Chapter 4: Threads

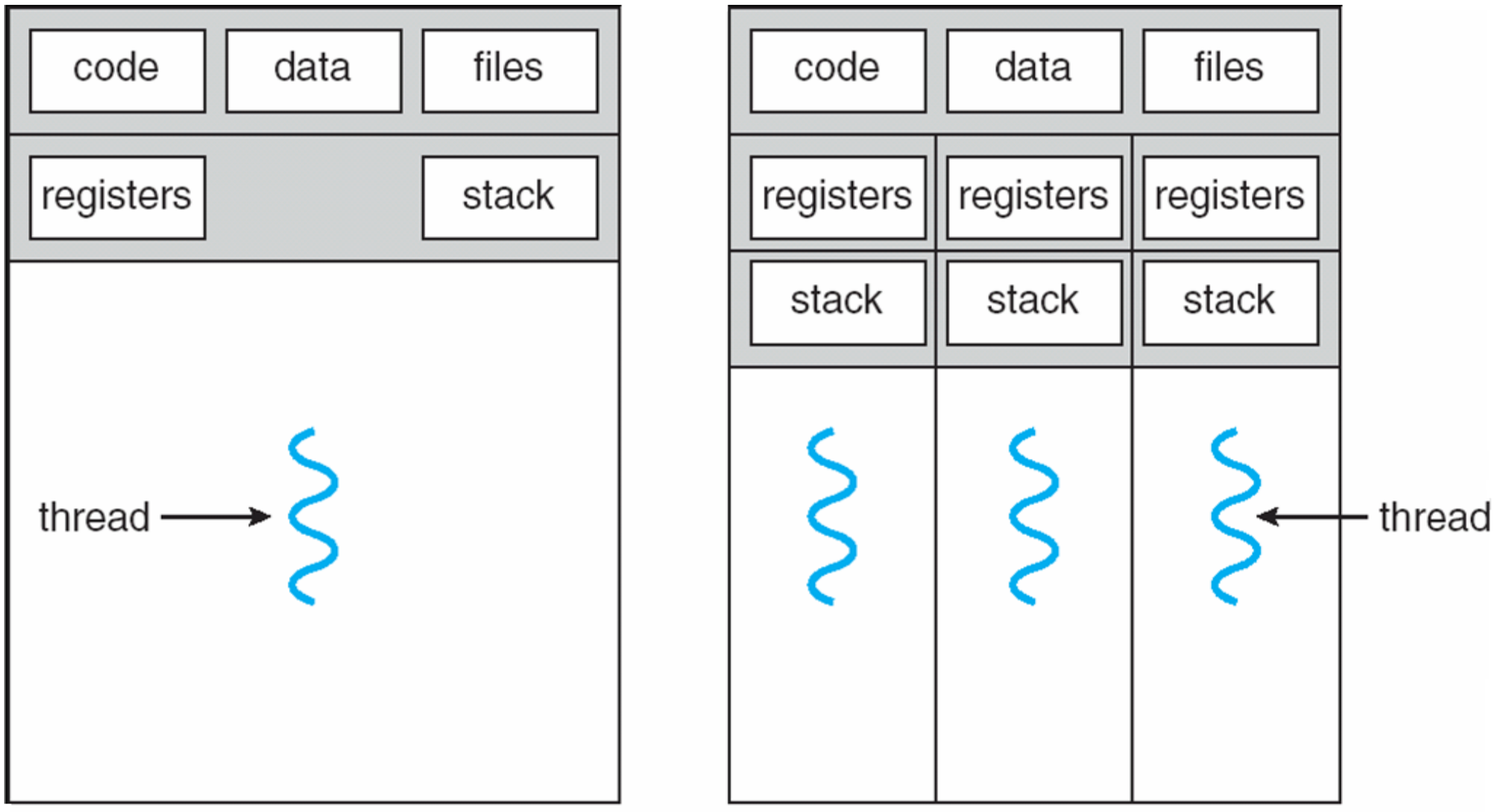
Threads: Basics

- Threads belong to the same process
  - Code section, data section, other operating system resources
  - Heavyweight process has a single thread of control

- Multiple tasks with the application can be implemented by separate threads
  - For example Web Browser, Word Processor/Powerpoint
    - Update display images
    - Answer a network request
    - Fetch data from keystrokes, display graphics
    - Spell checking

- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Single and Multithreaded Processes



single-threaded process

multithreaded process

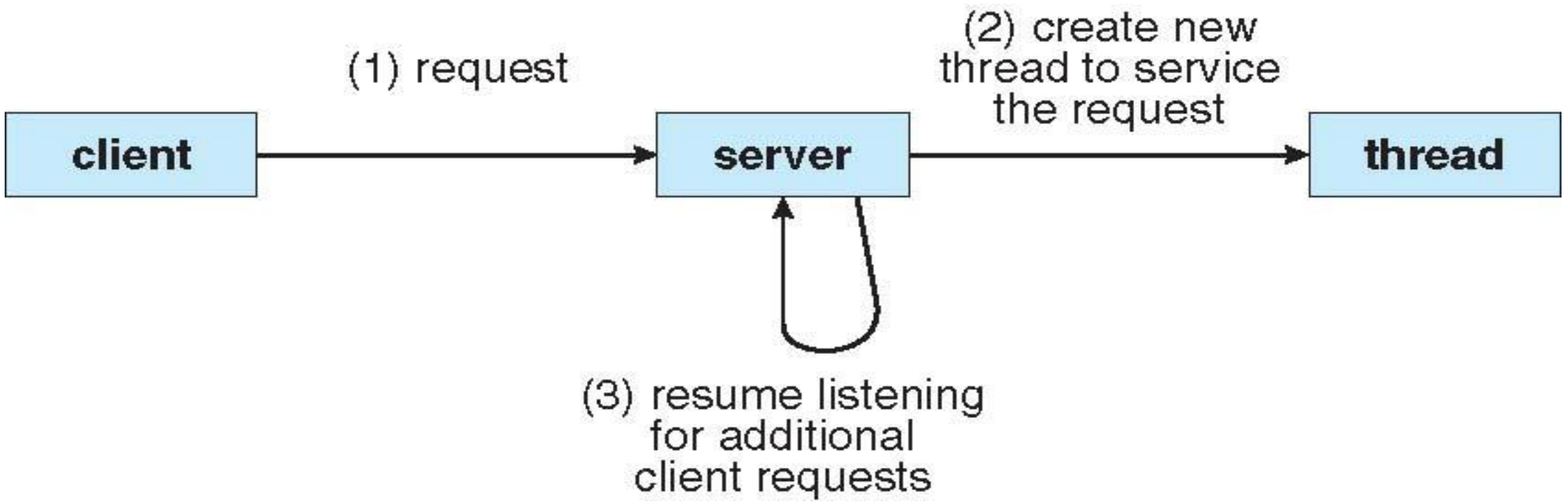
Benefits

- 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.
  - In Solaris, for ex: creating a process is thirty times slower than thread and context switching is about five times slower.

- Scalability – Process can take advantage of multicore architectures. Multiple threads can be running on multiprocessor/ multicore architecture called Multithreading

Multithreaded Server Architecture



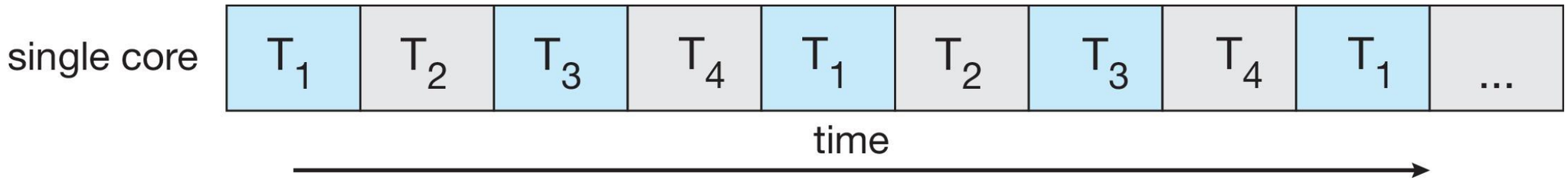
Multicore Programming

- Multicore or multiprocessor systems puts pressure on programmers, challenges include:
  - Dividing activities:
    - Examining applications that can divided and run on parallel on individual cores
  - Balance
    - Equal work and equal value for the division of process
  - Data splitting
    - Just as application, the associated data required must be divided to run on separate cores.
  - Data dependency
    - To check the data dependency among the tasks
  - Testing and debugging
    - Testing and debugging for multi-thread execution is more difficult than single threaded applications.

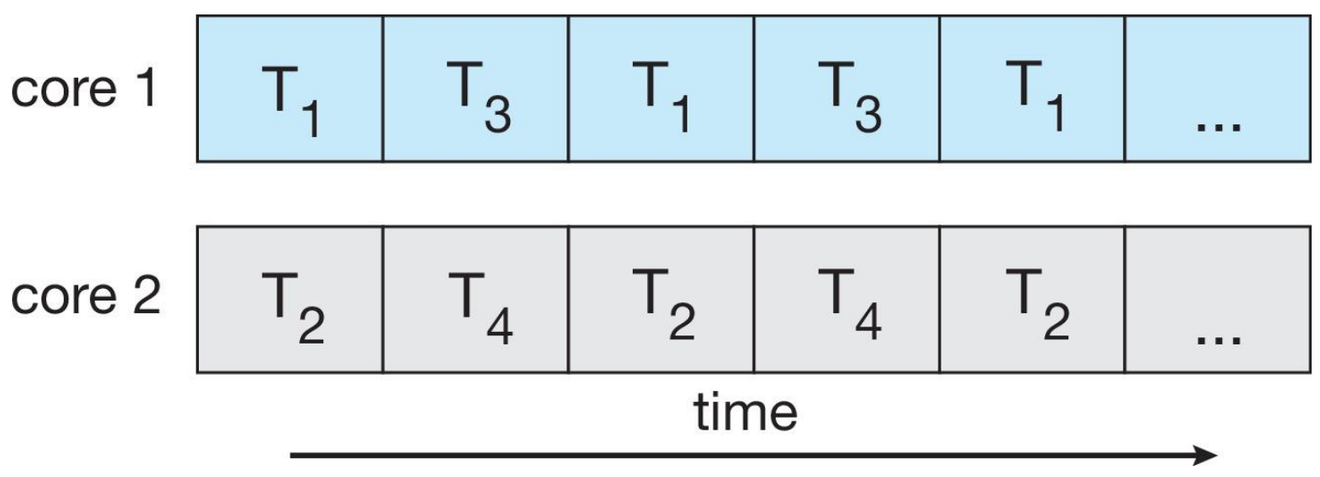
- *Parallelism* implies a system can perform more than one task simultaneously
- *Concurrency* supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Concurrency vs. Parallelism

- Concurrent execution on single-core system:

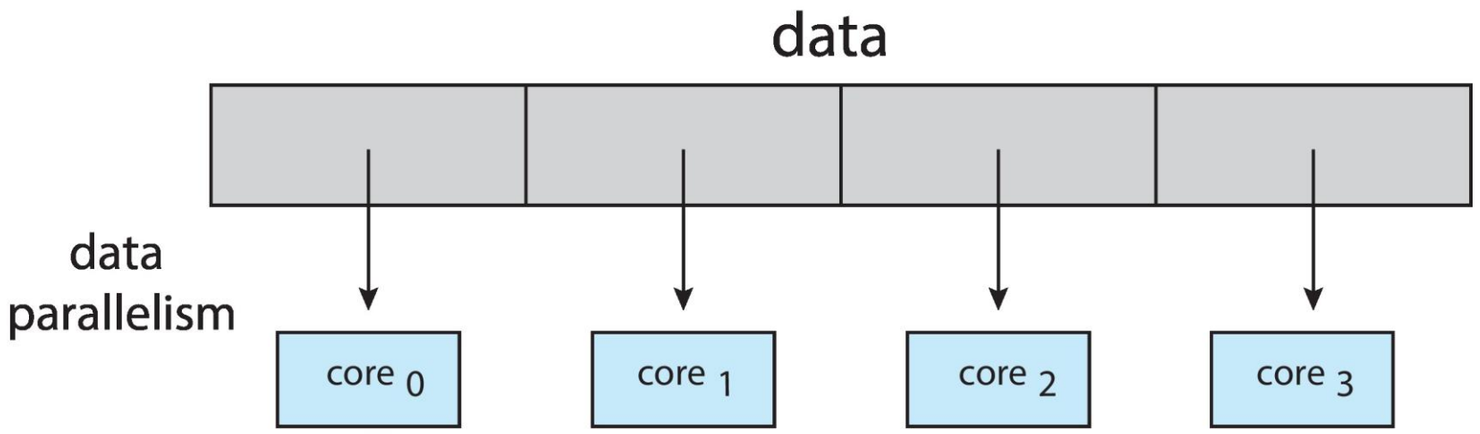


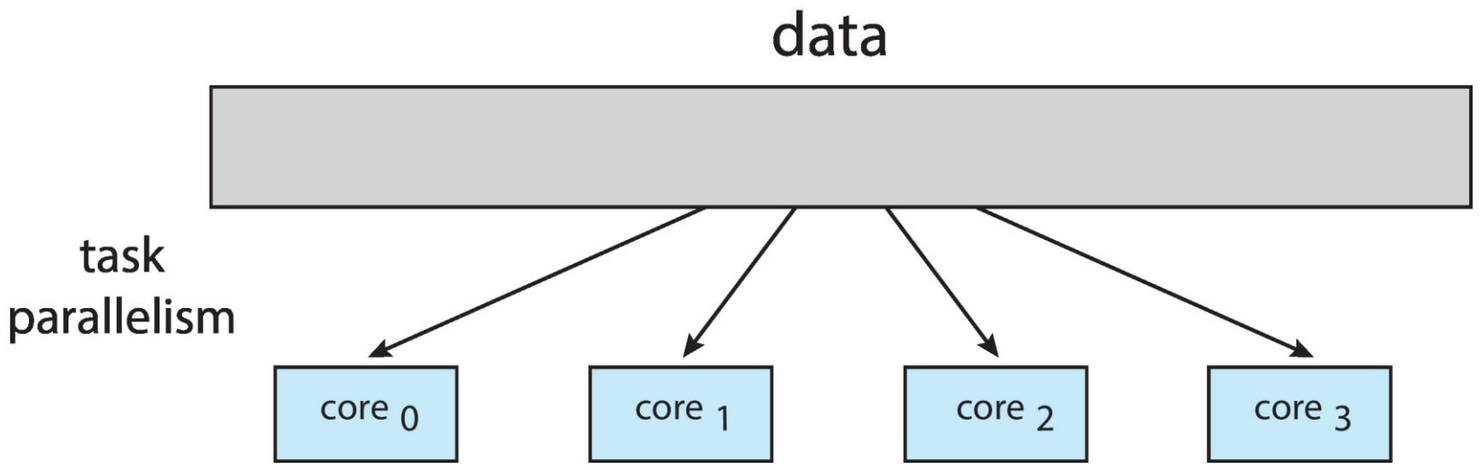
- Parallelism on a multi-core system:



Multicore Programming

- 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



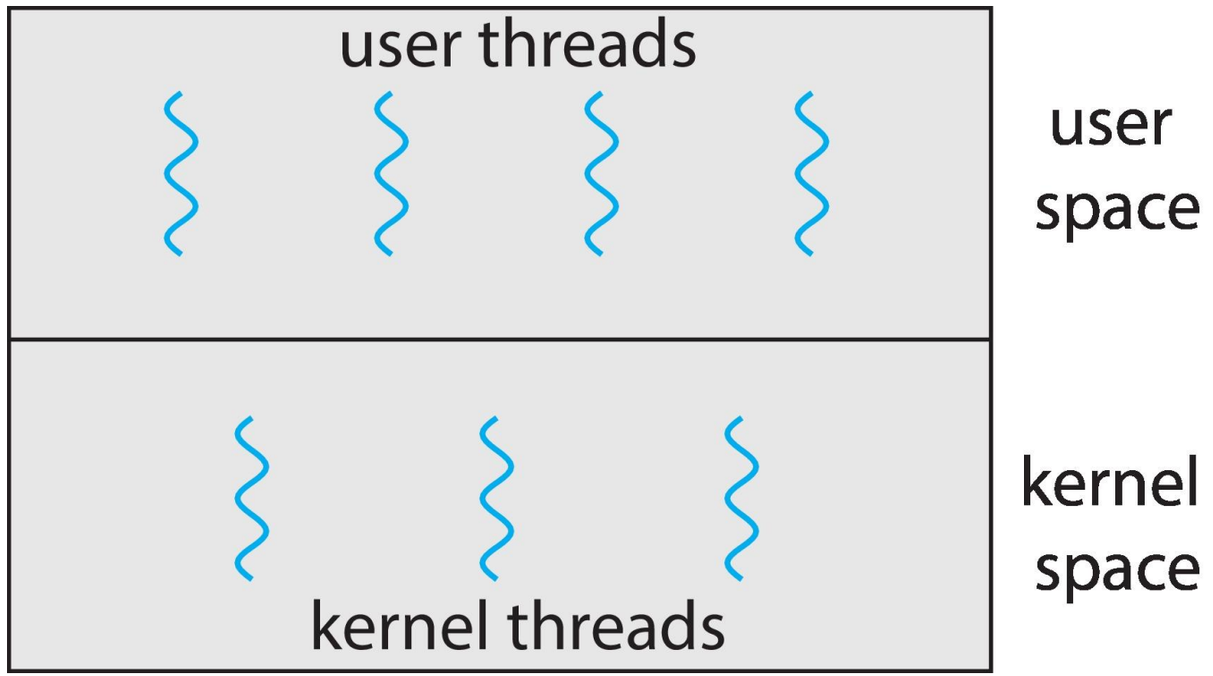


User Threads and Kernel Threads

- 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 managed by OS
- Examples – virtually all general-purpose operating systems, including:
  - Windows
  - Linux
  - Mac OS X
  - iOS
  - Android

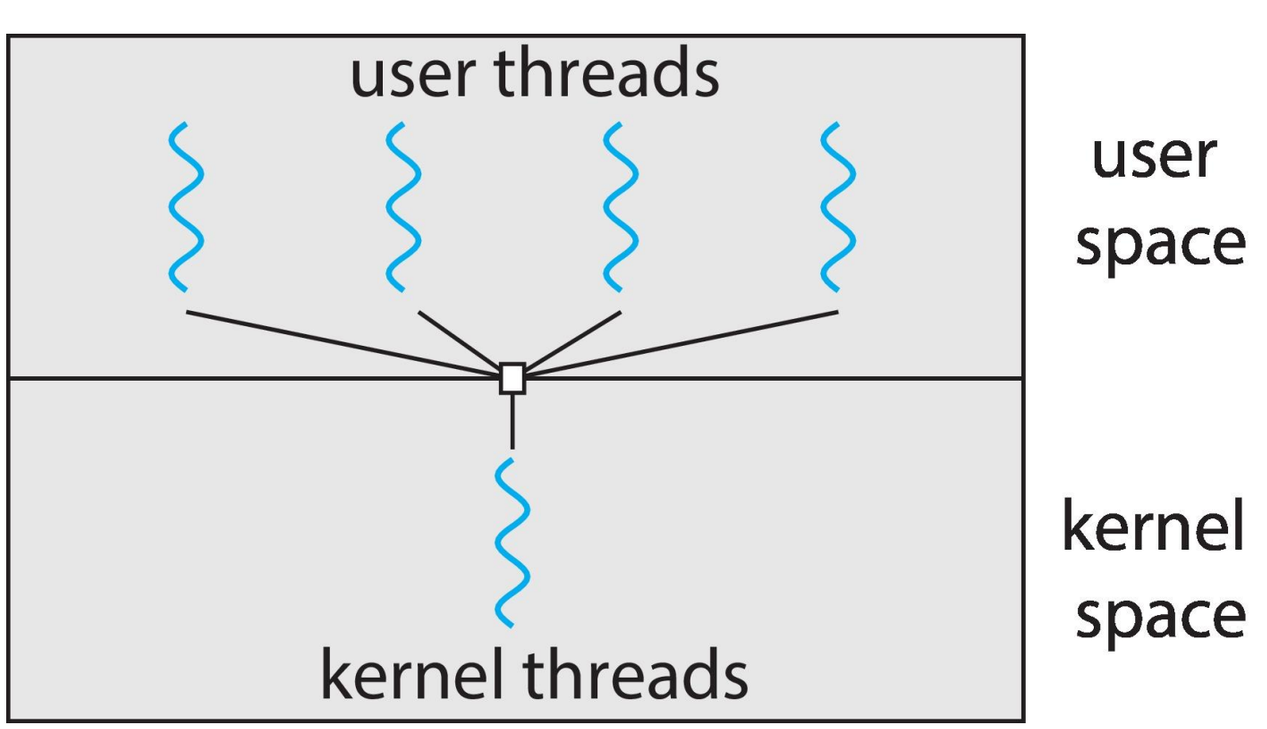
User and Kernel Threads



- Multithreading Models
  - Many-to-One
  - One-to-One
  - Many-to-Many

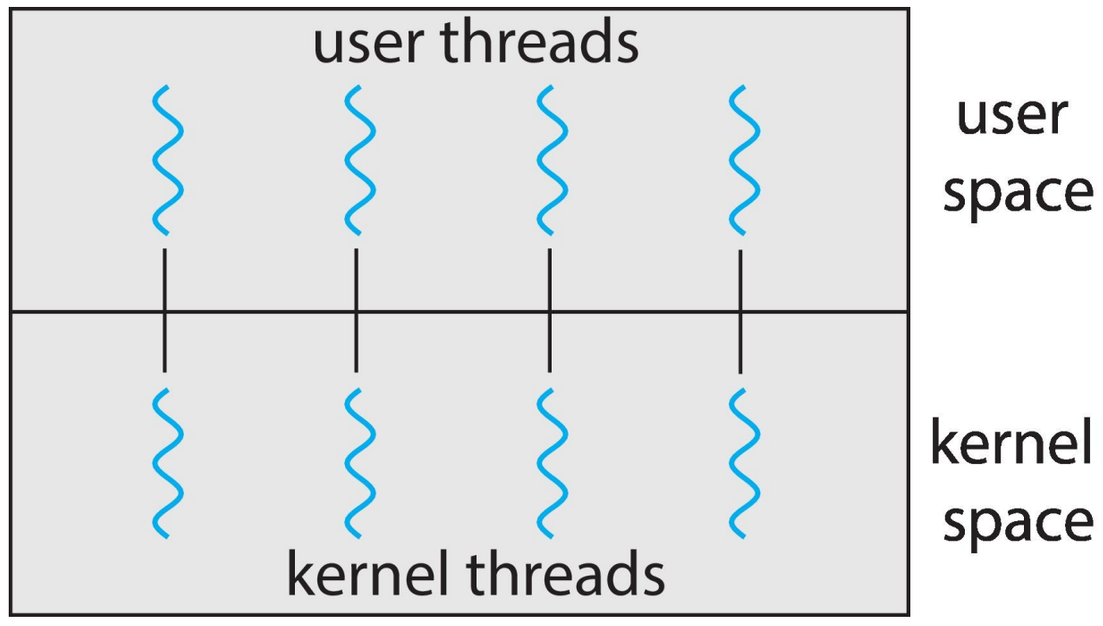
Many-to-One

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



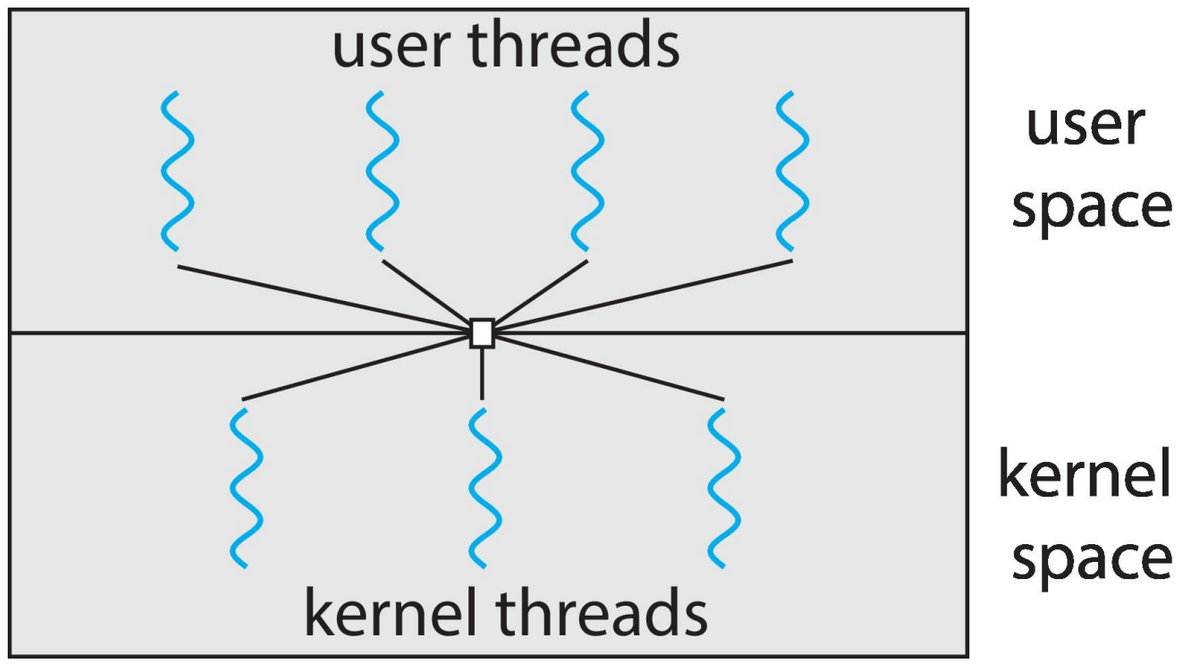
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Drawback is for each user thread a kernel thread is created
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux



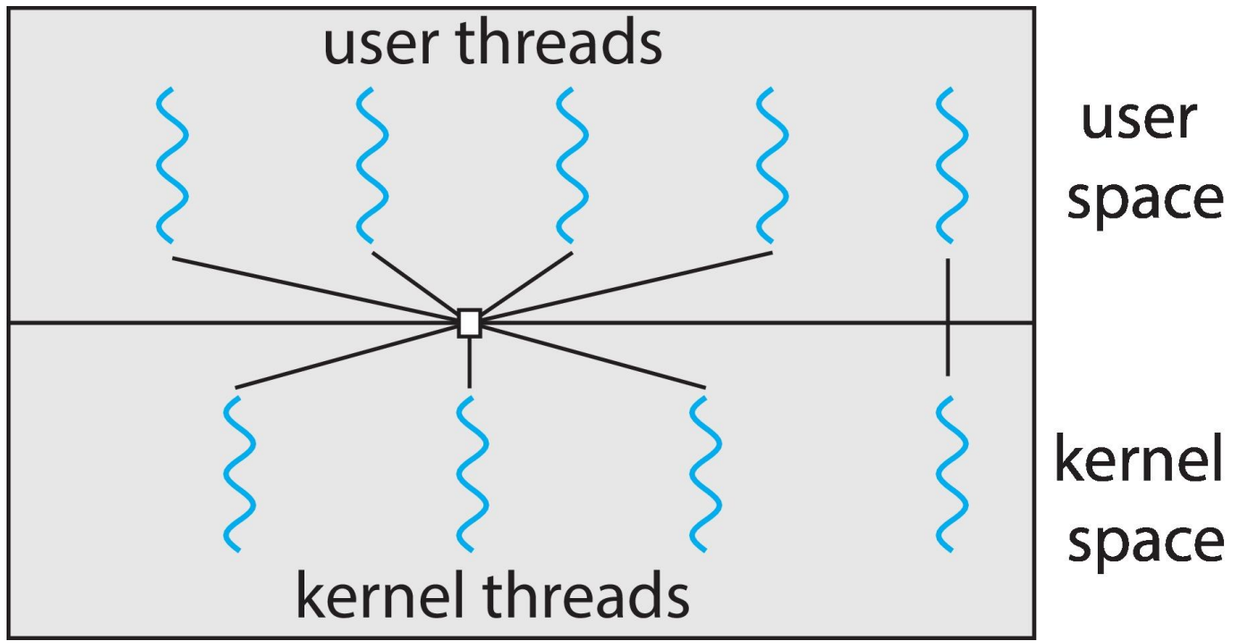
Many-to-Many Model

- Allows many user level threads to be mapped to many (smaller or equal) to kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the *ThreadFiber* package
- Otherwise not very common



Two-level Model

- Similar to many to many model, except that it allows a user thread to be bound to kernel thread



Thread Cancellation

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

Multiple-Processor Scheduling – Processor Affinity

- When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.
- We refer to this as a thread having affinity for a processor (i.e., “processor affinity”)

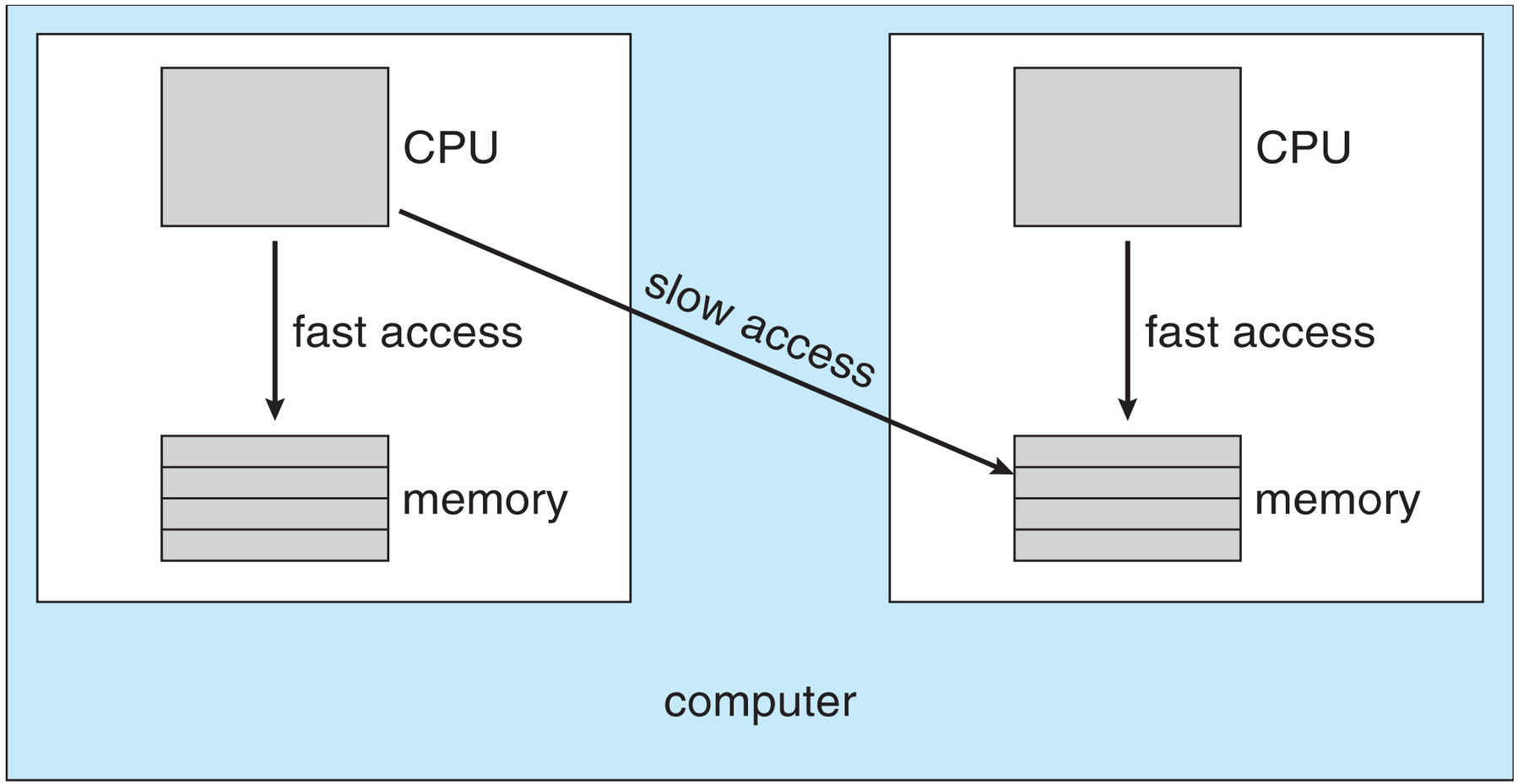
- Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off of.

- Soft affinity – the operating system attempts to keep a thread running on the same processor, but no guarantees.

- Hard affinity – allows a process to specify a set of processors it may run on and not to migrate on other processors.

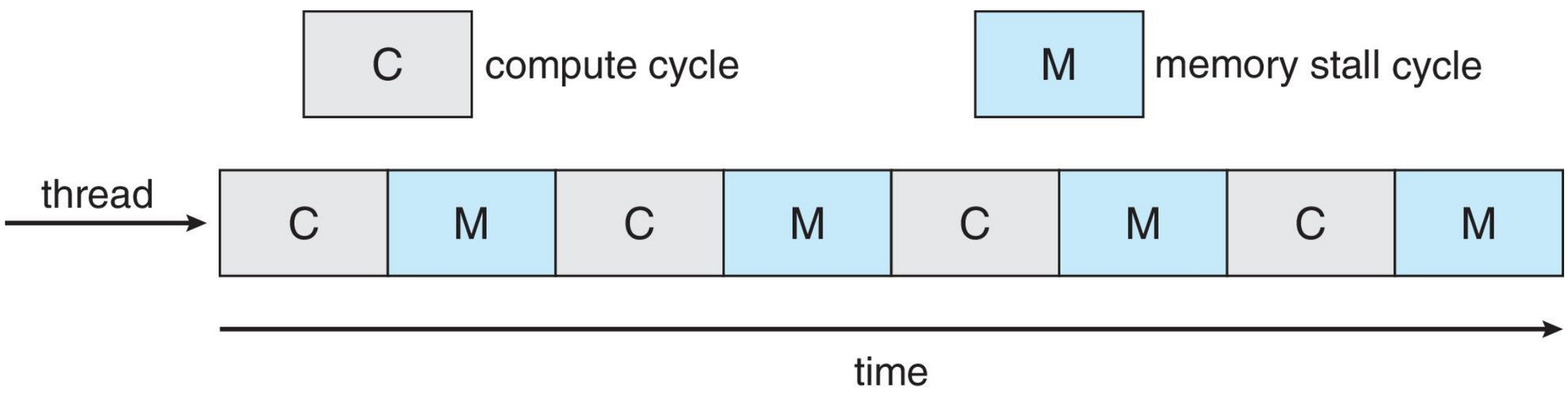
NUMA and CPU Scheduling

If the operating system is NUMA-aware, it will assign memory closes to the CPU the thread is running on.



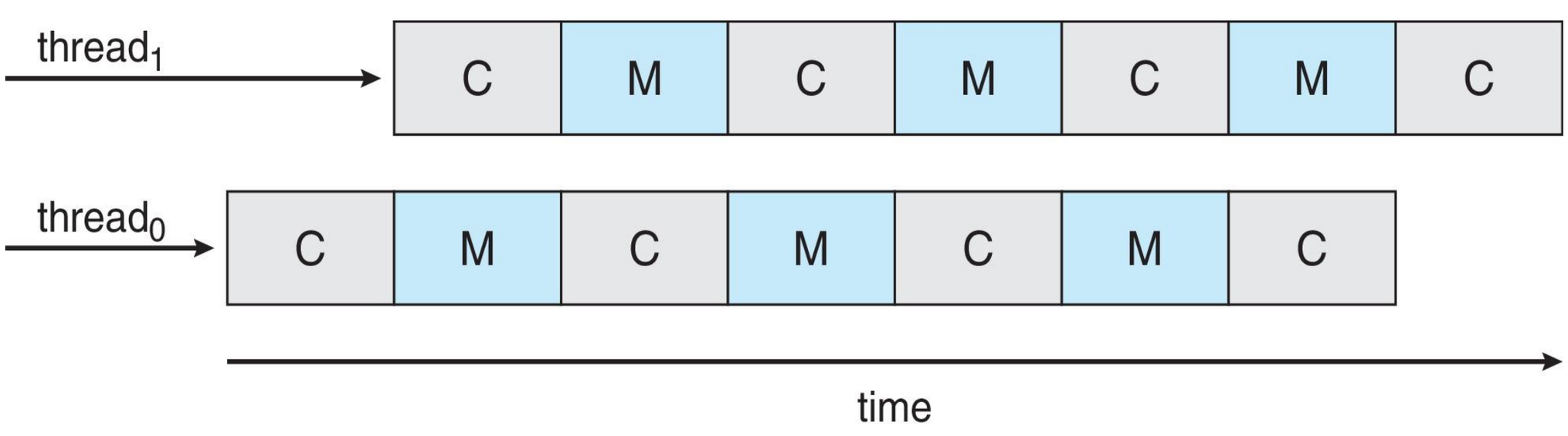
Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- A processor spends significant amount of time waiting for the data to be accessed by the memory called as memory stall
    - Cache miss (data not available in cache)
- Multiple threads per core also growing
    - Takes advantage of memory stall to make progress on another thread while memory retrieve happens



Multithreaded Multicore System

- Each core has > 1 hardware threads.
- If one thread has a memory stall, switch to another thread!
- Coarse grained- a thread executes until a long latency such as memory stall
- Fined grained- Switches to another thread at the completion of instruction cycle.



Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing