UNIVERSITY of STIRLING

www.cs.stir.ac.uk

Concurrent and Distributed Systems

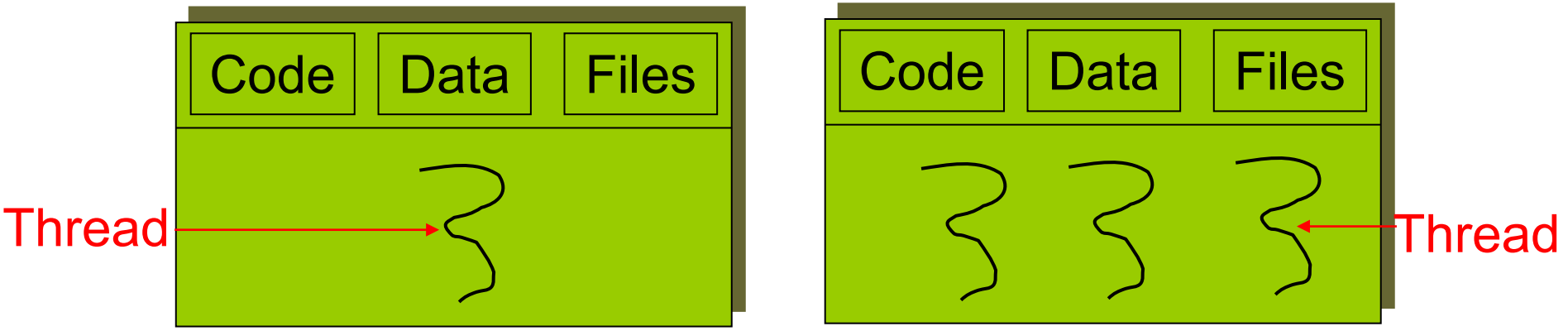
Threads

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
  - Code section
  - Data section
  - Open files, signals
- Many software applications are implemented in a single process with multiple threads of execution

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

Java Threads

- Unique – providing threads at language level
- All Java programs contain at least one thread in the JVM
- Creating additional threads
  - Extending `Thread` class
    - Not possible if a class already inherits another class (no multiple inheritance)
    - Applet already extends Applet class
  - Implementing the Runnable interface
    - A multithreaded Applet is created extending the Applet class and implementing the Runnable interface

A Java Thread

```
class  Worker1 extends Thread
{
    public void run() {
        System.out.println("I am a Worker Thread");
    }
}
```

- Extend the Thread class
- 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
- Define the run() method
- Thread class also implements Runnable, thus run() needs to be defined

Initialising the Runnable Thread

```
class  Worker2 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 …

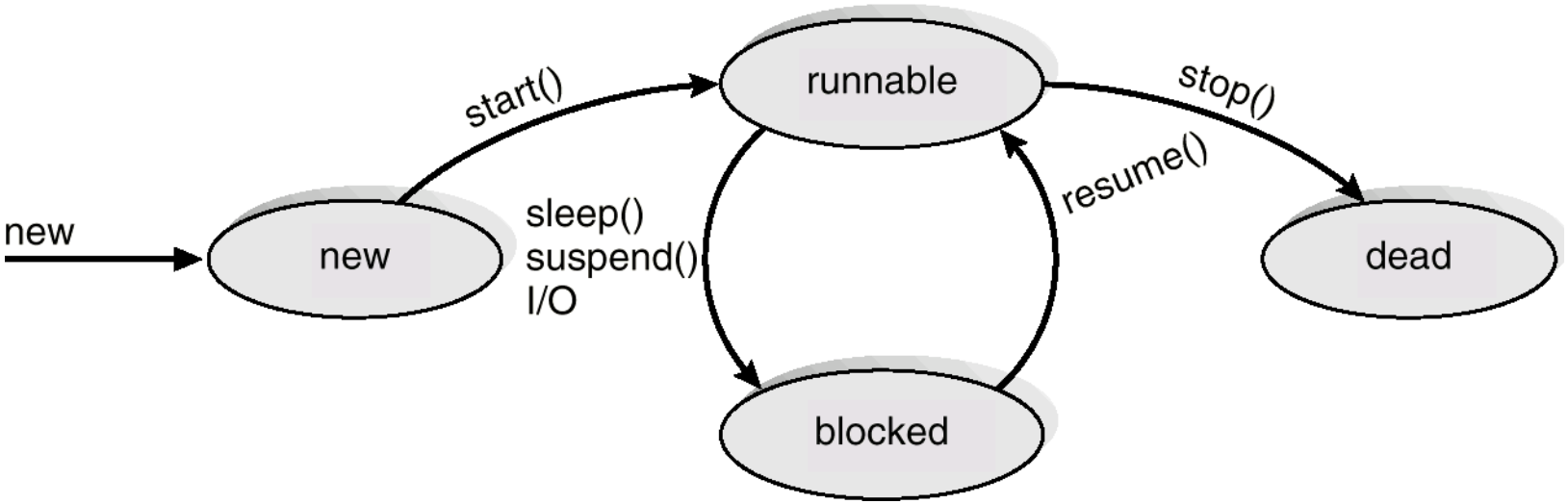
```
public class OurApplet extends Applet implements Runnable {
     public void init() {
        Thread th = new Thread(this);
         th.start();
    }

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

Managing Java Threads

- **sleep()** – puts the currently running thread to sleep for a specified amount of time.

- The following calls should **not** be called directly:
- **suspend()** – suspends execution of the currently running thread.
  - Applet (running as separate thread) displaying some graphics is not visible → suspend the thread
- **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;
}
```