OBJECT ORIENTED ANALYSIS & DESIGN DATA STRUCTURES & ALGORITHMS

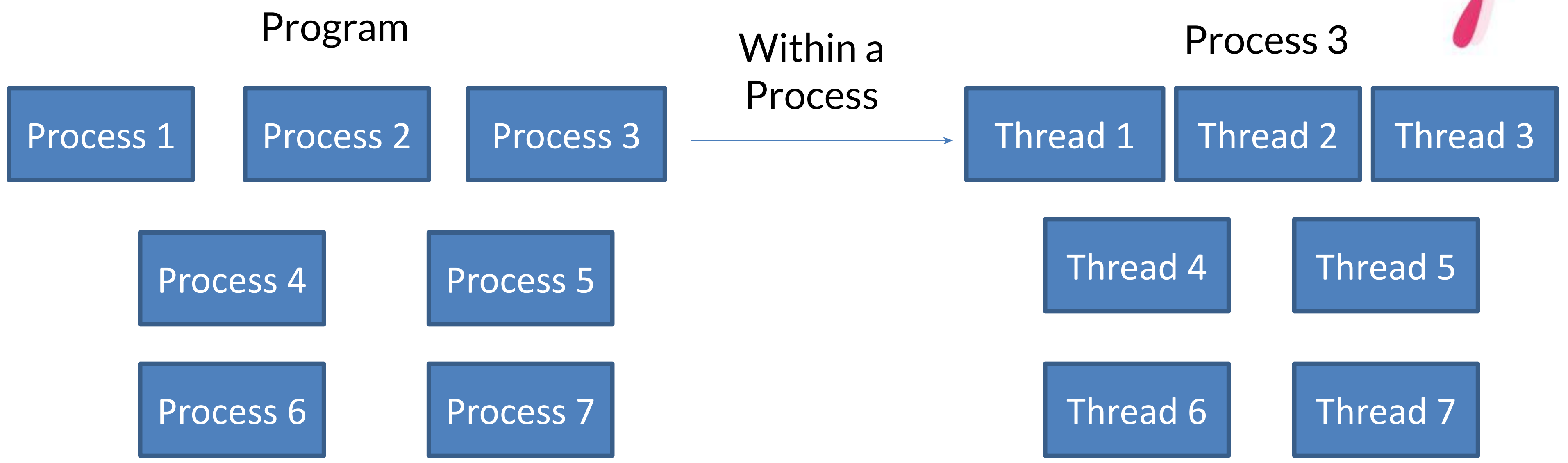
Multi Threading

Program Vs Process Vs Thread

- **Program**: A program is a set of instructions stored in the secondary storage device that are intended to carry out a specific job. It is read into the primary memory and executed by the kernel.

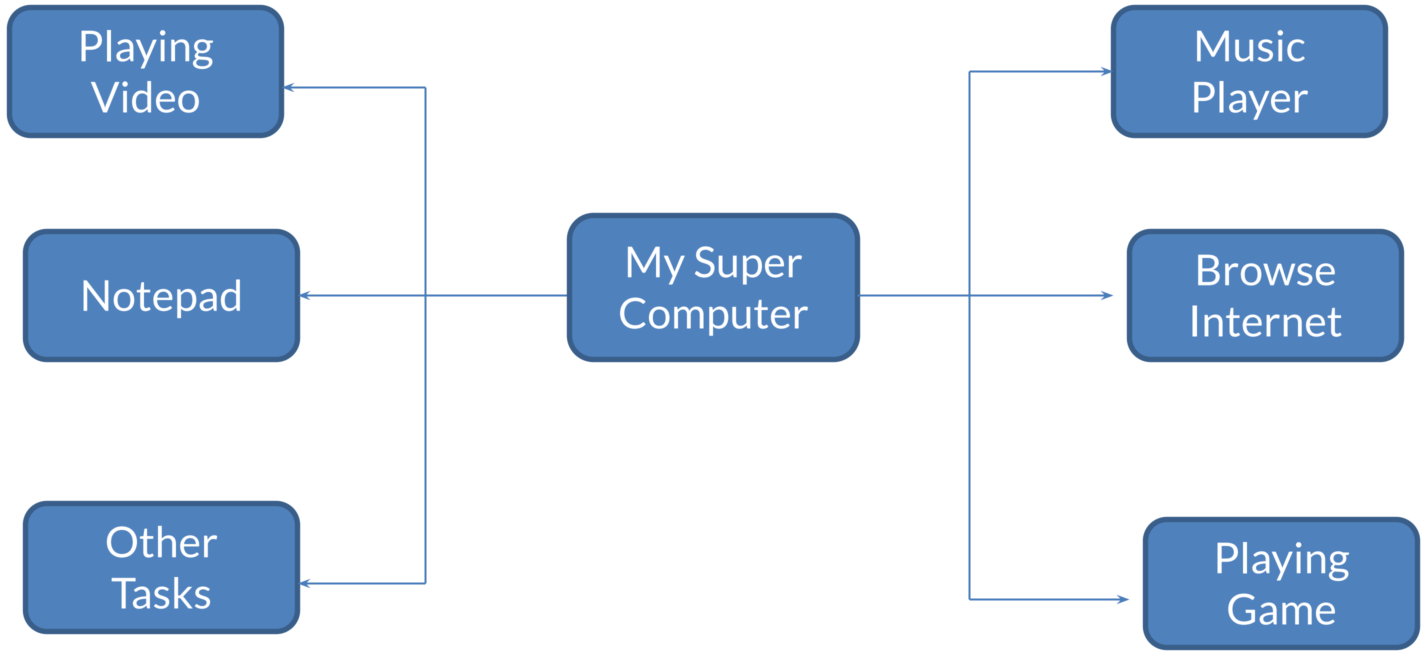
- **Process** : An executing instance of a program is called a process. It is also referred as a task.

- **Thread** : A thread is called a 'lightweight process'. It is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel.


Program
Process 1
Process 2
Process 3
Process 4
Process 5
Process 6
Process 7
Within a Process
Process 3
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7

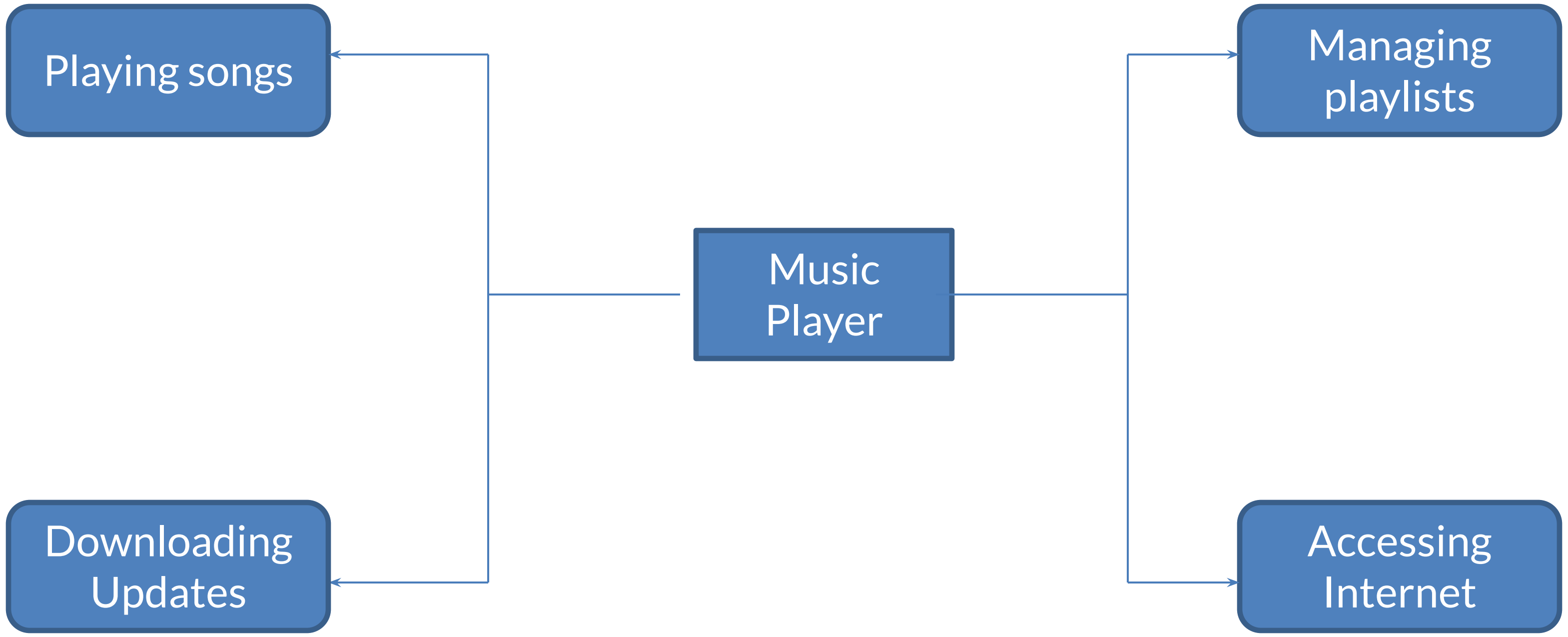
Multi Threading

Every computer has multiple processes running at a time.



Multi Threading (contd.)

Every application has multiple sub-process running at a time.



Multi Threading (contd.)

- Thread is a task to be performed. Multi threading is multiple tasks getting executed at the same time in the same program/process.
- Multi threading takes the same memory space for any number of threads.

What is a Java Thread?

- Thread is a lightweight sub process
- It is the smallest independent unit of a program
- Contains a separate path of execution
- Every Java program contains at least one thread
- A thread is created & controlled by the java.lang.Thread class

Creating a Thread

Creating A Thread

A thread in Java can be created using two ways

Thread Class

```
public class Thread
        extends Object
             implements
Runnable
```

Runnable Interface

```
public interface Runnable
```

Thread Class | Runnable Interface

1. Create a Thread class
2. Override run() method
3. Create object of the class
4. Invoke start() method to execute the custom threads run()

```
public class MyThread extends Thread {
   public void run(){
    System.out.println("My Thread...");
  }
   public static void main(String[] args) {
     MyThread obj = new MyThread();
     obj.start();
}
```

Thread Class | Runnable Interface

1. Create a Thread class implementing Runnable interface

2. Override run() method

3. Create object of the class

4. Invoke start() method using the object

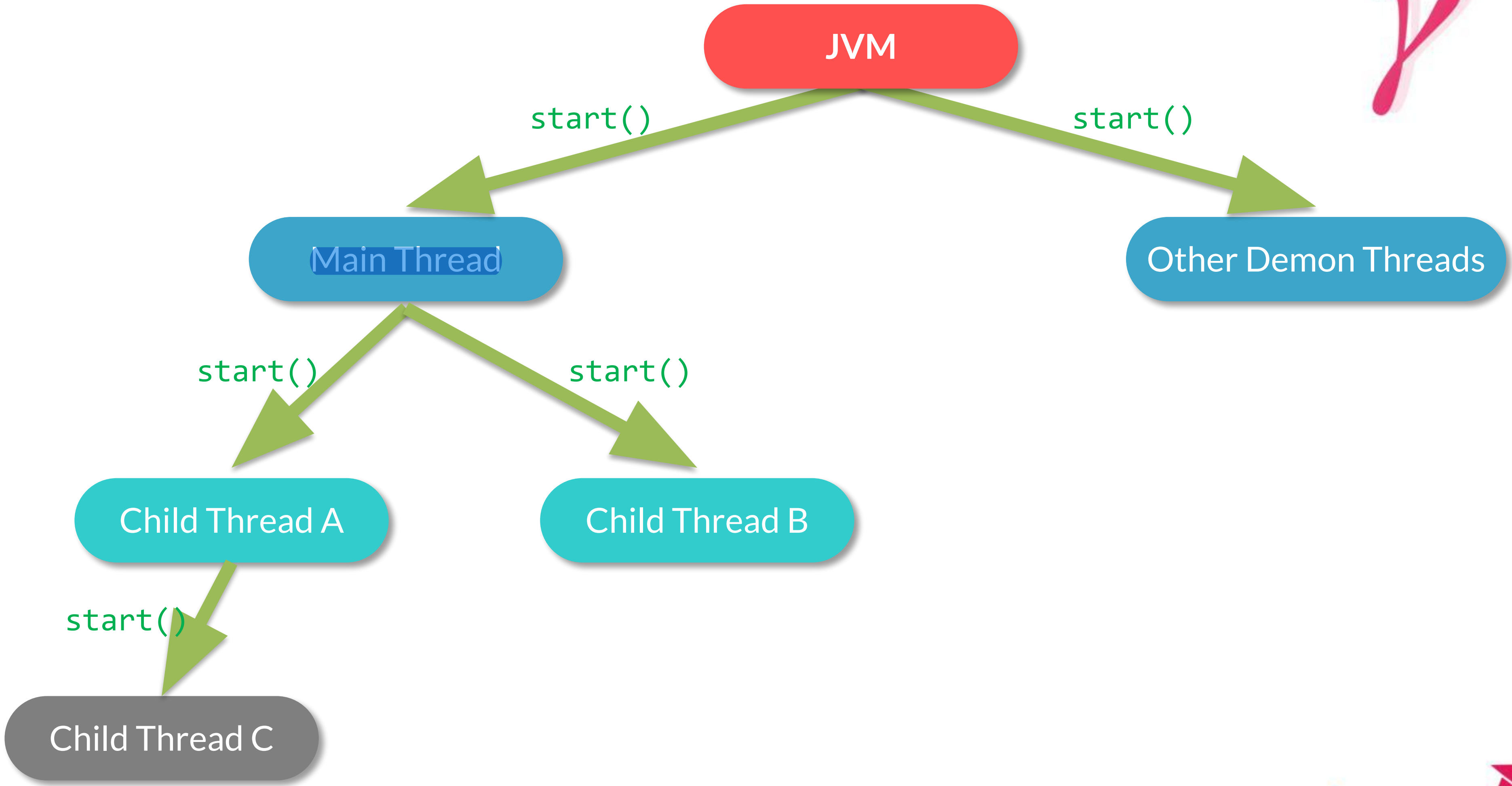
```
public class MyThread implements Runnable {
   public void run(){
     System.out.println(“My Thread…");
   }
   public static void main(String[] args) {
     Thread t = new Thread(new MyThread());
     t.start();
}
```

Java Main Thread

Java Main Thread

- Main thread is the most important thread of a Java Program
- It is executed whenever a Java program starts
- Every program must contain this thread for its execution to take place
- Java main Thread is needed because of the following reasons
    1. From this other “child” threads are spawned
    2. It must be the last thread to finish execution i.e when the main thread stops program terminates

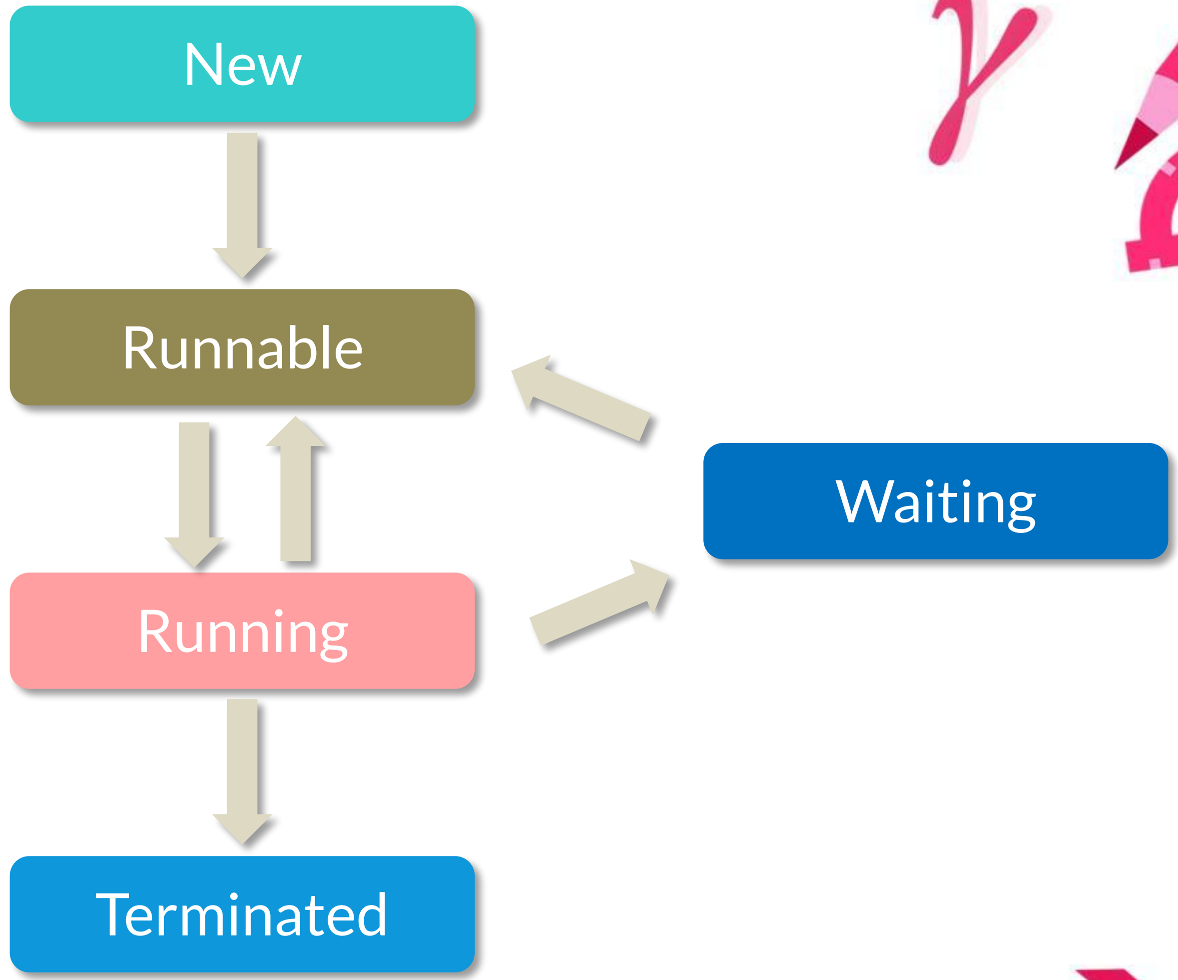
Java Main Thread



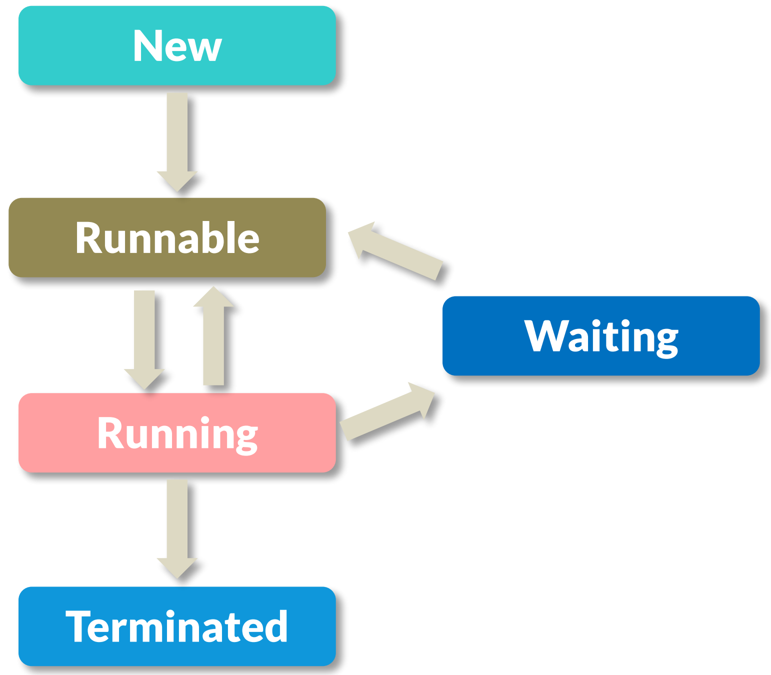
Java Thread Life-Cycle

Thread Lifecycle

A Java thread can lie only in one of the shown states at any point of time


$$\begin{tikzcd}
\text{New} \arrow[d] & \\
\text{Runnable} \arrow[d, shift right] & \\
\text{Running} \arrow[u, shift right] \arrow[d] \arrow[r] & \text{Waiting} \arrow[ul] \\
\text{Terminated} &
\end{tikzcd}$$

Thread Lifecycle



New

A new thread begins its life cycle in this state & remains here until the program starts the thread. It is also known as a **born thread**.

Runnable

Once a newly born thread starts, the thread comes under runnable state. A thread stays in this state is until it is executing its task.

Running

In this state a thread starts executing by entering run() method and the yield() method can send them to go back to the Runnable state.

Waiting

A thread enters this state when it is temporarily in an inactive state i.e it is still alive but is not eligible to run. It is can be in waiting, sleeping or blocked state.

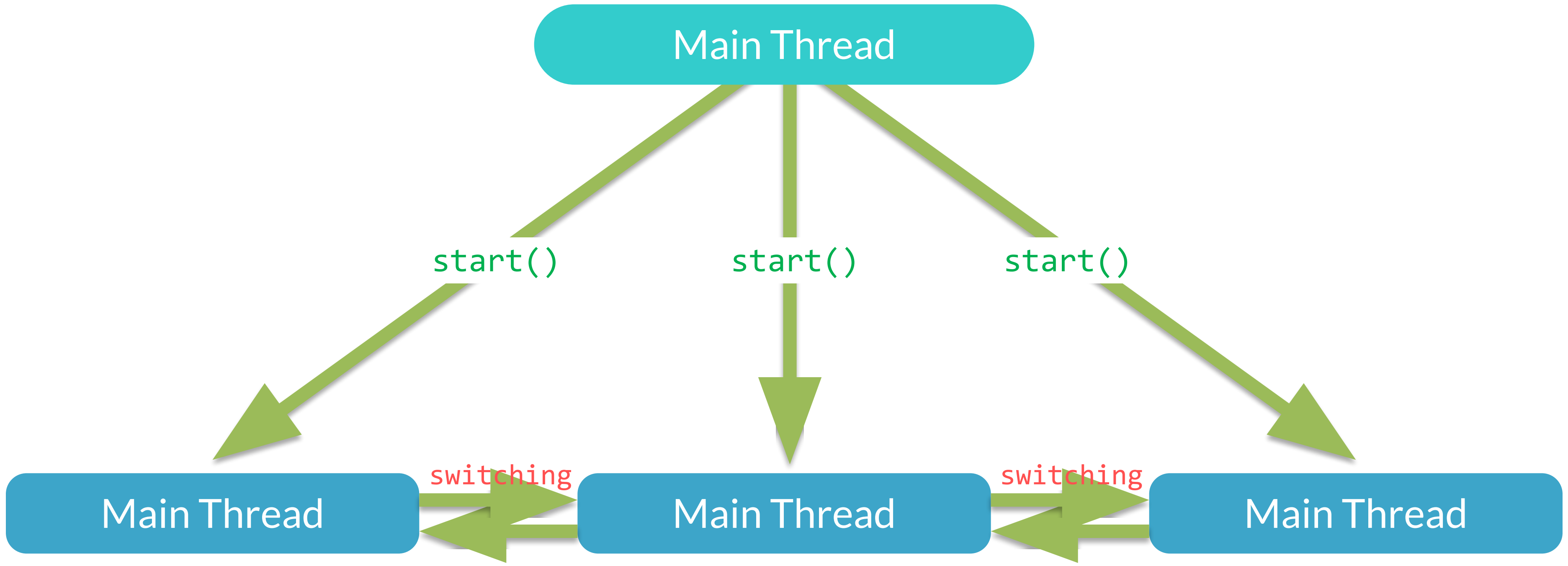
Terminated

A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Multi Threading In Java

Multi – Threading

Multi threading is the ability of a program to run two or more threads concurrently, where each thread can handle a different task at the same time making optimal use of the available resources



Thread Methods

Creating Multiple Threads

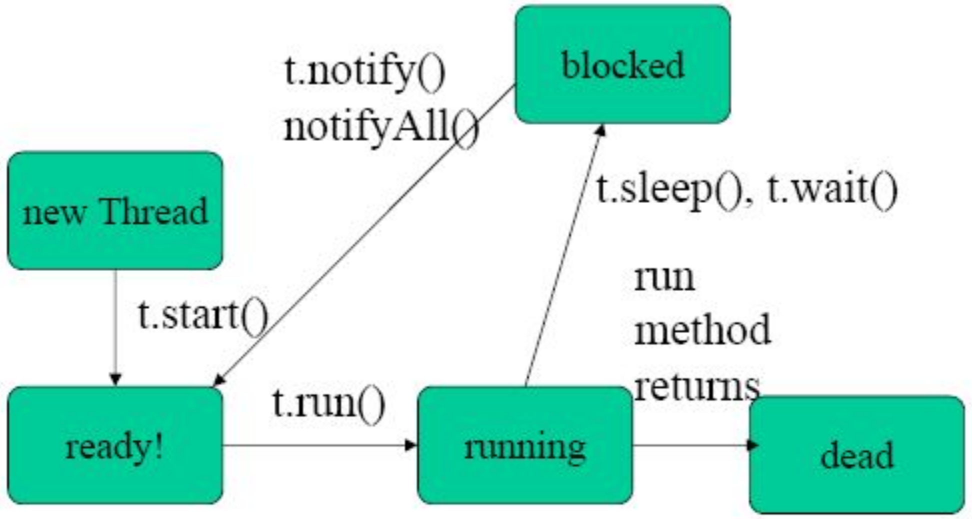
Joining Threads

Thread.sleep()

Inter Thread Communication

Daemon Thread

Thread -Methods

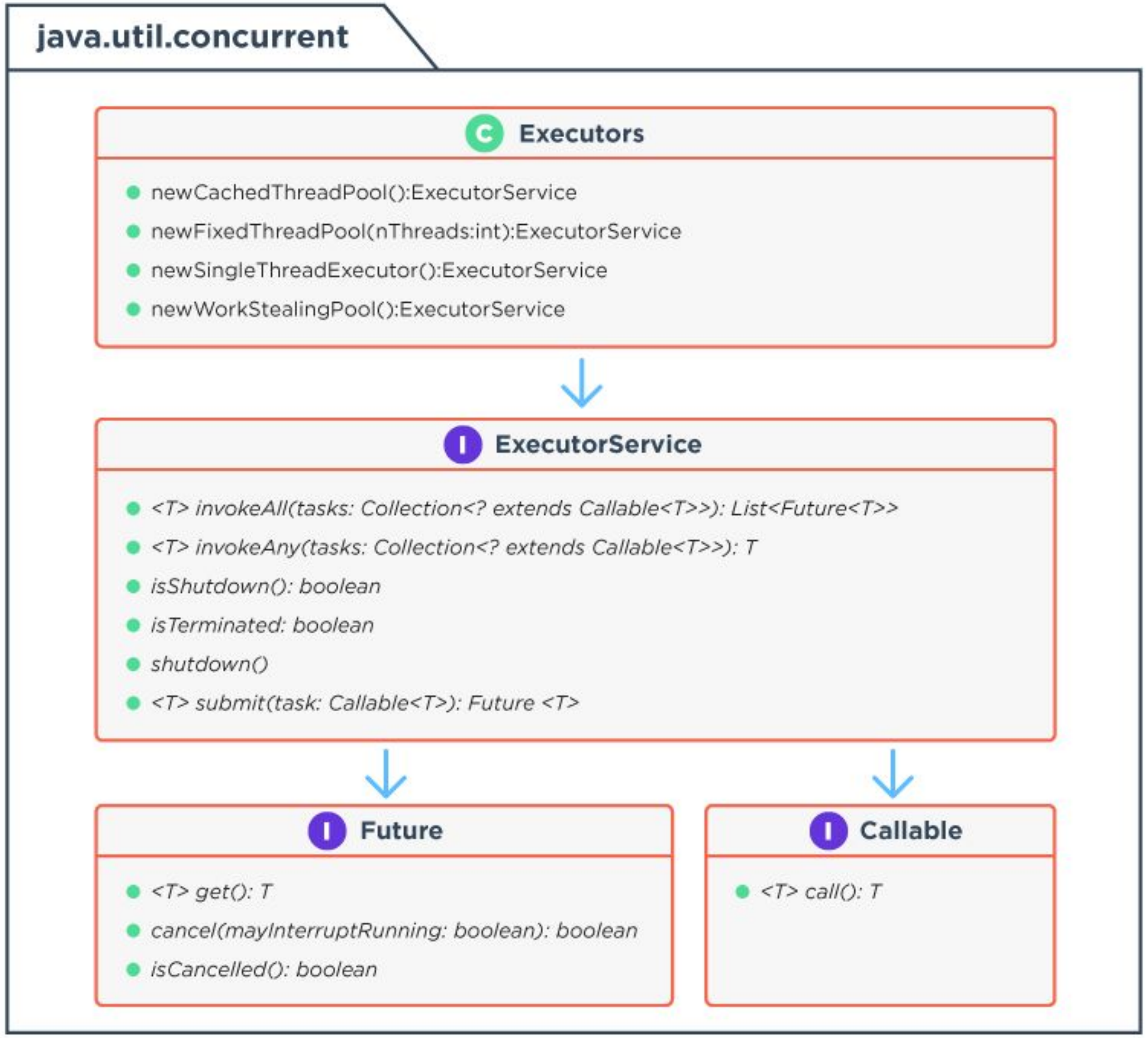


- Start() method should be called to start a thread.

- Start () will call the run () method.
- Run() will have the actual task to be performed by the thread.

Concurrent Package

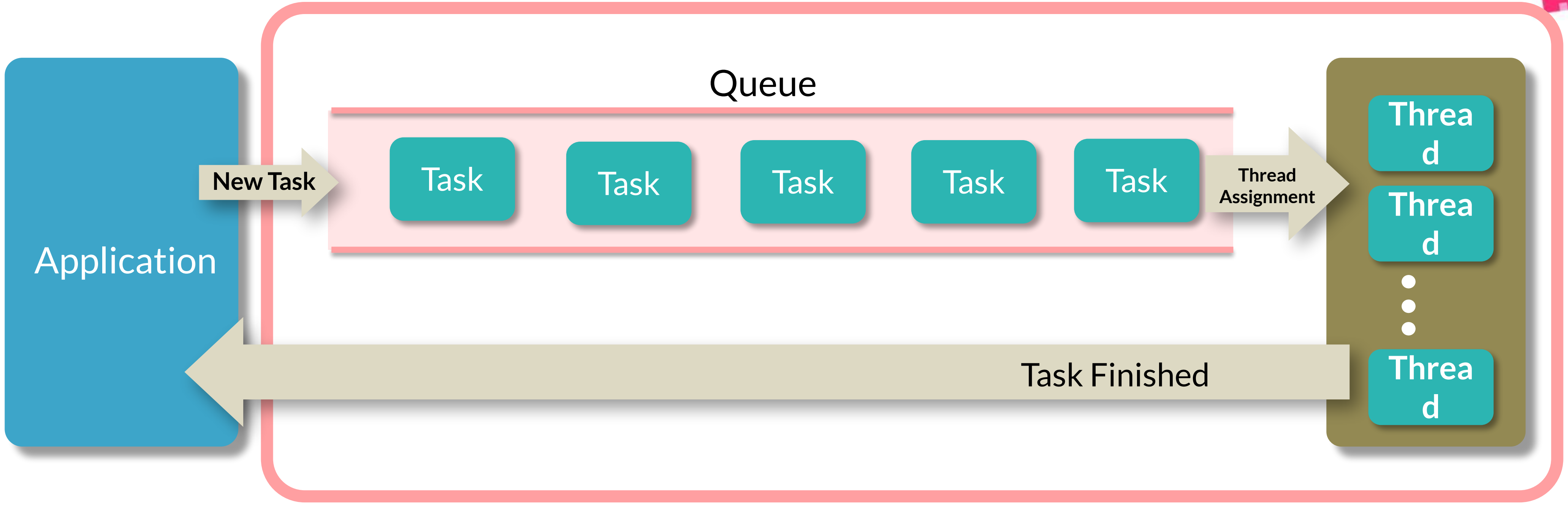
Java Concurrent Package


java.util.concurrent
C Executors
newCachedThreadPool():ExecutorService
newFixedThreadPool(nThreads:int):ExecutorService
newSingleThreadExecutor():ExecutorService
newWorkStealingPool():ExecutorService
I ExecutorService
<T> invokeAll(tasks: Collection<? extends Callable<T>>): List<Future<T>>
<T> invokeAny(tasks: Collection<? extends Callable<T>>): T
isShutdown(): boolean
isTerminated: boolean
shutdown()
<T> submit(task: Callable<T>): Future <T>
I Future
<T> get(): T
cancel(mayInterruptRunning: boolean): boolean
isCancelled(): boolean
I Callable
<T> call(): T

Thread Pool

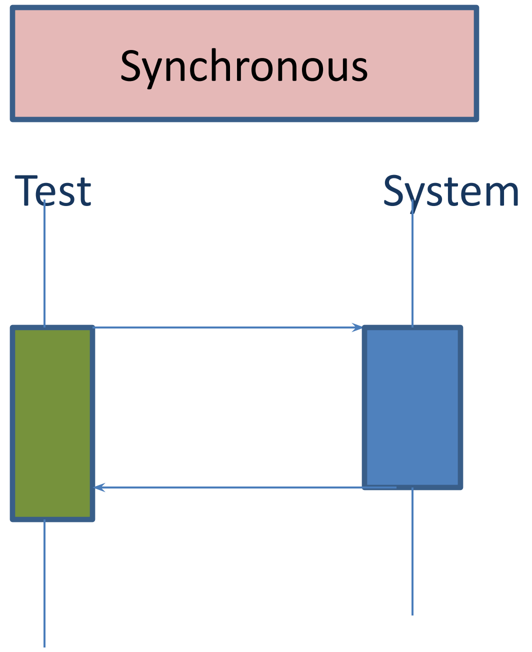
Thread Pool

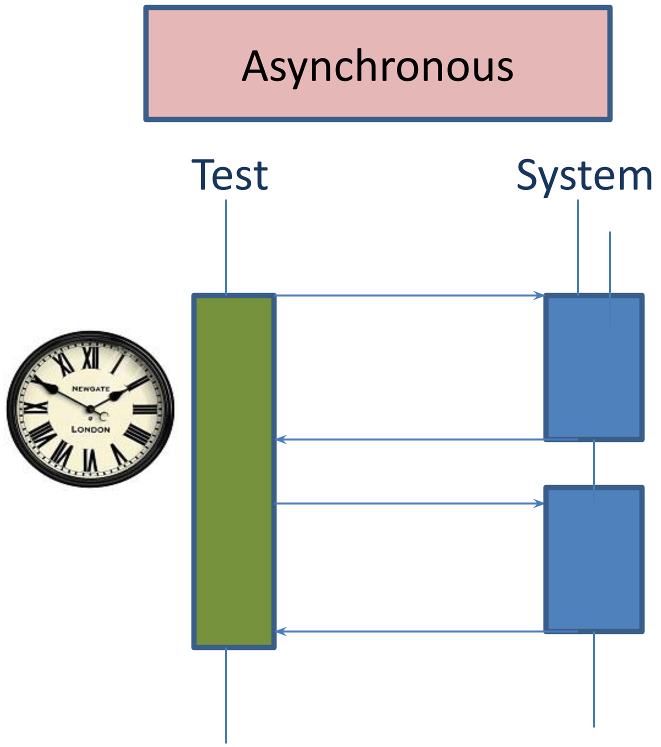
Java thread pool manages the pool of worker threads and contains a queue that keep the tasks waiting to get executed



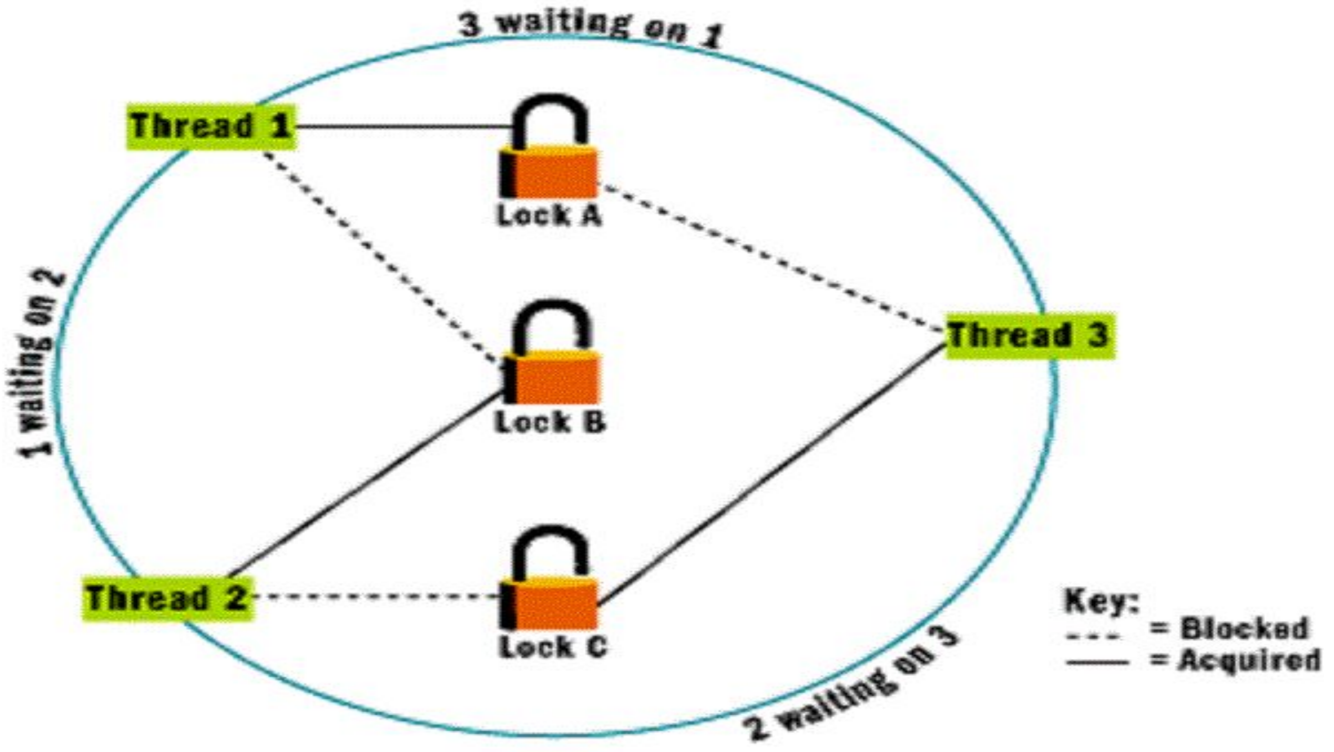
Synchronization

Synchronization is the coordination of events to operate a system in unison. Synchronization is achieved by the use of locks in Java.





Locks in Java



A lock is a tool for controlling access to a shared resource by multiple threads.

There are four different kinds of locks:

Fat locks: A fat lock is a lock with a history of contention (several threads trying to take the lock simultaneously) , or a lock that has been waited on (for notification).

Thin locks : A thin lock is a lock that does not have any Contention.

Recursive locks : A recursive lock is a lock that has been taken by a thread several times without having been released.

lazy locks: A lazy lock is a lock that is not released when a critical section is exited. Once a lazy lock is acquired by a thread, other threads that try to acquire the lock have
to ensure that the lock is , or can be , released.

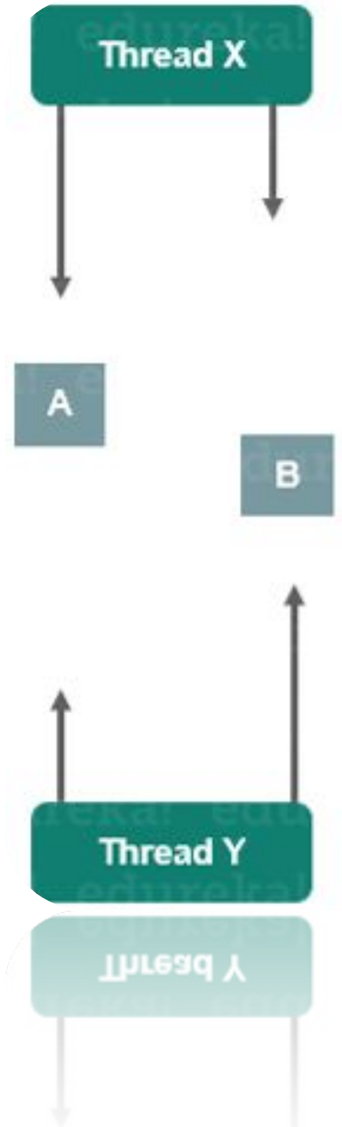
Deadlock

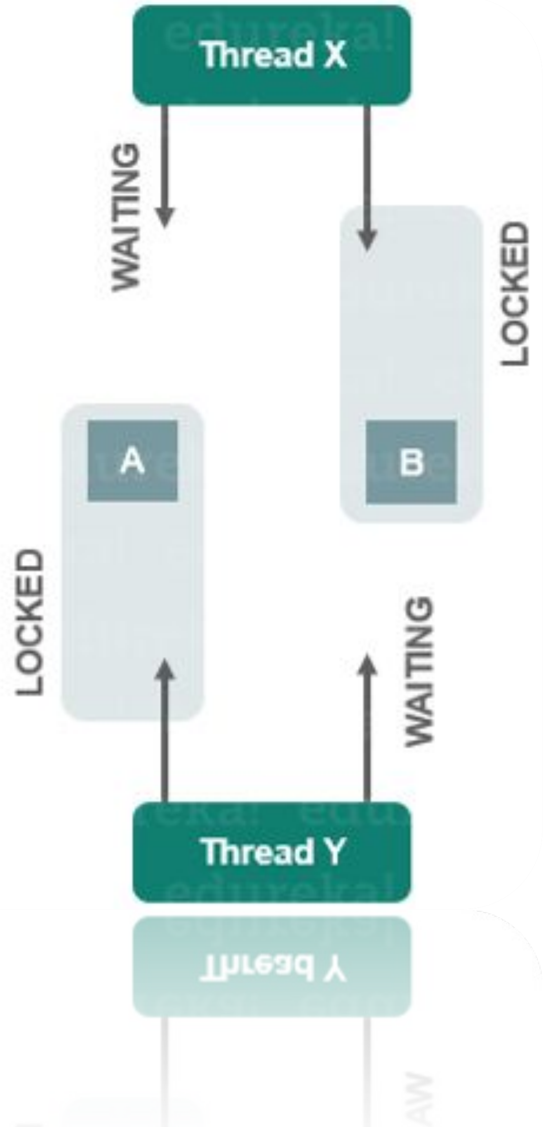
- Deadlock describes a situation where two or more threads are blocked forever, waiting for each other.

or

- A deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

Deadlock


Thread X
A
B
Thread Y


Thread X
WAITING
LOCKED
A
B
LOCKED
WAITING
Thread Y

Deadlock - Example

```
    Thread 1              Thread 2
       |                     |
       |                     |
  (I got Printer)      (I got Scanner)
       |                     |
       |                     |
 I need Scanner        I need Printer
    (Blocked)             (Blocked)
```

Thank You!