Concurrency vs. Parallelism

In this lesson, you'll understand the difference between concurrency and parallelism.

WE'LL COVER THE FOLLOWING

A lot of people confuse concurrency with parallelism because they both somewhat imply executing code simultaneously but they are two completely different concepts. This lesson will help you clear up some misconceptions caused by developers using these concepts interchangeably.

"Concurrency is about dealing with lots of things at once.

Parallelism is about doing lots of things at once"

In the previous lesson, we learned about concurrency so let’s discover parallelism now.

What is parallelism?

Parallelism is when we break up a task into subtasks and execute them simultaneously. Each of the subtasks is independent and may or may not be related. In short, we carry out many computations at the *same time* in parallelism.

The multicore processor in your computer is an example of parallelism where parallel tasks are run on multiple cores to solve problems. In this way, parallel computing helps us solve large problems efficiently by using more than one CPU to execute multiple computations at the same time, which saves time in the case of large datasets.

However, parallel programming is hard to achieve as we need to ensure the independence of tasks when it comes to dividing the problem and sharing the data. This is where concurrency comes into play!

Concurrency and Parallelism are not the same but are closely related to each other.

"Concurrency is about structure.

Parallelism is about execution."

When we say that parallelism is about the *execution* of tasks that are independent of each other, we first need to create these independent tasks through concurrency. That is what we do when we *design and structure* a big problem into smaller problems which can be solved independently. Concurrency will ensure that these independent tasks are able to coordinate and synchronize with each other such that we get the correct final result. Therefore concurrency has to be there for parallelism to exist.

Example: Let's Go to Work!

Let's draw a scenario to compare and contrast between concurrency and parallelism:

After waking up in the morning, you have to get ready to go to work. Let's divide this scenario into four steps:

1. Getting ready (Washing your face, changing clothes etc)

2. Preparing breakfast

3. Eating breakfast
4. Going to the office

Sequential Approach

You will first get ready and eventually move on to preparing breakfast for yourself. Then you'll eat your breakfast and leave for the office either by walking, driving or using public transport. Every task has to be in sequence with another task.

Concurrent Approach

Let's now be efficient and plan a better routine. You can try getting ready and preparing your breakfast at the same time. For example, if you have to toast your bread slice or boil water to make tea, you can start with these tasks and then switch to either washing your face or changing your clothes. This will save you time that you spend waiting for the bread to toast or for the water to boil. Note that you are the one coordinating and managing yourself between the two tasks. You are not doing both tasks at the *exact same time* but you have broken down the main tasks into individual independent tasks which allow you to switch back and forth between them. Also, you need signals from the toaster and the boiler that they are done so you can resume the *'Preparing breakfast'* task. This implies that a certain kind of communication needs to exist between the two tasks currently in progress. By synchronizing and communicating between these two tasks, you are using concurrency to solve your problem and making progress on both tasks simultaneously instead of doing them sequentially.

Furthermore, we can also use the concurrent approach for step 3 and step 4. Notice that step 2 is a pre-requisite to step 3. You cannot eat your breakfast if you haven't prepared it. Thus step 3 will have to follow step 2 in our current scenario. The same goes for step 4. You obviously can not go to the office without getting ready and having your breakfast (some people can still manage it but it is a very unhealthy thing to do)!

Coming back to designing our routine: you can make progress on both step 3 as well as step 4 by eating while walking or driving to the office. Obviously, it will be difficult to eat this way but the coordination and management of these will be difficult to eat this way but the coordination and management of these two tasks are totally up to you. This is where you will need to come up with a

concurrent design, i.e., you can either wait until traffic signals to take a bite of your breakfast while driving or you can walk for a certain distance, stop and eat.

In summary, the concurrent approach is all about you designing and structuring your problem in a way that suits you while making the process more efficient.

Parallel Approach

Let's discuss the parallel aspect of the problem now. To make step 1 and step 2 completely parallel, the *'Getting Ready'* and *'Prepare breakfast'* tasks have to run in parallel. Think of how it is possible. As you cannot simultaneously handle this yourself, let's make these tasks completely independent. This can happen if you have a ready-made breakfast delivered to you by a food-delivery service or maybe your roommate was nice enough to prepare yours while making their breakfast. In this way, you only have to worry about yourself getting ready while the breakfast preparation will be going on in parallel.

Regarding using parallelism in step 3 and step 4: if you use public transport you can eat your breakfast while sitting on a bus or train. The bus or train will take you towards your office in parallel to you eating your breakfast.

Concurrent and Parallel

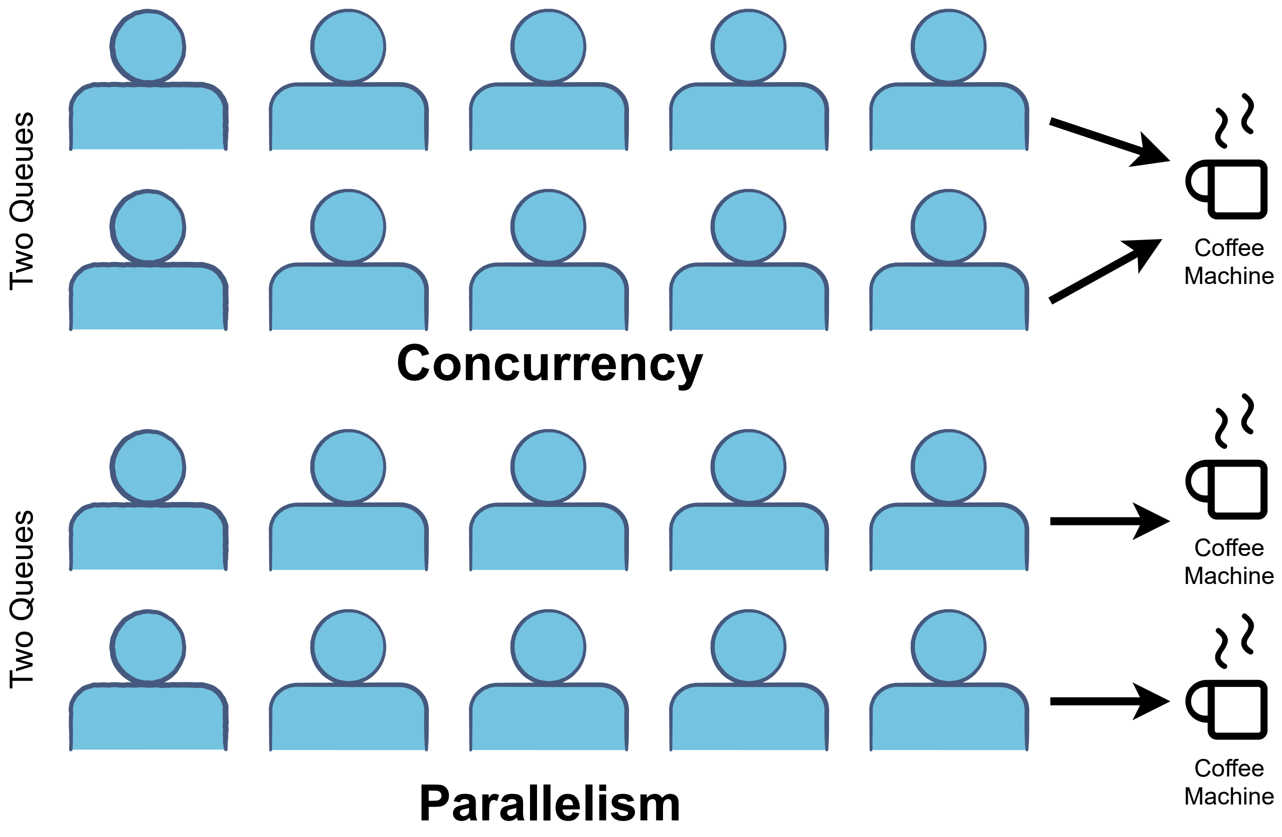
You can also use both approaches to solve your problem. For example, you can handle step 1 and step 2 concurrently while step 3 and step 4 can be executed using the parallel approach.

Hope things make sense and you are with me so far!

Again, let's reiterate that concurrency is about composing a solution to a problem while parallelism is solving the problem by running things in parallel. Remember concurrency does not imply parallelism. It is just a way to structure and design tasks independently so that we can use parallelism to make them efficient.

Example: Coffee Machine

Let's look at another example to make things more clear. In the illustration below, two queues are lined up to get coffee from the coffee machines. Concurrency will decide how the two queues will coordinate and manage themselves to get coffee from one coffee machine. The idea is to add a design which will ensure the independence of subtasks such that if they run in parallel, our problem can be more efficiently solved. On the other hand, in the parallelism approach, each queue has its own coffee machine and the two queues are completely independent of each other.



Concurrency vs. Parallelism

A famous talk on *Concurrency is not Parallelism* was given by Rob Pike who is known for his work on Go. You should definitely watch it for more understanding!

I am hoping that the concept regarding concurrency and parallelism makes more sense now. We'll learn about Communicating Sequential Processes (CSP) in the next lesson.