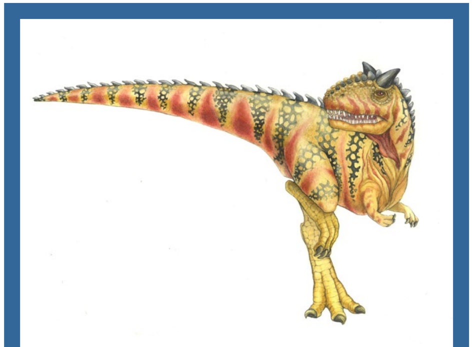
Chapter 4: Threads & Concurrency



Chapter 4: Threads

- Overview
- Multicore Programming
- Concurrency vs. Parallelism
- Amdahl's Law
- Multithreading Models

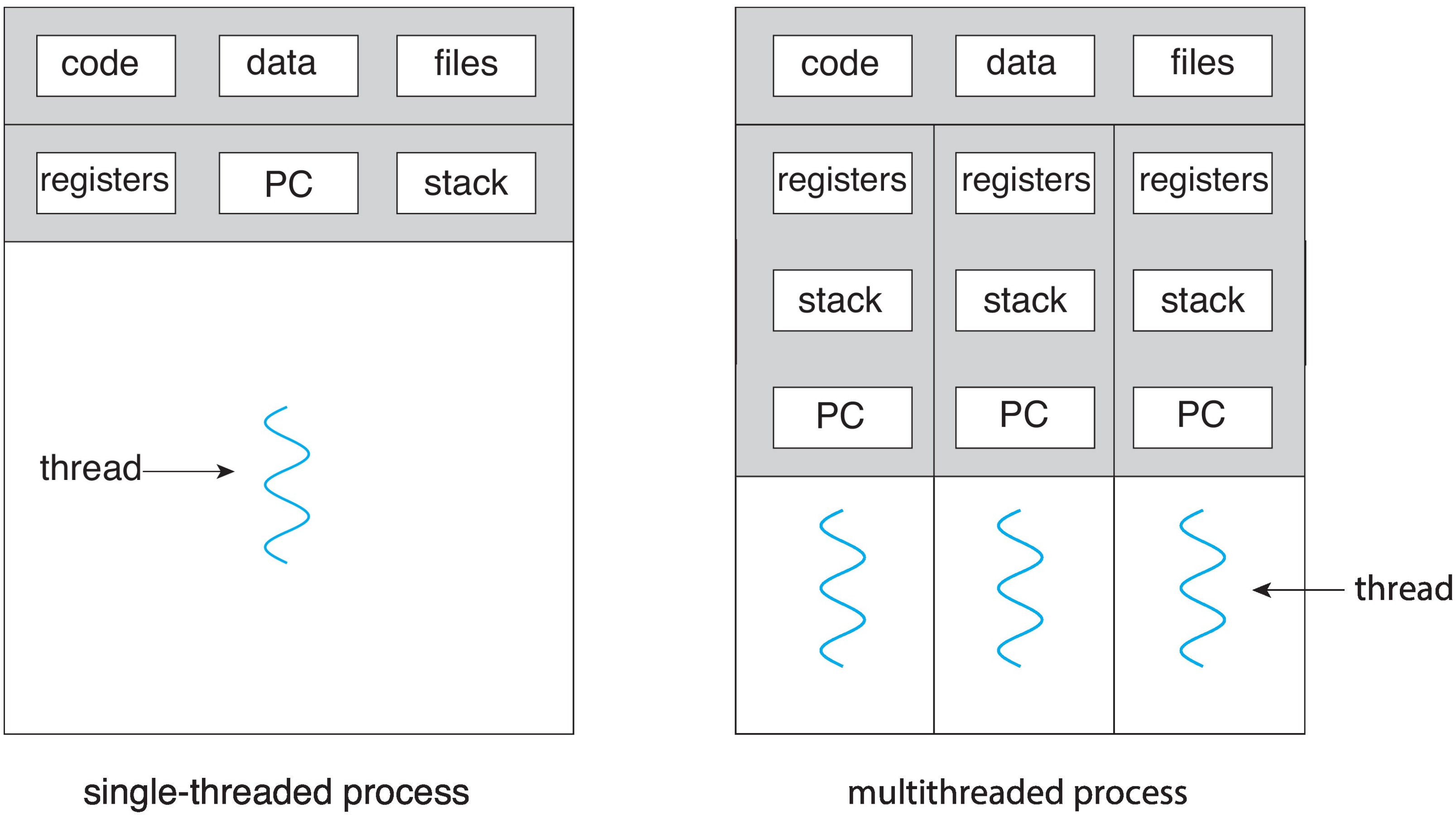
Objectives

- Identify the basic components of a thread, and contrast threads and processes
- Describe the benefits and challenges of designing multithreaded applications

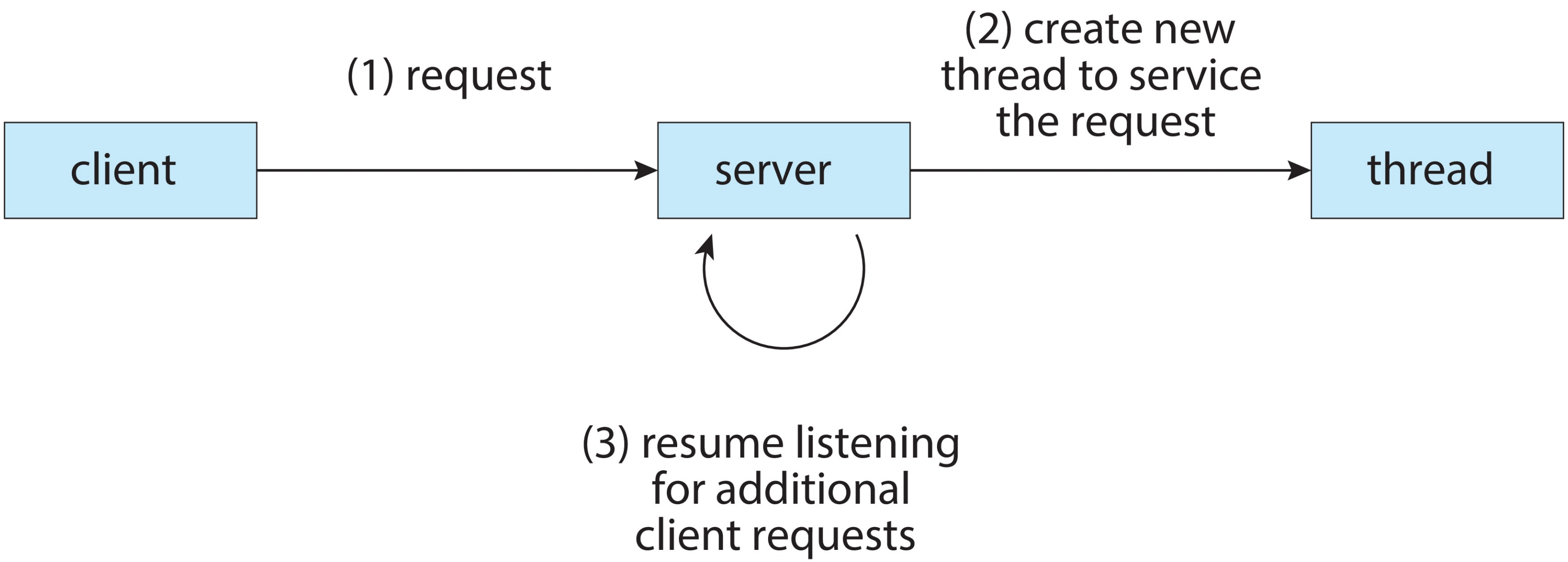
Motivation

- 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

Single and Multithreaded Processes



Multithreaded Server Architecture



Benefits

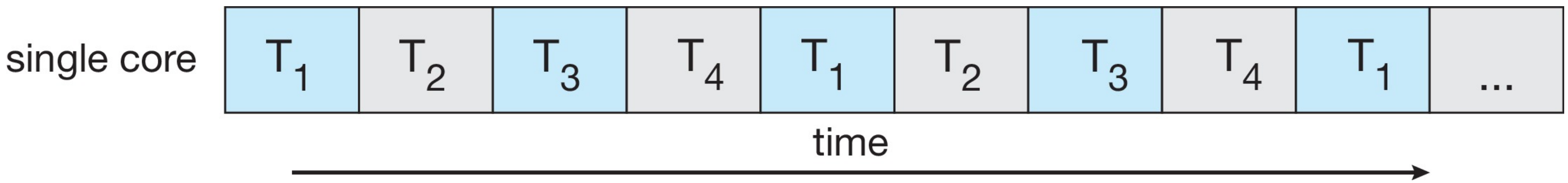
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multicore architectures

Multicore Programming

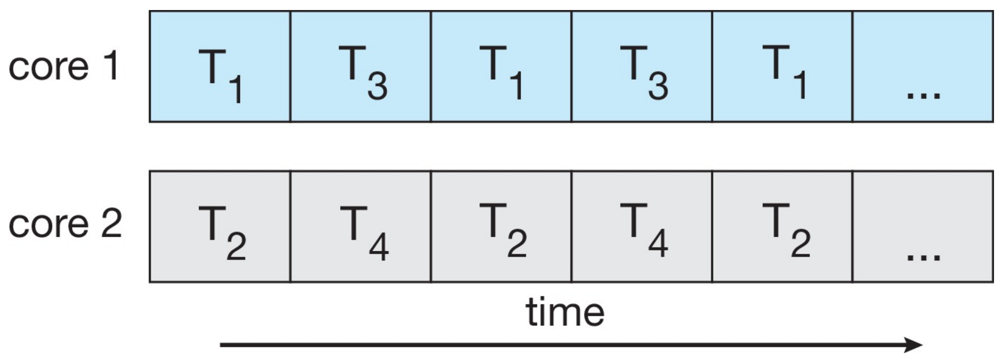
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- *Parallelism* implies a system can perform more than one task simultaneously
-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

Concurrency vs. Parallelism

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



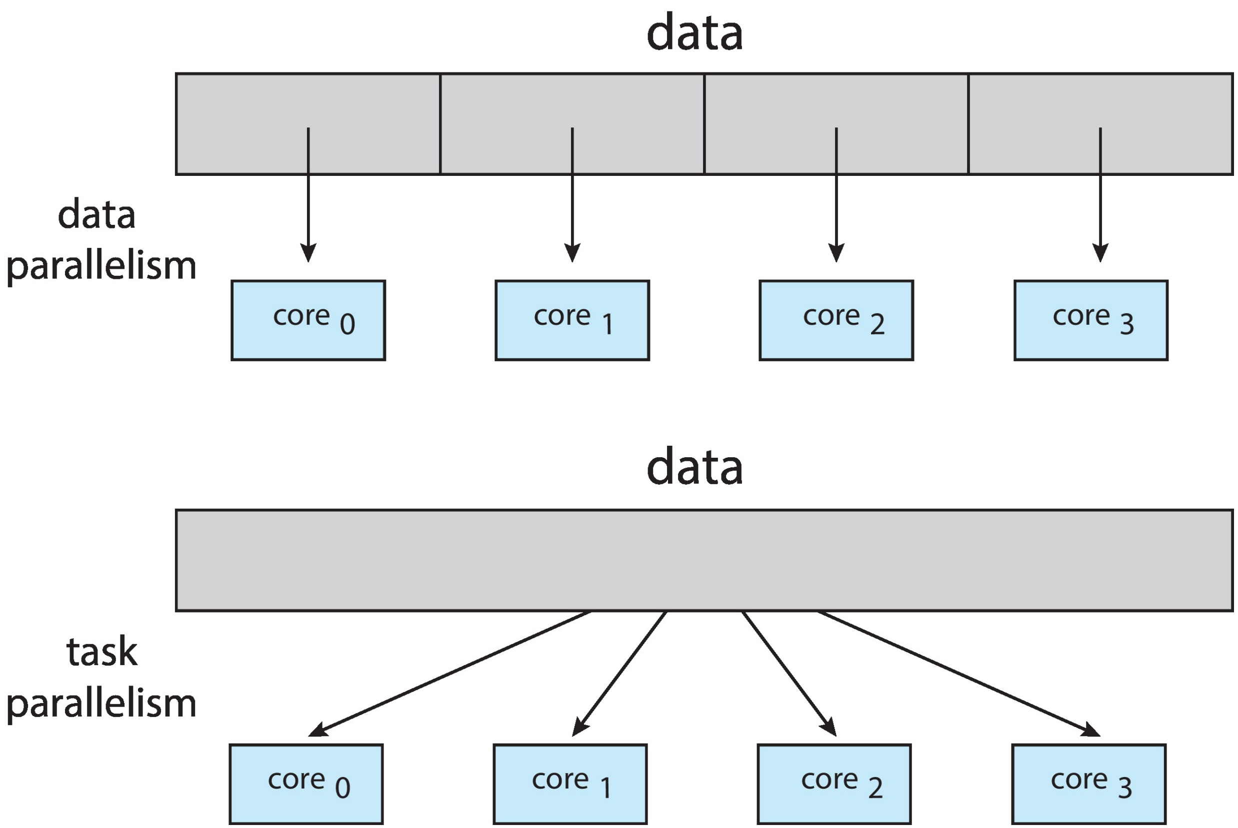
- **Parallelism on a multi-core system:**



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

Data and Task Parallelism



Amdahl's Law

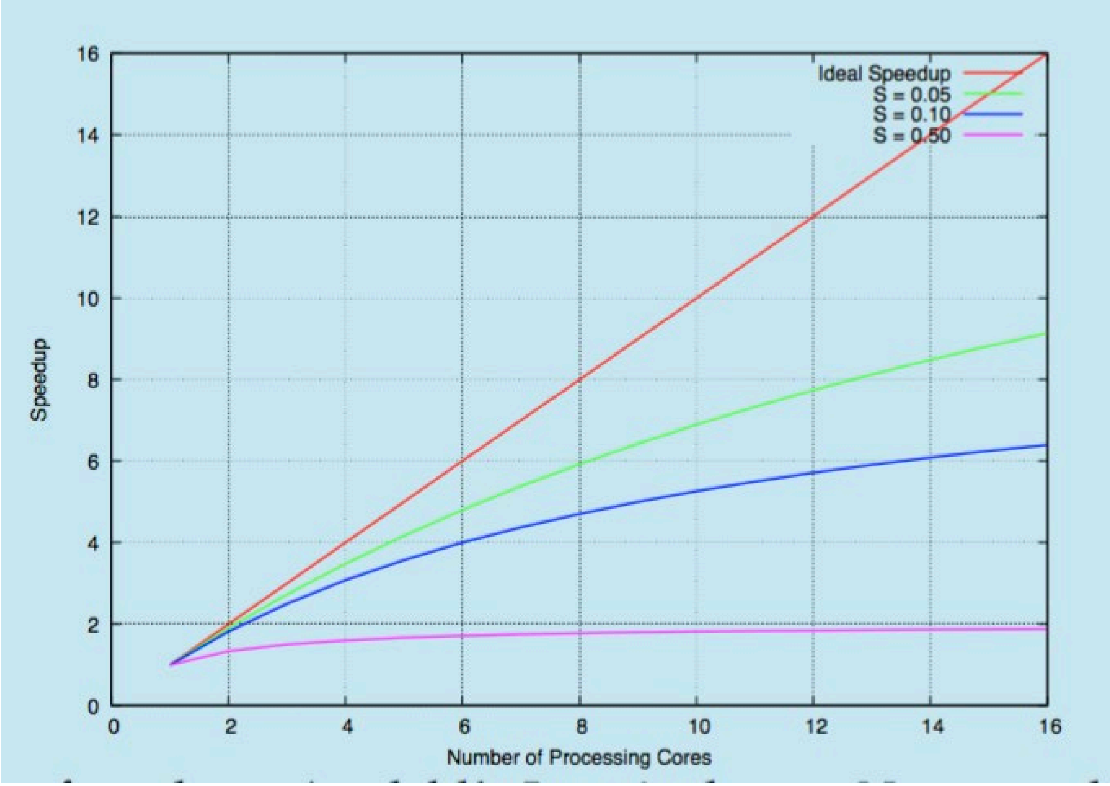
- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- *S* is serial portion
- *N* processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As *N* approaches infinity, speedup approaches 1 / *S*

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

Amdahl's Law



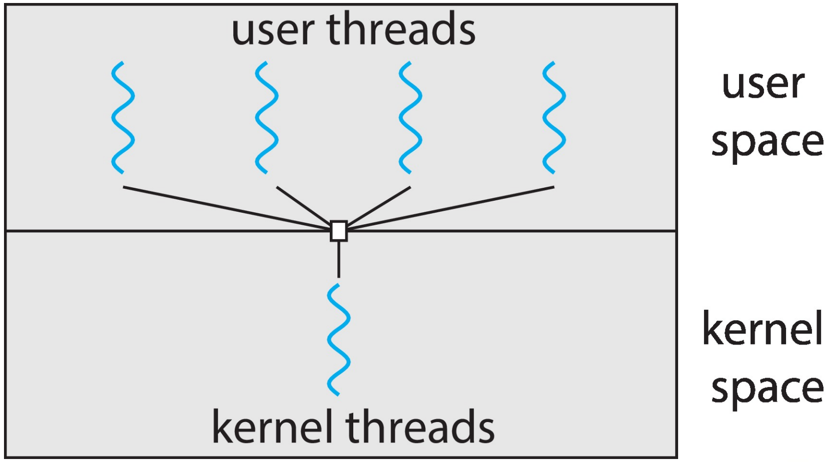
Ideal Speedup
S = 0.05
S = 0.10
S = 0.50
Speedup
Number of Processing Cores
0
2
4
6
8
10
12
14
16

Multithreading Models

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

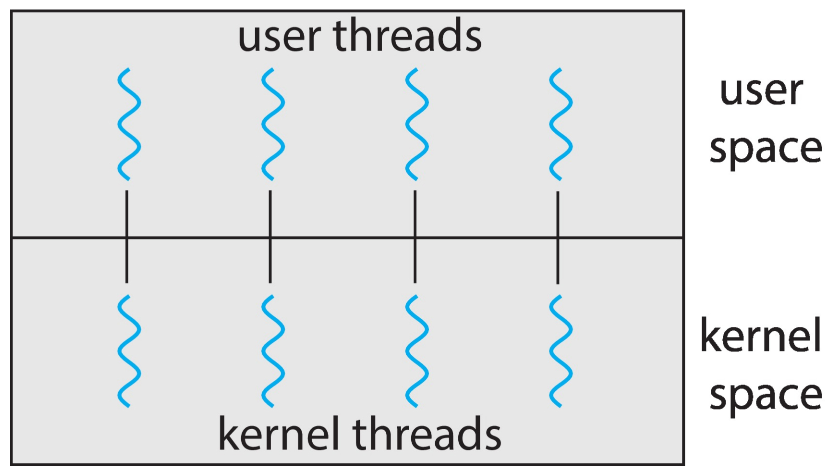
Many-to-One

- 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**



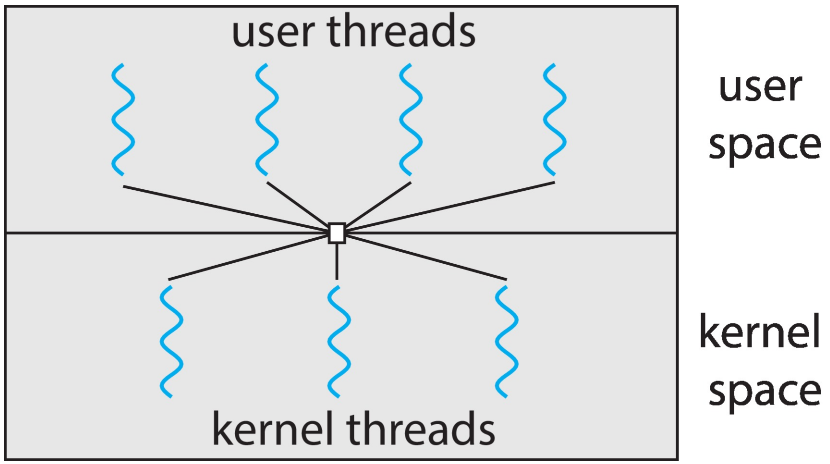
One-to-One

- 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



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
- Not very common



End of Chapter 4