# Synthèse des outils de parallélisation

Matthieu Pérotin

LI Tours

# Plan (1/2)

- Introduction
  - Pourquoi ?
  - Rappels architecturaux
- Paralléliser
  - Modèles de parallélisation
  - Algorithmes parallèles
  - Évaluation de la qualité

# Plan (2/2)

- Exemple avancé
  - Procédures par Séparation et Évaluation
- Conclusion

# Introduction

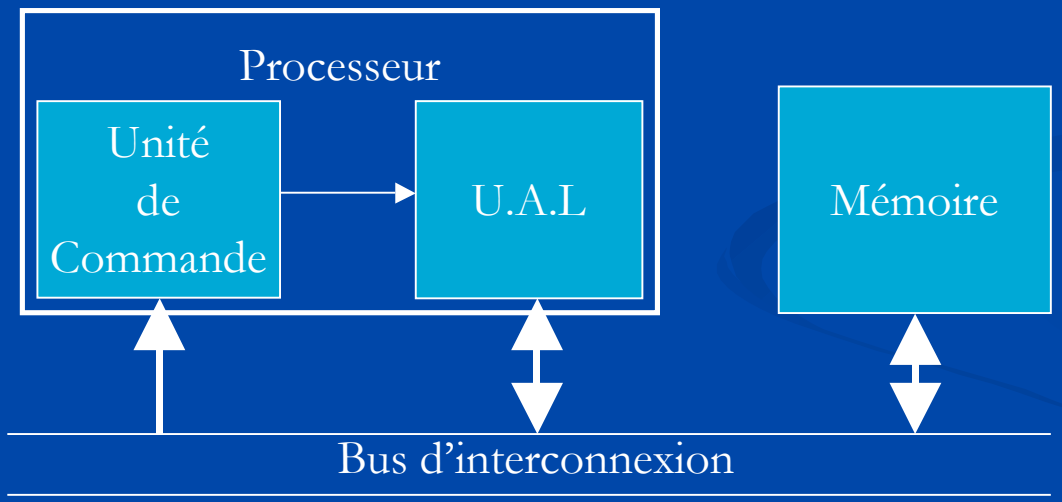
- Pourquoi Paralléliser ?
  - Gagner du temps
  - La modélisation parallèle est parfois évidente
- Comment paralléliser ?
  - Question non triviale
  - Propre à chaque architecture
  - Propre à chaque problème

# Introduction

- Rappels architecturaux
  - Flynn (1972) propose une classification basée sur:
    - Les capacités des machines à exécuter plusieurs instructions en parallèle
    - Les capacités à lire plusieurs sources d'informations simultanément

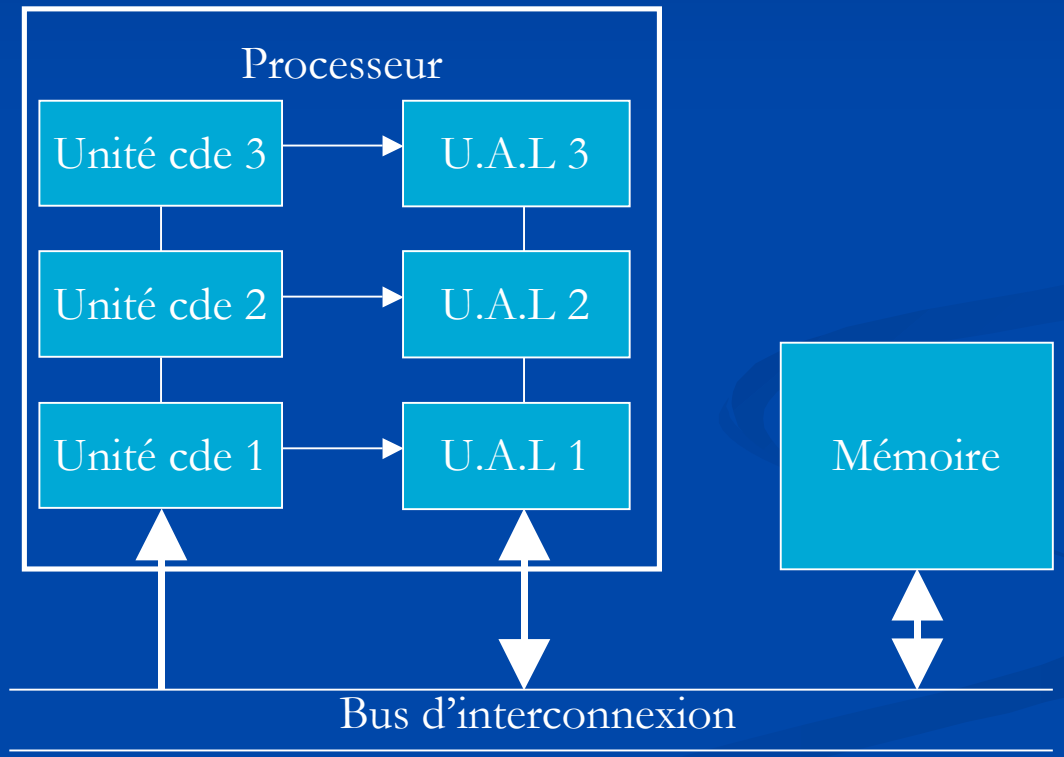
# Introduction

- 4 catégories:
  - Single Instruction Single Data



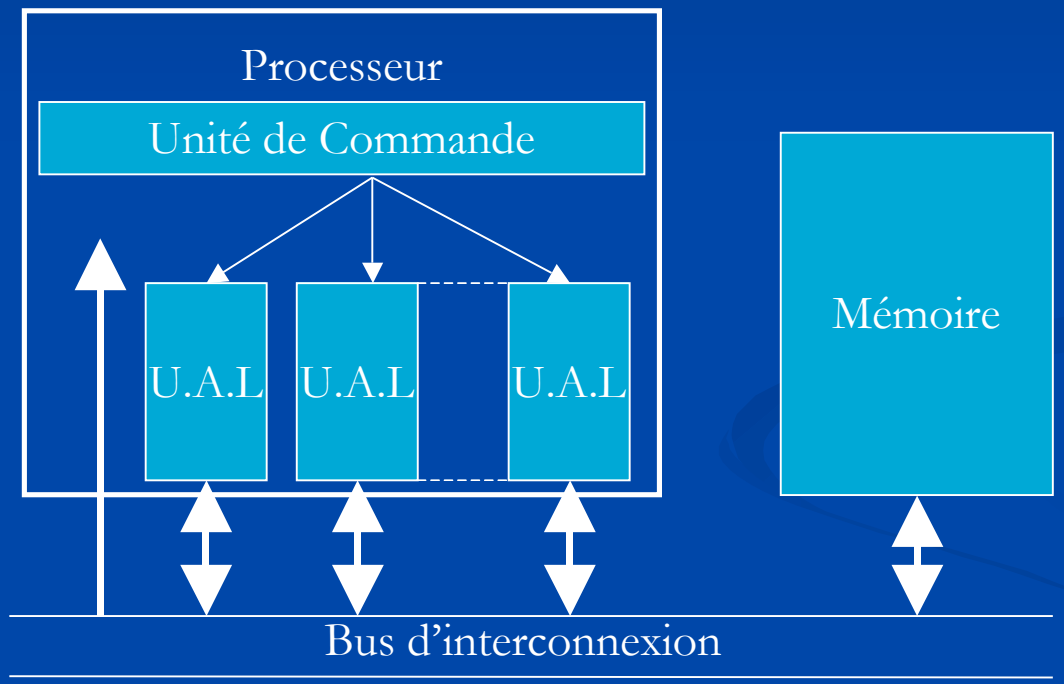
# Introduction

- Multiple Instructions Single Data



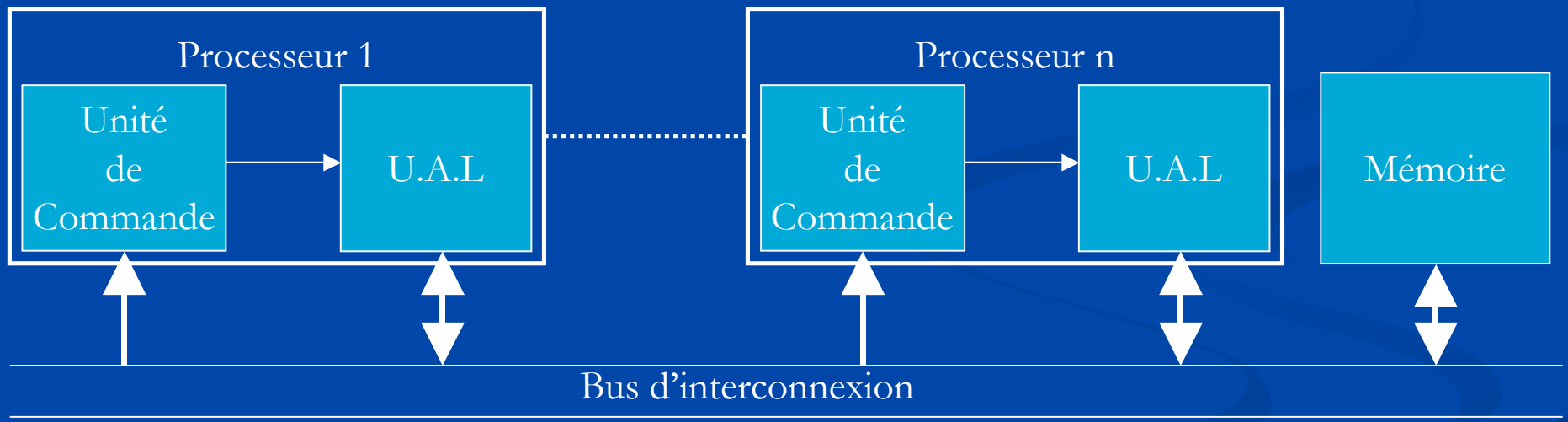
# Introduction

- Single Instruction Multiple Data



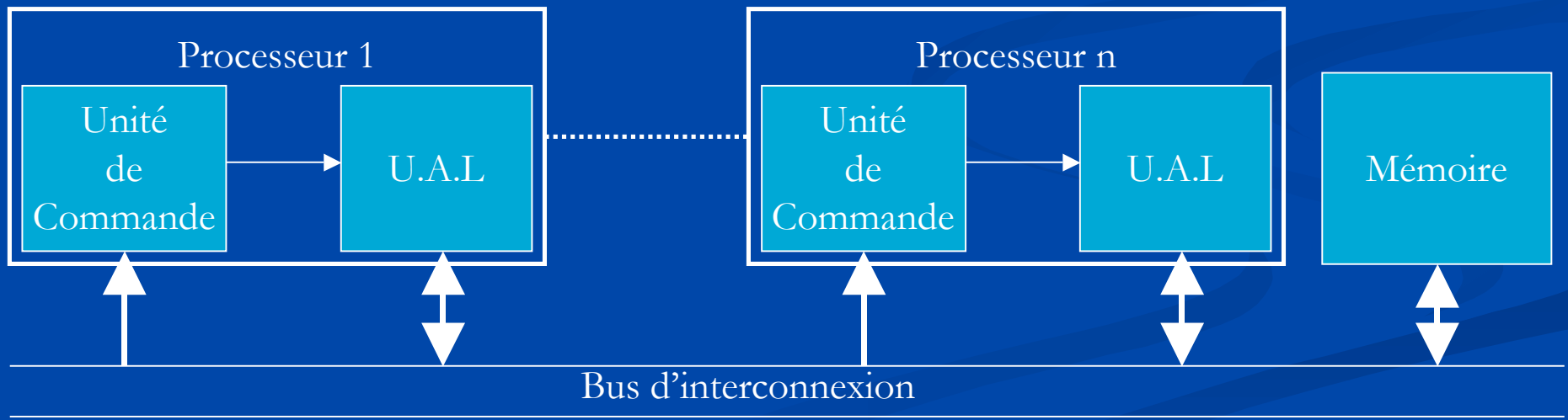
# Introduction

- Multiple Instruction Multiple Data



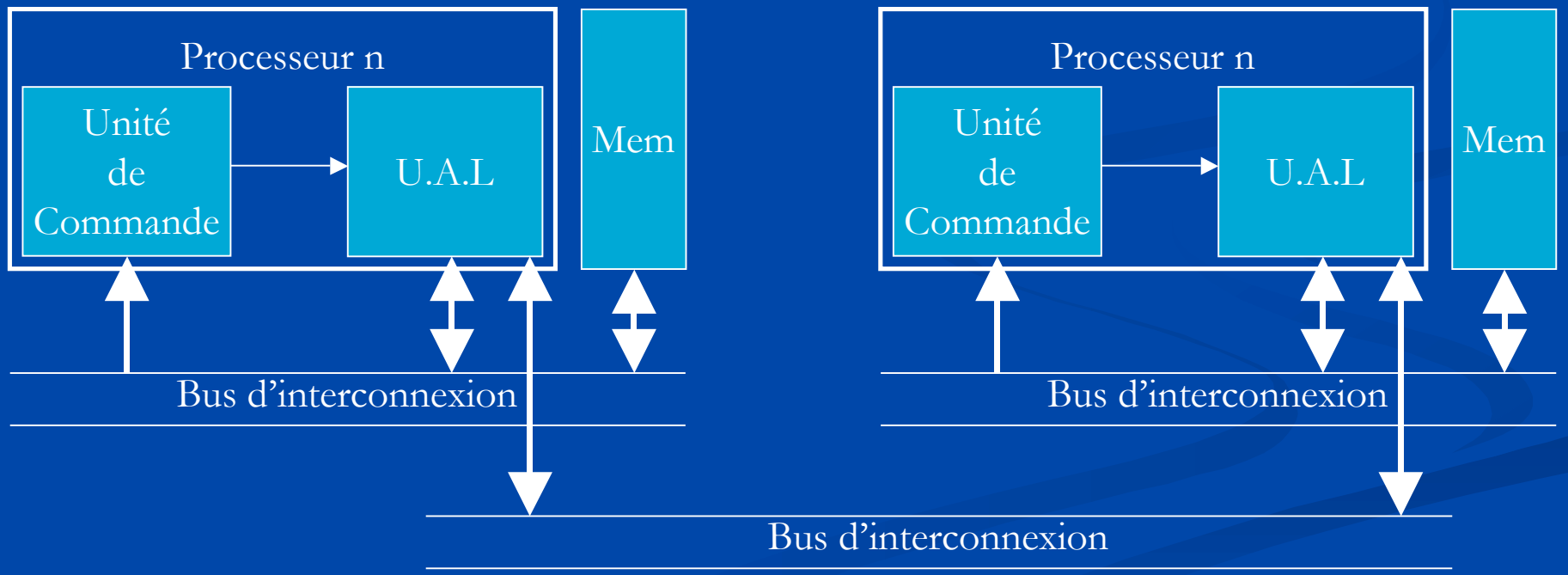
# Introduction

- Classification étendue par Johnson (1988)
  - Tient compte des types d'accès mémoire
  - Uniform Memory Access (UMA)



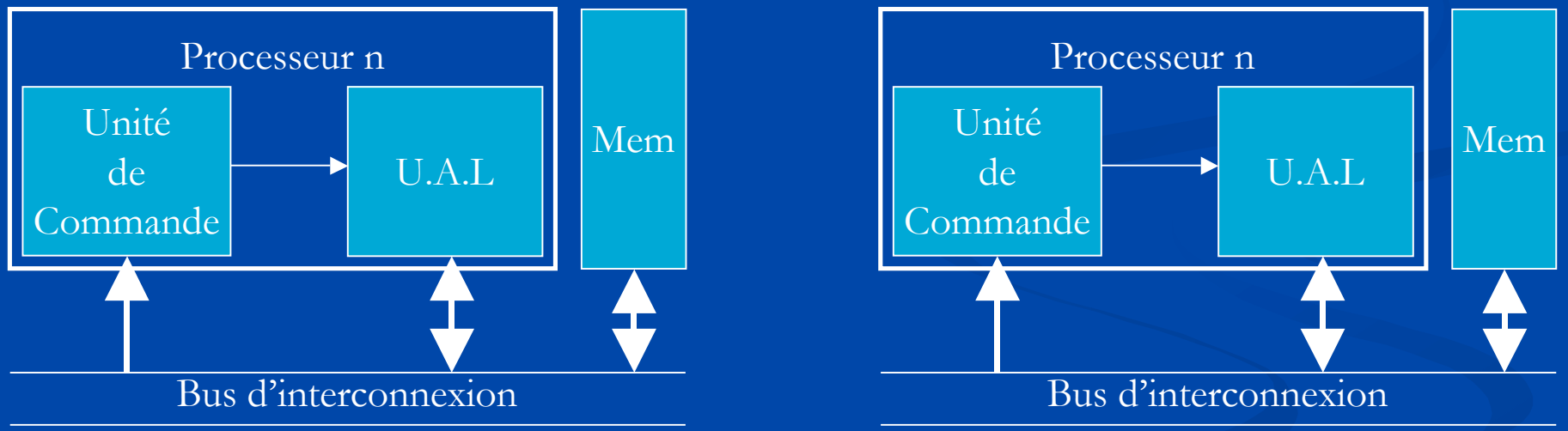
# Introduction

- Non Uniform Memory Access (NUMA)



# Introduction

- No Remote Memory Access (NORMA)



# Introduction

- Il est important de tenir compte de l'architecture matérielle de l'ordinateur qui va exécuter le code
- En particulier cette architecture détermine le grain de l'application
- Le grain est la taille des unités de calcul
- On peut le définir comme le rapport des temps de communication sur le temps d'exécution

# Intoduction

- Les architectures MIMD/NORMA sont peu adaptées aux applications à grain fin
- Une architecture MIMD/UMA l'est beaucoup plus

# Plan

- Introduction
  - Pourquoi ?
  - Rappels architecturaux
- Paralléliser
  - Modèles de parallélisation
  - Algorithmes parallèles
  - Évaluation de la qualité

# Paralléliser

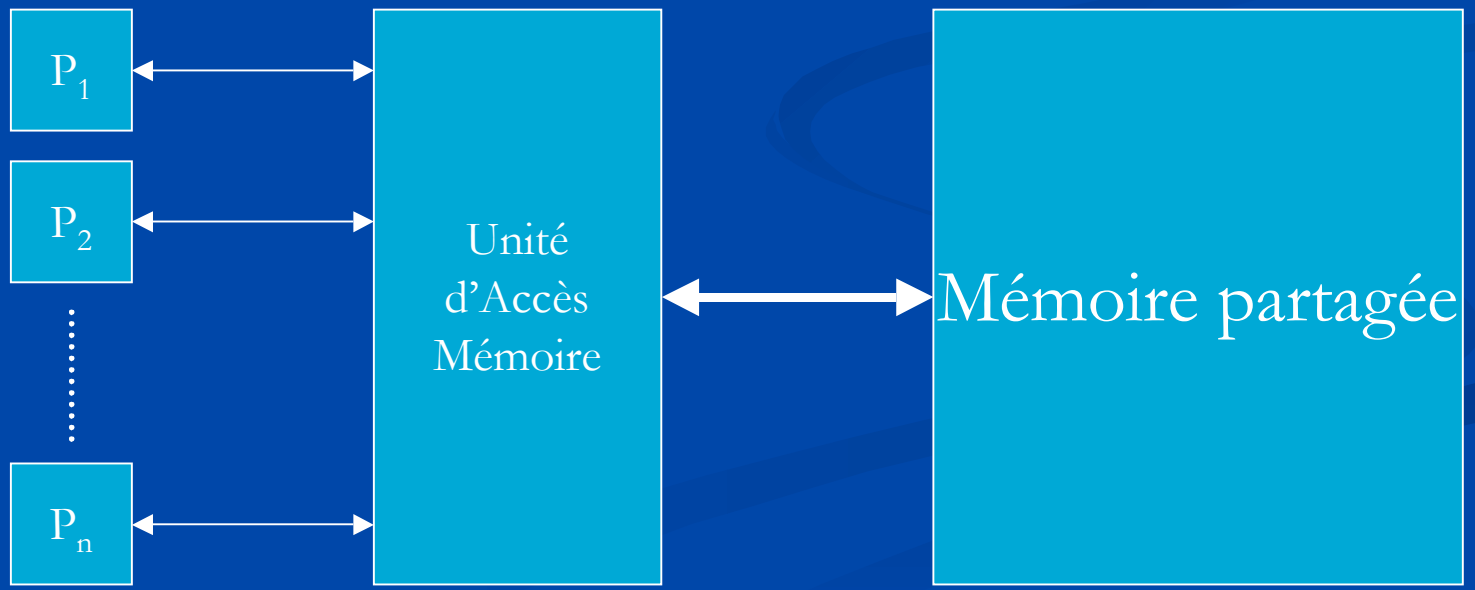
- Démarche d'étude:
  - Quels sont les modèles existants ?
  - Qu'est-ce qu'un algorithme parallèle ?
  - Comment évaluer sa qualité ?

# Paralléliser – Modèles

- Modèles de parallélisme
  - Un modèle de parallélisme est une description abstraite d'une machine parallèle.
  - Deux grands modèles
    - Mémoire partagée
    - Réseau d'interconnexion

# Paralléliser - Modèles

- Mémoire partagée
  - Les processeurs accèdent tous de manière identique à une zone de mémoire unique



# Parallélisme - Modèles

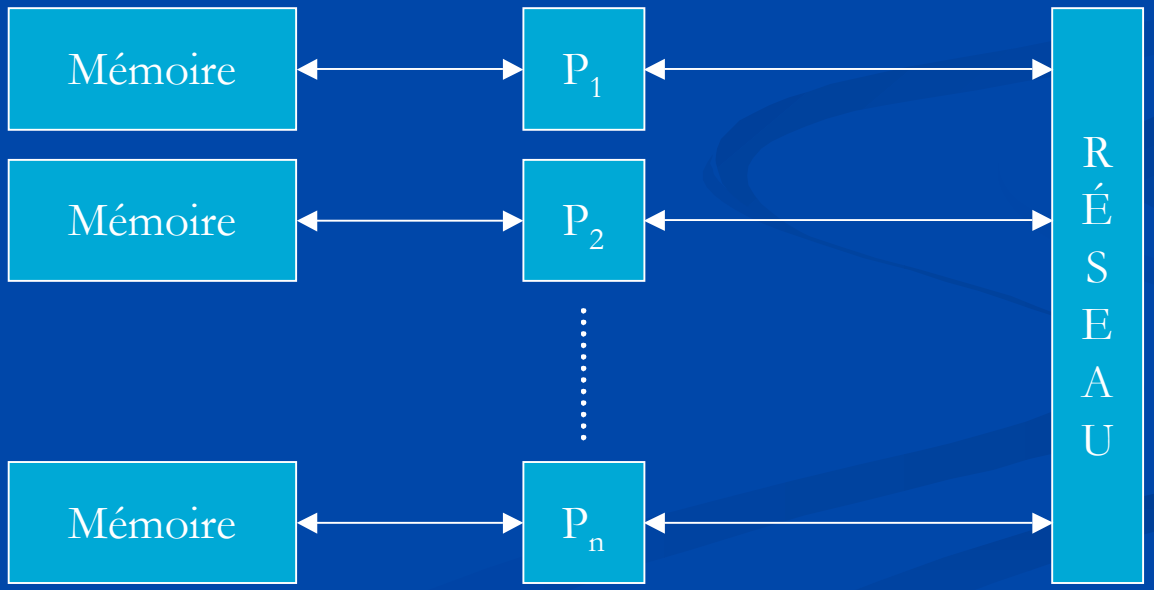
- Dans ce modèle
  - Les processeurs utilisent la mémoire pour communiquer les uns avec les autres
  - Un mécanisme d'exclusion mutuelle est mis en place
    - Plusieurs processeurs peuvent lire la même adresse
    - Un seul peut y écrire à un moment donné

# Parallélisme - Modèle

- Mémoire partagée
  - Existence physique
    - Systèmes SMP
    - Systèmes UMA
  - Existence logique
    - SSI
  - Attention: ceci ne remet pas en cause la classification matérielle !

# Paralléliser - Modèles

- Réseau
  - Les processeurs disposent chacun d'une mémoire qui leur est propre, mais sont reliés par un réseau



# Parallélisation - Modèles

- Dans ce modèle
  - Les processeurs communiquent par échange de message
  - Mécanisme d'exclusion mutuelle (ou pas) au niveau de l'accès réseau

# Paralléliser - Modèles

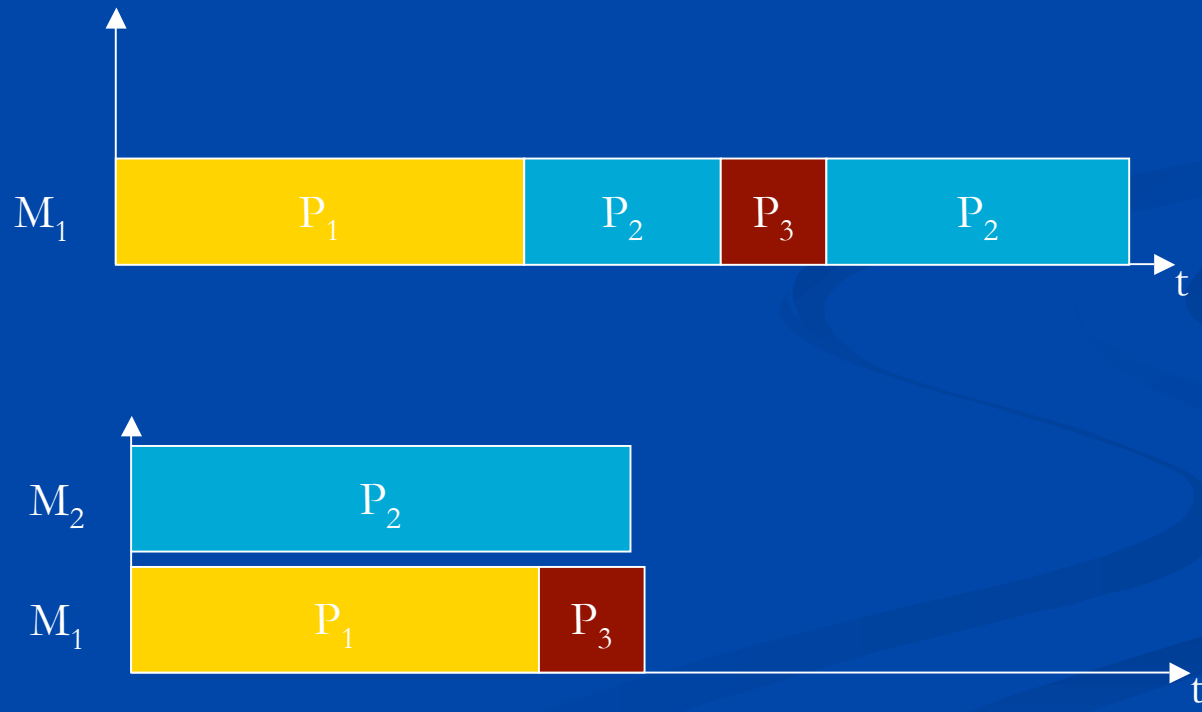
- Réseau d'interconnexion
  - Existence physique
    - Salle de TP
  - Existence logique
    - MPI, PVM…

# Parallélisation - Algorithmes

- Qu'est-ce qu'un algorithme parallèle ?
  - « Un algorithme parallèle est une méthode de résolution d'un problème dans laquelle le problème est découpé en sous problèmes de taille inférieure qui sont résolus de façon simultanée » [Akl00]
  - **Attention** : la simultanéité est ici un concept logique
    - Ex: application multi-threadée s'exécutant sur un unique processeur

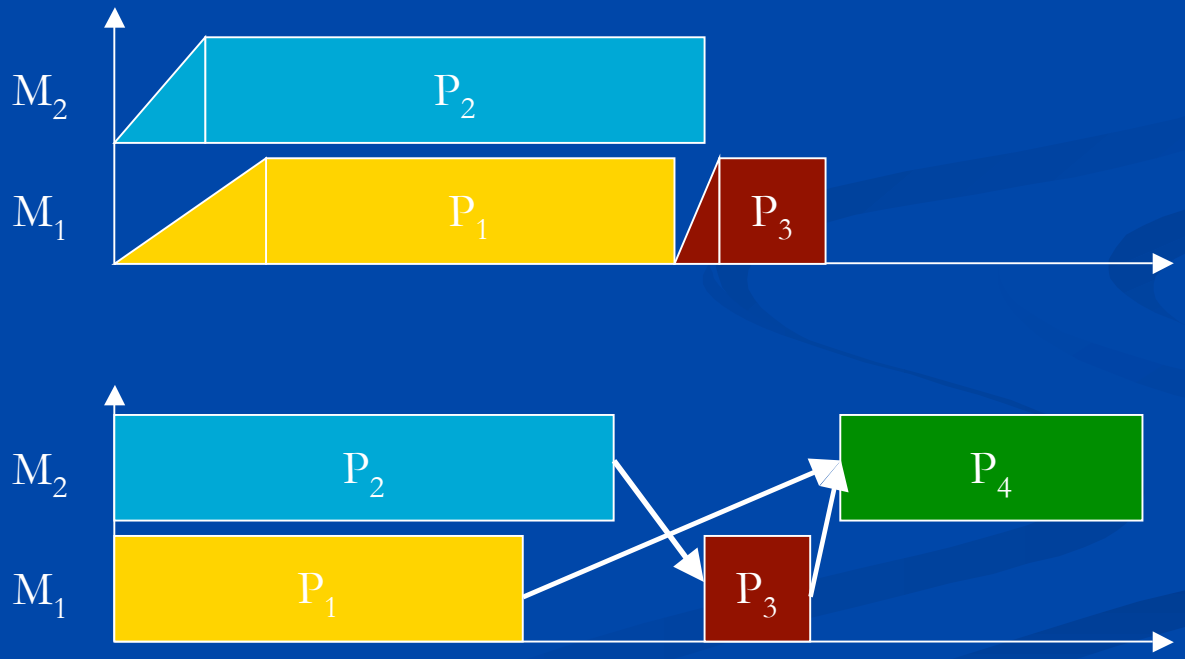
# Paralléliser - Algorithmes

- Formalisme
  - Graphique: Diagramme de Gantt



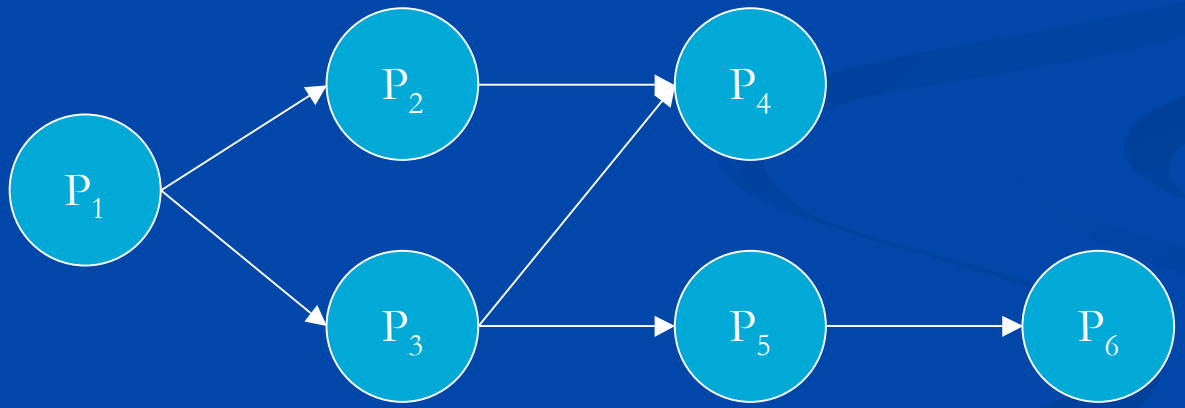
# Paralléliser - Algorithmes

- Formalisme
  - Graphique: diagramme de Gantt



# Paralléliser - Algorithmes

- Formalisme
  - Graphique : diagramme de précédence



# Parallélisme - Algorithmes

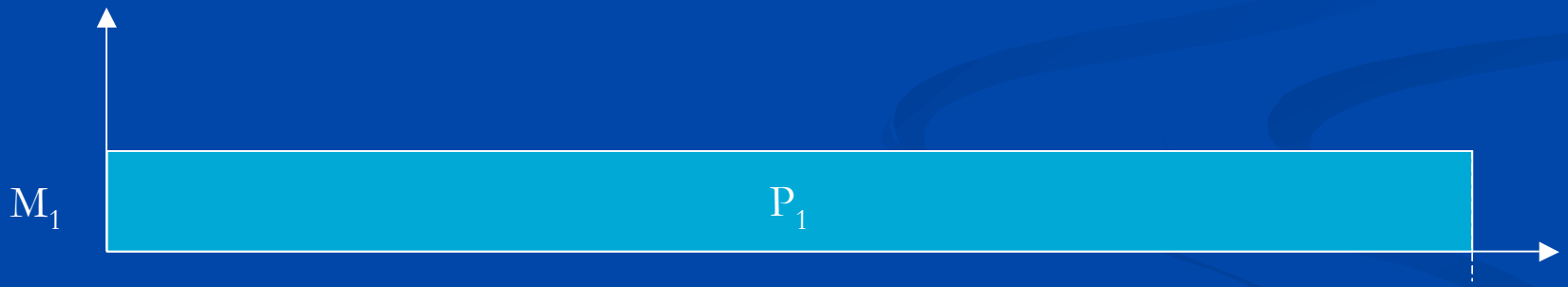
- Formalisme théorique
  - On suppose disposer de nouvelles primitives en pseudo code
    - **Démarrer(tache(arguments))**
      Lance la tâche en parallèle, de manière non bloquante
    - **Envoyer(P, message) et Recevoir(P, message)**
      Pour échanger des messages avec le processeur P
  - L'implémentation effective de ces primitives dépend du modèle et de l'architecture

# Paralléliser - Algorithmes

- Exemple
  - On doit effectuer un mailing
  - 1000 courriers à envoyer
    - Plier une feuille
    - Mettre dans une enveloppe
    - Adresser
    - Timbrer

# Paralléliser - Algorithmes

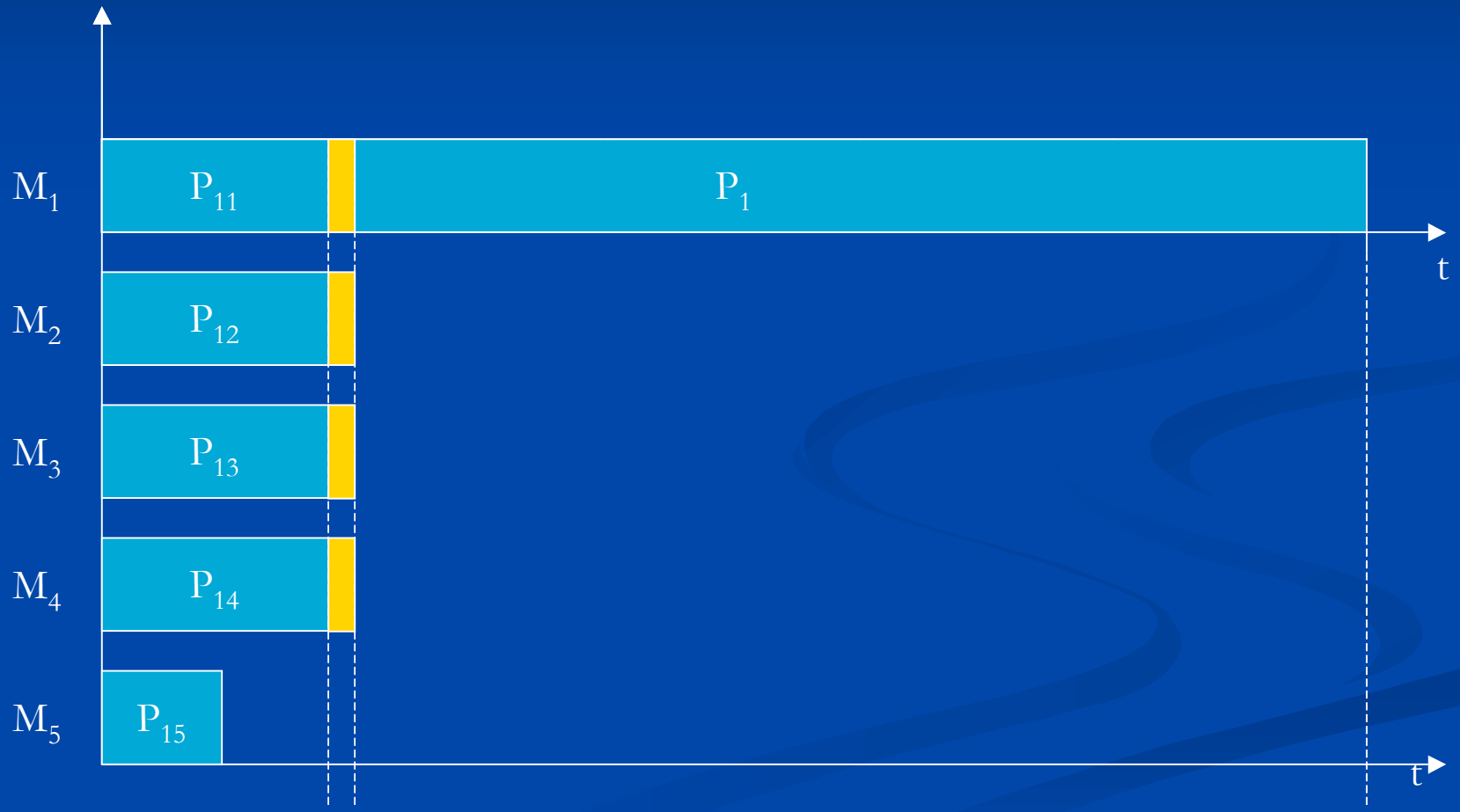
- Première version, séquentielle
  - **Tant que** les 1000 enveloppes ne sont pas faites
    - Traiter l'enveloppe



# Paralléliser - Algorithmes

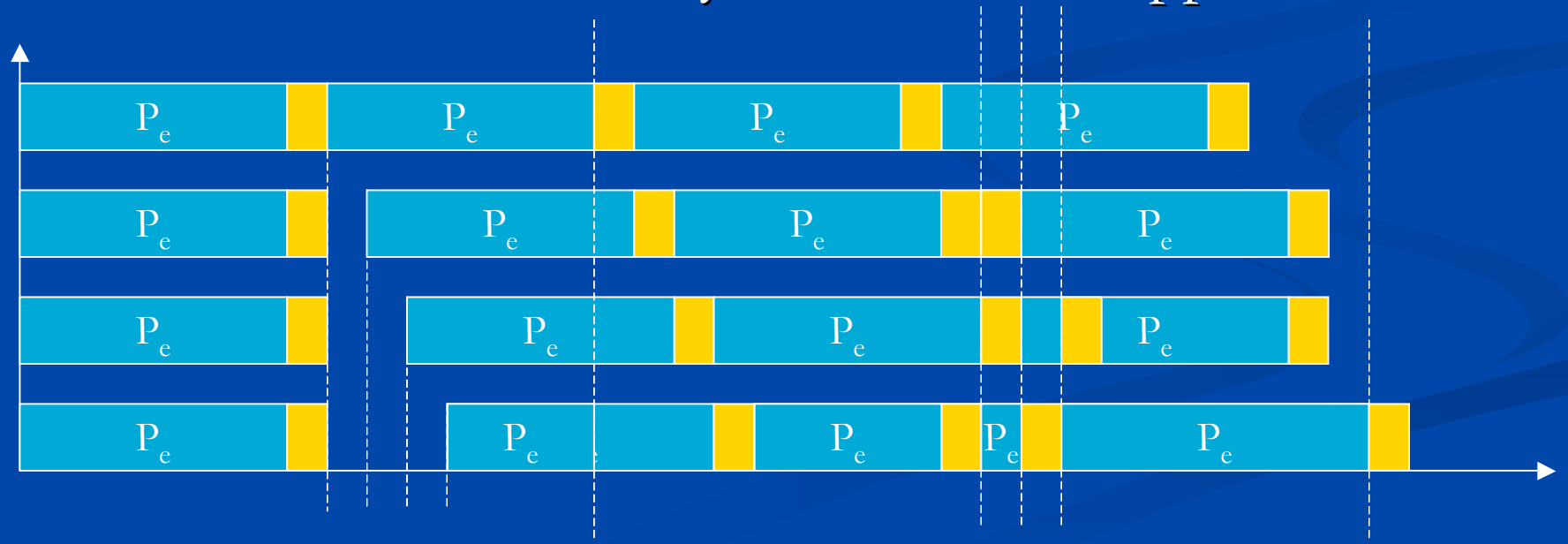
- Deuxième version, parallèle
  - On invite 4 amis à participer, chacun fait 200 enveloppes
- Pourquoi notre algorithme parallèle n'est-il pas optimal ?

# Paralléliser - Algorithmes



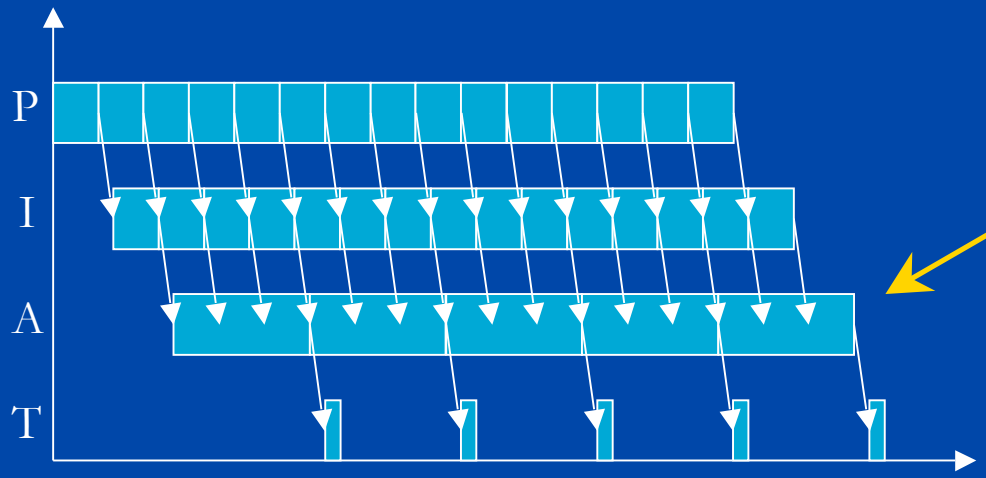
# Paralléliser - Algorithmes

- Une nouvelle version de l'algorithme serait
  - Tant que 1000 enveloppes n'ont pas été faites, faire des enveloppes
  - Induit un coût de synchronisation supplémentaire…



# Paralléliser - Algorithmes

- On peut aussi décider que chacun a une tâche bien définie
  - L'un timbre, l'autre écrit les adresses…
  - Sans se préoccuper des autres

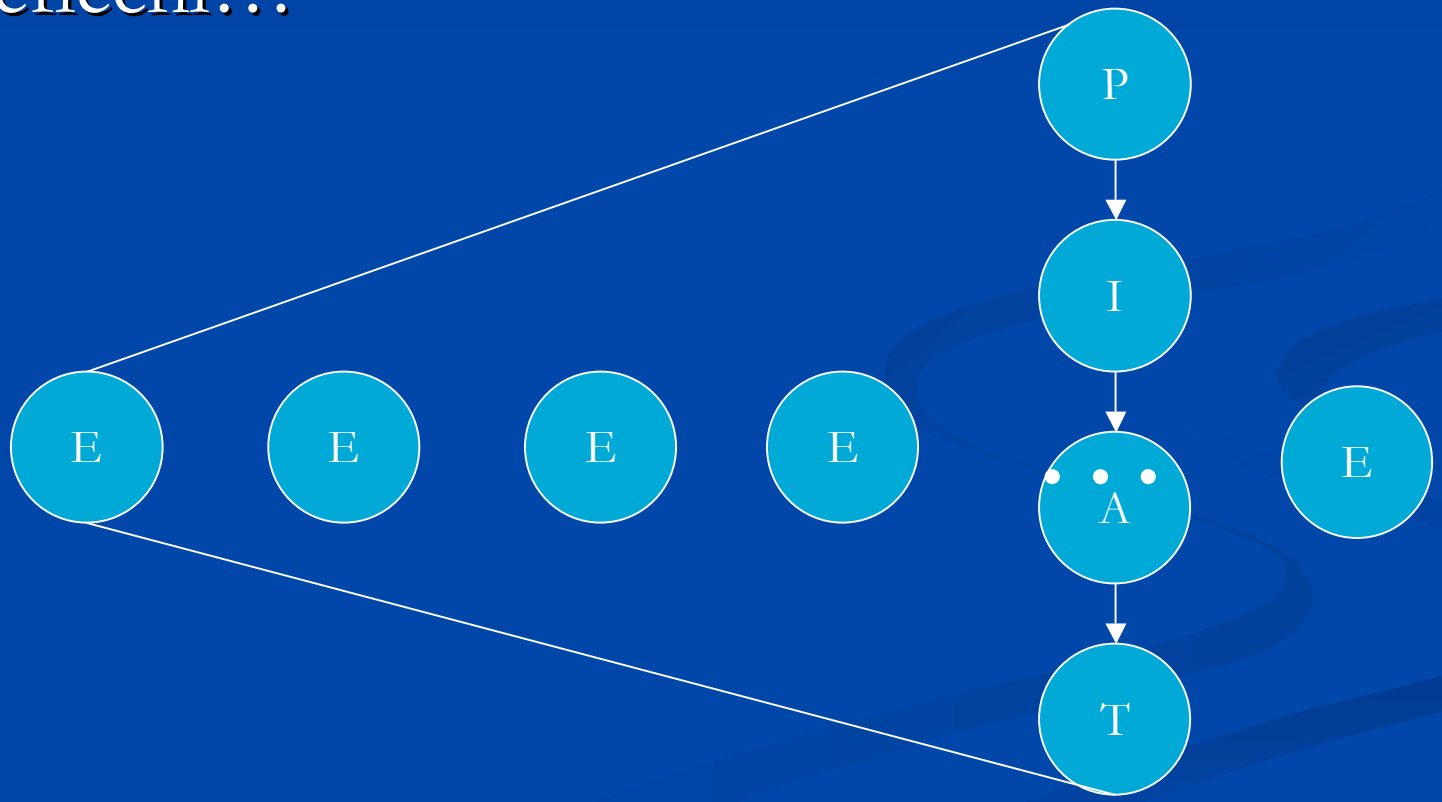



Un stock apparaît en entrée

Pour chaque enveloppe traitée, 3 arrivent…

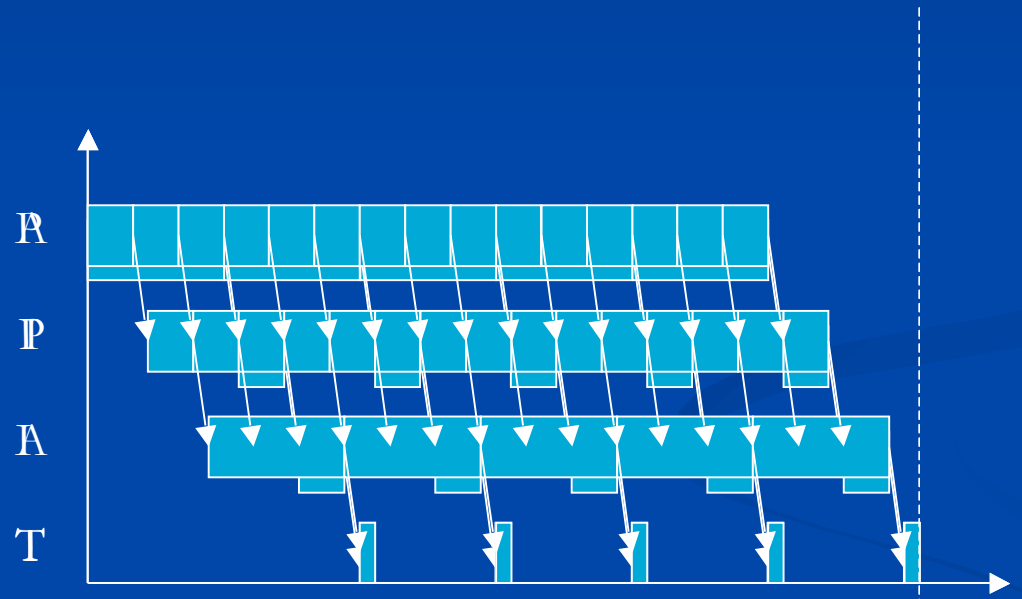
# Paralléliser - Algorithmes

- Sur cet exemple simple, nous n'avons pas réfléchi…



# Paralléliser - Algorithme

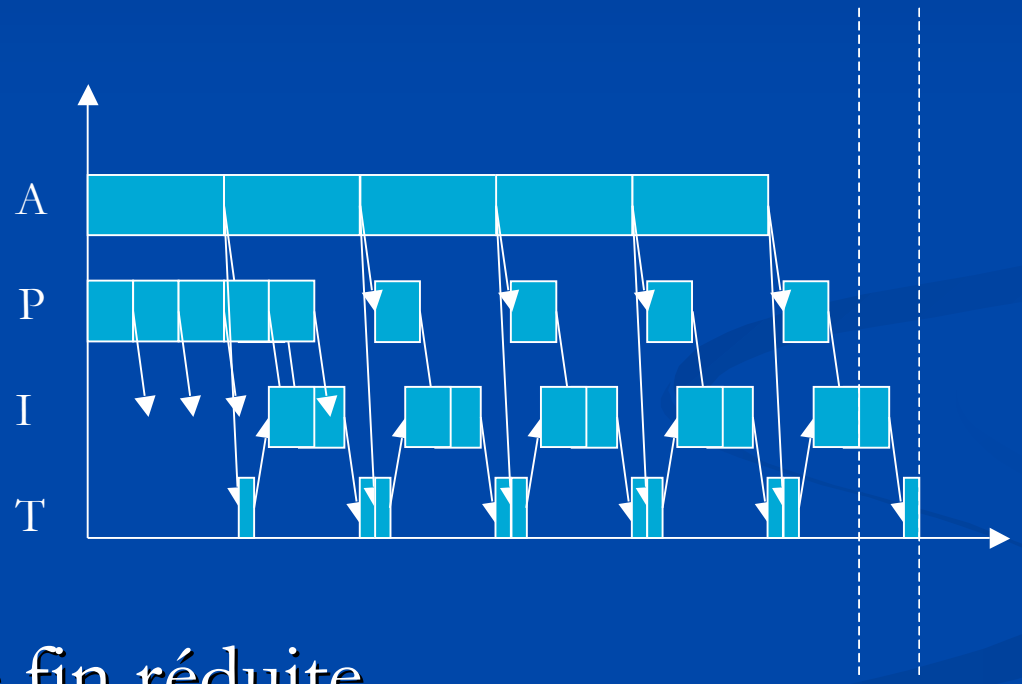
- Nous allons faire les opérations dans un autre ordre



- Plus de stock et la date de fin reste inchangée

# Paralléliser - Algorithmes

- Autre possibilité, je m'autorise à avoir du stock




- Date de fin réduite

# Paralléliser - Qualité

- Disposer de critères d'évaluation est indispensable
- 3 principaux critères
  - Le temps d'exécution total (Makespan, $C_{max}$)
  - Le nombre de processeurs
  - Le coût

# Parallélisme - Qualité

- Temps total d'exécution
  - Critère « intuitif » : moins un algorithme met de temps à s'exécuter, meilleur est cet algorithme
  - Dépend généralement de la taille des données, t(n)
  - Problème : évaluer le temps d'exécution d'un algorithme est un problème ouvert

# Parallélisation - Qualité

- Nombre de processeurs p(n)
  - Pour des raisons économiques, il est important de prendre en compte le nombre de processeurs que nécessite un algorithme
  - Quand le nombre de processeurs augmente, le $C_{max}$ diminue (en général)

# Parallélisation - Qualité

- Plusieurs coûts possibles
  - Coût des communications
  - Coût de stockage (espace mémoire)
  - Coût d'inactivité des machines

# Parallélisation - Qualité

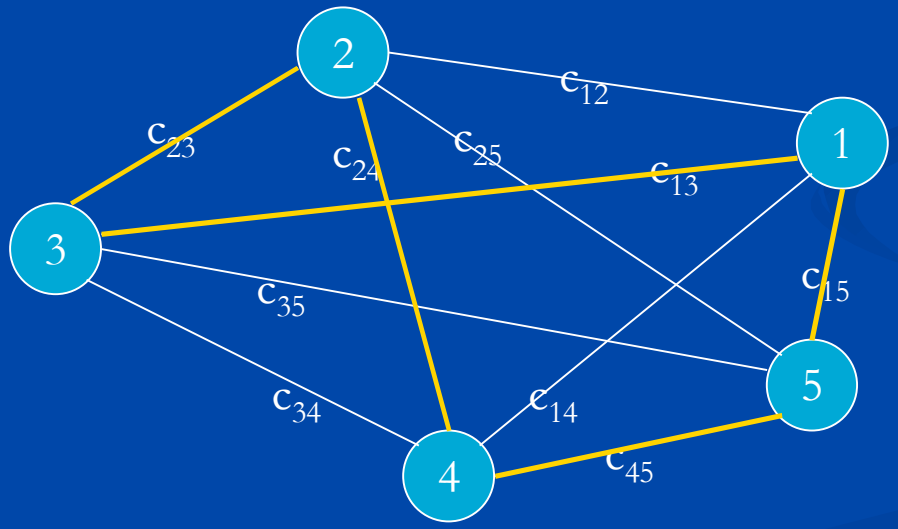
- Les critères d'évaluation sont souvent contradictoires
- Nous sommes bel et bien face à des problèmes d'optimisation multi-critères
- L'intervention d'un décideur est essentielle
  - On peut proposer un outil d'aide à la décision, mais pas un outil de décision automatique

# Plan

- Exemple avancé
  - Procédures par Séparation et Évaluation
- Conclusion

# Les PSE

- Principe en exemple
- Le problème du voyageur de commerce (PVC)



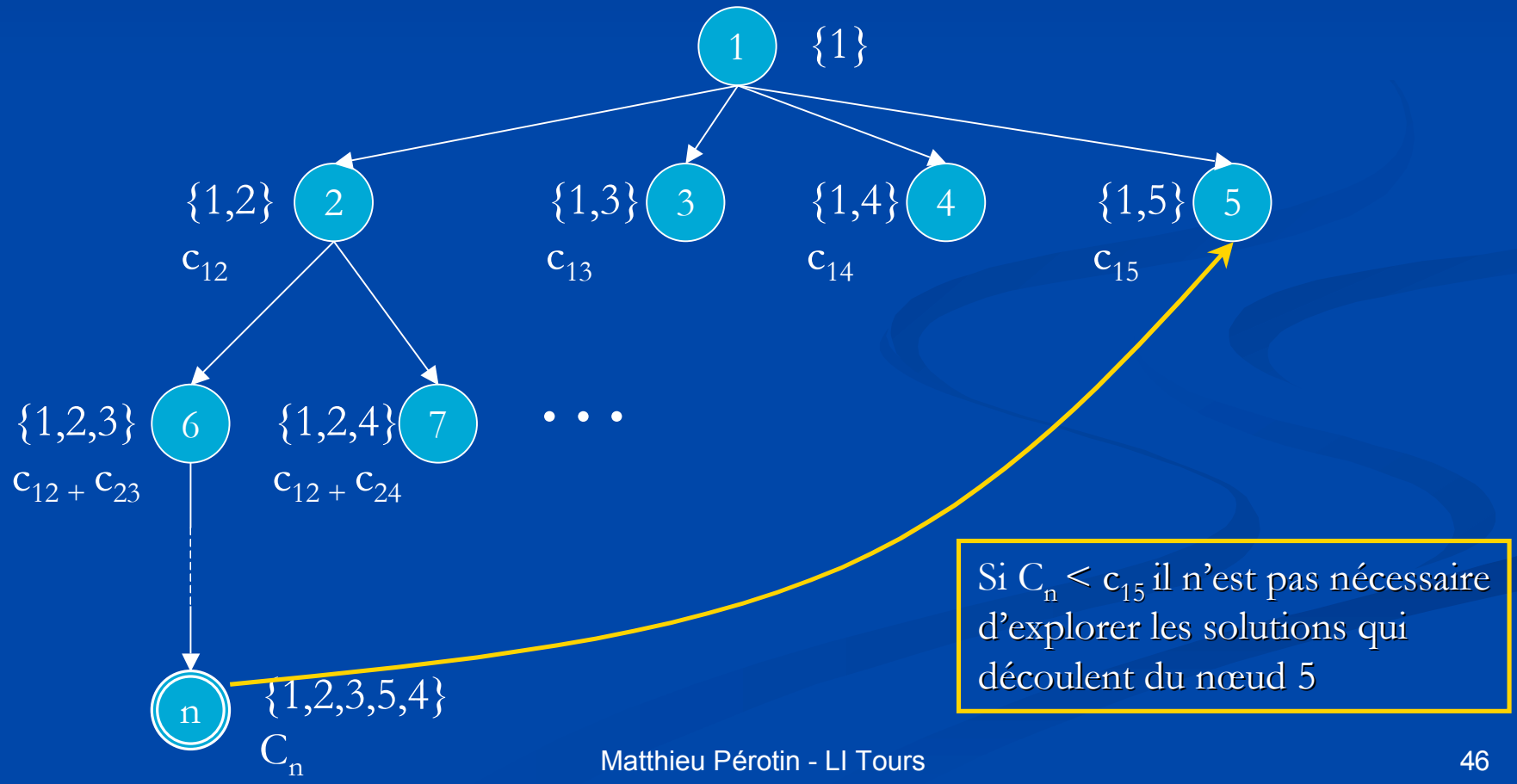

{1,5,4,2,3}

# Les PSE

- Le PVC est un problème NP-Difficile
  - Pas d'autre choix que d'essayer toutes les possibilités si l'on veut trouver la solution optimale
  - (n-1)! possibilités

# Les PSE

- Construction de notre PSE



# Les PSE

- En fait une PSE permet une énumération implicite, et pas trop bête
- On n'explore que les solutions à même de donner une solution meilleure que celles que l'on connaît déjà
- Deux règles de coupe
  - Tests de borne inférieure
  - Tests de dominance

# Les PSE

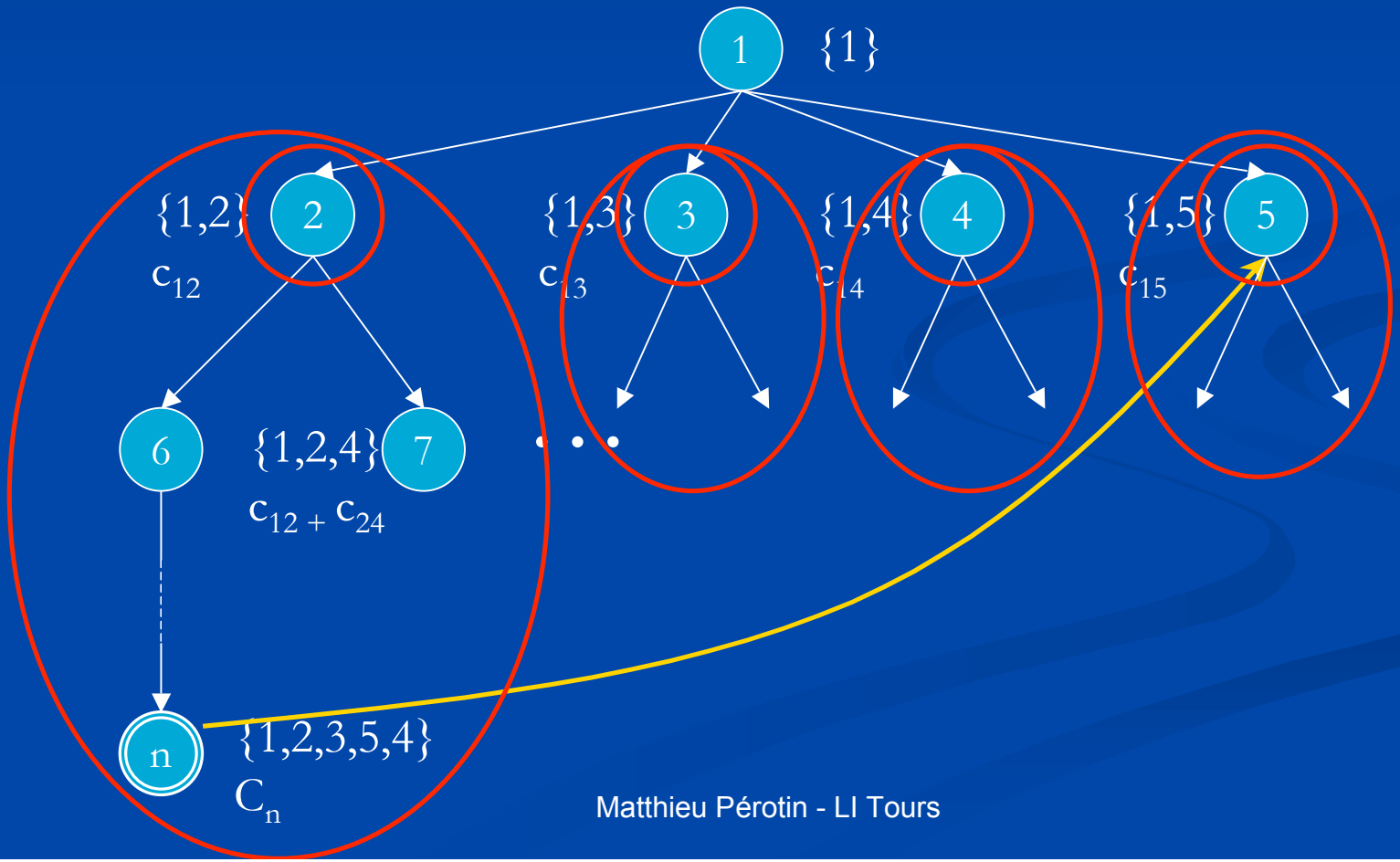
- Exemple de tests de dominance
  - Soit les deux solutions partielles
    - {1,2,3,4} de coût $c_{12}+c_{23}+c_{34} = z_1$
    - {1,3,2,4} de coût $c_{13}+c_{32}+c_{24} = z_2$
  - Si $z_1 = z_2$ alors on dit que $z_1$ domine $z_2$
  - On explorera qu'un des deux sous arbres au départ des noeuds

# Paralléliser une PSE

- L'arbre d'une PSE peut être vu comme un diagramme de précédence
- Le parallélisme apparaît donc de manière limpide

# Paralléliser une PSE

- Par exemple

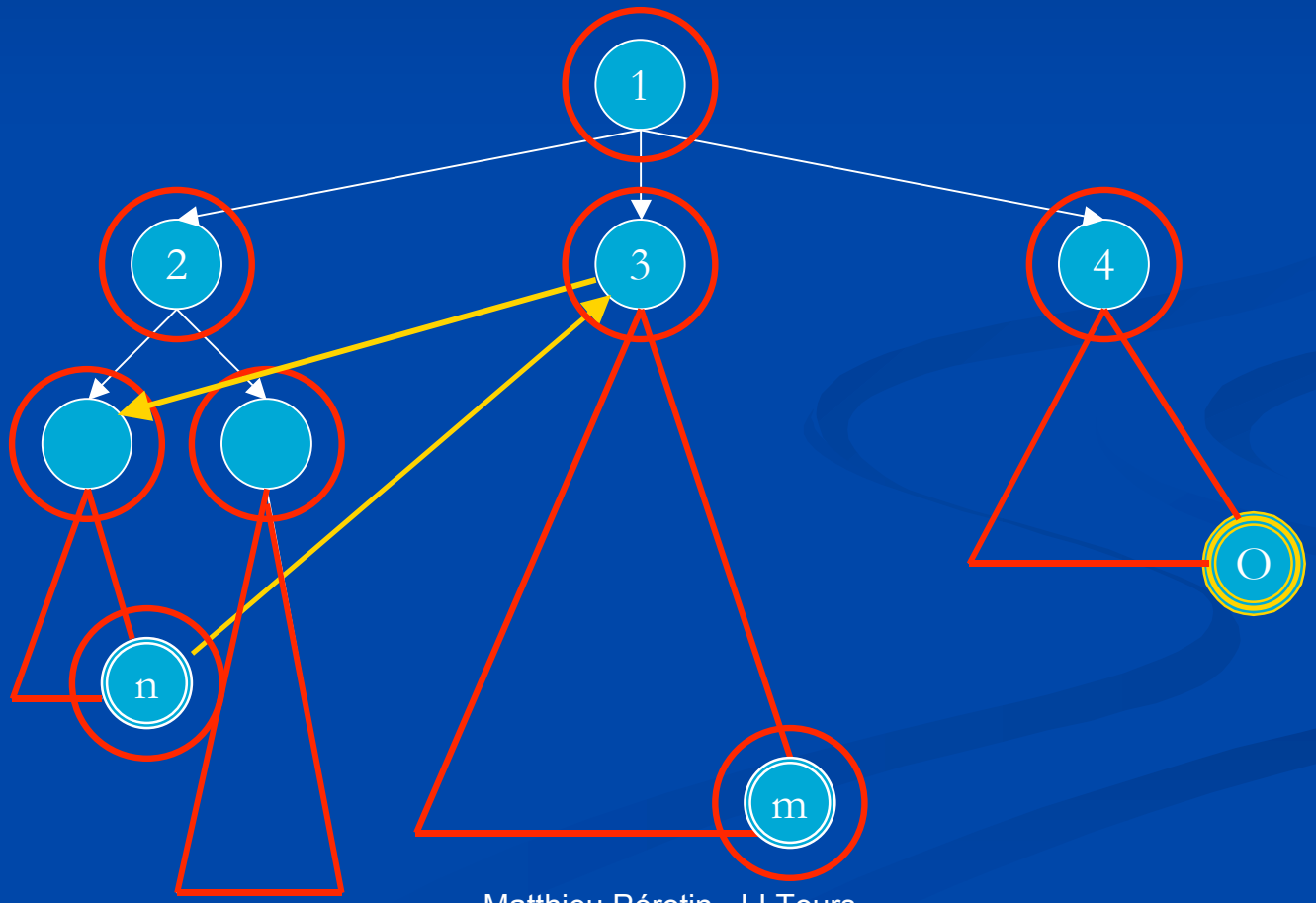


# Paralléliser une PSE

- Paralléliser une PSE est un problème intéressant à plus d'un titre
  - Les PSE sont utiles
  - Des phénomènes dignes d'intérêt apparaissent
    - Anomalie d'accélération
    - Anomalie de décélération

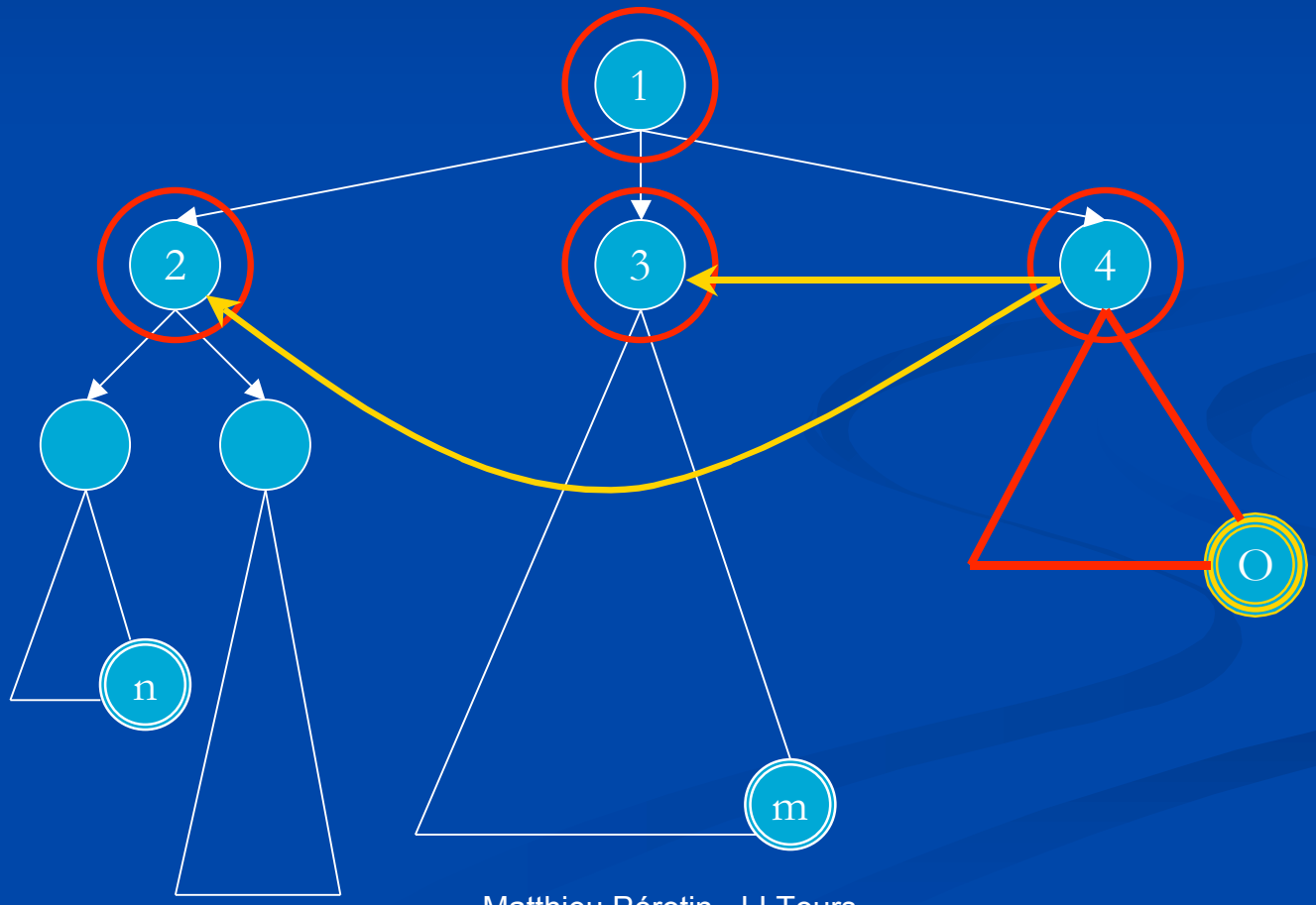
# Paralléliser une PSE

- Anomalie de décélération



# Paralléliser une PSE

- Anomalie d'accélération



# Paralléliser une PSE

- Certains critères existent pour éviter les anomalies de détérioration, et favoriser celles d'accélération

# Conclusion

- Paralléliser une application est une opération compliquée
- Pas de recette magique
- Il est nécessaire de **bien** réfléchir
  - Sur quelle architecture matérielle vais-je exécuter mon application ?
  - Quels sont mes critères d'évaluation ?
  - Ma solution logicielle est-elle réalisable ?

# Conclusion

- Il existe des outils théoriques qui permettent une modélisation fine
  - Théorie de l'ordonnancement mono et multi critères
  - Recherche opérationnelle
  - Algorithmique

- **Plus de processeurs n'implique pas nécessairement plus de vitesse**