1. Concurrency Models (Pros & Cons)

1. **Threads**:
   - **Pros**: Great for CPU-heavy tasks and can run tasks in parallel on multiple cores.
   - **Cons**: Risk of issues like race conditions, higher complexity, and context-switching overhead.
2. **Event-Driven**:
   - **Pros**: Efficient for I/O-bound tasks, lightweight, often used in web servers for low latency.
   - **Cons**: Harder to manage as it requires handling states, not ideal for CPU-bound work.
3. **Coroutines**:
   - **Pros**: Efficient for single-threaded, I/O-heavy applications with lower memory overhead.
   - **Cons**: Limited to single-threaded execution, so it’s not suitable for CPU-bound tasks.

2. Concurrency vs Parallelism

- **Concurrency**: Manages multiple tasks by switching between them, not necessarily at the same time. Think of it as task coordination to maximize resource use.
- **Parallelism**: Executes tasks simultaneously, often on multiple processors. It’s about speed-up, dividing tasks to run at the same time for CPU-bound work.

In Short: Concurrency is task management; parallelism is task execution.

3. Blocking vs Non-Blocking Concurrency Algorithms

1. **Blocking**:
   - **Usage**: A task waits until a resource is available, which is simpler but can cause delays.
   - **Example**: Traditional file I/O where tasks wait until resources are released.
   - **Drawback**: Reduces performance, prone to deadlocks.
2. **Non-Blocking**:
   - **Usage**: Tasks don’t wait; they try again later or use a different approach if resources aren’t available.
   - **Example**: Async I/O or lock-free algorithms that retry when resources are unavailable.
   - **Drawback**: More complex to implement but ideal for high-performance, scalable systems.