Systèmes et traitements répartis sur grille

Serge G. Petiton

28 septembre 2007

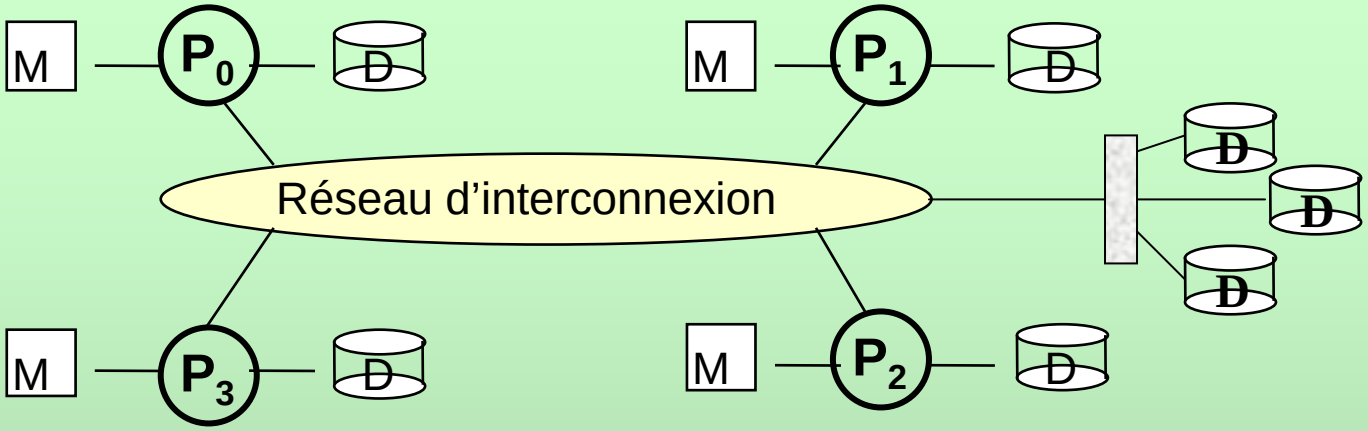
(avec des slides de Bernard Toursel)

Master 2, 2007-2008

- II -Architectures et calculs parallèles, data parallèles et vectoriels
- Introduction aux systèmes répartis -

1. Introduction
2. Paradigmes
3. Problématiques principales

C’est quoi un traitement distribué ?



- Traitement séquentiel exécuté dans un ordinateur
- Traitement distribué = traitement décomposé en plusieurs tâches coopérantes, réparties sur un ensemble d’ordinateurs interconnectés au travers d’un réseau de communication

Pourquoi des traitements distribués ?

- Développement des réseaux et des communications
- Explosion des masses de données (+ 30 % par an, croisant)
- Besoins en calculs de haute performance

(simulation, optimisation, génétique, …)

- Travail coopératif
- Données ou traitements pré-distribués
- Ressources disponibles non utilisées

Pourquoi des traitements distribués ? (2)

- Evolution des performances

- Processeurs : doublement tous les 18 mois (loi de Moore) [≡ + 33 % par an]

(mais limite prévisible :… consommantion… rayons cosmiques..)

- Mémoire disque : latence : +10% par an débit : +20% par an

- Réseau : latence : +20% par an débit : + 45% par an

(source « High performance cluster computing », Rajkumar Buyya)

Un peu de vocabulaire

- Traitements distribués / **répartis** (distributed computing)
  - Meta-computing, Cluster computing,

    traitements sur grappes de stations

    (COW=cluster of workstations, NOW=network of workstations)
  - Grid-computing, calculs sur grille, traitements à grande échelle, grappes de grappes, pair-à-pair (P2P, Peer-to-Peer)
  - Autonomous computing
  - Global-computing, Web-computing, internet computing
  - Ubiquitous computing, ambient computing
  - Desktop GRID

Paradigmes - 1 -

- **Contexte classique**
  - **Système unique**, ordonnancement des tâches
  - **Mémoire commune partagée** : point central unique, peut contenir un état global partagé, facile à contrôler (verrous, sémaphores…)
  - **Traitements séquentiels concurrents** : partage de données, conflits, synchronisation

Paradigmes -2-

- **Contexte distribué**
  - **Pas de système unique** : *systèmes hétérogènes sur des stations hétérogènes, pas de contrôle global des usagers, nécessité d'outils globaux : intergiciel (middleware),créer-gérer-utiliser une plate-forme d'exécution dynamique*
  - **Absence de mémoire commune, mais des mémoires distribuées**: *pas d'information globale, données dupliquées, problèmes de cohérence*
  - **Communication lentes** des informations au travers d'un réseau : *recouvrement temporel des communications par le calcul*
  - **Asynchronisme des traitements simultanés** : *indéterminisme, synchronisations complexes*
  - **Problèmes de distribution** : *transparence, répartition optimale*
  - **Risques accrus de pannes** : *pannes processeurs et réseau*

Problématiques - 1 -

- **Environnement de programmation, méthode de conception**
  - Intergiciels (middleware),
  - Outils de communication simples
  - Indépendance vis à vis de la plate-forme d'exécution
  - Transparence de la localisation
  - Nouveaux langages
- **Aspects algorithmiques**
  - Absence d'état global observable (validité des algorithmes, complexité)
  - Ordonnancement des évènements, gestion du temps
  - Masquage des communication (asynchronisme, recouvrement)

Problématiques - 2 -

- **Partage des informations**
  - Conflits, duplication, problèmes de cohérence
- **Distribution des tâches et des données**
  - Transparence, efficacité, équilibrage de charge (i.e. répartition des tâches tenant compte des performances et de la charge des stations, des évolutions des calculs)
- **Tolérance aux pannes**
  - Redondance des traitements et/ou des données, mécanisme de restauration, transactions distribuées
- **Sécurité**
  - Contrôle des accès : Authentification, certification, cryptage

Les plates-formes d’exécution

1. **Architectures faiblement versus fortement couplées**
2. **Réseaux**

2. Les plates-formes d'exécution

- **2 catégories d'architectures**

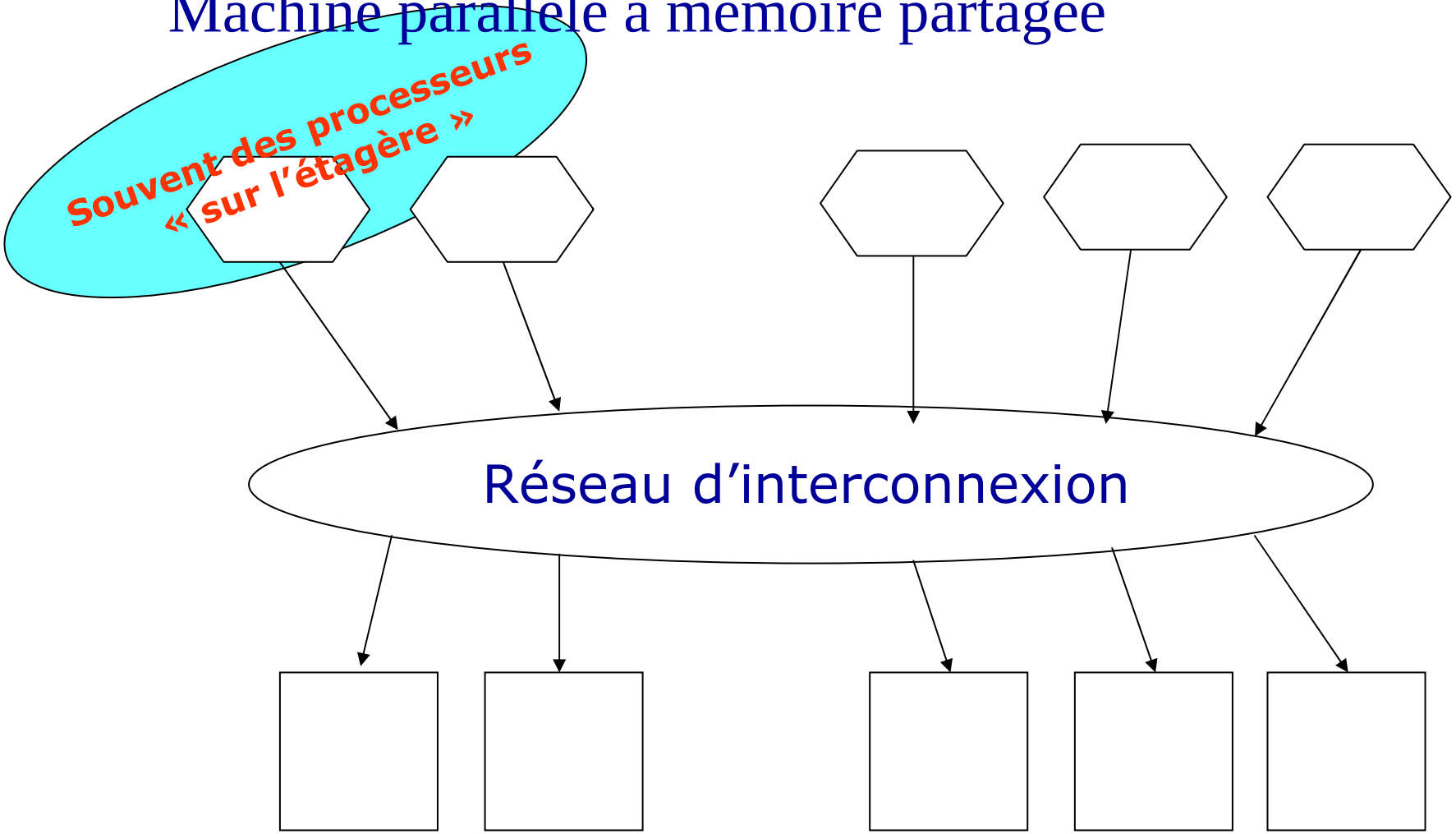
  - **Architectures fortement couplées** *(tightly coupled arch.)* : processeurs physiquement regroupés, en général autour d'une mémoire commune

  - **Architectures faiblement couplées** *(loosely coupled arch.)* : proc. géographiquement dispersés, avec des mémoires locales à chaque nœud et interconnectés par des réseaux

Les différents types de plate-formes

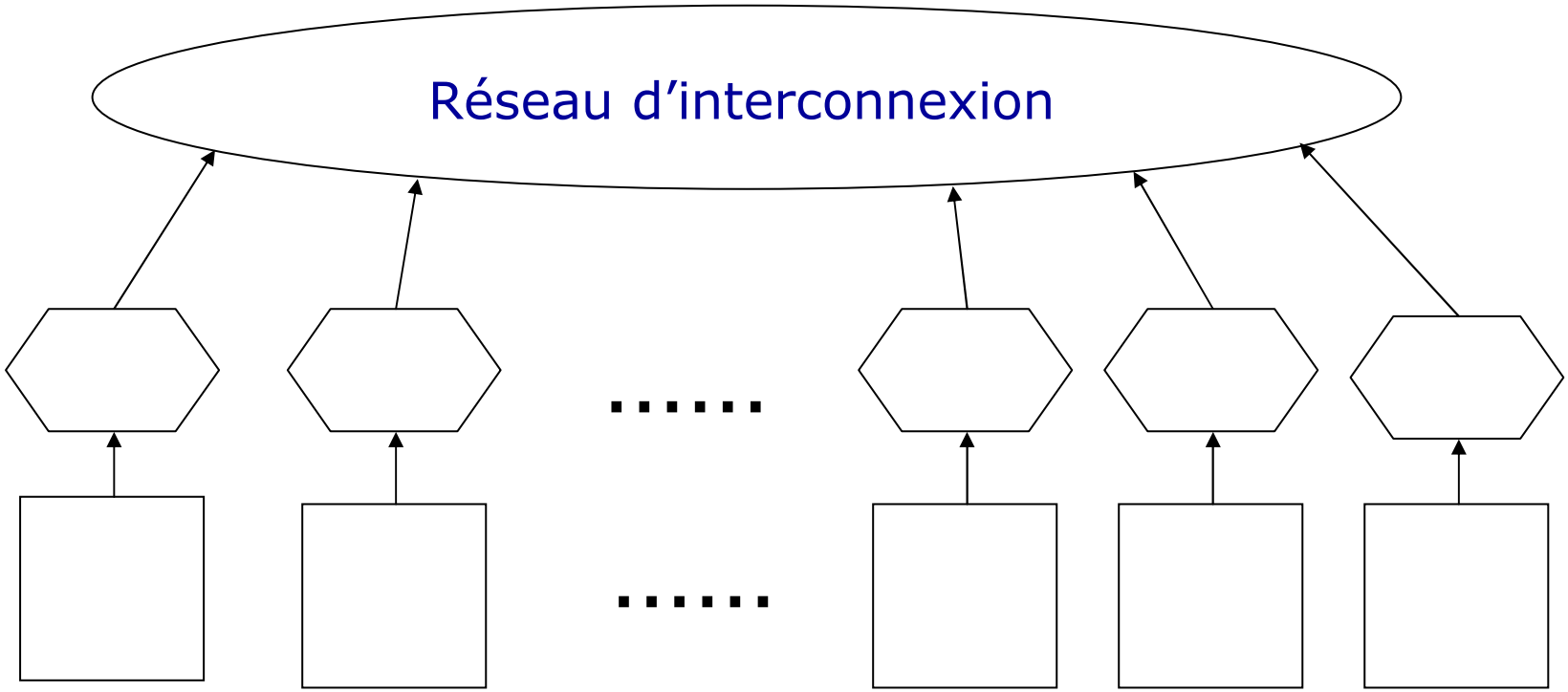
- **Architectures fortement couplées**
  - MPP : Massively Parallel Processor
  - SMP : Symmetric Multi-Processor
  - Architectures CC-NUMA : Cache Coherent Non Uniform Memory Access
- **Architectures faiblement couplées**
  - Clusters spécialisés : réseaux rapides et SSI (Single System Image)
  - Clusters : réseaux locaux de stations
  - Grid : systèmes distribués à grande échelle

Machine parallèle à mémoire partagée



Uniform Memory Acces (UMA) architectures : Cray YMP, SX, etc..
Programmation en MPI ou OpenMP principalement

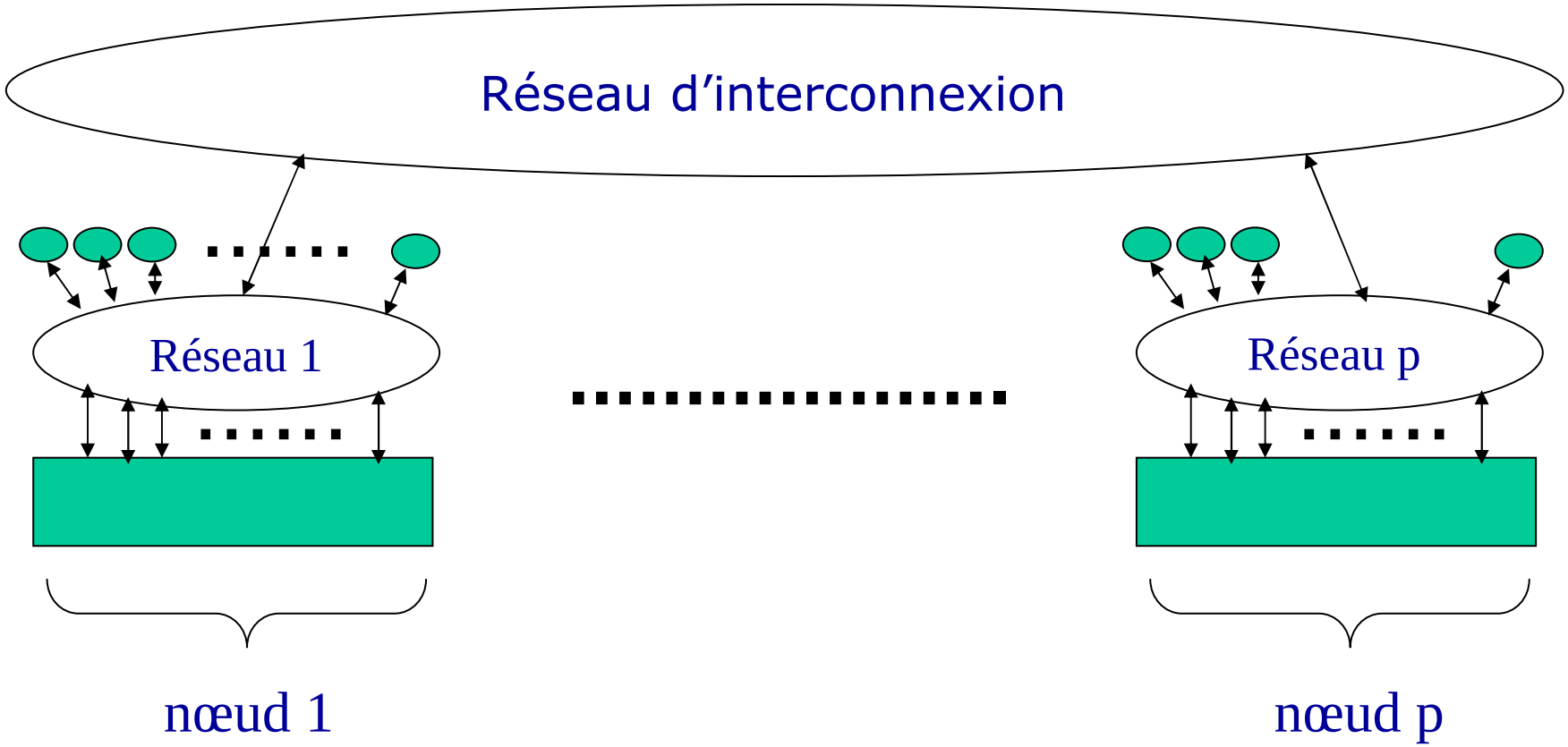
Machine parallèle à mémoires distribuées



Non Uniform Memory Access (NUMA) architectures : Hypercube d'INTEL, CM2 et CM5 (avec processeurs vectoriels), nCube, …….

Programmation en MPI principalement, avec langage data parallèle pour les CMs

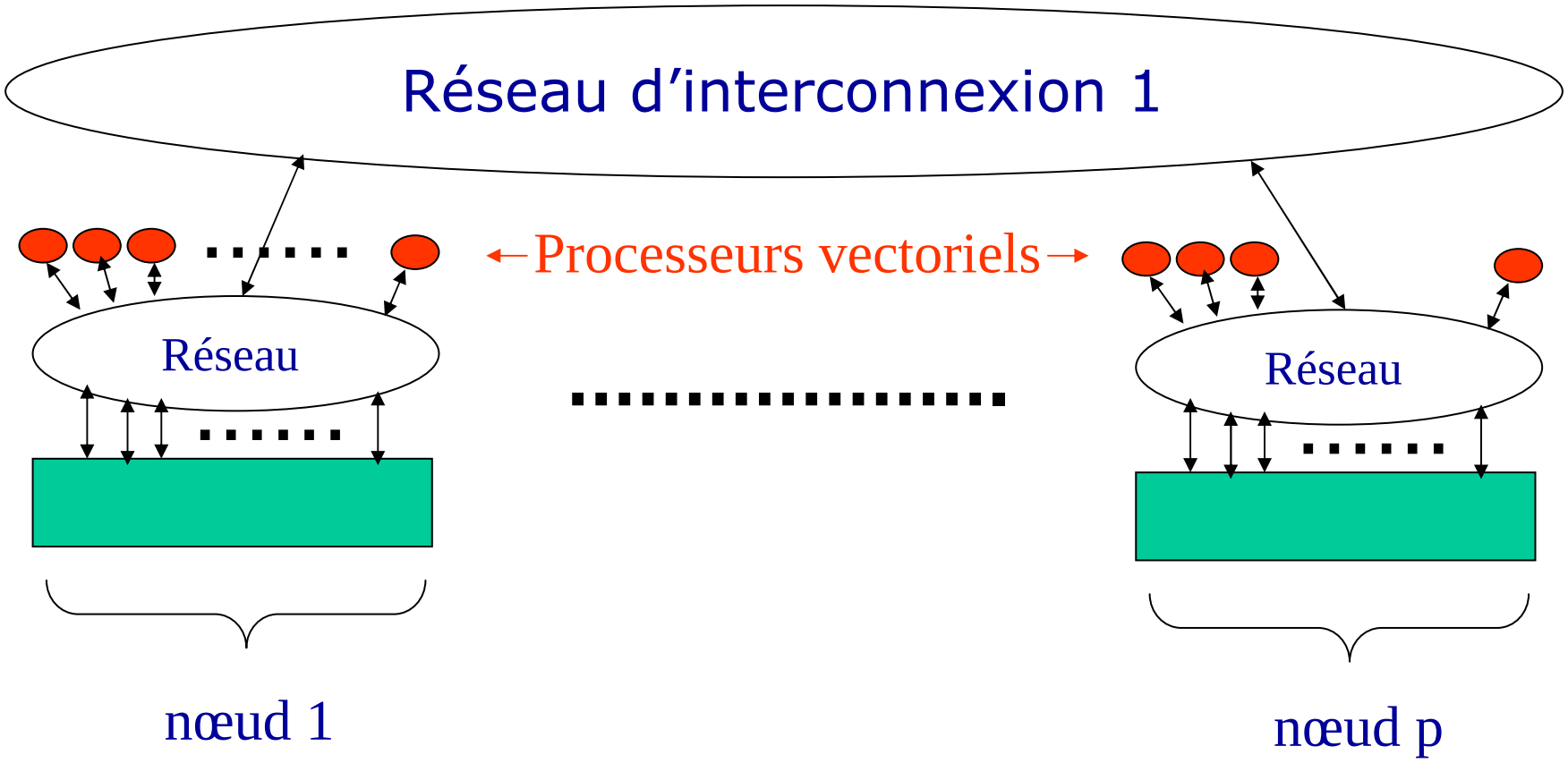
(Massively Parallel Processors) MPP



IBM SP4 par exemple, programmation en MPI et/ou OpenMP principalement

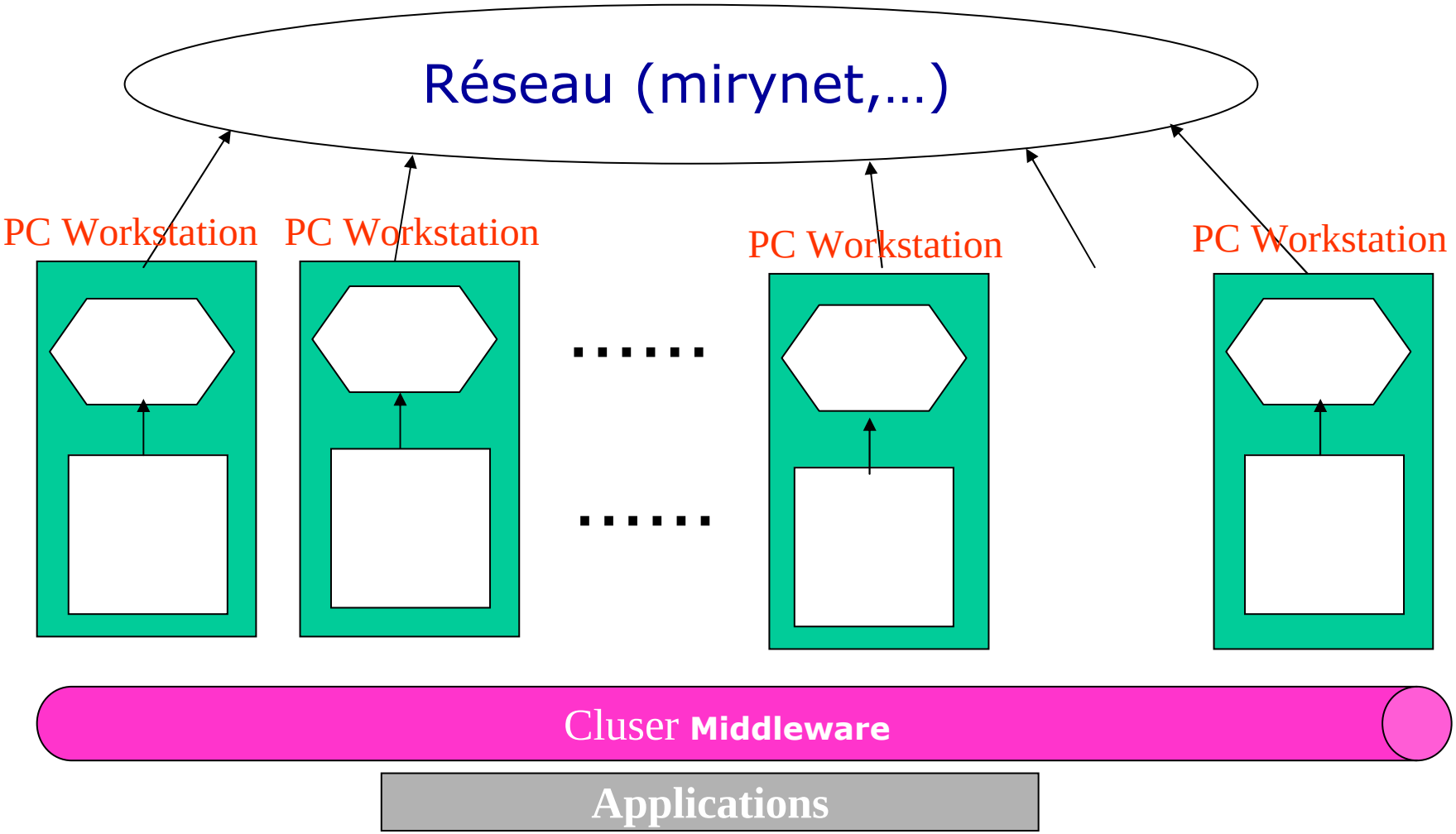
MPP/*Earth Simulator*

(computenick)

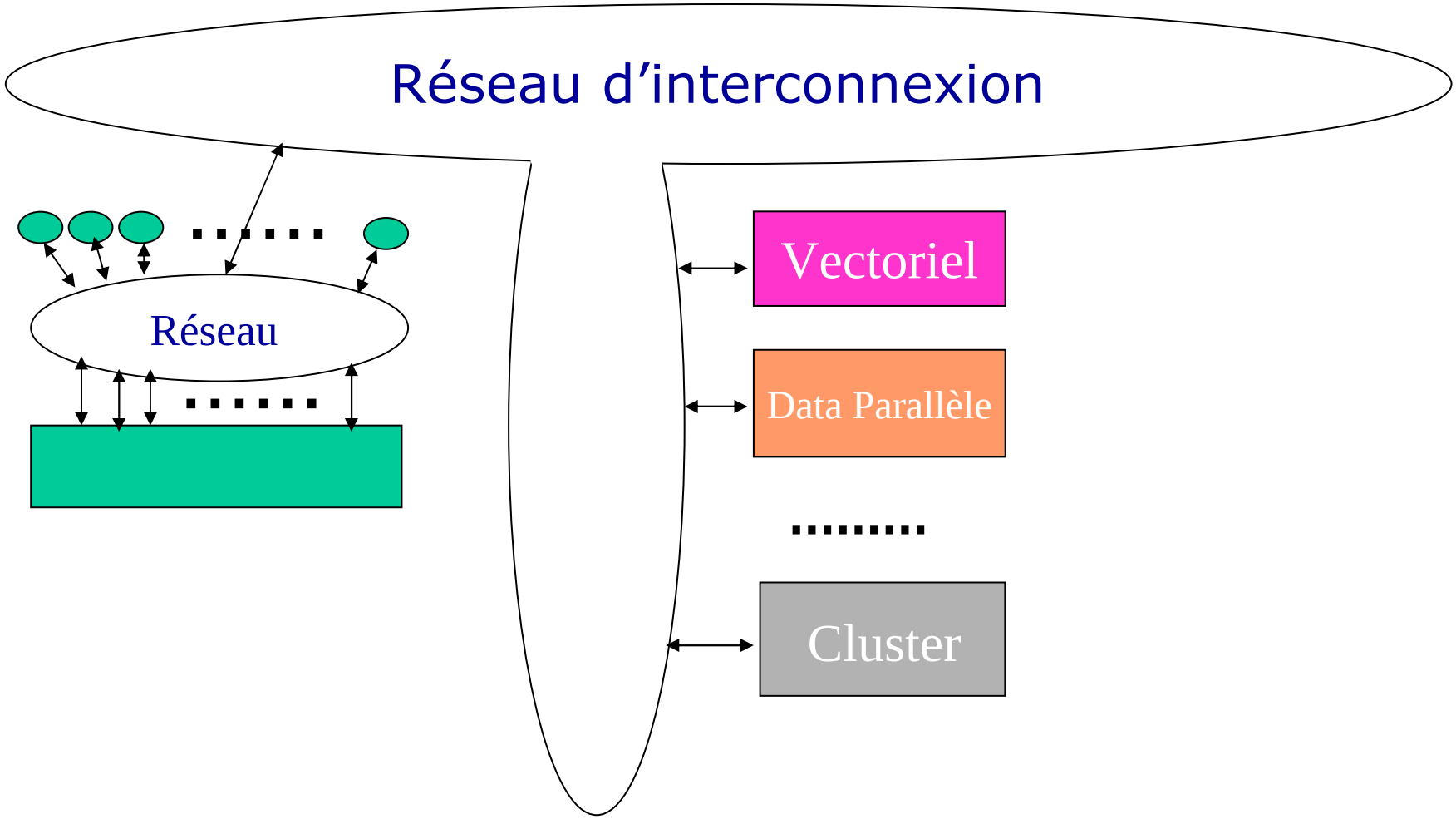


Programmation en MPI et/ou OpenMP ou en HPF

Grappes (Cluster, farm,…)



Calculs Intensifs Hétérogènes

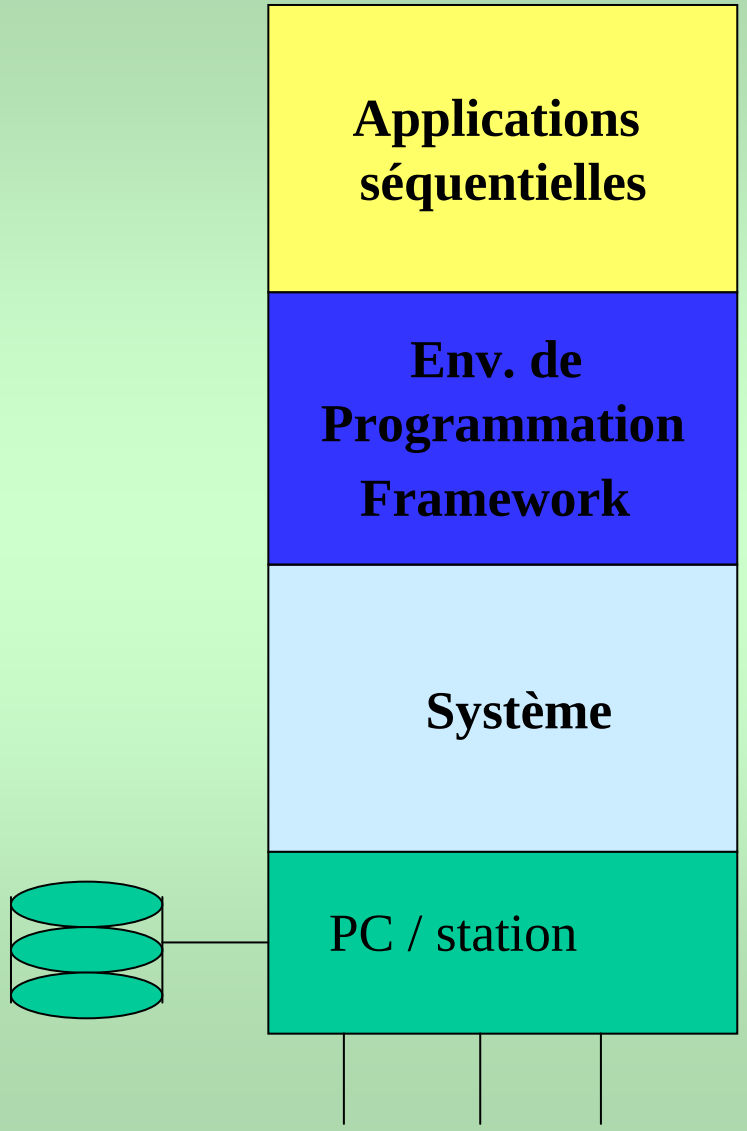


	SMP CC-NUMA	Clusters	Réseau
Nbre noeuds	10 -100	qques 10	10-1000
Comm. Inter-nœuds	-Centralisée - DSM*	-messages -DSM possible	Varié (IPC, messages, fichiers partagés…)
Job scheduling	Queue unique	Queues multiples souvent coordonnées	Queues indépendantes
Système	Unique	Homogènes ou hétérogènes	Hétérogènes
SSI support	Oui	- souhaité - souvent dans clusters spécialisés	Non
Propriétaire	unique	un ou plusieurs	multiples

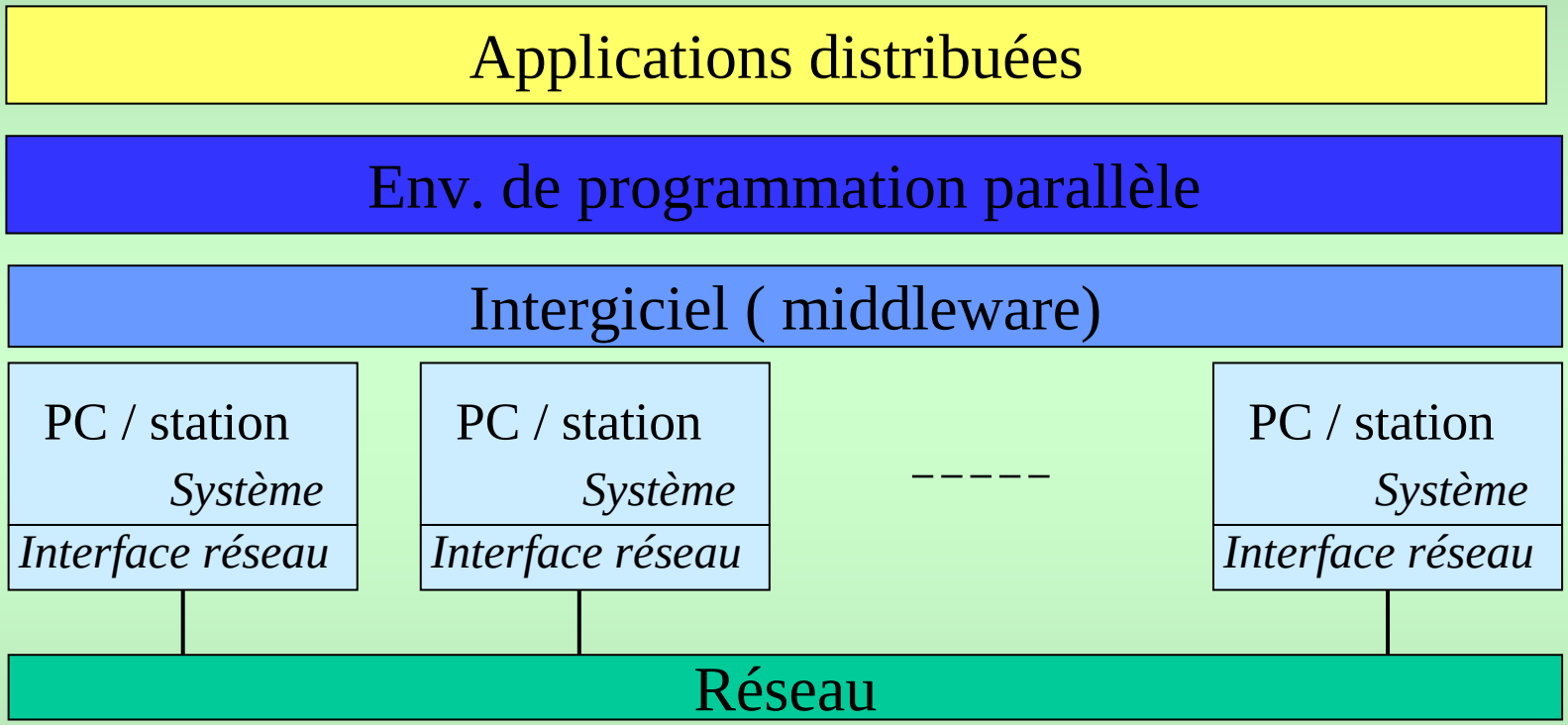
*DSM = Distributed Shared Memory = mémoire partagée répartie (MPR)

Machines Petaflopiques anoncées : Grille + milliers de nœuds a mémoire partagée comporant des dizaines de processeurs à des dizaines de cœurs.

◆ Séquentiel


Applications séquentielles
Env. de Programmation Framework
Système
PC / station

◆ Grappe de stations



Intergiciciel (Middleware)

- **Fonctions**
  - Fournir une interface (API) de haut niveau aux applications
  - Masquer l'hétérogénéité des systèmes sous-jacents
  - Rendre la répartition invisible (transparente)
  - Fournir des services répartis d'usage courant
- **Vise à faciliter la programmation répartie**
  - Développement, évolution, réutilisation des applications
  - Portabilité des applications entre plates-formes
  - Inter-opérabilité d'applications hétérogènes

Les réseaux

- **Partie critique des systèmes distribués**
  - Performances du réseau (débit, latence) → performances de la plate-forme
  - Ratio perf réseau/perf. Processeurs → choix de la granularité des traitements
- **Types de réseaux**
  - WAN : Wide Area Network
  - LAN : Local Area Network
  - SAN : Storage Area Network (pour clusters)

- **Quelques réseaux**
  - **Ethernet** (10Mb/s), **Fast Ethernet** (100Mb/s), **Gigabit Ethernet**(Gb/s) :
    - Même approche facilitant l'interconnexion de segments différents
    - Gain en latence plus faibles O(1ms)
  - **ATM** (Asynchronous Transfert Mode) (155Mb/s) : circuits virtuels et commutation de paquets de taille fixe (53 octets)
  - **Myrinet** : routage de type cut-through, débit 1,28Gb/s, latence 5 à 10 µs
  - **SCI** (Scalable Coherent Interface) standard IEEE :
    - Assure cohérence des caches (mise en oeuvre DSM)
    - Débit 5Gb/s, latence matérielle 1 µs, débit 8Gb/s, latence 2.4 µs
    - Latence totale <12 µs message de longueur nulle avec MPI sur Sun Sparc
  - **Autres réseaux :** Giganet (lat. 8 µs, débit 1.25Gb/s HiPPI, Server Net, Memory Channel…

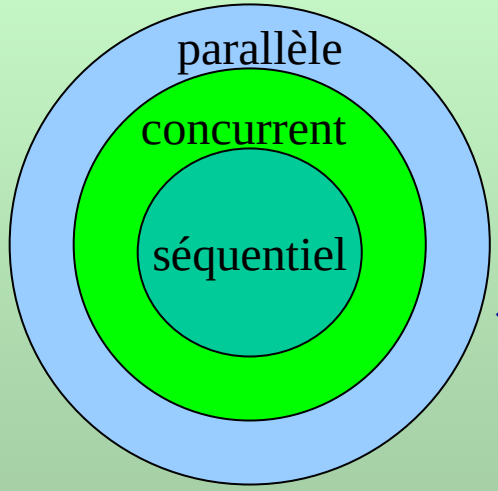
- **Remarque :**

  Performances annoncées / performances mesurées

  - **Utilisation d'une librairie de communication influence les performances** (couches logicielles supplémentaires)
  - **Ex.** : Performances en utilisant une librairie de communication MPI optimisée (Thèse Auguin 2000)

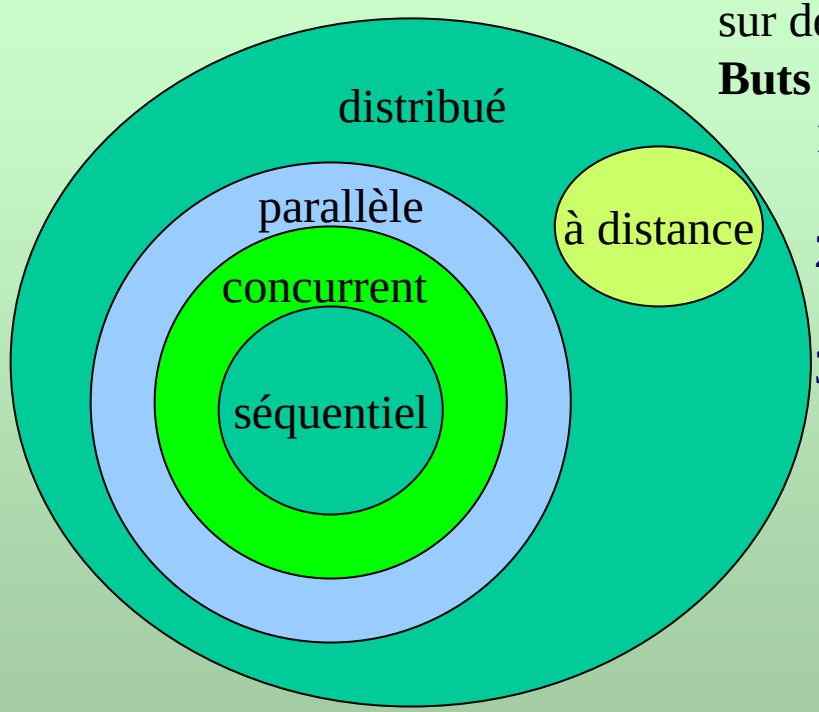
	Giganet	Myrinet	SCI
Latence mesurée *(annoncée)*	20 µs *(8 µs)*	12 µs *(8 µs)*	6 µs *(2.4 µs)*
Débit mesurée *(annoncé)*	96 Mb/s *(1.25 Gb/s)*	100 Mb/s *(1.28 Gb/s)*	70 Mb/s *(8 Gb/s)*

i. Parallélisme et distribution : panorama



- **Traitement séquentiel**
  Exécution séquentielle d'un flux d'instructions
  sur un seul processeur
- **Tâches concurrentes**
  Exécution simultanée de plusieurs tâches concurrentes
  But : meilleure utilisation CPU,
  meilleures performances globales
- **Traitements parallèles**
  Exécution simultanée de plusieurs traitements
  (sous-tâches coopérant à la réalisation d'un traitement)
  But : accroître la rapidité d'exécution pour
  1) obtenir plus rapidement un résultat,
  2) traiter des problèmes de plus grande taille,
  3) améliorer la qualité du résultat

Parallélisme et distribution : panorama *(suite)*



◆**Traitement distribué**

Répartition géographique de traitements simultanés sur des processeurs distincts et coopérants entre eux

Buts :

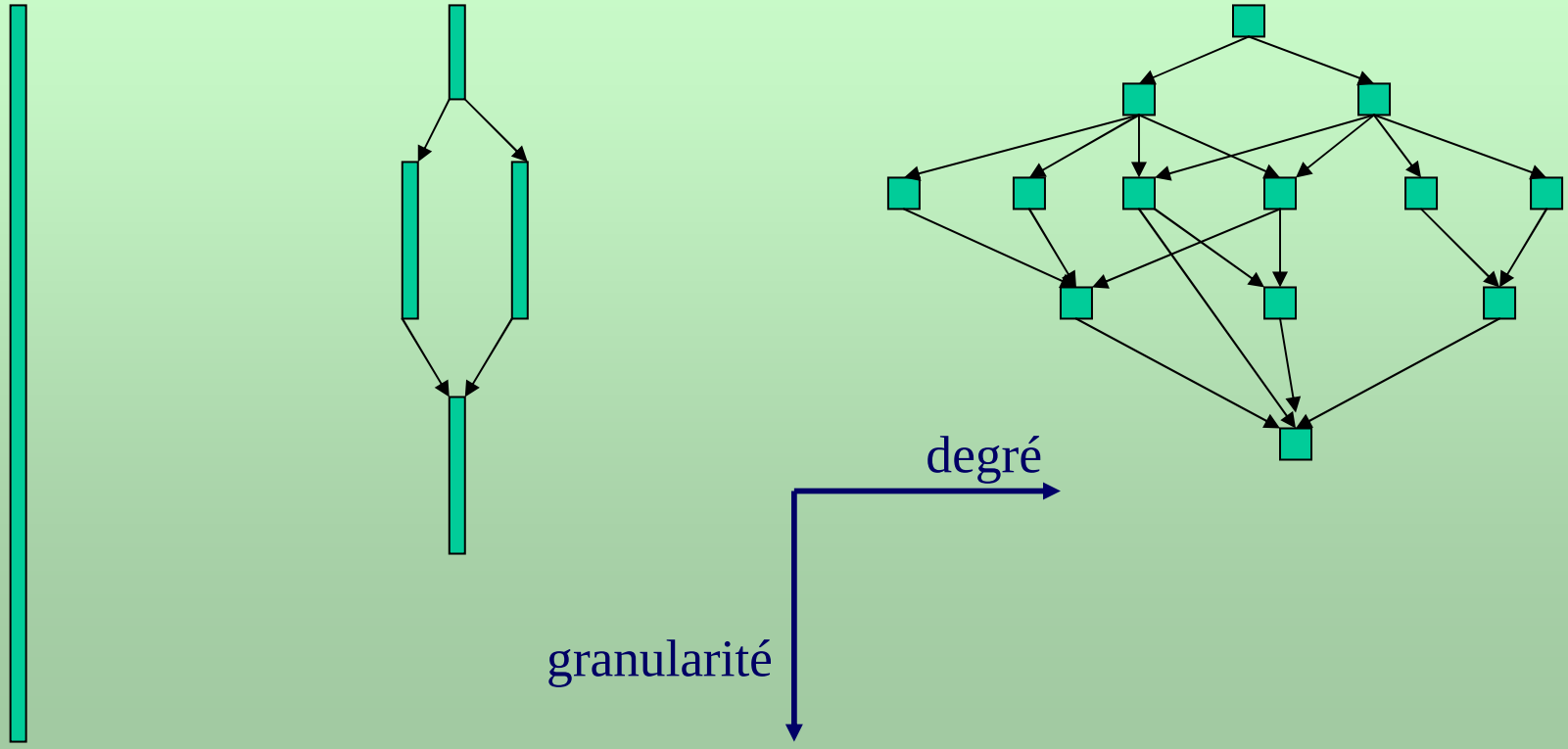
1. Mettre en œuvre le parallélisme pour augmenter l'efficacité de l'exécution,
2. Utiliser les ressources disponibles non utilisées,
3. Exploiter une distribution pré-existante des informations ou des traitements

ii. Degré et granularité du parallélisme

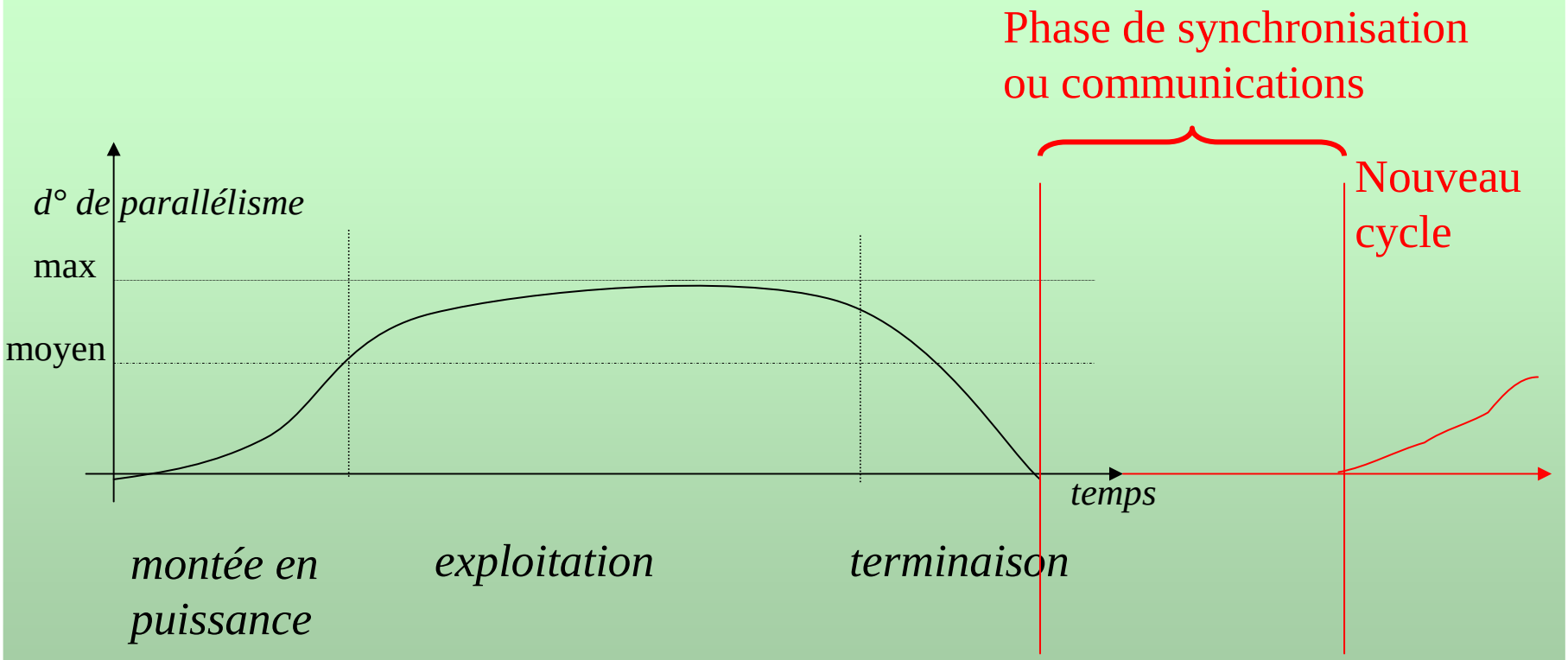
Séquentiel

Gros grain (*coarse grain)*

Fin grain (*fine grain)*



Degré de parallélisme



iii. Accélération et lois classiques

- **Accélération (speed-up)** **$S(n)=t_s / t_p(n)$** **(ou $1- t_s / t_p(n)$)**
- **Loi d'Amdhal (n processeurs)**
  - $S(n) = n / (1+(n-1)\, s)$ où s = fraction du temps de l'exécution séquentielle qui ne peut être parallélisée (exécutée en séquentiel)

$\Rightarrow$ Accélération maximale $S(n) \rightarrow 1/s$ quand $n \rightarrow \infty$

Exemple : s=10%, N=10 processeurs => accélération max 5,3 %
s=20%, N=10 processeurs => accélération max 3,6 %
s=5%, N=20 processeurs =>accélération max 10.26%

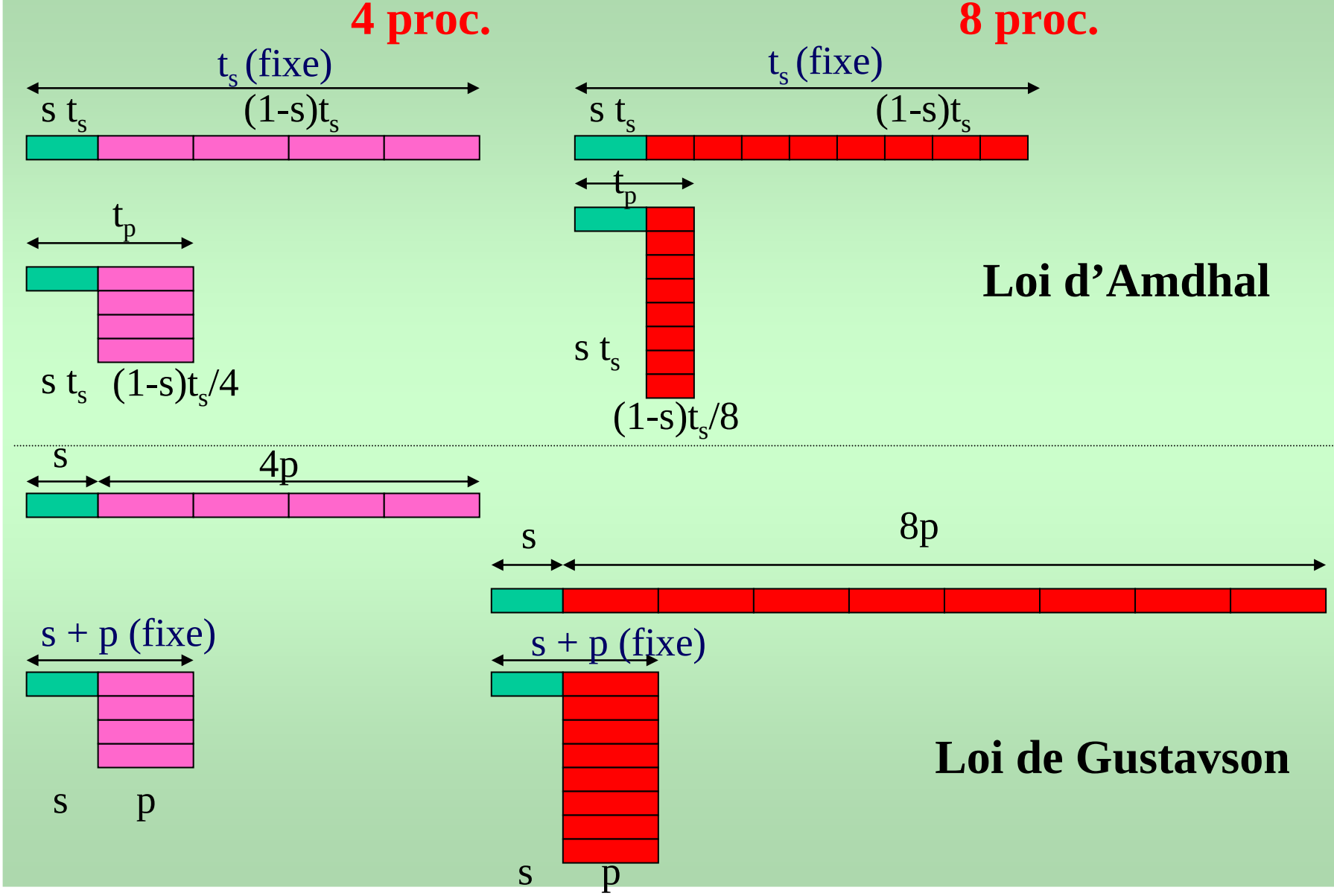
- **Loi de Gustavson**
  - $S(n) = n + (1-n)\, s$

s=5%, N=20 processeurs =>accélération max 19.05%

- **Lois basées sur des hypothèses différentes**

- Loi d'Amdhal : taille du pb constant t_s fixe
- Loi de Gustavson : Tps d'exécution constant (pb de plus grande taille) s+p fixe

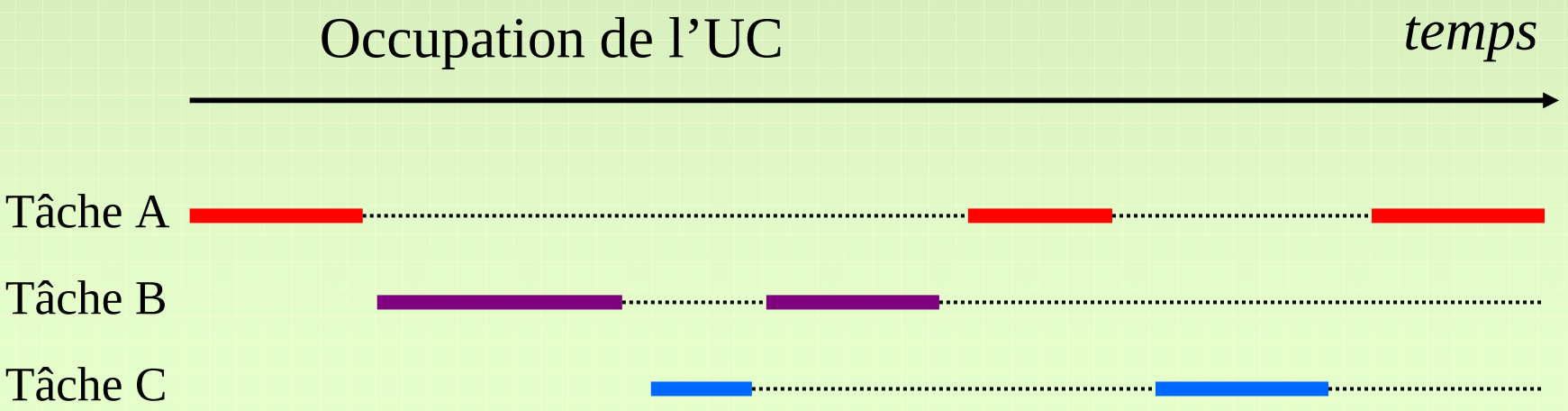
◆ **Illustration**



iv. Les formes de parallélisme

- **Concurrence**
  - Partage de l'UC
- **Le parallélisme de contrôle**
  - Graphe de tâches
- **Le parallélisme de données (SPMD)**
  - Fragmentation des données
- **Le parallélisme de flux**
  - Fonctionnement pipe-line

◆ Traitements concurrents

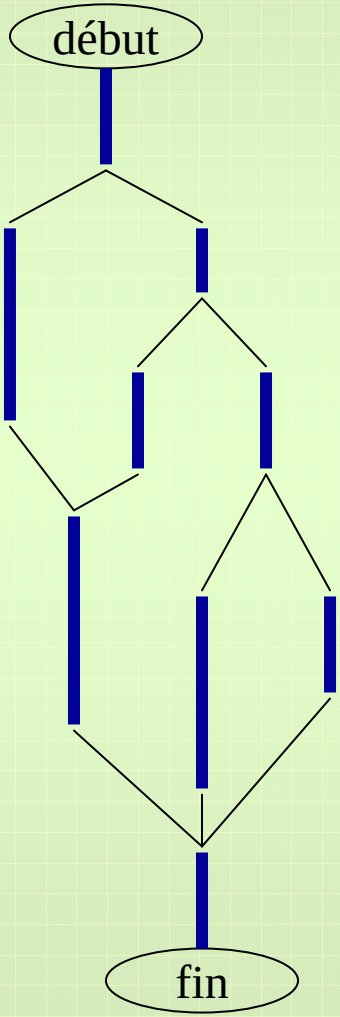


Une seule tâche est allouée à la fois à l'UC par le scheduler, sauf dans le cas de processeur disposant de l'hyperthreading (comme INTEL 2600, 2800, 3000 MHz)

◆ **Le parallélisme de contrôle** (basé sur le flux d'instructions)

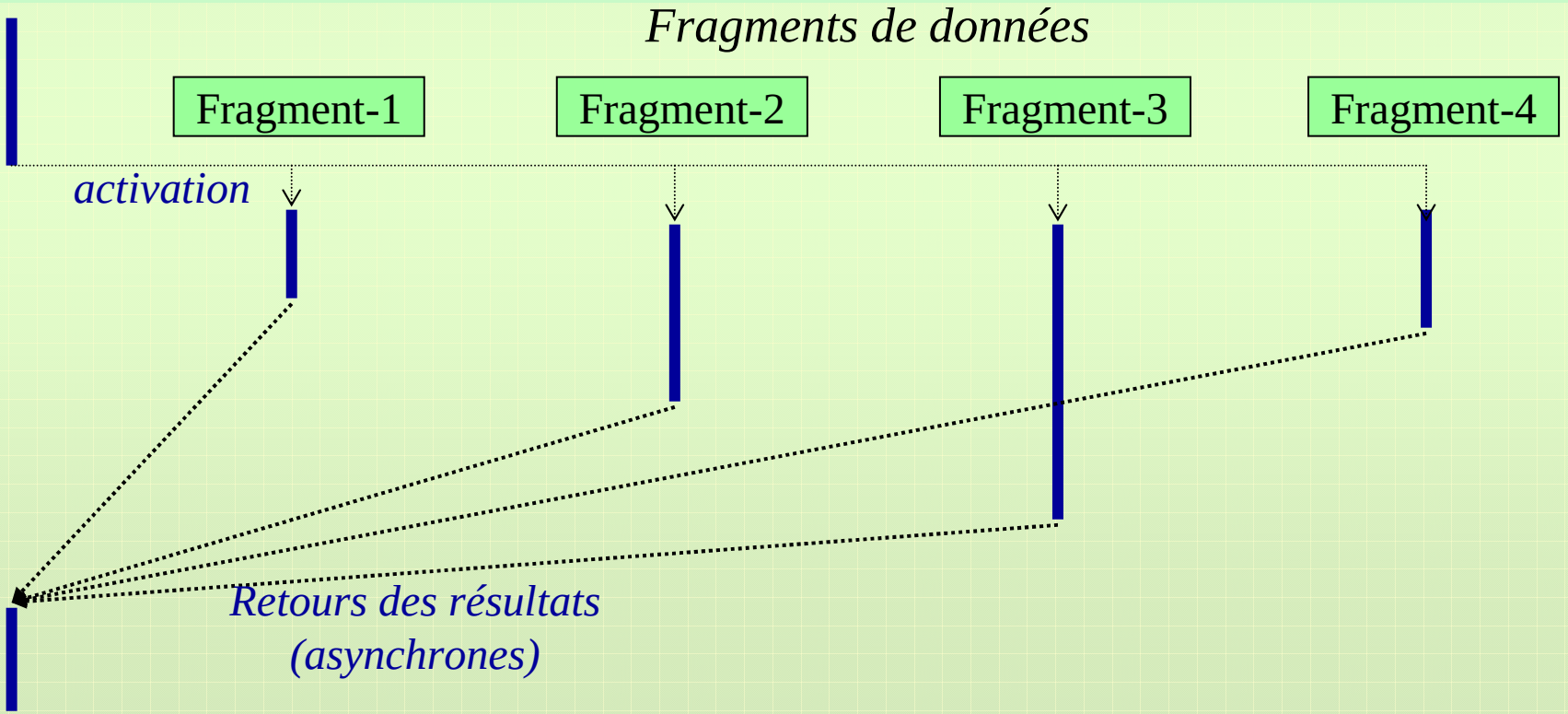
Découpage de la tâche en un **graphe de sous-tâches** à distribuer (statique)

Une bonne distribution suppose une connaissance suffisante des temps d'exécution respectifs des sous-tâches



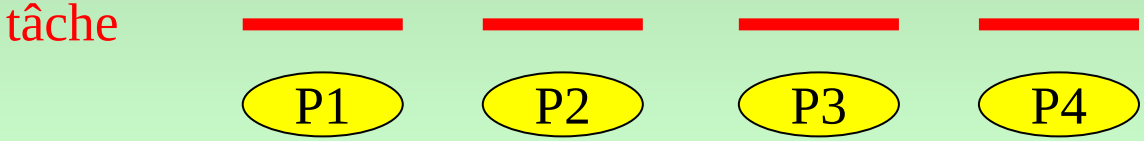
◆ Le parallélisme de données

- mode SPMD : Single Program Multiple Data
- s'appuie sur une fragmentation des données : une tâche s'exécute sur chacun des fragments de données

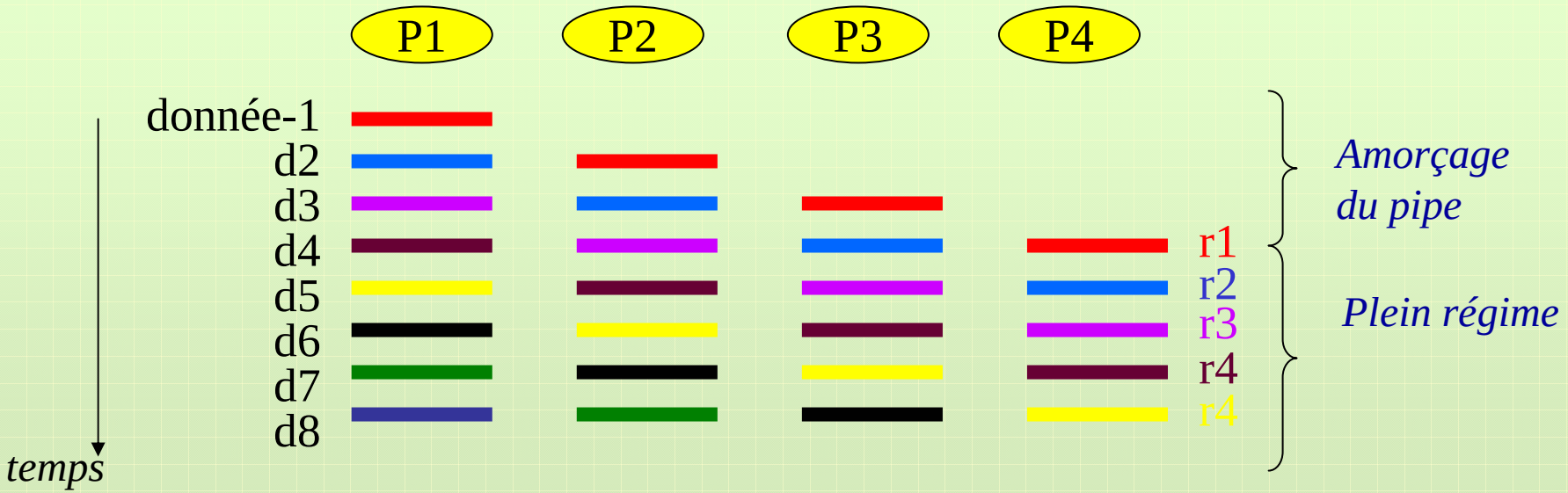


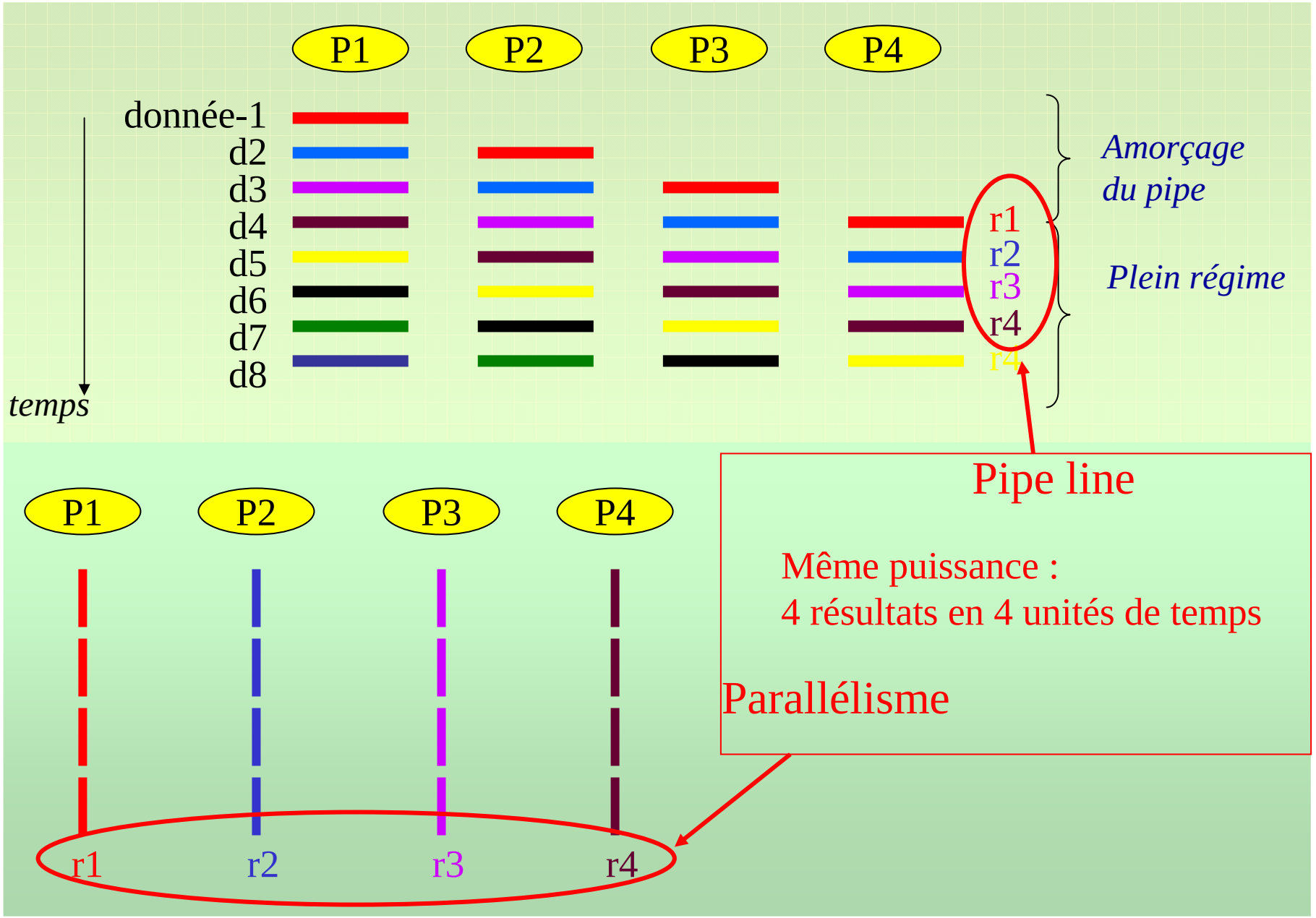
◆ Le parallélisme de flux (mode pipe line)

- une tâche est divisée en une succession de sous-tâches qui sont exécutées sur une suite de processeurs



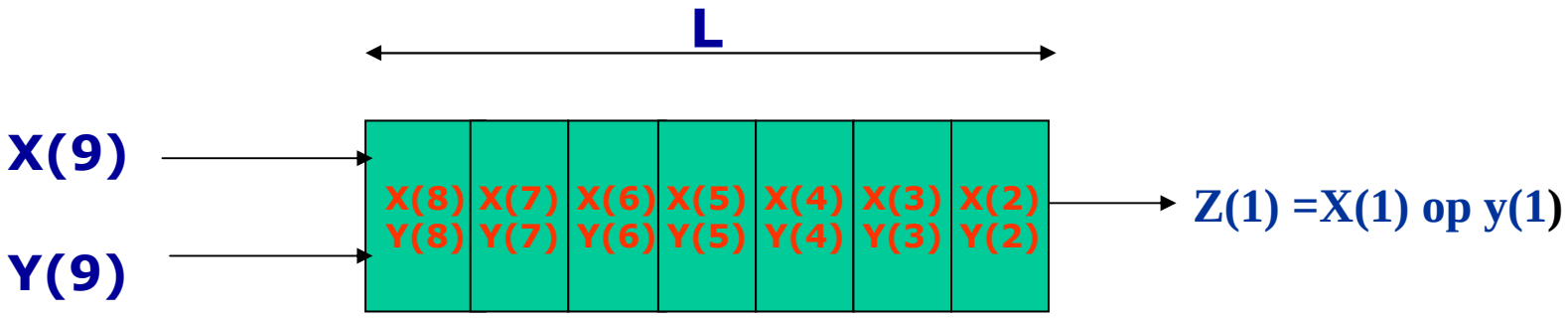
- Les données à traiter passent les unes à la suite des autres dans chacun des processeurs




P1
P2
P3
P4
donnée-1
d2
d3
d4
d5
d6
d7
d8
temps
r1
r2
r3
r4
r4
Amorçage du pipe
Plein régime
Pipe line
Même puissance :
4 résultats en 4 unités de temps
Parallélisme
P1
P2
P3
P4
r1
r2
r3
r4

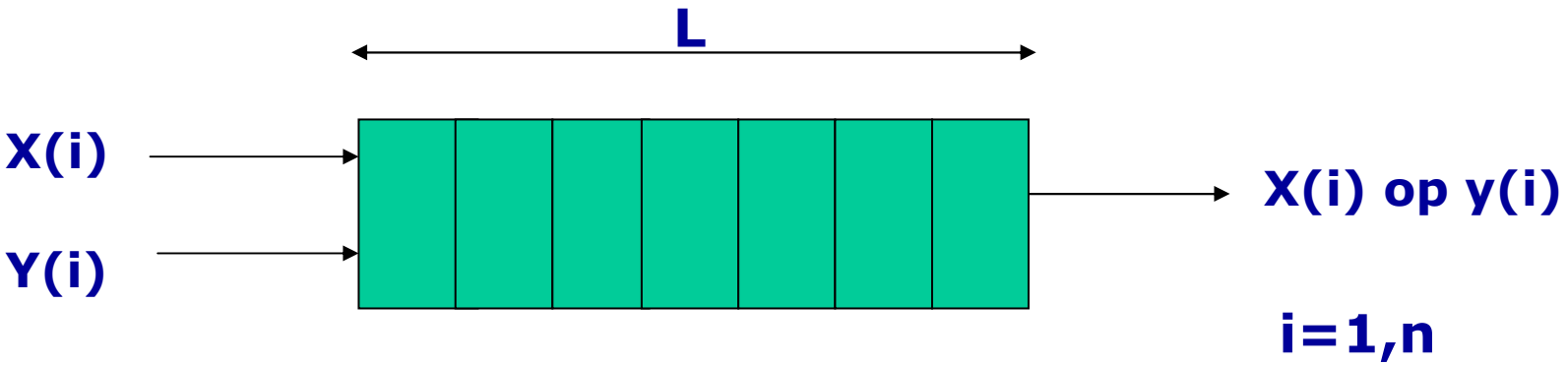
Calcul séquentiel versus vectoriel

- Calcul VLIW, optimisation des caches
- Pipeline arithmétique, calcul vectoriel



Calcul séquentiel versus vectoriel

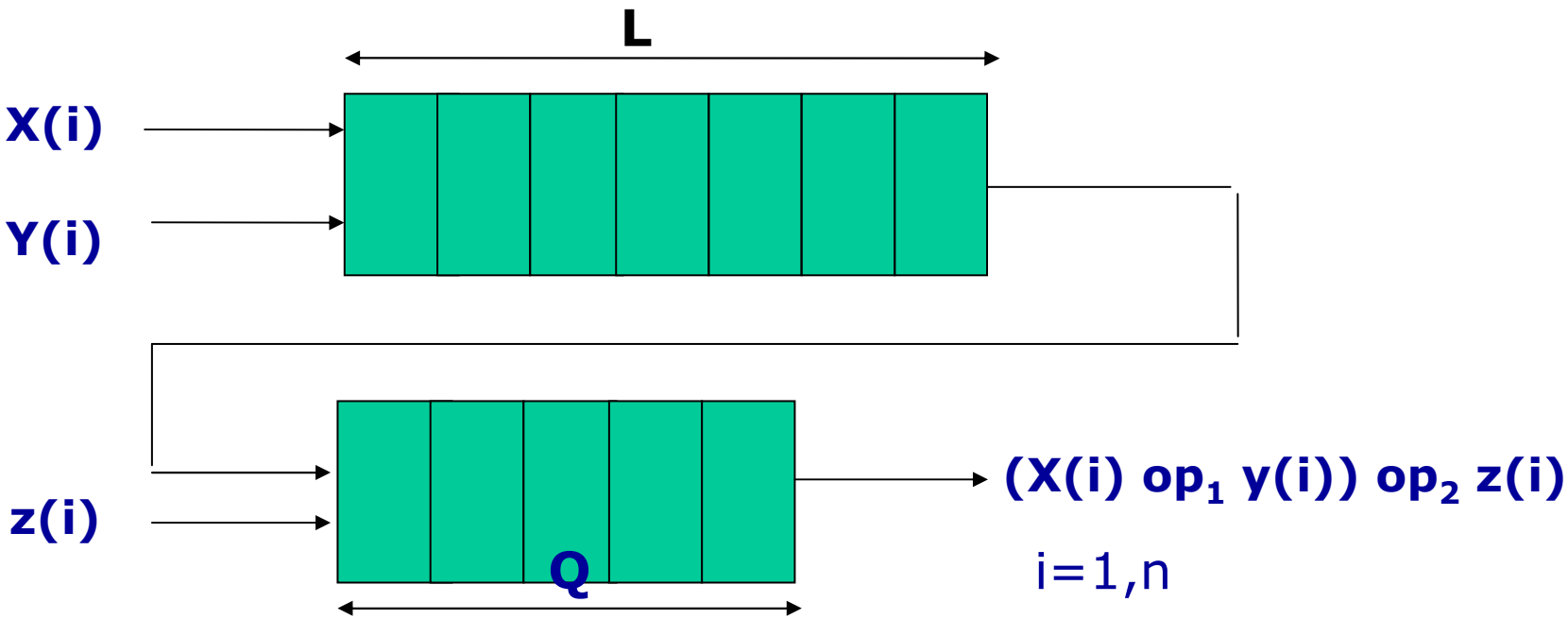
- Calcul VLIW, optimisation des caches
- Pipeline arithmétique, calcul vectoriel



$t_{seq} = t_{horloge}\ n\ L$

$t_{vect} = (L-1)t_{horloge} + n\ t_{horloge}$

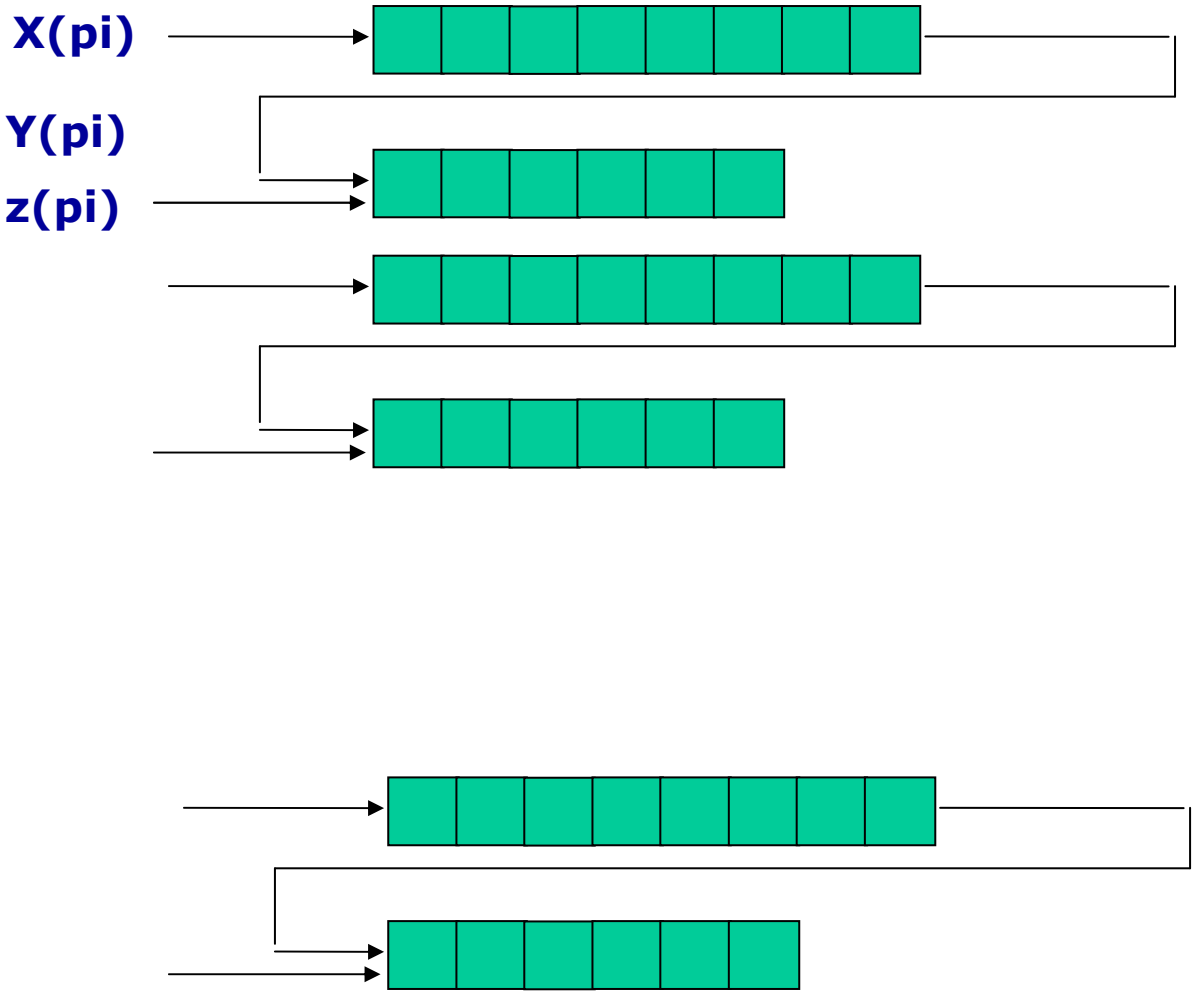
Calcul vectoriel, chaînage de pipes



$t_{seq} = t_{horloge}\ n\ (\ L + Q\ )$

$t_{vect} = (L+Q-1)t_{horloge} + n\ t_{horloge}$

Chaînage de pipes et parallélisme



v. Les tâches distribuées

- **Processus et threads**
  - Niveau processus
  - Multi-threading
    - Threads utilisateurs
    - Threads systèmes
    - Threads Posix et processus légers
- **Transactions**
  - Transactions distribuées (utilisant des données partagées distribuées)
  - Récupération en cas de panne lors d'une transaction distribuée
  - Gestion et cohérence des informations partagée

iii. MPI : Message Passing Interface

- Bibliothèque offrant au programmeur un environnement de programmation permettant d'exploiter des primitives de communication de haut niveau
- permet les communications entre des programmes (Fortran, C, …) dans des environnements hétérogènes
- Les différents aspects :
  1. Les communicateurs : groupe de processus et contexte
  2. Les communications point à point
  3. Les types de données
  4. Les communications globales

- **Les communicateurs**
  - Groupe de processus
    - ensemble de processus numérotés à partir de 0
    - Opérations d'accès au groupe :
      - taille du groupe
      - Rang du processus dans le groupe
      - Comparer les rang du même processus dans 2 groupes
      - Comparer 2 groupes
    - Constructeurs : création de nouveaux groupes à partir de groupes existants
      - Primitives MPI_GROUP_UNION, MPI_GROUP_INTERSECTION, MPI_GROUP_DIFFERENCE(*group1*, *group2*, *newgroup*),
      - MPI_GROUP_INCL, MPI_GROUP_EXCL(*group*, *n*, *rank*, *newgroup*) crée un nouveau groupe incluant ou excluant les n premiers proc de group
  - Contexte : permet de différencier des ensembles de communication (équiv. Marquage, coloriage des messages)
  - Gestion des communicateurs
    - Ex. MPI_COMM_DUP(*comm*, *newcom*), MPI_COMM_CREATE(*comm*, *group*, *newcom*), MPI_COMM_SPLIT(…)

- **Les communications point à point :**
  - 2 opérations « envoyer un message » : send( ) et « recevoir un message » : receive( )
  - Envoi bloquant

    MPI_SEND(*buf, count, datatype, dest,tag,comm*)

    où : *buf, count, datatype* précisent les informations à transmettre

    dest précise le destinataire i.e. rang dans le groupe

    tag marque le message (contexte)

    comm désigne le communicateur concerné
  - Réception bloquante

    MPI_RECV(*buf, count, datatype, source,tag,comm, status*)

    Variantes : MPI_ANY_SOURCE( )

    MPI_ANY_TAG ( )

    infos associées au message transmises dans *status*
  -

- Ordres non bloquants : préfixés par la lettre I
  - MPI_ISEND(*buf, count, datatype, dest,tag,comm, request*) : l'appel alloue un objet requête qui pourra être interrogé plus tard pour connaître l'état de la communication à l'aide d'ordres comme
  - MPI_WAIT(*request, status*), MPI_TEST(*request, flag, status*), ou
  - MPI_WAIT_ANY, MPI_WAIT_ALL, MPI_WAITSOME - pour communications multiples-

Les communications point à point (suite)

- Modes d'envois et de réception des messages

- **Standard** : l'envoi est exécuté et est bloqué jusqu'à ce que le message soit bufferisé chez le récepteur ou lu par le récepteur. Dès que l'on est assuré que le message est stocké chez le destinatire, le programme peut continuer
- **Buffered** : le message est systématiquement bufferisé pour l'envoi et libère le programme émetteur ; opération local, le programmeur peut gérer explicitement le buffer
- **Synchronous** : l'ordre send ne peut se terminer que quand le destinataire a reçu le message (a commencé la réception)
- **Ready** : l'envoi ne peut avoir lieu que quand le destinataire a commencé l'exécution de l'ordre de réception receive

- Notations

En préfixant les ordres SEND et ISEND par B (buffered), S (synchronous) ou R (ready)

Ex. MPI_BSEND( ), MPI_IBSEND( ) etc…

- Remarque : MPI garantit le non-déséquencement des messages dans les communications point à point

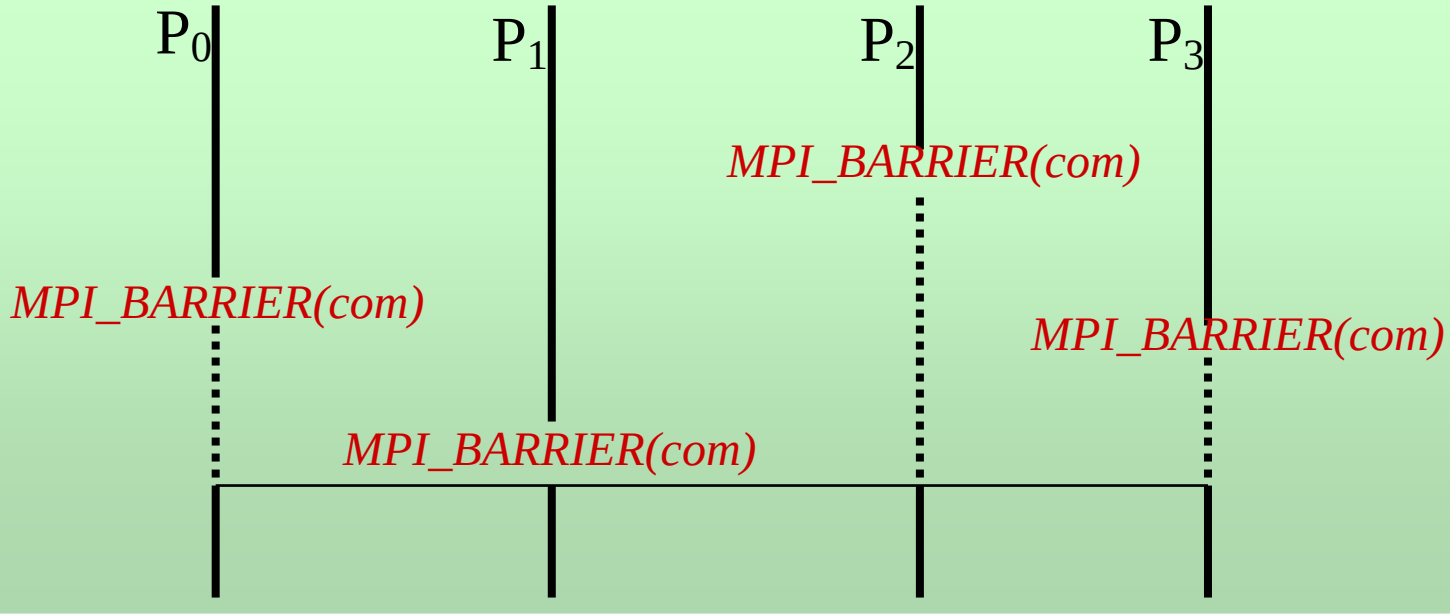
Les types de données

- Types de données standards pour les communications pour supporter l'hétérogénéité
- Recouvrent les types de base de C et Fortran
- Constructeurs de types à partir des types de base
  - MPI_TYPE_CONTIGUOUS ( )
  - MPI_TYPE_VECTOR ( )
  - MPI_TYPE_INDEXED ( )
  - MPI_TYPE_STRUCT ( )
- Pas de transmission de structures de données comportant des pointeurs → sérialisation
  - emballer les structures de manière contiguë dans des buffers pour l'expédition : MPI_PACK ( )
  - Déballer à la réception : MPI_UNPACK ( )

◆ Les communications globales

(*collective communications en MPI)*

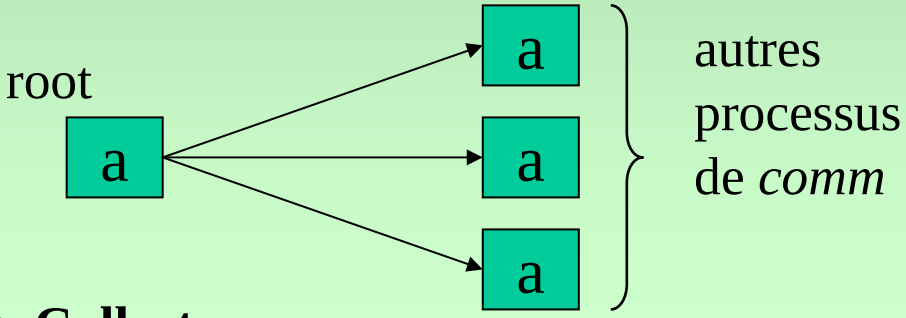
- **<u>Barrière de synchronisation</u>** : MPI_BARRIER (*comm*) bloque l'exécution du processus qui l'exécute jusqu'à ce que tous les processus du groupe du communicateur passé en argument atteignent l'ordre de barrière



Les communications globales (2)

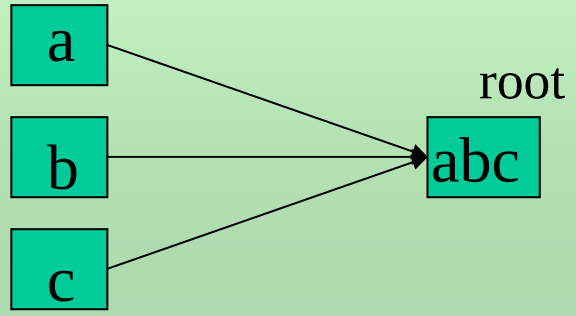
Diffusion

MPI_BCAST (*buffer, count, datatype, root,comm*)



Collecte

MPI_GATHER (*sendbuffer, sendcount, sendtype, recvbuffer, recvcount, recvtype, root, comm*)

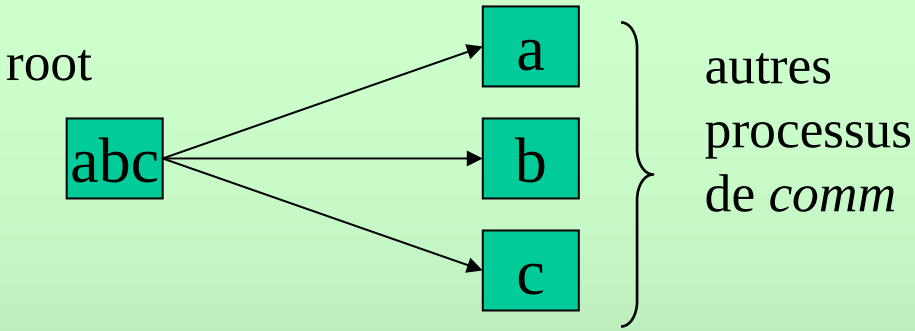


Variante :
MPI_GATHERV ( )
rassemble des données de tailles différentes

➢ **Distribution**

MPI_SCATTER ( ) et MPI_SCATTERV ( )

Opération inverse de gather

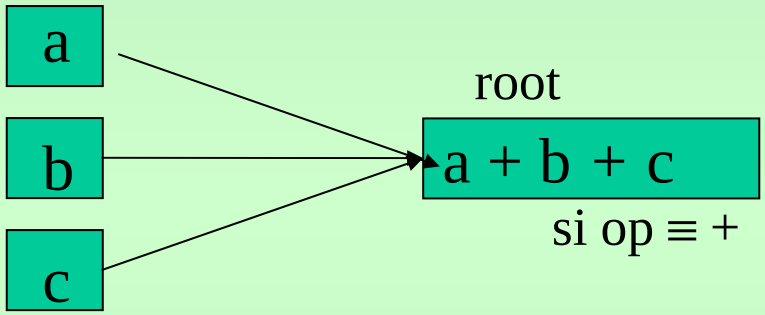


+ autres primitives plus générales MPI_ALLGATHER ( )

Les opération de réduction

réduction

MPI_REDUCE (*sendbuffer*, *recvbuffer*, *count*, *datatype* *op*, *root*, *comm*) identique à la primitive gather mais avec exécution de l'opération *op* à chaque réception de donnée



- Op = opérations prédéfinies de base comme maximum, minimum, somme, produit ou opérations logiques
- Définition de *op* par le programmeur (commutative ou non)

MPI_OP_CREATE (*function*, *commute*, *op*)

Réductions préfixes

MPI_SCAN( ) : le processus de rang i reçoit le résultat de l'opération sur les messages de rang 0 à i-1

Chap. II

Algorithmique distribuée

II. Algorithmique distribuée

1. Introduction : problèmes et outils
2. Exemple 1 : l'exclusion mutuelle distribuée
3. Le temps dans les systèmes distribués
4. Adaptation d'un algorithme centralisé à un univers distribué
5. Interblocage dû aux communication : illustration du calcul diffusant
6. Routage distribué
7. Asynchronisme et recouvrement des communications

1. Introduction : problèmes et outils algorithmiques

- **Tâches distribuées coopèrent**

  ⇒ tâches partagent de l ’information

  - information locale : privée
  - information globale : partagée
    - unique et localisation fixe

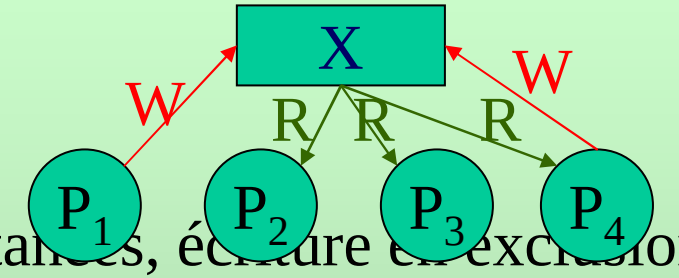
      → pb. Accès distants systématiques
      Concentration des demandes d ’accès
    - unique et localisation variant durant l ’exécution

      → pb. Gestion de la localisation
    - dupliquée

      → pb. Gestion de la cohérence des copies

Différentes approches

- **Disponibilité d ’une mémoire commune**

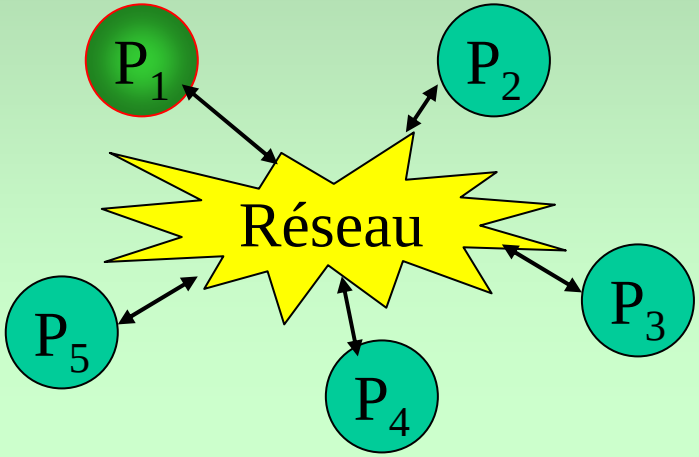


➔ lectures simultanées, écriture en exclusion mutuelle

➔ goulot d ’étranglement : traitement séquentiel de requêtes distantes

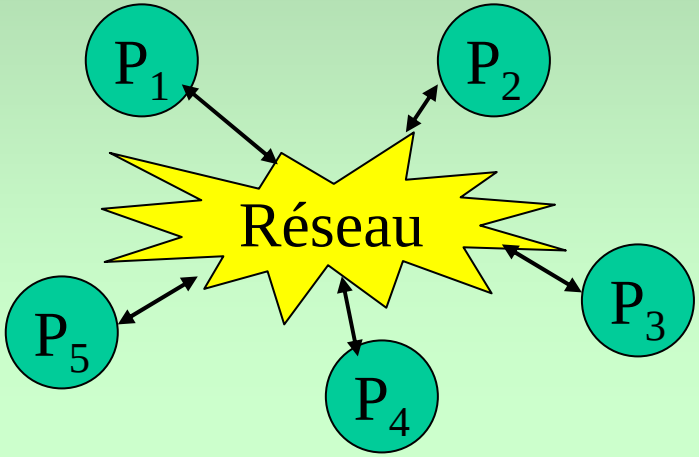
➔ cohérence des copies dans les caches

◆ Traitement distribué centralisé



- ➔ un site propriétaire gestionnaire (serveur) : P_1
- ➔ les autres sites (clients) s ’adressent au site maître
- ➔ mêmes problèmes qu ’avec mémoire commune (goulot d ’étranglement, exclusion mutuelle, cohérence des copies)
- ➔ problème de latence du réseau

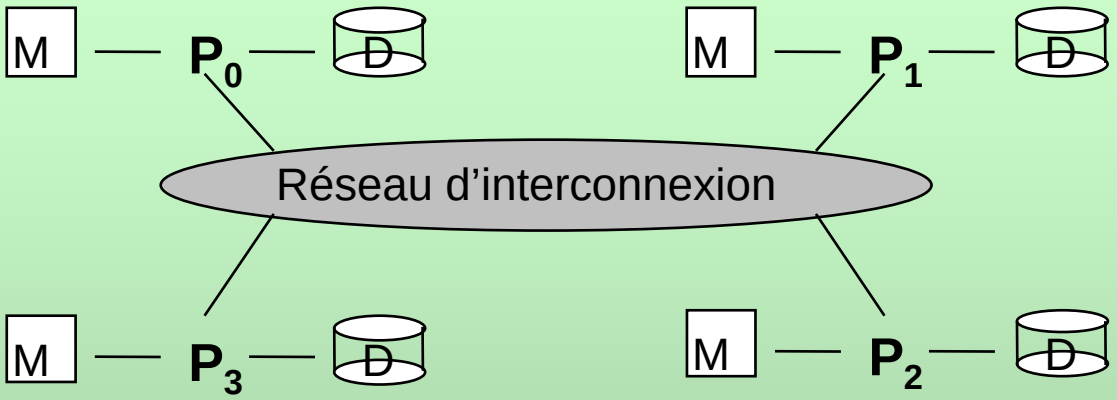
◆ Traitement totalement distribué



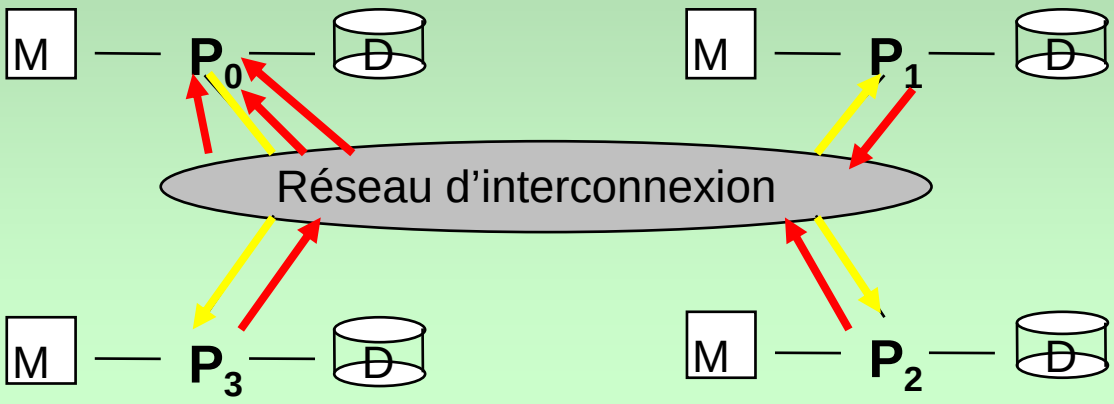
- ➔ pas de site privilégié (P2P)
- ➔ l ’information globale peut se trouver dans n ’importe quel site
- ➔ problèmes de localisation, de mise à jour (rafraîchissement), de cohérence des copies
- ➔ problème de latence du réseau

Absence d ’état global observable

- **Validité de l ’information globale**

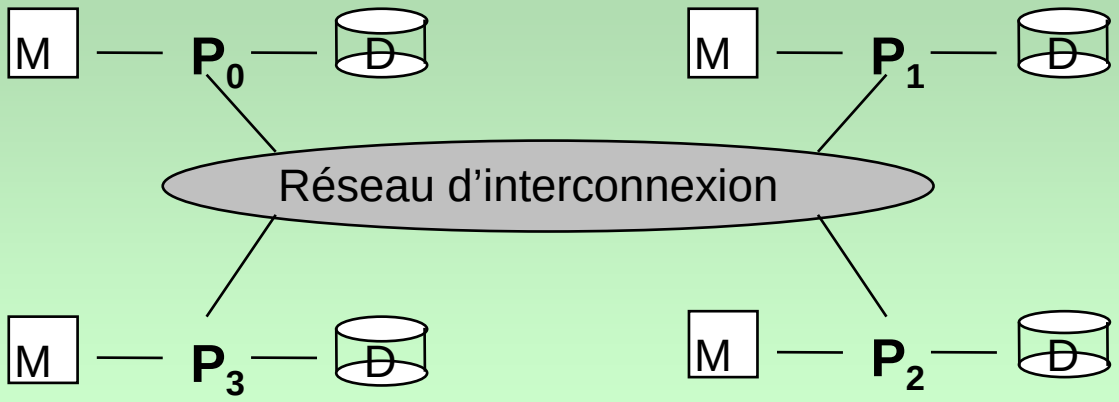


Exemple : chaque site P_i calcule en permanence sa charge c_i
P_0 désire connaître la charge globale du système (c_0,c_1,c_2,c_3)



- P_0 envoie à la date t une requête aux autres P_i
- cette requête est reçue, puis prise en compte par les P_i aux temps t_i
- chaque P_i envoie à P_0 sa charge $c_i(t_i')$
- P_0 reçoit plus tard les c_i $C = < c_0(t), c_1(t_1'), c_2(t_2'), c_3(t_3') >$

C = valeur approchée du vecteur des charges



absence de mémoire commune
temps de communication ⇒ **État global approché**

Questions :
1. Validité de l 'algorithme utilisant une information globale
2. Disponibilité/rafraîchissement de l 'état global approché

Ordre des évènements

- **2 évènements se produisent dans des sites différents :**

  évènement *a* dans P_0 et évènement *b* dans P_1

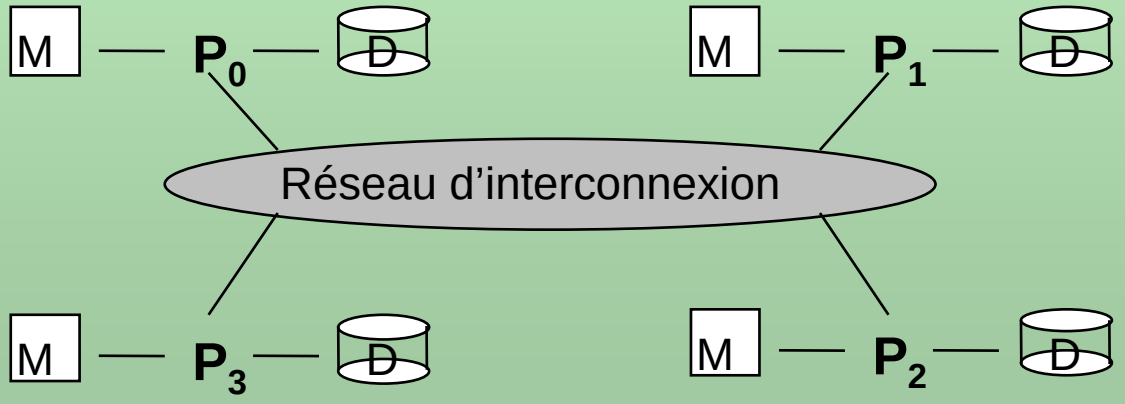
  <u>Questions</u> :

  - l ’évènement *a* précède-t-il l ’événement *b* ?

  Existence d ’une mesure globale du temps

  - l ’événement *a* a-t-il pu influencer l ’événement *b*?

  Notion de dépendance/indépendance causale



Outils pour l ’algorithmique distribuée

- **Le jeton circulant** :

  Principe : faire circuler un « privilège »

  - l ’unique détenteur à un moment donné est le seul autorisé à exécuter certaines actions

  - le jeton peut contenir des informations globales mises à jour au fur et à mesure de ses déplacements

- **l ’estampillage**

  Gestion d ’« horloges » logiques servant à dater les communications

- **le calcul difffusant**

  Construction d ’un arbre de recouvrement des processus pour diffuser un traitement, puis concentrer les résultats

◆ L ’estampillage simple

processus (site) $P_i \rightarrow$ horloge logique h_i ($\equiv$ compteur)

les messages sont estampillés (m, h_i, i)

Gestion de h_i par P_i :

- lors de l ’émission d ’un message m par P_i :

  h_i est incrémentée avant l ’émission de (m, h_i, i)
- lors de la réception de (m, h_j, j) par P_i :

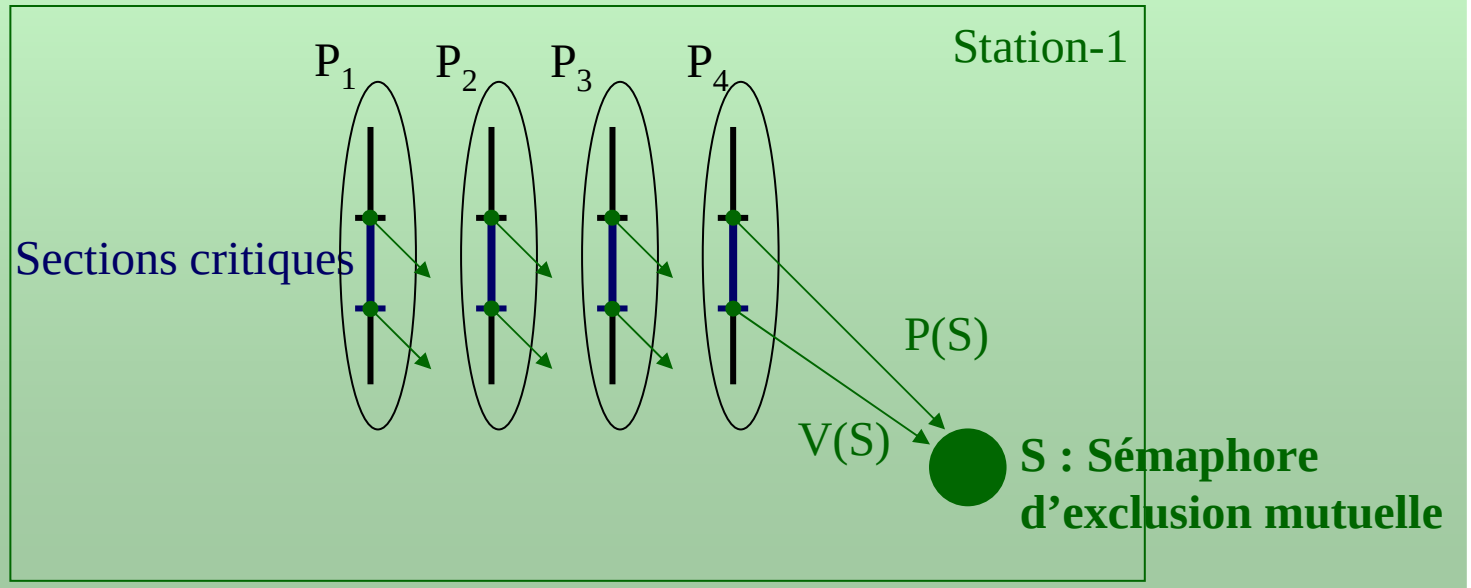
  $h_i = \text{Max}(h_i, h_j) + 1$
- h_i n ’est pas modifiée par d ’autres évènements internes

  ($\neq$ émission / réception)

2. Un premier exemple : l ’exclusion mutuelle distribuée

- **Le problème de l’exclusion mutuelle : des processus possèdent une partie de code à exécuter en exclusion mutuelle (section critique)**

Situation classique : processus concurrents



3. Le temps dans les systèmes distribués

- **Temps logique (temps causal)**
  - ➔ **précédence entre évènements sur des sites distants**
  - ➔ **interne au programme (pendant son exécution)**
- **Temps réel**
  - ➔ **date, durée temps global commun**
  - ➔ **temps externe**

Le temps causal

- **3 types d ’évènements**

  i) émission d ’un message

  ii)réception d ’un message

  iii) événement interne

- **Contraintes fondamentales**

  C1 : Les évènements sur un même site sont ordonnés

  C2 : Pour tout message *m*, l ’événement « émission de *m* » précède l ’événement « réception de *m* »

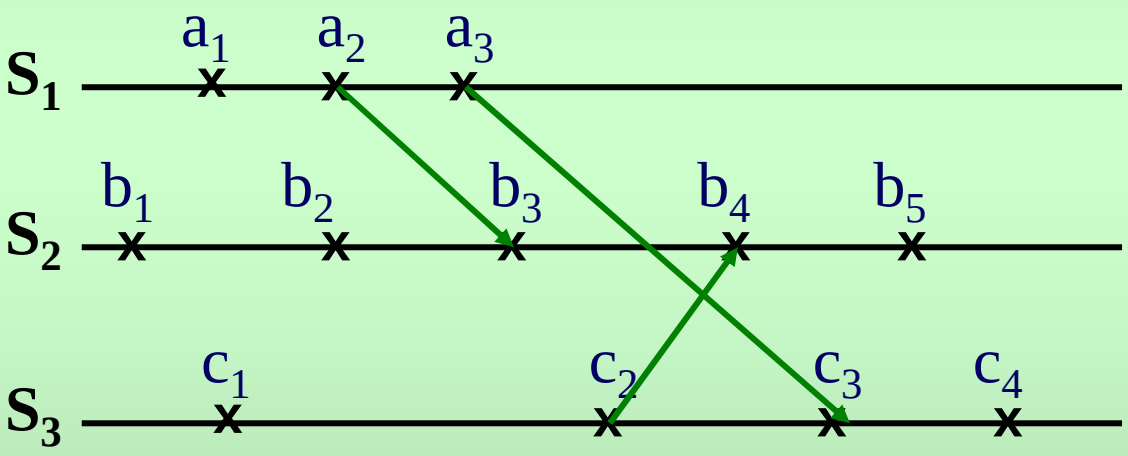
◆ Relation de causalité

a précède b (a→b) si 1 des 3 conditions suivantes sont vérifiées :

i) a et b ont lieu sur le même site et a s ’est produit avant b

ii) a=émission de *m* et b=réception du même message *m*

iii) ∃ événement c tel que a→c et c→b (transitivité)



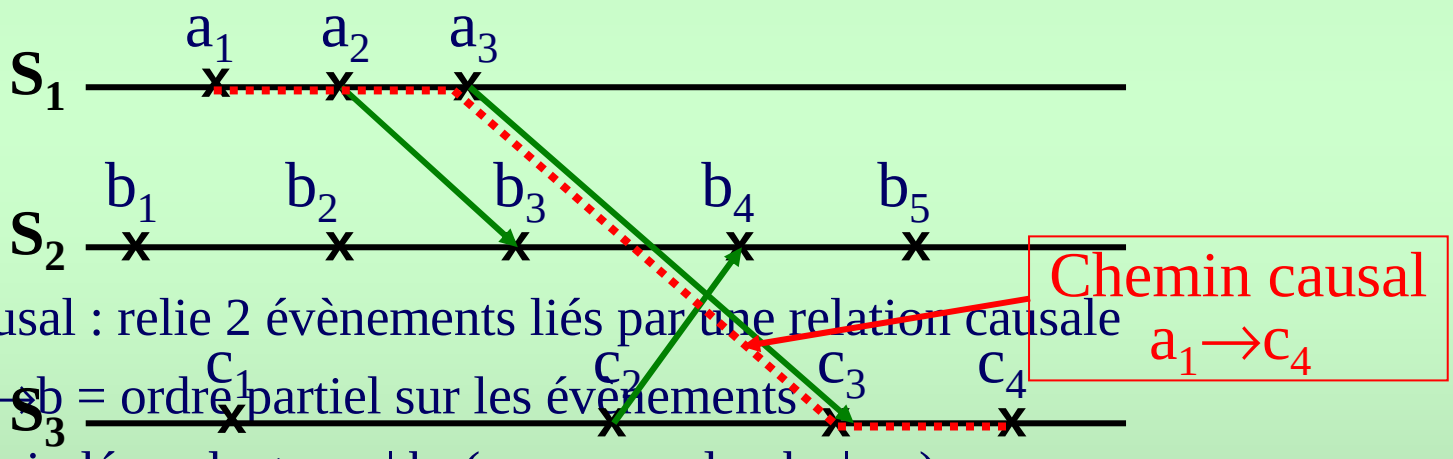
◆ Relation de causalité

a précède b (a→b) si 1 des 3 conditions suivantes sont vérifiées :

i) a et b ont lieu sur le même site et a s ’est produit avant b

ii) a=émission de *m* et b=réception du même message *m*

iii) ∃ événement c tel que a→c et c→b (transitivité)



➔ Chemin causal : relie 2 évènements liés par une relation causale

➔ causalité a→b = ordre partiel sur les évènements

➔ évènements indépendants : a | b (par exemple : $b_2 \mid c_3$)

4. Adaptation d'un algorithme centralisé à un univers distribué : exemple de l'interblocage

- **Deux types :**
  - **Lié à l'allocation des ressources** (classique)

    Les processus sont en attente réciproque de ressources détenues par des processus eux-mêmes bloqués.
  - **Lié à la communication des messages** (propre au distribué)

    Les processus sont en attente de messages en provenance de processus eux-mêmes bloqués en attente de message.

- **Deux approches possibles**

  - **Technique de prévention** (approche pessimiste)

    Contrôler en permanence pour empêcher l'occurrence d'une situation d'interblocage

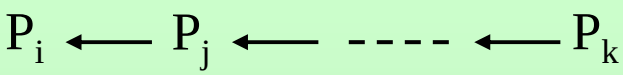
  - **Technique de détection** (approche optimiste)

    Laisser faire sans contrôle, mais vérifier de temps en temps la présence d'une situation d'interblocage et réagir éventuellement en conséquence pour rétablir une situation normale

- **Caractérisation de l'interblocage lié aux ressources**
  - Hypothèse simplificatrice :

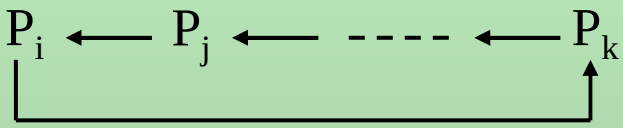
    ressource unique et à accès exclusif
  - Graphe de dépendance (des attentes, des conflits)

P_i attend une ressource détenue par P_j



P_i dépendant de P_k

  - Interblocage



Interblocage ≡ circuit dans le graphe

- **Algorithme de prévention de l'interblocage (algorithme de Lomet)**
- **Principes**
  - Mécanisme par « annonces » : chaque processus annonce les ressources qu'il est susceptible d'utiliser
  - Graphe de dépendance :

    $P_i \rightarrow P_j \equiv P_i$ utilise une ressource appartenant à l'annonce de P_j
  - Interblocage ≡ circuit dans le graphe de dépendance
  - Algorithme de prévention : l'allocation d'une ressource à un processus n'est autorisée que si elle ne crée pas de circuit dans le graphe de dépendance

    ⇒ Maintenir en permanence le graphe de dépendance

L’algorithme CMH (Chandy-Misra- Haaz)

- **Hypothèse**
  - Pas de déséquencement des messages
- **Principe général**
  - **Un processus bloqué (passif) détermine s’il est en situation d’interblocage en lançant un calcul diffusant**
  - Construction d’un arbre de diffusion de P_i vers les ED(P_i)
  - Puis remontée des réponses vers la racine
  - Seuls les processus passifs participent au calcul diffusant
  - Plusieurs lancements de calculs diffusants simultanés possibles