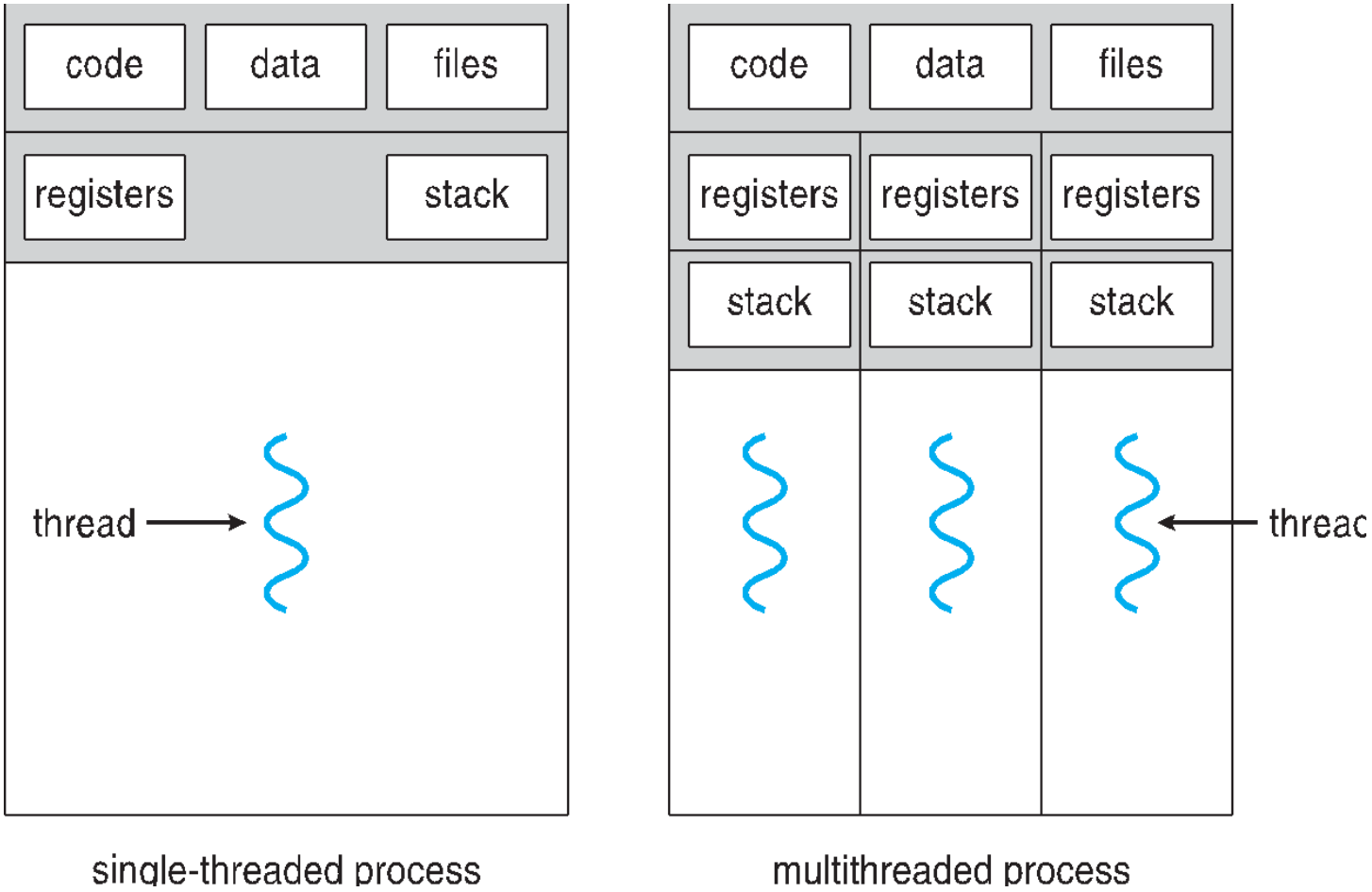
# Chapter 3: Threads

# Chapter 3: Threads

- Overview
- Single and Multithreaded Processes
- Multicore Programming
- Multithreaded Server Architecture
- User Threads and Kernel Threads
- Multithreading Models
- Thread Libraries

# Single and Multithreaded Processes



# Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

# Benefits

**Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces

**Resource Sharing –** threads share resources of process, easier than shared memory or message passing

**Economy –** cheaper than process creation, thread switching lower overhead than context switching

**Scalability –** process can take advantage of multiprocessor architectures

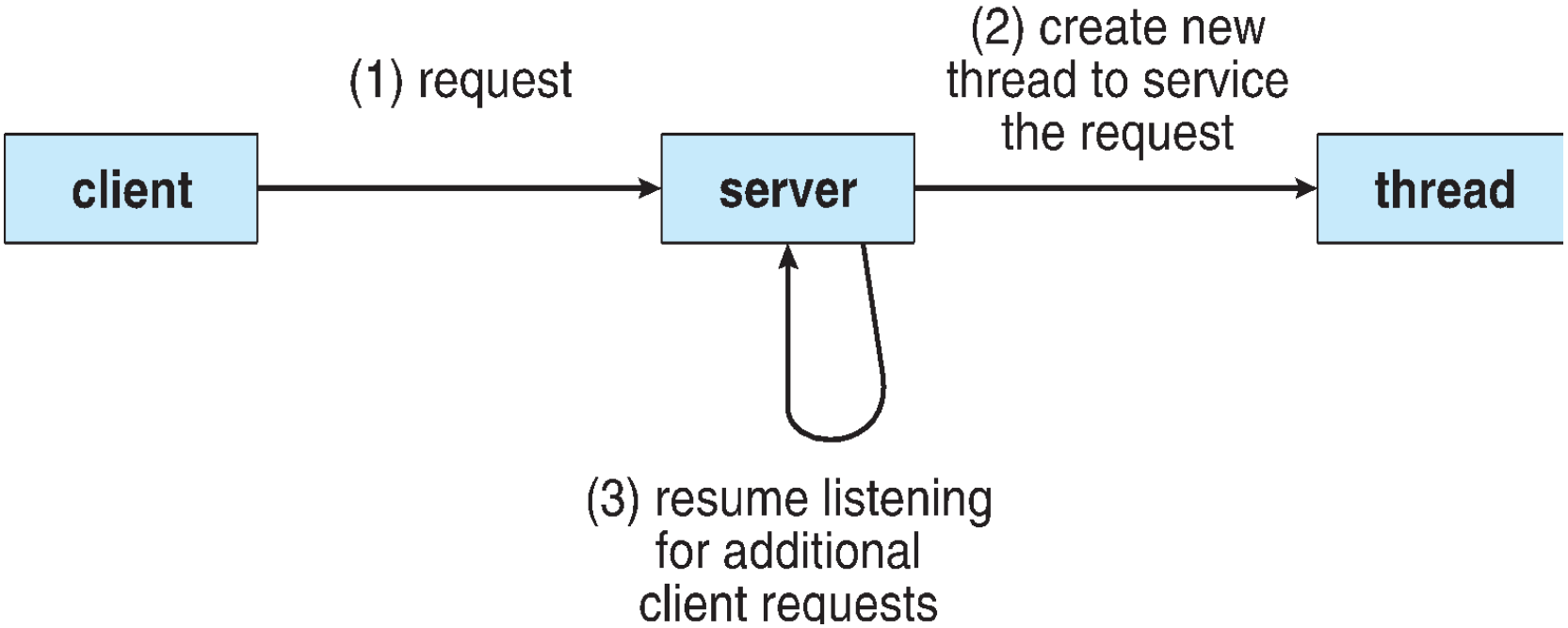
# Multicore Programming

- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- *Parallelism* implies a system can perform more than one task simultaneously
- *Concurrency* supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

# Multicore Programming (Cont.)

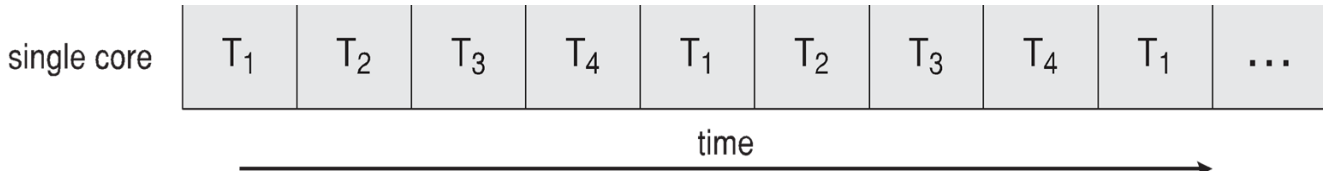
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

# Multithreaded Server Architecture

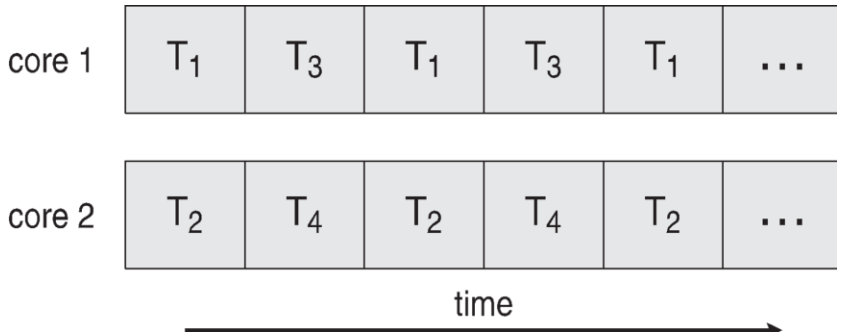


# Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**




- **Parallelism on a multi-core system:**



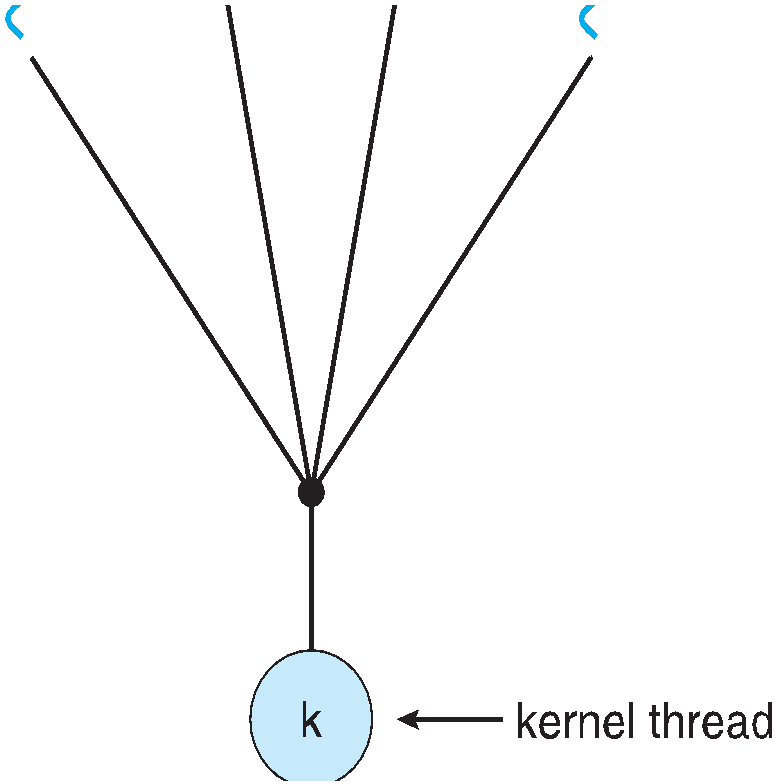
# User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

# Multithreading Models

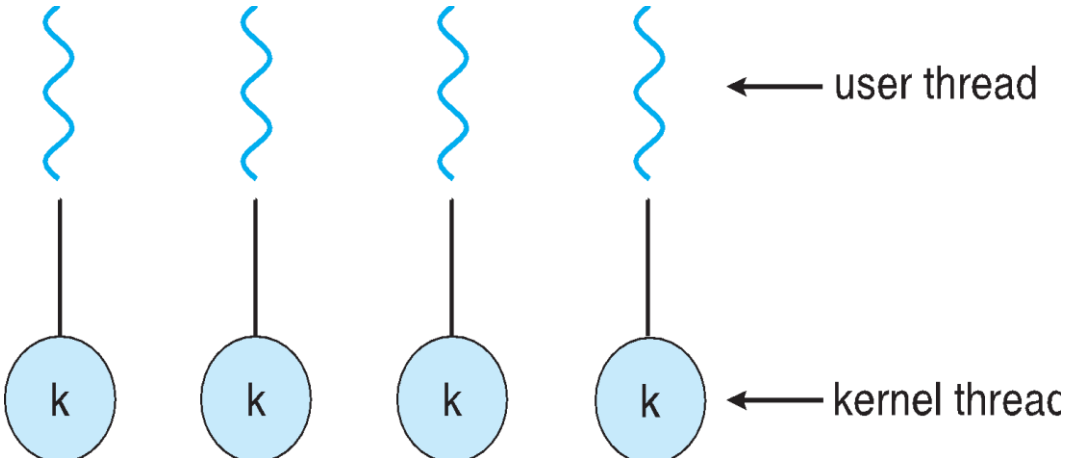
- Many-to-One
- One-to-One
- Many-to-Many

# Many-to-One



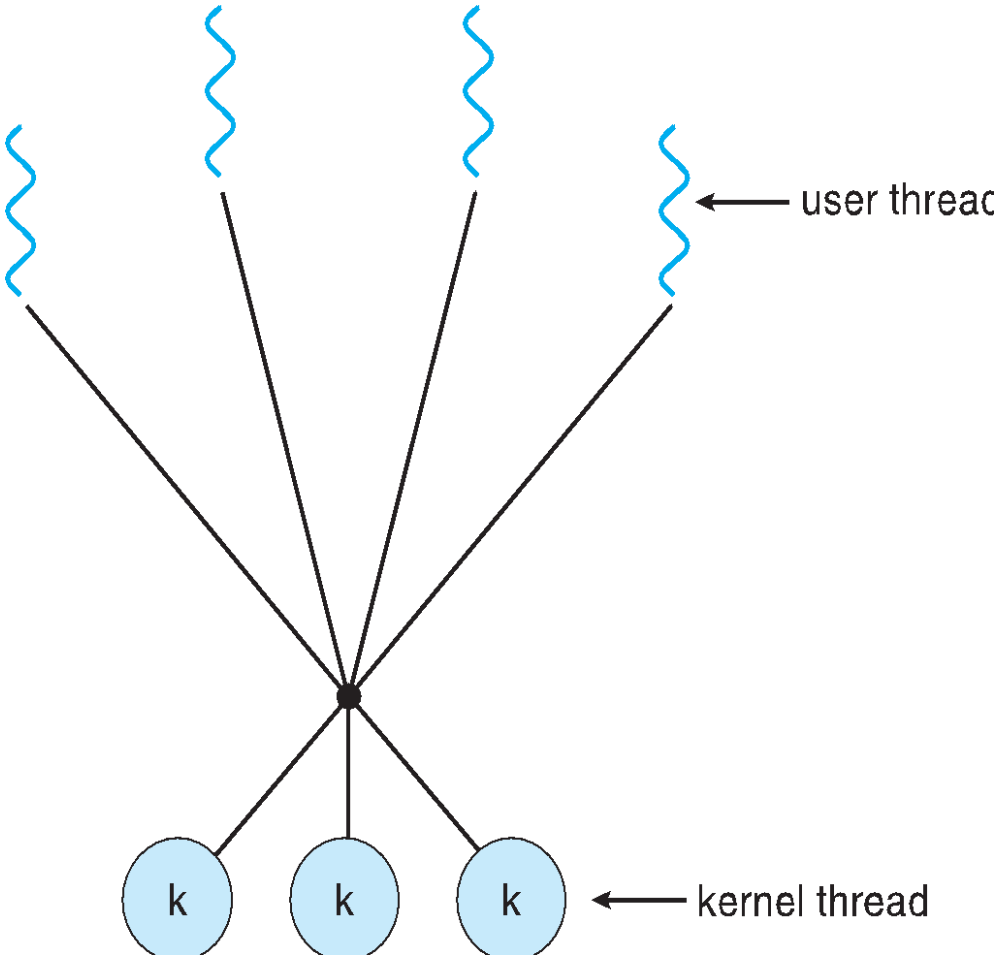

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

# One-to-One



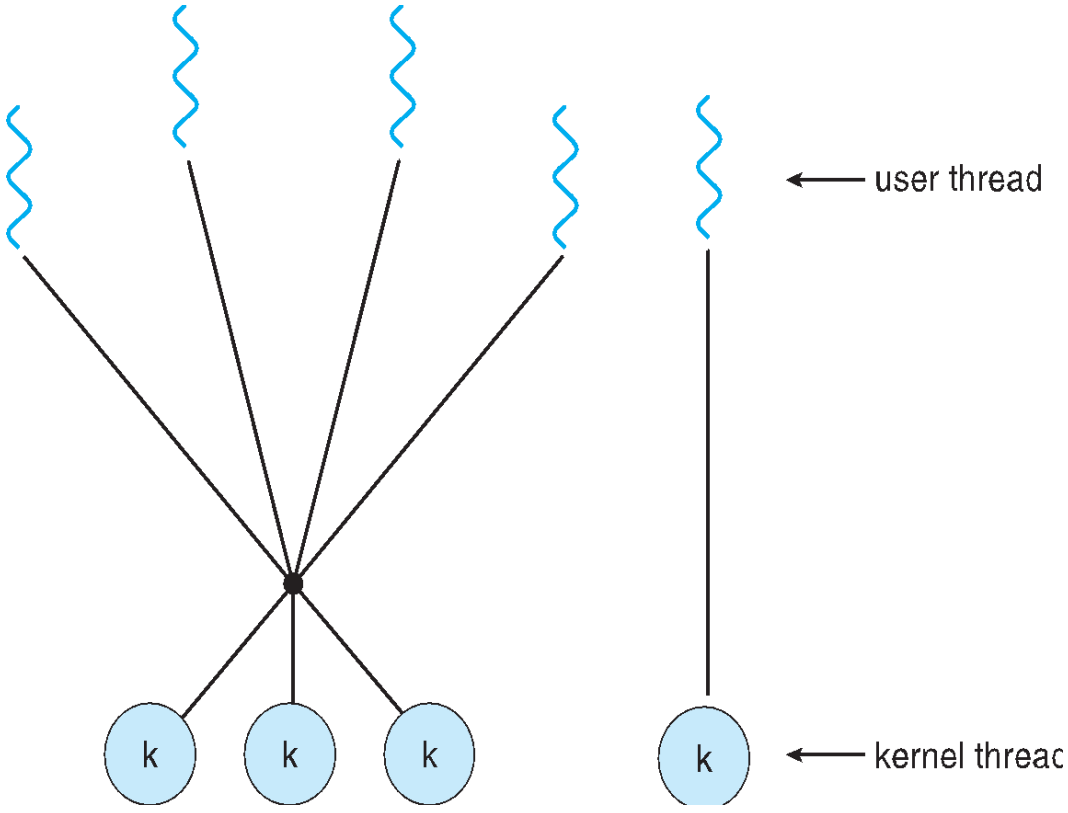

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later

# Many-to-Many Model




- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package

# Two-level Model




- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

# Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

# Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:

```
public interface Runnable
{
    public abstract void run();
}
```

  - Extending Thread class
  - Implementing the Runnable interface

# User threads vs Kernel threads

**User Threads:**

- Managed entirely by user-level libraries and applications, without direct involvement from the operating system kernel
- The kernel is unaware of the existence of individual user threads; it only sees the process as a single unit of execution.

**Kernel Threads:**

- Managed and scheduled directly by the operating system kernel.
- The kernel is aware of and manages each individual kernel thread.

# Multiprogramming vs Multithreading

**Multiprogramming**

- Running multiple programs (processes) on a single CPU by switching between them so that the CPU is never idle.
- Each program runs as a separate process.
- Processes do not share memory.

**Multithreading**

- Running multiple threads (smaller tasks) within a single program at the same time.
- Threads share the same memory of the parent process.
- Faster communication between threads compared to processes.
- Helps in performing multiple tasks inside the same program simultaneously.

# Concurrency vs Parallelism

**Concurrency**

- Concurrency is about managing multiple tasks at the same time, but not necessarily doing them simultaneously. It's like multitasking tasks start, pause, and resume, sharing the CPU.
- Programming Example:
  In a single-core CPU, a program can download a file and update the UI at the same time by switching between the tasks rapidly. This is concurrency, not true parallel execution.

**Parallelism**

- Parallelism is about doing multiple tasks at the exact same time. This requires multiple processors/cores.
- Programming Example:
  On a multi-core CPU, one core can process user input while another core downloads a file at the exact same time. This is true parallel execution.

# CPU Core vs CPU Thread

**CPU Core**

- A core is a physical processor inside the CPU.
- Each core can handle one task at a time

**CPU Thread**

- A thread is like a task or job that a core can work on.
- Some CPUs can handle 2 threads per core (this is called hyper-threading).
- So a single core can do multiple threads by switching between them quickly.

# Physical Memory vs Virtual Memory

**Physical Memory**

- The actual RAM chips installed in your computer.

**Virtual Memory**

- A combination of RAM and a space on your hard drive (called the page file or swap space) that your computer uses when RAM is full. It makes your computer think it has more memory than it physically does.

# End of Chapter 3