The Abdus Salam
International Centre
for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

Overview on Common Strategies for Parallelization

Ivan Girotto – igirotto@ictp.it

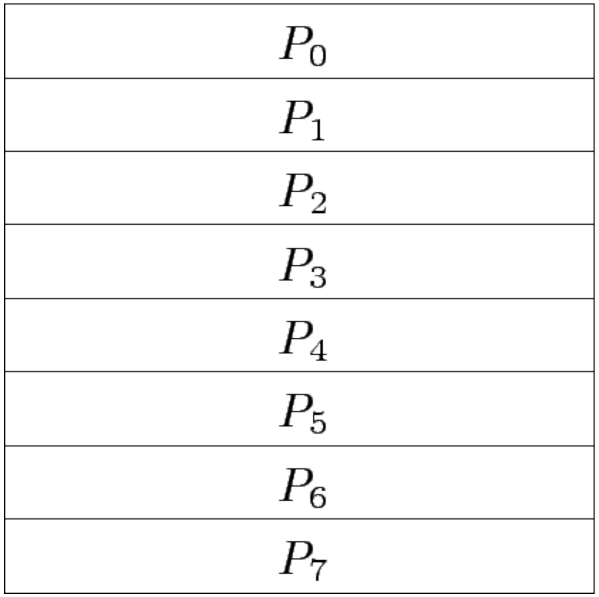
Information & Communication Technology Section (ICTS)

International Centre for Theoretical Physics (ICTP)

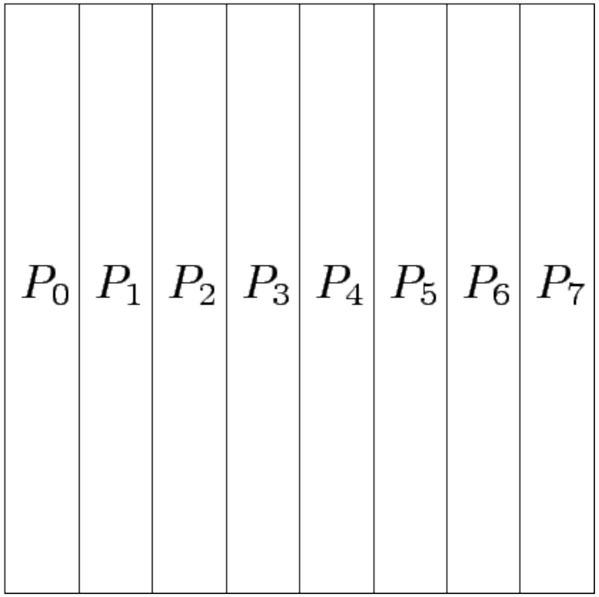
Static Data Partitioning

The simplest data decomposition schemes for dense matrices are 1-D block distribution schemes.

row-wise distribution



column-wise distribution



Distributed Data Vs Replicated Data

- Replicated data distribution is useful if it helps to reduce the communication among process at the cost of bounding scalability
- Distributed data is the ideal data distribution but not always applicable for all data-sets
- Usually complex application are a mix of those techniques

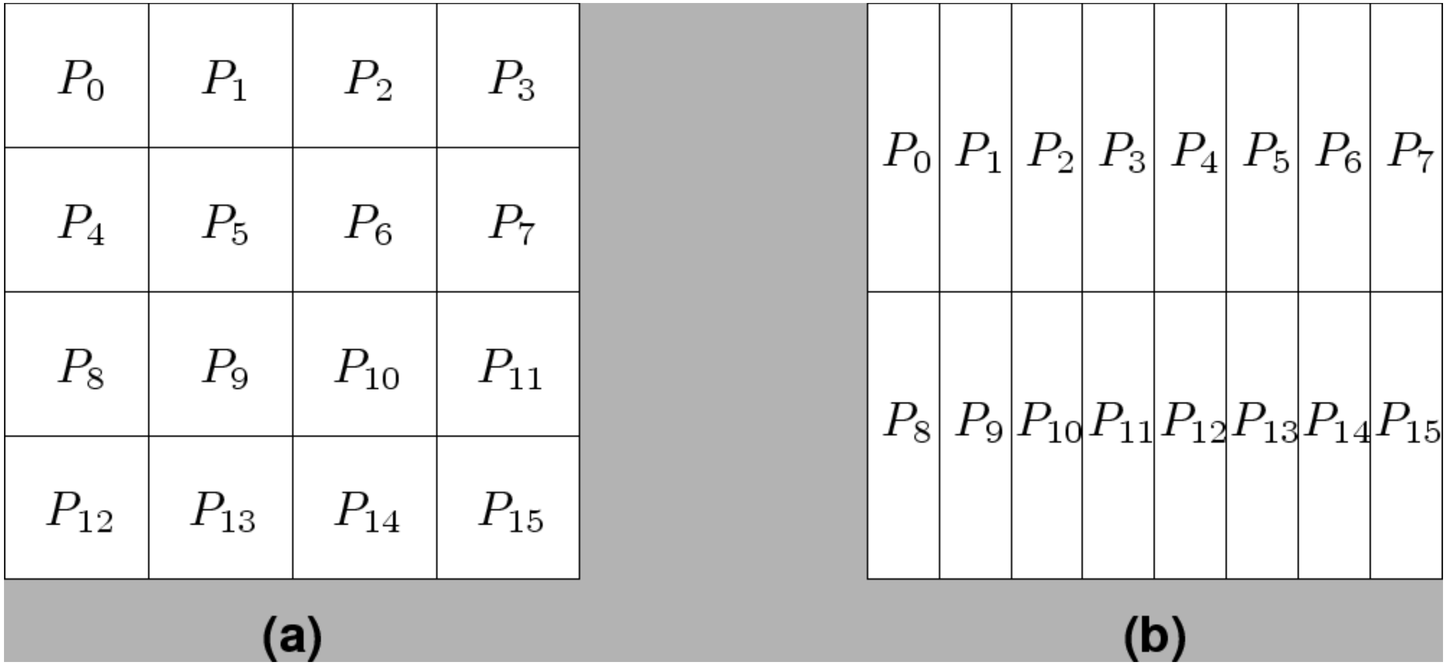
Global Vs Local Indexes

- In sequential code you always refer to global indexes
- With distributed data you must handle the distinction between global and local indexes (and possibly implementing utilities for transparent conversion)

Local Idx	1	2	3	1	2	3	1	2	3
Global Idx	1	2	3	4	5	6	7	8	9

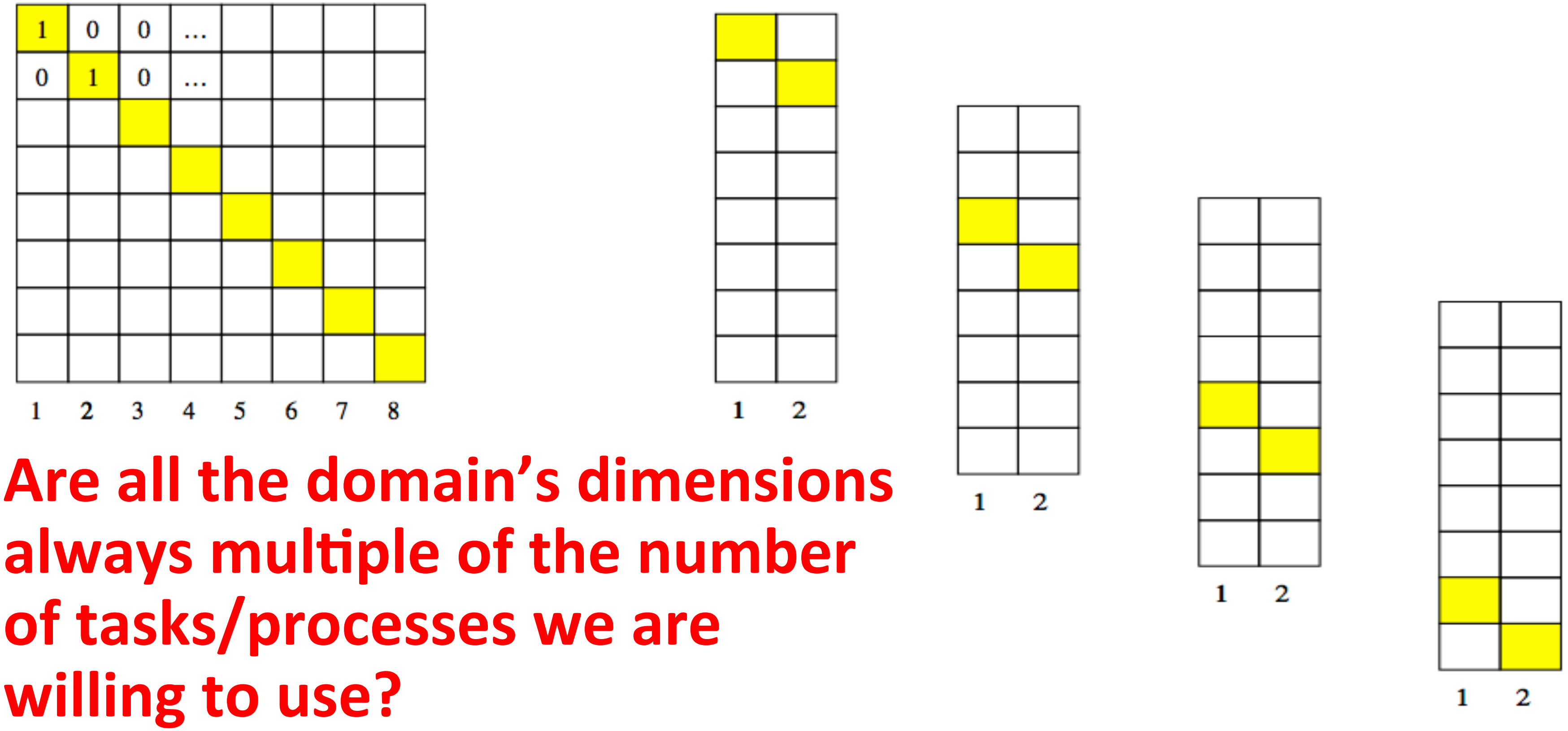
Block Array Distribution Schemes

Block distribution schemes can be generalized to higher dimensions as well.


P_0 P_1 P_2 P_3
P_4 P_5 P_6 P_7
P_8 P_9 P_{10} P_{11}
P_{12} P_{13} P_{14} P_{15}
(a)
P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7
P_8 P_9 P_{10} P_{11} P_{12} P_{13} P_{14} P_{15}
(b)

Degree to which tasks/data can be subdivided is limit to concurrency and parallel execution!!

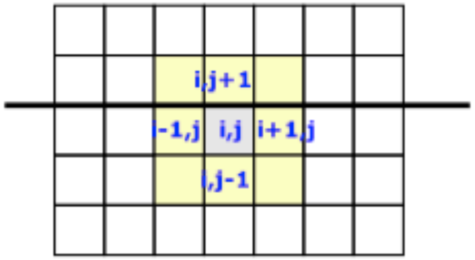
Collaterals to Domain Decomposition /1



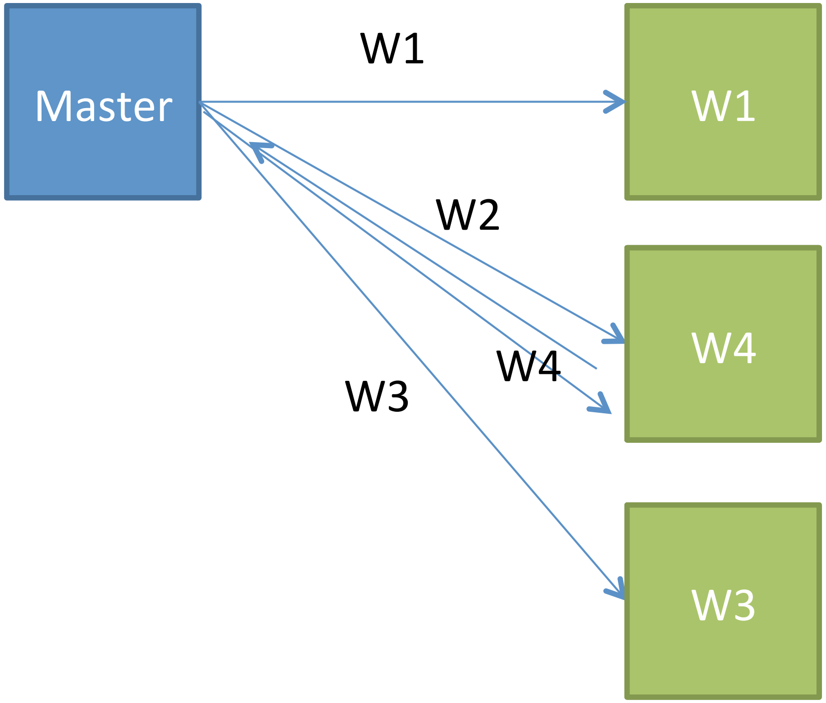
Are all the domain's dimensions always multiple of the number of tasks/processes we are willing to use?

Collaterals to Domain Decomposition /2

sub-domain boundaries



Master/Slave



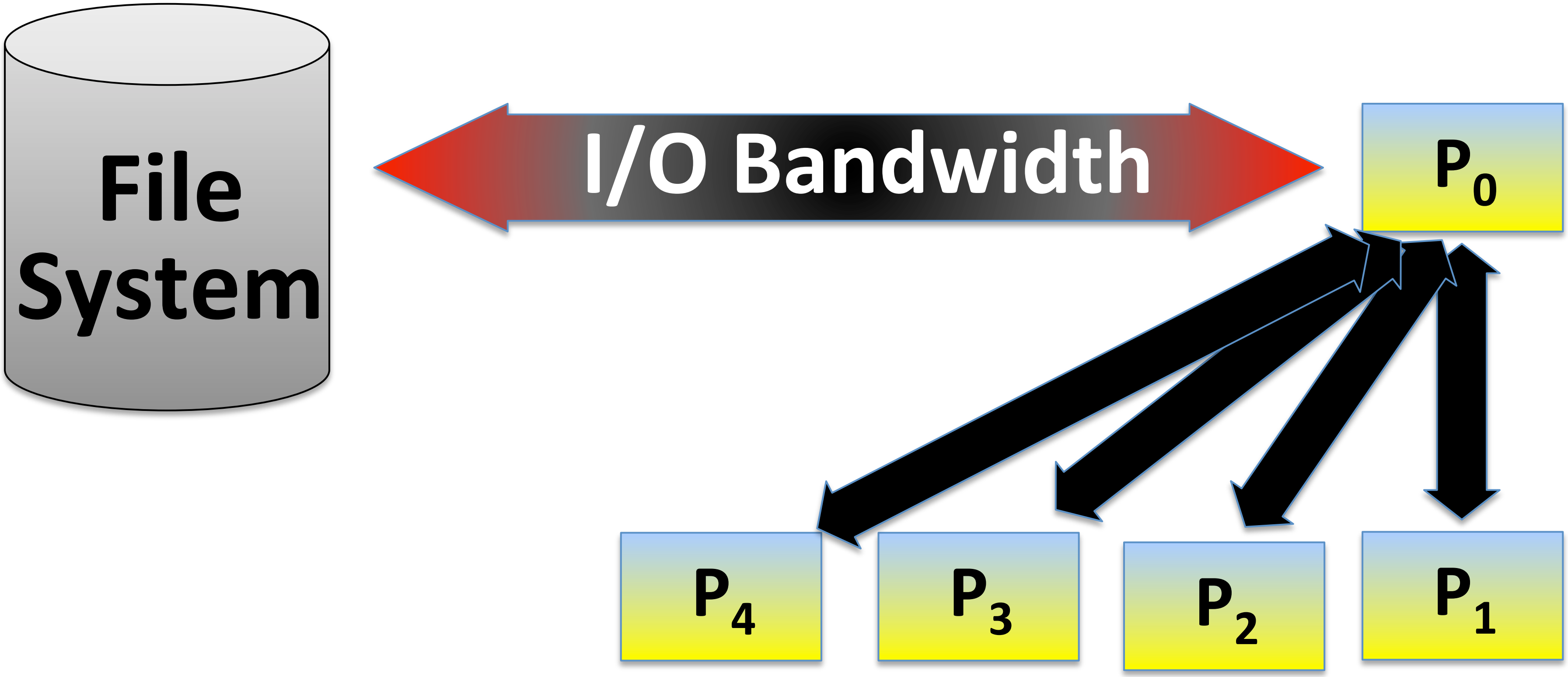
Task Farming

- Many independent programs (tasks) running at once
  - each task can be serial or parallel
  - “independent” means they don’t communicate directly
  - Processes possibly driven by the mpirun framework

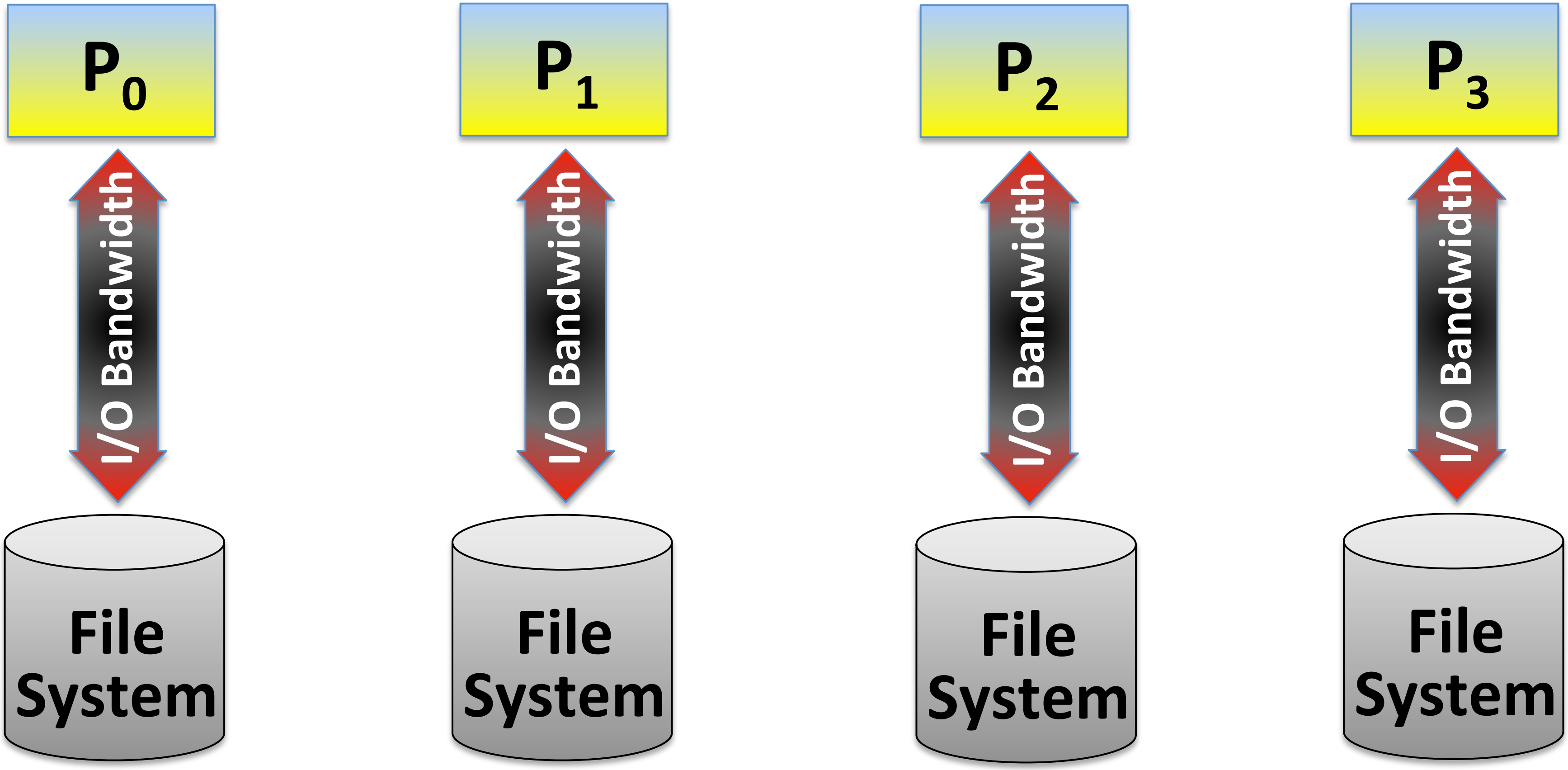
```
[igirotto@localhost]$ more my_shell_wrapper.sh
#!/bin/bash
#example for the OpenMPI implementation
./prog.x --input input_${OMPI_COMM_WORLD_RANK}.dat

[igirotto@localhost]$ mpirun -np 400 ./my_shell_wrapper.sh
```

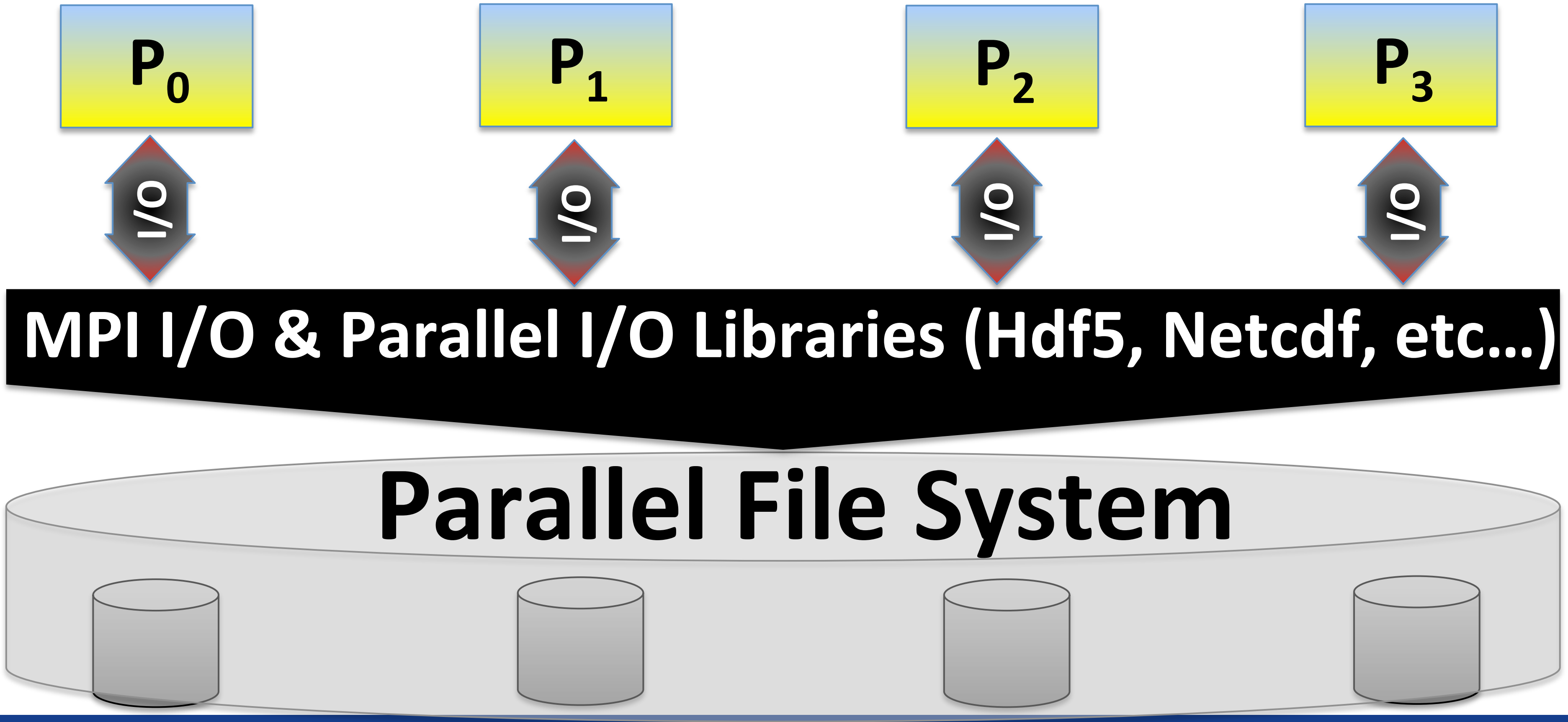
Parallel I/O



Parallel I/O


P_0
P_1
P_2
P_3
I/O Bandwidth
I/O Bandwidth
I/O Bandwidth
I/O Bandwidth
File System
File System
File System
File System

Parallel I/O



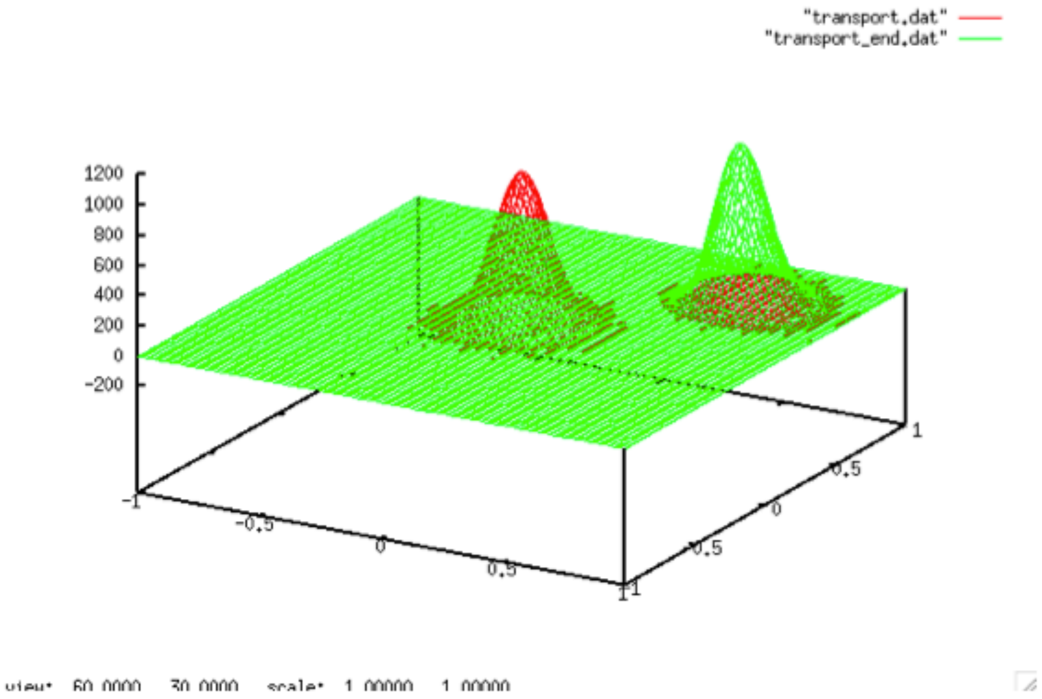
P_0 P_1 P_2 P_3

I/O I/O I/O I/O

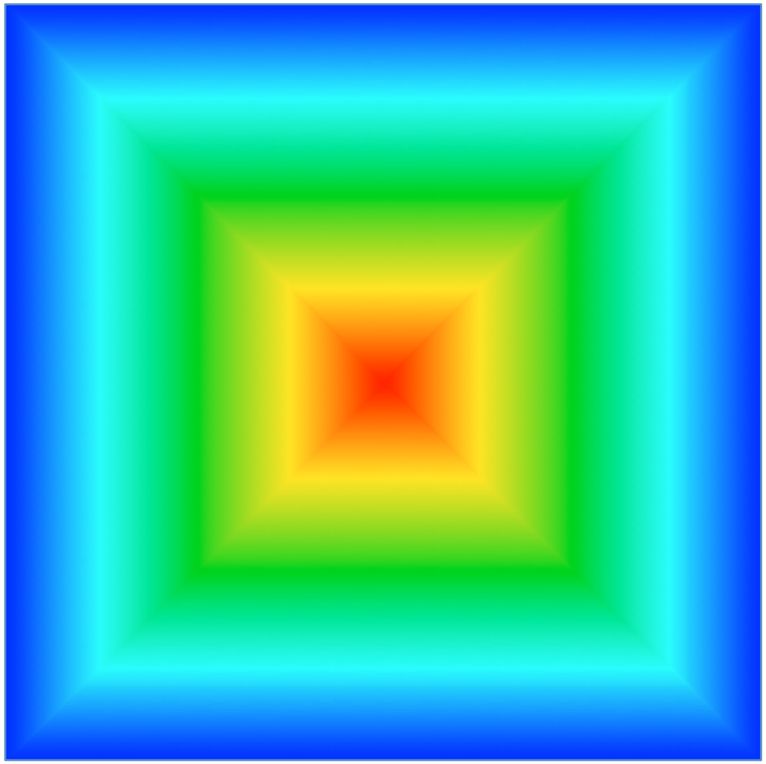
MPI I/O & Parallel I/O Libraries (Hdf5, Netcdf, etc...)

Parallel File System

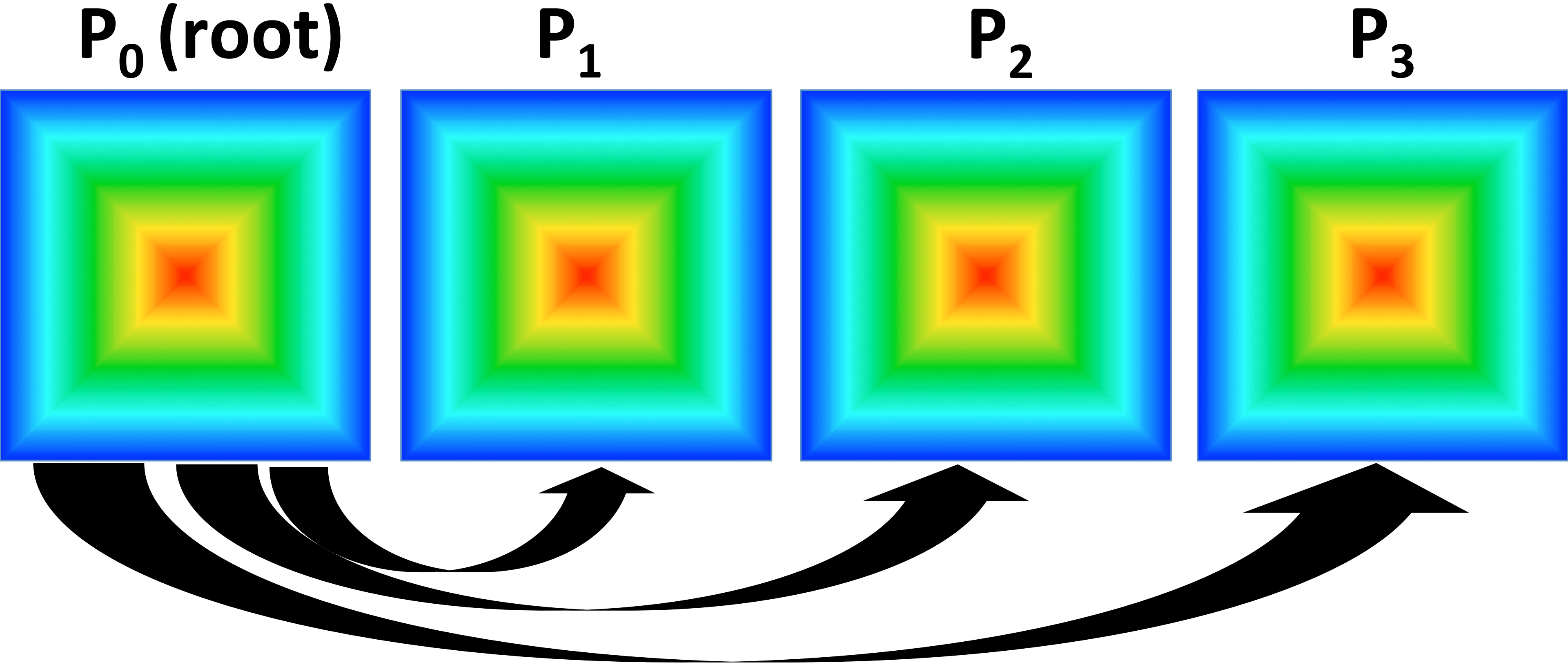
The Transport Code - Parallel Version

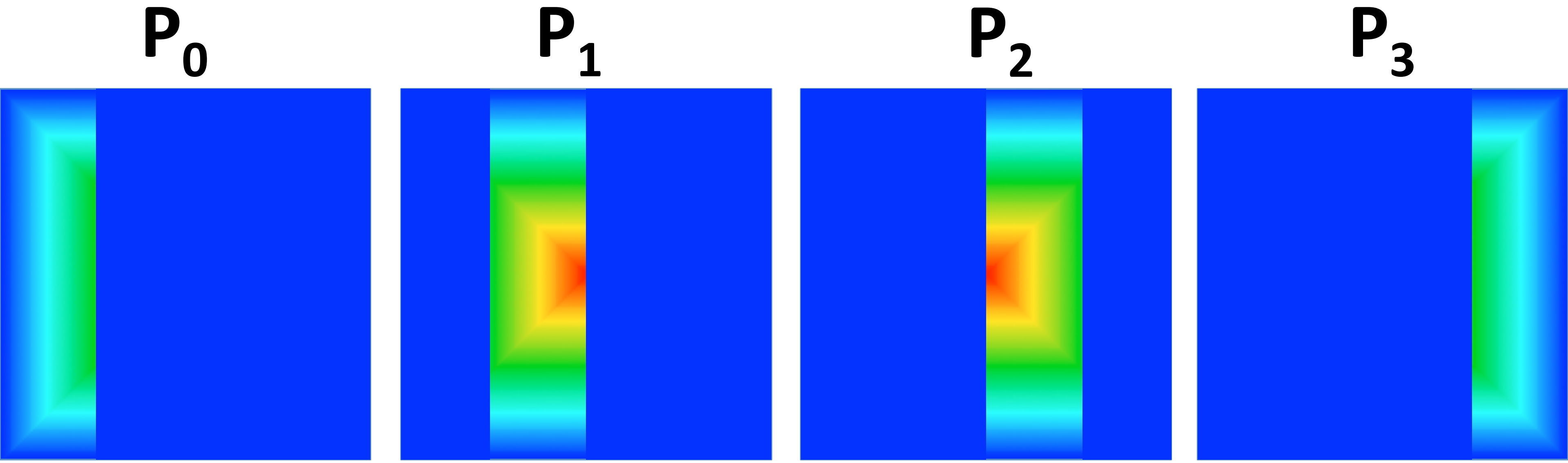


P_0



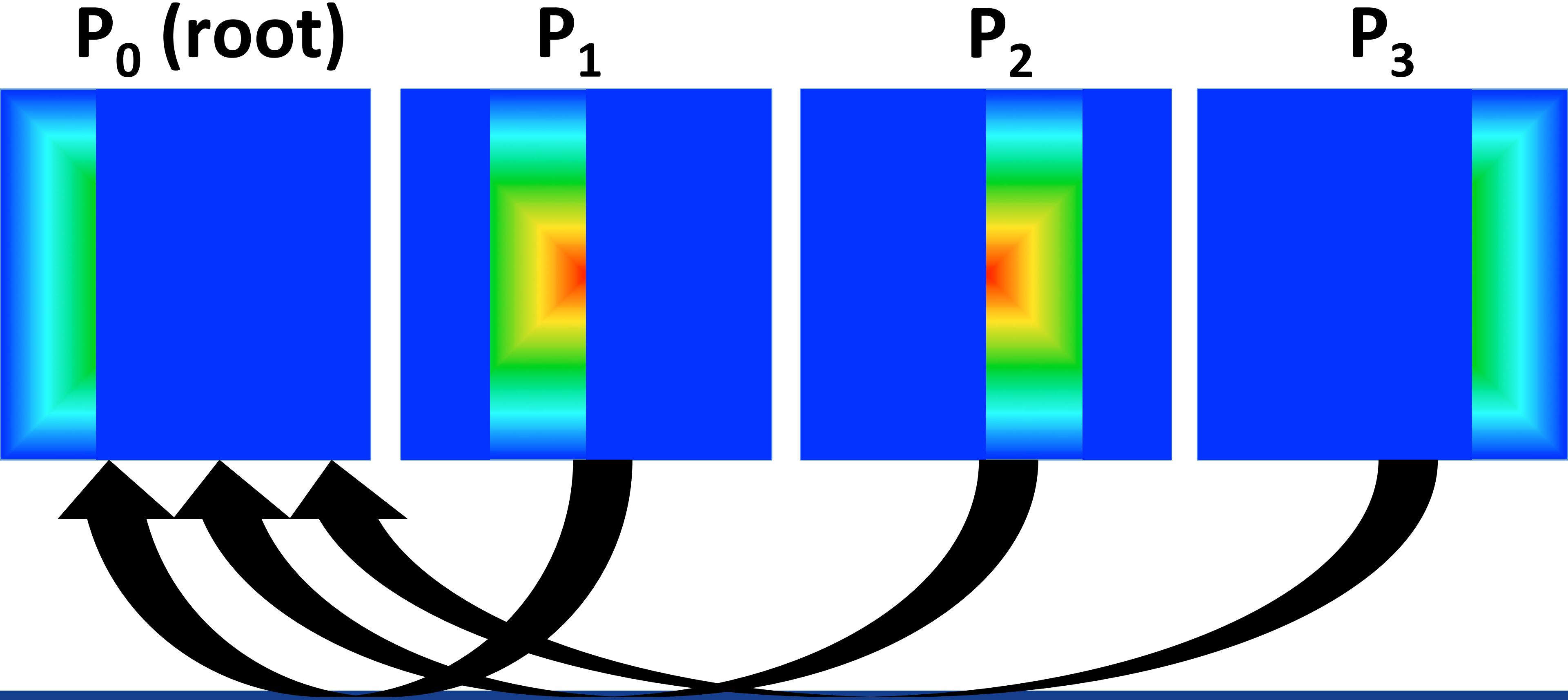
call MPI_BCAST(...)





call evolve(dtfact)

call MPI_REDUCE(..., MPI_SUM, ...)



P_0 (root)

P_1

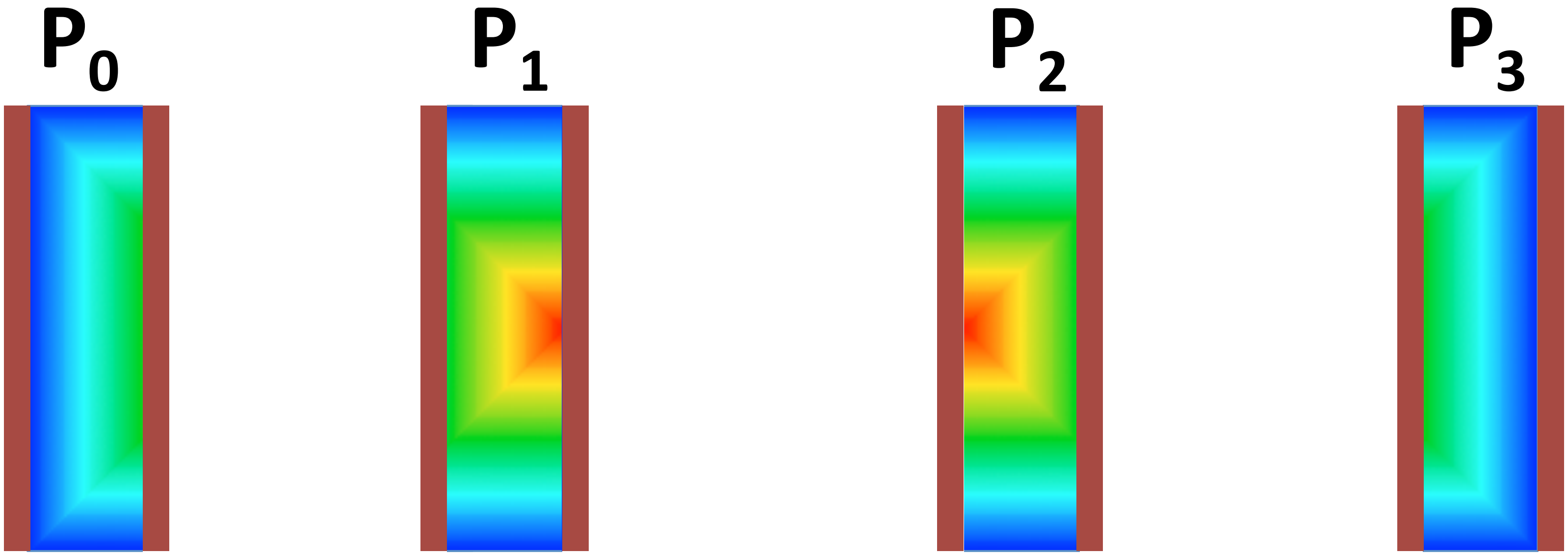
P_2

P_3

The Transport Code - Parallel Version

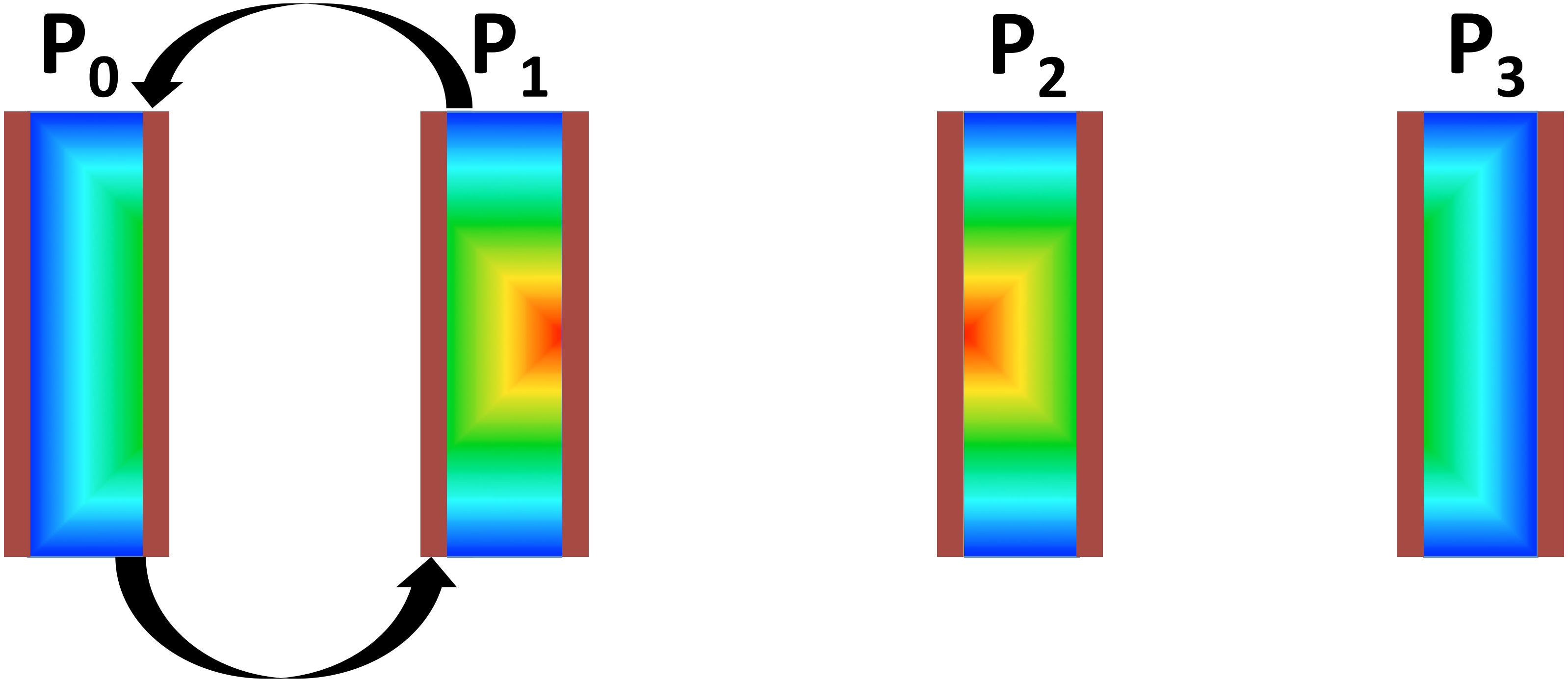
- Replicated data
- Compute domain (and workload) distribution among processes
- Master-slaves: P_0 drives all processes
- Large amount of data communication
  - at each step P_0 distribute data to all processes and collect the contribution of each process
- Problem size scaling limited in memory capacity

The Transport Code - Parallel Version



```
call evolve( dtfact )
```

Data exchange among processes



```
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
```

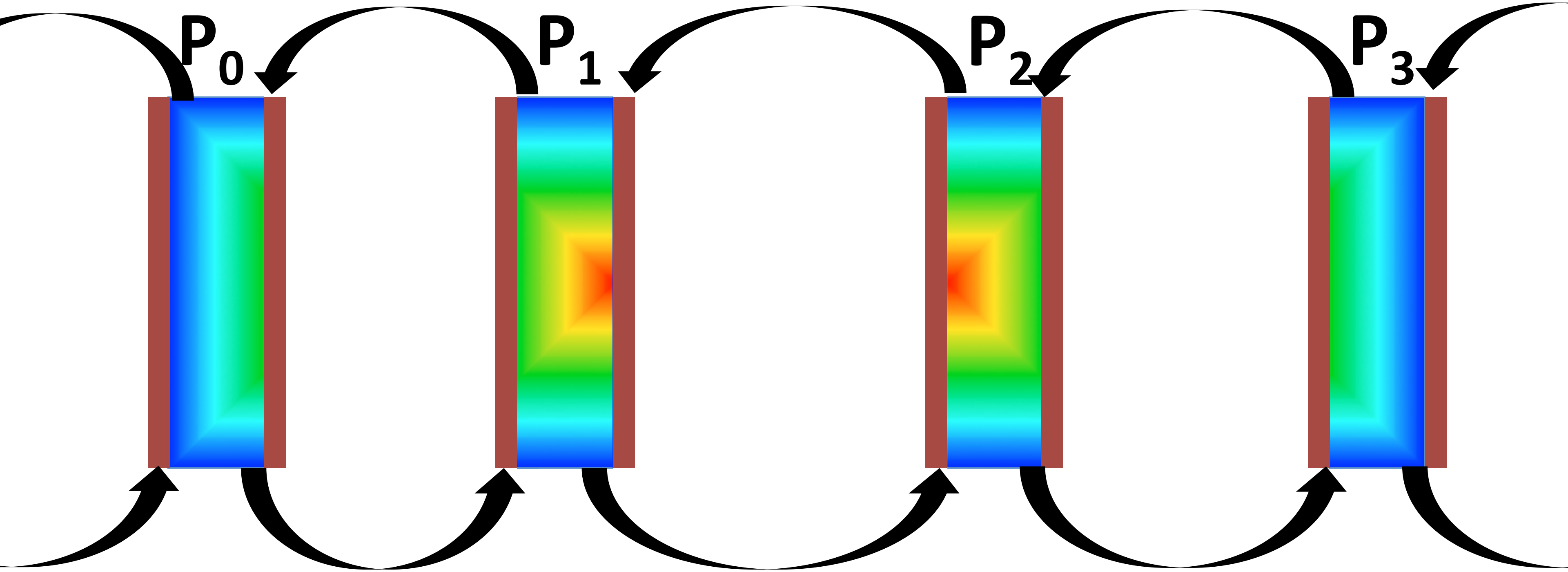
```
PROGRAM error_lock
        INCLUDE ‘mpif.h‘
        INTEGER :: ierr, myid, nproc, status(MPI_STATUS_SIZE)
        REAL :: A(2), B(2)
        CALL MPI_INIT(ierr)
        CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
        CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
        IF( myid .EQ. 0 ) THEN
                a(1) = 2.0
                a(2) = 4.0
                CALL MPI_SEND(a, 2, MPI_REAL, 1, 10, MPI_COMM_WORLD, ierr)
                CALL MPI_RECV(b, 2, MPI_REAL, 1, 11, MPI_COMM_WORLD, status, ierr)
        ELSE IF( myid .EQ. 1 ) THEN
                a(1)  = 3.0
                a(2)  = 5.0
                CALL MPI_SEND(a, 2, MPI_REAL, 0, 11, MPI_COMM_WORLD, ierr)
                CALL MPI_RECV(b, 2, MPI_REAL, 0, 10, MPI_COMM_WORLD, status, ierr)
        END IF
        WRITE(6,*) myid, ’: a(1)=’, a(1), ’ a(2)=’, a(2)
        CALL MPI_FINALIZE(ierr)
END
```

```
PROGRAM error_lock
      INCLUDE ‘mpif.h‘
      INTEGER :: ierr, myid, nproc, status(MPI_STATUS_SIZE)
      REAL :: A(2), B(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) = 2.0
            a(2) = 4.0
            CALL MPI_SEND(a, 2, MPI_REAL, 1, 10, MPI_COMM_WORLD, ierr)
            CALL MPI_RECV(b, 2, MPI_REAL, 1, 11, MPI_COMM_WORLD, status, ierr)
      ELSE IF( myid .EQ. 1 ) THEN
            a(1)  = 3.0
            a(2)  = 5.0
            CALL MPI_SEND(a, 2, MPI_REAL, 0, 11, MPI_COMM_WORLD, ierr)
            CALL MPI_RECV(b, 2, MPI_REAL, 0, 10, MPI_COMM_WORLD, status, ierr)
      END IF
      WRITE(6,*) myid, ’: a(1)=’, a(1), ’ a(2)=’, a(2)
      CALL MPI_FINALIZE(ierr)
END
```

Deadlock!!

```
PROGRAM error_lock
    INCLUDE ‘mpif.h‘
    INTEGER :: ierr, myid, nproc, status(MPI_STATUS_SIZE)
    REAL :: A(2), B(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) = 2.0
        a(2) = 4.0
        CALL MPI_RECV(b, 2, MPI_REAL, 1, 11, MPI_COMM_WORLD, status, ierr)
        CALL MPI_SEND(a, 2, MPI_REAL, 1, 10, MPI_COMM_WORLD, ierr)
    ELSE IF( myid .EQ. 1 ) THEN
        a(1)  = 3.0
        a(2)  = 5.0
        CALL MPI_SEND(a, 2, MPI_REAL, 0, 11, MPI_COMM_WORLD, ierr)
        CALL MPI_RECV(b, 2, MPI_REAL, 0, 10, MPI_COMM_WORLD, status, ierr)
    END IF
    WRITE(6,*) myid, ’: a(1)=’, a(1), ’ a(2)=’, a(2)
    CALL MPI_FINALIZE(ierr)
END
```

proc_down = mod(proc_me - 1 + nprocs , nprocs)



proc_up = mod(proc_me + 1 , nprocs)

STANDARD NO-BLOCKING SEND - RECV

- Basic point-2-point communication routines in MPI.

MPI_ISEND(buf, count, type, dest, tag, comm, req, ierr)

MPI_IRECV(buf, count, type, source, tag, comm, req, ierr)

Buf array of MPI type **type**.

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

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

Dest (INTEGER) rank of the destination process / **Source** (INTEGER) rank of the source process

Tag (INTEGER) number identifying the message

Comm (INTEGER) communicator of the sender and receiver

Req (INTEGER) output, identifier of the communications handle

Ierr (INTEGER) error code

STANDARD NO-BLOCKING WAIT

- A call to this subroutine cause the code to wait until the communication pointed by req is complete
- Handler for no-blocking communication

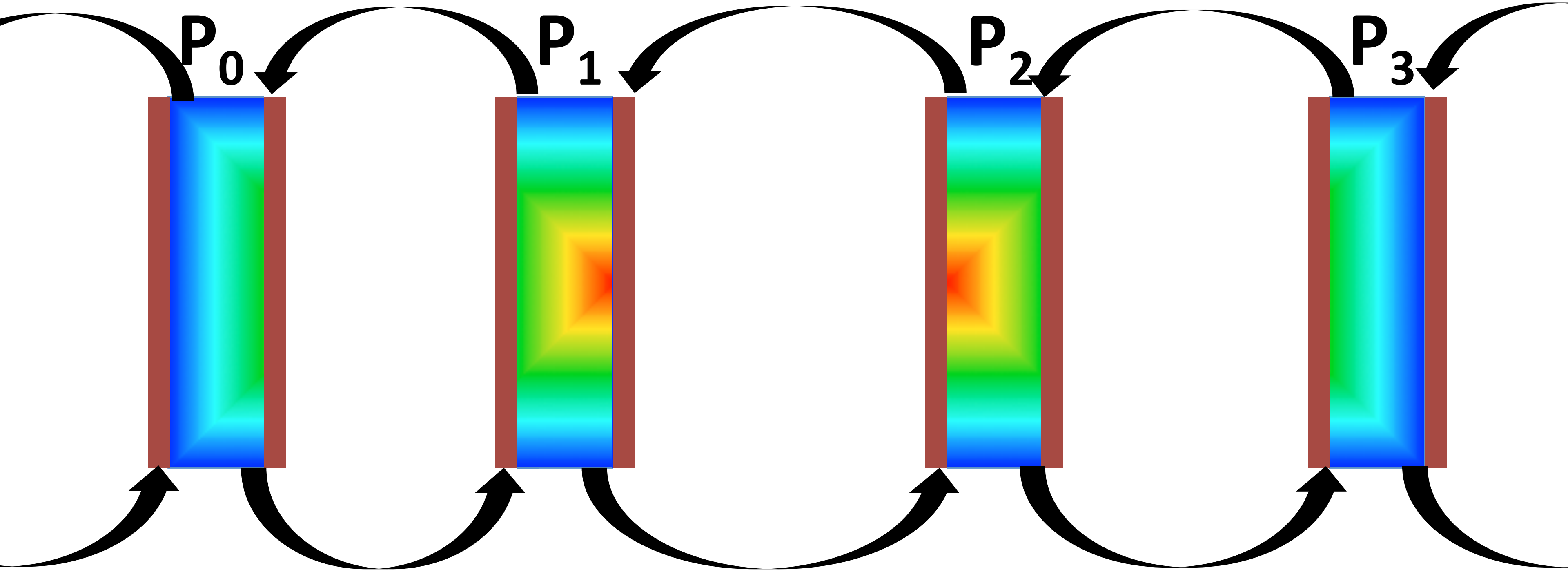
MPI_WAIT(req, status, ierr)

Req (INTEGER) output, identifier of the communications handle

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

Ierr (INTEGER) error code

The Transport Code - Parallel Version



The Transport Code - Parallel Version

- Distributed Data
- Global and Local Indexes
- Ghost Cells Exchange Between Processes
  - Compute Neighbor Processes
- Serialized Output on Process 0 (provided)

SendRecv

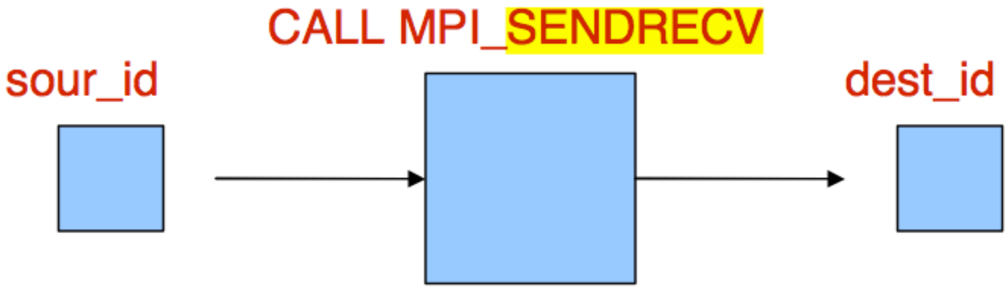
The easiest way to send and receive data without warring about deadlocks

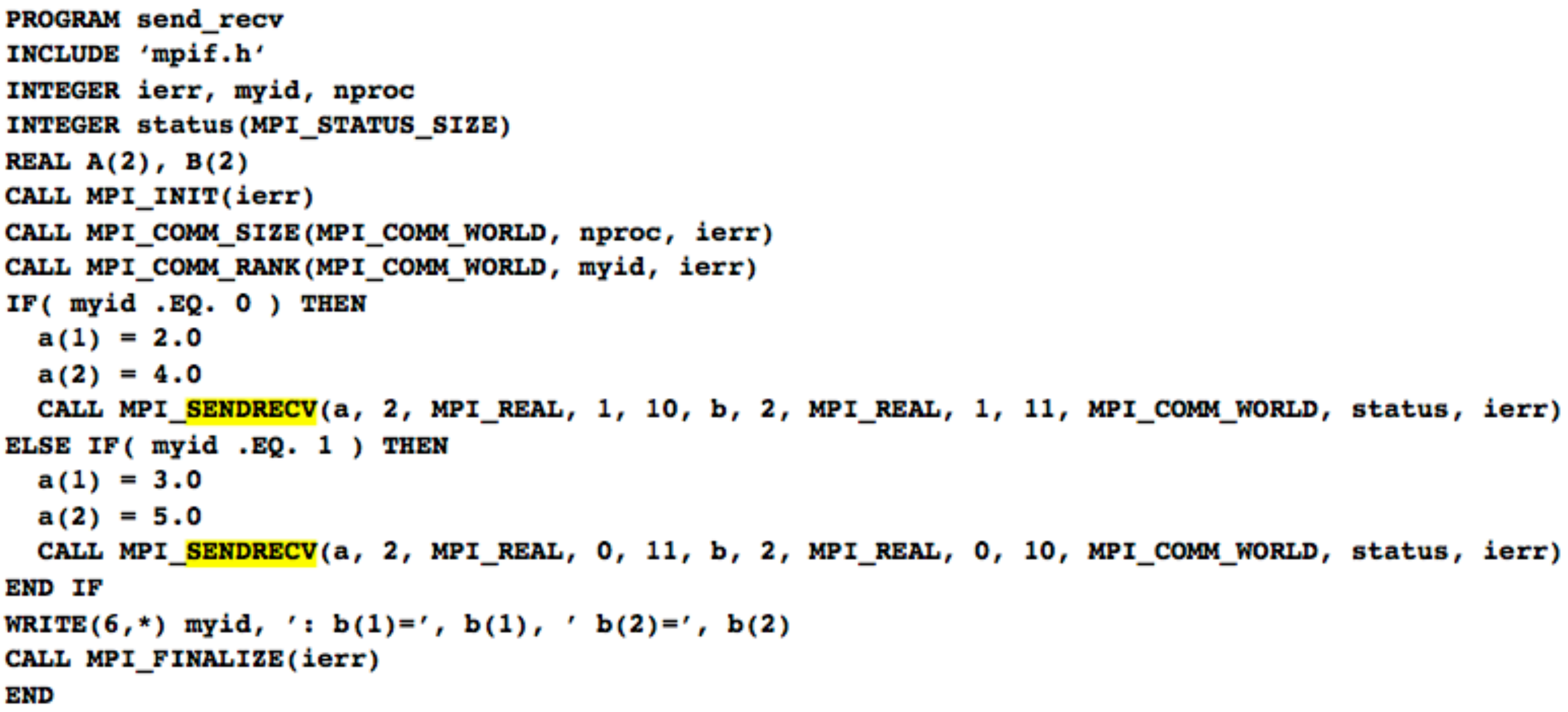
Sender side

Fortran:

```
CALL MPI_SENDRECV(sndbuf, snd_size, snd_type, dest_id, tag,
rcvbuf, rcv_size, rcv_type, sour_id, tag, comm, status, ierr)
```

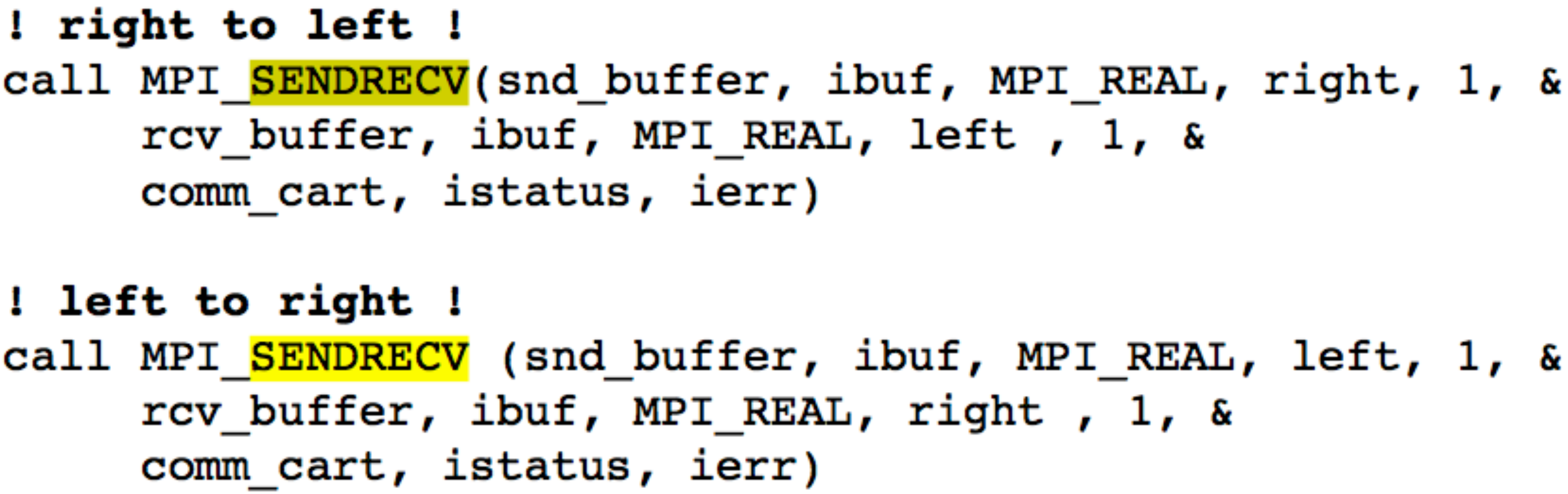
Receiver side





```
PROGRAM send_recv
INCLUDE ’mpif.h’
INTEGER ierr, myid, nproc
INTEGER status(MPI_STATUS_SIZE)
REAL A(2), B(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) = 2.0
  a(2) = 4.0
  CALL MPI_SENDRECV(a, 2, MPI_REAL, 1, 10, b, 2, MPI_REAL, 1, 11, MPI_COMM_WORLD, status, ierr)
ELSE IF( myid .EQ. 1 ) THEN
  a(1) = 3.0
  a(2) = 5.0
  CALL MPI_SENDRECV(a, 2, MPI_REAL, 0, 11, b, 2, MPI_REAL, 0, 10, MPI_COMM_WORLD, status, ierr)
END IF
WRITE(6,*) myid, ’: b(1)=’, b(1), ’ b(2)=’, b(2)
CALL MPI_FINALIZE(ierr)
END
```

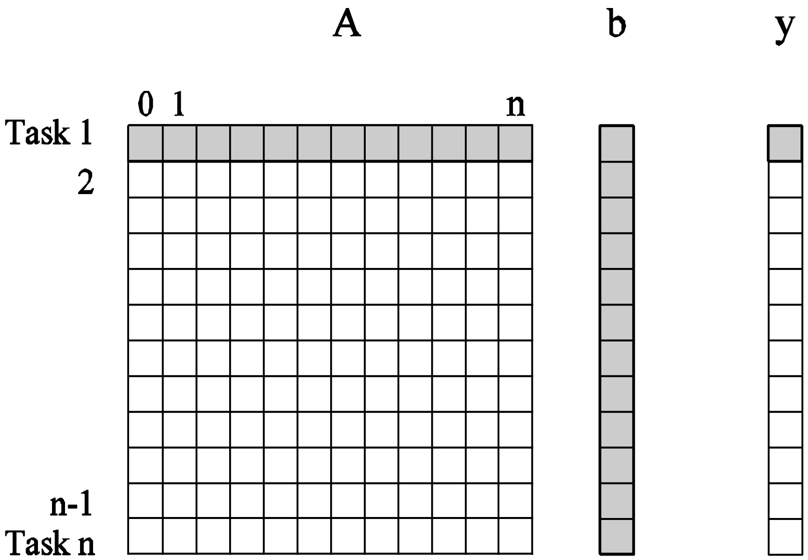
Communication Cycle



```
! right to left !
call MPI_SENDRECV(snd_buffer, ibuf, MPI_REAL, right, 1, &
     rcv_buffer, ibuf, MPI_REAL, left , 1, &
     comm_cart, istatus, ierr)

! left to right !
call MPI_SENDRECV (snd_buffer, ibuf, MPI_REAL, left, 1, &
     rcv_buffer, ibuf, MPI_REAL, right , 1, &
     comm_cart, istatus, ierr)
```

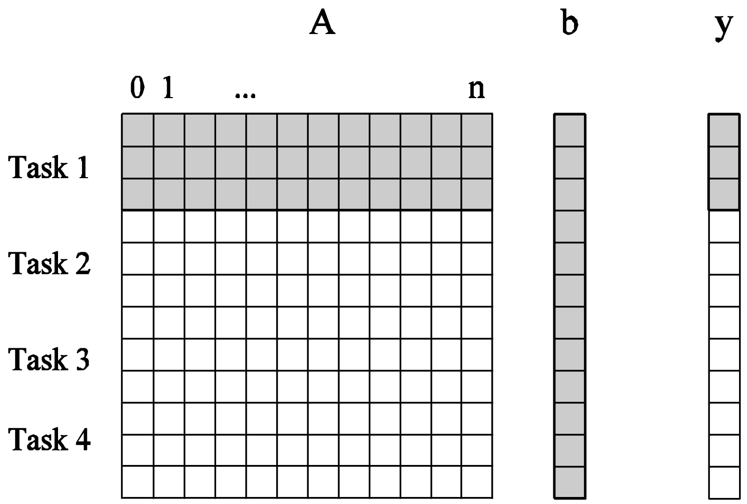
Replication vs Distribution



Under the hypothesis that the vector b is replicated, computation of each element of output vector ***y*** is independent of other elements. Based on this, a dense matrix-vector product can be decomposed into ***n*** tasks.

Granularity of Task Decompositions

- The number of tasks into which a problem is decomposed determines its granularity.
- Decomposition into a large number of tasks results in fine-grained decomposition and that into a small number of tasks results in a coarse grained decomposition.



A coarse grained counterpart to the dense matrix-vector product example. Each task in this example corresponds to the computation of three elements of the result vector.

Granularity, and Communication

- Finest granularity helps for a larger parallelism and to exploit different levels of parallelism
- But in general, if the granularity of a decomposition is finer, the associated overhead (as a ratio of useful work associated with a task) increases.

References

- MPI Documentation
- MPI APIs (list) Documentation

Thanks for your attention!!