Python Concurrency / Parallel Computing / Distributed Computing

A picture containing table

Description automatically generated

**Parallel computing architecture**

SIMD = Single Instruction, Multiple Data

* All processors execute the same instruction, but each operate on different data element
* Same algorithm, but on different data (e.g. chop carrot, chop celery)
* E.g. image processing, same handful of operations on many different images – GPU uses SIMD instruction

MISD = Multiple Instruction, Single Data – this is not used b/c not practical – both chopping / peeling same carrot at the same time

**MIMD = Multiple Instruction, Multiple Data** (MIMD)

* Every processing unit can execute a different series of instructions, on different set of data – (peel carrot, chop celery at the same time)
* Most commonly use. E.g. multicore PC, network cluster computers

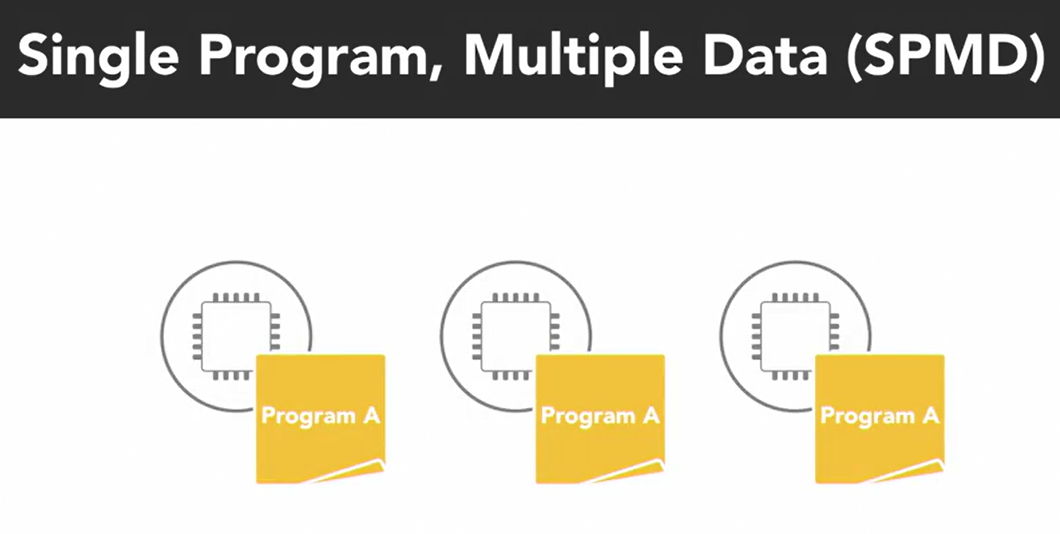
Diagram

Description automatically generated with medium confidence Diagram

Description automatically generated

MIMD is divided into subcategories:

* **SPMD (Single Program, Multiple Data)** – most common style
  + Multiple processing unit execute the same program, each use different data
    - not SIMD because although execute the **same program**, may **not execute the same instruction at the same time**
  + Same recipe, each processor handle a different part of the program.
* MPMD (Multiple Program, Multiple Data) – not common, for functional decomposition
  + Processors execute different independent programs at the same time on different data
  + One processing node is selected as host or manager which runs one program that farms out data to the other nodes running a second program
    - The other nodes do their work and return their results to the manager

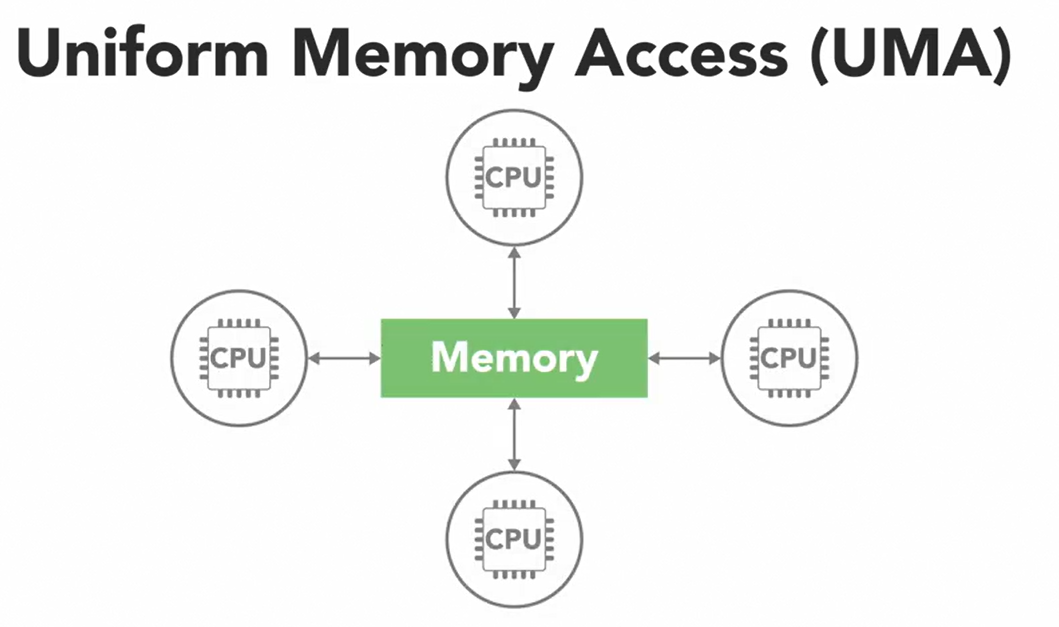
Diagram

Description automatically generated

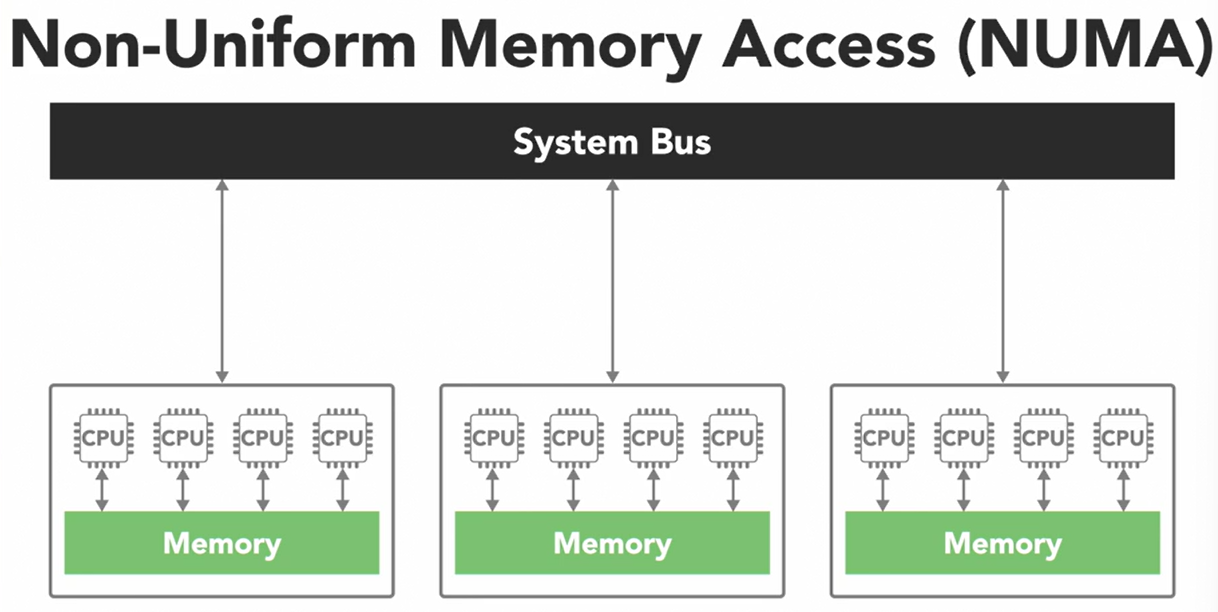
**Memory Organization (Shared vs. Distributed)**

**Shared memory**:

* All processors access the same memory with same global address space
  + If one processor changes memory, all other processor will see that change
  + All processors see EVERYTHING that happens in the shared memory space
* Subcategories: (based on how to processors are connected to memory and how quickly they can access it)
  + **Uniform Memory Access (UMA)**:
    - All processors have equal access to memory (i.e. equally fast)
    - **Symmetric Multiprocessing System (SMP)** – most common type
      * Processors connected to a single shared memory through System Bus
      * Note Cache coherency (handled by processor hardware)
  + Non-Uniform Memory Access (NUMA):
    - Physically connection multiple SMP systems together
    - Access is Non-uniform b/c some processors have quicker access to memory
    - Overall every processor can still see everything in memory
    - Does not scale well, adding more processors can increase traffic on shared memory bus, + maintain cache coherency, programmer has to synchronize memory accesses to ensure correct behaviour

Timeline

Description automatically generated



**Distributed** memory:

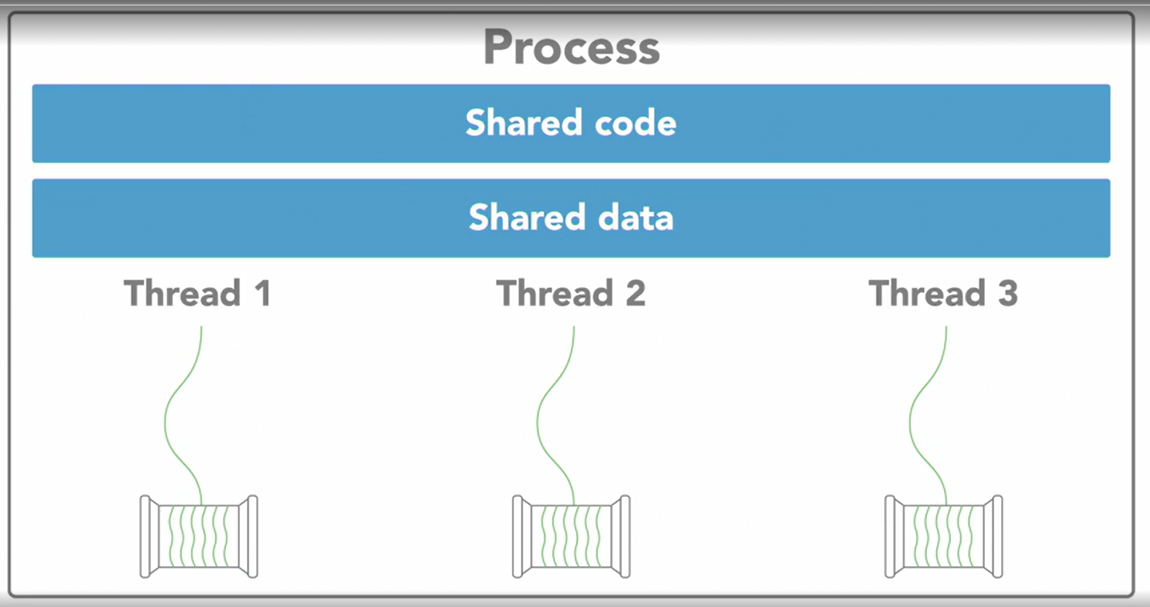
* Each processor has its own local memory and address space, so the concept of global memory space does not exist
* All processors connected through ethernet, processors operates independently, changes are not automatically reflected in memory of other processors
  + it’s up to the programmer to define how and when data is connected between nodes – communication is tough
  + advantage – its scalable

Timeline

Description automatically generated

Thread vs Process

* **Process**
  + an instance of a program executing:
    - code, data, state information
    - independent and has its own separate address space and memory
* **Thread**
  + Within each process, there are small sub-elements called threads
  + independent path of execution
    - e.g. a different sequence of instruction,
    - Threads are the basic units that the operating system manages, and it allocates time on the processor
  + **Threads** that belong to the same process **share** the process’ address space which give them access to the **same resources** and **memory**, including program’s executable **code** and **data**.
* Program to use …
  + Multiple Processes: working together towards a common goal
  + Multiple threads within a single process



**Concurrency** vs. **Parallel** execution:

* Concurrency execution: (structure of a program)
  + Ability of a program / algorithm to be broken into parts that can be executed out of order, without affecting result
  + Its about how a program is structured and the composition of independently executing processes
  + Decompose a program into concurrent tasks (order independent)
* Parallel execution: (hardware, simultaneous execution, actually doing multiple things at once)
  + requires Parallel hardware (multi-core, GPU, computer cluster)

