MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT2100 Semester 2 2019

Lecture 7 (Part A): Threads

(Reading: Stallings, Chapter 4)

WEEK 8

GROUP OF EIGHT AUSTRALIA

Lecture 4 (Part 1): Learning Outcomes

- Upon the completion of this lecture, you should be able to:
  - Understand the distinction between process and thread
  - Describe the basic design issues for threads
  - Explain the difference between user-level threads and kernel-level threads
  - Discuss thread management in Unix/Linux*

*Reading from Stallings, Chapter 4 (4.6)

WHAT DO WE UNDERSTAND ABOUT PROCESSES?

- A process ‘owns’ resources
  - Space in main memory, open files, I/O devices, etc.
  - An independent **process image**
  - One process is prevented from interfering with another process’s resources and image as allocated by the OS

- Scheduling and execution
  - A process follows a ‘path’ of execution
  - Execution may be interleaved with other processes
  - Scheduled and dispatched by the OS.

What is a *thread* for a process?

The Concept of Threads

- The unit of dispatching for execution is referred to as:
  - Thread or
  - Lightweight process

- The unit of resource ownership is referred to as:
  - Process or task

The Concept of Threads

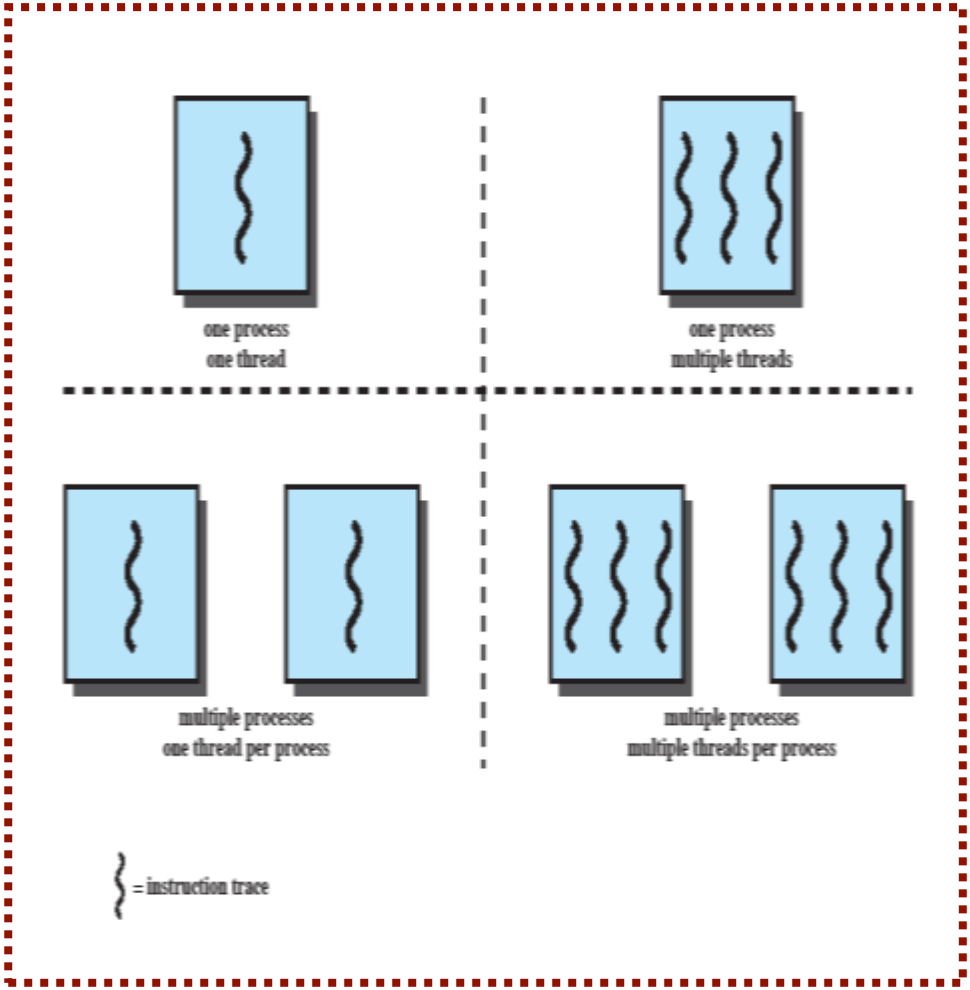
- The unit of dispatching for execution is referred to as:
  - Thread or
  - Lightweight process

- The unit of resource ownership is referred to as:
  - Process or task

- Multi-threading:
  - The ability of an OS to support multiple concurrent paths of execution within a single process
  - **1 process : multiple threads of execution.**

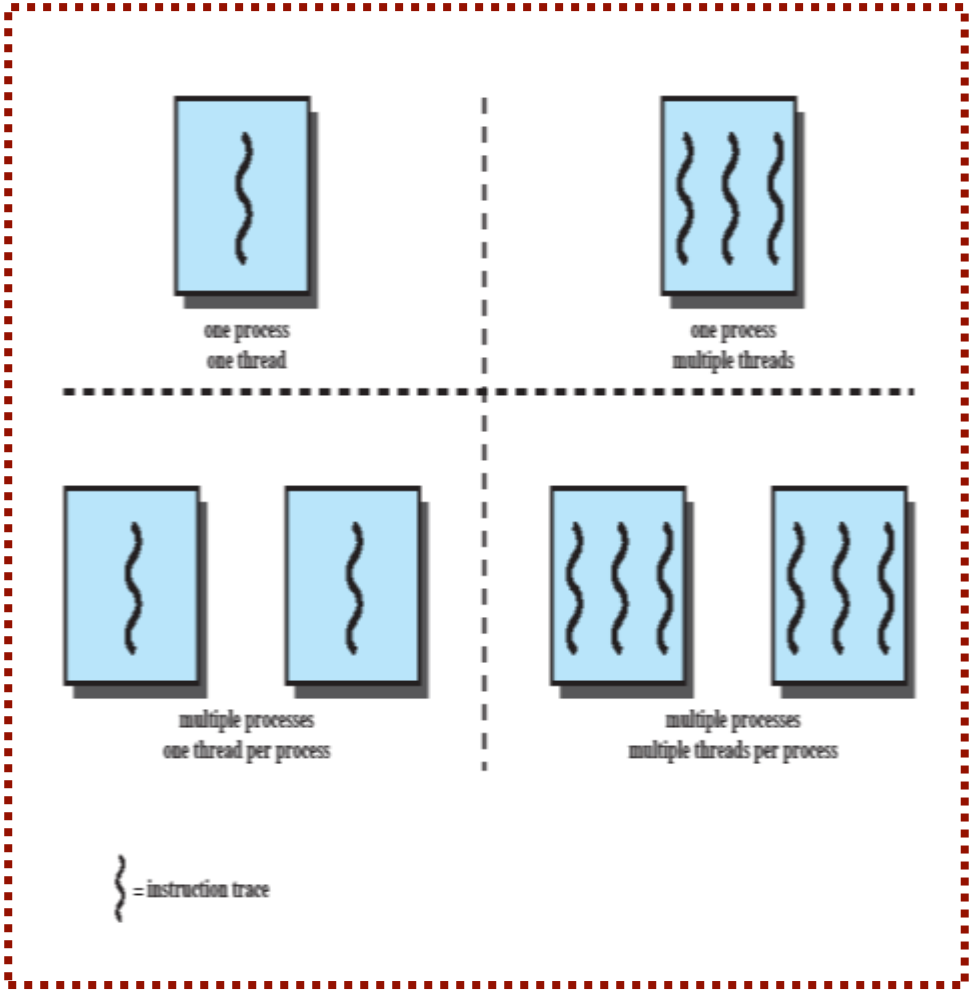
Single-Threaded Approaches

- Left side of figure:
  - A single thread of execution per process.
- The concept of a thread is not recognised — referred as a single-threaded approach.
- Example: MS-DOS, Windows 3.1



Multi-Threaded Approaches

- The right half of the figure depicts the multi-threaded approach.
- One process with multiple threads.
- Example: Windows, Linux.



The Concept of Processes (revisit)

- The unit of resource allocation and a unit of protection.
- A (virtual) address space that holds the process image.
- Protected access to:
  - Processor(s)
  - Other processes
  - Files
  - I/O resources

Threads within a Process

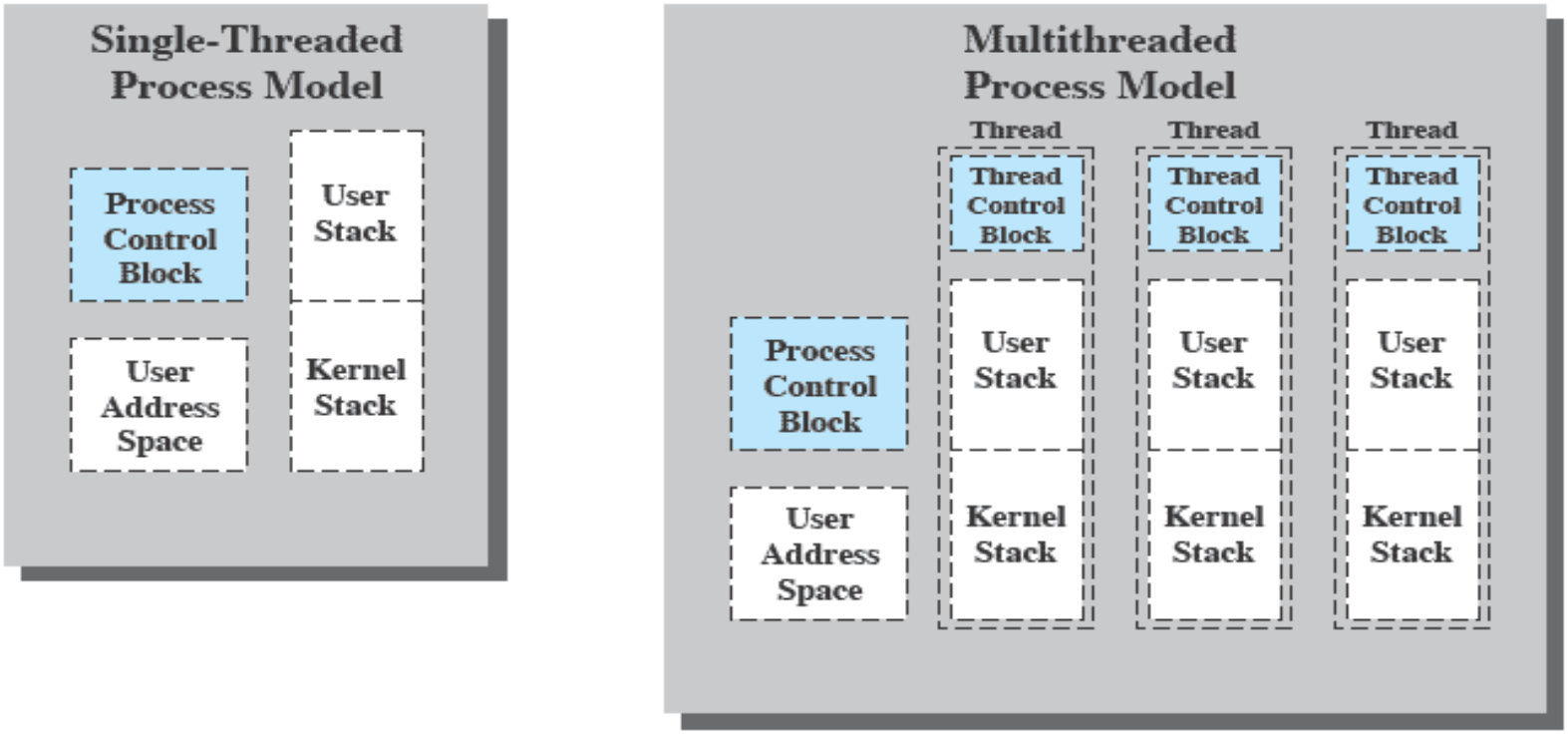
- Different part of a program may do different things and they can be executed concurrently to improve response time (or completion time).
  - Example: one thread may do a processor-bound task like rendering an image, while another thread responds to user interaction in the same program.
- If there is an interaction between different parts of the programs — concurrency control need to be applied.
- Example: accessing and modifying a common variable — mutual exclusion need to be satisfied.

Attributes of a Thread

Each thread has:

- an execution state (Running, Ready, etc.)
- saved thread context when not running
- an execution stack
- some per-thread static storage for local variables
- access to the memory and resources of its process (all threads of a process share this)

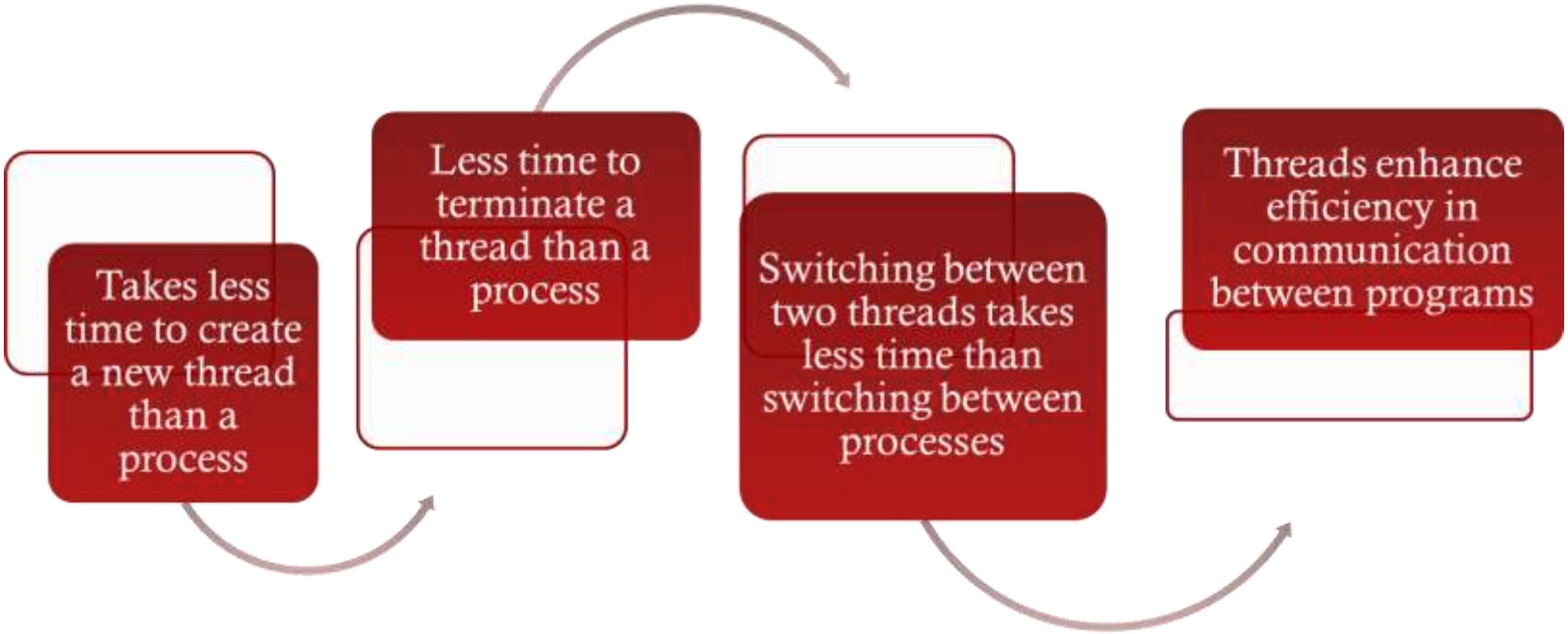
Threads vs. Processes



Single Threaded and Multithreaded Process Models

Benefits of Threads

Threads == 'lightweight processes'



More on Threads

- For an OS that supports threads, scheduling and dispatching is done on a thread basis.
- Most of the state information dealing with execution is maintained in thread-level data structures:
  - Suspending a process involves suspending all threads of a process.
  - Termination of a process terminates all threads within the process.

All threads within a process share the same address space.

MONASH
University

What are the *thread states*?

Thread Execution States

Thread States

- The key states for a thread:
  - RUNNING
  - READY
  - BLOCKED

State Transition

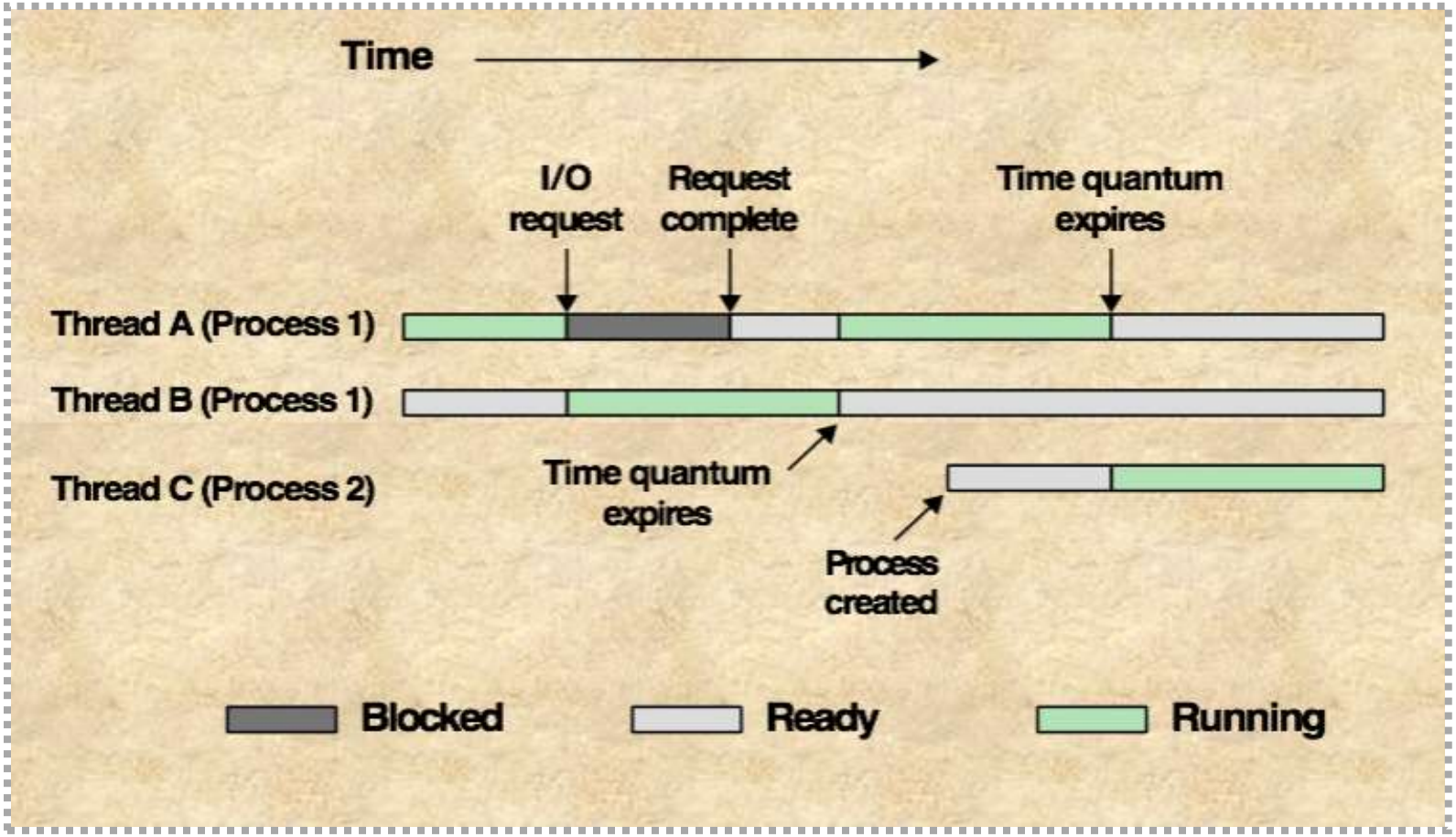
- Thread operations associated with a change in thread state:
  - Spawn
  - Block
  - Unblock
  - Finish

New threads

Wait for an event

A BLOCKED event occurs — move the thread to the READY queue

Multi-threading on a Uniprocessor



e.g. One thread may run while another thread is blocked.

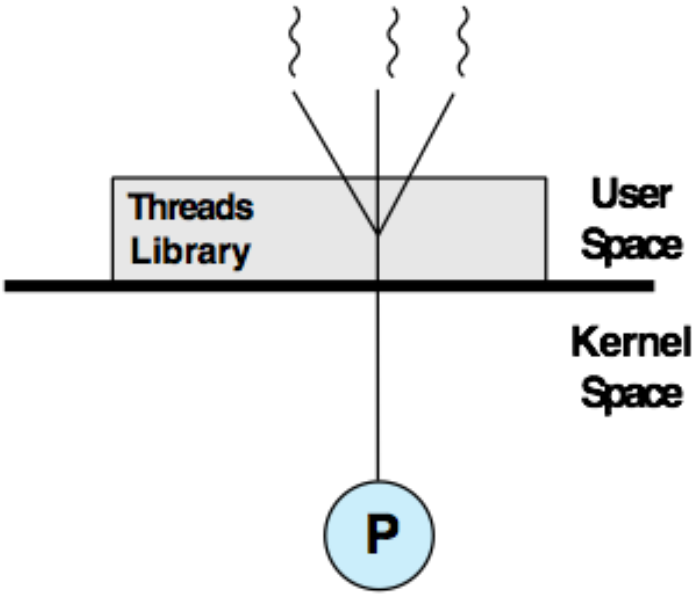
Thread Synchronisation

- It is necessary to synchronise the activities of various threads.
- All threads of a process share the same address space (e.g. global variables) and other system resources.
- Any alteration of a resource by one thread affects the other threads in the same process.

User-Level Thread (UTL) and Kernel-Level Thread (KLT)

User-Level Threads (ULTs)

- All thread management is done by the application.
- The kernel is not aware of the existence of threads.
- Any application can be programmed to be multi-threaded by using a threads library.
  - Even if OS does not support threads.



Pure user-level

ULTS: Advantages

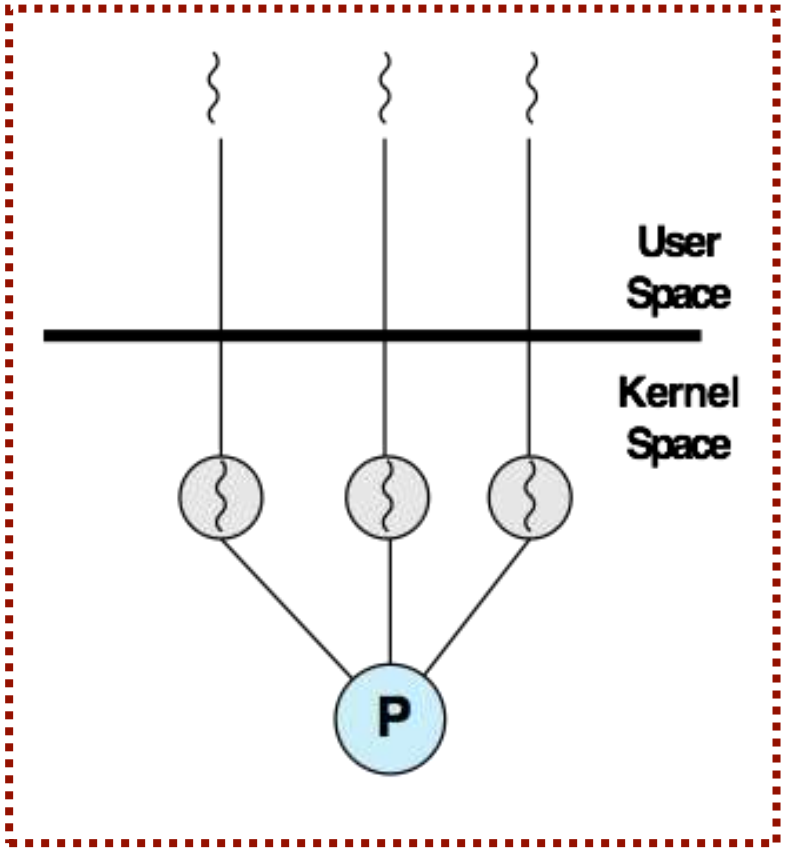
- Thread switching does not required kernel mode privileges
- Scheduling can be application specific
- ULTs can run on any Operating System

ULTS: Disadvantages

- In a typical OS, many system calls are *blocking*.
- When a ULT executes a system call, not only is that thread gets blocked, but all of the threads within the process are also blocked.
- In a pure ULT strategy, a multi-threaded application cannot take the full advantage of multiprocessing.

Kernel-Level Threads (ULTs)

- Thread management is done by the kernel.
- No thread management is done by the application — through API to the kernel thread facility.
- Example: Windows, Linux



Pure kernel-level

KLTS: Advantages

- The kernel can simultaneously schedule multiple threads from the same process on **multiple processors**.
- If one thread in a process is blocked, the kernel can schedule another thread of the same process.
- The kernel routines can also be multi-threaded.

KLTS: Disadvantages

- The transfer of control from one thread to another thread within the same process requires a mode switch to the kernel.
  - Some overhead here.

How do Unix/Linux systems manage threads?

Linux Tasks

A process (or task) in Linux is represented by a *task_struct* data structure.

- Execution stage
- Scheduling information
- Identifiers (IDs)
- Links to parents/children/siblings
- Timers
- File system
- Address space
- Processor-specific context information

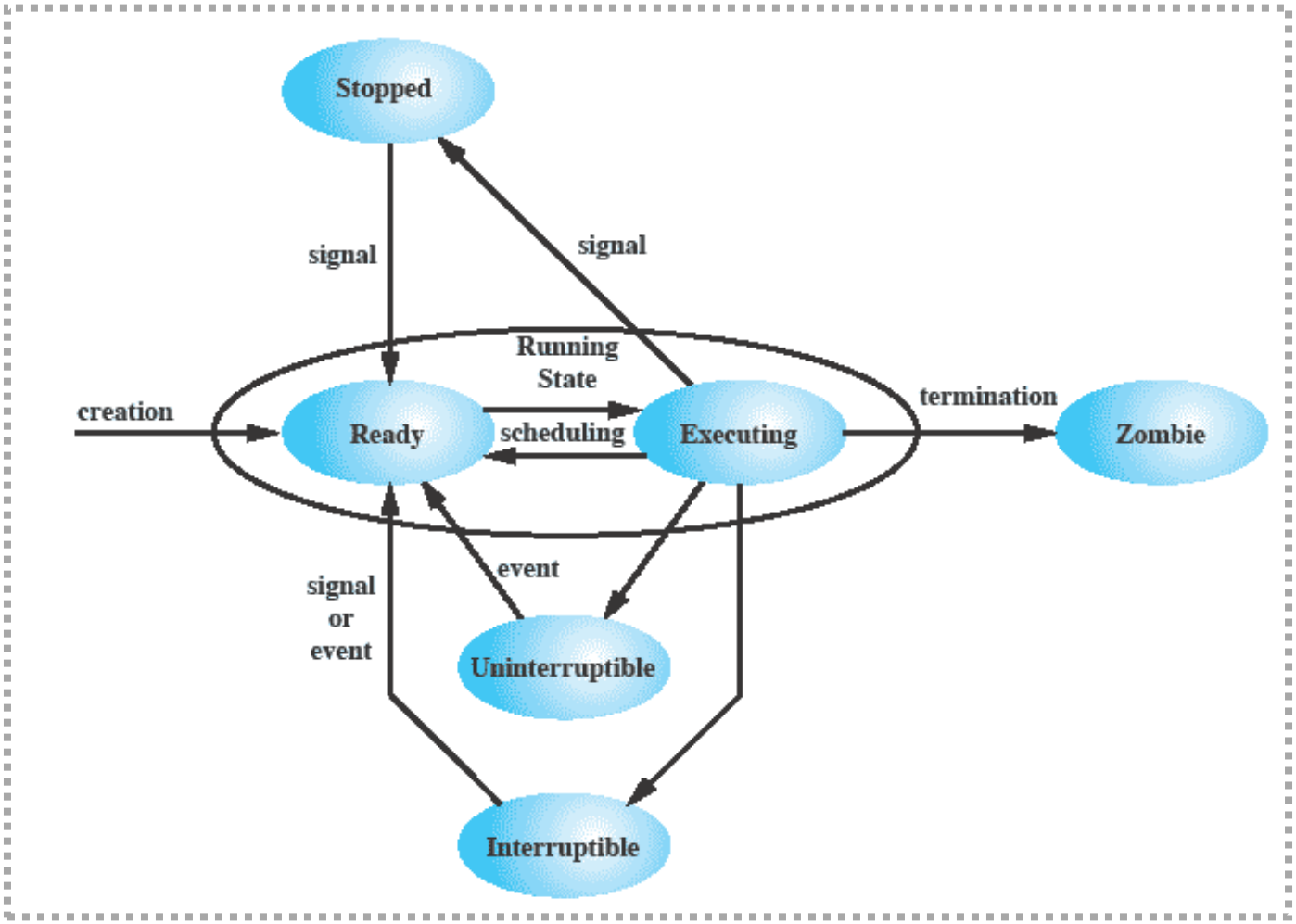
PTHREADS

- Threads in Linux are known as **pthreads**. Managed through a separate API.

- The **pthread** library must be included and linked into the program in order to use threads.
  - **`#include <pthread.h>`**
  - Add **`-lpthread`** to the end of the **gcc** command to **link** the program against the pthread library

- **`pthread_create()`** – spawn a new thread
- **`pthread_join()`** – wait for another thread to terminate

Linux: Process/Thread Model



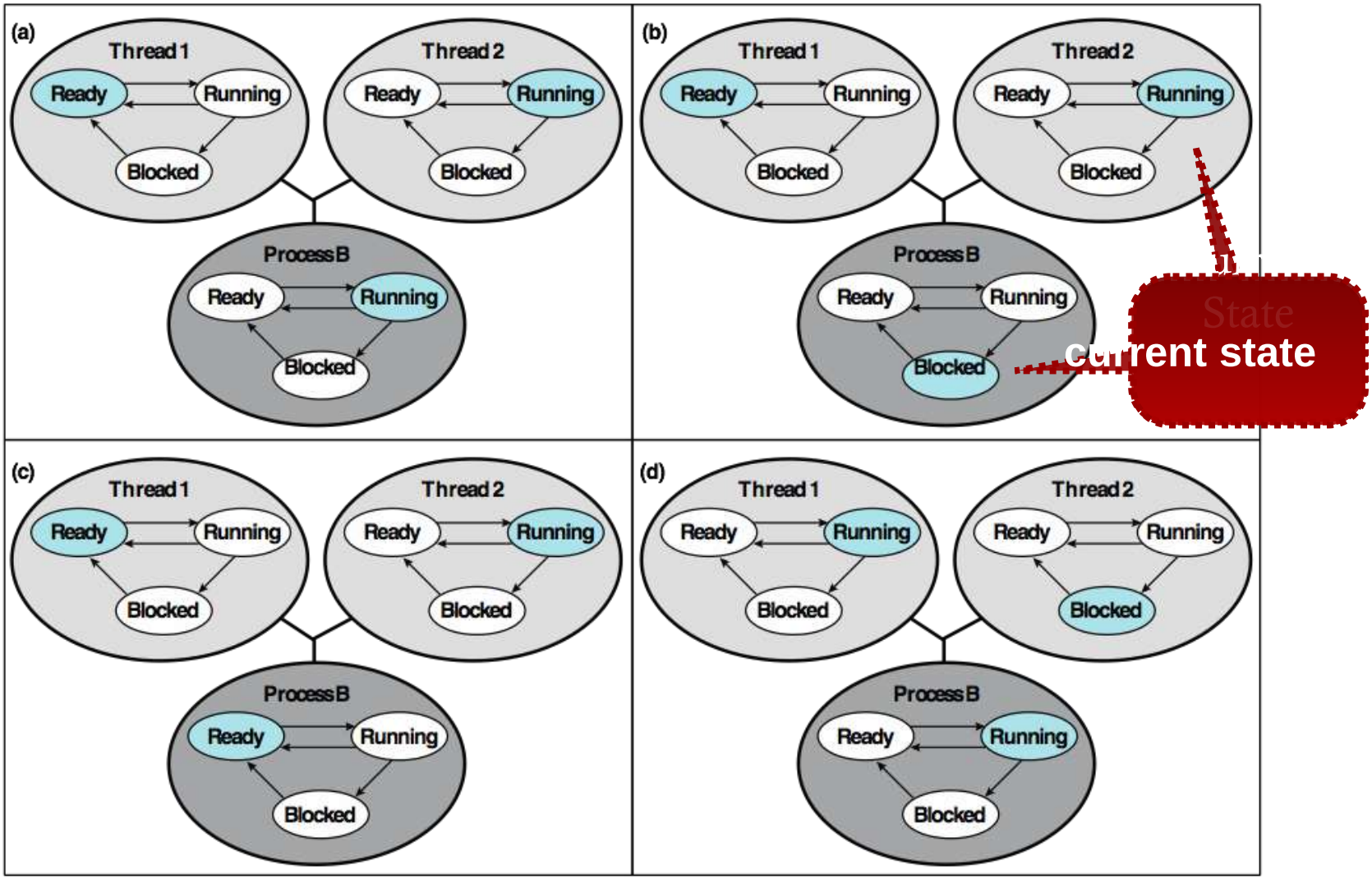
Summary of Lecture 4 (Part A)

- The concept of process is related to resource ownership.
- The concept of thread is related to program execution.
- In multi-threaded system, multiple concurrent threads may be defined with a single process.
- Two types of threads: user-level and kernel level.

Reading from Stallings, Chapter 4: 4.1, 4.2 and 4.6

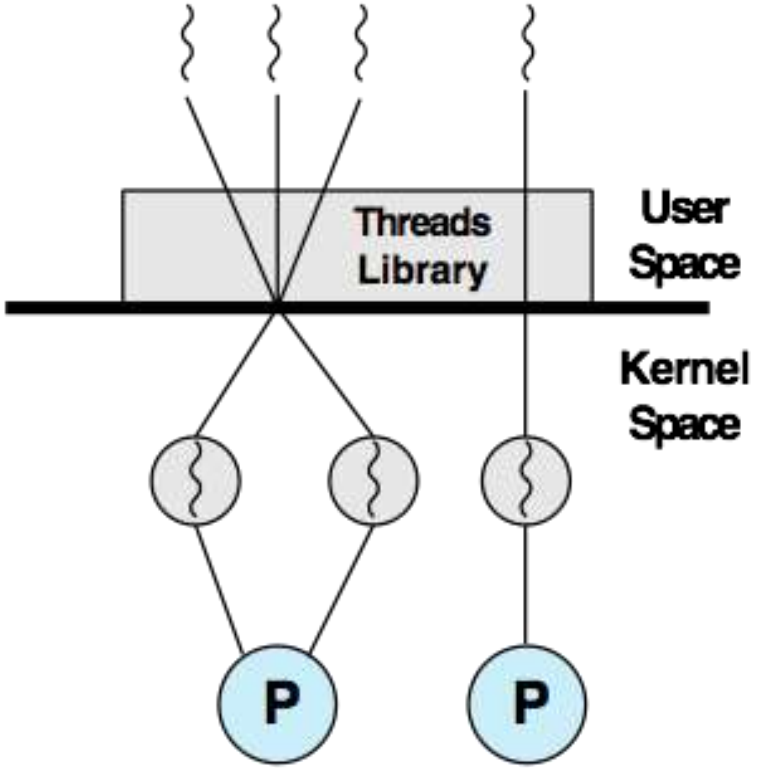
SUPPLEMENTARY SLIDES

Thread Scheduling and Process Scheduling



Combined Approaches (ULT and KLT)

- Thread creation is done in the user space.
- Bulk of scheduling and synchronisation of threads are by the application.
- Multiple ULTs from a single application is mapped onto smaller (or equal) number of KLTs.
- Example: Solaris (Unix)



Linux: Namespaces

- A namespace enables a process to have a different view of the system than other processes that have other associated namespaces.
- One of the overall goals to support the implementation of control groups (*cgroups*).
- A tool for lightweight *virtualisation* — provides a process or group of processes with the illusion that they are the only processes on the system.
- Six namespaces in Linux:

Relationships between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Linux: Threads