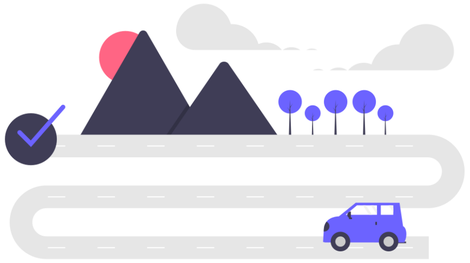
CME 213, ME 339 Spring 2023
Introduction to parallel computing using MPI, openMP, and CUDA

Eric Darve, ICME, Stanford

"Make everything as simple as possible, but not simpler."
Albert Einstein

Stanford University

Recap



- Distributed memory vs. shared memory computing
- MPI: message passing interface
- Independent programs run and exchange data using send and receive operations.

Collective communications

Collective communication operations

- In cases where multiple processes need to communicate (e.g., for a reduction or broadcast operation), using an MPI collective communication is simpler and more efficient.
- This is preferred over many MPI_Send/MPI_Recv.
- MPI collective communication can make optimal use of the network topology.

Examples of networks

a) complete graph

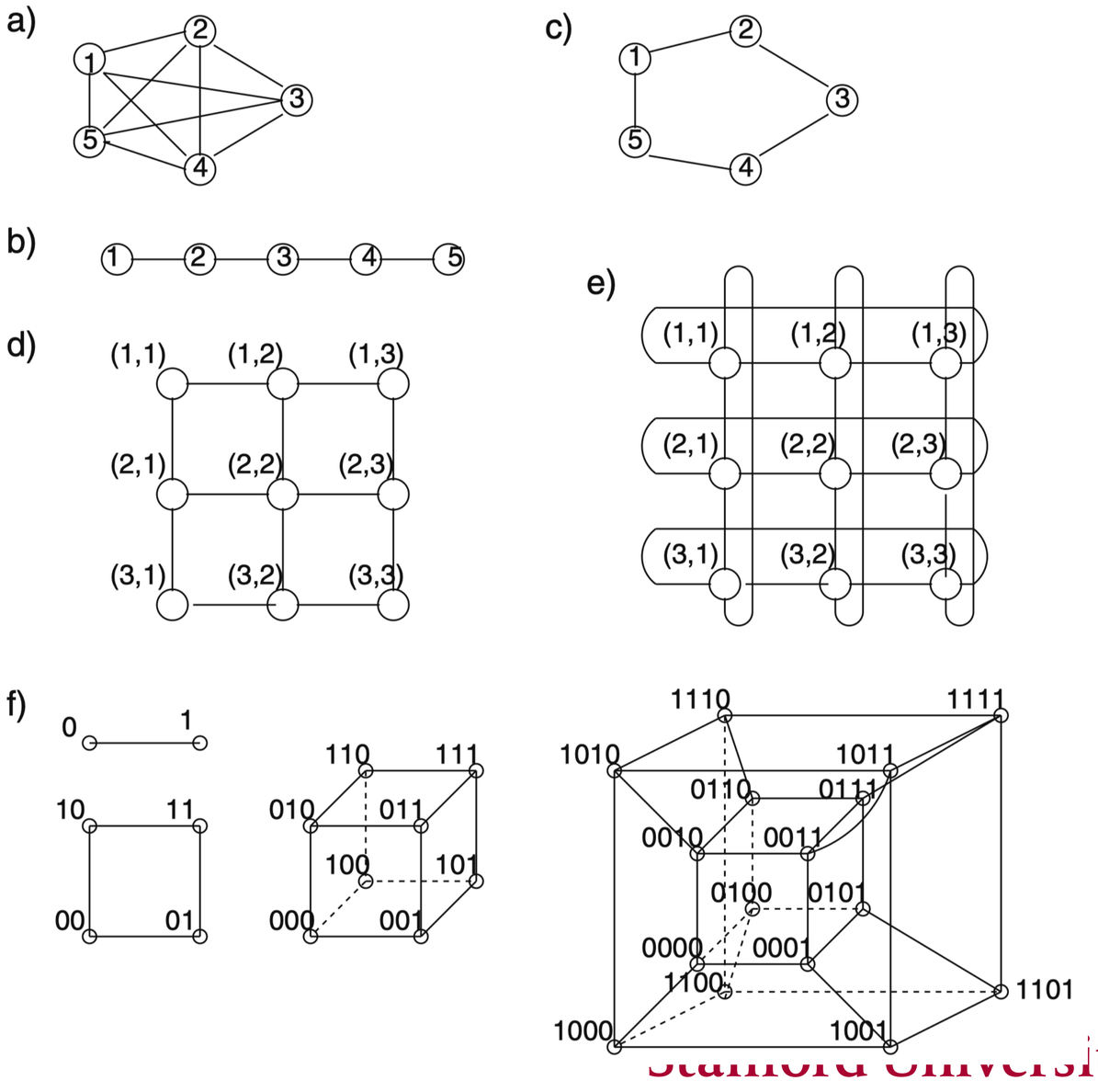
b) linear array

c) ring

d) 2D mesh

e) 2D torus

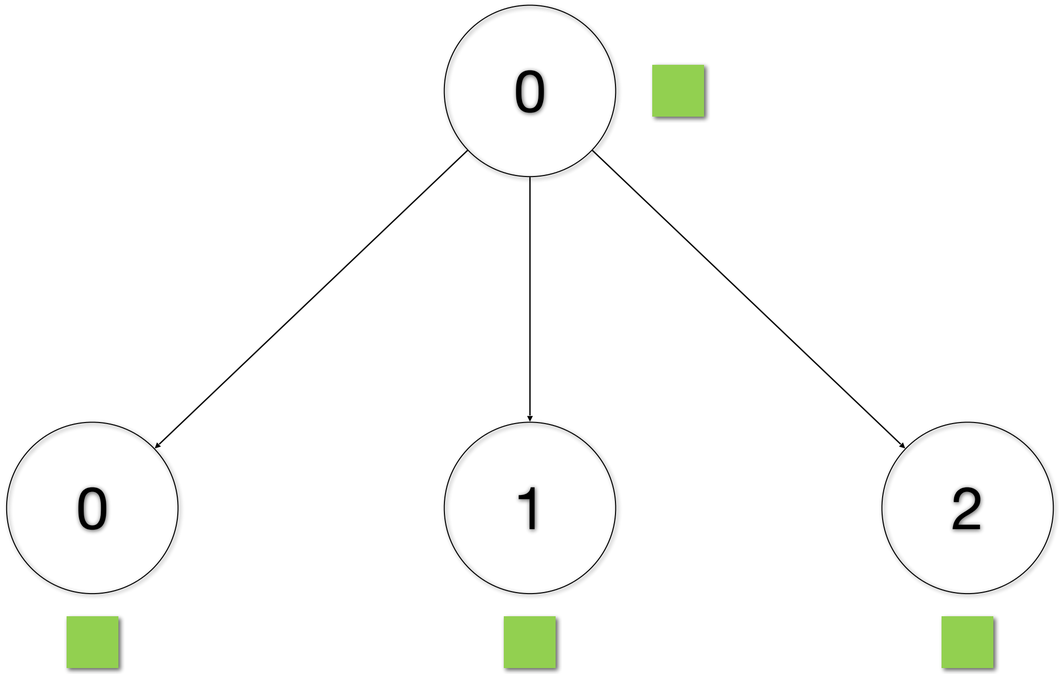
f) k-dimensional cube



Let's review the main collective functions in MPI

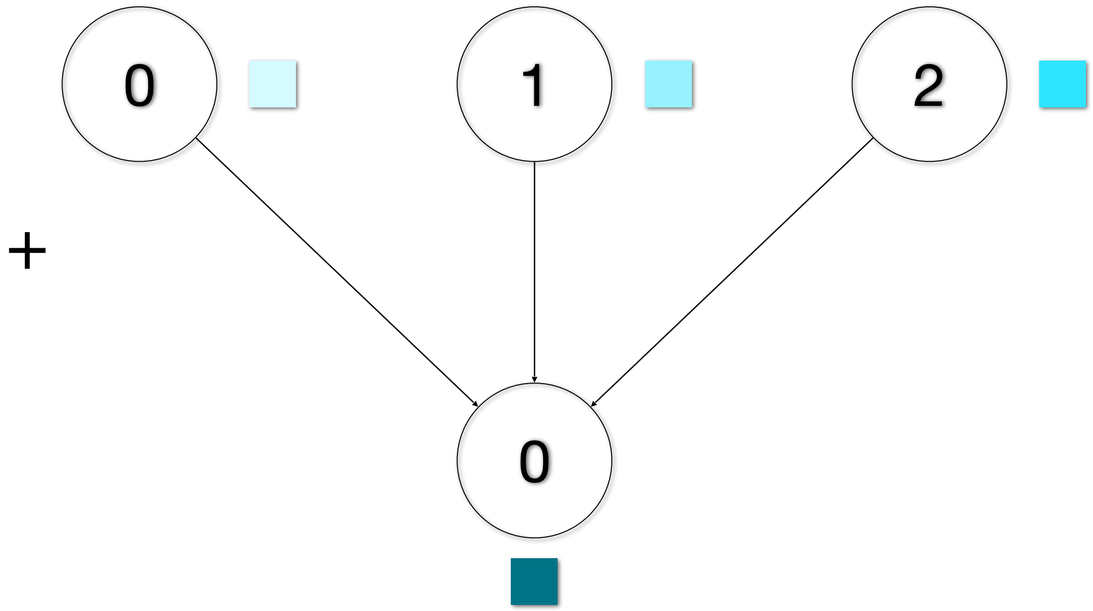
`MPI_Bcast(&buffer,count,datatype,root,comm)`

One process sends data to all the other processes



```
MPI_Reduce(&sendbuf,&recvbuf,count,datatype,op,root,comm)
```

One process receives the result of reducing data from all the processes.



All MPI reduction operations

MPI Reduction Operation		C Data Types	Fortran Data Type
MPI_MAX	maximum	integer, float	integer, real, complex
MPI_MIN	minimum	integer, float	integer, real, complex
MPI_SUM	sum	integer, float	integer, real, complex
MPI_PROD	product	integer, float	integer, real, complex
MPI_LAND	logical AND	integer	logical
MPI_BAND	bit-wise AND	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LOR	logical OR	integer	logical
MPI_BOR	bit-wise OR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer	logical
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double	real, complex,double precision
MPI_MINLOC	min value and location	float, double and long double	real, complex, double precision

Example: `mpi_prime.cpp`

```
// Total number of primes found by all processes:     MPI_SUM
MPI_Reduce(&pc, &pcsum, 1, MPI_INT, MPI_SUM, ROOT, MPI_COMM_WORLD);

// The largest prime that was found by all processes: MPI_MAX
MPI_Reduce(&foundone, &maxprime, 1, MPI_INT, MPI_MAX, ROOT, MPI_COMM_WORLD);
```

We perform two reductions.

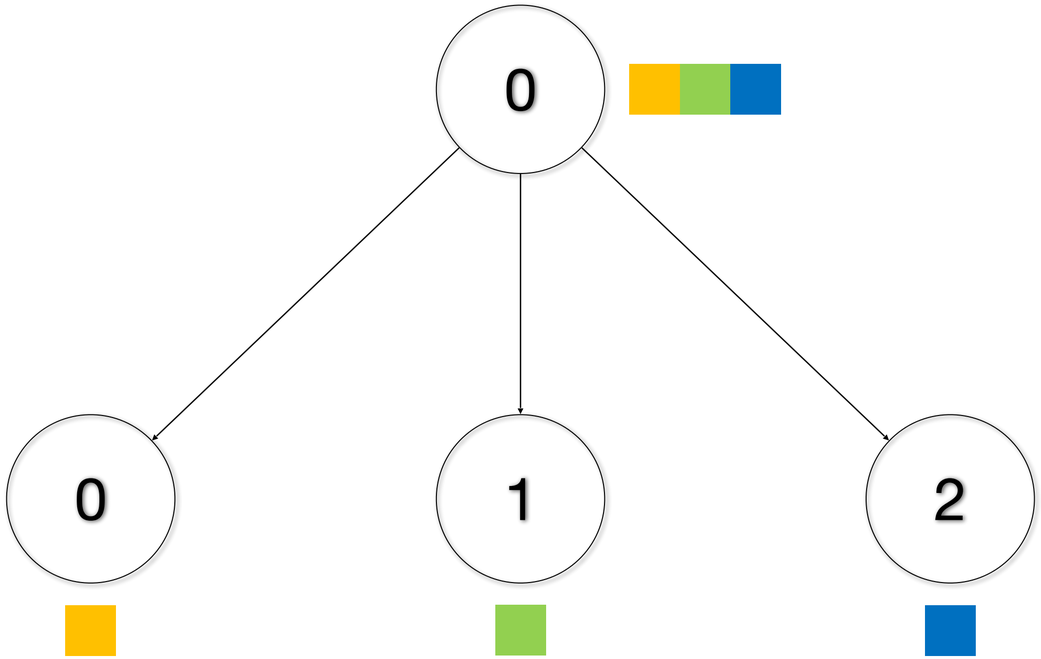
- `MPI_SUM` calculates the total number of primes that were found.
- `MPI_MAX` calculates the largest prime number that was found.

Output

```
darve@icme-gpu1:~/2023/MPI/Lecture_16$ srun -n 4 -p CME ./mpi_prime
MPI task 1 has started on icmet01 [total number of processors 4]
MPI task 2 has started on icmet01 [total number of processors 4]
MPI task 3 has started on icmet01 [total number of processors 4]
MPI task 0 has started on icmet01 [total number of processors 4]
Using 4 tasks to scan 40000000 numbers...
Done.
Largest prime is 39999983.
Total number of primes found: 2433654
Wall clock time elapsed: 12.75 seconds
darve@icme-gpu1:~/2023/MPI/Lecture_16$
```

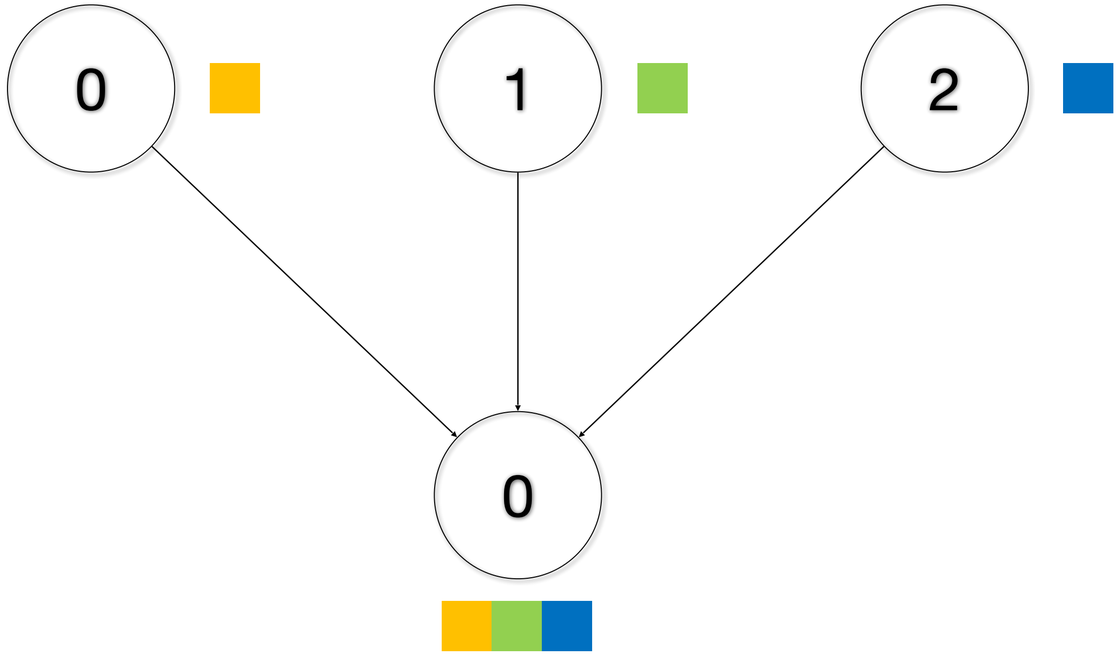
```
MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root,comm)
```

One process sends different data to the other processes



```
MPI_Gather(&sendbuf,sendcnt,sendtype,&recvbuf,recvcount,recvtype,root,comm)
```

One process gathers data from the other processes



Final project

- Rank 0 reads MNIST data from disk.
- `MPI_Scatter` can send the images to all the other processes.

Number of GPUs = 3

- Another MPI function, `MPI_Scatterv`, is required when the number of processes does not divide the number of images, e.g., the number of GPUs is 3.

- In that case, process 0 needs to send a different number of images to the other processes.

Variant: `MPI_Scatterv`

Scatters a buffer in parts to all tasks in a group.

```
int MPI_Scatterv(&sendbuf, &sendcounts, &displs,
sendtype, &recvbuf, recvcount, recvtype, root, comm)
```

`MPI_Scatterv` extends the functionality of `MPI_Scatter` by allowing a **varying count of data** to be sent to each process since `sendcounts` is now an array. It also allows more flexibility as to where the data is taken from on the root, by providing the new argument, `displs`.

Example

This example uses a stride.

The root process scatters sets of 100 ints to the other processes, but the sets of 100 are **stride** ints apart in the sending buffer.

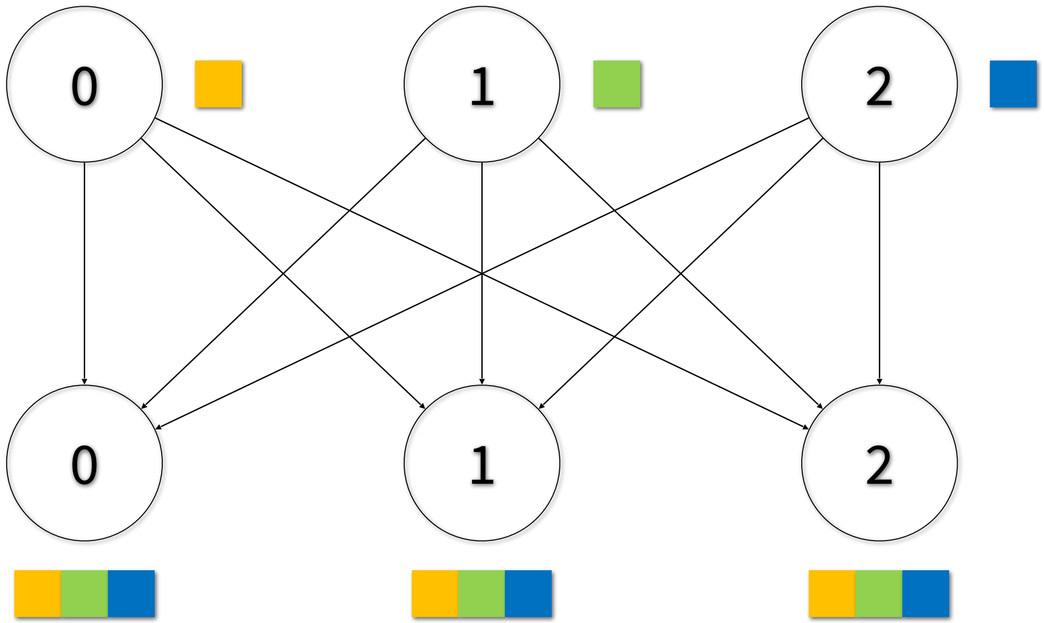
See more details and examples in the documentation.

https://www.open-mpi.org/doc/v4.1/man3/MPI_Scatterv.3.php

```
MPI_Comm comm;
int gsize,*sendbuf;
int root, rbuf[100], i, *displs, *scounts;
...
MPI_Comm_size(comm, &gsize);
sendbuf = (int *)malloc(gsize*stride*sizeof(int));
...
displs = (int *)malloc(gsize*sizeof(int));
scounts = (int *)malloc(gsize*sizeof(int));
for (i=0; i<gsize; ++i) {
  displs[i] = i*stride;
  scounts[i] = 100;
}
MPI_Scatterv(sendbuf, scounts, displs, MPI_INT,
             rbuf, 100, MPI_INT, root, comm);
```

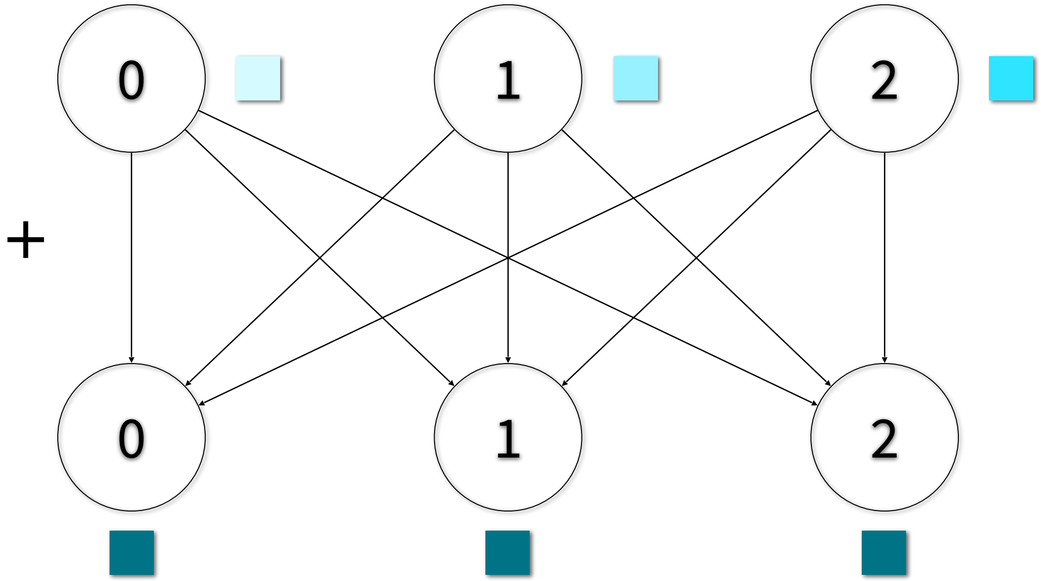
```
MPI_Allgather(&sendbuf,sendcount,sendtype,&recvbuf,recvcount,recvtype,comm)
```

All processes gather data from all the other processes.



```
MPI_Allreduce(&sendbuf,&recvbuf,count,datatype,op,comm)
```

Same as reduction but result is stored at all processes.



Final project

There are a few reductions that you will have to perform using MPI.

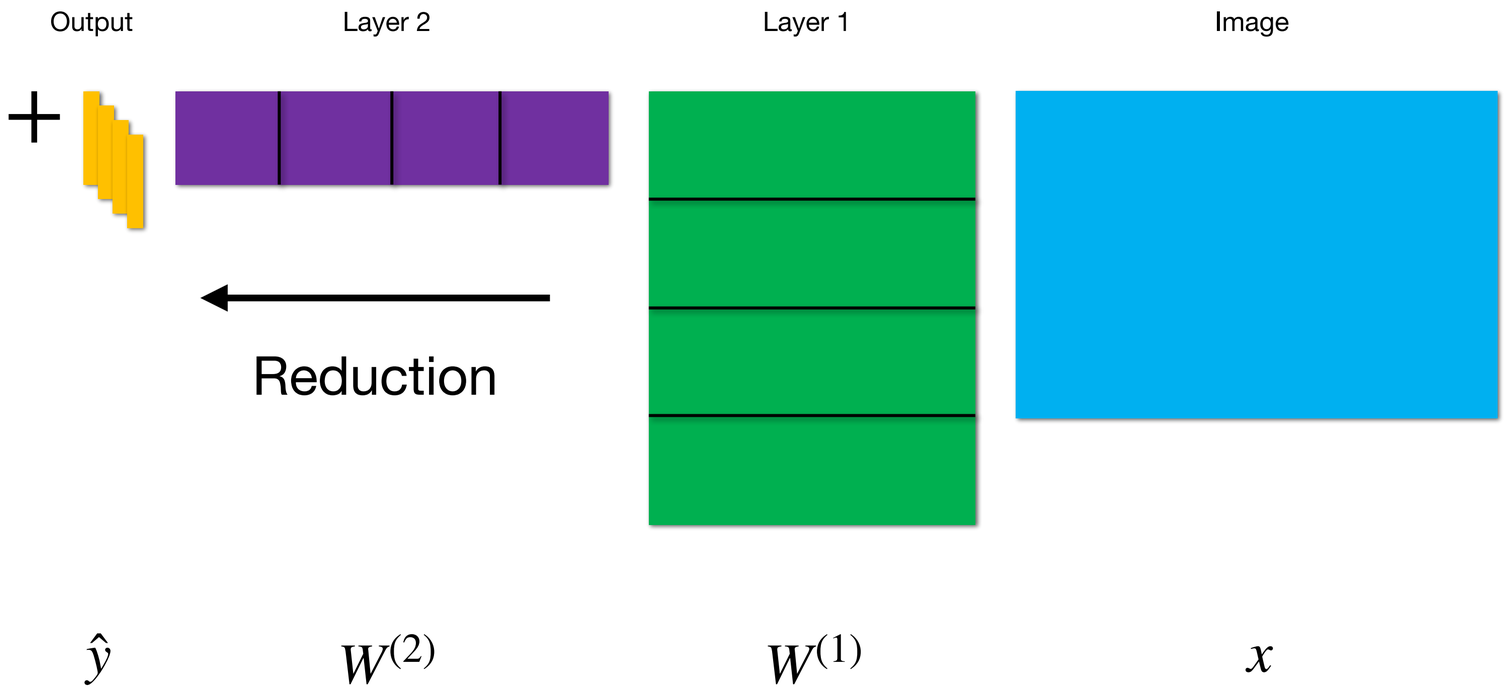
Implementation 1: reduction over the input images.

$$p \leftarrow p - \frac{\alpha}{N} \nabla_p \sum_{i=1}^{N} H(y_i, \hat{y}_i(p))$$

Reduction is required over the index i.

`MPI_Allreduce` to get the complete gradient on all processes.

Final project



Implementation 2: reduction for layer 2 in order to obtain the final output.

Example: computing the global minimum

- All processes have an array of integer.
- We compute the minimum over all processes.
- The global minimum is then sent to all processes.

MPI_Allreduce

- `localres` stores the minimum and the rank of the process
- `MPI_MINLOC`: first integer in `localres` is used to compute the global minimum.
- The second integer is the rank of the process where the global minimum was found.

```
int localres[2];
int globalres[2];
// Compute the minimum of localarr and store the result in localres[0]
localres[0] = localarr[0];

for (int i = 1; i < locn; i++)
  if (localarr[i] < localres[0]) {
    localres[0] = localarr[i];
  }

// The second entry is the rank of this process.
localres[1] = rank;

/* MPI_Allreduce: the minimum across all processes is computed.
 * The result is broadcast to all processes.
 * MPI_MINLOC: this is like the operator MIN. The difference is that it
 * takes as input two numbers; the first one is used to determine the
 * minimum value.
 * The second number just "goes along for the ride."
 * MPI_2INT: MPI type for 2 integer values. */
MPI_Allreduce(localres, globalres, 1, MPI_2INT, MPI_MINLOC, MPI_COMM_WORLD);

// The difference with MPI_Reduce is that all processes now have the result.
if (rank == 0) {
  printf("Rank %2d has the lowest value of %d\n\n", globalres[1],
         globalres[0]);
}
```

Example output

Rank 3 has the lowest value.

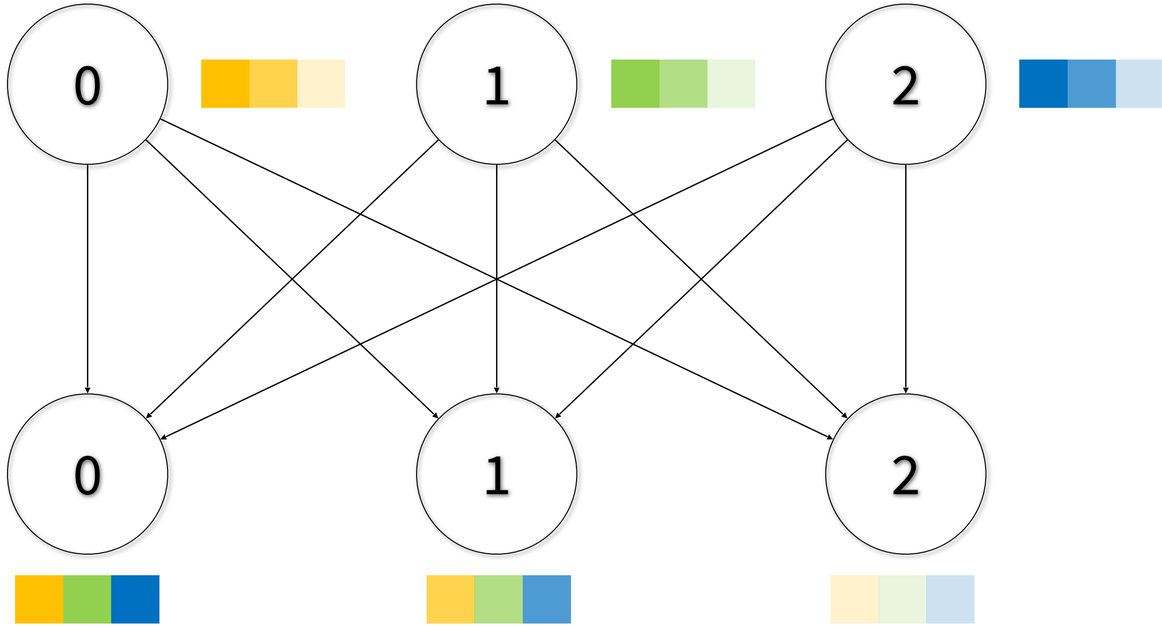
This value is sent to all processes.

```
Rank   0 has values:   2422   8536   5688   3004   4656
Rank   1 has values:   9537   5372    923   7969   2686
Rank   2 has values:   8812   6499   6363   6749   5055
Rank   3 has values:    628   1178   6188   5565   9765

Rank  3 has the lowest value of 628

Rank  0 has received the value:   628
Rank  1 has received the value:   628
Rank  2 has received the value:   628
Rank  3 has received the value:   628
```

```
MPI_Alltoall(&sendbuf,sendcount,sendtype,&recvbuf,recvcnt,recvtype,comm)
```

This is the equivalent of a matrix transpose.

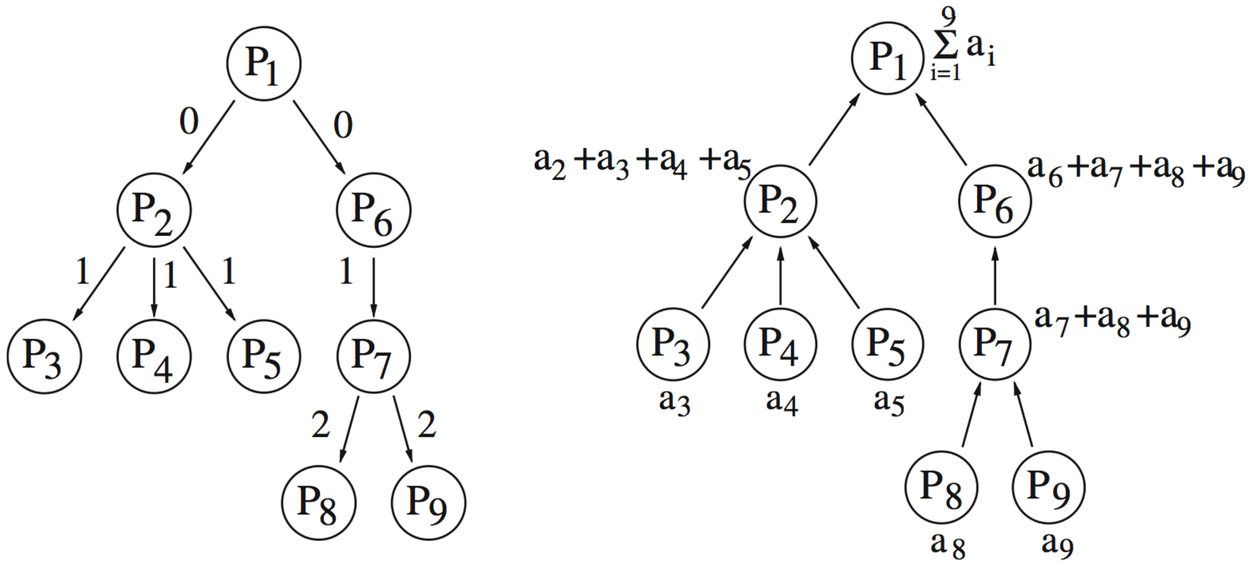
Each process sends data to all other processes.



Theoretical connections between the different collective communications

Duality

- Some communication operations are dual to each other.
- Communication operations are dual if one can be obtained by reversing the direction and the sequence of communication of the other.

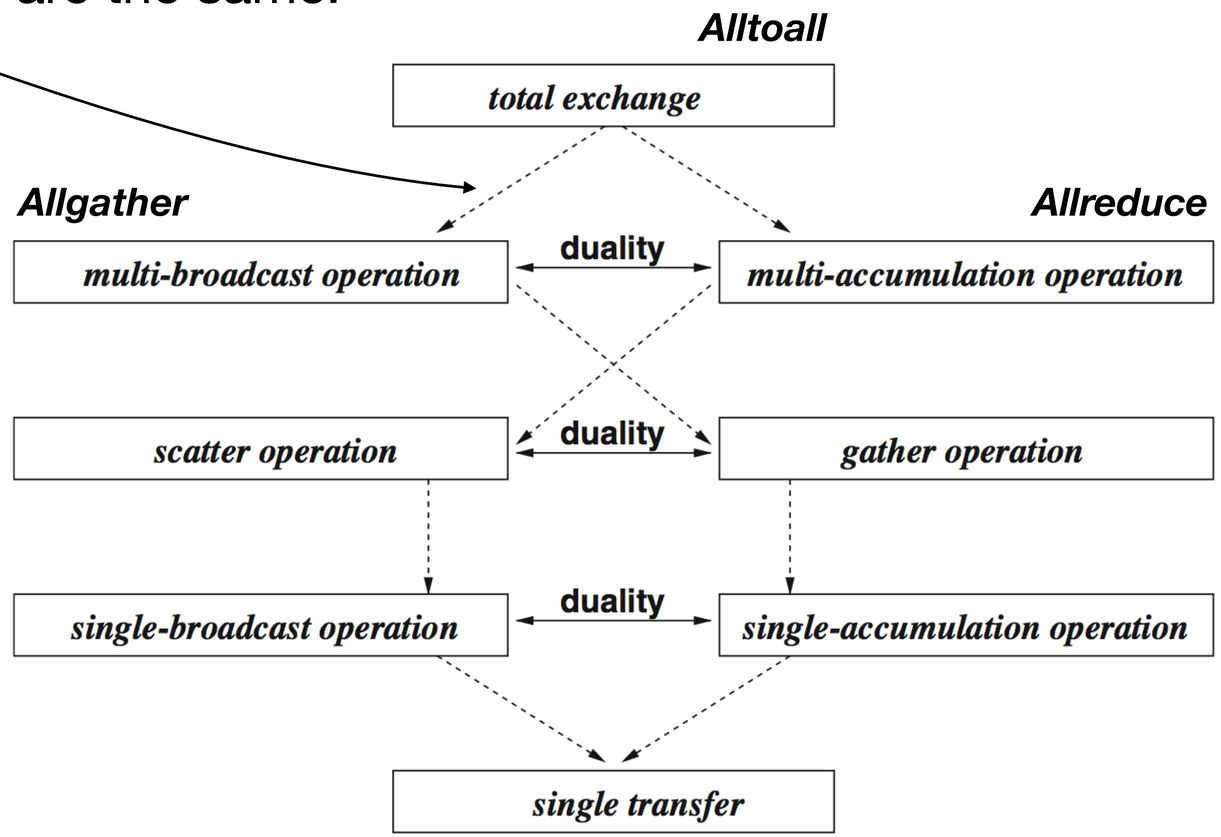


Left: single broadcast operation using a spanning tree.
Right: single accumulation that uses the same communication tree.

Relation between collective communication operations

Specialization, e.g., multi-broadcast is the same as total exchange if the p data blocks of a process are the same.

- Dual = “reverse” flow and direction of communications
- Important to understand performance and how to optimally implement these algorithms on different network topologies.



MPI process mapping

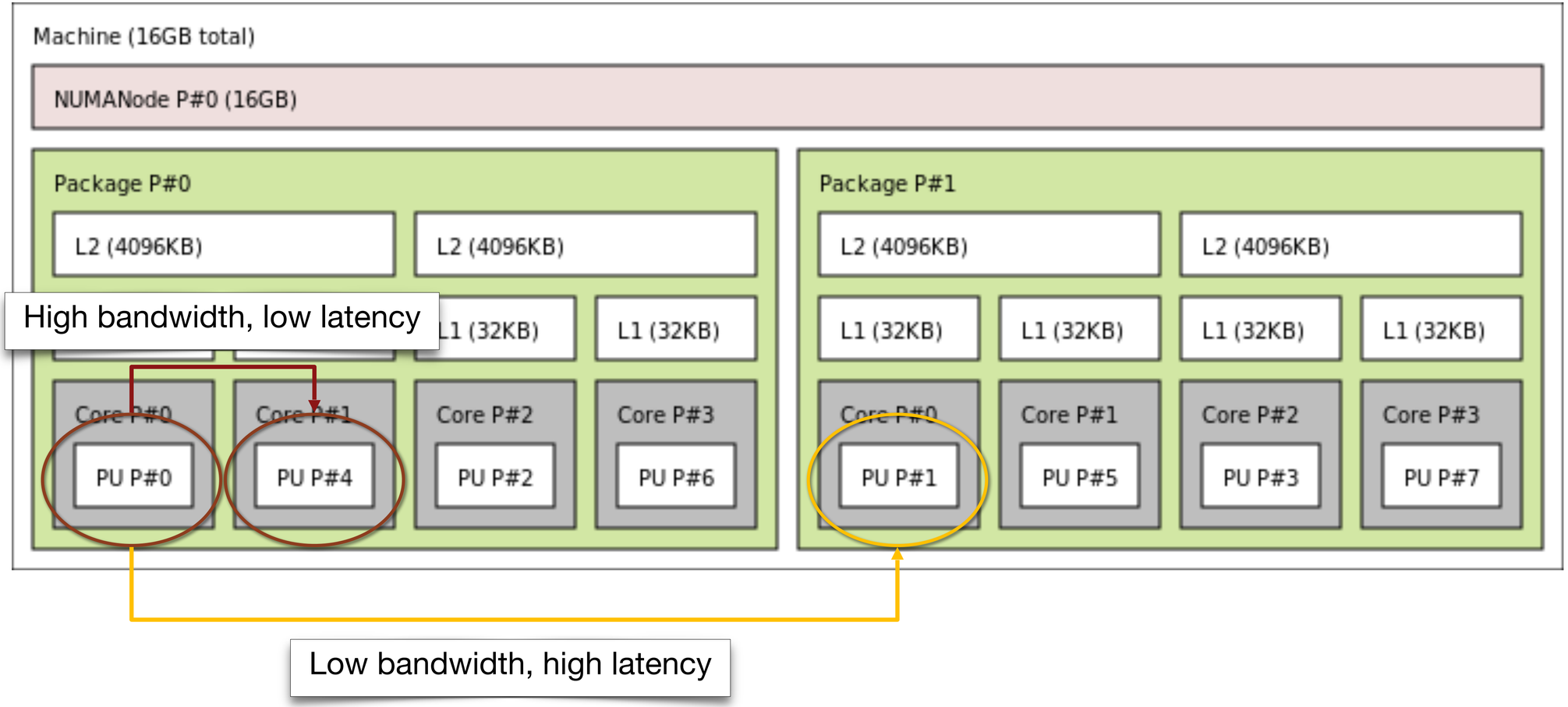
Process mapping

- It is important to control the binding and mapping of the processes.
- Mapping: assign a starting (default) location for a thread, e.g., which core or socket?
- Binding: the OS can move threads between cores or sockets. How should we restrict the movement of a thread?
  - a specific hardware thread
  - any hardware thread on a core
  - any hardware thread on a socket
  - any hardware thread on a NUMA domain

Why mapping? Optimization!

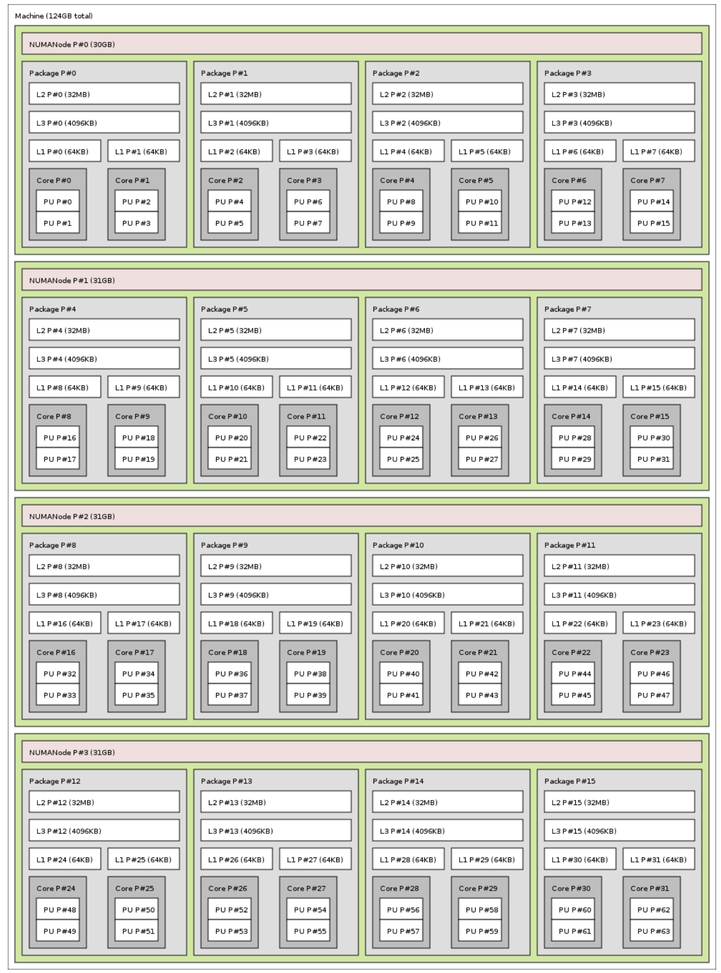
- The mapping of processes is important.
  - You may want to use more nodes to access more memory.
  - You may want to have all threads on the same socket for faster cache-sharing access.
  - You may want to have threads on different sockets to better utilize the memory buses (NUMA).
  - You may want to have as many processes per node as the number of GPU processors.
  - You may want to use OpenMP with your MPI code.
- General guidelines:
  - If two threads exchange a lot of data, they should be nearby in the system (high bandwidth).
  - If two threads make a lot of memory accesses, they should be in different NUMA nodes.

Example: 2-package quad-core Xeon (pre-Nehalem, with 2 dual-core dies into each package)



Example: PPC64-based system with 32 cores (each with 2 hardware threads)

- The topology of the network and data exchange between processes can become quickly very complicated.
- Profiling and tuning are required to find a good mapping.



Mapping/binding

Mapping relies on two concepts:

1. Mapping: assign process to hardware component by default

2. Binding: restrict the motion of processes between hardware components

Process binding

- OS is responsible for assigning a hardware thread to each MPI process.
- How do you control the placement of process threads?

`–bind–to object`

- Specifies the size of the hardware element to restrict each process thread to.
- This determines how the OS can migrate a process. Does the process stay with the same hardware thread or is it allowed to migrate to another thread (say on the same socket)?

`bind-to` options

Get all options using

```
$ mpirun -help {mapping,ranking,binding}
```

`mapping:` how to map processes to resources

`ranking:` how to assign ranks (ID) to processes

`binding:` restricting the movement of processes

Options for bind-to

Options	Action
`hwthread`	bind to hardware thread
`core`	bind to core
`l1cache`	bind to process on L1 cache domain
`l2cache`	bind to process on L2 cache domain
`l3cache`	bind to process on L3 cache domain
`socket`	bind to socket
`numa`	bind to NUMA domain
`board`	bind to motherboard

map-by

- `map-by` object
- Skip over the object between mapping.
- `slot, hwthread, core, L1cache, L2cache, L3cache, socket, numa, board, node`
- Usage example:

```
$ mpirun -bind-to core -map-by core -np 4 ./a.out
```

if you want information on bindings, and to assign more than one process per hardware thread.

https://www.open-mpi.org/doc/current/man1/mpirun.1.php

Examples from icme-gpu

```
darve@icme-gpu1:~/2023/MPI/Lecture_17$ salloc -n 16 -p CME mpirun --report-bindings -bind-to hwthread -map-by hwthread -np 16 ./mpi_binding
salloc: Granted job allocation 27645
salloc: Waiting for resource configuration
salloc: Nodes icmet01 are ready for job
[icmet01:629973] MCW rank 0 bound to socket 0[core 0[hwt 0]]: [B./../../../../../../..]
[icmet01:629973] MCW rank 1 bound to socket 0[core 0[hwt 1]]: [.B/../../../../../../..]
[icmet01:629973] MCW rank 2 bound to socket 0[core 1[hwt 0]]: [../B./../../../../../..]
[icmet01:629973] MCW rank 3 bound to socket 0[core 1[hwt 1]]: [../.B/../../../../../..]
[icmet01:629973] MCW rank 4 bound to socket 0[core 2[hwt 0]]: [../../B./../../../../..]
[icmet01:629973] MCW rank 5 bound to socket 0[core 2[hwt 1]]: [../../.B/../../../../..]
[icmet01:629973] MCW rank 6 bound to socket 0[core 3[hwt 0]]: [../../../B./../../../..]
[icmet01:629973] MCW rank 7 bound to socket 0[core 3[hwt 1]]: [../../../.B/../../../..]
[icmet01:629973] MCW rank 8 bound to socket 0[core 4[hwt 0]]: [../../../../B./../../..]
[icmet01:629973] MCW rank 9 bound to socket 0[core 4[hwt 1]]: [../../../../.B/../../..]
[icmet01:629973] MCW rank 10 bound to socket 0[core 5[hwt 0]]: [../../../../../B./../..]
[icmet01:629973] MCW rank 11 bound to socket 0[core 5[hwt 1]]: [../../../../../.B/../..]
[icmet01:629973] MCW rank 12 bound to socket 0[core 6[hwt 0]]: [../../../../../../B./..]
[icmet01:629973] MCW rank 13 bound to socket 0[core 6[hwt 1]]: [../../../../../../.B/..]
[icmet01:629973] MCW rank 14 bound to socket 0[core 7[hwt 0]]: [../../../../../../../B.]
[icmet01:629973] MCW rank 15 bound to socket 0[core 7[hwt 1]]: [../../../../../../../.B]
salloc: Relinquishing job allocation 27645
darve@icme-gpu1:~/2023/MPI/Lecture_17$
```

A process per hwthread.

Examples from icme-gpu

```
darve@icme-gpu1:~/2023/MPI/Lecture_17$ salloc -n 16 -p CME mpirun --report-bindings -bind-to hwthread -map-by core -np 16 ./mpi_binding
salloc: Granted job allocation 27647
salloc: Waiting for resource configuration
salloc: Nodes icmet01 are ready for job
[icmet01:630097] MCW rank 0 bound to socket 0[core 0[hwt 0]]: [B./../../../../../../..]
[icmet01:630097] MCW rank 1 bound to socket 0[core 1[hwt 0]]: [../B./../../../../../..]
[icmet01:630097] MCW rank 2 bound to socket 0[core 2[hwt 0]]: [../../B./../../../../..]
[icmet01:630097] MCW rank 3 bound to socket 0[core 3[hwt 0]]: [../../../B./../../../..]
[icmet01:630097] MCW rank 4 bound to socket 0[core 4[hwt 0]]: [../../../../B./../../..]
[icmet01:630097] MCW rank 5 bound to socket 0[core 5[hwt 0]]: [../../../../../B./../..]
[icmet01:630097] MCW rank 6 bound to socket 0[core 6[hwt 0]]: [../../../../../../B./..]
[icmet01:630097] MCW rank 7 bound to socket 0[core 7[hwt 0]]: [../../../../../../../B.]
[icmet01:630097] MCW rank 8 bound to socket 0[core 0[hwt 1]]: [.B/../../../../../../..]
[icmet01:630097] MCW rank 9 bound to socket 0[core 1[hwt 1]]: [../.B/../../../../../..]
[icmet01:630097] MCW rank 10 bound to socket 0[core 2[hwt 1]]: [../../.B/../../../../..]
[icmet01:630097] MCW rank 11 bound to socket 0[core 3[hwt 1]]: [../../../.B/../../../..]
[icmet01:630097] MCW rank 12 bound to socket 0[core 4[hwt 1]]: [../../../../.B/../../..]
[icmet01:630097] MCW rank 13 bound to socket 0[core 5[hwt 1]]: [../../../../../.B/../..]
[icmet01:630097] MCW rank 14 bound to socket 0[core 6[hwt 1]]: [../../../../../../.B/..]
[icmet01:630097] MCW rank 15 bound to socket 0[core 7[hwt 1]]: [../../../../../../../.B]
salloc: Relinquishing job allocation 27647
darve@icme-gpu1:~/2023/MPI/Lecture_17$
```

Mapping per core, bind to hwthread.

Examples from icme-gpu

```
darve@icme-gpu1:~/2023/MPI/Lecture_17$ salloc -n 16 -p CME mpirun --report-bindings -bind-to numa -map-by hwthread -np 16 ./mpi_binding
salloc: Granted job allocation 27656
salloc: Waiting for resource configuration
salloc: Nodes icmet01 are ready for job
[icmet01:630495] MCW rank 0 bound to socket 0[core 0[hwt 0-1]], socket 0[core 1[hwt 0-1]]: [BB/BB/../../../../../..]
[icmet01:630495] MCW rank 1 bound to socket 0[core 0[hwt 0-1]], socket 0[core 1[hwt 0-1]]: [BB/BB/../../../../../..]
[icmet01:630495] MCW rank 2 bound to socket 0[core 0[hwt 0-1]], socket 0[core 1[hwt 0-1]]: [BB/BB/../../../../../..]
[icmet01:630495] MCW rank 3 bound to socket 0[core 0[hwt 0-1]], socket 0[core 1[hwt 0-1]]: [BB/BB/../../../../../..]
[icmet01:630495] MCW rank 4 bound to socket 0[core 2[hwt 0-1]], socket 0[core 3[hwt 0-1]]: [../../BB/BB/../../../..]
[icmet01:630495] MCW rank 5 bound to socket 0[core 2[hwt 0-1]], socket 0[core 3[hwt 0-1]]: [../../BB/BB/../../../..]
[icmet01:630495] MCW rank 6 bound to socket 0[core 2[hwt 0-1]], socket 0[core 3[hwt 0-1]]: [../../BB/BB/../../../..]
[icmet01:630495] MCW rank 7 bound to socket 0[core 2[hwt 0-1]], socket 0[core 3[hwt 0-1]]: [../../BB/BB/../../../..]
[icmet01:630495] MCW rank 8 bound to socket 0[core 4[hwt 0-1]], socket 0[core 5[hwt 0-1]]: [../../../../BB/BB/../..]
[icmet01:630495] MCW rank 9 bound to socket 0[core 4[hwt 0-1]], socket 0[core 5[hwt 0-1]]: [../../../../BB/BB/../..]
[icmet01:630495] MCW rank 10 bound to socket 0[core 4[hwt 0-1]], socket 0[core 5[hwt 0-1]]: [../../../../BB/BB/../..]
[icmet01:630495] MCW rank 11 bound to socket 0[core 4[hwt 0-1]], socket 0[core 5[hwt 0-1]]: [../../../../BB/BB/../..]
[icmet01:630495] MCW rank 12 bound to socket 0[core 6[hwt 0-1]], socket 0[core 7[hwt 0-1]]: [../../../../../../BB/BB]
[icmet01:630495] MCW rank 13 bound to socket 0[core 6[hwt 0-1]], socket 0[core 7[hwt 0-1]]: [../../../../../../BB/BB]
[icmet01:630495] MCW rank 14 bound to socket 0[core 6[hwt 0-1]], socket 0[core 7[hwt 0-1]]: [../../../../../../BB/BB]
[icmet01:630495] MCW rank 15 bound to socket 0[core 6[hwt 0-1]], socket 0[core 7[hwt 0-1]]: [../../../../../../BB/BB]
salloc: Relinquishing job allocation 27656
darve@icme-gpu1:~/2023/MPI/Lecture_17$
```

Bind to numa set.

Examples from icme-gpu

```
darve@icme-gpu1:~/2023/MPI/Lecture_17$ salloc -n 16 -p CME mpirun --report-bindings -bind-to hwthread -map-by numa -np 16 ./mpi_binding
salloc: Granted job allocation 27657
salloc: Waiting for resource configuration
salloc: Nodes icmet01 are ready for job
[icmet01:630576] MCW rank 0 bound to socket 0[core 0[hwt 0]]: [B./../../../../../../..]
[icmet01:630576] MCW rank 1 bound to socket 0[core 2[hwt 0]]: [../../B./../../../../..]
[icmet01:630576] MCW rank 2 bound to socket 0[core 4[hwt 0]]: [../../../../B./../../..]
[icmet01:630576] MCW rank 3 bound to socket 0[core 6[hwt 0]]: [../../../../../../B./..]
[icmet01:630576] MCW rank 4 bound to socket 0[core 0[hwt 1]]: [.B/../../../../../../..]
[icmet01:630576] MCW rank 5 bound to socket 0[core 2[hwt 1]]: [../../.B/../../../../..]
[icmet01:630576] MCW rank 6 bound to socket 0[core 4[hwt 1]]: [../../../../.B/../../..]
[icmet01:630576] MCW rank 7 bound to socket 0[core 6[hwt 1]]: [../../../../../../.B/..]
[icmet01:630576] MCW rank 8 bound to socket 0[core 1[hwt 0]]: [../B./../../../../../..]
[icmet01:630576] MCW rank 9 bound to socket 0[core 3[hwt 0]]: [../../../B./../../../..]
[icmet01:630576] MCW rank 10 bound to socket 0[core 5[hwt 0]]: [../../../../../B./../..]
[icmet01:630576] MCW rank 11 bound to socket 0[core 7[hwt 0]]: [../../../../../../../B.]
[icmet01:630576] MCW rank 12 bound to socket 0[core 1[hwt 1]]: [../.B/../../../../../..]
[icmet01:630576] MCW rank 13 bound to socket 0[core 3[hwt 1]]: [../../../.B/../../../..]
[icmet01:630576] MCW rank 14 bound to socket 0[core 5[hwt 1]]: [../../../../../.B/../..]
[icmet01:630576] MCW rank 15 bound to socket 0[core 7[hwt 1]]: [../../../../../../../.B]
salloc: Relinquishing job allocation 27657
darve@icme-gpu1:~/2023/MPI/Lecture_17$
```

Map by numa set.

Examples from icme-gpu

```
darve@icme-gpu1:~/2023/MPI/Lecture_17$ salloc -n 16 -p CME mpirun --report-bindings -np 4 ./mpi_binding
salloc: Granted job allocation 27664
salloc: Waiting for resource configuration
salloc: Nodes icmet02 are ready for job
[icmet02:163820] MCW rank 0 bound to socket 0[core 0[hwt 0-1]], socket 0[core 1[hwt 0-1]]: [BB/BB/../../../../../..]
[icmet02:163820] MCW rank 1 bound to socket 0[core 2[hwt 0-1]], socket 0[core 3[hwt 0-1]]: [../../BB/BB/../../../..]
[icmet02:163820] MCW rank 2 bound to socket 0[core 4[hwt 0-1]], socket 0[core 5[hwt 0-1]]: [../../../../BB/BB/../..]
[icmet02:163820] MCW rank 3 bound to socket 0[core 6[hwt 0-1]], socket 0[core 7[hwt 0-1]]: [../../../../../../BB/BB]
salloc: Relinquishing job allocation 27664
darve@icme-gpu1:~/2023/MPI/Lecture_17$
```

Default binding; `–bind–to numa –map–by numa`

hwloc

- `hwloc` is a command to query the organization of the cores, cache, memory, and sockets on a node.
- mpirun uses `hwloc` for mapping and binding.
- The command `hwloc-ls` displays the topology of the node.

Example output on icme-gpu

```
$ salloc -p CME hwloc-ls
```

https://www.open-mpi.org/projects/hwloc/doc/v2.9.1/

```
Machine (251GB total)
  Package L#0
    NUMANode L#0 (P#0 251GB)
    L3 L#0 (12MB)
      L2 L#0 (1280KB) + L1d L#0 (48KB) + L1i L#0 (32KB) + Core L#0
        PU L#0 (P#0)
        PU L#1 (P#8)
      L2 L#1 (1280KB) + L1d L#1 (48KB) + L1i L#1 (32KB) + Core L#1
        PU L#2 (P#1)
        PU L#3 (P#9)
      L2 L#2 (1280KB) + L1d L#2 (48KB) + L1i L#2 (32KB) + Core L#2
        PU L#4 (P#2)
        PU L#5 (P#10)
      L2 L#3 (1280KB) + L1d L#3 (48KB) + L1i L#3 (32KB) + Core L#3
        PU L#6 (P#3)
        PU L#7 (P#11)
      L2 L#4 (1280KB) + L1d L#4 (48KB) + L1i L#4 (32KB) + Core L#4
        PU L#8 (P#4)
        PU L#9 (P#12)
      L2 L#5 (1280KB) + L1d L#5 (48KB) + L1i L#5 (32KB) + Core L#5
        PU L#10 (P#5)
        PU L#11 (P#13)
      L2 L#6 (1280KB) + L1d L#6 (48KB) + L1i L#6 (32KB) + Core L#6
        PU L#12 (P#6)
        PU L#13 (P#14)
      L2 L#7 (1280KB) + L1d L#7 (48KB) + L1i L#7 (32KB) + Core L#7
        PU L#14 (P#7)
        PU L#15 (P#15)
```

MPI point-to-point communication

MPI buffers

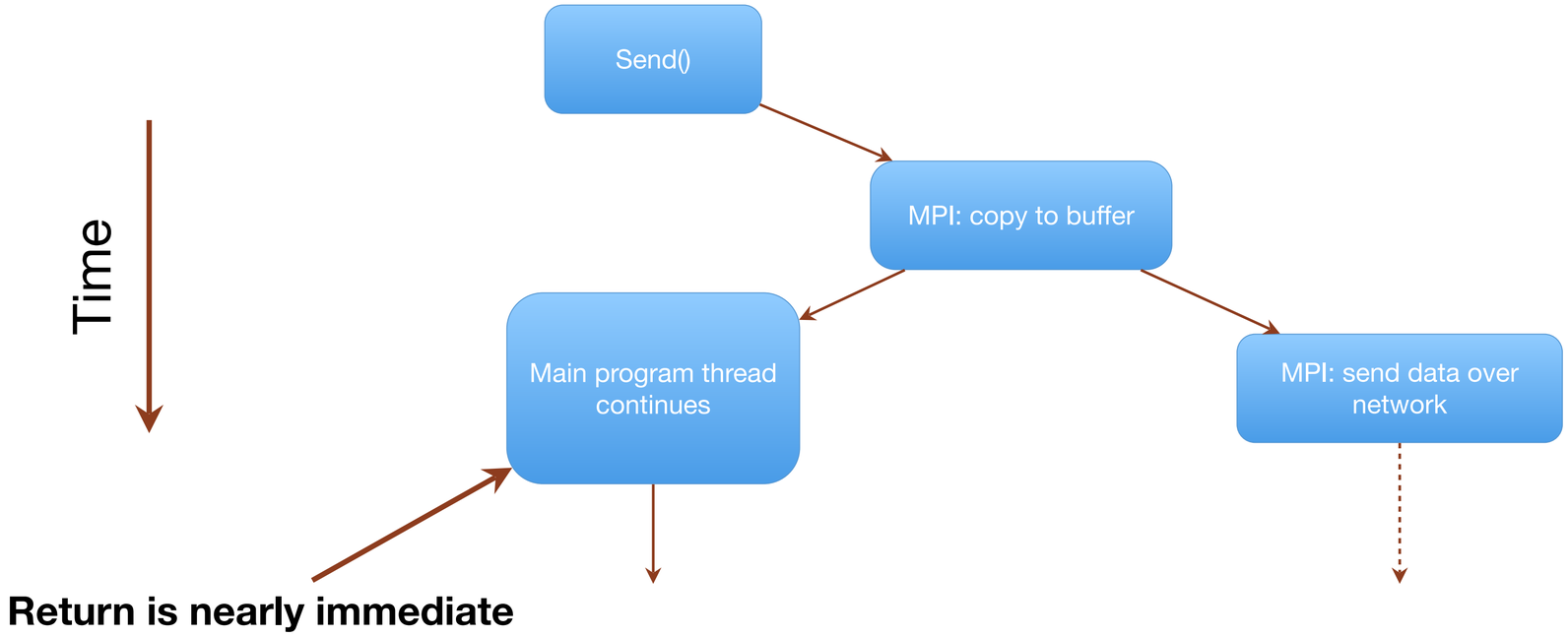
Two strategies:

1. Buffered: send/receive appear to complete immediately

2. Non-buffered: saves memory but requires waiting

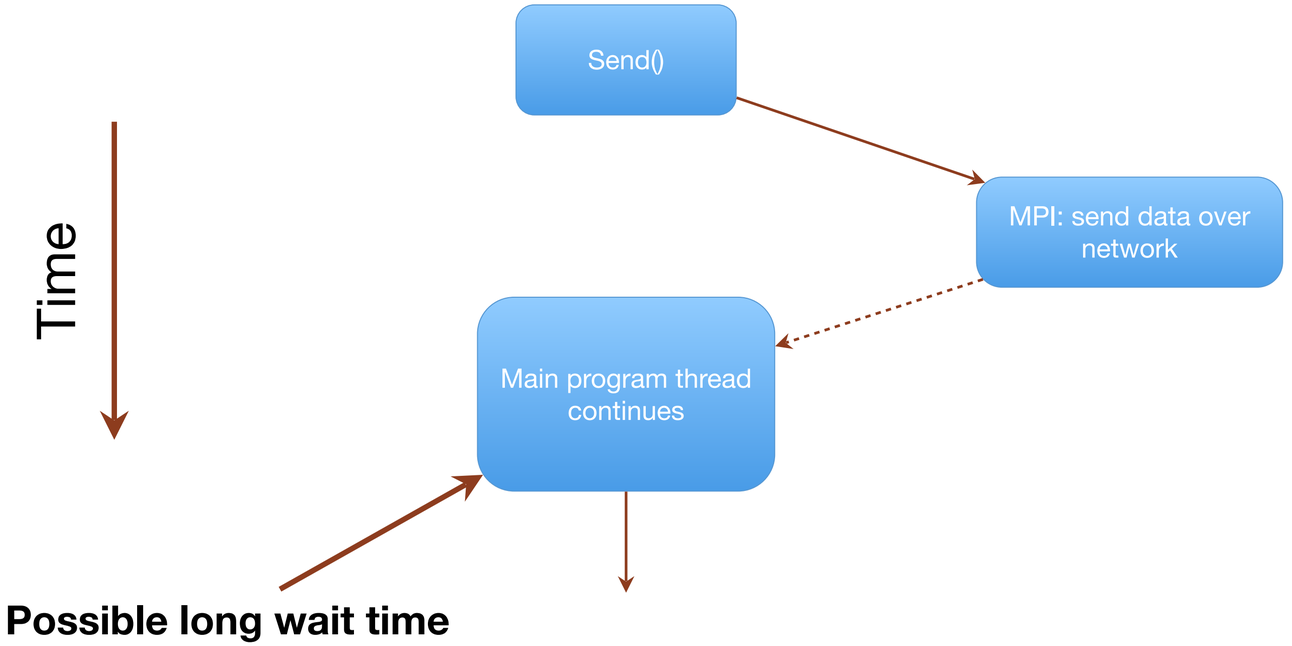
Using an MPI system buffer

To optimize the communication the MPI library uses two different strategies for communication: buffered and non-buffered.



Send using a buffer

Without an MPI system buffer



Main thread cannot make progress until the communication is completed

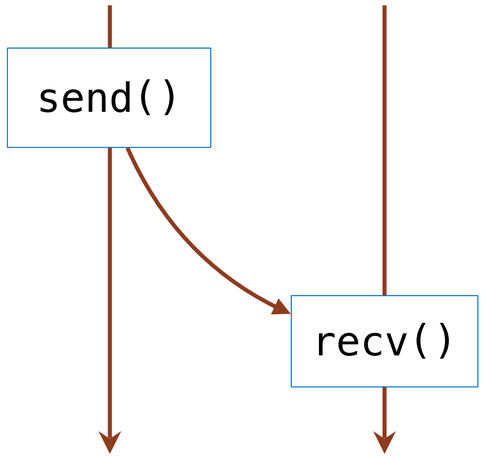
Summary of difference

- Send and Recv are blocking operations:
  - The call does not return until the resources are available again
  - Send: data in buffer can be changed
  - Recv: data in buffer is available and can be used
- Send—If MPI uses a separate system buffer, the data in `smessage` (user buffer space) is copied (fast); then the main thread resumes.
- If MPI does not use a separate system buffer, the main thread must wait until the communication over the network is complete.
- Recv—If communication happens before the call, the data is stored in an MPI system buffer, and then simply copied into the user provided `rmessage` when `MPI_Recv()` is called.
- The user cannot decide whether a buffer is used or not; the MPI library makes that decision based on the resources available and other factors.

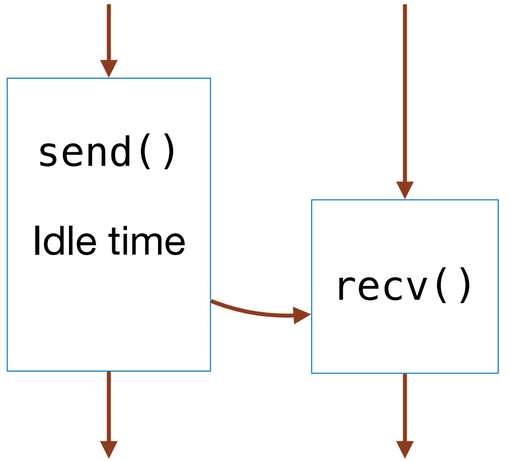
Example execution with and without buffering

Copy to buffer and resume

With MPI buffer



Without MPI buffer



No buffer is used; the thread needs to wait for the recv()

Deadlocks

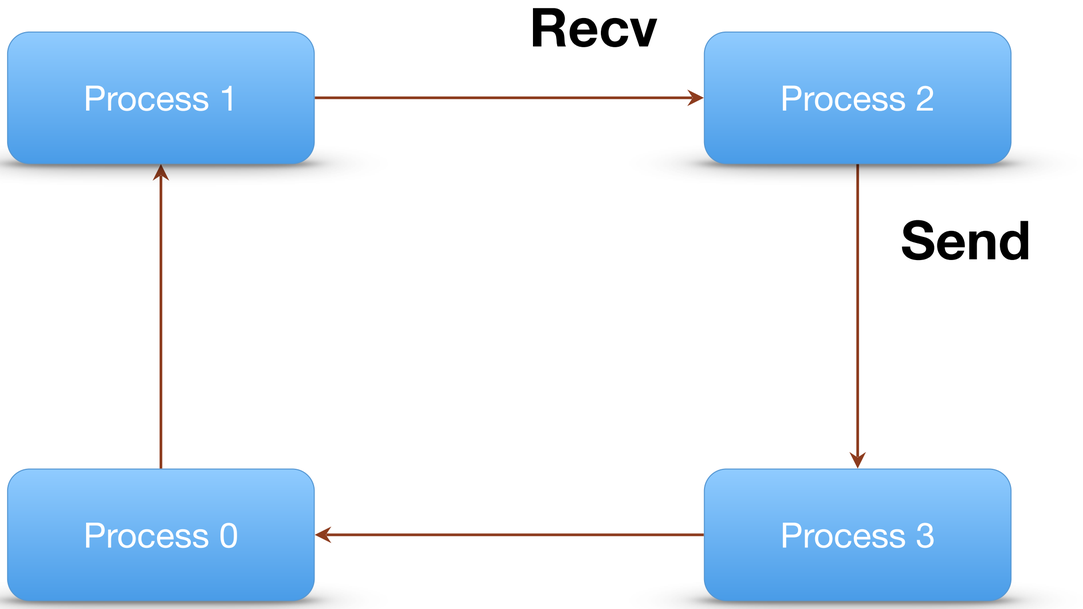
Deadlocks

- Because we use blocking routines, deadlocks can occur.
- **Secure implementation: code is guaranteed to never deadlock;** this should be true independent of whether buffers are used or not.

	Process 0	Process 1	Deadlock
	Recv() Send()	Recv() Send()	Always
	Send() Recv()	Send() Recv()	Depends on whether a buffer is used or not
	Send() Recv()	Recv() Send()	Secure

Example of deadlocks

Example: ring communication



Code with deadlock

```
// Receive from the lower process and send to the higher process.
int rank_sender = rank == 0 ? world_size - 1 : rank - 1;
MPI_Recv(&number_recv, 1, MPI_INT, rank_sender, 0, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
printf("Process %d received \t %2d from process %d\n", rank, number_recv,
        rank_sender);

int rank_receiver = rank == world_size - 1 ? 0 : rank + 1;
MPI_Send(&number_send, 1, MPI_INT, rank_receiver, 0, MPI_COMM_WORLD);
printf("Process %d sent \t\t %2d to   process %d\n", rank, number_send,
        rank_receiver);
```

Deadlock avoided with buffering

```
// Receive from the lower process and send to the higher process.
int rank_receiver = rank == world_size - 1 ? 0 : rank + 1;
MPI_Send(&number_send, 1, MPI_INT, rank_receiver, 0, MPI_COMM_WORLD);
printf("Process %d sent \t\t %2d to   process %d\n", rank, number_send,
    rank_receiver);

int rank_sender = rank == 0 ? world_size - 1 : rank - 1;
MPI_Recv(&number_recv, 1, MPI_INT, rank_sender, 0, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE);
printf("Process %d received \t %2d from process %d\n", rank, number_recv,
    rank_sender);
```

Correct code

```
if (rank % 2 == 0) {
  int rank_receiver = rank == world_size - 1 ? 0 : rank + 1;
  MPI_Send(&number_send, 1, MPI_INT, rank_receiver, 0, MPI_COMM_WORLD);
  printf("Process %d sent \t\t %2d to   process %d\n", rank, number_send,
         rank_receiver);
} else {
  int rank_sender = rank == 0 ? world_size - 1 : rank - 1;
  MPI_Recv(&number_recv, 1, MPI_INT, rank_sender, 0, MPI_COMM_WORLD,
           MPI_STATUS_IGNORE);
  printf("Process %d received \t %2d from process %d\n", rank, number_recv,
         rank_sender);
}

if (rank % 2 == 1) {
  int rank_receiver = rank == world_size - 1 ? 0 : rank + 1;
  MPI_Send(&number_send, 1, MPI_INT, rank_receiver, 0, MPI_COMM_WORLD);
  printf("Process %d sent \t\t %2d to   process %d\n", rank, number_send,
         rank_receiver);
} else {
  int rank_sender = rank == 0 ? world_size - 1 : rank - 1;
  MPI_Recv(&number_recv, 1, MPI_INT, rank_sender, 0, MPI_COMM_WORLD,
           MPI_STATUS_IGNORE);
  printf("Process %d received \t %2d from process %d\n", rank, number_recv,
         rank_sender);
}
```

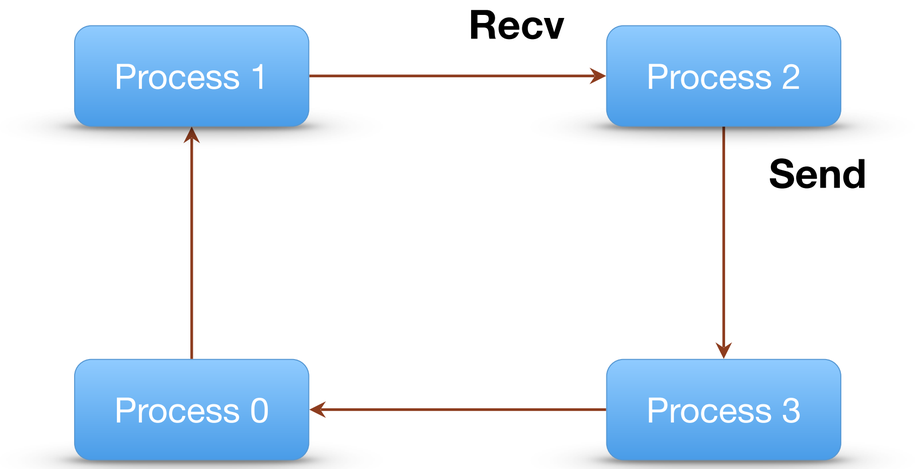
Variant using `MPI_Sendrecv`

```
int rank_receiver = rank == world_size - 1 ? 0 : rank + 1;
int rank_sender = rank == 0 ? world_size - 1 : rank - 1;
MPI_Sendrecv(&number_send, 1, MPI_INT, rank_receiver, 0, &number_recv, 1,
             MPI_INT, rank_sender, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

MPI_Sendrecv

- The send-receive operation combines in one call the sending of a message to one destination and the receiving of another message, from another process.
- The two (source and destination) are possibly the same.
- This is useful for executing a **shift operation** across a chain of processes.
- If blocking sends and receives are used for such a shift, then one needs to order the sends and receives correctly (for example, even processes send, then receive; odd processes receive first, then send) in order to prevent cyclic dependencies that lead to deadlock.
- **When a send-receive operation is used, the communication subsystem takes care of these issues.**

MPI_Sendrecv




```
MPI_Sendrecv(void *sendbuf, int sendcount,
     MPI_Datatype sendtype,
     int dest, int sendtag,
     void *recvbuf, int recvcount,
     MPI_Datatype recvtype,
     int source, int recvtag,
     MPI_Comm comm, MPI_Status *status)
```