

CS 4504

Parallel and Distributed Computation

Message Passing Interface (MPI)

Kun Suo

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

Review

- MPI introduction
 - Helloworld of MPI
- Performance evaluation
- Example: how to solve problems in MPI
 - Trapezoidal problem

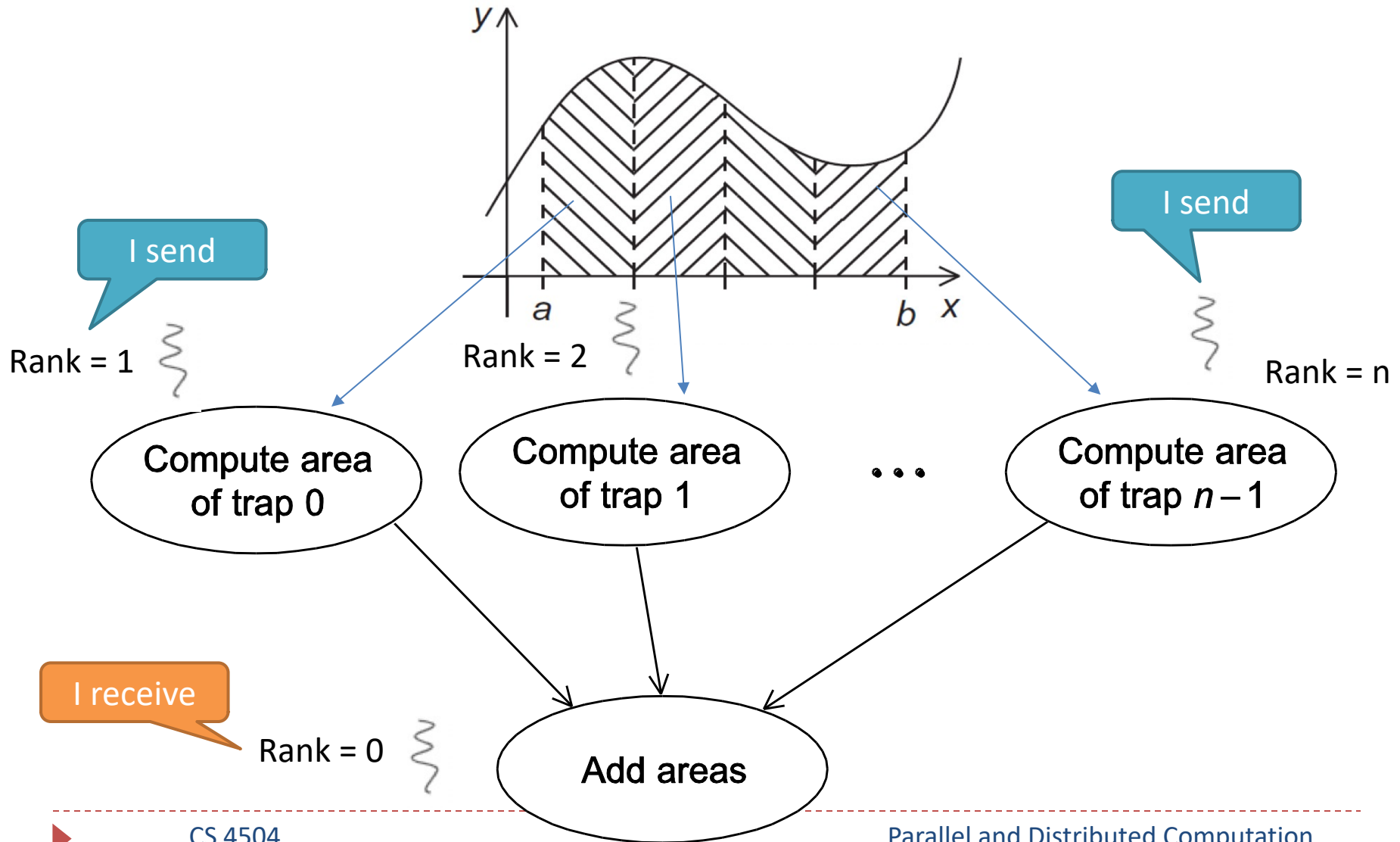


Outline

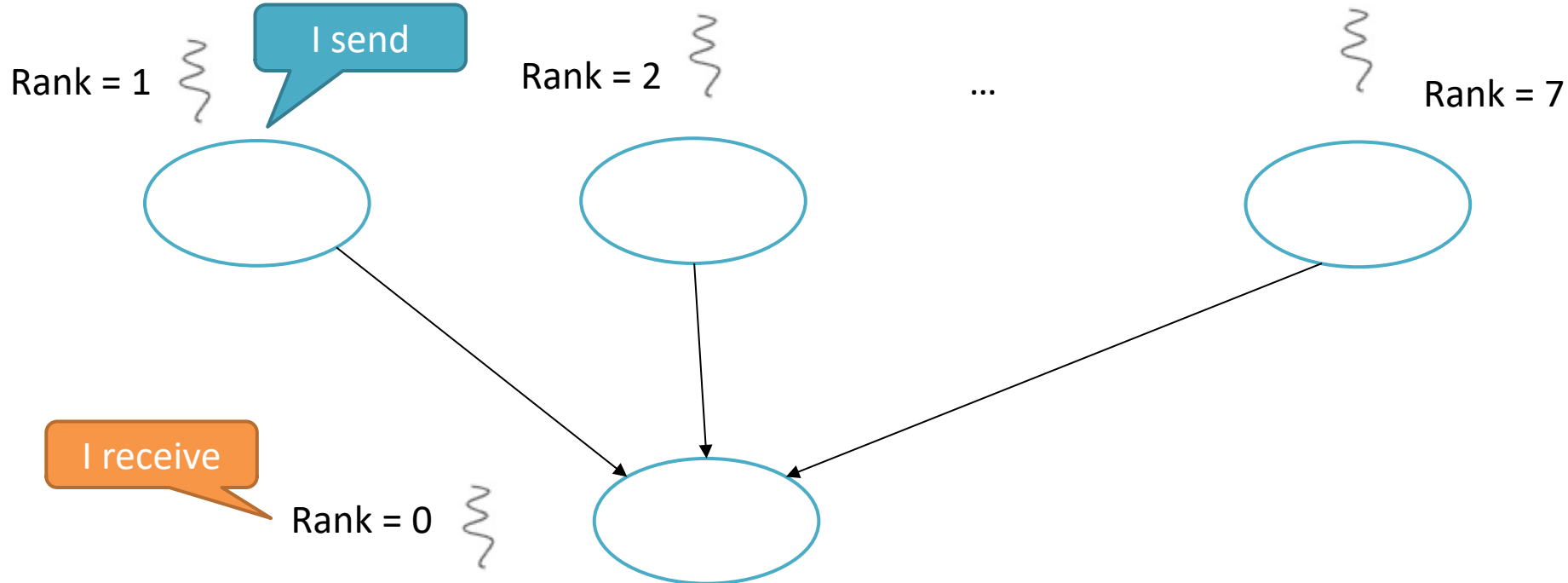
- MPI_Send/MPI_Receive
- MPI_Reduce/MPI_Allreduce
- MPI_Broadcast
- MPI_Scatter
- MPI_Gather



Communication



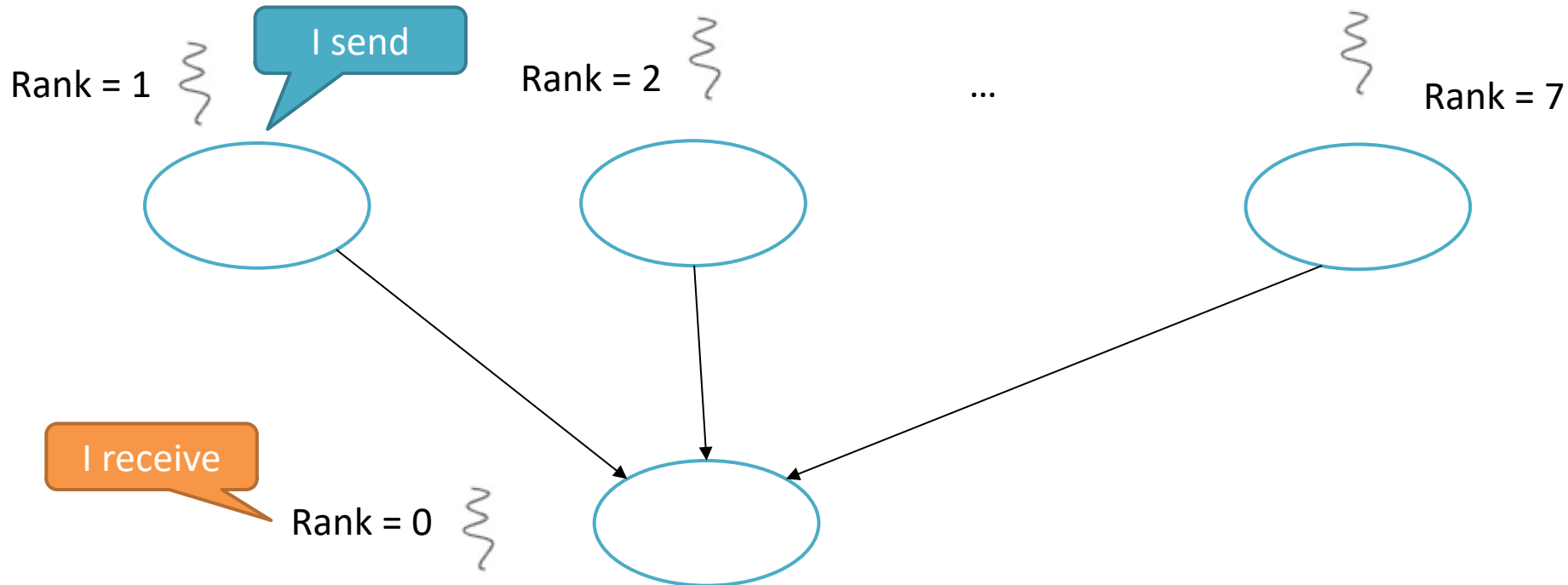
Point-to-Point communication



How many send/receive and add operation?

- 7 sends and 7 receives
- 7 adds

Point-to-Point communication

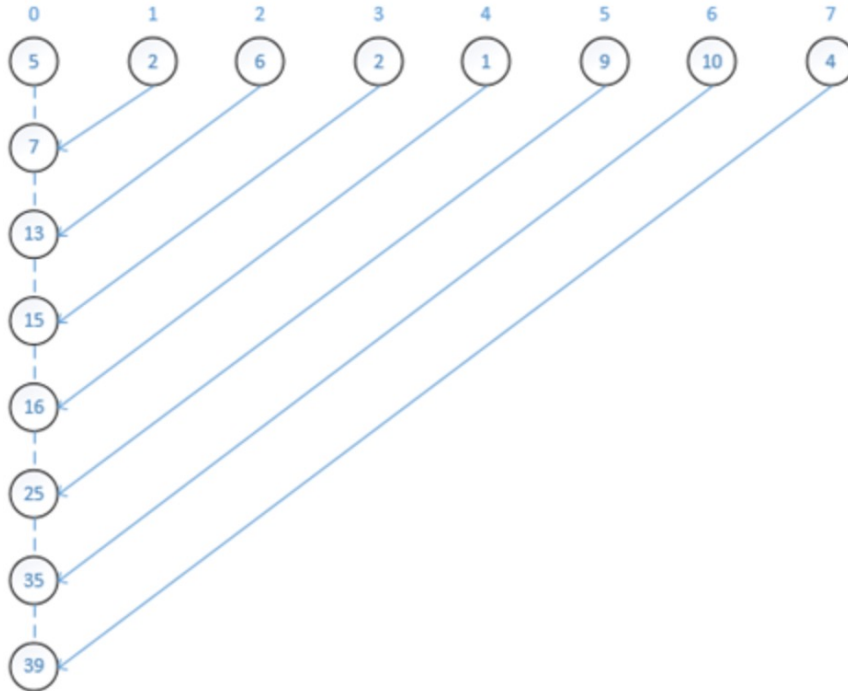


How many send/receive and add operation on master node (Rank = 0)?

- 7 sends and 7 receives
- 7 adds



Point-to-Point communication



How many send/receive and add operation on master node (Rank = 0)?

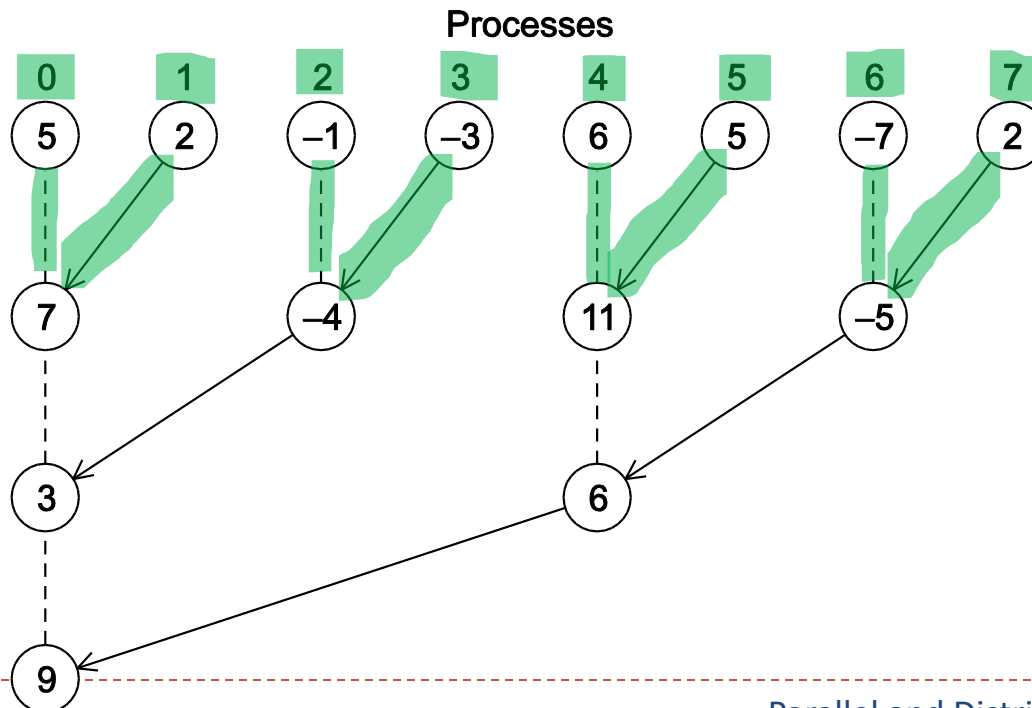
- 7 sends and 7 receives
- 7 adds



Tree-structured communication

In the first phase:

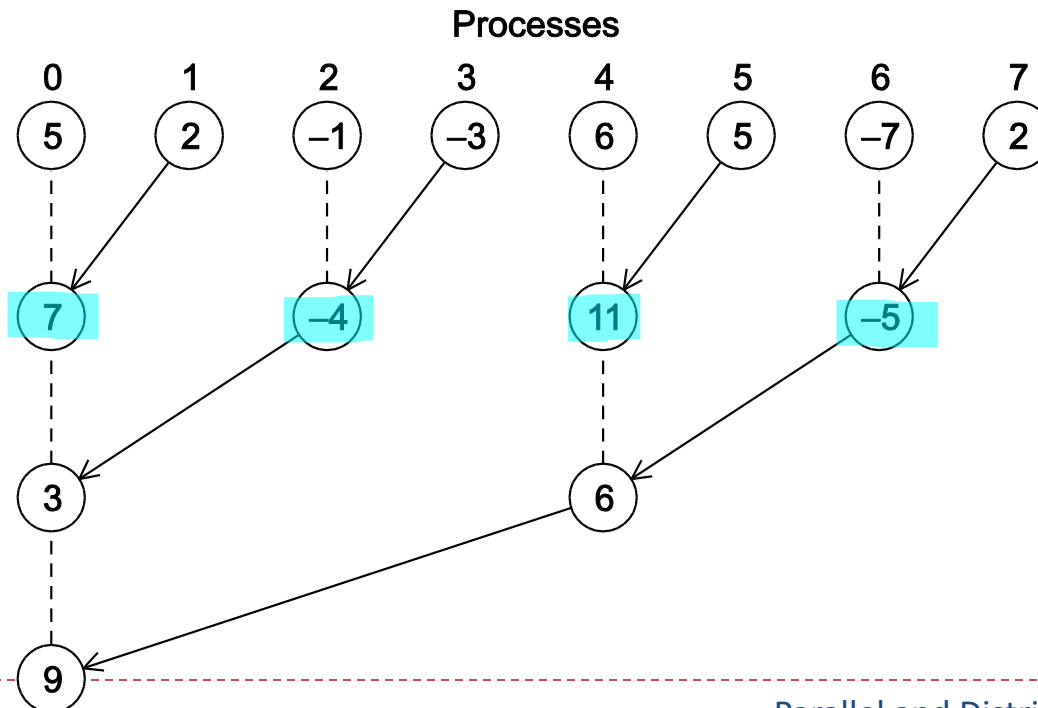
- (a) Process 1 sends to 0; 3 sends to 2; 5 sends to 4; and 7 sends to 6.
- (b) Processes 0, 2, 4, and 6 add in the received values.
- (c) Processes 2 and 6 send their new values to processes 0 and 4, respectively.
- (d) Processes 0 and 4 add the received values into their new values.



Tree-structured communication

In the first phase:

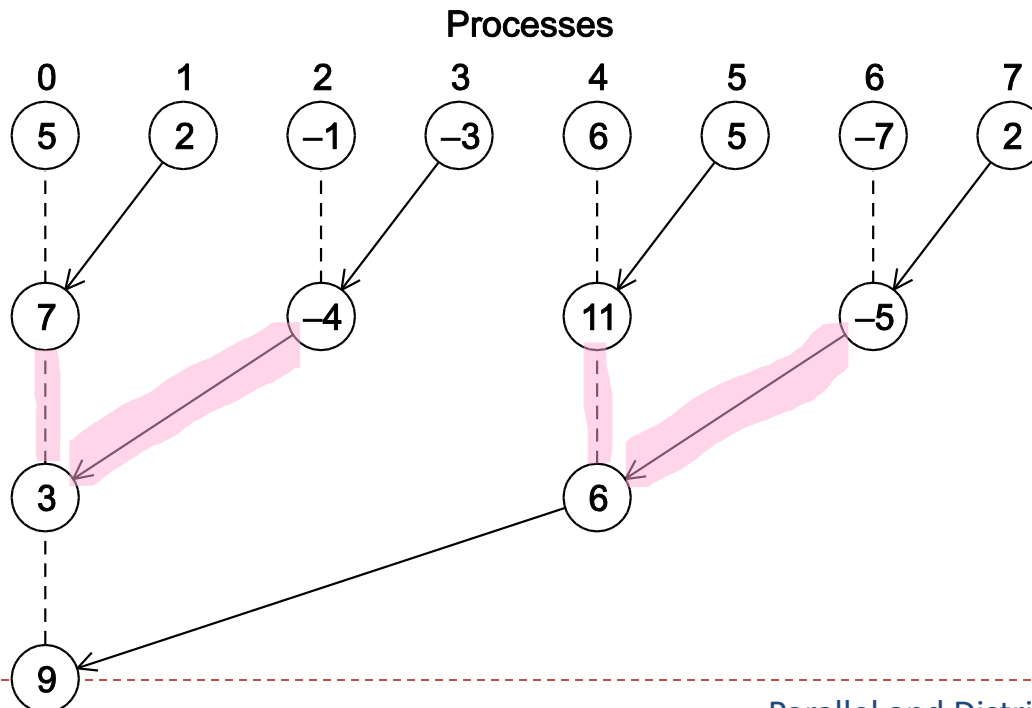
- (a) Process 1 sends to 0; 3 sends to 2; 5 sends to 4; and 7 sends to 6.
- (b) Processes 0, 2, 4, and 6 add in the received values.
- (c) Processes 2 and 6 send their new values to processes 0 and 4, respectively.
- (d) Processes 0 and 4 add the received values into their new values.



Tree-structured communication

In the first phase:

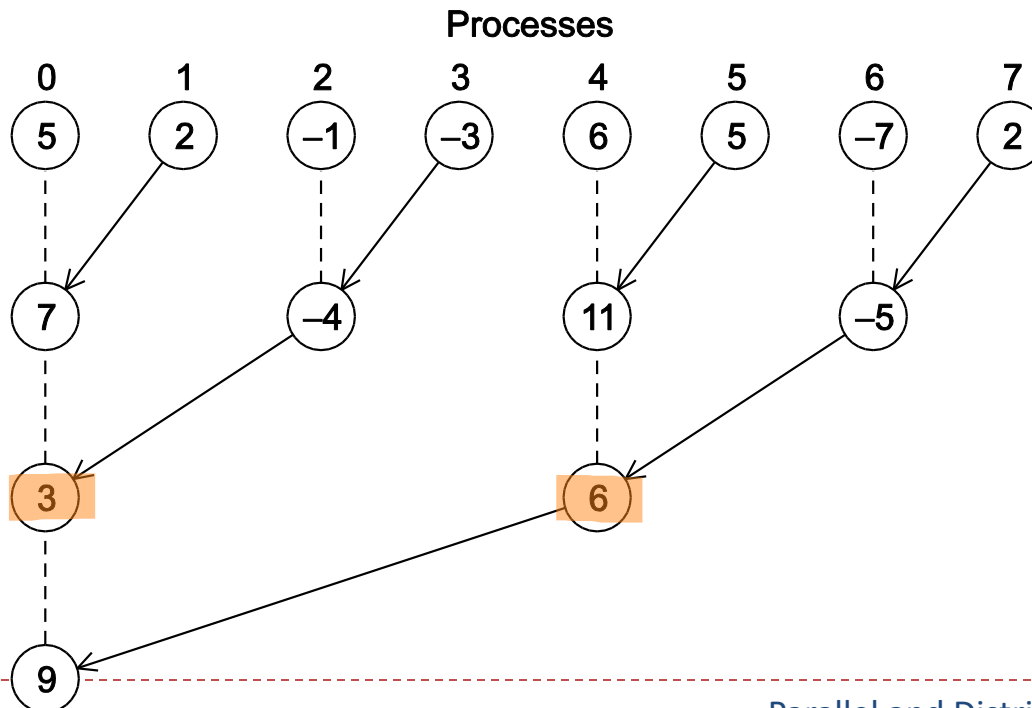
- (a) Process 1 sends to 0; 3 sends to 2; 5 sends to 4; and 7 sends to 6.
- (b) Processes 0, 2, 4, and 6 add in the received values.
- (c) Processes 2 and 6 send their new values to processes 0 and 4, respectively.
- (d) Processes 0 and 4 add the received values into their new values.



Tree-structured communication

In the first phase:

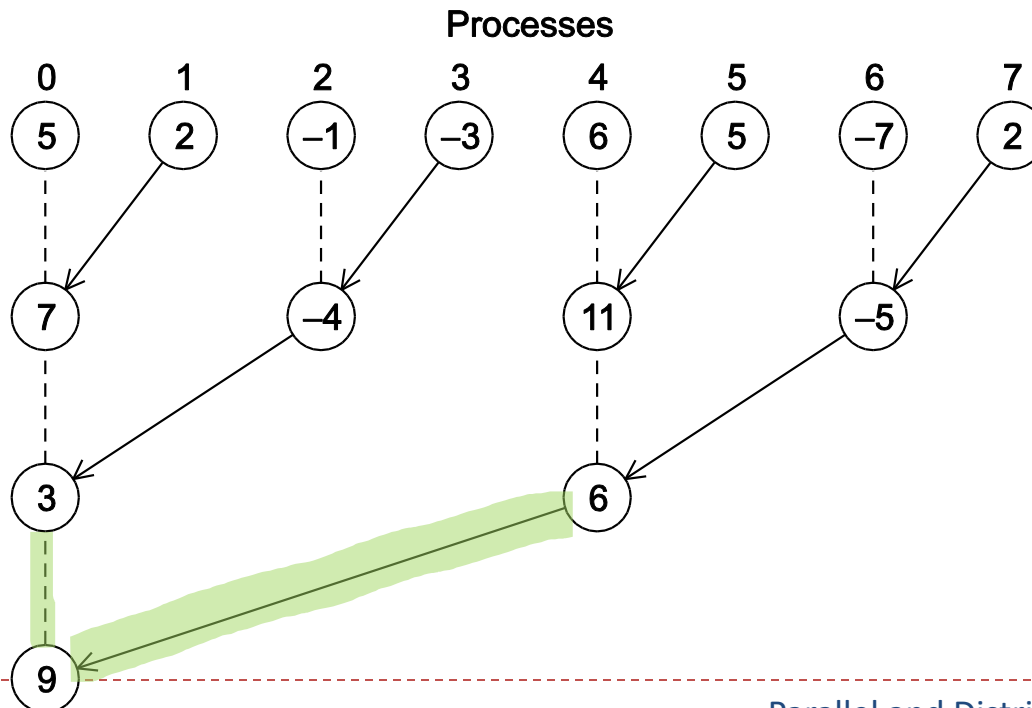
- (a) Process 1 sends to 0; 3 sends to 2; 5 sends to 4; and 7 sends to 6.
- (b) Processes 0, 2, 4, and 6 add in the received values.
- (c) Processes 2 and 6 send their new values to processes 0 and 4, respectively.
- (d) Processes 0 and 4 add the received values into their new values.



Tree-structured communication

In the second phase:

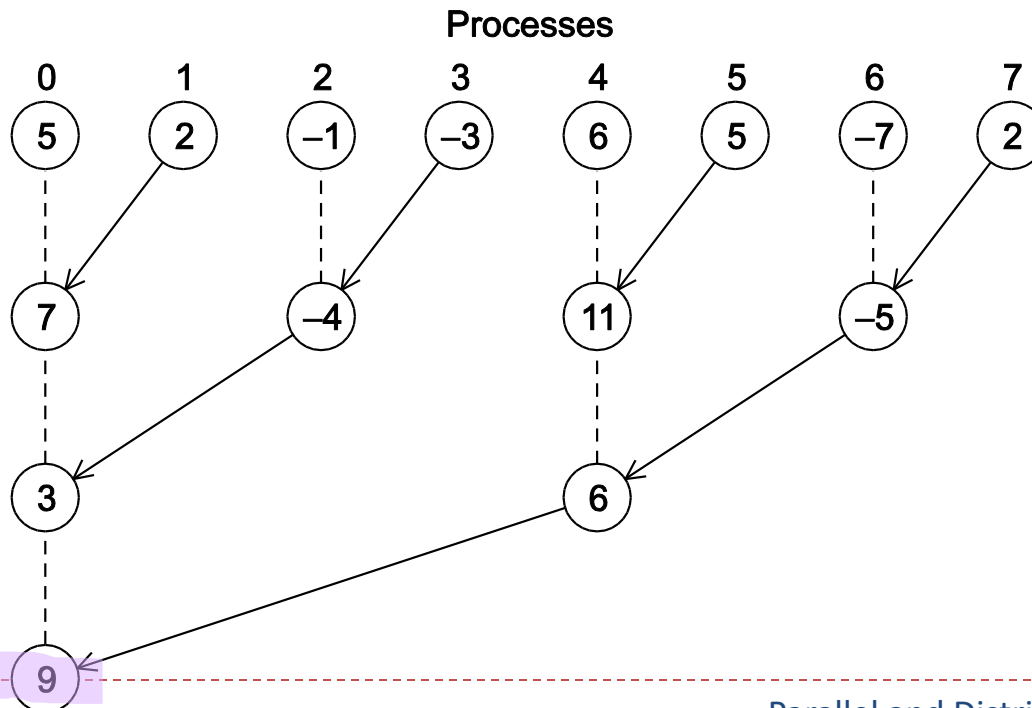
- (a) Process 4 sends its newest value to process 0.
- (b) Process 0 adds the received value to its newest value.



Tree-structured communication

In the second phase:

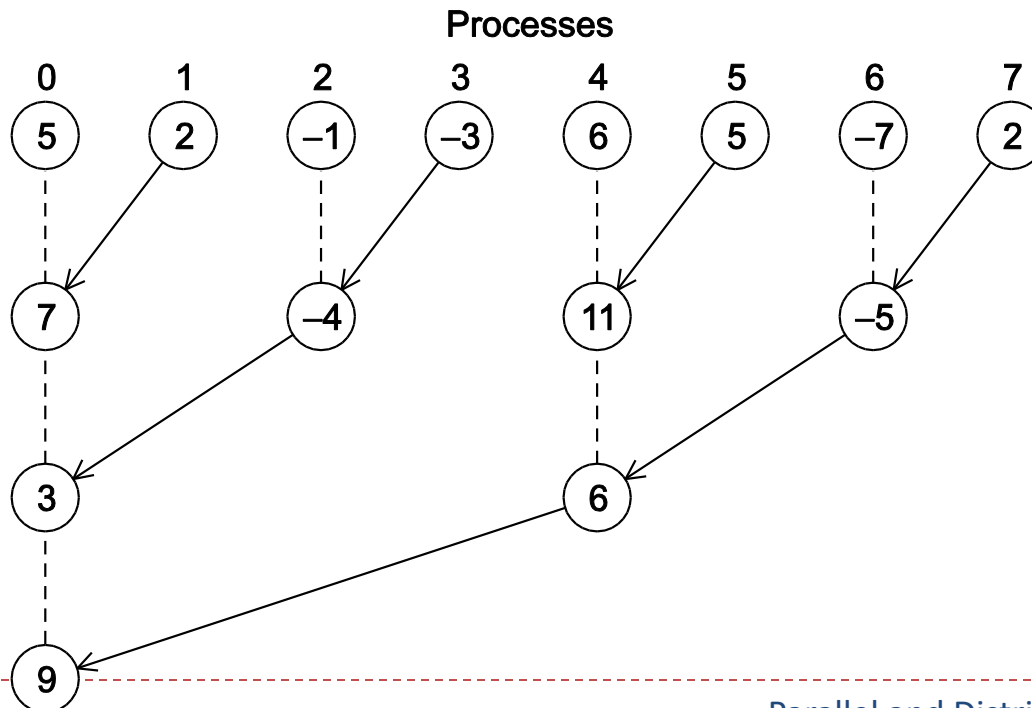
- (a) Process 4 sends its newest value to process 0.
- (b) Process 0 adds the received value to its newest value.



Tree-structured communication

How many send/receive and add operation?

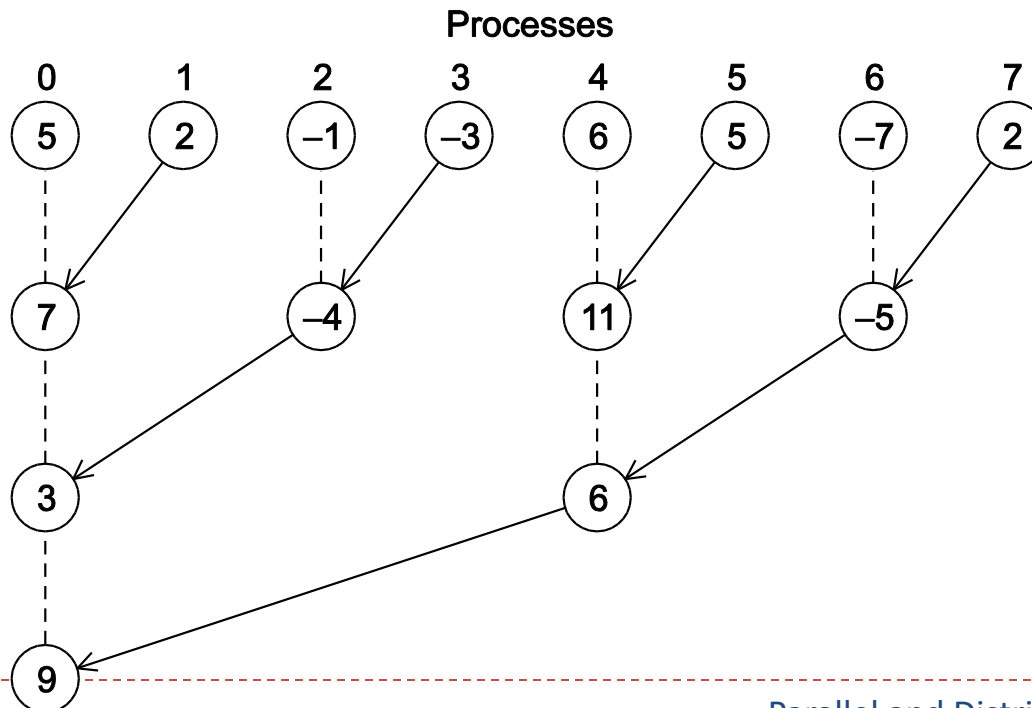
- 7 sends and 7 receives
- 7 adds



Tree-structured communication

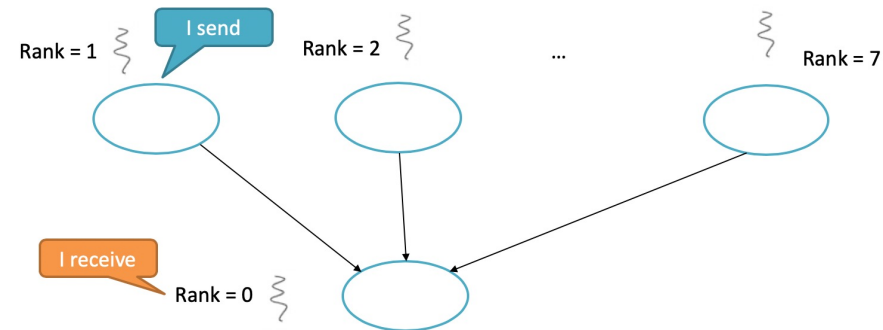
How many send/receive and add operation on master node (Rank = 0)?

- 3 sends and 3 receives
- 3 adds



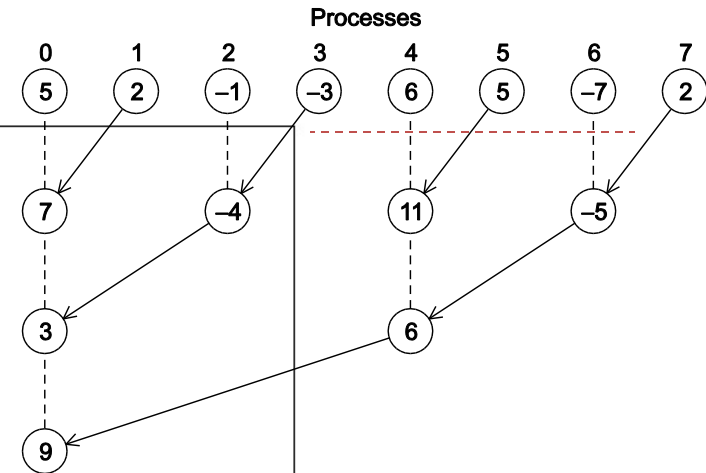
Point-to-Point communication code

```
1 int main(void) {
2     int my_rank, comm_sz, n = 1024, local_n;
3     double a = 0.0, b = 3.0, h, local_a, local_b;
4     double local_int, total_int;
5     int source;
6
7     MPI_Init(NULL, NULL);
8     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
9     MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
10
11     h = (b-a)/n;          /* h is the same for all processes */
12     local_n = n/comm_sz; /* So is the number of trapezoids */
13
14     local_a = a + my_rank*local_n*h;
15     local_b = local_a + local_n*h;
16     local_int = Trap(local_a, local_b, local_n, h);
17
18     if (my_rank != 0) {
19         MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
20                 MPI_COMM_WORLD);
21     } else {
22         total_int = local_int;
23         for (source = 1; source < comm_sz; source++) {
24             MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
25                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26             total_int += local_int;
27         }
28     }
29 }
```



Collective (tree-structure) communication code

```
1 int main(void) {
2     int my_rank, comm_sz, n = 1024, local_n;
3     double a = 0.0, b = 3.0, h, local_a, local_b;
4     double local_int, total_int;
5     int source;
6
7     MPI_Init(NULL, NULL);
8     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
9     MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
10
11     h = (b-a)/n;          /* h is the same for all processes */
12     local_n = n/comm_sz; /* So is the number of trapezoids */
13
14     local_a = a + my_rank*local_n*h;
15     local_b = local_a + local_n*h;
16     local_int = Trap(local_a, local_b, local_n, h);
17
18     if (my_rank != 0) {
19         MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
20                 MPI_COMM_WORLD);
21     } else {
22         total_int = local_int;
23         for (source = 1; source < comm_sz; source++) {
24             MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
25                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26             total_int += local_int;
27         }
28     }
29 }
```



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

MPI_Reduce

```
int MPI_Reduce(  
    void*      input_data_p    /* in */,  
    void*      output_data_p   /* out */,  
    int        count           /* in */,  
    MPI_Datatype datatype      /* in */,  
    MPI_Op      operator       /* in */,  
    int        dest_process    /* in */,  
    MPI_Comm    comm          /* in */);
```

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

input

output

Will do sum up for
the inputs

Process 0
will do it



MPI_Reduce

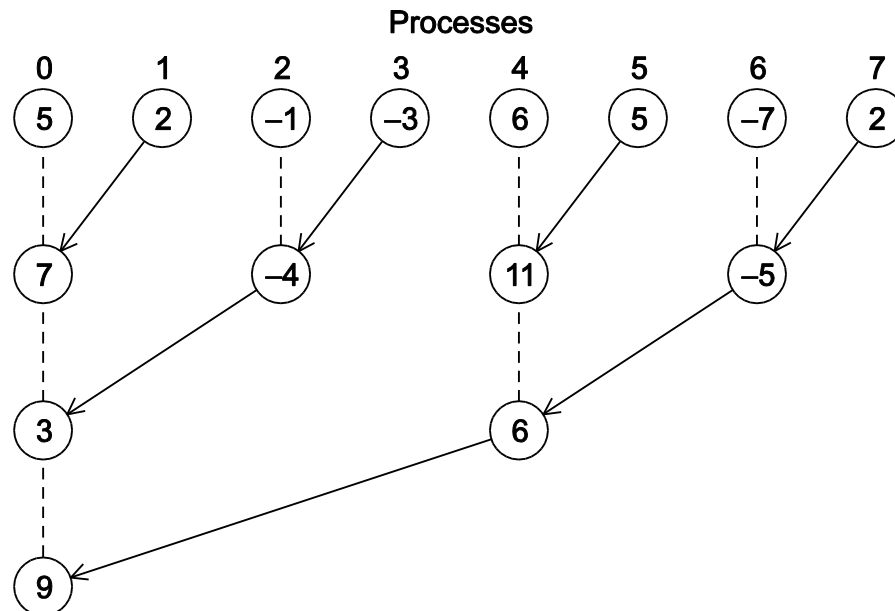
```
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,  
MPI_COMM_WORLD);
```

input

output

Will do sum up for
the inputs

Process 0
will do it



Besides Sumup, it can also do:

Operation Value	Meaning
<code>MPI_MAX</code>	Maximum
<code>MPI_MIN</code>	Minimum
<code>MPI_SUM</code>	Sum
<code>MPI_PROD</code>	Product
<code>MPI_LAND</code>	Logical and
<code>MPI_BAND</code>	Bitwise and
<code>MPI_LOR</code>	Logical or
<code>MPI_BOR</code>	Bitwise or
<code>MPI_LXOR</code>	Logical exclusive or
<code>MPI_BXOR</code>	Bitwise exclusive or
<code>MPI_MAXLOC</code>	Maximum and location of maximum
<code>MPI_MINLOC</code>	Minimum and location of minimum

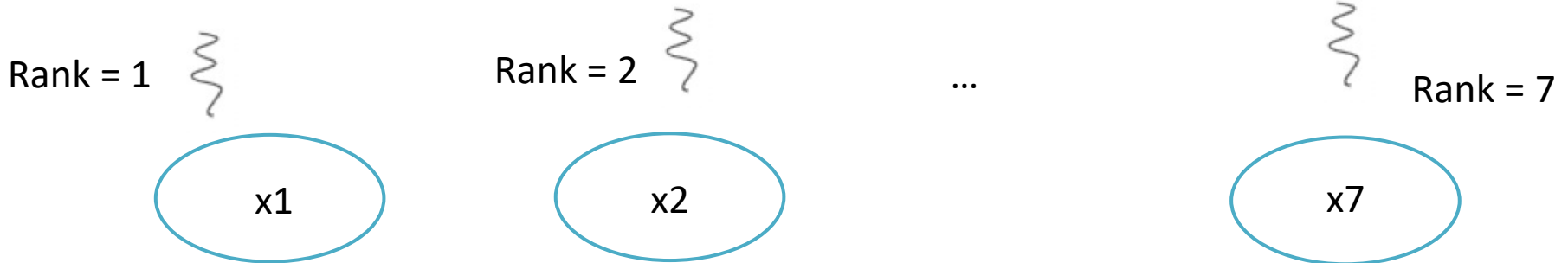


Besides Sumup, it can also do:

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

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

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



$\text{MAX}(x1, x2, \dots, x7)$

$\text{MIN}(x1, x2, \dots, x7)$

$x1 * x2 * \dots * x7$



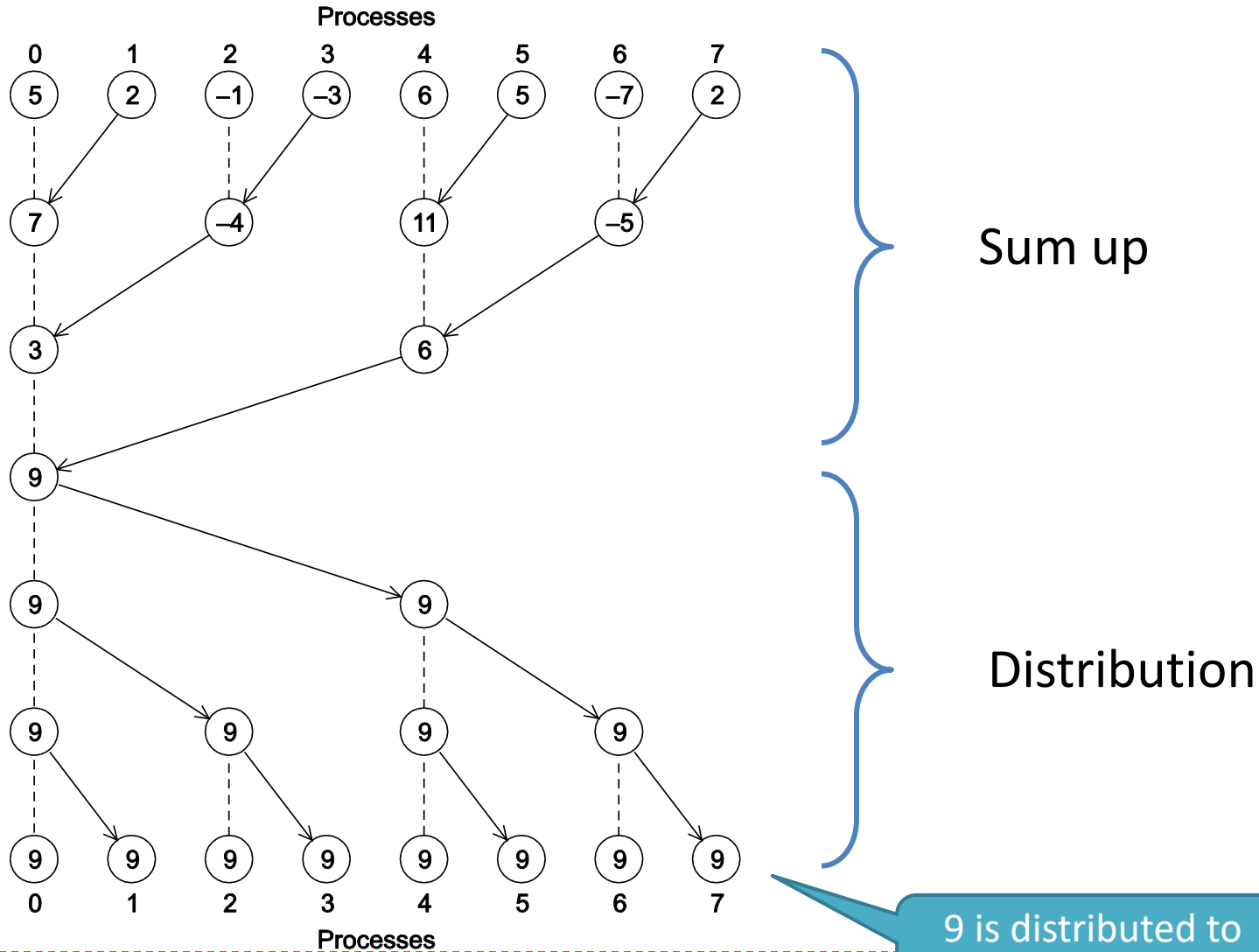
MPI_Allreduce

- Useful in a situation in which all of the processes need the final result.
 - Sum up first and then distribute to everyone

```
int MPI_Allreduce(  
    void*      input_data_p    /* in */,  
    void*      output_data_p   /* out */,  
    int        count           /* in */,  
    MPI_Datatype datatype      /* in */,  
    MPI_Op      operator       /* in */,  
    MPI_Comm    comm           /* in */);
```



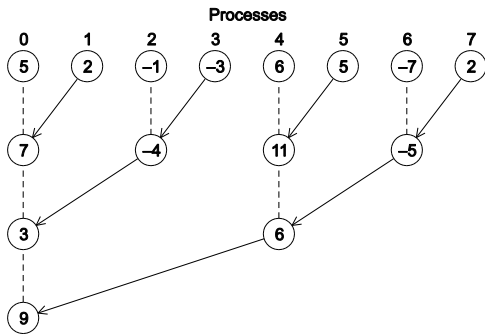
A global sum followed by distribution of the result



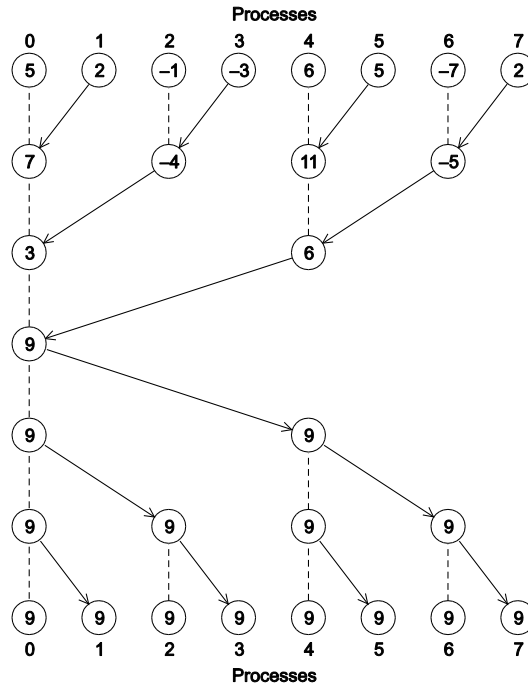
9 is distributed to every process

What if I only need to distribute?

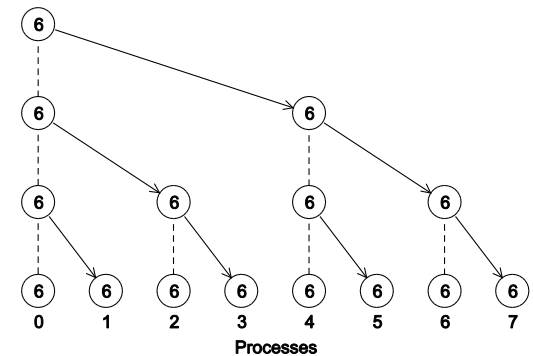
MPI_Reduce



MPI_Allreduce



?

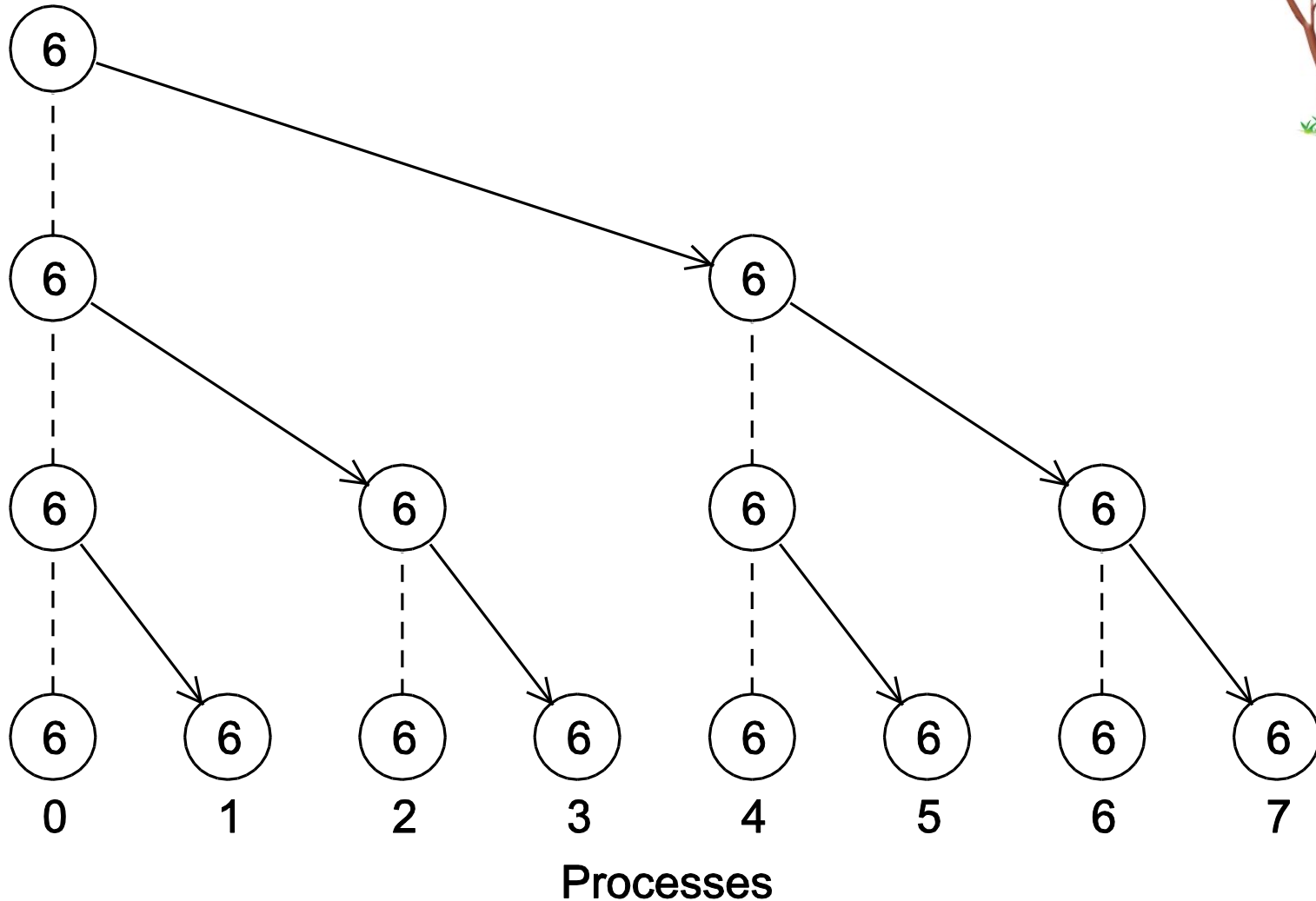


Broadcast

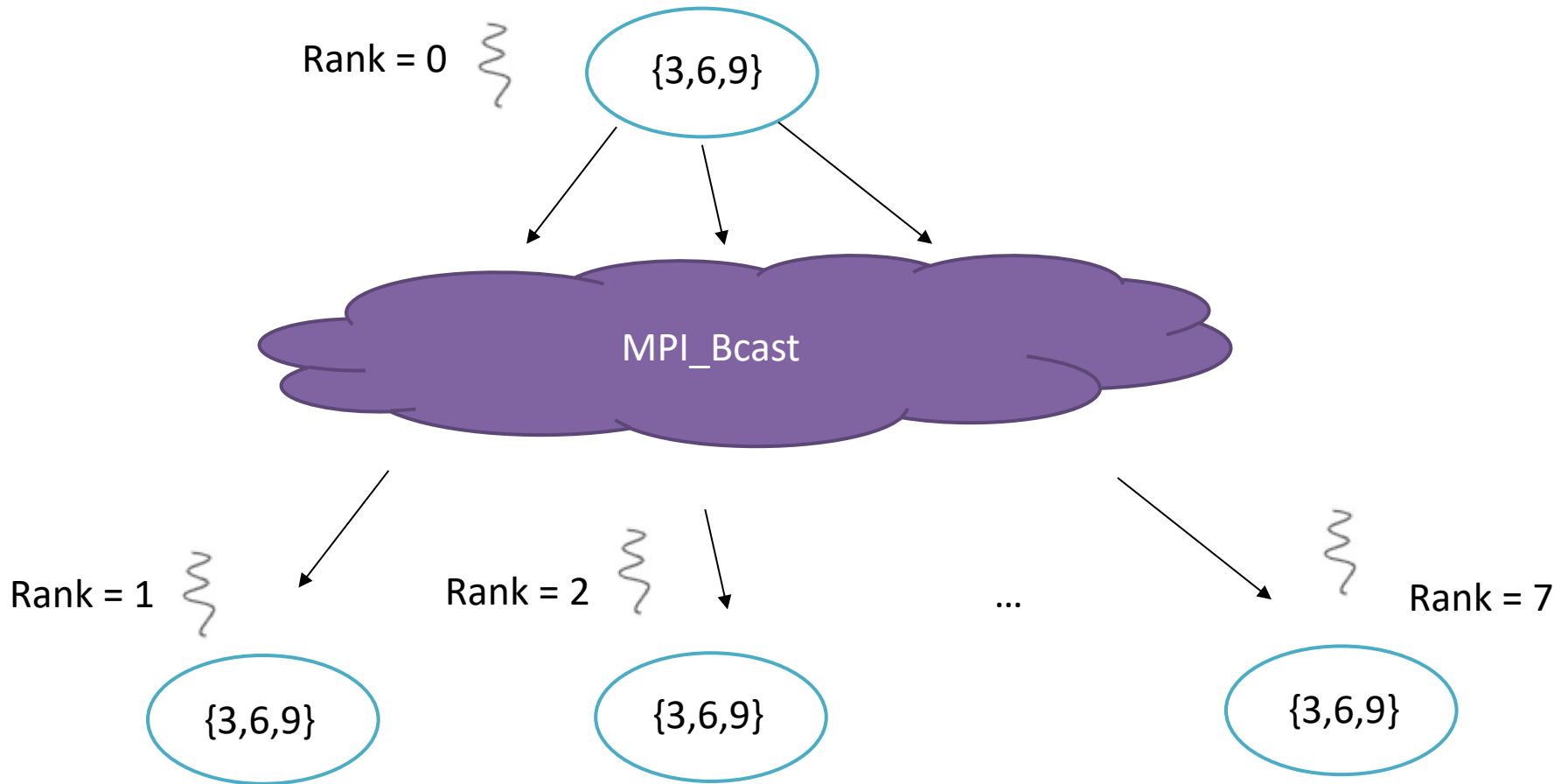
- Data belonging to a single process is sent to all of the processes in the communicator.

```
int MPI_Bcast(  
    void* data_p          /* in/out */,  
    int count             /* in      */,  
    MPI_Datatype datatype /* in      */,  
    int source_proc       /* in      */,  
    MPI_Comm comm         /* in      */);
```

A tree-structured broadcast



Example of MPI_Bcast



Example of MPI_Bcast

```
int main(int argc, char* argv[])
{
    blog3 test;
    test.TestForMPI_Bcast(argc, argv);
}
```

```
void blog3::TestForMPI_Bcast(int argc, char* argv[])
{
    int rankID, totalNumTasks;

    MPI_Init(&argc, &argv);
    MPI_Barrier(MPI_COMM_WORLD);
    double elapsed_time = -MPI_Wtime();

    MPI_Comm_rank(MPI_COMM_WORLD, &rankID);
    MPI_Comm_size(MPI_COMM_WORLD, &totalNumTasks);

    int sendRecvBuf[3] = { 0, 0, 0 };
```



```
void blog3::TestForMPI_Bcast(int argc, char* argv[])
{
    int rankID, totalNumTasks;

    MPI_Init(&argc, &argv);
    MPI_Barrier(MPI_COMM_WORLD);
    double elapsed_time = -MPI_Wtime();

    MPI_Comm_rank(MPI_COMM_WORLD, &rankID);
    MPI_Comm_size(MPI_COMM_WORLD, &totalNumTasks);

    int sendRecvBuf[3] = { 0, 0, 0 };

    if (!rankID) {
        sendRecvBuf[0] = 3;
        sendRecvBuf[1] = 6;
        sendRecvBuf[2] = 9;
    }

    int count = 3;
    int root = 0;
    MPI_Bcast(sendRecvBuf, count, MPI_INT, root, MPI_COMM_WORLD); //MPI_Bcast can be seen from all processes

    printf("my rankID = %d, sendRecvBuf = {%d, %d, %d}\n", rankID, sendRecvBuf[0], sendRecvBuf[1], sendRecvBuf[2]);

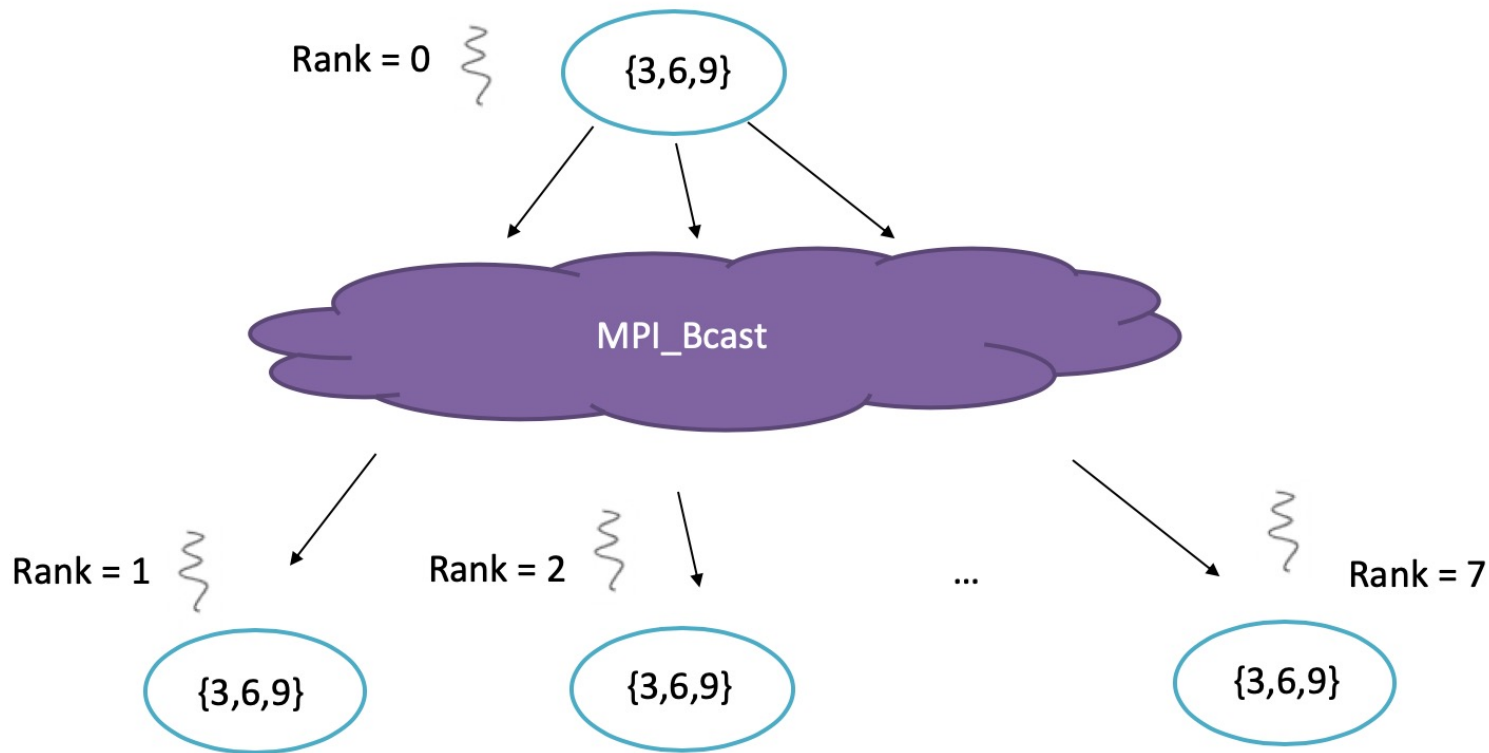
    elapsed_time += MPI_Wtime();
    if (!rankID) {
        printf("total elapsed time = %10.6f\n", elapsed_time);
    }

    MPI_Finalize();
}
```

buffer is {0,0,0}

rankID = 0 will init
buffer as {3,6,9}

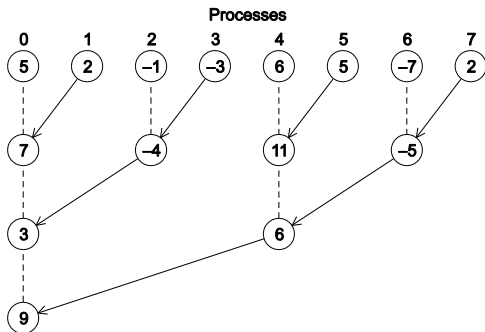
Print the received data



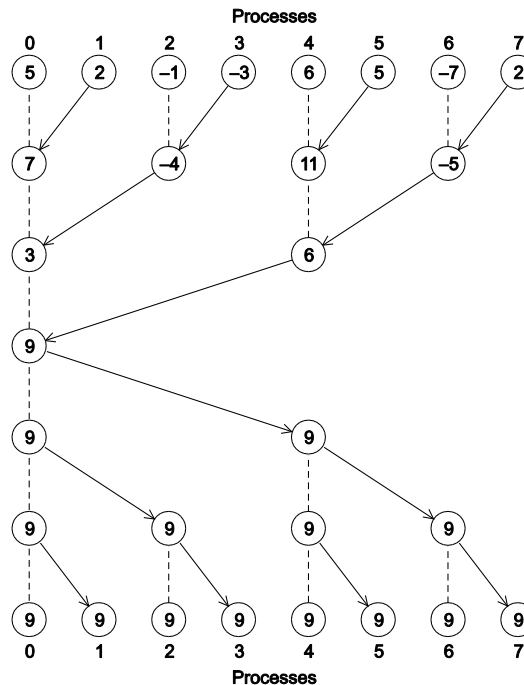
```
E:\CPPTTest\PaiallelWorkspace\Debug>mpiexec -n 8 Paiallel.exe
my rankID = 6, sendRecvBuf = {3, 6, 9}
my rankID = 3, sendRecvBuf = {3, 6, 9}
my rankID = 4, sendRecvBuf = {3, 6, 9}
my rankID = 7, sendRecvBuf = {3, 6, 9}
my rankID = 0, sendRecvBuf = {3, 6, 9}
total elapsed time = 0.000120
my rankID = 2, sendRecvBuf = {3, 6, 9}
my rankID = 1, sendRecvBuf = {3, 6, 9}
my rankID = 5, sendRecvBuf = {3, 6, 9}
```

What if I only need to distribute?

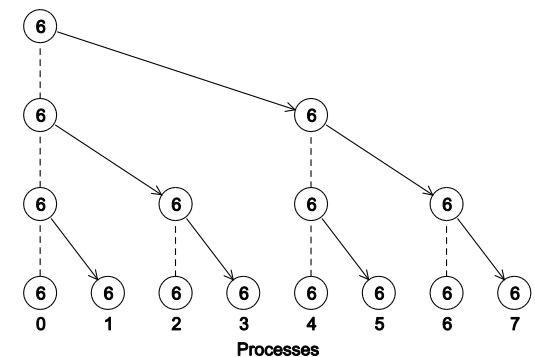
MPI_Reduce



MPI_Allreduce



MPI_Bcast



Data distributions: vector addition

a

1		-1		3		3	
0.6	+	0	+	1	=	1.6	
-2		0.2		0		-1.8	

b

1	0	+	0	-1	=	1	-1
-1	0		-1	0		-2	0



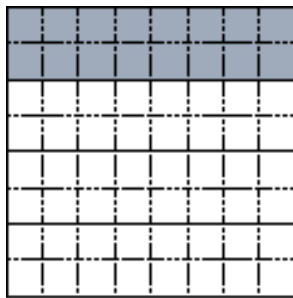
Partitioning options

- Block partitioning
 - Assign blocks of consecutive components to each process.
- Cyclic partitioning
 - Assign components in a round robin fashion.
- Block-cyclic partitioning
 - Use a cyclic distribution of blocks of components.

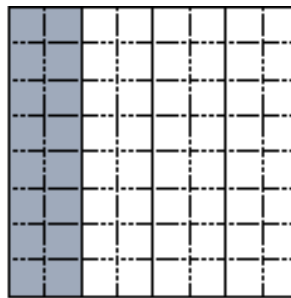


Partitioning options

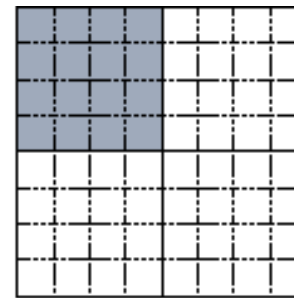
- Block partitioning vs Cyclic partitioning



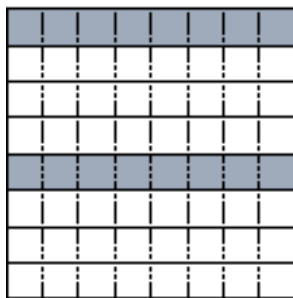
$\{ \text{BLOCK}, * \}$



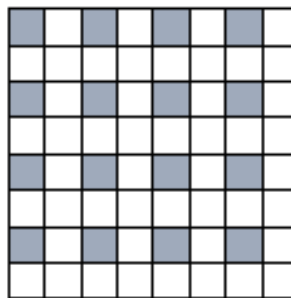
$\{ *, \text{BLOCK} \}$



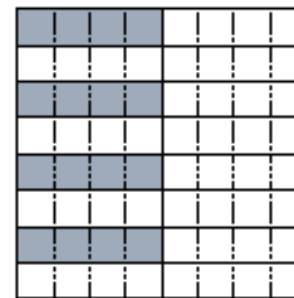
$\{ \text{BLOCK}, \text{BLOCK} \}$



$\{ \text{CYCLIC}, * \}$



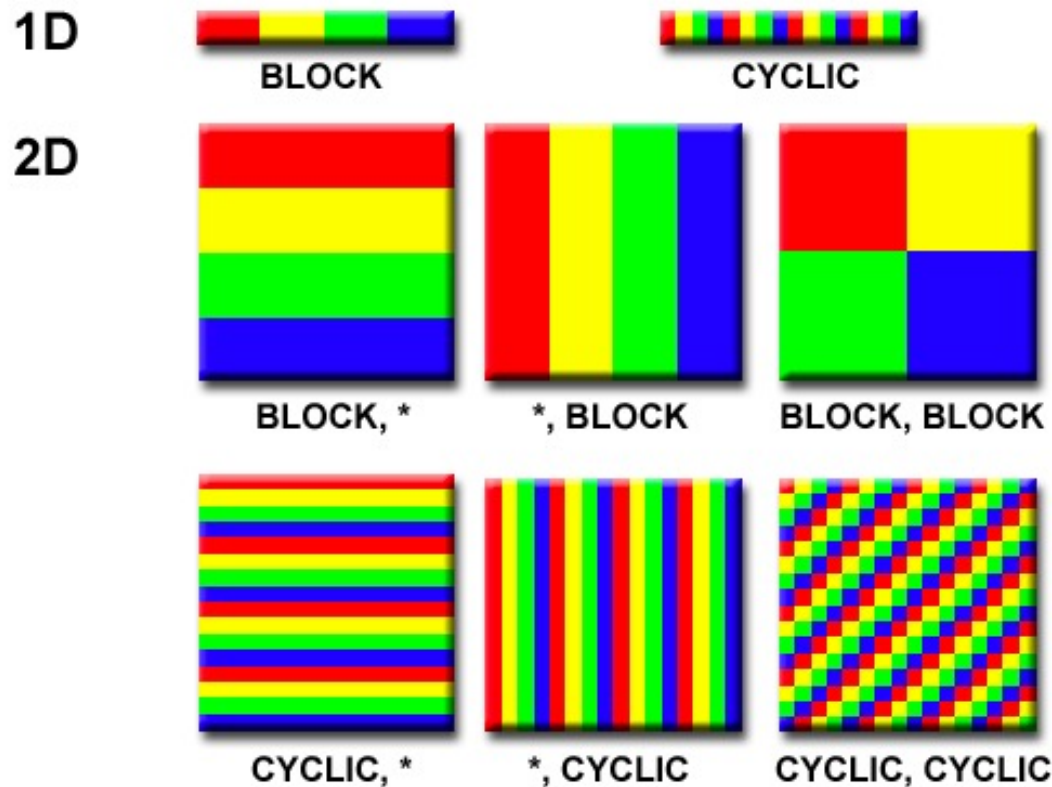
$\{ \text{CYCLIC}, \text{CYCLIC} \}$



$\{ \text{CYCLIC}, \text{BLOCK} \}$

Partitioning options

- Block partitioning vs Cyclic partitioning

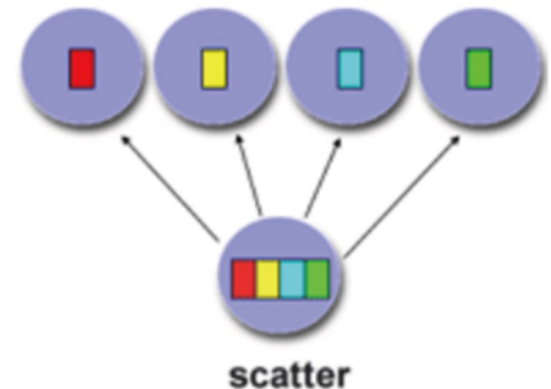


Scatter: partition & distribution

- MPI_Scatter can be used in a function that reads in an entire vector on process 0 but only sends the needed components to each of the other processes.

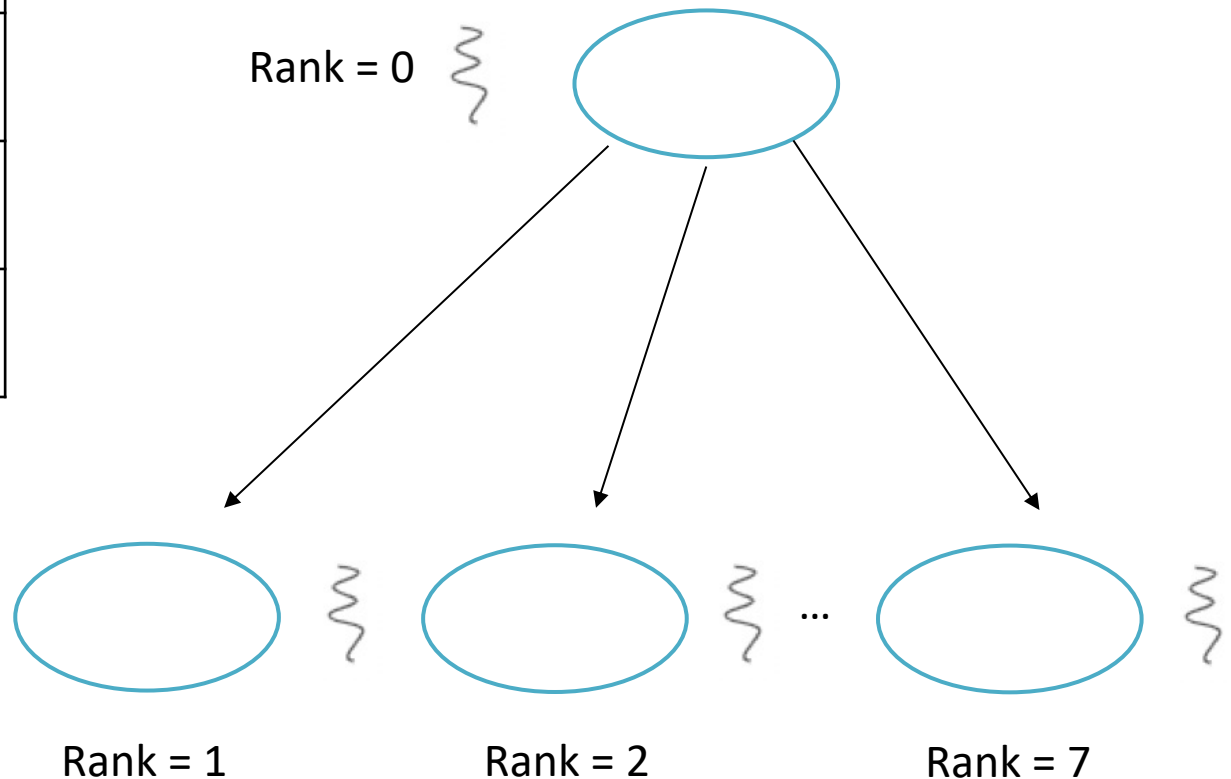
```
int MPI_Scatter(  
    void* send_buf_p, /* in */  
    int send_count, /* in */  
    MPI_Datatype send_type, /* in */  
    void* recv_buf_p, /* out */  
    int recv_count, /* in */  
    MPI_Datatype recv_type, /* in */  
    int src_proc, /* in */  
    MPI_Comm comm /* in */);
```

sendCount, each
process gets n data



Example of MPI_Scatter

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



Example of MPI_Scatter

```
int main(int argc, char* argv[])
{
    blog3 test;

    test.TestForMPI_Scatter(argc, argv);

    return 0;
}

void blog3::TestForMPI_Scatter(int argc, char* argv[])
{
    int totalNumTasks, rankID;

    float sendBuf[SIZE][SIZE] = {
        { 1.0, 2.0, 3.0, 4.0 },
        { 5.0, 6.0, 7.0, 8.0 },
        { 9.0, 10.0, 11.0, 12.0 },
        { 13.0, 14.0, 15.0, 16.0 }
    };

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rankID);
    MPI_Comm_size(MPI_COMM_WORLD, &totalNumTasks);
}
```

Size = 4

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Example of MPI_Scatter

```
if (totalNumTasks == SIZE) {  
    int source = 0;  
    int sendCount = SIZE;  
    int recvCount = SIZE;  
    float recvBuf[SIZE];  
    //scatter data from source process to all processes in MPI_COMM_WORLD  
    MPI_Scatter(sendBuf, sendCount, MPI_FLOAT,  
               recvBuf, recvCount, MPI_FLOAT, source, MPI_COMM_WORLD);  
  
    printf("my rankID = %d, receive Results: %f %f %f %f, total = %f\n",  
           rankID, recvBuf[0], recvBuf[1], recvBuf[2], recvBuf[3],  
           recvBuf[0] + recvBuf[1] + recvBuf[2] + recvBuf[3]);  
}  
else if (totalNumTasks == 8) {  
    int source = 0;  
    int sendCount = 2;  
    int recvCount = 2;  
    float recvBuf[2];  
  
    MPI_Scatter(sendBuf, sendCount, MPI_FLOAT,  
               recvBuf, recvCount, MPI_FLOAT, source, MPI_COMM_WORLD);  
  
    printf("my rankID = %d, receive result: %f %f, total = %f\n",  
           rankID, recvBuf[0], recvBuf[1], recvBuf[0] + recvBuf[1]);  
}  
else {  
    printf("Please specify -n %d or -n %d\n", SIZE, 2 * SIZE);  
}  
  
MPI_Finalize();
```

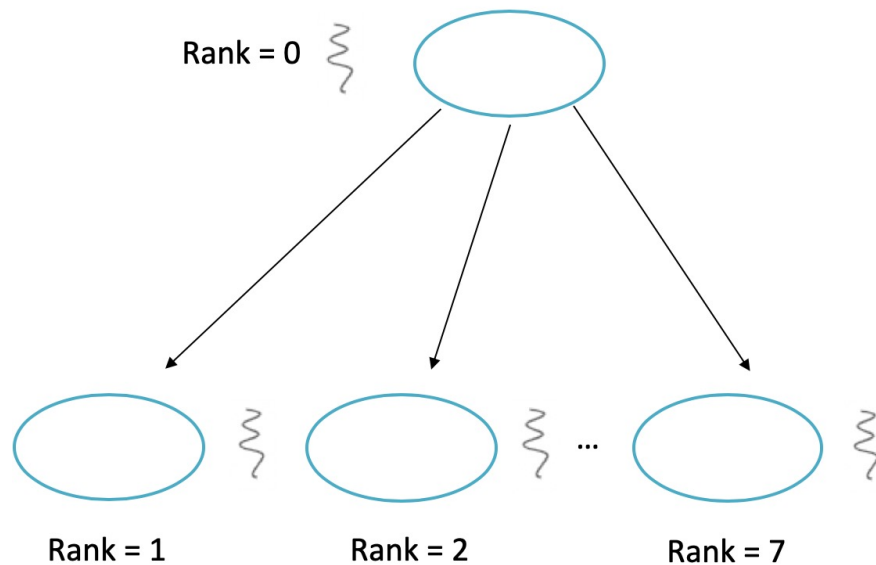
Size = 4

sendCount = 2, each
process gets 2 data

After each process
receives the data,
sum them up

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Example of MPI_Scatter



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

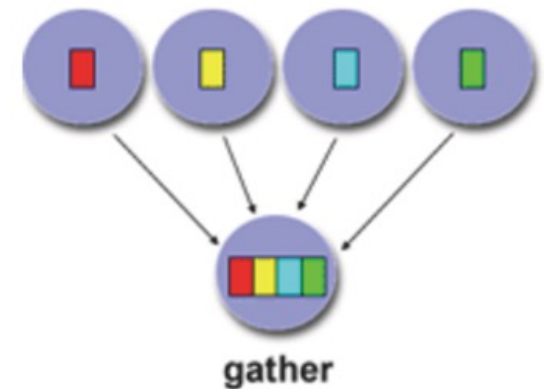
```
E:\CPPTest\PaiallelWorkspace\Debug>mpiexec -n 8 Paiallel.exe
my rankID = 3, receive result: 7.000000 8.000000, total = 15.000000
my rankID = 5, receive result: 11.000000 12.000000, total = 23.000000
my rankID = 2, receive result: 5.000000 6.000000, total = 11.000000
my rankID = 0, receive result: 1.000000 2.000000, total = 3.000000
my rankID = 7, receive result: 15.000000 16.000000, total = 31.000000
my rankID = 4, receive result: 9.000000 10.000000, total = 19.000000
my rankID = 1, receive result: 3.000000 4.000000, total = 7.000000
my rankID = 6, receive result: 13.000000 14.000000, total = 27.000000
```


Gather

- Collect all of the components of the vector onto process 0, and then process 0 can process all of the components.

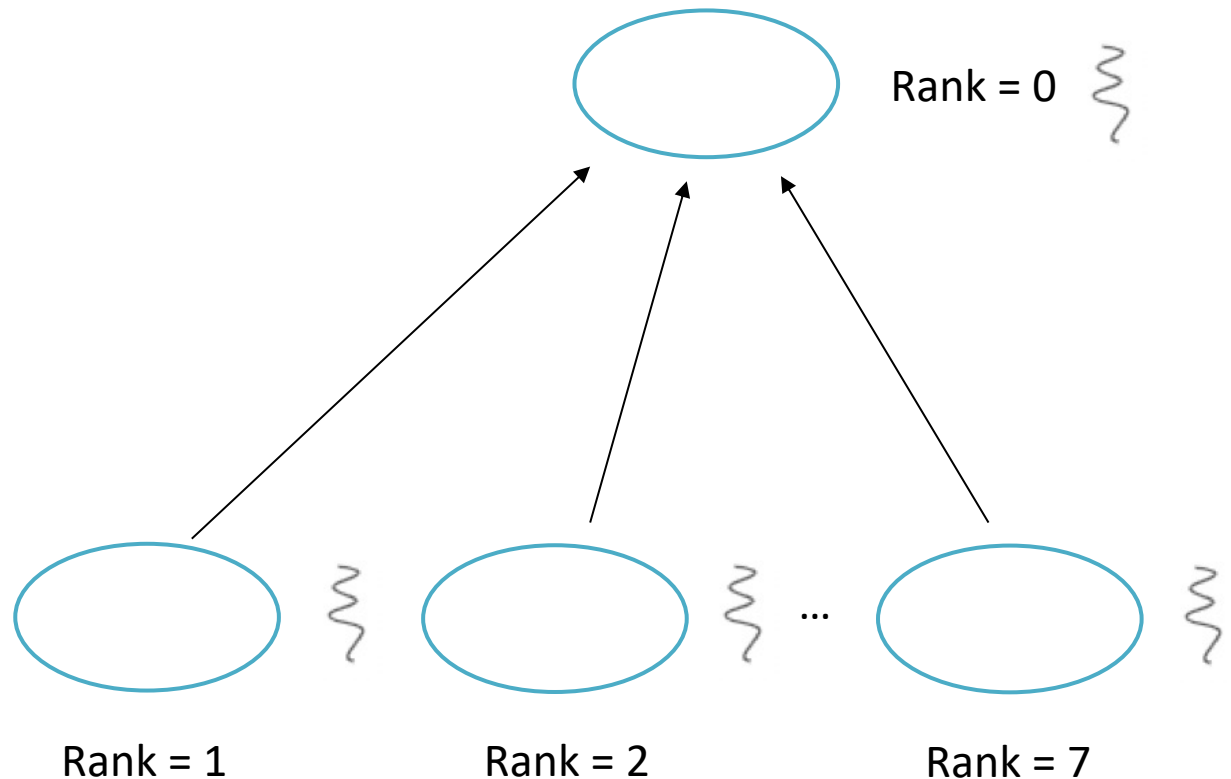
```
int MPI_Gather(  
    void* send_buf_p    /* in */,  
    int send_count      /* in */,  
    MPI_Datatype send_type /* in */,  
    void* recv_buf_p    /* out */,  
    int recv_count      /* in */,  
    MPI_Datatype recv_type /* in */,  
    int dest_proc       /* in */,  
    MPI_Comm comm       /* in */);
```

Send data



Receiver ID

Example of MPI_Gather



Each process sends
its rankID to the
array in process 0

Example of MPI_Gather

```
int main(int argc, char* argv[])
{
    blog3 test;

    test.TestForMPI_Gather(argc, argv);

    return 0;
}

void blog3::TestForMPI_Gather(int argc, char* argv[])
{
    int rankID, totalNumTasks;

    MPI_Init(&argc, &argv);
    MPI_Barrier(MPI_COMM_WORLD);
    double elapsed_time = -MPI_Wtime();

    MPI_Comm_rank(MPI_COMM_WORLD, &rankID);
    MPI_Comm_size(MPI_COMM_WORLD, &totalNumTasks);
```



Example of MPI_Gather



```
int* gatherBuf = (int *)malloc(sizeof(int) * totalNumTasks);
if (gatherBuf == NULL) {
    printf("malloc error!");
    exit(-1);
    MPI_Finalize();
}

int sendBuf = rankID; //for each process, its rankID will be sent out

int sendCount = 1;
int recvCount = 1;
int root = 0;
MPI_Gather(&sendBuf, sendCount, MPI_INT, gatherBuf, recvCount, MPI_INT, root, MPI_COMM_WORLD);

elapsed_time += MPI_Wtime();
if (!rankID) {
    int i;
    for (i = 0; i < totalNumTasks; i++) {
        printf("gatherBuf[%d] = %d, ", i, gatherBuf[i]);
    }
    putchar('\n');
    printf("total elapsed time = %10.6f\n", elapsed_time);
}

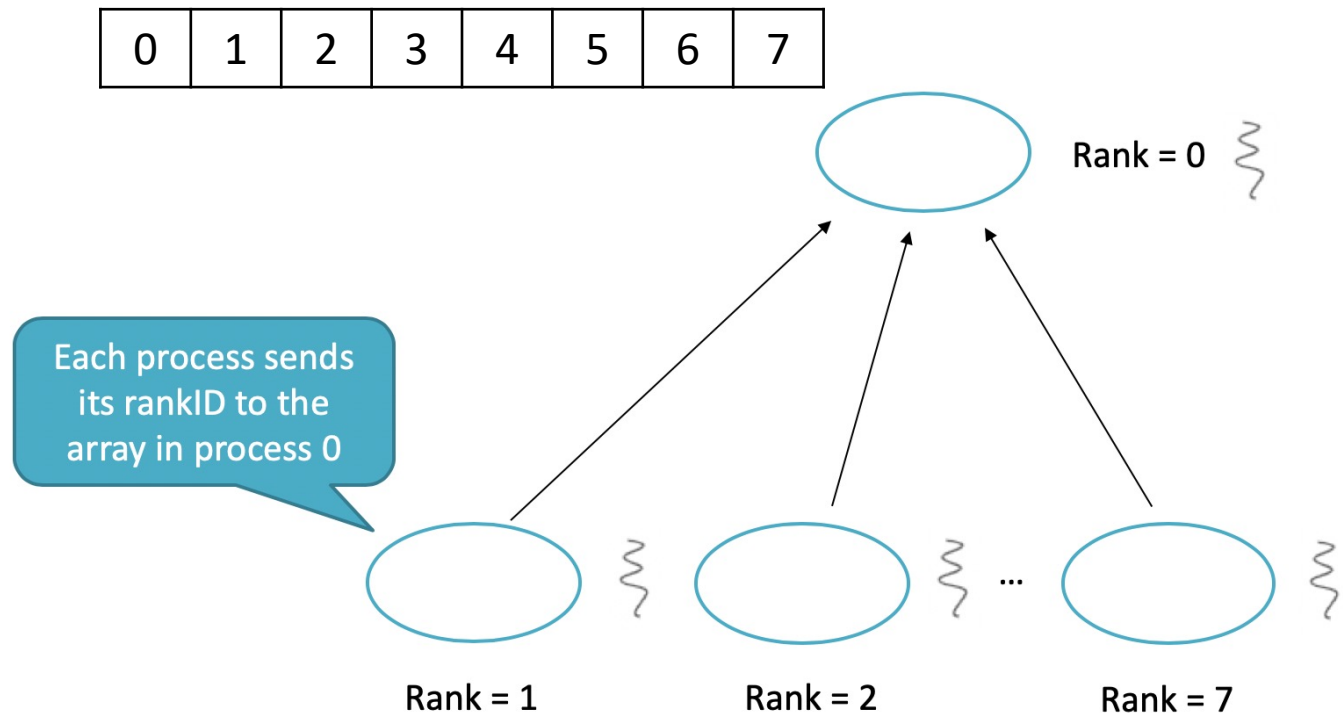
MPI_Finalize();
```

The sent content is the rankID

Data size is 1

Receiver is process 0

Example of MPI_Gather



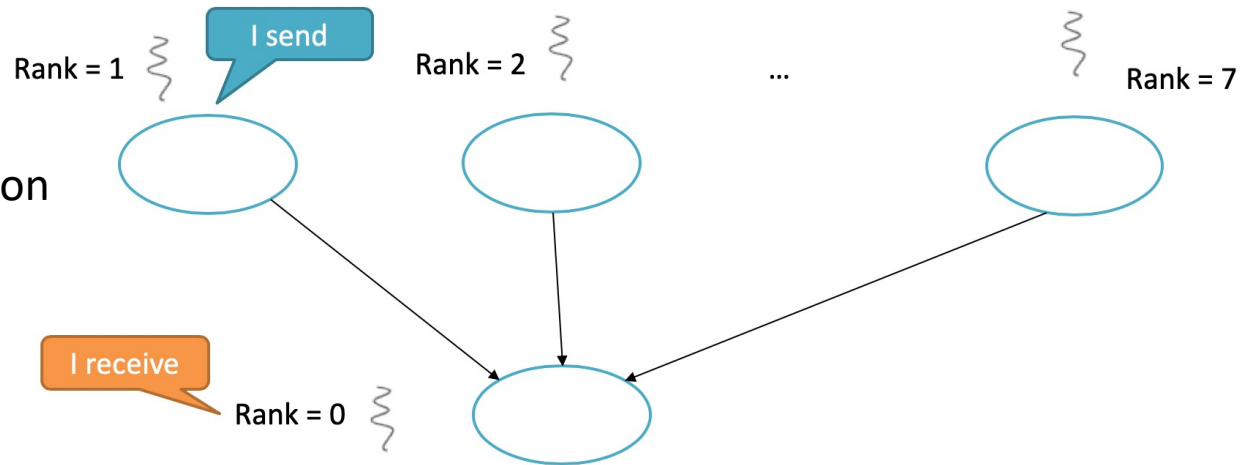
```
E:\CPPTest\PaiallelWorkspace\Debug>mpiexec -n 8 Paiallel.exe
gatherBuf[0] = 0, gatherBuf[1] = 1, gatherBuf[2] = 2, gatherBuf[3] = 3, gatherBuf[4] = 4, gatherB
uf[5] = 5, gatherBuf[6] = 6, gatherBuf[7] = 7,
total elapsed time = 0.000850
```

Conclusion

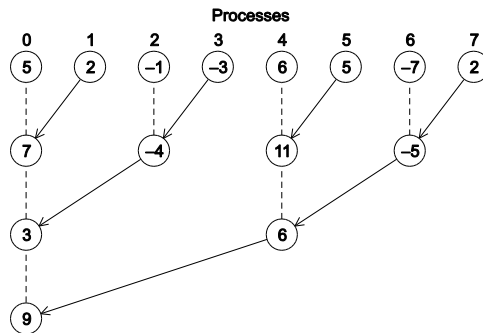
Point-to-Point communication

MPI_Send

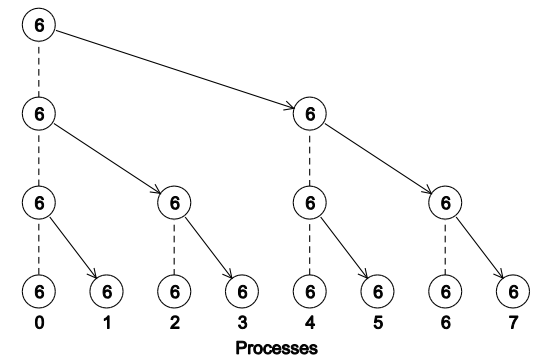
MPI_Receive



MPI_Reduce

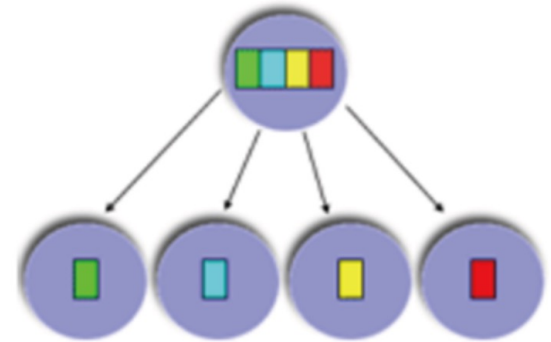
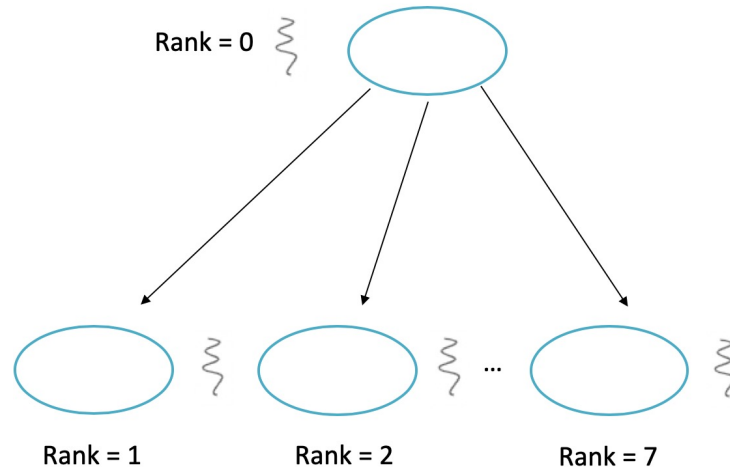


MPI_Bcast

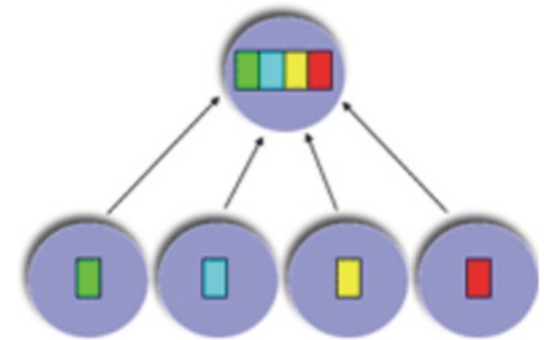
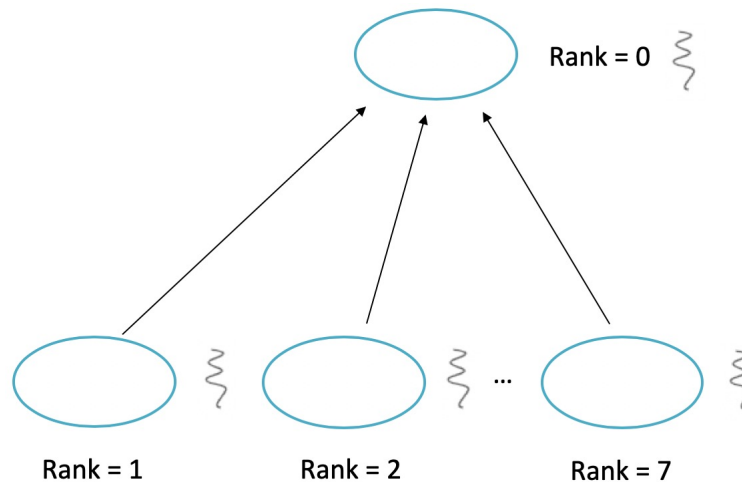


Conclusion

MPI_Scatter



MPI_Gather

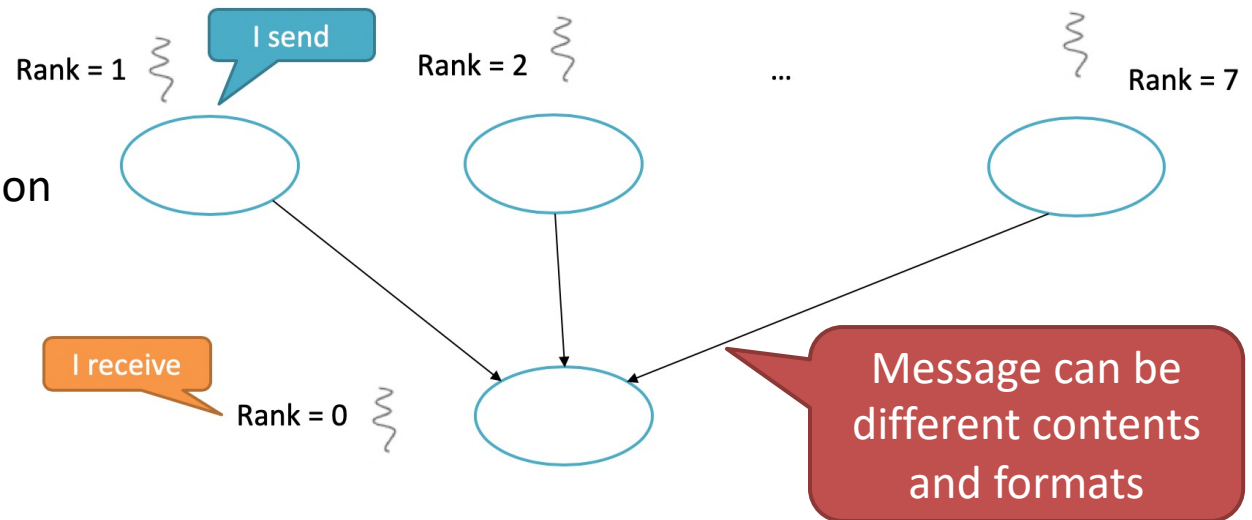


What is the Difference between Point-to-Point communication and MPI_Gather?

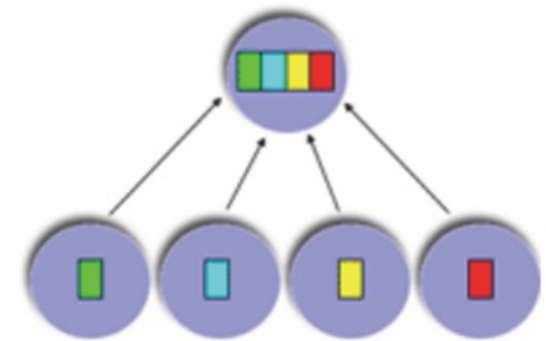
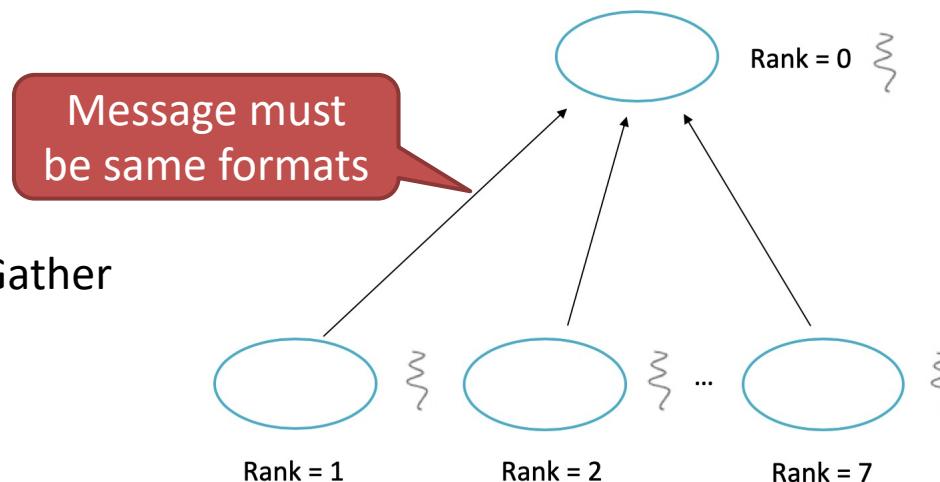
Point-to-Point communication

MPI_Send

MPI_Receive



MPI_Gather



Hint of Project 1

$$\int_0^1 \frac{4}{(1+x^2)} dx = \pi$$

```
#define NUMSTEPS 1000000
```

```
int main() {
    int i;
    double x, pi, sum = 0.0;
    struct timespec start, end;

    clock_gettime(CLOCK_MONOTONIC, &start);
    double step = 1.0/(double) NUMSTEPS;
    x = 0.5 * step;

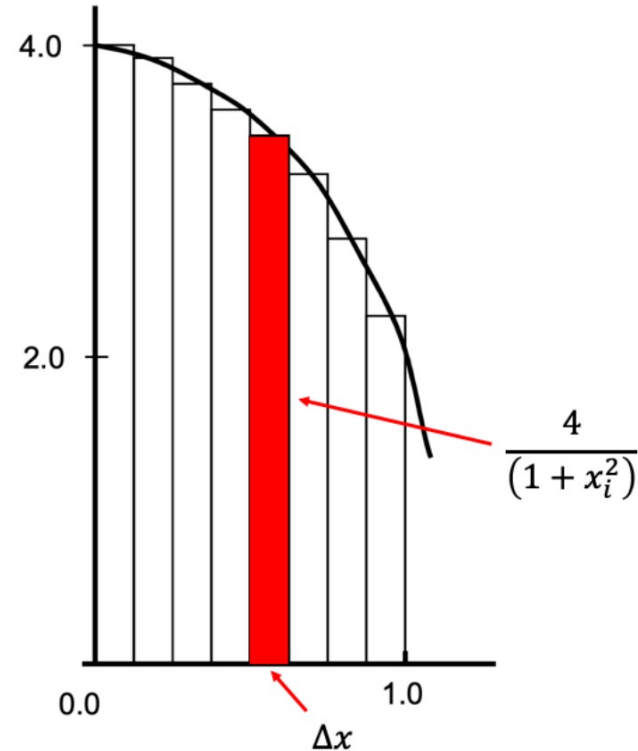
    for (i=0; i<= NUMSTEPS; i++){
        x+=step;
        sum += 4.0/(1.0+x*x);

    }
    pi = step * sum;
    clock_gettime(CLOCK_MONOTONIC, &end);
    u_int64_t diff = 1000000000L * (end.tv_sec - start.tv_sec) + end.tv_nsec -
start.tv_nsec;

    printf("PI is %.20f\n",pi);
    printf("elapsed time = %llu nanoseconds\n", (long long unsigned int) diff);

    return 0;
}
```

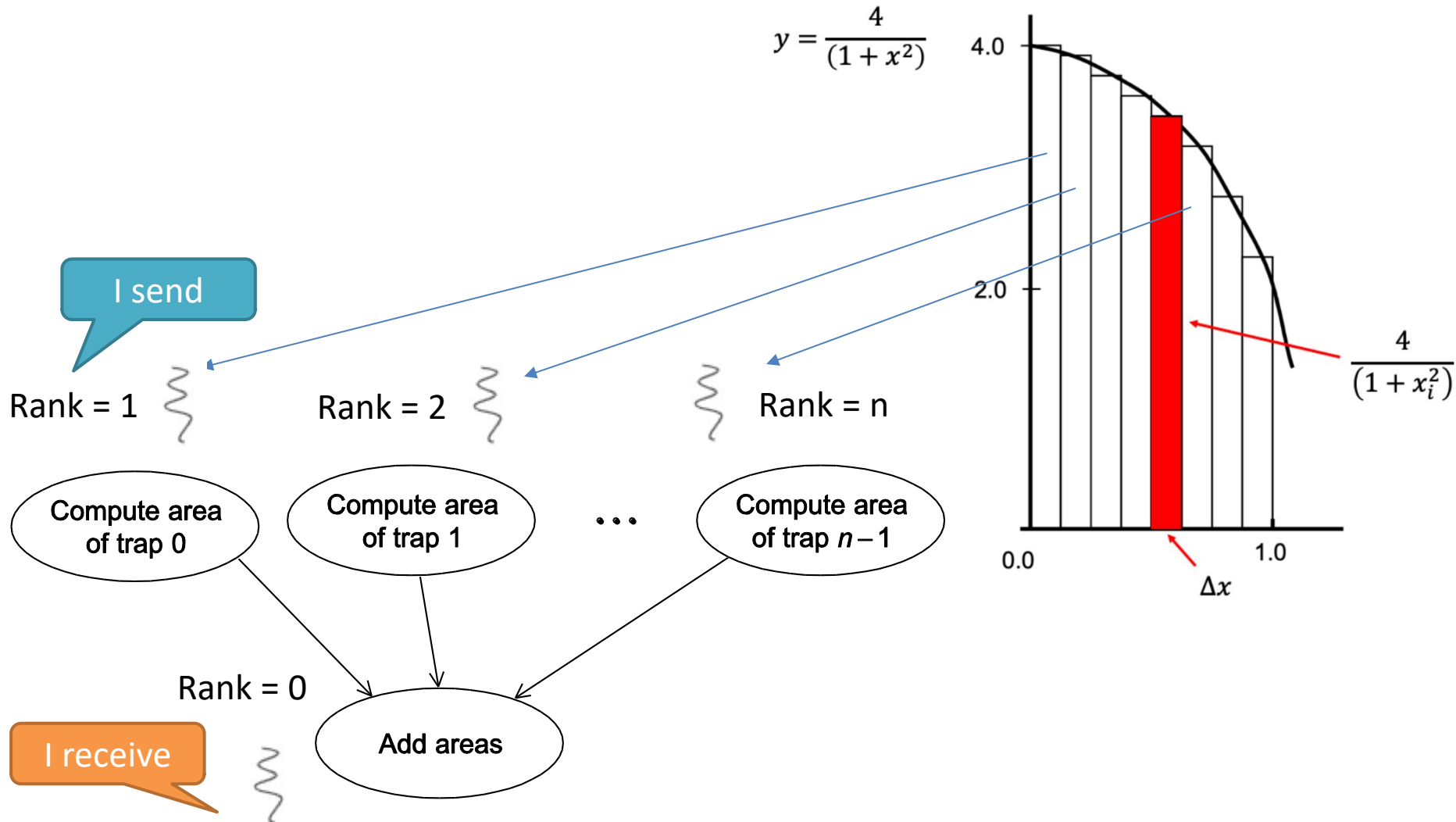
$$y = \frac{4}{(1+x^2)}$$



nputation

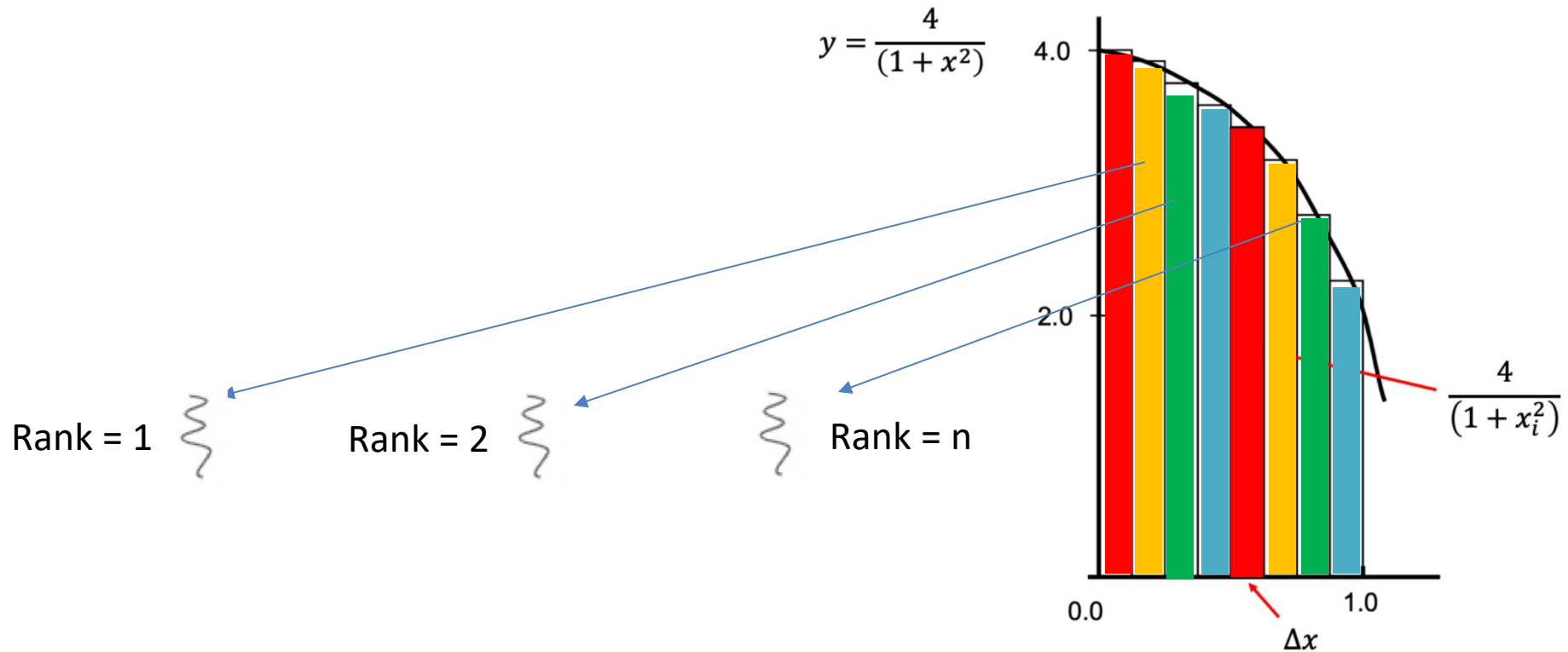
Hint of Project 1

$$\int_0^1 \frac{4}{(1+x^2)} dx = \pi$$



Hint of Project 1

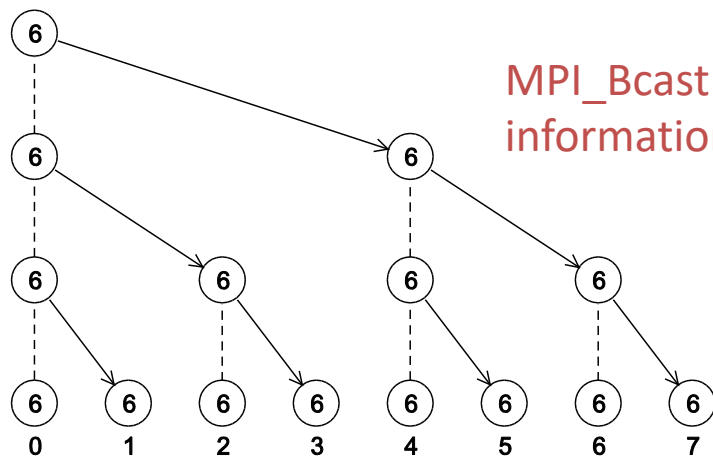
$$\int_0^1 \frac{4}{(1+x^2)} dx = \pi$$



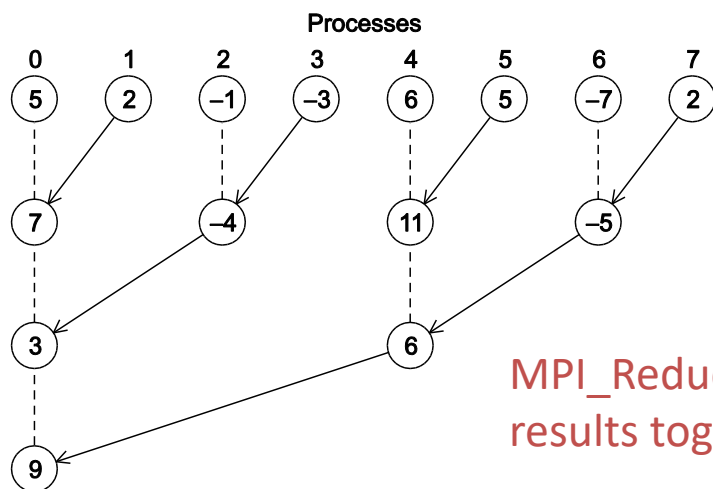
Hint of Project 1

$$\int_0^1 \frac{4}{(1+x^2)} dx = \pi$$

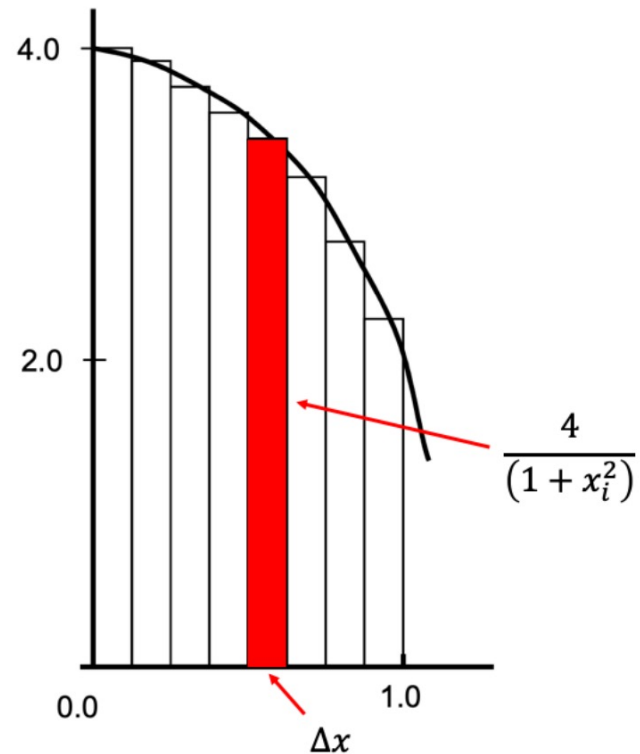
$$y = \frac{4}{(1+x^2)}$$



MPI_Bcast: broadcast
information to all threads



MPI_Reduce: sum up results together



Overview

