

Parallel Computing - MPI

Message Passing Interface



By : Om Jadhav
HPC - Tech, CDAC Pune

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 remember 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 () ;

- ☐ - - - - -

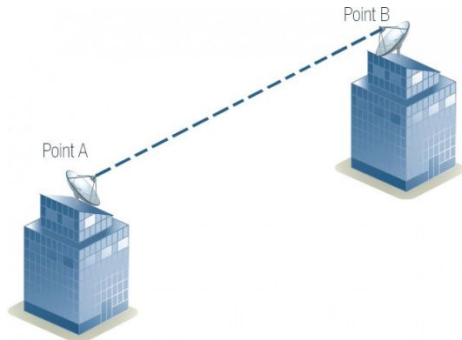
MPI - Communication

MPI - Communication

Point to Point Commⁿ

MPI - Communication

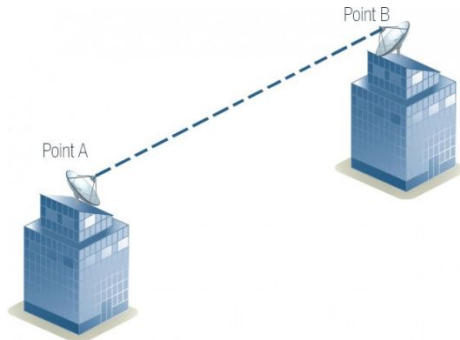
Point to Point Commⁿ



MPI - Communication

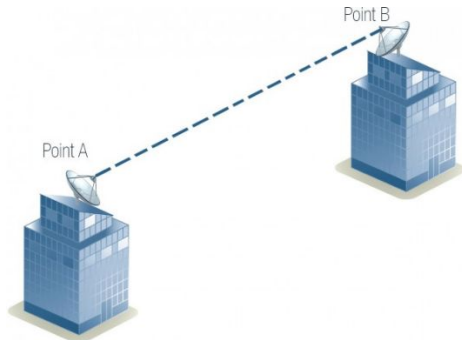
Point to Point Commⁿ

Collective Commⁿ

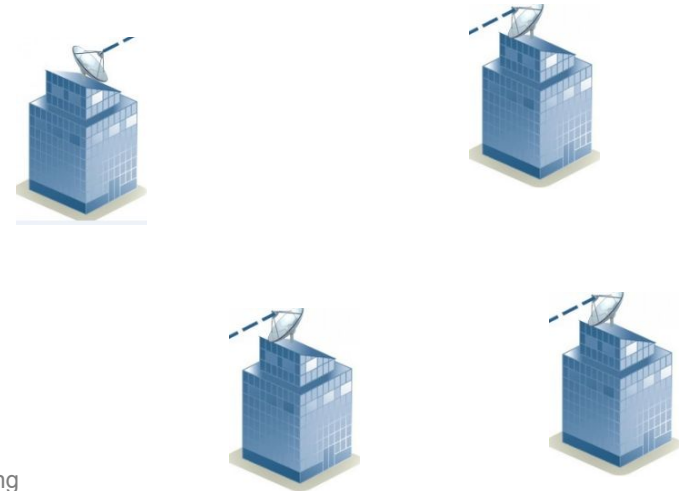


MPI - Communication

Point to Point Commⁿ

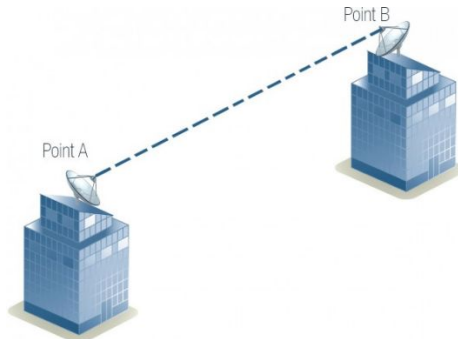


Collective Commⁿ

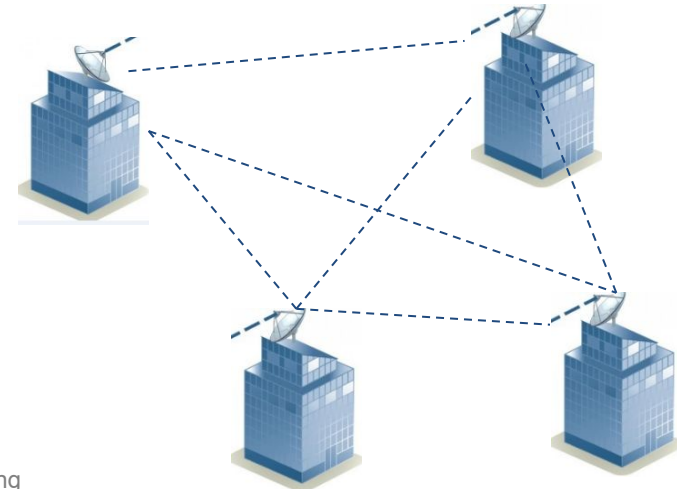


MPI - Communication

Point to Point Commⁿ



Collective Commⁿ



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

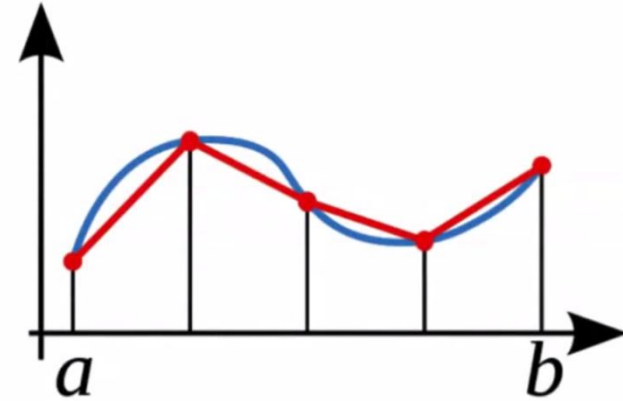
It's Always Better to understand anything with example.....



...Do You Agree ?

Numerical Integration : Trapezoidal Rule

Numerical Integration : Trapezoidal Rule



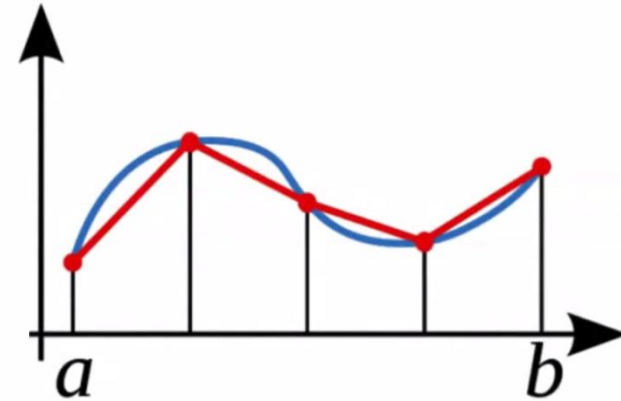
Numerical Integration : Trapezoidal Rule

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 $x_i = a + ih, i = 0, 1, \dots, n$



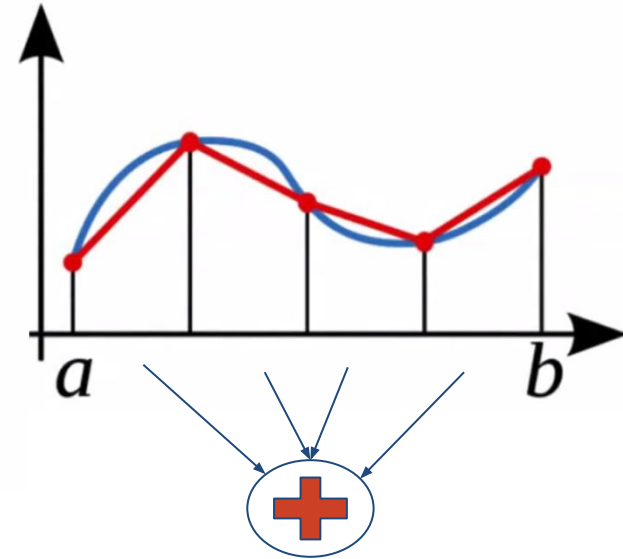
Numerical Integration : Trapezoidal Rule

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 $x_i = a + ih, i = 0, 1, \dots, n$



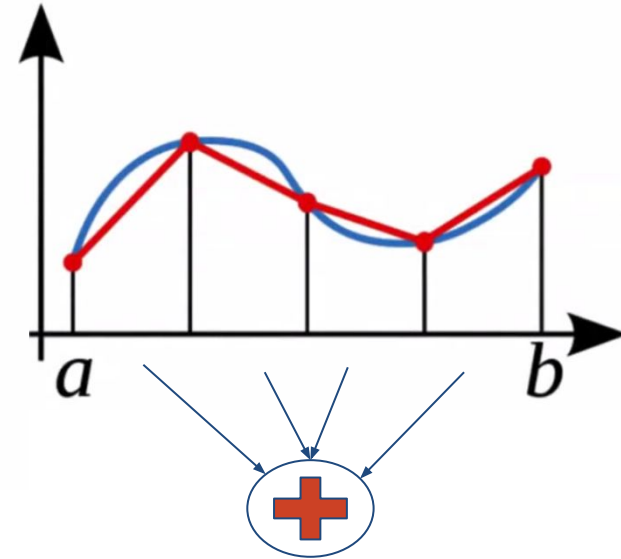
Numerical Integration : Trapezoidal Rule

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 $x_i = a + ih, i = 0, 1, \dots, n$



A decorative graphic on the left side of the slide features a dark green curved shape, a green and yellow patterned area, and a stylized map of India at the bottom left.

How do you achieve it Serially .. ?

```
/* traprul_serial.c */
```

```
float f(float x) ;
```

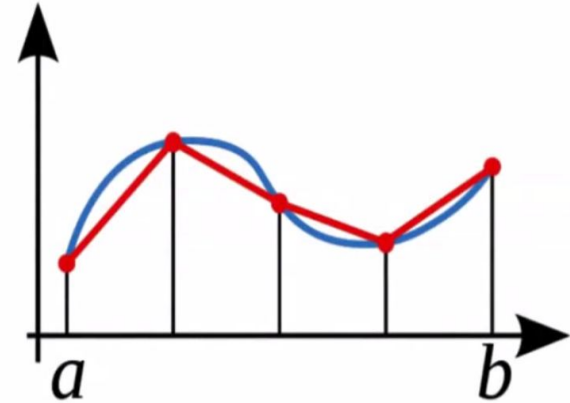


Function which we want to integrate

```
/* trapezule_serial.c */
```

```
float f(float x) ;
```

Function which we want to integrate



```
/* traprul_serial.c */
```

```
float f(float x);
```

Function which we want to integrate

```
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;
```

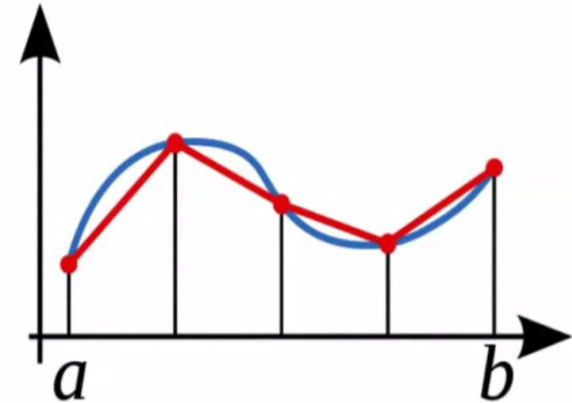
```
        integral = integral + f(x);
```

```
    }
```

```
    return integral*h;
```



```
}
```



```
/* traprul_serial.c */
```

```
float f(float x);
```

Function which we want to integrate

```
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;
```

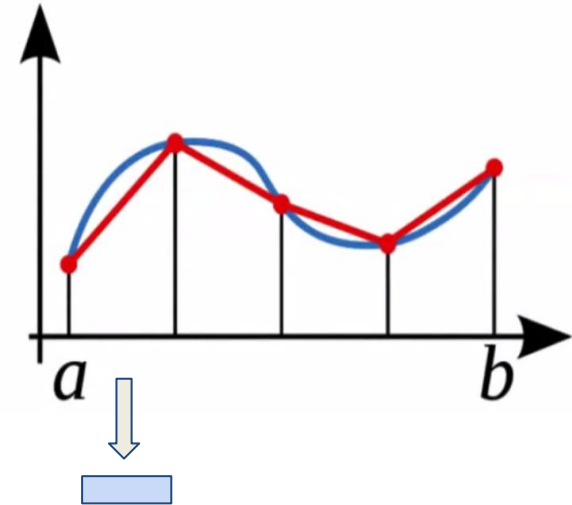
```
        integral = integral + f(x);
```

```
    }
```

```
    return integral*h;
```



```
}
```



```
/* traprul_serial.c */
```

```
float f(float x);
```

Function which we want to integrate

```
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;
```

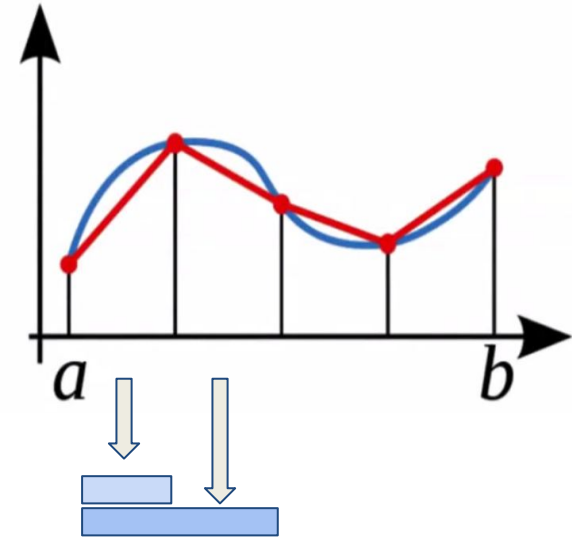
```
        integral = integral + f(x);
```

```
    }
```

```
    return integral*h;
```



```
}
```



```
/* traprul_serial.c */
```

```
float f(float x);
```

Function which we want to integrate

```
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;
```

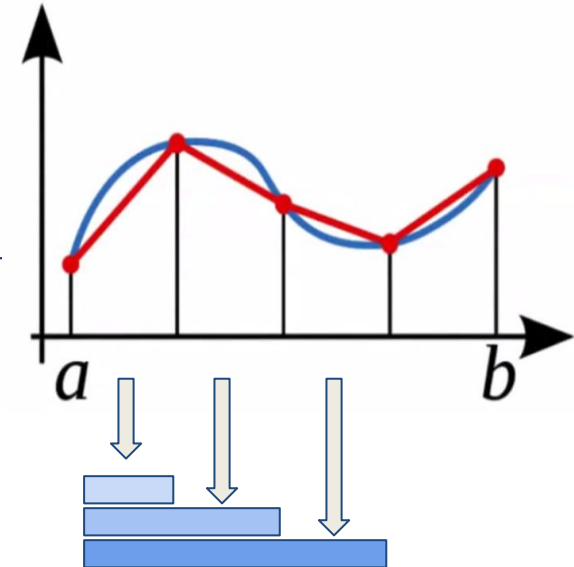
```
        integral = integral + f(x);
```

```
    }
```

```
    return integral*h;
```



```
}
```



```
/* traprul_serial.c */
```

```
float f(float x);
```

Function which we want to integrate

```
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;
```

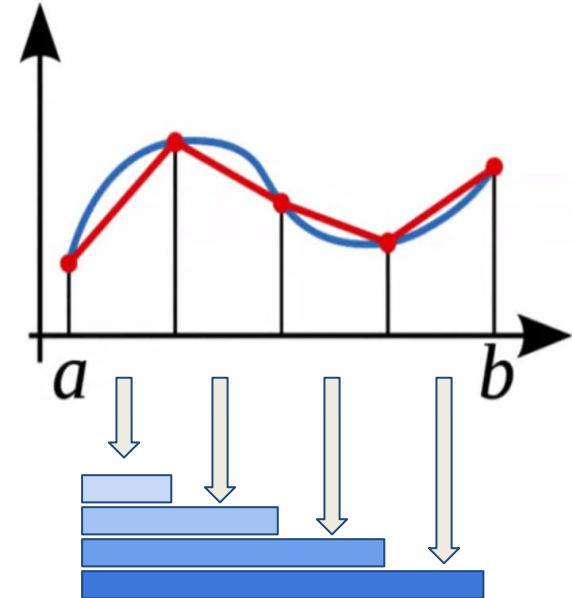
```
        integral = integral + f(x);
```

```
    }
```

```
    return integral*h;
```



```
}
```




```
/* traprul_serial.c */
```

```
float f(float x);
```

Function which we want to integrate

```
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;
```

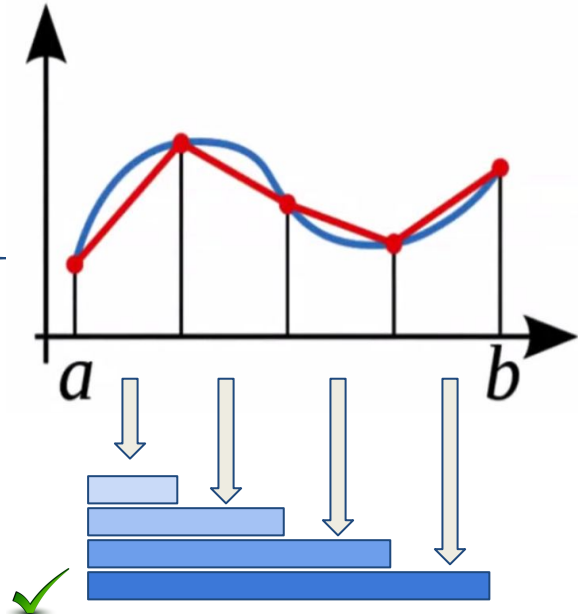
```
        integral = integral + f(x);
```

```
    }
```

```
    return integral*h;
```



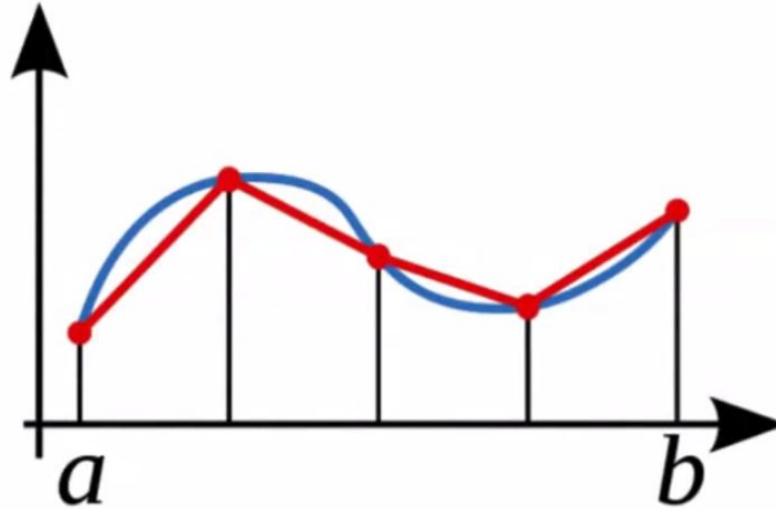
```
}
```



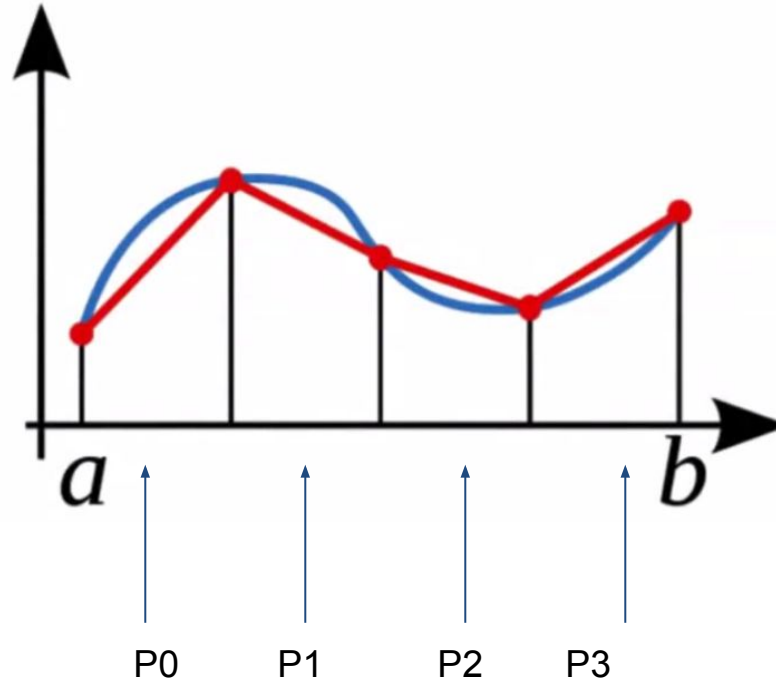
A decorative graphic on the left side of the slide features a dark green curved shape, a green and yellow patterned area, and a stylized map of India at the bottom left.

How we can do it Parallely .. ?

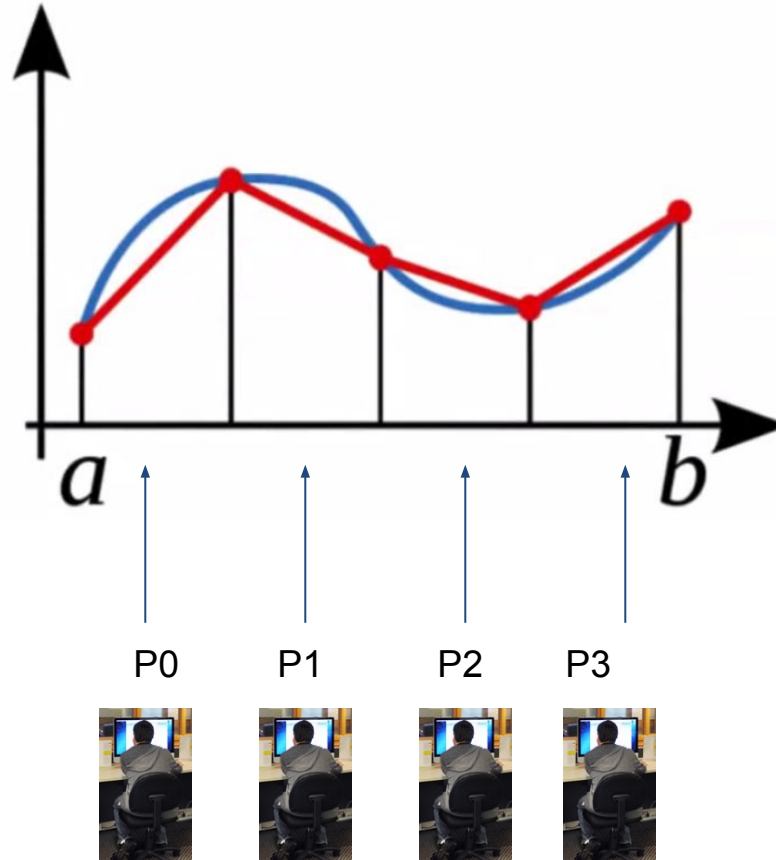
Trap Rule : Parallel Approach



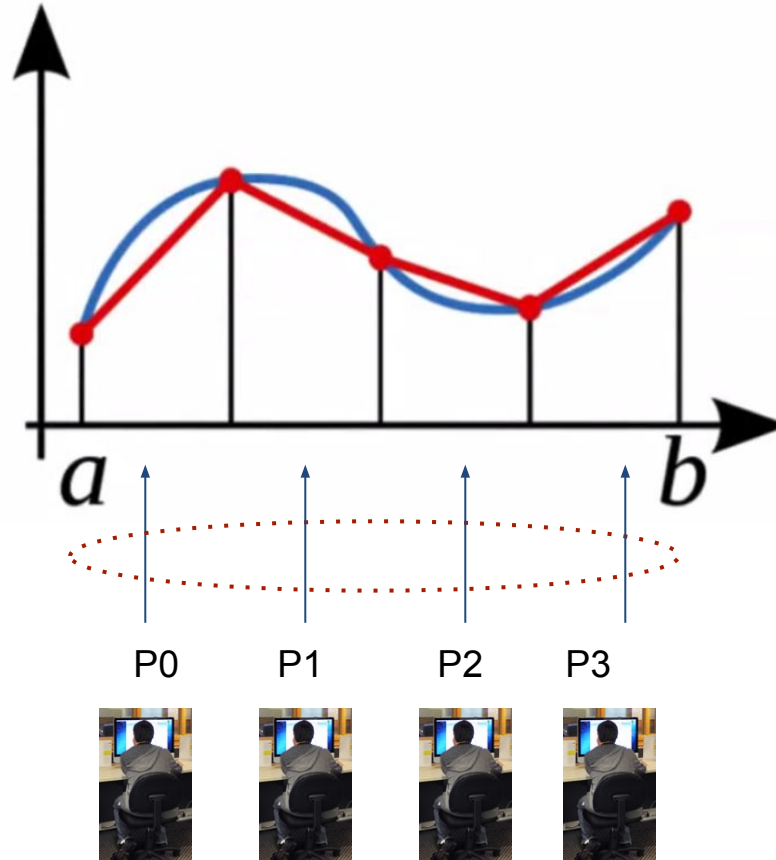
Trap Rule : Parallel Approach



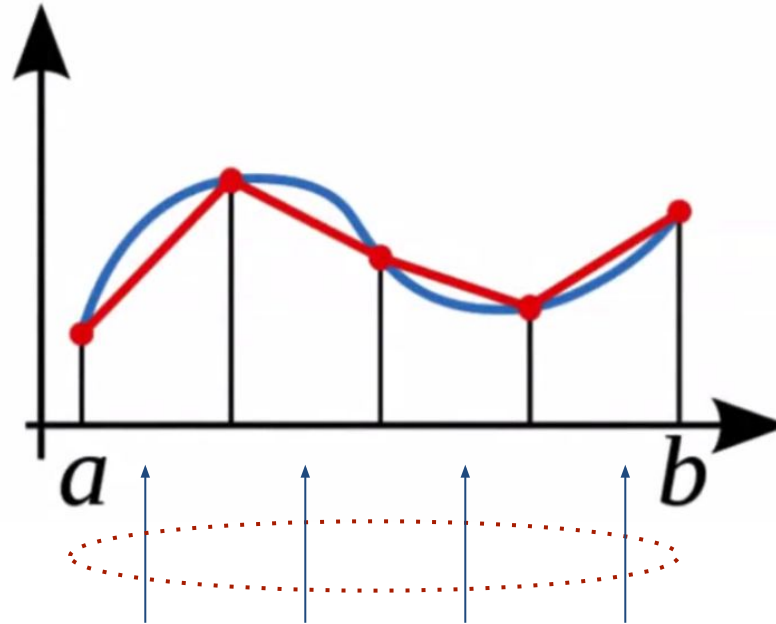
Trap Rule : Parallel Approach



Trap Rule : Parallel Approach



Trap Rule : Parallel Approach

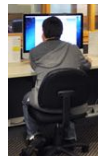


P0

P1

P2

P3



Trap Rule : Parallel Approach

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

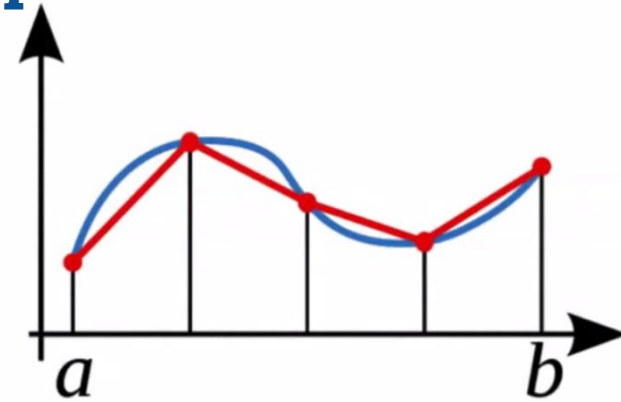
with $h = (b - a)/n$ is

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

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

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

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



I

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

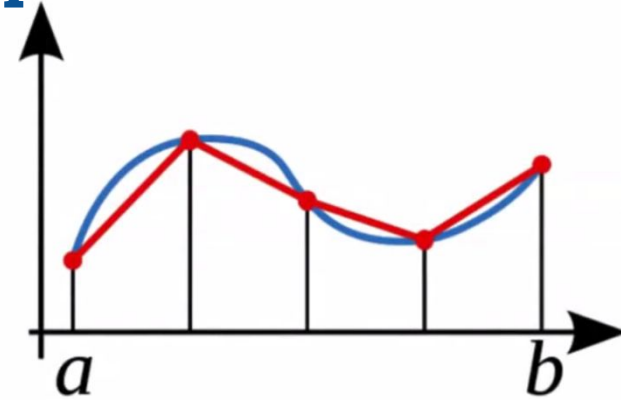
Trap Rule : Parallel Approach

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 $x_i = a + ih, i = 0, 1, \dots, n$



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

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

I

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 $\int_a^b f(x)dx$

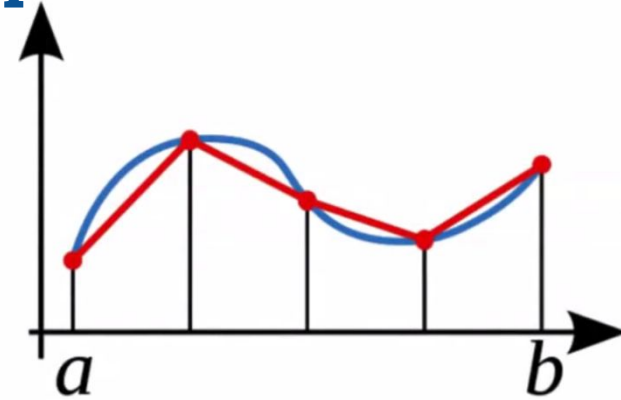
with $h = (b - a)/n$ is

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

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

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

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



I

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 $\int_a^b f(x)dx$

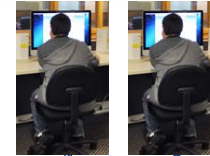
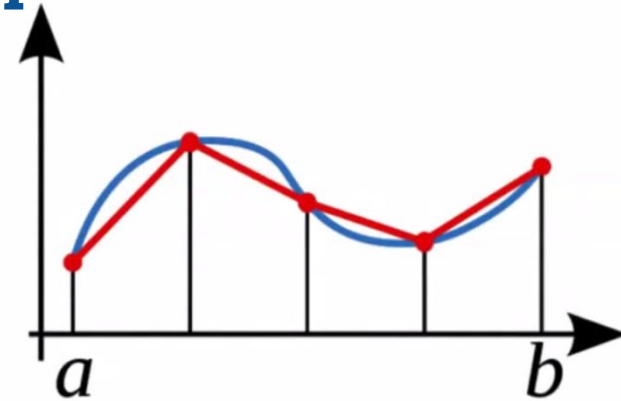
with $h = (b - a)/n$ is

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

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

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

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



I

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 $\int_a^b f(x)dx$

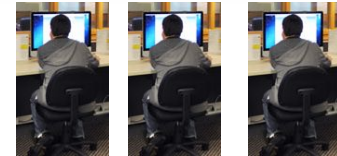
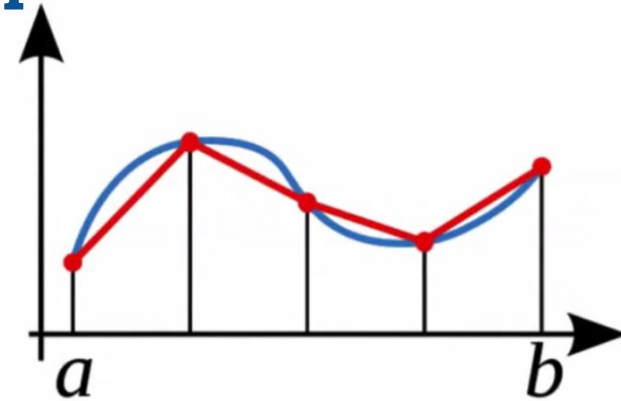
with $h = (b - a)/n$ is

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

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

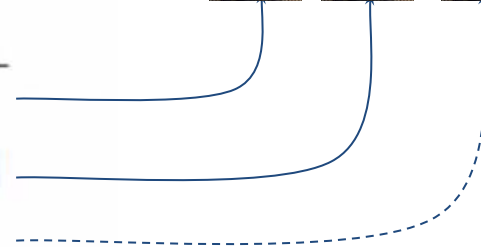
Given p processes, each process can work on n/p intervals

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



I

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 $\int_a^b f(x)dx$

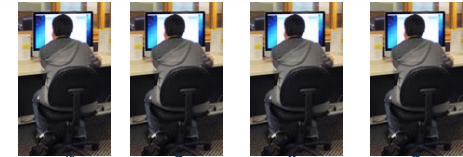
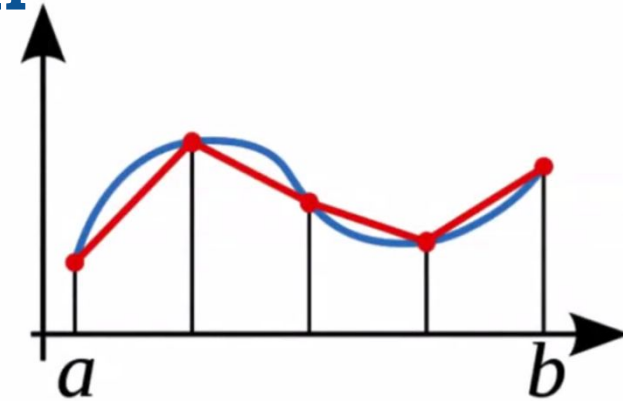
with $h = (b - a)/n$ is

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

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

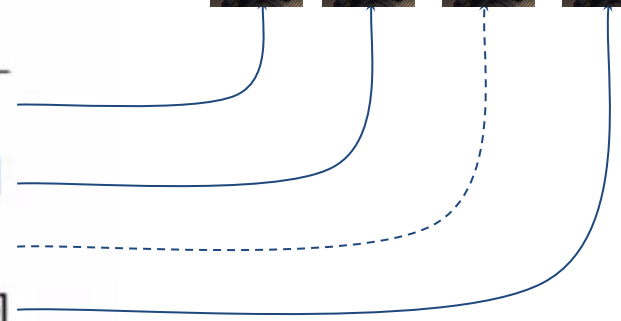
Given p processes, each process can work on n/p intervals

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



I

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>
```

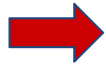
```
void Get_data(int p, int my_rank, double* a_p, double* b_p, int* n_p);
```

```
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);
```

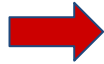
```
double Trap(double local_a, double local_b, int local_n, double h);
```

```
/* 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 */
```

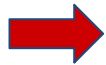
```
double f(double x);
```

```
/* 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 */
```

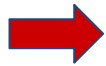
```
#include <stdio.h>
```

```
#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;
    int        p;
    double     a;
    double     b;
    int        n;
    double     h;
    double     local_a;
    double     local_b;
    int        local_n;
    double     my_area;
    double     total;
    int        source;
    int        dest = 0;
    int        tag = 0;
    MPI_Status status;
```

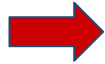
```
/* My process rank */
/* The number of processes */
/* Left endpoint */
/* Right endpoint */
/* Number of trapezoids */
/* Trapezoid base length */
/* Left endpoint my process */
/* Right endpoint my process */
/* Number of trapezoids for */
/* Integral over my interval */
/* Total area */
/* Process sending area */
/* All messages go to 0 */
```

```
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);

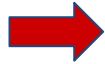
h = (b-a)/n; **/* h is the same for all processes */**

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;`

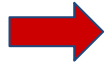
`my_area = Trap(local_a, local_b, local_n, h);`

→ `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 */
```

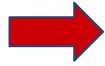
```
local_a = a + my_rank*local_n*h;
```

```
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);
```

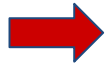


```
h = (b-a)/n;                                /* h is the same for all processes */  
local_n = n/p;                               /* So is the number of trapezoids */
```

```
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);
```



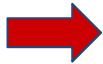
```
h = (b-a)/n;                                /* h is the same for all processes */  
local_n = n/p;                               /* So is the number of trapezoids */
```

```
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 */
```



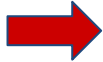
```
local_a = a + my_rank*local_n*h;  
local_b = local_a + local_n*h;  
my_area = Trap(local_a, local_b, local_n, h);
```

```
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);
```






```
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 */
```



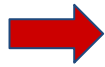
```
if (my_rank == 0)
{
    printf("With n = %d trapezoids, our estimate\n", n);
    printf("of the area from %f to %f = %.15f\n", a, b, total);
}

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();


Releases the MPI resources

```
return 0;
```

```
}
```

```
/* END of MAIN */
```

```
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 */
```

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 :

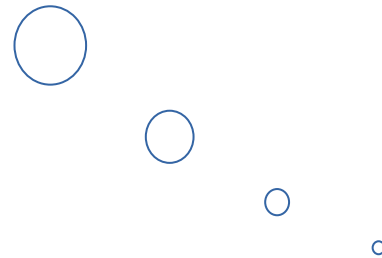
- **Serial and Parallel Approach -**
- **MPI_Comm_rank(...)**
- **MPI_Comm_size(...)**
- **Point to point communication**
- **MPI_Send(...)**
- **MPI_Recv(...)**
- **Blocking and Non-blocking Point to Point communication - Cases !**

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**

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/>



MPI_Comm_rank(.....)

MPI_Comm_rank(.....)

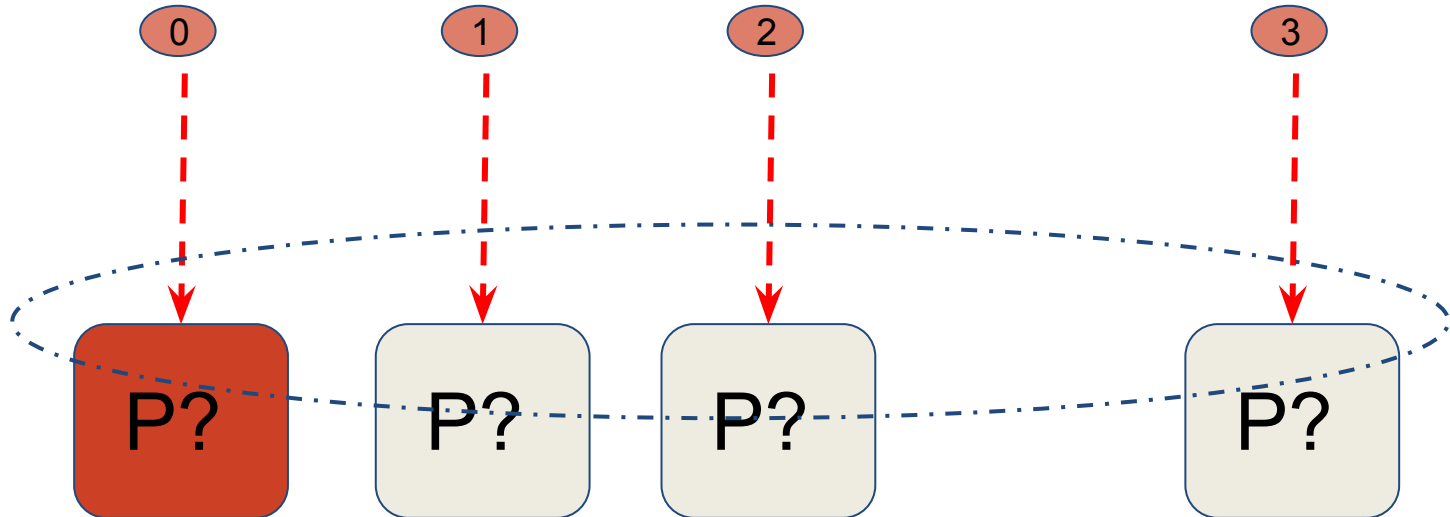
Syntax :

➔ `MPI_Comm_rank (MPI_Comm communicator , int * rank) ;`

MPI_Comm_rank(.....)

Syntax :

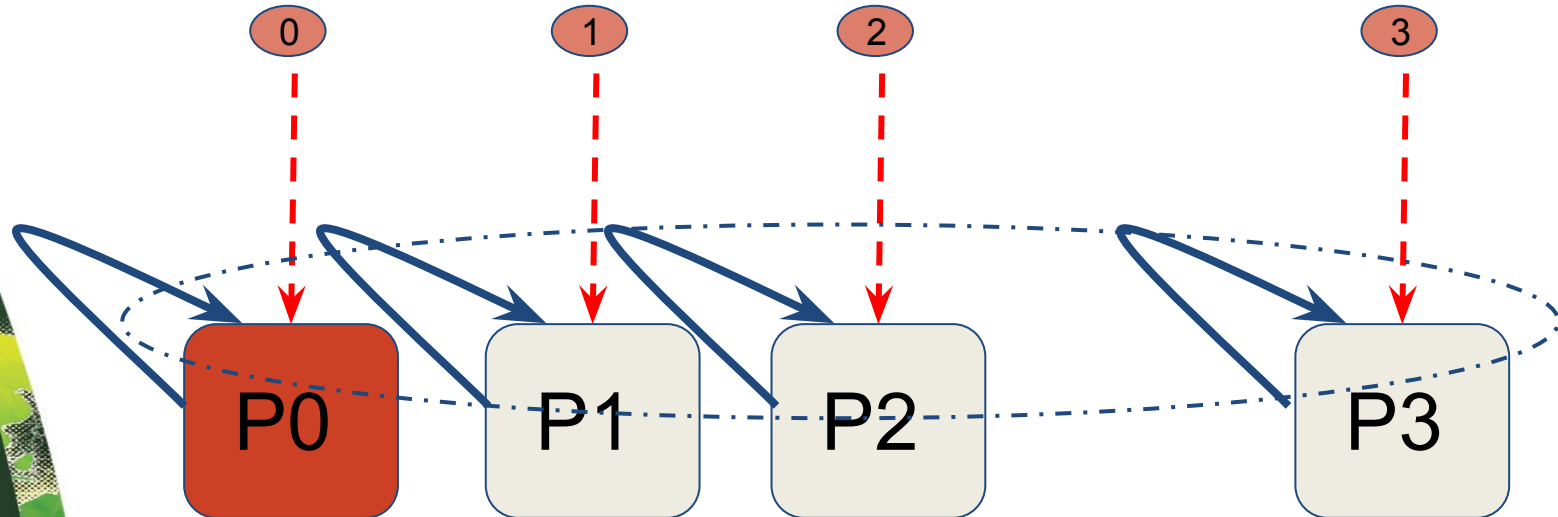
➔ `MPI_Comm_rank (MPI_Comm communicator , int * rank) ;`



MPI_Comm_rank(.....)

Syntax :

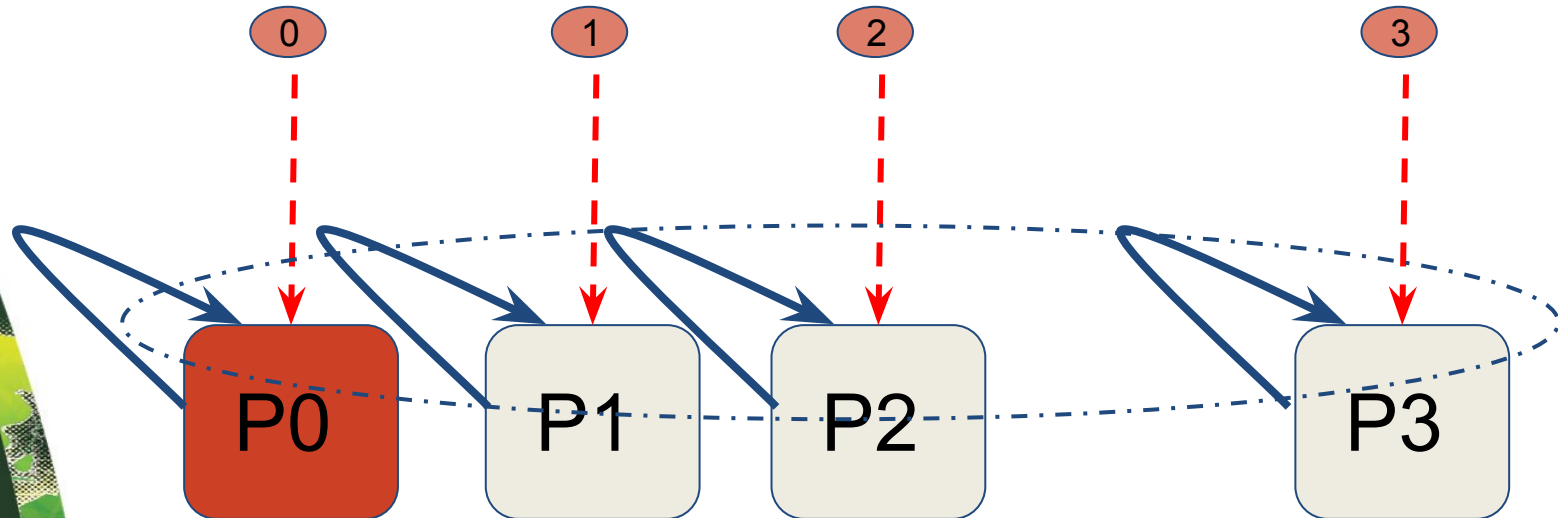
➔ `MPI_Comm_rank (MPI_Comm communicator , int * rank) ;`



MPI_Comm_rank(.....)

Syntax :

➔ `MPI_Comm_rank (MPI_Comm communicator , int * rank) ;`



MPI_Comm_size(.....)

MPI_Comm_size(.....)

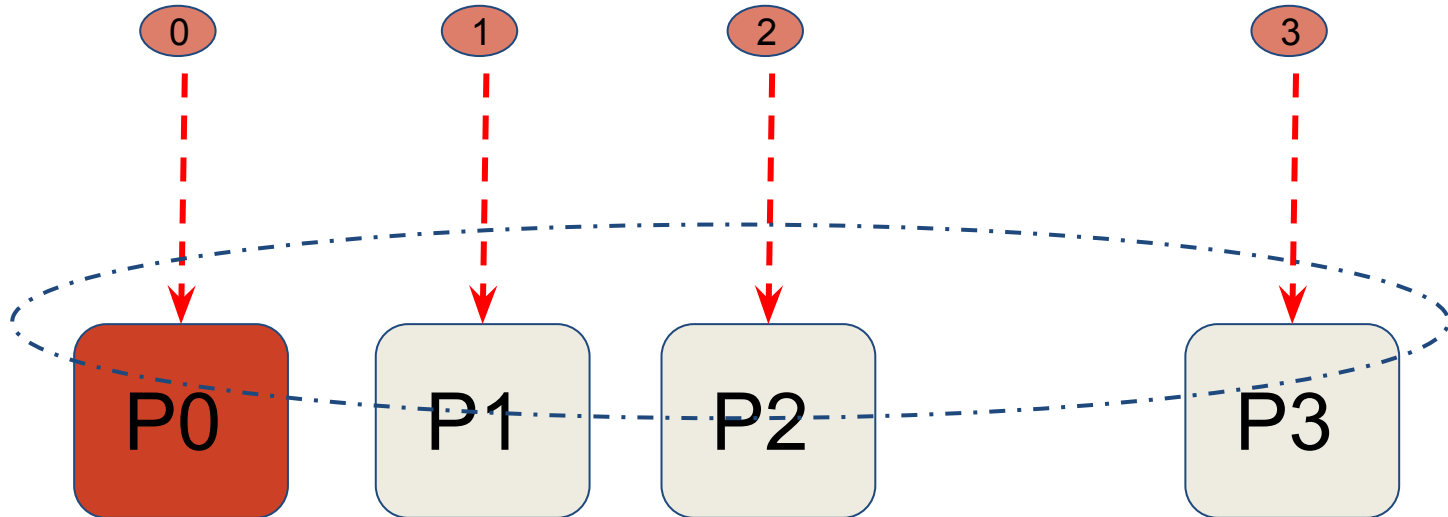
Syntax :

➔ `MPI_Comm_size (MPI_Comm communicator , int * size) ;`

MPI_Comm_size(.....)

Syntax :

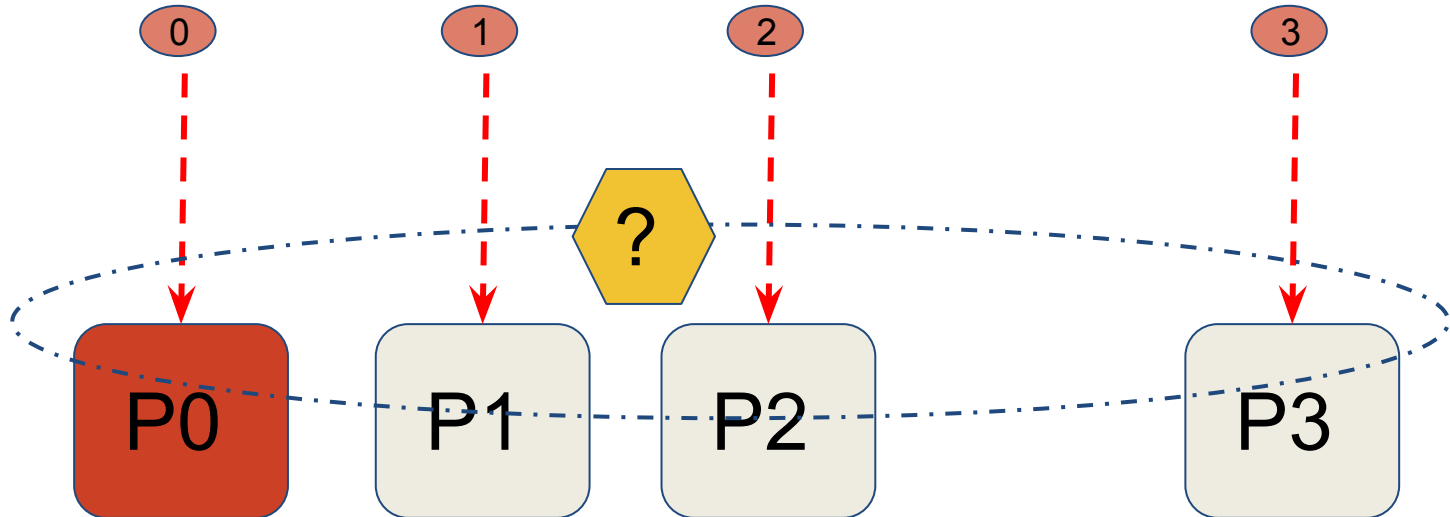
➔ `MPI_Comm_size (MPI_Comm communicator , int * size) ;`



MPI_Comm_size(.....)

Syntax :

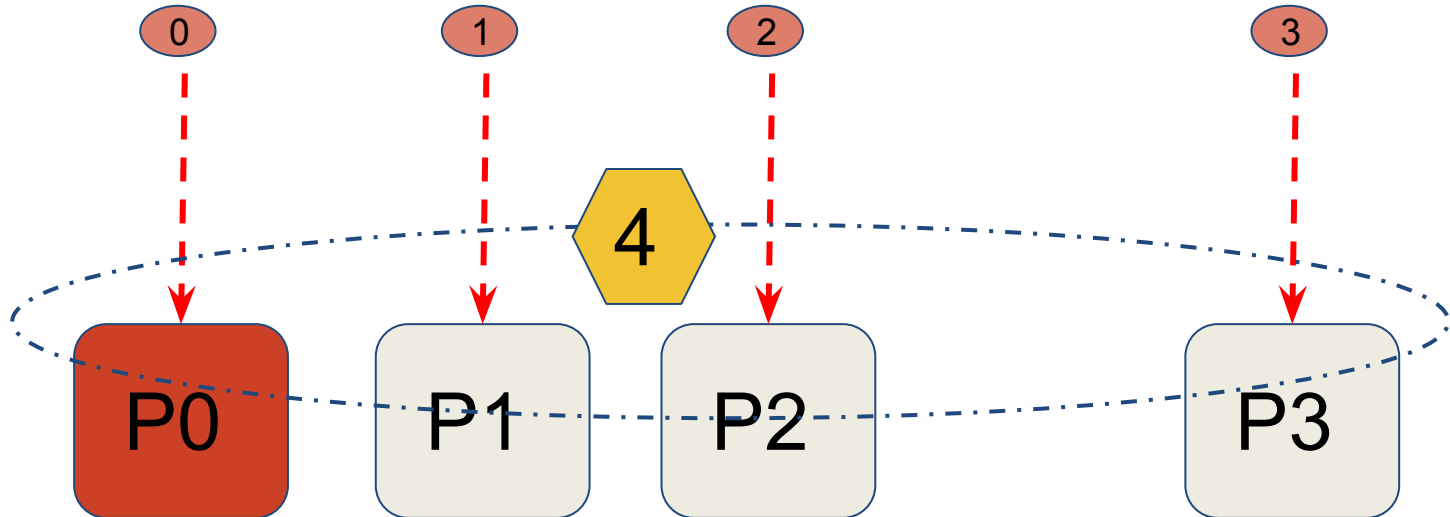
➔ `MPI_Comm_size (MPI_Comm communicator , int * size) ;`



MPI_Comm_size(.....)

Syntax :

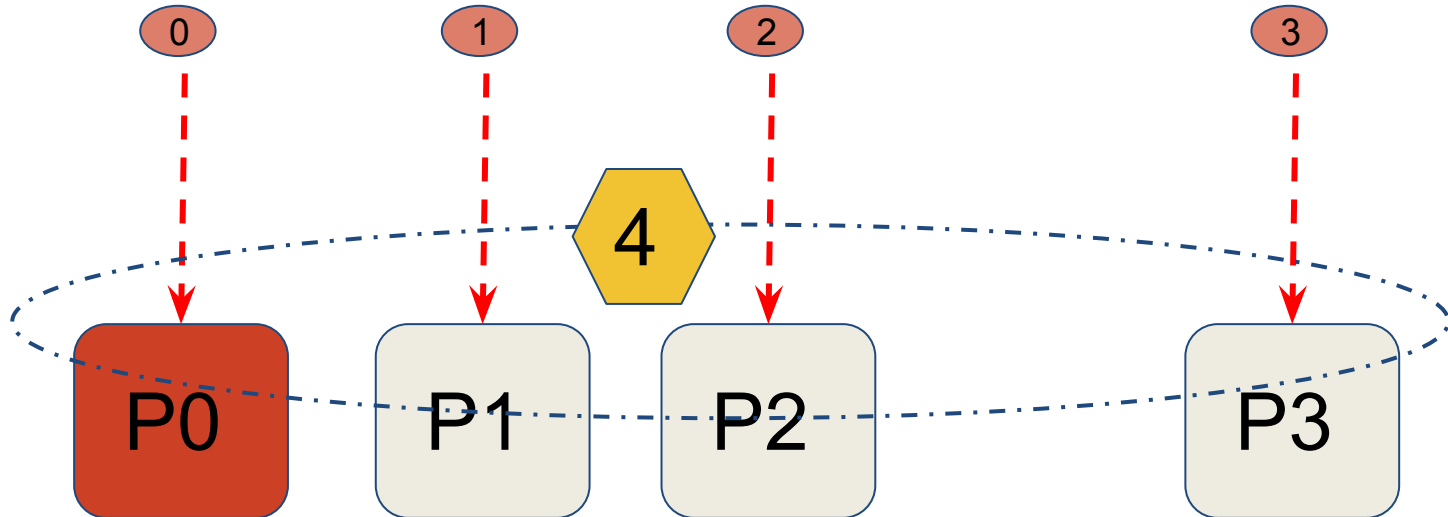
➔ `MPI_Comm_size (MPI_Comm communicator , int * size) ;`



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 :

→ MPI_Send (void* ¹msg_buffer , Int ²msg_size, MPI_Datatype ³msg_type,
Int ⁴destination, Int ⁵tag , MPI_Comm ⁶communicator) ;

MPI_Send(.....)

Syntax :

→ MPI_Send (void* ¹msg_buffer , Int ²msg_size, MPI_Datatype ³msg_type,
Int ⁴destination, Int ⁵tag , MPI_Comm ⁶communicator) ;

¹ Address of Message buffer

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹msg_buffer , **Int** ²msg_size, **MPI_Datatype** ³msg_type,
Int ⁴destination, **Int** ⁵tag , **MPI_Comm** ⁶communicator) ;

¹ Address of Message buffer

² Message size

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹msg_buffer , **Int** ²msg_size, **MPI_Datatype** ³msg_type,
Int ⁴destination, **Int** ⁵tag , **MPI_Comm** ⁶communicator) ;

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

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹ **msg_buffer** , **Int** ² **msg_size**, **MPI_Datatype** ³ **msg_type**,
Int ⁴ **destination**, **Int** ⁵ **tag** , **MPI_Comm** ⁶ **communicator**) ;

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

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹msg_buffer , **Int** ²msg_size, **MPI_Datatype** ³msg_type,
Int ⁴destination, **Int** ⁵tag , **MPI_Comm** ⁶communicator) ;

¹ Address of Message buffer

² Message size

³ Data Type

⁴ Destination process rank

⁵ Tag - Message Identifier/..

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹msg_buffer , **Int** ²msg_size, **MPI_Datatype** ³msg_type,
Int ⁴destination, **Int** ⁵tag , **MPI_Comm** ⁶communicator) ;

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

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹ **msg_buffer** , **Int** ² **msg_size** , **MPI_Datatype** ³ **msg_type** ,
Int ⁴ **destination** , **Int** ⁵ **tag** , **MPI_Comm** ⁶ **communicator**) ;

MPI_Send(.....)

Syntax :

→ MPI_Send (void* ¹msg_buffer , Int ²msg_size, MPI_Datatype ³msg_type,
Int ⁴destination, Int ⁵tag , MPI_Comm ⁶communicator) ;

¹ ² ³

MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹msg_buffer , **Int** ²msg_size, **MPI_Datatype** ³msg_type,
Int ⁴destination, **Int** ⁵tag , **MPI_Comm** ⁶communicator) ;

Tell us information about message



¹ ² ³

MPI_Send(.....)

Syntax :

→ MPI_Send (¹void* ²msg_buffer , ²Int ²msg_size, ³MPI_Datatype ³msg_type, ⁴Int ⁴destination, ⁵Int ⁵tag , ⁶MPI_Comm ⁶communicator) ;

Tell us information about message



MPI_Send(.....)

Syntax :

➔ **MPI_Send** (**void*** ¹msg_buffer , **Int** ²msg_size, **MPI_Datatype** ³msg_type,
Int ⁴destination, **Int** ⁵tag , **MPI_Comm** ⁶communicator) ;

Tell us information about message



¹ ² ³

Tell us, where and How to send a message



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 :

→ MPI_Recv (void* ¹msg_buffer , Int ²buf_size, MPI_Datatype ³buf_type,
Int ⁴source, Int ⁵tag , MPI_Comm ⁶communicator, MPI_Status* ⁷);

MPI_Recv(.....)

Syntax :

→ MPI_Recv (void* ¹msg_buffer , Int ²buf_size, MPI_Datatype ³buf_type,
Int ⁴source, Int ⁵tag , MPI_Comm ⁶communicator, MPI_Status* ⁷);

¹ Address of Message buffer

² Buffer size

³ Data Type

MPI_Recv(.....)

Syntax :

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

1 2 3 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 :

→ MPI_Recv (void* ¹msg_buffer , Int ²buf_size, MPI_Datatype ³buf_type,
Int ⁴source, Int ⁵tag , MPI_Comm ⁶communicator, MPI_Status* ⁷);

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

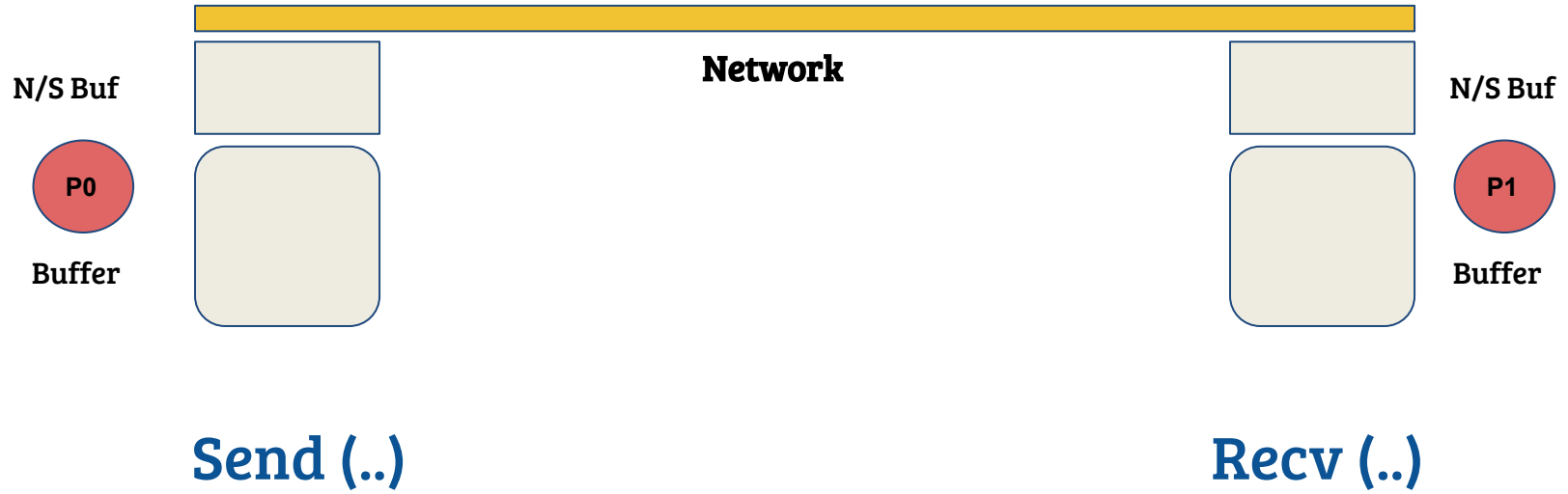
➔ **MPI_Recv** (**void*** msg_buffer , **Int** buf_size, **MPI_Datatype** buf_type, **Int** source, **Int** tag , **MPI_Comm** communicator, **MPI_Status***);

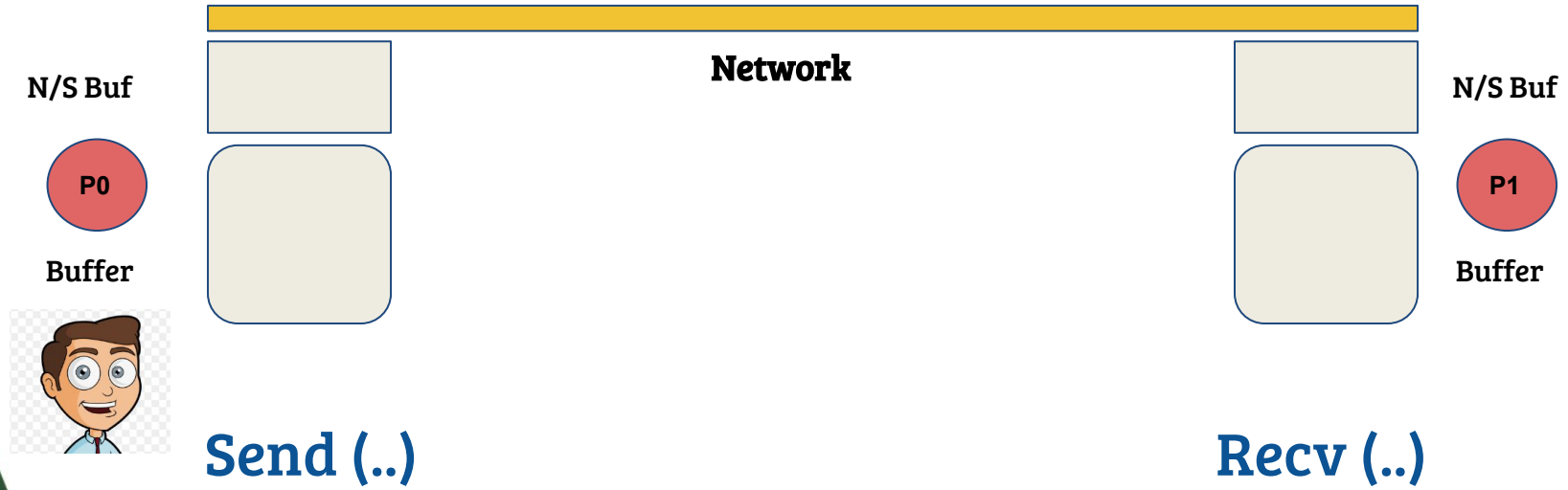
```
recv_comm = send_comm  
recv_tag  = send_tag  
dest = Destination process rank  
Src = Source process rank
```

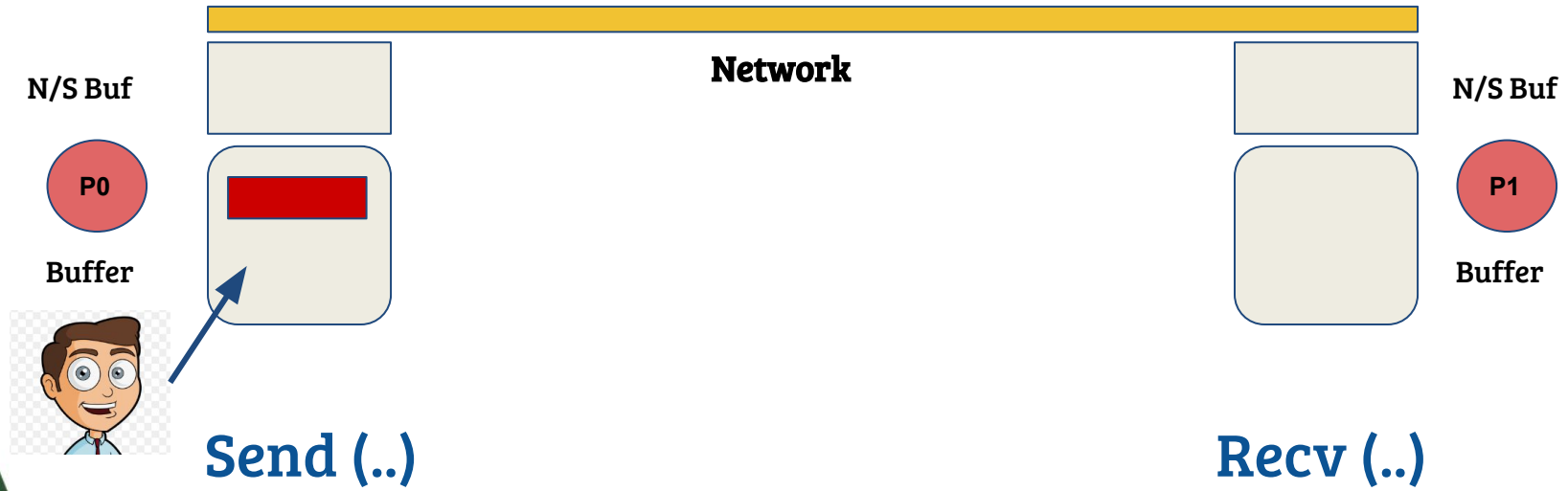


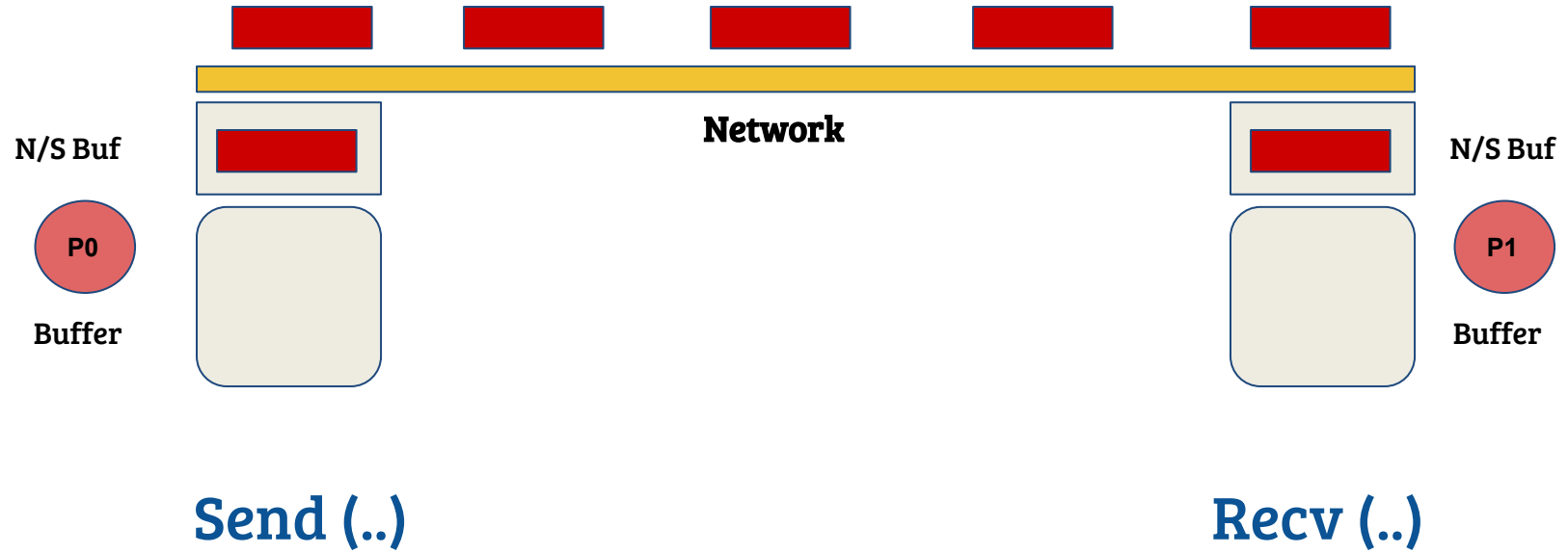
How message is transferred ... ?

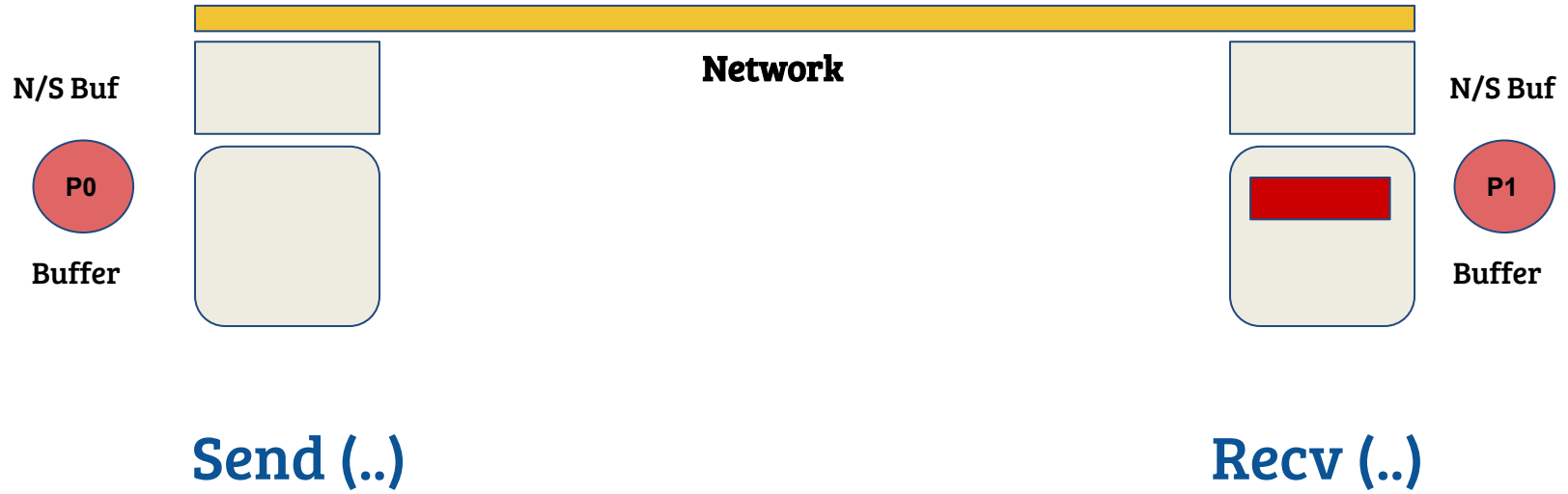
... Different cases !

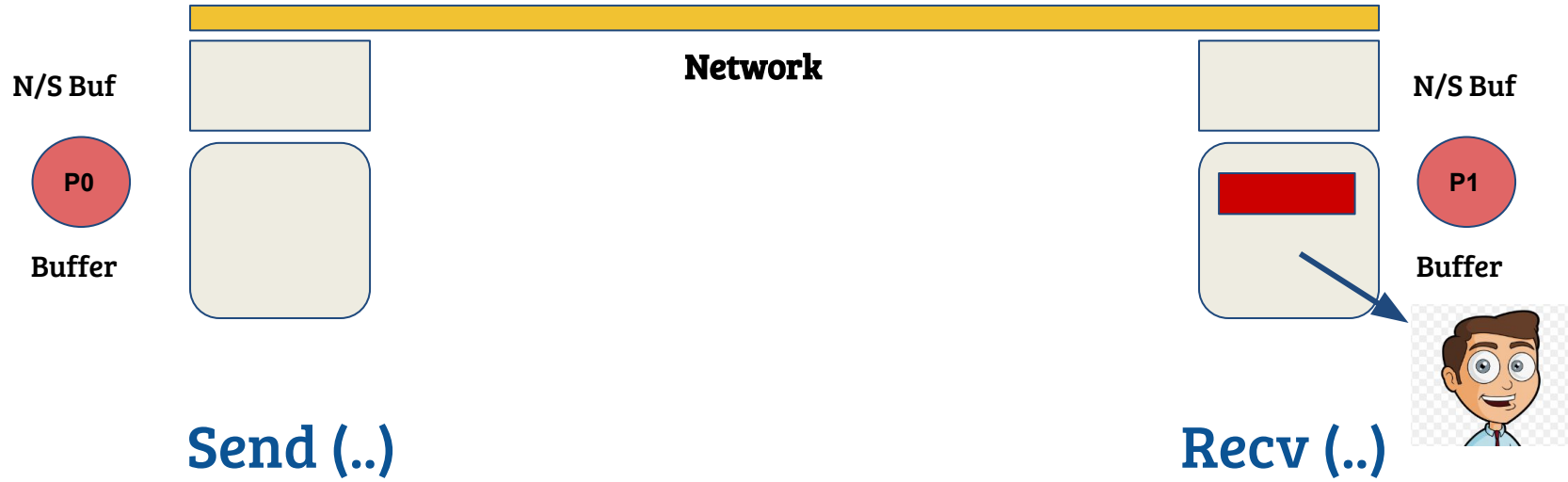


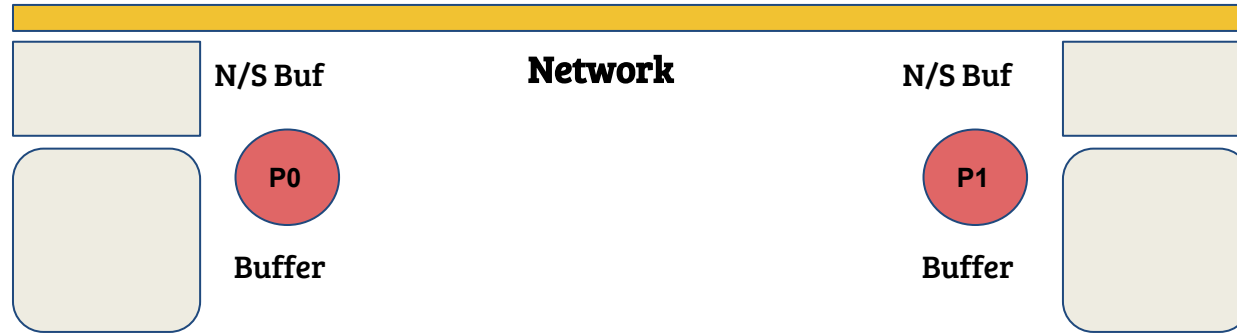




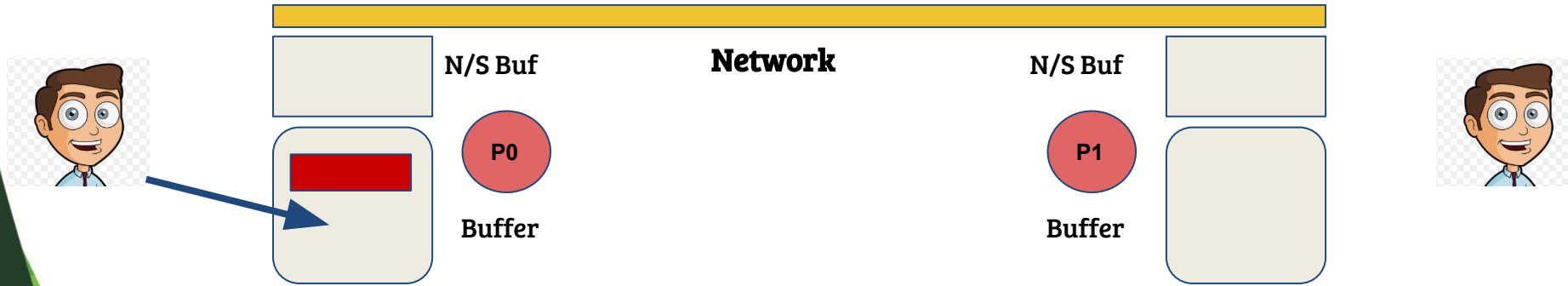




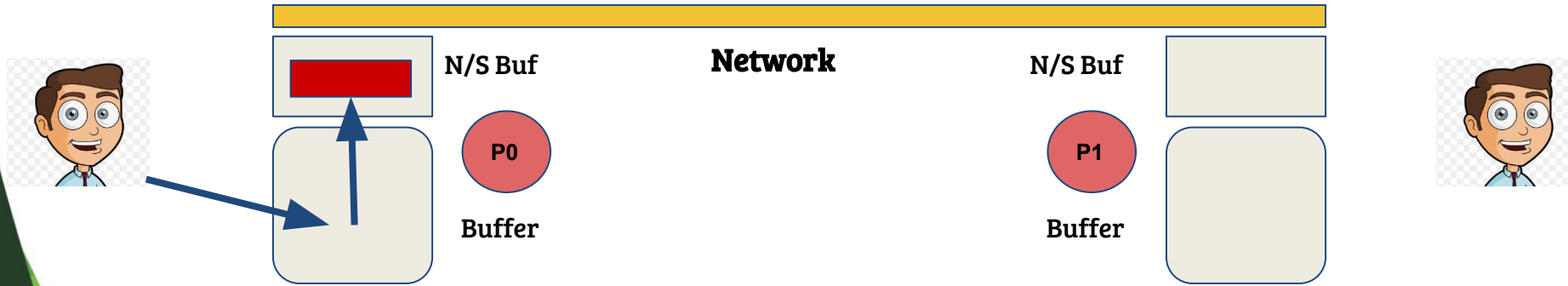




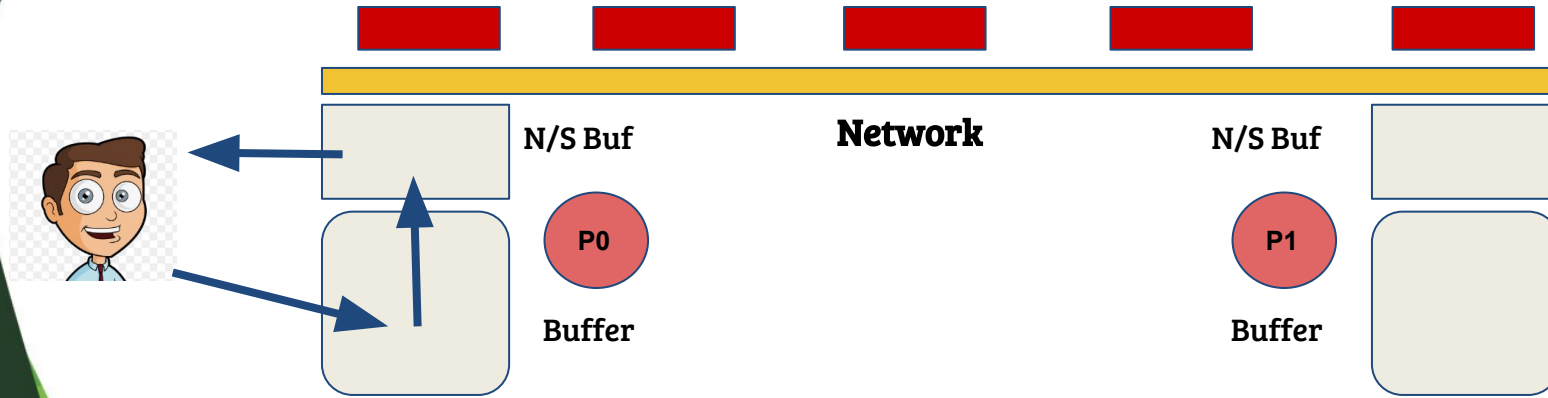
Case 1 : Blocking - Point to Point Communication



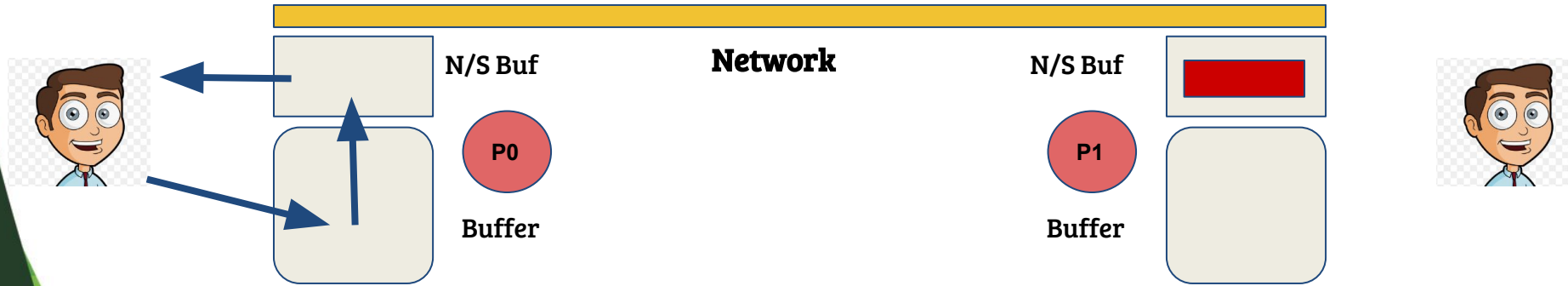
Case 1 : Blocking - Point to Point Communication



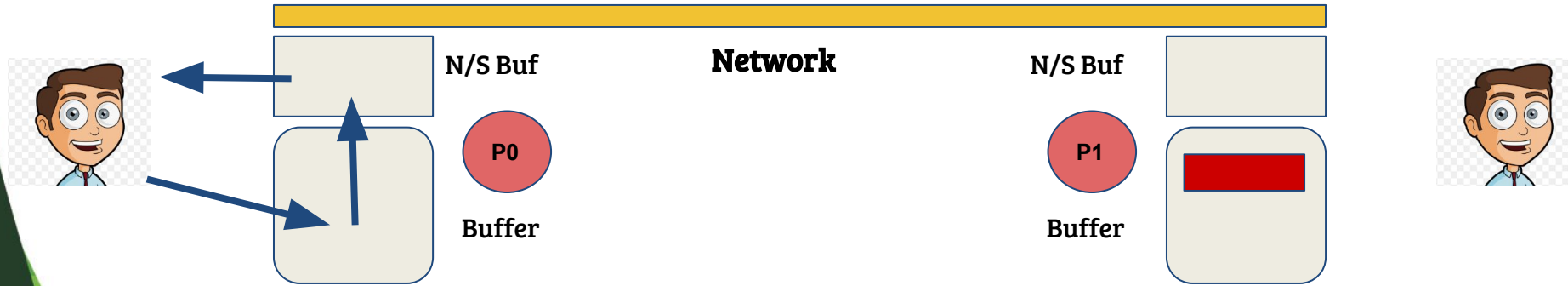
Case 1 : Blocking - Point to Point Communication



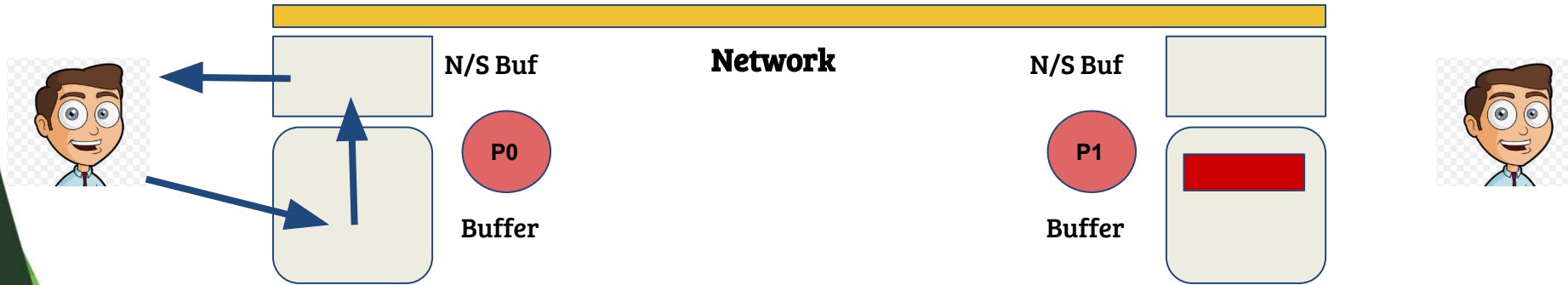
Case 1 : Blocking - Point to Point Communication



Case 1 : Blocking - Point to Point Communication

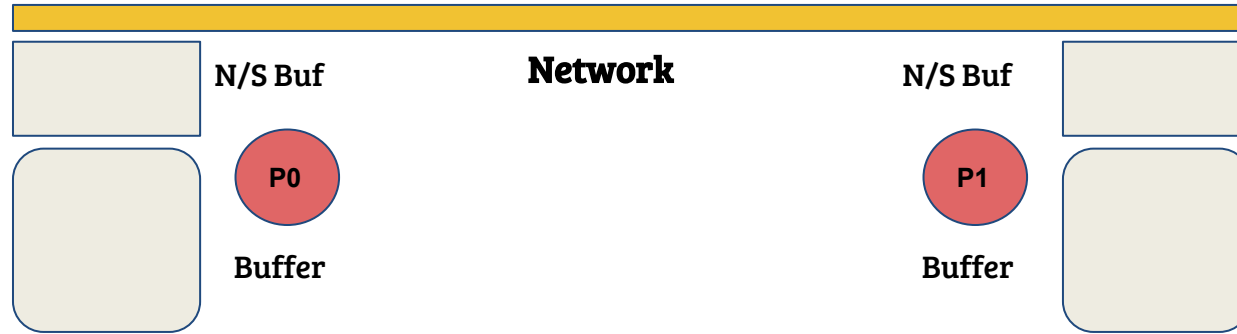


Case 1 : Blocking - Point to Point Communication

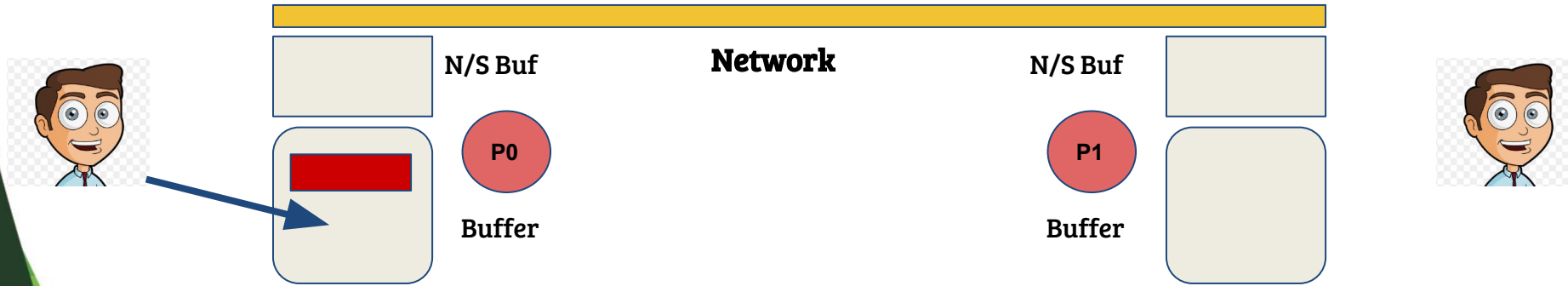


Case 1 : Blocking - Point to Point Communication

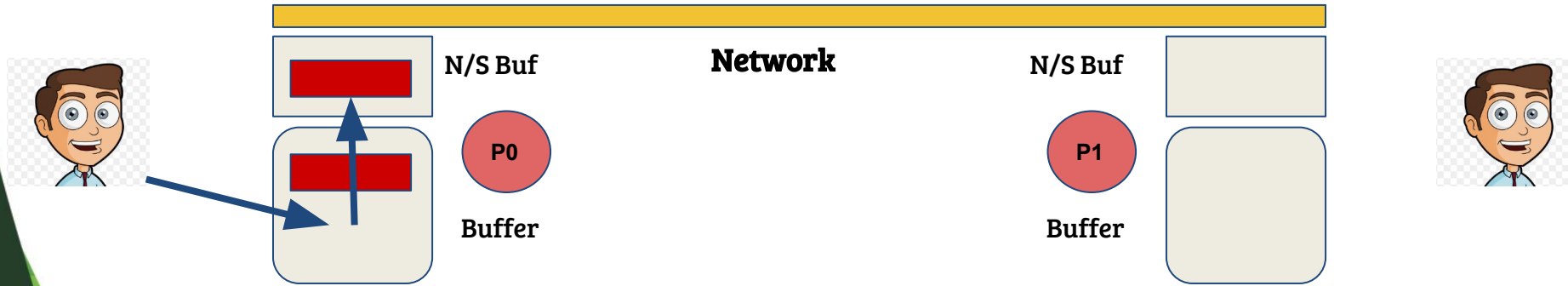
- ➡ **MPI_Send** (void* msg_buffer , Int msg_size, MPI_Datatype msg_type, Int destination, Int tag , MPI_Comm communicator) ;
- ➡ **MPI_Recv** (void* msg_buffer , Int buf_size, MPI_Datatype buf_type, Int source, Int tag , MPI_Comm communicator, MPI_Status*);



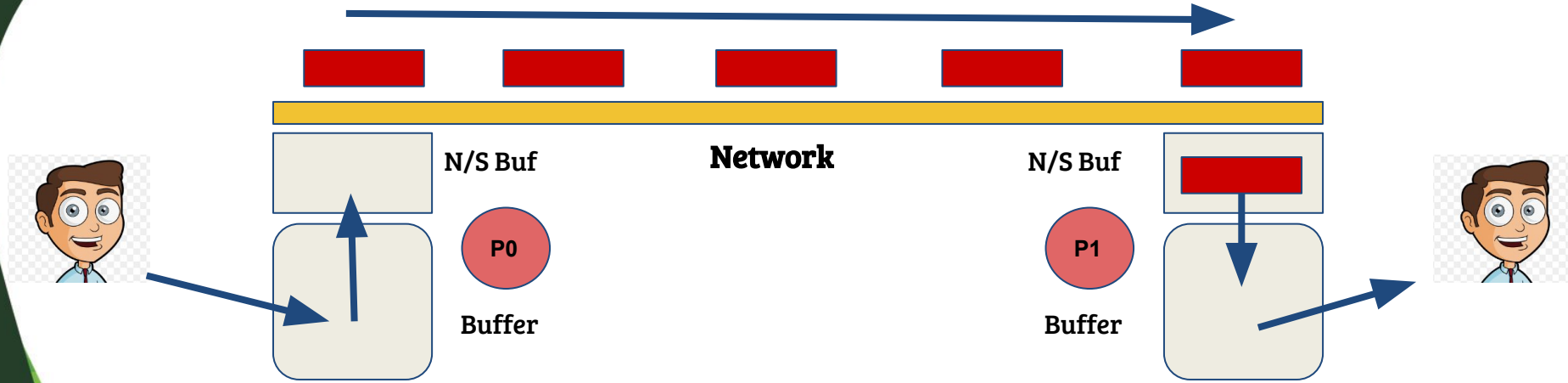
Case 2 : Synchronous Blocking - Point to Point Communication



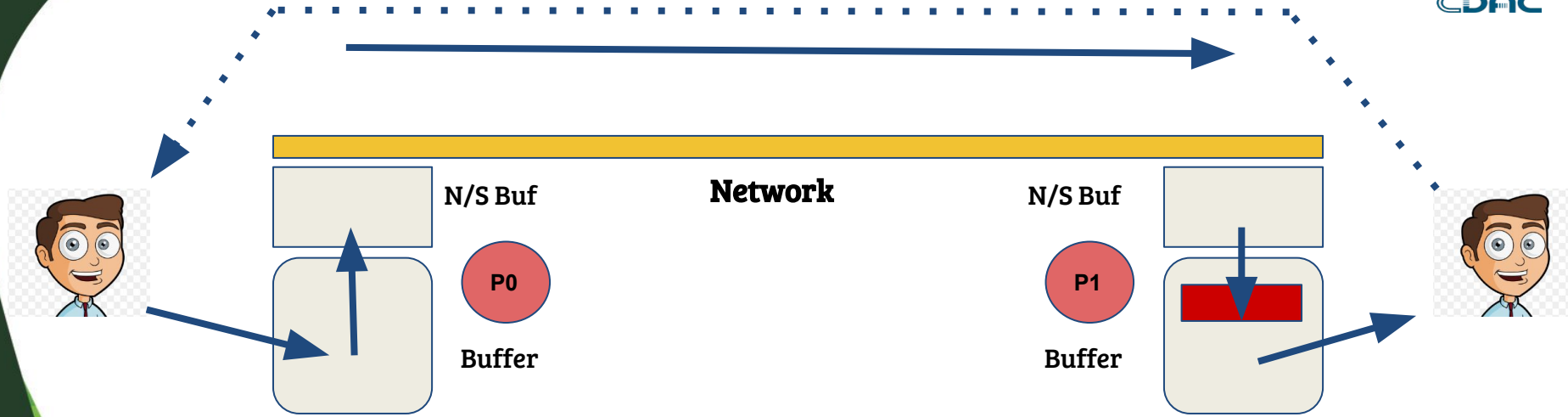
Case 2 : Synchronous Blocking - Point to Point Communication



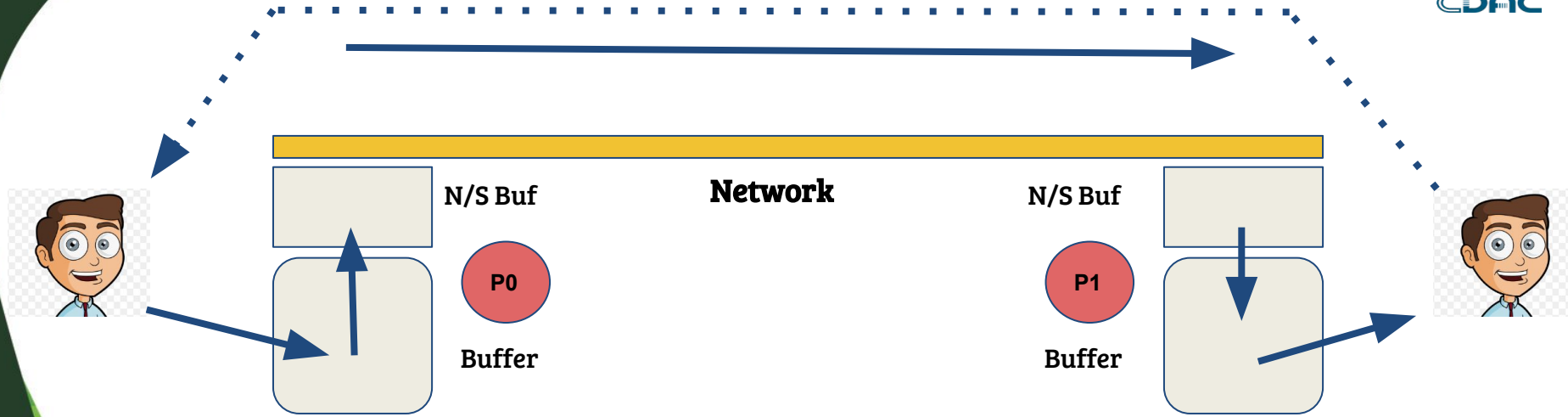
Case 2 : Synchronous Blocking - Point to Point Communication



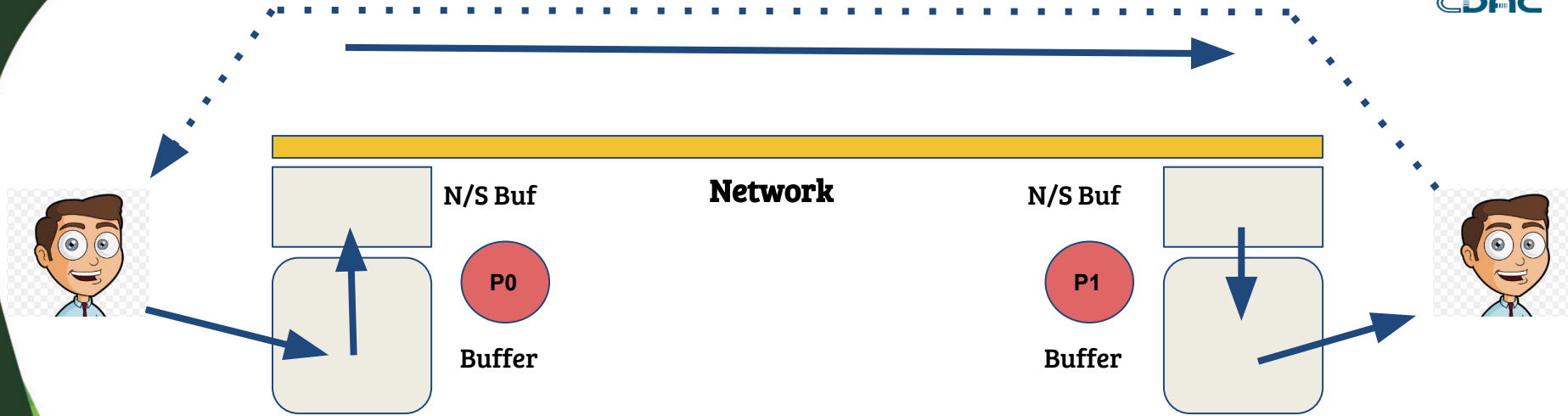
Case 2 : Synchronous Blocking - Point to Point Communication



Case 2 : Synchronous Blocking - Point to Point Communication

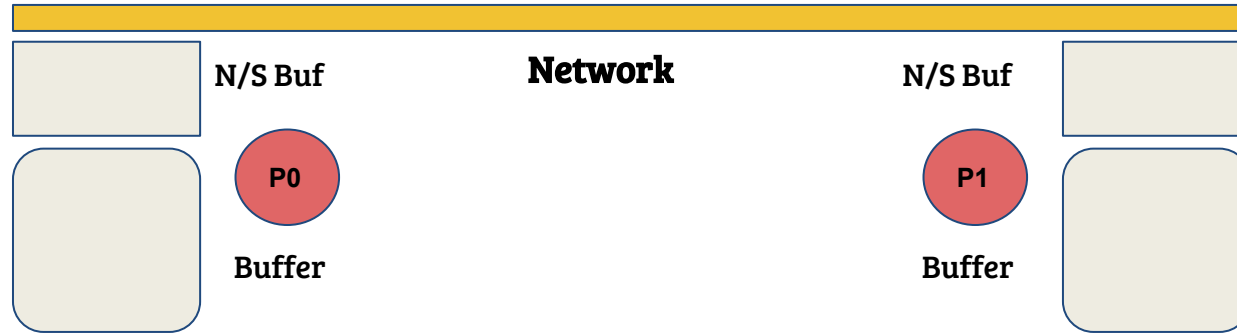


Case 2 : Synchronous Blocking - Point to Point Communication

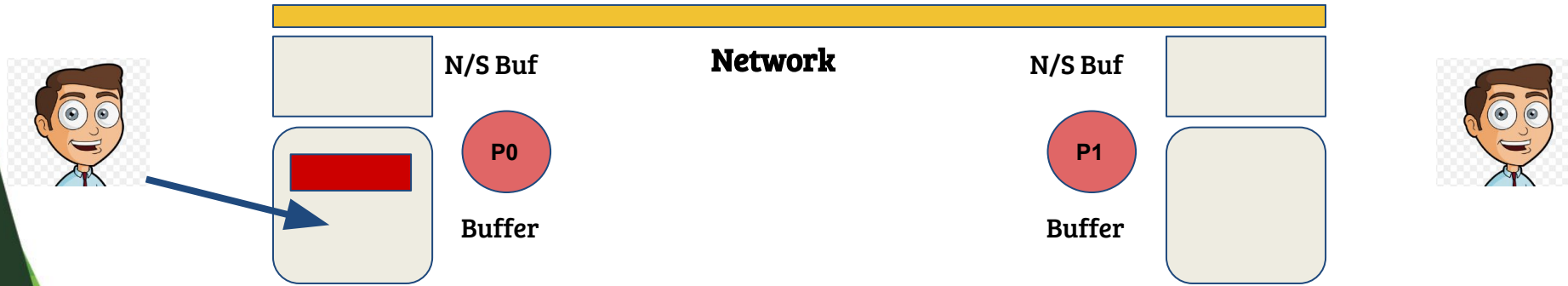


Case 2 : Synchronous Blocking - Point to Point Communication

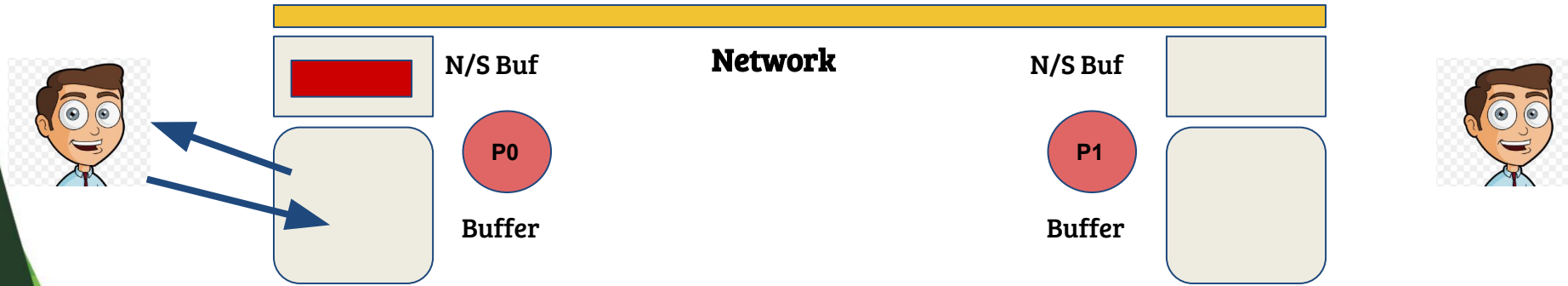
➡ **MPI_Ssend** (void* msg_buffer , Int msg_size, MPI_Datatype msg_type,
Int destination, Int tag , MPI_Comm communicator) ;



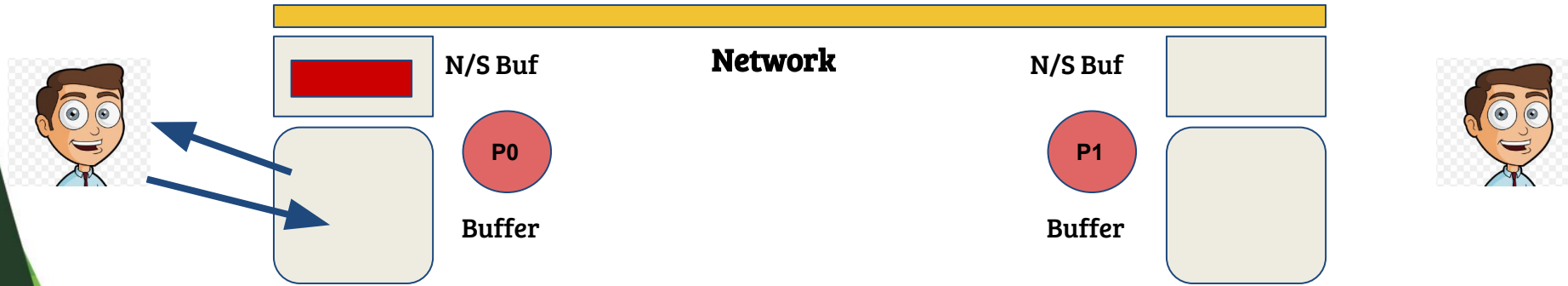
★ Case 3 : Non Blocking - Point to Point Communication



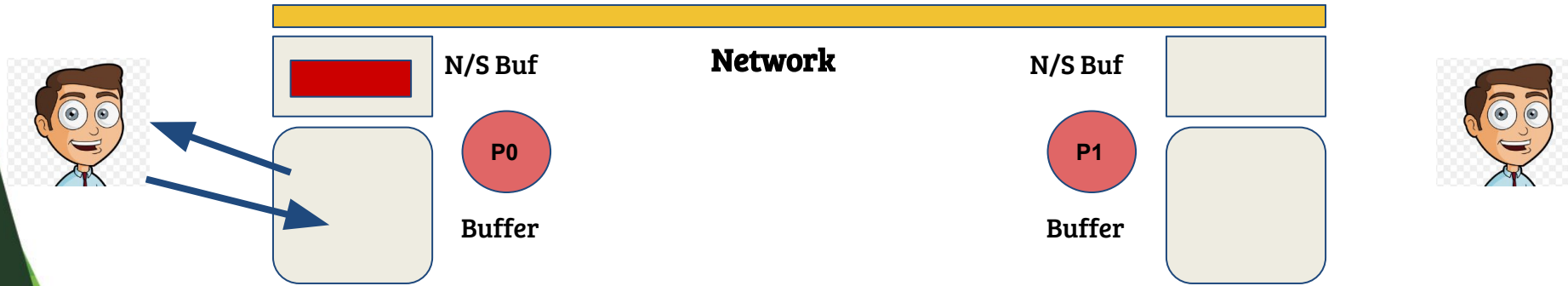
★ Case 3 : Non Blocking - Point to Point Communication



★ Case 3 : Non Blocking - Point to Point Communication

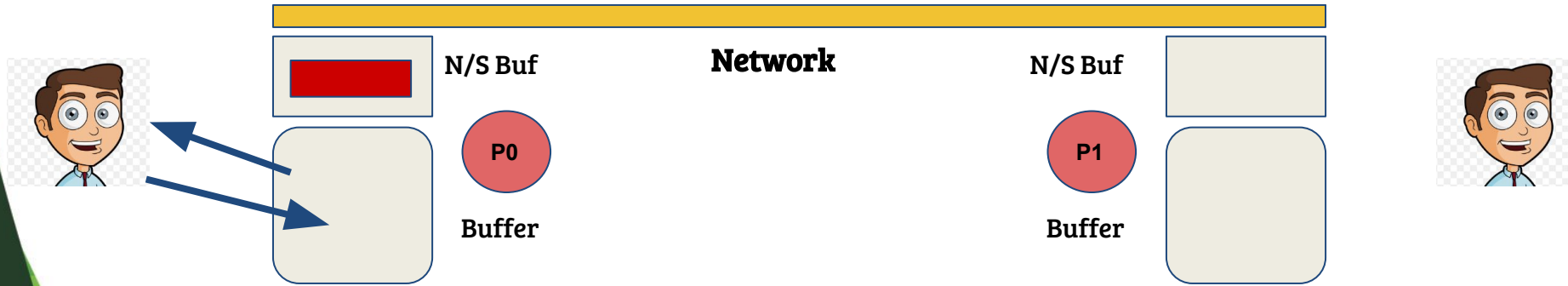


★ Case 3 : Non Blocking - Point to Point Communication

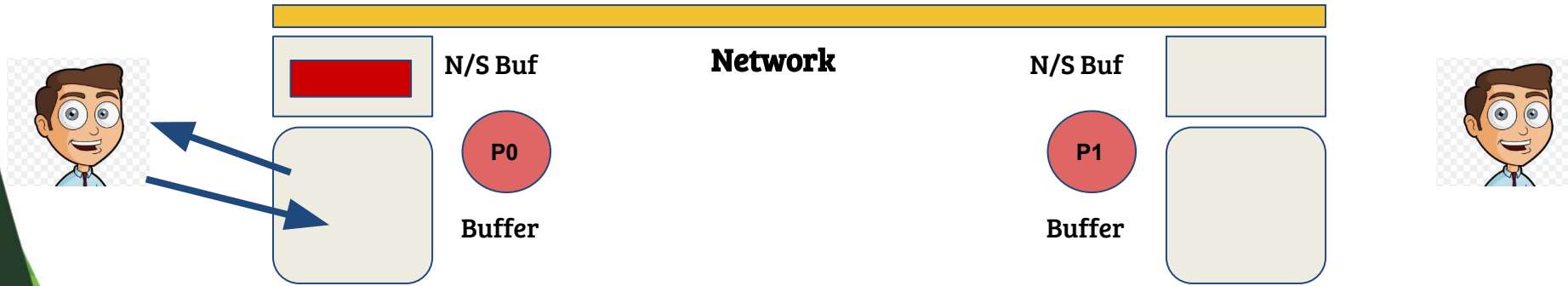


★ Case 3 : Non Blocking - Point to Point Communication

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

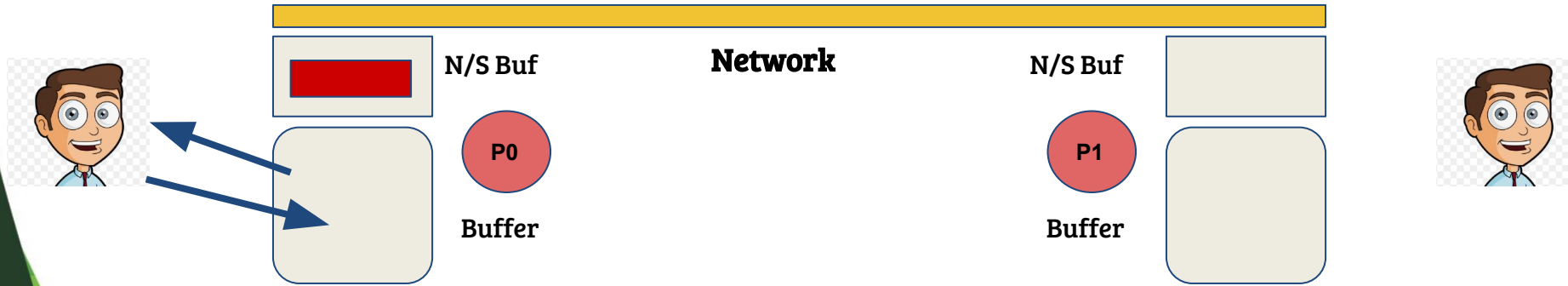


★ Case 3 : Non Blocking - Point to Point Communication



★ Case 3 : Non Blocking - Point to Point Communication

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

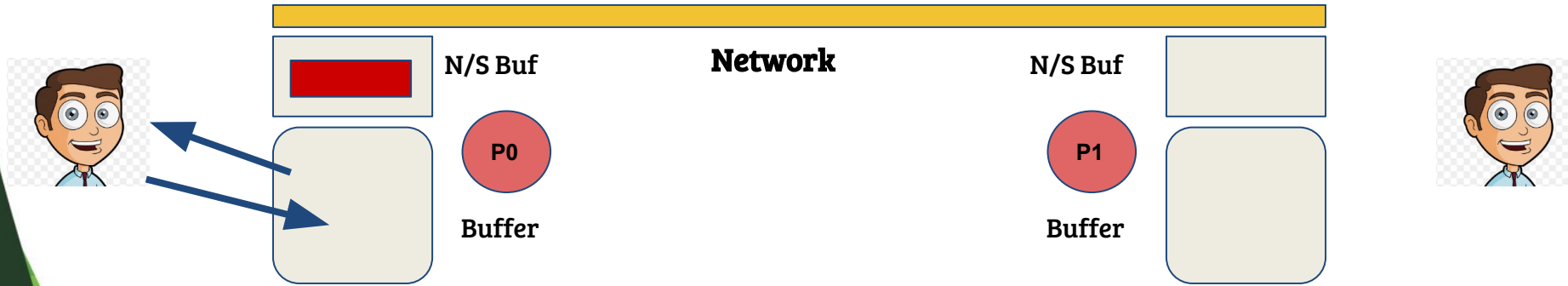


★ Case 3 : Non Blocking - Point to Point Communication

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

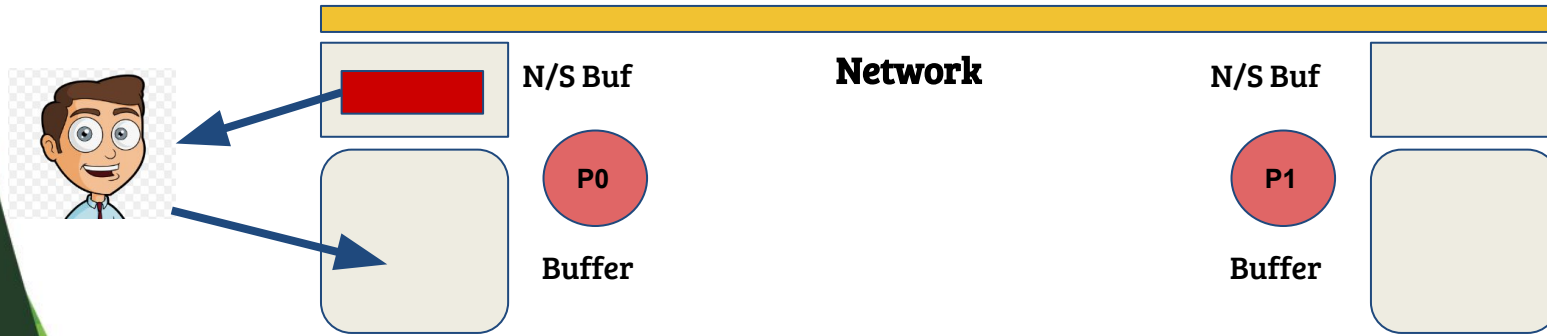


`MPI_Test(MPI_Request *req, int *flag, MPI_Status *status) ;`



★ Case 3 : Non Blocking - Point to Point Communication

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

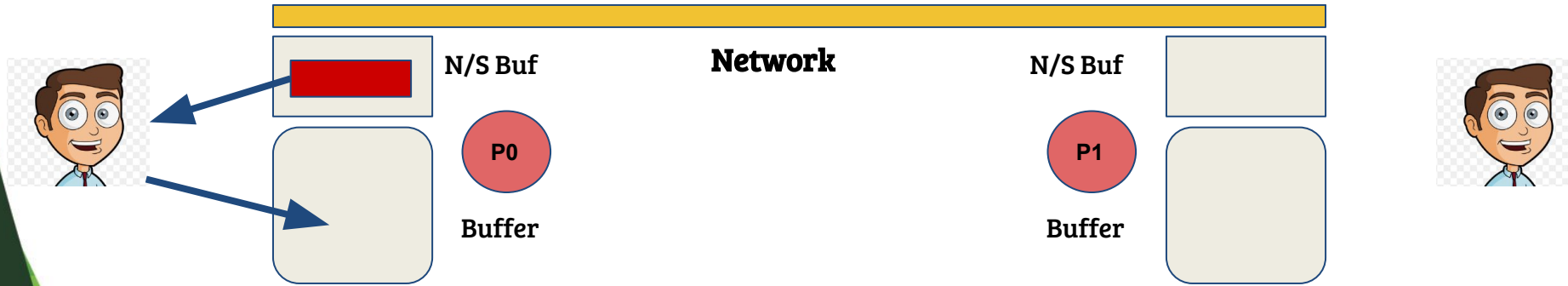


★ Case 3 : Non Blocking - Point to Point Communication

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



`MPI_Wait(MPI_Request *request, MPI_Status *status)`



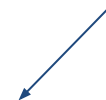
★ Case 3 : Non Blocking - Point to Point Communication



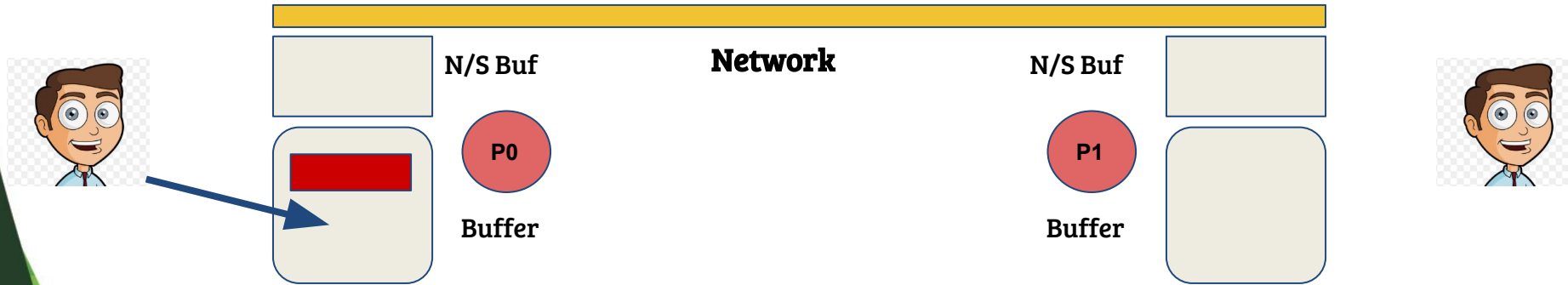
Data may be in buffer till next message arrive ..!



MPI_Wait(MPI_Request *request, MPI_Status *status)

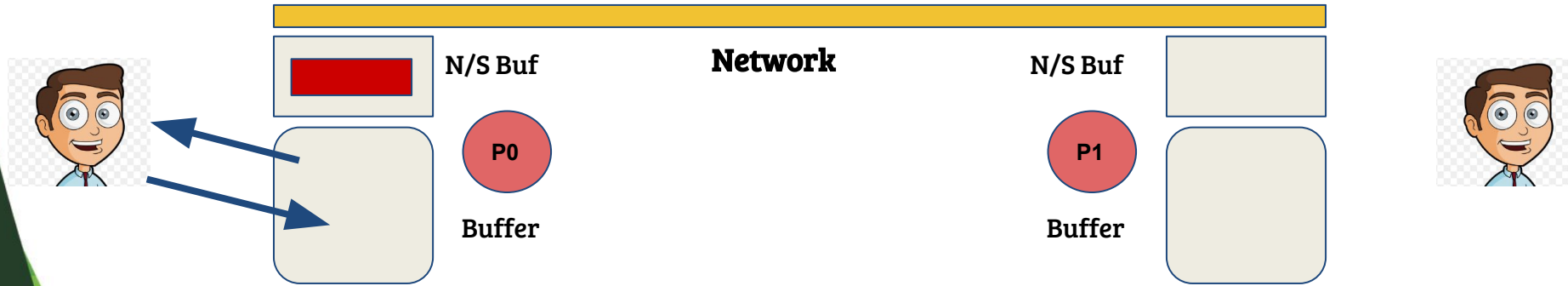


Block and Wait till
operation get
finish..



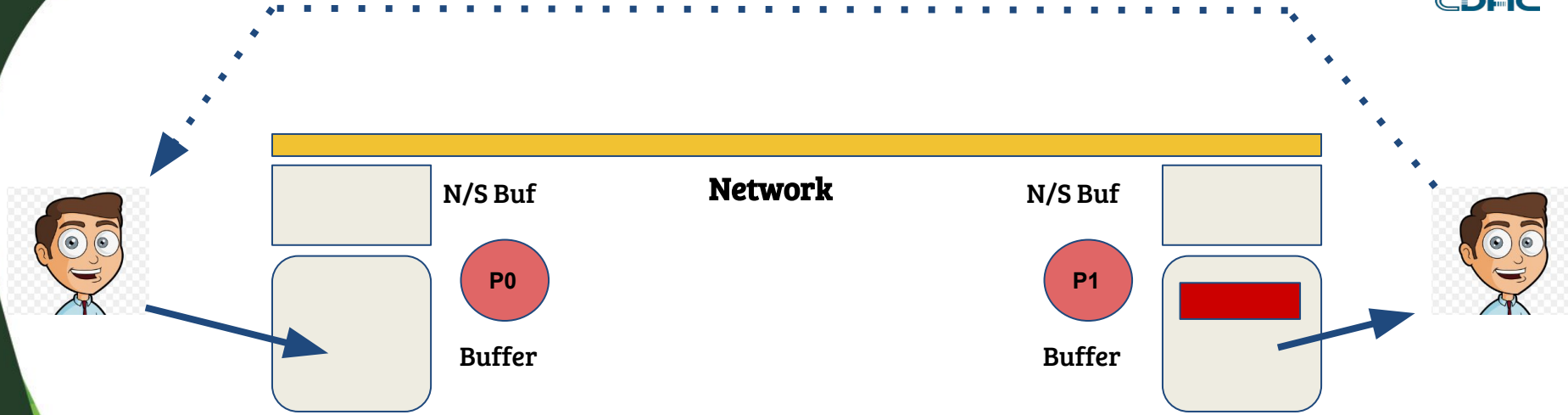
Case 4 : Synchronous Non Blocking - Point to Point Communication

➡ `MPI_`**l**`send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type, Int destination, Int tag , MPI_Comm communicator, req *) ;`



Case 4 : Synchronous Non Blocking - Point to Point Communication

➔ `MPI_`**l**`send (void* msg_buffer , Int msg_size, MPI_Datatype msg_type, Int destination, Int tag , MPI_Comm communicator, req *) ;`



Case 4 : Synchronous Non Blocking - Point to Point Communication

➡ Similar to Non-Blocking **except** `MPI_Wait(...)` and `MPI_Test(...)`
Completes their operation when **destination process receive the message**

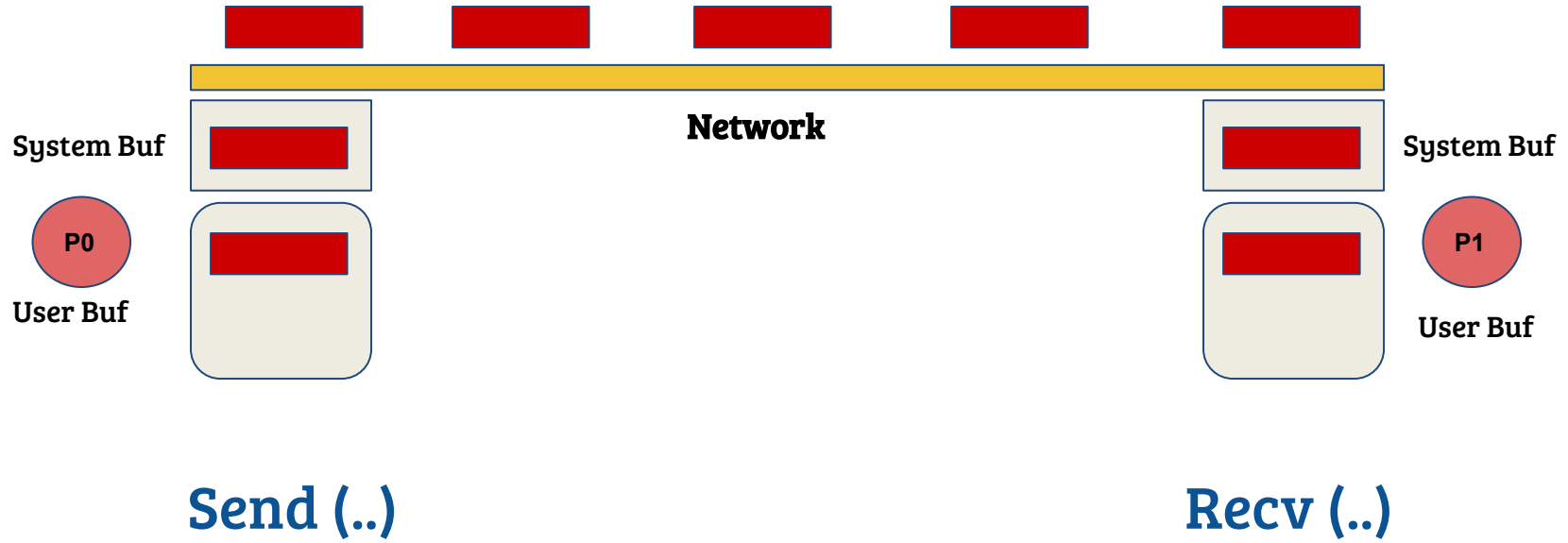
❖ Got it ?

❖ Got it ?



❖ Got it ?





❏ Point to Point Communication Routines

Non - Blocking point to point communication :

Asynchronous communication

- MPI_Isend (&buf,count,datatype,dest,tag,comm,&request)
- MPI_Irecv (&buf,count,datatype,source,tag,comm,&request)
- MPI_Issend (&buf,count,datatype,dest,tag,comm,&request)
 - Synchronous non-blocking send.
- Check for Asynchronous Transfer :
 - MPI_Test(MPI_Request *request, int *flag, MPI_Status * status)
 - **Flag:**
 - if flag == 0, the send/receive operation is **not yet complete**
 - if flag != 0, the send/receive operation is **complete** and the variable **status** contains information about the message
 - **status:** contains information about the message (use the information **only** if flag != 0)

Collective Communication Routines

- Used for performing operation on all processes simultaneously
- Approx -- 16

⇒ 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
:
:
:
:

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

⇒ MPI_Allreduce (

void*	input_data ,
void*	output_data ,
Int	count ,
MPI_Datatype	datatype ,
MPI_Op	operator ,
MPI_Comm	comm

);

Ex : MPI_Allreduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

Collective Communication Routines

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

Eg : MPI_Bcast(a, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD) ;

⇒ 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
);

Eg : MPI_Scatter (a, local_n, MPI_DOUBLE, local_a, local_n, MPI_DOUBLE, 0, comm) ;

❏ Collective Communication Routines

⇒ 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

);

Eg : MPI_Gather (local_b, local_n, MPI_DOUBLE, b, local_n, MPI_DOUBLE, 0, comm) ;

⇒ 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

);

Eg : MPI_Allgather (local_b, local_n, MPI_DOUBLE, b, local_n, MPI_DOUBLE, comm) ;