# Operating Systems & Concurrency
# Week 3 - Threads


**Dr. Thiago Braga Rodrigues**

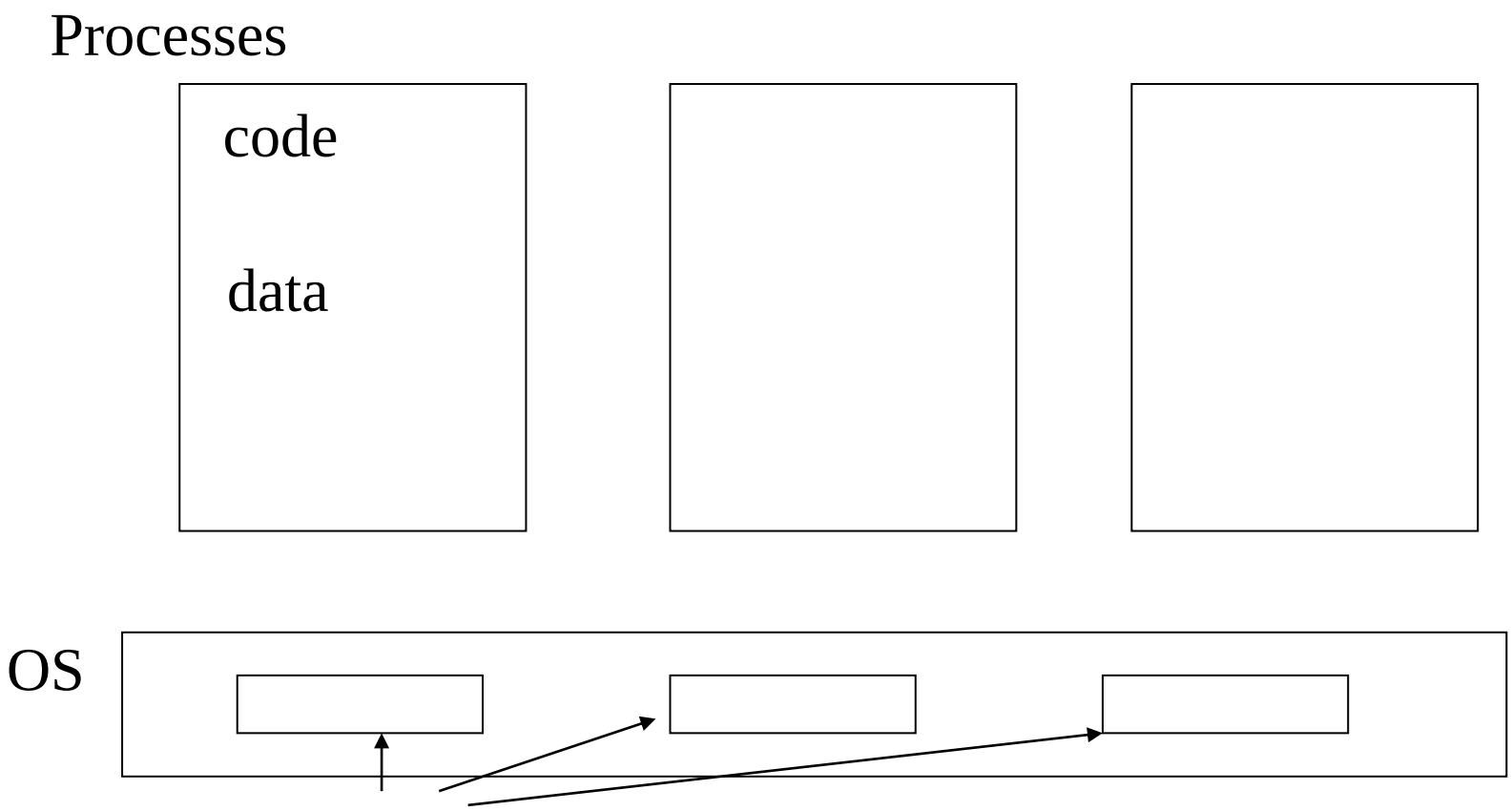
**tbrodrigues@ait.ie**

**Athlone Institute of Technology**
**2020**

# Processes

- In most operating systems, each process has its own private address space and a single thread of control.
- Processes can be switched on and off the CPU by the operating system.
- The OS  saves (and restores) the volatile environment (registers), memory management information (location of page table directory) and information about open files for processes.

# Multiple Processes



# Multiple Processes

- Machine code instructions in multiple processes can be arbitrarily interleaved by the runtime environment (the operating system).
- Normally processes don't share memory.

# Multiple Threads

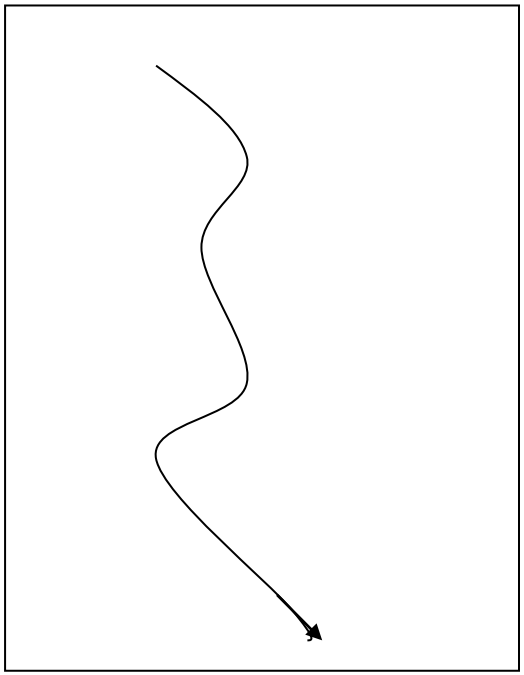
- A thread is like a mini-process inside a single (Java) program.
- Machine code instructions in multiple threads can be arbitrarily interleaved by the runtime environment (the JVM or the OS for C++ programs).
- Threads normally do share memory.
  - This has enormous repercussions. Examples?
- How many threads can run at a time in a PC?
  - Hardware and software support!

# Multiple Threads

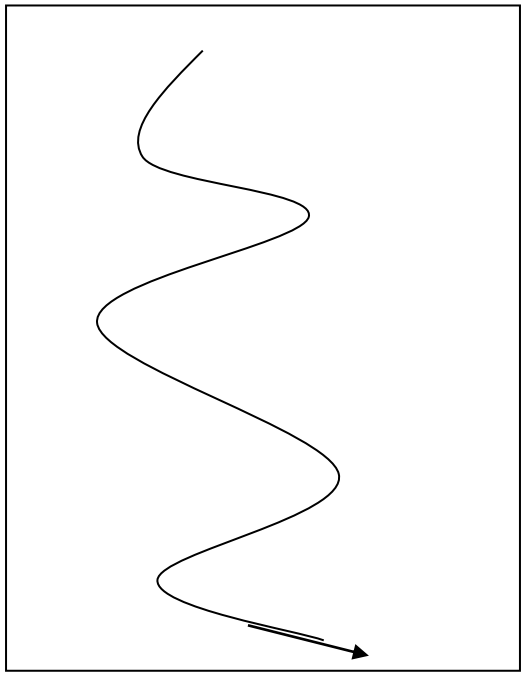
- Threads now have a thread state, i.e.
  - ready, running, blocked
- When one thread inside a process is blocked, another thread can be running.
- [Two threads or processes running at the same time are said to be running concurrently.]

# Multiple processes

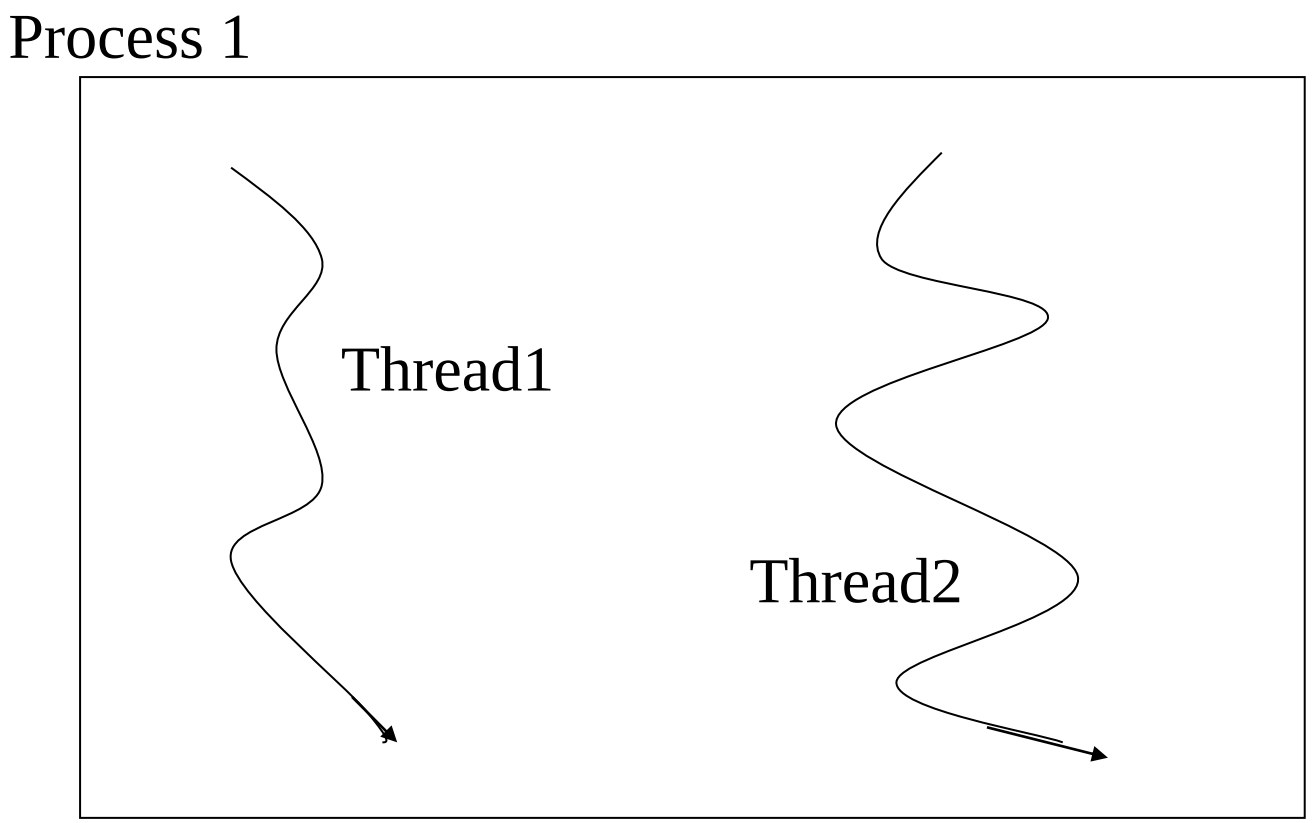
Process 1



Process 2



# Multiple Threads

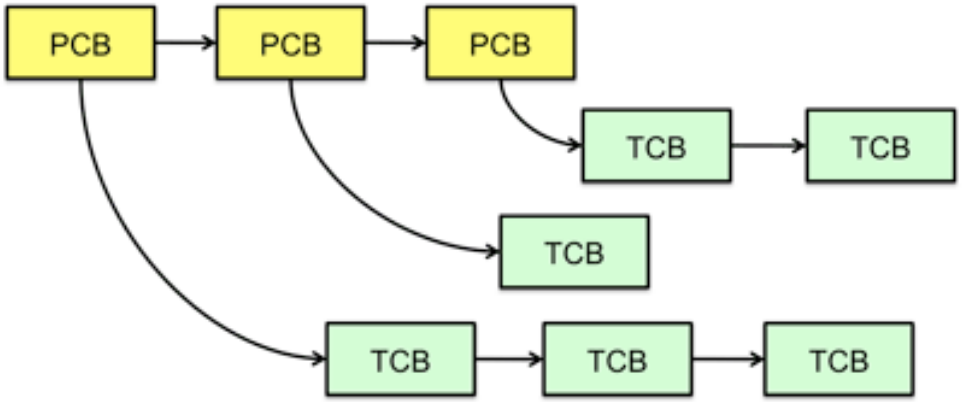


# Multiple Threads (cont)

- A thread is a single sequential flow of control through a program.
- One thread can execute for a while.
- Then there is a thread switch (mini-context switch), and its volatile environment is saved.
- Another thread executes.
- Operations in multiple threads can be arbitrarily interleaved by the runtime environment (JVM).

# Multiple Threads (cont)

- A process may be multithreaded where same program contain multiple concurrent threads.
- All of the process threads share same memory and open files
  - Each thread gets its own stack, instruction pointer, and registers.
- An OS keep track of threads and stored its per thread information in a data structure called thread control block (TCB).



# Example – Multi-threaded File Server

- Two HTTP requests for file are handled simultaneously by two threads.
- First Thread
  - reads request from client
  - issues I/O request for requested file
  - goes to sleep (each Thread has its own Thread state)
- Second Thread can now run
  - reads request from client
  - issues I/O request for requested file
  - goes to sleep
- So multiple requests can be handled at the same time.
- Each thread has its own state. One can be blocked while the other is running

# Advantages of threads

# Advantages of using threads

- Threads are more efficient
  - The OS does not need to create a new memory map for a new thread (like a process).
  - Does not need to keep track of the state of open files.
- Makes certain programming easy
  - Shared memory makes it trivial to share data among threads
  - Global and static variables can be read and written among all threads in a process
- A multithreaded application can scale performance as the number of processors or cores increases in a system
  - More threads

# Threads in Java

# Threads in Java

- Java supports the existence of multiple threads in the one program.
- There are two ways to provide a body for a thread.
- The first is to subclass the Thread class and override the Thread class's run() method.
- [The second is to override the Runnable interface.]

# Example 1

```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}
```

# Starting Two SimpleThreads

```
class TwoThreads1 {
    public static void main (String[] args) {
        Thread t1 = new SimpleThread1("Jamaica") ;
        Thread t2 = new SimpleThread1("Fiji") ;
        t1.start();
        t2.start();
    }
}
```

# SimpleThread1

- The class SimpleThread1 contains
  - a constructor for the class which calls the constructor from the super class.
  - a run() method.

```
public Thread(String name)
// creates a new thread with a specified name
public void run()
// subclasses of Thread should override this method.
```

# Multiple Threads

- The above example gives two independent threads running at the same time.
- It is completely up to the Java interpreter (JVM) when each thread runs.
- The JVM can time-slice the two threads, like the OS does with processes.
- Up the size of the loop and see what happens.

# Multiple Threads

- You can not and should not make any assumptions about when each thread will run.

# Thread Methods

- start() - makes the thread runnable.
- run() - called by JVM when it is the turn of the thread to run.

# Example 2

```
public void run() {
    for (int i = 0; i < 10; i++) {
        System.out.println(i + " " + getName());
        try {
            sleep(1000);
        } catch (InterruptedException e) {}
    }
    System.out.println("DONE! " + getName());
}
```

# sleep(), yield()

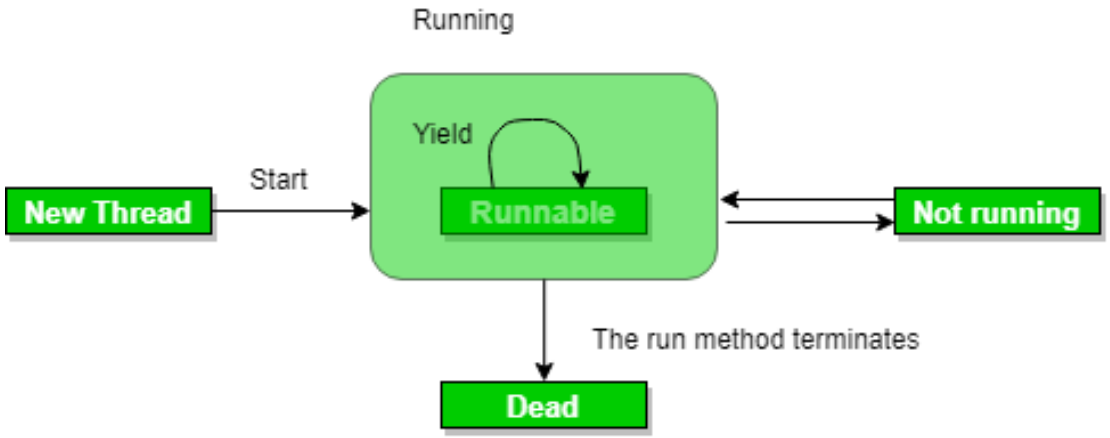
- sleep() – causes the currently executing thread to pause for a specified number of milliseconds.
- When one thread sleeps for a second, the other thread is scheduled.
- yield() – causes the thread to temporarily pause and allow other threads to run.

# Thread Methods

- void run()
  - body of the thread implemented by programmer
- void start()
  - make the thread runnable (called by programmer)
- void sleep(int milliseconds)
  - block the thread for a number of milliseconds

# Thread Methods

- void yield()
  - temporarily stop and let any other runnable thread run
- join()
  - t.join() - stop the current (parent) thread until thread t finishes



# Creating Threads - Implementing the Runnable Interface

# The Runnable Interface

- The second way to create a Thread in Java is to implement the Runnable Interface.
  - An interface that is to be implemented by a class whose instances are intended to be executed by a thread
- Write a class that implements the interface Runnable.
- It should implement the method public void run().

# The Runnable Interface

- To start a Thread, construct a Thread object using the constructor
  - Thread(Runnable target, String name)
- Then invoke start() on this Thread.

# Example (cont)

```
class TwoThreadsRunnable {
  public static void main (String args[]) {
    Runnable s1 = new SimpleThreadRunnable("Jamaica") ;
    Thread t1 = new Thread(s1) ;
    Runnable s2 = new SimpleThreadRunnable("Fiji") ;
    Thread t2 = new Thread(s2) ;
    t1.start();
    t2.start();
  }
}
```

# Example

```
class SimpleThreadRunnable implements Runnable {
  String name ;

  public SimpleThreadRunnable(String name) {
    this.name = name ;
  }

  // implement run() as before.
```

# Threads - Summary

# Summary

- It is possible to write a program that consists of a number of threads.
- Threads are like mini-processes.
- The main difference between process and a thread is that threads normally share data (variables) while processes normally don't.
- When a number of Java threads execute, the JVM can decide when and for how long to run each thread.

# Summary (cont)

- How instructions from different threads are interleaved can not be determined and will normally vary from one execution of the program to the next.
- To create a thread in Java write a class that
  - extends the class Thread
  - implements the method public void run()

# Summary (cont)

- Some thread methods
  - start() – make the thread runnable
  - run() – implement this method
  - sleep(long time) – block the thread
  - yield() – let another thread run
  - join() – wait for a thread to finish

# Summary (cont)

- A second way to create a Thread in Java is to
  - Write a class that implements the interface Runnable
  - [implement the run() method]
  - Construct a thread object and pass a Runnable object to the Thread constructor.

# Lab