CSE502: Foundations of Parallel Programming

Lecture 20: Introduction to Distributed Memory Parallel Programming using the Message Passing Interface

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Last Class

- Tasking constructs in OpenMP

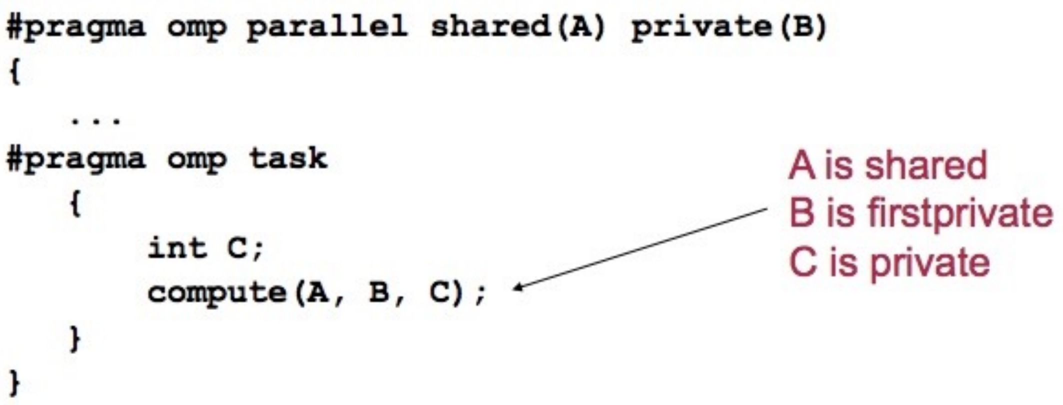
```
#pragma omp parallel
{
  #pragma omp master
   {
      #pragma omp task
         fred();
      #pragma omp task
         daisy();
      #pragma omp taskwait
      #pragma omp task
         billy();
   }
}
```

Can we have?
`#pragma omp single`

Tasks executed by some thread in some order

fred() and **daisy()** must complete before **billy()** starts

```
#pragma omp single
{
   for (i=0; i<ONEZILLION; i++)
      #pragma omp task untied
        process(item[i]);
}
```



```
#pragma omp parallel shared(A) private(B)
{
     ...
#pragma omp task
     {
          int C;
          compute(A, B, C);
     }
}
```

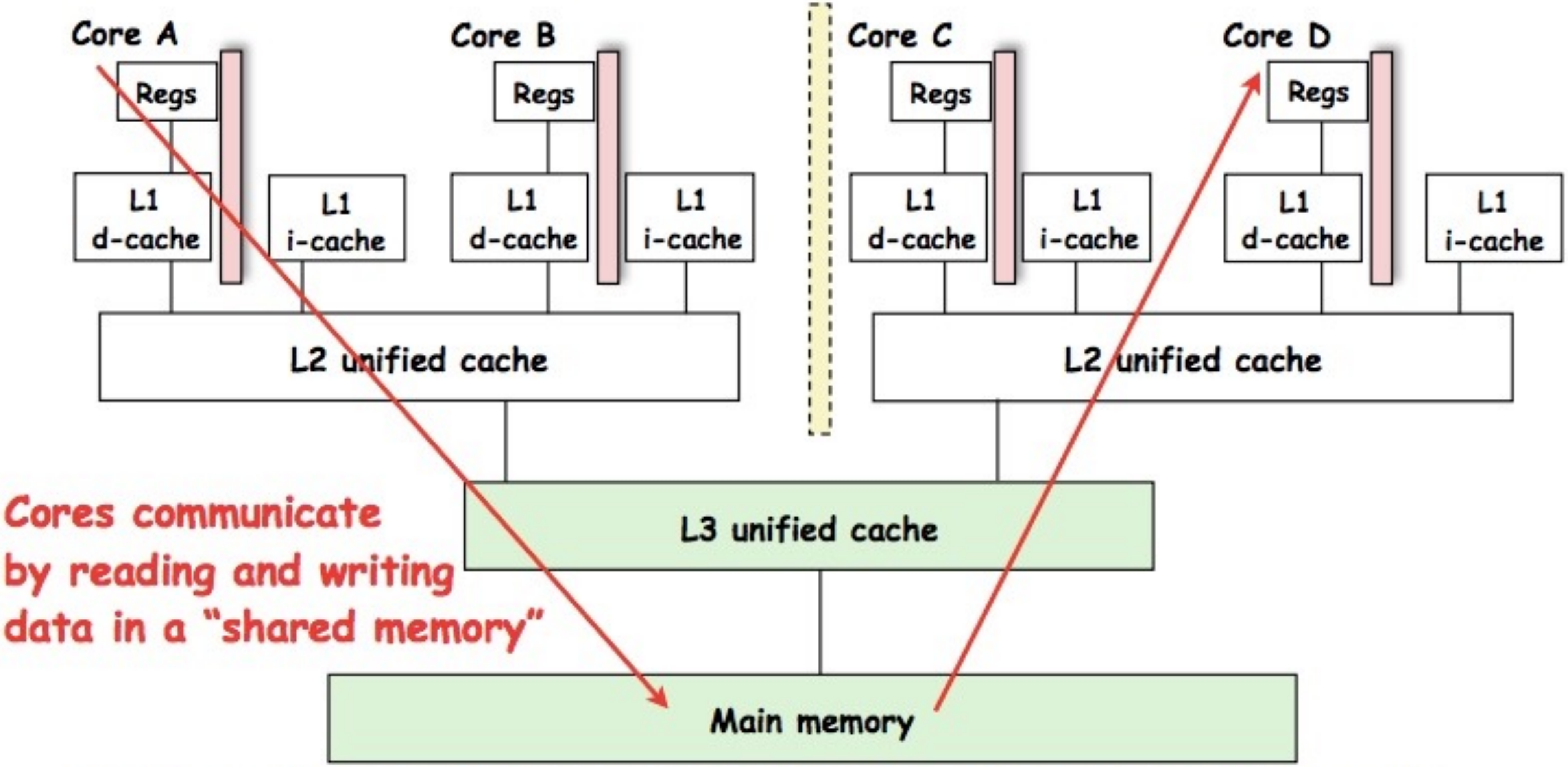
A is shared
B is firstprivate
C is private

Today's Class

- Distributed memory parallel programming using Message Passing Interface
  - An introduction
- Quiz-4

Acknowledgements: Slides in this lecture are adapted from COMP322 course at Rice University and from the MPI tutorial available at LLNL website (https://computing.llnl.gov/tutorials/mpi/)

Organization of a Shared-Memory Multicore Symmetric Multiprocessor (SMP)



Cores communicate by reading and writing data in a "shared memory"

- **Memory hierarchy for a single Intel Xeon (Nehalem) Quad-core processor chip**

Organization of a Distributed Memory Multiprocessor

Figure (a)

- Host node (Pc) connected to a cluster of processor nodes (P0 … Pm)
- Processors P0 … Pm communicate via an interconnection network

Figure (b)

- Each processor node consists of a processor, memory, and a Network Interface Card (NIC) connected to a router (R) in the interconnect

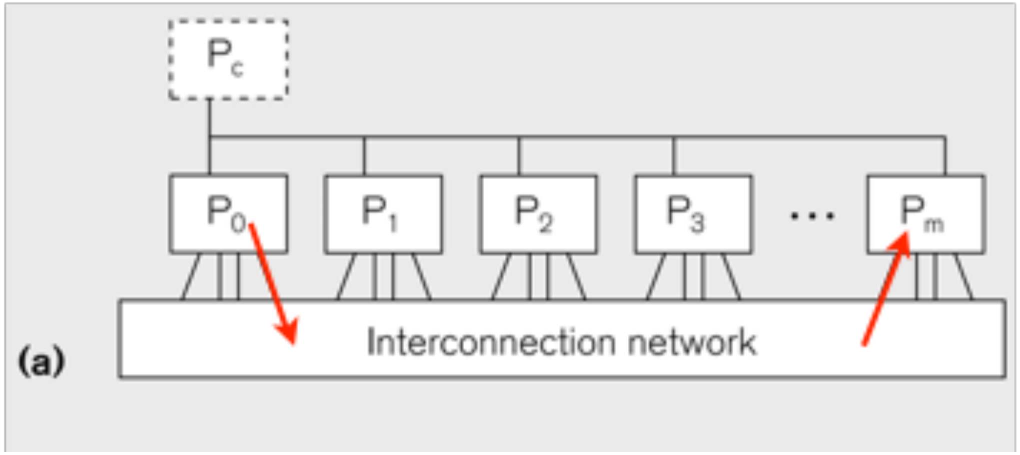
PBS sample jobscript

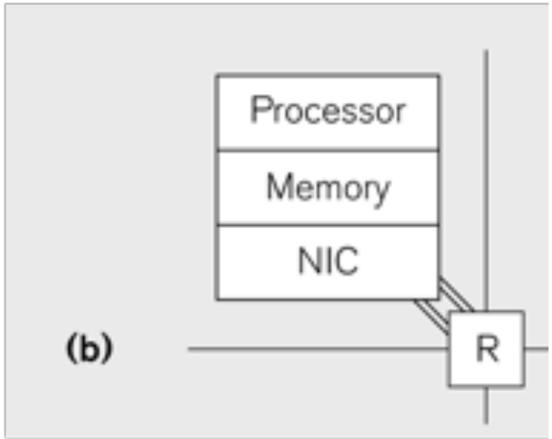
```
#!/bin/bash
#PBS –q <queue name>
#PBS –l <core count>
#PBS –l walltime=00:02:30
cd $PBS_O_WORKDIR
aprun –n <Processes> -N <process/node> -d <core/node> ./exe
>./output.log
```

```
$qsub <jobscript>
$qstat –f jobid
```

Comparing with HClib places: Each node is like a "distributed place" with no sharing of memory

==> Processors communicate by sending messages via an interconnect

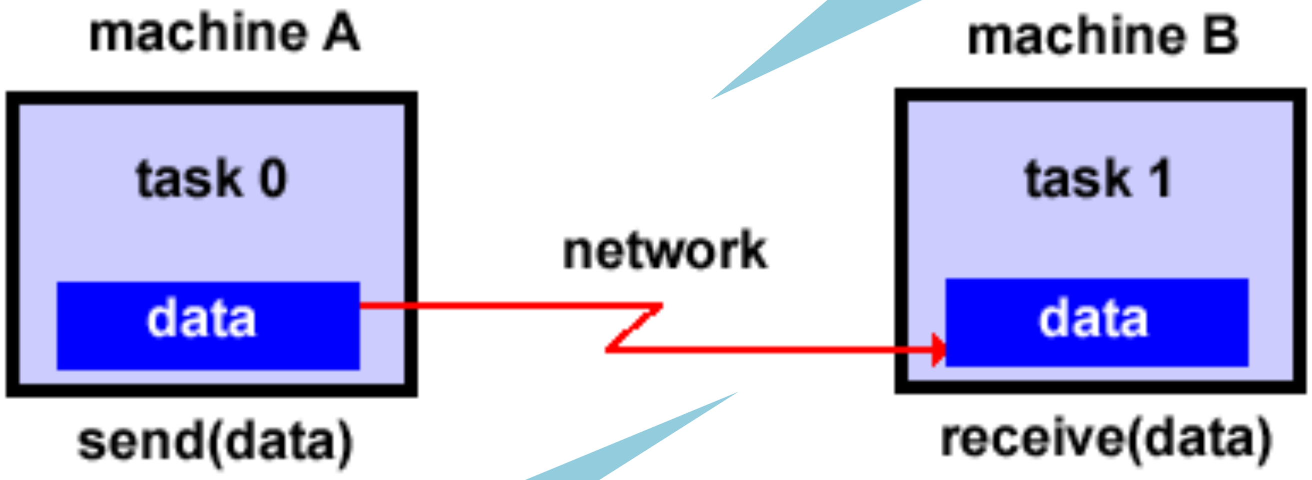




Message Passing Model Characteristics

Message Passing

Paired Communication



Distributed Computing

Data Distribution: Local View in Distributed-Memory Systems

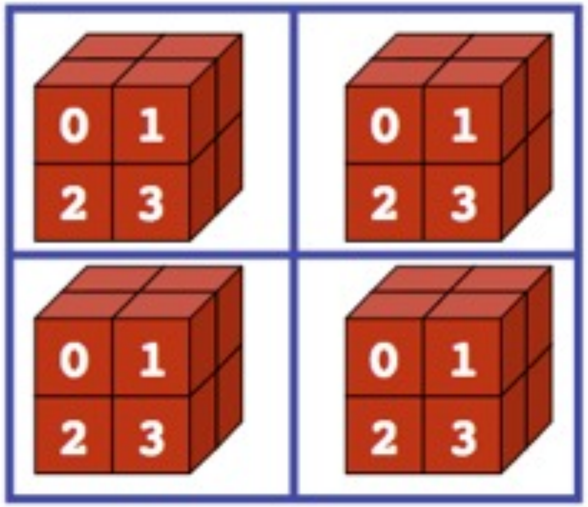
Distributed memory

- Each process sees a local address space
- Processes send messages to communicate with other processes

Data structures

- Presents a Local View instead of Global View
- Programmer must make the mapping

Global View



Local View (4 processes)

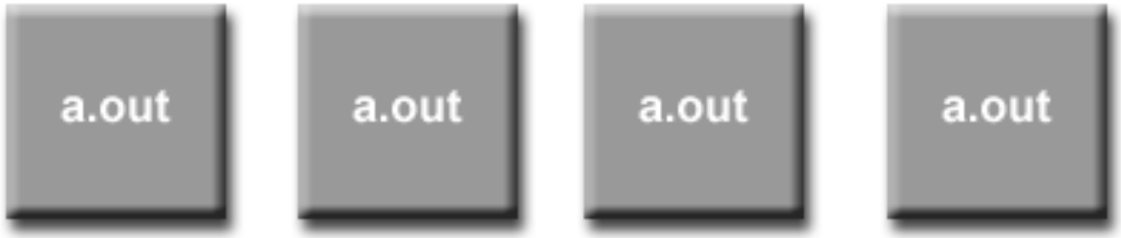
Message Passing for Distributed Memory Multiprocessors

- The logical view of a machine supporting the message-passing paradigm consists of *p processes*, each with its own exclusive address space, that are capable of executing on different nodes in a distributed-memory multiprocessor
    1. Each data element must belong to one of the partitions of the space; hence, data must be explicitly partitioned and placed.
    2. All interactions (read-only or read/write) require cooperation of two processes - the process that has the data and the process that wants to access the data.
- These two constraints, while onerous, make underlying costs very explicit to the programmer.
- In this loosely synchronous model, processes synchronize infrequently to perform interactions. Between these interactions, they execute completely asynchronously.

MPI: The Message Passing Interface

- MPI is a ***specification*** for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be
- Reasons for using MPI
  - Standardization
    - Supported on almost every HPC platforms
  - Portability
    - Same code will even run on another platform
  - Performance Optimization
    - Vendors apply optimizations specific to their HPC platform
  - Availability
    - Both vendor specific as well as open-sourced

SPMD Pattern



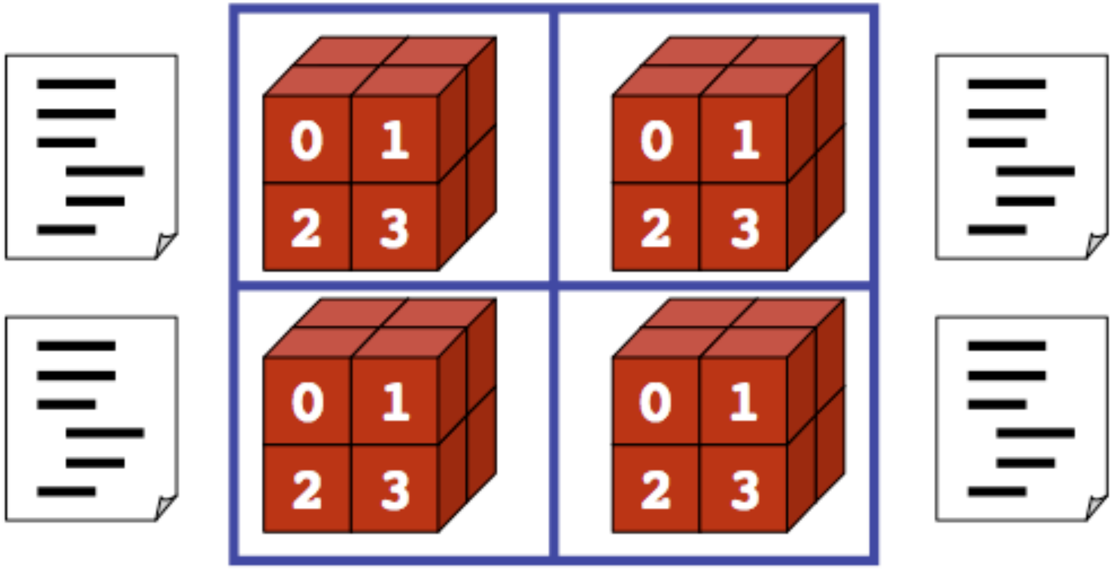
Process_1 (rank=0)
Process_2 (rank=1)
Process_3 (rank=2)
Process_4 (rank=3)

- SPMD: Single Program Multiple Data
- Run the same program on P processing elements (PEs)
- Use the "rank" … an ID ranging from 0 to (P-1) … to determine what computation is performed on what data by a given PE
- Different PEs can follow different paths through the same code

Modeling the SMPD Model

SPMD code

– Write one piece of code that executes on each processor



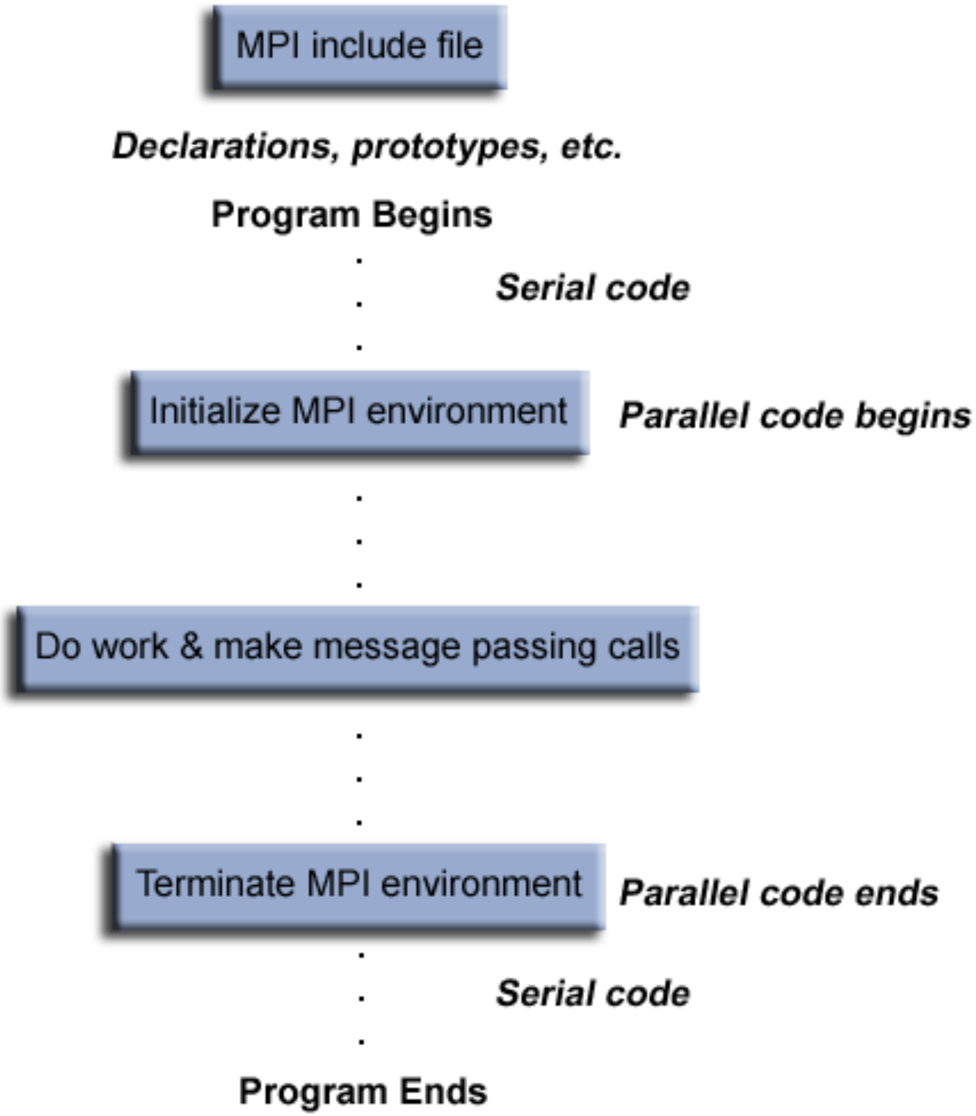
Local View (4 processes)

- Processors must communicate via messages for non-local data accesses

How Big is MPI ?

- There are over 430+ routines defined in MPI-3
  - Most MPI programs can be written using a dozen or less routines

General MPI Program Structure



MPI include file

Declarations, prototypes, etc.

Program Begins

Serial code

Initialize MPI environment

Parallel code begins

Do work & make message passing calls

Terminate MPI environment

Parallel code ends

Serial code

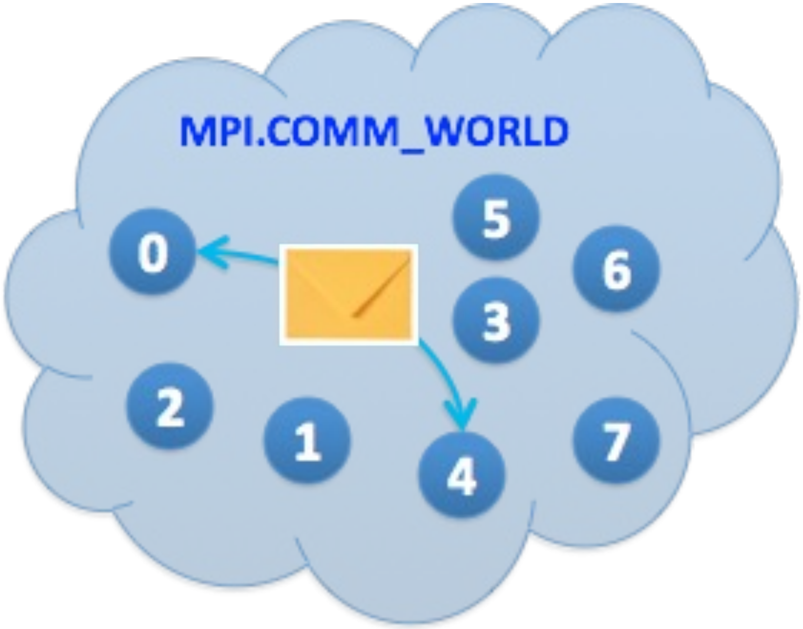
Program Ends

Our First MPI Program

```c
// the header file containing MPI APIs
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv) {
    // Initialize the MPI runtime
    MPI_Init(argc, argv);
    int rank, nprocs;
    // Get the total number of processes in MPI_COMM_WORLD
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    // Get the rank of this process in MPI_COMM_WORLD
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf(“My rank is %d in world of size %d\n”, rank, nprocs);
    // Terminate the MPI runtime
    MPI_Finalize();
    return 0;
}
```

MPI Communicators

- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other
- Most MPI routines require you to specify a communicator as an argument
- Default communicator is MPI_COMM_WORLD
  - All processes are its members
  - It has a size (the number of processes)
  - Each process has a rank within it
  - Can think of it as an ordered list of processes



Next Lecture

- Point to point communications in MPI
- Lecture-21 on Saturday (Tuesday-TT)
- Lab-6 on Saturday in **LHC - L321** from 2-3pm

Reading Material

- Tutorial on MPI by LLNL
  - https://computing.llnl.gov/tutorials/mpi/