

Introduction to Parallel Programming 2

Teeraparb Chantavat

Institute for Fundamental Study
Naresuan University

Flynn's Taxonomy

Concurrency VS Parallelism

Distributed Memory System

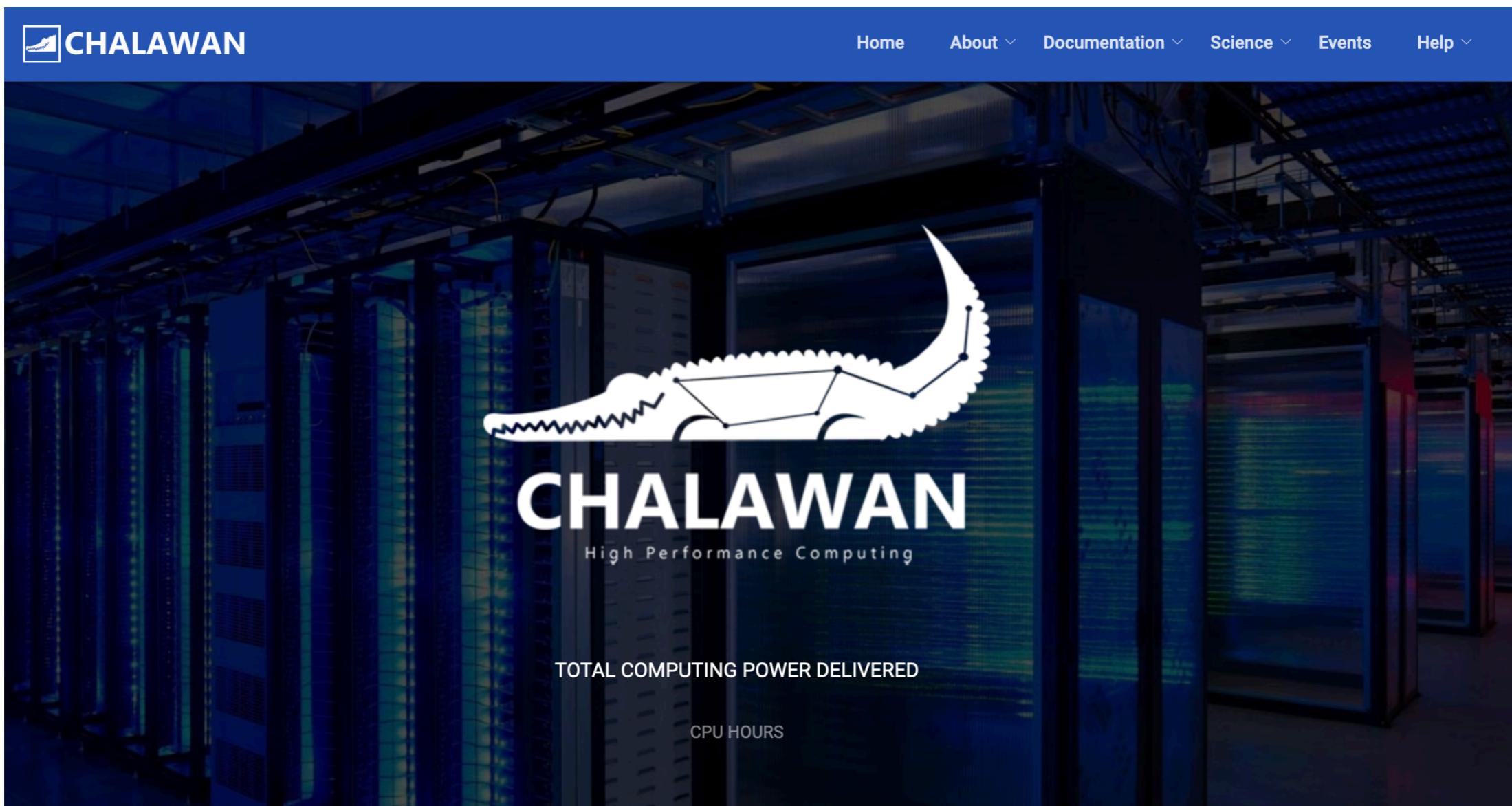
```
[ POLLUX:~] $ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                28
On-line CPU(s) list:  0-27
Thread(s) per core:   1
Core(s) per socket:   14
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
Stepping:               4
CPU MHz:               2600.000
BogoMIPS:              5200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:               1024K
L3 cache:               19712K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27
```

Parallel Programming Problems

- Deadlock
- Starvation
- Race Conditions

Chalawan Cluster

<http://chalawan.narit.or.th/>

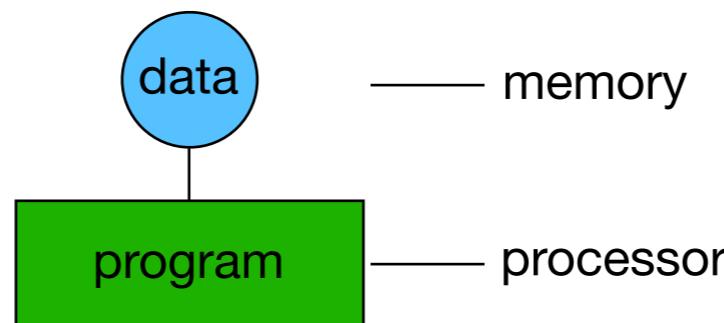


Using Slurm

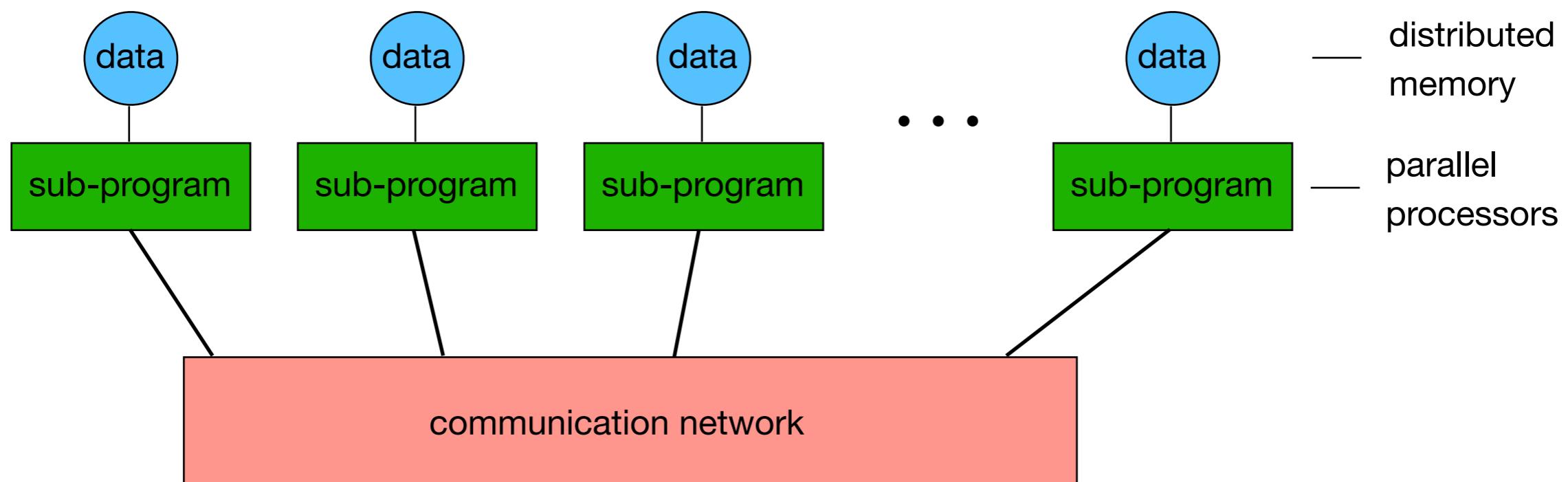
Threading

The Message-Passing Programming Paradigm

- Sequential Programming Paradigm

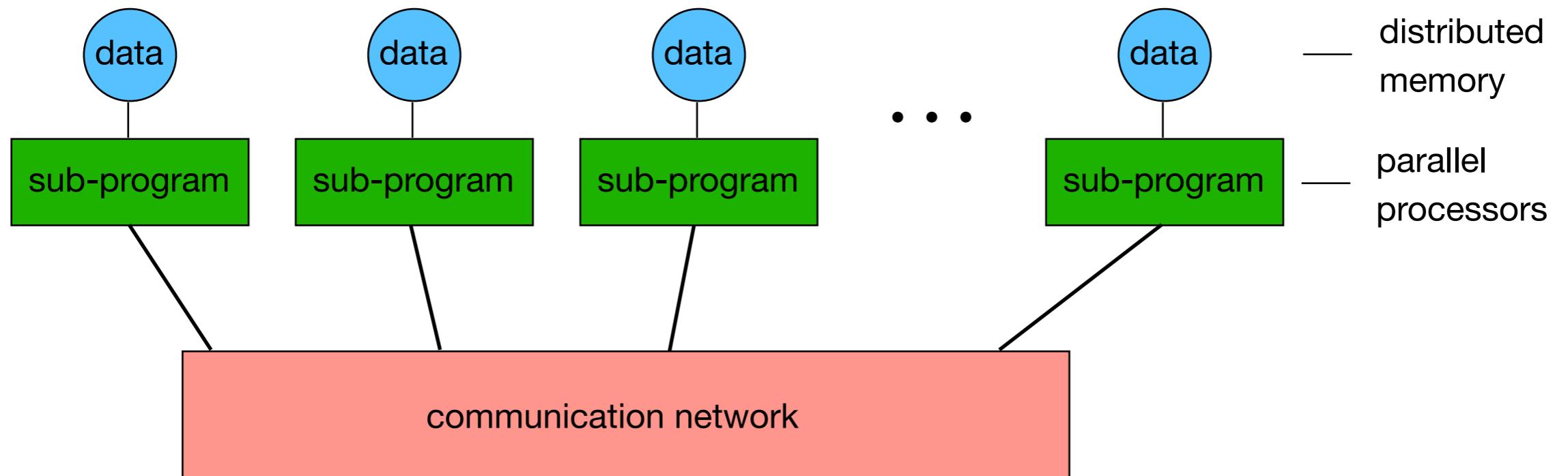


- Message-Passing Programming Paradigm



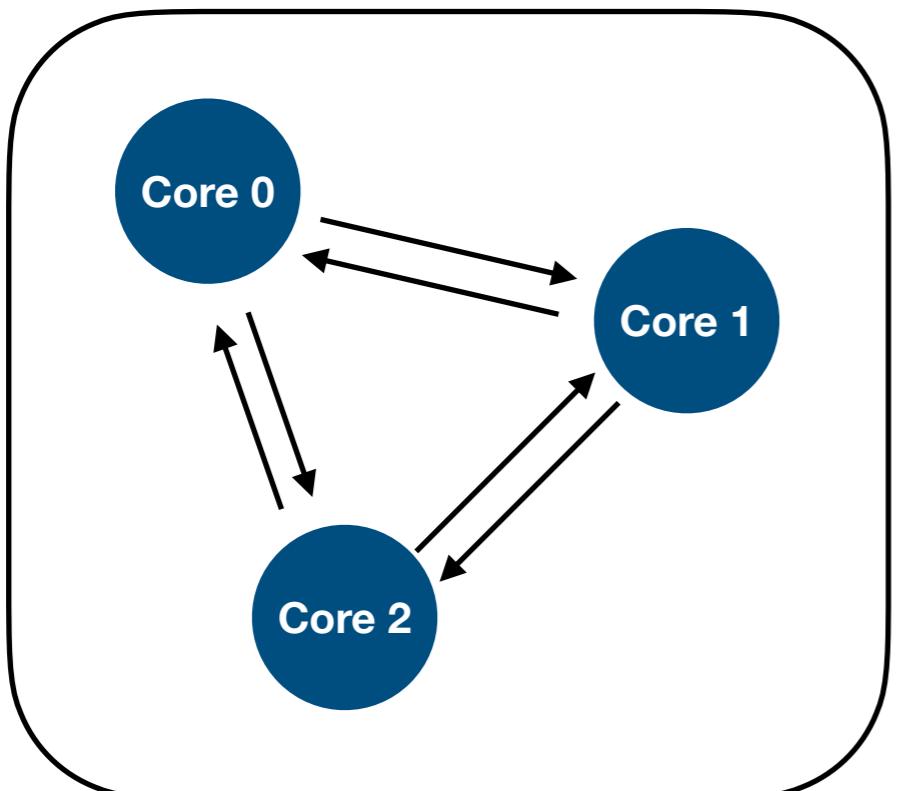
The Message-Passing Programming Paradigm

- Each processor in a message passing program
 - written in a conventional sequential language, e.g., C/C++, Fortran, or Python
 - typically the same on each processor (SPMD)
 - the variables of each sub-program have
 - communicate via
 - **the same name**
 - **but different locations (distributed memory) and different data!**
 - **i.e. all variables are private**
 - communicate via special send & receive routines (**message passing**)

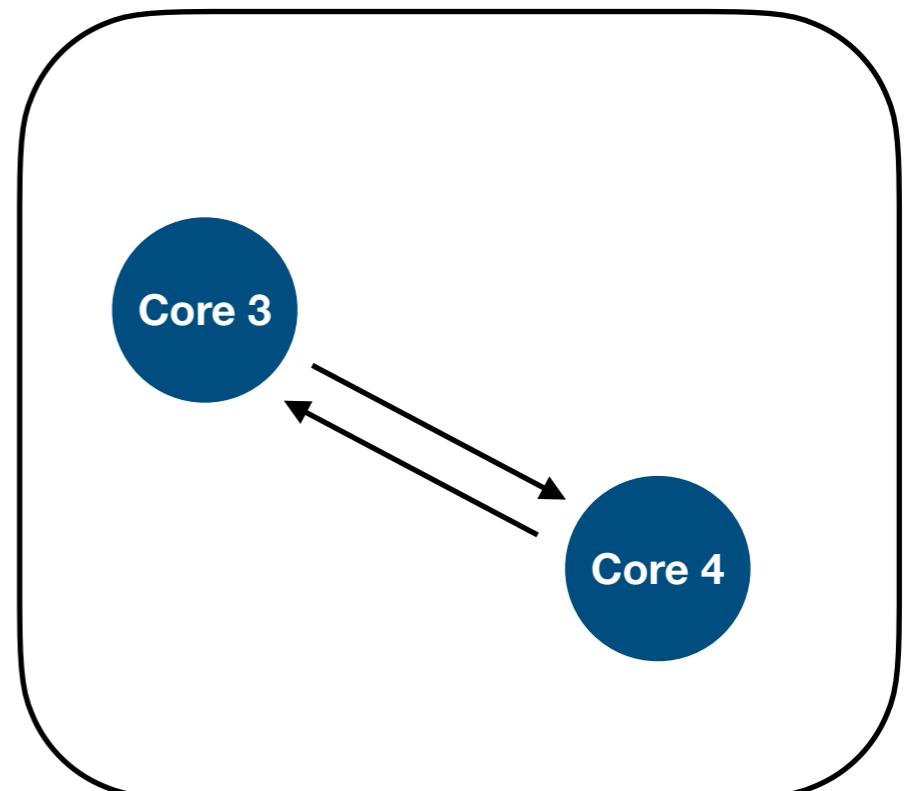


Communicators

A **communicator** is a group of cores that can communicate to one another. Each cores is independent and can run its own process.



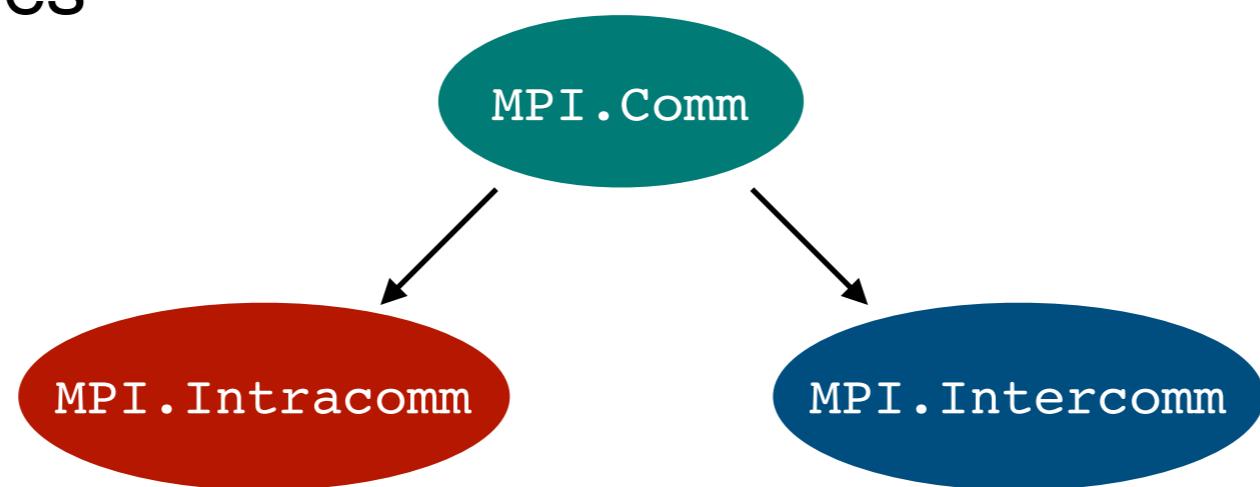
Communicator A



Communicator B

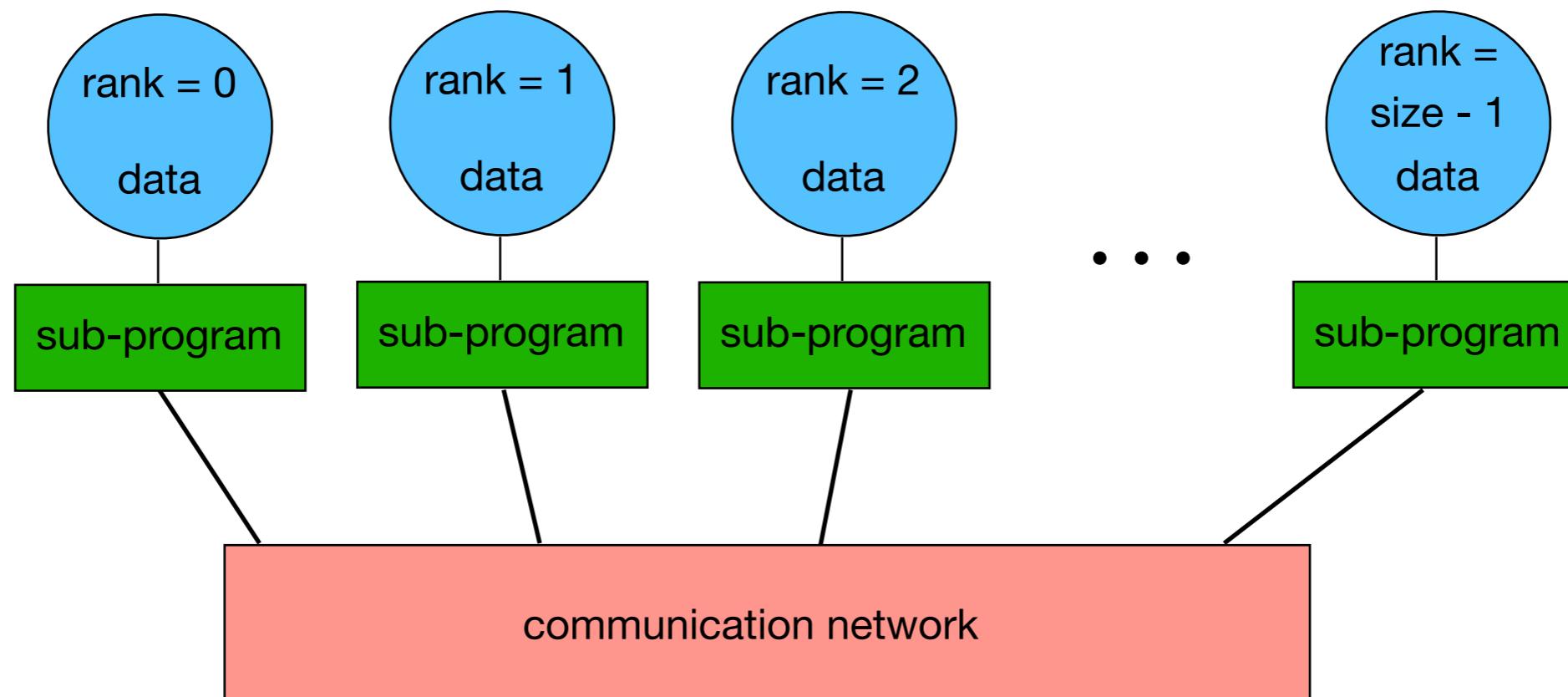
Communicators

- In *MPI for Python*, `MPI.Comm` is the base class of communicators.
- `MPI.Intracomm` is a sub-class of `MPI.Comm` mainly used for local nodes. `MPI.COMM_WORLD` and `MPI.COMM_SELF` are pre-defined instances of `MPI.Intracomm`.
- `MPI.Intercomm` is also a sub-class of `MPI.Comm` mainly used for remote nodes



Data and Work Distribution

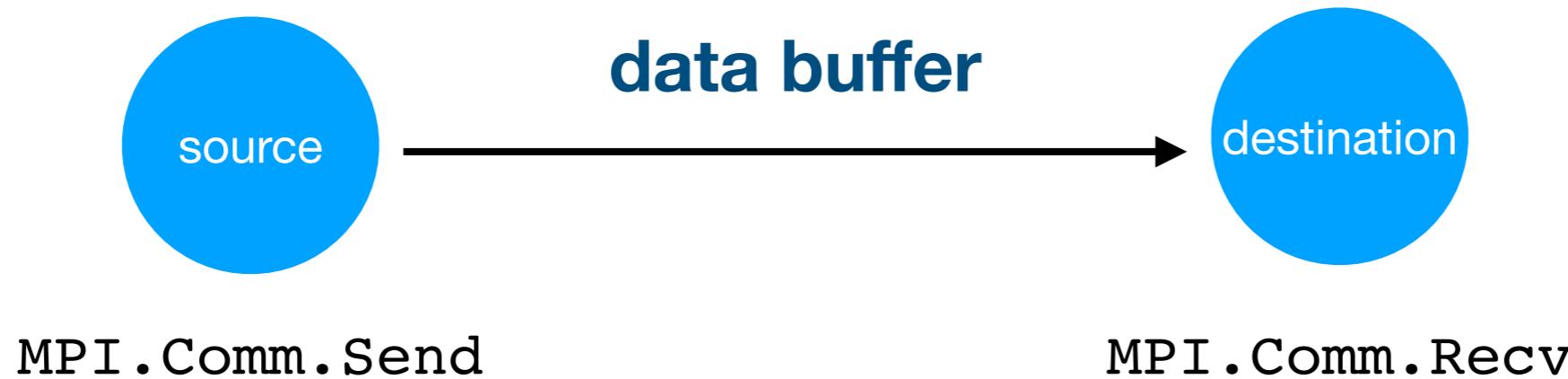
- The value of **rank** is returned by special library routine.
- The system of **size** processes is started by special MPI initialization.
- All distribution decision are based on rank
- i.e., which process work on which data



See MPI/ex00_mpi_greeting.py

Point-to-point Communications

- Simplest form of message passing.
- One process (core) sends a message to another. Another process (core) will receive the message



```
MPI.Comm.Send(buf, dest, tag=0)
```

```
MPI.Comm.Recv(buf, source=ANY_SOURCE, tag=ANY_TAG,  
status=None)
```

See MPI/ex01_mpi_send_recv0.py

MPI DataType

- When sending an array of data, the type of MPI needs to know the type of data.
- Sometime the data type has to be specified in the data buffer.

<code>MPI.CHAR</code>	<code>MPI.LONG</code>
<code>MPI.BYTE</code>	<code>MPI.FLOAT</code>
<code>MPI.SHORT</code>	<code>MPI.DOUBLE</code>
<code>MPI.INT</code>	<code>MPI.COMPLEX</code>

See `MPI/ex02_mpi_send_recv1.py`

MPI for Python object

- On top of Standard MPI commands, `mpi4py` has commands for generic Python objects using all lowercases.
- Python objects are serialized / deserialized using `pickle` module

```
MPI.Comm.send(object, dest, tag=0)
```

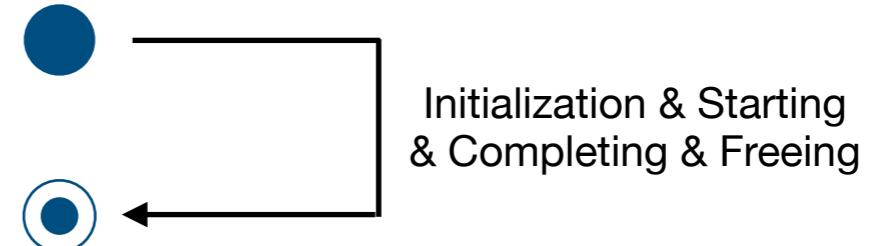
```
MPI.Comm.recv(object, source=ANY_SOURCE, tag=ANY_TAG,  
              status=None)
```

See MPI/ex03_mpi_send_recv2.py

MPI Operations

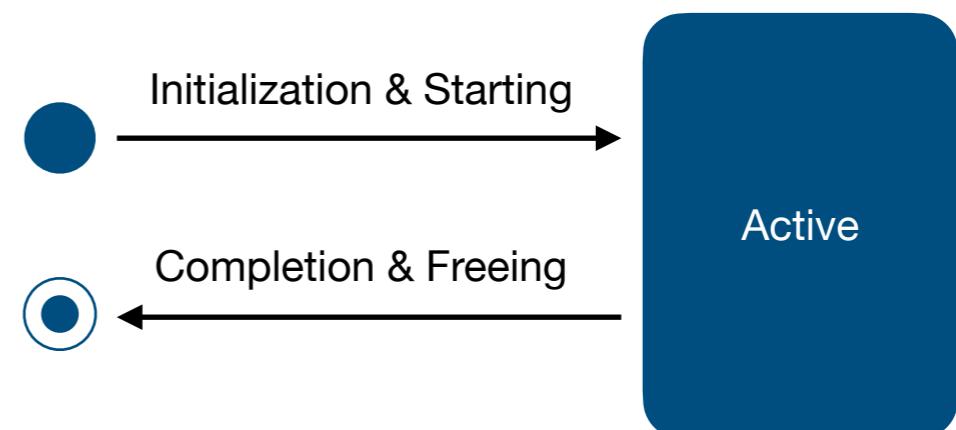
- **Blocking**

Initialization & Starting & Completion & Freeing



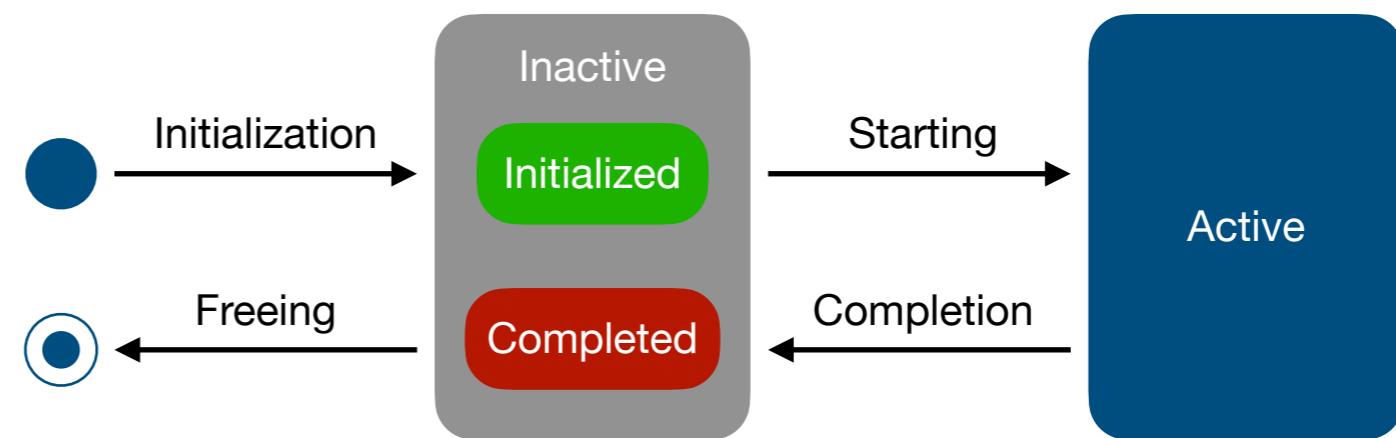
- **Non-blocking**

Initialization & Starting
Completion & Freeing



- **Persistent**

Initialization
Starting
Completion
Freeing



See MPI/ex04_send_recv3.py

MPI Send Communication Modes

`MPI_Send` has 4 modes of communication while `MPI_Recv` has only 1 mode.

Both can be blocking or non-blocking.

- **Standard**

The MPI library decide whether or not the non-local buffered the outgoing data.

- **Buffered (Asynchronous)**

The MPI library decide to use buffer for outgoing data if no matching receiver has been posted.

- **Synchronous**

The outgoing data buffer can be reused once the receiving process starting receiving the data.

- **Ready**

The outgoing data buffer can be reused once the receiving process has been posted.

MPI Send Variants

Communication Mode	Blocking	Non-blocking	Persistent
Standard	<code>MPI.Comm.Send</code> <code>MPI.Comm.send</code>	<code>MPI.Comm.Isend</code> <code>MPI.Comm.isend</code>	<code>MPI.Comm.Send_init</code>
Buffered (Asynchronous)	<code>MPI.Comm.Bsend</code> <code>MPI.Comm.bsend</code>	<code>MPI.Comm.Ibsend</code> <code>MPI.Comm.ibsend</code>	<code>MPI.Comm.Bsend_init</code>
Synchronous	<code>MPI.Comm.Ssend</code> <code>MPI.Comm.ssend</code>	<code>MPI.Comm.Isend</code> <code>MPI.Comm.isend</code>	<code>MPI.Comm.Ssend_init</code>
Ready	<code>MPI.Comm.Rsend</code>	<code>MPI.Comm.Irsend</code>	<code>MPI.Comm.Rsend_init</code>

for generic Python objects

MPI Recv Variants

Communication Mode	Blocking	Non-blocking	Persistent
Standard	<code>MPI.Comm.Recv</code> <code>MPI.Comm.recv</code>	<code>MPI.Comm.Irecv</code> <code>MPI.Comm.irecv</code>	<code>MPI.Comm.Recv_Init</code>

for generic Python objects

MPI Sendrecv Variants

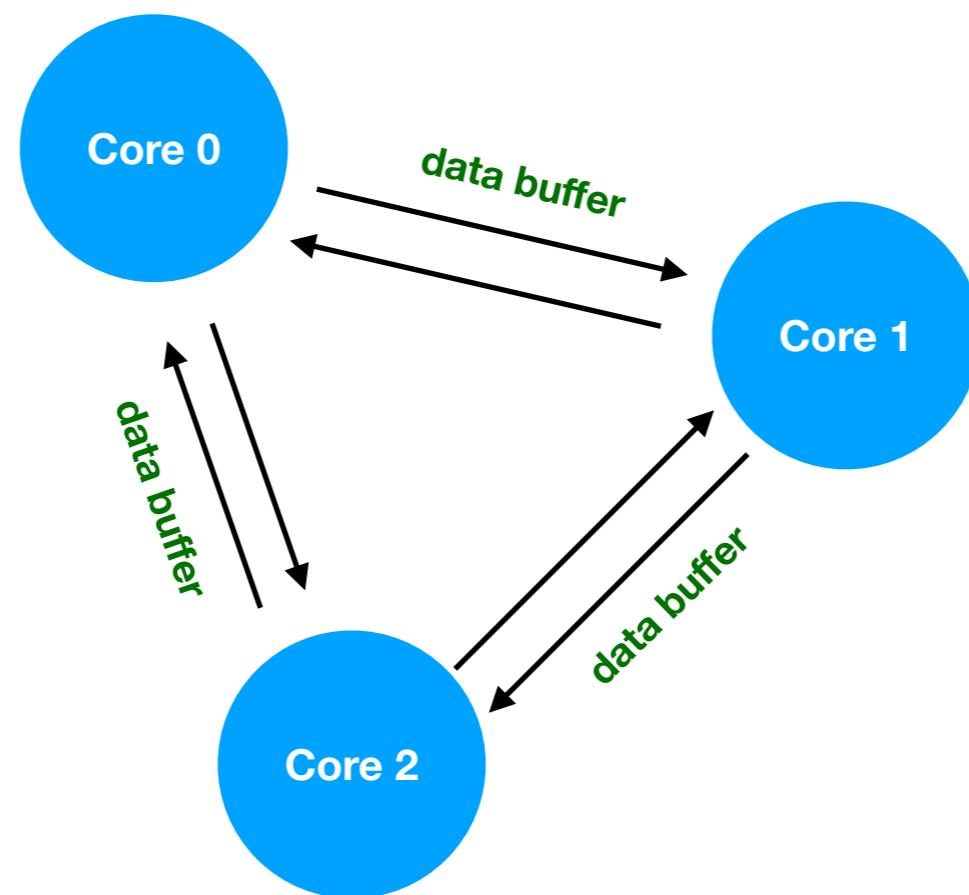
- The **send-receive** operation combine in one operation the sending of a message to one destination and the receiving of another message, from another process

Communication Mode	Blocking
Standard	<code>MPI.Comm.Sendrecv</code> <code>MPI.Comm.Sendrecv_replace</code> <code>MPI.Comm.sendrecv</code>

See MPI/ex05_mpi_send_recv4.py

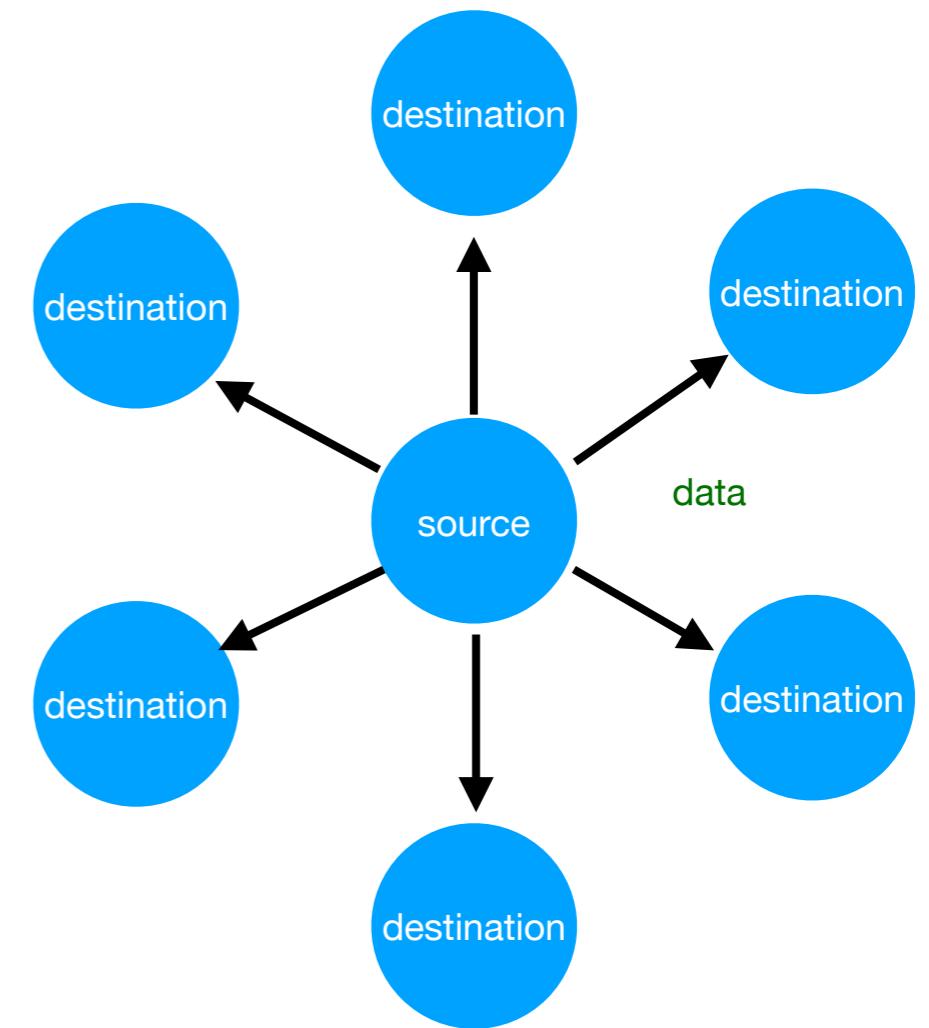
Collective Communication

- Collective communication routines are higher level routines.
- Several processes are involved at a time



Broadcast

- A one-to-many communication.
- All core receive the same copy of the data
- Not very efficient if we have only one communication channel.



```
MPI.Comm.Bcast(buf, root=0)
```

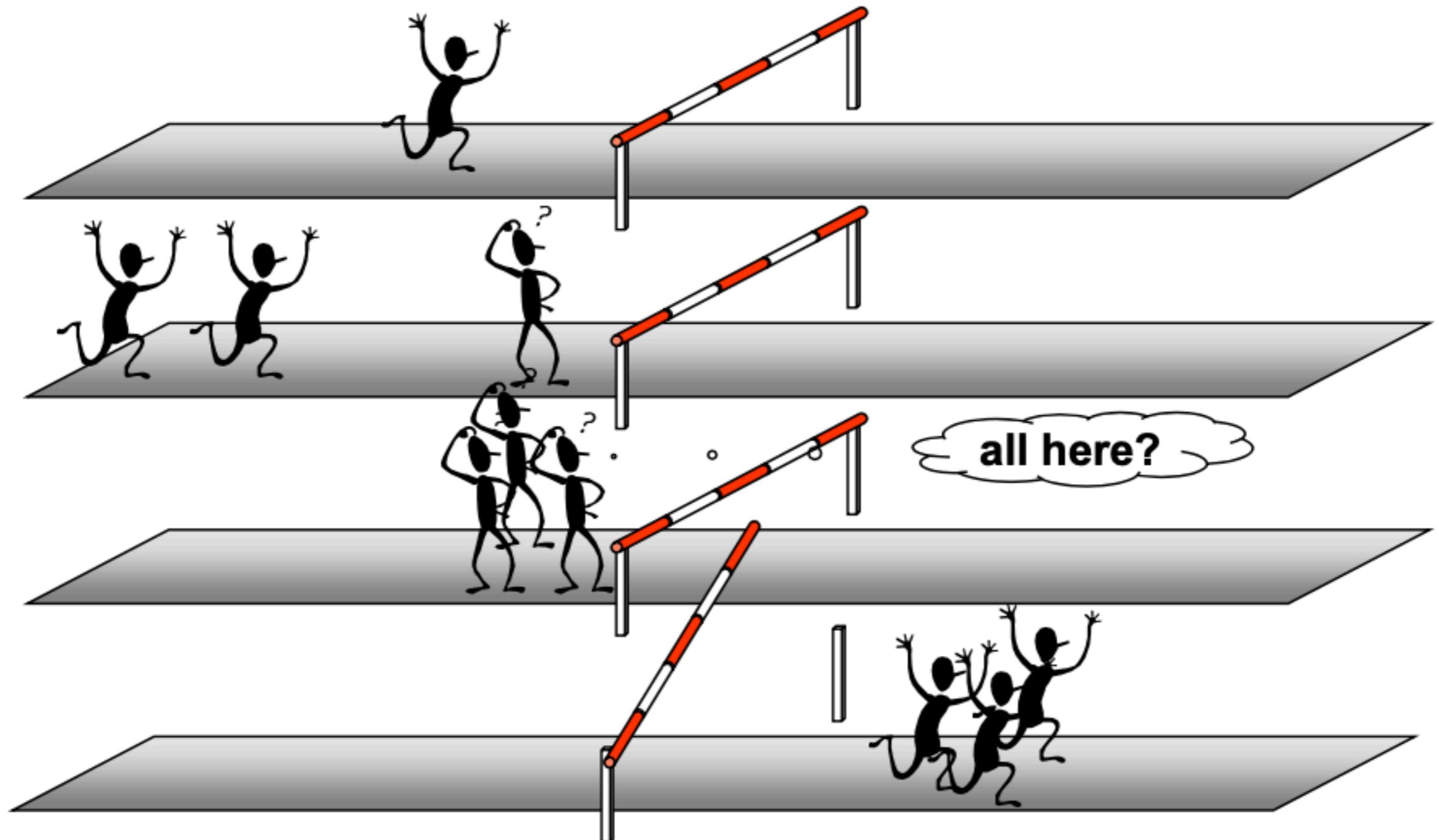
MPI Bcast Variants

Communication Mode	Blocking	Non-blocking
Standard	<code>MPI.Comm.Bcast</code> <code>MPI.Comm.bcast</code>	<code>MPI.Comm.Ibcast</code>
<code>MPI.Comm.Ibcast(buf, root=0)</code>		
<code>MPI.Comm.bcast(object, root=0)</code>		

See MPI/ex06_mpi_bcast.py

Barrier

- Synchronize processes

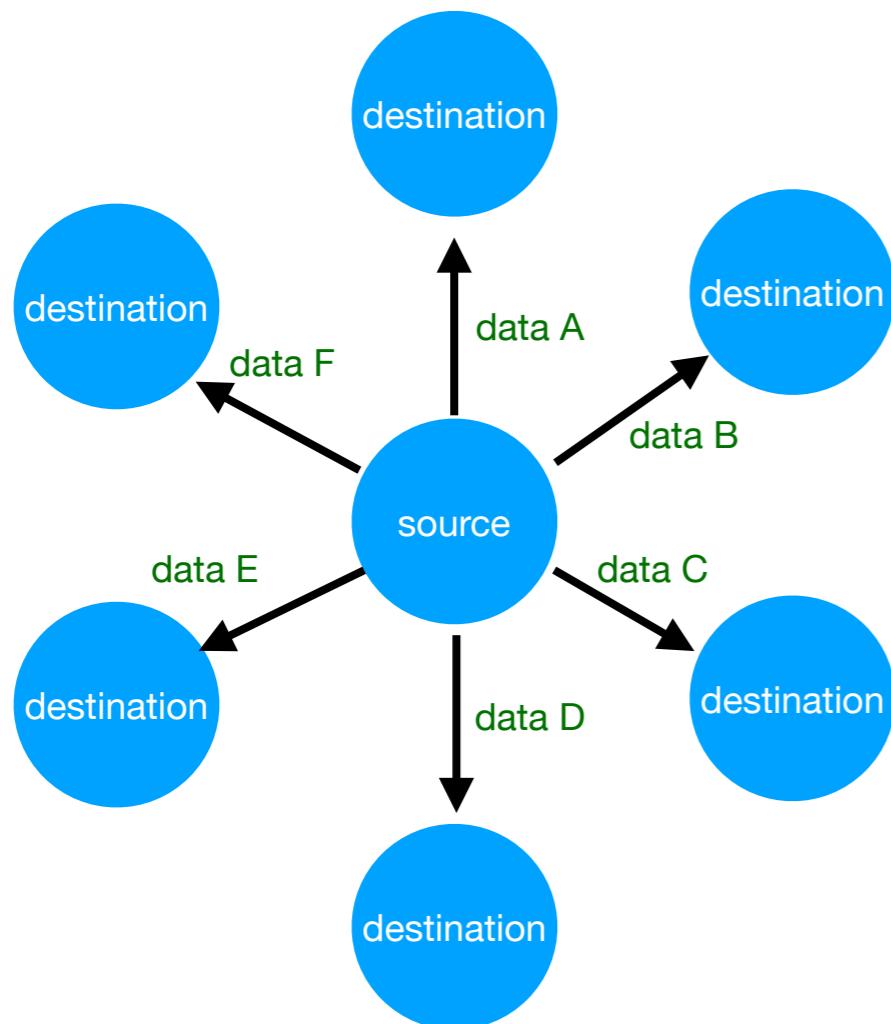


MPI Barrier Variants

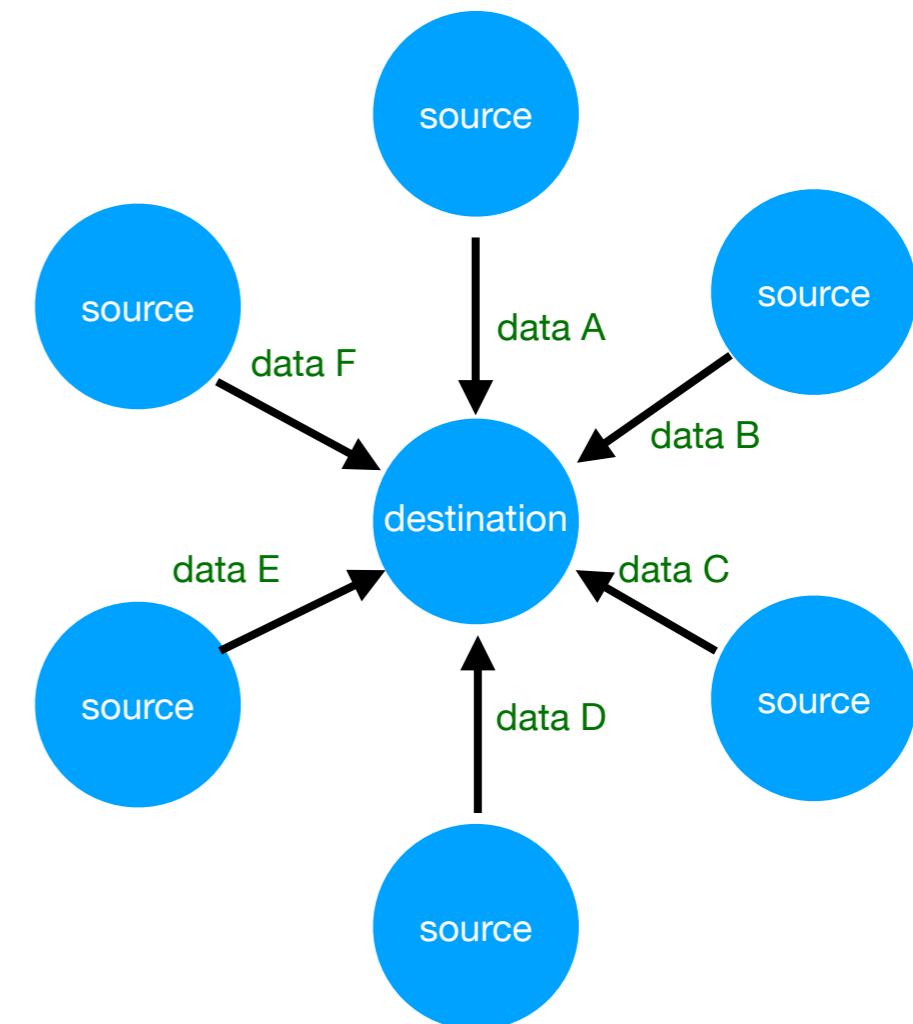
Communication Mode	Blocking	Non-blocking
Standard	<code>MPI.Comm.Barrier</code> <code>MPI_Comm.barrier</code>	<code>MPI.Comm.Ibarrier</code>
	<code>MPI.Comm.Barrier()</code>	<code>MPI.Comm.Ibarrier()</code>
	<code>MPI.Comm.barrier()</code>	

Scatter and Gather

- Split the data equally to all other processes.



Scatter

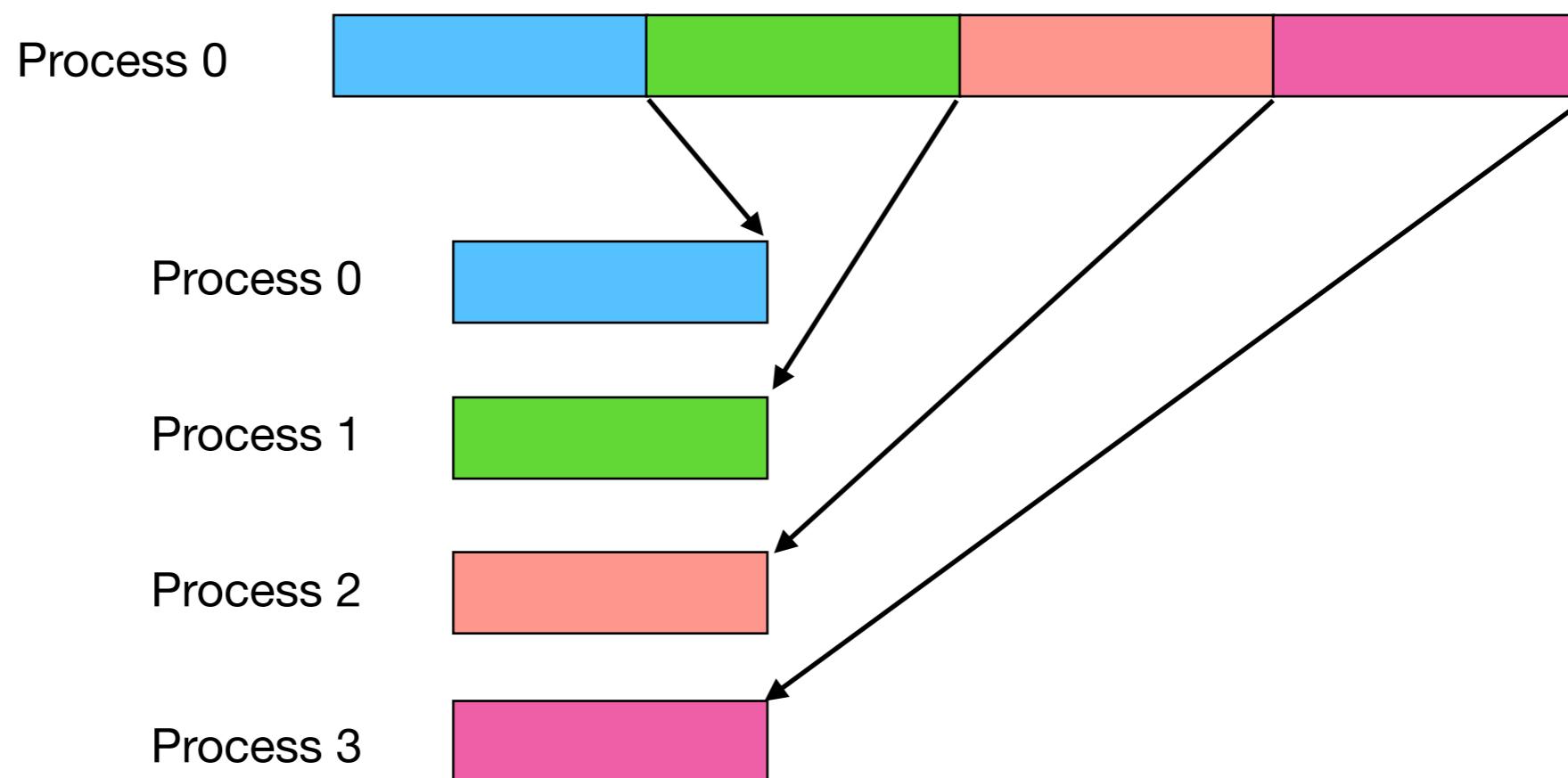


Gather

See MPI/ex07_mpi_scatter_gather.py

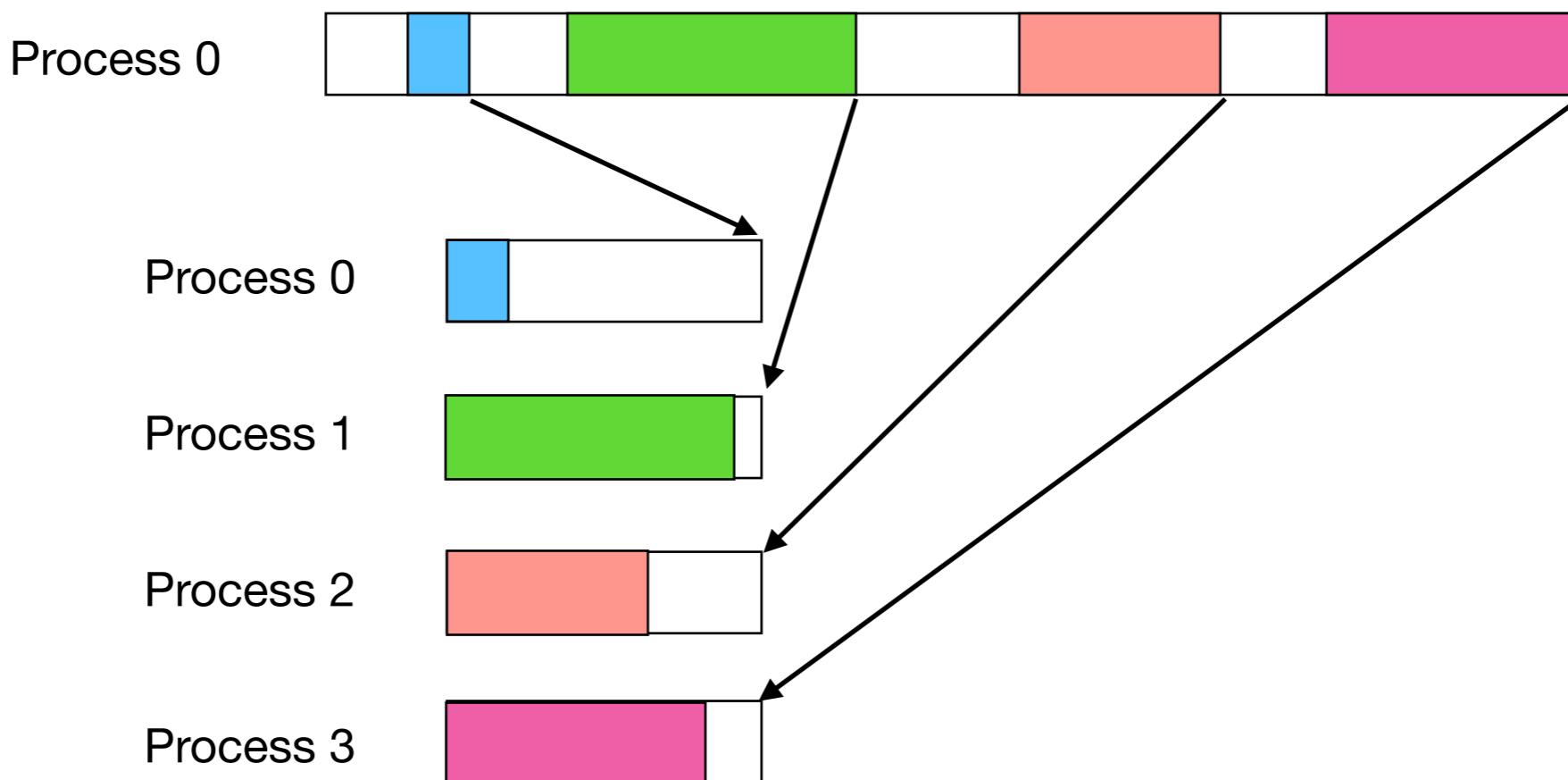
Scatter and Gather

- Scatter (Gather) requires contiguous data and uniform data size



Scatter and Gather (Vector data)

- Scatterv (Gatherv) allows gaps between messages in source data
- Irregular message size are allowed.
- Data can be distributed to processes in any order



See MPI/ex08_mpi_scatterv_gatherv.py

MPI Scatter / Gather Variants

Communication Mode

Blocking

Non-blocking

Standard

`MPI_Comm.Scatter`
`MPI_Comm.Scatterv`
`MPI_Comm.scatter`

`MPI_Comm.Iscatter`
`MPI_Comm.Iscatterv`

Communication Mode

Blocking

Non-blocking

Standard

`MPI_Comm.Gather`
`MPI_Comm.Gatherv`
`MPI_Comm.gather`

`MPI_Comm.Igather`
`MPI_Comm.Igahterv`

Allgather and Allgatherv

- Gathers data from all members of a group and sends the data to all members of the group.
- The Allgather (Allgatherv) function is similar to the MPI_Gather function, except that it sends the data to all processes instead of only to the root.

See MPI/ex09_allgather.py

MPI Allgather Variants

Communication Mode	Blocking	Non-blocking
Standard	<code>MPI.Comm.Allgather</code> <code>MPI.Comm.Allgatherv</code> <code>MPI.Comm.allgather</code>	<code>MPI.Comm.Iallgather</code> <code>MPI.Comm.Iallgatherv</code>
<code>MPI.Comm.Allgather(sendbuff, recvbuff)</code>		
<code>MPI.Comm.Allgatherv(sendbuff, recvbuff)</code>		
<code>MPI.Comm.allgather(sendobj)</code>		

All to all

- Gathers data from and scatters data to all members of a group.
- The **Alltoall** is an extension **Allgather** function.
- Each process sends distinct data to each of the receivers. The j th block that is sent from process i is received by process j and is placed in the i th block of the receive buffer.

All to all Variants

Communication Mode	Blocking	Non-blocking
Standard	<code>MPI.Comm.Alltoall</code> <code>MPI.Comm.Alltoallv</code> <code>MPI.Comm.Alltoallw</code> <code>MPI.Comm.alltoall</code>	<code>MPI.Comm.Ialltoall</code> <code>MPI.Comm.Ialltoallv</code> <code>MPI.Comm.Ialltoallw</code>

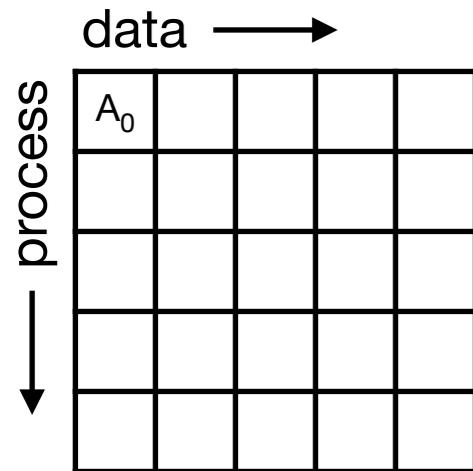
```
MPI.Comm.Alltoall(sendbuff, recvbuff)
```

```
MPI.Comm.Alltoallv(sendbuff, recvbuff)
```

```
MPI.Comm.alltoall(sendobj)
```

See MPI/ex10_alltoall.py

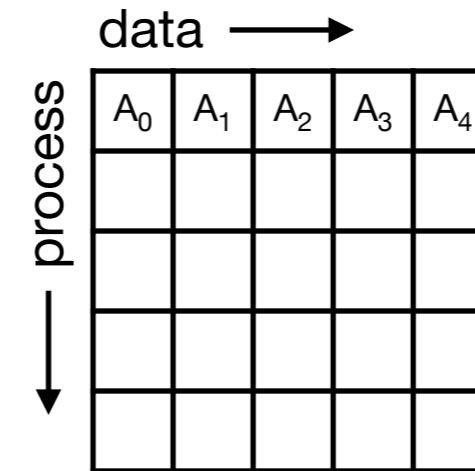
Summary of Collective Communication



broadcast



A ₀				

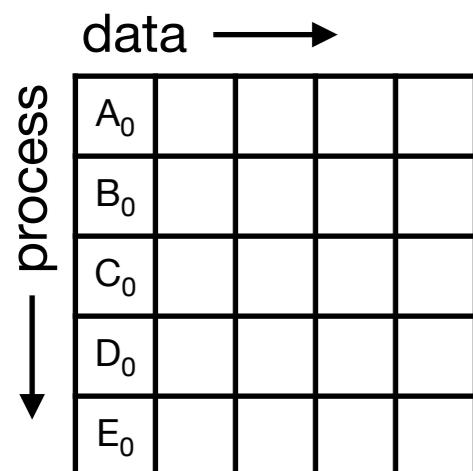


scatter



gather

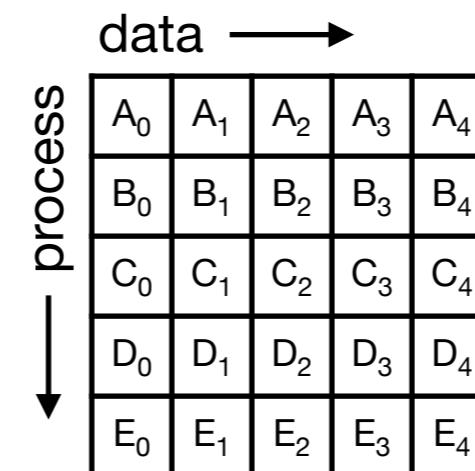
A ₀				
A ₁				
A ₂				
A ₃				
A ₄				



allgather



A ₀	B ₀	C ₀	D ₀	E ₀
A ₀	B ₀	C ₀	D ₀	E ₀
A ₀	B ₀	C ₀	D ₀	E ₀
A ₀	B ₀	C ₀	D ₀	E ₀
A ₀	B ₀	C ₀	D ₀	E ₀



alltoall



A ₀	B ₀	C ₀	D ₀	E ₀
A ₁	B ₁	C ₁	D ₁	E ₁
A ₂	B ₂	C ₂	D ₂	E ₂
A ₃	B ₄	C ₃	D ₃	E ₃
A ₄	B ₅	C ₄	D ₄	E ₄

Reduce / Allreduce

- Performs a global reduce operation across all members of a group.

Communication Mode	Blocking	Non-blocking
Standard	<code>MPI.Comm.Reduce</code> <code>MPI.Comm.Allreduce</code> <code>MPI.Comm.reduce</code> <code>MPI.Comm.allreduce</code>	<code>MPI.Comm.Ireduce</code> <code>MPI.Comm.Iallreduce</code>

```
MPI.Comm.Reduce(sendbuff, recvbuff, op=MPI.SUM, root=0)
```

See MPI/ex11_reduce.py

MPI Operation Object

- Use with Reduce / Allreduce to perform reduction operation

MPI.MAX	Returns the maximum element
MPI.MIN	Returns the minimum element
MPI.SUM	Sums the elements
MPI.PROD	Multiplies all elements

Reduce_scatter

- Performs a global reduce operation across all members of a group and scatter the result.

Communication Mode	Blocking	Non-blocking
Standard	<code>MPI.Comm.Reduce_scatter</code> <code>MPI.Comm.Reduce_scatter_block</code>	<code>MPI.Comm.Ireduce_scatter</code> <code>MPI.Comm.Ireduce_scatter_block</code>

```
MPI.Comm.Reduce_scatter(sendbuff, recvbuff, recvcounts=None,  
op=SUM)
```

```
MPI.Comm.Reduce_scatter_block(sendbuff, recvbuff, op=SUM)
```

See MPI/ex12_reduce_scatter.py