Multithreading and Generic Classes

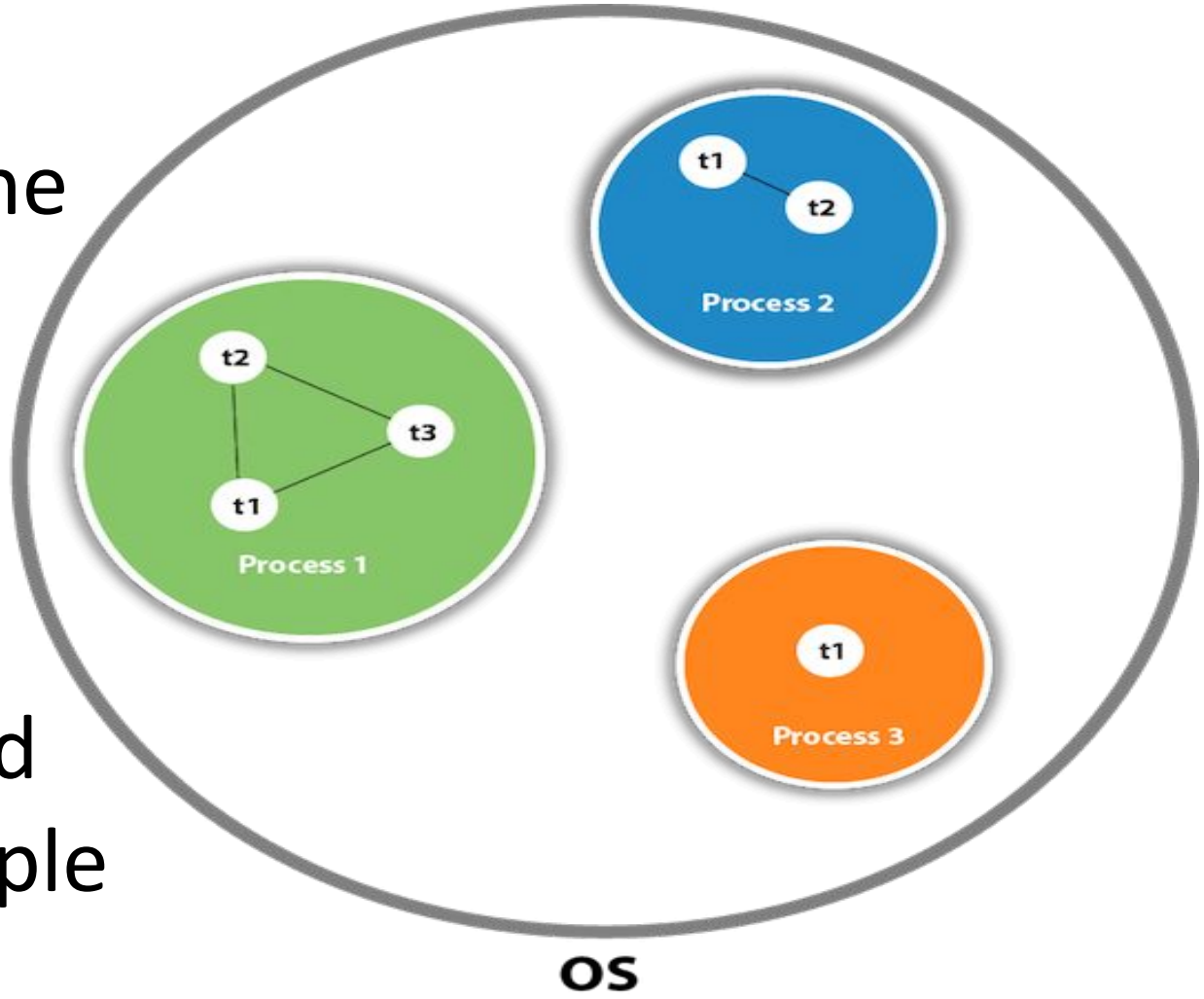
Unit IV

- Java is a multi-threaded programming language which means we can develop multi-threaded program using Java.
- A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.
- A thread is a lightweight sub-process, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.

- Threads use a shared memory area.
- They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- Threads are independent.
- If there occurs exception in one thread, it doesn't affect other threads.
- It uses a shared memory area.

How it looks with respect to Operating System

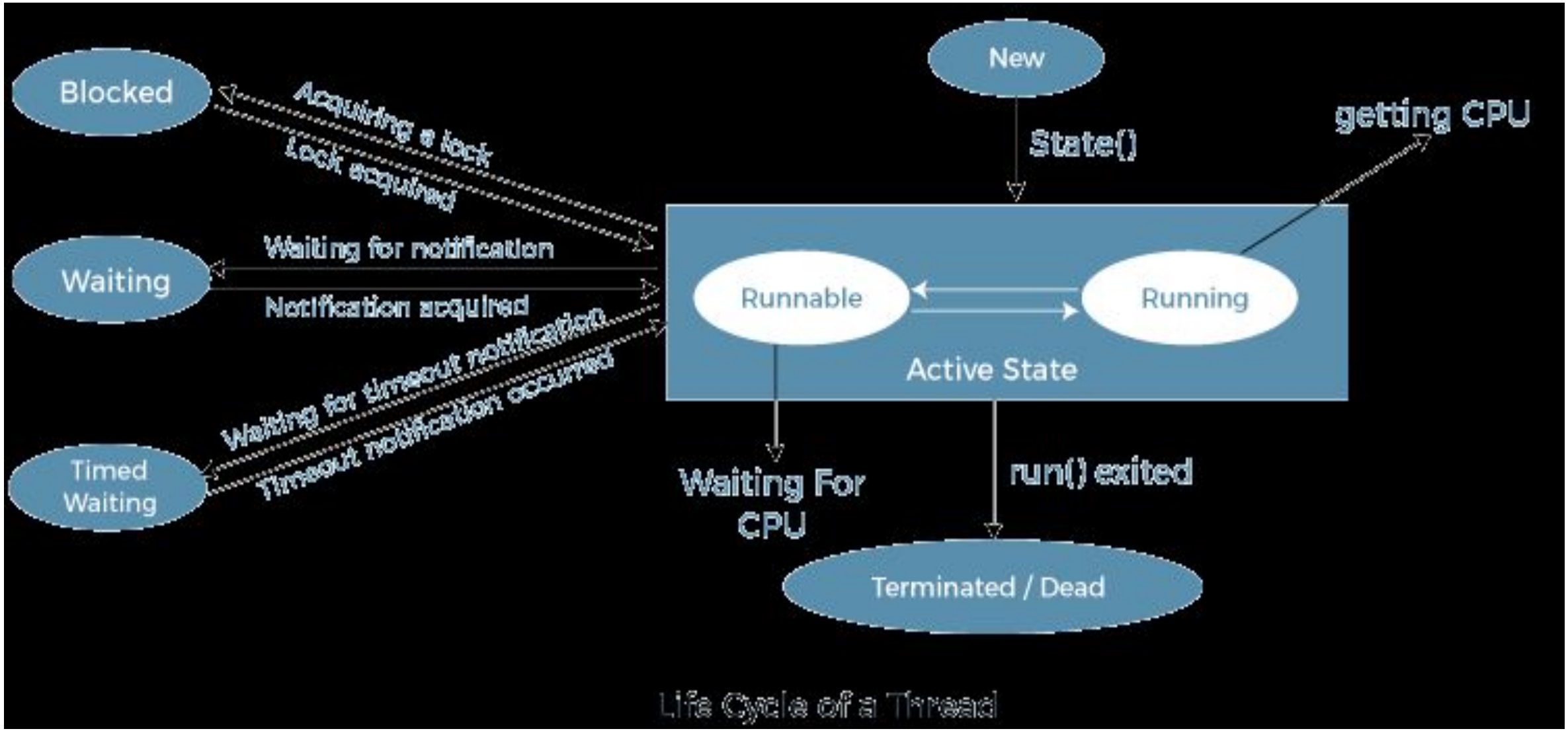
- Thread is executed inside the process.
- There is context-switching between the threads.
- There can be multiple processes inside the OS, and one process can have multiple threads.



Java Thread class

- Java provides **Thread class** to achieve thread programming.
- Thread class provides constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.
- Threads can be created by using two mechanisms :
  1. Extending the Thread class
  2. Implementing the Runnable Interface

Life cycle of a Thread



- In Java, a thread always exists in any one of the following states. These states are:
  1. New
  2. Active
  3. Blocked / Waiting
  4. Timed Waiting
  5. Terminated

- **New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.
- **Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.
  - **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.
  - **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.
- **Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

- **Timed Waiting:** Sometimes, waiting for leads to starvation.
- For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section.
- In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation.
- To avoid such scenario, a timed waiting state is given to thread B.
- Thus, thread lies in the waiting state for a specific span of time, and not forever.
- A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier
- **Terminated:** A thread reaches the termination state because of the following reasons:
  - When a thread has finished its job, then it exists or terminates normally.
  - **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

How to create a thread in Java

- **Thread class:**
  - Thread class provide constructors and methods to create and perform operations on a thread.
  - Thread class extends Object class and implements Runnable interface.
- **Commonly used Constructors of Thread class:**
  - Thread()
  - Thread(String name)
  - Thread(Runnable r)
  - Thread(Runnable r,String name)

- **Runnable interface:**
  - The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
  - Runnable interface have only one method named run().
  - **public void run():** is used to perform action for a thread.

ThreadGroup in Java

- Java provides a convenient way to group multiple threads in a single object.
- In such a way, we can suspend, resume or interrupt a group of threads by a single method call.
- A ThreadGroup represents a set of threads.
- A thread group can also include the other thread group.
- The thread group creates a tree in which every thread group except the initial thread group has a parent.
- A thread is allowed to access information about its own thread group, but it cannot access the information about its thread group's parent thread group or any other thread groups.

Constructors of ThreadGroup class

No.	Constructor	Description
1)	ThreadGroup(String name)	creates a thread group with given name.
2)	ThreadGroup(ThreadGroup parent, String name)	creates a thread group with a given parent group and name.

```
1.ThreadGroup tg1 = new ThreadGroup("Group A");
2.Thread t1 = new Thread(tg1,new MyRunnable(),"one");
3.Thread t2 = new Thread(tg1,new MyRunnable(),"two");
4.Thread t3 = new Thread(tg1,new MyRunnable(),"three");
```

Synchronization in Java

- Synchronization in Java is the capability to control the access of multiple threads to any shared resource.
- The synchronization is mainly used to
  1. To prevent thread interference.
  2. To prevent consistency problem.

Thread Synchronization

- There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

- Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:
  1. By Using Synchronized Method
  2. By Using Synchronized Block
  3. By Using Static Synchronization