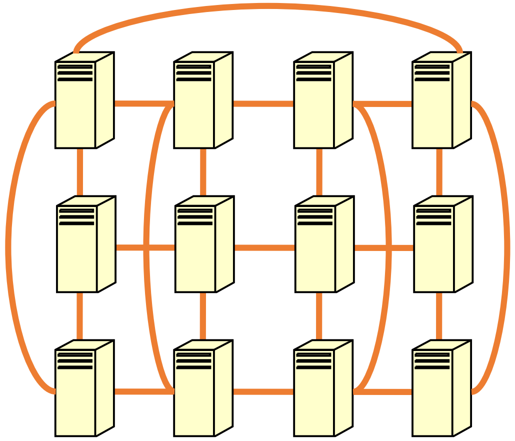
MPI-II

Jan 15, 2019

Recap of Basic Concepts and Terminology

- Message passing
- Node
- Cluster
- Process
- Network topology
- MPI Process
- Rank
- Message
- Message tag, source, destination
- Communicator
- MPI point-to-point communication

Torus Interconnect (Correction)



What is the advantage of torus over mesh?

- Diameter 2(√p/2)
- Bisection width 2√p
- Cost 2p

Example 4 (from previous class)

```
if (myrank == 0 || myrank == 2)
   MPI_Send(arr, 20, MPI_INT, 1, 99, MPI_COMM_WORLD);
else if (myrank == 1)
{
   int count, recvarr[3][20];
   MPI_Recv(recvarr[0], 20, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
   printf("Rank %d of %d received from rank %d\n", myrank, size,
status.MPI_SOURCE);
   MPI_Recv(recvarr[2], 20, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
   printf("Rank %d of %d received from rank %d\n", myrank, size,
status.MPI_SOURCE);
 }
```

Is it the right way?

Is it the right way?

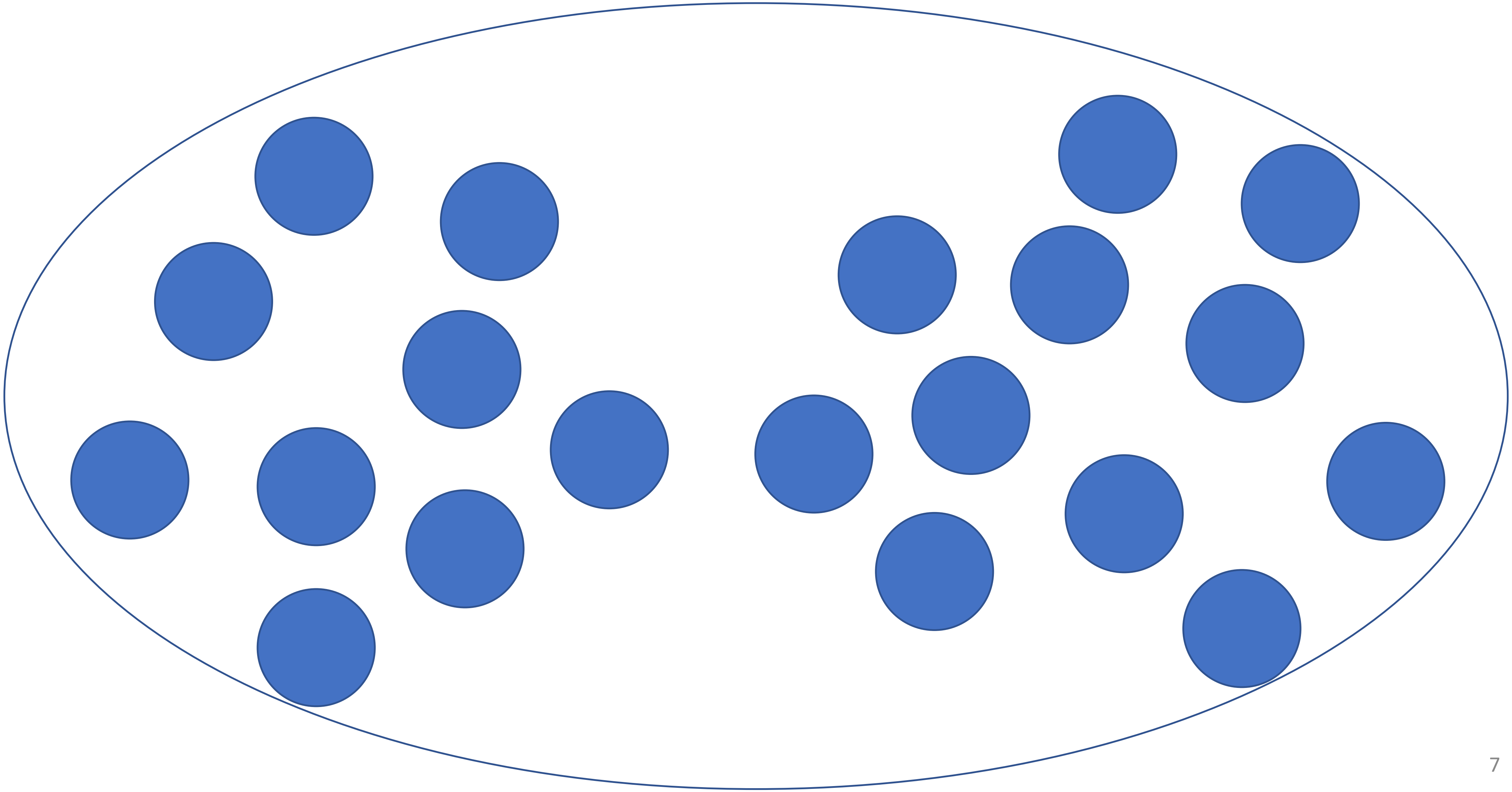
MPI_Recv Error Handling

```
Fatal error in MPI_Recv: Message truncated, error stack:

MPI_Recv(200)....................: MPI_Recv(buf=0x7ffdb75742b0,
count=40, MPI_INT, src=0, tag=99, MPI_COMM_WORLD,
status=0x7ffdb7574290)                              failed

MPIDI_CH3U_Receive_data_found(131): Message from rank 0 and tag
99 truncated; 320 bytes received but buffer size is 160
```

Communicator

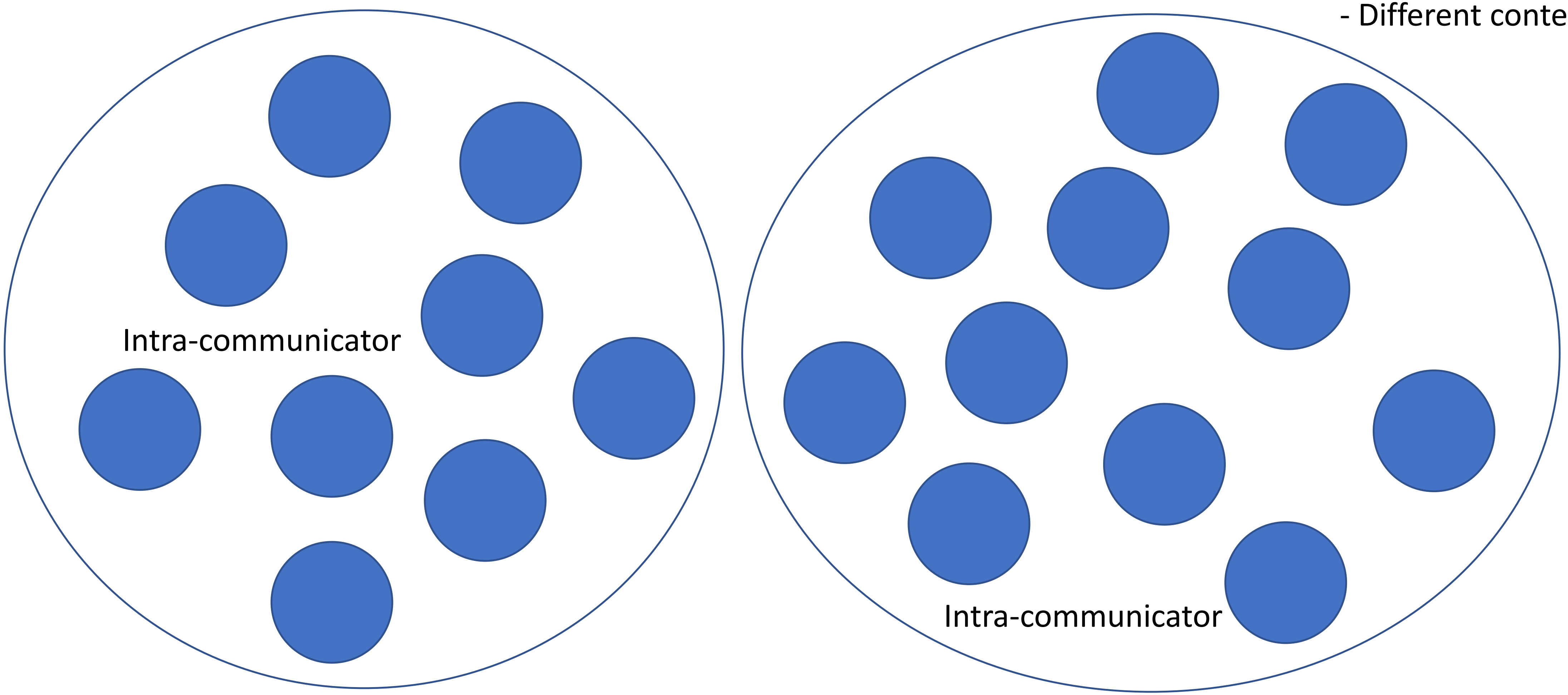
- Object containing a group of processes
- Representative of communication domain
- Associated with a context ID (in MPICH)
- Predefined:
  - MPI_COMM_WORLD (when is it defined?)
  - MPI_COMM_SELF
- Contains a mapping from MPI process ranks to processor ids
- Memory proportional to #processes in the group

MPI_COMM_WORLD



Subcommunicator

- Logical subset
- Different contexts

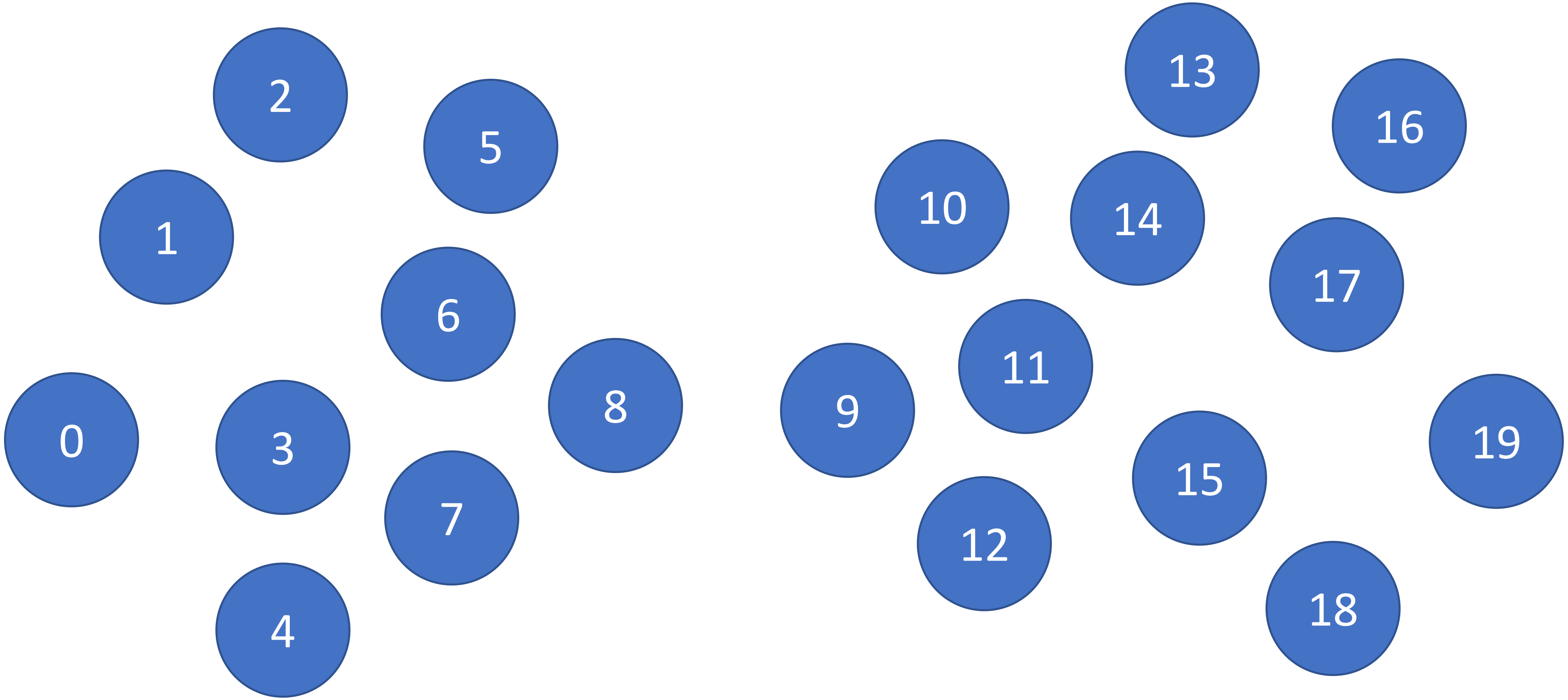


MPI_COMM_SPLIT

MPI_Comm_split (MPI_Comm oldcomm, int color, int key, MPI_Comm *newcomm)

- Collective call
- Logically divides based on *color*
  - Same color processes form a group
  - Some processes may not be part of newcomm (MPI_UNDEFINED)
- Rank assignment based on *key*

Logical subsets of processes



0 → 0
2 → 1
4 → 2
…

1 → 0
3 → 1
5 → 2
…

How do you assign one color to odd processes and another color to even processes ?
color = rank % 2

Example code

```
int newrank, newsize, color = myrank%3;
MPI_Comm newcomm;

MPI_Comm_split (MPI_COMM_WORLD, color, myrank, &newcomm);

MPI_Comm_rank (newcomm, &newrank);
MPI_Comm_size (newcomm, &newsize);
printf ("%d: %d of %d\n", myrank, newrank, newsize);

MPI_Comm_free (&newcomm);
```

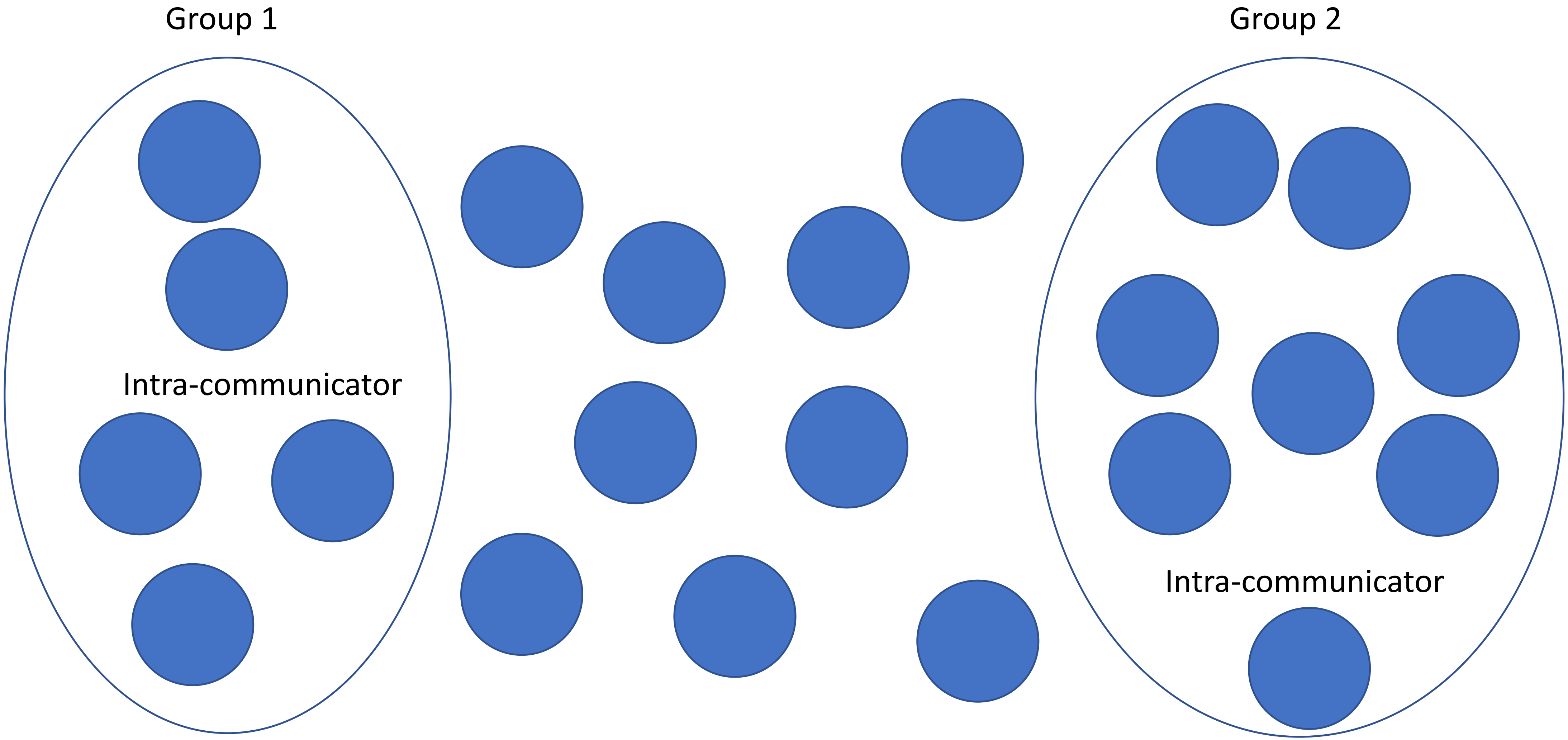
OUTPUT for n=9

```
0:  0 of 3
1:  0 of 3
2:  0 of 3
3:  1 of 3
4:  1 of 3
5:  1 of 3
6:  2 of 3
7:  2 of 3
8:  2 of 3
```

Process Group

- Ordered set of processes
- Ranks are contiguous
- Base group – associated with MPI_COMM_WORLD
- MPI_Group object
    - MPI_Group_rank, MPI_Group_size
- Unions and intersections of groups
- Not used in communication context

Inter and Intra Communicators



Collective Communications

- Must be called by all processes that are part of the communicator

Types

- Synchronization (MPI_Barrier)
- Global communication (MPI_Bcast, MPI_Gather, …)
- Global reduction (MPI_Reduce, ..)

Barrier

- MPI_Barrier (comm)
- Collective call
- Caller returns only after all processes have entered the call

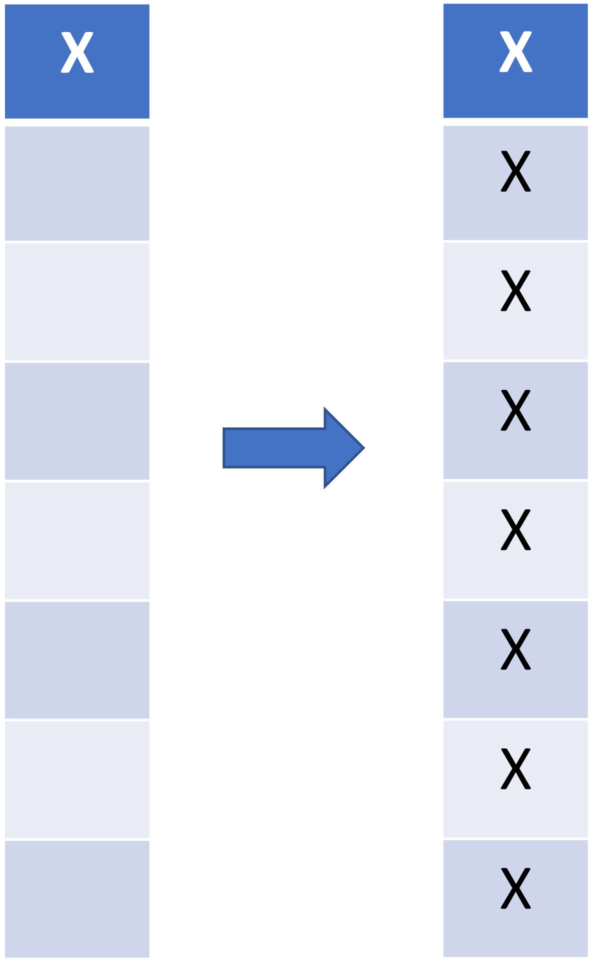
n=4

```
if (myrank != 0)
   MPI_Barrier (MPI_COMM_WORLD);
printf(“%d\n”, rank);
```

Output?

Broadcast

- Root process sends message to all processes
- Any process can be root process but has to be the same
- int MPI_Bcast (buffer, count, datatype, root, comm)
- Number of elements in buffer – count
- Tag?
- buffer – Input or output?

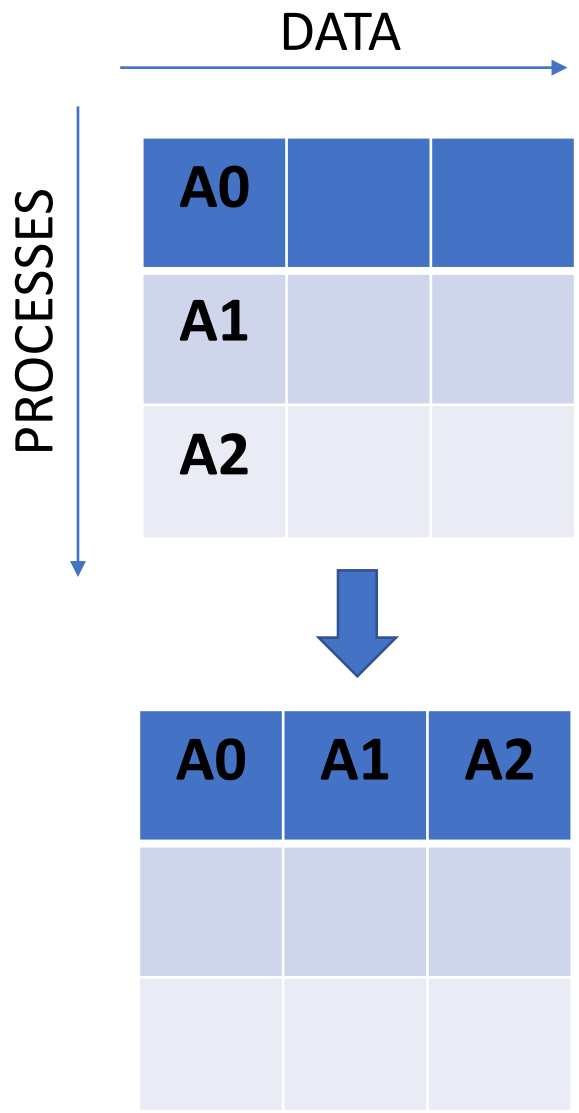


Q1: Can you use point-to-point communication for the same?

Q2: Buffer size of "buf" array is not known apriori at non-root processes, how should root broadcast buf?

Gather

- Gathers values from all processes to a root process
- int MPI_Gather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
- Arguments recv* not relevant on non-root processes

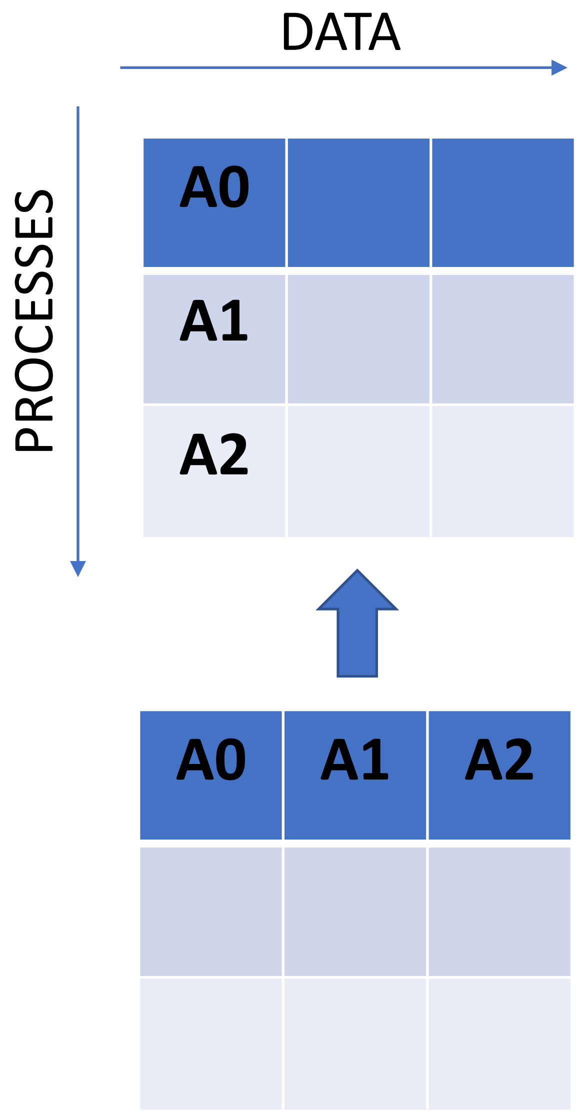


Q: Equivalent point-to-point communications for the same?

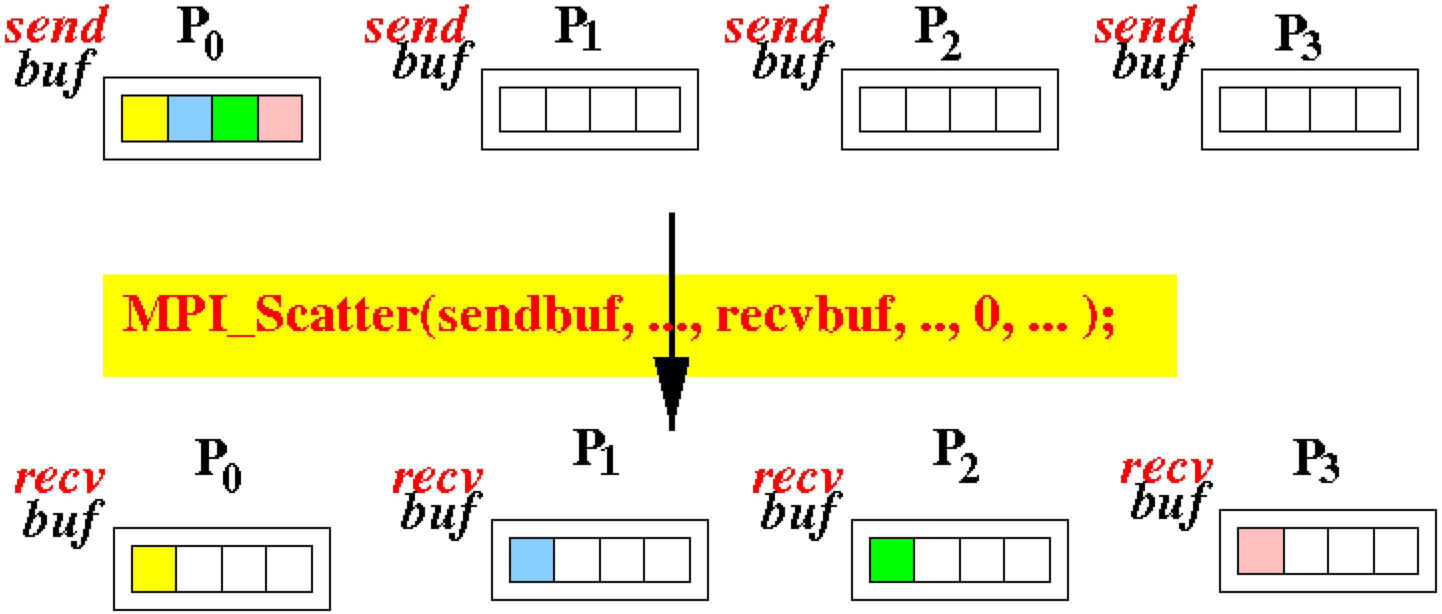
- MPI_Recv at root
- MPI_Send at non-root

Scatter

- Scatters values to all processes from a root process
- int MPI_Scatter (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
- Arguments send* not relevant on non-root processes
- Output parameter – recvbuf



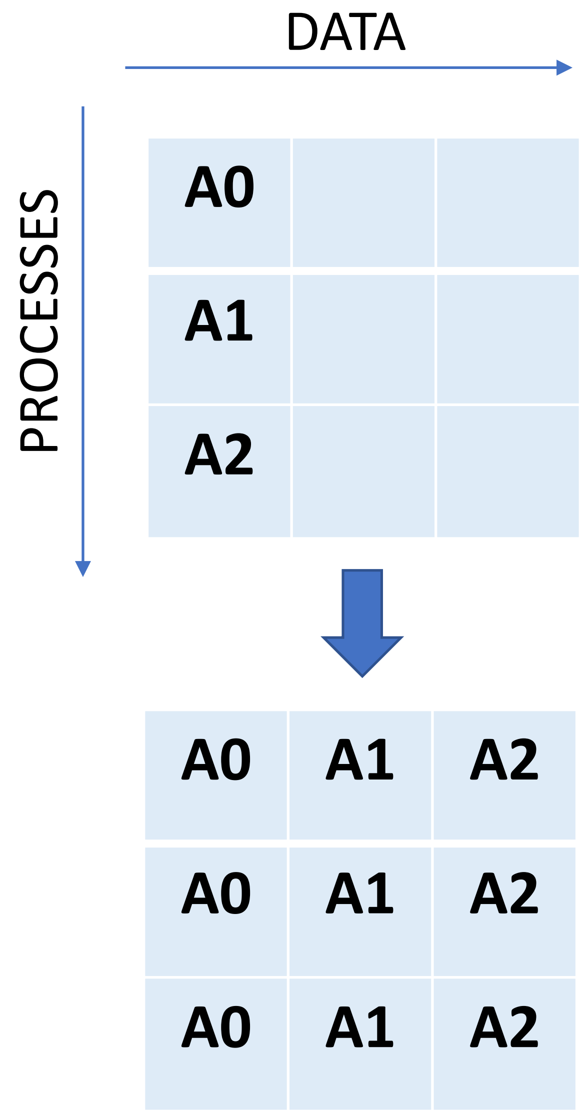
MPI_Scatter Illustration



Credit: Shun Yan Cheung

Allgather

- All processes gather values from all processes
- int MPI_Allgather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)



DATA

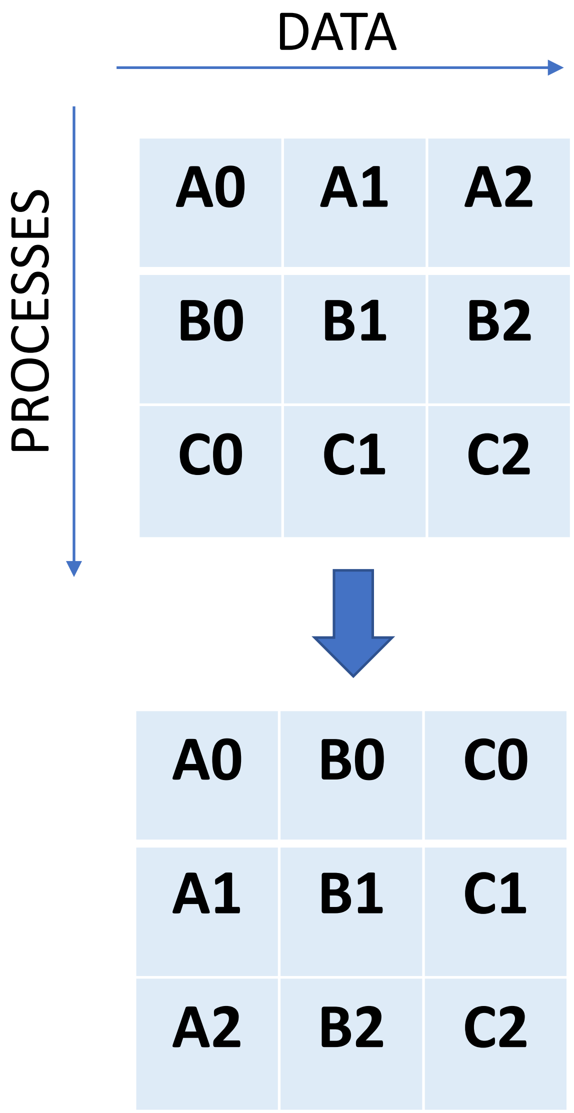
PROCESSES

A0		
A1		
A2		

A0	A1	A2
A0	A1	A2
A0	A1	A2

Alltoall

- Send data from all processes to all processes
- int MPI_Alltoall (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
- Output parameter – recvbuf



A0	A1	A2
B0	B1	B2
C0	C1	C2

A0	B0	C0
A1	B1	C1
A2	B2	C2

Equivalent collective?

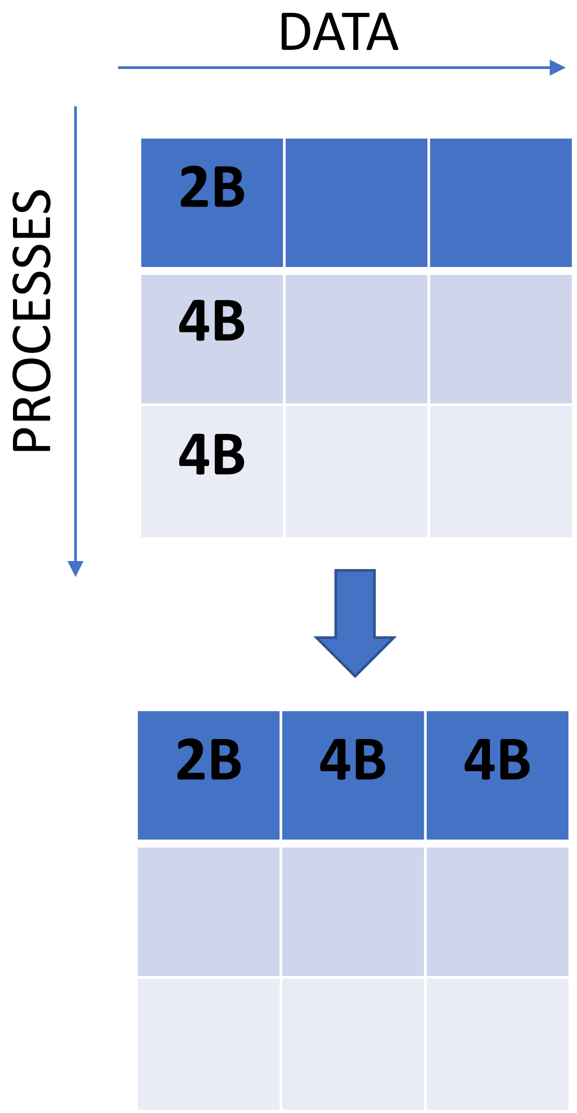
- MPI_Scatter at all processes
- Cons?

Gatherv

- Root gathers values of different lengths from all processes
- int MPI_Gatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm)
- recvcounts – Number of elements to be received from each process
- displs – Displacement at which to place received data

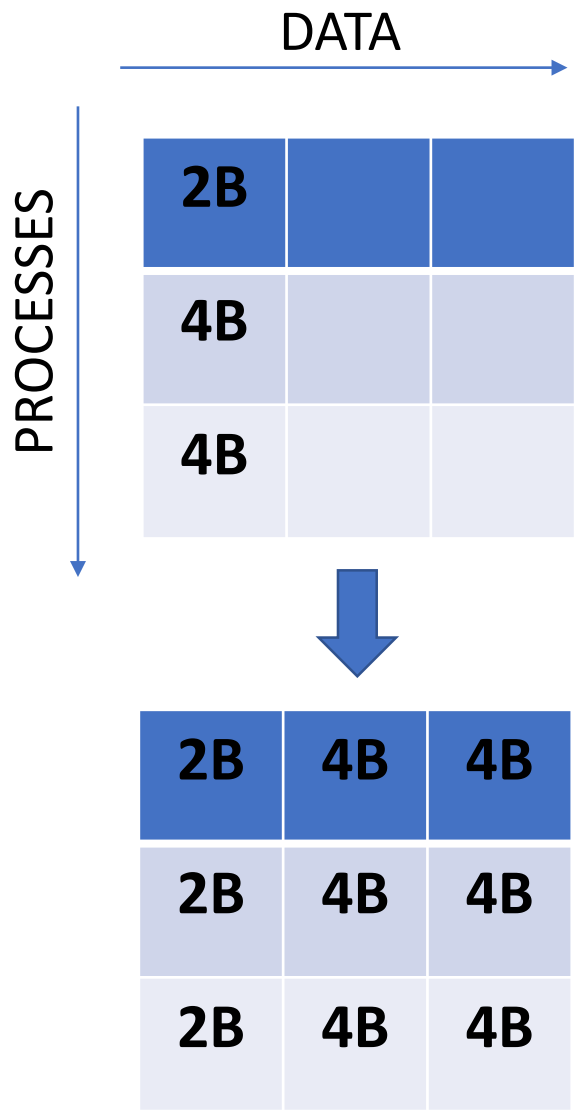
MPI_Recv (recvbuf+displs[i], recvcounts[i], recvtype, i, i, comm, &status) at root

MPI_Send at non-root



Allgatherv

- All processes gather values of different lengths from all processes
- int MPI_Allgatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm)
- recvcounts – Number of elements to be received from each process
- displs – Displacement at which to place received data



DATA

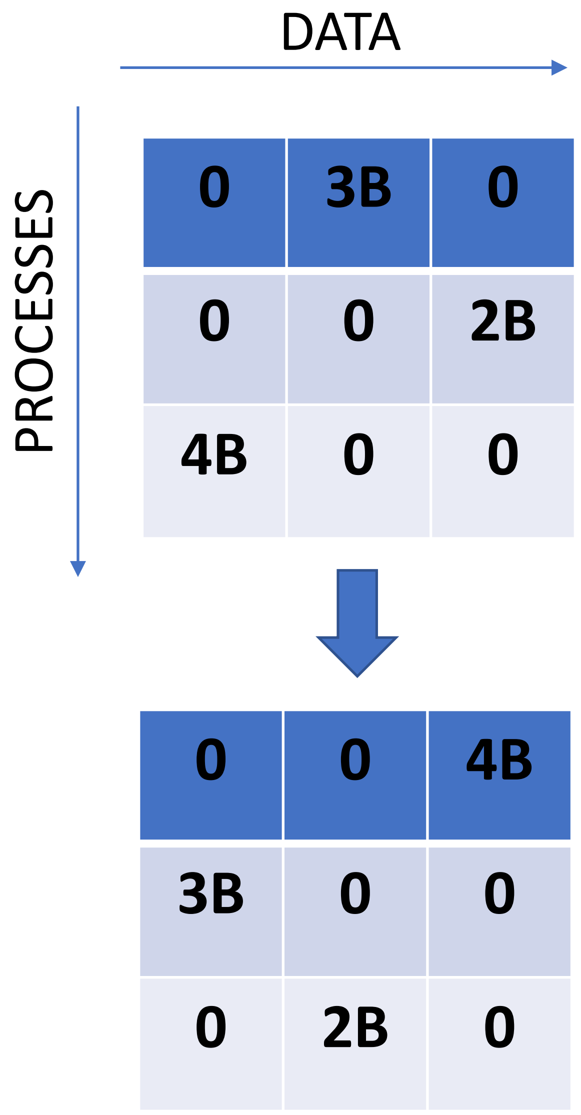
PROCESSES

2B		
4B		
4B		

2B	4B	4B
2B	4B	4B
2B	4B	4B

Alltoallv

- Every process sends data of different lengths to other processes
- int MPI_Alltoallv (sendbuf, sendcount, sdispls, sendtype, recvbuf, recvcount, rdispls, recvtype, comm)
- Output parameter – recvbuf



0	3B	0
0	0	2B
4B	0	0

0	0	4B
3B	0	0
0	2B	0

Assignments

- Directory named AssignmentN (not assignment, Assignment- etc.)
- Reporter access at least (not guest)
- Indicate your choice of score reduction or utilizing extra day
- Start coding early
  - Parallel programs are harder to debug!
  - Your success depends on system availability

Assignment 1

1.1: Implement a modified version of the classic producer-consumer problem on distributed memory systems using N processes. You may choose P producers and C consumers, where P+C=N and P=C=N/2. Assume that each producer produces D bytes of data (use doubles as datatype). Assume that there is a one-to-one mapping between producers and consumers.

1.2: Implement 1.1 using MPI collectives.

1.3: (Bonus question) Choose the P producer processes and P→C mapping optimally among the N processes such that performance is expected to improve.