CISC 3320 MW3
Threads and Multithread Model

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook

Outline

- Overview
- Multicore Programming
- Multithreading Models

- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples

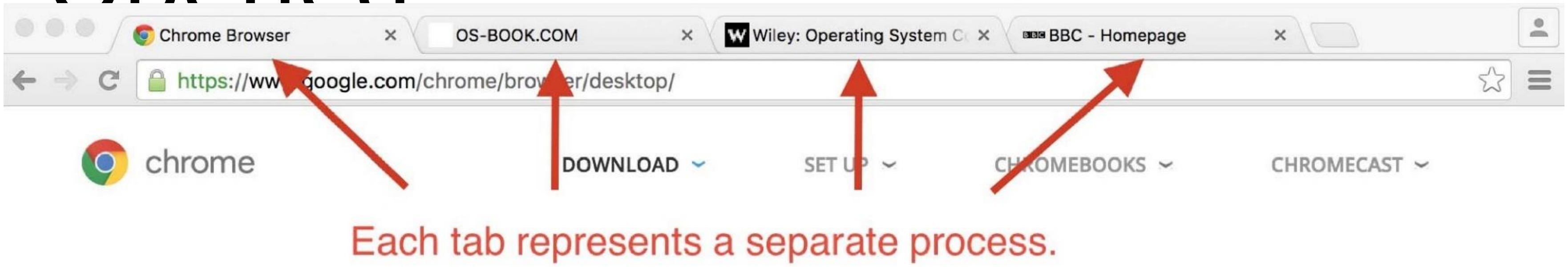
Recap: Multiprogramming and Multiprocess Architecture

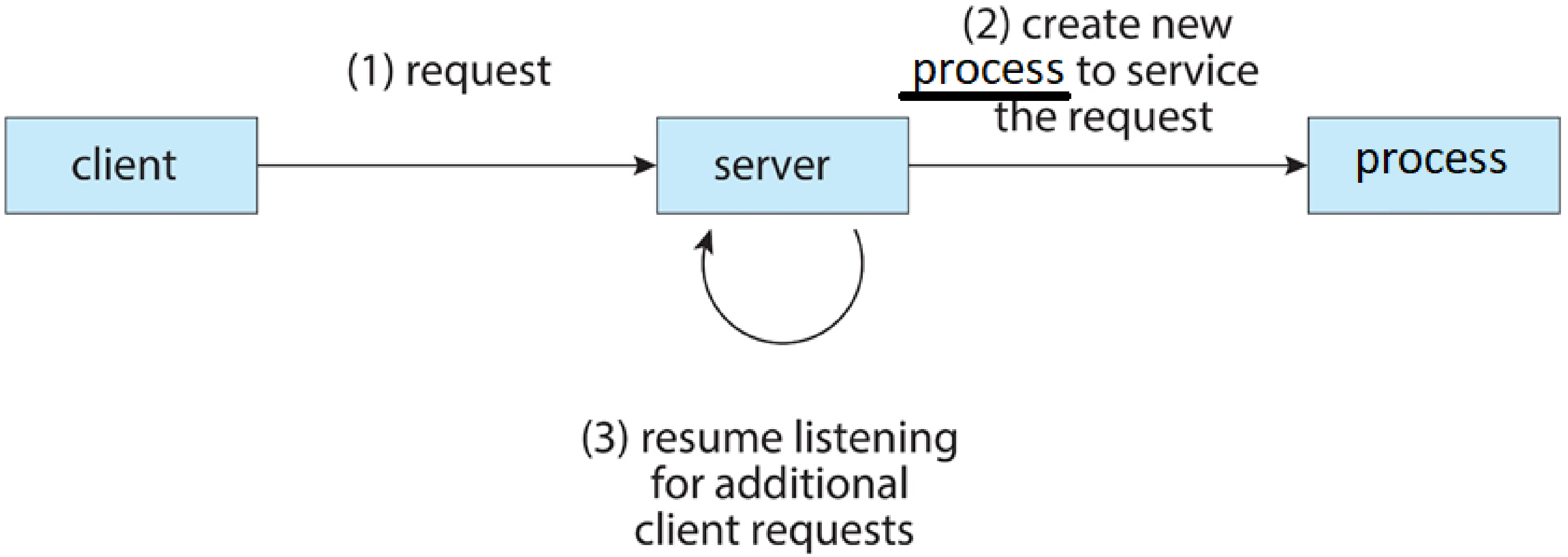
- OS loads multiple programs and execute them concurrently
    - Notation of process
- Independent processes
- Cooperating processes

Recap: Multiprocess Architecture

- Example applications
  - Chrome Web browser
  - The instructor’s Monte Carlo simulation application

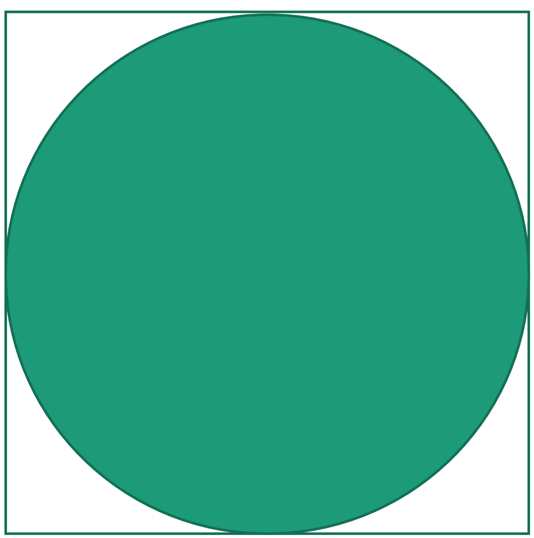
Recap: New Renderer Created for Each Website Opened





Recap: Computation Speed-up by Cooperating Processes

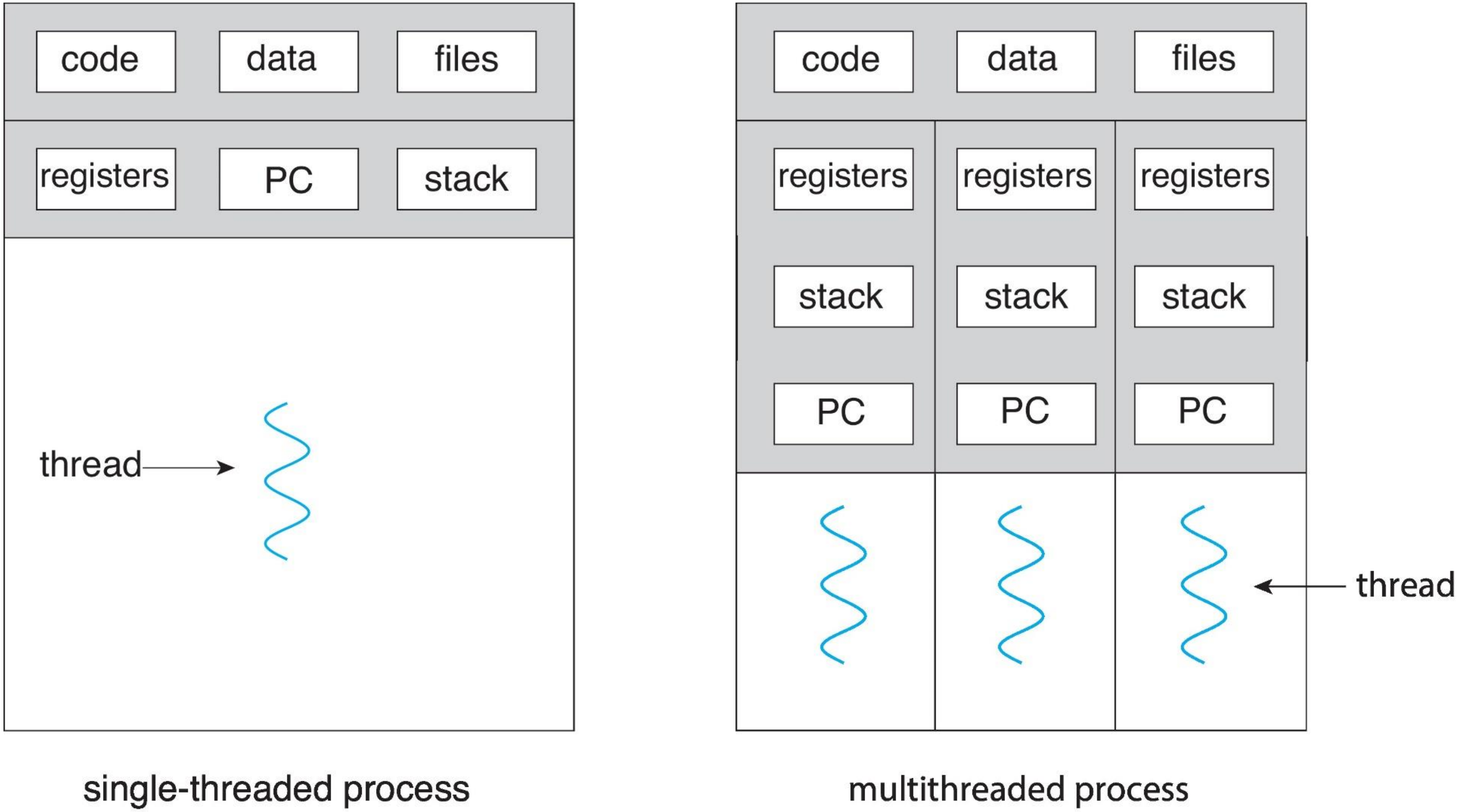
- Estimate π using a Monte Carlo simulation
  - What if a machine has multiple CPU cores? Can we take advantage of it?



Motivation for Threads

- Multiprocess architecture
  - Cost (or overhead)
    - Process creation and switching are heavy-weight
    - Project 2: quantifying process context switching cost
  - Communication
    - Inter-process communications (slow or complex)

Introducing Multithreaded Processes



Example: Observing Threads and Threads Switching

- We can write an application to run multiple functions concurrently, and each becomes thread
- Example: implement it in UNIX systems in user mode
  - Understand concept of context
  - Understand context switching
- We will examine this example closer again when we discuss CPU scheduling

Questions?

- Some motivations for threads
- Concepts of threads and comparison with process

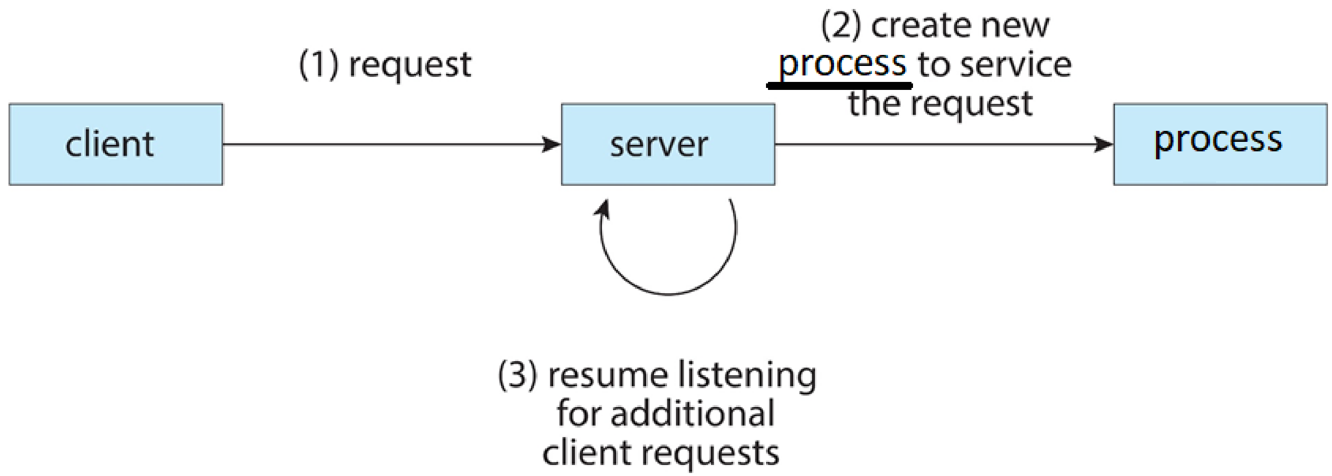
Benefits of Multithread Architecture

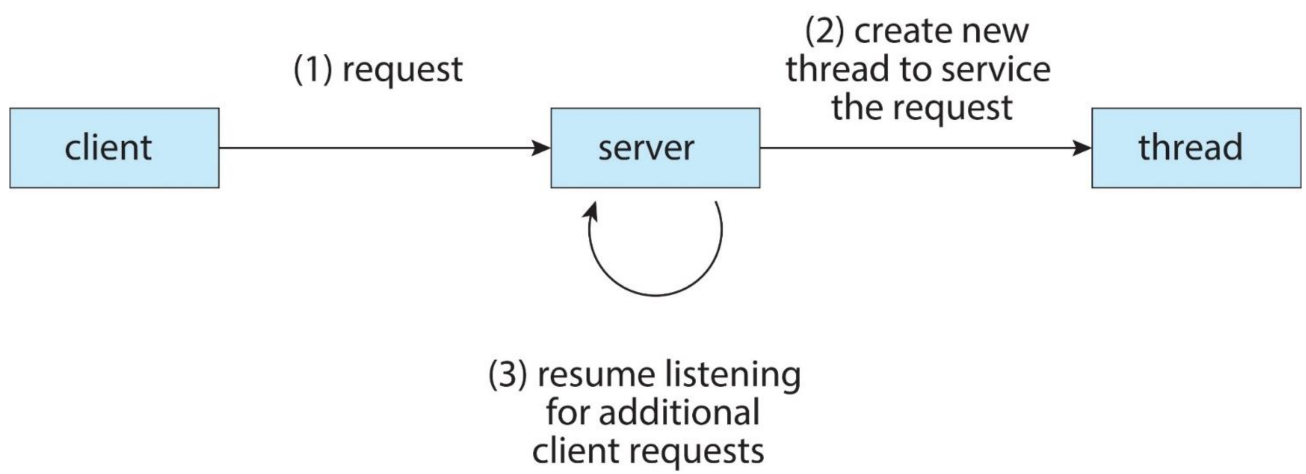
- Improving responsiveness
- Easing resource sharing
- Can be made more economic (less overhead)
- Can be more scalable (to multicore architecture)

Multiprocess and Multithread Architectures

- Benefits of threads when compared to multiprocess architecture

Examples of Multiprocess and Multithread Architectures





Multithread Architecture: Responsiveness

- May allow continued execution if part of process is blocked
  - Especially important for user interfaces
- Example: let’s look at event-driven programming for graphical user interface

Event-Driven Programming for Graphical User Interface

- The main body of the program is an event loop (in pseudo code)

```
do {
    e = getNextEvent()
    processEvent(e)
} while (e != EXIT_EVENT)
```

- This event loop typically implemented by the platform, and runs in a thread
- The events are in a queue called event queue
- Users write event handler routines (user's programs) to process events
  - processEvent in the above will invoke your event handler routines
  - Event thus drives user's programs

Event-Loop Runs in a Thread

- Event loop

```
do {
    e = getNextEvent()
    processEvent(e)
} while (e != EXIT_EVENT)
```

- What happens if the processEvent method takes a long time to complete?
  - Also consider the events are in a queue called event queue (capacity?)

Example Application: π Estimator as GUI Application

- Two versions
  - Without thread and with thread (for π estimation)
- Observe how responsive the application becomes when we increase the random points generated.

Recap: Multithread Architecture: Responsiveness

- May allow continued execution if part of process is blocked
  - Especially important for user interfaces
- Example: let’s look at even-programming for graphical user interface

- Event loop

```
do {
    e = getNextEvent()
    processEvent(e)
} while (e != EXIT_EVENT)
```

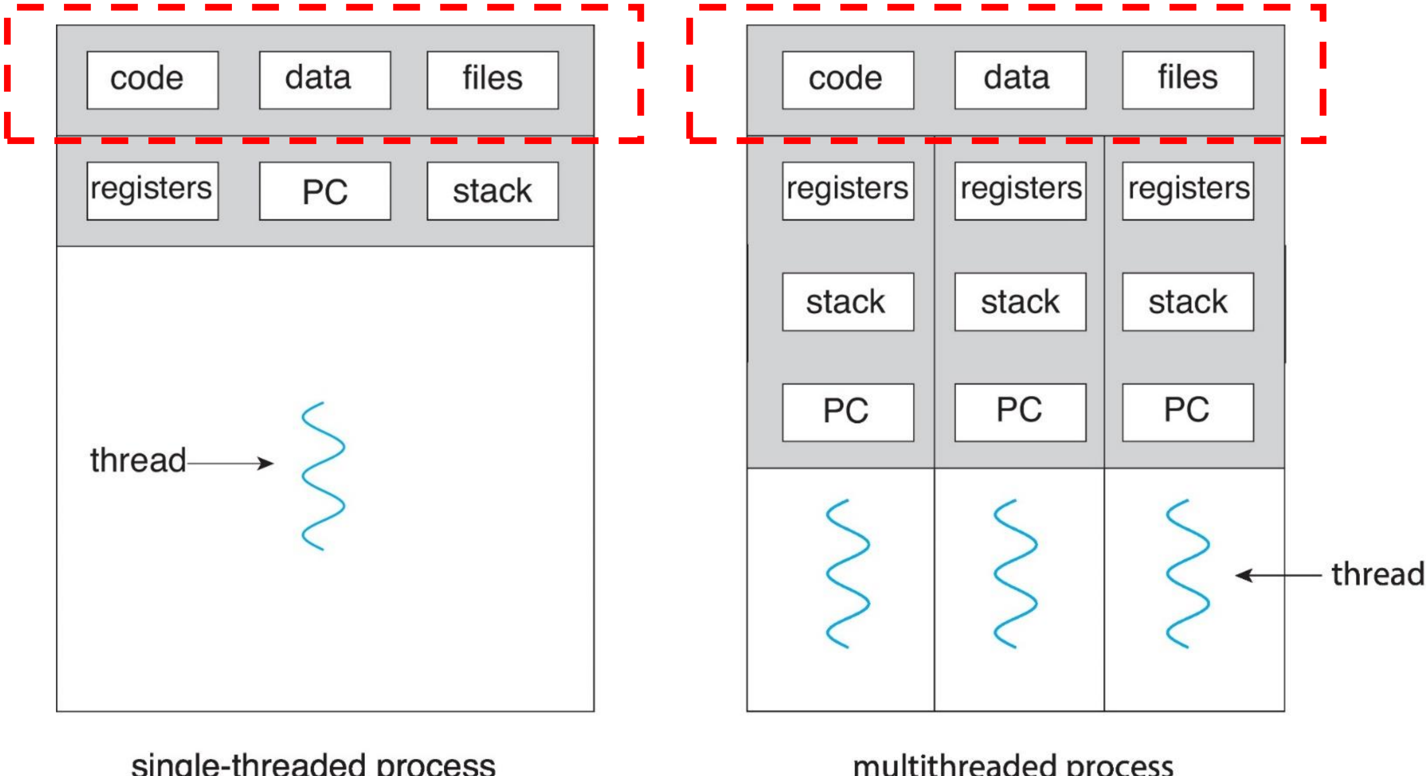
Expected to be completed very quickly (a fraction of a second)

Questions?

- Improve application responsiveness via a multithreaded architecture

Benefits of Threads: Resource Sharing

- Threads share resources of process, easier than shared memory or message passing



Benefits of Threads: Scalability

- A process can take advantage of multicore architectures using multiple threads

Example: The Multiprocess or Multithread π Estimator

- Estimate π using a Monte Carlo simulation
  - What if a machine has multiple CPU cores? Can we take advantage of it?
    - Case 1: we try 80000000 trials on 1 CPU core (1 worker process or 1 worker thread)
    - Case 2: we try 80000000/2 trials for each process on 2 CPU core (2 worker processes or 2 worker threads)
    - Case 3: we try 80000000/4 trials for each process on 4 CPU core (4 worker processes or 2 worker threads)
  - Worker processes need to send master process the results (in this example, via a named pipe)
  - Worker threads write results to the heap or the global variables

Example: The Multiprocess or Multithread π Estimator

- Your observations?
  - Which methods of sharing data is more convenient for programmers?
  - Do you observe computation speed up?
  - Based on our experiment, are threads always light-weight?

Questions?

- Multiprocess architecture vs. multithread architecture
  - Resource sharing?
  - Scalability?

Benefits of Threads: Economy

- Threads creation is generally cheaper than process creation
- Thread context switching is generally of lower overhead than process context switching
- Project 2: quantifying cost of context switch

Questions?

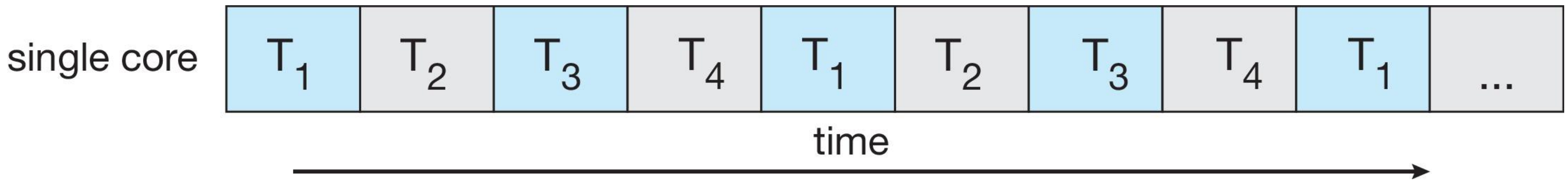
- Benefits of Multithread Architecture
- Responsiveness
  - may allow continued execution if part of process is blocked, especially important for user interfaces
- Resource Sharing
  - threads share resources of process, easier than shared memory or message passing
- Economy
  - cheaper than process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

Multicore Programming

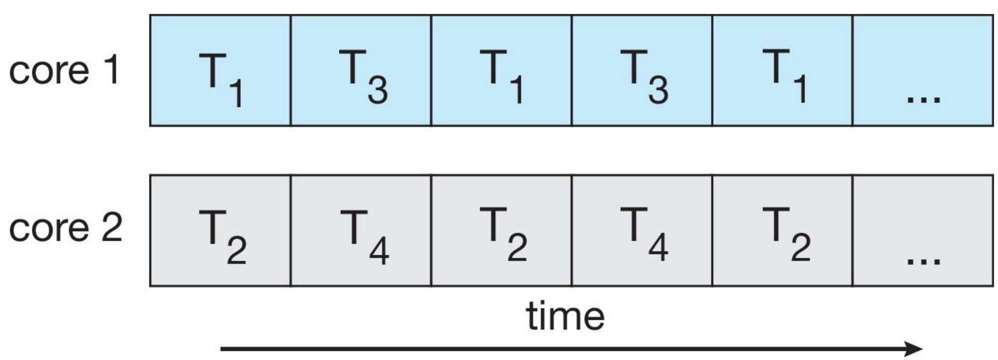
- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
    - Dividing activities
    - Balance
    - Data splitting
    - Data dependency
    - Testing and debugging
- *Parallelism* implies a system can perform more than one task simultaneously
- *Concurrency* supports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

Concurrency vs. Parallelism

- Concurrent execution on single-core system



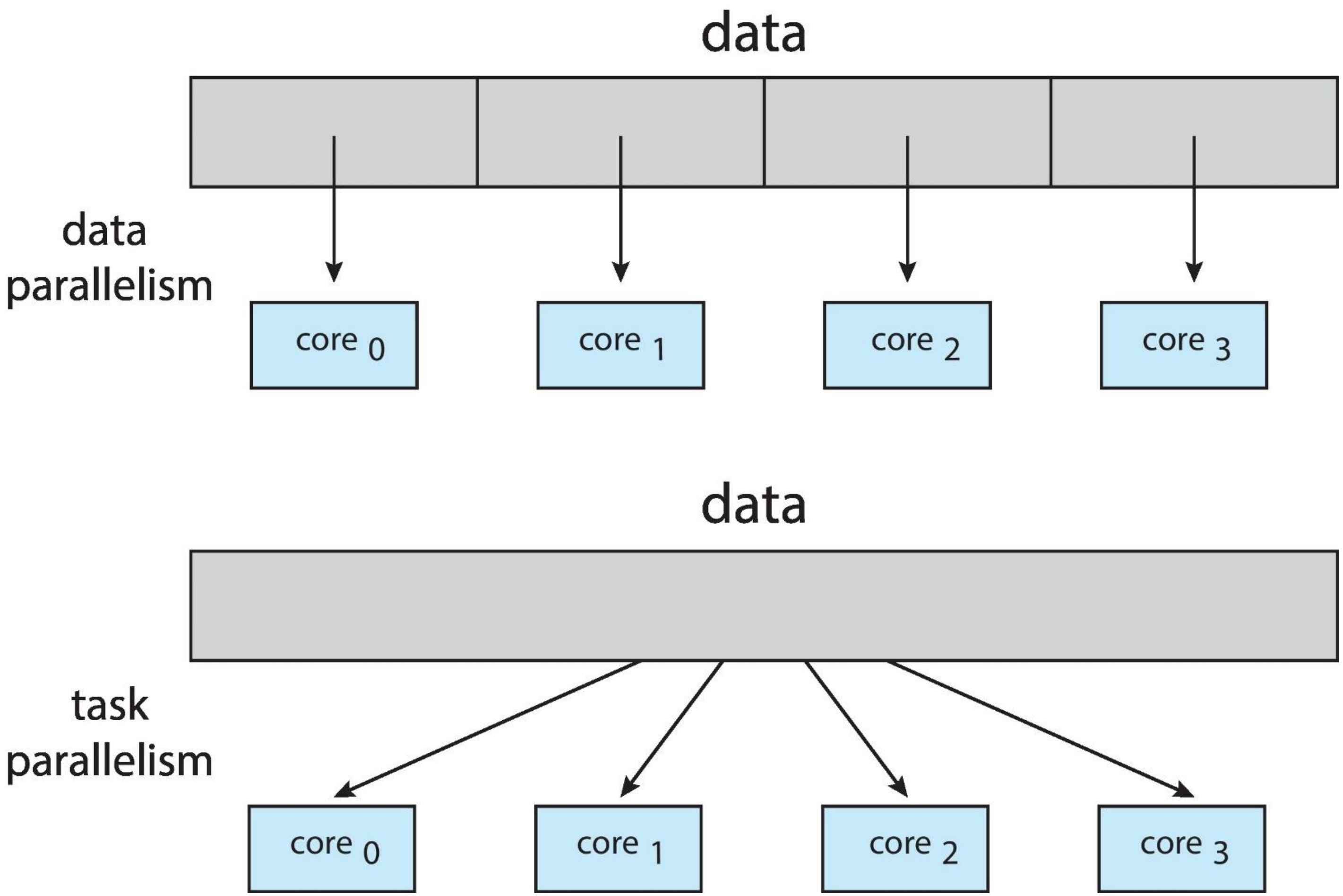
- Parallelism on a multi-core system



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



Performance Gain via Parallelism

- How much do we gain in this example?
- Is this example data parallelism or task parallelism?
- If it is data parallelism, can you revise this example to exhibit task parallelism or vice versus?
- How much can we gain from parallelism?

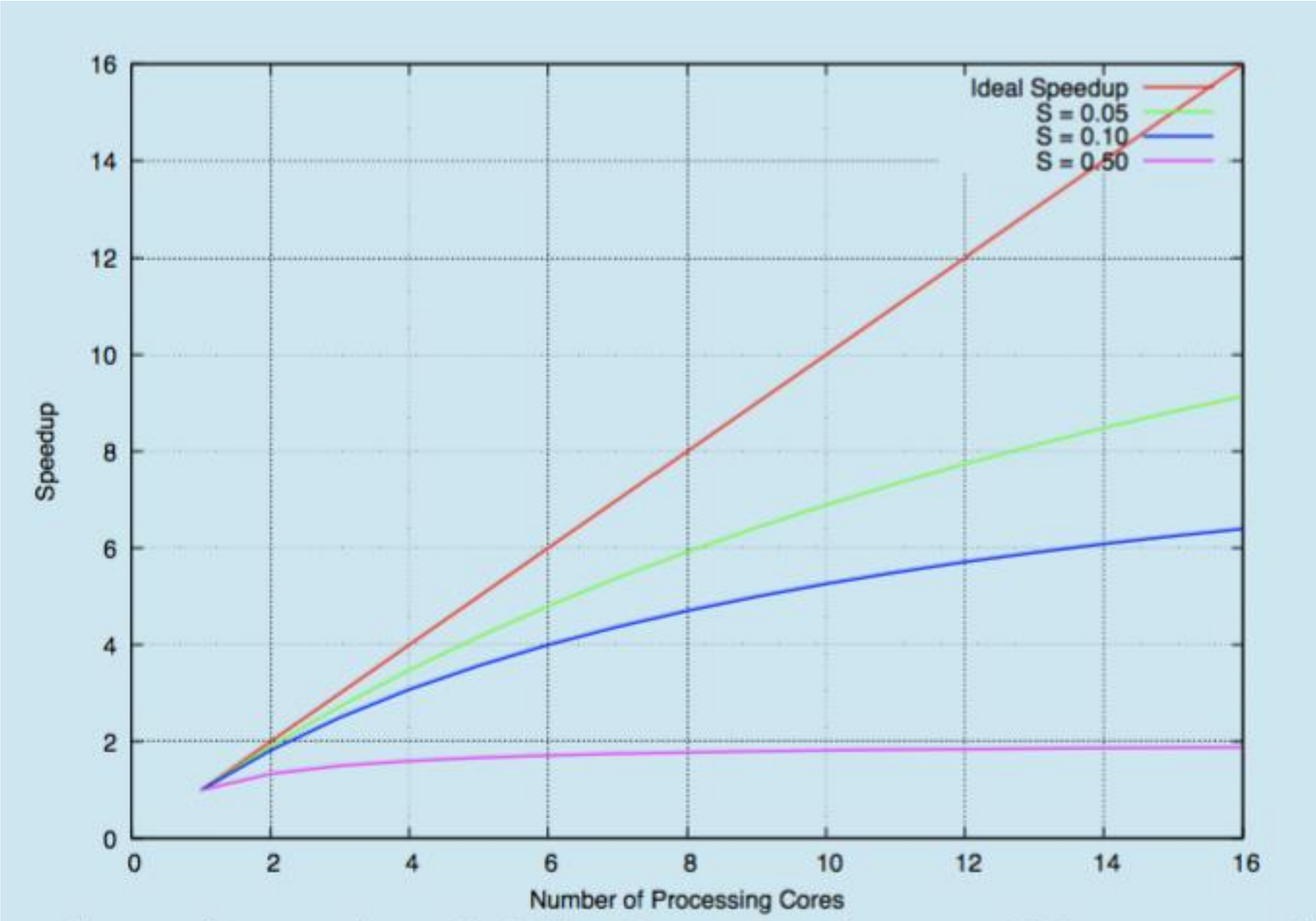
Amdahl's Law

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

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

Amdahl's Law: Example

- That is, if application is 75% parallel / 25% serial
  - 1/(0.25 + 0.75/2) = 1.6
  - moving from 1 to 2 cores results in speedup of 1.6 times
- As N approaches infinity, speedup approaches 1 / S
  - $\lim_{N \to \infty} \frac{1}{S + \frac{1-S}{N}} = \frac{1}{S}$
- Serial portion of an application has disproportionate effect on performance gained by adding additional cores


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

Questions?

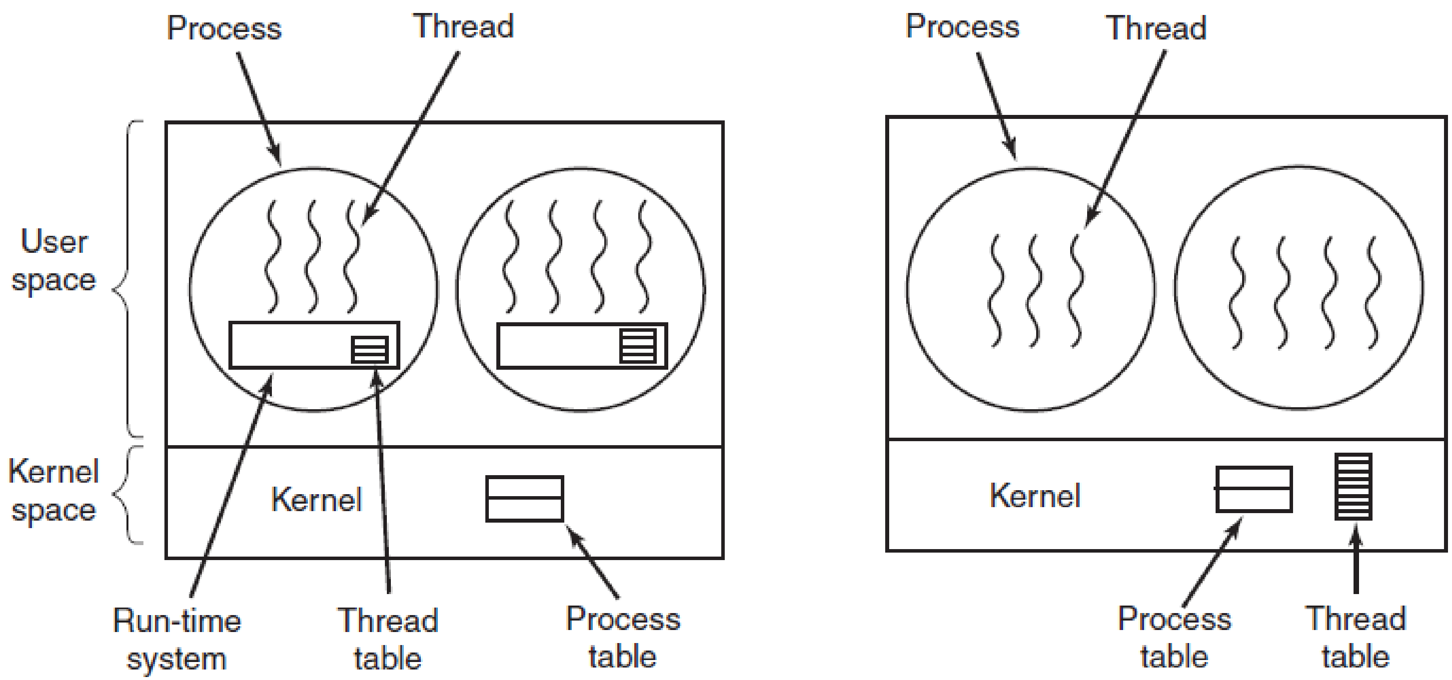
- Amdahl’s Law
- Does the law take into account contemporary multicore systems?
- Recall: the two versions of the π estimator

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

Threads In User or Kernel Space

- In the user space, or in the kernel

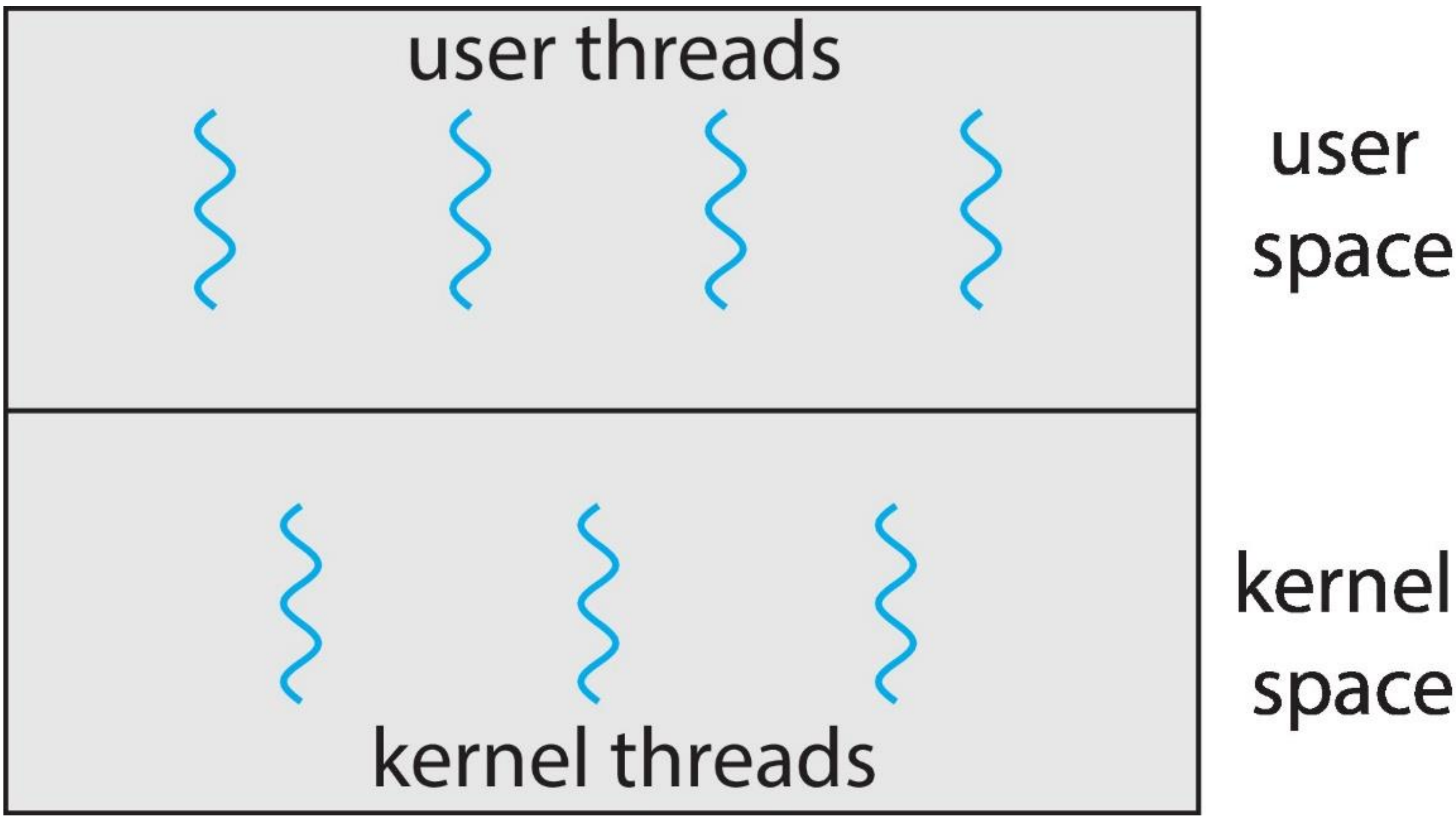


User- and kernel-level threads [Figure 2-16 in Tanenbaum & Bos, 2014]

Efficiency and Concurrency

- Kernel threads are more expensive to create
  - Can support multiple processors
- User threads can be blocked by the process
  - Less concurrency, in particular, on multiprocessor/multicore systems
- Recall: the user mode two thread program discussed at the beginning

User and Kernel Threads



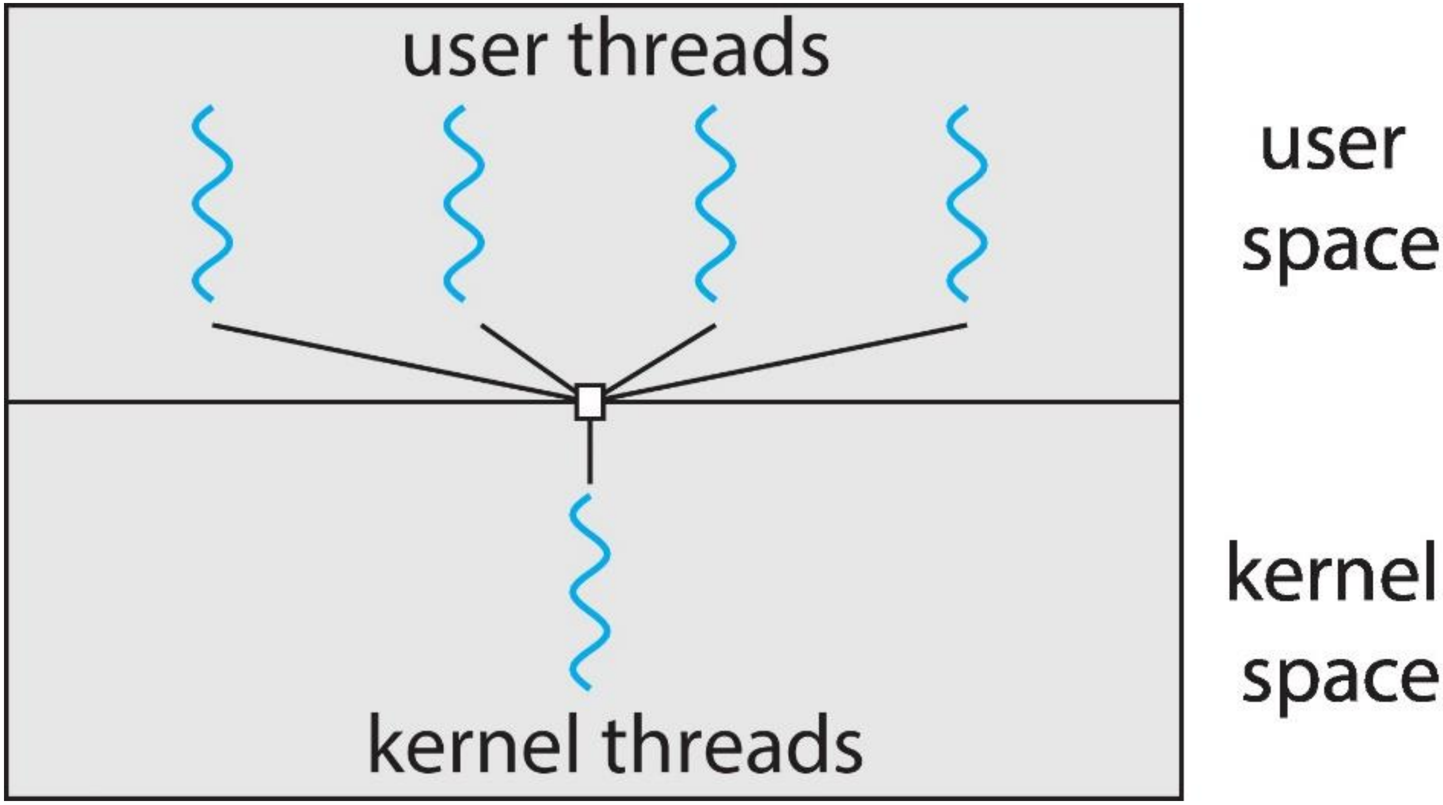
Multithreading Models

- 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 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**

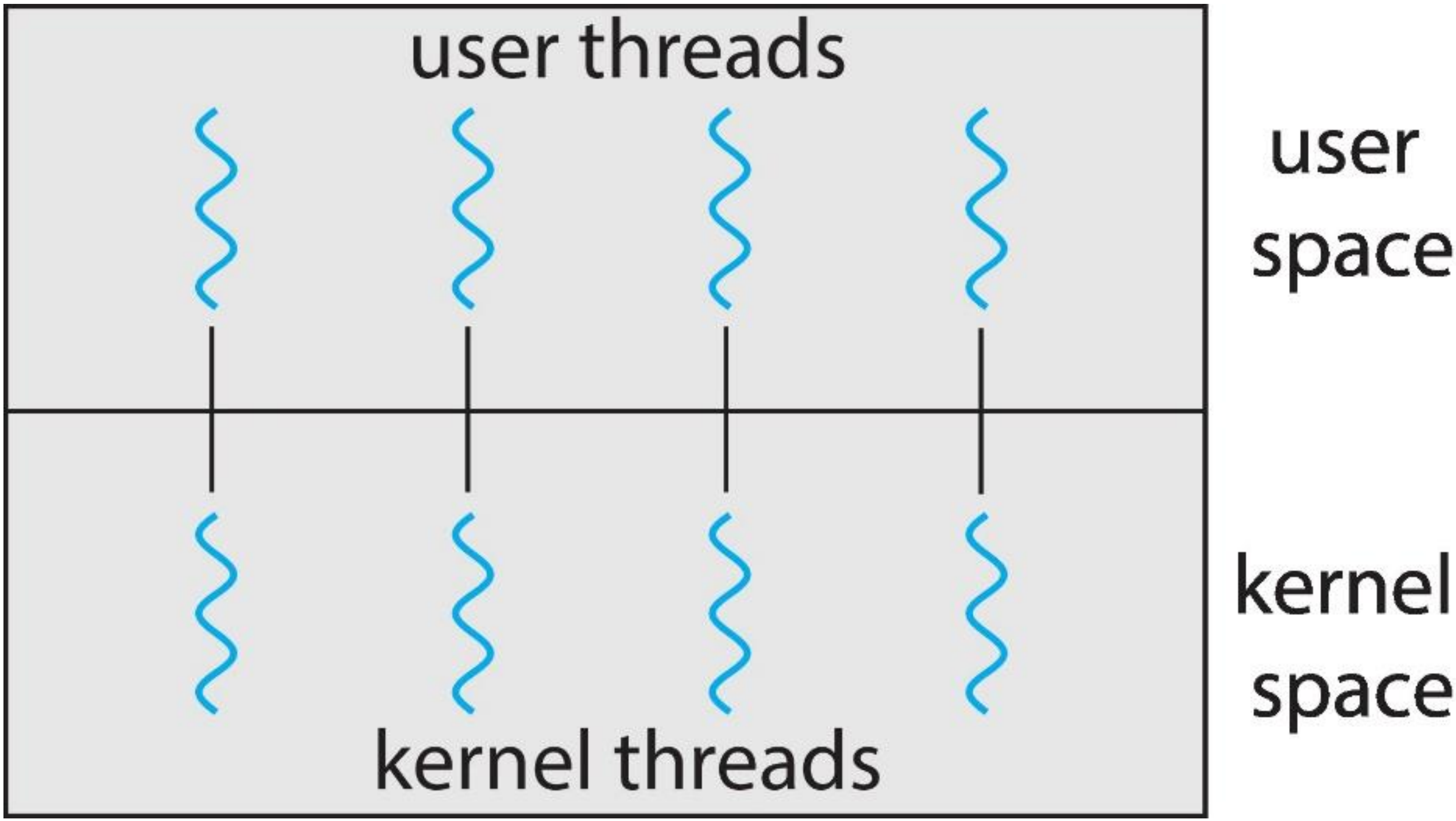
Many-to-One



One-to-One

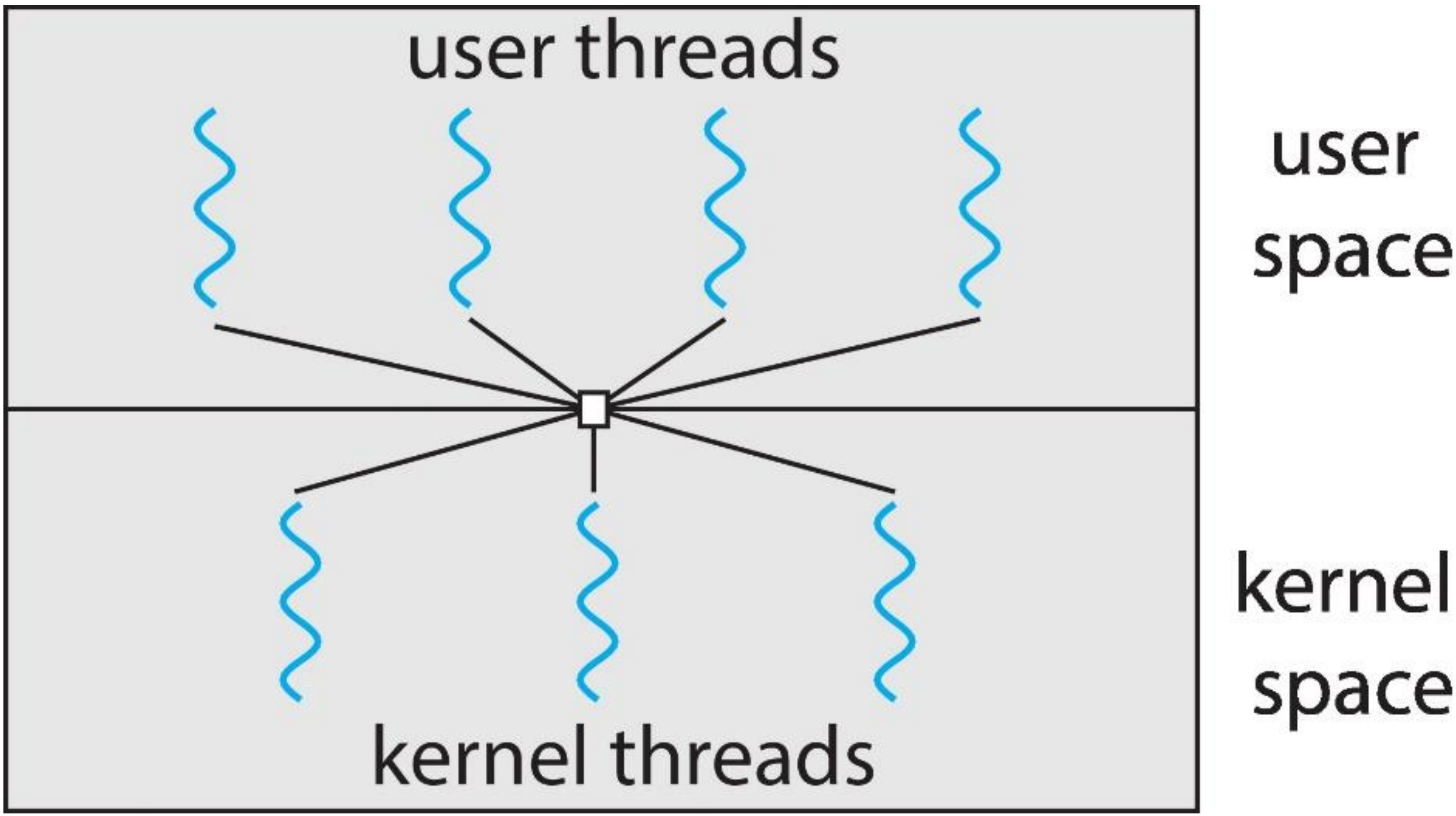
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux

One-to-One



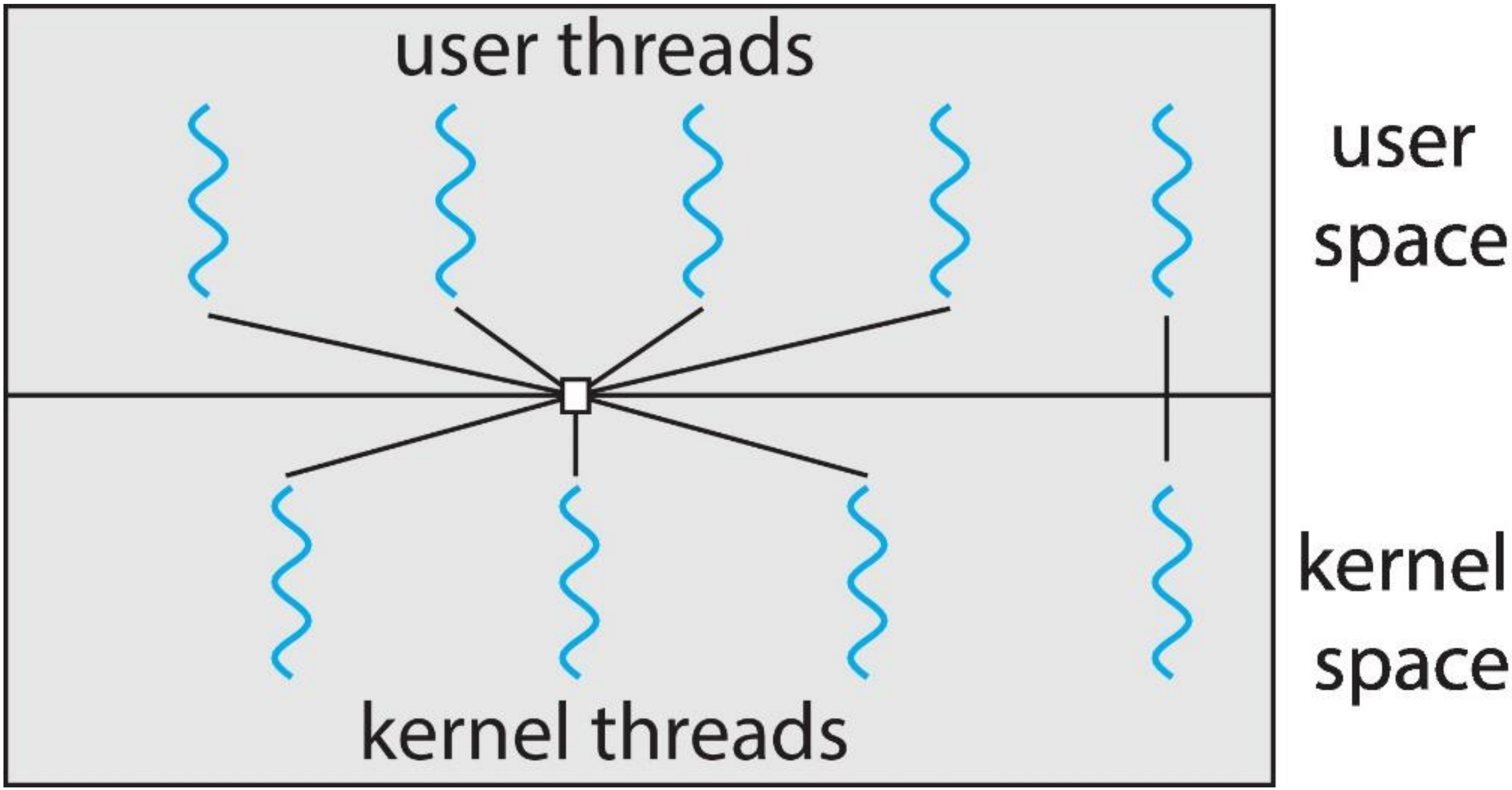
Many-to-Many Model

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


user threads
user
space
kernel
space
kernel threads

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

Questions?

- Concept of thread
- Parallelism and concurrency
- Data and task parallelism
- Amdahl's Law
- Multithread model