Concurrency

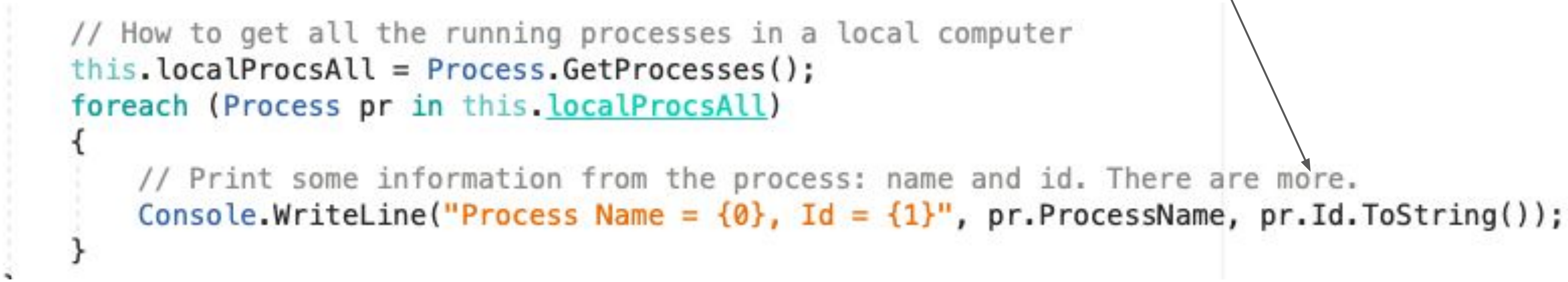
Week 7

Agenda

A Review of Exercises

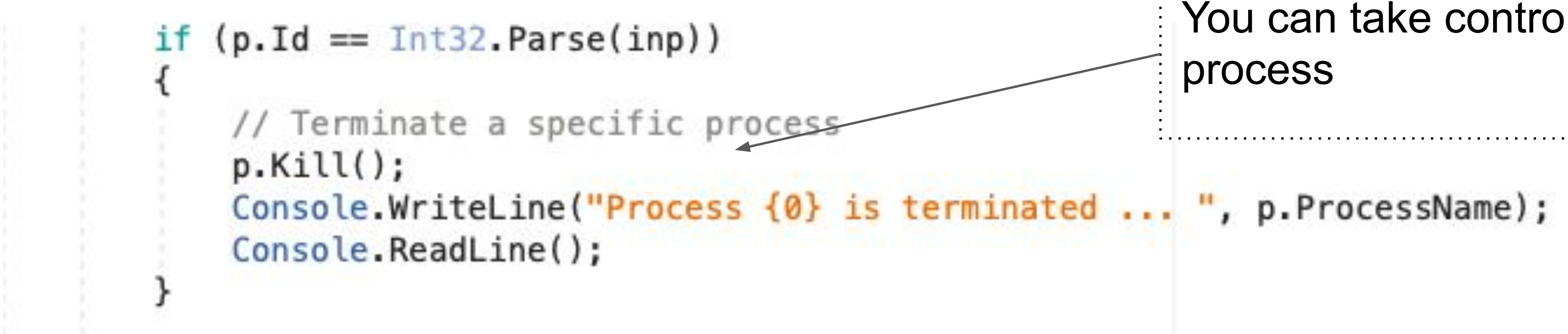
Processes

You have access to all the processes, with a unique PID



```
// How to get all the running processes in a local computer
this.localProcsAll = Process.GetProcesses();
foreach (Process pr in this.localProcsAll)
{
    // Print some information from the process: name and id. There are more.
    Console.WriteLine("Process Name = {0}, Id = {1}", pr.ProcessName, pr.Id.ToString());
}
```

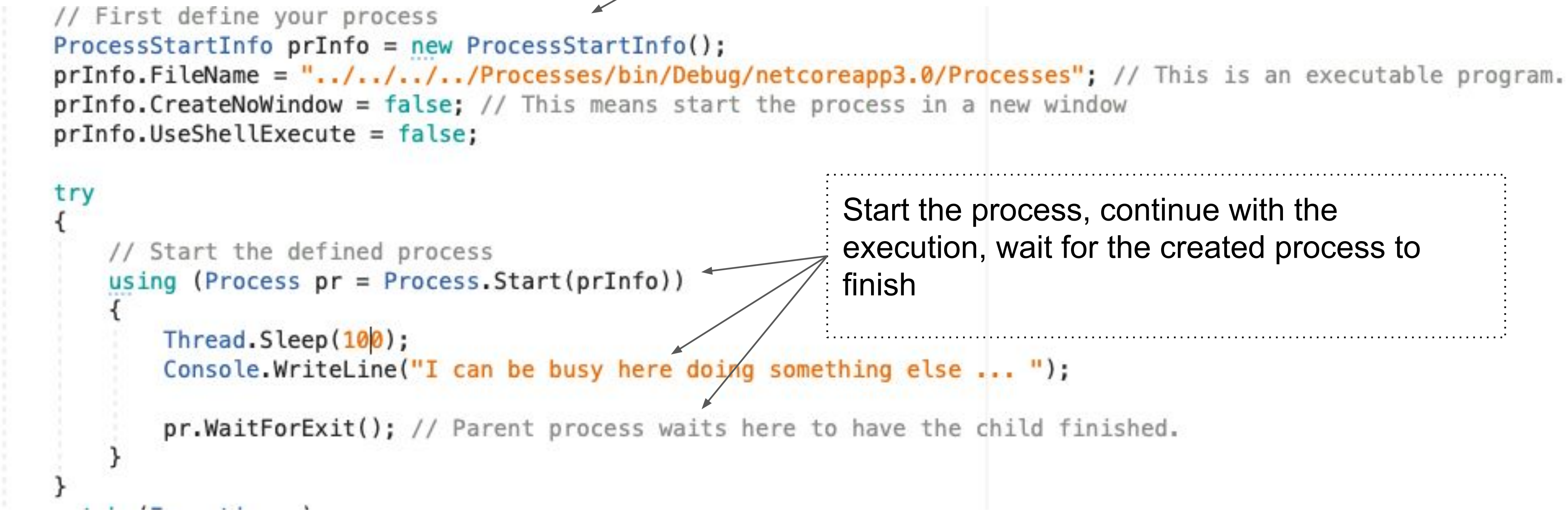
You can take control of an executing process



```
if (p.Id == Int32.Parse(inp))
{
    // Terminate a specific process
    p.Kill();
    Console.WriteLine("Process {0} is terminated ... ", p.ProcessName);
    Console.ReadLine();
}
```

Process Creation

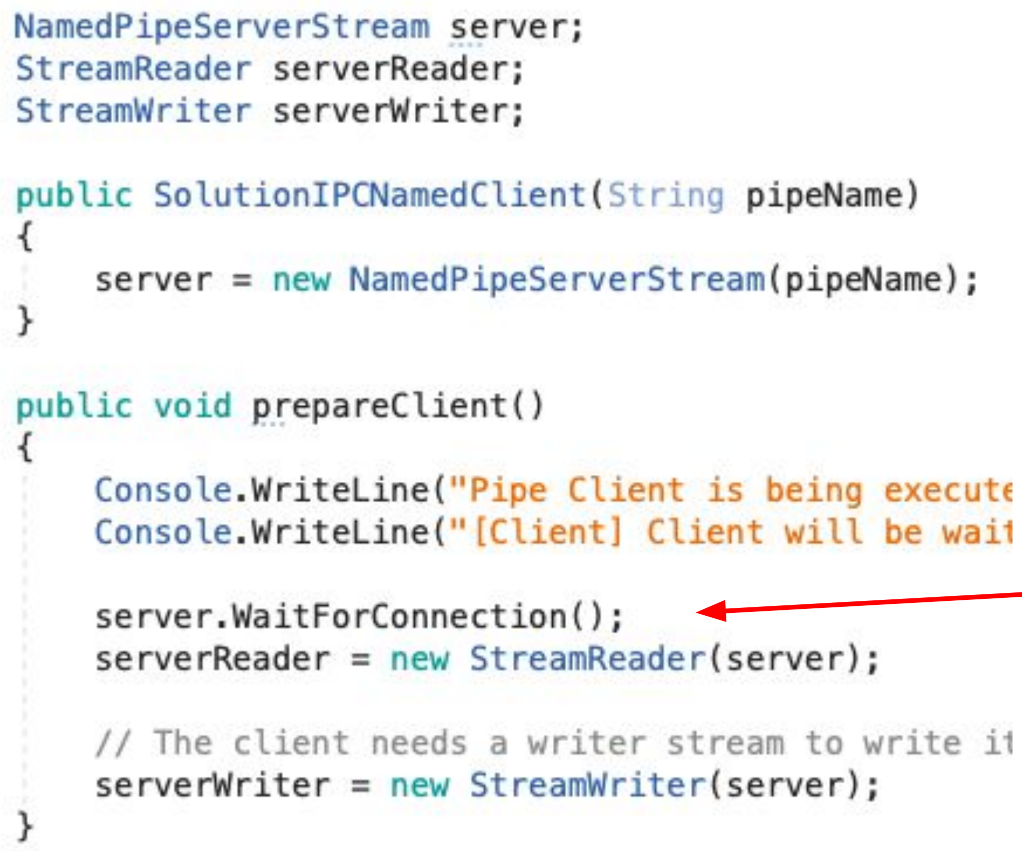
An instance that defines Process information



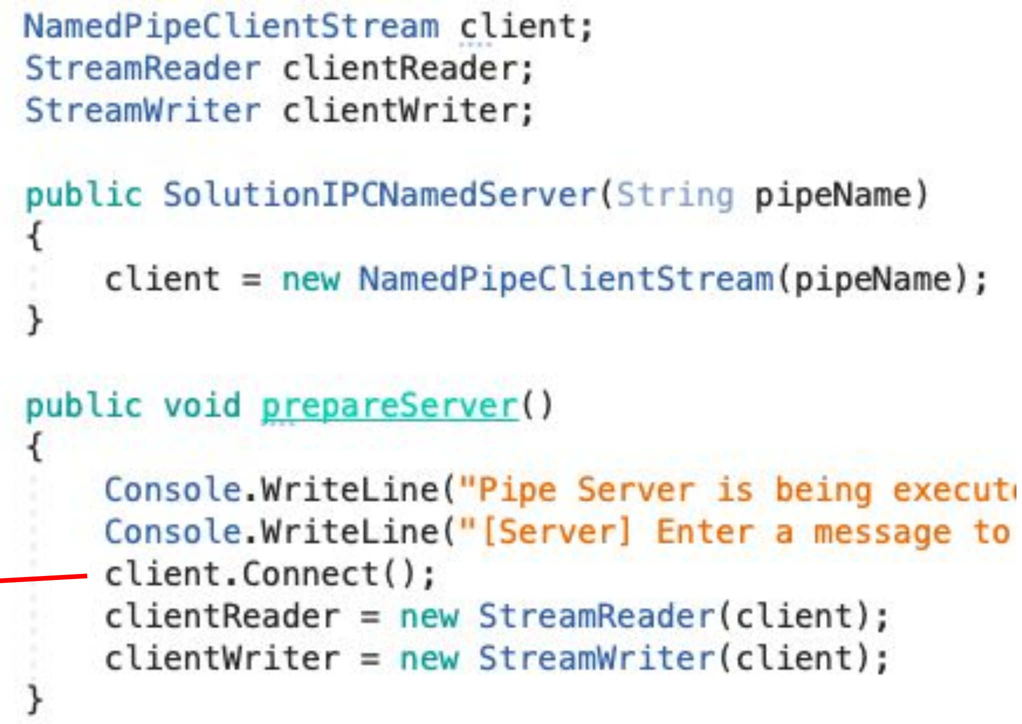
```
// First define your process
ProcessStartInfo prInfo = new ProcessStartInfo();
prInfo.FileName = "../../../../Processes/bin/Debug/netcoreapp3.0/Processes"; // This is an executable program.
prInfo.CreateNoWindow = false; // This means start the process in a new window
prInfo.UseShellExecute = false;

try
{
    // Start the defined process
    using (Process pr = Process.Start(prInfo))
    {
        Thread.Sleep(100);
        Console.WriteLine("I can be busy here doing something else ... ");

        pr.WaitForExit(); // Parent process waits here to have the child finished.
    }
}
```

Start the process, continue with the execution, wait for the created process to finish

IPC: Pipes (Named)



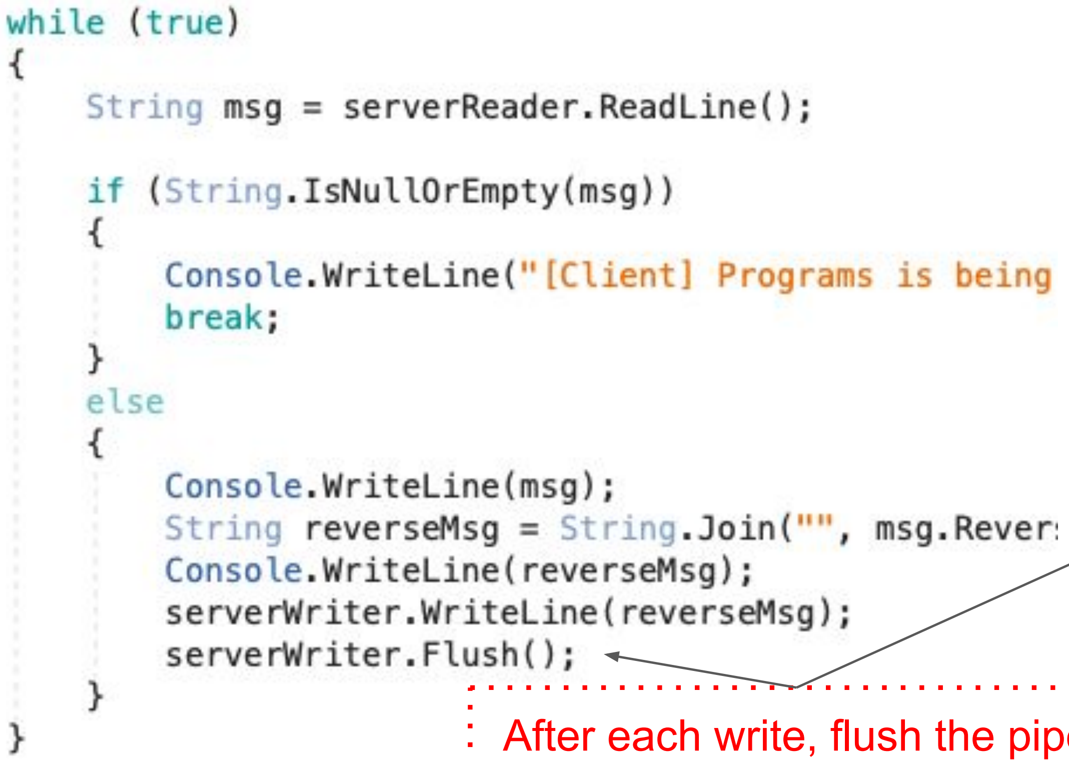
```
NamedPipeServerStream server;
StreamReader serverReader;
StreamWriter serverWriter;

public SolutionIPCNamedClient(String pipeName)
{
    server = new NamedPipeServerStream(pipeName);
}

public void prepareClient()
{
    Console.WriteLine("Pipe Client is being execute
    Console.WriteLine("[Client] Client will be wait

    server.WaitForConnection();
    serverReader = new StreamReader(server);

    // The client needs a writer stream to write it
    serverWriter = new StreamWriter(server);
}
```



```
NamedPipeClientStream client;
StreamReader clientReader;
StreamWriter clientWriter;

public SolutionIPCNamedServer(String pipeName)
{
    client = new NamedPipeClientStream(pipeName);
}

public void prepareServer()
{
    Console.WriteLine("Pipe Server is being execut
    Console.WriteLine("[Server] Enter a message to
    client.Connect();
    clientReader = new StreamReader(client);
    clientWriter = new StreamWriter(client);
}
```

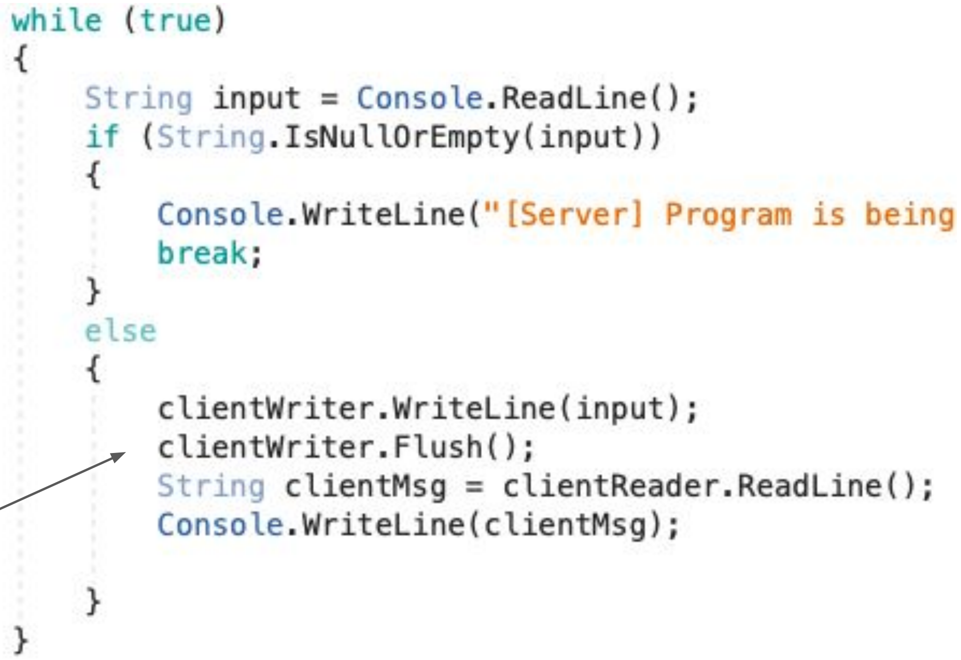
IPC: Communicate

client



```
while (true)
{
    String msg = serverReader.ReadLine();

    if (String.IsNullOrEmpty(msg))
    {
        Console.WriteLine("[Client] Programs is being
        break;
    }
    else
    {
        Console.WriteLine(msg);
        String reverseMsg = String.Join("", msg.Rever
        Console.WriteLine(reverseMsg);
        serverWriter.WriteLine(reverseMsg);
        serverWriter.Flush();
    }
}
```

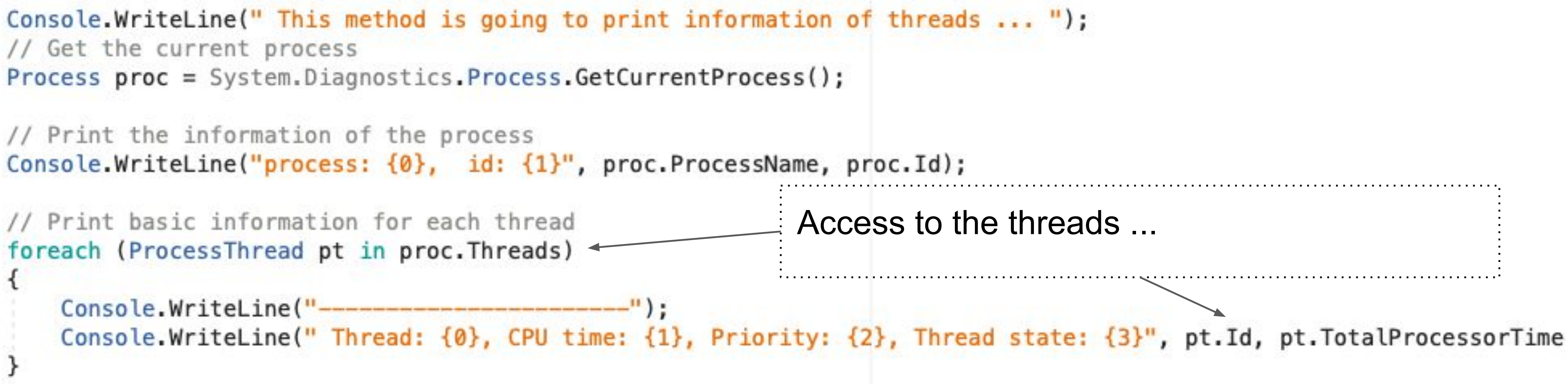
server



```
while (true)
{
    String input = Console.ReadLine();
    if (String.IsNullOrEmpty(input))
    {
        Console.WriteLine("[Server] Program is being
        break;
    }
    else
    {
        clientWriter.WriteLine(input);
        clientWriter.Flush();
        String clientMsg = clientReader.ReadLine();
        Console.WriteLine(clientMsg);

    }
}
```

After each write, flush the pipe

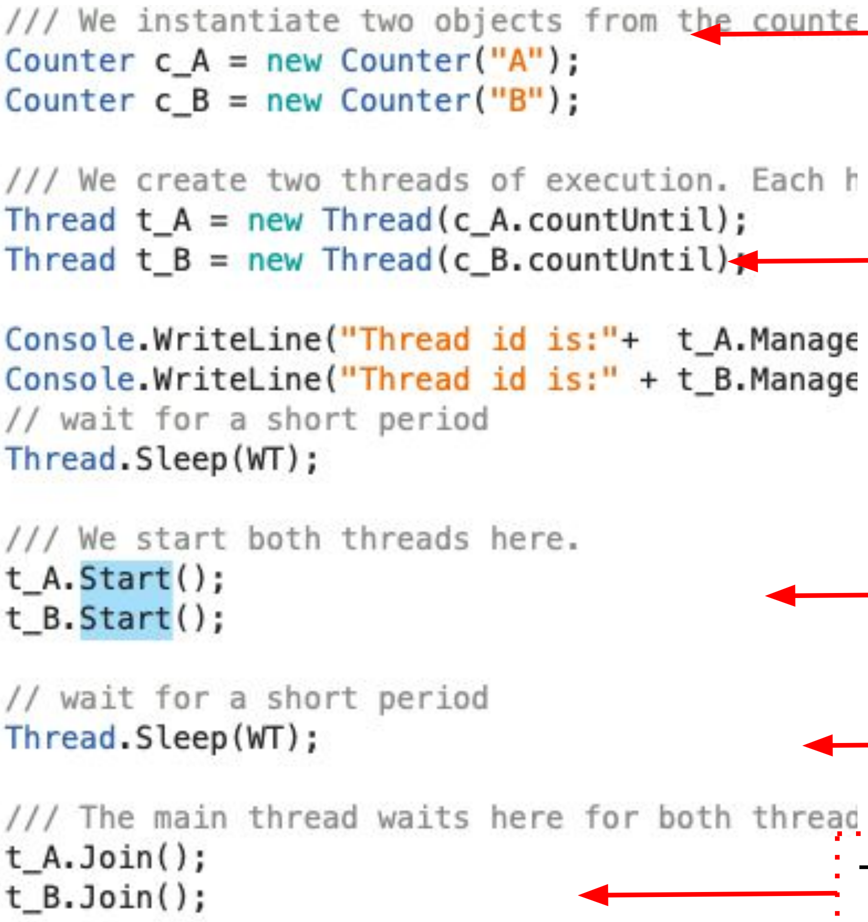
Threads



```
Console.WriteLine(" This method is going to print information of threads ... ");
// Get the current process
Process proc = System.Diagnostics.Process.GetCurrentProcess();

// Print the information of the process
Console.WriteLine("process: {0},  id: {1}", proc.ProcessName, proc.Id);

// Print basic information for each thread
foreach (ProcessThread pt in proc.Threads)
{
    Console.WriteLine("-----------------------");
    Console.WriteLine(" Thread: {0}, CPU time: {1}, Priority: {2}, Thread state: {3}", pt.Id, pt.TotalProcessorTime
}
```

Access to the threads ...

Threads: Creation



```
/// We instantiate two objects from the counte
Counter c_A = new Counter("A");
Counter c_B = new Counter("B");

/// We create two threads of execution. Each h
Thread t_A = new Thread(c_A.countUntil);
Thread t_B = new Thread(c_B.countUntil);

Console.WriteLine("Thread id is:"+  t_A.Manage
Console.WriteLine("Thread id is:" + t_B.Manage
// wait for a short period
Thread.Sleep(WT);

/// We start both threads here.
t_A.Start();
t_B.Start();

// wait for a short period
Thread.Sleep(WT);

/// The main thread waits here for both thread
t_A.Join();
t_B.Join();
```

Instances responsible for (concurrent) tasks

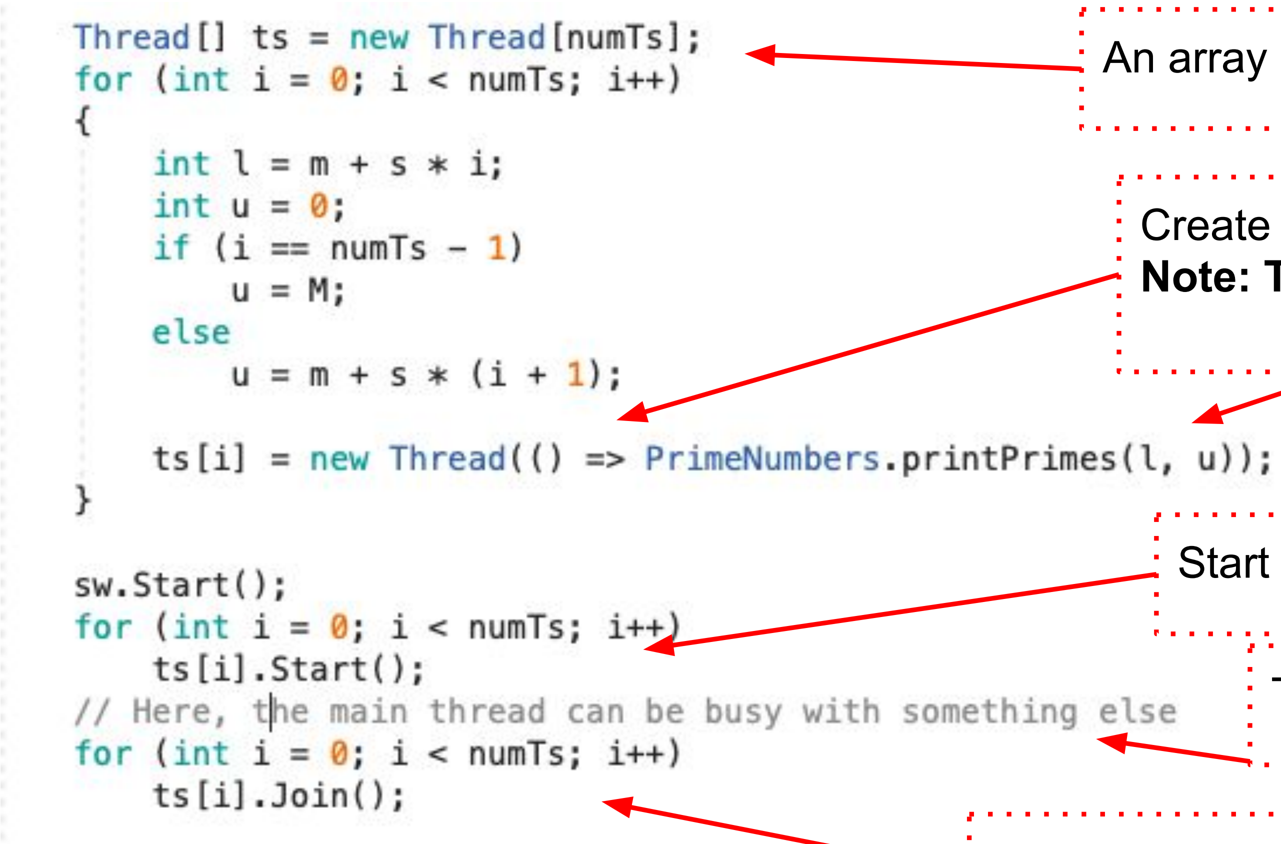
Create threads: **<u>What is the task?</u>**
<u>Note: Only name of the method, no parameter</u>

Start the threads

The main thread can continue its tasks ...

The main thread joins the other threads

Threads: Multiple threads



```
Thread[] ts = new Thread[numTs];
for (int i = 0; i < numTs; i++)
{
    int l = m + s * i;
    int u = 0;
    if (i == numTs - 1)
        u = M;
    else
        u = m + s * (i + 1);

    ts[i] = new Thread(() => PrimeNumbers.printPrimes(l, u));
}

sw.Start();
for (int i = 0; i < numTs; i++)
    ts[i].Start();
// Here, the main thread can be busy with something else
for (int i = 0; i < numTs; i++)
    ts[i].Join();
```

An array of threads

Create threads: **<u>What is the task?</u>**
Note: This needs parameters

Start the threads

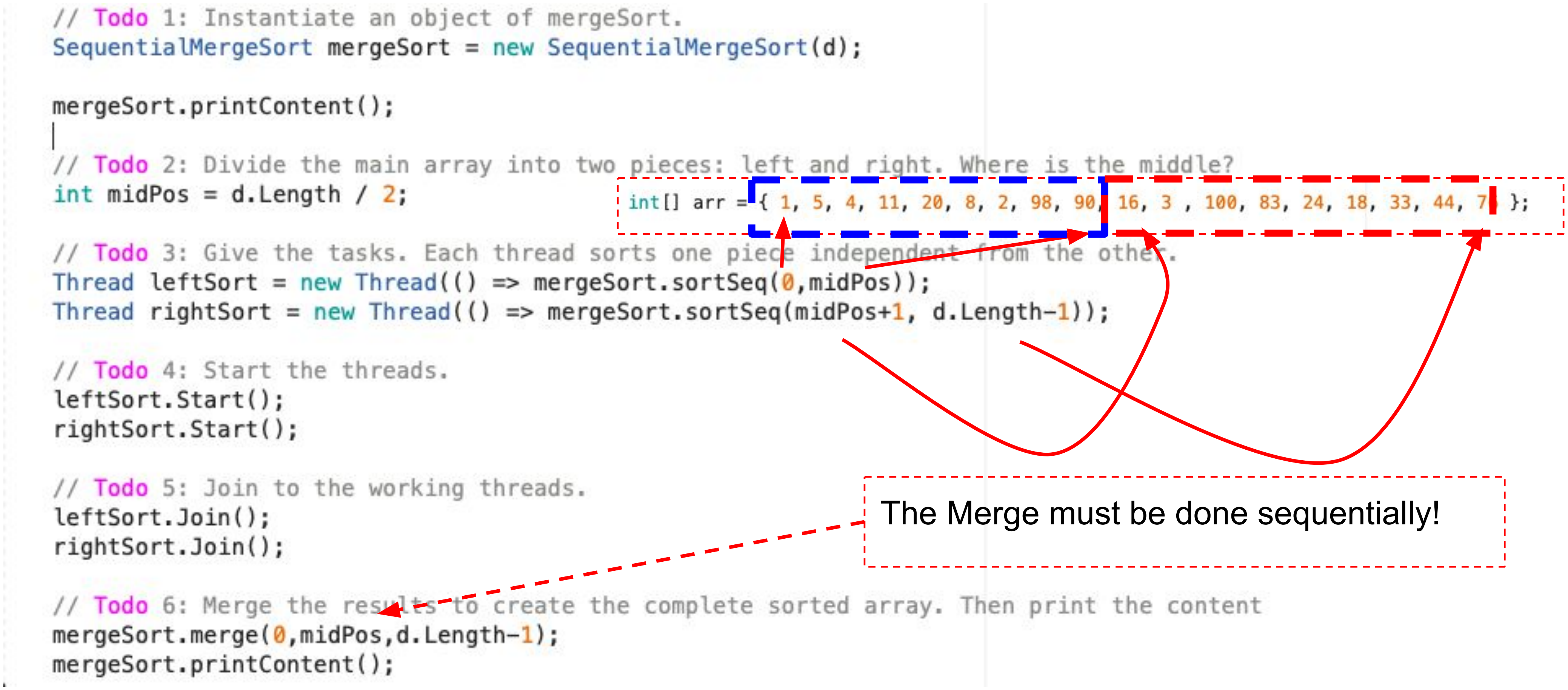
The main thread can continue its tasks ...

The main thread joins the other threads

Threads: passing params to the thread

```
class TestTasks{
    public void methObjParam(Object o)
    {
        int val =  (int)o;
        Console.WriteLine("This is the parameter {0}:",val);
    }
    public void methIntParam(int i)
    {   Console.WriteLine("This is the parameter {0}:", i);    }
    public void task()
    {
        Thread t1 = new Thread( methObjParam);  // correct
        t1. Start(10) ;
        t1.Join();
        Thread t2 = new Thread( methIntParam); // Compile Error:
        t2. Start(10) ;
        t2.Join();
        Thread t3 = new Thread( ()  => methIntParam(10) ); // correct
        t3. Start() ;
        t3.Join();
    }
 }
```

Threads: Tasks divisions

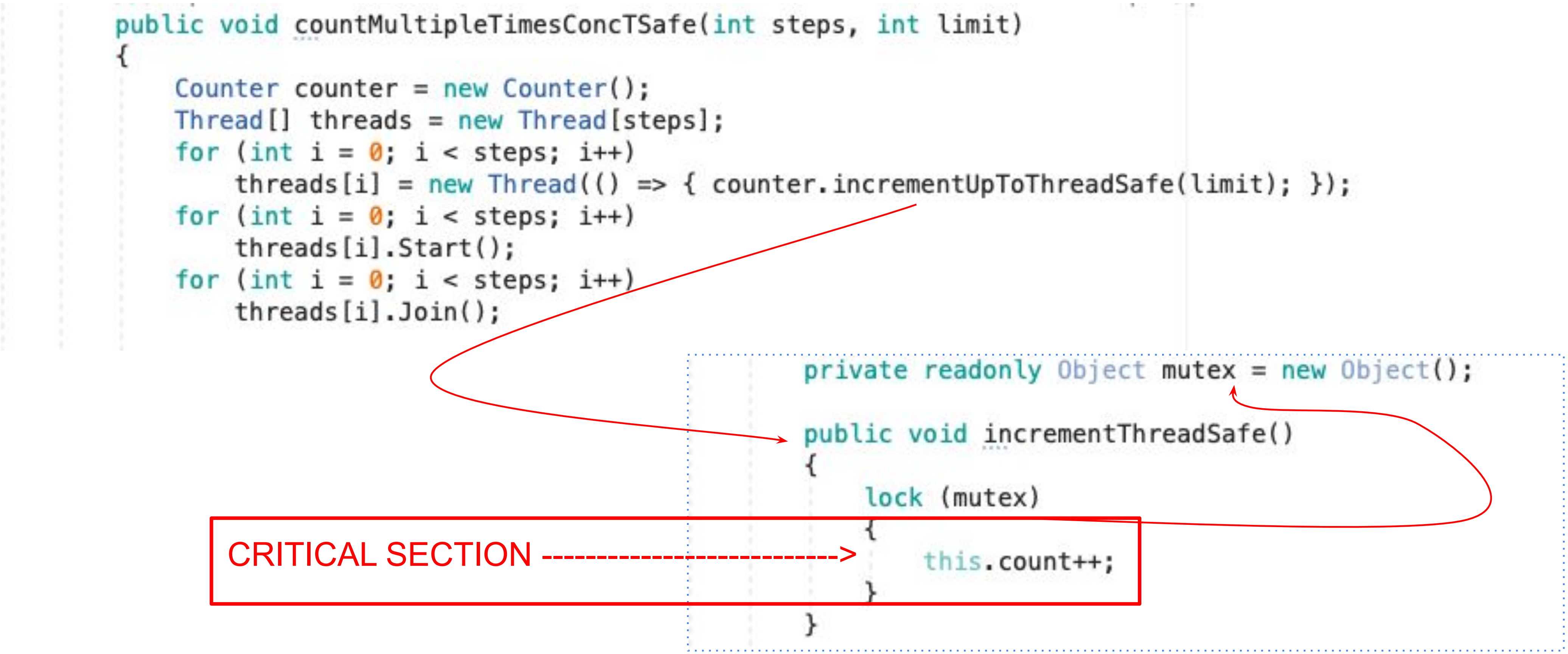


```
// Todo 1: Instantiate an object of mergeSort.
SequentialMergeSort mergeSort = new SequentialMergeSort(d);

mergeSort.printContent();

// Todo 2: Divide the main array into two pieces: left and right. Where is the middle?
int midPos = d.Length / 2;

// Todo 3: Give the tasks. Each thread sorts one piece independent from the other.
Thread leftSort = new Thread(() => mergeSort.sortSeq(0,midPos));
Thread rightSort = new Thread(() => mergeSort.sortSeq(midPos+1, d.Length-1));

// Todo 4: Start the threads.
leftSort.Start();
rightSort.Start();

// Todo 5: Join to the working threads.
leftSort.Join();
rightSort.Join();

// Todo 6: Merge the results to create the complete sorted array. Then print the content
mergeSort.merge(0,midPos,d.Length-1);
mergeSort.printContent();
```

```
int[] arr ={ 1, 5, 4, 11, 20, 8, 2, 98, 90, 16, 3 , 100, 83, 24, 18, 33, 44, 7 };
```

The Merge must be done sequentially!

Lock

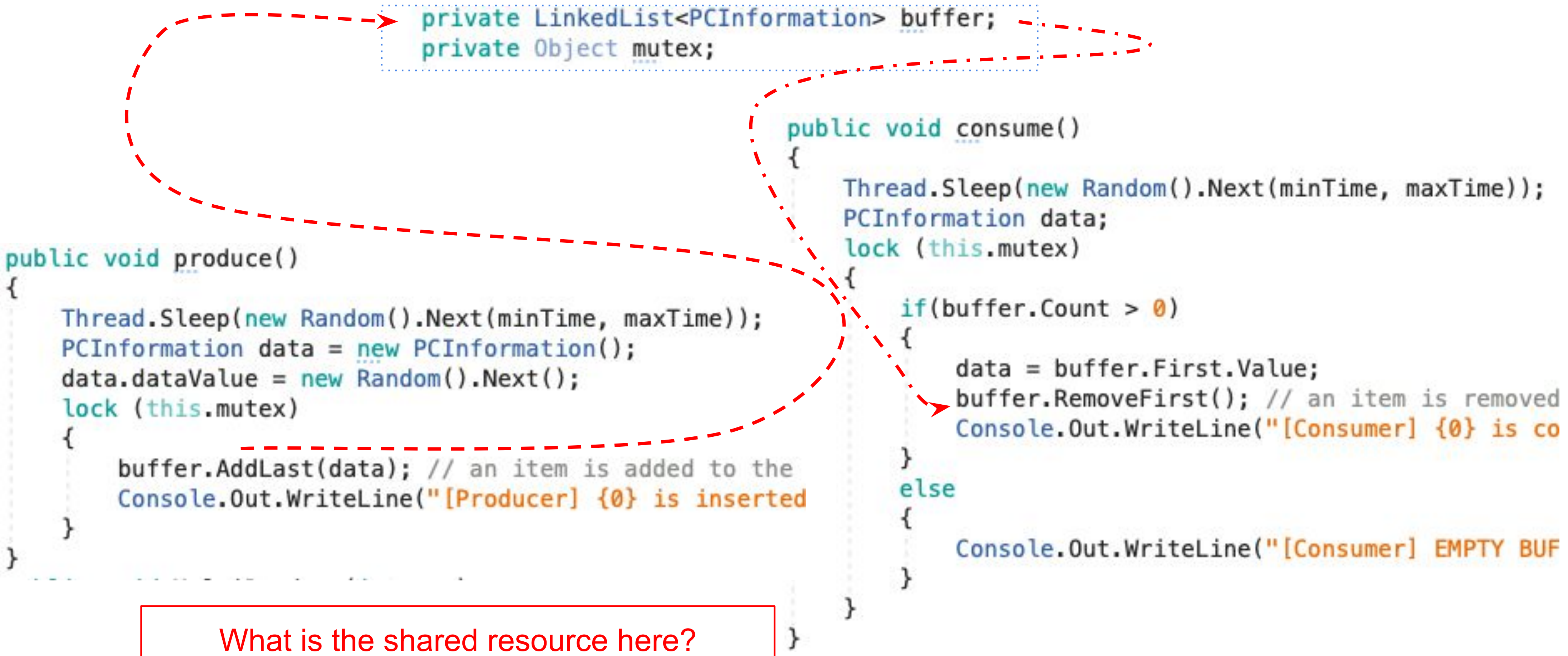


```
public void countMultipleTimesConcTSafe(int steps, int limit)
{
    Counter counter = new Counter();
    Thread[] threads = new Thread[steps];
    for (int i = 0; i < steps; i++)
        threads[i] = new Thread(() => { counter.incrementUpToThreadSafe(limit); });
    for (int i = 0; i < steps; i++)
        threads[i].Start();
    for (int i = 0; i < steps; i++)
        threads[i].Join();
```

```
private readonly Object mutex = new Object();

public void incrementThreadSafe()
{
    lock (mutex)
    {
        this.count++;
    }
}
```

CRITICAL SECTION ----------------------------->

Protection



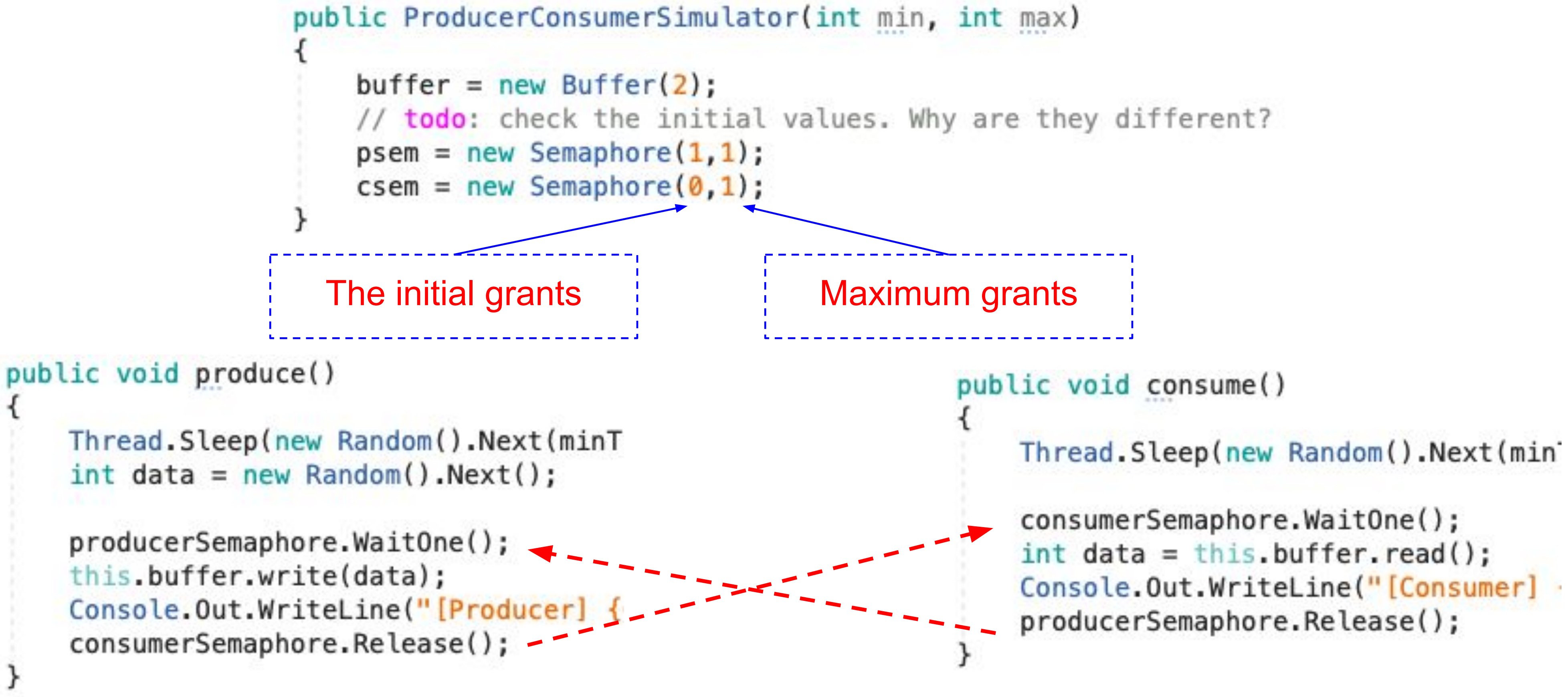
```
private LinkedList<PCInformation> buffer;
private Object mutex;
```

```
public void produce()
{
    Thread.Sleep(new Random().Next(minTime, maxTime));
    PCInformation data = new PCInformation();
    data.dataValue = new Random().Next();
    lock (this.mutex)
    {
        buffer.AddLast(data); // an item is added to the
        Console.Out.WriteLine("[Producer] {0} is inserted
    }
}
```

```
public void consume()
{
    Thread.Sleep(new Random().Next(minTime, maxTime));
    PCInformation data;
    lock (this.mutex)
    {
        if(buffer.Count > 0)
        {
            data = buffer.First.Value;
            buffer.RemoveFirst(); // an item is removed
            Console.Out.WriteLine("[Consumer] {0} is co
        }
        else
        {
            Console.Out.WriteLine("[Consumer] EMPTY BUF
        }
    }
}
```

What is the shared resource here?

Semaphores



```
public ProducerConsumerSimulator(int min, int max)
{
    buffer = new Buffer(2);
    // todo: check the initial values. Why are they different?
    psem = new Semaphore(1,1);
    csem = new Semaphore(0,1);
}
```

The initial grants

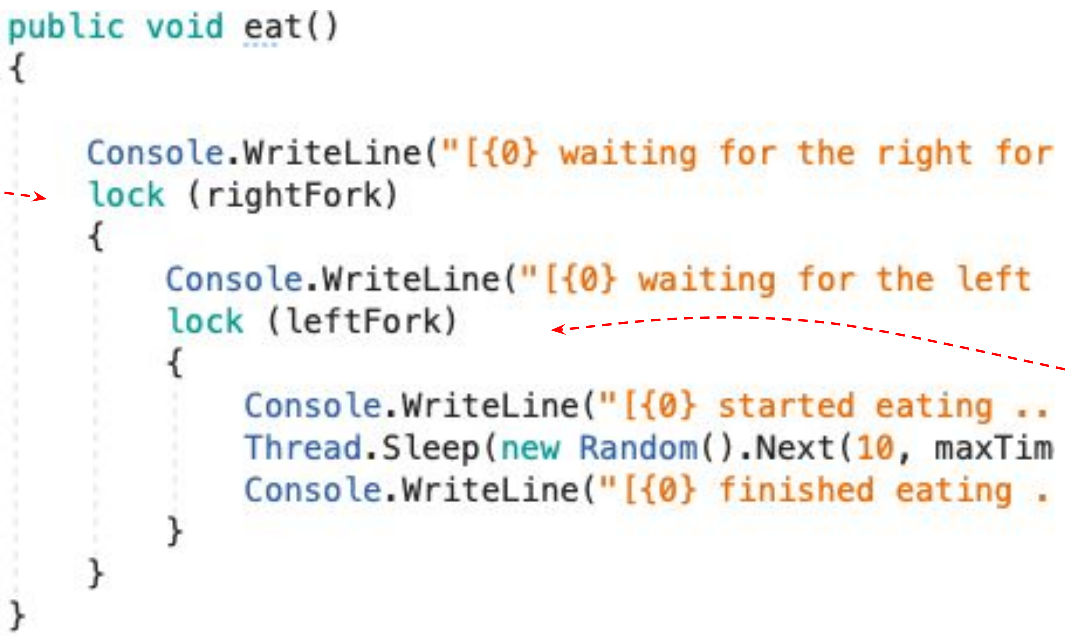
Maximum grants

```
public void produce()
{
    Thread.Sleep(new Random().Next(minT
    int data = new Random().Next();

    producerSemaphore.WaitOne();
    this.buffer.write(data);
    Console.Out.WriteLine("[Producer] {
    consumerSemaphore.Release();
}
```

```
public void consume()
{
    Thread.Sleep(new Random().Next(min

    consumerSemaphore.WaitOne();
    int data = this.buffer.read();
    Console.Out.WriteLine("[Consumer]
    producerSemaphore.Release();
}
```

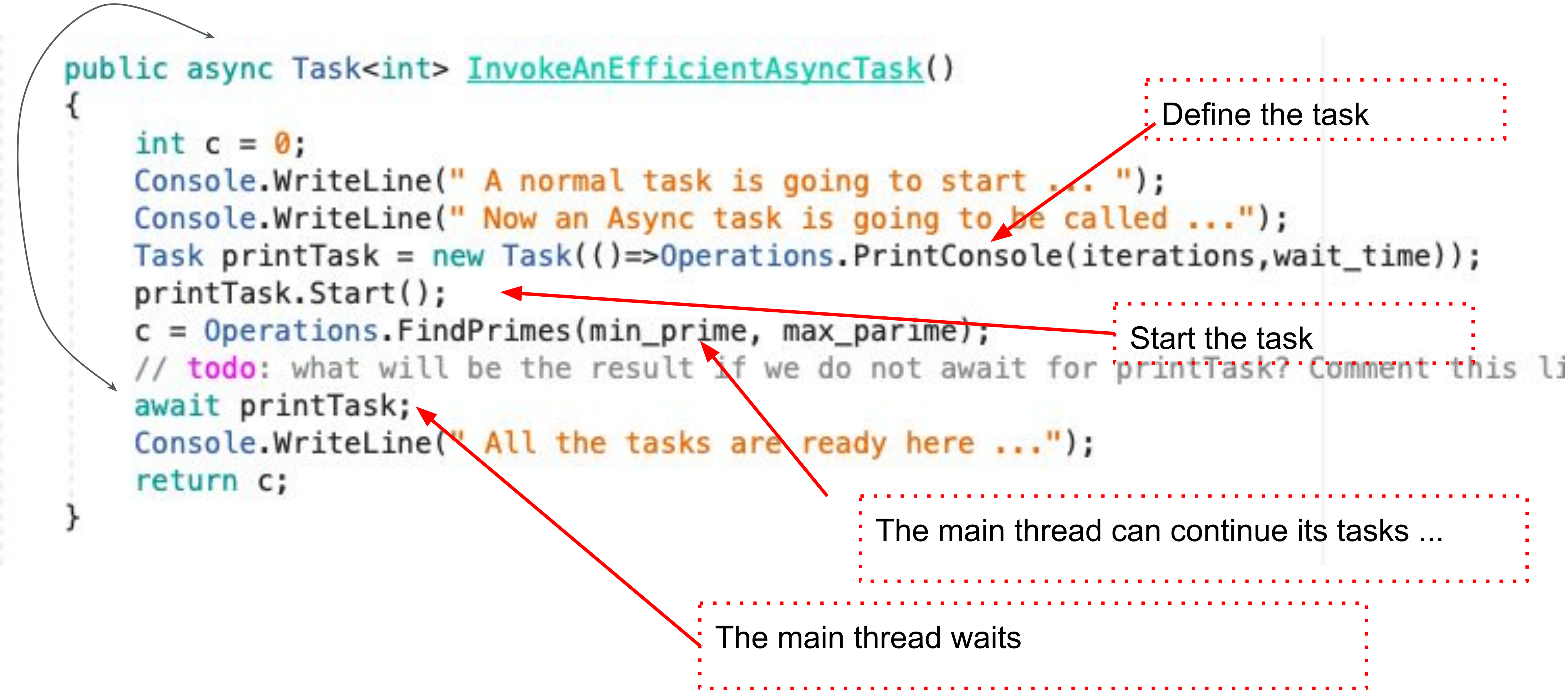
Deadlocks

Exclusive access, Incremental access (Hold-and-wait), Circular waiting, No preemption



```
public void eat()
{

    Console.WriteLine("[{0} waiting for the right for
    lock (rightFork)
    {
        Console.WriteLine("[{0} waiting for the left
        lock (leftFork)
        {
            Console.WriteLine("[{0} started eating ..
            Thread.Sleep(new Random().Next(10, maxTim
            Console.WriteLine("[{0} finished eating .
        }
    }
}
```

Asynchronous Operations



```
public async Task<int> InvokeAnEfficientAsyncTask()
{
    int c = 0;
    Console.WriteLine(" A normal task is going to start ... ");
    Console.WriteLine(" Now an Async task is going to be called ...");
    Task printTask = new Task(()=>Operations.PrintConsole(iterations,wait_time));
    printTask.Start();
    c = Operations.FindPrimes(min_prime, max_parime);
    // todo: what will be the result if we do not await for printTask? Comment this li
    await printTask;
    Console.WriteLine(" All the tasks are ready here ...");
    return c;
}
```

Define the task

Start the task

The main thread can continue its tasks ...

The main thread waits