Parallel Computing - MPI

Message Passing Interface

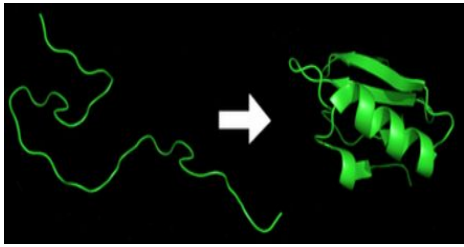
Samir Shaikh
HPC - Tech, CDAC Pune

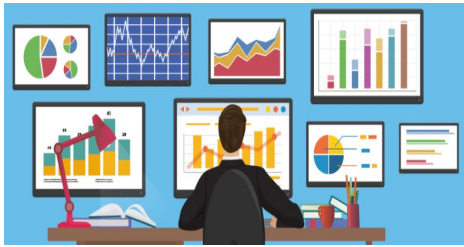
Agenda

- Why Parallel Computing ?
- Why we need ever-increasing Performance ?
- Parallel programing Architectures/Model ..
- MPI - Message Passing Interface
    - What is MPI ?, Need and Evolution of MPI.
    - MPI program - Compile and Execution
    - MPI Program Structure
    - MPI Routines
    - ……

Why we need Ever-Increasing Performance ?

- Accurate medical imaging
- Fast and accurate web searches
- Realistic computer games, Entertainment
- Climate modeling
- Protein folding
- Artificial Intelligence
- Energy research
- Data analysis .....
- .....



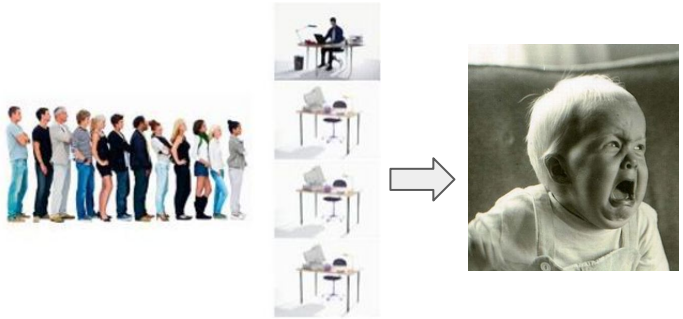


Why Parallel Computing ?

- Aren't single processor systems fast enough ?

Why Parallel Computing ?

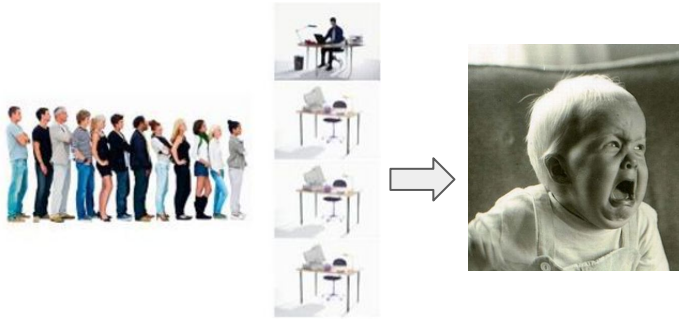
- **Aren't single processor systems fast enough ?**



Serial Computing

Why Parallel Computing ?

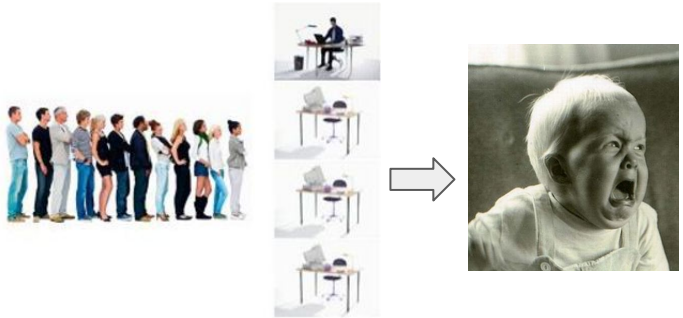
- Aren't single processor systems fast enough ?
- Why to build parallel systems ? Why build systems with multiple processors ?



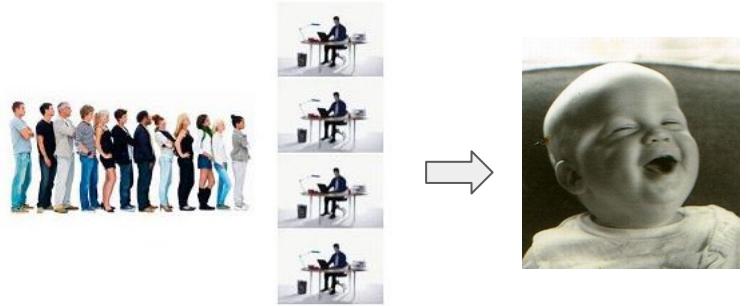
Serial Computing

Why Parallel Computing ?

- Aren't single processor systems fast enough ?
- Why to build parallel systems ? Why build systems with multiple processors ?



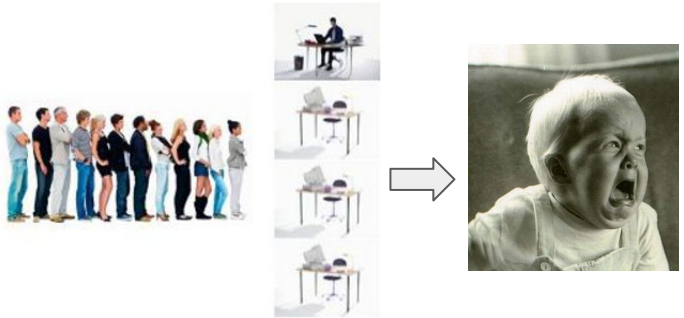
Serial Computing



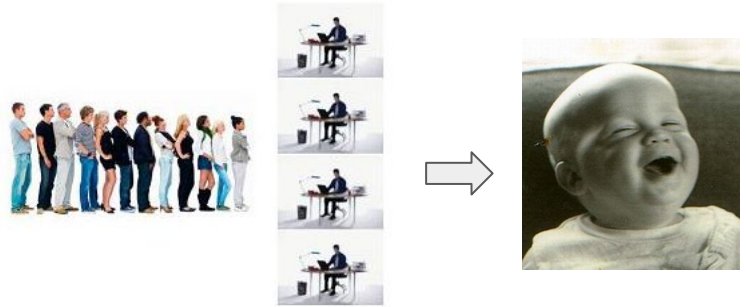
Parallel Computing

Why Parallel Computing ?

- Aren't single processor systems fast enough ?
- Why to build parallel systems ? Why build systems with multiple processors ?
- Why can't we write programs that will automatically convert serial programs to parallel programs ?



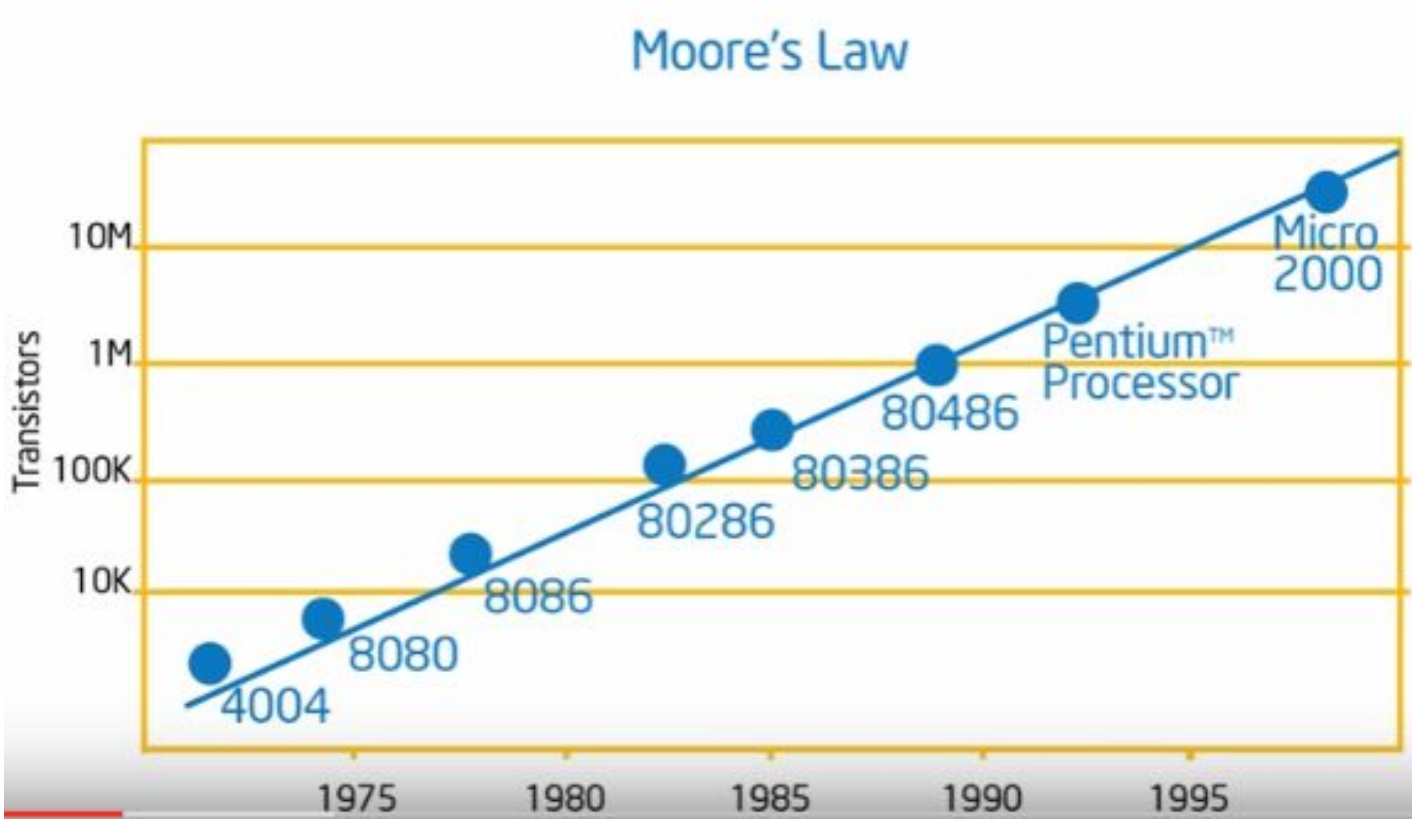
Serial Computing



Parallel Computing

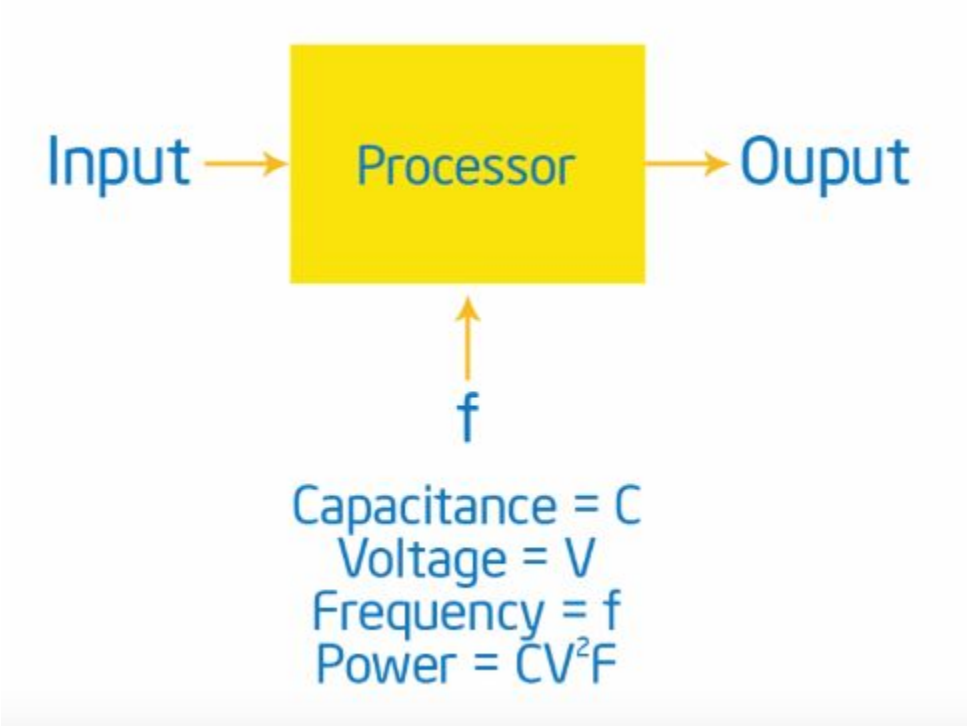
What Moore's Law tells.. ?

What Moore's Law tells.. ?



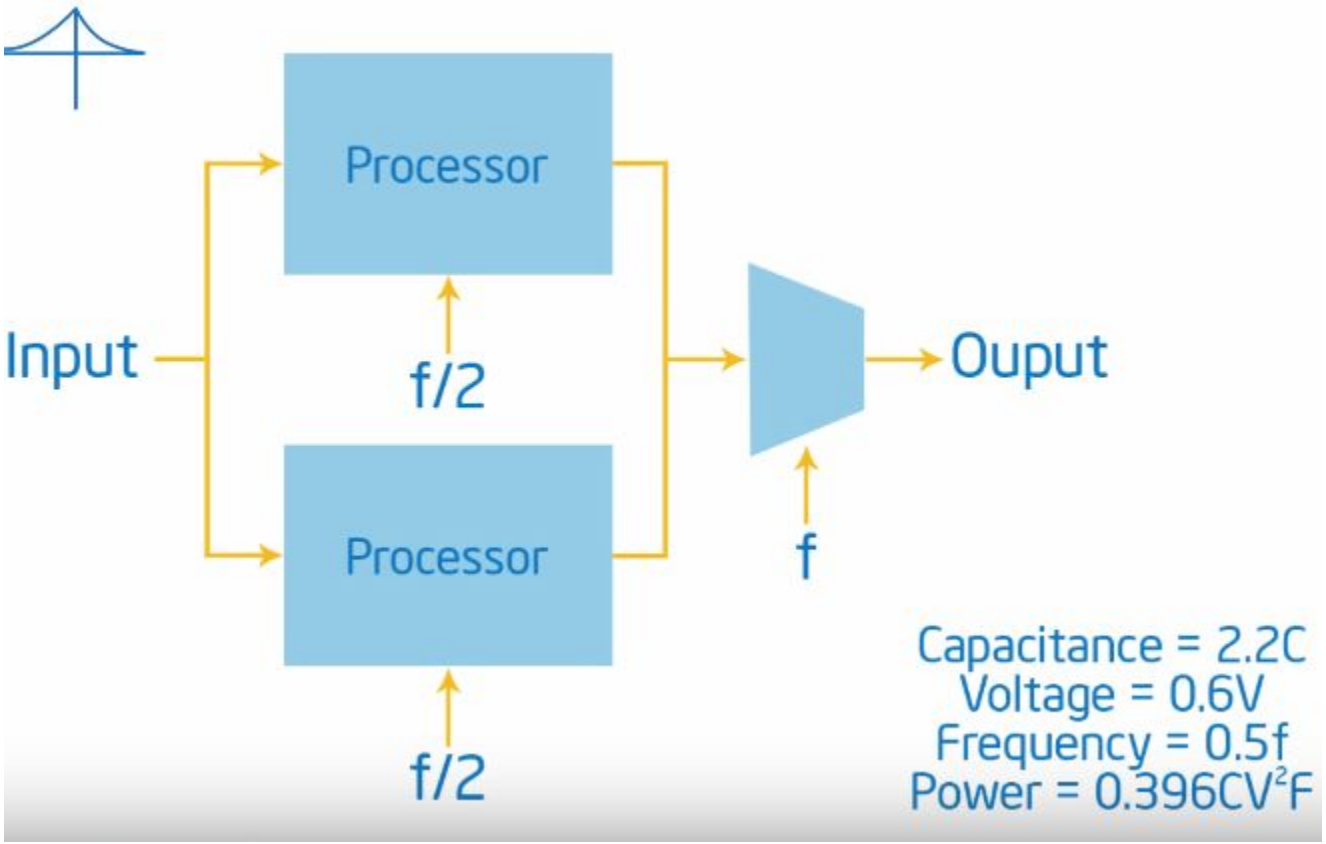
Uniprocessor ?

Uniprocessor ?



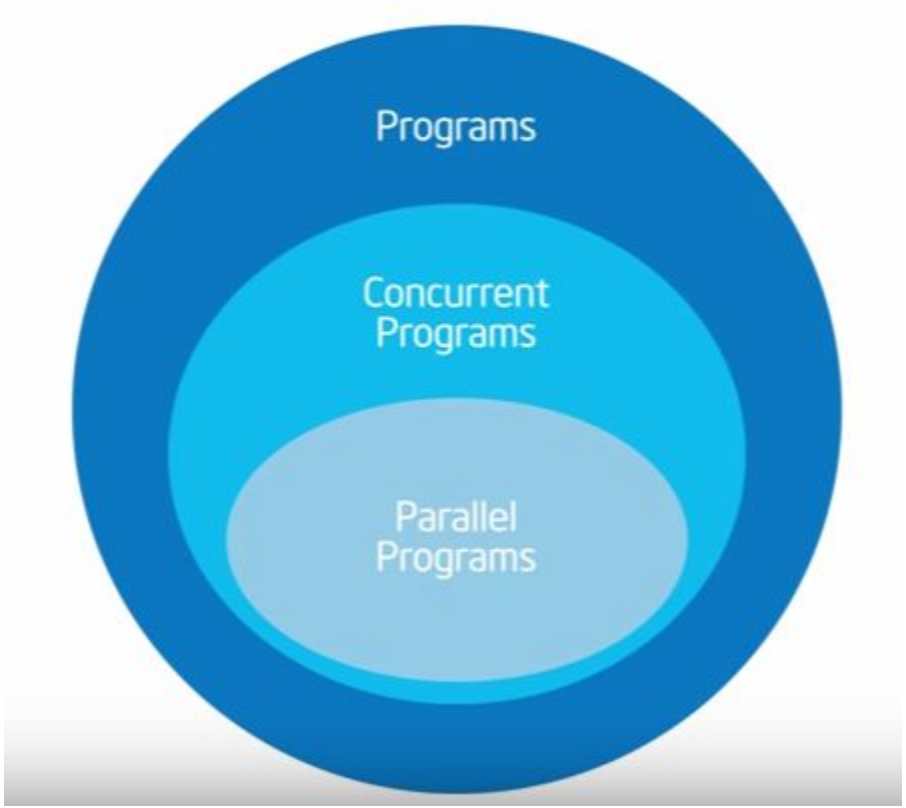
Parallel Architecture ?

Parallel Architecture ?



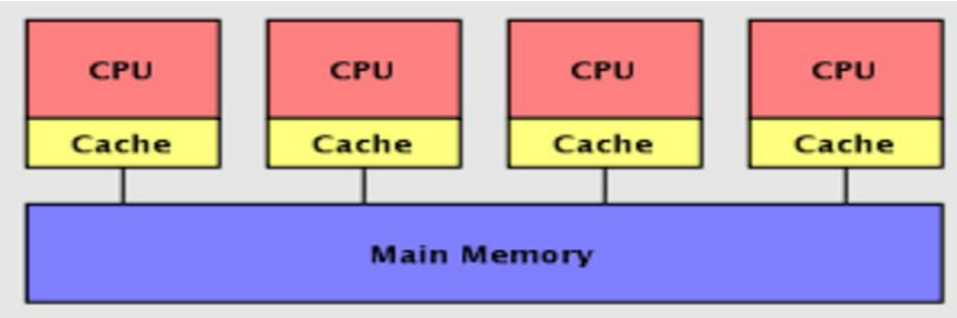
Parallel program

Parallel program



Parallel Programming Models..

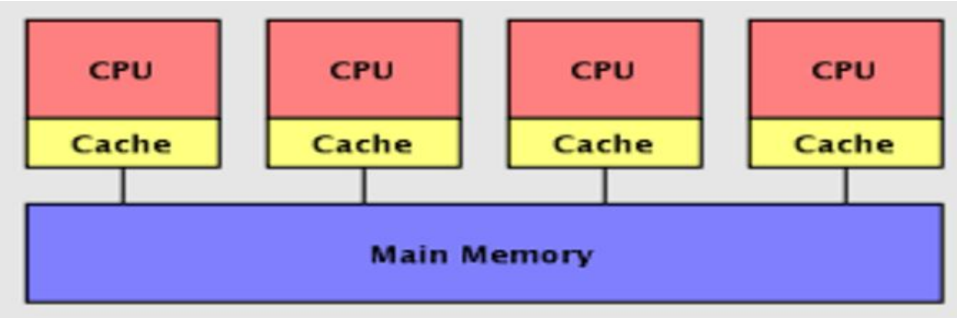
❑ Shared-memory Model



- UMA - Uniform Memory Access
- NUMA - Non-Uniform Memory Access

Parallel Programming Models..

❑ Shared-memory Model

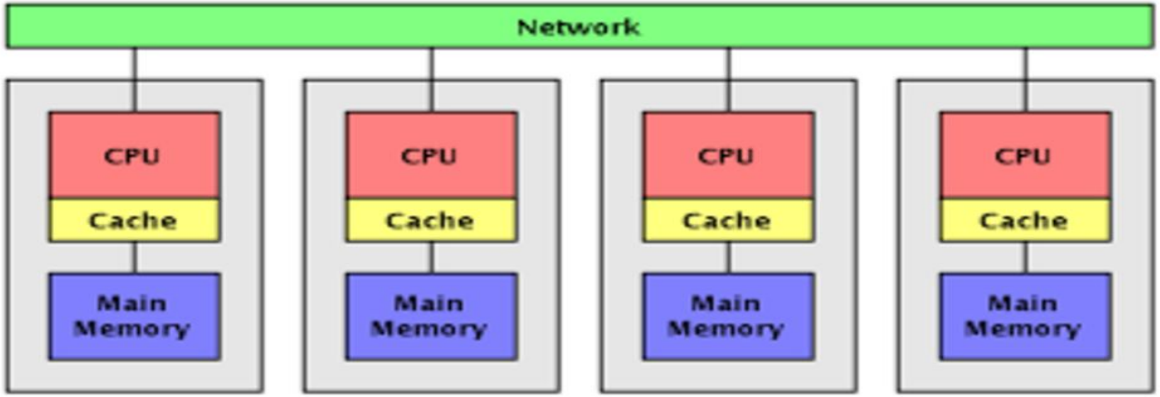


- UMA - Uniform Memory Access
- NUMA - Non-Uniform Memory Access

❖ openMP
❖ Pthreads ...

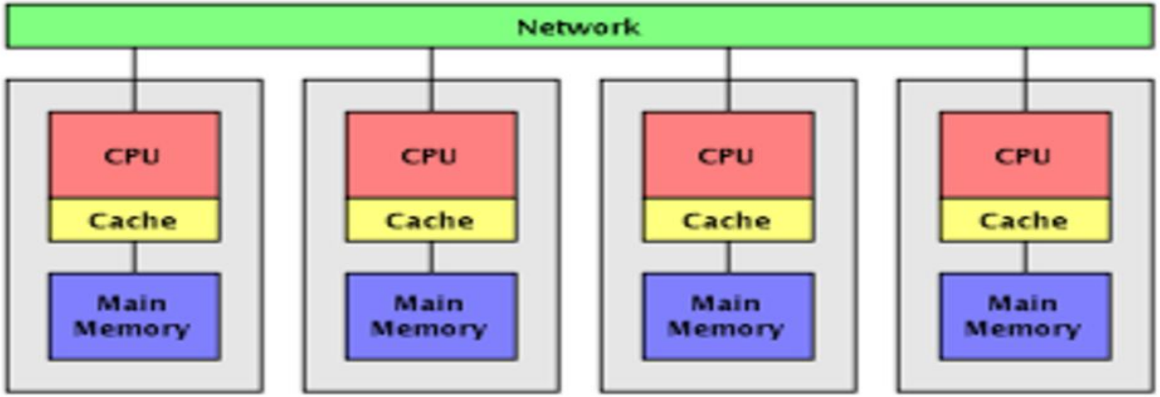
Parallel Programming Models..

- **Distributed-memory Model**



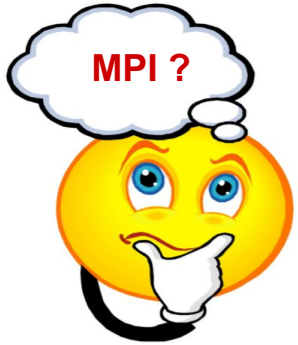
Parallel Programming Models..

- **Distributed-memory Model**



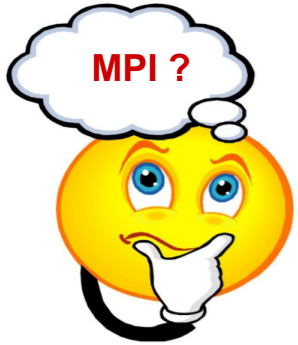
- **MPI - Message Passing Interface**

MPI - Message Passing Interface

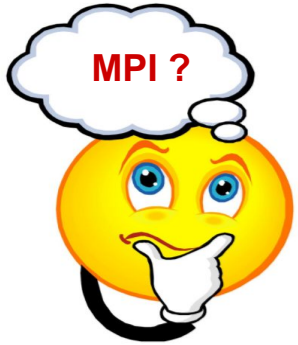


MPI - Message Passing Interface

- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum



MPI - Message Passing Interface

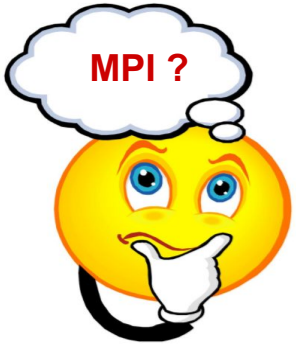


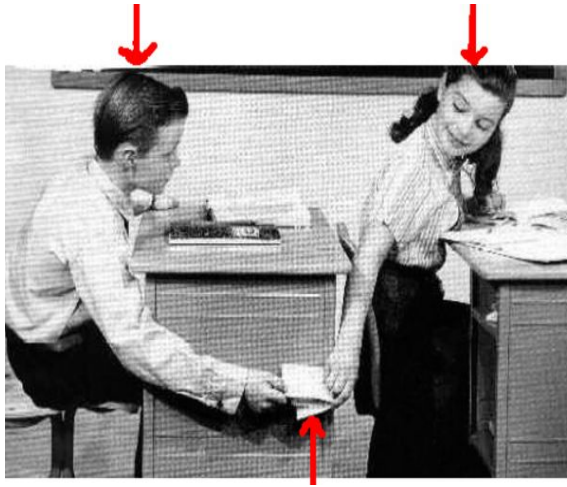
- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process

MPI - Message Passing Interface

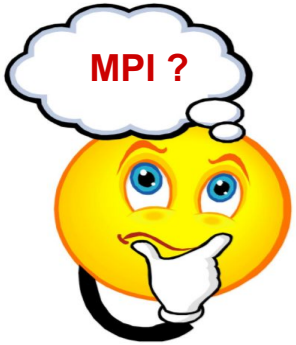
- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum

- In MPI a Message is passed from one process to another process

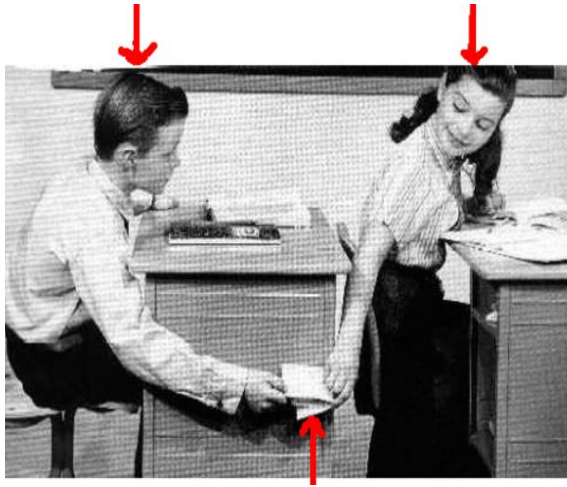




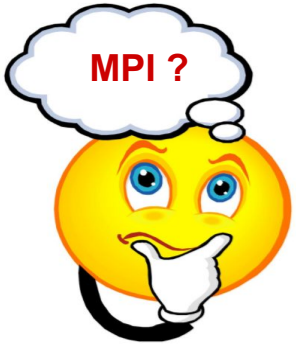
MPI - Message Passing Interface



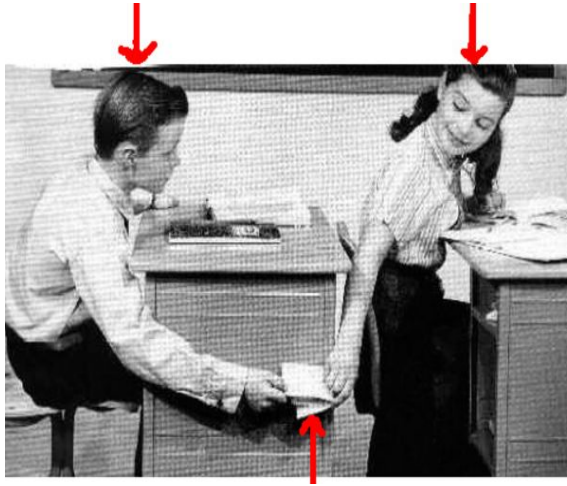
- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process
- MPI is based on Routines.



MPI - Message Passing Interface



- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process
- MPI is based on Routines.
- MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms.



MPI - Development

- The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.x
  - MPI-3.1 - Jun 2015
  - MPI-3.0 - Sep 2012 Standard was approved
  - MPI-2.2 - Sep 2009
  - MPI-2.1 - Sep 2008
  - MPI-1.3 - May 2008
  - MPI-1.2 - July 1997
  - MPI-1.1 - June 1995
  - MPI-1.0 - May 1994

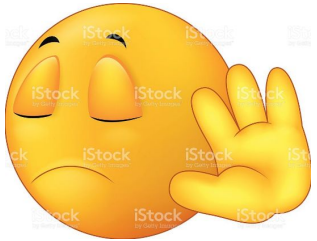
MPI - Development

- The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.x

  - MPI-3.1 - Jun 2015
  - MPI-3.0 - Sep 2012 Standard was approved
  - MPI-2.2 - Sep 2009
  - MPI-2.1 - Sep 2008
  - MPI-1.3 - May 2008
  - MPI-1.2 - July 1997
  - MPI-1.1 - June 1995
  - MPI-1.0 - May 1994

Wait....

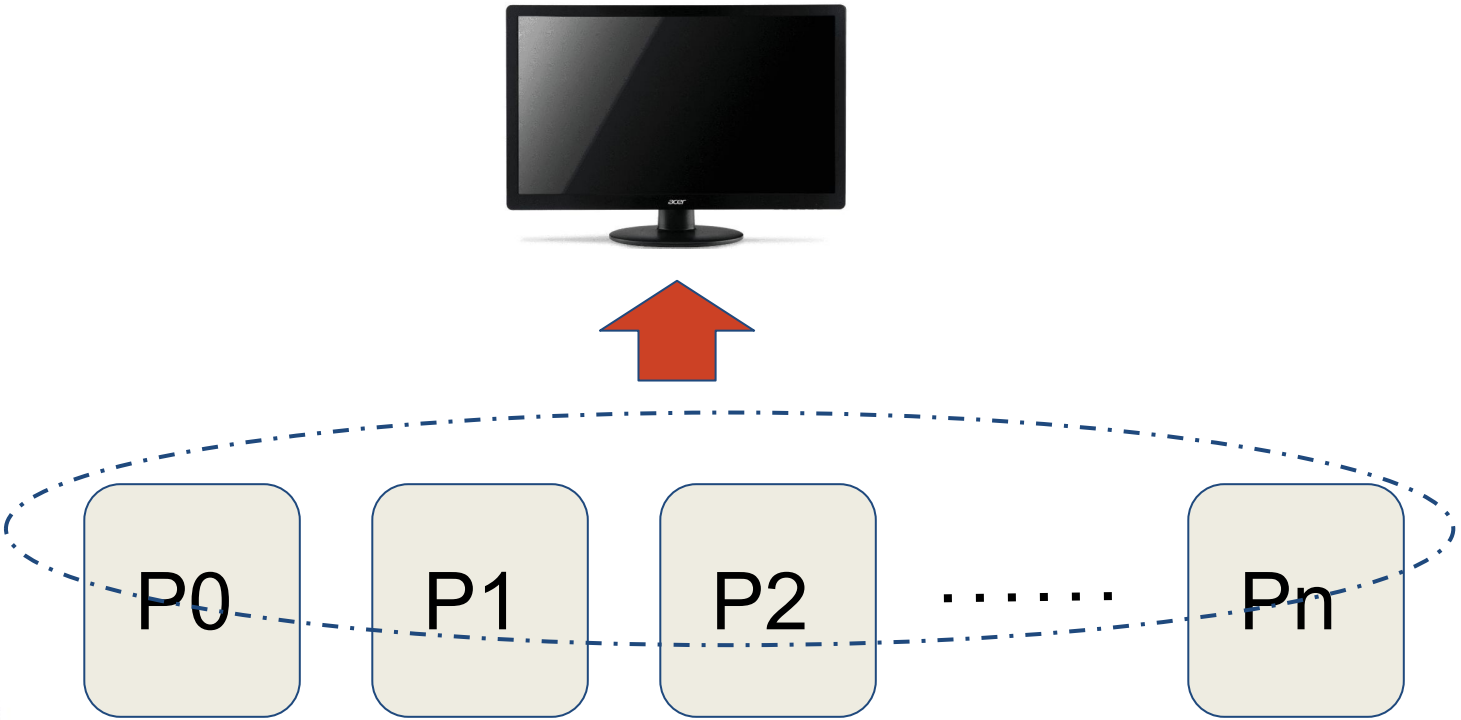
..Answer me first



- What is Process ?

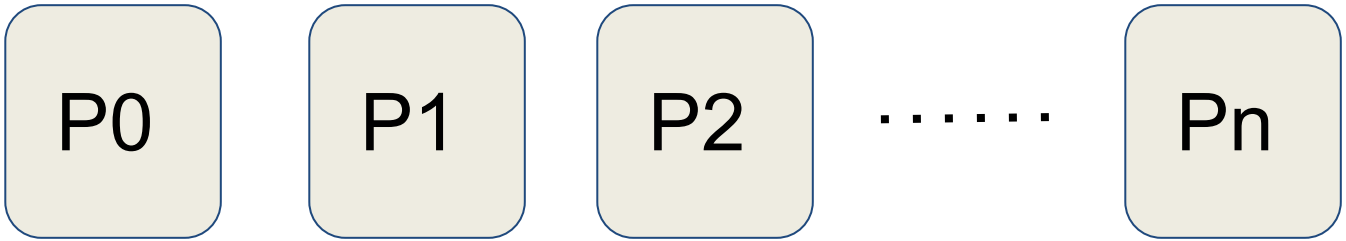
- Is MPI a new programming Language ..?

The Goal ..?

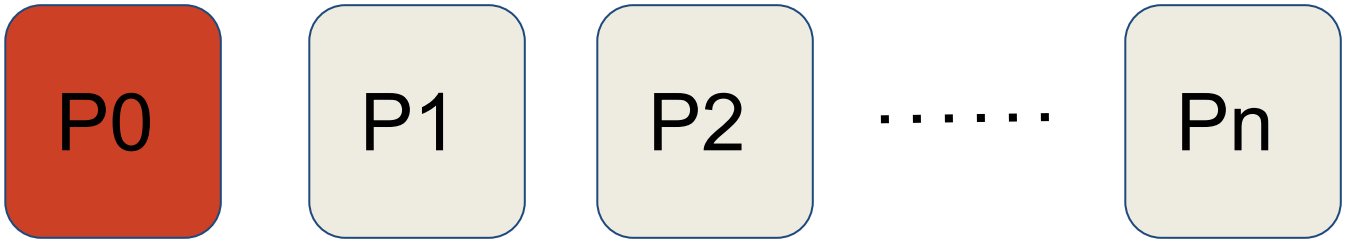


How to Achieve it ..?

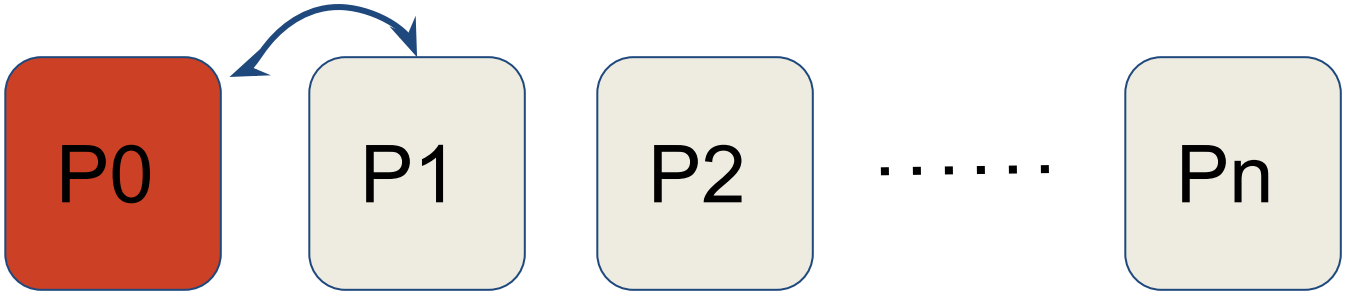
How to Achieve it ..?



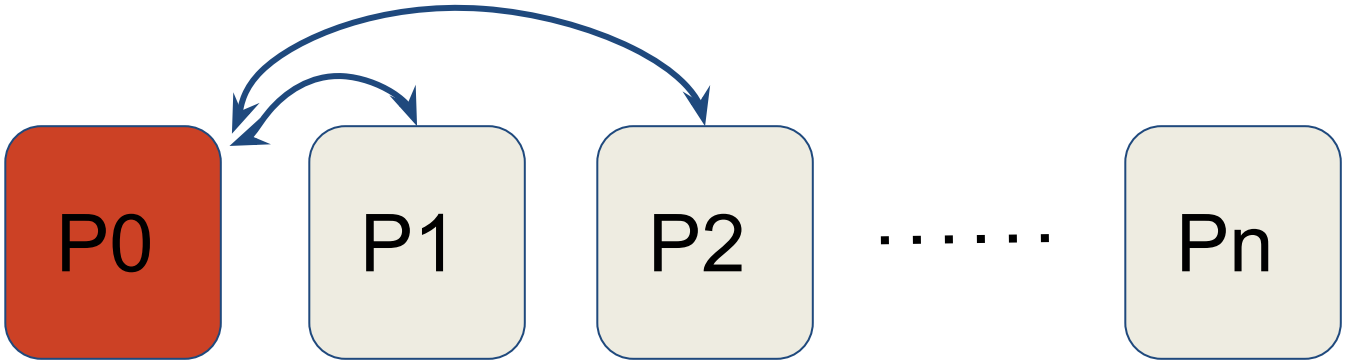
How to Achieve it ..?



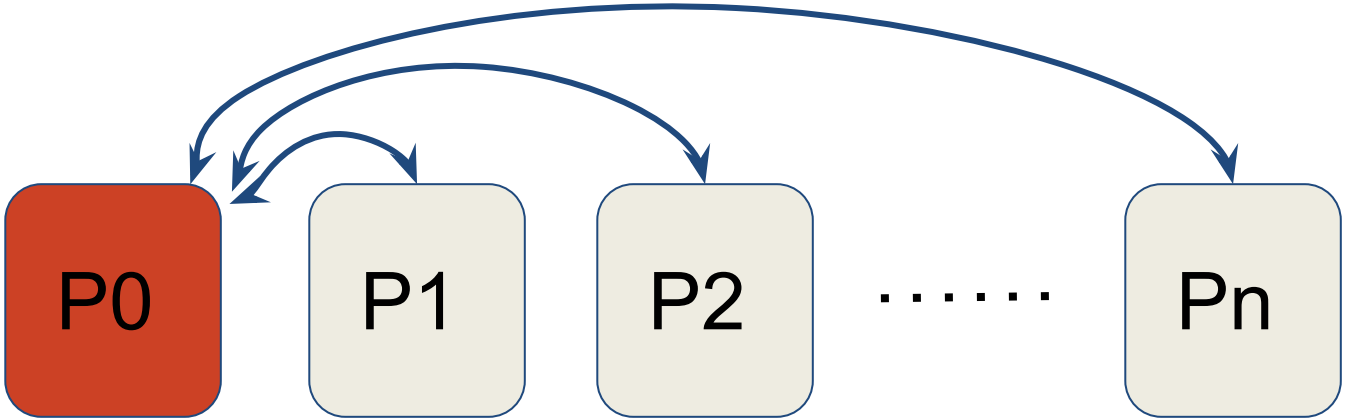
How to Achieve it ..?



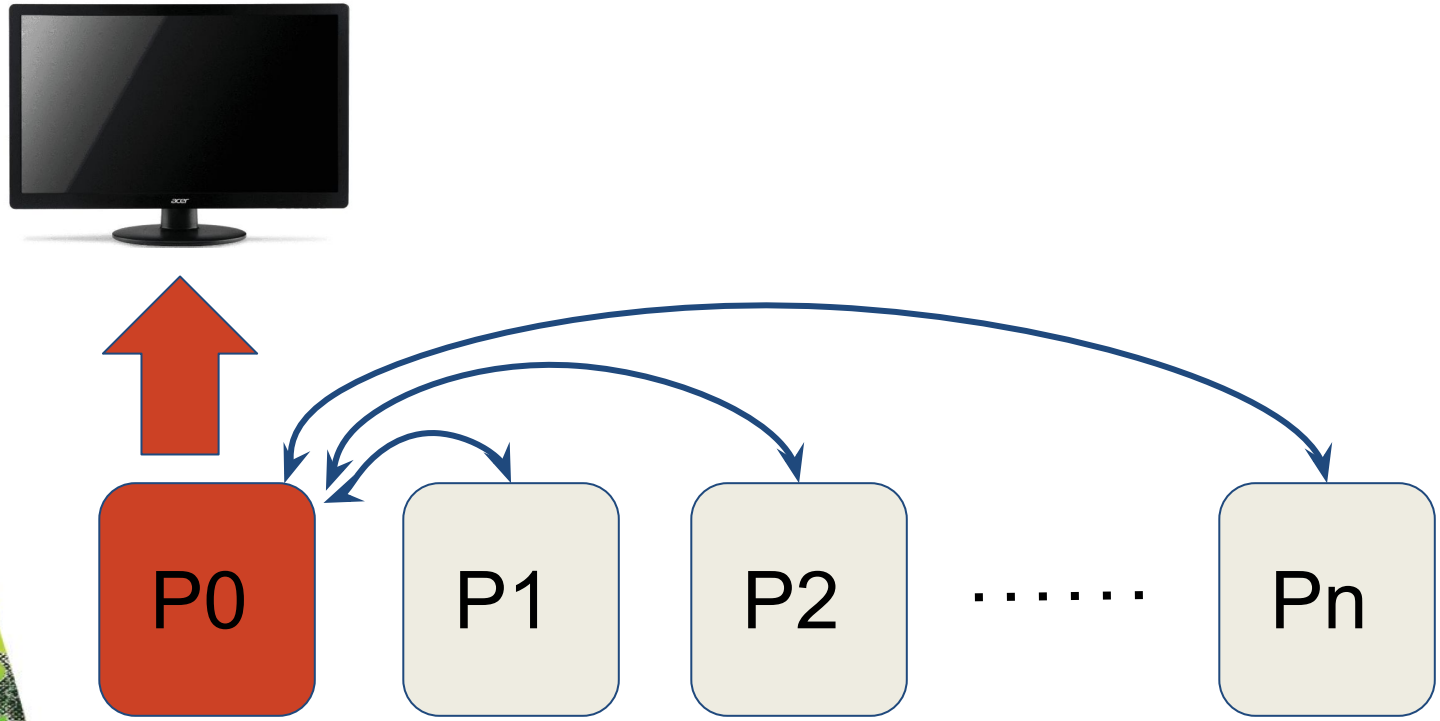
How to Achieve it ..?



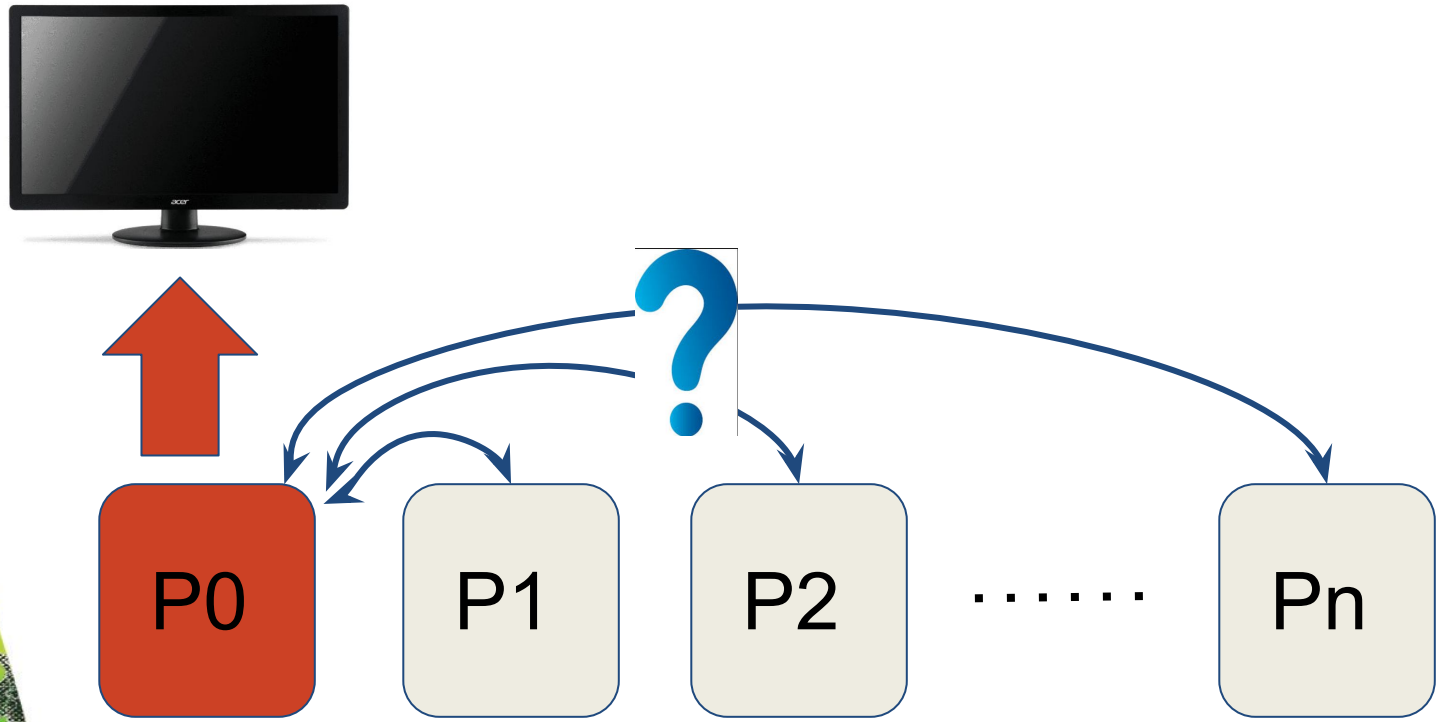
How to Achieve it ..?



How to Achieve it ..?

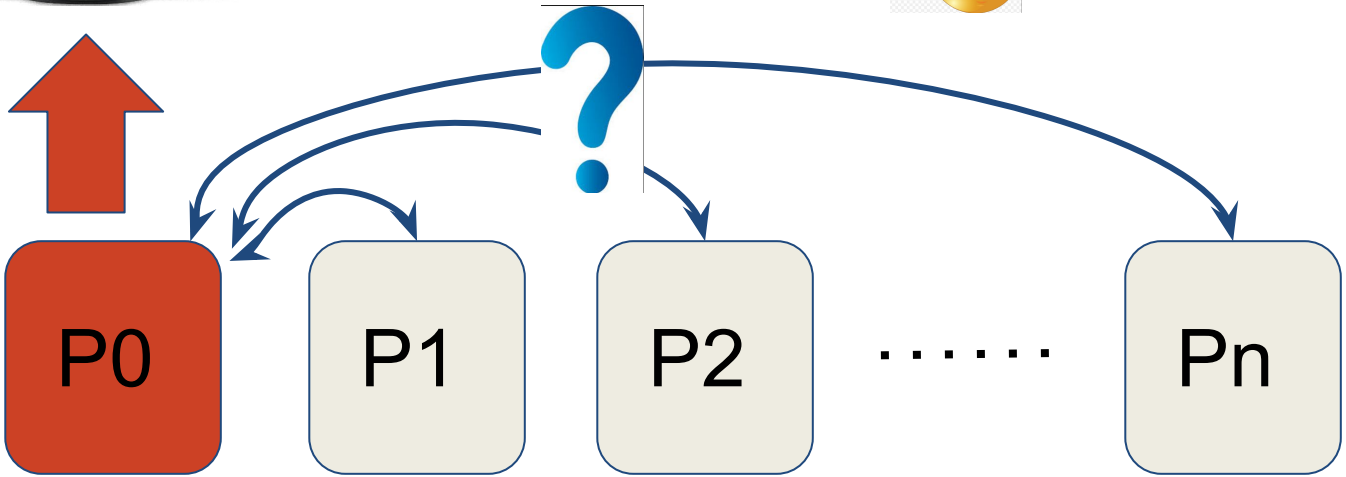


How ..?



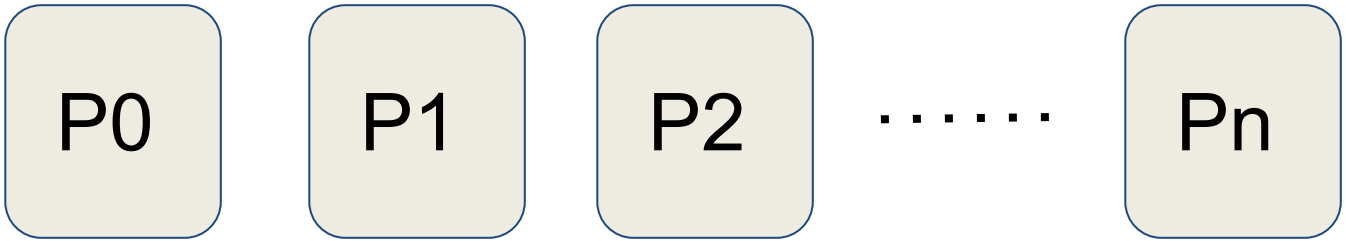
How ..?

MPI 



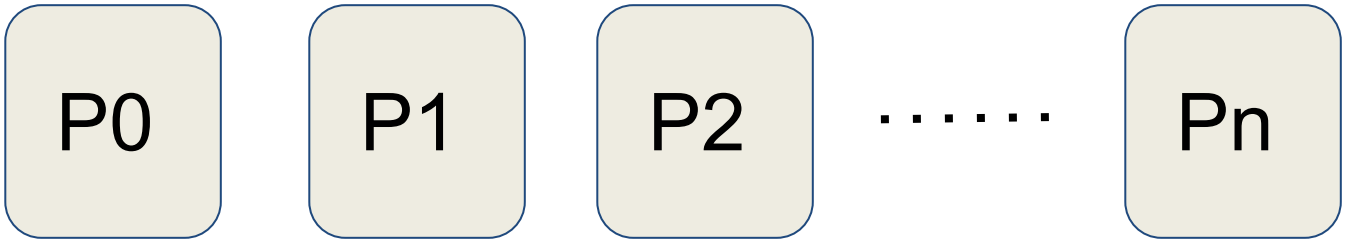
How MPI Works ..?

How MPI Works ..?



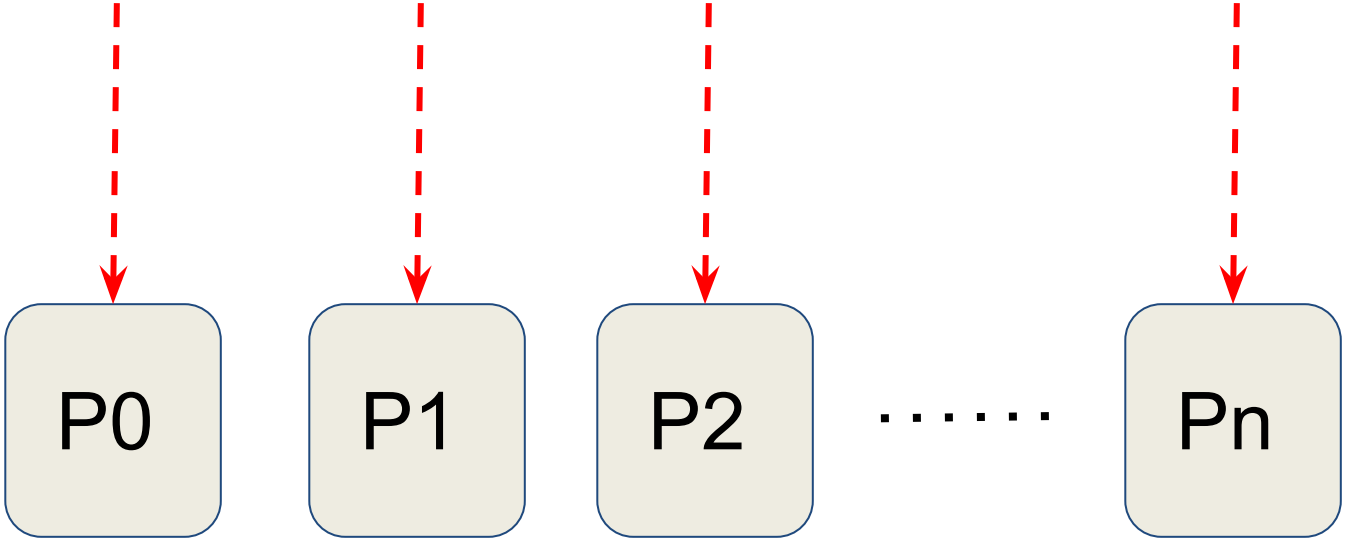
How MPI Works ..?

- **Creates Instances of same program on Every Processor involved..!**



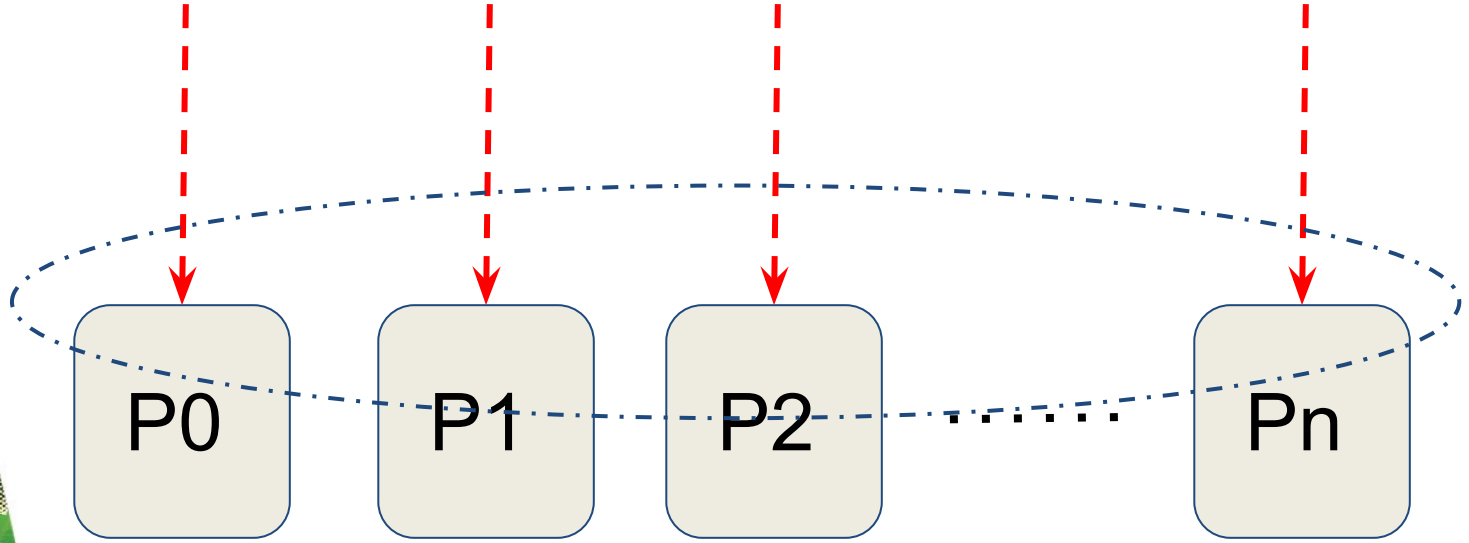
How MPI Works ..?

- Creates Instances of same program on Every Processor involved..!



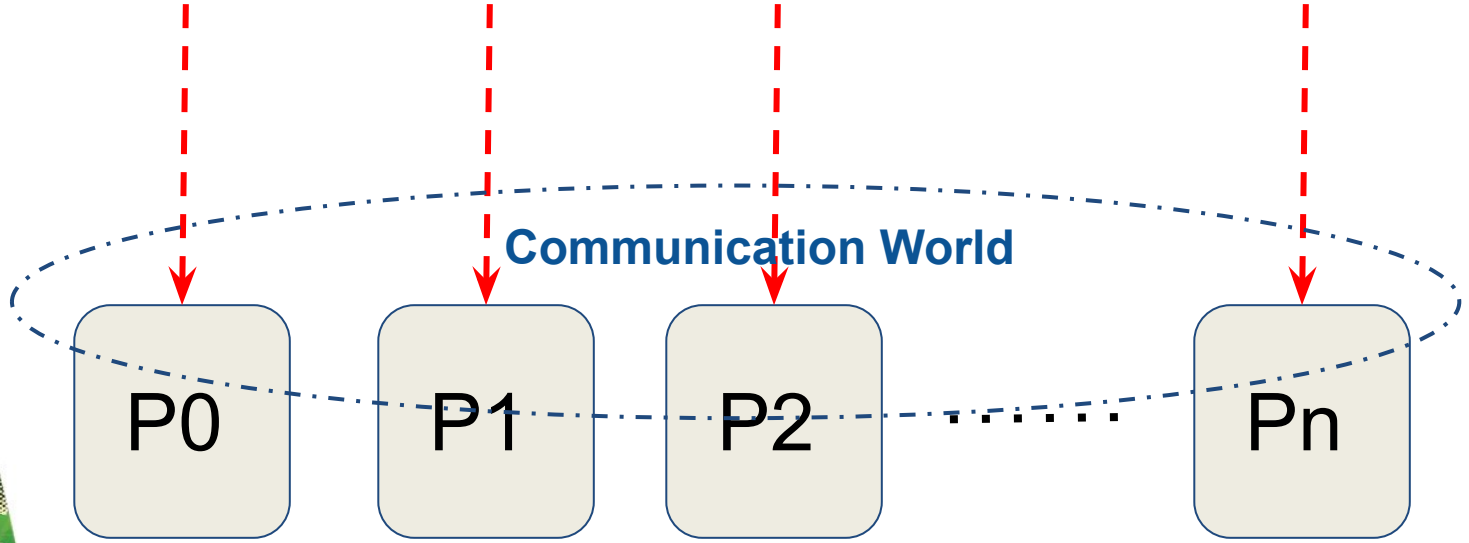
How MPI Works ..?

- Creates Instances of same program on Every Processor involved..!



How MPI Works ..?

- Creates Instances of same program on Every Processor involved..!



❖ Got it ?

❖ Got it ?

- Got it ?

- Let's try to understand with example...

- Got it ?

- Let's try to understand with example...

```
#include<stdio.h>
#include<string.h>
#include<mpi.h>
#define MASTER  0
Int main(void)
{
        char  greeting[MAX_STRING];
        int      comm_sz ;
        int      my_rank ;

        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

```
#include<stdio.h>
#include<string.h>
#include<mpi.h>
#define MASTER  0
Int main(void)
{
        char  greeting[MAX_STRING];
        int      comm_sz ;
        int      my_rank ;

        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

```
#include<stdio.h>
#include<string.h>
#include<mpi.h>
#define MASTER  0
Int main(void)
{
        char  greeting[MAX_STRING];
        int      comm_sz ;
        int      my_rank ;

        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

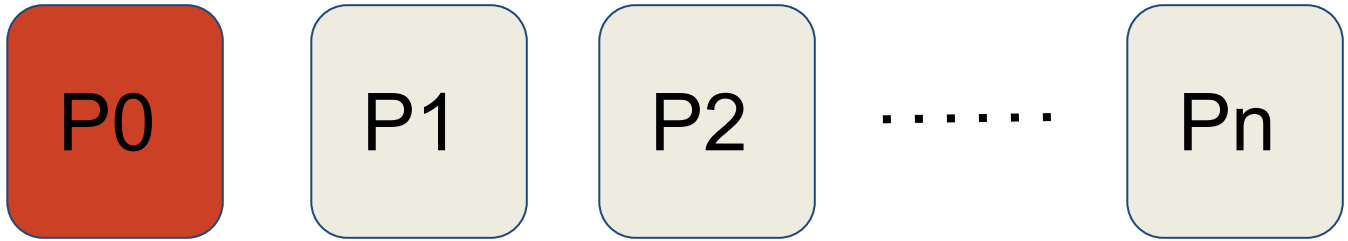
```
#include<stdio.h>
#include<string.h>
#include<mpi.h>
#define MASTER  0
Int main(void)
{
        char  greeting[MAX_STRING];
        int      comm_sz ;
        int      my_rank ;

        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

```
#include<stdio.h>
#include<string.h>
#include<mpi.h>
#define MASTER  0
Int main(void)
{
        char  greeting[MAX_STRING];
        int      comm_sz ;
        int      my_rank ;

        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

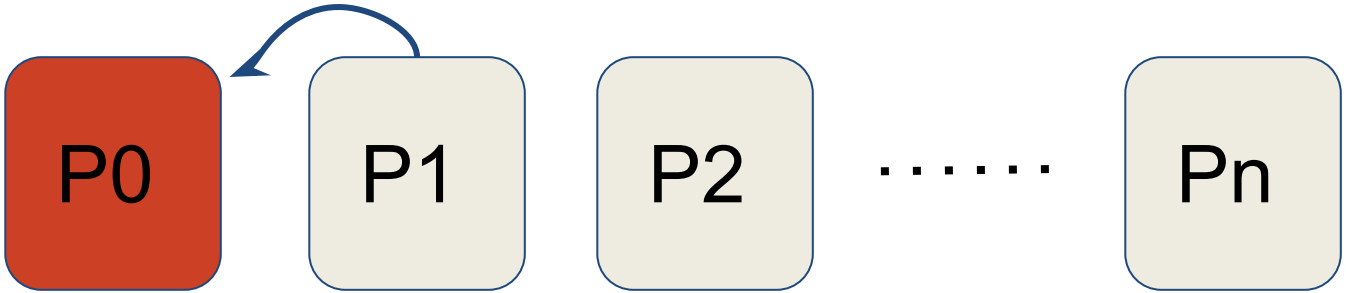
```
if(my_rank != MASTER)
{
        sprintf(greeting, “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD) ;
}
```

```
if(my_rank != MASTER)
{
        sprintf(greeting, “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD) ;
}
```

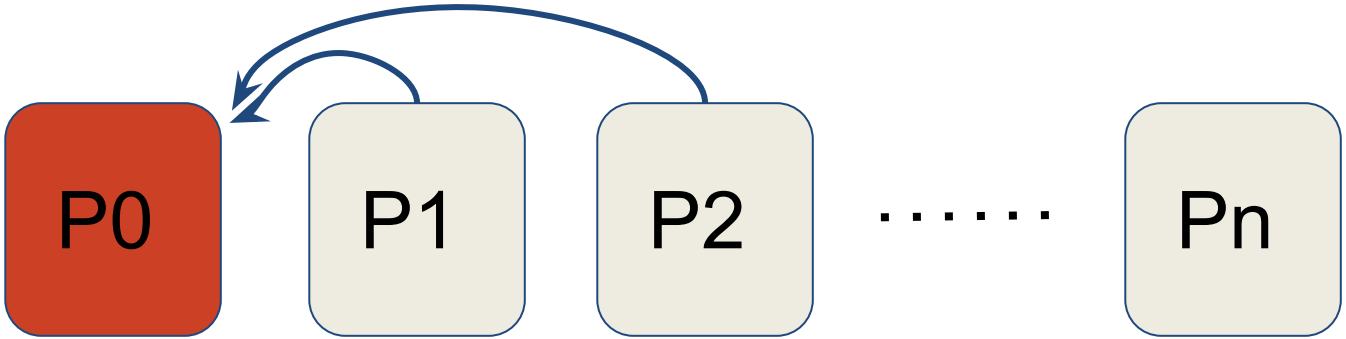
```
if(my_rank != MASTER)
{
        sprintf(greeting, “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD) ;
}
```



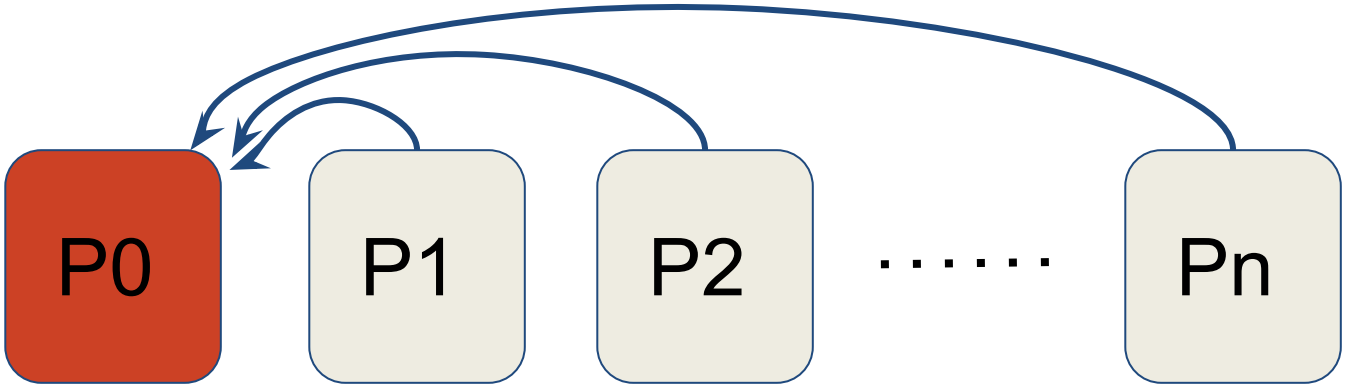
```
if(my_rank != MASTER)
{
        sprintf(greeting, “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD) ;
}
```



```
if(my_rank != MASTER)
{
        sprintf(greeting, “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD) ;
}
```



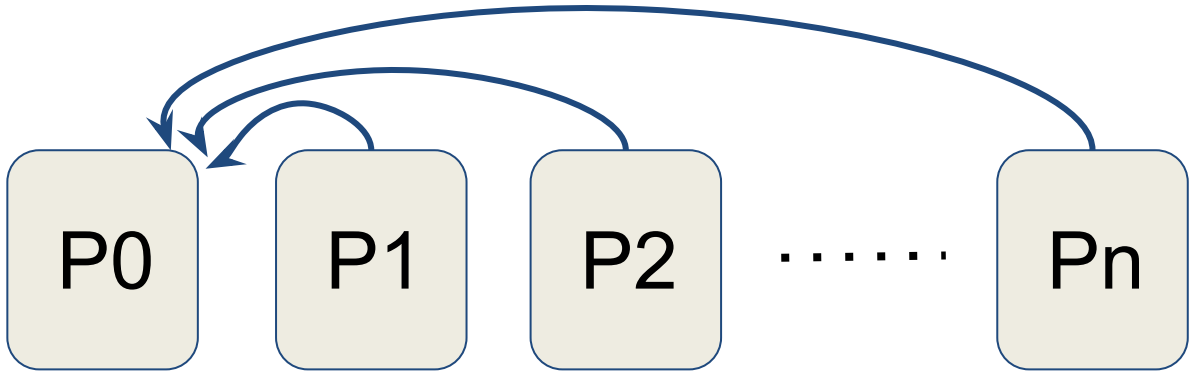
```
if(my_rank != MASTER)
{
        sprintf(greeting, “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD) ;
}
```



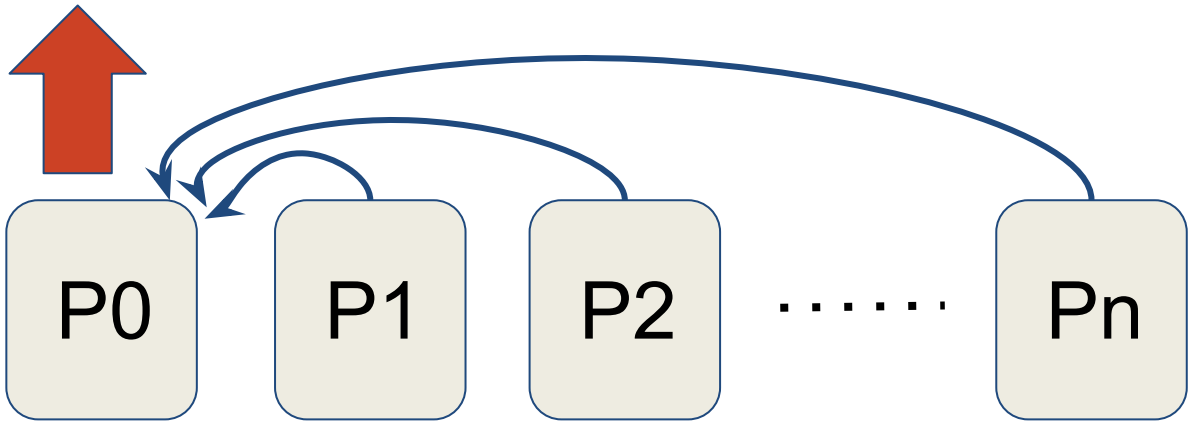
```
else
{
        printf( “Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ”,  my_rank, comm_sz) ;
        for(int q=1; q < comm_sz; q++)
        {
                MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD,
                                MPI_STATUS_IGNORE) ;
                printf(“%s \n”, greeting) ;
        }
 }

   MPI_Finalize();
   return 0;
}/* END */
```

```
else
{
        printf("Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ",  my_rank, comm_sz) ;
        for(int q=1; q < comm_sz; q++)
        {
        MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD,
                                MPI_STATUS_IGNORE) ;
                printf("%s \n", greeting) ;
        }
 }

   MPI_Finalize();
   return 0;
}/* END */
```

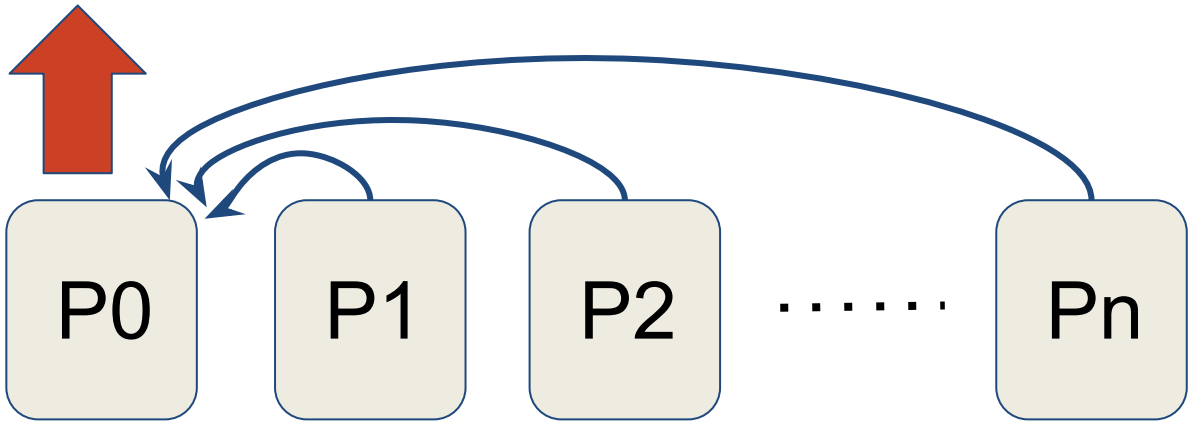
```
else
{
        printf("Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ",  my_rank, comm_sz) ;
        for(int q=1; q < comm_sz; q++)
        {
        MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD,
                                MPI_STATUS_IGNORE) ;
                printf("%s \n", greeting) ;
        }
 }

   MPI_Finalize();
   return 0;
}/* END */
```



```
else
{
        printf("Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ",  my_rank, comm_sz) ;
        for(int q=1; q < comm_sz; q++)
        {
        MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD,
                                MPI_STATUS_IGNORE) ;
                printf("%s \n", greeting) ;
        }
}

   MPI_Finalize();
   return 0;
}/* END */
```

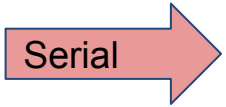


```
else
{
        printf("Welcome to the world of Parallel Computing. I am Process no %d out of %d
                        ",  my_rank, comm_sz) ;
        for(int q=1; q < comm_sz; q++)
        {
        MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD,
                                MPI_STATUS_IGNORE) ;
                printf("%s \n", greeting) ;
        }
}

MPI_Finalize();
  return 0;
}/* END */
```



❖ How to compile and run it ?

❖ How to compile and run it ?



❖ How to compile and run it ?

Serial

- $ gcc -o test test_serial.c
- $ ./test

❖ How to compile and run it ?

Serial

- $ gcc -o test test_serial.c
- $ ./test

Parallel

❖ How to compile and run it ?

Serial

- $ gcc -o test test_serial.c
- $ ./test

Parallel

- $ mpicc -o mpi_test mpi_test.c
- $ mpirun -np n ./mpi_test

❖ Output ..

- $ mpicc -o mpi_test mpi_test.c
- $ mpirun -np 4 ./mpi_test

❖ Output ..

- $ mpicc -o mpi_test mpi_test.c
- $ mpirun -np 4 ./mpi_test

❖ Output ..

- $ mpicc -o mpi_test mpi_test.c
- $ mpirun -np 4 ./mpi_test

Welcome to the world of Parallel Computing. I am Process no 0 out of 4
Welcome to the world of Parallel Computing. I am Process no 1 out of 4
Welcome to the world of Parallel Computing. I am Process no 3 out of 4
Welcome to the world of Parallel Computing. I am Process no 2 out of 4

❖ Got it ?

❖ Got it ?

-np 4

❖ Got it ?

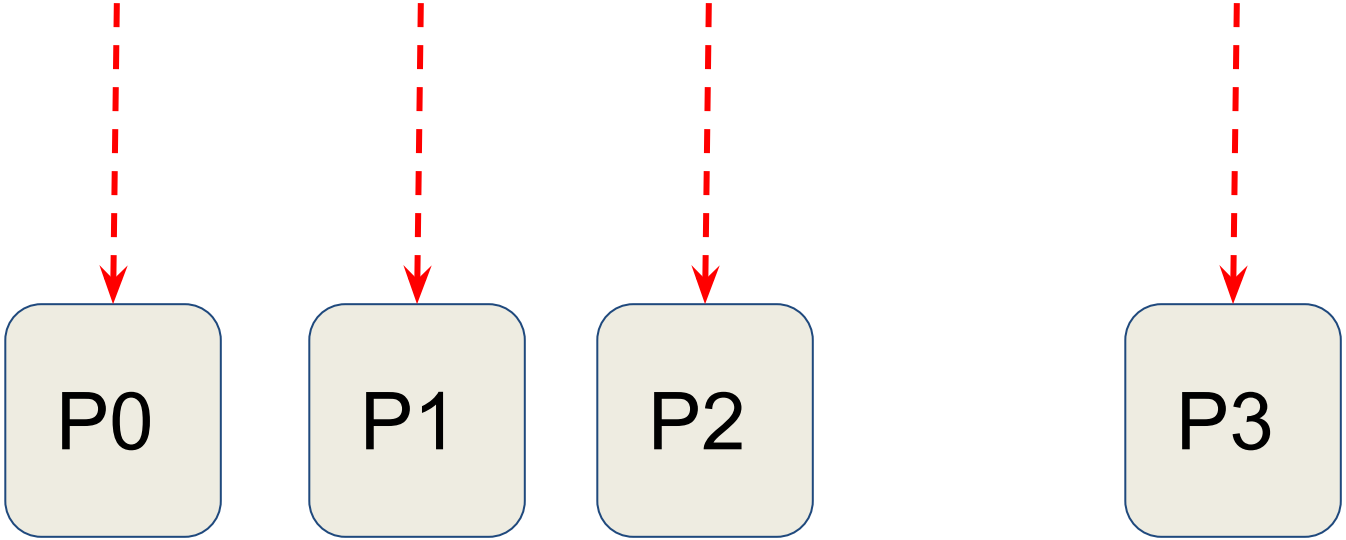
P0

P1

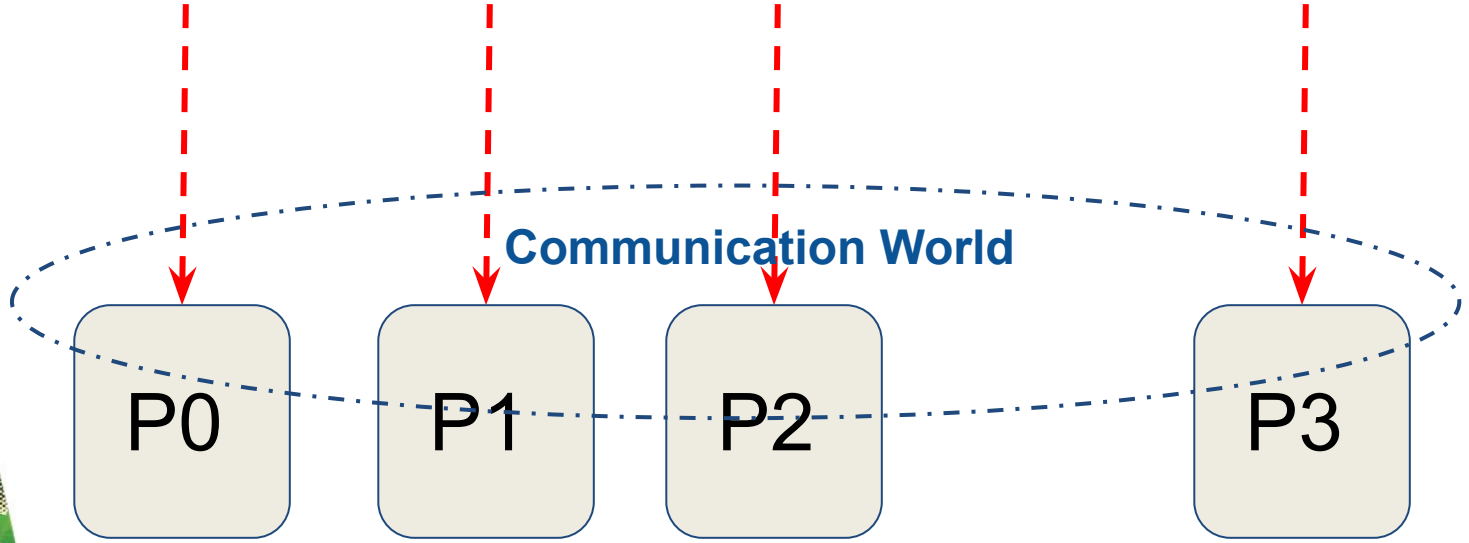
P2

P3

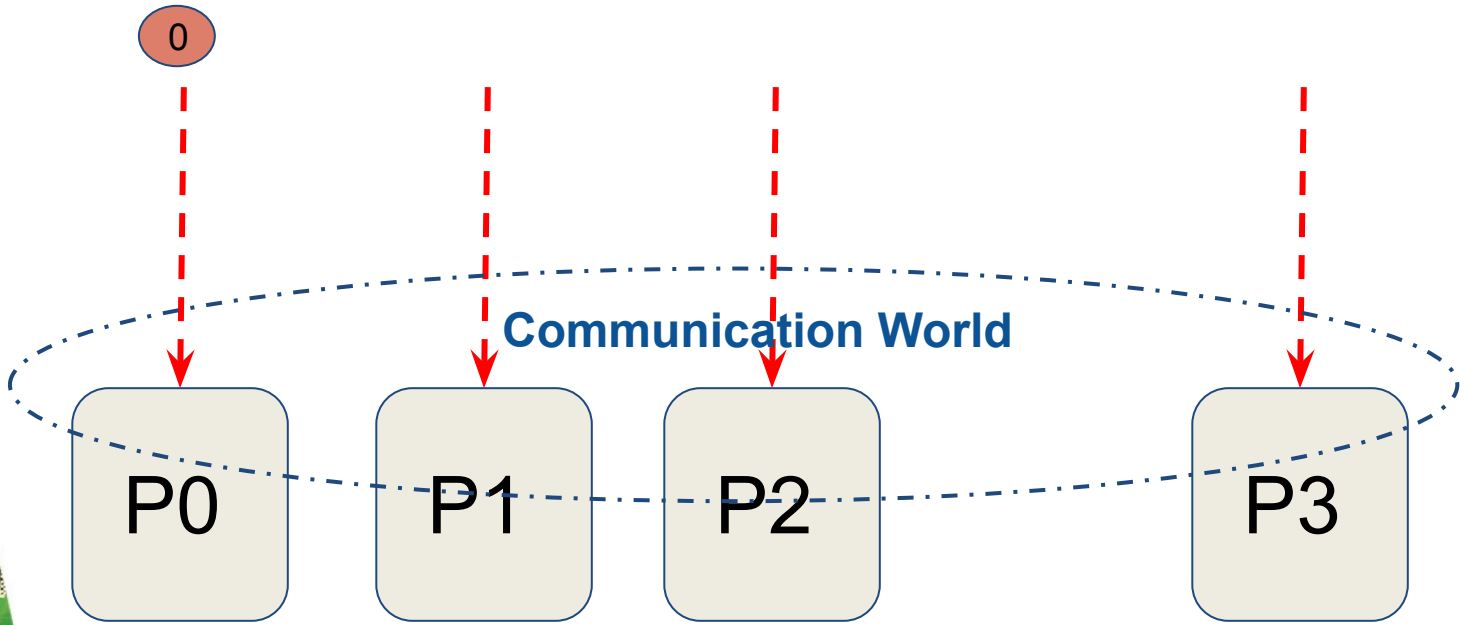
❖ Got it ?



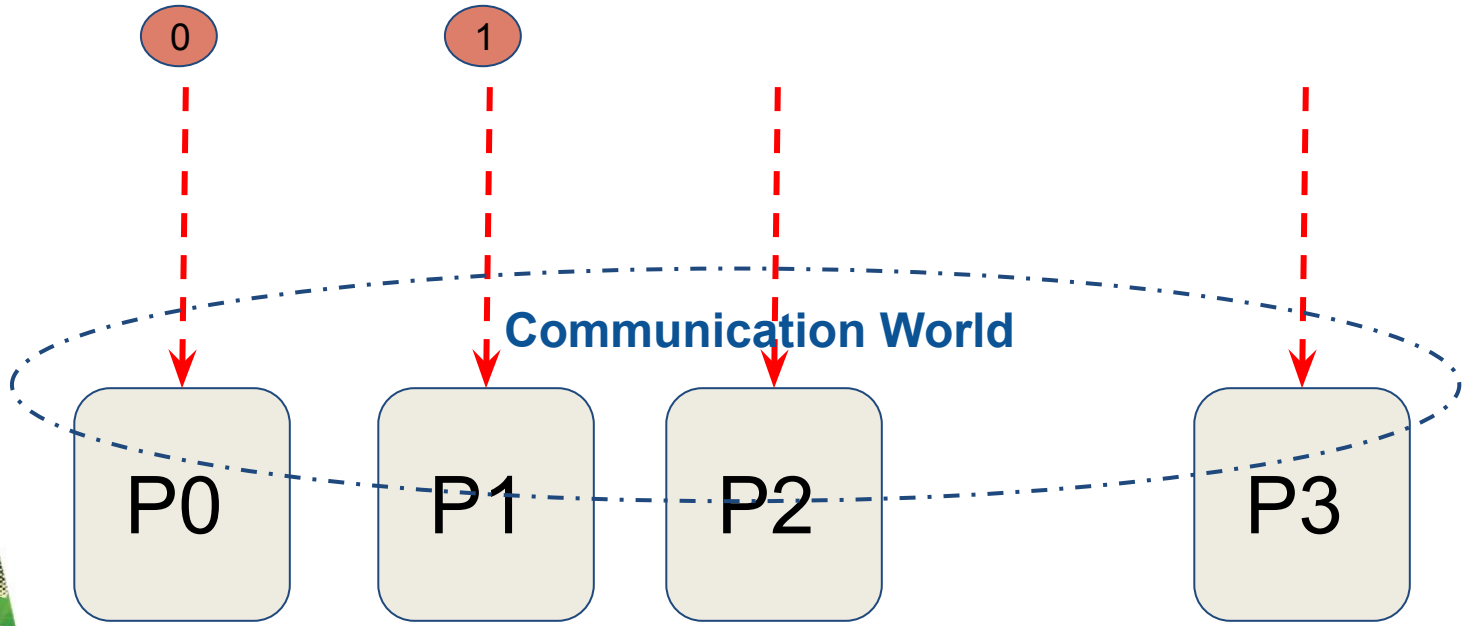
❖ Got it ?



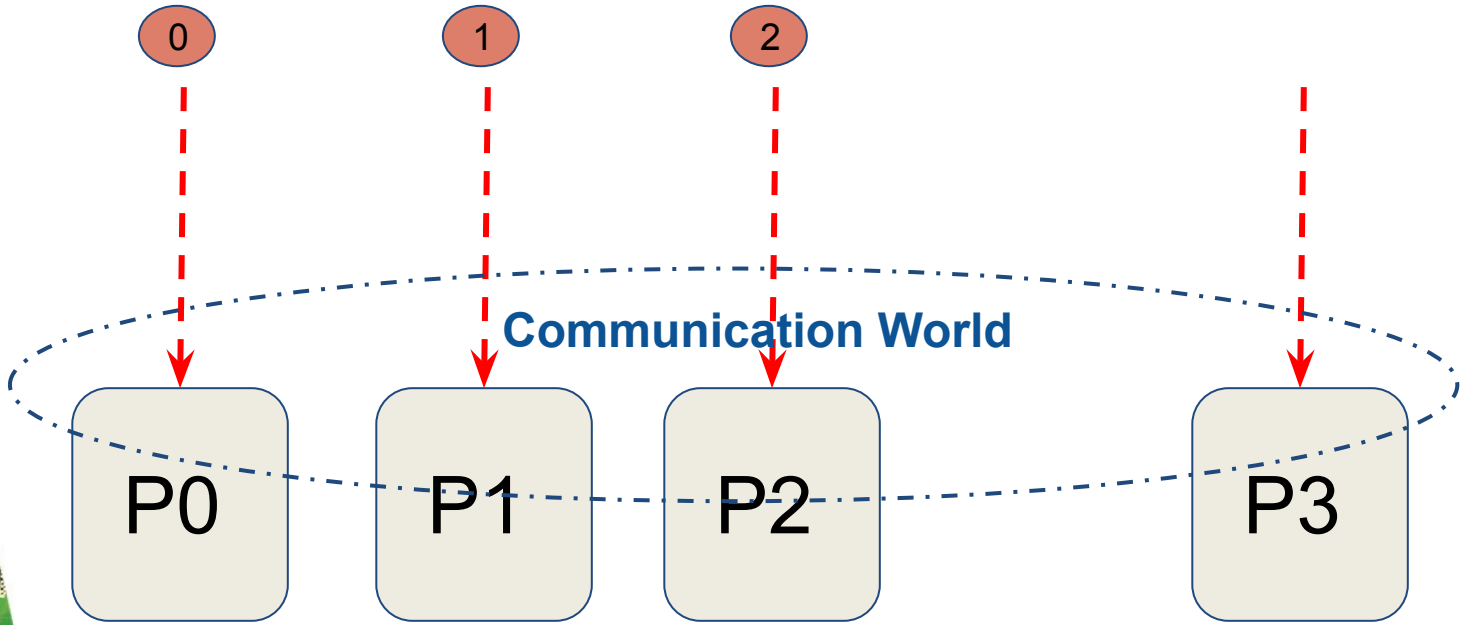
❖ Got it ?



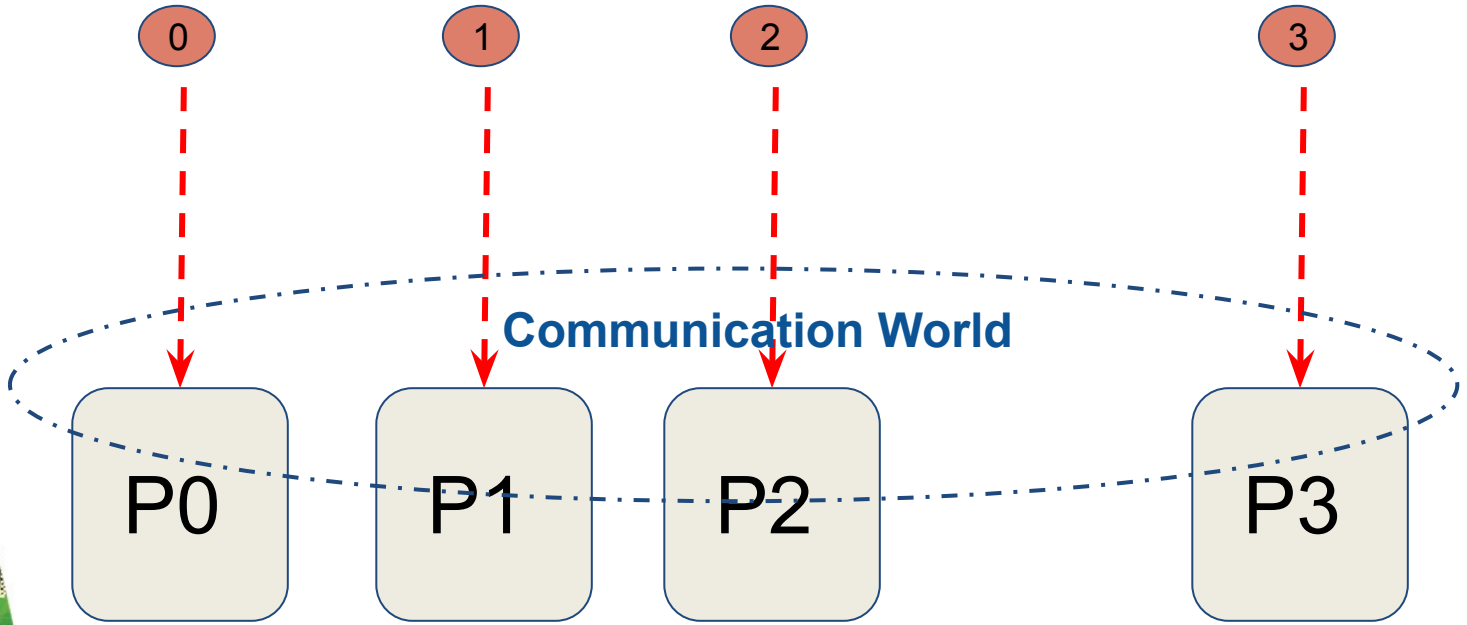
❖ Got it ?



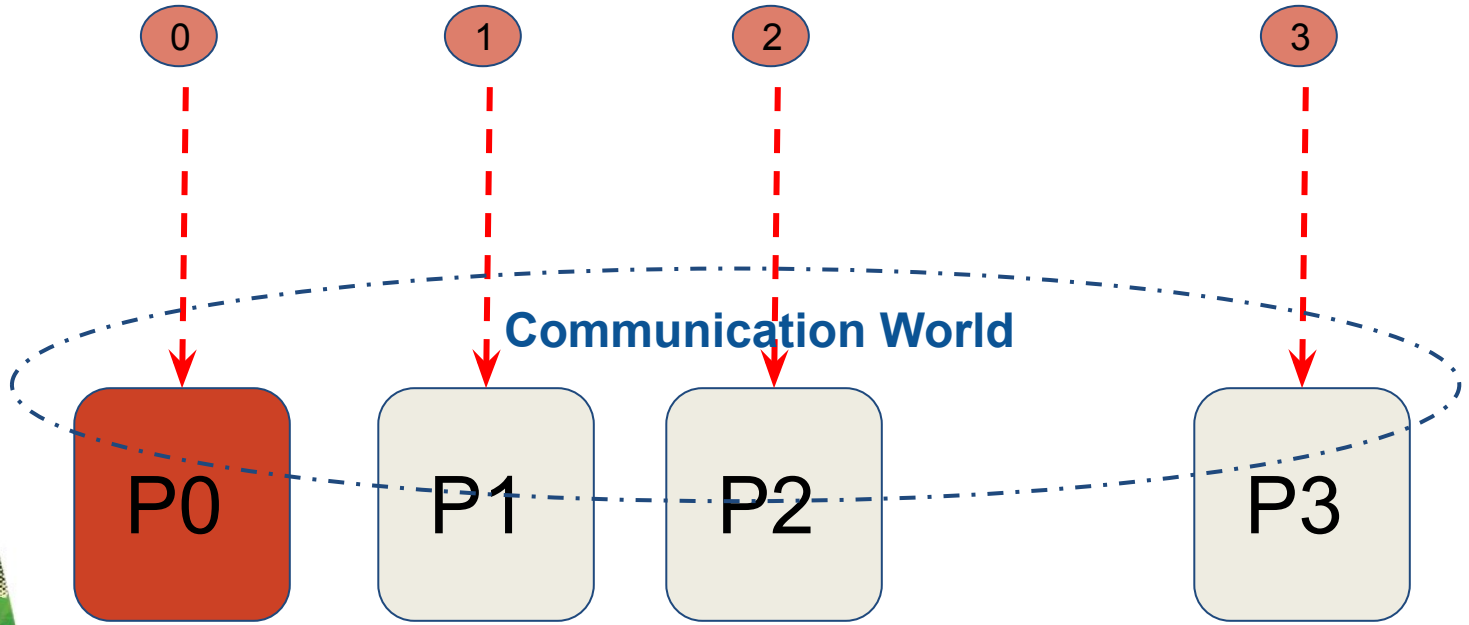
❖ Got it ?



❖ Got it ?



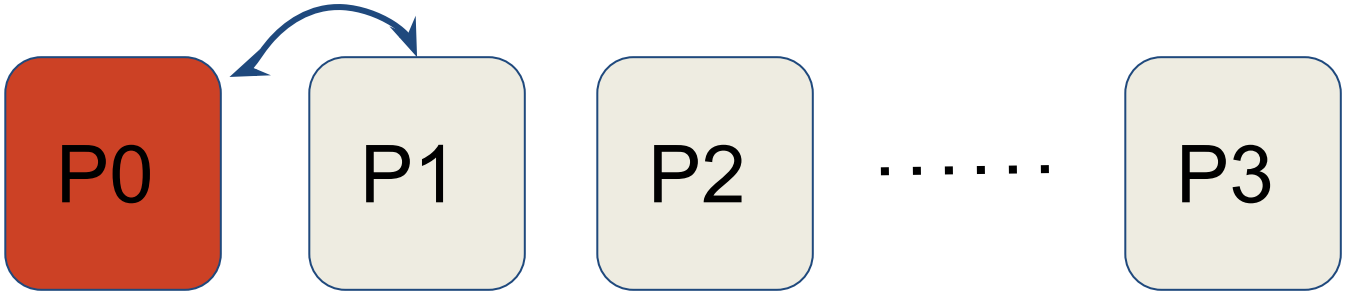
❖ Got it ?



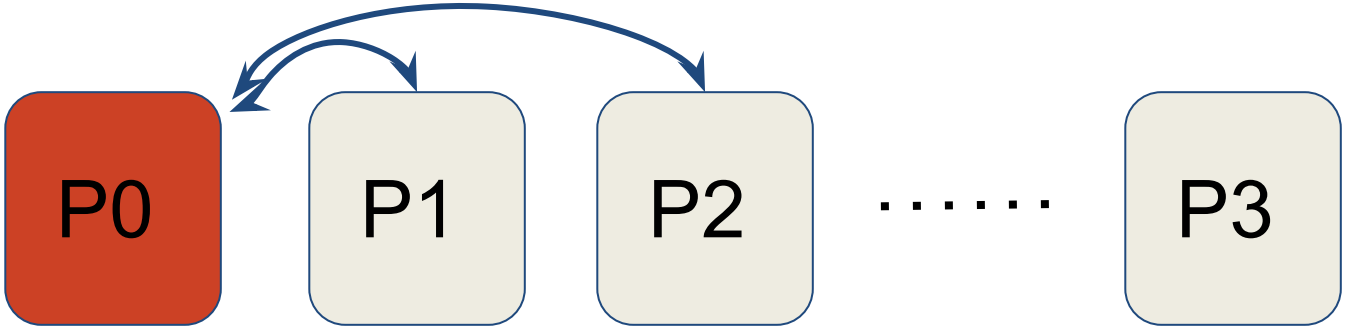
❖ Got it ?

P0 P1 P2 ..... P3

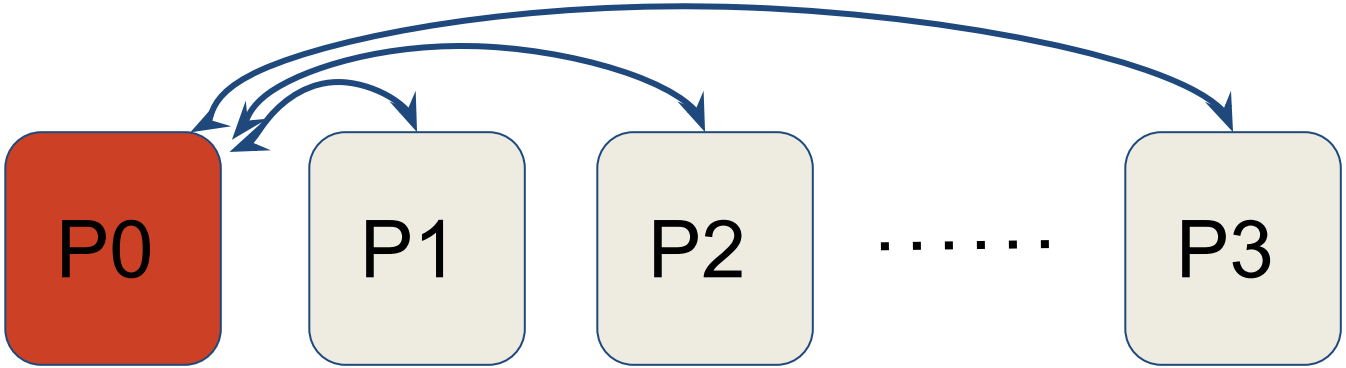
❖ Got it ?



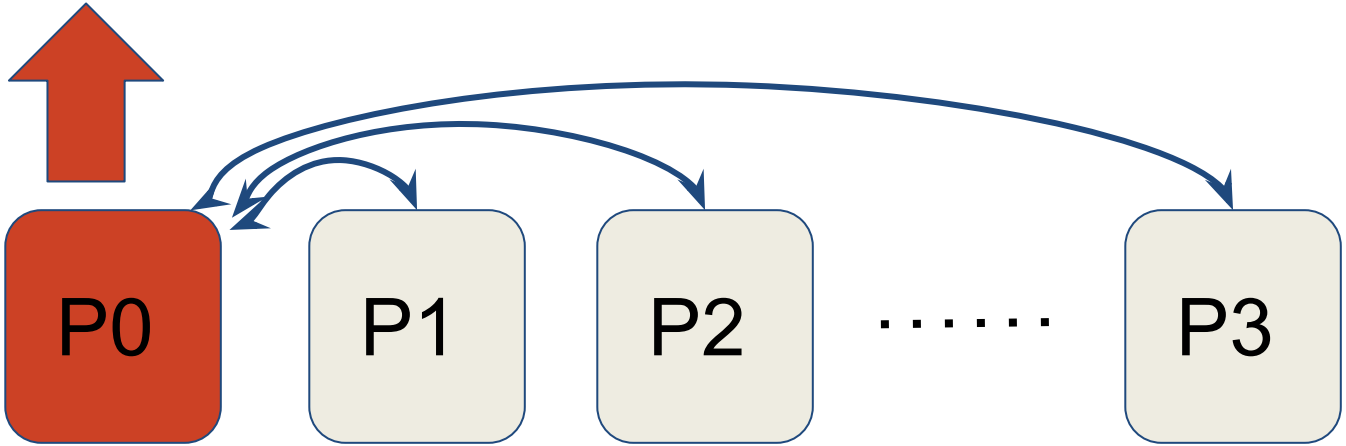
❖ Got it ?



❖ Got it ?



❖ Got it ?



❖ Got it ?

❖ Got it ?

MPI - Message Passing Interface

MPI is built on ‘Routines’

MPI - Message Passing Interface

MPI is built on ‘Routines’

The basic MPI Routines :-

- MPI_Init () ;
- MPI_Comm_rank () ;
- MPI_Comm_size () ;
- MPI_Send () ;
- MPI_Recv () ;
- MPI_Finalize () ;

- - - -- - - - - - - - -

MPI - Program structure

MPI - Program structure

```
#include <mpi.h>

main( int argc, char** argv )
{
    MPI_Init( &argc, &argv );

    /* main part of the program */

  /*
    Use MPI function call depend on your data
  partitioning and the parallelization
  architecture
  */

    MPI_Finalize();
}
```

MPI - Program structure

```
#include <mpi.h>

main( int argc, char** argv )
{
    MPI_Init( &argc, &argv );

    /* main part of the program */

  /*
    Use MPI function call depend on your data
  partitioning and the parallelization
  architecture
  */

    MPI_Finalize();
}
```

Hope so you got it...!!!

MPI - Program structure

```
#include <mpi.h>

main( int argc, char** argv )
{
    MPI_Init( &argc, &argv );

    /* main part of the program */

  /*
    Use MPI function call depend on your data
  partitioning and the parallelization
  architecture
  */

    MPI_Finalize();
}
```

Hope so you got it...!!!

Recap --

Recap --

- Parallel programming

Recap --

- Parallel programming
- General parallel programming models

Recap --

- Parallel programming
- General parallel programming models
- MPI

Recap --

- Parallel programming
- General parallel programming models
- MPI
- Need of MPI

Recap --

- Parallel programming
- General parallel programming models
- MPI
- Need of MPI
- How it works ..?

Recap --

- Parallel programming
- General parallel programming models
- MPI
- Need of MPI
- How it works ..?
- Understanding of Basic MPI routines with Example

Recap --

- Parallel programming
- General parallel programming models
- MPI
- Need of MPI
- How it works ..?
- Understanding of Basic MPI routines with Example
- General MPI program structure

Recap --

- Parallel programming
- General parallel programming models
- MPI
- Need of MPI
- How it works ..?
- Understanding of Basic MPI routines with Example
- General MPI program structure


Thank
You

Open source Versions of MPI -

- Two popular free versions of MPI are MPICH2 and OpenMPI.

  - MPICH - basis for derivative of MPI implementations to meet special purpose needs.
  - **Network Technology support :** One common complaint about MPICH is that it does not support InfiniBand, However, **MVAPICH** and **Intel MPI** - both of which are MPICH derivatives - support InfiniBand. (Cray Seastar, Gemini, Arise, IBM Blue gene,). *MVAPICH2 - preferred implementation in nearly all cases.
  - **Feature support :** -- ☆☆☆☆☆
  - **Process Management :** --☆☆

  - OpenMPI - targets the common case, both in terms of usage and network conduits.
  - **Network Technology support :** openMPI support InfiniBand. (Cray Gemini, *but not by Cray)
  - **Feature support :** --☆☆
  - **Process Management :** -- ☆☆☆☆☆