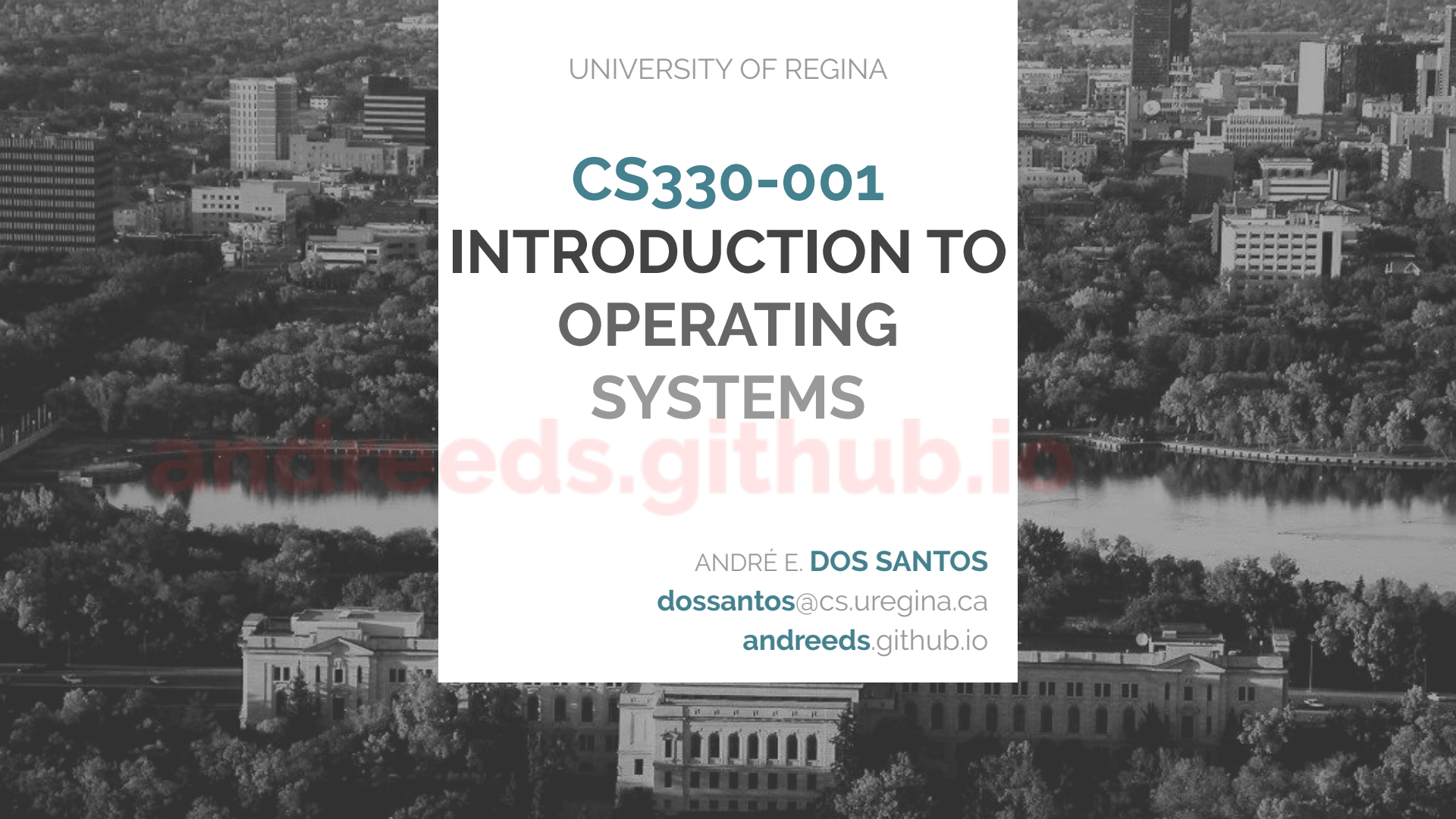
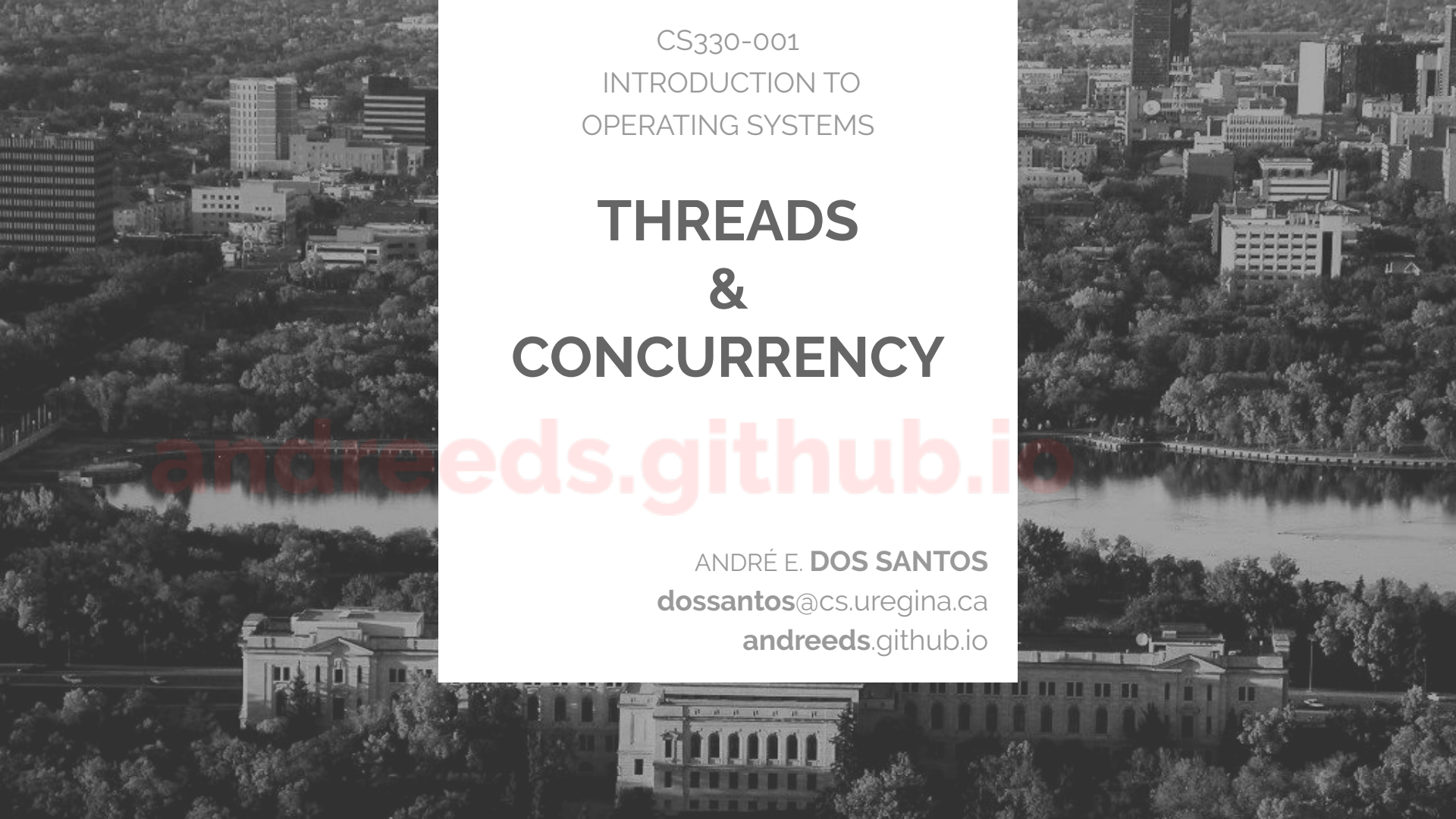


UNIVERSITY OF REGINA

CS330-001
INTRODUCTION TO OPERATING SYSTEMS

ANDRÉ E. **DOS SANTOS**
dossantos@cs.uregina.ca
andreeds.github.io

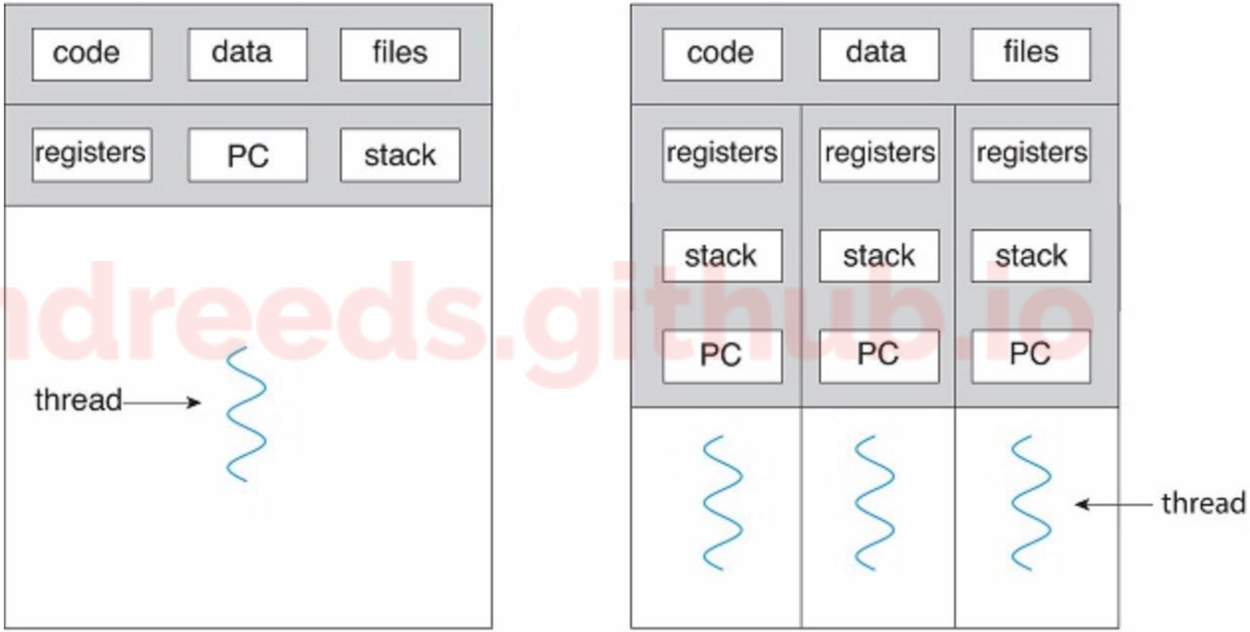


CS330-001
INTRODUCTION TO
OPERATING SYSTEMS

THREADS & CONCURRENCY

ANDRÉ E. **DOS SANTOS**
dossantos@cs.uregina.ca
andreeds.github.io

SINGLE AND MULTITHREADED PROCESSES



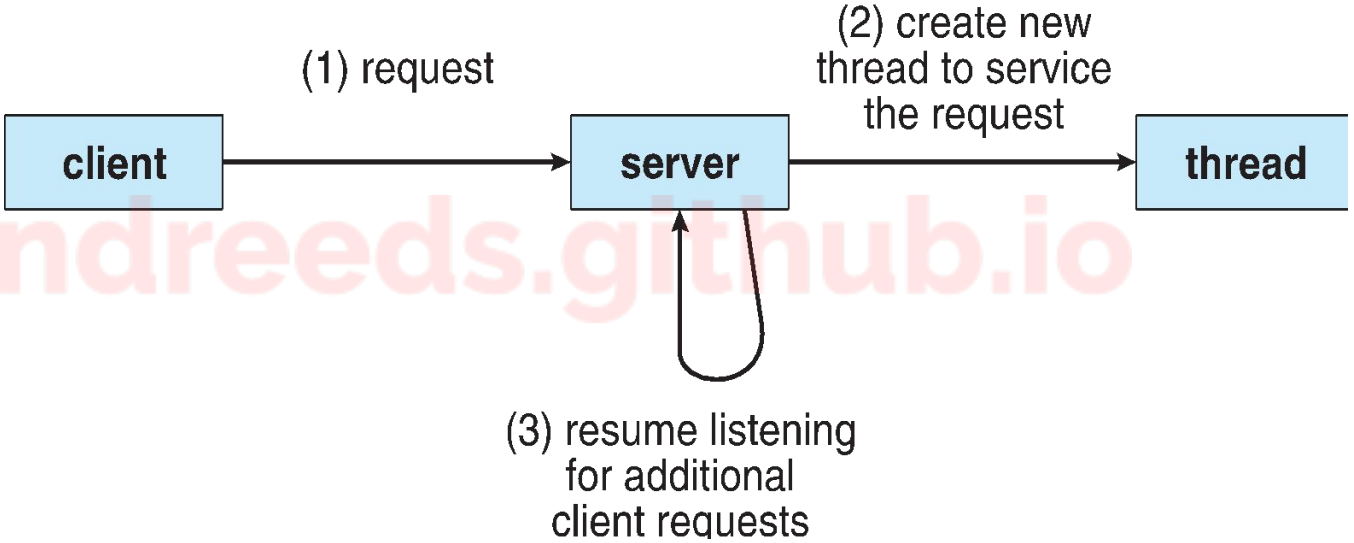
single-threaded process

multithreaded process

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

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 multiprocessor architectures

MULTICORE PROGRAMMING

- **Multicore** or **multiprocessor** systems challenges include:
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging

- **Concurrency** supports **more than one task making progress**
  - Single processor / core, scheduler providing concurrency
- **Parallelism** implies a system can perform **more than one task simultaneously**
  - Parallelism requires hardware with multiple processing units, essentially
  - In single core CPU, you may get concurrency but NOT parallelism

CONCURRENCY VS. PARALLELISM

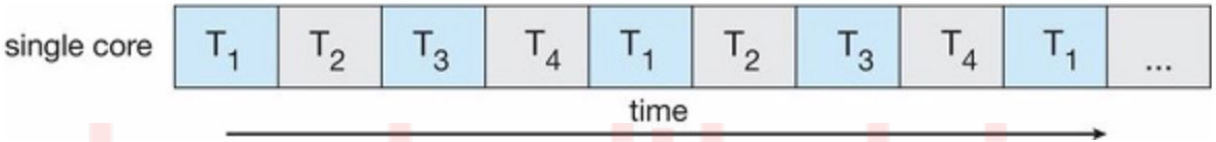


Figure 4.3 Concurrent execution on a single-core system.

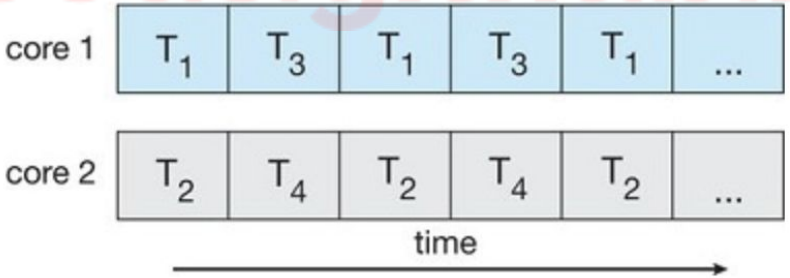


Figure 4.4 Parallel execution on a multicore system.

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

TYPES OF PARALLELISM

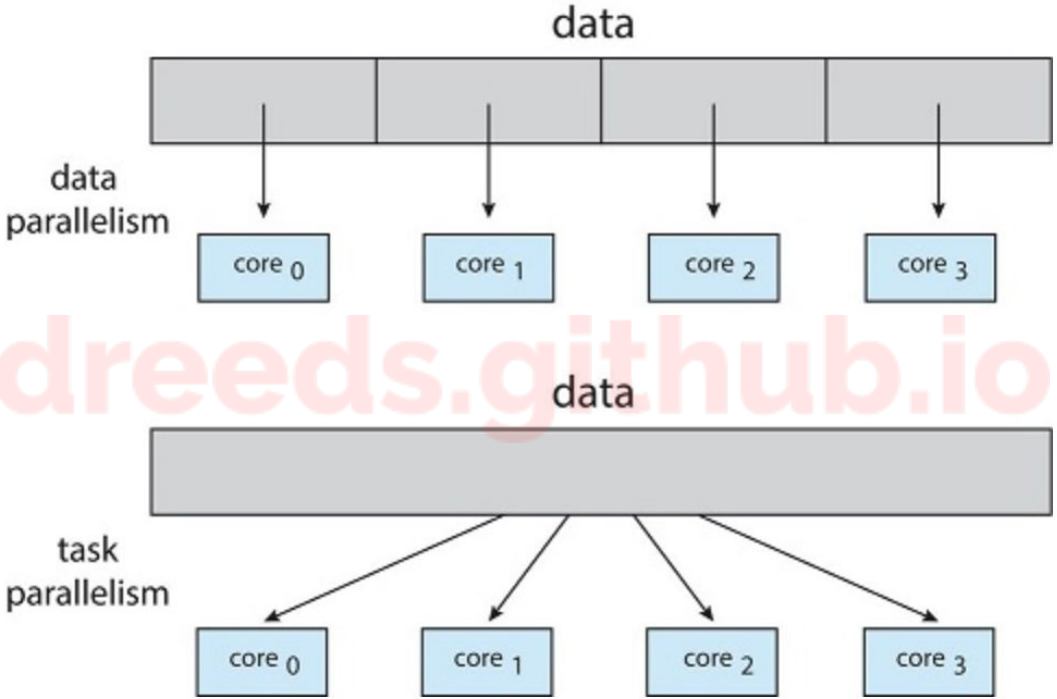


Figure 4.5 Data and task parallelism.

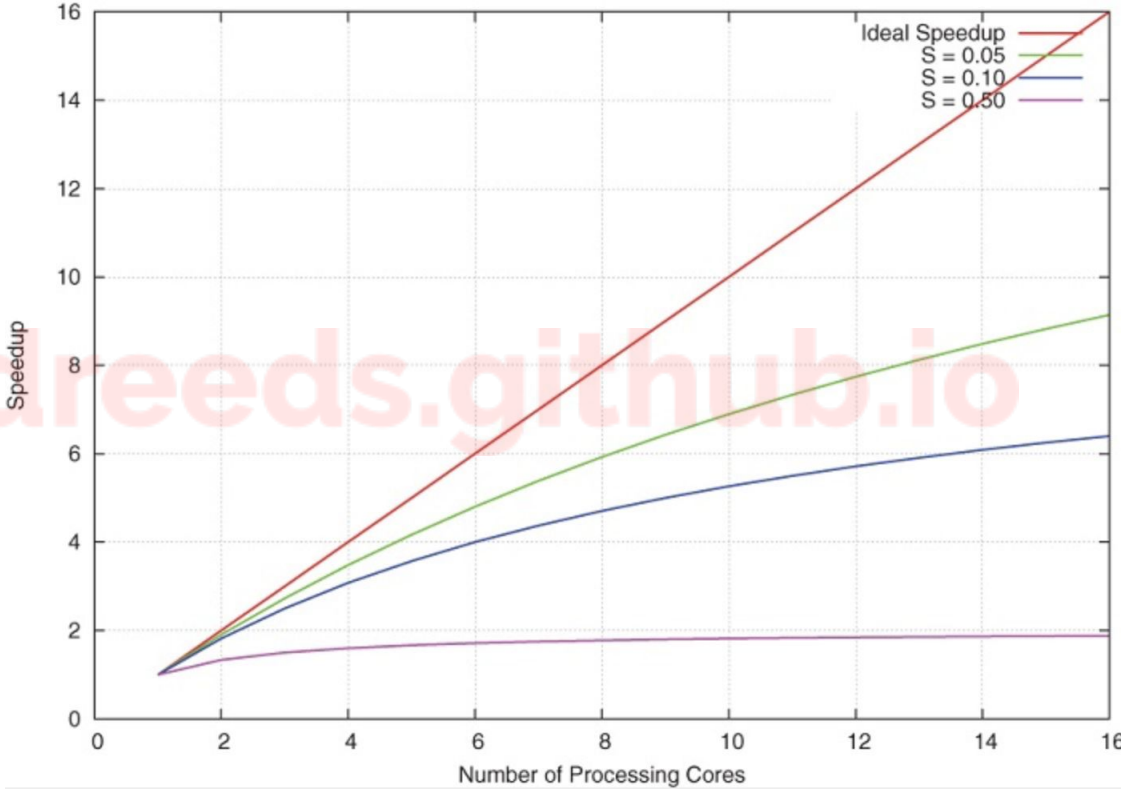
AMDAHL'S LAW

- 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`

AMDAHL'S LAW



USER THREADS AND KERNEL THREADS

User threads

- management done by user-level threads library
  - POSIX Pthreads

Kernel threads

- Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

MULTITHREADING MODELS

Many-to-One
One-to-One
Many-to-Many
Two-level Model

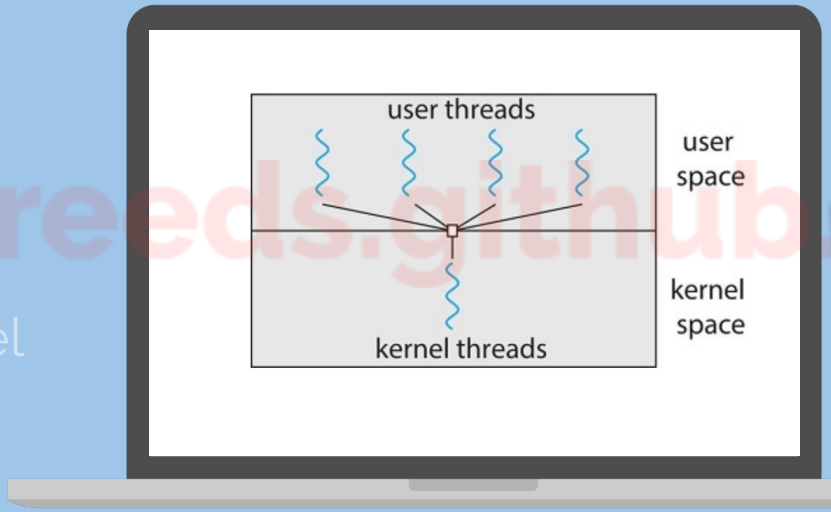
MULTITHREADING MODELS

Many-to-One

One-to-One

Many-to-Many

Two-level Model



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

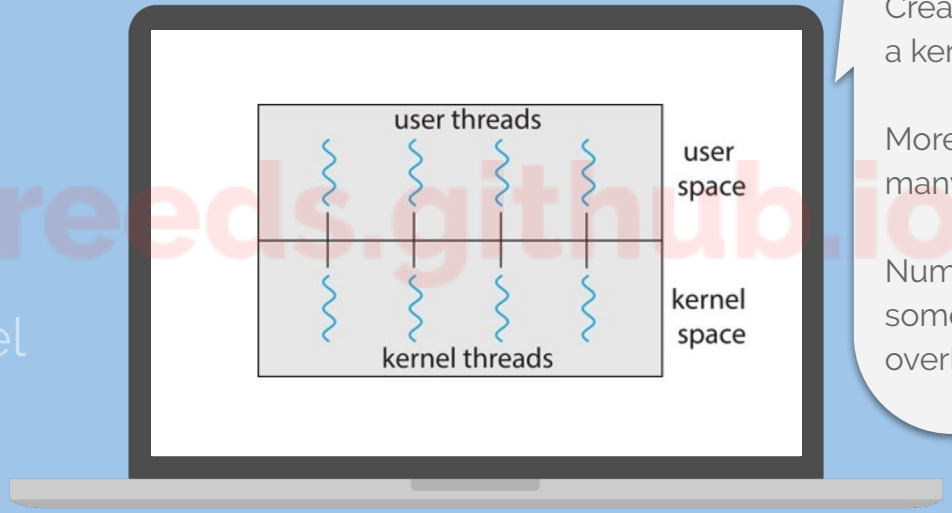
MULTITHREADING MODELS

Many-to-One

One-to-One

Many-to-Many

Two-level Model



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

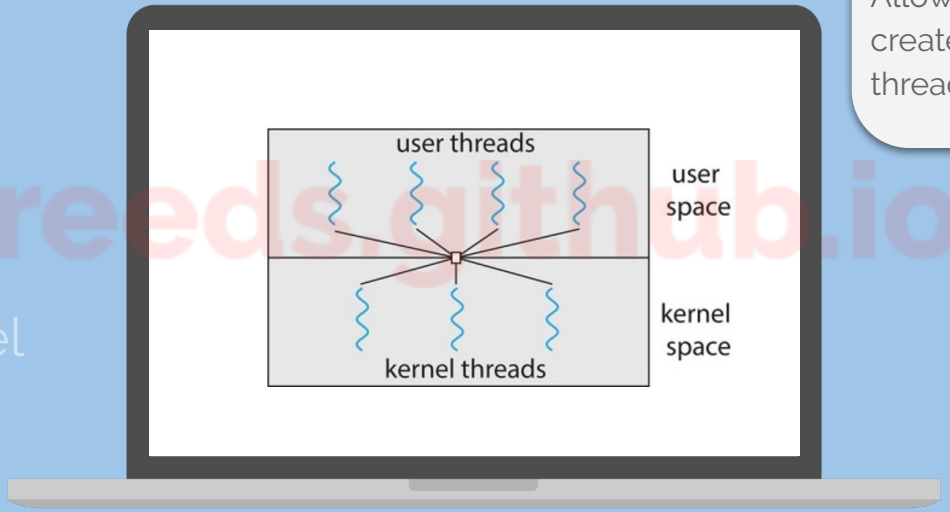
MULTITHREADING MODELS

Many-to-One

One-to-One

Many-to-Many

Two-level 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

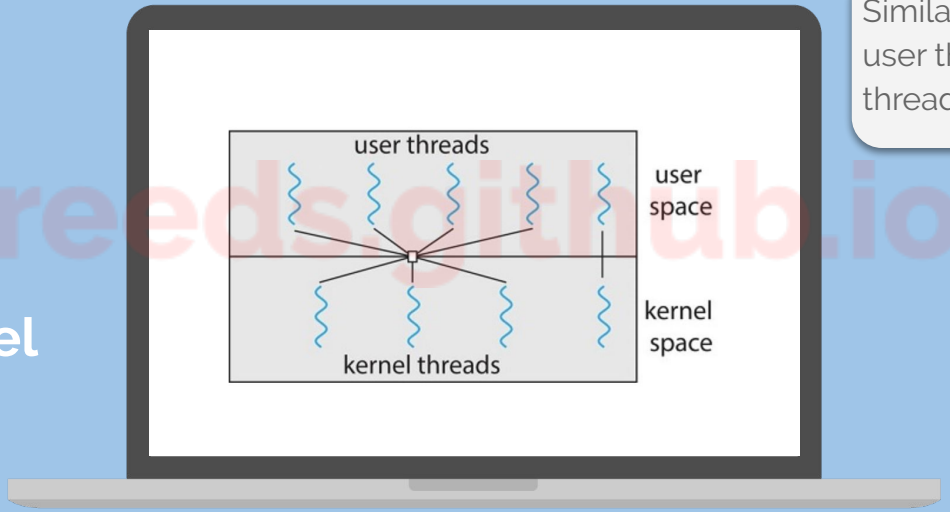
MULTITHREADING MODELS

Many-to-One

One-to-One

Many-to-Many

Two-level Model



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

IMPLICIT THREADING

- Growing in popularity as numbers of threads increase
  - program correctness more difficult with explicit threads
- Creation and management of threads **done by compilers and run-time libraries** rather than programmers

IMPLICIT THREADING

- **Thread Pools**
  - A number of threads created at process startup and placed in a pool, where they sit and wait for work
- **Fork Join**
  - A strategy for thread creation in which the main parent thread creates (forks) one or more child threads and then waits for the children to terminate and join with it
- **OpenMP**
  - OpenMP is a set of compiler directives as well as an API for programs that provides support for parallel programming in shared-memory environments
  - OpenMP identifies parallel regions as blocks of code that may run in parallel
- **Grand Central Dispatch (GCD)**
  - GCD is a technology developed by Apple for its macOS and iOS operating systems
  - It is a combination of a run-time library, an API, and language extensions that allow developers to identify sections of code (tasks) to run in parallel
  - Like OpenMP, GCD manages most of the details of threading

THREADING ISSUES

- Semantics of `fork()` and `exec()` system calls
- Signal handling
- Thread cancellation of target thread
- Thread-local storage
- Scheduler Activations

THREADING ISSUES

Semantics of `fork()` and `exec()` system calls

- Does **`fork()`** duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
- **`exec()`** usually works as normal – replace the running process including all threads

THREADING ISSUES

Signal handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.
- A **signal handler** is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled by one of two signal handlers:
    - default
    - user-defined
- Every signal has **default handler** that kernel runs when handling signal
  - **User-defined signal handler** can override default
  - For single-threaded, signal delivered to process

THREADING ISSUES

Thread cancellation of target thread

- 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
- **On Linux systems, thread cancellation is handled through signals**

THREADING ISSUES

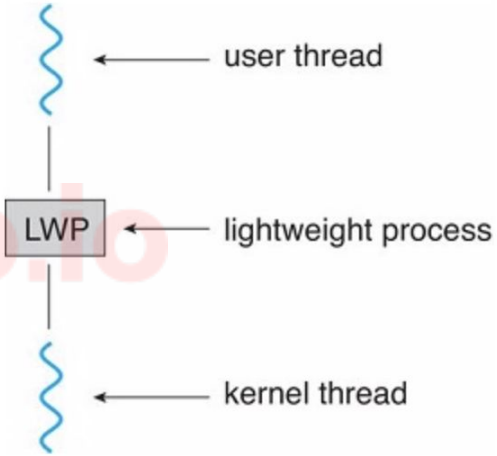
Thread-local storage

- **Thread-local storage (TLS)** allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process
  - i.e., when using a thread pool
- Different from local variables
  - Local variables visible only during single function invocation
  - TLS visible across function invocations
  - TLS is unique to each thread

THREADING ISSUES

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- 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
  - How many LWPs to create?



REVIEW QUESTIONS
THREADS & CONCURRENCY

A traditional (or heavyweight) process has a single thread of control. *True* or *False*?

Each thread has its own register set and stack. *True* or *False*?

_____ involves distributing tasks across multiple computing cores.

A) Concurrency
B) Task parallelism
C) Data parallelism
D) Parallelism

According to Amdahl's Law, what is the speedup gain for an application that is 60 parallel component for a two processing cores

A) 1.82
B) .7
C) .55
D) 1.43

It is possible to have concurrency without parallelism.*True* or *False*?

REVIEW QUESTIONS
THREADS & CONCURRENCY

The ____ multithreading model multiplexes many user-level threads to a smaller or equal number of kernel threads.

A) many-to-one model
B) one-to-one model
C) many-to-many model
D) many-to-some model

The _____ model maps many user-level threads to one kernel thread.

A) many-to-many
B) two-level
C) one-to-one
D) many-to-one

The _____ model allows a user-level thread to be bound to one kernel thread.

A) many-to-many
B) two-level
C) one-to-one
D) many-to-one

REVIEW QUESTIONS
THREADS & CONCURRENCY

Determine if the following problems exhibit task or data parallelism:

- Using a separate thread to generate a thumbnail for each photo in a collection
- Transposing a matrix in parallel
- A networked application where one thread reads from the network and another writes to the network
- The fork-join array summation application described
- The Grand Central Dispatch system

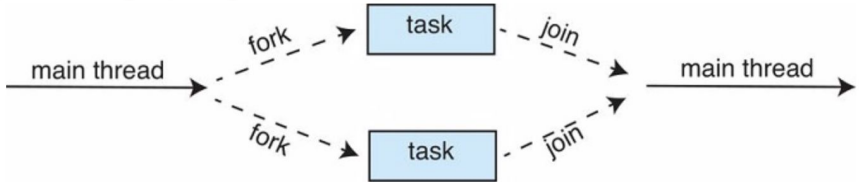


Figure 4.16 Fork-join parallelism.