MULTIPROCESSING (Deep Explanation)

🔷 What is Multiprocessing?

At the deepest level:

Multiprocessing means running multiple separate Python processes instead of threads.

Each process:

- Has its **own memory**
- Has its **own Python interpreter**
- Has its **own GIL**

So now you get **true parallelism**.

🔷 Why was multiprocessing created?

Because of GIL problem in threading.

In threading:

- Many threads
- But only one can compute at a time

In multiprocessing:

- Many processes
- Each has its own GIL
- They can run truly in parallel on multiple CPU cores.

🔷 Simple Multiprocessing Example

```
import multiprocessing

def task(num):
    print(f"Processing {num}")

p1 = multiprocessing.Process(target=task, args=(1,))
p2 = multiprocessing.Process(target=task, args=(2,))

p1.start()
p2.start()

p1.join()
p2.join()

print(“Done")
```

Here you get:

- 1 main process
- 2 child processes

Total = 3 processes running.

🔷 How this looks in memory

With threading:

```
Process 1:
   Thread A
   Thread B
   Shared Memory
```

With multiprocessing:

```
Process 1: Main
Process 2: Worker 1 (own memory)
Process 3: Worker 2 (own memory)
```

👉 They **do NOT share memory automatically.**

That is both:

- Advantage → no race conditions
- Disadvantage → harder communication

🔷 Real Parallelism Example

Let's compare speed:

Threading (slow for CPU)

```
import threading

def count():
    total = 0
    for i in range(10_000_000):
        total += 1

t1 = threading.Thread(target=count)
t2 = threading.Thread(target=count)

t1.start(); t2.start()
t1.join(); t2.join()
```

Multiprocessing (actually fast)

```
import multiprocessing

def count():
    total = 0
    for i in range(10_000_000):
        total += 1

p1 = multiprocessing.Process(target=count)
p2 = multiprocessing.Process(target=count)

p1.start(); p2.start()
p1.join(); p2.join()
```

Here multiprocessing **really uses two CPU cores.**

🔷 How processes communicate? (Very important)

Since they don’t share memory, we use:

1️⃣ Queue

```
from multiprocessing import Process, Queue

def worker(q):
    q.put("Hello from process")

q = Queue()
p = Process(target=worker, args=(q,))
p.start()
p.join()

print(q.get())
```

Output:

```
Hello from process
```

2️⃣ Shared Value

```
from multiprocessing import Value

x = Value('i', 0)
```

This allows controlled sharing.

🔷 When should you use Multiprocessing?

Use it when:

- You have heavy CPU work
- You are doing:
    - Machine learning
    - Image processing
    - Big data processing
    - Mathematical simulations
    - Large loops

🔥 Final Comparison (Core Cheat Sheet)

Feature	Threading	Async	Multiprocessing
Parallel CPU	❌ No	❌ No	✅ Yes
Best for I/O	✅ Yes	✅ Best	❌ Not ideal
Memory shared	✅ Yes	✅ Yes	❌ No
Complexity	Medium	High	High
GIL issue	Yes	No (single thread)	No

One-line takeaways

- **Threading = many workers, same brain, same memory.**
- **Async = one brain, very smart scheduling.**
- **Multiprocessing = many brains, many memories, true parallelism.**