Introduction to MPI

Branislav Jansík

IT4Innovations
national10£$01
supercomputing
center0!£0#010

Resources

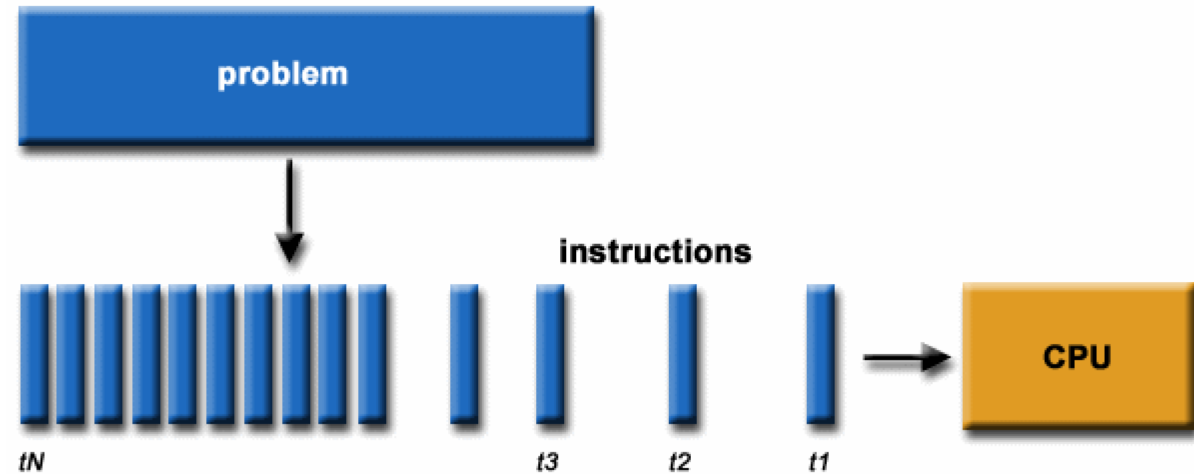
https://computing.llnl.gov/tutorials/mpi/

http://www.mpi-forum.org/

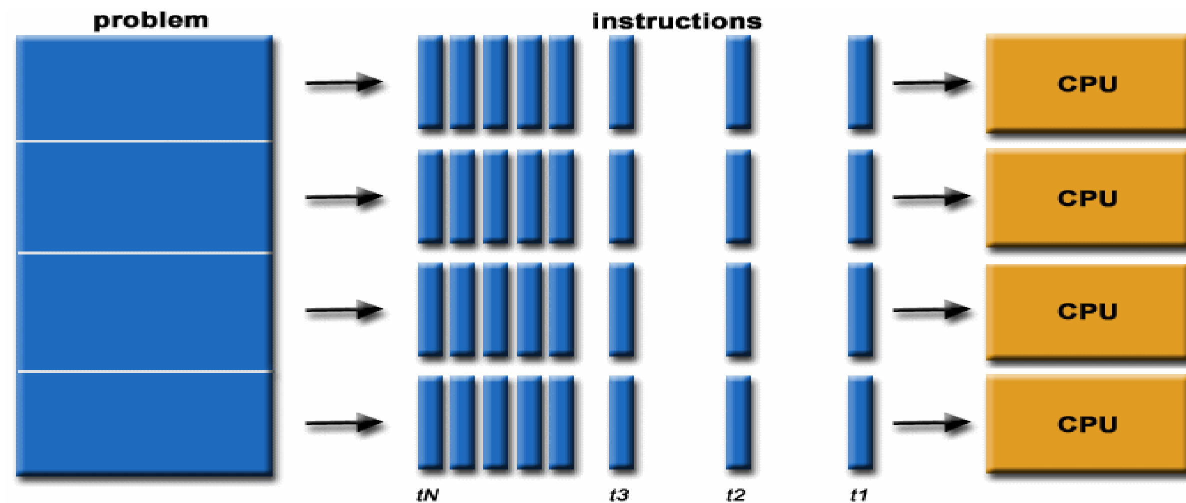
https://www.open-mpi.org/doc/

What is parallel computing

Serial



Parallel



What is MPI?

- MPI: Message Passing Interface
  - The MPI Forum organized in 1992 with broad participation by:
    - Vendors: IBM, Intel, TMC, SGI, Convex, Meiko
    - Portability library writers: PVM, p4
    - Users: application scientists and library writers
    - MPI-1 finished in 18 months
  - Incorporates the best ideas in a "standard" way
    - Each function takes fixed arguments
    - Each function has fixed semantics
      - Standardizes what the MPI implementation provides and what the application can and cannot expect
      - Each system can implement it differently as long as the semantics match
- MPI is not…
  - a language or compiler specification
  - a specific implementation or product

Slide courtesy Pavan Balaji

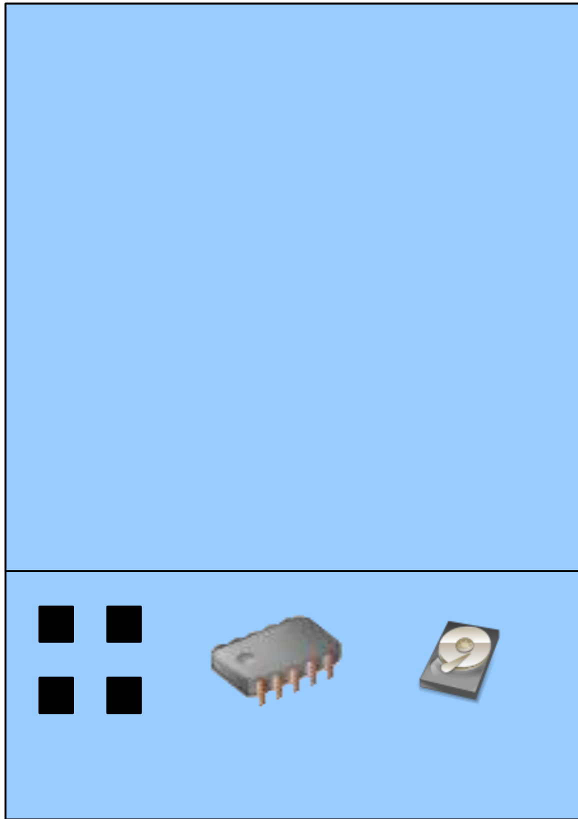
Following MPI Standards

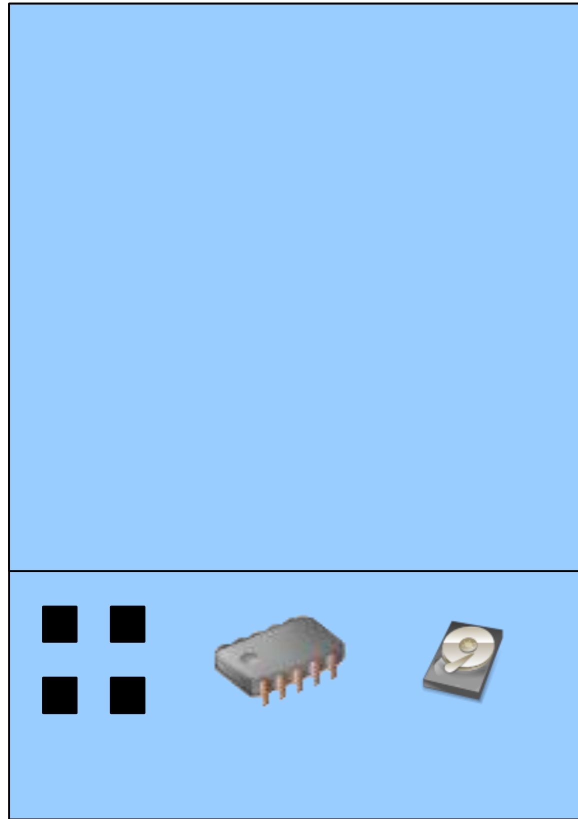
- MPI-2 was released in 1997
  - Several additional features including MPI + threads, MPI-I/O, remote memory access functionality and many others
- MPI-2.1 (2008) and MPI-2.2 (2009) were recently released with some corrections to the standard and small features
- MPI-3 (2012) added several new features to MPI
- The Standard itself:
  - at http://www.mpi-forum.org
  - All MPI official releases, in both postscript and HTML
- Other information on Web:
  - at http://www.mcs.anl.gov/mpi
  - pointers to lots of material including tutorials, a FAQ, other MPI pages

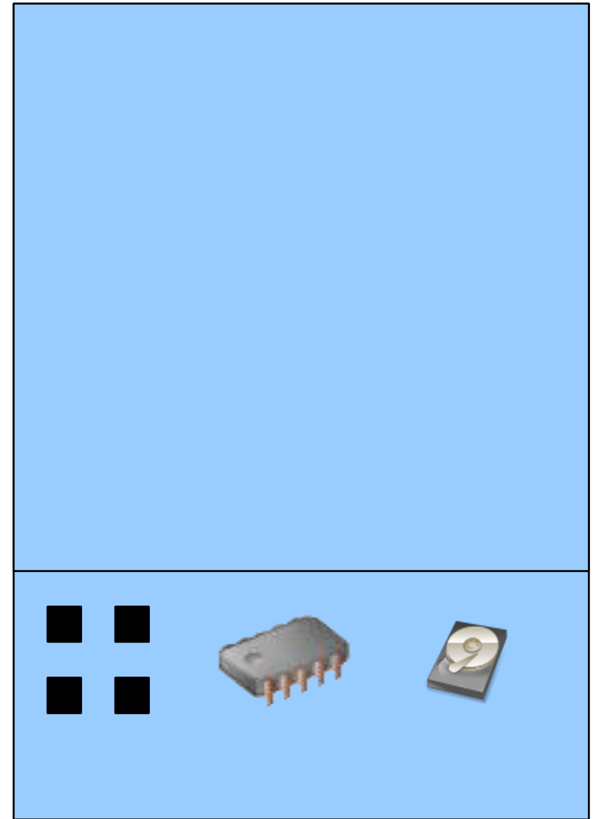
MPI 3.1: Latest Version of MPI Standard

- Available from www.mpi-forum.org/docs/docs.html
- Minor updates/corrections to MPI-3

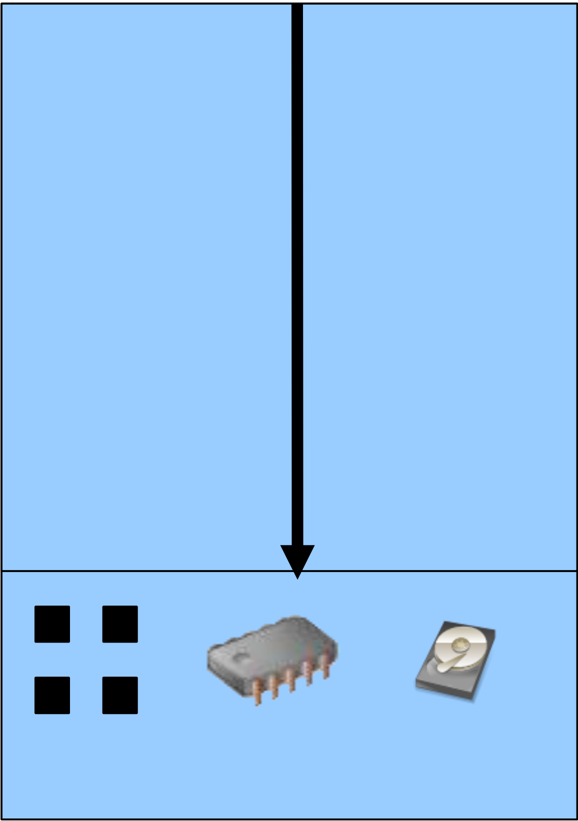
Node
Process
Rank

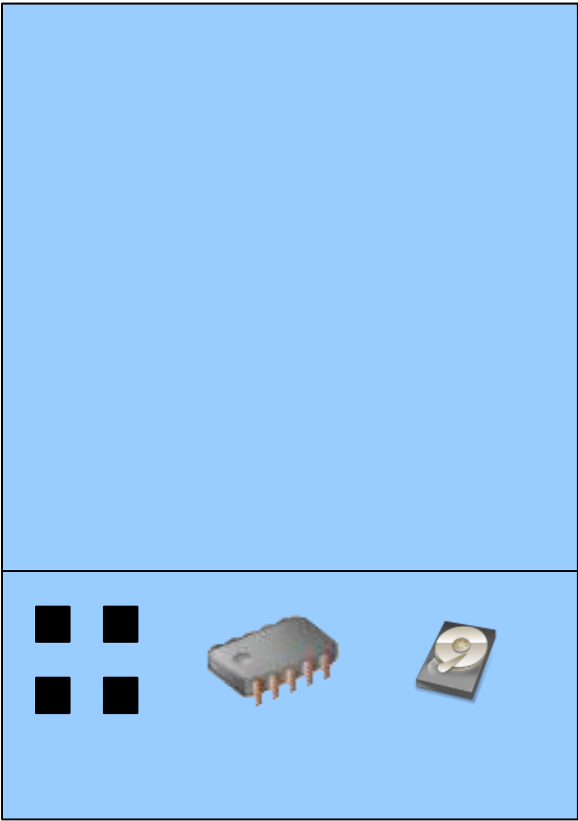


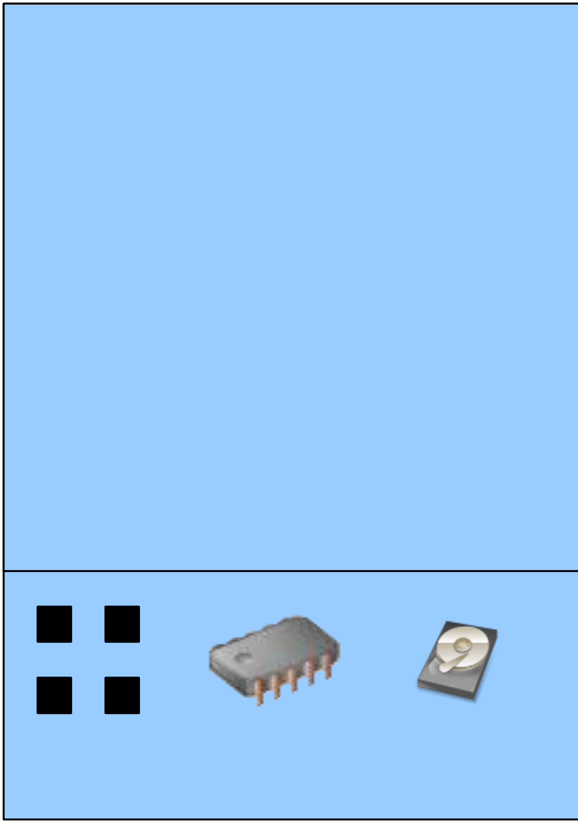




Node
Process
Rank



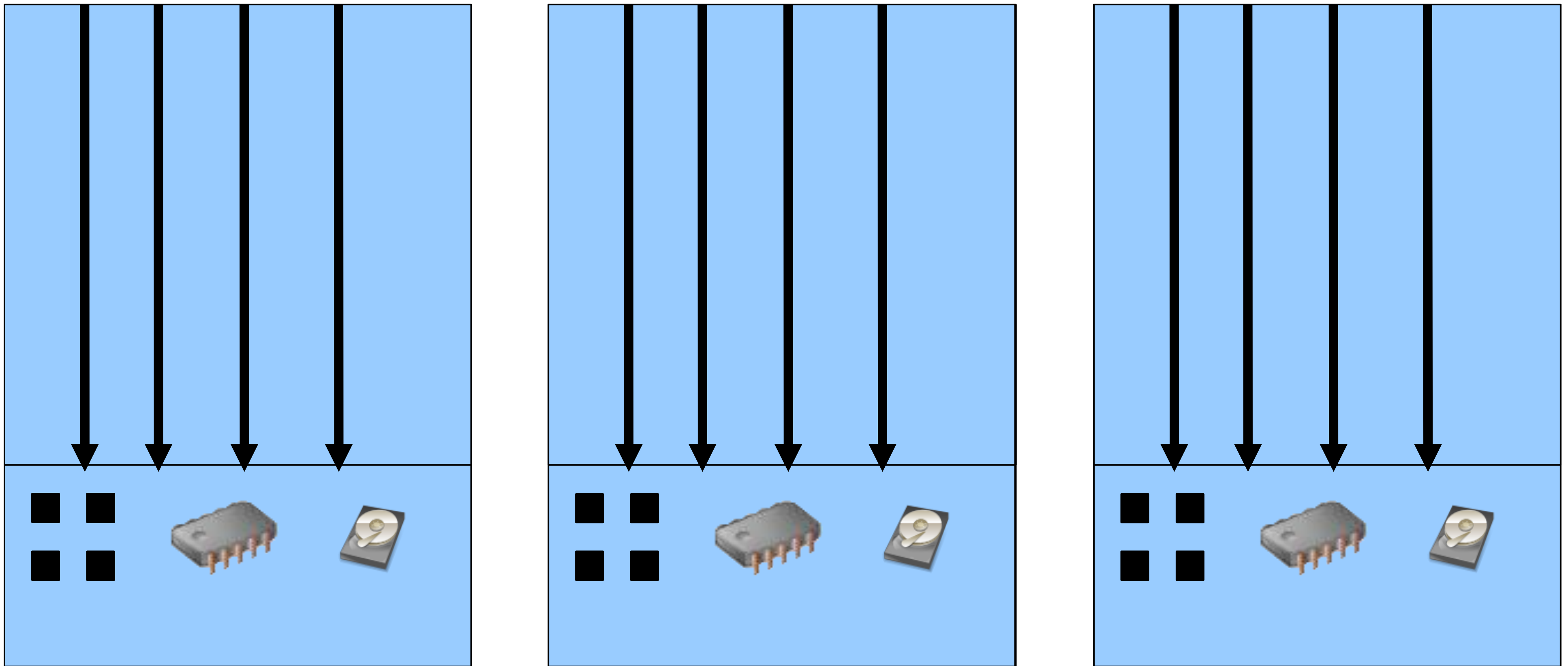




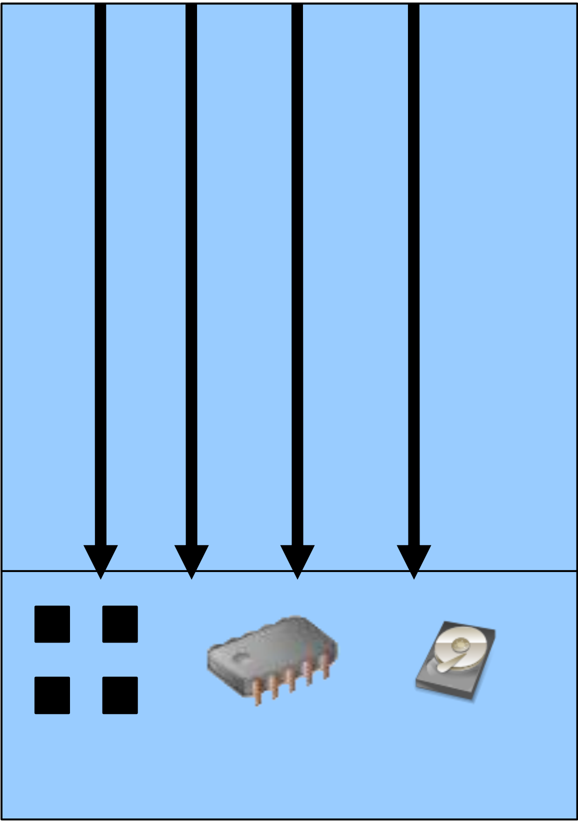
MPI (Message Passing Interface)

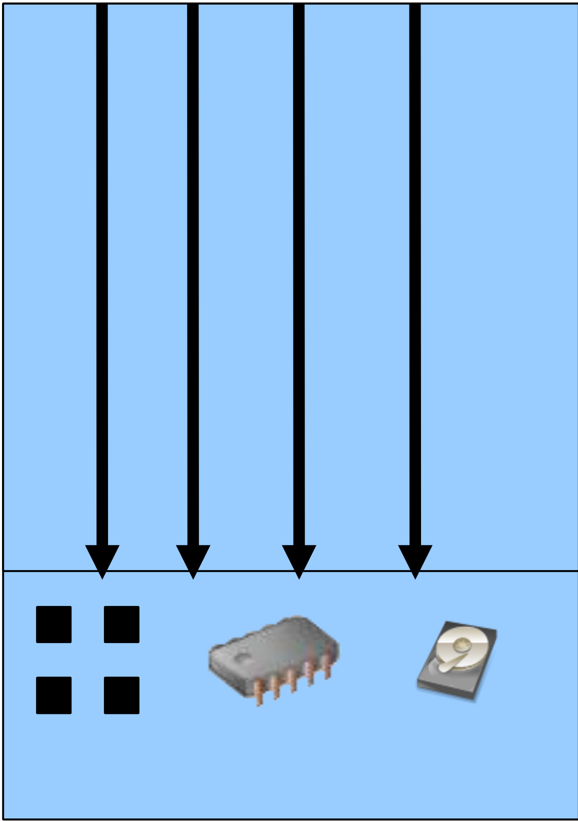
multi-process, distributed memory parallelism

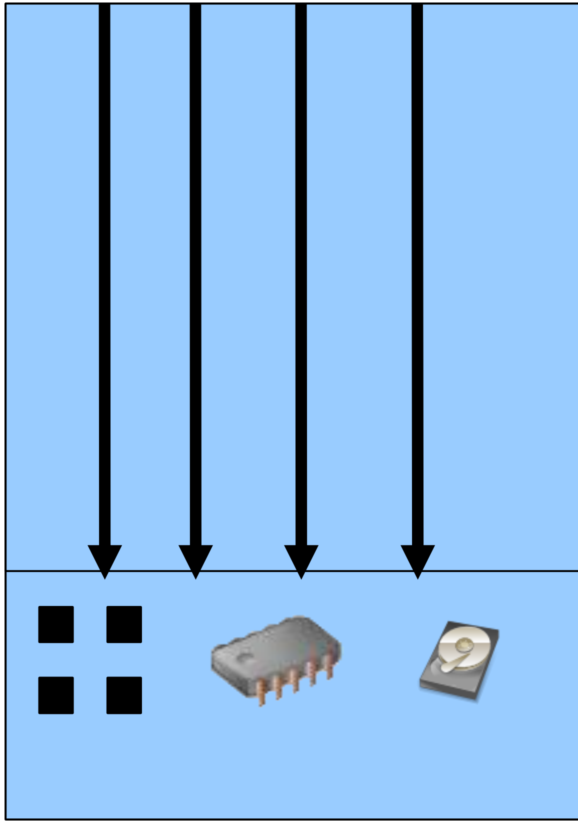
SPMD



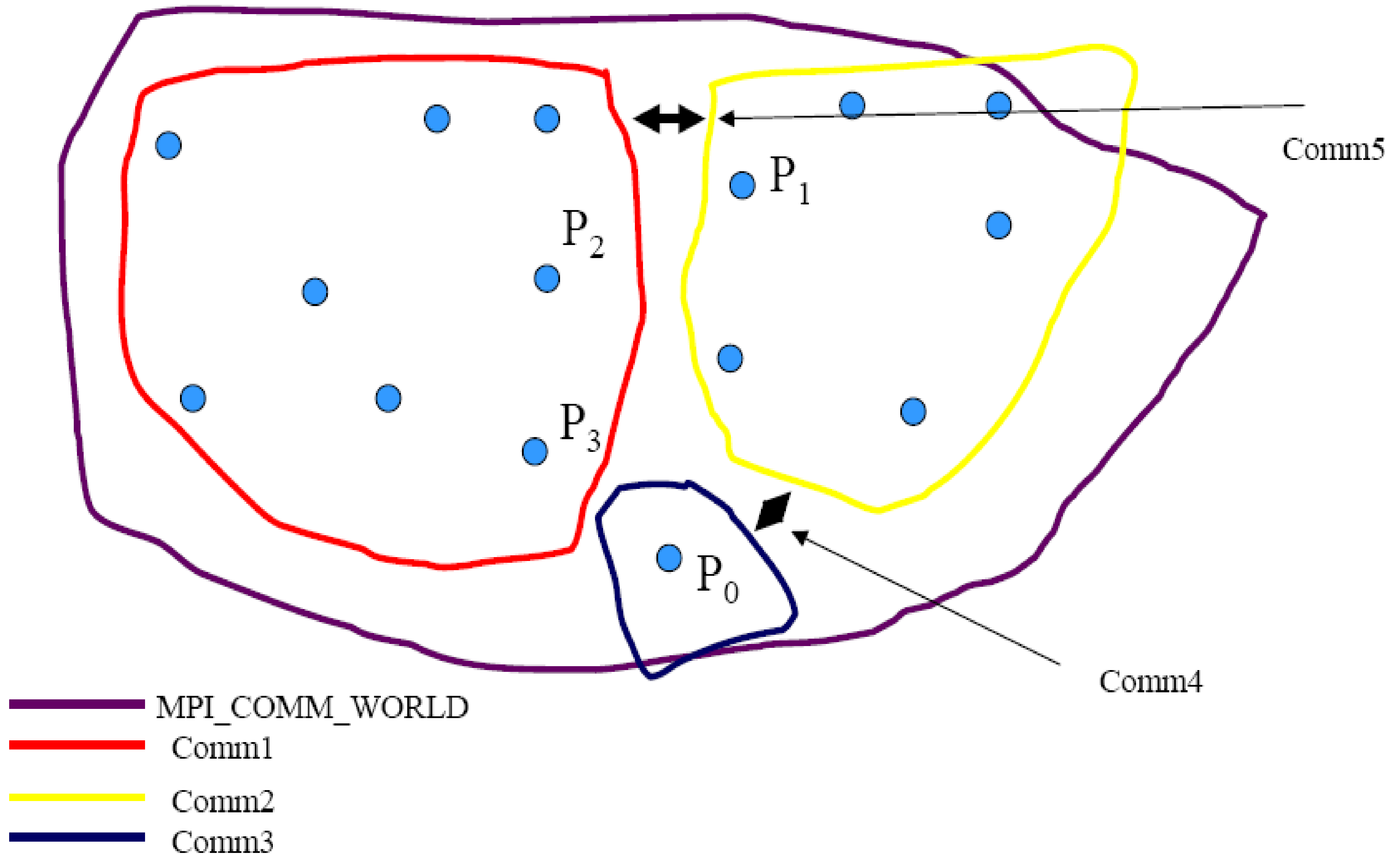
Node
Process
Rank







Communicators and groups



Courtesy David Cronk

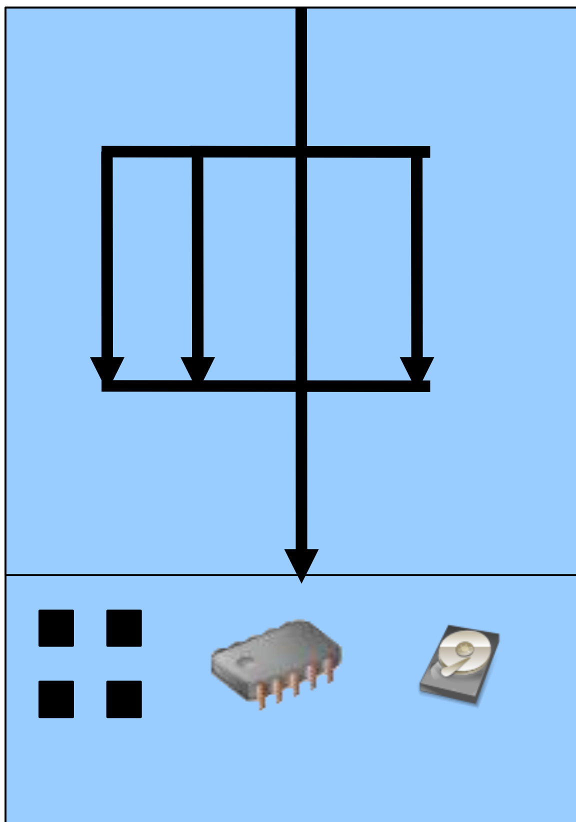
Communicators and groups

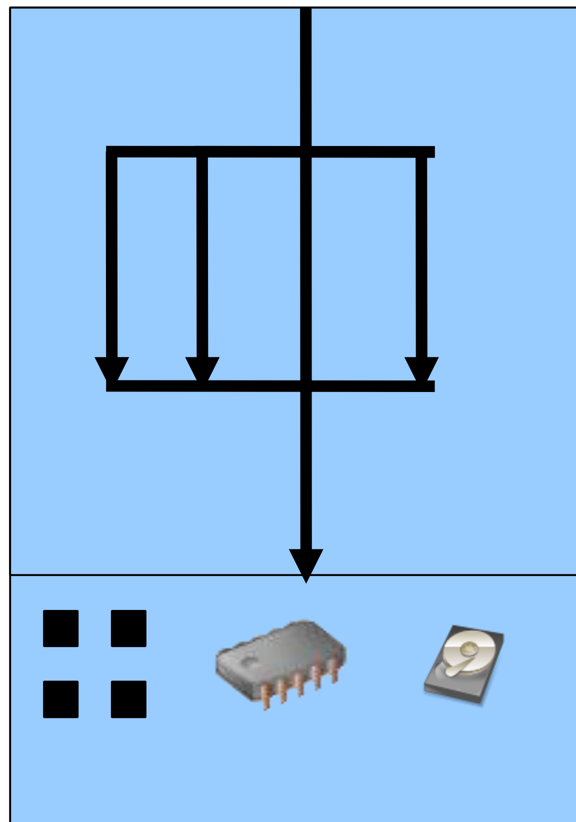
Group is separate object, has own handle
Comminicator is yet another object, has own handle
They are the same from programmer's perspective

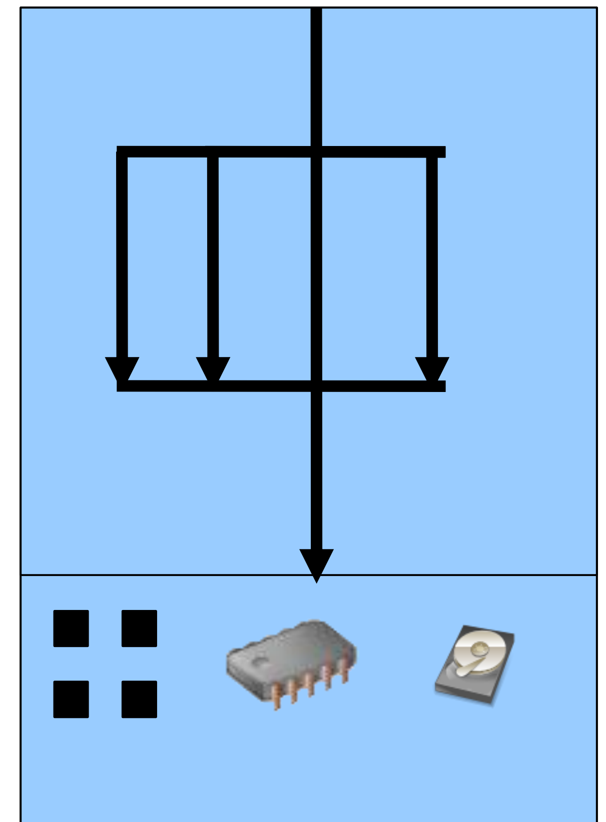
MPI_COMM_GROUP(in: Comm, out: group_handle)
MPI_GROUP_INCL(), MPI_GROUP_RANK()
MPI_COMM_CREATE()

MPI / Open MP Hybrid

multi-thread multi-process, shared/distributed memory parallelism







How is MPI implemented?

MPI

Implemented as standalone library of MPI_routines
user calls MPI_routines from his code
bunch of standalone support utilities

MPI

You design and write the code
You call MPI_routines when appropriate
MPI_routines take care of delivering messages (communication,data) among instances of the program across different nodes for you!

ls /apps/impi/5.1.3.181/lib64

libfmpich.so libjmi_pbs.so libjmi_pbs.so.1 libjmi_pbs.so.1.0 libjmi_slurm.so libjmi_slurm.so.1 libjmi_slurm.so.1.1 libmpi.a libmpichcxx.so libmpichf90.so libmpich.so libmpicxx.a libmpicxx.so libmpicxx.so.12 libmpicxx.so.12.0 libmpi.dbg libmpi_dbg.a libmpi_dbg_mt.a libmpi_dbg_mt.so libmpi_dbg_mt.so.3.2 libmpi_dbg_mt.so.4 libmpi_dbg_mt.so.5 libmpi_dbg.so libmpi_dbg.so.3.2 libmpi_dbg.so.4 libmpi_dbg.so.5 libmpifort.a libmpifort.so libmpifort.so.12 libmpifort.so.12.0 libmpigc4.a libmpigc4.so libmpigc4.so.3.2 libmpigc4.so.4 libmpigc4.so.5 libmpigf.a libmpigf.so libmpigf.so.3.2 libmpigf.so.4 libmpigf.so.5 libmpigi.a libmpi_gpfs.so libmpi_gpfs.so.5.0 libmpiic4.a libmpiic4.so libmpiic4.so.3.2 libmpiif.a libmpiif.so libmpiif.so.3.2 libmpi_ilp64.a libmpi_ilp64.so libmpi_ilp64.so.4 libmpi_ilp64.so.4.1 libmpi_lustre.so libmpi_lustre.so.4.0 libmpi_lustre.so.4.1 libmpi_lustre.so.5.0 libmpi_mt.a libmpi_mt.dbg libmpi_mt.so libmpi_mt.so.12 libmpi_mt.so.3.2 libmpi_mt.so.4 libmpi_mt.so.5 libmpi_panfs.so libmpi_panfs.so.4.0 libmpi_panfs.so.4.1 libmpi_panfs.so.5.0 libmpi_pvfs2.so libmpi_pvfs2.so.4.0 libmpi_pvfs2.so.4.1 libmpi_pvfs2.so.5.0 libmpi.so libmpi.so.12 libmpi.so.3.2 libmpi.so.4 libmpi.so.5 libmpitune.so libmpitune.so.1 libmpitune.so.1.0 libmpl.so libopa.so libtmip_mx.so libtmip_mx.so.1.0 libtmip_psm2.so libtmip_psm2.so.1.0 libtmip_psm.so libtmip_psm.so.1.0 libtmip_psm.so.1.1 libtmip_psm.so.1.2 libtmi.so libtmi.so.1.0 libtmi.so.1.1 libtmi.so.1.2 libtvmpi.so libtvmpi.so.12 libtvmpi.so.12.0 libtvmpi.so.3.2 libtvmpi.so.4 libtvmpi.so.5

ls /apps/impi/5.1.3.181/bin

compchk.sh cpuinfo hydra_nameserver hydra_persist IMB-MPI1 IMB-NBC IMB-RMA Llama llamaMPIClient.py mpd mpdallexit mpdallexit.py mpdboot mpdboot.py mpdcheck mpdcheck.py mpdchkpyver.py mpdcleanup mpdcleanup.py mpdexit mpdexit.py mpdgdbdrv.py mpdhelp mpdhelp.py mpdkilljob mpdkilljob.py mpdlib.py mpdlistjobs mpdlistjobs.py mpdman.py mpd.py mpdringtest mpdringtest.py mpdroot mpdrun mpdrun.py mpdsigjob mpdsigjob.py mpdtrace mpdtrace.py mpicc mpicleanup mpicxx mpiexec mpiexec.hydra mpiexec.py mpif77 mpif90 mpifc mpigcc mpigxx mpiicc mpiicpc mpiifort mpirun mpitune mpivars.csh mpivars.csh.orig.easybuild mpivars.sh mpivars.sh.orig.easybuild mtv.so pmi_proxy tune

Compilation and linking MPI program

Compilation

```
mpiicc -c myprog.c

icc -c myprog.c  -I/apps/impi/5.1.3.181/include64
```

Linking

```
mpiicc myprog.o -o myprog.x

icc myprog.o  -o myprog.x
-L/apps/impi/5.1.3.181/lib64 -lmpi
```

Compilation and linking

```
mpiicc myprog.c -o myprog.x

icc myprog.c -o myprog.x
-I/apps/impi/5.1.3.181/include64
-L/apps/impi/5.1.3.181/lib64 -lmpi
```

Compilation and linking MPI program

Compilation

```
mpiicc -qopenmp -c myprog.c

icc -qopenmp -c myprog.c
-I/apps/impi/5.1.3.181/include64
```

Linking

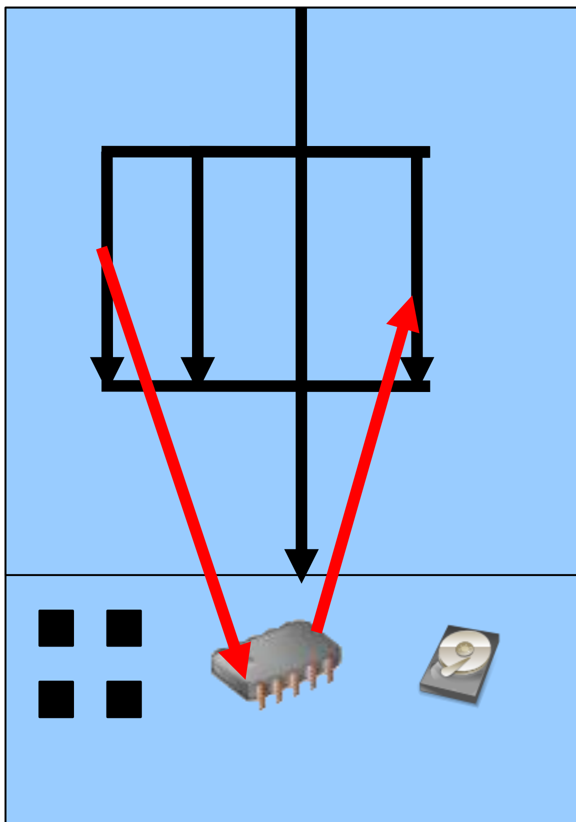
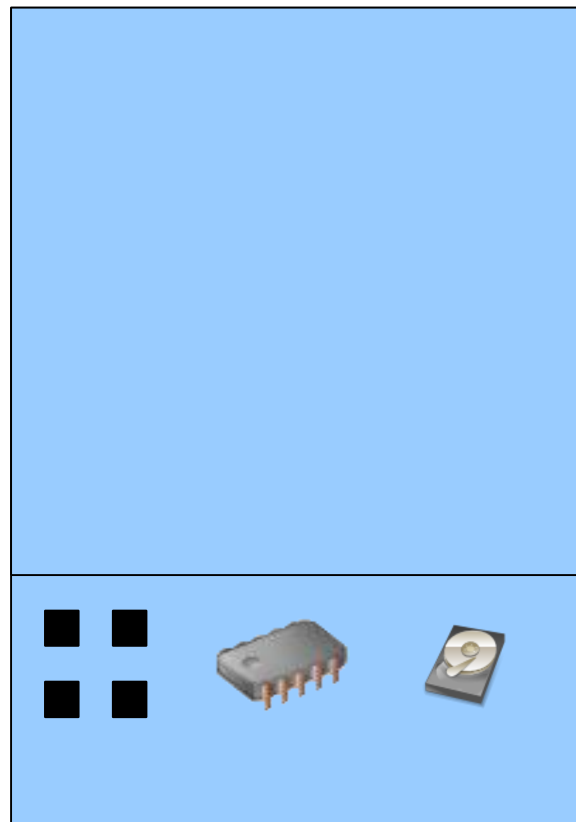
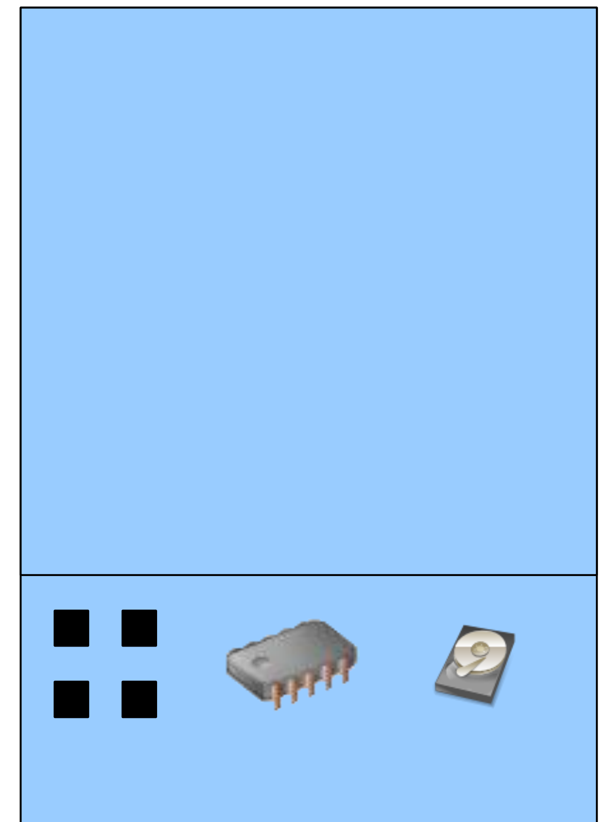
```
mpiicc myprog.o -o myprog.x

icc myprog.o  -o myprog.x
-L/apps/impi/5.1.3.181/lib64 -lmpi
```

Compilation and linking

```
mpiicc myprog.c -o myprog.x

icc -qopenmp myprog.c -o myprog.x
-I/apps/impi/5.1.3.181/include64
-L/apps/impi/5.1.3.181/lib64 -lmpi
```

Open MP

Communication via shared memory

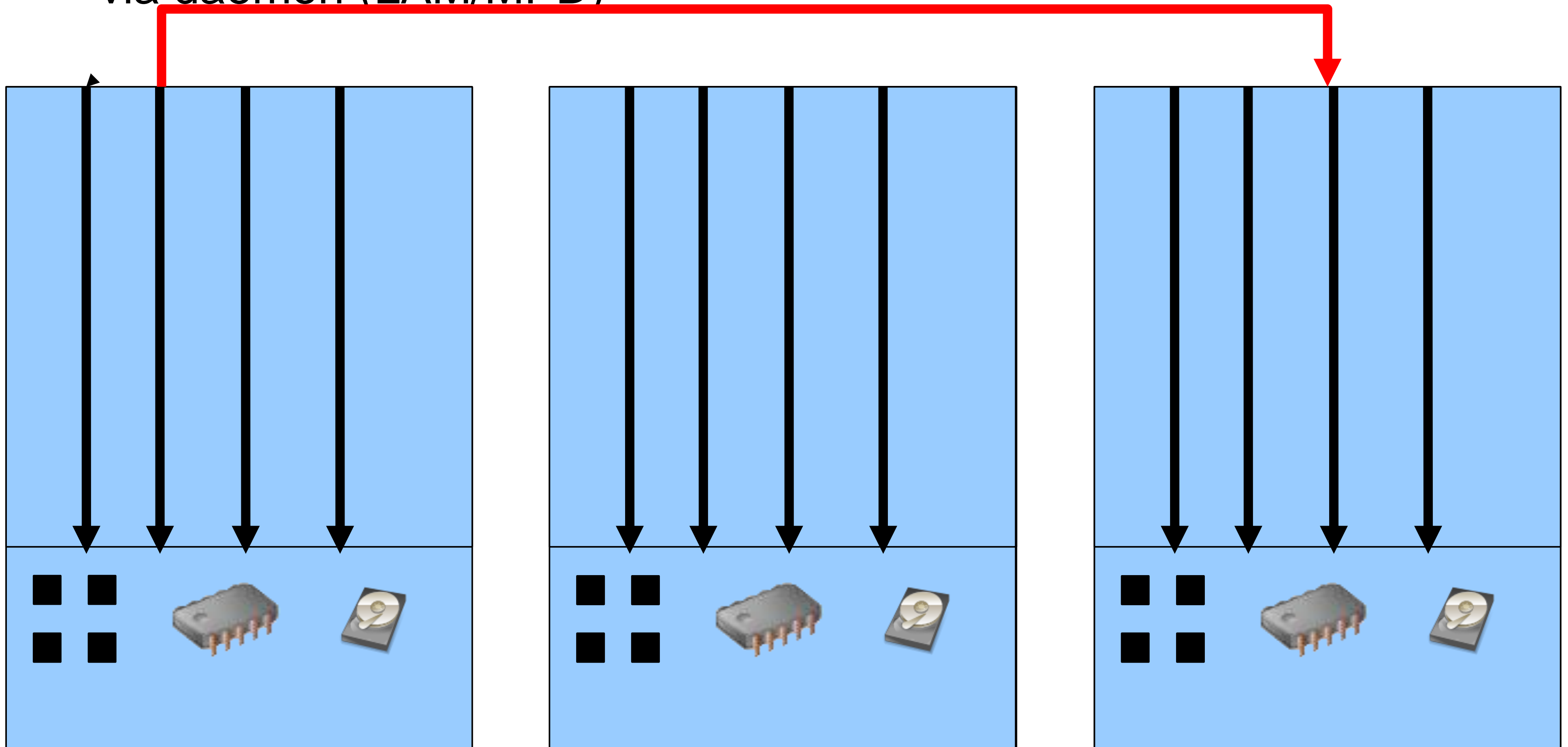




MPI

Communication over interconnect media (infiniband)

Direct

via daemon (LAM/MPD)



MPI

Communication over interconnect media (ethernet/infiniband)

Direct

via daemon (LAM/MPD)

Understanding MPI startup

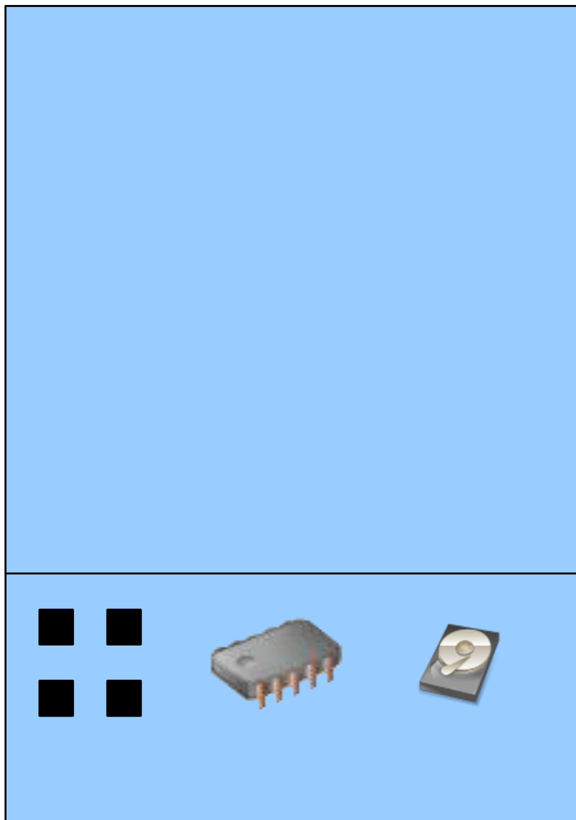
Allocate nodes
Find out which nodes are mine
Start MPD daemon, default 1 per node
Decide how many instances to run
Spawn processes and run job

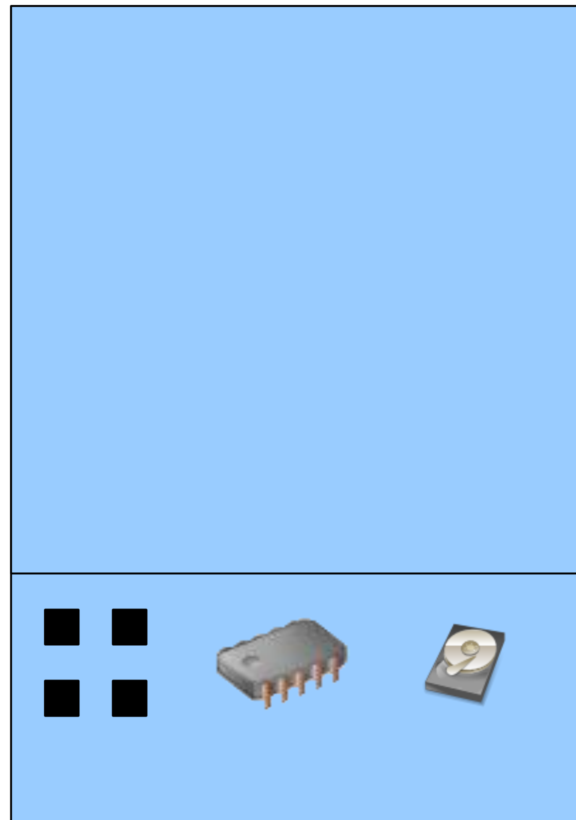
```
qsub -l select=3 [jobscript | -I]
```

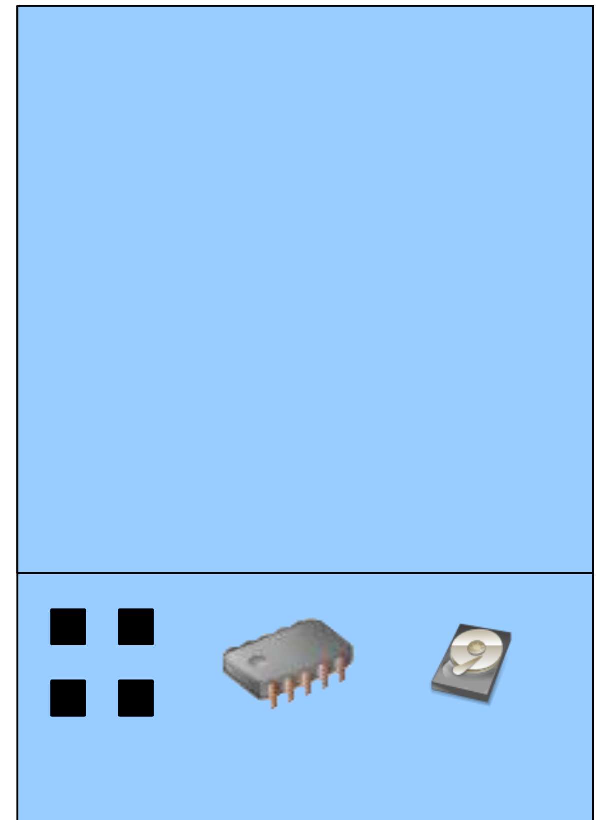
Understanding MPI startup

Allocate nodes
Find out which nodes are mine
Start MPD daemon, default 1 per node
Decide how many instances to run
Spawn processes and run job

$PBS_NODEFILE

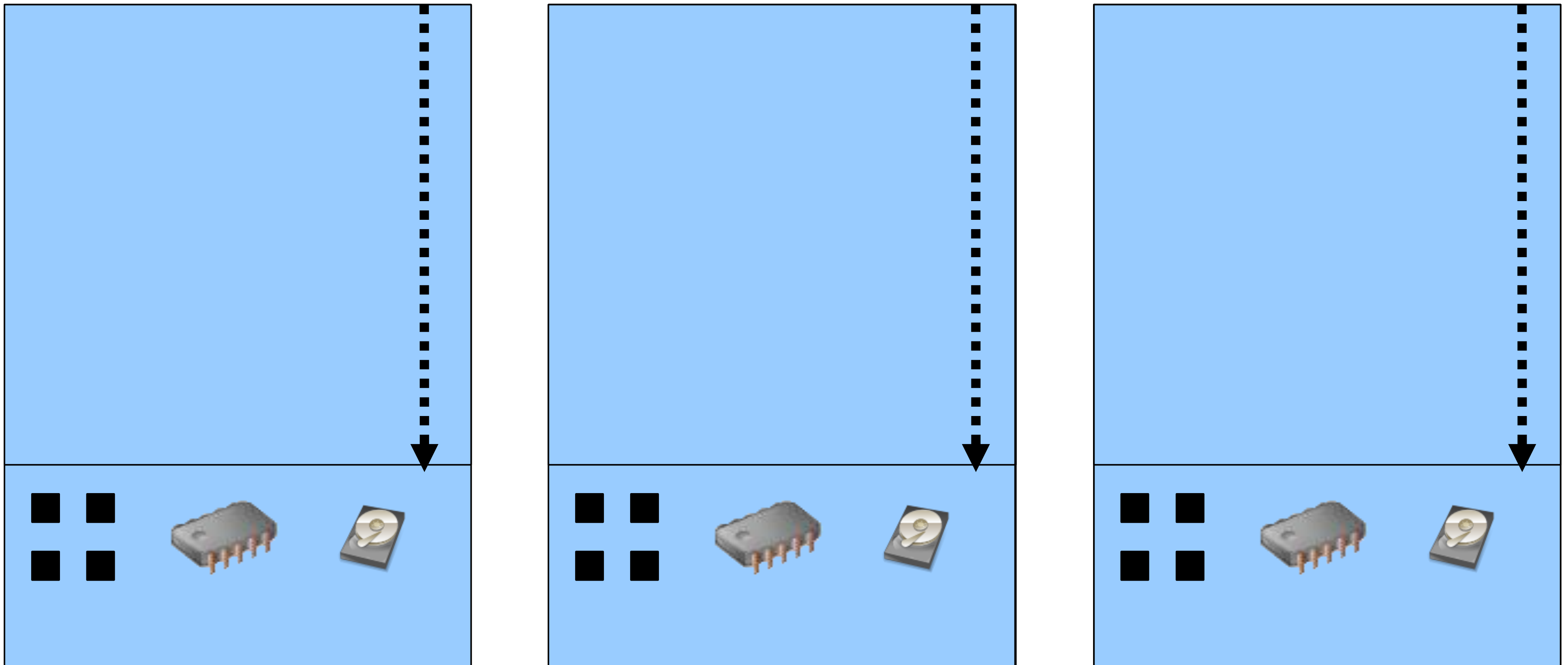






Understanding MPI startup

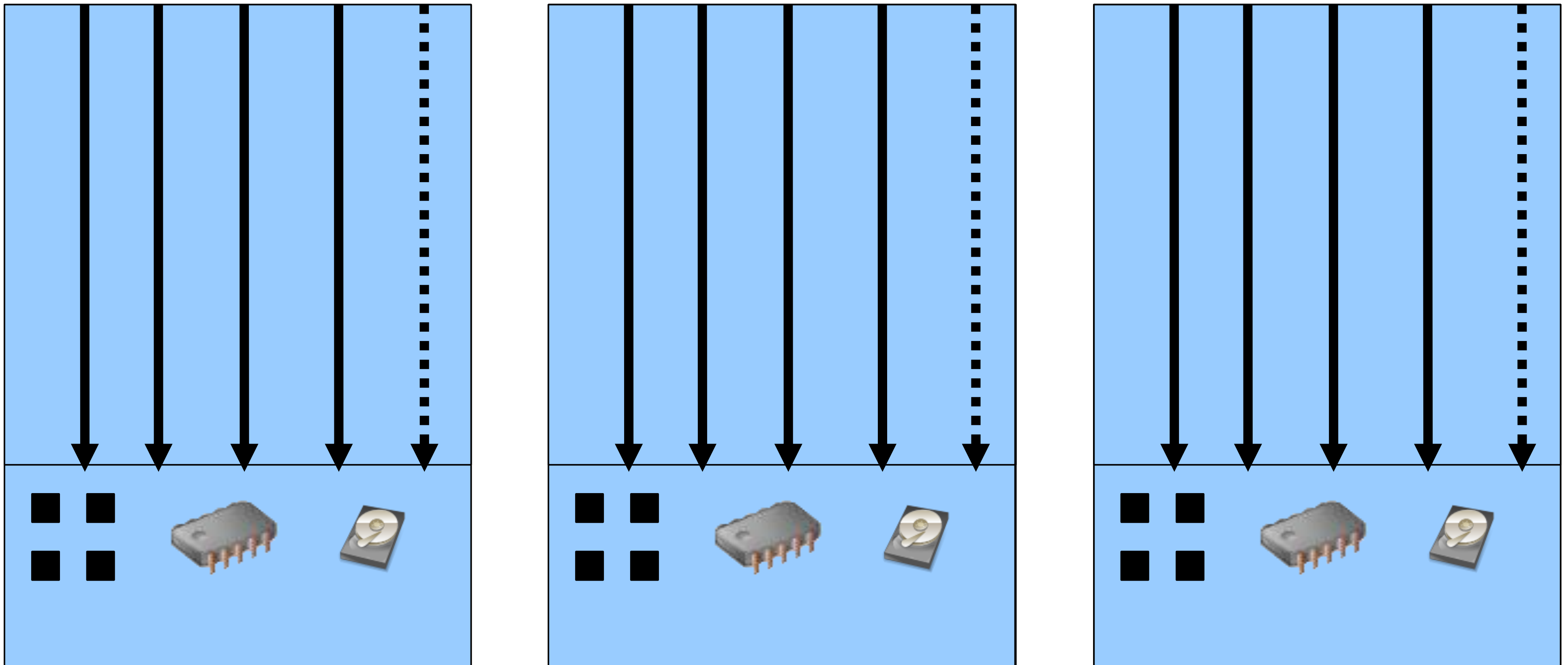
Allocate nodes
Find out which nodes are mine
Start MPD daemon, default 1 per node (via ssh or rsh)
Decide how many instances to run
Spawn processes and run job



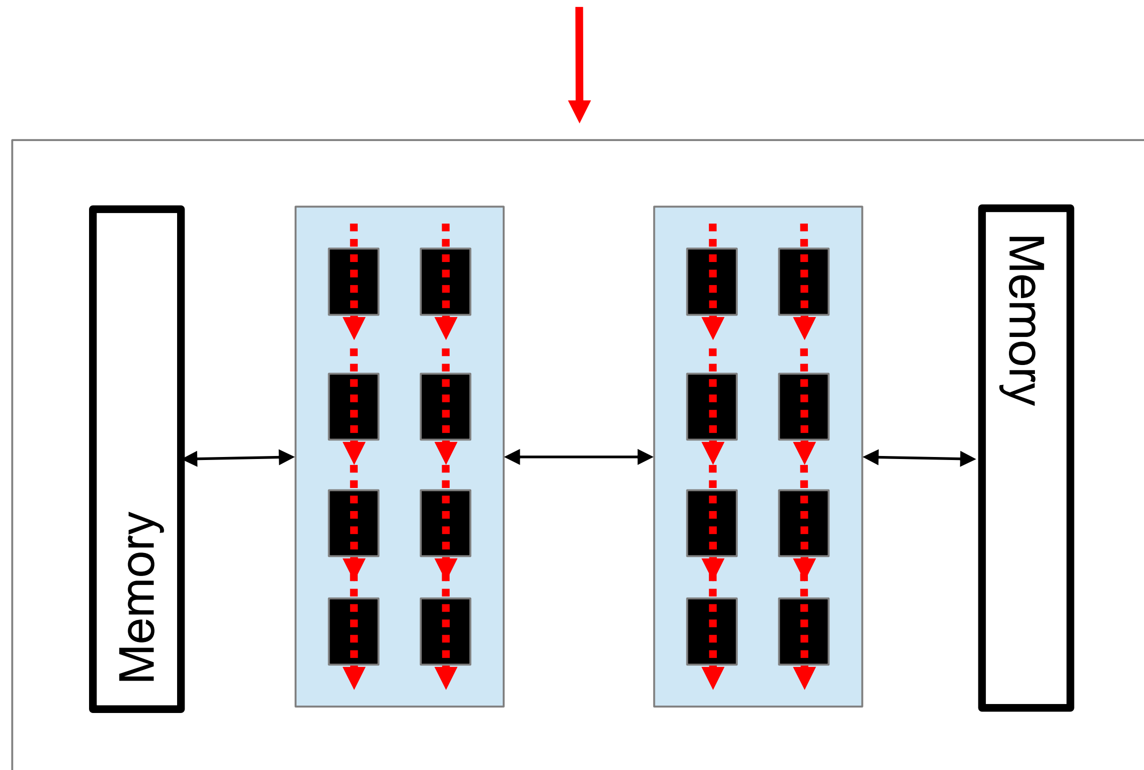
Understanding MPI startup

Allocate nodes
Find out which nodes are mine
Start MPD daemon, default 1 per node
Decide how many instances to run
Spawn processes and run job

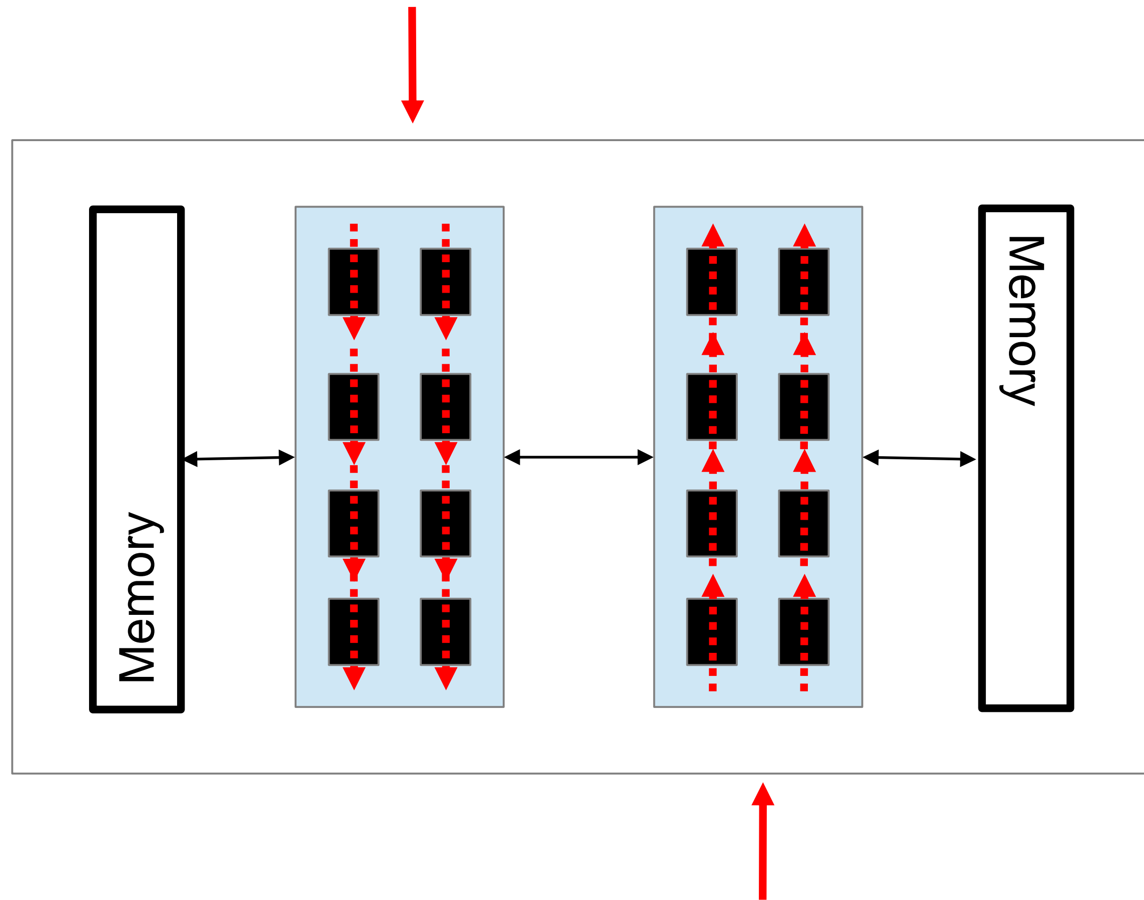
```
mpiexec -n 12 myprog.x
```



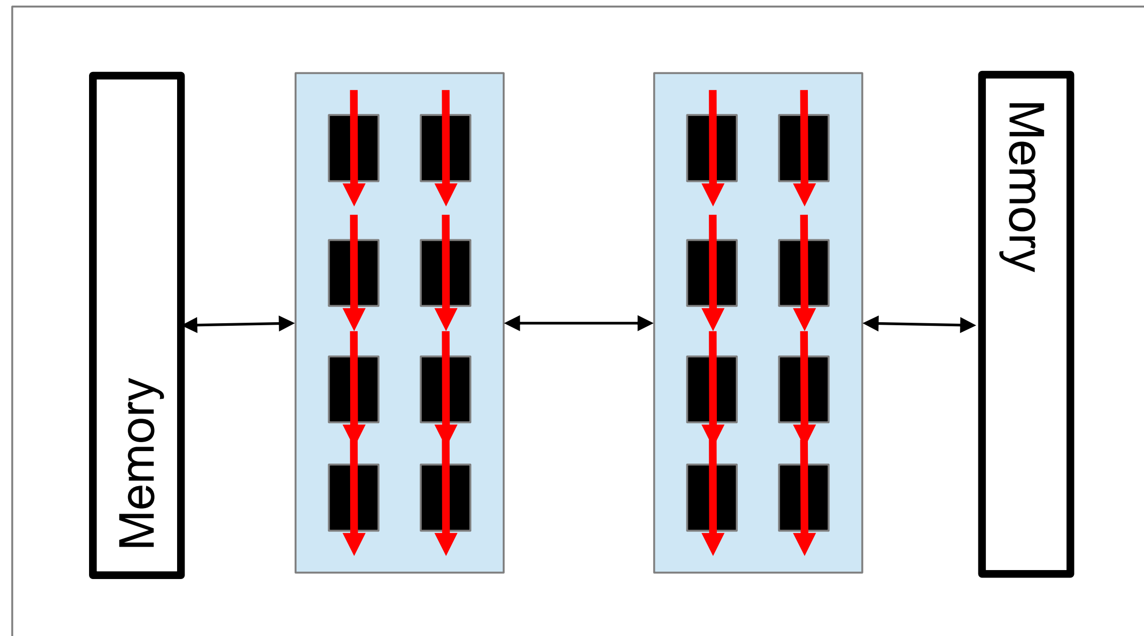
Ways to run MPI programs



Ways to run MPI programs



Ways to run MPI programs



MPI is simple

Initialization and finalization

MPI_Init(), MPI_Comm_rank(), MPI_Comm_size(), MPI_Finalize

Collective communication

MPI_BCAST(), MPI_REDUCE()

Point-to-point communication

MPI_SEND(), MPI_RECV()

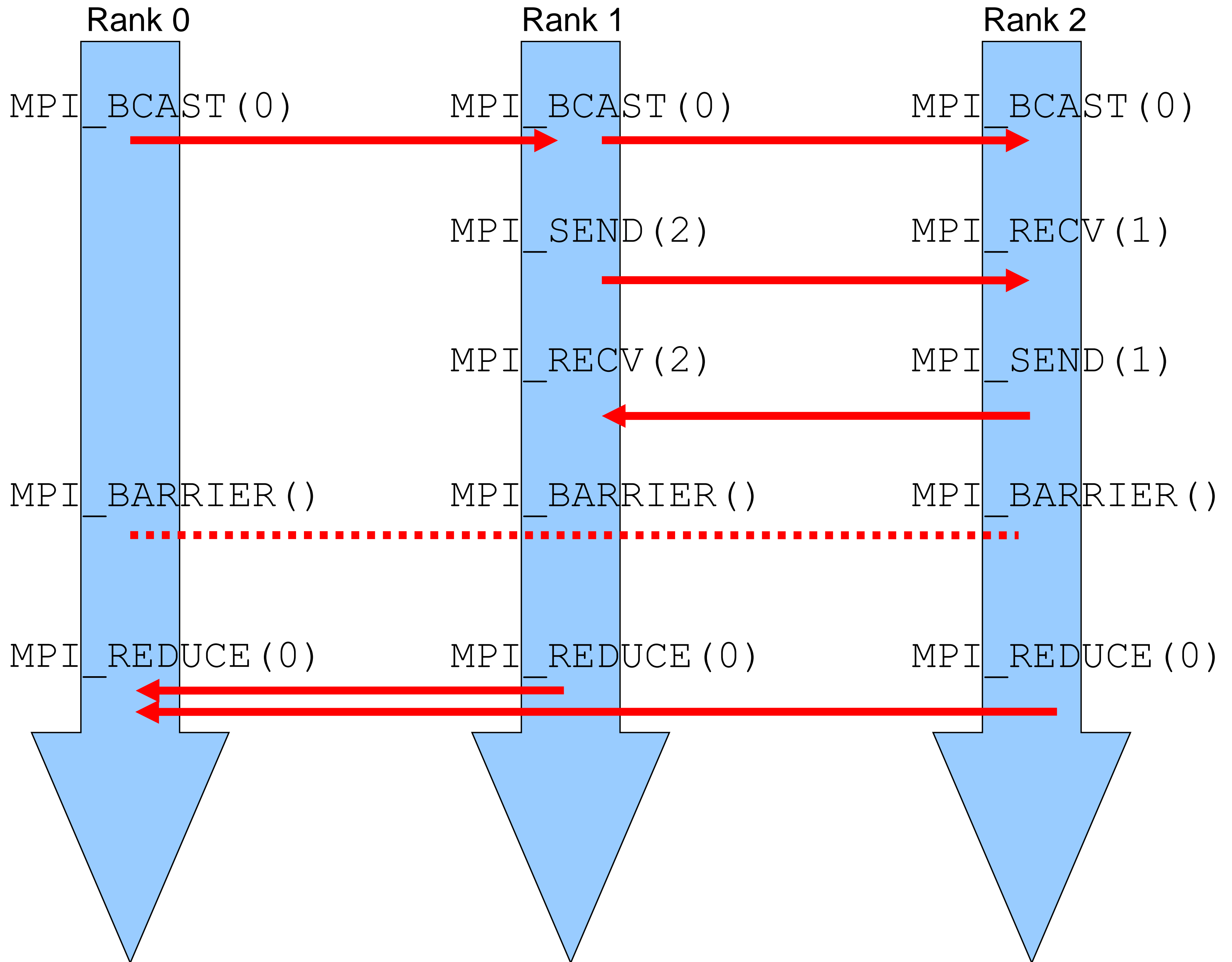
Synchronization

MPI_BARRIER()

Simple MPI Program Identifying Processes

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char ** argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("I am %d of %d\n", rank + 1, size);

    MPI_Finalize();
    return 0;
}
```

Basic requirements for an MPI program


Rank 0
Rank 1
Rank 2
MPI_BCAST(0)
MPI_BCAST(0)
MPI_BCAST(0)
MPI_SEND(2)
MPI_RECV(1)
MPI_RECV(2)
MPI_SEND(1)
MPI_BARRIER()
MPI_BARRIER()
MPI_BARRIER()
MPI_REDUCE(0)
MPI_REDUCE(0)
MPI_REDUCE(0)

Data Communication

- Data communication in MPI is like email exchange
  - One process sends a copy of the data to another process (or a group of processes), and the other process receives it
- Communication requires the following information:
  - Sender has to know:
    - Whom to send the data to (receiver's process rank)
    - What kind of data to send (100 integers or 200 characters, etc)
    - A user-defined "tag" for the message (think of it as an email subject; allows the receiver to understand what type of data is being received)
  - Receiver "might" have to know:
    - Who is sending the data (OK if the receiver does not know; in this case sender rank will be **`MPI_ANY_SOURCE`**, meaning anyone can send)
    - What kind of data is being received (partial information is OK: I might receive *up to* 1000 integers)
    - What the user-defined "tag" of the message is (OK if the receiver does not know; in this case tag will be **`MPI_ANY_TAG`**)

More Details on Describing Data for Communication

- MPI Datatype is very similar to a C or Fortran datatype
  - `int` → `MPI_INT`
  - `double` → `MPI_DOUBLE`
  - `char` → `MPI_CHAR`
- More complex datatypes are also possible:
  - E.g., you can create a structure datatype that comprises of other datatypes → a char, an int and a double.
  - Or, a vector datatype for the columns of a matrix

Slide courtesy Pavan Balaji

MPI Collective Communication

- Communication and computation is coordinated among a group of processes in a communicator
- Tags are not used; different communicators deliver similar functionality
- Non-blocking collective operations in MPI-3
  - Covered in the advanced tutorial (but conceptually simple)
- Three classes of operations: synchronization, data movement, collective computation

Order:

MPI guarantees that messages will not overtake each other.

If a sender sends two messages (Message 1 and Message 2) in succession to the same destination, and both match the same receive, the receive operation will receive Message 1 before Message 2.

If a receiver posts two receives (Receive 1 and Receive 2), in succession, and both are looking for the same message, Receive 1 will receive the message before Receive 2.

Order rules do not apply if there are multiple threads participating in the communication operations.

Fairness:

MPI does not guarantee fairness - it's up to the programmer to prevent "operation starvation".

Example: task 0 sends a message to task 2. However, task 1 sends a competing message that matches task 2's receive. Only one of the sends will complete.

More collective communication

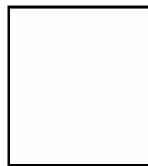
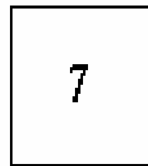
MPI_Bcast

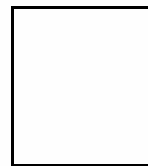
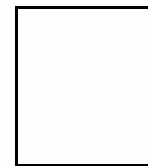
Broadcasts a message to all other processes of that group

```
count = 1;
source = 1;
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

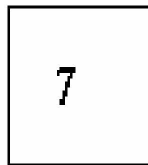
broadcast originates in task 1

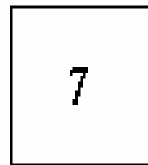
task 0 task 1 task 2 task 3

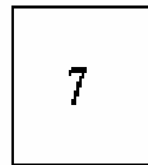



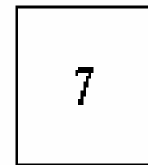



msg (before)









msg (after)

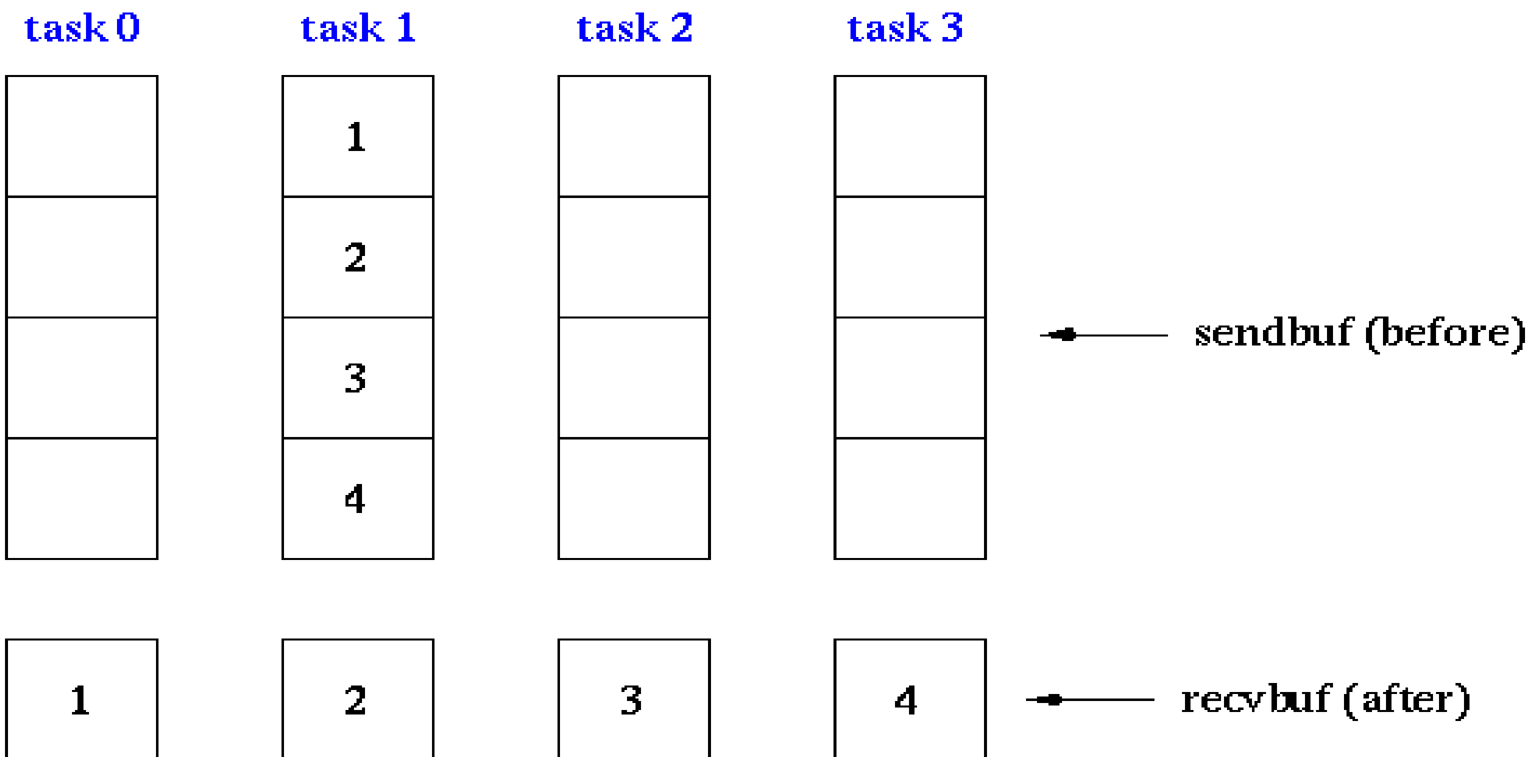
More collective communication

MPI_Scatter

Sends data from one task to all other tasks in a group

```
sendcnt = 1;
recvcnt = 1;
src = 1;
MPI_Scatter(sendbuf, sendcnt, MPI_INT,
            recvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```

task 1 contains the message to be scattered



More collective communication

MPI_Gather

Gathers together values from a group of processes

```
sendcnt = 1;
recvcnt = 1;
src  = 1;                messages will be gathered in task 1
MPI_Gather(sendbuf, sendcnt, MPI_INT,
           recvbuf, recvcnt, MPI_INT,
           src, MPI_COMM_WORLD);
```

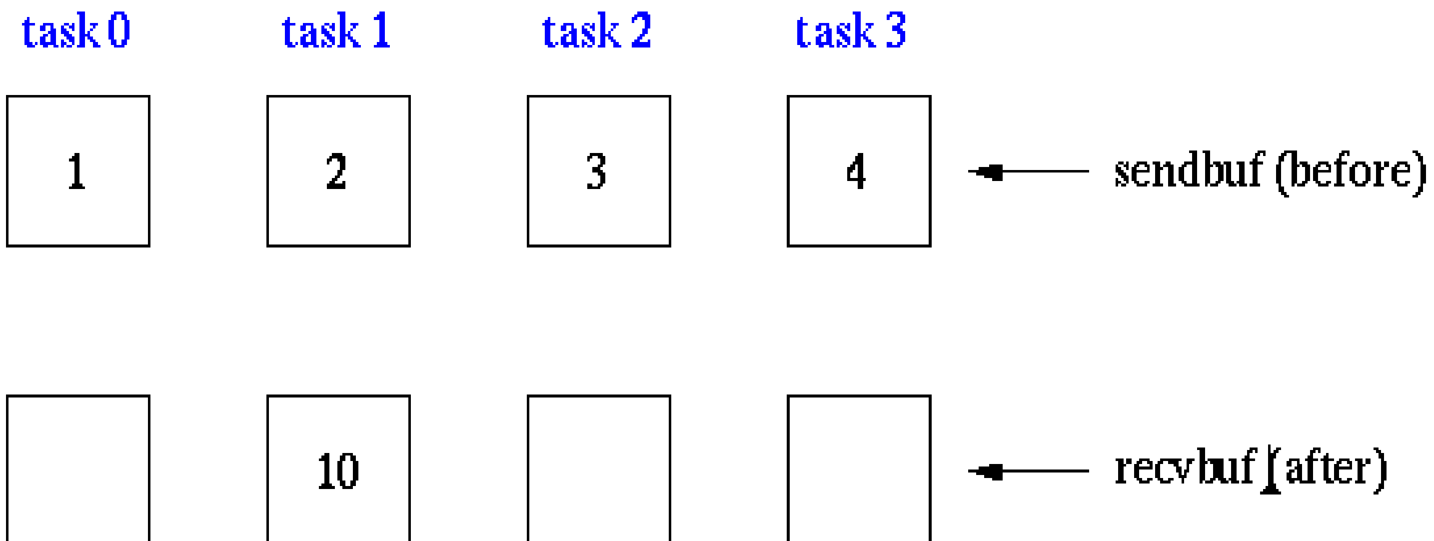
	task 0	task 1	task 2	task 3
sendbuf (before)	1	2	3	4
recvbuf (after)		1		
		2		
		3		
		4		

More collective communication

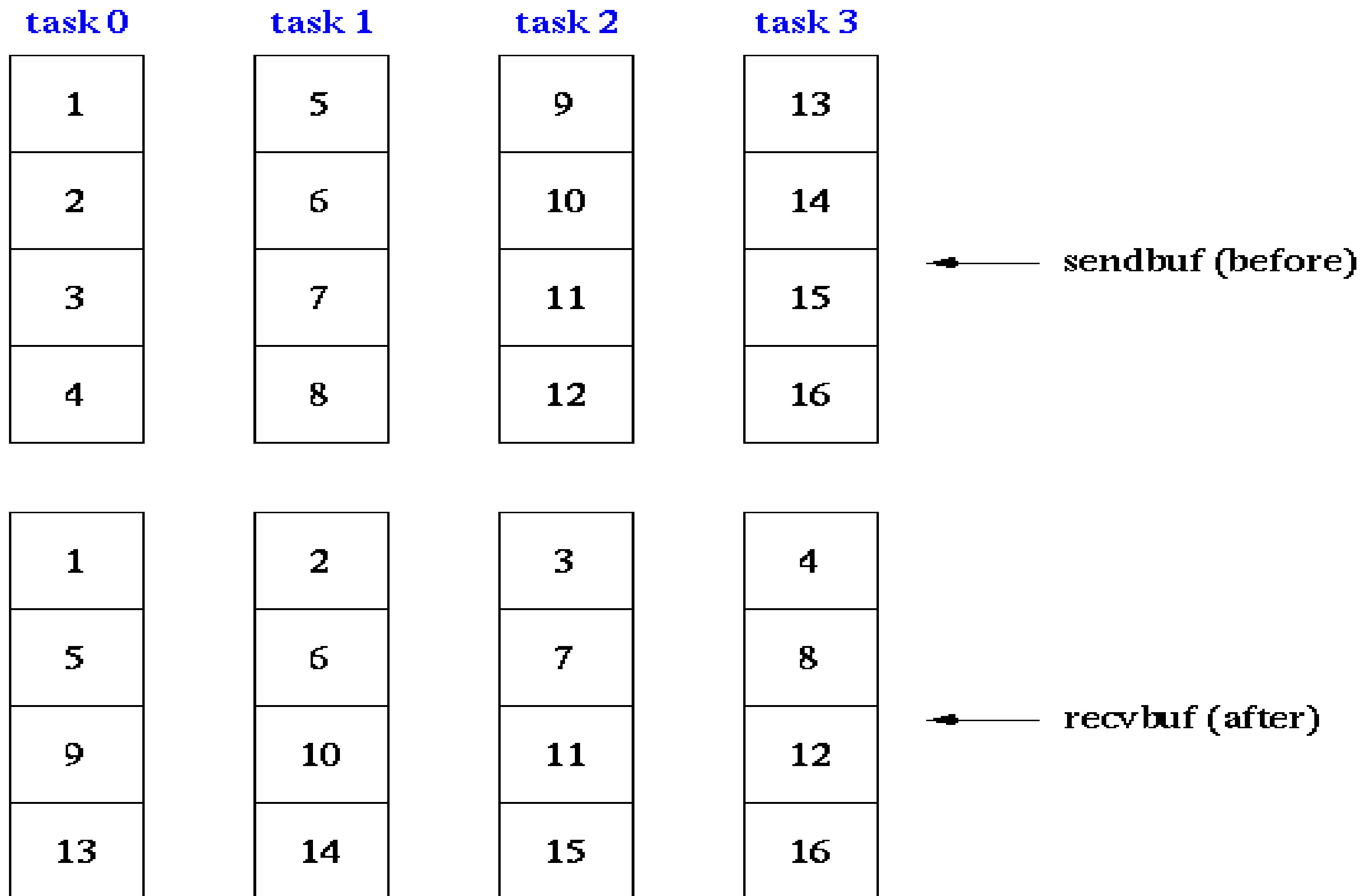
MPI_Reduce

Perform and associate reduction operation across all tasks in the group and place the result in one task

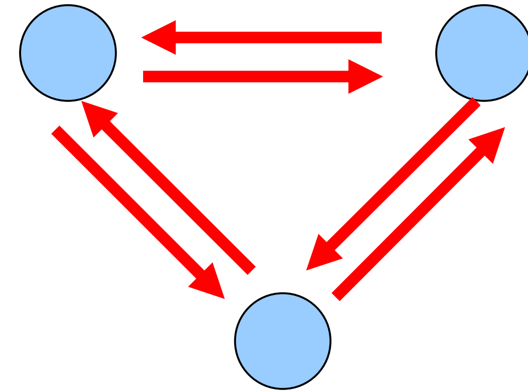
```
count = 1;
dest = 1;                    result will be placed in task 1
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
           dest, MPI_COMM_WORLD);
```



More collective communication



Non-blocking communication



Point-to-point (also collective in MPI2)
MPI_ISEND(), MPI_IRECV()

MPI_ISEND() will return immediately
message will be sent ... at some point
MPI_RECV() will return immediately
message fills buffer whenever it arrives

Synchronization
MPI_WAIT[ALL](), MPI_TEST()

Non-Blocking Communication

- Non-blocking (asynchronous) operations return (immediately) "request handles" that can be waited on and queried
  - `MPI_ISEND(buf, count, datatype, dest, tag, comm, request)`
  - `MPI_IRECV(buf, count, datatype, src, tag, comm, request)`
  - `MPI_WAIT(request, status)`
- Non-blocking operations allow overlapping computation and communication
- One can also test without waiting using **MPI_TEST**
  - **MPI_TEST(request, flag, status)**
- Anywhere you use **MPI_SEND** or **MPI_RECV**, you can use the pair of **MPI_ISEND/MPI_WAIT** or **MPI_IRECV/MPI_WAIT**

Slide courtesy Pavan Balaji

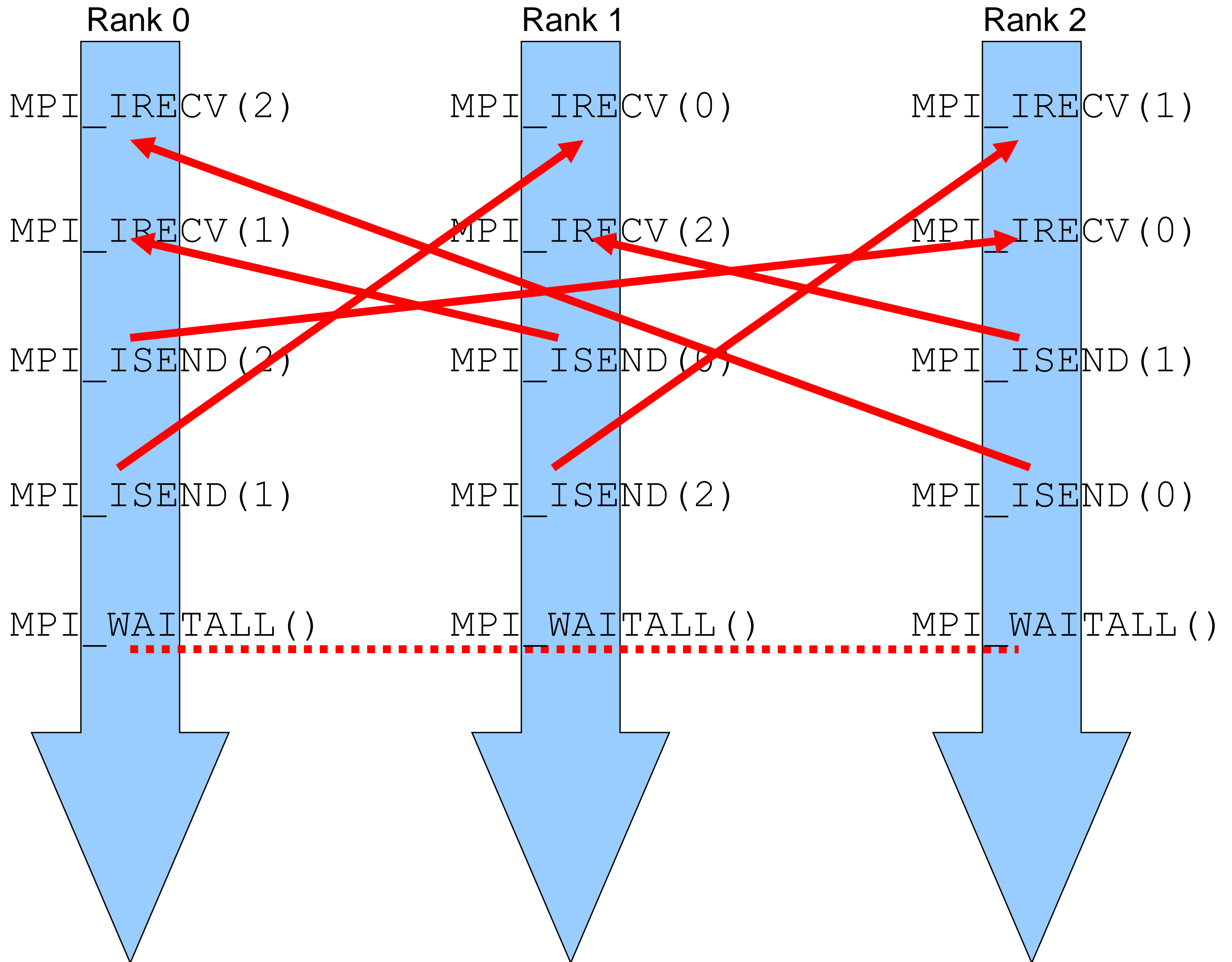
Message Completion and Buffering

- For a communication to succeed:
  - Sender must specify a valid destination rank
  - Receiver must specify a valid source rank (including MPI_ANY_SOURCE)
  - The communicator must be the same
  - Tags must match
  - Receiver's buffer must be large enough
- A send has completed when the user supplied buffer can be reused

```
*buf = 3;
MPI_Send(buf, 1, MPI_INT …)
*buf = 4; /* OK, receiver will always
             receive 3 */
```

```
*buf = 3;
MPI_Isend(buf, 1, MPI_INT …)
*buf = 4; /* Receiver may get 3, 4, or
             anything else */
MPI_Wait(…);
```

- Just because the send completes does not mean that the receive has completed
  - Message may be buffered by the system
  - Message may still be in transit


Rank 0
Rank 1
Rank 2
MPI_IRECV(2)
MPI_IRECV(0)
MPI_IRECV(1)
MPI_IRECV(1)
MPI_IRECV(2)
MPI_IRECV(0)
MPI_ISEND(2)
MPI_ISEND(0)
MPI_ISEND(1)
MPI_ISEND(1)
MPI_ISEND(2)
MPI_ISEND(0)
MPI_WAITALL()
MPI_WAITALL()
MPI_WAITALL()

Thank you!

Branislav Jansík

IT4Innovations
national10£$01
supercomputing
center0!£0#010