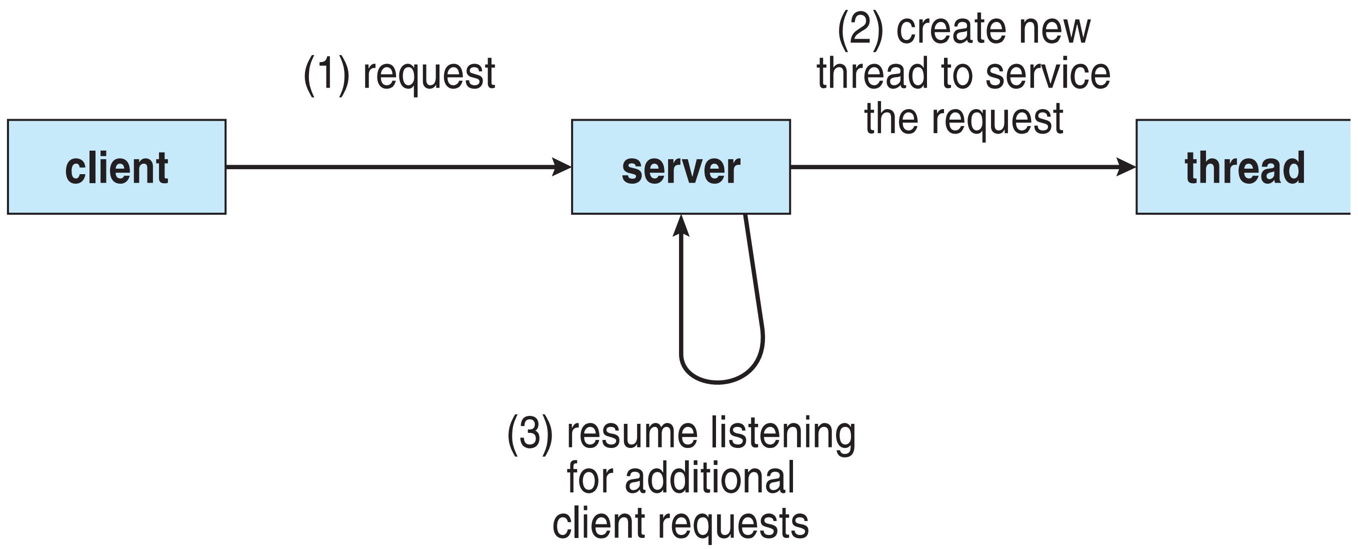
Threads

Background

Motivation

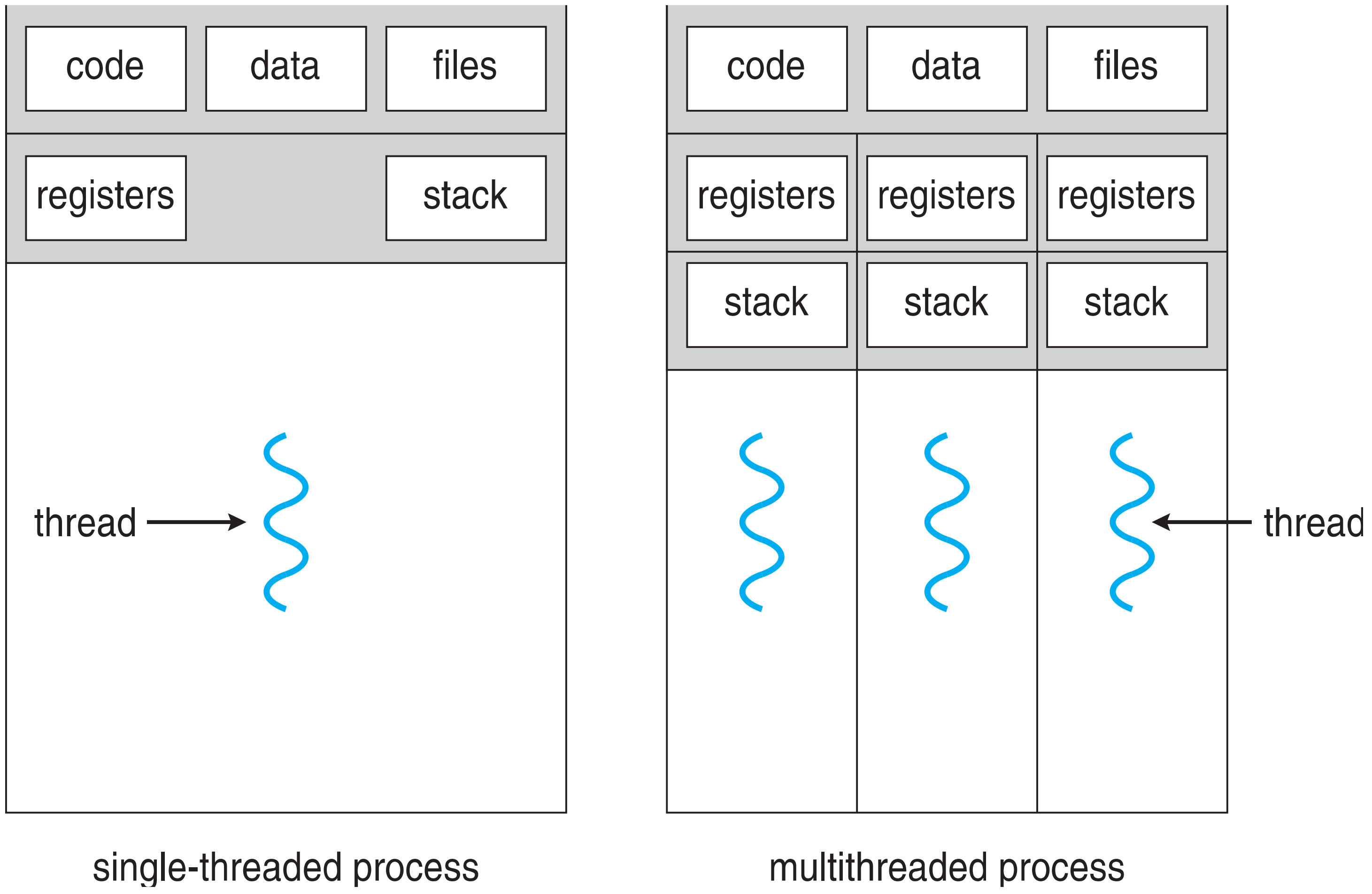
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Chrome browser
    - Fetcher
    - Renderer
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Multithreaded Server Architecture


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

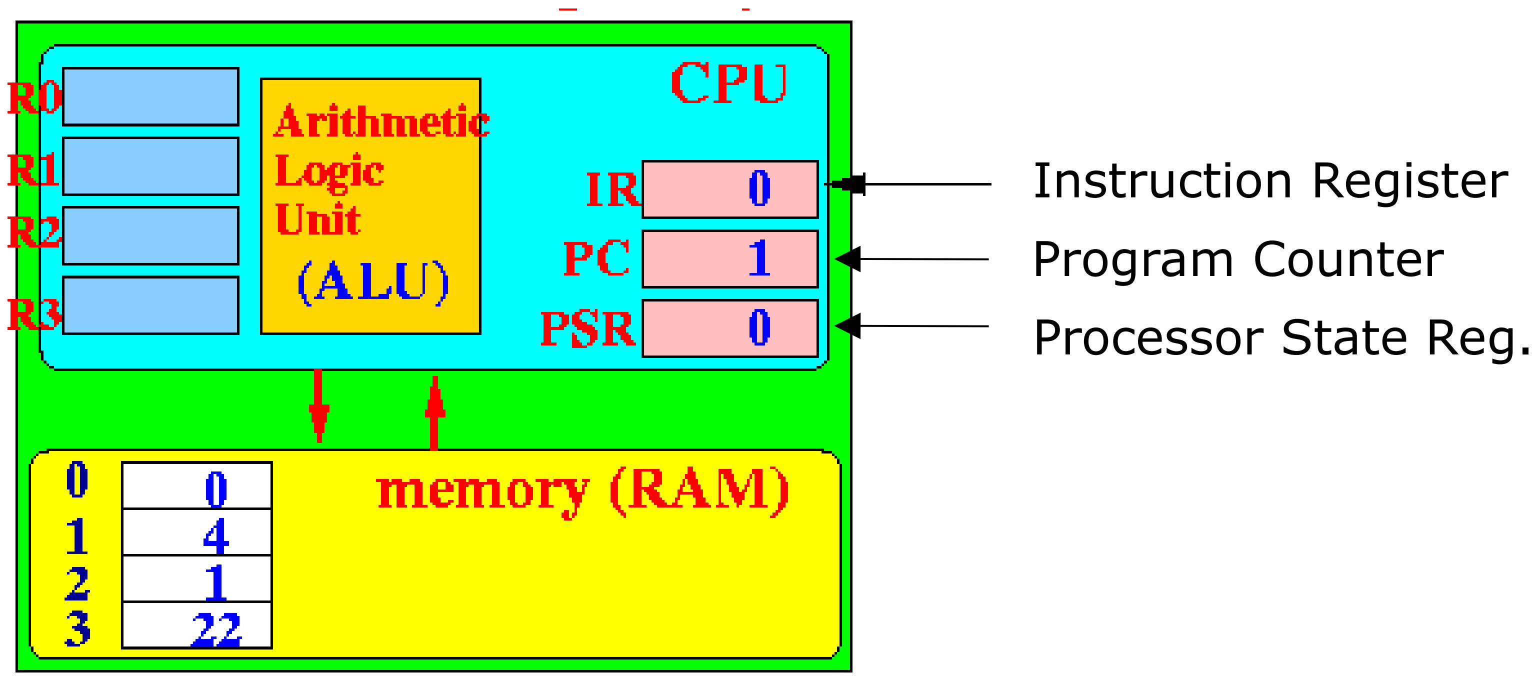
What are threads?

Single and Multithreaded Processes



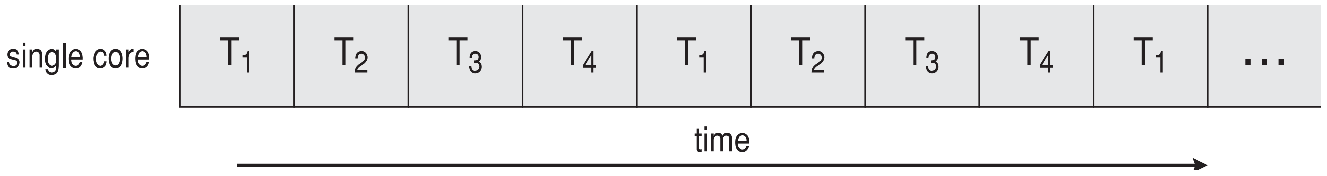
Relating it to Assembly

- Each thread has a unique set of registers
- Memory can be shared among threads
- In some situations memory may be completely thread specific

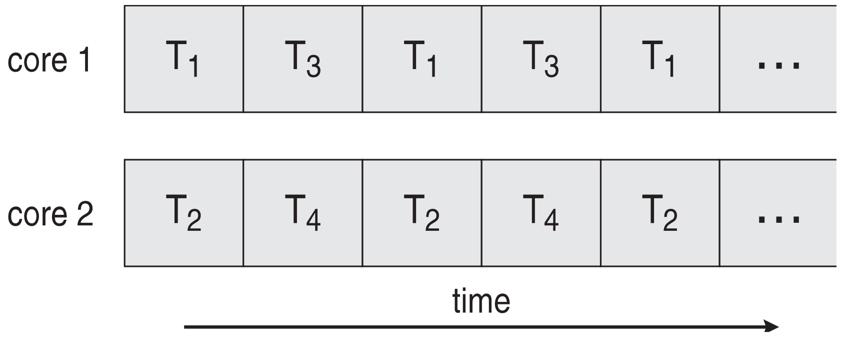


Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectures

Thread Properties

Multicore Programming

- **Multicore** or **multiprocessor** systems offer real speed up if multiple threads are used
- Programmers must accurately use threads
  - **Divide** independent problems and assign to threads
  - Keep a **balance** in dividing activities, do not overburden one or more threads
  - **Split Data** too**,** so threads can work in parallel
  - **Data** should be split properly, so that **dependency** among threads remains minimum
  - **Testing and debugging** of the multi threaded applications is difficult
- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** enables more than one task making progress

User Threads and Kernel Threads

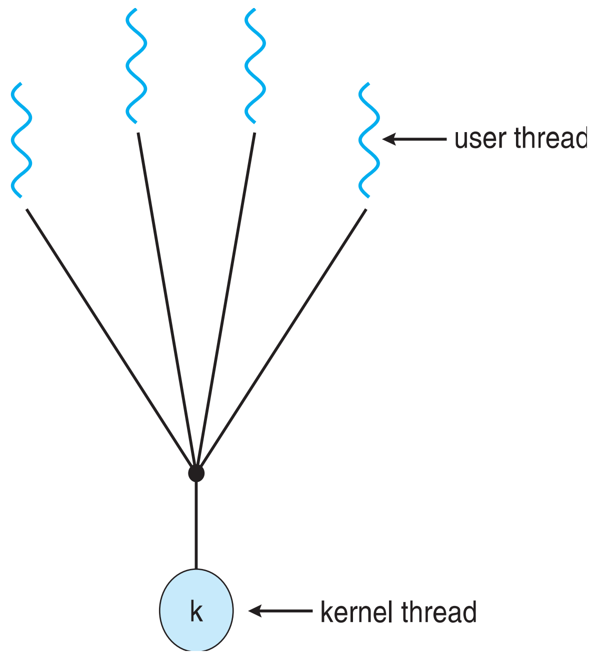
- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

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

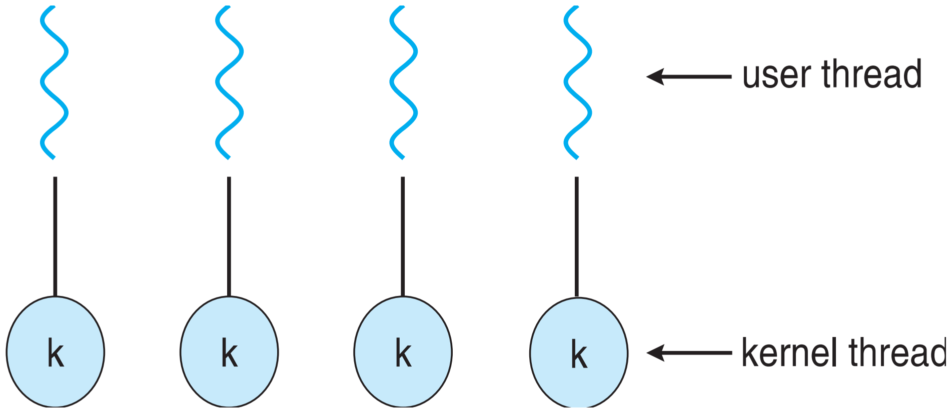
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



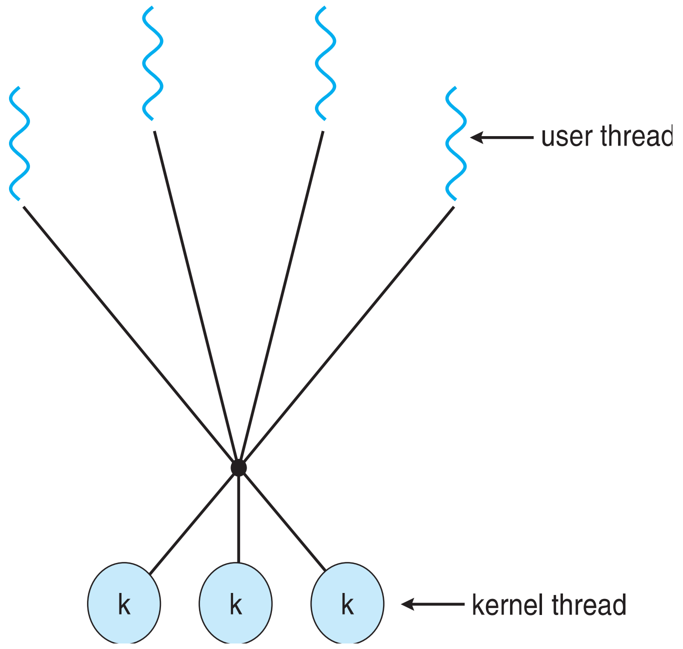
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



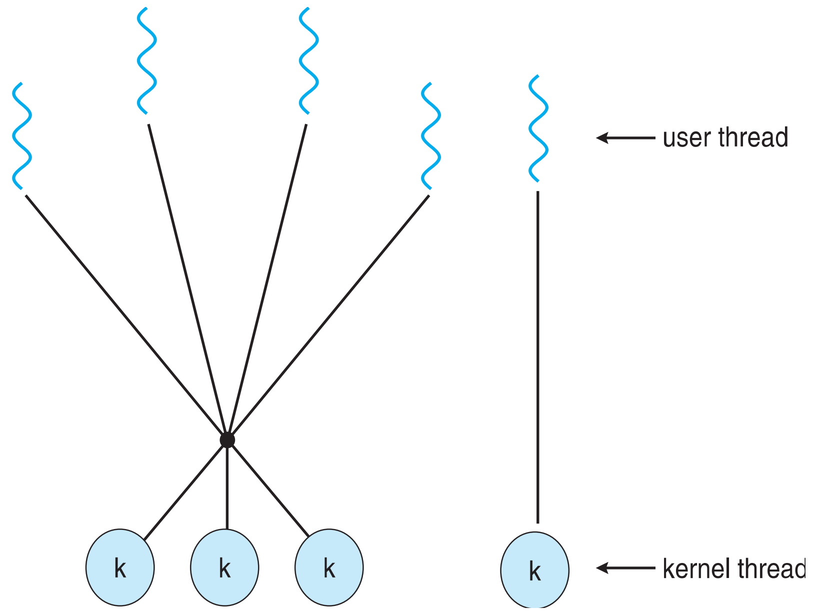
Many-to-Many Model

- 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 prior to version 9
- Windows with the *ThreadFiber* package



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - i.e.Tasks could be scheduled to run periodically