Master in High Performance Computing

PARALLEL PROGRAMMING

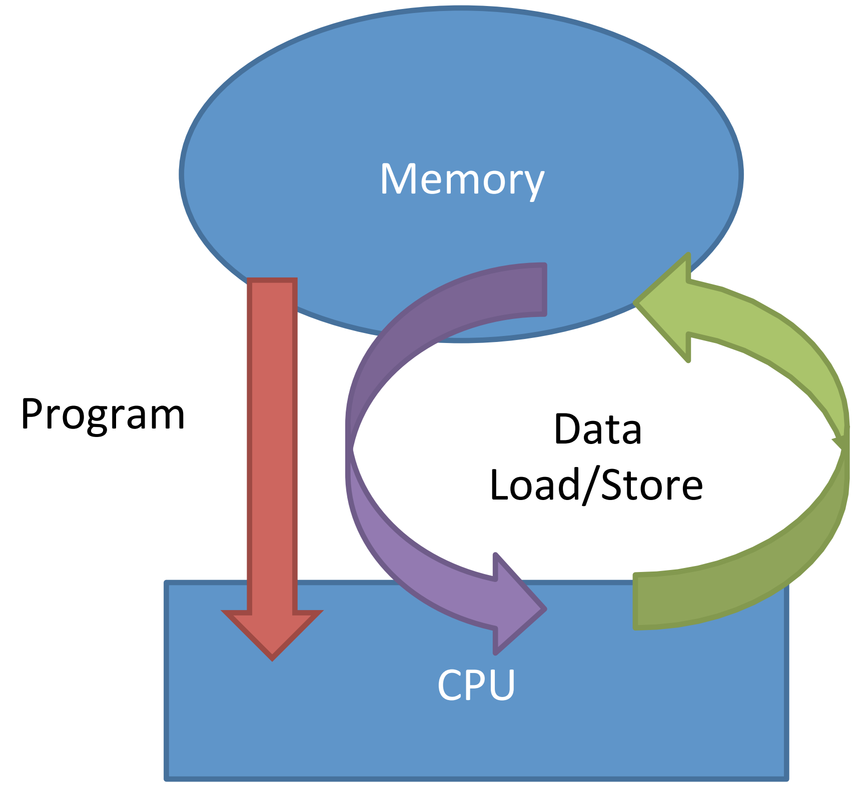
Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

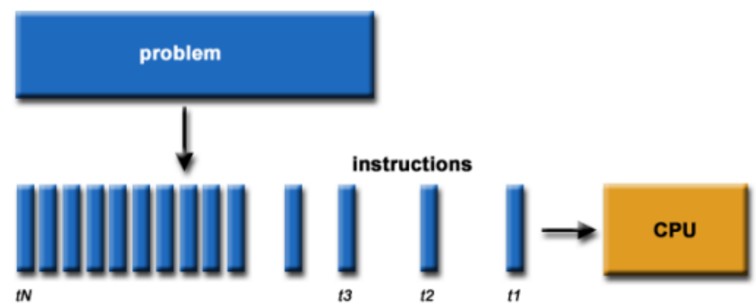
Why Parallel Programming

- Solve larger problems
- Run memory demanding codes
- Solve problems with greater speed

Serial Programming

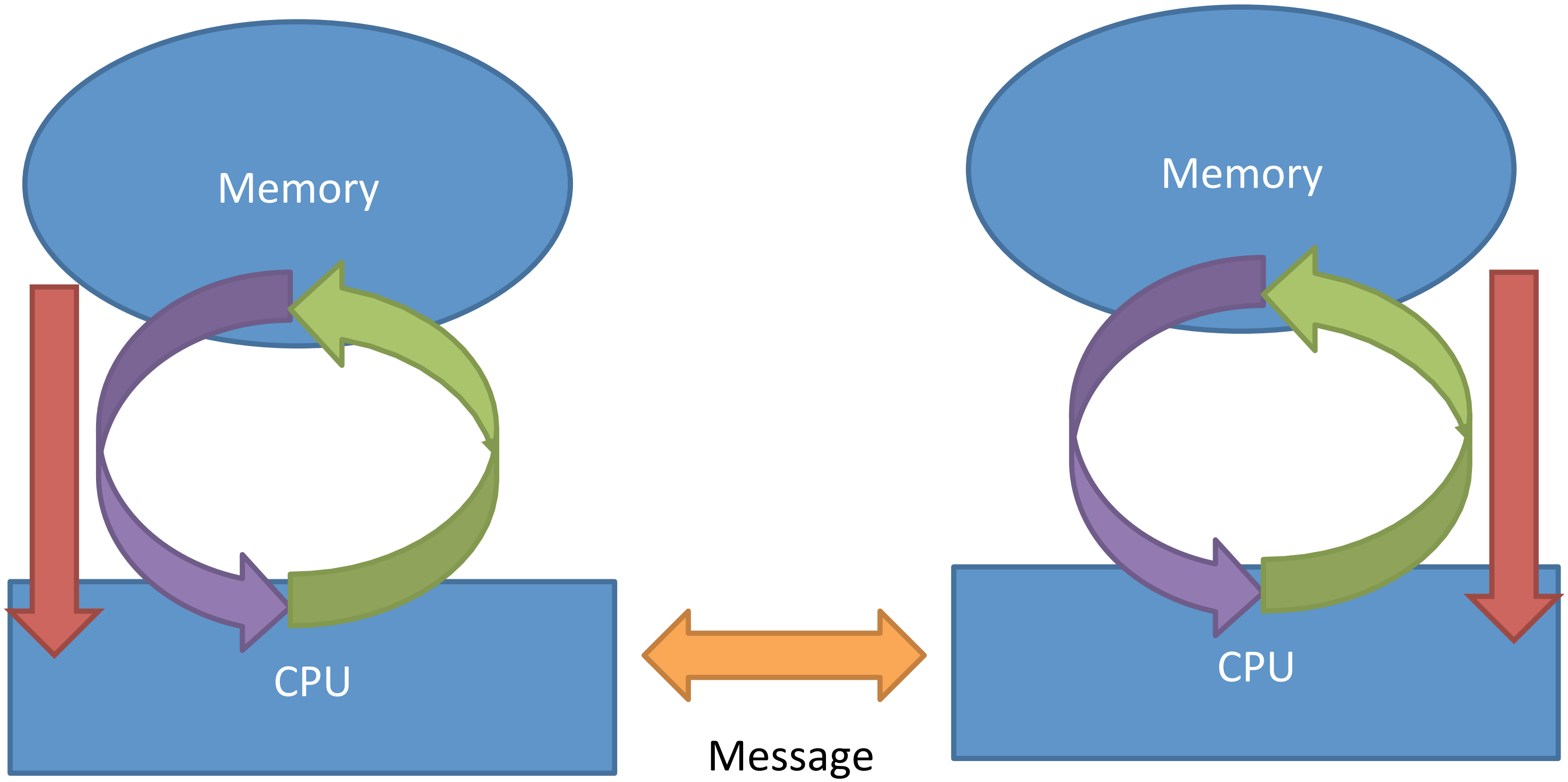


A problem is broken into a discrete series of instructions.
Instructions are executed one after another.
Only one instruction may execute at any moment in time.



Ivan Girotto
igirotto@ictp.it

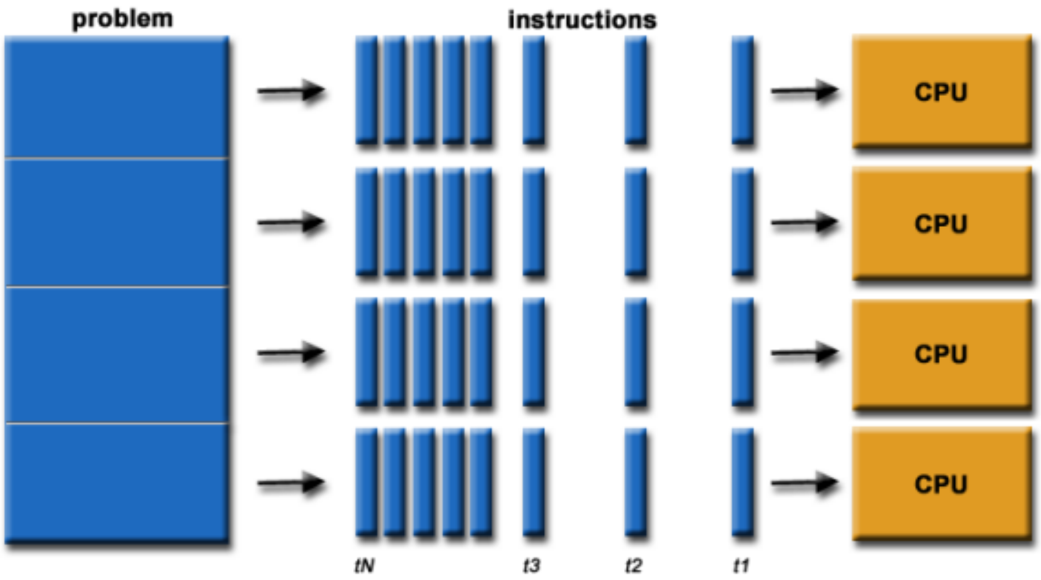
Parallel Programming



Ivan Girotto
igirotto@ictp.it

Concurrency

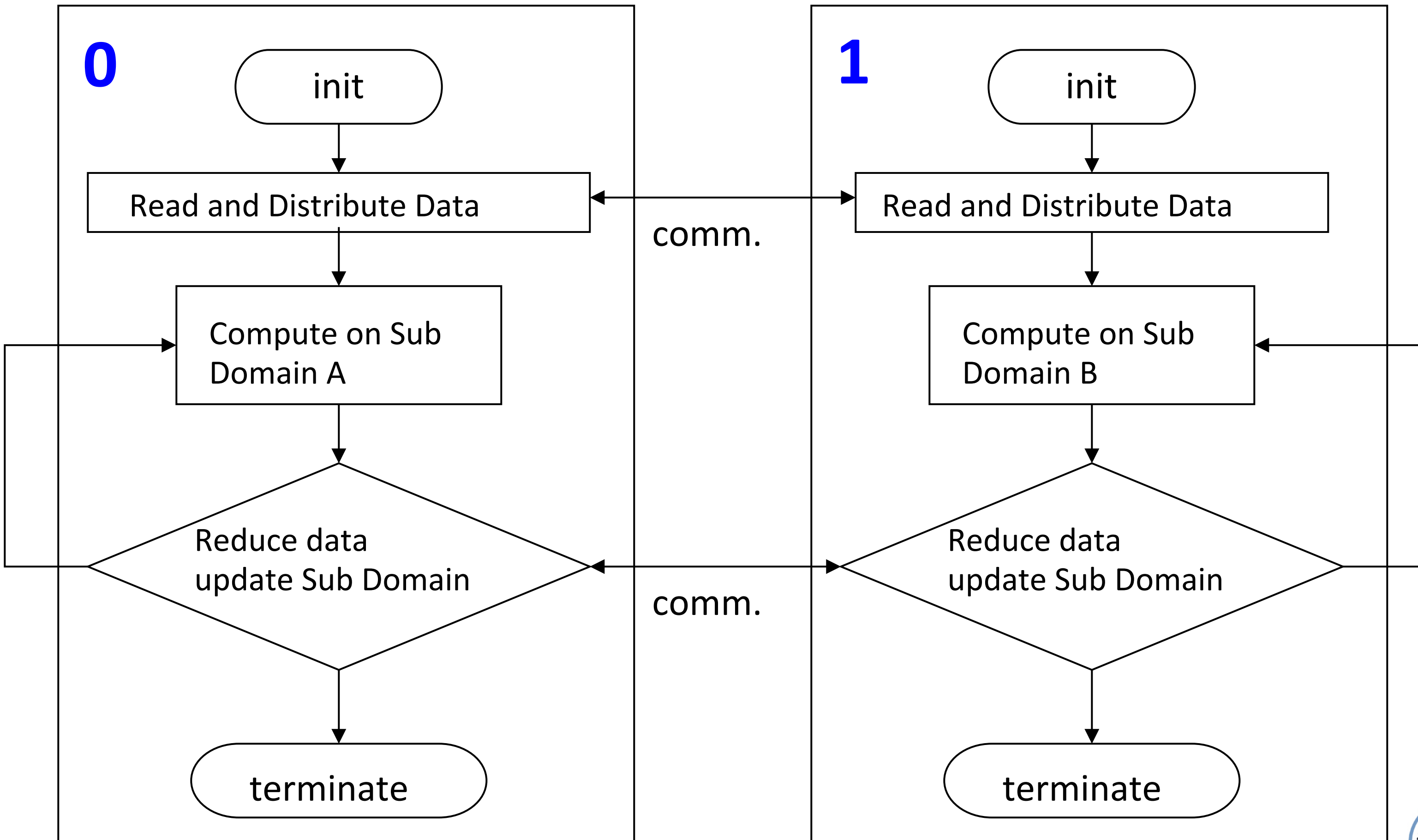
The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently



A problem is broken into discrete parts that can be solved concurrently
Each part is further broken down to a series of instructions
Instructions from each part execute simultaneously on different processors
An overall control / coordination mechanism is employed

Master in High Performance Computing

What is a Parallel Program



Modern Parallel Architectures

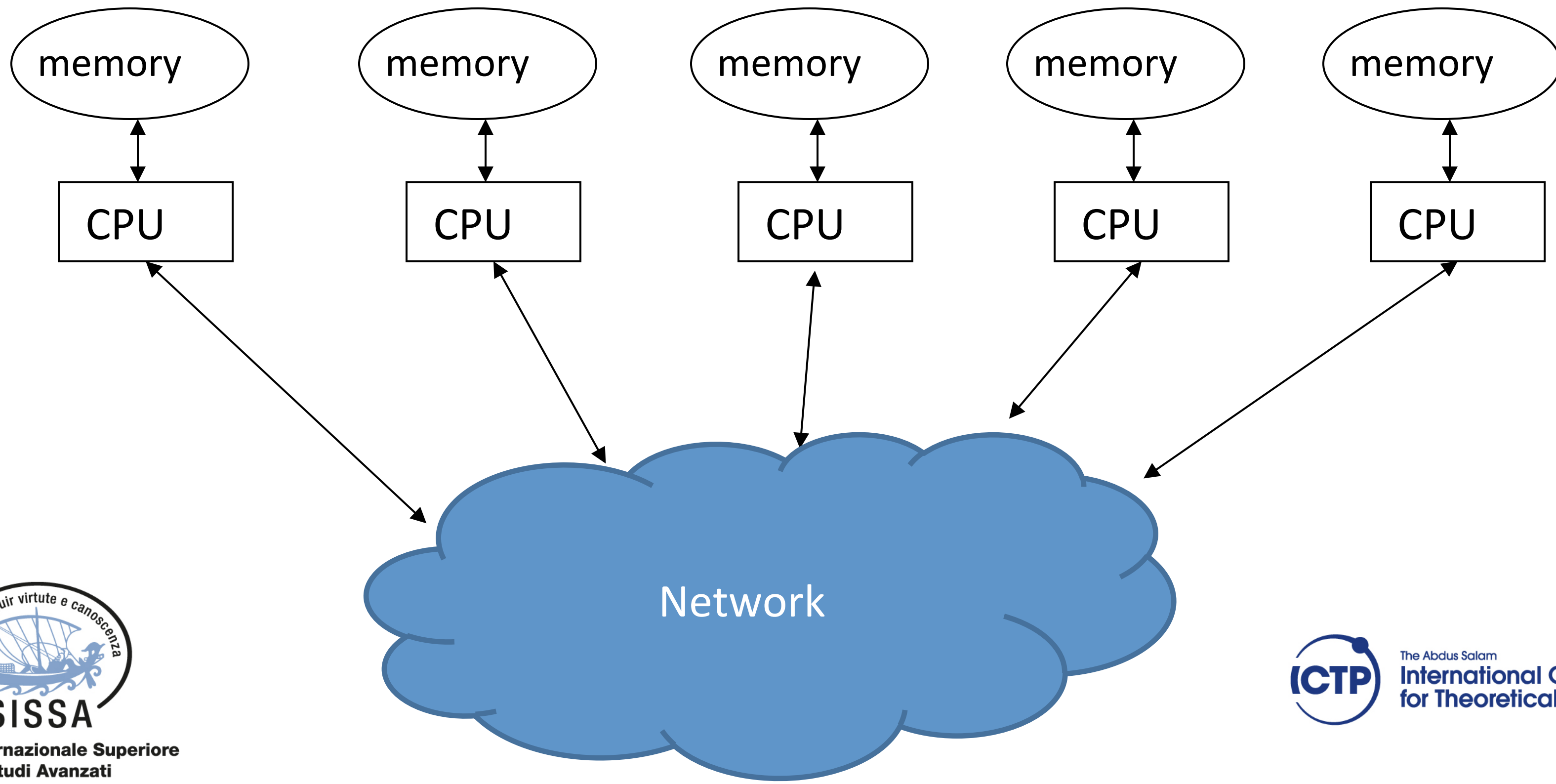
Two basic architectural scheme:

Distributed Memory

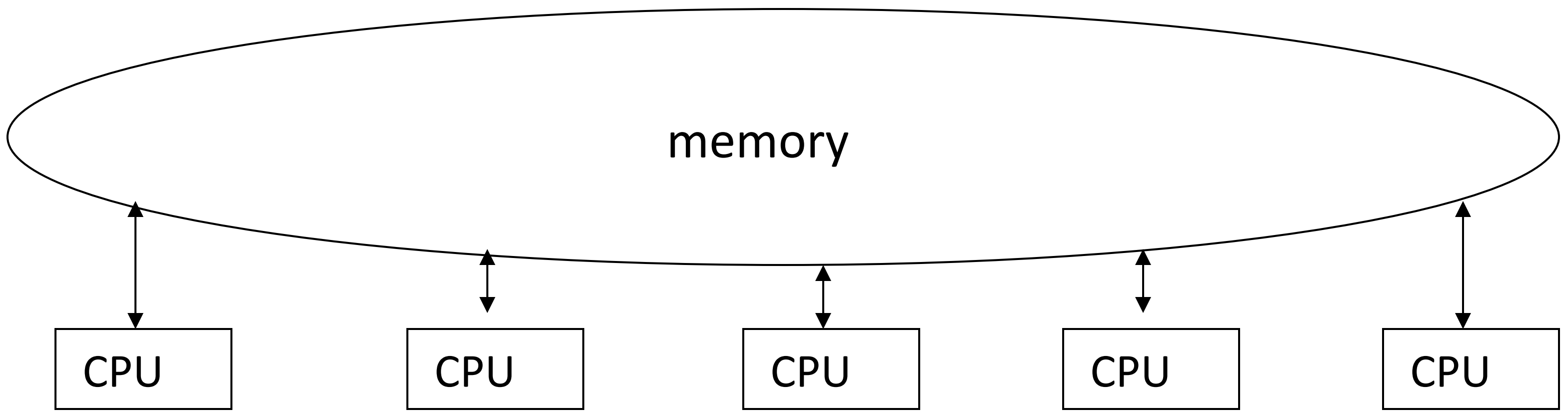
Shared Memory

Now most computers have a mixed architecture

Distributed Memory

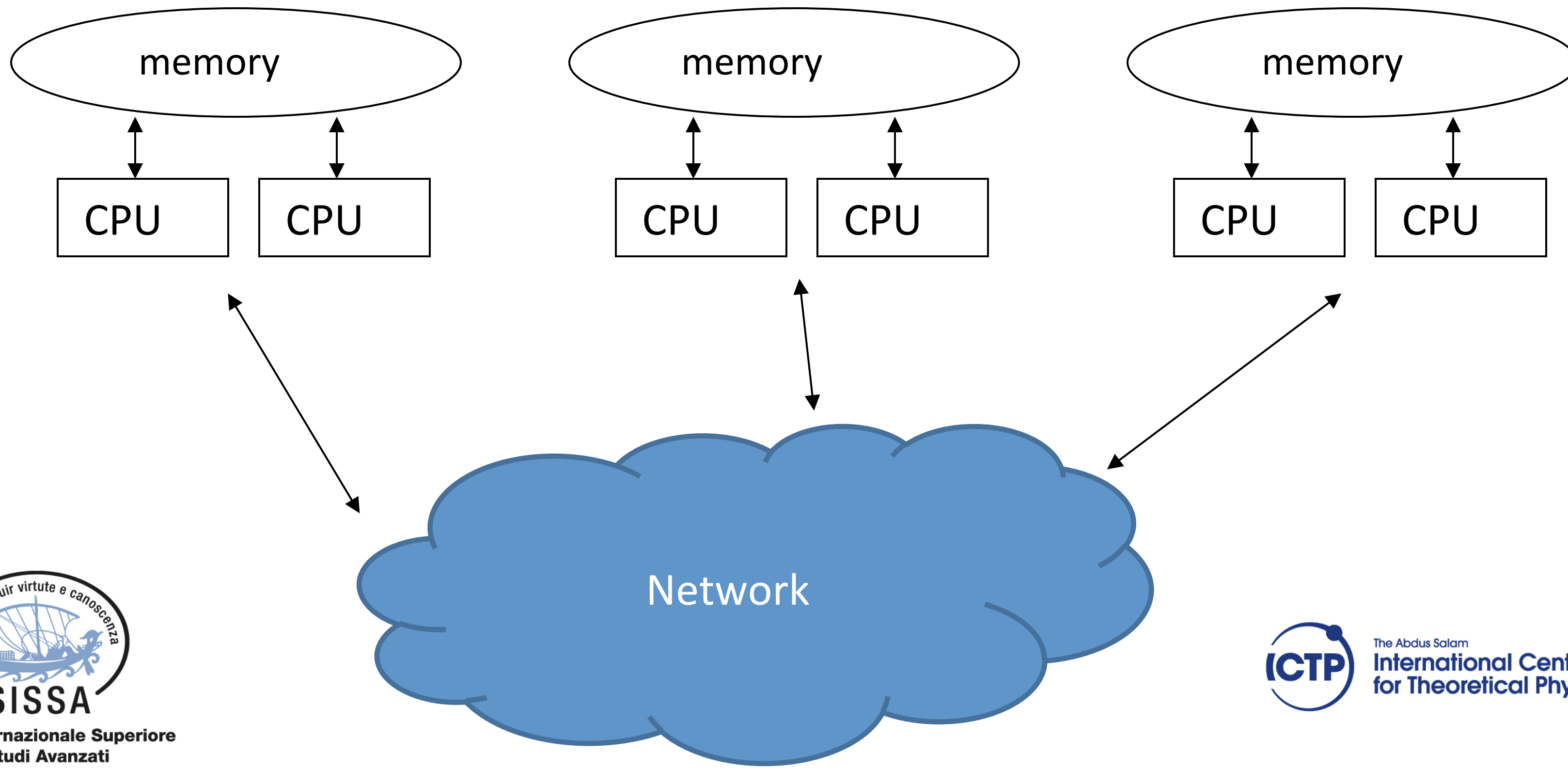


Shared Memory



MHPC
Master in High Performance Computing

Mixed Architectures



SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP
The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

Programming Parallel Paradigms

Are the tools we use to express the parallelism for on a given architecture

They differ in how programmers can manage and define key features like:

- parallel regions
- concurrency
- process communicatio
- synchronism

SISSA
Scuola Internazionale Superiore di Studi Avanzati

Logical Machine Organization

The logical organization, seen by the programmer, could be different from the hardware architecture.

Its quite easy to logically partition a Shared Memory computer to reproduce a Distributed memory Computers.

The opposite is not true.

Message Passing Programming Paradigm

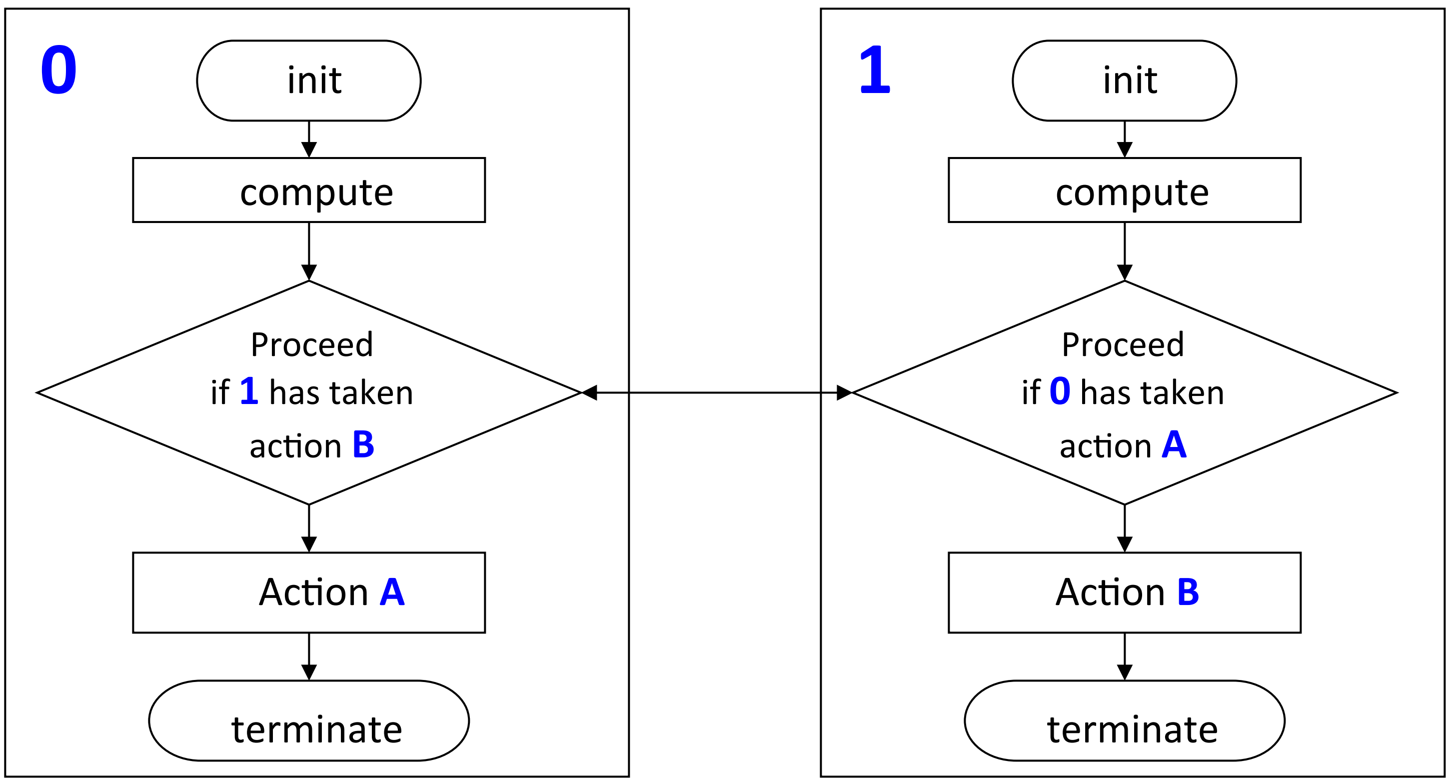
Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

International Centre for Theoretical Physics (ICTP)

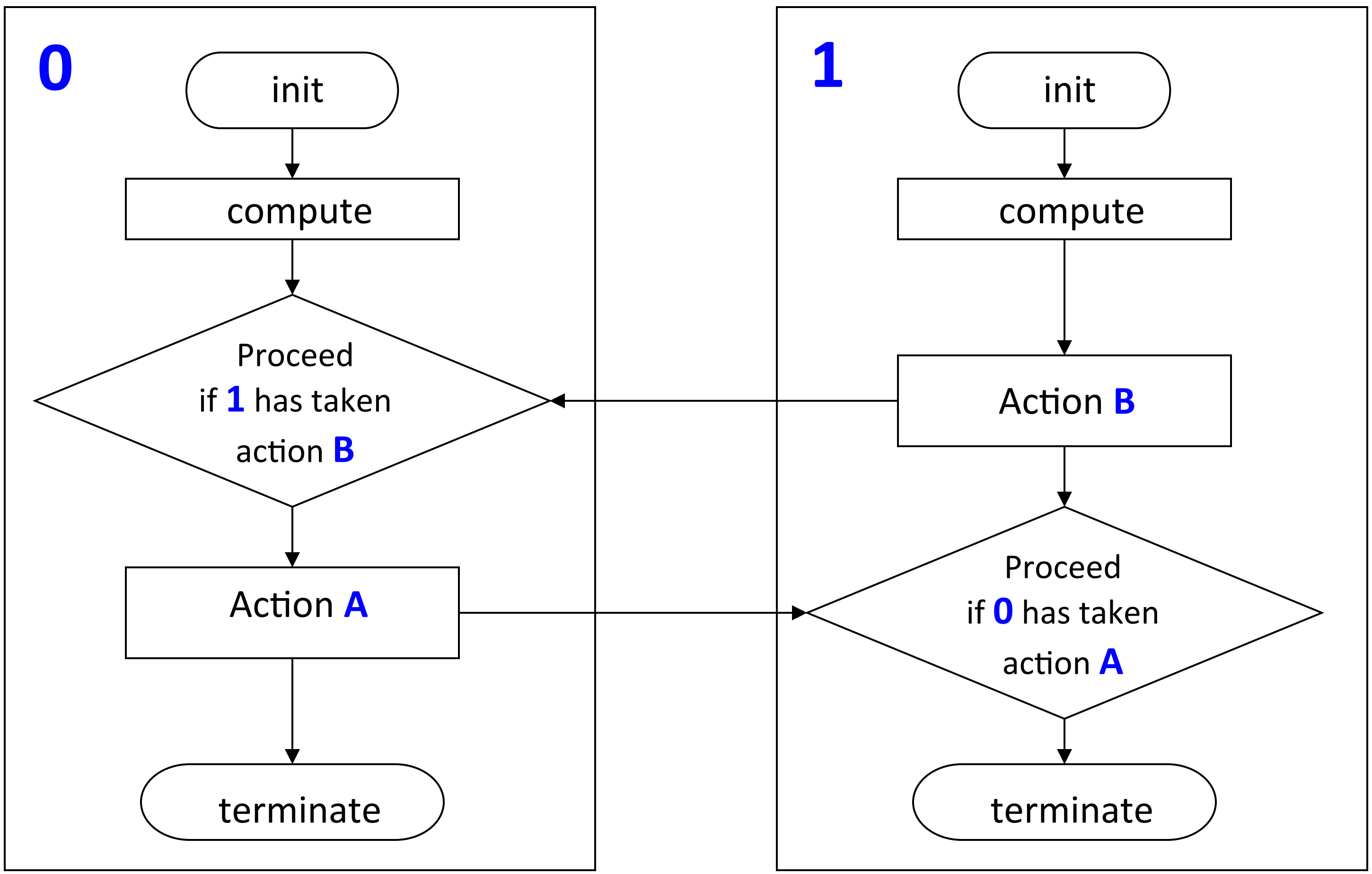
DEADLOCK

Deadlock occurs when 2 (or more) processes are blocked and each is waiting for the other to make progress.



MHPC
Master in High Performance Computing

Avoiding DEADLOCK



Load Balancing

- Equally divide the work among the available resource: processors, memory, network bandwidth, I/O, ...
- This is usually a simple task for the problem decomposition model
- It is a difficult task for the functional decomposition model

Minimizing Communication

When possible reduce the communication events:

Group lots of small communications into large one.

Eliminate synchronizations as much as possible. Each synchronization level off the performance to that of the slowest process.

Overlap Communication and Computation

When possible code your program in such a way that processes continue to do useful work while communicating.

This is usually a non trivial task and is afforded in the very last phase of parallelization.

If you succeed, you have done. Benefits are enormous.

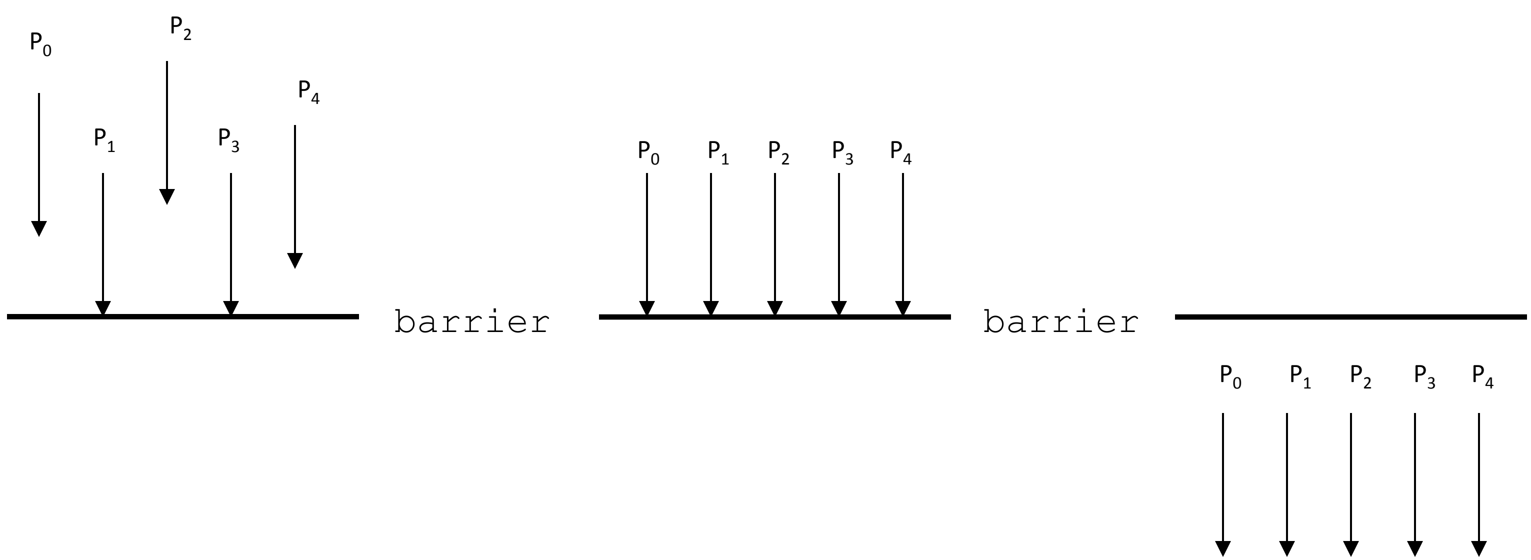
Parallel programming

Lecture 3

Carlo Cavazzoni

MHPC
Master in High Performance Computing

Barrier and Synchronization



ma per seguir virtute e canoscenza
SISSA
Scuola Internazionale Superiore
di Studi Avanzati

The Abdus Salam
International Centre
for Theoretical Physics

Master in High Performance Computing

MESSAGE PASSING PROGRAMMING MODEL

Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

MPI Program Design

- Multiple and <u>separate</u> processes (can be local and remote) concurrently that are coordinated and exchange data through "messages" a "share nothing" parallelization
- Best for coarse grained parallelization
- Distribute large data sets; replicate small data
- Minimize communication or overlap communication and computing for efficiency
- Amdahl's law: speedup is limited by the fraction of serial code plus communication

SISSA Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

What is MPI?

- A standard, i.e. there is a document describing how the API are named and should behave; multiple “levels”, MPI-1 (basic), MPI-2 (advanced), MPI-3 (new) http://www.mpi-forum.org
- A library or API to hide the details of low-level communication hardware and how to use it
- Implemented by multiple vendors
  - Open source and commercial versions
  - Vendor specific versions for certain hardware
  - Not binary compatible between implementations

ma per seguir virtute e canoscenza
SISSA
Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

Goals of MPI

- Allow to write software (source code) that is portable to many different parallel hardware. i.e. agnostic to actual realization in hardware
- Provide flexibility for vendors to optimize the MPI functions for their hardware
- No limitation to a specific kind of hardware and low-level communication type. Running on heterogeneous hardware is possible.
- Fortran77 and C style API as standard interface

Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

MPI in C versus MPI in Fortran

The programming interface (“bindings”) of MPI in C and Fortran are closely related (wrappers for many other languages exist)

MPI in C:

- Use '#include <mpi.h>' for constants and prototypes
- Include only once at the beginning of a file

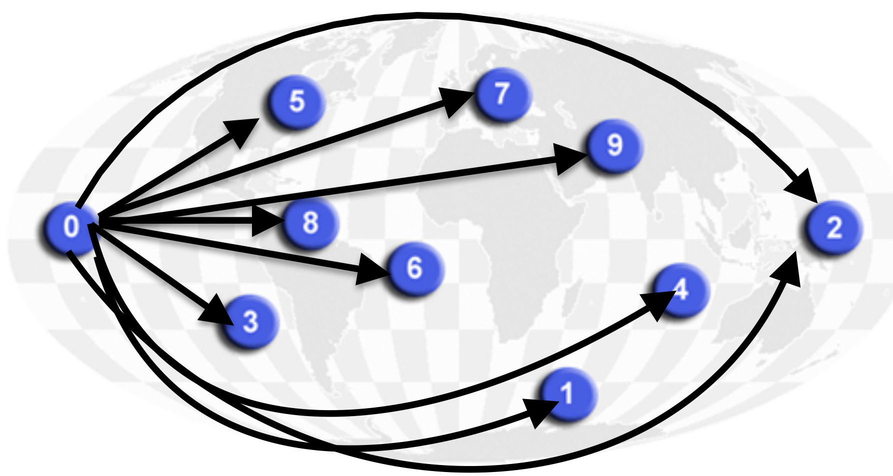
MPI in Fortran:

- Use 'include “mpif.h”' for constants
- Include at the beginning of each module
- All MPI functions are “subroutines” with the same name and same order and type of arguments as in C with return status added as the last argument

MPI Communicators

- Is the fundamental communication facility provided by MPI library. Communication between 2 processes
- Communication take place within a communicator: Source/s and Destination/s are identified by their rank within a communicator

MPI_COMM_WORLD



Communicator Size & Process Rank

A "communicator" is a label identifying a group of processors that are ready for parallel computing with MPI

By default the **MPI_COMM_WORLD** communicator is available and contains all processors allocated by mpirun

Size: How many MPI tasks are there in total?

> **CALL MPI_COMM_SIZE(comm, size, status)**
>
> After the call the integer variable **size** holds the number of processes on the given communicator

Rank: What is the ID of "me" in the group?

> **CALL MPI_COMM_RANK(comm, rank, status)**
>
> After the call the integer variable **rank** holds the ID or the process. This is a number between **0** and **size-1**.

Scuola Internazionale Superiore di Studi Avanzati

Fortran

```
PROGRAM hello
    INCLUDE `mpif.h`
    INTEGER :: ierr, rank, size
    CALL MPI_INIT(ierr)
    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(ierr)
END
```

Important: call MPI_INIT before parsing arguments

Scuola Internazionale Superiore di Studi Avanzati

Phases of an MPI Program

1) Startup

Parse arguments (mpirun may add some)

Identify parallel environment and rank in it

Read and distribute all data

2) Execution

Proceed to subroutine with parallel work

(can be same of different for all parallel tasks)

3) Cleanup

Master in High Performance Computing

MPI Startup / Cleanup

Initializing the MPI environment:

CALL MPI_INIT(STATUS)

Status is integer set to MPI_SUCCESS, if operation was successful; otherwise to error code

Releasing the MPI environment:

CALL MPI_FINALIZE(STATUS)

NOTES:

All MPI tasks have to call **MPI_INIT** & **MPI_FINALIZE**

MPI_INIT may only be called once in a program

No MPI calls allowed outside of the region between calling **MPI_INIT** and **MPI_FINALIZE**

SISSA
Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

The Message

A message is an array of elements of some particular MPI data type

MPI defines a number of constants that correspond to language *datatypes* in Fortran and C

When an MPI routine is called, the Fortran (or C) datatype of the data being passed must match the corresponding MPI integer constant

Message Structure

envelope				body		
source	destination	communicator	tag	buffer	count	datatype

SISSA
Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Calling MPI_BCAST

MPI_BCAST(buffer, count, type, sender, comm, err)

buffer:	buffer with data
count:	number of data items to be sent
type:	type (=size) of data items
sender:	rank of sending processor of data
comm:	group identifier, MPI_COMM_WORLD
err:	error status of operation

NOTES:

- buffers must be large enough (can be larger)
- Data type must match (MPI does not check this)
- all ranks that belong to the communicator must call this

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

  print *, "After Bcast operation I'm ", myrank, & " and my message content is ", imesg

  call MPI_FINALIZE(ierr)

end program bcast
```

```
program bcast
  implicit none
  include "mpif.h"
  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
```

P_0

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_1

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_2

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_3

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

```
program bcast
  implicit none
  include "mpif.h"
  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
  call MPI_INIT(ierr)
```

P_0

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast
  implicit none
  include "mpif.h"
  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
```

P_0

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

Scuola Internazionale Superiore di Studi Avanzati

```
program bcast
  implicit none
  include "mpif.h"
  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
     " and my message content is ", imesg
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
     " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

  print *, "After Bcast operation I'm ", myrank, &
      " and my message content is ", imesg
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```fortran
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

  print *, "After Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_FINALIZE(ierr)
```

P_0

- myrank = 0
- ncpus = 4
- imesg = 0
- ierr = MPI_SUC...
- comm = MPI_C...

P_1

- myrank = 1
- ncpus = 4
- imesg = 0
- ierr = MPI_SUC...
- comm = MPI_C...

P_2

- myrank = 2
- ncpus = 4
- imesg = 0
- ierr = MPI_SUC...
- comm = MPI_C...

P_3

- myrank = 3
- ncpus = 4
- imesg = 0
- ierr = MPI_SUC...
- comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

  print *, "After Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_FINALIZE(ierr)

end program bcast
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUCC
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...