Sistemas Operativos 1

Threads

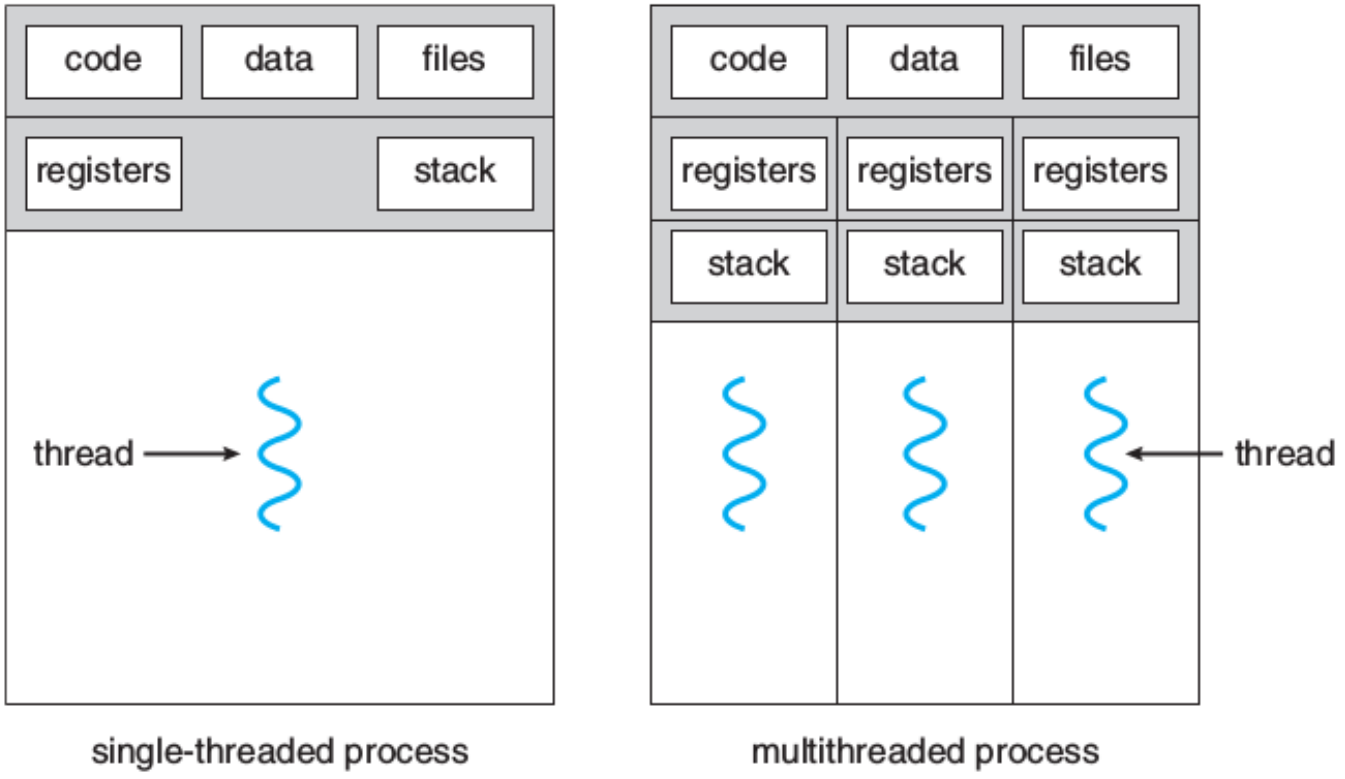
Threads (hilos de ejecución)

- Un thread es una **unidad básica de utilización de CPU**; incluye:
  - Thread ID
  - Program counter
  - Register set
  - Stack

- Un thread **comparte** con otros threads del **mismo proceso**:
  - Code section,
  - Data section,
  - Recursos del sistema operativo, por ej:
    - open files
    - signals.

Threads (hilos de ejecución)

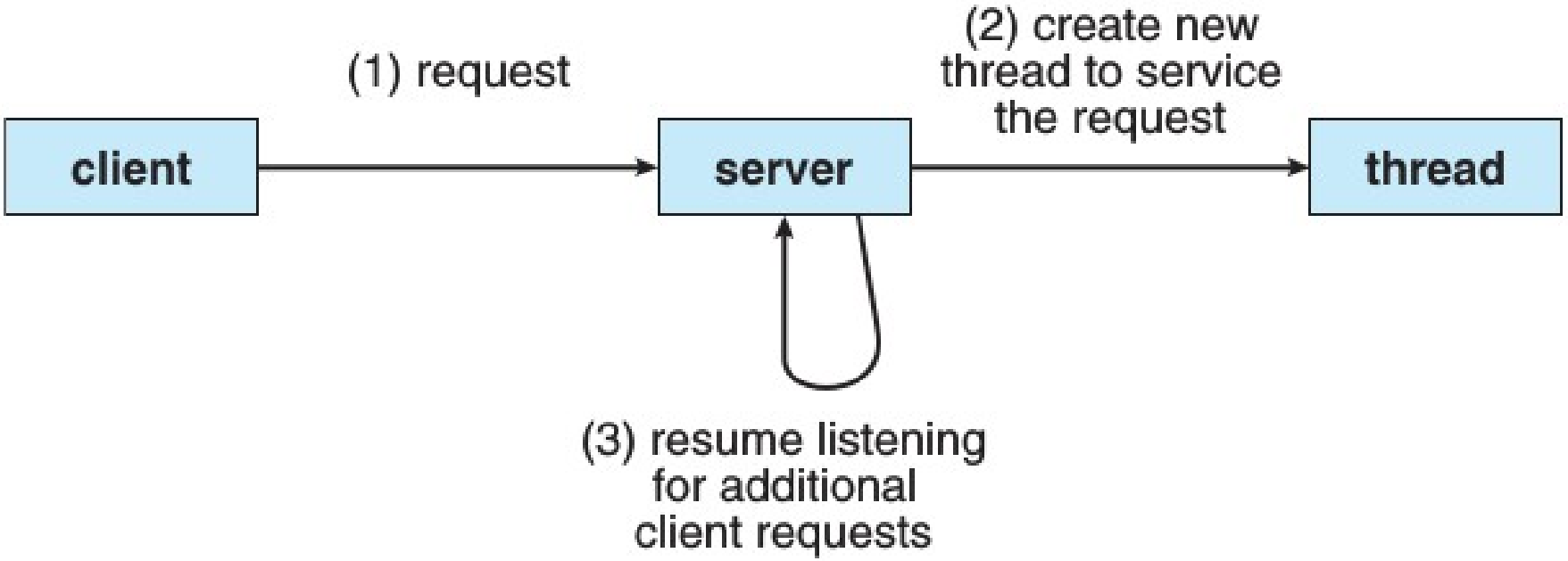
- Un proceso tradicional (o “heavyweight”) tiene un único thread de control.
- Si un proceso tiene múltiples threads de control, éste pude hacer mas de una tarea al mismo tiempo.



single-threaded process

multithreaded process

Servidor Multithreaded



Beneficios del Multithread

- **Respuesta** – permite a un programa continuar ejecutando aún cuando parte está bloqueada en una operación larga (ej. un web browser cargando una imagen)
- **Compartir Recursos** – los threads comparten memoria y otros recursos del proceso
  - **Economía** – es más económico crear y cambiar de contexto threads (comparten los recursos del proceso al cual pertenecen)
    - En Solaris crear un proceso es 30 veces más lento que crear un thread y 5 veces más lento en cambiar de contexto
- **Escalabilidad** (en arquitecturas multi-procesador) – el mismo proceso puede usar varias CPUs simultáneamente utilizando threads

Multicore Programming

Los sistemas multicore imponen nuevos desafíos a los programadores:

- **Dividir actividades:** encontrar partes del programa que pueden ser divididas en tareas concurrentes separadas y de esta forma ejecutarlas en paralelo en distintos cores

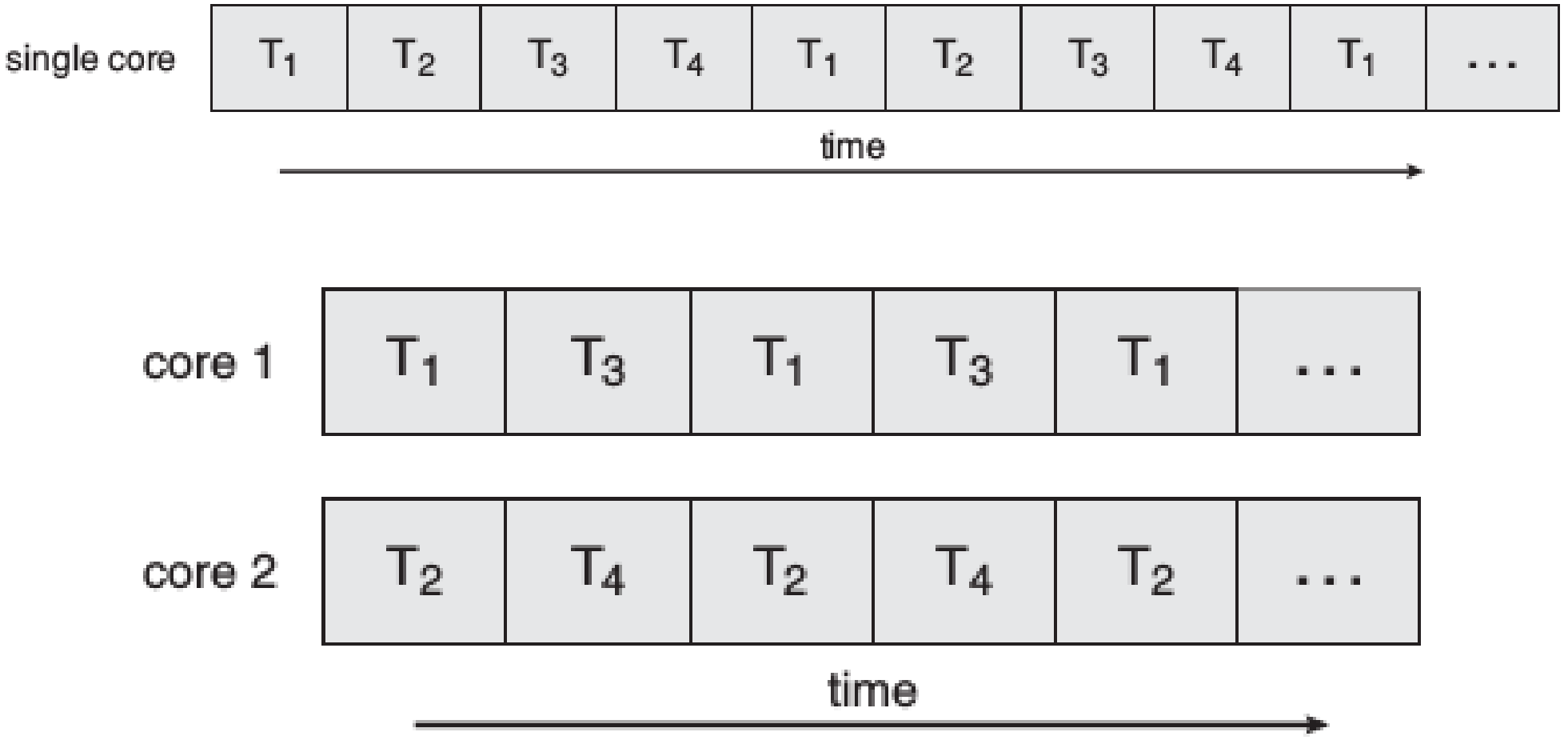
- **Balanceo**: asegurar que cada tarea ejecuta trabajo del mismo valor

- **División de datos:** asegurarnos que los datos se dividen para ser procesados en cores separados como las tareas.

- **Dependencia de datos:** asegurar que la ejecución de tareas está sincronizada para obedecer a la dependencia de datos

- **Testing y debugging:** es mucho más difícil que en aplicaciones single-threaded dado que hay muchos caminos de ejecución diferentes cuando el programa ejecuta

Single Core vs Multicore



Kernel Threads & User Threads

- Los **kernel threads** son creados y planificados por el **kernel**:
  - Son más caros de crear que un user thread
- Los **user threads** son creados y planificados por una librería de threading que se ejecuta en modo usuario.
  - Todos los user threads pertenecen al proceso que los creo.
  - El kernel **no sabe** de la existencia de los user threads
- La mayor diferencia se da cuando se usan sistemas multiprocesador:
  - Los user threads son completamente manejados por la librería, no pueden ejecutar en paralelo en diferentes CPUs
  - Dado que los kernel threads son planificados por el kernel, distintos threads pueden ejecutar en distintas CPUs

Los kernel threads son los únicos threads considerados por el Scheduler de CPU.

Multithreading Models

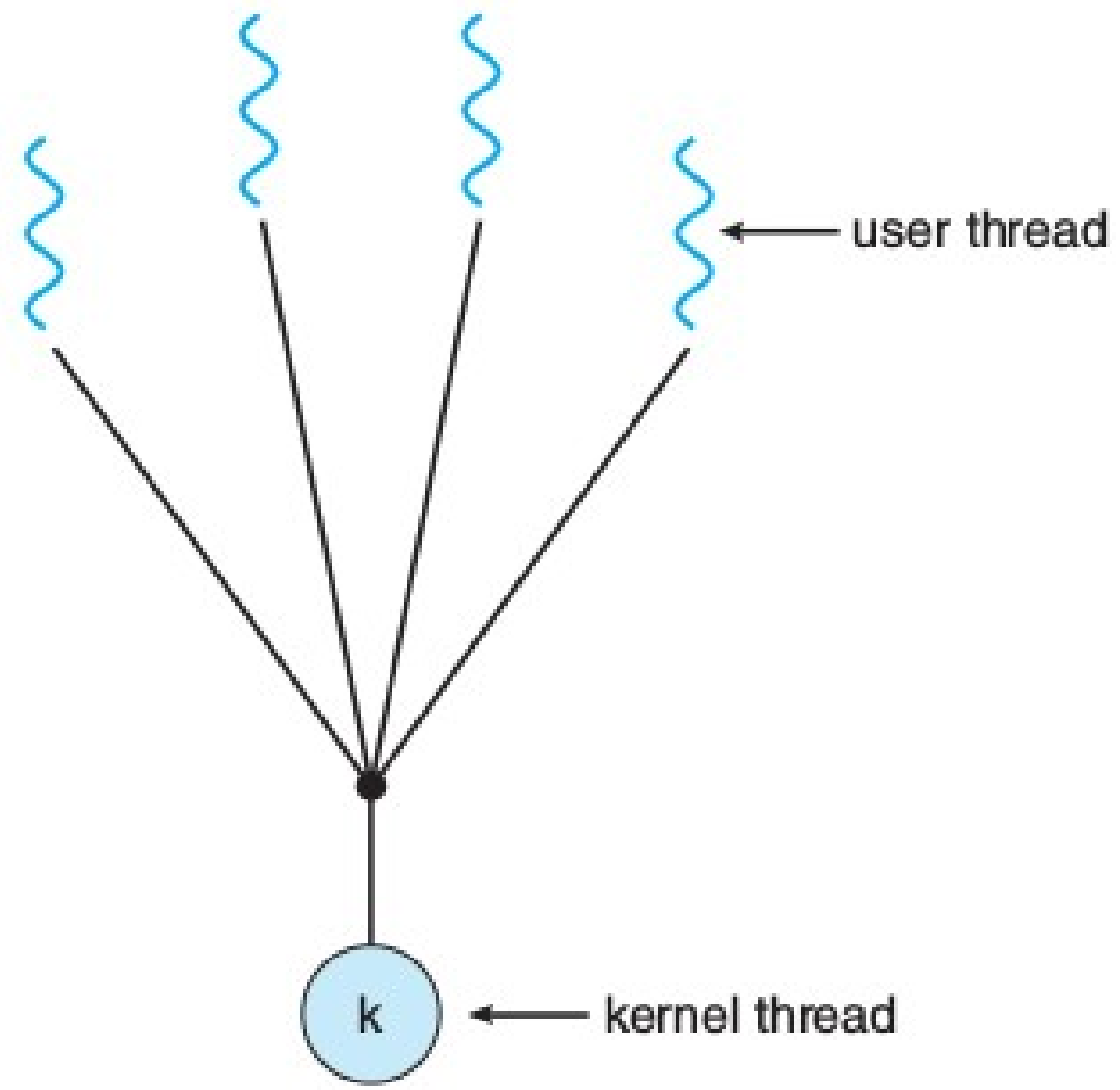
- Many to One
- One to One
- Many to Many
- Two Level

Many to One Model

- Muchos user threads son mapeados a un único kernel thread
- El manejo de los threads lo hace la librería en espacio del usuario.
- **Desventaja** – el proceso entero se bloquea cuando el thread hace una llamada al sistema bloqueante
- Ejemplos:
  - Solaris Green Threads
  - GNU Portable Threads

Green threads, a thread library available for Solaris systems and adopted in early versions of Java, used the many-to-one model.
However, very few systems continue to use the model because of its inability to take advantage of multiple processing cores

Many to One Model

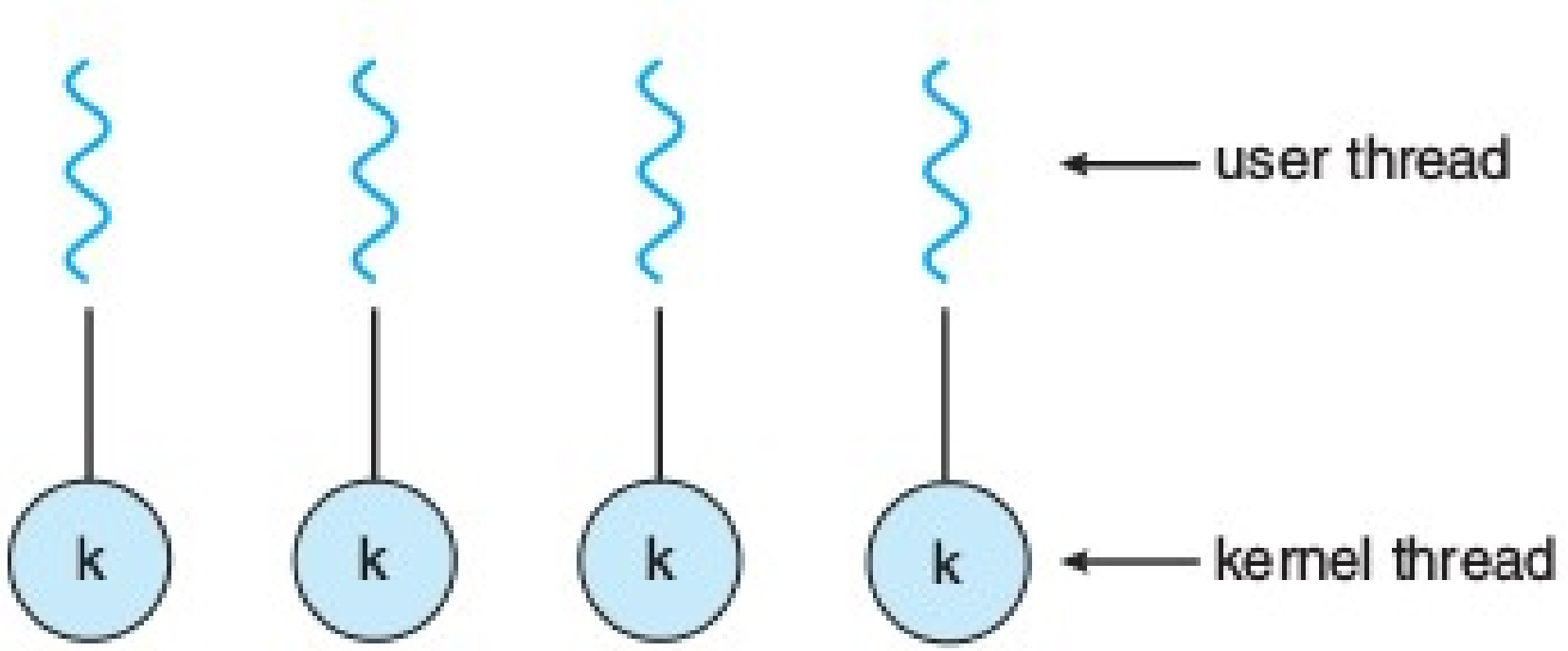


One to One Model

- Cada user thread es mapeado a un kernel thread
- Más concurrencia comparado al modelo many-to-one dado que otro thread puede ejecutar cuando uno hace una llamada bloqueante
- Permite la ejecución en paralelo de múltiples threads en multiprocesadores
- **Desventajas**:
  - Crear un user thread requiere la creación del correspondiente kernel thread
  - Si se crean demasiados kernel threads, la performance se va a degradar, con lo cual la mayoría de las implementaciones de este modelo tiene una restricción en el número de threads permitidos
    - Ej en linux: cat /proc/sys/kernel/threads-max

- Ejemplos
  - Windows NT/XP/2000 y posteriores, Linux, Solaris 9 y posteriores

One to One Model

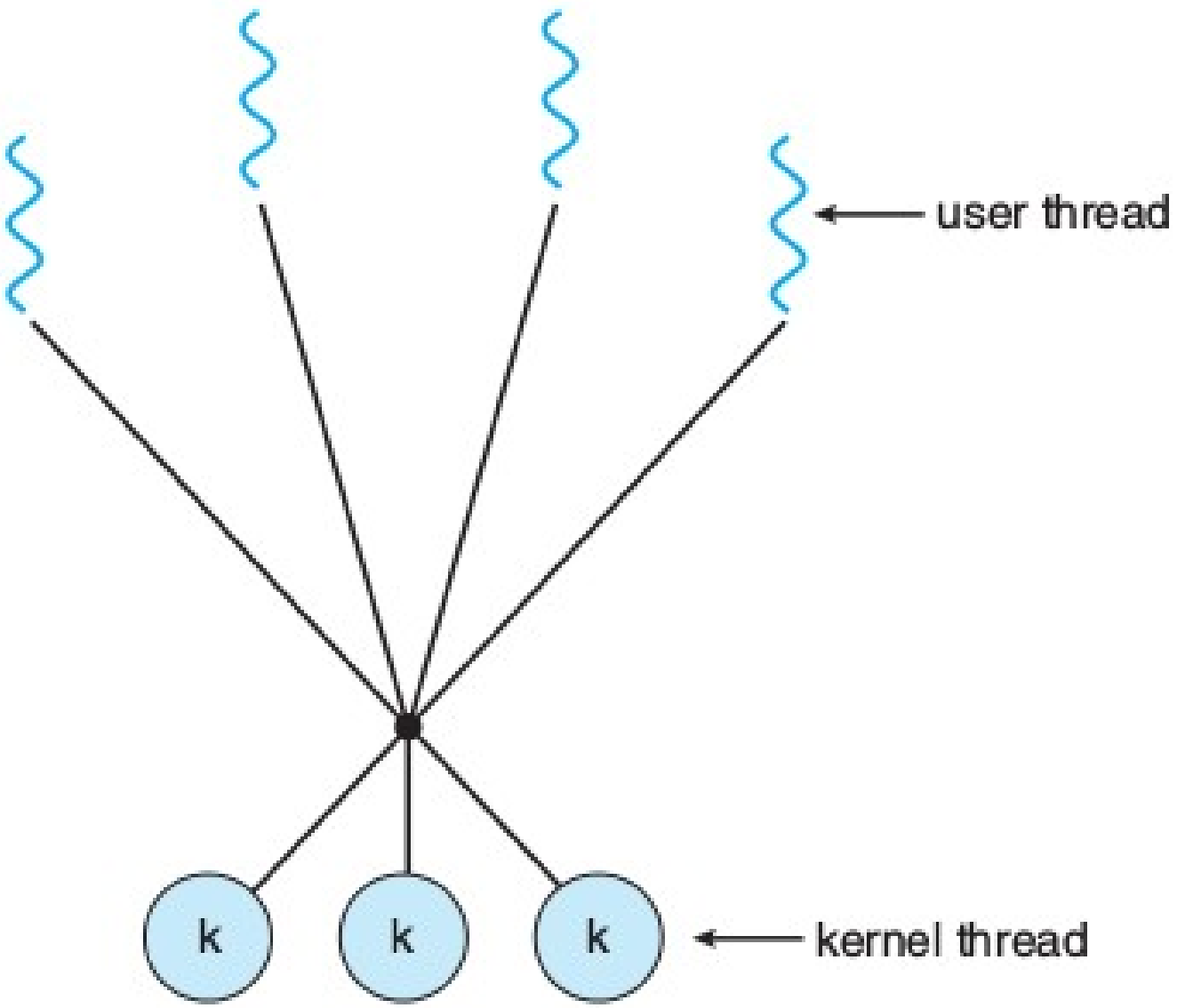


Many to Many Model

- Muchos user threads pueden ser mapeados a un menor o igual número de kernel threads
- Permite al SO crear la cantidad suficiente de kernel threads
- Resuelve las desventajas de los dos modelos anteriores:
  - Los developers pueden crear tantos user threads como quieran
  - Los kernel threads corren concurrentemente (aunque un thread se bloquee, otro thread puede ser scheduleado para la ejecución)

- Ejemplos:
  - Solaris previo a la versión 9
  - Windows NT/2000 con el paquete ThreadFiber

Many to Many Model

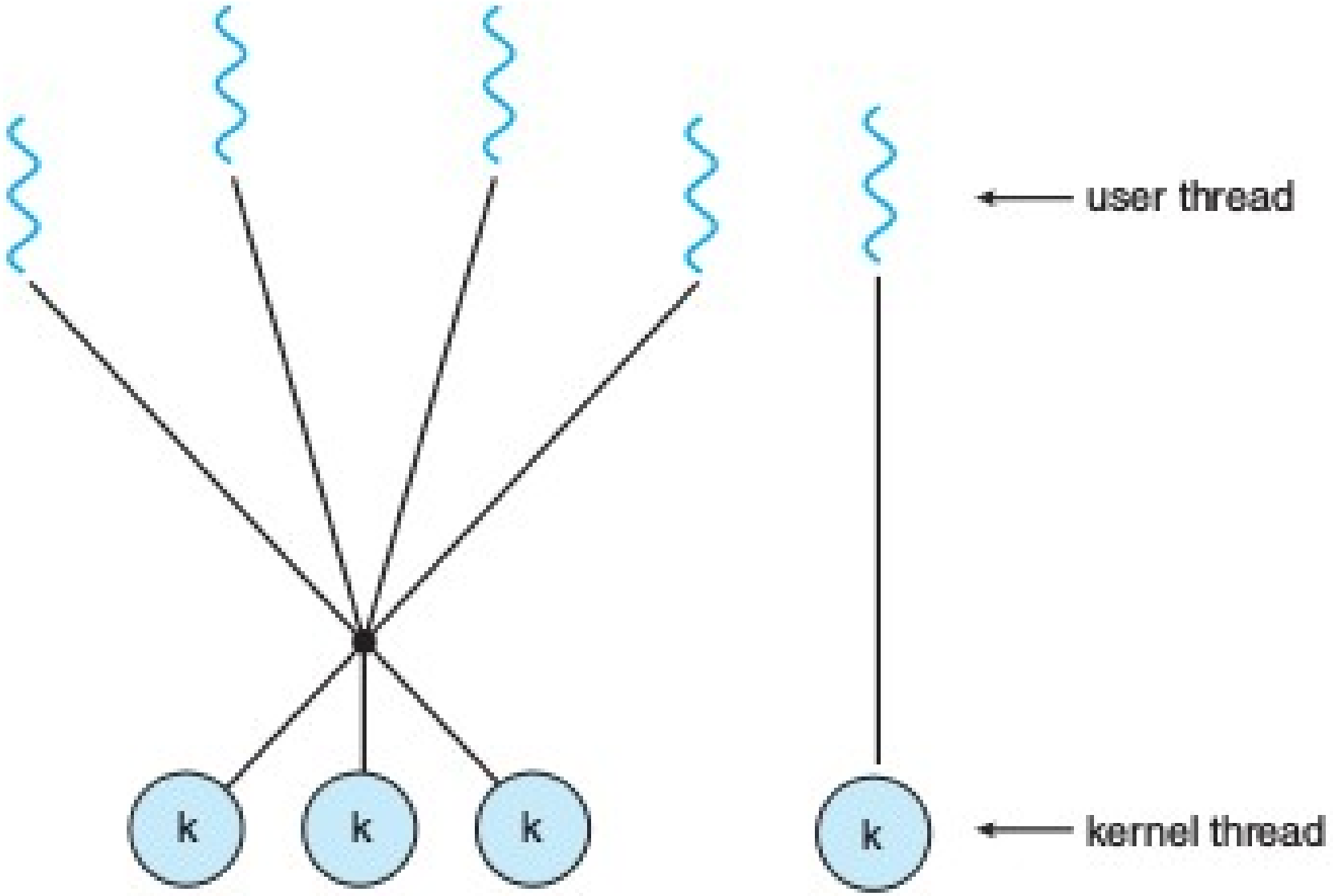


Two Level Model

- Similar al modelo Many-to-Many, excepto que permite a un user thread estar asociado a un kernel thread fijo

- Ejemplos
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 y anteriores

Two Level Model



Thread Libraries

- Provee al programador de una API para crear y administrar threads
- Dos tipos de implementación:
  - **User Space**: Código y data en user space, Las invocaciones son funciones locales (no son system calls)
  - **Kernel-level**: una Kernel Library soportado por el S.O. Código y data en kernel space, Las invocaciones son system calls

Thread Libraries

Las implementaciones principales son :

- **Pthreads**: Extensiones del standard **POSIX**, puede ser provista como librería user-level o kernel-level

- **Windows thread library**: es una libreria kernel-level disponible en los S.O. Windows.

- **Java Thread API**: Permite crear threads y ser administrados directamente dentro de los programas JAVA.
  - Como la JVM corre sobre un S.O, generalmente es implementada usando la librería de threads provista por el S.O host (huésped)

Thread Pools

- Crear un número de threads en un pool y esperar trabajo
- Ventajas:
  - Usualmente más rápido satisfacer un pedido que al crear un nuevo thread
  - Permite poner un límite al número de threads que utiliza una aplicación
- La creación de threads ilimitada puede sobrecargar el sistema, como tiempo de CPU y memoria