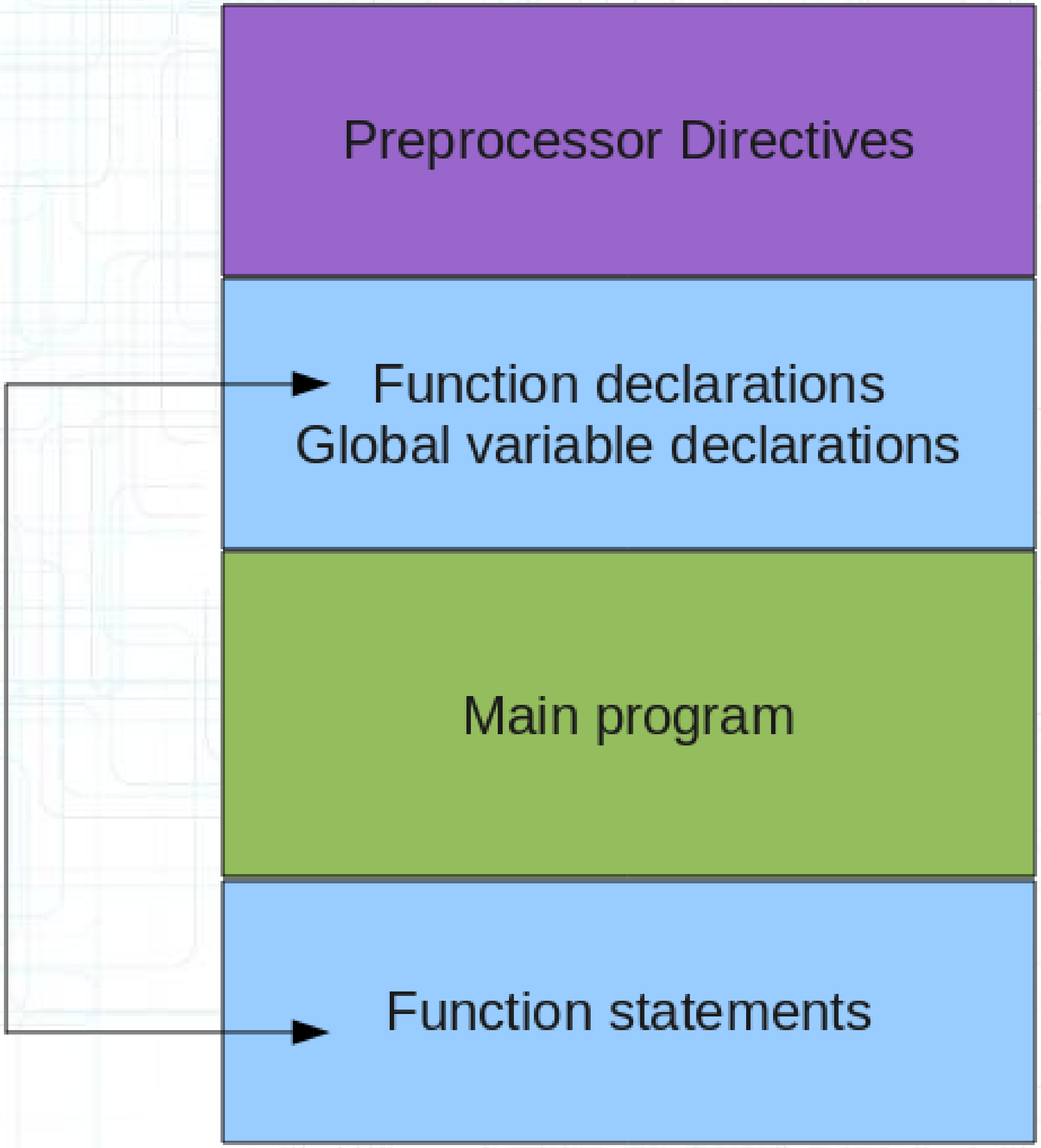
MPI programming with C

Structure of a C program revisited


Preprocessor Directives

Function declarations
Global variable declarations

Main program

Function statements

Modifications on a C source file to implement MPI

- 1 preprocessor directive.
- Function calls on the main program.
- Compile with new compiler command.
- Execute with run-time command.

How do MPI programs work?

- Basically a copy of the program is executed on every computer (node) in the network (cluster) where it is launched.
- They communicate with each other by sending messages.
- You specify on how many nodes you want it to run.
- Some constraints apply (execution time, number of nodes, no interactivity) when using research center clusters (TACC, Teragrid, etc.).

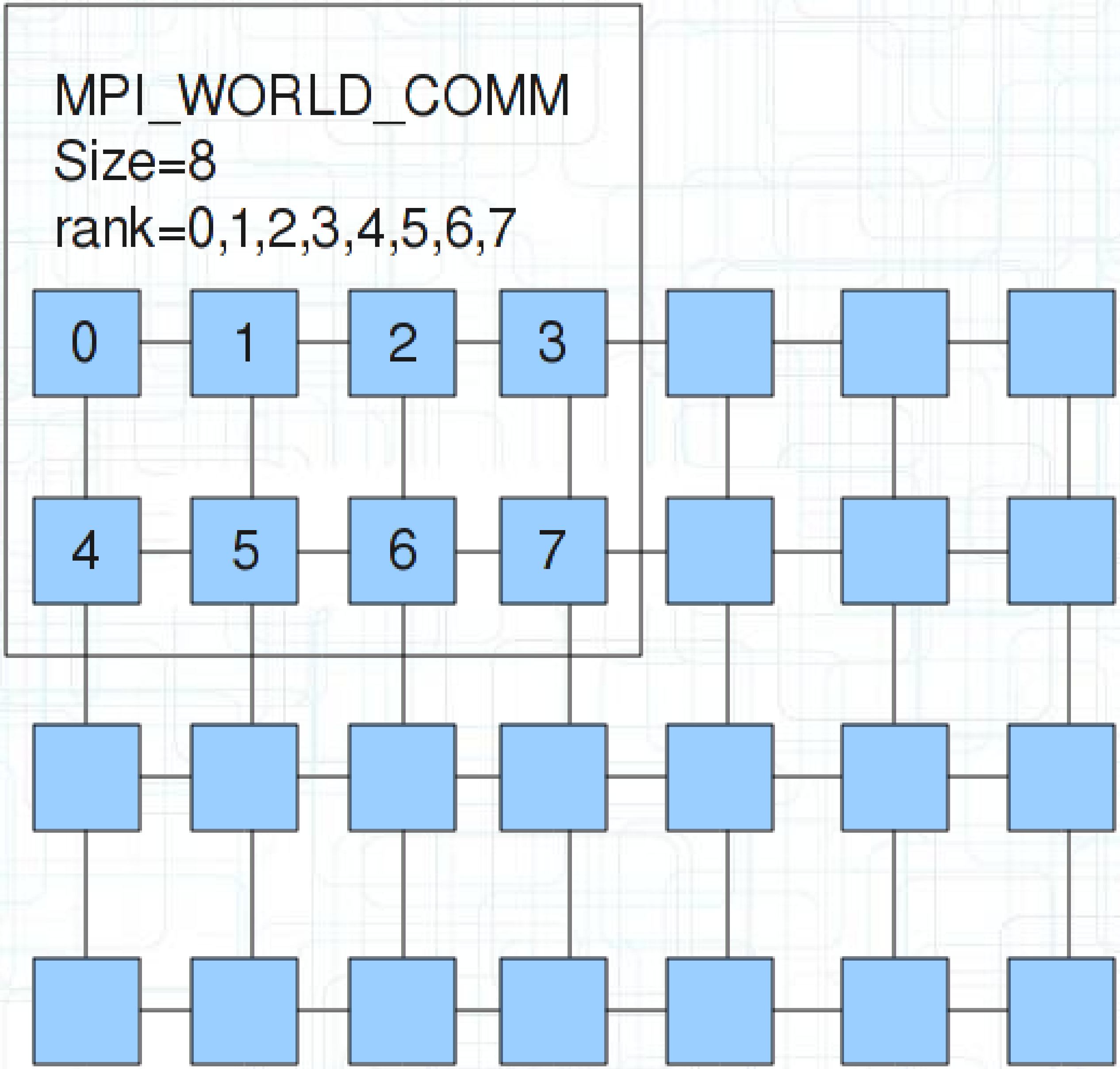
Master/Worker paradigm

- One node distributes work to others and receives results from them to prepare a final output.
- Serves as an intermediary with the user.
- Reads input and writes output.
- May or may not do work.

Communicator

- An execution “environment” within the cluster created when a MPI program is executed in it.
- Default name is MPI_COMM_WORLD
- Numbers nodes from *0* to *n-1* for *n* nodes that execute the program.
- Node 0 is usually the *master* or *root* node.

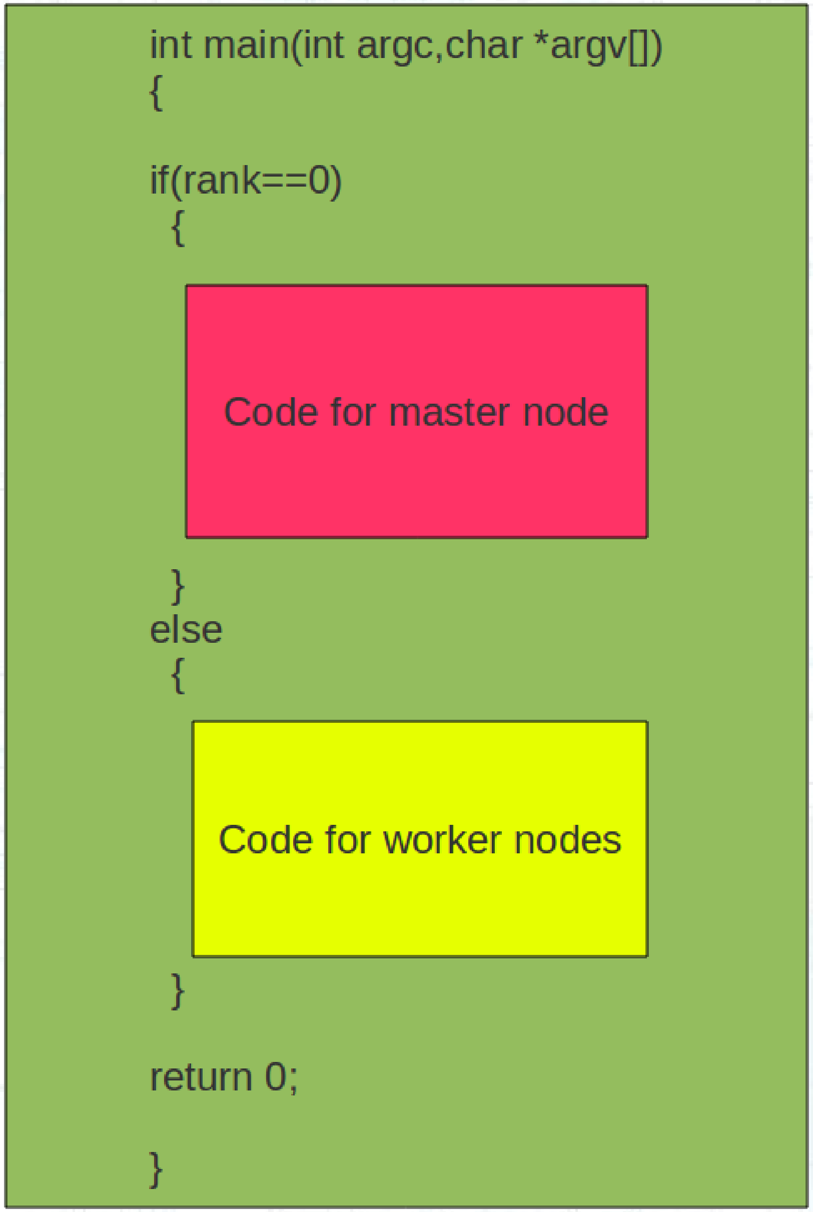
Example



MPI source files

- No need to write a separate program for every node.
- Just make a block of code that corresponds to either the master or worker nodes.
- We differentiate them through their *rank*.

Sample program



```
int main(int argc,char *argv[])
{

if(rank==0)
  {

    Code for master node

  }
else
  {

    Code for worker nodes

  }

return 0;

}
```

Basic Modifications

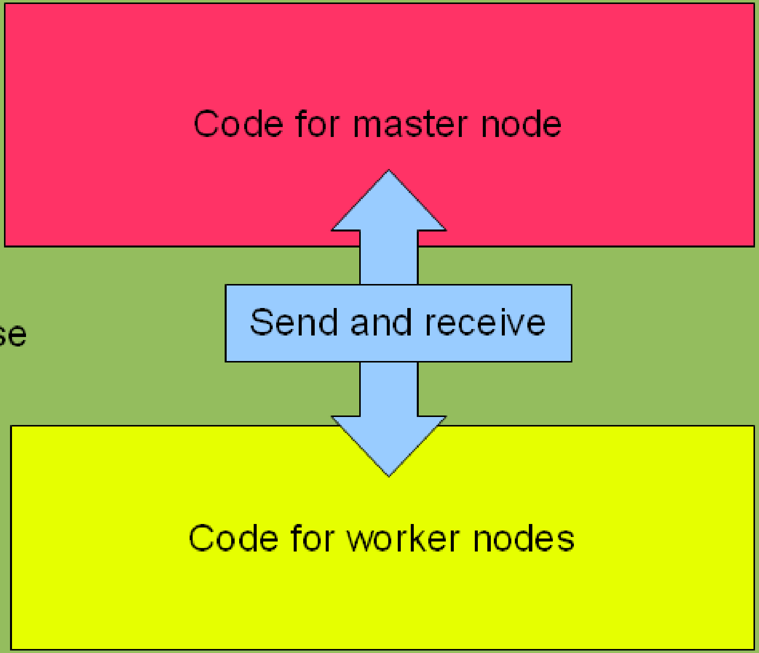
- #include <mpi.h>
- MPI_Init(&argc,&argv);
- MPI_Comm_rank(MPI_COMM_WORLD,&rank);
- MPI_Comm_size(MPI_COMM_WORLD,&size);
- MPI_Finalize();

```
#include <mpi.h>
int main(int argc,char *argv[])
{

int rank,size;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_WORLD_COMM,&size);
MPI_Comm_rank(MPI_WORLD_COMM,&rank);
if(rank==0)
  {

          Code for master node

  }
else
  {

          Code for worker nodes

  }

MPI_Finalize();
return 0;

}
```

Communication

- Root and worker nodes will communicate by interchanging messages between each other.
- Done with MPI_Send() and MPI_Recv() function calls.
- MPI_Send(&data_snd,number,MPI_Type, dest,tag,MPI_WORLD_COMM);
- MPI_Recv(&data_rcv,number,MPI_Type,source,tag,MPI_WORLD_COMM,&status);

```
#include <mpi.h>
int main(int argc,char *argv[])
{

int rank,size;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_WORLD_COMM,&size);
MPI_Comm_rank(MPI_WORLD_COMM,&rank);
if(rank==0)
  {
```



Code for master node

Send and receive

Code for worker nodes

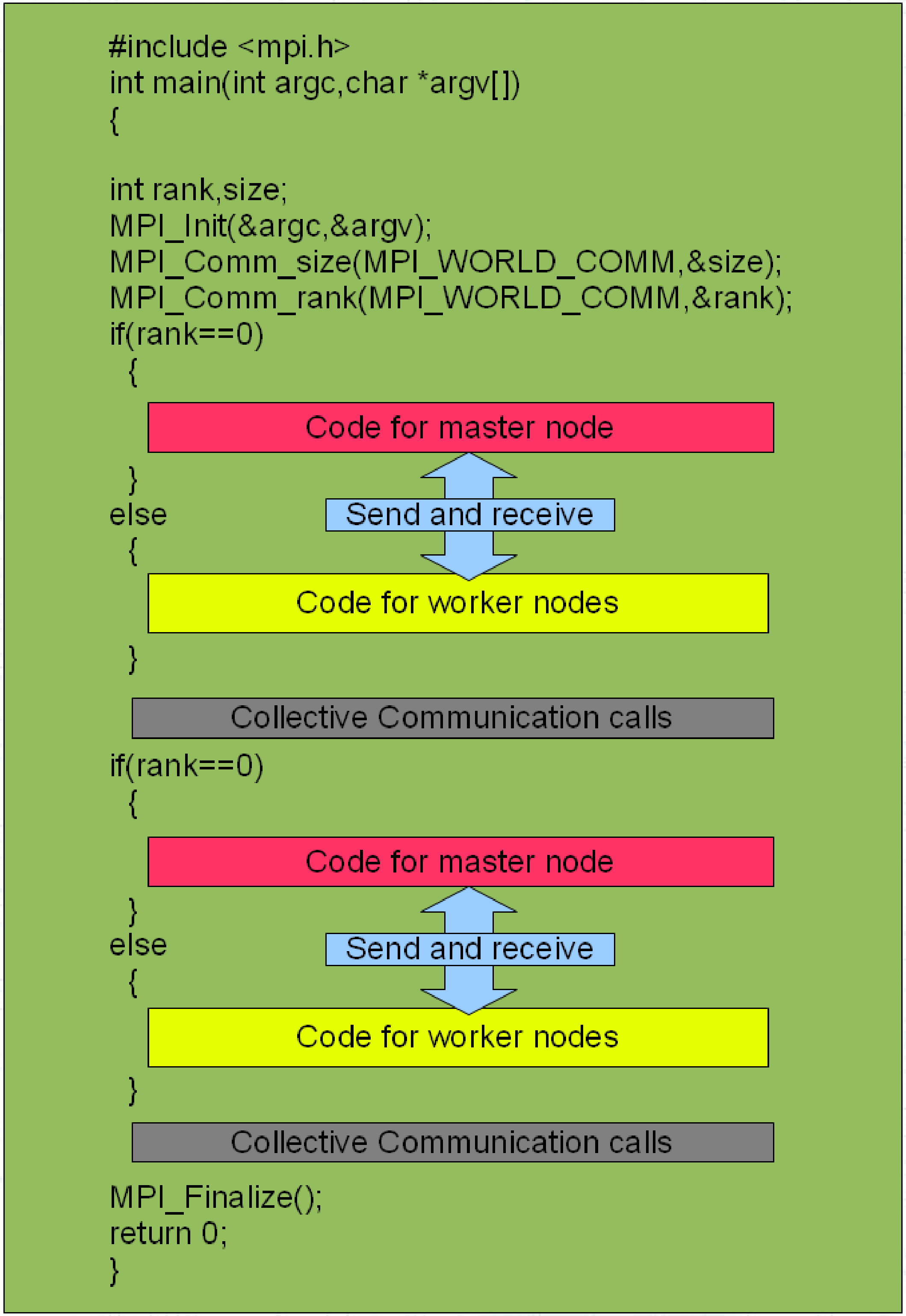
```
  }
else
  {

  }

MPI_Finalize();
return 0;

}
```

Collective communication

- MPI_Send() and MPI_Recv() are point-to-point communications, but can be looped to communicate with all nodes.
- There are MPI functions to send or receive messages to and from all nodes in the communicator.
- In order to work every node in the communicator must execute the function call.

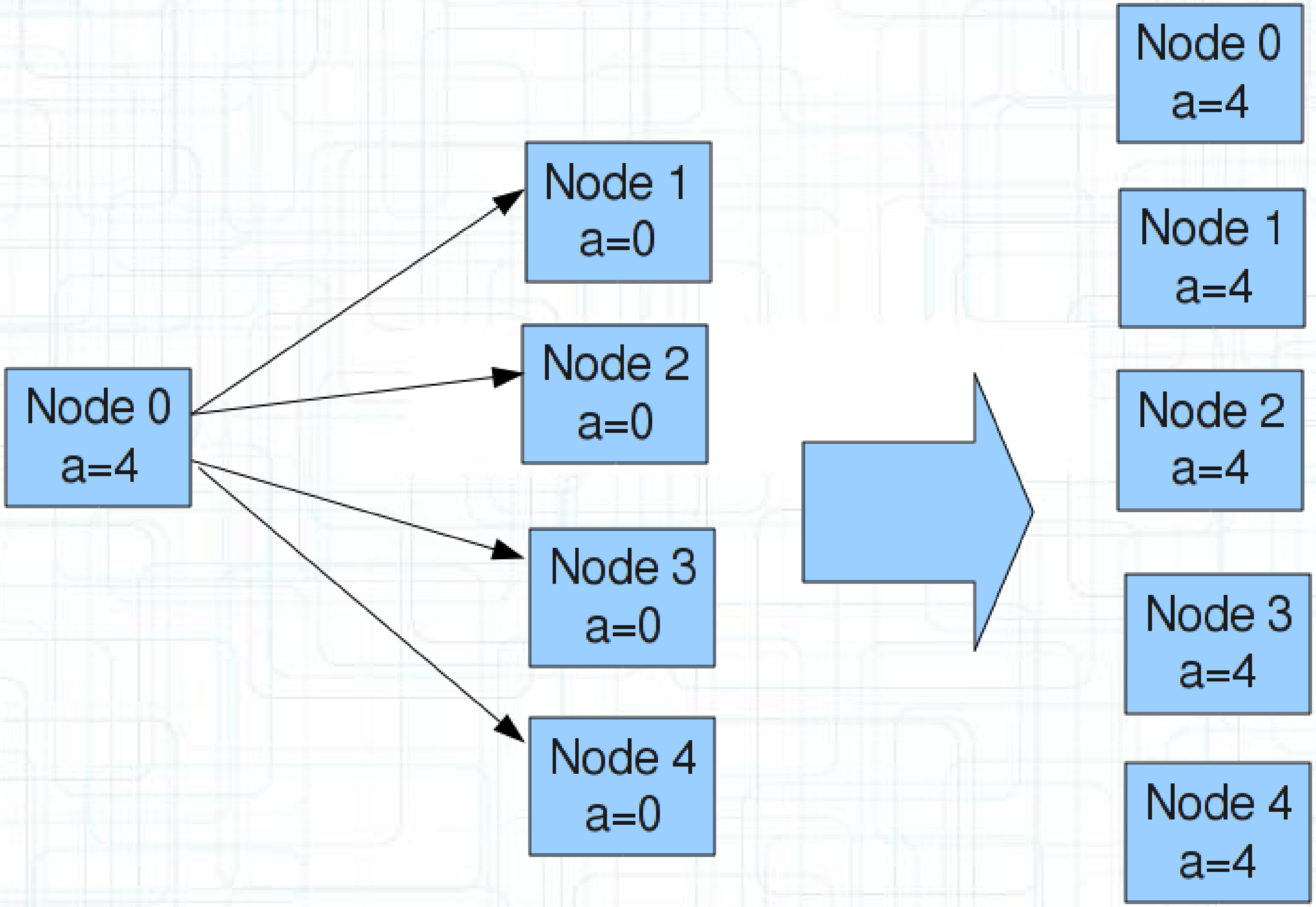
Collective Communication functions

- MPI_Bcast(); broadcast a message to all nodes in the communicator.
- MPI_Reduce(); get a message from every node in the communicator and do an operation on them.
- MPI_Scatter(); distribute an array to every node in the communicator.
- MPI_Gather(); fill an array with elements from every node in the communicator.
- MPI_Barrier(); set a synchronization barrier.

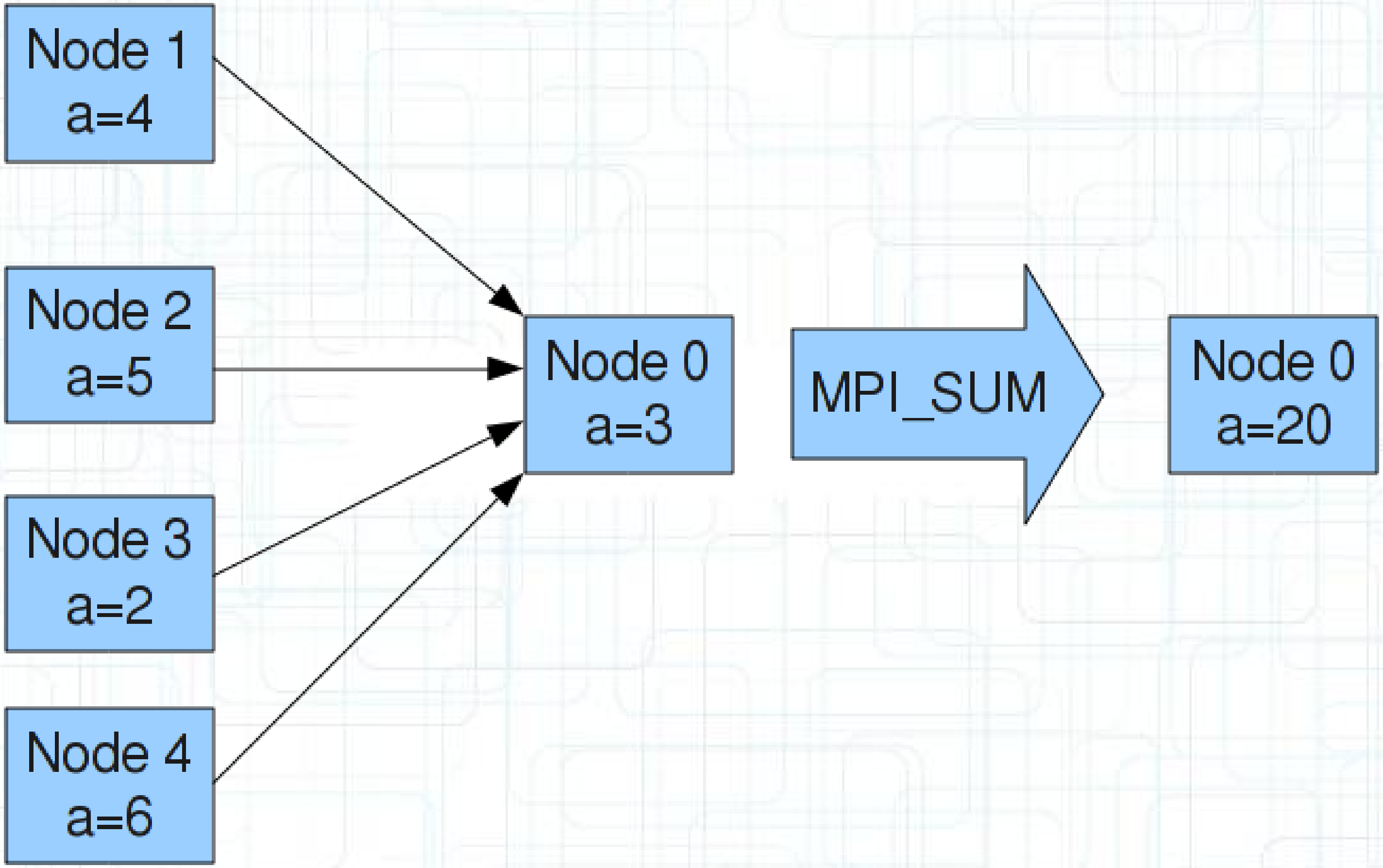


```
#include <mpi.h>
int main(int argc,char *argv[])
{

int rank,size;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_WORLD_COMM,&size);
MPI_Comm_rank(MPI_WORLD_COMM,&rank);
if(rank==0)
  {
    Code for master node
  }
else                Send and receive
  {
    Code for worker nodes
  }

  Collective Communication calls
if(rank==0)
  {
    Code for master node
  }
else                Send and receive
  {
    Code for worker nodes
  }

  Collective Communication calls
MPI_Finalize();
return 0;
}
```

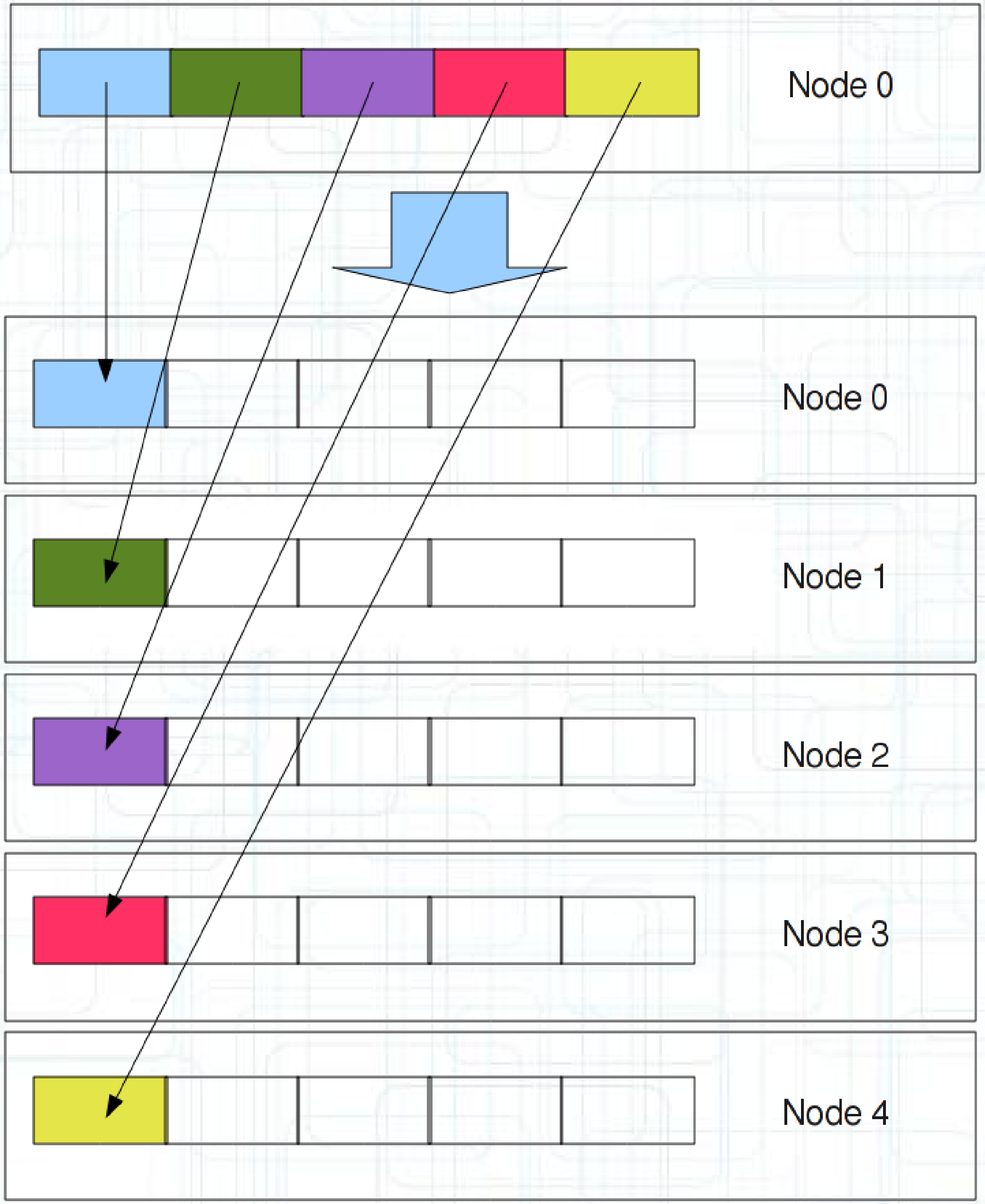
Bcast


Node 0
a=4
Node 1
a=0
Node 2
a=0
Node 3
a=0
Node 4
a=0
Node 0
a=4
Node 1
a=4
Node 2
a=4
Node 3
a=4
Node 4
a=4

MPI_Reduce



MPI_Scatter



Compilation

- for c (gcc myfile.c -o myfile)
- for MPI (mpicc mympifile.c -o mympifile)
- for c++(g++ myfile.cpp -o myfile)
- for MPI(mpi++ mympifile.cpp -o mympifile)

Execution

- mpirun -n # mympifile
- mpiexec -n # mympifile