# Java Threads

Eng. Aya El-Sayed

# What is a Thread?

A Java thread is the smallest unit of execution in a program. It is a lightweight subprocess that runs independently but shares the same memory space with other threads, enabling concurrent execution of multiple tasks.

Every Java program starts with a main thread, and additional threads can be created to handle background or parallel tasks without interrupting the main program's flow.

# ⚙ Why Use Threads?

**Threads allow a program to:**

- Run multiple tasks at the same time (concurrently).
- Keep the application responsive (e.g., in GUIs or servers).
- Use multi-core processors efficiently.

**Example:**

Imagine downloading a file while showing a loading animation — one thread can handle downloading, another can update the UI.

# Life Cycle of Threads

**A thread goes through these states:**

1. New State
2. Runnable State
3. Waiting/Blocked State
4. Timed Waiting State
5. Terminated State

| State | Description |
| --- | --- |
| **New** | Created but not yet started (new Thread()) |
| **Runnable** | Ready to run or running (start()) |
| **Blocked/Waiting** | Waiting for a resource or another thread |
| **Timed Waiting** | Sleeping for a specific time (sleep()) |
| **Terminated** | Finished running |

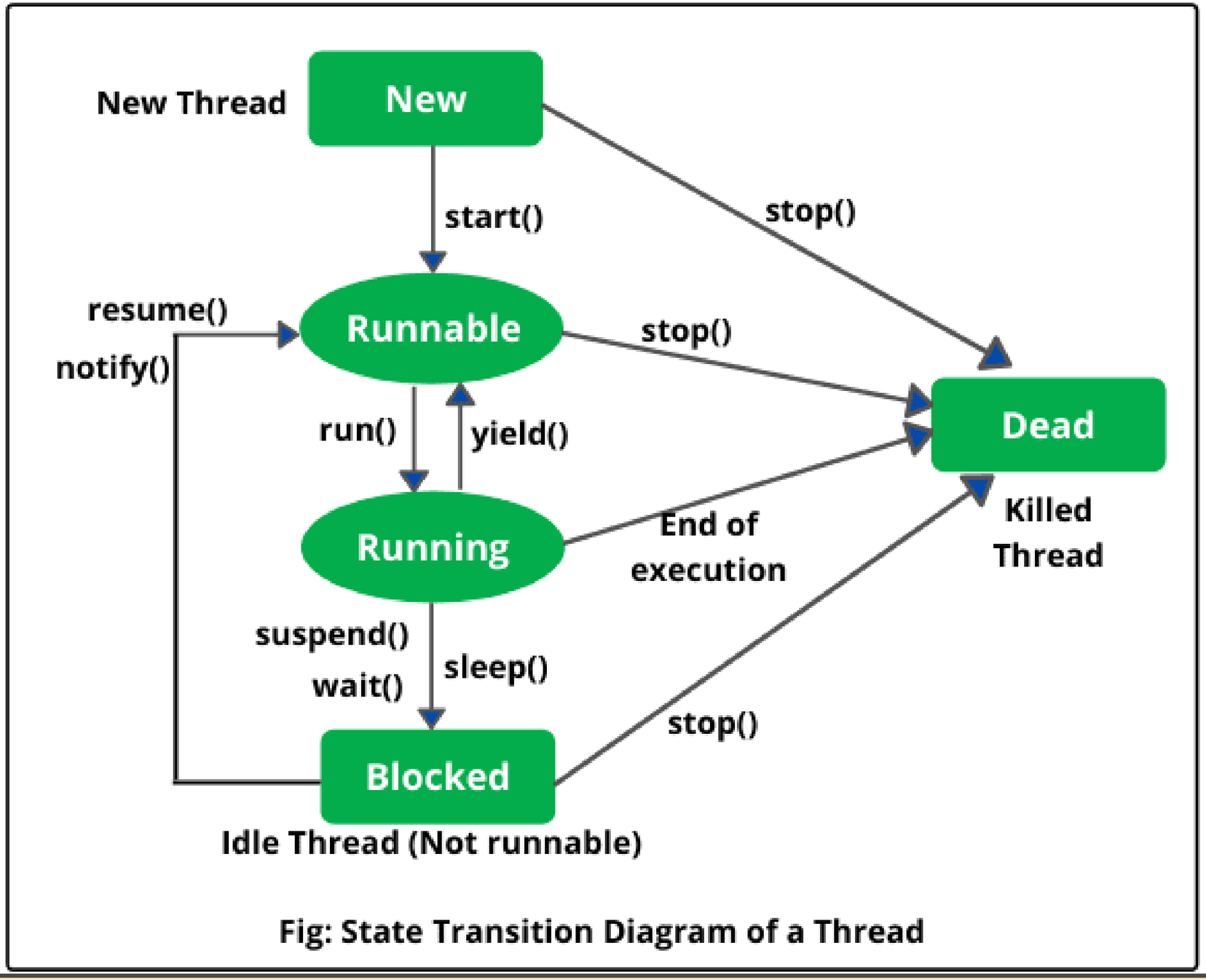
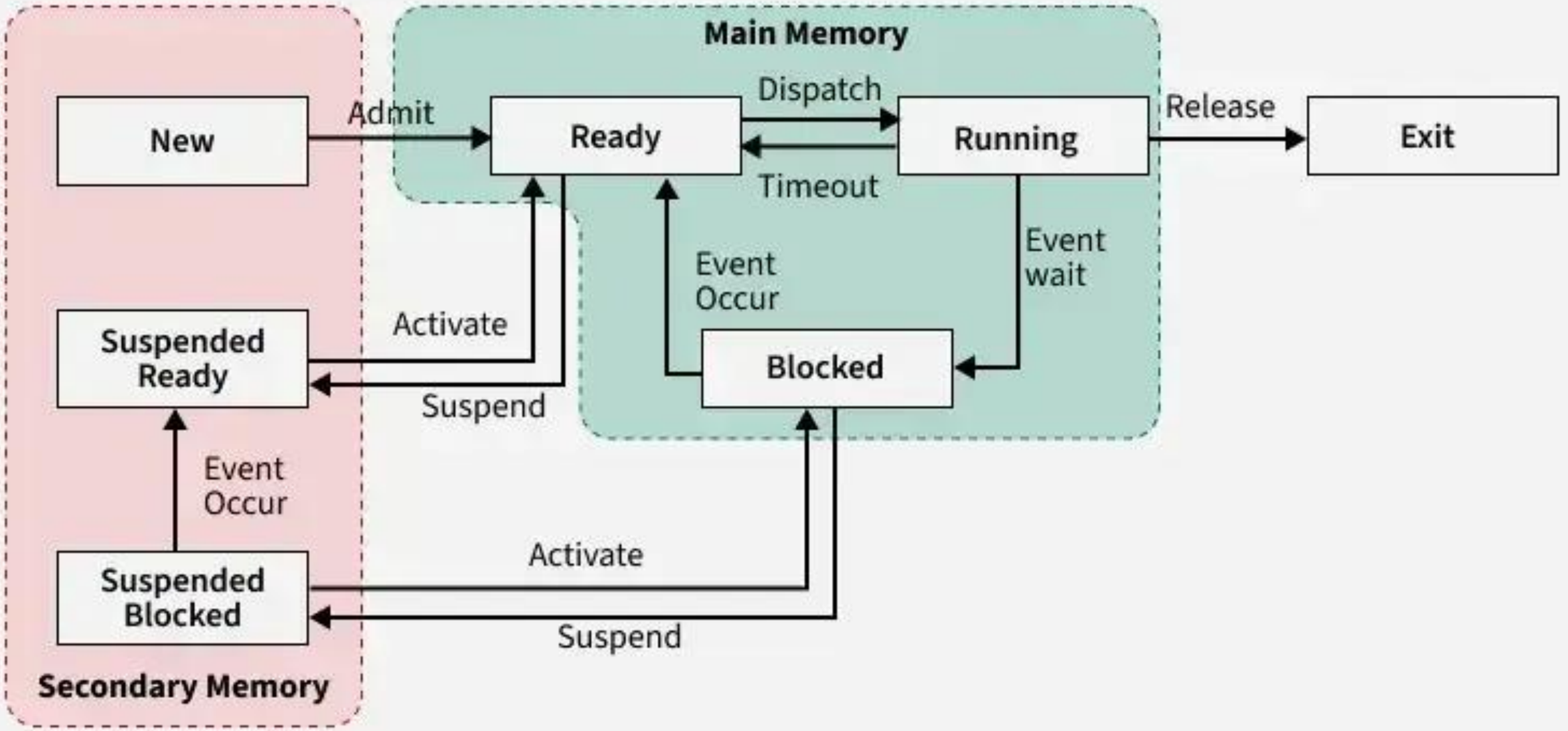


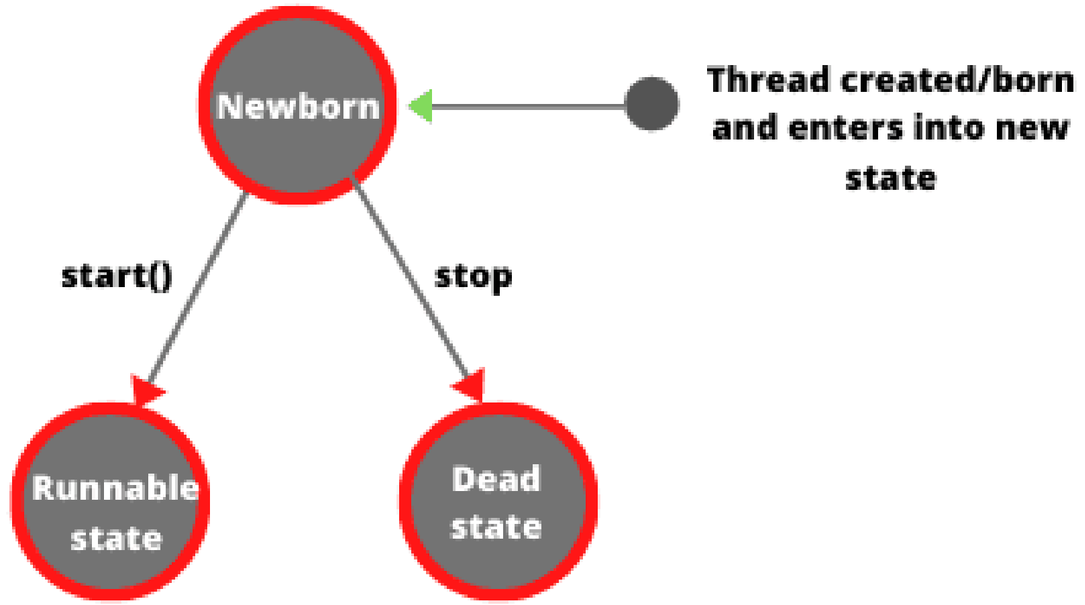

Fig: State Transition Diagram of a Thread


Main Memory
New
Admit
Ready
Dispatch
Running
Release
Exit
Timeout
Event
wait
Event
Occur
Activate
Suspended
Ready
Blocked
Suspend
Event
Occur
Activate
Suspended
Blocked
Suspend
Secondary Memory

# New State

When we create a thread object using the Thread class, the thread is born and enters the New state. This means the thread is created, but the start() method has not yet been called on the instance.

The thread remains in this state until you start its execution by calling the **start()** method.



# Creating Threads in Java

```
class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("Thread is running...");
    }

    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

# Creating Threads in Java

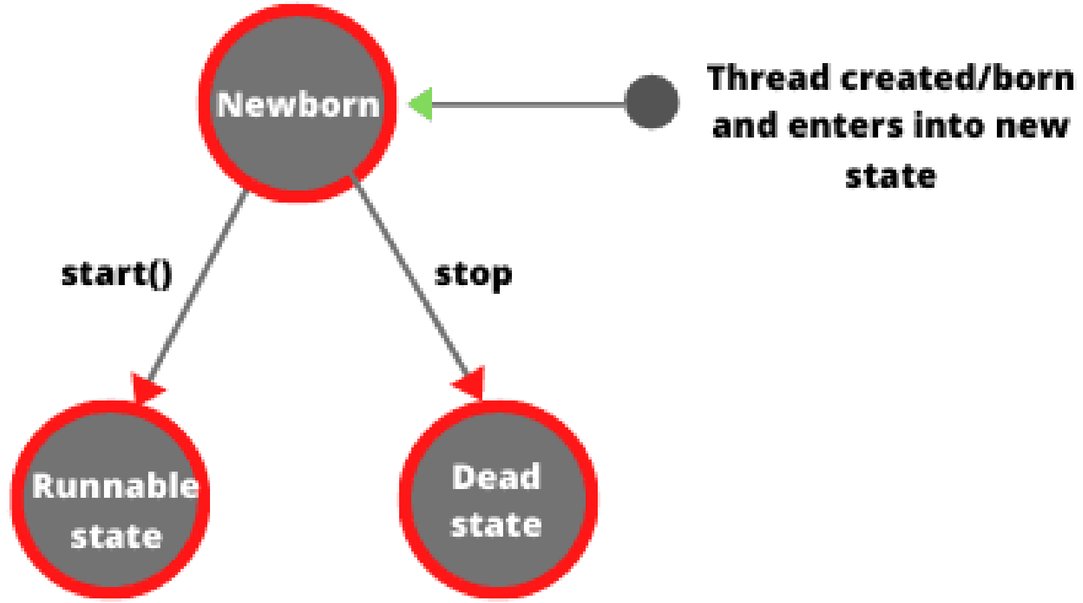
```
class MyRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("Runnable thread running...");
    }
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

**Preferred when your class already extends another class**

# Runnable State

Runnable state means a thread is ready for execution of any statement. When the start() method is called on a new thread, thread enters into from New to a Runnable state.

If all threads have equal priority, CPU allocates time slots for thread execution on the basis of first-come, first-serve manner.



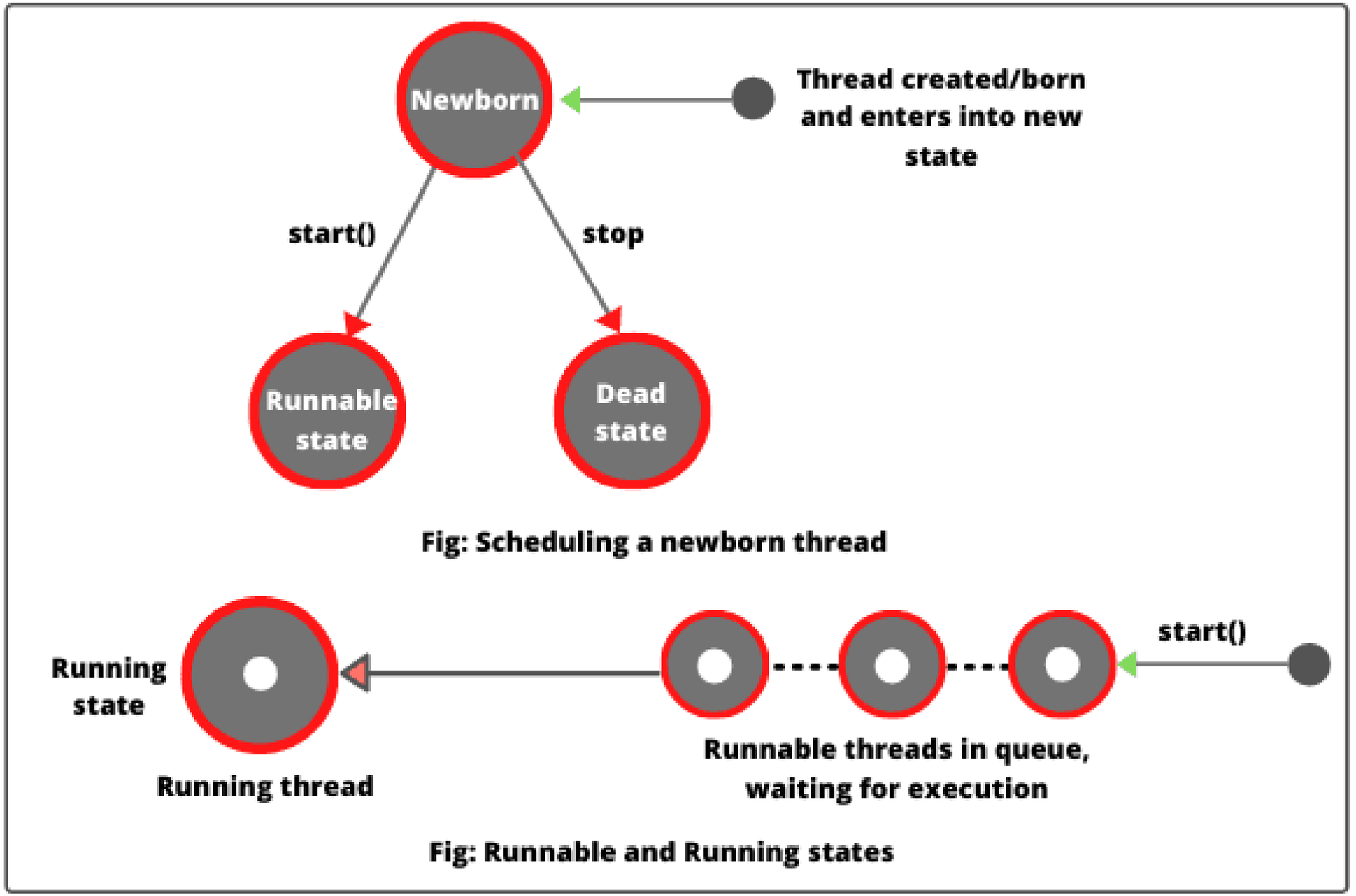



Fig: Scheduling a newborn thread




Fig: Runnable and Running states

# Running State

Running means Processor (CPU) has allocated time slot to thread for its execution.

When thread scheduler selects a thread from the runnable state for execution, it goes into running state.

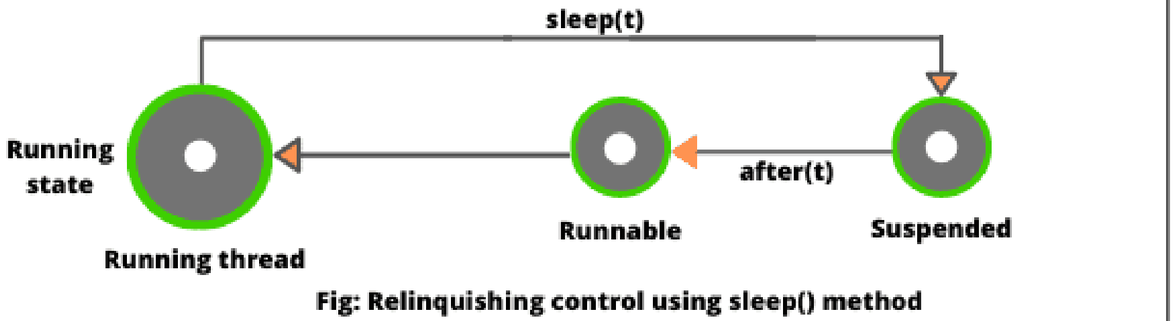



Fig: Relinquishing control using sleep() method

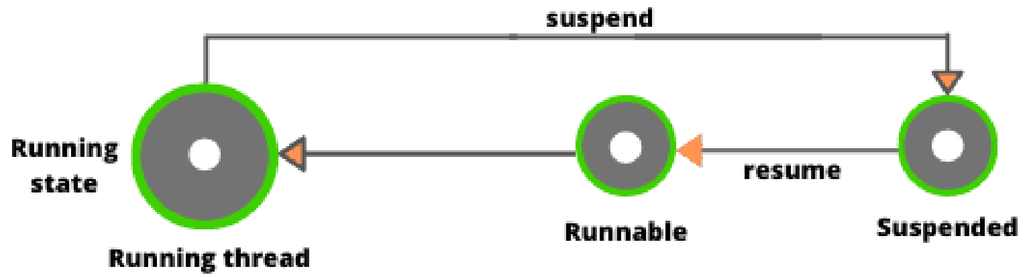



Fig: Relinquishing control using suspend() method

```
public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("Thread is in RUNNING state, executing run() method.");
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        // Create a new thread. This thread is in NEW state.
         MyThread thread = new MyThread();
         Thread th = new Thread(thread);

         // Check thread state (NEW)
        System.out.println("Thread state before start: " + thread.getState());

         // Start the thread by calling start() method. Now the thread moves from New to Runnable state.
        thread.start();

        // Check the current thread state after calling start.
        System.out.println("Thread state after start: " + thread.getState());
    }
}
```

**Output:**

Thread state before start: NEW

Thread state after start: RUNNABLE

Thread is in RUNNING state, executing run() method.

# Blocked State

A thread is considered to be in the blocked state when it is **suspended**, **sleeping**, or **waiting** for a specified amount of time to satisfy a certain condition.

For example, a thread enters the Blocked state when it is waiting to acquire a lock or a monitor that is held by another thread. This typically occurs when multiple threads attempt to access a synchronized block or method.

# Dead or Terminated State

A thread dies or moves into a dead state automatically when its run() method completes the execution of statements.

That is, a thread is terminated or dead when a thread comes out of run() method.

A thread can also be dead when the stop() method is called. Once a thread is in the DEAD state, it cannot be started again.

| Method | Description |
| --- | --- |
| **start()** | Starts the execution of a thread by calling its run() method in a new thread. |
| **run()** | Contains the code that defines what the thread will do when it runs. |
| **sleep(long ms)** | Makes the current thread pause for a specified number of milliseconds. |
| **join()** | Waits for another thread to finish before continuing. |
| **isAlive()** | Returns true if the thread is still running, otherwise false. |
| **yield()** | Pauses the current thread to allow other threads of equal priority to execute. |
| **interrupt()** | Interrupts a sleeping or waiting thread. |
| **currentThread()** | Returns a reference to the currently executing thread. |
| **getName()/setName(String name)** | Gets or sets the name of the thread. |
| **getPriority()/setPriority(int priority)** | Gets or sets the thread’s priority (1 to 10). |

# Thread Synchronization

When multiple threads share data (e.g., updating the same variable), we need synchronization to avoid conflicts.

```
synchronized void increment() {
    count++;
}
```

synchronized ensures only one thread runs this method at a time.

# Real-Life Examples

- Downloading files while showing progress
- Bank transactions (one thread deposits, another withdraws)
- Games (graphics, input, and sound handled by separate threads)
- Web servers (each client connection handled by a new thread)

# THANK YOU