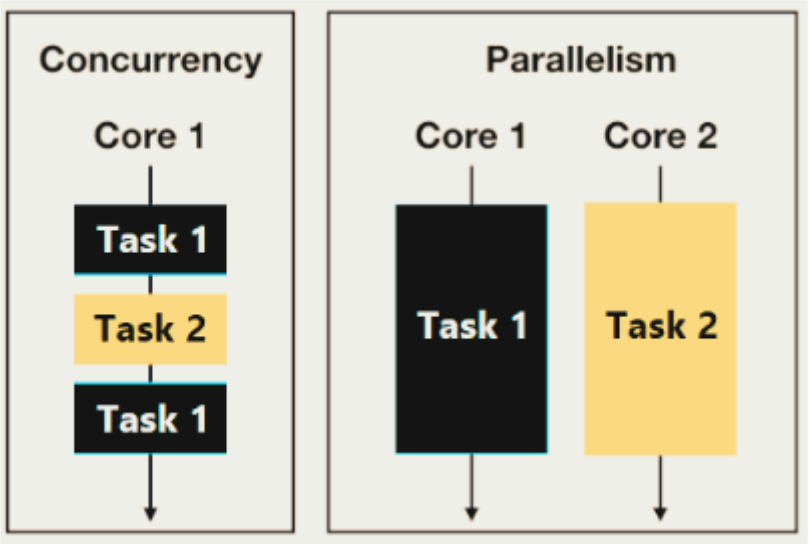
Concurrency is when multiple tasks start, run, and complete with each other to progress in overlapping time periods, in no specific order. Parallelism is when multiple tasks OR several parts of a unique task run at the same time in a multi-core processor.



Remember that concurrency and parallelism are NOT the same things. Let's understand more in detail what I mean when I say Concurrency vs. Parallelism.

1. What is Concurrency?

Concurrency is essentially applicable when we talk about a minimum of two tasks or more. When an application is capable of executing two tasks virtually at the same time, we call it a concurrent application. This is why **concurrency is also referenced as virtual parallelism**.

Though, in this case, tasks look like they are running simultaneously, but essentially they MAY not. They take advantage of the CPU time-slicing feature of the operating system where each task runs part of its task and then goes to the waiting state. When the first task is waiting, the CPU is assigned to the second task to complete its part of the task.

The operating system based on the priority of tasks, thus, assigns CPU and other computing resources e.g. memory; turn by turn to all tasks and gives them a chance to complete. To the end-user, it seems that all tasks are running in parallel. This helps by concurrency solve multiple tasks faster.

2. What is Parallelism?

Parallelism does not require two tasks to exist. It, literally, physically runs parts of tasks OR multiple tasks, at the same time using the multi-core infrastructure of the CPU, by assigning one core to each task or sub-task.

Generally, in the case of parallelism, a task is split into subtasks across multiple CPU cores. These subtasks are computed in parallel and each of them represents a partial solution for the given task. By joining these partial solutions, we obtain the final solution. Ideally, solving a task in parallel should result in less wall-clock time than in the case of solving the same task sequentially.

In a nutshell, **in parallelism, at least two threads are running at the same time** which means that parallelism can solve a single task faster. Parallelism requires the hardware with multiple processing units, essentially.

> In a single-core CPU, we may get concurrency but NOT parallelism.

3. Differences between concurrency vs. parallelism

Now let's list down the remarkable differences between concurrency and parallelism.

- Concurrency is when two tasks can start, run, and complete in overlapping time periods. Parallelism is when tasks literally run at the same time, eg. on a multi-core processor.
- Concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations.
- Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.
- An application can be concurrent but not parallel, which means that it processes more than one task at the same time, but no two tasks are executed at the same time instant.
- An application can be parallel but not concurrent, which means that it processes multiple sub-tasks of a task in a multi-core CPU at the same time.
- An application can be neither parallel nor concurrent, which means that it processes all tasks one at a time, sequentially.

- An application can be both parallel and concurrent, which means that it processes multiple tasks concurrently in a multi-core CPU at the same time.

Typically, we measure parallelism efficiency in *latency* (the amount of time needed to complete the task), while the efficiency of concurrency is measured in *throughput* (the number of tasks that we can solve).

That's all about **Concurrency vs. Parallelism**, a very important concept in Java multi-threading concepts.

Happy Learning !!