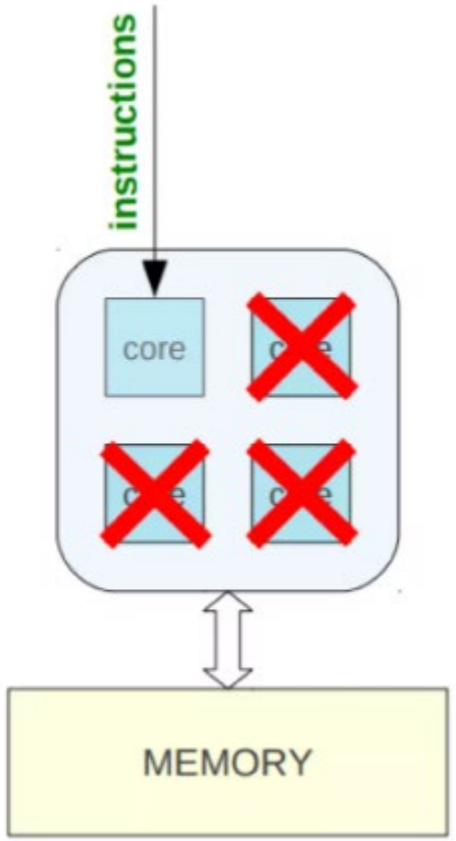
Parallel Programming with OPENMP & MPI

OpenMp Motivation

When you run sequential program

- Instructions executed on 1 core
- Other cores are idle

Waste of available resources. We want all cores to be used to execute program.



OpenMp Motivation

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
//Do this part in parallel

  printf("Hello World");

  return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main()
{
 omp_set_num_threads(16);
//Do this part in parallel
 #pragma omp parallel
 {
   //structured block of code
   printf("Hello World");
 }
  return 0;
}
```

OpenMp Overview

- Collection of compiler directives and library functions for creating parallel programs for shared-memory computers.
- The "MP" in OpenMP stands for "multi-processing"(shared-memory parallel computing)
- Combined with C, C++, or Fortran to create a multithreading programming language, in which all processes are assumed to share a single address space.
- Based on the fork / join programming model: all programs start as a single (master) thread, fork additional threads where parallelism is desired (the parallel region), then join back together.
- Version 1.0 with fortran in 1997, supporting C & C++ there

OpenMp Goals

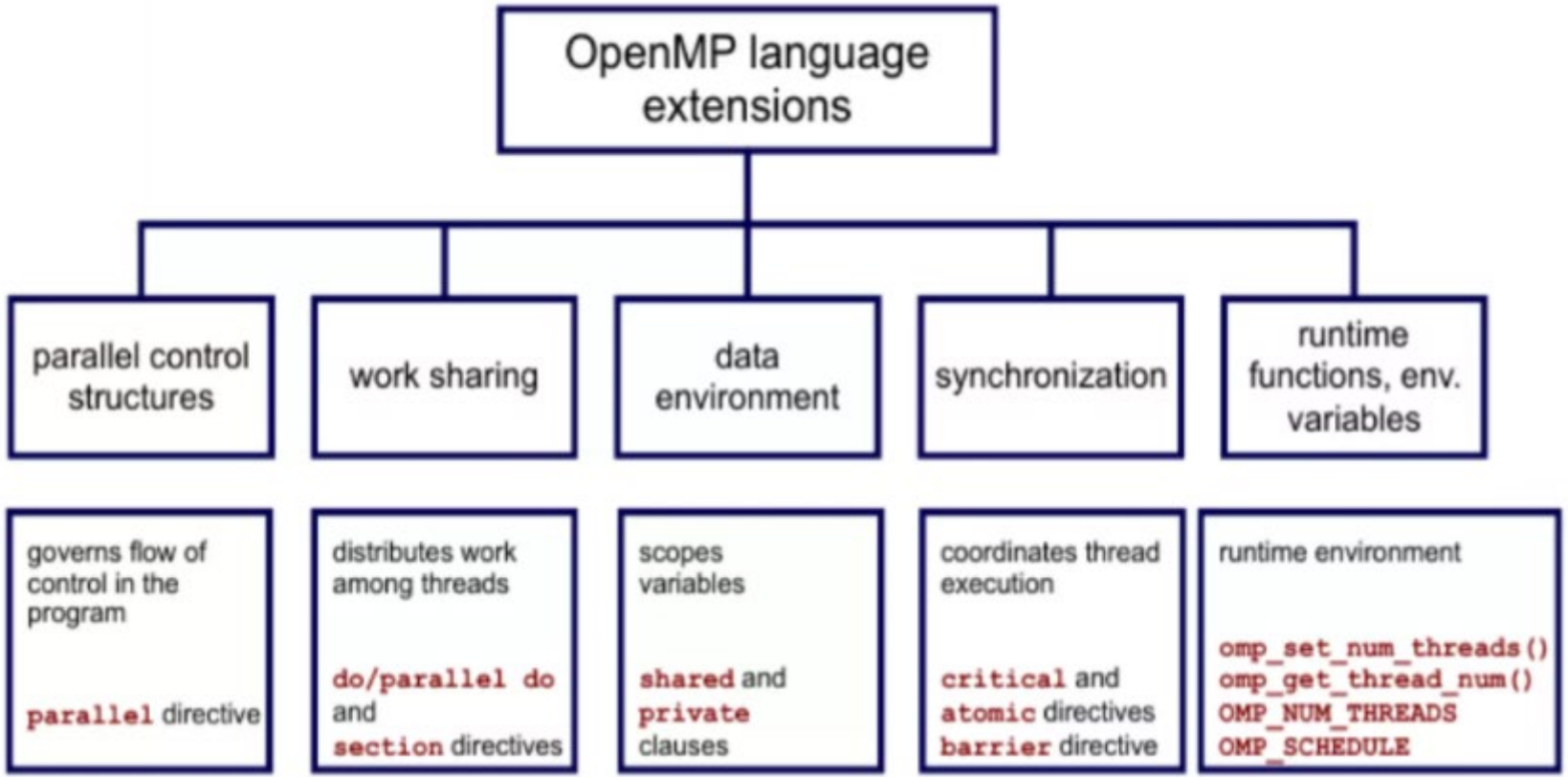
Standardization: Provide a standard among a variety of shared memory architectures/platforms

Lean and Mean: Establish a simple and limited set of directives for programming shared memory machines. Significant parallelism can be implemented by using just 3 or 4 directives.

Ease of Use: Provide capability to incrementally parallelize a serial program. Provide the capability to implement both coarse-grain and fine-grain parallelism

Portability: Supports Fortran (77, 90, 95...), C, and C++. Public forum for API and membership

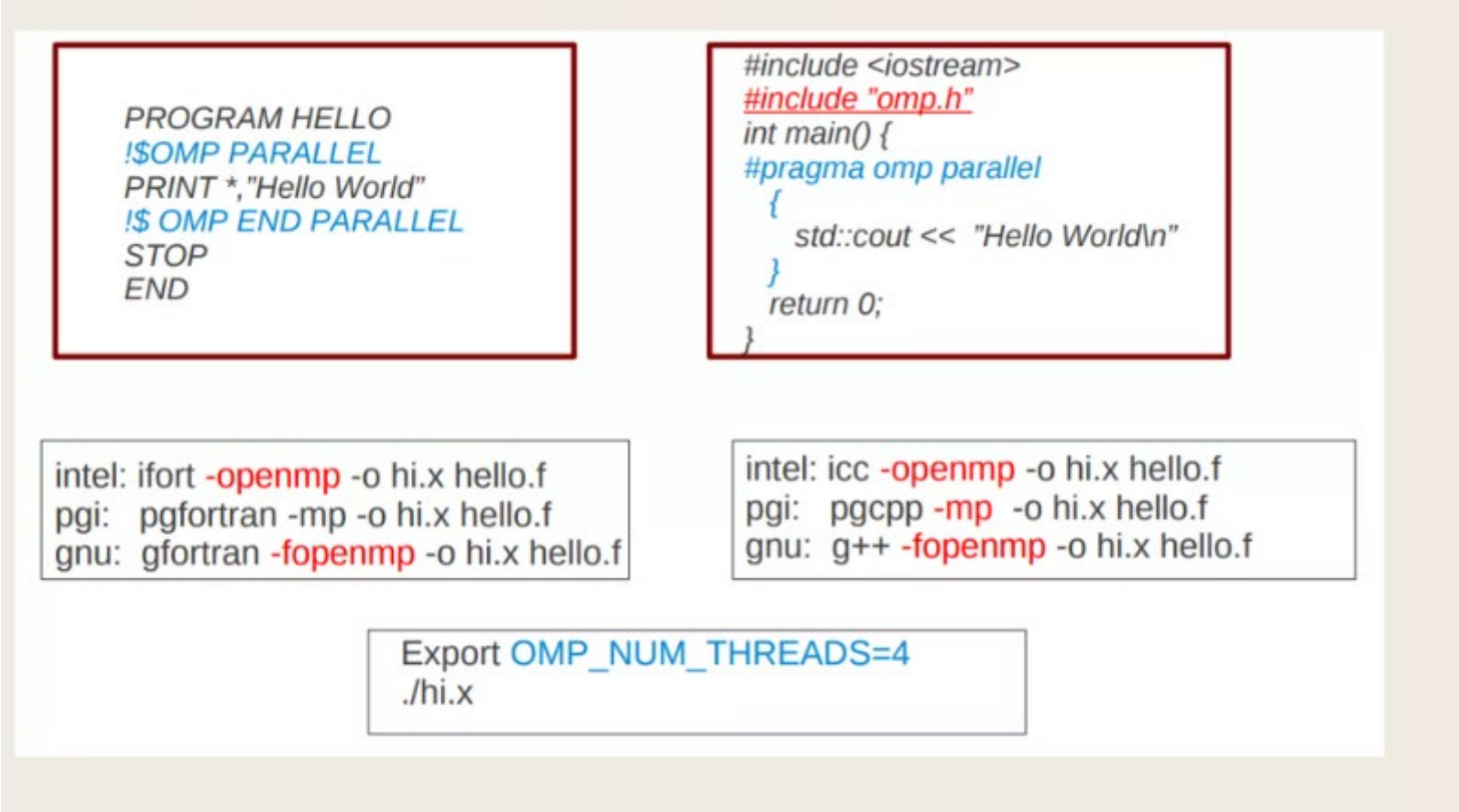
OpenMp Goals



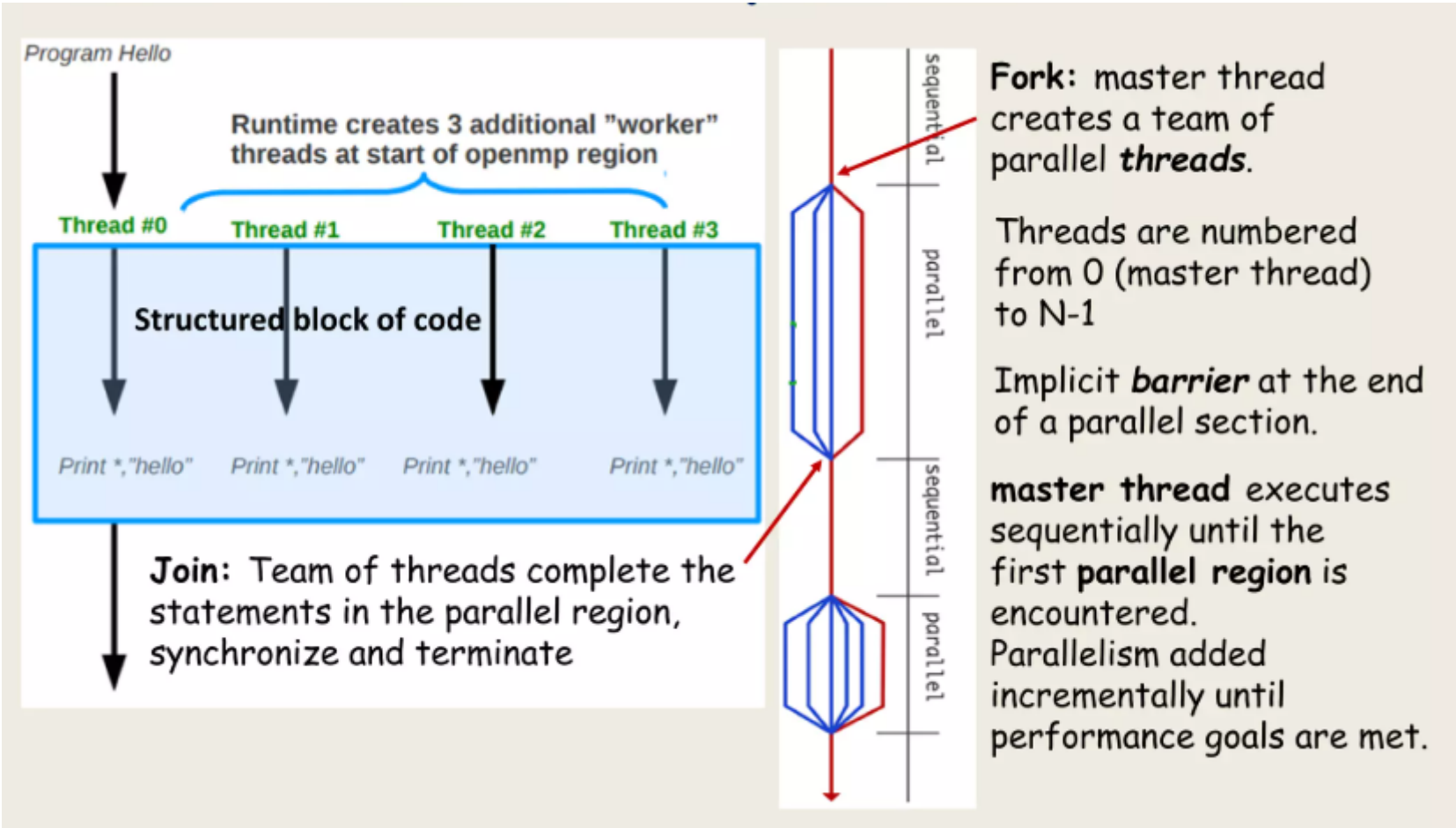
OpenMp #pragma

Special preprocessor instructions.
Typically added to a system to allow behaviors that aren't part of the basic C specification.
Compilers that don't support the pragmas ignore them.

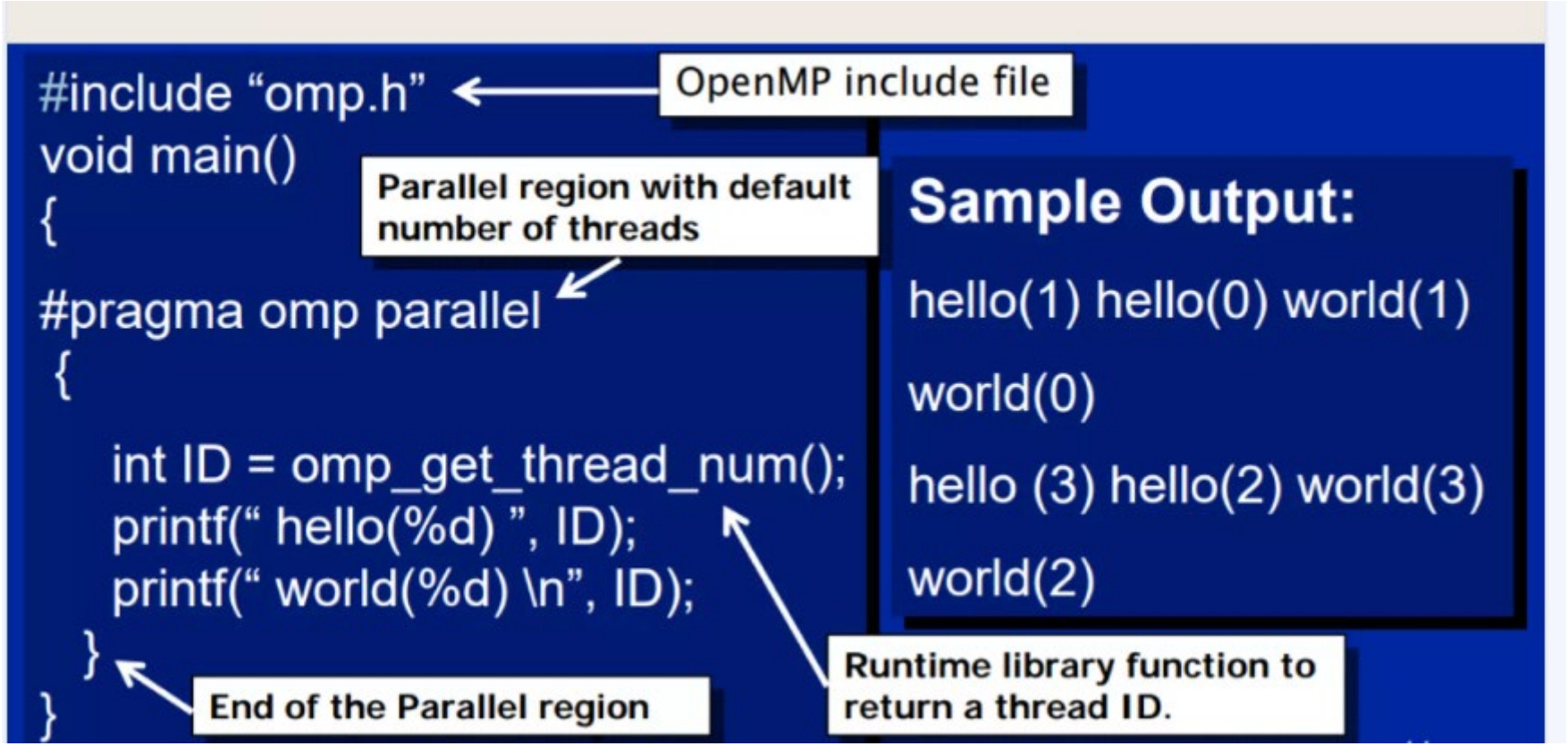
OpenMp #pragma



OpenMp: Hello World



Basic Functions



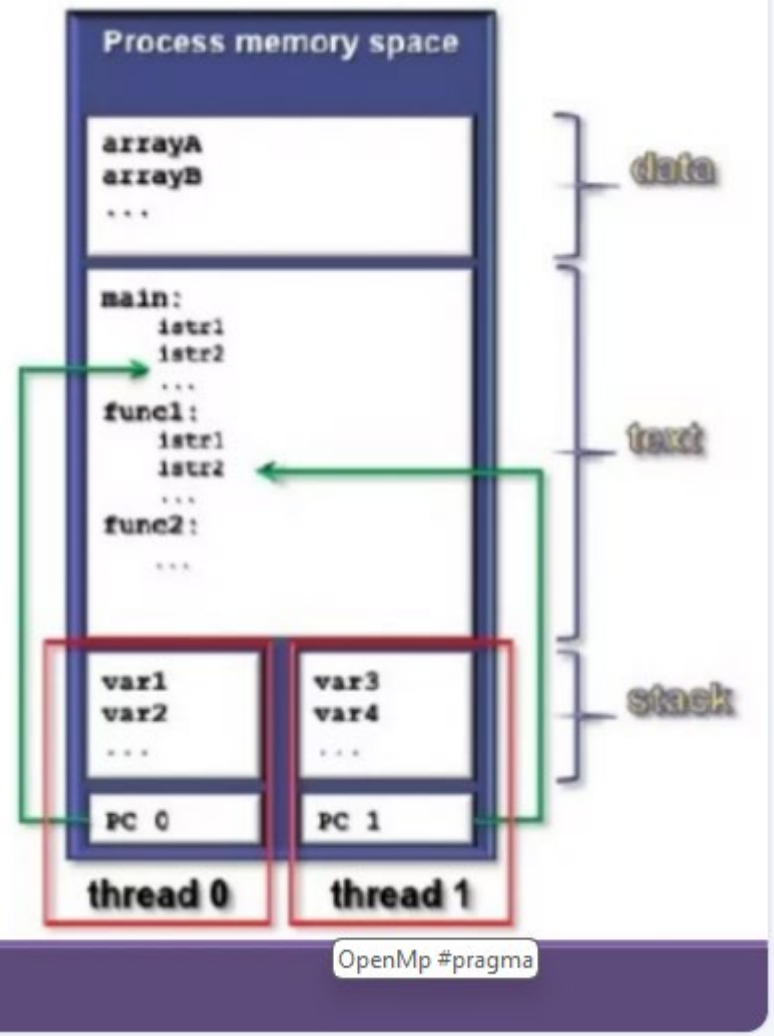
Each thread has its own stack, so it will have its own private (local) variables.

Each thread gets its own rank - **omp_get_thread_num**

The number of threads in the team - **omp_get_num_threads**

In OpenMP, **stdout** is shared among the threads, so each thread can execute the **printf** statement.

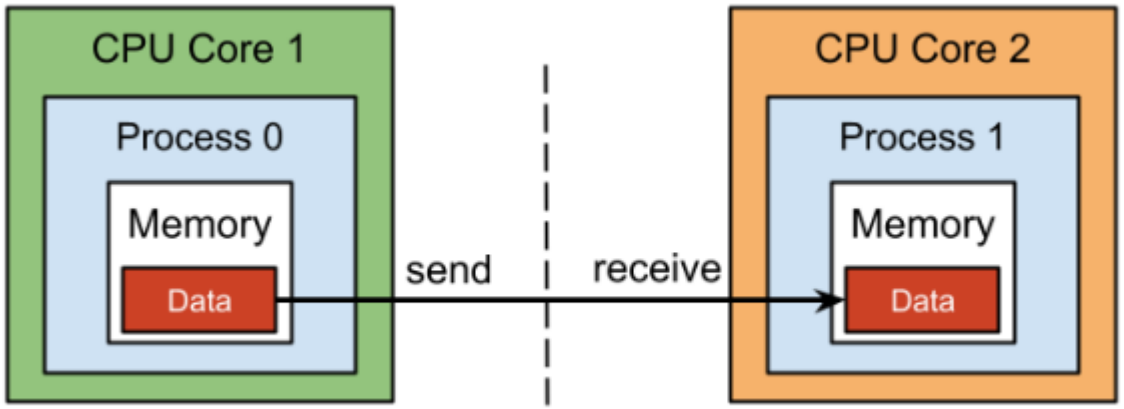
There is no scheduling of access to **stdout**, output is non-deterministic.



Message Passing Interface

- Message passing interface (MPI) is a standard specification of message-passing interface for parallel computation in distributed-memory systems.
- MPI isn’t a programming language. It’s a library of functions that programmers can call from C, C++, or Fortran code to write parallel programs.

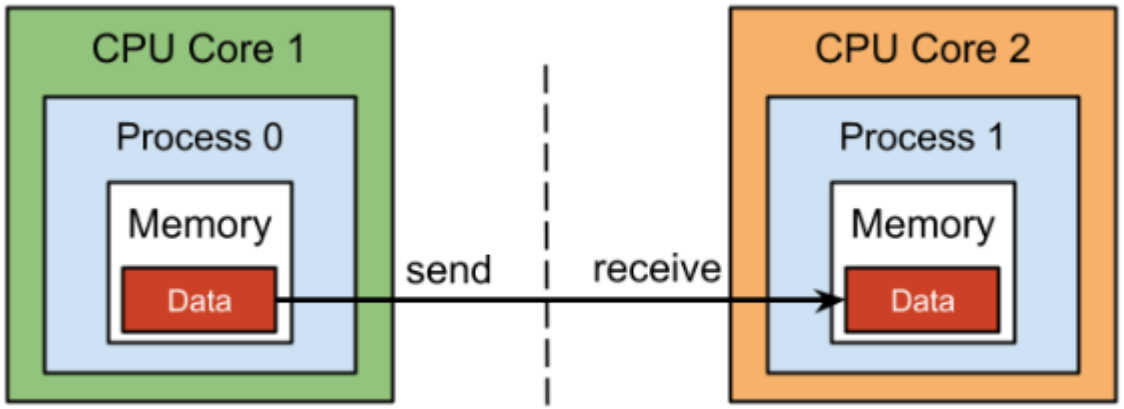
MPI Communication Methods



Point-to-Point Communication

- It involves the transfer of a message from one process to a particular process in the same communicator.
- MPI provides blocking (synchronous) and non-blocking (asynchronous) Point-to-Point communication.

MPI Communication Methods



Collective Communication

- With this type of MPI communication method, a process broadcasts a message is to all processes in the same communicator including itself.

Common MPI Distribution

- **Message passing interface chameleon (MPICH)** – It is a high-performance, open-source, portable implementation of message passing interface for parallel computation in distributed-memory systems distributed-memory.
- **Intel MPI Library** - programmer can use the Intel MPI Library to create advanced and more complex parallel applications that run on clusters with Intel-based processors.
- **MVAPICH** – It is an MPI implementation over the InfiniBand, Omni-Path, Ethernet iWARP, and RoCE packages.
- **Open message passing interface (OpenMPI)** is an open-source implementation of MPI that's maintained by large communities form industry and academia.

MPI	OpenMP
Available from different vendors and gets compiled on Windows, macOS, and Linux operating systems..	An add-on in a compiler such as a GNU compiler and Intel compiler.
Supports parallel computation for distributed-memory and shared-memory systems.	Supports parallel computation for shared-memory systems only.
A process-based parallelism.	A thread-based parallelism.
With MPI, each process has its own memory space and executes independently from the other processes.	With OpenMP, threads share the same resources and access shared memory.
Processes exchange data by passing messages to each other.	There is no notion of message-passing. Threads access shared memory.
Process creation overhead occurs one time.	It depends on the implementation. More overhead can occur when creating threads to join a task.

Any Question?