
What is the difference between concurrency and parallelism?
www.yzokul.com.tr

What is the difference between concurrency and parallelism?



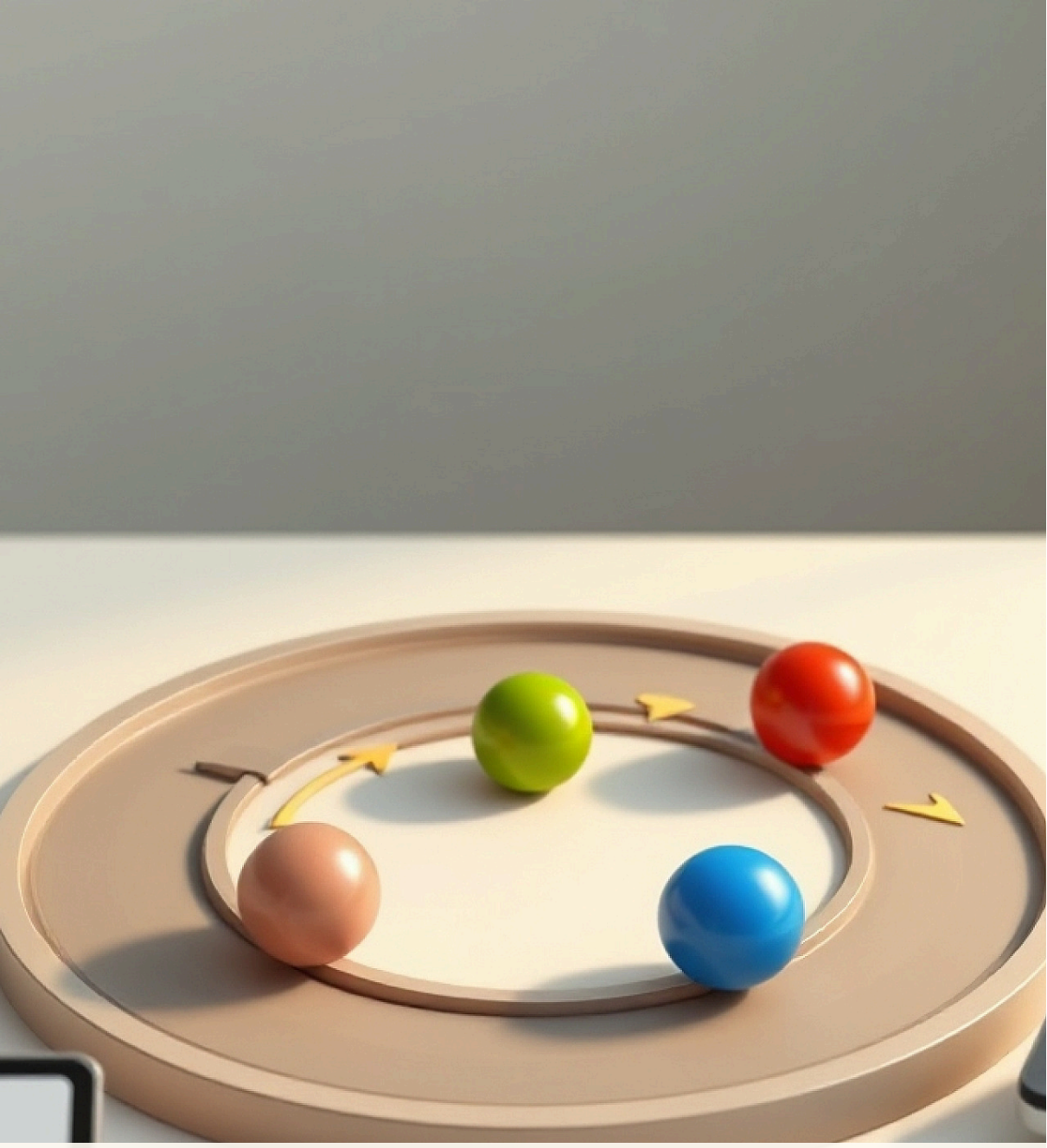
You may have heard the words concurrency and parallelism used as if they mean the same thing, but they describe different ideas. The short answer is: concurrency is managing several tasks' progress together, while parallelism is really running tasks at the same time. Everything else in this book unpacks that sentence with simple pictures and examples. By the end, you will be able to tell the two apart clearly.



A task is simply a piece of work a program has to do, like cooking one dish. It could be reading a file, downloading a web page, or calculating a sum. Tasks are the basic unit we will talk about for the rest of the book. When you hear about managing tasks or running tasks, think of them as separate pieces of work that need to get done.



Picture a single cook in a kitchen with several dishes to prepare. The cook starts one dish, stirs it, and while it simmers switches to chopping vegetables for another dish. The cook is one worker, but several dishes are all making progress together. No two dishes are being worked on at the exact same instant, yet everything moves forward.



Concurrency is the coordinated management of several tasks so they all move forward, even if only one is being worked on at any instant. The single cook switching between dishes is exactly this: one worker managing many tasks. The key is coordination, deciding which task to work on next so that all tasks make progress. Concurrency is about dealing with many things at once by managing them well.



Now picture several cooks in the same kitchen, each working on a different dish at the very same moment. One chops, another stirs, a third watches the oven. This is parallelism: tasks are actually executed at the same time on different processing resources. Each cook is a separate worker, and real simultaneous work is happening. Several dishes progress at once because several workers are active.



Parallelism means real simultaneous execution, which needs more than one processing resource. It is not about switching fast between tasks; it is about two or more things happening at once. For example, two cooks chopping at the same time are truly parallel. If you have only one cook, you cannot have parallelism no matter how fast the cook switches. Parallelism requires multiple workers.



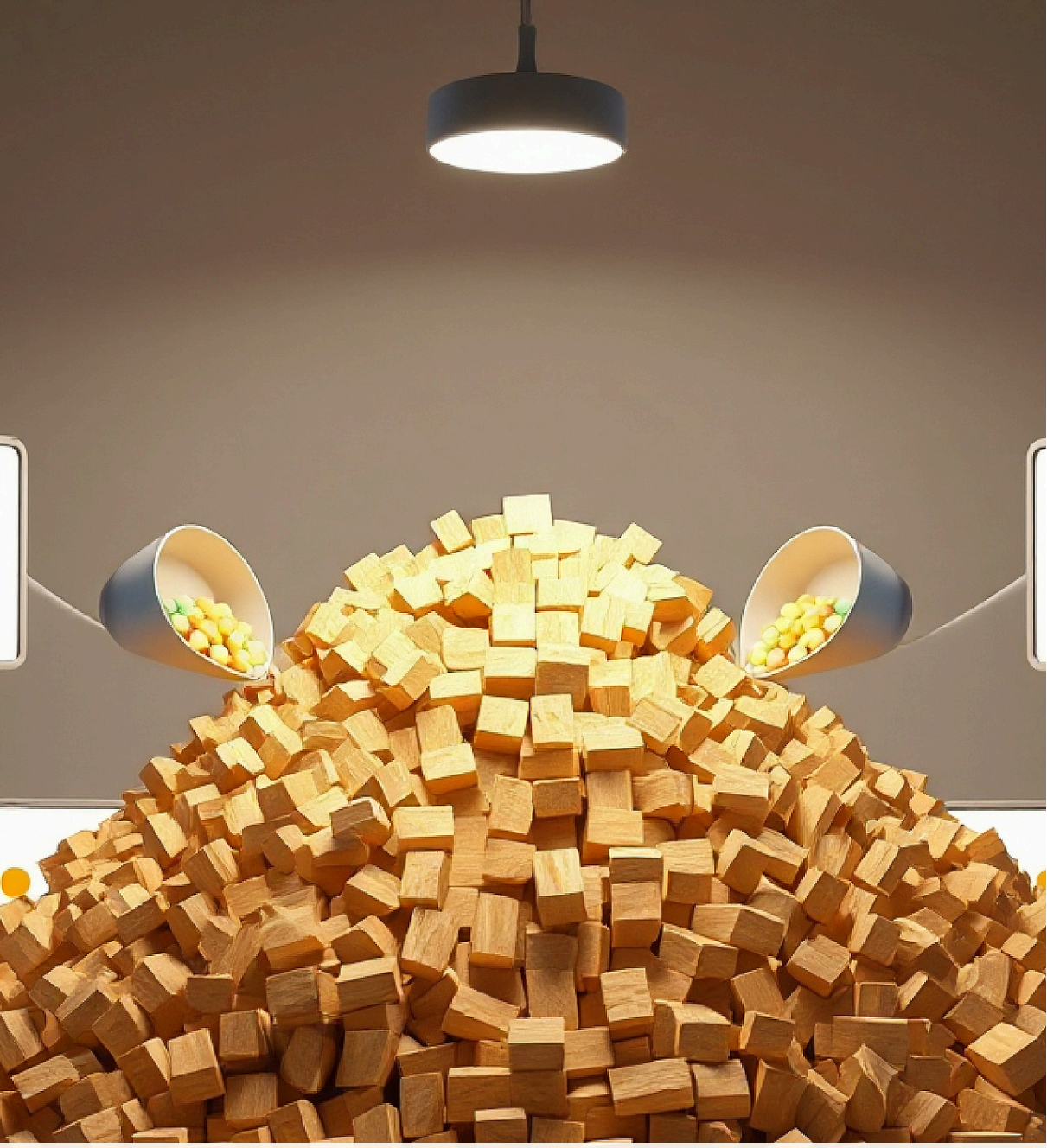
Compare the two pictures directly. One cook switching between dishes is concurrency: many tasks are managed together, but only one is worked on at any instant. Several cooks working side by side is parallelism: tasks run simultaneously on separate resources. Concurrency is about managing many tasks; parallelism is about running them at the same time. The cooking analogy makes the difference easy to see.



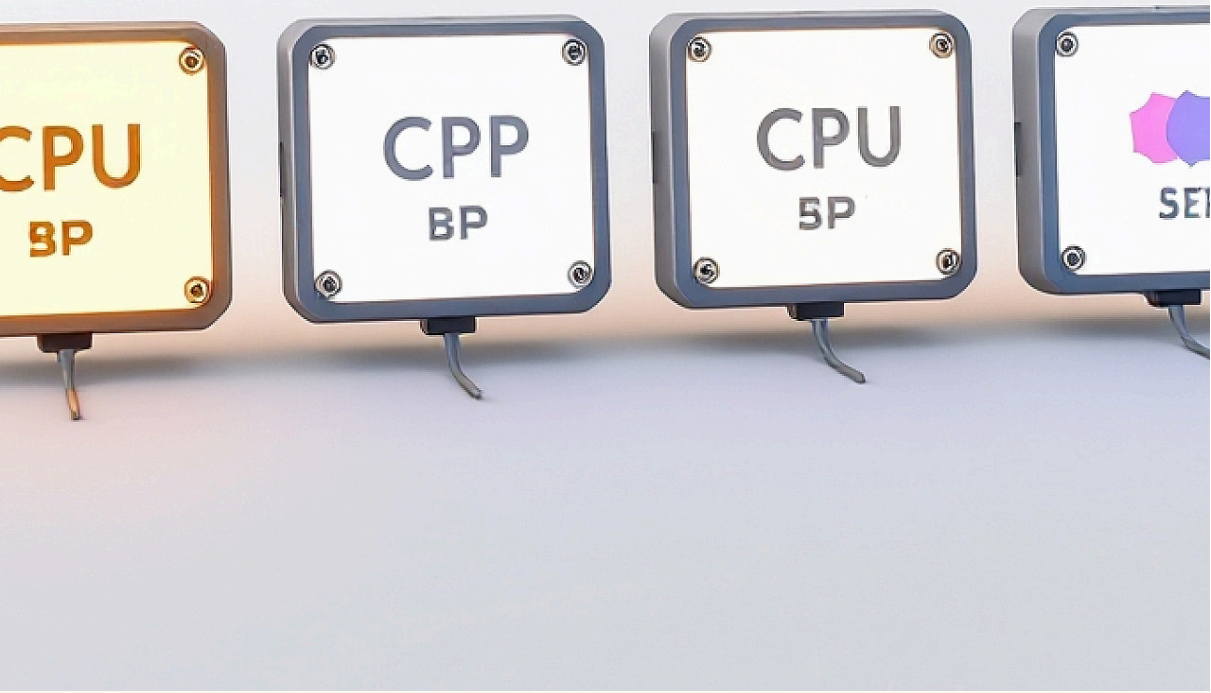
A program can be concurrent without being parallel, just like one cook handling many dishes alone. The tasks make progress together, but nothing happens at the exact same moment. For example, a single-core computer can run a concurrent program by switching between tasks quickly. The tasks appear to advance together, yet only one instruction runs at a time. Concurrency does not require multiple processing resources.



One reason to use these approaches is waiting times. While one task waits, such as a dish simmering or a file downloading, the worker can do something useful elsewhere. This keeps the worker from sitting idle. Concurrency shines here: a single cook can start a long-simmering dish and then chop vegetables for another. Managing tasks well means less time wasted waiting.



The second reason is heavy computation. When there is a lot of computational work, splitting it across several processing resources gets it done faster. This is where parallelism earns its keep. For example, several cooks each chopping a pile of vegetables will finish faster than one cook doing all the chopping. Parallelism uses multiple resources to reduce the time needed for large workloads.



A processing resource is a worker that can do one thing at a time, like a cook or a CPU core. A CPU core is a part of a computer processor that executes instructions. Parallelism needs more than one of these resources to run tasks simultaneously. If you have only one core, you can still manage many tasks concurrently, but you cannot execute them in parallel. More cores allow more true simultaneous work.



Switching between tasks is not free, just as a cook loses a moment when moving between dishes. Each switch takes time and attention, and too much switching can slow things down. Concurrency is useful, but it is still management, not magic. For example, if a cook switches dishes every few seconds, little gets done. Good concurrency minimizes unnecessary switching and keeps progress steady.



A common beginner mistake is treating concurrency and parallelism as identical words. They overlap in everyday talk, but one is about managing progress and the other about simultaneous execution. For example, a single cook handling many dishes is concurrent but not parallel. Several cooks working at once are parallel. Keeping the two ideas separate helps you choose the right approach for a problem.



Another common mistake is thinking that adding workers or parallelism always helps. If the work is mostly waiting, managing tasks well can matter more than having more processing resources. For example, ten cooks waiting for water to boil will not finish faster than one cook who starts other tasks. Match the approach to the work: concurrency for waiting, parallelism for heavy computation.



Practice deciding which situation fits which idea. Many dishes and one cook points to concurrency: tasks are managed together, but only one is worked on at a time. Many cooks at once points to parallelism: tasks run simultaneously on separate resources. Check your understanding against the cooking analogy. If you can tell whether there is one worker or several, you can name the idea.



Remember the answer: concurrency coordinates several tasks making progress together, while parallelism runs tasks truly at the same time on separate resources. A program can be concurrent without being parallel, as one cook can handle many dishes alone. Both approaches help with waiting times or heavy computation. Use concurrency to manage many tasks well, and parallelism to speed up work with multiple workers.