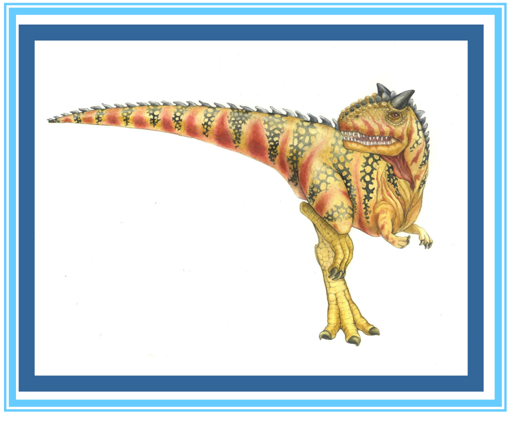
Chapter 4: Threads



Chapter 4: Threads

- Overview
- Multithreading Models
- Thread Libraries

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

What's in a process?

- A process consists of (at least):
  - an address space
  - the code for the running program
  - the data for the running program
  - an execution stack and stack pointer (SP)
    - traces state of procedure calls made
  - the program counter (PC), indicating the next instruction
  - a set of general-purpose processor registers and their values
  - a set of OS resources
    - open files, network connections, sound channels, …

Concurrency

- Imagine a web server, which might like to handle multiple requests concurrently
  - While waiting for the credit card server to approve a purchase for one client, it could be retrieving the data requested by another client from disk, and assembling the response for a third client from cached information
- Imagine a web browser, which might like to initiate multiple requests concurrently
  - While browser displays images or text, it retrieves data from the network.

  - A word processor
  - For example, displaying graphics, responding to keystrokes from the user, and performing spelling and grammar checking in the background.

What’s needed?

- In each of these examples of concurrency (web server, web browser, word processor):
  - Everybody wants to run the same code
  - Everybody wants to access the same data
  - Everybody has the same privileges (most of the time)
  - Everybody uses the same resources (open files, network connections, etc.)
- But you’d like to have multiple hardware execution states:
  - an execution stack and stack pointer (SP)
    - traces state of procedure calls made
  - the program counter (PC), indicating the next instruction
  - a set of general-purpose processor registers and their values

How could we achieve this?

- Given the process abstraction as we know it:
  - fork several processes
- This is really inefficient!!
  - Resource intensive□ ex: space: PCB, page tables, etc.
  - Time consuming□ creating OS structures, fork and copy address space, etc.
- So any support that the OS can give for doing multi-threaded programming is a win

Single-Threaded Example

- Imagine the following C program:

```
main() {
    ComputePI(“pi.txt”);
    PrintClassList(“clist.text”);
}
```

- What is the behavior here?
  - Program would never print out class list, because “ComputePI” would never finish.

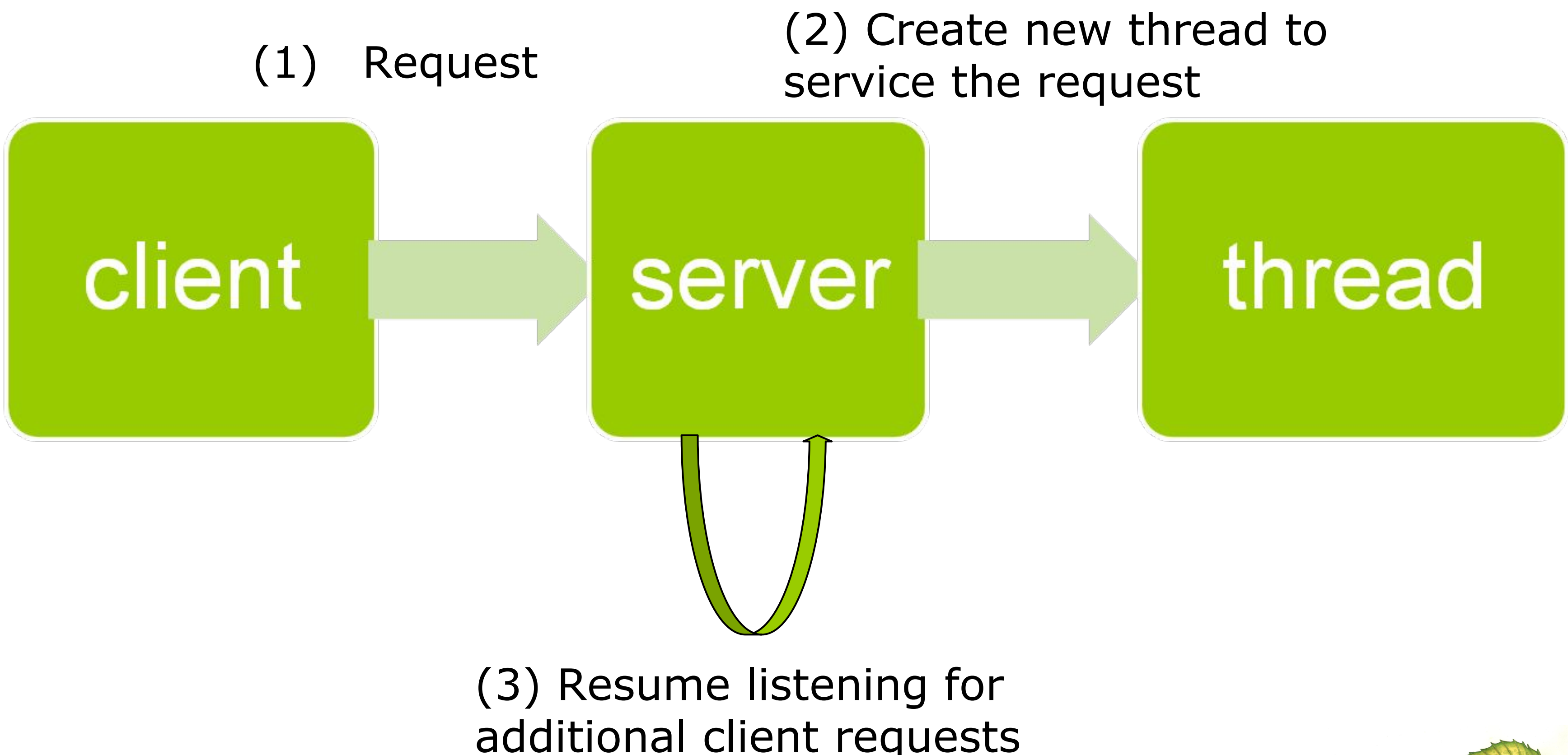
Use of Threads

- Version of program with Threads:

```
main() {
   CreateThread(ComputePI(“pi.txt”));
   CreateThread(PrintClassList(“clist.text”));
   }
```

- What does “CreateThread” do?
  - Start independent thread running for a given procedure
- What is the behavior here?
  - Now, you would actually see the class list
  - This *should* behave as if there are two separate CPUs

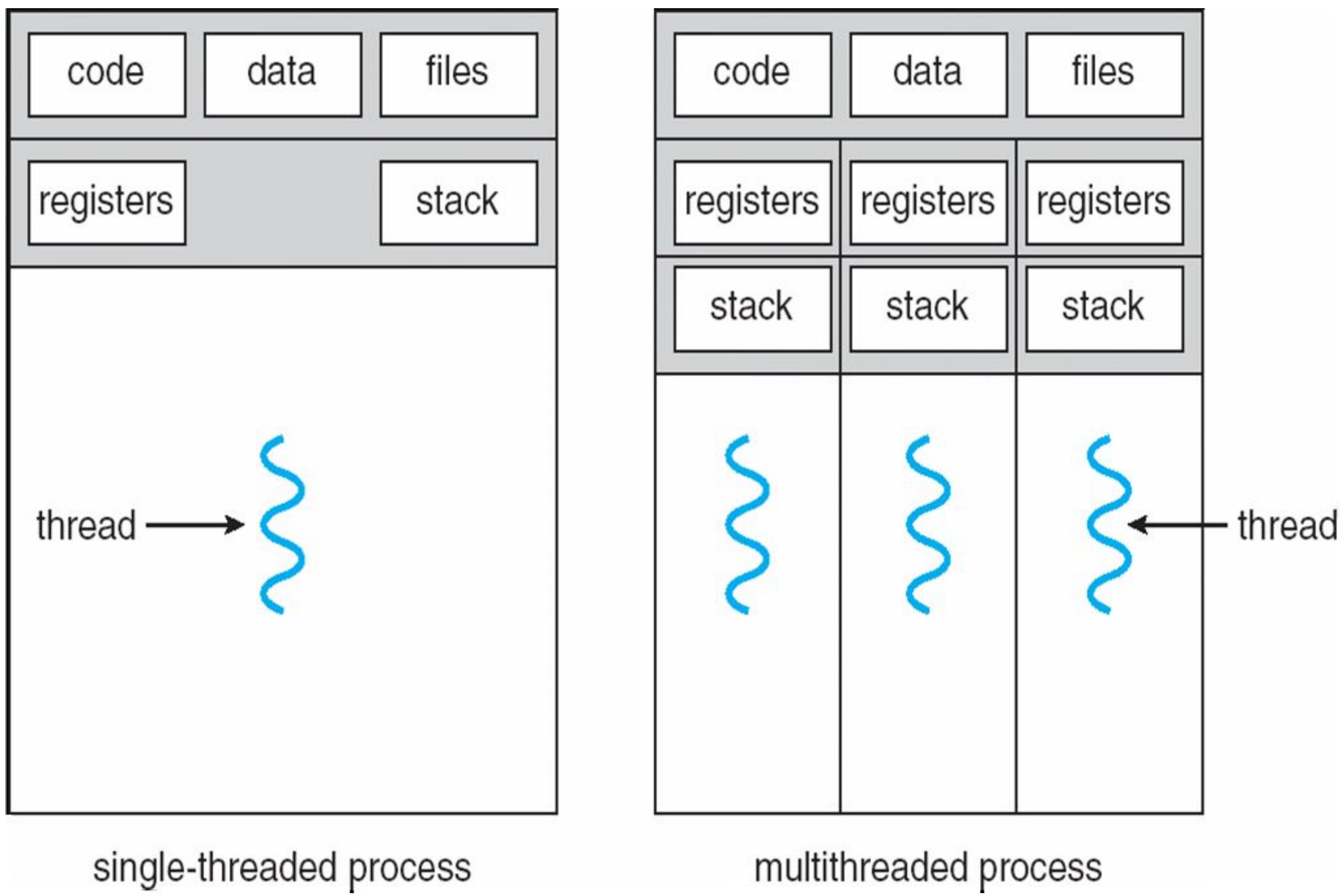
Multithreaded server architecture



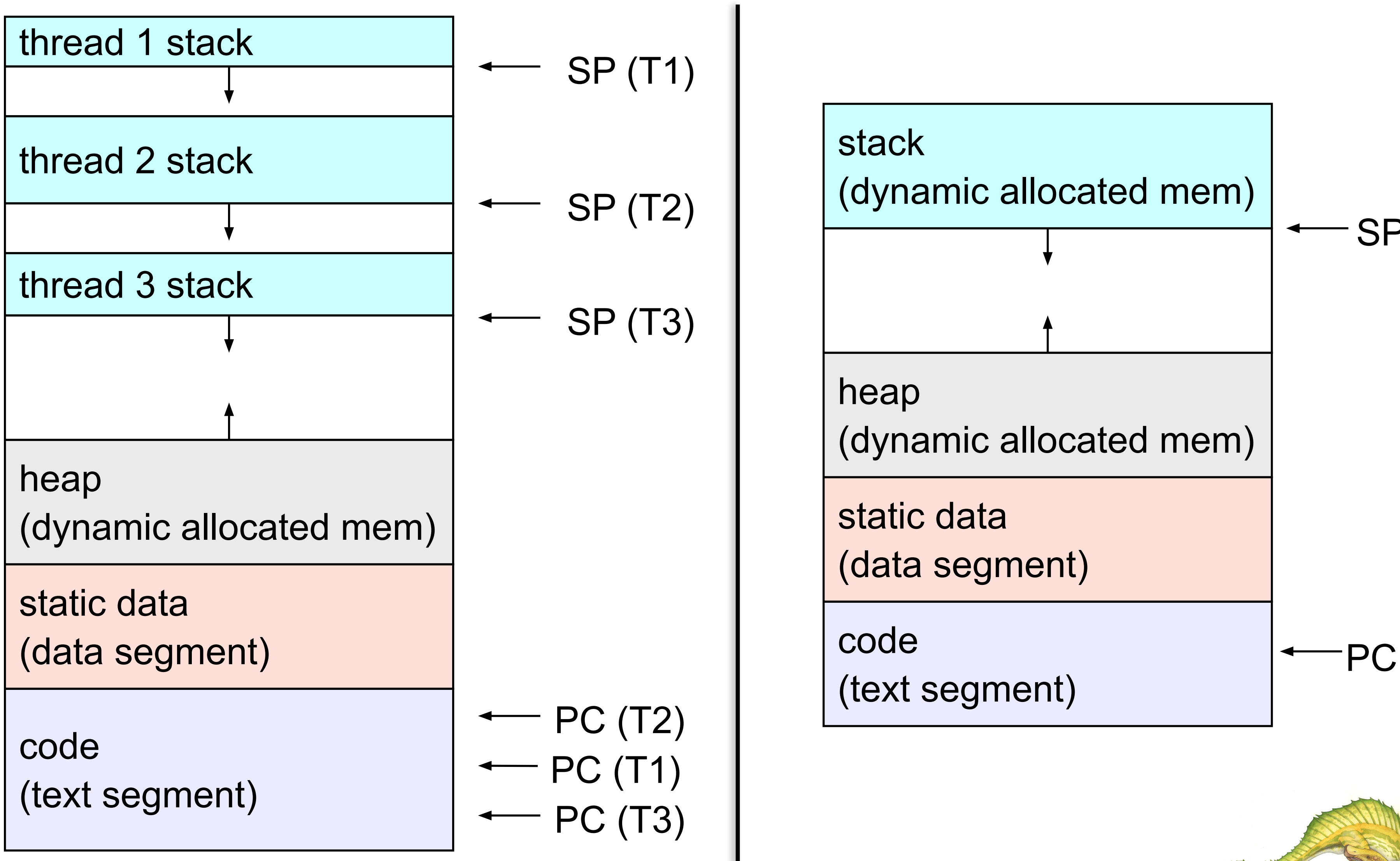
Threads and processes

- Most modern OS's (NT, modern UNIX, etc) therefore support two entities:
  - the process, which defines the address space and general process attributes (such as open files, etc.)
  - the thread, which defines a sequential execution stream within a process
- A thread
  - is a basic unit of CPU utilization; it comprises a thread ID, PC, a register set, and a stack.
  - Shares with other threads belonging to the same process its code and data sections, and other OS resources (ex: open files and signals)
  - Threads of the same process are not protected from each other.

Single and Multithreaded Processes



Process address space



Benefits of multithreaded

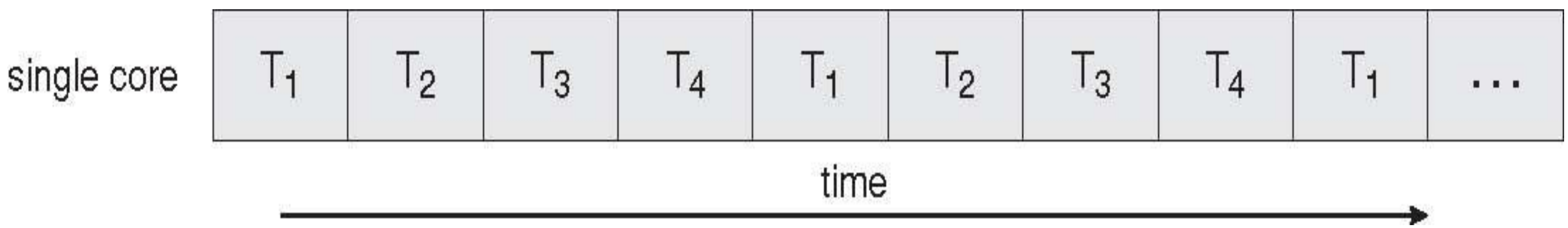
- Responsiveness:
  - A multithreaded interactive application allows a program to continue running even if part of it is blocked or performing a lengthy operation. Thereby increasing responsiveness to the user.
- Resource Sharing (code, data, files)
  - Threads share the memory and resources of the process to which they belong by default.
  - Sharing data between threads is cheaper than processes □ all see the same address space.
- Economy
  - Creating and destroying threads is cheaper than processes.
  - Context switching between threads is also cheaper.
  - It's much easier to communicate between threads.

Benefits of multithreaded

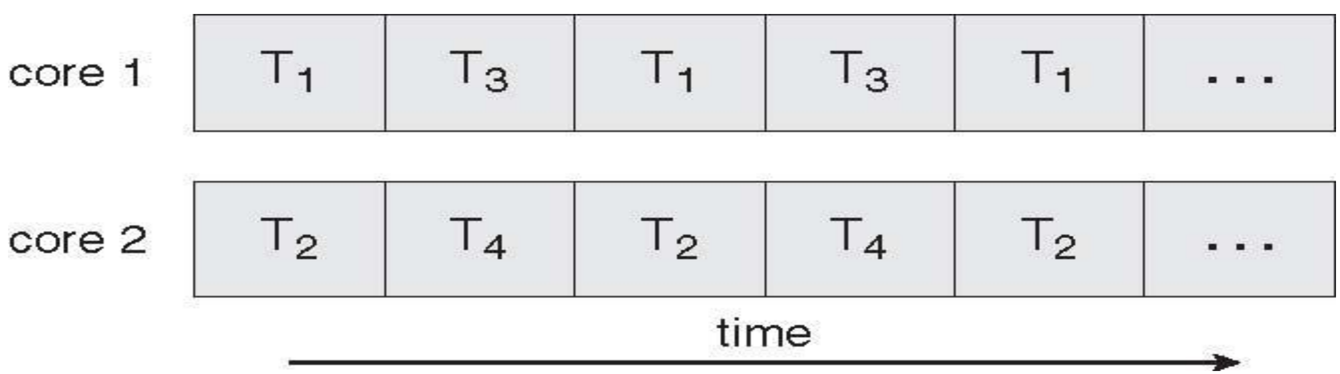
- Scalability
  - Multithreading can be greatly increased in a multiprocessor systems
  - Threads may be running in parallel on different processors.

Multicore Programming

- On a single-core system, concurrency means that the execution of threads will be interleaved over time – executing only one thread at a time.



- Parallel execution for threads on a multi-core system.



User and Kernel Threads

- User threads:
  - are visible to the programmer and unknown to the kernel.
  - thread management done by user-level threads library, without kernel support.

- Kernel threads:
  - Most OS kernels are multi-threaded.
  - Several threads operate in the kernel, each performing a specific task.
  - Ex: managing devices, interrupt handling.
  - Supported and managed directly by the Kernel.
  - Examples: Windows XP/2000, Solaris, Linux, Tru64 UNIX, Mac OS X.

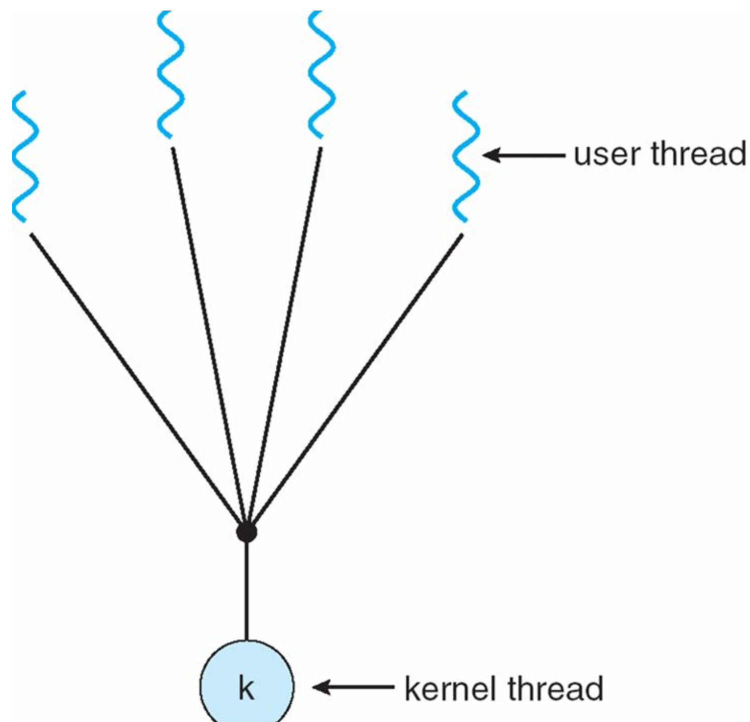
- User-level threads are faster to create and manage than are kernel threads.
  - Why?
    - Because no intervention from the kernel is required.

Multithreading Models

- A relationship must exist between user threads and kernel threads, established by one of three ways:
  - Many-to-One
  - One-to-One
  - Many-to-Many

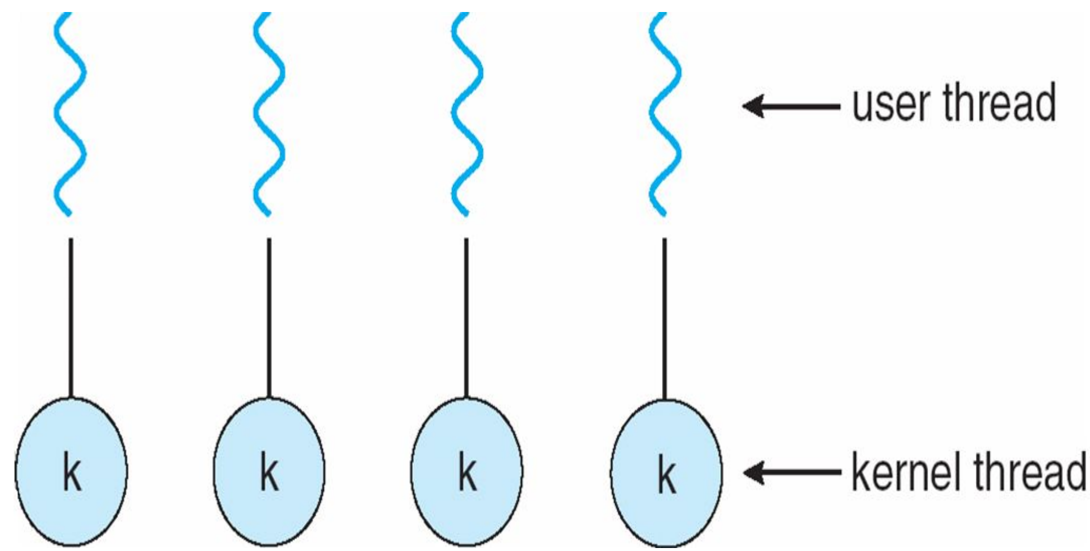
Many-to-One

- Many user-level threads mapped to single kernel thread:
  - Thread management is done by the thread library in user space efficient.
  - The entire process will block if a thread makes a blocking system call.
  - Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



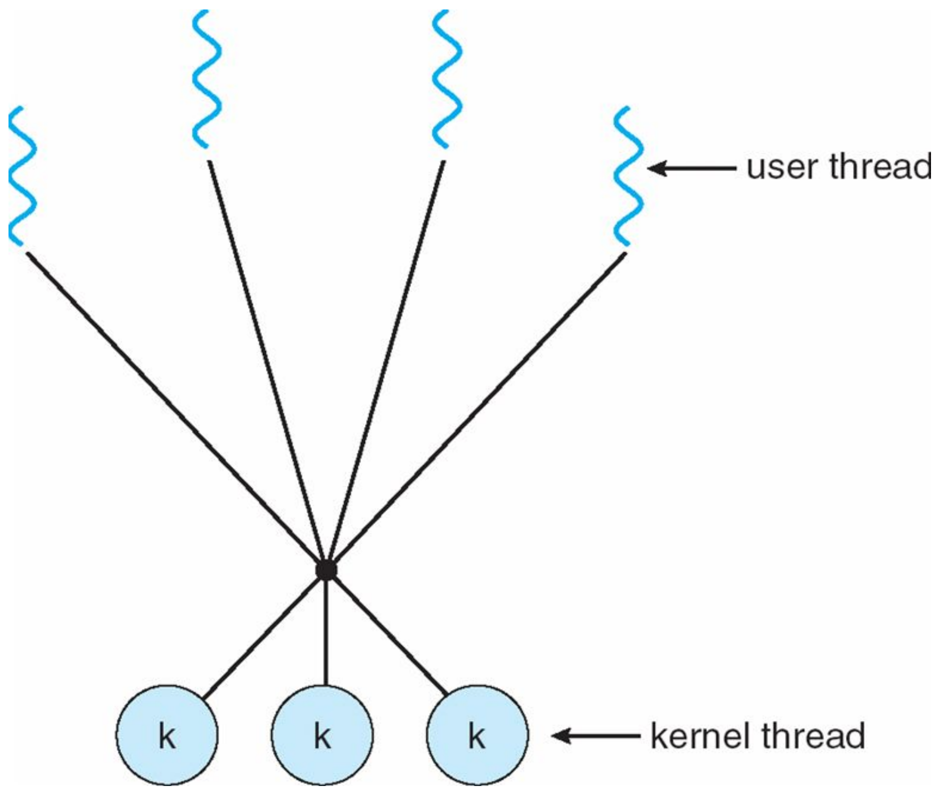
One-to-One

- Each user-level thread maps to kernel thread
  - Adv : allows another thread to run when a thread makes a blocking system call □ more concurrency.
  - Adv : allows multiple threads to run in parallel on multiprocessors.
  - Dis : creating a user thread requires creating corresponding kernel thread □ can burden the applications performance.
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later



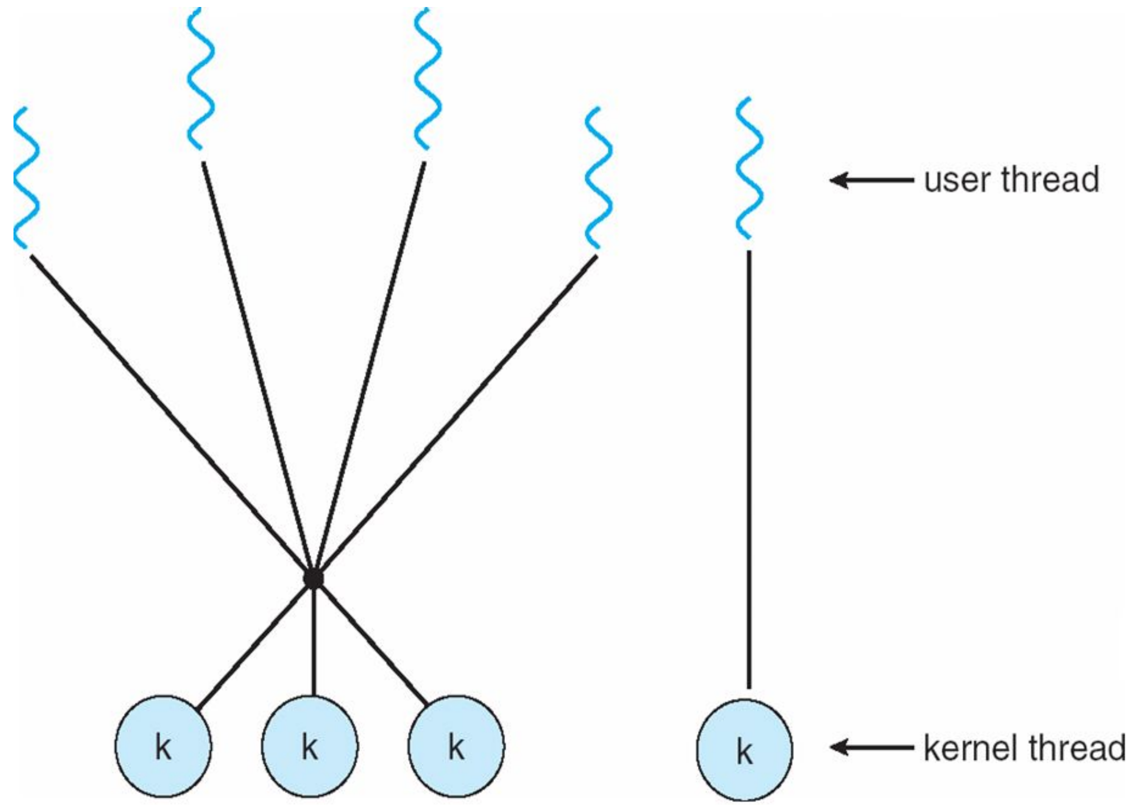
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Does not suffer from the shortcomings of the previous two models. How? read P159
- Solaris prior to version 9



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 a thread library:
  - Library entirely in user space (all code and data structures for the library in user space)
    - Invoking a function in the API ->local function call in user space and not a system call.
  - Kernel-level library supported by the OS (all code and data structures for the library in kernel space)
    - Invoking a function in the API -> system call to the kernel.

- Three primary thread libraries:
  - POSIX Pthreads (maybe KL or UL), common in UNIX operating systems
  - Win32 threads (KL), in Windows systems.
  - Java threads (UL), in JVM.

End of Chapter 4