Thread Concept Overview, Type of Threads, Multicore and Multi-threading, Multi-threading Models

WS 4.1-4.2 (pg.177- 189)
WS 4.3 (pg.190- 195)

Multithreading

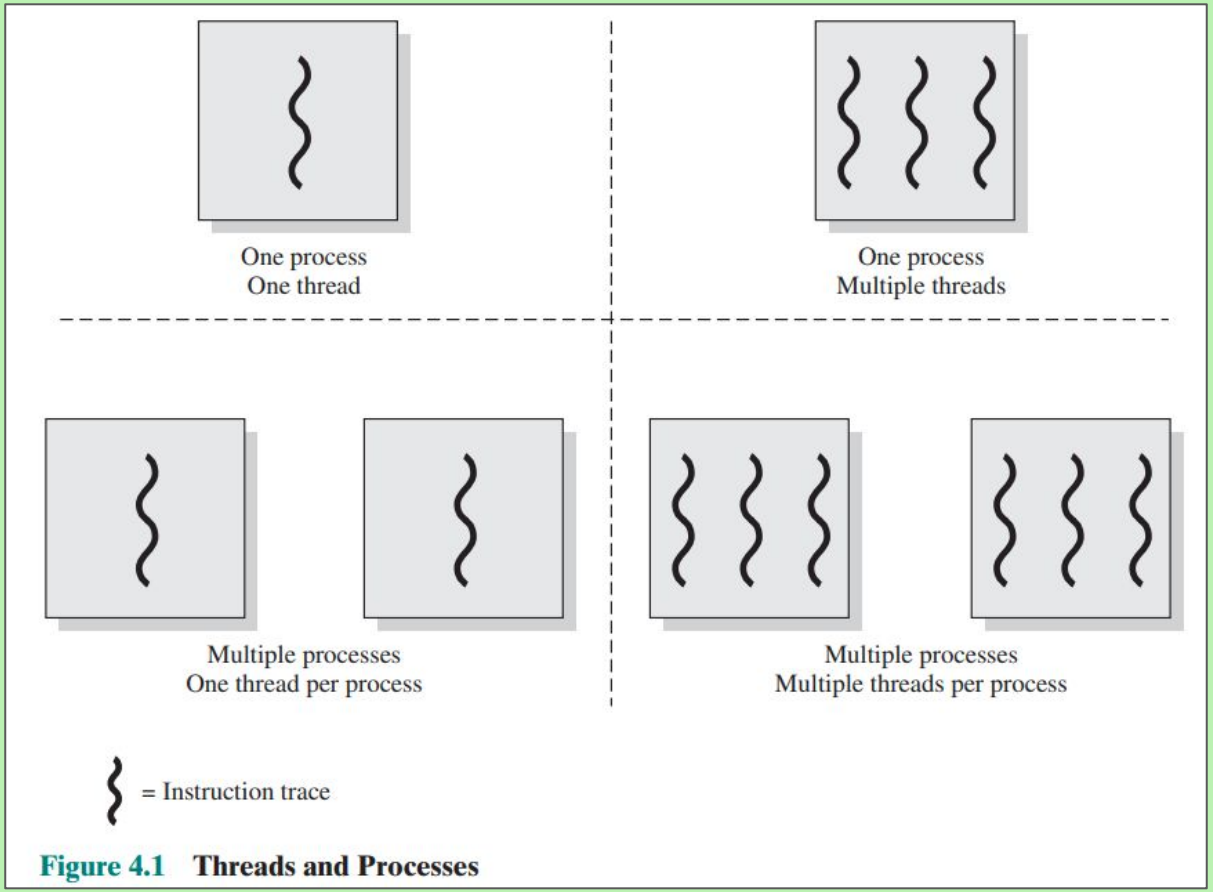


Figure 4.1 Threads and Processes

Multithreading

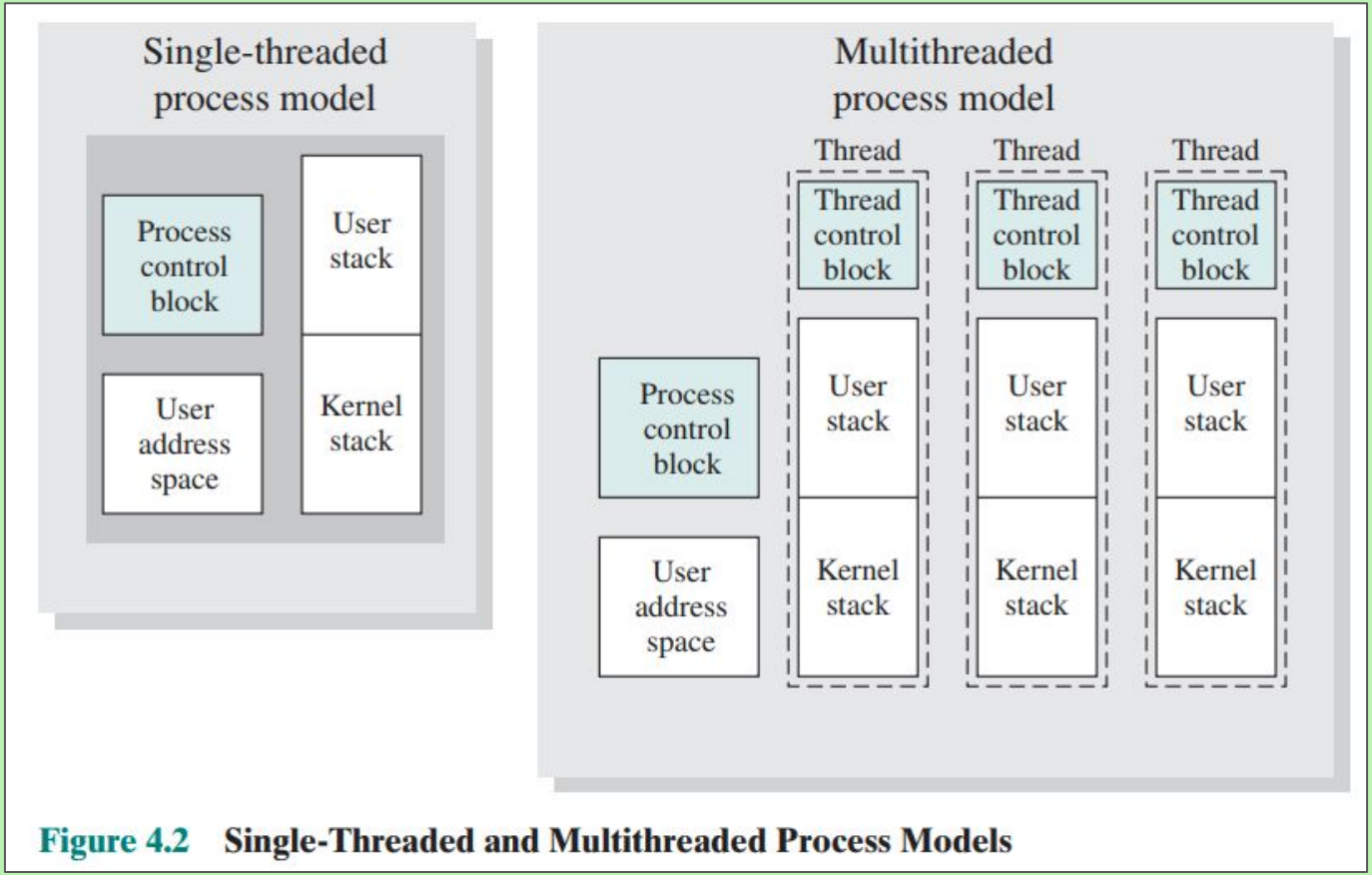


Figure 4.2 **Single-Threaded and Multithreaded Process Models**

Multithreading

The key benefits of threads derive from the performance implications:

- 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.
- It takes less time to terminate a thread than a process.
- It takes less time to switch between two threads within the same process than to switch between processes.
- Threads enhance efficiency in communication between different executing programs.

Multithreading

[LETW88] gives four examples of the uses of threads in a single-user multiprocessing system:

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

Thread Functionality

- Thread States: As with processes, the key states for a thread are Running, Ready, and Blocked.
- There are four basic thread operations associated with a change in thread state:
  - Spawn
  - Block
  - Unblock
  - Finish

Thread Functionality

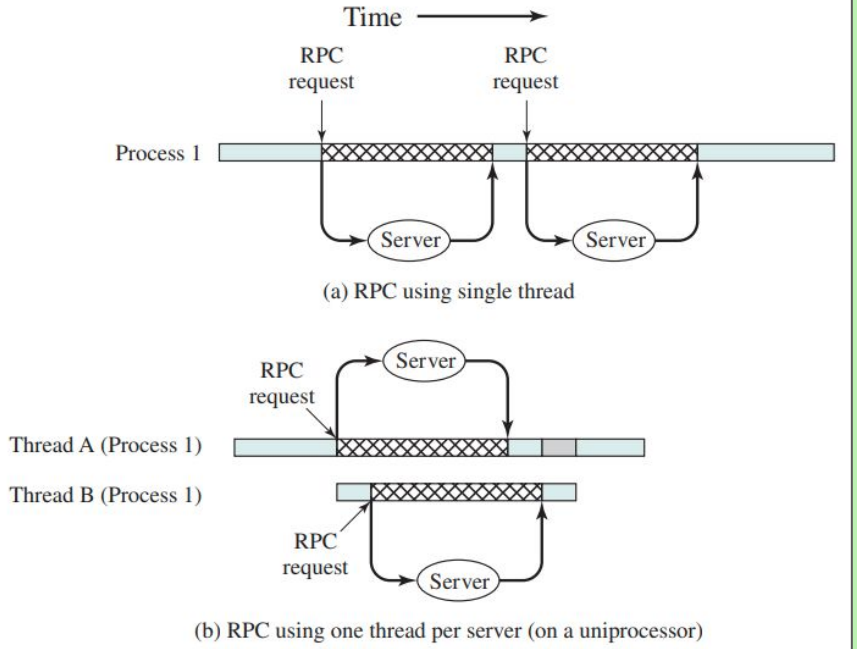


Figure 4.3 **Remote Procedure Call (RPC) Using Threads**

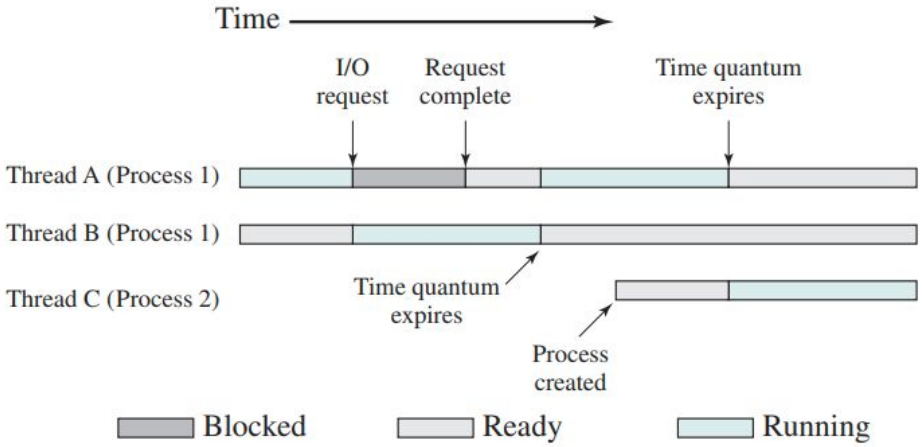


Figure 4.4 **Multithreading Example on a Uniprocessor**

TYPES OF THREADS: User-Level and Kernel-Level Threads

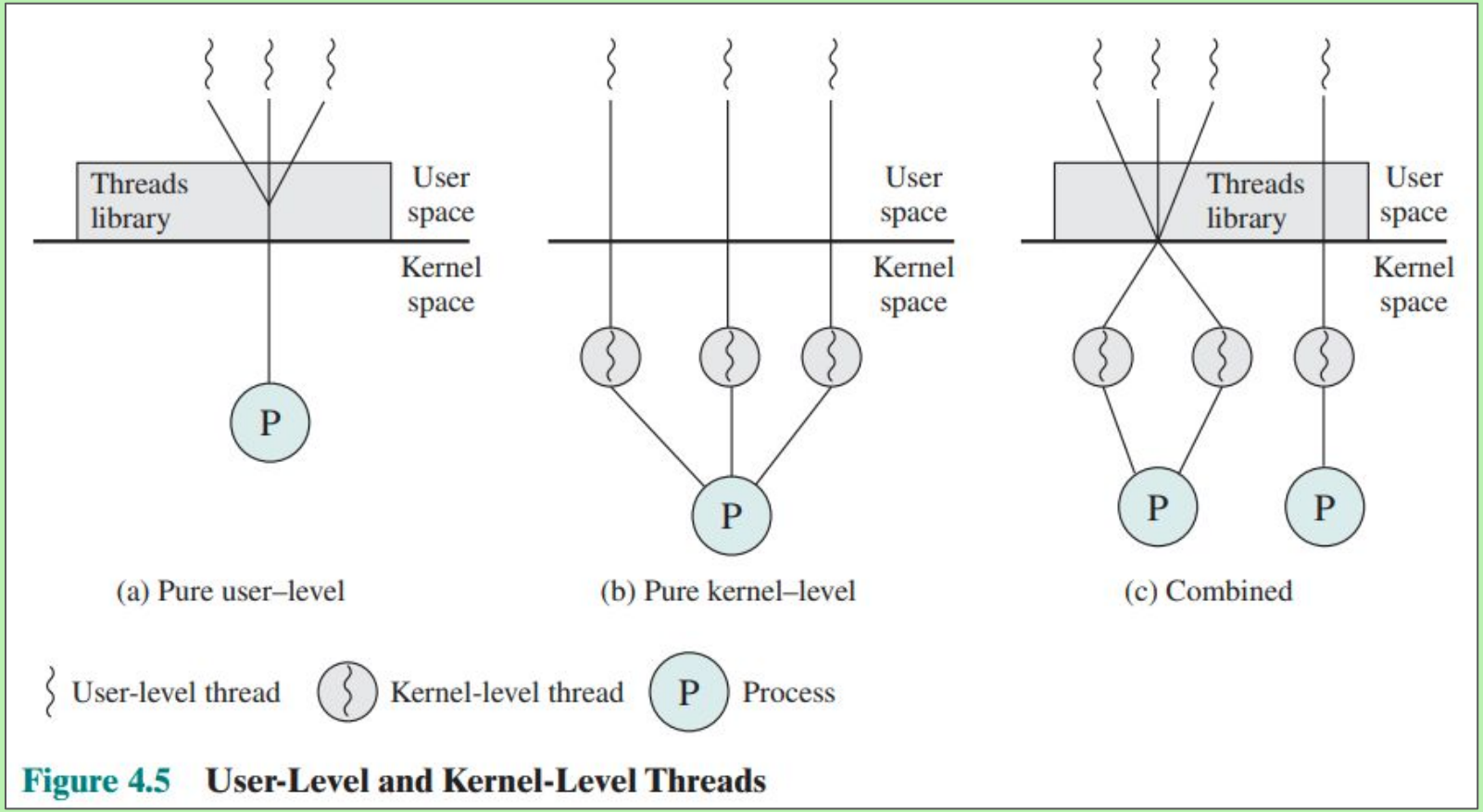


Figure 4.5 **User-Level and Kernel-Level Threads**

TYPES OF THREADS

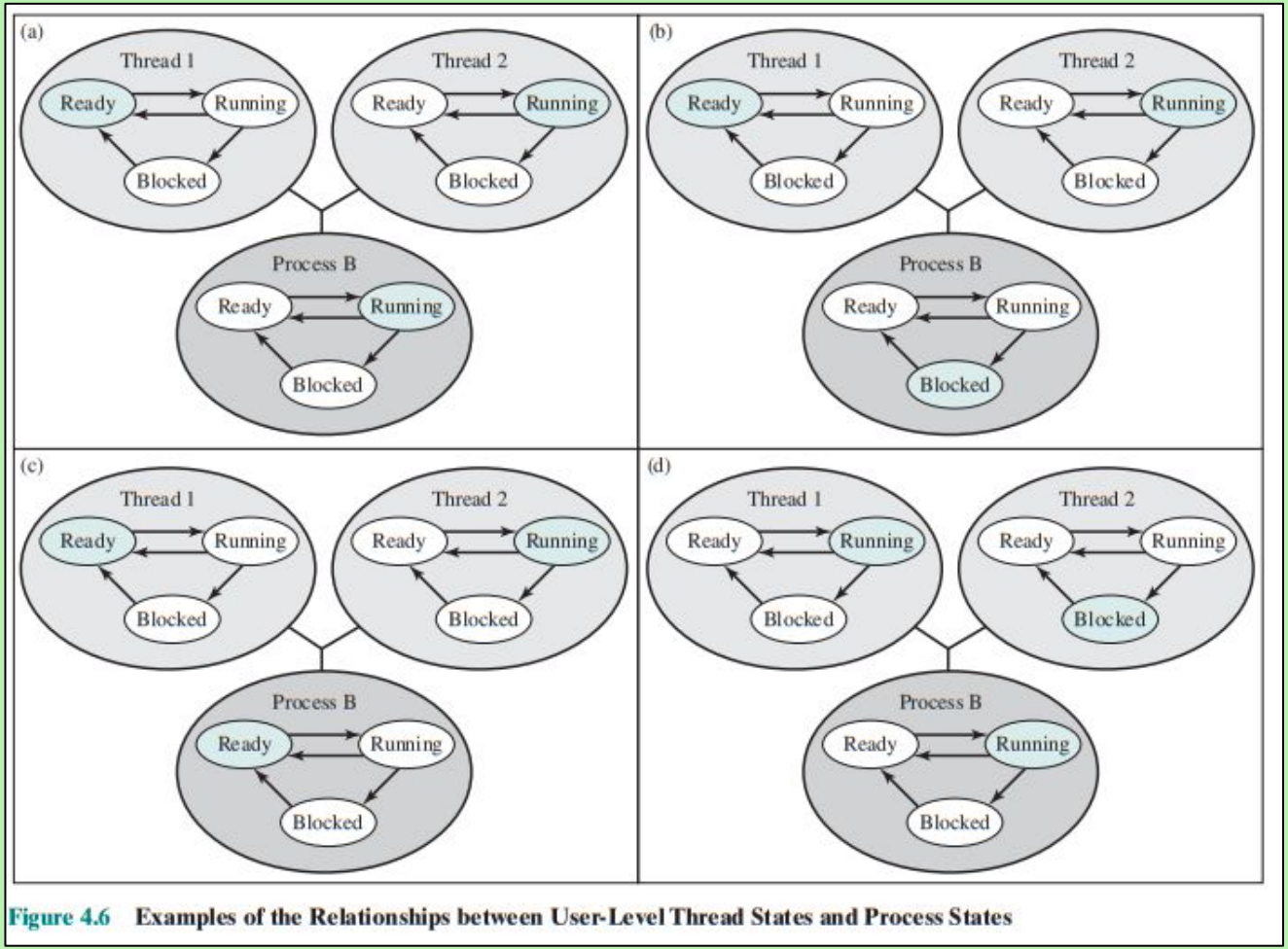


Figure 4.6 Examples of the Relationships between User-Level Thread States and Process States

Advantages to the use of ULTs instead of KLTs

- Thread switching does not require kernel-mode privileges because all of the thread management data structures are within the user address space of a single process.
- Scheduling can be application specific.
- ULTs can run on any OS. No changes are required to the underlying kernel to support ULTs. The threads library is a set of application-level functions shared by all applications.

Disadvantages to the use of ULTs instead of KLTs

- 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 as well.
- In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing. A kernel assigns one process to only one processor at a time. Therefore, only a single thread within a process can execute at a time.

Relationships between threads & 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	It combines attributes of M:1 and 1:M cases.	TRIX

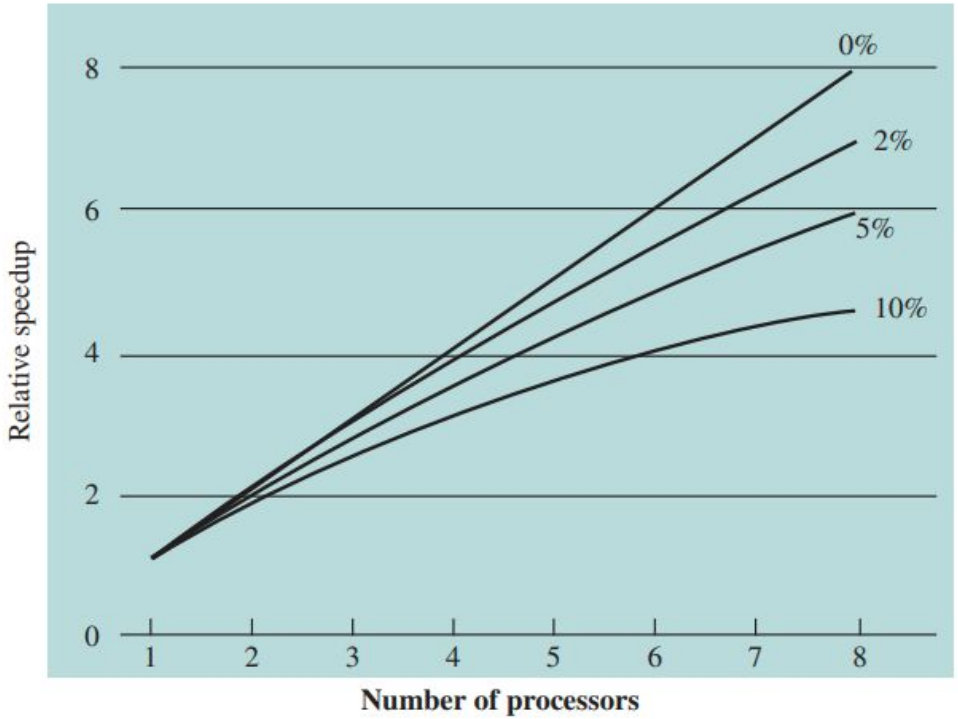
Performance of Software on Multicore

The potential performance benefits of a multicore organization depend on the ability to effectively exploit the parallel resources available to the application. Let us focus first on a single application running on a multicore system. Amdahl's law (see Appendix E) states that:

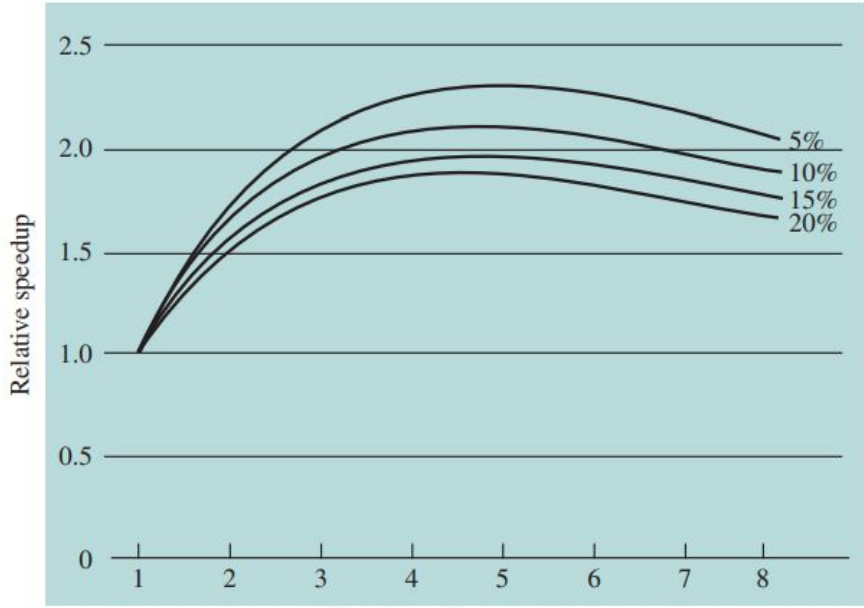
$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{1}{(1 - f) + \frac{f}{N}}$$

The law assumes a program in which a fraction $(1 - f)$ of the execution time involves code that is inherently serial, and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead.

Performance of Software on Multicore



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads

Figure 4.7 **Performance Effect of Multiple Cores**