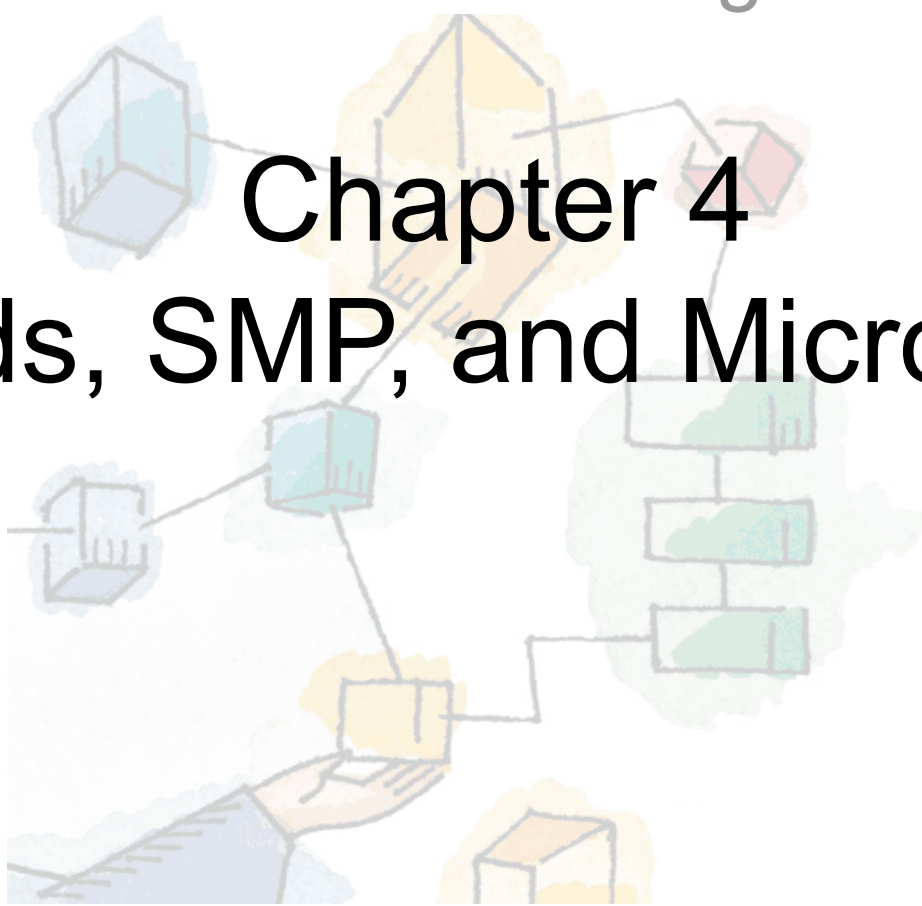
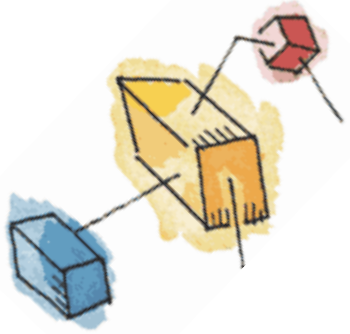


*Operating Systems:  
Internals and Design Principles, 6/E*  
William Stallings



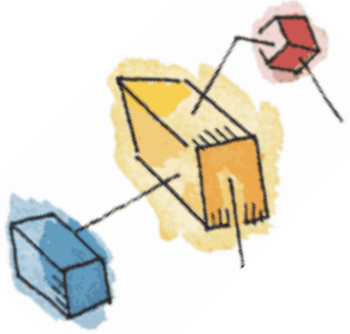
**Chapter 4**  
**Threads, SMP, and Microkernels**



# Processes and Threads

- Resource ownership - process includes 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



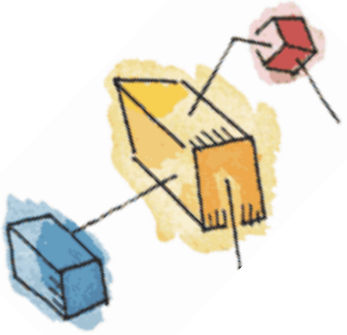


# Processes and Threads

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

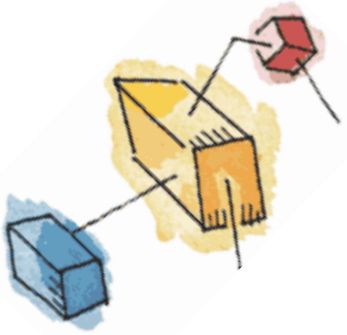


# Multithreading



- OS supports multiple, concurrent paths of execution within a single process

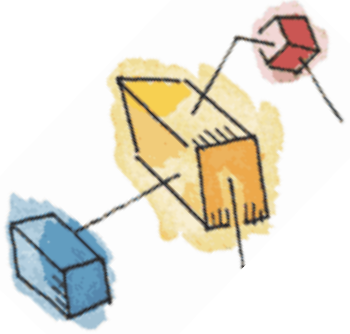




# Multithreading

- MS-DOS supports a single user process and thread
- UNIX supports multiple user processes but only supports one thread per process
- Java run-time environment is a single process with multiple threads





# Threads and Processes

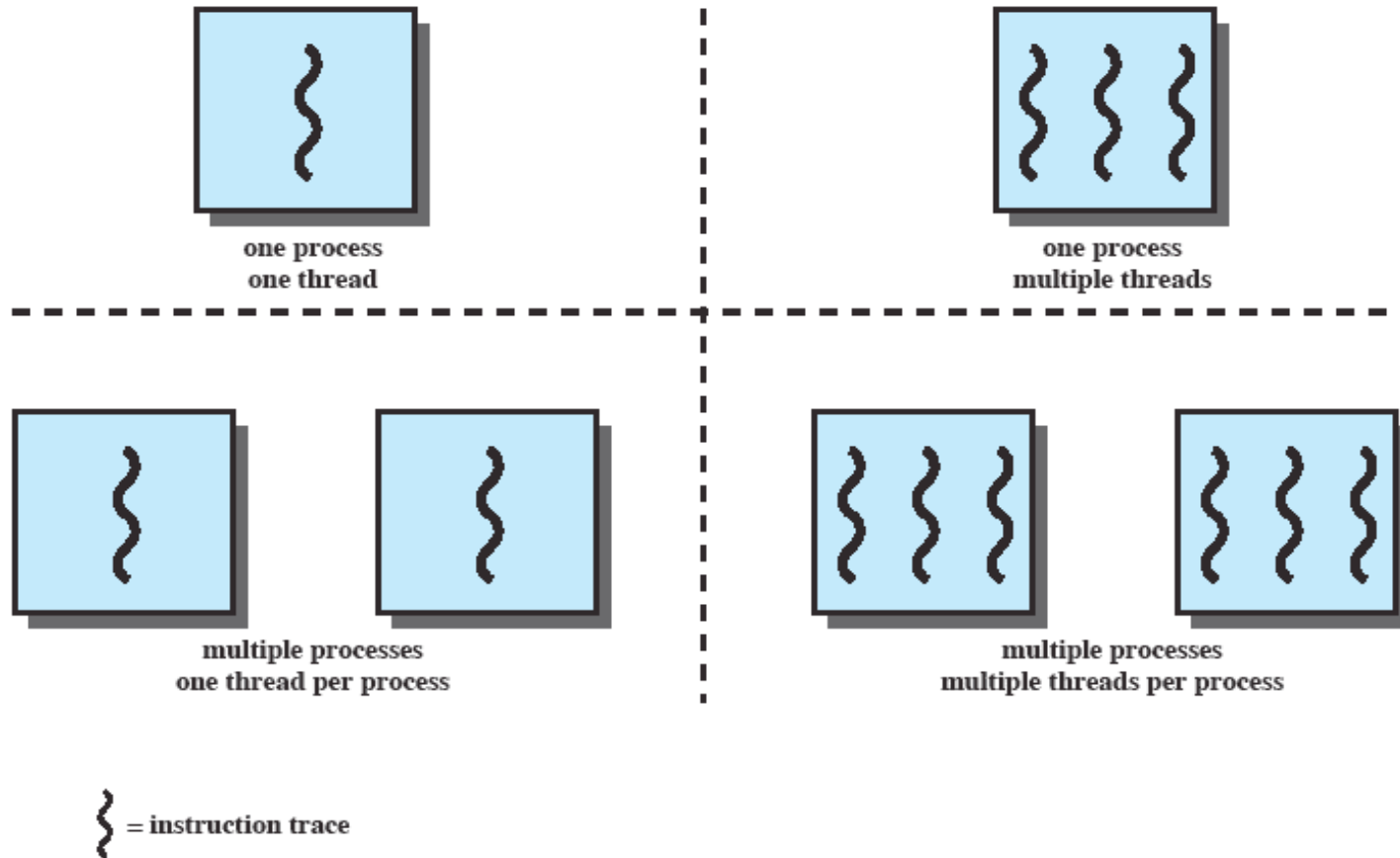
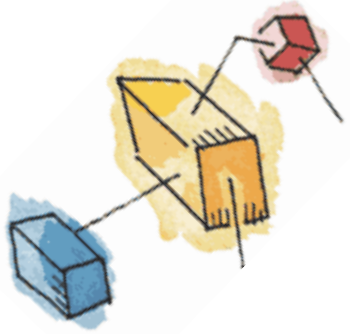


Figure 4.1 Threads and Processes [ANDE97]





# Processes

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





# One or More Threads in Process

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





# One or More Threads in Process

- Some per-thread static storage for local variables
- Access to the memory and resources of its process
  - all threads of a process share this



# Threads

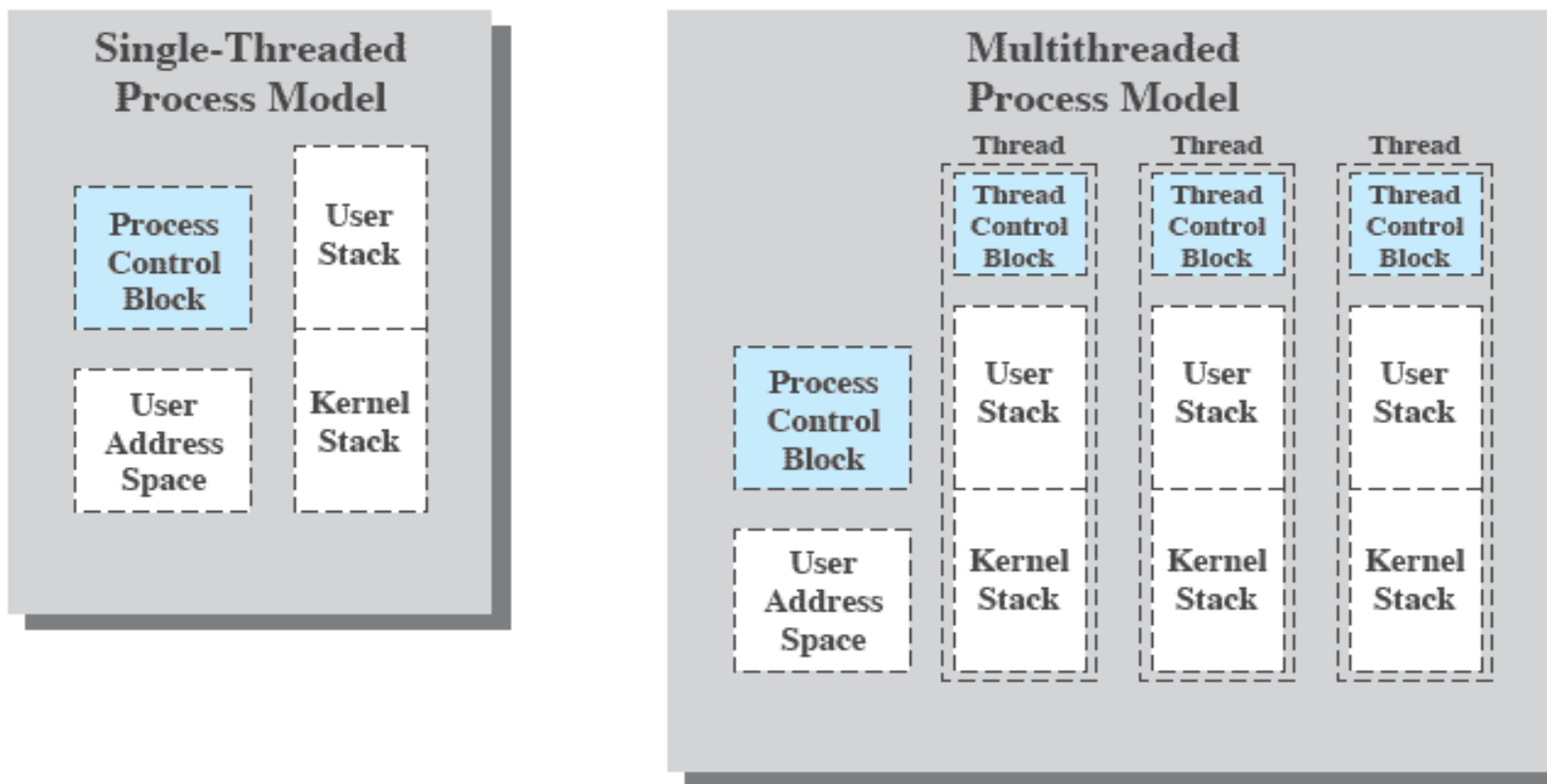
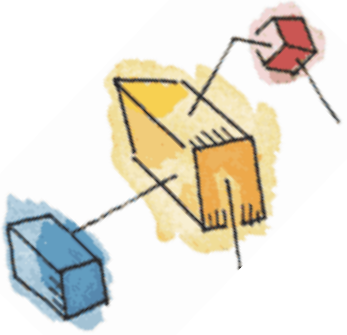


Figure 4.2 Single Threaded and Multithreaded Process Models

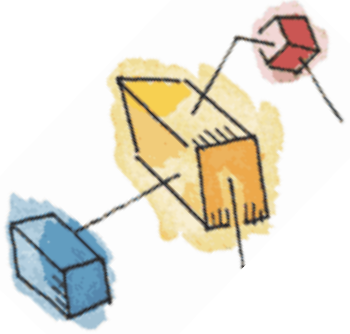




# Benefits of Threads

- 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

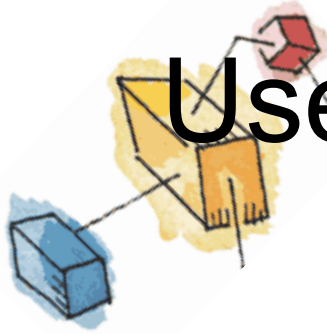




# Benefits of Threads

- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

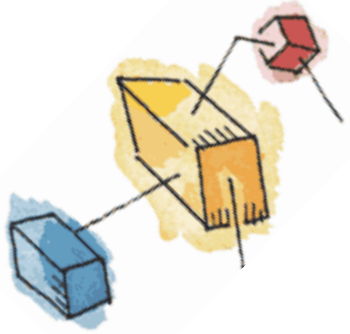




# Uses of Threads in a Single-User Multiprocessing System

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

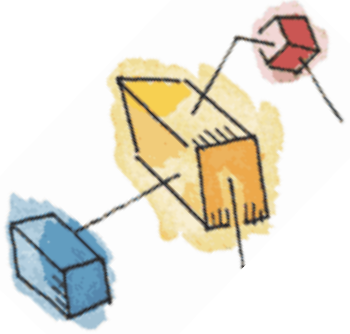




# Threads

- 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

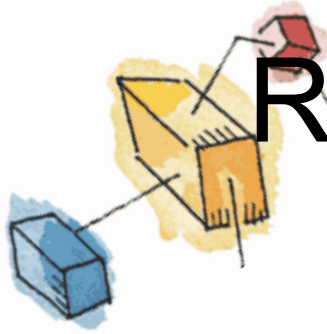




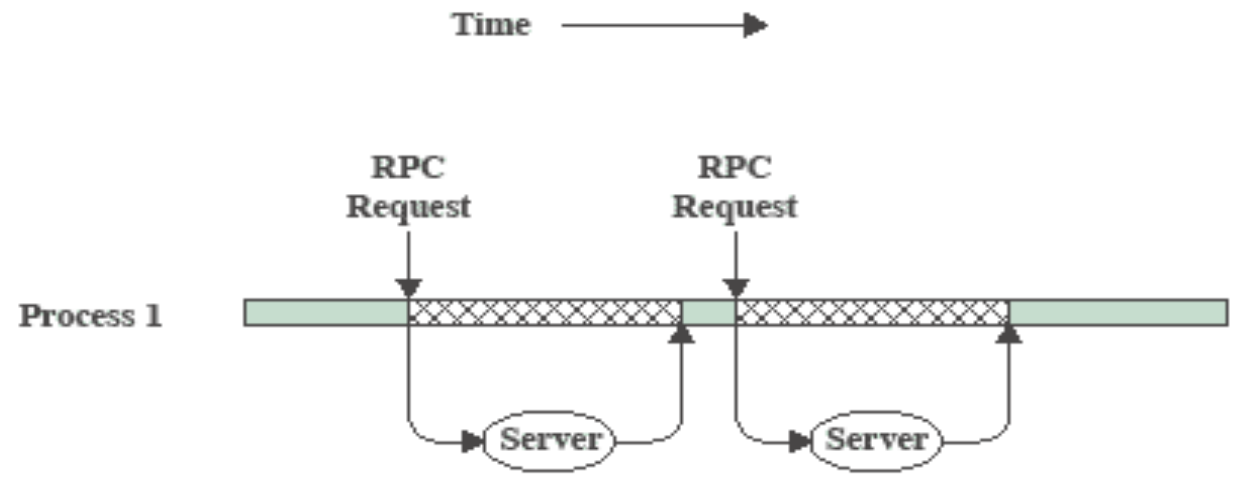
# Thread States

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



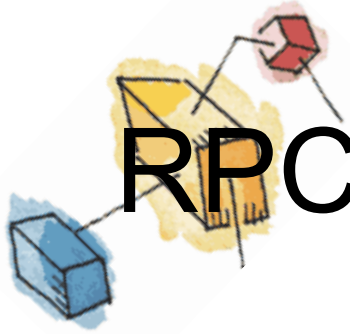


# Remote Procedure Call Using Single Thread

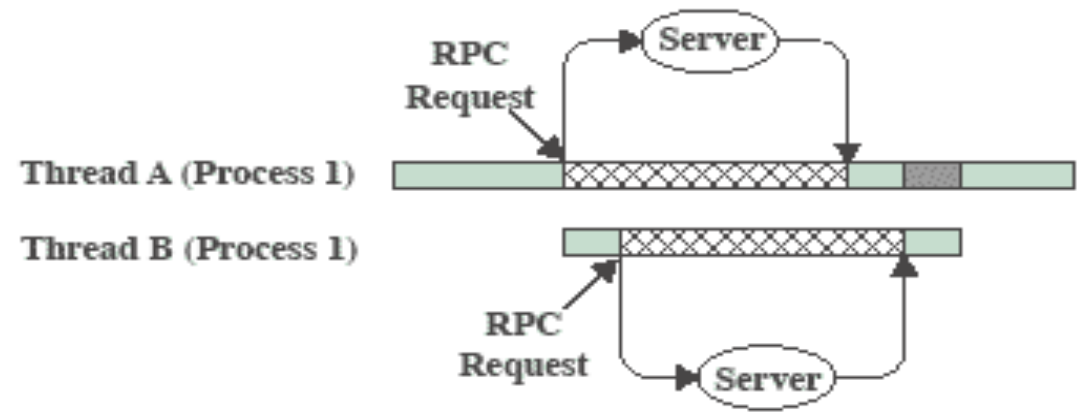


(a) RPC Using Single Thread








# RPC Using One Thread per Server



(b) RPC Using One Thread per Server (on a uniprocessor)

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



# Multithreading

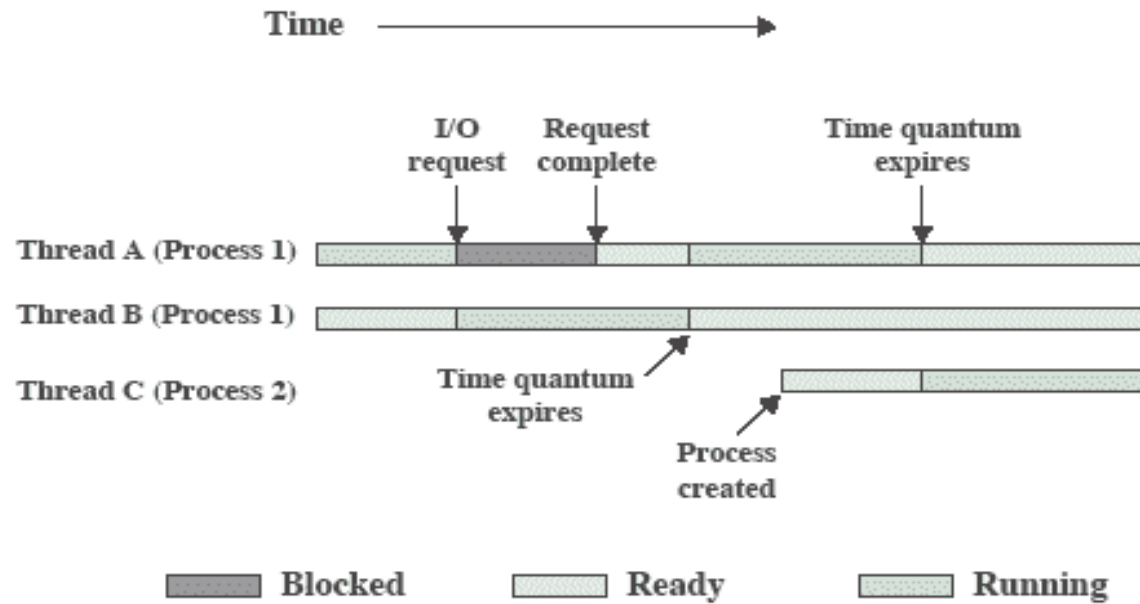


Figure 4.4 Multithreading Example on a Uniprocessor



# Adobe PageMaker

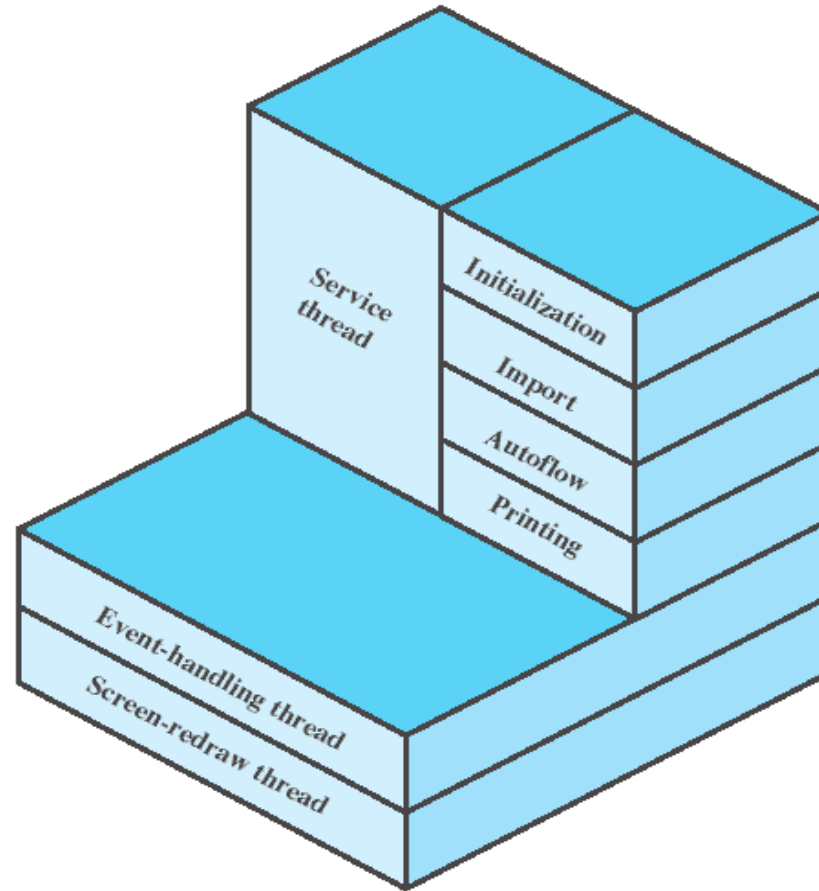
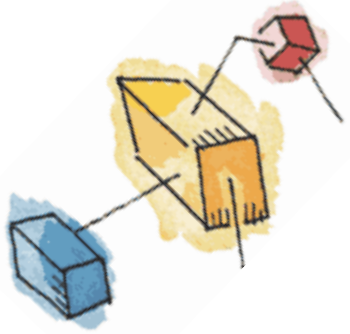


Figure 4.5 Thread Structure for Adobe PageMaker



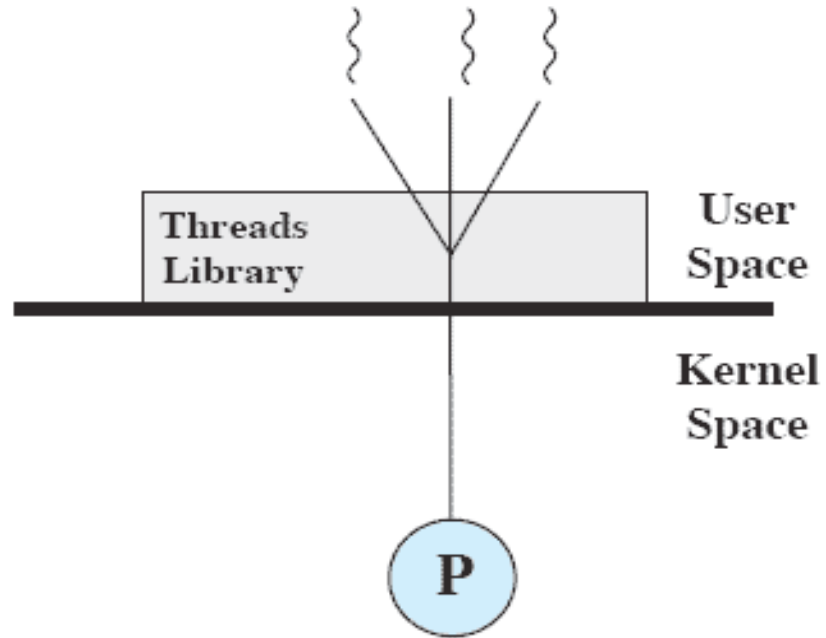


# User-Level Threads

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

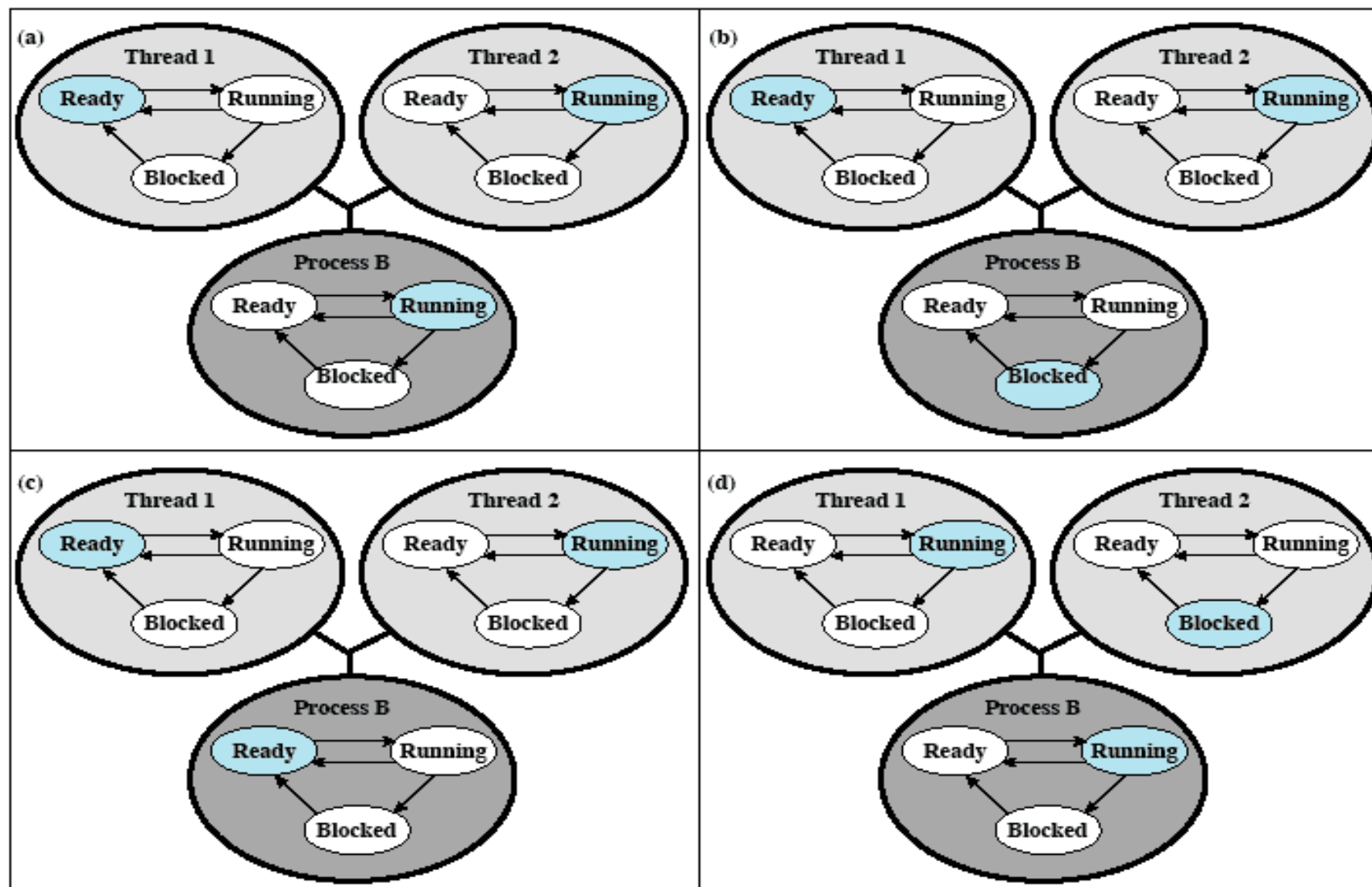


# User-Level Threads



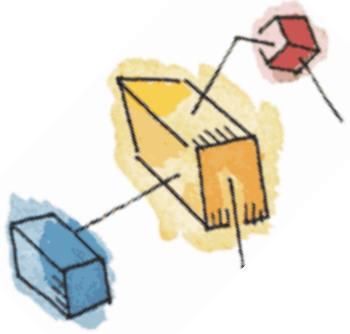
(a) Pure user-level





Colored state  
is current state

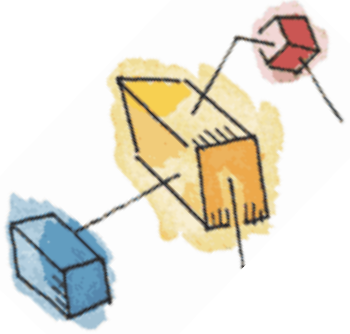
**Figure 4.7** Examples of the Relationships Between User-Level Thread States and Process States



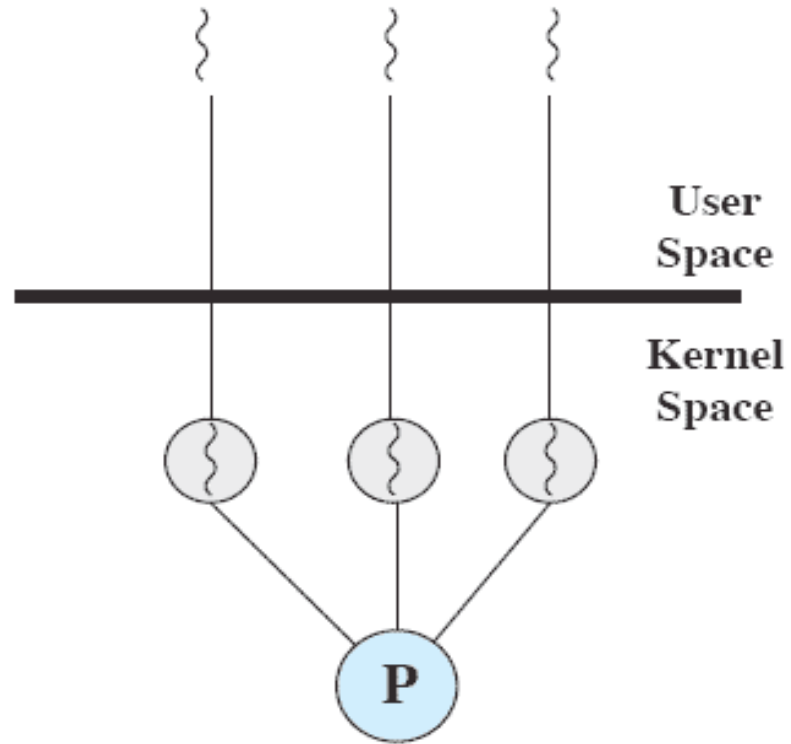
# Kernel-Level Threads

- Windows is an example of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis



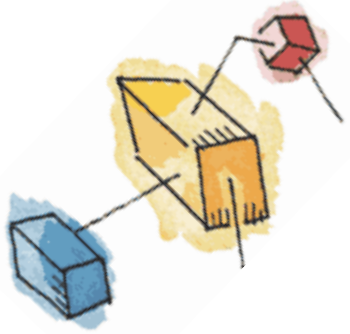


# Kernel-Level Threads



(b) Pure kernel-level

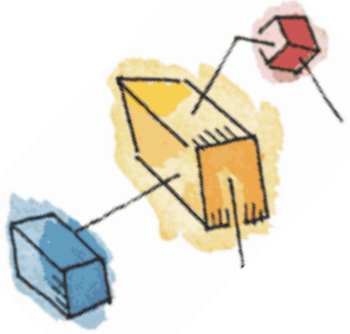




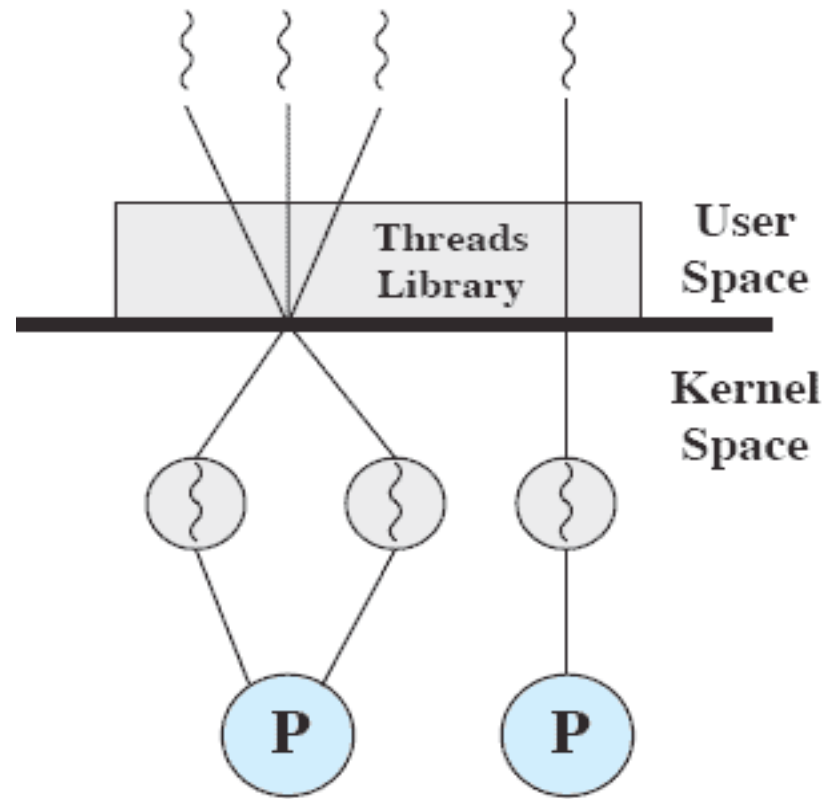
# Combined Approaches

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



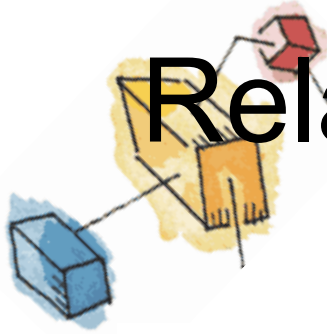


# Combined Approaches



(c) Combined



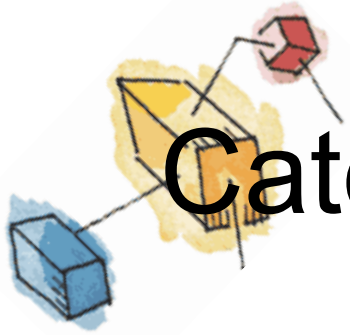


# Relationship Between Thread and Processes

Table 4.2 Relationship Between Threads and Processes

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

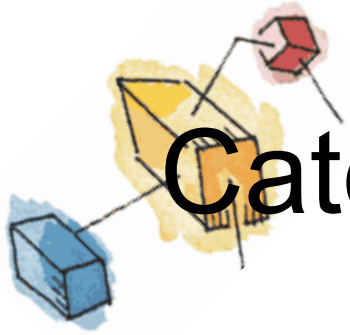




# Categories of Computer Systems

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

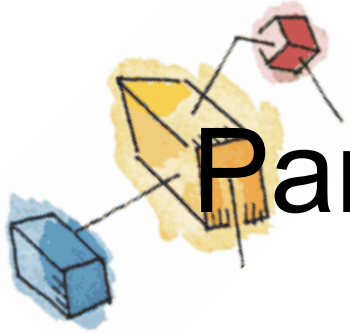




# Categories of Computer Systems

- Multiple Instruction Single Data (MISD) stream
  - 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





# Parallel Processor Architectures

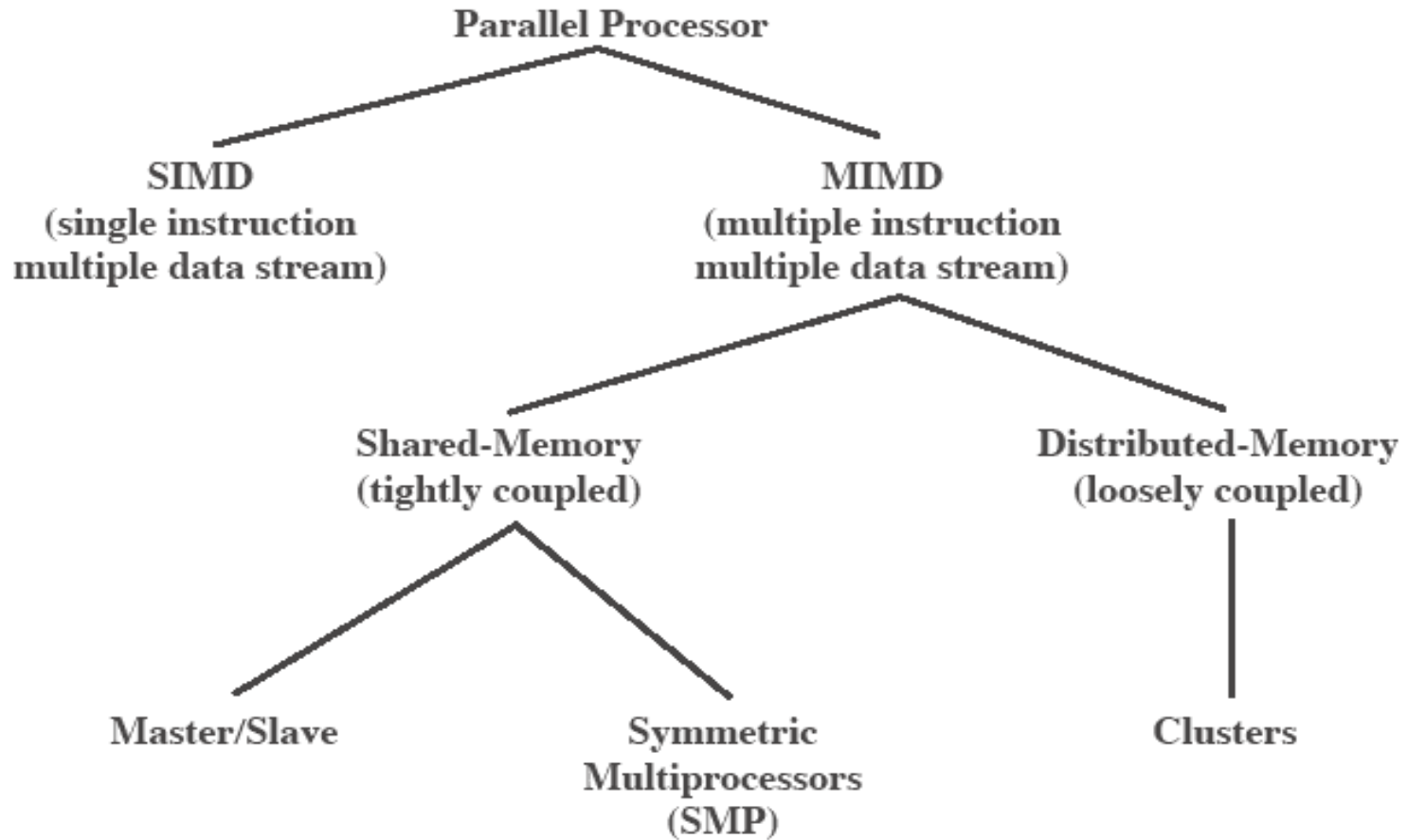
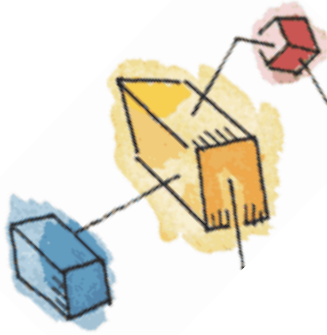


Figure 4.8 Parallel Processor Architectures

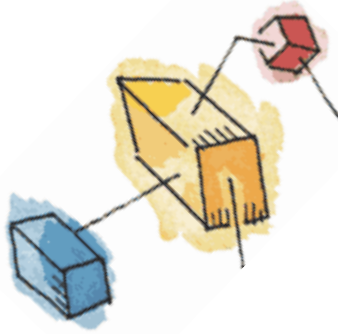




# Symmetric Multiprocessing

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





# Symmetric Multiprocessor Organization

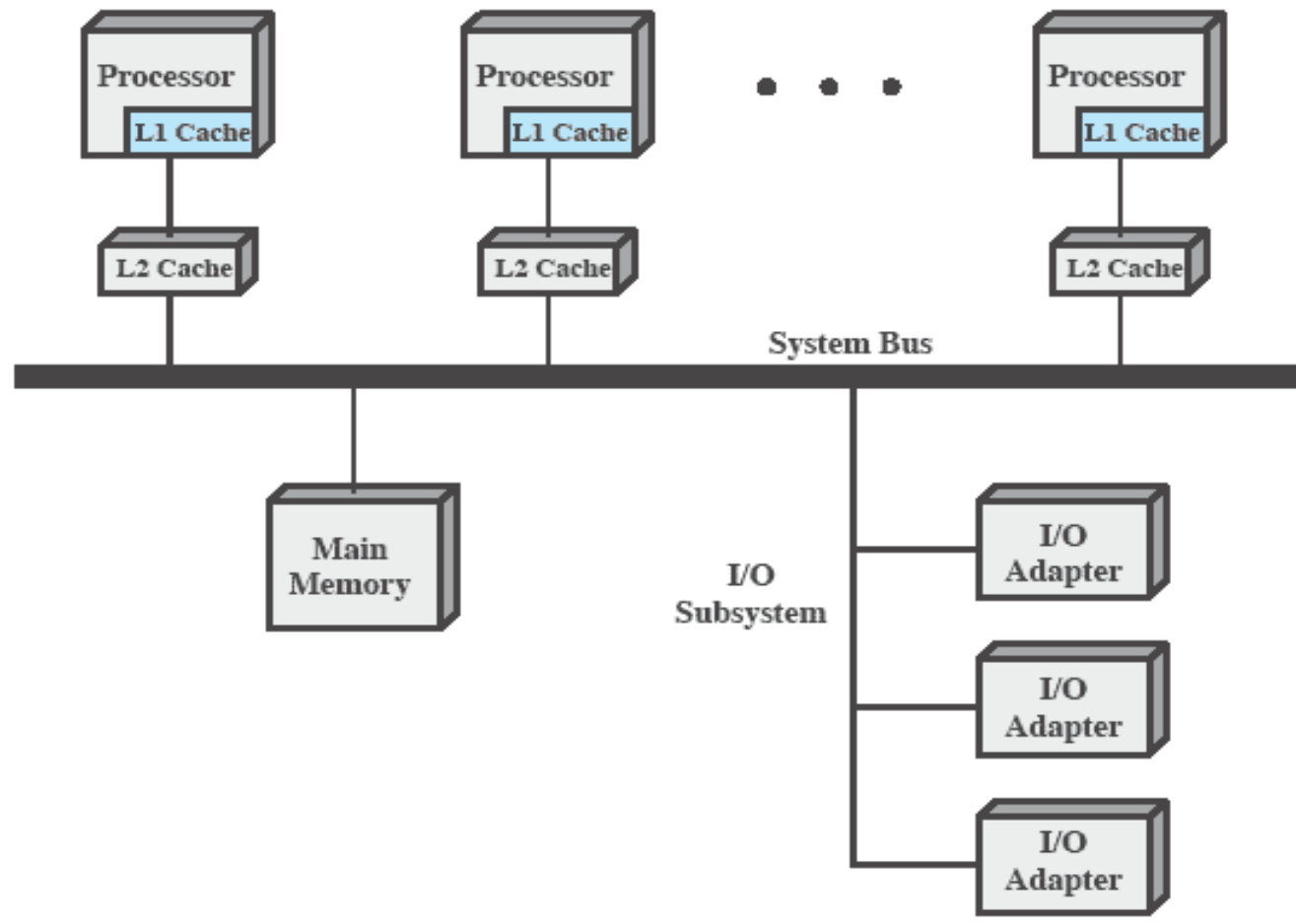
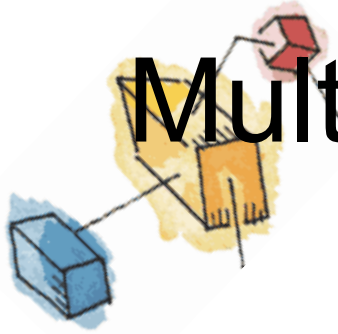


Figure 4.9 Symmetric Multiprocessor Organization

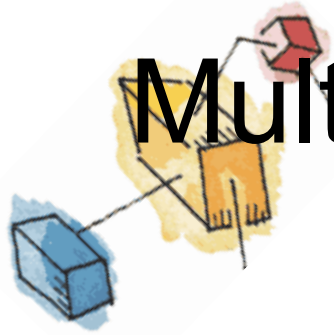




# Multiprocessor Operating Systems Design Considerations

- Simultaneous concurrent processes or threads
- Scheduling
- Synchronization





# Multiprocessor Operating Systems Design Considerations

- Memory management
- Reliability and fault tolerance



# Kernel Architecture

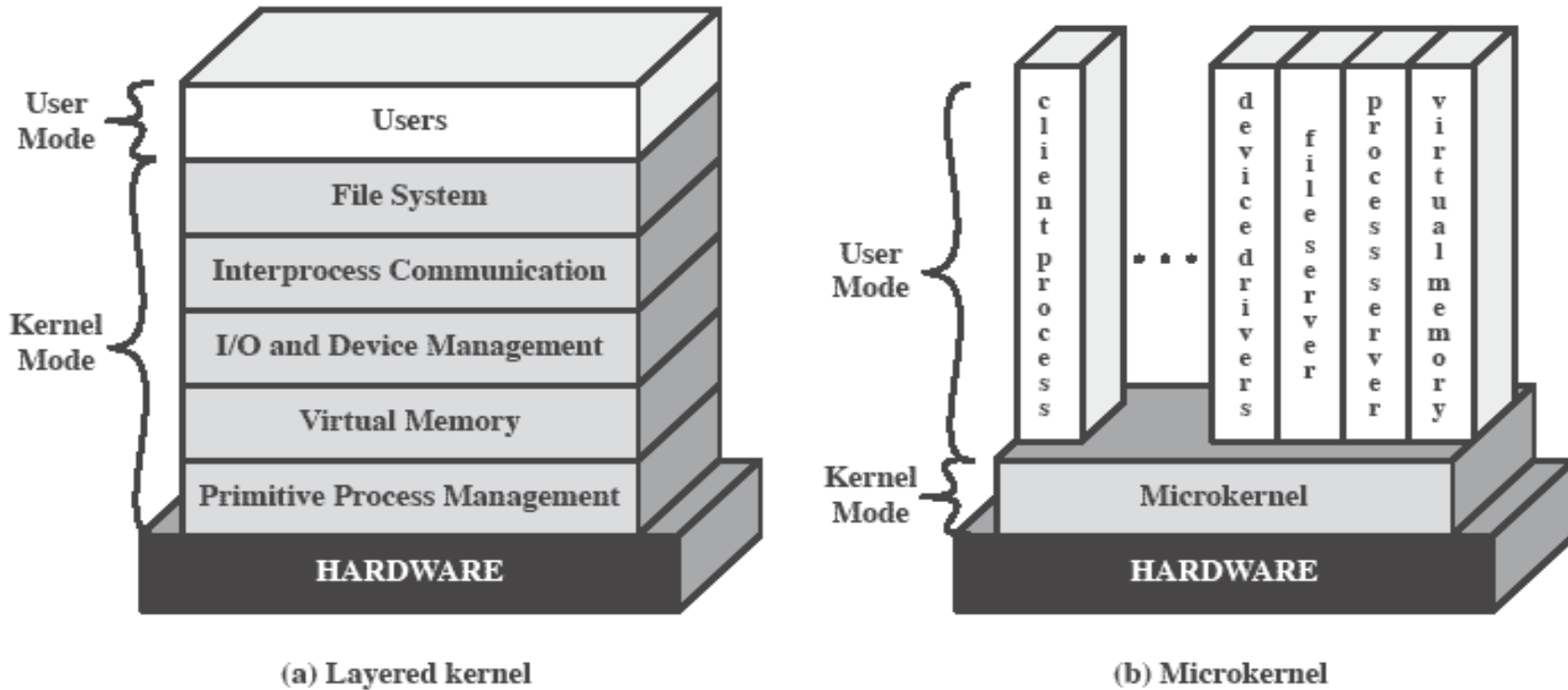
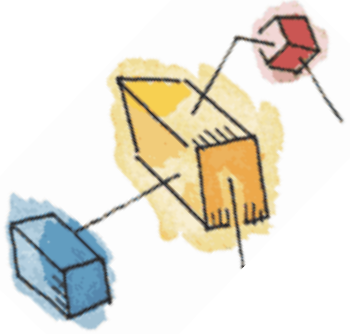


Figure 4.10 Kernel Architecture

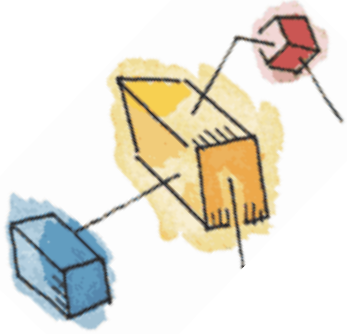




# Microkernels

- Small operating system core
- Contains only essential core operating systems functions

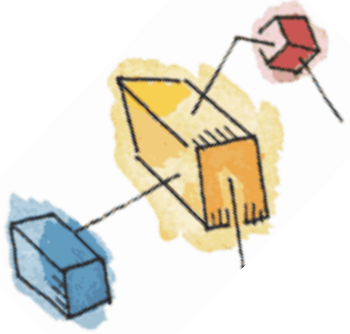




# Benefits of a Microkernel Organization

- Uniform interfaces
  - Don't distinguish between kernel-level and user-level services
  - All services are provided by means of message passing
- Extensibility
- Flexibility
  - New features added
  - Existing features can be subtracted

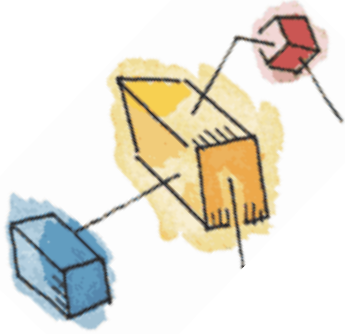




# Benefits of a Microkernel Organization

- Portability
  - Changes needed to port the system to a new processor is changed in the
- Reliability
  - Small microkernel can be rigorously tested

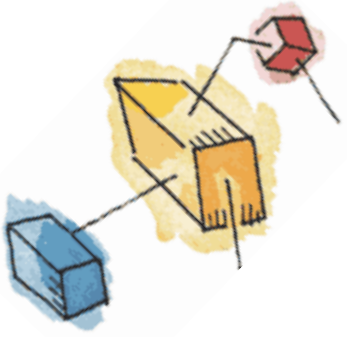




# Benefits of a Microkernel Organization

- Distributed system support
  - Message are sent without knowing what the target machine is
- Object-oriented operating systems
  - 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

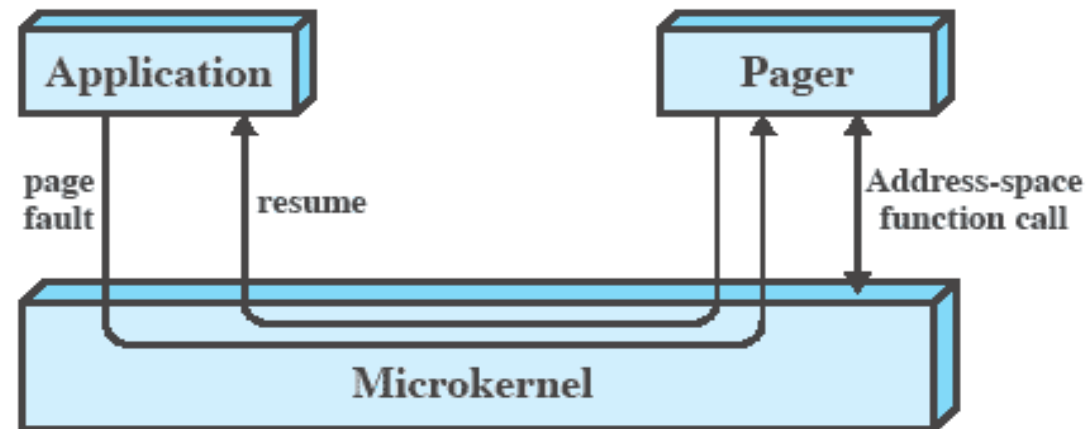
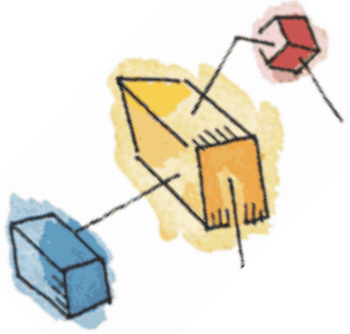


Figure 4.11 Page Fault Processing

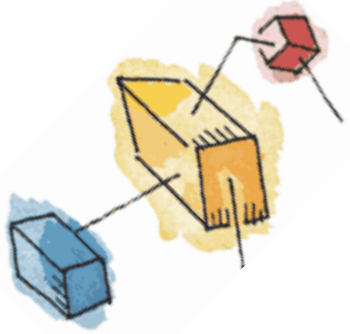




# Microkernel Design

- Interprocess communication
- I/O and interrupt management





# Windows Processes

- Implemented as objects
- An executable process may contain one or more threads
- Both processes and thread objects have built-in synchronization capabilities



# Windows Processes

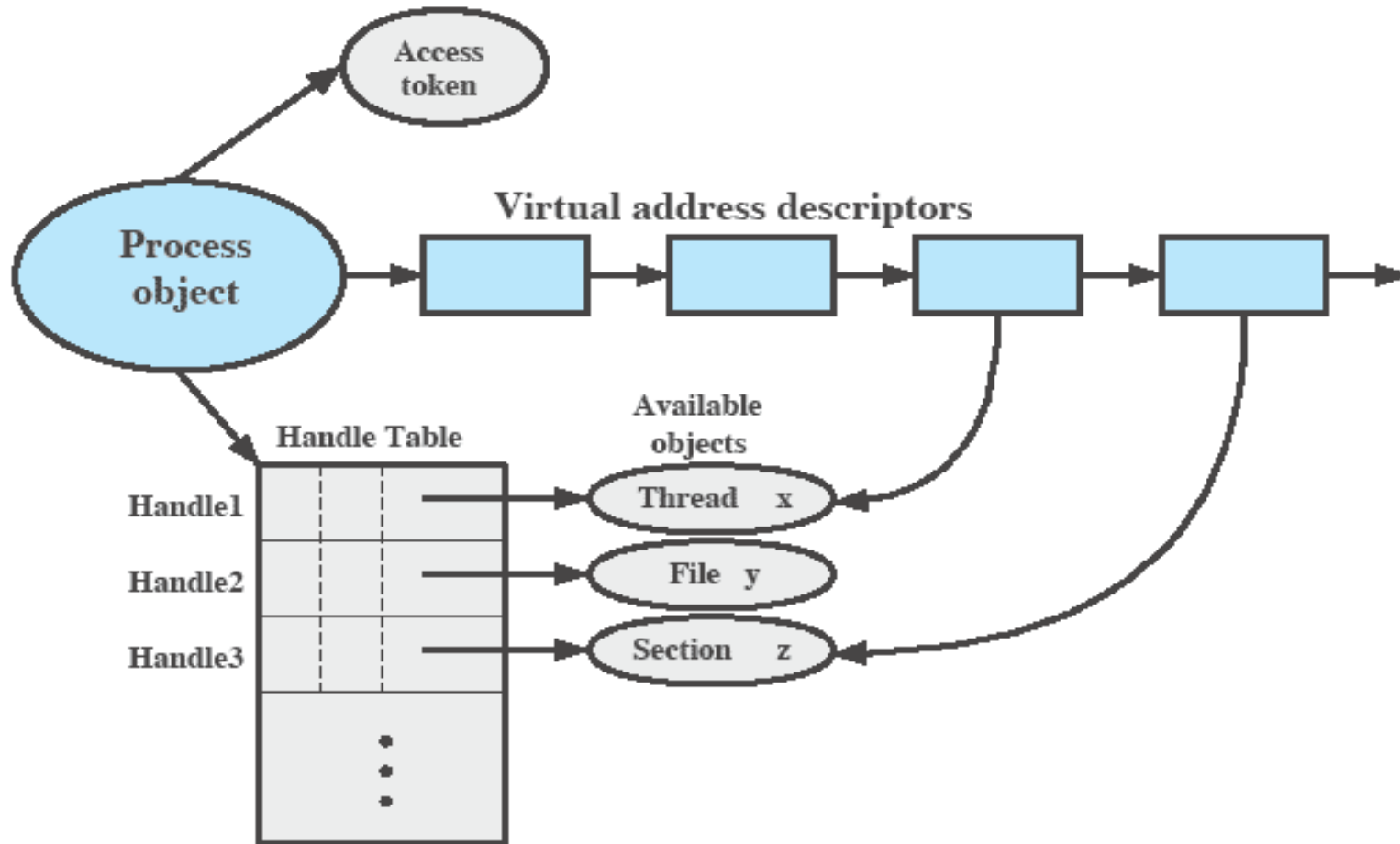
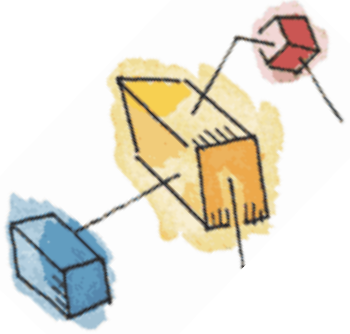
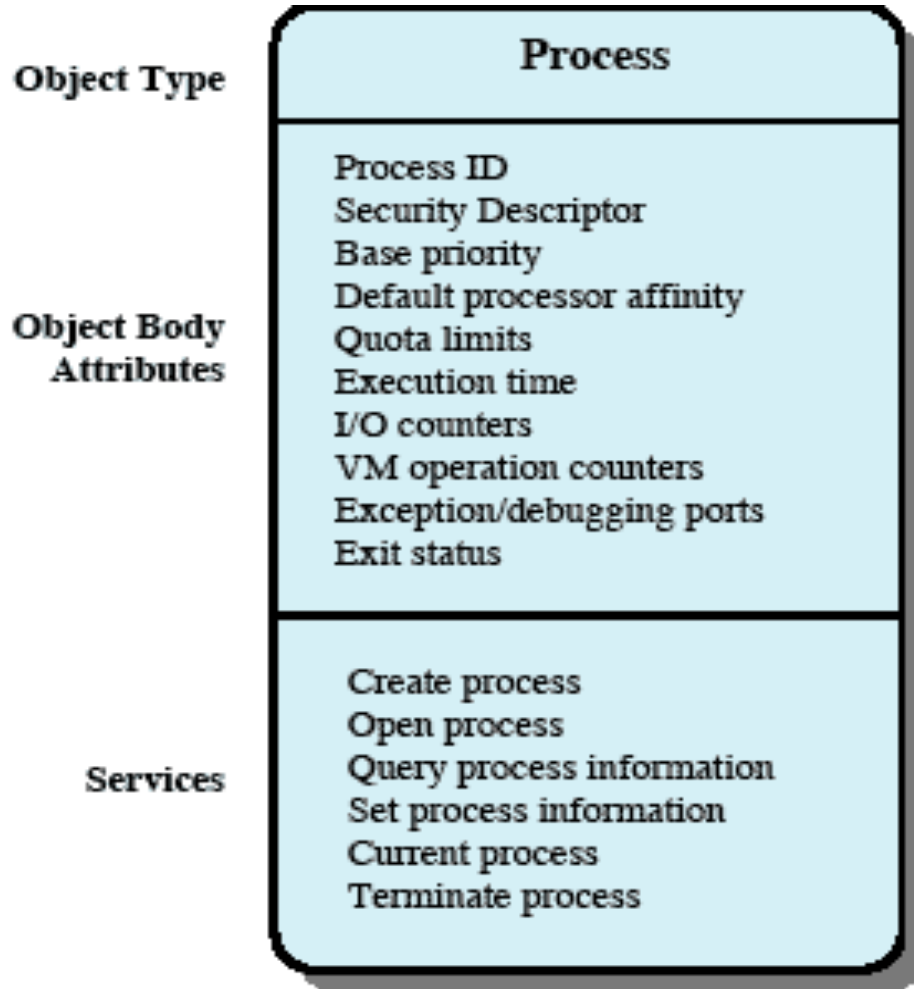


Figure 4.12 A Windows Process and Its Resources

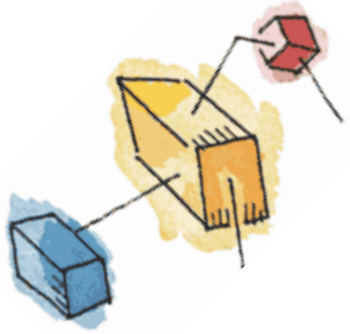


# Windows Process Object



(a) Process object





# Windows Thread Object

**Object Type**

**Thread**

**Object Body Attributes**

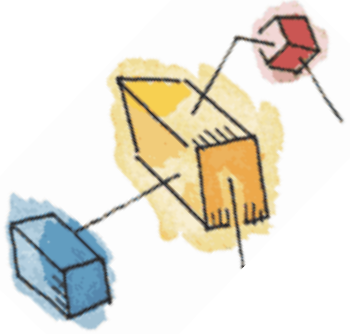
Thread ID  
Thread context  
Dynamic priority  
Base priority  
Thread processor affinity  
Thread execution time  
Alert status  
Suspension count  
Impersonation token  
Termination port  
Thread exit status

**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





# Thread States

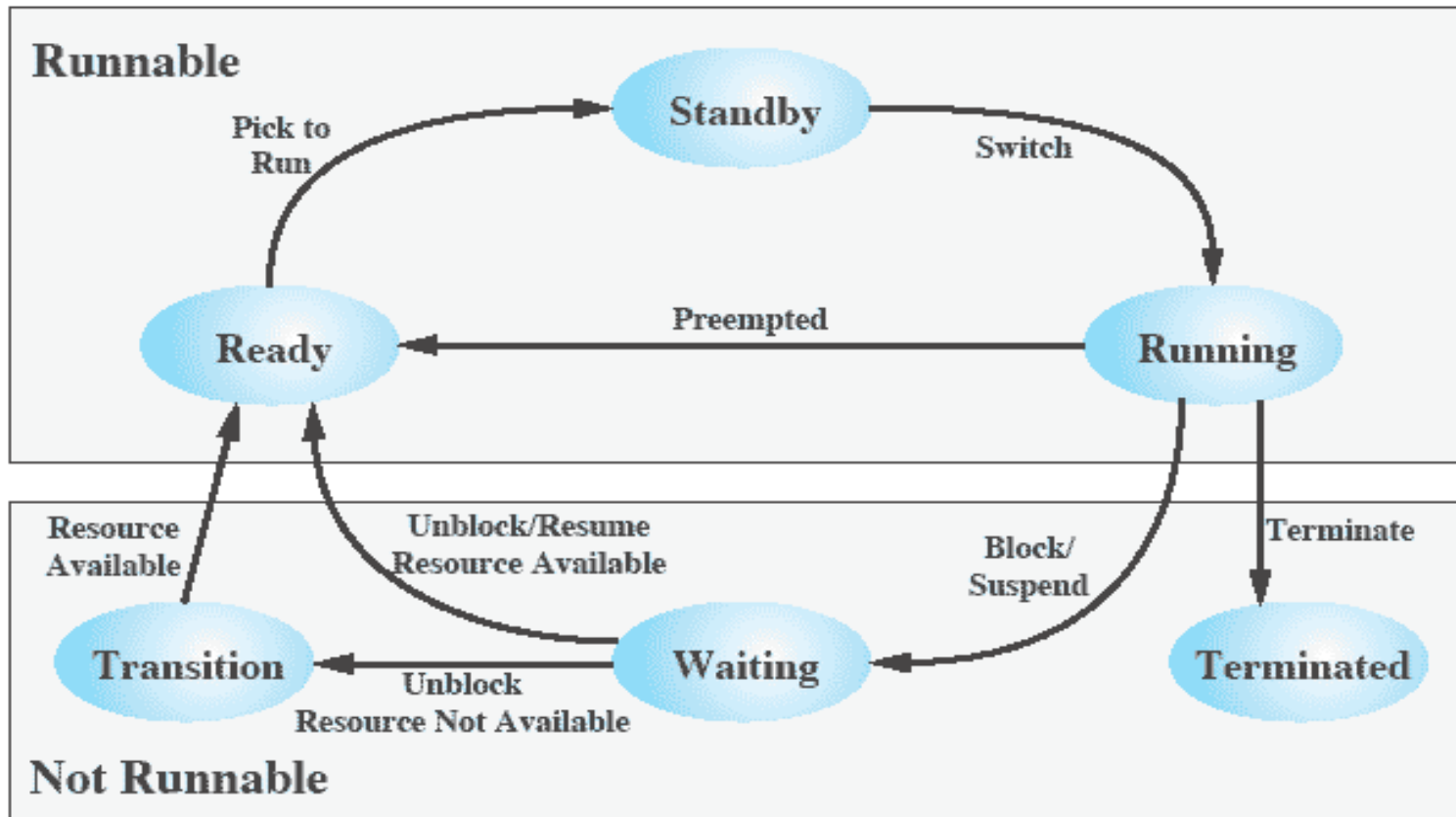
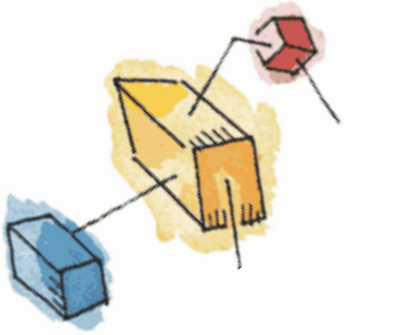


Figure 4.14 Windows Thread States

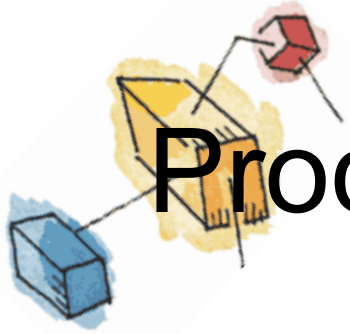


# Solaris



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





# Processes and Threads in Solaris

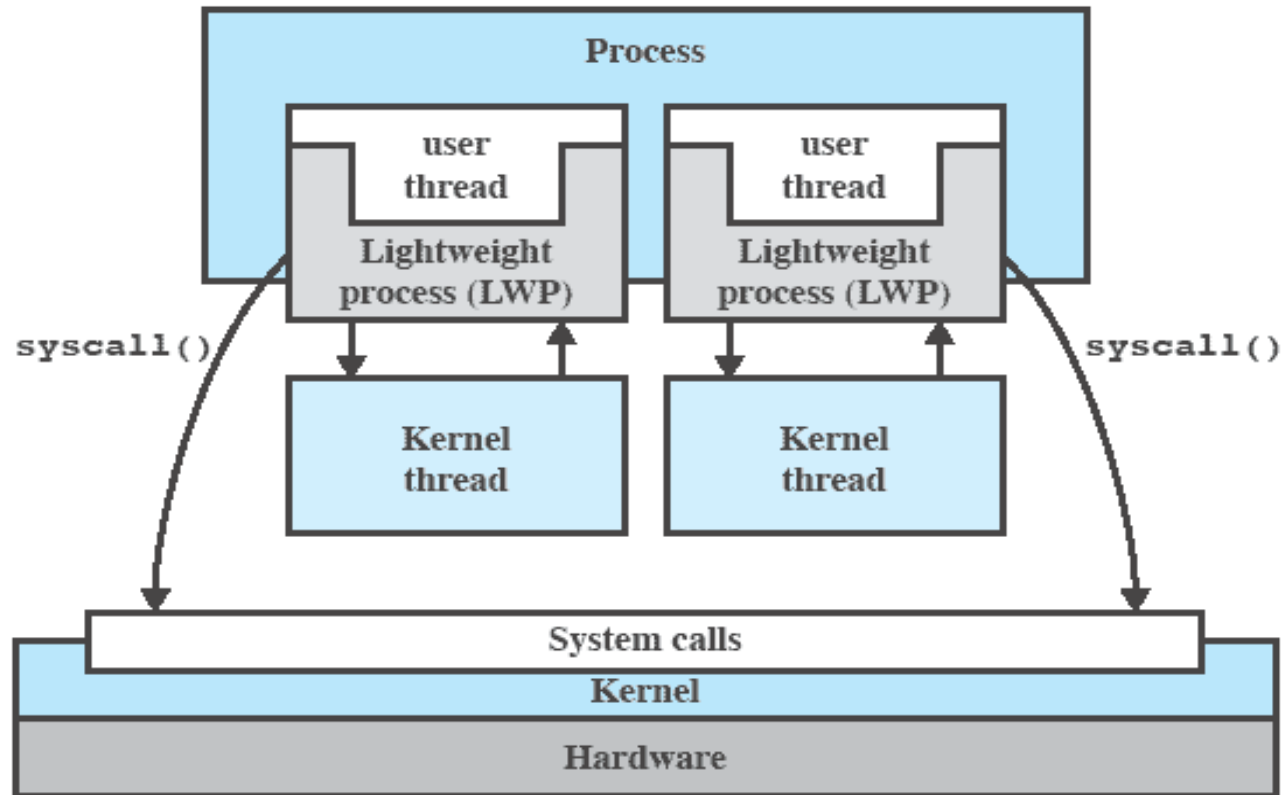
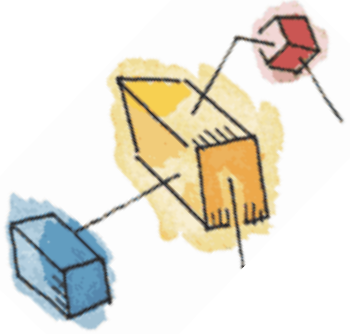


Figure 4.15 Processes and Threads in Solaris [MCDO07]

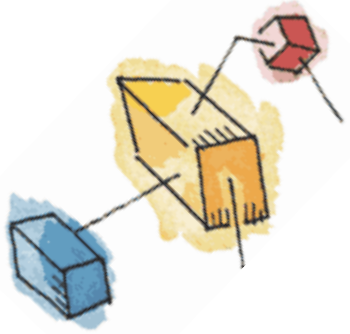




# LWP Data Structure

- Identifier
- Priority
- Signal mask
- Saved values of user-level registers

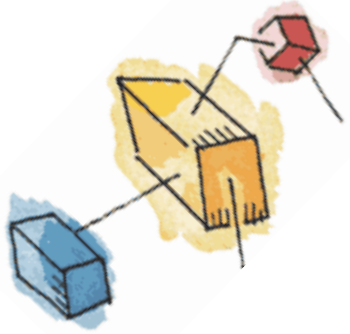




# LWP Data Structure

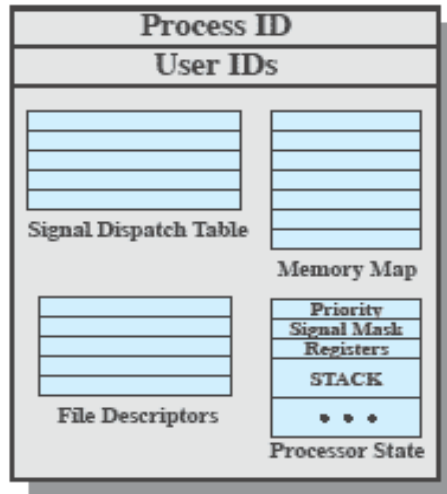
- Kernel stack
- Resource usage and profiling data
- Pointer to the corresponding kernel thread
- Pointer to the process structure





# Process Structure

## UNIX Process Structure



## Solaris Process Structure

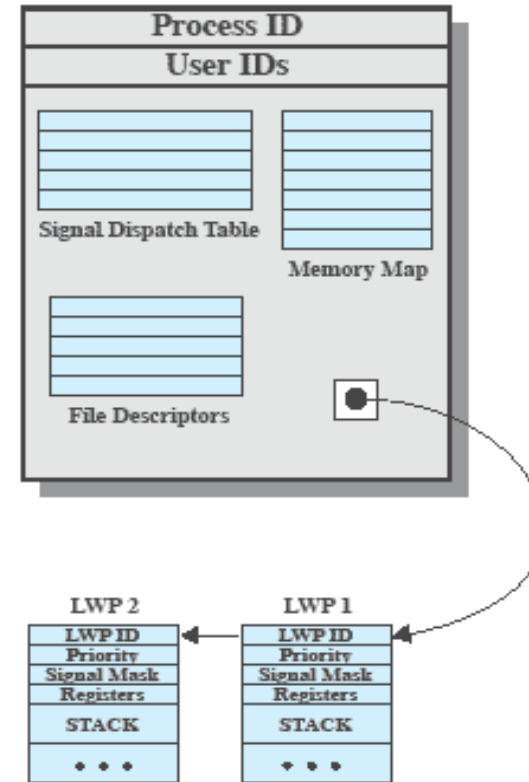
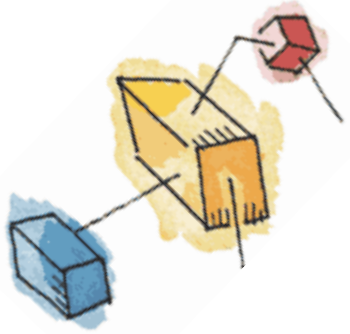


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]





# Solaris Thread States

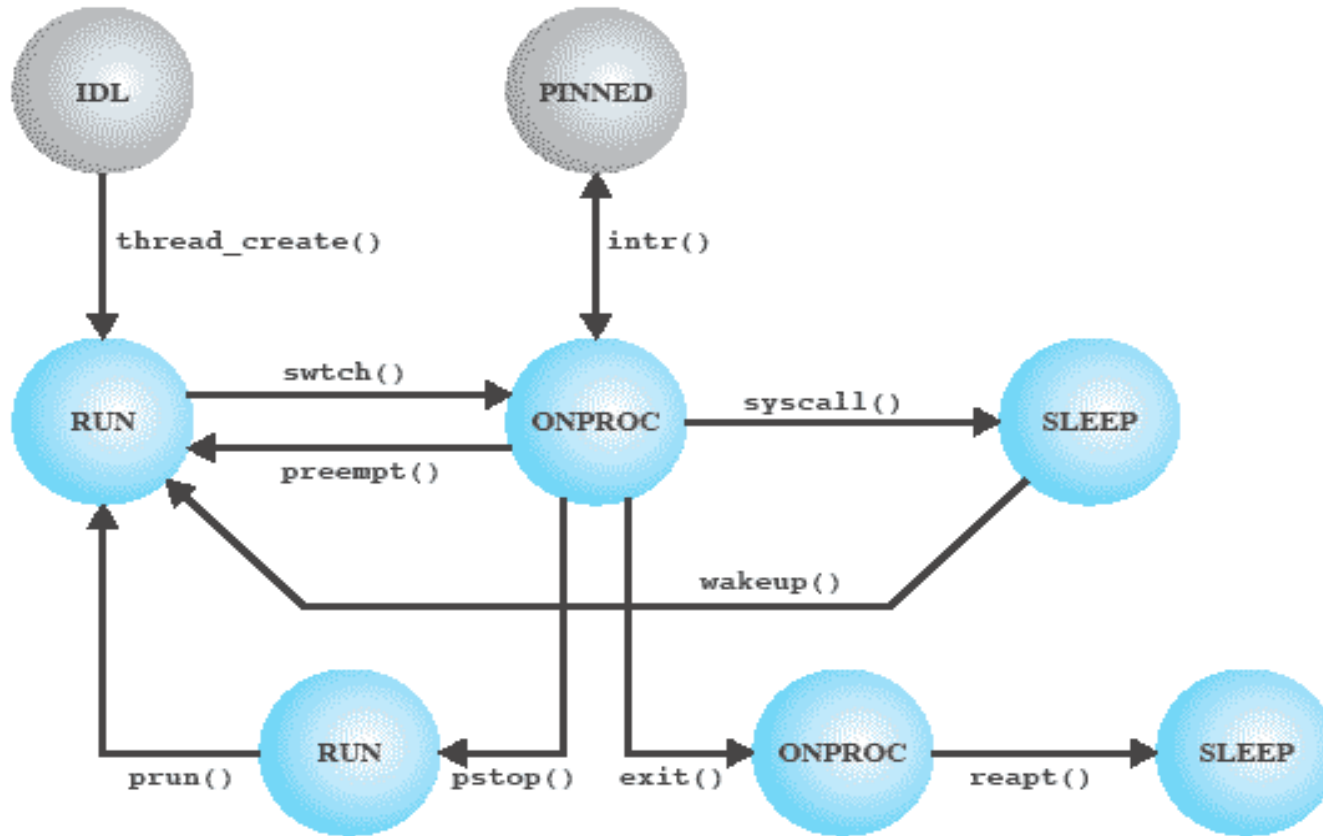
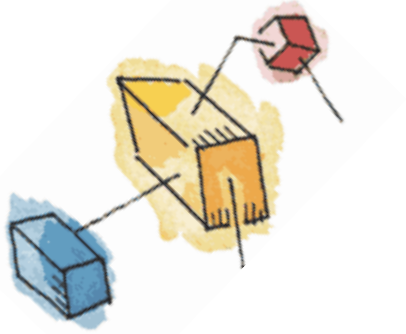


Figure 4.17 Solaris Thread States [MCDO07]



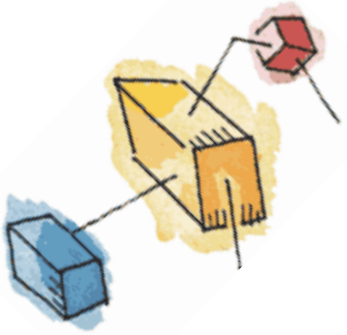
# Linux Tasks



- State
- Scheduling information
- Identifiers
- Interprocess communication

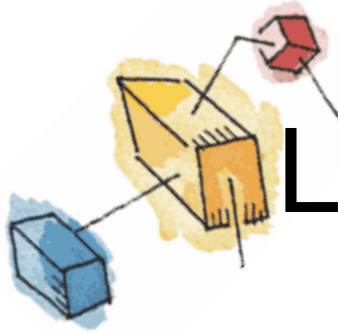


# Linux Tasks



- Links
- Times and timers
- File system
- Address space
- Processor-specific context





# Linux Process/Thread Model

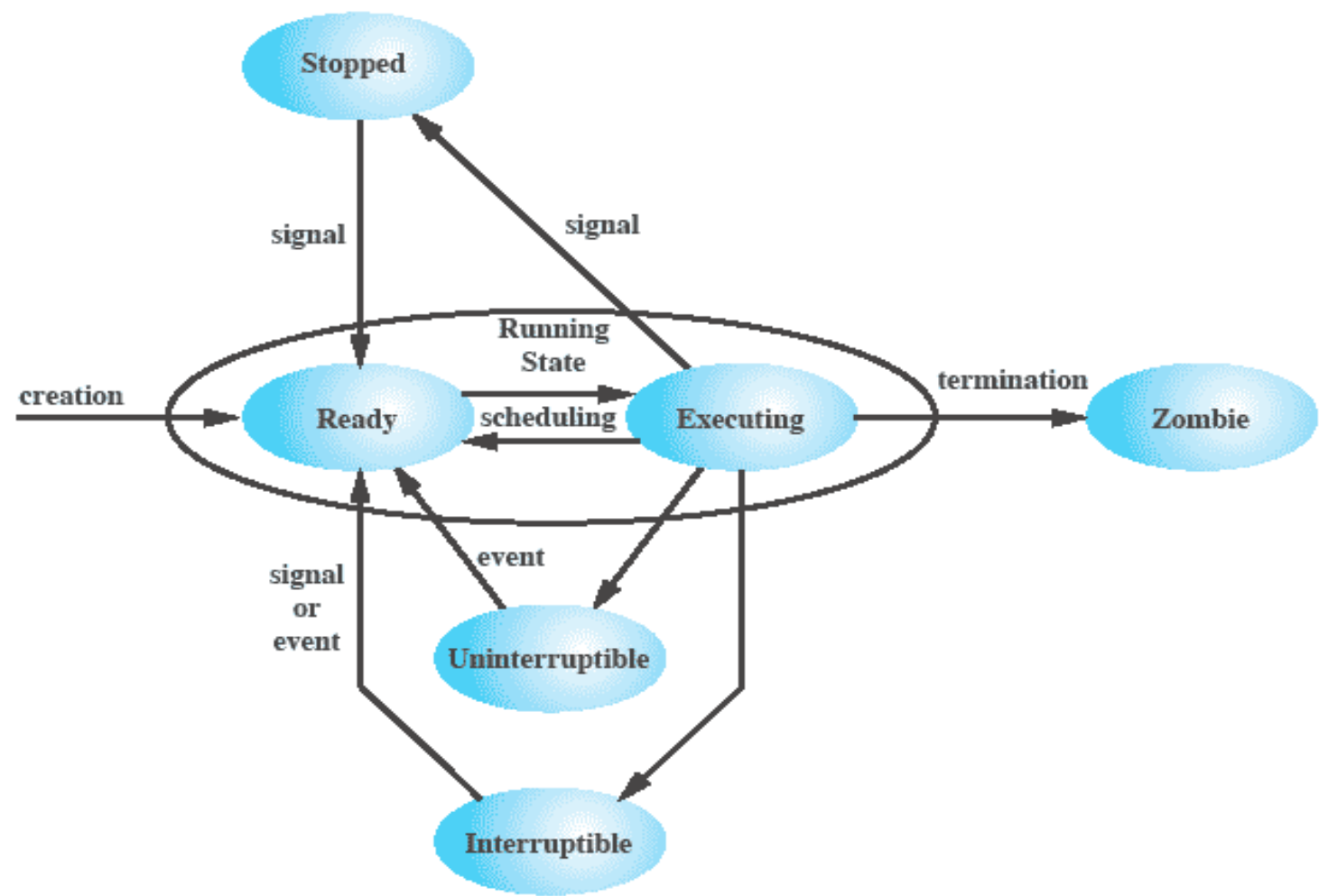


Figure 4.18 Linux Process/Thread Model

