**Lecture 1 - Intro**

**What?**

***concurrent***

there are several tasks in execution at the same moment, that is, task 2 is started before task 1 ends:  
A black and white drawing of a rectangular object

Description automatically generated

***parallel***

(implies at least a degree of concurrency) there are several processing units that work simultaneaously, executing parts of several tasks at the same time.

***distributed***

(implies parallelism) the processing units are spatially distributed.

**Why?**

***optimize resource utilization***

Historically, the first. While a task was performing an I/O operation, the CPU was free to process another task. Also, when a user thinks about what to type next, the CPU is free to handle input from another user.

***increase computing power***

Single-processor systems reach their physical limits, given by the speed of light (300mm/ns) and the minimum size of components. Even single processors have been using parallelism between phases of execution (pipeline).

***integrating local systems***

A user may need to access mostly local resources (data), but may need also some data from other users. For performance (time to access, latency) and security reasons, it is best to have local data local, but we need a mechanism for easily (transparently) accessing remota data also.

***redundancy***

We have multiple units able to perform the same task; when one fails, the others take over. Note that, for software faults (bugs), it is possible that the backup unit has the same bug and fails, too.

**Why not (difficulties)**

***increased complexity***

***race conditions***

What happens if, while executing an operation, some of the state that is relevant for it, is changed by a concurrent operation?

***deadlocks***

Task A waits for task B to do something, while task B waits for task A to do someother thing.

***non-determinism***

The result of a computation depends on the order of completion of concurrent tasks, which in turn may depend on external factors.

***lack of global state; lack of universal chronology (distributed systems only)***

A process can read a local variable, but cannot read a remote variable (that resides in the memory of another processor); it can only request the value to be sent, and, by the time the value arrives, the original value may have changed

**Clasification**

**Flynn taxonomy**

***SISD (single instruction, single data)***

***SIMD (single instruction, multiple data)***

***MISD (multiple instruction, single data)***

***MIMD (multiple instruction, multiple data)***

**Shared memory vs message passing**

**Shared memory**

***SMP (symmetrical multi-processing)***

identical processors (cores) accessing the same main memory

***AMP (asymmetrical multi-processing)***

like SMP, but processors have different capabilities (for example, only one can request I/O)

***NUMA (non-uniform memory access)***

each processor has a local memory but can also access a main memory and/or the local memory of the other processors.

**Message passing**

***cluster***

many computers packed together, maybe linked in a special topology (star, ring, tree, hyper-cube, etc)

***grid***

multiple computers, maybe of different types and characteristics, networked together and with a middle-ware that allows treating them as a single system.