UNIT – 3
PART – 3
MULTITHREADING

Multitasking and Multithreading

- Multitasking refers to a computer’s ability to perform multiple jobs concurrently
  - More than one program are running concurrently
- A thread is s single sequence of execution within a program
- Multithreading refers to multiple threads of control within a single program

Multitasking

- Multitasking is a process of executing multiple tasks simultaneously.
- We use multitasking to utilize the CPU.
- Multitasking can be achieved in two ways:
  - Process-based Multitasking (Multiprocessing)
  - Thread-based Multitasking (Multithreading)

Process-based Multitasking (Multiprocessing)

- Each process has an address in memory.
- In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

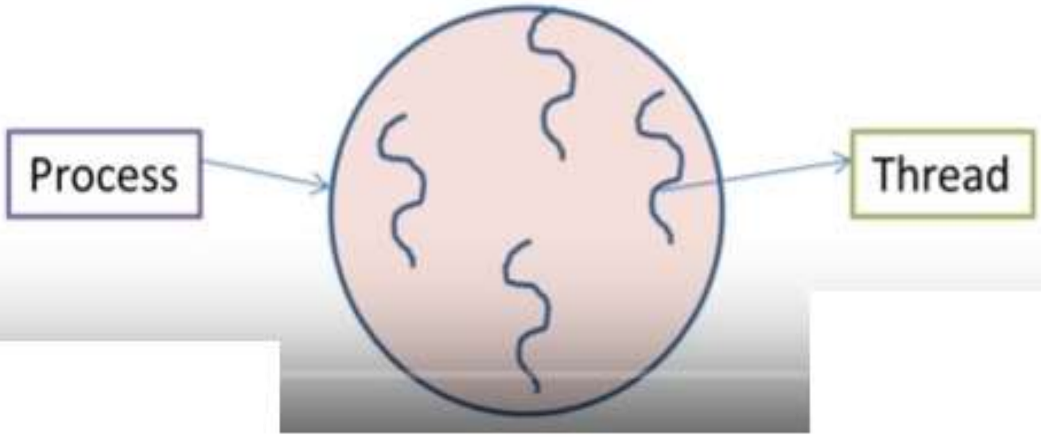
Multithreading

Handling multiple tasks – Multitasking
Ability to initiate multiple processes – Multithreading

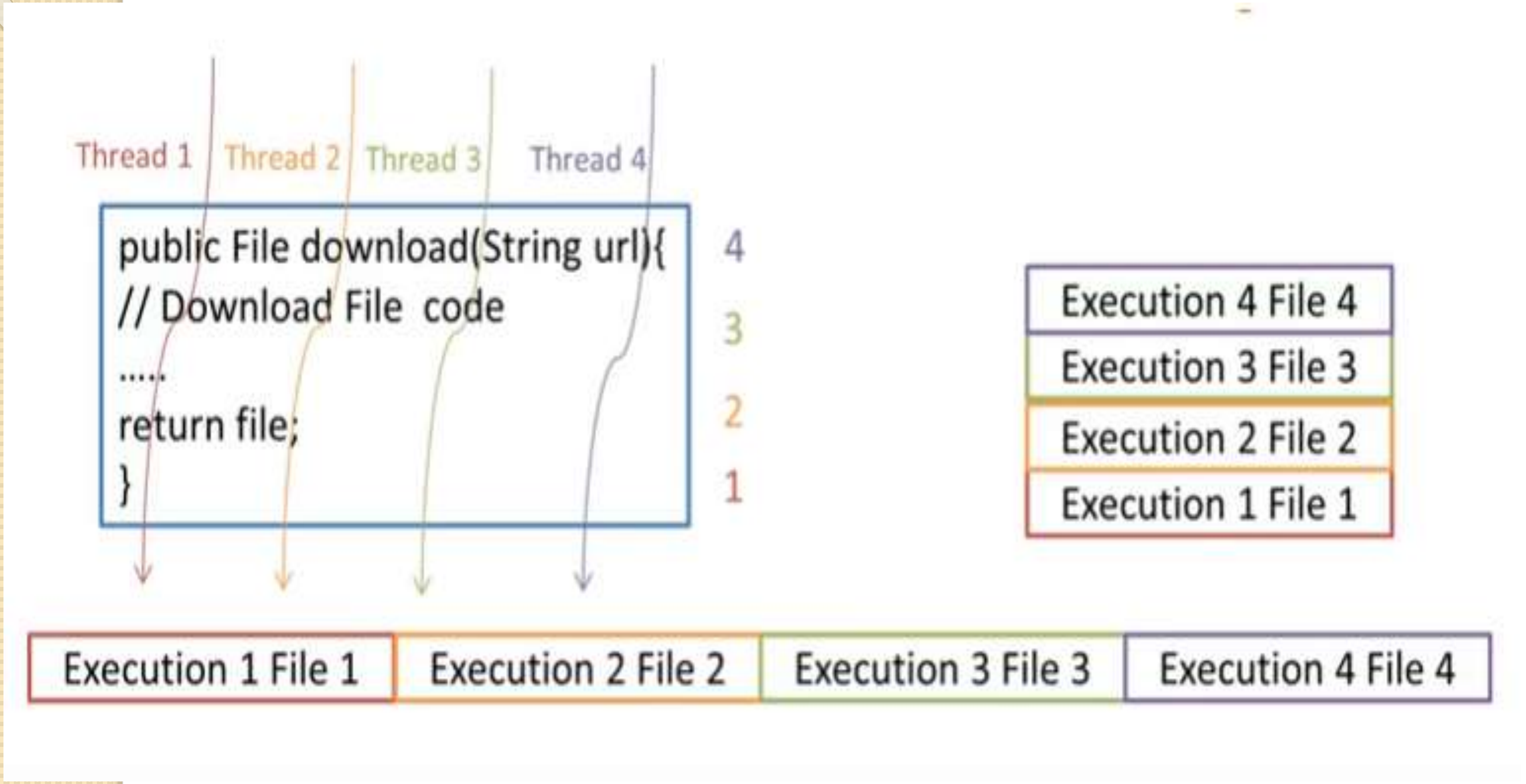
Process v/s Thread

A single application running in OS is a process.
A process can have multiple threads.

Spell checker in Word can be considered as a thread.



Multithreading



```
public File download(String url){
// Download File  code
.....
return file;
}
```

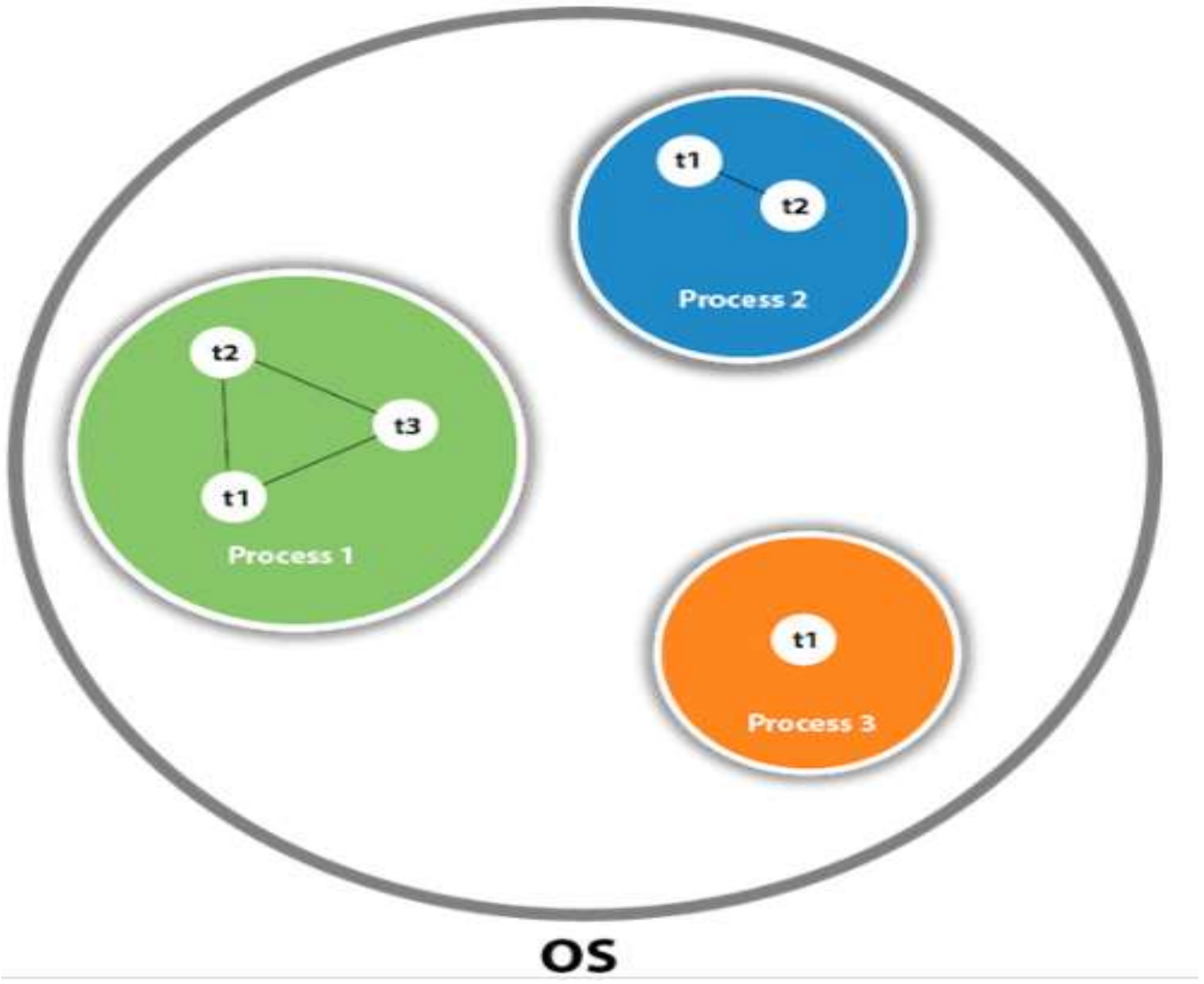
Multithreading

- Multithreading in Java is a process of executing multiple threads simultaneously.
- Thread is basically a **lightweight sub-process**, a smallest unit of processing.
- **Multiprocessing and multithreading, both are used to achieve multitasking.**
- Java Multithreading is mostly **used in games, animation** etc.
- When main() method is called a thread known as main thread is created to execute the program.
- It is the OS which schedules the threads to be processed by the processor. So the scheduling behavior is dependent on the OS.
- Nothing can be guaranteed about the threads execution.

What is a thread?

- Threads are separate parts of execution which are functionally independent of each other.
- Multithreading as the name itself tells that it is regarding, multi tasks(thread).
- **process:**
  - A process consists of the memory space allocated by the operating system that can contain one or more threads.
  - A thread cannot exist on its own; it must be a part of a process.
  - A process remains running until all of the threads are done executing.
- **Use:** Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

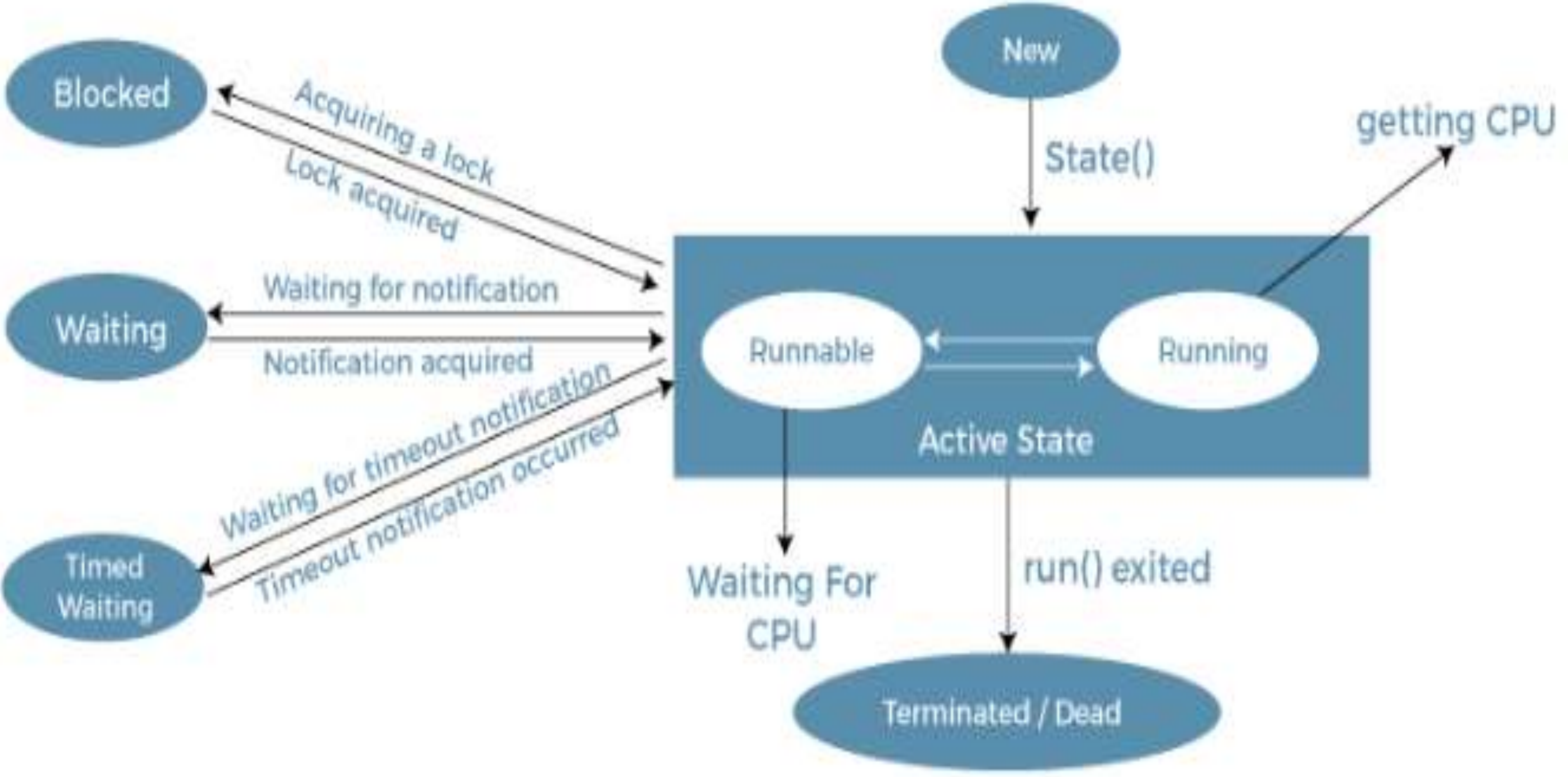
Thread

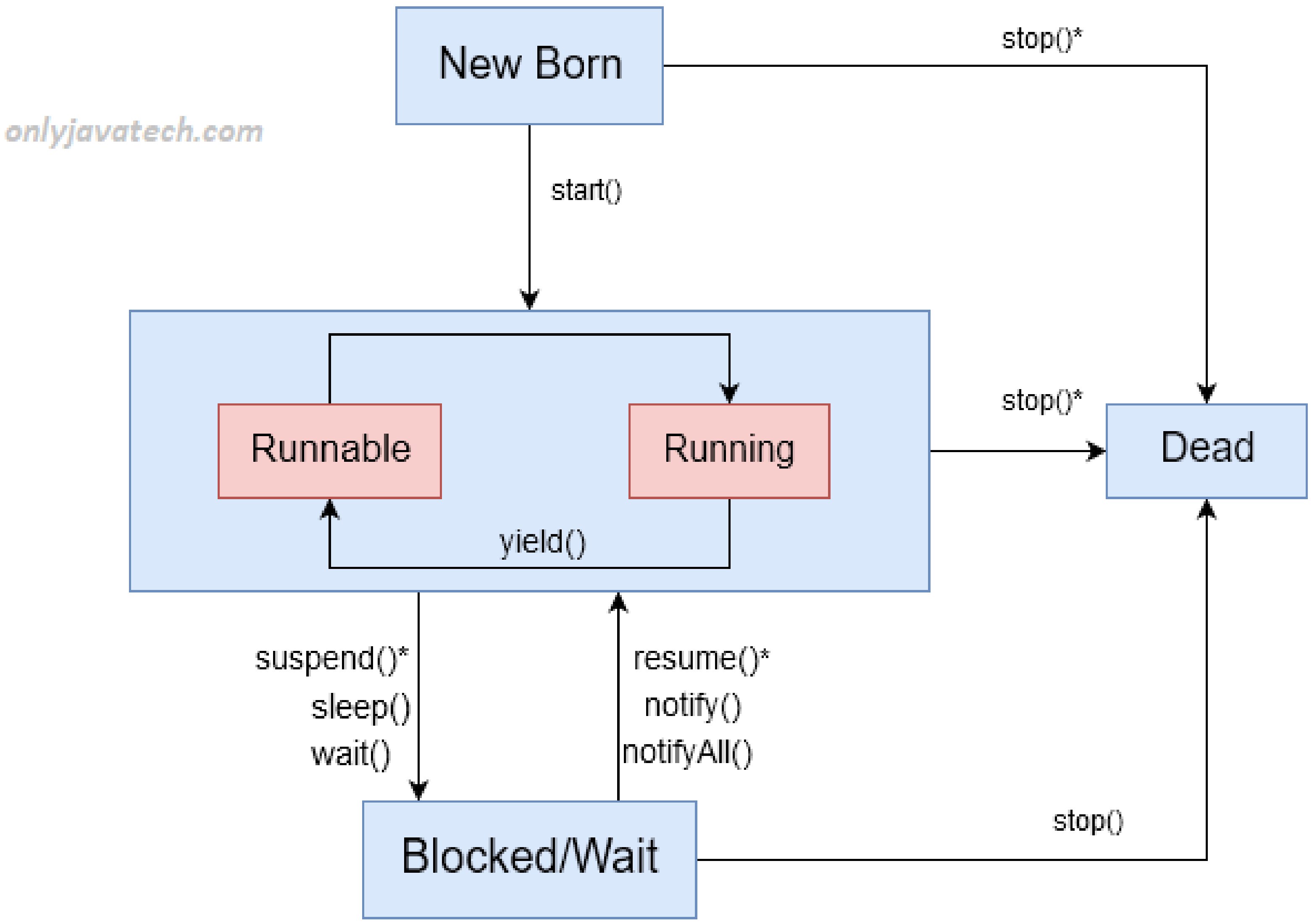


Life cycle of Thread

- There are five stages in thread life cycle:
  - Newborn State
  - Runnable State
  - Running State
  - Blocked State
  - Dead State

Thread Life Cycle





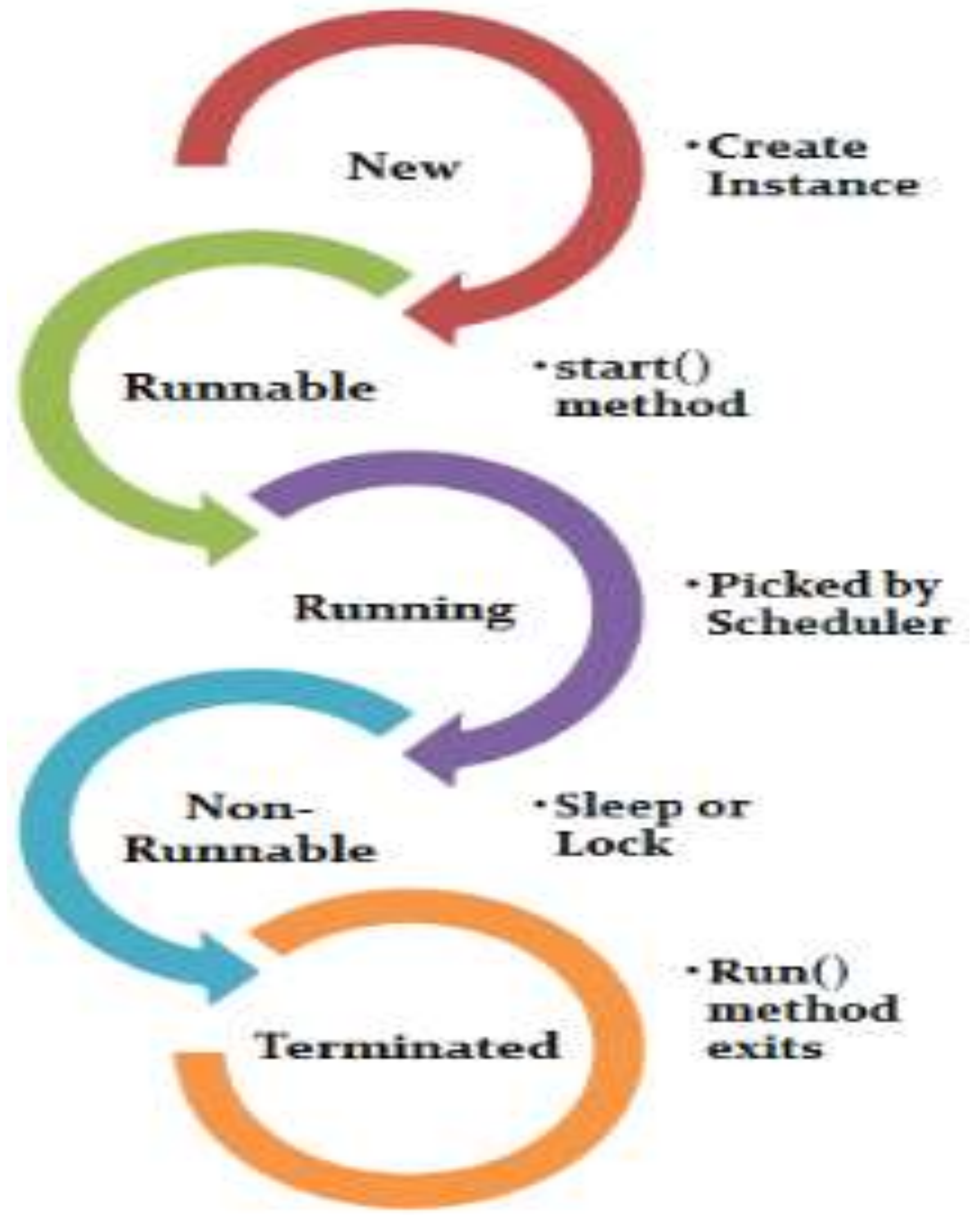
*** denote deprecated Methods**

Fig : Life Cycle of Thread

Life Cycle….

- **New :**
  - The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- **Runnable :**
  - The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- **Running:**
  - The thread is in running state if the thread scheduler has selected it.
- **Non-Runnable (Blocked) :**
  - This is the state when the thread is still alive, but is currently not eligible to run.
- **Terminated :**
  - A thread is in terminated or dead state when its run() method exits.

Thread Life Cycle



Creating a Thread

- Java defines 2 ways to achieve multithreading :
  - By extending the **Thread** class itself.
  - By implementing the **Runnable** interface

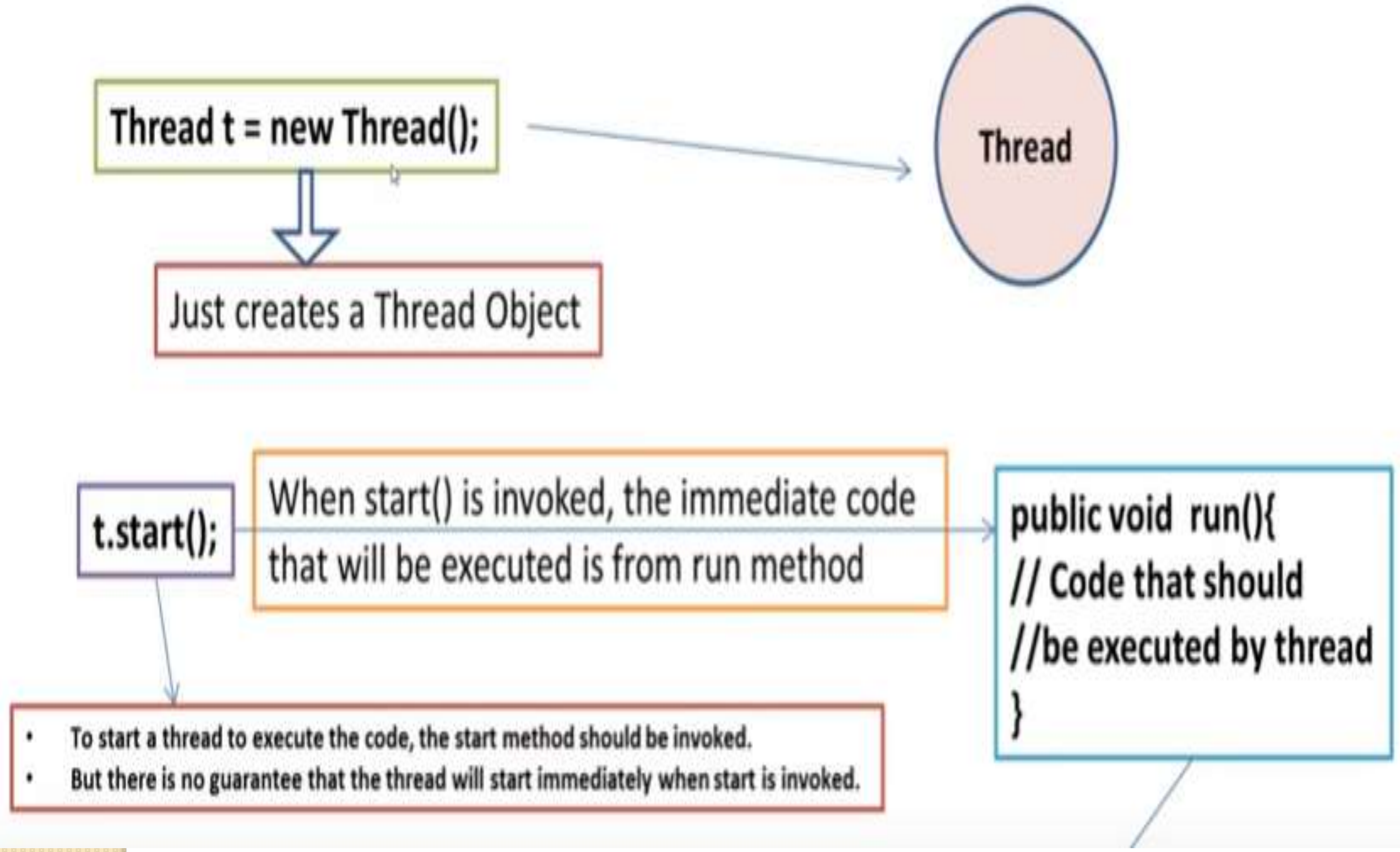
- Extending a Thread class :

```
        class Main extends Thread
        {
                    public void run()
                   {
                    System.out.println("This code is running in a thread");
                   }
        }
```

- Implementing Runnable interface :

```
        class Main implements Runnable
        {
                   public void run()
                   {
                   System.out.println("This code is running in a thread");
                   }
        }
```

Creating Threads



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

Just creates a Thread Object

```
t.start();
```

When start() is invoked, the immediate code that will be executed is from run method

```
public void  run(){
// Code that should
//be executed by thread
}
```

- To start a thread to execute the code, the start method should be invoked.
- But there is no guarantee that the thread will start immediately when start is invoked.

Thread Class

- **Thread() :**

  Allocates a new Thread object.
- **Thread(String name):**

  Allocates a new Thread object with user define name.
- **Thread(Runnable target):**

  Allocates a new Thread object

  target - the object whose run method is called.
- **Thread(Runnable target, String name):**

  Allocates a new Thread object

  target - the object whose run method is called.

  name - the name of the new thread.

Methods…

Method Name	Description
setName()	to give a name to thread
getName()	return thread's name
getPriority()	return thread's priority
setPriority()	Set priority of thread
isAlive()	checks if thread is still running or not
join( )	Wait for a thread to end
run( )	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method

Methods…

Method Name	Description
resume()	Resumes this Thread’s execution.
suspend()	Suspends this Thread's execution.
void yield()	Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Create Thread by Extending Thread:

- The First way to create a thread is to create a new class that extends Thread, and then to create an instance of that class.
- The extending class must override the run() method, which is the entry point for the new thread.
- It must also call start( ) to begin execution of the new thread.
- Example

Create Thread by Implementing Runnable

- The easiest way to create a thread is to create a class that implements the Runnable interface.
- To implement Runnable, a class needs to only implement a single method called run( ), which is declared like this:
- **public void run( )**
  - You will define the code that represents the new thread inside run() method.
- Example

Cont…

- After creating a class that implements Runnable, have to instantiate an object of type **Thread** from within that class.
- Thread defines several constructors. The one that we will use is shown here:
- **Thread(Runnable threadOb, String threadName);**
- Here, **threadOb** is an instance of a class that implements the Runnable interface and the name of the new thread is specified by **threadName**.
- After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread.

Using Sleep()

- Syntax:
  - static void sleep(long miliseconds)
- used to sleep(pause) a thread for the specified milliseconds of time.
- Example

Priority of a Thread

- Priorities are represented by a number between 1 and 10.

- **Constants of Thread class:**
  - public static int MIN_PRIORITY
  - public static int NORM_PRIORITY
  - public static int MAX_PRIORITY

- Default priority of a thread is 5 (NORM_PRIORITY).

- The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

- Example