**Multithreading vs Multiprocessing — Simple Comparison**

# What is multithreading?

* **Many threads inside one Python process** sharing the same memory.
* They **take turns** on the CPU so only one thread runs Python code at a time.
* Great for **I/O-bound** work: waiting on files, network, databases.
* **Lightweight** to start; easy to share data (same memory), but you must use locks to avoid races.

# What is multiprocessing?

* **Multiple processes**, each with its **own memory space**.
* Can run on **multiple CPU cores truly in parallel** (no GIL between processes).
* Best for **CPU-bound** work: number crunching, image/video processing, ML feature builds.
* **Heavier** to start; data must be **serialized (pickled)** to move between processes.

 **Multithreading**: One kitchen with **several cooks sharing the same counter**. They can prep ingredients while others wait for the oven (great for I/O). But only one can use the main chopping board at once (GIL).

 **Multiprocessing**: **Several kitchens** with their own counters and ovens. True parallel cooking (great for CPU), but moving dishes between kitchens takes time (serialization).

|  |  |  |
| --- | --- | --- |
| **Topic** | **Multithreading** | **Multiprocessing** |
| What it is | Many threads inside one program | Many separate programs (processes) |
| Memory | Shared (same memory space) | Separate memory per process |
| Best for | I/O-bound work (waiting on files/network/DB) | CPU-bound work (heavy calculations) |
| GIL impact (Python) | Affected → no CPU speed-up for pure Python math | Not affected → true multi-core use |
| Parallelism | Concurrency; not true CPU parallel for Python bytecode | True parallel across CPU cores |
| Data sharing | Easy (same objects) but needs care (locks) | Via queues/pipes; data is copied/pickled |
| Overhead | Lightweight, starts fast, low RAM | Heavier, slower start, more RAM |
| Failure isolation | A bad thread can crash the whole process | Crash is mostly isolated to that process |
| Simplicity of code | Simple sharing, but needs locks to avoid races | Less shared state, but IPC needed |
| Typical Python tools | threading, ThreadPoolExecutor | multiprocessing, ProcessPoolExecutor |

**Takeaway: Use threads for lots of waiting (I/O). Use processes for heavy CPU work.**