Message Passing Interface (MPI)
Introduction

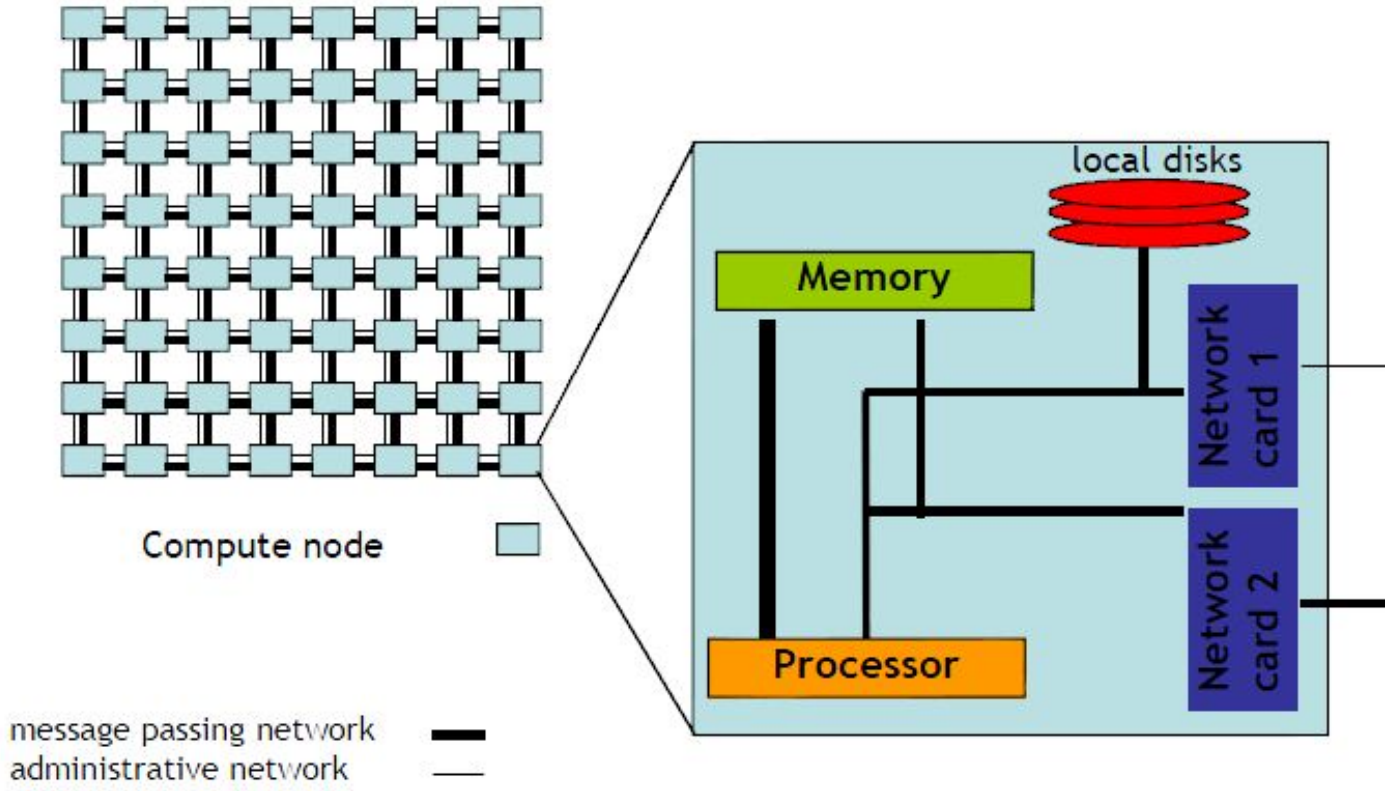
Amal KHABOU

amal.khabou@lri.fr

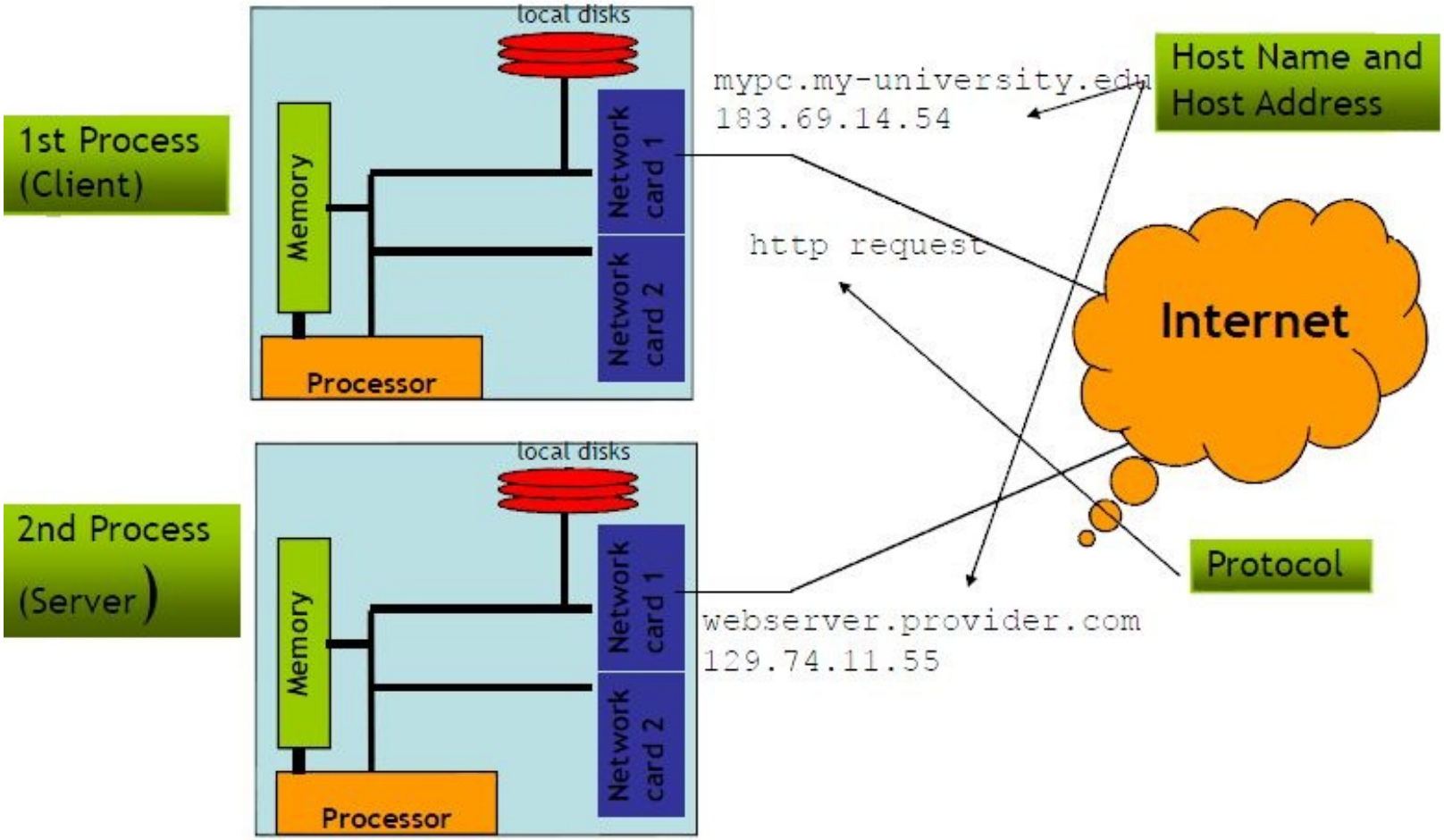
Aperçu

- Systèmes distribués
- Les principes basiques du MPI (Message Passing Interface)
  - addressage
  - démarrage
  - échange de données
  - gestion des processus
  - communication

Machines à mémoire distribuée



Communication entre différentes machines sur internet



Communication entre différentes machines sur internet

- Adressage:
  - hostname et/ou adresse IP
- Communication:
  - Basée sur des protocolse, e.g. http ou TCP/IP
- Démarrage des processus:
  - chaque processus (= application) doit être démarré séparément

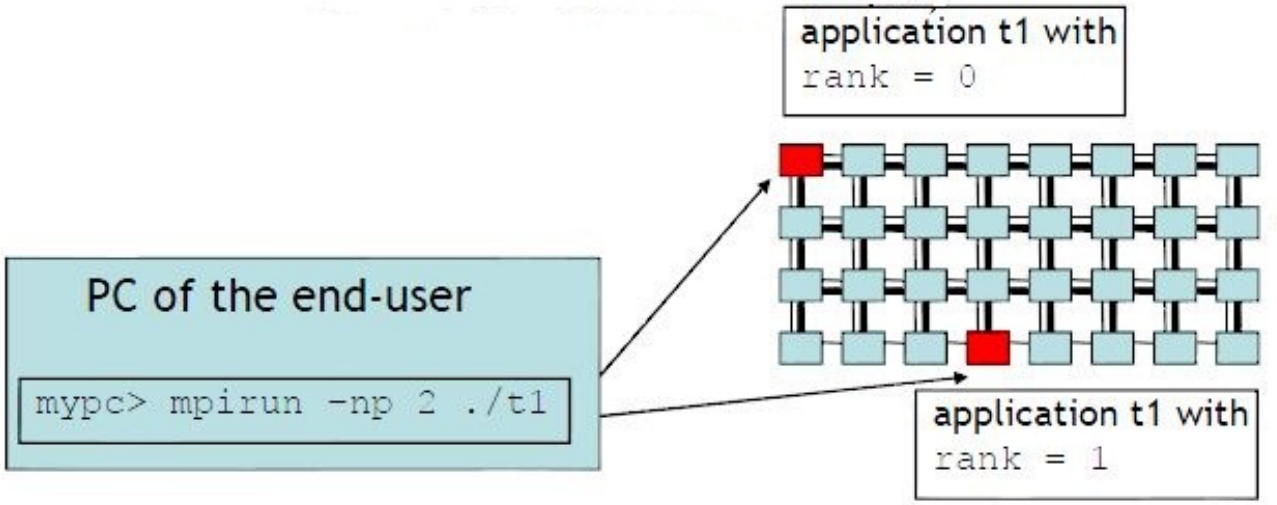
L'univers MPI

- Démarrage des processus:
  - on souhaite lancer *n*-processus qui travailleront sur le même problème
  - le mécanisme du démarrage des n-processus est défini dans la bibliothèque MPI
- Adressage:
  - chaque processus a un identifiant unique, son rang. La valeur du rang est comprise entre *0 et n-1*
- Communication:
  - MPI définit des interfaces/routines pour envoyer des données à un processus et recevoir des données d'un processus. Il n'y a pas de protocole spécifié.

Historique

- Jusqu'au début des années 90:
  - tous les vendeurs des machines parallèles ont leurs propres bibliothèques d'échange de messages
  - certaines bibliothèques sont disponibles dans le domaine publique
  - toutes ces bibliothèques ne sont pas compatibles
  - un effort considérable pour porter les codes
- Juin 1994: Version 1.0 de MPI présentée par les forum MPI
- Juin 1995: Version 1.1 (errata of MPI 1.0)
- 1997: MPI 2.0 – ajout de nouvelles fonctionnalités à MPI
- 2008: MPI 2.1
- 2009: MPI 2.2
- Juin 2015: MPI 3.1

Un exemple simple



mpirun lance l'application t1

- deux fois (comme précisé par l'argument -np)
- sur deux processeurs disponibles de la machine parallèle
- informant un processus qu'il est de rang 0
- et l'autre processus de rang 1

Un exemple simple

```
#include “mpi.h”
int main ( int argc, char **argv )
{
    int rank, size;
    MPI_Init ( &argc, &argv );
    MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
    MPI_Comm_size ( MPI_COMM_WORLD, &size );
    printf (“Hello World from process %d. Running
    processes %d\n”,
    Rank, size);
    MPI_Finalize ();
    return (0);
}
```

Les bases du MPI

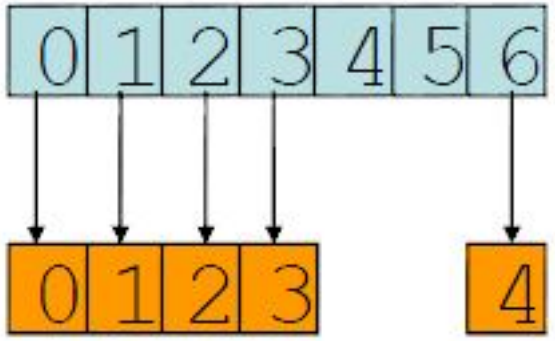
- `mpirun` lance le nombre de processus requis
  - chaque processus a un identifiant unique, son rang, compris entre 0 et n-1
  - Aucun identifiant n'est dupliqué, tous les identifiants sont utilisés
- Tous les processus lancés par `mpirun` sont organisés dans un groupe (`communicator`) appelé `MPI_COMM_WORLD`
- `MPI_COMM_WORLD` est statique
  - le nombre de processus ne peut pas changer
  - les processus participants ne peuvent pas évoluer

Les bases du MPI (II)

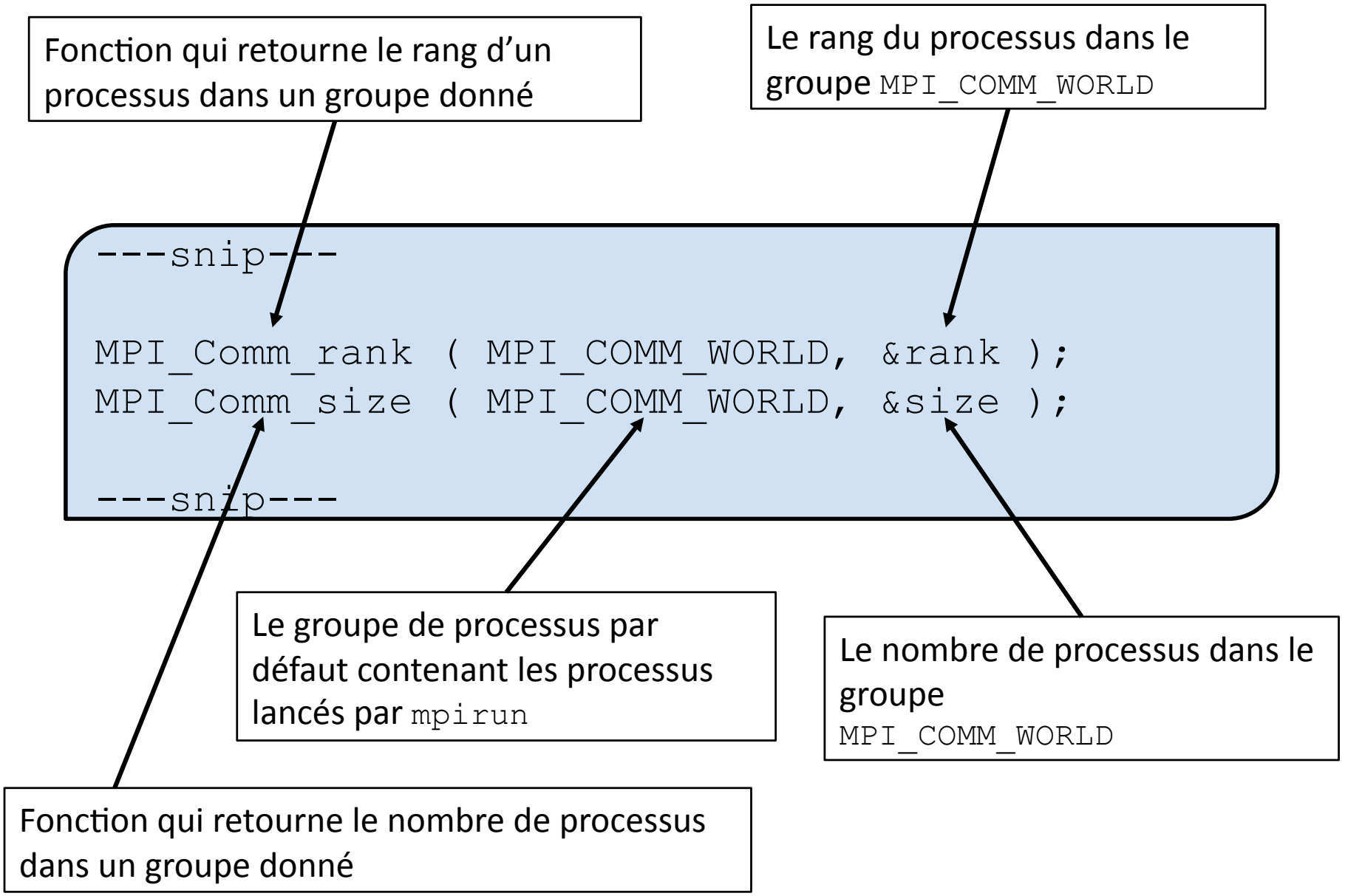
- Le rang d'un processus est toujours lié à un groupe de processus
  - e.g. un processus est identifié d'une façon unique par le couple (`rank, process group`)
- Un processus peut faire partie de plusieurs groupes
  - i.e. un processus a un rang dans chaque groupe

`MPI_COMM_WORLD`, size=7

`new process group`, size = 5



Un exemple simple



Un exemple simple

Fonction qui configure l'environnement parallèle:

- configuration de la connexion entre les processus
- configuration du groupe de processus par défaut (`MPI_COMM_WORLD`)
- ça doit être la première fonction executée dans une application parallèle

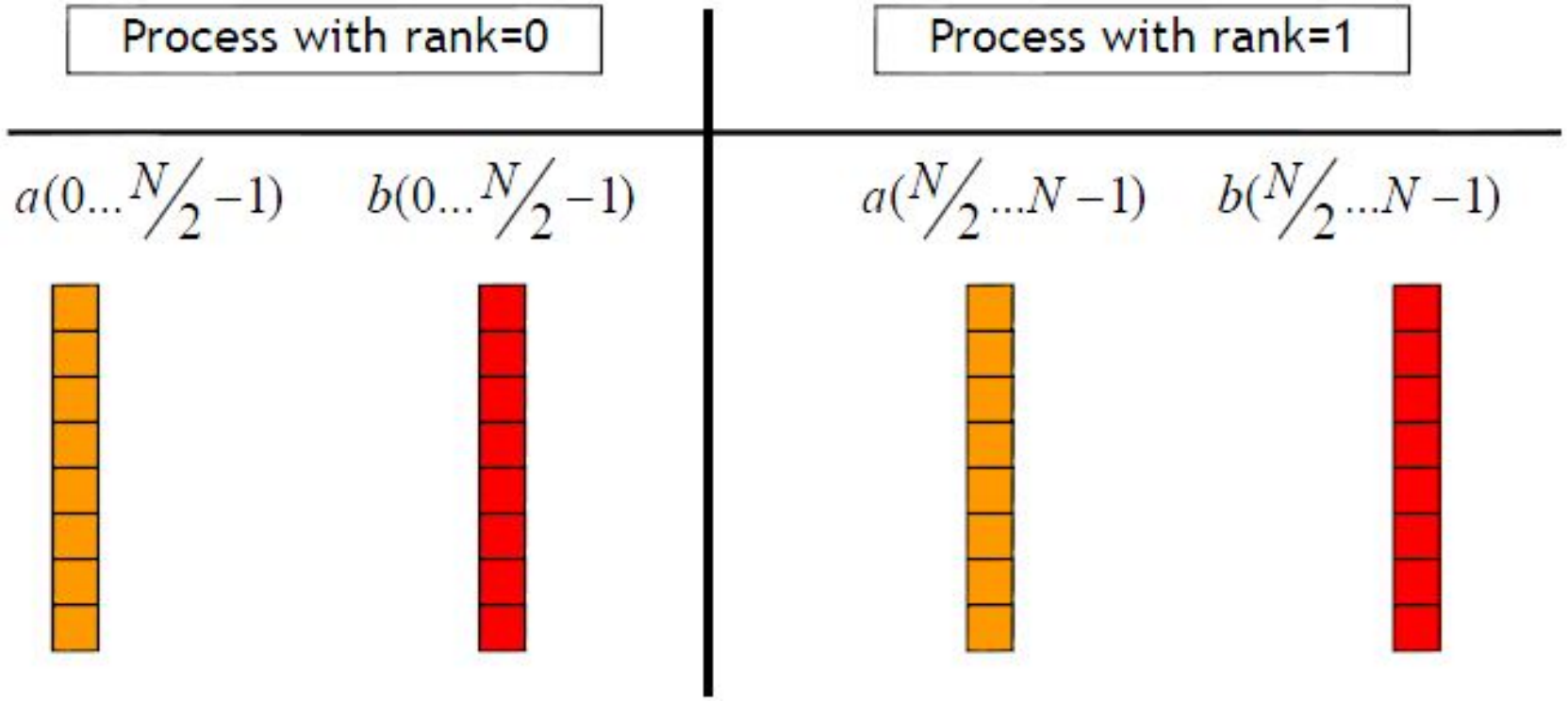
```
---snip---
MPI_Init (&argc, &argv );
---snip---
MPI_Finalize ();
---snip---
```

Fonction qui clôt l'environnement parallèle:

- ça doit être la dernière fonction executée dans une application parallèle
- arrête tous les processus

Produit scalaire de deux vecteurs

- Deux vecteurs sont distribués sur deux processeurs
  - chaque processus a la moitié du vecteur original



Produit scalaire de deux vecteurs

- La vue globale vs la vue locale des données

Process with rank=0	Process with rank=1
$a(0 \ldots N/2 - 1)$	$a(N/2 \ldots N-1)$
$a_{local}(0) \Rightarrow a(0)$	$a_{local}(0) \Rightarrow a(N/2)$
$a_{local}(1) \Rightarrow a(1)$	$a_{local}(1) \Rightarrow a(N/2+1)$
$a_{local}(2) \Rightarrow a(2)$	$a_{local}(2) \Rightarrow a(N/2+2)$
$\vdots$	$\vdots$
$a_{local}(n) \Rightarrow a(N/2 - 1)$	$a_{local}(n) \Rightarrow a(N-1)$

Produit scalaire de deux vecteurs

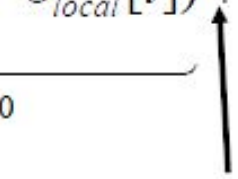
- Produit scalaire

$$s = \sum_{i=0}^{N-1} a[i] * b[i]$$

- Algorithme parallèle

$$s = \sum_{i=0}^{N/2-1} (a[i] * b[i]) + \sum_{i=N/2}^{N-1} (a[i] * b[i])$$

$$= \underbrace{\sum_{i=0}^{n} (a_{local}[i] * b_{local}[i])}_{rank=0} + \underbrace{\sum_{i=0}^{n} (a_{local}[i] * b_{local}[i])}_{rank=1}$$



  - demande une communication entre les processus

Code parallèle pour le produit scalaire

```
#include “mpi.h”

int main ( int argc, char **argv )
{
   int i, rank, size;
   double a_local[N/2], b_local[N/2];
   double s_local, s;
   MPI_Init ( &argc, &argv );
   MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
   MPI_Comm_size ( MPI_COMM_WORLD, &size );
   s_local = 0;
   for ( i=0; i<N/2; i++ ) {
   s_local = s_local + a_local[i] * b_local[i];
```

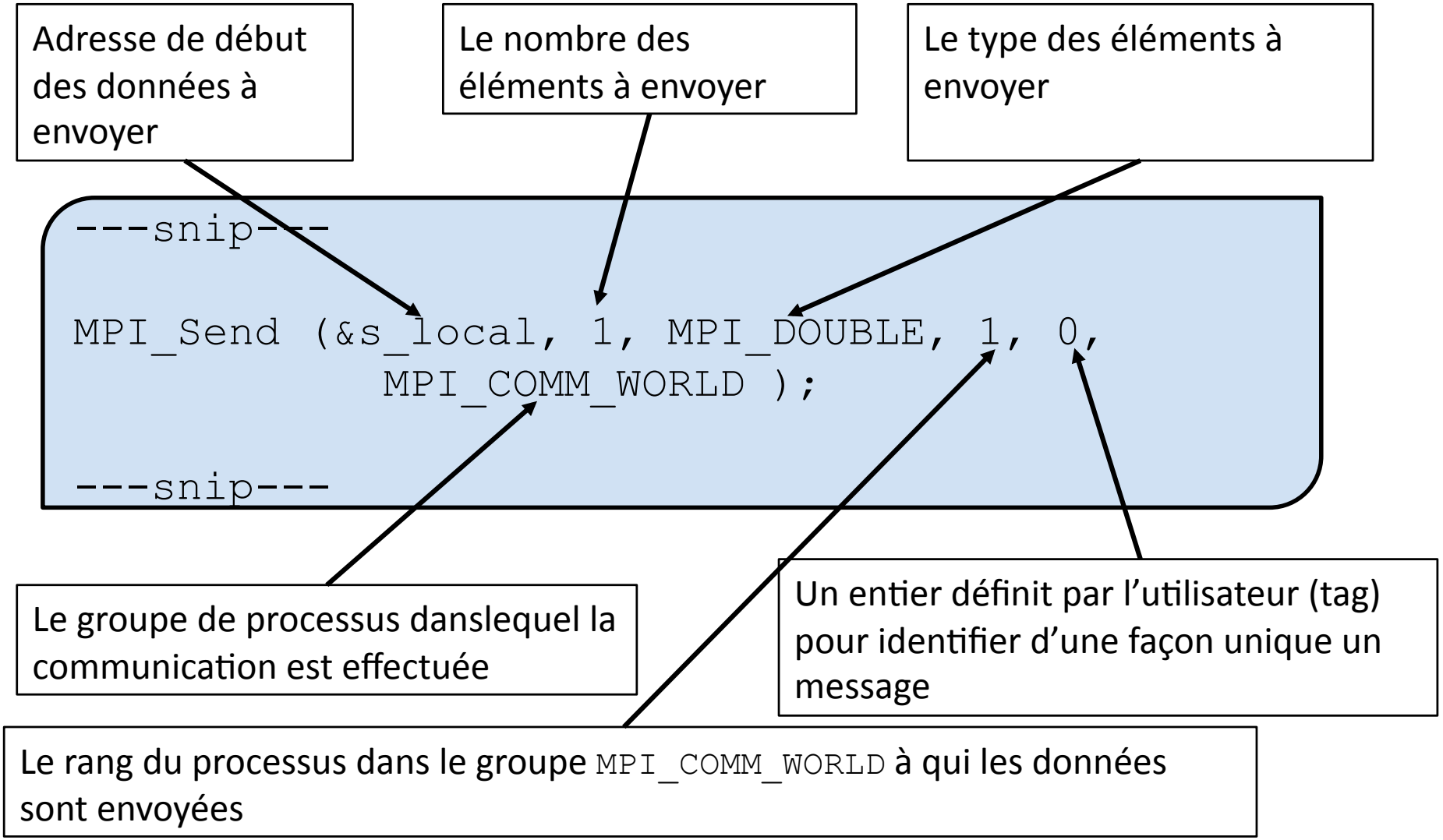
Code parallèle pour le produit scalaire

```
if ( rank == 0 ) {
      /* Send the local result to rank 1 */
      MPI_Send ( &s_local, 1,
MPI_DOUBLE, 1,          0, MPI_COMM_WORLD);
}
if ( rank == 1 ) {
      MPI_Recv ( &s, 1, MPI_DOUBLE, 0,
0,          MPI_COMM_WORLD, &status );
/* Calculate global result */
S = s + s_local;
```

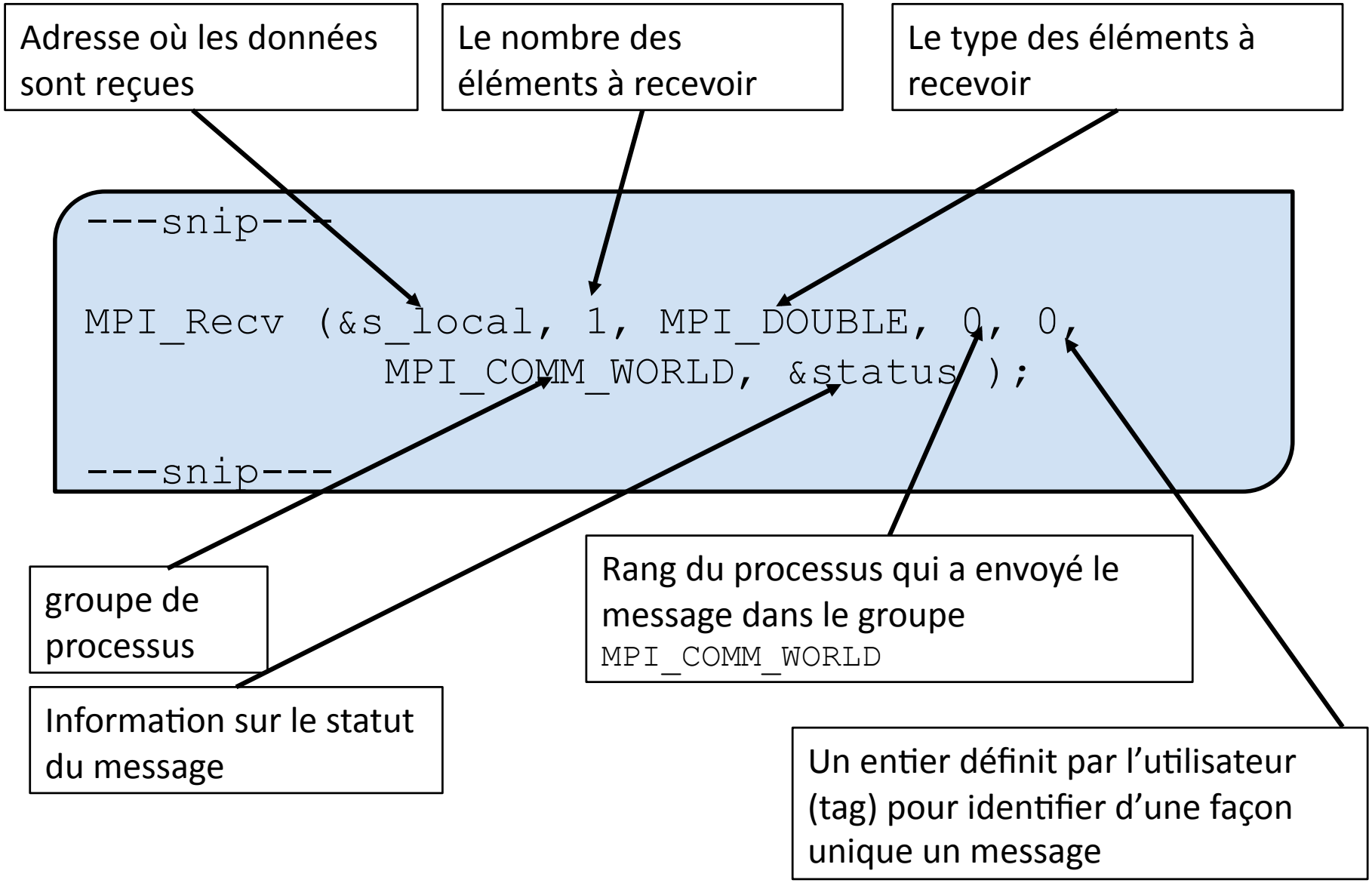
Code parallèle pour le produit scalaire

```
   /* Rank 1 holds the global result and
   sends it now to rank 0 */
   if ( rank == 0 ) {
      MPI_Recv (&s, 1, MPI_DOUBLE, 1, 1,
                  MPI_COMM_WORLD, &status );
   }
   if ( rank == 1 ) {
      MPI_Send (&s, 1, MPI_DOUBLE, 0, 1,
                  MPI_COMM_WORLD);
   }
   /* Close the parallel environment */
   MPI_Finalize ();
   return (0);
}
```

Envoi des données



Réception des données



Exemples d'erreurs(I)

- Défaut d'appariement entre envoi et réception
  - si le rang n'existe pas (e.g. `rank > size of MPI_COMM_WORLD`), la bibliothèque MPI retourne une erreur
  - si le rang existe (`0<rank<size of MPI_COMM_WORLD`) mais n'envoie pas de message=> `MPI_Recv` attend pour toujours => deadlock

```
if ( rank == 0 ) {
/* Send the local result to rank 1 */
   MPI_Send ( &s_local, 1, MPI_DOUBLE, 1,
         0, MPI_COMM_WORLD);
}
if ( rank == 1 ) {
   MPI_Recv ( &s, 1, MPI_DOUBLE, 5, 0,
         MPI_COMM_WORLD, &status );
}
```

Exemples d'erreurs (II)

- Défaut d'appariement du Tag
  - Si le tag est au dessus des valeurs authorisées (e.g. `0<tag<MPI_TAG_UB`) la bibliothèque MPI retourne une erreur
  - Si le tag dans `MPI_Recv` est différent de celui du `MPI_Send` => `MPI_Recv` attend pour toujours => deadlock

```
if ( rank == 0 ) {
/* Send the local result to rank 1 */
MPI_Send ( &s_local, 1, MPI_DOUBLE, 1, 0,
          MPI_COMM_WORLD);
}
if ( rank == 1 ) {
MPI_Recv ( &s, 1, MPI_DOUBLE, 0, 18,
          MPI_COMM_WORLD, &status );
}
```

Ce qu'on a appris jusqu'ici

- Six fonctions MPI sont suffisantes pour programmer des machines à mémoire distribuée

```
MPI_Init(int *argc, char ***argv);
MPI_Finalize ();

MPI_Comm_rank (MPI_Comm comm, int *rank);
MPI_Comm_size (MPI_Comm comm, int *size);

MPI_Send (void *buf, int count, MPI_Datatype dat,
int dest, int tag, MPI_Comm comm);
MPI_Recv (void *buf, int count, MPI_Datatype dat,
int source, int tag, MPI_Comm comm, MPI_Status
*status);
```

Pourquoi ne pas s'arrêter là?

- Performance
  - besoin de fonctions parmettant d'exploiter au mieux les capacités des architectures
  - Besoin de fonctions pour les modèles de communication typiques
- Facilité d'utilisation
  - besoin de fonctions pour simplifier les tâches récurrentes
  - besoin de fonctions pour simplifier la gestion des applications parallèles

Pourquoi ne pas s'arrêter là?

- Performance
  - communication point à point
  - communications collectives
  - les types de données dérivés …
- Facilité d'utilisation
  - fonctions de groupement de processus
  - gestion de processus
  - la gestion des erreurs ….

Quelques liens utiles

- MPI Forum:
  - http://www.mpi-forum.org
- Open MPI:
  - http://www.open-mpi.org
- MPICH:
  - http://www-unix.mcs.anl.gov/mpi/mpich/