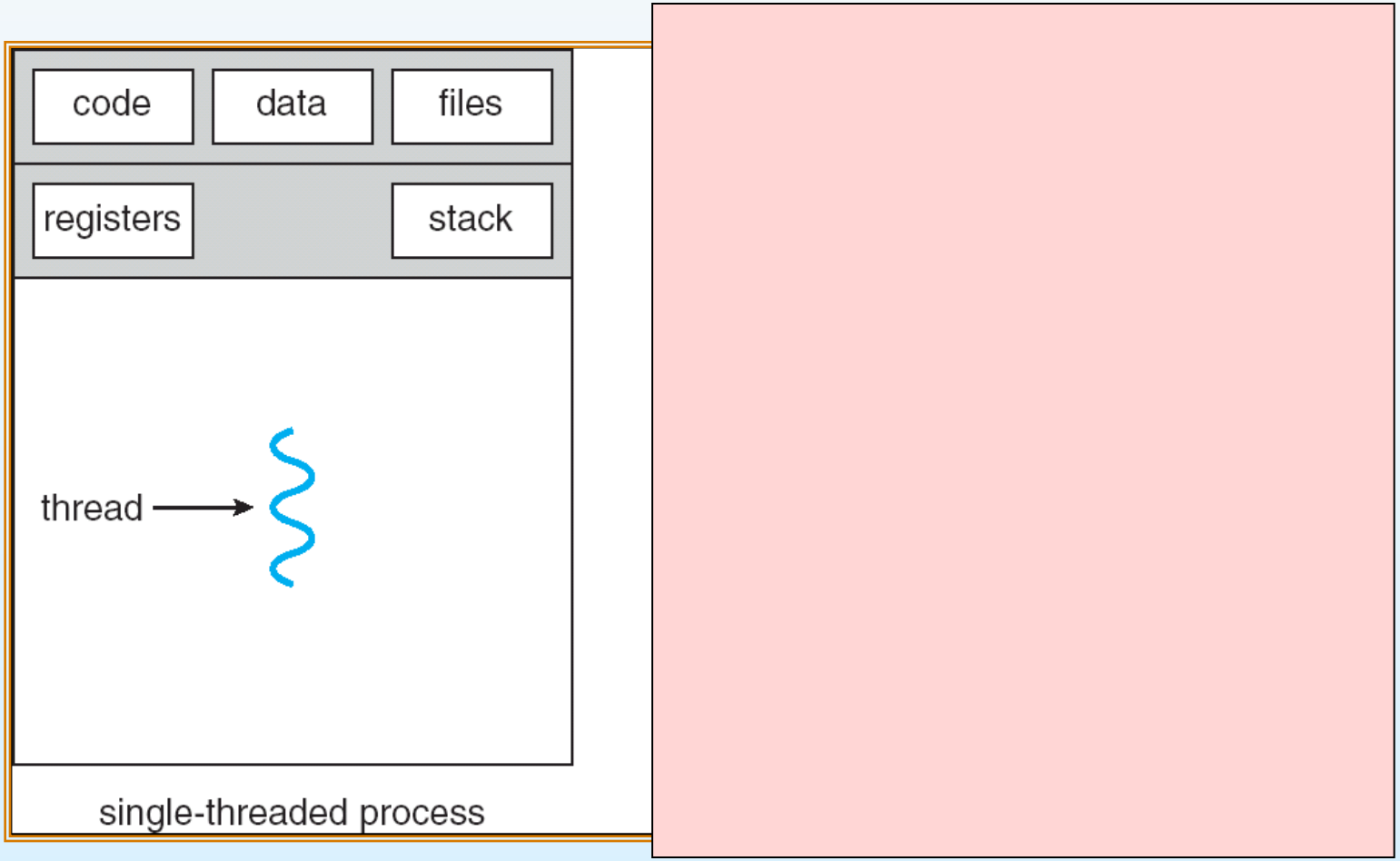
The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

Chapter 4: Threads

Chapter 4: Threads

- Overview
- Multithreading Models
- Threading Issues
- Pthreads
- Windows XP Threads
- Linux Threads
- Java Threads

Single and Multithreaded Processes



Benefits

- Responsiveness
- Resource Sharing
- Utilization of MP & Multicore Architectures

Types

- User-level thread
- Kernel-Level Thread

User Threads

- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

Kernel Threads

- Supported by the Kernel
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

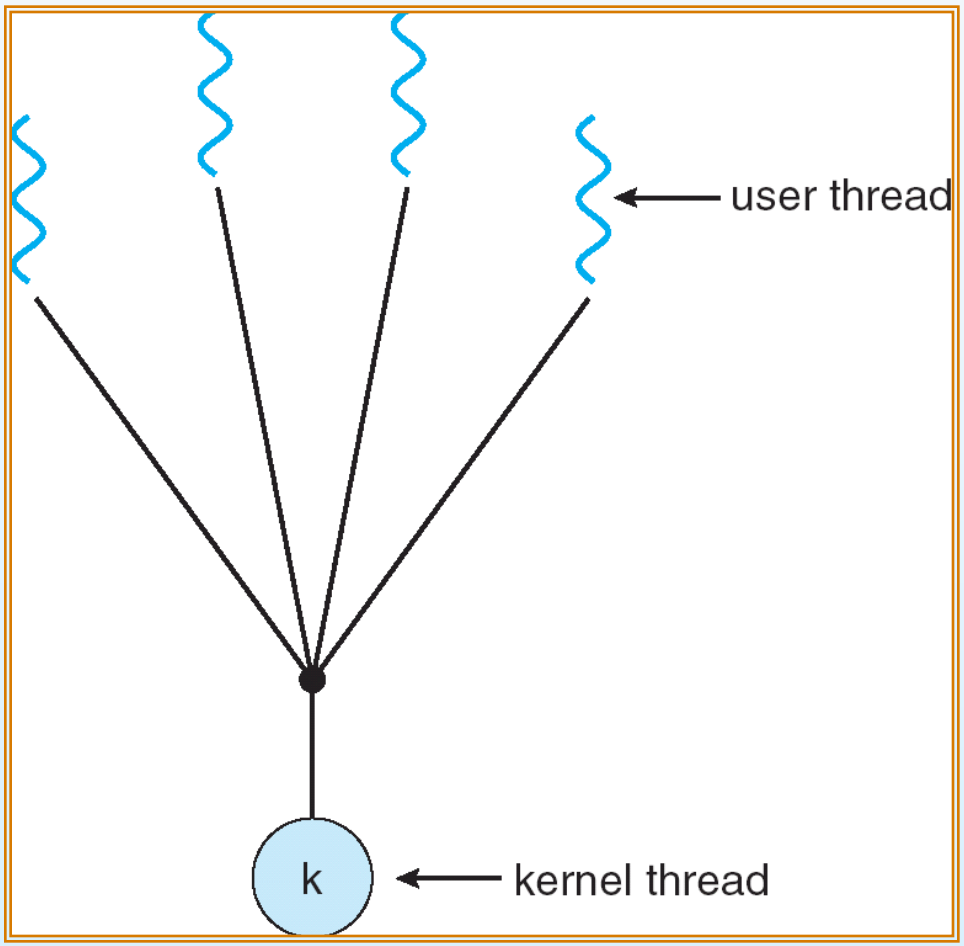
How user-level threads are mapped to kernel ones.

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

Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

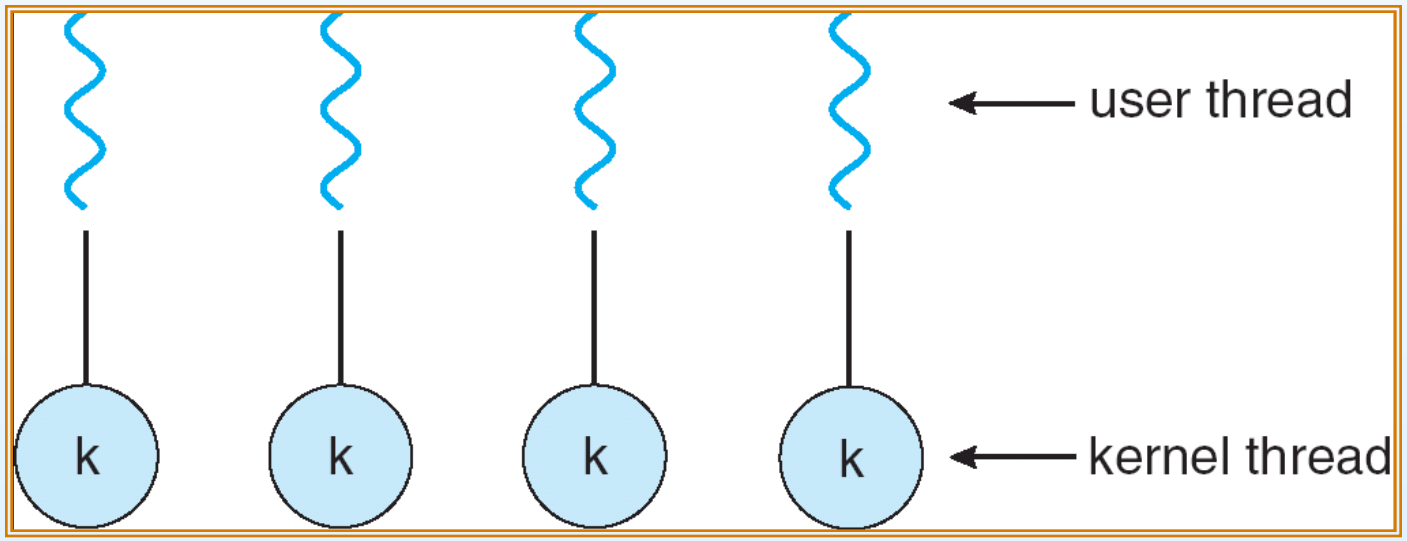
Many-to-One Model



One-to-One

- Each user-level thread maps to kernel thread
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

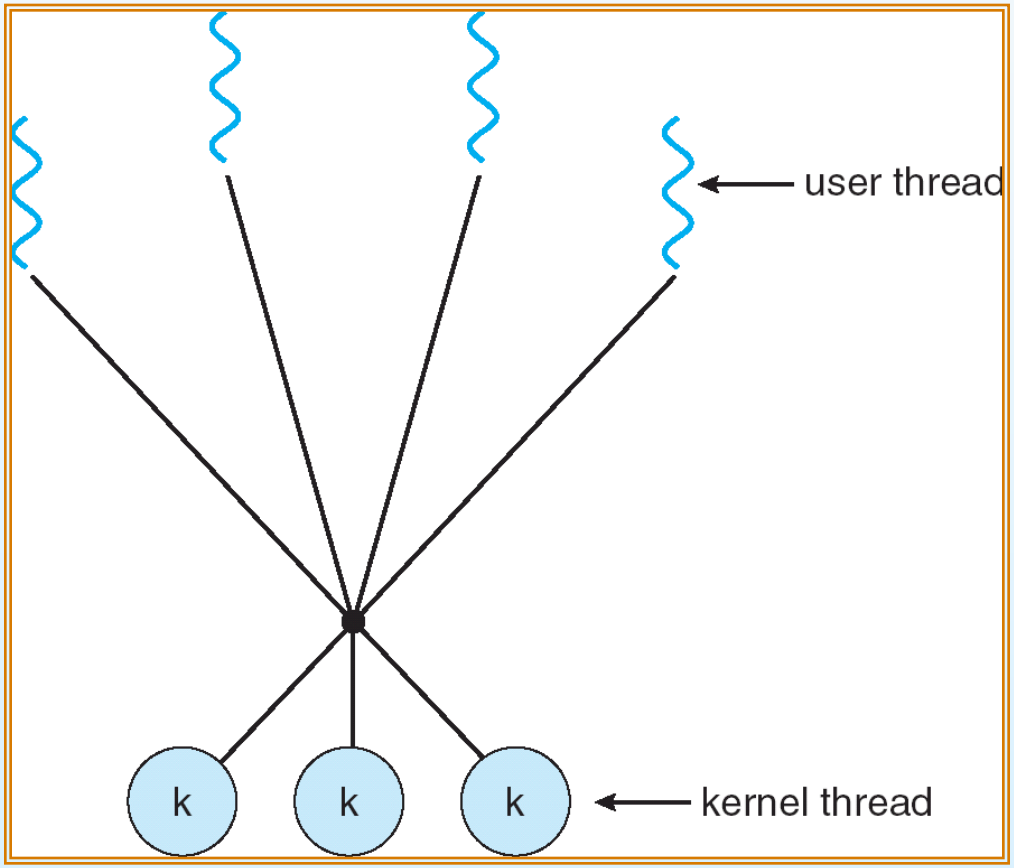
One-to-one Model



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 NT/2000 with the *ThreadFiber* package

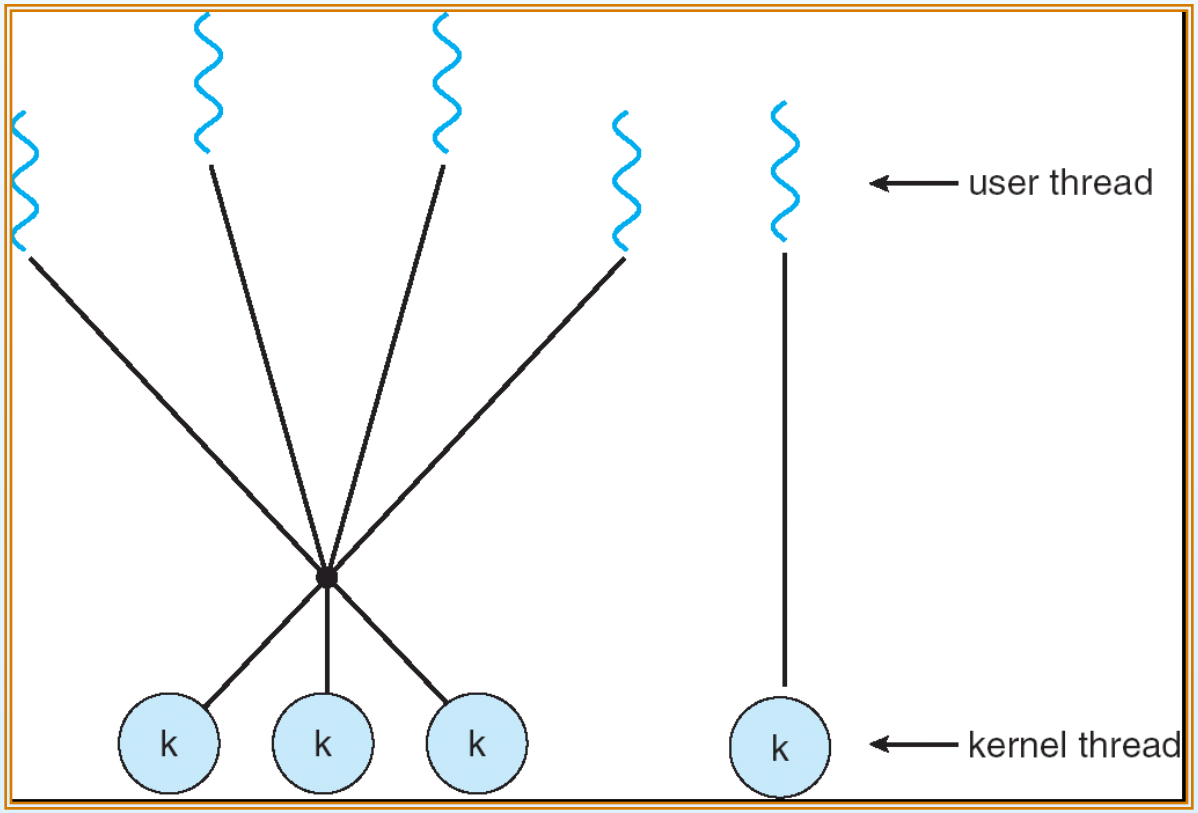
Many-to-Many Model



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

Two-level Model



Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Thread cancellation
- Signal handling
- Thread pools
- Thread specific data
- Scheduler activations

Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?

Thread Cancellation

- Terminating a thread before it has finished
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Thread Programming Paradigms

- On-demand - create a thread whenever you need
  - Easy to program
  - More overheads
- Thread pool - create a pool of threads, and then assign tasks to them.
  - More efficient
  - Difficult to program due to you have to manage threads in your code

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages: (over thread on demand approach)
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool

Thread Specific Data

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the **context** of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

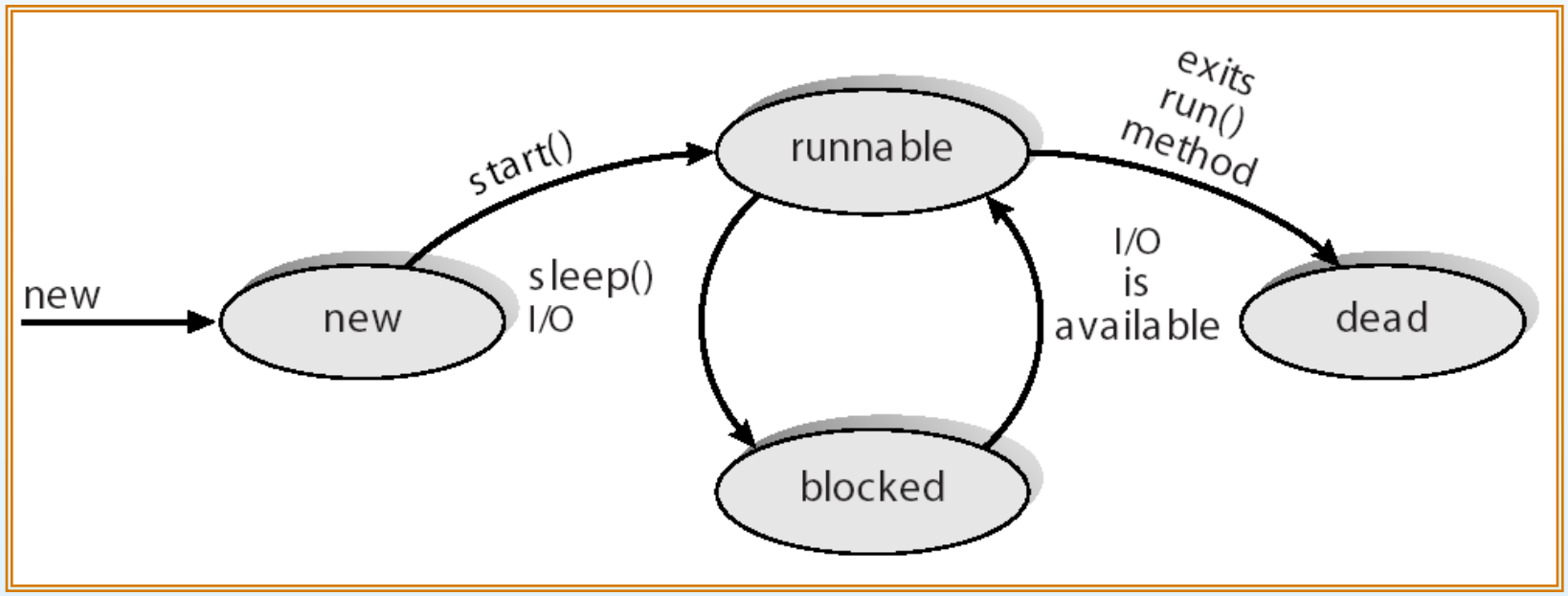
Linux Threads

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

Java Thread States

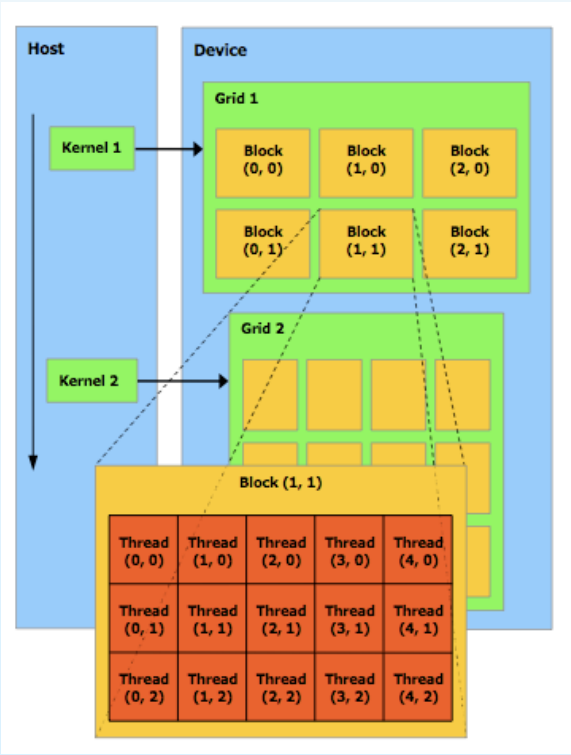


CUDA Thread

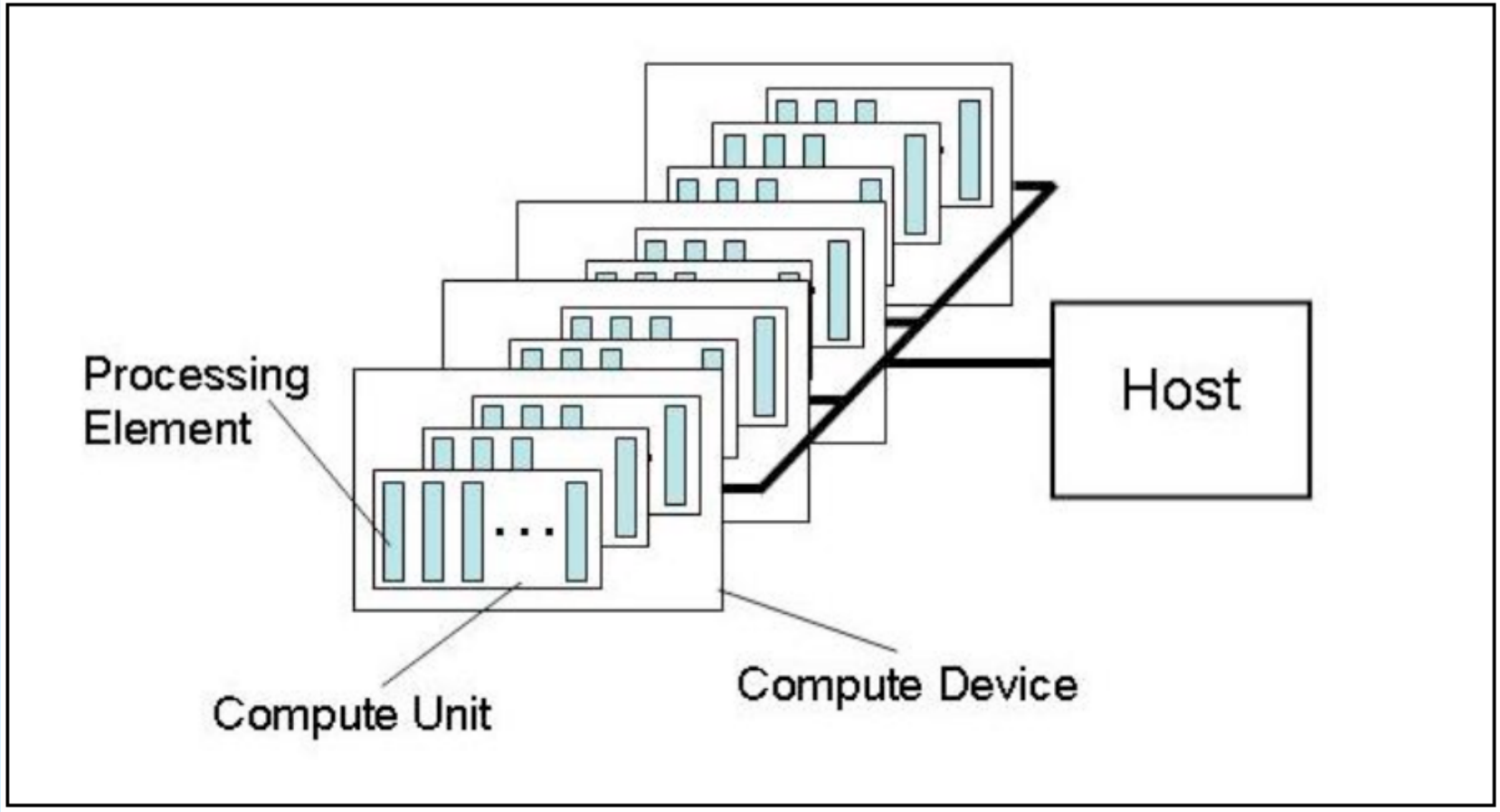
- CUDA is a language extension supported for GPGPU
- Only for NVDIA GPU
- Extremely lightweight

GPGPU Programming with CUDA

- CUDA (Compute Unified Device Architecture) is a SDK and API that allow a programmer to write C and Fortran programs to execute on GPGPU.
- Works with NVIDIA G80 or later and Tesla
- The GPGPU is viewed as a compute device



MC+GPU Platform Model



CPUs+GPU platforms

Programming Model: A Massively Parallel Coprocessor

NVIDIA

- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs 1000's of threads in parallel
- Data-parallel portions of an application execute on the device as kernels which run many cooperative threads in parallel
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
    - Instruction level thread switching
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few
  - Threads are non-persistent
    - Run and exit

This slide is from NVDIA CUDA tutorial

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

End of Chapter 4