ie
SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

OPERATING SYSTEMS & PARALLEL COMPUTING

Parallel Computing

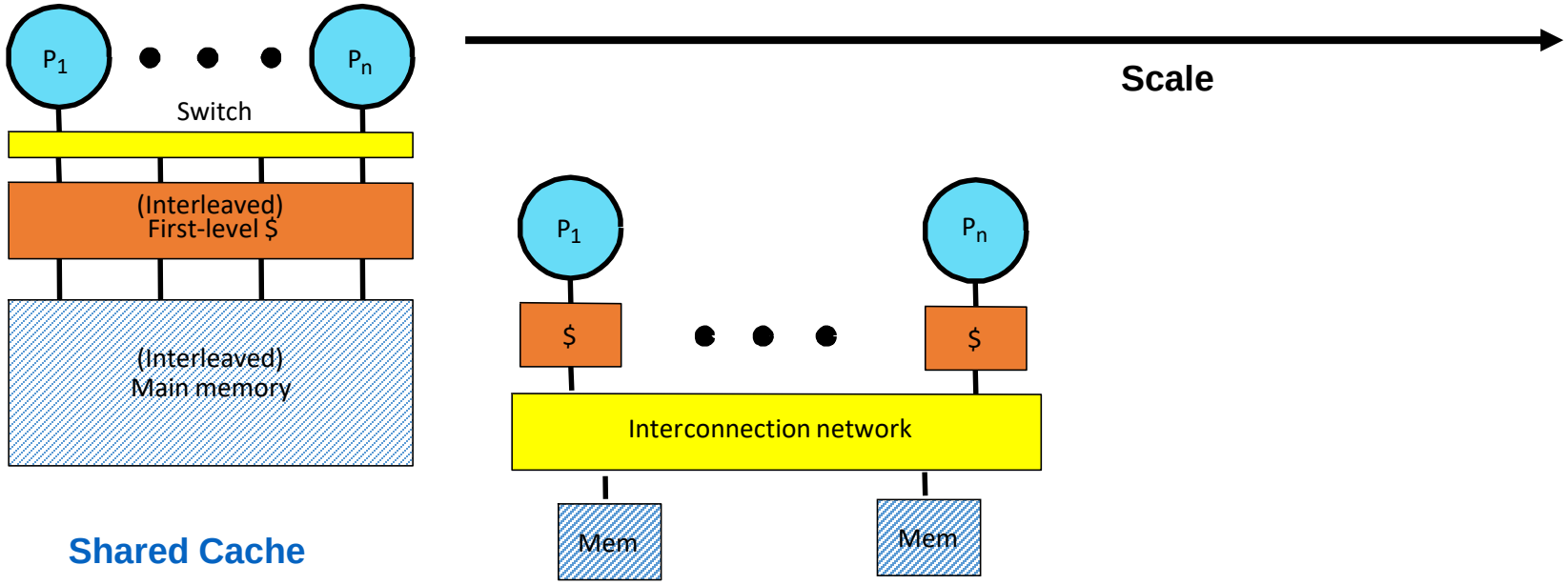
Examples with Python

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Parallel Computer Memory Architectures

- Shared Memory.
- Distributed Memory.
- Hybrid Distributed-Shared Memory.

Natural Extensions of Memory System

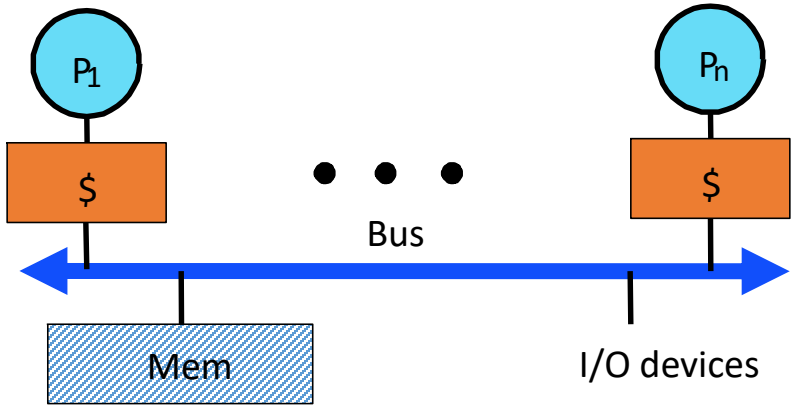


Shared Cache

Centralized Memory
Dance Hall, UMA

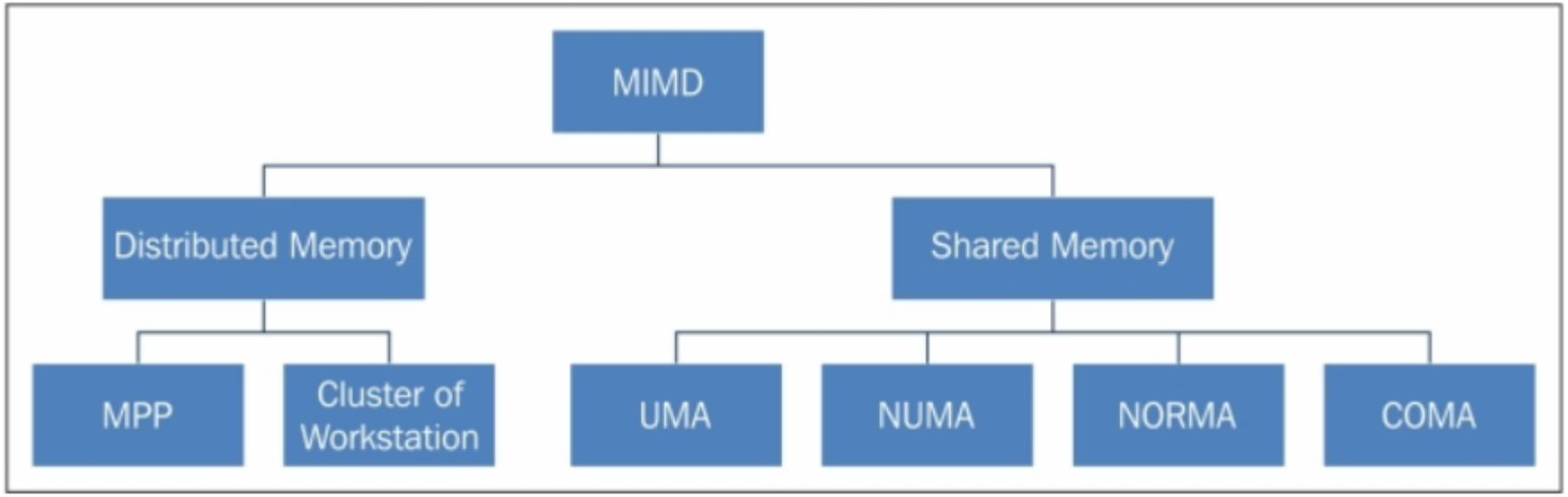
Distributed Memory (NUMA)

Bus-Based Symmetric Shared Memory



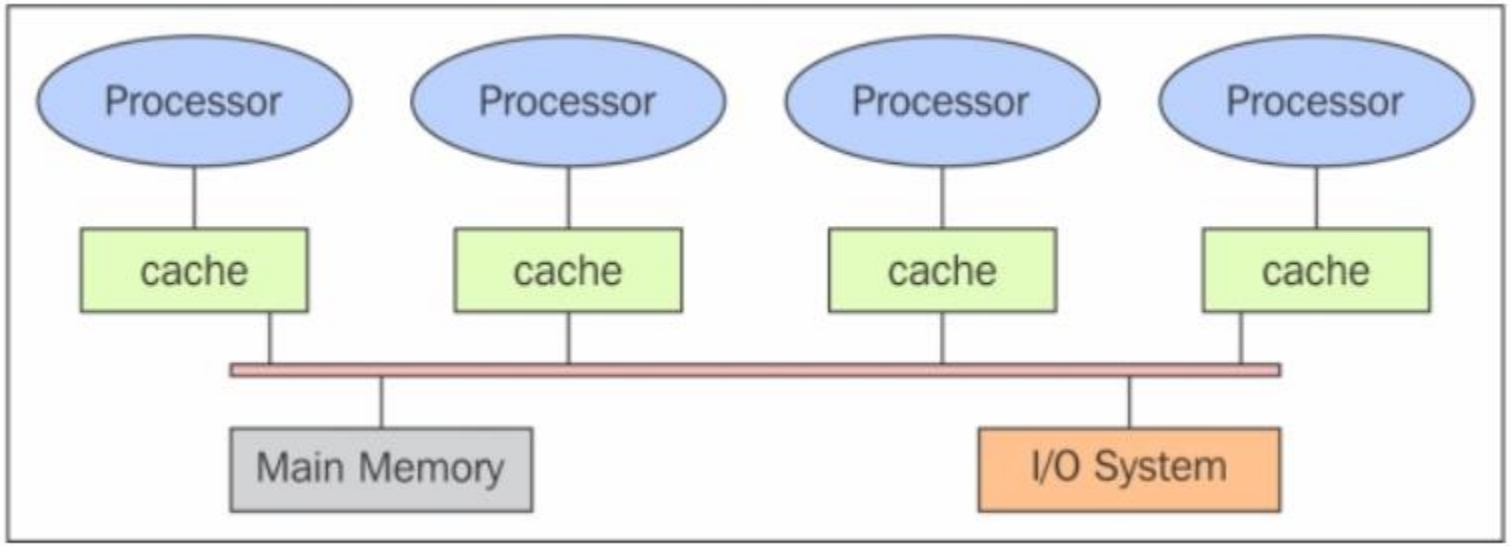
- Still an important architecture – even on chip (until very recently)
  - Building blocks for larger systems; arriving to desktop
- Attractive as throughput servers and for parallel programs
  - Fine-grain resource sharing
  - Uniform access via loads/stores
  - Automatic data movement and coherent replication in caches
  - Cheap and powerful extension
- Normal uniprocessor mechanisms to access data
  - Key is extension of memory hierarchy to support multiple processors

MIMD Architecture

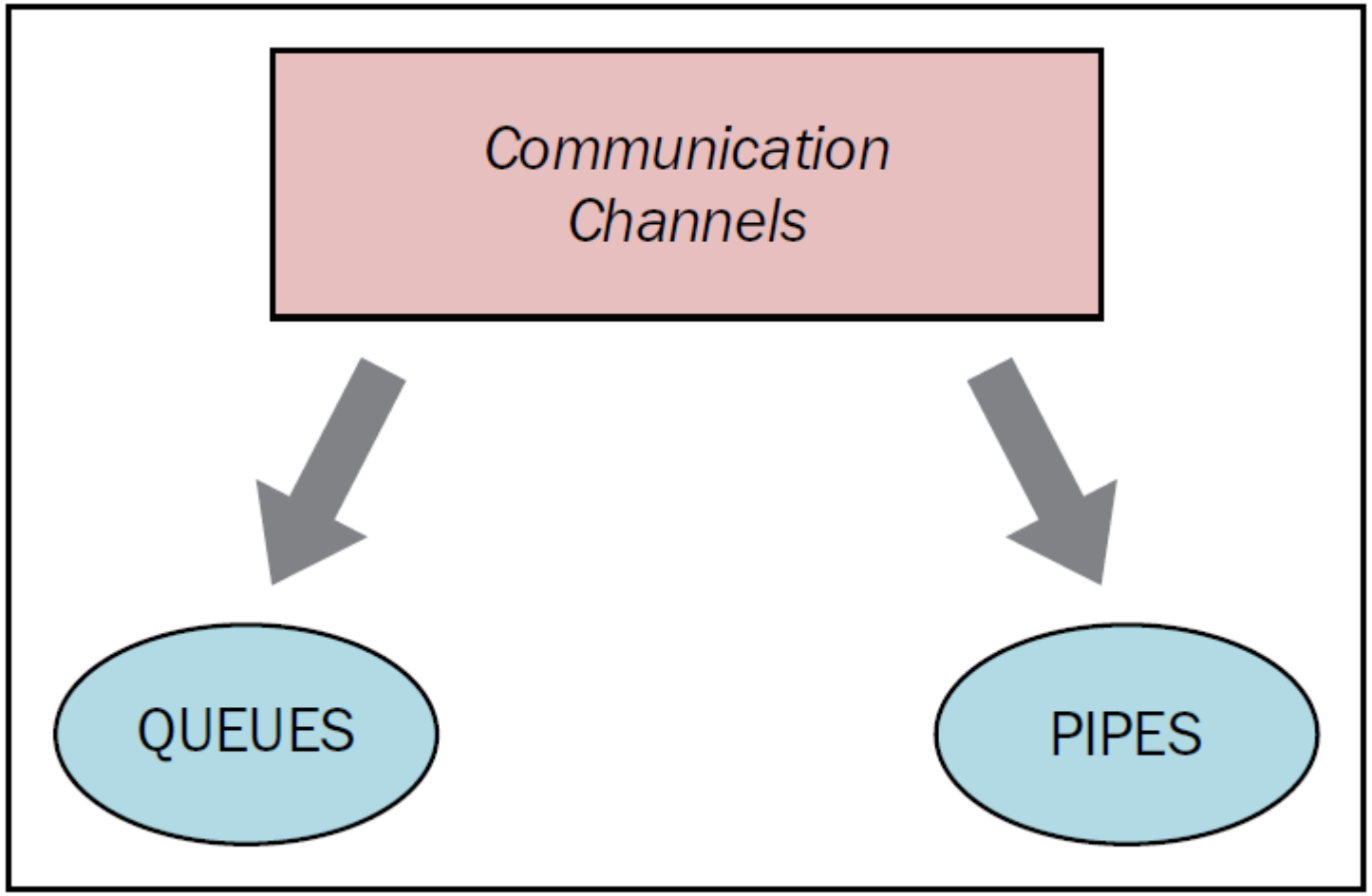


- **Uniform memory access** (**UMA**): The fundamental characteristic of this system is the access time to the memory that is constant for each processor and for any area of memory. For this reason, these systems are also called as **symmetric multiprocessor** (**SMP**). They are relatively simple to implement, but not very scalable; the programmer is responsible for the management of the synchronization by inserting appropriate controls, semaphores, locks, and so on in the program that manages resources.
- **Non-uniform memory access** (**NUMA**): These architectures divide the memory area into a high-speed access area that is assigned to each processor and a common area for the data exchange, with slower access. These systems are also called as **Distributed Shared Memory Systems** (**DSM**). They are very scalable, but complex to develop.
- **No remote memory access** (**NORMA**): The memory is physically distributed among the processors (local memory). All local memories are private and can only access the local processor. The communication between the processors is through a communication protocol used for exchange of messages, the message-passing protocol.
- **Cache only memory access** (**COMA**): These systems are equipped with only cache memories. While analyzing NUMA architectures, it was noticed that these architectures kept the local copies of the data in the cache and that these data were stored as duplication in the main memory. This architecture removes duplicates and keeps only the cache memories, the memory is physically distributed among the processors (local memory). All local memories are private and can only access the local processor. The communication between the processors is through a communication protocol for exchange of messages, the message-passing protocol.

Shared Memory Architecture

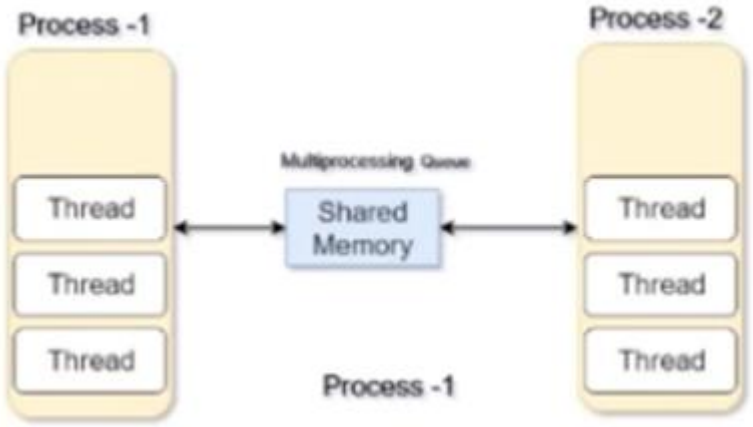


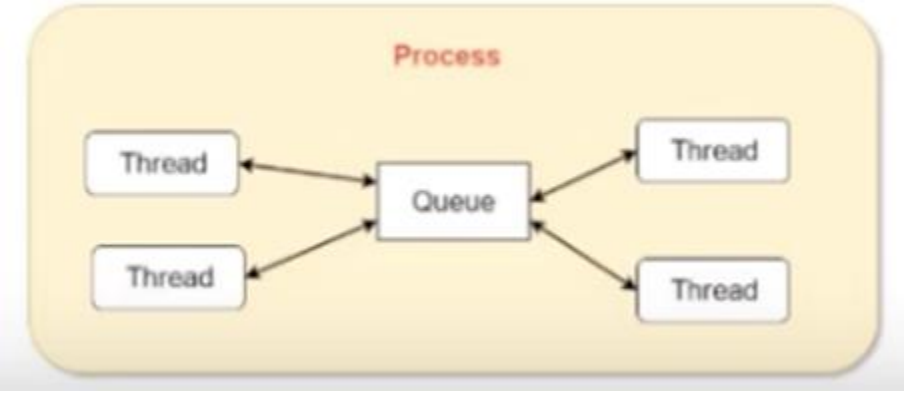
How to Exchange information between processes?



Shared Memory with Python

Multiprocessing Queue	Queue Module
import multiprocessing	import queue
multiprocessing.Queue()	q = queue.Queue()
Lives in shared memory.	Lives in in-process Memory.
Used to share data between process	Used to share data between threads.





Examples to test Queue & Pipe in Python

- - Using Queue Communication
  - https://github.com/operard/opsys_parallel/blob/master/parallel_computing/session18/communicating_with_queue.py

- Using Pipe Communication
  - https://github.com/operard/opsys_parallel/blob/master/parallel_computing/session18/communicating_with_pipe.py

Parallelization Techniques: OpenMP

- OpenMP is sort of an HPC standard for shared memory programming
- OpenMP version 4.5 released in 2015 and includes accelerator
- support as an advanced feature
- API for thread-based parallelism
- Explicit programming model, compiler interprets directives
- Based on a combination of compiler directives, library routines, and
- environment variables
- Uses the fork-join model of parallel execution
- Available in most Fortran and C compilers

OpenMP Goals

- Standardization: standard among all shared memory architectures and hardware platforms
- Lean: simple and limited set of compiler directives for shared memory programming. Often significant performance gains using just 4-6 directives in complex applicaCons.
- Ease of use: supports incremental parallelization of a serial program, not an all-or-nothing approach.
- Portability: supports Fortran, C, and C++

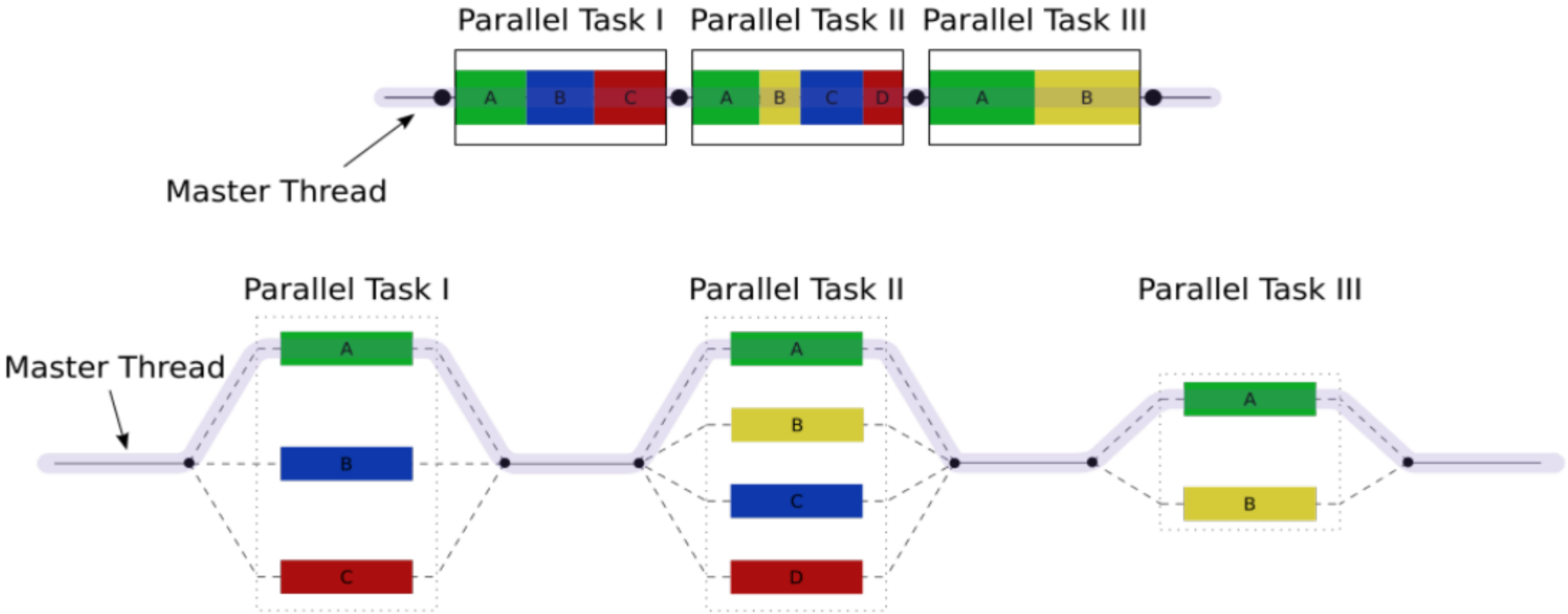
OpenMP Building Blocks

- Compiler Directives (embedded in code)
  - Parallel regions (PARALLEL)
  - Parallel loops (PARALLEL DO)
  - Parallel workshare (PARALLEL WORKSHARE)
  - Parallel sections (PARALLEL SECTIONS)
  - Parallel tasks (PARALLEL TASK)
  - Serial sections (SINGLE)
  - Synchronization (BARRIER, CRITICAL, ATOMIC, …)
  - Data structures (PRIVATE, SHARED, REDUCTION)
- Run-time library routines (called in code)
  - OMP_SET_NUM_THREADS
  - OMP_GET_NUM_THREADS
- UNIX Environment Variables (set before program execution)
  - OMP_NUM_THREADS

Fork-Join Model

- Parallel execution is achieved by generating threads which are executed in parallel
- Master thread executes in serial until the first parallel region is encountered
- Fork – The master thread created a team of threads which are executed in parallel
- Join – When the team members complete the work, they synchronize and terminate. The master thread continues sequentially.

Fork Join Model

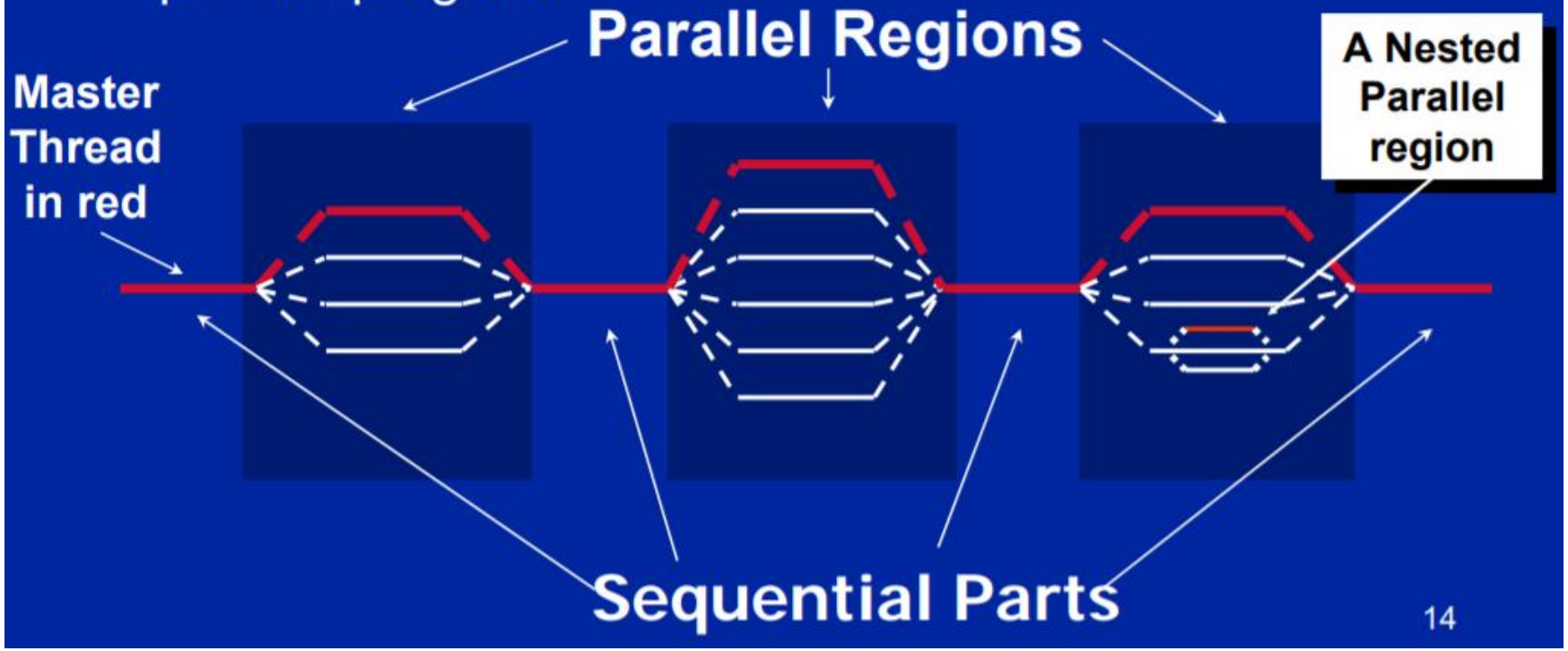


ie
SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

OpenMP Programming Model:

Fork-Join Parallelism:

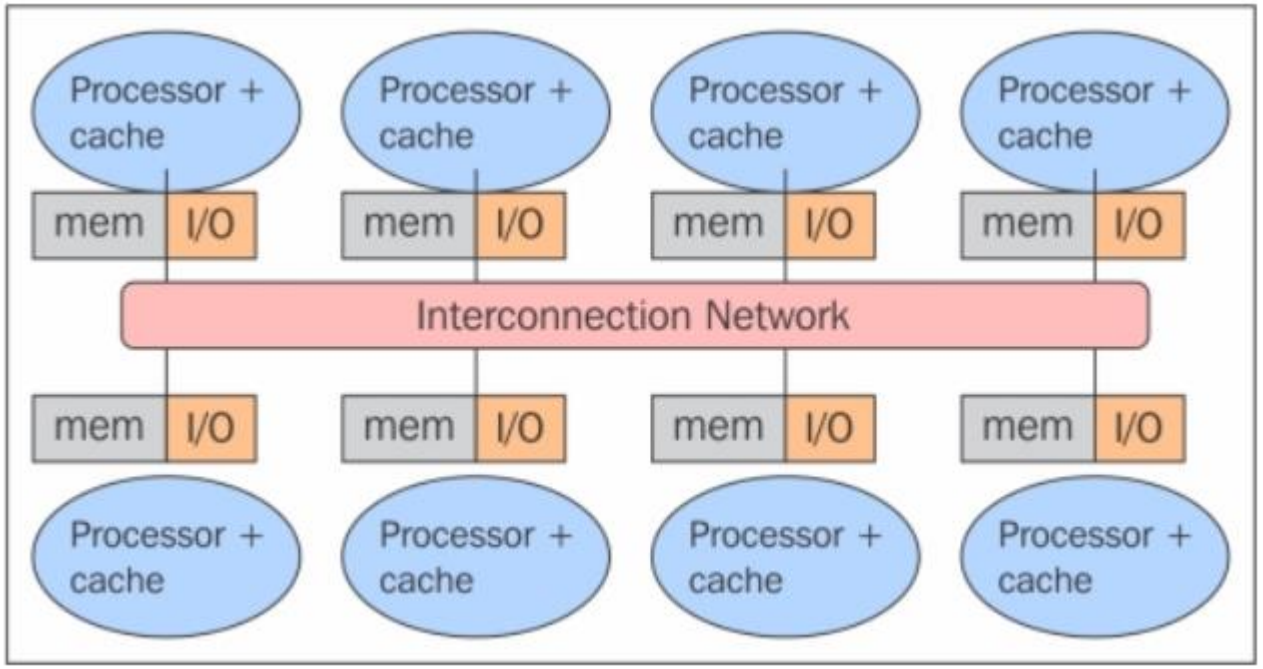
- Master thread spawns a team of threads as needed.
- Parallelism added incrementally until performance goals are met: i.e. the sequential program evolves into a parallel program.



OpenMP Overview

- How do threads interact?
- OpenMP is a multi-threading, shared address model.
  - Threads communicate by sharing variables.
- Unintended sharing of data causes race conditions:
  - race condition: when the program's outcome changes as the threads are scheduled differently.
- To control race conditions:
  - Use synchronization to protect data conflicts.
- Synchronization is expensive so:
  - Change how data is accessed to minimize the need for synchronization.

Distributed Memory



- Memory is physically distributed between processors; each local memory is directly accessible only by its processor.
- Synchronization is achieved by moving data (even if it's just the message itself) between processors (communication).
- The subdivision of data in the local memories affects the performance of the machine—it is essential to make a subdivision accurate, so as to minimize the communication between the CPUs. In addition to this, the processor that coordinates these operations of decomposition and composition must effectively communicate with the processors that operate on the individual parts of data structures.
- The message-passing protocol is used so that the CPU's can communicate with each other through the exchange of data packets. The messages are discrete units of information; in the sense that they have a well-defined identity, so it is always possible to distinguish them from each other.

Message Passing

- Communication on distributed memory systems are a significant aspect of performance and correctness
- Messages are relatively slow, with startup times (latency) taking thousands of cycles
- Once message passing has started, the additional time per byte (bandwidth) is relatively small

Performance on Gartner

- Intel Xeon E5-2683 Processor (Haswell)
- Processor speed: 2,000 cycles per microsecond (µsec)
- 16 FLOPs/cycle: 32,000 FLOPs per µsec
- MPI message latency = ~2.5 µsec or 80,000 FLOPs
- MPI message bandwidth = ~7,000 bytes/µsec = 4.57 FLOPs/byte

ie
SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Reducing Latency

- Reduce the number of messages by mapping communicating entities onto the same processor
- Combine messages having the same sender and destination
- If processor A has data needed by processor B, have A send it to B, rather than waiting for B to request it. Processor A should send as soon as the data is ready, processor B should read it as late as possible to increase the probability that the data has arrived

Other issues with Message-Passing

- Network Congestion
- Deadlock
  - Blocking: a processor cannot proceed until the message is finished
  - With blocking communication, you may reach a point where no processor can proceed
  - Non-blocking communication: easiest way to prevent deadlock; processors can send and proceed before receive is finished

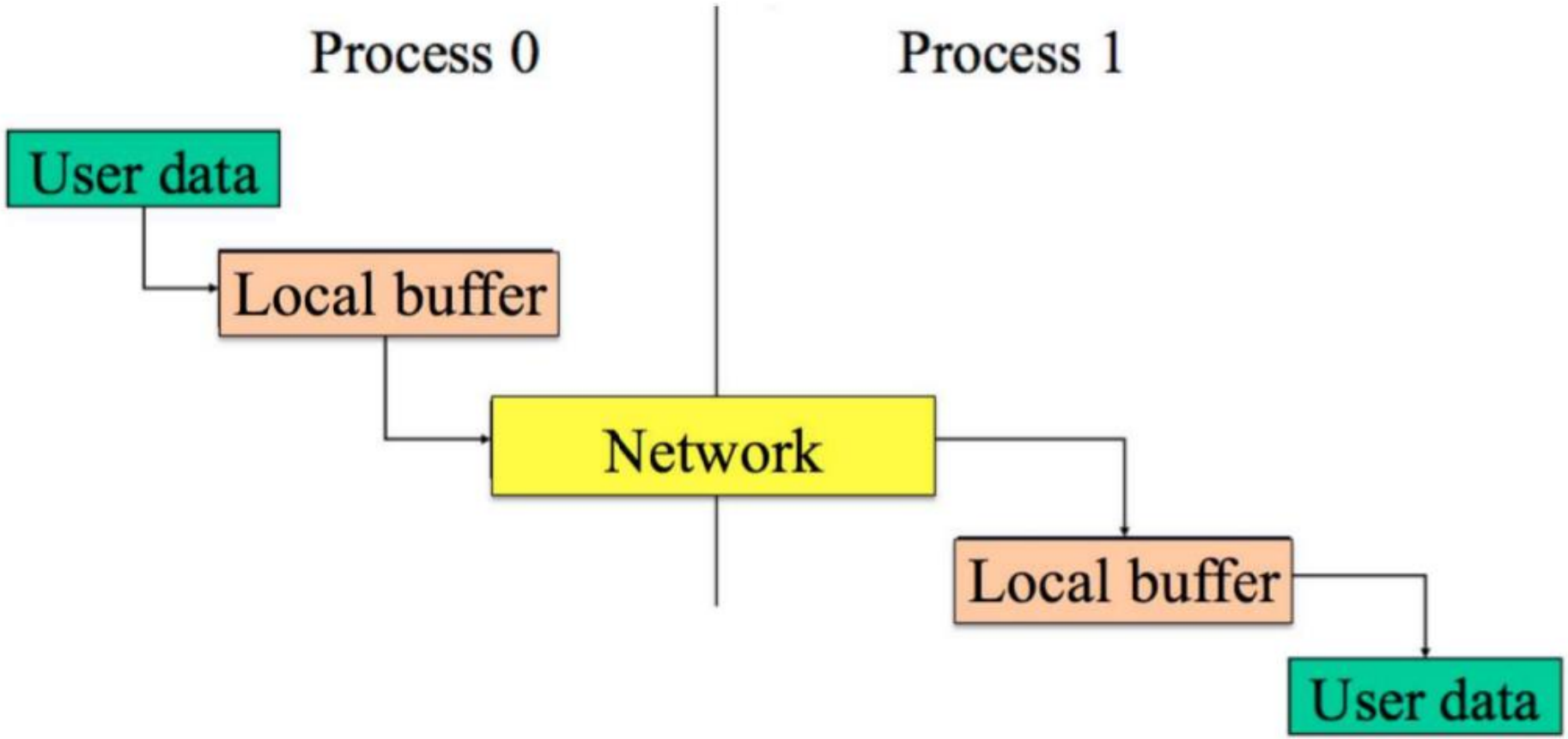
Message Passing Interface (MPI)

- International Standard
- MPI 1.0 released in 1994
- Current version is 3.1 (June 2015)
- MPI 4.0 standard in the works
- Available on all parallel systems
- Interfaces in C/C++ and Fortran
- Bindings for MATLAB, Python, R, and Java
- Works on both distributed memory and shared memory hardware
- Hundreds of functions, but you will need ~6-10 to get started

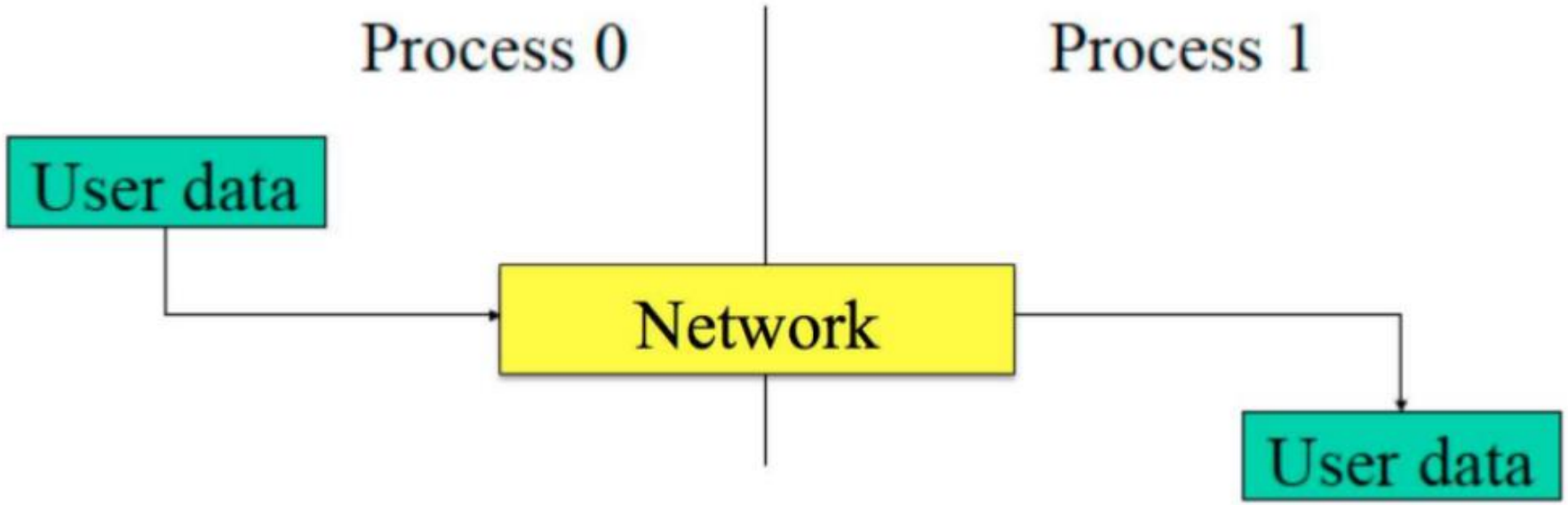
MPI Basics

- Two common MPI functions:
  - MPI_SEND() - to send a message
  - MPI_RECV() - to receive a message
- Function like write and read statements
- Both are blocking operations
- However, a system buffer is used that allows small messages to be non-blocking, but large messages will be blocking
- The system buffer is based on the MPI implementation, not the standard
- Blocking communication may lead to deadlocks

Small Messages



Large Messages



ie
SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

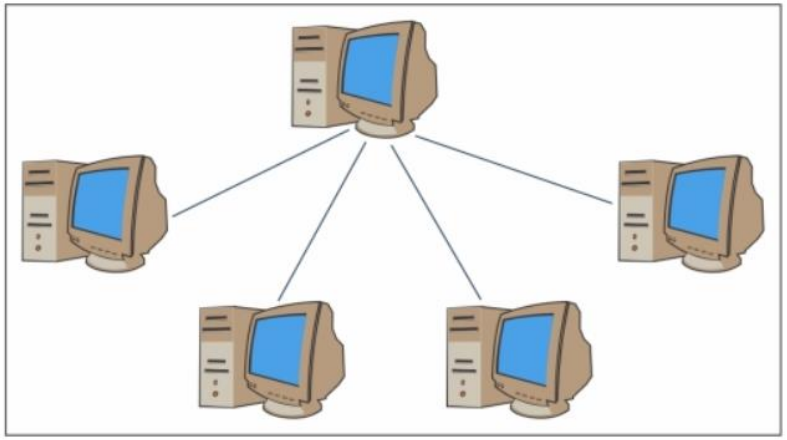
Non-Blocking Communication

- Non-Blocking operations
  - MPI_ISEND()
  - MPI_IRECV()
  - MPI_WAIT()
- The user can check for data at a later stage in the program without waiting
  - MPI_TEST()
- Non-blocking operations will perform better than blocking operations
- Possible to overlap communication with computation

MPI Synchronization

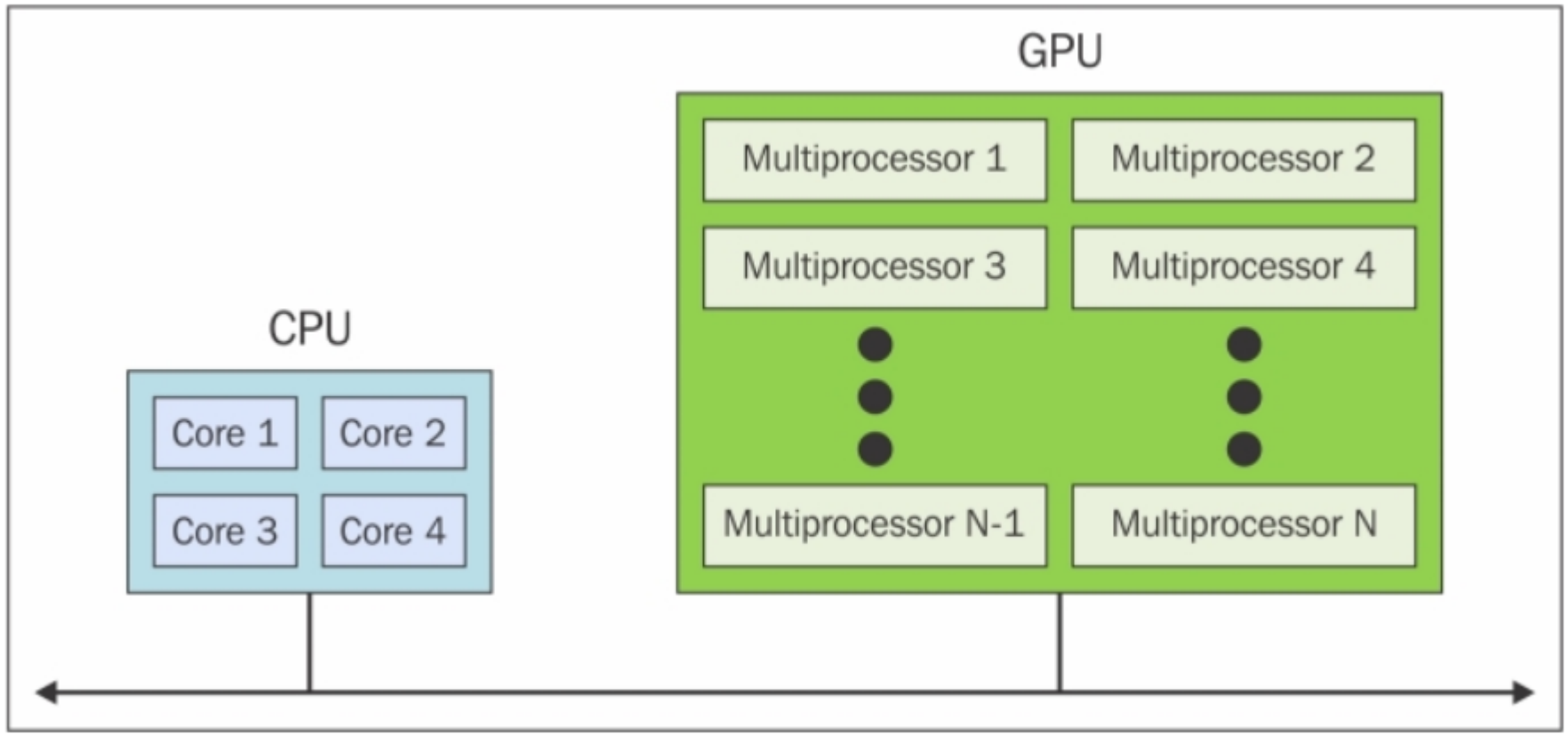
- Implicit synchronization
  - Blocking communication
  - Collective communication
- Explicit synchronization
  - MPI_Wait
  - MPI_Waitany
  - MPI_Barrier
- Remember, synchronization can hinder performance

Massively Parallel Processing



- Cluster of workstations
  - **The fail-over cluster**: In this, the node's activity is continuously monitored, and when one stops working, another machine takes over the charge of those activities. The aim is to ensure a continuous service due to the redundancy of the architecture.
  - **The load balancing cluster**: In this system, a job request is sent to the node that has less activity. This ensures that less time is taken to complete the process.
  - **The high-performance computing cluster**: In this, each node is configured to provide extremely high performance. The process is also divided in multiple jobs on multiple nodes. The jobs are parallelized and will be distributed to different machines.

Heterogeneous architecture



SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY