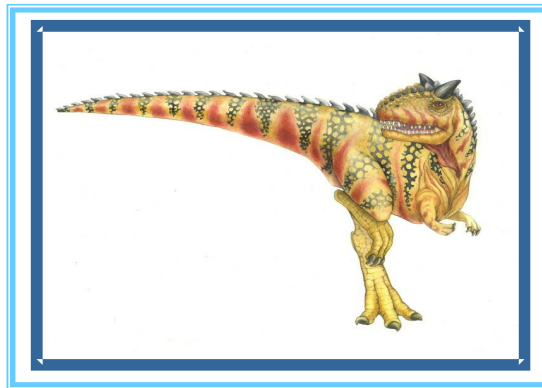# Lecture 6,7 : Thread
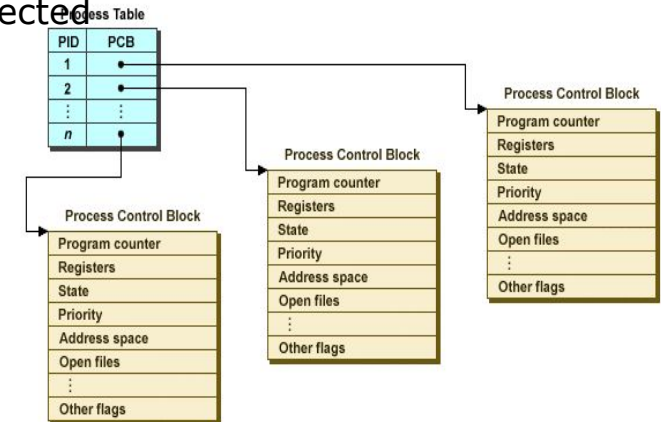
From Processes to Threads

The Soul of a Process

Shared data, has IPC, execution been affected

What is similar in cooperating processes?

They all share the same code and data (address space)

They all share the same privileges

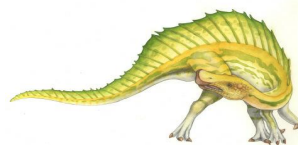They all share the same resources (files, sockets, etc.)

What don't they share?
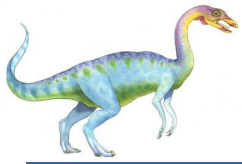
Each has its own execution state: PC, SP, and registers

Key idea: Why don't we separate the concept of a process from its execution state?

Process: address space, privileges, resources, etc.

Execution state: PC, SP, registers

Exec state also called thread of control, or thread
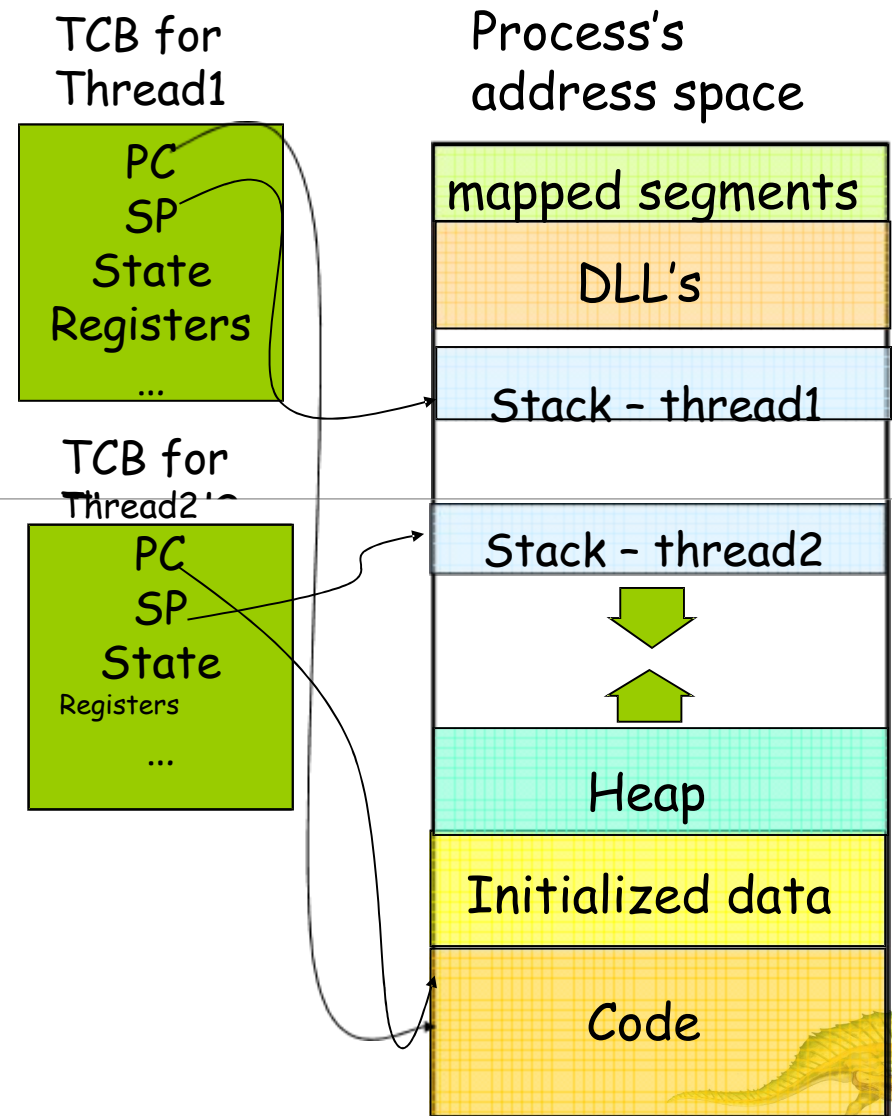
Processes and Threads

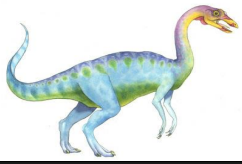Processes define an address space; threads share the address space

Process Control Block (PCB) contains process-specific information

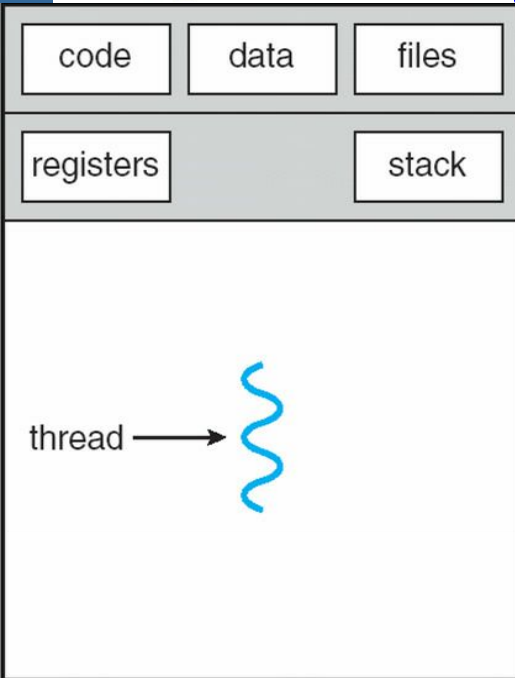Owner, PID, heap pointer, priority, active thread, and pointers to thread information

Thread Control Block (TCB) contains thread-specific information

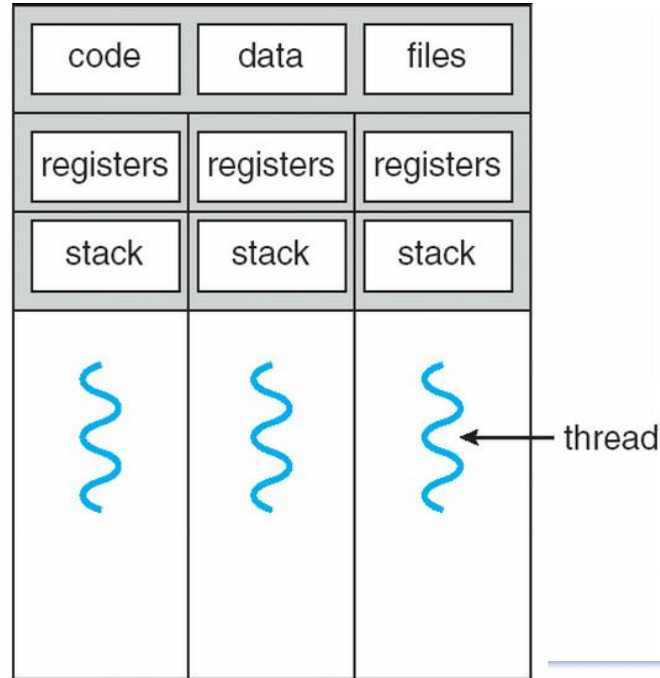Stack pointer, PC, thread state (running, …), register values, a pointer to PCB, …

TCB for Thread1

PC
SP
State
Registers
…

TCB for Thread2

PC
SP
State
Registers
…

Process's address space

mapped segments

DLL's

Stack – thread1

Stack – thread2

Heap

Initialized data

Code

Single and Multithreaded Processes

single-threaded process

multithreaded process

Thread 1	Stack (T1)
Thread 2	Stack (T2)
Thread 3	Stack (T3)
	Heap
	Static Data
PC (T2)	Code
PC (T3)	
PC (T1)	
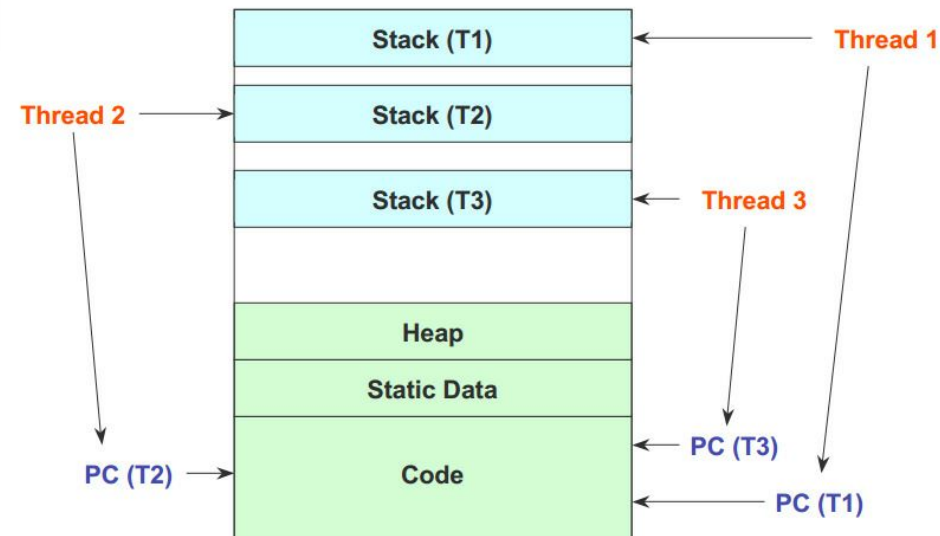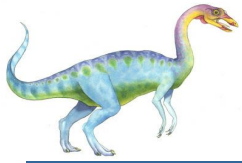
Threads Benefits

A thread represents an abstract entity that executes a sequence of instructions

- It has its own set of CPU registers

- It has its own stack

- There is no thread-specific heap or data segment (unlike process)

Threads are lightweight

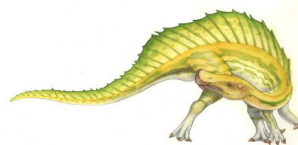- Creating a thread more efficient than creating a process.

- Communication between threads easier than processes.

- Context switching between threads requires fewer CPU cycles and memory references than switching processes.

- Threads only track a subset of process state (share list of open files, pid, …)

Examples:

- OS-supported: Windows' threads, Sun's LWP, POSIX threads

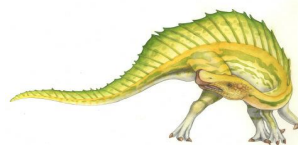# Context switch time for which entity is greater?

1. Process
2. Thread

Multithreads vs processor

	T1	C1+I1	TT=C1+I1+C2	Case1: Simple
P1			TT=C1+ Max (I1, C2)	Case2: Multithreaded process@uniprocessor
	T2	C2	TT=Max(C1+I1, C2)	Case2: Multithreaded process@multiprocessors [Not work for Many to one Model]

How Can it Help?

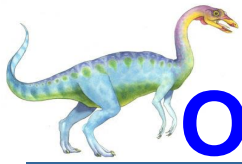Consider a Web server

Create a number of threads, and for each thread do

get network message from client

get URL data from disk

send data over network

What did we gain?

Overlapping Requests (Concurrency)

Request 1
Thread 1

Request 2
Thread 2

get network message
(URL) from client

get URL data from disk

(disk access latency)

get network message
(URL) from client

get URL data from disk

(disk access latency)

send data over network

send data over network

- Total time is less than request 1 + request 2

Time

Threads vs. Processes

Threads

A thread has no separate data segment or heap

A thread cannot live on its own, it must live within a process

There can be more than one threads in a process, the first thread calls main & has the process's stack

If a thread dies, its stack is reclaimed

Inter-thread communication via memory.

Inexpensive creation and context switch

Processes

- A process has code/data/heap & other segments

- There must be at least one thread in a process

- Threads within **a process share** code/data/heap, share I/O, but each has its own stack & registers

- If a process dies, its resources are reclaimed & all threads die

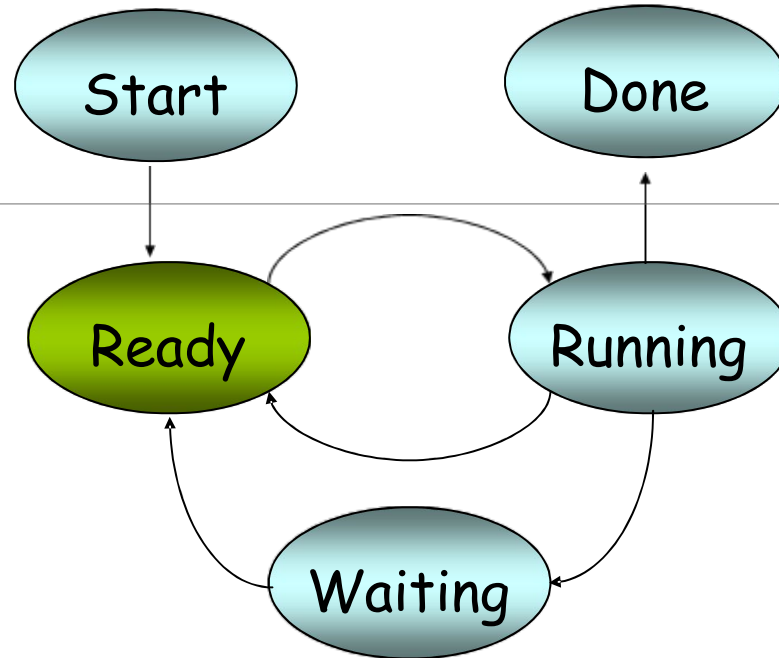- Inter-process communication via OS /data copying/message passing.
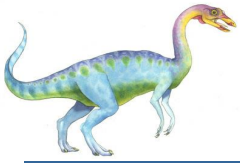
- Expensive creation and context switch

Threads' Life Cycle

Threads (just like processes) go through a sequence of *start*, *ready*, *running*, *waiting*, and *done* states

```
        Start                    Done

        Ready                  Running

                  Waiting
```

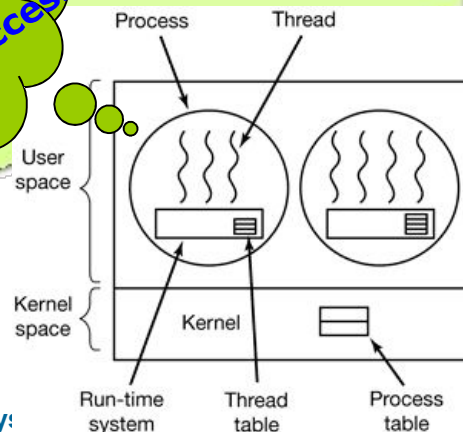# Threads have the same scheduling states as processes

1. True
2. False

Implementing Threads

User Level

- Threads package entirely in user space
- Kernel knows nothing about threads
- Fast to create and switch-scheduler as local procedure
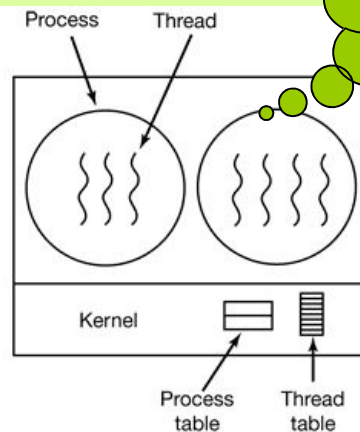
. Blocking System Call . CPU access

Windows NT Windows 2000

Kernel Level

- No runtime system
- Global Thread table, updated by kernel call
- Do not block process for systemcall
- Thread switching: same or another process
- Not so fast as runtime system
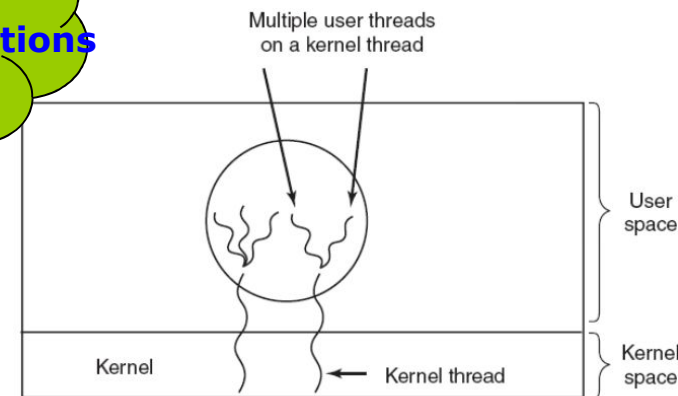
Thread Recycling Thread pool
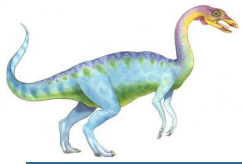
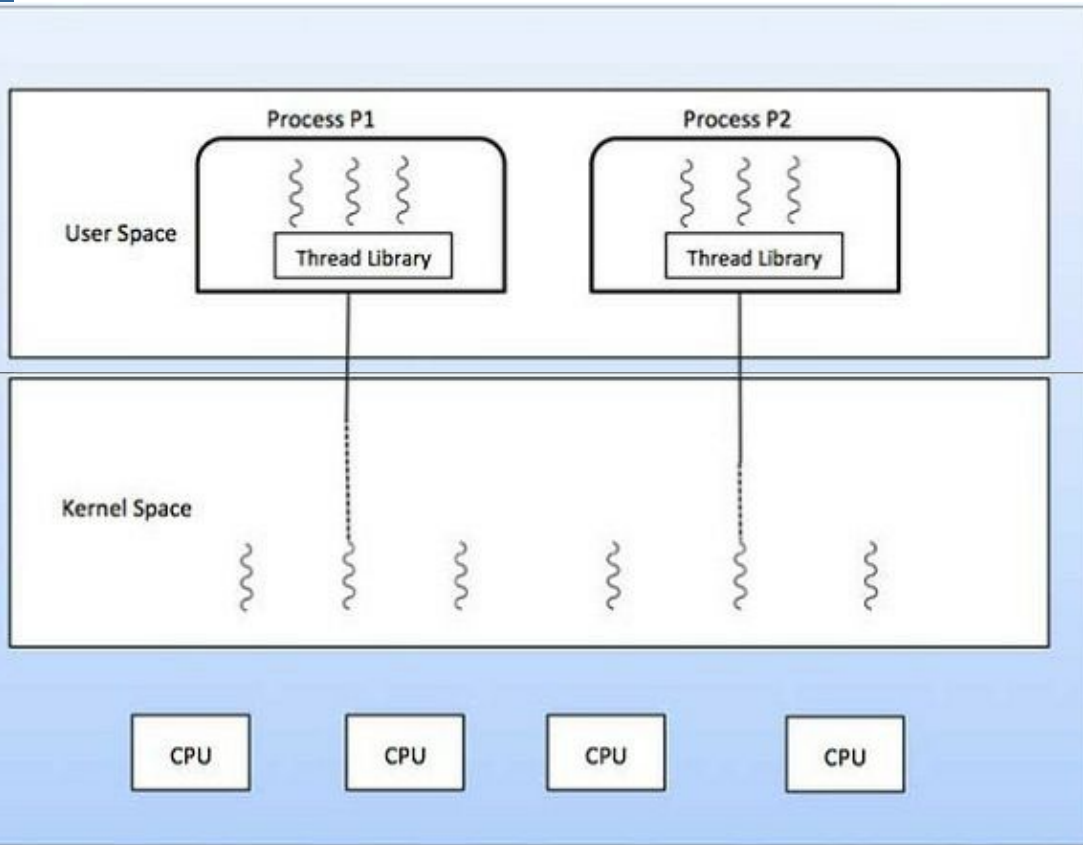**Frequent thread operations Need many System calls**

Solaris

Hybrid

- Use kernel-level threads and then multiplex user-level threads onto same or all kernel threads.

Process   Thread

User space

Kernel space   Kernel

Run-time system   Thread table   Process table

Process   Thread

User space

Kernel space   Kernel

Process table   Thread table

Multiple user threads on a kernel thread

User space

Kernel space   Kernel   Kernel thread

Multithreading Models

Many to one Model
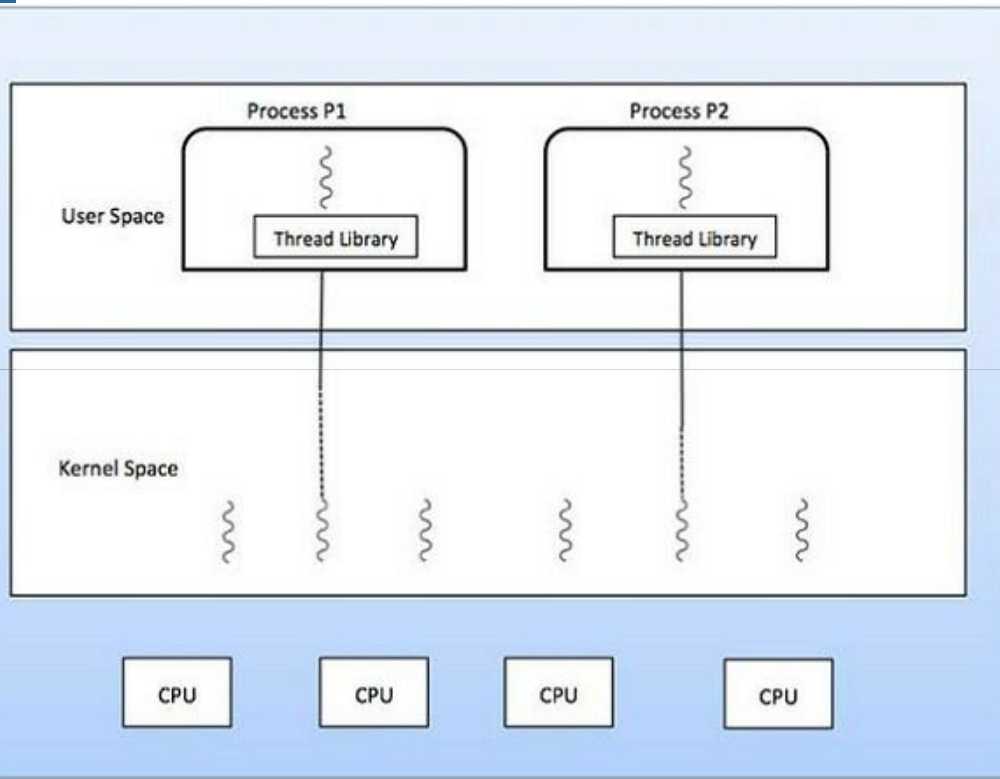

- Thread management is done in user space.

- Entire process is block if a thread makes blocking system call.

- No used in multi processors system. No concurrency.

- Green threads – Solaris thread library.

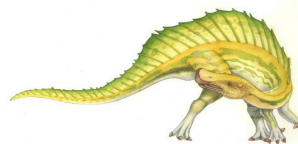https://www.tutorialspoint.com/operating_system/os_multi_threading.htm
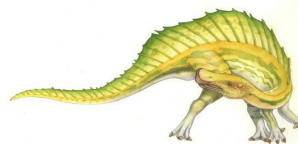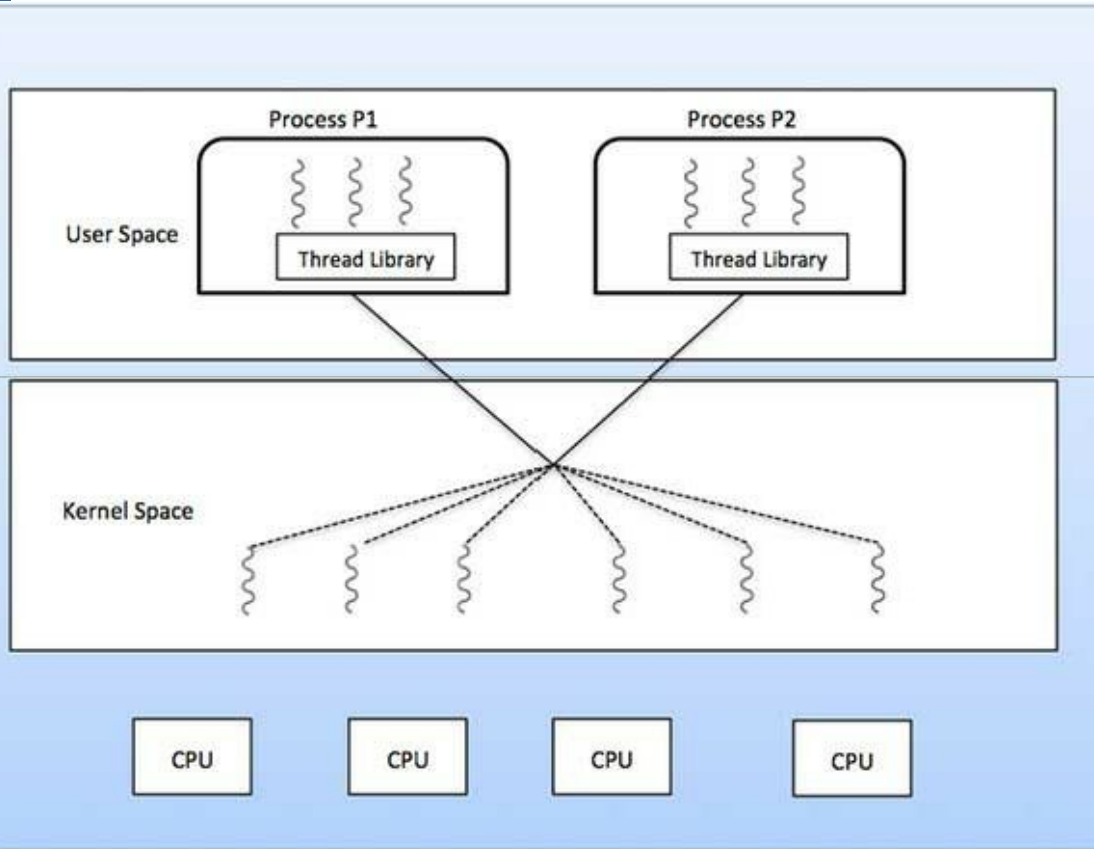
Multithreading Models

One to one Model

•Provide more concurrency than many to one.

•another thread will run when a thread makes a blocking system call.

•Multiple threads to run in parallel on microprocessors.

•Drawback: creating a user thread, requires to create a kernel thread.

•Restriction has # User (or kernel) threads are supported by the system.

•Windows NT, Windows 2000

Multithreading Models

Many to Many Model

https://www.tutorialspoint.com/operating_system/os_multi_threading.htm

- Many users level threads to a Greater or equal number of kernel threads.

- Solve one-to-one, Many-to-one Model's restrictions.

- Users can create as many user threads as need
  - corresponding kernel threads can run parallel @ multiprocessor

- Solaris 2, IRIX, HP-UX
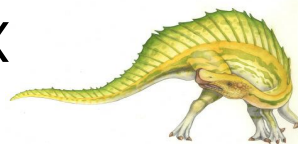
Course materials:
Galvin book   5.1-5.2,5.3.1

End of Lecture 6,7