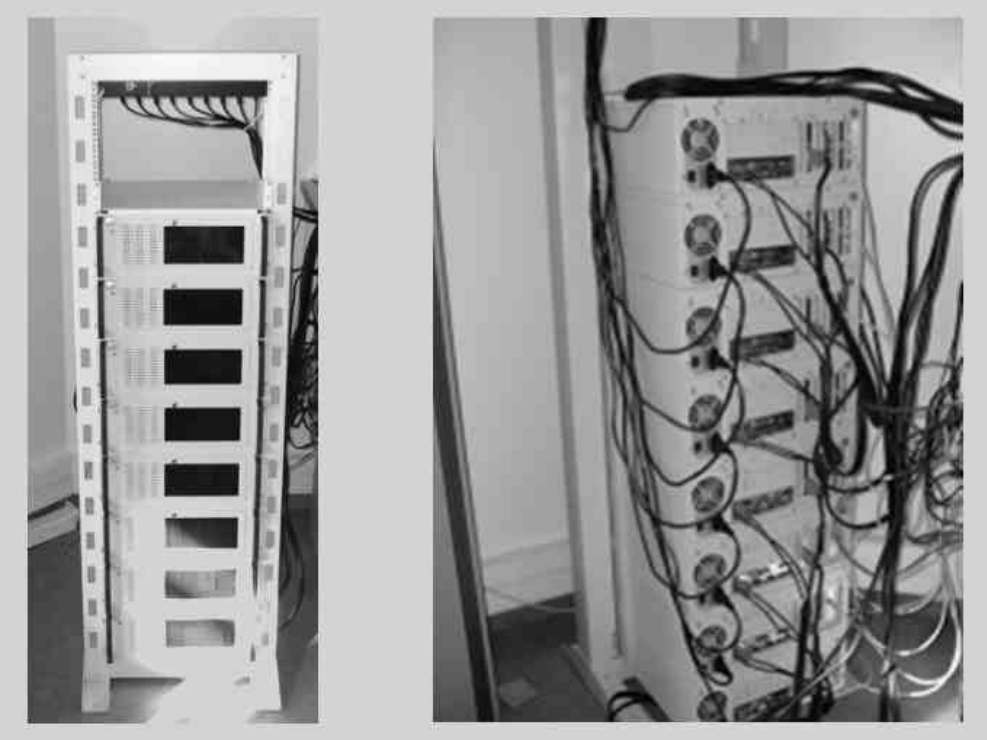
Optimisations de la bibliothèque de communication MPI pour machines parallèles de type « grappe de PCs » sur une primitive d'écriture distante



Olivier Glück

UPMC/LIP6/ASIM

Olivier.Gluck@lip6.fr

Plan

1. Introduction : contexte et objectifs
2. MPI-MPC1 : MPI sur une primitive d'écriture distante
3. MPI-MPC2 : optimisations bas niveau
4. MPI-MPC3 : traductions d'adresses optimisées
5. Résultats expérimentaux
6. Conclusion

Contexte et hypothèses

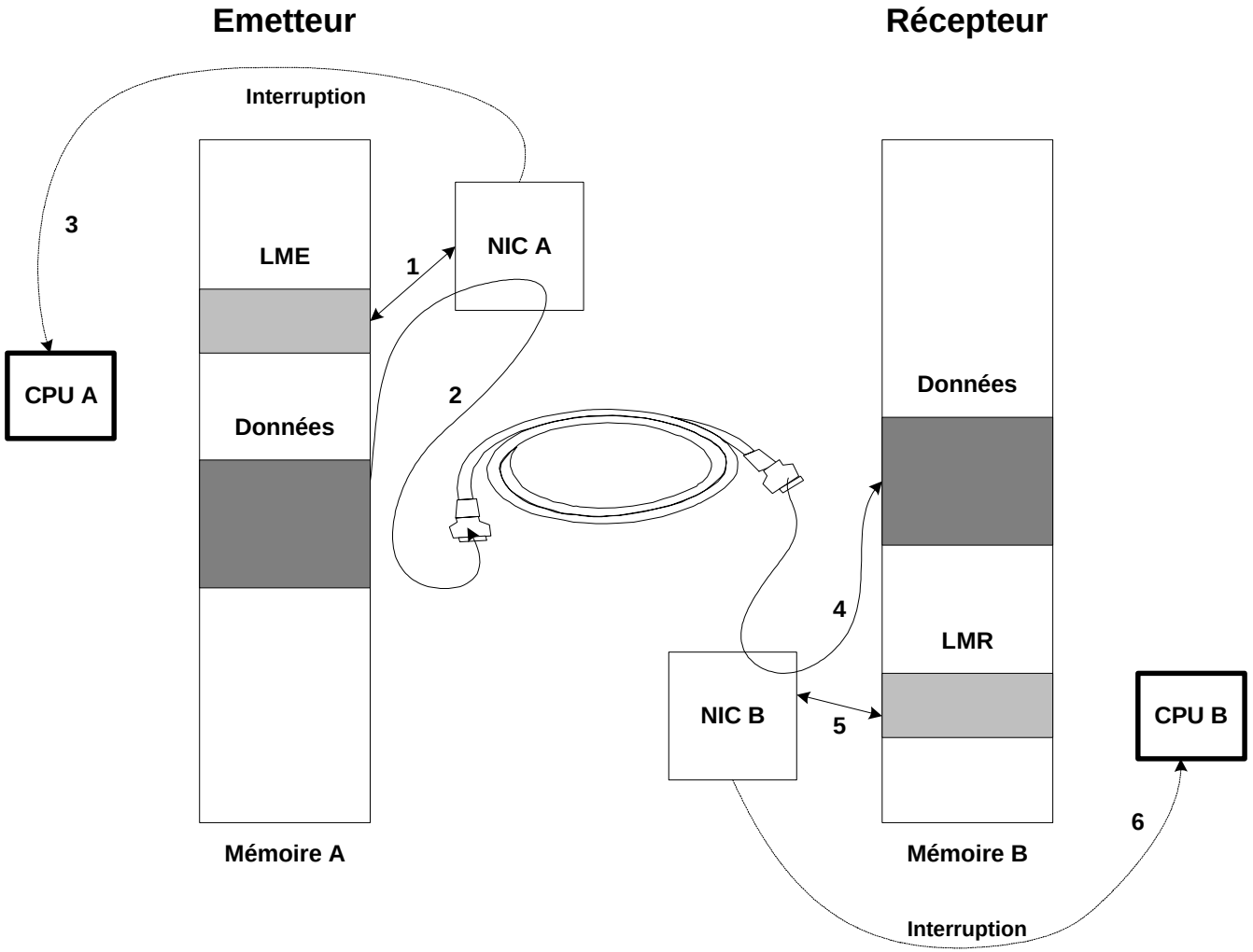
Machines parallèles de type grappe de PCs sous UNIX

Applications MPI en mode « batch »

Réseau de communication : primitive d'écriture distante

- **le réseau est fiable**
- **le contrôleur réseau utilise des accès DMA en lecture et en écriture pour accéder à la mémoire du nœud hôte**
- **le contrôleur réseau ne peut transférer que des zones contiguës en mémoire physique**
- **le nœud émetteur doit savoir à l'avance où déposer les données sur le nœud récepteur**

La primitive d'écriture distante sur MPC



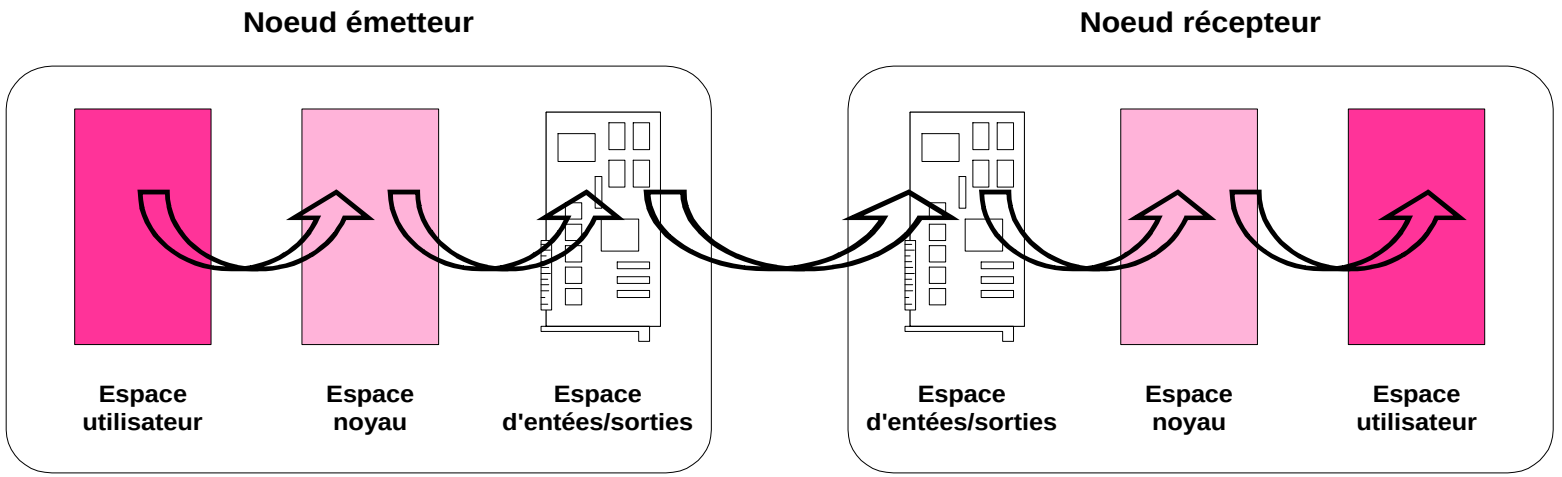
Objectifs

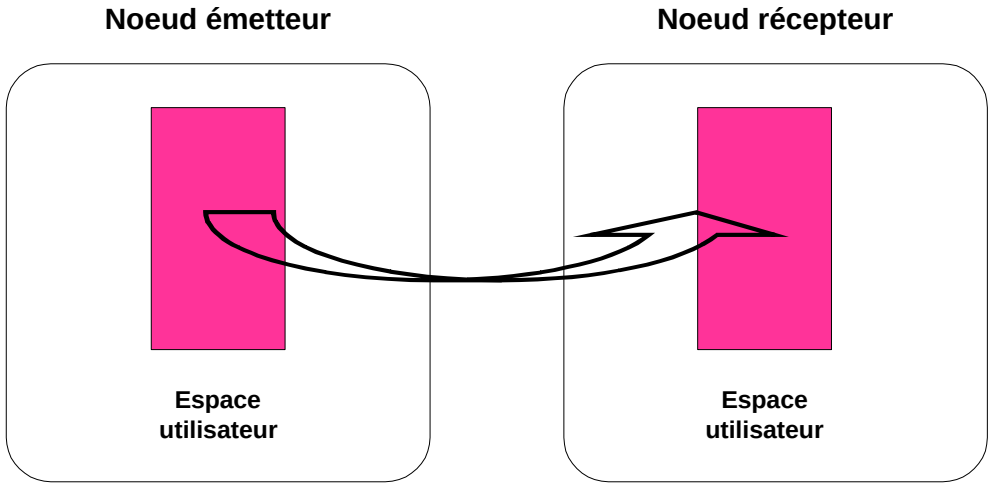
- **Fournir l'environnement MPI :**
  - **machines parallèles de type « grappe de PCs »**
  - **primitive d'écriture distante**
- **Réduire le chemin critique logiciel**

Les techniques d'optimisation

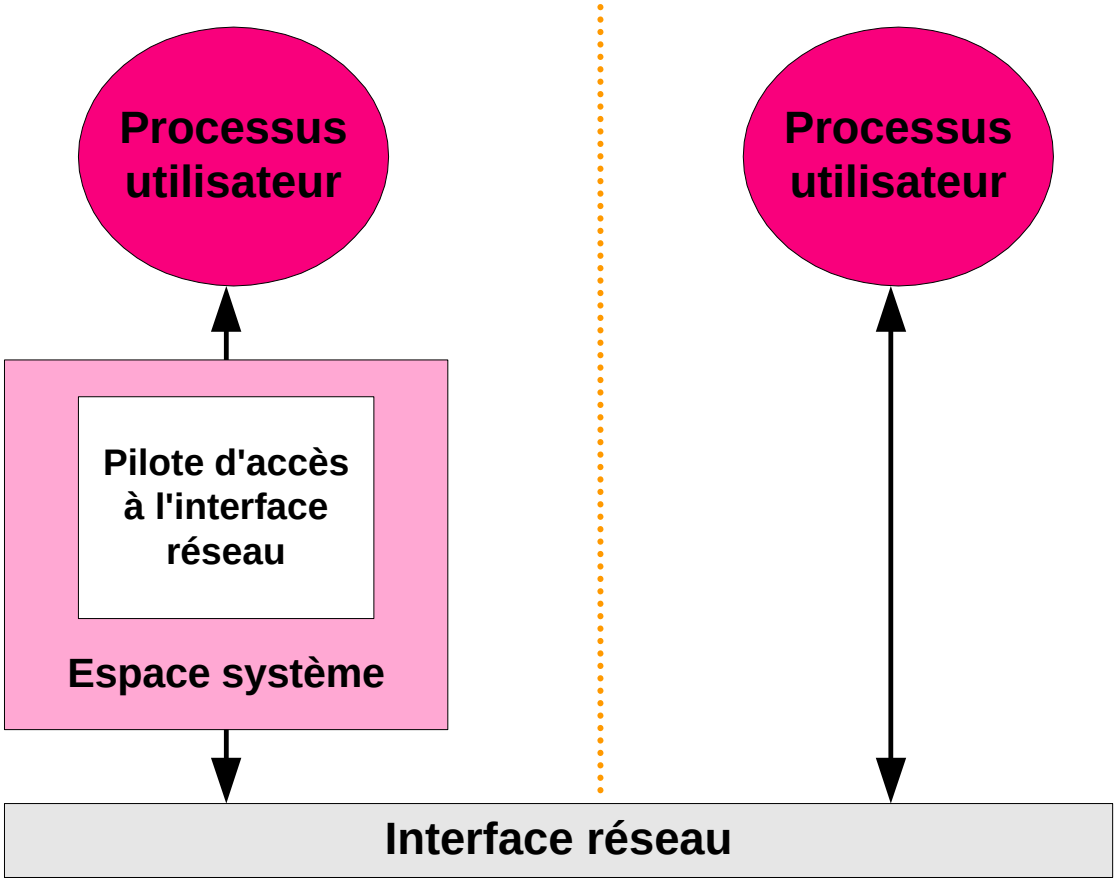
- Eviter la traversée d'un grand nombre de couches de communication
- Réduire le nombre de copies des données
- Réduire le nombre d'appel système en réalisant les communications en espace utilisateur
- Eviter l'utilisation d'interruptions pour la signalisation des communications
- Réduire le coût des opérations de traduction d'adresses virtuelles/physiques

Une stratégie « zéro-copie »

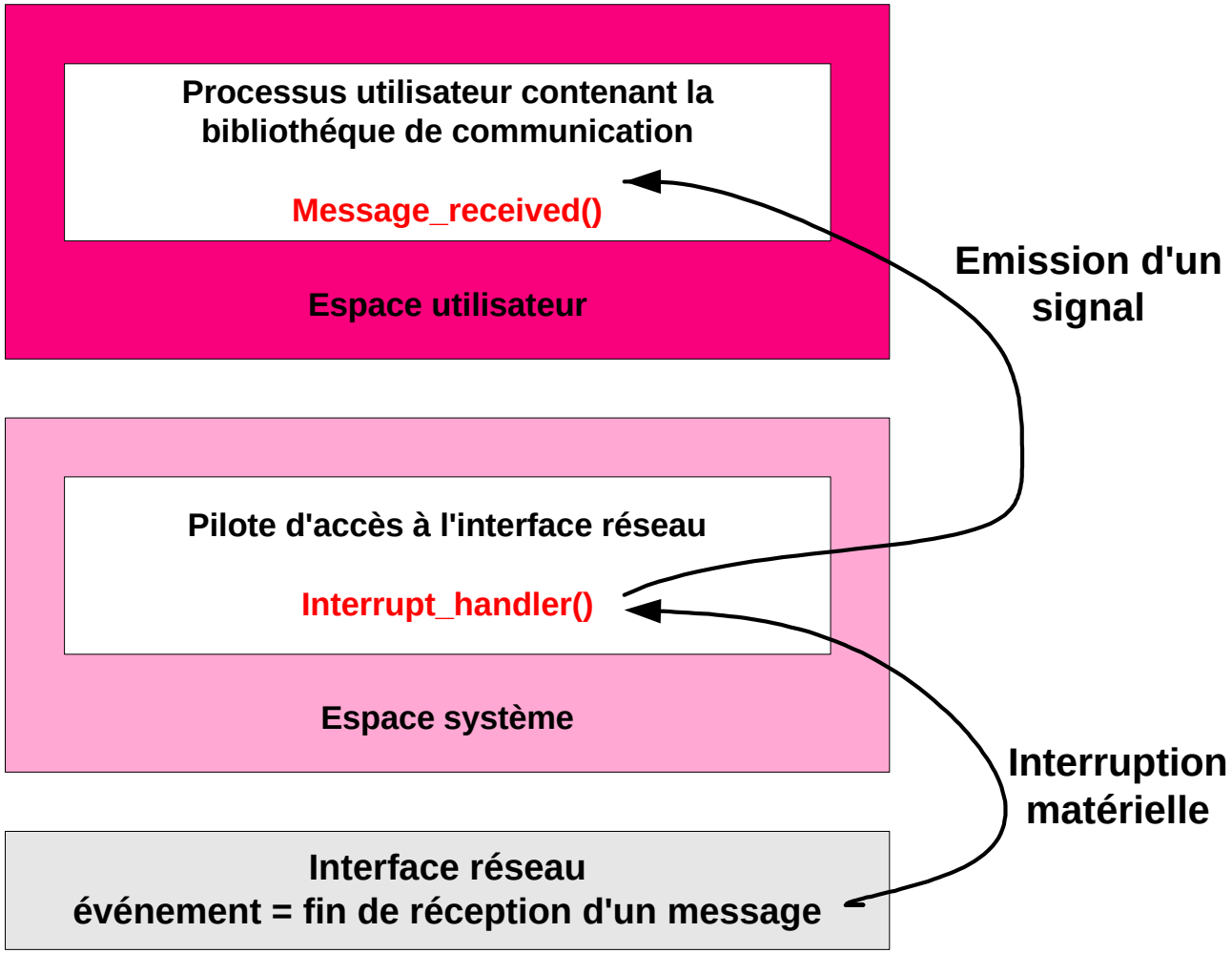




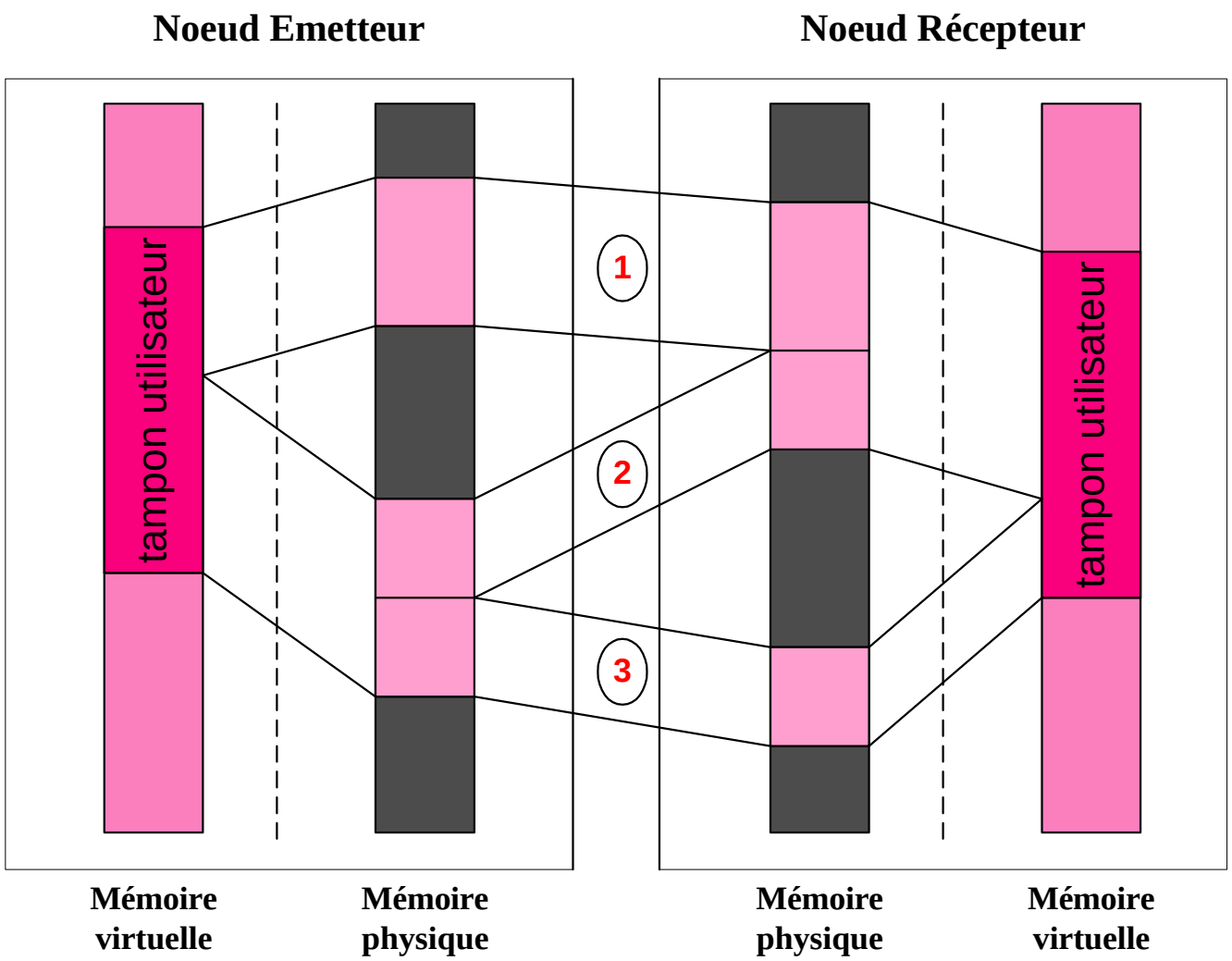
Les appels système



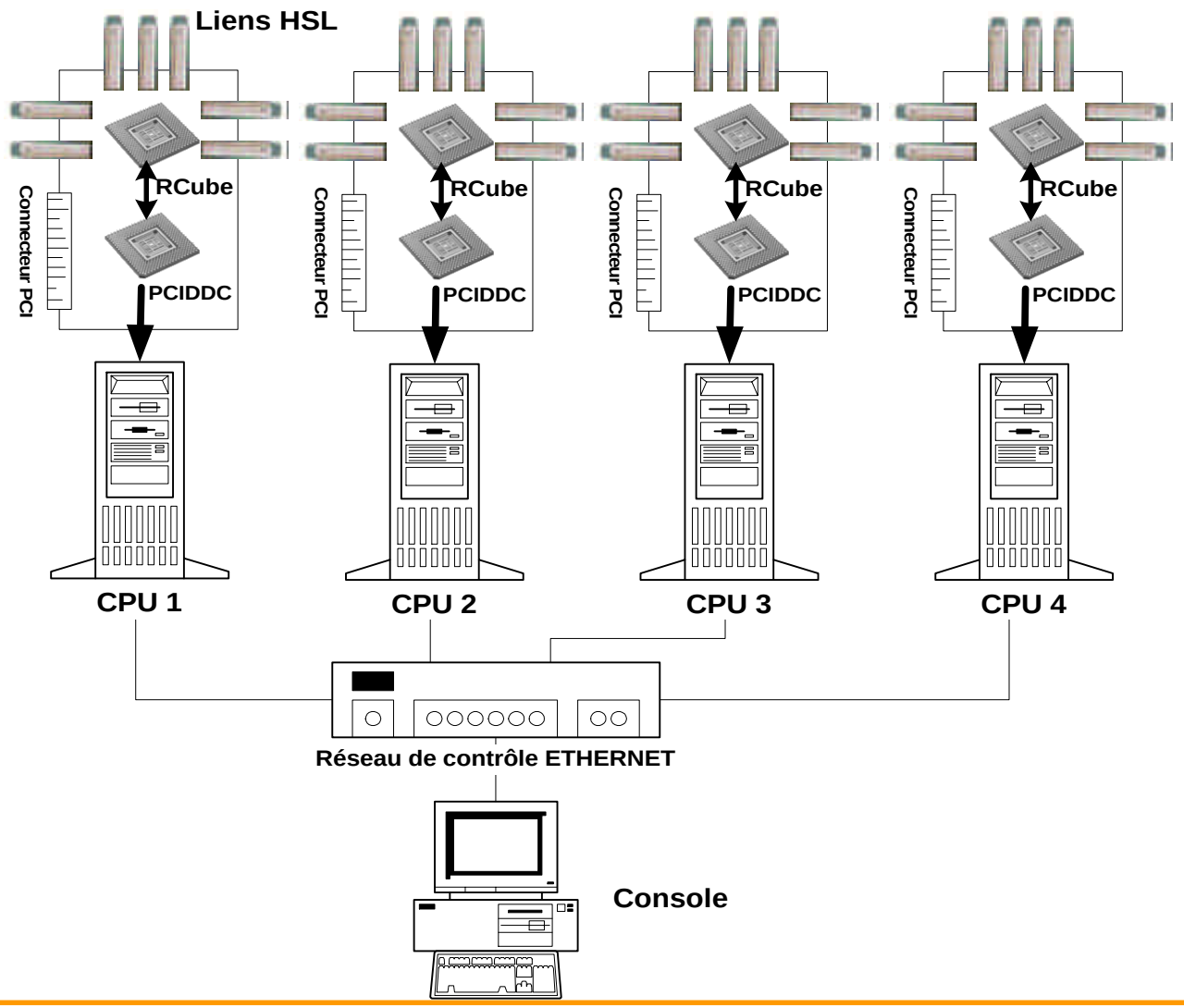
Les interruptions



Les traductions d'adresses

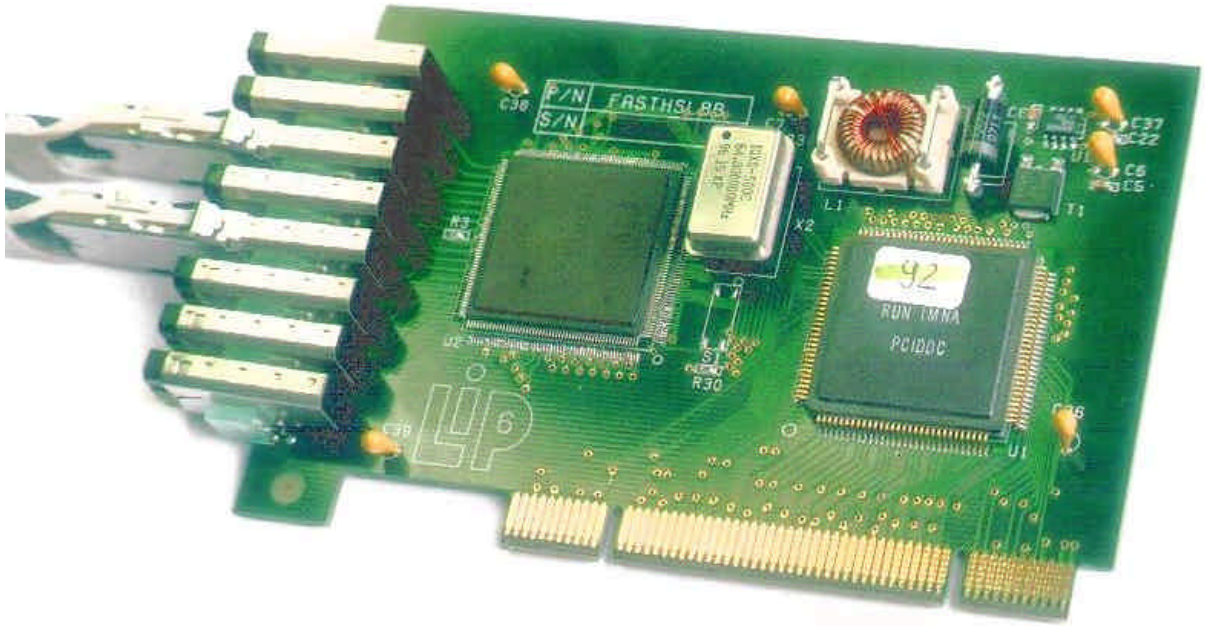


La machine MPC



La machine MPC

La carte FastHSL



Performances matérielles :

- **latence : 2 µs**
- **débit maximum sur le lien : 1 Gbits/s (dans chaque sens)**
- **débit utile matériel théorique : 512 Mbits/s**

Plan

1. Introduction : contexte et objectifs

2. MPI-MPC1 : MPI sur une primitive d'écriture distante

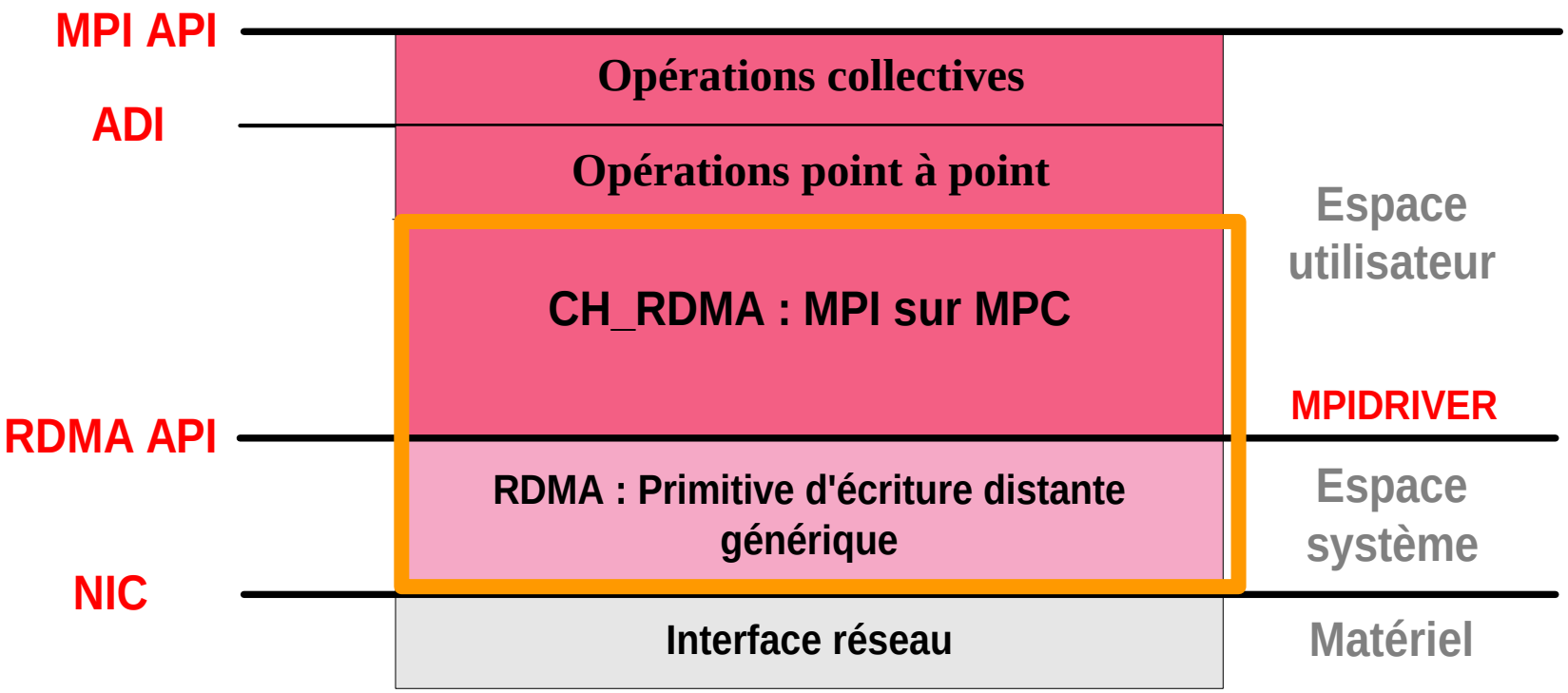
3. MPI-MPC2 : optimisations bas niveau

4. MPI-MPC3 : traductions d'adresses optimisées

5. Résultats expérimentaux

6. Conclusion

MPICH : une architecture en couches



L'API RDMA

RDMA : *Remote Direct Memory Access*

API générique d'écriture en mémoire distante

Objectifs :

- **définir la brique de base de notre implémentation de MPI**
- **masquer les particularités des réseaux fournissant une primitive d'écriture distante**

Enjeu :

- **faire en sorte que notre travail puisse être exploitable sur d'autres plate-formes matérielles**

L'API RDMA

L'écriture distante :

- **RDMA_SEND(nsrc, ndst, plad, prad, len, ctrl, sid, rid, ns, nr)**

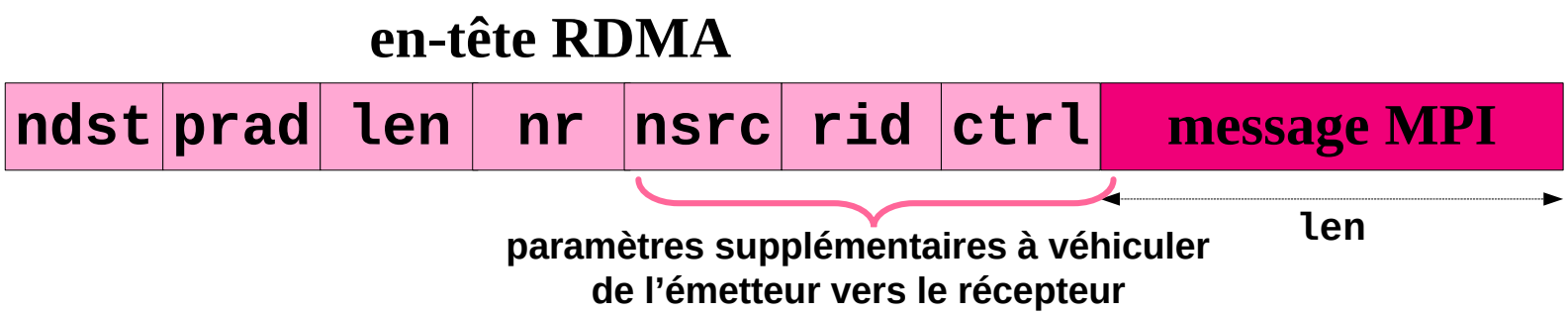
La notification :

- **RDMA_SENT_NOTIFY(ctrl, sid)**
- **RDMA_RECV_NOTIFY(nsrc, ctrl, rid)**

La signalisation par scrutation (optionnelle) :

- **RDMA_NET_LOOKUP(blocking)**

Format des messages :



Les problèmes à résoudre

Les problèmes liés à la primitive d'écriture distante :

- **le dépôt direct en mémoire : l'émetteur doit connaître les adresses physiques des tampons distants**
- **l'utilisation d'adresses physiques alors que l'application manipule des adresses virtuelles**

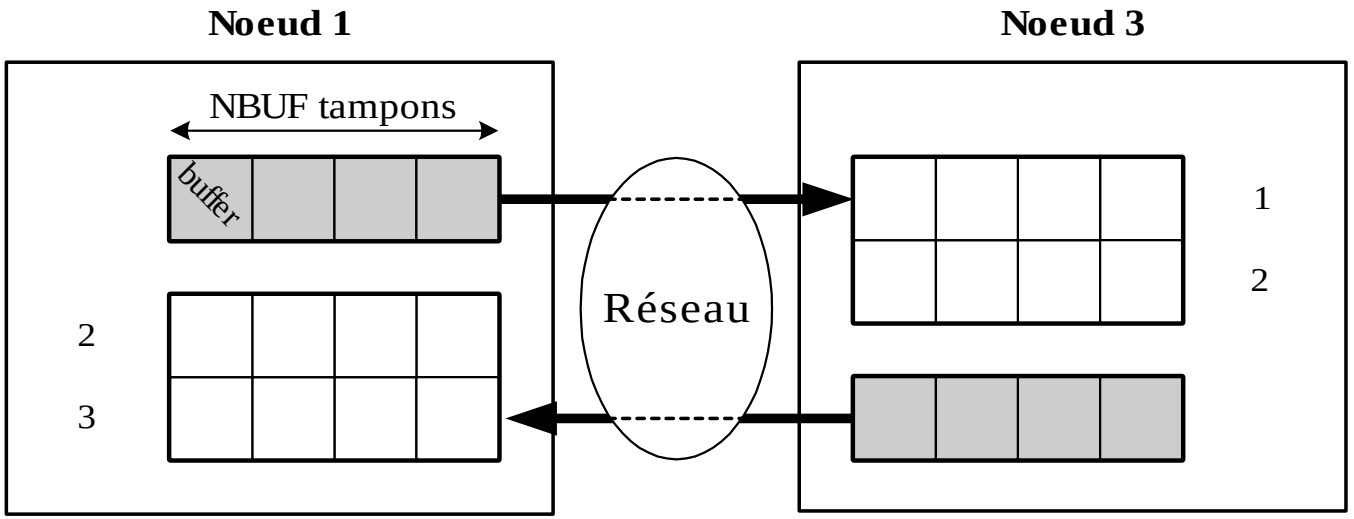
Les services à fournir à MPICH :

- **transmission des messages CTRL/DATA**
- **signalisation des événements réseau**
- **mécanisme de contrôle de flux**
- **fixer le seuil optimal entre les messages CTRL/DATA**

Les messages CTRL

- **Transfert d'informations de contrôle ou de données de taille limitée**
- **Utilisation de tampons intermédiaires contigus en mémoire physique et alloués au démarrage de l'application**

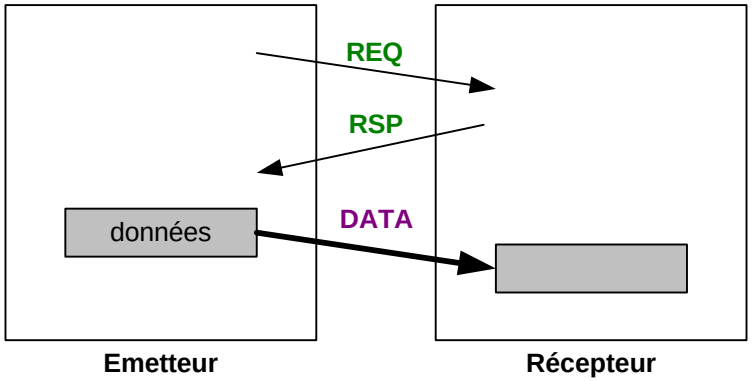
⇨ **pas de traduction d'adresses, adresse du tampon distant connue sur l'émetteur, recopie systématique, contrôle de flux**



Les messages DATA

- Transfert des données de l'application en mode « zéro-copie »
- Utilisation d'un protocole de rendez-vous entre l'émetteur et le récepteur

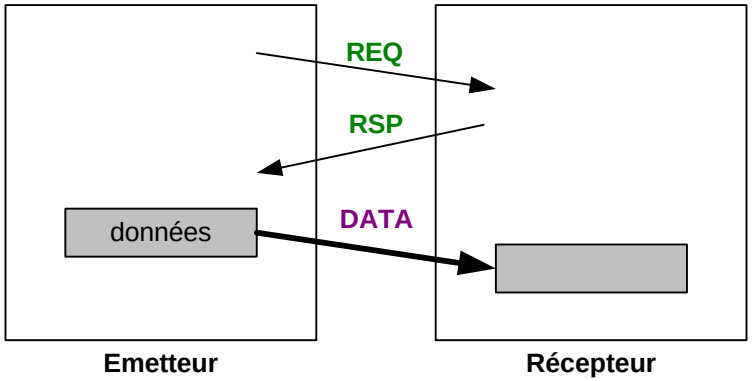
⇨ traduction d'adresses sur l'émetteur et le récepteur, le message RSP contient la description en mémoire physique du tampon de réception

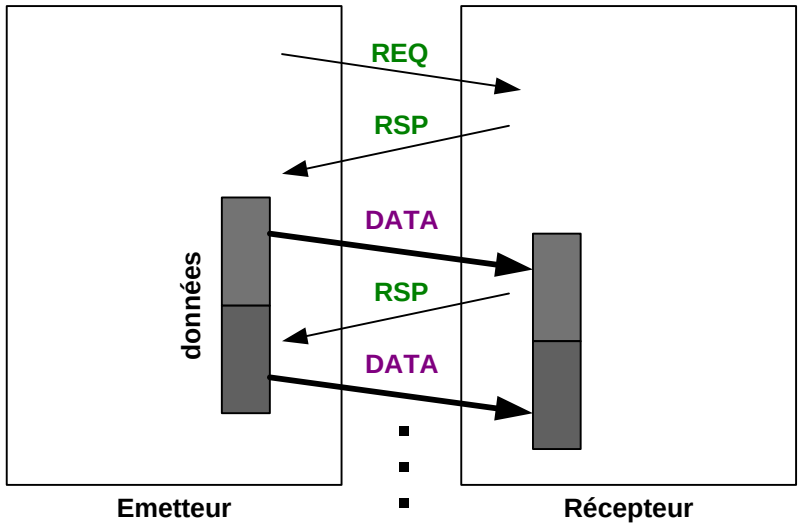


Les messages DATA

- Transfert des données de l'application en mode « zéro-copie »
- Utilisation d'un protocole de rendez-vous entre l'émetteur et le récepteur

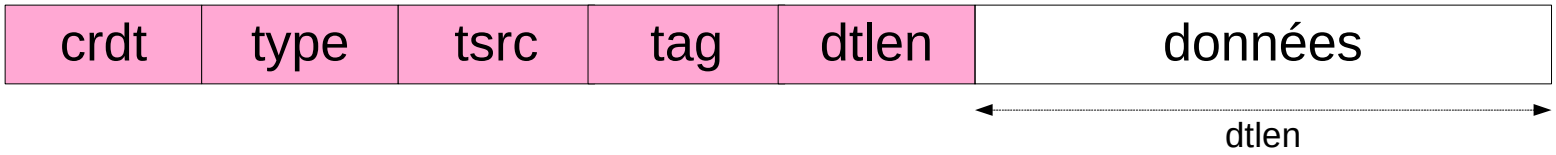
⇨ traduction d'adresses sur l'émetteur et le récepteur, le message RSP contient la description en mémoire physique du tampon de réception





Le format des messages CTRL

SHORT : 20 octets + dtlen

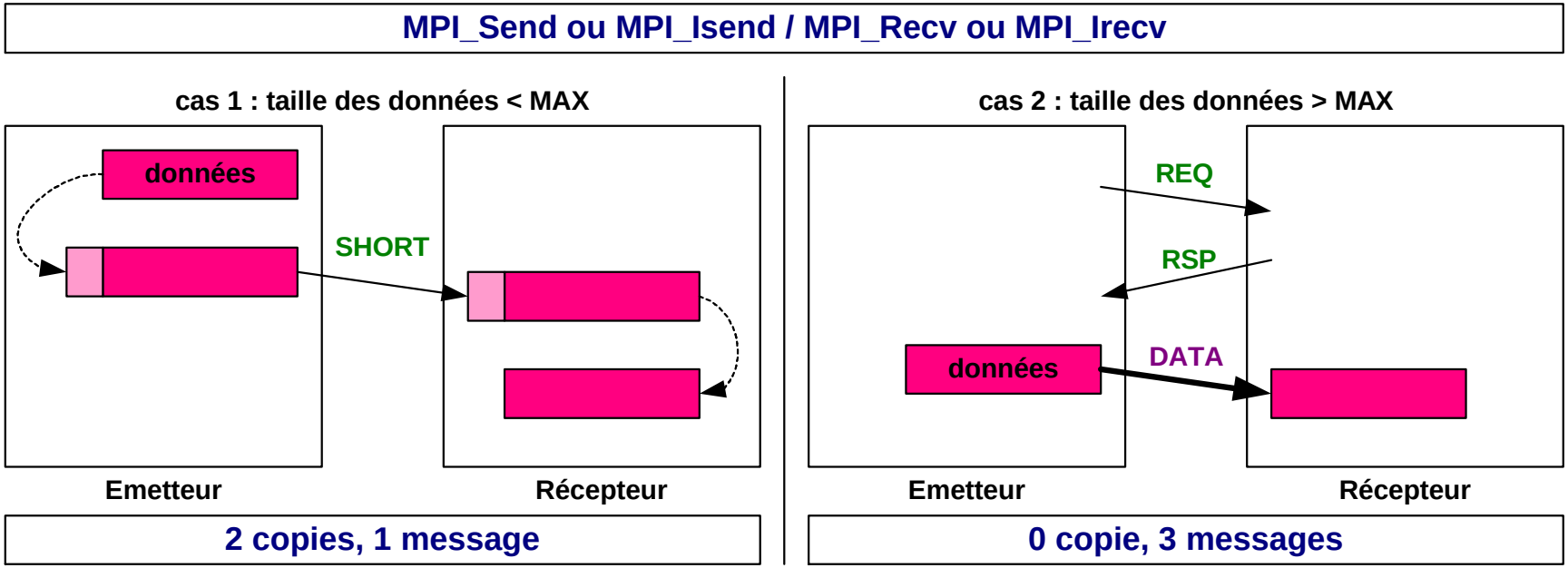


REQ : 24 octets

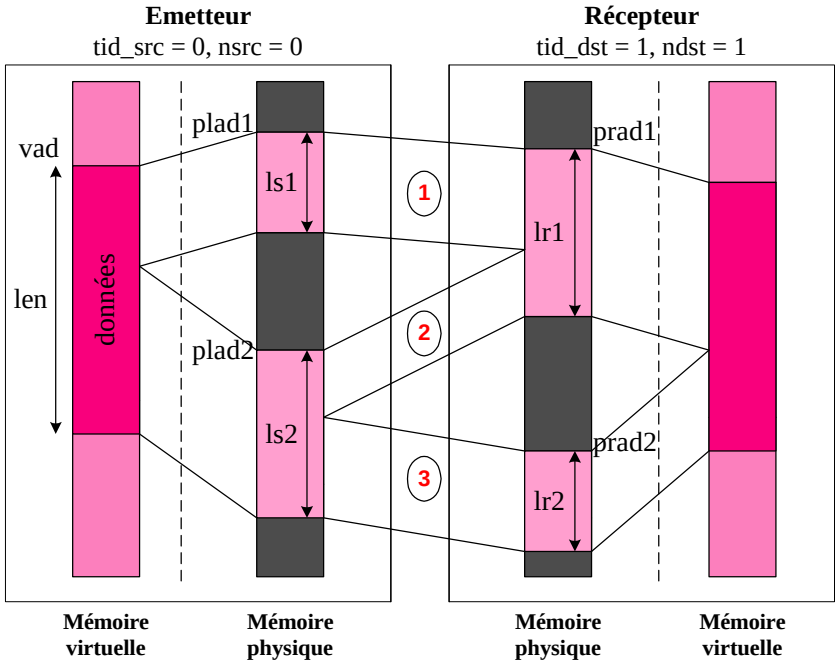
RSP : 20 octets + 8*NB_DMA

CREDIT : 12 octets

Le mode standard dans MPI



Liens entre MPI et l'API RDMA



RDMA_SEND() **REQ**

(prad1, lr1) (prad2, lr2)

RDMA_SEND() **RSP**

(plad1, ls1) (plad2, ls2)

1 RDMA_SEND()

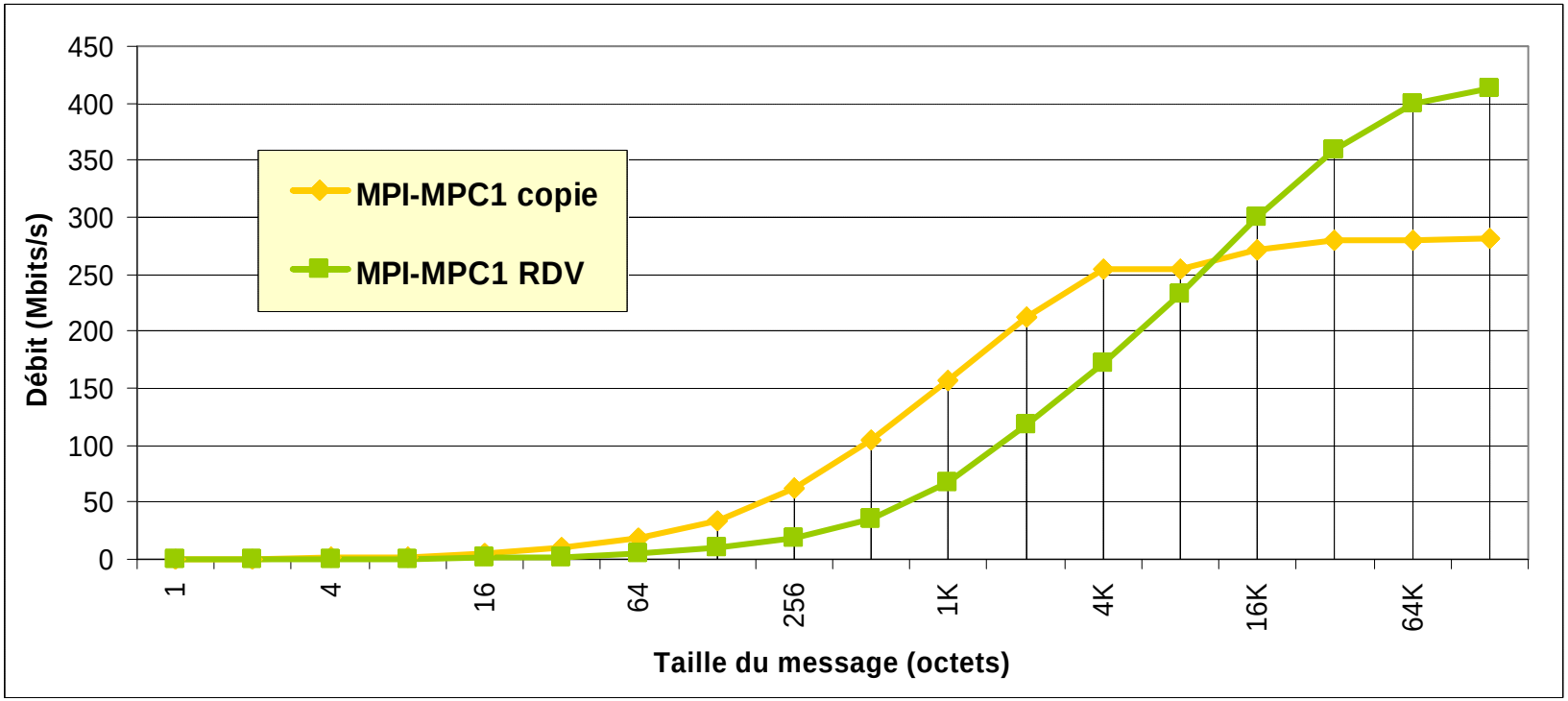
2 RDMA_SEND()

3 RDMA_SEND()

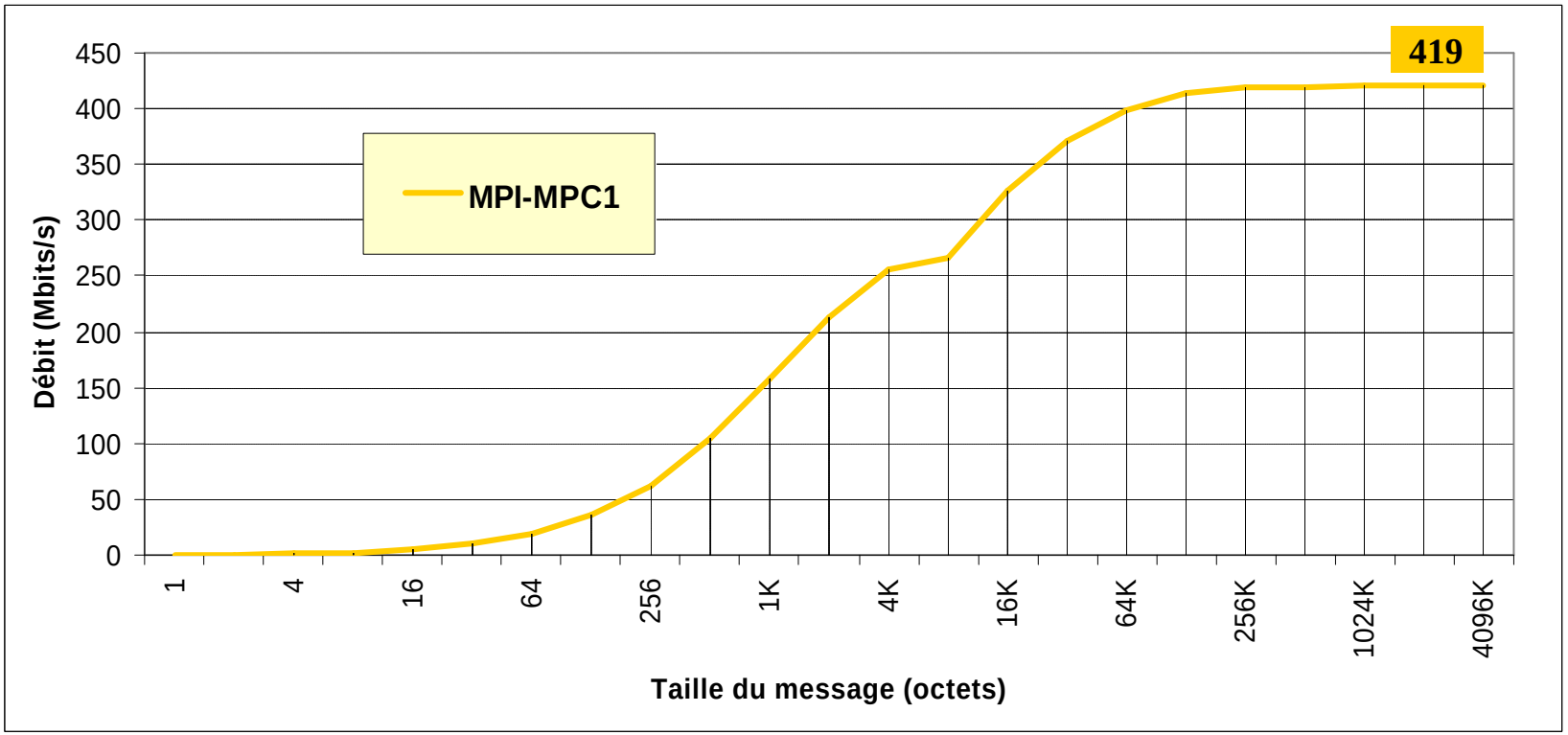
DATA

Performances de MPI-MPC1

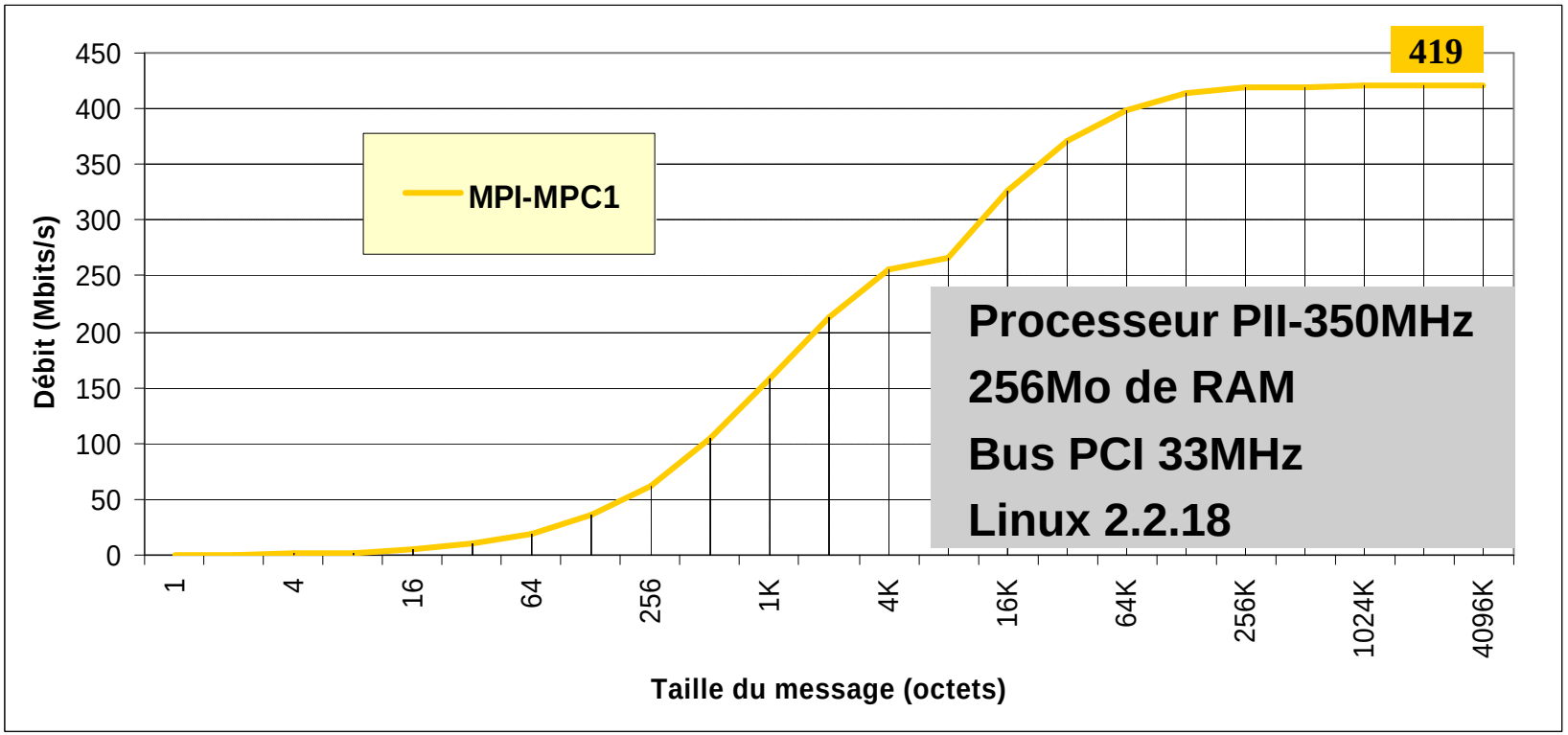
Performances de MPI-MPC1



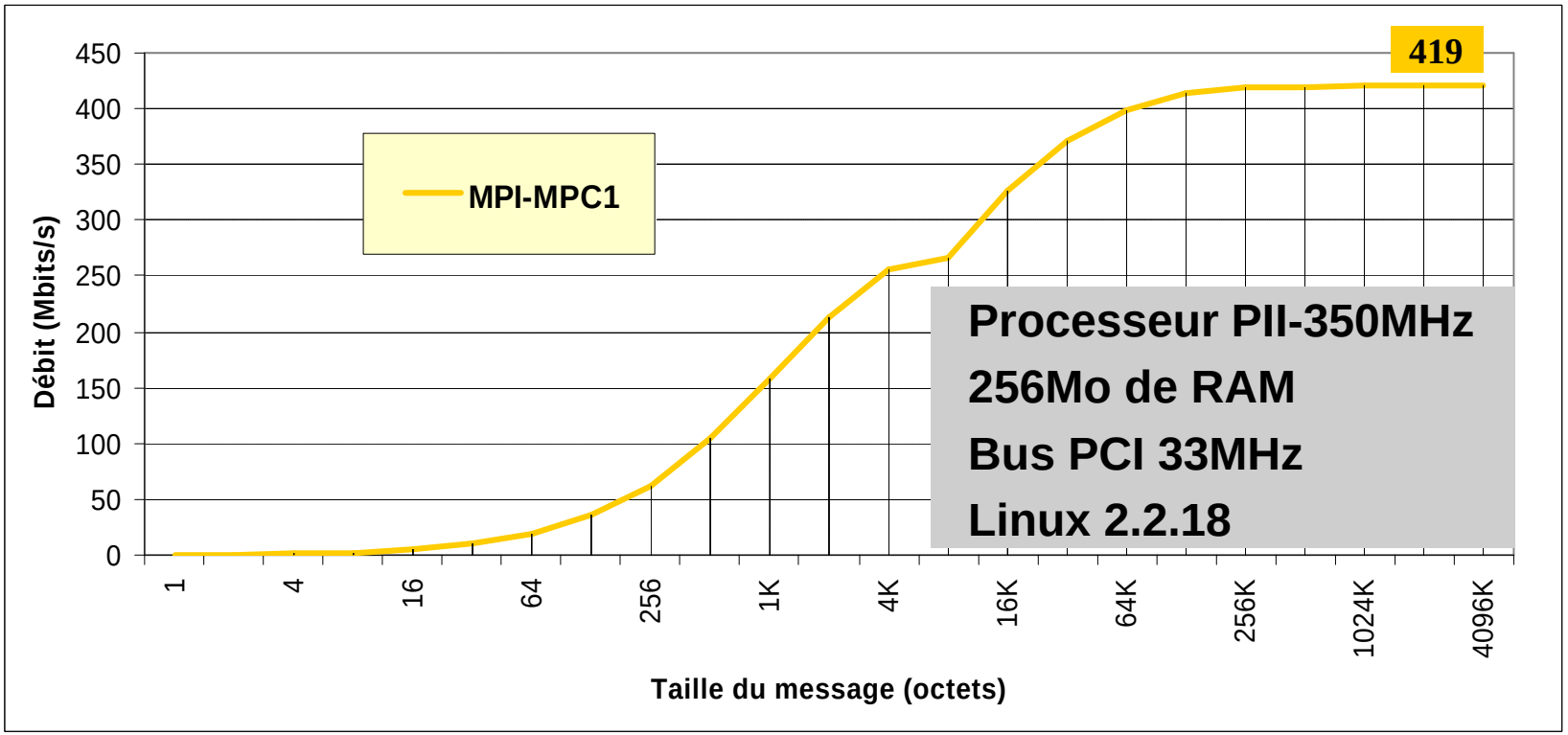
Performances de MPI-MPC1



Performances de MPI-MPC1

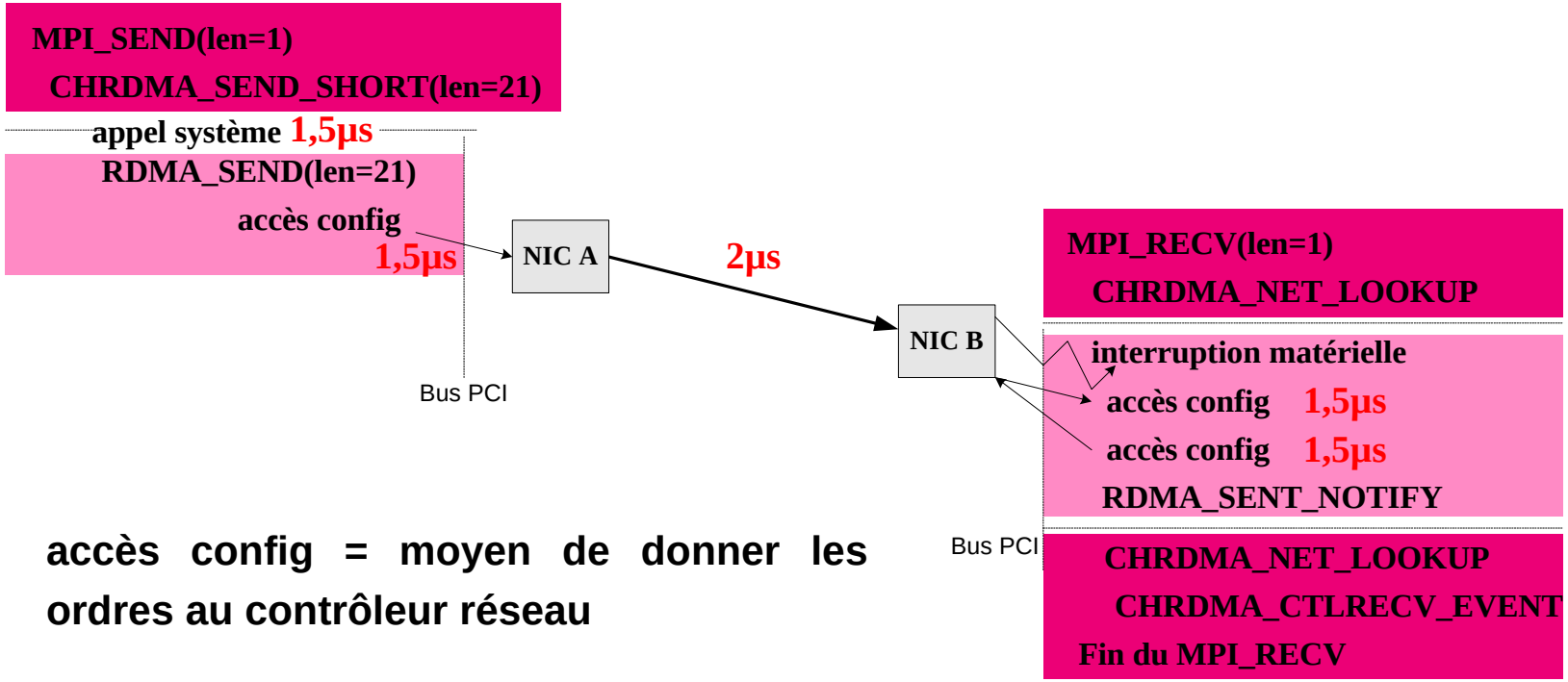


Performances de MPI-MPC1



	Latence (µs)	Débit Max. (Mb/s)	Seuil (Ko)	Demi débit (Ko)	Latence RDV (µs)
MPI-MPC1	26	419	16	2	105

Analyse des performances de MPI-MPC1



accès config = moyen de donner les ordres au contrôleur réseau

	Appels systèmes		Interruptions		Recopies		RDMA_SEND	
Protocole	short	rdv	short	rdv	short	rdv	short	rdv
MPI-MPC1	**1**	**1+6F**	**2**	**2+4F**	**2**	**0**	**1**	**1+F+NB_DMA*F**

Plan

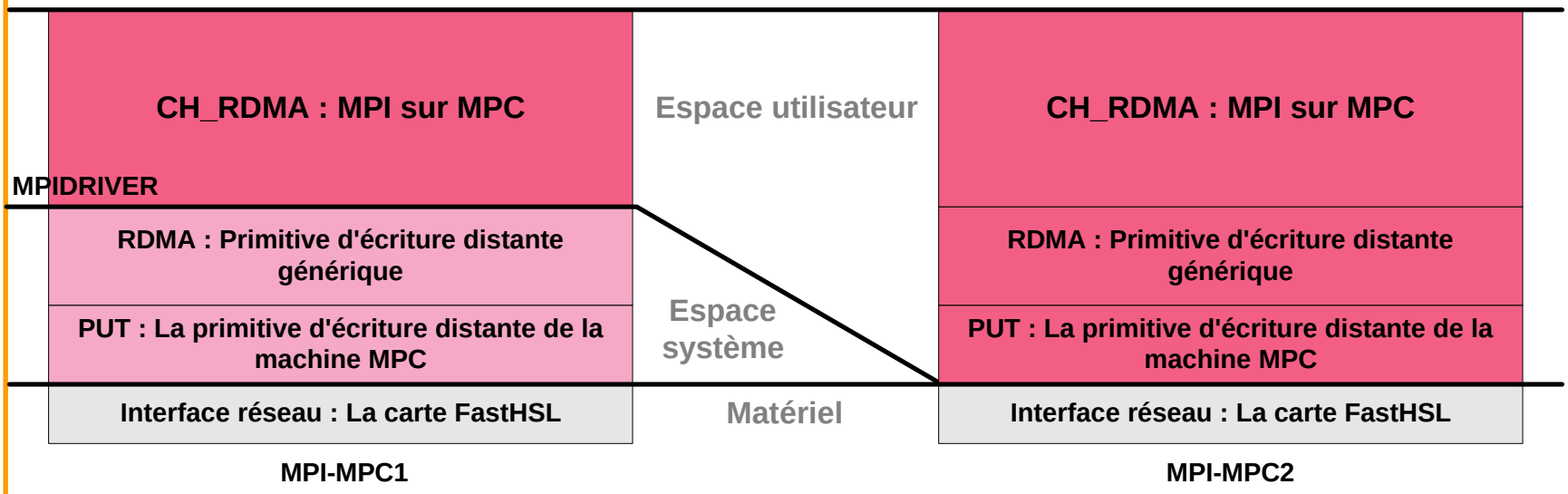
1. Introduction : contexte et objectifs
2. MPI-MPC1 : MPI sur une primitive d'écriture distante
3. MPI-MPC2 : optimisations bas niveau
4. MPI-MPC3 : traductions d'adresses optimisées
5. Résultats expérimentaux
6. Conclusion

Objectifs de MPI-MPC2

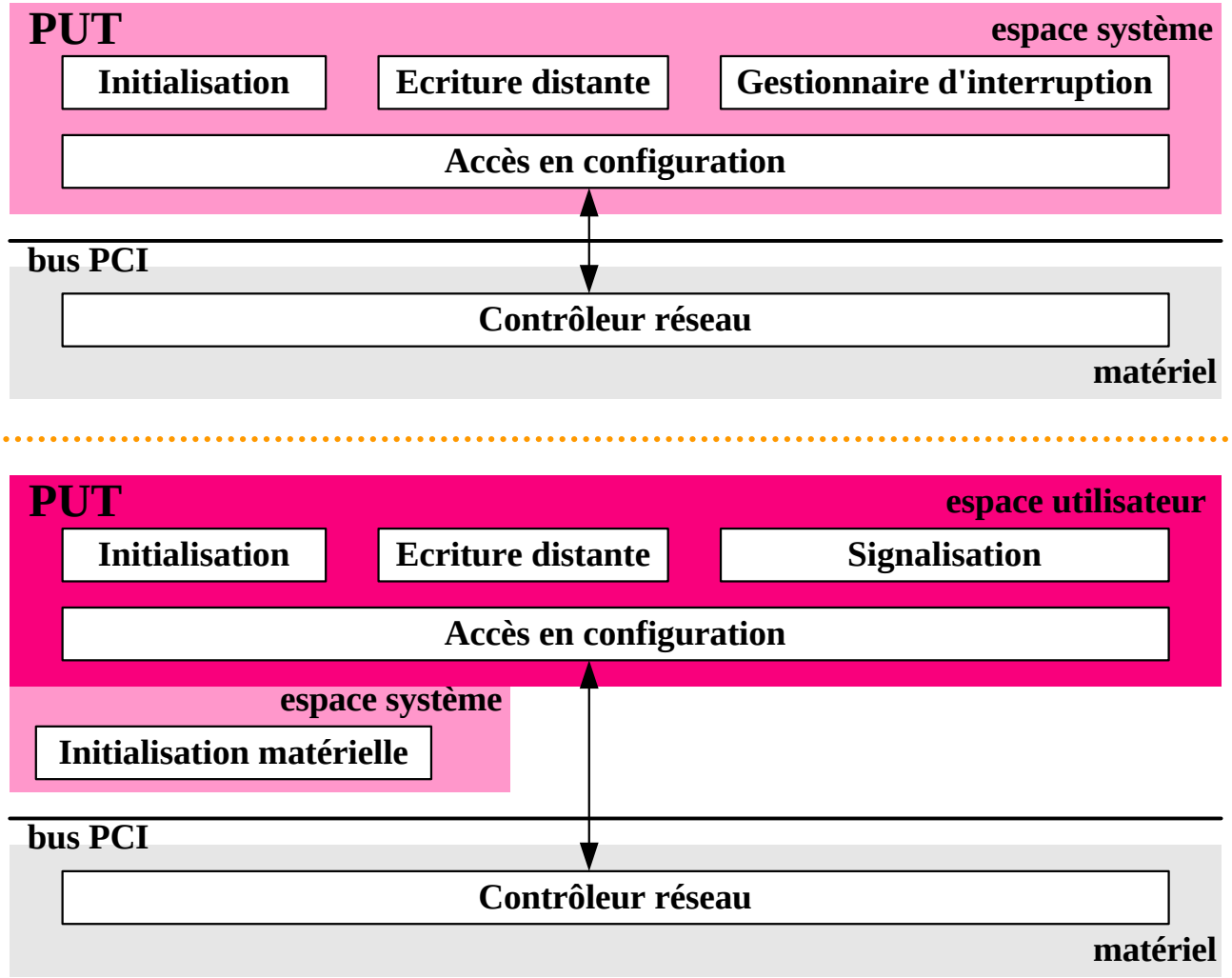
Réaliser les communications en mode utilisateur en partageant les ressources réseau.

Remplacer les interruptions matérielles par une signalisation par scrutation.

Etudier l'impact de ces changements sur les performances.



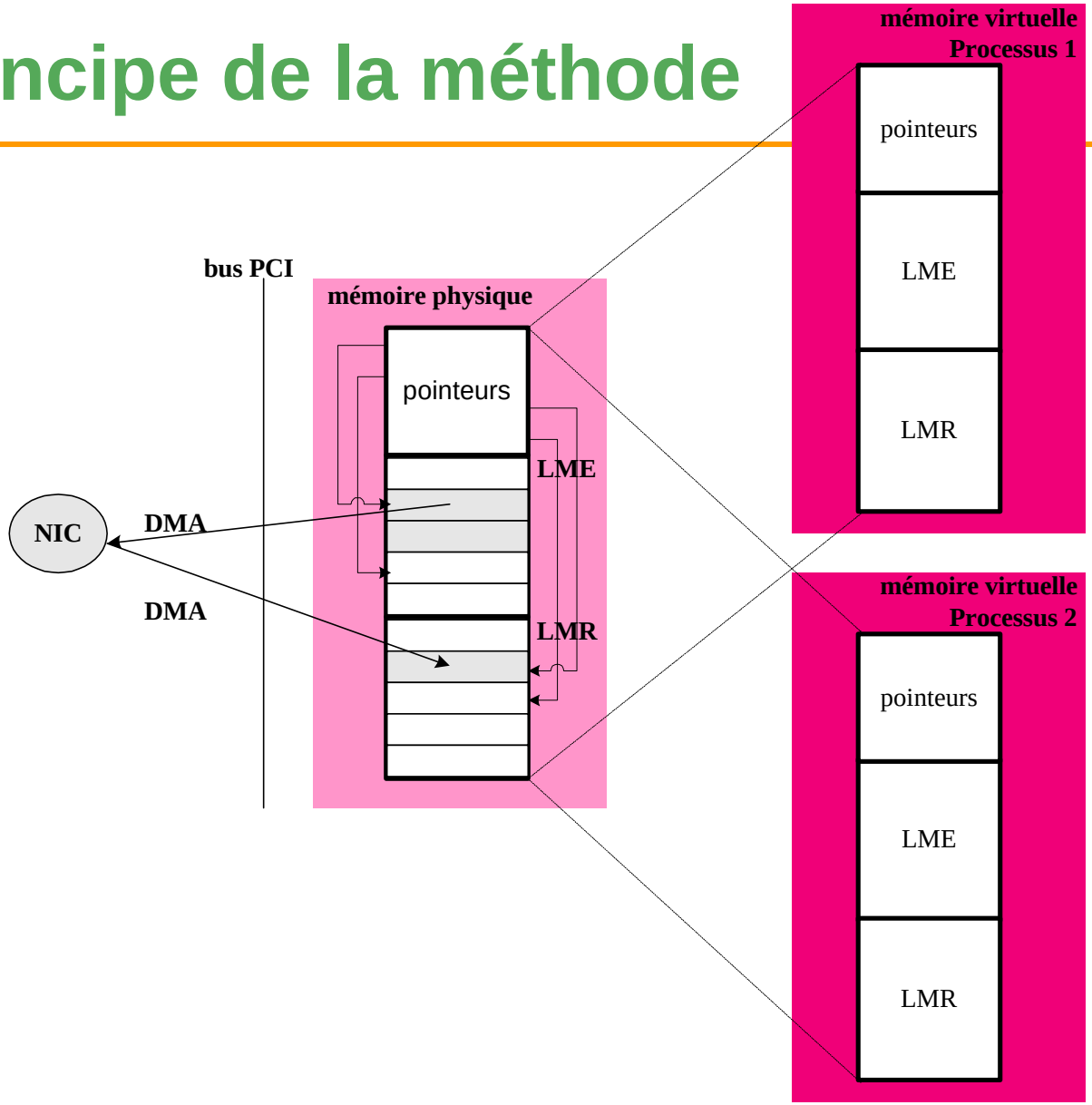
Une écriture distante en mode utilisateur



Les problèmes à résoudre

- **Comment protéger l'accès en mode utilisateur aux ressources partagées : LME, LMR, registres internes du contrôleur réseau ?**
- **Comment réaliser une signalisation par scrutation ?**
- **Comment garantir que toutes les entrées d'un même message soient consécutives dans la LME ?**

Principe de la méthode



La signalisation

On a rajouté deux primitives pour mettre à jour les états de toutes les communications en cours :

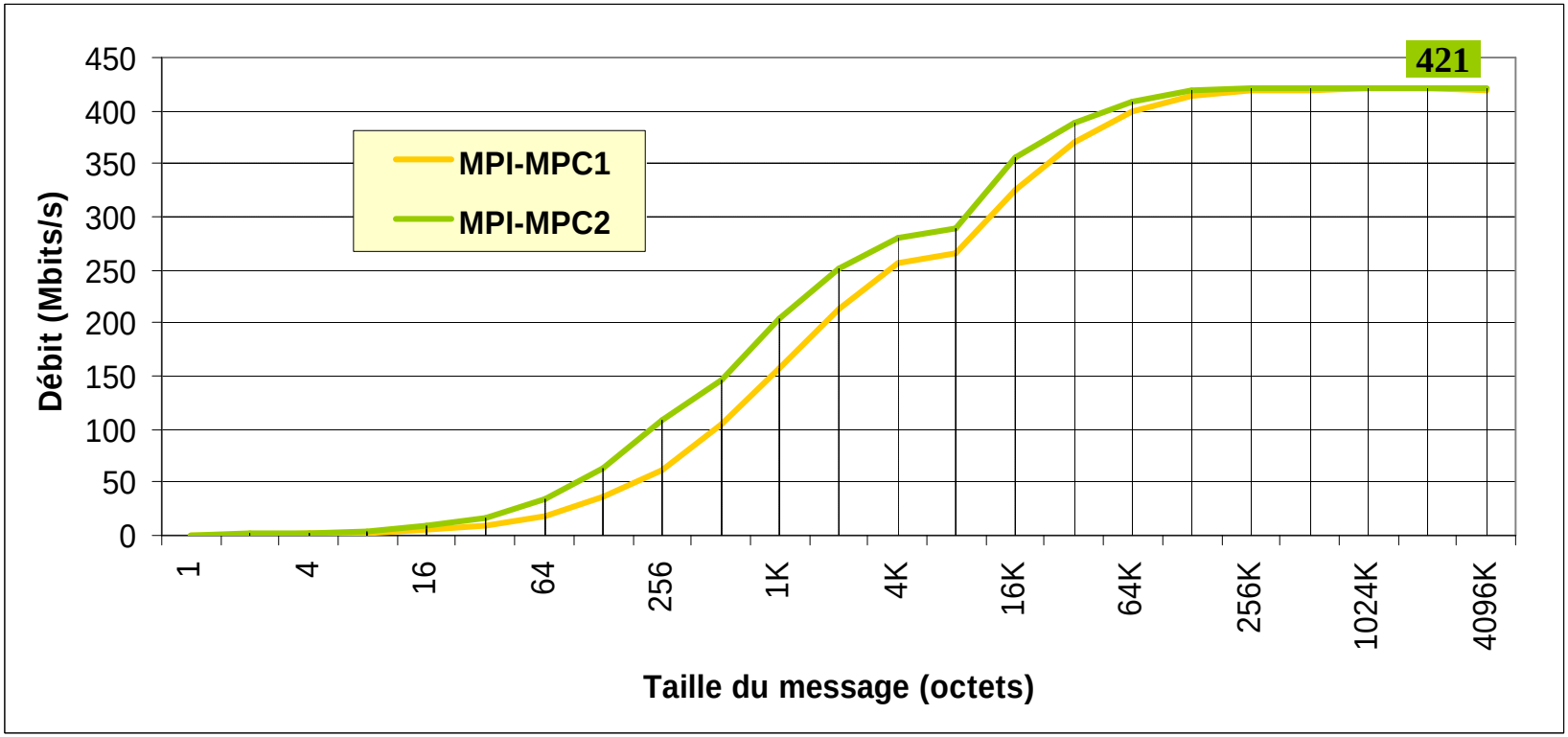
- **scrutation de la LME en émission**
- **scrutation de la LMR en réception**

La scrutation est à l'initiative de MPI

Les verrous

- verrou = variable mémoire partagée entre les différents processus
- à chaque opération est associé un verrou
- prise et libération d'un verrou atomique au niveau du processeur (test&set en mode utilisateur)
- plusieurs verrous pour ne verrouiller que ce qui est nécessaire à l'opération en cours

Performances de MPI-MPC2



	Latence (µs)	Débit Max. (Mb/s)	Seuil (Ko)	Demi débit (Ko)	Latence RDV (µs)
MPI-MPC1	26	419	16	2	105
MPI-MPC2	**15**	**421**	**8**	**1**	**72**

Analyse des performances

	Appels systèmes		Interruptions		Recopies		RDMA_SEND	
Protocole	short	rdv	short	rdv	short	rdv	short	rdv
MPI-MPC1	1	1+6F	2	2+4F	2	0	1	1+F+NB_DMA*F
MPI-MPC2	**0**	**4F**	**0**	**0**	**2**	**0**	**1**	**1+F+NB_DMA*F**

En ce qui concerne la latence :

- **gain supérieur à 40% : de 26µs avec MPI-MPC1 à 15µs**
- **15µs = 5µs (matérielle) + 10µs (logicielle soit 3500 cycles)**
- **gain significatif en débit en ce qui concerne les messages de petite taille**

En ce qui concerne le débit maximum :

- **très faible gain car indépendant de la taille des données**

Plan

1. Introduction : contexte et objectifs
2. MPI-MPC1 : MPI sur une primitive d'écriture distante
3. MPI-MPC2 : optimisations bas niveau
4. MPI-MPC3 : traductions d'adresses optimisées
5. Résultats expérimentaux
6. Conclusion

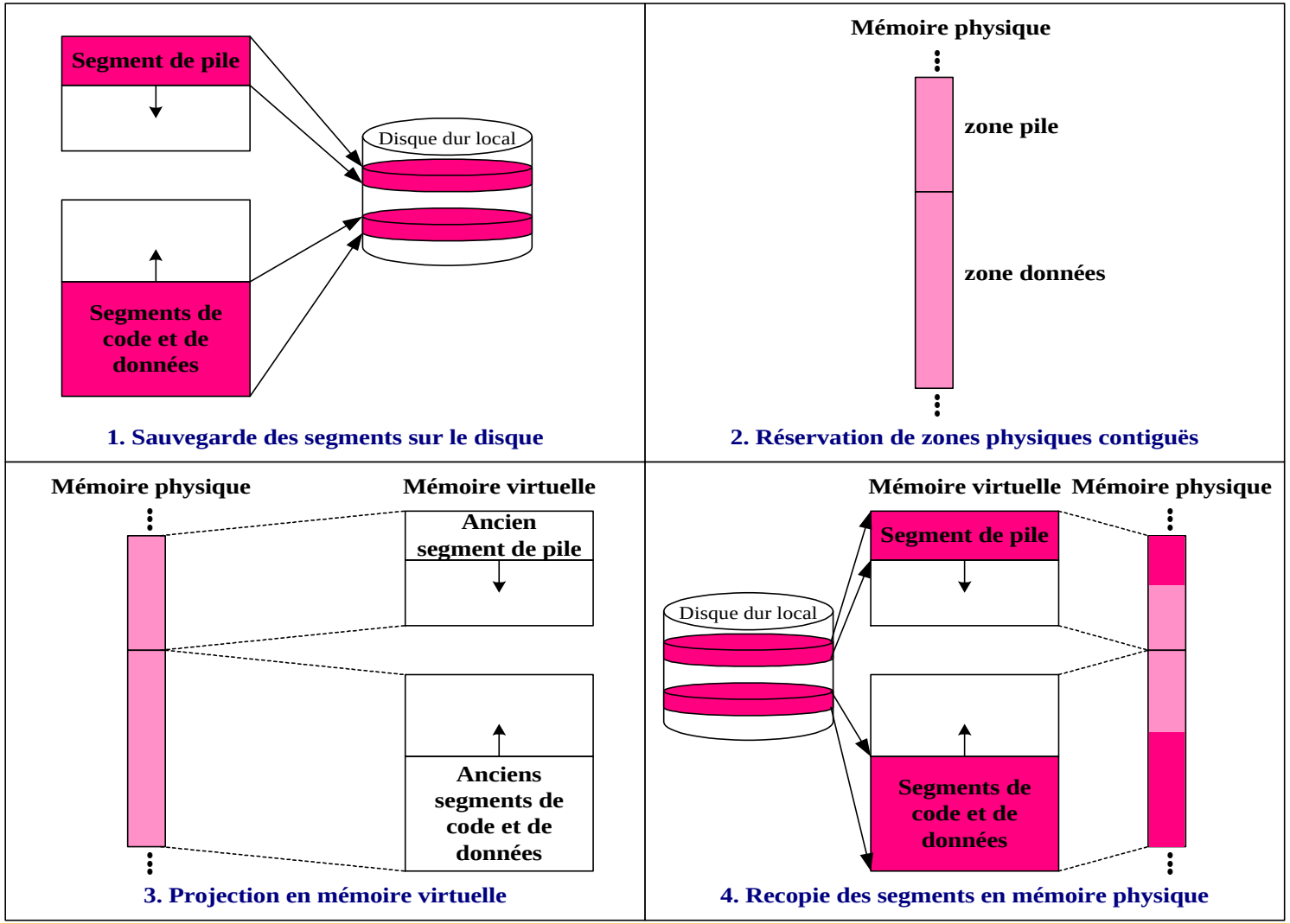
Objectifs de MPI-MPC3

- Supprimer les appels système liés aux traductions d'adresses
- S'affranchir de la discontinuité des tampons de l'application en mémoire physique
- Faire en sorte que la méthode proposée soit transparente :
  - pas de modification de l'OS et de la librairie C
  - pas d'ajout de primitives au standard MPI
  - pas de modification du code de l'application, compatibilité binaire (pas besoin de recompiler les fichiers objets)

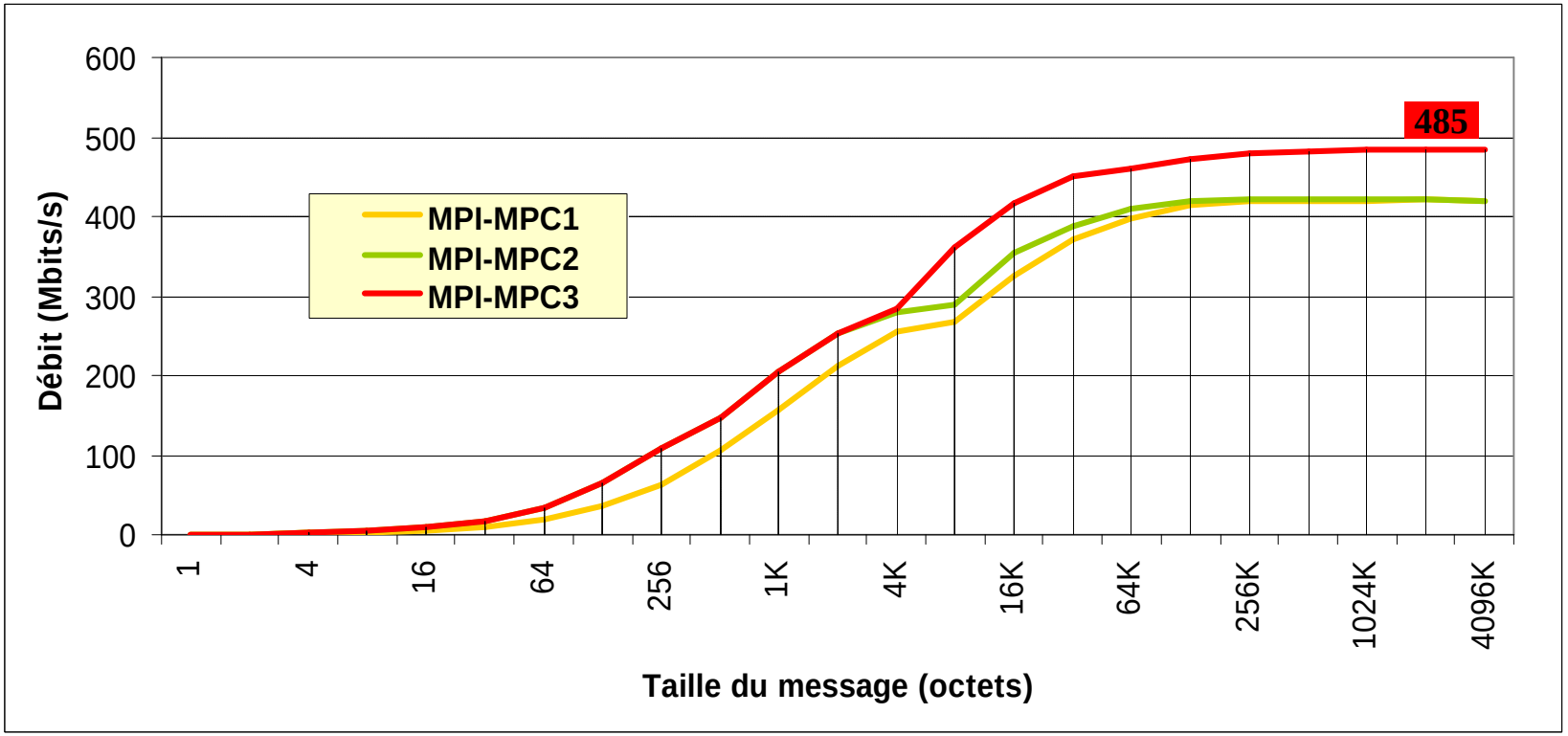
⇨ Pour toutes les données susceptibles d'être transférées :

adresse virtuelle = adresse physique + déplacement

Principe de la méthode



Performances de MPI-MPC3



	Latence (µs)	Débit Max. (Mb/s)	Seuil (Ko)	Demi débit (Ko)	Latence RDV (µs)
MPI-MPC1	26	419	16	2	105
MPI-MPC2	15	421	8	1	72
MPI-MPC3	**15**	**485**	**4**	**1,5**	**49**

Analyse des performances

	Appels systèmes		Interruptions		Recopies		RDMA_SEND	
Protocole	short	rdv	short	rdv	short	rdv	short	rdv
MPI-MPC1	1	1+6F	2	2+4F	2	0	1	1+F+NB_DMA*F
MPI-MPC2	0	4F	0	0	2	0	1	1+F+NB_DMA*F
MPI-MPC3	**0**	**0**	**0**	**0**	**2**	**0**	**1**	**1+F+F**

- **Suppression des appels système liés aux traductions d'adresses**
- **Réduction du nombre d'écritures distantes pour transmettre un message DATA : de plusieurs centaines à 1 seule pour transmettre 1024Ko**
- **Réduction de la taille du message RSP dans le protocole de rendez-vous : de plusieurs Ko à 28 octets**

Plan

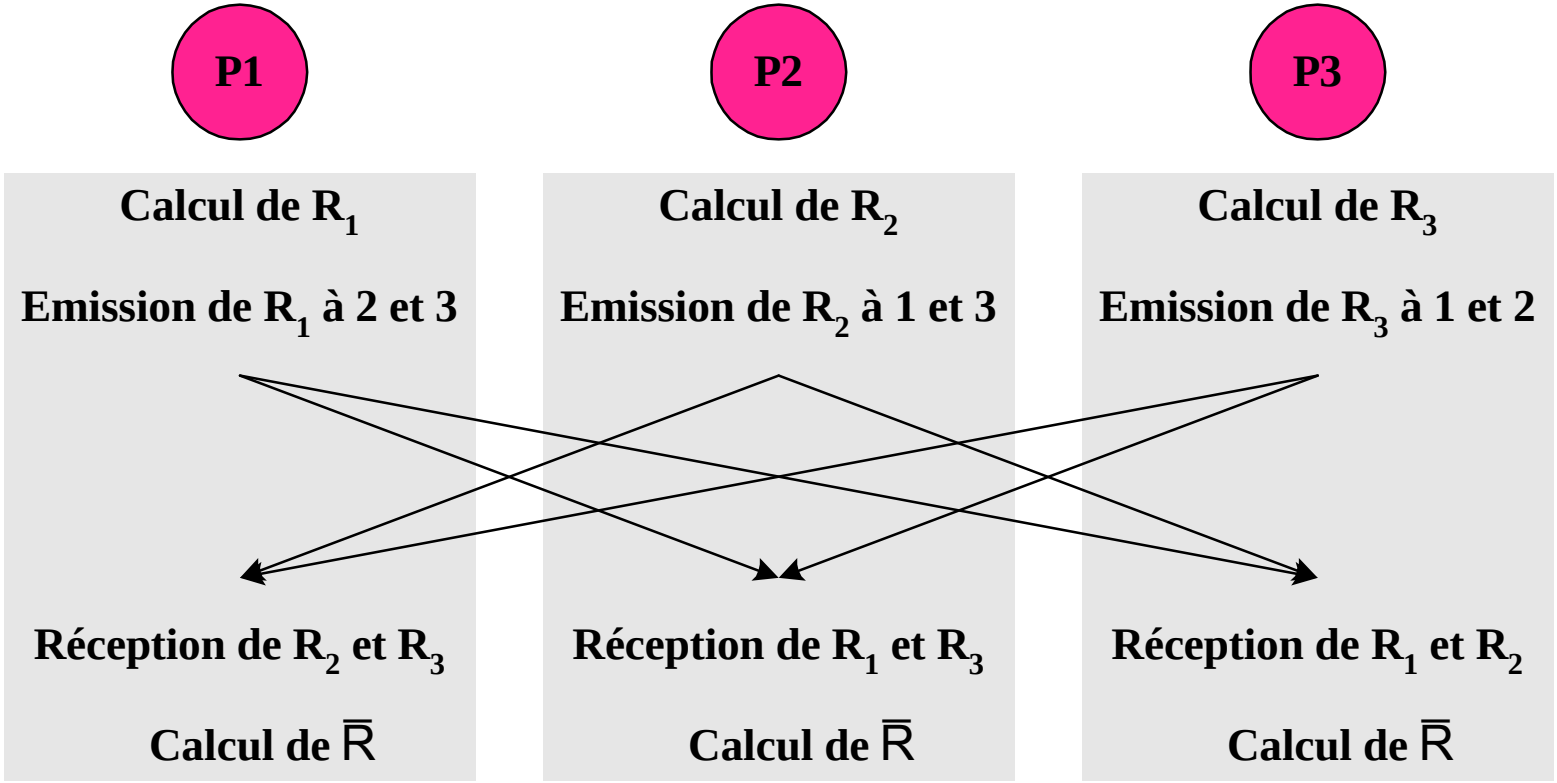
1. Introduction : contexte et objectifs
2. MPI-MPC1 : MPI sur une primitive d'écriture distante
3. MPI-MPC2 : optimisations bas niveau
4. MPI-MPC3 : traductions d'adresses optimisées
5. Résultats expérimentaux
6. Conclusion

Résultats avec CADNA

CADNA : logiciel pour prendre en compte les erreurs d'arrondi dans les calculs numériques

Résultats avec CADNA

CADNA : logiciel pour prendre en compte les erreurs d'arrondi dans les calculs numériques



Résultats avec CADNA

CADNA : logiciel pour prendre en compte les erreurs d'arrondi dans les calculs numériques

Taille du système linéaire	Nombre d'échanges	Temps des communications (sec.)		Temps d'un échange (µs)	
		MPI-MPC1	MPI-MPC2	MPI-MPC1	MPI-MPC2
800	646682	51	31	79	48
1200	1450450	101	66	70	46
1600	2574140	191	128	74	50
2000	4018285	288	177	72	44
Temps moyen d'un échange (µs)				**74**	**47**

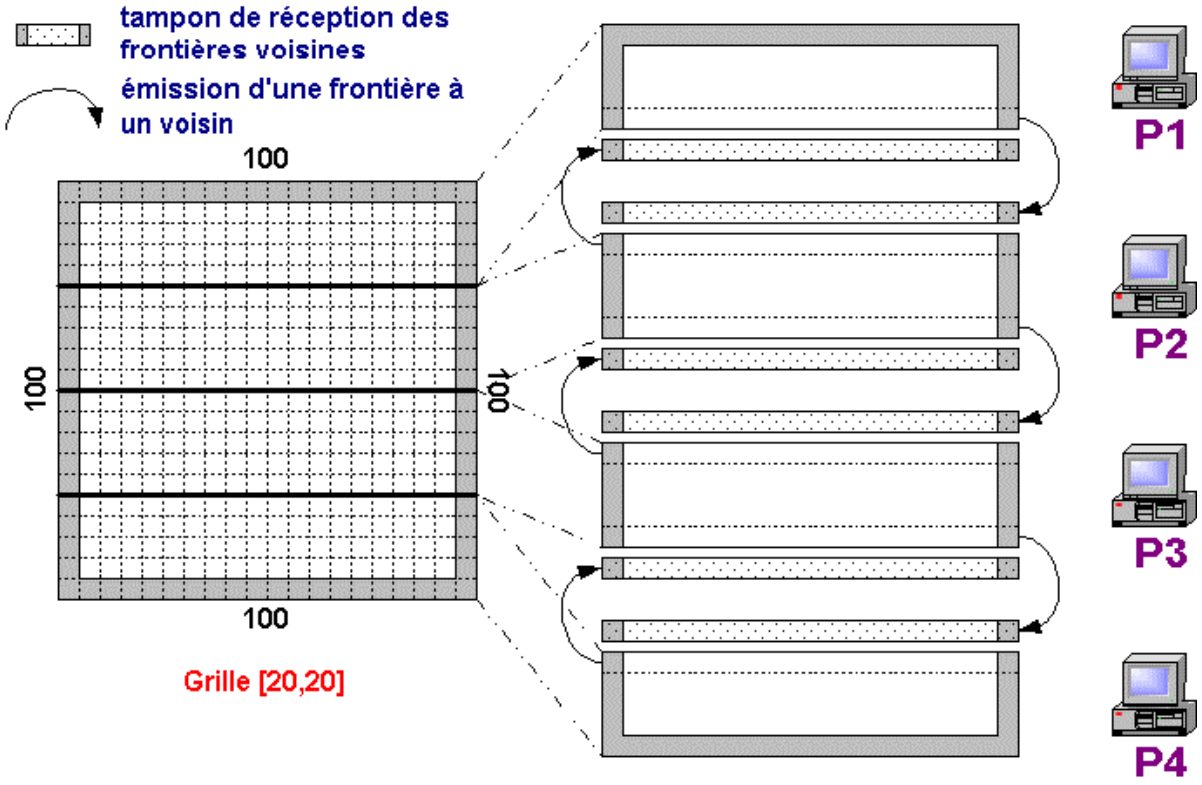
⇨ Gain de 36% avec MPI-MPC2

Résultats avec Laplace

Résolution de l'équation de Laplace par la méthode de Jacobi sur 4 processeurs

Résultats avec Laplace

Résolution de l'équation de Laplace par la méthode de Jacobi sur 4 processeurs



Résultats avec Laplace

Résolution de l'équation de Laplace par la méthode de Jacobi sur 4 processeurs

Nombre de colonnes	Nombre d'itéra-tions	Temps séquentiel (sec.)	Temps d'exécution parallèle (sec.)		*Speed-up*	
			MPI-MPC2	MPI-MPC3	MPI-MPC2	MPI-MPC3
12800	3150	327,6	92,6	92,3	3,5	3,5
25600	3150	689,3	187,9	185,4	3,7	3,7
51200	3150	1441,3	391,5	384,8	3,7	3,7

Nombre de colonnes	Taille Frontière (Ko)	Nombre d'échanges de frontières	Temps d'échanges des frontières (sec.)		Différence (sec.)	Gain (%)
			MPI-MPC2	MPI-MPC3		
12800	50	1,5	15,8	13,9	1,9	12%
25600	100	1,5	31,0	26,9	4,1	13%
51200	200	1,5	63,4	55,3	8,1	13%

Plan

1. Introduction : contexte et objectifs
2. MPI-MPC1 : MPI sur une primitive d'écriture distante
3. MPI-MPC2 : optimisations bas niveau
4. MPI-MPC3 : traductions d'adresses optimisées
5. Résultats expérimentaux
6. Conclusion

Objectifs atteints

- **Une implémentation optimisée de MPI sur une primitive d'écriture distante générique (RDMA)**
- **Suppression des appels système et des interruptions lors des phases de communication**

⇨ Gain en latence

- **Simplification des traductions d'adresses :**

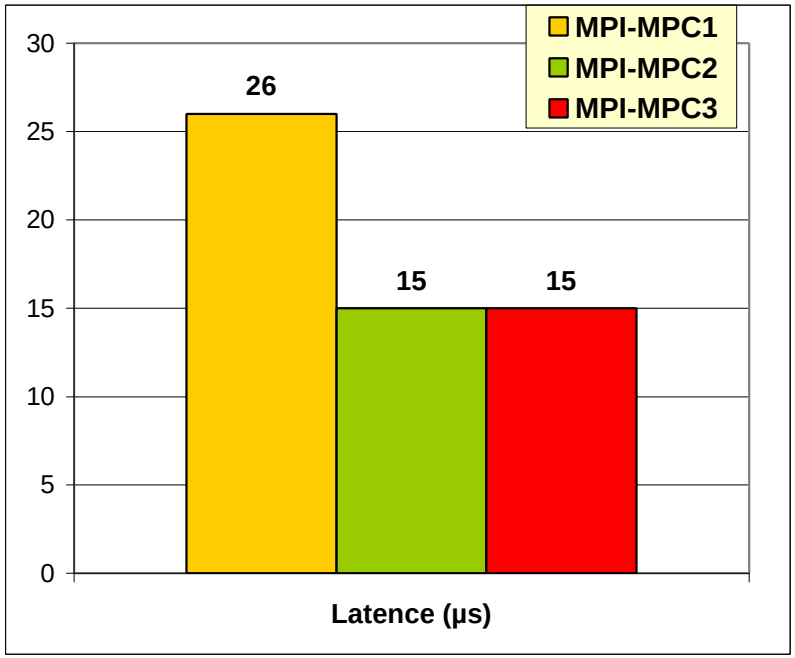
adresse virtuelle = adresse physique + déplacement

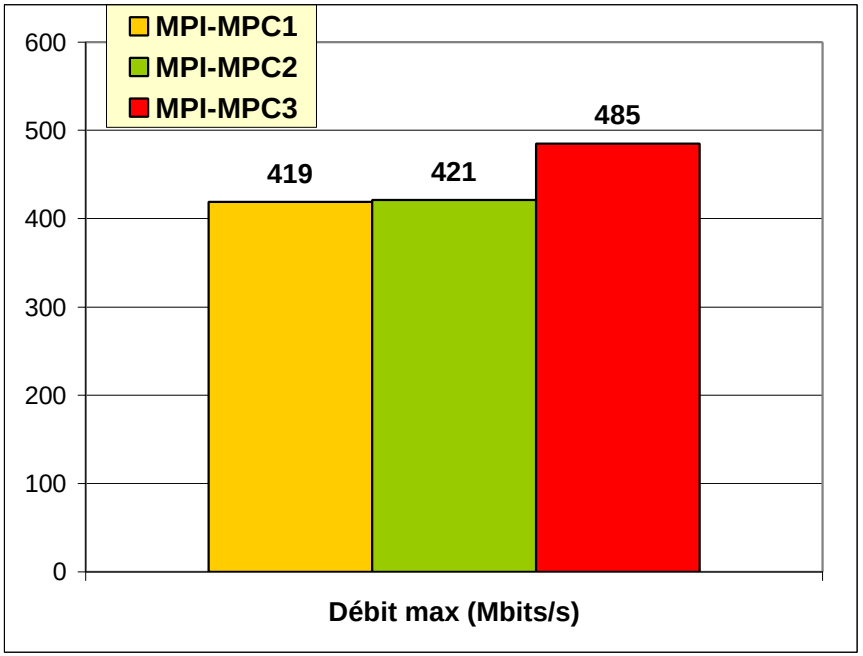
⇨ Gain en débit

- **Confirmation des résultats obtenus avec un ping-pong sur deux applications réelles**

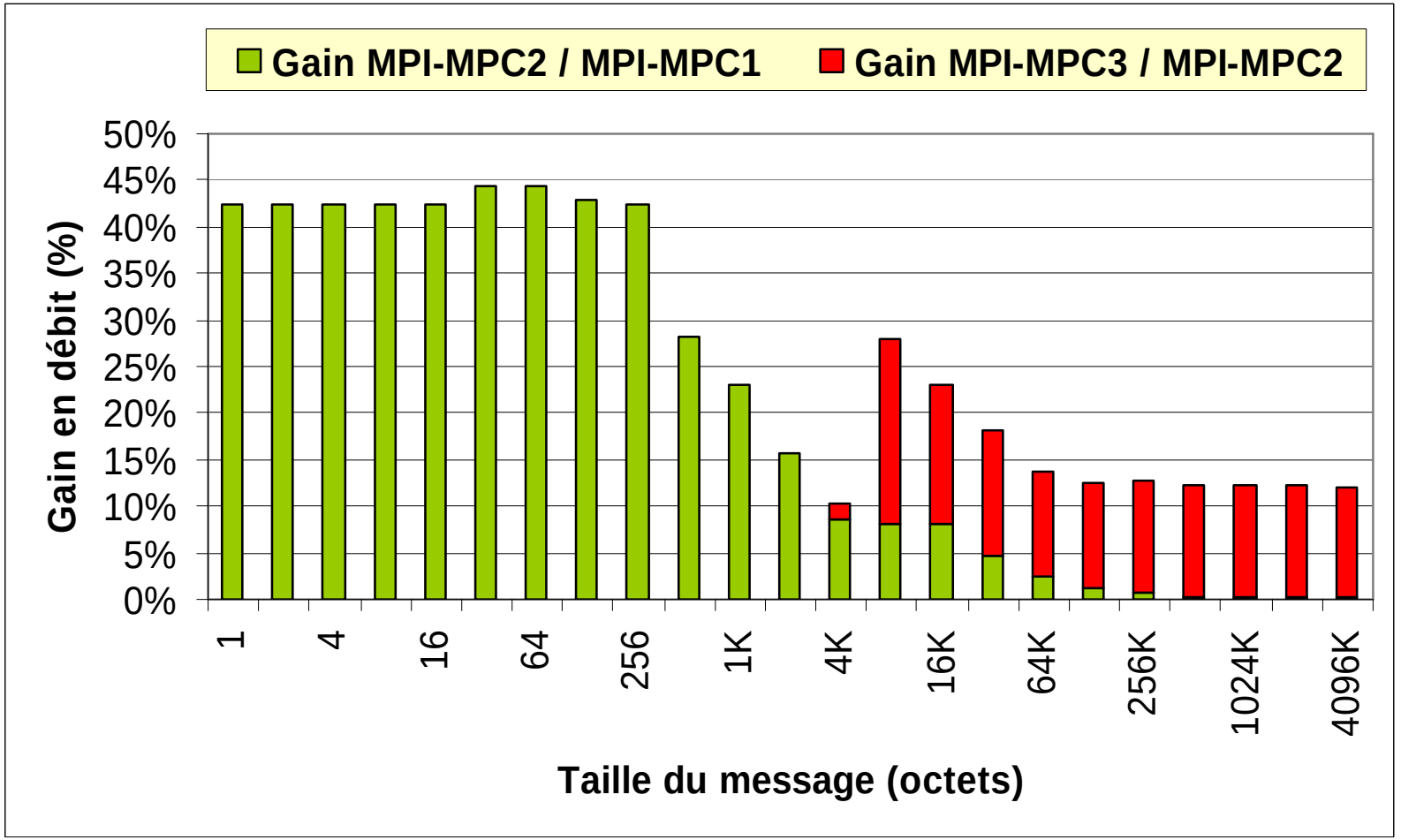
Résultats obtenus

Résultats obtenus





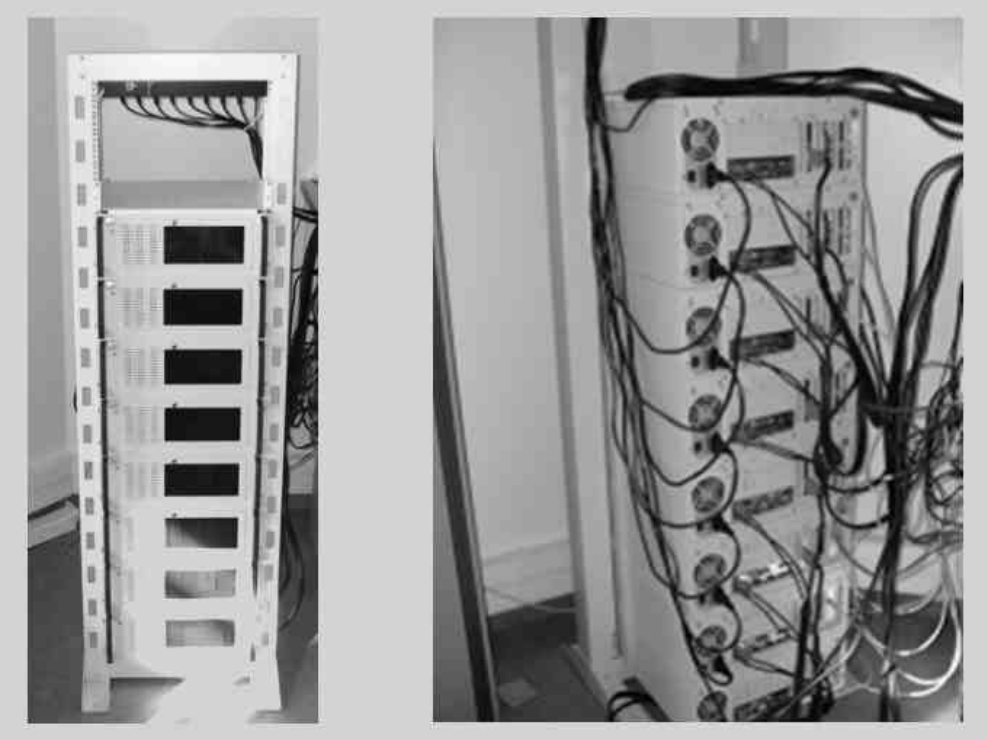
Résultats obtenus



Perspectives

- **Recouvrement calcul/communication : une implémentation multi-threads**
- **Le standard MPI2 :**
  - **création dynamique de tâches**
  - **primitives de communication unidirectionnelle sans action du processeur distant**
- **Portage sur d'autres plate-formes matérielles :**
  - **Myrinet**
  - **ANI/HSL**

Optimisations de la bibliothèque de communication MPI pour machines parallèles de type « grappe de PCs » sur une primitive d'écriture distante

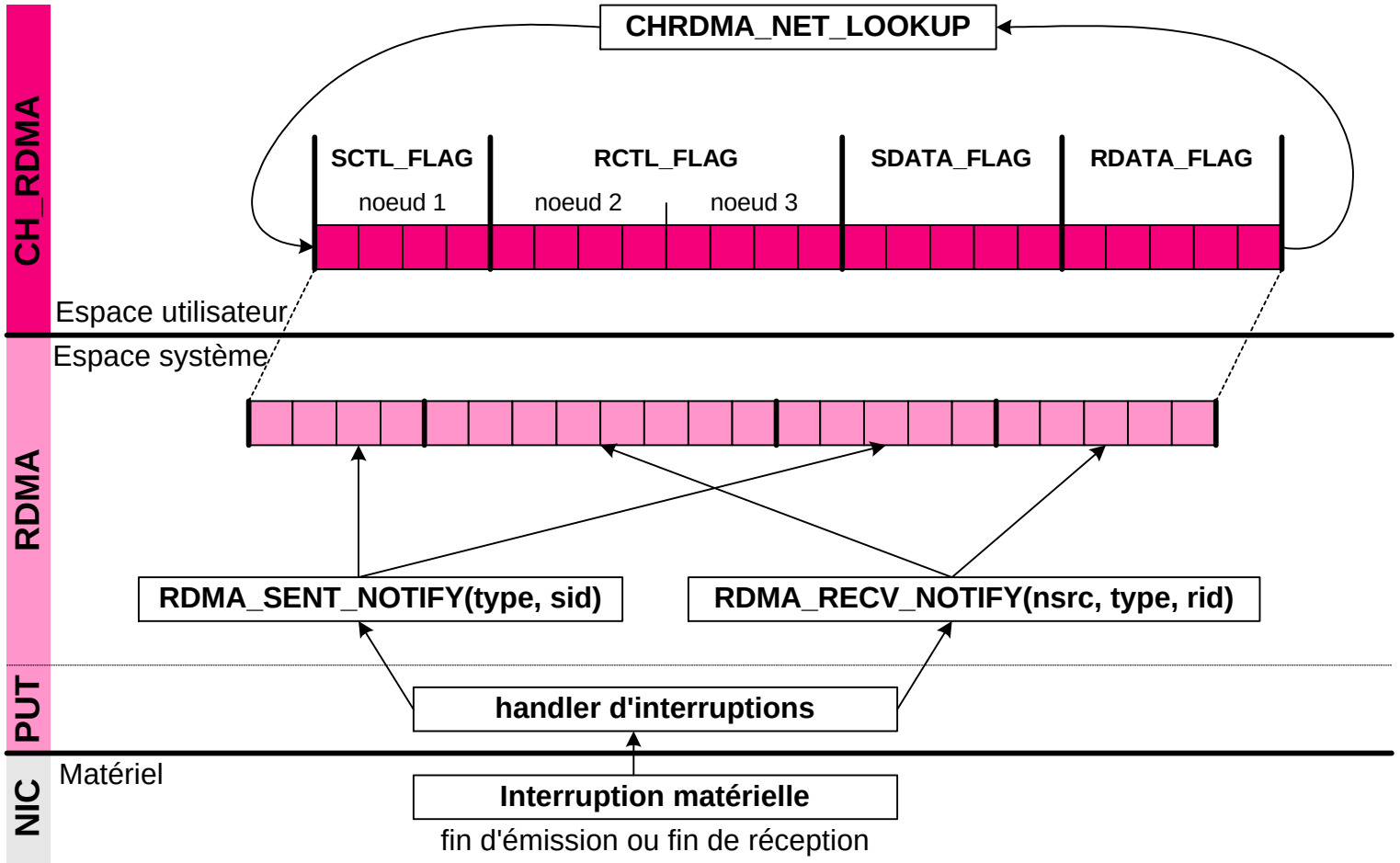


Olivier Glück

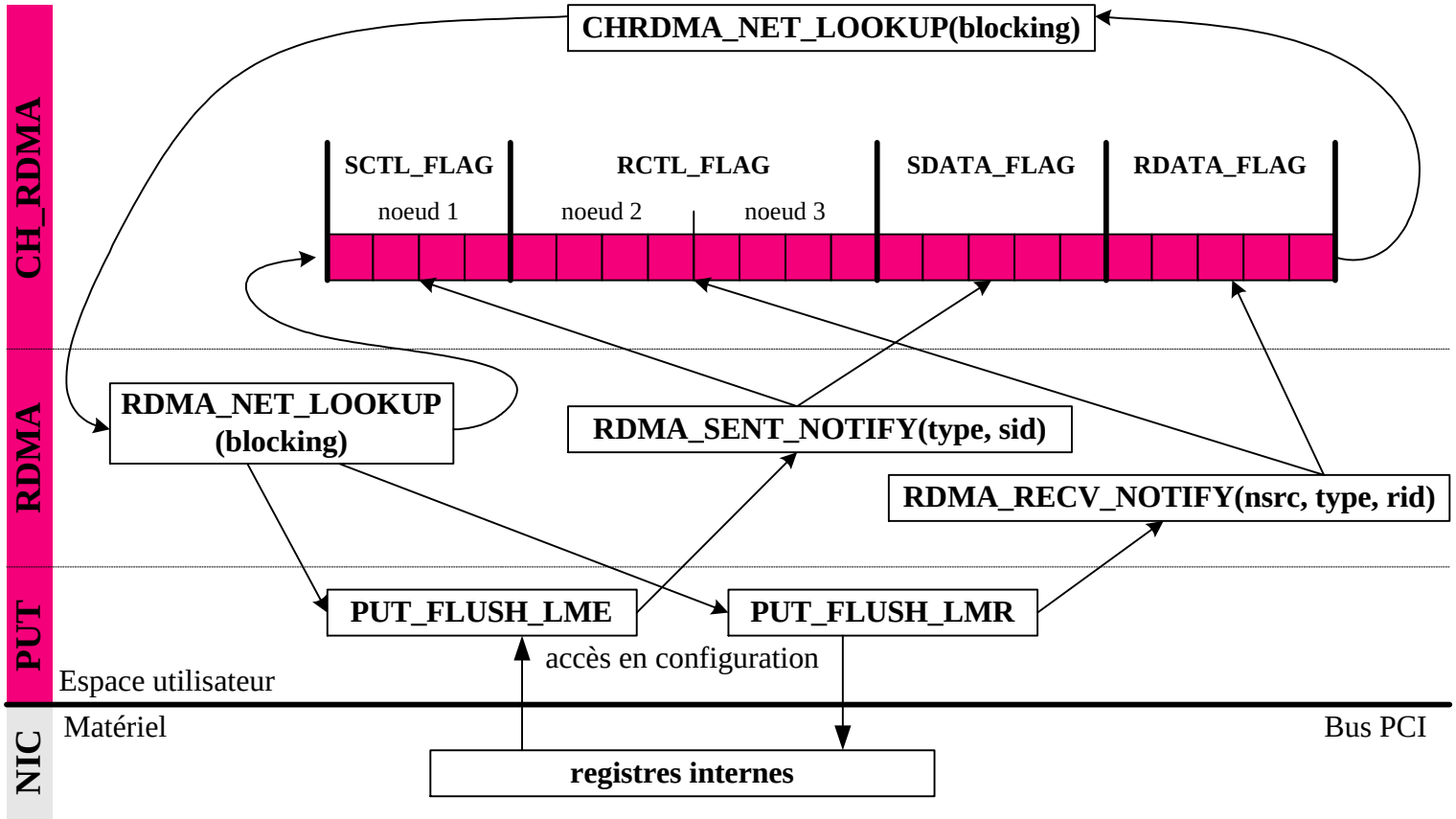
UPMC/LIP6/ASIM

Olivier.Gluck@lip6.fr

La signalisation

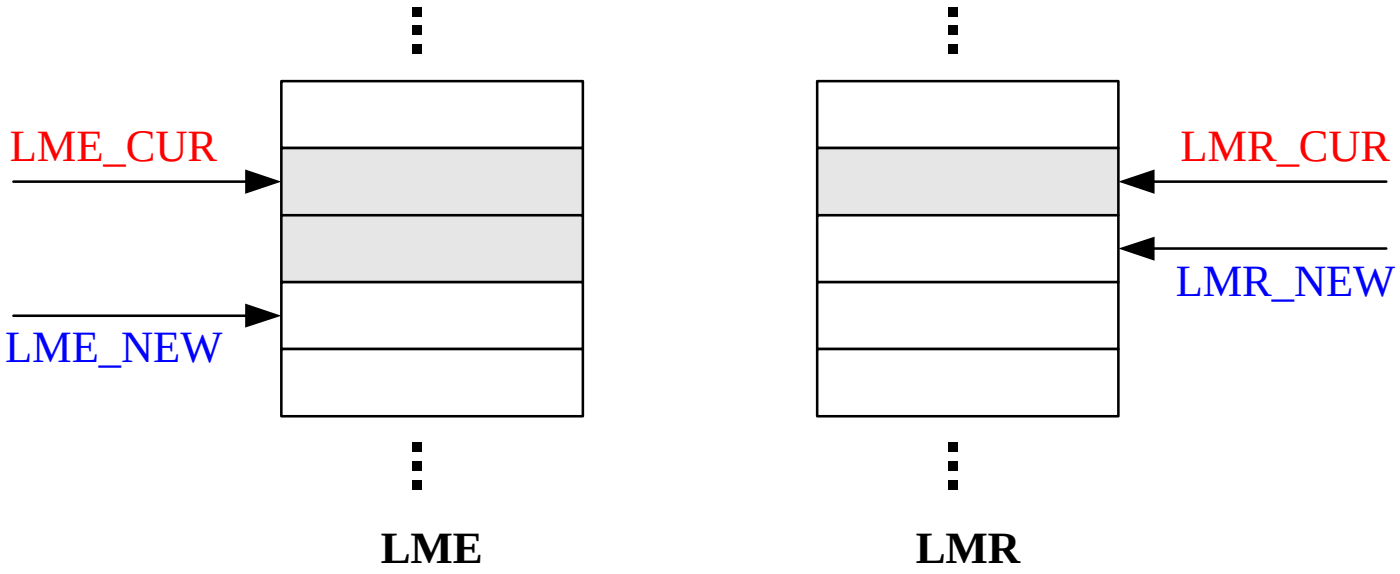


La signalisation



Les ressources partagées

- La Liste des Messages à Emettre (LME)
- La Liste des Messages Reçus (LMR)
- Les registres internes du contrôleur réseau

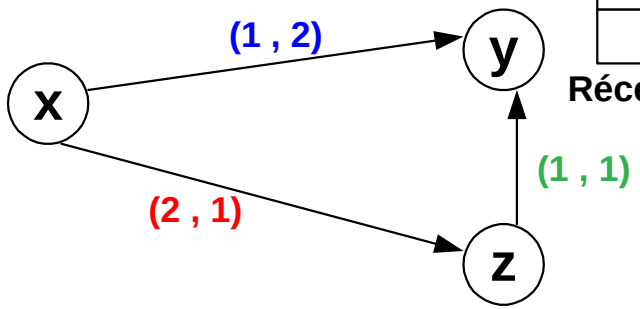


Les émissions/réceptions simultanées

Pourquoi : pouvoir traiter plusieurs émissions/réceptions simultanément ⇨ besoin d'identifier une communication

sid	rid
1	2
2	1
3	libre
4	libre

Emissions de x



rid	sid
1	1
2	1
3	libre
4	libre

Réceptions de y

On utilise des tables associatives :

- **sid** choisi par l'**émetteur**
- **rid** choisi par le **récepteur**

1 canal (sid,rid) à la durée de vie d'un message

sid	rid
1	1
2	libre
3	libre
4	libre

Emissions de z

rid	sid
1	2
2	libre
3	libre
4	libre

Réceptions de z

Les problèmes à résoudre

Comment faire pour projeter les zones contiguës en mémoire physique dans la mémoire virtuelle du processus à l'emplacement des anciens segments ?

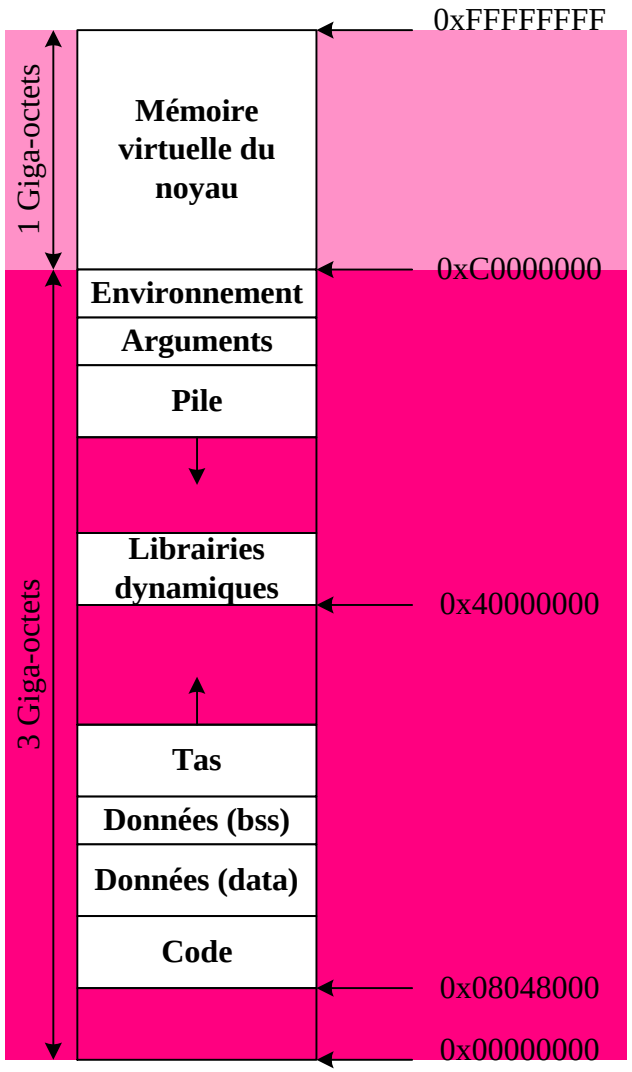
⇨ **utilisation d'une librairie dynamique : exécution de code en dehors du segment de code du processus**

⇨ **déplacement du segment de pile du processus**

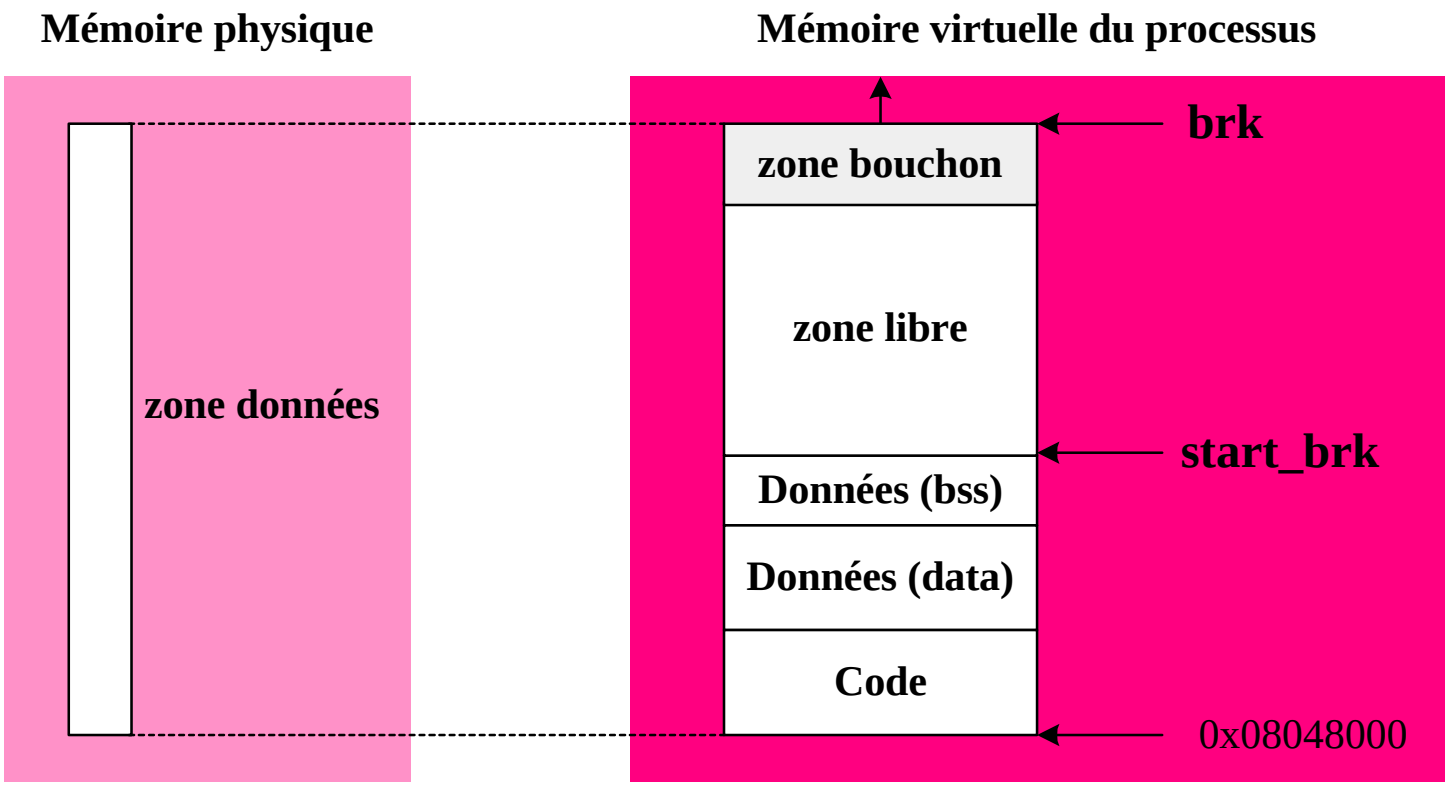
Comment réaliser ces opérations de façon transparente pour l'utilisateur avant l'exécution de l'application ?

⇨ **utilisation d'un langage objet permettant d 'exécuter du code avant le début de l'application : ces opérations sont réalisées dans le constructeur de l'objet**

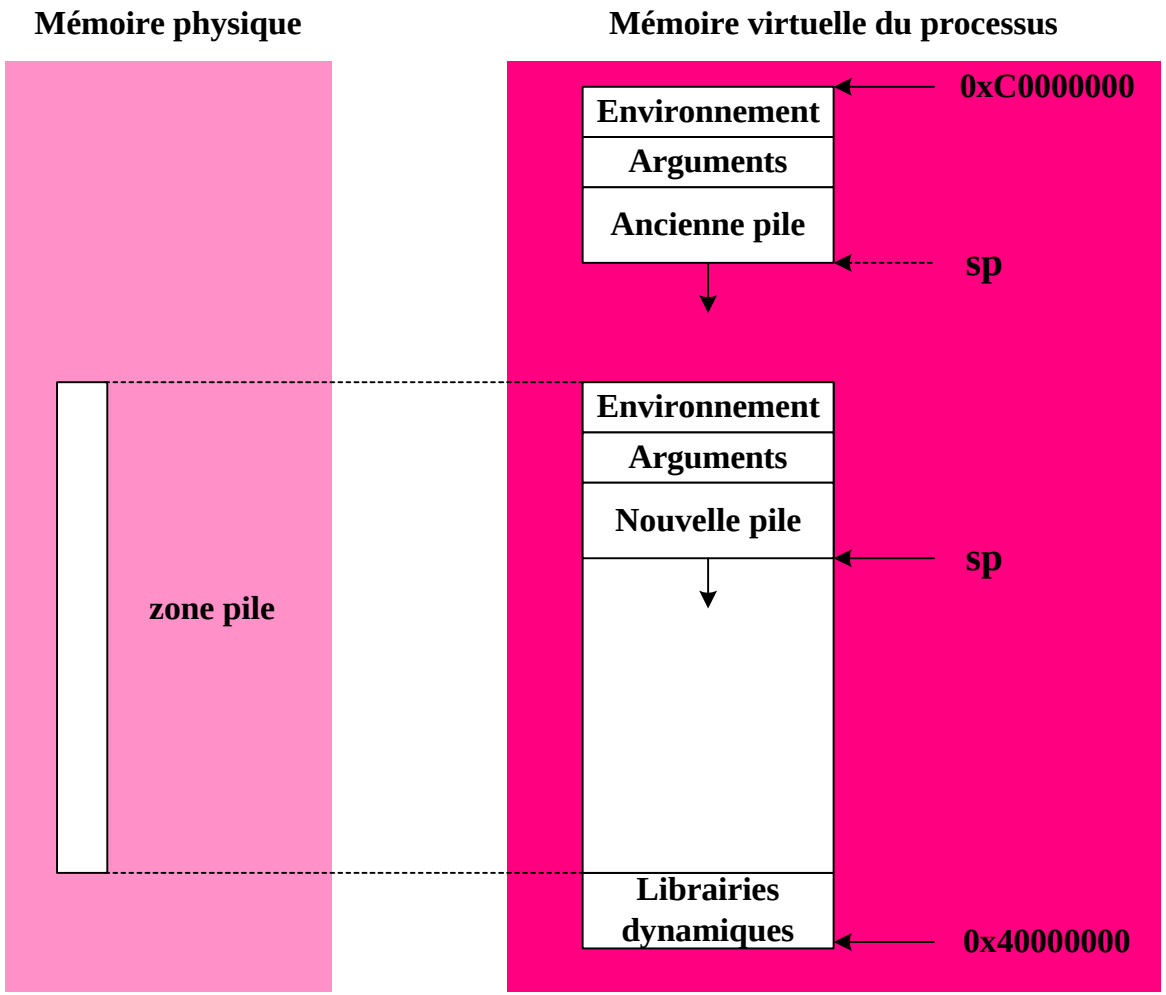
Espace d'adressage



Le tas


Mémoire physique
Mémoire virtuelle du processus
zone données
zone bouchon
zone libre
Données (bss)
Données (data)
Code
brk
start_brk
0x08048000

La pile


Mémoire physique
Mémoire virtuelle du processus
0xC0000000
Environnement
Arguments
Ancienne pile
sp
Environnement
Arguments
Nouvelle pile
sp
zone pile
Librairies dynamiques
0x40000000

Compilation

```
class Redistrib
{
  Redistrib();
  ~Redistrib();
  ...
}
```

Redistrib.hh

```
#include "Redistrib.hh"
// constructeur
Redistrib::Redistrib()
{
  //opérations à réaliser
  //avant le main
  ...
}
// destructeur
Redistrib::~Redistrib()
{
  ...
}
```

Redistrib.cc

Librairie dynamique (libRedistrib.so)

```
main()
{

  ...

}
```

Monappli.c ou **Monappli.cc**

```
#include "Redistrib.hh"

Redistrib MPC;
```

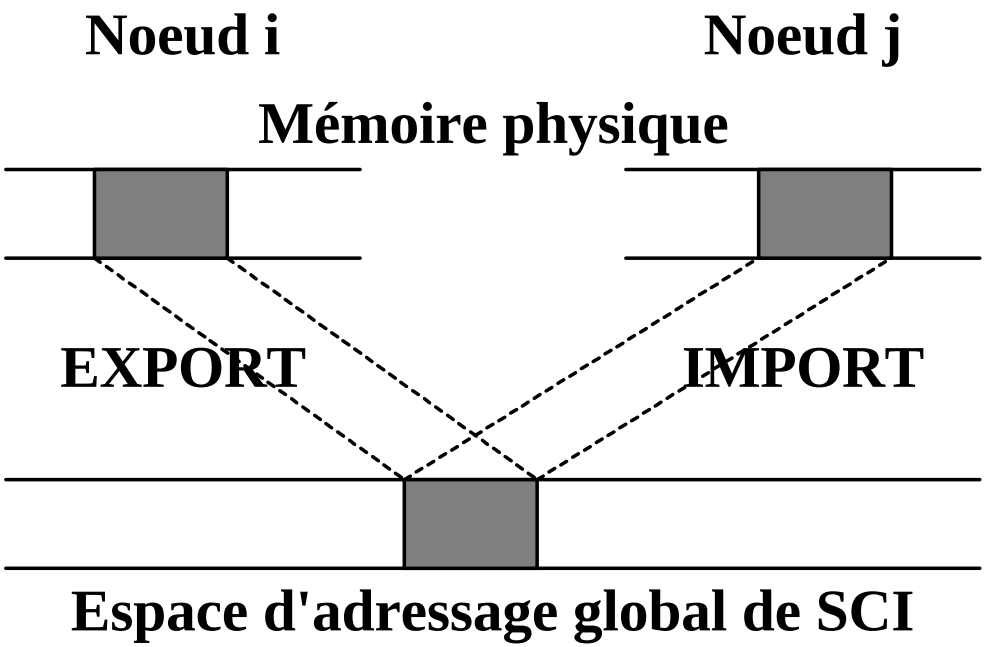
mpc.cc

Application MPI

Les différentes étapes

- informations utilisateur
- taille des zones « pile » et « données »
- augmentation taille du tas
- sauvegarde sur disque
- réservation mémoire physique
- projection
- recopie des segments sauvegardés
- translation du pointeur de pile
- modification des arguments
- empilement des paramètres du main puis call
- restauration du contexte + ret
- appel du destructeur

Le réseau SCI



- lectures/écritures distantes mais paradigme de programmation différent
- zones exportées/importées limitées
- correspondance virtuelle/physique dans l'interface réseau

GM

- « registering memory » : primitives pour verrouiller un tampon et informer le contrôleur réseau de la correspondance virtuelle/physique
- gm_directed_send(port, src_buf, dst_buf, len, dst_id, …) : écriture distante vers une zone enregistrée au préalable (désactivé par défaut)