TRAINING'S PROGRAM

- Day 1: HPC basics / *Olympe* cluster insight / hands ON
  - 09h00 – 10h15 : HPC systems architecture
    - HPC systems Architecture and Processor Architecture
    - cluster room visit + Coffee Break
  - 10h45 - 12h15 : HPC programming
    - Code tuning
    - Parallel programming MPI, OpenMP
  - 12h15 – 14h00 Lunch Break
  - 14h00 - 15h30 : Atos-Bull intervention / hands-on
    - Olympe Supercomputer presentation+ Batch Scheduler
  - 15h30 – 15h45 : Coffee Break
  - 15h45 – 17h30 : Atos-Bull intervention / hands-on
    - module + compilation
  - 17h30 : end of Day 1

Parallel Programming

➢How to make it happen (Scientific Computation)

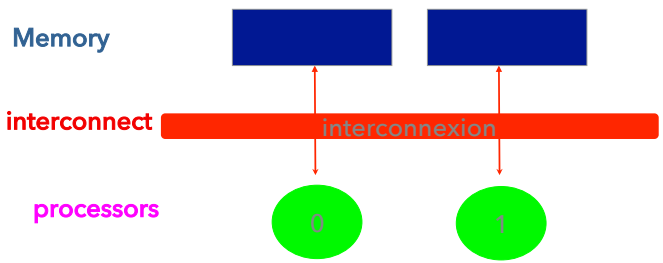
- Explicit way :
- Message passing
- Introduce modification in your program
- Explicitly handle parallelism
- Standard : MPI (it's an API)
- Works on any parallel machine

- implicit/semi-implicit
- Slight indication in your program :

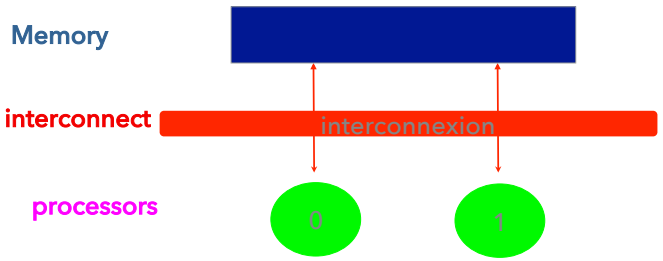
« do it parallel »

- System(OS/compiler) handle part of parallelism
- Standard : OPENMP (now with compilers)
- Only on SMP systems

➢Distributed Memory



➢Shared memory



Parallel Programming : Message passing, Why ?

➢ Why are we exchanging 'messages' (data) ?

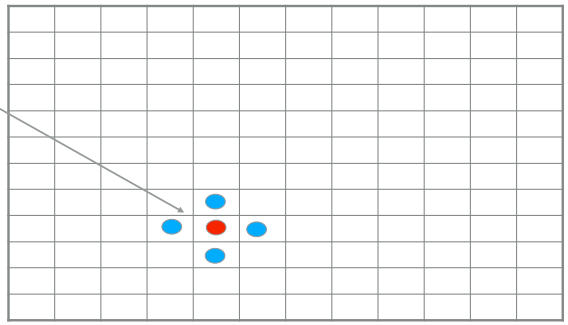
✓Poisson problem

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = f \quad in \quad \Omega = [0,1] \times [0,1] \\ u(x,y) = 0 \quad on \quad \partial\Omega = \text{boundary} \end{cases}$$

Solved Numerically by : Jacobi iterative method on a 2D grid

(Bluid a sequence (U_n), converging to solution)

$$u_{(i,j)}^{n+1}$$



Compute the Red (i,j) point thanks to values of blue points (neighbors)

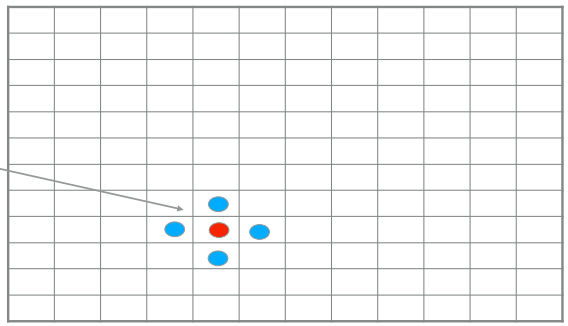
2D Grid - (i,j) : spatial point

Parallel Programming : Message passing, Why ?

➢ Why are we exchanging 'messages' (data) ?

✓Numerical method : Jacobi iterative method

$$u_{(i,j)}^{n+1}$$



For all (i,j) point of 2D Grid

✓Algorithm:

- While error > epsilon

  For all discretized point (i,j) of the domain

  Apply jacobi iteration

  End for all points (i,j)

  Compute new error

  n=n+1

- End while loop

Parallel Programming : Message passing, Why ?

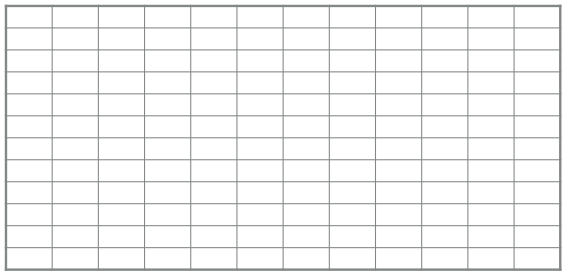
calmip

➢ Why are we exchanging 'messages' (data) ?

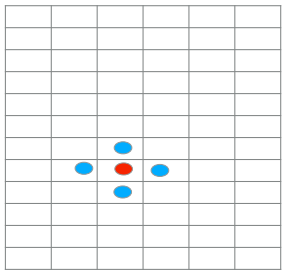
✓Parallel version : for the sake of simplicity : 2 process

✓sub-domain decomposition : each process works a part of the mesh (mesh split in two)

✓ Same algorithm (Jacobi) on each domain

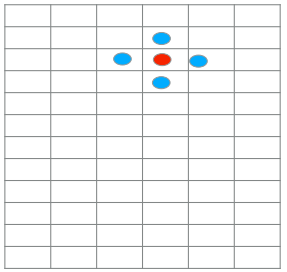


Domain/mesh



Processor 0

Sub-Domain 0

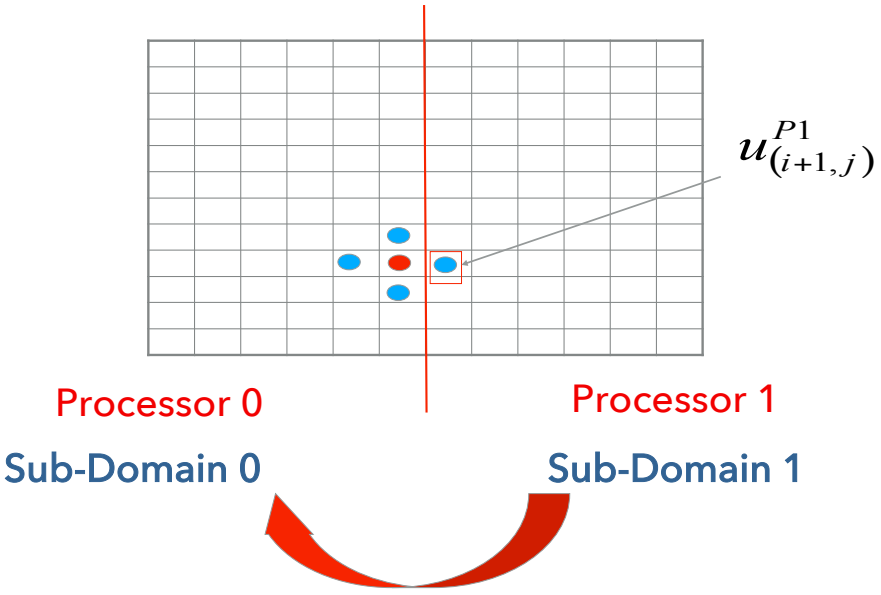


Processor 1

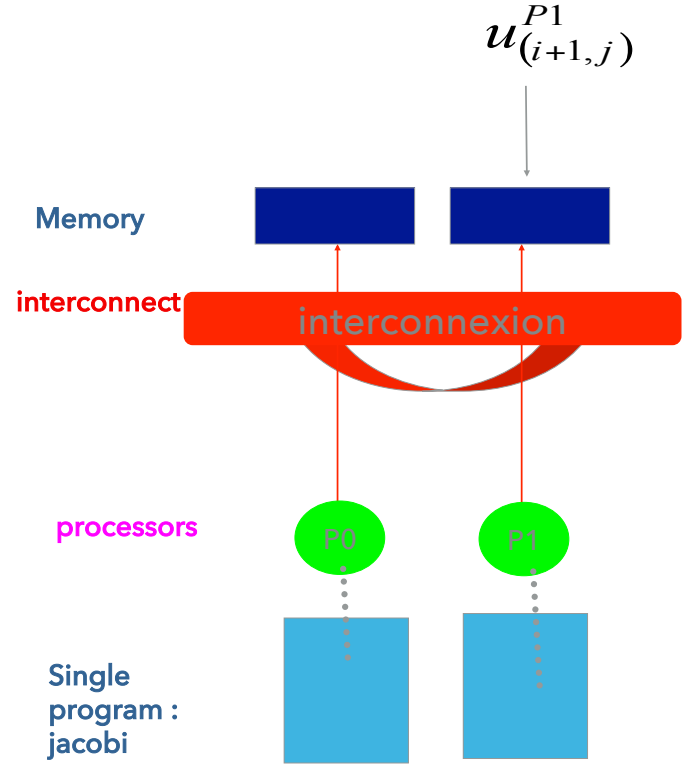
Sub-Domain 1

Parallel Programming : Message passing, Why ?

➢ Why are we exchanging 'messages' (data) ?



Need Data ('message') transfert('passing') : $u^{P1}_{(i+1,j)}$



Parallel Programming : Message Passing, How ?

➢From raw data to Message Passing concept

- Message : data + meta-data passing from the source to the target

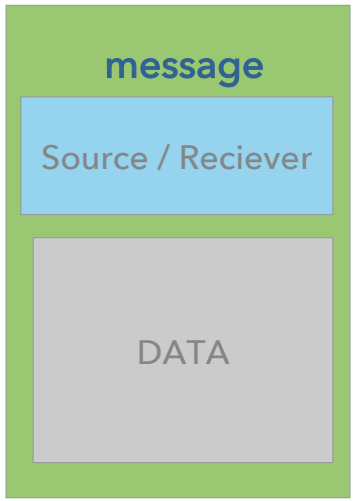
As mail (from: ...., to : ....), fax, ...

- Message :

DATA (variables, arrays (vectors), SIZE,...)

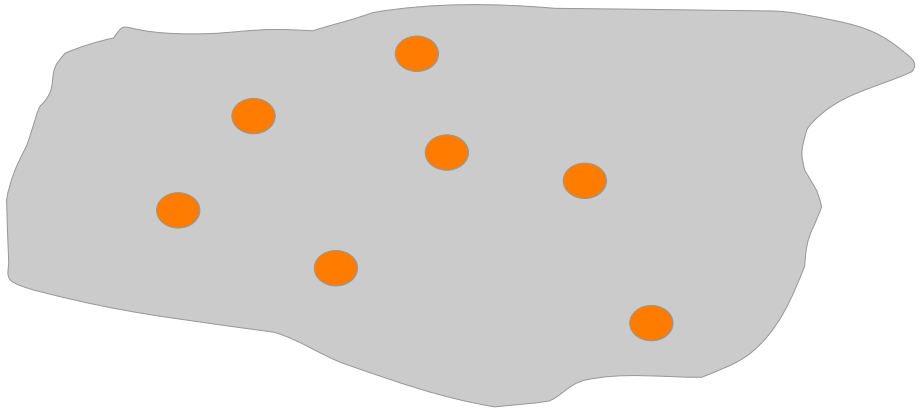
IDENTIFICATION of SOURCE (process)

IDENTIFICATION of RECIEVER (process)

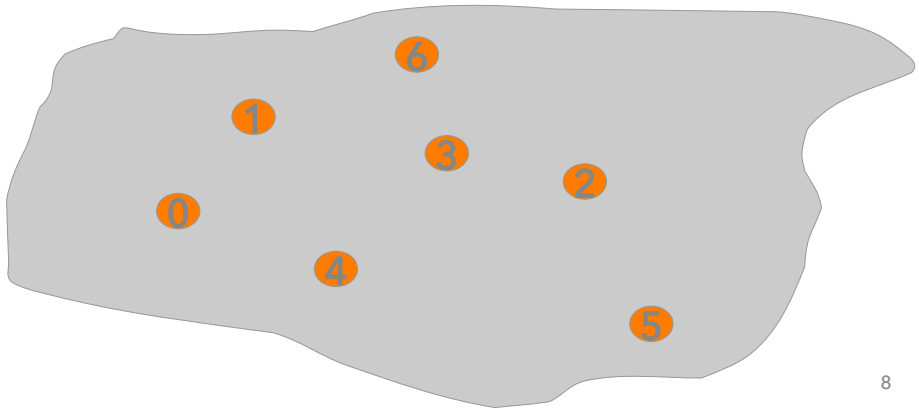


➢Parallel Programming point-of-view : question # 1 we want to answer

➢A bunch of workers in a group



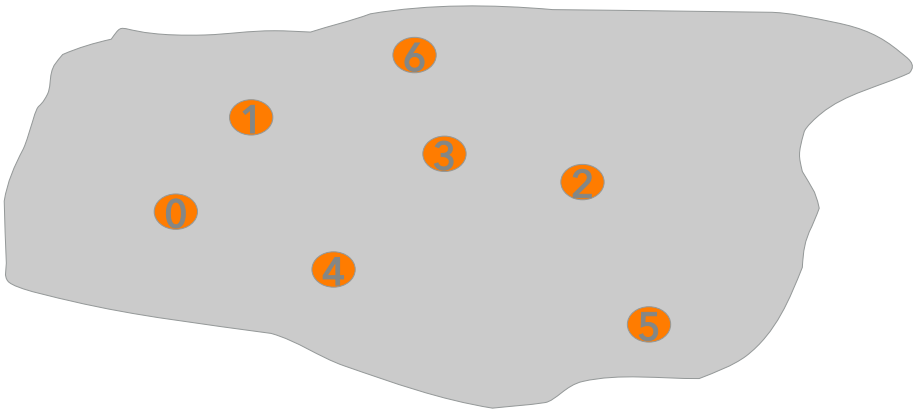
➢ identify each worker



➢Programming point-of-view : question #1 we want to answer

calmip

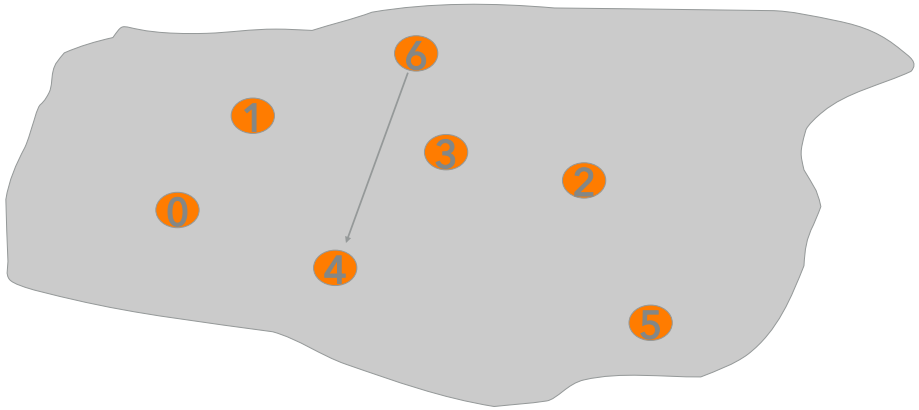
➢Who am'I (in my « world ») ?



➢How Many Are We (in our « world »)?

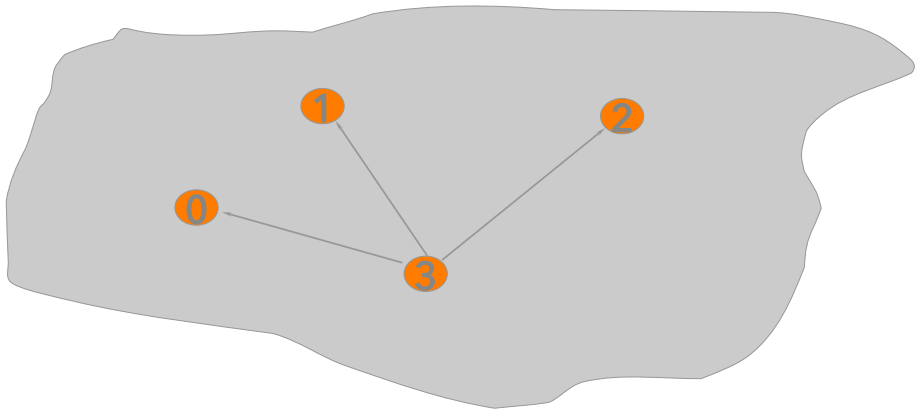
➢Programming point-of-view : question #2 we want to answer

➢Someone want to send something to someone else (in our group/ »world »)?



➢Programming point-of-view : question #2 bis we want to answer

➢Someone want to send something to every one (in our group/« world »)?



Parallel Programming : Message Passing, How ?

➢Message passing concept

MPI : MESSAGE PASSING INTERFACE

- An MPI program : autonomous processes communicate thanks routine from MPI library:

  MPI routine : High level programing

  Low level (implementation) : depend on machine architecture

A lot of MPI routine, need a few to make a parallel program

- Environnement (question #1)
- Point-to-point Communication (question #2)
- collective communication (question #2 bis)

➢MPI : Environnement routine (question #1)

➢Execution of Parallel Program 'Hello_World ' with 7 'workers ':

Program

Hello_World

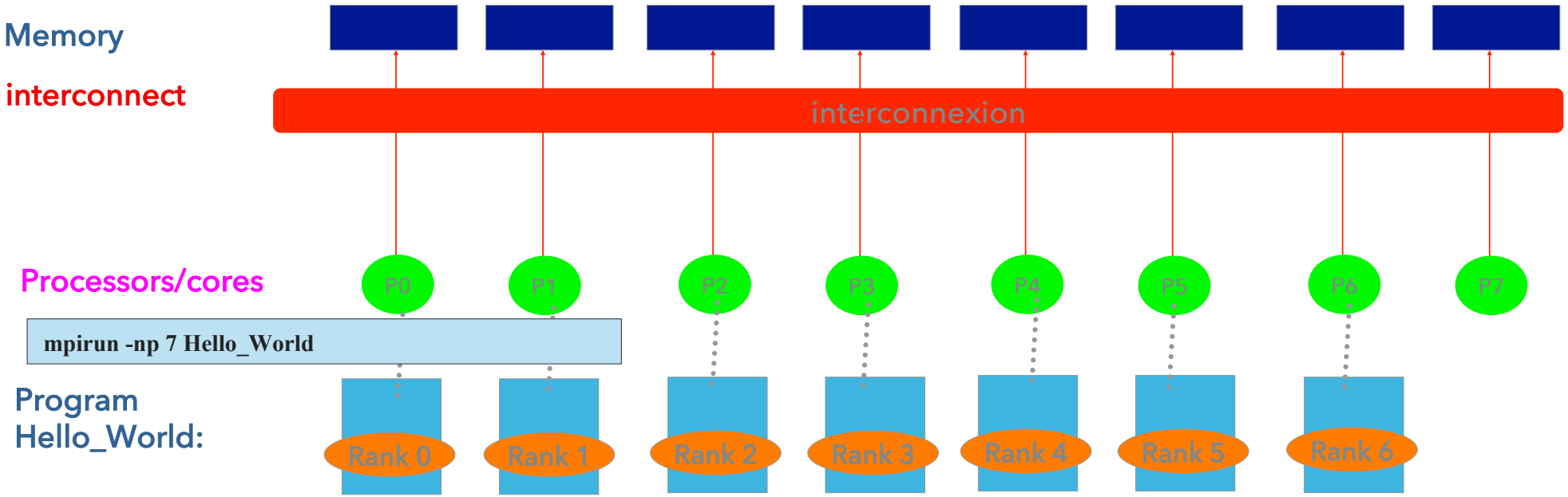
```
Program Hello_World
USE MPI
Integer :: code, nbre,rank
call MPI_INIT(code)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nbre)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
Print*, 'Hello World I am process', rank, 'among', nbre
call MPI_FINALIZE(code)

end program Hello_world
```

```
mpirun -np 7 Hello_World
```

```
Hello World I am process    1  among    7
Hello World I am process    2  among    7
Hello World I am process    4  among    7
Hello World I am process    5  among    7
Hello World I am process    0  among    7
Hello World I am process    3  among    7
Hello World I am process    6  among    7
```

« Appear on Screen »

Parallel Programming (question #1)



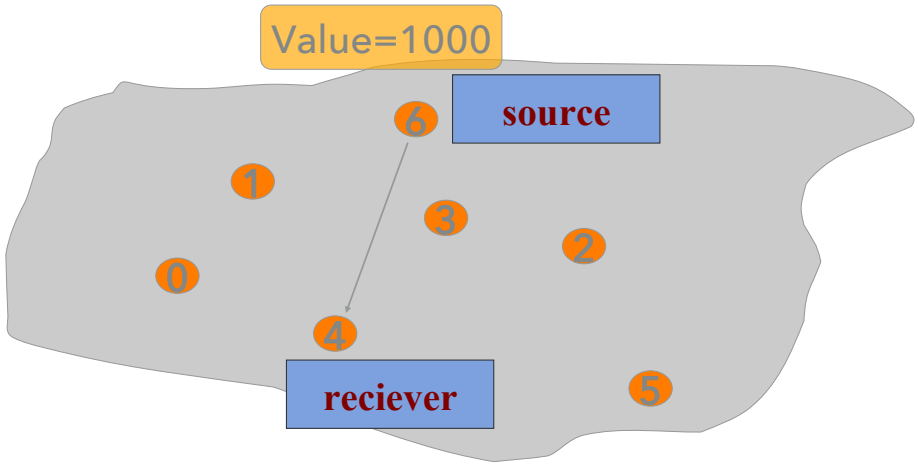
Rank numbering : 0 to 6 (that's make 7 'workers' ....)

Parallel Programming (question #2)

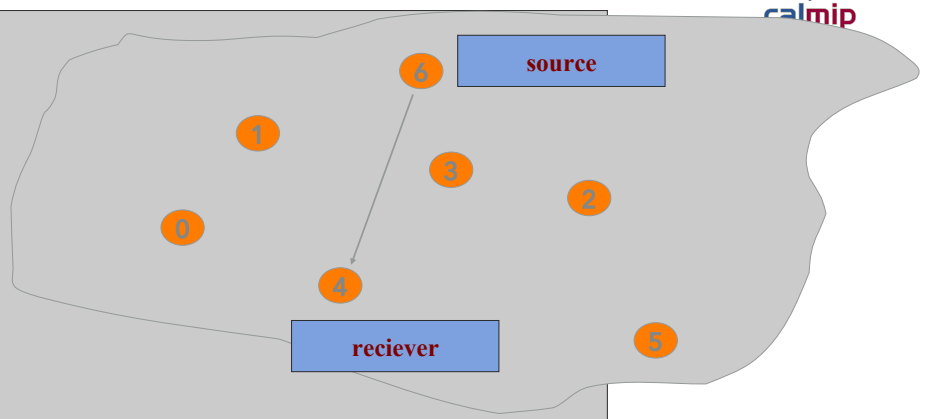
➢ Program Point To Point

Program

Point-to-point



```
program point_to_point
USE MPI
call MPI_INIT(code)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
 if (rank == 6) then
   value=1000
   call MPI_SEND(value,1,MPI_INTEGER,4,...,MPI_COMM_WORLD)
 elseif (rank == 4) then
   call MPI_RECV(value,1,MPI_INTEGER,6,...,MPI_COMM_WORLD)
   print *,'I, process',4, 'I recieved', value,' from process 6.'
 end if
 call MPI_FINALIZE(code)
end program point_to_point
```

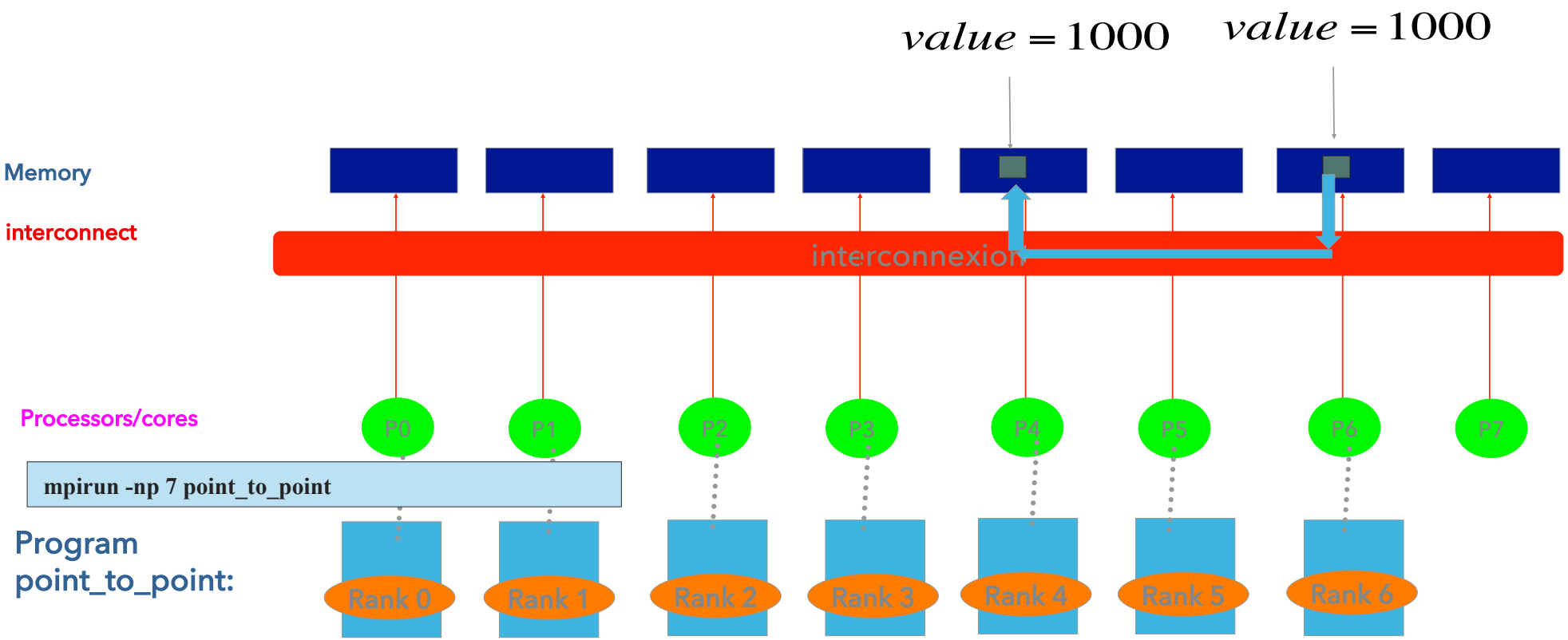



```
> mpirun -np 7 point_to_point
```

```
I, process 4, recieved      1000  from process 6.
```

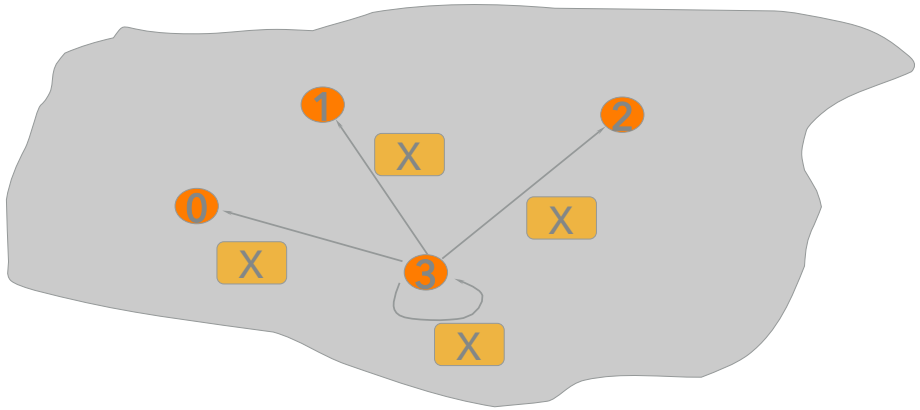
« Appear on Screen »

Parallel Programming (question #2)



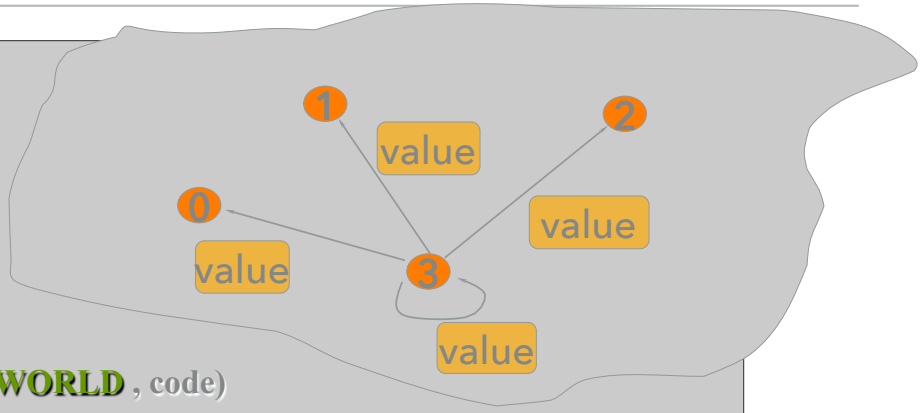
Parallel Programming : question #2 bis

➢ MPI : collectives communications: BROADCAST



Parallel Programming : question #2 bis

```
program broadcast
call MPI_INIT(code)
call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
    if (rank==3) then value=rank+1000
    call MPI_BCAST(value,1,MPI_INTEGER,3 , MPI_COMM_WORLD , code)
print *,'I, process',rank, recieved ', value ,' from process 3'
 call MPI_FINALIZE(code)
end program broadcast
```

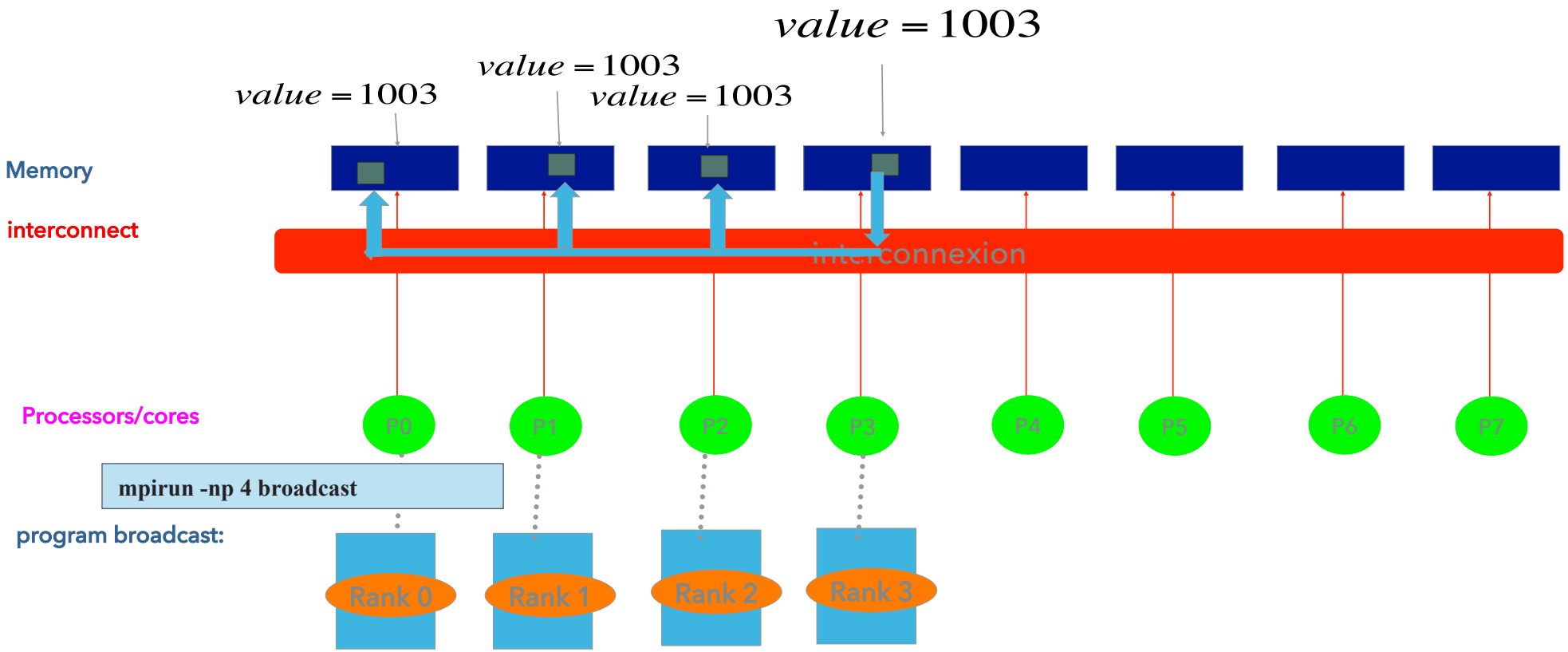



```
mpirun -np 4 broad
```

```
I, process      3 , recieved      1003  from process 3
I, process      0 , recieved      1003  from process 3
I, process      1 , recieved      1003  from process 3
I, process      2 , recieved      1003  from process 3
```

« Appear on Screen »

Parallel Programming (question #2)

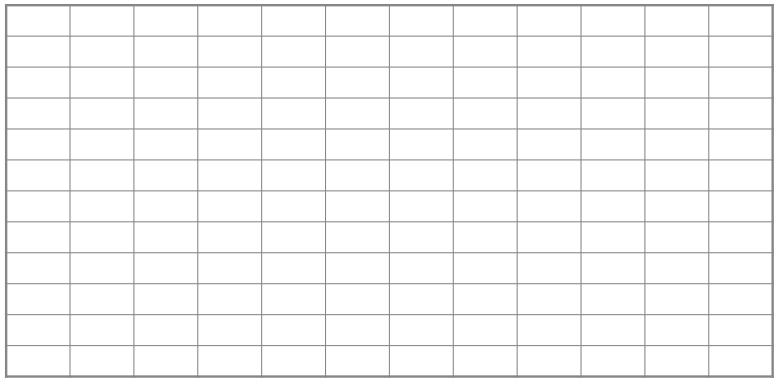


Parallel Programing

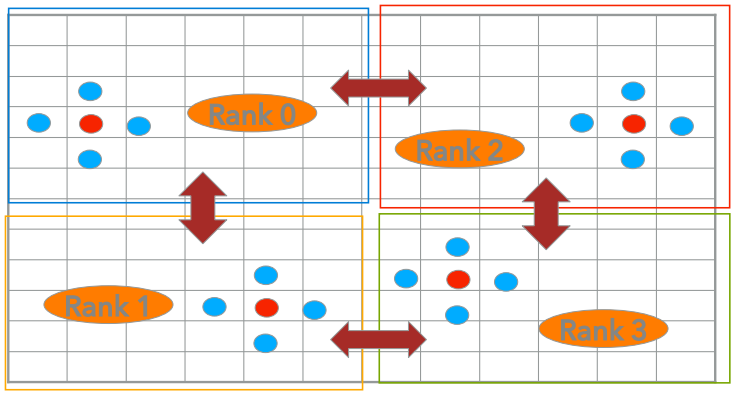
✓sub-domain decomposition : each process works a part of the mesh (mesh split in Four)

✓ Same algorithm (Jacobi) on each domain

Communication ⟷



Domain/mesh



4 sub-domains

+

Communication (MPI_SEND+RECV; BCAST)

Parallel Programming

➢Message Passing interface : MPI

- Programming :

  according to parallel algorithm : roughly 10% to 40% more coding

- MPI evolution : MPI1, MPI2, MPI3
- Works on every kind of parallel architecture : Cluster, UMA, ccNUMA,…
- Different Distribution and implementations :

  OPENMPI, IntelMPI , …

  Some (Vendors) are optimised (low level) according

  Architecture

  Interconnect

Trainings : http://www.idris.fr/ , http://www.cines.fr/ , http://www.prace-ri.eu/

TRAINING'S PROGRAM

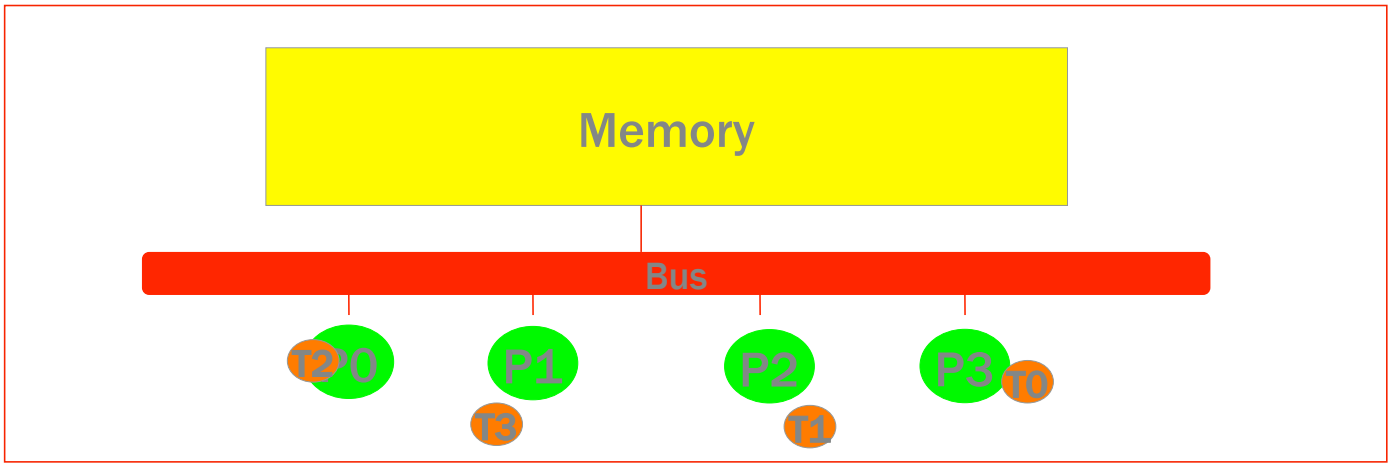
- Day 1: HPC basics / *EOS* cluster insight / hands ON
  - 09h00 – 10h15 : HPC systems architecture
    - HPC systems Architecture and Processor Architecture
    - cluster room visit + Coffee Break
  - 10h45 - 12h15 : HPC programming
    - Code tuning
    - Parallel programming MPI, OpenMP
  - 12h15 – 14h00 Lunch Break
  - 14h00 - 15h30 : Atos-Bull intervention / hands-on
    - EOS Supercomputer presentation+ Batch Scheduler
  - 15h30 – 15h45 : Coffee Break
  - 15h45 – 17h30 : Atos-Bull intervention / hands-on
    - module + compilation
  - 17h30 : end of Day 1

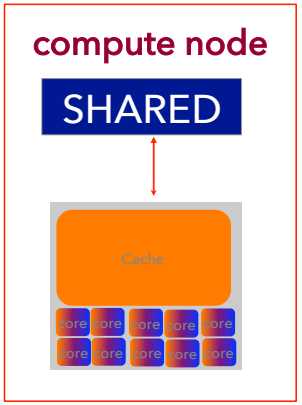
Parallel Programming : Shared Memory

➢ **Shared Memory programming (Multi-Threading) : OpenMP (Open Multi Processing)**

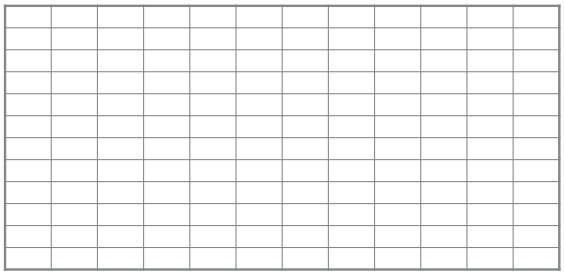
➢ OpenMP : Standard

➢ Woks ONLY on Shared Memory Systems : SMP, ccNUMA



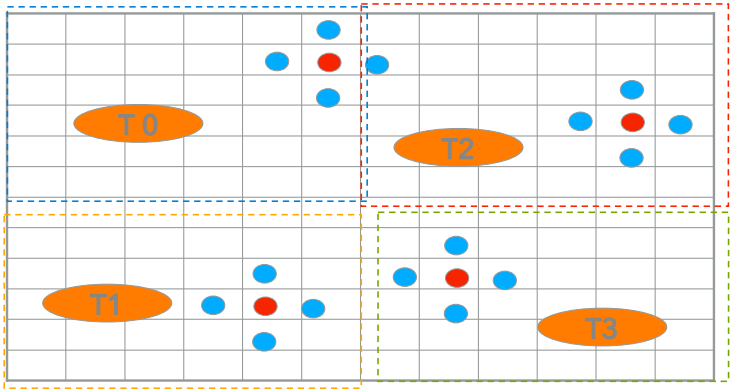


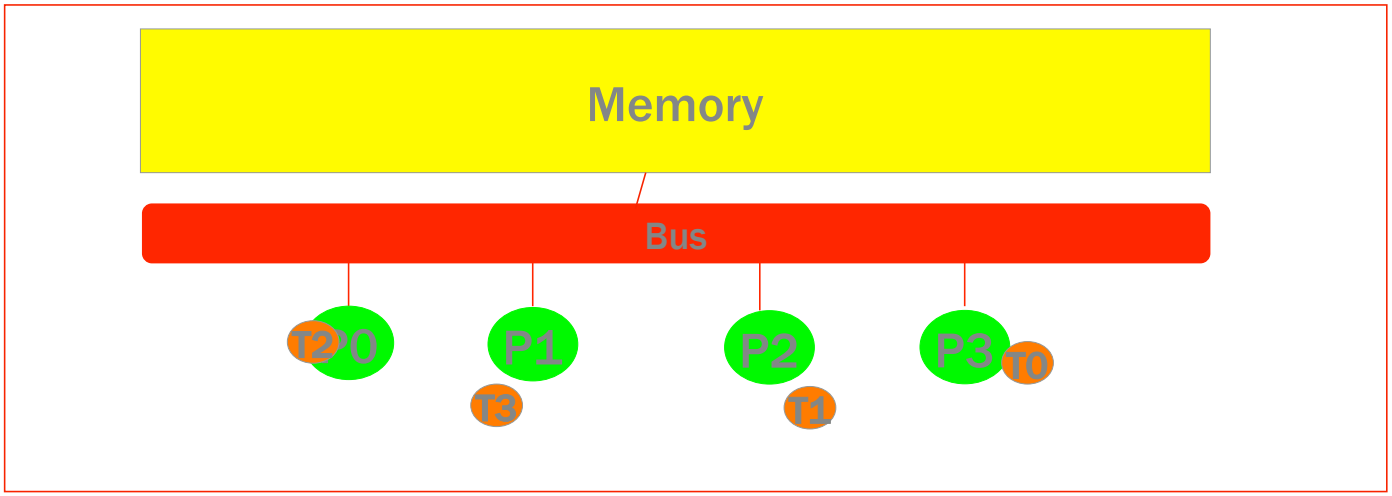
Parallel Programming : Shared memory



Domain/mesh

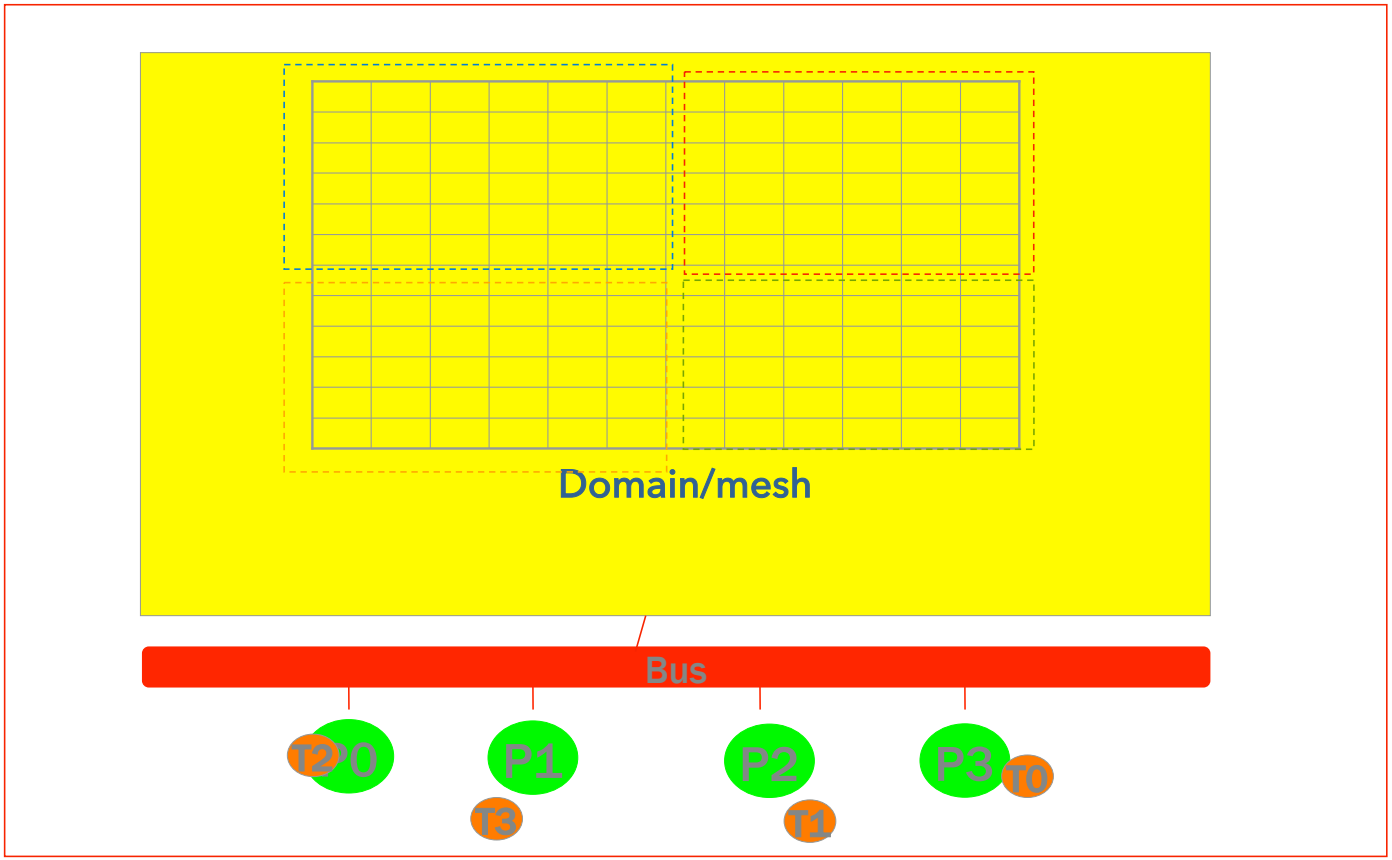
Split Work, NO explicit Communication'





Parallel Programming : Shared memory

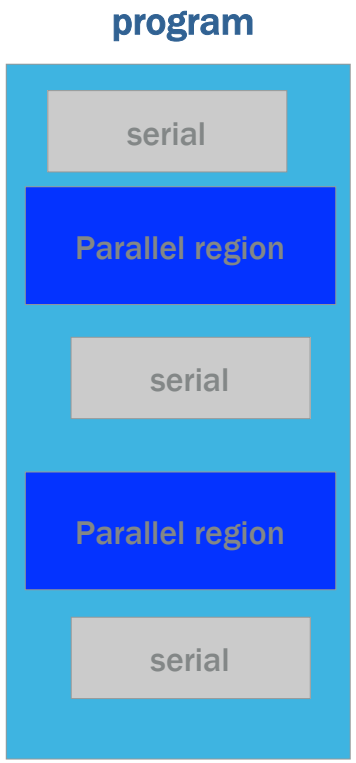
The Whole Mesh in the Shared Memory

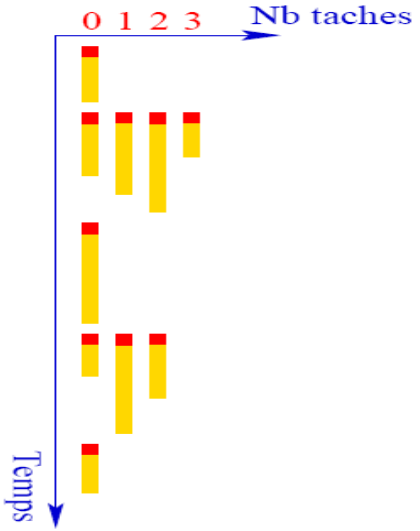


Parallel Programming : OpenMP

➢ **OpenMP :**

- ➢ Identify parallel regions
- ➢ Share Work to do (multi-threading)
  - Loop Iteration
  - Different program's section
  - …..





Parallel Programming : OpenMP

➢ Create a parallel region

```
#include <stdio.h>
#include <omp.h>
int main()
{
      float a;
      a = 13450. ;

      #pragma omp parallel
      {
      printf("a vaut : %f \n",a);
      }
 return 0;
}
```

Compile and set

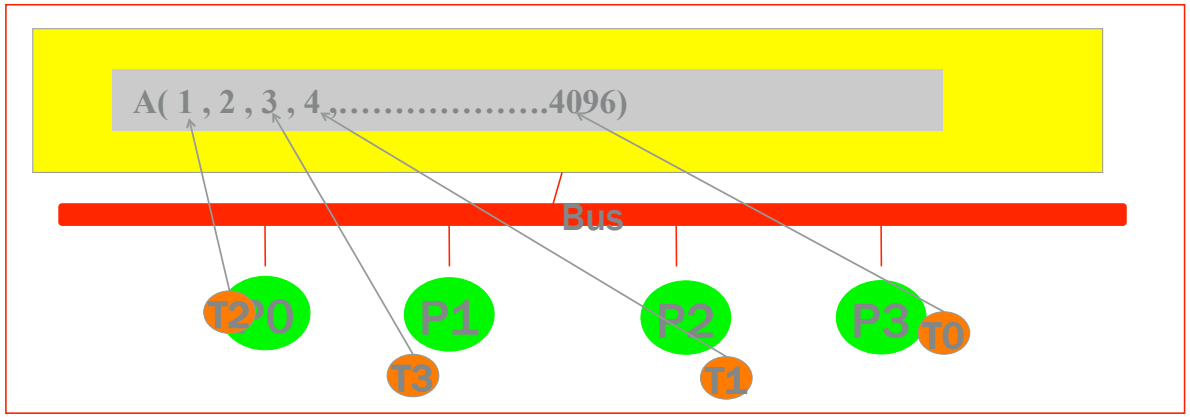
```
> icc -openmp para.c
➢ export OMP_NUM_THREADS=4
➢ ./a.out
```

« Appear on Screen »

```
a vaut : 13450
a vaut : 13450
a vaut : 13450
a vaut : 13450
```

➢ Share work : Loop parallelization

```
#include <stdio.h>
#include <omp.h>
#define N 4096
int main()
{
        float a[N];
        int i;


        #pragma omp parallel for
        for (i=0; i<N; i++) {
         a[i] = 13000. + (float) i ;
        }
         return 0;
}
```



Parallel Programming : OpenMP

calmip

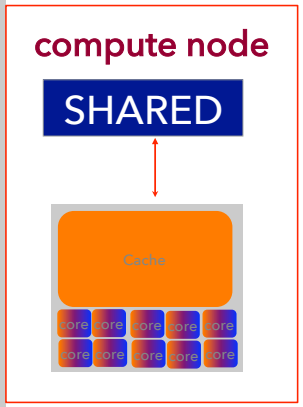
- **Shared memory/ OpenMP :**
  - **Much easier**
  - **… expect less gain**
  - **Automatic Implicit parallelization**
  - **Cannot make something not parallel, parallel**
  - **OpenMP evolution : different norms (4.5, 5.0); with compilers (gnu, Intel, pgi)**
  - **only on Shared Memory Systems**
  - **Works for GPU too**

Trainings : http://www.idris.fr/ , http://www.cines.fr/ , http://www.prace-ri.eu/

Parallel Drivectives for GPU : OpenMP, OpenACC

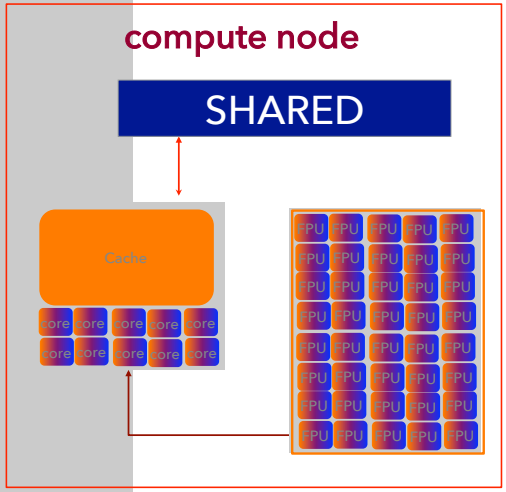
Loop parallelization on CPU

```
#include <stdio.h>
#include <omp.h>
#define N 4096
int main()
{
      float a[N];
      int i;


      #pragma omp parallel for
      for (i=0; i<N; i++) {
       a[i] = 13000. + (float) i ;
      }
       return 0;
}
```



Loop parallelization on GPU

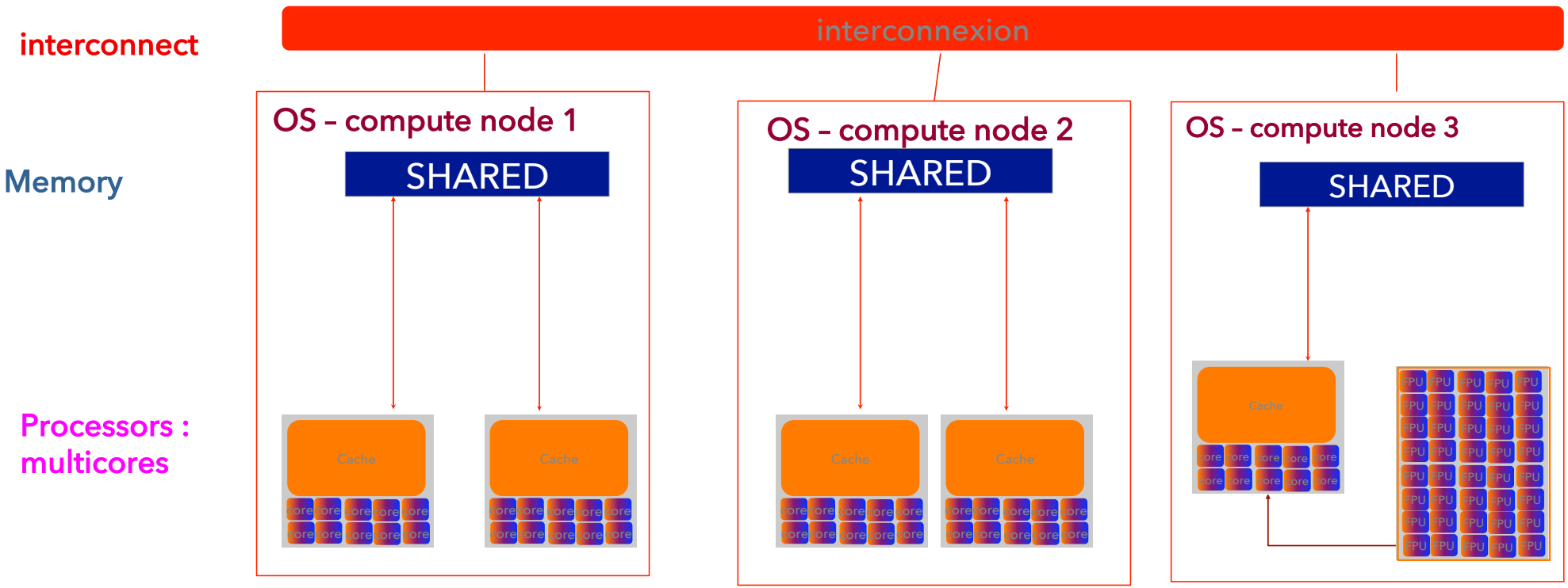
```
#include <stdio.h>
#include <omp.h>
#define N 4096
int main()
{
      float a[N];
      int i;

 #pragma target
{
      #pragma omp parallel for
      for (i=0; i<N; i++) {
       a[i] = 13000. + (float) i ;
      }
}
       return 0;
}
```



Data exchange CPU <-> GPU

OpenMP & OpenACC for GPU : http://www.idris.fr/formations/openacc/fiche-openacc.html

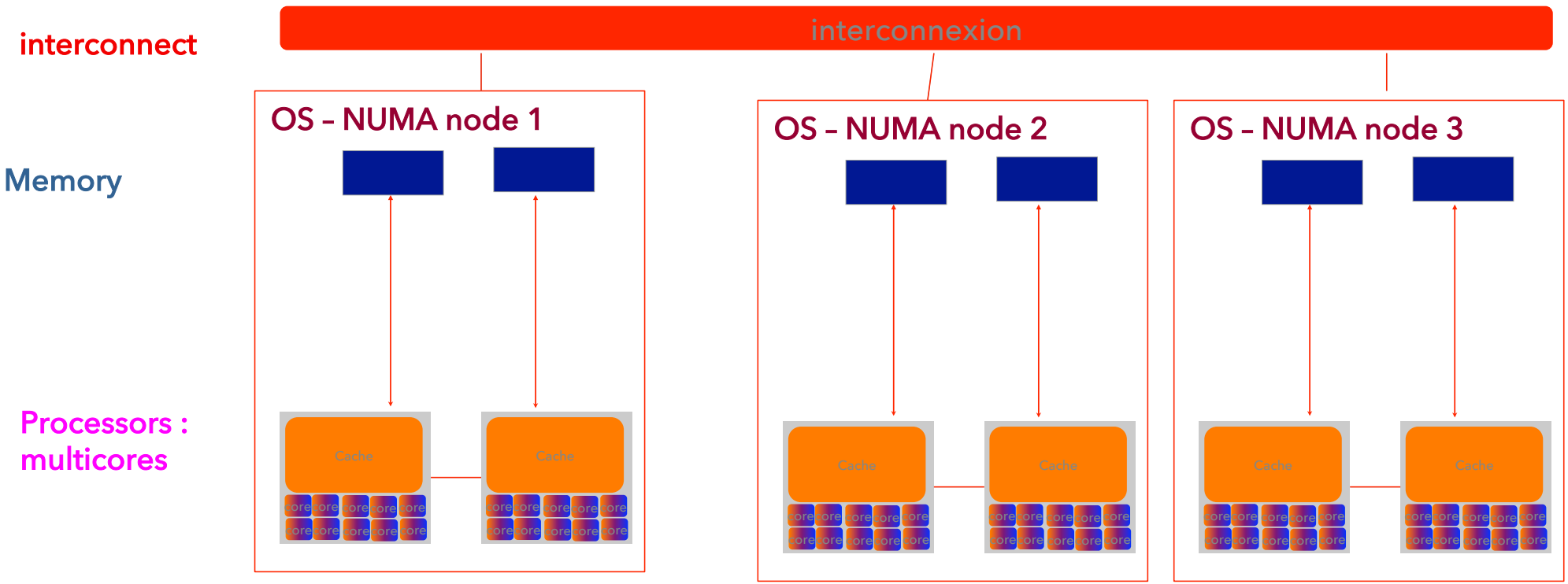
Hybrid Architecture / Hybrid Programming



TRAINING'S PROGRAM

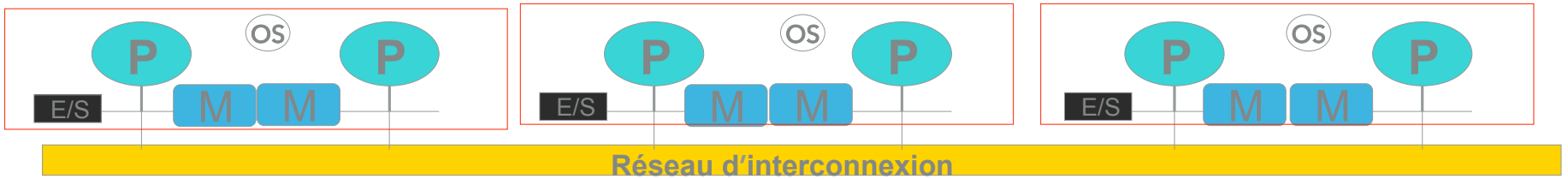
- Day 1: HPC basics / *EOS* cluster insight / hands ON
  - 09h00 – 10h15 : HPC systems architecture
    - HPC systems Architecture and Processor Architecture
    - cluster room visit + Coffee Break
  - 10h45 - 12h15 : HPC programming
    - Code tuning
    - Parallel programming MPI, OpenMP
  - 12h15 – 14h00 Lunch Break
  - 14h00 - 15h30 : Atos-Bull intervention / hands-on
    - Olympe Supercomputer presentation+ Batch Scheduler
  - 15h30 – 15h45 : Coffee Break
  - 15h45 – 17h30 : Atos-Bull intervention / hands-on
    - module + compilation
  - 17h30 : end of Day 1

Hybrid Architecture / Hybrid Programming



Programmation hybride : MPI+OpenMP

- OpenMP ET MPI
- OpenMP : machine à mémoire partagée uniquement
- Couplage MPI-OpenMP : cluster de machine à mémoire partagée



- Premier jour
  - Matin : Concepts fondamentaux
    - Introduction à l'Architecture des systèmes HPC
      - *Calcul Intensif et Panorama des Systèmes*
      - *Architecture Processeurs/ Accélérateurs*
      - *Présentation système de Calcul CALMIP : EOS*
      - *Visite salle Machine*
    - Introduction programmation sur les systèmes HPC
      - *Optimisation de codes*
      - *Programmation Parallèle*
        - » *Echange de Message MPI*
        - » *Mémoire partagée : OpenMP*
        - » *Conclusion*

Parallel Programming

$$Speedup = \frac{T_{CodeSequentiel}}{T_{CodeParallel}}$$

•**Good if:**

$$speedup \approx number\ of\ cores$$

✓Parallel computing time :

$$T_{Parallel} = T_{Comput} + T_{Overhead}$$

$$Nbre.\Pr oc \uparrow \Rightarrow T_{Calcul} \downarrow + T_{Overhead} \uparrow$$

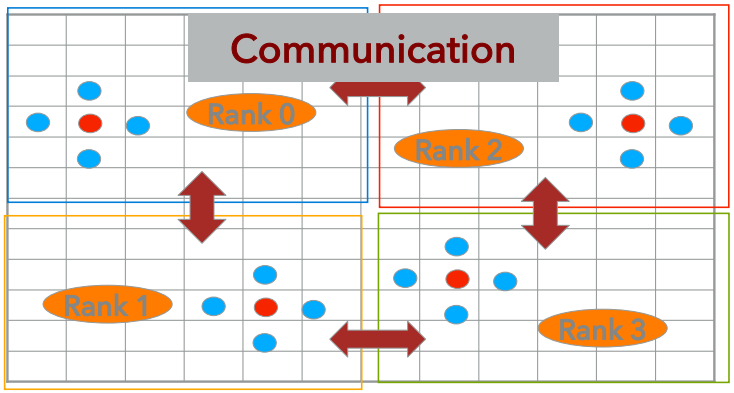
Processors/ cores

Interconnect

✓MPI : Overhead = Communication

✓OpenMP : Overhead = thread handler, binding, cache Coherency, etc ….

Parallel Programming



MPI_SEND / MPI_RECV

MPI_REDUCE / BCAST

✓Each process running in parallel

- While global error > epsilon

  For all discretized point (i,j) of the sub-domain

  (if necessary) send data to neighbor

  (if necessary) recieve data from neigbor

  Apply jacobi iteration

  End for all points (i,j) of sub-domain

Compute new global error

n=n+1

- End while loop