MONASH University

MONASH
INFORMATION
TECHNOLOGY

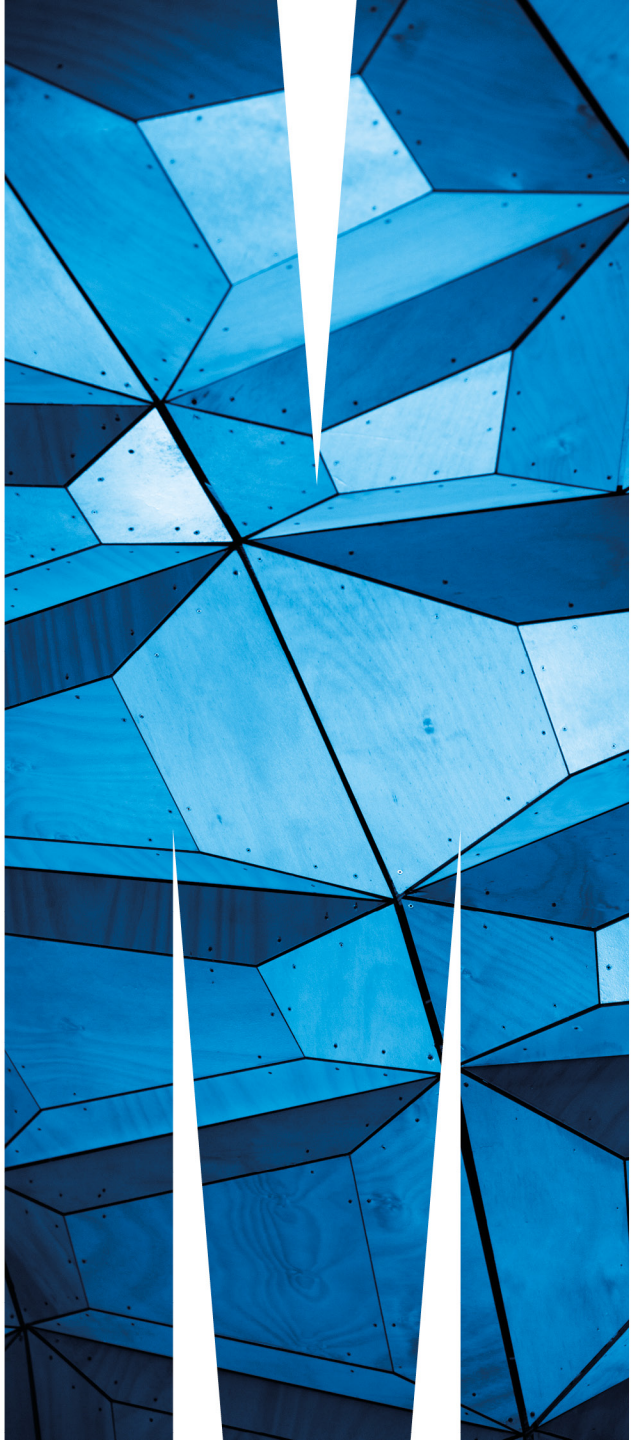
FIT2100 Semester 2 2017

Lecture 4 (Part A): Threads

(Reading: Stallings, Chapter 4)

Jojo Wong

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 the thread management in Unix/Linux

MONASH University

What do we understand about processes?

Processes and Threads

Resource Ownership

- Process includes a (virtual) address space to hold the process image.
- OS performs a protection function to prevent unwanted interference between processes with system resources.

Scheduling/Execution

- Follows an execution path that may be interleaved with others processes.
- A process has an **execution state** (RUNNING/READY/ BLOCKED/SUSPENDED) and a **dispatching priority**.
- It is scheduled and dispatched by 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

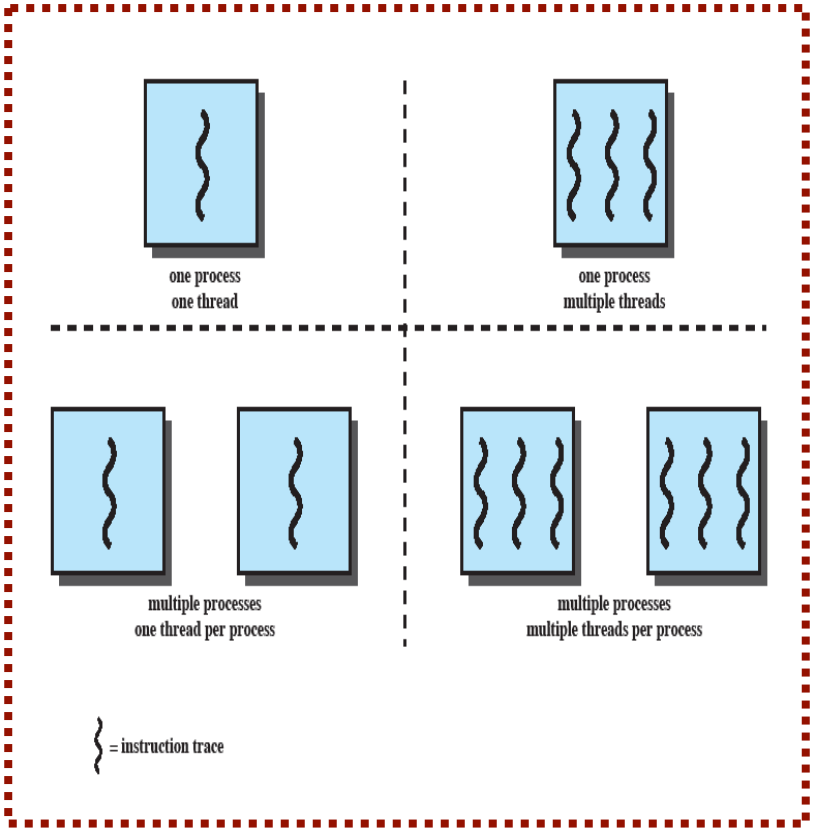
- The unit of resource ownership is referred to as:
  - Process or task

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

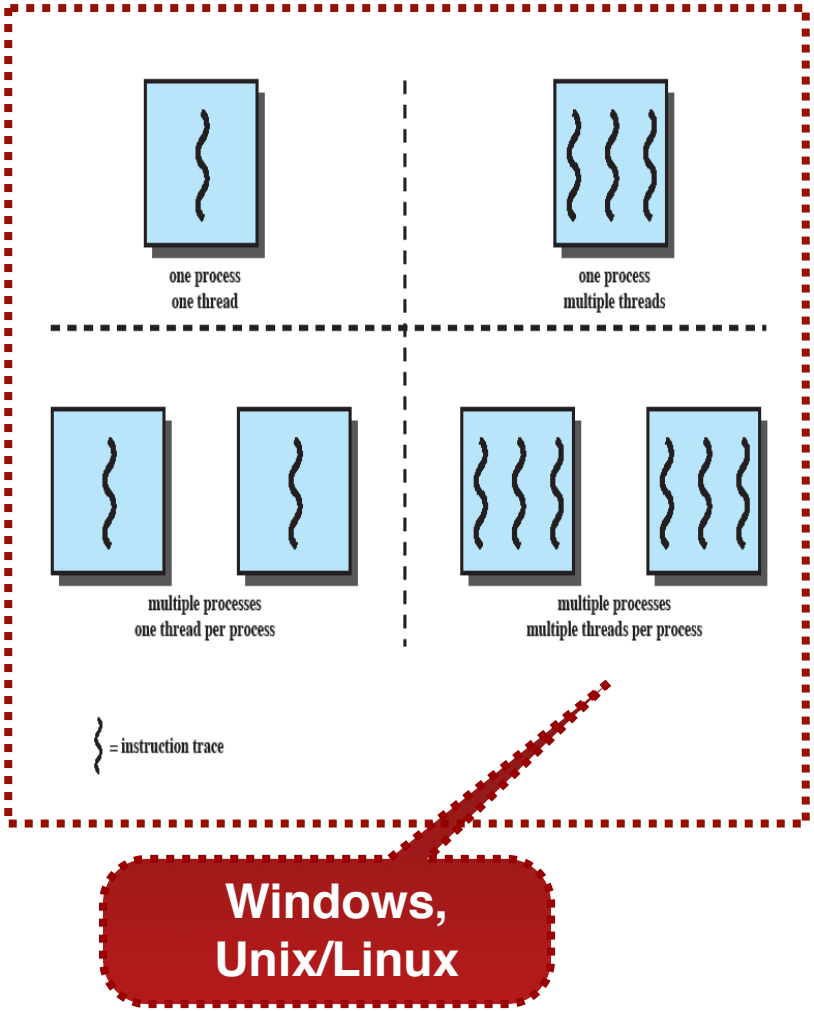
Single-Threaded Approaches

- A single thread of execution per process.
- The concept of a thread is not recognised — referred as a **single-threaded** approach.
- Example: MS-DOS



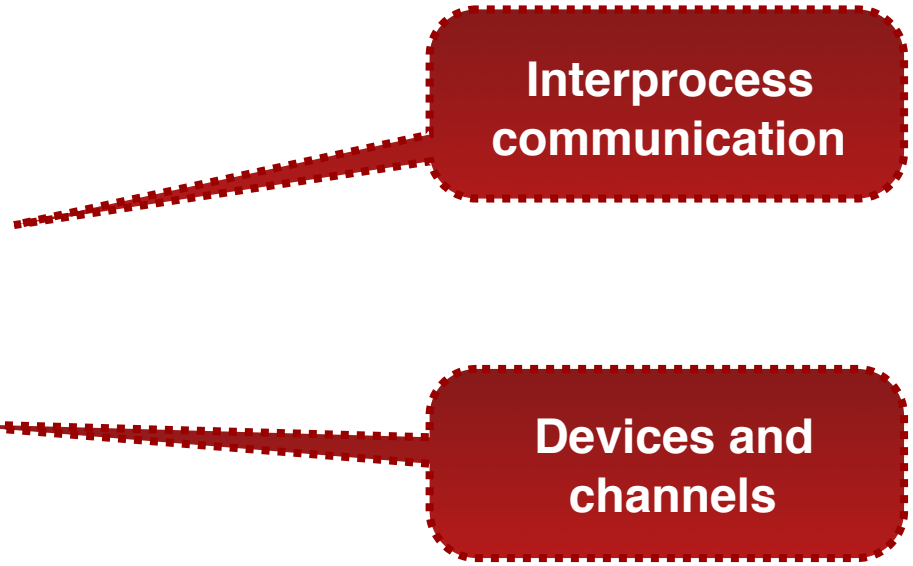
Multi-Threaded Approaches

- There right half of the figure depicts the **multi-threaded** approach.
- One process with multiple threads.
- Example: A Java run-time environment



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).
- If there is an interaction between different parts of the programs, then **concurrency control** need to be applied.
- Example: accessing and modifying a common variable — mutual exclusion need to be satisfied.

Each thread can be thought of a lightweight process and hence can be in different *states*

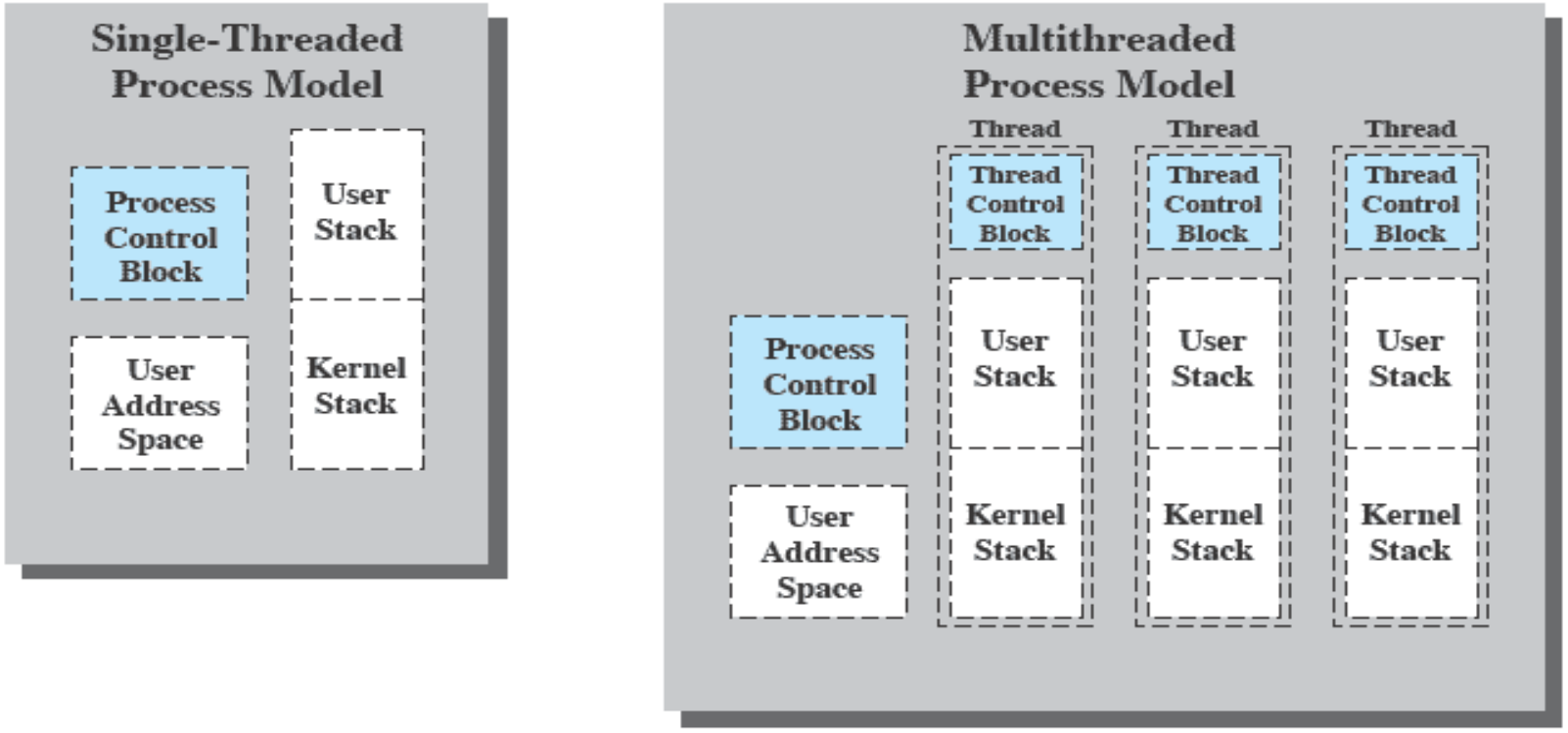
What are the attributes of a thread?

Attributes of a Thread

Each thread has:

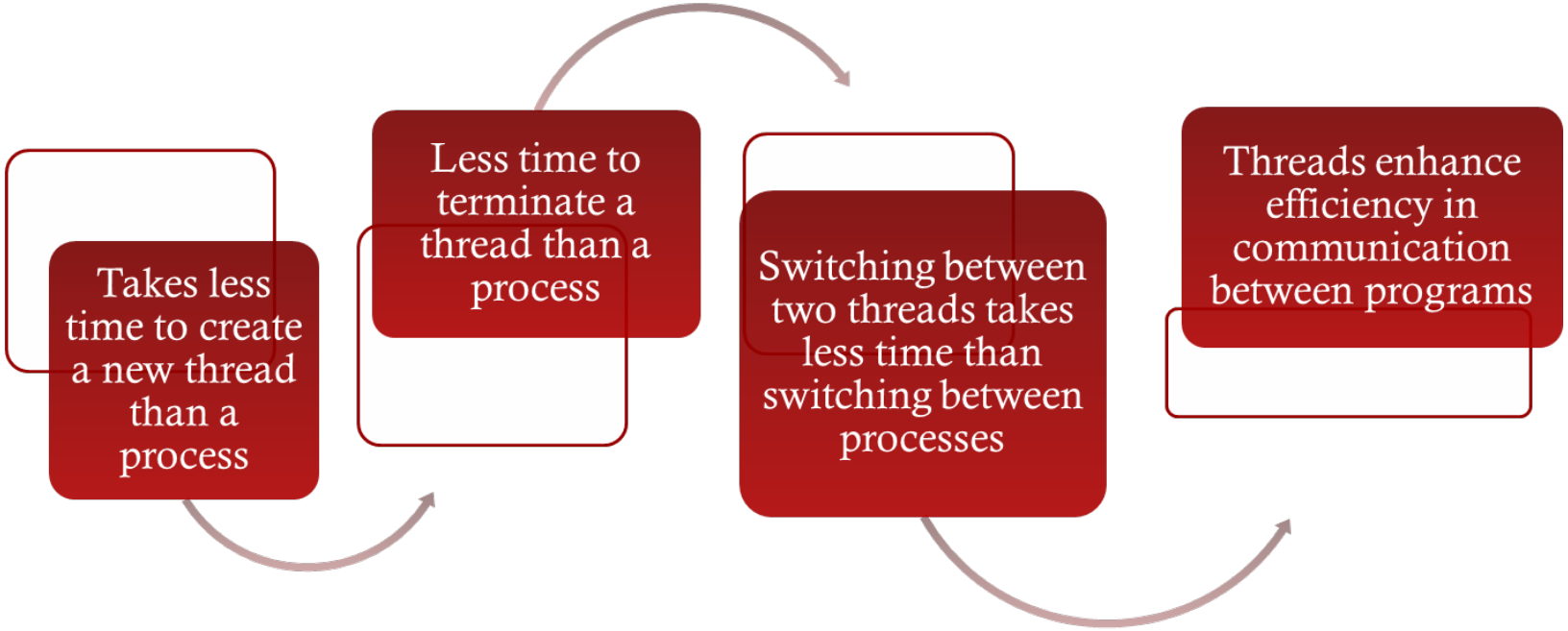
- an execution state (Running, Ready, etc.)
- saved thread context when not running
- an execution stack
- some per-thread static storage for local variables
- access to the memory and resources of its process (all threads of a process share this)

Threads vs. Processes



Single Threaded and Multithreaded Process Models

Benefits of Threads



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

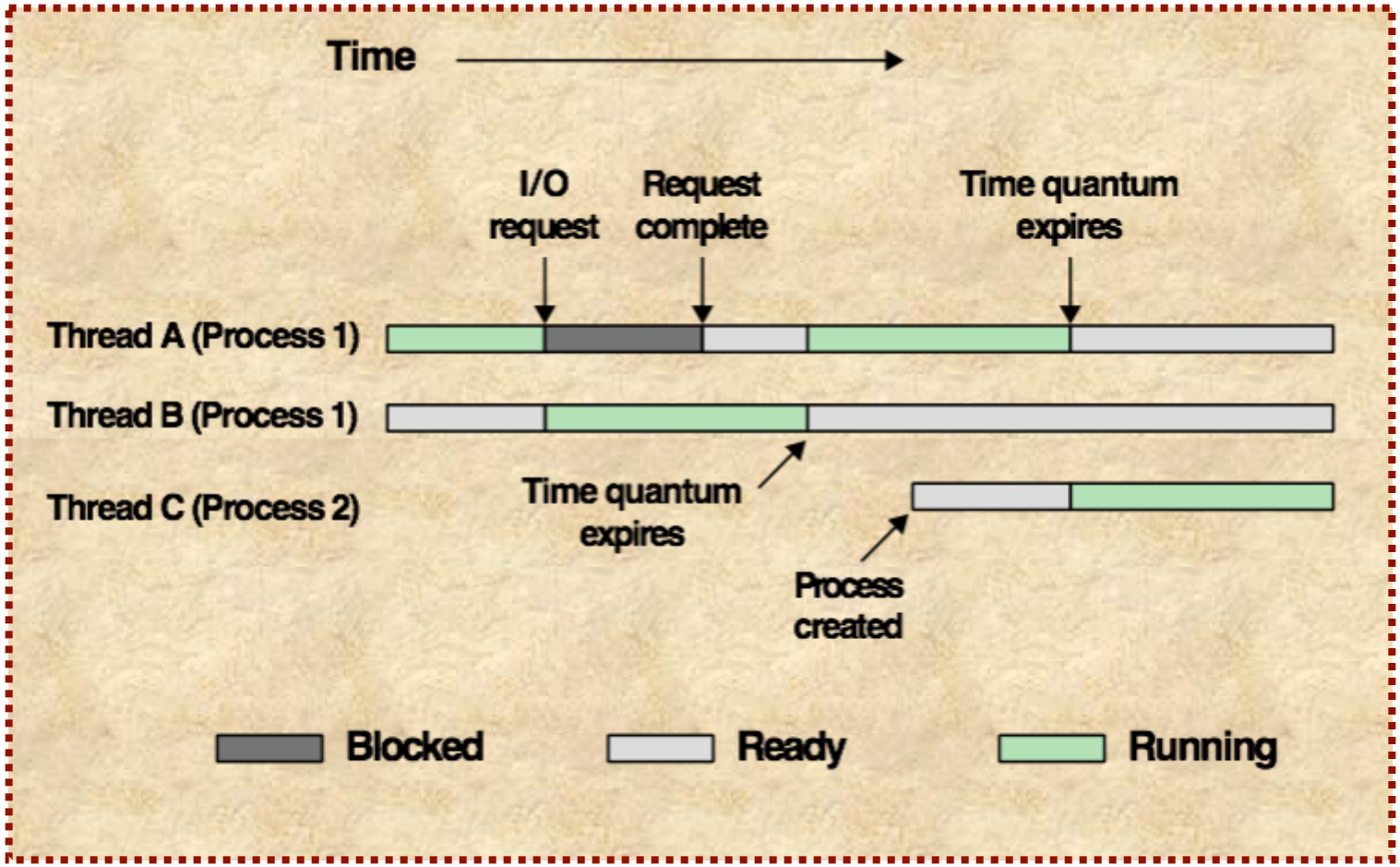
- Thread operations associated with a change in thread state:
  - Spawn
  - Block
  - Unblock
  - Finish

New threads

Wait for an event

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

Multi-threading on a Uniprocessor



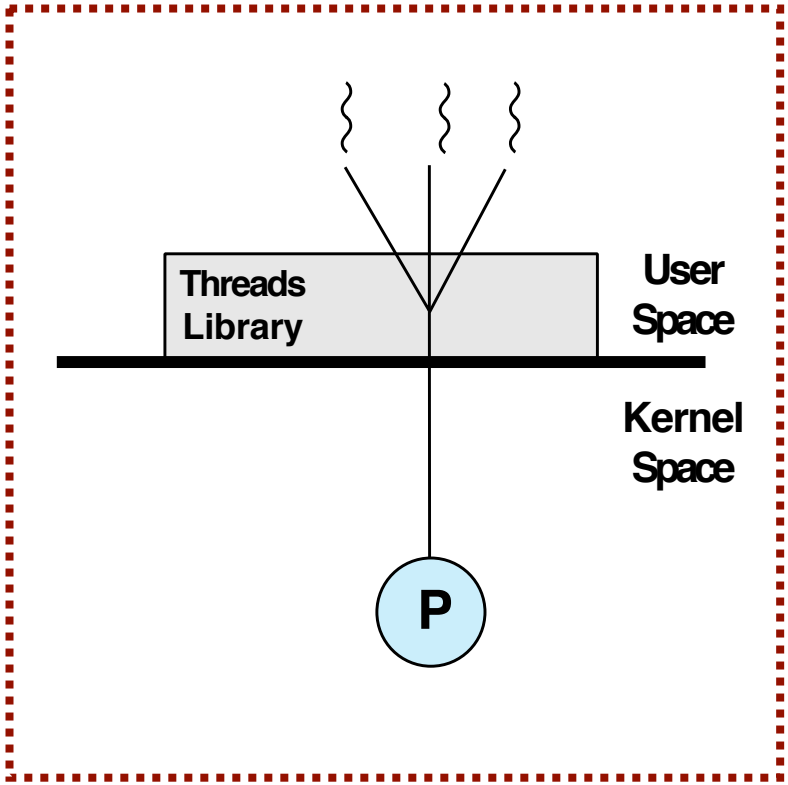
Thread Synchronisation

- It is necessary to synchronise the activities of various threads.
- All threads of a process share the same address space 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)

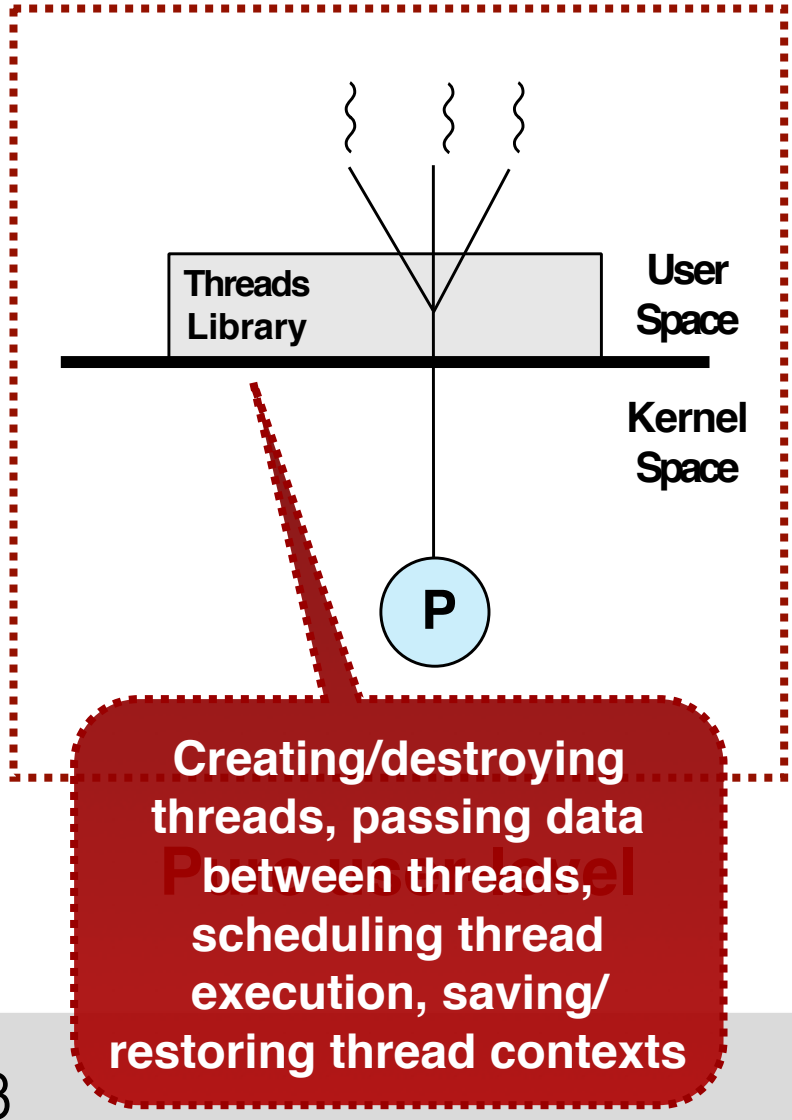
- All thread management is done by the application.
- The kernel is not aware of the existence of threads.
- Any application can be programmed to be multi-threaded by using a threads library.



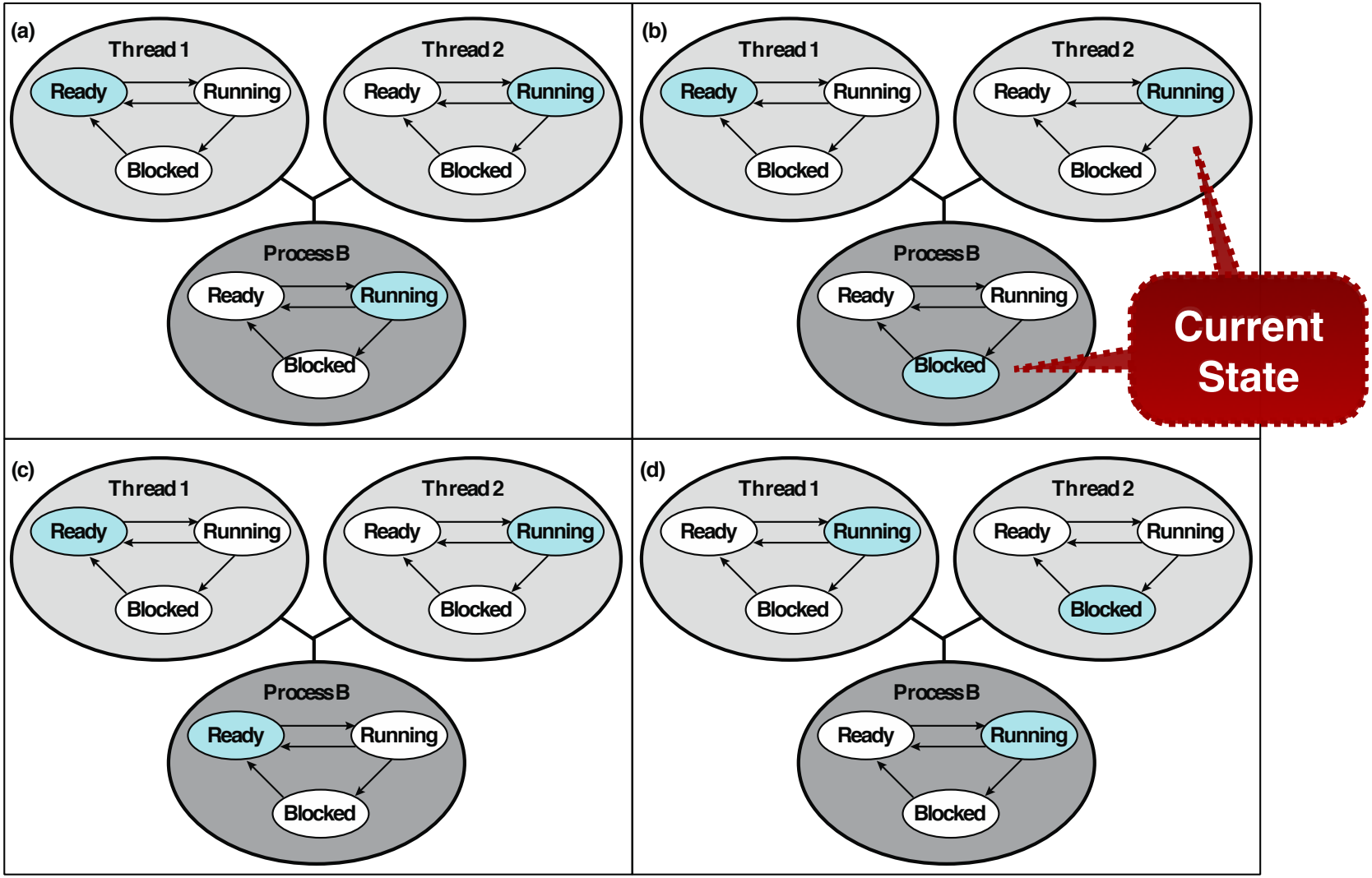
Pure user-level

User-Level Threads (KLTs)

- 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.



Thread Scheduling and Process Scheduling



ULTS: Advantages

Thread switching does not required kernel mode privileges

Scheduling can be application specific

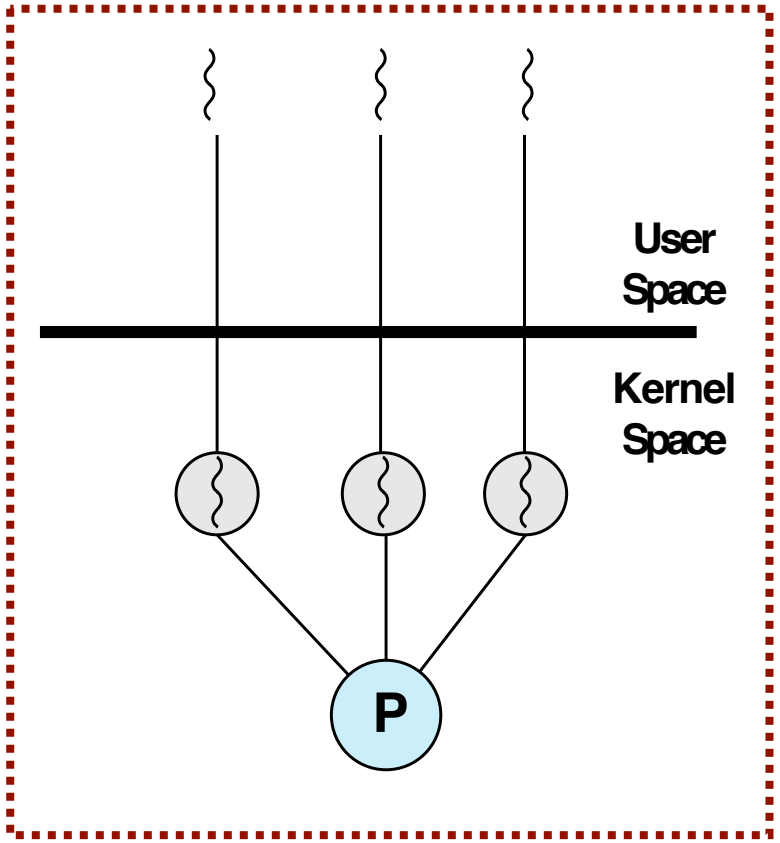
ULTs can run on any Operating System

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



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

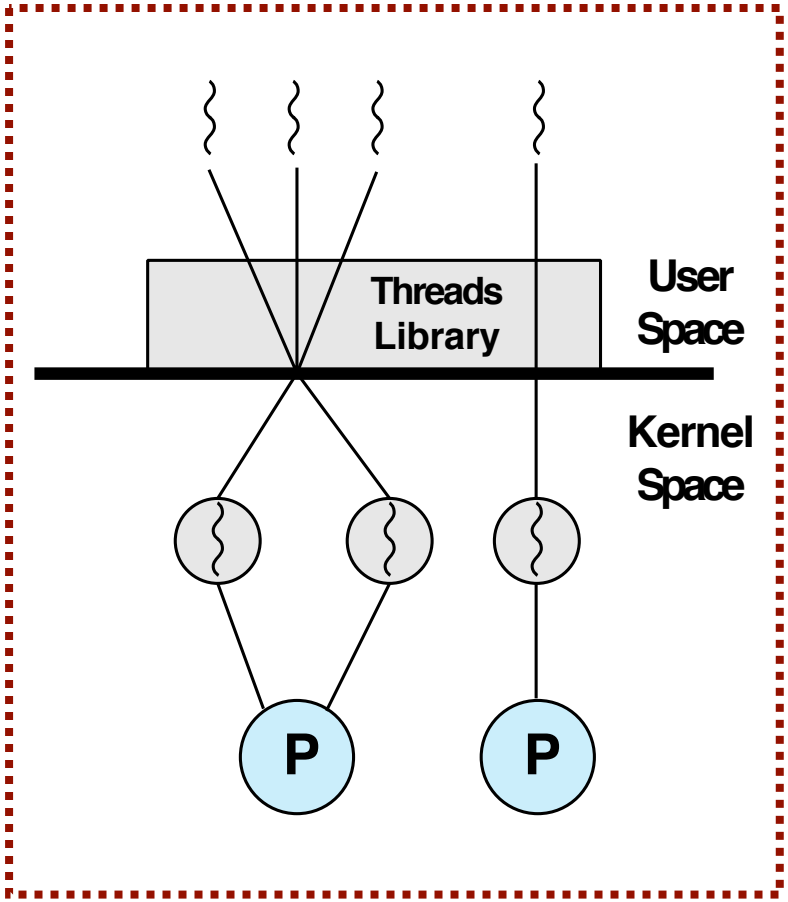
- The transfer of control from one thread to another thread within the same process requires a mode switch to the kernel.

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Thread and Process Operation Latencies (on a uniprocessor computer with Unix-like OS)

Combined Approaches

- 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)

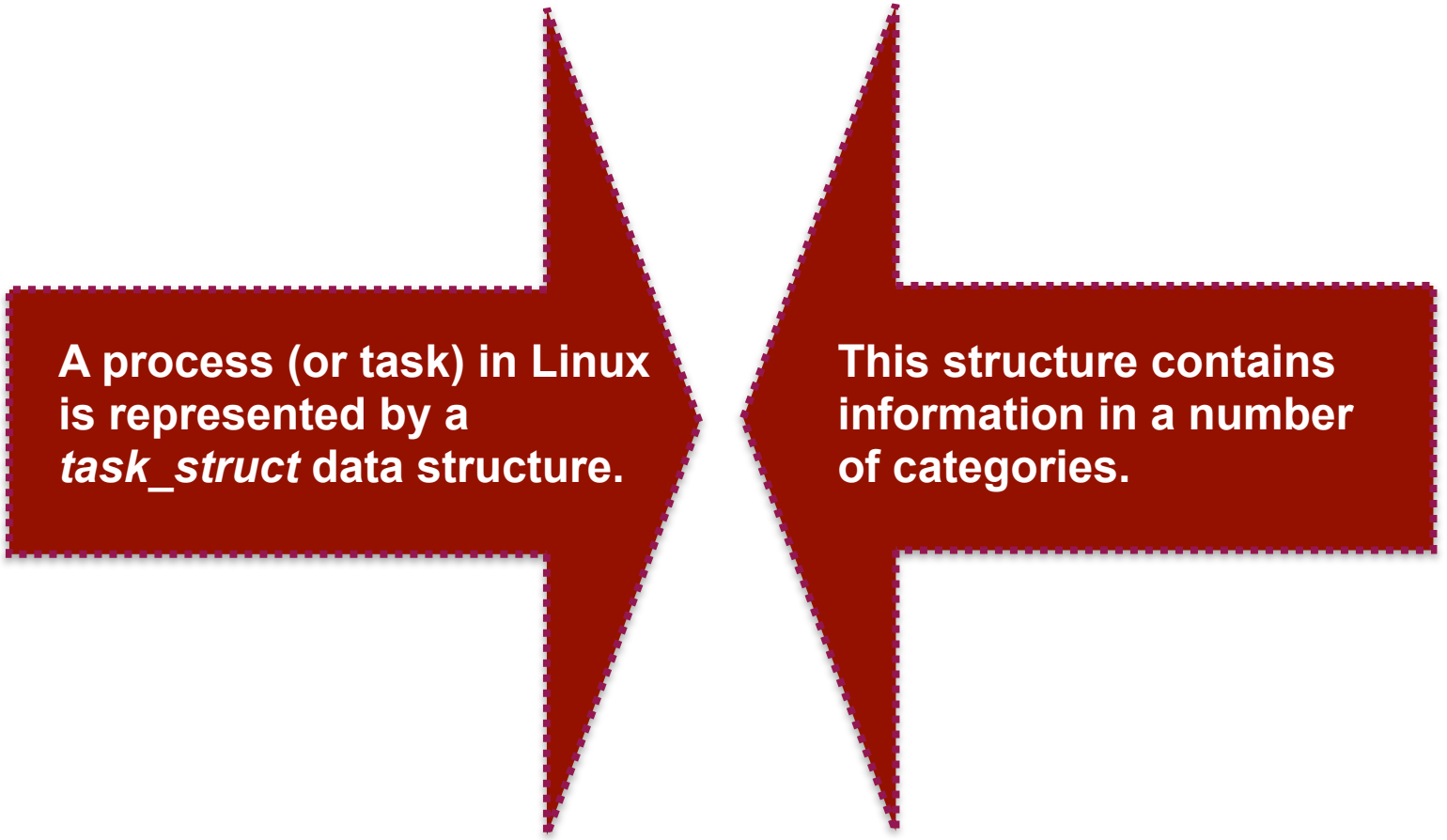


Relationships between Threads and Processes

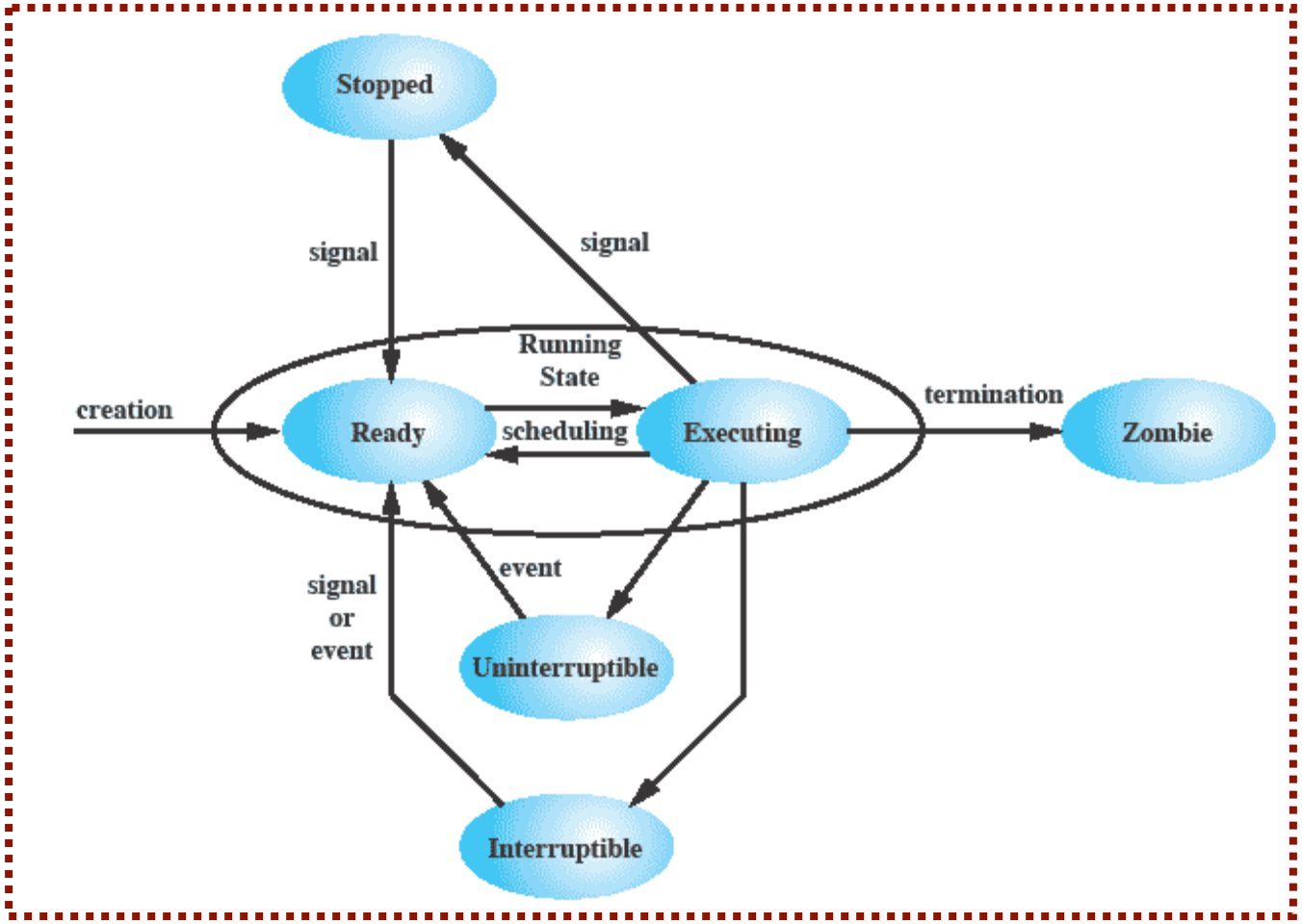
Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	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

How do Unix/Linux systems manage threads?

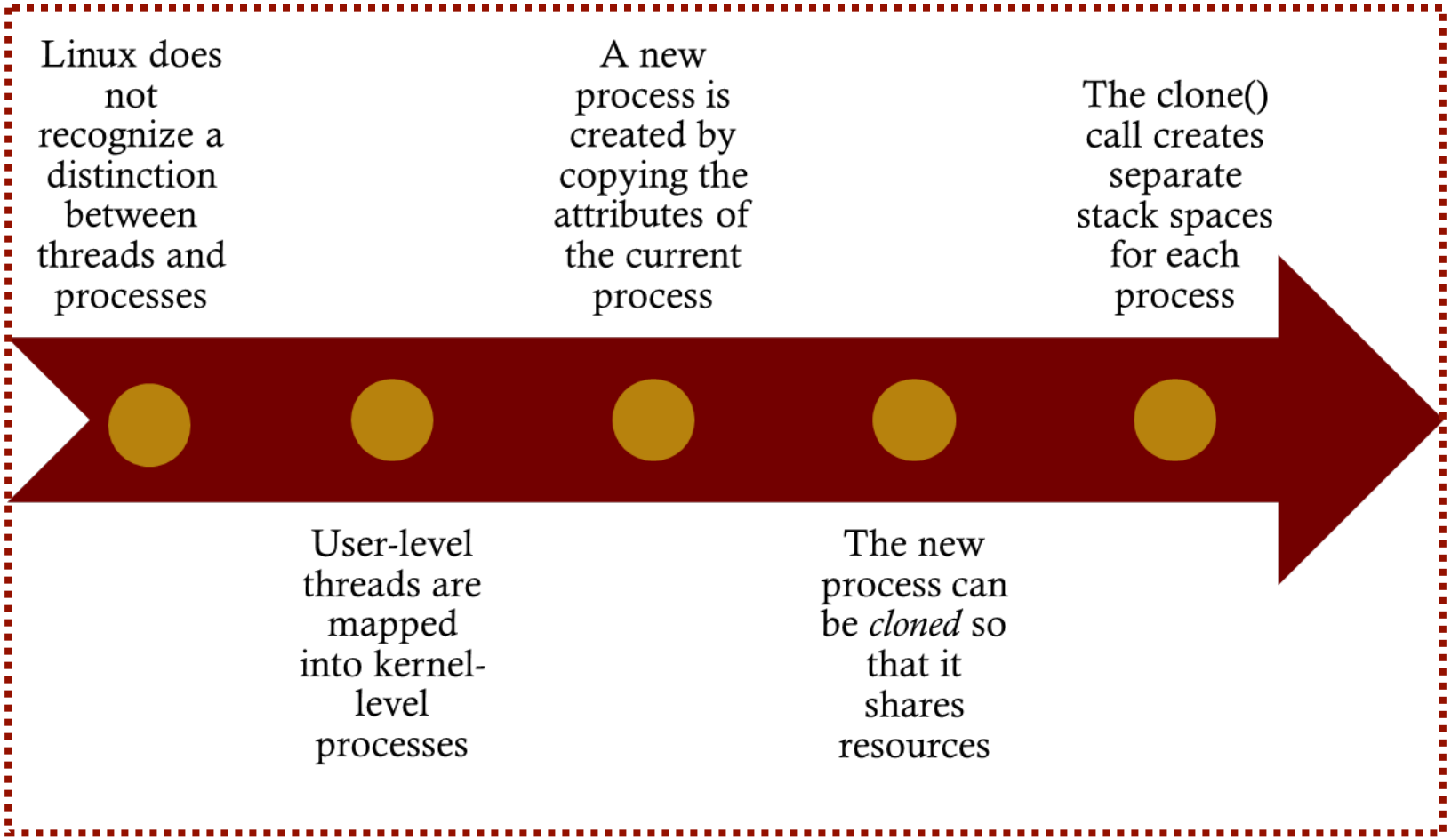
Linux Tasks


A process (or task) in Linux is represented by a *task_struct* data structure.
This structure contains information in a number of categories.

Linux: Process/Thread Model



Linux: Threads



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:

virtual machine

mnt **pid** **net** **ipc** **uts** **user**

Summary of Lecture 4 (Part 1)

- 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.