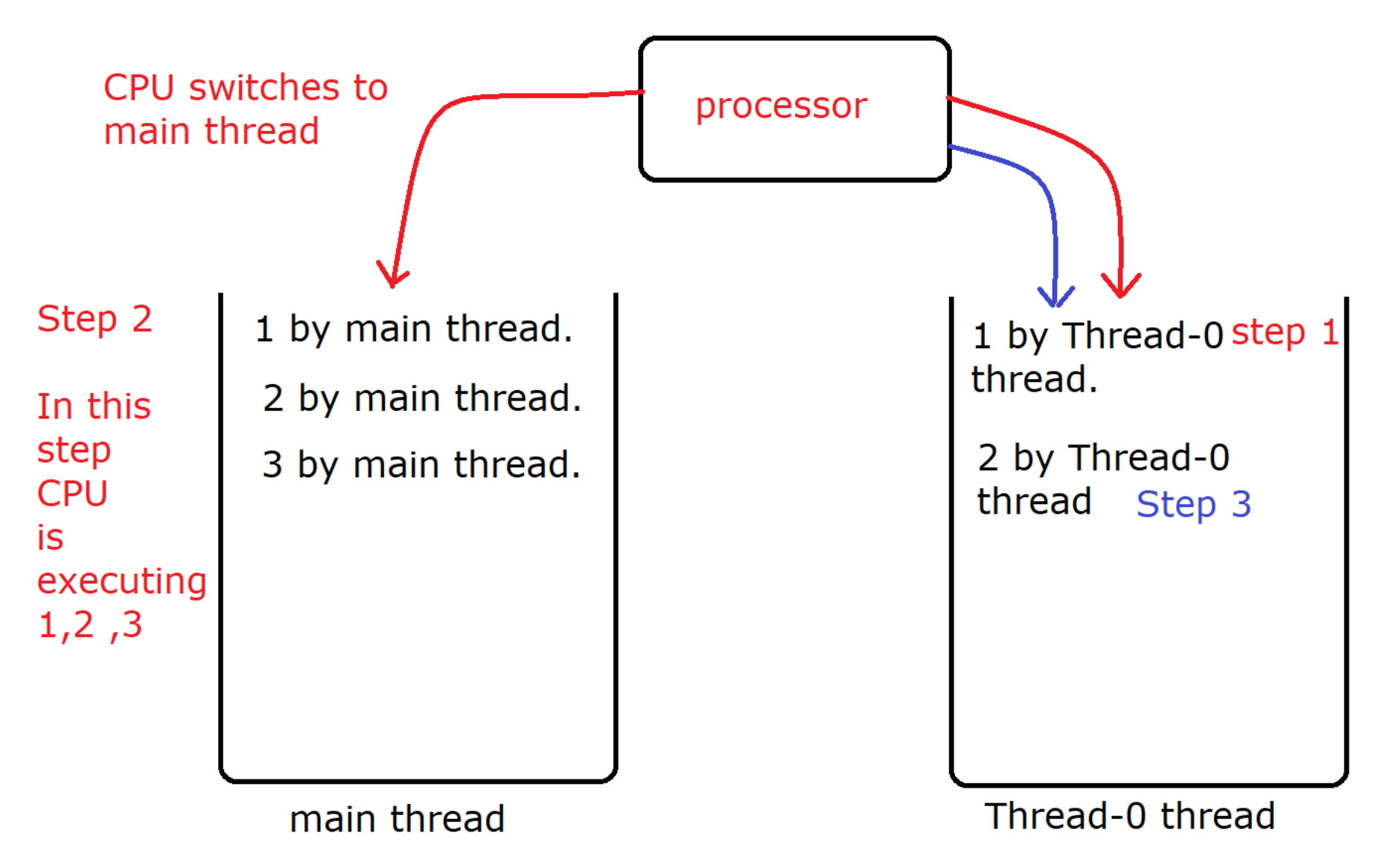
WAP to show that while working with multithreading, the output is unpredictable.

```java
package com.ravi.basic;

class Sample extends Thread
{
    @Override
    public void run()
    {
        String name = Thread.currentThread().getName();
        for(int i=1; i<=10; i++)
        {
            System.out.println(i+" by "+name+" thread.");
        }

    }
}

public class ThreadLoop
{
    public static void main(String[] args)
    {
        Sample s1 = new Sample();
        s1.start();

        String name = Thread.currentThread().getName();

        for(int i=1; i<=10; i++)
        {
            System.out.println(i+" by "+name+" thread.");
        }

        int x = 1;
        do
        {
            System.out.println("Java");
            x++;
        }
        while(x<=10);

    }

}
```

> Note : Here processor is frequently switching from main thread to Thread-0 thread so output is un-predicatable.
>
> We have something called Thread Schedular which is responsible to schedule the thread that means it is schedular who will decide which thread will get the processor time.

Diagram for the above program :



* In order to hold the currently executing instruction of each Thread, We have a separate PC register.

How to set and get the name of the Thread :

```java
public class Thread implements Runnable
{
   private volatile String name;  //non static field

   public final synchronized void setName(String name) //setter
   {
       this.name = name;
   }

   public final String getName()  //getter
   {
      return this.name;
   }
}
```

Whenever we create a userdefined Thread in java then by default JVM assigns the name of thread is Thread-0, Thread-1, Thread-2 and so on.

If a user wants to assign some user defined name of the Thread, then Thread class has provided a predefined method called setName(String name) to set the name of the Thread.

On the other hand we want to get the name of the Thread then Thread class has provided a predefined method called getName().

public final synchronized void setName(String name) //setter

public final String getName() //getter

```java
package com.ravi.basic;

class DoStuff extends Thread
{
    @Override
    public void run()
    {
        String name = Thread.currentThread().getName();
        System.out.println("Running Thread name is :"+name);
    }
}
public class ThreadName1
{
    public static void main(String[] args)
    {
        DoStuff t1 = new DoStuff();
        DoStuff t2 = new DoStuff();

        t1.start();
        t2.start();


    System.out.println(Thread.currentThread().getName()+" thread is running.....");
    }
}
```

We are not providing the user-defined names so by default the name of thread would be Thread-0, Thread-1.

```java
package com.ravi.basic;

class Demo extends Thread
{
    @Override
    public void run()
    {
         String name = Thread.currentThread().getName();
         System.out.println("Running Thread name is :"+name);
    }
}
public class ThreadName2
{
    public static void main(String[] args)
    {
        Thread t = Thread.currentThread();
        t.setName("Parent");

        Demo d1 = new Demo();
        Demo d2 = new Demo();

        d1.setName("Child1");
        d2.setName("Child2");

        d1.start();
        d2.start();

        String name = Thread.currentThread().getName();
        System.out.println(name + " Thread is running Here..");


    }
}
```

Note : Here we are providing the user-defined name i.e child1 and child2 for both the user-defined thread.

```java
package com.ravi.basic;

import java.util.InputMismatchException;
import java.util.Scanner;

class BatchAssignment extends Thread
{
    @Override
    public void run()
    {
        String name = Thread.currentThread().getName().toLowerCase();

        if(name !=null && name.equals("placement"))
        {
            this.placementBatch();
        }
        else if(name !=null && name.equals("regular"))
        {
            this.regularBatch();
        }
    }


    public void placementBatch()
    {
        System.out.println("I am a placement batch student.");
    }

    public void regularBatch()
    {
        System.out.println("I am a Regular batch student.");
    }
}


public class ThreadName3
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        try(sc)
        {
            System.out.print("Enter your Batch Title [Placement/Regular] :");
            String title = sc.next();

            BatchAssignment b = new BatchAssignment();
            b.setName(title);

            b.start();
        }
        catch(InputMismatchException e)
        {
            System.out.println("Invalid Input");
        }


    }

}
```

public static void sleep(long millisecond) :

* It is a predefined static method of Thread class.

* It is used to put a thread into temporarly waiting state (TIMED_WAITING), the waiting time of current thread will depend upon the time specified by the user as a **parameter** in millisecond.
Example :
Thread.sleep(1000); // The thread will wait for 1 sec

* It throws a **checked exception** i.e. java.lang.InterruptedException because this sleeping thread may be interrupted at runtime so provide either try catch OR declare the method as throws.

```java
package com.ravi.sleep;

class Sleep extends Thread
{
    @Override
    public void run()
    {
        String name = Thread.currentThread().getName();

        for(int i=1; i<=10; i++)
         {
            System.out.println(i+" by "+name+" thread");

            try
            {
                Thread.sleep(1000); //thread will wait for 1 sec
            }
            catch(InterruptedException e)
            {
                System.err.println("Thread is interrupted");
            }
         }
    }
}
public class SleepDemo1
{
    public static void main(String[] args)
    {
      Sleep s1 = new Sleep();
      s1.setName("Child1");
      s1.start();
    }
}
```