INTRO TO PARALLEL PROGRAMMING WITH MPI (CONCEPTUAL)

UPDATED 1/29/24

JANE HERRIMAN

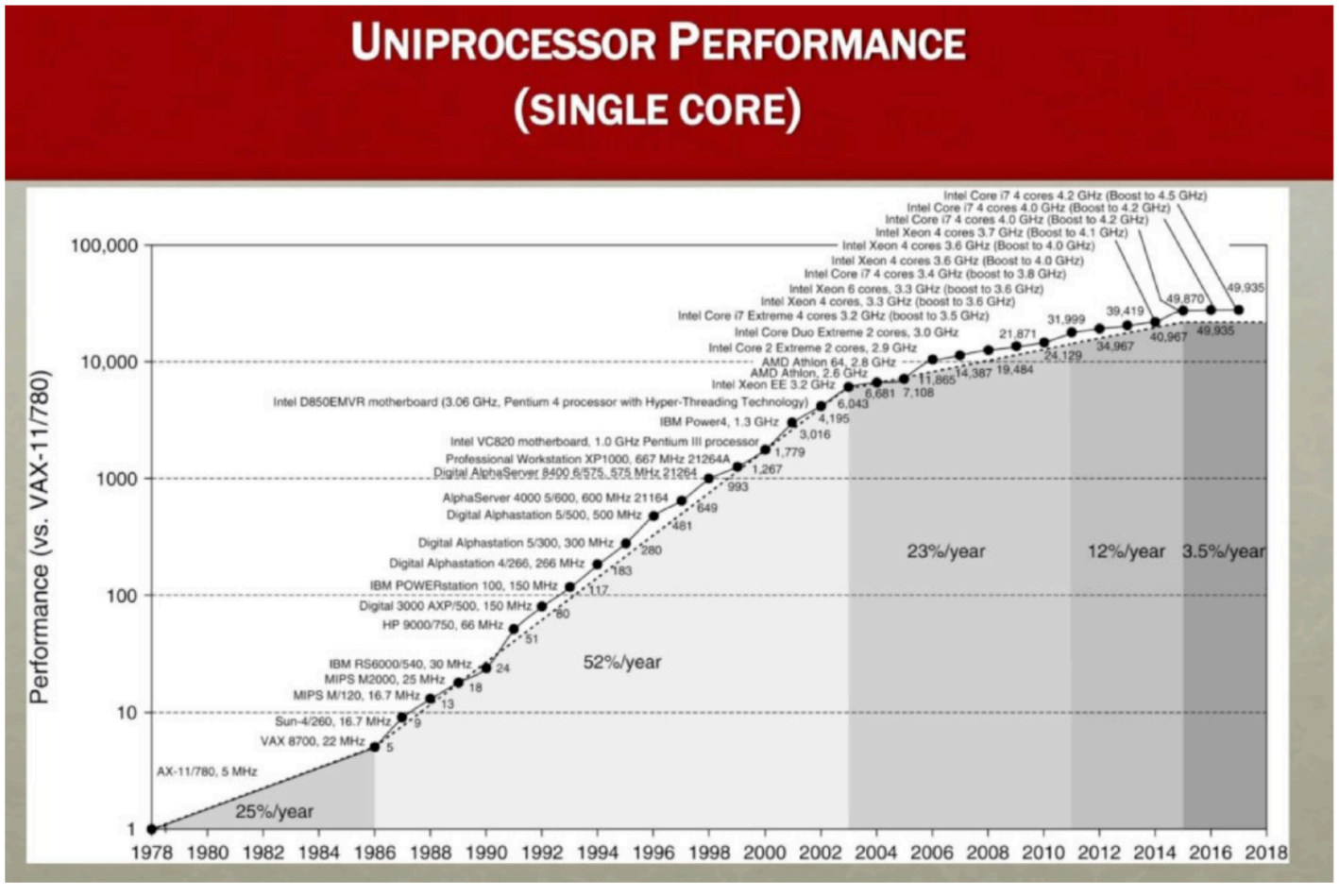
Prepared by LLNL under Contract DE-AC52-07NA27344.

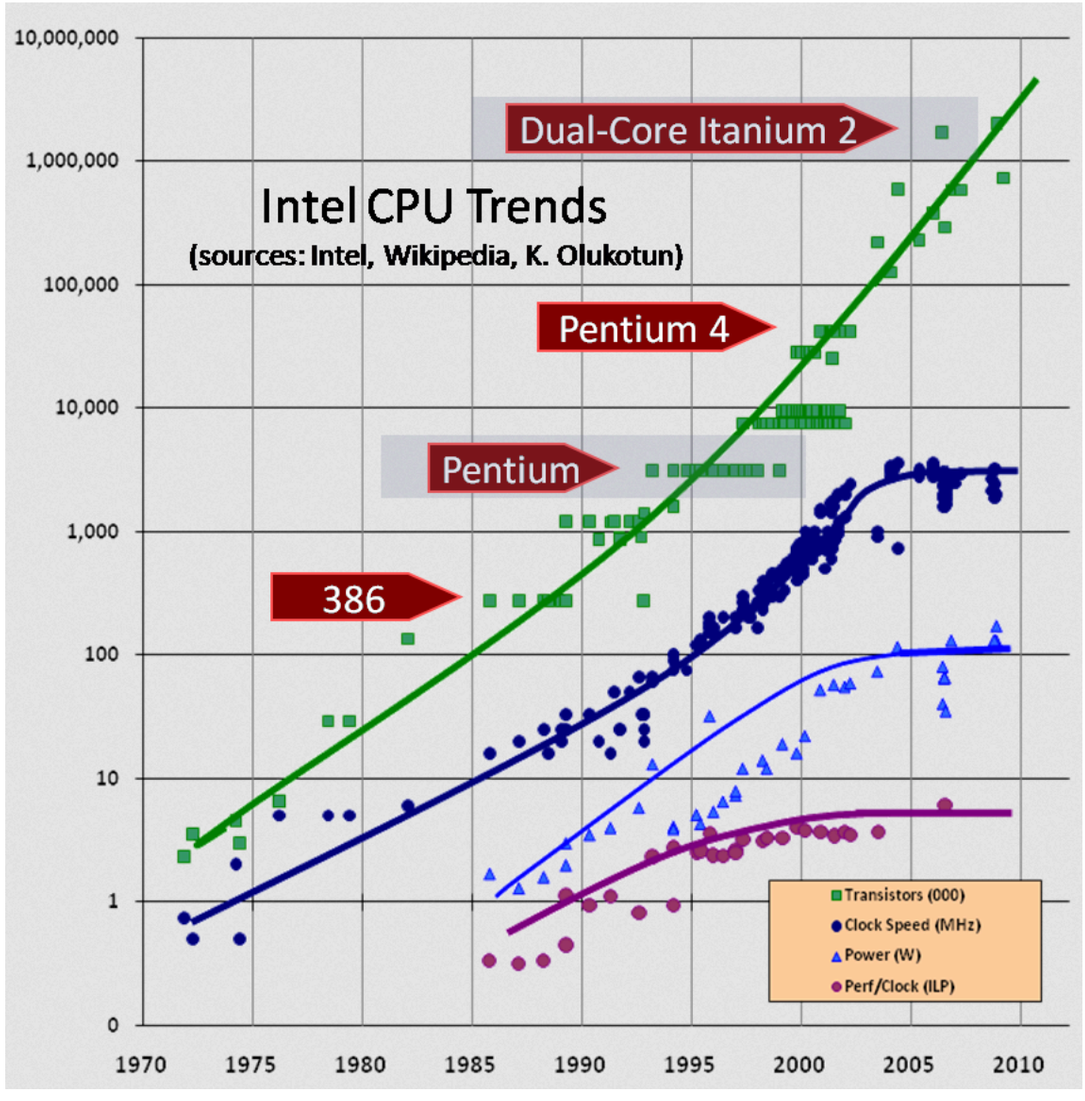
LLNL-PRES-859833

PARALLELIZATION: MOTIVATION

We want to get more done.

- Do things more quickly.
- Do more things at once.



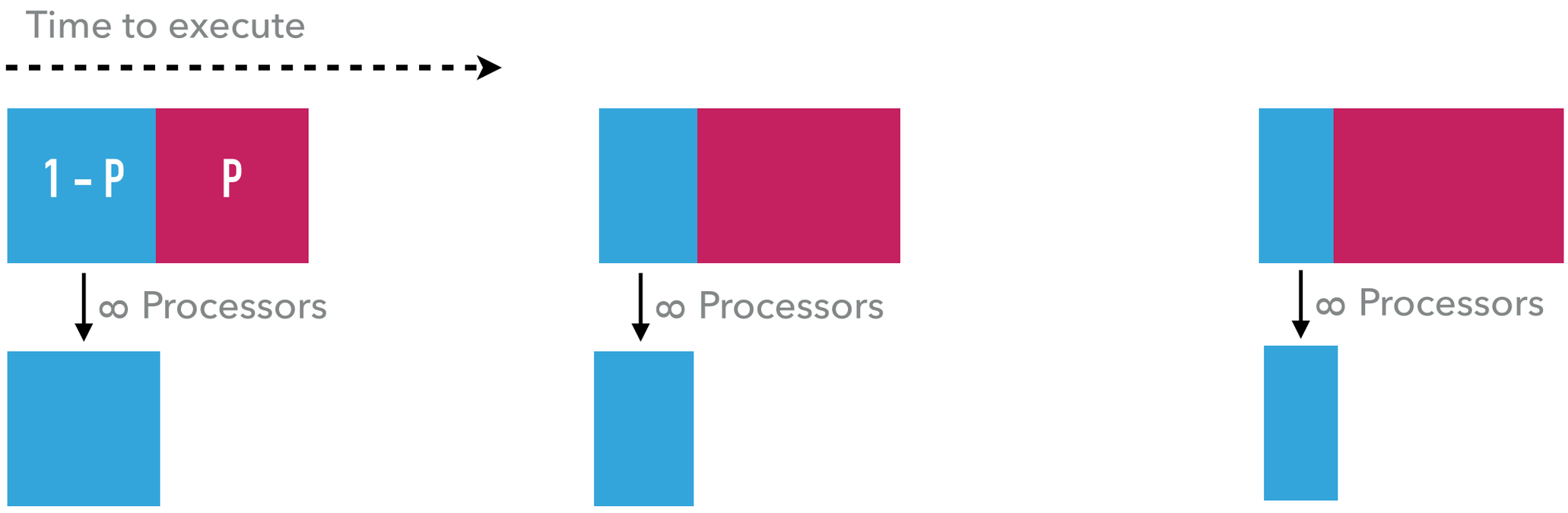


Sources:
http://www.gotw.ca/publications/concurrency-ddj.htm
https://www.nextbigfuture.com/2019/02/the-end-of-moores-law-in-detail-and-starting-a-new-golden-age.html

PARALLELIZATION: LIMITATIONS

- "With 1000 cores, my code should go 1000x as quickly!"
- Amdahl's law
  - Execution time for the serial parts of your code will dominate.
  - For example: SERIAL PARALLEL

$$S \leq \frac{1}{1-P}$$

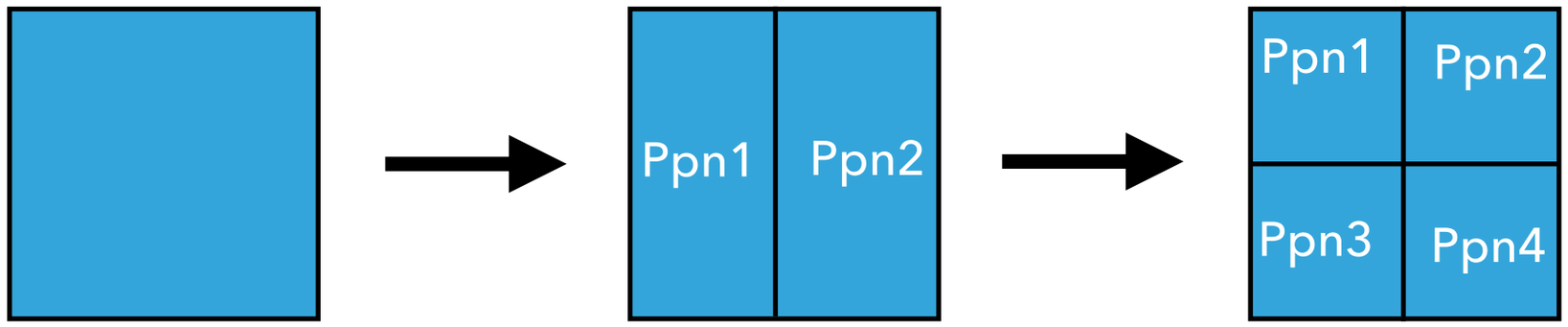


PARALLELIZATION: SCALING BEHAVIOR

Code “scales” when we can efficiently divide the work it does `N` ways and distribute it to `N` processors

Strong scaling:

Fixed problem size

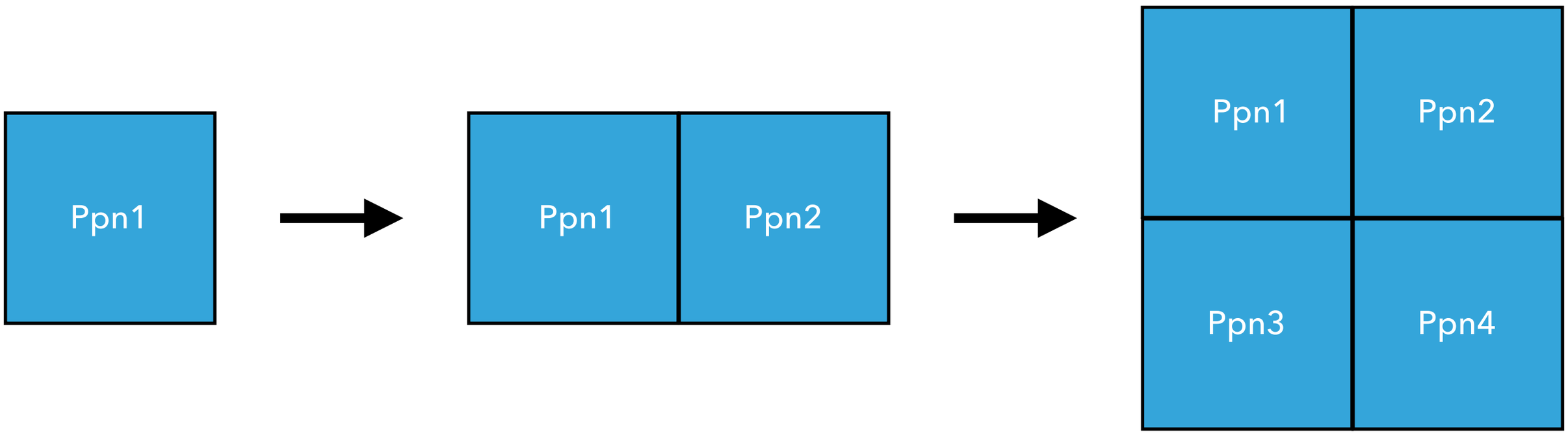


PARALLELIZATION: SCALING BEHAVIOR

Code “scales” when we can efficiently divide the work it does `N` ways and distribute it to `N` processors

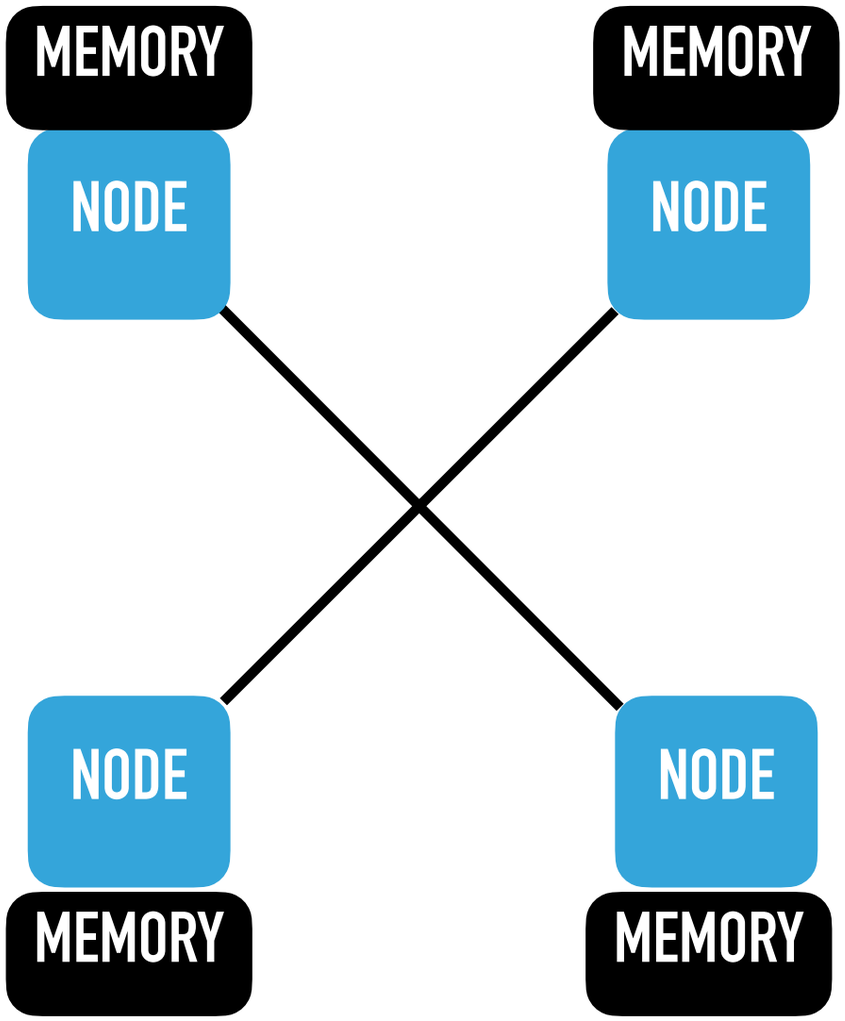
Weak scaling:

Problem size grows with N



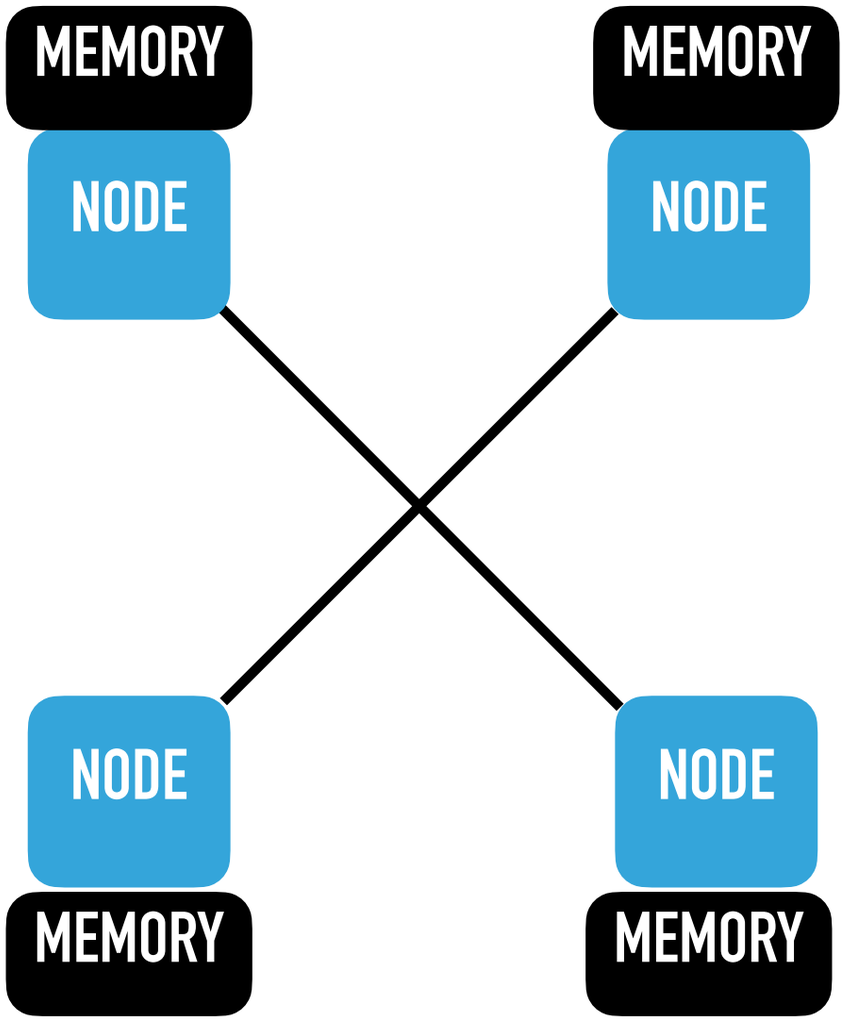
PARALLELIZATION: APPROACHES

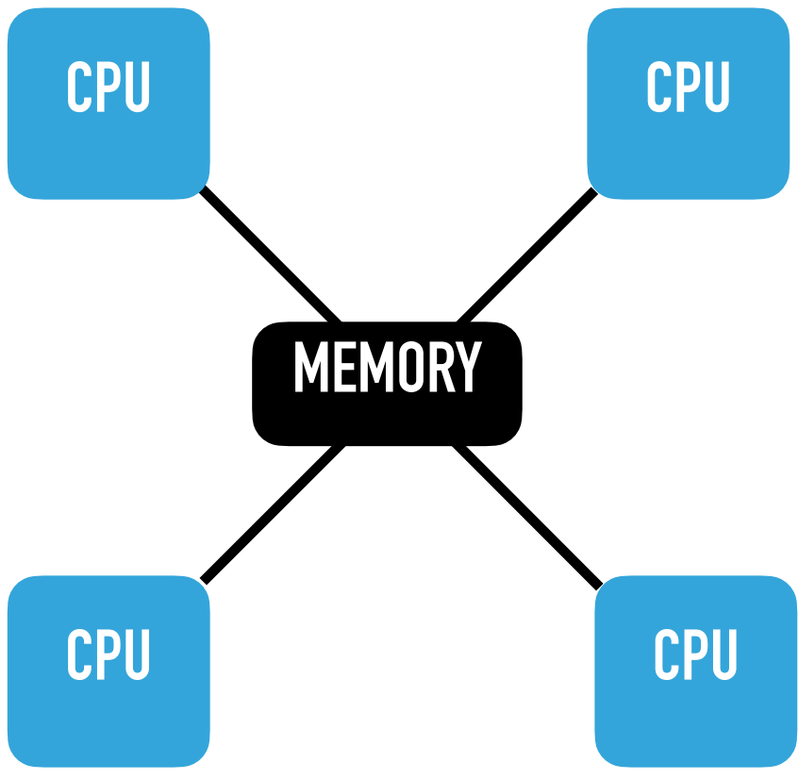
- Distributed or shared memory?
  - Distributed: create “tasks”/“processes”, usually use MPI



PARALLELIZATION: APPROACHES

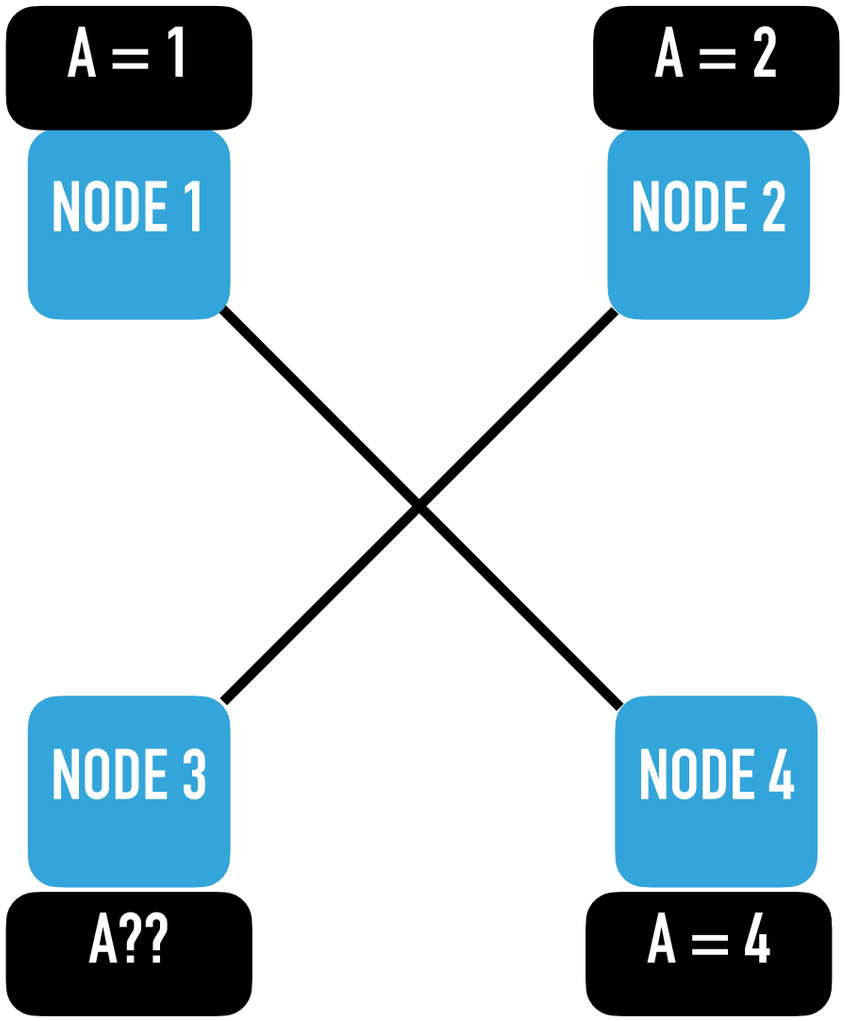
- Distributed or shared memory?
  - Distributed: create “tasks”/“processes”, usually use MPI
  - Shared: create “threads”, often use OpenMP or Pthreads

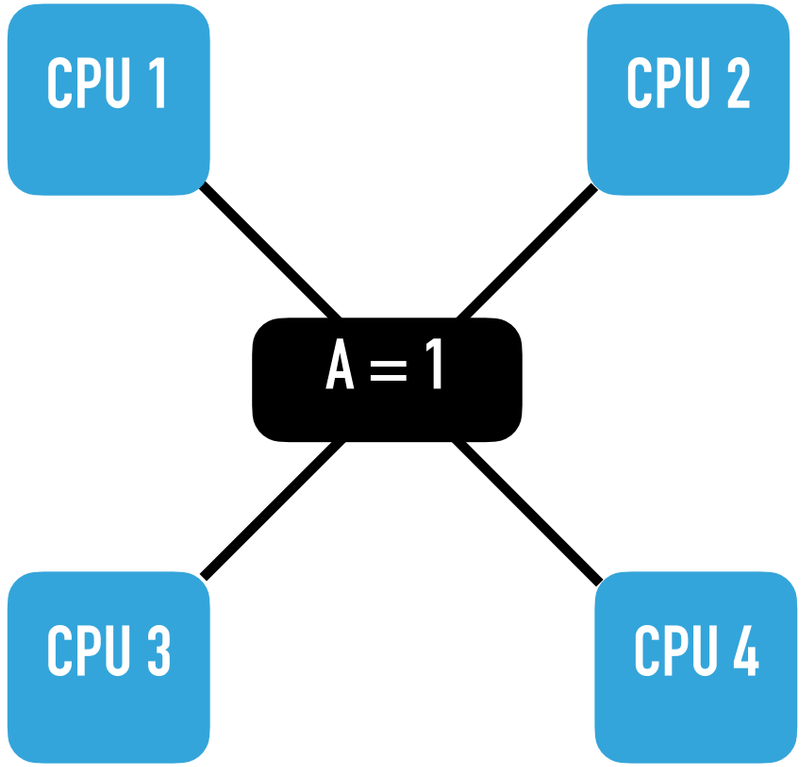




PARALLELIZATION: APPROACHES

- Distributed or shared memory?
  - Distributed: create “tasks”/“processes”, usually use MPI
  - Shared: create “threads”, often use OpenMP or Pthreads





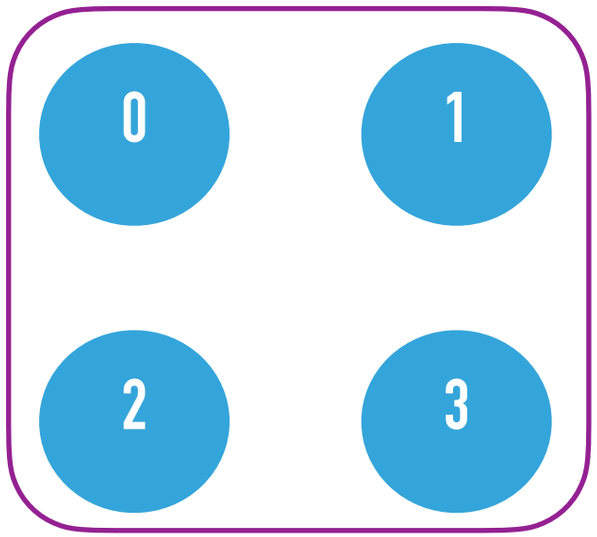
WHAT IS MPI?

- Message Passing Interface
- A standard, NOT a piece of software or a language
- Multiple MPI libraries & languages support MPI
    - libraries: MVAPICH & OpenMPI
    - languages: C, Fortran, Julia, Python, etc.

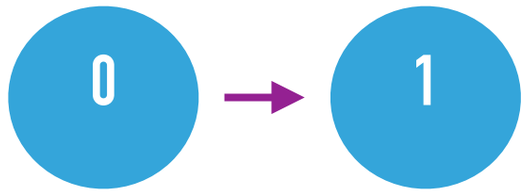
- Standard/protocol that defines a common set of syntax to be used for performing a common set of routines/operations

MPI CONCEPTS

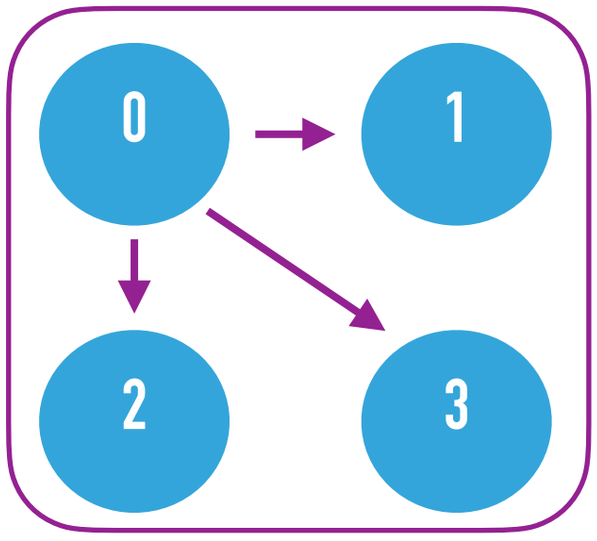
Task



Communicator



Point to point
communication routines



Collective
communication routines

HOW TO WRITE MPI PROGRAM: HELLO WORLD

C

```c
#include <stdio.h>

#include "mpi.h"


main(int argc, char** argv){

int my_task_num, comm_size;

MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD,
&comm_size);

MPI_Comm_rank(MPI_COMM_WORLD,
&my_task_num);


printf("Hello from task %d of %d.\n",
my_task_num, comm_size);

MPI_Finalize();

}
```

Julia

```julia
using MPI

MPI.Init()

comm_size =
MPI.Comm_size(MPI.COMM_WORLD)

my_task_num =
MPI.Comm_rank(MPI.COMM_WORLD)

println("Hello from task $(my_task_num) of $
(comm_size).\n")

MPI.Finalize()
```

HOW TO WRITE MPI PROGRAM: HELLO WORLD

C

```c
#include <stdio.h>

#include "mpi.h"

main(int argc, char** argv){

int my_task_num, comm_size;

MPI_Init(&a
MPI_Comm_
&comm_size

MPI_Comm_
&my_task_n

printf("Hello
my_task_nu

MPI_Finalize();

}
```

Julia

```julia
using MPI

MPI.Init()

MPI.Comm_size(MPI.COMM_WORLD)

of $

MPI.Finalize()
```

```
janeh@quartz380:~/practice_MPI/HPC_CEA$ mpicc hello.c -o c_hello
janeh@quartz380:~/practice_MPI/HPC_CEA$ srun -N1 -n8 -ppdebug c_hello
srun: job 5483668 queued and waiting for resources
srun: job 5483668 has been allocated resources
Hello from task 0 of 8.
Hello from task 3 of 8.
Hello from task 2 of 8.
Hello from task 1 of 8.
Hello from task 5 of 8.
Hello from task 7 of 8.
Hello from task 6 of 8.
Hello from task 4 of 8.
```

EXERCISE 1: REORDER TO GET MPI HELLO WORLD

A) MPI_Init(&argc, &argv);

B)
```
main(int argc, char** argv){

int my_task_num, comm_size;
```

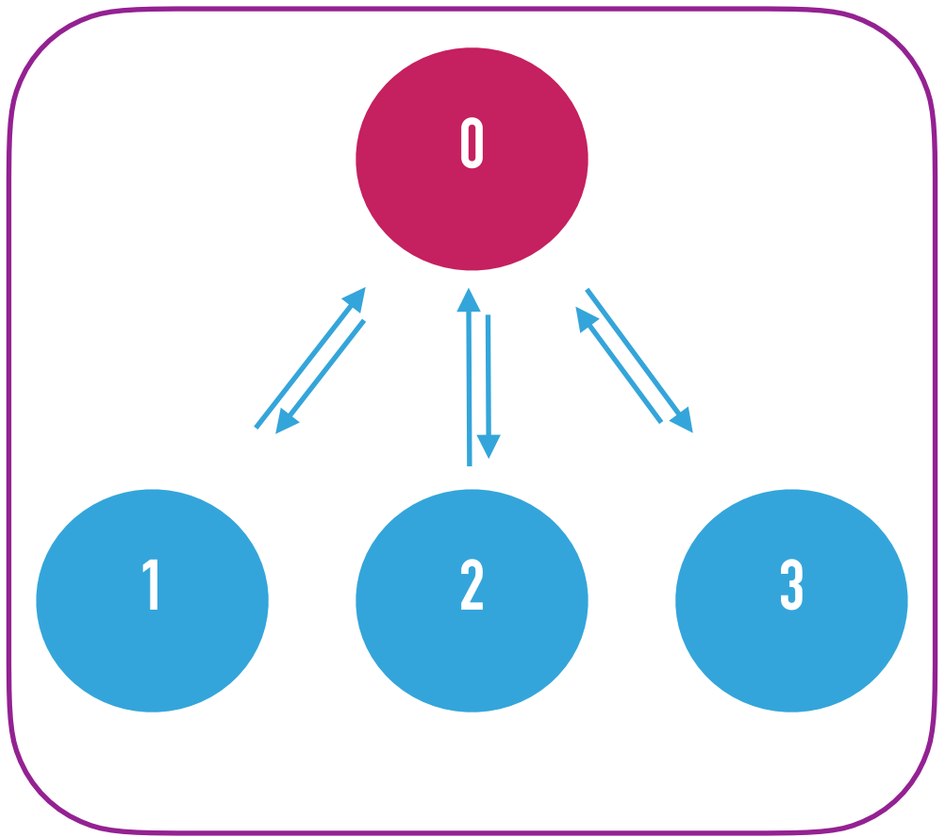
C) MPI_Finalize();

D)
```
MPI_Comm_size(MPI_COMM_WORLD, &comm_size);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
```

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

#include "mpi.h"
```

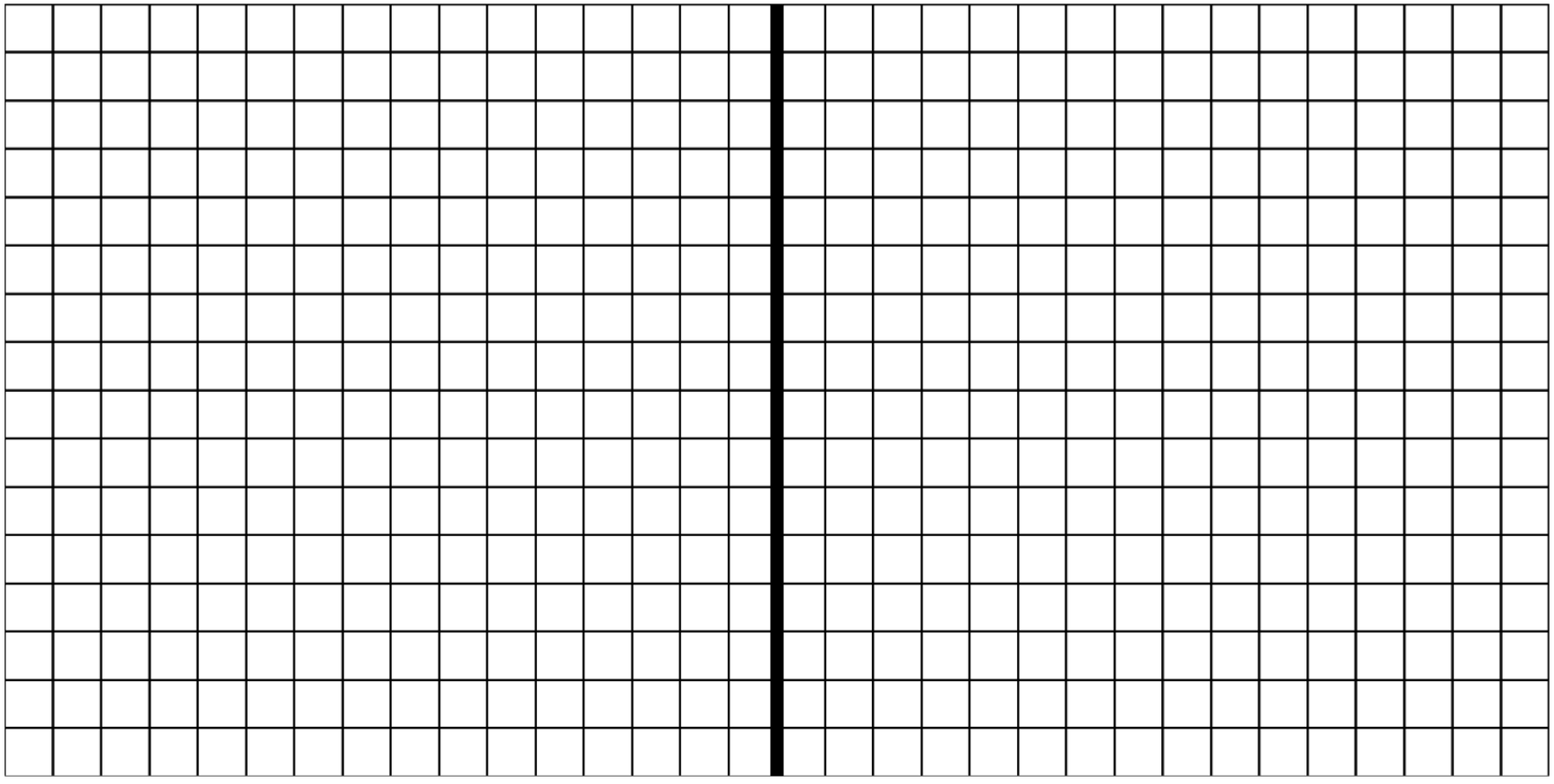
F) printf("Hello from task %d of %d.\n", my_task_num, comm_size);

MASTER-WORKER PARADIGM



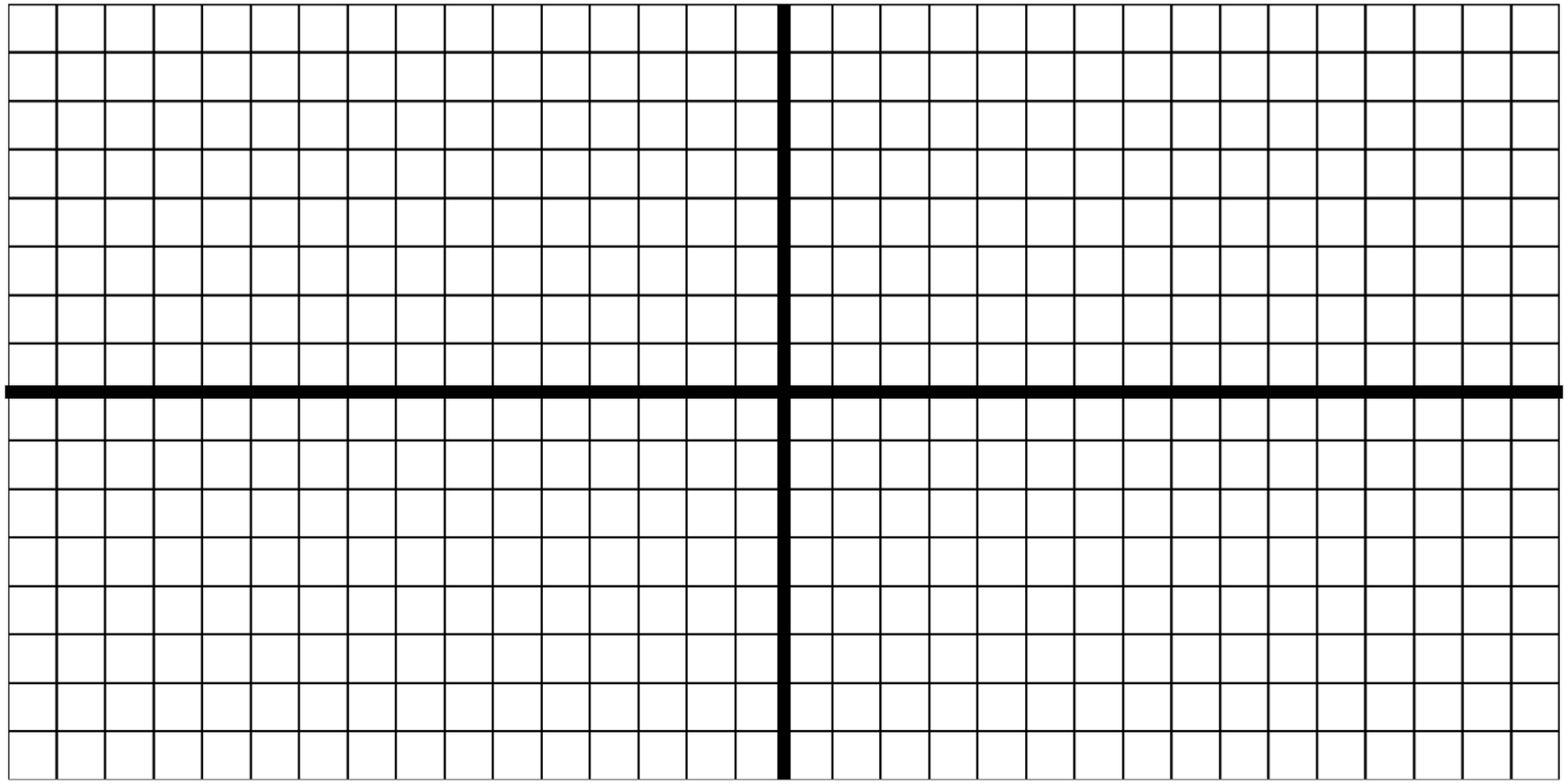
MASTER-WORKER PARADIGM: DOMAIN DECOMPOSITIONS

-n 2

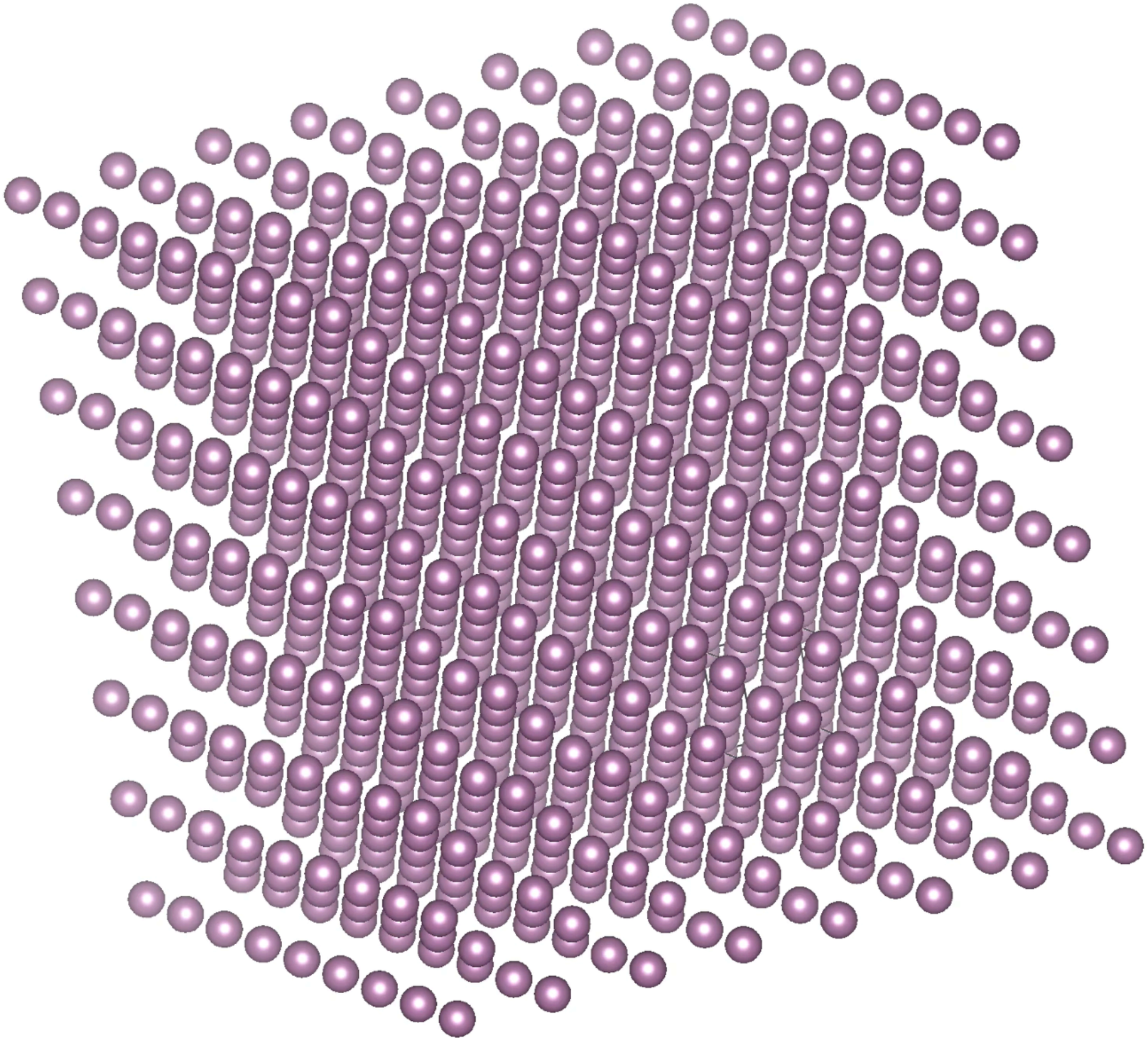


MASTER-WORKER PARADIGM: DOMAIN DECOMPOSITIONS

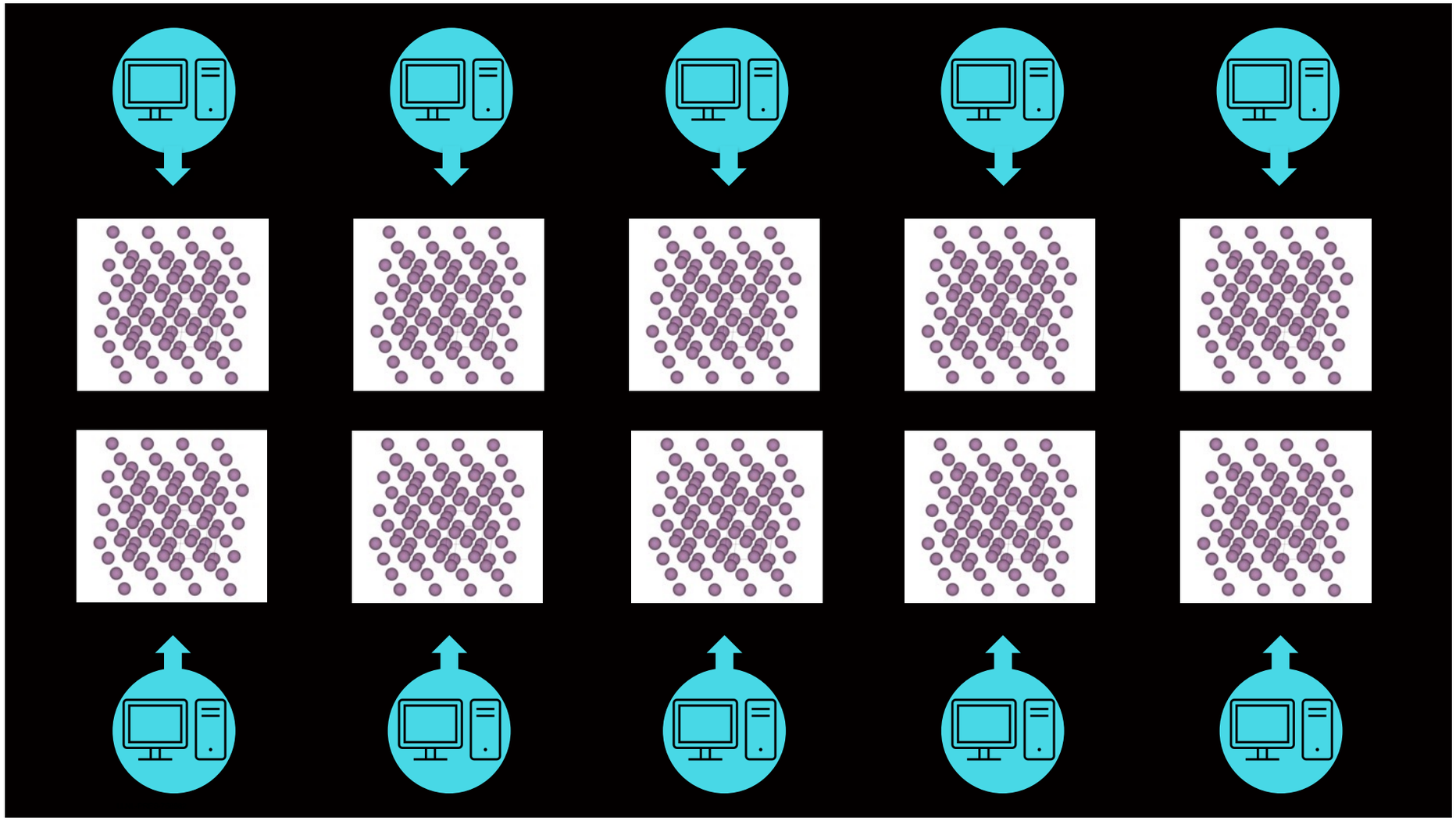
-n 4



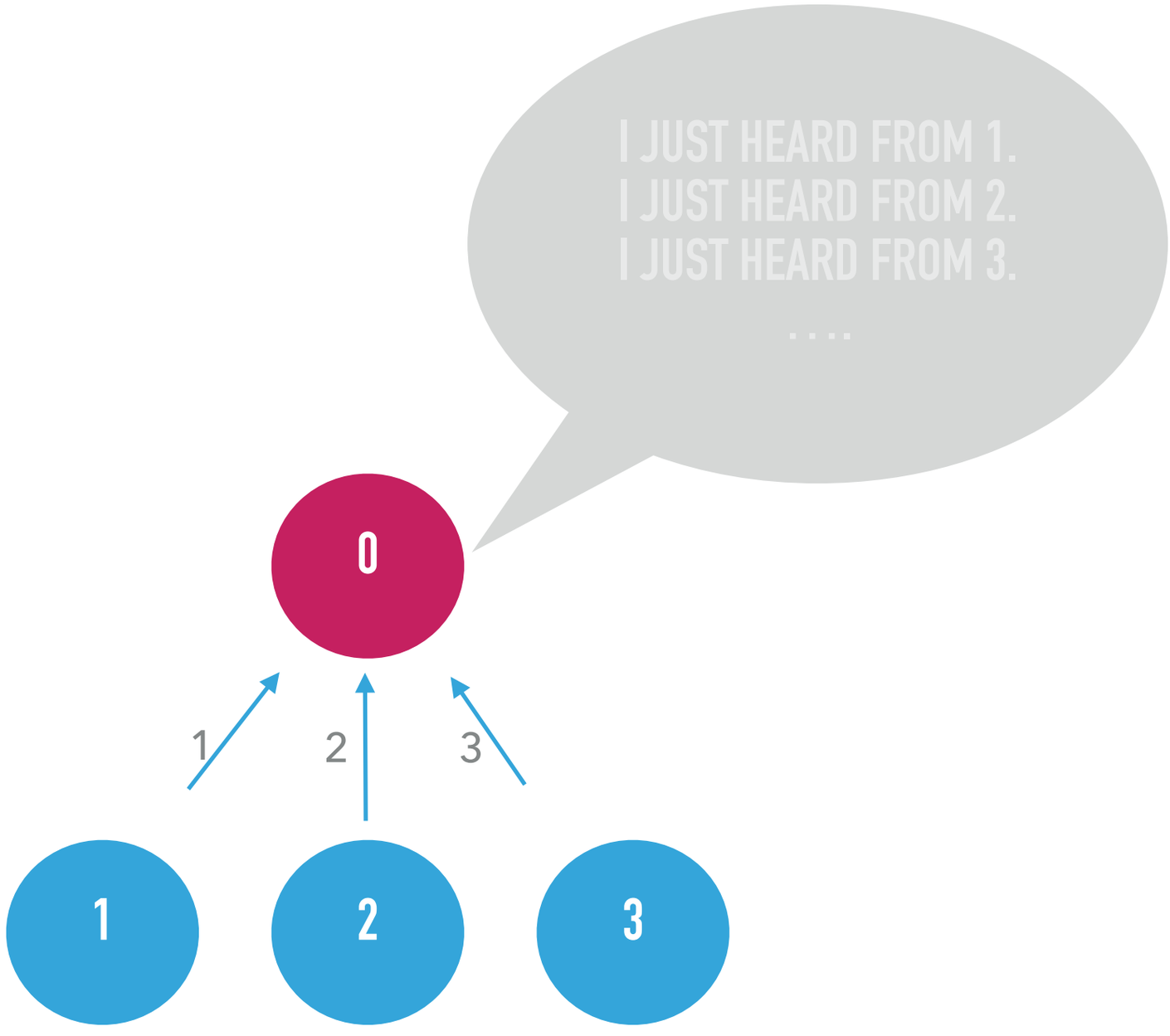
MASTER-WORKER PARADIGM: DOMAIN DECOMPOSITIONS



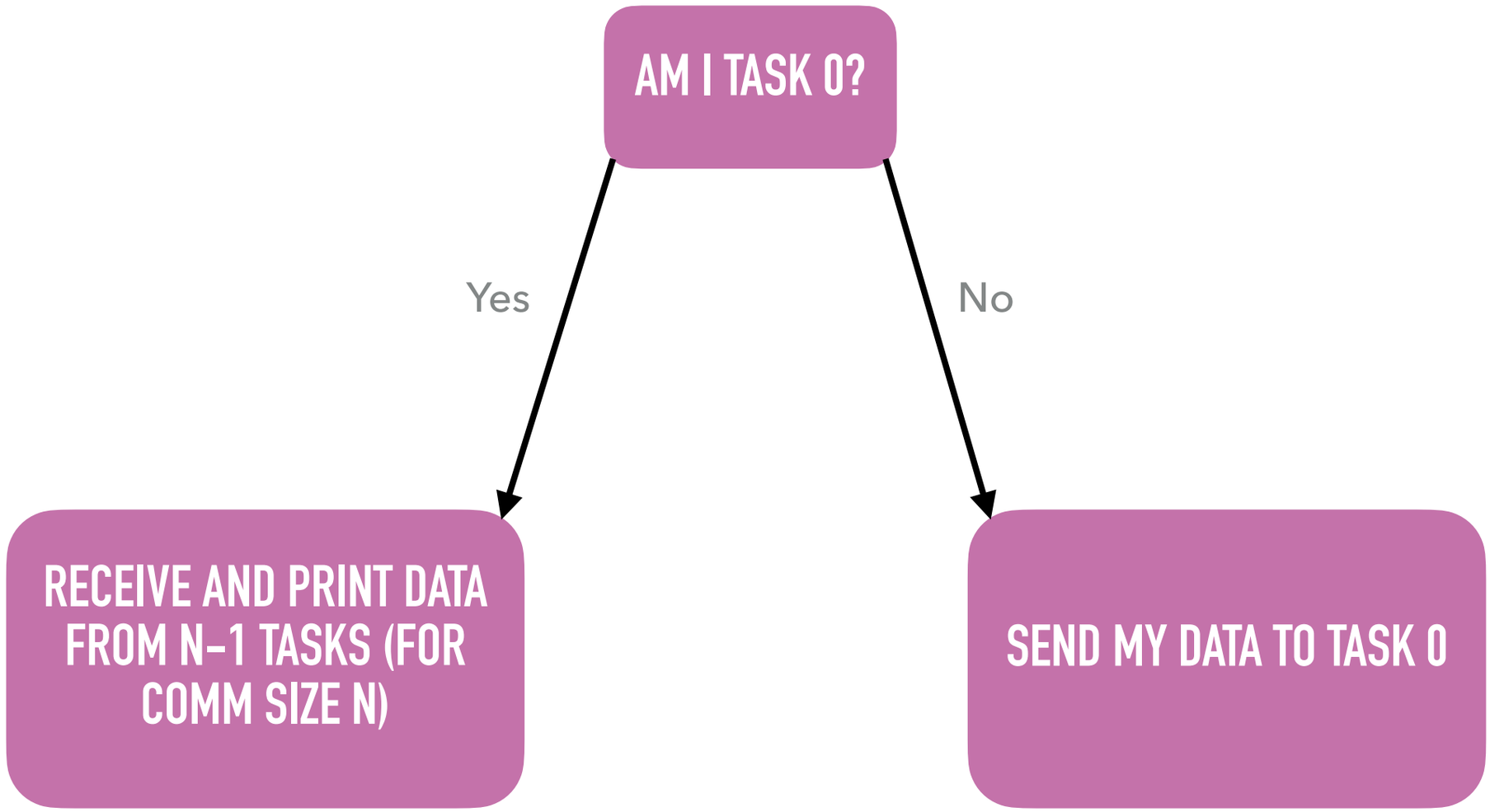
MASTER-WORKER PARADIGM: DOMAIN DECOMPOSITIONS



MASTER WORKER PARADIGM: SENDING HELLOS



MASTER WORKER PARADIGM: SENDING HELLOS



Why do you think that task 0 often coordinates what the workers do?

MASTER WORKER PARADIGM: SENDING HELLOS

Pseudo code

```
# Declare you want to use standard libraries and MPI

Initialize MPI

What's my task number?

How big is my communicator?

If I'm task 0:

    For all other tasks in my communicator

        Receive data from another task

        Print to notify that I received data from that other task

If I'm not task 0:

    Send my data to task 0.

Finalize MPI
```

MASTER WORKER PARADIGM: SENDING HELLOS

```
#include <stdio.h>

#include "mpi.h"

main(int argc, char** argv){

int my_task_num, comm_size;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

MPI_Comm_size(MPI_COMM_WORLD, &comm_size);

if (my_task_num == 0) {

    int i, fromwho;

    for (i=1; i < comm_size; i++) {

        MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        printf("I, 0, just heard from task %d.\n", fromwho); }

}else{

    MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }

MPI_Finalize(); }
```

MASTER WORKER PARADIGM: SENDING HELLOS

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

From: https://mpitutorial.com/tutorials/mpi-send-and-receive/

MASTER WORKER PARADIGM: SENDING HELLOS

```
#include <stdio.h>

#include "mpi.h"

main(int argc, char** argv){

int my_task_num, comm_size;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

MPI_Comm_size(MPI_COMM_WORLD, &comm_size);

if (my_task_num == 0) {

    int i, fromwho;

    for (i=1; i < comm_size; i++) {

        MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        printf("I, 0, just heard from task %d.\n", fromwho); }

}else{

    MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }

MPI_Finalize(); }
```

MASTER WORKER PARADIGM: SENDING HELLOS

```
janeh@quartz2306:~/practice_MPI/HPC_CEA$ srun -n10 -ppdebug master-work-send
I, 0, just heard from task 5.
I, 0, just heard from task 3.
I, 0, just heard from task 7.
I, 0, just heard from task 4.
I, 0, just heard from task 2.
I, 0, just heard from task 9.
I, 0, just heard from task 6.
I, 0, just heard from task 1.
I, 0, just heard from task 8.
```

EXERCISE 2: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that a different task (other than 0) is the master processor that receives data from the other tasks.

```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, comm_size;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == 0) {
9       int i, fromwho;
10      for (i=1; i < comm_size; i++) {
11          MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12          printf("I, 0, just heard from task %d.\n", fromwho); }
13  }else{
14      MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }
15  MPI_Finalize(); }
```

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

EXERCISE 2: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that a different task (other than 0) is the master processor that receives data from the other tasks.

```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, comm_size;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == 0) {
9       int i, fromwho;
10      for (i=1; i < comm_size; i++) {
11          MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12          printf("I, 0, just heard from task %d.\n", fromwho); }
13  }else{
14      MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }
15  MPI_Finalize(); }
```

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

EXERCISE 3: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that the "hellos" are received in order (use task 0 as the master).

```
I, 0, just heard from task 1.
I, 0, just heard from task 2.
I, 0, just heard from task 3.
I, 0, just heard from task 4.
I, 0, just heard from task 5.
I, 0, just heard from task 6.
I, 0, just heard from task 7.
I, 0, just heard from task 8.
I, 0, just heard from task 9.
```

```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, comm_size;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == 0) {
9       int i, fromwho;
10      for (i=1; i < comm_size; i++) {
11          MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12          printf("I, 0, just heard from task %d.\n", fromwho); }
13  }else{
14      MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }
15  MPI_Finalize(); }
```

EXERCISE 3: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that the "hellos" are received in order (use task 0 as the master).

```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, comm_size;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == 0) {
9       int i, fromwho;
10      for (i=1; i < comm_size; i++) {
11          MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12          printf("I, 0, just heard from task %d.\n", fromwho); }
13  }else{
14      MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }
15  MPI_Finalize(); }
```

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

EXERCISE 3: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that the "hellos" are received in order (use task 0 as the master).

```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, comm_size;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == 0) {
9       int i, fromwho;
10      for (i=1; i < comm_size; i++) {
11          MPI_Recv(&fromwho, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12          printf("I, 0, just heard from task %d.\n", fromwho); }
13  }else{
14      MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }
15  MPI_Finalize(); }
```

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

EXERCISE 3: WHICH LINES NEED TO BE UPDATED?

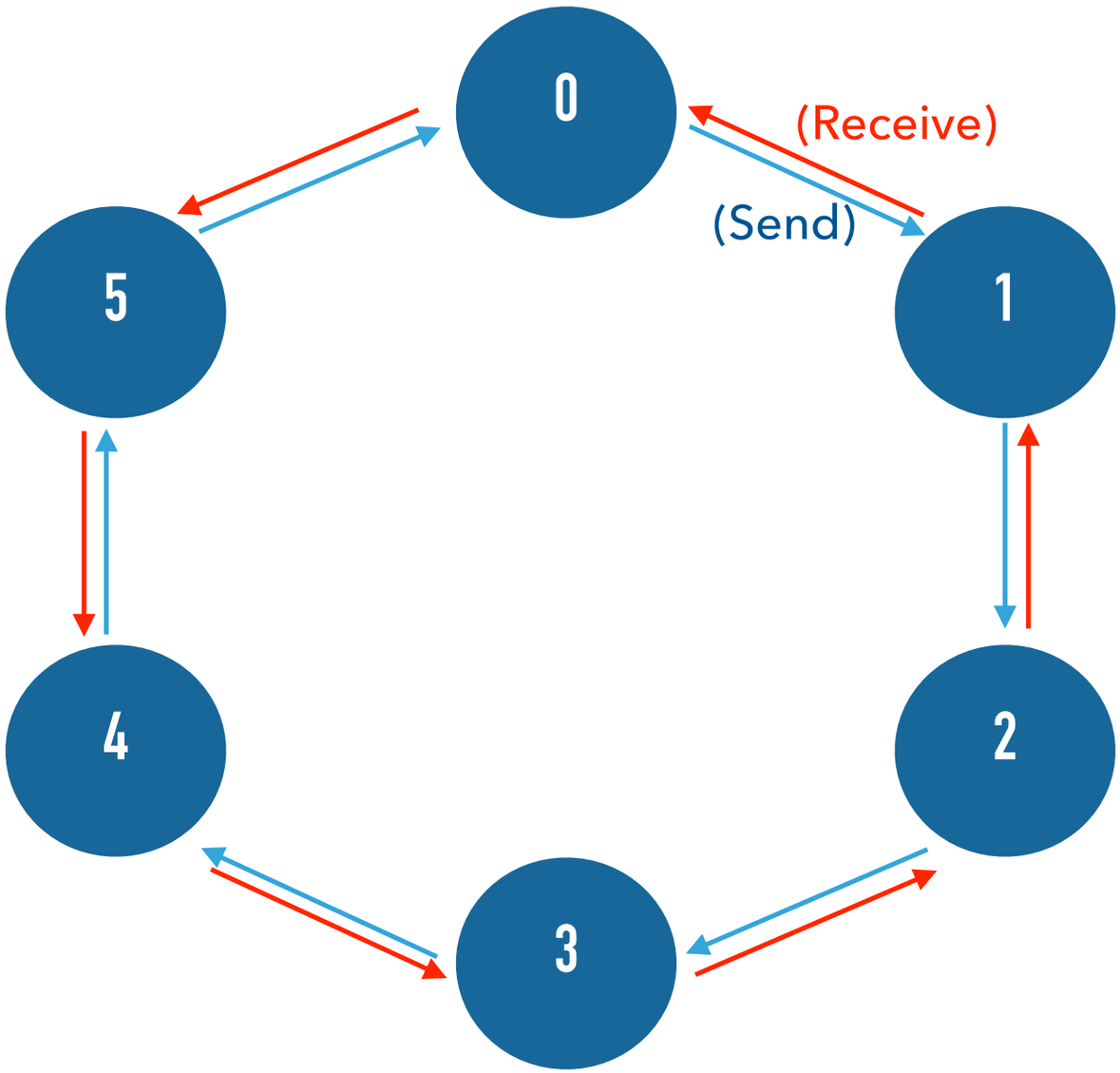
Goal: Change the code so that the "hellos" are received in order (use task 0 as the master).

```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, comm_size;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == 0) {
9       int i, fromwho;
10      for (i=1; i < comm_size; i++) {
11          MPI_Recv(&fromwho, 1, MPI_INT, i, i, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12          printf("I, 0, just heard from task %d.\n", fromwho); }
13  }else{
14      MPI_Send(&my_task_num, 1, MPI_INT, 0, my_task_num, MPI_COMM_WORLD); }
15  MPI_Finalize(); }
```

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

ROUND ROBIN SEND



ROUND ROBIN SEND

```
#include <stdio.h>

#include "mpi.h"

main(int argc, char** argv){

int my_task_num, neighbor_info, comm_size, neighbordown, neighborup;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

MPI_Comm_size(MPI_COMM_WORLD, &comm_size);

if (my_task_num == (comm_size - 1)) {

        neighborup = 0;

}else{

        neighborup = my_task_num + 1;}

MPI_Send(&my_task_num, 1, MPI_INT, neighborup, my_task_num, MPI_COMM_WORLD);

MPI_Recv(&neighbor_info, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

printf("I am %d and just received data from %d\n", my_task_num, neighbor_info);

MPI_Finalize();
```

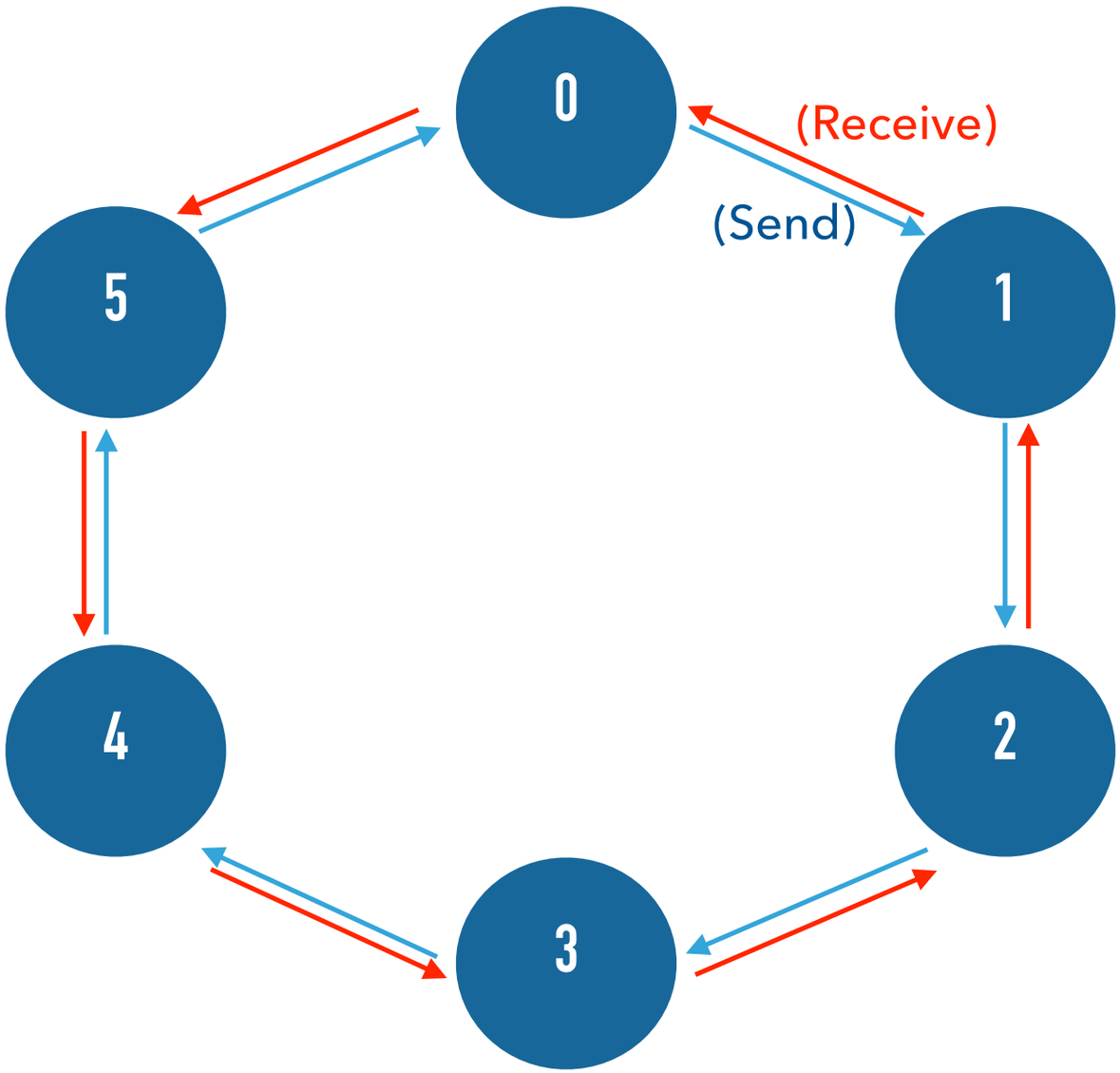
ROUND ROBIN SEND

```
if (my_task_num == (comm_size - 1)) {
        neighborup = 0;
}else{
        neighborup = my_task_num + 1;}

MPI_Send(&my_task_num, 1, MPI_INT, neighborup, my_task_num, MPI_COMM_WORLD);

MPI_Recv(&neighbor_info, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

printf("I am %d and just received data from %d\n", my_task_num, neighbor_info);
```

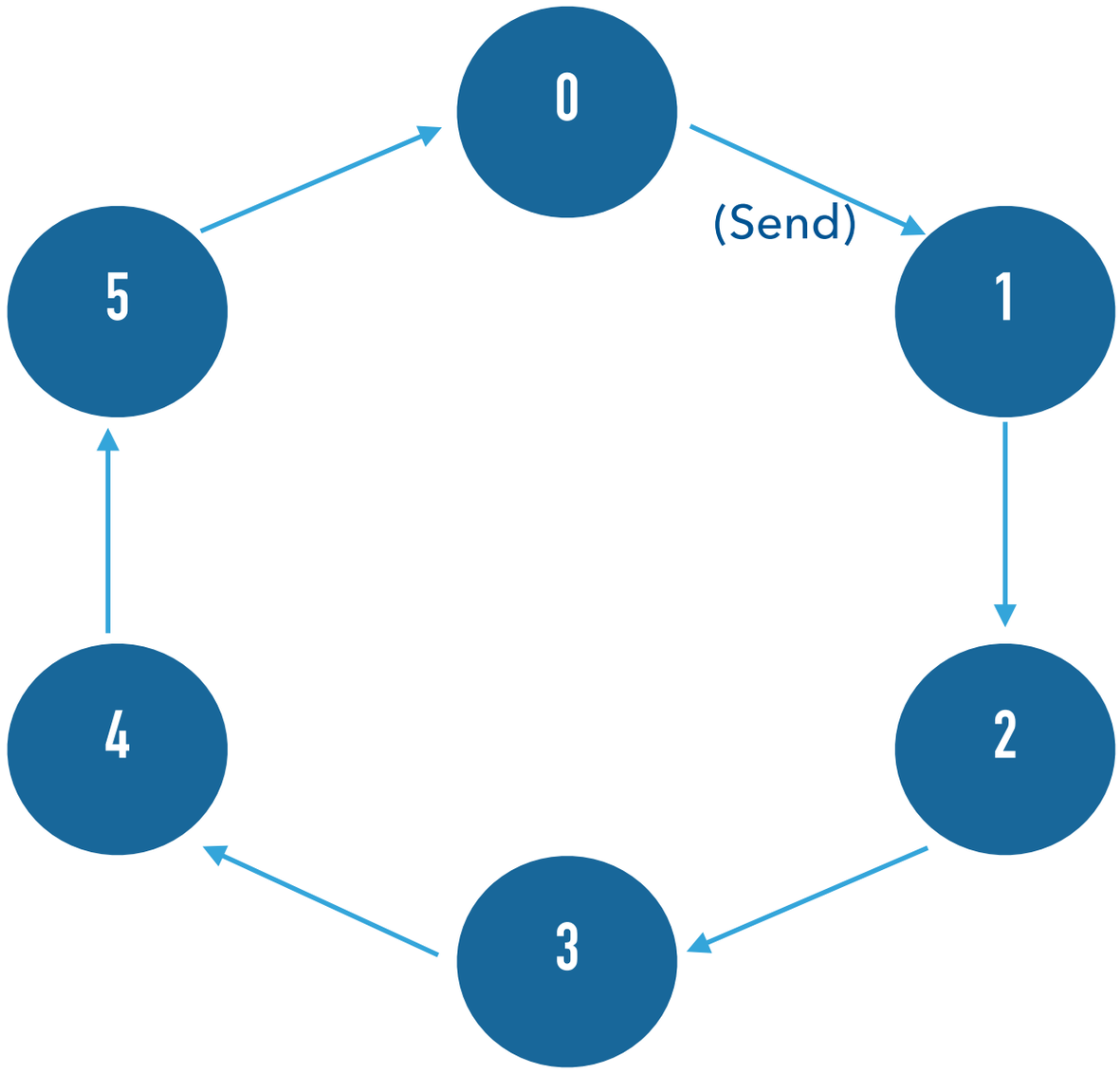
ROUND ROBIN SEND

```
janeh@quartz2306:~/practice_MPI/HPC_CEA$ srun -n10 -ppdebug round_robin
I am 0 and just received data from 9
I am 5 and just received data from 4
I am 6 and just received data from 5
I am 7 and just received data from 6
I am 9 and just received data from 8
I am 3 and just received data from 2
I am 4 and just received data from 3
I am 1 and just received data from 0
I am 2 and just received data from 1
I am 8 and just received data from 7
```

ROUND ROBIN SEND



PROBLEMS THAT COME UP: MESSAGE BLOCKING



“Non-blocking”

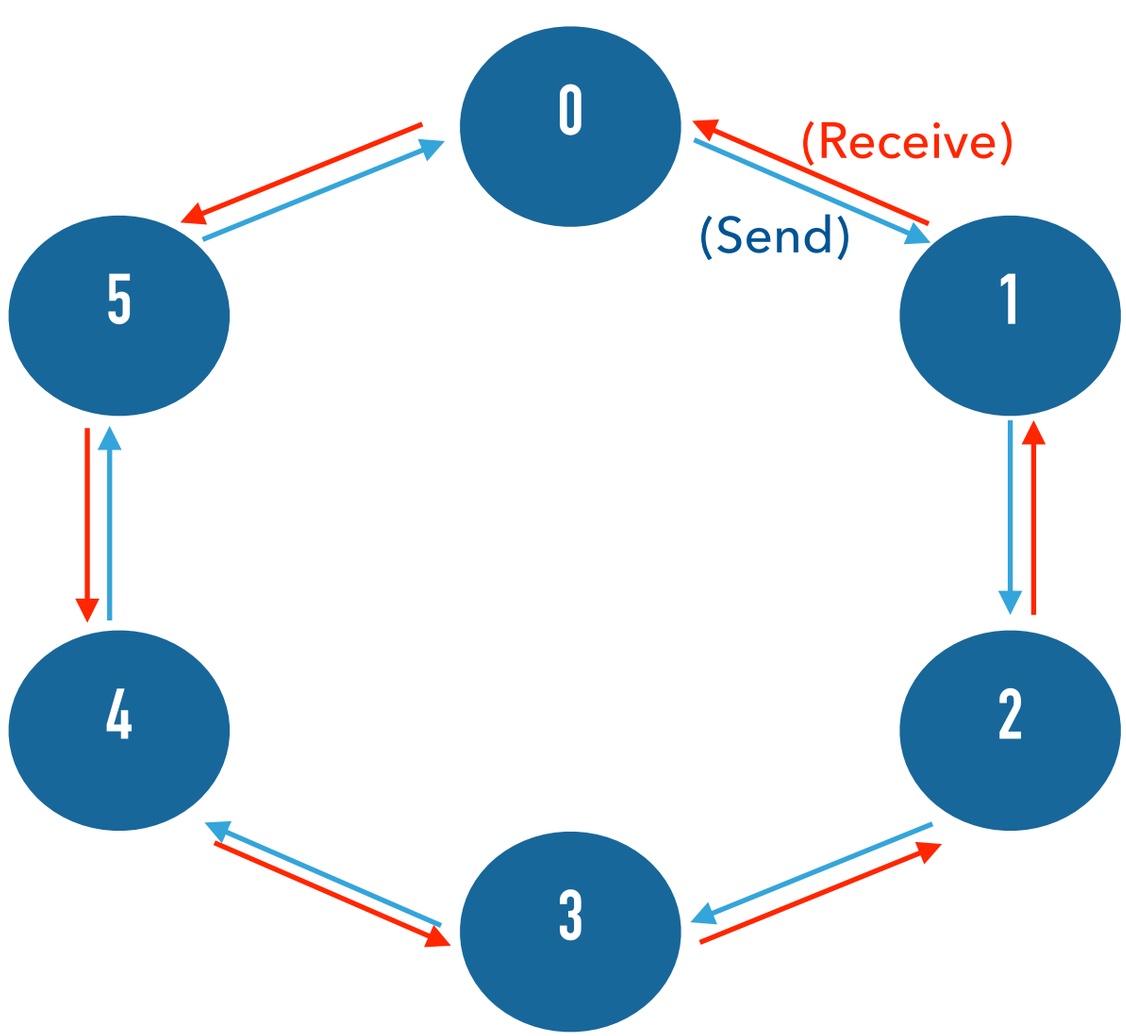
MPI_Isend();

MPI_Irecv();

PROBLEMS THAT COME UP: AVOID MESSAGE BLOCKING

```
int MPI_Isend(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int dest,
  int tag,
  MPI_Comm comm,
  MPI_Request *request
);
```

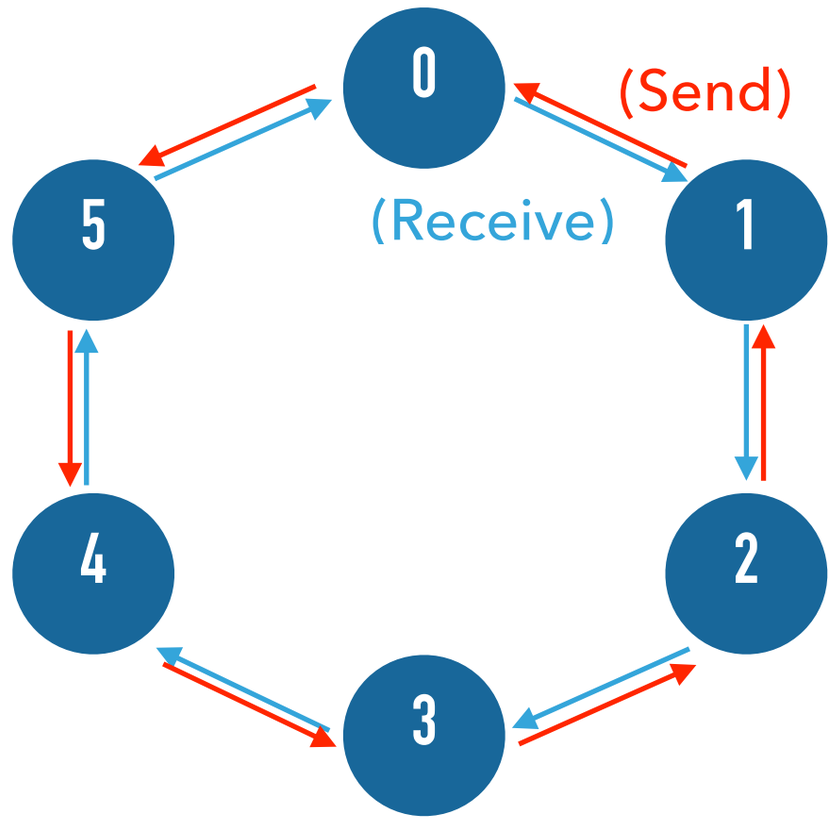
```
int MPI_Irecv(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int source,
  int tag,
  MPI_Comm comm,
  MPI_Request *request
);
```



EXERCISE 4: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that data is sent counterclockwise.

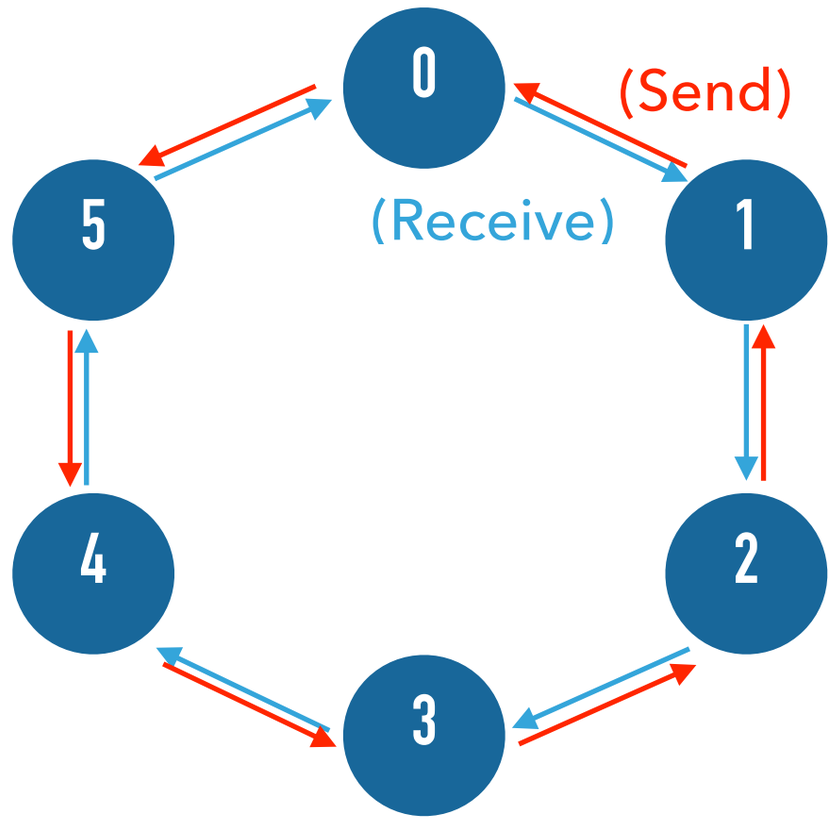
```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, neighbor_info, comm_size, neighbordown, neighborup;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == (comm_size - 1)) {
9         neighborup = 0;
10  }else{
11        neighborup = my_task_num + 1;}
12  MPI_Send(&my_task_num, 1, MPI_INT, neighborup, my_task_num, MPI_COMM_WORLD);
13  MPI_Recv(&neighbor_info, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
14  printf("I am %d and just received data from %d\n", my_task_num, neighbor_info);
15  MPI_Finalize();
```



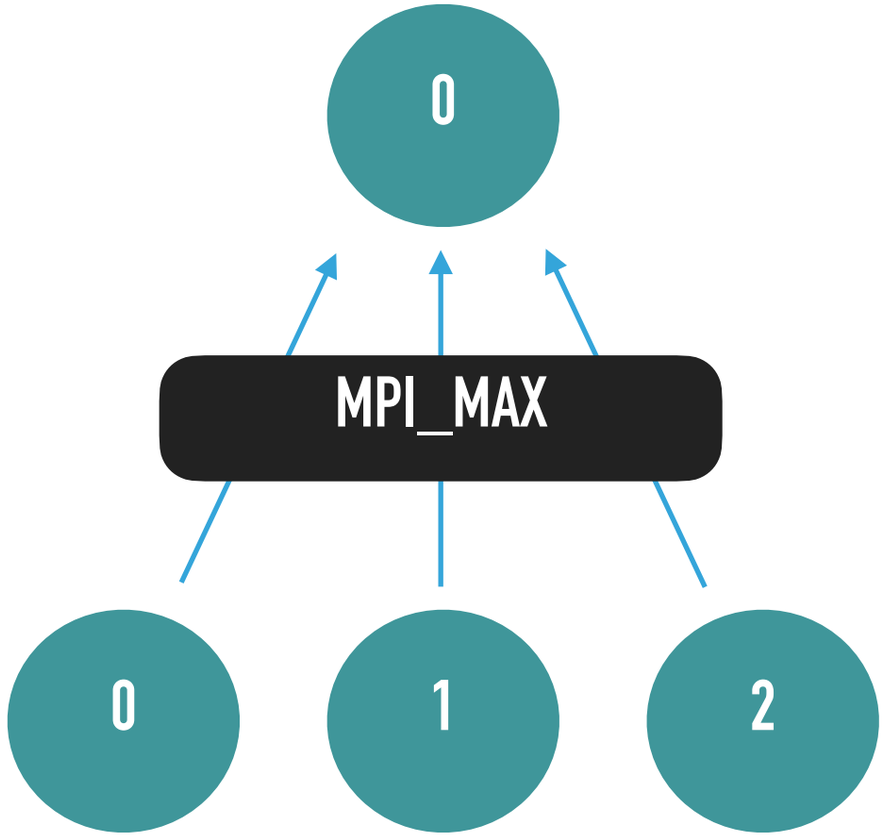
EXERCISE 4: WHICH LINES NEED TO BE UPDATED?

Goal: Change the code so that data is sent counterclockwise.

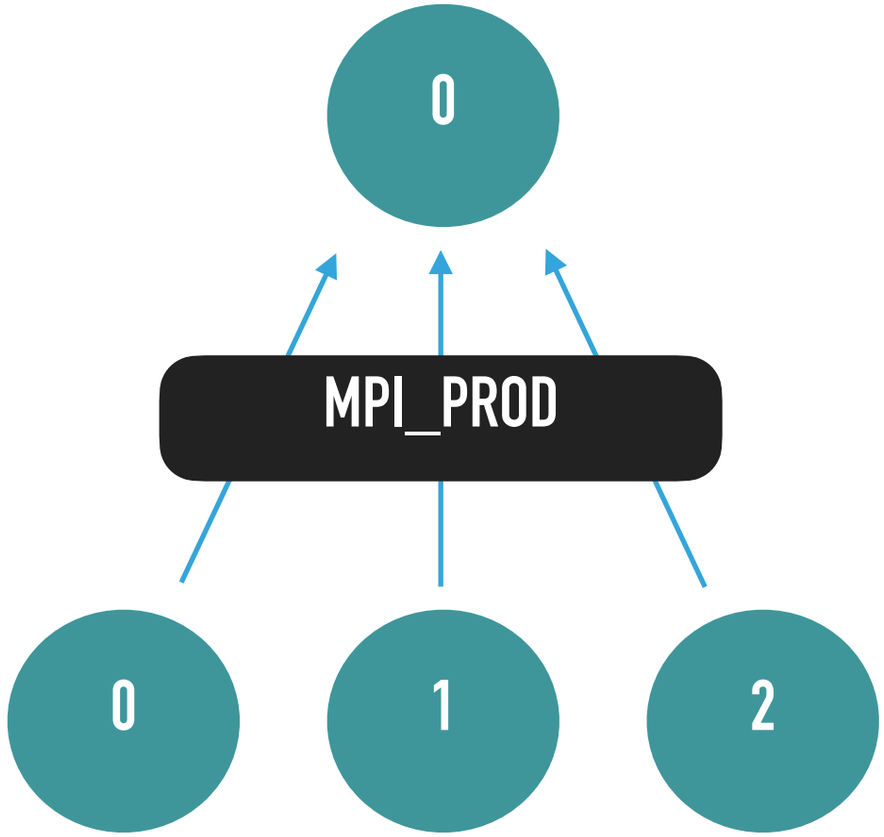
```
1   #include <stdio.h>
2   #include "mpi.h"
3   main(int argc, char** argv){
4   int my_task_num, neighbor_info, comm_size, neighbordown, neighborup;
5   MPI_Init(&argc, &argv);
6   MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);
7   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
8   if (my_task_num == (comm_size - 1)) {
9           neighborup = 0;
10  }else{
11          neighborup = my_task_num + 1;}
12  MPI_Send(&my_task_num, 1, MPI_INT, neighborup, my_task_num, MPI_COMM_WORLD);
13  MPI_Recv(&neighbor_info, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
14  printf("I am %d and just received data from %d\n", my_task_num, neighbor_info);
15  MPI_Finalize();
```



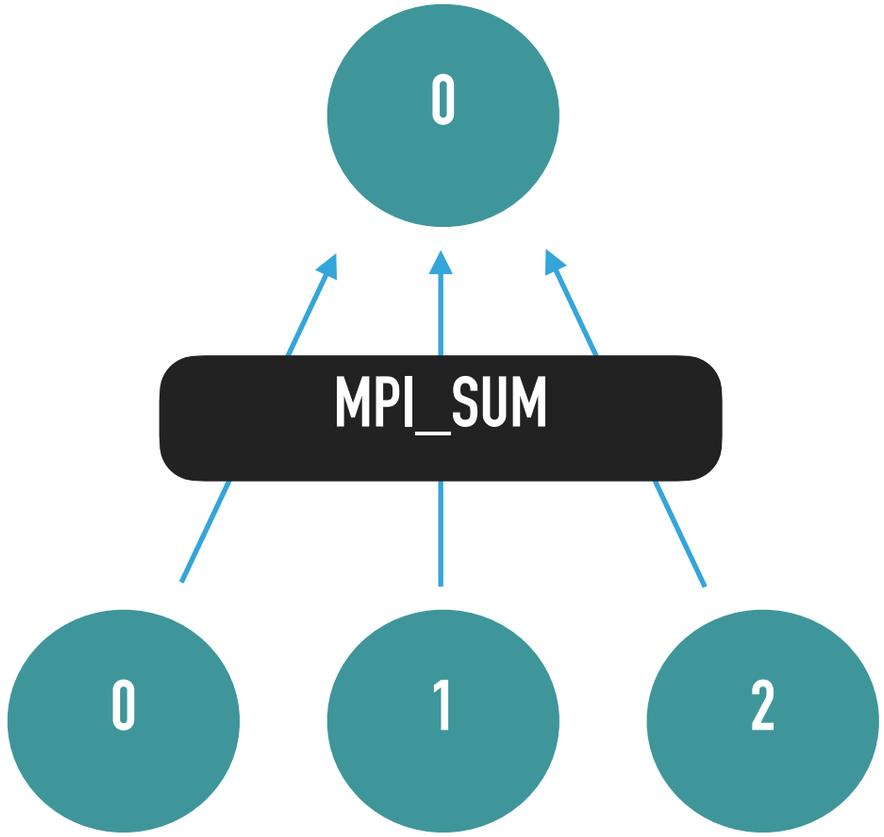
REDUCTION



REDUCTION



REDUCTION



REDUCTION: MPI_MAX

```
janeh@quartz2306:~/practice_MPI/HPC_CEA$ srun -n10 -ppdebug reduce-max
The maximum number in the communicator is 9.
janeh@quartz2306:~/practice_MPI/HPC_CEA$ srun -n15 -ppdebug reduce-max
The maximum number in the communicator is 14.
```

REDUCTION: MPI_MAX

```
#include <stdio.h>

#include "mpi.h"



main(int argc, char** argv){

int my_task_num, largest;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

MPI_Reduce(&my_task_num, &largest, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);

if (my_task_num == 0) {

      printf("The maximum number in the communicator is %d.", largest); }

MPI_Finalize();}
```

REDUCTION

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

Source: https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/

REDUCTION: MPI_PROD (EX: FACTORIAL)

```
#include <stdio.h>

#include "mpi.h"



main(int argc, char** argv){

int my_task_num, commsize, n, factorial;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

MPI_Comm_size(MPI_COMM_WORLD, &commsize);

n = my_task_num + 1;

MPI_Reduce(&n, &factorial, 1, MPI_INT, MPI_PROD, 2, MPI_COMM_WORLD);

if (my_task_num == 2) {

      printf("%d! is %d.", commsize, factorial);}

MPI_Finalize();}
```

REDUCTION: MPI_PROD (EX: FACTORIAL)

```
janeh@quartz2306:~/practice_MPI/HPC_CEA$ srun -n10 -ppdebug factorial
10! is 3628800.
janeh@quartz2306:~/practice_MPI/HPC_CEA$ srun -n5 -ppdebug factorial
5! is 120.
```

```
julia> factorial(10)
3628800

julia> factorial(5)
120
```

EXERCISE 5: WHAT DOES THIS PROGRAM DO?

```
#include <stdio.h>

#include "mpi.h"



main(int argc, char** argv){

int my_task_num, commsize, n, factorial;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

MPI_Comm_size(MPI_COMM_WORLD, &commsize);

n = my_task_num + 1;

MPI_Reduce(&n, &factorial, 1, MPI_INT, MPI_SUM, 2, MPI_COMM_WORLD);

if (my_task_num == 2) {

        printf("The answer is %d.", factorial);}

MPI_Finalize();}
```

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

Q: If you were to run the resulting executable with 4 tasks, what answer would you get?

A) 4

B) 24

C) 6

D) 10

EXERCISE 5

```
#include <stdio.h>

#include "mpi.h"



main(int argc, char** argv){

int my_task_num, commsize, n, factorial;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &commsize);

n = my_task_num + 1;

MPI_Reduce(&n, &factorial, 1, MPI_INT, MPI_SUM, 2, MPI_COMM_WORLD);

if (my_task_num == 2) {

      printf("The sum is %d.", factorial);}

MPI_Finalize();}
```

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

Q: If you were to run the resulting executable with 4 tasks, what answer would you get?

A) 4

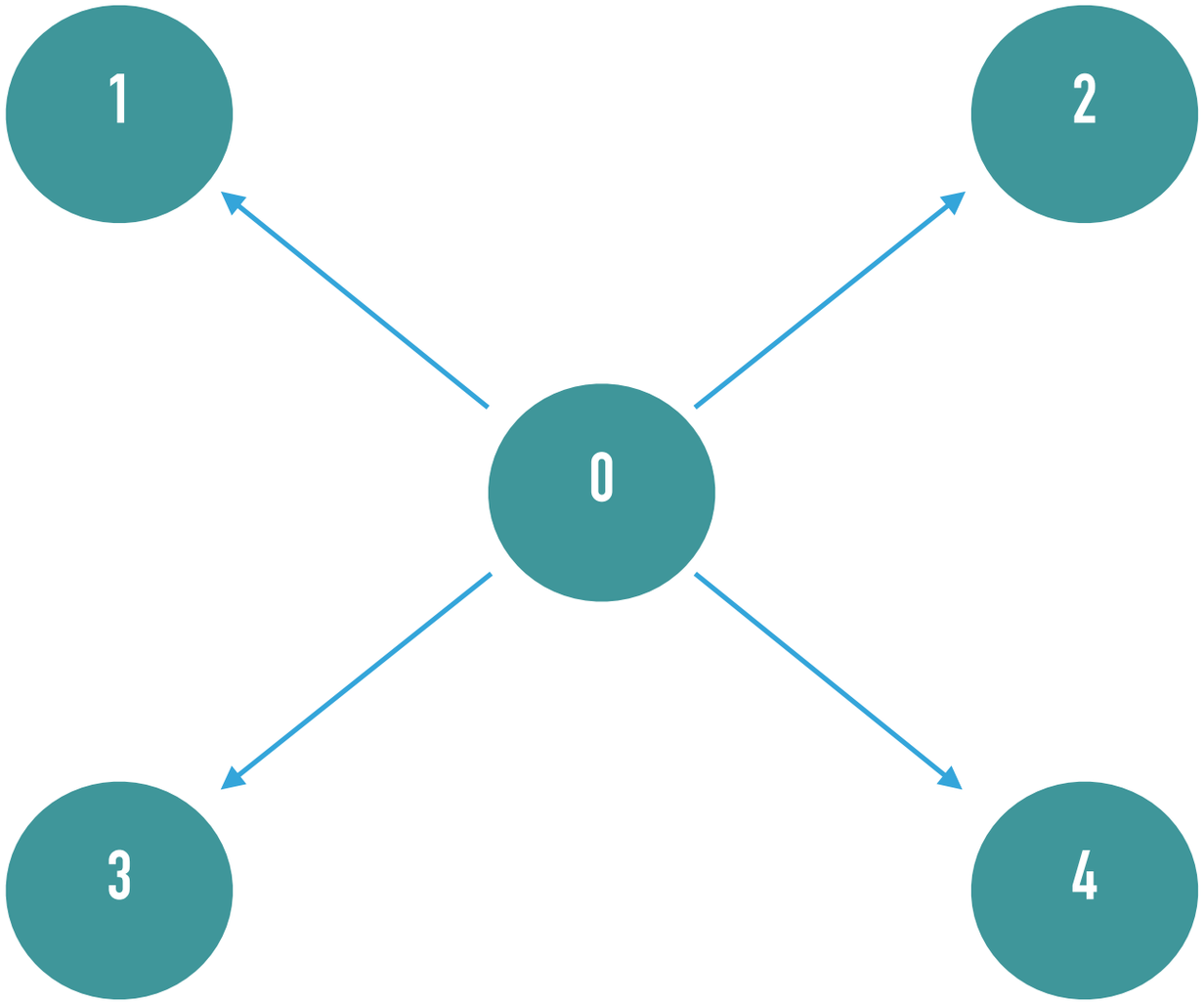
B) 24

C) 6

D) 10

```
janeh@pascal83:~/practice_MPI/HPC_CEA$ mpicc reduce-sum.c -o reduce-sum
janeh@pascal83:~/practice_MPI/HPC_CEA$ srun -n4 -ppvis reduce-sum
The answer is 10.
```

BROADCAST




```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

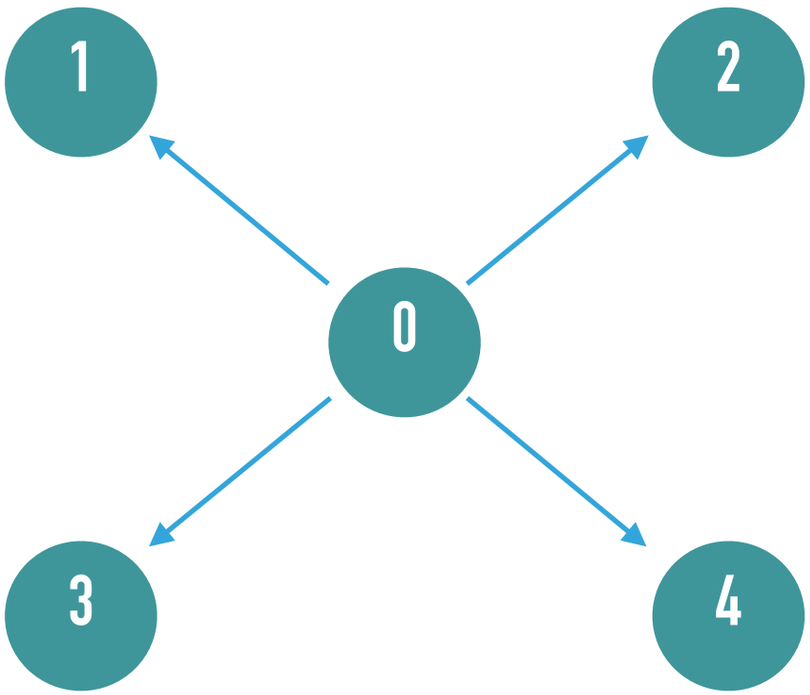
EXERCISE 6

Goal: Get `broadcast.c` working: send `answer` from 0 to all other processors.

```
#include <stdio.h>

#include "mpi.h"

main(int argc, char** argv){

int my_task_num, answer;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &my_task_num);

if (my_task_num == 0){

        answer = 42;

}

printf("Before broadcast, I am %d and the answer is %d.\n", my_task_num, answer);

/* Call broadcast here!*/

printf("After broadcast, I am %d and the answer is %d.\n", my_task_num, answer);

MPI_Finalize();}
```

EXERCISE 6: HOW WOULD YOU CALL BROADCAST?

Goal: Get `broadcast.c` working: send `answer` from 0 to all other processors.




```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

A) MPI_Bcast(&answer, 1, MPI_INT, 0, MPI_COMM_WORLD);

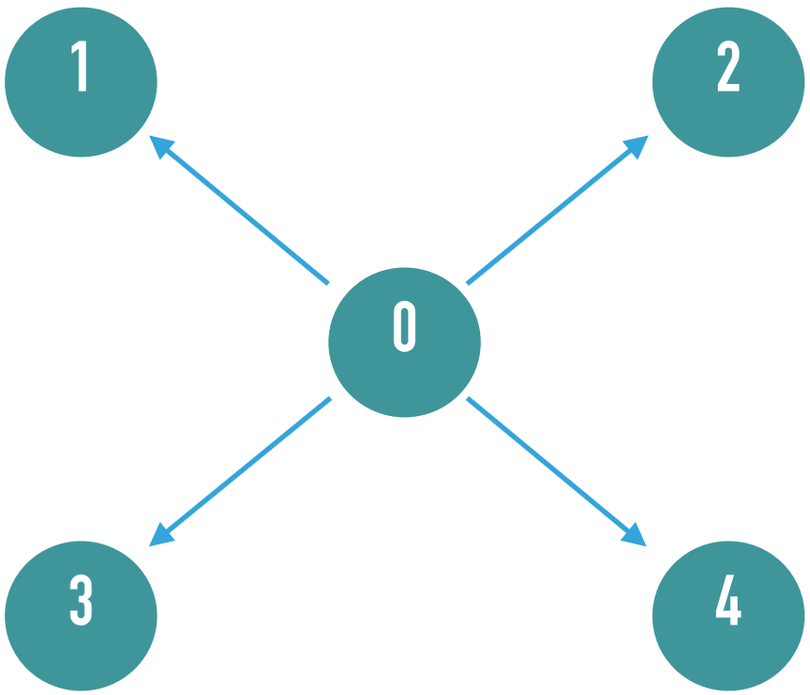
B) MPI_Bcast(&answer, 0, MPI_COMM_WORLD, 1, MPI_INT);

C) MPI_Bcast(&answer, 1, MPI_COMM_WORLD, 0, MPI_INT);

D) MPI_Bcast(&answer, 0, MPI_INT, 1, MPI_COMM_WORLD);

EXERCISE 6: HOW WOULD YOU CALL BROADCAST?

Goal: Get `broadcast.c` working: send `answer` from 0 to all other processors.




```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

A) MPI_Bcast(&answer, 1, MPI_INT, 0, MPI_COMM_WORLD);

B) MPI_Bcast(&answer, 0, MPI_COMM_WORLD, 1, MPI_INT);

C) MPI_Bcast(&answer, 1, MPI_COMM_WORLD, 0, MPI_INT);

D) MPI_Bcast(&answer, 0, MPI_INT, 1, MPI_COMM_WORLD);

DISCLAIMER