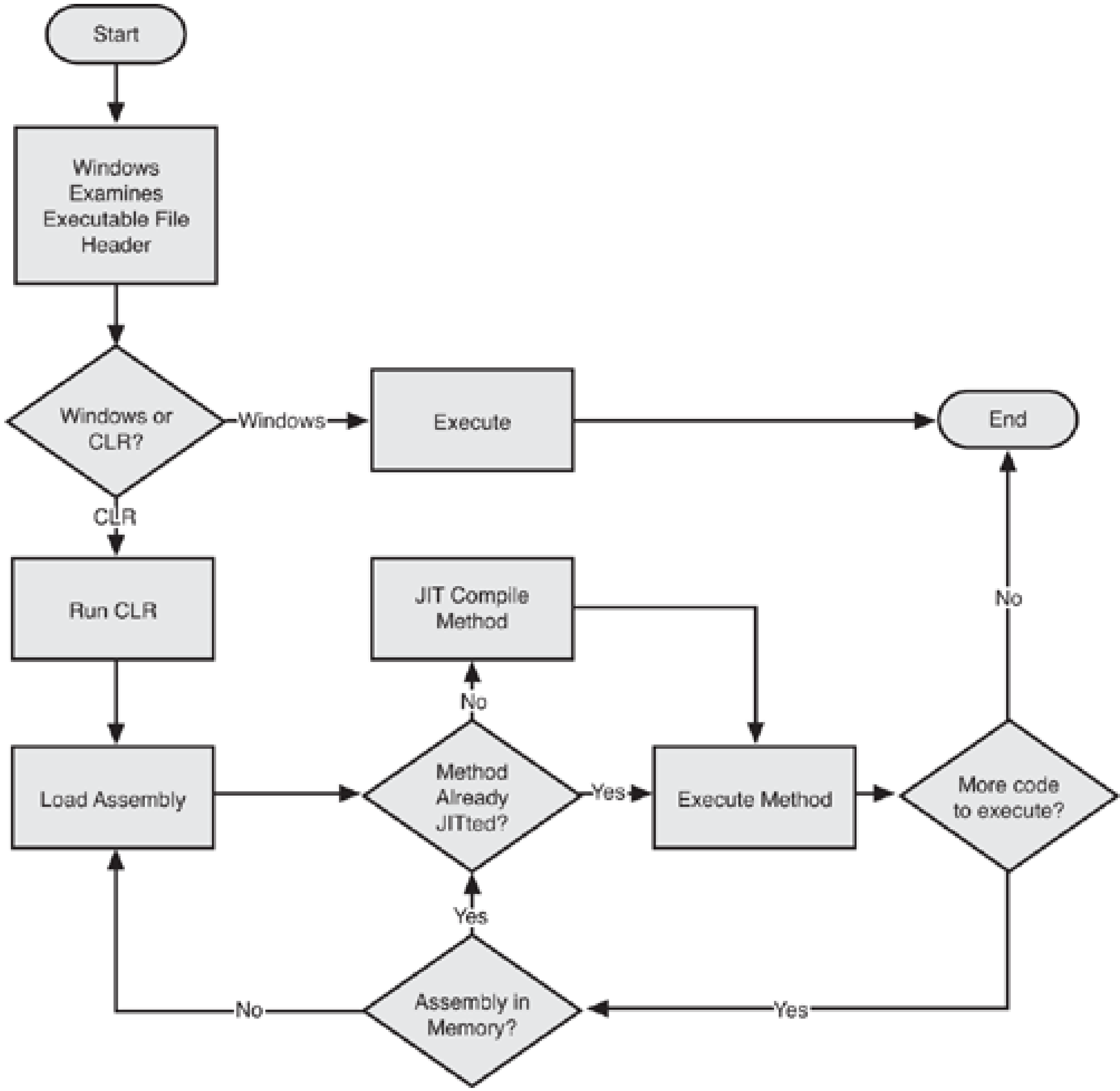
Day 5 : .NET & C# Basics

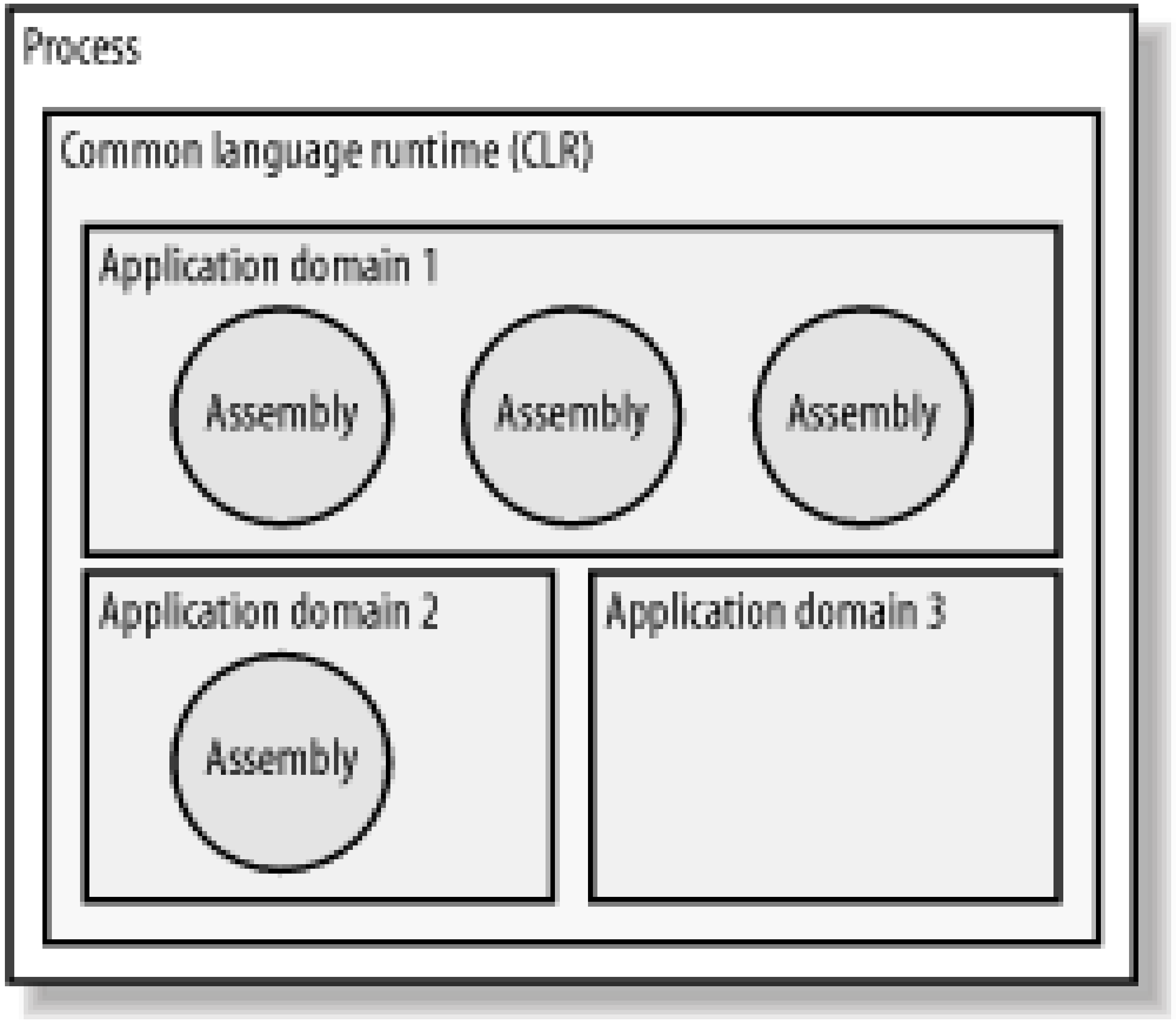
Application Execution Flow



Process & App Domains

Process is an OS concept

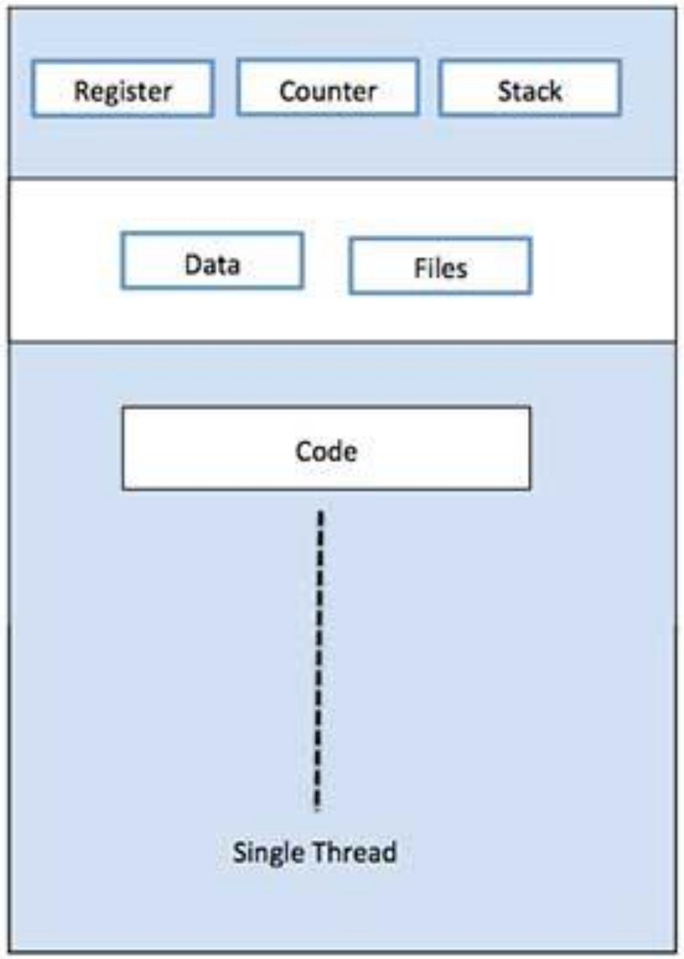
App Domains is a .NET Concept



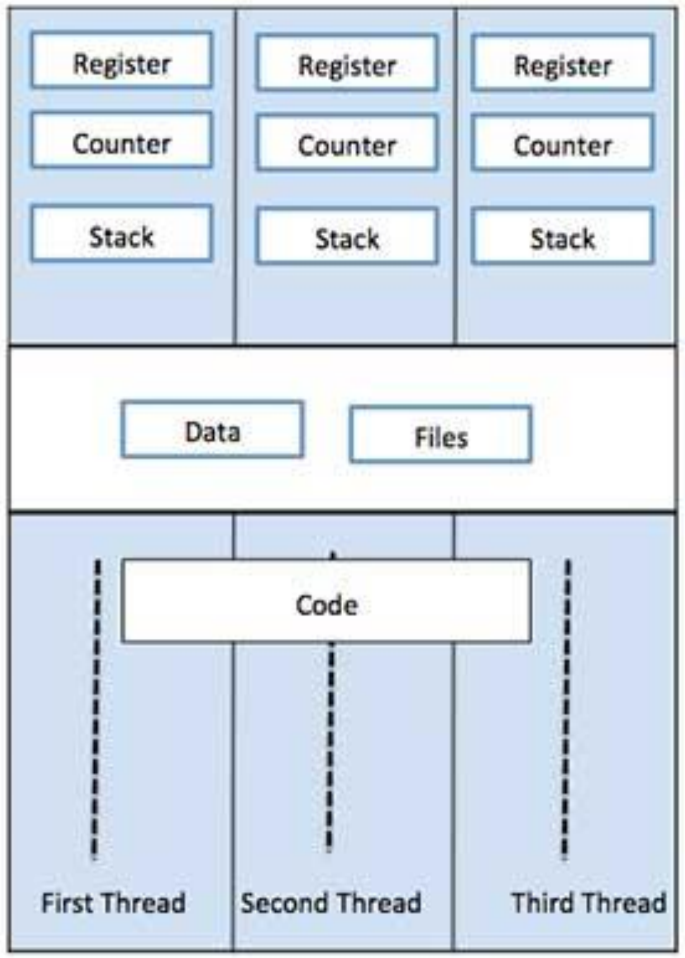
Threads in C#

- Basic unit of execution
- Is an independent set of instructions in a program
- A C# program starts in a single thread which is created automatically by CLR
- Namespace is System.Threading
- Two Types
  - Foreground
  - Background

Process vs Threads

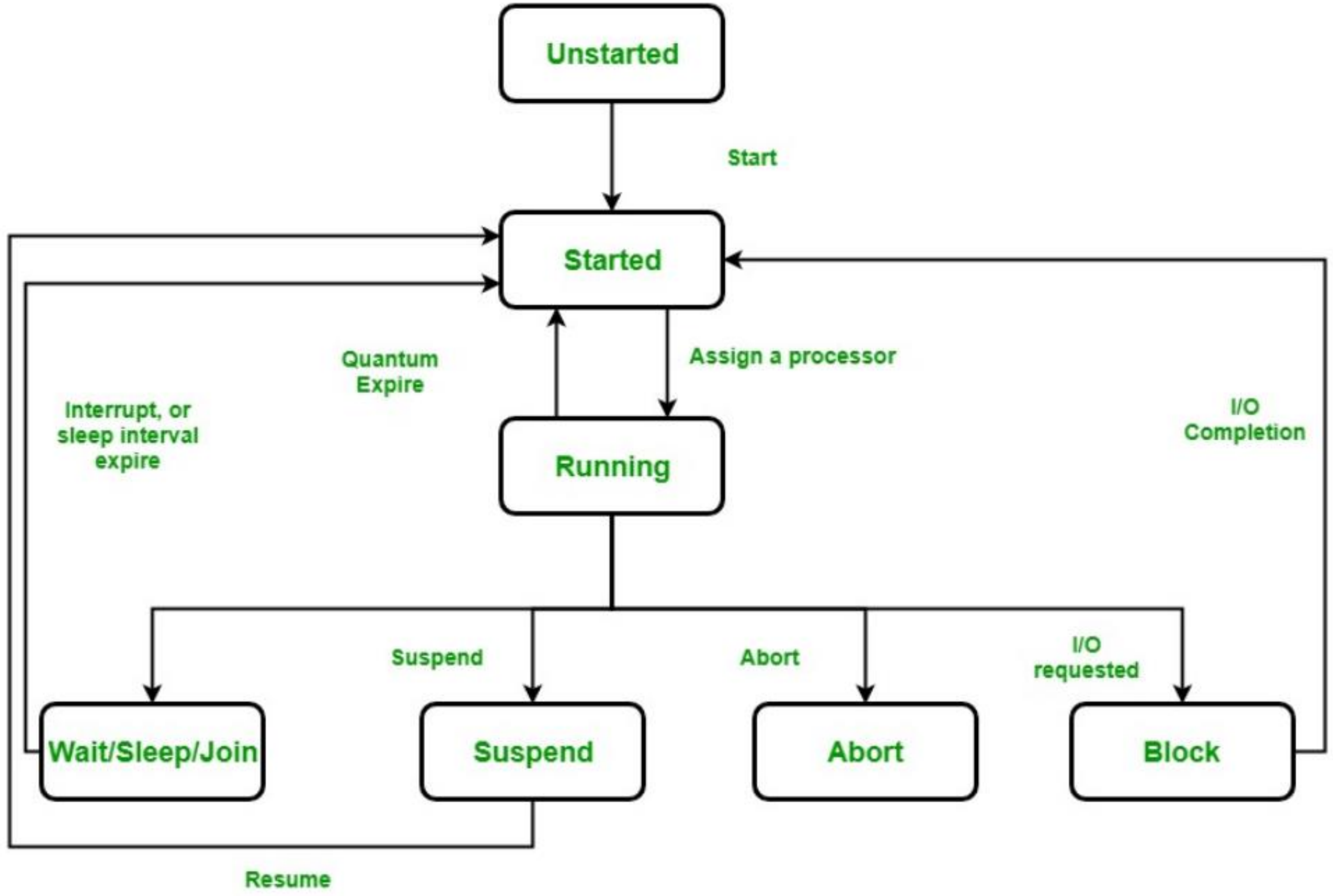


Single Process P with single thread



Single Process P with three threads

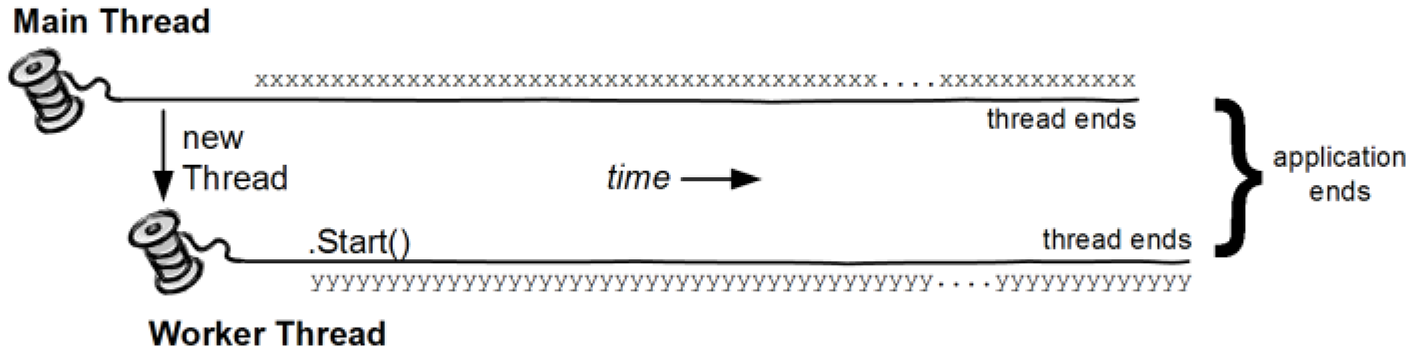
Lifecycle


Unstarted
Start
Started
Quantum Expire
Assign a processor
Interrupt, or sleep interval expire
I/O Completion
Running
Suspend
Abort
I/O requested
Wait/Sleep/Join
Suspend
Abort
Block
Resume

Example

```
static void Main(String[] args)
{
  //creates a new thread and executes the method WriteUsingNewThread
  //Thread t = new Thread(WriteUsingNewThread);
  //t.Start();
}

static void WriteUsingNewThread()
{
    Console.WriteLine(“Running in a new thread”)
    for (int i = 1000; i >1; i--)
       Console.WriteLine($" Number is {i}");
}
```


Main Thread
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx....xxxxxxxxxxxxx
thread ends
new Thread
time
application ends
.Start()
thread ends
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy....yyyyyyyyyyyyy
Worker Thread

Thread Types - Foreground

- Are those threads which keeps on running even after the main thread is finished
- Has the ability to prevent the application from terminating
- CLR won’t shut down the application until all the threads running in the foreground are completed
- Every thread we create is a foreground thread by default

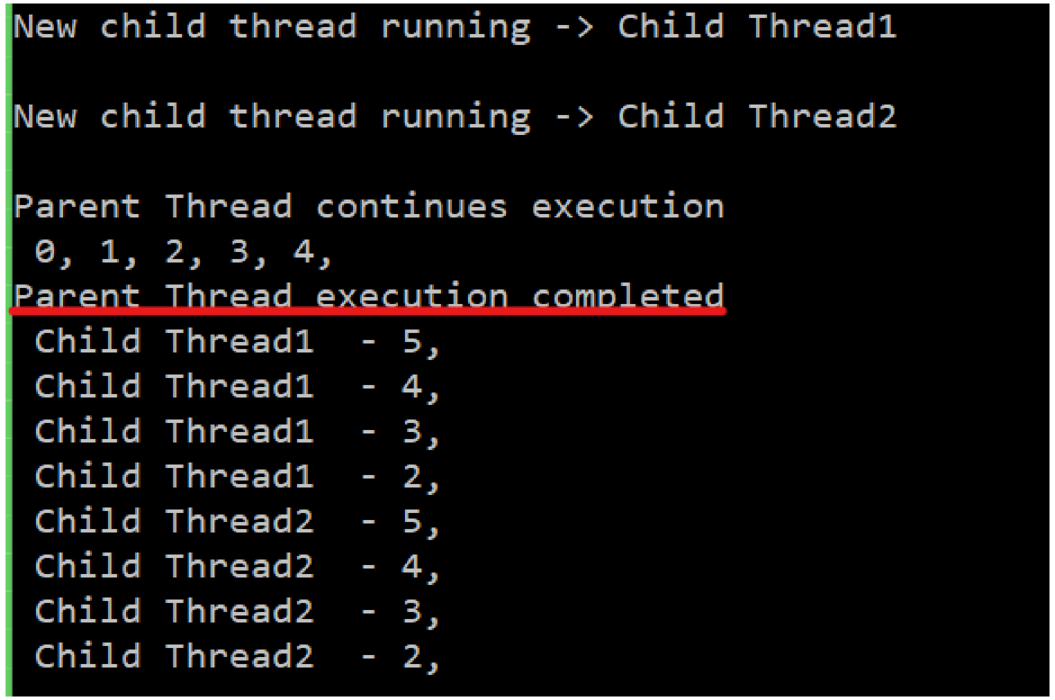
Example

```
static void Main(String[] args) {
    Thread t = new Thread(WriteUsingNewThread);
    t.Name = "Child Thread1";
    t.Start();

    Thread t1 = new Thread(WriteUsingNewThread);
    t1.Name = "Child Thread2";
    t1.Start();

    Console.WriteLine("\nParent Thread continues execution ");
    for (int i = 0; i < 5; i++)
        Console.Write($" {i},");
    Console.WriteLine("\nParent Thread execution completed");
}

static void WriteUsingNewThread() {
    Console.WriteLine(“Running in a new thread”)
    for (int i = 1000; i > 1; i--)
        Console.WriteLine($" Number is {i}");
}
```



```
New child thread running -> Child Thread1

New child thread running -> Child Thread2

Parent Thread continues execution
 0, 1, 2, 3, 4,
Parent Thread execution completed
 Child Thread1  - 5,
 Child Thread1  - 4,
 Child Thread1  - 3,
 Child Thread1  - 2,
 Child Thread2  - 5,
 Child Thread2  - 4,
 Child Thread2  - 3,
 Child Thread2  - 2,
```

Thread Types - Background

- Is also known as Daemon threads
- All the background threads will be terminated when the main application quits
- Set `IsBackground` property to true if you want the thread to run in the background.

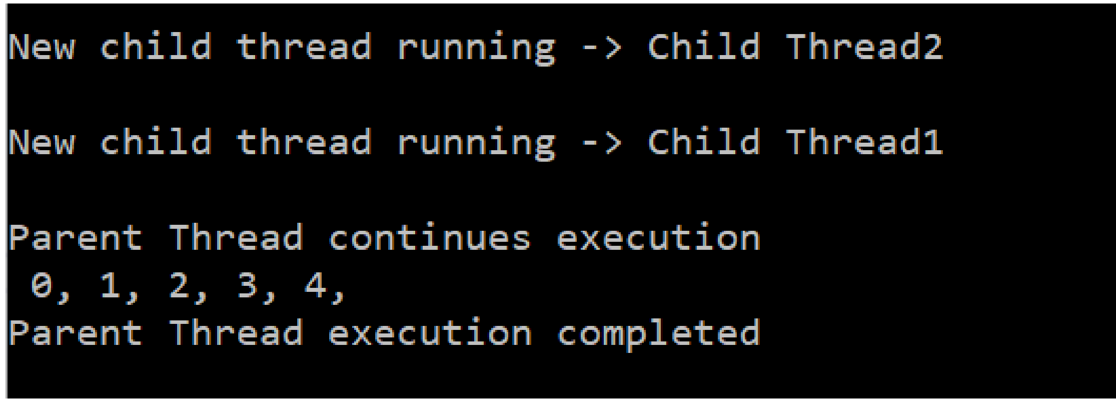
Example

```
static void Main(String[] args) {
    Thread t = new Thread(WriteUsingNewThread);
    t.Name = "Child Thread1";
    t.IsBackground = true;
    t.Start();

    Thread t1 = new Thread(WriteUsingNewThread);
    t1.Name = "Child Thread2";
    t1.IsBackground = true;
    t1.Start();

    Console.WriteLine("\nParent Thread continues execution ");
    for (int i = 0; i < 5; i++)
        Console.Write($" {i},");
    Console.WriteLine("\nParent Thread execution completed");
}

static void WriteUsingNewThread() {
    Console.WriteLine(“Running in a new thread”)
    for (int i = 1000; i > 1; i--)
        Console.WriteLine($" Number is {i}");
}
```



```
New child thread running -> Child Thread2

New child thread running -> Child Thread1

Parent Thread continues execution
 0, 1, 2, 3, 4,
Parent Thread execution completed
```

Thread Synchronization

- lock keyword is used to acquire a lock for a piece of code
- Make sures that only one thread can access it at a point of time
- Particularly useful in cases where you are writing to a shared memory in a multithreaded application

```
void ThreadSafetyDemo() {
    //obtain an exclusive lock and then manipulate data /when two threads tries to simulataneously access data, one    will have access to data
    //and the other will wait until the one have the lock finishes. This is called thread safe code
    lock(locker) {
        if (!isFinished) {
            isFinished = true;
            Console.WriteLine("Finished executing the method");
        }
    }
}
```