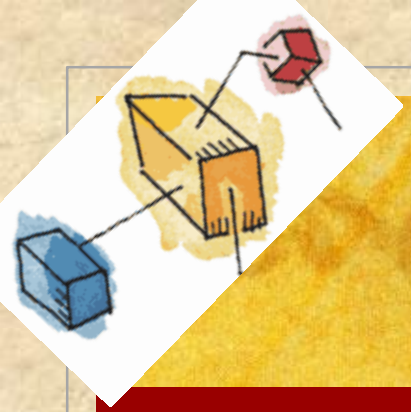
IT 540 Operating Systems ECE519 Advanced Operating Systems

Prof. Dr. Hasan Hüseyin BALIK

(4th Week)

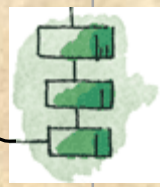
(Advanced) Operating Systems

4. Threads



4. Outline

- Processes and Threads
- Types of Threads
- Multicore and Multithreading



Processes and Threads

Resource Ownership

Process includes a virtual address space to hold the process image

- the OS performs a protection function 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

Processes and Threads

- The unit of dispatching is referred to as a ***thread*** or ***lightweight process***
- The unit of resource ownership is referred to as a ***process*** or ***task***
- ***Multithreading*** **-** The ability of an OS to support multiple, concurrent paths of execution within a single process

Single Threaded Approaches

- A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
- MS-DOS is an example

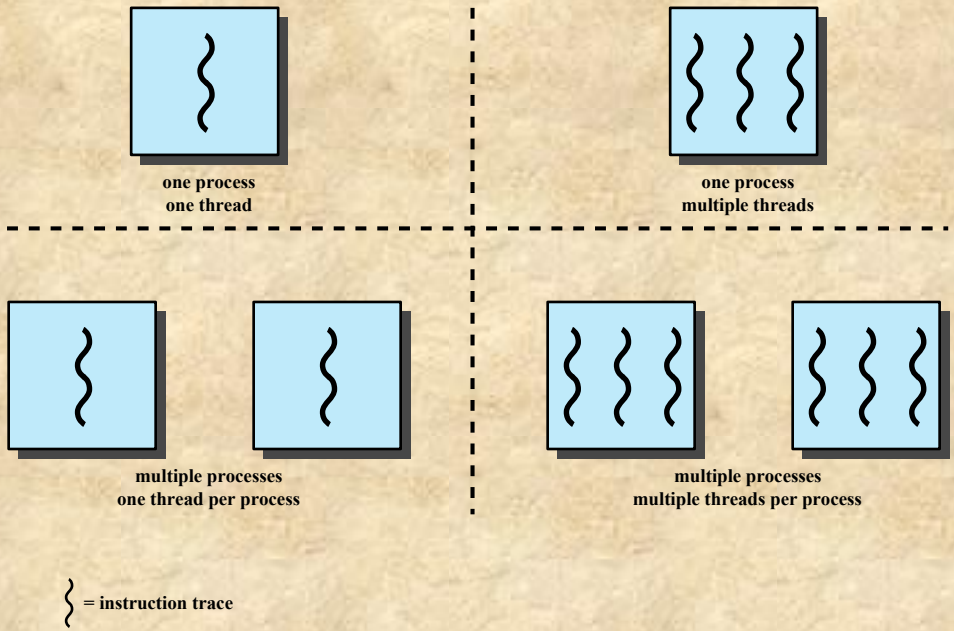


Figure 4.1 Threads and Processes

Multithreaded Approaches

- The right half of Figure 4.1 depicts multithreaded approaches
- A Java run-time environment is an example of a system of one process with multiple threads

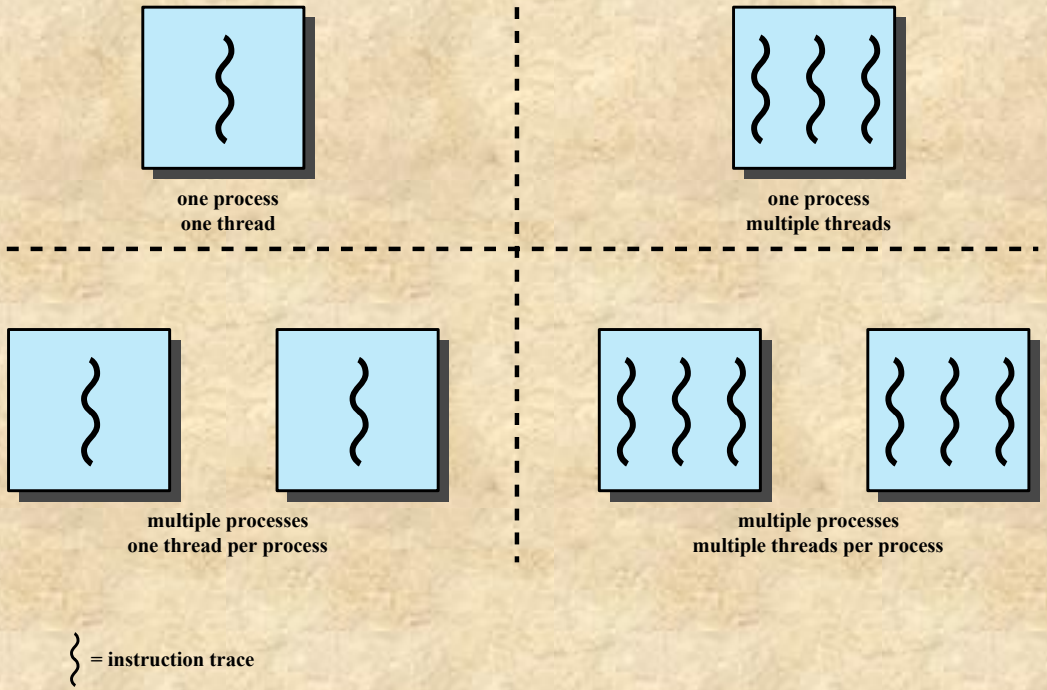


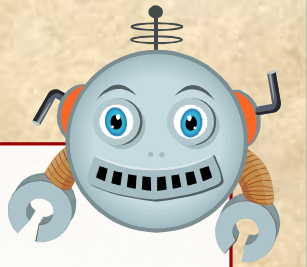
Figure 4.1 Threads and Processes

Processes

- The unit or resource allocation and a unit of protection
- A virtual address space that holds the process image
- Protected access to:
  - processors
  - other processes
  - files
  - I/O resources

One or More Threads in a Process

Each thread has:



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

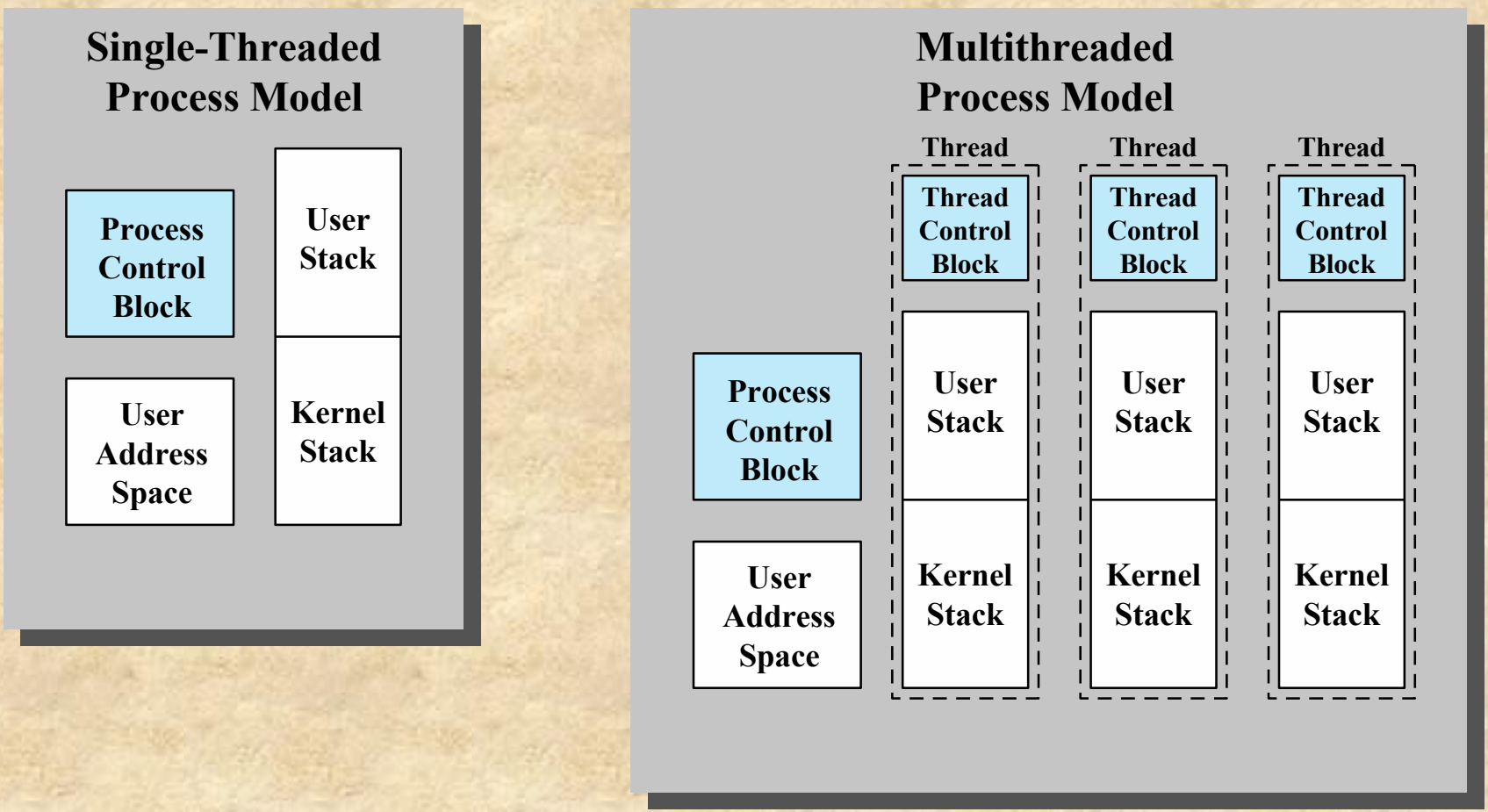
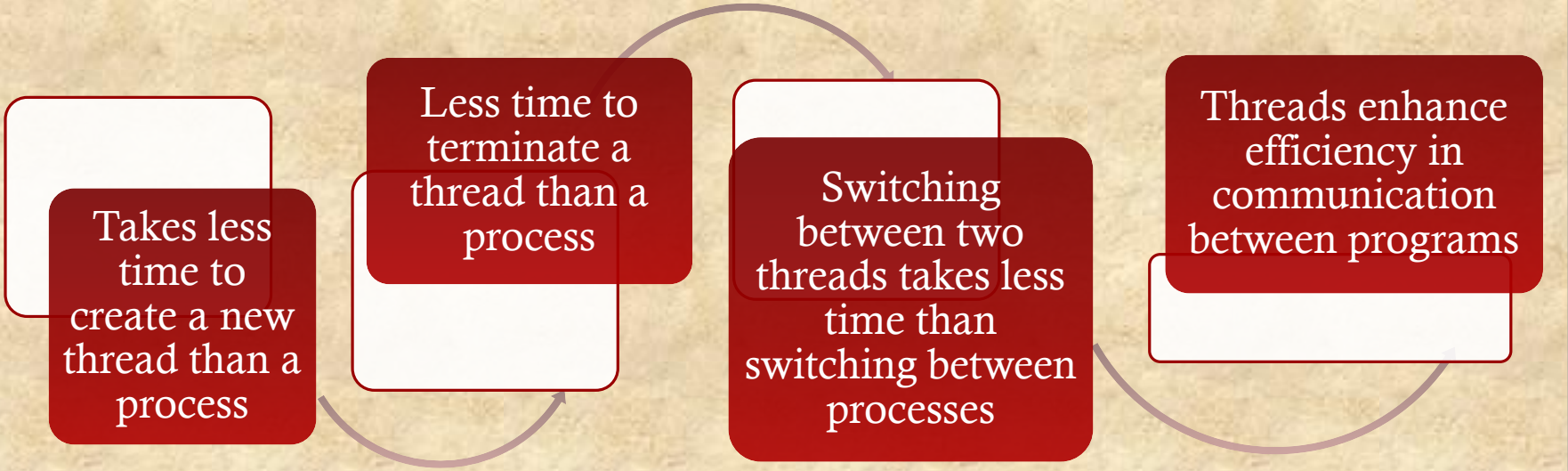


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads



Thread Use in a Single-User System

- Foreground and background work
- Asynchronous processing
- Speed of execution
- Modular program structure

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

Thread Execution States

The key states for a thread are:

- Running
- Ready
- Blocked

Thread operations associated with a change in thread state are:

- Spawn
- Block
- Unblock
- Finish

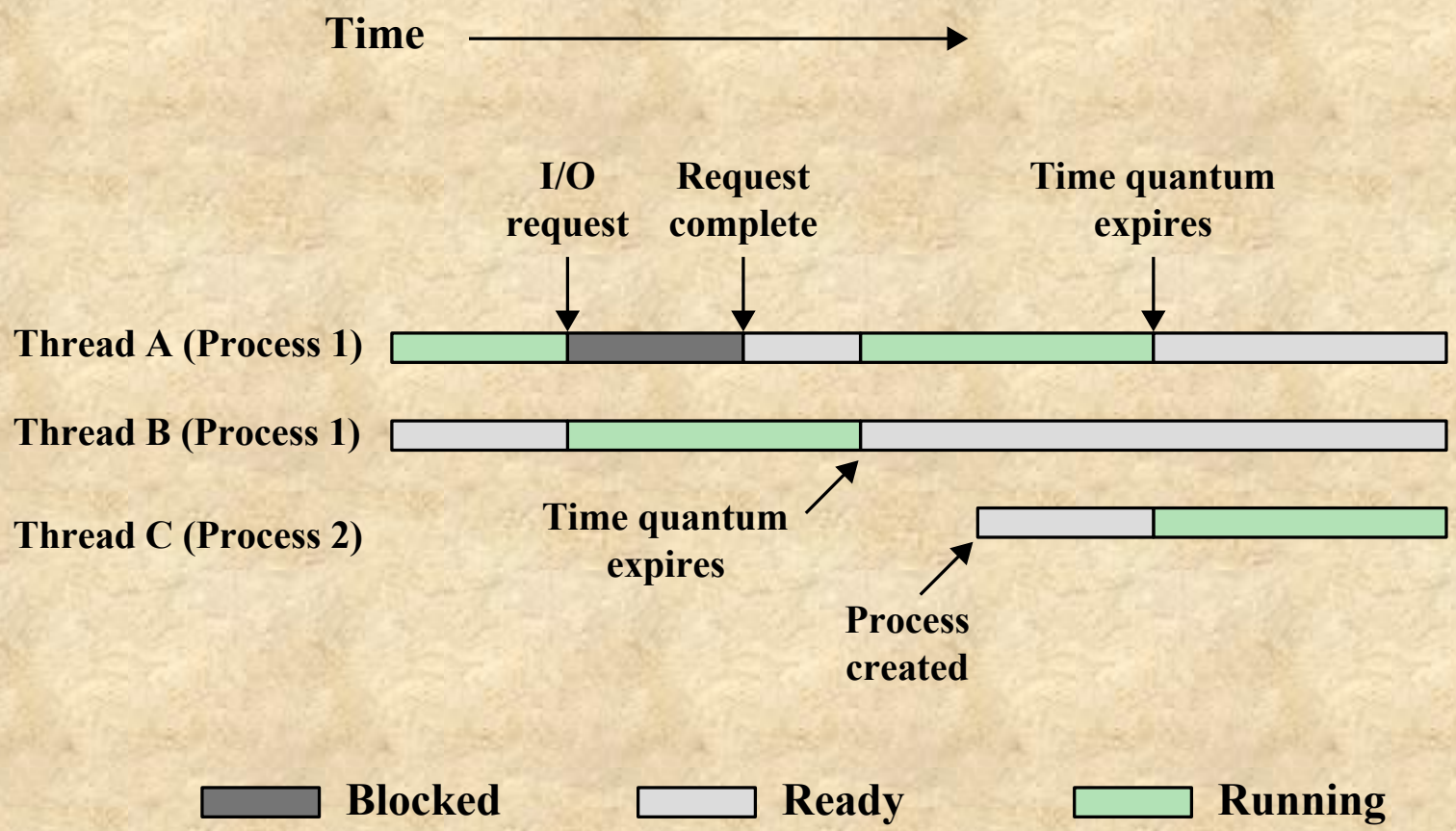
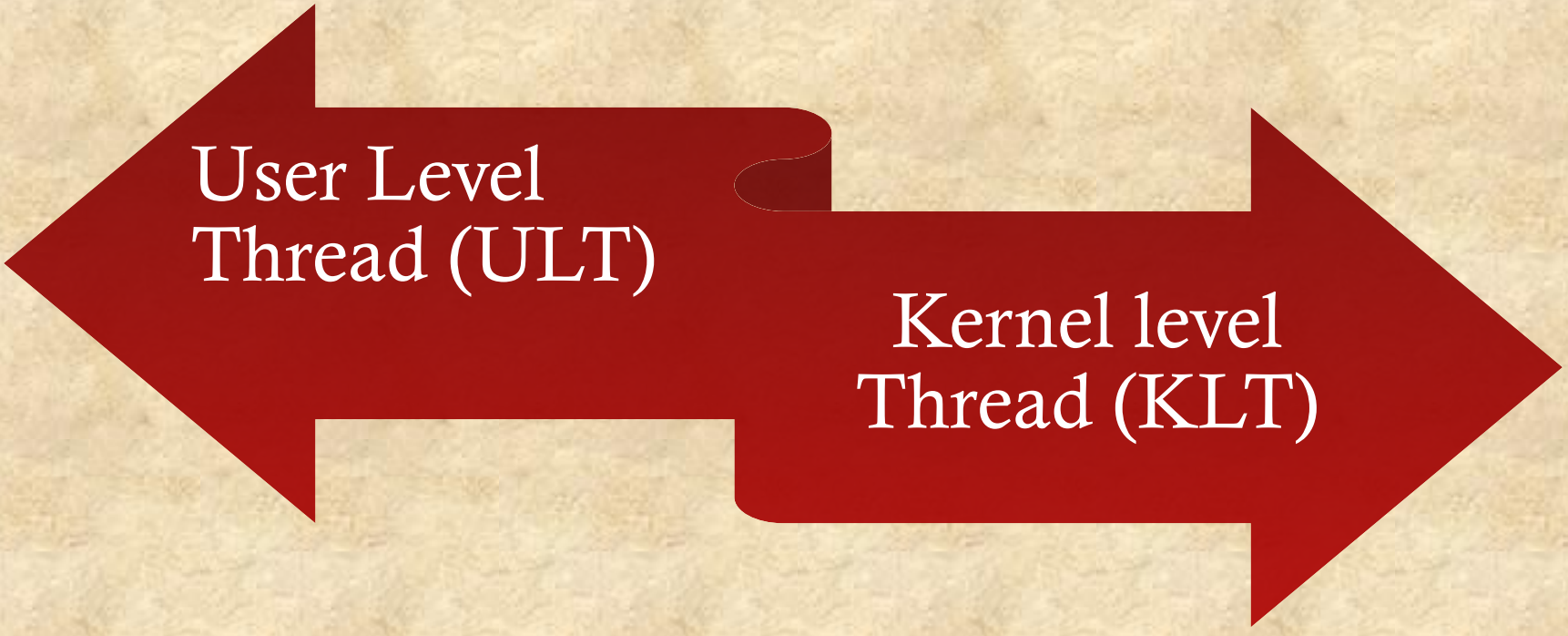


Figure 4.4 Multithreading Example on a Uniprocessor

Thread Synchronization

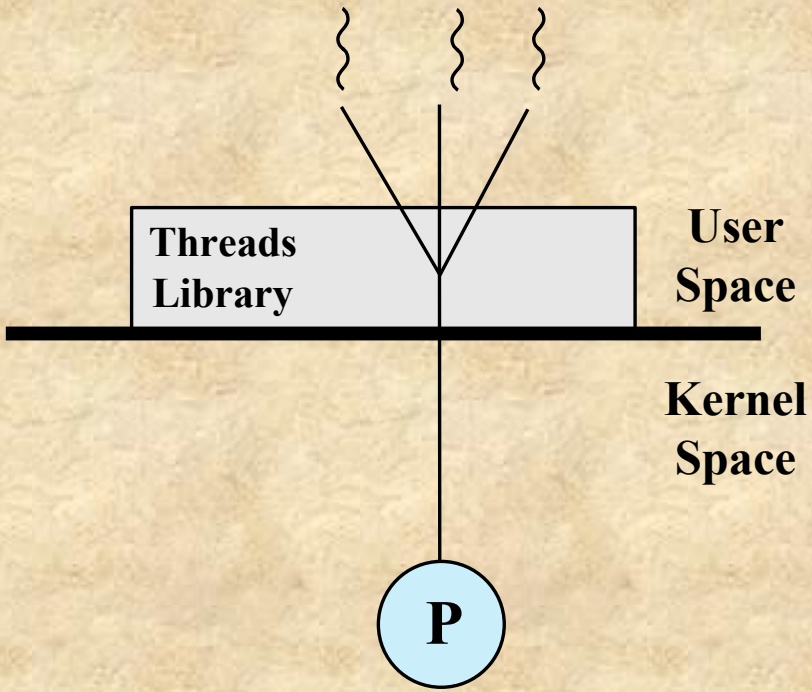
- It is necessary to synchronize the activities of the various threads
  - all threads of a process share the same address space and other resources
  - any alteration of a resource by one thread affects the other threads in the same process

Types of Threads



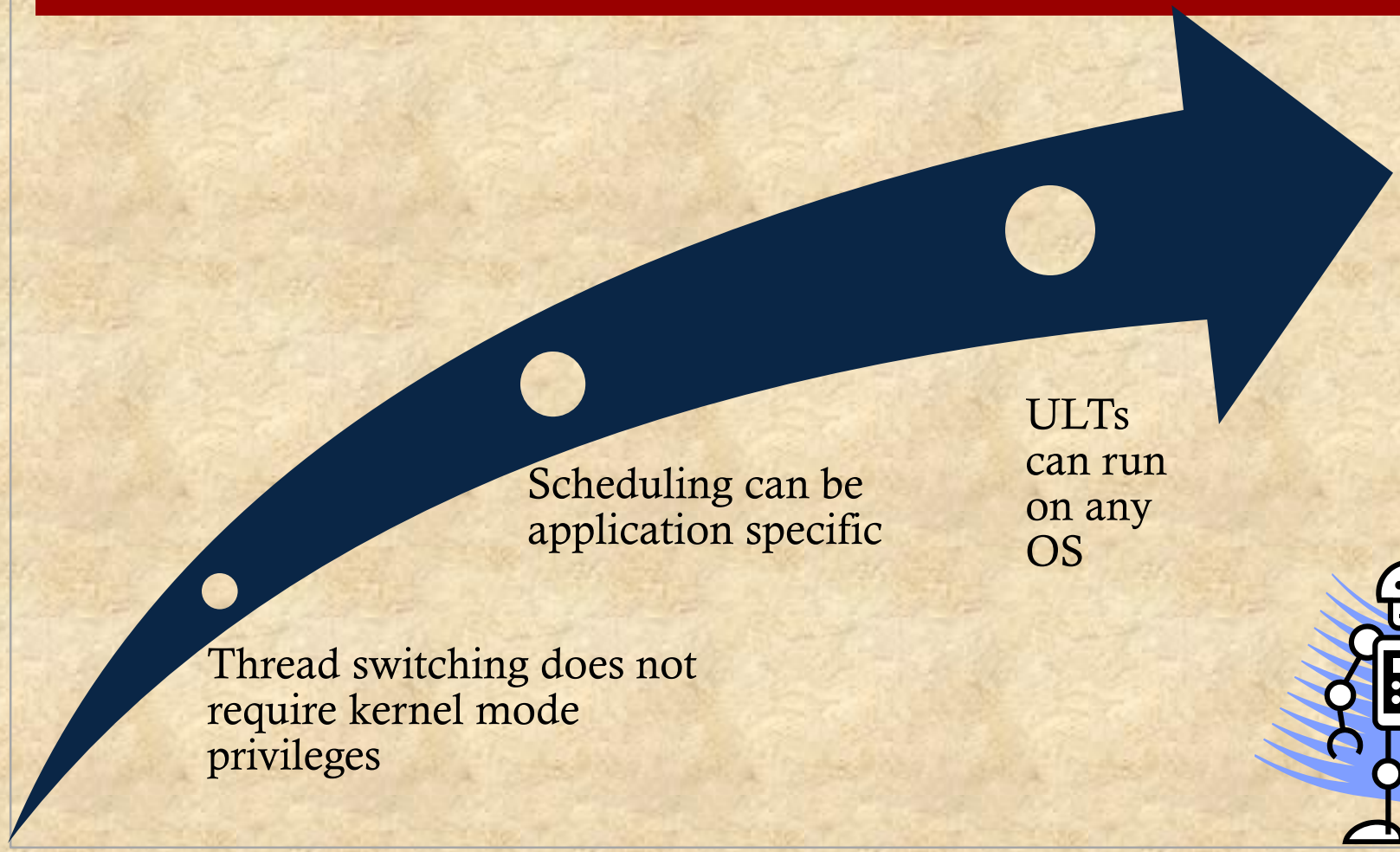
User-Level Threads (ULTs)

- All thread management is done by the application
- The kernel is not aware of the existence of threads



(a) Pure user-level

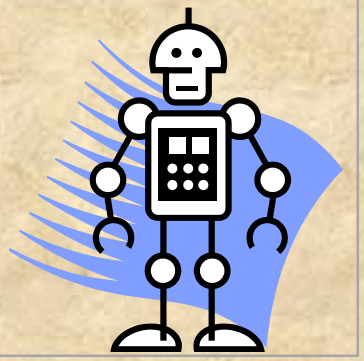
Advantages of ULTs



Thread switching does not require kernel mode privileges

Scheduling can be application specific

ULTs can run on any OS



Disadvantages of ULTs

- In a typical OS many system calls are blocking
  - as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
- In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing

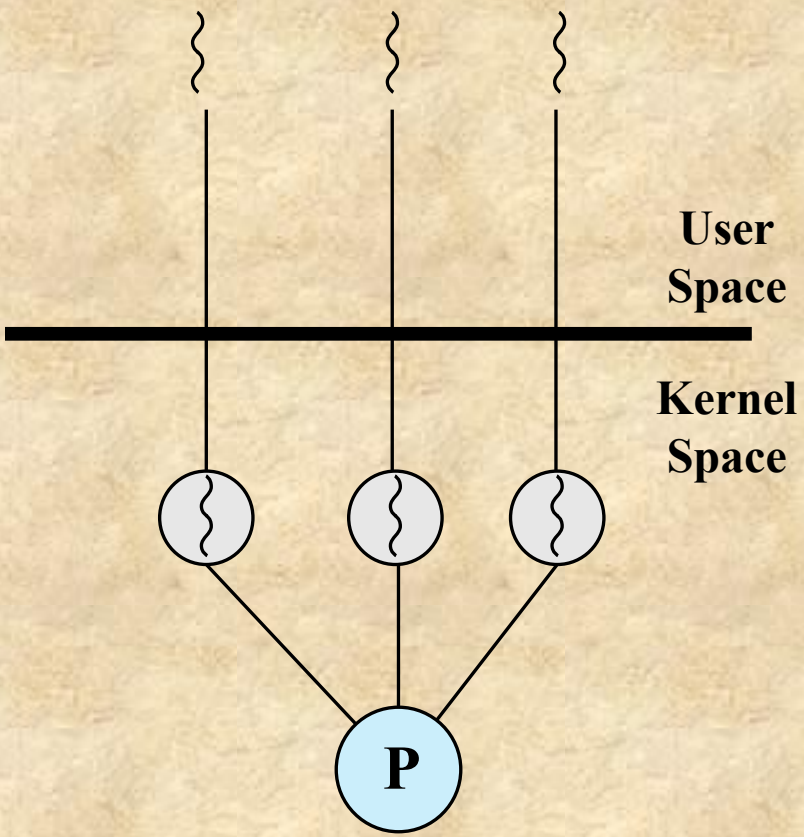
Overcoming ULT Disadvantages

Jacketing

- converts a blocking system call into a non-blocking system call

Writing an application as multiple processes rather than multiple threads

Kernel-Level Threads (KLTs)

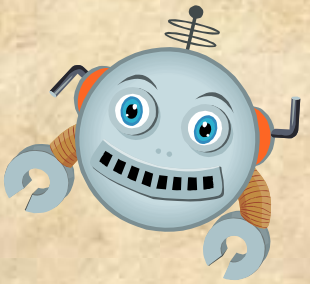


(b) Pure kernel-level

- Thread management is done by the kernel
  - 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
- Kernel routines can be multithreaded

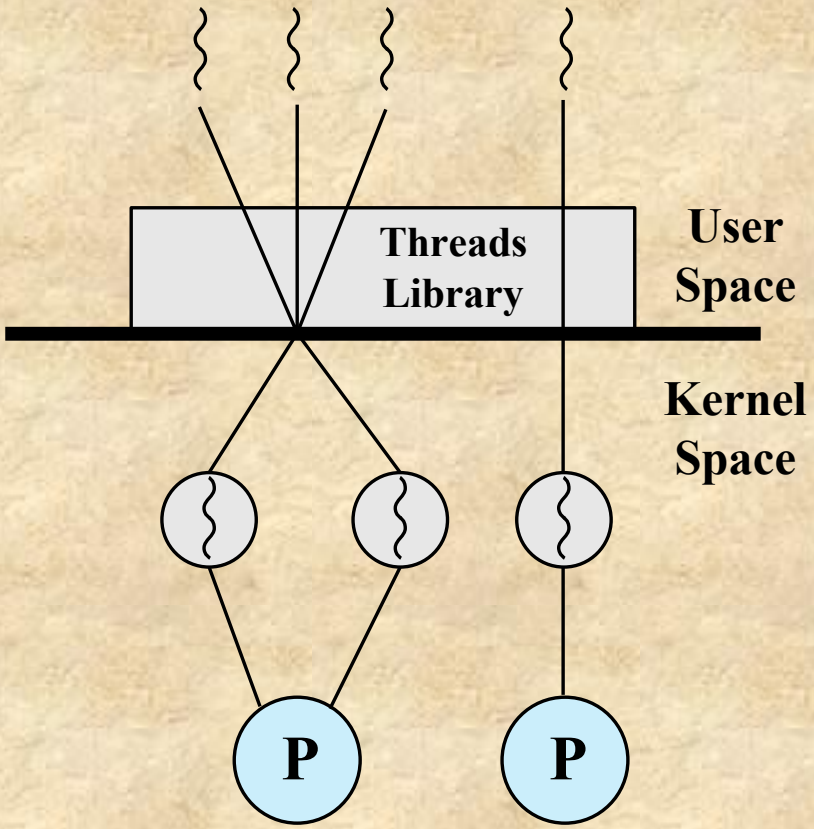


Disadvantage of KLTs

- The transfer of control from one thread to another within the same process requires a mode switch to the kernel

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

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Relationship between Threads and Processes