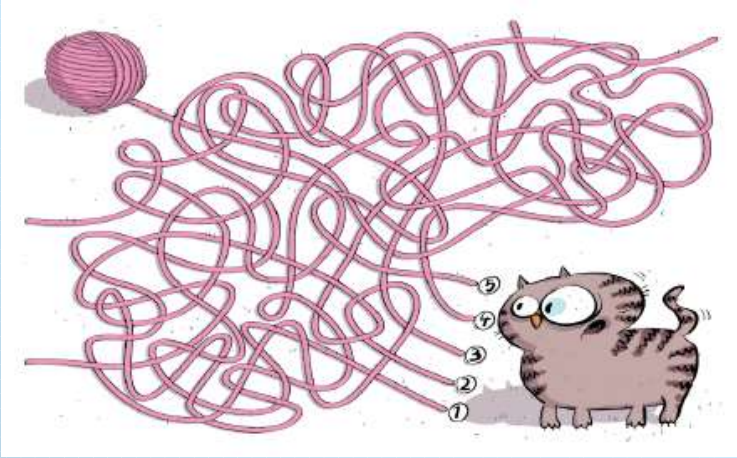
Threads
Part 1



Objectives

- At the end of this topic, you should be able to
  - Understand what is thread and process
  - Define, Instantiate and Start a thread object
  - Start and running multiple threads in a program

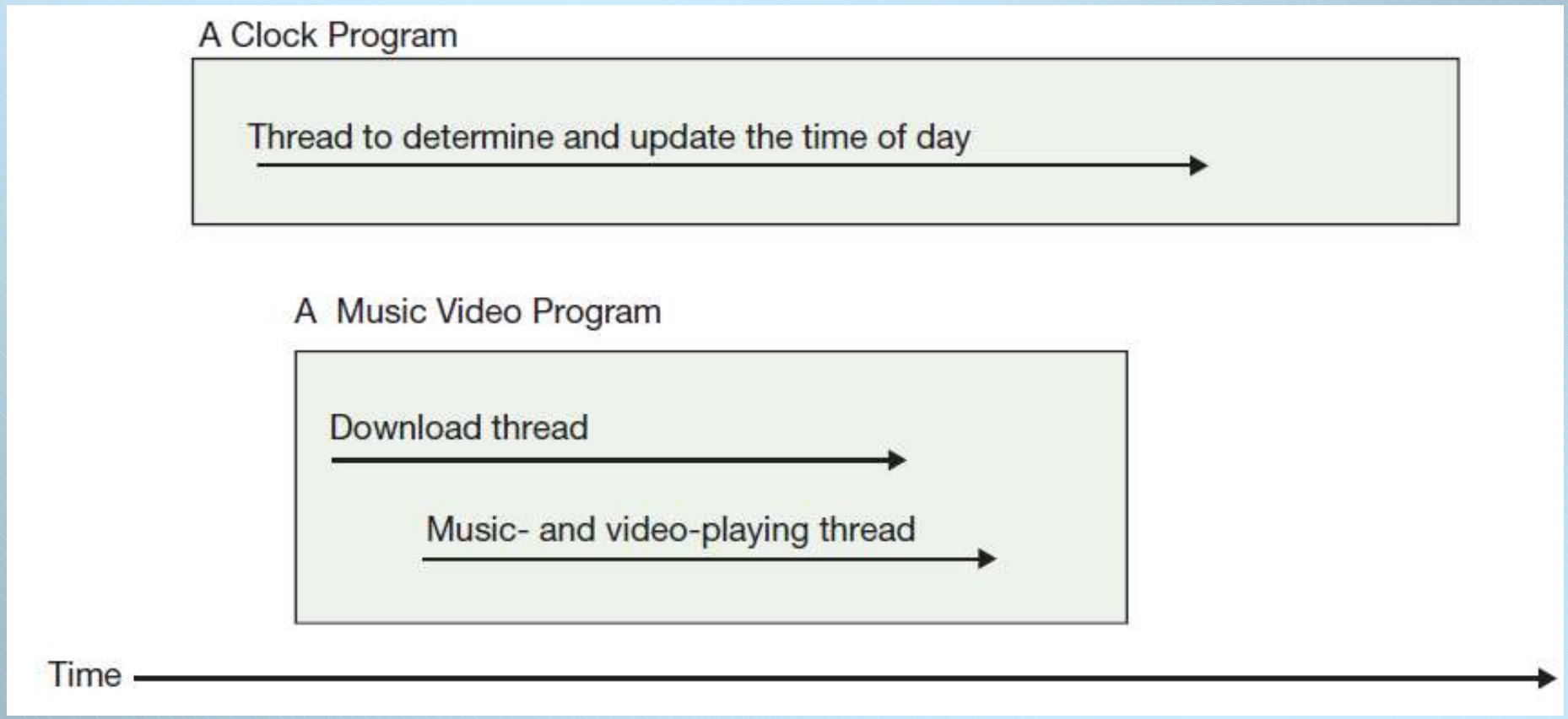
What is a process?

- In computer science, concurrency is the concept of executing several programs, or several parts of a program, at the same time
- A program in execution is referred to as a process
- A multitasking OS can create several processes and run them on multiple CPUs or the cores of a single CPU
- The OS will switch the CPU between these processes, giving the illusion of concurrently running the processes (context switching)
- Each process contains its own memory space and does not share it with the other processes

What is a thread?

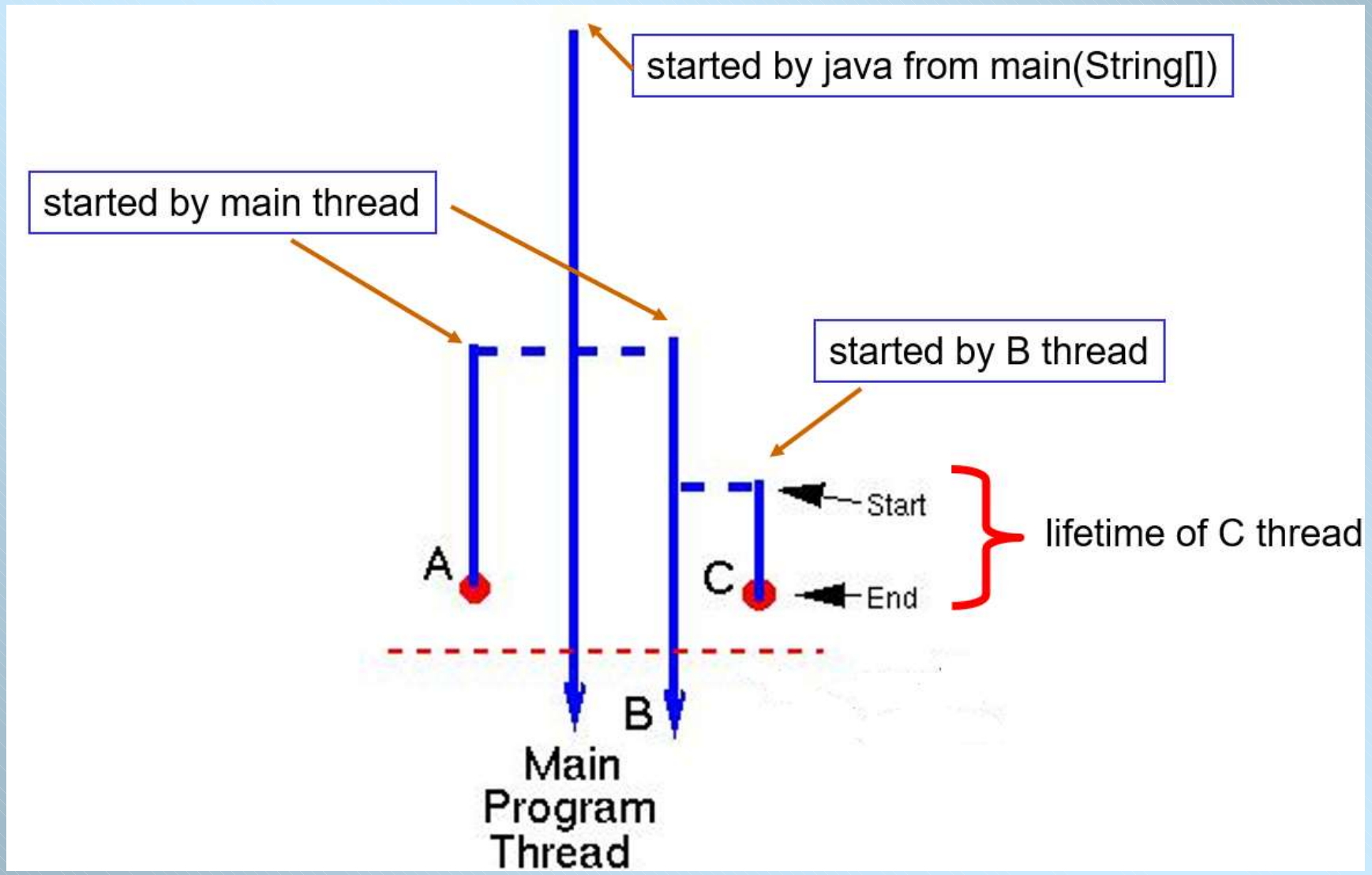
- A thread is a subprocess within a program, a.k.a lightweight process
- A thread represents a separate execution path within a program
- Multiple threads within a process can run concurrently & they share the process's resources such as memory & CPU time
  - E.g.: in a Web browser we may do the following tasks concurrently:
  - 1. scroll a page,
  - 2. download a file,
  - 3. play sound,
  - 4 print a page.
- Context switching between threads is generally faster than context switching between different processes

Single-Thread Process vs Multi-Thread Process



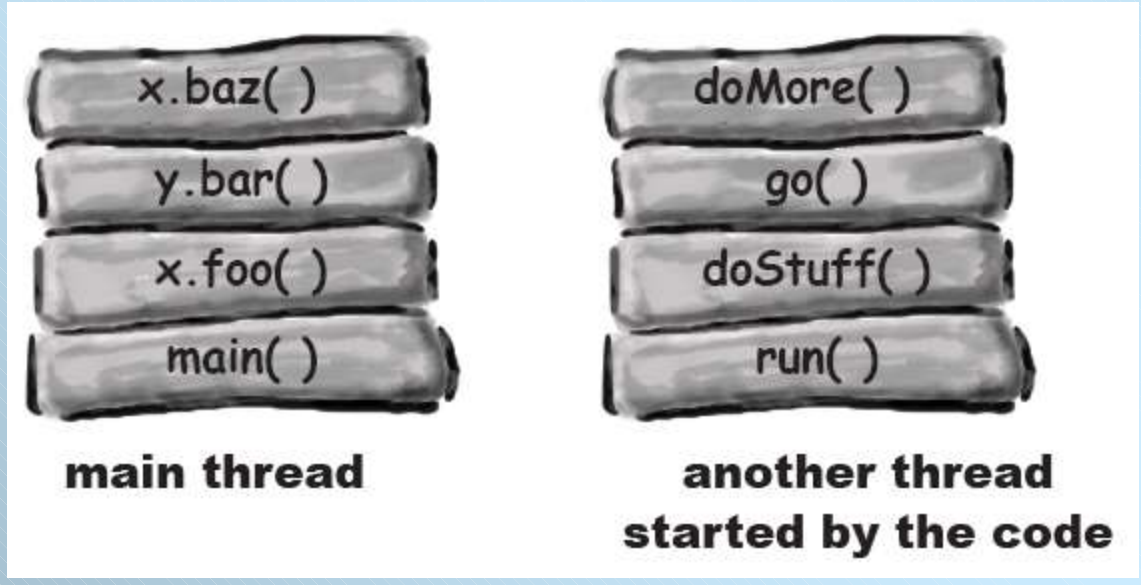
Multi-threads in Java

All Java program has at least one main thread created by the JVM that runs the codes in the main() method



Multi-threads in Java

- Each thread in Java has its own call stack.
- A call stack is a data structure used by the JVM to track method invocations and execution contexts.
- When a method is called, a new frame is created on top of the call stack
- The JVM is responsible for starting the main thread & the main() method is put on the bottom of the stack.



Multi-threads in Java

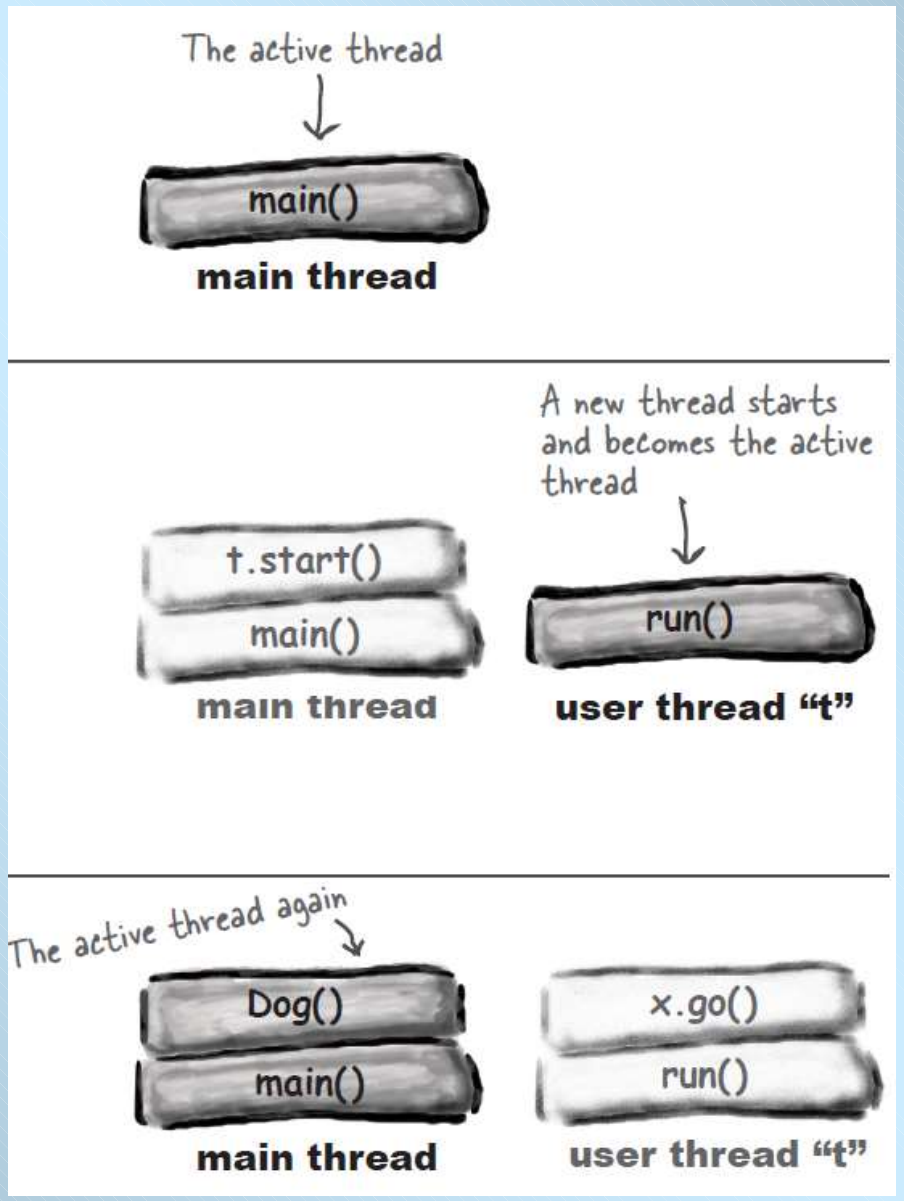
1. The JVM is responsible for starting the main thread & the main() method is put on the bottom of the main call stack.
2. If main() starts a new thread, a new call stack is created. The main thread may be temporarily frozen while the new thread starts running.
3. The JVM switches between the new thread and the original main thread, until both threads complete.

Multi-threads in Java

1.

2.

3.



Defining Thread

Two ways to define a thread:

1. Extend the Thread class
2. Implement the Runnable interface

Defining Thread

1. Extend the Thread class

- Define a subclass of the Thread class
- Override the run() method

Example:

```
Class MyThread extends Thread {
   public void run() {
       System.out.println("Hello! From MyThread");
   }
}
```

Defining Thread

2. Implement the Runnable interface

- Define a class that implements the Runnable interface
- Override the run() method

Example:

```
Class MyRunnable implements Runnable {
   public void run() {
       System.out.println(“Hello! From MyThread");
   }
}
```

Defining Thread

2. Implement the Runnable interface

- Since the Runnable interface is a functional interface, a lambda expression can used to implement the run() method.

Example:

```
Runnable  r = () ->
      System.out.println("Hello! From MyThread");
```

Creating Thread object

- A thread object must be instantiated before the thread can be executed
- If thread is defined as a subclass of Thread class just create an object from the subclass:

```
MyThread  t = new MyThread();
```

Creating Thread object

- If thread is defined as a class that implements the Runnable interface:
  - Instantiate the Runnable object
  - Pass the Runnable to the Thread object

```
MyRunnable  r = new MyRunnable();
Thread t = new Thread(r);
```

OR

- Pass the lambda expression to the Thread object

```
Thread t = new Thread(r);
```

Thread Constructors

- Some overloaded constructors of Thread class
  - Thread()
  - Thread(Runnable target)
  - Thread(Runnable target, String name)
  - Thread(String name)

Starting a Thread

- To start the execution of a thread object in a new call stack, call the thread object start() method

t.start();

- The thread move from a " new " state to the
- "runnable" state
- When the thread gets the CPU time, it moves to the “running” state, and the run() method will run
- If the run() method completes, the thread moves to the “terminated" state

Starting a Thread

NOTE: Calling the run() method directly :

t.run();

is legal but will not start a new thread and its call stack
(The run() method just goes to the current call stack)

Starting & Running Multiple Threads

- In this example, there is a single Runnable & 3 Thread objects
- Each thread is given a unique name:

```
public class ManyNames {
    public static void main(String[] args) {
        Runnable r = () -> {
            for (int x = 1; x <= 3; x++) {
                System.out.println("Run by " + Thread.currentThread().getName() + ", x is " + x);
            }
        };
        Thread one = new Thread(r,"Ali");
        Thread two = new Thread(r,"Bob");
        Thread three = new Thread(r,"Jim");
        one.start();
        two.start();
        three.start();
    }
}
```

Starting & Running Multiple Threads

- Each thread will start and each thread will run to completion
- A thread is done(dies) when its run() method completes
- Once a thread has been started, it can never be started again
- The output of the program cannot be guaranteed to be the same every time the program is executed
- The order of which runnable threads are chosen to be run (by the Thread scheduler) is not guaranteed