Great question! The key idea is that **the loop and the executor service work independently**. Let's break it down step by step to understand why the loop does not wait, even when there is a Thread.sleep(2000) inside the task.

**1️⃣ Understanding How executor.submit() Works**

When you call:

executor.submit(() -> {

System.out.println("Task " + tasknumber + " by " + Thread.currentThread().getName());

Thread.sleep(2000); // Simulates long-running task

});

* The **lambda function is passed to the executor**.
* The **executor takes responsibility** for running this task on a separate thread from the thread pool.
* The **main loop does not execute the task directly**—it just submits it.

**2️⃣ What Happens in the Loop?**

Consider this simple version of your code:

ExecutorService executor = Executors.newFixedThreadPool(3);

for (int i = 0; i < 5; i++) { // Main thread runs this loop

final int tasknumber = i;

executor.submit(() -> { // Task is given to the executor

System.out.println("Task " + tasknumber + " by " + Thread.currentThread().getName());

Thread.sleep(2000); // Simulates long task

});

}

executor.shutdown();

**Execution Flow**

1️⃣ The **main thread** runs the for loop.  
2️⃣ For each iteration, it **submits the task to the executor**.  
3️⃣ The executor **immediately takes the task and assigns it to a worker thread** (from the fixed thread pool).  
4️⃣ The main thread **does NOT wait for the task to complete**—it moves to the next loop iteration.  
5️⃣ Meanwhile, the **worker thread starts executing the task in the background** and sleeps for 2 seconds.  
6️⃣ The **loop continues** submitting tasks **even though the previous ones are still running**.

**3️⃣ Why Doesn't the Sleep in the Task Affect the Loop?**

✔ The Thread.sleep(2000); **is inside the task, NOT in the loop itself.**  
✔ The **loop runs on the main thread**, but **tasks run on separate threads**.  
✔ Since tasks run **asynchronously**, the main loop **keeps running and submitting new tasks** without waiting.  
✔ **Each thread that picks a task will sleep for 2 seconds**, but that doesn’t block the loop because it’s running on a different thread.

**4️⃣ What If Sleep Was Inside the Loop Instead?**

If you had:

for (int i = 0; i < 5; i++) {

System.out.println("Submitting task " + i);

Thread.sleep(2000); // THIS WILL BLOCK THE MAIN THREAD

}

☠ The **main thread itself would be sleeping**, meaning each iteration would take 2 seconds, and the total loop execution would take **5 × 2 = 10 seconds**.  
☠ The **tasks would be submitted one by one, with a delay of 2 seconds each**.  
☠ This would make the program **sequential instead of concurrent**.

**5️⃣ Key Takeaways**

✅ **Executor Service submits tasks asynchronously** → The main thread does NOT wait.  
✅ **Loop is just submitting tasks** → Execution happens separately in worker threads.  
✅ **Sleep inside a task affects only that thread**, not the loop or other tasks.  
✅ **If sleep was inside the loop, it would block the main thread.**

Would you like a real-world analogy? 🚀