Amirkabir University of Technology
(Tehran Polytechnic)

# Operating Systems

# Threads and Concurrency

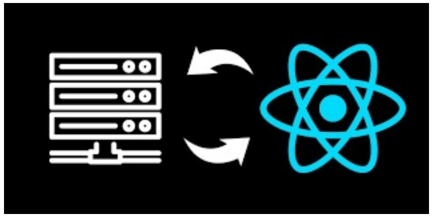
Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2021

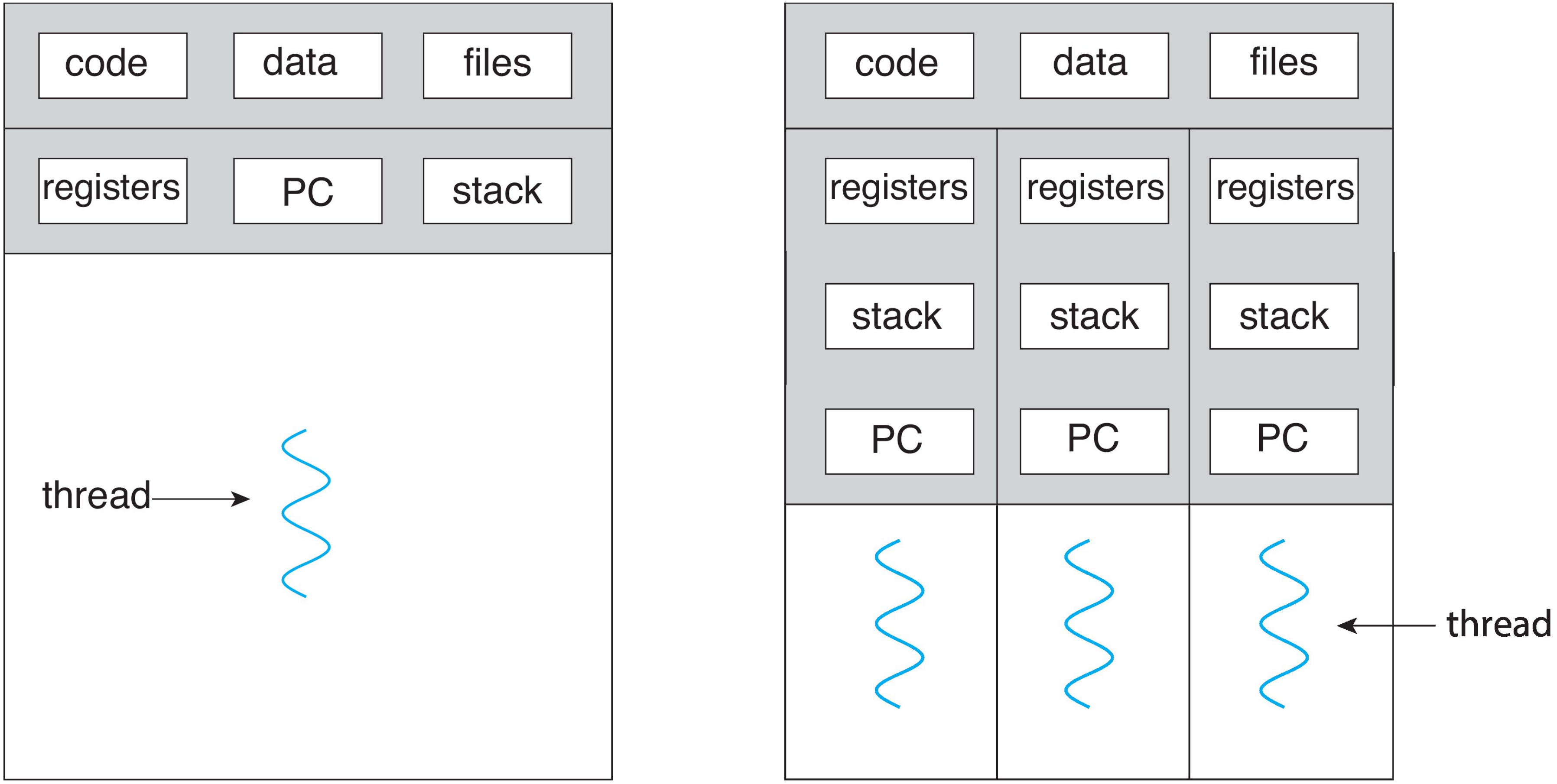
# Motivation

- Most modern applications are multithreaded
- Multiple tasks with the application can be implemented by threads
  - Update display
  - Fetch data
  - Spell checking



- Process creation is heavy-weight while thread creation is light-weight
- Kernels are generally multithreaded

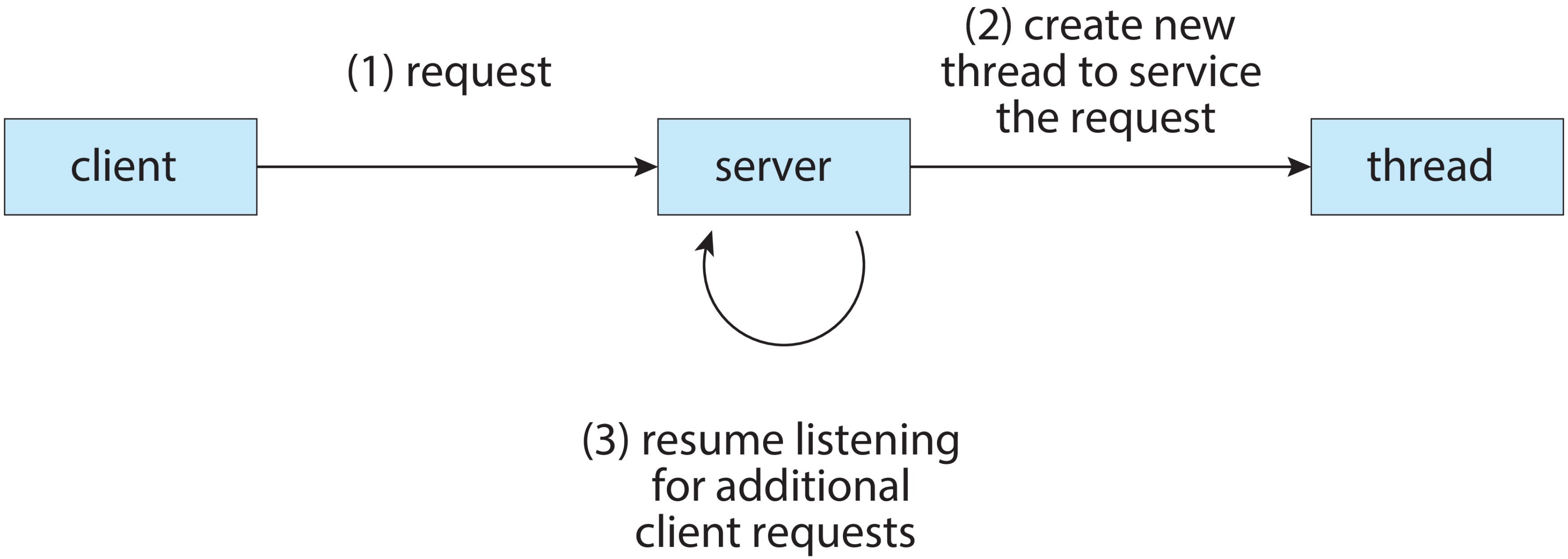
# Single and Multithreaded Processes



single-threaded process

multithreaded process

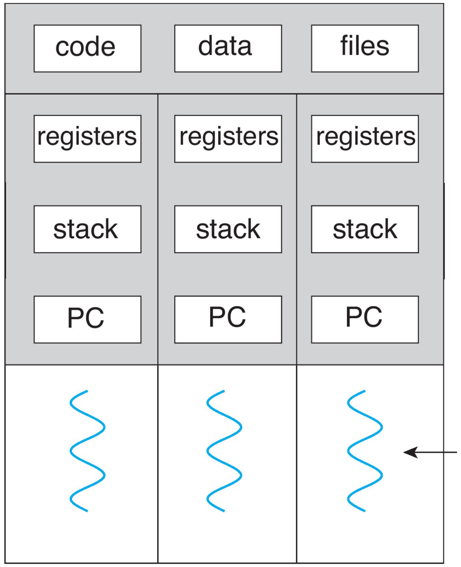
# Multithreaded Server Architecture


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

# Benefits

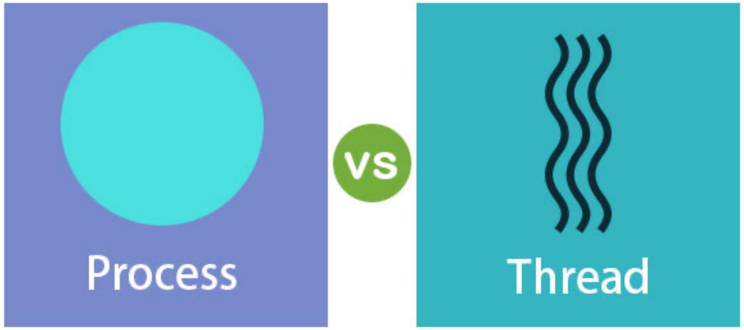
- **Responsiveness**
  - Allow continued execution if part of process is blocked
  - Especially important for user interfaces

- **Resource Sharing**
  - Threads **share** resources of process
  - **Easier** than shared memory or message passing.

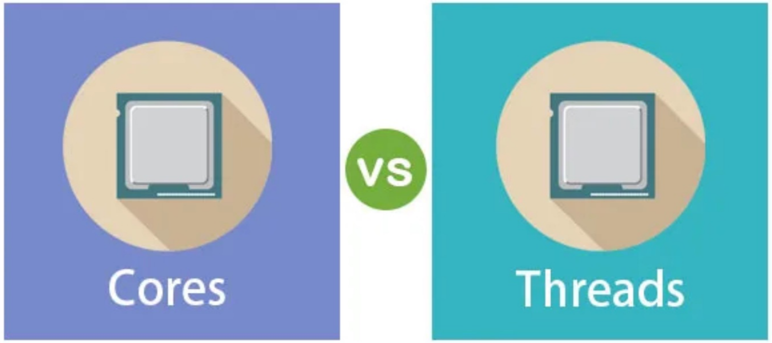


# Benefits (cont.)

- **Economy**
  - Cheaper than process creation
  - Thread switching lower overhead than context switching.

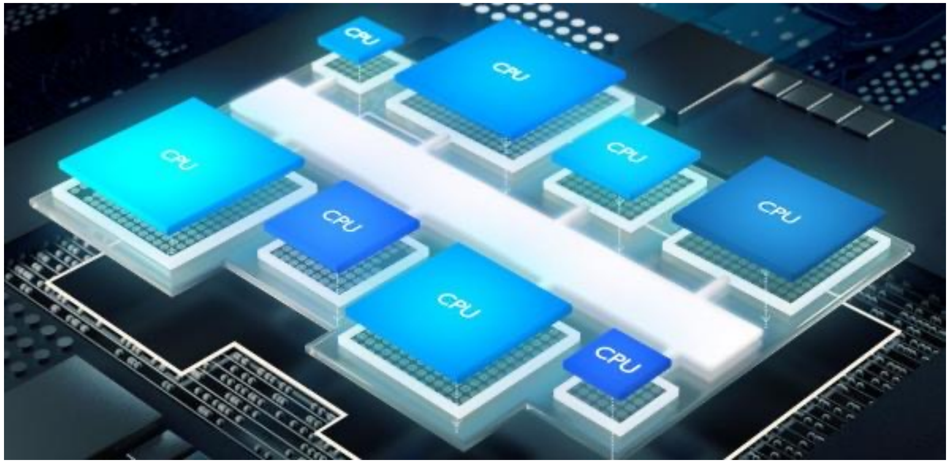


- **Scalability**
  - Process can take advantage of multicore architectures.



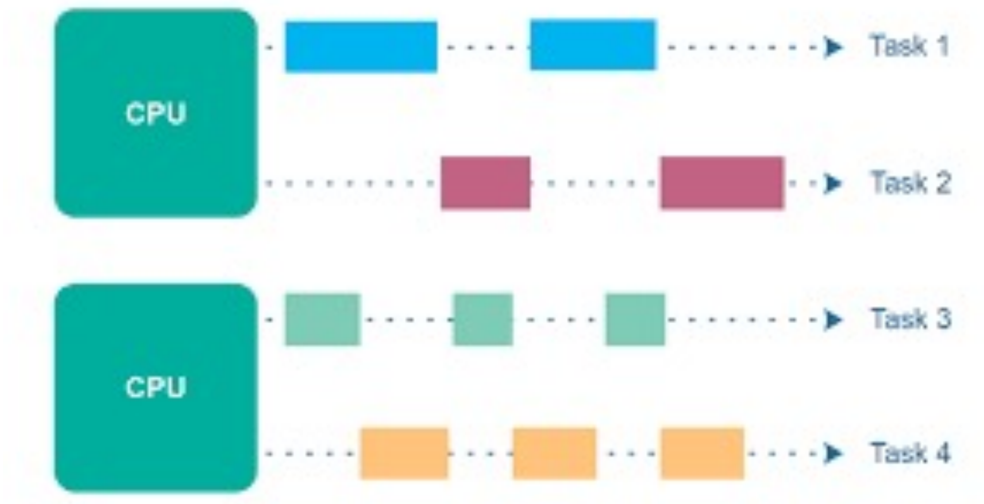
# Multicore Programming

- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - Dividing activities
  - Balance: ensuring that the tasks perform equal work of equal value.
  - Data splitting
  - Data dependency
  - Testing and debugging



# Multicore Programming (cont.)

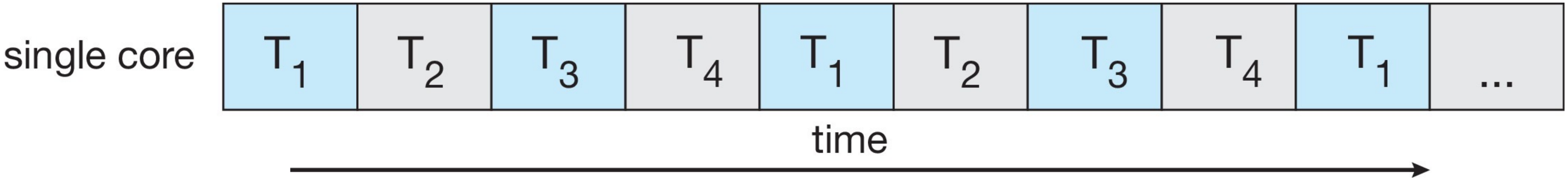
- **Parallelism** implies performing more than one task **simultaneously**



- **Concurrency** supports more than one task **making progress**
  - Single processor or core, scheduler providing concurrency

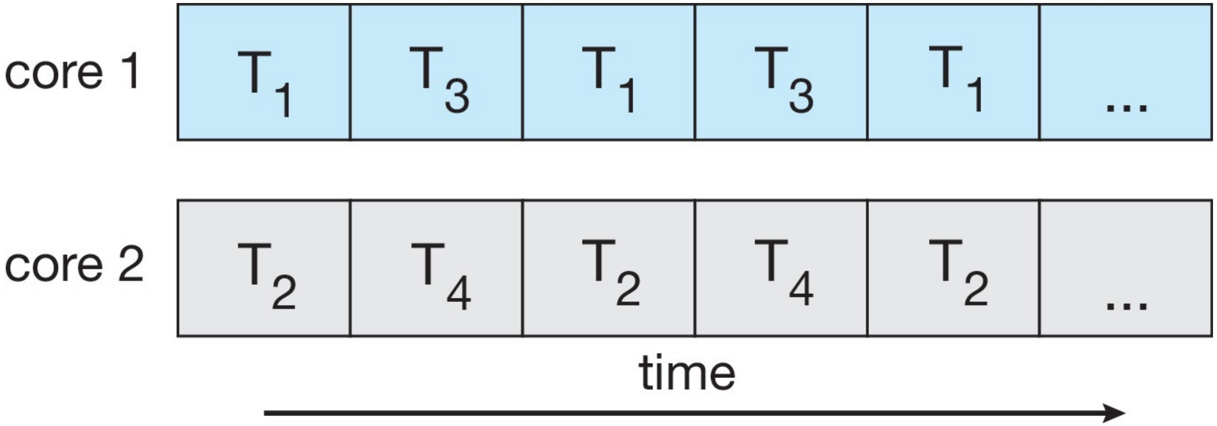
# Concurrency vs. Parallelism

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



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| single core | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_1$ | ... |

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

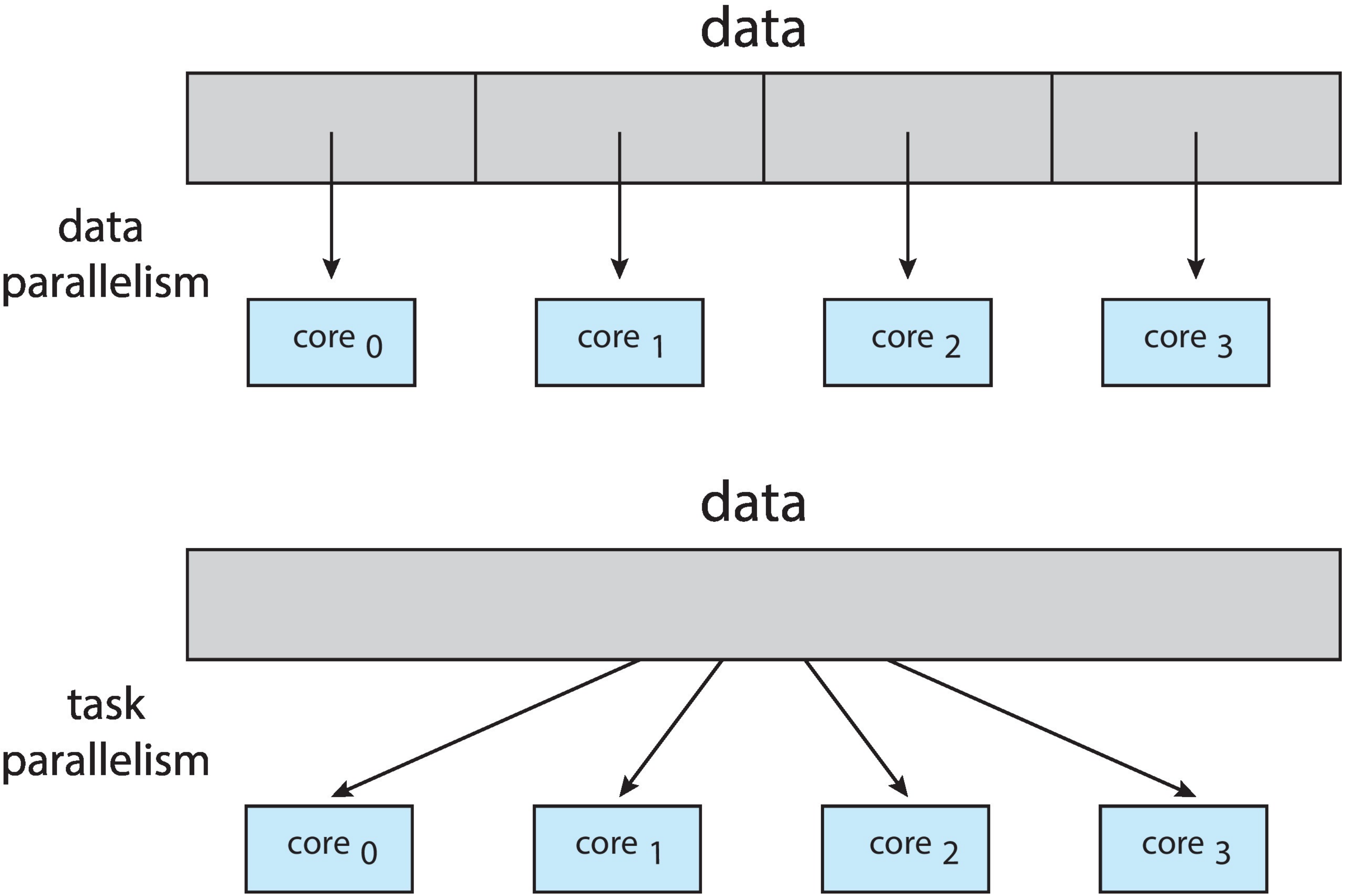


| | | | | | | |
|---|---|---|---|---|---|---|
| core 1 | $T_1$ | $T_3$ | $T_1$ | $T_3$ | $T_1$ | ... |
| core 2 | $T_2$ | $T_4$ | $T_2$ | $T_4$ | $T_2$ | ... |

# Multicore Programming-Types of Parallelism

- **Data parallelism**
  - Distributes subsets of the same data across multiple cores, same operation on each
  - Example: summing the contents of an array of size N.
- **Task parallelism**
  - Distributing threads across cores, each thread performing unique operation
  - Example: Unique statistical operation on the array of elements.

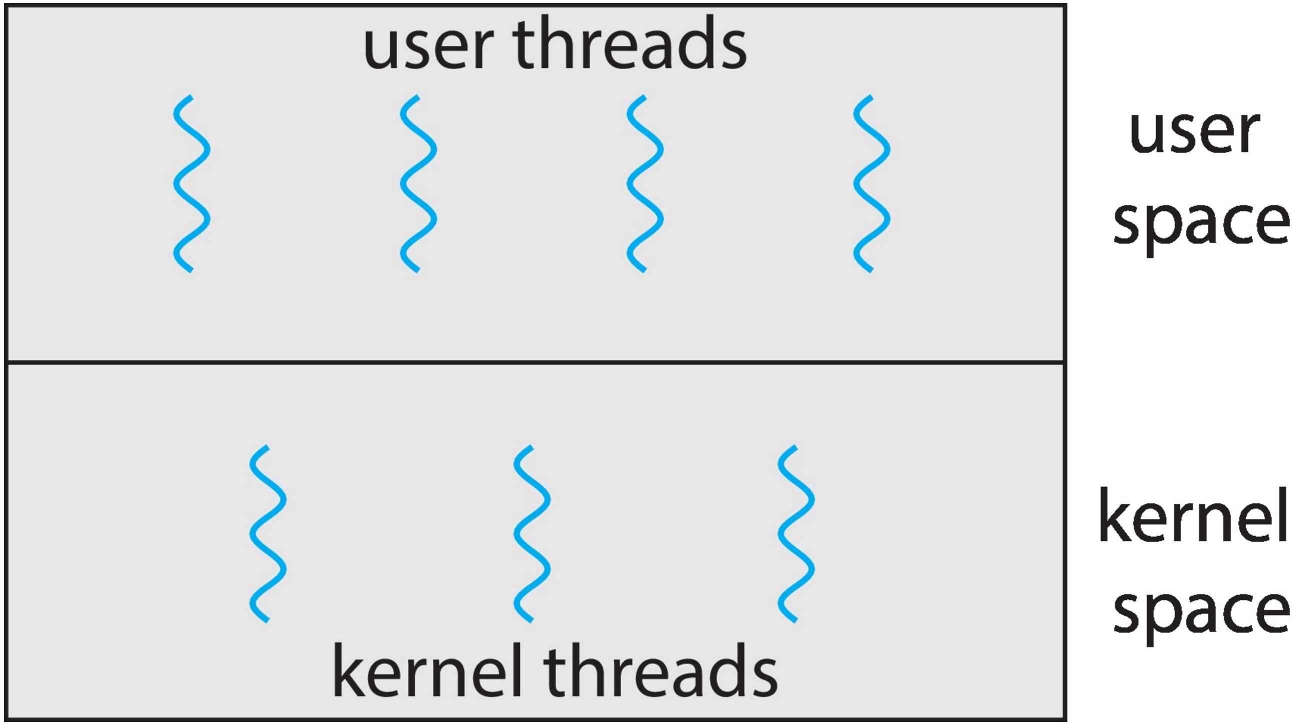
# Data and Task Parallelism



# 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, Linux, Mac OS X, iOS, Android

# User and Kernel Threads



Additional review: https://www.geeksforgeeks.org/difference-between-user-level-thread-and-kernel-level-thread/

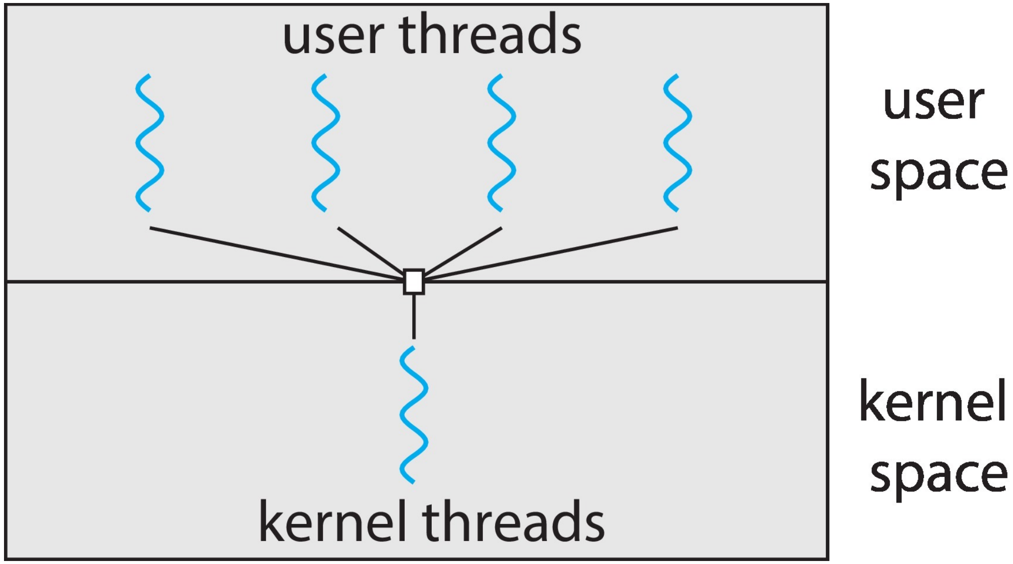
# Multithreading Models

- How to map user threads to kernel threads?
- Many-to-One
- One-to-One
- Many-to-Many

Additional review:
https://stackoverflow.com/questions/14791278/threads-why-must-all-user-threads-be-mapped-to-a-kernel-thread

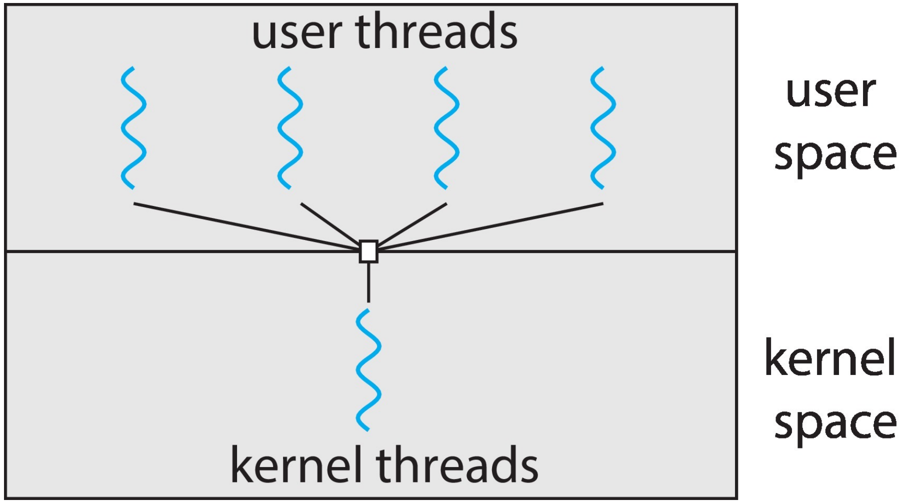
# Many-to-One

- Many user-level threads mapped to single kernel thread
- Thread management is done by the thread library in user space
  - So it is efficient



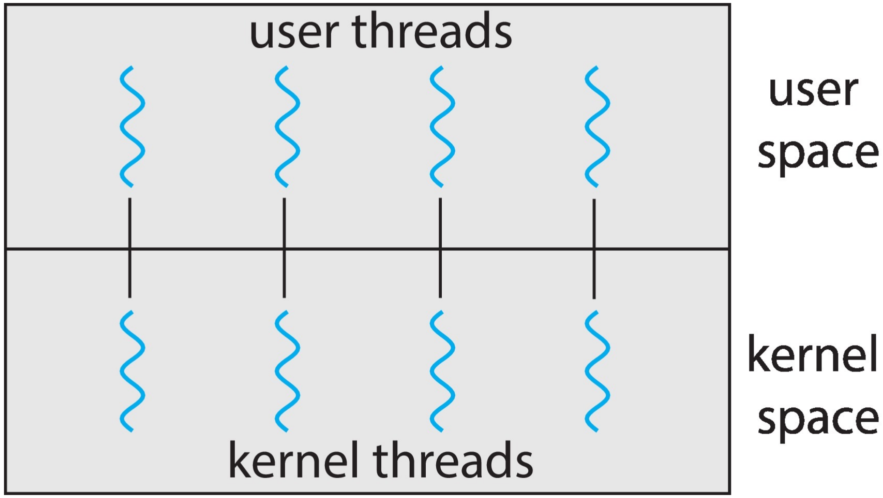
# Many-to-One (cont.)

- 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



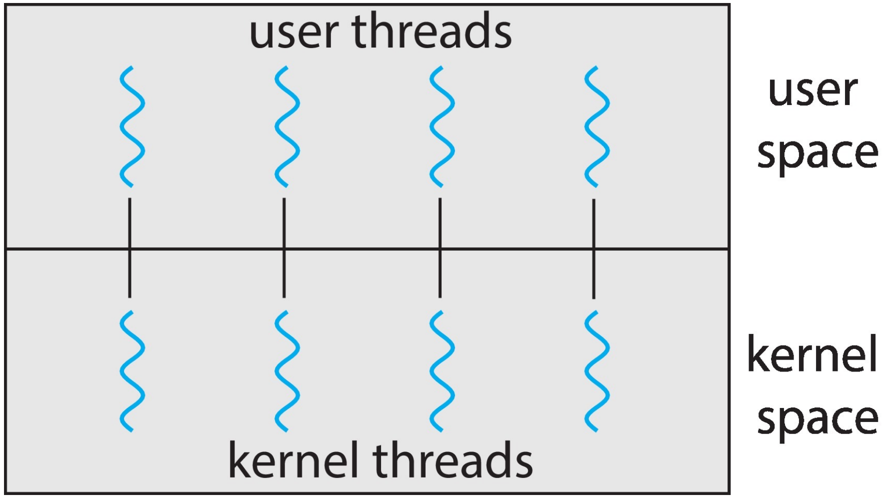
# One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread



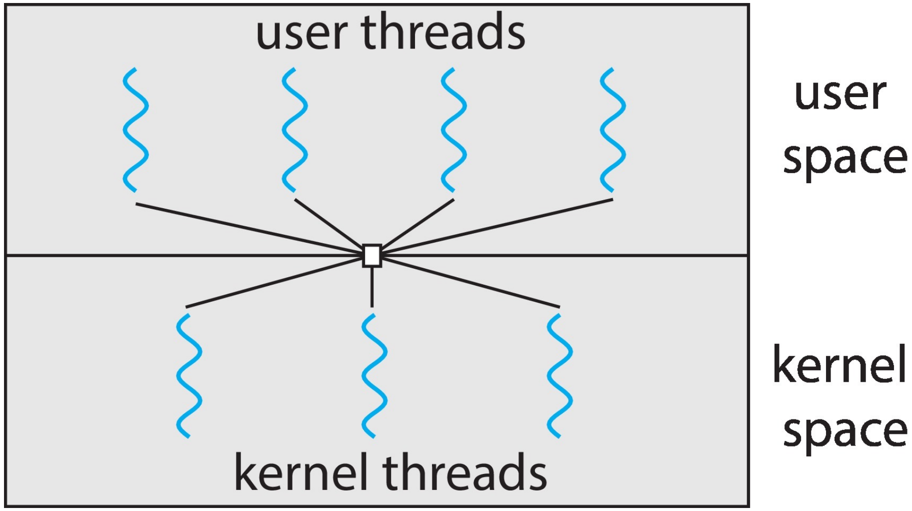
# One-to-One (cont.)

- **More concurrency than many-to-one**
- Number of threads per process may be restricted due to overhead
- Examples
  - Windows
  - Linux



# Many-to-Many Model

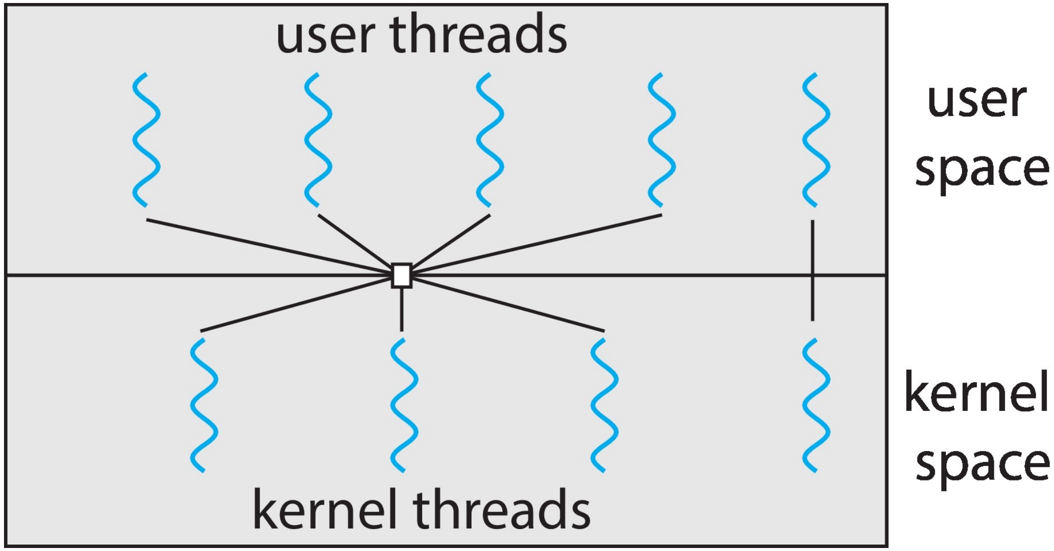
- Many user level threads to be mapped to many kernel threads
- Operating system can create a sufficient number of kernel threads



- Examples
  - Windows  with the *ThreadFiber* package
  - Otherwise not very common

# Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread


user threads
user space
kernel space
kernel threads