The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

2068-16

Advanced School in High Performance and GRID Computing - Concepts and Applications

30 November - 11 December, 2009

Introduction to MPI

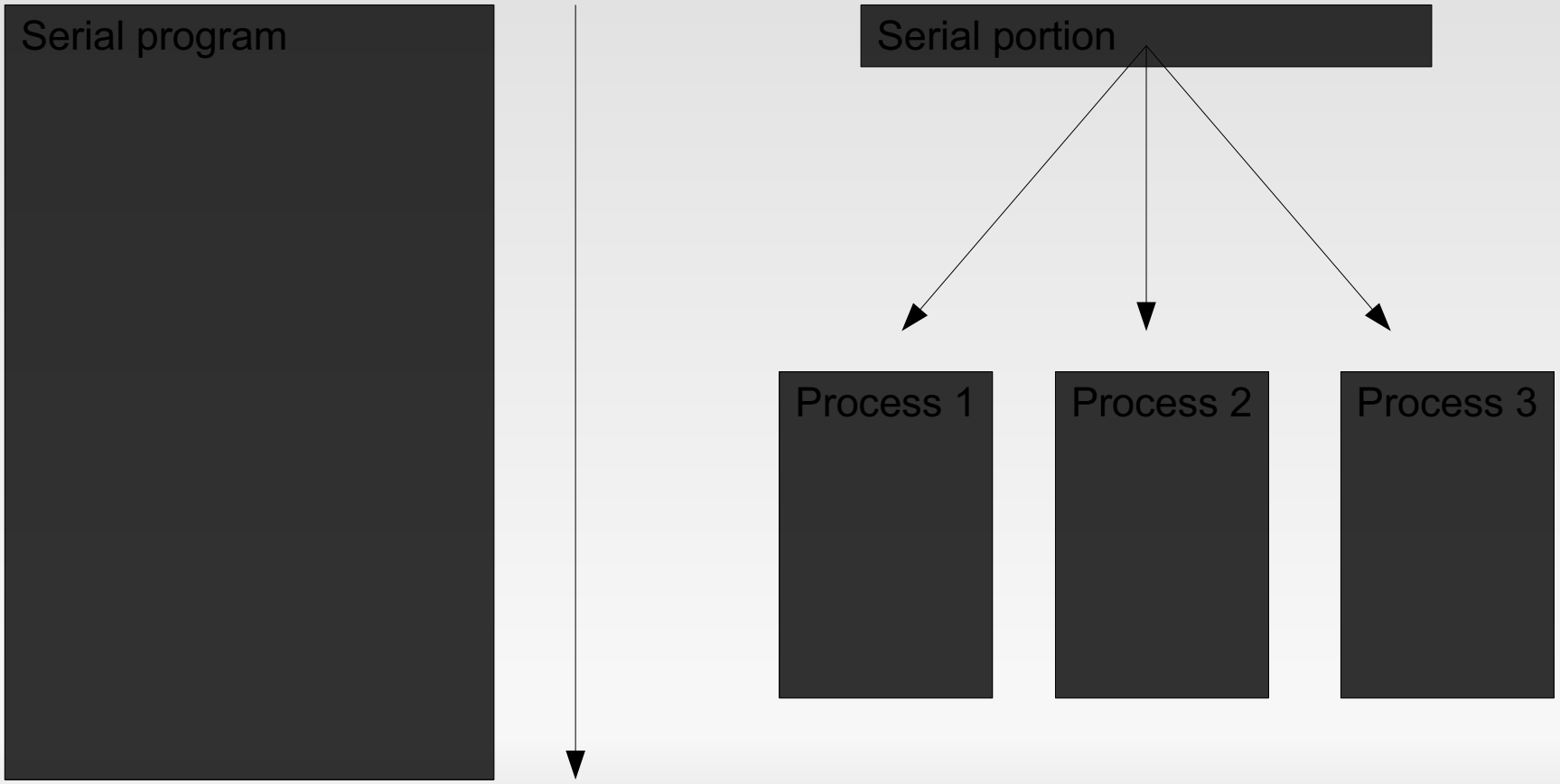
S.T. Brown
Carnegie Mellon University
Pittsburgh
USA

Introduction to MPI

Shawn T. Brown
Senior Scientific Specialist
Pittsburgh Supercomputing Center
stbrown@psc.edu

Programming in MPI

- Basic Idea

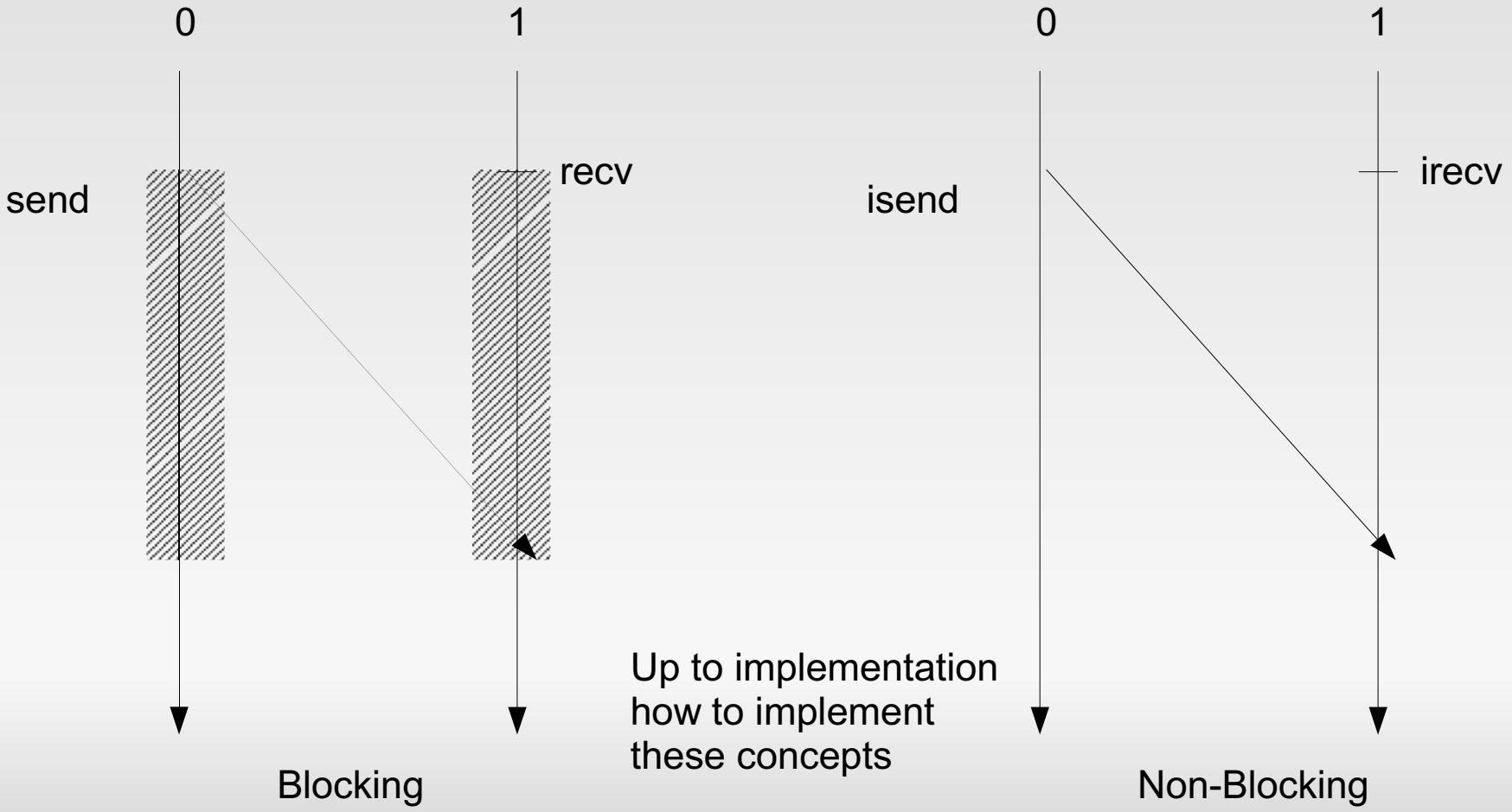


Programming in MPI

- Most MPI implementations
  - Provide a library to link into your code
  - C/C++ and FORTRAN interfaces
- Basically four types of MPI functions
  - Initialization and managment calls
  - Point-to-Point communication
    - Communication between 2 processes
  - Collective operations
    - Communication between groups of processes
  - Data type creation

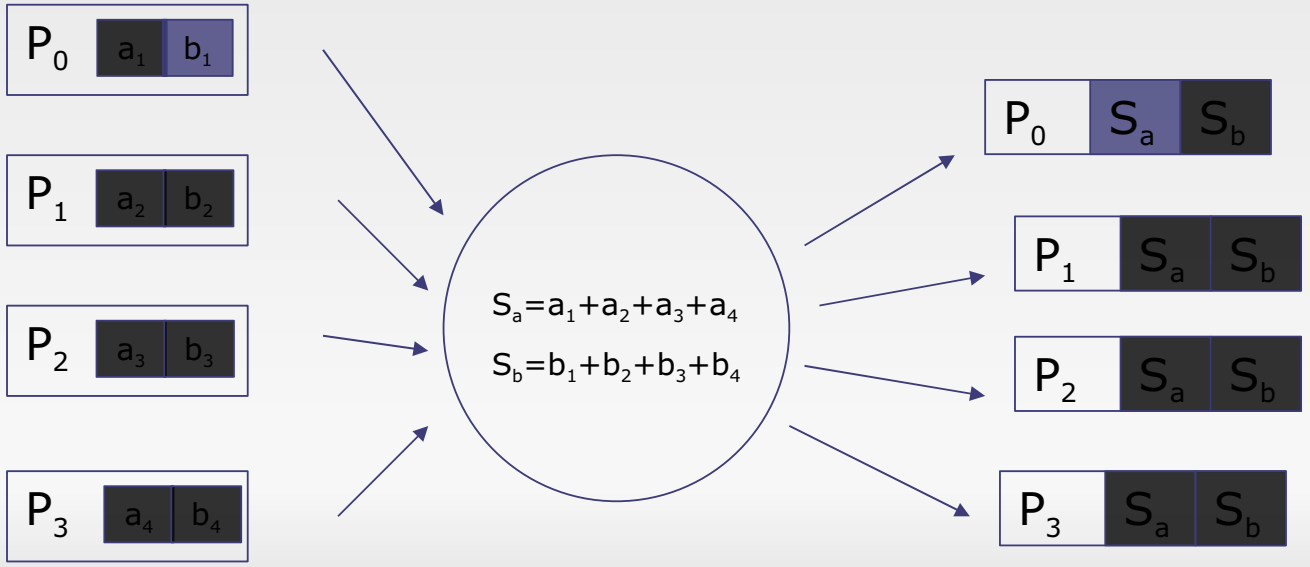
Point-to-Point communication

- In MPI standard, point-to-point communication is facilitated by sends and recieves



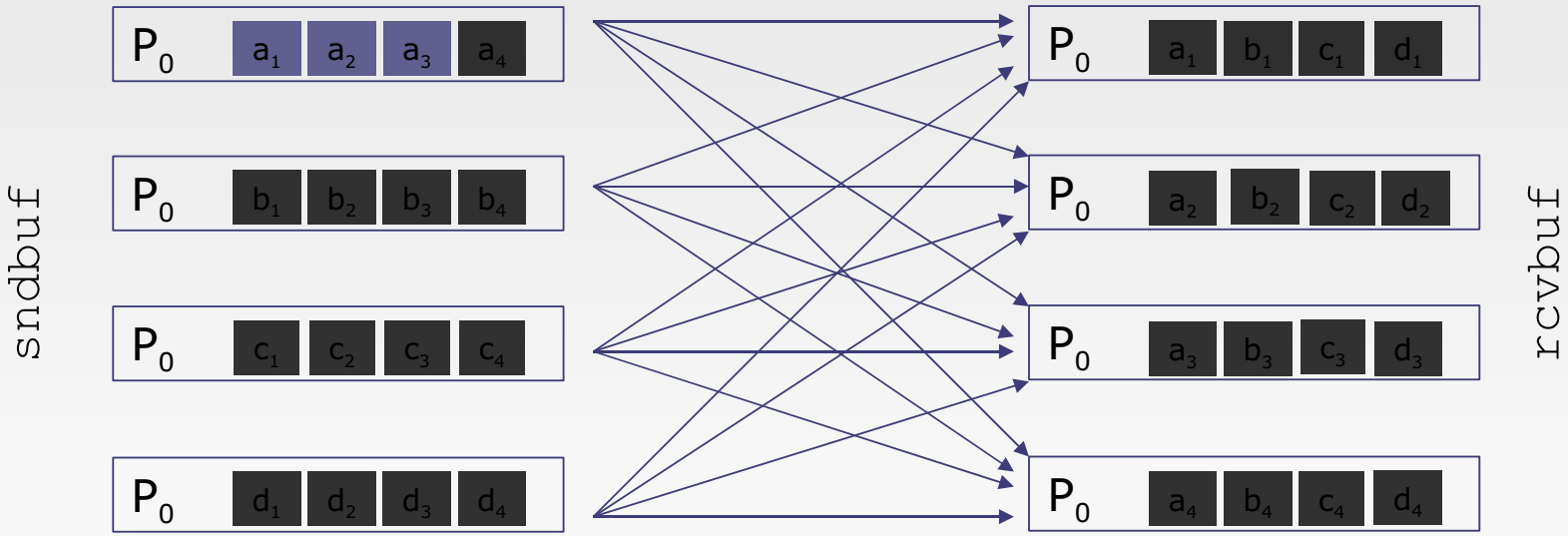
Collective communication

- Reduction
  - Reduce disparate arrays on a group of processors by some defined operations
    - e.g. summation



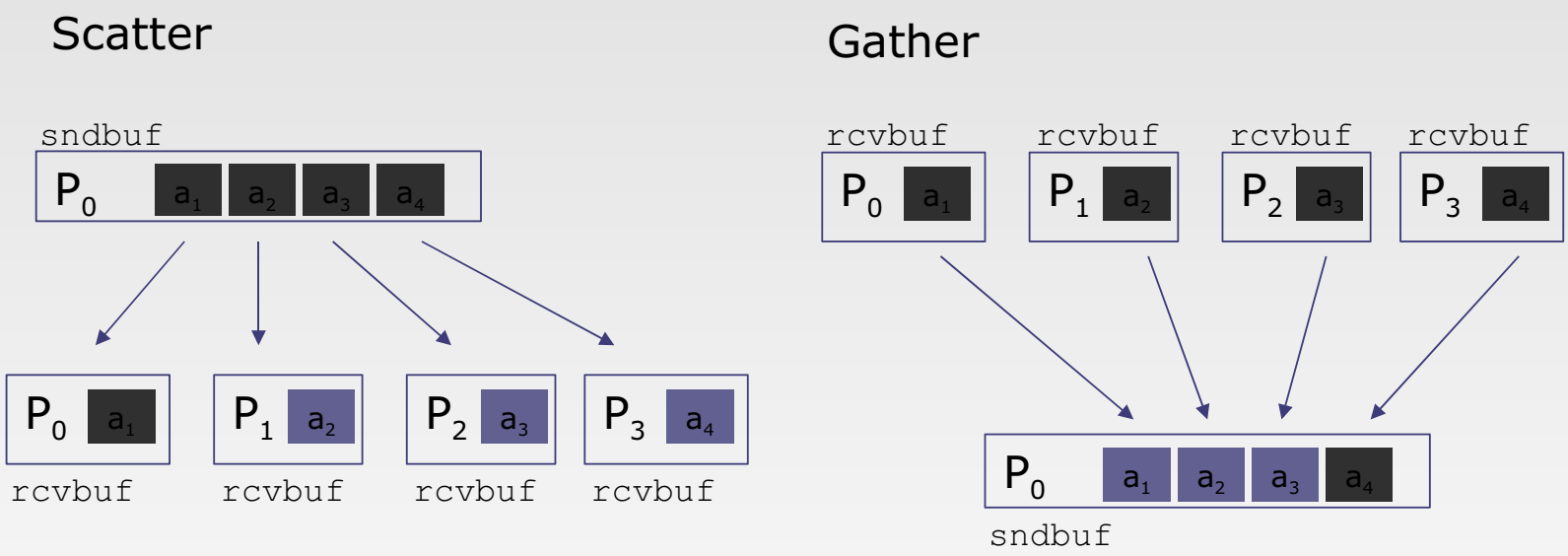
Collective communications

- All To All operations
  - Broadcasting



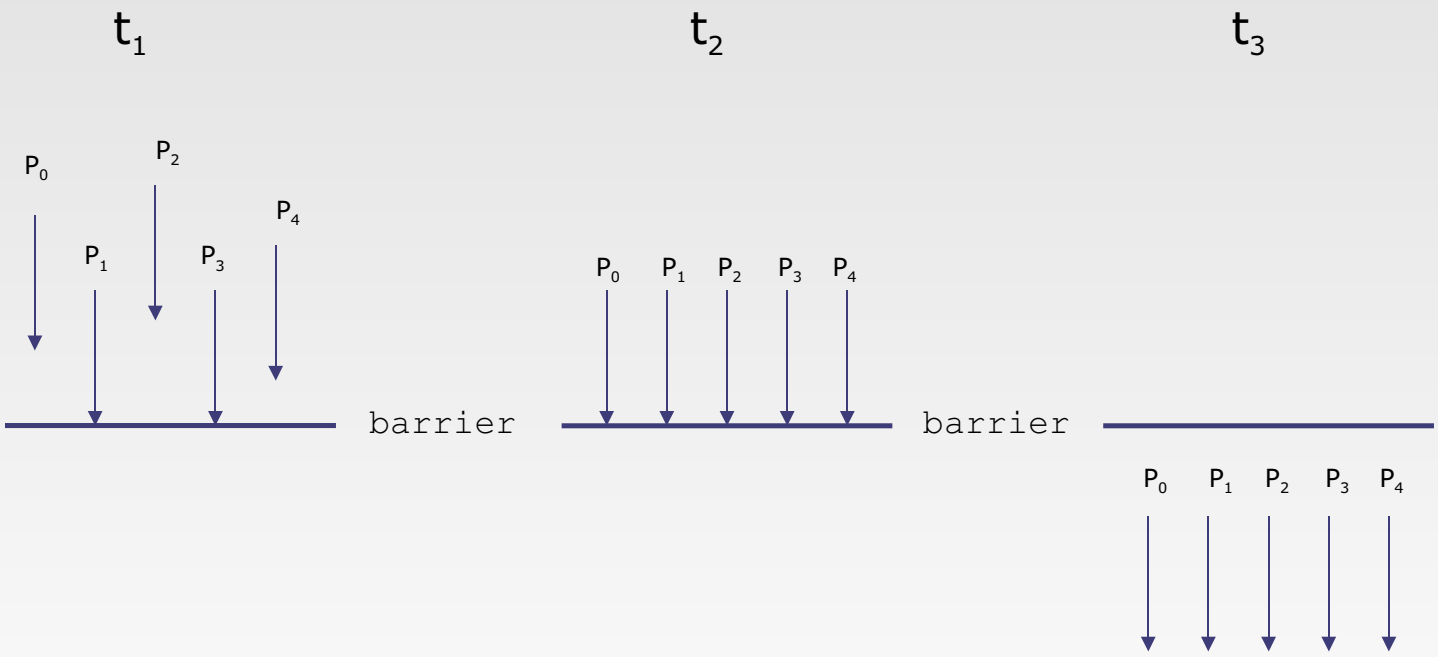
Collective Communications

- Gather-Scatter (One-to-All)



Synchronization

- Barriers
  - Define an explicit barrier to hold an execution point.



MPI Nuts and Bolts

- My first MPI program

```
#include <stdio.h>

#include <mpi.h>

void 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, size );

  MPI_Finalize();

  return 0;

}
```

MPI Nuts and Bolts

- Header File
  - Need to include to have MPI functions defined

```
#include <stdio.h>

#include <mpi.h>

void 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, size );

  MPI_Finalize();

  return 0;

}
```

MPI Nuts and Bolts

- Initialization
  - Connects the program to the processes created by the mpirun script
  - Creates the global communicator data structure MPI_COMM_WORLD

```
#include <stdio.h>

#include <mpi.h>

void 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, size );

  MPI_Finalize();

  return 0;

}
```

Communicators

- A communicator is a datastructure that defines a group of processes
  - Each communicator has:
    - Size: the number of processes associated with it
    - Ranks: indexing to define the processes within the communicator
- MPI_COMM_WORLD
  - Created for every MPI program
  - The communicator that defines all of the processes in the current program.
  - Created by MPI_Init

MPI Nuts and Bolts

- These two functions can be used to access the rank and size for a communicator.

```
#include <stdio.h>

#include <mpi.h>

void 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, size );

  MPI_Finalize();

  return 0;

}
```

MPI Nuts and Bolts

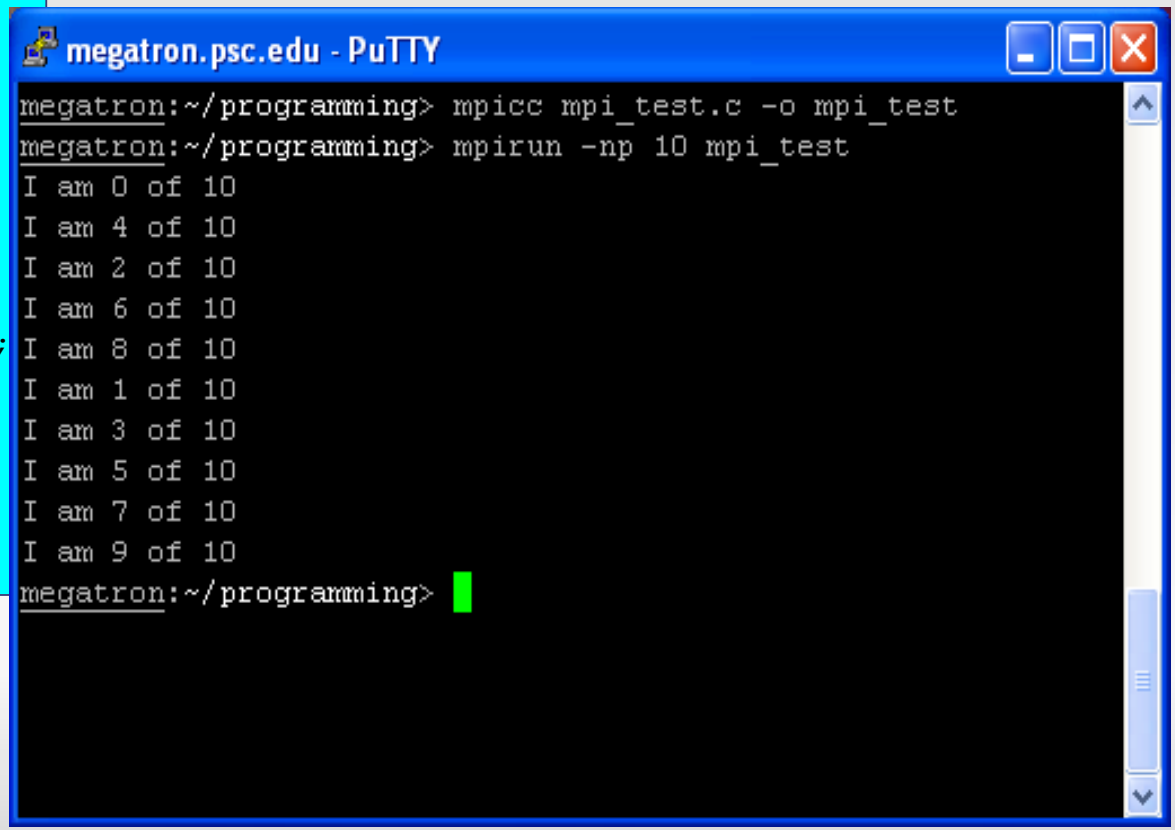
- Finalize
  - Cleans up and detaches all data from the processes.

```
#include <stdio.h>

#include <mpi.h>

void 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, size );

  MPI_Finalize();

  return 0;

}
```

Example

```
#include <stdio.h>

#include <mpi.h>

void 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, size );

  MPI_Finalize();

  return 0;

}
```

Notice that the printout is not in order!



Ok, here it is in Fortran

```
PROGRAM hello
 INCLUDE `mpif.h`
 INTEGER err
 CALL MPI_INIT(err)
 call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
 call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
 print *, 'I am ', rank, ' of ', size
 CALL MPI_FINALIZE(err)
END
```

How much MPI do you need?

- One can survive knowing only 6 MPI calls.

  MPI_INIT()
  MPI_COMM_SIZE()
  MPI_COMM_RANK()
  MPI_SEND
  MPI_RECV
  MPI_FINALIZE

Example

- Let's write a program to send one integer from process 0 to process 1.

```
Program MPI
   Implicit None
!
   Include 'mpif.h'
!
   Integer                                    :: rank
   Integer                                    :: buffer
   Integer, Dimension( 1:MPI_status_size )    :: status
   Integer                                    :: error
!
   Call MPI_init( error )
   Call MPI_comm_rank( MPI_comm_world, rank, error )
!
   If( rank == 0 ) Then
      buffer = 33
      Call MPI_send( buffer, 1, MPI_integer, 1, 10, &
                     MPI_comm_world, error )
   End If
!
   If( rank == 1 ) Then
      Call MPI_recv( buffer, 1, MPI_integer, 0, 10, &
                     MPI_comm_world, status, error )
      Print*, 'Rank ', rank, ' buffer=', buffer
      If( buffer /= 33 ) Print*, 'fail'
   End If
   Call MPI_finalize( error )
End Program MPI
```

Anatomy of a MPI call

The simplest call:

MPI_send(buffer, count, data_type, destination,tag, communicator**)**

where:

BUFFER: data to send

COUNT: number of elements in buffer .

DATA_TYPE : which kind of data types in buffer ?

DESTINATION the receiver

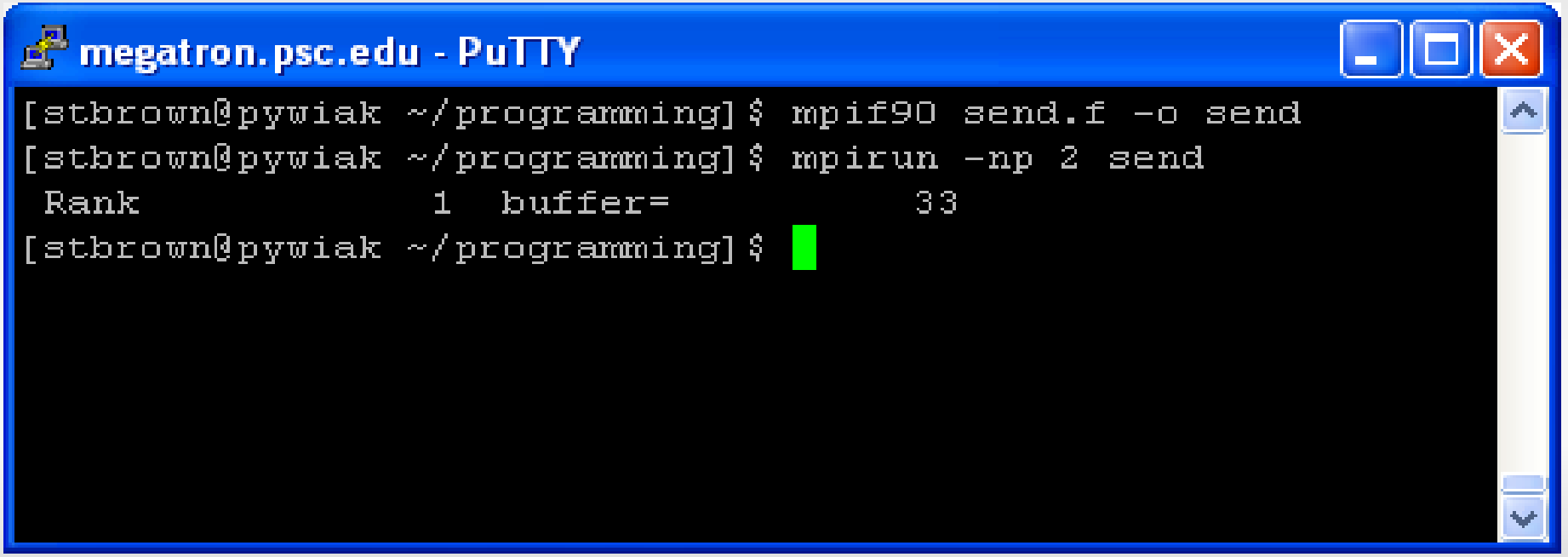
TAG: the label of the message

COMMUNICATOR set of processors involved

Example

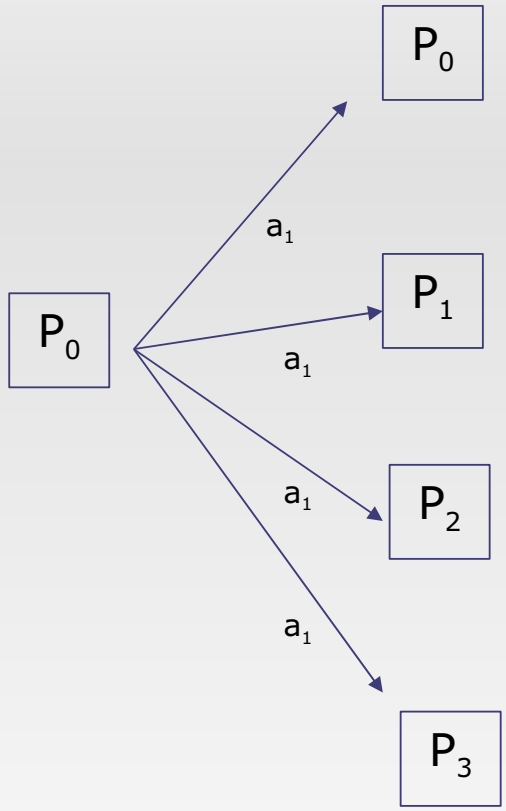
- Let's write a program to send one integer from process 0 to process 1.

```
Program MPI
   Implicit None
!
   Include 'mpif.h'
!
   Integer                                    :: rank
   Integer                                    :: buffer
   Integer, Dimension( 1:MPI_status_size )    :: status
   Integer                                    :: error
```



Broadcast Example

```
PROGRAM broad_cast
 INCLUDE 'mpif.h'
 INTEGER ierr, myid, nproc, root
 INTEGER status(MPI_STATUS_SIZE)
 REAL A(2)
 CALL MPI_INIT(ierr)
 CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
 CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
 root = 0
 IF( myid .EQ. 0 ) THEN
   a(1) = 2.0
   a(2) = 4.0
 END IF
 CALL MPI_BCAST(a, 2, MPI_REAL, 0, MPI_COMM_WORLD, ierr)
 WRITE(6,*) myid, ': a(1)=', a(1), 'a(2)=', a(2)
 CALL MPI_FINALIZE(ierr)
 END
```



MPI_BCast

One-to-all communication: same data sent from root process to all others in the communicator

Fortran:

```
INTEGER count, type, root, comm, ierr

CALL MPI_BCAST(buf, count, type, root, comm, ierr)
```

Buf array of type **type**

C:

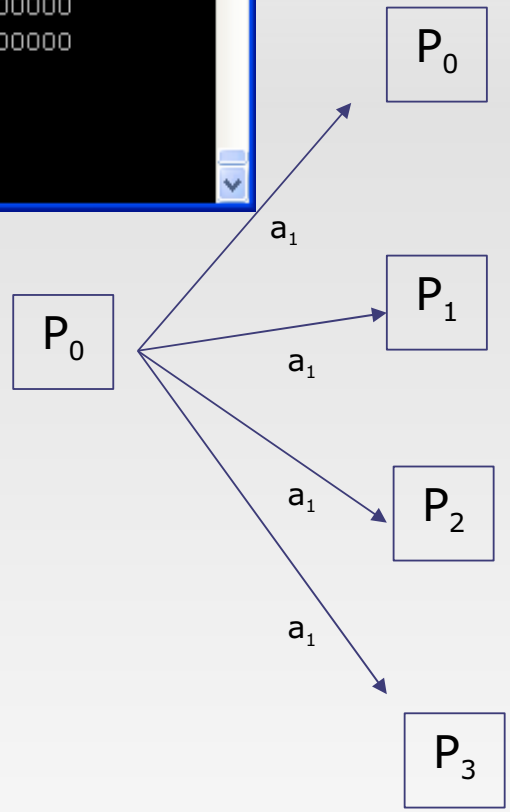
```
int MPI_Bcast(void *buf, int count, MPI_Datatype
  datatypem int root, MPI_Comm comm)
```

All processes must specify same **root**, **rank** and **comm**

Broadcast Example

```
megatron.psc.edu - PuTTY
[stbrown@pywiak ~/programming]$ mpif90 bcast.f -o bcast
[stbrown@pywiak ~/programming]$ mpirun -np 4 bcast
           0 : a(1)=   2.000000     a(2)=   4.000000
           1 : a(1)=   2.000000     a(2)=   4.000000
           2 : a(1)=   2.000000     a(2)=   4.000000
           3 : a(1)=   2.000000     a(2)=   4.000000
[stbrown@pywiak ~/programming]$
```

```
PROGRAM broad_cast
 INCLUDE 'mpif.h'
 INTEGER ierr, myid, nproc, root
 INTEGER status(MPI_STATUS_SIZE)
 REAL A(2)
 CALL MPI_INIT(ierr)
 CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
 CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
 root = 0
 IF( myid .EQ. 0 ) THEN
   a(1) = 2.0
   a(2) = 4.0
 END IF
 CALL MPI_BCAST(a, 2, MPI_REAL, 0, MPI_COMM_WORLD, ierr)
 WRITE(6,*) myid, ': a(1)=', a(1), 'a(2)=', a(2)
 CALL MPI_FINALIZE(ierr)
 END
```



Reduction Example

```
 PROGRAM reduce
 INCLUDE 'mpif.h'
 INTEGER ierr, myid, nproc, root
 INTEGER status(MPI_STATUS_SIZE)
 REAL A(2), res(2)
 CALL MPI_INIT(ierr)
 CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
 CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
 root = 0
 a(1) = 2.0
 a(2) = 4.0
 CALL MPI_REDUCE(a, res, 2, MPI_REAL, MPI_SUM, root,
&  MPI_COMM_WORLD, ierr)
 IF( myid .EQ. 0 ) THEN
   WRITE(6,*) myid, ': res(1)=', res(1), 'res(2)=', res(2)
 END IF
 CALL MPI_FINALIZE(ierr)
 END
```

Reduction Calls

Fortran:

`MPI_REDUCE(snd_buf,rcv_buf,count,type,op,root,comm,ierr)`

`snd_buf` input array of type `type` containing local values.

`rcv_buf` output array of type `type` containing global results

`count` `(INTEGER)` number of element of `snd_buf` and `rcv_buf`

`type` `(INTEGER)` MPI type of `snd_buf` and `rcv_buf`

`op` `(INTEGER)` parallel operation to be performed

`root` `(INTEGER)` MPI id of the process storing the result

`comm` `(INTEGER)` communicator of processes involved in the operation

`ierr` `(INTEGER)` output, error code (if `ierr=0` no error occours)

`MPI_ALLREDUCE( snd_buf,rcv_buf,count,type,op,comm,ierr)`

The argument `root` is missing, the result is stored to all processes.

In C

C:

```
int MPI_Reduce(void * snd_buf, void * rcv_buf, int count,
   MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm)
```

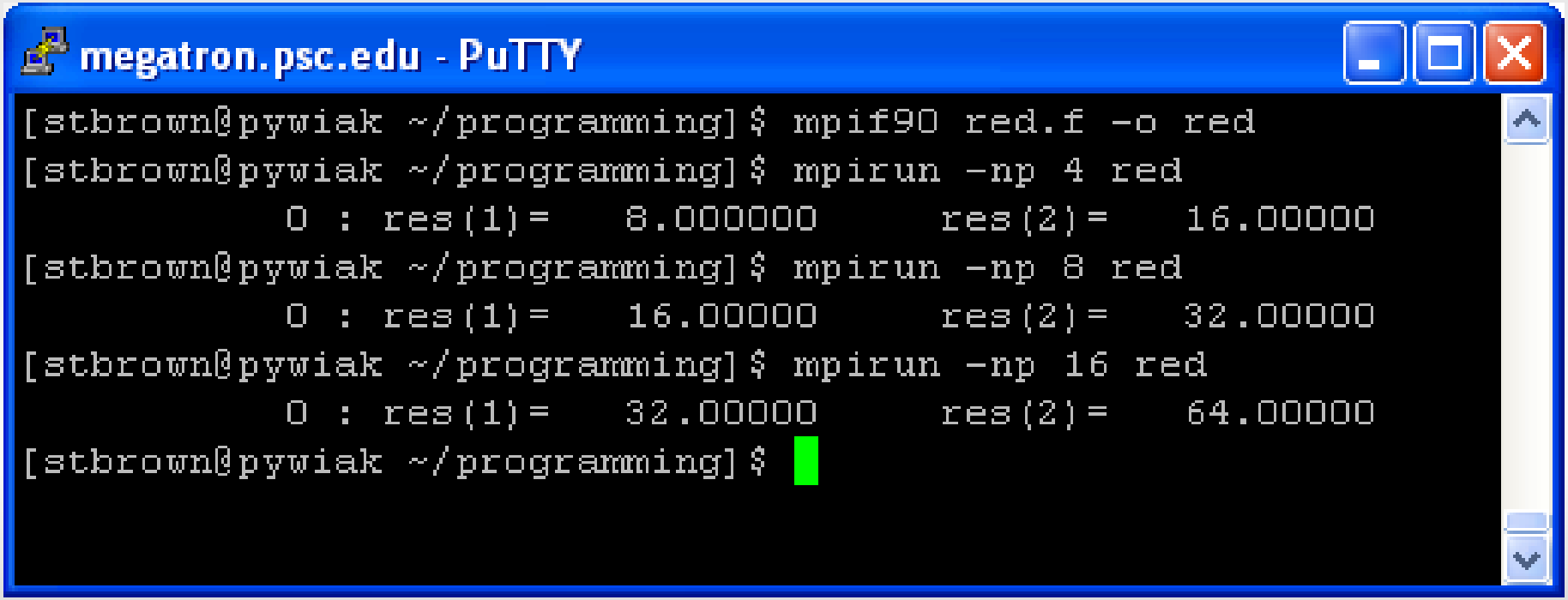
```
int MPI_Allreduce(void * snd_buf, void * rcv_buf, int count,
   MPI_Datatype type, MPI_Op op, MPI_Comm comm)
```

Predefined Reductions

MPI op	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

Reduction Example

```
PROGRAM reduce
INCLUDE 'mpif.h'
INTEGER ierr, myid, nproc, root
INTEGER status(MPI_STATUS_SIZE)
REAL A(2), res(2)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
```



Good resources ...

- http://webct.ncsa.uiuc.edu:8900/public/MPI/
  - Online MPI lecture and tutorial at NCSA.
- http://www-unix.mcs.anl.gov/mpi/usingmpi/examples/main.htm
  - Examples from the Using MPI book
- http://www.lam-mpi.org/tutorials/
  - A collection of links to more MPI tutorials