Introduction au standard MPI

Jean-Baptiste Keck

Laboratoire Jean Kuntzmann - Grenoble

1er février 2017

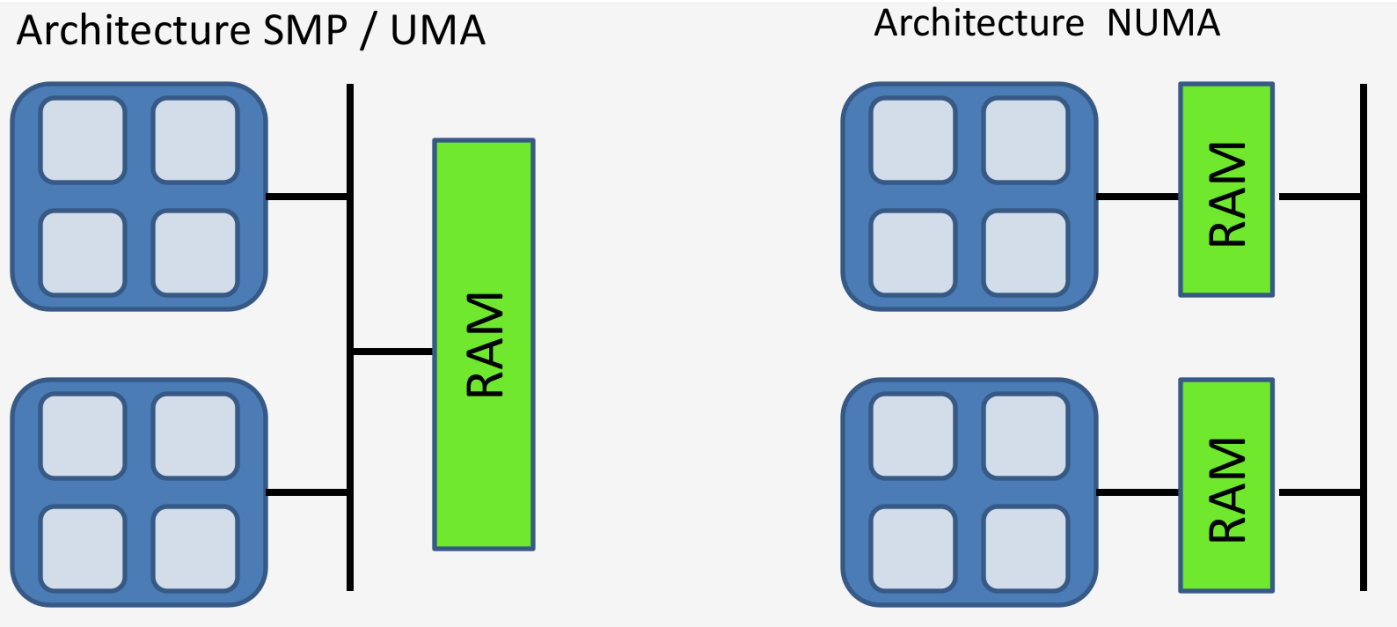
Table des matières

1. Rappel sur les architectures distribuées
2. Parallélisme par passage de messages
3. Gestion du contexte de communication et des processus
4. Passage de messages
5. Introduction à la notion de communicateurs de topologie
6. Exemple : Multiplication matrice-vecteur en parallèle

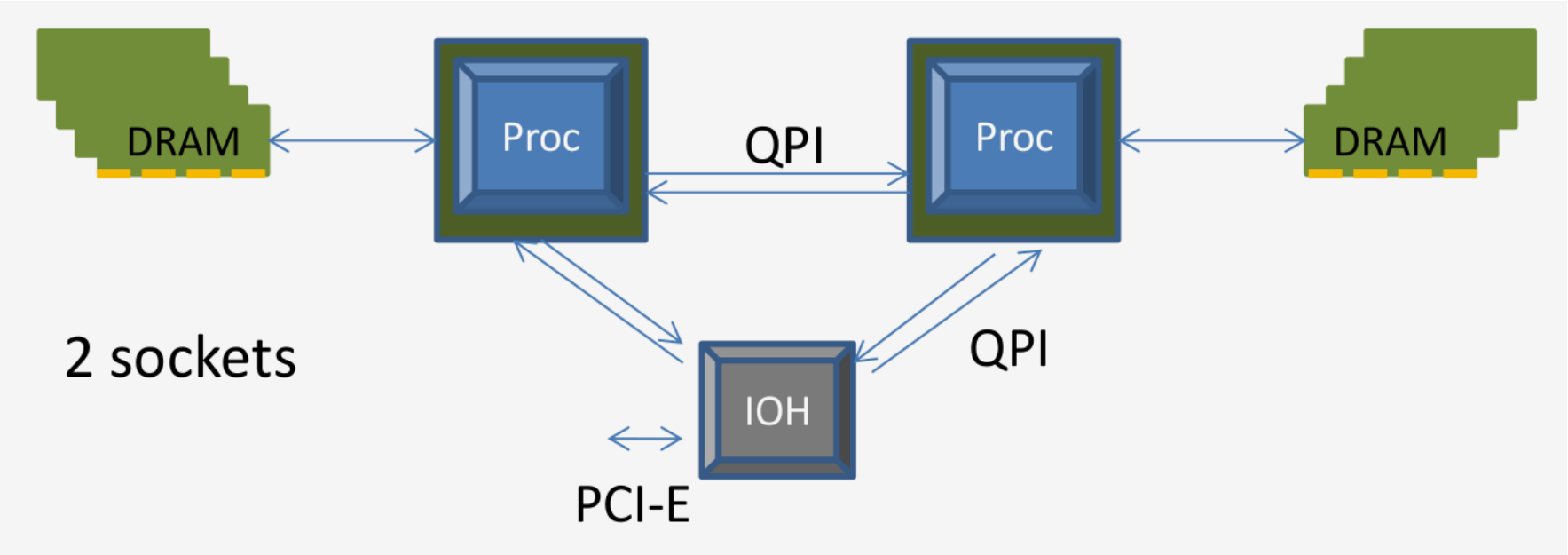
1. Rappel sur les architectures distribuées
   - Nœud et cluster de calcul
   - Technologies réseau
2. Parallélisme par passage de messages
   - Modèle par passage de messages
   - Standard actuel : MPI (1993)
   - MPI en pratique
3. Gestion du contexte de communication et des processus
   - Communicateur par défaut
   - Fonctions de base
   - Créer des sous-communicateurs
4. Passage de messages
   - Les différents types de communications
   - Communications point à point
   - Communications collectives
5. Introduction à la notion de communicateurs de topologie
6. Exemple : Multiplication matrice-vecteur en parallèle

Modèles d'architectures partagées

- Architecture UMA (Uniform Memory Access) : Temps d'accès à un emplacement mémoire quelconque identique pour les les processeurs.
- Architecture NUMA (Non Uniform Memory Access) : Les processeurs peuvent accéder à la totalité de la mémoire les mais temps d'accès peuvent différer selon la distance cœur-mémoire.

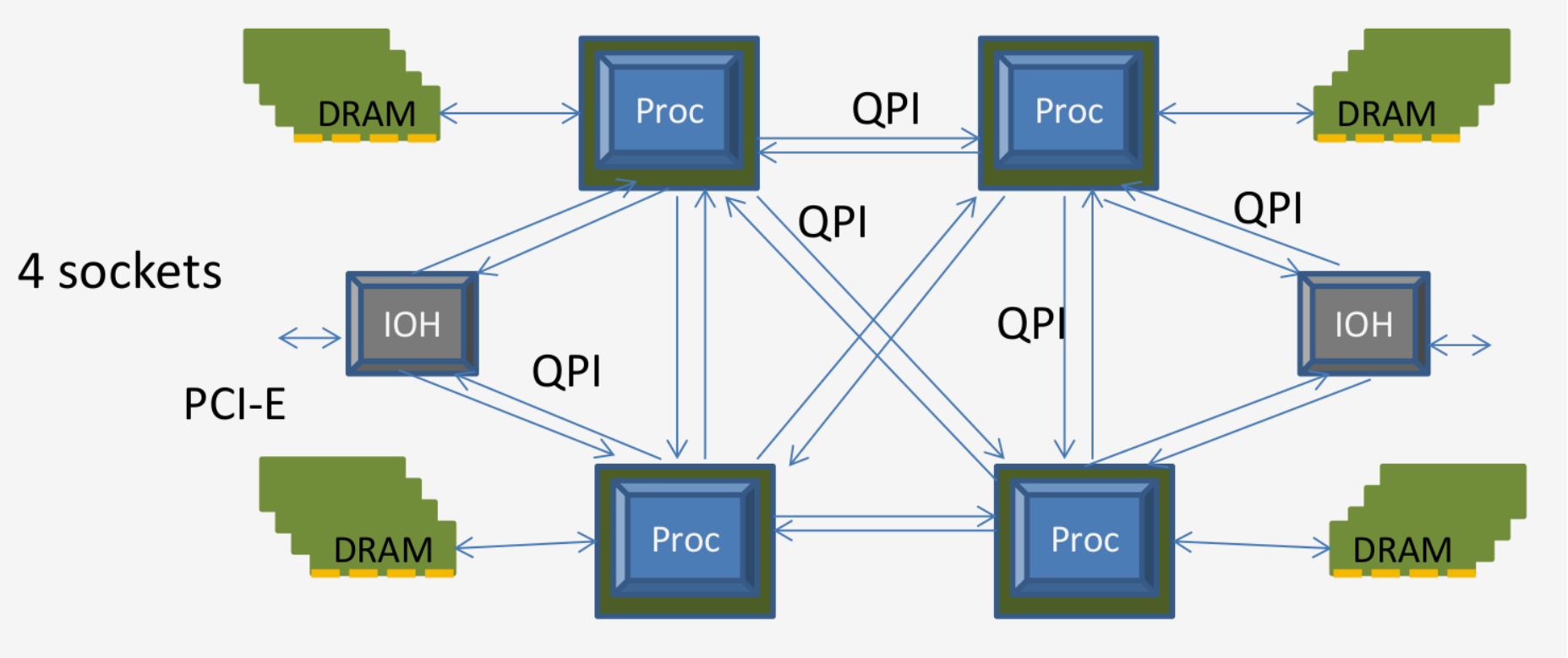


Machines à plusieurs sockets



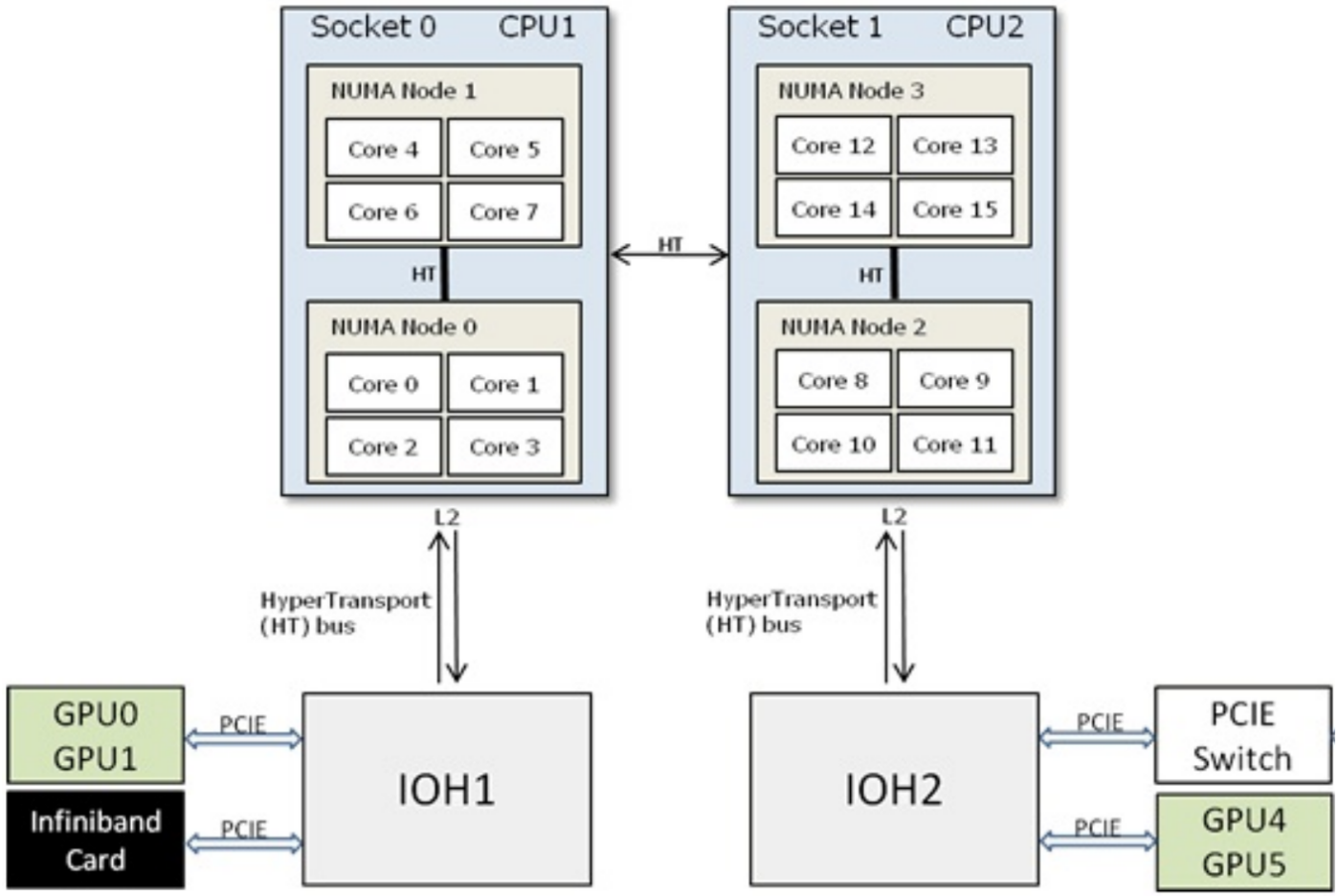
Nœud NUMA à deux sockets

Machines à plusieurs sockets

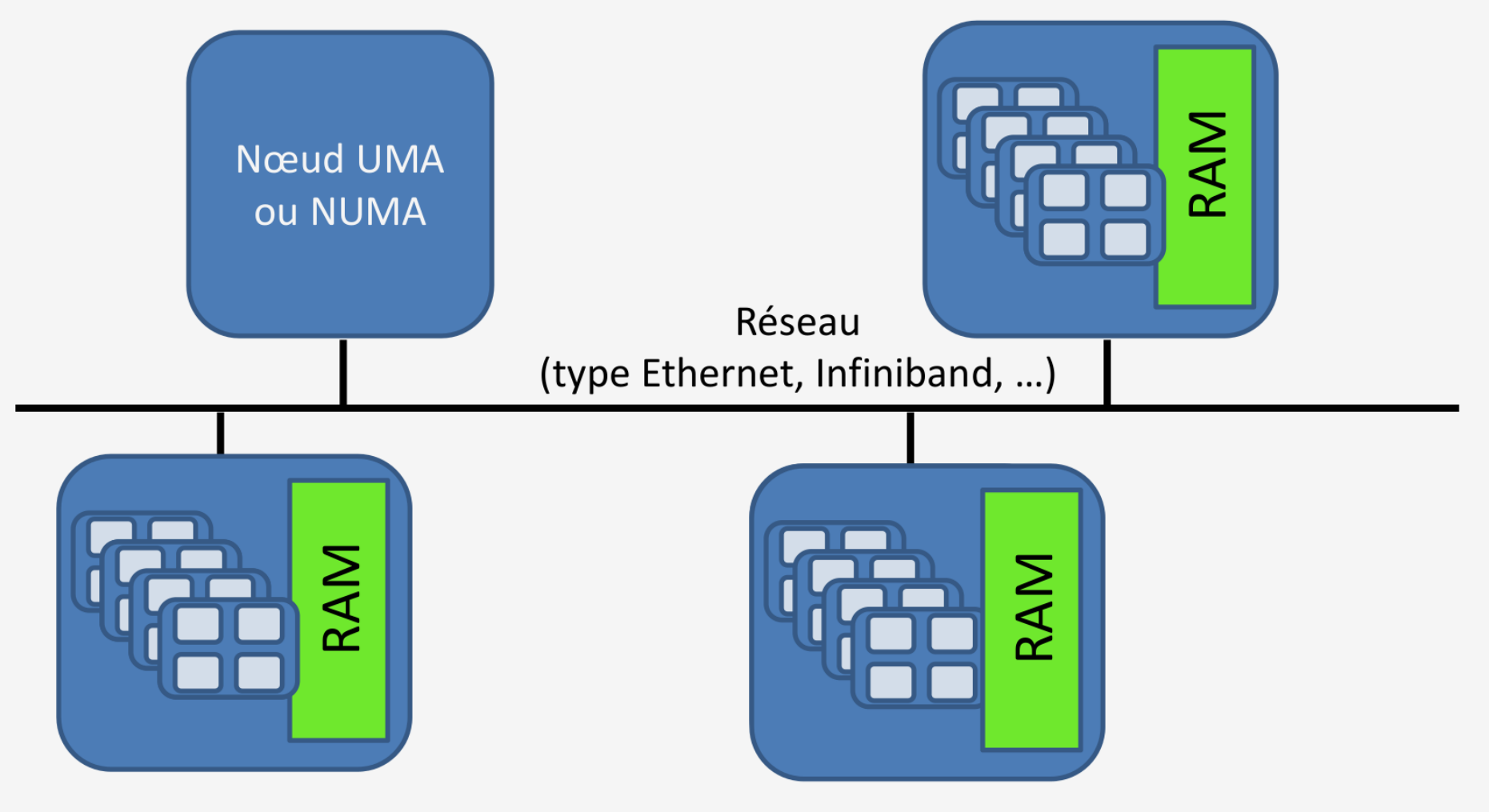


Nœud NUMA à quatre sockets

Vue globale d'un nœud de calcul

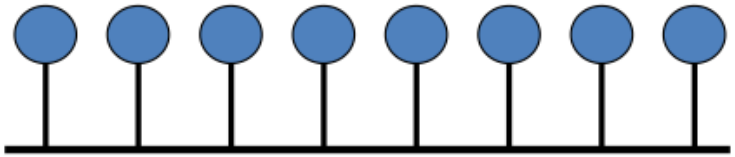


Architecture à mémoire distribuée

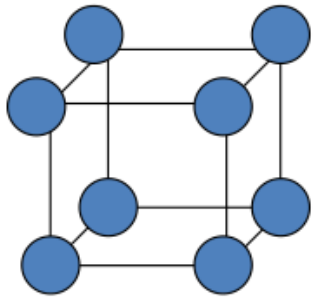


Topologie des réseaux

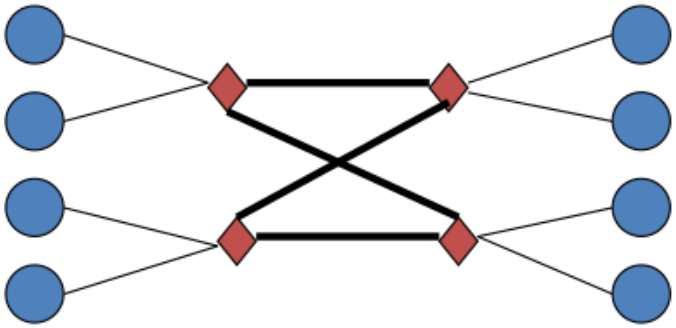
Bus



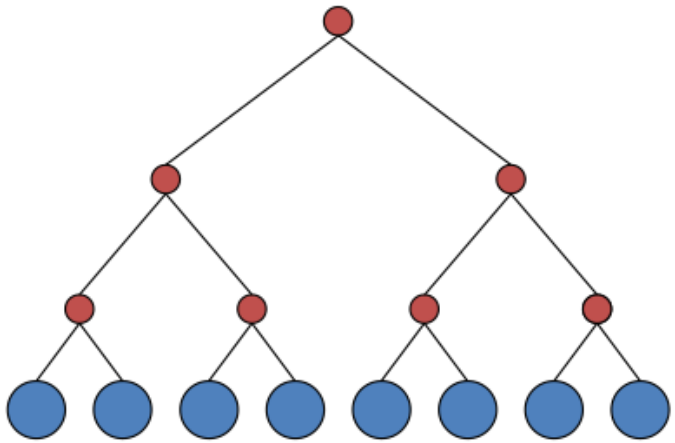
Hypercube



Réseau à plusieurs étages



Arbre



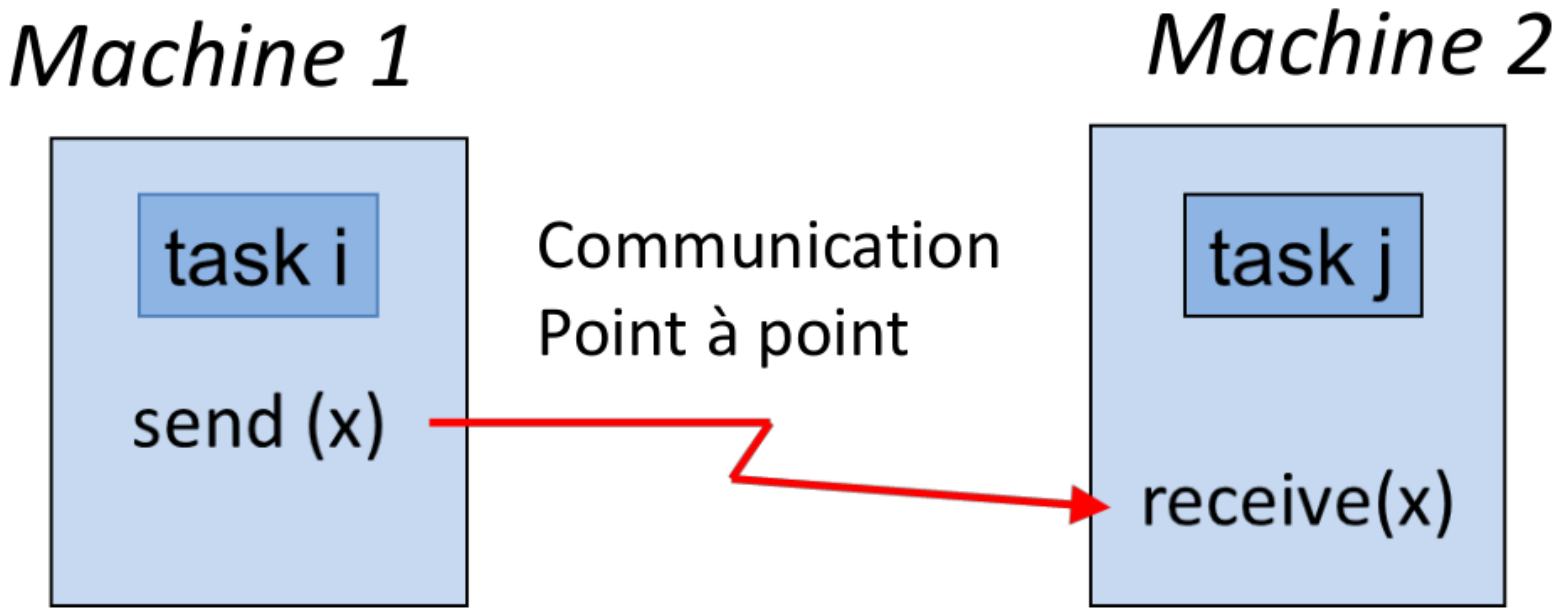
Technologies réseau pour le HPC

Technologie	Vendeur	Latence MPI usec, short msg	Bande passante
NUMAlink 5	SGI	~1	15 Go/s
QsNet II	Quadrics	~1	900 Mo/s
Infiniband	Mellanox, Qlogic, (Voltaire)	~1	SDR (2.5 Gb/s), DDR (20 Gb/s), QDR (40 Gb/s), FDR (56Gb/s)
Myri-10G	Myricom	~2	1.2 Go/s
Ethernet 10 Gb		~10	10 Gb/s
Ethernet 1 Gb		~50	1 Gb/s

1. Rappel sur les architectures distribuées
   - Nœud et cluster de calcul
   - Technologies réseau
2. Parallélisme par passage de messages
   - Modèle par passage de messages
   - Standard actuel : MPI (1993)
   - MPI en pratique
3. Gestion du contexte de communication et des processus
   - Communicateur par défaut
   - Fonctions de base
   - Créer des sous-communicateurs
4. Passage de messages
   - Les différents types de communications
   - Communications point à point
   - Communications collectives
5. Introduction à la notion de communicateurs de topologie
6. Exemple : Multiplication matrice-vecteur en parallèle

Modèle par passage de messages

- Un ensemble de tâches travaillent sur leur mémoire locale (processus lourds) et s'exécutent sur une ou plusieurs machines.
- Ces tâches échangent des données via des communication de messages.
- Un transfert de données est la résultante d'une coopération entre tâches.
- C'est au programmeur d'expliciter toutes les communications.
- Les messages peuvent être synchrones ou asynchrones.



Standard actuel : MPI (1993)

MPI (Message Passing Interface) est un standard définissant une bibliothèque de fonctions pour le passage de messages.

- Compatible `C`,`C++`, `Fortran`
- Il existe des surcouches pour `Python`, `Java`, `Matlab`, `Ocaml`, `R`, ...
- Standard pour la communication inter-nœuds.
- Performance : Chaque fabricant optimise son implémentation MPI pour son matériel.
- Fournit également des mécanismes pour faire du RDMA (Remote Direct Memory Access).

Les communications entre processus sont optimisées :

- Entre deux nœuds : Communication par le réseau.
- Entre deux cœurs d'un même nœud : Les implémentation choisissent en général le passage par la mémoire partagée.

MPI en pratique

Le standard `MPI` est définit par une norme :

- MPI-1 (1993), MPI-2.1 (2008), MPI-2.2 (2009), MPI-3 (2012)
- Dans cette présentation on ne fera pas de différence entre les différents standards.
- Le standard de référence sera `MPI-3` mais la plus part des fonctions présentées sont issus des standards précédents.

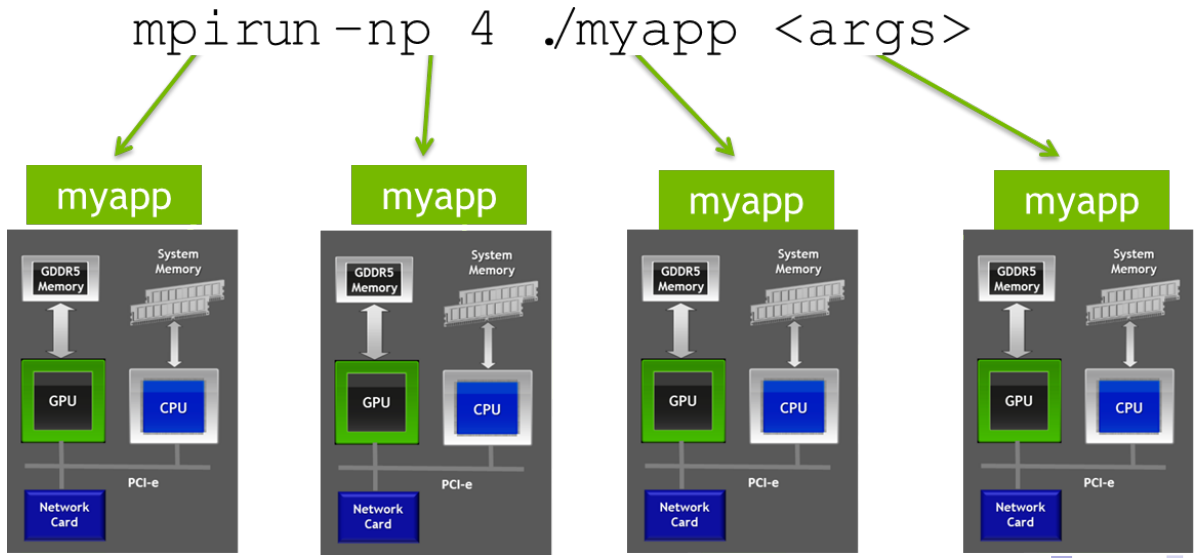
Différentes implémentations/librairies :

- Open source : `OpenMPI`, `MPICH`.
- Propriétaires : Spécifique à une machine et/ou à une architecture (optimisée par le constructeur).

En pratique : `MPI` est une interface portable mais les fonctionnalités et performances peuvent varier d'une implémentation à l'autre, en fonction de la machine, du compilateur, du langage utilisé, ...

Notions clés

- Un processus `MPI` est virtuel. Le système d'exploitation est en charge de les placer sur des processeurs et/ou des cœurs différents ou identiques.
- Un processus `MPI` est un processus lourd et est donc complètement isolé de tous les autres processus.
- Chaque processus a donc ses propres variables et n'ont pas accès directement aux autres processus.
- Le partage de données se fait donc par envois et receptions explicites de messages.



Premiers pas avec MPI

- Programme : Le programme devra créer et organiser un contexte de communication, répartir les tâches et gérer les échanges de messages.
- Compilation : Compiler le programme avec les sur couches compilateur fournies (`mpicc, mpicxx, mpifort, ...`) ou utiliser des outils de compilation dédiés du type `CMake` ou `GNU autotools`.
- Execution : Distribuer l'exécutable sur les nœuds distants, répartir la charge les cœurs physiques des machines, et faire exécuter le programme sur chacun des processus.

  La façon la plus simple de lancer un exécutable `MPI` est :
  `mpirun -np [nprocess] -host [noeuds] [executable] [options executable]`

  Il existe de nombreuse options pour répartir la charge. On peut également faire exécuter différents executables en même temps.

Comment écrire un programme minimal en MPI

Quelque soit le langage, la structure d'un programme `MPI` ressemblera en général à la suivante :

1. Inclusion du module `MPI` (header, package, ...)
2. Initialisation du contexte `MPI`
3. Organisation des processus et création de contextes de communications.
4. Calculs locaux et échanges de messages entre processus.
5. Fin du contexte `MPI`.

Dans la suite de cette présentation on donnera des exemples en langage `C`, mais le passage à un autre langage n'est qu'une question de sémantique.

Programme minimal

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int world_size, world_rank, name_len;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    /* Initialize the MPI environment */
    MPI_Init (argc, argv);

    /* Get the rank and the number of processes */
    MPI_Comm_rank (MPI_COMM_WORLD , &world_rank);
    MPI_Comm_size (MPI_COMM_WORLD , &world_size);

    /* Get the name of the processor */
    MPI_Get_processor_name (processor_name, &name_len);

    /* Print a hello world message on each process */
    printf ("Hello world from processor %s, rank %d out of %d processors\n",
            processor_name, world_rank, world_size);

    /* Finalize the MPI environment */
    MPI_Finalize ();
}
```

1 Rappel sur les architectures distribuées
- Nœud et cluster de calcul
- Technologies réseau

2 Parallélisme par passage de messages
- Modèle par passage de messages
- Standard actuel : MPI (1993)
- MPI en pratique

3 Gestion du contexte de communication et des processus
- Communicateur par défaut
- Fonctions de base
- Créer des sous-communicateurs

4 Passage de messages
- Les différents types de communications
- Communications point à point
- Communications collectives

5 Introduction à la notion de communicateurs de topologie

6 Exemple : Multiplication matrice-vecteur en parallèle

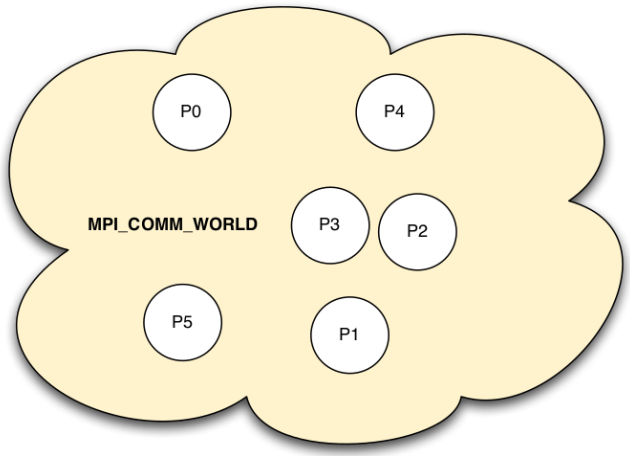
Organisation des processus

Un communicateur est un ensemble de processus susceptibles de communiquer entre eux.

Un communicateur est caractérisé par :

- Son nom
- Un groupe de processus ordonnés
  1. Taille : nombre de processus qu'il contient.
  2. Rangs : identifient chaque processus dans le communicateur.

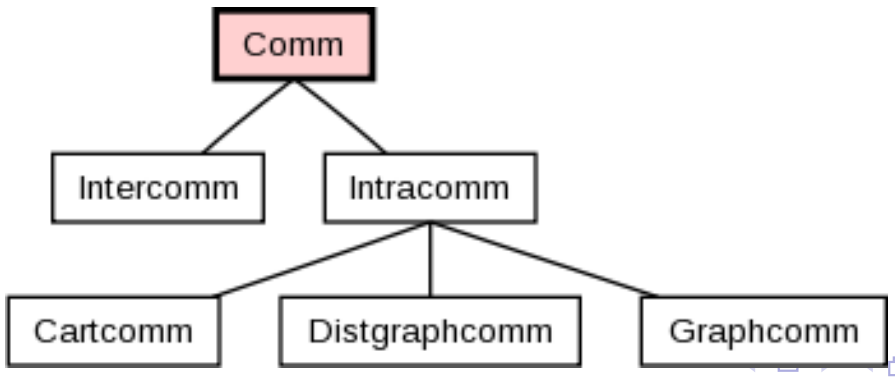
Le communicateur par défaut est MPI_COMM_WORLD :



Récupérer des informations sur un communicateur

```
/* Récuperer le nom du communicateur */
int MPI_Comm_get_name (MPI_Comm comm, char *comm_name, int *resultlen);

/* Récuperer des informations sur le communicateur */
int MPI_Comm_get_info (MPI_Comm comm, MPI_Info *info_used);

/* Récuperer le groupe de processus associé au communicateur */
int MPI_Comm_group (MPI_Comm comm, MPI_Group *group);

/* Récuperer le rang du processus dans le communicateur */
int MPI_Comm_rank (MPI_Comm comm, int *rank);

/* Récuperer le nombre de processus présents dans le communicateur */
int MPI_Comm_size (MPI_Comm comm, int *size);
```

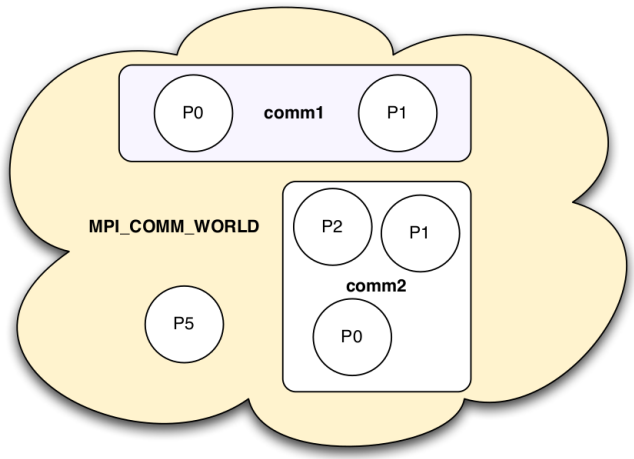
Les différents types de communicateurs :



Sous-communicateurs

On peut créer des sous-communicateurs :

- MPI_COMM_WORLD est le communicateur par défaut, il contient tous les processus attribués à l'application. Il est créé lors de l'initialisation du contexte (`MPI_Init`).
- Les processus ne peuvent communiquer que s'ils partagent un même communicateur.
- La création de sous communicateurs permettra de :
  1. Réduire la portée des communications à un sous ensemble de processus.
  2. Classer et organiser les processus (topologies de grilles ou de graphes).

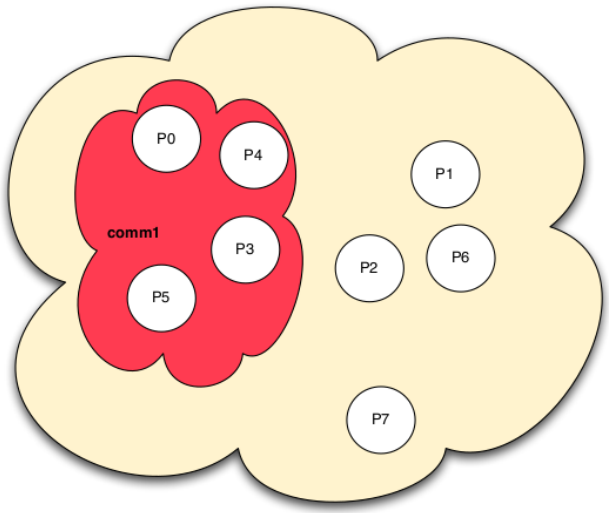


Créer un communicateur à partir d'un groupe

On peut créer des communicateurs à partir d'un groupe de processus :

```
/* Version collective au groupe associé à comm */
int MPI_Comm_create (MPI_Comm comm, MPI_Group group,
                     MPI_Comm *newcomm);

/* Version locale au groupe associé à newcomm */
int MPI_Comm_create_group (MPI_Comm comm, MPI_Group group, int tag,
                           MPI_Comm *newcomm);
```

Les processus non concernés obtiennent : MPI_COMM_NULL



Opérations sur les groupes

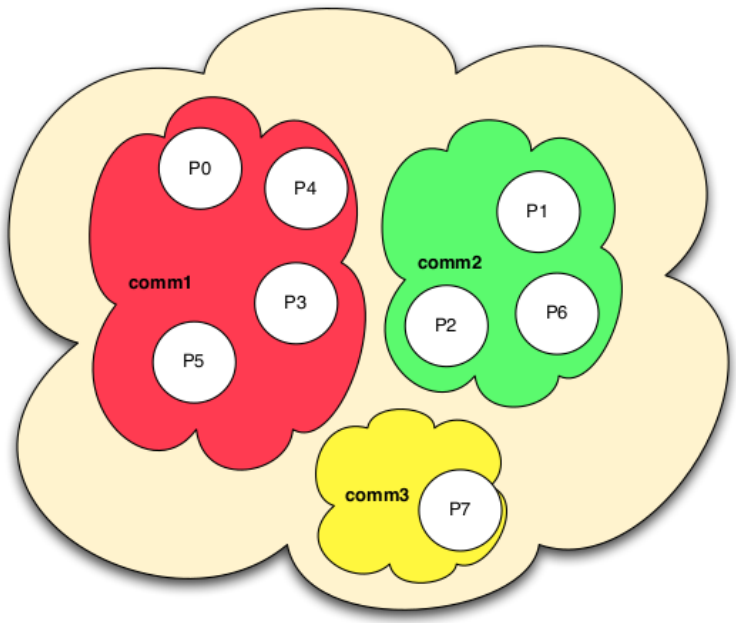
Pour créer un groupe, on peut utiliser les routines suivantes :

```
/* Inclusion et exclusion de processus */
int MPI_Group_incl (MPI_Group group, int n, const int ranks[],
                    MPI_Group *newgroup);
int MPI_Group_excl (MPI_Group group, int n, const int ranks[],
                    MPI_Group *newgroup);

/* Inclusion et exclusion de processus par fourchette (début, fin ,incrément) */
int MPI_Group_range_incl (MPI_Group group, int n, int ranges[][3],
                          MPI_Group *newgroup);
int MPI_Group_range_excl (MPI_Group group, int n, int ranges[][3],
                          MPI_Group *newgroup);

/* Intersection , union, différence */
int MPI_Group_union (MPI_Group lhs, MPI_Group rhs, MPI_Group *newgroup);
int MPI_Group_difference (MPI_Group lhs, MPI_Group rhs, MPI_Group *newgroup);
int MPI_Group_intersection (MPI_Group lhs, MPI_Group rhs,
                            MPI_Group *newgroup);
```

Partitionner un communicateur

On peut partitionner un communicateur en plusieurs sous-communicateurs d'un coup :

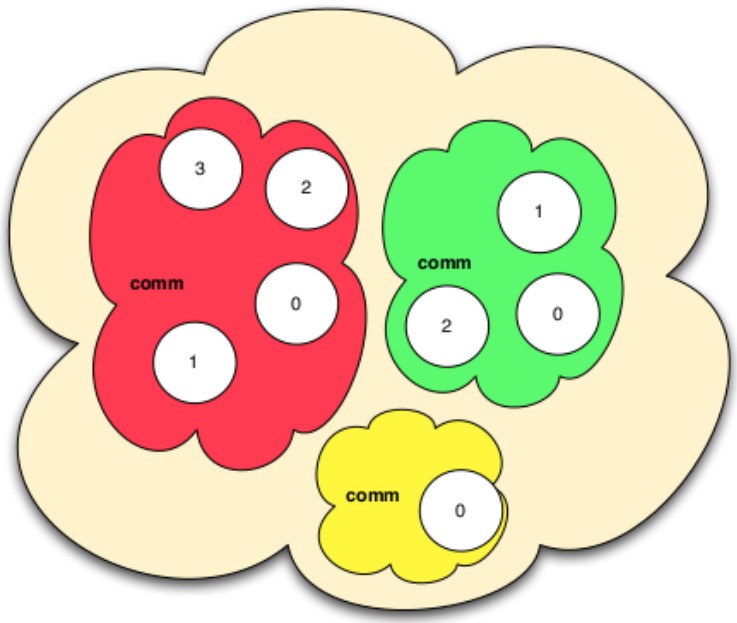
```
/* Partitionne le communicateur par couleur */
int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm *newcomm);

/* Partitionne le communicateur par type de processus (MPI_COMM_TYPE_SHARED) */
int MPI_Comm_split_type (MPI_Comm comm, int split_type, int key,
                         MPI_Info info, MPI_Comm *newcomm);
```



Partitionner un communicateur

On peut partitionner un communicateur en plusieurs sous-communicateurs d'un coup :

```
/* Partitionne le communicateur par couleur */
int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm *newcomm);

/* Partitionne le communicateur par type de processus (MPI_COMM_TYPE_SHARED) */
int MPI_Comm_split_type (MPI_Comm comm, int split_type, int key,
                         MPI_Info info, MPI_Comm *newcomm);
```



Exemple de partition en processus pairs/impairs

```
#include <mpi.h>

int main(int argc, char** argv) {
    int world_size, world_rank, name_len;
    int local_rank, local_size;
    MPI_Comm local_comm;

    /* Initialisation de l'environnement MPI */
    MPI_Init (argc, argv);
    MPI_Comm_rank (MPI_COMM_WORLD , &world_rank);
    MPI_Comm_size (MPI_COMM_WORLD , &world_size);

    /* Partition en processus pairs et impairs */
    {
        int color = world_rank%2;
        int key   = world_rank/2;
        MPI_Comm_split (MPI_COMM_WORLD , color, key, &local_comm);
        MPI_Comm_rank (local_comm, &local_rank);
        MPI_Comm_size (local_comm, &local_size);
    }

    /* Finalisation */
    MPI_Comm_free (&local_comm);
    MPI_Finalize ();
}
```

1 Rappel sur les architectures distribuées
- Nœud et cluster de calcul
- Technologies réseau

2 Parallélisme par passage de messages
- Modèle par passage de messages
- Standard actuel : MPI (1993)
- MPI en pratique

3 Gestion du contexte de communication et des processus
- Communicateur par défaut
- Fonctions de base
- Créer des sous-communicateurs

4 Passage de messages
- Les différents types de communications
- Communications point à point
- Communications collectives

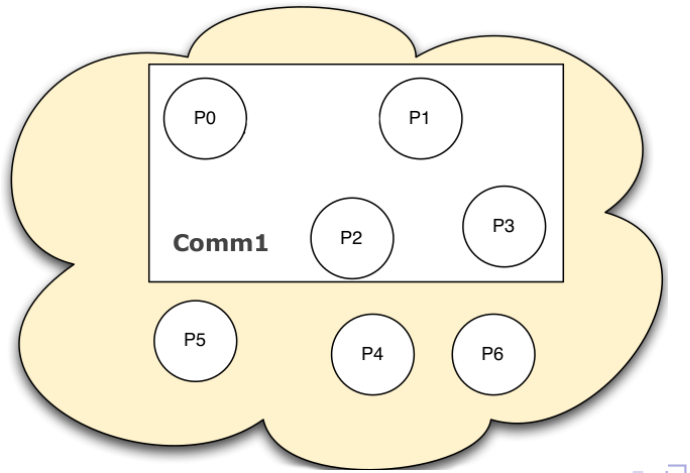
5 Introduction à la notion de communicateurs de topologie

6 Exemple : Multiplication matrice-vecteur en parallèle

Communication entre les processus

Étant donné un communicateur, on peut classer les communication en différentes catégories selon les processus sources et les processus destinataires du message :

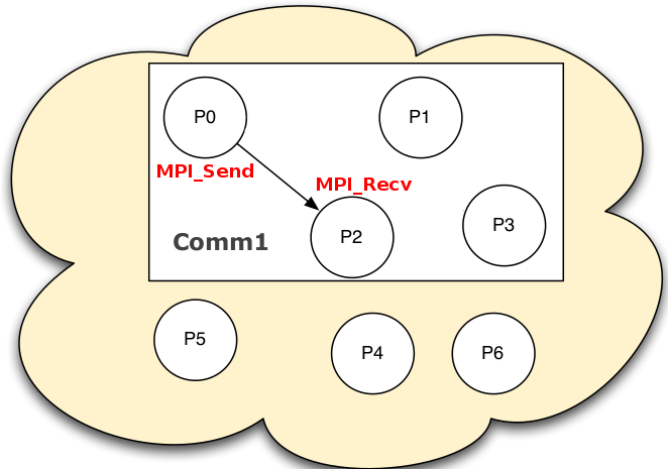
- One-to-one : Communication point à point entre deux processus.
- One-to-all : Communication collective d'un processus source vers tous les autres.
- All-to-one : Communication collective de tous les processus vers un processus cible.
- All-to-all : Communication collective de tous les processus vers tous les processus.



Communication point à point

On peut envoyer des données de n'importe quel source vers n'importe quelle destination dans un même communicateur :

```
/* Envoi de 'count' données de type 'datatype' à destination du processus de rang 'dest'
 * dans le communicateur 'comm' */
int MPI_Send (const void *buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm);

/* Reception d'au maximum 'count' données de type 'datatype' venant du processus 'src'
 * dans le communicateur 'comm' */
int MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Status *status);
```



Communication point à point

```
int MPI_Send (const void *buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm);

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

- La source et le tag permettent de différencier des messages successifs.
- On peut recevoir de n'importe quelle source et/ou de n'importe quel tag en utilisant MPI_ANY_SOURCE et MPI_ANY_TAG
- On peut récupérer la source, le tag, le nombre d'éléments reçus et les erreurs éventuelles grâce à la structure **MPI_Status**.
- On peut aussi ignorer le status avec MPI_STATUS_IGNORE.
- On peut utiliser les données directement après l'appel.
- Attention : Ces appels sont bloquants → deadlock.
  Il faut que chaque processus puisse atteindre sont `Send` ou son `Recv`.

Communication point à point non bloquante (Immediate)

```
int MPI_Isend (const void *buf, int count, MPI_Datatype datatype, int dest,
               int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm, MPI_Request *request);
```

- A la place d'un **MPI_Status** on obtient un **MPI_Request**.
- Plus de risque de deadlocks.
- On ne peut pas utiliser les données directement après l'appel.
- On peut tester ou attendre la complétion de la requête avec les fonctions suivantes :

```
int MPI_Wait (MPI_Request *request, MPI_Status *status);
int MPI_Waitall (int count, MPI_Request array_of_requests[],
                 MPI_Status *array_of_statuses);
int MPI_Waitany (int count, MPI_Request array_of_requests[],
                 int *index, MPI_Status *status);
int MPI_Waitsome (int incount, MPI_Request array_of_requests[],
                  int *outcount, int array_of_indices[],
                  MPI_Status array_of_statuses[]);
```

Communication point à point non bloquante (Immediate)

```
int MPI_Isend (const void *buf, int count, MPI_Datatype datatype, int dest,
               int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm, MPI_Request *request);
```

- A la place d'un **MPI_Status** on obtient un **MPI_Request**.
- Plus de risque de deadlocks.
- On ne peut pas utiliser les données directement après l'appel.
- On peut tester ou attendre la complétion de la requête avec les fonctions suivantes :

```
int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Testall (int count, MPI_Request array_of_requests[], int *flag,
                 MPI_Status array_of_statuses[]);
int MPI_Testany (int count, MPI_Request array_of_requests[], int *index,
                 int *flag, MPI_Status *status);
int MPI_Testsome (int incount, MPI_Request array_of_requests[],
                  int *outcount, int array_of_indices[],
                  MPI_Status array_of_statuses[]);
```

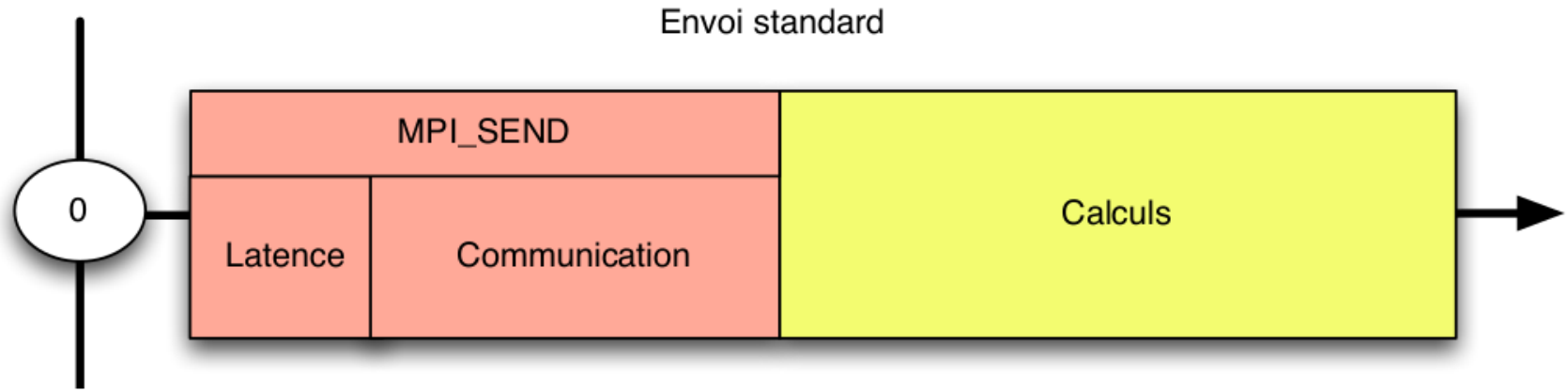
Communication point à point non bloquante (Immediate)

```
int MPI_Isend (const void *buf, int count, MPI_Datatype datatype, int dest,
               int tag, MPI_Comm comm, MPI_Request *request);

int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int source,
               int tag, MPI_Comm comm, MPI_Request *request);
```

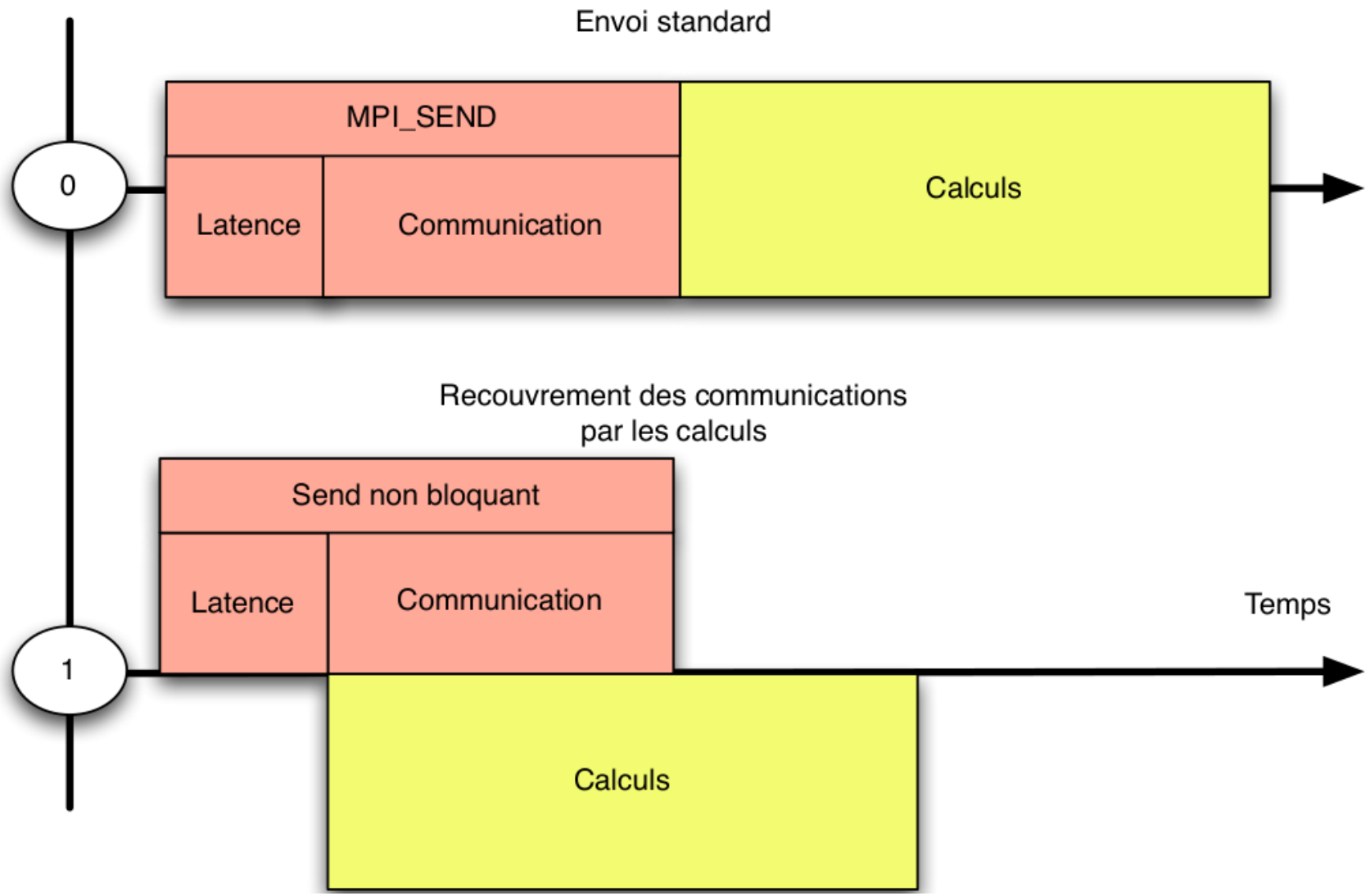
- A la place d'un **MPI_Status** on obtient un **MPI_Request**.
- Plus de risque de deadlocks.
- On ne peut pas utiliser les données directement après l'appel.
- On peut tester ou attendre la complétion de la requête avec les fonctions suivantes :

```
int MPI_Request_get_status (MPI_Request request, int *flag,
                            MPI_Status *status);
int MPI_Request_free (MPI_Request *request);
```

Comparaison des deux modes de communication



Comparaison des deux modes de communication



Les types prédéfinis par le standard

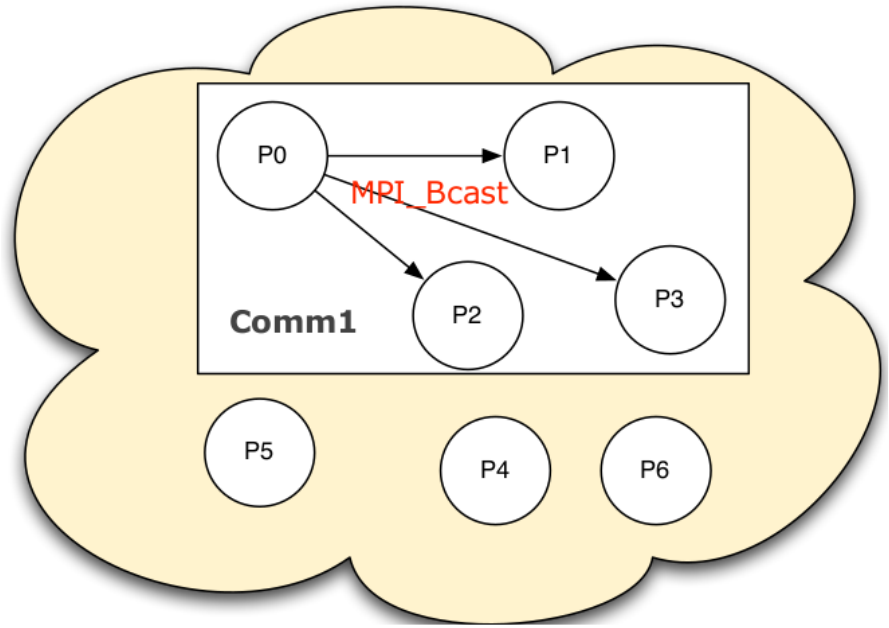
Liste non exhaustive pour le C/C++ :

- Les types de bases :
  MPI_CHAR, MPI_INTEGER, MPI_SHORT, MPI_INT, MPI_LONG
  MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED
  MPI_LONG_LONG, MPI_UNSIGNED_LONG
  MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE
  MPI_INT8_T, MPI_INT16_T, MPI_INT32_T, MPI_INT64_T
  MPI_UINT8_T, MPI_UINT16_T, MPI_UINT32_T, MPI_UINT64_T
- Les types de base spéciaux (packed) :
  MPI_2INT, MPI_LONG_INT, MPI_SHORT_INT
  MPI_FLOAT_INT, MPI_DOUBLE_INT, MPI_LONG_DOUBLE_INT
- Les types construits par l'utilisateur :
  MPI_PACK (voir `MPI_Pack` et `MPI_Unpack`)
  Et tous les autres types créés avec `MPI_Type_commit`.

Il y a aussi l'équivalent pour le Fortran.

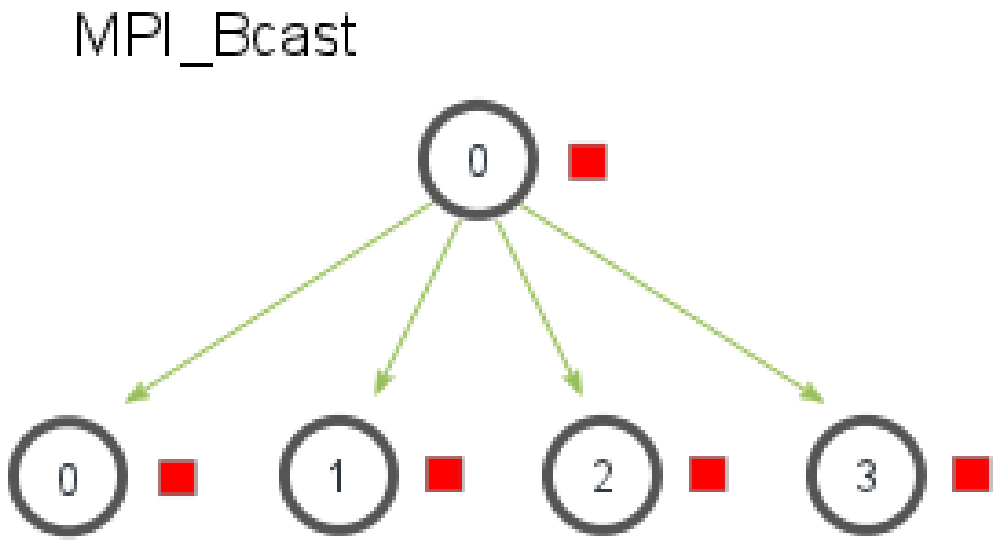
Communications collectives : One-to-All

```
/* Broadcast */
int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype,
               int root, MPI_Comm comm);
int MPI_Ibcast (void *buffer, int count, MPI_Datatype datatype,
                int root, MPI_Comm comm, MPI_Request *request);
```



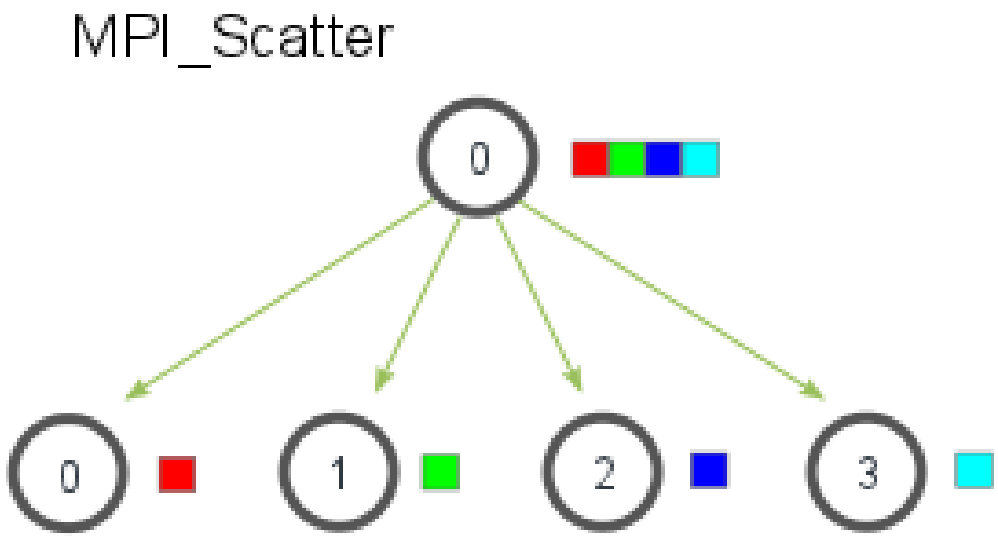
Communications collectives : One-to-All

```
/* Broadcast */
int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype,
                             int root, MPI_Comm comm);
int MPI_Ibcast (void *buffer, int count, MPI_Datatype datatype,
                              int root, MPI_Comm comm, MPI_Request *request);
```



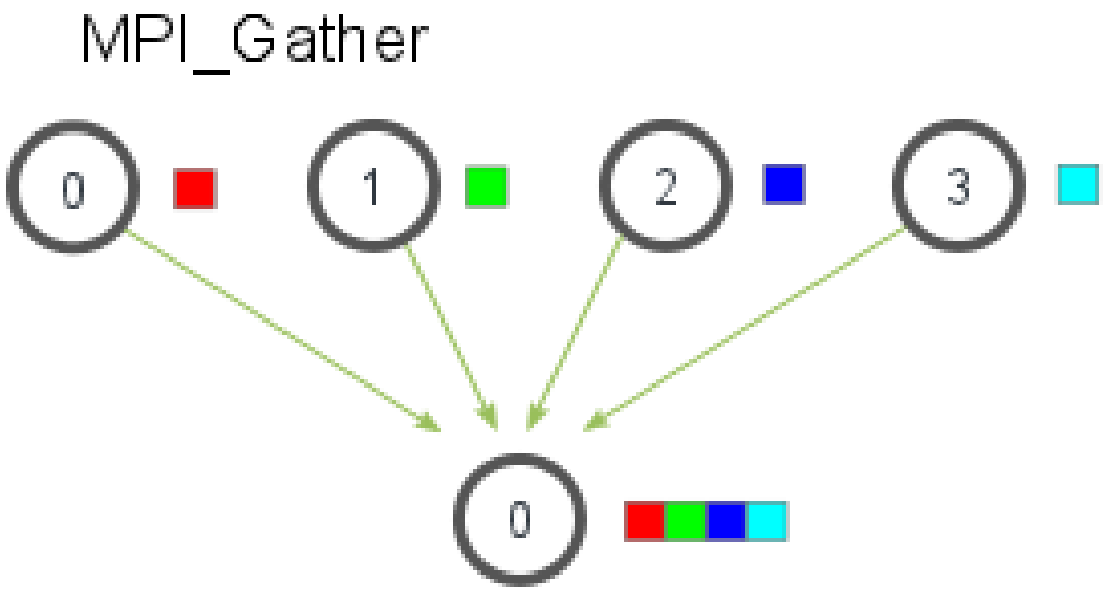
Communications collectives : One-to-All

```
/* Scatter */
int MPI_Scatter (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
                 int root, MPI_Comm comm);
int MPI_Iscatter (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                  void *recvbuf, int recvcount, MPI_Datatype recvtype,
                  int root, MPI_Comm comm, MPI_Request *request);
```



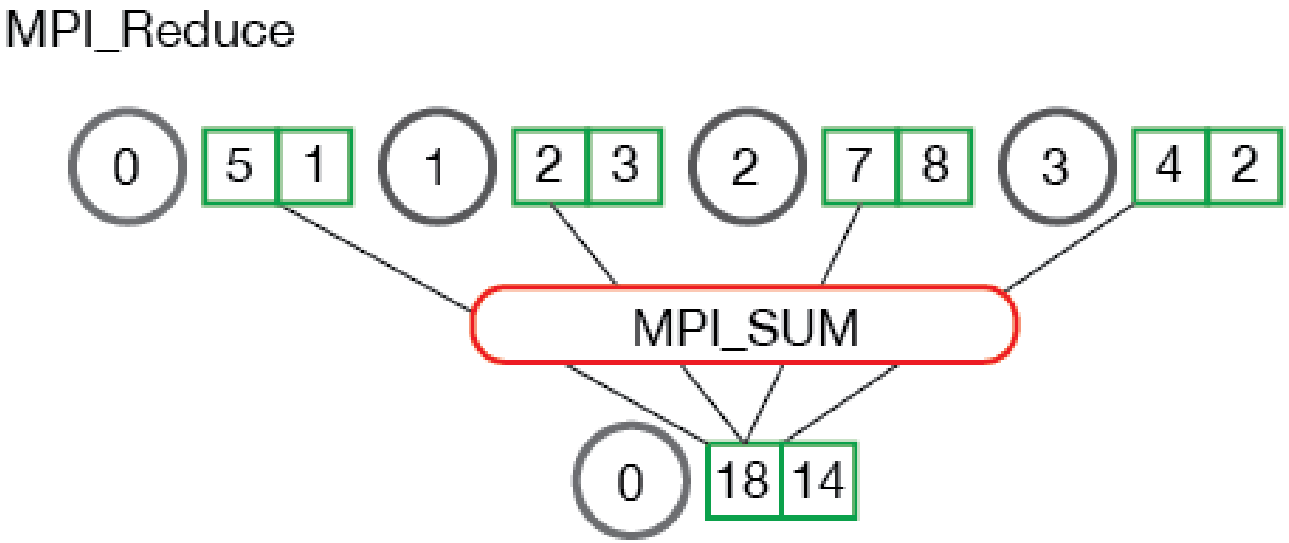
Communications collectives : All-to-one

```
/* Gather */
int MPI_Gather (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                void *recvbuf, int recvcount, MPI_Datatype recvtype,
                int root, MPI_Comm comm);
int MPI_Igather (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                void *recvbuf, int recvcount, MPI_Datatype recvtype,
                int root, MPI_Comm comm, MPI_Request *request);
```



Communications collectives : All-to-one

```
/* Reduction */
int MPI_Reduce (const void *sendbuf, void *recvbuf,
                int count, MPI_Datatype datatype, MPI_Op op,
                int root, MPI_Comm comm);
int MPI_Ireduce (const void *sendbuf, void *recvbuf,
                int count, MPI_Datatype datatype, MPI_Op op,
                int root, MPI_Comm comm, MPI_Request *request);
```



Fonctions disponibles pour les réductions

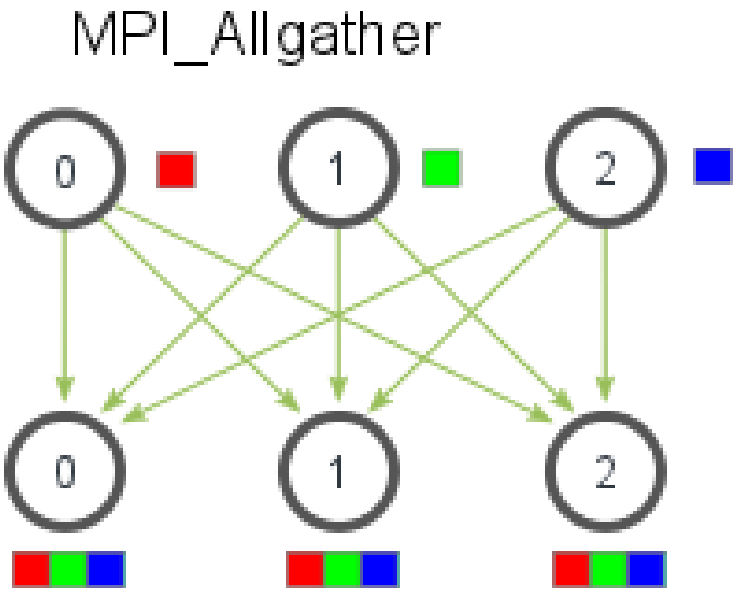
```
/* Reduction */
int MPI_Reduce (const void *sendbuf, void *recvbuf,
                int count, MPI_Datatype datatype, MPI_Op op,
                int root, MPI_Comm comm);
int MPI_Ireduce (const void *sendbuf, void *recvbuf,
                int count, MPI_Datatype datatype, MPI_Op op,
                int root, MPI_Comm comm, MPI_Request *request);
```

Le standard fournit les fonctions suivantes :

- MPI_SUM, MPI_PROD
- MPI_MIN, MPI_MAX
- MPI_MINLOC, MPI_MAXLOC
- MPI_LAND, MPI_LOR, MPI_LXOR
- MPI_BAND, MPI_BOR, MPI_BXOR

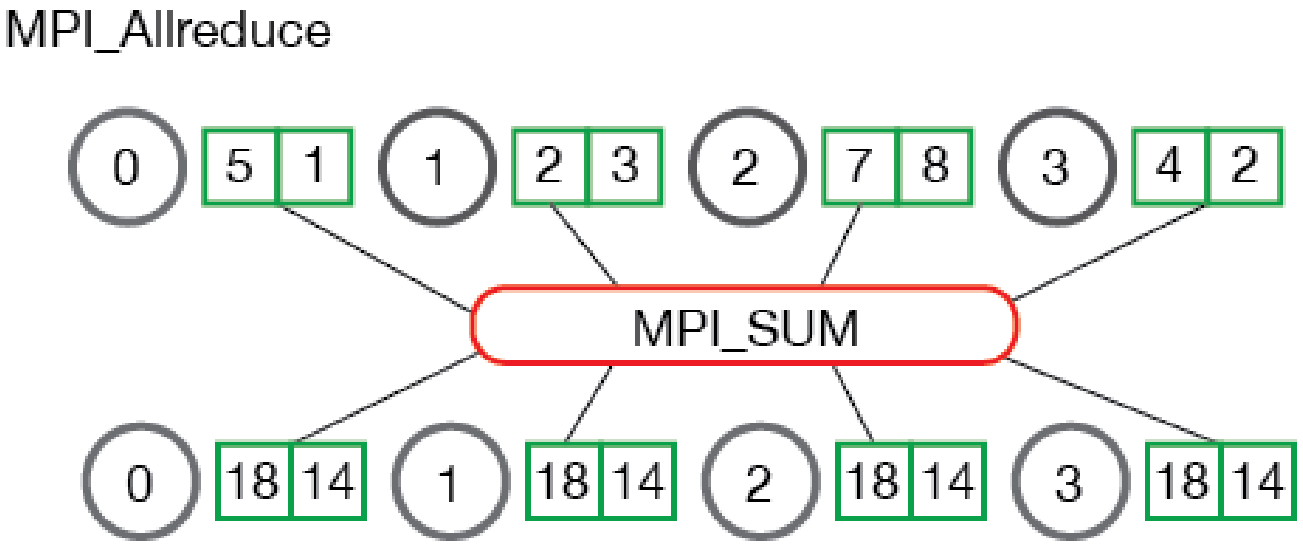
Communications collectives : All-to-all

```
int MPI_Allgather (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                   void *recvbuf, int recvcount,
                   MPI_Datatype recvtype, MPI_Comm comm);
int MPI_Iallgather (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                    void *recvbuf, int recvcount,
                    MPI_Datatype recvtype, MPI_Comm comm,
                    MPI_Request *request);
```



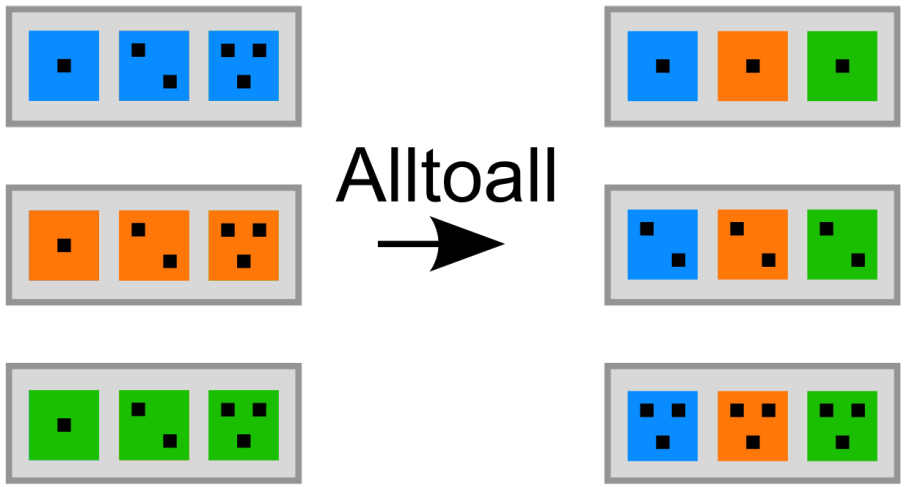
Communications collectives : All-to-all

```
int MPI_Allreduce (const void *sendbuf, void *recvbuf, int count,
                   MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
int MPI_Iallreduce (const void *sendbuf, void *recvbuf, int count,
                   MPI_Datatype datatype, MPI_Op op, MPI_Comm comm,
                   MPI_Request *request);
```



Communications collectives : All-to-all

```
int MPI_Alltoall (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                  void *recvbuf, int recvcount,
                  MPI_Datatype recvtype, MPI_Comm comm);
int MPI_Ialltoall (const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                   void *recvbuf, int recvcount,
                   MPI_Datatype recvtype, MPI_Comm comm,
                   MPI_Request *request);
```

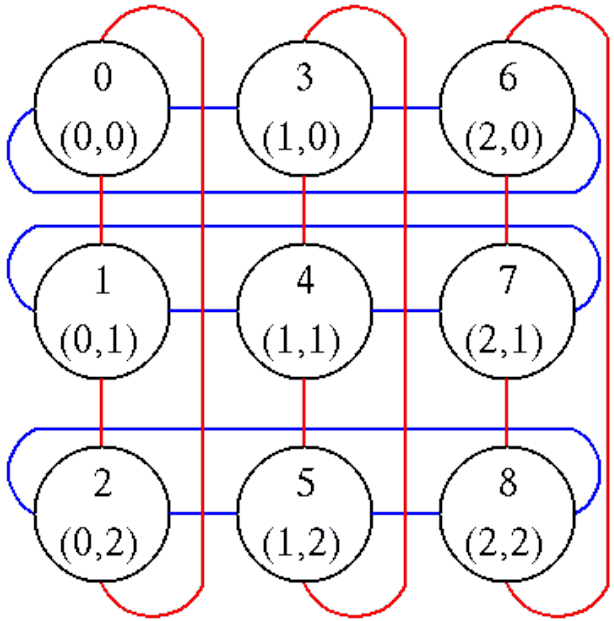


1. Rappel sur les architectures distribuées
   - Nœud et cluster de calcul
   - Technologies réseau
2. Parallélisme par passage de messages
   - Modèle par passage de messages
   - Standard actuel : MPI (1993)
   - MPI en pratique
3. Gestion du contexte de communication et des processus
   - Communicateur par défaut
   - Fonctions de base
   - Créer des sous-communicateurs
4. Passage de messages
   - Les différents types de communications
   - Communications point à point
   - Communications collectives
5. Introduction à la notion de communicateurs de topologie
6. Exemple : Multiplication matrice-vecteur en parallèle

Topologie de communicateurs

Le standard `MPI` prédéfinit deux types de topologies de communicateurs :

- Grille ou tore : Agence les noeuds de calcul sur une grille de dimension n avec possibilité de périodicité sur certains axes.

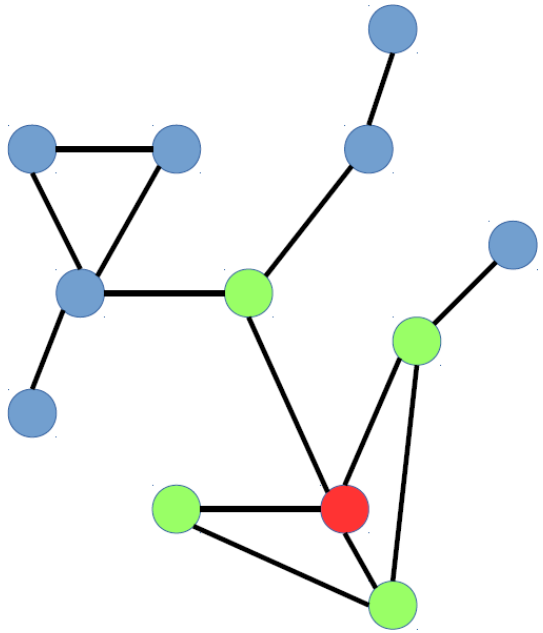


Le placement des noeuds physiques dans la grille peut être optimisé par l'implémentation afin réduire les latences de communications et/ou augmenter la bande passante entre noeuds voisins dans la grille. On peut la créer grâce à la fonction `MPI_Cart_create()`.

Topologie de communicateurs

Le standard MPI prédéfinit deux types de topologies de communicateurs :

- Graphe : Agence les noeuds de calcul selon un graphe quelconque.

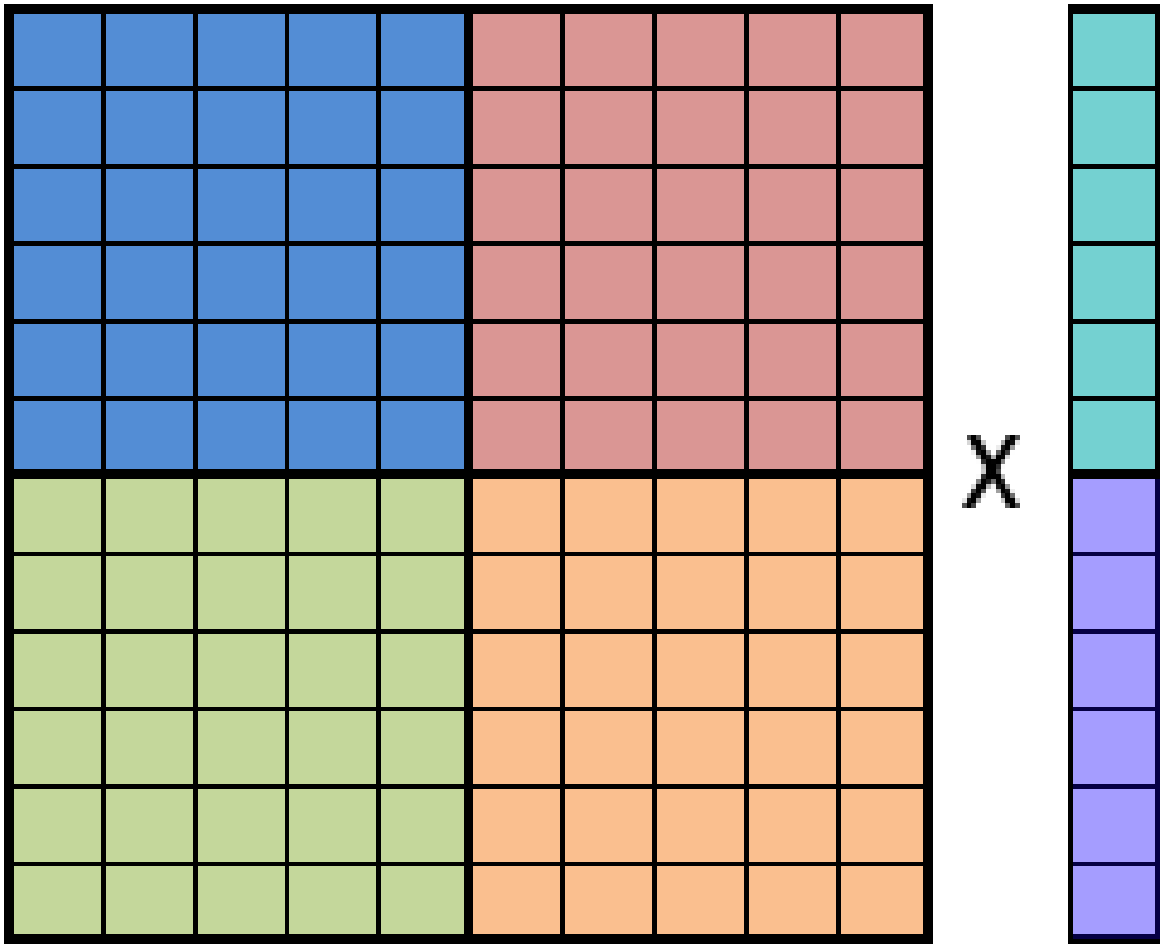


Le placement des noeuds physique peut également être optimisé au niveau des communications avec chaque voisin. On peut créer une topologie de communicateur de graphe grâce à la fonction `MPI_Graph_create()`.

1. Rappel sur les architectures distribuées
   - Nœud et cluster de calcul
   - Technologies réseau
2. Parallélisme par passage de messages
   - Modèle par passage de messages
   - Standard actuel : MPI (1993)
   - MPI en pratique
3. Gestion du contexte de communication et des processus
   - Communicateur par défaut
   - Fonctions de base
   - Créer des sous-communicateurs
4. Passage de messages
   - Les différents types de communications
   - Communications point à point
   - Communications collectives
5. Introduction à la notion de communicateurs de topologie
6. Exemple : Multiplication matrice-vecteur en parallèle

Multiplication matrice-vecteur en parallèle

- Partition des données entre noeuds :



Multiplication matrice-vecteur en parallèle

- Modèle maître-esclave : Le noeud 0 dirige tous les autres.