Concurrency

1. Why concurrent programming?

- Performance
- Throughput
- Utilization of system resources

2. Evolution

- **Single user system**:
  - First systems supported one single activity at a time.
  - Late 1950s - One general-purpose processor and one or more special-purpose processors for input and output operations

- **Multiprocessing systems**: These systems use the CPU while an input/output operation is being performed. Note that programmers cannot control this task explicitly.

- **Multitasking systems**: These systems give the impression that there are several CPUs serving different users at the same time. Programmers cannot control task scheduling. This is done by the operating system.

- **Multithreading systems**: This is becoming very popular with Java. Programmers can split their programs into several threads and schedule them.

- **Multiprocessor systems**:

    - These are systems with several processing elements (PEs) and programmers can concurrently use all these PEs.
    - Single-Instruction Multiple-Data (SIMD) machines:
        - The same instruction goes to all processors, each with different data - e.g., *vector processors*
        - Multiple-Instruction Multiple-Data (MIMD) machines: Independent processors that can be synchronized (unit-level concurrency)