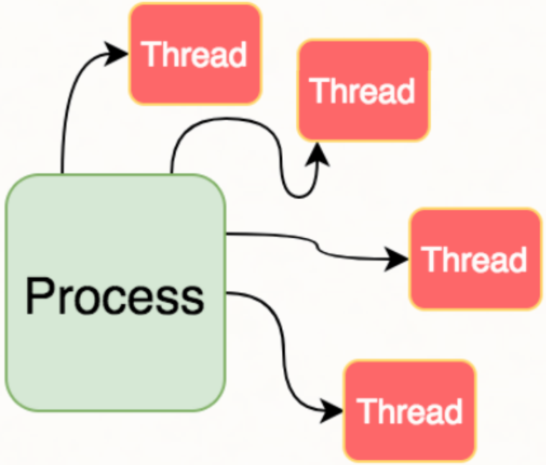
Chapter 4

Threads



The Basic Unit of CPU Utilization

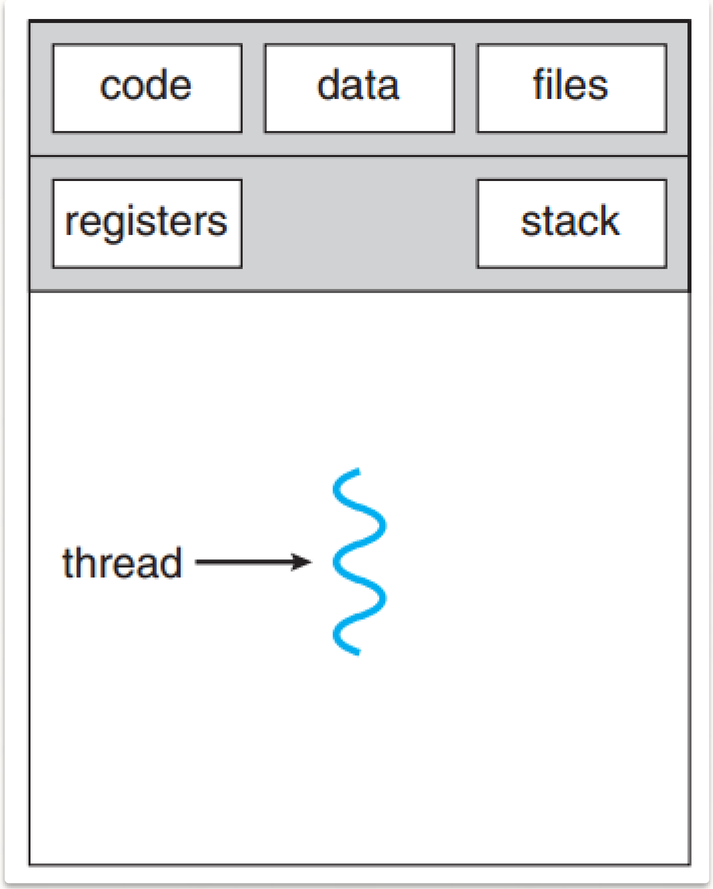
Chapter Objectives

- To introduce the notion of a thread—a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To discuss the APIs for the Pthreads, Win32, and Java thread libraries
- To examine issues related to multithreaded programming

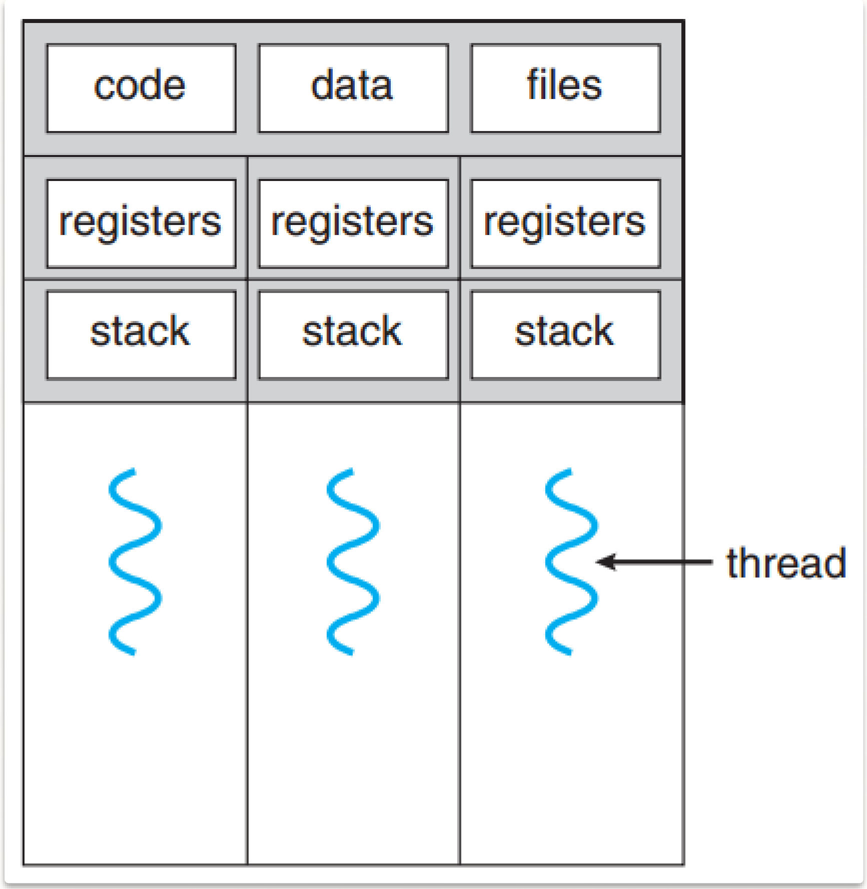
Introduction

- A thread is a basic unit of CPU utilization
- A thread consists of :
  1. a thread ID,
  2. a program counter,
  3. a register set, and
  4. a stack.
- When the same process creates multiple threads, they share common:
  - code section,
  - data section, and other operating-system resources (e.g. openfiles)
  - A traditional (or **heavyweight**) process has a single thread of control.
  - A **multi-threaded** process can perform more than one task at a time.

Single-threaded vrs Multi-threaded Process

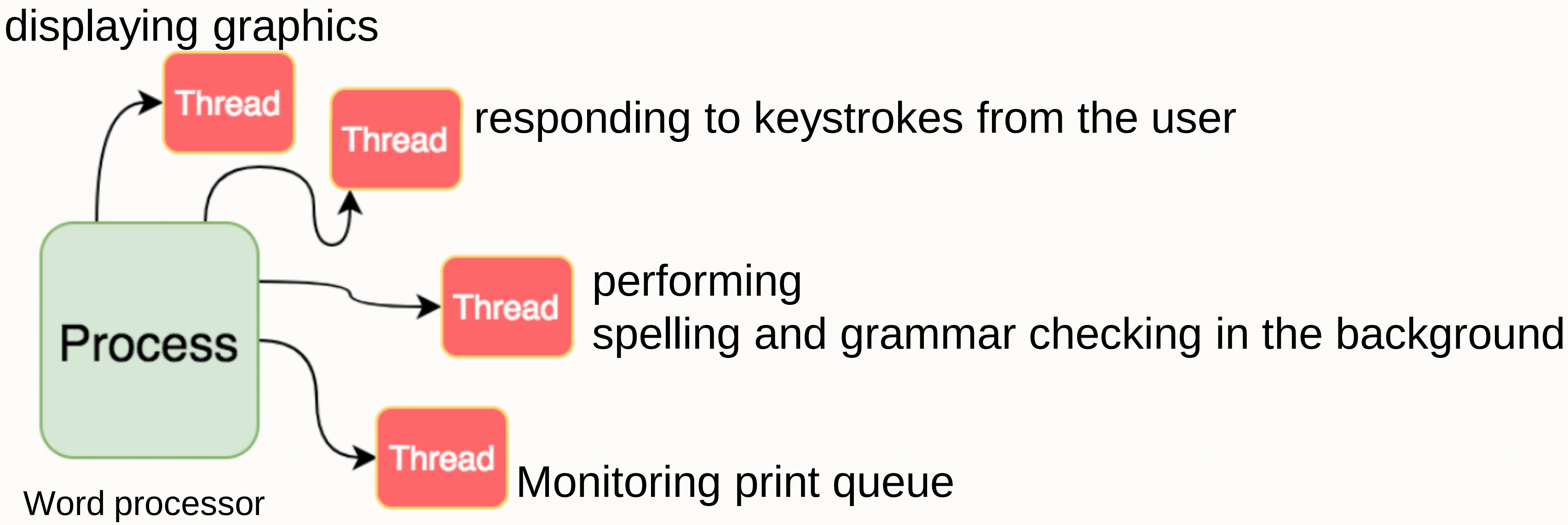


Single threaded process



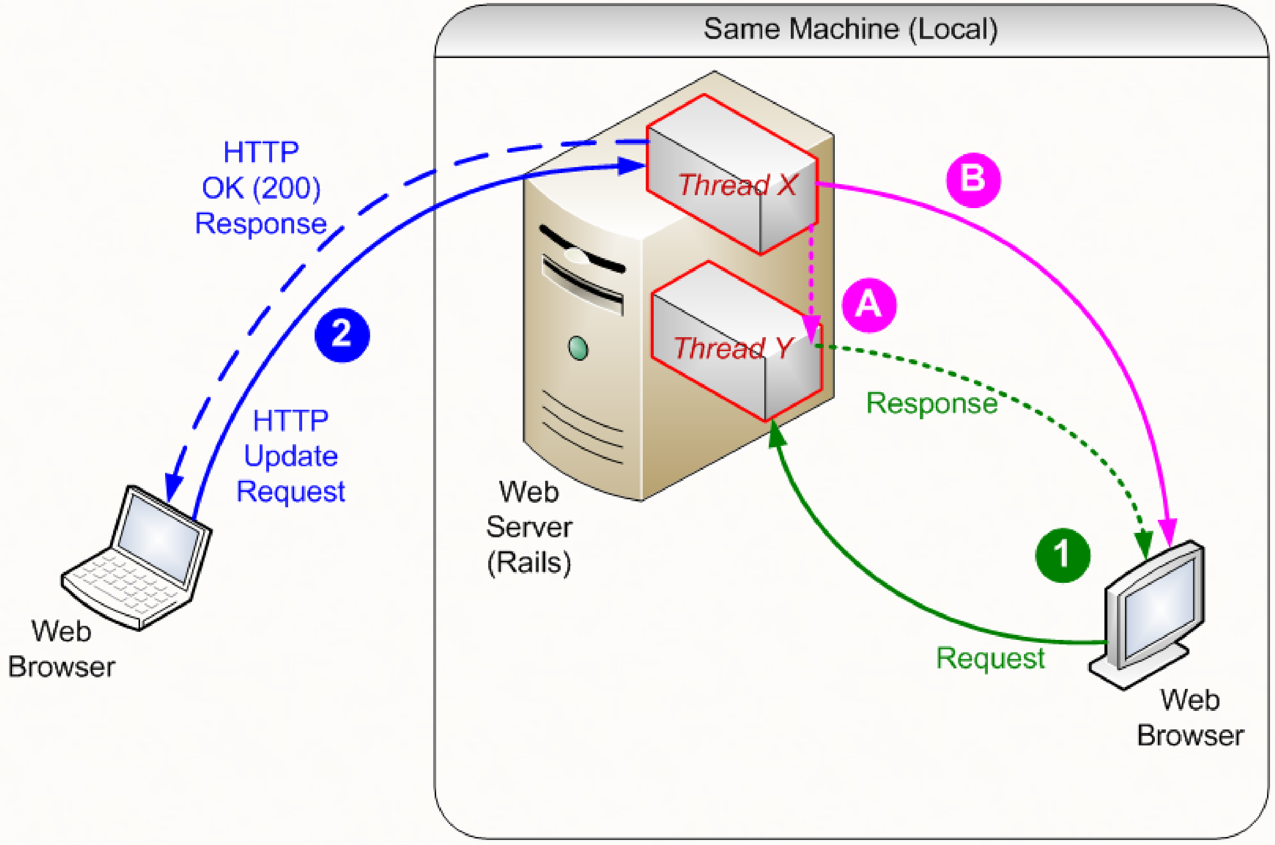
Multi-threaded process

- An application typically is implemented as a separate process with several threads of control.



Why threads (and not multi-process)

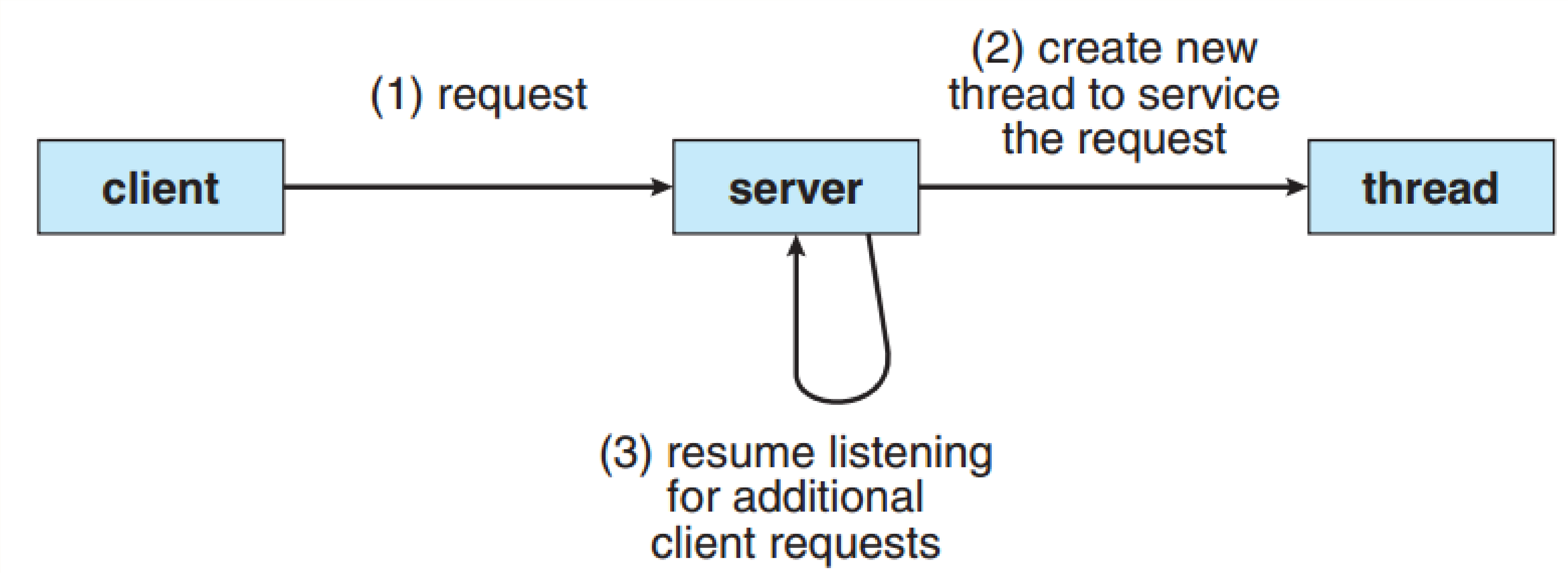
Consider a Web Server



Million concurrent client requests

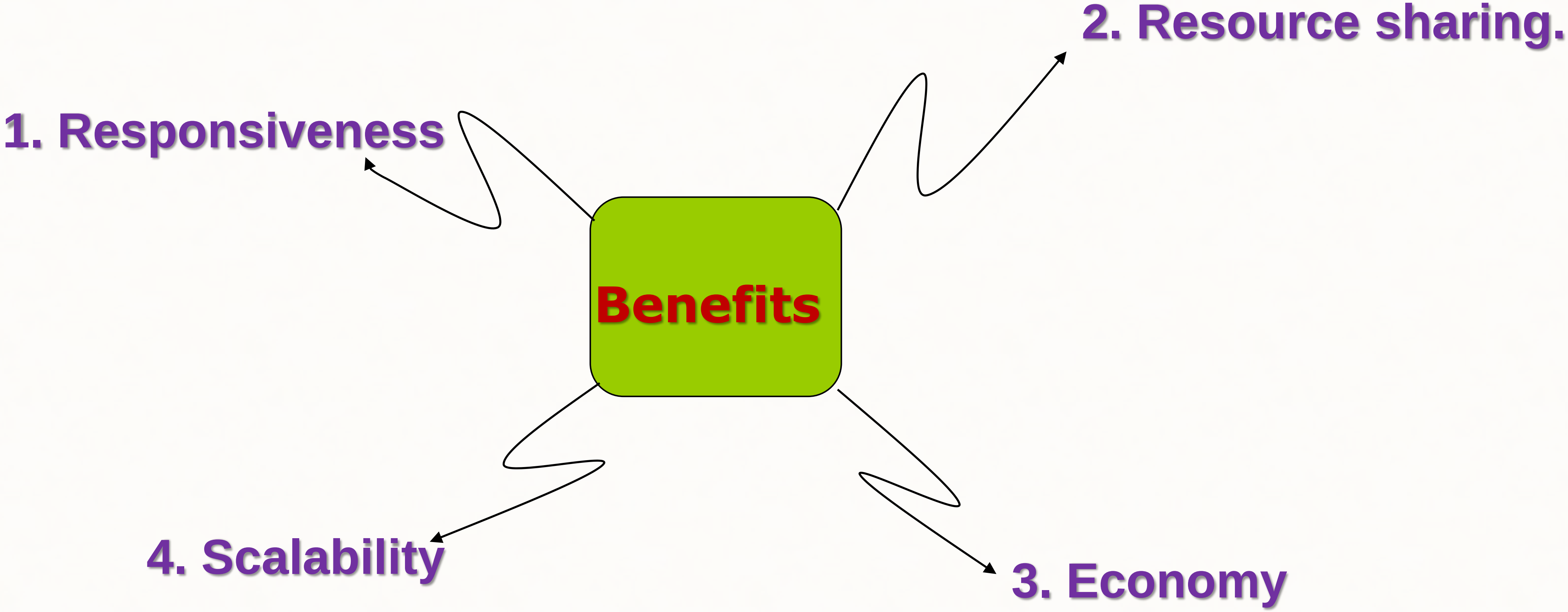
Multiple threads or multi-processes

Overhead?


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

- t is generally more efficient to use one process that contains multiple threads

Benefits of Threads



Benefits

1. Responsiveness

2. Resource sharing.

3. Economy

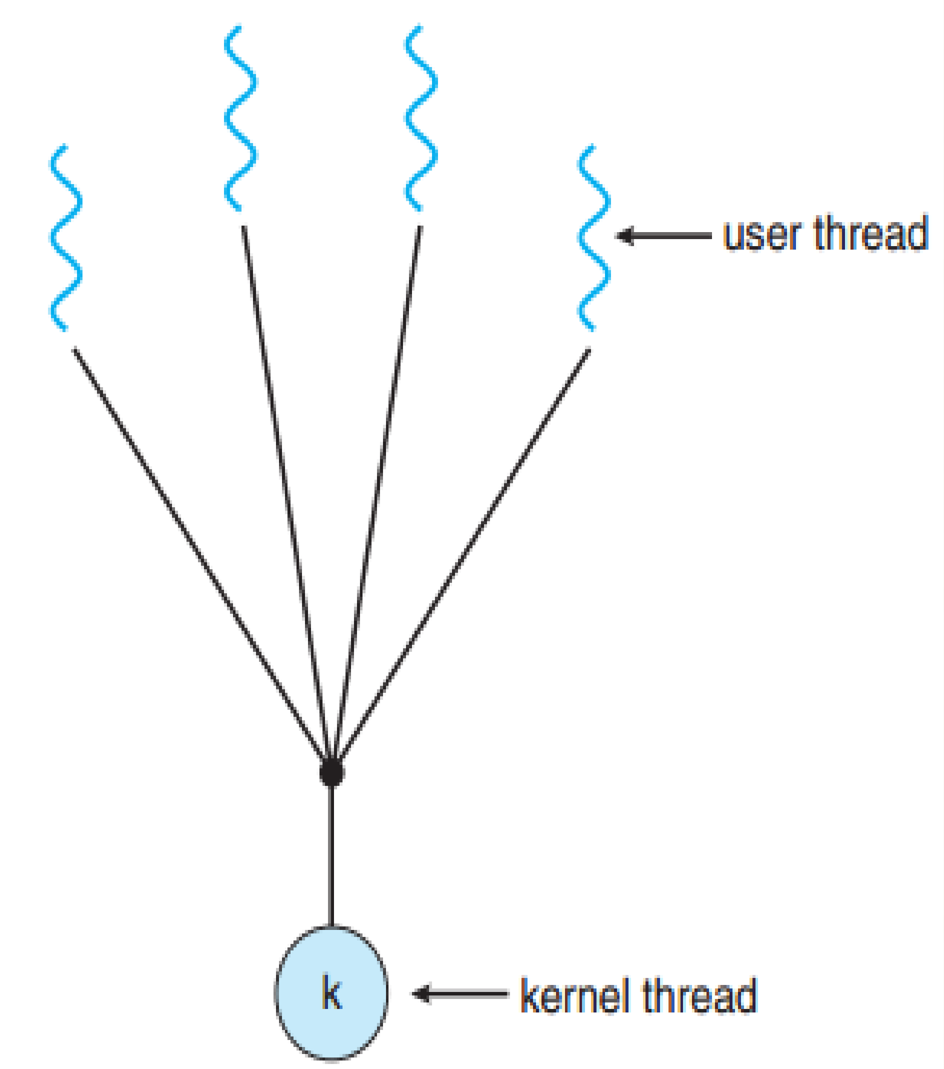
4. Scalability

Multithreading Models

- Threads may be created either at the user level, for **user threads**, or by the kernel, for **kernel threads**.
- User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system
- Ultimately, a relationship must exist between user threads and kernel threads:
  - Many-to-one model
  - One-to-one model
  - Many-to-many model

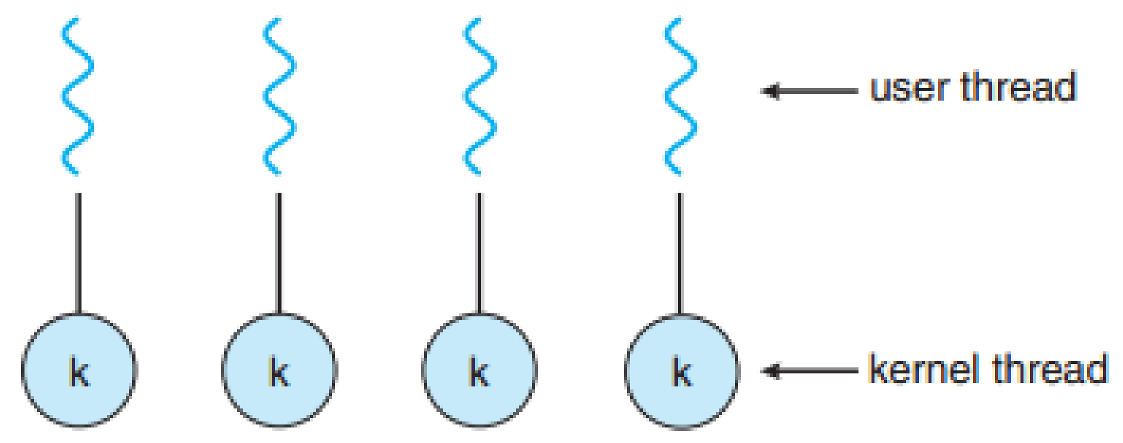
Many-to-one Model

- This model maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient; but the entire process will block if a thread makes a blocking system call
- only one thread can access the kernel at a
  time, therefore multiple threads are unable to run in parallel on multiprocessors



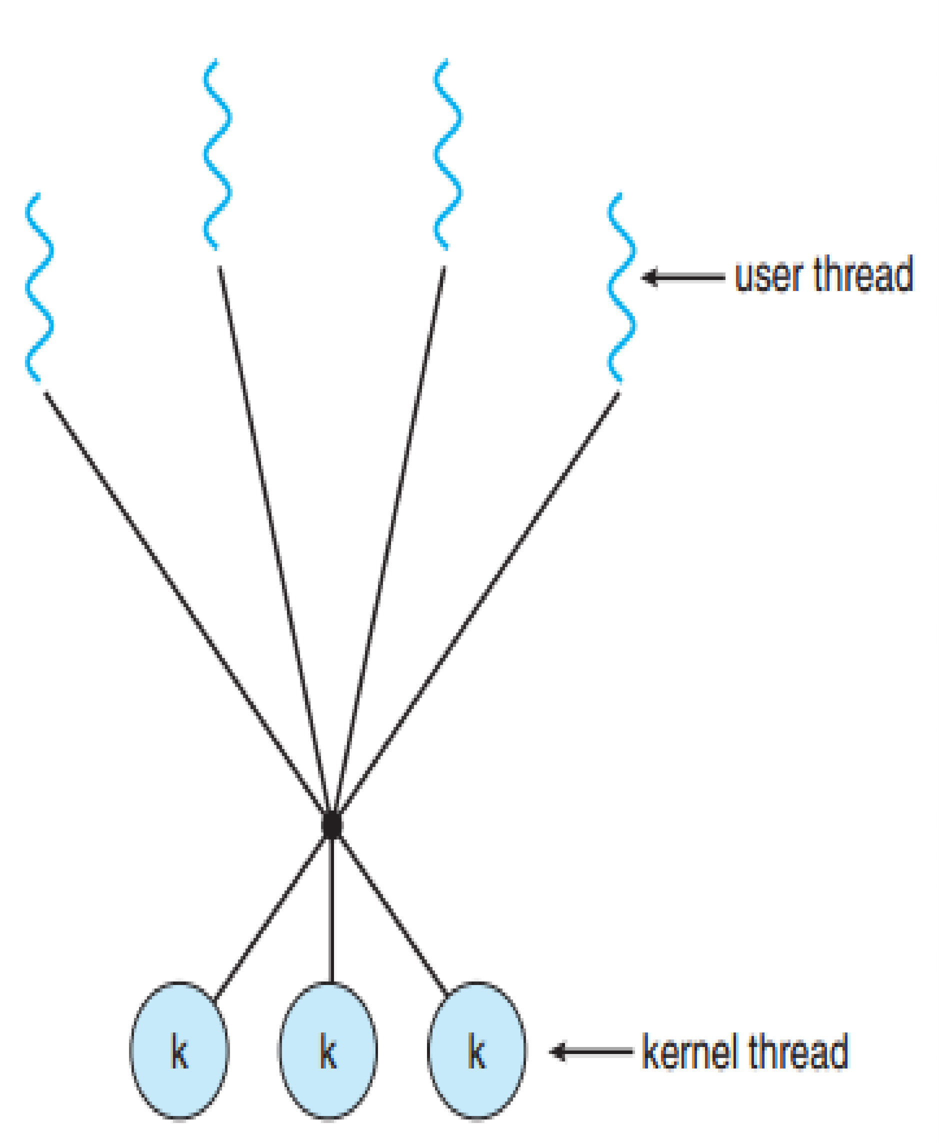
One-to-one Model

- This model maps each user thread to a kernel thread.
- It provides more concurrency than the many-to-one model by allowing another
  thread to run when a thread makes a blocking system call;
- it also allows multiple threads to run in parallel on multiprocessors.
- Drawback: creating a user thread requires creating the corresponding kernel thread.



Many-to-many Model

- This model multiplexes many user-level threads to a smaller or equal number of kernel threads.
- developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
- when a thread performs a blocking system call, the kernel can schedule another thread for execution


user thread
k
k
k
kernel thread