Concurrent and Distributed Systems

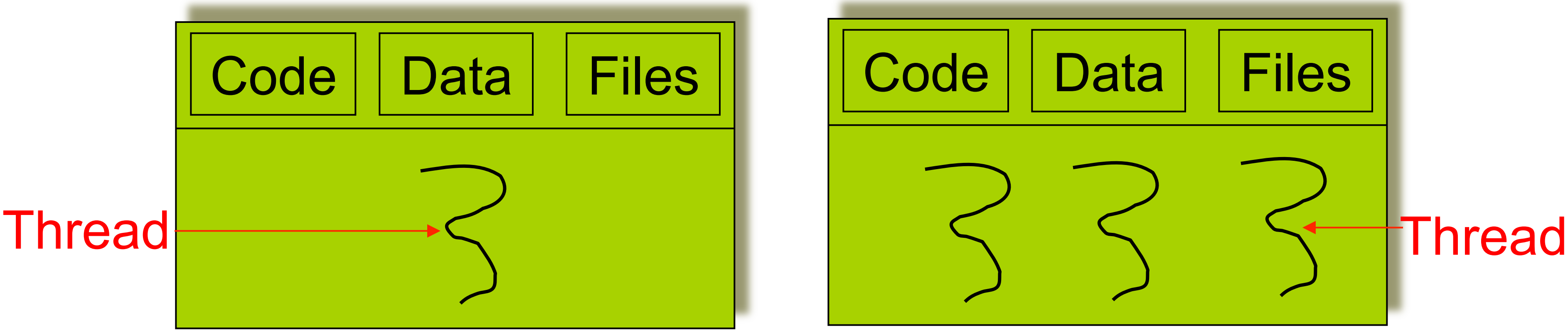
Threads

Introduction

- Threads
- Benefits
- Thread Implementation
- Multithreading Models
- Threads in Java

Threads (lightweight Process)

- So far implied that a process has ONE thread of execution
- Many OS have extended the process model to allow processes to have more than one thread of execution
- Process scheduling and context switching is heavyweight
- Thread scheduling and switching is light weight



Threads

- Basic unit of CPU utilisation
- Comprises
  - Thread ID
  - Program counter
  - Register set
  - Stack
- Shares (which differences with a process?)
  - Code section
  - Data section
  - Open files, signals

Problems with Processes

- Many software applications are implemented in a single process with multiple threads of execution
- Text processing
  - Display graphics
  - Get keystrokes from the user
  - Perform spell checking
- Web browsing
  - Display text/images
  - Retrieve data from network
- Web server
  - Single process – long wait for some requests
  - Create a process per request – <u>enormous overhead</u>
  - Create a thread per request

Benefits of Threads

- **Responsiveness** – an application continues running even if part of it is blocked or performing a lengthy operation
- **Resource sharing** – Threads share memory and resources of the process they belong to
- **Economy** – allocating memory and resources to processes is costly. Creating and switching between threads is more cost effective as the overhead is smaller
- **Utilisation of multiprocessor architectures** – each thread may run on a different processor. In a single processor environment, the context switching allows pseudo parallelism

User and Kernel Threads

- Threads may be provided by a user library
  - Posix threads, Mach C-threads
  - Library supports creating, scheduling and management
  - OS kernel is unaware of user threads
  - Threads are fast to create and manage
  - But, if a thread performs a blocking system call, e.g. reading from keyboard, => ?

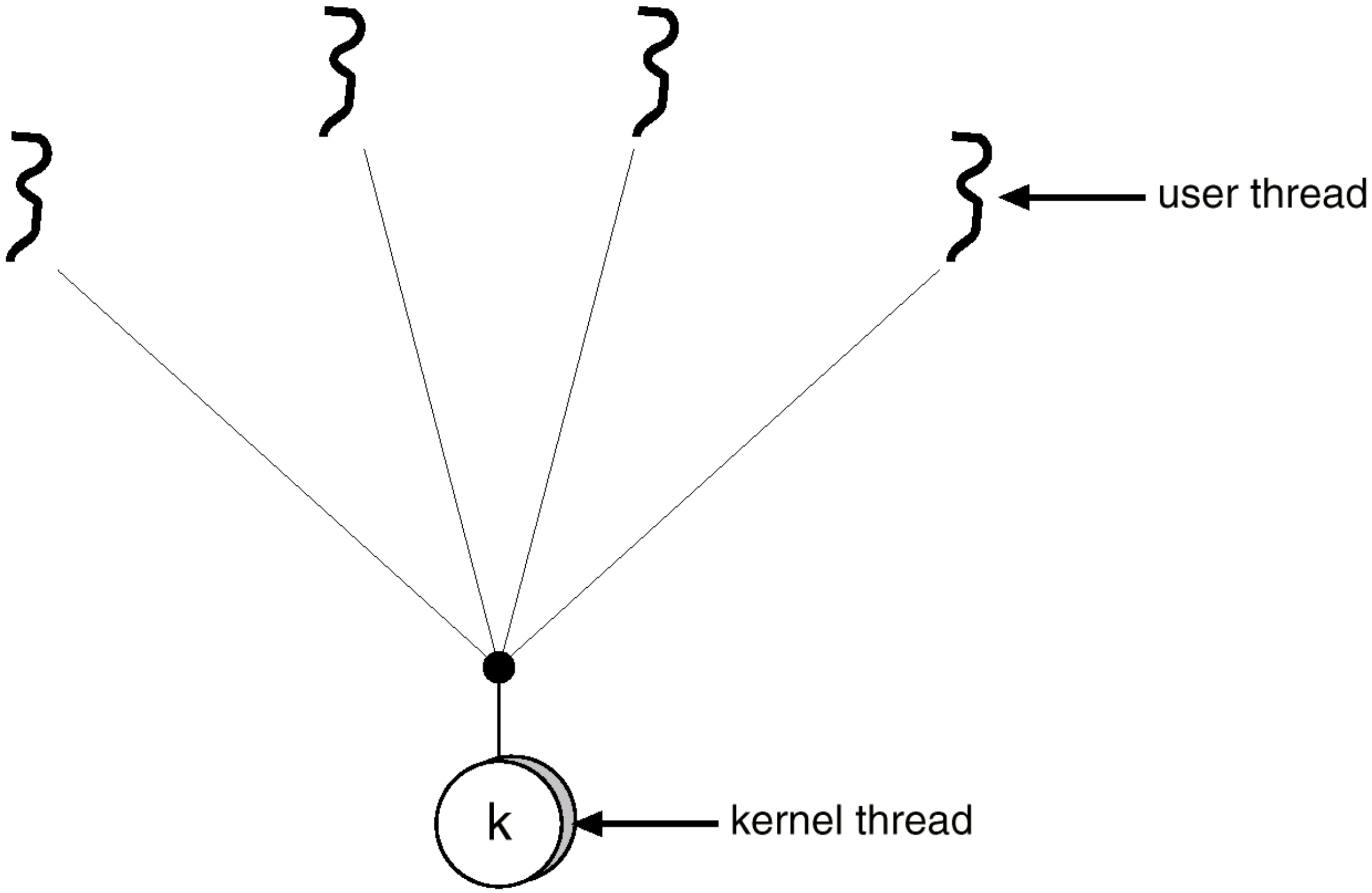
User and Kernel Threads

- Threads may be provided by a user library
  - Posix threads, Mach C-threads
  - Library supports creating, scheduling and management
  - OS kernel is unaware of user threads
  - Threads are fast to create and manage
  - But, if a thread performs a blocking system call, e.g. reading from keyboard, => all threads are blocked
- Threads may be provided by the OS kernel
  - Windows NT, Solaris, Digital UNIX
  - Generally slower to create and manage than User threads
  - Concurrent threads may proceed during a blocking system call
  - Kernel can schedule threads to run on different processors on a Multiprocessor

Multithreading Models

- Different Multithreading Models
  - Many-to-One Model
    - Many User-Level Threads Mapped to Single Kernel Thread.
    - Used on Systems That Do Not Support Kernel Threads.

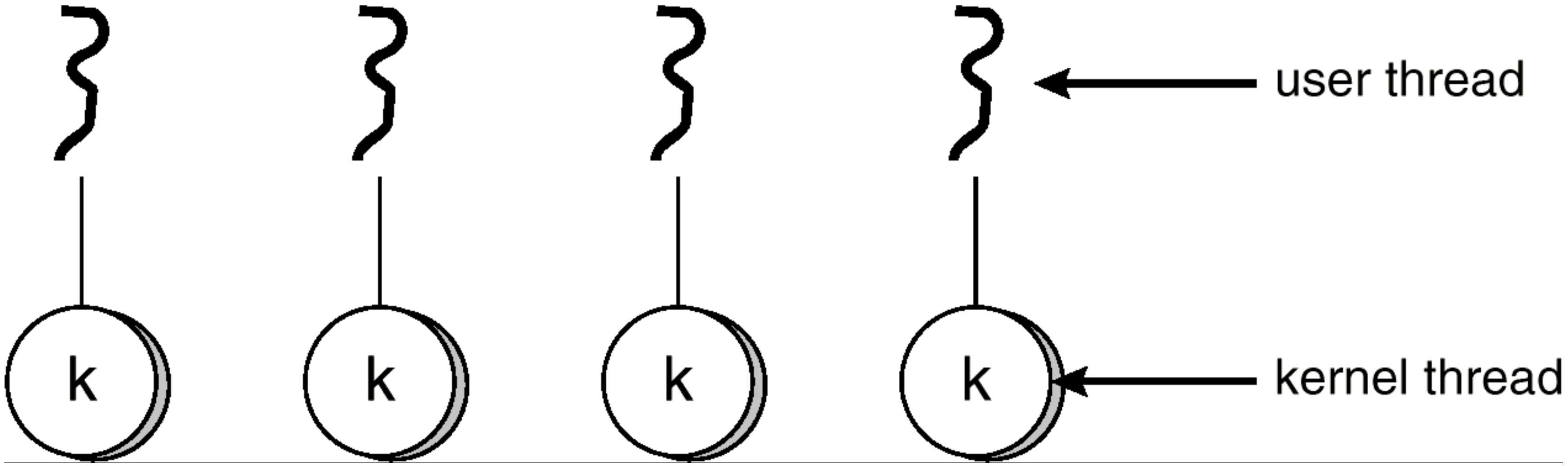
Many-to-One Model



Multithreading Models

- Different Multithreading Models
  - Many-to-One Model
    - Many User-Level Threads Mapped to Single Kernel Thread.
    - Used on Systems That Do Not Support Kernel Threads.
  - One-to-One Model
    - Each User-Level Thread Maps to Kernel Thread.
    - Creating a user thread requires creating a kernel thread
    - Examples: Windows NT, OS/2

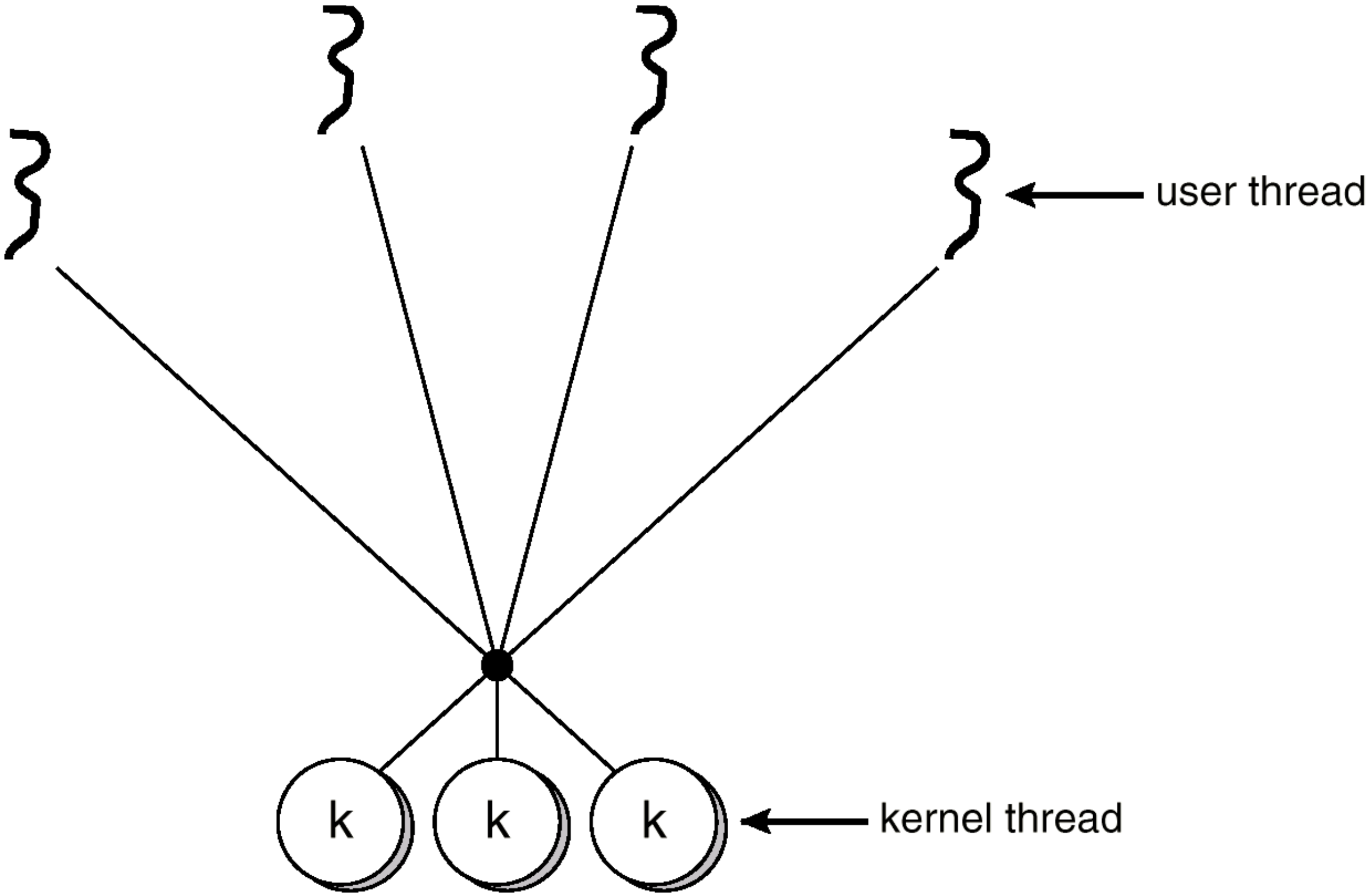
One-to-One Model



Multithreading Models

- Different Multithreading Models
  - Many-to-One Model
    - Many User-Level Threads Mapped to Single Kernel Thread.
    - Used on Systems That Do Not Support Kernel Threads.
  - One-to-One Model
    - Each User-Level Thread Maps to Kernel Thread.
    - Creating a user thread requires creating a kernel thread
    - Examples: Windows NT, OS/2
  - Many-to-Many Model
    - Multiplexes many user-level threads to fewer or equal kernel threads
    - Examples: Solaris, IRIX, Digital Unix

Many-to-Many Model



Java Threads: How To

- Providing threads at language level (… bit more complex!)
- All Java programs run at least one thread in the JVM
- Processes in JAVA**?** (a bit different from C)

1. Extending `Thread` class
   - Not possible if a class already inherits another class (no multiple inheritance)
2. Implementing the `Runnable` interface (Java interface**?**)
   - e.g. `Applet` already extends `Panel` class
   - A multithreaded `Applet` is created extending the `Applet` class and implementing the `Runnable` interface

A Java Thread

```
class  Worker_1 extends Thread
{
    public void run() {
            System.out.println(“I am a Worker Thread”);
    }
}
```

1. Extend the `Thread` class
2. Overwrite the `run()` method

Initialising a Thread

```
public class First{
    public static void main(String args[]) {
        Worker runner = new Worker1();
        runner.start();
        System.out.println(“I am the main thread”);
    }
}
```

- A thread is created by calling `start()`
  - memory is allocated
  - A new thread within the JVM is initialised
  - `run()` of the object is called
- Do not call `run()` yourself!
- Two threads are created: the application thread and the `runner` thread

The Runnable Interface

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

A thread can also be created by implementing the `Runnable` interface

1. Define the `run()` method
2. `Thread` class also implements `Runnable`, thus `run()` needs to be defined

Initialising the Runnable Thread

```
class  Worker_2 implements Runnable
{
    public void run() {
            System.out.println(“I am a Worker Thread”);
    }
}
```

- Similar to extending the Thread class
- Initialising the new thread is slightly different to extending Thread
- No access to static or instance methods (such as start()) of Thread!
- However, start() is needed!

Creating a Thread

```
public class Second
{
    public static void main(String args[]) {
        Runnable runner = new Worker2();
        Thread thrd = new Thread(runner);
        thrd.start();
        System.out.println(“I am the main thread”);
    }
}
```

- A new Thread object is created and the Runnable object is passed as parameter to its constructor
- Thread is created calling start()
- Execution begins in the run() method of the Runnable object

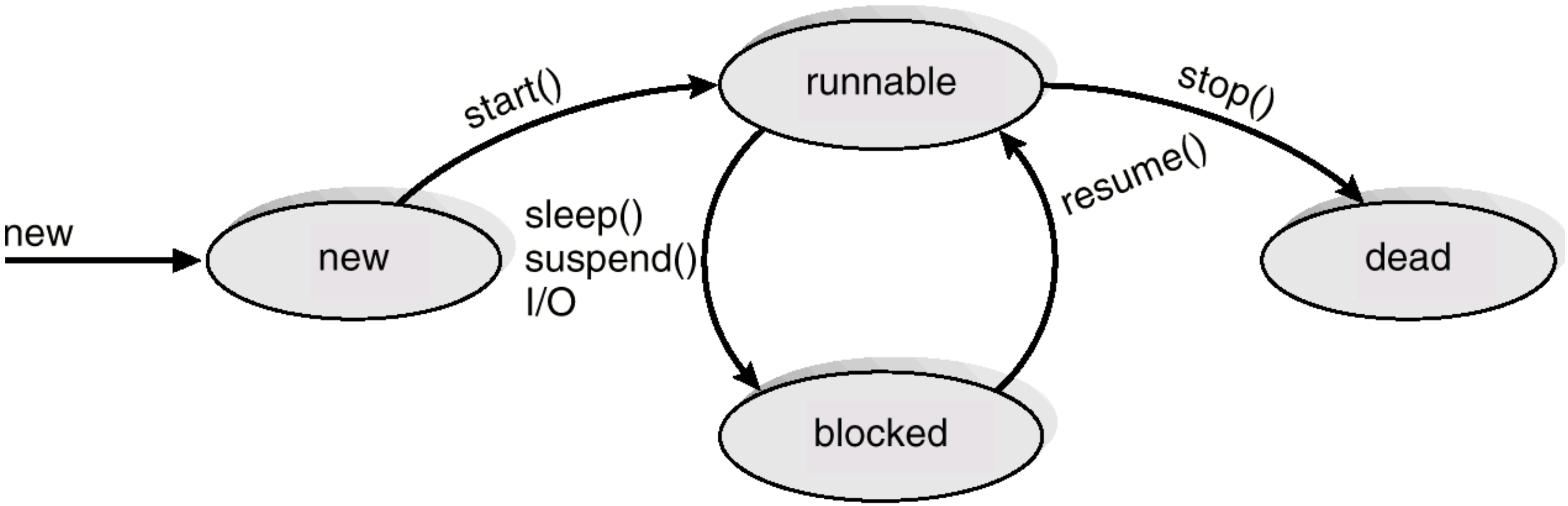
Another Example …

```
public class OurApplet extends Applet implements Runnable {

    public void init() {
      Thread th = new Thread(this);
       th.start();
   }

   public void run() {
          System.out.println(“I am a Worker Thread”);
    }
}
```

Managing Java Threads

- **suspend()** – suspends execution of the currently running thread.
  - Applet (running as separate thread) displaying some graphics is not visible → suspend the thread
- **sleep()** – puts the currently running thread to sleep for a specified amount of time.
- **resume()** – resumes execution of a suspended thread.
  - Applet is visible again → resume the thread and processing
- **stop()** – stops execution of a thread.
  - Thread cannot be resumed.

Java Thread States



Producer-Consumer Problem

```
public class Server {
     public Server() {
              MessageQueue mailBox = new MessageQueue();

              Producer producerThread = new Producer(mailBox);
              Consumer consumerThread = new Consumer(mailBox);

              producerThread.start();
              consumerThread.start();
   }
   public static void main(String args[])   {
              Server server = new Server();
   }
}
```

The Producer

```
class Producer extends Thread {
    public Producer(MessageQueue m) {
          mbox = m;
    }

  public void run() {
     while (true) {
           // produce an item & enter it into the buffer
    Date message = new Date();
          mbox.send(message);
     }
  }
  private  MessageQueue mbox;
}
```

The Consumer

```
class Consumer extends Thread {
    public Consumer(MessageQueue m) {
          mbox = m;
    }

  public void run() {
     while (true) {
    Date message = (Date)mbox.receive();
          if (message != null)
                    // consume the message
    }
  }
  private  MessageQueue mbox;
}
```

Summary

- Threads are ‘lightweight processes’
- Allow for more efficient use of resources
- User and Kernel Threads – Mappings

- Threads in Java language
  - Extend the Thread class
  - Implement the Runnable interface
- Threads can change states like processes