Athapascan-1 sur grappe de PCs hétérogènes

Spécialisation de l'ordonnancement

Rémi Revire

ID-IMAG Équipe Apache

Grappes 2001

Laboratoire Informatique et Distribution

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Institut National Polytechnique de Grenoble

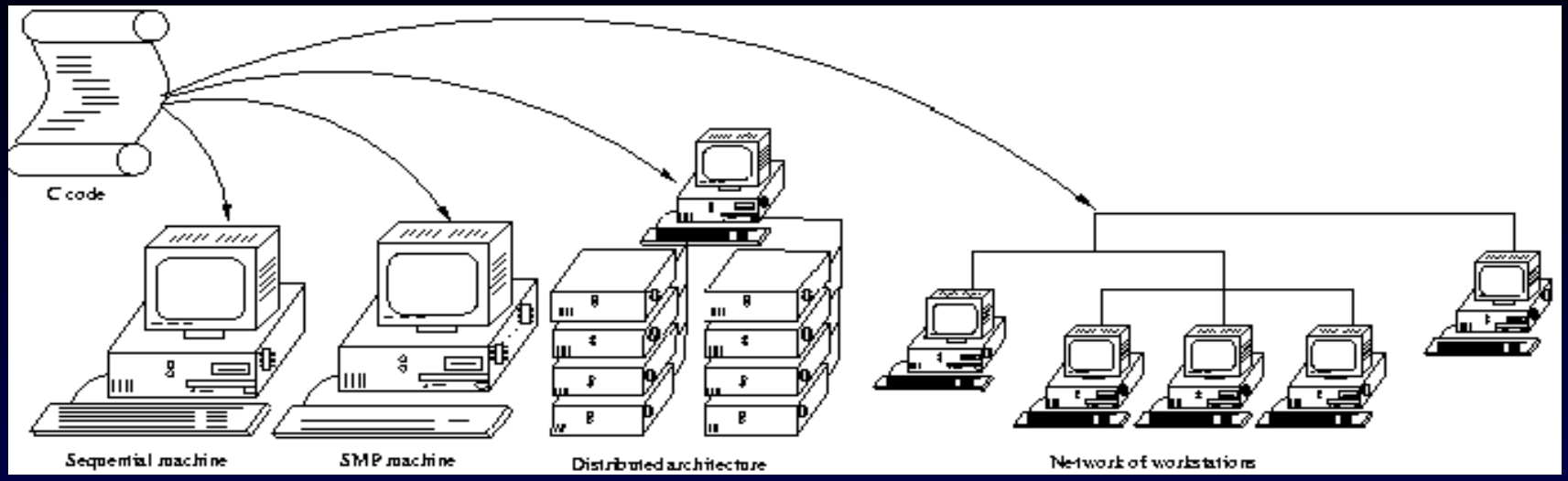
INRIA INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

GRENOBLE 1 UNIVERSITÉ JOSEPH FOURIER SCIENCES,TECHNOLOGIE,MÉDECINE

Plan

- Présentation d ’Athapascan1
  - Choix fondateur : calcul du flot de données
  - Interface de programmation
  - Exemple
- Couplage d ’ordonnancement
  - Les ordonnancements spécialisés en Athapascan1
  - Spécialisation pour grappe de PC hétérogènes
    - Couplage dynamique d ’ordonnancement
  - Couplage Ordo Séquentiel / Ordo glouton
  - Couplage Ordo glouton / Ordo statique
- Conclusions

Problématique



- Langage objet, bcp de parallélisme: comment le répartir ?
  - identifier le parallélisme
  - répartition efficace :
    - replier le parallélisme : granularité
    - minimiser le temps d ’exécution : tirer parti de la localité des données
- Choix fondateurs pour Athapascan:
  - utiliser une description (prédiction) de l ’exécution pour calculer l ’ordo
  - description = flot de données macroscopique

Calcul du flot de données par interprétation
Le modèle de programmation Athapascan1

① *Granularité explicite*

- grain de donnée = **objet partagé**

  Shared< int **>** n ; … **Shared<** array **>** tab ;
- grain de calcul = **tâche** (appel de procédure)

  Fork fn(n) ;

② *Synchronisation* entre les tâches *implicite*

➔ Indication des actions des tâches sur les objets partagés

par typage explicite Î **{** lecture **r**, écriture **w**, modification **r_w**, accumulation **cw**}

void fn (Shared_**r** < int > a) { … }

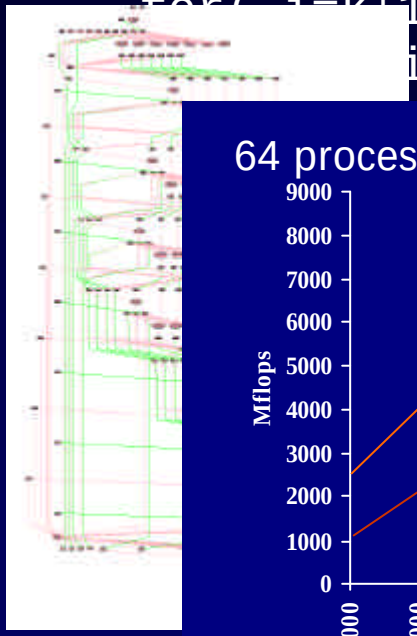
③ *Sémantique naturelle,* de type *séquentielle*

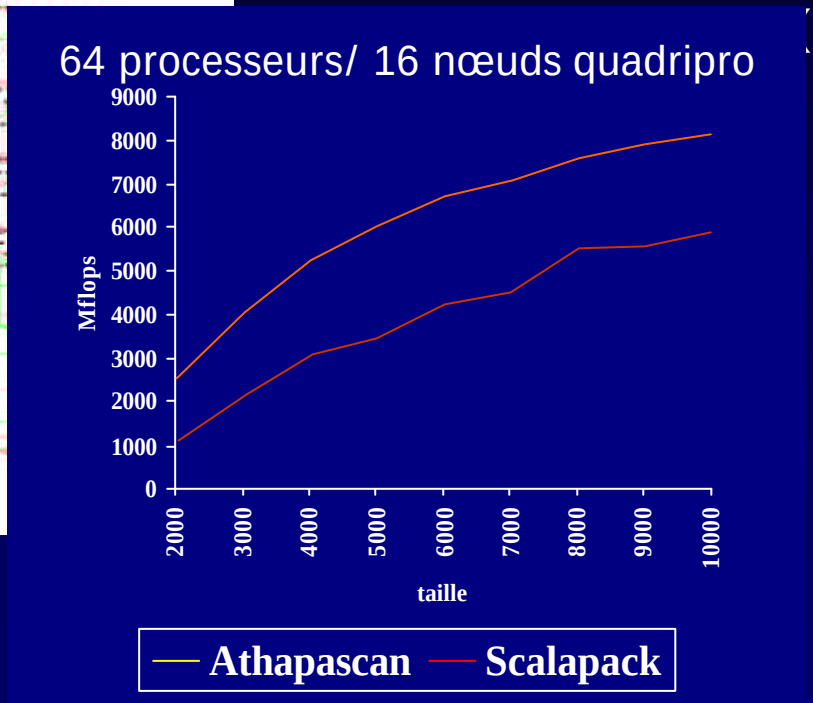
- toute lecture d'une donnée voit la dernière écriture dans l'exécution séquentielle
- séquentialisation implicite

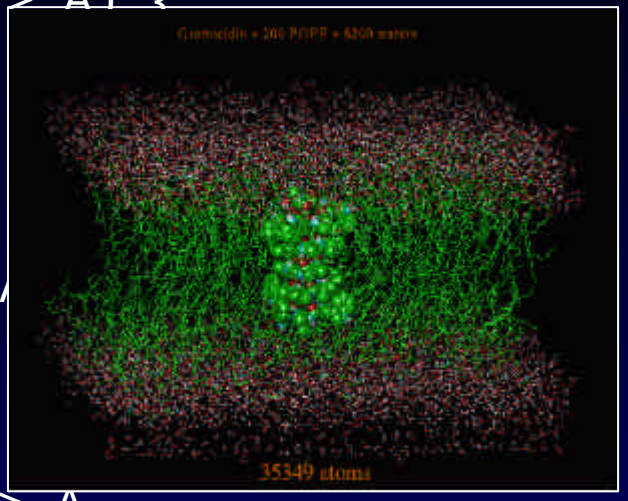
=> prédiction de l 'exécution du programme : flot de données

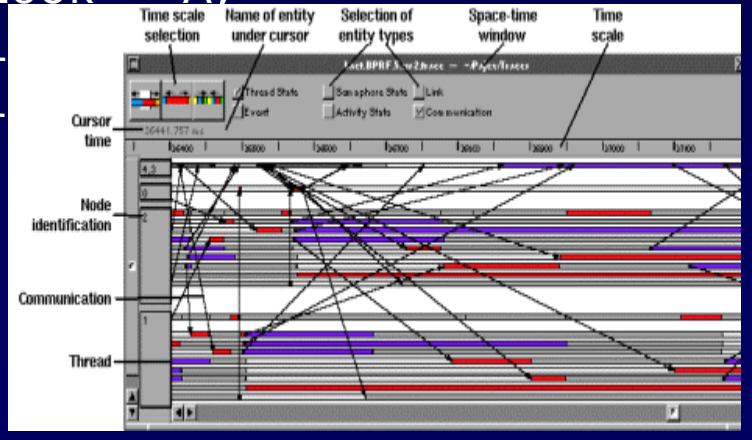
De l ’API C++ à l ’exécution

```
void pardtporf (matrix< Shared< block > > A) {
  for( k=0; k<N-1; k++ ){
    Fork< dtporf >( A(k,k) );
    for( i=k+1; i<N; i++ )
      for( j=k+1; j<N; j++ )
```









Spécialisation de l ’ordonnancement
=> performances

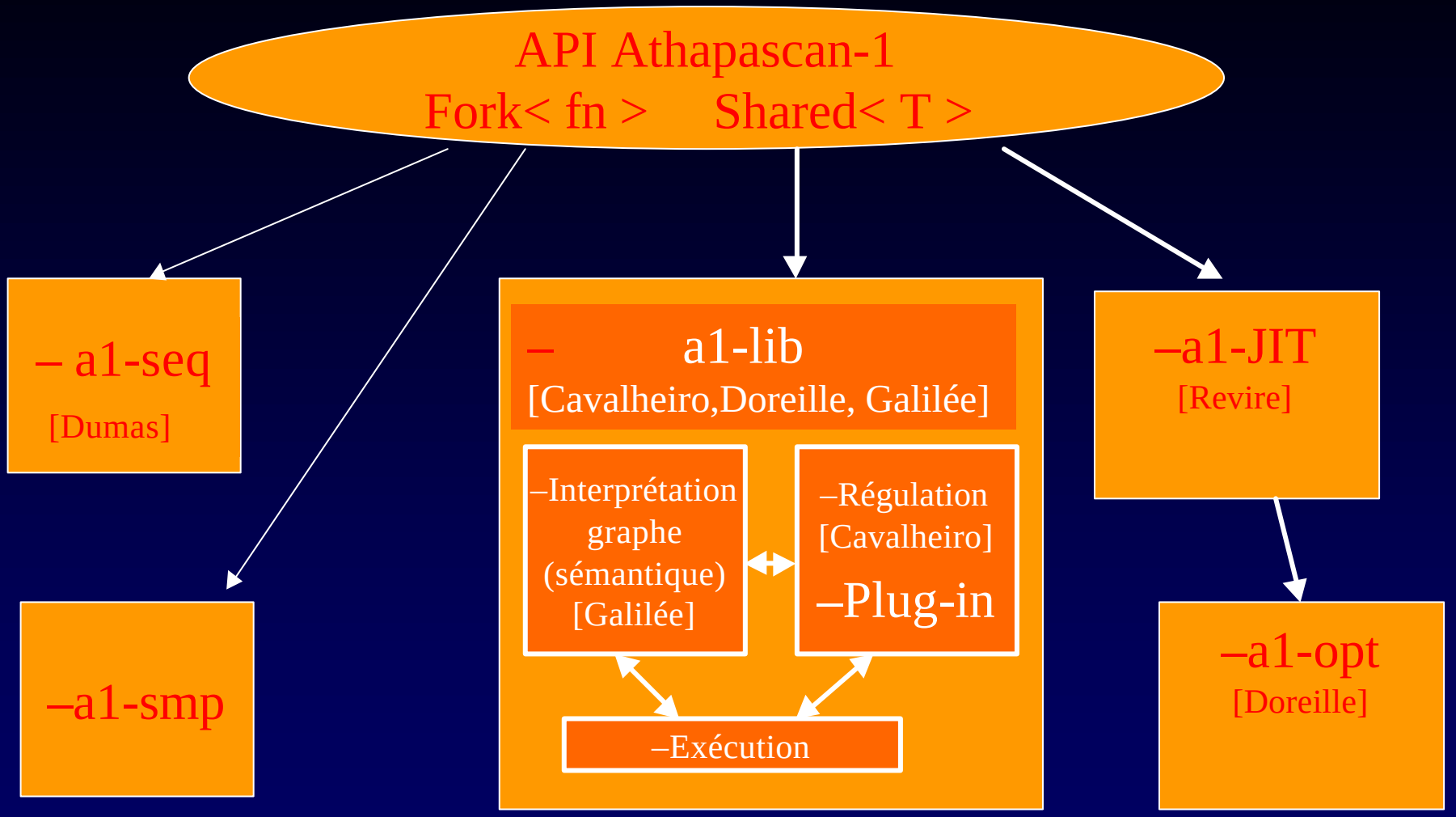
Plan

- Présentation d ’Athapascan1
  - Choix fondateur : calcul du flot de données
  - Interface de programmation
  - Exemple
- Couplage d ’ordonnancement
  - Les ordonnancements spécialisés en Athapascan1
  - Spécialisation pour grappe de PC hétérogènes
    - Couplage dynamique d ’ordonnancement
  - Couplage Ordo Séquentiel / Ordo glouton
  - Couplage Ordo glouton / Ordo statique
- Conclusions

Importance des *scheduler* spécialisés

- Pour s ’adapter à un certain type de machine :
  - SMP : [glouton/Work-stealing]
    - Régulation sur inactivité
  - distribuée mono-processeur :[Jade, Rapid]
    - compilation des communications
- Pour optimiser d ’autres critères que le temps:
  - temps + espace mémoire [Nesl, Scandal]
- Pour une classe d ’applications :
  - recherche arborescente [Socrates/Cilk]
  - graphe diamant [Decker]

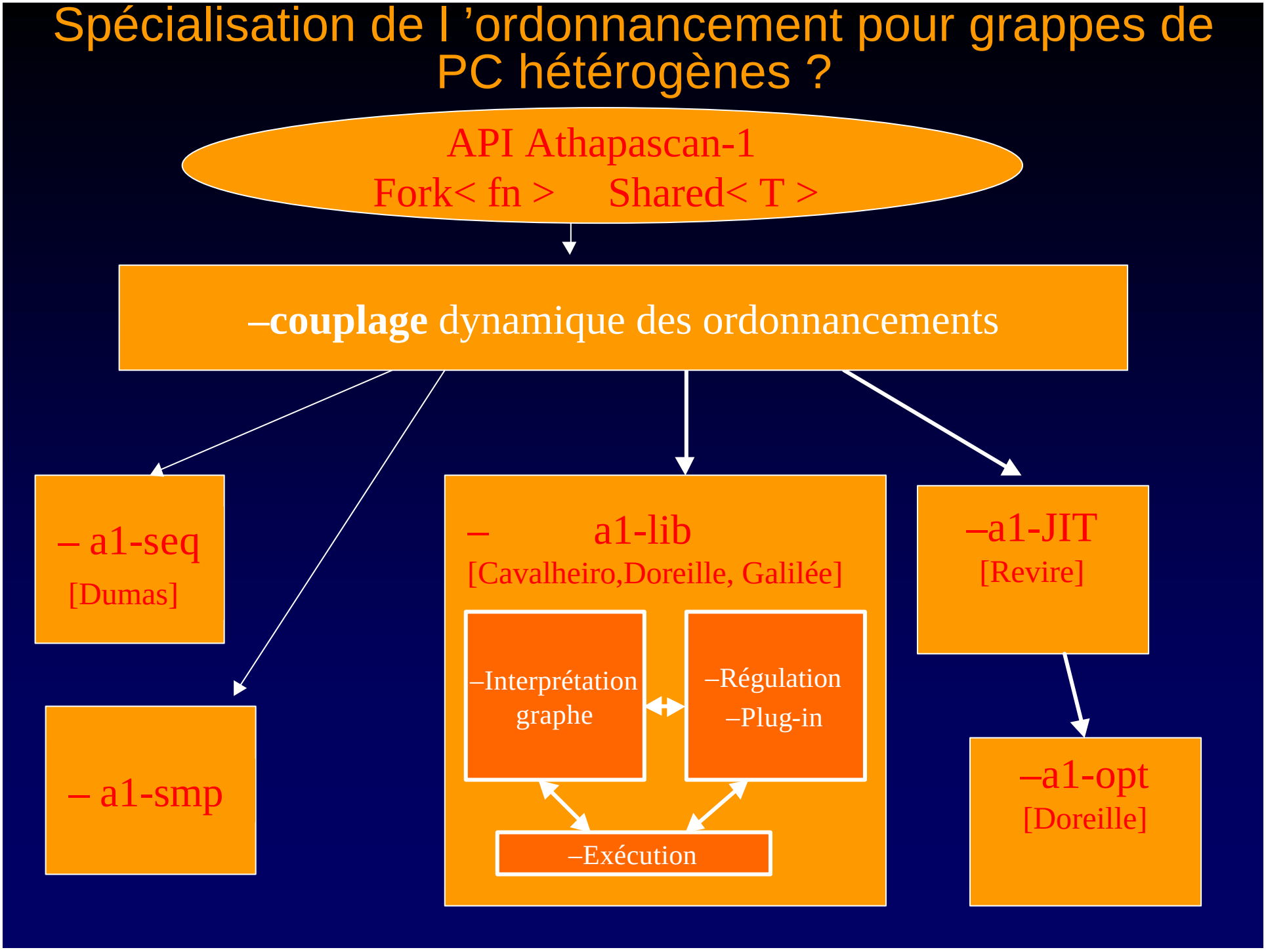
Spécialisation de l ’ordonnancement en Athapascan1



Ordonnancements spécialisés

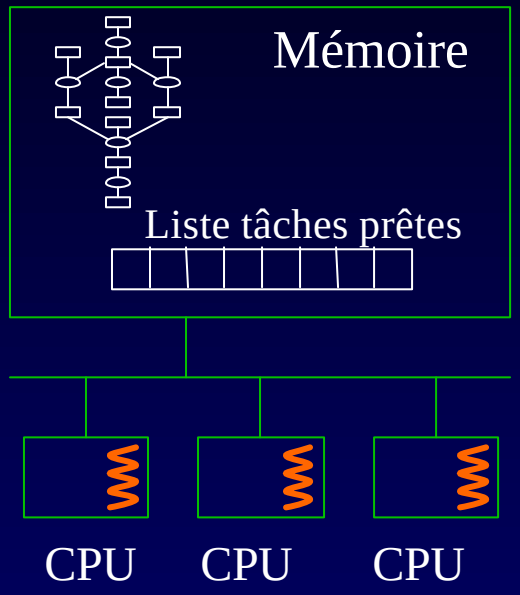
Couplage interprétation/ordonnancement

Spécialisation de l ’ordonnancement pour grappes de PC hétérogènes ?



Couplage Ordo séquentiel / Ordo glouton

- Exemple de Cilk [Blumofe] sur SMP :
  - Ordonnancement glouton pour limiter l 'inactivité
  - Surcoût au niveau des vols
    - Pour limiter le surcoût si pas de vol : création de tâche = appel de fonction séquentiel
  - File d 'attente avec priorité pour limiter l 'espace mémoire utilisé



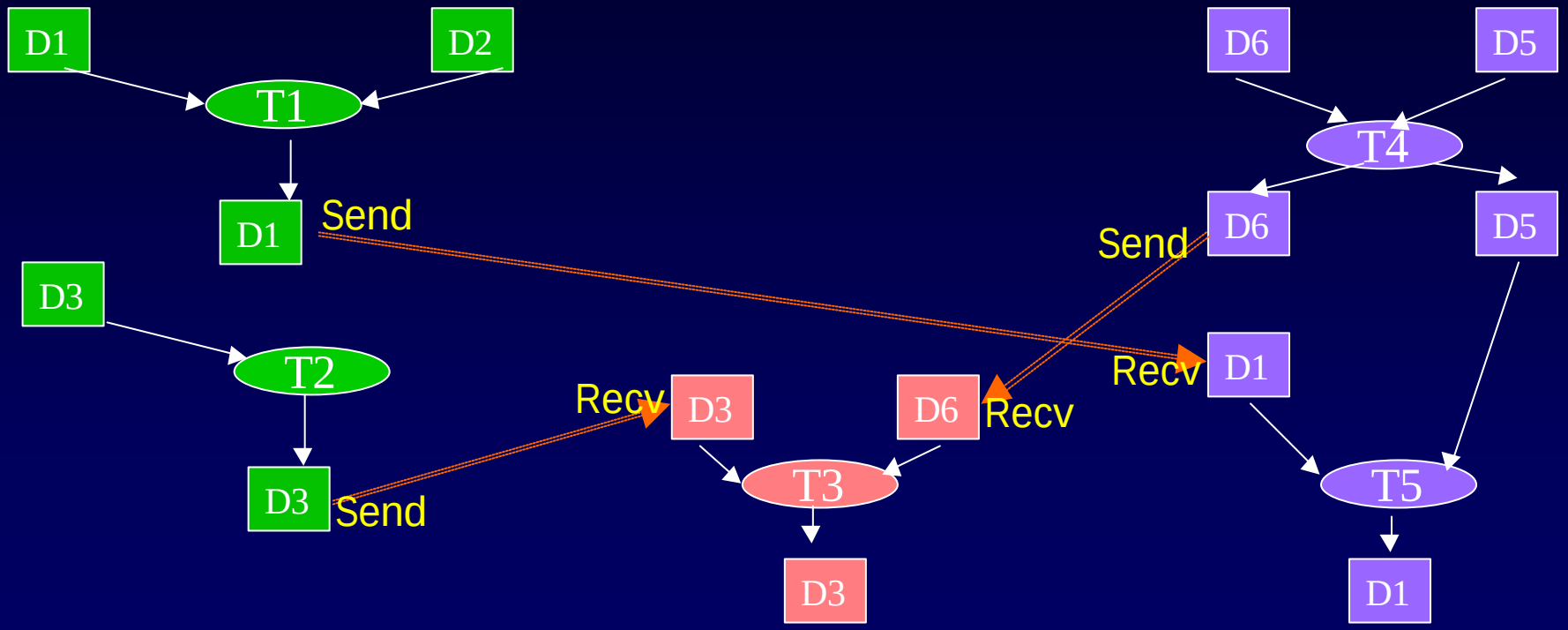
- En Athapascan1 :
  - Conversion des objets partagés entre ordonnancement (A. Bermand)
    - ordo séquentiel : pointeur sur une donnée
    - ordo permettant le vol de tâches : chaînage des versions

Couplage ordo Glouton / ordo statique

- Intérêt : garder les avantages des 2 ordts.
  - limiter les communications.
  - Régulation de type glouton pour prendre en compte :
    - l ’hétérogénéité des processeurs
    - la difficulté à prédire la charge des processeurs
- Principe :
  - pré-distribution des tâches sur chaque nœud de la grappe.
  - Tâches créés dynamiquement : dégénération SMP/Seq
  - Régulation globale par remise en cause sur inactivité locale

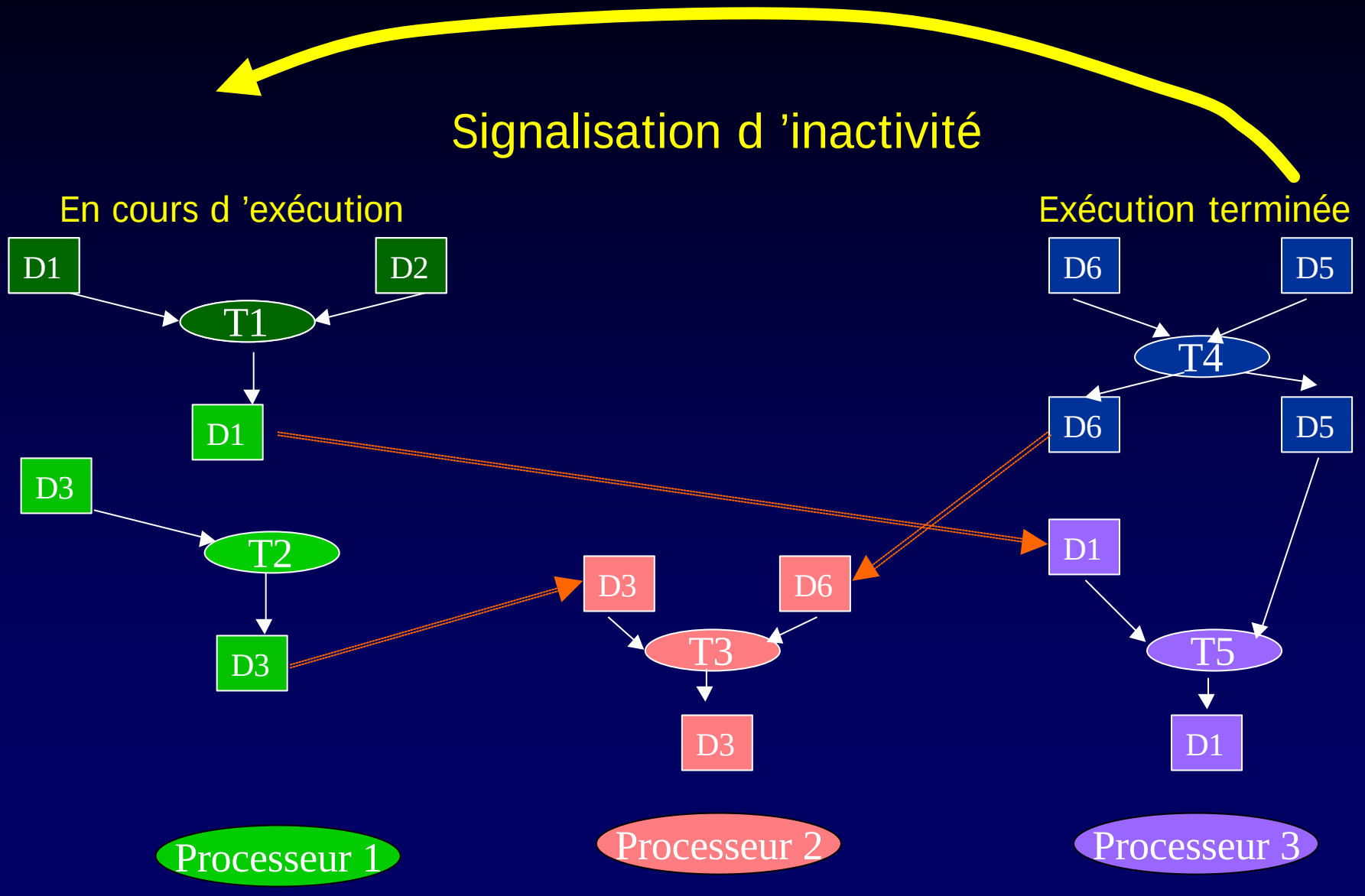
couplage glouton/statique: exemple

- Pré-distribution des tâches
- compilation des communications

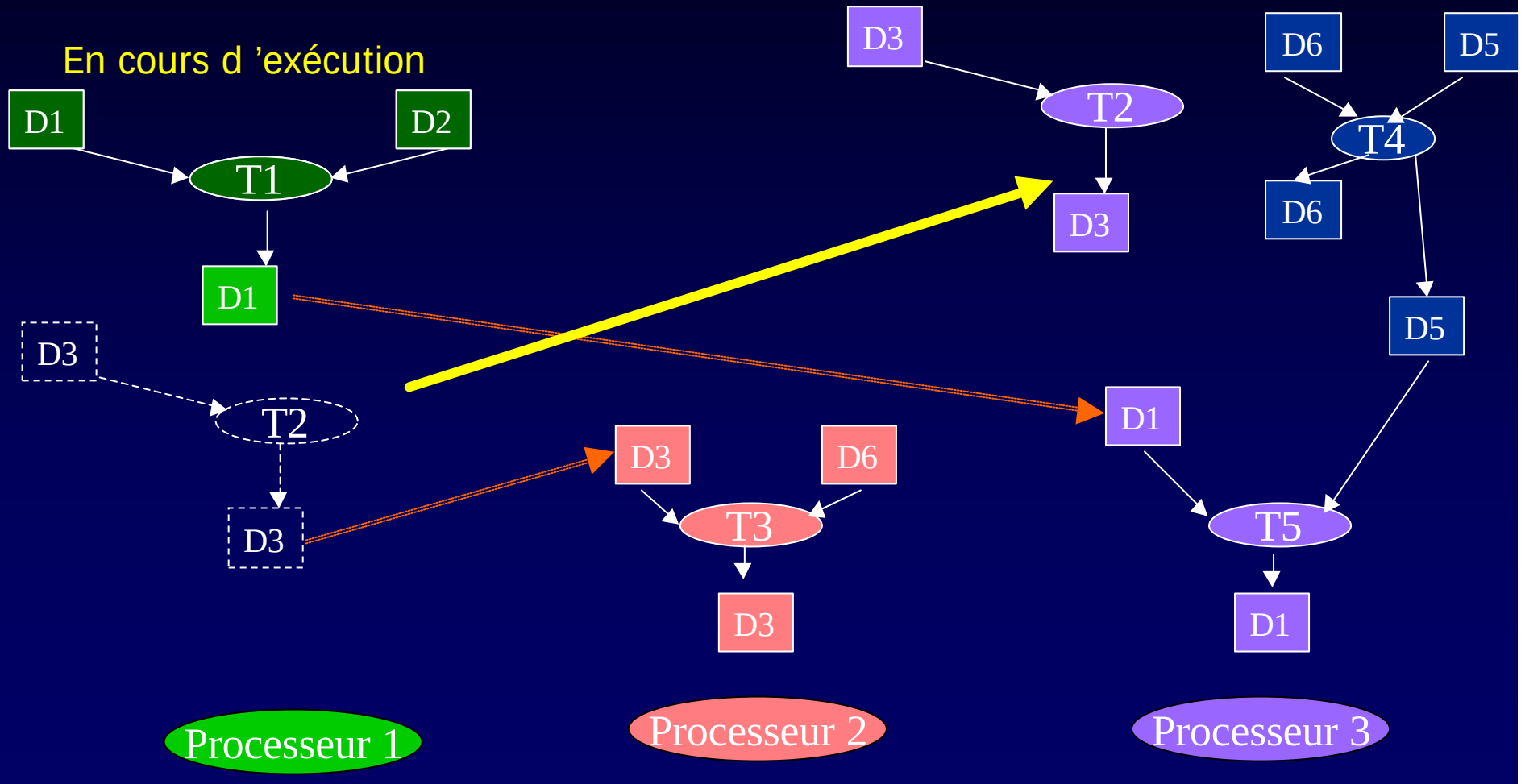


Problèmes posés par le couplage d ’ordt ?

Premier problème : quand remettre en cause ?

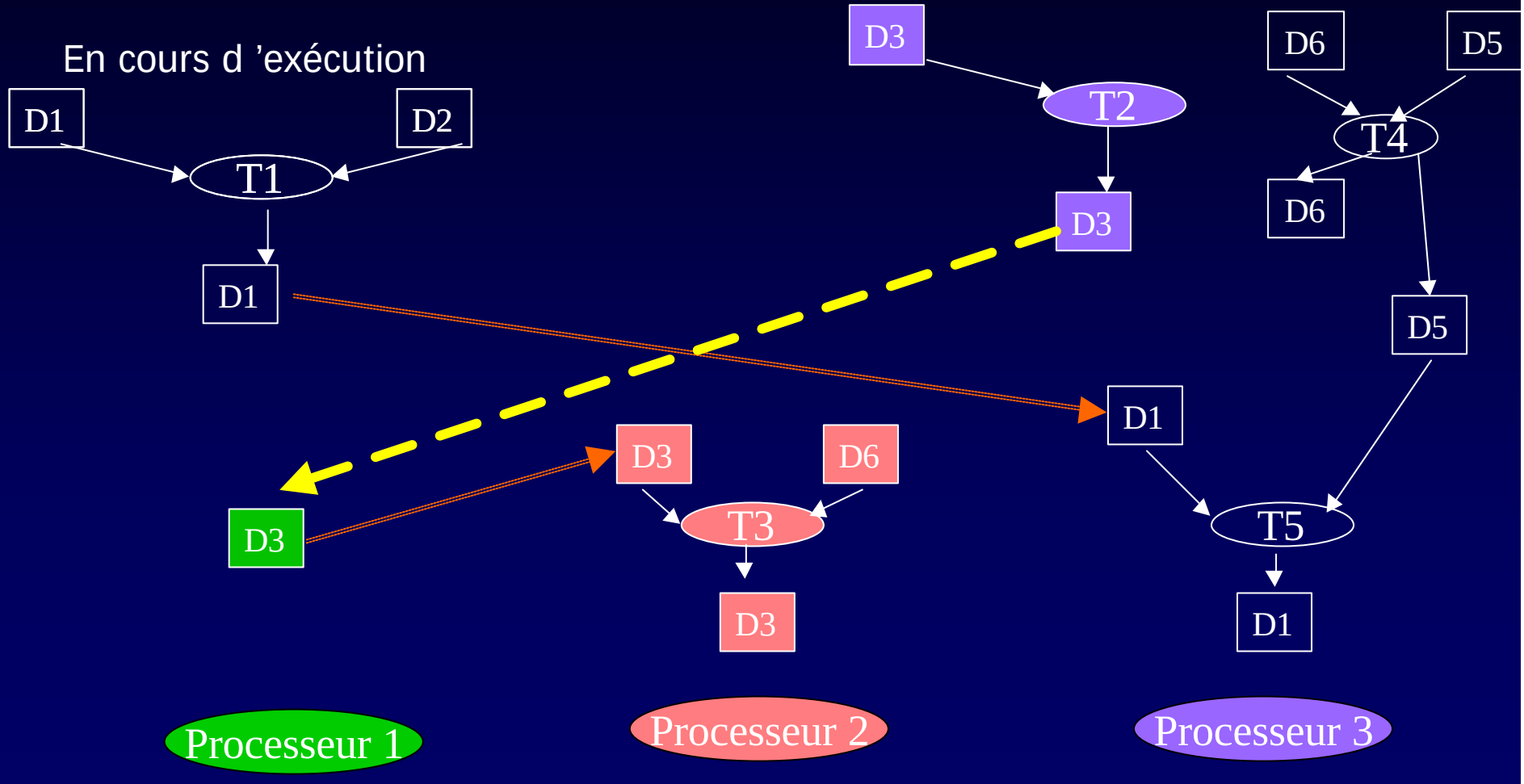


2eme problème : comment mieux répartir ?



3ème problème : conséquences de la remise en cause

- Si nécessaire : mise en place de liens de poursuite



Les points critiques

- Quand remettre en cause un ordonnancement ?
  - Sur Inactivité
    - Un processeur n ’a plus de tâches à exécuter.
    - Il reste des tâches mais elles ne sont pas prêtes.
- Comment remettre en cause un ordonnancement ?
  - Signalisation locale : vol de travail
  - signalisation globale : nouveau calcul de la distribution des tâches

Conclusions

- Spécialisation de l’ordo = clef pour les performances
- Spécialisation pour grappes de PC :
  - Couplage :
    - D’algorithmes de placement / régulation de charge
    - De différentes façons d ’interpréter le graphe
- A faire :
  - Justification pratique (Eval de performances)
  - Justifications théorique
    - Influence de l ’ordo sur les défauts de page
    - ...

Fin