Multithreading et Calcul Hautes Performances

L’approche OpenMP

Le standard OpenMP

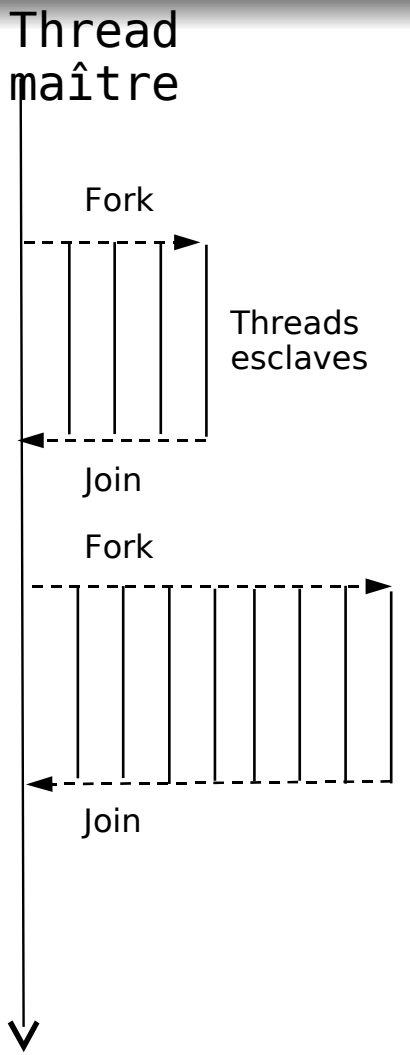
- Interface pour écrire des applications multithreads portables (sur SMP)
  - Objectif = simplicité + portabilité
  - Directives de compilation (C, C++, Fortran) + routines spécifiques
  - http://www.openmp.org
  - V1.0 en 1997-1998, V5 en mai 2005
- Modèle de programmation
  - « Fork-Join », parallélisation des blocs et des boucles

Principaux compilateurs OpenMP

- Commerciaux
  - SGI MPISpro, C, C++
  - IBM XL Fortran, C, C++
  - Compaq Fortran
  - Portland Group Fortran, C, C++
  - Intel Fortran, C, C++
  - Fujitsu-Siemens Fortran
- Libres
  - OdinMP C, C++ (KTH Stockholm)
  - Omni C, Fortran77 (RWCP, Japon)
  - Nanos Fortran77
  - OMPi C (Univ. Ioannina, Univ. Kassel)
  - GOMP C, C++, Fortran95 (GNU)

À propos du modèle fork/join

- Thread maître
  - Exécute le code séquentiel
  - Fork : création de threads esclaves
- Threads esclaves
  - Exécutent la portion parallèle
  - Join : destruction des esclaves et retour du contrôle au maître
- Le nombre d'esclaves peut varier d'une région parallèle à l'autre
- Technique permettant de paralléliser incrémentalement un code



Programme simple

- Création d'une région parallèle

```
#include <openmp.h>
#include <stdio.h>

int main()
{

#pragma omp parallel
  printf("Hello World!\n");

}
```

Programme simple (suite)

- Exécution

```
[toto@biproc] ./main
Hello World!
Hello World!
[toto@biproc] export OMP_NUM_THREADS=4
[toto@biproc] ./main
Hello World!
Hello World!
Hello World!
Hello World!
```

Sections parallèles

- Exécution de codes différents en parallèle

```
int main()
{
#pragma omp parallel sections
   {
#pragma omp section
      f();
#pragma omp section
      g();
   }
}
```

À propos des régions parallèles

- Barrière de synchronisation implicite à la fin des régions parallèles
  - Le maître attend la fin des esclaves
    - Sauf si clause *nowait*
  - Attention à la répartition homogène du travail !
- Régions parallèles emboîtées
  - Pas supporté par tous les compilateurs
  - Difficulté d'évaluer le nombre de threads à créer par région

Parallélisation des boucles

```
int main()
{
   int i;
   double m[N];

#pragma omp parallel for
   for(i=0; i<N; i++)
      m[i] = f(i);
}
```

- Il existe des restrictions sur les opérateurs utilisables dans le for(;;)

Équilibrage de charge

```
int main()
{
  …
#pragma omp parallel for schedule(dynamic)
  for(i=0; i<N; i++)
    m[i] = f(i);
}
```

- Les itérations sont réparties dynamiquement (approche gloutonne) parmi les threads
  - Distribution par blocs possible :
    schedule(dynamic, bloc_size)

Directives utiles

- pragma omp barrier
- pragma omp flush(var, ...)

- pragma omp critical
- pragma omp atomic

- pragma omp master
- pragma omp single

Directives utiles

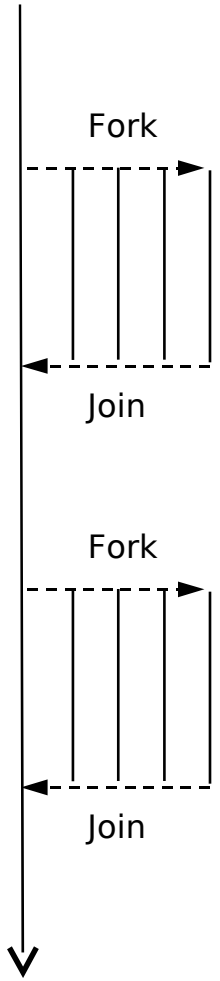
- private(var, ...)
- shared(var, ...)

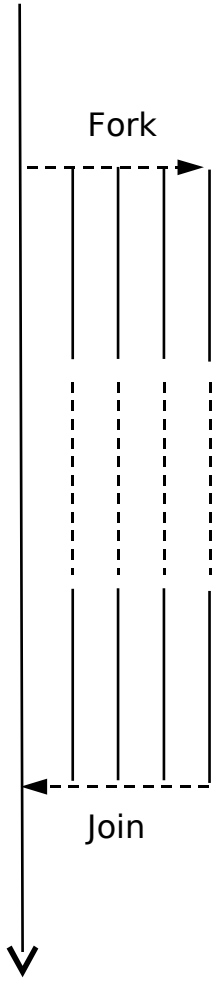
- firstprivate(var, ...)
- lastprivate(var, ...)

- reduction(op:var)

- nowait

Une histoire de grain…





Une histoire de grain…

```
{
#pragma omp parallel for
  for(i=0; i<N; i++)
      A[i] = f(i);

  {
    /* section séquentielle */
    …
  }

#pragma omp parallel for
  for(j=0; i<M; i++)
      B[j] = g(i);
}
```

```
#pragma omp parallel
{
#pragma omp for
  for(i=0; i<N; i++)
      A[i] = f(i);

#pragma omp master
  {
    /* section séquentielle */
    …
  }

#pragma omp for
  for(j=0; i<M; i++)
      B[j] = g(i);
}
```

Limites du langage OpenMP

- Parallélisme essentiellement « SPMD »
  - Applications irrégulières ?
  - Applications composites ?
- Exploitation des machines hiérarchiques
  - *Feedback* sur la topologie
- Expressivité des directives
  - Affinités threads <-> mémoire

Communications sur architectures distribuées

Objectifs du chapitre

- Comprendre
  - Les techniques utilisées au sein des bibliothèques de communication pour exploiter efficacement les réseaux rapides
  - Les limitations de ces bibliothèques

Plan du chapitre

- Caractéristiques des réseaux rapides contemporains
- Techniques de base pour l'obtention de bonnes performances
- Quelques interfaces de bas niveau
- L'interface de haut niveau MPI
- Points délicats
  - Recouvrement
  - Progression des communications non bloquantes
  - Messages complexes
- Vers des interfaces plus puissantes

Caractéristiques des réseaux rapides contemporains

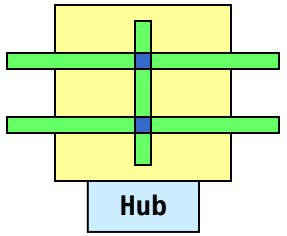
Principes de fonctionnement

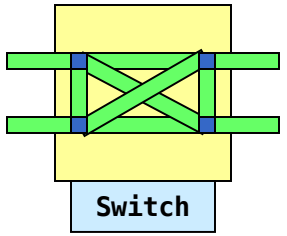
Les réseaux rapides

- Réseaux faible distance utilisés au sein des grappes de calculateurs
  - Myrinet, Quadrics, SCI, …
- Les performances sont remarquables
  - Faible latence: approx. 1 µs
  - Haut débit: approx. 10 Gb/s
- Les protocoles utilisés sont spécifiques et « légers »
  - Routage statique des messages
  - Pas nécessaire de fragmenter les paquets
  - Pas même nécessaire de former des paquets

(Fast|Giga)-Ethernet

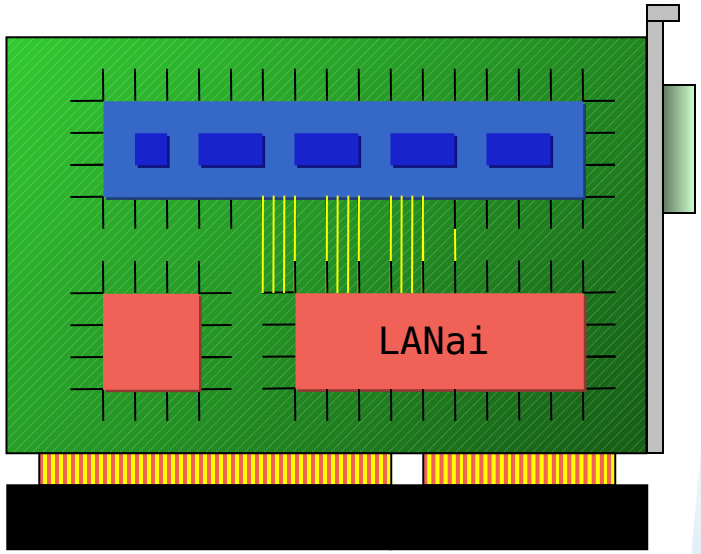
- Interconnexion
  - Hub ou switch
- Câblage
  - Cuivre ou fibre optique
- Latence
  - ~10 µs
- Débit
  - De 100 Mb/s à 10Gb/s
- Note
  - Compatibilité avec l’Ethernet classique





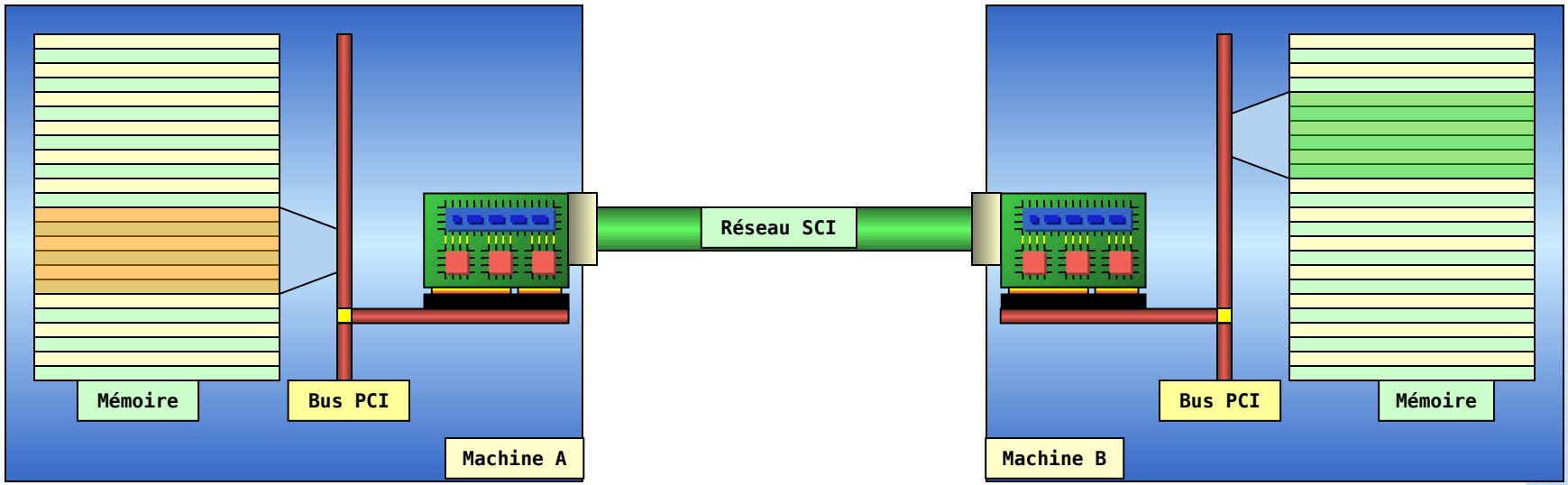
Myrinet

- Société Myricom
- Interconnexion
  - Switch
- Cartes équipées
  - D’un processeur
  - D’une mémoire SRAM ~ 4Mo
- Latence
  - 1~2 µs
- Débit
  - 10 Gb/s
- Note
  - Durée de vie limitée des messages (routage wormhole)



SCI

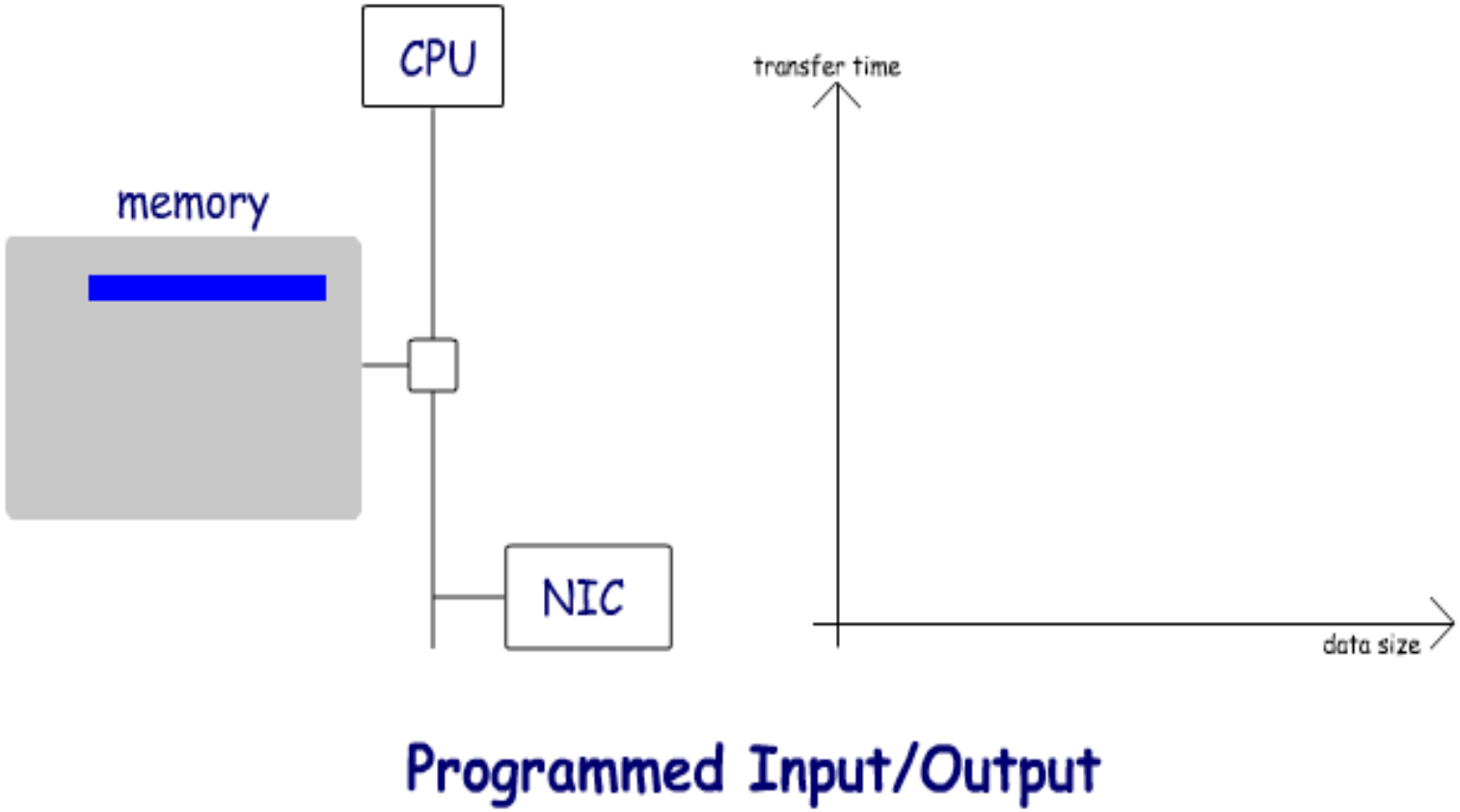
- Scalable Coherent Interface
  - Norme IEEE (1993)
  - Société Dolphin
- Fonctionnement par accès mémoire distants
  - Projections d'espaces d'adressage



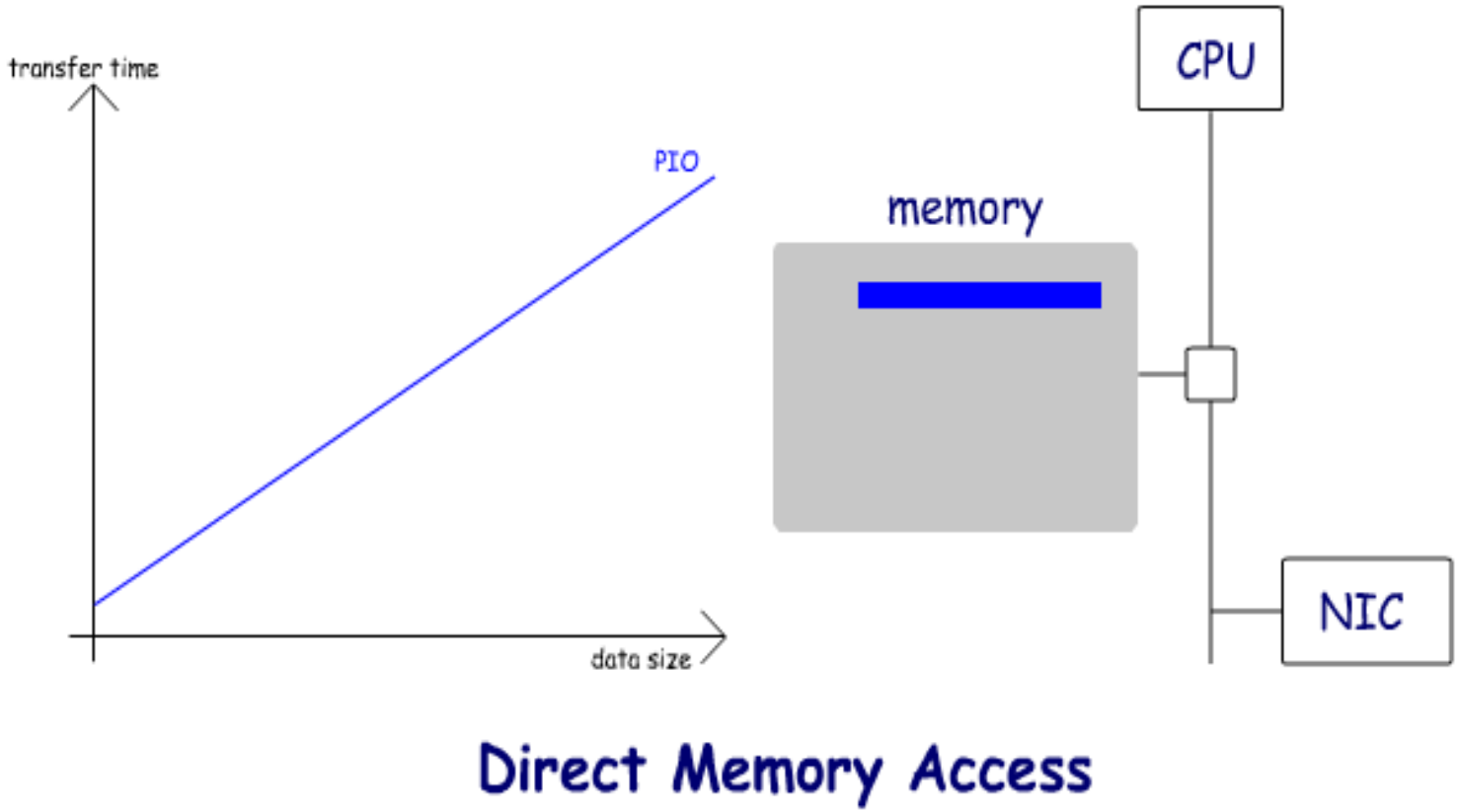
Techniques de base pour l’obtention de bonnes performances

Zéro-copie, OS-bypass, etc.

Interactions avec la carte réseau : le mode PIO



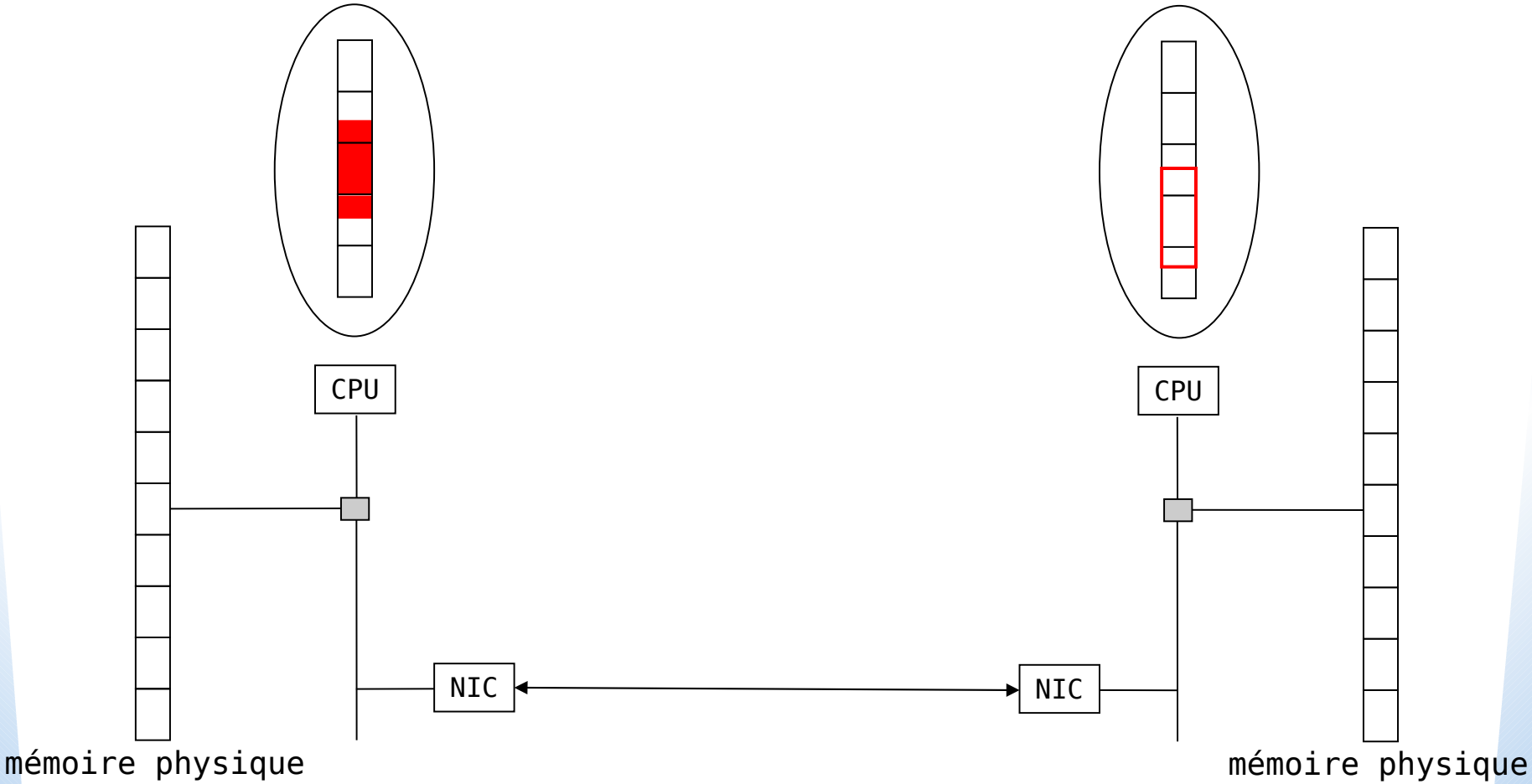
Interactions avec la carte réseau : le mode DMA



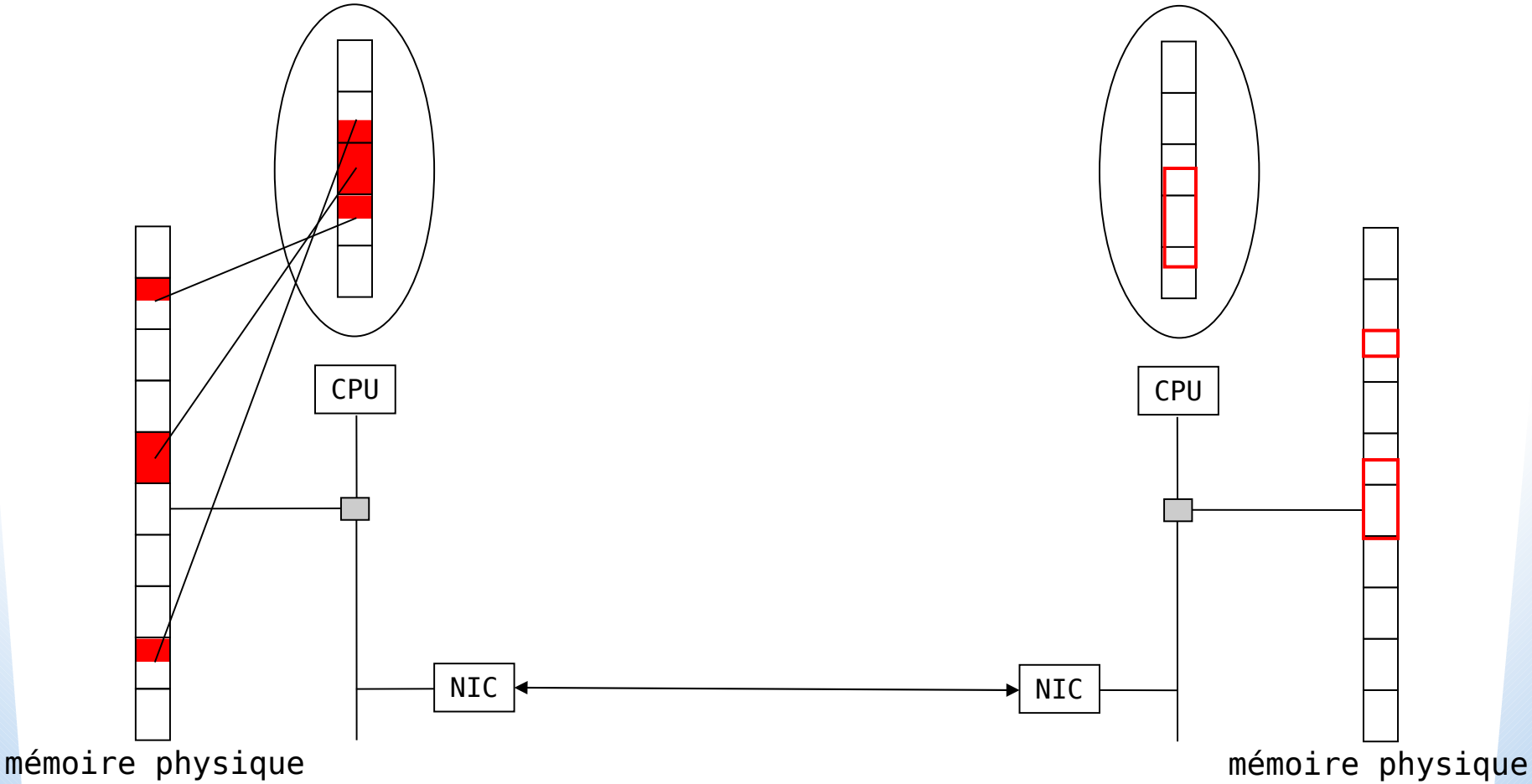
Transmissions zéro-copie

- Objectifs
  - Diminuer le temps de transfert
    - Le temps de recopie est non négligeable
      - Toutefois, il peut être masqué en partie par effet pipeline
  - Diminuer l'occupation du processeur
    - Actuellement, les « memcpy » sont effectués par les processeurs
- Idée
  - Faire en sorte que la carte réseau récupère/dépose les données directement depuis/vers la mémoire de l'application

Transmissions zéro-copie



Transmissions zéro-copie



Transmissions zéro-copie en émission

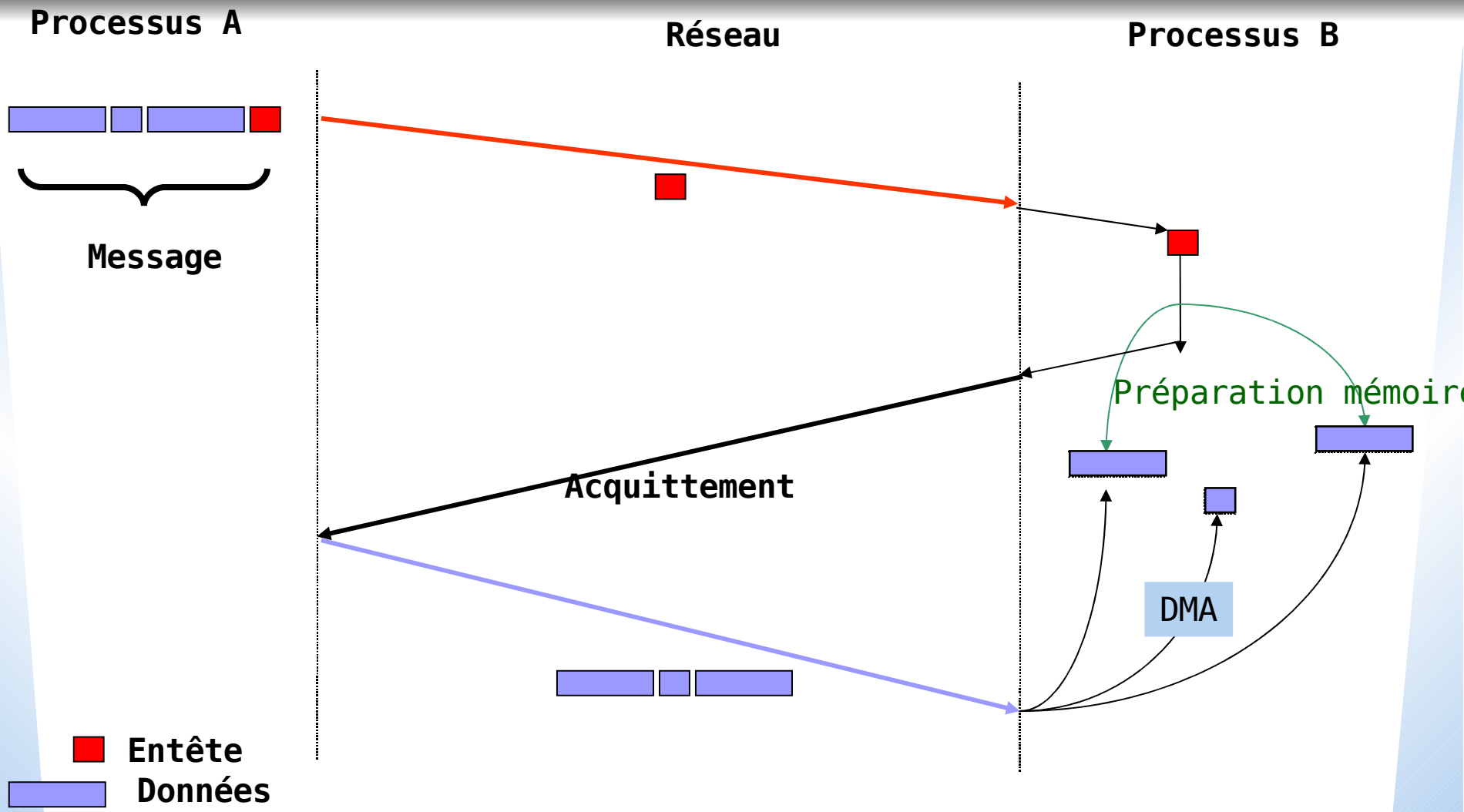
- Transfert en mode PIO
  - Zéro-copie trivial
- Transfert en mode DMA
  - Données non-contigues en mémoire physique
  - Ajout d'entêtes pour le protocole

  → Nécessité d'utiliser des DMA « chaînés »
    - Limitations sur le nombre de segments non contigus

Transmissions zéro-copie en réception

- Une carte réseau ne peut pas « geler » la réception d'un message sur le cable
  - Si le récepteur a posté l'opération « *recv* » avant l'arrivée du message
    - Zéro-copie OK si la carte peut filtrer les messages entrants
    - Sinon, zéro-copie possible avec les messages de taille bornée en utilisant une approche optimiste
  - Si le récepteur n'est pas encore prêt
    - Il faut mettre en place un rendez-vous pour les gros messages
    - Les petits messages peuvent être stockés dans un tampon interne

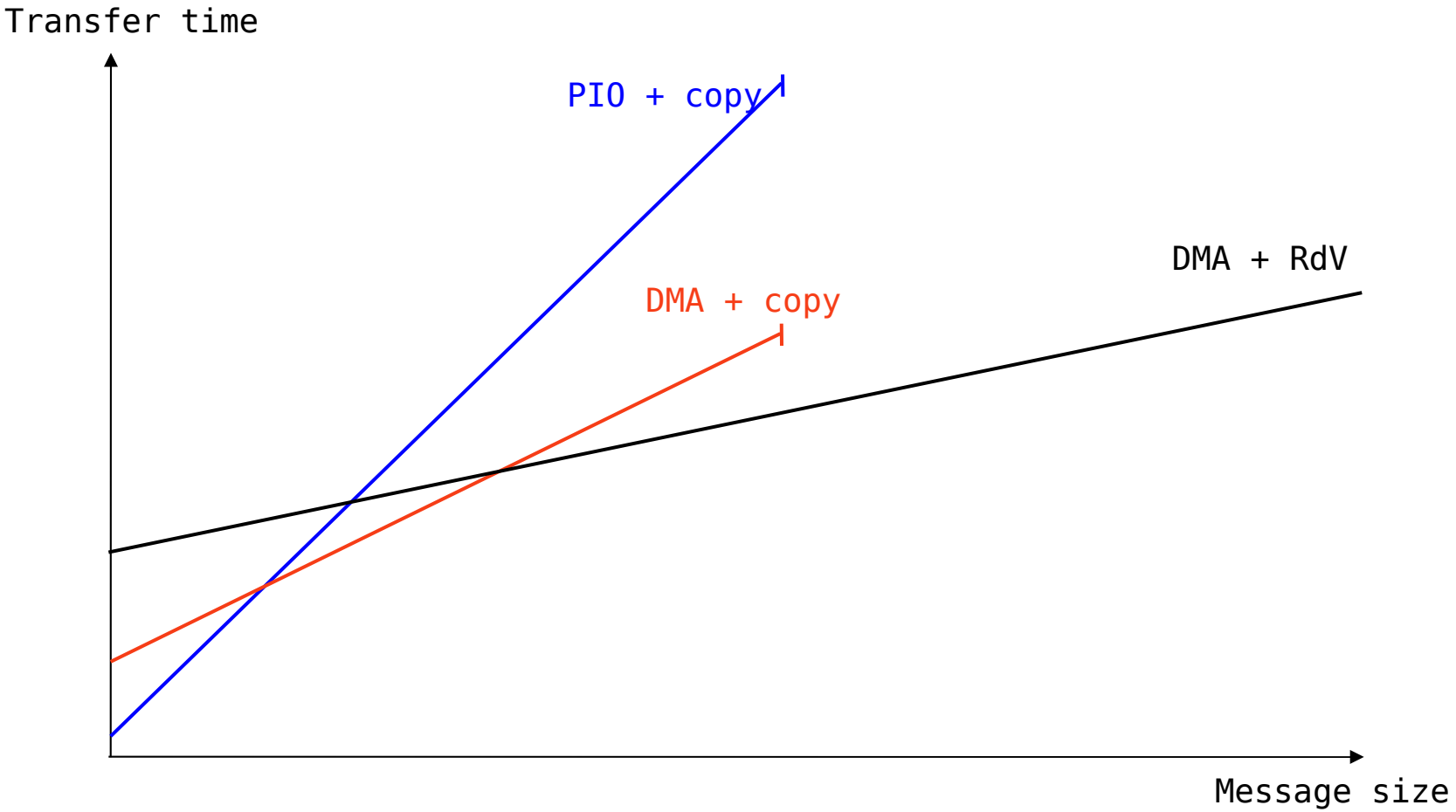
Utilisation d'un rendez-vous



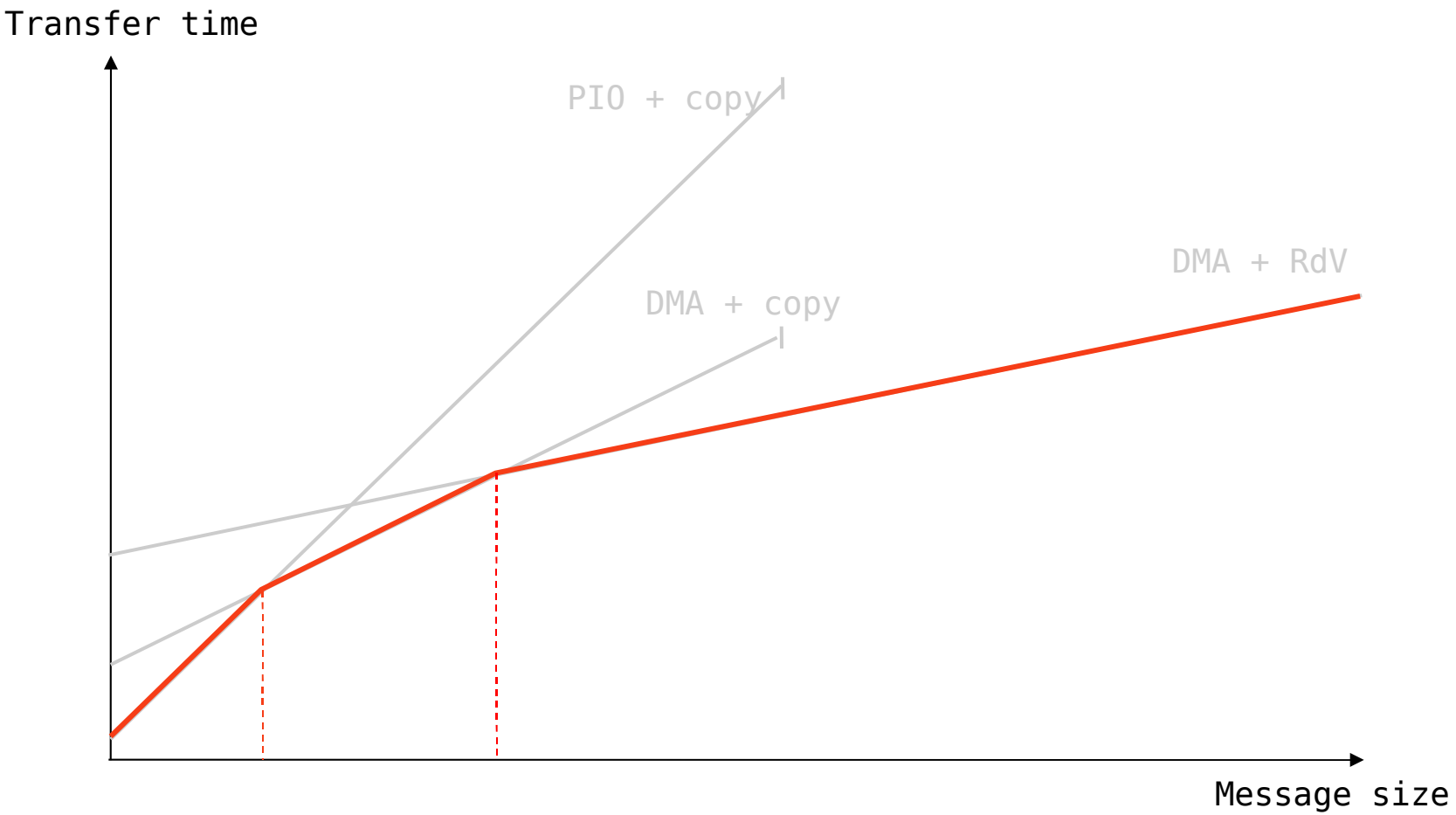
Considérations supplémentaires

- The receiving side plays an important role
  - Flow-control is mandatory
  - Zero-copy transfers
    - The sender has to ensure that the receiver is ready
    - A Rendezvous (REQ+ACK) can be used
- Communications in user-space introduce some difficulties
  - Direct access to the NIC
    - Most technologies impose “pinning” memory pages
- Network drivers have limitations

Choix du protocole de transmission

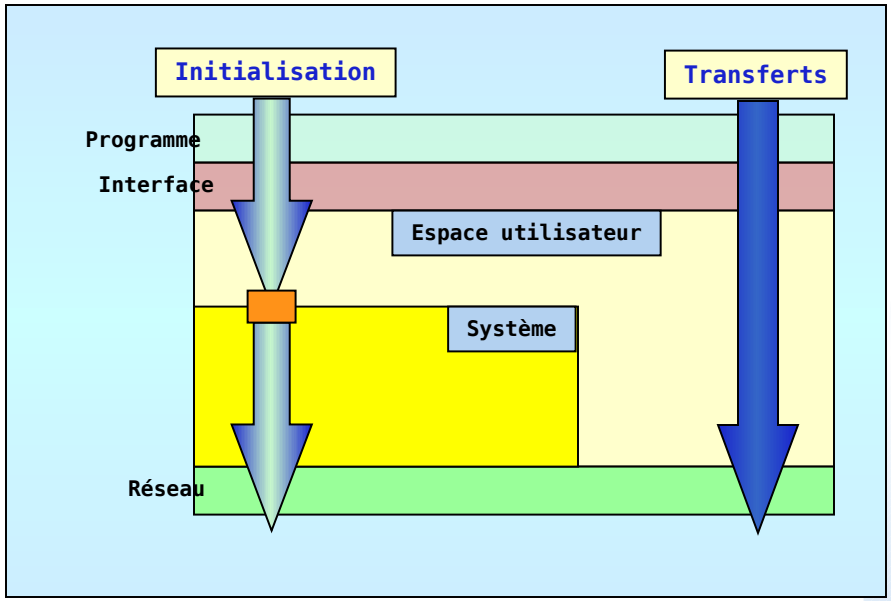


Choix du protocole de transmission



Court-circuitage du système (*OS bypass*)

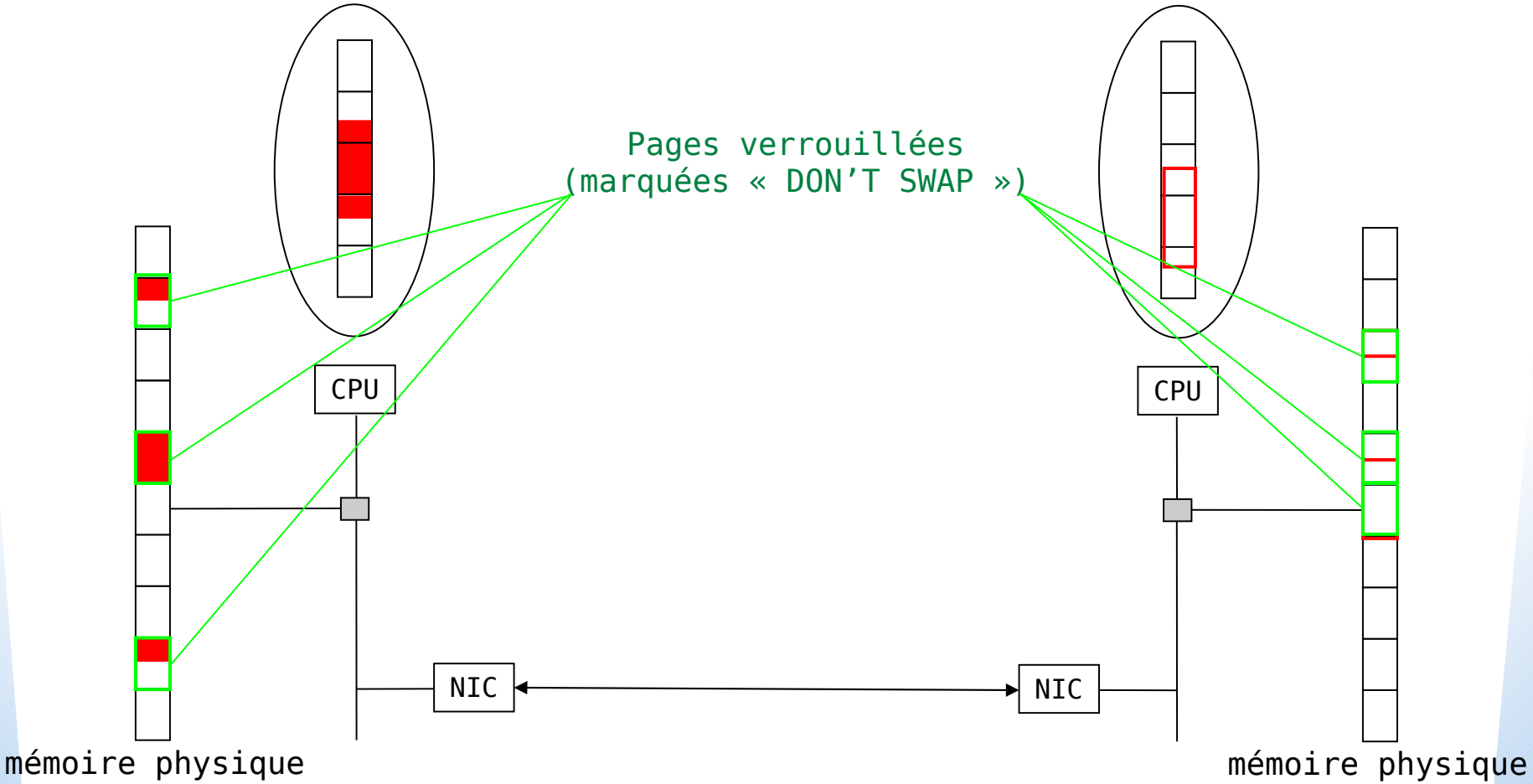
- Initialisation
  - Appels système classiques
  - Uniquement en début de session
- Transferts
  - Directs depuis l'espace utilisateur
  - Pas d'appels systèmes
  - « moins » d'interruptions
- Euh… Et la sécurité ?



OS-bypass + zéro-copie

- Problème
  - La technique du zéro-copie utilise le DMA qui manipule des adresses physiques
  - La correspondance (addr_virt, addr_phys) n'est connue que
    - Par le processeur (MMU)
    - Par le système d'exploitation (table des pages)
  - Il faut que
    - La bibliothèque connaisse ces correspondances
    - Les correspondances ne soient pas modifiées pendant la transmission
      - Ex: Swap du système d'exploitation, copy-on-write, etc.
  - Impossible à garantir en espace utilisateur !

OS-bypass + zéro-copie



OS-bypass + zéro-copie

- Première solution utilisée
  - « Enregistrement » des pages dans le noyau pour empêcher le swap
  - Maintien d'un cache de correspondances (addr_virt, addr_phys)
    - En espace utilisateur ou sur la carte réseau
  - Rattrapage des appels systèmes qui pourraient occasionner des modifications de l'espace d'adressage
- Seconde solution
  - Maintien d'un cache de correspondances sur la carte réseau
  - Patch du système d'exploitation pour « avertir » la carte en cas de changement
  - Approche choisie par MX/Myrinet ou Elan/Quadrics

Conséquences directes

- La mesure de la latence peut varier selon les régions mémoire impliquées
  - Certaines pages sont « enregistrées » auprès de la carte réseau
- Le cas idéal concerne les échanges de type ping-pong
  - Les mêmes pages sont réutilisées des centaines de fois
- Le cas le pire concerne les applications manipulant de nombreuses zones de données différentes...

Quelques interfaces de bas niveau

BIP/Myrinet, MX/Myrinet, SISCI, VIA

BIP/Myrinet

- Basic Interface for Parallelism
  - L. Prylli et B. Tourancheau
- Dédiée aux réseaux Myrinet
- Caractéristiques
  - Transmission asynchrone
  - Pas de détection d’erreur
  - Pas de contrôle de flux
    - Petits messages recopiés dans un tampon fixe en réception
    - Gros messages perdus si récepteur pas prêt !

MX/Myrinet

- Myrinet eXpress
  - Pilote officiel diffusé par Myricom
- Interface très simple destinée à faciliter l'implémentation de MPI
  - Contrôle de flux
  - Transmission fiables
  - Messages non contigus
  - Multiplexage

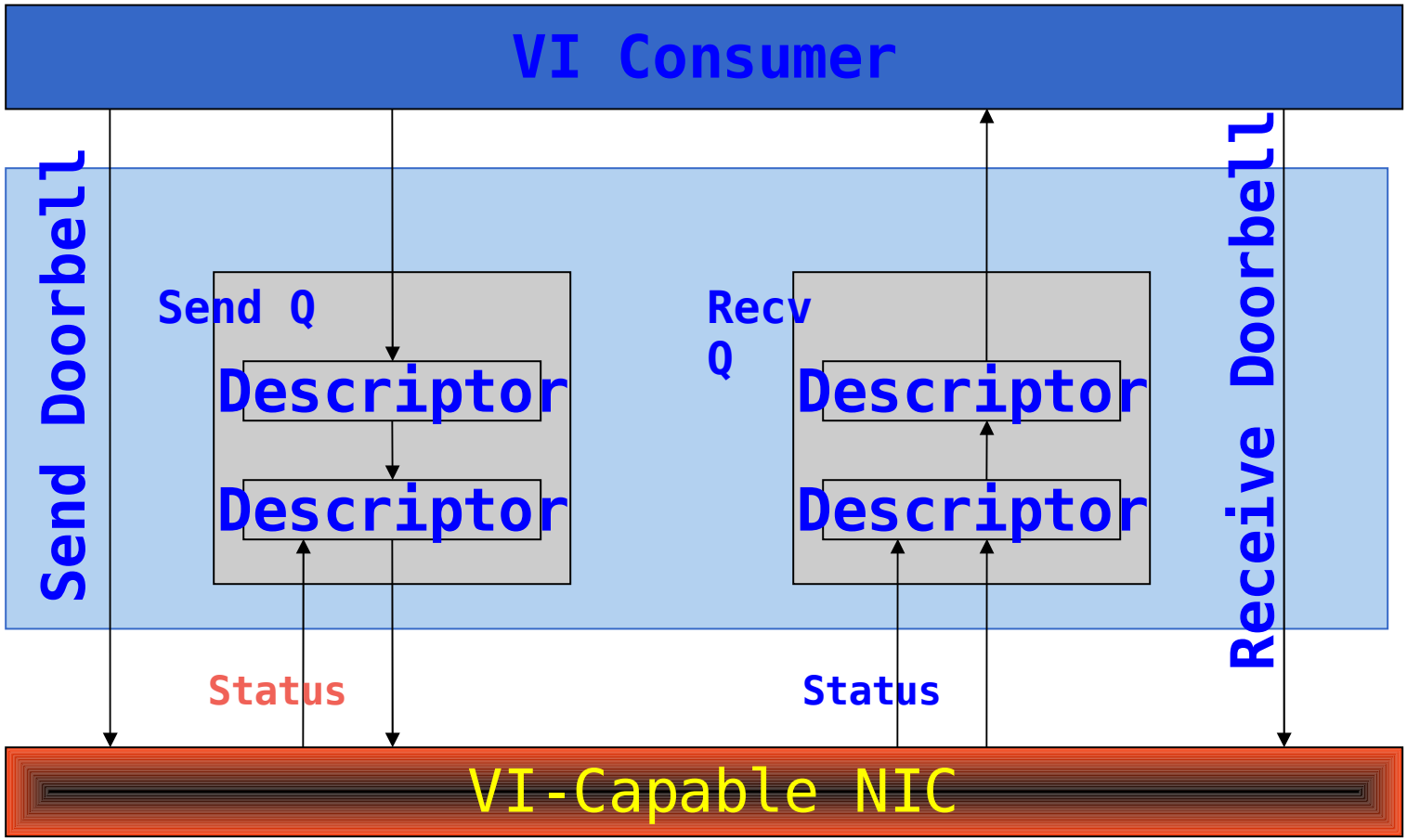
SiSCI/SCI

- Pilote des cartes SCI
- Modèle de programmation
  - Accès mémoire à distance
    - Explicites : RDMA
    - Implicites : projections mémoire
- Performances
  - Nécessité d’utiliser des opérations explicites
    - « flush » mémoire
    - SCI_memcpy
    - RDMA

VIA

- Virtual Interface Architecture
- Tentative de standardisation
  - Beaucoup d'industriels impliqués
    - Microsoft, Intel, Compaq, etc.
- Caractéristiques
  - Principe d'interfaces virtuelles
    - Files de descripteurs (émission et réception)
  - Enregistrement mémoire explicite
  - Lectures/Ecritures distantes
    - RDMA

A Virtual Interface



Bilan

- Interfaces très spécifiques, dédiées à certaines technologies (excepté VIA)
  - Paradigmes de programmation différents
  - Portabilité quasi-nulle
- Il n’est pas raisonnable de programmer une application scientifique directement au-dessus de telles interfaces !

L’approche de haut niveau MPI

Message Passing Interface

L’interface standard MPI

- Message Passing Interface
  - MPI-Forum v1.0 1994 v2.0 1997
- Caractéristiques
  - Interface (implémentation)
  - Diverses implémentations
    - MPICH
    - LAM-MPI
    - OpenMPI
    - Et bien sûr tous les MPI « constructeurs »
  - Version 2.0 encore peu implémentée

Échange de messages avec MPI

- Différentes variantes
  - MPI_Send (standard)
    - L'appel se termine lorsque l'on peut réutiliser les données sans risque
  - MPI_Bsend (buffered)
    - Le message est copié localement s'il ne peut pas être émis immédiatement
  - MPI_Rsend (ready)
    - L'émetteur « promet » que le récepteur est prêt
  - MPI_Ssend (synchronous)
    - L'appel se termine lorsque la réception a commencé (~propriété d'une barrière de synchronisation)

Primitives non bloquantes

- MPI_Isend / MPI_Irecv (immediate)

```
MPI_request r;

MPI_Isend(…, data, len, …, &r)

// Calcul qui ne modifie pas data

MPI_wait(&r, …);
```

- Il faut utiliser ces primitives le plus souvent possible

À propos des implémentations de MPI

- MPI est disponible sur (presque) tous les réseaux et protocoles existant !
  - Ethernet, Myrinet, SCI, Quadrics, Infiniband, IP, shared memory, …
- Les implémentations de MPI sont aujourd'hui très performantes
  - Latence faible (difficile), débit élevé (pas difficile)
  - Les constructeurs fournissent des versions optimisées (IBM, SGI)
  - Les implémentations s'appuient parfois directement sur des interfaces de très bas niveau
    - MPICH/Myrinet, MPICH/Quadrics
- MAIS ces « bonnes performances » sont souvent observées sur des programmes de ping-pong…

À propos des implémentations de MPI

- Les implémentations souffrent souvent de défauts cachés qui se révèlent lorsque l'on développe des applications complexes
- Exemple : types « indexés » MPI
  - Normalement utilisés pour transférer des messages non-contigus
  - À cause des limitations des interfaces de bas-niveau, leur prise en charge est souvent peu efficace
    - Transferts en plusieurs étapes
    - Recopies
- Ce n'est malheureusement pas tout…

Points délicats

Recouvrement des calculs

Recouvrement des calculs grâce aux opérations non-bloquantes

- Problème : parfois, ça ne recouvre rien du tout !
- Exemple de programme test
  - Circulation d'un message sur un anneau de processus

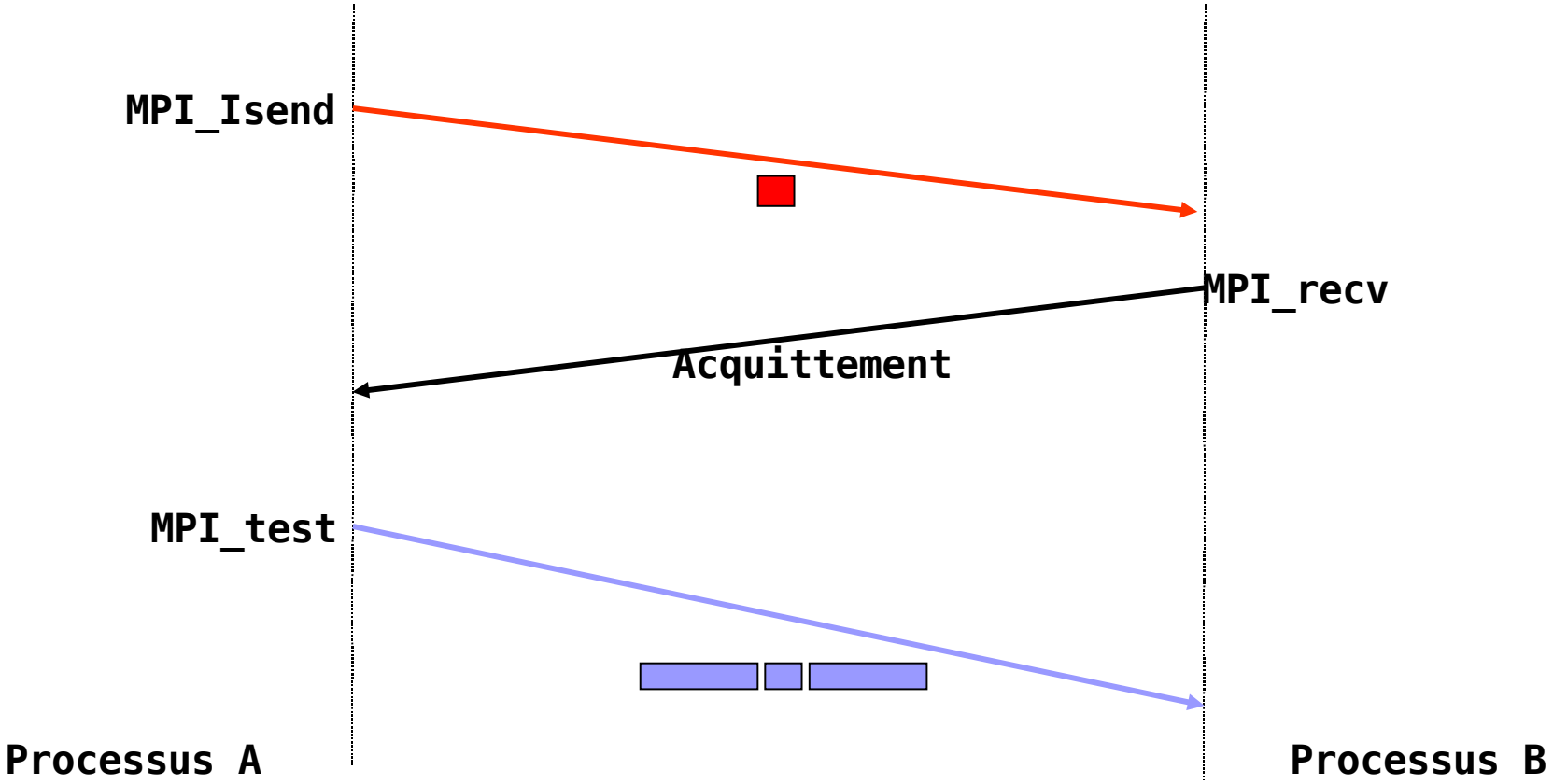
```
// chaque processus exécute
MPI_recv(from_prev, data, len, …);
MPI_Isend(to_next, data, len, …, &r)

sleep(1);

MPI_wait(&r, …);
```

Transfert lors du MPI_Isend

- Dans leur course à la latence, les bibliothèque de bas-niveau sacrifient parfois l'occupation du processeur
  - En fonction de la taille du message
    - Transactions PIO pour les petits
    - Copies pipelinées avec DMA pour les moyens
    - DMA zéro-copie pour les gros
  - Exemple : la limite moyen/gros est fixé à 32Ko en MX
    - L'émission des messages de 0 à 32Ko ne peut pas recouvrir des calculs

Transfert lors du MPI_wait

- Parvient-on vraiment à assurer du recouvrement ?



Transfert lors du MPI_wait

- Problème
  - Lorsque l'acquittement du rendez-vous arrive sur la carte, le processus fait autre chose…
- Solutions
  - Utiliser des threads au sein de l'implémentation de MPI
    - MPICH/Madeleine
  - Embarquer une partie du protocole sur la carte réseau
    - MPICH/GM
  - Utiliser les lectures mémoire à distance !

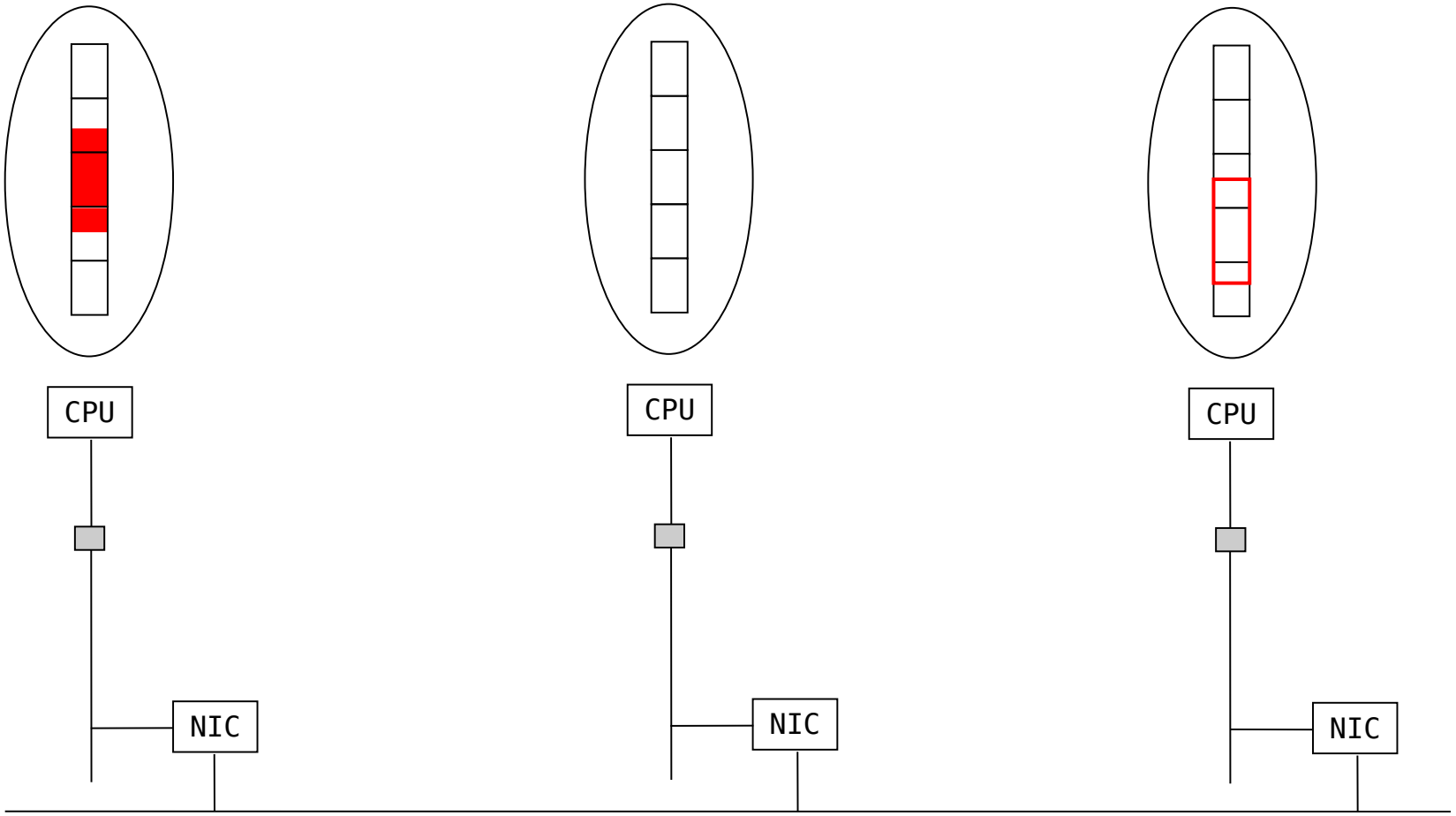
Points délicats

Progression des communications

Progression des communications

- La progression autonome des communications est importante dans certaines situations
  - Émissions en tâche de fond
    - MPI_Isend
  - Réception en tâche de fond
    - MPI_PUT, MPI_GET
  - Communications collectives
    - Forwarding de messages sur les nœuds de l’arbre

Progression des communications



Progression des communications

- Seules les implémentations sophistiquées y parviennent !
  - Threads internes à la bibliothèque
  - Spécialisation du *firmware* de la carte réseau
    - Myrinet, Quadrics

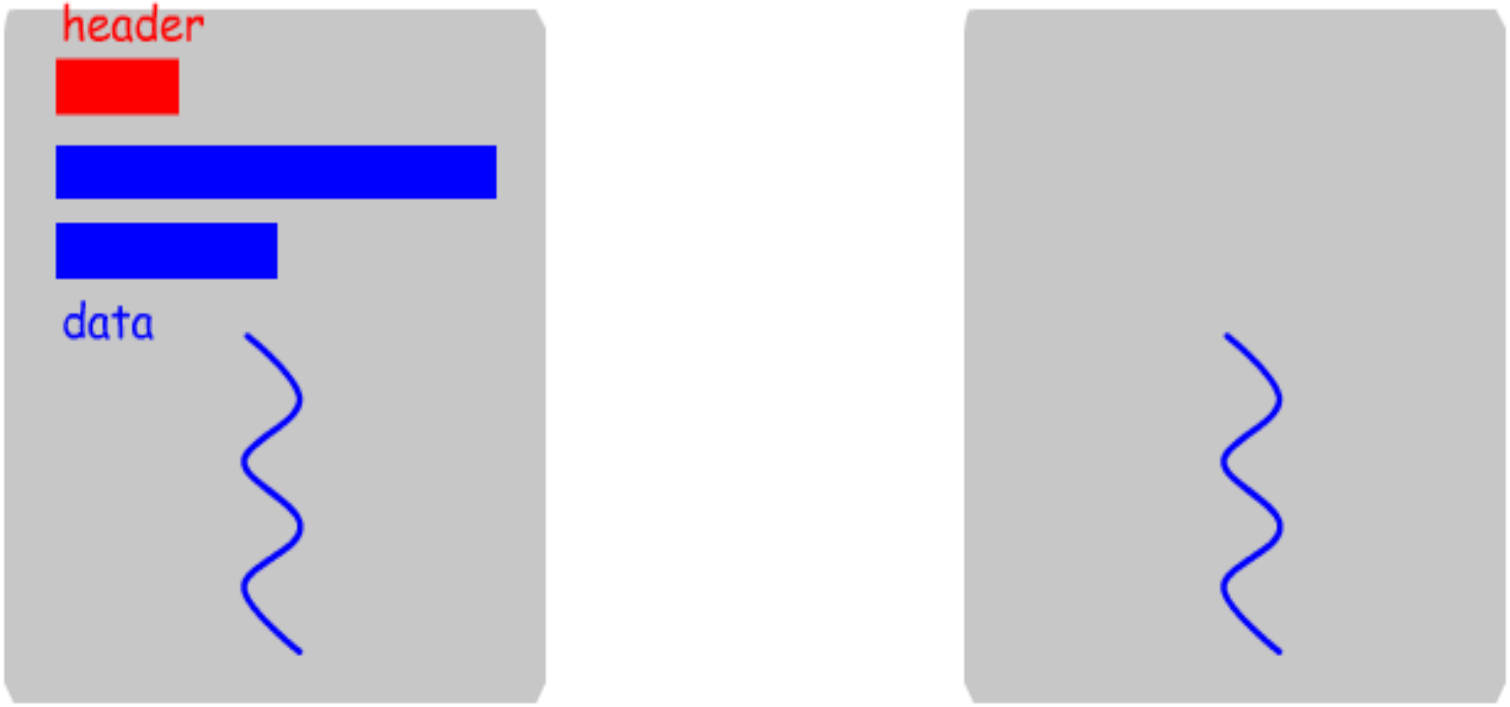
Points délicats

Messages complexes
(i.e. avec dépendances entre les segments)

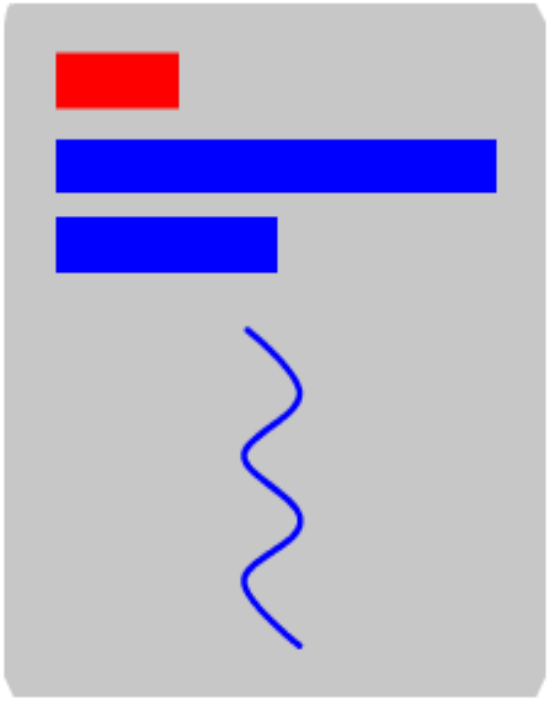
Limites des interfaces de type send/recv

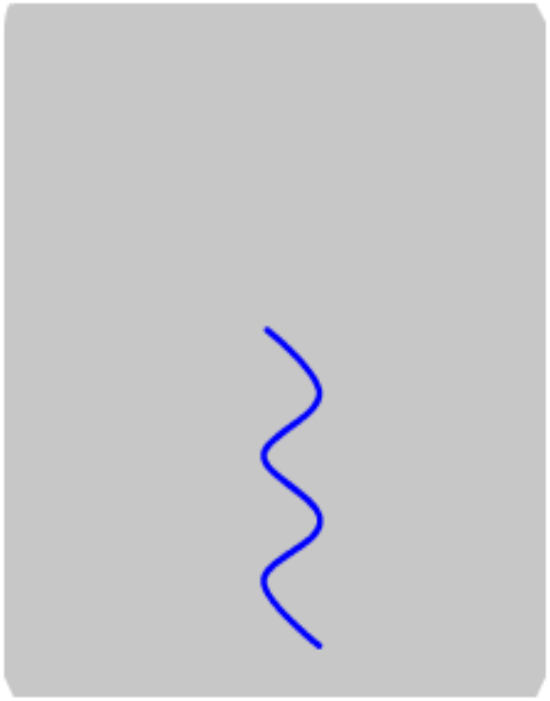
- Les interfaces orientées « send/recv » sont bien adaptées aux applications dans lesquelles
  - Les schémas de communication sont réguliers
    - Jamais de message « inattendu »
  - La structure des messages est simple
    - Pas de dépendances internes
- De nombreuses applications ont toutefois des exigences plus contraignantes
  - Application de type clients/serveur
  - Systèmes à mémoire virtuellement partagée
  - Environnements de programmation distribués multithreads
  - En bref: les applications qui utilisent plutôt des interactions de type « appels de procédures à distance »

Réalisation d’un appel de procédure à distance

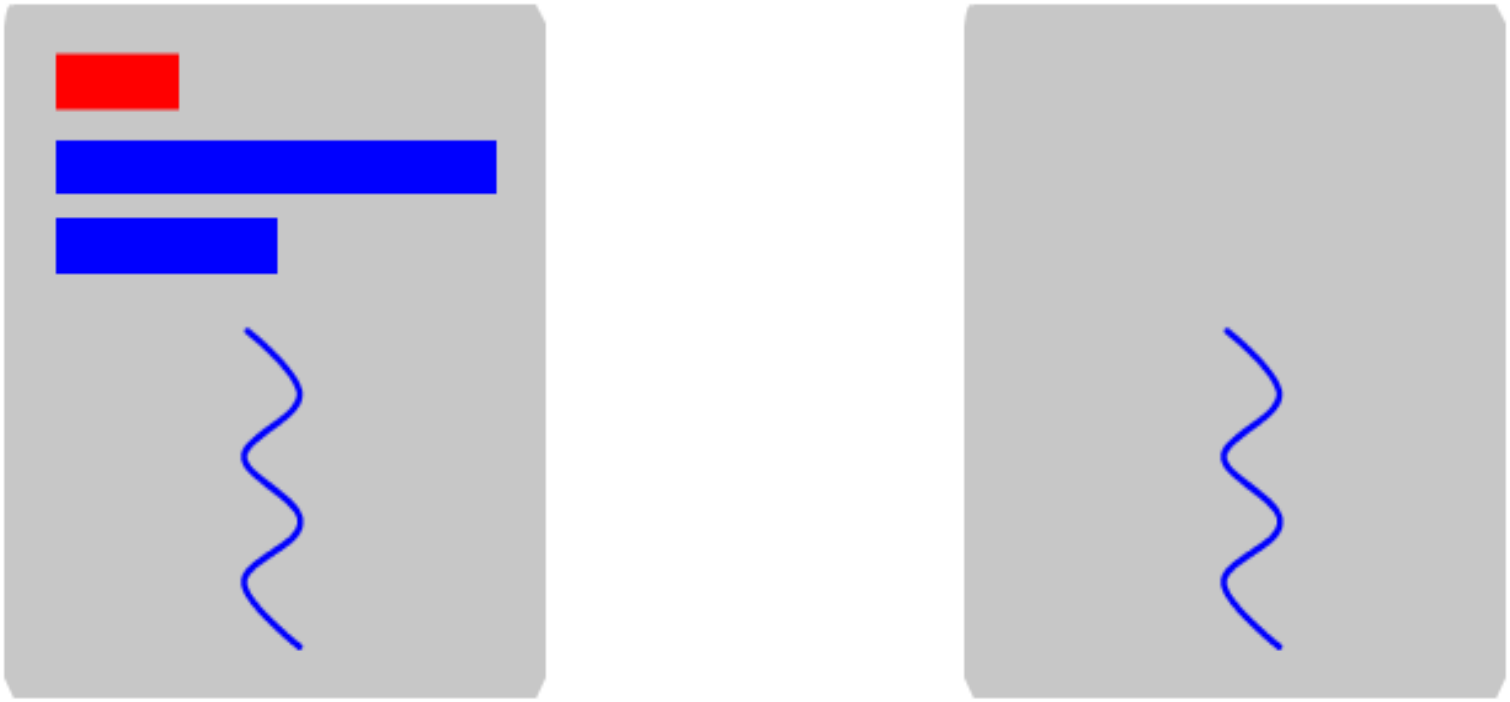


Concrètement, devrait-on procéder comme ceci ?





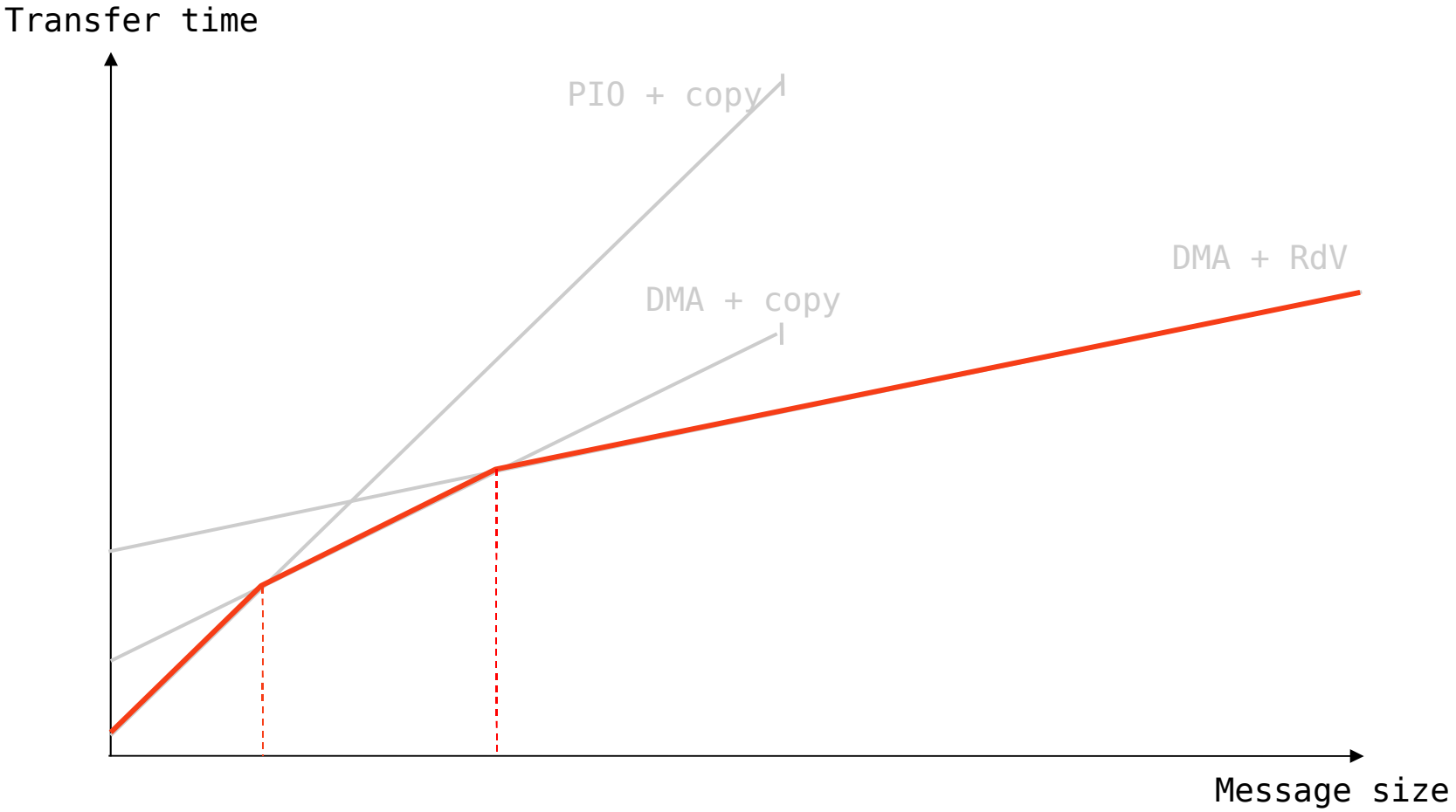
Ou plutôt comme cela ?



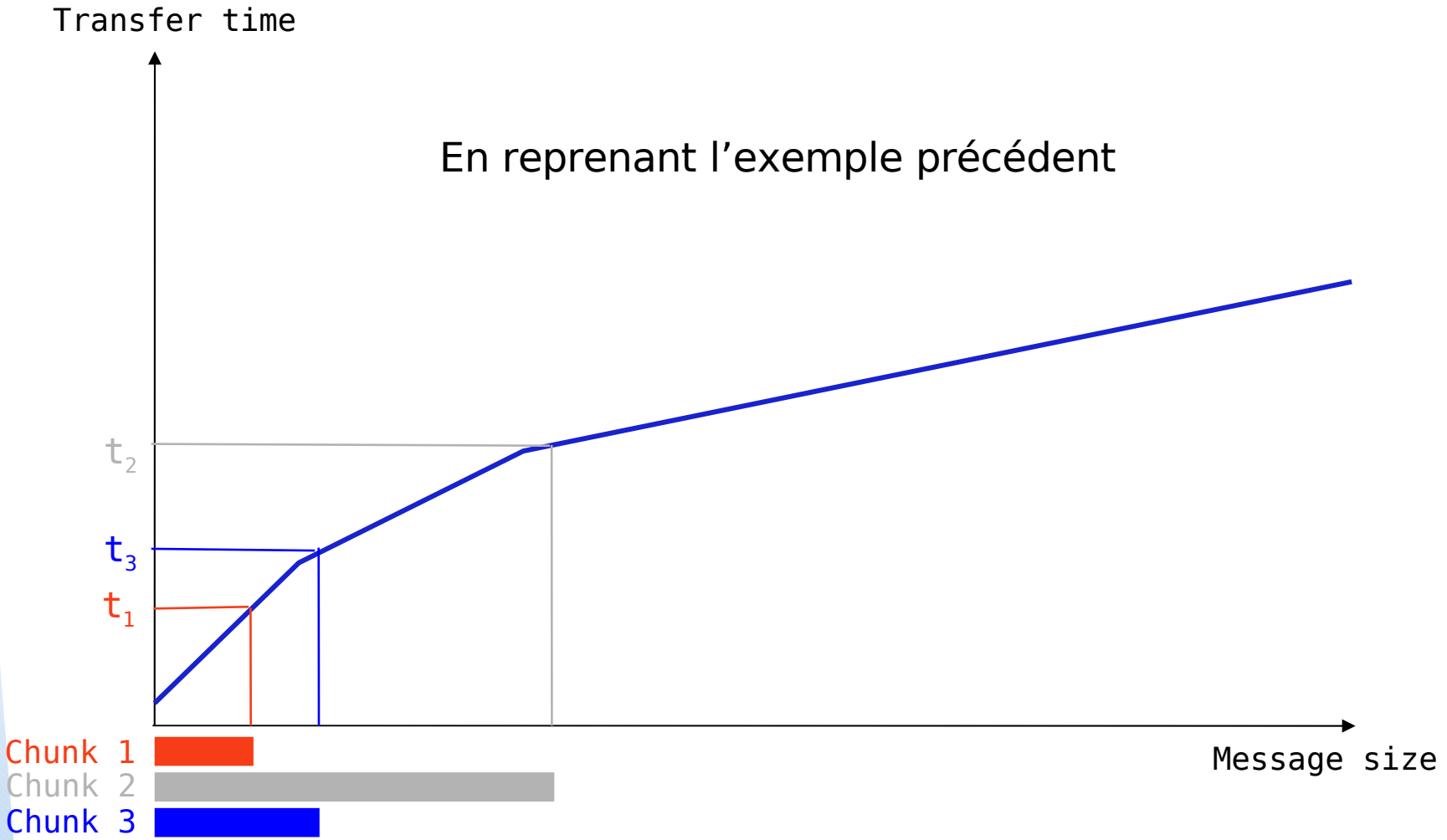
Ou encore d'une autre façon ?

- Cela dépend du réseau sous-jacent!
  - Les pilotes réseau offrent des caractéristiques très diverses
    - Latence
    - Performance des transferts PIO & DMA
    - Fonctions *Gather/Scatter*
    - Fonctions *Remote DMA*
    - Etc.
- En fait, cela dépend également des caractéristiques des nœuds de calcul
  - Performance des copies mémoire
  - Performance du bus d'E/S

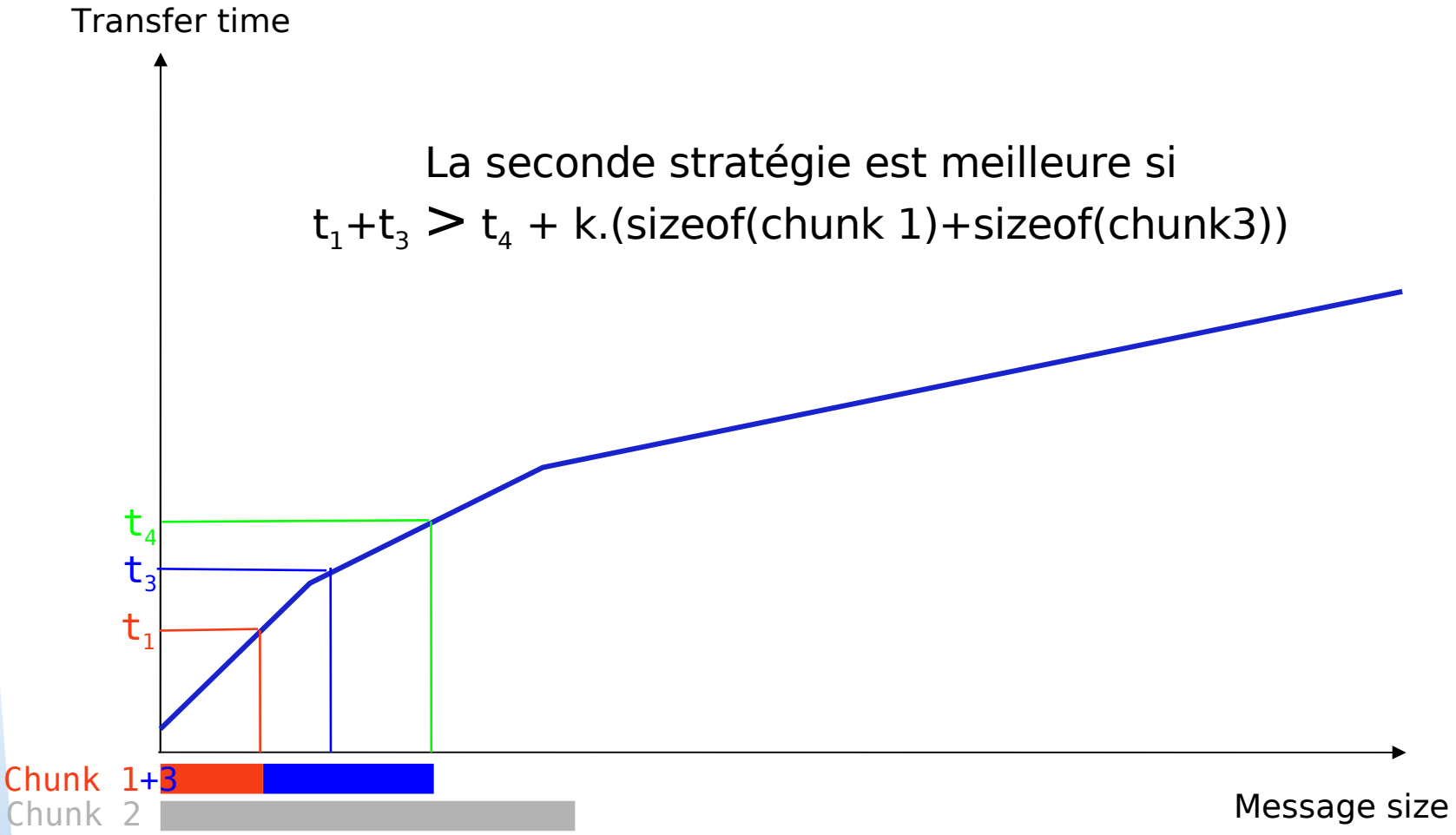
À propos de la stratégie optimale



À propos de la stratégie optimale



À propos de la stratégie optimale



Comment faire avec MPI ?

- Le choix du protocole est effectué par l'application
  - Implique de connaître les caractéristiques du réseau sous-jacent
    - Donc rend l'application non portable !
- D'où vient le problème ?
  - De l'interface de MPI qui n'est pas assez expressive...

Vers des interfaces plus puissantes

Meilleure coopération entre l’application et le support exécutif

Exemple : l'interface Madeleine

- Objectif
  - Fournir l'expressivité nécessaire pour échanger des messages complexes de manière portable
- Caractéristiques
  - Construction incrémentale des messages avec spécifications de dépendances internes
    - L'application spécifie des contraintes (sémantiques) et des dépendances à respecter
    - Madeleine sélectionne automatiquement la stratégie d'optimisation la plus appropriée
  - Communications multi-protocoles
    - Plusieurs cartes réseaux (différentes) utilisées simultanément
  - Bibliothèque *thread-aware*

Construction des messages

- Sender:

begin_send(dest)

pack(&len, sizeof(int))
pack(data, len)

end_send()

- Receiver:

begin_recv()

unpack(&len, sizeof(int))
data = malloc(len)
unpack(data, len)

end_recv()

Construction des messages

- Sender:

begin_send(dest)

pack(&len, sizeof(int),
r_express)
pack(data, len, **r_cheaper**)

end_send()

- Receiver:

begin_recv()

unpack(&len, sizeof(int),
r_express)
data = malloc(len)
unpack(data, len,
r_cheaper)

end_recv()

Construction des messages

- Sender:

```
begin_send(dest)

pack(&len, sizeof(int),
    r_express)

pack(data, len, r_cheaper)

pack(data2, len, r_cheaper)

end_send()
```

- Receiver:

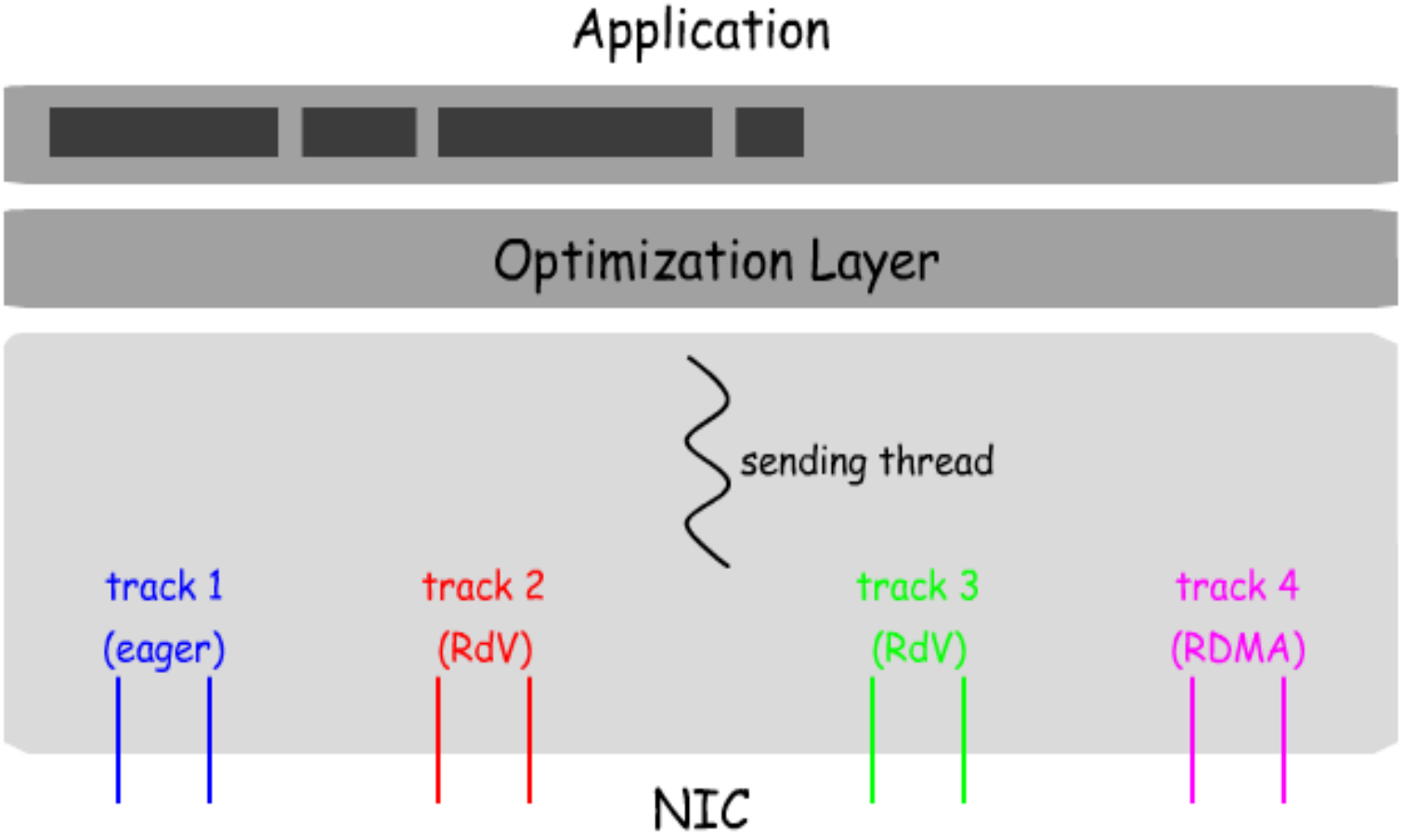
```
begin_recv()

unpack(&len, sizeof(int),
    r_express)
data = malloc(len)
unpack(data, len, r_cheaper)
data2 = malloc(len)
unpack(data2, len,
    r_cheaper)

end_recv()
```

Comment implémenter les optimisations ?

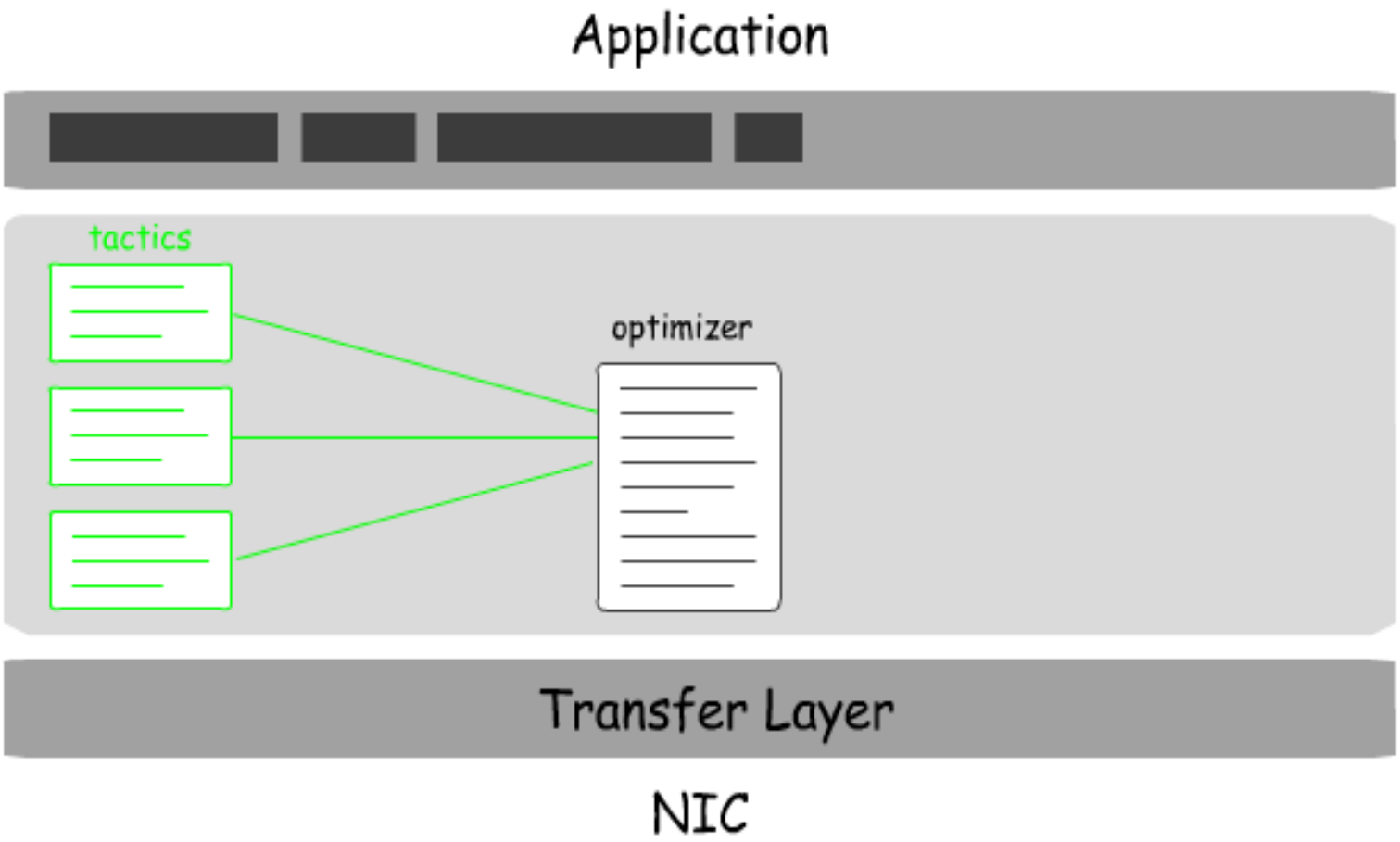
- Certaines optimisations sont hors d'atteinte
  - Dans la version stable actuelle, la stratégie est déterministe
    - Émetteur et récepteur prennent toujours les mêmes décisions
    - Heuristique utilisant uniquement l'historique + paramètres des (un)pack
    - Avantage: les données sont transmises de manière brute
  - Le futur « proche » de l'application serait pourtant utile !
- Quelles évolutions pour des optimisations plus agressives ?
  - Abandonner le déterminisme des heuristiques
    - Autoriser l'envoi de messages dans le désordre (au prix d'un surcoût raisonnable pour les petits messages)

      -> optimisations opportunistes/optimistes possibles
  - Permettre l'enrichissement dynamique des stratégies d'optimisation (plug-ins)

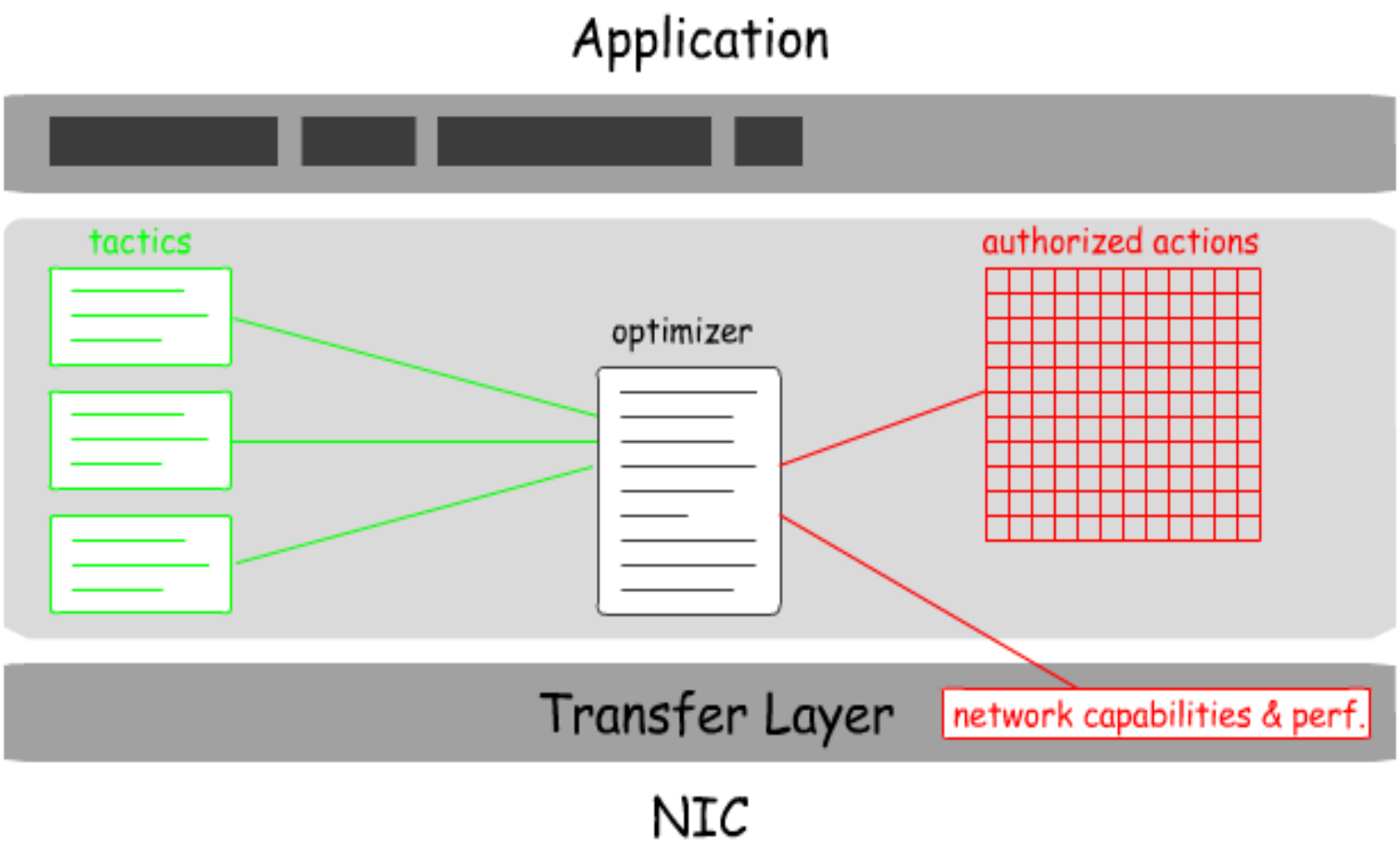
Fonctionnement guidé par la carte réseau



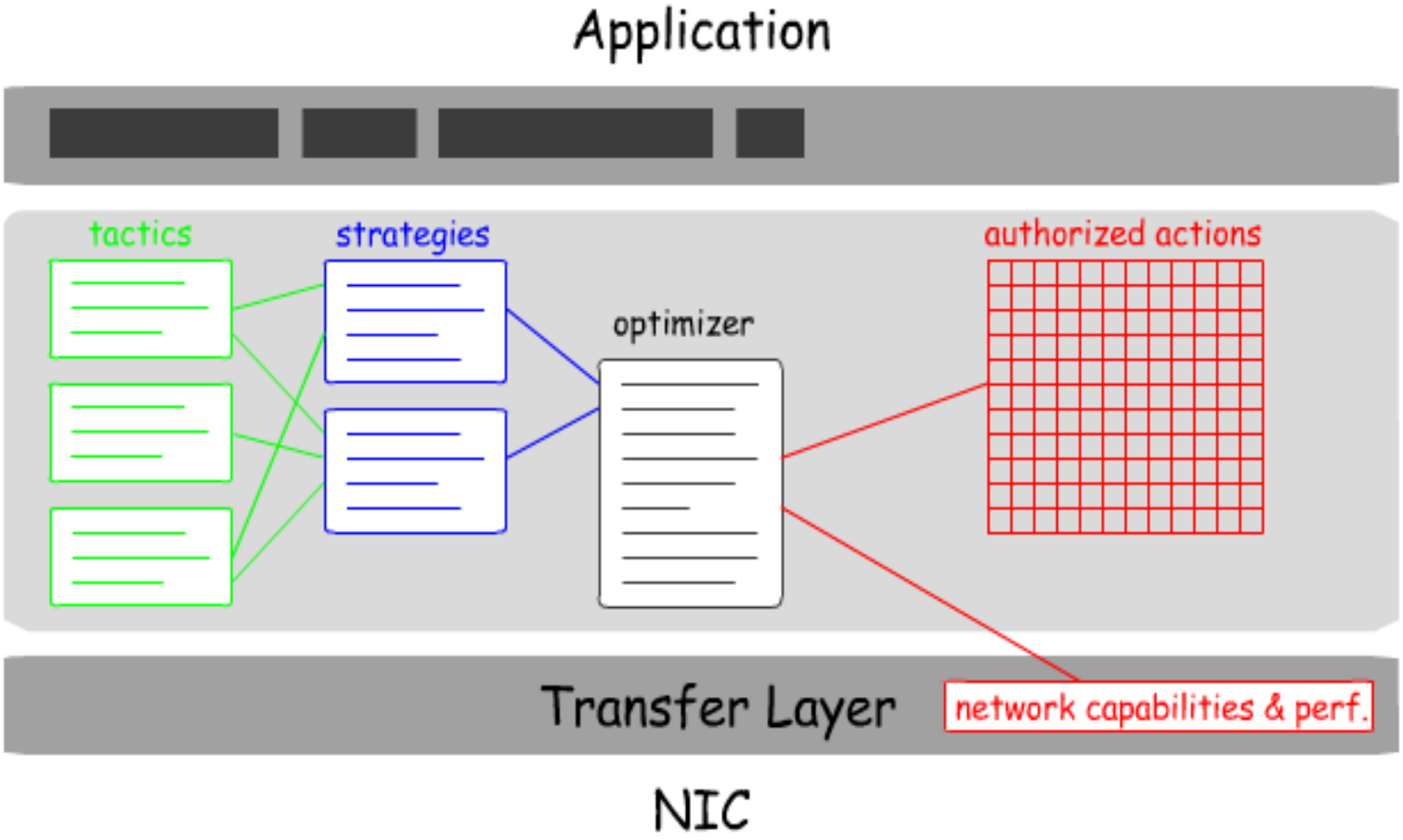
Optimisations « Just-in-Time »



Optimisations « Just-in-Time »



Optimisations « Just-in-Time »



Qu’apporte une telle interface ?

- L’application reste portable
- La bibliothèque peut utiliser un vaste éventail de stratégies pour minimiser le temps de transfert
  - Agrégation des données (petits segments)
  - « Piggybacking » des messages de contrôle
  - Utiliser des mécanismes de gather/scatter si possible
  - Maximiser le remplissage des tampons internes si nécessaire
- Les optimisations peuvent être opportunistes
  - Optimisations globales à tous les communicateurs
  - Équilibrage de charge sur différents liens (multi-rail)
- Problème : l’interface n’est pas standard…
  - MPICH/Madeleine

Quels sont les défis ?

- Optimiser tout en restant équitable…
  - Ex: ne pas favoriser une catégorie de messages
- Définir des stratégies optimales semble très difficile
  - Dans certains cas particuliers, la stratégie optimale peut-être calculée en temps linéaire
    - Exemple: O(n) lorsque
      - Une seule tactique: “agrégation par copie”
      - Pas de limite sur la taille des tampons utilisés par les copies
  - Dans le cas général, on ne sait pas encore
    - C’est pourquoi il est important de pouvoir expérimenter plusieurs stratégies
- Trouver une interface qui puisse encourager les gens à essayer autre chose que MPI !

Et plus généralement…

- Gestion des configurations multi-rails
  - Aiguillage suivant les caractéristiques des réseaux
  - Équilibrage de charge
- Interconnexions de grappes
  - « forwarding » des messages entre cartes
- Nœuds dotés de nombreux « cœurs »
  - Vous avez dit *off-loading* ou *on-loading* ?
- Gestion des configurations à grande échelle
  - Grilles

Conclusion

La réalisation de communications efficaces reste difficile !

Éléments à retenir

- L’exploitation des réseaux rapides requiert l’utilisation de nombreuses techniques d’optimisation
- Ces optimisations, souvent mono-critère, sont enfouies à l’intérieur des bibliothèques de communication
- Il faut parfois utiliser des interfaces de bas-niveau pour conserver un contrôle total sur les communications

Perspectives

- Renforcement de la coopération entre les langages et les bibliothèques de communication
- Meilleure prise en charge des configurations hiérarchiques
- Répartion des traitements réseau sur les grosses architectures multiprocesseurs