Chap. 4) Multithreaded Programming

경희대학교 컴퓨터공학과

이 승 룡

Processes

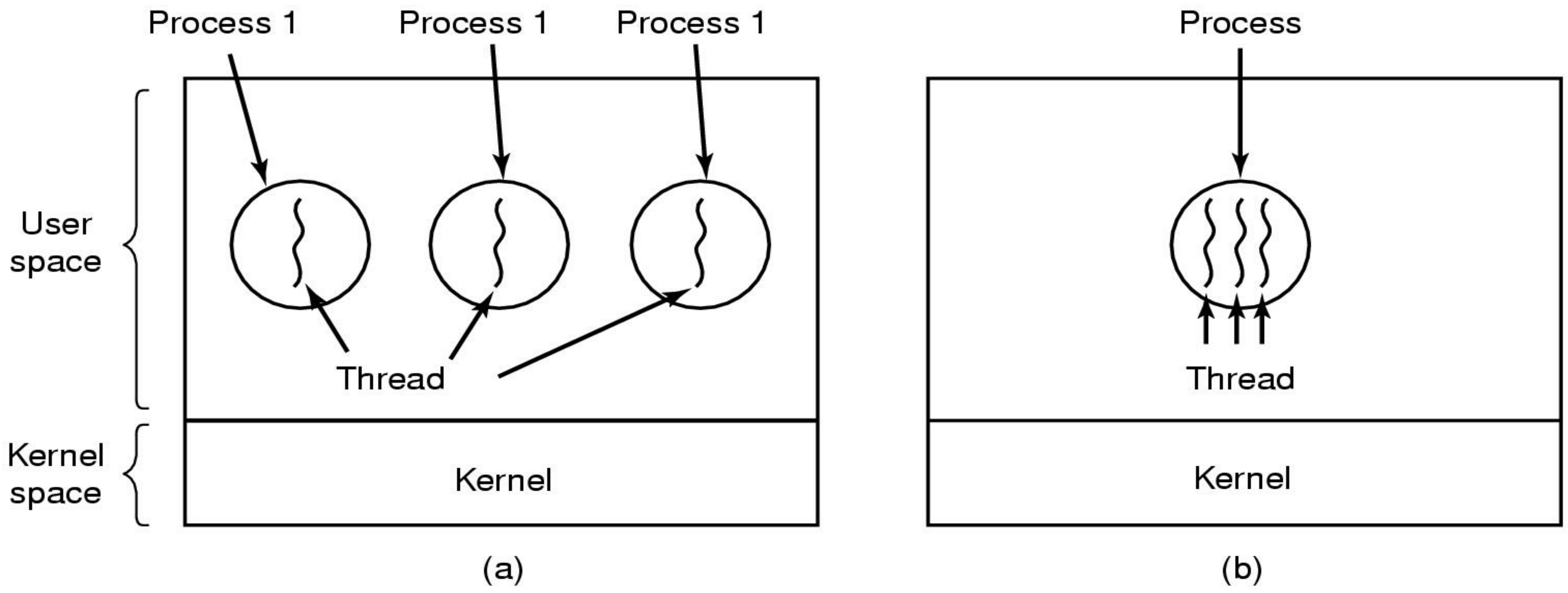
- Heavy-weight
  - ✓ A process includes many things:
    - An address space (all the code and data pages)
    - OS resources (e.g., open files) and accounting info.
    - Hardware execution state (PC, SP, registers, etc.)
  - ✓ Creating a new process is costly because all of the data structures must be allocated and initialized
    - Linux: over 100 fields in task_struct

      (excluding page tables, etc.)
  - ✓ Inter-process communication is costly, since it must usually go through the OS
    - Overhead of system calls and copying data

Thread Concept: Key Idea

- Separate the concept of a process from its execution state
  - ✓ Process: address space, resources, other general process attributes (e.g., privileges)
  - ✓ Execution state: PC, SP, registers, etc.
  - ✓ This execution state is usually called
    - a thread of control,
    - a thread, or
    - a lightweight process (LWP)

Thread Concept: Key Idea (Cont'd)



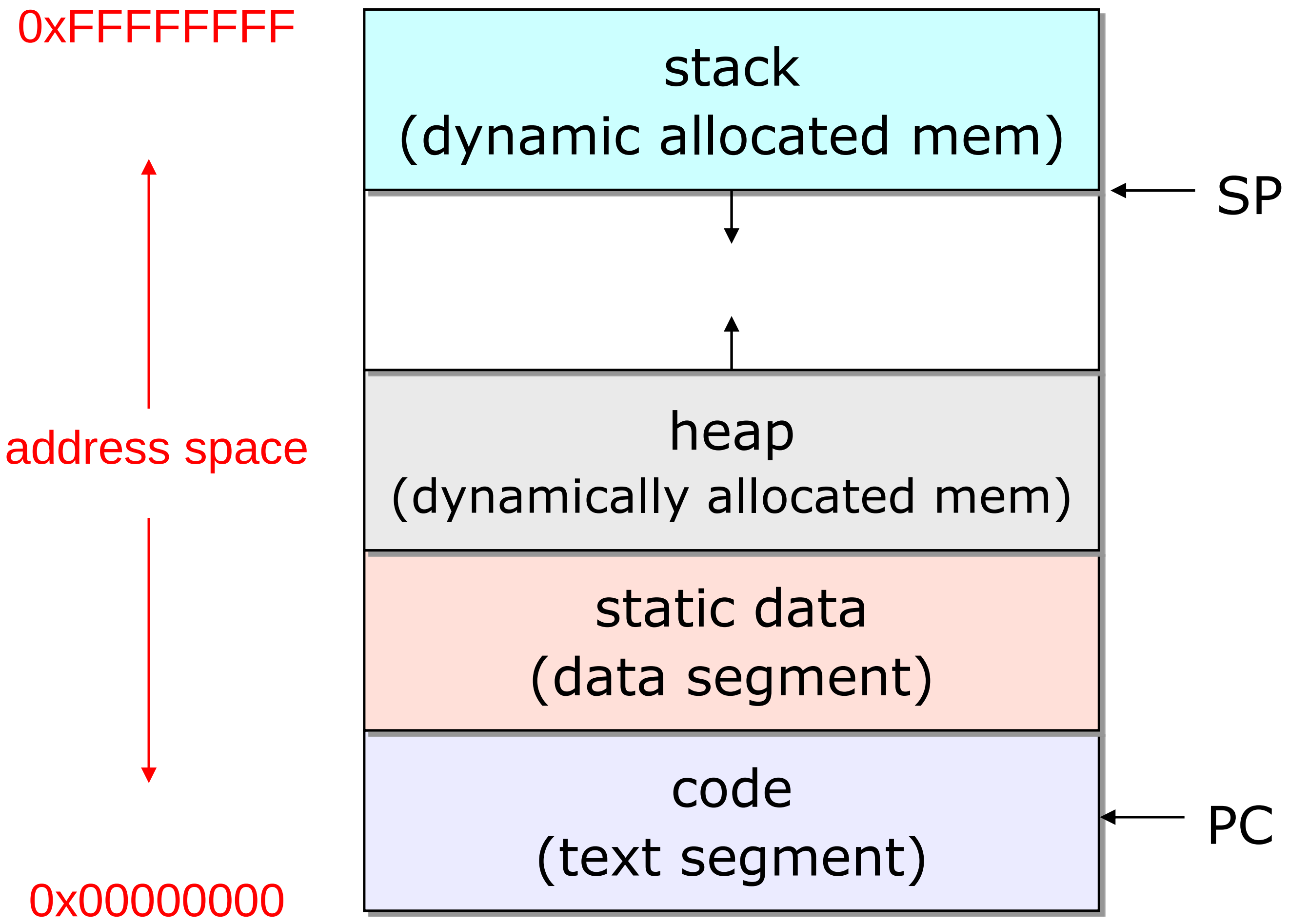
What is a Thread?

- A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:
  - ✓ program counter
  - ✓ register set
  - ✓ stack space
- A thread shares with its peer threads its:
  - ✓ code section
  - ✓ data section
  - ✓ operating-system resources
  - ✓ collectively known as a *task* or process
- A traditional or *heavyweight* process is equal to a task with one thread

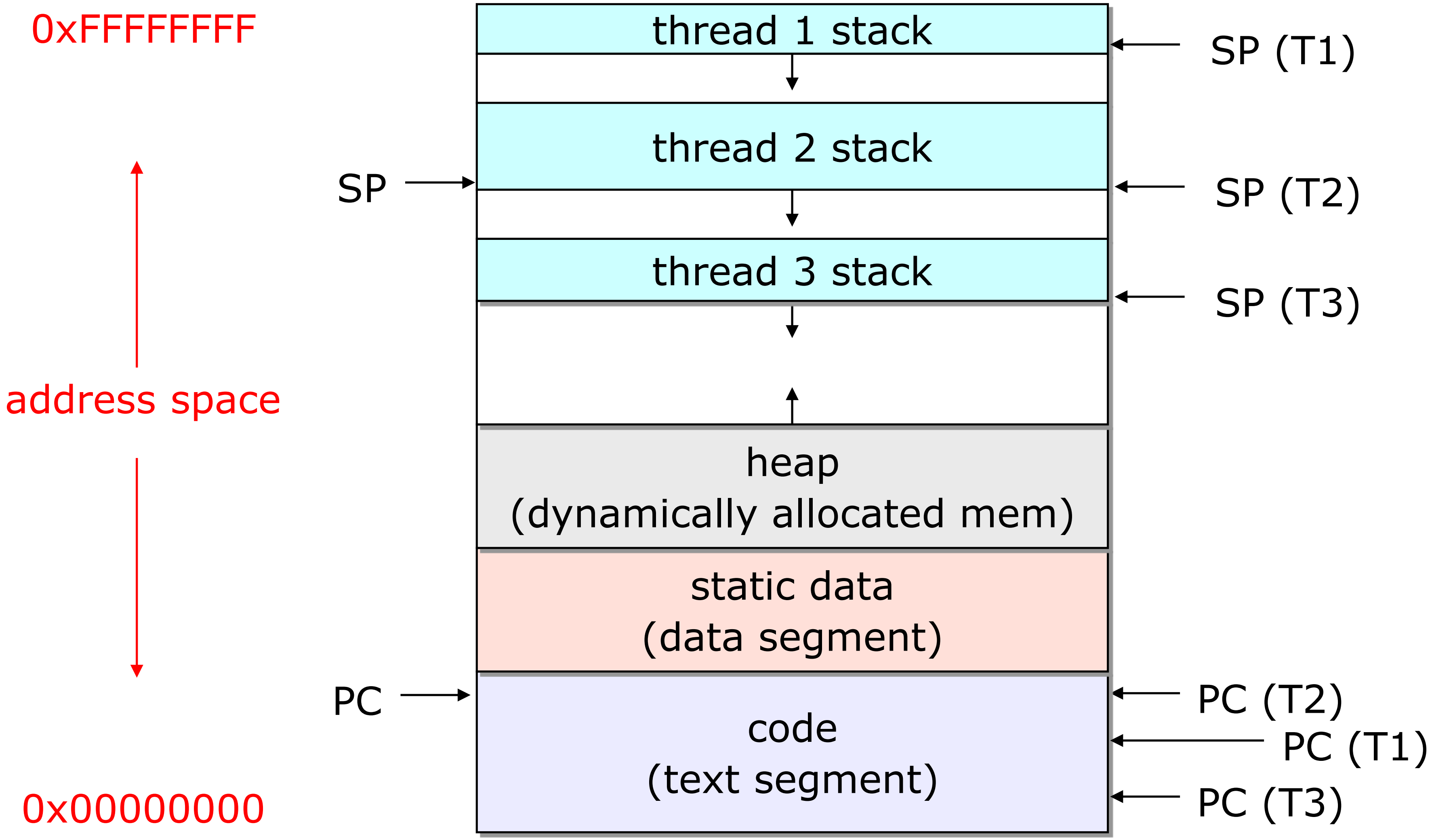
Process vs. Thread

- Processes vs. Threads
  - ✓ A thread is bound to a single process
  - ✓ A process, however, can have multiple threads
  - ✓ Sharing data between threads is cheap: all see the same address space
  - ✓ Threads become the unit of scheduling
  - ✓ Processes are now containers in which threads execute
  - ✓ Processes become static, threads are the dynamic entities

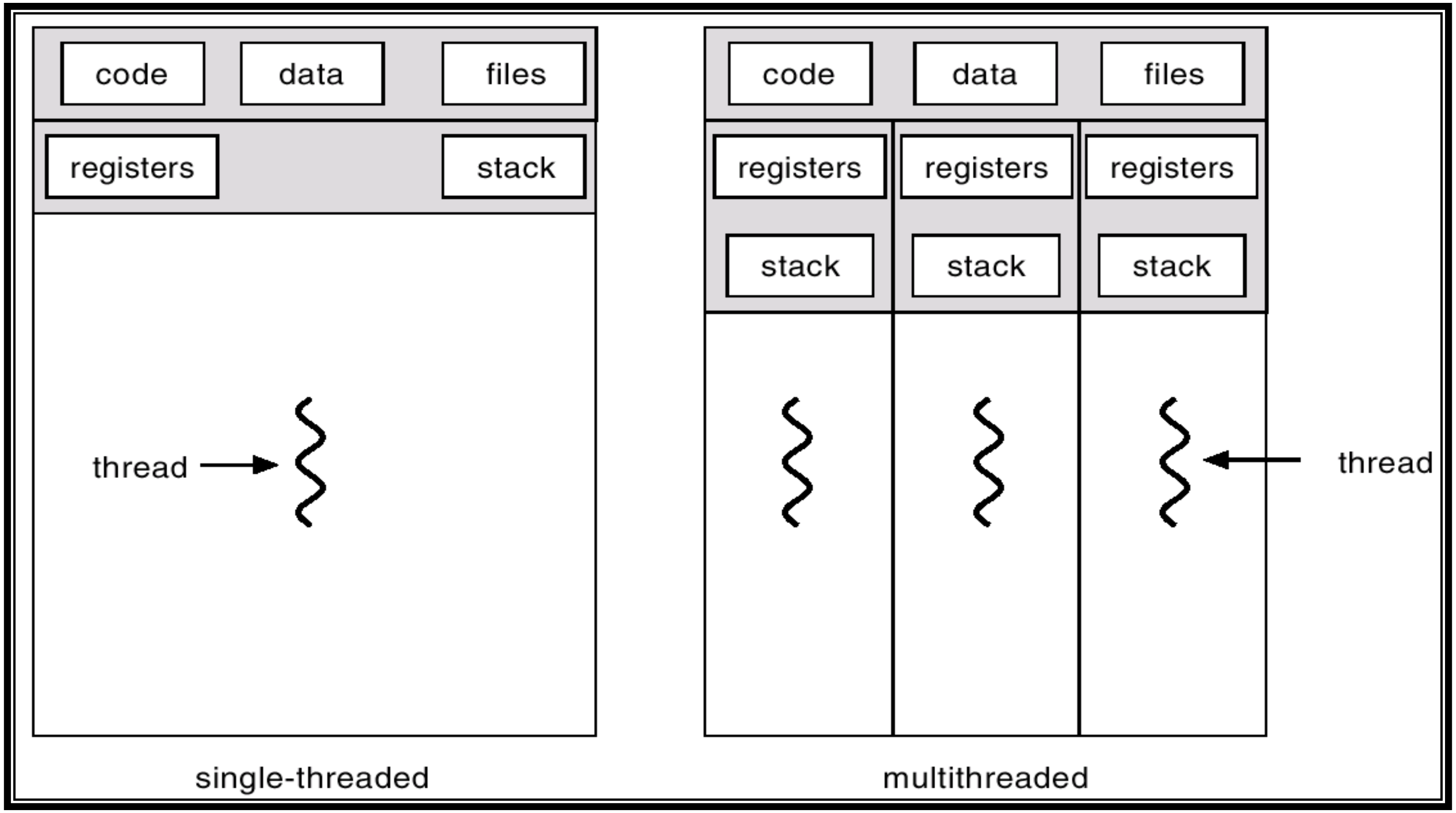
Process Address Space



Address Space with Threads



Single and Multithreaded Processes



Concurrent Servers: Processes

- Web server example
  - ✓ Using fork() to create new processes to handle requests in parallel is overkill for such a simple task.

```
While (1) {
    int sock = accept();
    if ((pid = fork()) == 0) {
        /* Handle client request */
    } else {
        /* Close socket */
    }
}
```

Concurrent Servers: Threads

- Using threads
  - ✓ We can create a new thread for each request

```
webserver ()
{
    While (1) {
        int sock = accept();
        thread_fork (handle_request, sock);
    }
}
handle_request (int sock)
{
    /* Process request */
    close (sock);
}
```

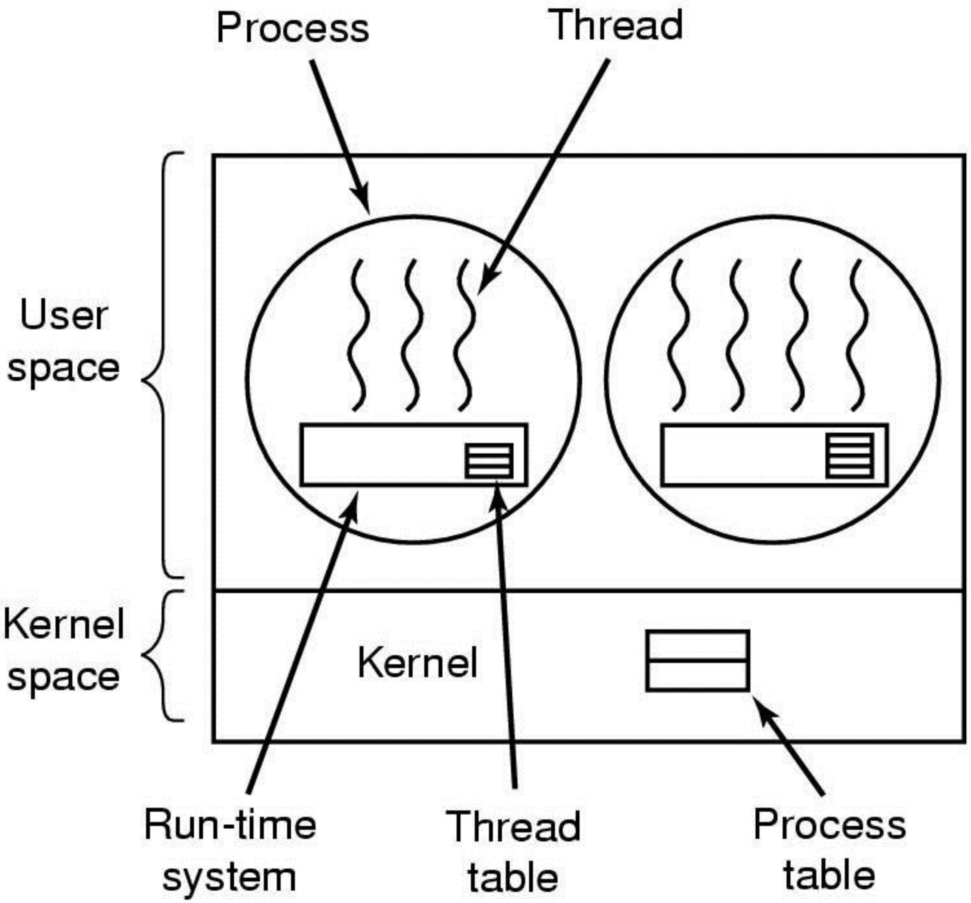
Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

User Threads

- Thread management done by user-level threads library

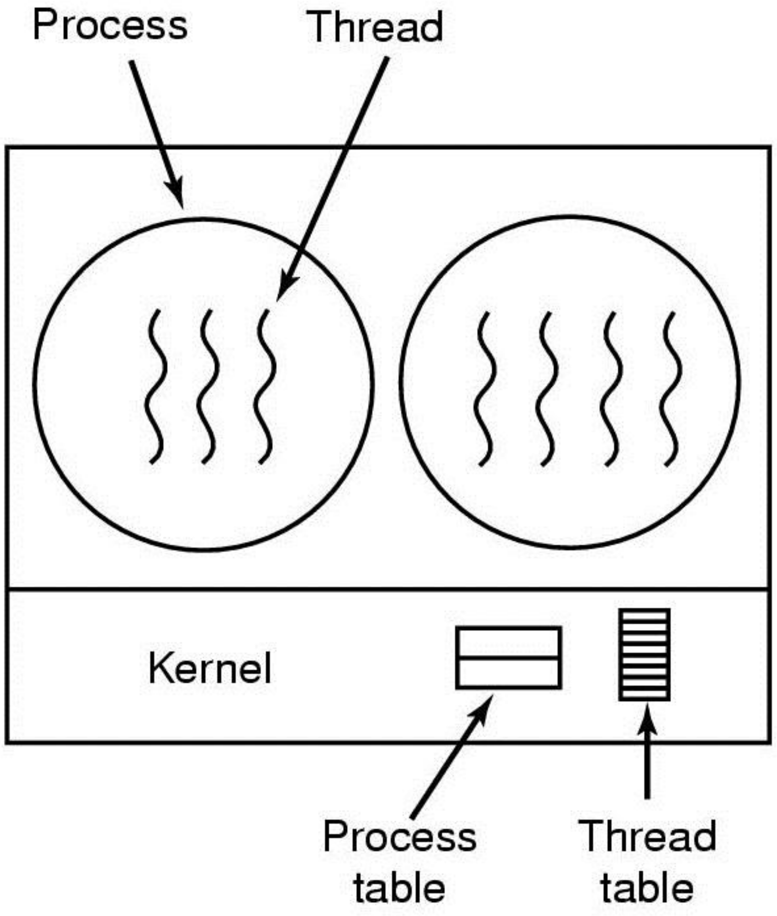
- Examples
  - ✓ POSIX *Pthreads*
  - ✓ Mach *C-threads*
  - ✓ Solaris *threads*



Kernel Threads

- Supported by the Kernel
  - ✓ thread creation and management requires system calls

- Examples
  - ✓ Windows 95/98/NT/2000
  - ✓ Solaris
  - ✓ Tru64 UNIX
  - ✓ BeOS
  - ✓ Linux



User-level Threads vs. Kernel-level Threads

- User-level threads
  - ✓ The user-level threads library implements thread operations
  - ✓ They are small and fast
  - ✓ User-level threads are invisible to the OS
  - ✓ OS may make poor decisions
    - e.g. blocking I/O
  - ✓ Thread scheduling
    - Non-preemptive scheduling: yield()
    - Preemptive scheduling: timer through signal
- Kernel-level threads
  - ✓ All thread operations are implemented in the kernel
  - ✓ The OS schedules all of the threads in a system
  - ✓ Kernel threads are cheaper than processes
  - ✓ They can still be too expensive

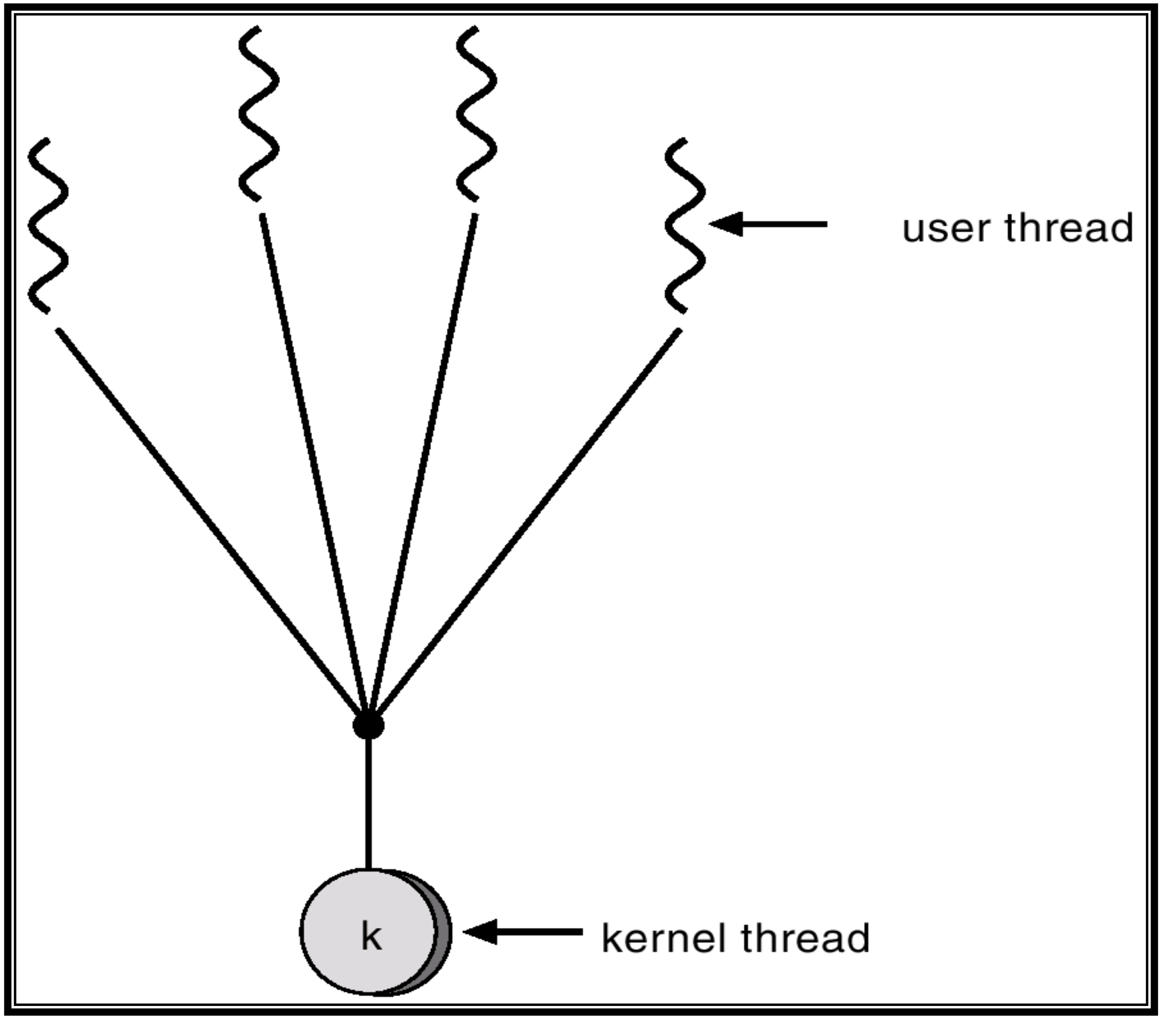
Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One

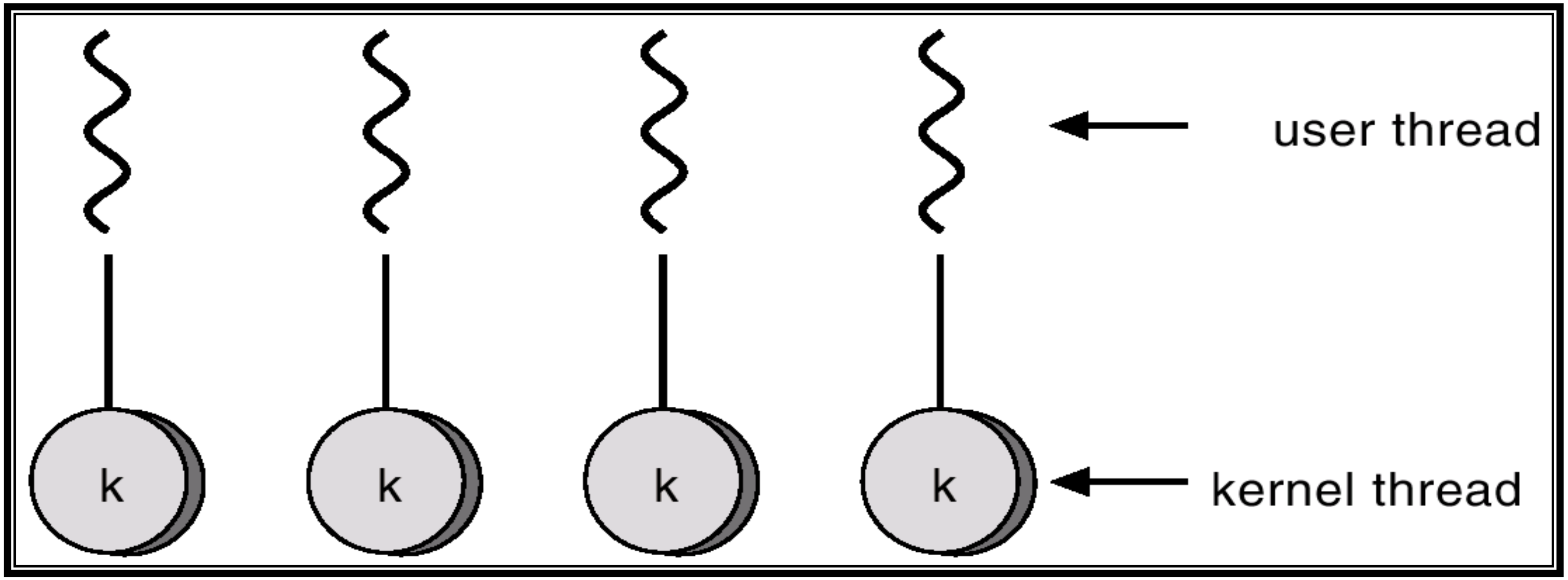
- Many user-level threads mapped to single kernel thread

- Used on systems that do not support kernel threads



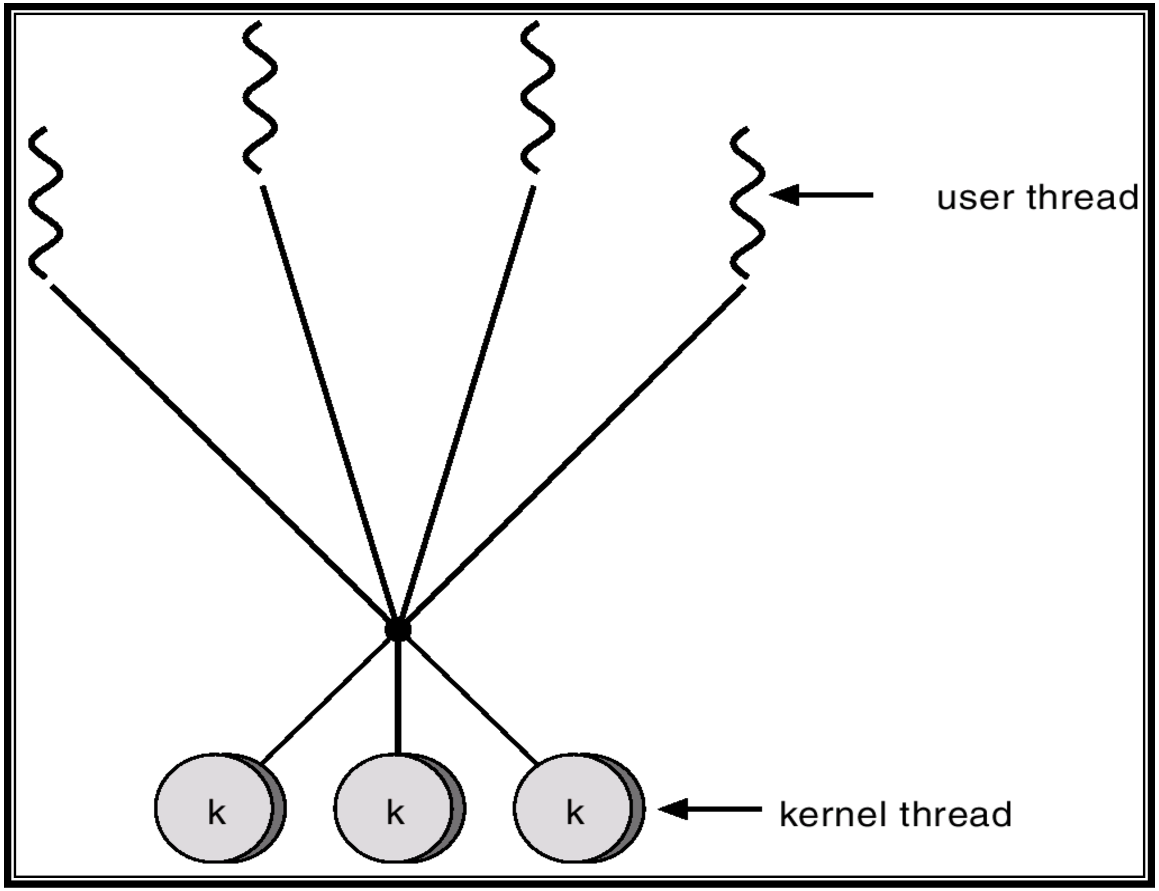
One-to-One

- Each user-level thread maps to kernel thread
- Examples
  - ✓ Windows 95/98/NT/2000/XP
  - ✓ OS/2



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 2
- Windows NT/2000/XP with the *ThreadFiber* package



Threading Issues

- Semantics of fork() and exec() system calls
  - ✓ Two versions of fork()
- Thread cancellation
  - ✓ Asynchronous cancellation
  - ✓ Deferred cancellation
- Signal handling
  - ✓ To the thread to which the signal applies
  - ✓ To every thread in the process
  - ✓ To certain threads in the process
  - ✓ Assign a specific thread to receive all signals for the process
- Thread pools
  - ✓ Create a number of threads at process startup
- Thread specific data

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems

Threads Interface

- POSIX-style threads
  - ✓ Pthreads
  - ✓ DCE threads (early version of Pthreads)
  - ✓ Unix International (UI) threads (Solaris threads)
    - Sun Solaris 2, SCO Unixware 2

- Microsoft-style threads
  - ✓ Win32 threads
    - Microsoft Windows 98/NT/2000/XP
  - ✓ OS/2 threads
    - IBM OS/2

Pthreads

- Thread creation/termination

```
int pthread_create (pthread_t *tid,
                    pthread_attr_t *attr,
                    void *(start_routine)(void *),
                    void *arg);
```

```
void pthread_exit   (void *retval);
```

```
int pthread_join     (pthread_t tid,
                      void **thread_return);
```

Pthreads (Cont’d)

- Mutexes

```
int pthread_mutex_init
                         (pthread_mutex_t *mutex,
                          const pthread_mutexattr_t *mattr);
```

```
int pthread_mutex_destroy
                         (pthread_mutex_t *mutex);
```

```
int pthread_mutex_lock
                         (pthread_mutex_t *mutex);
```

```
int pthread_mutex_unlock
                         (pthread_mutex_t *mutex);
```

Pthreads (Cont'd)

- Condition variables

```
int pthread_cond_init
                   (pthread_cond_t *cond,
                    const pthread_condattr_t *cattr);
```

```
int pthread_cond_destroy
                   (pthread_cond_t *cond);
```

```
int pthread_cond_wait
                   (pthread_cond_t *cond,
                    pthread_mutex_t *mutex);
```

```
int pthread_cond_signal
                   (pthread_cond_t *cond);
```

```
int pthread_cond_broadcast
                   (pthread_cond_t *cond);
```

Win32 Threads

- Thread creation/termination

```
HANDLE CreateThread (lpThreadAttributes, dwStackSize,
                     lpStartAddress, lpParameter,
                     dwCreationFlags, lpThreadId);
```

```
void ExitThread   (dwExitCode);
```

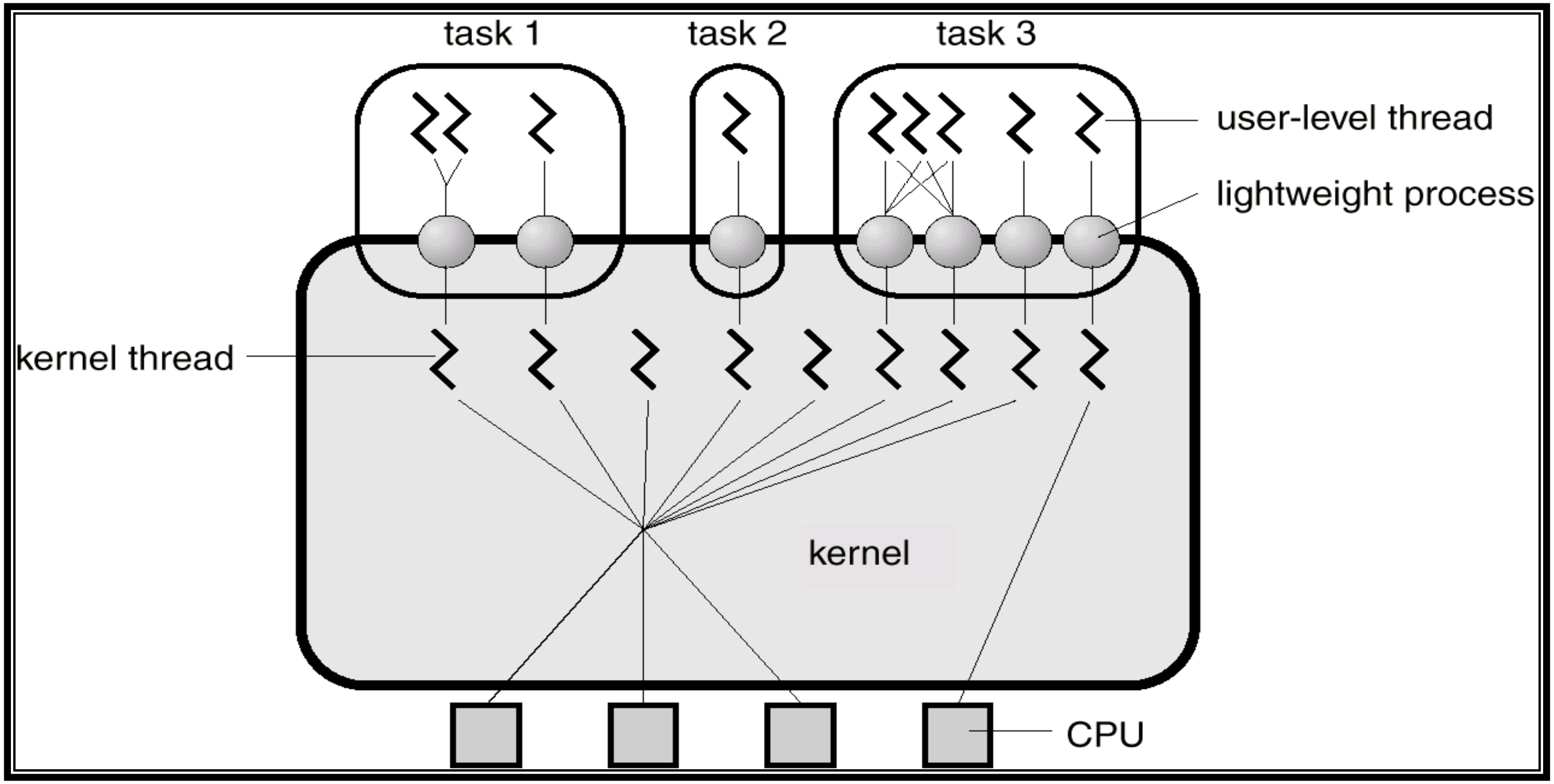
Java Threads

- Thread creation/termination

Create a new class derived from **`Thread`** class
Override run() method

Create a new class that implements the **`runnable`** interface

Solaris 2 Threads



Solaris 2 Threads

- LWP (Lightweight Process)
  - ✓ A virtual CPU for executing code or system calls
  - ✓ Each process contains at least one LWP
  - ✓ Each LWP is connected to exactly one kernel-level thread
  - ✓ Each LWP is separately dispatched by the kernel, may
    - perform independent system calls
    - incur independent page faults
    - run in parallel on a multiprocessor, etc.
  - ✓ The thread library dynamically adjusts the number of LWPs in the pool to ensure the best performance for the application
  - ✓ It also “ages” LWPs and deletes them when they are unused for a long time.
  - ✓ An LWP is a kernel data structure

- *For implementing many-to-many model*

Windows XP Threads

- Implements the one-to-one mapping

- Each thread contains
  - ✓ a thread id
  - ✓ register set
  - ✓ separate user and kernel stacks
  - ✓ private data storage area

- Cf) Fibers
  - ✓ Fibers are often called “lightweight” threads
  - ✓ Fibers are invisible to the kernel
  - ✓ Fibers provide a functionality of the many-to-many model

Linux Threads

- Linux 2.4 introduces a concept of “thread groups”
  - ✓ *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)
  - ✓ So, there exist POSIX compatibility problems

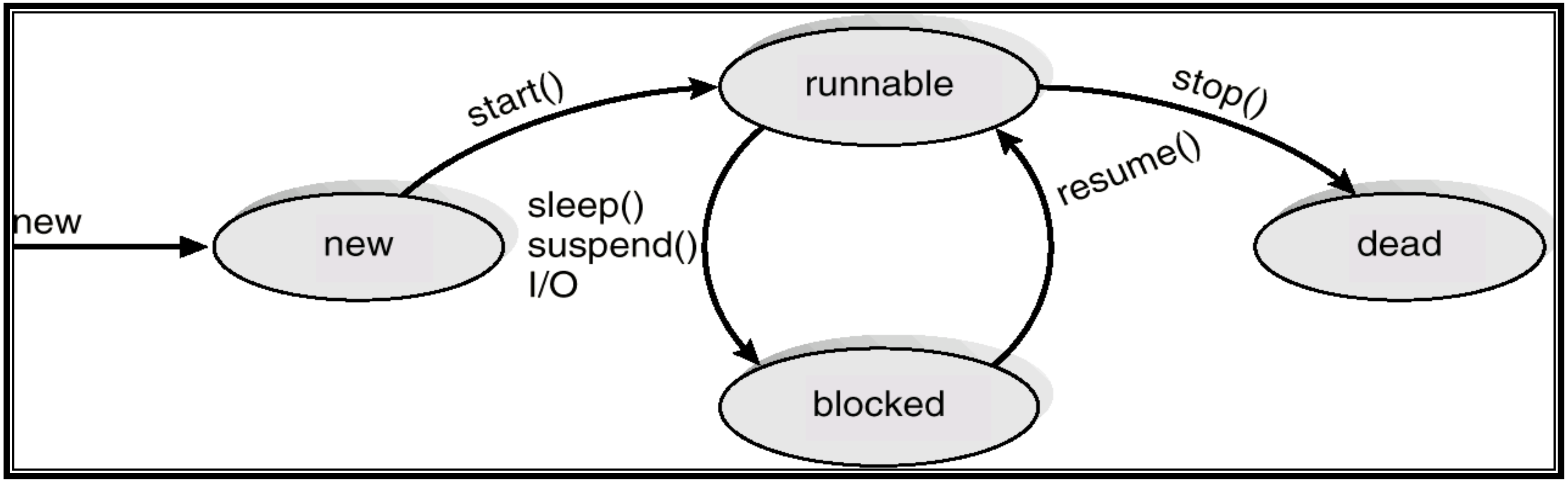
- Approaches for POSIX compliance
  - ✓ NPTL (Native POSIX Threading Library) – by RedHat
    - 1:1 model
  - ✓ NGPT (Next Generation POSIX Threading) – by IBM
    - M:N model → Linux 2.6

Java Threads

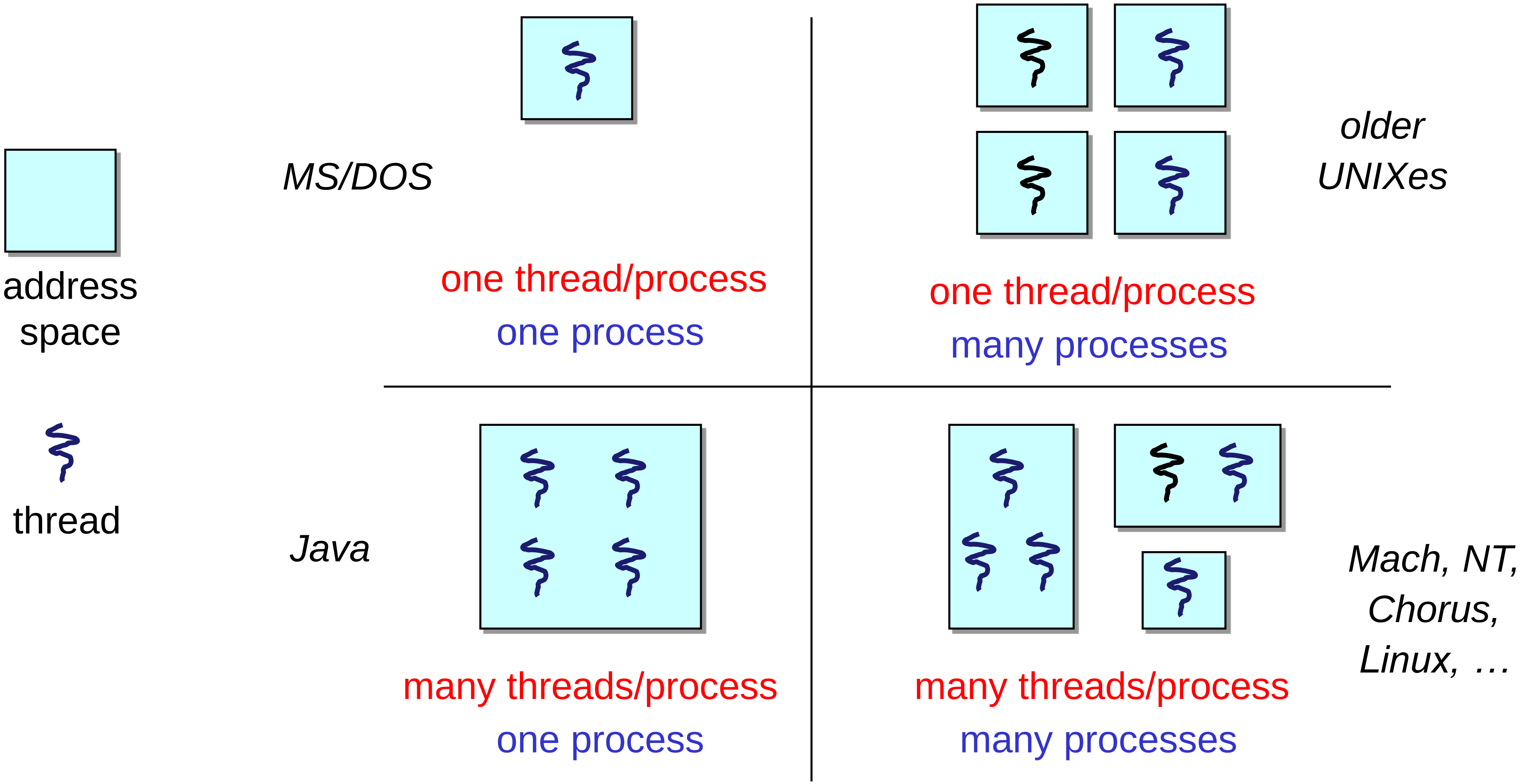
- Java threads may be created by:
  - ✓ Extending Thread class
  - ✓ Implementing the Runnable interface

- Java threads are managed by the JVM

- Java thread states



Threads Design Space



address space

thread

MS/DOS

one thread/process

one process

older UNIXes

one thread/process

many processes

Java

many threads/process

one process

Mach, NT, Chorus, Linux, …

many threads/process

many processes

Summary

- Thread concept
  - ✓ Separate the concept of a process from its execution state
  - ✓ Execution state: PC, SP, registers, etc.
- Multithreading models
  - ✓ User threads to kernel threads mapping
  - ✓ Many-to-one
    - User-level threads implementation
  - ✓ One-to-one
    - MS-Windows
  - ✓ Many-to-many
    - Solaris, Linux
- Multithreaded programming
  - ✓ Unix, Linux: POSIX-style threads (Pthread API)
  - ✓ MS-Windows: Win32 threads (Win32 API)
  - ✓ Java: Java threads (Thread class)