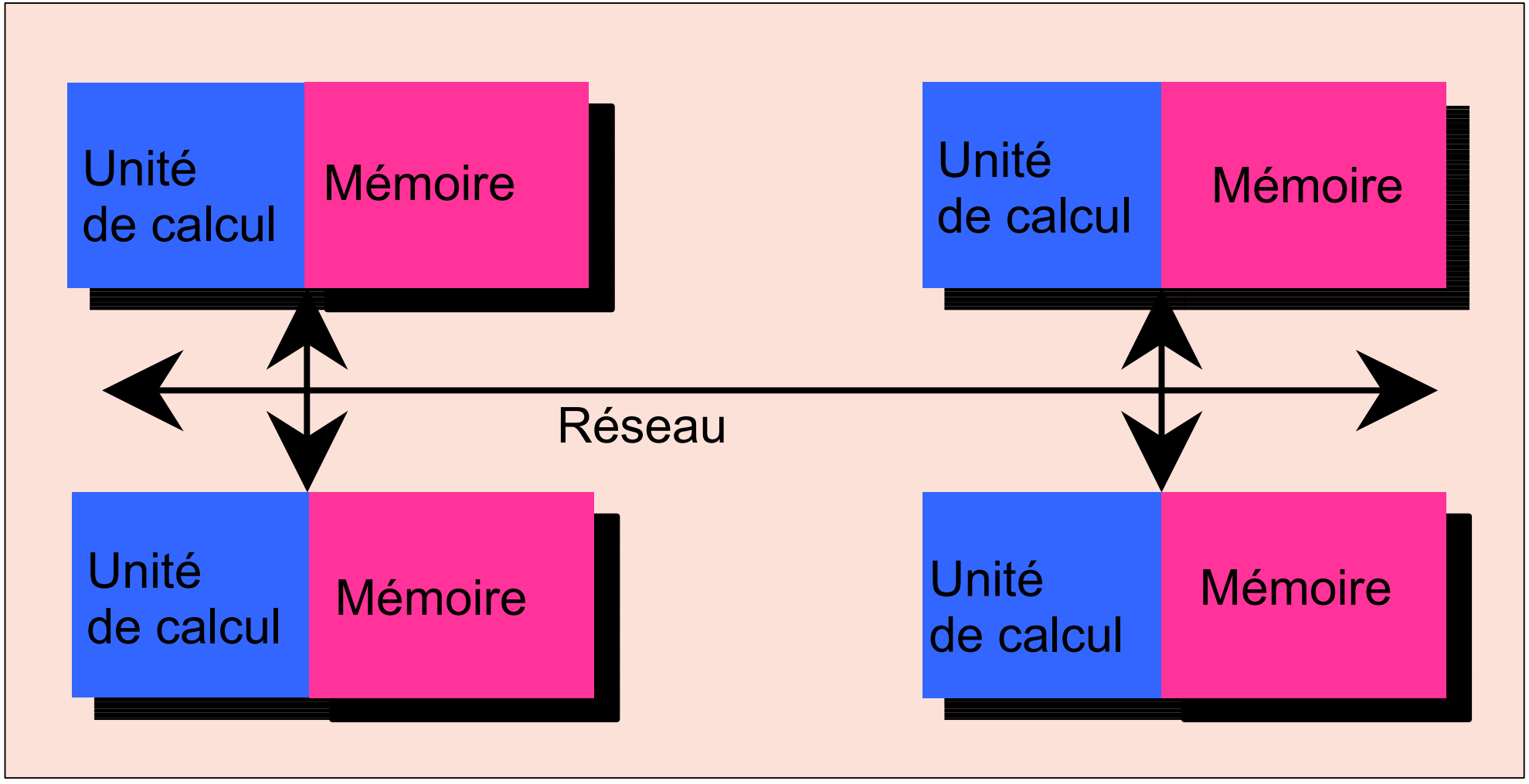
Introduction
Algorithmes Parallèles

Mémoire et performance

- **Localité**
  - Organisation hiérarchique : globale → caches
  - Différence de rapidité d'accès
- **Synchronisation**
  - Copie vers locale pour les calculs
  - Mettre à jour les données en globale
- **Parcours**
  - **Aléatoire** (ex : recherche dichotomique)
  - **Séquentiel** (ex : itération sur un tableau)
    - cohérent, donc synchronisation par bloc possible

Architecture à mémoire distribuée

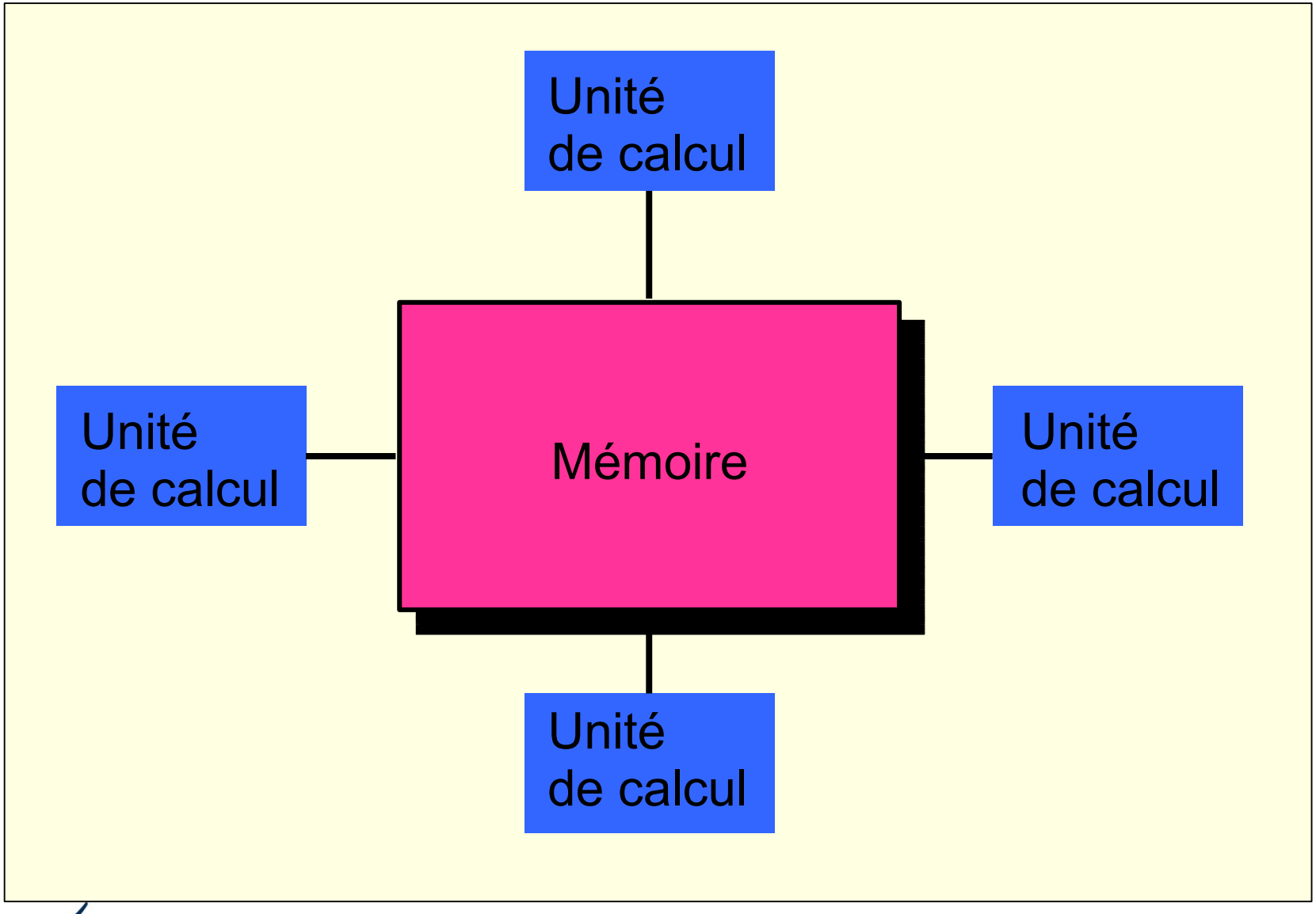


Architecture à mémoire distribuée

- Exemples
  - Grilles de calculs

    Ensemble de machines connectés
- Avantages
  - Ajout de composantes facilités
- Désavantages
  - Comment répartir et synchroniser les données ?
  - Réseau rapide

Architecture à mémoire partagée



Architecture à mémoire partagée

- Exemples
  - CPU multi-coeur, multi-processeurs, GPU
- Avantages
  - Synchronisation classique des données
- Désavantages
  - Passage à l'échelle moins souple

- Conclusion
  - Architectures hybrides

Problème des accès mémoires

- CPU ou GPU
  - 1 **accès mémoire** ↔ qques centaines de cycles !!
    (1 cycle ↔ 1 opération flottante)
- Solutions
  - CPU
    - Caches énormes, hiérarchiques
    - Prefetching (mise en cache) automatiques
  - GPU
    - Mini caches supposant des accès cohérents
    - Résultats temporaires dans des registres

Parallélisation

- Répartition du calcul
  - **Même tâche** sur plusieurs processeurs/cœurs
    - Multi-cœur / GPU / Cluster
  - **Tâches spécialisées**
    - Cell, CPU vs GPU, …
- Deux actions primordiales
  - **Distribuer** les données / calculs
  - **Récupérer** les données / solutions
- Pb majeur : **synchronisation** des données

Approches

- Modèle **SIMD** : single instruction multiple data
  - Mémoire partagée et synchronisée entre processeurs
  - Même instructions en parallèle
  - Ex : GPU, multi- cœurs
- Question de lecture / écriture
  - En parallèle, exclusif ?
  - **CREW** : concurent-read, exclusive write
  - **EREW** : exclusive-read, exclusive-write
  - **CRCW** : concurent-read, concurent-write

Opération Vectorielle vs Scalaire

- Constatation
  - Manipulation fréquente de **vecteurs**
    - 3D (position dans l'espace)
    - 4D (couleur)
  - Opérations similaires par composantes
    - Addition de deux vecteurs, division par un scalaire, ...
  - Opérations sur les vecteurs
    - Produit scalaire, vectoriel

Opération Vectorielle vs Scalaire

- Constatation
  - Manipulation fréquente de **vecteurs**
  - Opérations similaires par composantes
  - Opérations sur les vecteurs
- Solution
  - **Instructions SIMD** spécifique :
    - SSE, MMX, 3Dnow, altivec
  - Plusieurs données en entrée

    1 ou plusieurs données en sortie

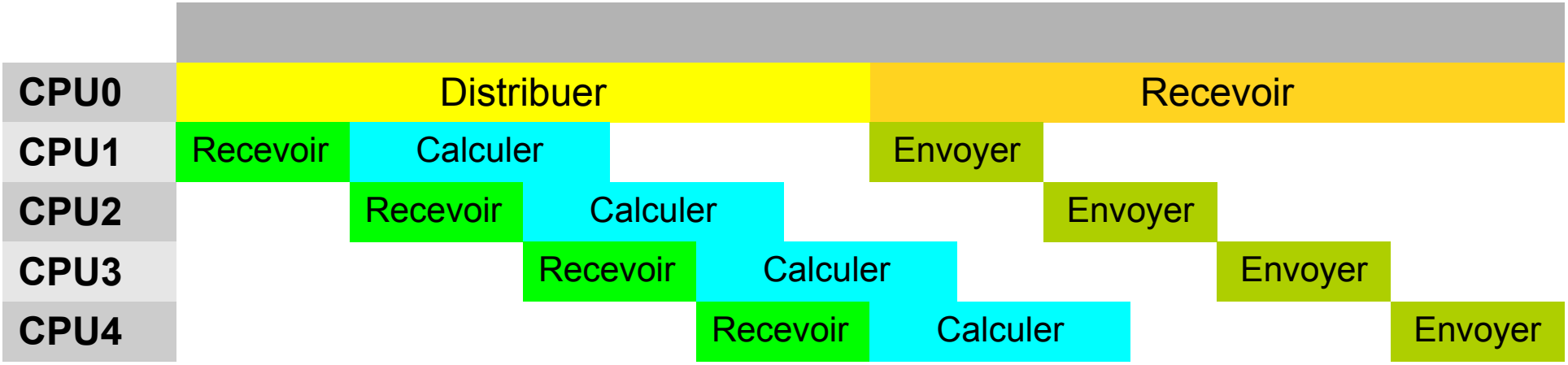
Opération Vectorielle vs Scalaire

- Constatation
- Solution
  - **Instructions SIMD** spécifique :
    - SSE, MMX, 3Dnow, altivec
  - Plusieurs données en entrée

    1 ou plusieurs données en sortie
- Utilisation
  - « A la main » en assembleur ou types prédéfinies
  - Vectorisation automatique par le compilateur

Notion de pipeline

- Pipeline
  - Ensemble de ressources matérielles synchronisées



- Speed-up / efficacité
  - Gain obtenue par la parallélisation

Notion de pipeline

- Pipeline
  - Ensemble de ressources matérielles synchronisées

- Speed-up / efficacité
  - Gain obtenue par la parallélisation
  - Idéal = P si P processeurs

Exercices : répartition sur P processeurs

- Sur le tri simple
  - Répartition de la recherche de minimum

- Tri par segmentation
  - Répartition des sous-appels aux tris

Conclusion

- Division en tâches élémentaires
  - Très simples
  - Pouvant être répéter de nombreuses fois

- Éviter les besoins en synchronisation
  - Attente de résultats
  - Mémoire