Taller Computación Altas Prestaciones

MPI- Python - mpi4Py

Pedro Antonio Varo Herrero

Antes de nada!!

- Cosas a instalar:
  - **OpenMPI**:
    - http://www.open-mpi.org/software/ompi/v1.8/downloads/openmpi-1.8.3.tar.gz
  - **Mpi4py**: pip install mpi4py
  - **Cython**: pip install cython

Que vamos a ver:

1. Que es esto del HPC
    1. Algunos Supercomputadores
2. Arquitecturas y Tecnologías
3. MPI4Py
    1. Como funciona
    2. Sus métodos y diferencias con C
4. A Programar!!!

Supercomputador Marenostrum - Barcelona

- **Peak Performance of 1,1 Petaflops**
- **100.8 TB of main memory**
- **Main Nodes**
  - **3,056 compute nodes – 6,112 Processors**
  - **2x Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 GH**
  - **3056 * 8cores*2Intel SB = 48,896 Cores**
  - **64 nodes with 8x16 GB DDR3-1600 DIMMS (8GB/core)**
  - **64 nodes with 16x4 GB DDR3-1600 DIMMS (4GB/core)**
  - **2880 nodes with 8x4 GB DDR3-1600 DIMMS (2GB/core)**

http://www.bsc.es/marenostrum-support-services/mn3

BSC NVIDIA GPU Cluster MinoTauro - Barcelona

NVIDIA GPU is a cluster with 128 Bull B505 blades each blade with the following configuration:

- **2 Intel E5649 (6-Core) processor at 2.53 GHz**
- **2 M2090 NVIDIA GPU Cards**
- **Cada Nodo: 12 Núcleos Intel + 512 *2 Núcleos Cuda**
- **Total: 1536 Intel Cores + 256 GP-GPU**

- **24 GB of Main memory**
- **Peak Performance: 185.78 TFlops**
- **250 GB SSD (Solid State Disk) as local storage**
- **2 Infiniband QDR (40 Gbit each) to a non-blocking network**
- **RedHat Linux**
- **14 links of 10 GbitEth to connect to BSC GPFS Storage**

http://www.bsc.es/marenostrum-support-services/other-hpc-facilities/nvidia-gpu-cluster

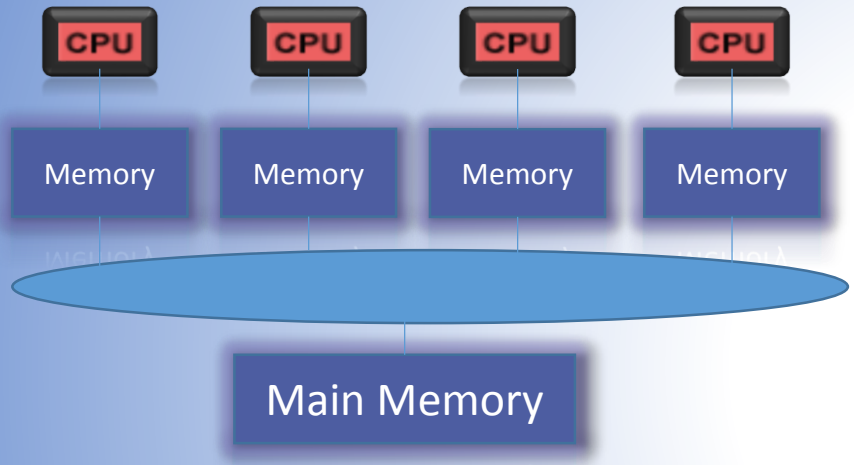
The Cray XC30- Swiss National Supercomputing Centre (CSCS)

Manufacturer:	**Cray Inc.**
Cores:	**115,984**
Linpack Performance (Rmax)	**6,271 TFlop/s**
Theoretical Peak (Rpeak)	**7,788.85 TFlop/s**
Nmax	**4,128,768**
Power:	**2,325.00 kW**
Processor:	**Xeon E5-2670 8C 2.6GHz**

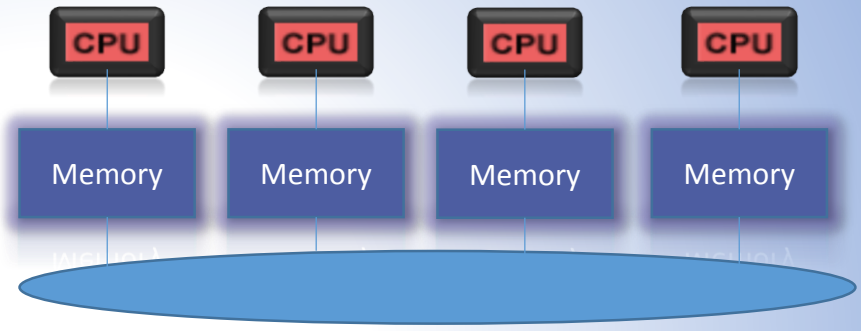
http://www.cscs.ch/computers/index.html
http://www.top500.org/system/177824 - Rank: 6 June 2014

Arquitecturas según memoria

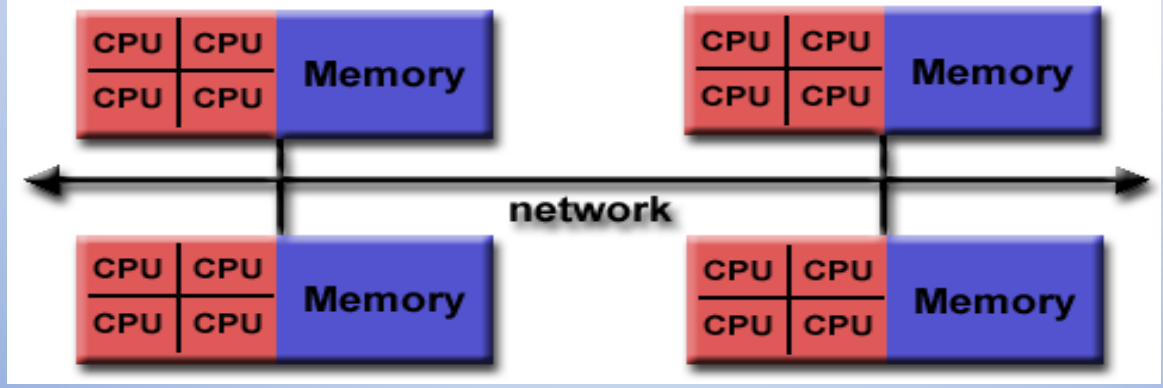
Uniform Memory Access
UMA



Non-Uniform Memory Access
NUMA

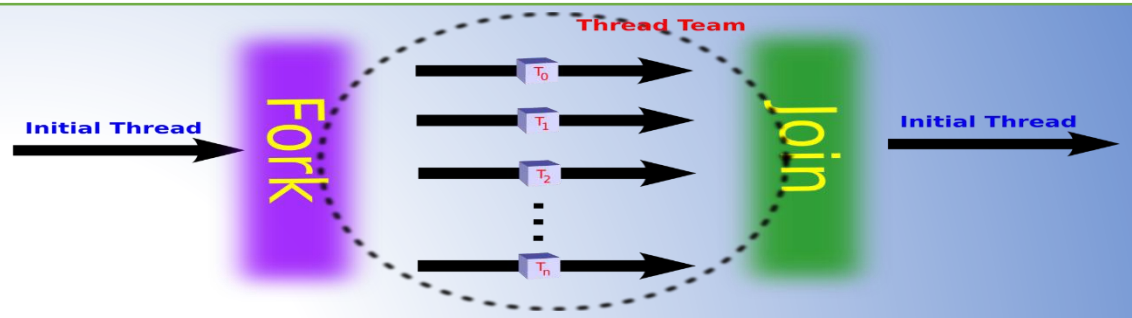


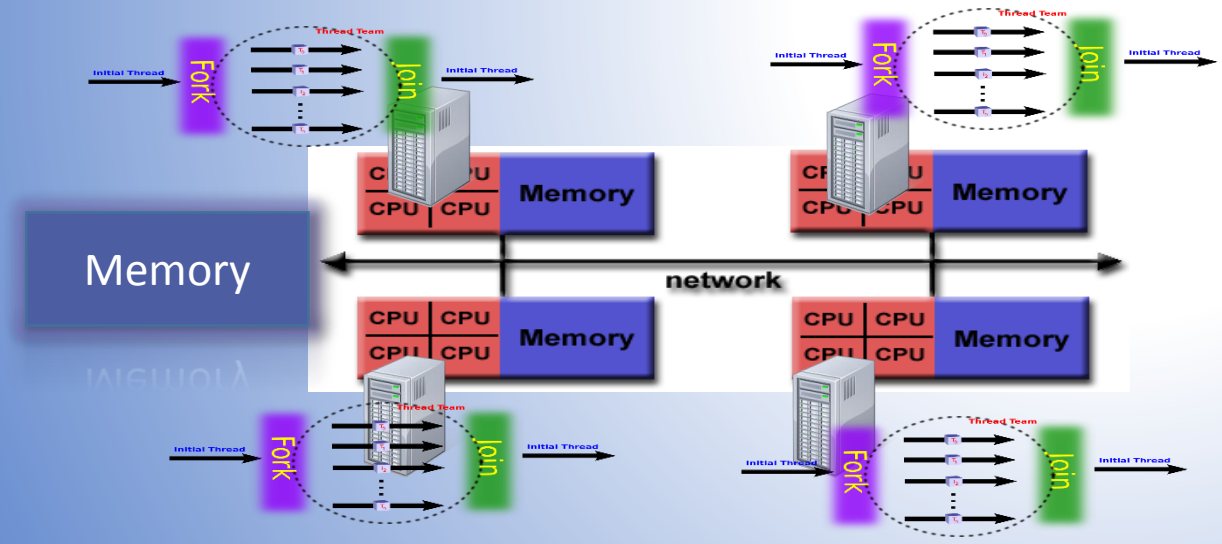
Memory Passing Message
MPM

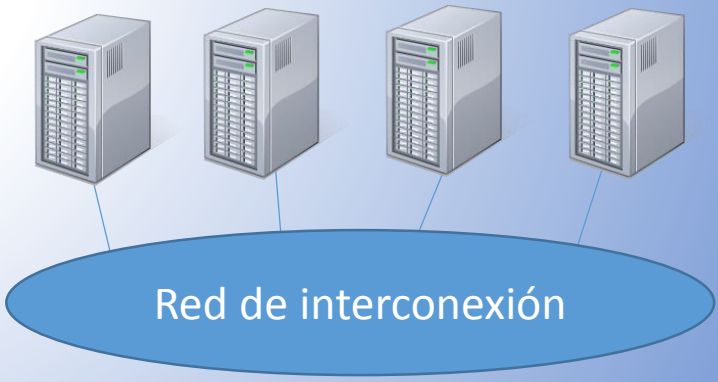


Paradigmas de Programación Paralela

- Por manejo de Threads.
- Por paso de mensajes.
- Hibrida: Threads + Paso de mensajes.







Multiprocessing

MPI4Py

PyCuda

PyOpenCL

“Estándares” de librerías de programación paralela

Multiprocessing

MPI4Py

PyCuda

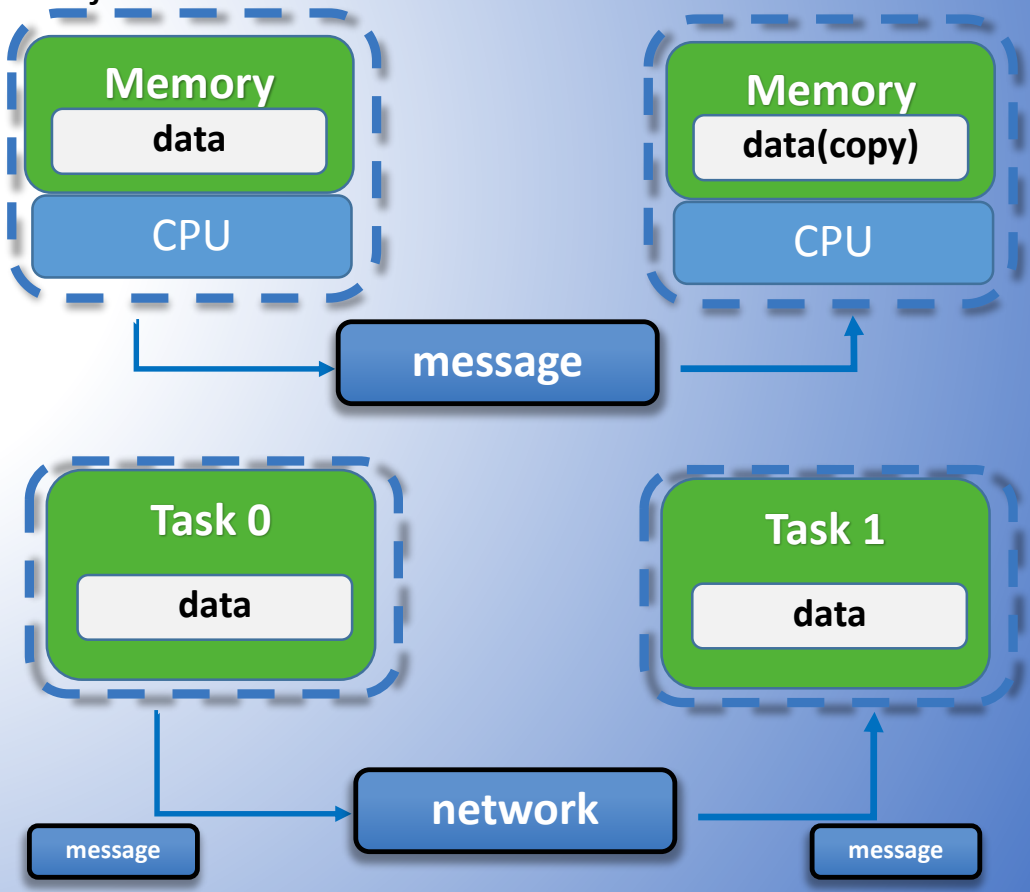
PyOpenCL

“Estándares” de librerías de programación paralela

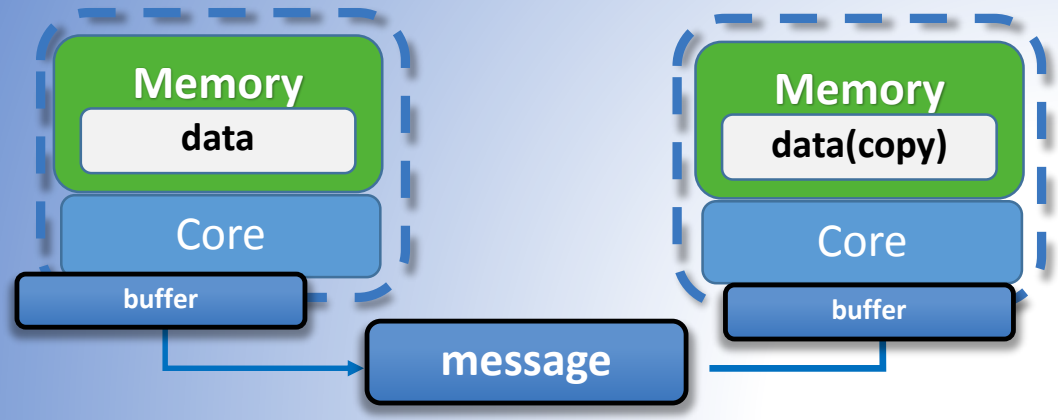
MPI – Message Passing Interface

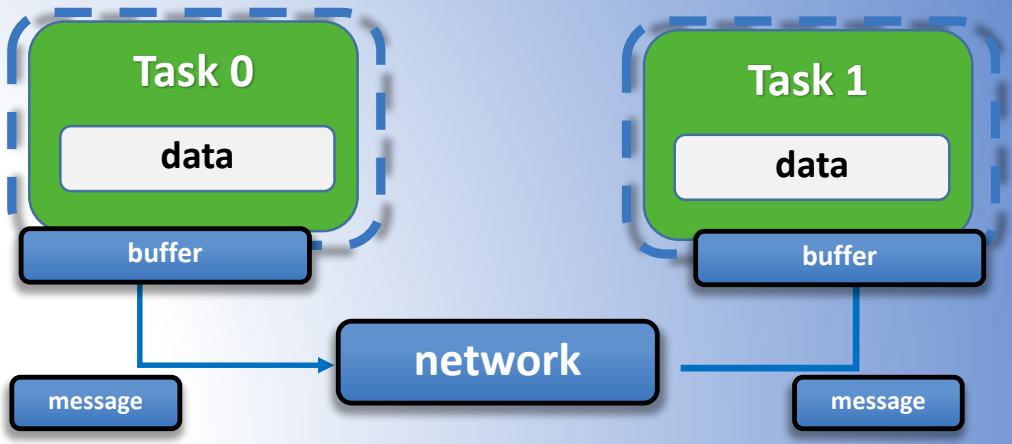
- Usado en arquitecturas de memoria distribuida.
- Da comunicación entre los distintos procesadores/nodos/maquinas del sistema.
- Se crean distintas procesos, cada uno con su propio espacio de memoria.
- Los datos entre procesos se comparten en el paso del mensaje.
- Código escalable.
- Estandar: MPI(C/C++,Fortran).

- **Python -> MPI4Py**
  - Proyecto de Lisandro Dalcin, basado en MPI-1/2/3
  - Implementa la mayoría de funciones de MPI
  - http://mpi4py.scipy.org/
  - Repo: https://bitbucket.org/mpi4py/mpi4py



MPI – mpi4py





- **Python -> MPI4Py**
  - Envia objetos Python o Arrays
  - Repo: https://bitbucket.org/mpi4py/mpi4py
  - API: http://mpi4py.scipy.org/docs/apiref/index.html

Des/Serialización de objetos, en ASCII o Binario:

- Pickle
- cPickle

MPI – mpi4py

- **Python -> MPI4Py**
  - Envia objetos Python o Arrays (Numpy)
  - Repo: https://bitbucket.org/mpi4py/mpi4py
  - API: http://mpi4py.scipy.org/docs/apiref/index.html
- **Que nos hace falta instalar:**
  - **Una versión de MPI: OpenMPI o MPICH**
    - http://www.open-mpi.org/software/ompi/v1.8/
  - **Python 2.7 – 3.4**
  - **Modulo: mpi4py**
    - **pip install mpi4py**

MPI – mpi4py

- Para empezar:
  - Importar :
    - `from mpi4py import MPI`
  - Parámetros a obtener:
    - `comm = MPI.COMM_WORLD`
    - `rank = MPI.COMM_WORLD.Get_rank()`
    - `size = MPI.COMM_WORLD.Get_size()`
    - `name = MPI.Get_processor_name()`
  - Para su ejecución:
    - `mpirun -np <P> python <code>.py`

MPI – mpi4py

- Para empezar:
  - Vamos a escribir un Hello World en paralelo y ejecutamos con diferentes números de procesos.
  - Ejemplo:
    - `mpirun -np 4 python helloworld.py`
  - Output:
    - `Hello World!!, This is process 0 of 4,  en “nombre host”`
    - `Hello World!!, This is process 2 of 4,  en “nombre host”`
    - `Hello World!!, This is process 1 of 4,  en “nombre host”`
    - `Hello World!!, This is process 4 of 4,  en “nombre host”`

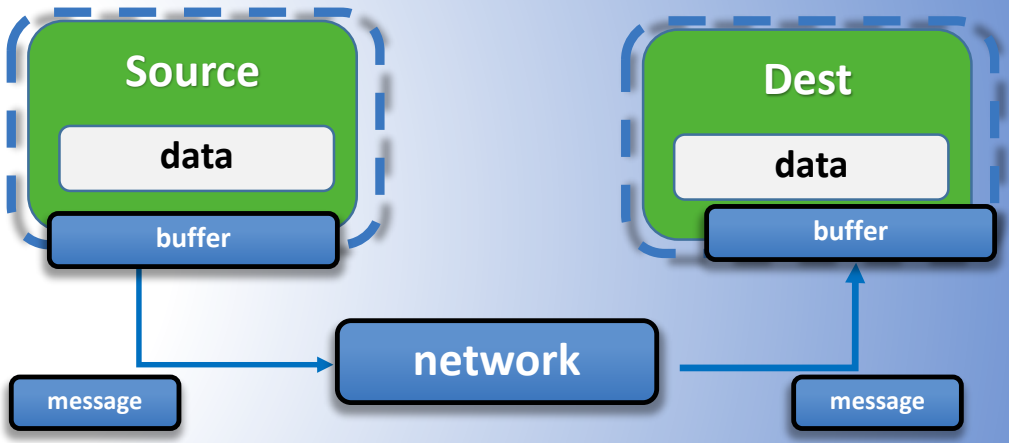
MPI – mpi4py

- Para empezar:
  - Hello World
  - `mpirun -np 4 python helloworld.py`

```
1. from mpi4py import MPI
2.
3. rank = MPI.COMM_WORLD.Get_rank()
4. size = MPI.COMM_WORLD.Get_size()
5. name = MPI.Get_processor_name()
6.
7. print("Hello, world! This is rank %d of %d running on %s" % (rank,
size, name))
```

MPI - mpi4py

- **Point-to-Point Comunications:**
  - Send(…) Rcv(…)
  - Blocking: Esperan a que los datos se puedan enviar y modificar con seguridad.
    - Send(),recv(),Send(), Recv(), SendRecv()
  - Non-blocking: No esperan a que los datos se puedan enviar con seguridad y el proceso continua su tarea.
    - Isend(), irecv(), Isend(), Irecv()



MPI - mpi4py

- MPI C vs MPI python – mpi4py

- En C las funciones de comunicación suelen tener: 6 parámetros.
  - Send/Recv(buf,size,datatype,dest,tag,comm)

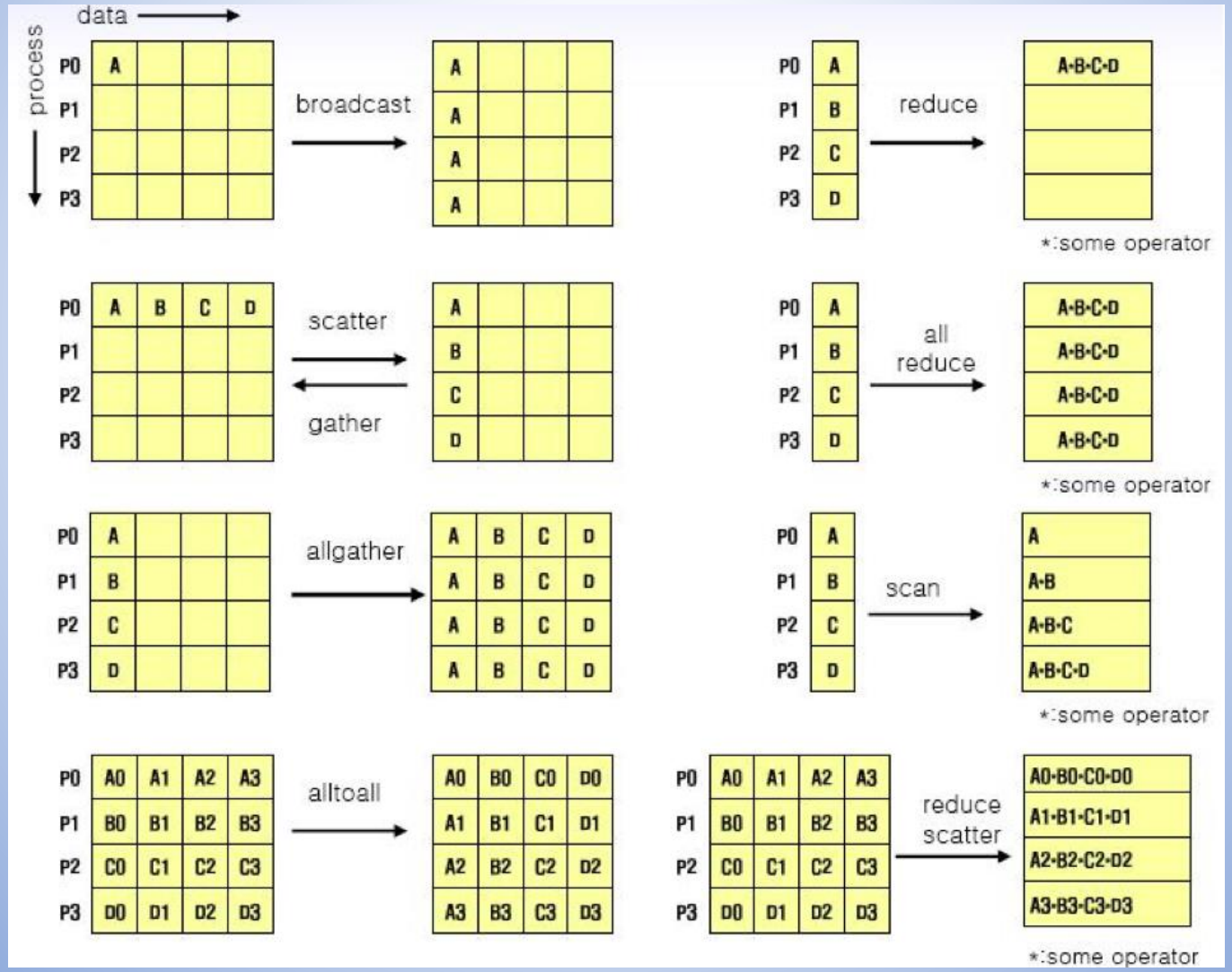
```
MPI_Recv(result, result sz, result type, MPI ANY SOURCE,result,
tag, comm, MPI STATUS IGNORE);
```

- En Python, sólo 3 y nada de punteros.
  - Send(data,dest, tag) / var=Recv(source,tag)
  - En data podemos enviar
    - Cualquier objeto Python
    - Si es un Array, indicando el tipo de dato del Array -> [Array,Type]

```
MPI.COMM_WORLD.Send(data,dest=4,tag=0)
Send([data,MPI.INT], dest=4,tag=0)
```

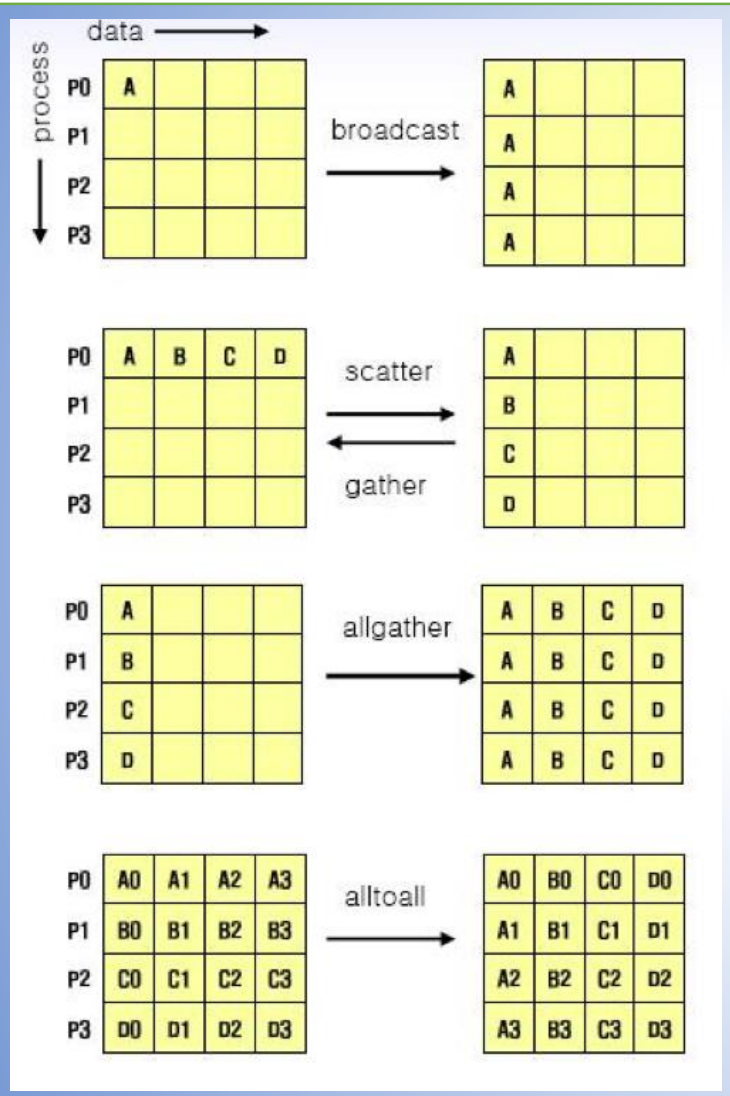
MPI - mpi4py

- Collective Communications:



MPI - mpi4py

- Collective Communications:

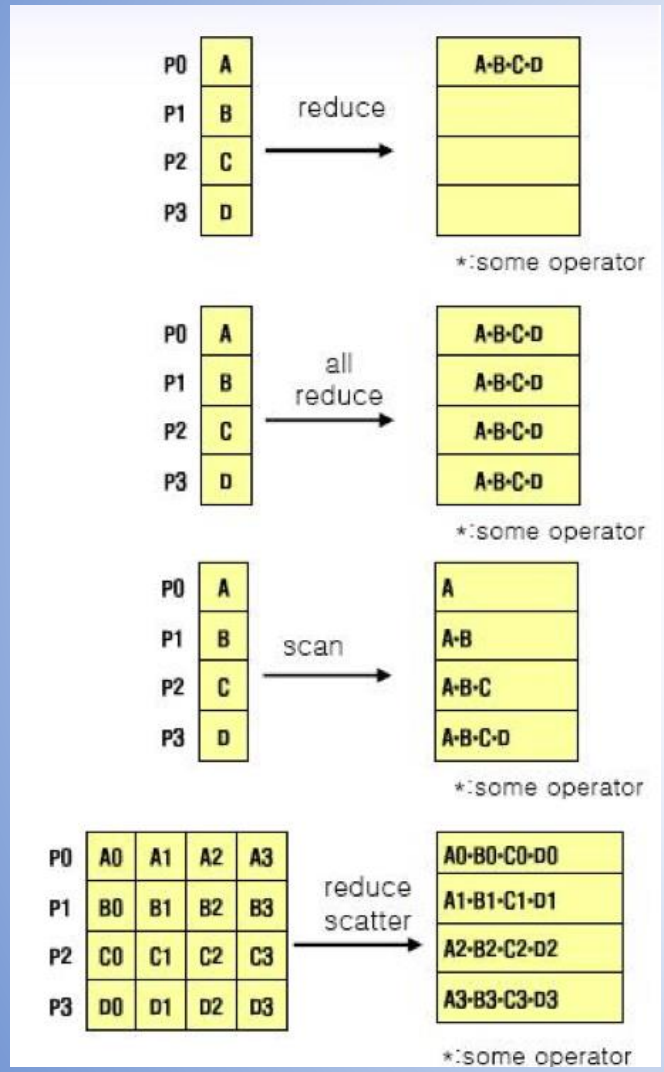


data=[0,1,2,3]

- `Bcast(data, root=0)`
- `Data=Scatter(data,root=0)`
- `Data=Gather(data,root=0)`
- `Allgather(data_send, data_recv)`
- `Alltoall(data_send, data_recv)`

MPI - mpi4py

- Collective Communications:




```
data=[0,1,2,3]
```

- `reduce(data, data_recv, op=SUM root=0)`
- `allreduce(data, data_recv, op=SUM)`
- `scan(data_send, data_recv, op=PROD)`
- `reduce_scatter(data_send, data_recv, op=PROD)`

MPI - mpi4py

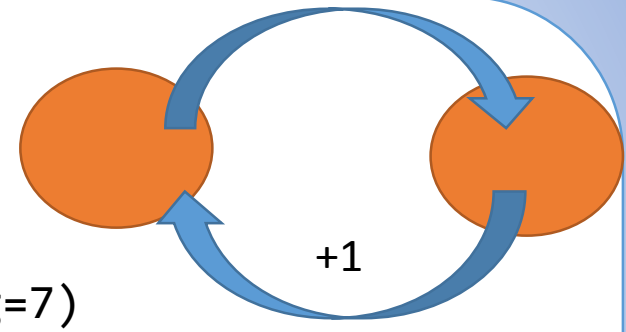
Now, Think in Parallel!!
Solve Problems, and Programming!!

MPI - mpi4py

- Probemos las funciones de MPI
  - Ping-Pong
  - Ring
- Calculo de PI en paralelo
- Multiplicación de Matrices
- Posible problema: TSP, Clasificaión con K-means, Comparación cadenas de ADN ……

MPI - mpi4py

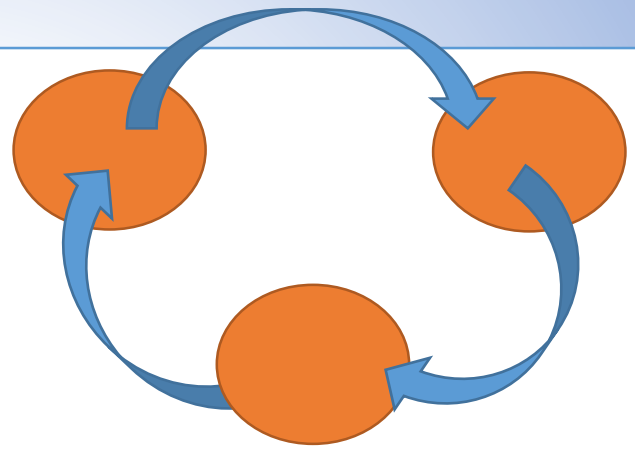
Ping-Pong




```
1. count = 0
2. for x in range(1000):
3.    if rank == 0:
4.        comm.send(count, dest=1, tag=7)
5.        count = comm.recv(source=1, tag=7)
6.    else:
7.        counter = comm.recv(source=0, tag=7)
8.        counter += 1
9.        comm.send(coun, dest=0, tag=7)
10.
11. if rank == 0:
12.    print("Total number of message exchanges: %d" % coun
t)
```

MPI - mpi4py

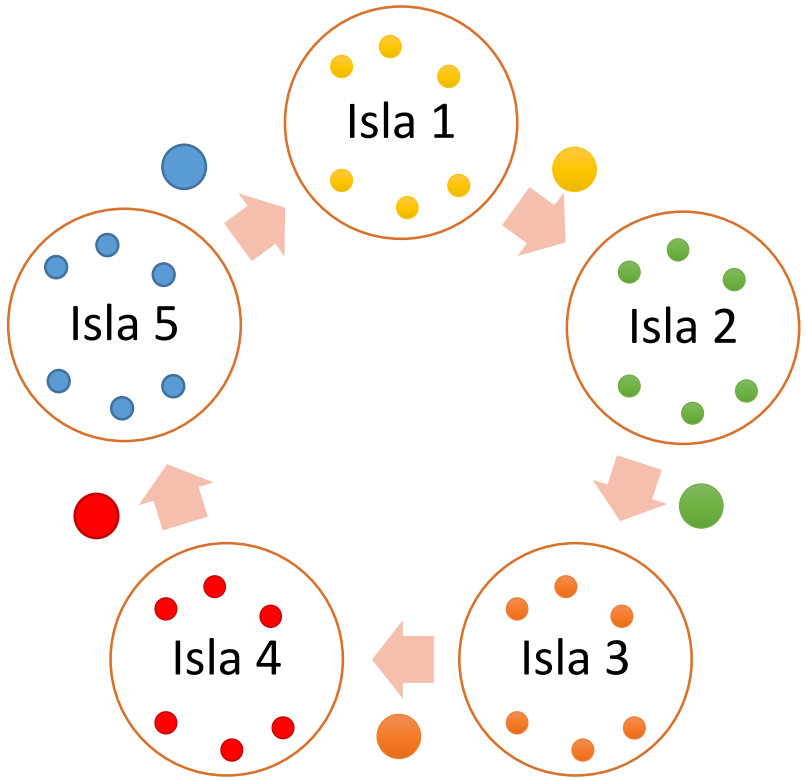
Ring



```
1.counter = 0
2.data = randint(0,100)
3.for x in range(size):
4.    comm.send(data, dest=(rank+1)%size, tag=7)
5.    data = comm.recv(source=(rank+size-1)%size, tag=7)
6.    counter += data
7.
8.print("[%d] Total sum: %d" % (rank,counter))
```

MPI - mpi4py

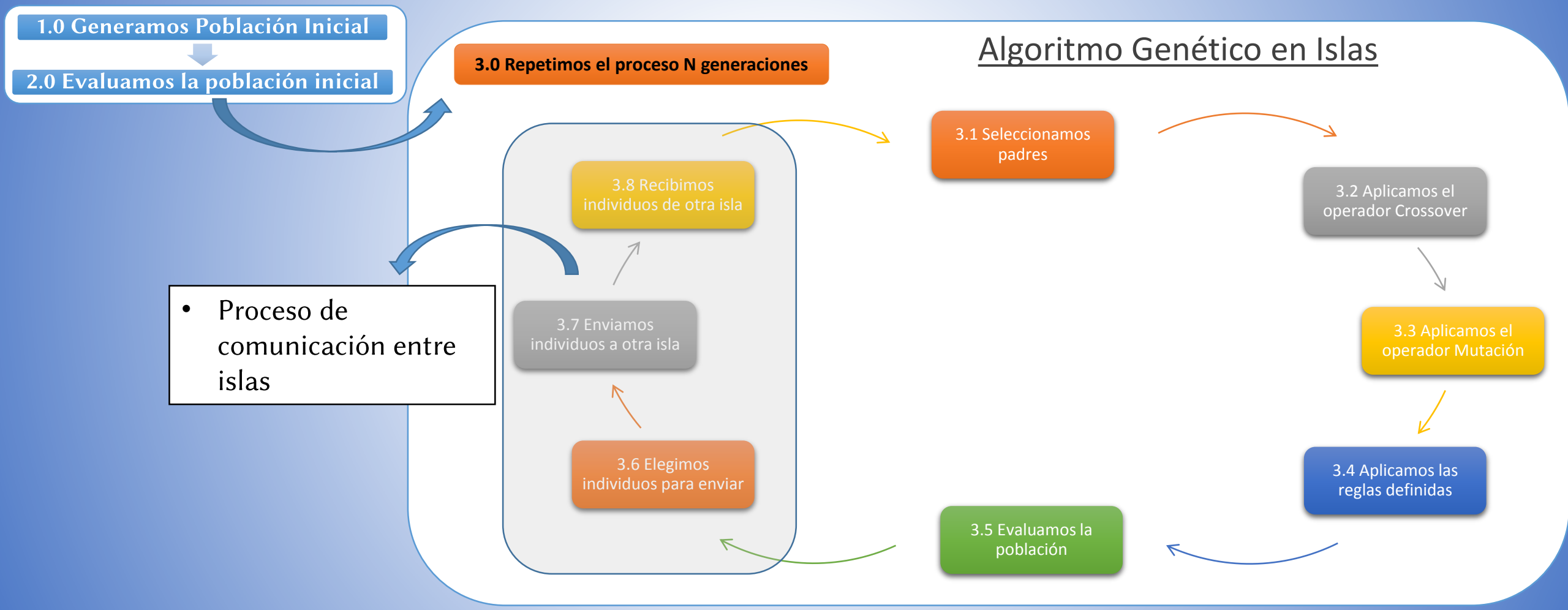
Algoritmo Genético en Islas



Cada isla:

- Un proceso
- Un algoritmo genético

MPI - mpi4py


1.0 Generamos Población Inicial
2.0 Evaluamos la población inicial
3.0 Repetimos el proceso N generaciones
Algoritmo Genético en Islas
3.1 Seleccionamos padres
3.2 Aplicamos el operador Crossover
3.3 Aplicamos el operador Mutación
3.4 Aplicamos las reglas definidas
3.5 Evaluamos la población
3.6 Elegimos individuos para enviar
3.7 Enviamos individuos a otra isla
3.8 Recibimos individuos de otra isla
• Proceso de comunicación entre islas

MPI - mpi4py

Pedro Antonio Varo Herrero


Pedro Varo Herrero


@pevahe91


Pedro Antonio Varo Herrero



Pedro Varo Herrero – pevahe@gmail.com

Now, Think in Parallel!!

Solve Problems, and Programming!!