Advanced MPI

Dec 8, 2019

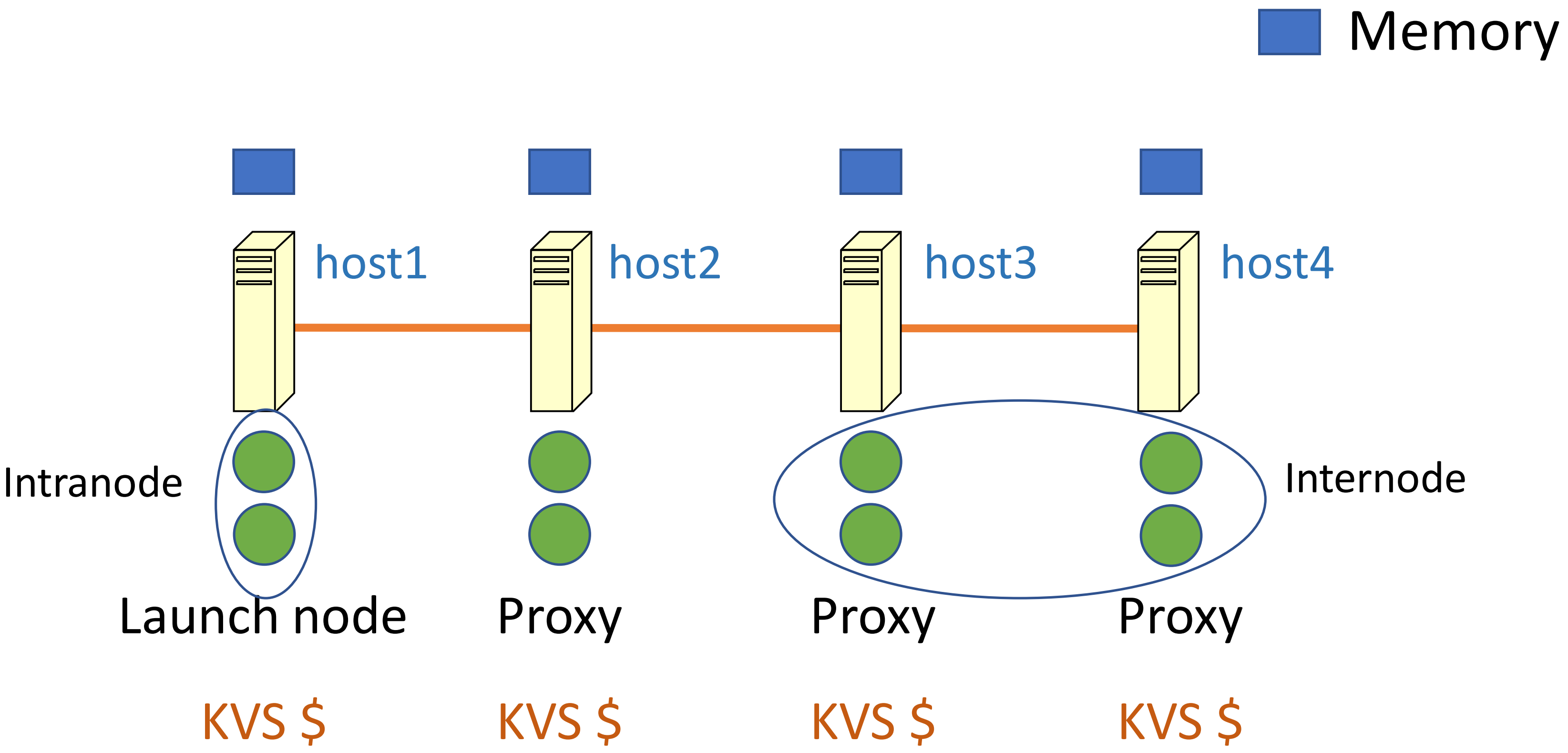
Recap

Program order

x = 1, y = 2	x = 2, y = 3	x = 3, y = 4
. . .	. . .	. . .
x++	y++	x++; y++
. . .	. . .	. . .
print x, y	print x, y	print x, y
2, 2	2, 4	4, 5

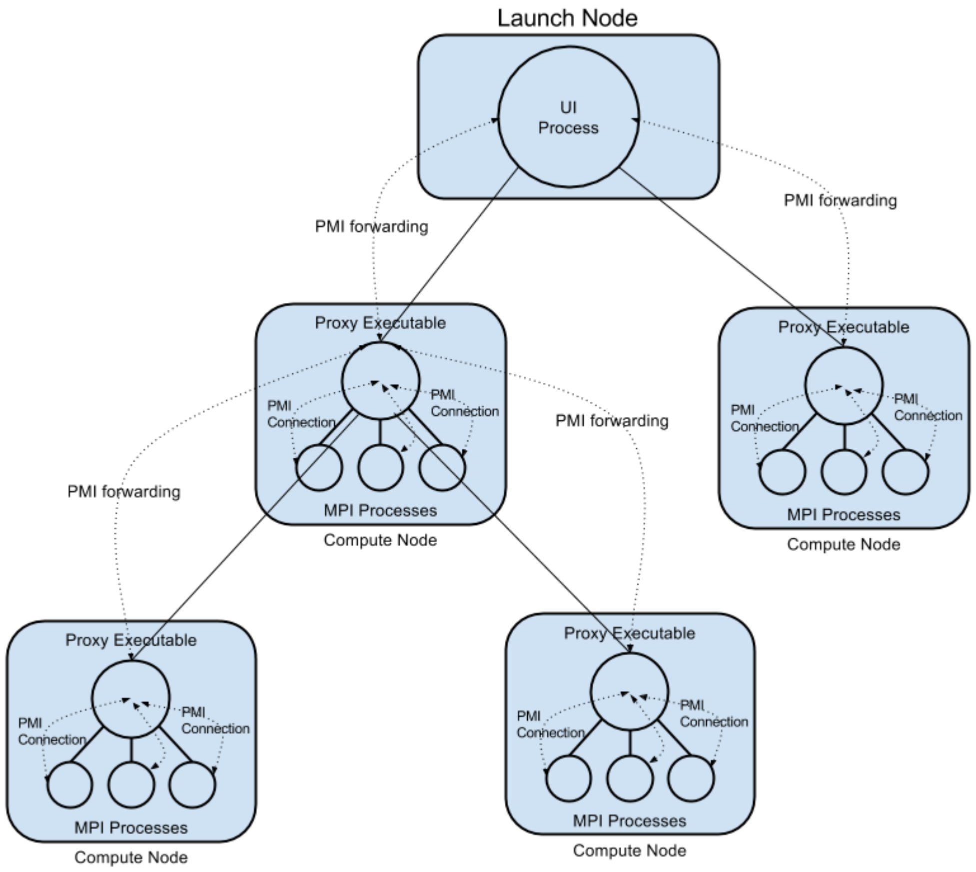
Distinct address space

Recap – Hydra Process Manager



mpiexec –n 4 –hosts host1,host2,host3,host4 ./exe

Hydra Process Manager



Source: wiki.mpich.org

Launch Node

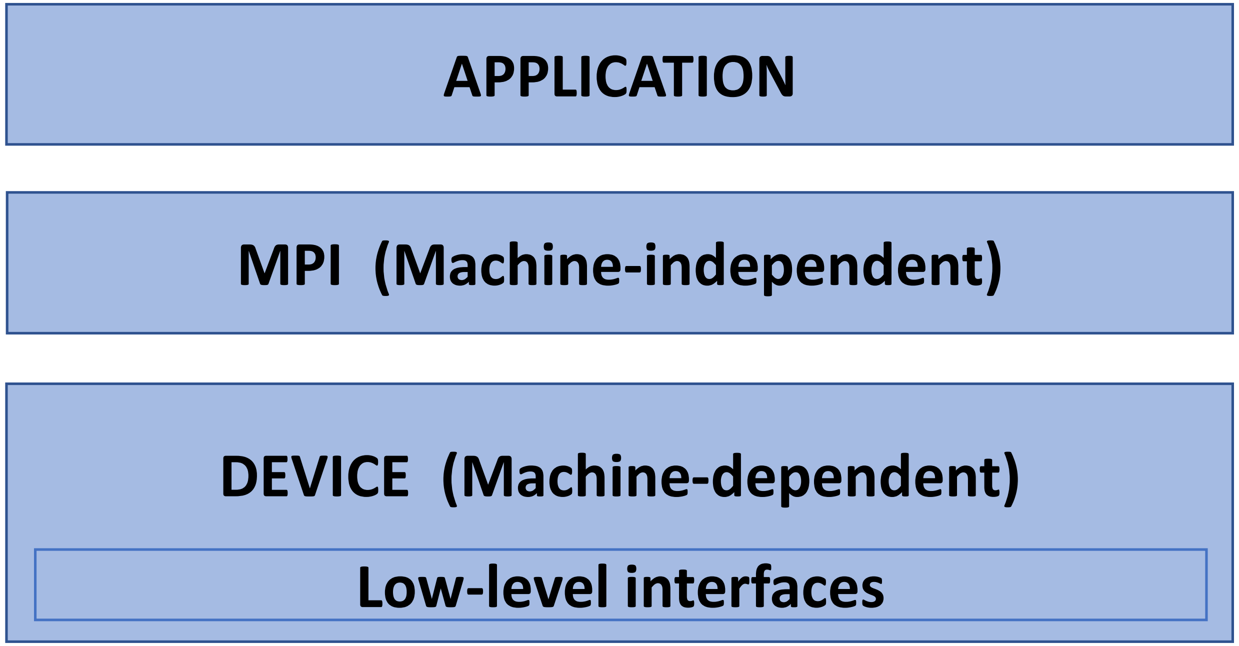
```
pmalakar 17952 17943  0 09:41 ?        00:00:00 /usr/lib/openssh/sftp-server
pmalakar 20853 16203  0 10:20 pts/1    00:00:00 mpiexec -np 8 -hosts 172.27.19.2 3 172.27.19.3 3 172.27
.19.4 3 ./IMB-MPI1 AllReduce
pmalakar 20854 20853  0 10:20 ?        00:00:00 /users/faculty/pmalakar/mpich-3.2.1-install/bin/hydra_p
mi_proxy --control-port 172.27.19.2:46385 --rmk user --launcher ssh --demux poll --pgid 0 --retries 10
--usize -2 --proxy-id 0
pmalakar 20855 20853  0 10:20 ?        00:00:00 /usr/bin/ssh -x 172.27.19.3 "/users/faculty/pmalakar/mp
ich-3.2.1-install/bin/hydra_pmi_proxy" --control-port 172.27.19.2:46385 --rmk user --launcher ssh --dem
ux poll --pgid 0 --retries 10 --usize -2 --proxy-id 1
pmalakar 20856 20853  0 10:20 ?        00:00:00 /usr/bin/ssh -x 172.27.19.4 "/users/faculty/pmalakar/mp
ich-3.2.1-install/bin/hydra_pmi_proxy" --control-port 172.27.19.2:46385 --rmk user --launcher ssh --dem
ux poll --pgid 0 --retries 10 --usize -2 --proxy-id 2
pmalakar 20857 20854 76 10:20 ?        00:00:03 ./IMB-MPI1 AllReduce
pmalakar 20858 20854 76 10:20 ?        00:00:03 ./IMB-MPI1 AllReduce
pmalakar 20859 20854 76 10:20 ?        00:00:03 ./IMB-MPI1 AllReduce
pmalakar 20861 17877  0 10:20 pts/4    00:00:00 ps -aef
```

Compute Nodes

```
pmalakar  8756  8728  0 10:18 pts/0    00:00:00 -bash
pmalakar  8759  8755  0 10:18 ?        00:00:00 /usr/lib/openssh/sftp-server
root      8781  1123  0 10:20 ?        00:00:00 sshd: pmalakar [priv]
pmalakar  8845  8781  0 10:20 ?        00:00:00 sshd: pmalakar@notty
pmalakar  8846  8845  0 10:20 ?        00:00:00 /users/faculty/pmalakar/mpich-3.2.1-install/bin/hydra_pmi_prox
y --control-port 172.27.19.2:46385 --rmk user --launcher ssh --demux poll --pgid 0 --retries 10 --usize -2 --p
roxy-id 1
pmalakar  8847  8846 99 10:20 ?        00:00:12 ./IMB-MPI1 AllReduce
pmalakar  8848  8846 99 10:20 ?        00:00:12 ./IMB-MPI1 AllReduce
pmalakar  8849  8846 99 10:20 ?        00:00:12 ./IMB-MPI1 AllReduce
```

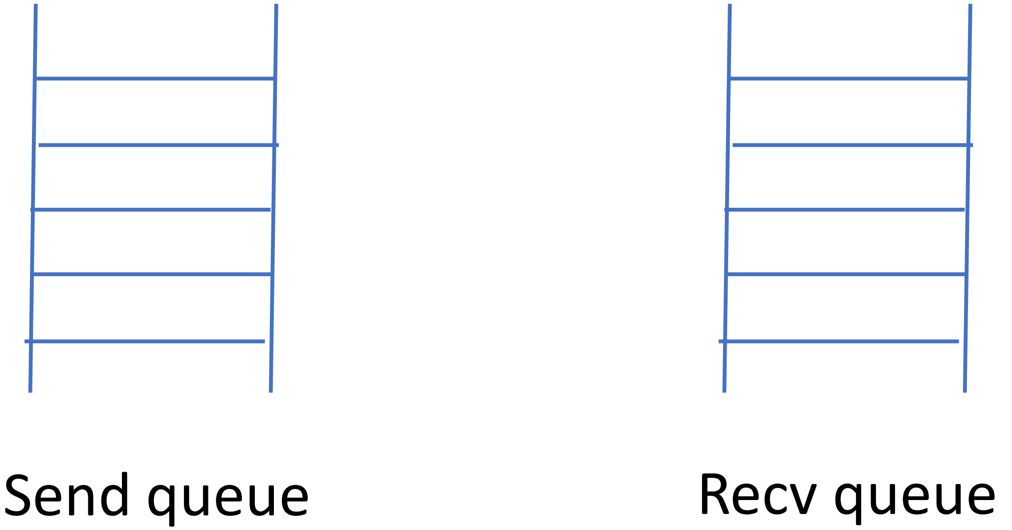
```
pmalakar  8838  8774  0 10:20 pts/1    00:00:00 -bash
pmalakar  8841  8837  0 10:20 ?        00:00:00 /usr/lib/openssh/sftp-server
root      8851  1250  0 10:20 ?        00:00:00 sshd: pmalakar [priv]
pmalakar  8915  8851  0 10:20 ?        00:00:00 sshd: pmalakar@notty
pmalakar  8916  8915  0 10:20 ?        00:00:00 /users/faculty/pmalakar/mpich-3.2.1-install/bin/hydra_p
mi_proxy --control-port 172.27.19.2:46385 --rmk user --launcher ssh --demux poll --pgid 0 --retries 10
--usize -2 --proxy-id 2
pmalakar  8917  8916 99 10:20 ?        00:00:14 ./IMB-MPI1 AllReduce
pmalakar  8918  8916 99 10:20 ?        00:00:14 ./IMB-MPI1 AllReduce
```

MPI Stack


APPLICATION
MPI (Machine-independent)
DEVICE (Machine-dependent)
Low-level interfaces

Communication Subsystem

- Communication using sockets (one option)
- MPI handles communications, progress etc.
- Communication channels determine performance
- Shared-memory queue for intranode messaging



Reference: Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem by Buntinas et al.

Revisions

```c
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
  int arr[20] = {0};
  int myrank, size;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  MPI_Comm_size( MPI_COMM_WORLD, &size );

  if (myrank != 1)
  {
    MPI_Send(arr, 20, MPI_INT, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)
  {
    int count, recvarr[size][20];
    for (int i=0; i<=size; i++)
    {
      if (i == myrank) continue;
      MPI_Recv(recvarr[i], 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_Finalize();
  return 0;
}
```

Correct code?

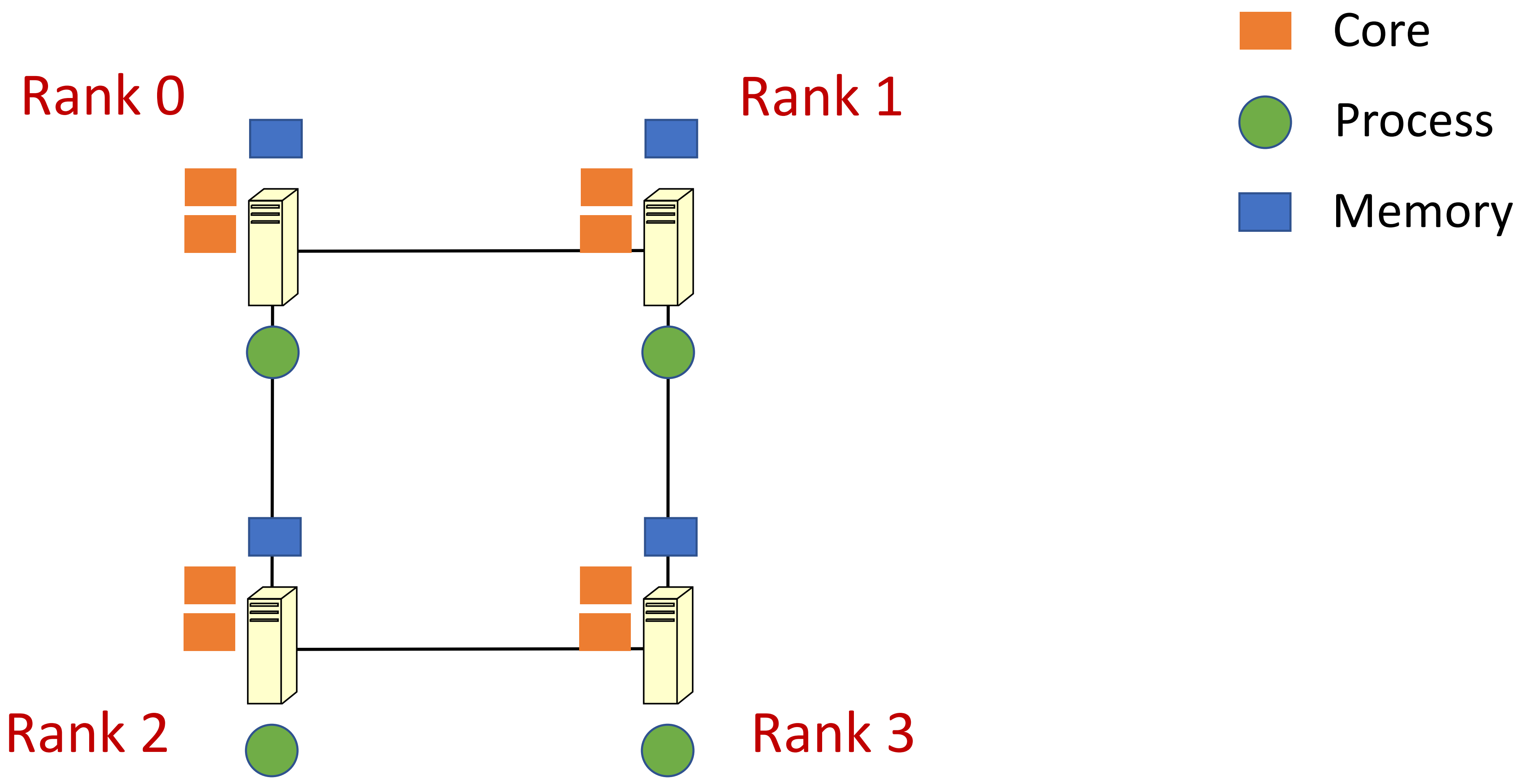
Parallel Programming is Hard!

- Programmer's responsibility to ensure correctness
  - Some processes may be waiting for data
  - Ensure that number of sends = number of receives
  - Avoid code that may lead to deadlocks

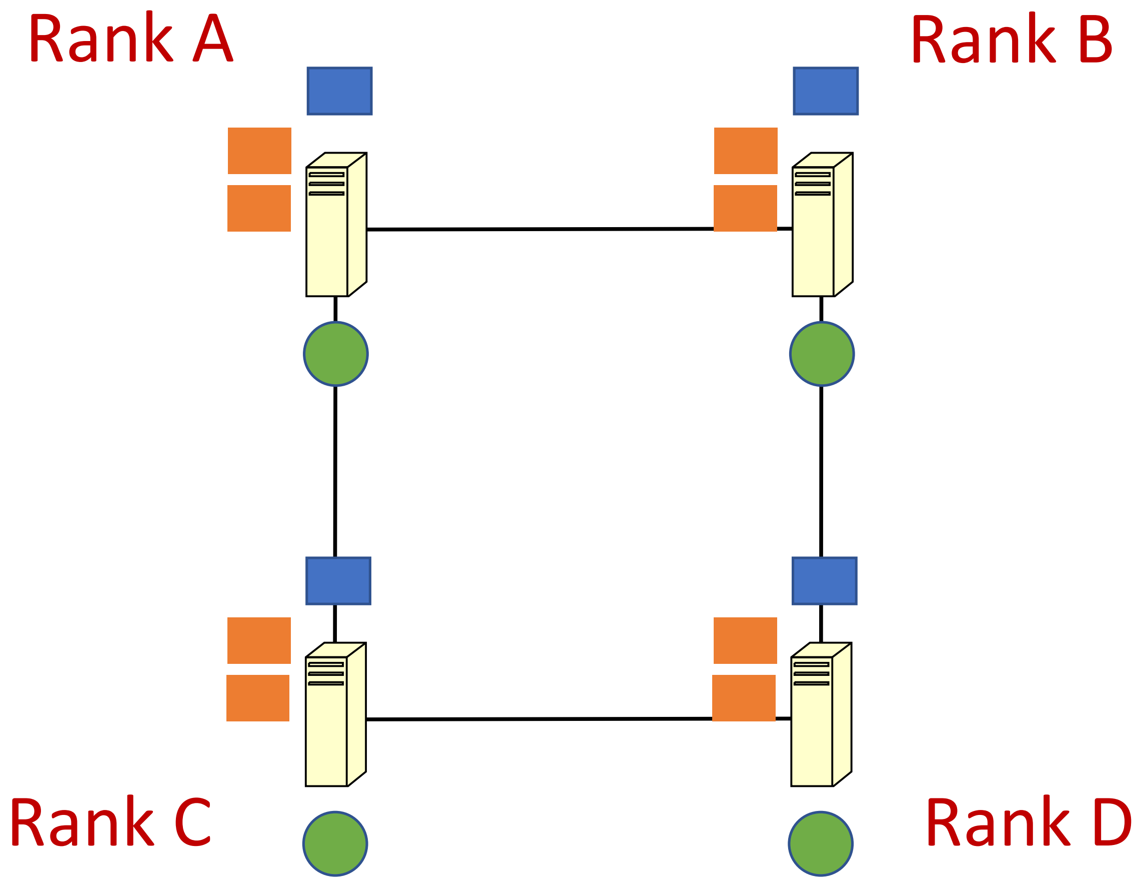
Eager vs. Rendezvous Protocol

- Eager
  - Send completes without acknowledgement from destination
  - MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE (check output of mpivars)
  - Small messages - typically 128 KB (at least in MPICH)
- Rendezvous
  - Requires an acknowledgement from a matching receive
  - Large messages

MPI Ranks

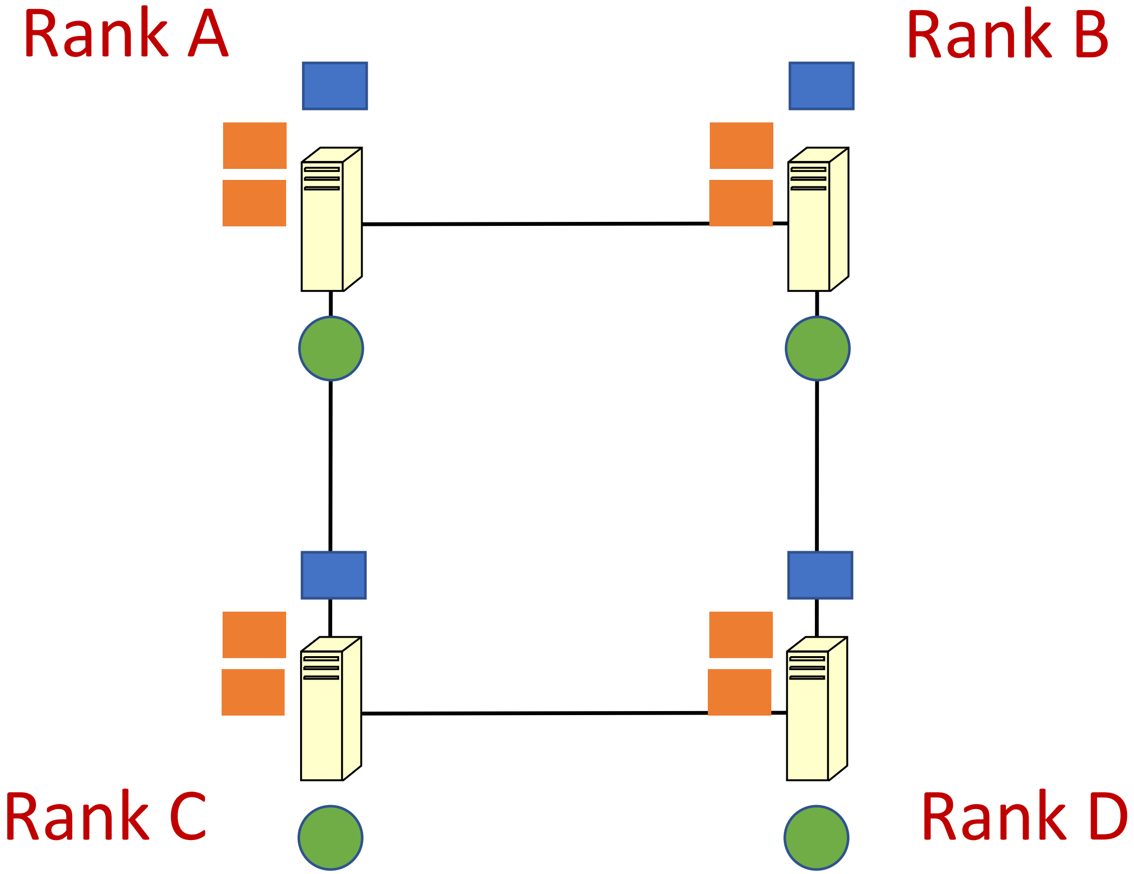


Process Placement



P2P communication between different pairs of processes may or may not take the same amount of time

Process Placement



Rank A – B

#Hops=1

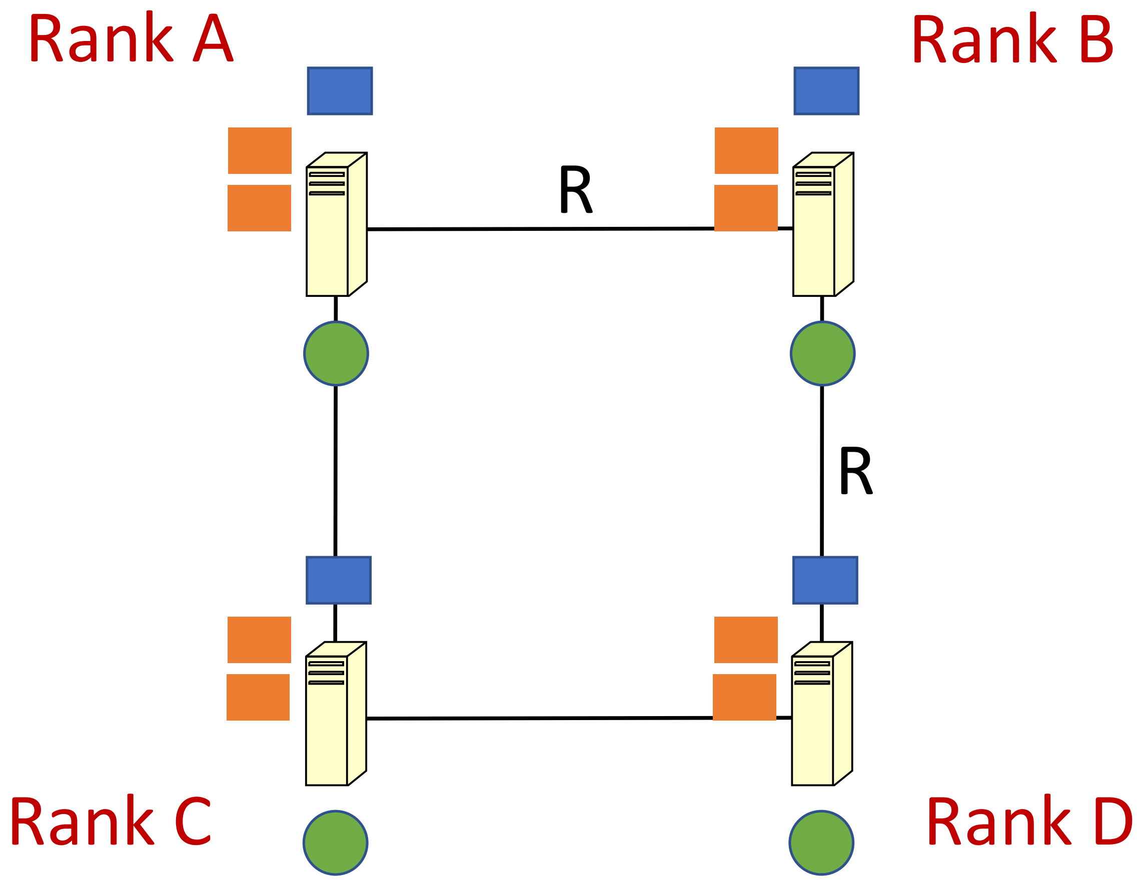
Rank A – D

#Hops=2

Rank A – C

#Hops=1

Communication Time



Rank A – B

$\#Hops=1$

$T_1 = D/R$

Rank A – D

$\#Hops=2$

$T_2 = D/R$

T_1 vs. T_2 ?

DEMO 1, 2, 3, 4

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, ..)

Today

- MPI_Barrier
- MPI_Bcast
- MPI_Gather
- MPI_Scatter
- MPI_Reduce

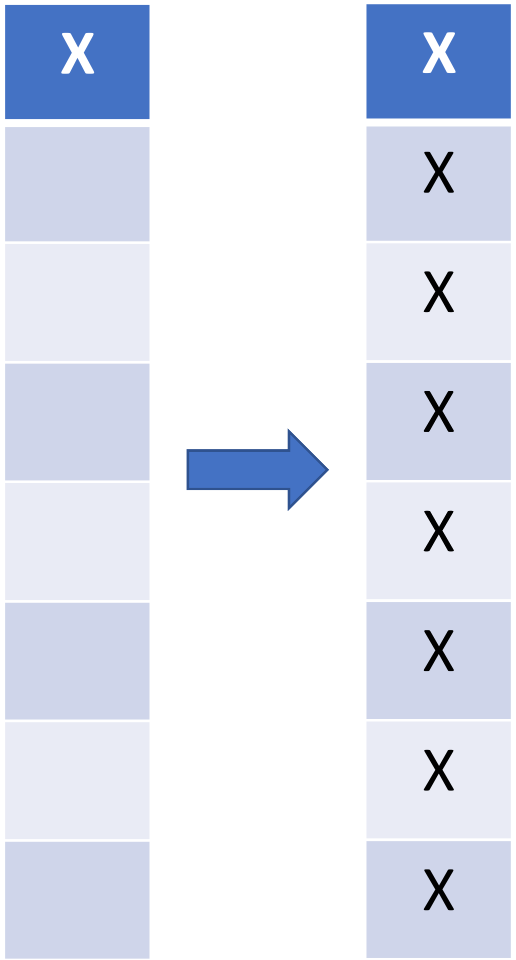
Barrier

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

```
Compute…
Communicate…
MPI_Barrier (MPI_COMM_WORLD);
```

Broadcast

- Root process sends message to all processes
- Any process can be root process but has to be the same across invocations from all processes
- int MPI_Bcast (buffer, count, datatype, root, comm)
- count – Number of elements in buffer
- buffer – Input at root only



How is broadcast implemented?

- P2P communications
- Naïve implementation?
- Binomial tree algorithm

Gather

- Gathers values from all processes to a root process
- int MPI_Gather (

```
const void *sendbuf,
int sendcount,
MPI_Datatype sendtype,
void *recvbuf,
int recvcount,
MPI_Datatype recvtype,
int root,
MPI_Comm comm
)
```

- Arguments recv* not relevant on non-root processes

Scatter

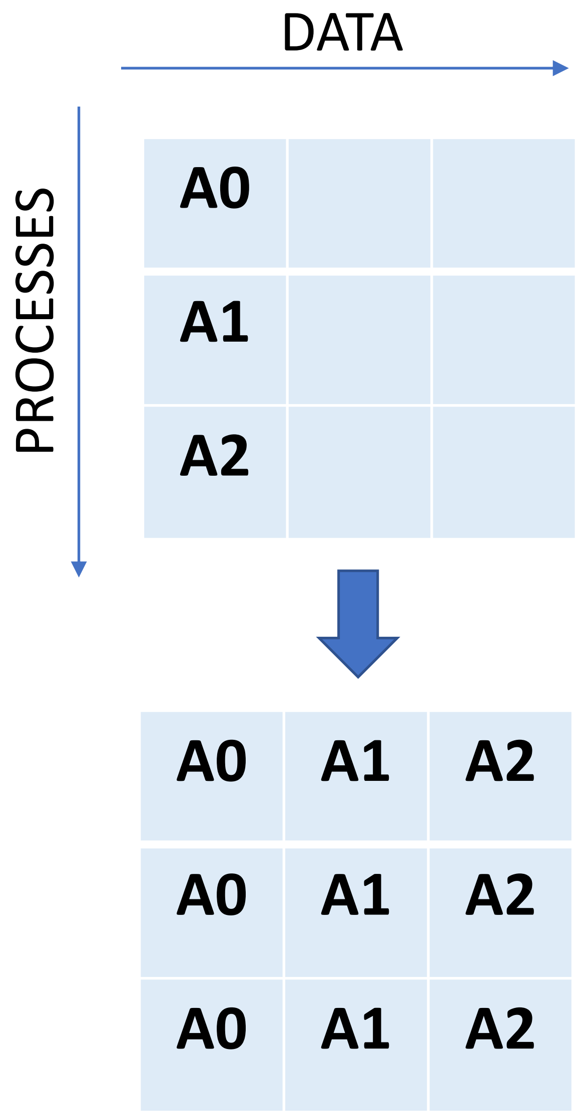
- Scatters values to all processes from a root process
- int MPI_Scatter (

```
const void *sendbuf,
int sendcount,
MPI_Datatype sendtype,
void *recvbuf,
int recvcount,
MPI_Datatype recvtype,
int root,
MPI_Comm comm
)
```

- Arguments send* not relevant on non-root processes
- Output parameter – recvbuf

Allgather

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

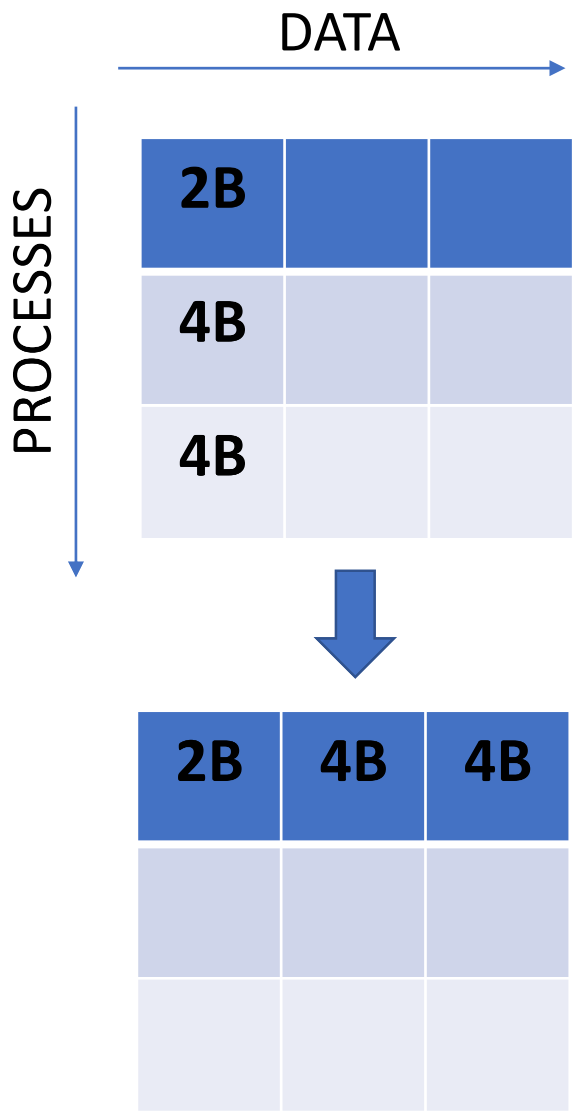


Gatherv

- 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

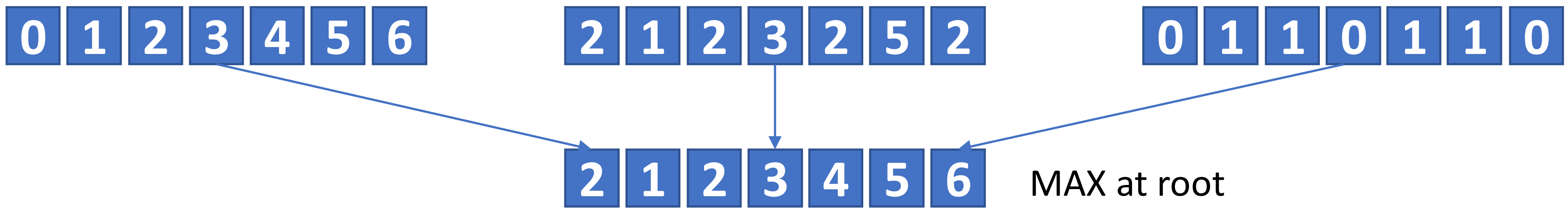
MPI_Recv (recvbuf+displs[i], recvcounts[i], recvtype, i, i, comm, &status) at root

MPI_Send at non-root



Reduce

- MPI_Reduce (inbuf, outbuf, count, datatype, op, root, comm)
- Combines element in inbuf of each process
- Combined value in outbuf of root
- op: MIN, MAX, SUM, PROD, …



Scalability

"The scalability of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processing elements." – Introduction to Parallel Computing

Strong scaling

- Fixed problem size
- Increase number of processes
- Efficiency decreases, in general

Weak scaling

- Fixed problem size per process
- Increase number of processes
- Increase problem size

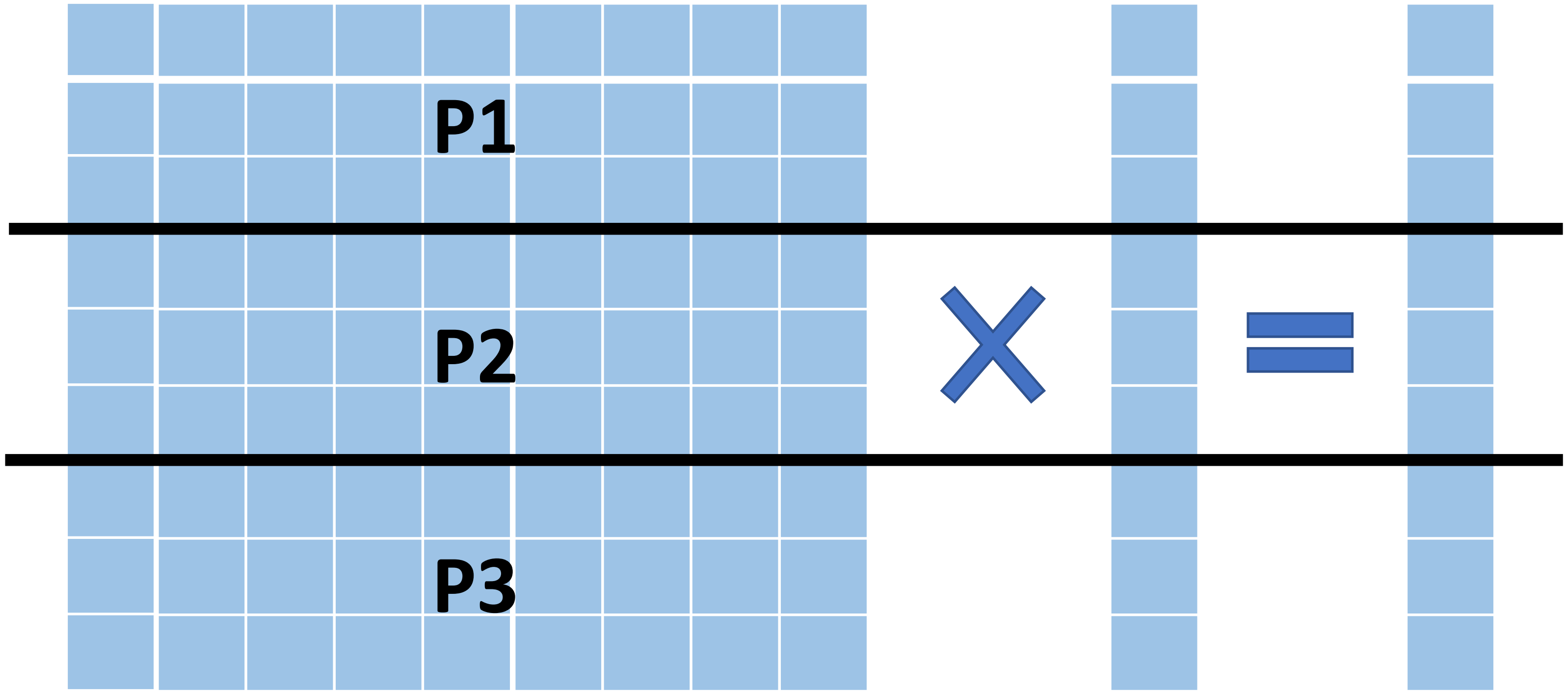
Demo

Strong and Weak Scaling using Reduce

Non-blocking Collectives

- MPI_Ibcast (buffer, count, datatype, root, comm, request)
- MPI_Igather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- MPI_Iscatter (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- MPI_Ireduce (sendbuf, recvbuf, count, datatype, op, root, comm, request)
- MPI_Igatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, request)

Parallelization – Matrix Vector Multiplication



P = 3

Decomposition

Assignment

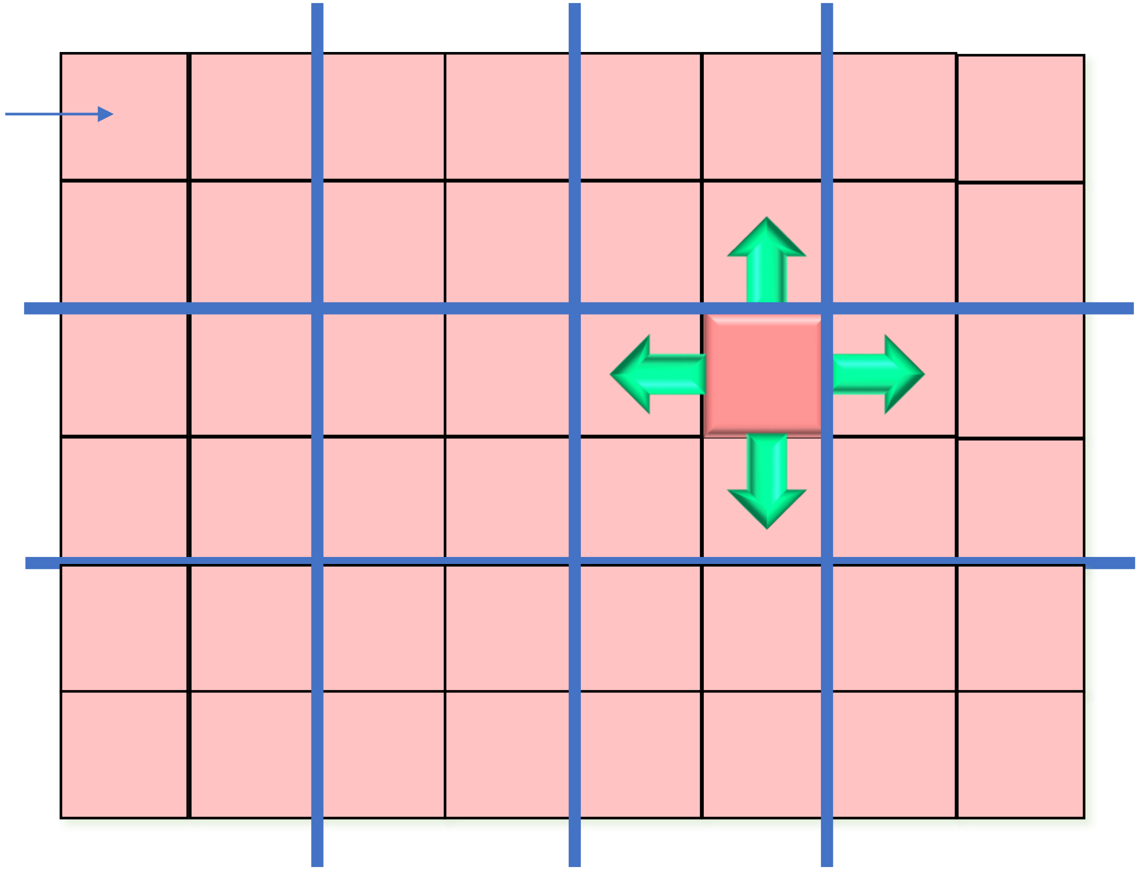
Communications

- Allgather
- Gather

Placement

Parallelization – Stencils

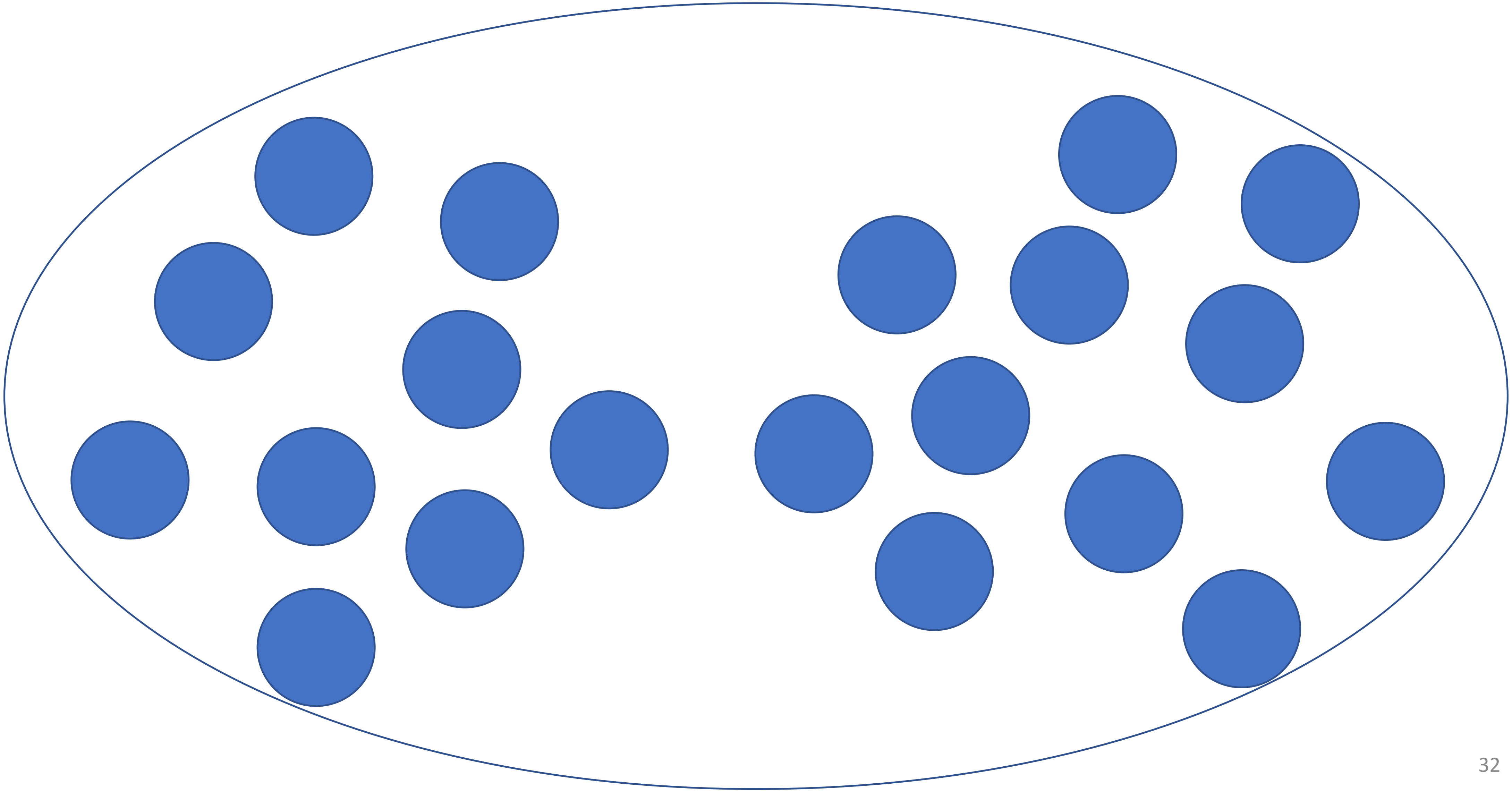
Grid point



Communications?

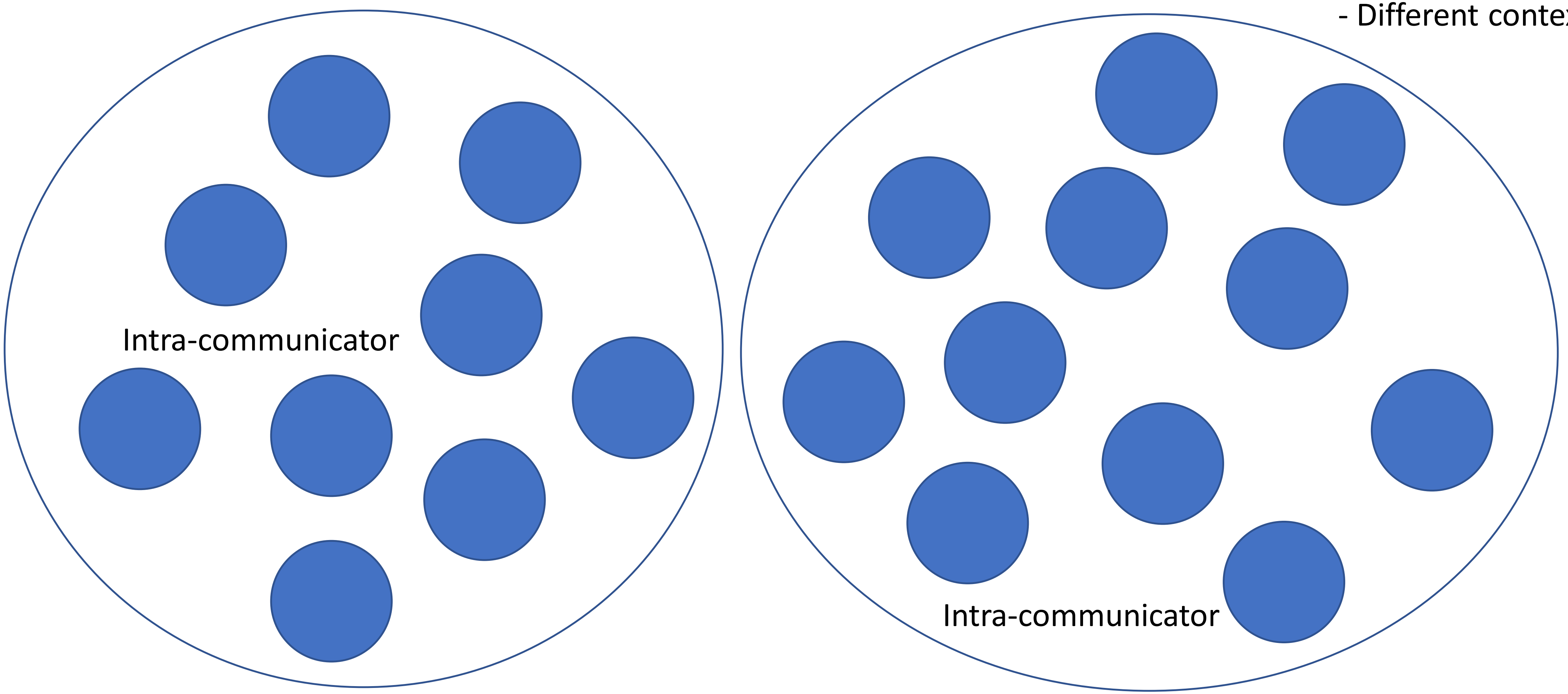
4 Isends()
4 Irecvs()

MPI_COMM_WORLD



Sub-communicator

- Logical subset
- Different contexts

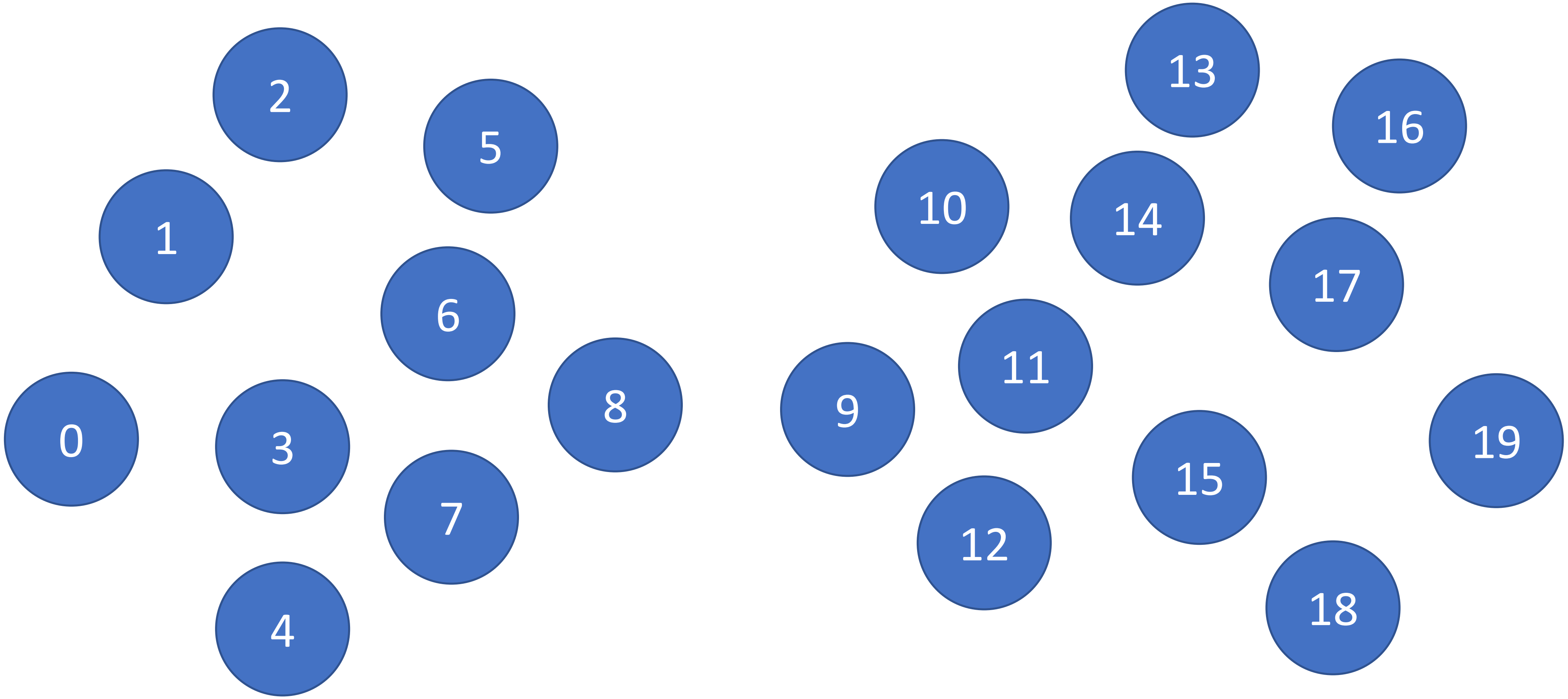


MPI_COMM_SPLIT

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

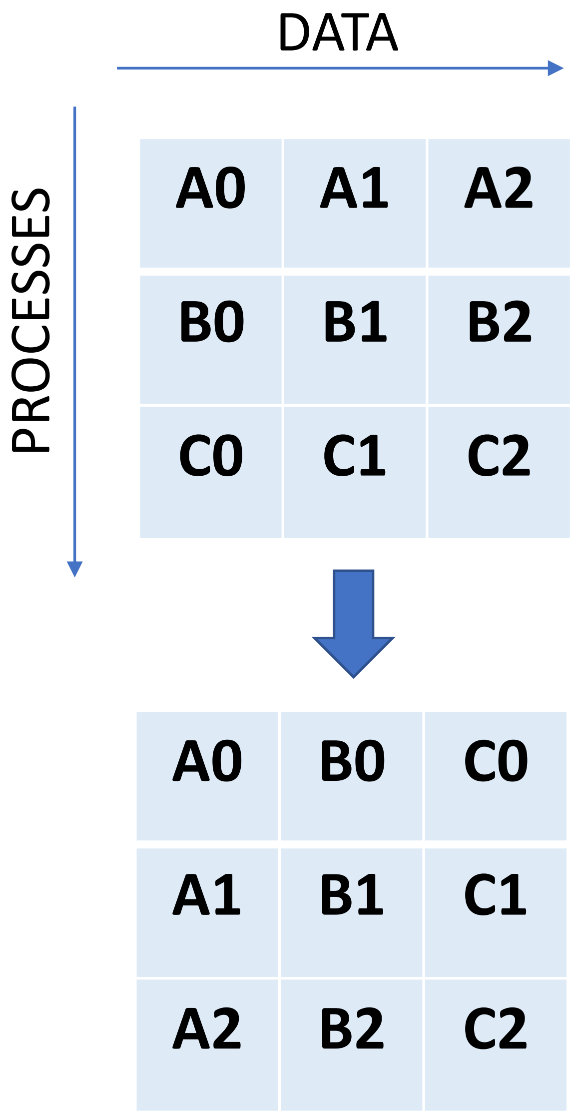
```
0:  0 of 3
1:  0 of 3
2:  0 of 3
3:  1 of 3
4:  1 of 3
5:  1 of 3
6:  2 of 3
7:  2 of 3
8:  2 of 3
```

Thank You

Backup

Alltoall

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

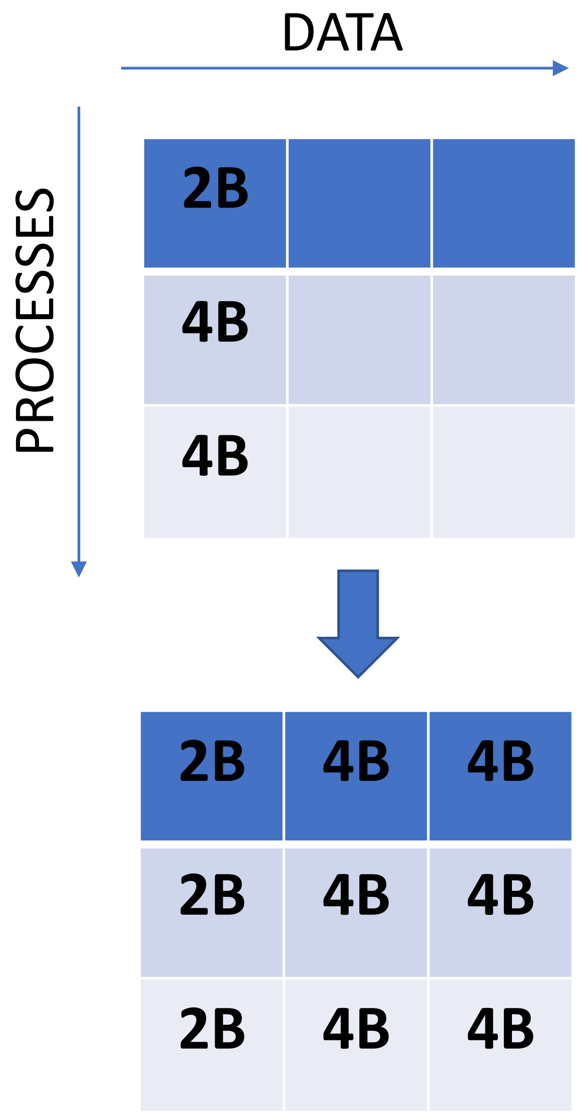


Equivalent collective?

- MPI_Scatter at all processes
- Cons?

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



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