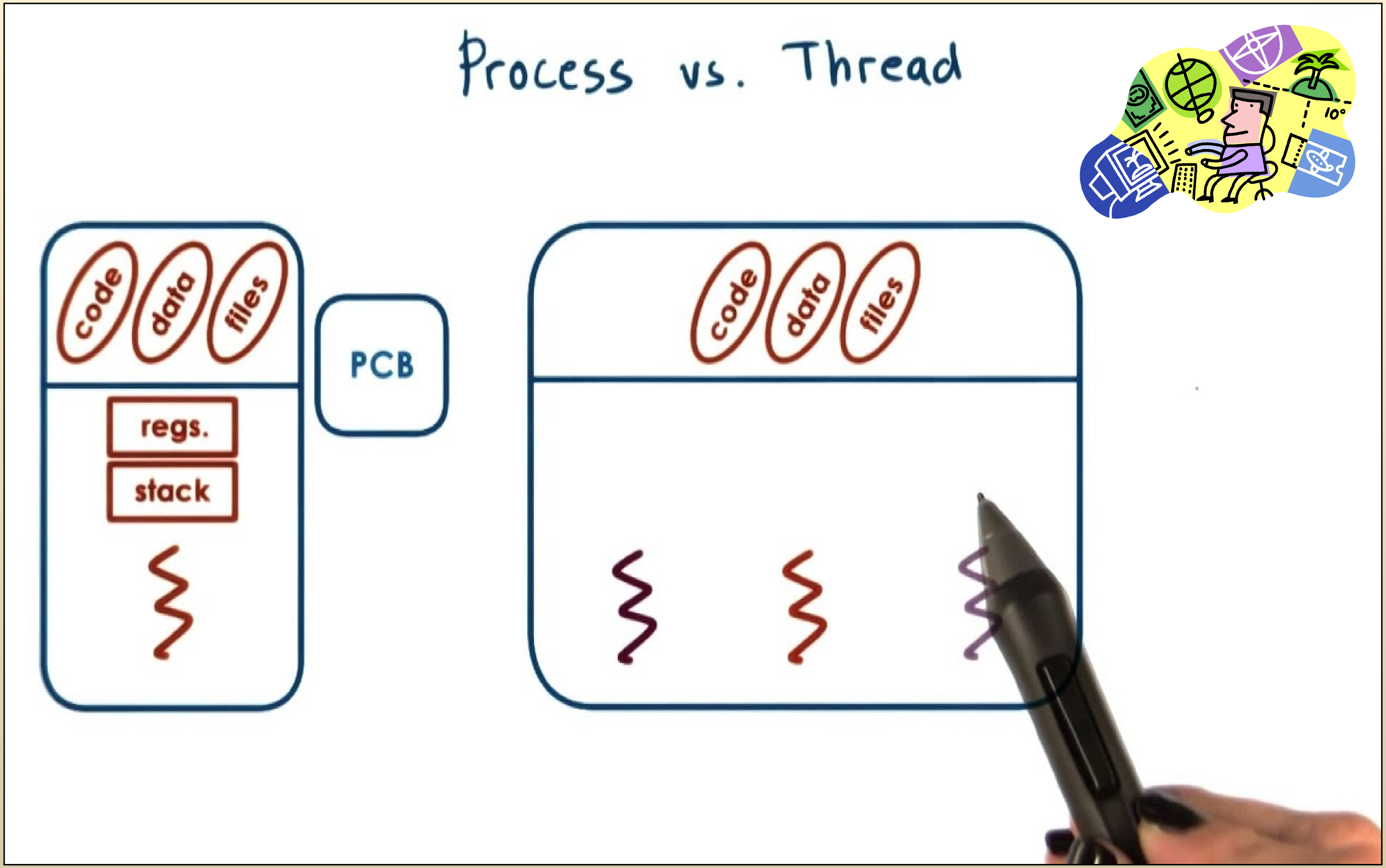


CS-3103 : Operating Systems : Sec-A (NB) : Threads & Concurrency

Processes and Threads

Traditional processes have two characteristics:

- Resource Ownership

  Process includes a virtual address space to hold the process image

  - the OS provides protection to prevent unwanted interference between processes with respect to resources

- Scheduling/Execution

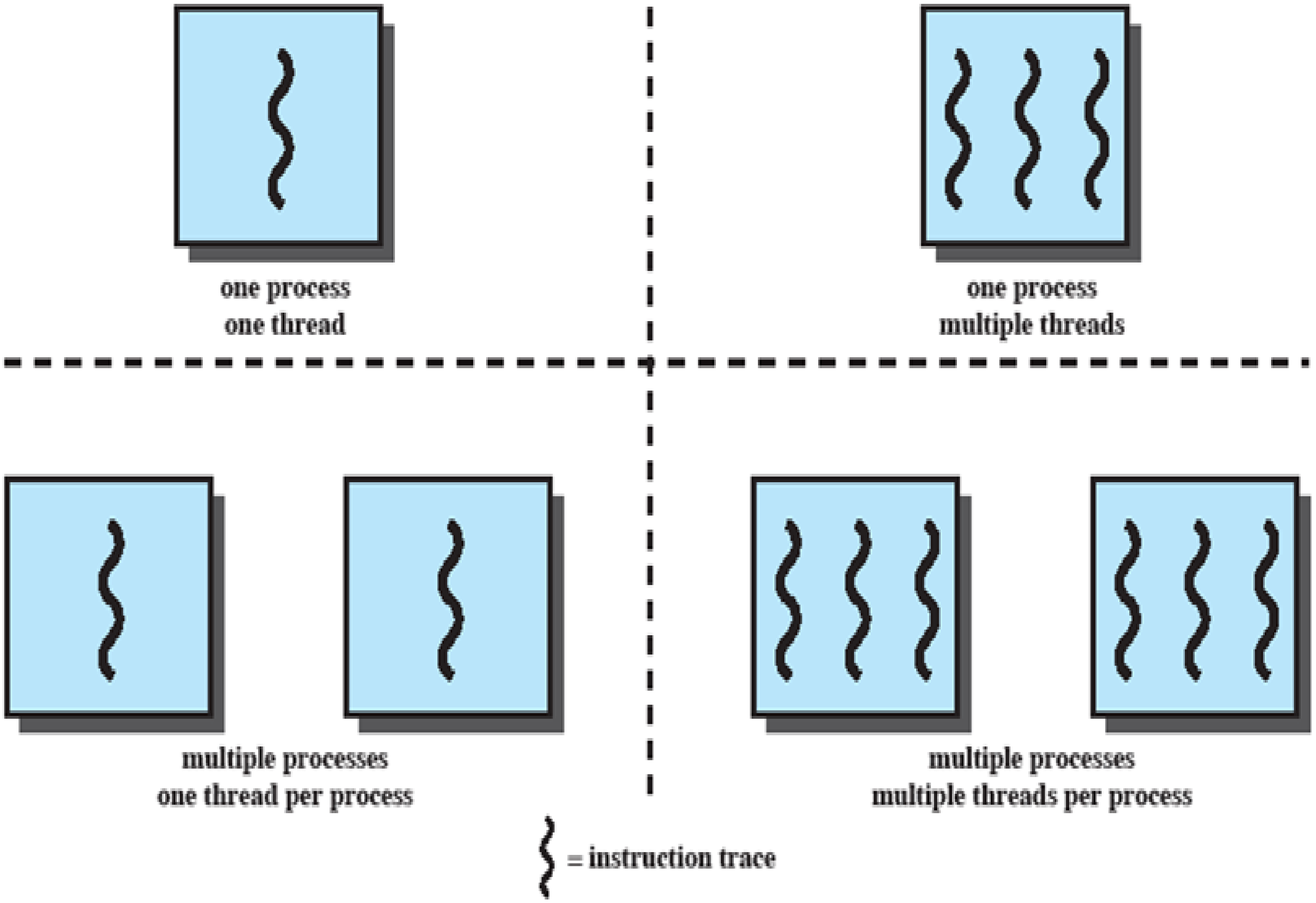
  Follows an execution path that may be interleaved with other processes

  - a process has an execution state (Running, Ready, etc.) and a dispatching priority and is scheduled and dispatched by the OS
  - Traditional processes are *sequential*; i.e. only *one* execution path

Threads

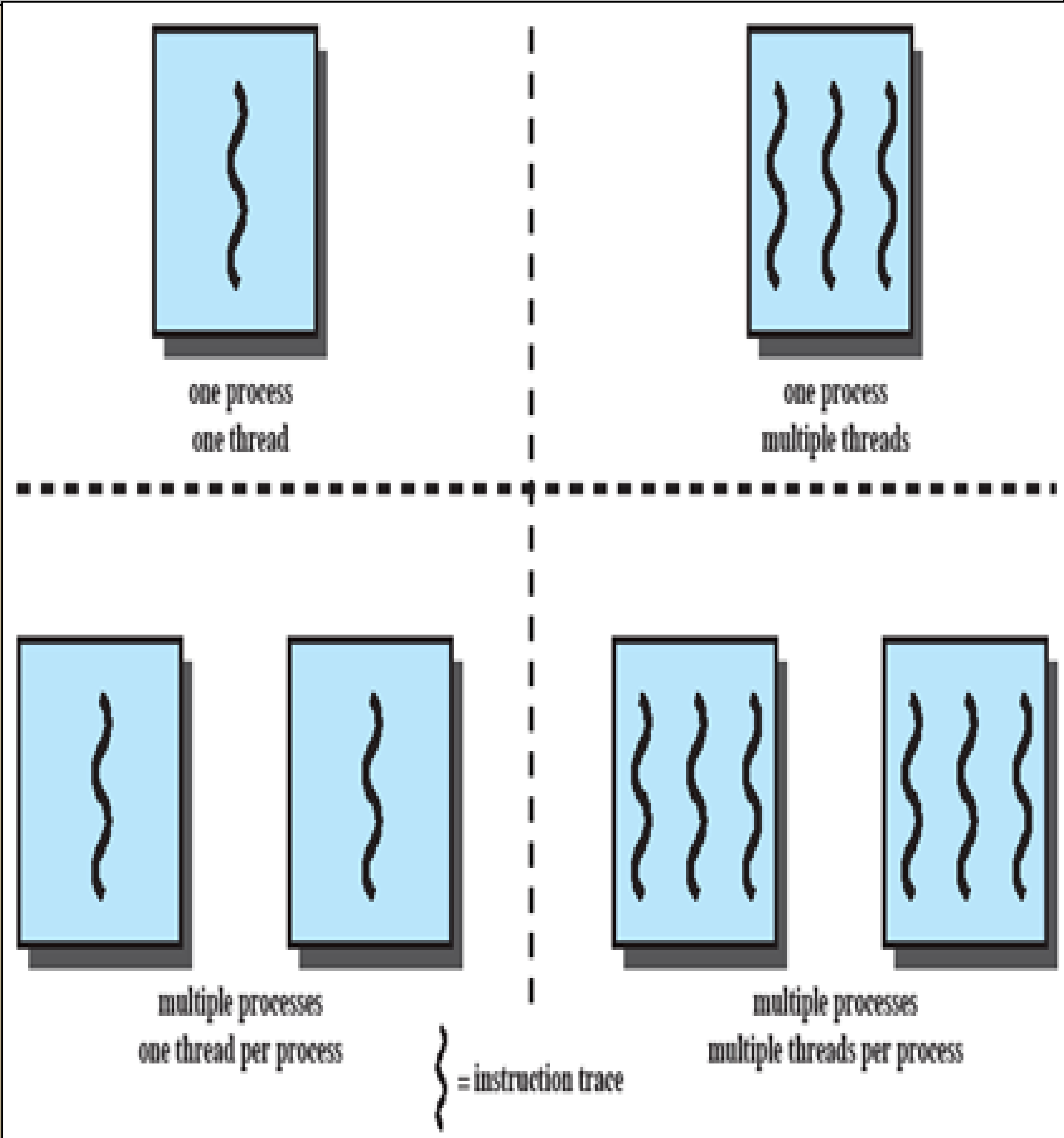
- In an OS that supports threads, scheduling and dispatching is done on a thread basis
- Most of the state information dealing with execution is maintained in thread-level data structures
  - suspending a process involves suspending all threads of the process
  - termination of a process terminates all threads within the process

Process and Threads



Multithreaded Approaches

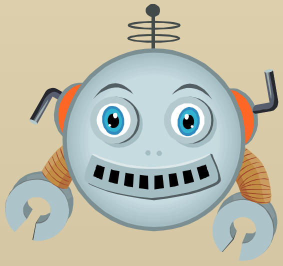
- The right half of the Fig. depicts multithreaded approaches
- A Java run-time environment is a system of *one* process with multiple threads; Windows, some UNIXes, support *multiple* multithreaded processes.



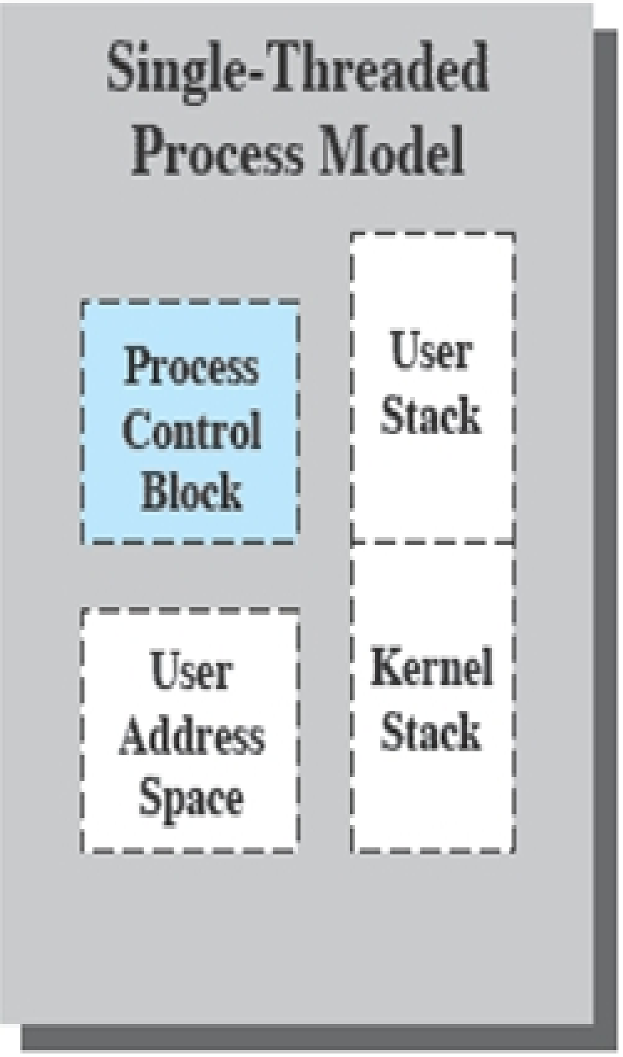
One or More Threads in a Process

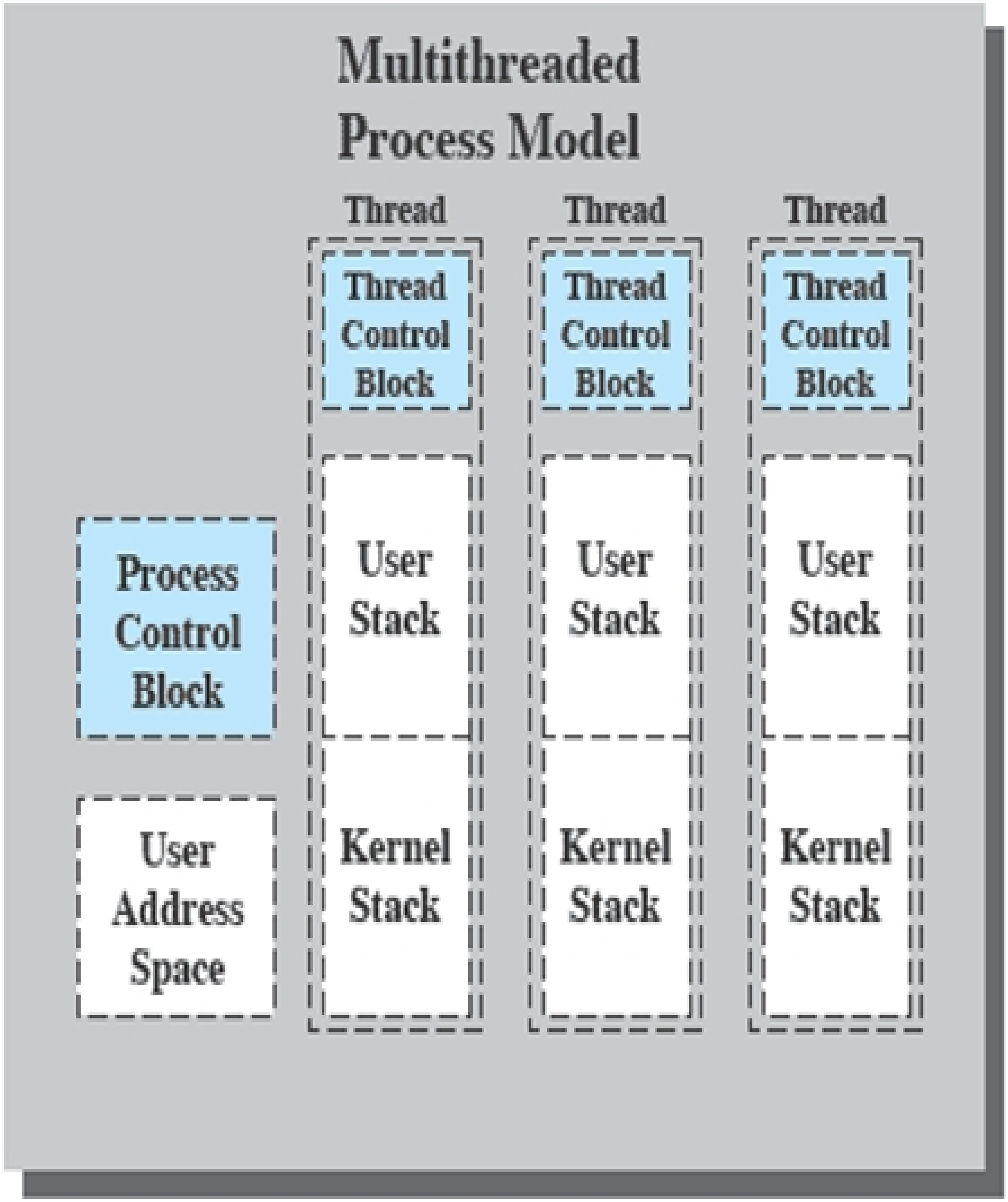
Each thread has:

- an execution state (Running, Ready, etc.)
- saved thread context when not running (TCB)
- an execution stack
- some per-thread static storage for local variables
- access to the shared memory and resources of its process (all threads of a process share this)



Threads vs. Processes





Thus, all of the threads of a process share the state and resources of that process. They reside in the same address space and have access to the same data. When one thread alters an item of data in memory, other threads see the results if and when they access that item. If one thread opens a file with read privileges, other threads in the same process can also read from that file.

The key benefits of threads derive from the performance implications:
1. It takes far less time to create a new thread in an existing process than to create a brand-new process. Studies done by the Mach developers show that thread creation is ten times faster than process creation in UNIX [TEVA87].
2. It takes less time to terminate a thread than a process.
3. It takes less time to switch between two threads within the same process than to switch between processes.
4. Threads enhance efficiency in communication between different executing programs. In most operating systems, communication between independent processes requires the intervention of the kernel to provide protection and the mechanisms needed for communication. However, because threads within the same process share memory and files, they can communicate with each other without invoking the kernel.
Thus, if there is an application or function that should be implemented as a set of related units of execution, it is far more efficient to do so as a collection of threads rather than a collection of separate processes.

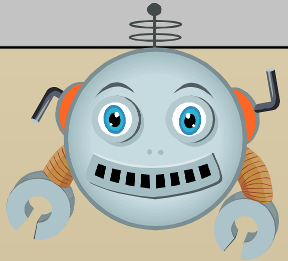
Benefits of Threads

Takes less time to create a new thread than a process

Less time to terminate a thread than a process

Switching between two threads takes less time than switching between processes

Threads enhance efficiency in communication between programs



In an OS that supports threads, scheduling and dispatching is done on a thread basis;
hence, most of the state information dealing with execution is maintained in thread-level data structures. There are, however, several actions that affect all of the threads in a process and that the OS must manage at the process level.
For example, suspension involves swapping the address space of one process out of main memory to make room for the address space of another process.
Because all threads in a process share the same address space, all threads are suspended at the same time. Similarly, termination of a process terminates all threads within that process.

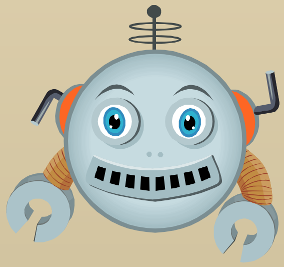
Thread Execution States

The key states for a thread are:

- Running
- Ready
- Blocked

Thread operations associated with a change in thread state are:

- Spawn (create)
- Block
- Unblock
- Finish



THREAD STATES

*As with processes, the key states for a thread are **Running**, **Ready**,* and **Blocked**. Generally, it does not make sense to associate suspend states with threads because such states are process-level concepts. In particular, if a process is swapped out, all of its threads are necessarily swapped out because they all share the address space of the process. There are four basic thread operations associated with a change in thread state [ANDE04]:

- **Spawn:** Typically, when a new process is spawned, a thread for that process is also spawned. Subsequently, a thread within a process may spawn another thread within the same process, providing an instruction pointer and arguments for the new thread. The new thread is provided with its own register context and stack space and placed on the ready queue.

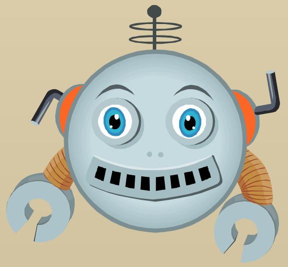
- **Block:** When a thread needs to wait for an event, it will block (saving its user registers, program counter, and stack pointers). The processor may now turn to the execution of another ready thread in the same or a different process.

- **Unblock:** When the event for which a thread is blocked occurs, the thread is moved to the Ready queue.

- **Finish:** When a thread completes, its register context and stacks are deallocated.

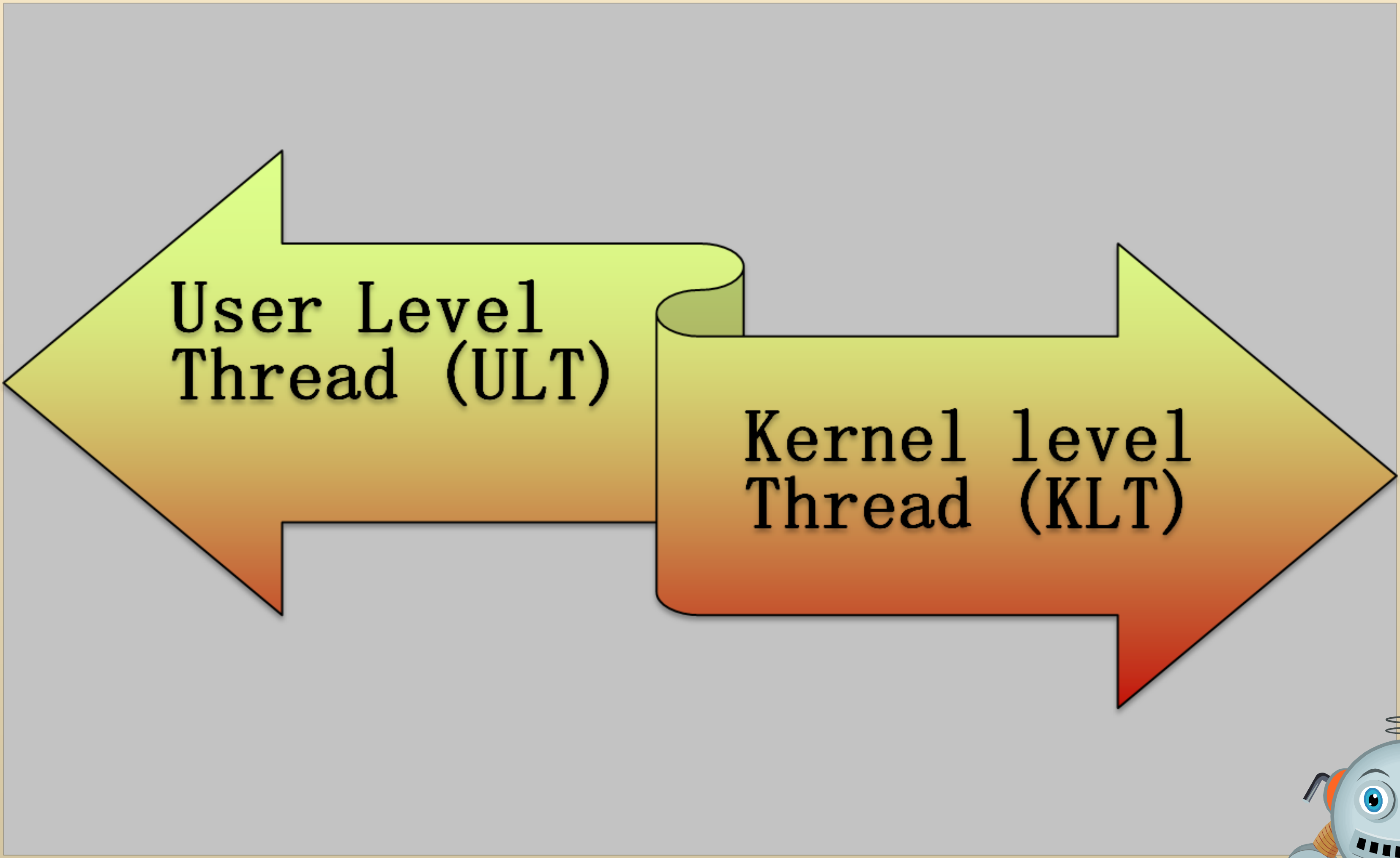
Thread Synchronization

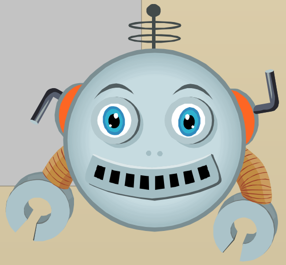
- It is necessary to synchronize the activities of the various threads
  - all threads of a process share the same address space and other resources
  - any alteration of a resource by one thread affects the other threads in the same process



START

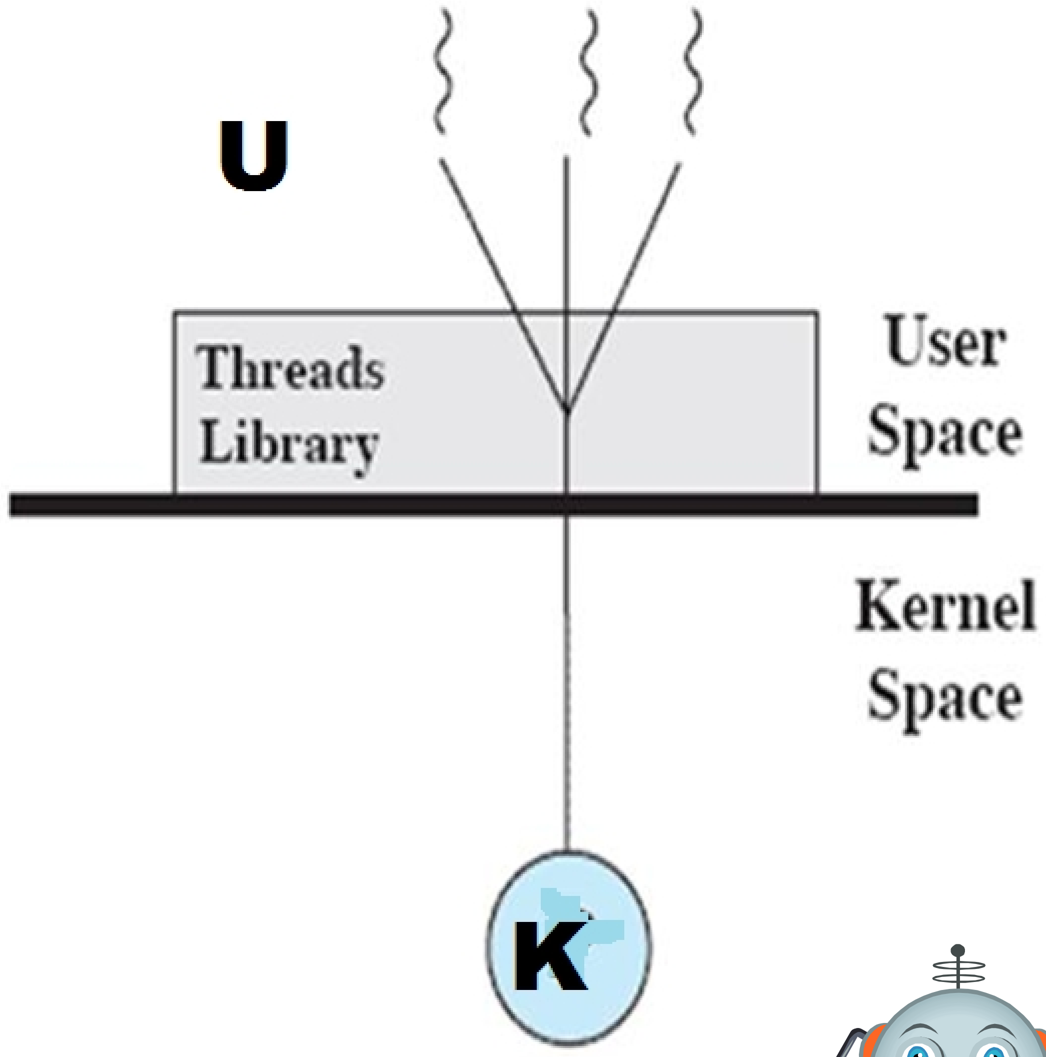
Types of Threads



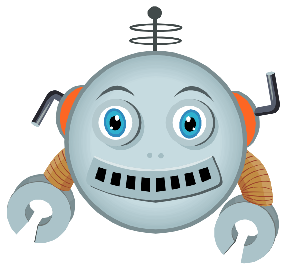


User-Level Threads (ULTs)

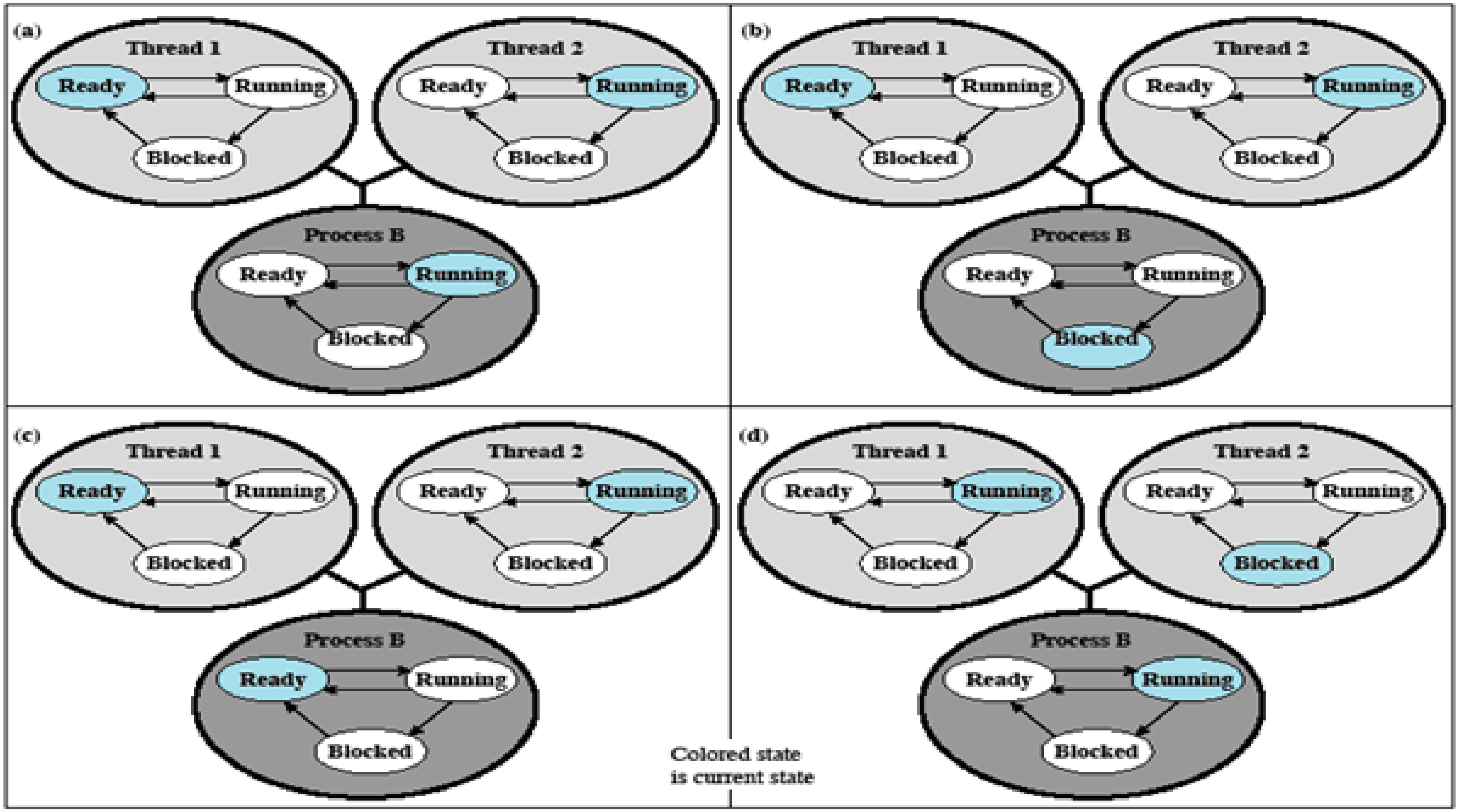
- Thread management is done by the application
- The kernel is not aware of the existence of threads
- User creates thread(s) using pthread_create(….) function.



(a) Pure user-level



Relationships Between ULT States and Process States



Possible transitions from Xa: Xa→Xb; Xa→Xc; Xa→Xd

Figure X: Examples of the Relationships between User-Level Thread States and Process States

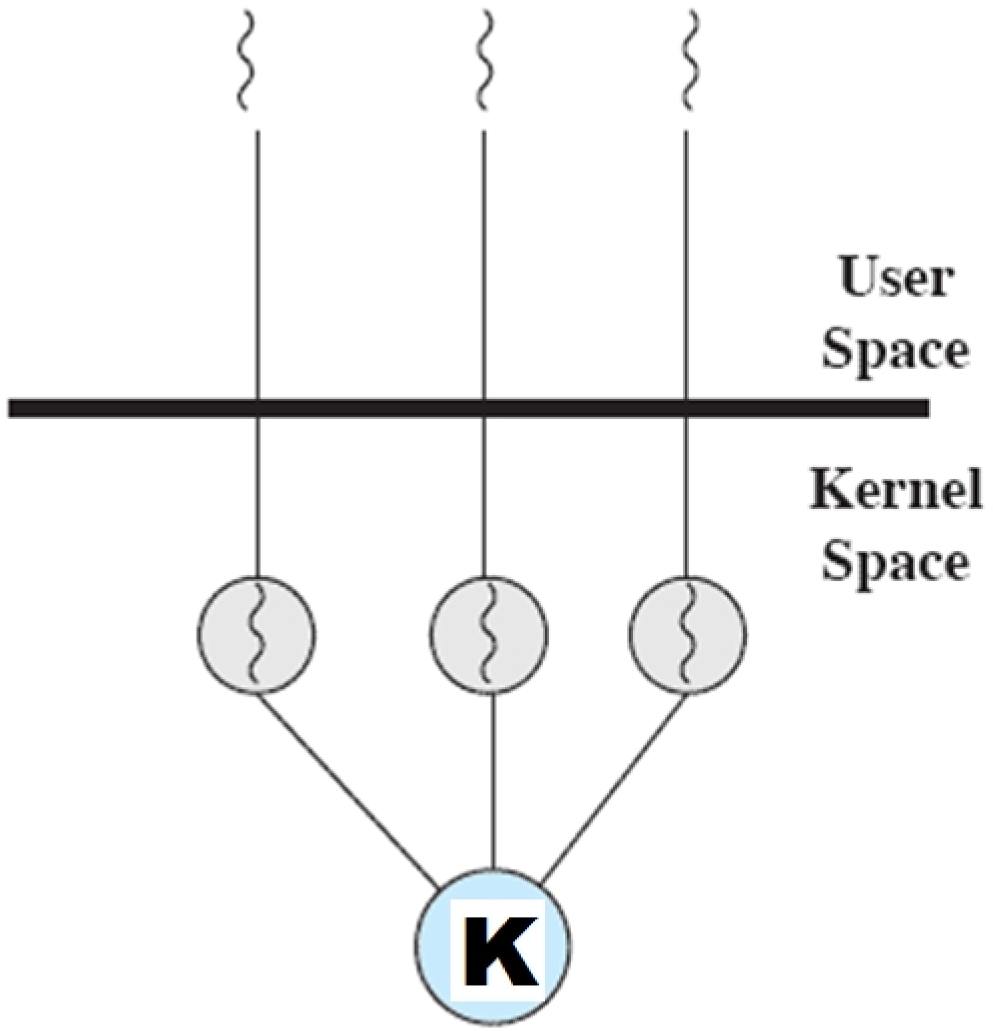
Advantages of ULTs

Thread switching does not require kernel mode privileges (no mode switches)

Scheduling can be application specific

ULTs can run on any OS

Kernel-Level Threads (KLTs)

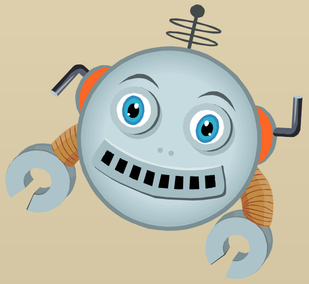


(b) Pure kernel-level

- Thread management is done by the kernel (could call them K*M*T)
  - no thread management is done by the application
    - Windows is an example of this approach

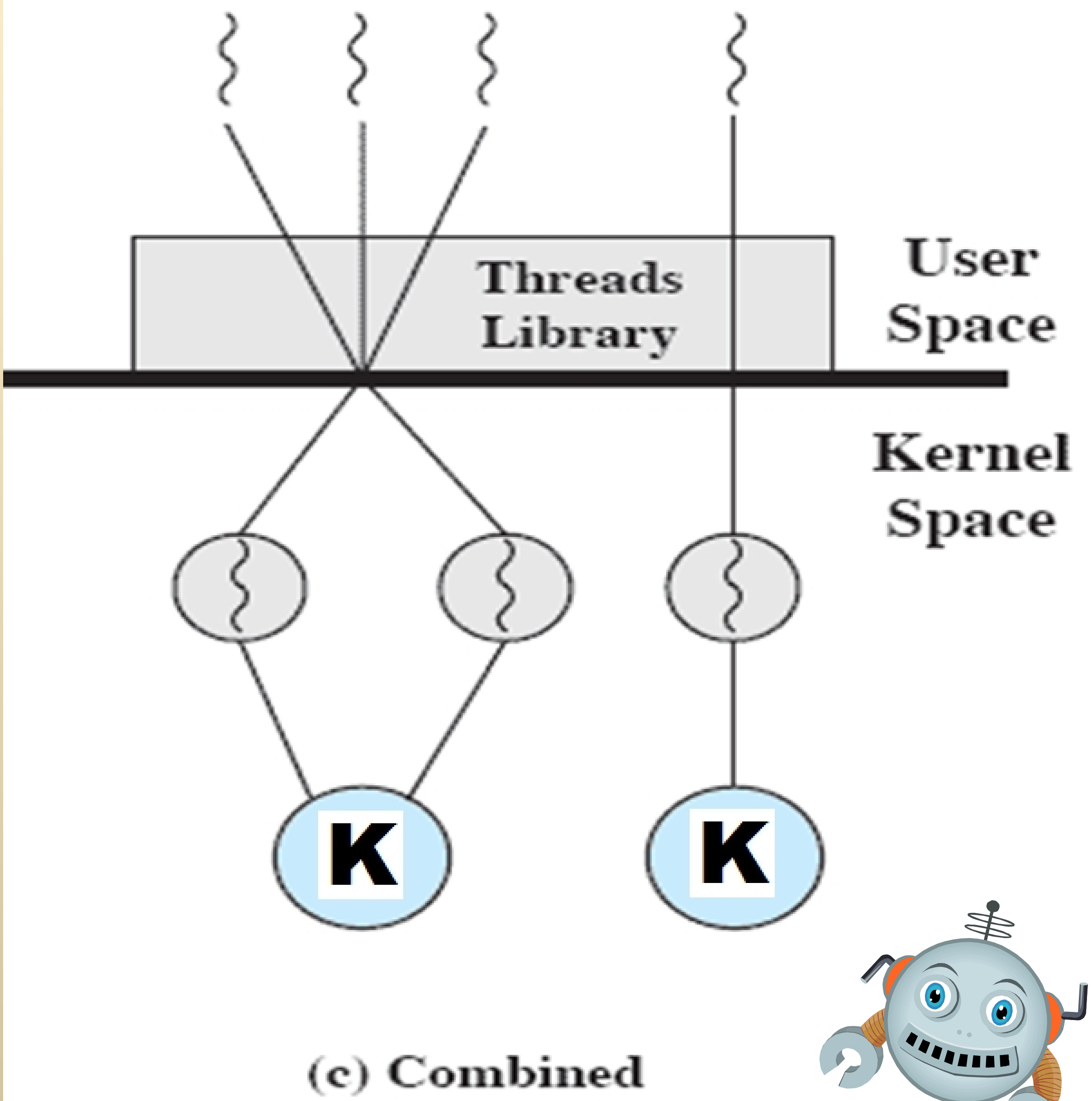
Advantages of KLTs

- The kernel can simultaneously schedule multiple threads from the same process on multiple processors
- If one thread in a process is blocked, the kernel can schedule another thread of the same process

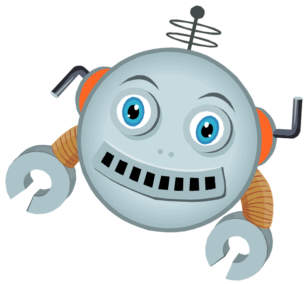


Combined Approaches

- Thread creation is done in the user space
- Bulk of scheduling and synchronization of threads is by the application
- Solaris is an example



(c) Combined



Questions…

1. Which one of the following is not shared by threads?

Answer: Option C

A. program counter **B.** stack

C. both (a) and (b) **D.** none of the mentioned

2. A process can be: **Answer: Option C**

A. single threaded **B.** multithreaded

C. both (a) and (b) **D.** none of the mentioned

3. If one thread opens a file with read privileges then:

A. other threads in the another process can also read from that file

B. other threads in the same process can also read from that file

C. any other thread can not read from that file Answer: Option B

D. all of the mentioned

Questions…

4. The time required to create a new thread in an existing process is:
A. greater than the time required to create a new process
B. less than the time required to create a new process
C. equal to the time required to create a new process
D. none of the mentioned

Answer: Option B

5. When the event for which a thread is blocked occurs,
A. thread moves to the ready queue
B. thread remains blocked
C. thread completes
D. a new thread is provided

Answer: Option A

6. Termination of the process terminates:
A. first thread of the process
B. first two threads of the process
C. all threads within the process
D. no thread within the process

Answer: Option C

Questions…

7. The register context and stacks of a thread are deallocated when the thread:
A. terminated **B.** blocks
C. unblocks **D.** spawns

Answer: Option A

8. Thread synchronization is required because:
A. all threads of a process share the same address space
B. all threads of a process share the same global variables
C. all threads of a process can share the same files
D. all of the mentioned

Answer: Option D

9. A thread shares its resources (like data section, code section, open files, signals) with:
A. other process similar to the one that the thread belongs to
B. other threads that belong to similar processes
C. other threads that belong to the same process
D. All of these

Answer: Option C

10. A process having multiple threads of control implies :
A. it can do more than one task at a time
B. it can do only one task at a time, but much faster
C. it has to use only one thread per process
D. None of these

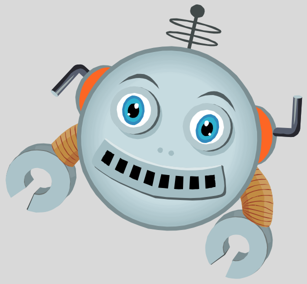
Answer: Option A

Assignments…

```
// C program to find maximum number of thread within a process
#include<stdio.h>
#include<pthread.h>

// This function demonstrates the work of thread which is of no use here, So
left blank
void *thread ( void *vargp){}

int main()
{
   int err = 0, count = 0;
   pthread_t tid;
   // on success, pthread_create returns 0 and on Error, it returns error
number
   // So, while loop is iterated until return value is 0
   while (err == 0)
   {  err = pthread_create (&tid, NULL, thread, NULL);
      count++;
   }
   printf("Maximum number of thread within a Process is : %d\n", count);
   return 0;
}
```



```
nilina@nilina-HP-Pro-3330-MT:~/Desktop/csen3113/thr$ gcc th1.c -o
th1 -pthread
nilina@nilina-HP-Pro-3330-MT:~/Desktop/csen3113/thr$ ./th1
Maximum number of thread within a Process is : 32755
```