switch From Process to Process

- A context switch occurs when the CPU switches from one process to another.



Topic : Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is pure overhead; the system does no useful work while switching
- The more complex the OS and the PCB □ the longer the context switch
- Time dependent on hardware support
- Some hardware provides multiple sets of registers per CPU □ multiple contexts loaded at once

Topic : Operations on Processes

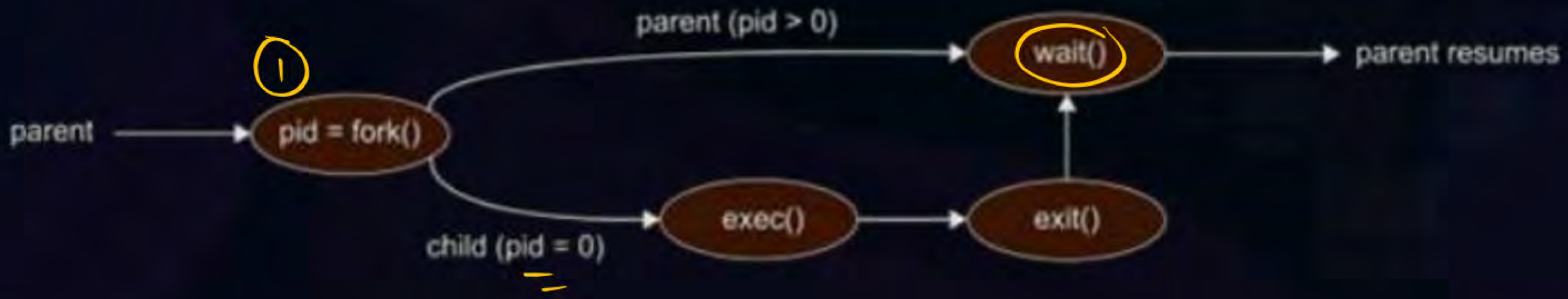
- System must provide mechanisms for:
- Process creation : fork
- Process termination : exit

Topic : Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources
- Execution options
- Parent and children execute concurrently
- Parent waits until children terminate

Topic : Process Creation (Cont.)

- Address space
- Child duplicate of parent
- Child has a program loaded into it
- UNIX examples
- fork() system call creates new process
- exec() system call used after a fork() to replace the process' memory space with a new program
- Parent process calls wait()waiting for the child to terminate



Topic : A Tree of Processes in Linux

Daemon (Continuously running Process as a Server)

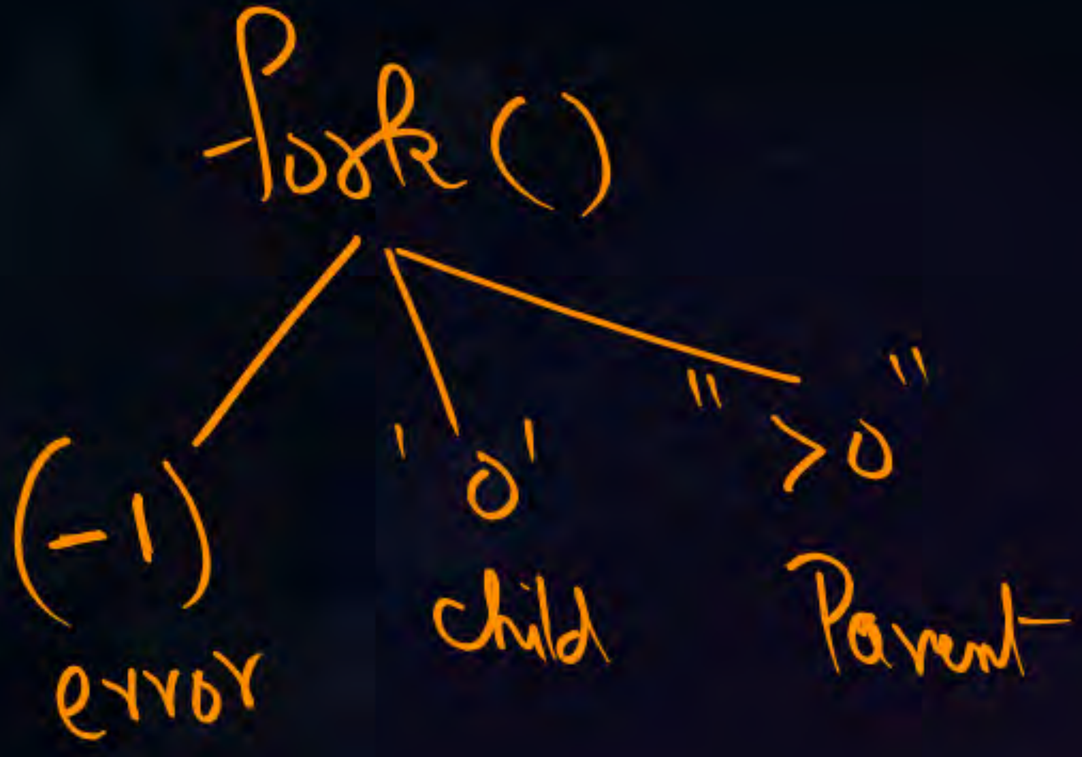
Topic : C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{ pid t pid;
        /* fork a child process */
        pid = fork() ;

if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
}
else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
}
else { /* parent process */
        /* parent will wait for the child to
complete */
        wait (NULL);
        printf("Child Complete");
}
return 0;
}
```

Trivial

Parent waits for child to complete



Topic : Process Termination

- Process executes last statement and then asks the operating system to delete it using the exit() system call.
- Returns status data from child to parent (via wait())
- Process’ resources are deallocated by operating system
- Parent may terminate the execution of children processes using the abort() system call. Some reasons for doing so:
- Child has exceeded allocated resources
- Task assigned to child is no longer required
- The parent is exiting, and the operating systems does not allow a child to continue if its parent terminates

Topic : Process Termination

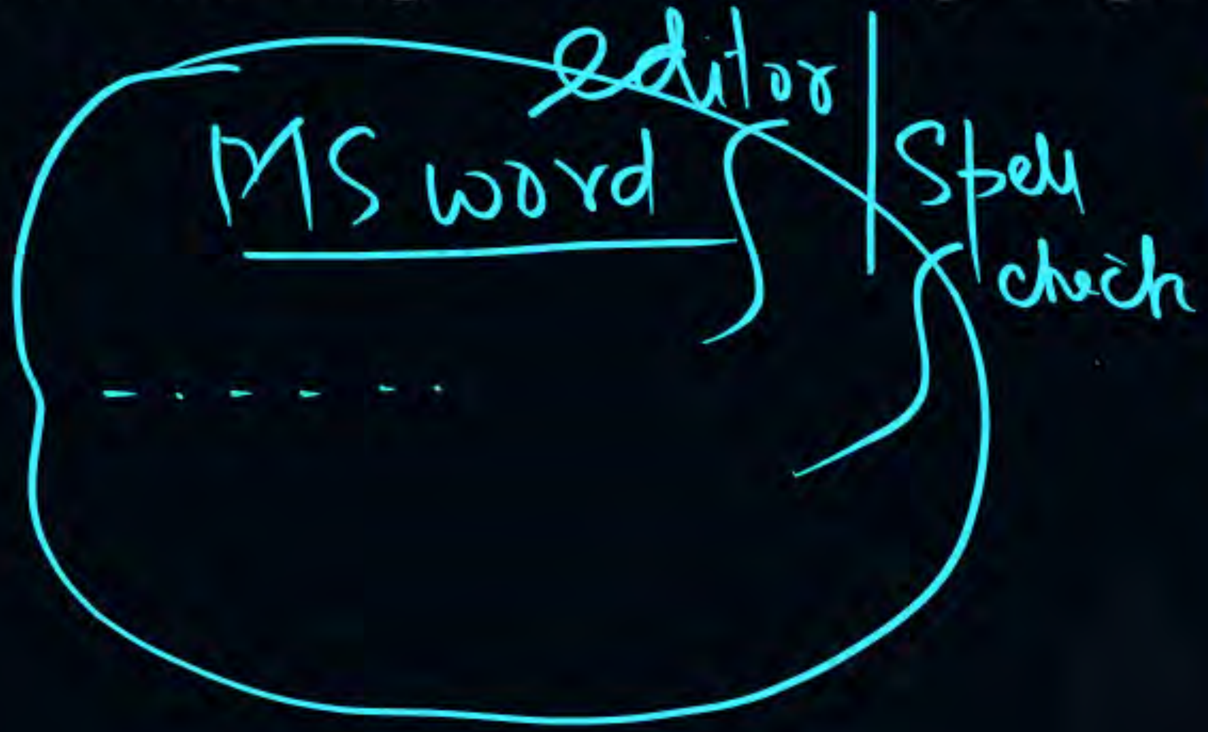
- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
- cascading termination. All children, grandchildren, etc., are terminated.
- The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the wait()system call. The call returns status information and the pid of the terminated process
- pid = wait(&status);
- If no parent waiting (did not invoke wait()) process is a zombie
- If parent terminated without invoking wait(), process is an orphan

Topic : Threads & Multithreading

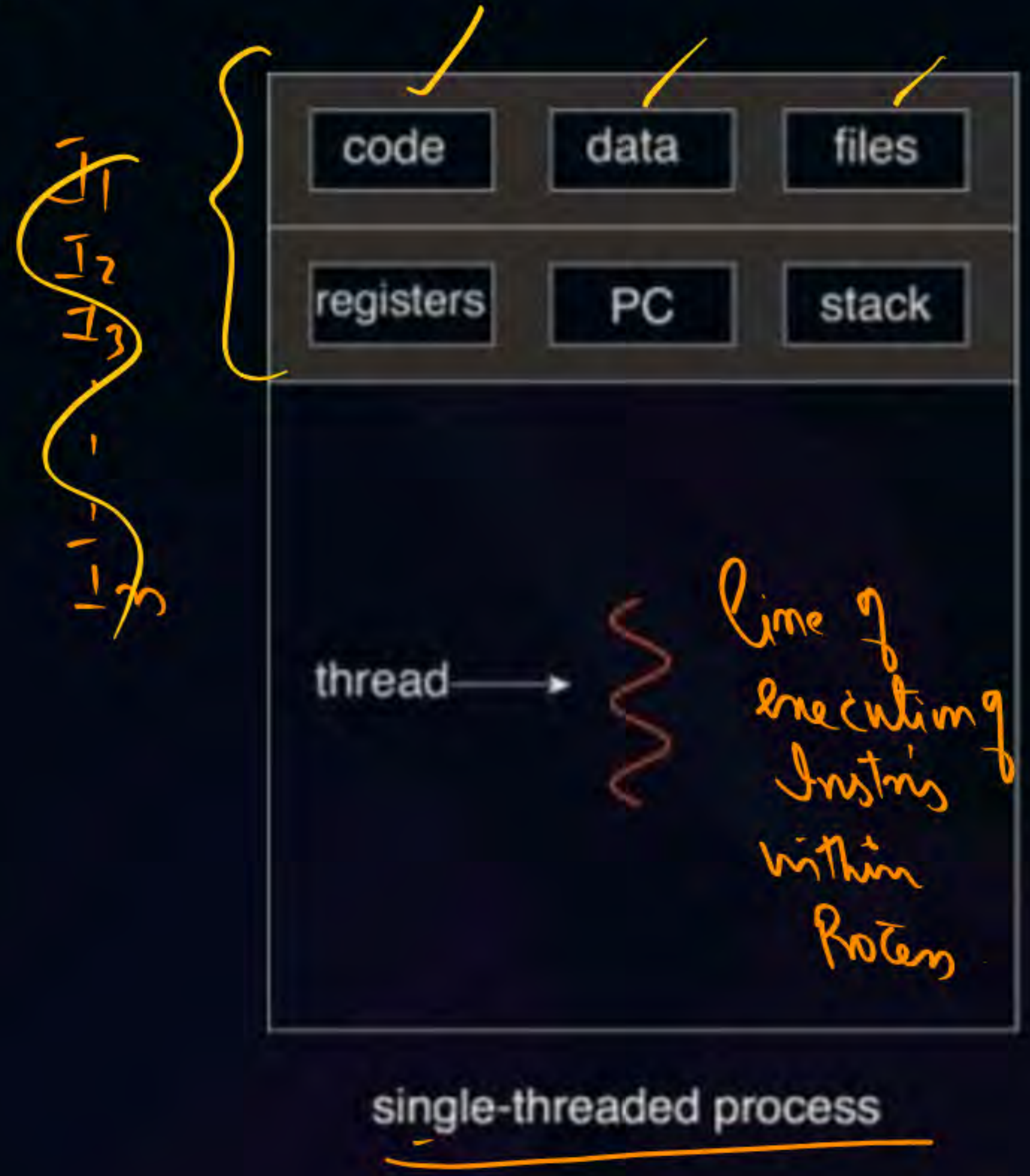
Multi Core Architectures

i3; i5; i7; ...

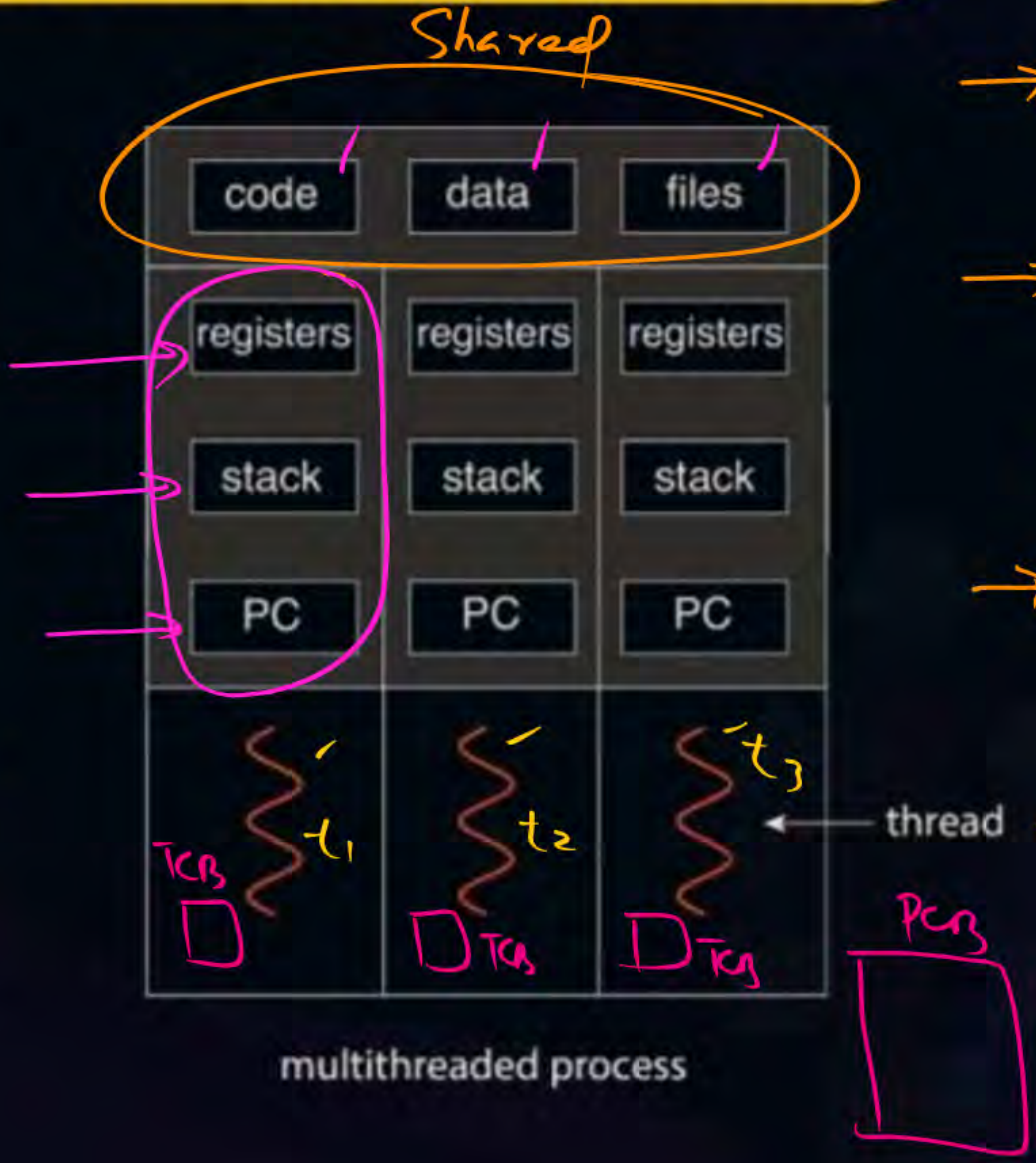
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded



Topic : Single and Multithreaded Processes



single-threaded process



multithreaded process

→ Thread is a L.W.P

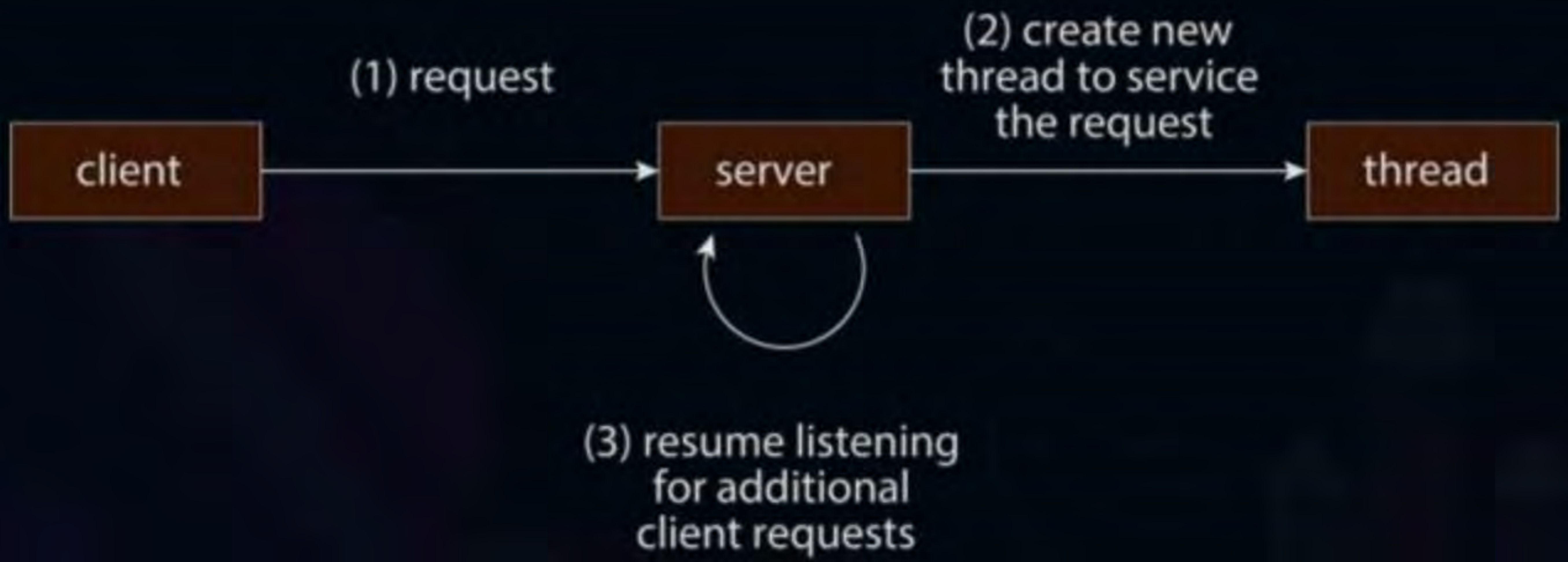
→ Part of Process

SSS

→ unit of cpu utilization

→ Thread can have its own context (TCB)

Topic : Multithreaded Server Architecture



Topic : Benefits

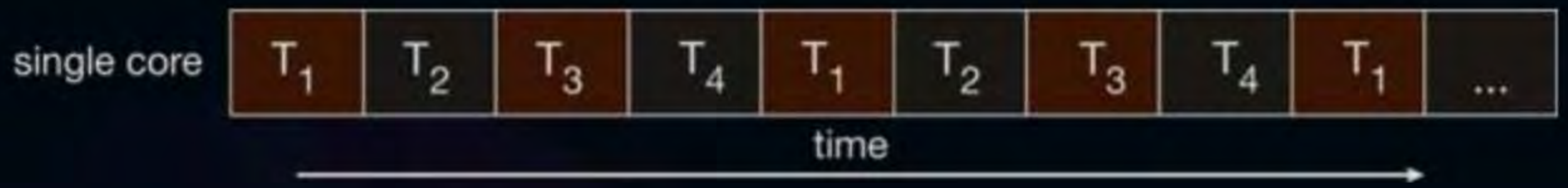
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multicore architectures.

Topic : Multicore Programming

- Multicore or multiprocessor systems puts pressure on programmers, challenges include:
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging
- **Parallelism** implies a system can perform more than one task simultaneously
- **Concurrency** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Topic : Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**

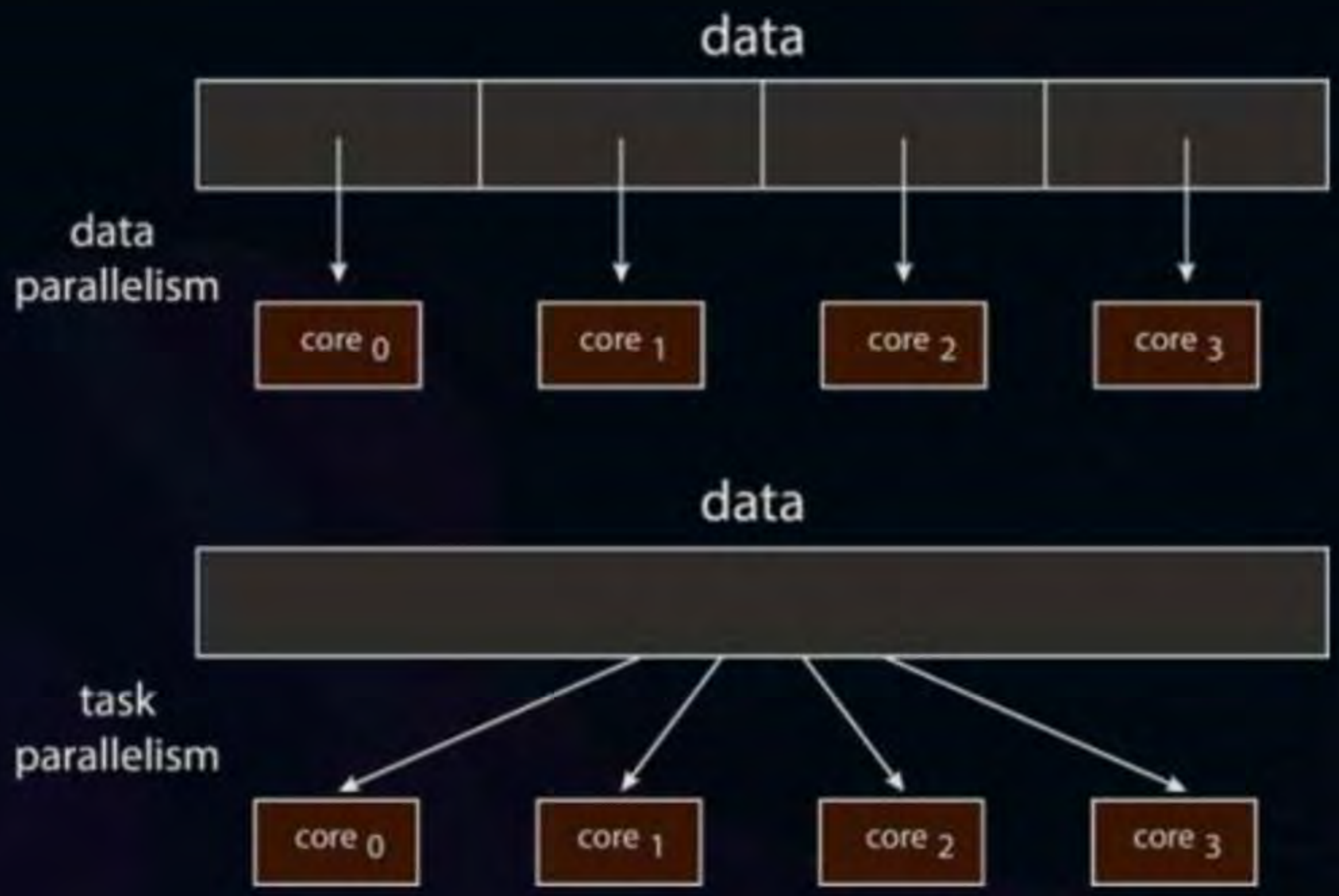


- **Parallelism on a multi-core system:**

Topic : Multicore Programming

- Types of parallelism
  - **Data parallelism –** distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism –** distributing threads across cores, each thread performing unique operation

Topic : Data and Task Parallelism



Topic : User Threads and Kernel Threads

- **User threads -** management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Windows threads
  - Java threads
- **Kernel threads -** Supported by the Kernel
- Examples – virtually all general-purpose operating systems, including:
  - Windows
  - Linux
  - Mac OS X
  - iOS
  - Android

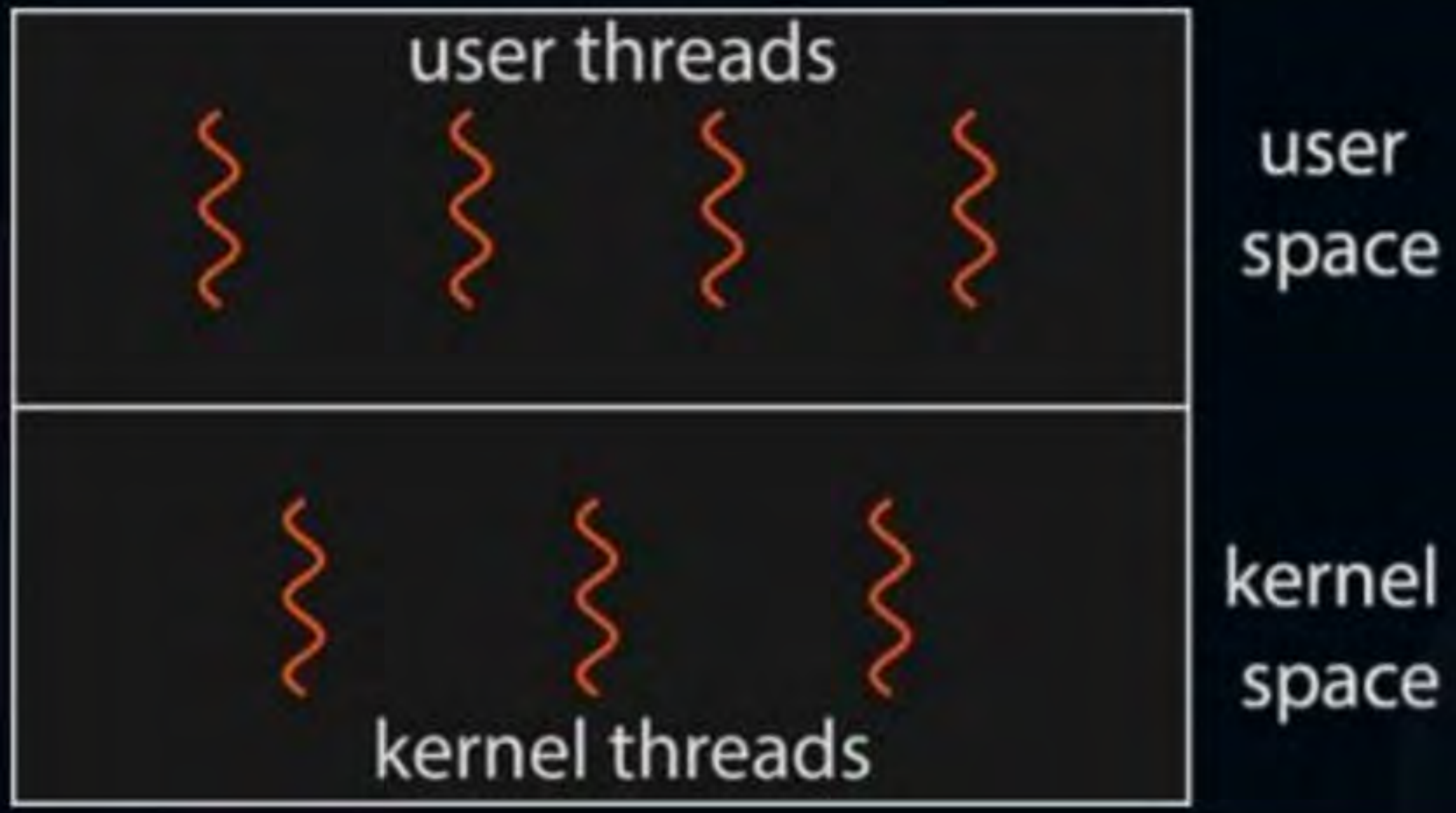
Benefits of U.L.T's

1) Flexibility
2) Transparency
3) Fastest CS
4) Improved Perf.

Drawback: of ULT

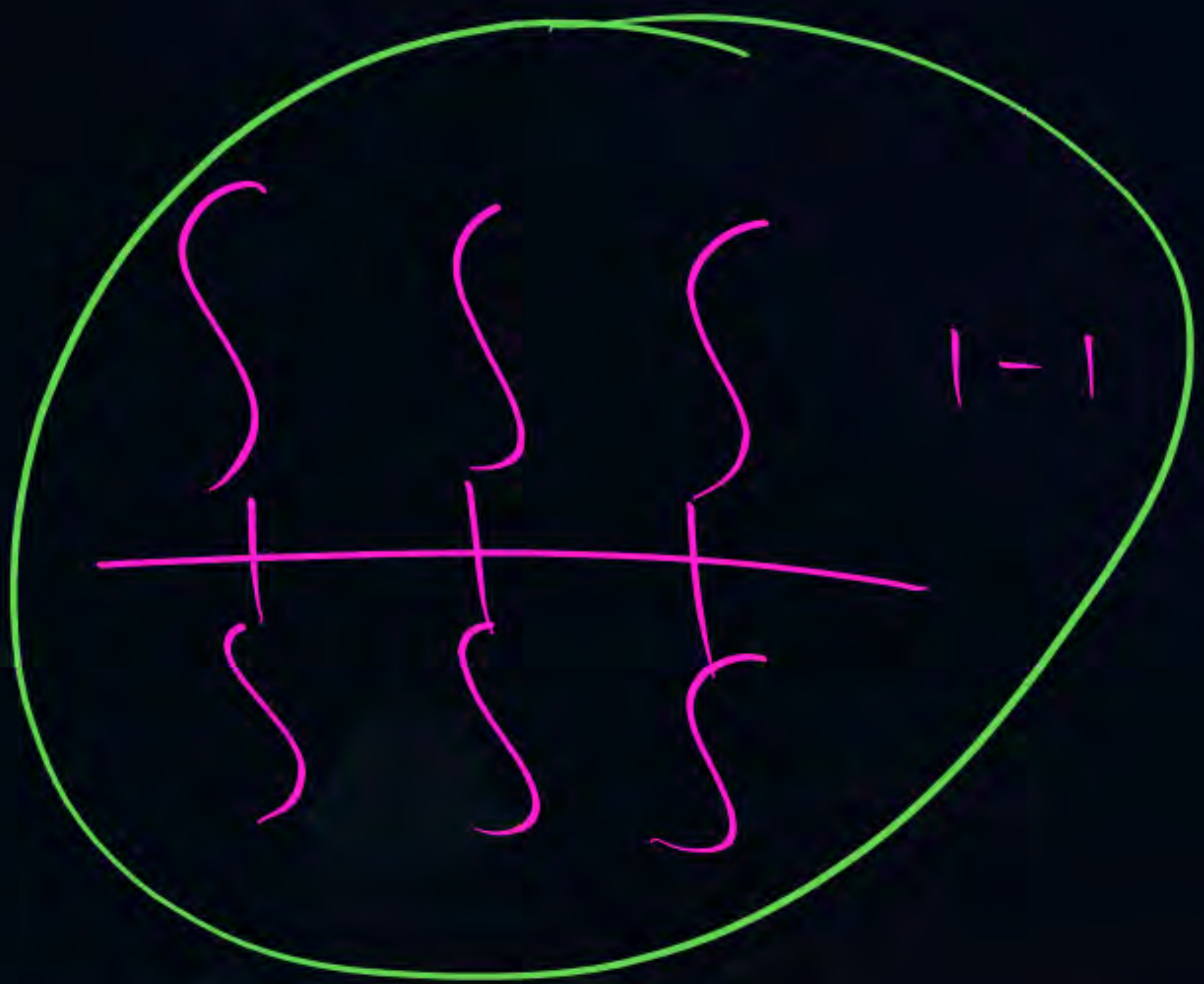
→ requirement of IO; Sys Call by one ULT will result in blocking of whole Process

Topic : User and Kernel Threads



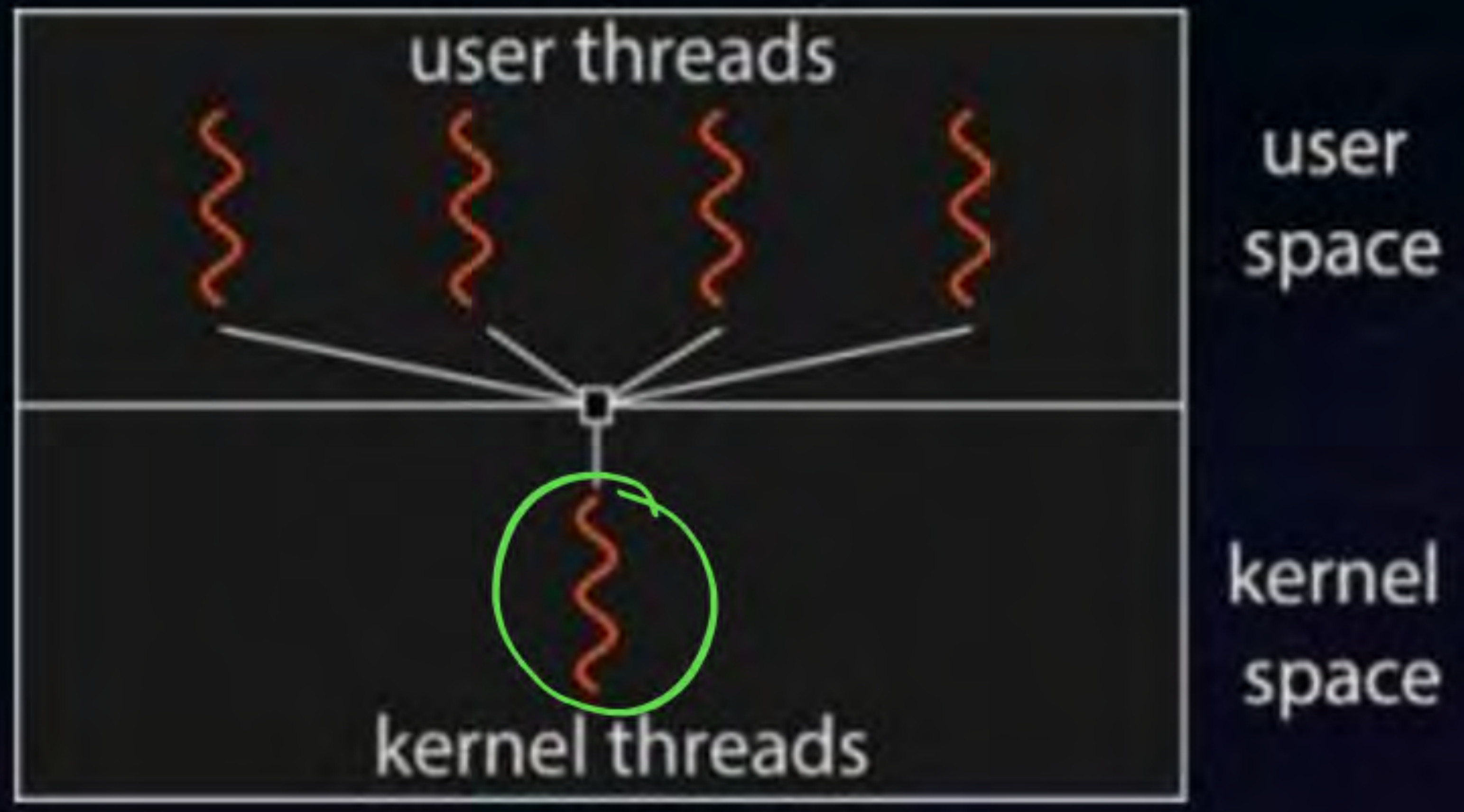
Topic : Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many



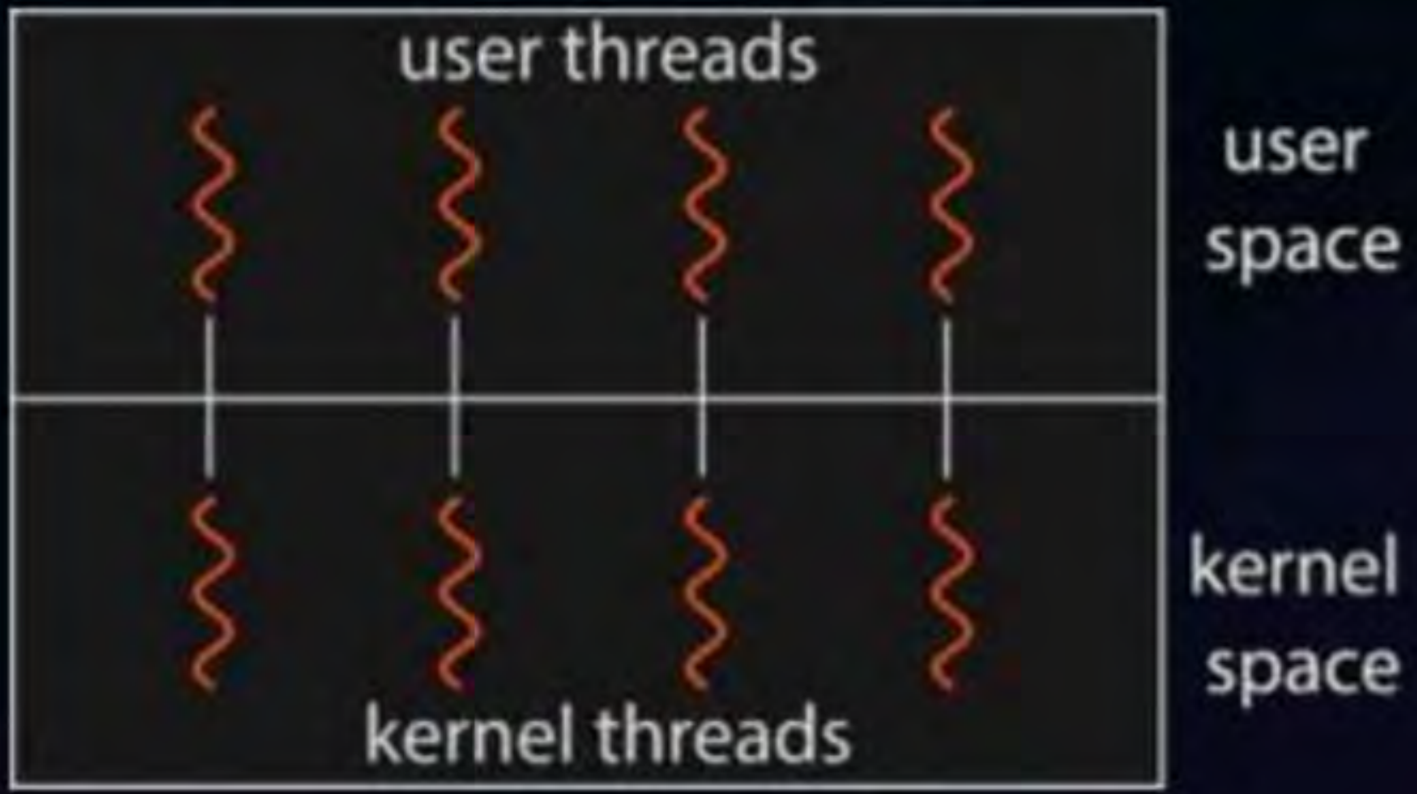
Topic : Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



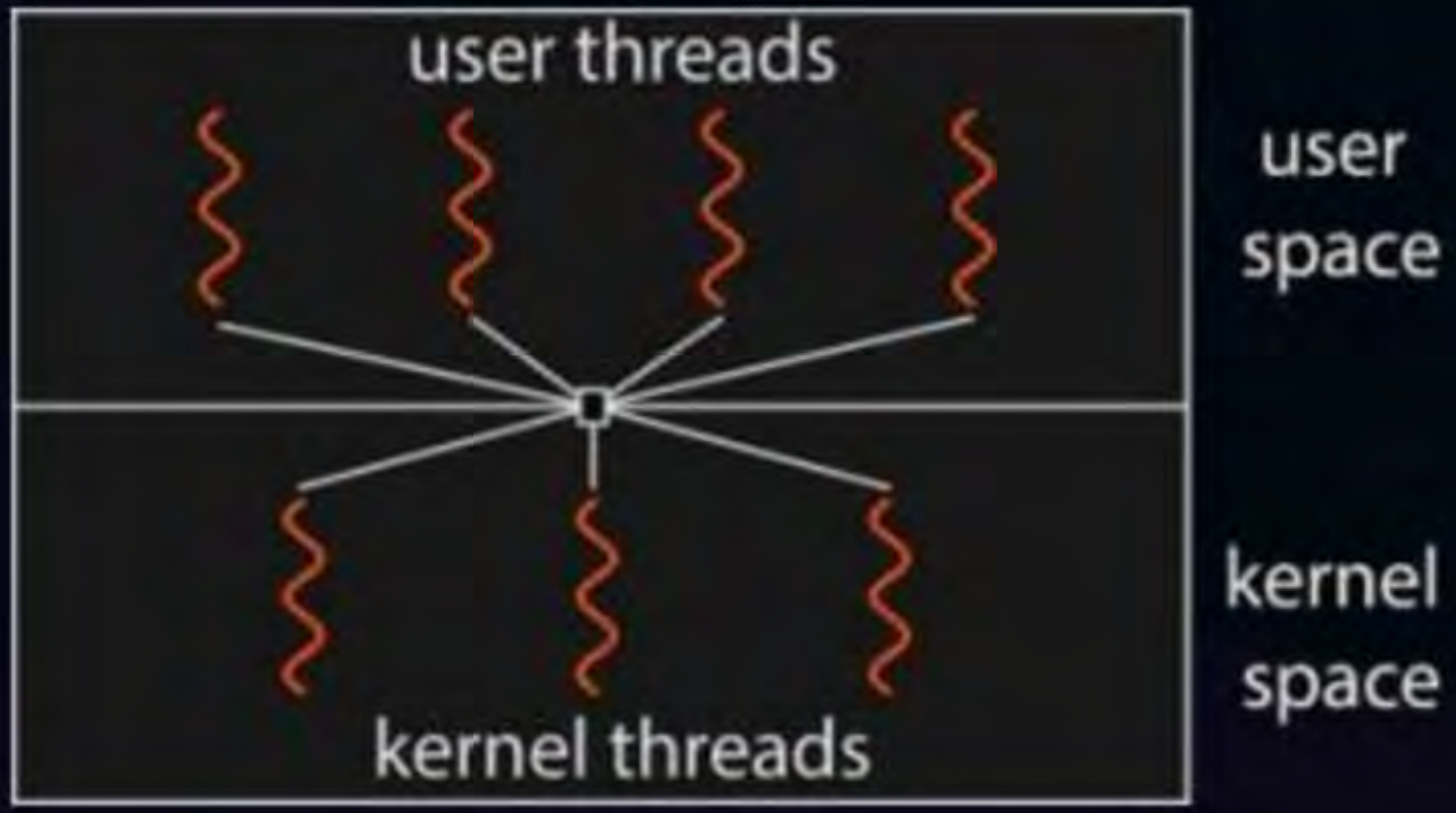
Topic : One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux



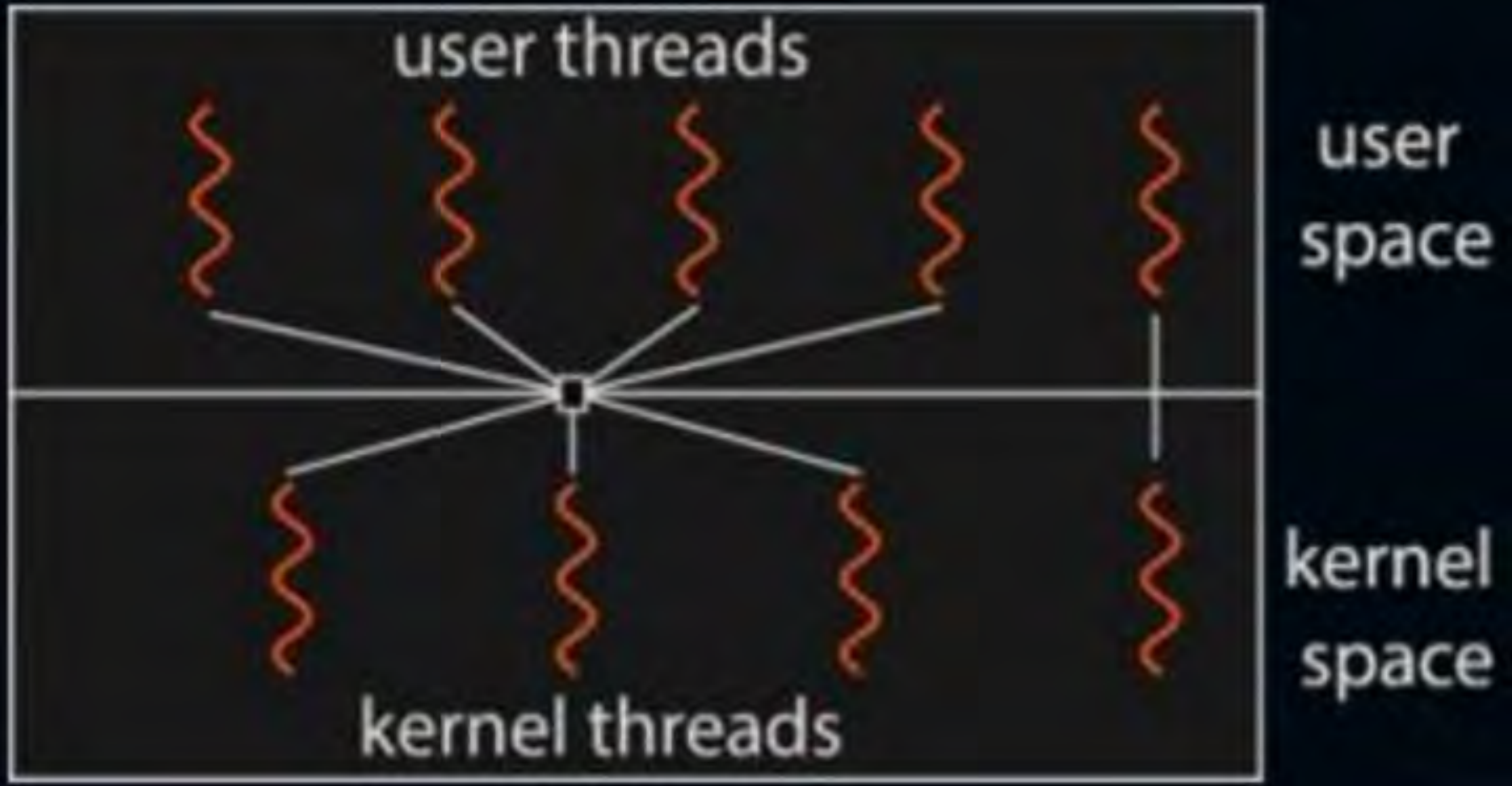
Topic : Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the ThreadFiber package
- Otherwise not very common



Topic : Two-level Model

- Similar to M:M, except that it allows a user thread to be bound to kernel thread



Topic : Thread Libraries

- Thread library provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Topic : Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- Specification, not implementation
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Linux & Mac OS X)

Topic : Pthreads Example

```
#include <pthread.h>
#include <stdio.h>

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner (void *param); /* threads call this function */

int main(int argc, char *argv[])
{
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of thread attributes */

    /* set the default attributes of the thread */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid, &attr, runner, argv[1]);
    /* wait for the thread to exit */
```

Topic : Pthreads Example (Cont.)

```
    pthread join (tid, NULL);

    printf("sum = %d\n",sum);
}
/* The thread will execute in this function */
void *runner (void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0);
}
```

Topic : Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

```
Public interface Runnable
{
    public abstract void run();
}
```

- The Executor is used as follows:

Topic : Java Threads

- **Implementing Runnable interface:**

```
class Task implements Runnable
{
        public void run() {
                System.out.println("I am a thread.");
        }
}
```

- **Creating a thread:**

```
Thread worker = new Thread (new Task());
Worker.start();
```

- **Waiting on a thread:**

```
Try {
    Worker.join();
}
Catch (InterruptedExeuction ie.)
```

Topic : Java Executor Framework

- Rather than explicitly creating threads, Java also allows thread creation around the Executor interface:

```
public interface Execuor
{
    Void execute (Runnable command);
}
```

- The Executor is used as follows:

```
    Executor service = new Executor;
    Service.execute(new Task());
```

Topic : Java Executor Framework

```
import java.util.concurrent.*;

class Summation implements Callable<Integer> {
      private int upper;
      public Summation (int upper) {
            this.upper = upper;
}
/* The thread will execute in this method */ public Integer call() {
      int sum = 0;
      for (int i 1; i <= upper; i++) =
            sum += i;

return new Integer (sum);
      }
}
```

Topic : Java Executor Framework (Cont.)

```
public class Driver
{
    public static void main(String[] args) {
        int upper = Integer.parseInt(args[0]);

        ExecutorService pool = Executors.newSingleThreadExecutor();
                Future<Integer> result = pool.submit(new Summation
(upper));

        try {
            System.out.println("sum " + result.get());
        } catch (InterruptedException | ExecutionException ie){}
    }
}
```

Q) Consider a System having n-cpu's & 'K' Processes, (K < n);

Calculate the Max (Ready ; Running ; Block)

$$\text{Ready} \rightarrow K \qquad \text{Running} \rightarrow K \qquad \text{Block} \rightarrow K$$

(1-CPU & 'K' processes)

Ready ; Running ; Block

$$\left(\text{Ready} \rightarrow K \qquad \text{Running} \rightarrow 1 \qquad \text{Block} \rightarrow K \right)$$

THANK - YOU