Threads

Consider

- A Process is a unit of:
  - Resource ownership
    - Address space
    - I/O channels, devices, files
  - Execution path
    - Interleaved with other processes
    - State
- What if we treat each independently?
  - Unit of resource ownership → process
  - Unit of execution → thread

Process Context Switch

Process Context switch:

allocate CPU from one process to another.

A Process context includes two portions:

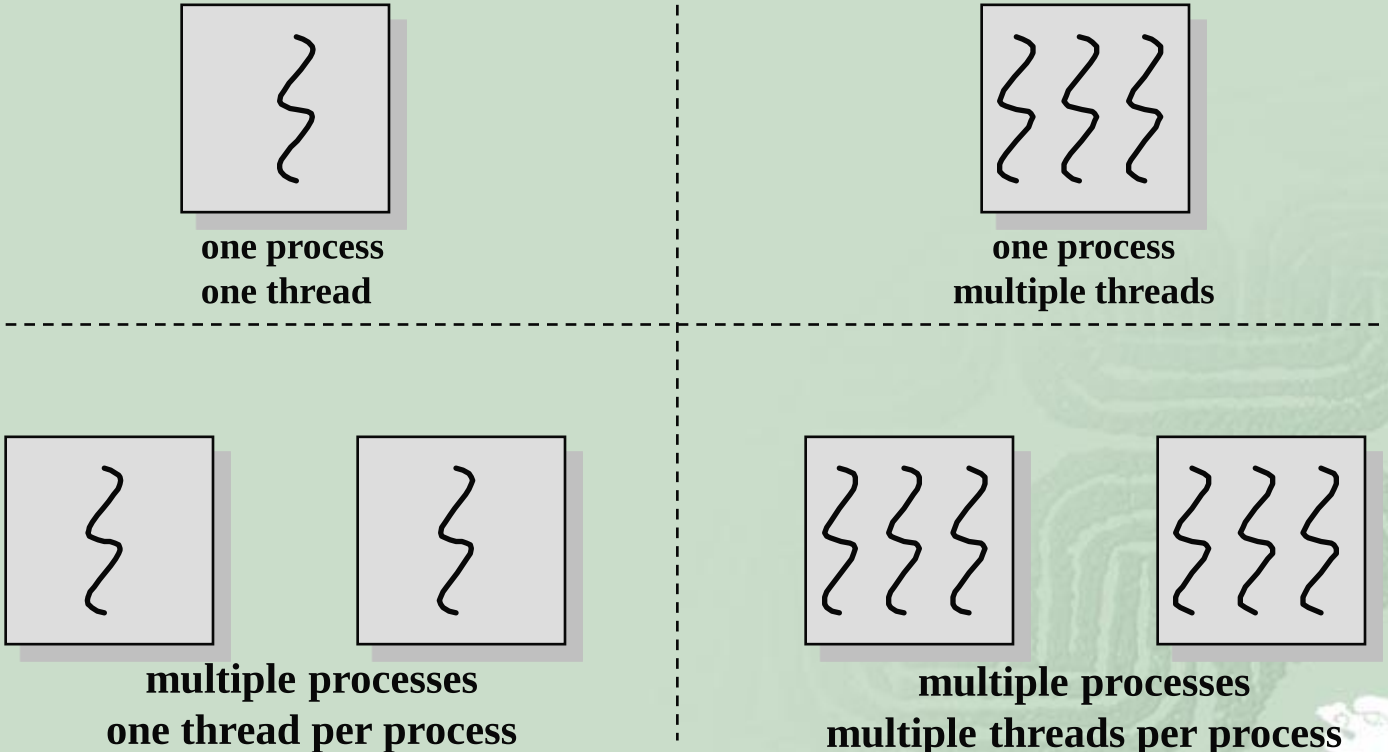
CPU context and Storage context.

- CPU context: program counter, registers, stack/heap pointers and other control registers. Easy to switch.
- Storage context: program code, data, address space, memory mapping, (disk) swapping, resources, etc. Hard and time consuming to switch.

Multi-threading

- Operating system supports multiple threads of execution within a single process.
  - Traditional approach is single-threaded.
- Examples:
  - MS-DOS supports a single user process and a single thread.
  - UNIX supports multiple user processes but only supports one thread per process.
  - Java run-time environment is a system of one process with multiple threads.
  - Windows 2000 (W2K), Solaris, Linux, Mach, and OS/2 support multiple processes, each of which supports multiple threads.

Threads and Processes



Threads

- One view of a thread is as an independent program counter operating within a process.

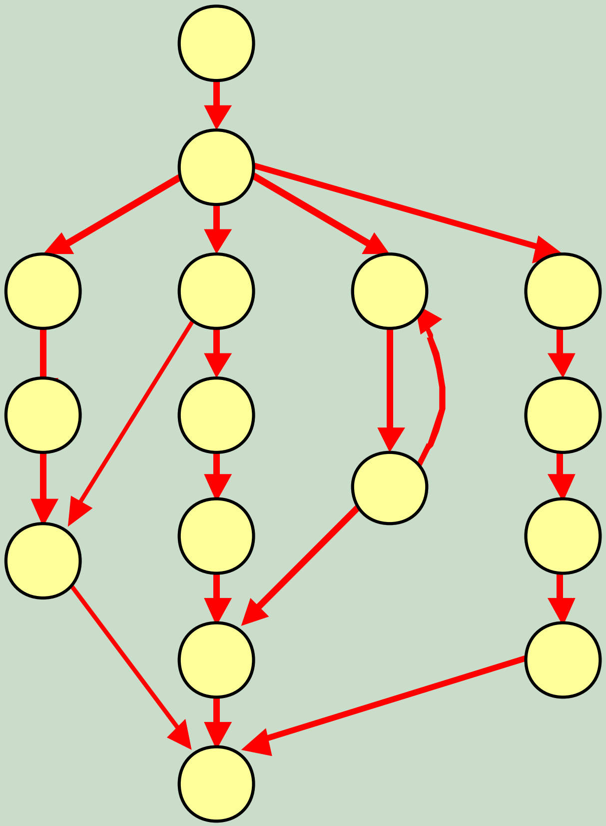
- A thread consists of:

  - a thread execution state (Running, Ready, etc.)
  - a CPU context (program counter, register set.)
  - an execution stack.
  - access to the memory and resources of its process (shared with all other threads in that process.)

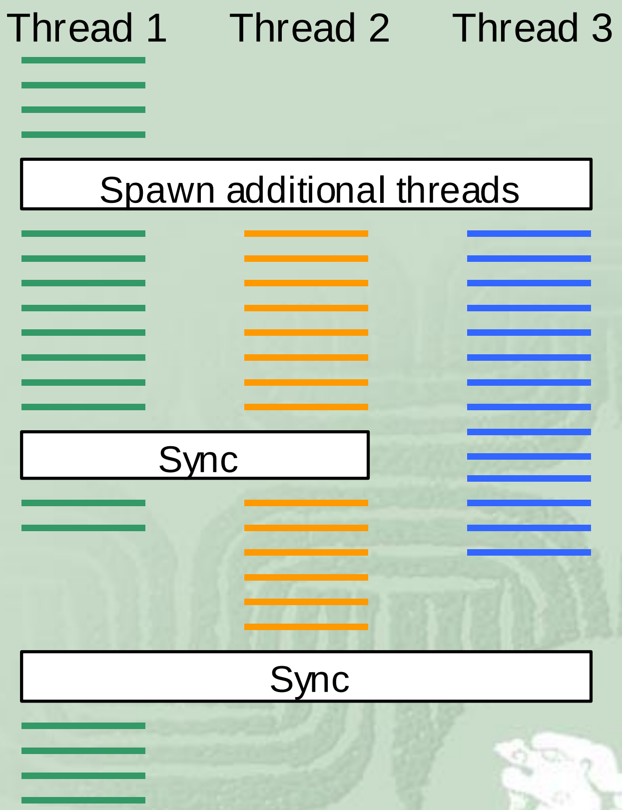
Threads (continued…)

- Thus, all of the threads of a process share the state and resources of the parent process (memory space and code section.)
- A process is defined to have at least one thread of execution (the process itself) and may launch other threads which execute concurrently with the process.
- Key benefits:
  - Far less time to create/terminate.
  - Switching between threads is faster.
  - No memory management issues, etc.
  - Can enhance communication efficiency.

Threads (continued…)

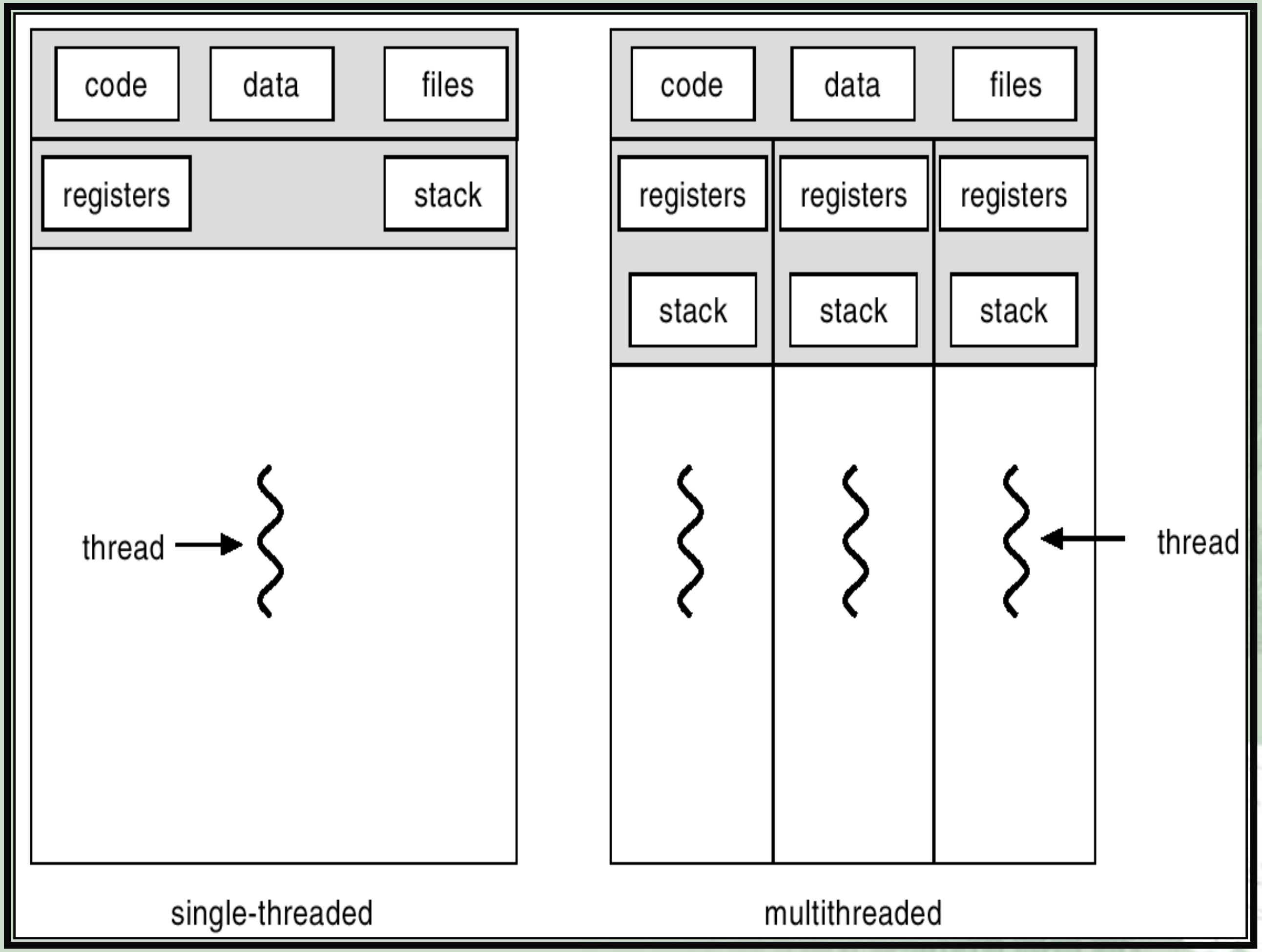


(a) Task graph of a program



(b) Thread structure of a task

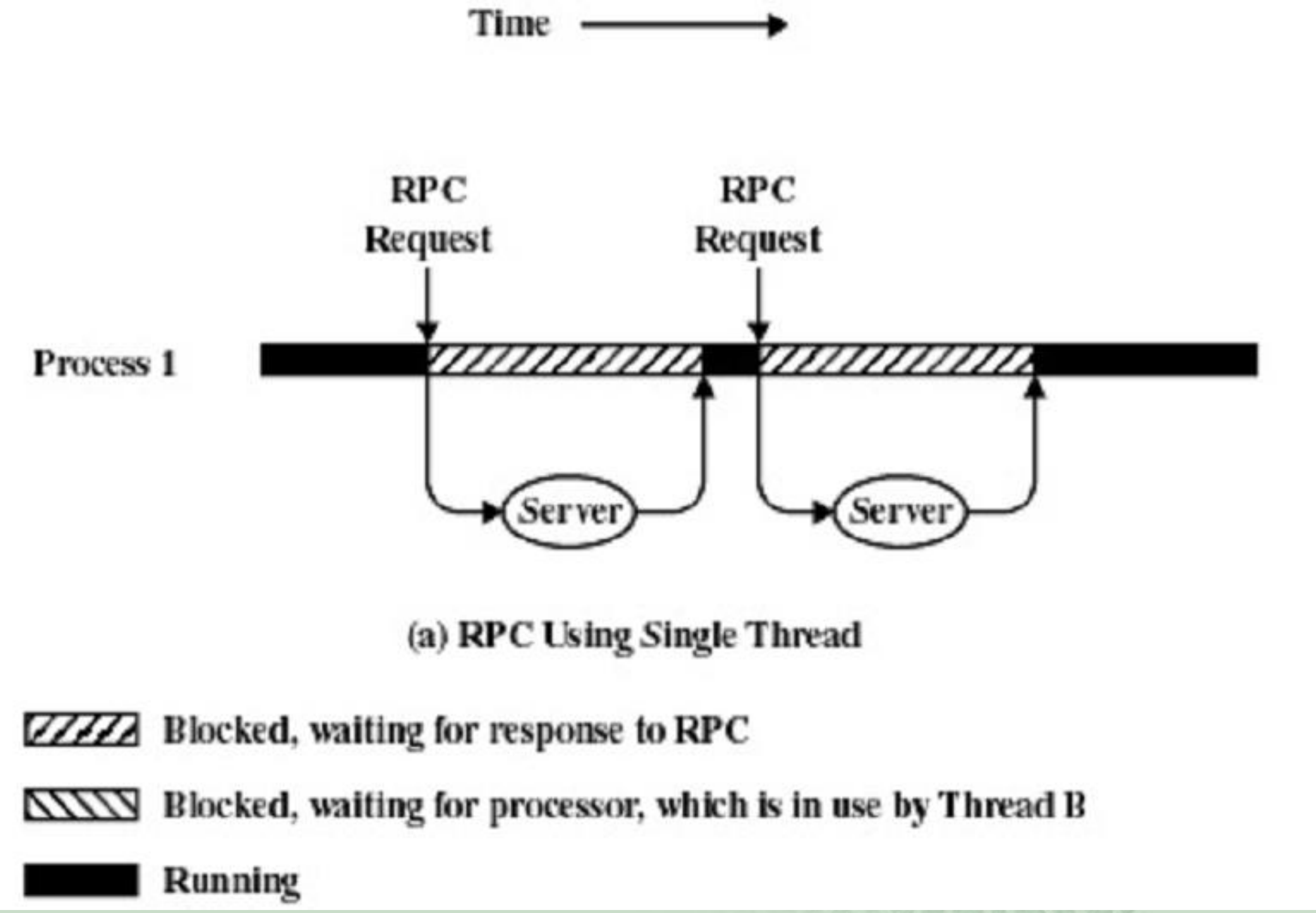
Single and Multithreaded Processes



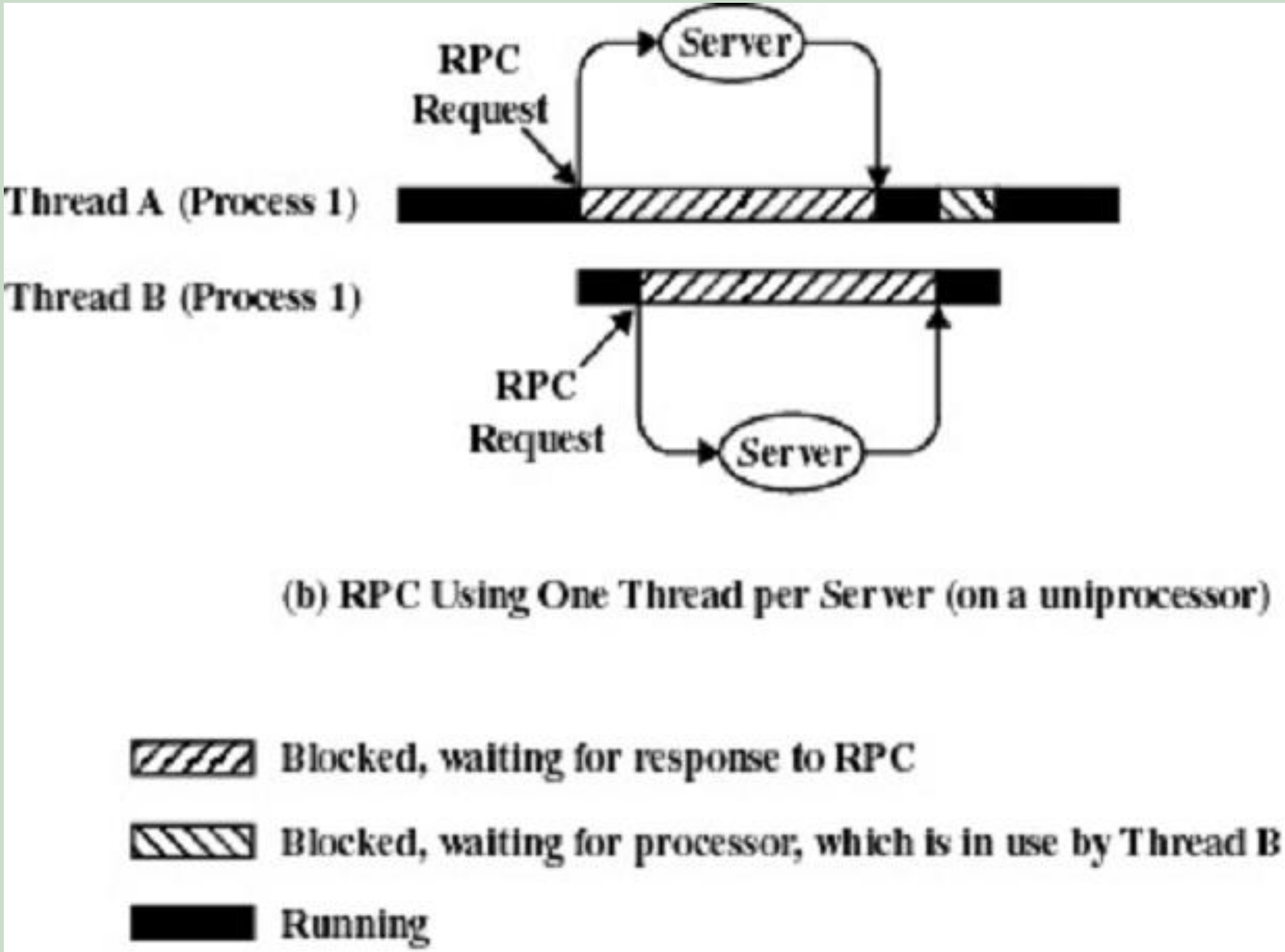
Using Threads

- Multiple threads in a single process
  - Separate control blocks for the process and each thread
  - Can quickly switch between threads
  - Can communicate without invoking the kernel
- Examples
  - Producer/Consumer
  - Concurrent services
  - Faster Execution – Read one set of data while processing another set
  - GUI and worker

Remote Procedure Calls

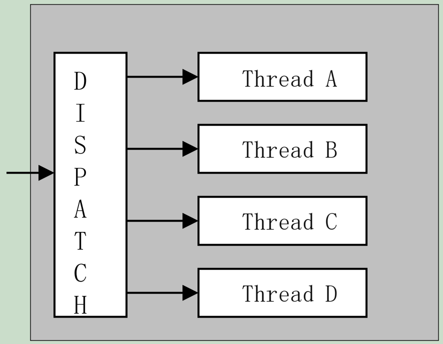


Remote Procedure Calls Using Threads

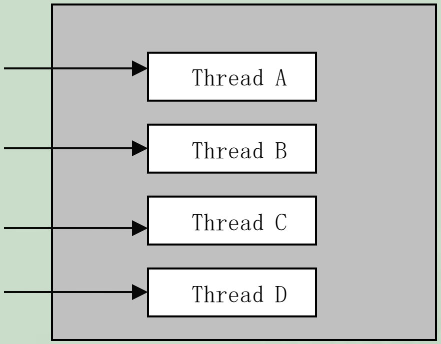


(b) RPC Using One Thread per Server (on a uniprocessor)

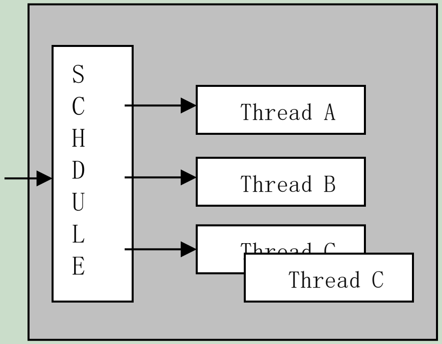
Design of Concurrent Server



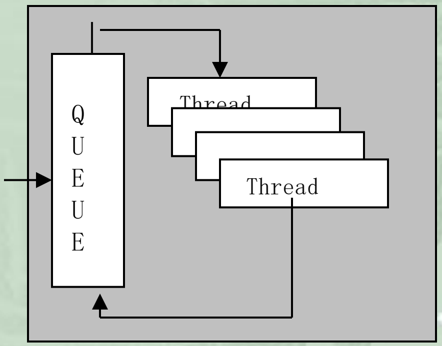
(a) Center distributor



(b) Concurrent threads



(c) Center scheduler



(d) Round-robin schedule

Thread States

- **Thread operations**
  - *Spawn* – Creating a new thread
  - *Block* – Waiting for an event
  - *Unblock* – Event happened, start new
  - *Finish* – This thread is completed
- **Generally a thread can block without blocking the remaining threads in the process**

Thread issues

- How should threads be scheduled compared to processes?
  - Equal to processes
  - Within the parent processes quantum
- How are threads implemented?
  - kernel support (system calls)
  - user level threads

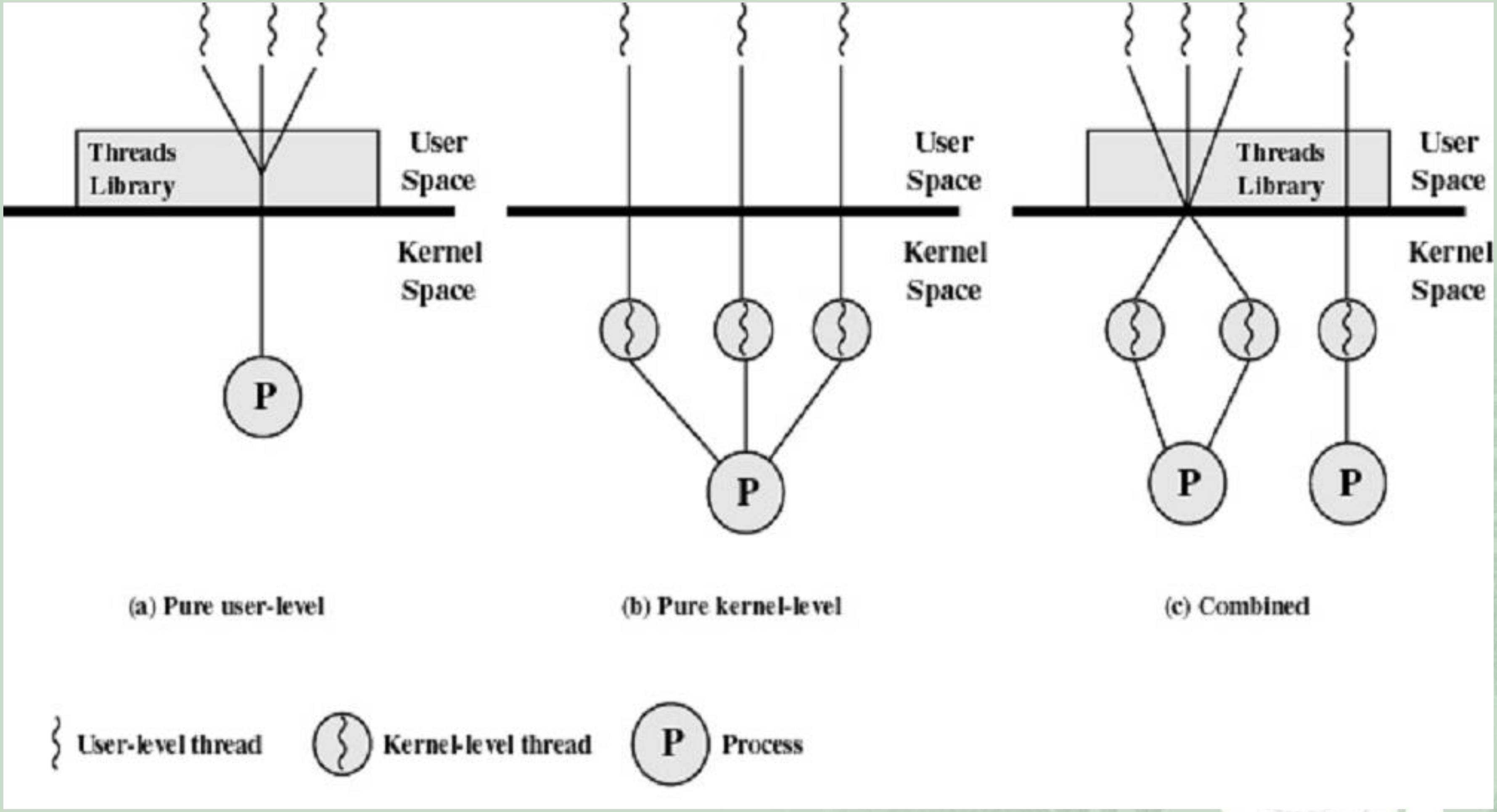
User-Level Threads

- All thread management is done by the application
- The kernel is not aware of the existence of threads
- Thread switching does not require kernel mode privileges
- Scheduling is application specific

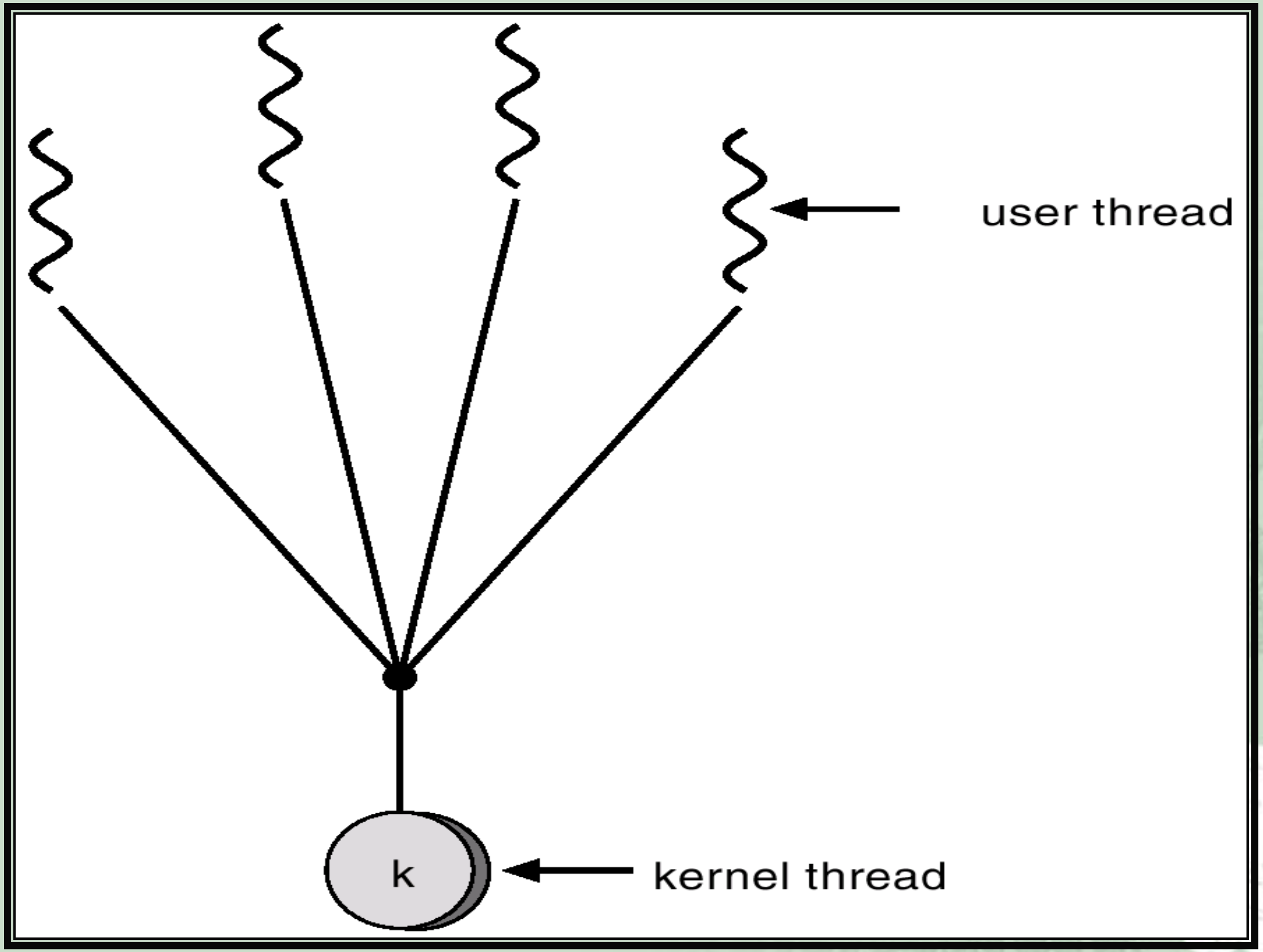
Kernel-Level Threads

- Kernel maintains context information for the process and the threads
- Switching between threads requires the kernel
- Windows NT and OS/2 are examples of this approach

User-Level and Kernel-Level Threads



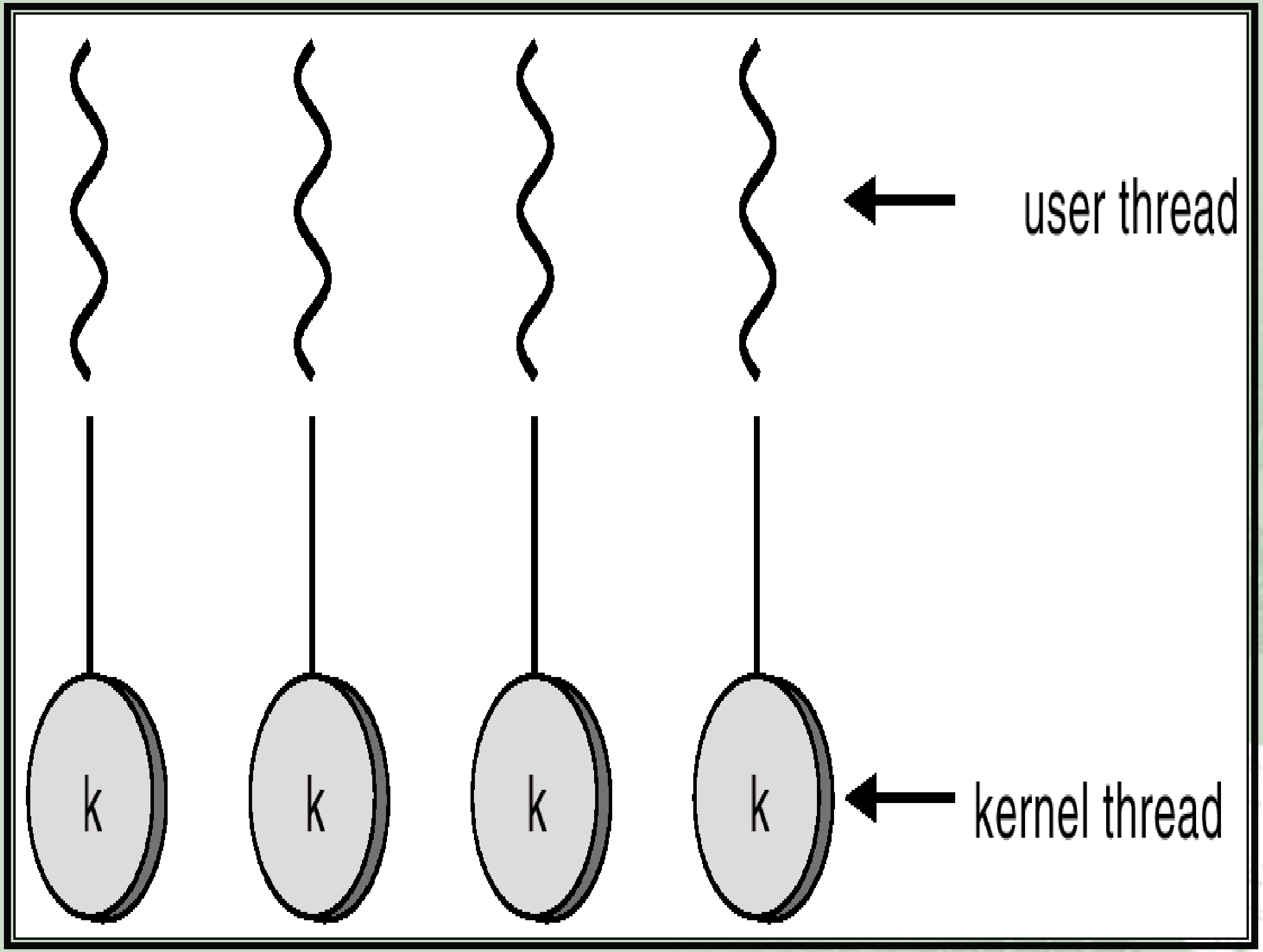
Many-to-One Model



Many-to-One

- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.
- Example: GNU Portable Threads

One-to-one Model



One-to-One

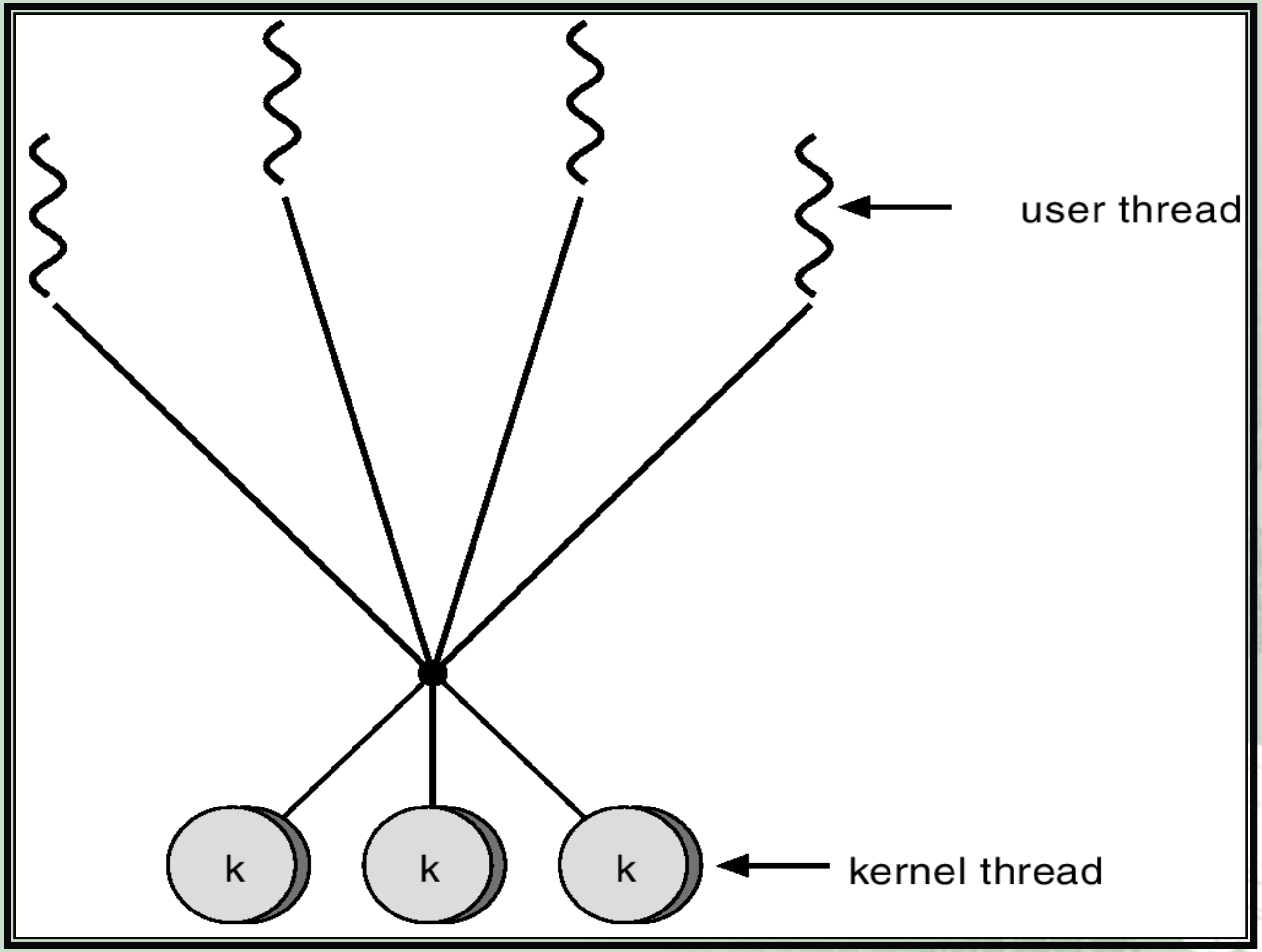
- Each user-level thread maps to kernel thread.

- Examples

  - Windows 32

  - old Linux, Solaris

Many-to-Many Model



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 7