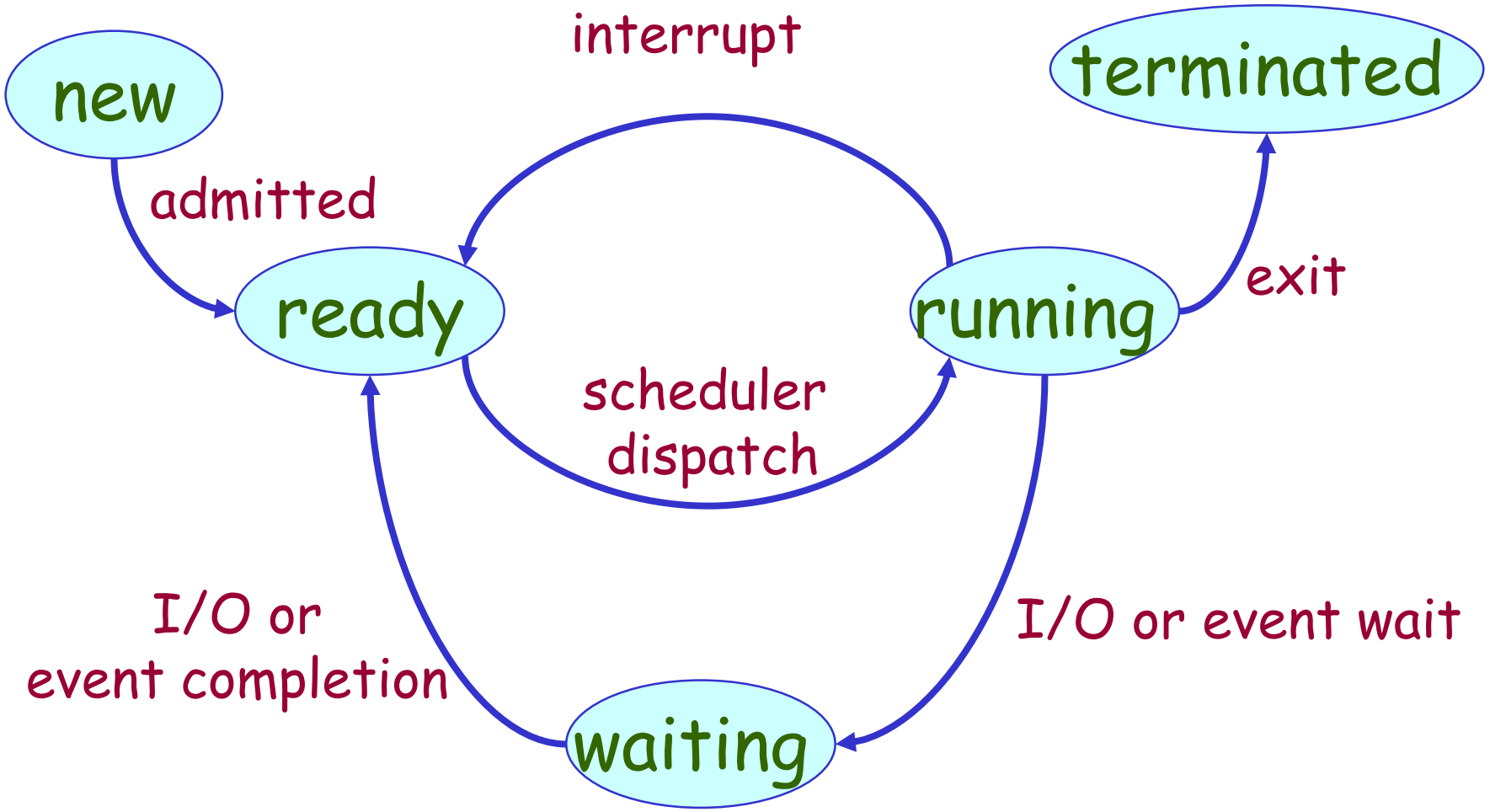
Review-Diagram of Process State



Threads

Example

- Imagine the following C program:

```
main() {
    Compute("deadloop.txt");
    PrintMyName("windows.txt");
}
```

- What is the behavior here?
  - Program would never execute **`PrintMyName()`**
  - Why?
    - **`Compute()`** would never finish

Use of Threads

- Version of program with Threads:

```
main() {
    CreateThread(Compute("deadloop.txt"));
    CreateThread(PrintMyName("windows.txt"));
}
```

- What does "**`CreateThread()`**" do?
  - Start an independent **thread** to run a given procedure

Outlines

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues

Overview

There is process already.
Why we still need thread?

Thread

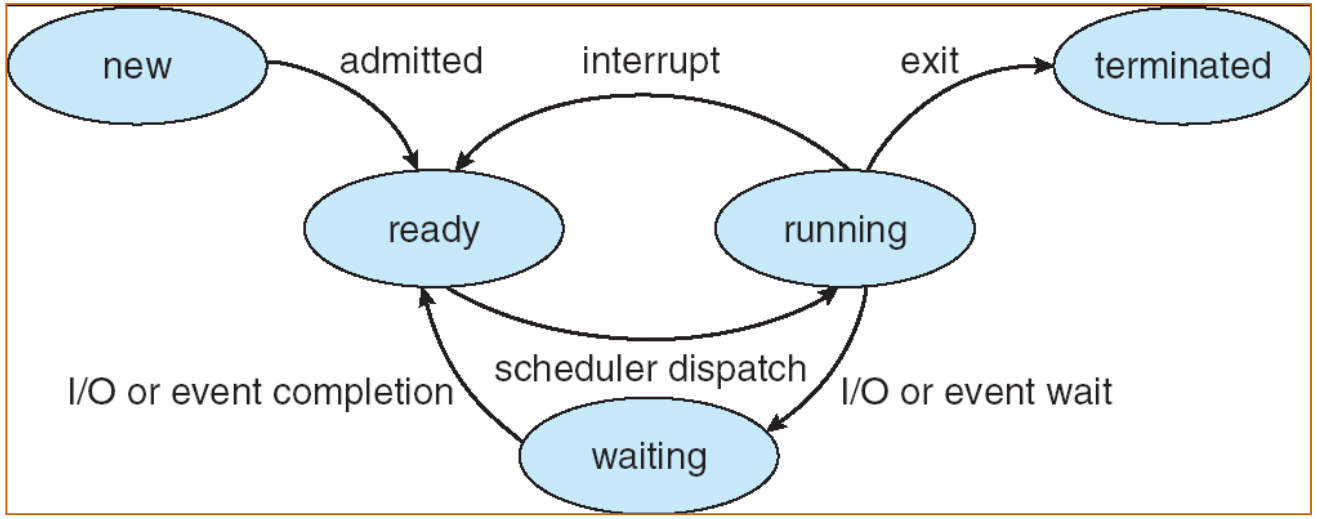
- Modern OS has extend the process concept to ***allow a process to have multiple threads of execution***

Thread

- **Thread:** a sequential execution stream within process
  - Process still contains a single address space
    - Each process starts with a main thread, but the new thread can be created dynamically inside the address of the process
  - No protection between threads
    - A thread can access all objects in the process
    - An object created by a thread that is visible to all threads in the same process.

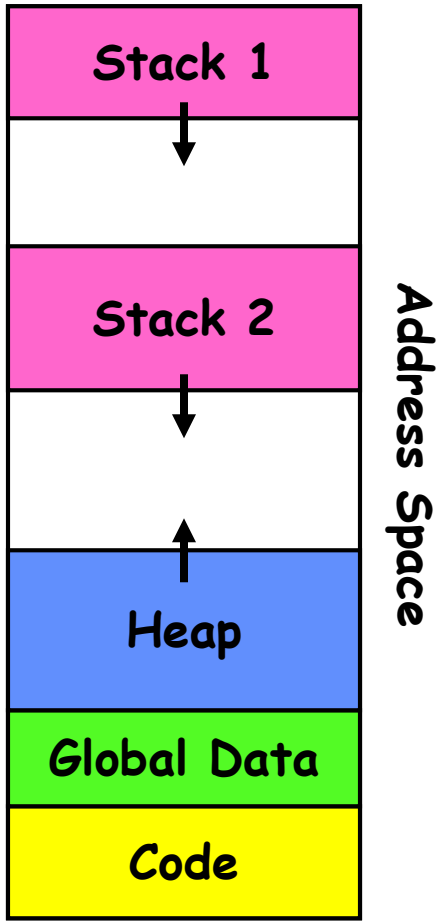
Lifecycle of a Thread

- As a thread executes, it changes the state:
  - **new**: the thread is being created
  - **ready**: the thread is waiting to run
  - **running**: instructions are being executed
  - **waiting**: the thread is waiting for some event to complete
  - **terminated**: the thread has finished execution

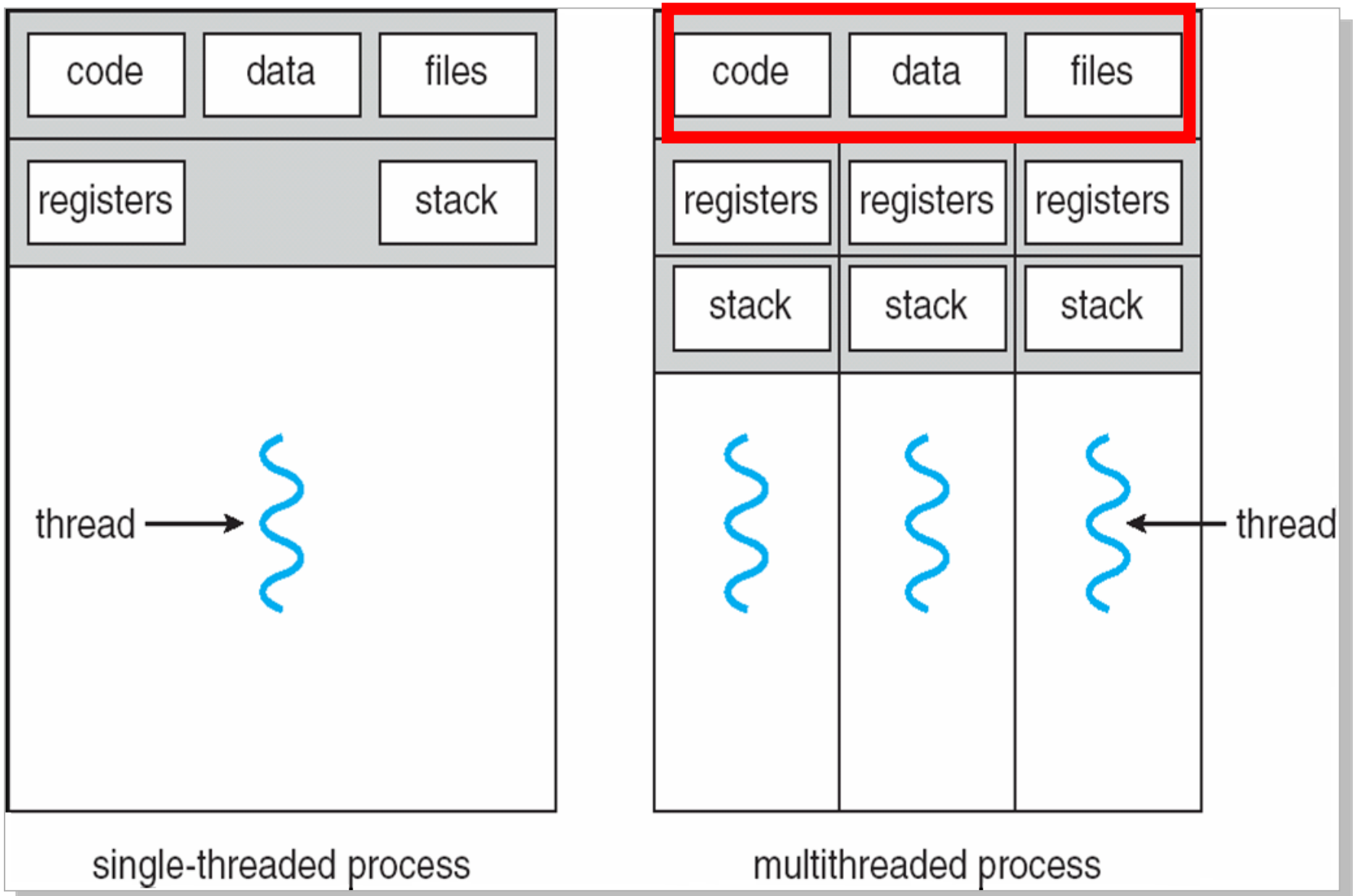


Memory Footprint of Two Threads

- If we stopped this program and examined it with a debugger, we would see
  - Two sets of CPU registers
  - Two sets of Stacks

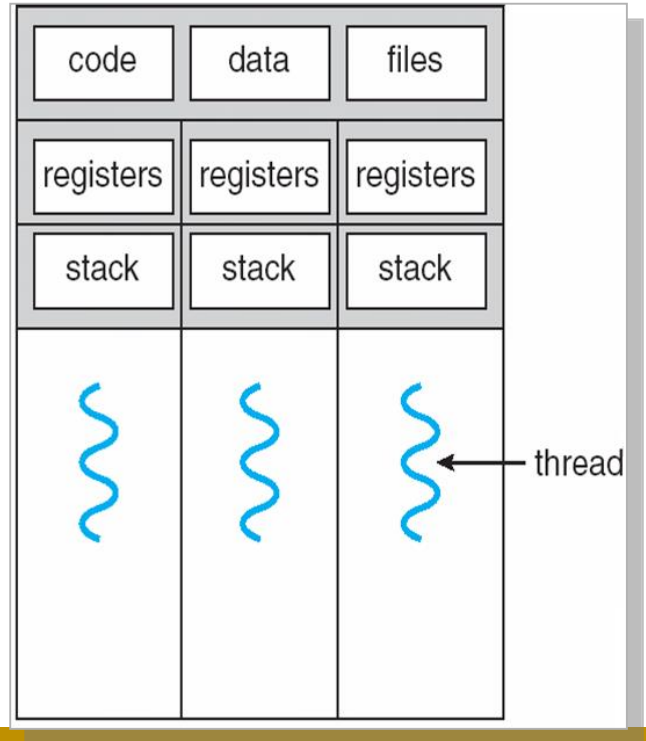


Single and Multithreaded Processes



Thread

- It comprises a ***thread ID***, a ***program counter***, a ***register set***, and a ***stack***.
- It shares something with other threads belonging to the same process
  - Code section
  - Data section
  - Other OS resources
    - such as open files and signals.



Use of Threads

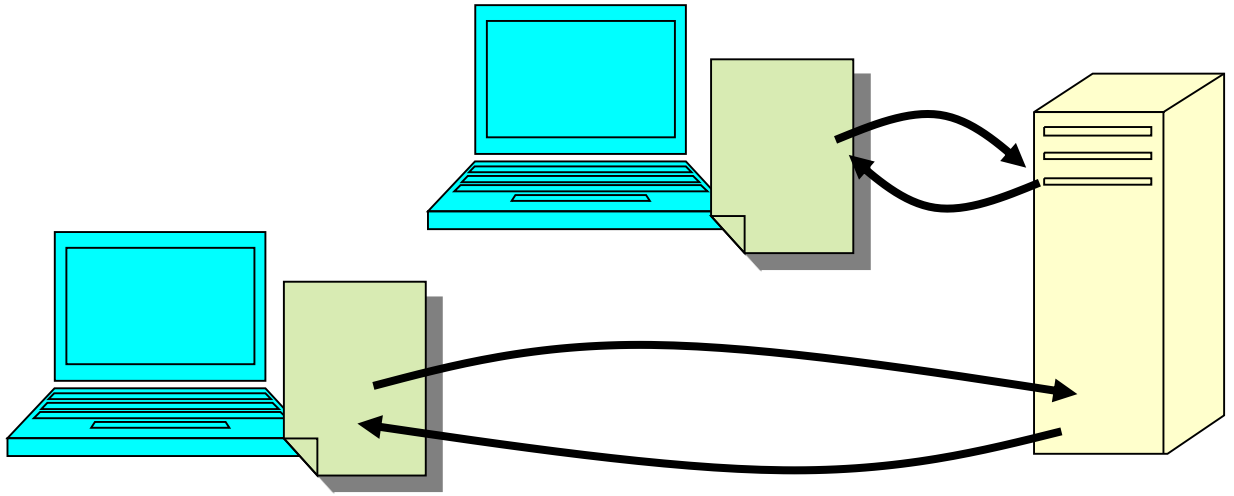
- Use "**CreateThread()**" to start an independent thread for running a given procedure
- Version of program with Threads:

```
main() {
    CreateThread(Compute("deadloop.txt"));
    CreateThread(PrintMyName("windows.txt"));
}
```

An Example: Busy Web Server

- Server must handle many requests from clients
- Non-cooperating version:

```
serverLoop() {
        connection = AcceptConnection();
        ProcessFork(ServiceWebPage(),connection);
}
```

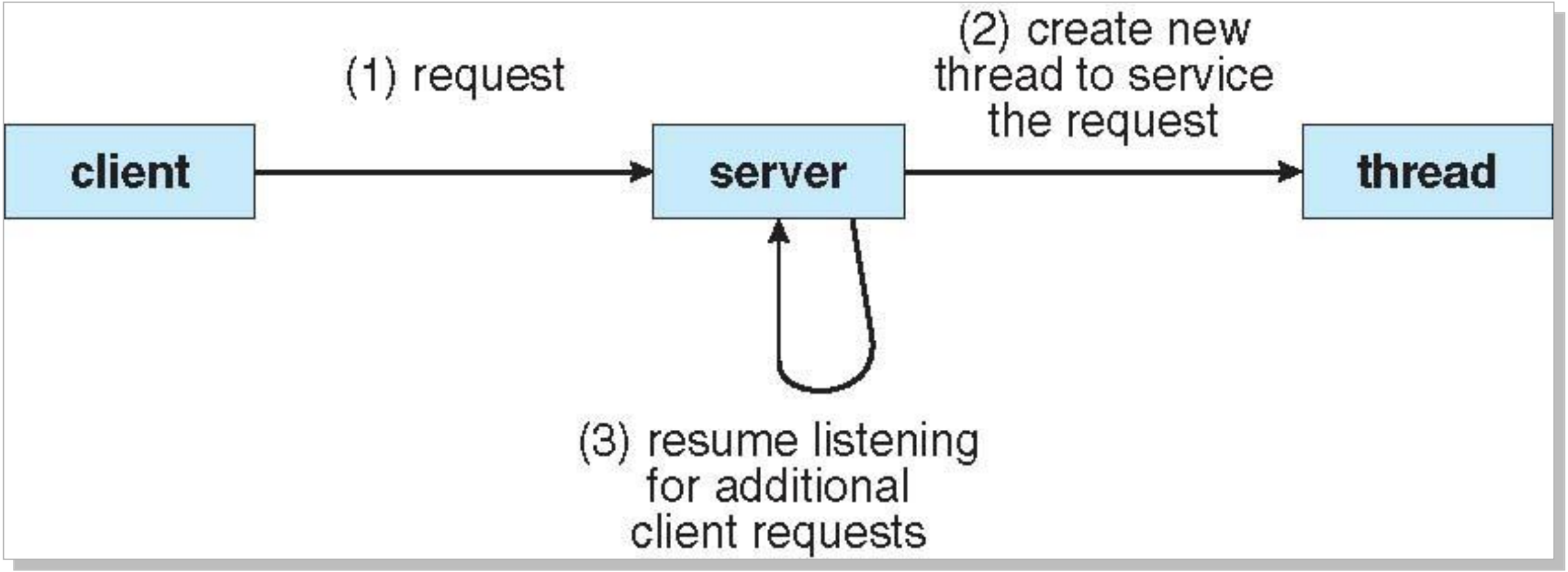


Multithreaded Server Architecture

- Now, use a single process
- The server creates **a separate thread** that would listen for clients requests, when a request was made, creates **a new thread** to service the request.
- Multithreaded (cooperating) version:

```
serverLoop() {
        connection = AcceptConnection();
        CreateThread(ServiceWebPage(),connection);
}
```

Multithreaded Server Architecture



Multithreaded Server Architecture

- Looks almost the same, but the multithreaded version has many advantages:
  - Can **share resources** in the process
    - File caches kept in memory
    - Results of CGI scripts, etc.
  - Threads are **much cheaper** to create than processes, so this has a lower per-request overhead

Benefits of Multithreaded Programming

- Responsiveness
- Resource Sharing
- Economy
- Utilization of Multiprocessor Architectures

Benefits of Multithreaded Programming

- **Responsiveness**: Allow a program to continue running even if part of it is blocked or is performing a lengthy operation.
- **Resource sharing**: several different threads of activity all within the **same** address space.

Benefits of Multithreaded Programming

- **Economy**: easy to create and destroy
  - In *Solaris*, creating a process is about 30 times slower than creating a thread, and context switching is about five times slower.
    - **Remark**: Using threads, context switches take less time.

Benefits of Multithreaded Programming

- **Utilization of multiprocessor architecture**: Several threads may be running in parallel on different processors.
  - Of course, multithreading a process may introduce concurrency control problem that requires the use of **critical sections** or **locks**.

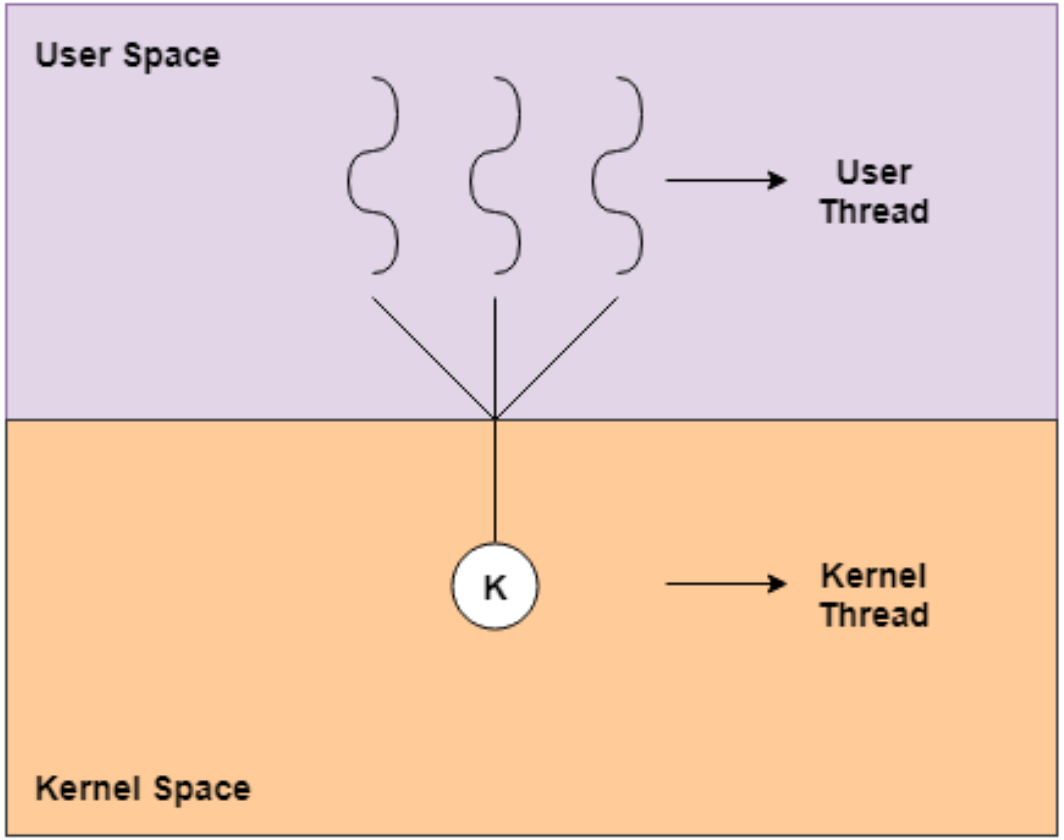
Thread vs Process

- Process has a complete resource platform, while threads have only essential resources, e.g. registers and stacks
- Threads reduce the time and space overhead of concurrent execution
  - Creation and termination time of thread is shorter than process
  - Thread switching time within the same process is shorter than the process
  - Communication between threads of the same process can be directly carried out without kernel

Multithreading Models

Types of Threads

- User-level threads
- Kernel-supported threads



Kernel Threads

- Supported by the Kernel
  - Native threads supported directly by the kernel
  - Every thread can run or block independently
  - One process may have several threads waiting on different things
- Drawback of kernel threads: a bit expensive
  - Need to make a crossing into kernel mode to schedule

User Threads

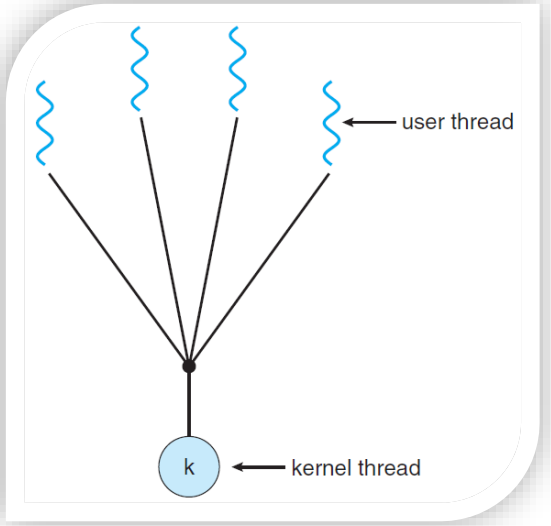
- Supported by a set of library calls at the user level
  - Thread management done by user-level threads library
    - User program provides scheduler and thread package
  - May have several user threads per kernel thread
- Advantages: cheap and fast
- Disadvantages: if the kernel is single threaded, system call from any thread can block the entire task
- Example thread libraries
  - POSIX Pthreads, Win32 threads, Java threads

Multithreading Models

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

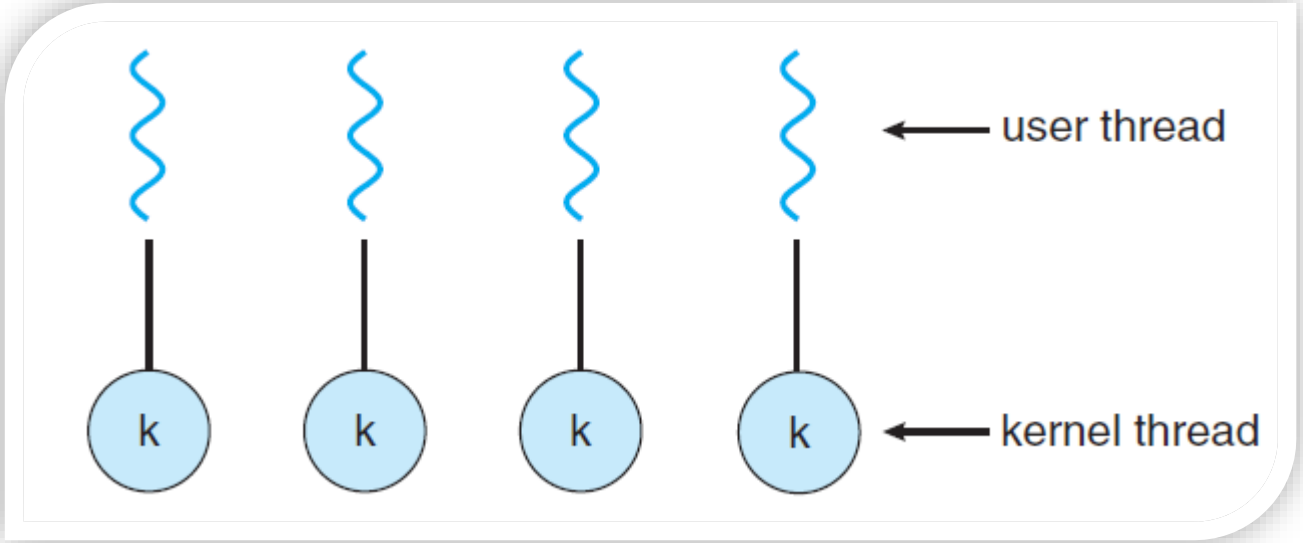
Many-to-One

- Many user-level threads mapped to a single kernel thread
  - Thread management is done by thread library in user space
  - The entire process will block if a thread makes a blocking system call



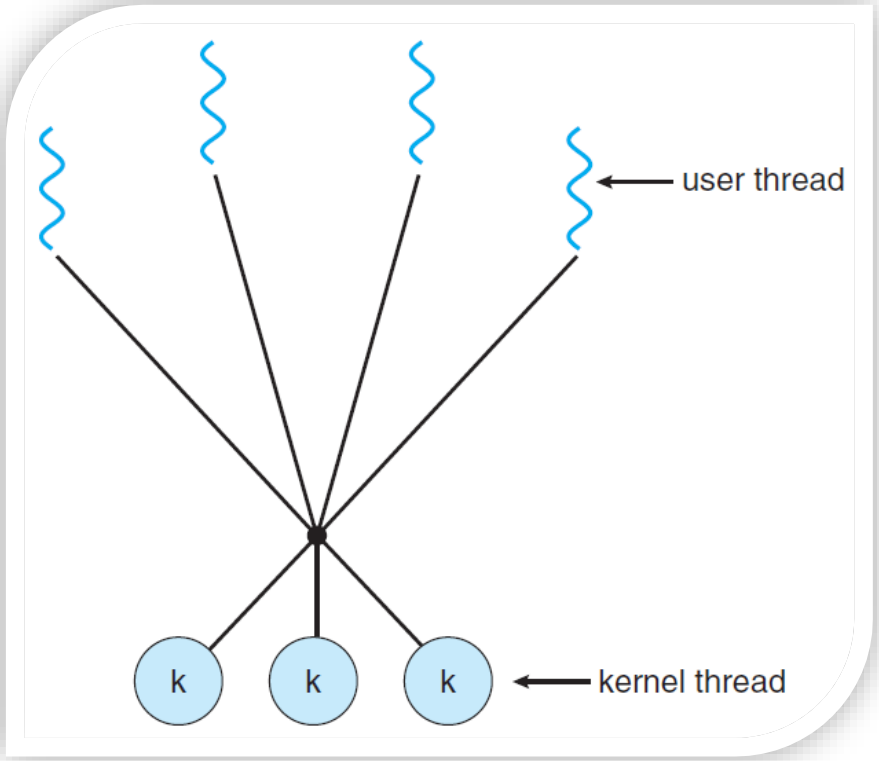
One-to-One

- Each user-level thread maps to a kernel thread
  - Allows multiple threads to run in parallel on multiprocessors
  - The overhead is high
- Examples
  - Windows NT/XP/2000
  - Linux



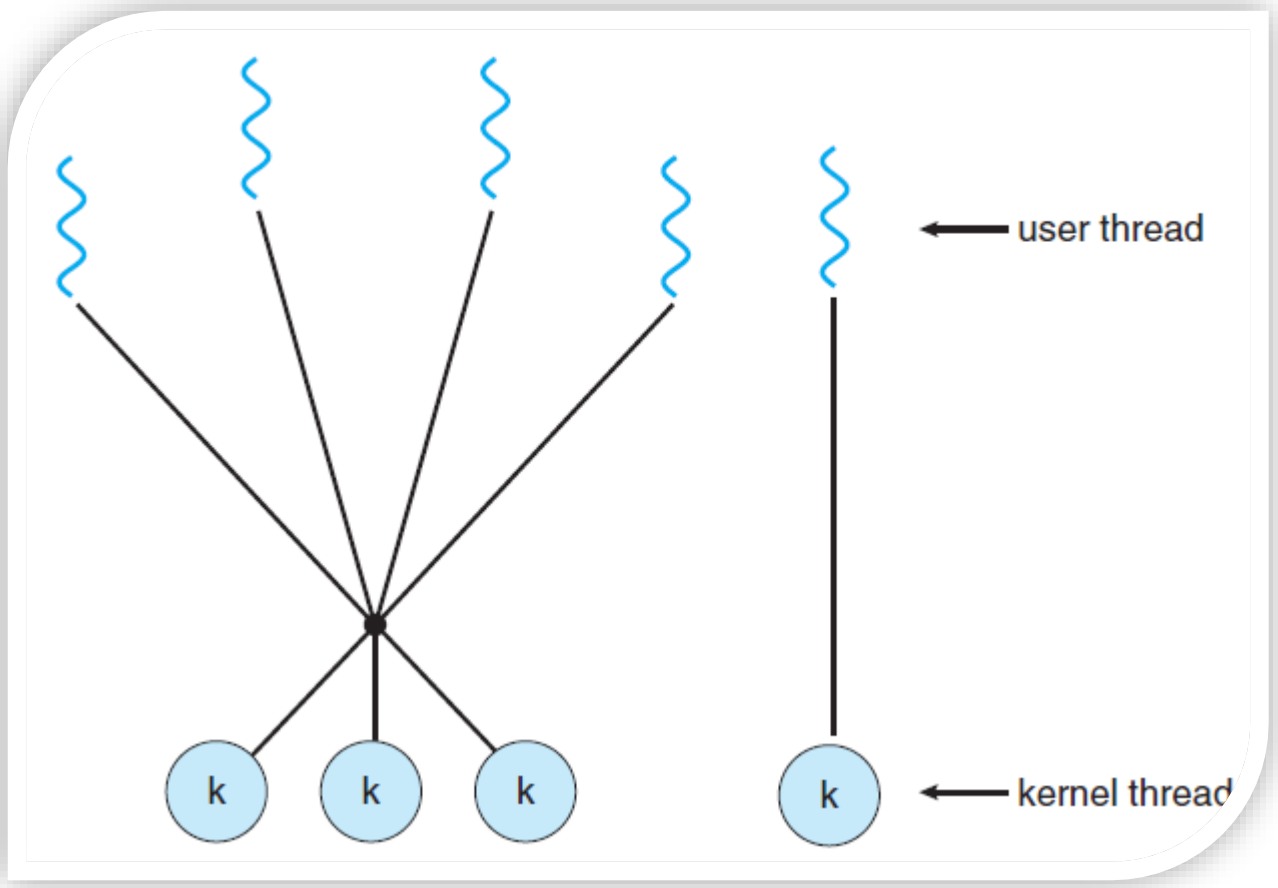
Many-to-Many

- Allows many user level threads to be mapped to many kernel threads
- Allows operating system to create a sufficient number of kernel threads
- Example
  - Solaris prior to version 9
  - Windows NT/2000 with Thread Fiber package



Two-level Model

- Support both Many-to-Many and One-to-One



Thread Libraries

Thread Libraries

- Thread library provides programmer with API for creating and managing threads
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems
  - Solaris, Linux, Mac OS X

Win32 Thread API

- Create a thread
  - CreateThread()
- Exit a thread
  - ExitThread()
- Sleep for sometime
  - Sleep()

Win32 Thread API

- CreateProcess()
  - OS create a process and a main thread.
- CreateThread()
  - Create a new thread based on the main thread.

Java Thread API

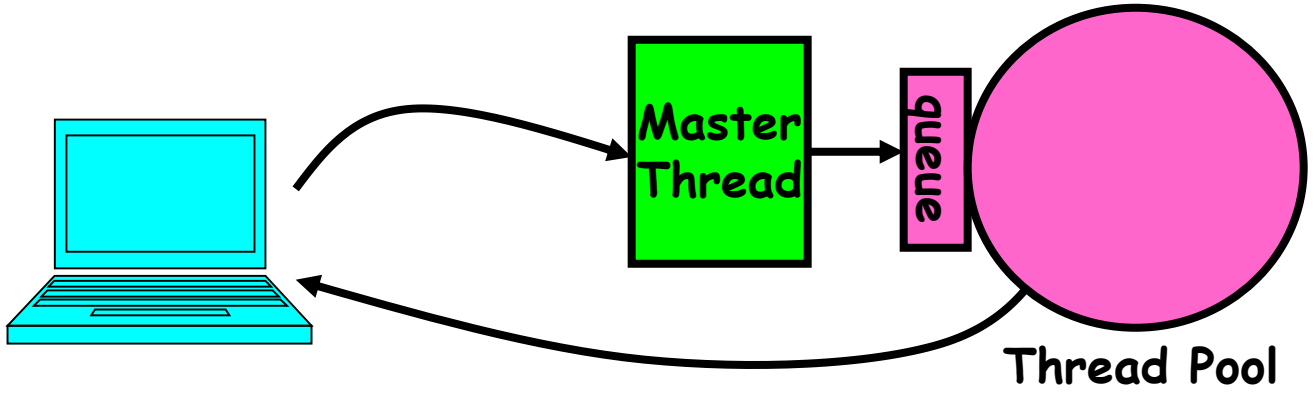
- Threads are the fundamental model of program execution in Java
- All Java programs comprise at least a single thread
- Creating thread in Java
  - Create a new class that is derived from **`Thread`**
  - Define a class that implements **`Runnable`** interface

Threading Issues

Threading Issues

- Problems of a multithreaded web server
  - The amount of time required to create the thread prior to serving the request
    - This thread will be discarded once it completed its work
  - No bound on the number of threads concurrently active in the system
    - Unlimited threads could exhaust system resources
- How to solve these problems?
  - Thread pool

Thread Pool




```
master() {
    allocThreads(worker,queue);
    while(TRUE) {
        con=AcceptCon();
        Enqueue(queue,con);
        wakeUp(queue);
    }
}
```

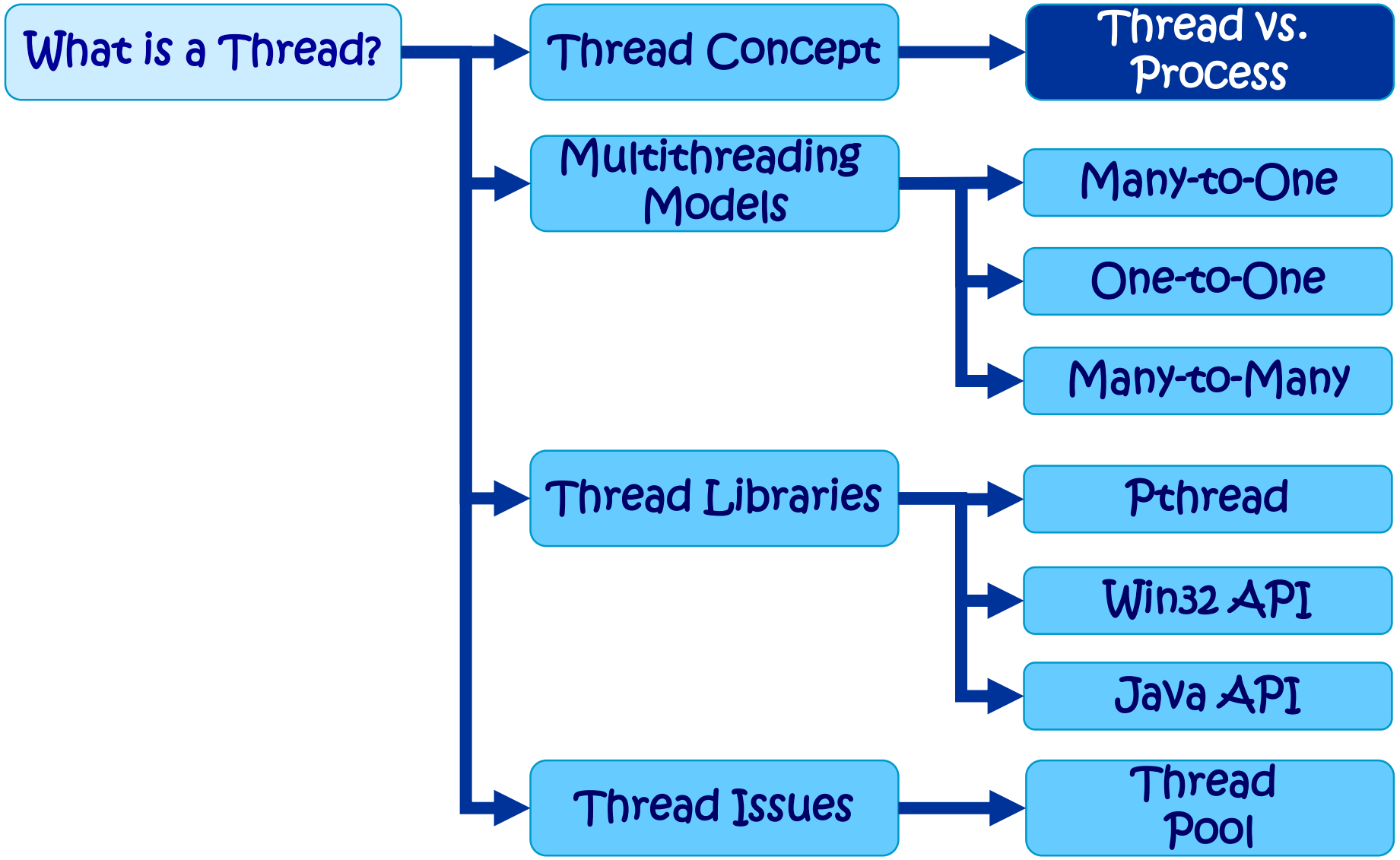
```
worker(queue) {
    while(TRUE) {
        con=Dequeue(queue);
        if (con==null)
            sleepOn(queue);
        else
            ServiceWebPage(con);
    }
}
```

Benefits of Thread Pool

- Servicing a request with an existing thread is **faster** than waiting to create a thread.
- A thread pool **limits** the number of threads that exist at any one point.
  - This is particularly important on systems that cannot support a large number of concurrent threads.

Summary

Summary


What is a Thread?
Thread Concept
Thread vs. Process
Multithreading Models
Many-to-One
One-to-One
Many-to-Many
Thread Libraries
Pthread
Win32 API
Java API
Thread Issues
Thread Pool

北京交通大学

Thank you !

Q & A