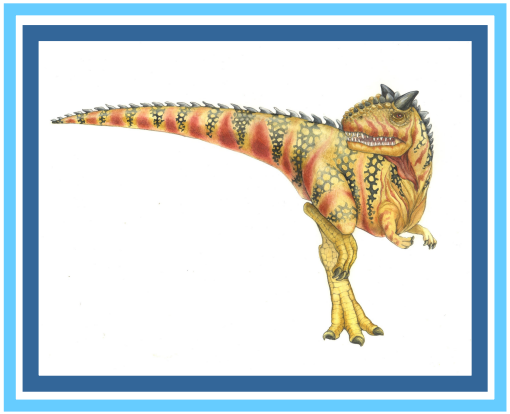
Chapter 4: Threads



Chapter 4: Threads

- Overview
- Multicore Programming
- Multithreading Models
- Threading Issues

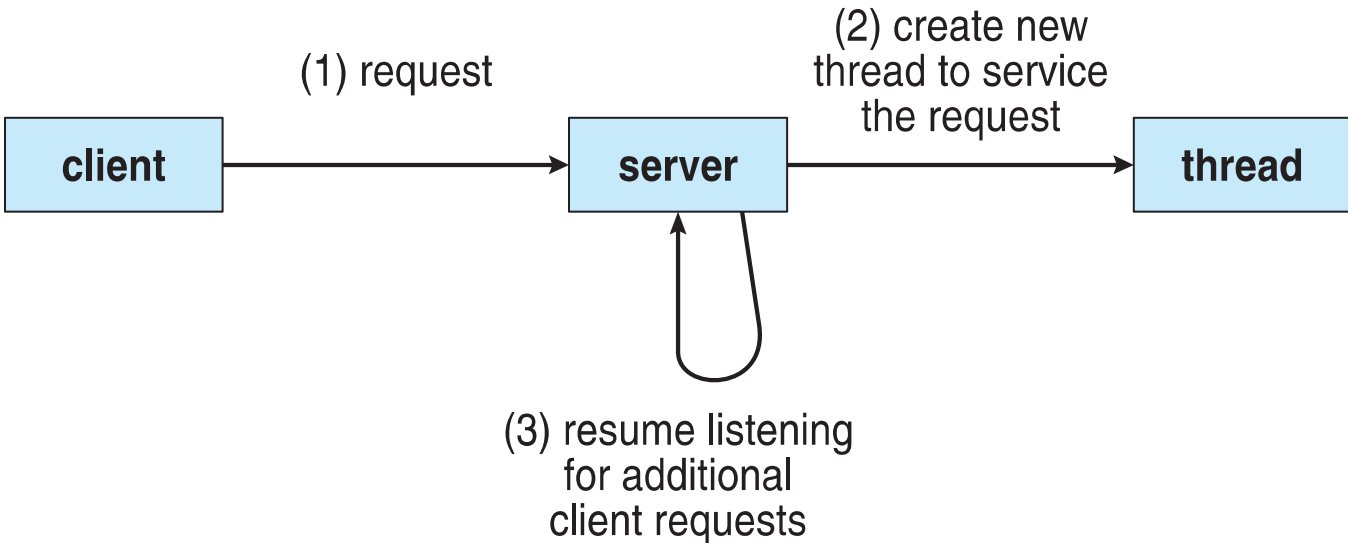
Definition

- **A *thread* is a process with a narrower context.**
  - **The minimum context a thread must contain consists of:**
    - **program counter**
    - **status register**
- Thus a thread may be referred to as a *lightweight process*
- while a (traditional) process may be referred to as a *heavyweight thread*.
- Moreover, a process may have (spawn) several threads.

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- OS Kernels are generally multithreaded

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 (for instance, registers), easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread's context switching requires less overhead than process' context switching
- **Scalability –** process can take advantage of multiprocessor architectures

Multicore Programming

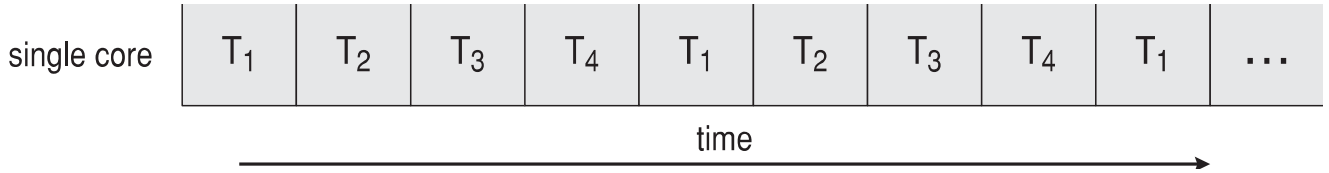
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** supports more than one task making progress
  - Single processor / core - scheduler provides concurrency with no or little parallelism
  - Multiprocessor / multicore - scheduler provides concurrency with parallelism
- DMA allows scheduler to provide parallelism even without concurrency

Multicore Programming (Cont.)

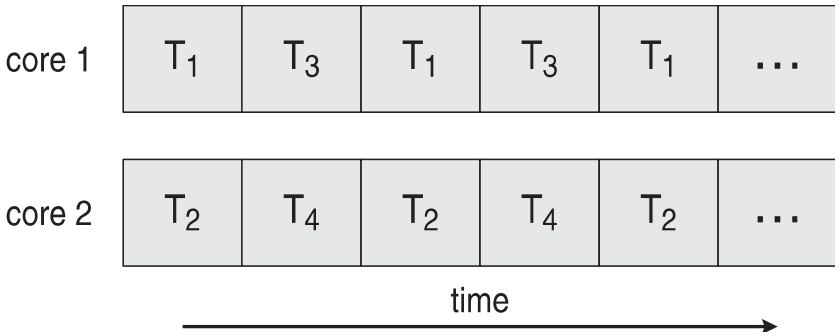
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

Concurrency vs. Parallelism

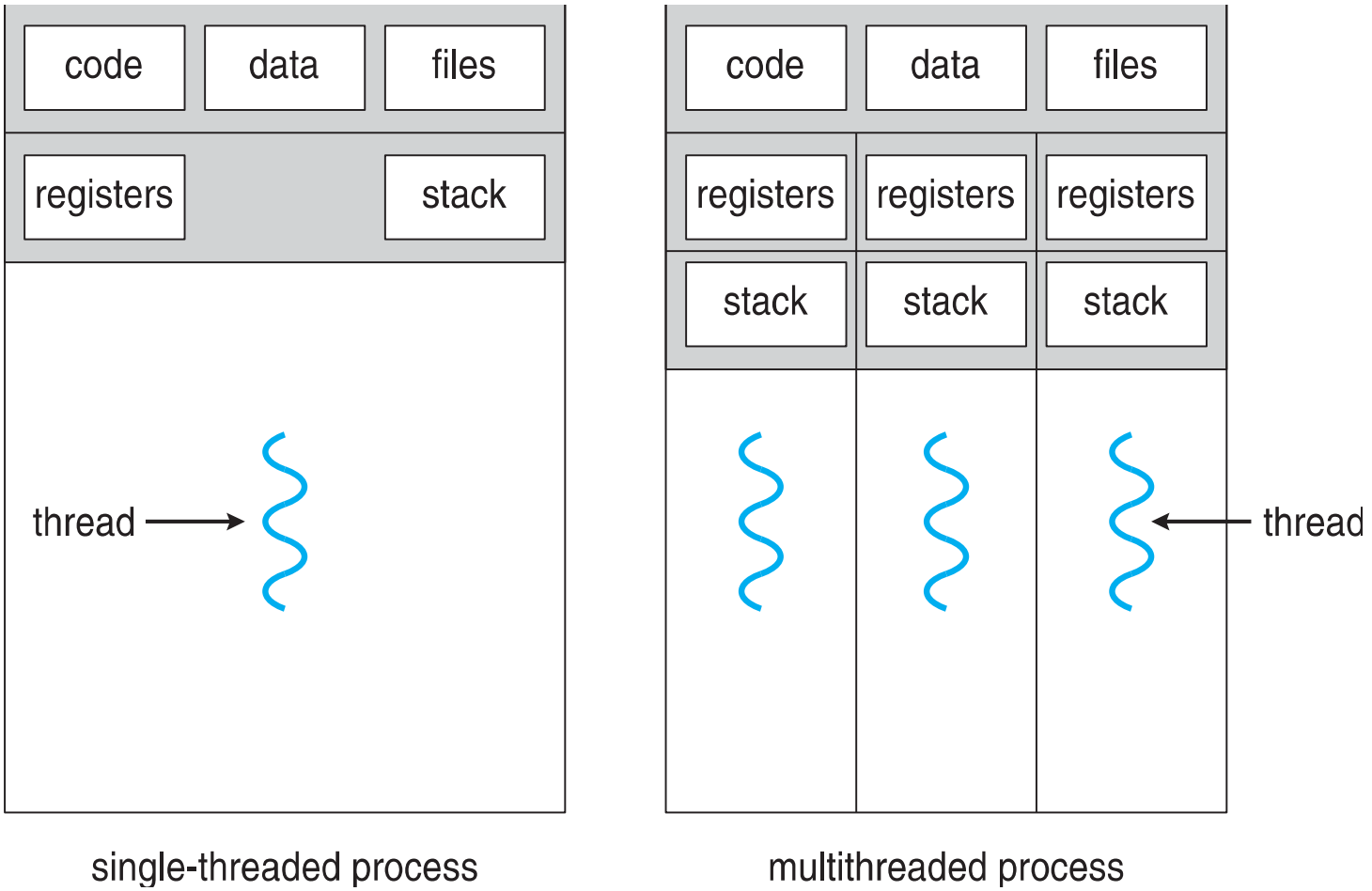
- **Concurrent execution on single-core system:**



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



Single and Multithreaded Processes



Amdahl's Law

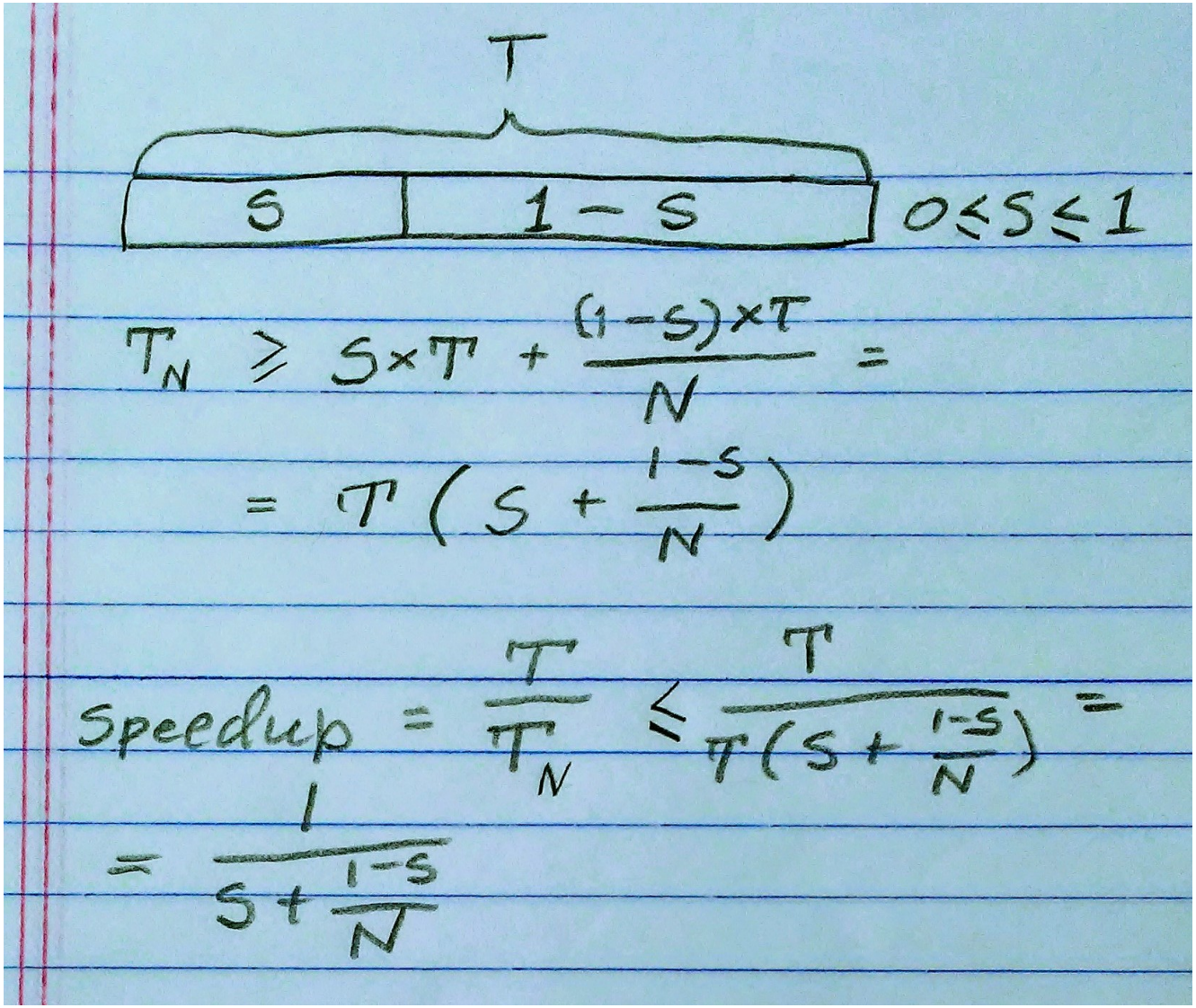
- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- *S* is sequential portion (cannot be parallelized)
- *For N* processors (cores), the upper bound on speedup is:

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

- 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 (slowing down) effect on performance gained by adding additional cores

Amdahl's Law

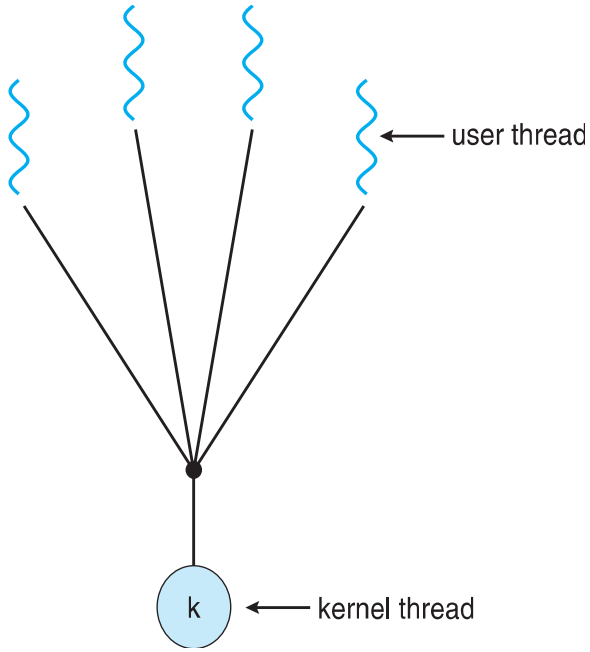

T
S
1 − S
0 ≤ S ≤ 1
$T_N \geq S \times T + \frac{(1-S) \times T}{N} =$
$= T\left(S + \frac{1-S}{N}\right)$
$\text{Speedup} = \frac{T}{T_N} \leq \frac{T}{T\left(S + \frac{1-S}{N}\right)} =$
$= \frac{1}{S + \frac{1-S}{N}}$

Multithreading Models

- Many-to-One (library-supported threads typically belong to this category)
- One-to-One (currently, the most typical scenario for OS-supported threads)
- Many-to-Many

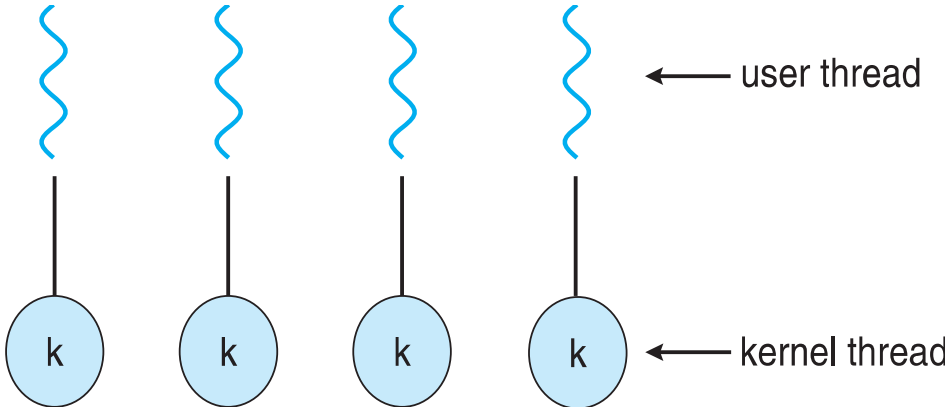
Many-to-One

- 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 the kernel at a time
- Few systems currently use this model
- Examples:
  - **Library-supported threads**
  - **Solaris Green Threads**
  - **GNU Portable Threads**



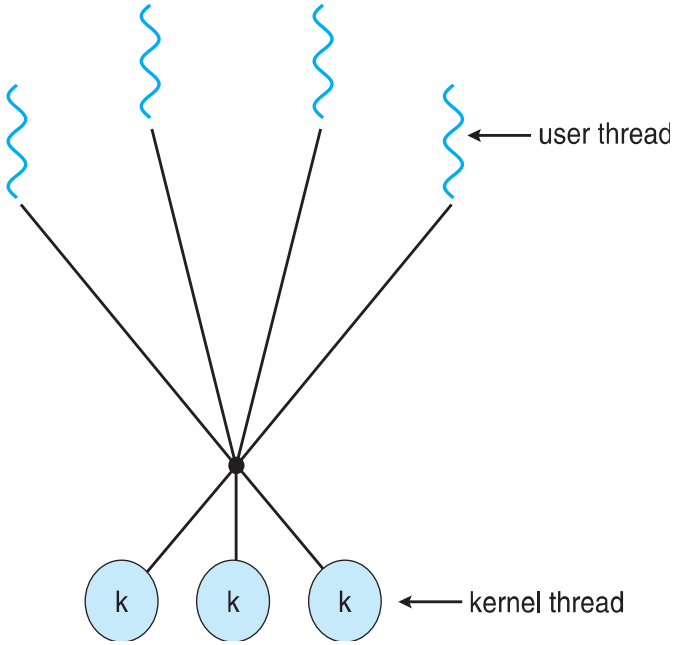
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- Allows more parallelism than many-to-one
- Number of threads per process may be restricted due to overhead of thread creation
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



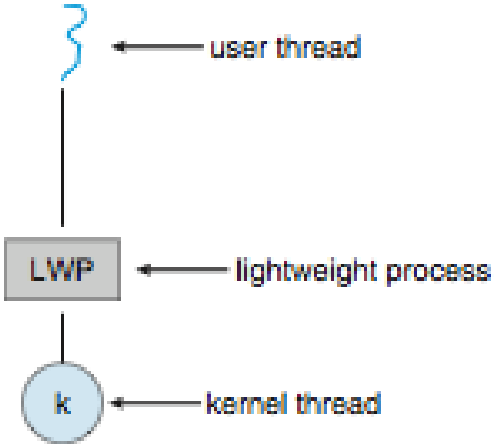
Many-to-Many Model

- 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 with the *ThreadFiber* package



Scheduler Activations

- Typically use an intermediate data structure between user and kernel threads – **lightweight process** (**LWP**)
  - Appears to be a virtual processor on which process can schedule user thread to run
  - Each LWP attached to kernel thread



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)
  - Flags control behavior

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

- **`struct task_struct`** points to process data structures (shared or unique)

End of Chapter 4