240-229: SDA (Operating Systems session)

Lecture 2: Process and Thread Concepts

Associate Professor Dr. Sangsuree Vasupongayya
sangsuree.v@psu.ac.th
Department of computer Engineering
Prince of Songkla University

Outline

- Lecture:
  - Process Concept
  - Operations on Processes
  - Thread Concept

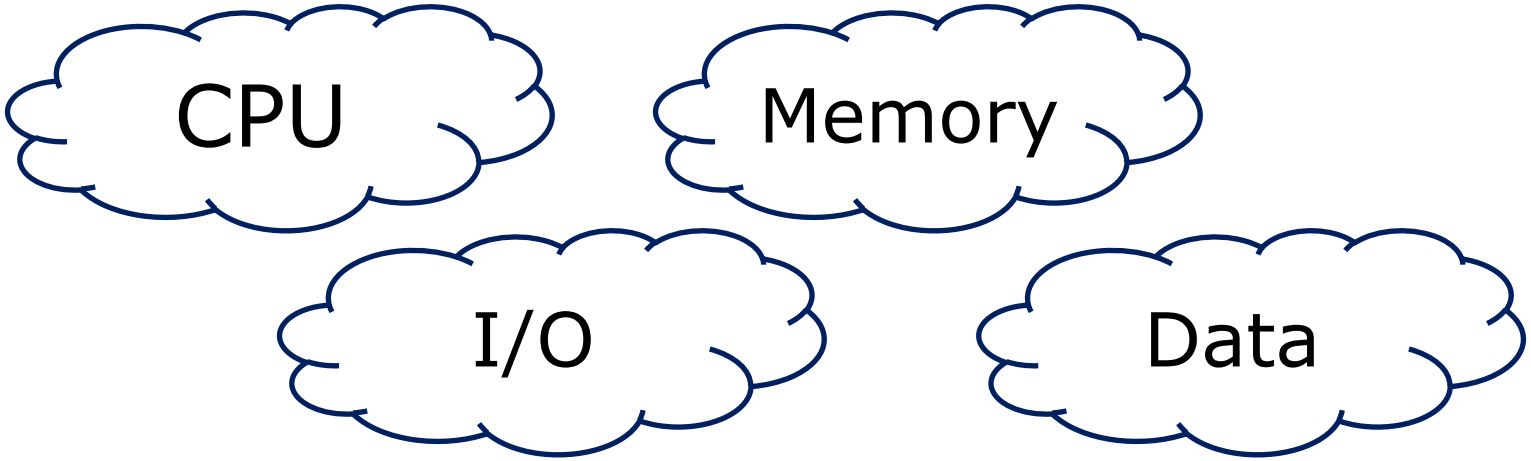
- Labs:
  - Process
  - Thread

Thinking

What is a process?

Process – a program in execution; process execution must progress in sequential fashion

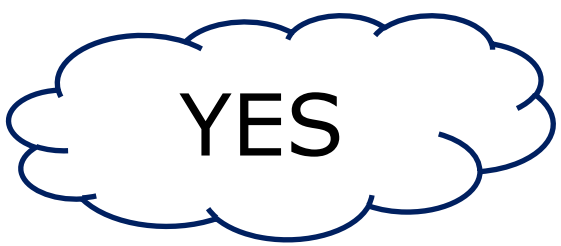
What a process need during execution?

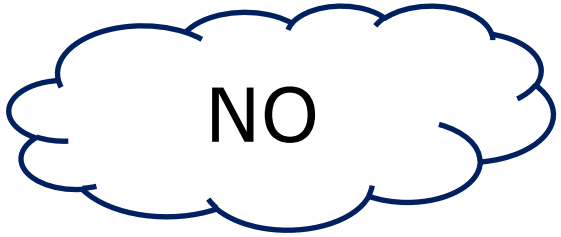


What does OS need to know about a process?

Thinking

Are all processes in the system having a different state during their lifetime?





How many state?

How do I know what state a process is at?

Process Concept

- **Process** – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section

heap: is a memory that is dynamically allocated during process runtime

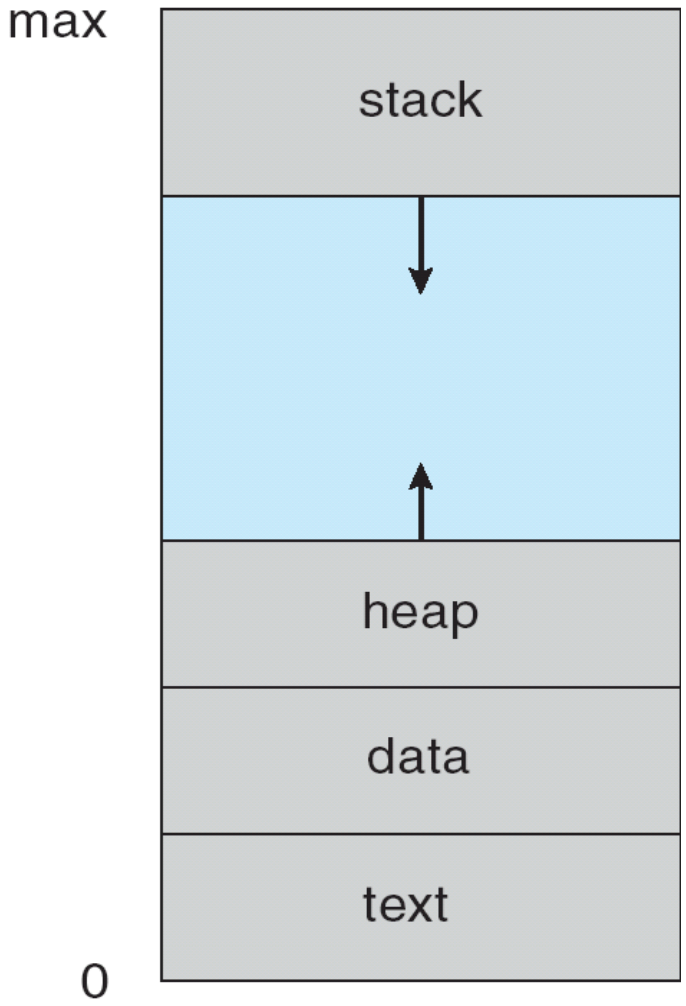
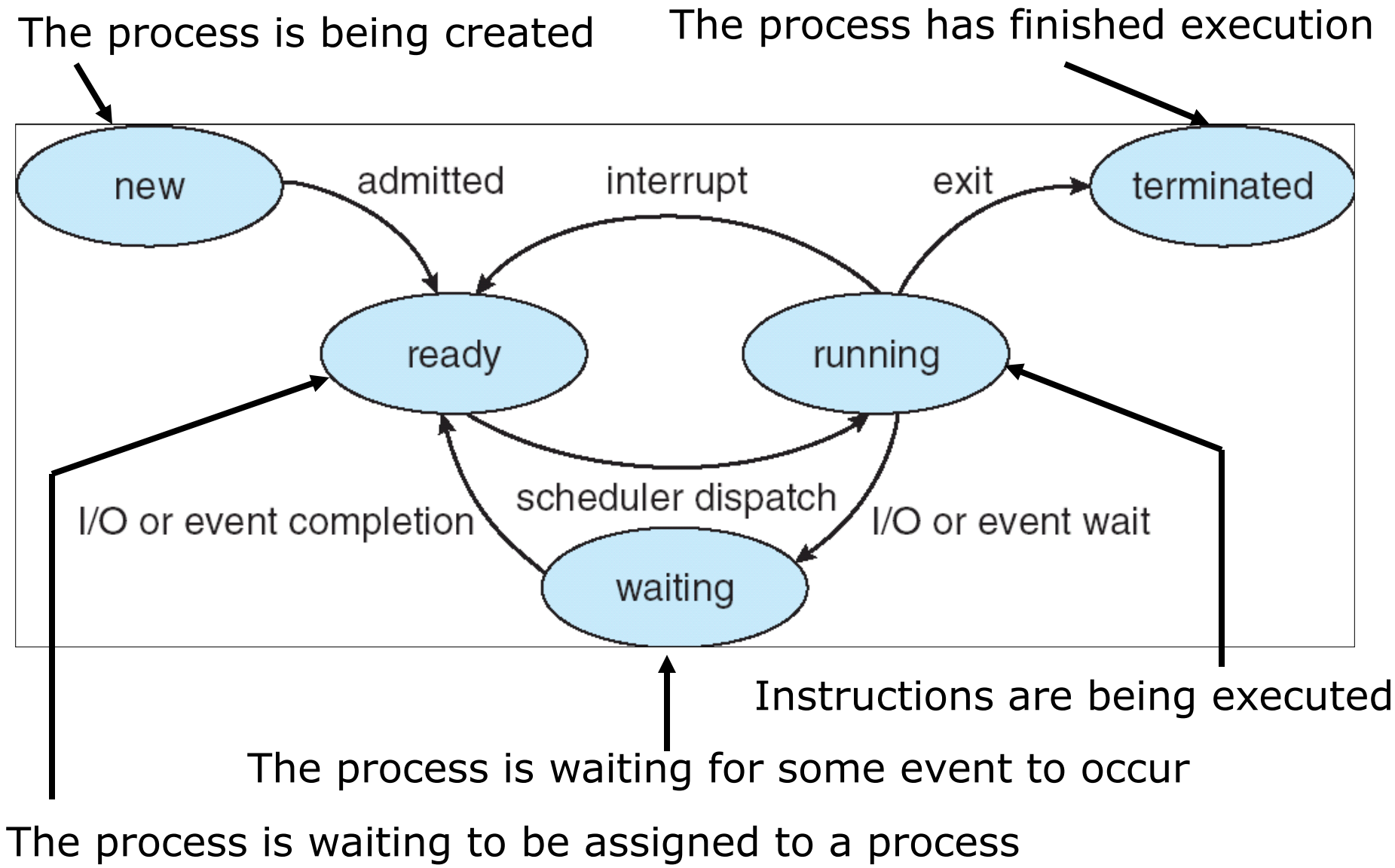


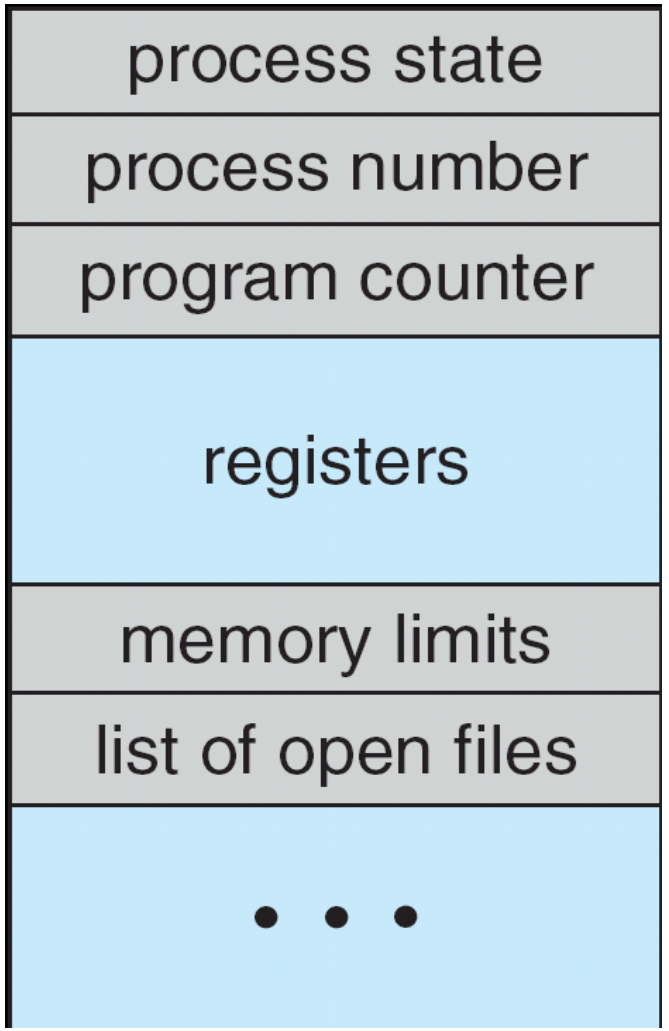
Diagram of Process State



Process Control Block (PCB)

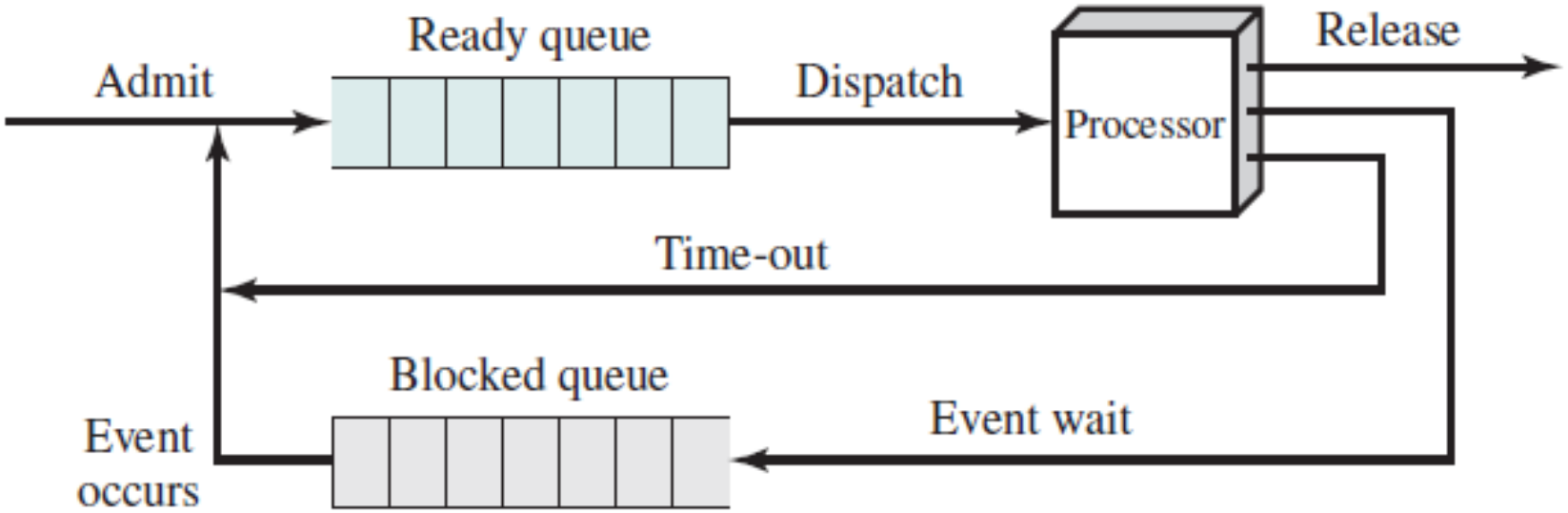
Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information



Process Scheduling Queues

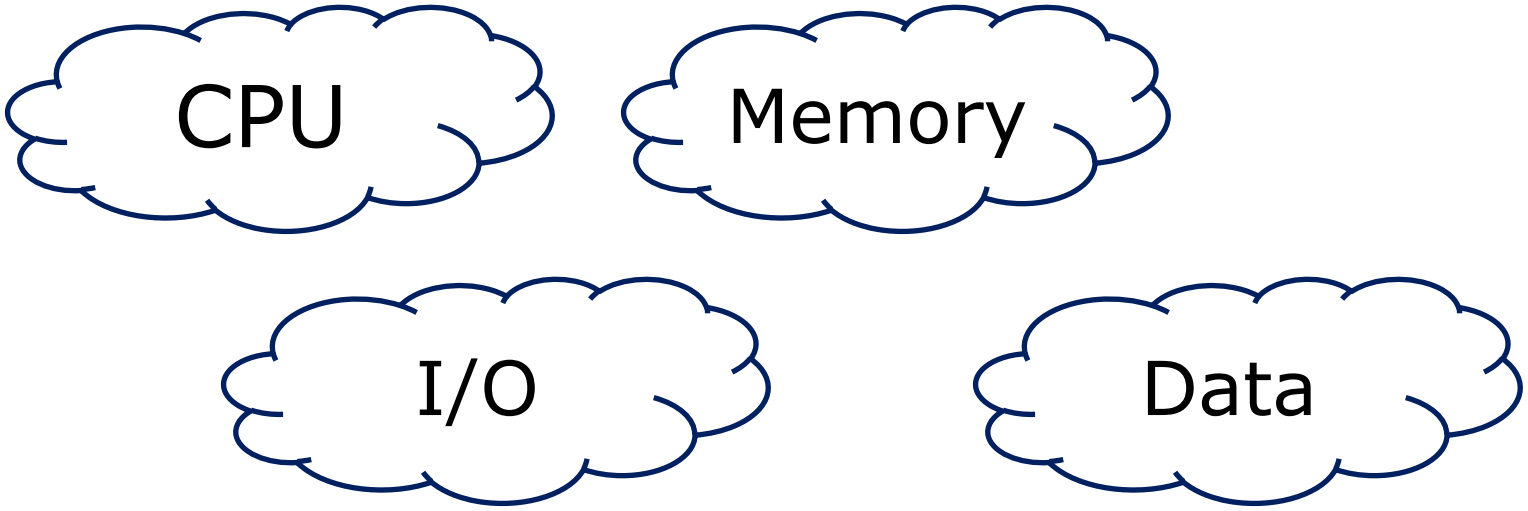
- Job queue – set of all processes in the system
- Ready queue – set of all processes residing in main memory, ready and waiting to execute
- Device queues – set of processes waiting for an I/O device
- Processes migrate among the various queues



(a) Single blocked queue

Thinking

What happen when you have more than one process on the system at the same time?



CPU: only a limited # of core

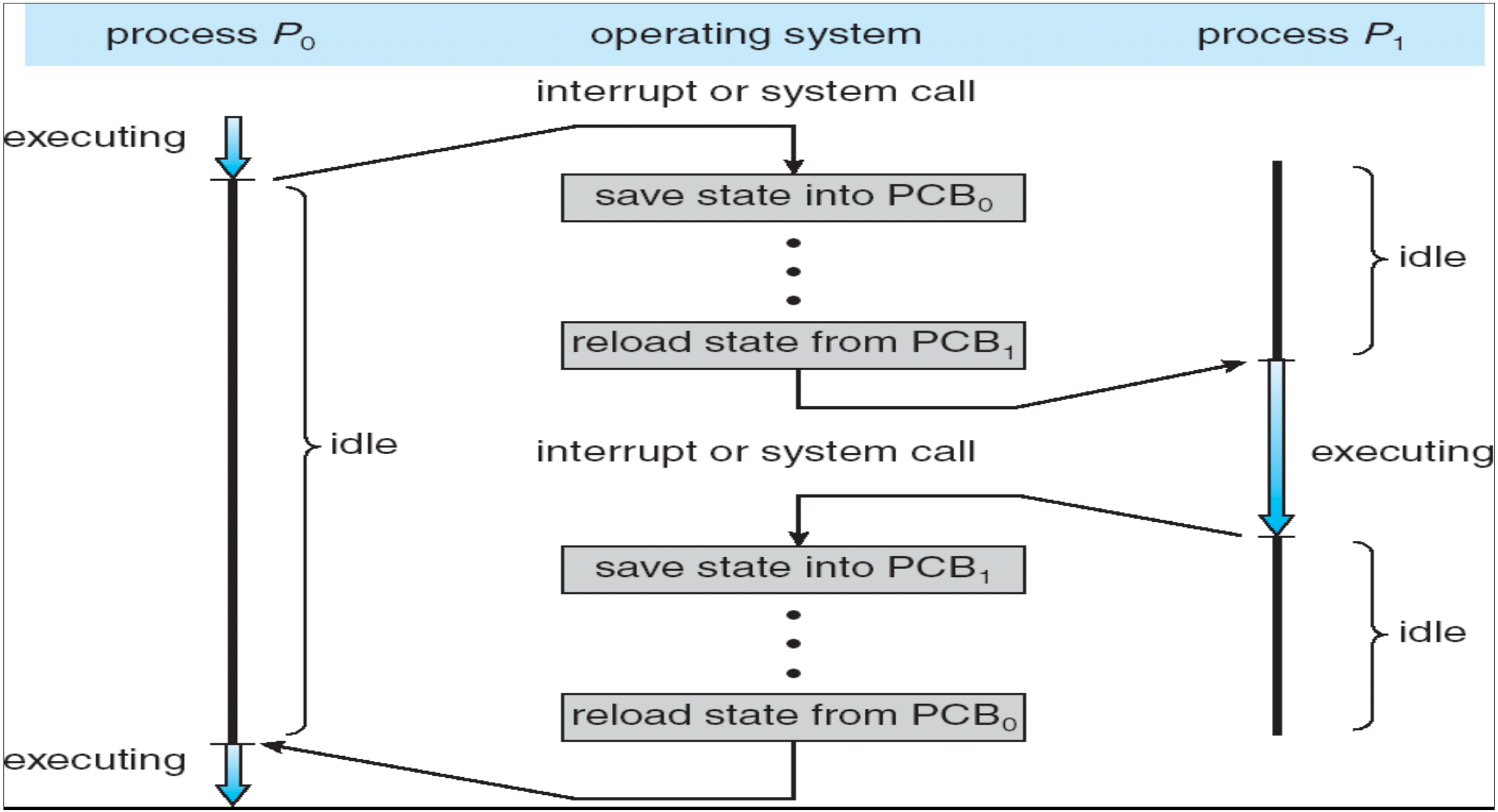
I/O: only one monitor

Memory: only limited space

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

CPU Switch From Process to Process

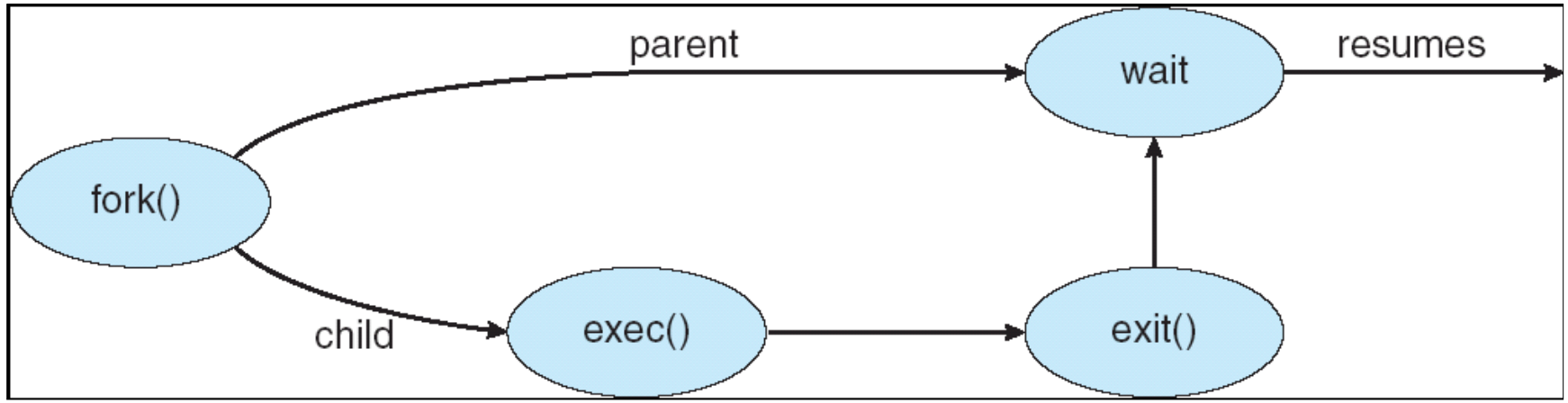


Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program



Process Creation (fork)

- Fork creates a new process by copying the calling process
- The new process has it own
  - Memory address space (copied from parent)
    - Instructions (same program as parent!)
    - Data
    - Stack
  - Register set (copied from parent)
  - Process table entry in the OS

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Lab: Fork

```
int main(int argc, char *argv[])
{
  int temp = 10;
  pid_t pid;
  /* fork a child process */
  pid = fork();   // return 0 to the child process
                  // whereas the nonzero process id of
                  // the child is returned to the parent

  if(pid < 0) { /* error occurred */
     fprintf(stderr,"Fork Failed");
     exit(-1);
  }
  else if(pid == 0) {  /* child process */
    // sleep(5);
    printf("This is a child process. pid = %d, parent pid = %d\n",getpid(),getppid());
    sleep(2);
    printf("This is a child process. I woke up! \n");
    execlp("/bin/ls","ls",NULL);
    //while(1);
    temp++;
    printf("CHILD: temp = %d\n",temp);
  }
  else { /* parent process*/
    //sleep(5);
    printf("This is a parent process. pid = %d, child pid = %d\n",getpid(),pid);
    /* parent will wait for the child to complete */
    wait(NULL);
    printf("PARENT: Child Complete.\n");
    printf("PARENT: temp = %d\n",temp);
    exit(0);
  }
}
```

Process have the following components

- An address space
- A collection of operating system state
- A CPU context .. Or thread of control

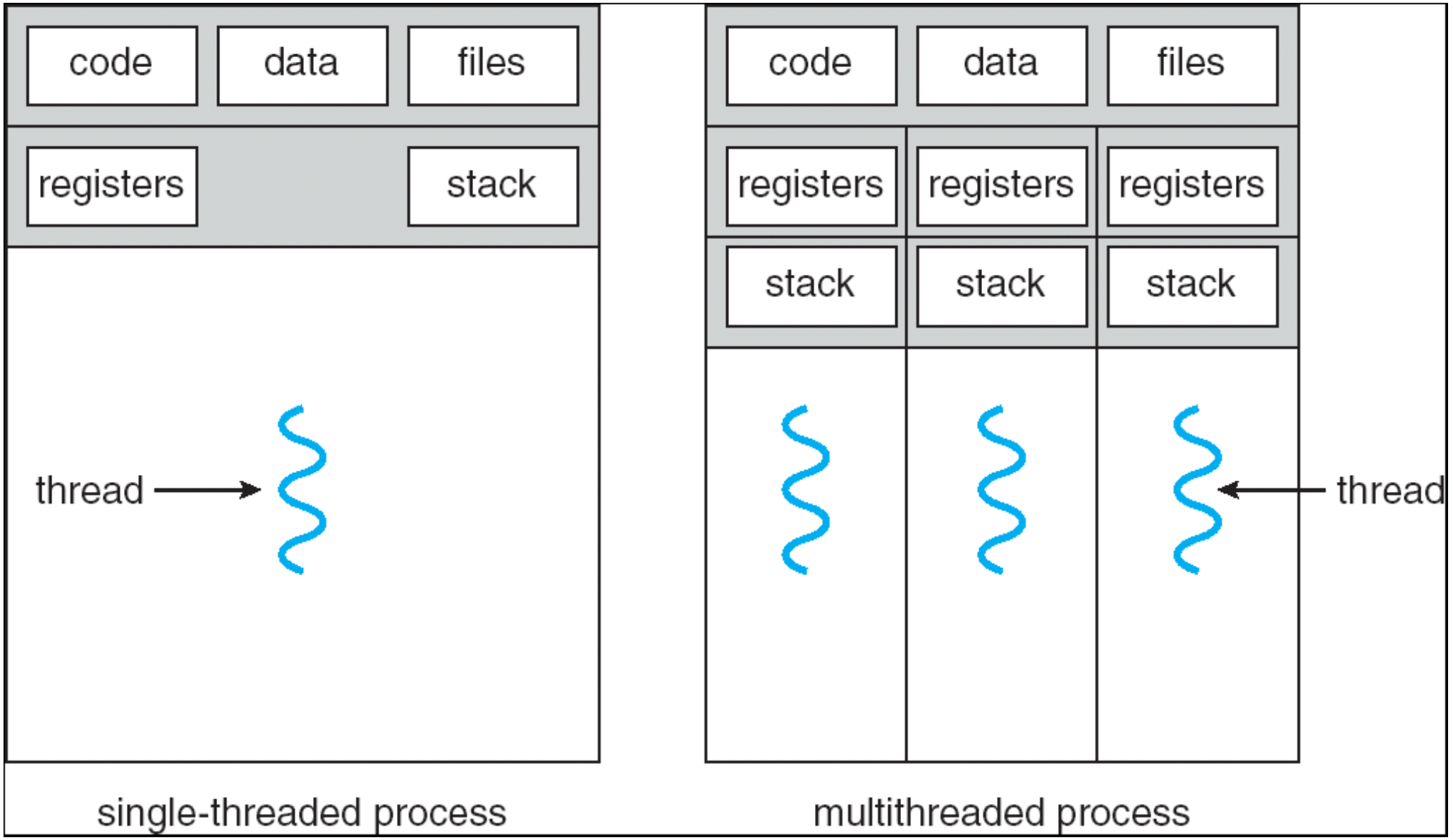
What is a Thread?

- A thread executes a stream of instructions
- A thread is an abstraction for control-flow

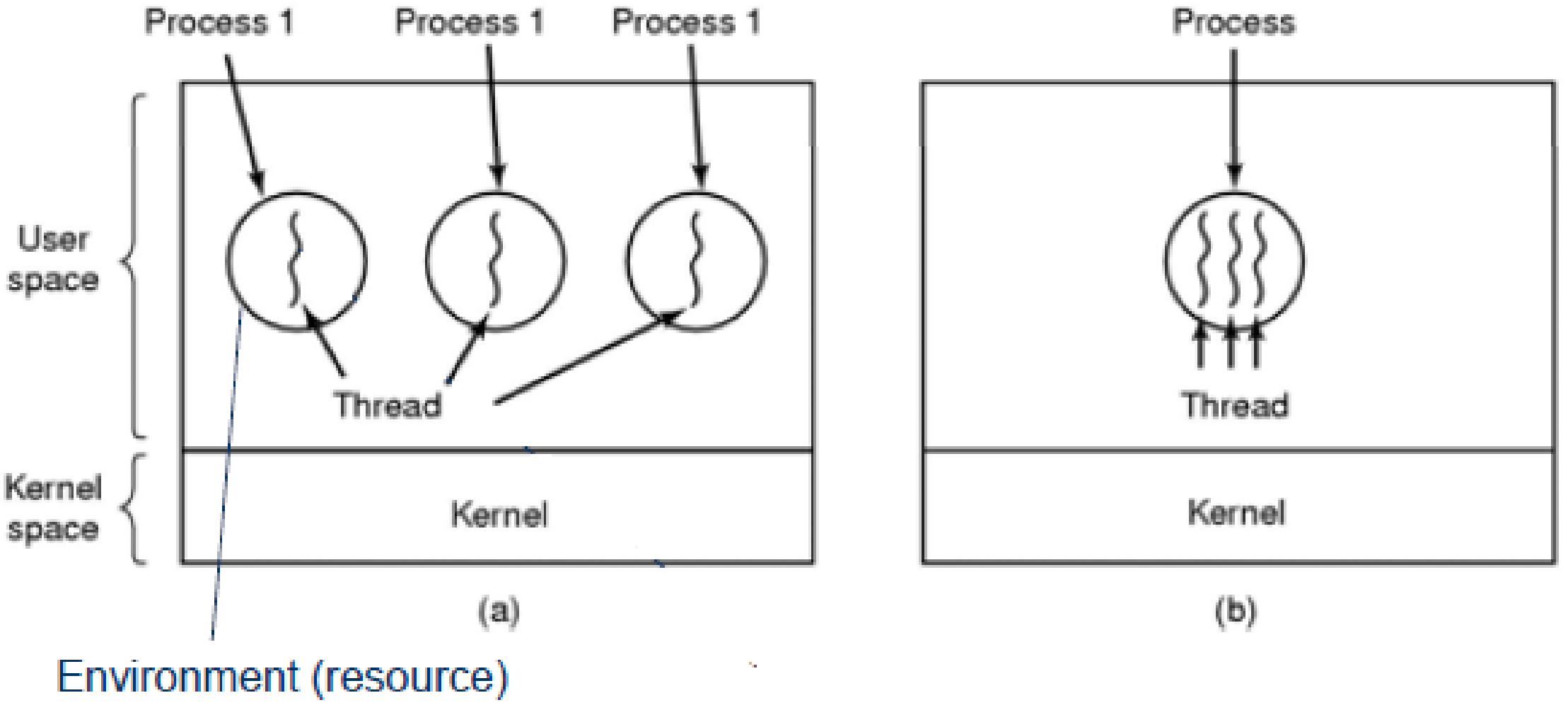
Thread is a processor context and stack

- Allocated a CPU by a scheduler
- Executes in a memory address space

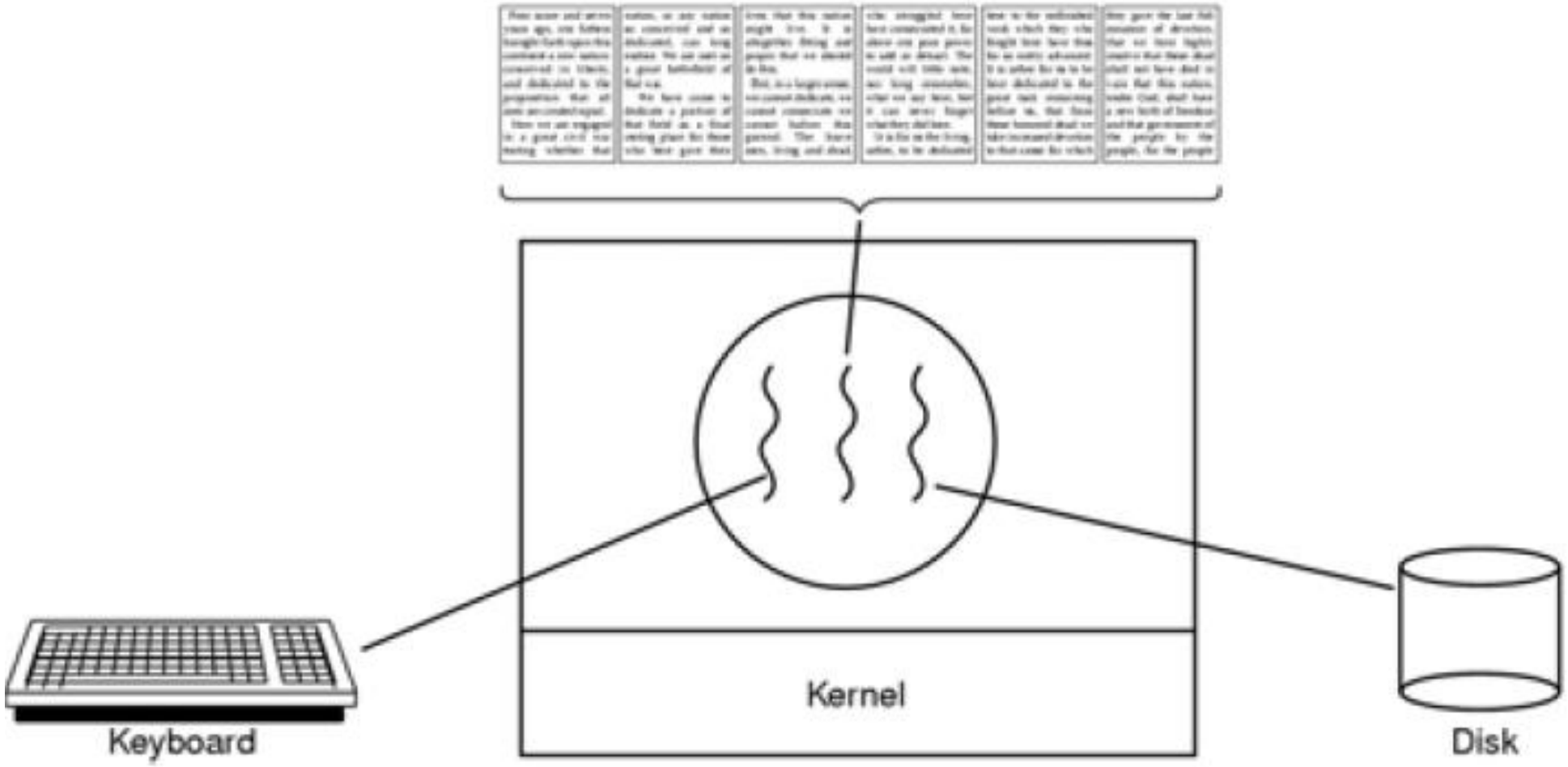
Single and Multithreaded Processes



3 processes vs 3 threads

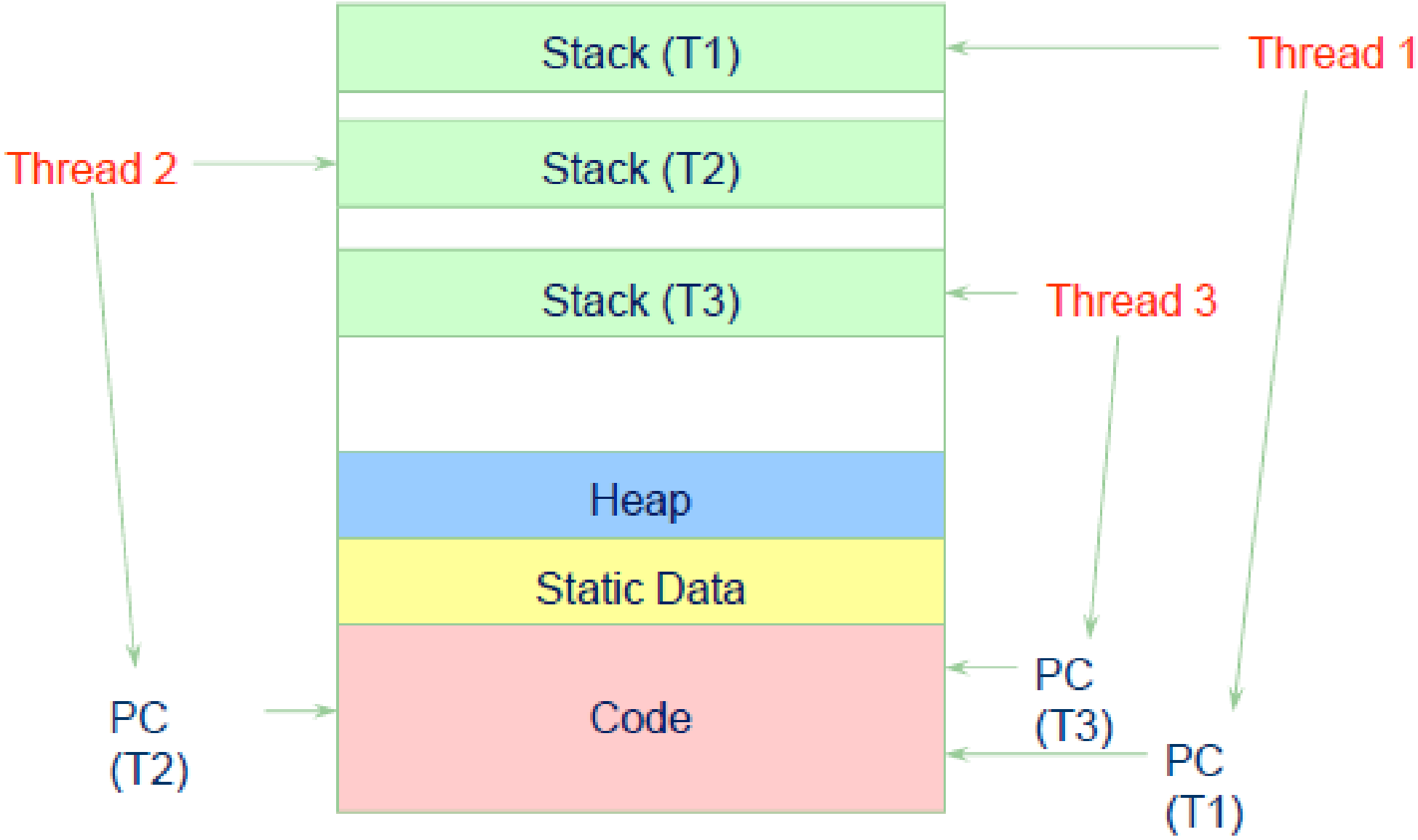


Example: Word processor



- One thread can wait while the other can still process

3 threads

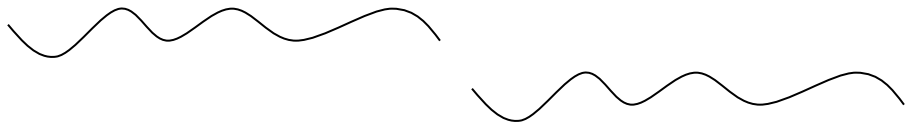

Stack (T1)
Stack (T2)
Stack (T3)
Heap
Static Data
Code
Thread 1
Thread 2
Thread 3
PC
(T2)
PC
(T3)
PC
(T1)

Why use threads?

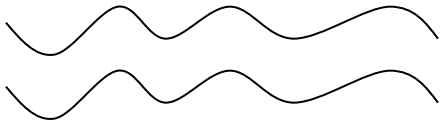
- Responsiveness
  - Handle asynchronous events
- Economic
  - Blocking due to I/O
  - Overlap computation
- Resource sharing
  - Low cost communication via shared memory
- Utilization of Multiprocessor Architectures

Sequential vs. concurrent threads

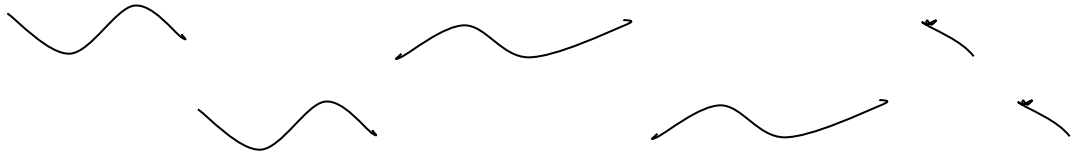
- Sequential threads



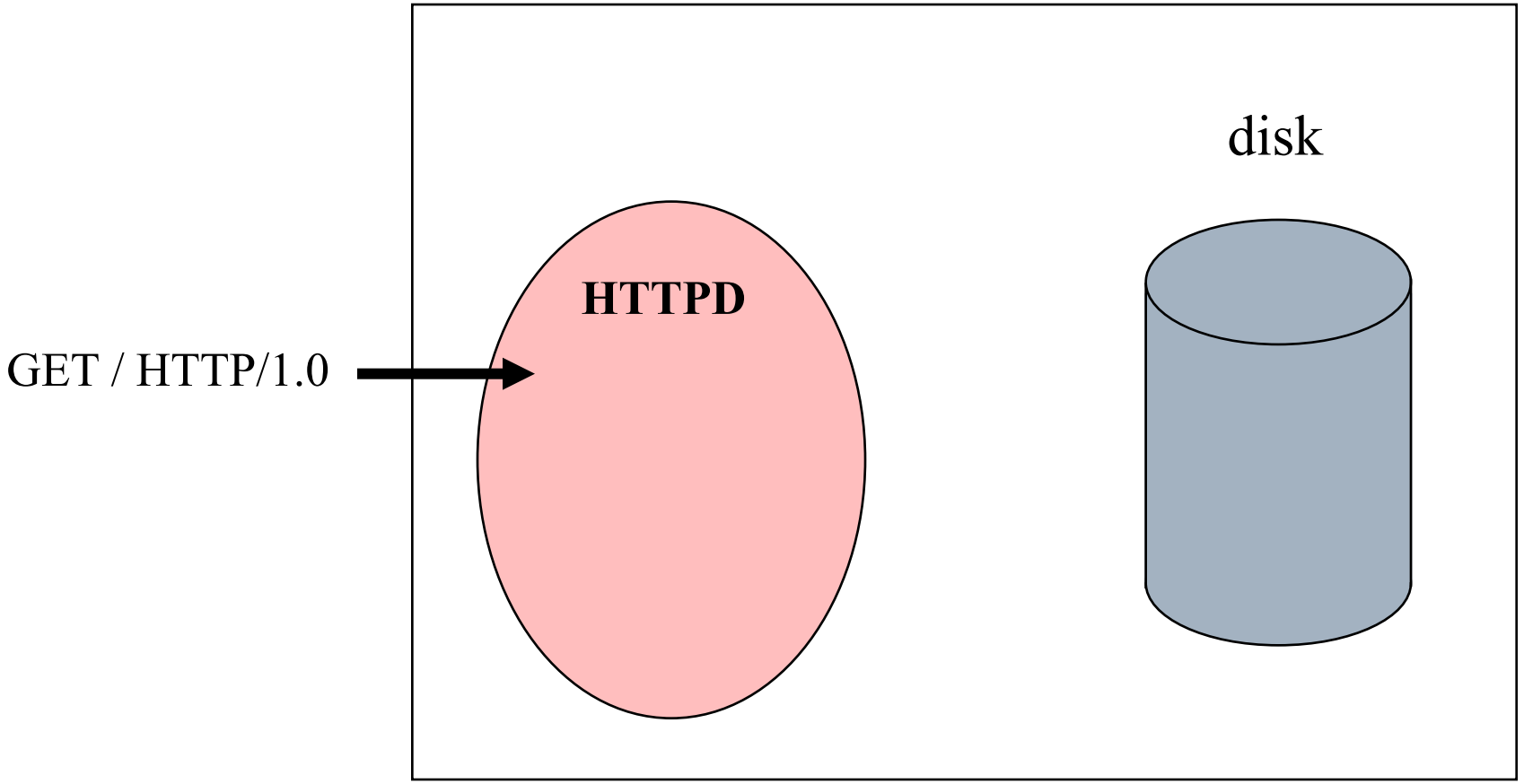
- Concurrent threads running simultaneously on two processors



- Concurrent threads (with gaps in their executions) interleaved on one processor

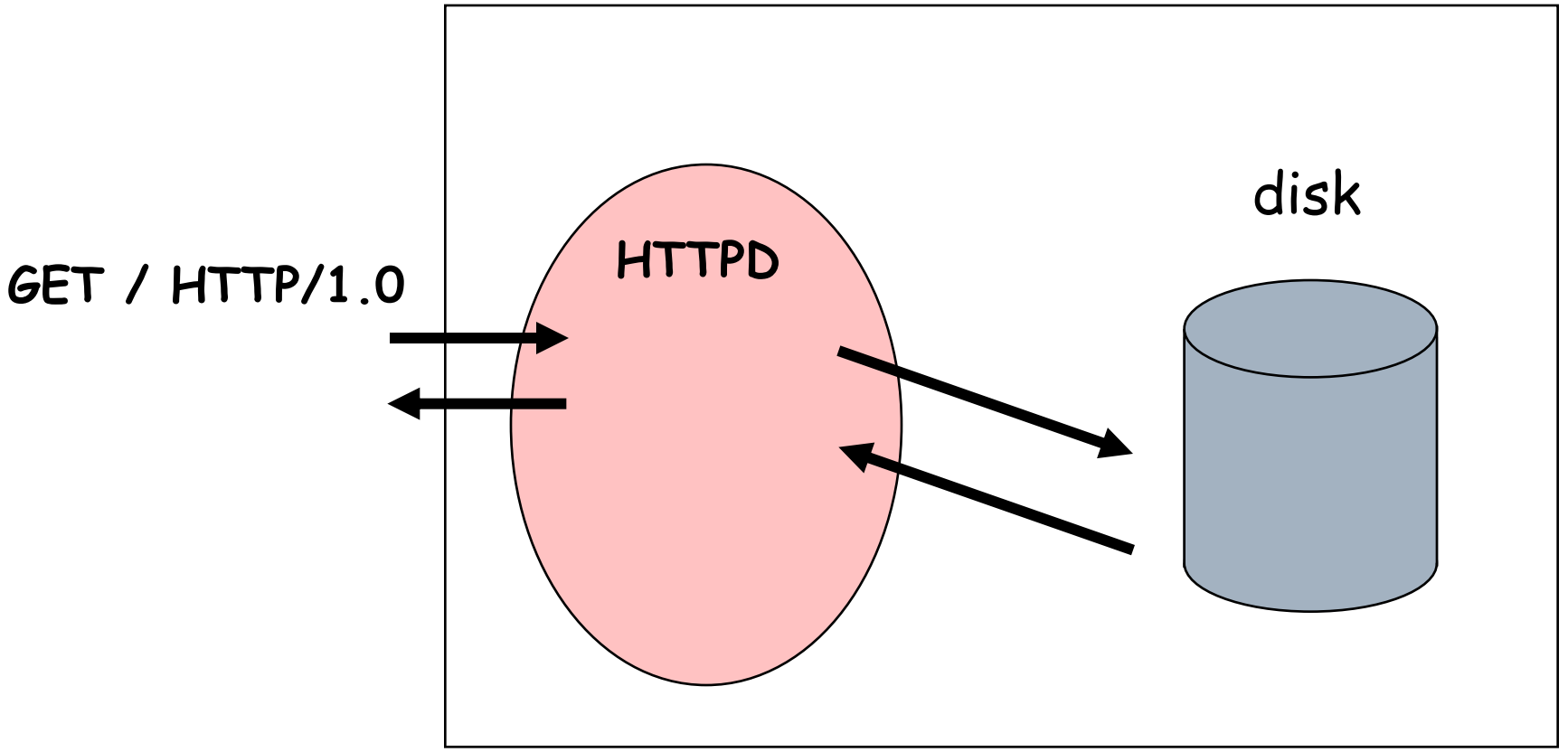


A Web Server Example



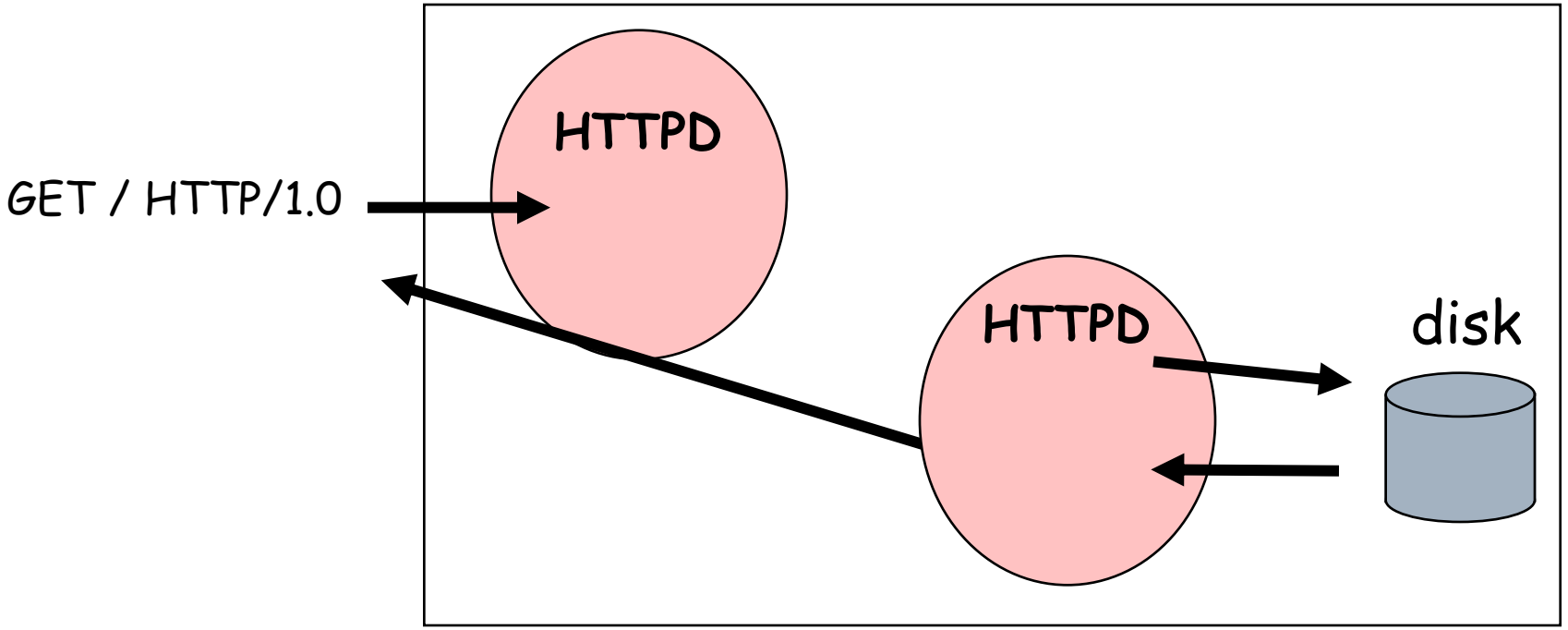
A Web Server Example

Why is this not a good web server design?



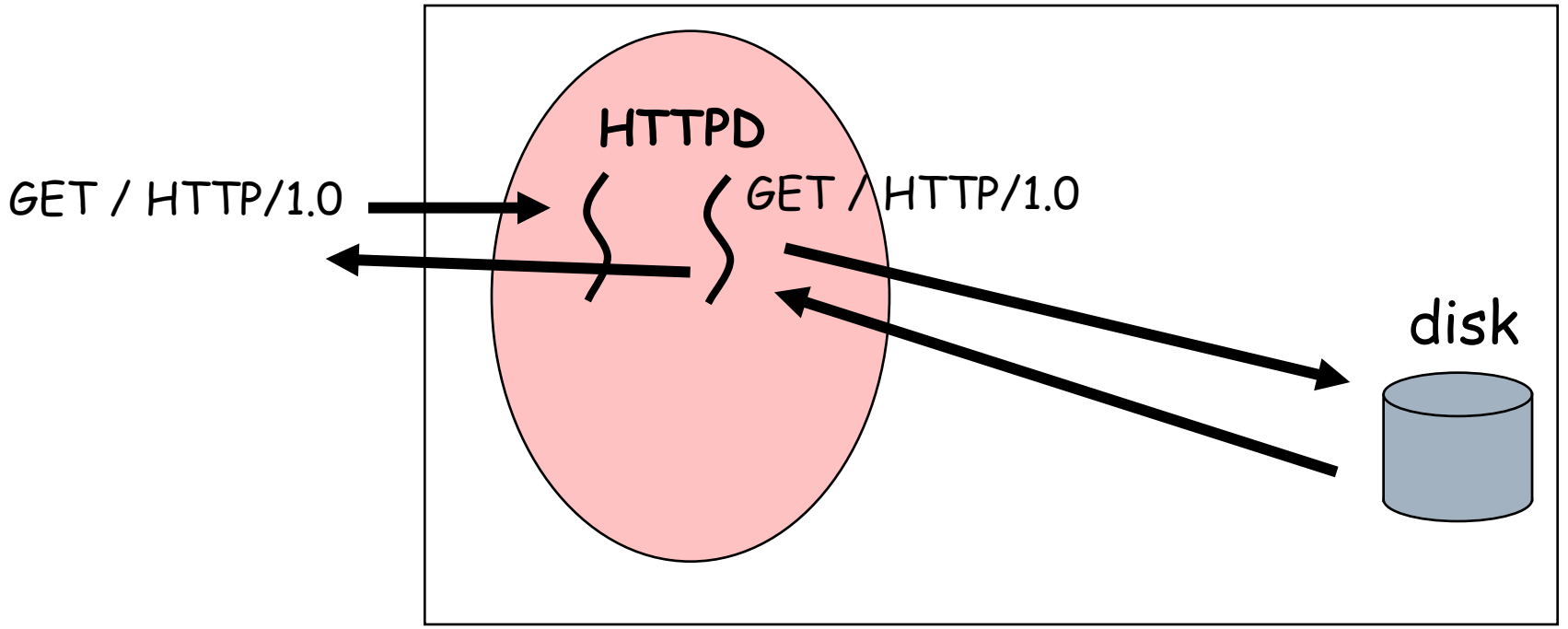
A Web Server Example

Fork more process



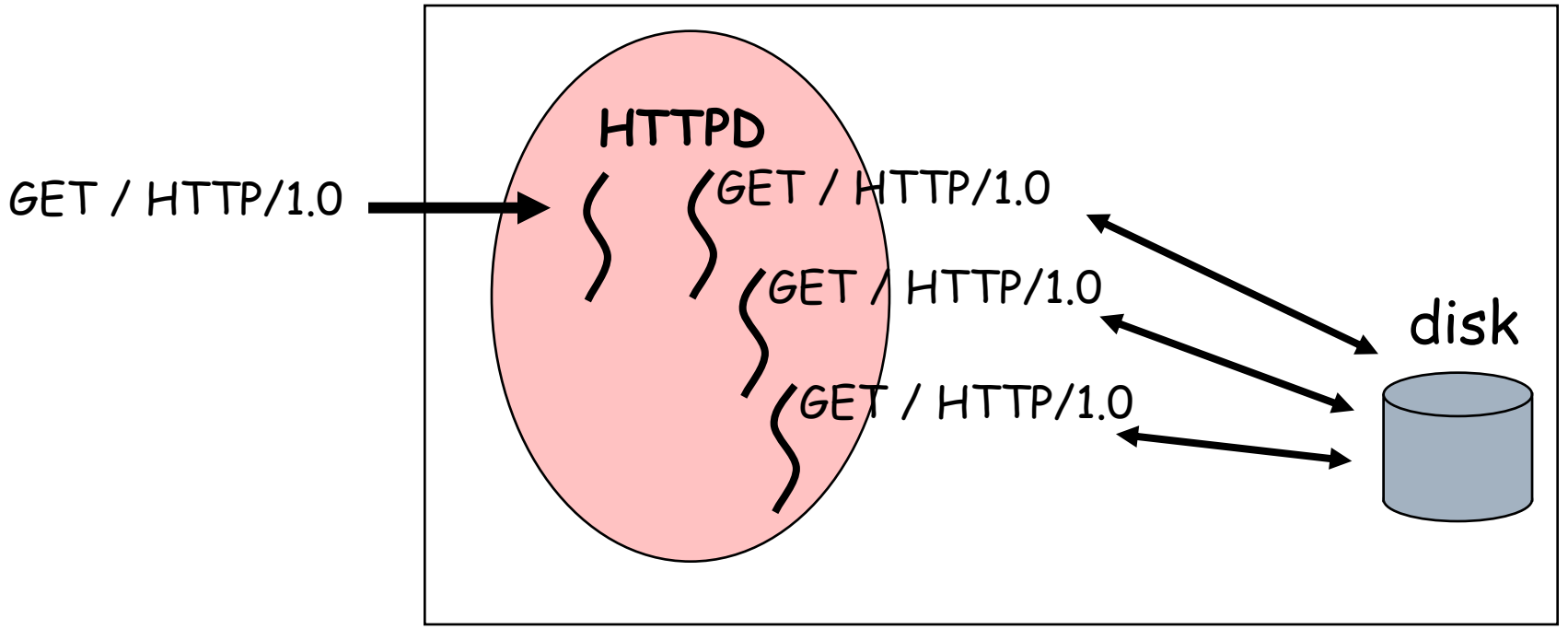
A Web Server Example

Create a thread



A Web Server Example

Have a pool of threads

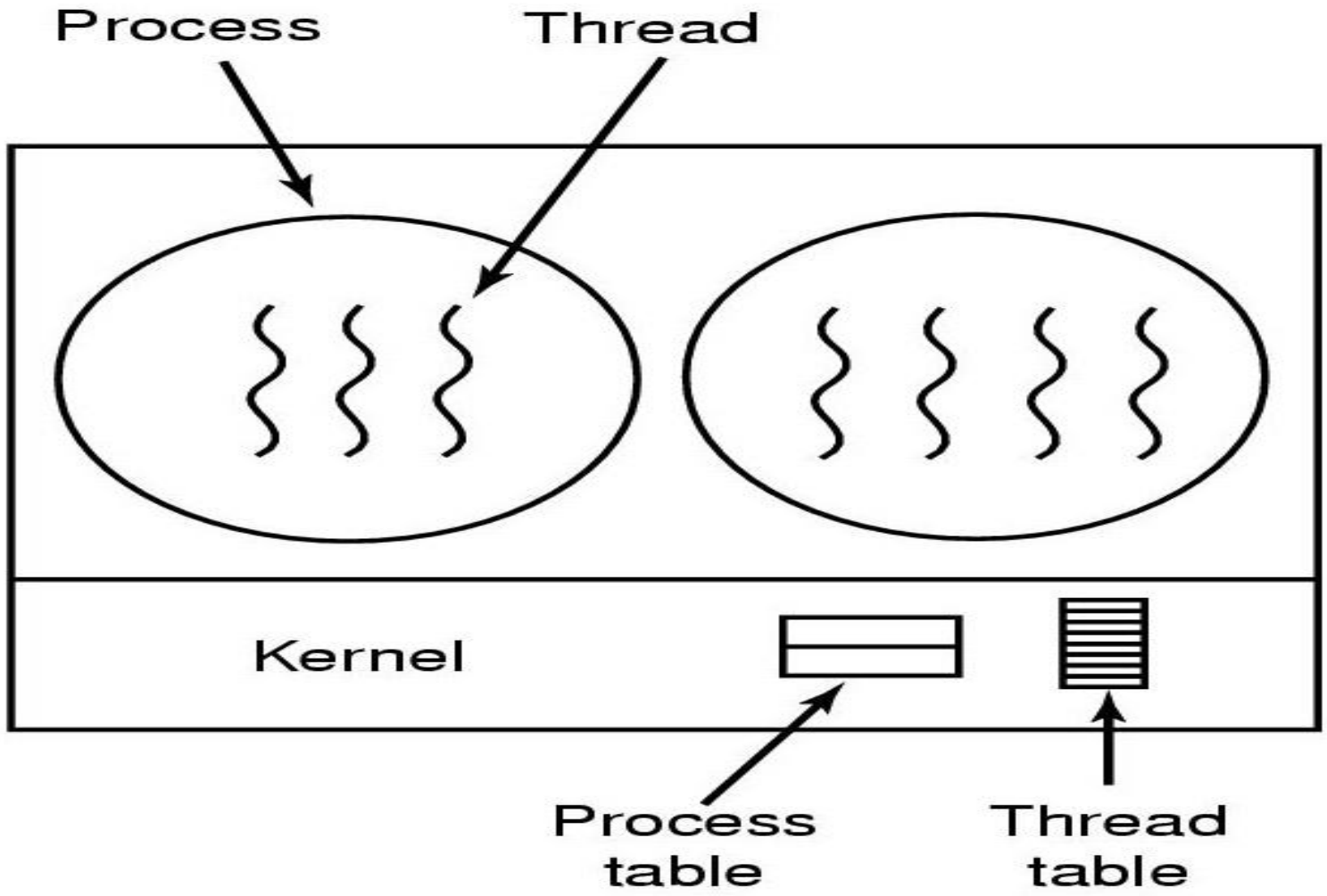


Common Thread Strategies

- Manager/worker
  - Manager thread handles I/O
  - Manager assigns work to worker threads
  - Worker threads created dynamically
  - ... or allocated from a *thread-pool*
- Pipeline
  - Each thread handles a different stage of an assembly line
  - Threads hand work off to each other in a *producer-consumer* relationship

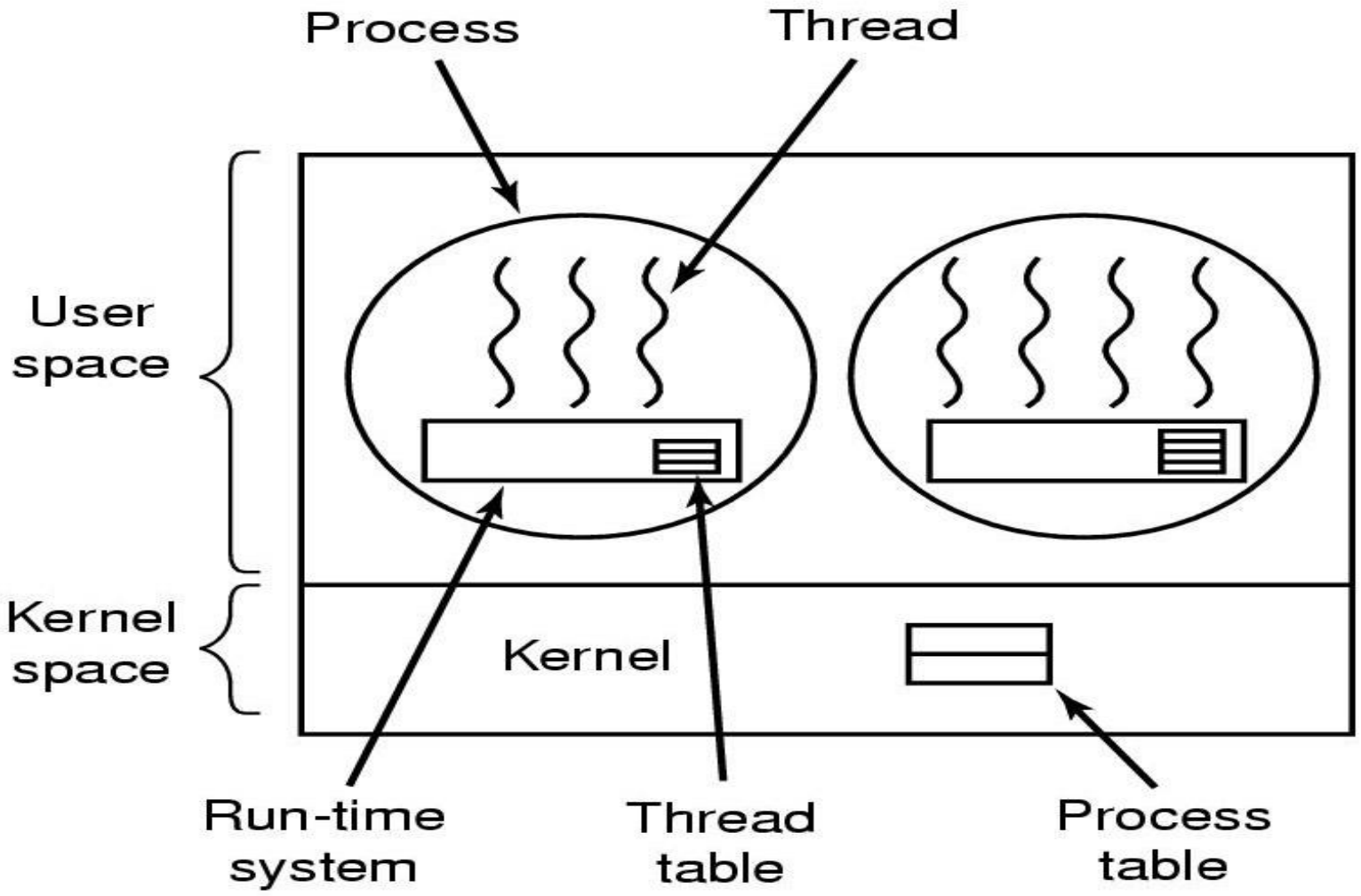
Kernel-Level Threads

Thread-switching code is in the kernel



User-Level Threads Package

The thread-switching code is in user space



Thread Libraries

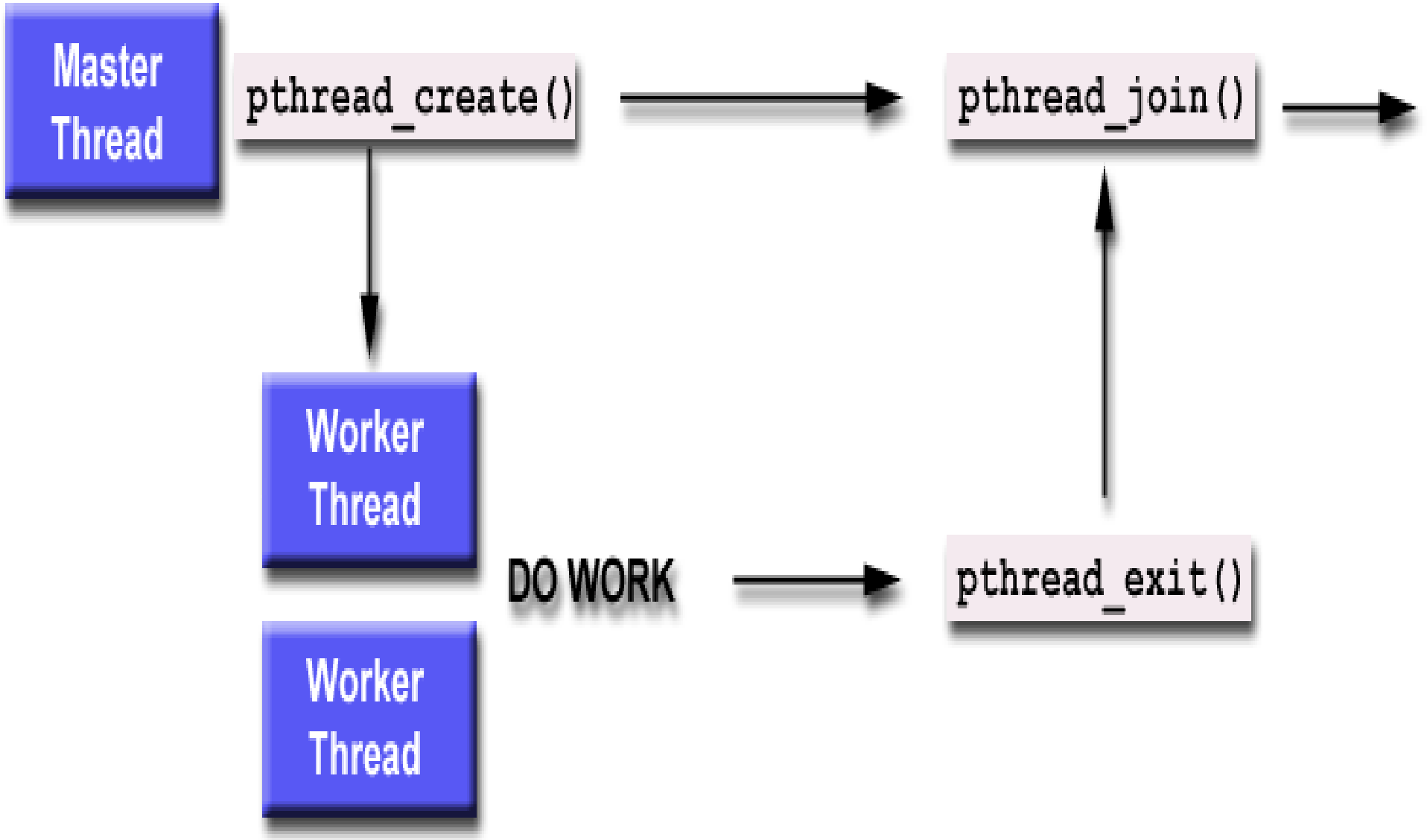
A thread library provides the programmer with an API for creating and managing threads

- Provide a library entirely in user space with no kernel support
- Implement a kernel-level library supported directly by the operating system
  - Code and data structures for the library exist in kernel space
  - Invoking a function in API for the library results in a system call to the kernel

pthread

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)
- First thread in main creates the others
  - pthread_create()
  - pthread_join()
  - pthread_exit()

Using Create, Join and Exit



Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Implementing the Runnable interface

```
public interface Runnable
{
    public abstract void run();
}
```