Utilisation de MPI avec Python

Loïc Gouarin

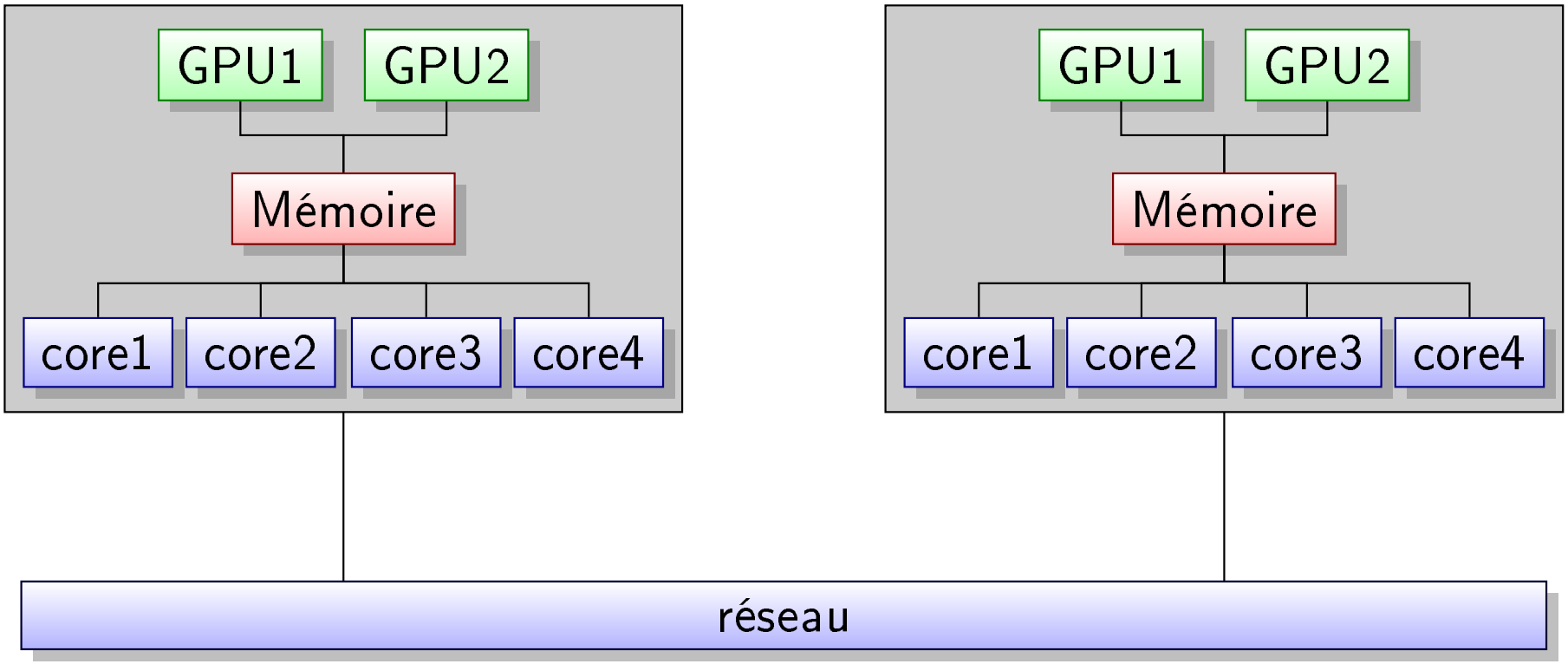
Laboratoire de mathématiques d'Orsay

9 décembre 2010

Plan

Plan

Comment paralléliser son code ?



Quels sont les outils pour paralléliser son code ?

1. Threads
2. PVM
   - pypvm
   - pynpvm
3. MPI
   - pyMPI
   - mpi4py
4. GPU
   - PyCUDA
   - PyOpenCL

Plan

MPI : Message Passing Interface

- conçue en 1993,
- norme définissant une bibliothèque de fonctions, utilisable avec les langages C et Fortran,
- permet d'exploiter des ordinateurs distants ou multiprocesseurs par passage de messages.

MPI : Message Passing Interface

Fonctionnalités de MPI-1

- nombre de processus, numéro du processus,...
- communications point à point,
- communications collectives,
- communicateurs,
- types dérivées,
- topologies.

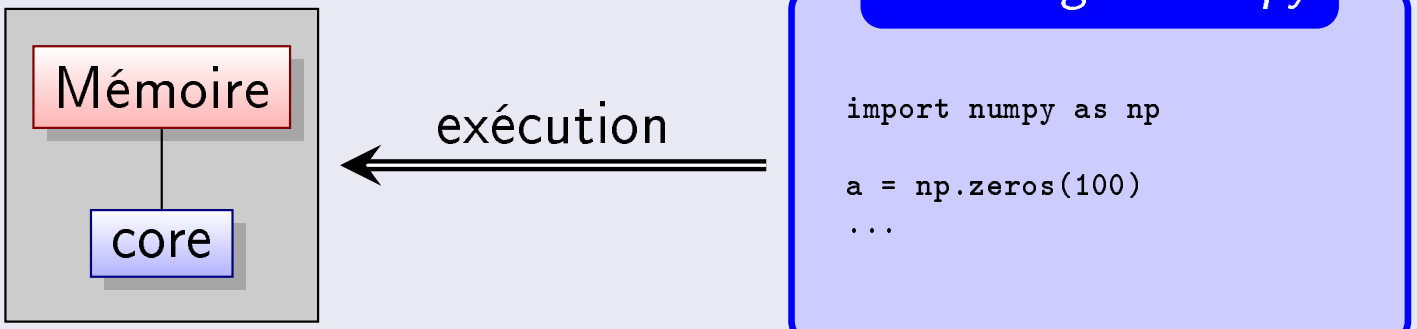
MPI : Message Passing Interface

Fonctionnalités de MPI-2

- gestion dynamique des processus,
- I/O parallèle,
- interfaçage avec Fortran95 et C++,
- extension des communications collectives aux intercommunicateurs,
- communications de mémoire à mémoire,
- ...

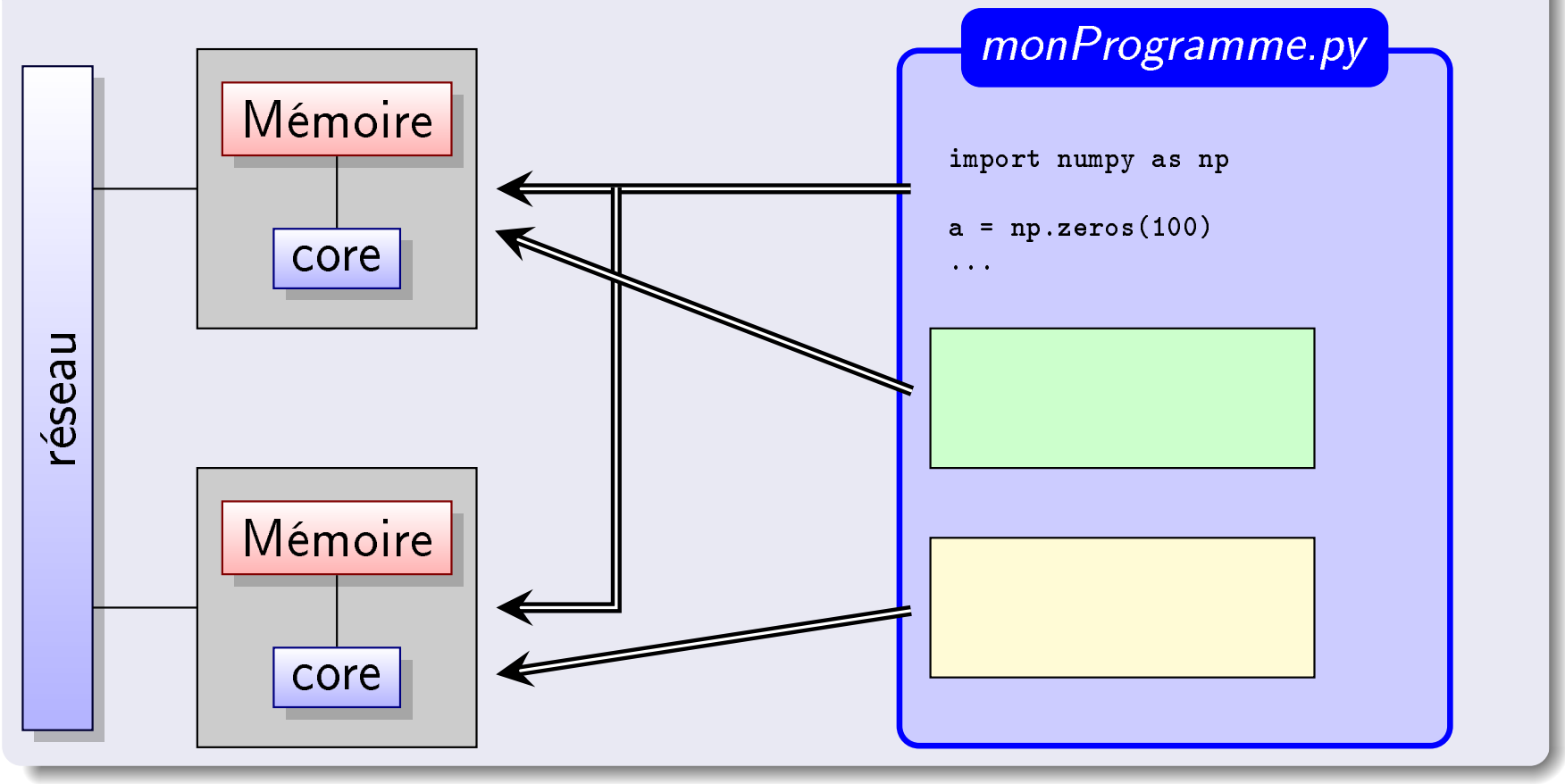
MPI : Message Passing Interface

Programme séquentiel



Principe de MPI

SPMD : Single Program Multiple Data



PyMPI

- projet inititié en 2002 par Patrick Miller
- interfaçage de la bibliothèque MPI-1 directement en API C
- ce module gère
  - nombre de processus, numéro du processus,...
  - communications point à point,
  - communications collectives,
  - création de sous-communicateurs.

mpi4py

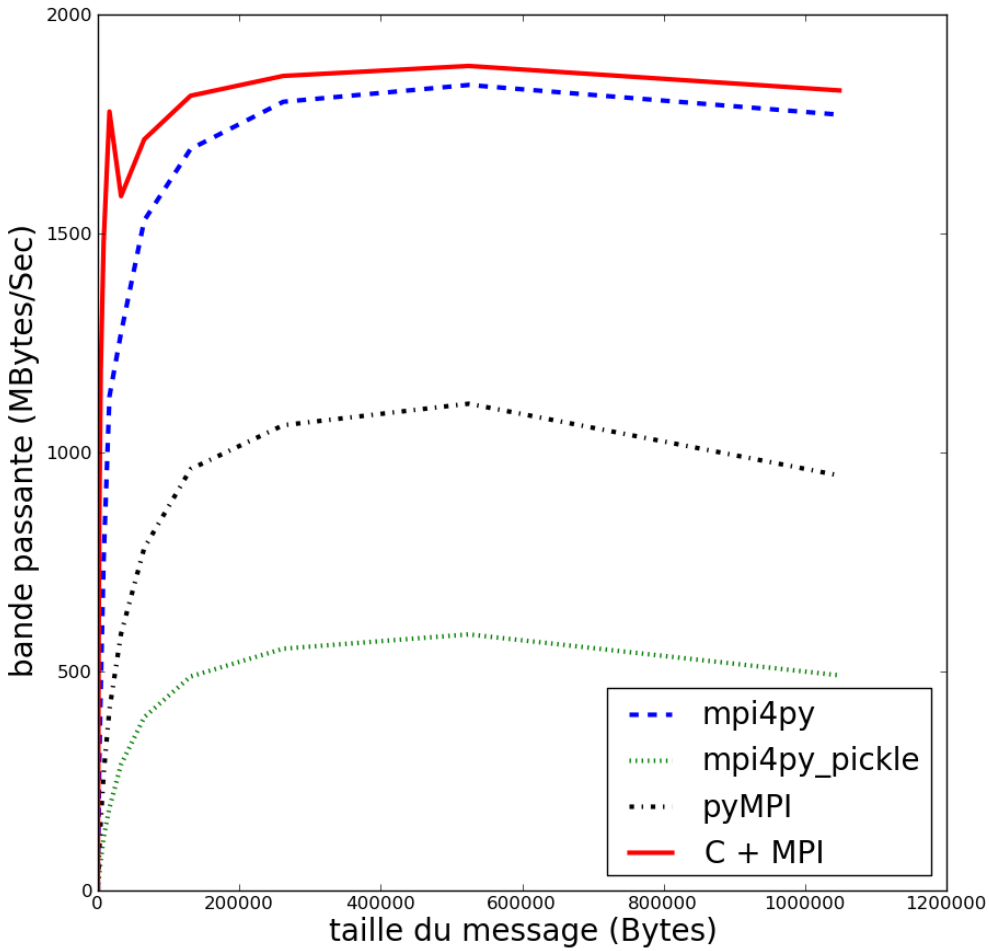
- projet inititié en 2006 par Lissandro Dalcin
- interfaçage de la bibliothèque MPI-1/2 avec Swig ($version < 1$)
- interfaçage de la bibliothèque MPI-1/2 avec Cython ($version >= 1$)
- ce module gère toutes les fonctionnalités que l'on peut trouver dans MPI-1/2

Tests de performance

Ping pong

- Chaque processus dispose d'un tableau de double.
- Envois alternés de paquets de données entre 2 processus.
- Les paquets sont de plus en plus grands.
- On regarde combien de temps il faut pour recevoir ces paquets.

Tests de performance



Un petit mot sur pickle

Le module **pickle** permet de convertir n'importe quel objet complexe Python en suites d'octets (*sérialisation*).

Ce flux peut

- être sauvegardé,
- être transmis via le réseau.

On peut ensuite le reconstruire (*désérialisation*).

Le module **cPickle** est une implémentation en C plus rapide que **pickle**.

Un petit mot sur pickle

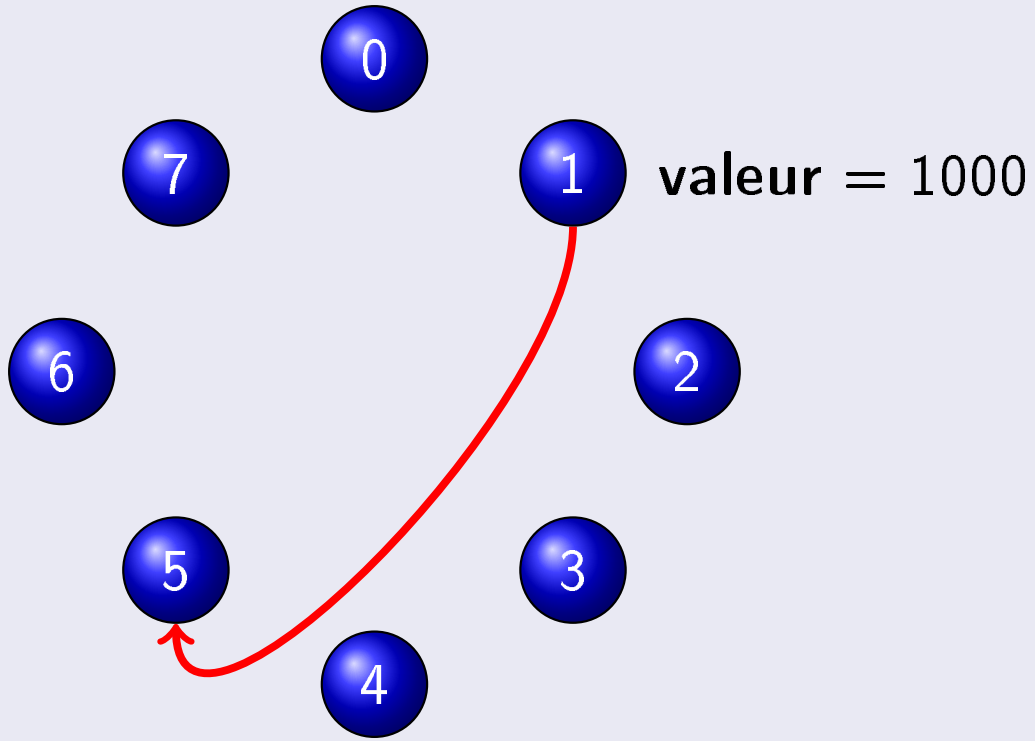
Exemple

```
import cPickle
import numpy as np

a = np.linspace(0., 1., 100)
b = "une chaine"

fichier = open("pickleData", "w")
cPickle.dump([a, b], fichier, 0)
```

Plan

Communication point à point bloquante

Exemple



Exemple Fortran

```
program point_a_point
  implicit none
  include 'mpif.h'
  integer, dimension(MPI_STATUS_SIZE) :: statut
  integer, parameter                  :: etiquette=100
  integer                             :: rang,valeur,code

  call MPI_INIT(code)

  call MPI_COMM_RANK(MPI_COMM_WORLD,rang,code)

  if (rang == 1) then
     valeur=1000
     call MPI_SEND(valeur,1,MPI_INTEGER,5,etiquette,MPI_COMM_WORLD,code)
  elseif (rang == 5) then
     call MPI_RECV(valeur,1,MPI_INTEGER,1,etiquette,MPI_COMM_WORLD,statut,code)
     print *,'Moi, processus 5, j''ai reçu ',valeur,' du processus 1.'
  end if

  call MPI_FINALIZE(code)
```

Même exemple avec Python

```
import mpi4py.MPI as mpi

rang = mpi.COMM_WORLD.rank

if rang == 1:
    valeur = 1000
    mpi.COMM_WORLD.send(valeur, dest = 5)
elif rang == 5:
    valeur = mpi.COMM_WORLD.recv(source = 1)
    print "Moi, Processus 5, j'ai recu", valeur, "du processus 1."
```

Un exemple un peu plus complexe

```
class point:
    def __init__(self, num, x, y):
        self.num = num
        self.x = x
        self.y = y

    def __str__(self):
        s = "coordonnees du point " + str(self.num) + " :\n"
        s += "x : " + str(self.x) + " , y : " + str(self.y) + "\n"
        return s
```

Un exemple un peu plus complexe (suite)

```
import mpi4py.MPI as mpi
from numpy import array
from point import point

rank = mpi.COMM_WORLD.rank

if rank == 0:
    sendValues = [point(1, 2., 4.5), array([3, 4, 8]), \
                          {1:'un', 2:'deux', 3:'trois'}]
    mpi.COMM_WORLD.send(sendValues, dest = 1)
else:
    recvValues = mpi.COMM_WORLD.recv(source = 0)
    for v in recvValues:
        print v
```

Envoi d'un tableau Numpy

```
import mpi4py.MPI as mpi
import numpy as np

rank = mpi.COMM_WORLD.rank
n = 10
if rank == 0:
    sendarray = np.linspace(0., 1., n)
    mpi.COMM_WORLD.Send([sendarray, mpi.DOUBLE], dest = 1)
else:
    recvarray = np.empty(n)
    mpi.COMM_WORLD.Recv([recvarray, mpi.DOUBLE], source = 0)
    print recvarray
```

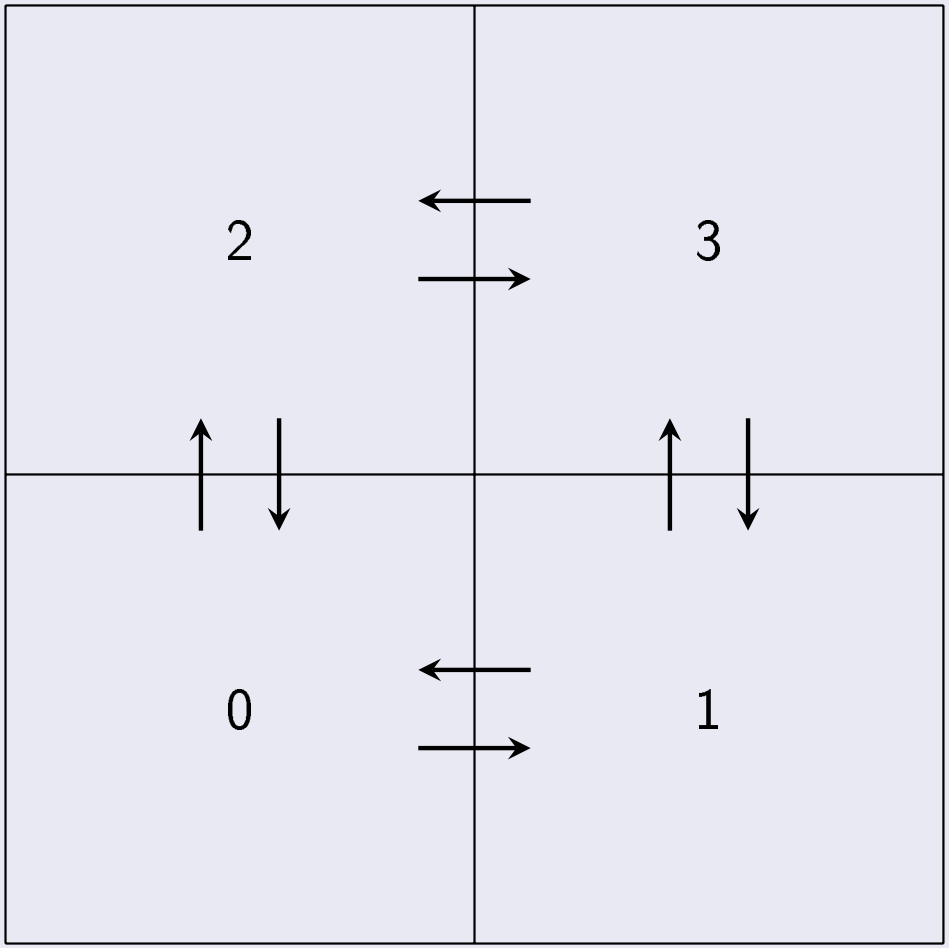
Résumé

Dans le module mpi4py.MPI

- *COMM_WORLD* : communicateur par défaut (englobe tous les processus lancés)
- *COMM_WORLD.size* : nombre de processus
- *COMM_WORLD.rank* : rang du processus
- *COMM_WORLD.send, COMM_WORLD.recv* : envoi et réception de messages via cPickle
- *COMM_WORLD.Send, COMM_WORLD.Recv* : envoi et réception de tableaux Numpy
- *INTEGER, FLOAT, DOUBLE, COMPLEX, ...* : types des données des tableaux Numpy envoyés et reçus

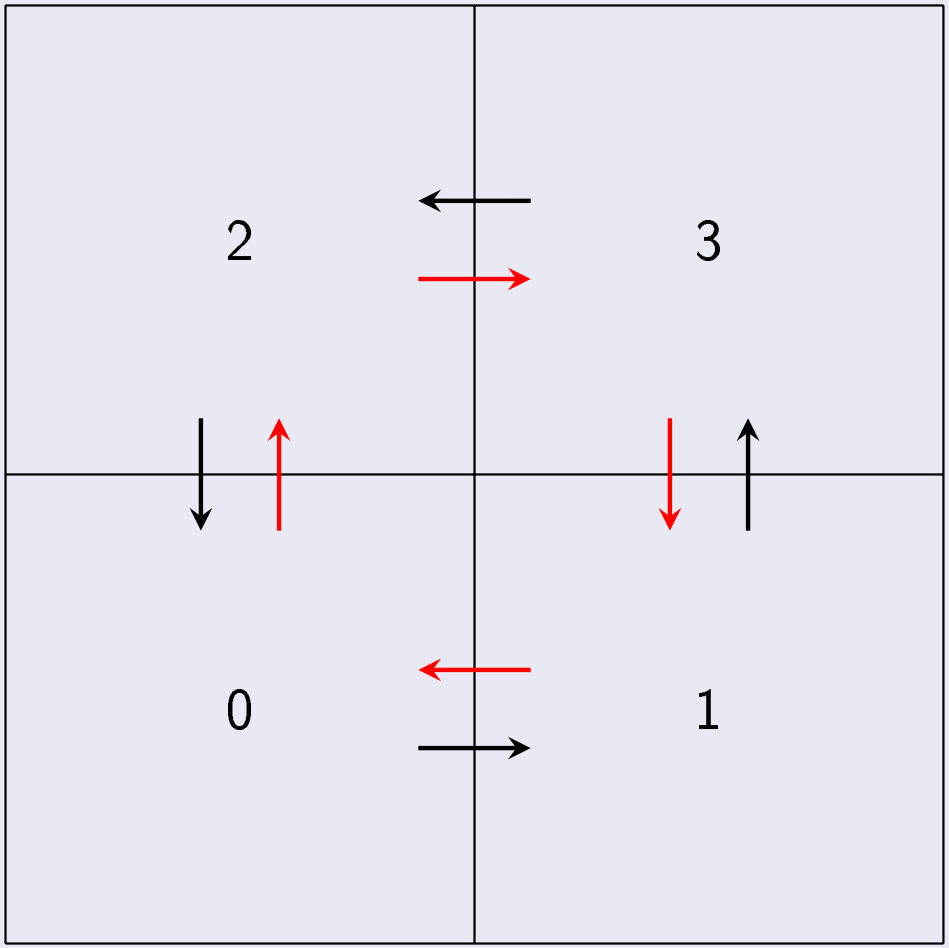
Communication point à point non bloquante

Exemple



Communication point à point non bloquante

Exemple

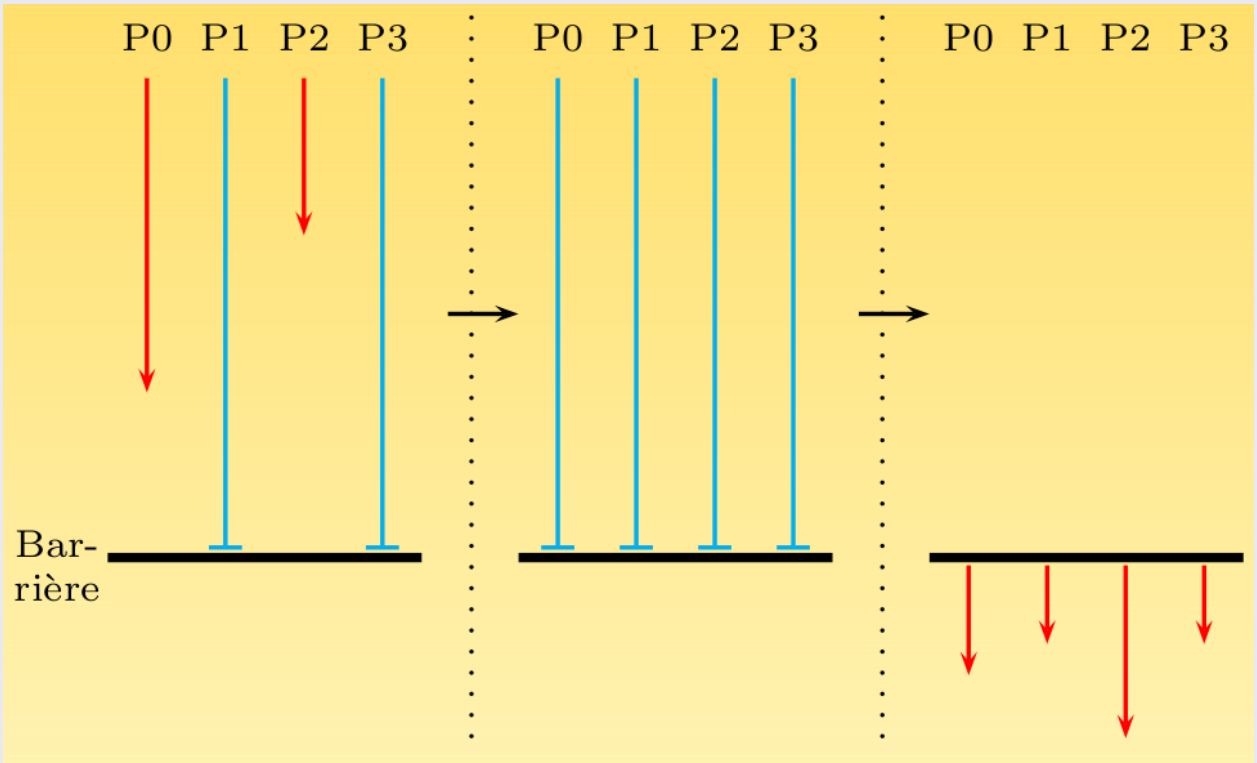


Communication point à point non bloquante

```
import mpi4py.MPI as mpi

rang, size = mpi.COMM_WORLD.rank, mpi.COMM_WORLD.size

if rang == 0:
    voisins = [2, 1]
if rang == 1:
    voisins = [0, 3]
if rang == 2:
    voisins = [3, 0]
if rang == 3:
    voisins = [1, 2]

recvalue = []
for v in voisins:
    mpi.COMM_WORLD.Isend([np.arange(1000), mpi.INT], v, tag=10*v + rang)
for v in voisins:
    recvalue.append(np.empty(1000, dtype=np.int))
    mpi.COMM_WORLD.Recv([recvalue[-1], mpi.INT], v, tag=10*rang + v)
print rang, voisins, recvalue
```

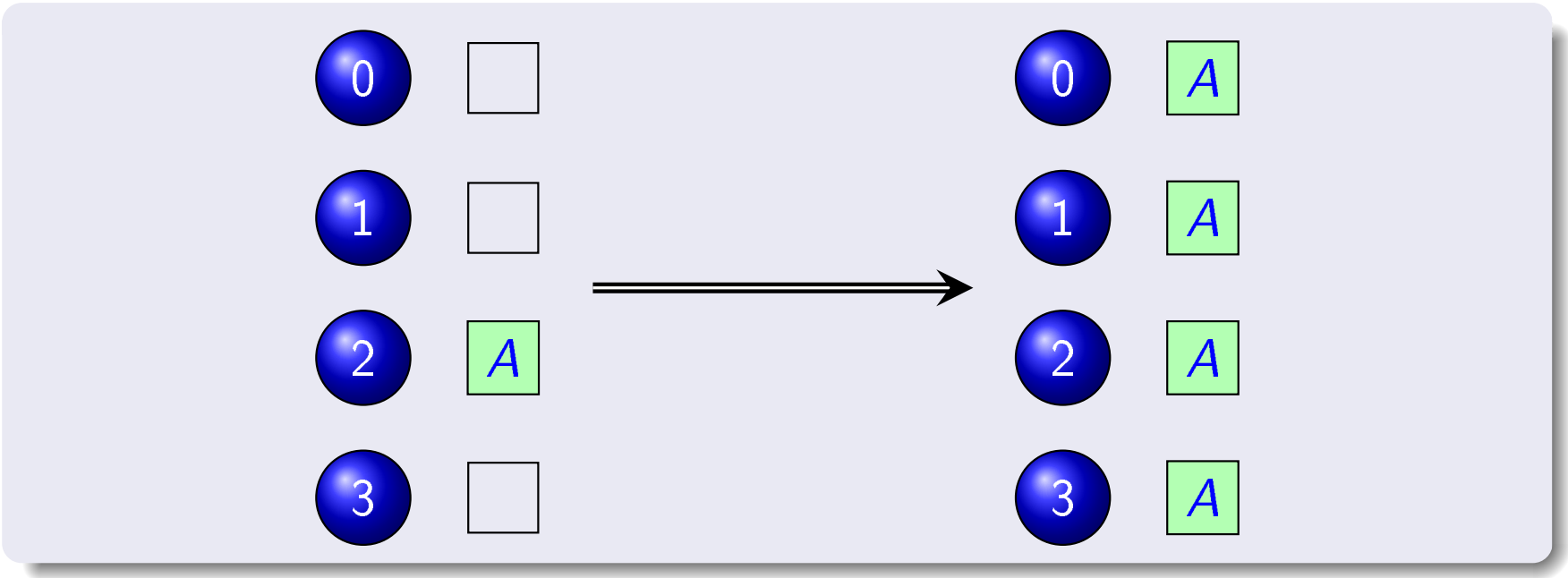
Synchronisation globale

mpi4py.MPI.COMM_WORLD.Barrier()



Diffusion générale

- bcast(obj = None, root = 0)
- Bcast(buf, root = 0)



Diffusion générale

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = [(1, 2), {2: 'toto', 3: 'titi'}]
    a = mpi.COMM_WORLD.bcast(a, 0)
else:
    a = mpi.COMM_WORLD.bcast(None, 0)
print a
```

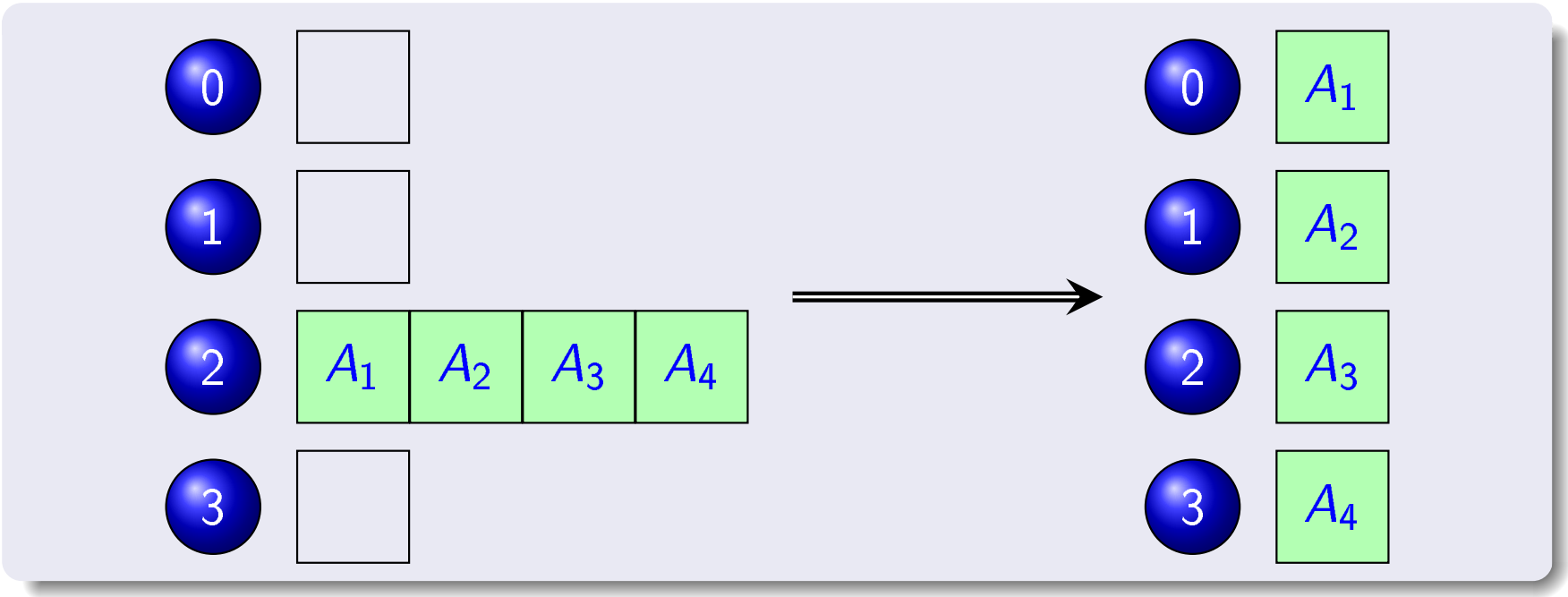
Diffusion générale

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 10
a = np.empty(n)

if mpi.COMM_WORLD.rank == 0:
    a = np.linspace(0, 1, n)

mpi.COMM_WORLD.Bcast([a, mpi.DOUBLE], 0)
print a
```

Communication dispersive

- scatter(sendobj=None, recvobj=None, root = 0)
- Scatter(sendbuf, recvbuf, root = 0)



Communication dispersive

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = [(1, 2), {2: 'toto', 3: 'titi'}]
    a = mpi.COMM_WORLD.scatter(a, 0)
else:
    a = mpi.COMM_WORLD.scatter(None, 0)
print a
```

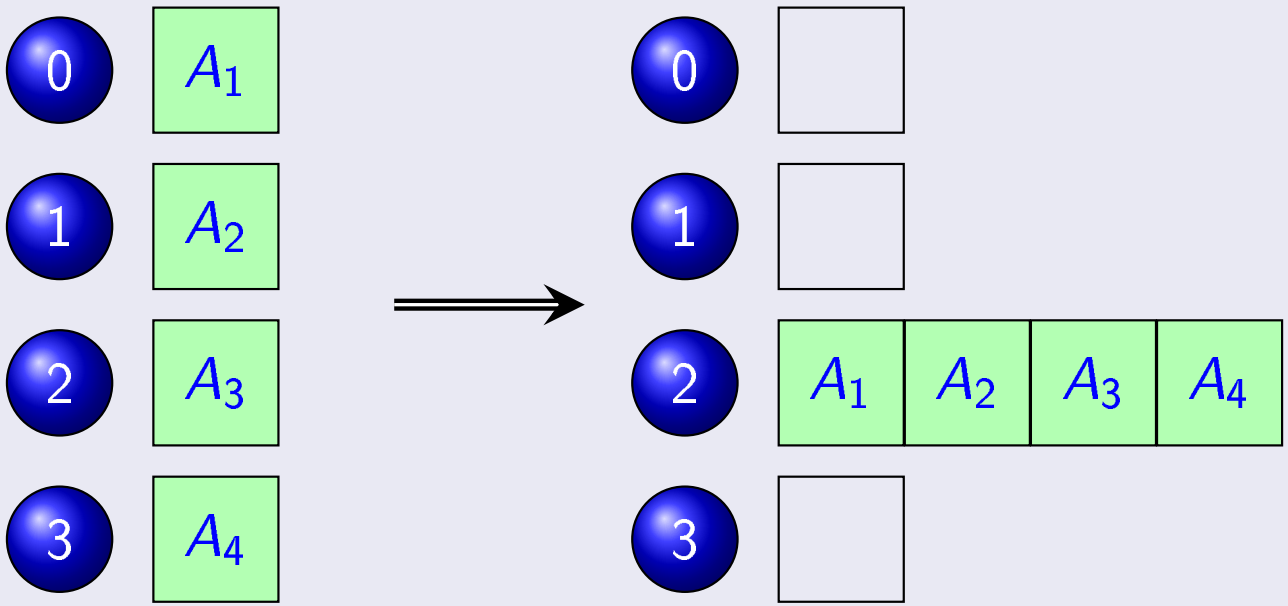
Communication dispersive

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 16
b = np.empty(n/4)

if mpi.COMM_WORLD.rank == 0:
    a = np.linspace(0, 1, n)
    mpi.COMM_WORLD.Scatter([a, mpi.DOUBLE], [b, mpi.DOUBLE], 0)
else:
    mpi.COMM_WORLD.Scatter(None, [b, mpi.DOUBLE], 0)
print b
```

Rassembler

- gather(sendobj=None, recvobj=None, root = 0)
- Gather(sendbuf, recvbuf, root = 0)



Rassembler

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = (1, 2)
    a = mpi.COMM_WORLD.gather(a, 0)
    print a
else:
    a = {2: 'toto', 3: 'titi'}
    mpi.COMM_WORLD.gather(a, 0)
```

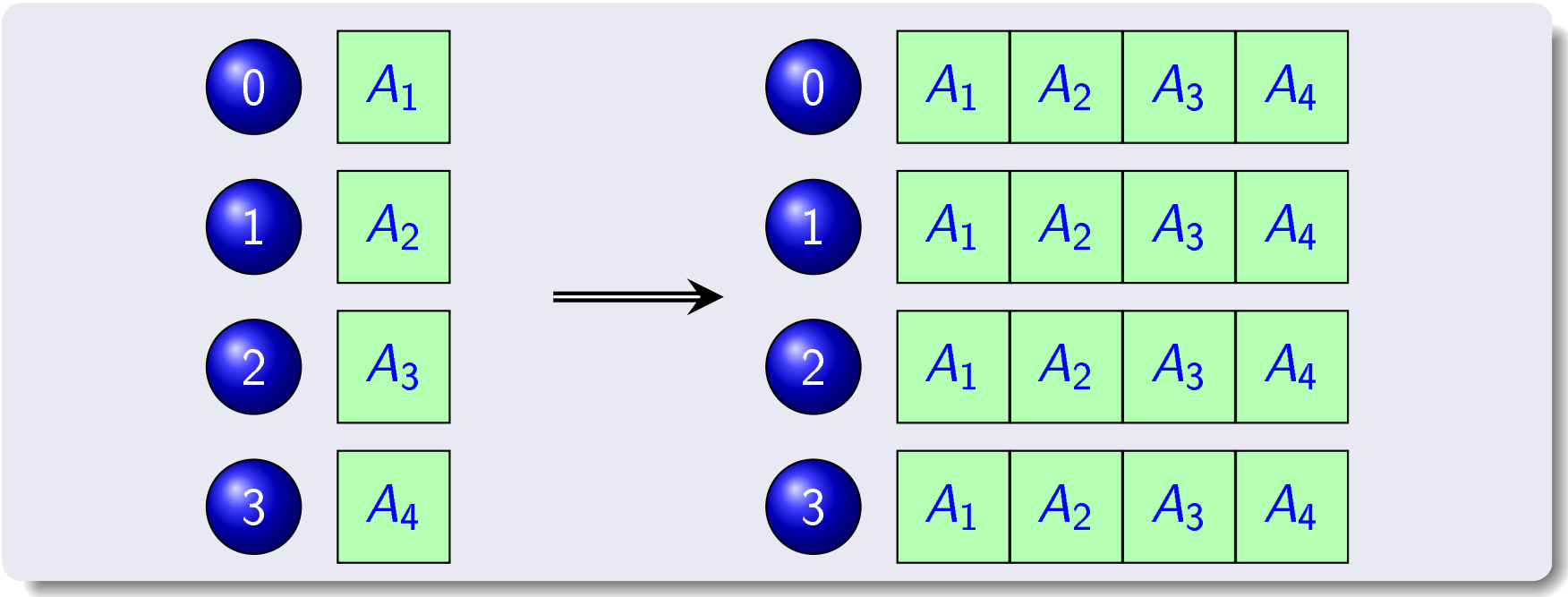
Rassembler

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 4
rank = mpi.COMM_WORLD.rank
size = mpi.COMM_WORLD.size
interval = mpi.COMM_WORLD.size*n - 1

deb = float(n*rank)/interval
fin = float(n*(rank + 1) - 1)/interval
a = np.linspace(deb, fin, n)

if mpi.COMM_WORLD.rank == 0:
    b = np.empty(n*size)
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], [b, mpi.DOUBLE], 0)
else:
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], None, 0)

if rank == 0:
    print b
```

Tout rassembler

- allgather(sendobj=None, recvobj=None)
- Allgather(sendbuf, recvbuf)



Rassembler

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = (1, 2)
    a = mpi.COMM_WORLD.gather(a, 0)
    print a
else:
    a = {2: 'toto', 3: 'titi'}
    mpi.COMM_WORLD.gather(a, 0)
```

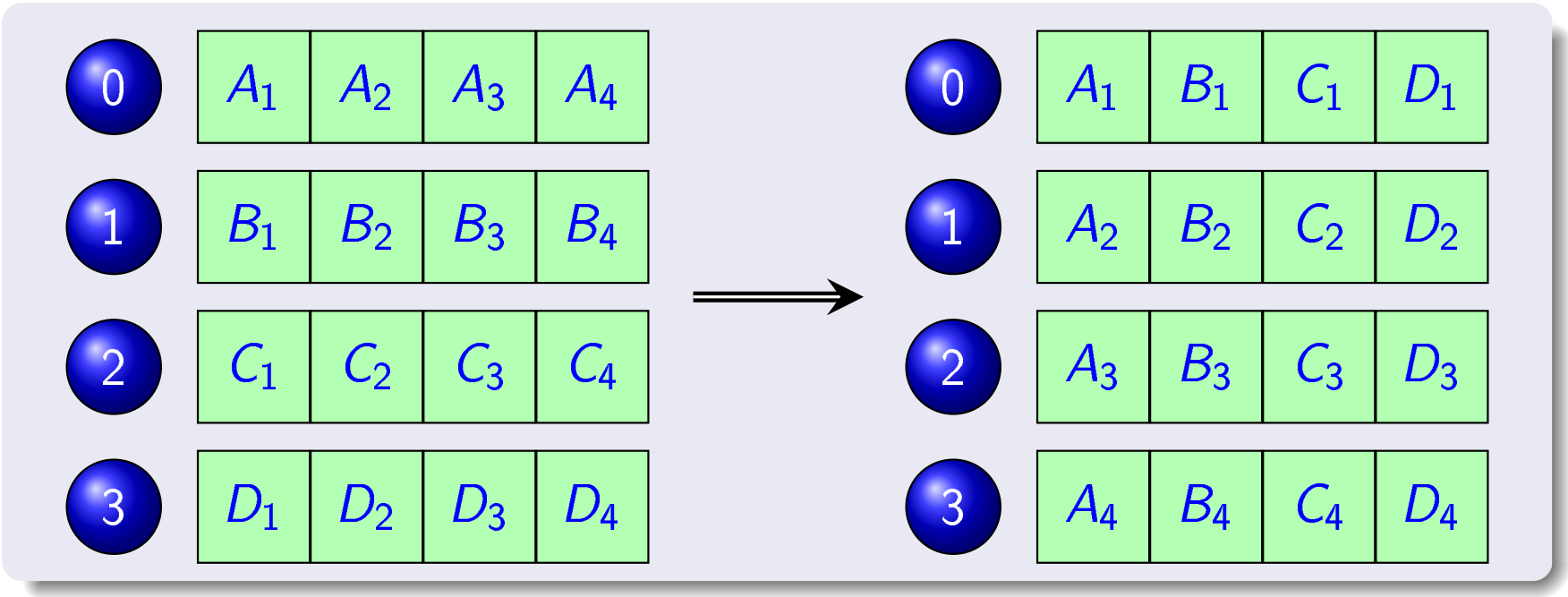
Rassembler

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 4
rank = mpi.COMM_WORLD.rank
size = mpi.COMM_WORLD.size
interval = mpi.COMM_WORLD.size*n - 1

deb = float(n*rank)/interval
fin = float(n*(rank + 1) - 1)/interval
a = np.linspace(deb, fin, n)

if mpi.COMM_WORLD.rank == 0:
    b = np.empty(n*size)
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], [b, mpi.DOUBLE], 0)
else:
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], None, 0)

if rank == 0:
    print b
```

Echanges croisés

- alltoall(sendobj=None, recvobj=None)
- Alltoall(sendbuf, recvbuf)



Réduction

- reduce(sendobj=None, recvobj=None, op=SUM, root=0)
- Reduce(sendbuf, recvbuf, op=SUM, root=0)
- allreduce(sendobj=None, recvobj=None, op=SUM)
- Alleduce(sendbuf, recvbuf, op=SUM)

- *mpi4py.MPI.SUM* : somme des éléments
- *mpi4py.MPI.PROD* : produit des éléments
- *mpi4py.MPI.MAX* : recherche du maximum
- *mpi4py.MPI.MIN* : recherche du minimum
- *mpi4py.MPI.MAXLOC* : recherche de l'indice du maximum
- *mpi4py.MPI.MINLOC* : recherche de l'indice du minimum
- ...

Réduction

On reprend notre classe point en y ajoutant

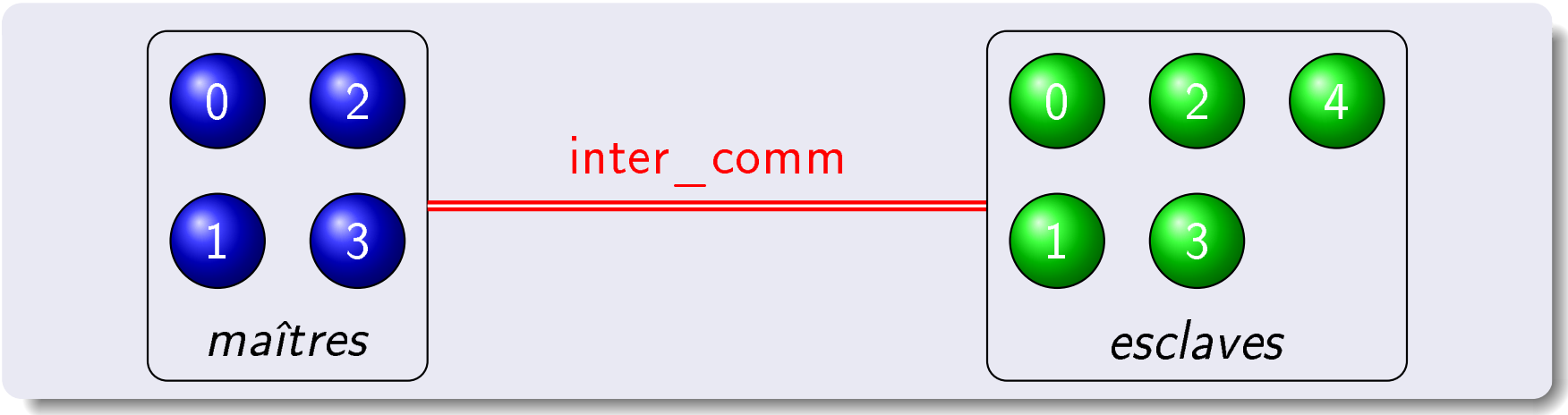
```
    def __add__(self, p2):
        return point(0, self.x + p2.x, self.y + p2.y)
```

Exemple avec cPickle

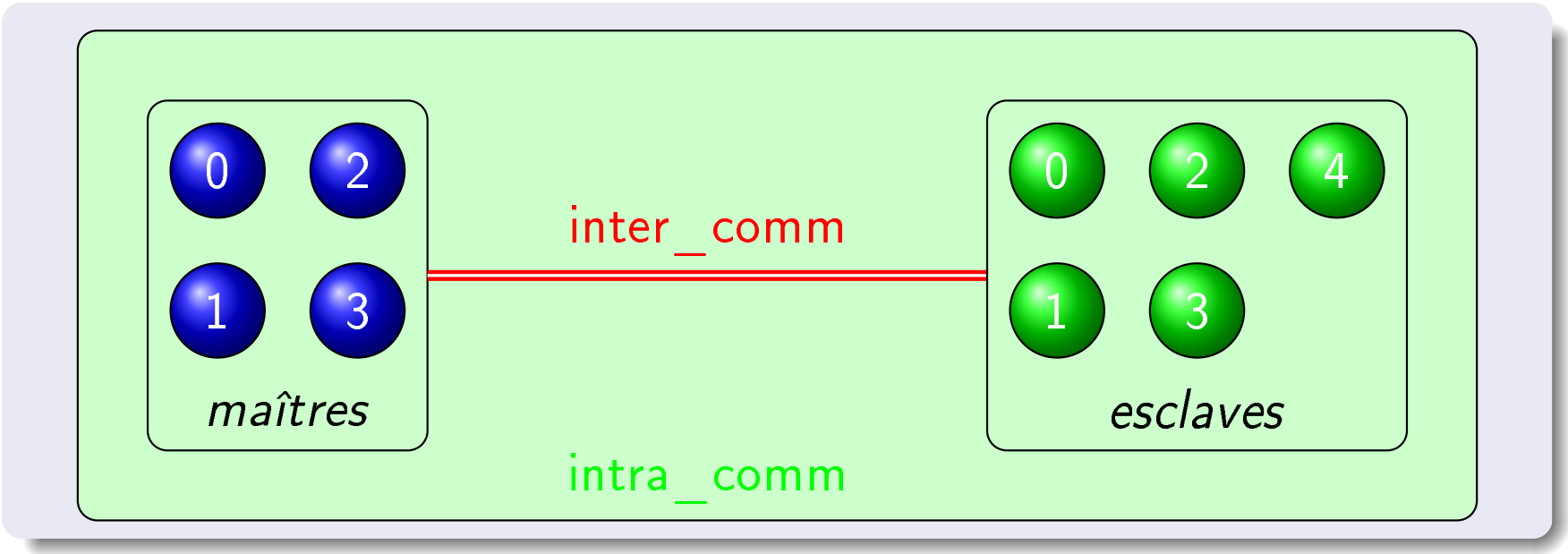
```
import mpi4py.MPI as mpi
from point import point

rank = mpi.COMM_WORLD.rank
p1 = point(0, rank, rank + 1)

p2 = mpi.COMM_WORLD.allreduce(p1, mpi.SUM)
print p2
```

Idée générale

Idée générale



Idée générale



La fonction Spawn

Création d'un lien entre maîtres et esclaves

```
Comm.Spawn(self, command, args=None, maxprocs=1,
           info=INFO_NULL, root=0)
```

Comm

- communicateur COMM_WORLD,
- communicateur COMM_SELF,
- communicateur que vous aurez créé.

Remarque : cette commande renvoie un communicateur représentant l'inter_comm.

Autres fonctions utiles

Comm.Get_parent()

Le processus esclave demande si il a un processus maître et lequel.

Comm.Merge(high=False)

Création de l'intra_comm entre le ou les maîtres et les esclaves.

Comm.Disconnect()

Déconnecte le processus d'un communicateur.

Exemple 1

master.py

```
#!/usr/bin/env python
import mpi4py.MPI as mpi

worker = mpi.COMM_WORLD.Spawn("slave.py", None, 2, root=1)
intra = worker.Merge()

print 'maitre:', mpi.COMM_WORLD.Get_rank(), \
                 intra.Get_rank(), intra.Get_size()

intra.Disconnect()
worker.Disconnect()
```

Attention

le fichier *slave.py* doit être exécutable.

Exemple 1

slave.py

```
#!/usr/bin/env python
import mpi4py.MPI as mpi

master = mpi.Comm.Get_parent()
intra = master.Merge()

print 'ouvrier', intra.Get_rank(), intra.Get_size()

intra.Disconnect()
master.Disconnect()
```

Exemple 2

master.py

```
#!/usr/bin/env python
import mpi4py.MPI as mpi

worker = mpi.COMM_SELF.Spawn("slave.py", None, 2, root=0)
intra = worker.Merge()

print 'maitre:', mpi.COMM_WORLD.Get_rank(), \
                 intra.Get_rank(), intra.Get_size()

intra.Disconnect()
worker.Disconnect()
```

On reprend le *slave.py* de l'exemple précédent.

Interfacer avec swig

testSwig.c

```
#include "testSwig.h"
int testSwig(MPI_Comm comm, int a)
{
  int sum, size, rank;

  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  MPI_Allreduce(&a, &sum, 1, MPI_INT, MPI_SUM, comm);
  return sum;
}
```

Interfacer avec swig

testSwig.i

```
%module testSwig
%{
#include "testSwig.h"
%}
%include mpi4py/mpi4py.i
%mpi4py_typemap(Comm, MPI_Comm);
%include "testSwig.h"
```

Interfacer avec swig

setup.py

```
from distutils.core import setup, Extension
import mpi4py

swig = mpi4py.get_include()
include = [swig, "/usr/lib/openmpi/include"]
sources = ["testSwig.c", "testSwig.i"]

setup( ext_modules = [
        Extension("_testSwig", sources = sources,
                  swig_opts = ["-I" + swig],
                  include_dirs = include
                  )
      ]
)
```

Interfacer avec swig

Création du module

```
$ python setup.py build_ext -i
```

Exemple d'utilisation

```
import mpi4py.MPI as mpi
import numpy as np
from _testSwig import testSwig

rank = mpi.COMM_WORLD.Get_rank()
sum = testSwig(mpi.COMM_WORLD, rank)
print sum
```

Interfacer avec f2py

testf2py.f90

```
subroutine testf2py(comm, a, sum)
  use mpi
  implicit none
  integer :: comm
  integer :: rank, size, nlen, ierr
  integer :: a, sum
!f2py integer, intent(out):: sum

  call MPI_Comm_rank(comm, rank, ierr)
  call MPI_Comm_size(comm, size, ierr)
  call MPI_allreduce(a, sum, 1, MPI_INTEGER ,
                     MPI_SUM, comm ,ierr)
end subroutine testf2py
```

Interfacer avec f2py

Création du module

```
$ f2py --f90exec=mpif90 -m testf2py -c testf2py.f90
```

Exemple d'utilisation

```
import mpi4py.MPI as mpi
from testf2py import testf2py

rank = mpi.COMM_WORLD.Get_rank()
sum = testf2py(mpi.COMM_WORLD.py2f(), rank)
print "processus", rank, "somme dans python", sum
```

Interfacer avec Cython

testcython.pyx

```
cimport mpi4py.MPI as MPI
from mpi4py.mpi_c cimport *

cdef extern from "stdio.h":
    int printf(char*, ...)

cdef int c_testcython(MPI_Comm comm, int a):
    cdef int size, rank, sum

    MPI_Comm_size(comm, &size)
    MPI_Comm_rank(comm, &rank)

    printf("Hello, World! I am process %d of %d.\n",
           rank, size)

    MPI_Allreduce(&a, &sum, 1, MPI_INT, MPI_SUM, comm)

    return sum
```

Interfacer avec Cython

testcython.pyx (suite)

```
def testcython(MPI.Comm comm not None, a ):
    cdef MPI_Comm c_comm = comm.ob_mpi
    return c_testcython(c_comm, a)
```

Interfacer avec Cython

setup.py

```
from distutils.core import setup, Extension
from Cython.Distutils import build_ext
import mpi4py

cythonInc = mpi4py.get_include()
includeDir = [cythonInc, "/usr/lib/openmpi/include"

setup(ext_modules = [
        Extension("testcython", sources = ["testcython.pyx"],
                  cython_opts = ["-I" + cythonInc],
                  include_dirs = includeDir
                 )],
      cmdclass = {'build_ext': build_ext}
)
```

Interfacer avec Cython

Création du module

```
$ python setup.py build_ext -i
```

Exemple d'utilisation

```
import mpi4py.MPI as mpi
from testcython import testcython

rank = mpi.COMM_WORLD.Get_rank()
sum = testcython(mpi.COMM_WORLD, rank)
print "processus", rank, "somme dans python", sum
```

Plan

Ressources

- cours MPI de l'IDRIS
- site de mpi4py
- site de pyMPI