Comparison of Concurrency Models

1. Thread-Based Concurrency

- **Pros**: Easy to understand and allows parallel execution on multi-core systems.
- **Cons**: High memory usage, performance hit from context switching, and complex synchronization.

2. Event-Driven Concurrency

- **Pros**: Lightweight and ideal for I/O-bound tasks.
- **Cons**: Limited for CPU-bound tasks, can become complex with callbacks.

3. Actor Model

- **Pros**: Reduces need for locks, highly scalable.
- **Cons**: Added latency with message passing, harder to debug.

4. Task-Based Concurrency (Futures/Promises)

- **Pros**: Easier to compose and handle errors.
- **Cons**: Can lead to complex callback management; potential memory overhead.

Concurrency vs. Parallelism

- **Concurrency**: Multiple tasks are handled by interleaving execution, giving the appearance of simultaneous operation. Works well for I/O-bound tasks.
- **Parallelism**: Multiple tasks run simultaneously on different cores or processors, ideal for CPU-bound tasks that benefit from true simultaneous execution.

Blocking vs. Non-Blocking Concurrency Algorithms

1. Blocking Algorithms

- **Description**: Use locks; threads wait until the lock is released.
- **Pros**: Simple and reliable.
- **Cons**: Risk of deadlocks, context switching overhead, and potential priority inversion.

2. Non-Blocking Algorithms

- **Description**: Use atomic operations (e.g., CAS) to allow threads to proceed without waiting.

- **Pros**: Avoids deadlocks, performs better in high-contention.
- **Cons**: More complex, potential livelock, and hardware dependencies.