MPI Collective Communication

Getting started

Login:

```
$ ssh  <username> @bw.ncsa.illinois.edu        <ENTER>
```

Interactive node request:

```
$ qsub  - I  - l  nodes=4:ppn=32:xe,walltime=03:00:00        <ENTER>
```

Download code:

```
$ wget  http://shodor.org/~mludin/BW_Capstone/mpi_collective_comm.tar        <ENTER>
```

Extract the Zip File:

```
$ tar  - xvvf  mpi_collective_comm.tar        <ENTER>
```

Change folders:

```
$ cd  mpi_collective_comm/        <ENTER>
```

```
$ ls  - l
```

Message Passing Interface (MPI)

- Standard for distributed memory parallelism.

- Allows for multiple nodes (or just multiple cores) to run a program in parallel.

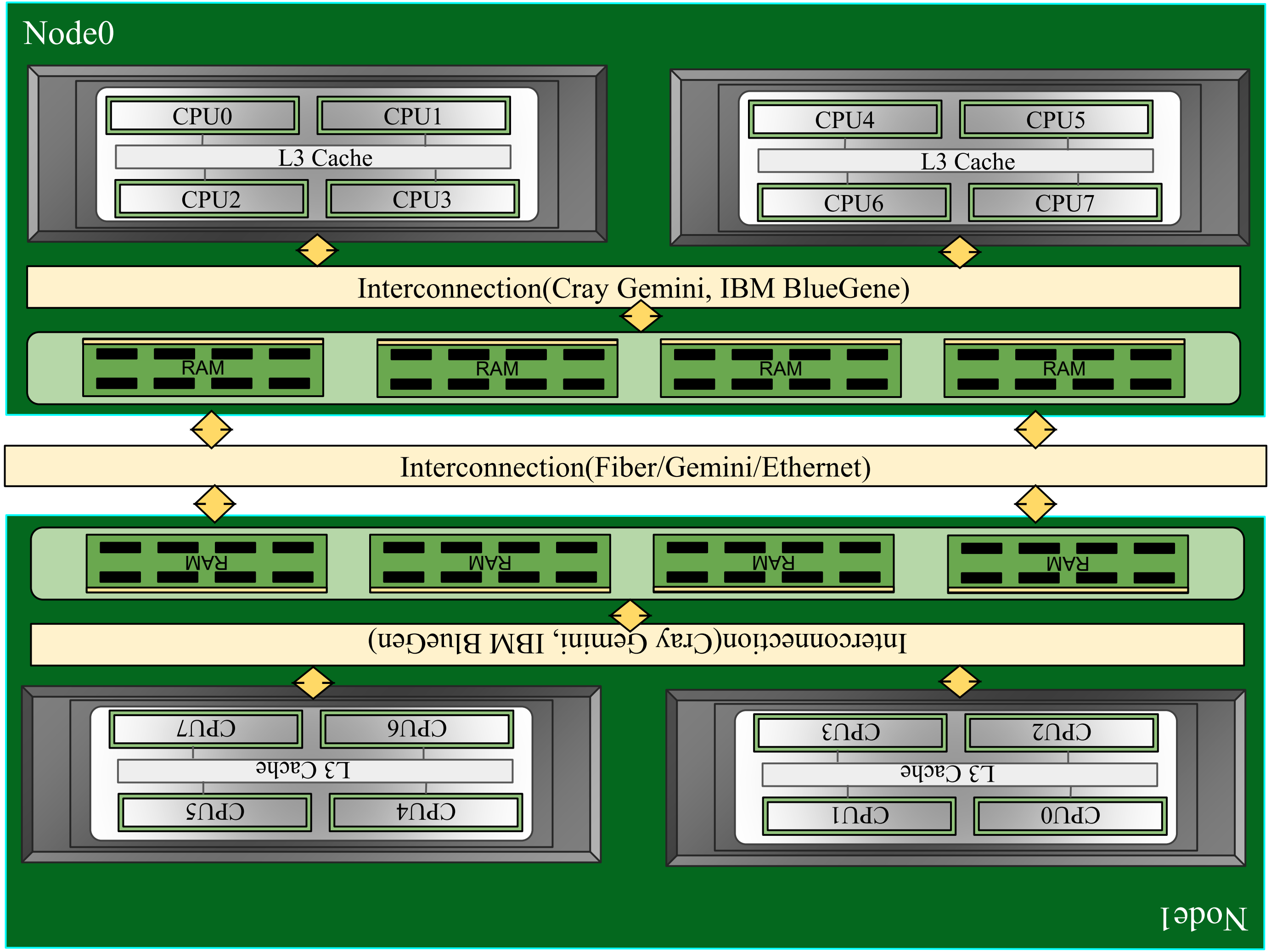
- Utilizes function calls as opposed to compiler directives.

- Syntax example: send a message:

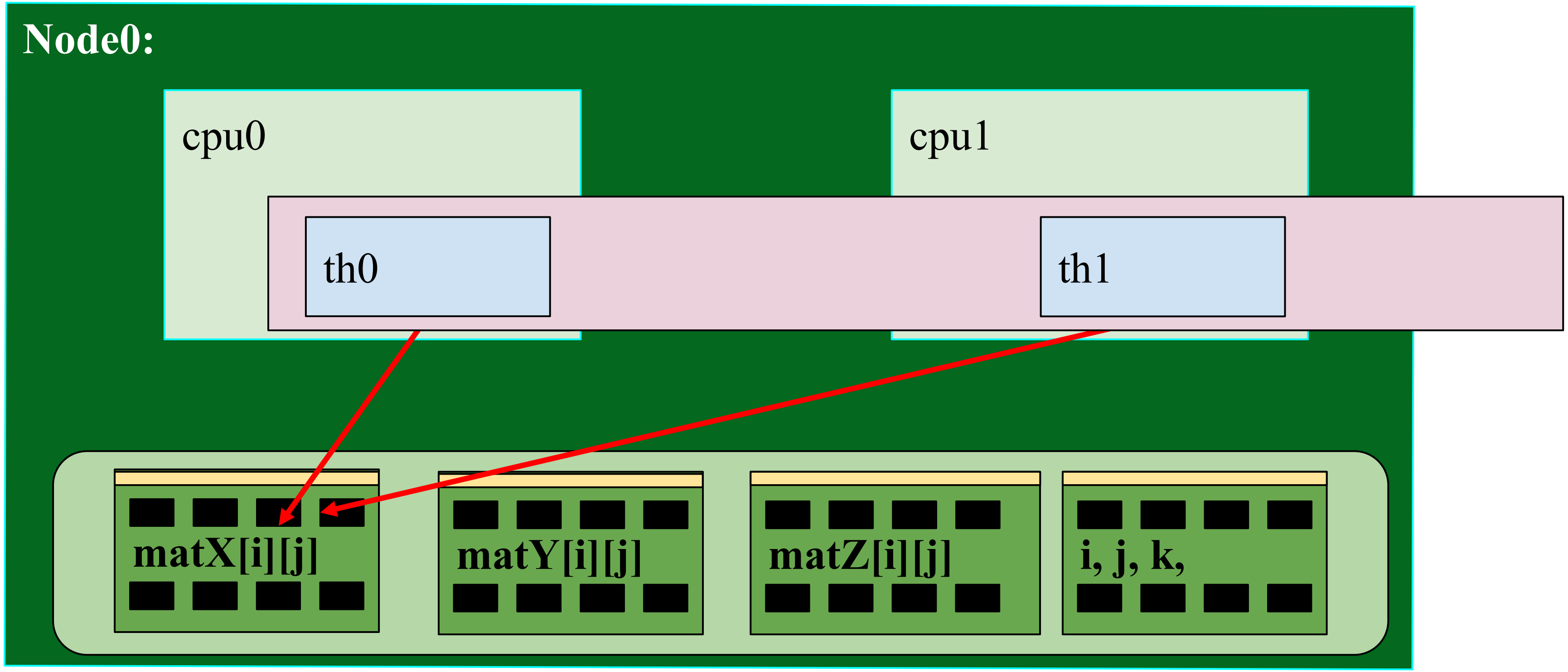
```
MPI_Send(&buffer, count, MPI_INT, destination, tag, MPI_COMM_WORLD);
```

Distributed Memory Multi-node System

$ numactl -- hardware

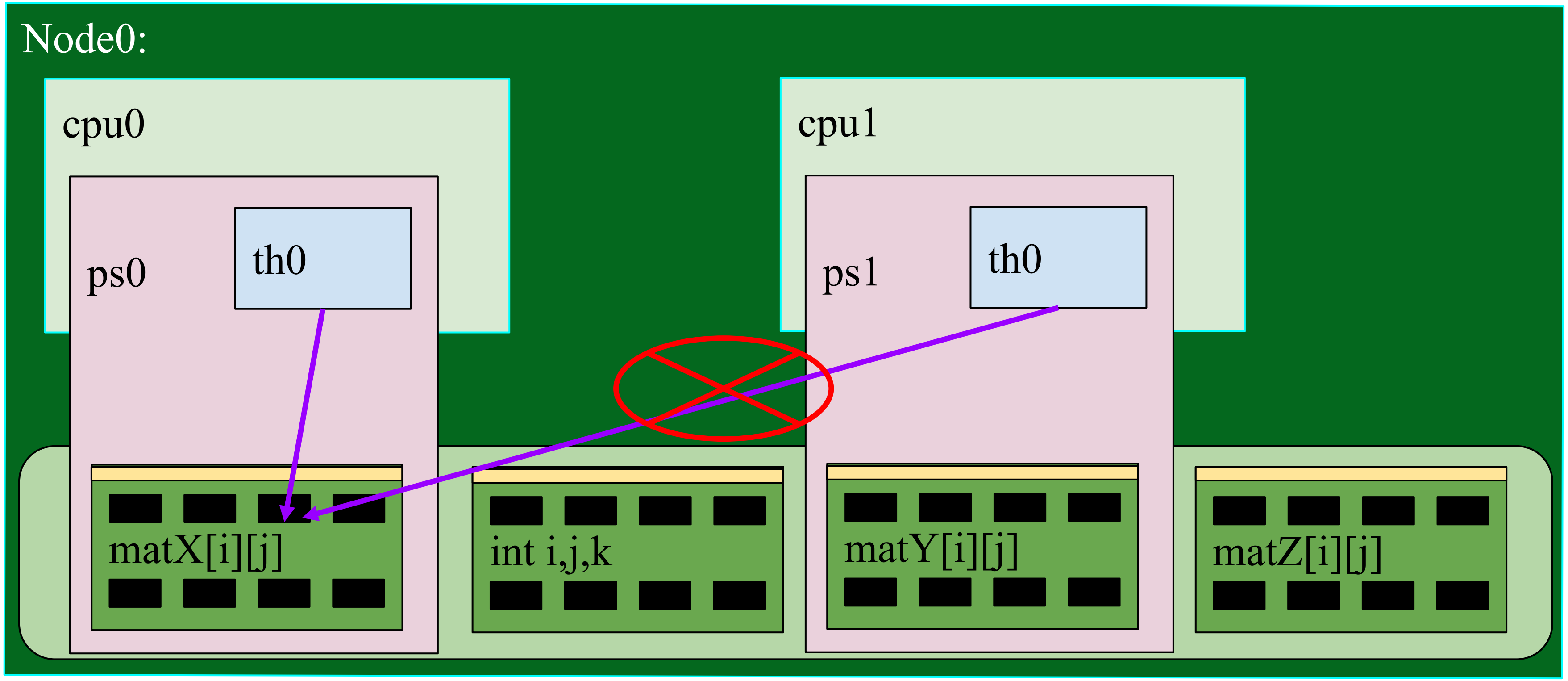


Shared-Memory Threads: (Review)



Shared-Memory: Threads (th0, th1) within a process accessing data.

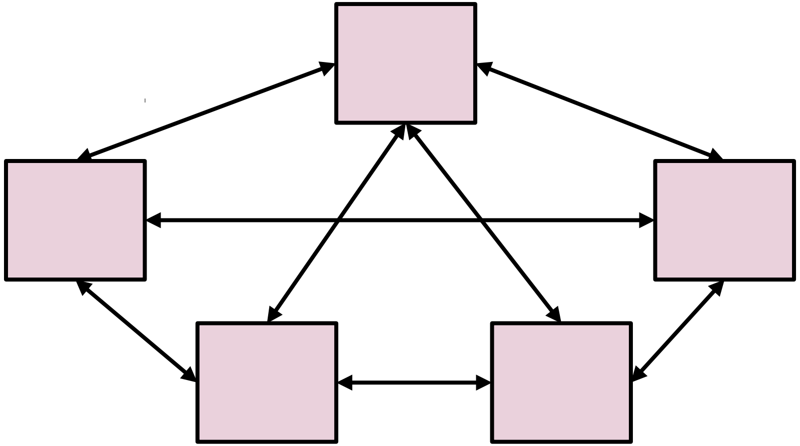
Distributed-Memory Processes:



Distributed-Memory: multiple processes within SPMD accessing data.

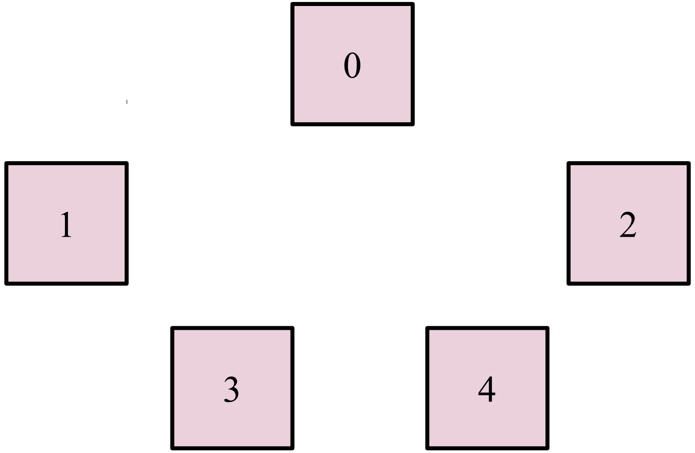
Communicator (grouping processes)

- A collection of MPI processes that can send and receive messages to and from each other.
- Normally this is all of the processes, and there is a constant defined for it, MPI_COMM_WORLD



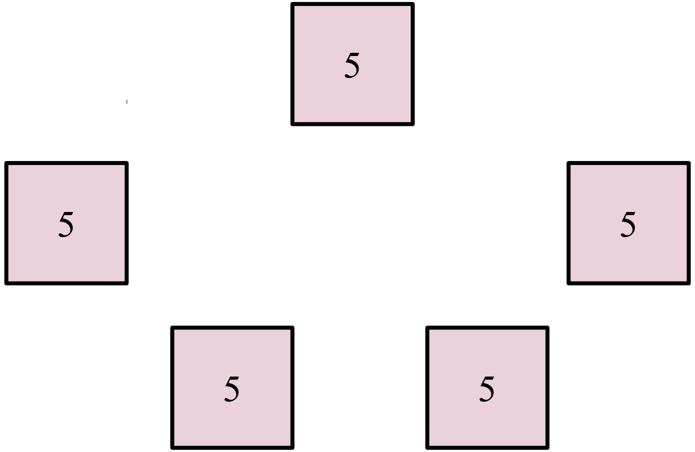
Rank

- Unique identifier for each process in the communicator.
- Usually an integer starting at 0 and counting upwards.

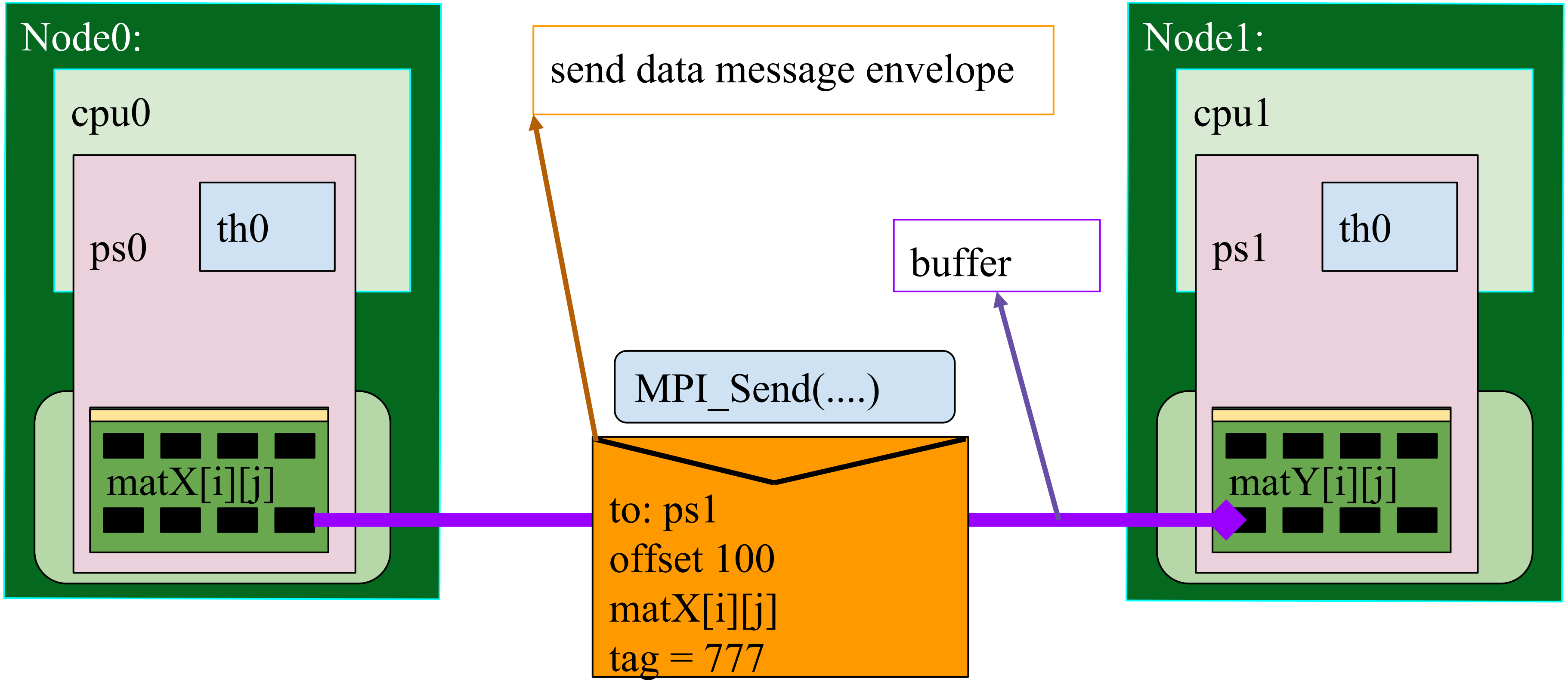


Size

- Number of processes in a communicator.
- Same for all processes in the communicator.



MPI_SEND(...):



Node0:

cpu0

ps0

th0

matX[i][j]

send data message envelope

buffer

MPI_Send(....)

to: ps1
offset 100
matX[i][j]
tag = 777

Node1:

cpu1

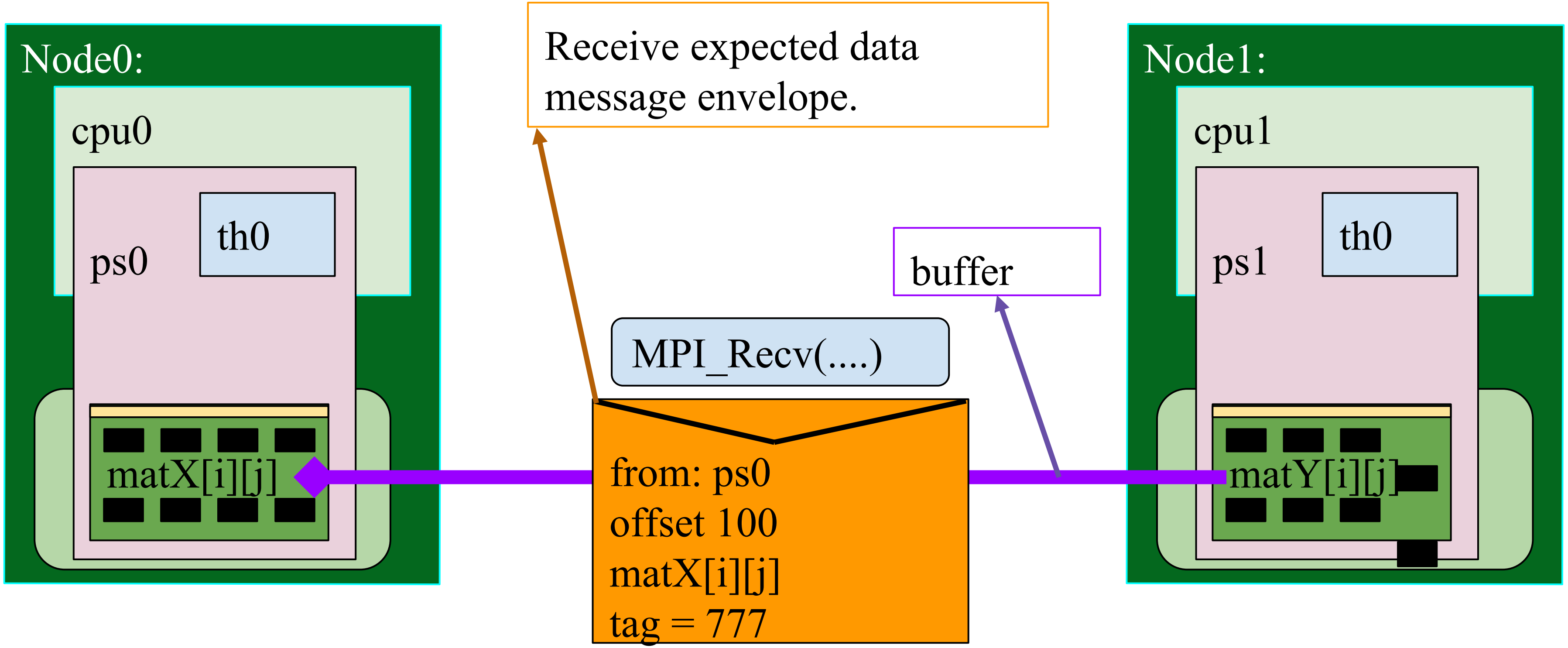
ps1

th0

matY[i][j]

ps0 places matrix matX[100-199] from local memory into buffer and calls send routine.

MPI_Recv(...):



ps1 places MPI_Recv(), and awaits until the data from ps0 gets to its buffer, before copy it to local storage

MPI Program Structure:

```
#include <mpi.h>         //#include mpi header file.
char message[200];       //message size
int my_rank, num_ps;         //Variable declaration
MPI_Init(&argc, &argv);  // Start MPI Environment now

if (my_rank == master ) {
   MPI_Recv(message)
}
else {
   MPI_Send(&message)
}
MPI_Finalize() //close MPI communication
```

Examples:

[mpi_bcast.c]

```
        $ less mpi_bcast.c
```

How to compile:

```
        $ make mpi_bcast
```

How to run:

```
        $ aprun -n 4 ./mpi_bcast.exe
```

[mpi_reduce.c]

```
        $ less mpi_reduce.c
```

How to compile:

```
        $ make mpi_reduce
```

How to run:

```
        $ aprun -n 4 ./mpi_reduce.exe
```

[mpi_pi_area.c]

```
        $ less mpi_pi_area.c
```

How to compile:

```
        $ make mpi_pi_area
```

How to run:

```
        $ aprun -n 4 ./mpi_pi_area.exe
```

[mpi_scatter.c]

```
        $ less mpi_scatter.c
```

How to compile:

```
        $ make mpi_scatter
```

How to run:

```
        $ aprun -n 4 ./mpi_scatter.exe
```

[mpi_gather.c]

```
        $ less mpi_gather.c
```

How to compile:

```
        $ make mpi_gather
```

How to run:

```
        $ aprun -n 4 ./mpi_gather.exe
```

[mpi_allgather.c]

```
        $ less mpi_allgather.c
```

How to compile:

```
        $ make mpi_allgather
```

How to run:

```
        $ aprun -n 4 ./mpi_allgather.exe
```

References / Further Readings

a. Open MPI Organization/Community
b. Top500 List of supercomputers in the world
c. MPI Library Man Pages
d. MPI Standards
e. MPI Tutorials
f. More MPI Tutorials