

# ***Threads, SMP, and Microkernels***

Athar Mahboob  
MIS & CS Department  
Institute of Business Administration  
[athar@atharmahboob.com](mailto:athar@atharmahboob.com)  
<http://www.atharmahboob.com>

# *Process*

- ◆ Resource ownership - process is allocated a virtual address space to hold the process image
- ◆ Scheduling/execution- follows an execution path that may be interleaved with other processes
- ◆ These two characteristics are treated independently by the operating system

# *Process*

- ◆ Dispatching is referred to as a thread
- ◆ Resource of ownership is referred to as a process or task

# *Multithreading*

- ◆ Operating system supports multiple threads of execution within a single process
- ◆ MS-DOS supports a single thread
- ◆ UNIX supports multiple user processes but only supports one thread per process
- ◆ Windows 2000, Solaris, Linux, Mach, and OS/2 support multiple threads

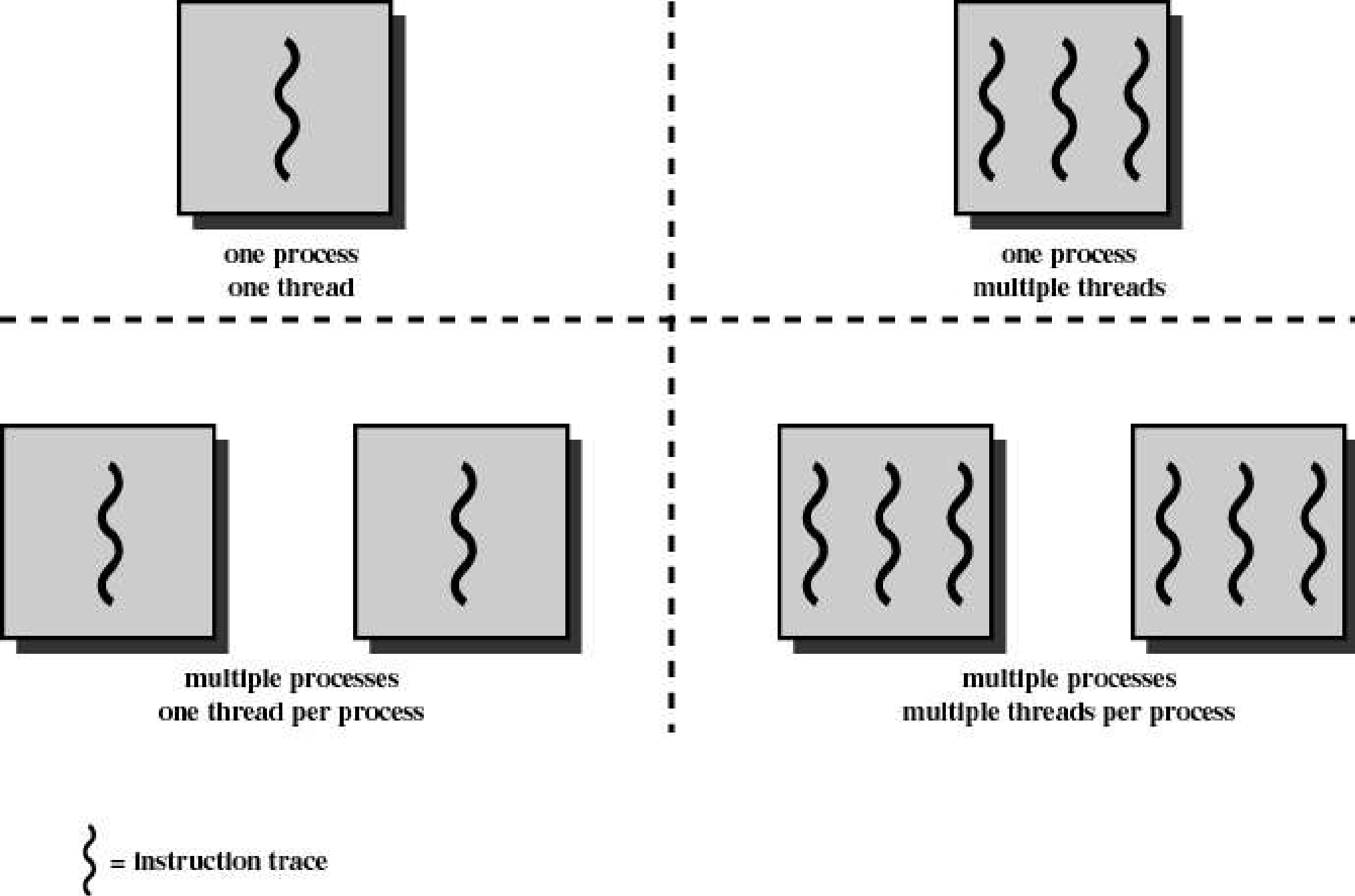


Figure 4.1 Threads and Processes [ANDE97]

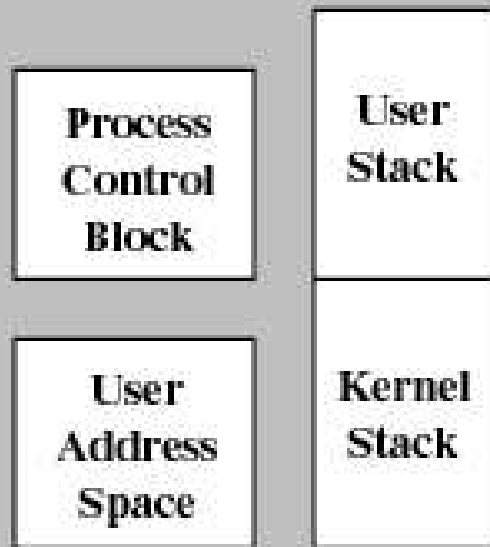
# *Process*

- ◆ Have a virtual address space which holds the process image
- ◆ Protected access to processors, other processes, files, and I/O resources

# *Thread*

- ◆ An execution state (running, ready, etc.)
- ◆ Saved thread context when not running
- ◆ Has 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

## Single-Threaded Process Model



## Multithreaded Process Model

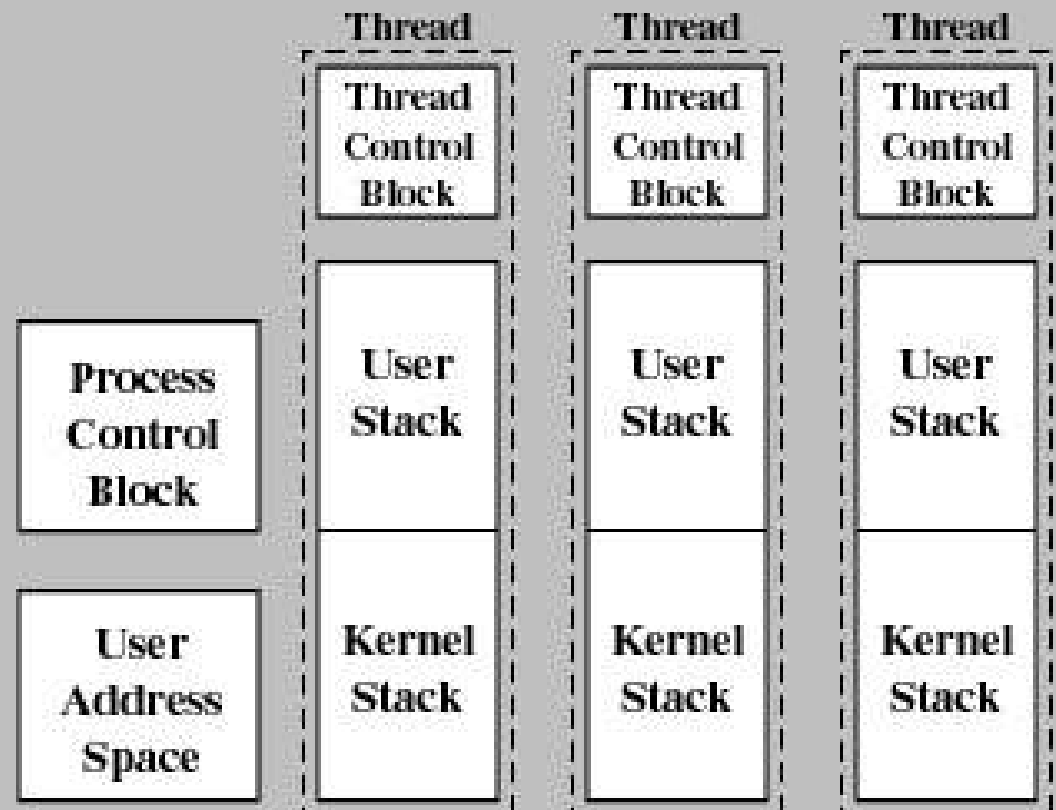


Figure 4.2 Single Threaded and Multithreaded Process Models

# *Benefits of Threads*

- ◆ Takes less time to create a new thread than a process
- ◆ Less time to terminate a thread than a process
- ◆ Less time to switch between two threads within the same process
- ◆ Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

# *Uses of Threads in a Single-User Multiprocessing System*

- ◆ Foreground to background work
- ◆ Asynchronous processing
- ◆ Speed execution
- ◆ Modular program structure

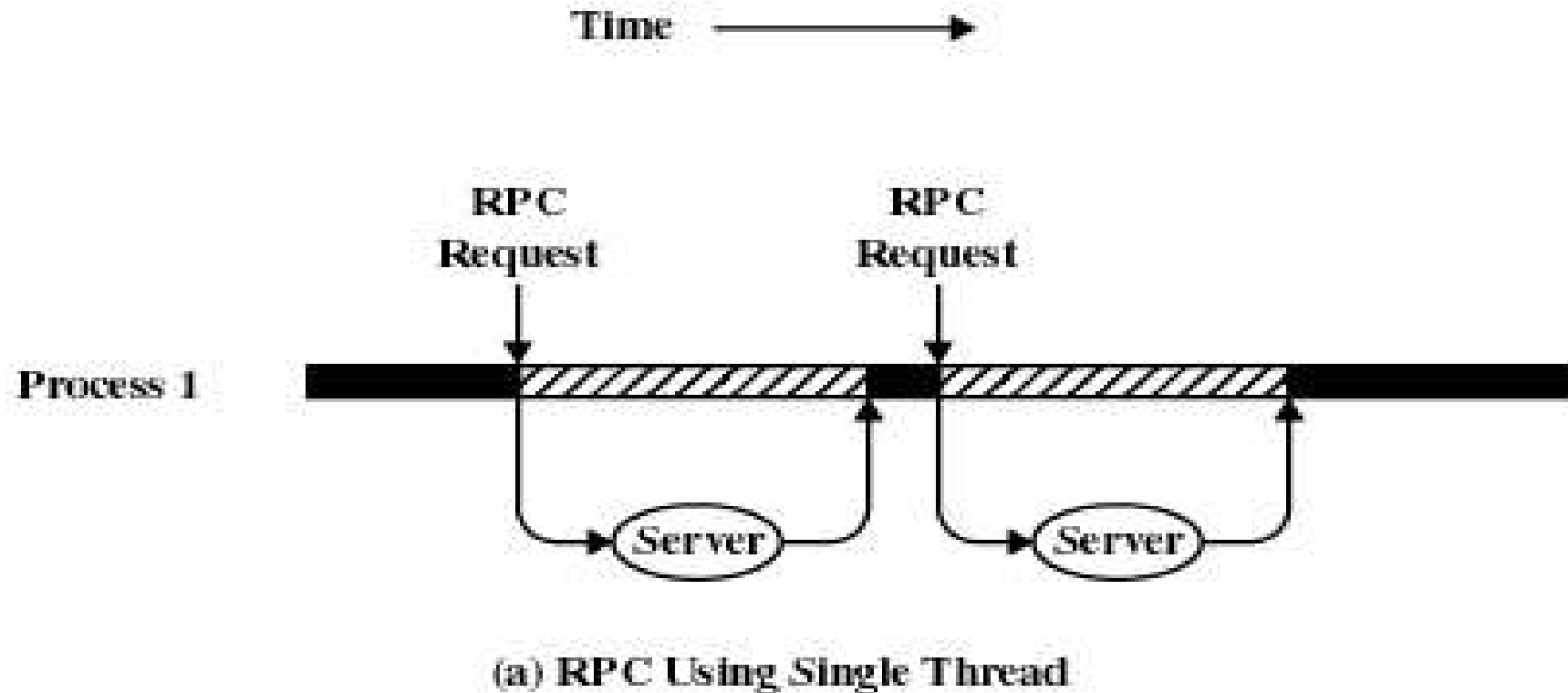
# *Threads*

- ◆ Suspending a process involves suspending all threads of the process since all threads share the same address space
- ◆ Termination of a process, terminates all threads within the process

# *Thread States*

- ◆ States associated with a change in thread state
  - ◆ Spawn
    - ◆ Spawn another thread
  - ◆ Block
  - ◆ Unblock
  - ◆ Finish
    - ◆ Deallocate register context and stacks

# Remote Procedure Call Using Threads






-  Blocked, waiting for response to RPC
-  Blocked, waiting for processor, which is in use by Thread B
-  Running

Figure 4.3 Remote Procedure Call (RPC) Using Threads

# Remote Procedure Call Using Threads

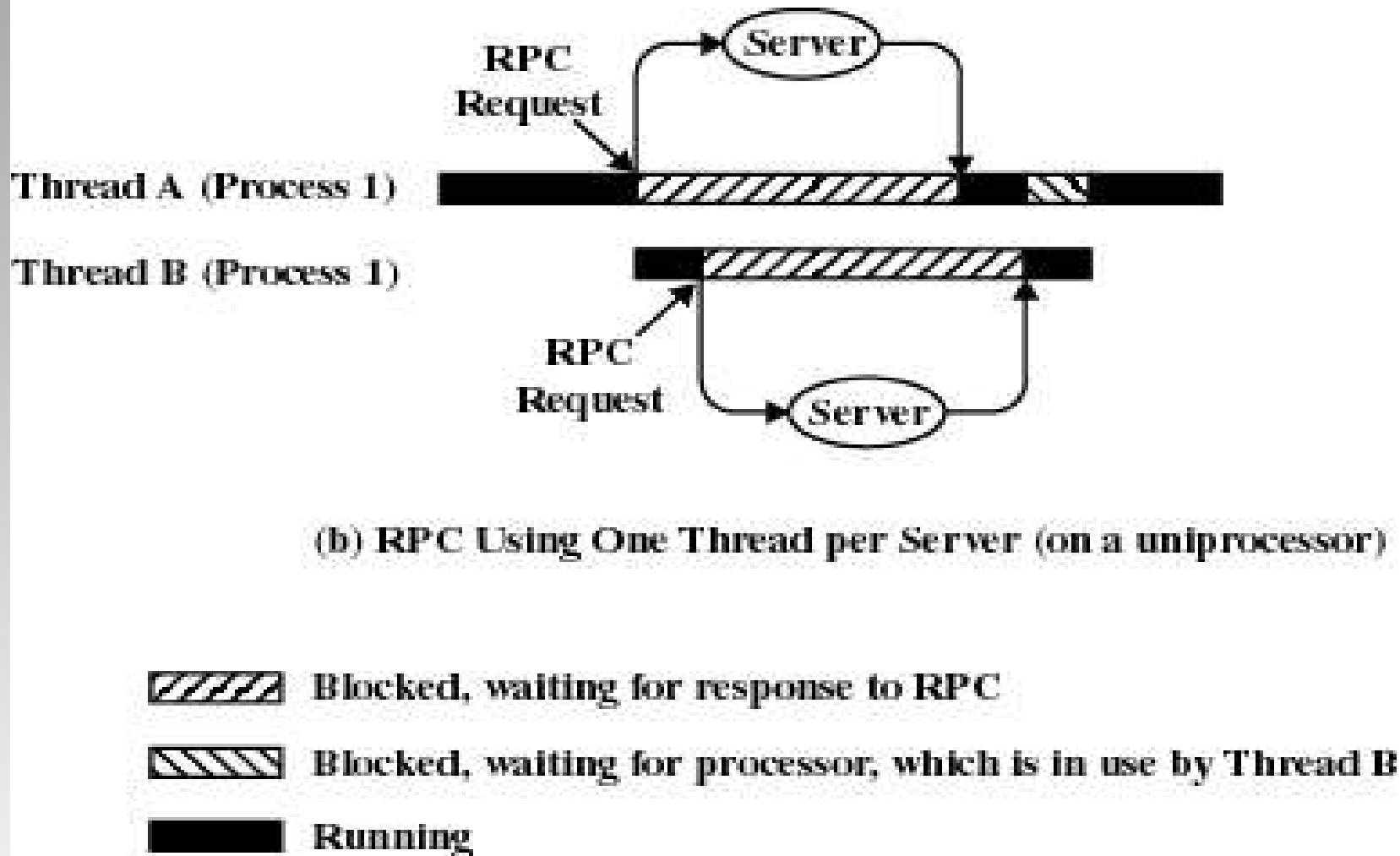


Figure 4.3 Remote Procedure Call (RPC) Using Threads

# *User-Level Threads*

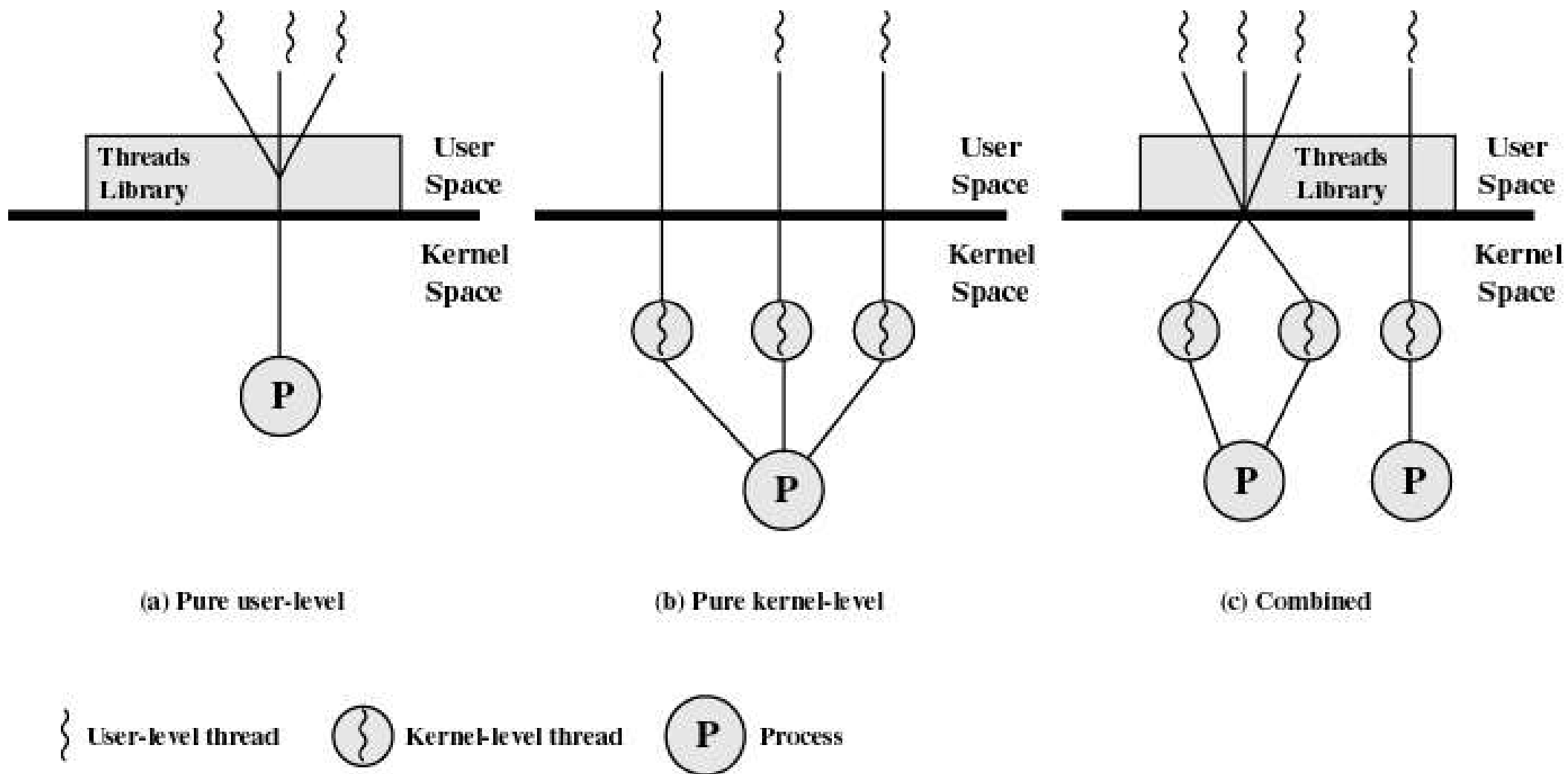
- ◆ All thread management is done by the application
- ◆ The kernel is not aware of the existence of threads

# *Kernel-Level Threads*

- ◆ W2K, Linux, and OS/2 are examples of this approach
- ◆ Kernel maintains context information for the process and the threads
- ◆ Scheduling is done on a thread basis

# *Combined Approaches*

- ◆ Example is Solaris
- ◆ Thread creation done in the user space
- ◆ Bulk of scheduling and synchronization of threads done in the user space



**Figure 4.6 User-Level and Kernel-Level Threads**

# *Relationship Between Threads and Processes*

<b>Threads:Process</b>	<b>Description</b>	<b>Example Systems</b>
<b>1:1</b>	<b>Each thread of execution is a unique process with its own address space and resources.</b>	<b>Traditional UNIX implementations</b>
<b>M:1</b>	<b>A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.</b>	<b>Windows NT, Solaris, OS/2, OS/390, MACH</b>

# *Relationship Between Threads and Processes*

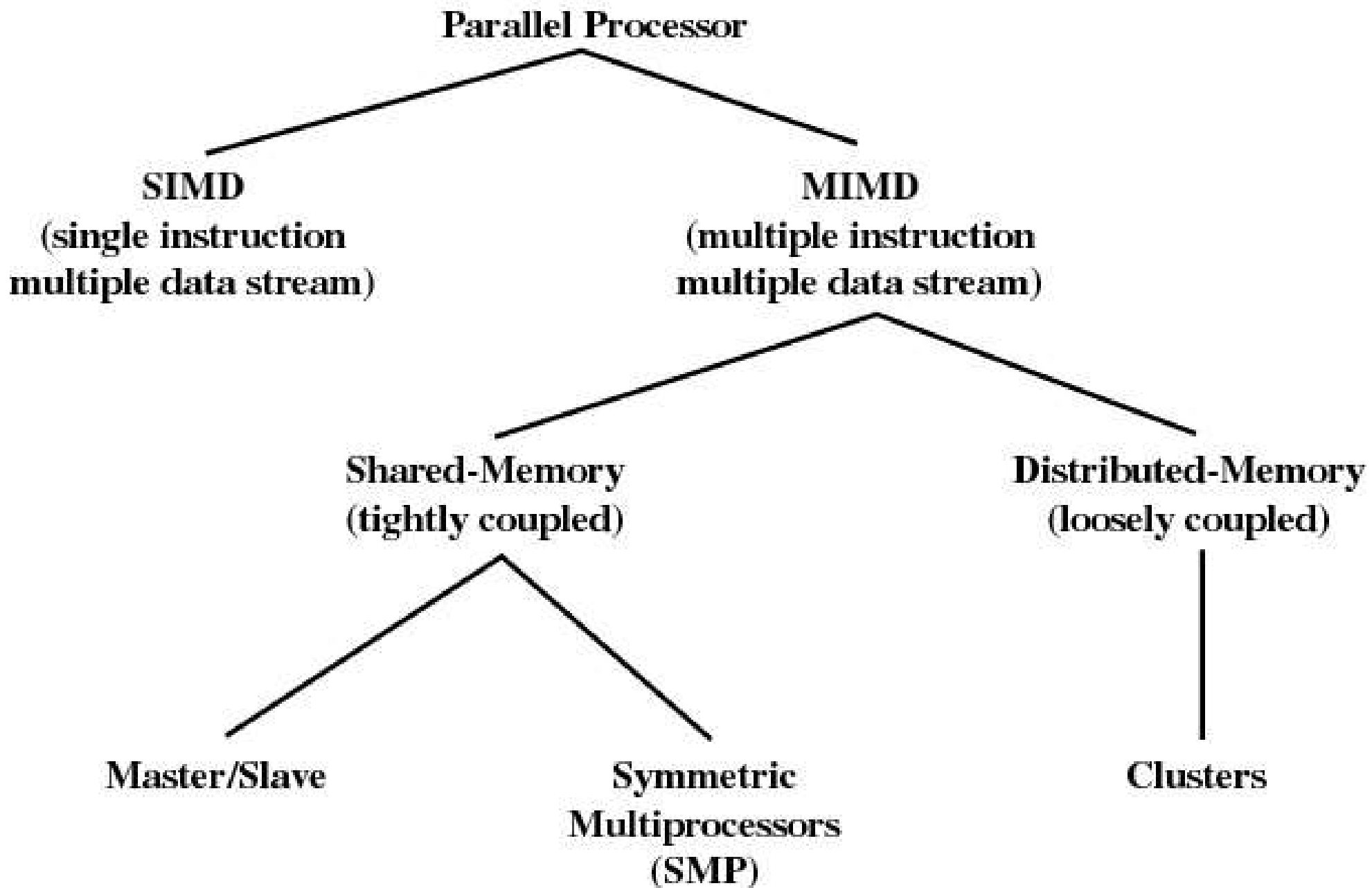
<b>Threads:Process</b>	<b>Description</b>	<b>Example Systems</b>
<b>1:M</b>	<b>A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.</b>	<b>Ra (Clouds), Emerald</b>
<b>M:M</b>	<b>Combines attributes of M:1 and 1:M cases</b>	<b>TRIX</b>

# *Categories of Computer Systems*

- ◆ Single Instruction Single Data (SISD)
  - ◆ single processor executes a single instruction stream to operate on data stored in a single memory
- ◆ Single Instruction Multiple Data (SIMD)
  - ◆ each instruction is executed on a different set of data by the different processors

# *Categories of Computer Systems*

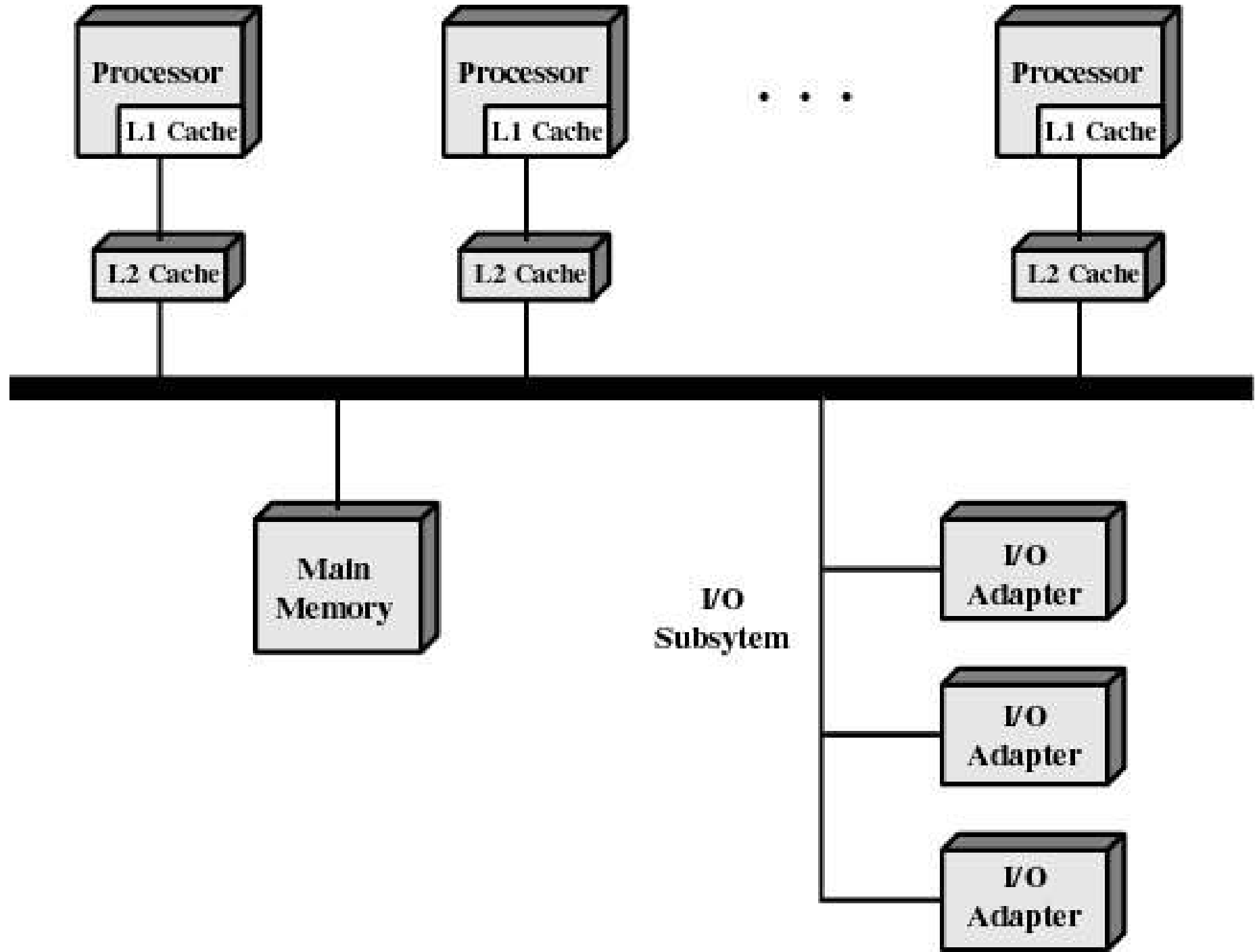
- ◆ Multiple Instruction Single Data (MISD)
  - ◆ a sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. Never implemented
- ◆ Multiple Instruction Multiple Data (MIMD)
  - ◆ a set of processors simultaneously execute different instruction sequences on different data sets



**Figure 4.7 Parallel Processor Architectures**

# *Symmetric Multiprocessing*

- ◆ Kernel can execute on any processor
- ◆ Typically each processor does self-scheduling from the pool of available process or threads



**Figure 4.9 Symmetric Multiprocessor Organization**

# ***Multiprocessor Operating System Design Considerations***

- ◆ Simultaneous concurrent processes or threads
- ◆ Scheduling
- ◆ Synchronization
- ◆ Memory Management
- ◆ Reliability and Fault Tolerance

# *Microkernels*

- ◆ Small operating system core
- ◆ Contains only essential operating systems functions
- ◆ Many services traditionally included in the operating system are now external subsystems
  - ◆ device drivers
  - ◆ file systems
  - ◆ virtual memory manager
  - ◆ windowing system
  - ◆ security services

# ***Benefits of a Microkernel Organization***

- ◆ Uniform interface on request made by a process
  - ◆ All services are provided by means of message passing
- ◆ Extensibility
  - ◆ Allows the addition of new services
- ◆ Flexibility
  - ◆ New features added
  - ◆ Existing features can be subtracted

# *Benefits of a Microkernel Organization*

- ◆ Portability
  - ◆ Changes needed to port the system to a new processor is changed in the microkernel - not in the other services
- ◆ Reliability
  - ◆ Modular design
  - ◆ Small microkernel can be rigorously tested

# *Benefits of Microkernel Organization*

- ◆ Distributed system support
  - ◆ Message are sent without knowing what the target machine is
- ◆ Object-oriented operating system
  - ◆ Components are objects with clearly defined interfaces that can be interconnected to form software

# *Microkernel Design*

- ◆ Low-level memory management
  - ◆ mapping each virtual page to a physical page frame
- ◆ Inter-process communication
- ◆ I/O and interrupt management

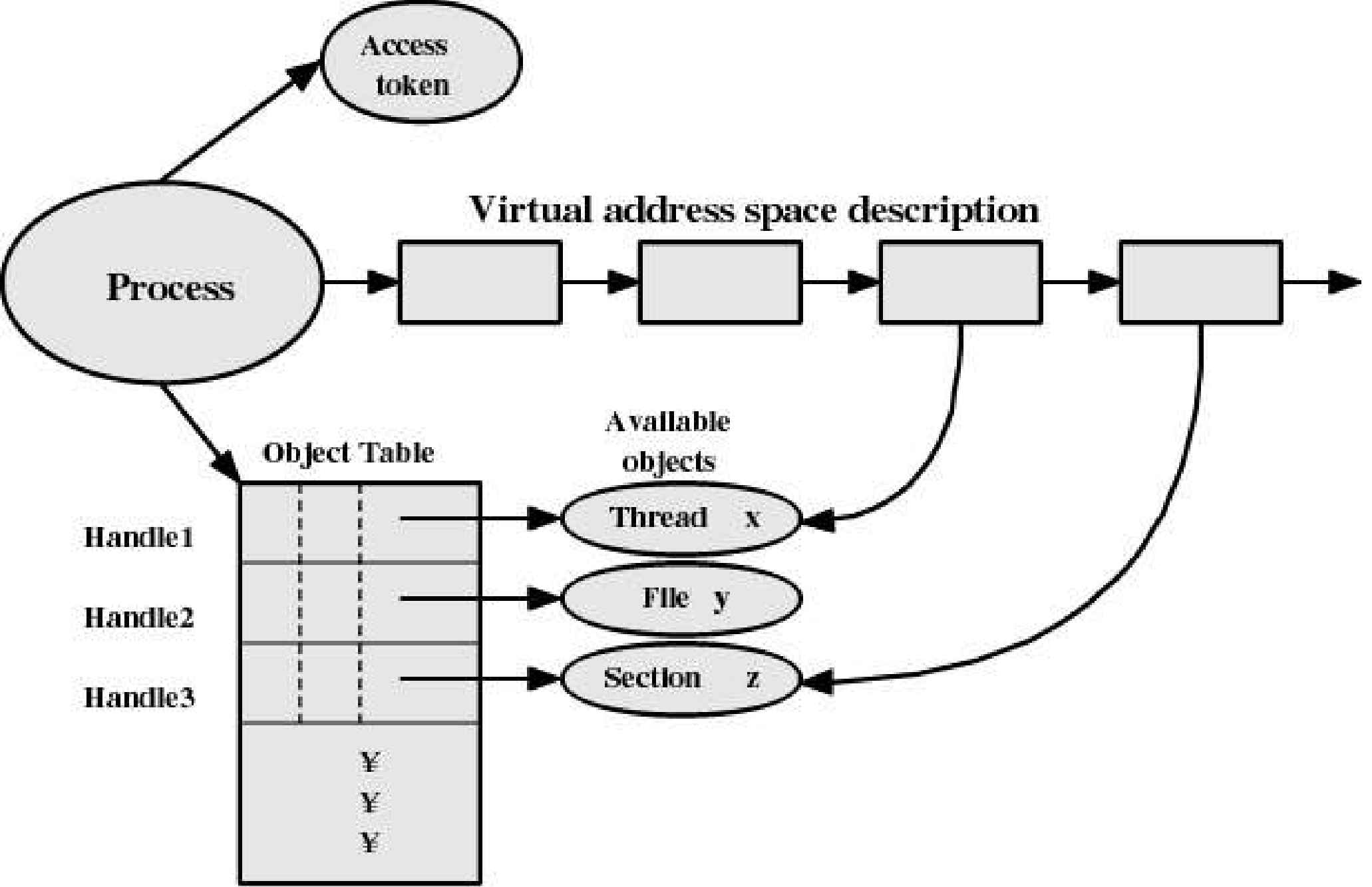
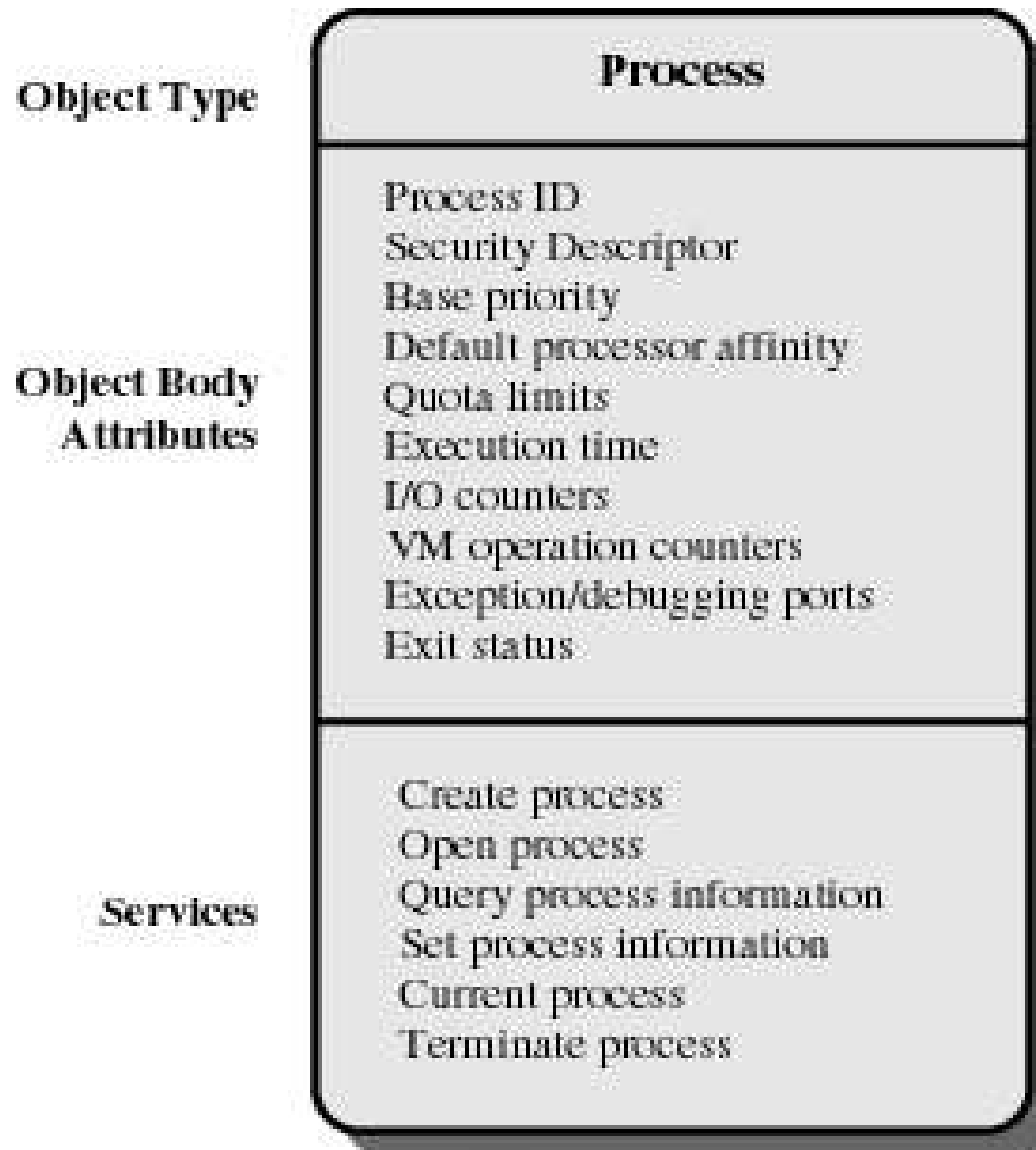


Figure 4.12 Windows 2000 Process and Its Resources

# *Windows 2000 Process Object*



(a) Process object

# Windows 2000 Thread Object

Object Type

**Thread**

Object Body  
Attributes

Thread ID  
Thread context  
Dynamic priority  
Base priority  
Thread processor affinity  
Thread execution time  
Alert status  
Suspension count  
Impersonation token  
Termination port  
Thread exit status

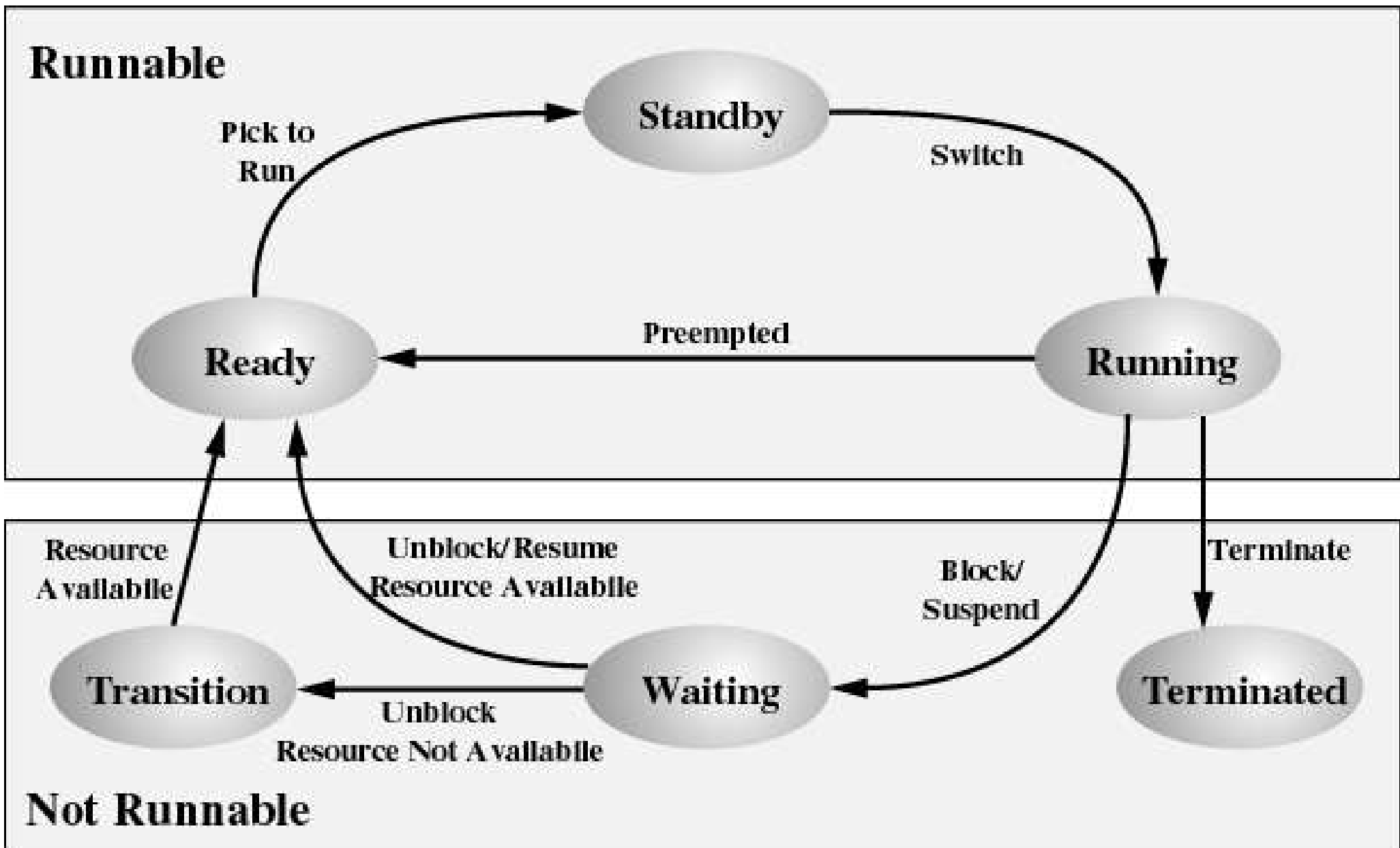
Services

Create thread  
Open thread  
Query thread information  
Set thread information  
Current thread  
Terminate thread  
Get context  
Set context  
Suspend  
Resume  
Alert thread  
Test thread alert  
Register termination port

(b) Thread object

# *Windows 2000 Thread States*

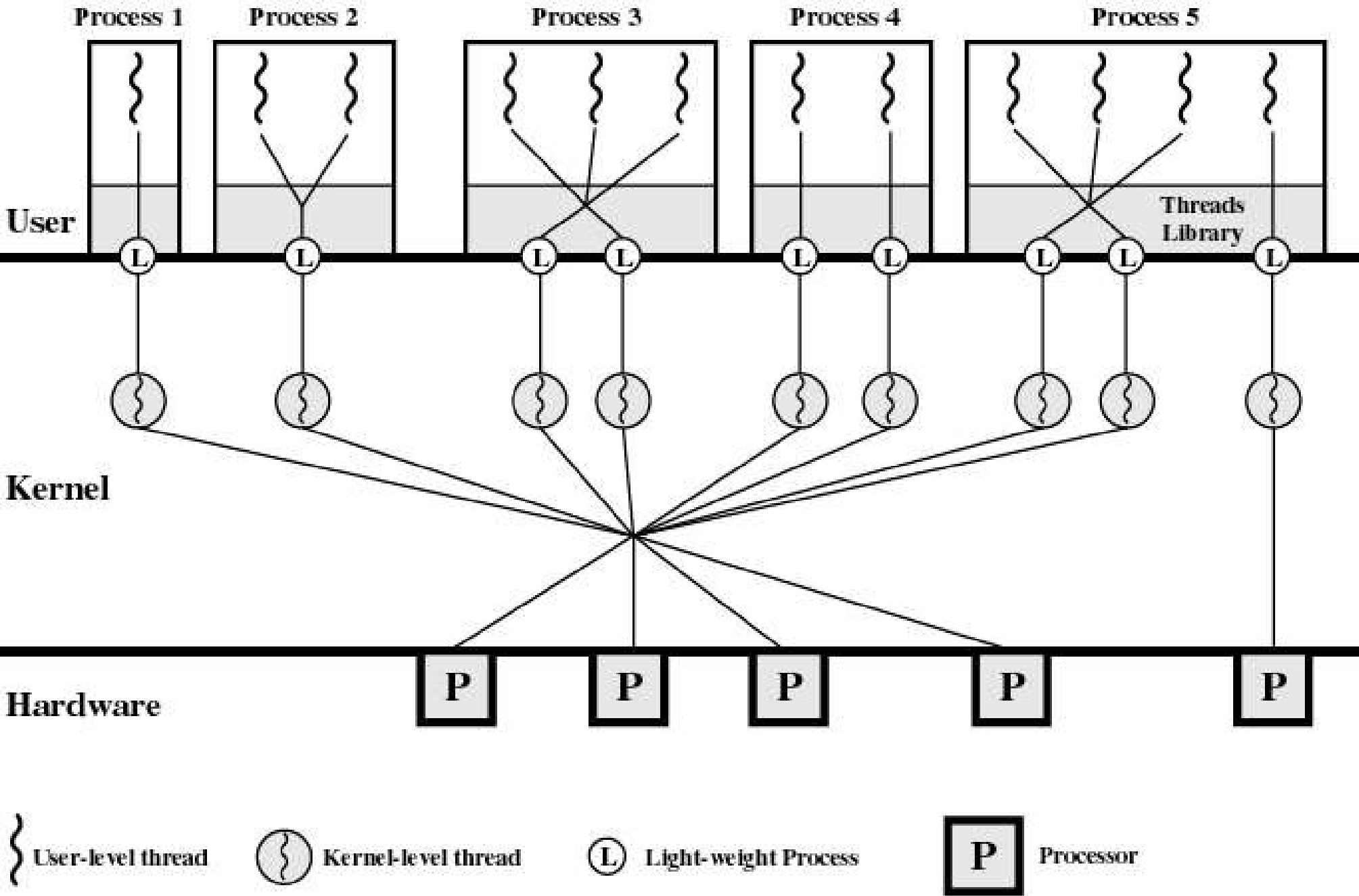
- ◆ Ready
- ◆ Standby
- ◆ Running
- ◆ Waiting
- ◆ Transition
- ◆ Terminated



**Figure 4.14 Windows 2000 Thread States**

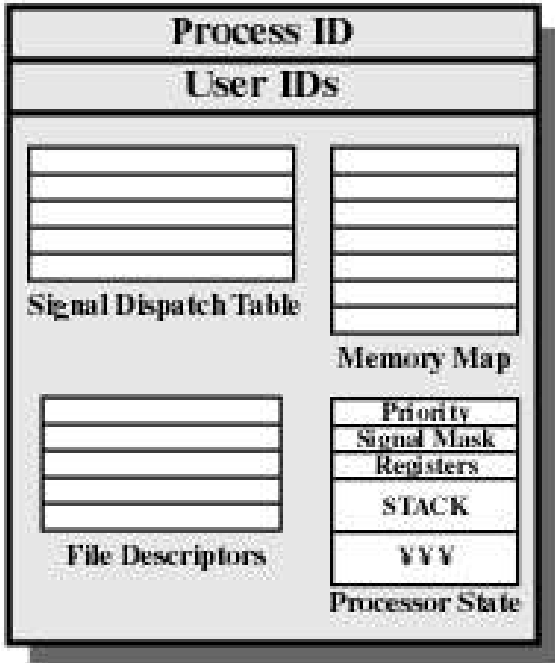
# *Solaris*

- ◆ Process includes the user's address space, stack, and process control block
- ◆ User-level threads
- ◆ Lightweight processes
- ◆ Kernel threads



**Figure 4.15 Solaris Multithreaded Architecture Example**

# UNIX Process Structure



# Solaris Process Structure

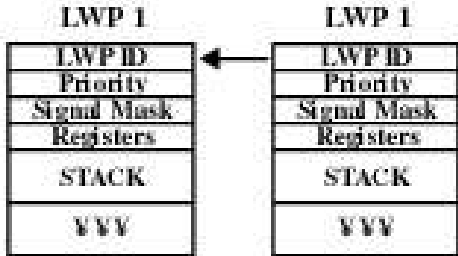
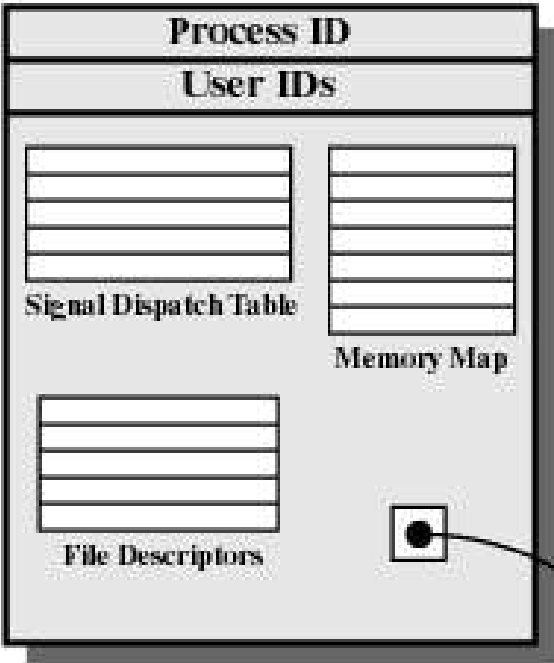


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

# *Solaris Thread Execution*

- ◆ Synchronization
- ◆ Suspension
- ◆ Preemption
- ◆ Yielding

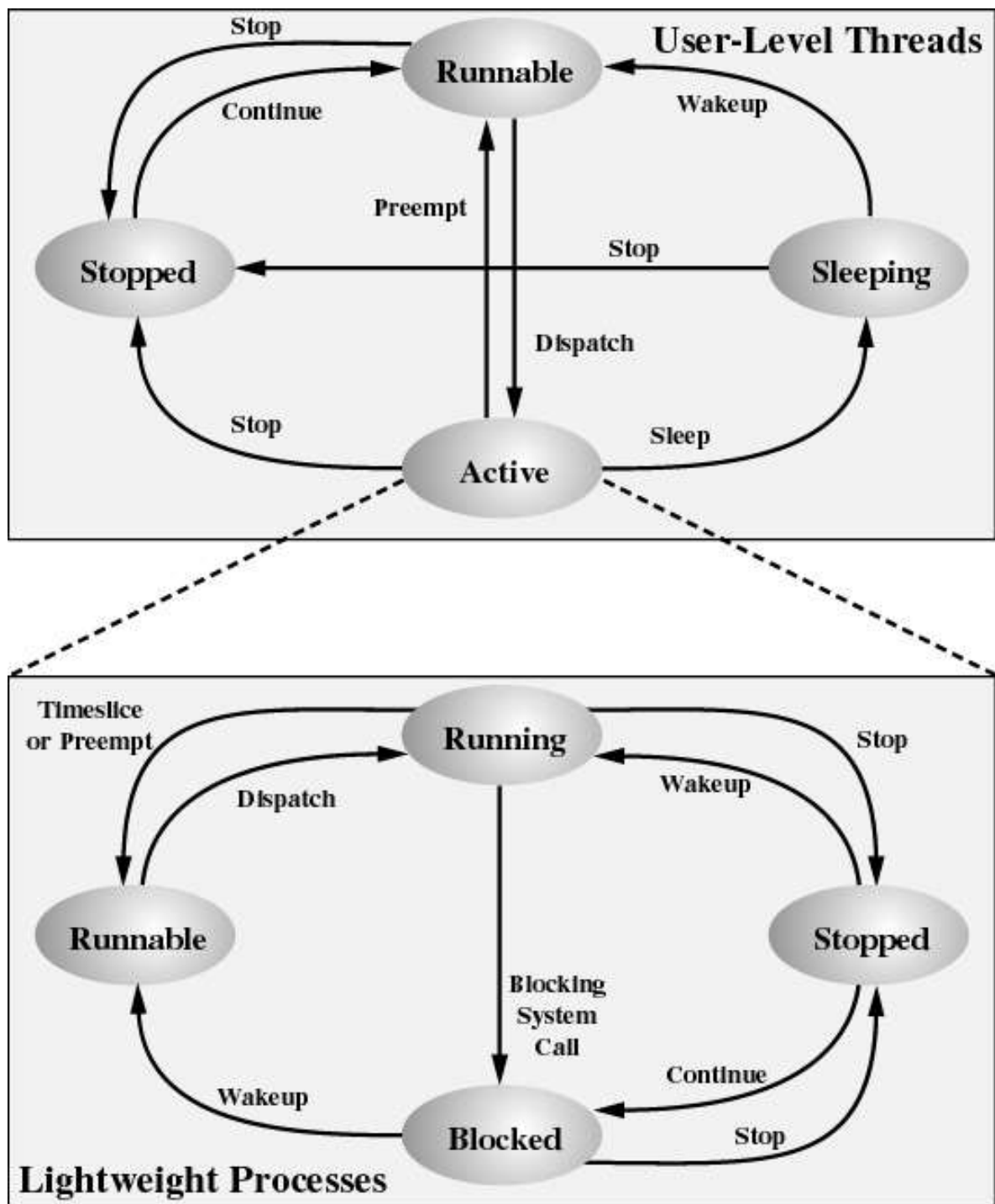


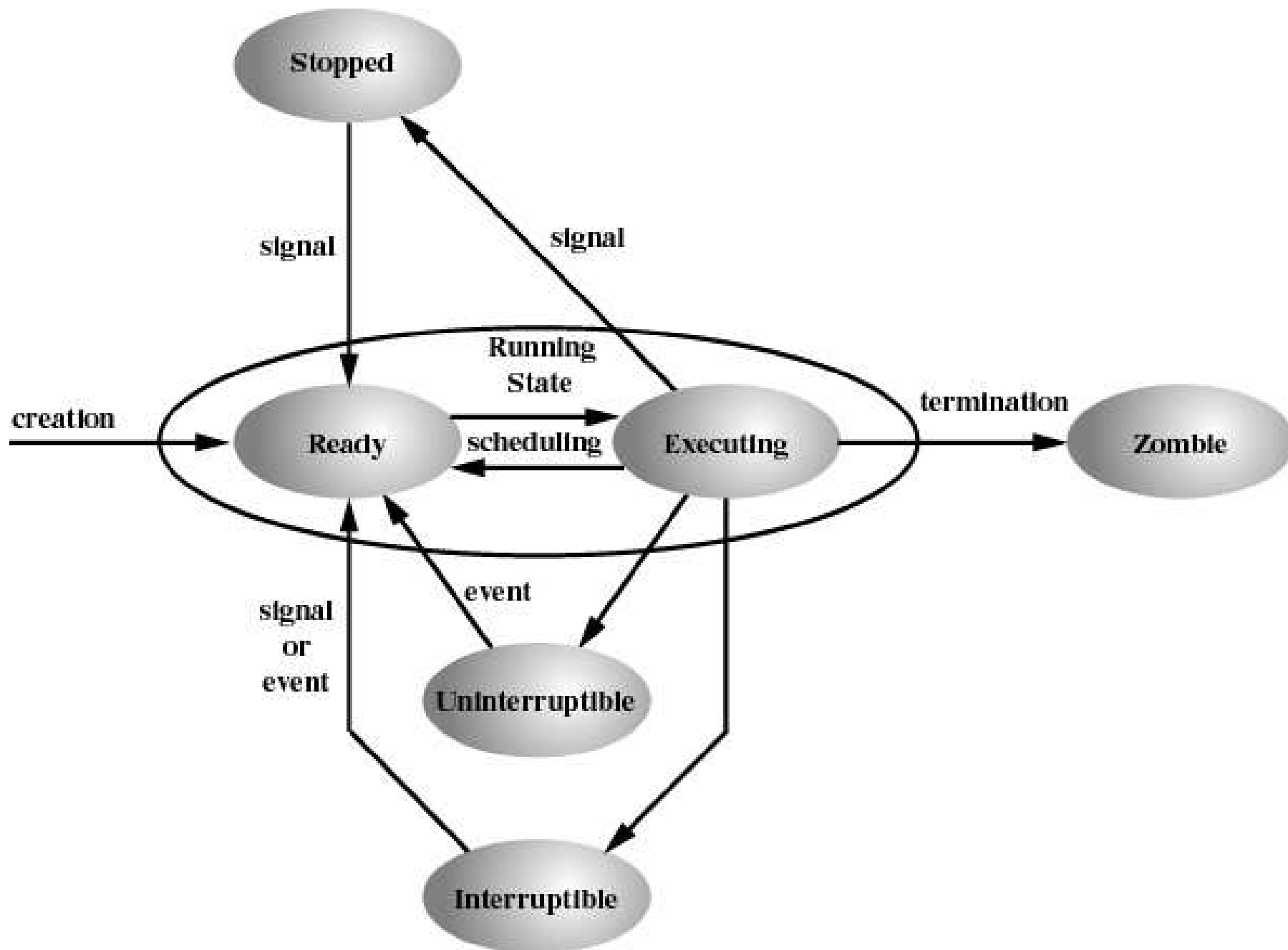
Figure 4.17 Solaris User-Level Thread and LWP States

# *Linux Process*

- ◆ State
- ◆ Scheduling information
- ◆ Identifiers
- ◆ Interprocess communication
- ◆ Links
- ◆ Times and timers
- ◆ File system
- ◆ Virtual memory
- ◆ Processor-specific context

# *Linux States of a Process*

- ◆ Running
- ◆ Interruptable
- ◆ Uninterruptable
- ◆ Stopped
- ◆ Zombie



**Figure 4.18 Linux Process/Thread Model**