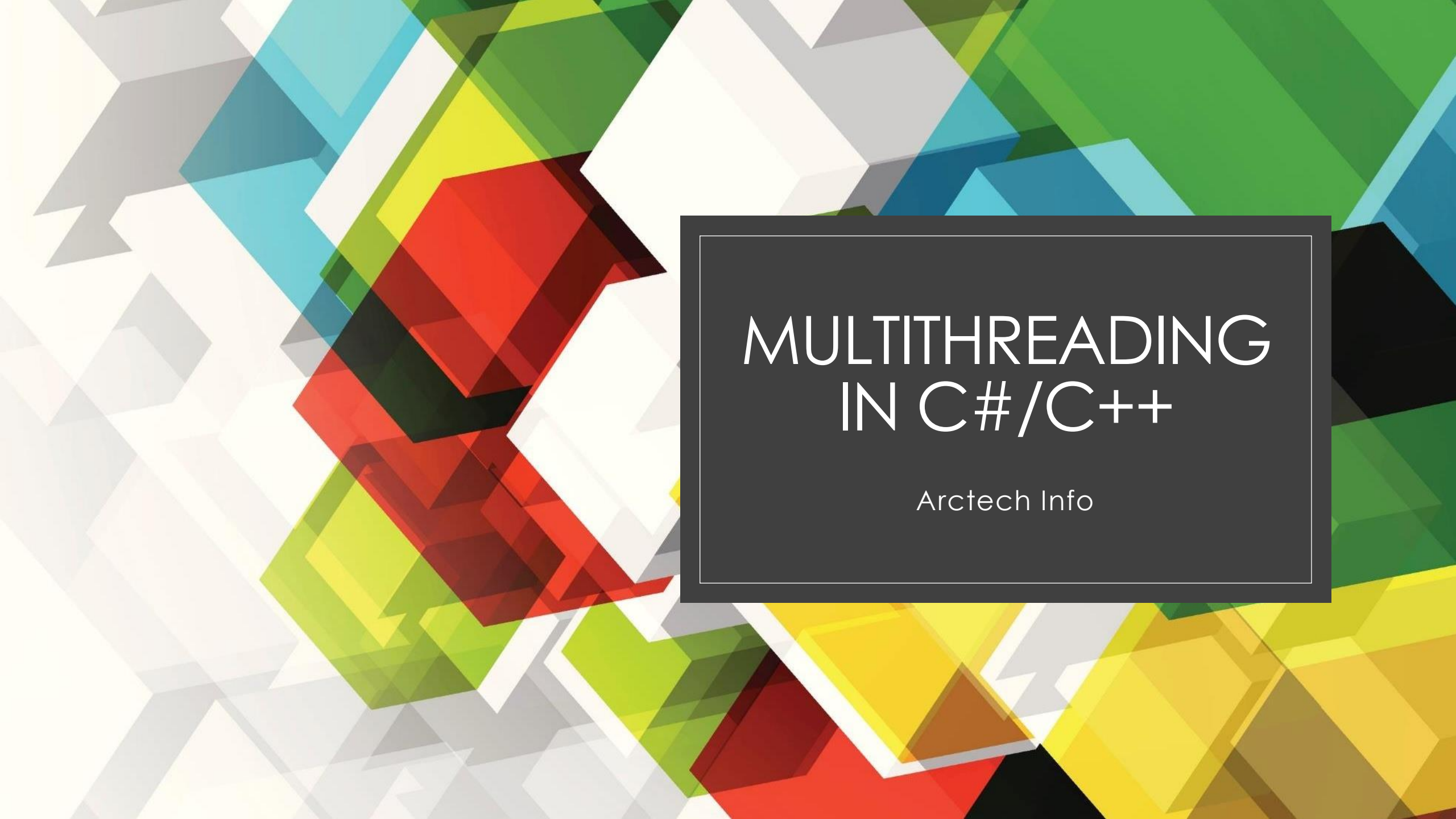

MULTITHREADING
IN C#/C++
Arctech Info

Multithreading in C#/C++

THREADS

THREADPOOL

TASKS

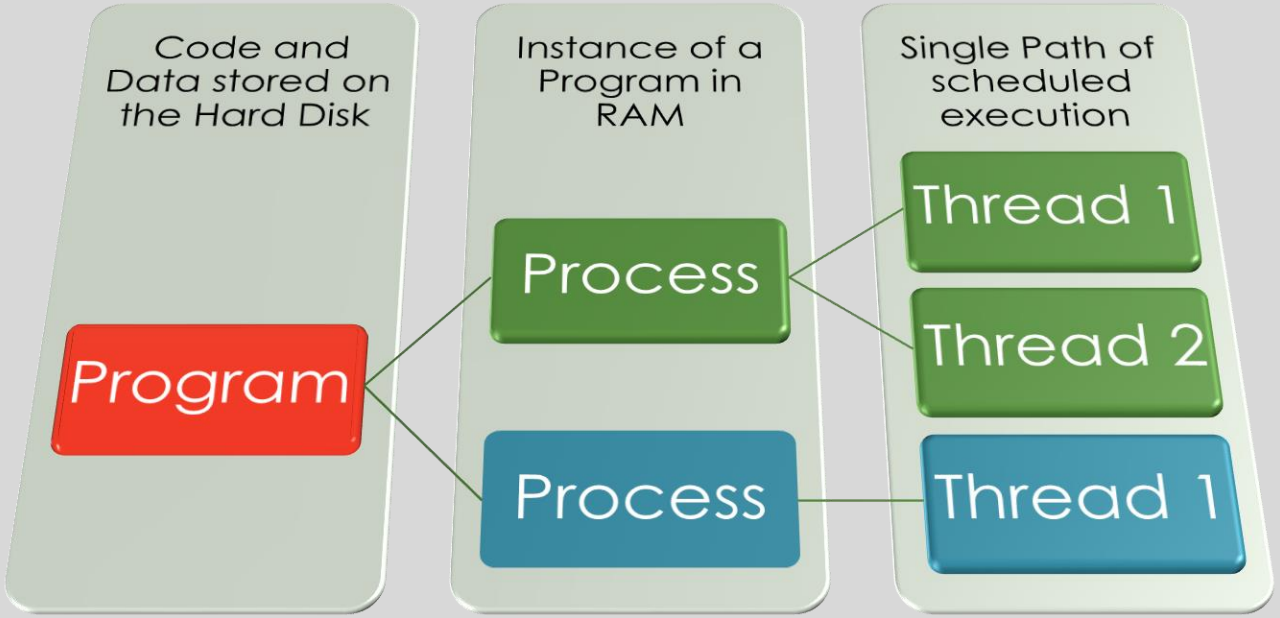
Multi-Tasking

- Multitasking is the simultaneous execution of multiple tasks or processes over a certain time interval.
- All modern operating systems (like Windows) implement multi-tasking.
- An Operating Systems
  - has its own programs files like boot program, the application manger, thread scheduler etc.
  - other Applications like Word, Excel, Notepad etc. (on Windows OS)
- An application consists of one or more processes
  - A process (in simple terms) is an executing program. See Task Manager in Windows
  - Every program that executes on your system is a process (or a collection of processes)
- Every single process can have one or more threads running in the context of the process.
- Note: Processes are heavier than threads
  - i.e. Processes have higher memory consumption and uses more battery power on a laptop

Some application examples

- Chrome supports multiple tabs via a multi-process architecture
  - 1 process per tab, so 10 tabs => 10 processes
  - Each process(tab) has 3 important threads and a few more threads
    - Main Thread
      - Allows users to interact with the tab, i.e. minimize, close, type a new url etc.
    - IO thread
      - Handles network communications.
      - Connects to the server and downloads the website
    - Renderer Thread
      - The downloaded website is a file in a specific format (HTML)
      - Parses the website file and displays it on the inside area of the Chrome Tab.
- FireFox supports multi-tabs via a multi-thread architecture
- Excel has 1 process per open instance/file (Most common architecture)
  - Each process has multiple threads to manage various functionality
  - E.g. Main Thread, formula calculation thread, etc.

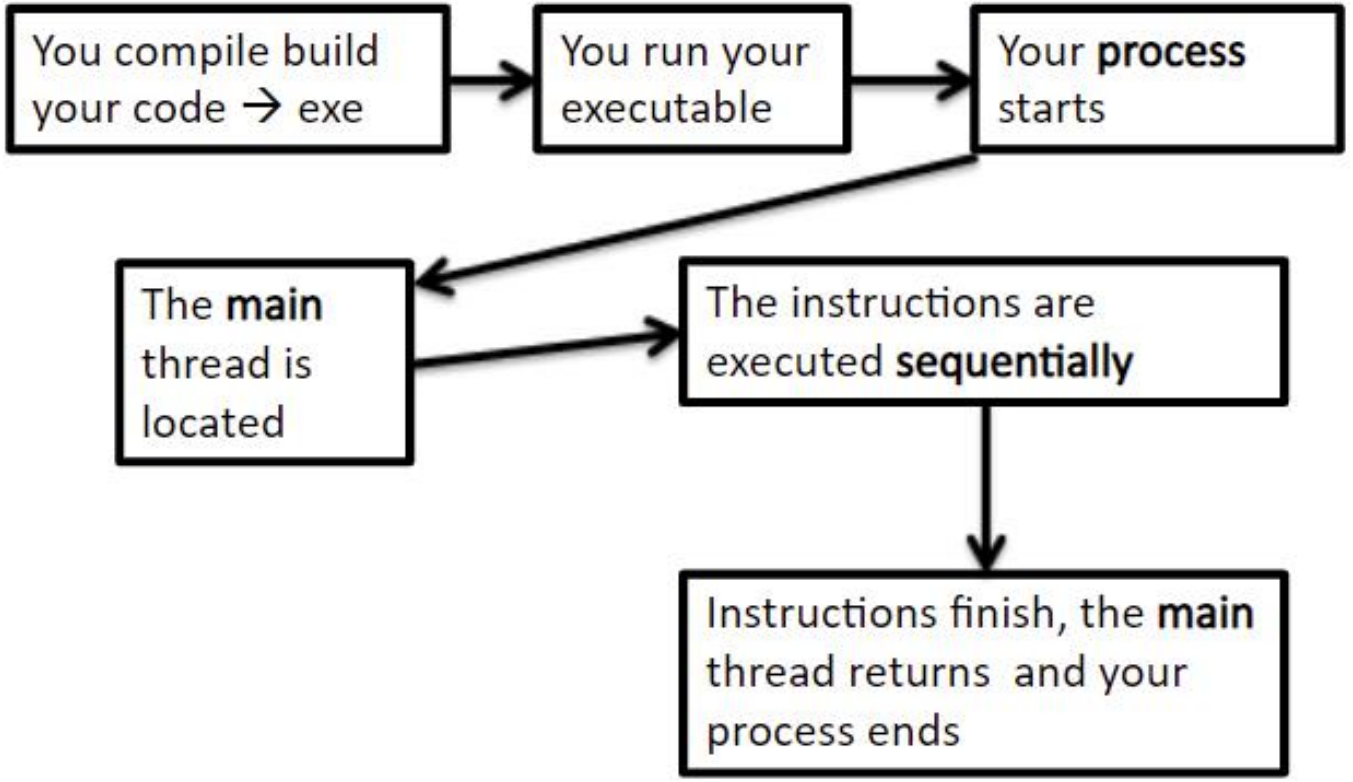
Processes & Threads



What is a Thread

- Basic unit of execution (A lightweight process)
- Managed by the Thread Scheduler which is part of a Operating System.
  - Is allocated processor time by the Operating System
- Every program has some logic, and a thread is responsible for executing this logic.
- Every program by default carries one thread to executes the logic of the program
  - the thread is known as the Main Thread,
- So, every program or application is by default single-threaded model.

A typical windows exe



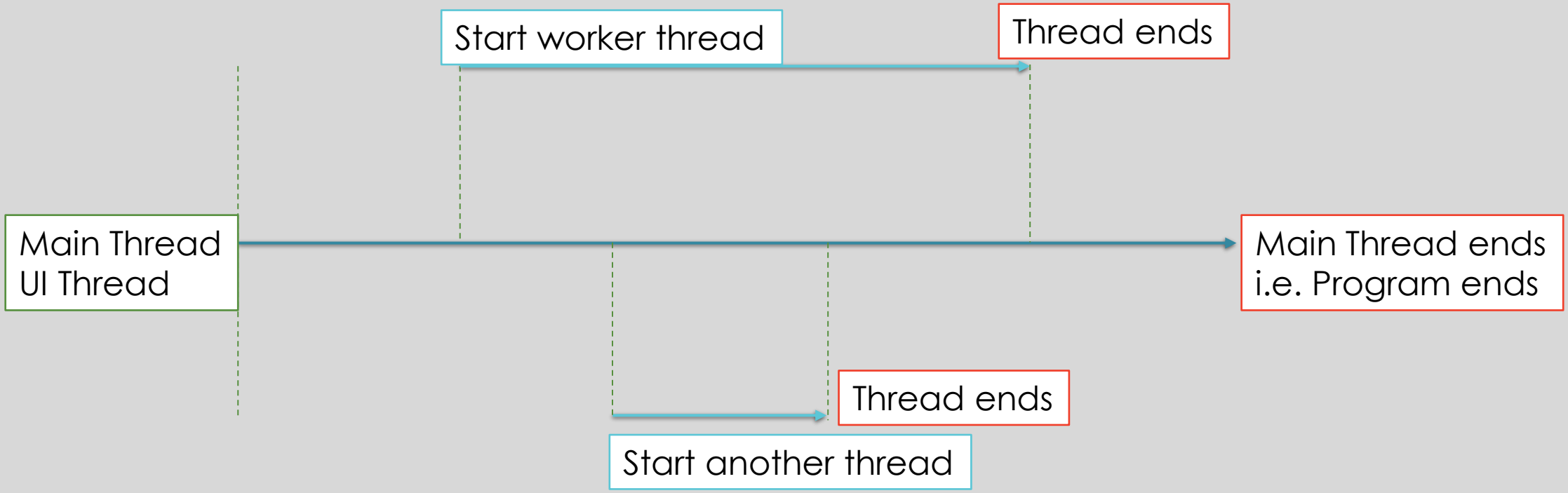
Drawbacks of Single Threaded Model

- A single thread runs all the instruction set (code blocks, methods, statements) present in the program in synchronized manner
  - one after another.
- The programs takes longer to run.
- For example, we have a class named as Animal and this class contains two different methods
  - Method1, Method2.
- Now the main thread is responsible for executing all these methods, so the main thread executes all these methods one by one.
  - The Main thread executes Method1 fully before executing Method2.

Multi-threading

- Multi-threading contains multiple threads within a single process.
- Here each thread performs different activities.
- For example, in the previous class, we can use multithreading, so each method is executed by a separate thread.
- The major advantage of multithreading is it works simultaneously
  - i.e. Multiple tasks can execute at the same time.
- This maximizes the utilization of the CPU because multithreading works on time-sharing concept.
- Each thread takes its own time for execution and does not affect the execution of another thread, this time interval is given by the operating system.

Main Thread and Worker Threads



Working with Threads in .NET

- Create and start a new thread
  - Create a new instance of the **System.Threading.Thread** class.
  - Provide the name of the method that you want to execute on a new thread to the constructor.
  - To start a created thread, call the **Thread.Start** method.
- Create and initialize two threads

```
Thread thread1 = new Thread(Show1);
Thread thread2 = new Thread(Show2);
```

- Show1 & Show2 are methods

```
void Show1() {} & void Show2() {}
```

- Now start the execution of both the threads.

```
thread1.Start();
thread2.Start();
```

Working with Threads in C++

- Create and start a new thread
  - Create a new instance of the **std::thread** class.
  - Provide the name of the method that you want to execute on a new thread to the constructor.
- Create and start two threads

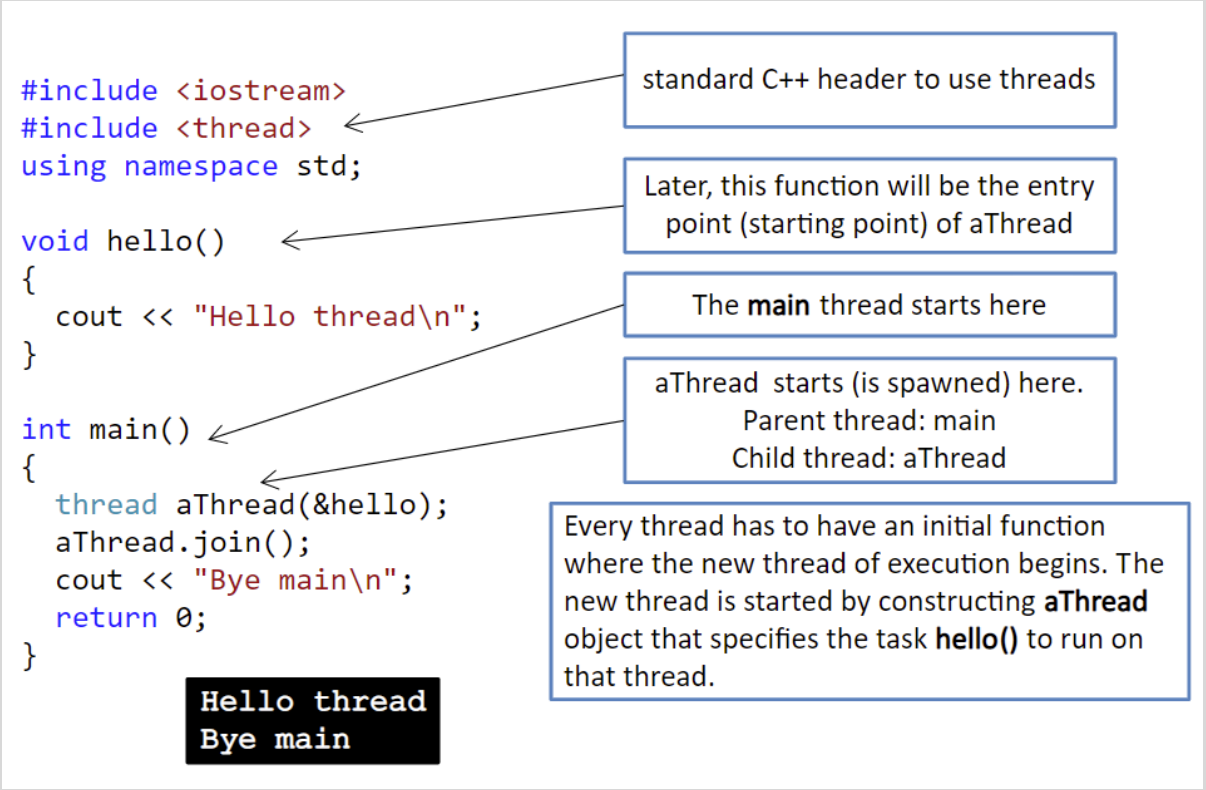
```
#include <thread>
using namespace std;
thread thread1(Show1);
thread thread1(Show2);
```

- Show1 & Show2 are methods

```
void Show1() {} & void Show2() {}
```

Threads in C++

- Till C++ 11, there was no standard way to work with threads
- 3rd party libraries/framework were used.
- Every library or framework implemented threads in different ways
- Now we use the standard thread implementation in the STL.



Review Example

- Both thread runs simultaneously and the processing of thread2 does not depend upon the processing of thread1 like in the single threaded model.
- **Note:** Output may vary due to context switching.
- Advantages of Multithreading:
  - It executes multiple code sequences simultaneously.
  - Maximize the utilization of CPU resources.
  - Time sharing between multiple process.

OS Scheduler - Allocate CPU time

- Single Processor vs Multi-Processor
  - Single processor Core
    - Only 1 thread can run at one time
  - Multiple cores e.g. 4 cores
    - 4 threads can run simultaneously
- OS Scheduler & multi-tasking
  - Allows unlimited threads (theoretically) to run simultaneously**
  - TimeSlicing
    - Rapidly switching execution between all active threads.
    - In Windows, timeslicing for each thread typically 10-20 milliseconds.
    - When a thread is interrupted by timeslicing, it is said to be Pre-empted.
    - The CPU is no longer executing a pre-empted thread
    - The OS scheduler will keep running and pre-empting threads, over and over again
    - A Thread itself has no control over when it is pre-empted.

FIFO Thread Scheduler – 2 core CPU

Ready Queue (FIFO)

- Thread 3
- Thread 4
- Thread 5
- Thread 6
- Thread 7
- Thread 8

Assign thread 1& 2 to the Run Queue

Run Queue (FIFO)

- Thread 1
- Thread 2

FIFO Thread Scheduler – 2 core CPU

Ready Queue

Thread 5

Thread 6

Thread 7

Thread 8

Thread 1

Thread 2

After some time**, the scheduler move thread 1 & thread 2 back to the ready queue

Move the next threads from the ready queue to the available slot on the Run Queue

Run Queue

Thread 3

Thread 4

FIFO Thread Scheduler – 2 core CPU

Ready Queue

Thread 7

Thread 8

Thread 1

Thread 2

Thread 3

Thread 4

This continues in a FIFO, round-robin fashion. Some Threads may be killed from the ready queue and new threads may be created.

Run Queue

Thread 5

Thread 6

Thread States

- Ready (**Ready Queue**)
  - Ready to run
- Executing (**Run Queue**)
  - Running
- Blocked
  - Waiting for an event
- Ended
  - The thread is no longer running

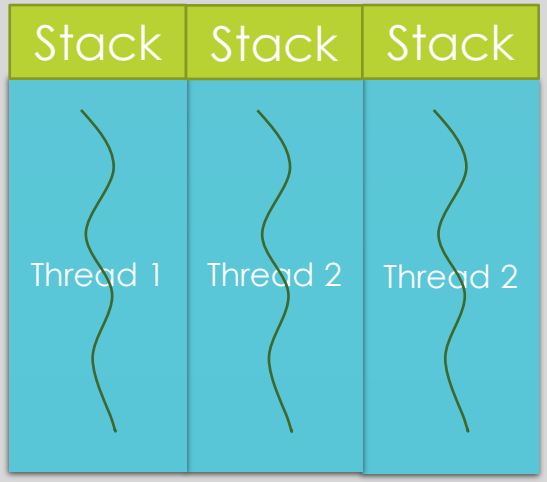
Priority Scheduler vs FIFO Scheduler

- Priority Scheduler is more common than FIFO
- Similar concept as the FIFO scheduling
- However, each thread has a priority associated with it
- The Scheduler places the high priority threads higher up in the **Ready Queue**
- It also leaves the high priority threads on the **Run Queue** for longer time.
- This ensures, the CPU starts working on the **high priority threads** 1st and also executes their instructions longer, than the **lower priority threads**

Local variables in Multiple Threading

- You can have multiple threads executing the same method.
- CLR assigns each thread its own local memory stack, to keep local variables separate.
- A separate copy of the local variables are created on that threads memory stack. See Example
- **Note:** All threads within a process share the same heap memory.

Single Threaded Process



Multi Threaded Process

Foreground vs Background Threads

- Background threads are identical to foreground threads with one difference
- A background thread does not keep the application running.
- Once all foreground threads have stopped, the system abruptly terminates all background threads and shuts down the application.
- No Exceptions are thrown, and the application simply ends without waiting for the background thread to finish.
- In C++, by default **std::thread** is a background thread.
- In C#, by default **System.Threading.Thread** is a foreground thread. To create a background thread.
  - **Thread t1 = new Thread(fn);**
  - **t1.IsBackground = true;**