Operating Systems

Threads

Slides Courtesy:

Dr Syed Mansoor Sarwar

Issues with Processes

- The fork() system call is expensive
- IPC is required to pass information between a parent and its child processes.

Thread Concept

- A thread is a “lightweight” process which executes within the address space of a process.
- A thread can be scheduled to run on a CPU as an independent unit and terminate.
- Multiple threads can run simultaneously.

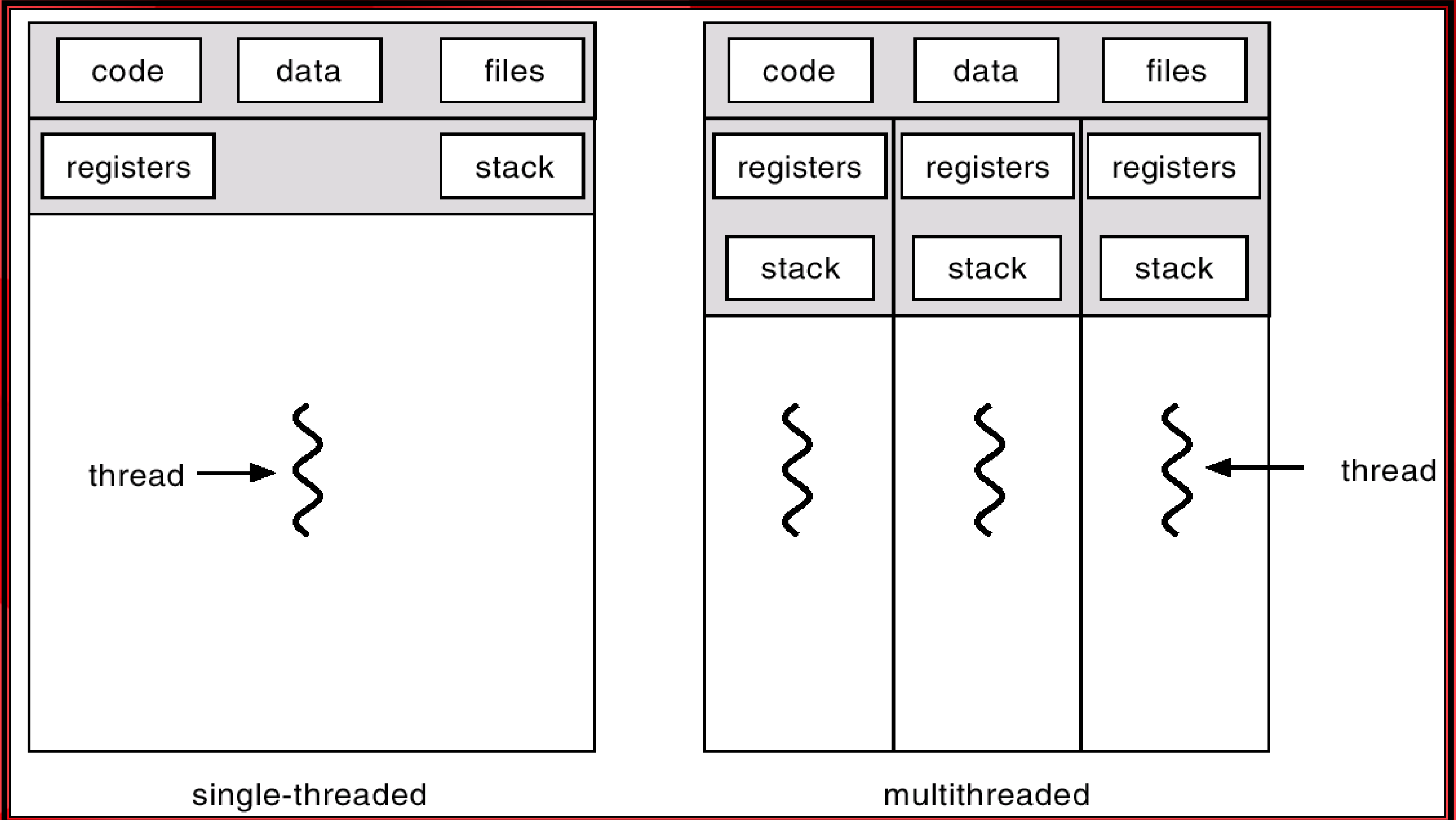
Thread Concept

- Threads have their own
  - Thread ID
  - CPU context (PC, SP, register set, etc.)
  - Stack
  - Priority
  - errno

Thread Concept

- Threads share
  - Code and data
  - Open files (through the PPFDT)
  - Current working directory
  - User and group IDs
  - Signal setups and handlers
  - PCB

Single and Multithreaded Processes



Threads are Similar to Processes

- A thread can be in states similar to a process (new, ready, running, blocked, terminated)
- A thread can create another thread

Threads are Different from Processes

- Multiple threads can operate within the same address space
- No “automatic” protection mechanism is in place for threads—they are meant to help each other

Advantages of Threads

- **Responsiveness**
  - Multi-threaded servers (e.g., browsers) can allow interaction with user while a thread is formulating response to a previous user query (e.g., rendering a web page)

Advantages of Threads

- Resource sharing
  - Process resources (code, data, etc.)
  - OS resources (PCB, PPFDT, etc.)

Advantages of Threads

- **Economy**
  - Take less time to create, schedule, and terminate
  - Solaris 2: thread creation is 30 times faster than process creation and thread switching is <u>five</u> times faster than process switching

Advantages of Threads

- **Performance** in multi-processor and multi-threaded architectures (e.g., Intel’s P4 HT)
  - Multiple threads can run simultaneously

Disadvantages of Threads

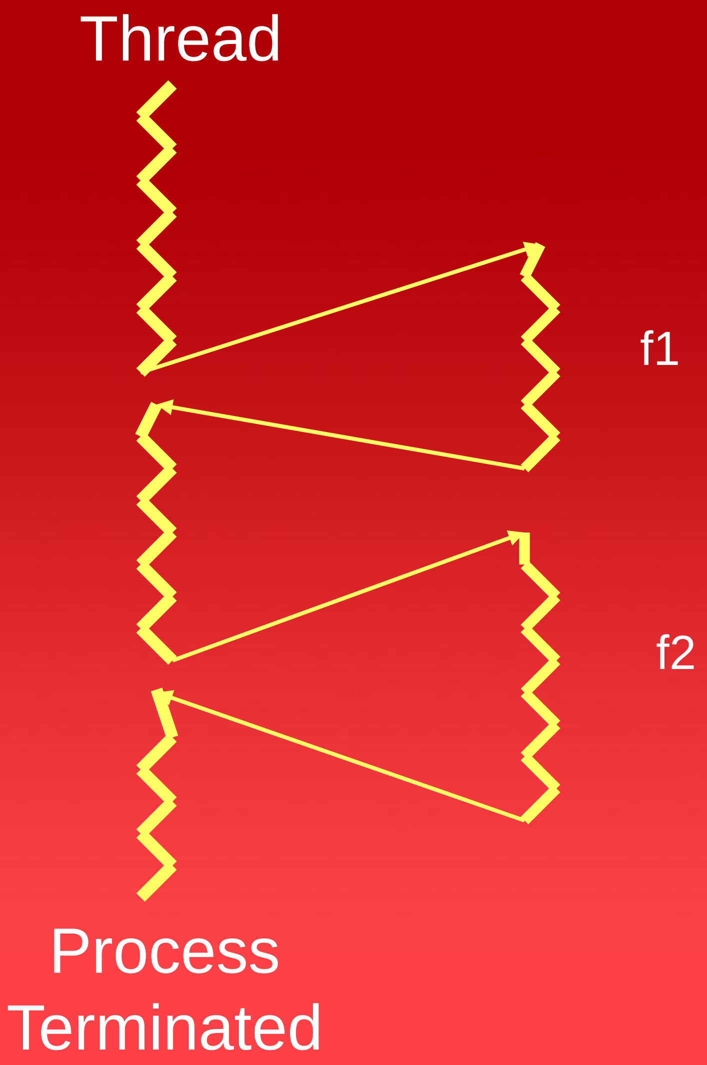
- Resource sharing—synchronization needed between threads
- Difficult to write and debug multi-threaded programs

Single-Threaded Process

```
main()
{
        …
        f1(…);
        …
        f2(…);
        …
}

f1(…)
{  …  }

f2(…)
{  …  }
```

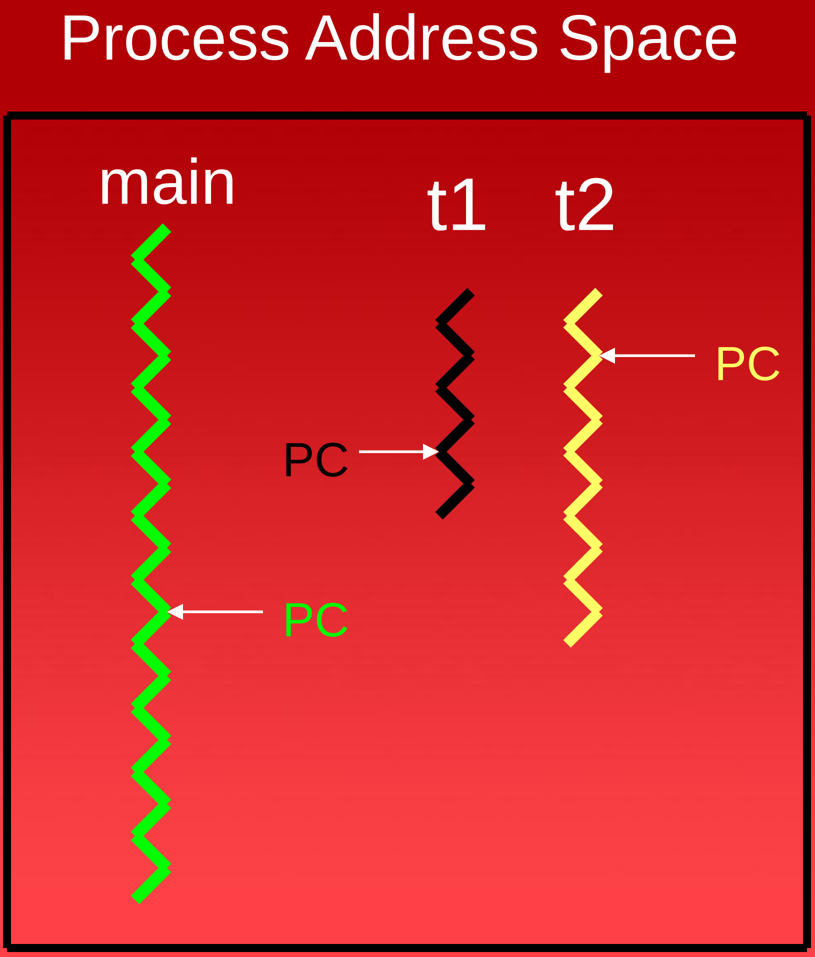


Multi-Threaded Process

```
main()
{
        …
        thread(t1,f1);
        …
        thread(t2,f2);
        …
}

f1(…)
{  …  }

f2(…)
{  …  }
```



User Threads

- Thread management done by user-level threads libraries
  - Kernel not aware of threads
  - CPU not interrupted during thread switching
  - A system call by a thread blocks the whole process
  - Fair scheduling: P1 has one thread and P2 has 100 threads

User Threads

- Examples
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris 2 *threads*

Kernel Threads

- Thread management done by kernel
  - Kernel aware of threads
  - CPU switched during context switching
  - A system call does not block the whole process
  - Fair scheduling: P1 has one thread and P2 has 100

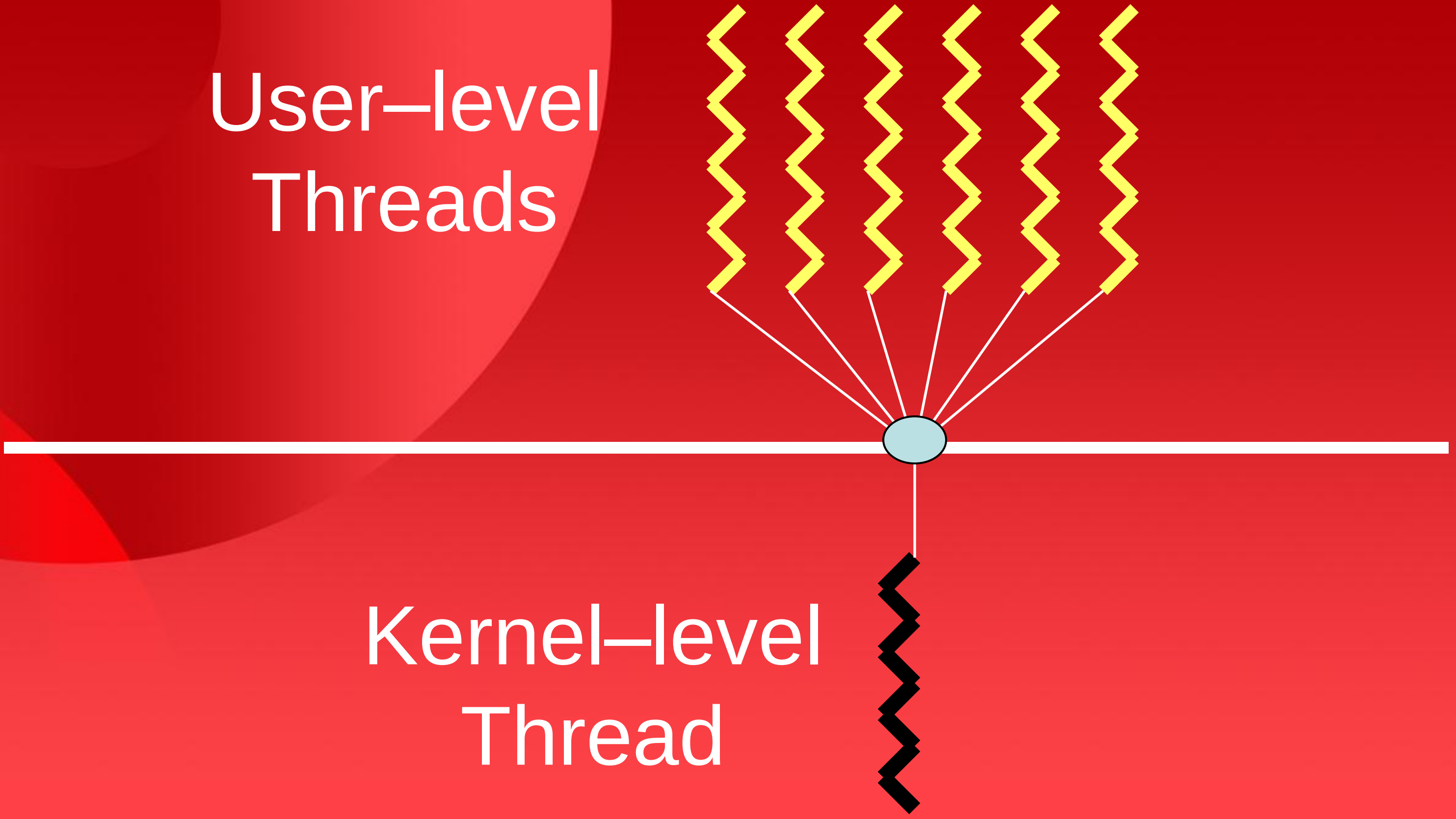
Kernel Threads

- Examples
  - Windows NT/2000
  - Solaris 2
  - Linux

Multithreading Models

- Support for both user and kernel threads
- **Many-to-One**: Many user threads per kernel thread; process blocks when a thread makes a system call
- Solaris Green threads
- Pthreads

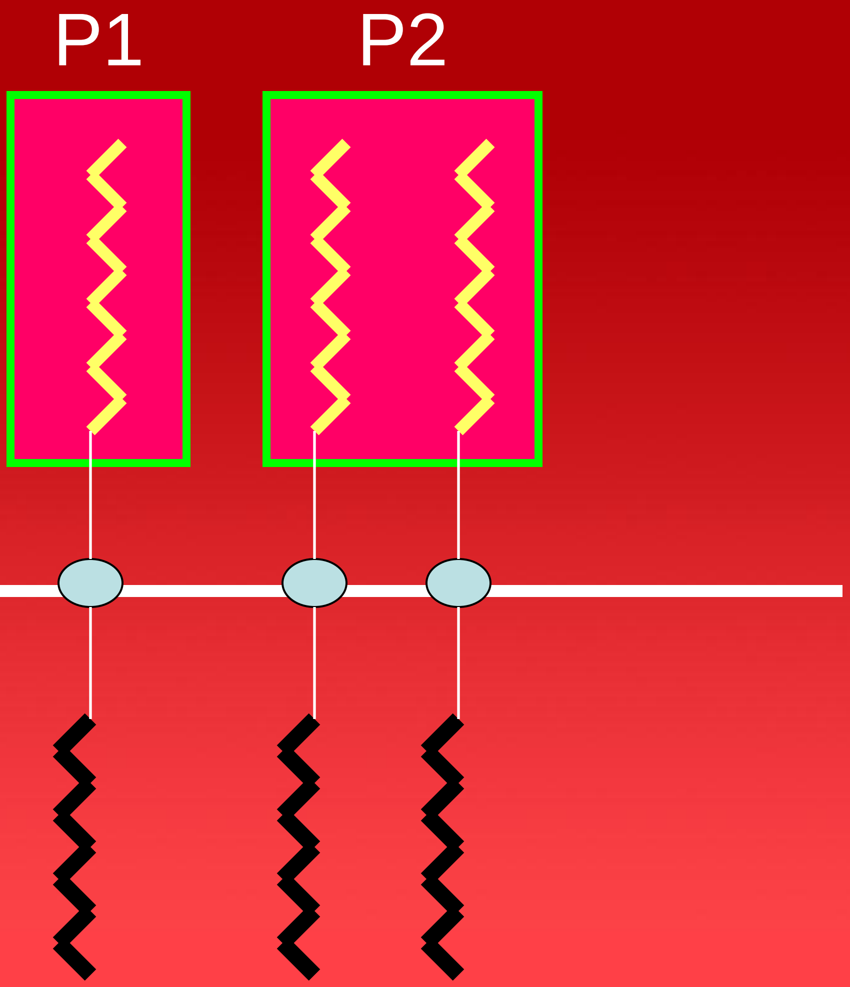
Many-to-One Model



Multithreading Models

- **One-to-One**: One user thread per kernel thread; process does not block when a thread makes a system call
- Overhead for creating a kernel thread per user thread
- True concurrency achieved
- Windows NT/2000, OS/2

One-to-One Model



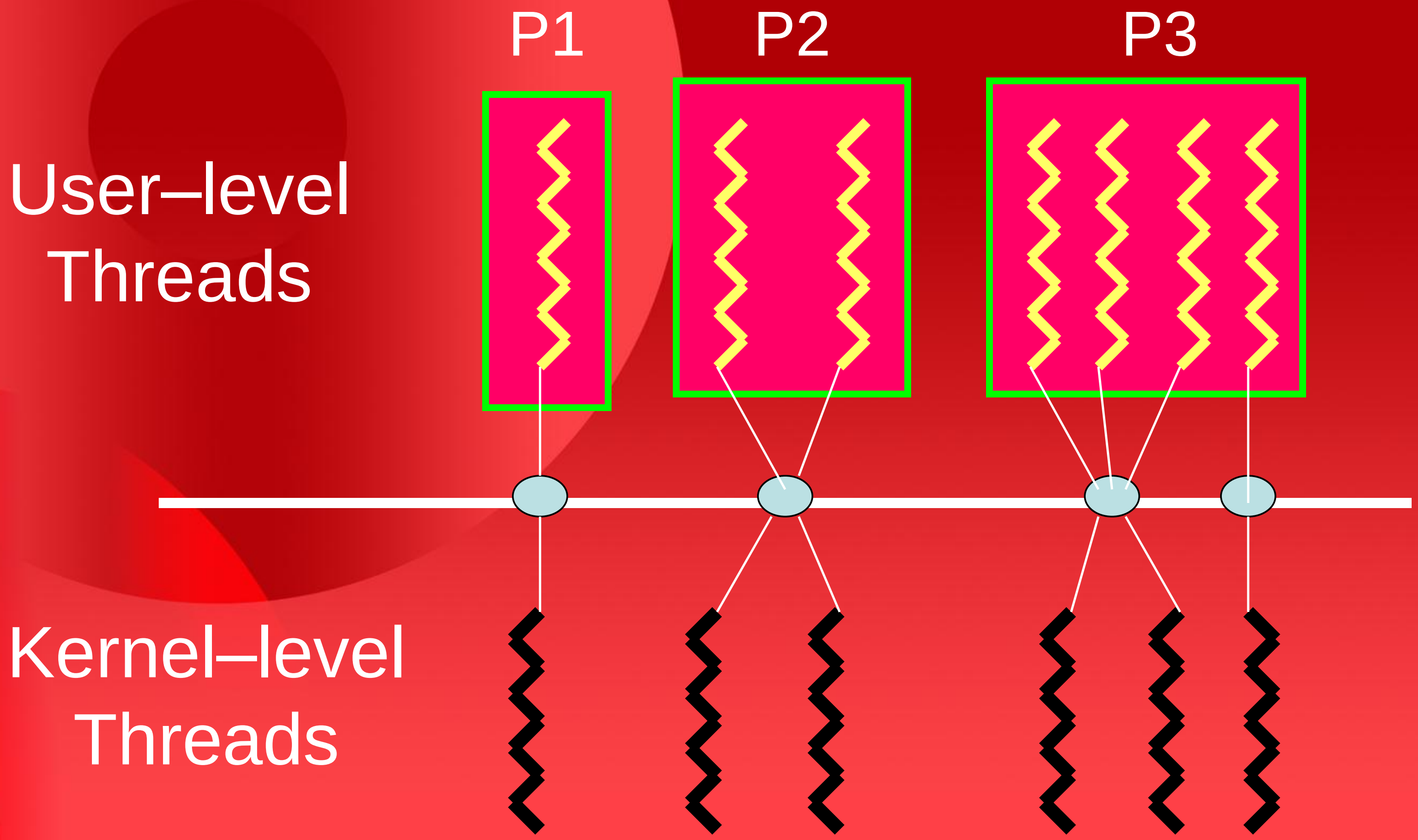
User–level Threads

Kernel–level Threads

Multithreading Models

- **Many-to-Many**: Multiple user threads multiplexed over a smaller or equal number of kernel threads
- True concurrency not achieved because kernel can only schedule one thread at a time
- Kernel can schedule another thread when a user thread makes a blocking system call
- Solaris 2, HP-UX

Many-to-Many Model



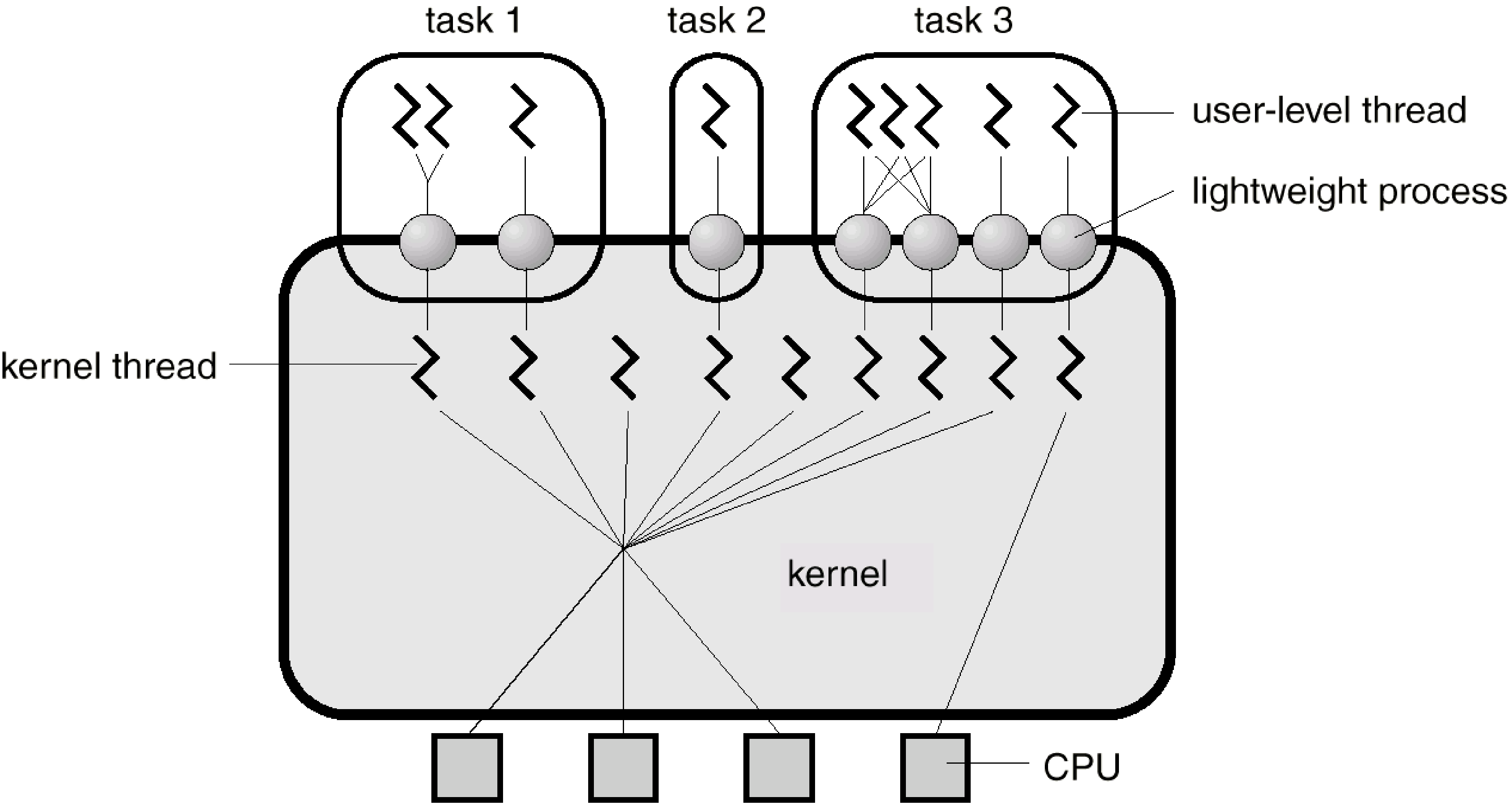
Solaris 2 Threads Model

- Solaris 2: threads, light-weight processes (LWPs), and processes
  - At least one LWP per process to allow a user thread to talk to a kernel thread
  - User level threads switched and scheduled among LWPs without kernel’s knowledge

Solaris 2 Threads Model

- One kernel thread per LWP; some kernel threads have no LWP (e.g., threads for clock interrupt handler and scheduling)

Solaris 2 Threads Model



Pthreads

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

Creating a Thread

-
  ```
  int pthread_create (pthread_t *threadp,
                      const pthread_attr_t *attr,
                      void* (*routine)(void *),
                      arg *arg);
  ```

Creating a Thread

Where:

`threadp` The thread we are trying to create—thread ID (TID)

`attr` Used to modify the thread attributes (stack size, stack address, detached, joinable, priority, etc.)

`routine` The thread function

`arg` Any argument we want to pass to the thread function. This does not have to be a simple native type, it can be a ‘struct’ of whatever we want to pass in.

Error Handling

- `pthread_create()` fails and returns the corresponding value if any of the following conditions is detected:
  - **EAGAIN** The system-imposed limit on the total number of threads in a process has been exceeded or some system resource has been exceeded (for example, too many LWPs were created).

Error Handling

- **EINVAL** The value specified by `attr` is invalid.
- **ENOMEM** Not enough memory was available to create the new thread.
- **Error handling:**
  - #include <errno.h>
  - Error handling code

Joining a Thread

- Waiting for a thread
- int pthread_join(pthread_t aThread,
  void **statusp);
- ‘statusp’ get return value of pthread_exit

Joining a Thread

- Cannot join with a detached thread
- Can only join with thread's in the same process address space
- Multiple threads can join with one thread but only one returns successfully; others return with an error that no thread could be found with the given TID

Terminating a Thread

- Main thread terminates
- Thread returns
- void pthread_exit(void *valuep)
- Returns value pointed to by ‘valuep’ to a joining thread, provided the exiting thread is not detached

Example 1

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
/* Prototype for a function to be passed to our thread */
void* MyThreadFunc(void *arg);
int main()
{
   pthread_t aThread;
   /* Create a thread and have it run the MyThreadFunction */
   pthread_create(&aThread, NULL, MyThreadFunc, NULL);
   /* Parent waits for the aThread thread to exit */
   pthread_join(aThread, NULL);
   printf ("Exiting the main function.\n");
   return 0;
}
```

Example 1

```
void* MyThreadFunc(void* arg)
{
        printf ("Hello, world! ... The threaded version.\n");
        return NULL;
}
```

```
$ gcc hello.c –o hello –lpthread –D_REENTRANT
$ hello
Hello, world! ... The threaded version.
Exiting the main function.
$
```

Example 2

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   printf("\n%d: Hello World!\n", threadid);
   pthread_exit(NULL);
}
```

Example 2

```
int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;

    for (t=0; t < NUM_THREADS; t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) {
            printf("ERROR; return code is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```