MHPC
Master in High Performance Computing

Message Passing Programming Paradigm

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)
International Centre for Theoretical Physics (ICTP)

SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP The Abdus Salam International Centre for Theoretical Physics

MHPC
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-3/4 (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

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

MPI Program Design

- Multiple and separate processes (can be local and remote) concurrently that are coordinated and exchange data through “messages”
  - ➢ a “share nothing” parallelization
- Best for coarse grained parallelization
- Minimize communication or overlap communication and computing for efficiency
  - ➢ Amdahl's law: speedup is limited by the fraction of serial code plus communication

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

MHPC
Master in High Performance Computing

A Fundamental Reference (list of the MPI-1 functions calls)

http://mpi-forum.org/docs/mpi-1.1/mpi-11-html/node182.html - Node182

When next writing an MPI program check all MPI function interfaces you introduce in your code at the above link

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

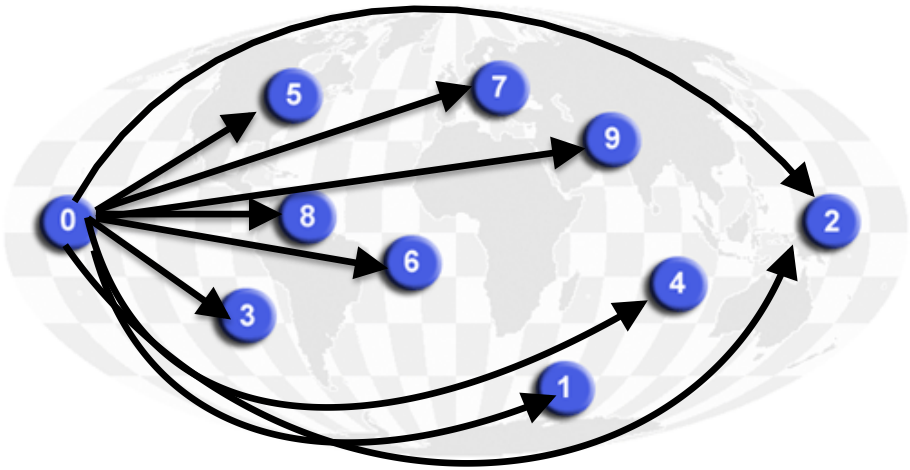
ICTP The Abdus Salam International Centre for Theoretical Physics

Master in High Performance Computing

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



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

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**.

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

MHPC
Master in High Performance Computing

Phases of an MPI Program

1. Startup

- Parse arguments (mpirun may add some!)
- Identify parallel environment and rank of process
- Read and distribute all data

2. Execution

- Proceed to subroutine with parallel work (can be same of different for all parallel tasks)

3. Cleanup

CAUTION: this sequence may be run only once!

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**

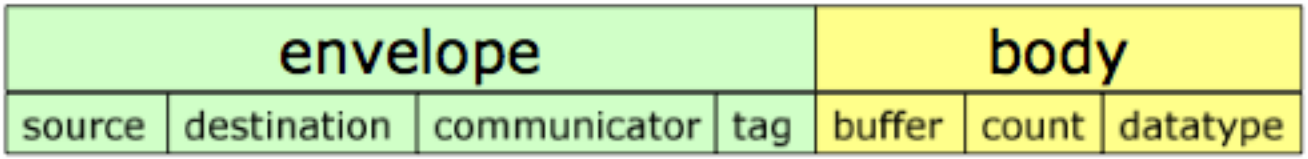
The Message

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

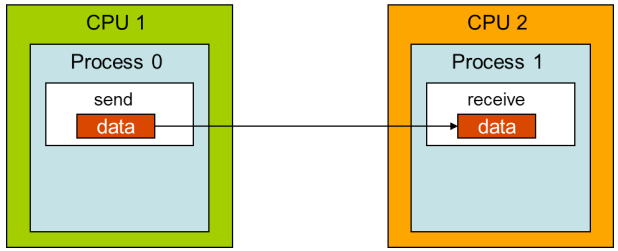
When an MPI routine is called, the Fortran (or C) datatype of the data being passed must match the corresponding MPI integer constant

Message Structure

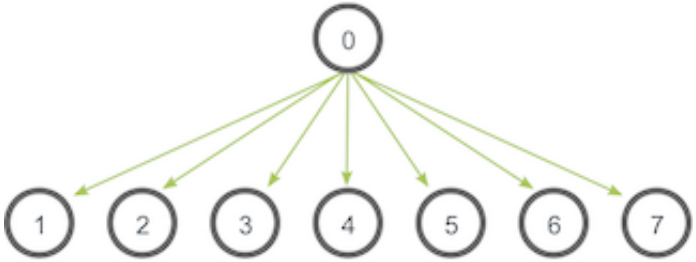


Communication Models in MPI

Point-to-Point Communication



Collective Communication



MPI DATA TYPES

MPI datatype handle	C datatype
MPI_INT	int
MPI_SHORT	short
MPI_LONG	long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_CHAR	char
MPI_BYTE	unsigned char

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...

```
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...

```
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...

MHPC
Master in High Performance Computing

STANDARD BLOCKING SEND - RECV

MPI_SEND(buf, count, type, dest, tag, comm, ierr)

MPI_RECV(buf, count, type, dest, tag, comm, status, ierr)

Buf array of MPI type **type**.

Count (INTEGER) number of element of **buf** to be sent/recv

Type (INTEGER) MPI type of **buf**

Dest (INTEGER) rank of the destination process

Tag (INTEGER) number identifying the message

Comm (INTEGER) communicator of the sender and receiver

*** Status** (INTEGER) array of size **MPI_STATUS_SIZE** containing communication status information

Ierr (INTEGER) error code

**** used only for receive operations***

MHPC
Master in High Performance Computing

Wildcards

Both in Fortran and C **MPI_RECV** accept wildcard:

- To ignore the receiving status object: MPI_STATUS_IGNORE
- To receive from any source: MPI_ANY_SOURCE
- To receive with any tag: MPI_ANY_TAG
- Actual source and tag are returned in the receiver's status parameter => status.MPI_SOURCE, status.MPI_TAG

MPI_GET_COUNT(status, datatype, count)

[IN status] return status of receive operation (Status)

[IN datatype] datatype of each receive buffer entry (handle)

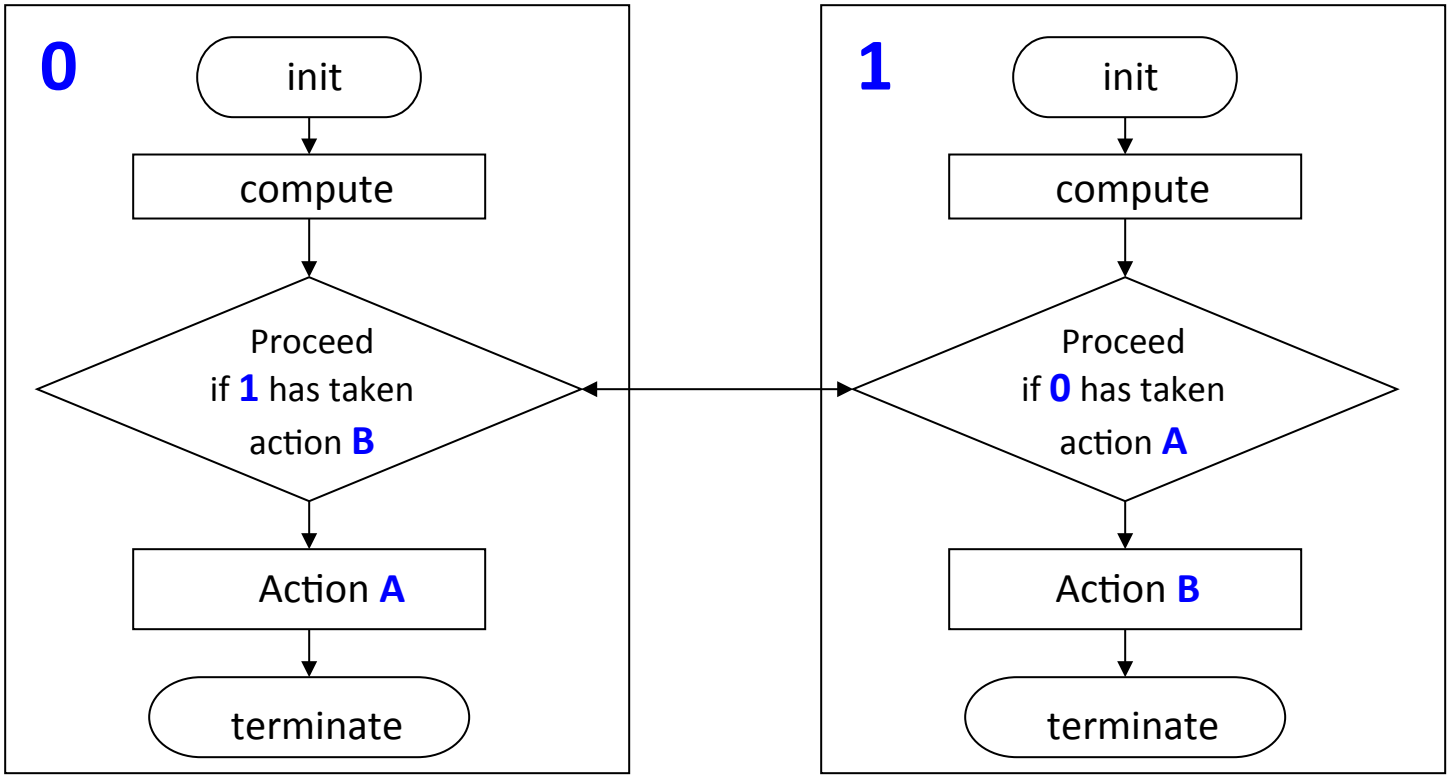
[OUT count] number of received entries (integer)

```
PROGRAM send_recv
      INCLUDE ‘mpif.h‘
      INTEGER :: ierr, myid, nproc, 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)
      IF( myid .EQ. 0 ) THEN
            A(1) = 3.0
            A(2) = 5.0
            CALL MPI_SEND(A, 2, MPI_REAL, 1, 10, MPI_COMM_WORLD, ierr)
      ELSE IF( myid .EQ. 1 ) THEN
            CALL MPI_RECV(A, 2, MPI_REAL, 0, 10, MPI_COMM_WORLD, status, ierr)
            WRITE(6,*) myid,’: a(1)=’,a(1),’ a(2)=’,a(2)
      END IF
      CALL MPI_FINALIZE(ierr)
END
```

MHPC
Master in High Performance Computing

DEADLOCK

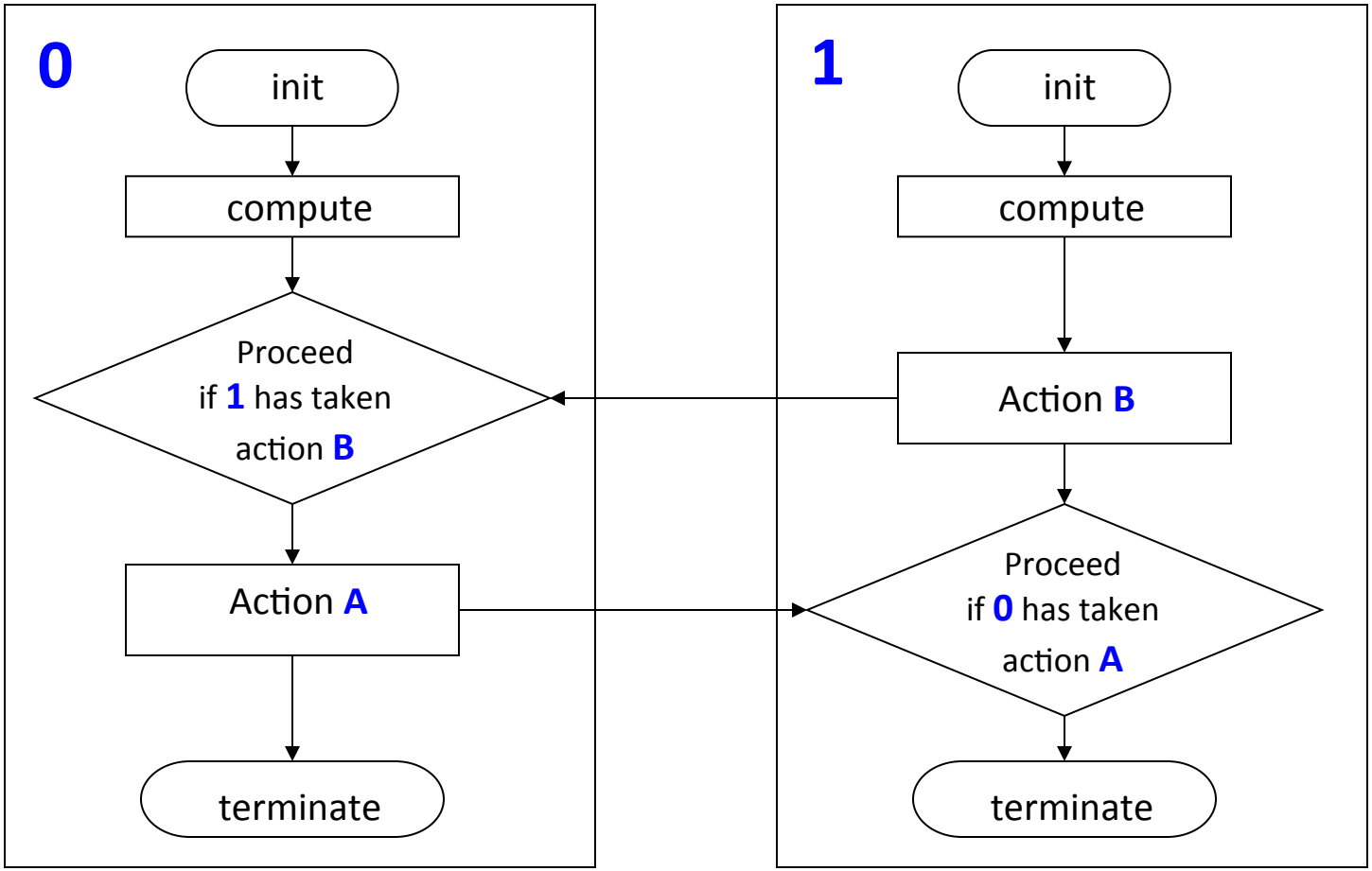
Deadlock occurs when 2 (or more) processes are blocked and each is waiting for the other to make progress.



SISSA
ICTP

MHPC
Master in High Performance Computing

Avoiding DEADLOCK



SISSA
ICTP