Introduction to Message Passing Interface

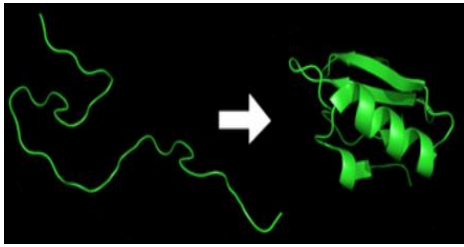
Mr. Om Jadhav
Senior Technical officer,
HPC Tech CDAC Pune
(omjadhav@cdac.in)

Agenda

- Why Parallel Computing ?
- Parallel programing Architectures/Model
- MPI - Message Passing Interface
  - What is MPI ?, Need and Evolution of MPI.
  - MPI Execution flow
  - Basic MPI routines
  - MPI program - Compile and Execution
- Point to Point Communication
  - Blocking P2P
  - Non Blocking P2P
- Collective communication
  - Basic Collective 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 .....
- ......

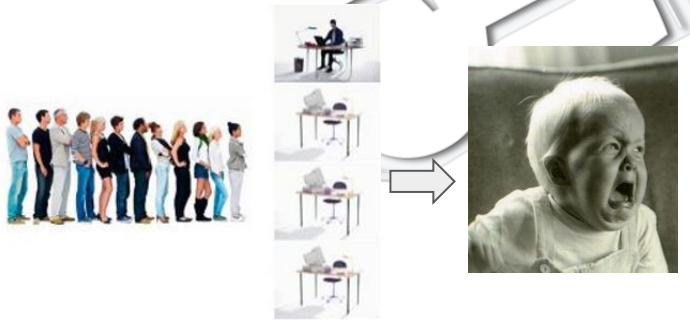


Why Parallel Computing ?

- Aren't single processor systems fast enough ?

Why Parallel Computing ?

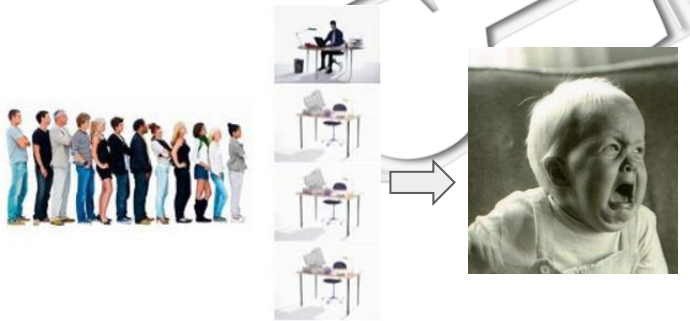
- **Aren't single processor systems fast enough ?**



Serial Computing

Why Parallel Computing ?

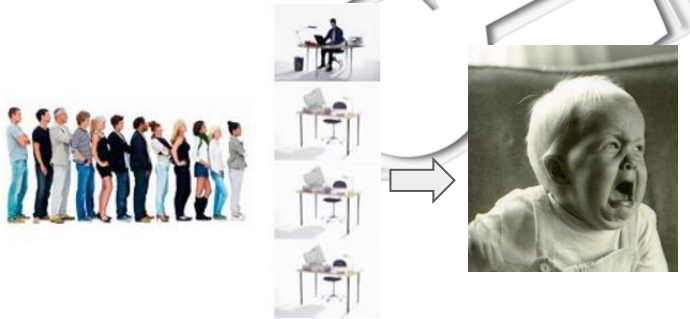
- Aren't single processor systems fast enough ?
- Why to build parallel systems ? Why build systems with multiple processors ?



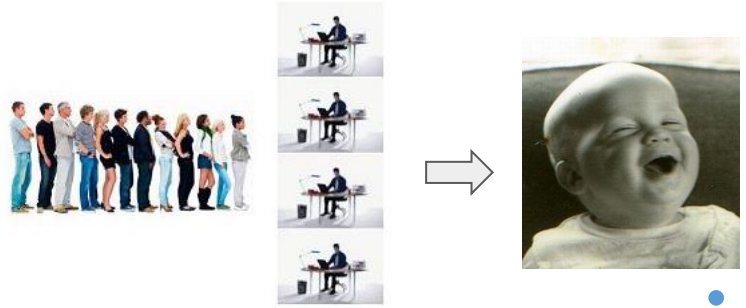
Serial Computing

Why Parallel Computing ?

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



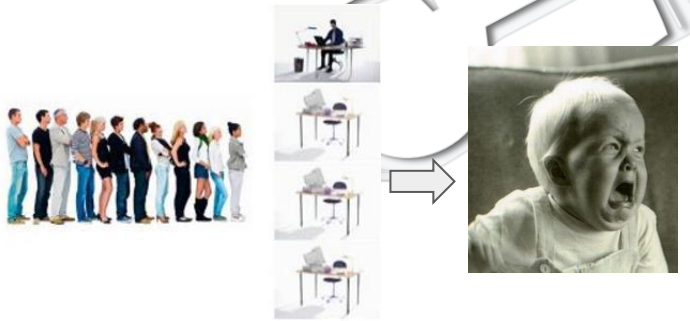
Serial Computing



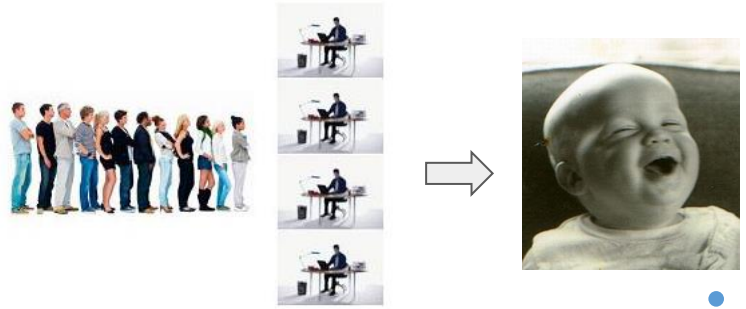
Parallel Computing

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 ?



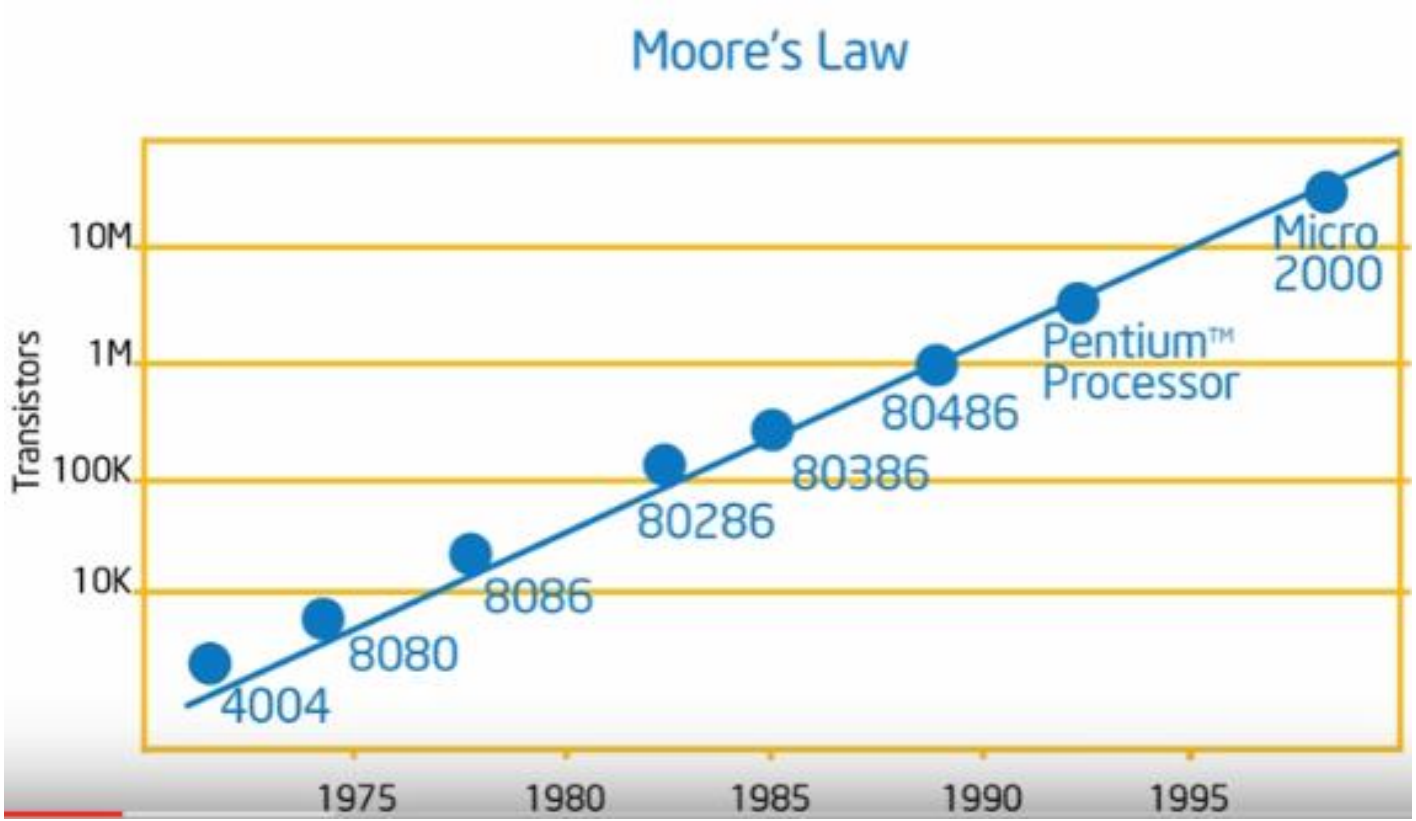
Serial Computing



Parallel Computing

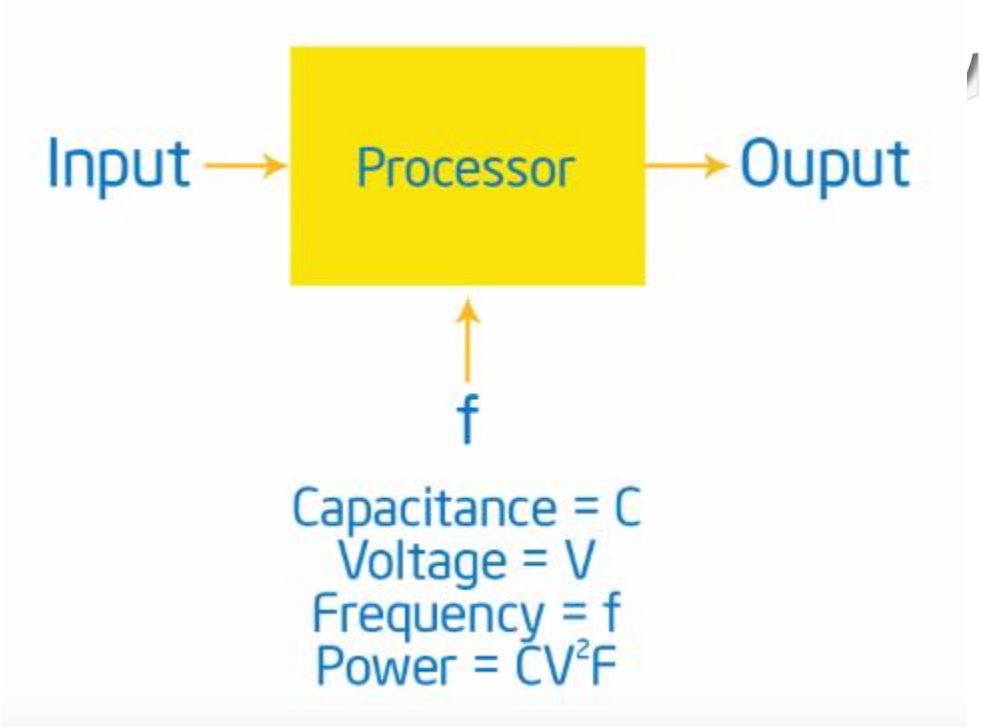
What Moore's Law tells.. ?

What Moore's Law tells.. ?



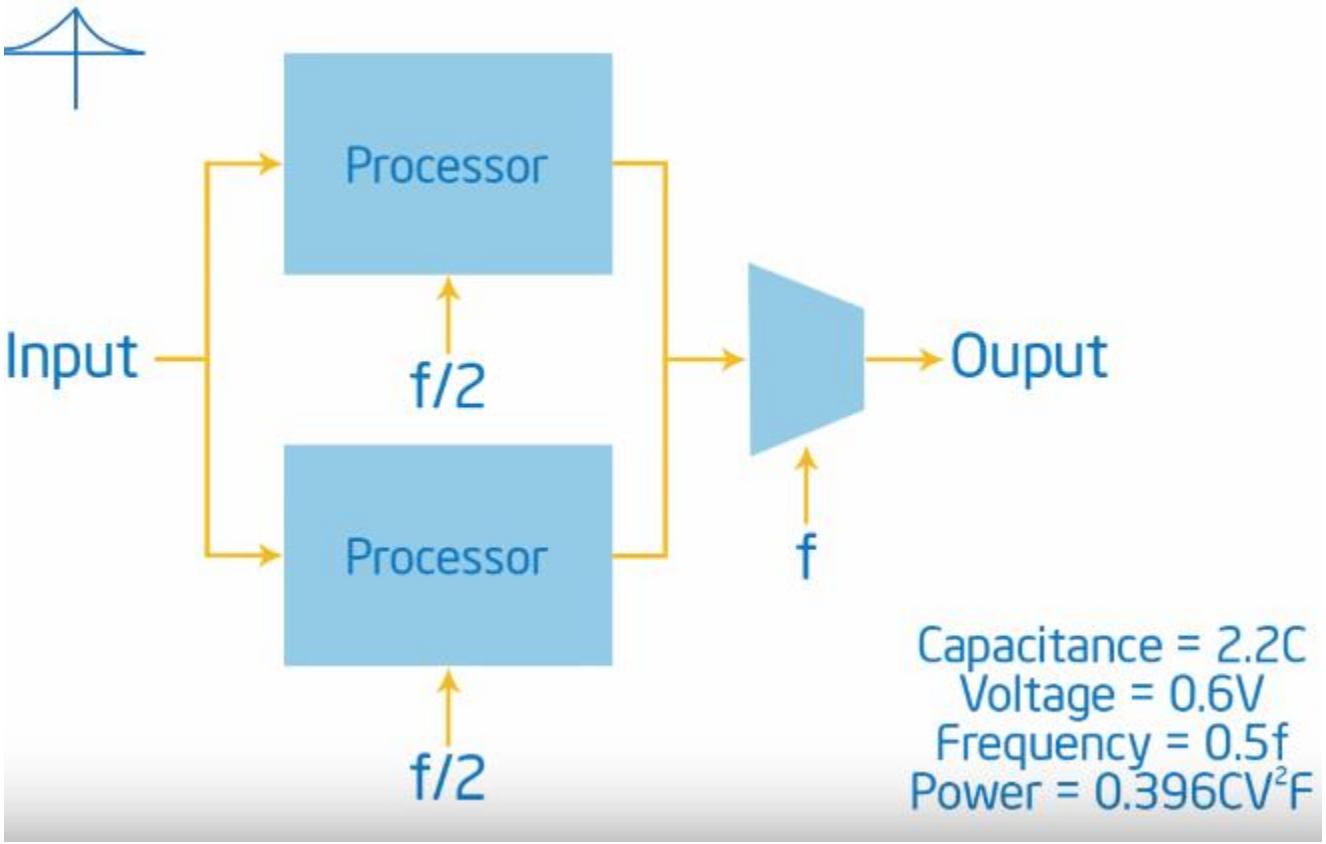
Uniprocessor ?

Uniprocessor ?



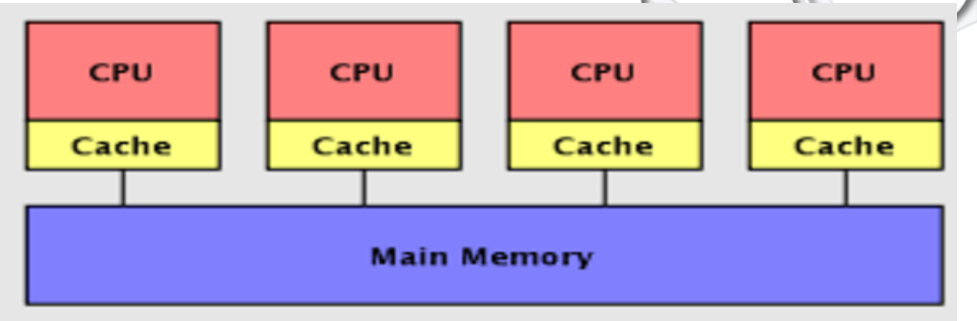
Parallel Architecture ?

Parallel Architecture ?



Parallel Programming Models..

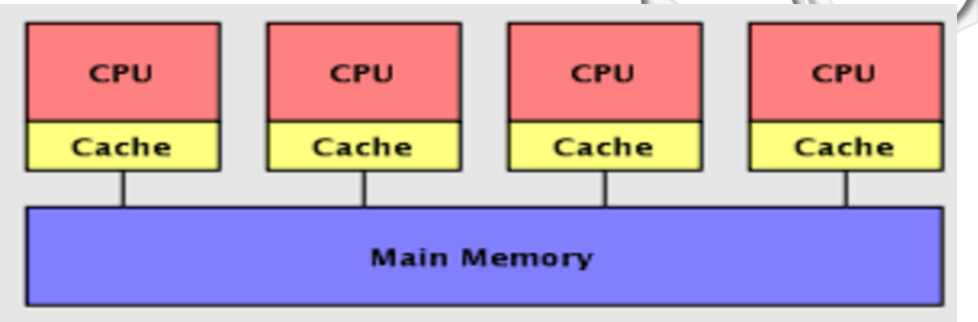
❑ Shared-memory Model



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

Parallel Programming Models..

❑ Shared-memory Model

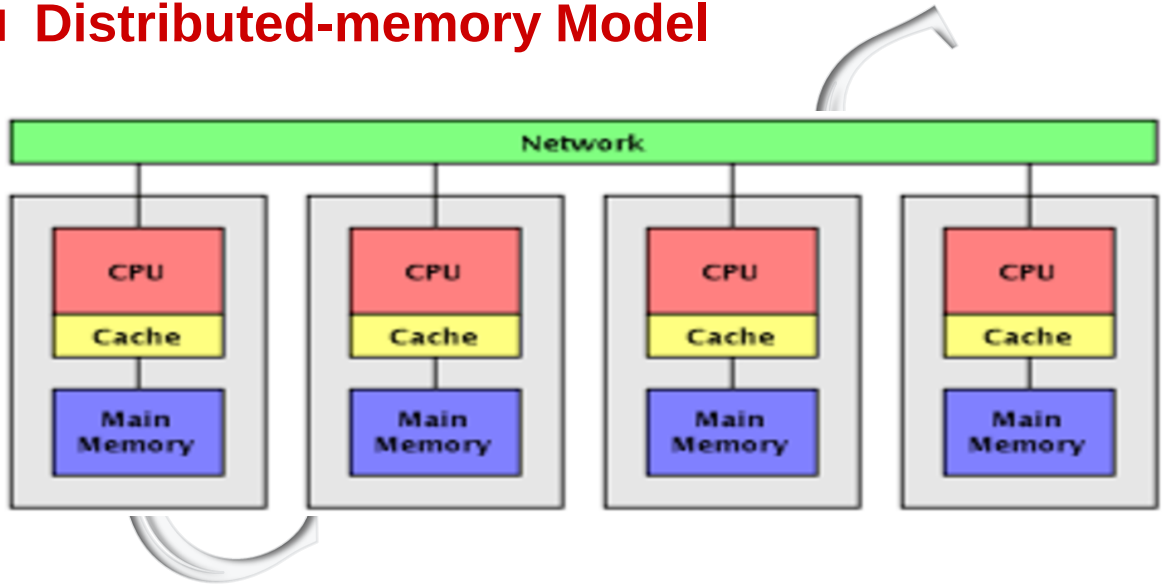


- UMA - Uniform Memory Access
- NUMA - Non-Uniform Memory Access

❖ openMP
❖ Pthreads ...

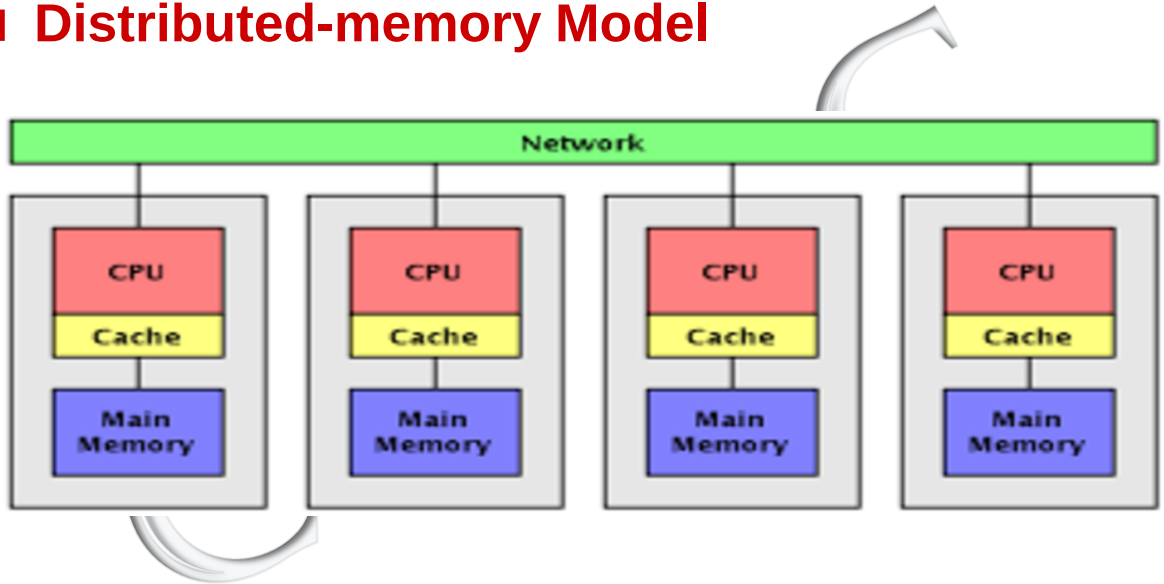
Parallel Programming Models..

- **Distributed-memory Model**



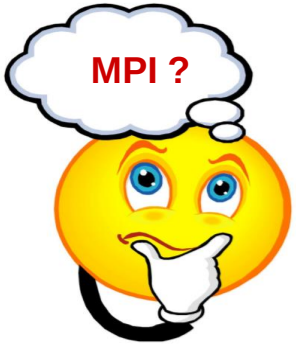
Parallel Programming Models..

- **Distributed-memory Model**



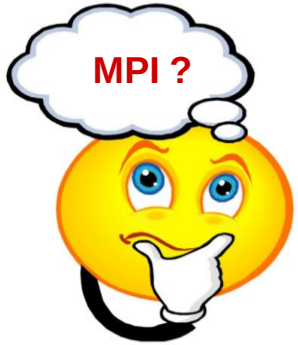
- **MPI - Message Passing Interface**

MPI - Message Passing Interface

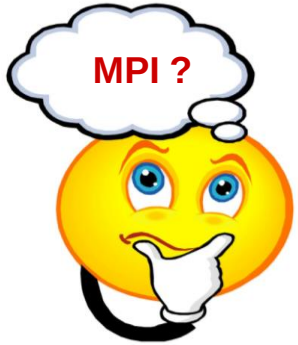


MPI - Message Passing Interface

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

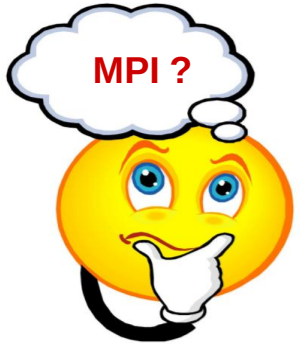


MPI - Message Passing Interface

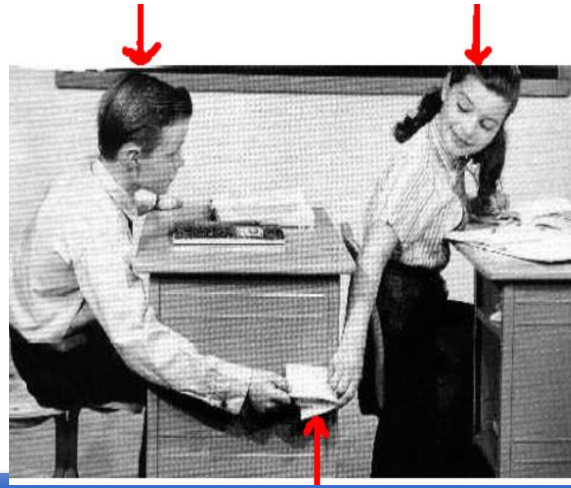


- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process

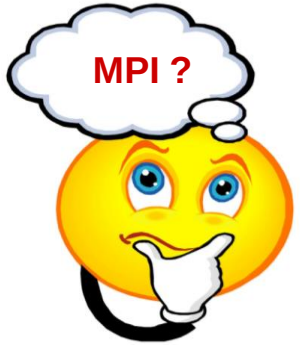
MPI - Message Passing Interface



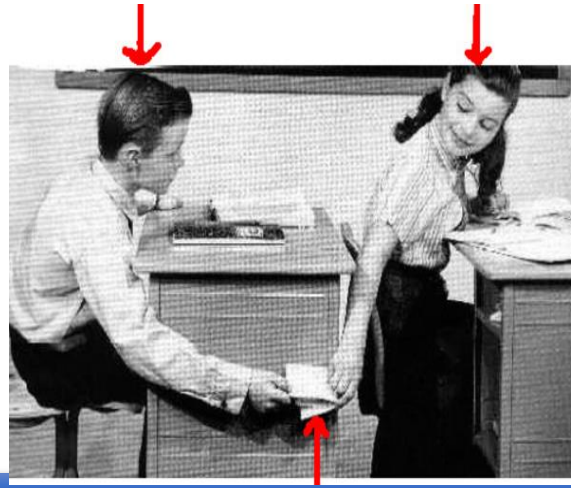
- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process



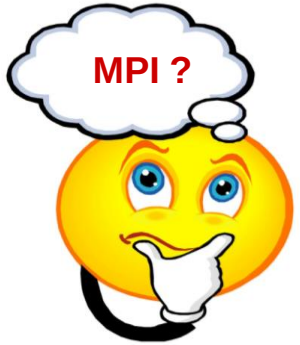
MPI - Message Passing Interface



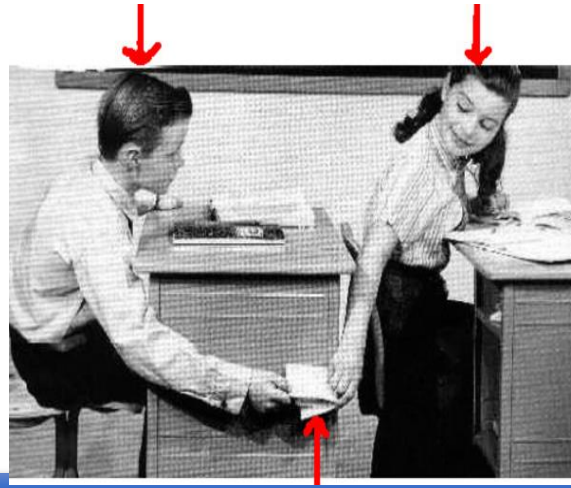
- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process
- MPI is based on Routines.



MPI - Message Passing Interface



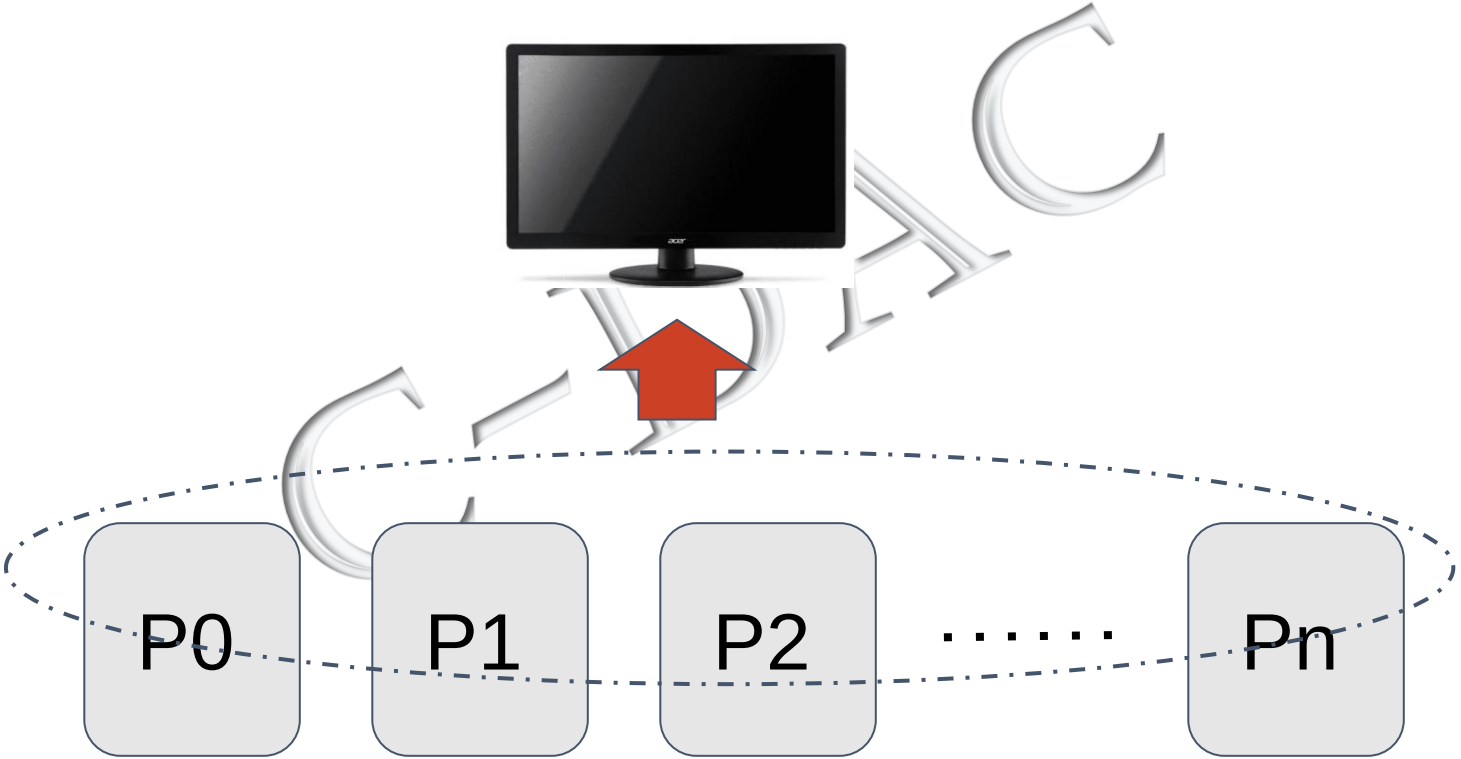
- The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum
- In MPI a Message is passed from one process to another process
- MPI is based on Routines.
- MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard"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



Wait....

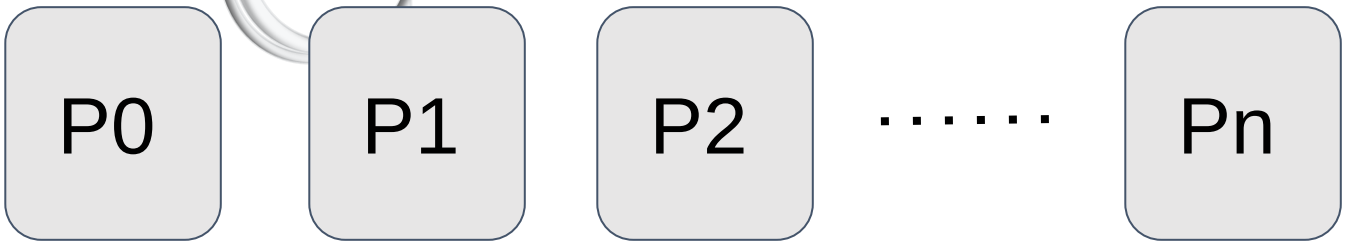
- What is Process ?
- Is MPI a new programming Language ..?

The Goal ..?

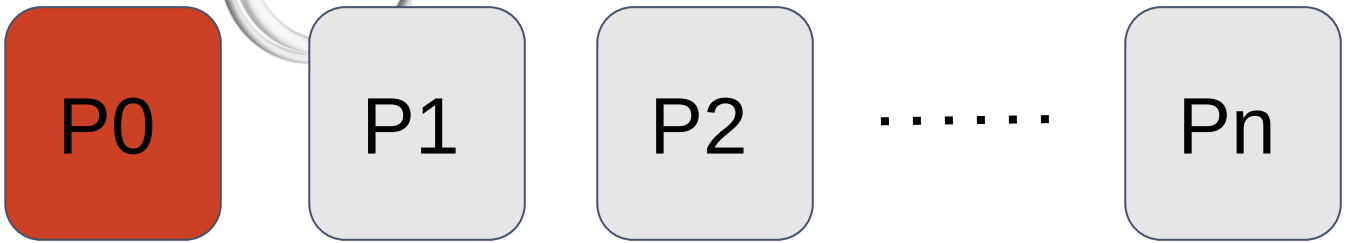


How to Achieve it ..?

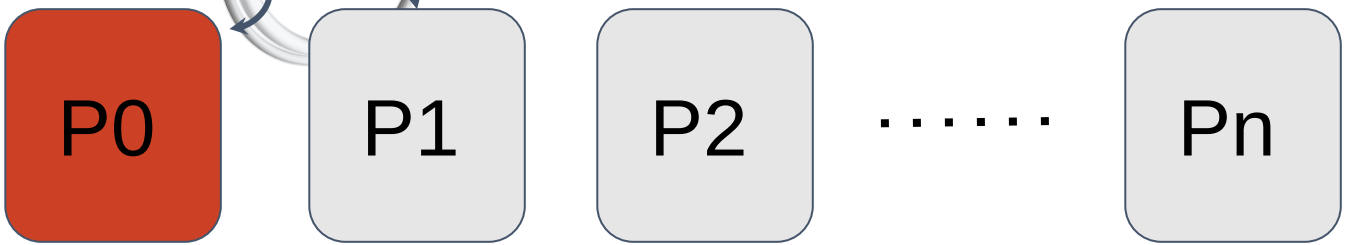
How to Achieve it ..?



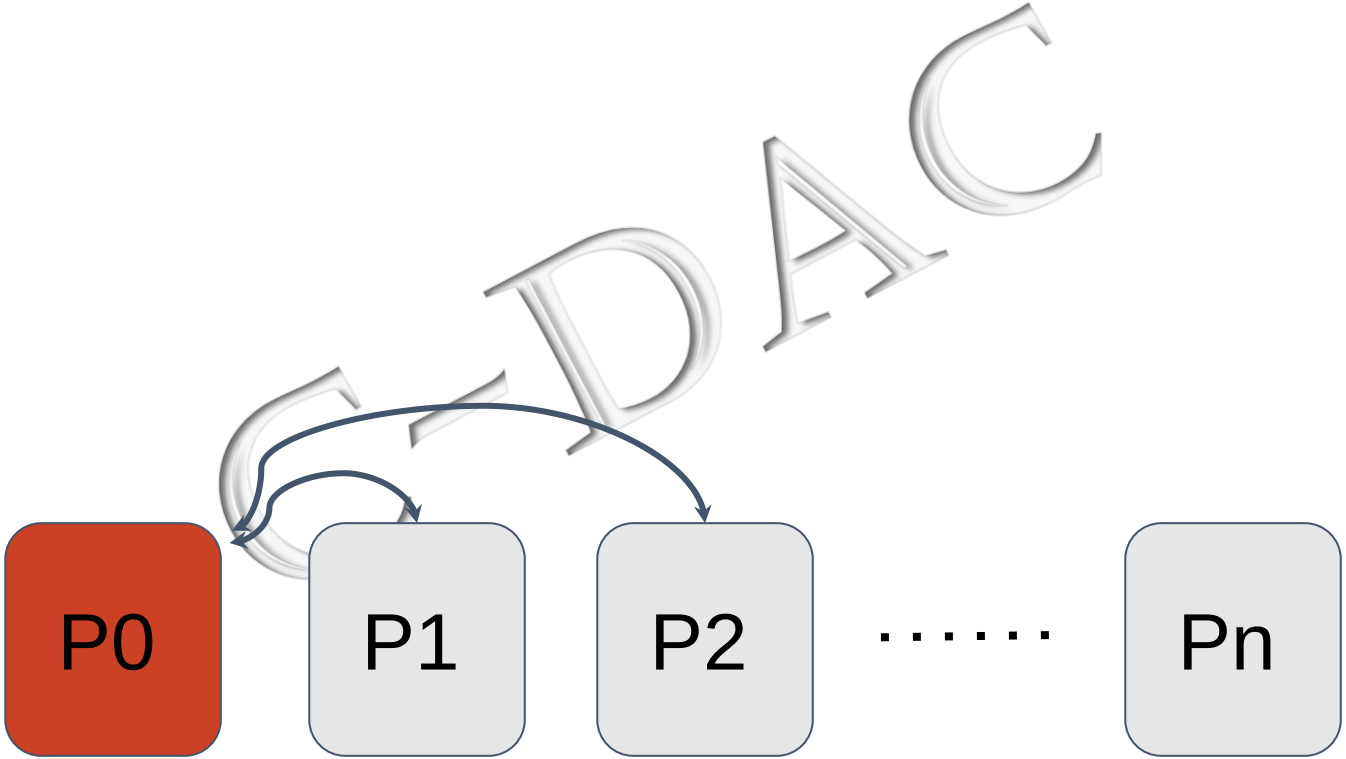
How to Achieve it ..?



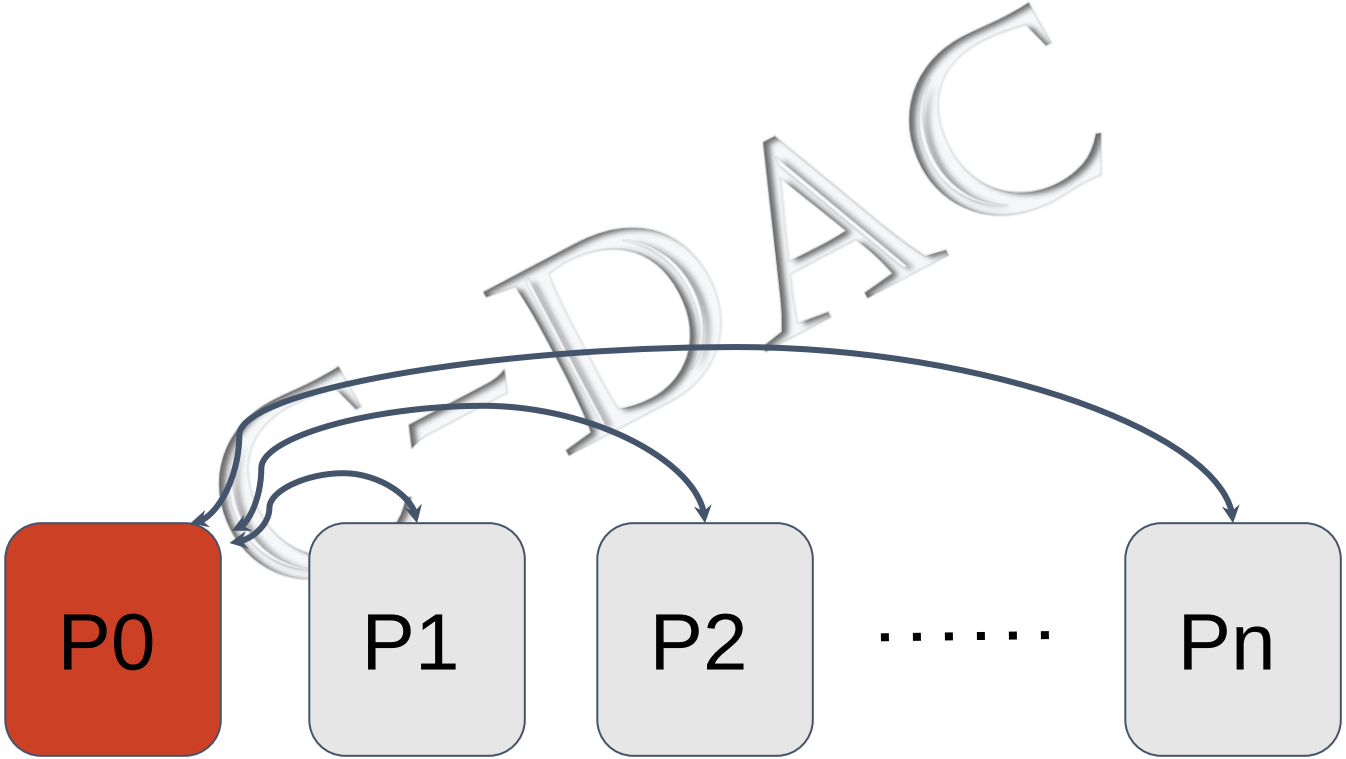
How to Achieve it ..?



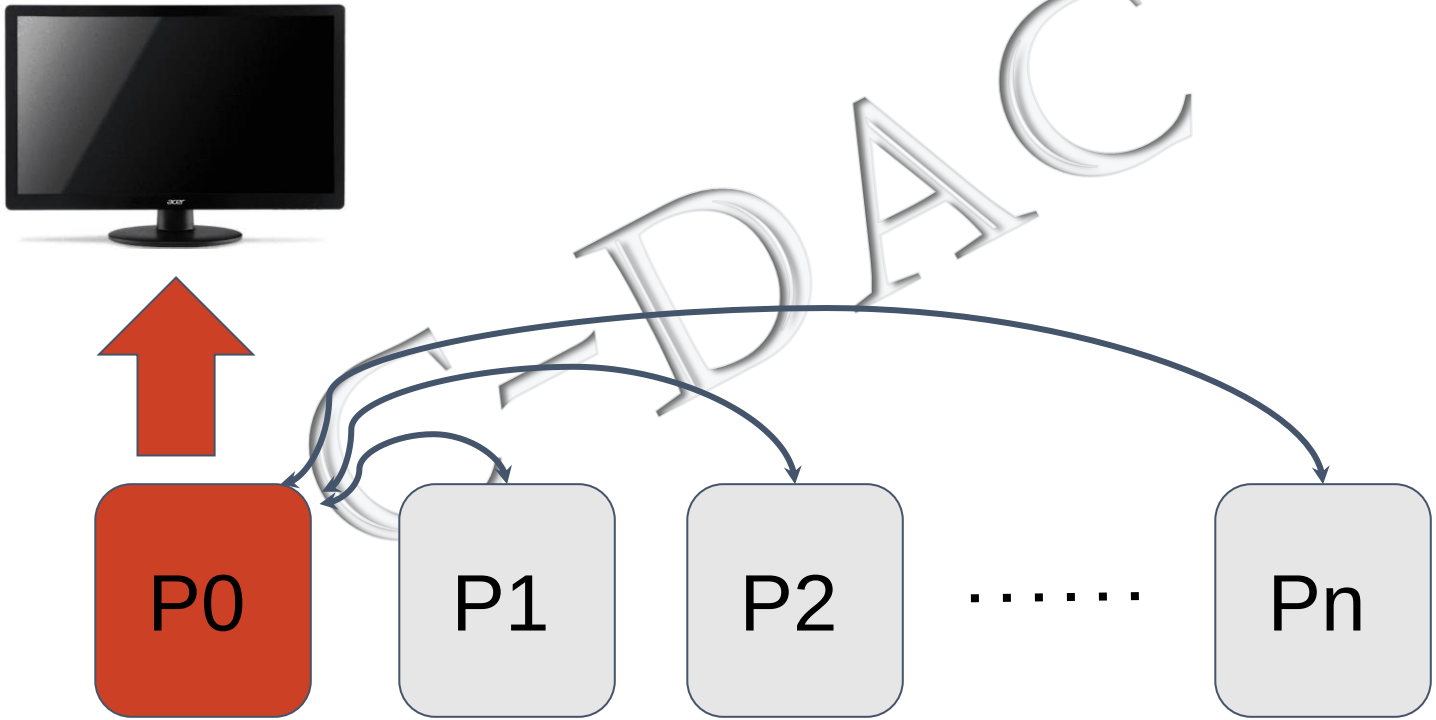
How to Achieve it ..?



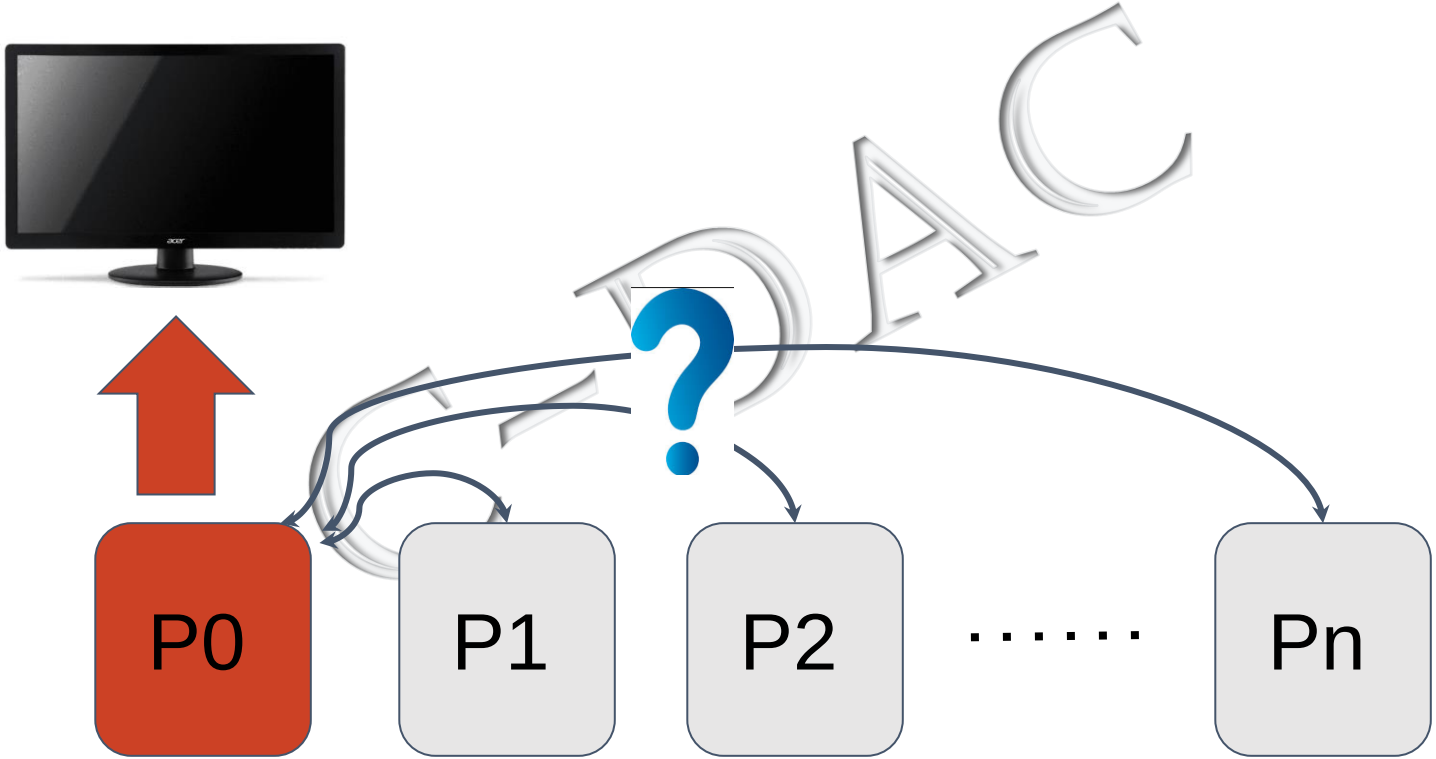
How to Achieve it ..?



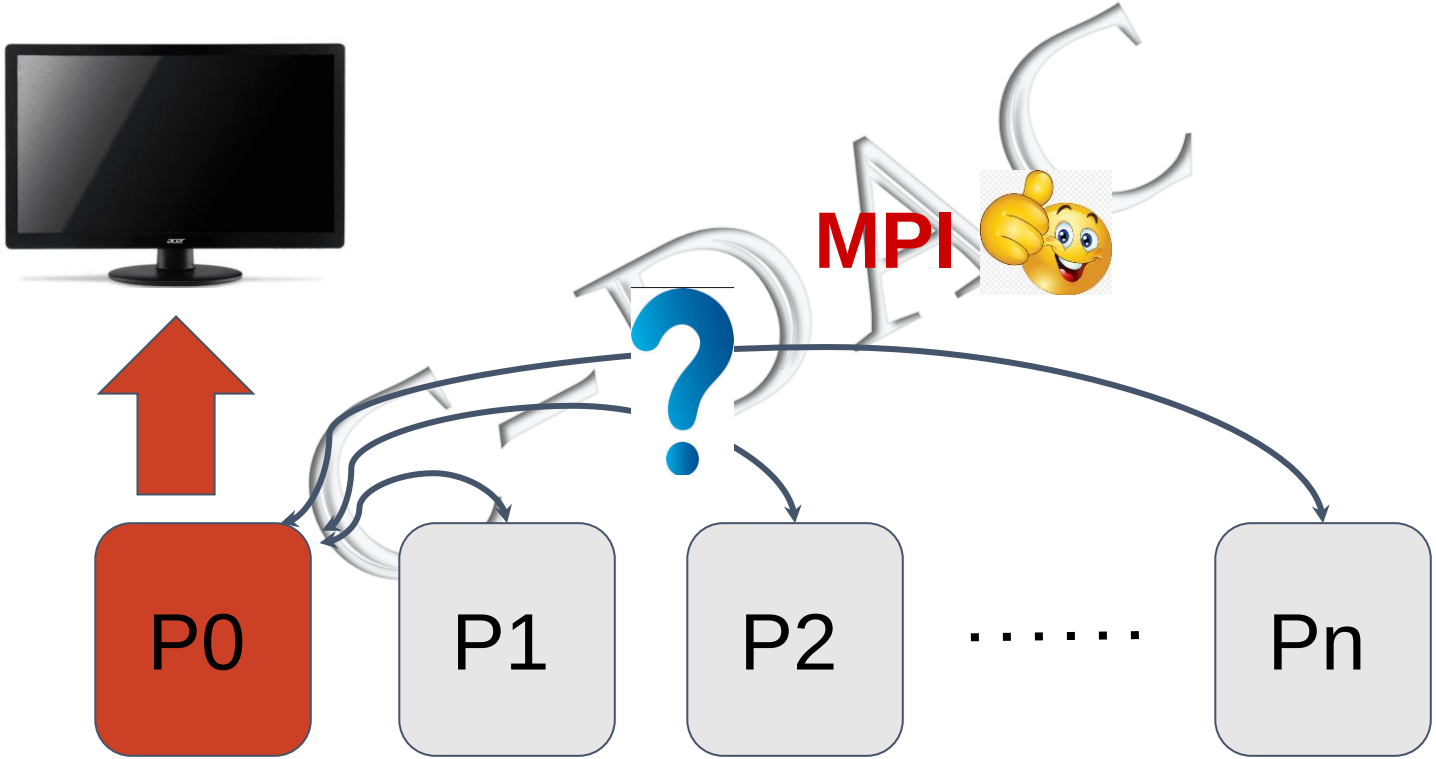
How to Achieve it ..?



How ..?

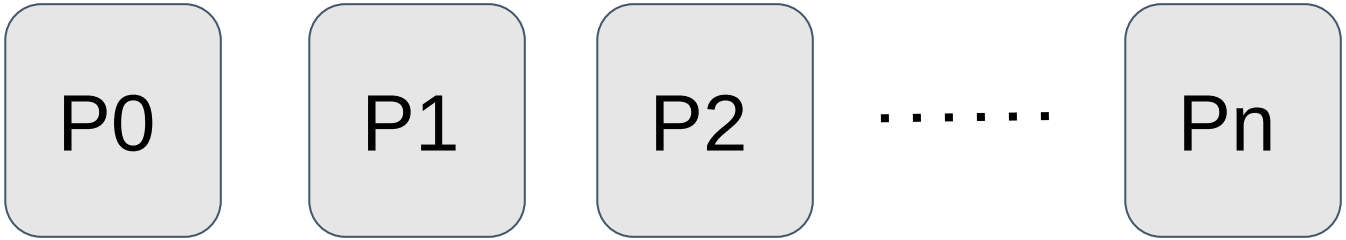


How ..?



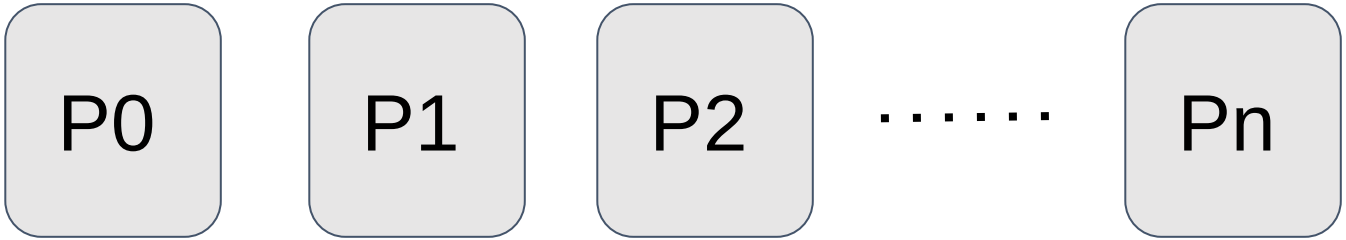
How MPI Works ..?

How MPI Works ..?



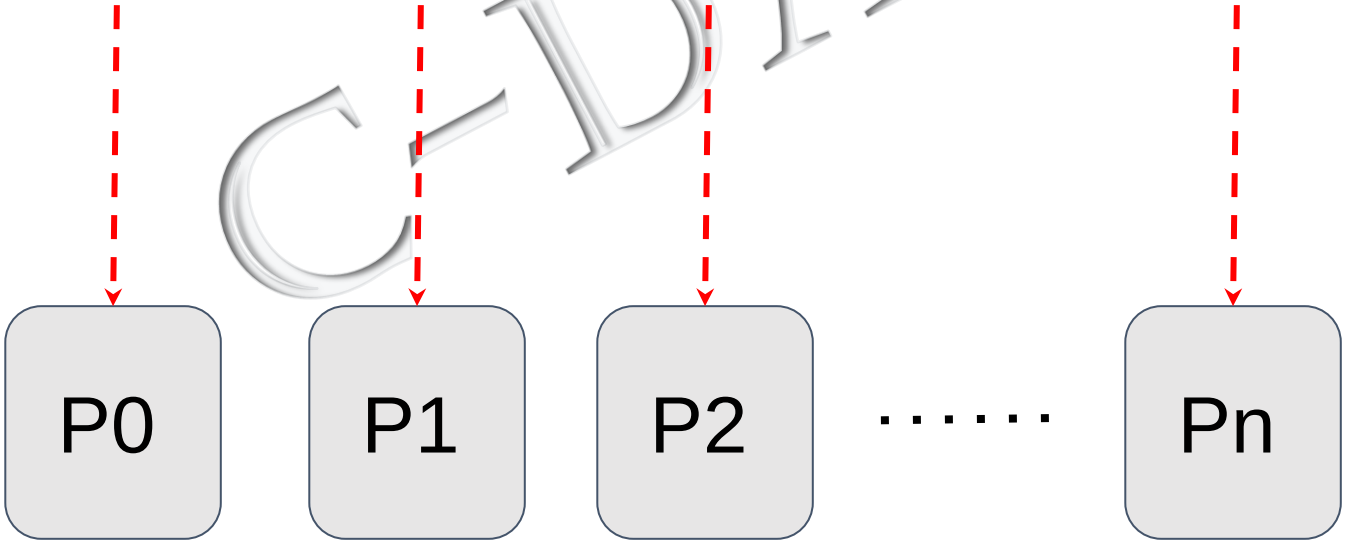
How MPI Works ..?

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



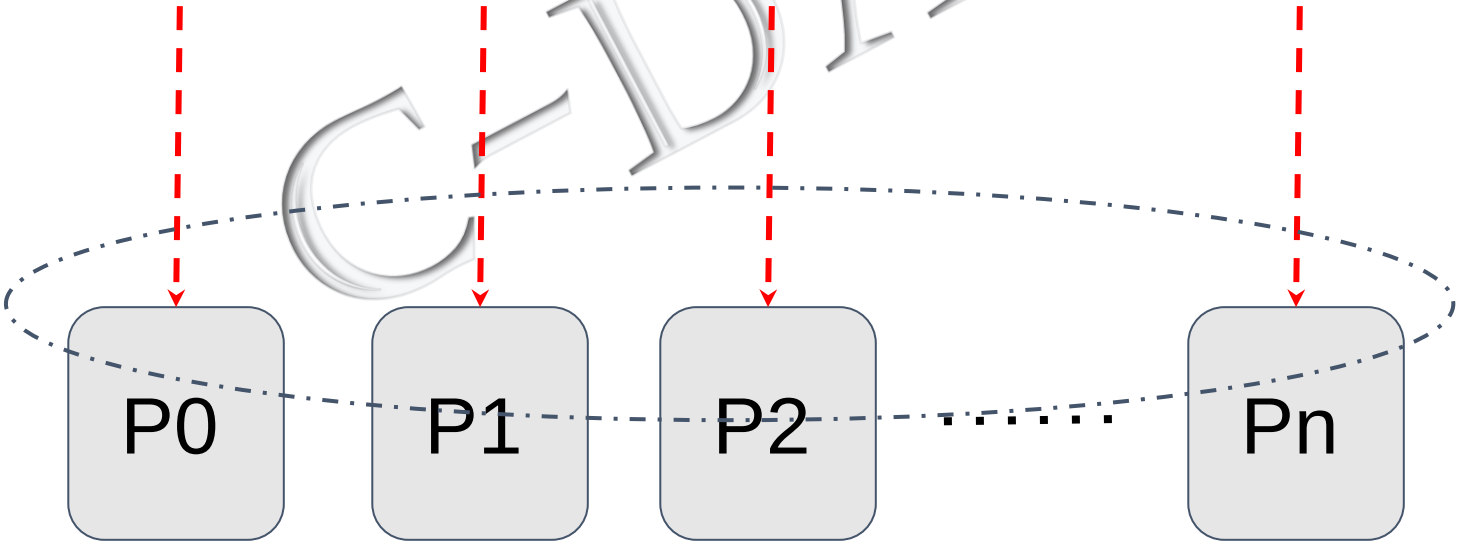
How MPI Works ..?

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



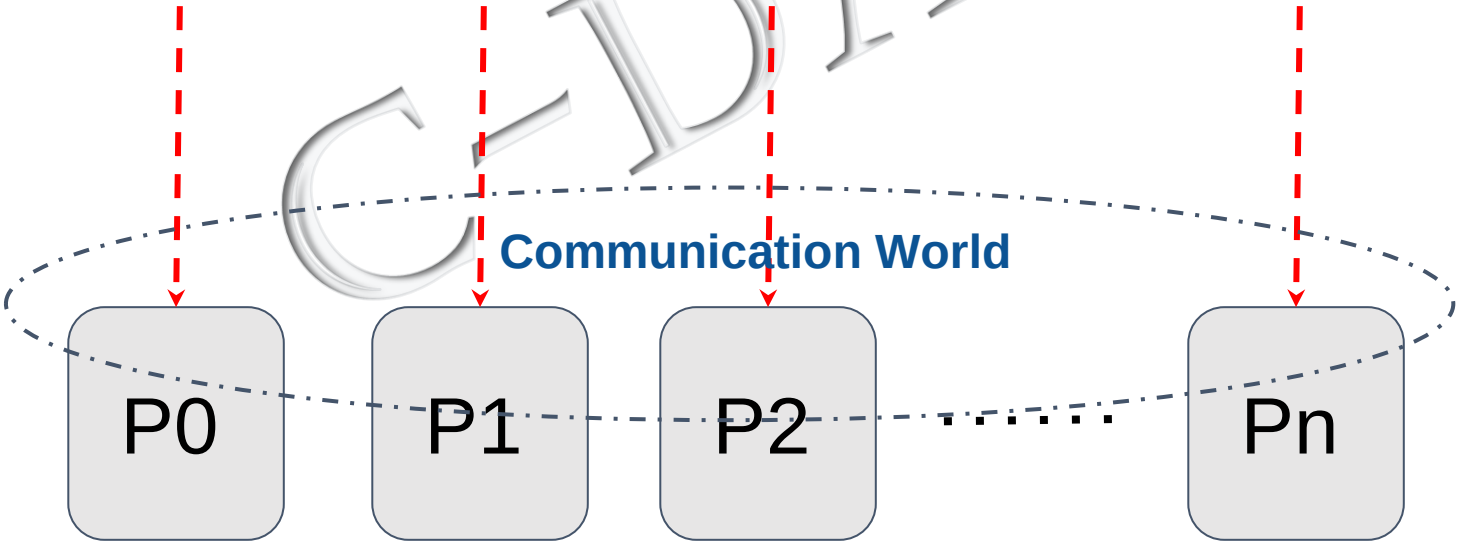
How MPI Works ..?

- **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 ?

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 () ;

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

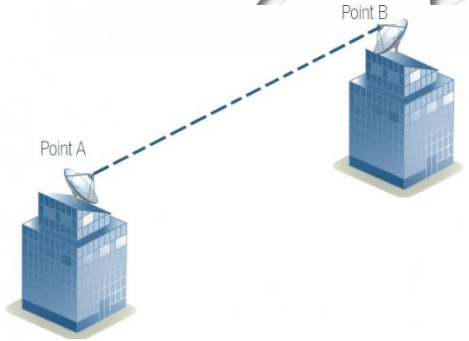
MPI - Communication

MPI - Communication

Point to Point Comm^{n}

MPI - Communication

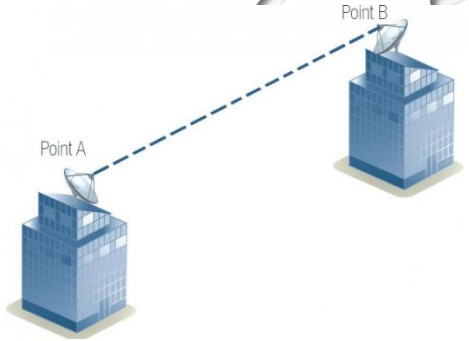
Point to Point Comm^{n}



MPI - Communication

Point to Point Comm^{n}

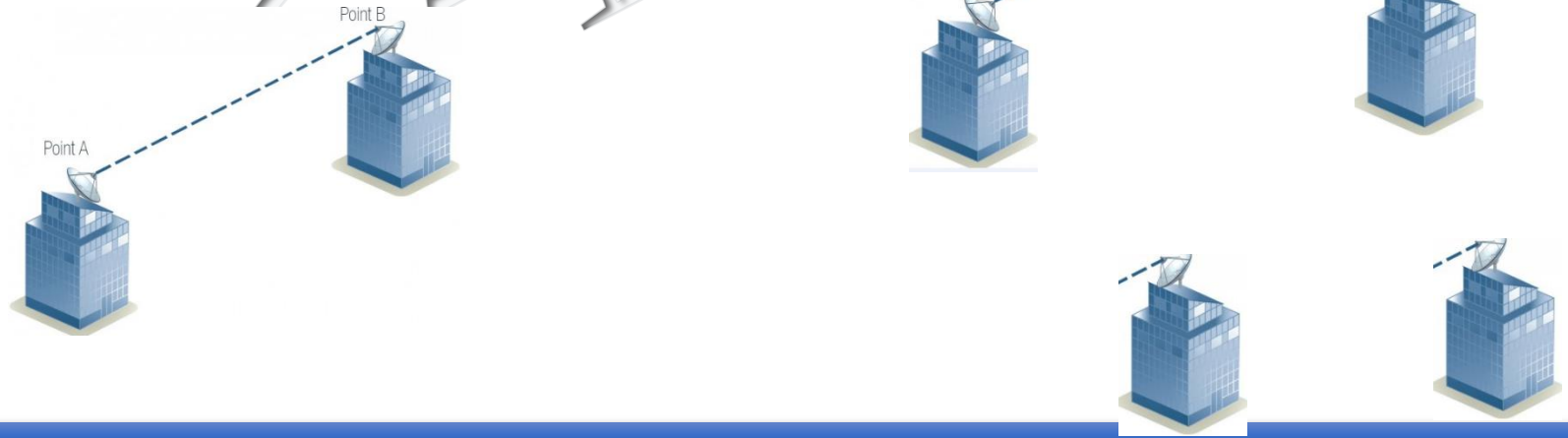
Collective Comm^{n}



MPI - Communication

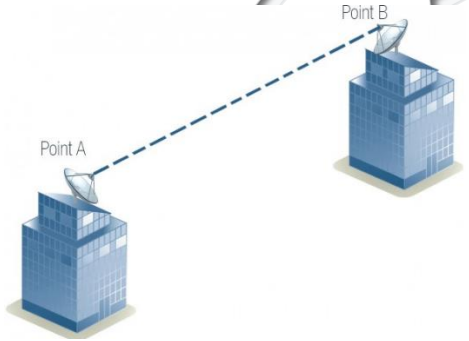
Point to Point Comm^{n}

Collective Comm^{n}

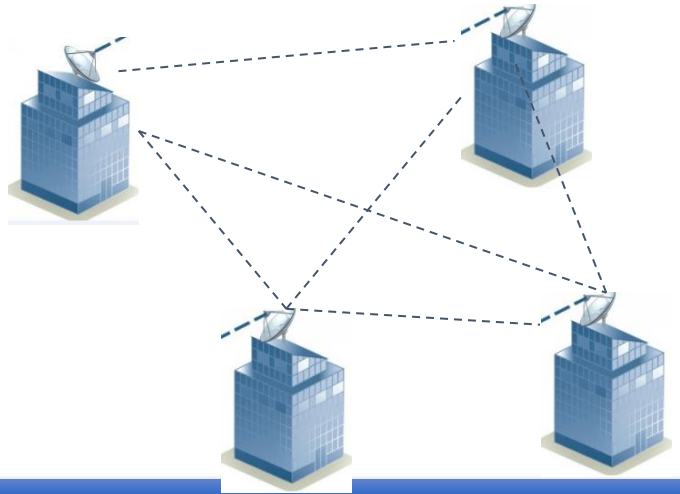


MPI - Communication

Point to Point Comm^{n}

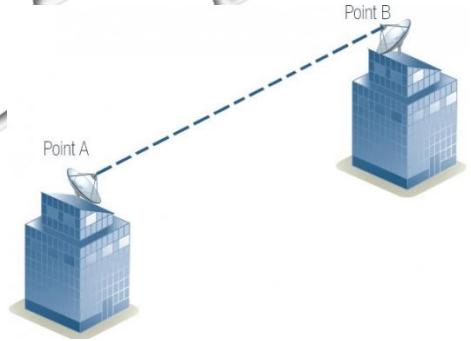


Collective Comm^{n}



MPI - Communication

Point to Point Commⁿ



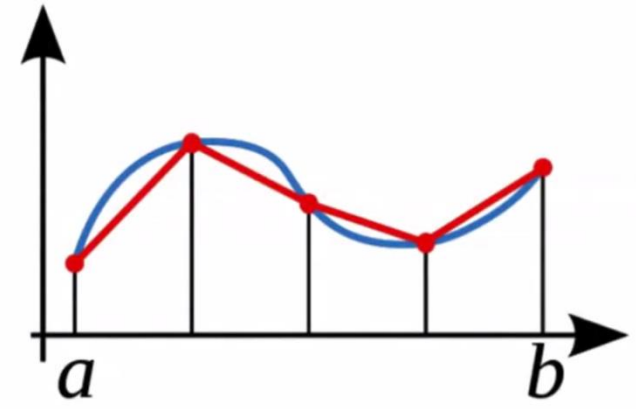
It's Always Better to understand anything with example.....

It’s Always Better to understand anything with example…..

…Do You Agree ?

Numerical Integration : Trapezoidal Rule

Numerical Integration : Trapezoidal Rule



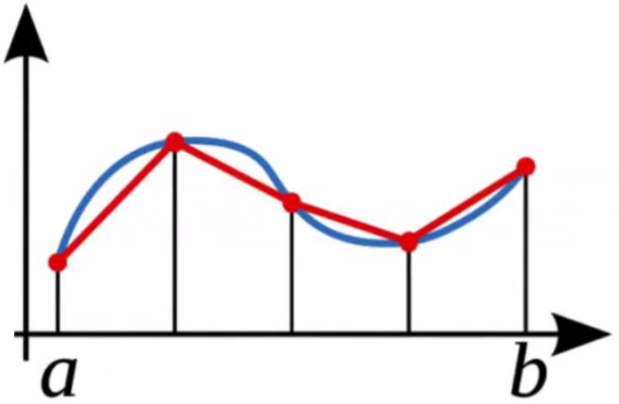
Numerical Integration : Trapezoidal Rule

Trapezoid rule for integrating $\int_a^b = f(x)dx$

with $h = (b-a)/n$ is

$$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$$

where $x_i = a + ih, i = 0, 1, ..., n$



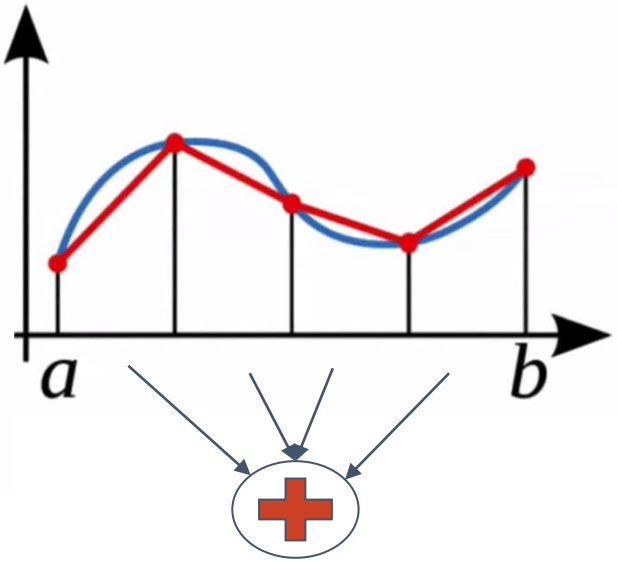
Numerical Integration : Trapezoidal Rule

Trapezoid rule for integrating $\int_a^b = f(x)dx$

with $h = (b-a)/n$ is

$$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$$

where $x_i = a + ih, i = 0, 1, ..., n$



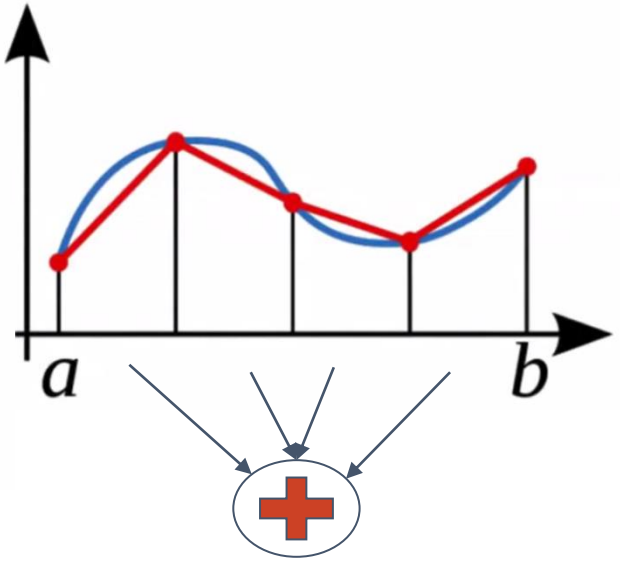
Numerical Integration : Trapezoidal Rule

Trapezoid rule for integrating $\int_a^b = f(x)dx$

with $h = (b-a)/n$ is

$$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$$

where $x_i = a + ih, i = 0, 1, ..., n$



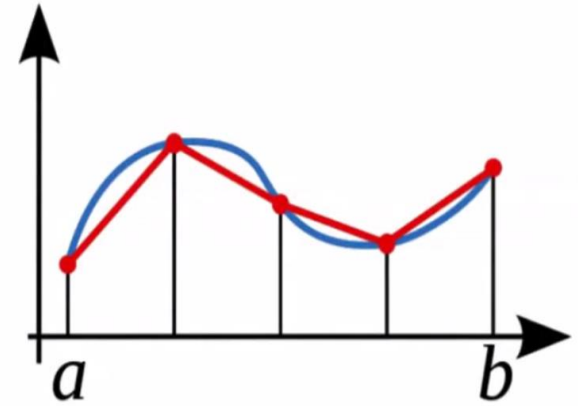
How do you achieve it Serially .. ?

```
/* traprule_serial.c */

float f(float x) ;
```

Function which we want to integrate

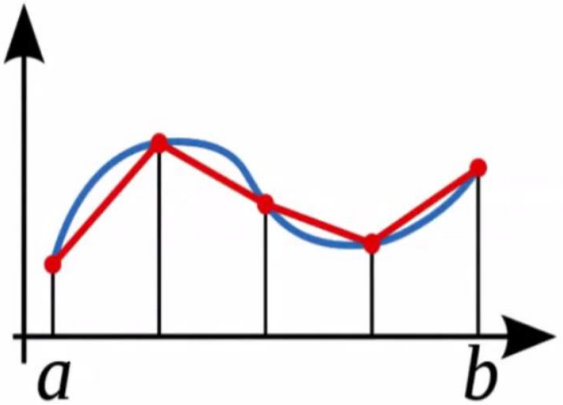
```
/* traprule_serial.c */

float f(float x) ;
```

Function which we want to integrate



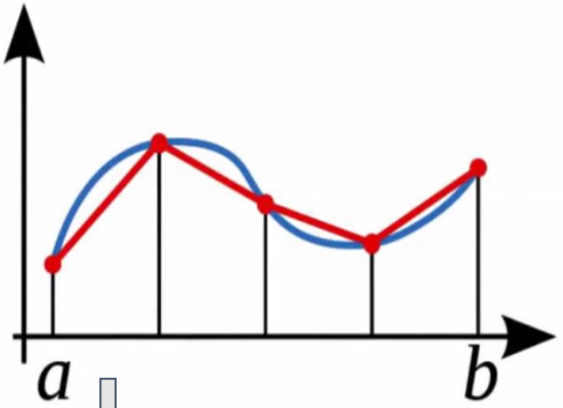
```
/* traprule_serial.c */

float f(float x) ;

float Trap(float a, float b, int n, float h)
        {
                float integral, x ;
                int i ;
                integral = (f(a) + f(b)) / 2.0 ;
                x = a ;
                for (i=1; i<= n-1; i++)
                        {
                                x = x + h;
                                integral = integral + f(x)
;
                        }
```

Function which we want to integrate



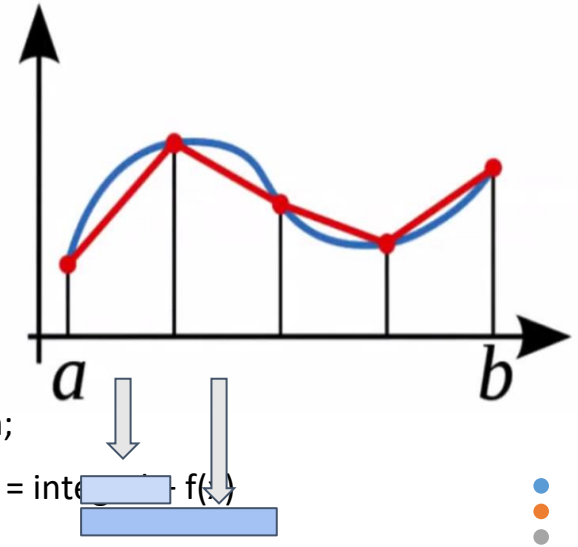
```
/* traprule_serial.c */

float f(float x) ;

float Trap(float a, float b, int n, float h)
        {
                float integral, x ;
                int i ;
                integral = (f(a) + f(b)) / 2.0 ;
                x = a ;
                for (i=1; i<= n-1; i++)
                        {
                x = x + h;
                integral = inte[redacted] f(x)
                        }
;
```

Function which we want to integrate



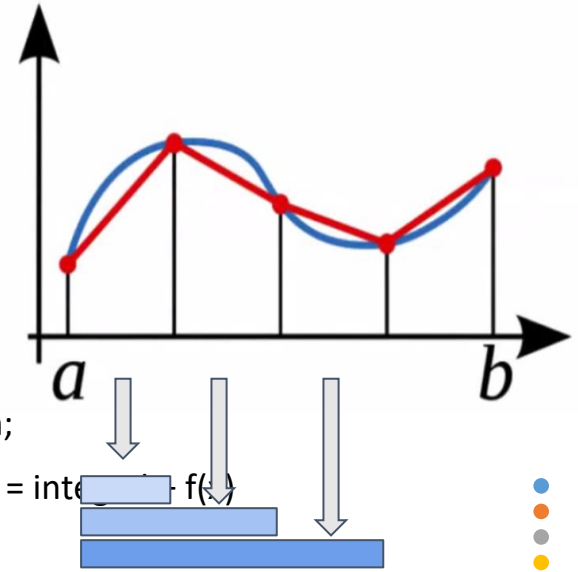
```
/* traprule_serial.c */

float f(float x) ;

float Trap(float a, float b, int n, float h)
        {
                float integral, x ;
                int i ;
                integral = (f(a) + f(b)) / 2.0 ;
                x = a ;
                for (i=1; i<= n-1; i++)
                        {
                x = x + h;
                integral = inte      f(x)
                        }
;
```

Function which we want to integrate



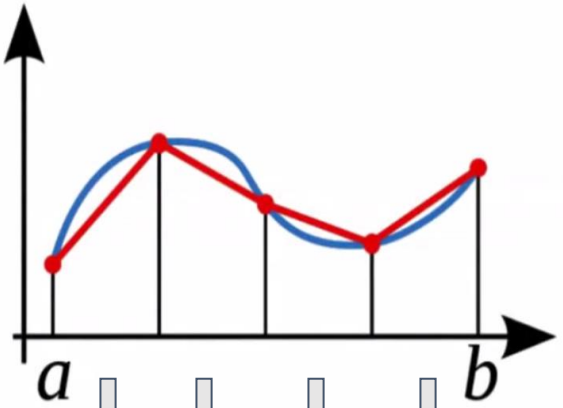
```
/* traprule_serial.c */

float f(float x) ;

float Trap(float a, float b, int n, float h)
        {
                float integral, x ;
                int i ;
                integral = (f(a) + f(b)) / 2.0 ;
                x = a ;
                for (i=1; i<= n-1; i++)
                        {
                x = x + h;
                integral = inte      f(x)
;
                        }
```

Function which we want to integrate



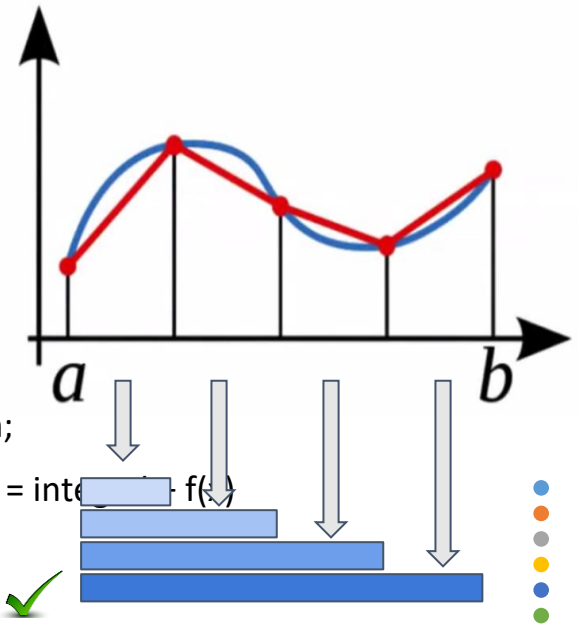
```
/* traprule_serial.c */

float f(float x) ;

float Trap(float a, float b, int n, float h)
        {
                float integral, x ;
                int i ;
                integral = (f(a) + f(b)) / 2.0 ;
                x = a ;
                for (i=1; i<= n-1; i++)
                        {
                x = x + h;
                integral = inte      f(x)
;
                        }
```

Function which we want to integrate



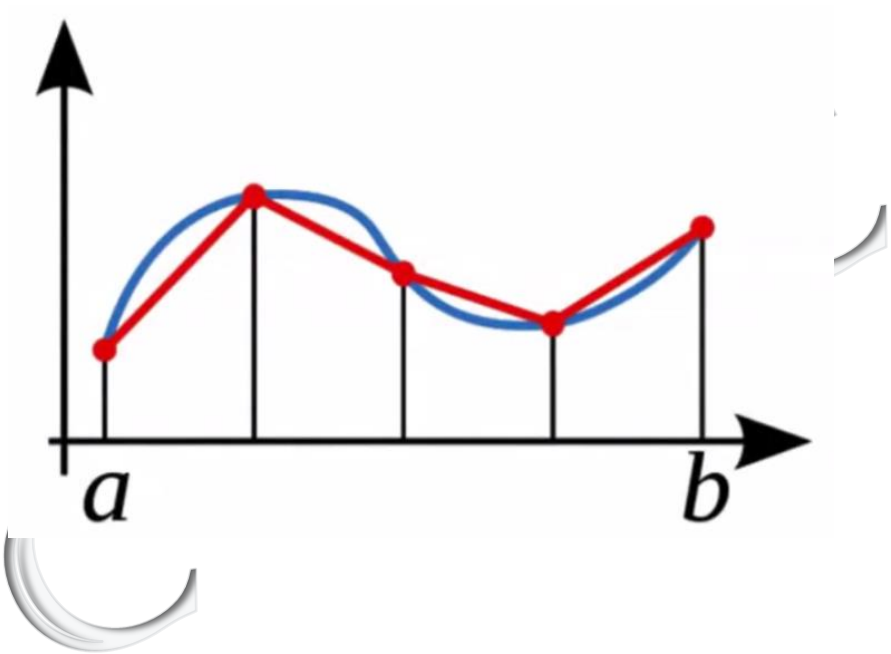
```
/* traprule_serial.c */

float f(float x) ;

float Trap(float a, float b, int n, float h)
        {
                float integral, x ;
                int i ;
                integral = (f(a) + f(b)) / 2.0 ;
                x = a ;
                for (i=1; i<= n-1; i++)
                        {
                x = x + h;
                integral = inte      f(x)
                        }
;
```

Function which we want to integrate

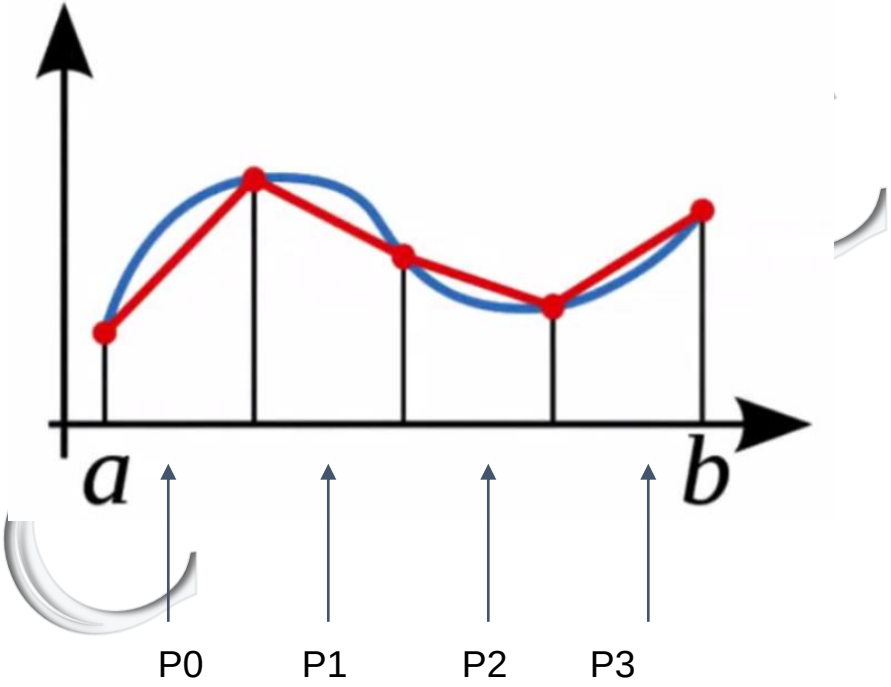


How we can do it Parallely .. ?

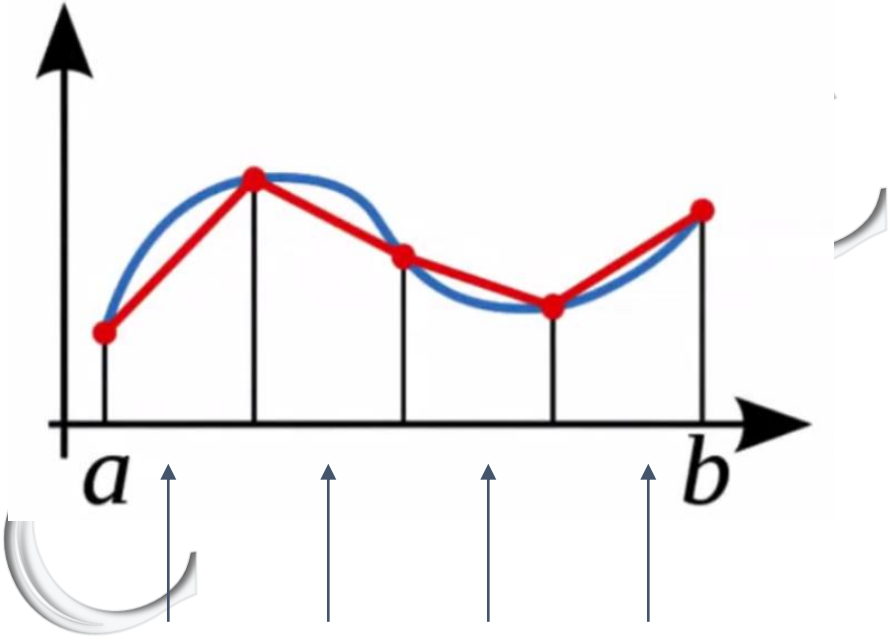
Trap Rule : Parallel Approach



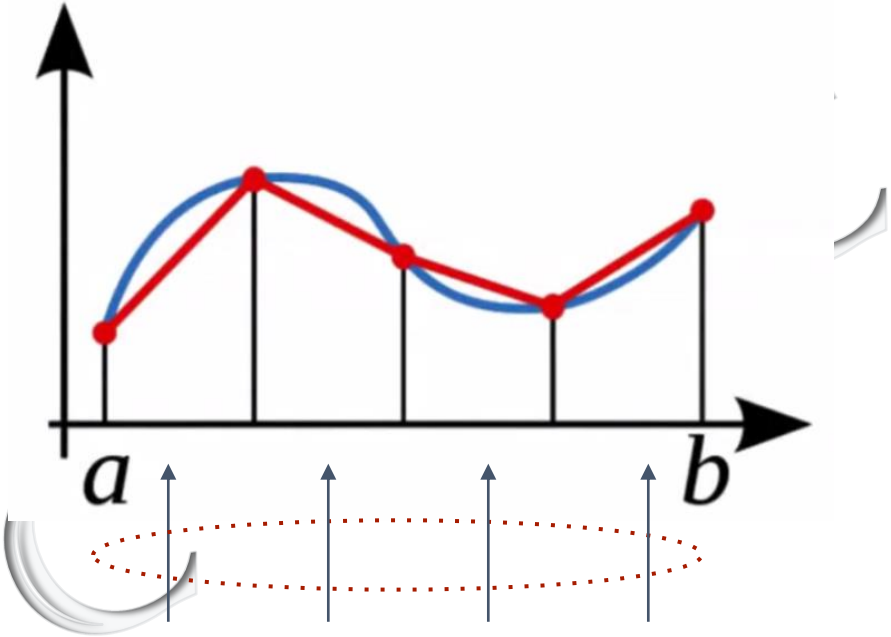
Trap Rule : Parallel Approach



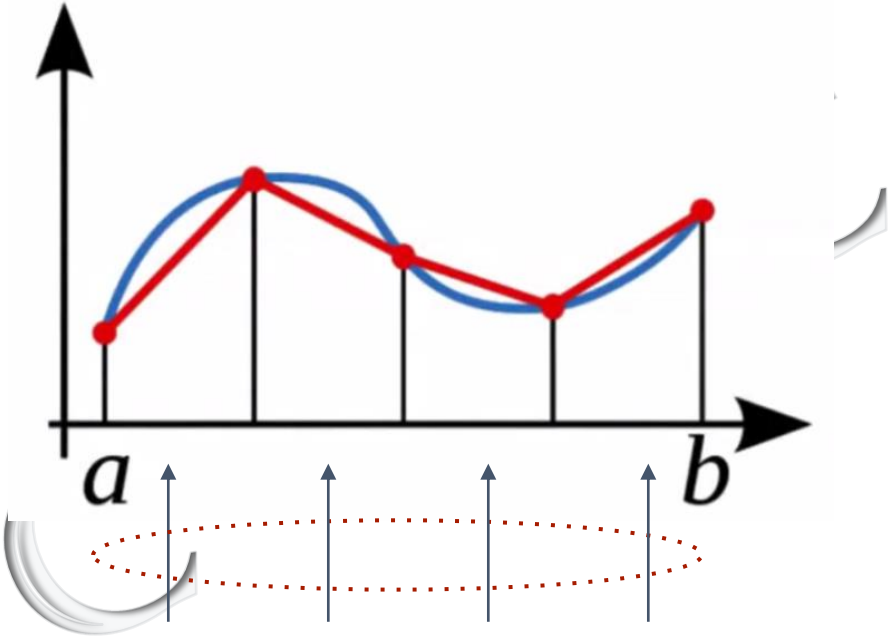
Trap Rule : Parallel Approach

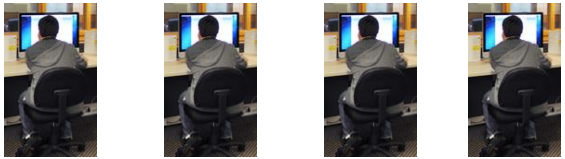


Trap Rule : Parallel Approach

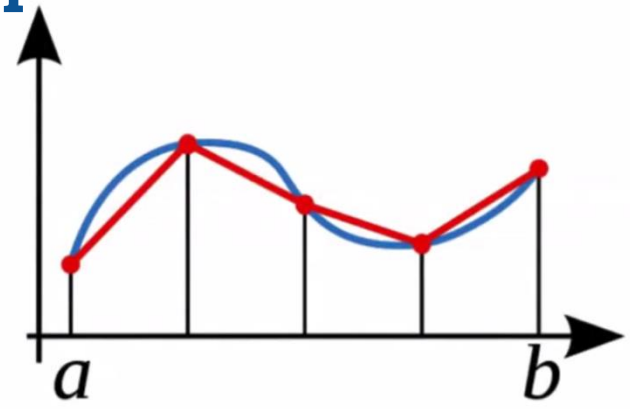


Trap Rule : Parallel Approach





Trap Rule : Parallel Approach



Trapezoid rule for integrating $\int_a^b = f(x)dx$

with $h = (b-a)/n$ is

$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$

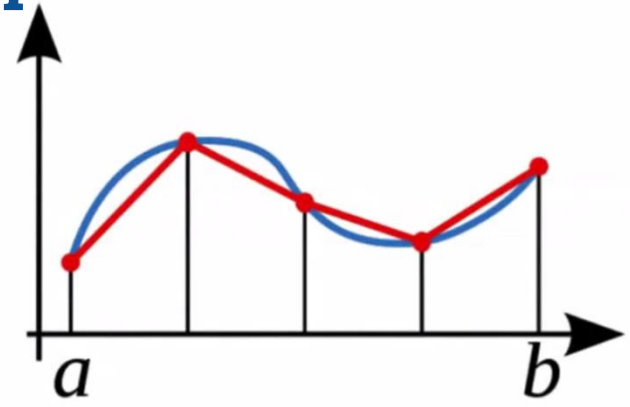
where $x_i = a + ih, i = 0, 1, ..., n$

Given p processes, each process can work on n/p intervals

Note: for simplicity will assume n/p is an integer

process	interval
0	$[a, a + \frac{n}{p}h]$
1	$[a + \frac{n}{p}h, a + 2\frac{n}{p}h]$
...	...
p-1	$[a + (p-1)\frac{n}{p}h, b]$

Trap Rule : Parallel Approach



Trapezoid rule for integrating $\int_a^b = f(x)dx$

with $h = (b-a)/n$ is

$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$

where $x_i = a + ih, i = 0, 1, ..., n$

Given p processes, each process can work on n/p intervals

Note: for simplicity will assume n/p is an integer

process	interval
0	$[a, a + \frac{n}{p}h]$
1	$[a + \frac{n}{p}h, a + 2\frac{n}{p}h]$
...	...
p-1	$[a + (p-1)\frac{n}{p}h, b]$

Trap Rule : Parallel Approach

Trapezoid rule for integrating $\int_a^b = f(x)dx$

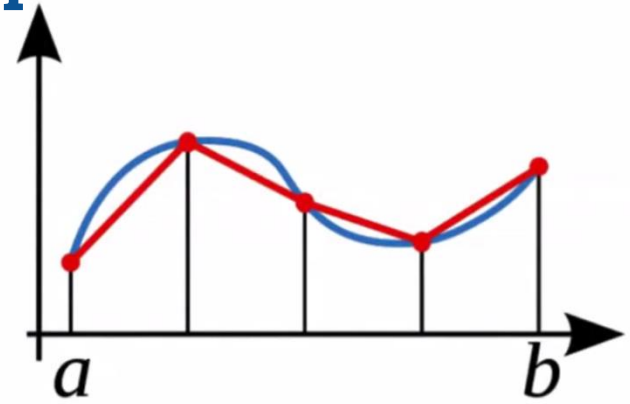
with $h = (b-a)/n$ is

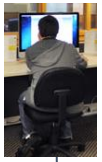
$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$

where $x_i = a + ih, i = 0, 1, ..., n$

Given p processes, each process can work on n/p intervals

Note: for simplicity will assume n/p is an integer





process	interval
0	$[a, a + \frac{n}{p}h]$
1	$[a + \frac{n}{p}h, a + 2\frac{n}{p}h]$
...	...
p-1	$[a + (p-1)\frac{n}{p}h, b]$

Trap Rule : Parallel Approach

Trapezoid rule for integrating $\int_a^b = f(x)dx$

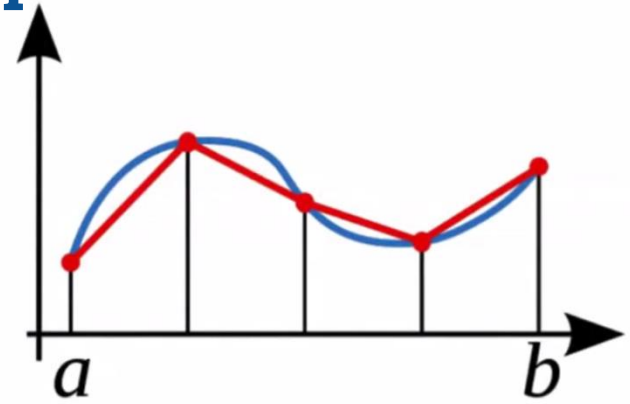
with $h = (b-a)/n$ is

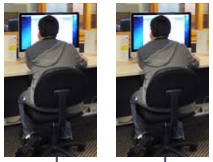
$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$

where $x_i = a + ih, i = 0, 1, ..., n$

Given p processes, each process can work on n/p intervals

Note: for simplicity will assume n/p is an integer





process	interval
0	$[a, a + \frac{n}{p}h]$
1	$[a + \frac{n}{p}h, a + 2\frac{n}{p}h]$
...	...
p-1	$[a + (p-1)\frac{n}{p}h, b]$

Trap Rule : Parallel Approach

Trapezoid rule for integrating $\int_a^b = f(x)dx$

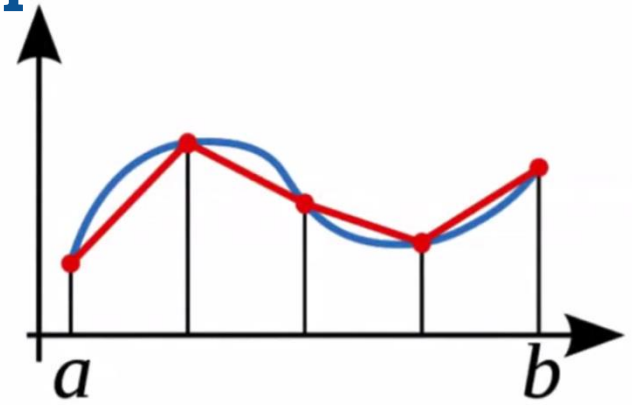
with $h = (b-a)/n$ is

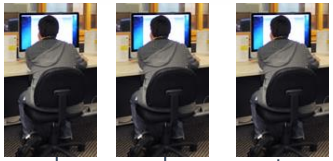
$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$

where $x_i = a + ih, i = 0, 1, ..., n$

Given p processes, each process can work on n/p intervals

Note: for simplicity will assume n/p is an integer





process	interval
0	$[a, a + \frac{n}{p}h]$
1	$[a + \frac{n}{p}h, a + 2\frac{n}{p}h]$
...	...
p-1	$[a + (p-1)\frac{n}{p}h, b]$

Trap Rule : Parallel Approach

Trapezoid rule for integrating $\int_a^b = f(x)dx$

with $h = (b-a)/n$ is

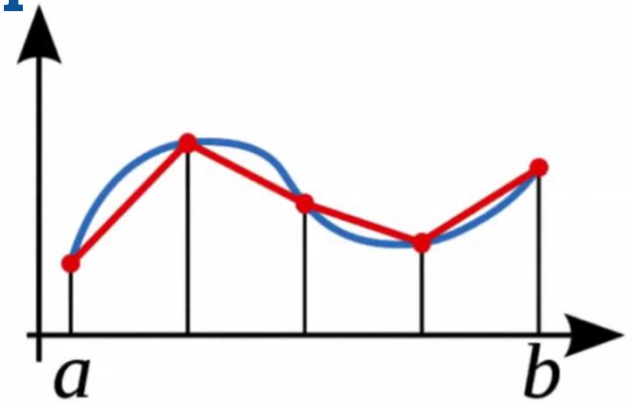
$f(x) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i)$

where $x_i = a + ih, i = 0, 1, ..., n$

Given p processes, each process can work on n/p intervals

Note: for simplicity will assume n/p is an integer

process	interval
0	$[a, a + \frac{n}{p}h]$
1	$[a + \frac{n}{p}h, a + 2\frac{n}{p}h]$
...	...
p-1	$[a + (p-1)\frac{n}{p}h, b]$



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

void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);

double Trap(double local_a, double local_b, int local_n, double h);
                                        /* Calculate local area  */

double f(double x);                     /* function we're integrating */
```

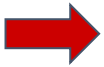
```
#include <stdio.h>
#include <mpi.h>

void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);

double Trap(double local_a, double local_b, int local_n, double h);
                                        /* Calculate local area  */

double f(double x);                     /* function we're integrating */
```

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

void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);

double Trap(double local_a, double local_b, int local_n, double h);
                                        /* Calculate local area  */

double f(double x);                     /* function we're integrating */
```

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

void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);

double Trap(double local_a, double local_b, int local_n, double h);
                                        /* Calculate local area  */

double f(double x);                     /* function we're integrating */
```

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

void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);

double Trap(double local_a, double local_b, int local_n, double h);
                                        /* Calculate local area  */

double f(double x);                     /* function we're integrating */
```

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

void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);

double Trap(double local_a, double local_b, int local_n, double h);
                                        /* Calculate local area  */

double f(double x);                     /* function we're integrating */
```



```
int main(int argc, char** argv)
{
        int         my_rank;            /* My process rank         */
        int         p;                  /* The
number of processes   */
        double      a;                  /* Left
endpoint        */
        double      b;                  /* Right
endpoint        */
        int         n;
  /* Number of trapezoids                */
        double      h;                  /* Trapezoid
base length     */
        double      local_a;                /* Left endpoint my
process  */
        double      local_b;            /* Right endpoint my process
*/
        int         local_n;            /* Number
of trapezoids for  */
        double      my_area;                /* Integral over my
```

```
 MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Environment Management Routines

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

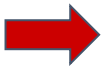
```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

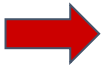
```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

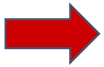
```
 MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```



```
 MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```



```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```



```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Example : Let - n=100, a=0, b=100, p=4,

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Example : Let - n=100, a=0, b=100, p=4,

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

P0

Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

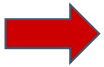
Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

P0

- **local_a = 0 + 0*25*1 = 0**
- **local_b = 0 + 25*1 = 25**

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```



P1

Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

P1

- **local_a = 0 + 1*25*1 = 25**
- **local_b = 25 + 25*1 = 50**

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

P2

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

P2

- **local_a = 0 + 2*25*1 = 50**
- **local_b = 50 + 25*1 = 75**

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

P3

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;
local_n = n/p;

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```

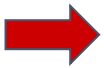
Example : Let - n=100, p=4, a=0, b=100.

- **h**= (100-0)/100 = 1
- **local_n**=100/4 = 25

P3

- **local_a = 0 + 3*25*1 = 75**
- **local_b = 75 + 25*1 = 100**

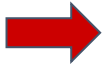
```
 MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

Get_data(p, my_rank, &a, &b, &n);

h = (b-a)/n;                    /* h is the same for all processes */
local_n = n/p;                  /* So is the number of trapezoids */

local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
my_area = Trap(local_a, local_b, local_n, h);
```



```
 if (my_rank == 0)
{
            total = my_area;
            for (source = 1; source < p; source++)
     {
            MPI_Recv(&my_area, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
            total = total + my_area;
            }
} else
      {
                        MPI_Send(&my_area, 1, MPI_DOUBLE, dest,  tag, MPI_COMM_WORLD);
       }
```

```
if (my_rank == 0)
{
        total = my_area;
        for (source = 1; source < p; source++)
    {
        MPI_Recv(&my_area, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
        total = total + my_area;
        }
} else
    {
            MPI_Send(&my_area, 1, MPI_DOUBLE, dest,  tag, MPI_COMM_WORLD);
     }
```

```
 if (my_rank == 0)
{
        total = my_area;
        for (source = 1; source < p; source++)
    {
        MPI_Recv(&my_area, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
        total = total + my_area;
        }
} else
    {
                MPI_Send(&my_area, 1, MPI_DOUBLE, dest,  tag, MPI_COMM_WORLD);
     }
```



```
 if (my_rank == 0)
{
        total = my_area;
        for (source = 1; source < p; source++)
    {
        MPI_Recv(&my_area, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
        total = total + my_area;
        }
} else
    {
            MPI_Send(&my_area, 1, MPI_DOUBLE, dest,  tag, MPI_COMM_WORLD);
     }
```

```
    if (my_rank == 0)
    {
        printf("With n = %d trapezoids, our estimate\n", n);
        printf("of the area from %f to %f = %.15f\n", a, b, total);
    }

    MPI_Finalize();

    return 0;
}

    /* END of MAIN */
```

```
    if (my_rank == 0)
    {
        printf("With n = %d trapezoids, our estimate\n", n);
        printf("of the area from %f to %f = %.15f\n", a, b, total);
     }

     MPI_Finalize();

    return 0;
}

    /* END of MAIN */
```

```
        if (my_rank == 0)
        {
            printf("With n = %d trapezoids, our estimate\n", n);
            printf("of the area from %f to %f = %.15f\n", a, b, total);
        }

        MPI_Finalize();

        return 0;
}

        /* END of MAIN */
```

Releases the MPI resources

```
void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p)
{
        int             q;
        MPI_Status status;
        if (my_rank == 0)
      {
            printf("Enter a, b, and n\n");
            scanf("%lf %lf %d", a_p, b_p, n_p);
                        for (q = 1; q < p; q++) {
                        MPI_Send(a_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(b_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(n_p, 1, MPI_INT, q, 0, MPI_COMM_WORLD);
                        }
        } else
      {
            MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        }
}
```

```
void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p)
{
        int             q;
        MPI_Status status;
        if (my_rank == 0)
      {
            printf("Enter a, b, and n\n");
            scanf("%lf %lf %d", a_p, b_p, n_p);
                        for (q = 1; q < p; q++) {
                        MPI_Send(a_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(b_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(n_p, 1, MPI_INT, q, 0, MPI_COMM_WORLD);
                        }
        } else
      {
            MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        }
}
```

```
void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p)
{
        int             q;
        MPI_Status status;
        if (my_rank == 0)
      {
          printf("Enter a, b, and n\n");
          scanf("%lf %lf %d", a_p, b_p, n_p);
                        for (q = 1; q < p; q++) {
                        MPI_Send(a_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(b_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(n_p, 1, MPI_INT, q, 0, MPI_COMM_WORLD);
                        }
      } else
      {
          MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
          MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
          MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
      }
```

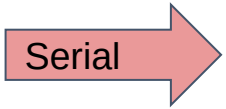
```
void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p)
{
        int             q;
        MPI_Status status;
        if (my_rank == 0)
       {
            printf("Enter a, b, and n\n");
            scanf("%lf %lf %d", a_p, b_p, n_p);
                        for (q = 1; q < p; q++) {
                        MPI_Send(a_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(b_p, 1, MPI_DOUBLE, q, 0, MPI_COMM_WORLD);
                        MPI_Send(n_p, 1, MPI_INT, q, 0, MPI_COMM_WORLD);
                        }
        } else
       {
            MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        }
}
```

```
double Trap(double  local_a, double  local_b , int         local_n , double  h )
        {
            double my_area;                  /* Store my result in my_area  */
            double x;
            int i;
            my_area = (f(local_a) + f(local_b))/2.0;
            x = local_a;
            for (i = 1; i <= local_n-1; i++)
            {
                        x = local_a + i*h;
                        my_area = my_area + f(x);
            }
            my_area = my_area*h;
            return my_area;
        }
```

```
double f(double x)
 {
        double return_val;

        return_val = x*x + 1.0;

        return return_val;
}

    /* END of Program */
```

❖ How to compile and run it ?

❖ How to compile and run it ?



❖ How to compile and run it ?

Serial

```
➢ $  gcc -o trap trap_serial.c
```

❖ How to compile and run it ?

Serial

- $ gcc -o trap trap_serial.c
- $ ./trap

❖ How to compile and run it ?

Serial

```
➢ $  gcc -o trap trap_serial.c
➢ $ ./trap
```

Parallel

❖ How to compile and run it ?

Serial

- $ gcc -o trap trap_serial.c
- $ ./trap

Parallel

- $ mpicc -o trap_mpi trap_mpi.c

❖ How to compile and run it ?

Serial

- $ gcc -o trap trap_serial.c
- $ ./trap

Parallel

- $ **mpicc** -o trap_mpi trap_mpi.c

❖ How to compile and run it ?

Serial

- $ gcc -o trap trap_serial.c
- $ ./trap

Parallel

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

❖ How to compile and run it ?

Serial

- $ gcc -o trap trap_serial.c
- $ ./trap

Parallel

- $ mpicc -o trap_mpi trap_mpi.c
- $ mpirun **-np n** ./trap_mpi

❖ Output ..

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

❖ Output ..

- $ mpicc -o trap_mpi trap_mpi.c
- $ mpirun **-np 4** ./trap_mpi

```
[om@shrestha1 Trap]$ mpicc -o trap_mpi trap_mpi.c
[om@shrestha1 Trap]$ ls
trap_mpi  trap_mpi_backup.c  trap_mpi.c
[om@shrestha1 Trap]$ mpirun -np 4 ./trap_mpi
With n = 100000000 trapezoids, our estimate
of the area from 1.000000 to 1000000000000.000000 = 333333333333325575245376068381573120.000000000000000
[om@shrestha1 Trap]$
```

Recap :

Recap :

- Serial and Parallel Approach -

Recap :

- Serial and Parallel Approach -
- MPI_Comm_rank(...)

Recap :

- Serial and Parallel Approach -
- MPI_Comm_rank(...)
- MPI_Comm_size(...)

Recap :

- Serial and Parallel Approach -
- MPI_Comm_rank(...)
- MPI_Comm_size(...)
- Point to point communication

Recap :

- Serial and Parallel Approach -
- MPI_Comm_rank(...)
- MPI_Comm_size(...)
- Point to point communication
- MPI_Send(...)

Recap :

- Serial and Parallel Approach -
- MPI_Comm_rank(...)
- MPI_Comm_size(...)
- Point to point communication
- MPI_Send(...)
- MPI_Recv(...)

Recap :

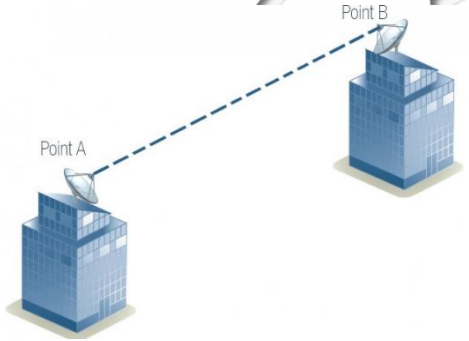
- Serial and Parallel Approach -
- MPI_Comm_rank(...)
- MPI_Comm_size(...)
- Point to point communication
- MPI_Send(...)
- MPI_Recv(...)
- Blocking and Non-blocking Point to Point communication - Cases !

Recap :

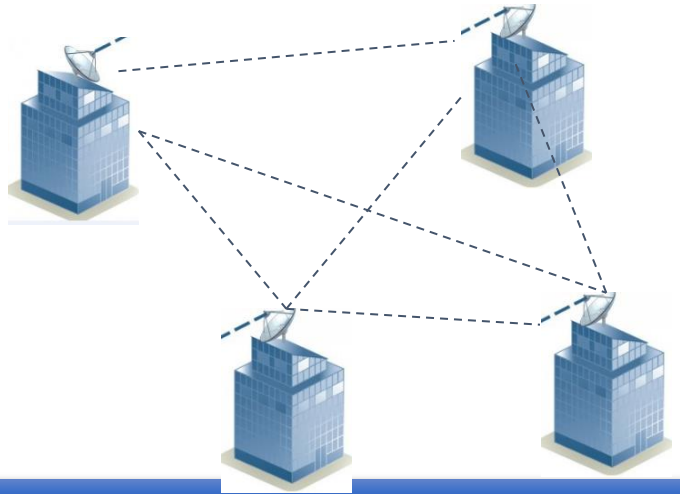
- Serial and Parallel Approach -
- MPI_Comm_rank(...)
- MPI_Comm_size(...)
- Point to point communication
- MPI_Send(...)
- MPI_Recv(...)
- Blocking and Non-blocking Point to Point communication - Cases !
- ..... Trapezoidal Rule Example

MPI - Communication

Point to Point Comm^{n}



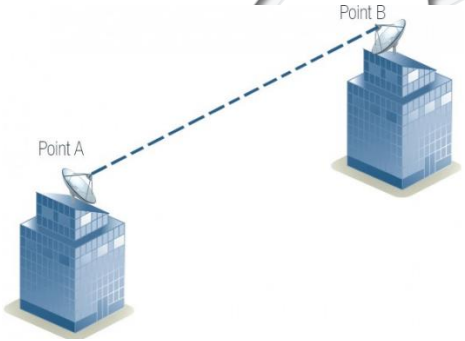
Collective Comm^{n}

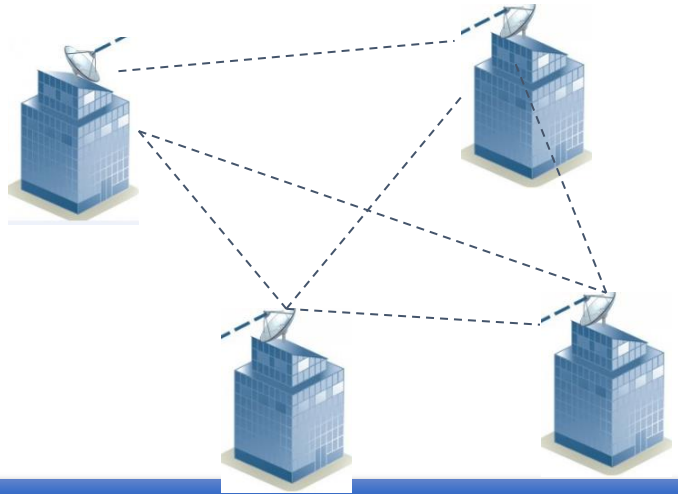


MPI - Communication

Point to Point Comm^{n}

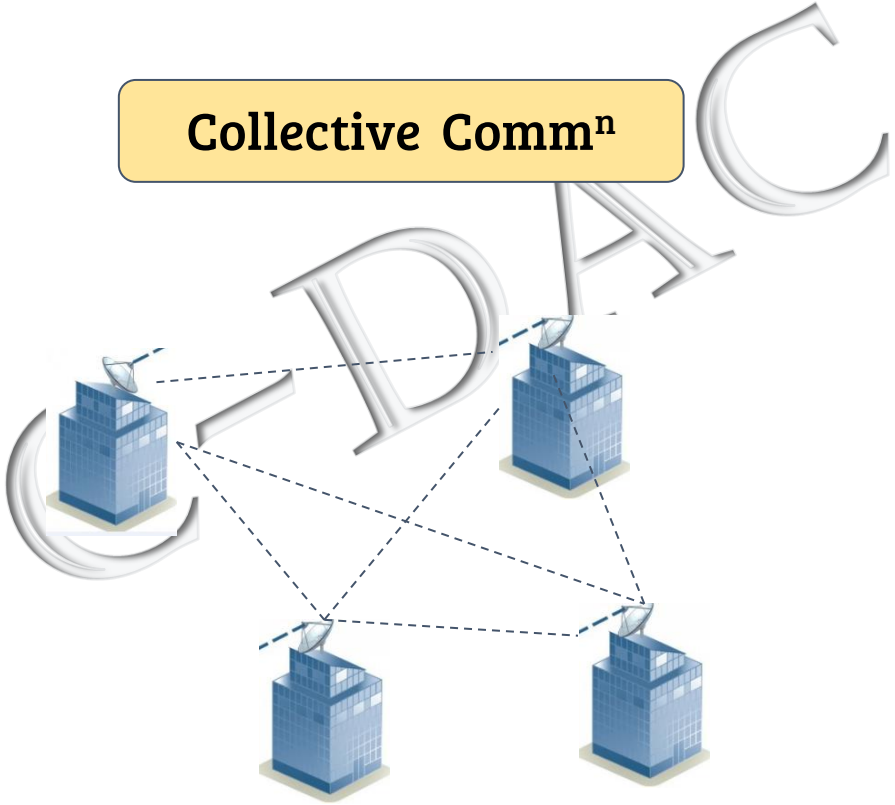
Collective Comm^{n}





MPI - Communication

Collective Comm^{n}



MPI - Collective Communication

- Collective communication must involve all processes in the scope of a communicator.
- Involve coordinated communication within a group of processes identified by an MPI communicator.

Types of Collective Operations

- Synchronization - Processes wait until all members of the group have reached the synchronization point.
- Data Movement - broadcast, scatter/gather, all to all
- Collective Computation (reductions) - one member of the group collects data from the other members and performs an operation (min,max, add, multiply, etc.) on that data.

Basic Collective Communication Routines

- MPI_Bcast( ) - Broadcast (one to all)
- MPI_Scatter( ) - Scatter (one to all)
- MPI_Gather( ) - Gather (all to one)
- MPI_Reduce( ) - Reduce (all to one)
- MPI_Allgather( ) - (all to all)
- MPI_Allreduce( ) - (all to all)

MPI - Broadcast

Syntax :

MPI_Bcast (void* data , Int count , MPI_Datatype datatype , Int source_process , MPI_Comm comm) ;

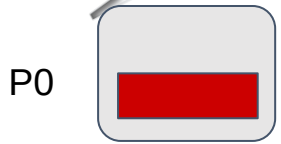
- One process sends the same data to all processes in a communicator.

MPI - Broadcast

Syntax :

MPI_Bcast (void* data , Int count , MPI_Datatype datatype , Int source_process , MPI_Comm comm) ;

- One process sends the same data to all processes in a communicator.

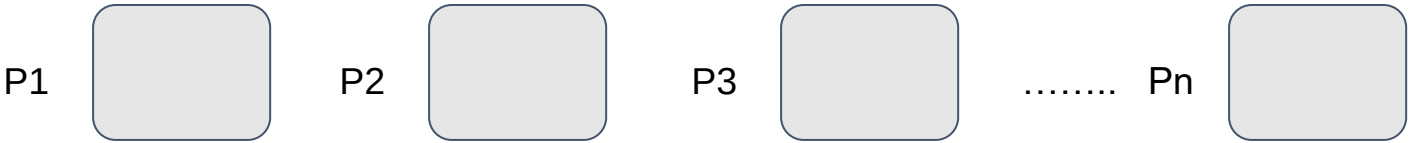


MPI - Broadcast

Syntax :

MPI_Bcast (void* data , Int count , MPI_Datatype datatype , Int source_process , MPI_Comm comm) ;

- One process sends the same data to all processes in a communicator.

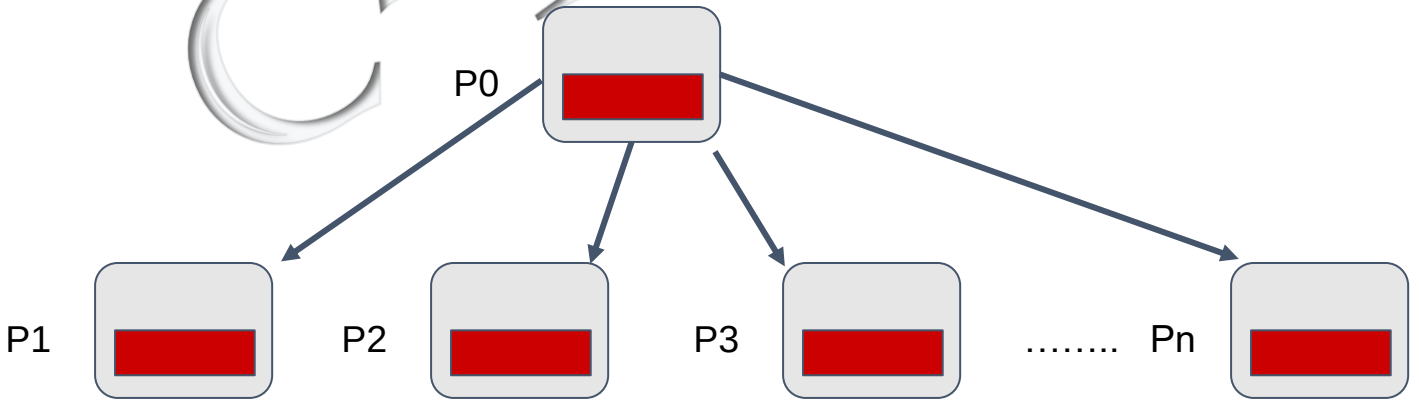


MPI - Broadcast

Syntax :

MPI_Bcast (void* data , Int count , MPI_Datatype datatype , Int source_process , MPI_Comm comm) ;

- One process sends the same data to all processes in a communicator.



MPI - Broadcast : Example

```
void Get input(int my rank ,Int comm_sz , double a_p , double  b_p , int*  n_p )
        {
                                if (my rank == 0)
                                {
                                        printf("Enter a, b, and n \n");
                                        scanf("%lf %lf %d", a_p, b_p, n_p);
                                }

    MPI Bcast(a_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

                MPI Bcast(b_p, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

                MPI Bcast(n_p, 1, MPI _INT, 0, MPI_COMM_WORLD);
          }
```

MPI - Reduce

Syntax :

MPI_Reduce (void* input_data , void* output_data , Int count ,
MPI_Datatype datatype , MPI_Op operator , Int
Dest_process, MPI_Comm comm) ;

MPI - Reduce

Syntax :

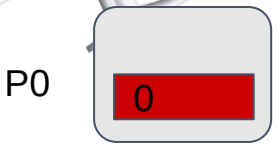
MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;

- MPI_MAX
- MPI_MIN
- MPI_SUM
- MPI_PROD
- MPI_LAND
- :
- :
- :

MPI - Reduce

Syntax :

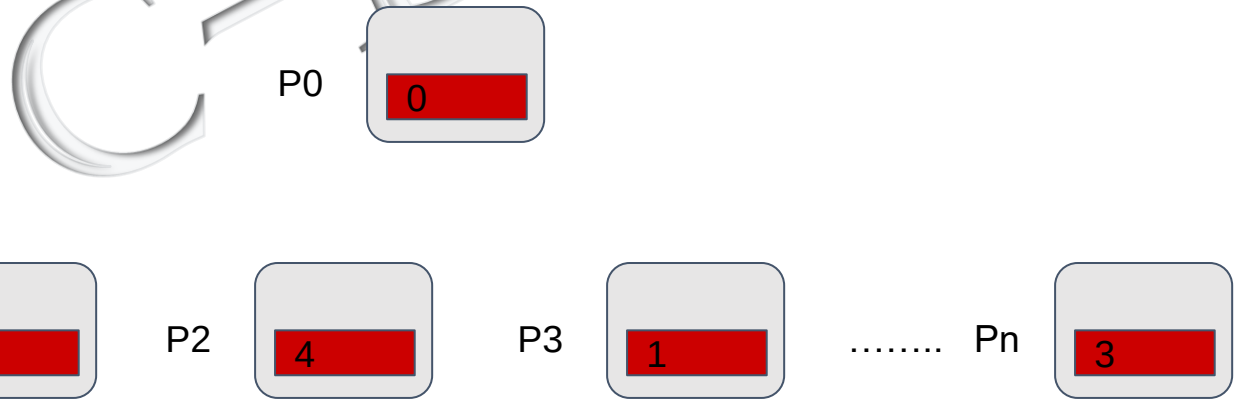
MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;



MPI - Reduce

Syntax :

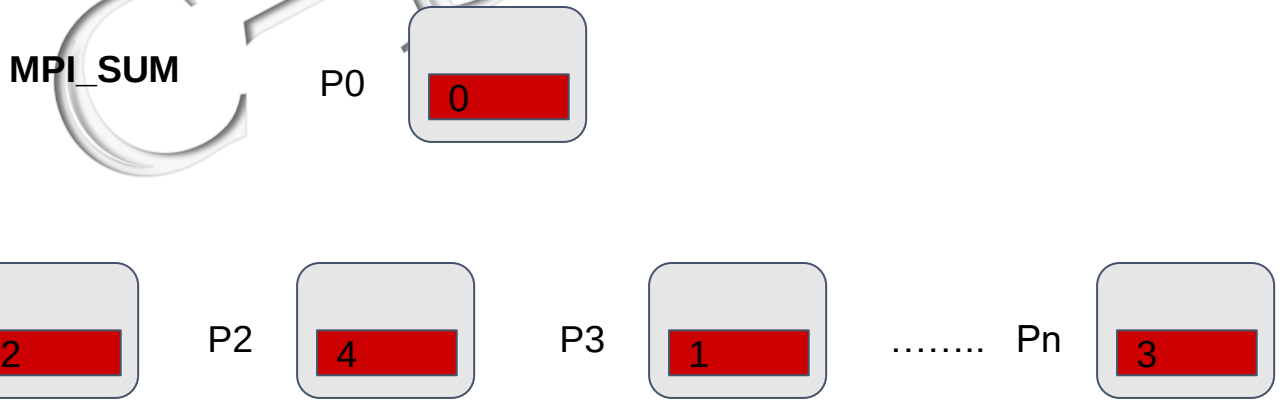
MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;



MPI - Reduce

Syntax :

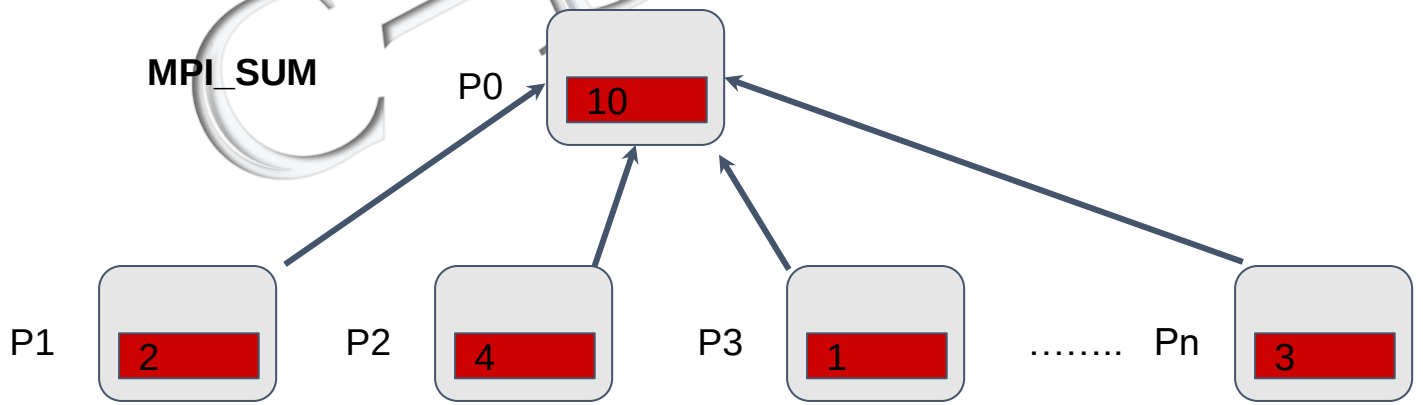
MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;



MPI - Reduce

Syntax :

MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;



MPI - Reduce

Syntax :

MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;

Example : Many lines in Trap. example programs are replaced by this single line ...

MPI - Reduce

Syntax :

MPI_Reduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , Int Dest_process, MPI_Comm comm) ;

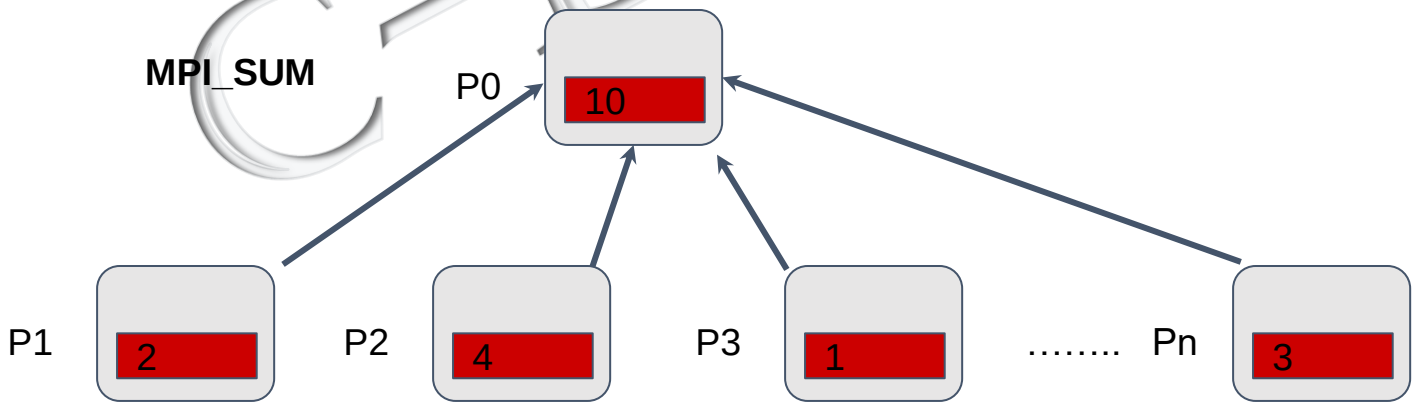
Example : Many lines in Trap. example programs are replaced by this single line ...

```
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,
                    MPI_COMM_WORLD) ;
```

MPI - Allreduce

Syntax :

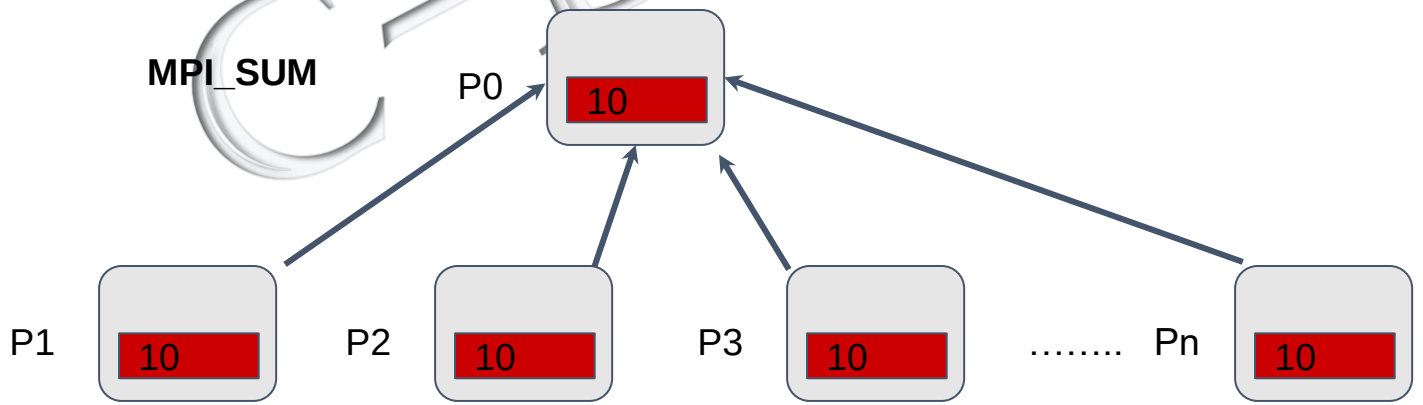
MPI_Allreduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , MPI_Comm comm) ;



MPI - Allreduce

Syntax :

MPI_Allreduce (void* input_data , void* output_data , Int count , MPI_Datatype datatype , MPI_Op operator , MPI_Comm comm) ;

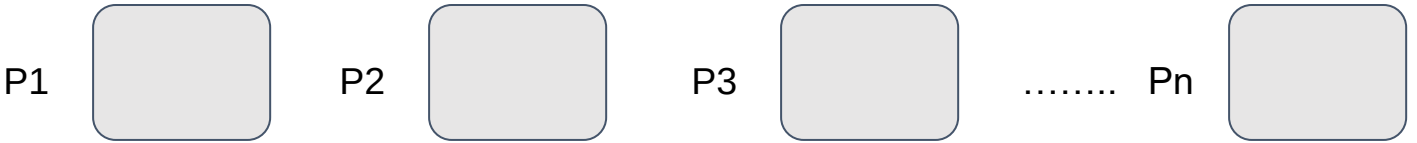


MPI - Scatter

Syntax :

```
MPI_Scatter (void* send_buffer , Int send_count , MPI_Datatype
             send_datatype , void* recv_buffer , Int recv_count ,
             MPI_Datatype recv_datatype , Int source_process ,
             MPI_Comm comm ) ;
```

- MPI_Scatter sends chunks of data to different processes..

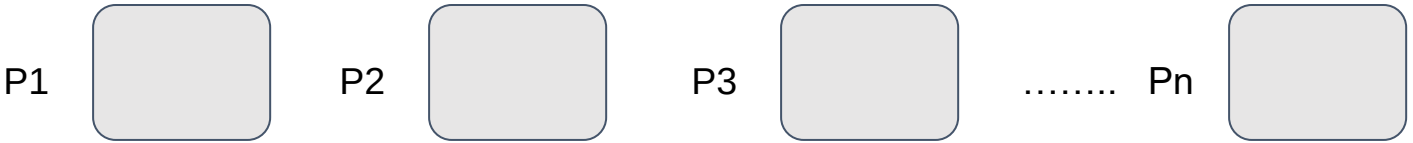


MPI - Scatter

Syntax :

```
MPI_Scatter (void* send_buffer , Int send_count , MPI_Datatype
             send_datatype , void* recv_buffer , Int recv_count ,
             MPI_Datatype recv_datatype , Int source_process ,
             MPI_Comm comm ) ;
```

- MPI_Scatter sends chunks of data to different processes..

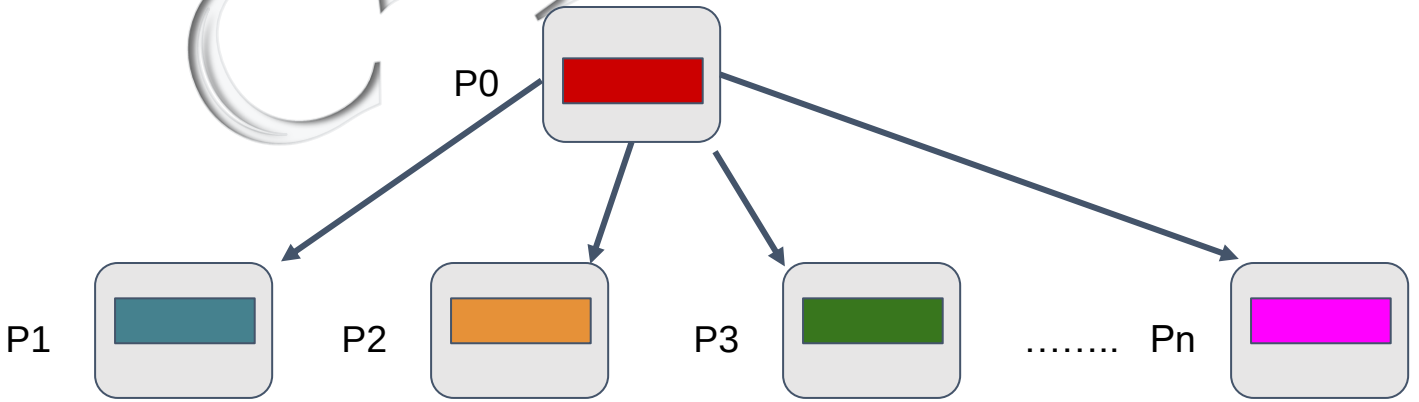


MPI - Scatter

Syntax :

```
MPI_Scatter (void* send_buffer , Int send_count , MPI_Datatype
             send_datatype , void* recv_buffer , Int recv_count ,
             MPI_Datatype recv_datatype , Int source_process ,
             MPI_Comm comm ) ;
```

- MPI_Scatter sends chunks of data to different processes..

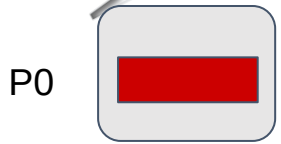


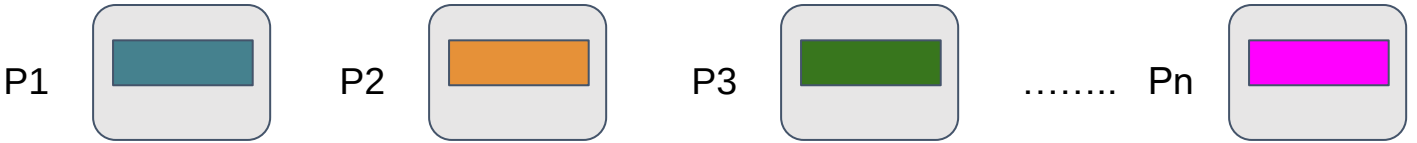
MPI - Gather

Syntax :

```
MPI_Gather (void* send_buffer , Int send_count , MPI_Datatype
            send_datatype , void* recv_buffer , Int recv_count ,
            MPI_Datatype recv_datatype , Int destination_process
,
            MPI_Comm comm ) ;
```

- MPI_Gather collects chunks of data from different processes..



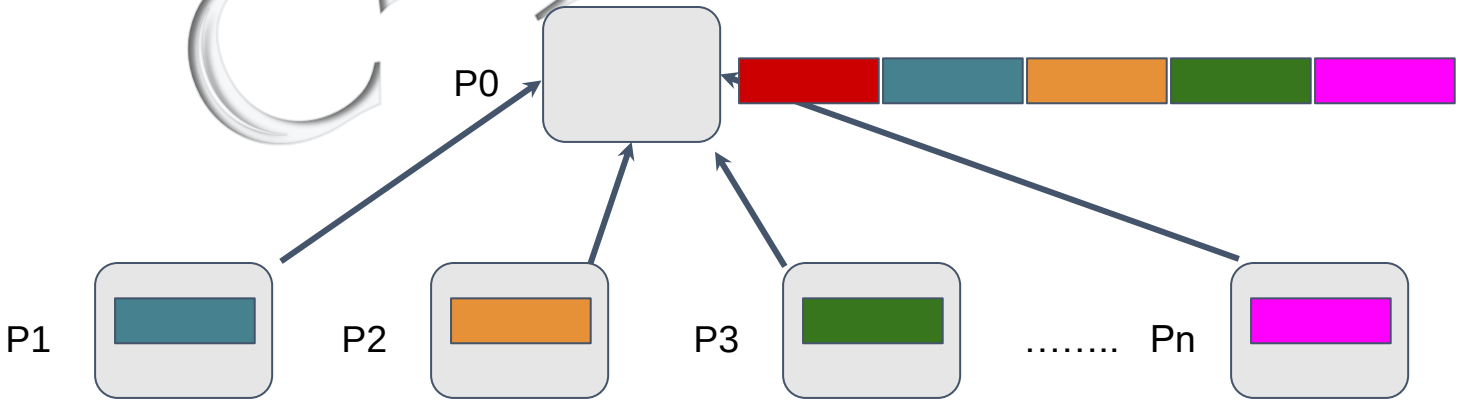


MPI - Gather

Syntax :

MPI_Gather (void* send_buffer , Int send_count , MPI_Datatype send_datatype , void* recv_buffer , Int recv_count , MPI_Datatype recv_datatype , Int destination_process , MPI_Comm comm) ;

- MPI_Gather collects chunks of data from different processes..

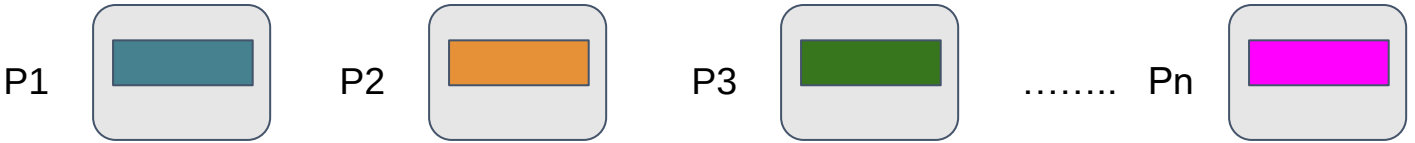


MPI - Gather

Syntax :

MPI_Gather (void* send_buffer , Int send_count , MPI_Datatype send_datatype , void* recv_buffer , Int recv_count , MPI_Datatype recv_datatype , Int destination_process , MPI_Comm comm) ;

- MPI_Gather collects chunks of data from different processes..



MPI - Allgather

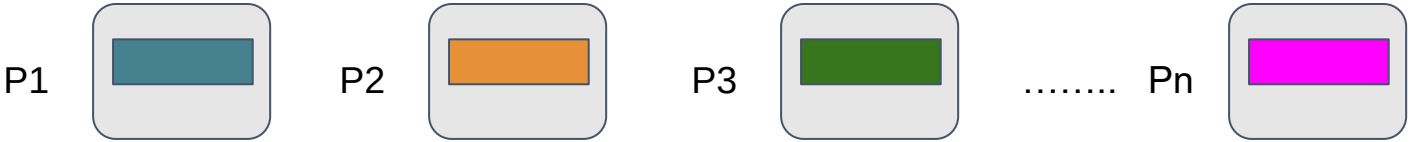
Syntax :

```
MPI_Allgather (void* send_buffer , Int send_count , MPI_Datatype
               send_datatype , void* recv_buffer , Int recv_count ,
               MPI_Datatype recv_datatype ,          ,
               MPI_Comm comm ) ;
```

MPI - Allgather

Syntax :

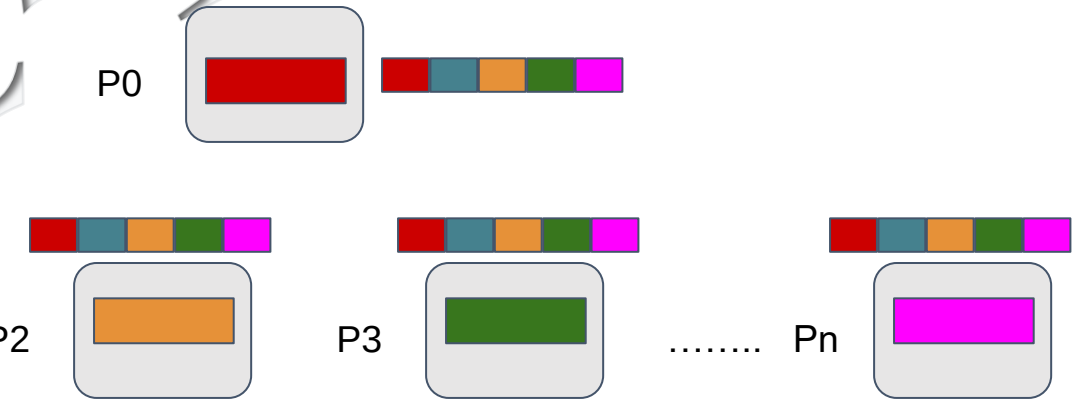
```
MPI_Allgather (void* send_buffer , Int send_count , MPI_Datatype
        send_datatype , void* recv_buffer , Int recv_count ,
        MPI_Datatype recv_datatype ,          ,
        MPI_Comm comm ) ;
```



MPI - Allgather

Syntax :

```
MPI_Allgather (void* send_buffer , Int send_count , MPI_Datatype
               send_datatype , void* recv_buffer , Int recv_count ,
               MPI_Datatype recv_datatype ,                ,
               MPI_Comm comm ) ;
```



MPI - Synchronization

MPI - Barrier

Syntax :

MPI_Barrier (MPI_Comm communicator) ;

MPI - Barrier

Syntax :

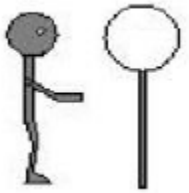
MPI_Barrier (MPI_Comm communicator) ;

- Used to block the calling process until all processes have entered the function. The call will return at any process only after all the processes or group members have entered the call
- The MPI_BARRIER routine blocks the calling process until all group processes have called the function. When MPI_BARRIER returns, all processes are synchronized at the barrier

MPI - Barrier

Syntax :

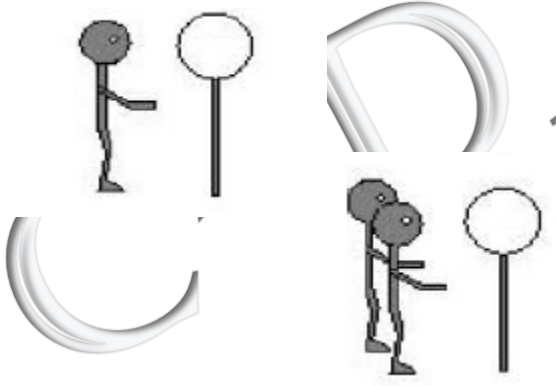
MPI_Barrier (MPI_Comm communicator) ;



MPI - Barrier

Syntax :

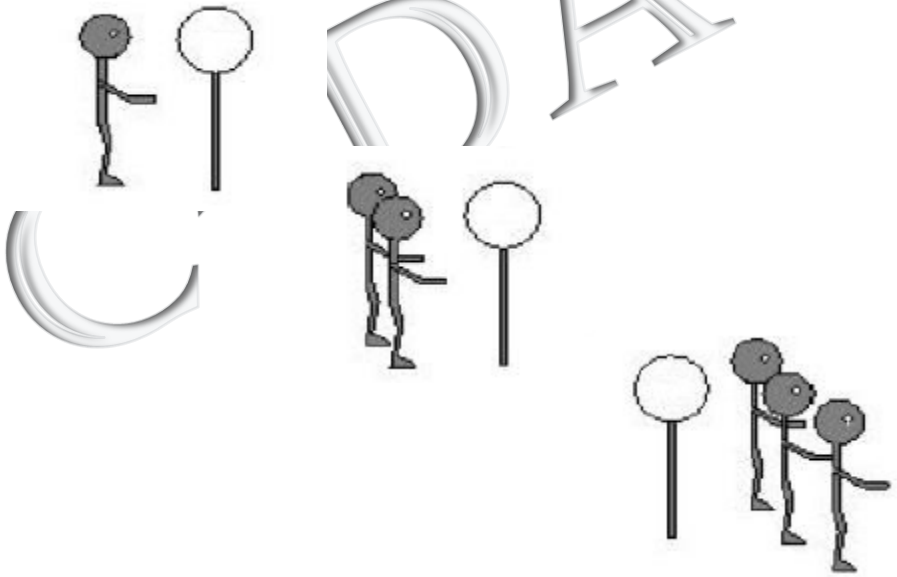
MPI_Barrier (MPI_Comm communicator) ;



MPI - Barrier

Syntax :

MPI_Barrier (MPI_Comm communicator) ;



Recap :

- Point to Point Vs Collective communication -
- MPI_Broadcast(...)
- MPI_Scatter(...)
- MPI_Reduce(...)
- MPI_Allreduce(...)
- MPI_Gather(...)
- MPI_Allgather(...)
- Miss MPI routines !
- ....

References :

[1] Barker, Brandon. "Message passing interface (mpi)." *Workshop: High Performance Computing on Stampede*. Vol. 262. 2015.

[2] Yuan, Chung-Tsz, and Shenjian Chen. "Message Passing Interface (MPI)." (1996).

[3] https://computing.llnl.gov/tutorials/mpi/

C-DAC

MPI_Comm_rank(.....)

MPI_Comm_rank(.....)

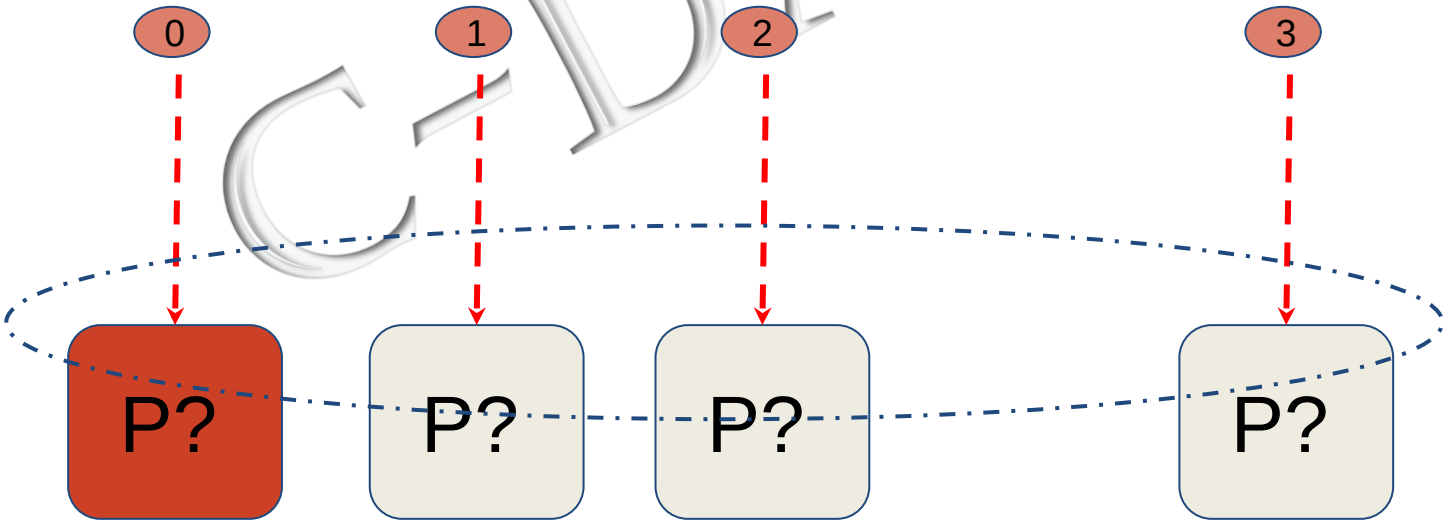
Syntax :

MPI_Comm_rank (MPI_Comm communicator , int * rank) ;

MPI_Comm_rank(.....)

Syntax :

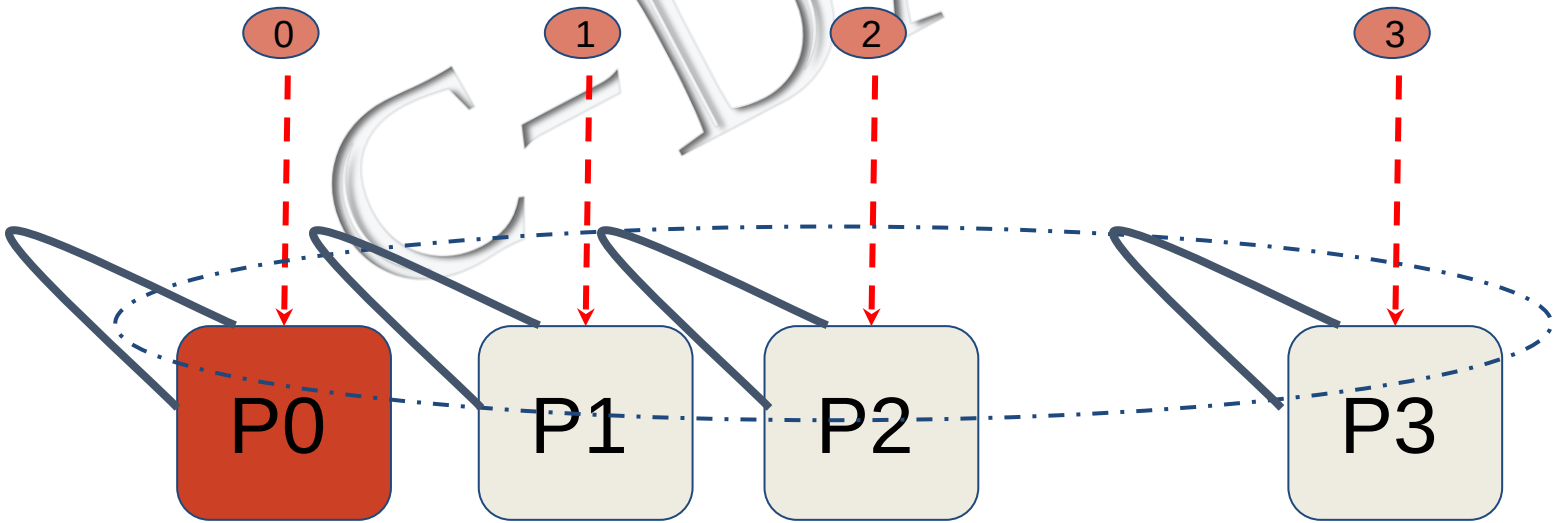
MPI_Comm_rank (MPI_Comm communicator , int * rank) ;



MPI_Comm_rank(.....)

Syntax :

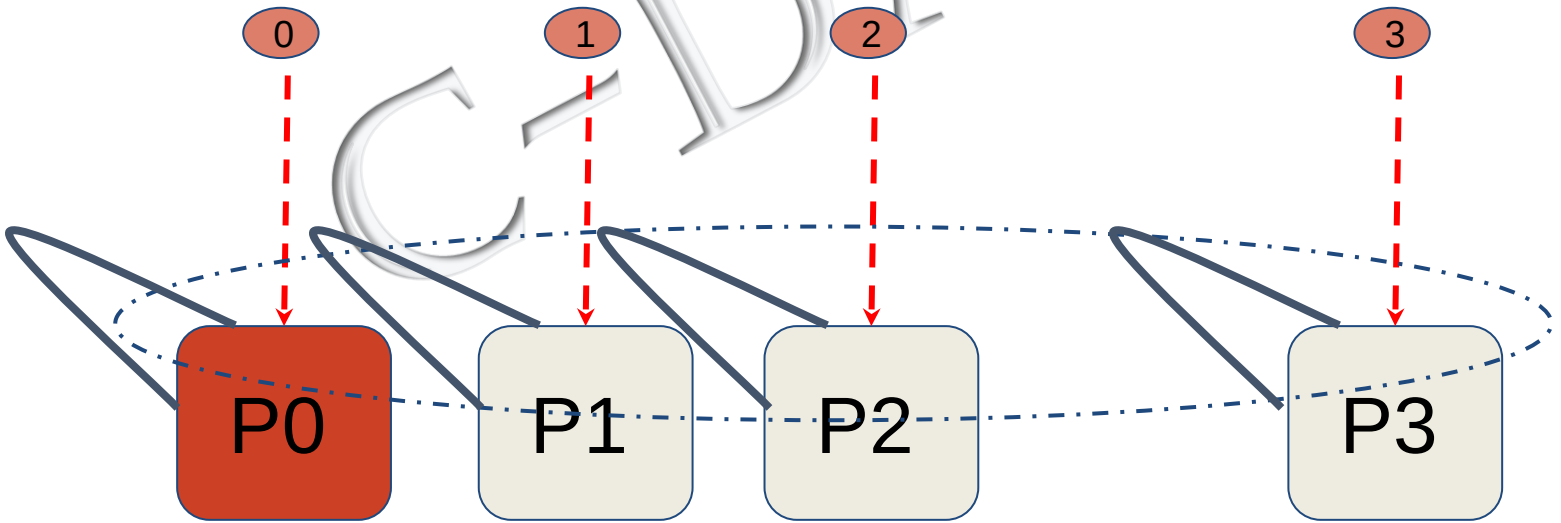
MPI_Comm_rank (MPI_Comm communicator , int * rank) ;



MPI_Comm_rank(.....)

Syntax :

MPI_Comm_rank (MPI_Comm communicator , int * rank) ;



MPI_Comm_size(.....)

MPI_Comm_size(.....)

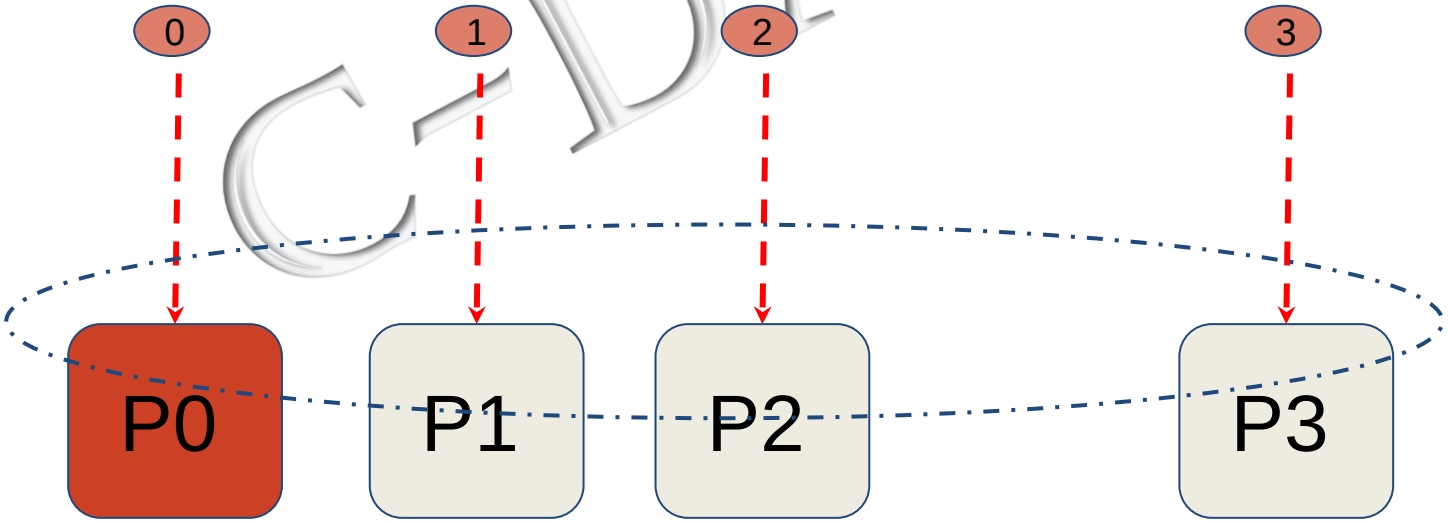
Syntax :

MPI_Comm_size (MPI_Comm communicator , int * size) ;

MPI_Comm_size(.....)

Syntax :

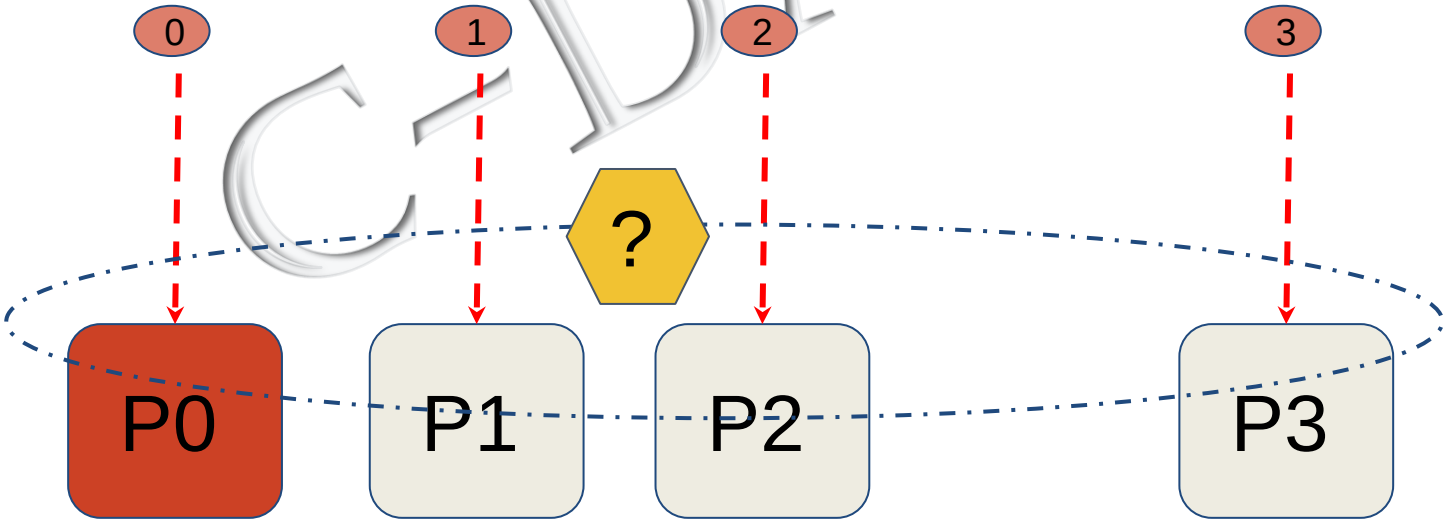
MPI_Comm_size (MPI_Comm communicator , int * size) ;



MPI_Comm_size(.....)

Syntax :

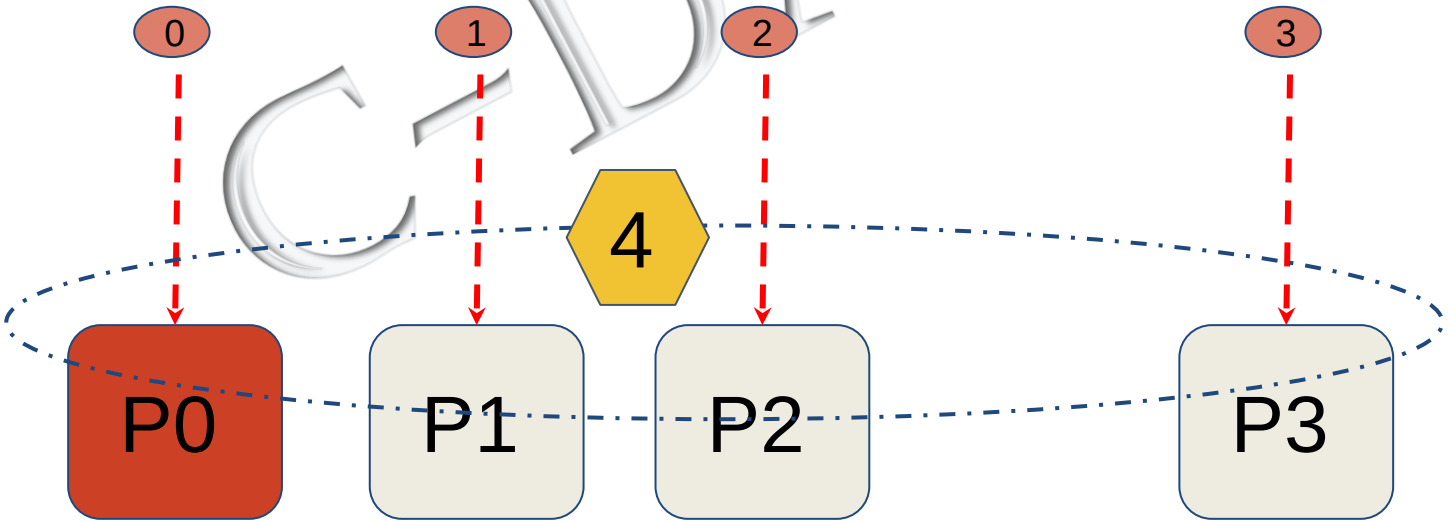
MPI_Comm_size (MPI_Comm communicator , int * size) ;



MPI_Comm_size(.....)

Syntax :

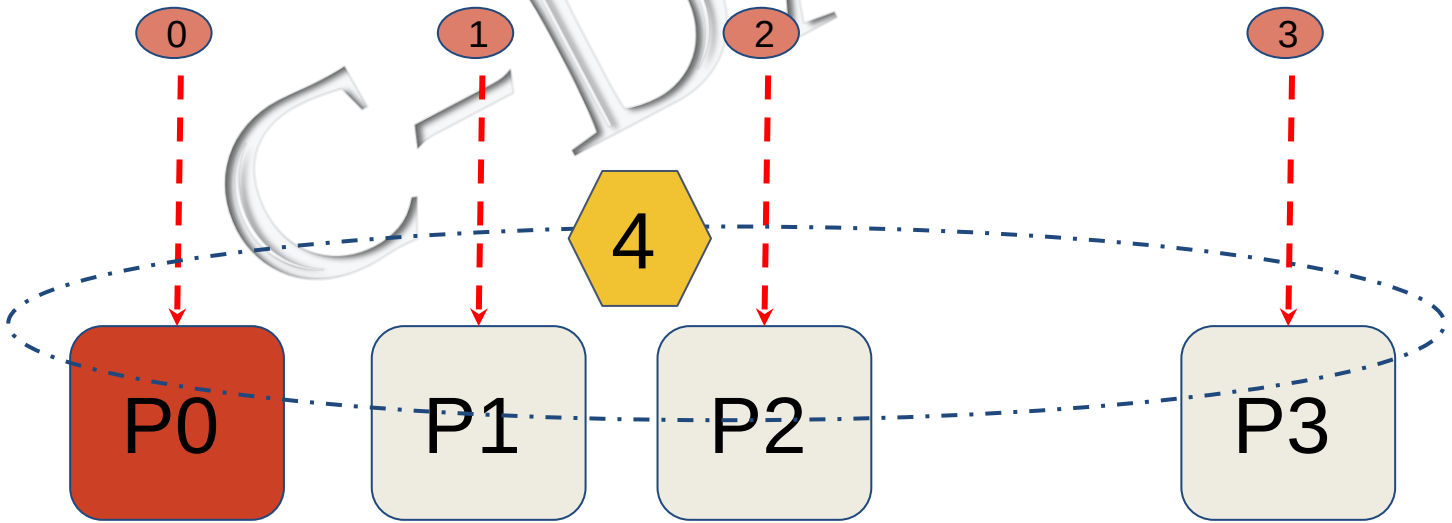
MPI_Comm_size (MPI_Comm communicator , int * size) ;



MPI_Comm_size(.....)

Syntax :

MPI_Comm_size (MPI_Comm communicator , int * size) ;



MPI_Send(.....)

MPI_Send(.....)

Syntax :

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

1 Address of Message buffer

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

1 Address of Message buffer

2 Message size

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

1. Address of Message buffer
2. Message size
3. Data Type

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

1. Address of Message buffer
2. Message size
3. Data Type
4. Destination process rank

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

1. Address of Message buffer
2. Message size
3. Data Type
4. Destination process rank
5. Tag - Message Identifier/..

MPI_Send(.....)

Syntax :

1 2 3

```
MPI_Send (void*  msg_buffer , Int  msg_size, MPI_Datatype  msg_type,
          Int destination,  Int  tag , MPI_Comm  communicator ) ;
```

4 5 6

1. Address of Message buffer
2. Message size
3. Data Type
4. Destination process rank
5. Tag - Message Identifier/..
6. Communicator

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

1 2 3

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

Tell us information about message

1 2 3

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

Tell us information about message

1 2 3 4 5 6

MPI_Send(.....)

Syntax :

1 2 3

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

4 5 6

Tell us information about message

Tell us, where and How to send a message

1 2 3

4 5 6

MPI_Recv(.....)

MPI_Recv(.....)

Syntax :

MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
Int source, Int tag , MPI_Comm communicator, MPI_Status*);

MPI_Recv(.....)

Syntax :

1 2 3

MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
Int source, Int tag , MPI_Comm communicator, MPI_Status*);

4 5 6 7

MPI_Recv(.....)

Syntax :

1 2 3

MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
Int source, Int tag , MPI_Comm communicator, MPI_Status*);

4 5 6 7

1. Address of Message buffer
2. Buffer size
3. Data Type

MPI_Recv(.....)

Syntax :

1 2 3

MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
Int source, Int tag , MPI_Comm communicator, MPI_Status*);

4 5 6 7

1. Address of Message buffer
2. Buffer size
3. Data Type
4. Source process rank
5. Tag - Message Identifier/..
6. Communicator

MPI_Recv(.....)

Syntax :

1 2 3

MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
Int source, Int tag , MPI_Comm communicator, MPI_Status*);

4 5 6 7

1. Address of Message buffer
2. Buffer size
3. Data Type
4. Source process rank
5. Tag - Message Identifier/..
6. Communicator
7. Status of Received message

Successful transmission of Message..

- MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
  Int destination, Int tag , MPI_Comm communicator) ;

- MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
  Int source, Int tag , MPI_Comm communicator, MPI_Status*);

Successful transmission of Message..

MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;

MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type,
Int source, Int tag , MPI_Comm communicator, MPI_Status*);

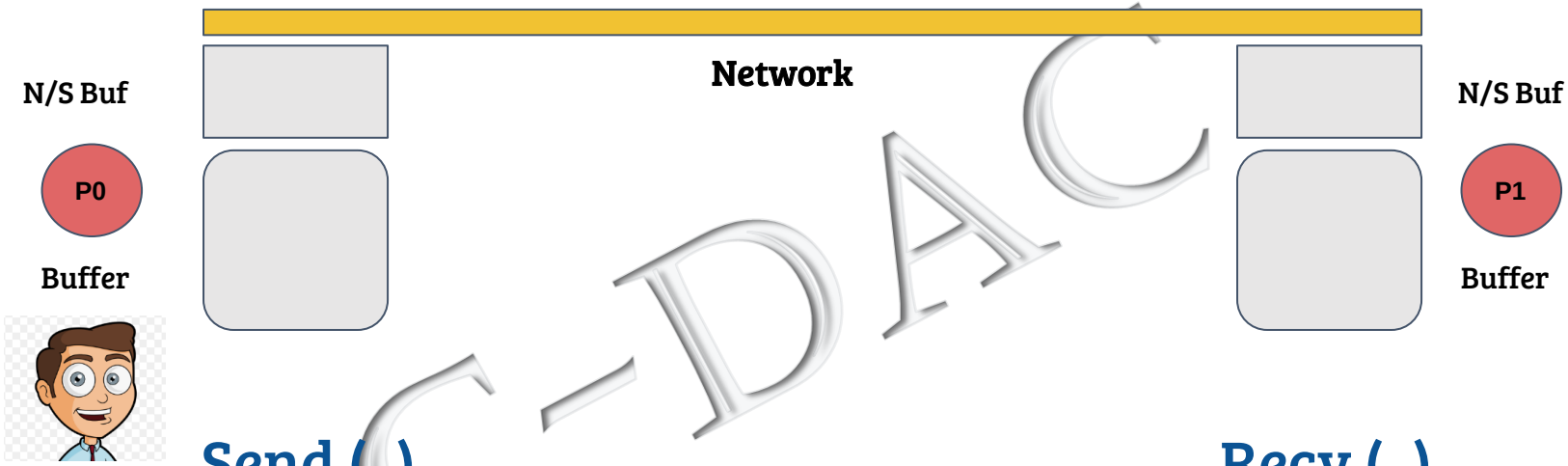
recv_comm = send_comm
recv_tag = send_tag
dest = Destination process rank
Src = Source process rank

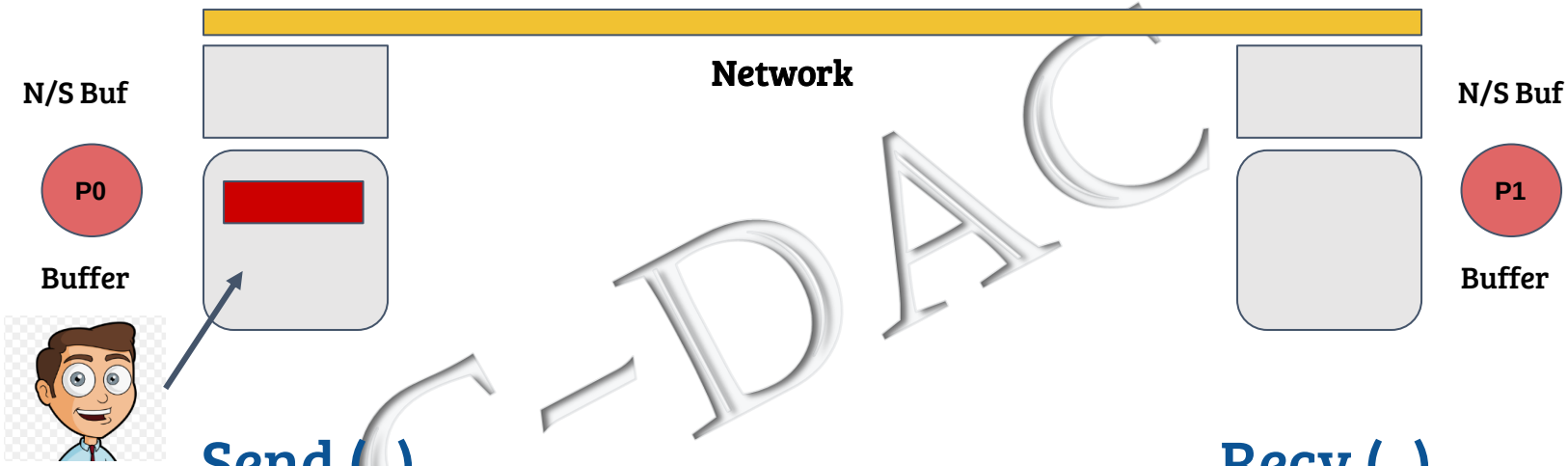
How message is transferred … ?

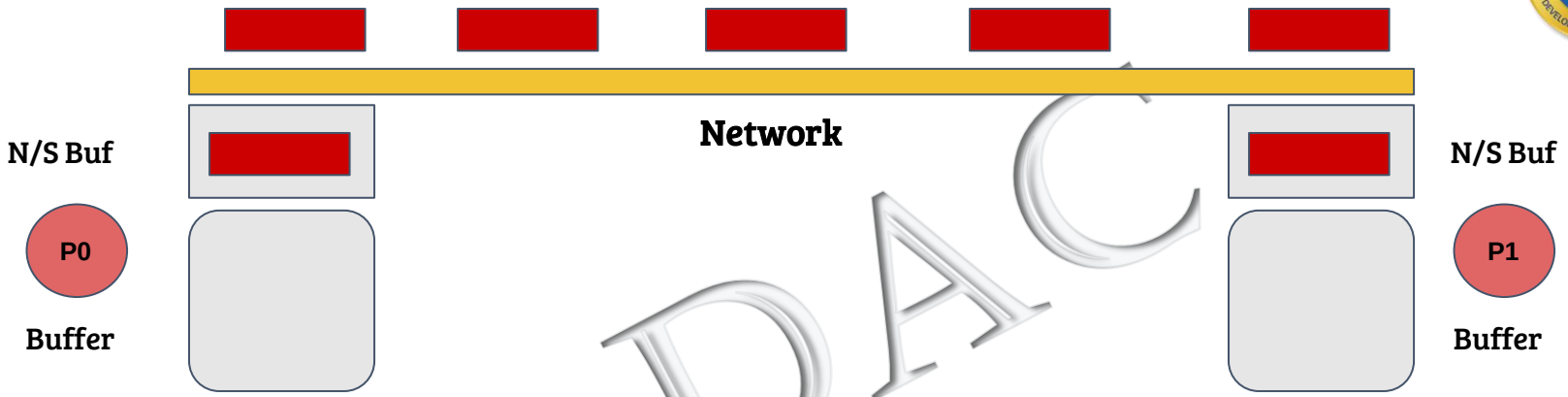
… Different cases !

Send (..)

Recv (..)





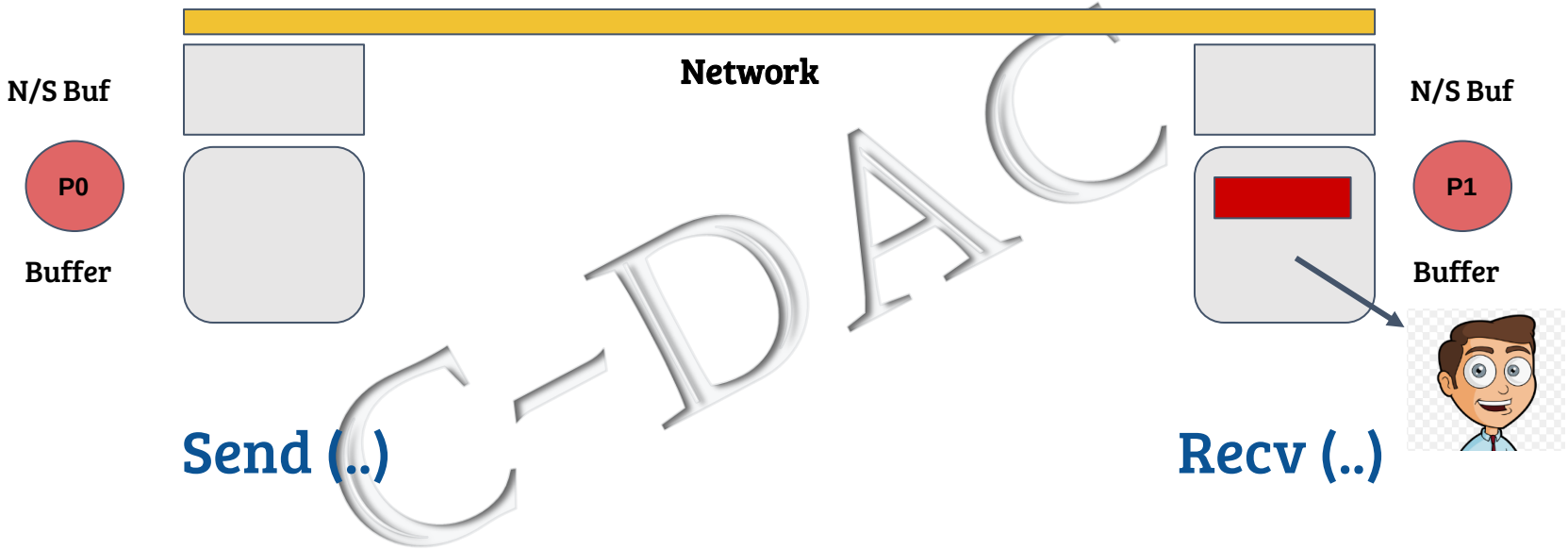


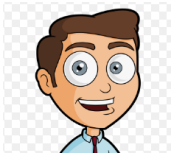
Send (..)

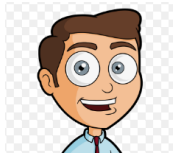
Recv (..)

Send (..)

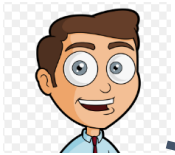
Recv (..)



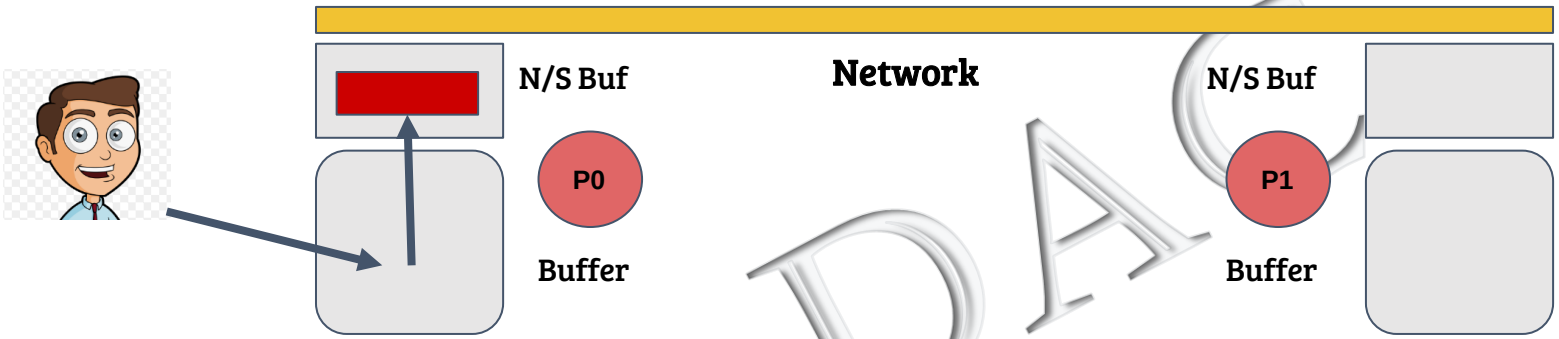




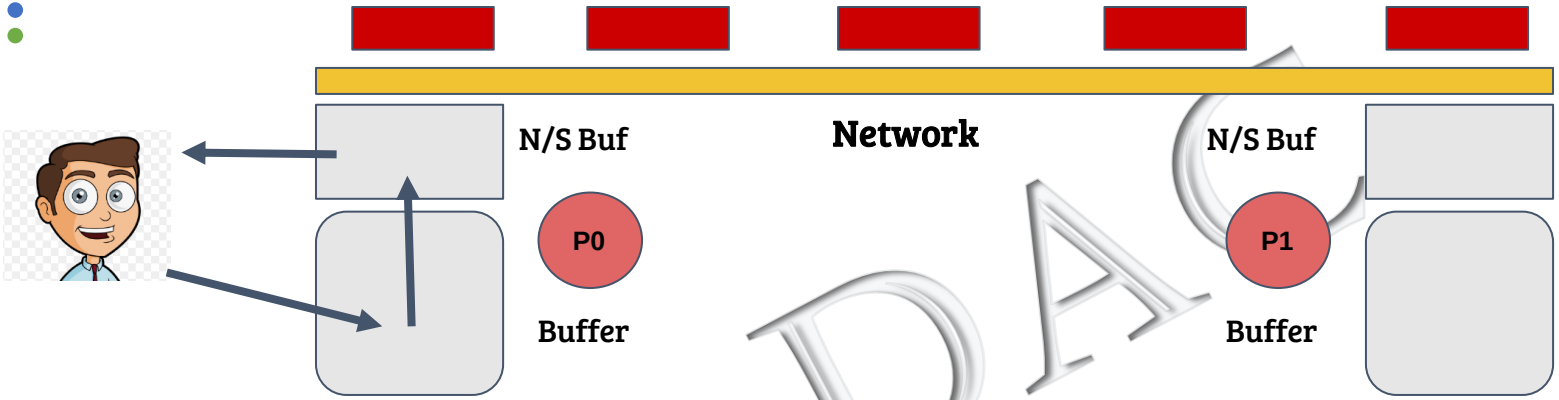
Case 1 : Blocking - Point to Point Communication



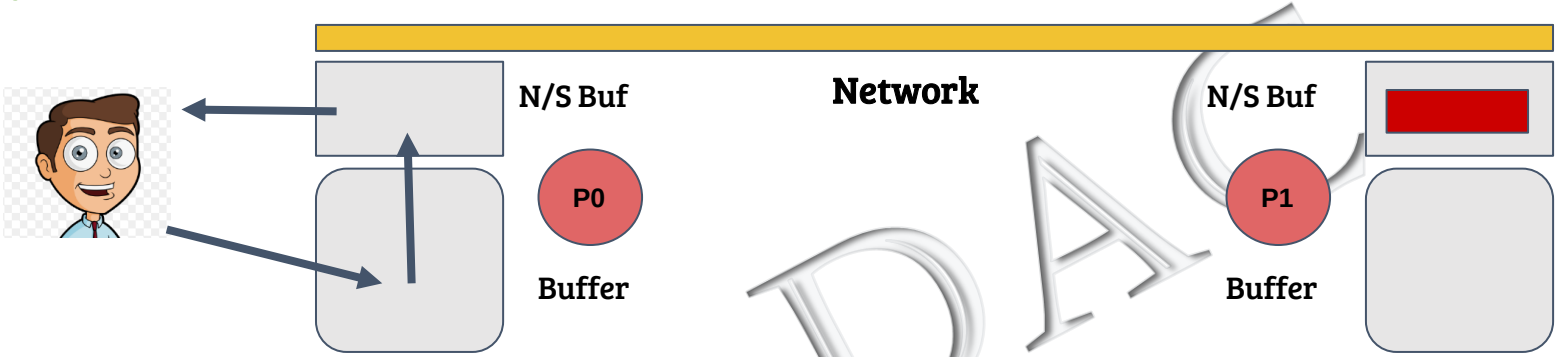
Case 1 : Blocking - Point to Point Communication



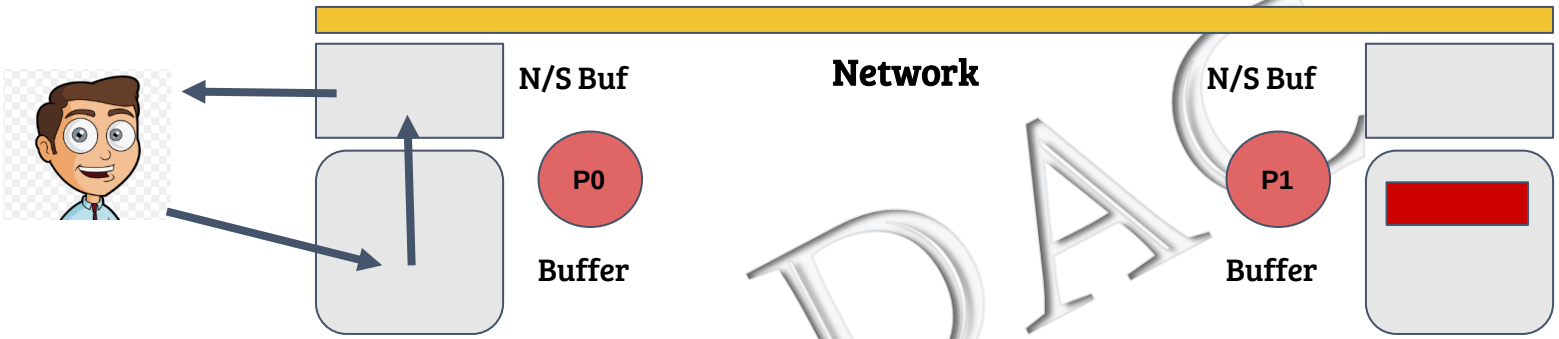
Case 1 : Blocking - Point to Point Communication



Case 1 : Blocking - Point to Point Communication



Case 1 : Blocking - Point to Point Communication

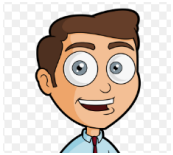


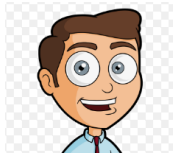
Case 1 : Blocking - Point to Point Communication

Case 1 : Blocking - Point to Point Communication

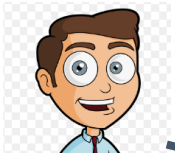
- MPI_Send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type, Int destination, Int tag , MPI_Comm communicator) ;

- MPI_Recv (void* msg_buffer , Int buf_size, MPI_Datatype buf_type, Int source, Int tag , MPI_Comm communicator, MPI_Status*);

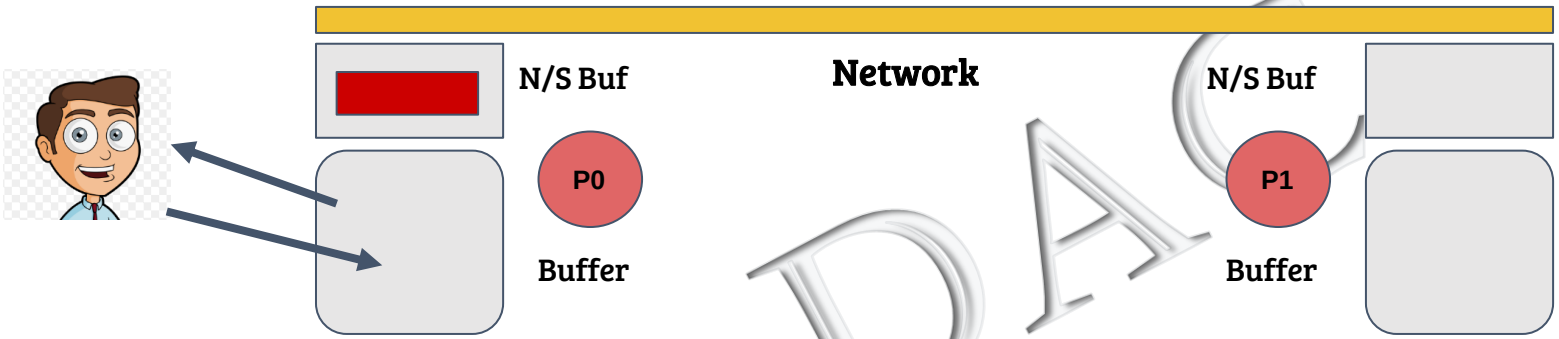




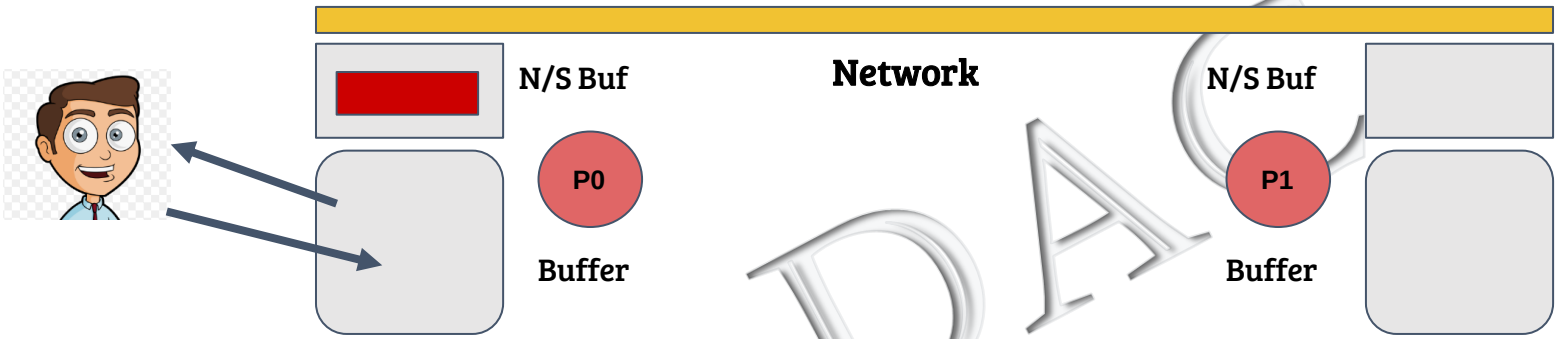
★ Case 2 : Non Blocking - Point to Point Communication



★ Case 2 : Non Blocking - Point to Point Communication



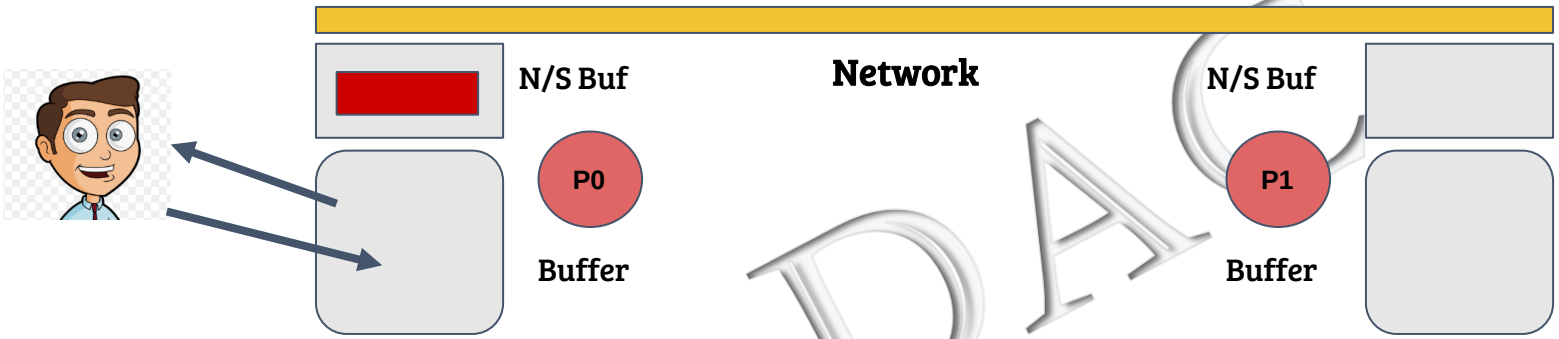
★ Case 2 : Non Blocking - Point to Point Communication



★ Case 2 : Non Blocking - Point to Point Communication

Case 2 : Non Blocking - Point to Point Communication

MPI_Isend (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator, req *) ;



★ Case 2 : Non Blocking - Point to Point Communication

Case 2 : Non Blocking - Point to Point Communication

How we will know, whether that message has been transferred or not ?

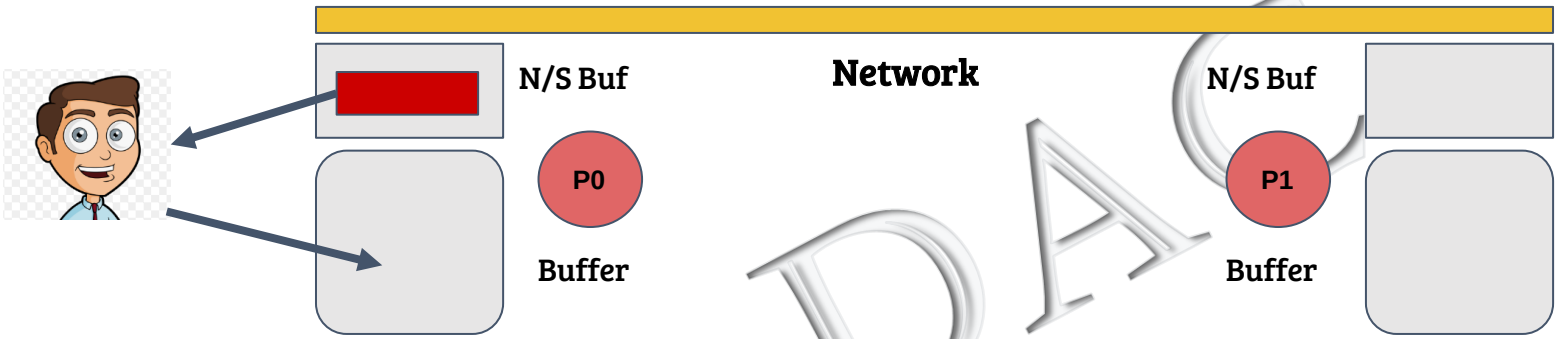
Case 2 : Non Blocking - Point to Point Communication

How we will know, whether that message has been transferred or not ?

MPI_Test(MPI_Request *req, int *flag, MPI_Status *status) ;

Case 2 : Non Blocking - Point to Point Communication

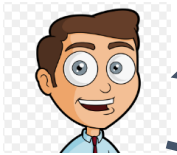
Data may be in buffer till next message arrive ..!

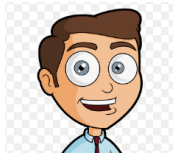


Case 2 : Non Blocking - Point to Point Communication

Data may be in buffer till next message arrive ..!

MPI_Wait(MPI_Request *request, MPI_Status *status)





Case 2 : Non Blocking - Point to Point Communication

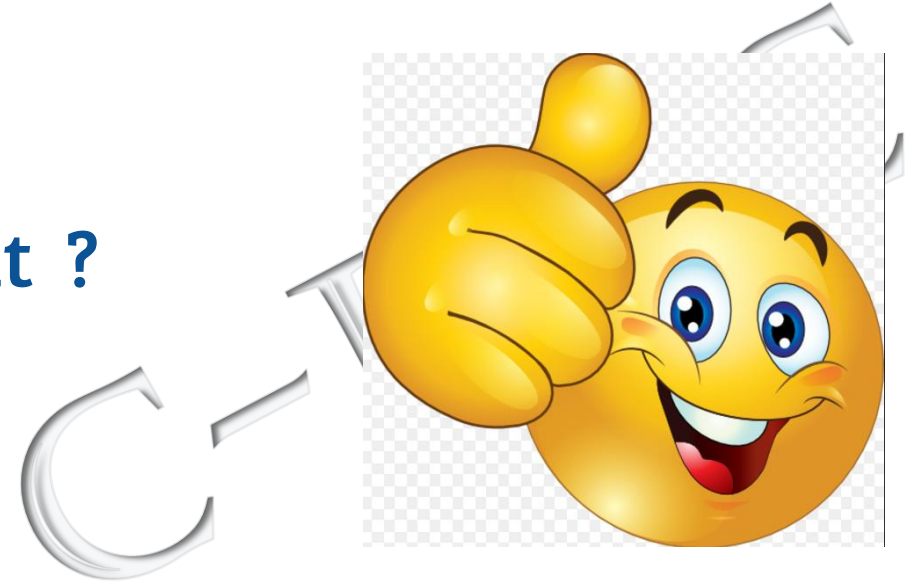
Data may be in buffer till next message arrive ..!

Block and Wait till operation get finish..

MPI_Wait(MPI_Request *request, MPI_Status *status)

❖ Got it ?

❖ Got it ?



❖ Got it ?