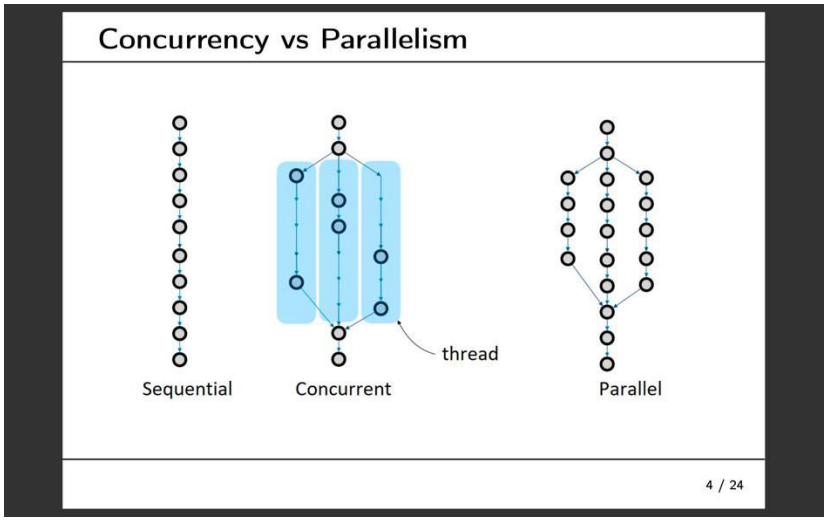
CS2030 Parallelism and Asynchronous Programming

Parallel Progamming

What is Parallel Programming?

Parallel programming is to use multiple threads to speed up the computation of a program, to more efficiently utilize computational resources.



What kind of tasks are best suited for parallelism?
In terms of parallel streams, stateless operations without side effects are best for usage, especially commutative operations!

- This is because as tasks are completed, other threads may still be processing the previous operation, hence there is no inherent order to the completion of each sub-operation in the program.

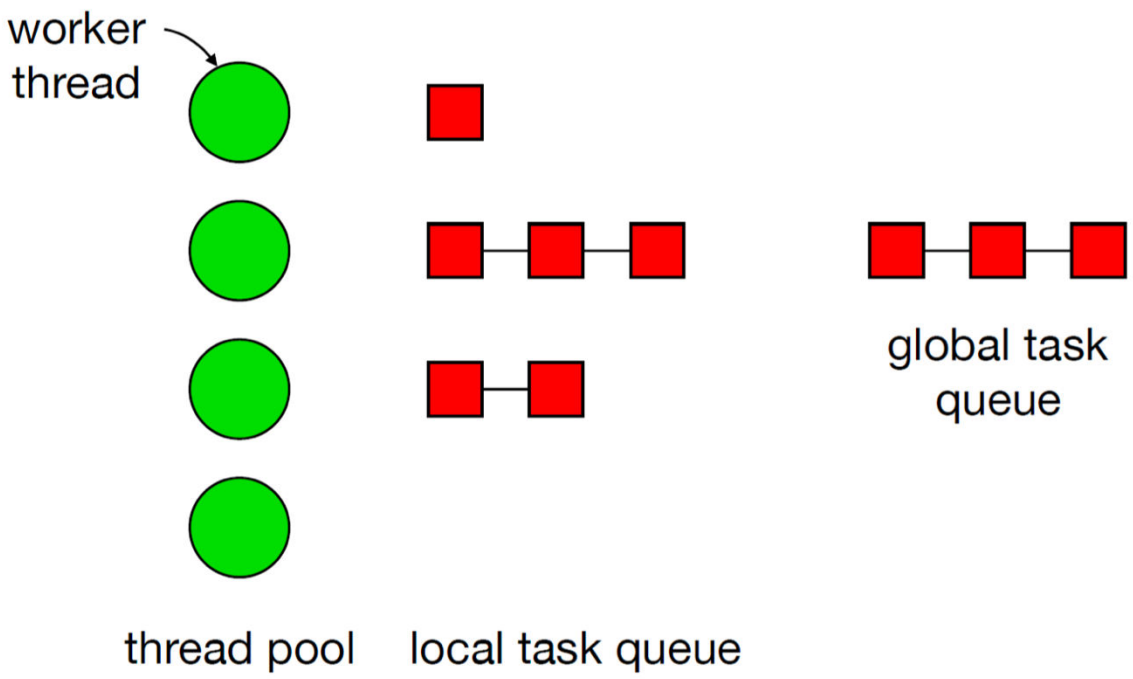
The task should also be relatively computationally expensive.

- This is because there are associated overhead costs with the implementation of parallelism that only make it worthwhile to compute in parallel when dealing with a large problem and outweigh the overheads involved.

Under the hood of Parallelism:

How parallelism works is that, under the hood when parallelism is enabled, work is first processed by the main thread in a double ended queue, with other threads also having one of their own.

Then the main thread forks a task, lets say Matrix Multiplication of a $2^n * 2^n$ matrix.
The more related task is kept by the main thread to continue working on it, while the partitioned out part is added to a work deque for the thread.



This subtask is forked and thrown to the work deque.

Then a worker thread that is free steals the work from the **back of the queue** from main.

A similar approach happens with each thread.

When the main is done with the subtask, it takes work from the front of its queue.

Hence Work Stealing from other threads is from the back of the queue while taking from your own queue is from the front (LIFO).

Asynchronous Programming using Completable Future

In Asynchronous Programming, it uses the same backend logic as that of Parallelism, However, the value of asynchronous programming is that we **can do a totally unrelated task** as that of a task thrown to the worker pool while waiting for that task, (likely computationally expensive) to complete.

This requires an understanding of how Java Multi-threading occurs.

Note that CompletableFuture is a Functor and a Monad, and hence can be chained together like a stream or an optional.