
# Programmation par tâches : une vision du futur pour le calcul parallèle

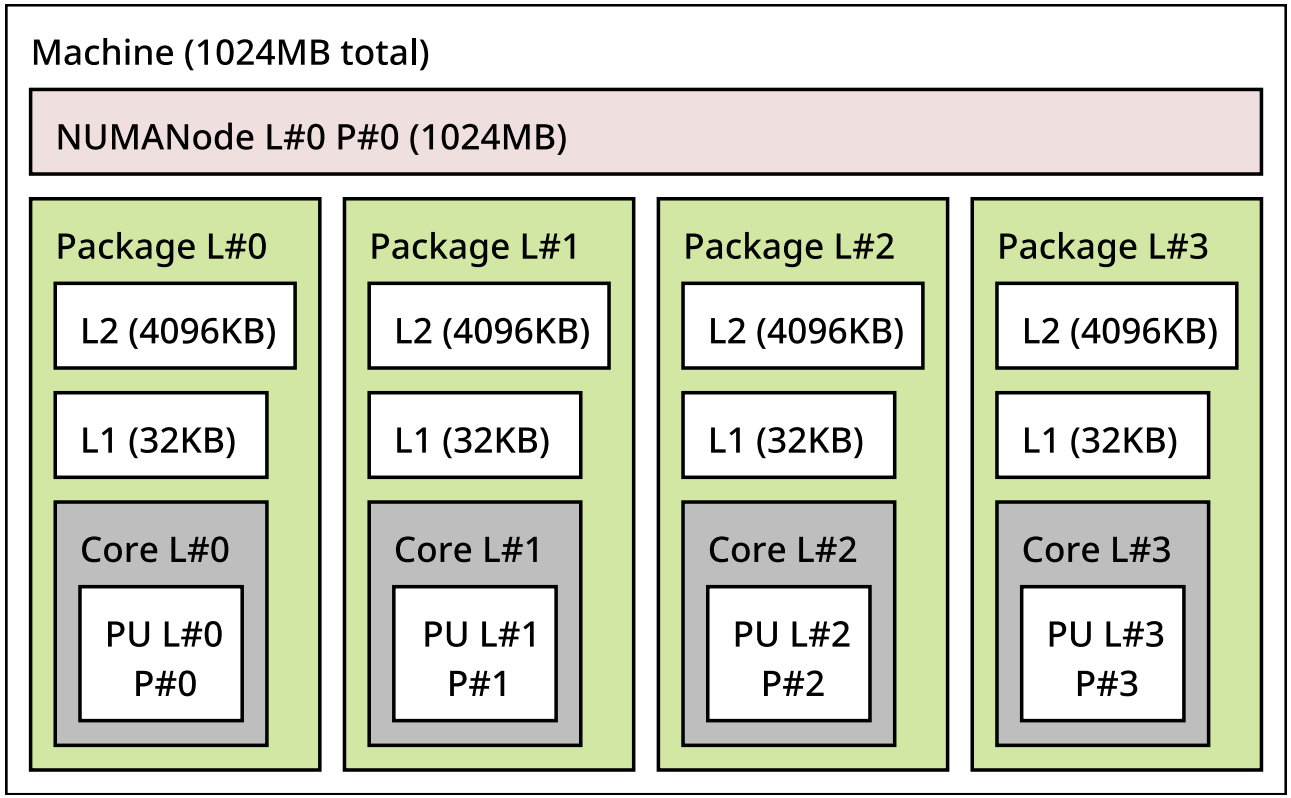
Samuel Thibault

INRIA Bordeaux - Sud-Ouest -- STORM Team

# Once upon a time...

## Un parallélisme à plat

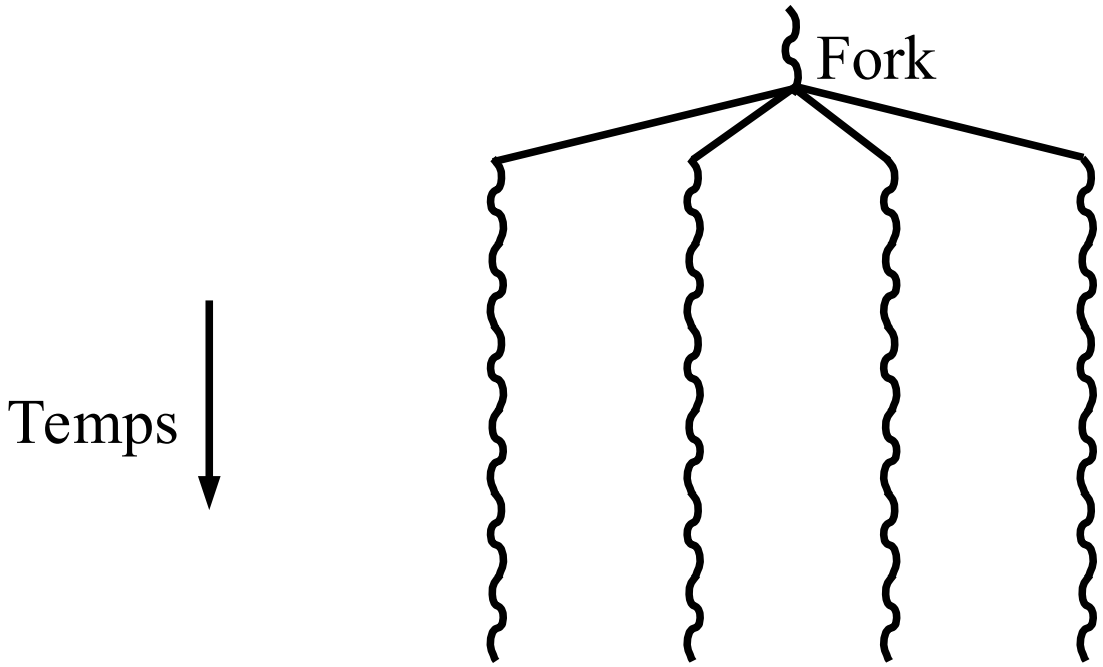
- Des cœurs bien alignés



# Once upon a time...

Un parallélisme à plat

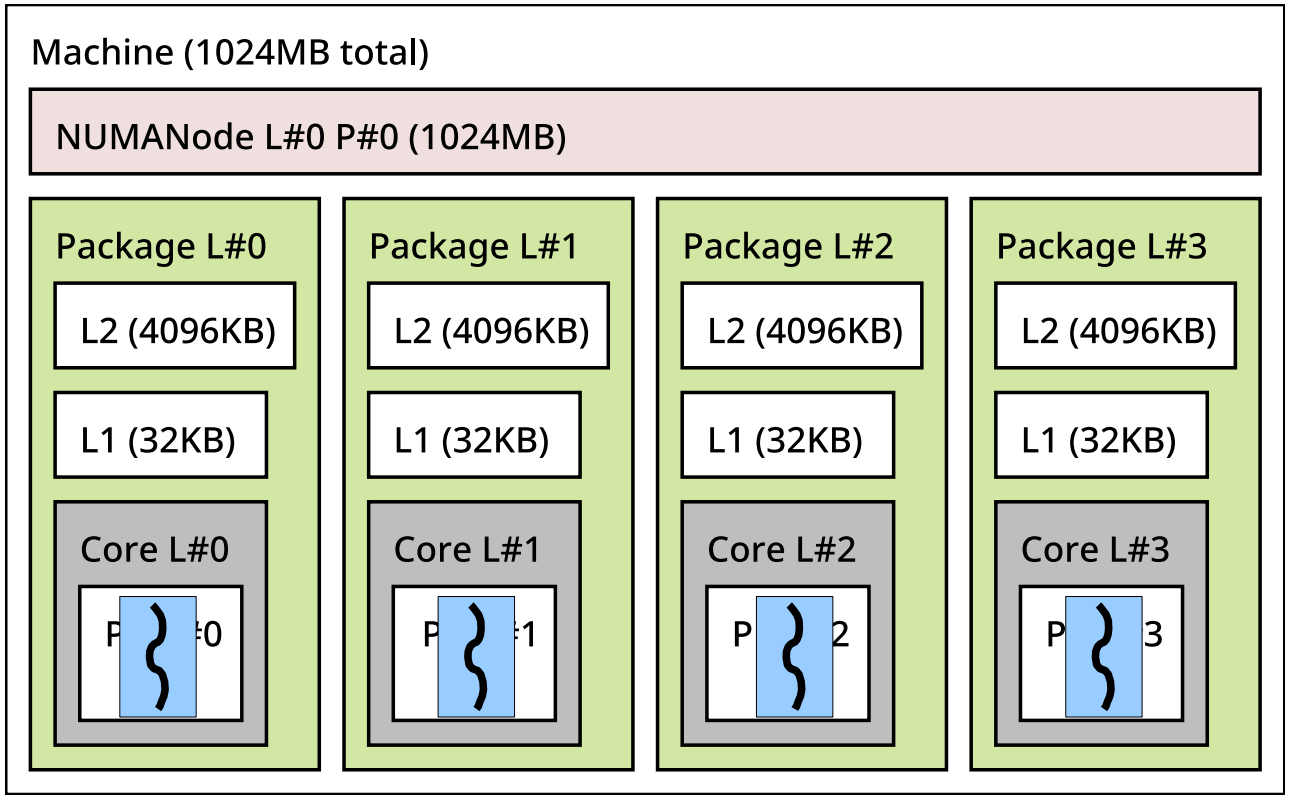
- Des threads bien alignés (Bulk-Synchronous Programming)



# Once upon a time...

## Un parallélisme à plat

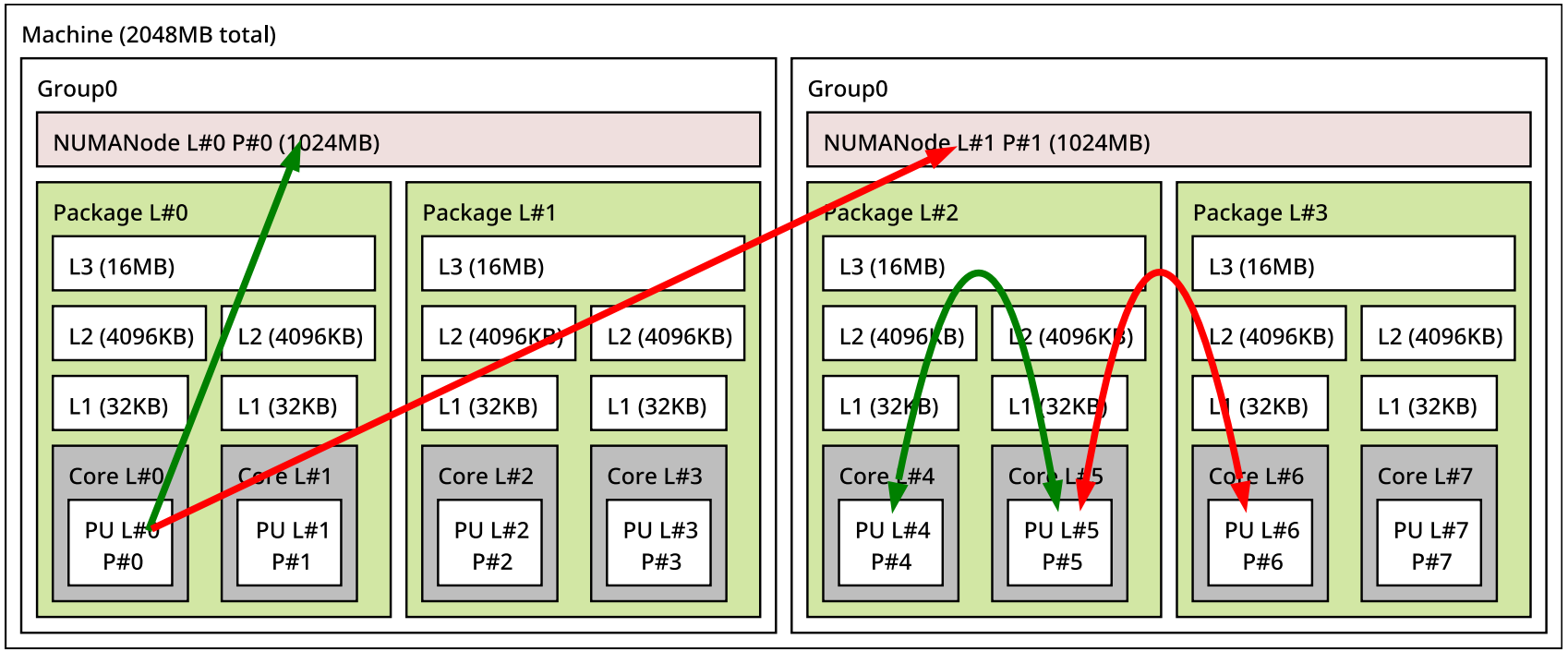
- Un alignement parfait



# Later on...

Un parallélisme pas si plat que ça... (~2005)

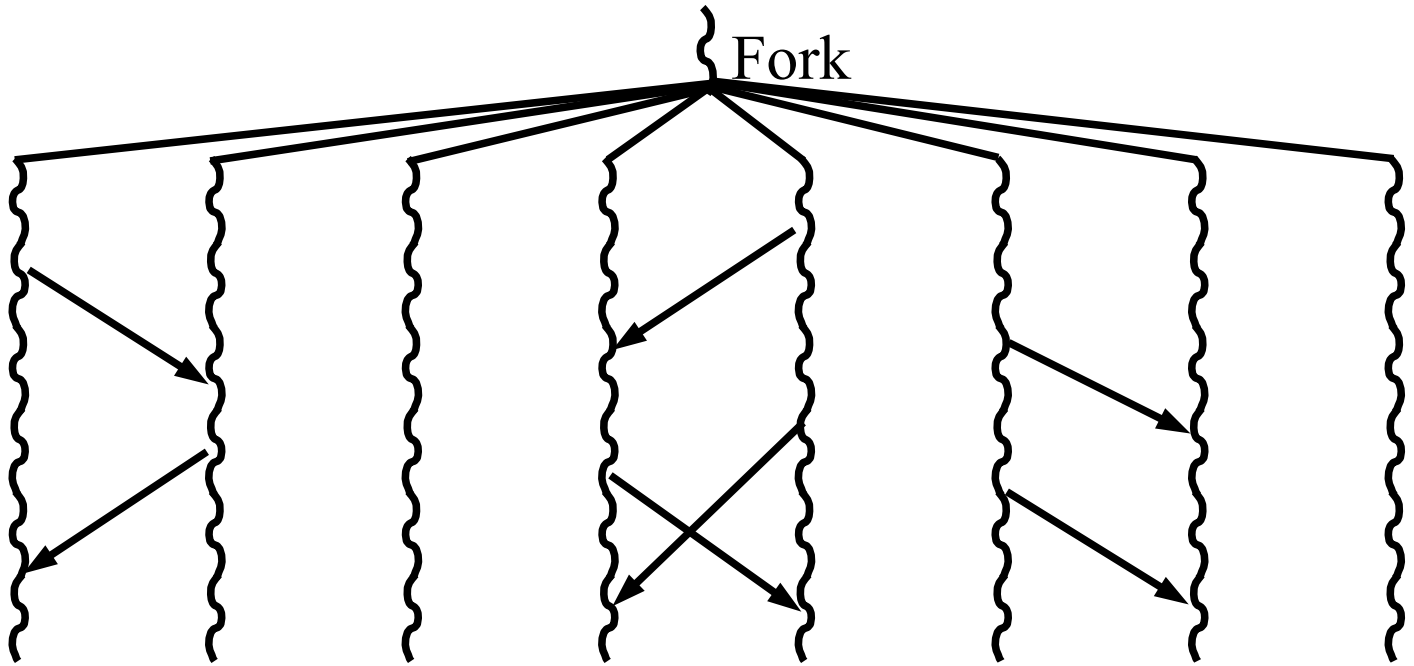
- Affinité NUMA
- Affinité de cache



# Later on...

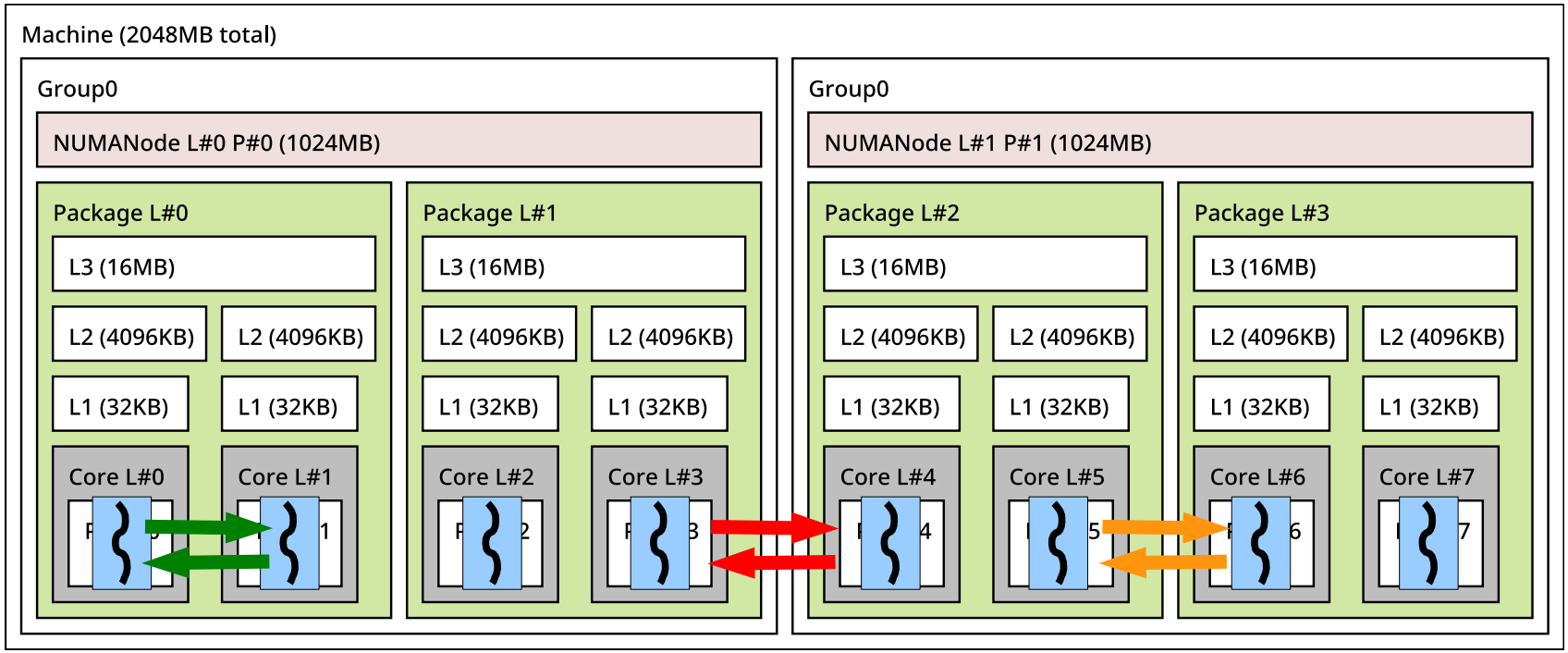
Un parallélisme pas si plat que ça...

- Échanges de données entre threads



# Later on...

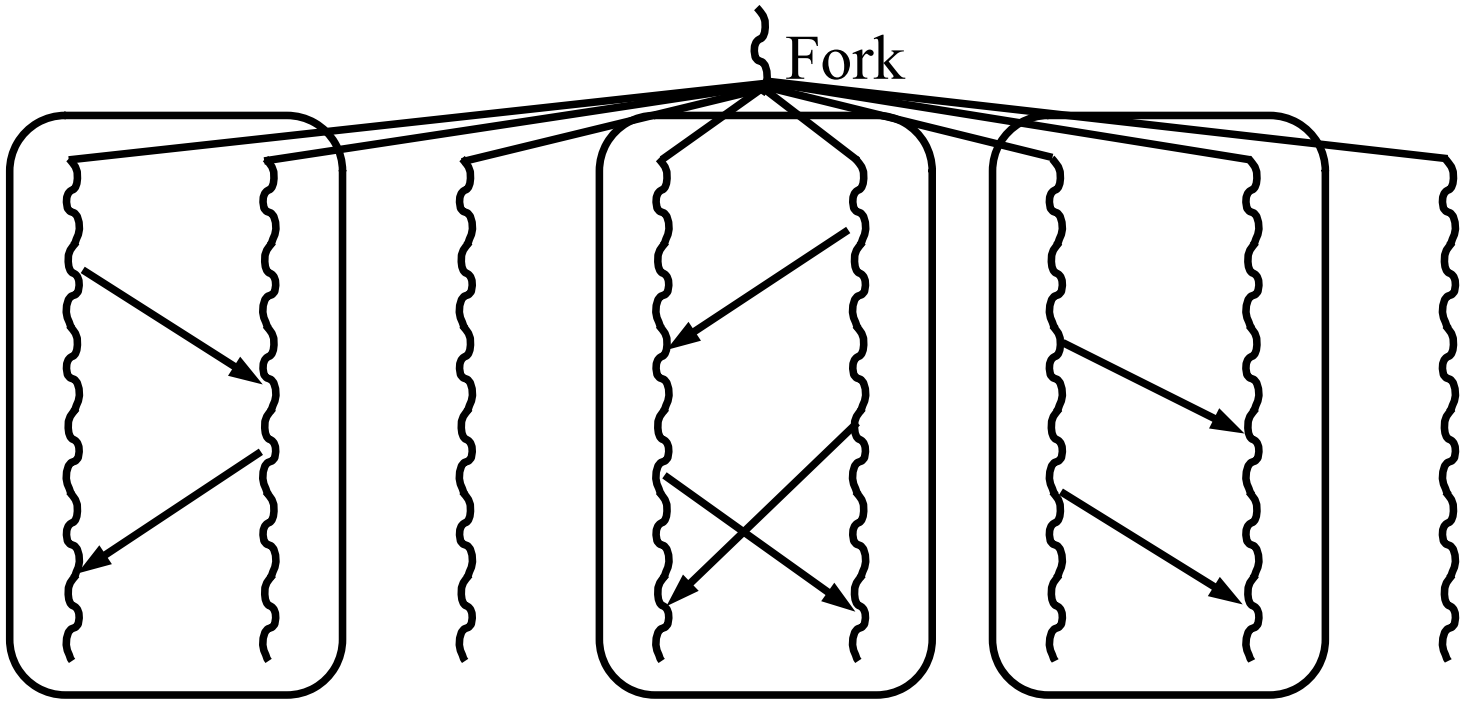
## Un parallélisme pas si plat que ça...



# Later on...

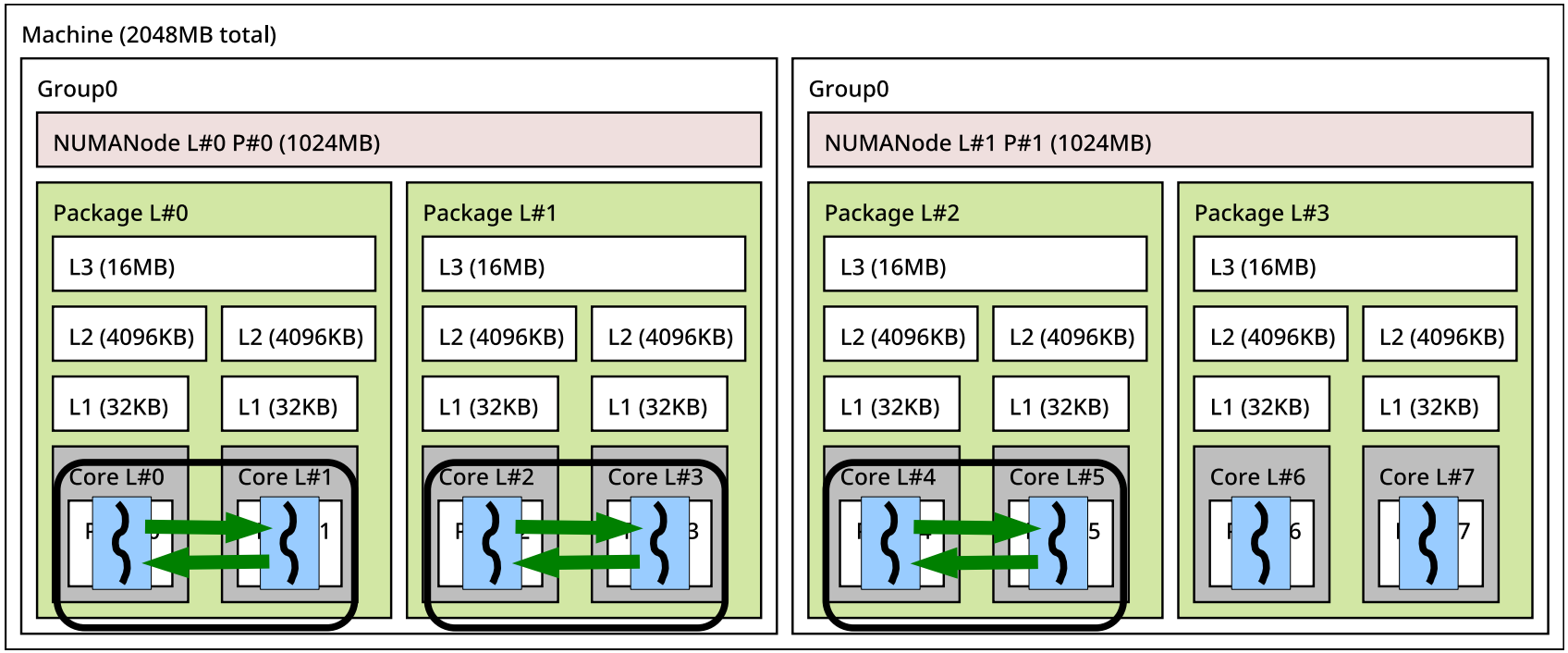
Un parallélisme pas si plat que ça...

- Exprimer la structure : bulles (~2007)



# Later on...

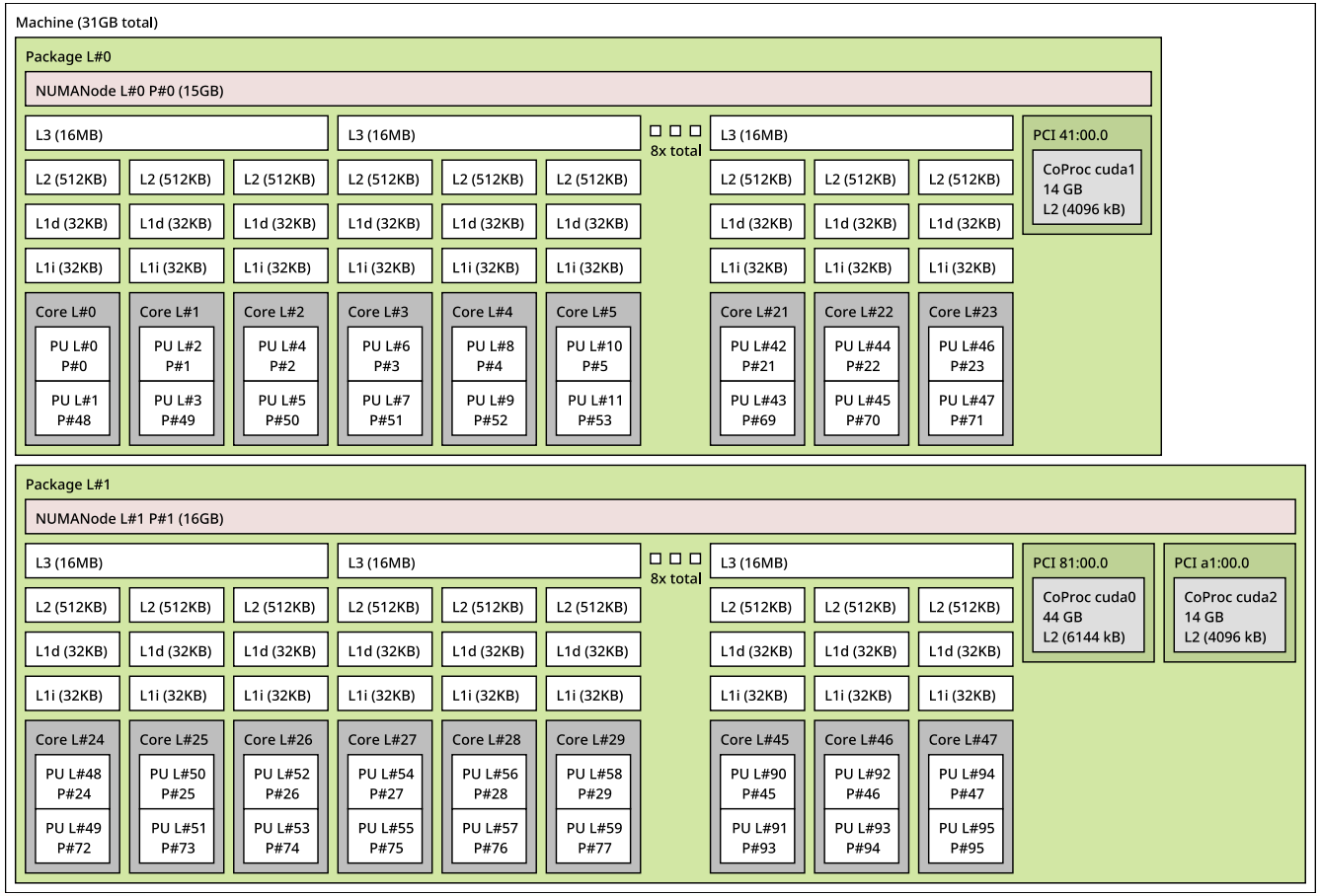
- Utilisateur
  - Binding du runtime, du job scheduler
- OS
  - Migration de processus/threads



# Nowadays...

## Un parallélisme pas si plat que ça...

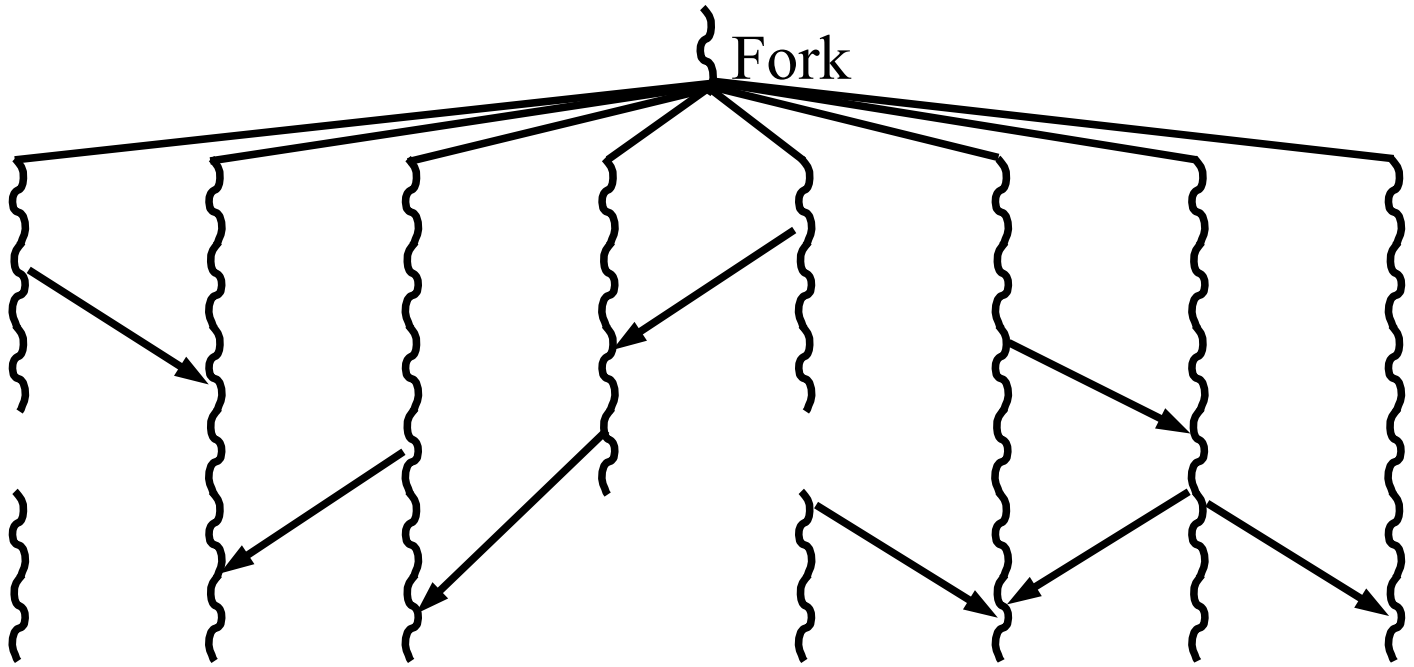
- Des GPUs !
  - Vitesse de calcul très différente sur CPU et sur GPU, transferts de données



# Nowadays...

Un parallélisme pas si plat que ça...

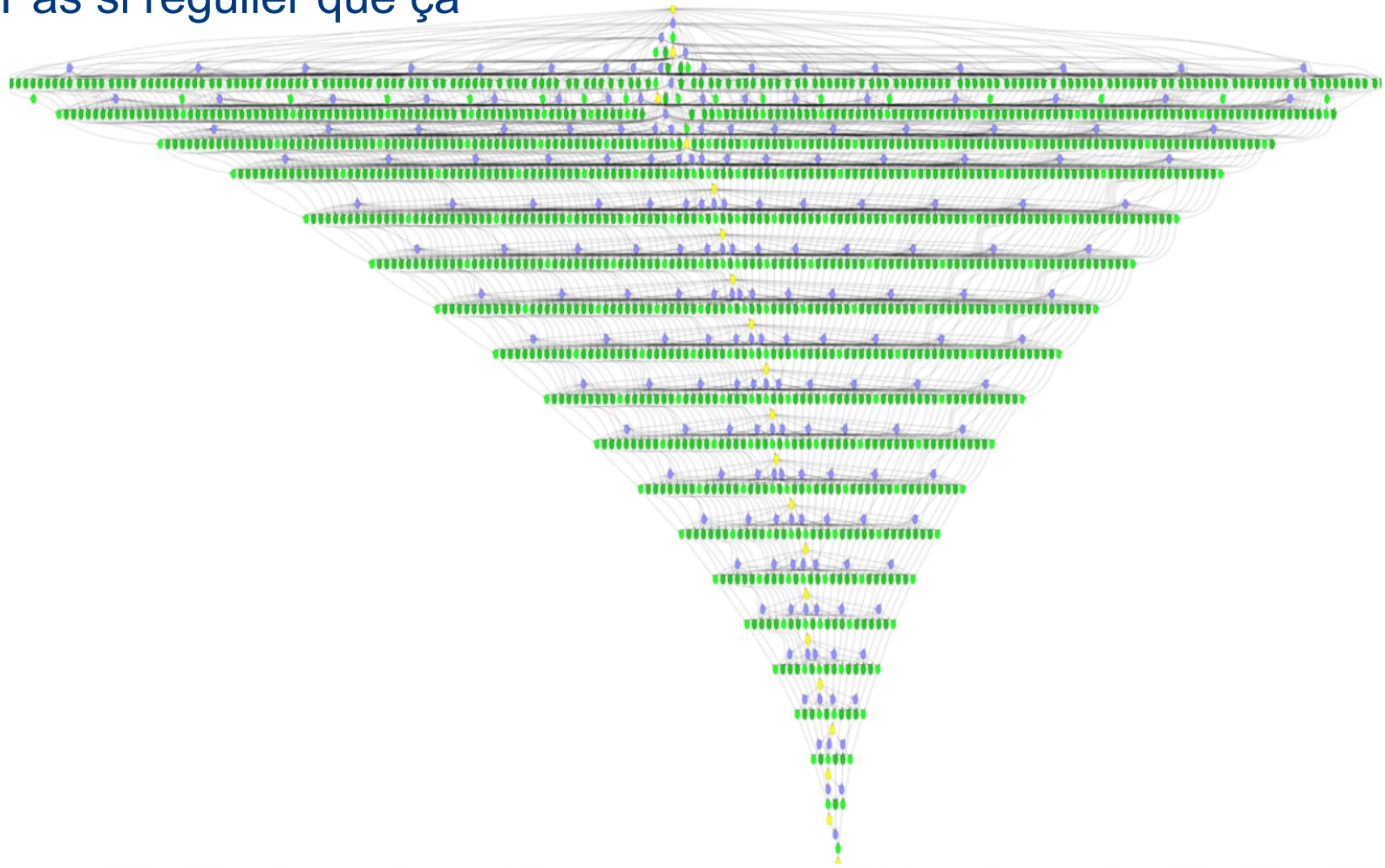
- Pas si régulier que ça



# Nowadays...

Un parallélisme pas si plat que ça...

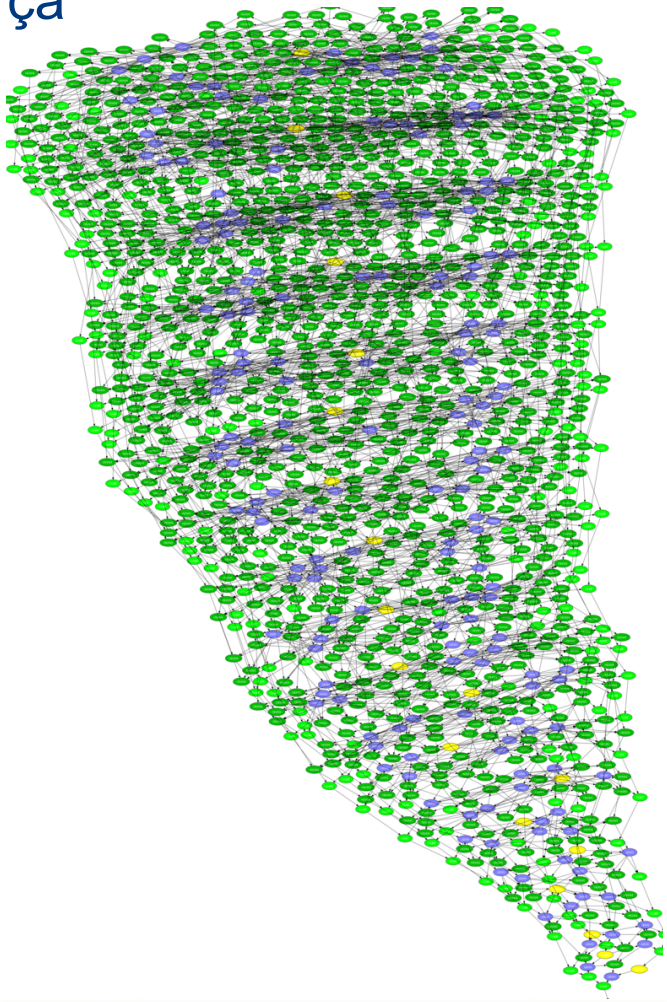
- Pas si régulier que ça



# Nowadays...

Un parallélisme pas si plat que ça...

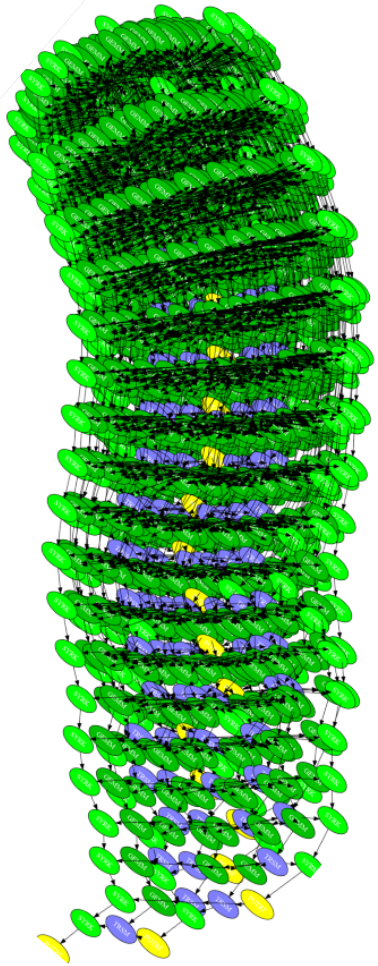
- Pas si régulier que ça



# Nowadays...

Un parallélisme pas si plat que ça...

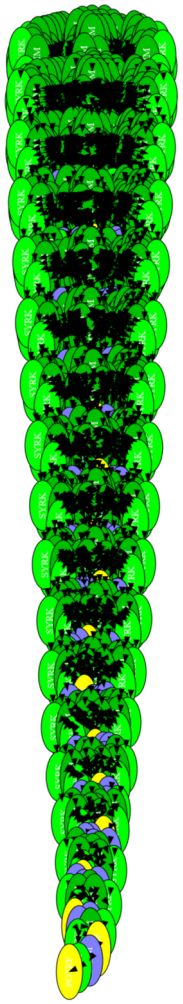
- Pas si régulier que ça



# Nowadays...

Un parallélisme pas si plat que ça...

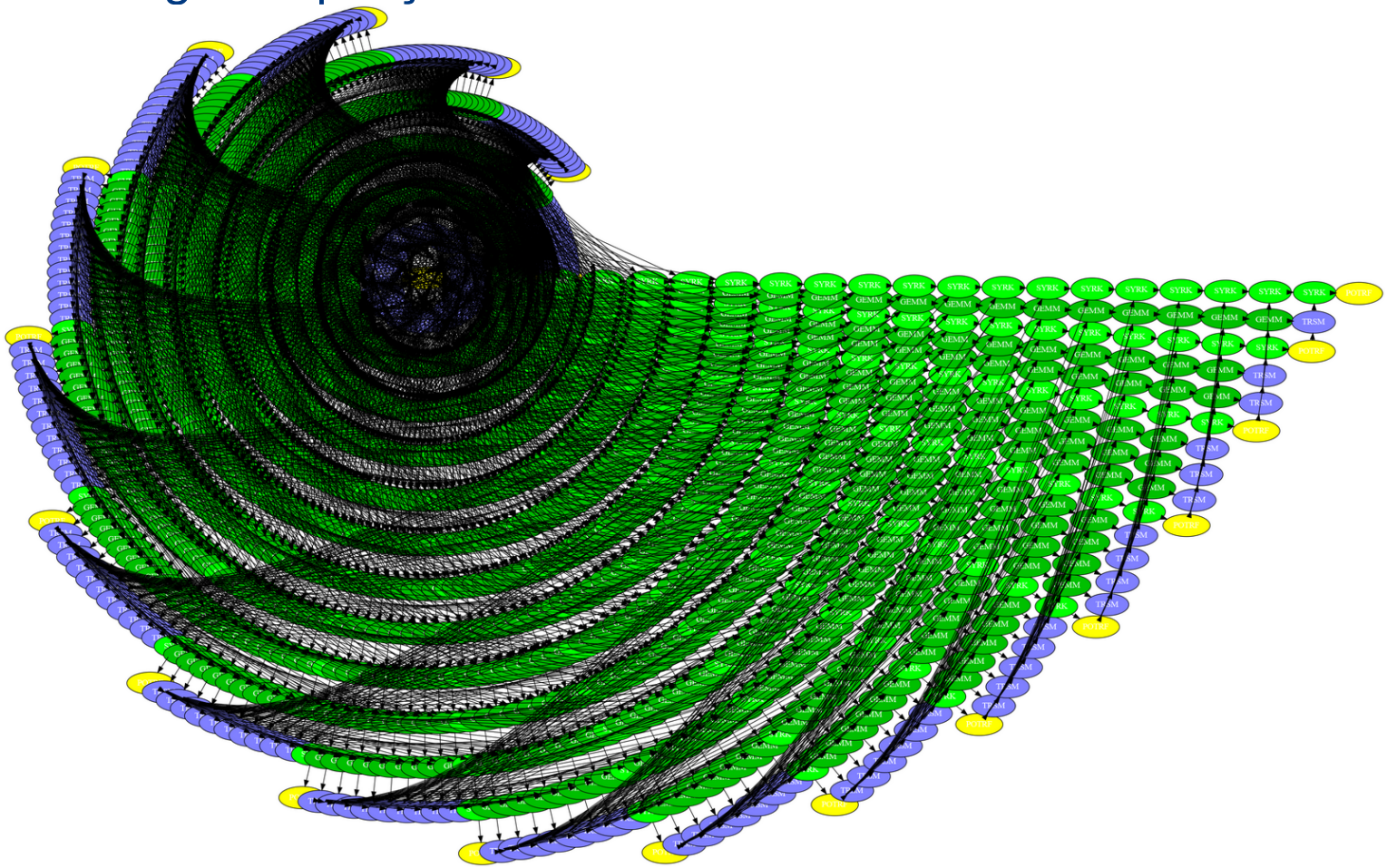
- Pas si régulier que ça



# Nowadays...

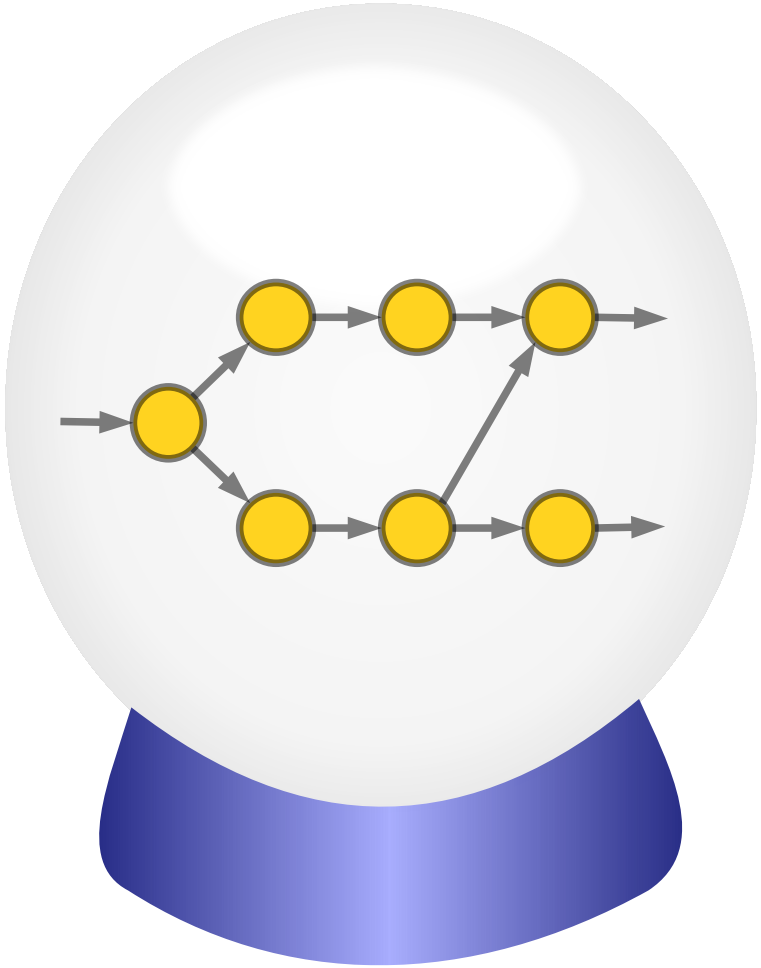
Un parallélisme pas si plat que ça...

- Pas si régulier que ça



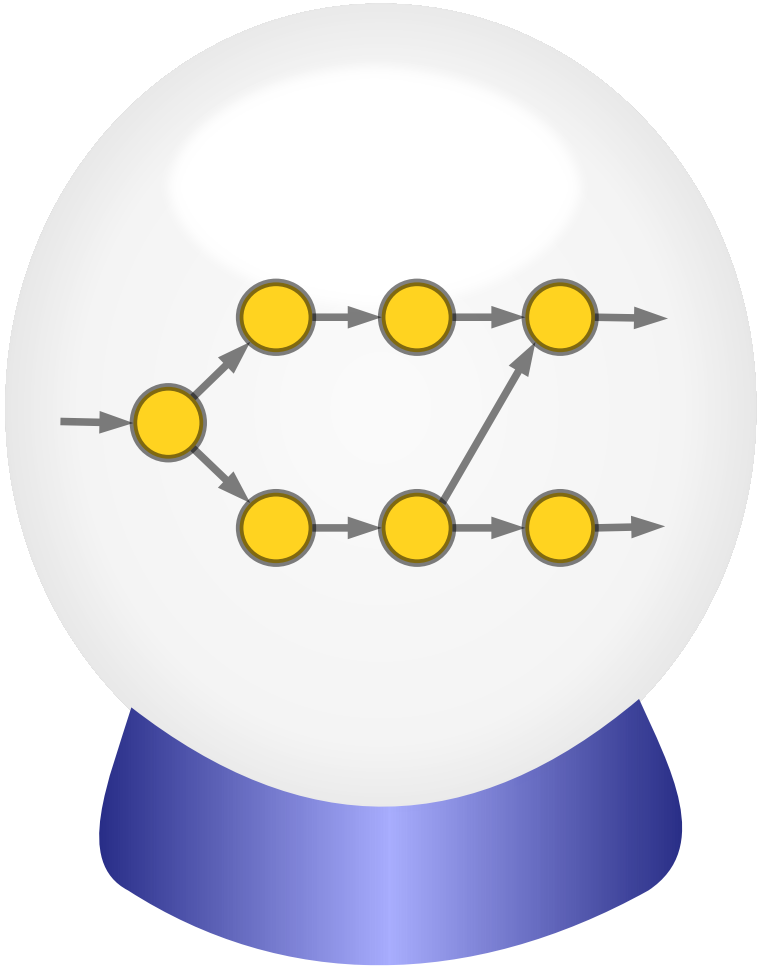
# Nowadays...

- Prédire le futur ?

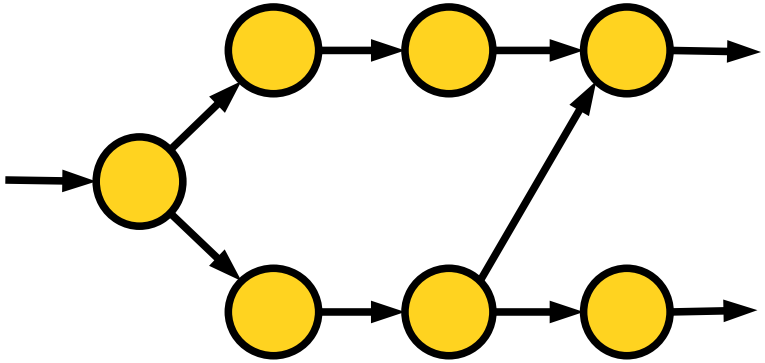


# Nowadays...

- ~~Prédire~~ Choisir le futur ?



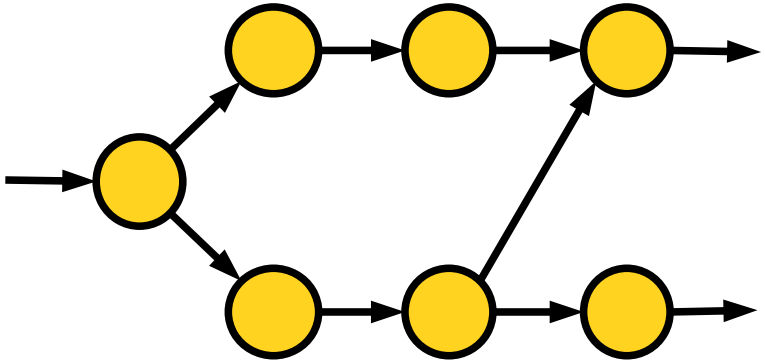
# Ordonnancement de tâches



# Graphes de tâches

Paradigme très anciens

- Travaux en ordonnancement depuis les années 60 !
- Runtimes depuis la fin des années 90
- Très en vogue récemment (~2010)
  - Utilisation des GPUs



# Ordonnancement de graphe de tâches

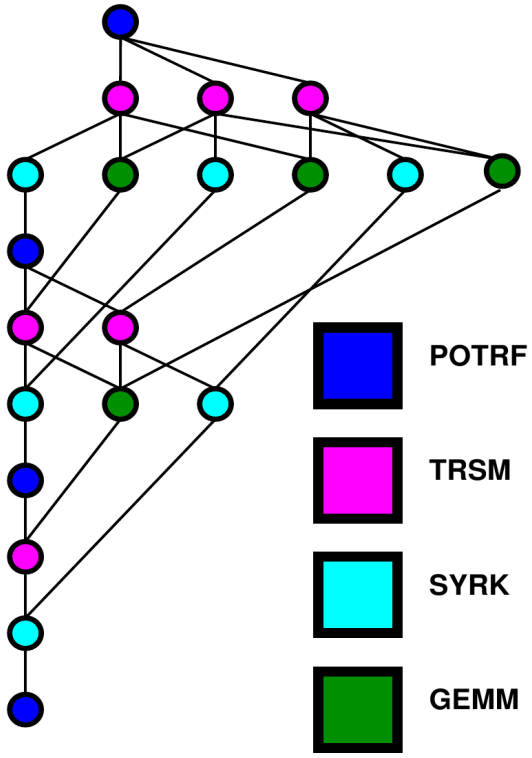
CPUs+GPUs, disque, ...

- Ressources de calcul hétérogènes
- Coût de communication

Objectif : temps de terminaison, en général

(pourrait aussi être l'énergie)

- Attention au parallélisme
- Attention à profiter des GPUs
  - Durée des tâches
- Attention au chemin critique
  - Priorité des tâches
- Attention à la localité
  - Pénalité des transferts de données



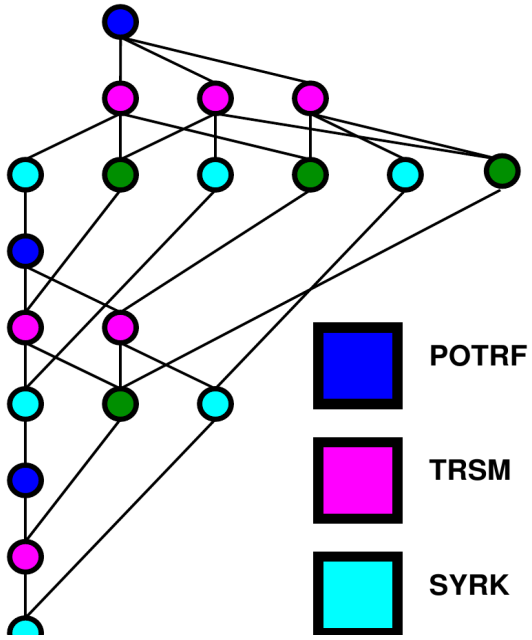
# Ordonnancement de graphe de tâches

E.g. graphe de tâches de Cholesky, N*N blocs

- POTRF: O(N) tâches, ≈ 2x GPU accel, chemin critique
- TRSM: O(N²) tâches, ≈ 11x GPU accel, (chemin critique)
- SYRK: O(N²) tâches, ≈ 26x GPU accel, (chemin critique)
- GEMM: O(N³) tâches, ≈ 29x GPU accel

- O(N³) tâches
- O(N³) arêtes

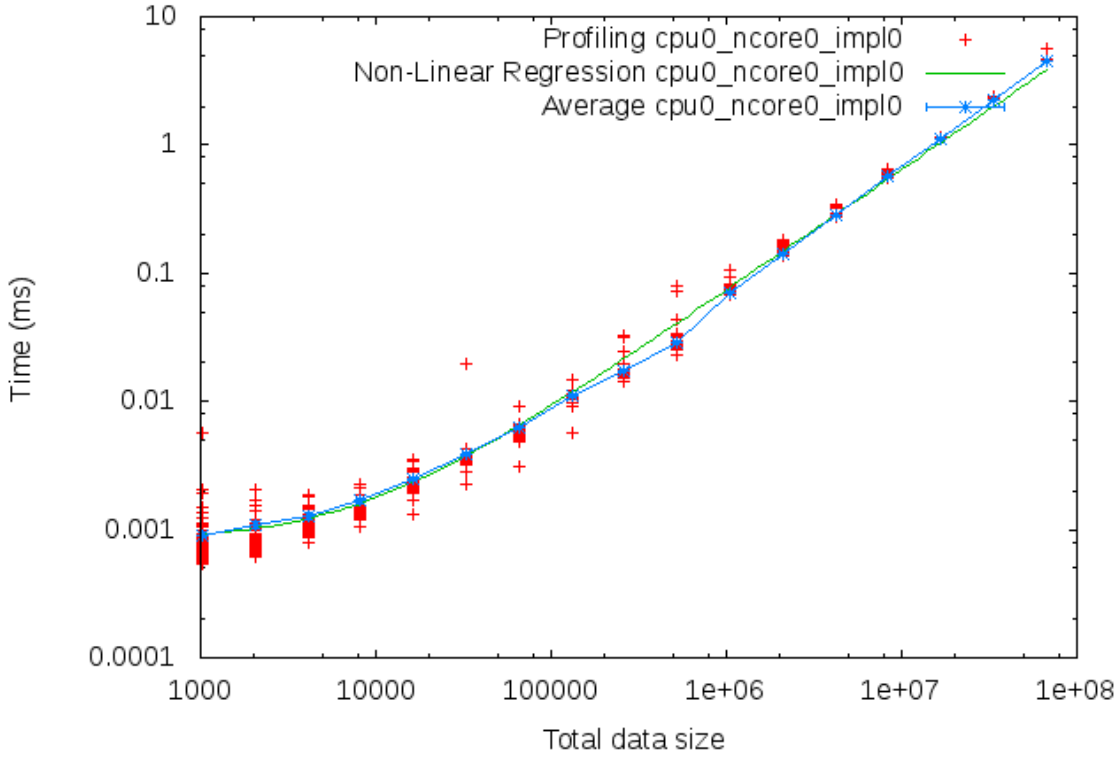
- N peut être 10-100



# Ordonnancement de graphe de tâches

Ordonnancement : suppose que la durée des tâches peut être prévue

- Fonction fournie par l'application
- Modèle paramétrique
- Historique des durées mesurées
- Interpolation



# Interpolation multi-linéaire

## Complexité théorique des tâches GEQRT

$$T_{\texttt{GEQRT}} = a + 2b(NB^2 \times MB) - 2c(NB^3 \times BK) + \frac{4d}{3}NB^3$$

- On peut régler a, b, c, d avec une régression linéaire
- Pourquoi ces paramètres ?

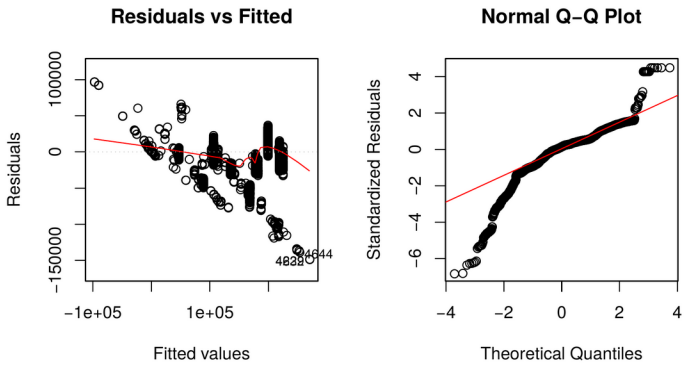
| | GEQRT Duration | |
|---|---|---|
| Constant | $-2.49 \times 10^{1}$ $(-2.83 \times 10^{1}, -2.14 \times 10^{1})$ | *** |
| $NB^2 \times MB$ | $5.49 \times 10^{-7}$ $(5.46 \times 10^{-7}, 5.51 \times 10^{-7})$ | *** |
| $NB^3 \times BK$ | $-5.52 \times 10^{-7}$ $(-5.57 \times 10^{-7}, -5.48 \times 10^{-7})$ | *** |
| $NB^3$ | $1.50 \times 10^{-5}$ $(1.30 \times 10^{-5}, 1.70 \times 10^{-5})$ | *** |
| Observations | 493 | |
| $R^2$ | 0.999 | |
| *Note:* | *p<0.1; **p<0.05; ***p<0.01 | |

- Fonctionne bien avec des tâches régulières

# Interpolation multi-linéaire

## Premiers essais pour ScalFMM, avec les paramètres évidents

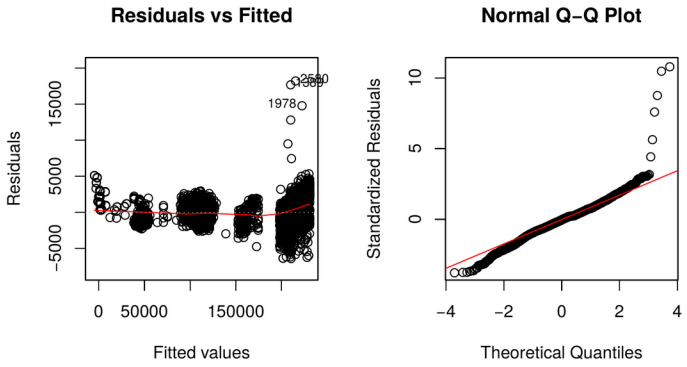
| | | |
|---|---|---|
| $TreeLevel$ | $7.73 \times 10^{1}$ $(7.32 \times 10^{1}, 8.14 \times 10^{1})$ | *** |
| $NbCells$ | $1.52 \times 10^{2}$ $(1.46 \times 10^{2}, 1.59 \times 10^{2})$ | *** |
| $IntervalSize$ | $-5.55 \times 10^{-1}$ $(-6.65 \times 10^{-1}, -4.45 \times 10^{-1})$ | *** |
| $TreeLevel^2$ | $-6.73$ $(-7.10, -6.37)$ | *** |
| $NbCells^2$ | $-6.08 \times 10^{-3}$ $(-7.94 \times 10^{-3}, -4.21 \times 10^{-3})$ | *** |
| $IntervalSize^2$ | $-3.63 \times 10^{-12}$ $(-6.36 \times 10^{-12}, -9.04 \times 10^{-13})$ | |
| $TreeLevel \times NbCells$ | $-4.68$ $(-5.88, -3.48)$ | *** |
| $TreeLevel \times IntervalSize$ | $6.24 \times 10^{-2}$ $(5.02 \times 10^{-2}, 7.46 \times 10^{-2})$ | *** |
| $NbCells \times IntervalSize$ | $1.81 \times 10^{-4}$ $(1.39 \times 10^{-4}, 2.24 \times 10^{-4})$ | *** |
| $TreeLevel \times NbCells \times IntervalSize$ | $-2.08 \times 10^{-5}$ $(-2.55 \times 10^{-5}, -1.61 \times 10^{-5})$ | *** |
| Constant | $-2.27 \times 10^{2}$ $(-2.40 \times 10^{2}, -2.14 \times 10^{2})$ | *** |
| Observations | 4,987 | |
| $R^2$ | 0.906 | |
| *Note:* | *p<0.1; **p<0.05; ***p<0.01 | |



# Interpolation multi-linéaire

## Introduction du paramètre NbInteraction

| | | |
|---|---|---|
| *TreeLevel* | $9.10 \times 10^2$ $(8.46 \times 10^2, 9.74 \times 10^2)$ | *** |
| *NbCells* | 5.34 (5.17, 5.50) | *** |
| *NbInteractions* | $8.59 \times 10^{-1}$ $(8.58 \times 10^{-1}, 8.60 \times 10^{-1})$ | *** |
| Constant | −7.09 (−7.50, −6.67) | *** |
| Observations | 4,987 | |
| $R^2$ | 0.999 | |
| *Note:* | *p<0.1; **p<0.05; ***p<0.01 | |

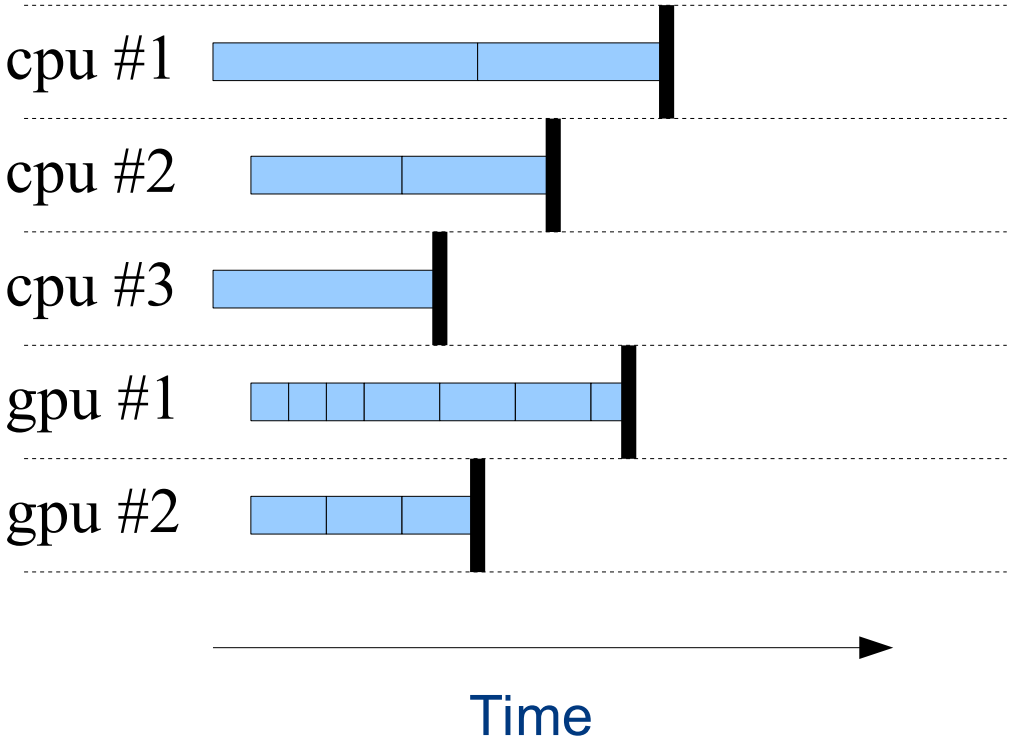


# Ordonnancement de graphe de tâches

Suppose que la durée des tâches peut être prévue ✔

Ordonnancement HEFT (Topcuoglu, HCW'99)

- Heterogeneous Earliest Finish Time

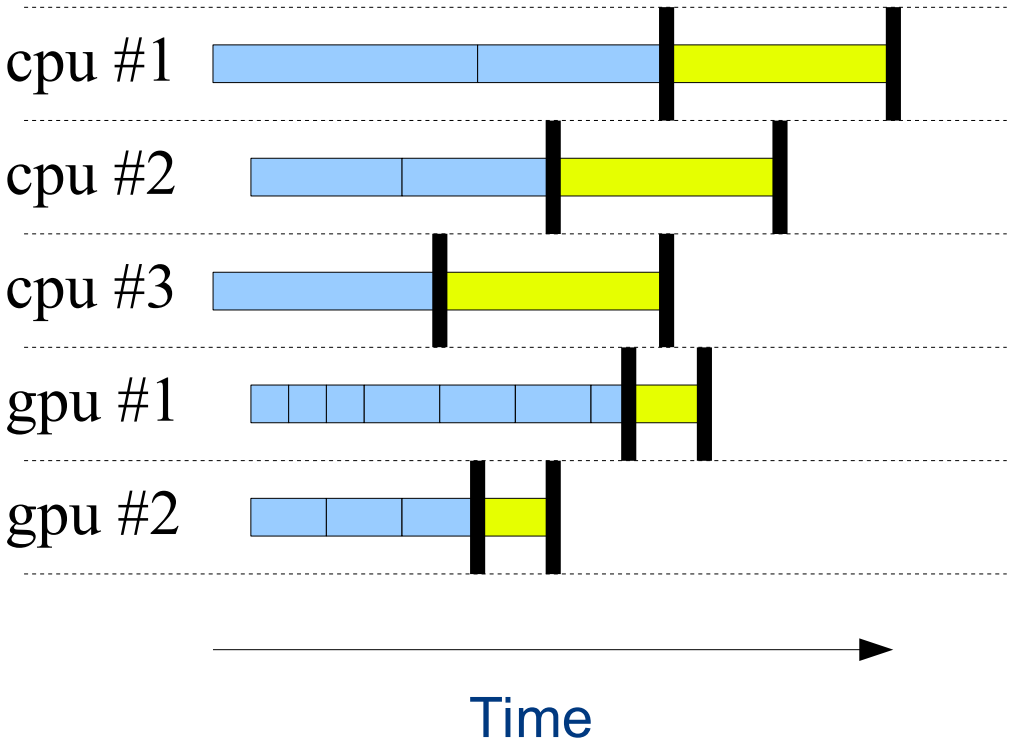


# Ordonnancement de graphe de tâches

Suppose que la durée des tâches peut être prévue ✔

Ordonnancement HEFT (Topcuoglu, HCW'99)

- Heterogeneous Earliest Finish Time

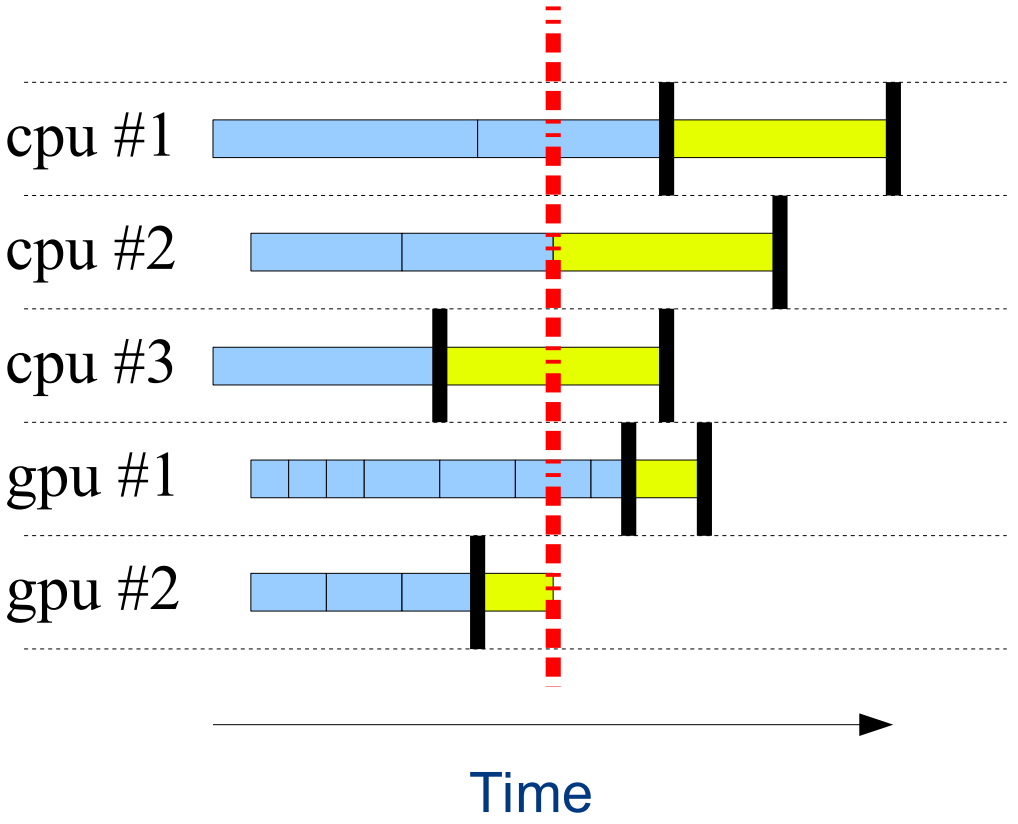


# Ordonnancement de graphe de tâches

Suppose que la durée des tâches peut être prévue ✔

Ordonnancement HEFT (Topcuoglu, HCW'99)

- Heterogeneous Earliest Finish Time

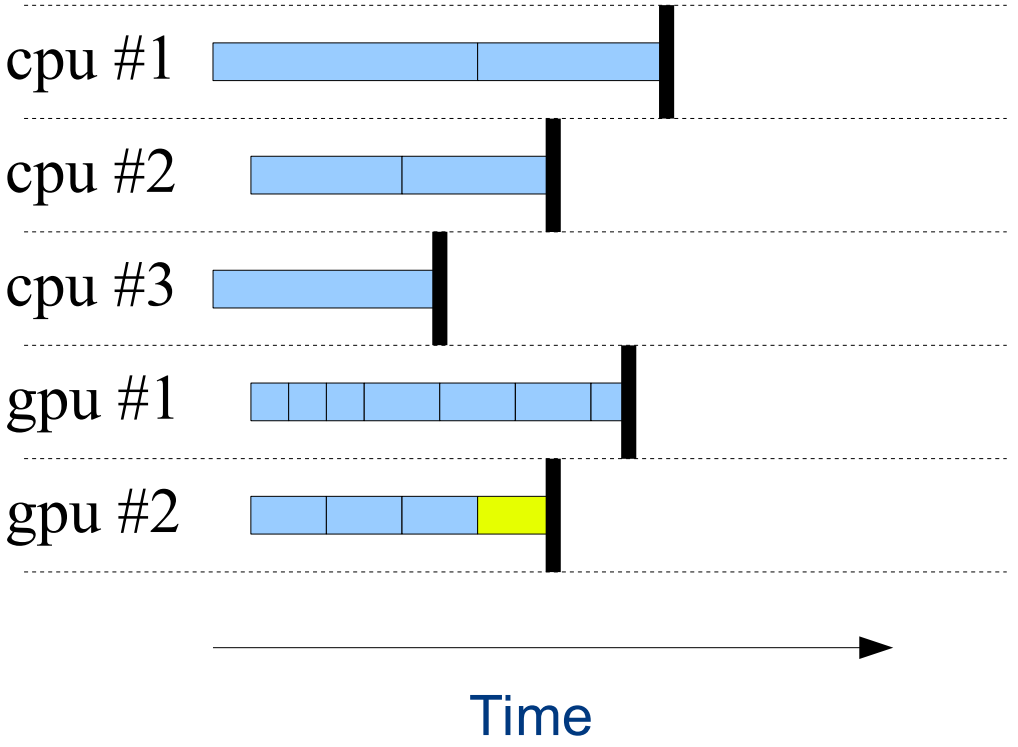


# Ordonnancement de graphe de tâches

Suppose que la durée des tâches peut être prévue ✔

Ordonnancement HEFT (Topcuoglu, HCW'99)

- Heterogeneous Earliest Finish Time



# Ordonnancement de graphe de tâches

Suppose que la durée des tâches peut être prévue ✔

Ordonnancement HEFT (Topcuoglu, HCW'99)

- Heterogeneous Earliest Finish Time

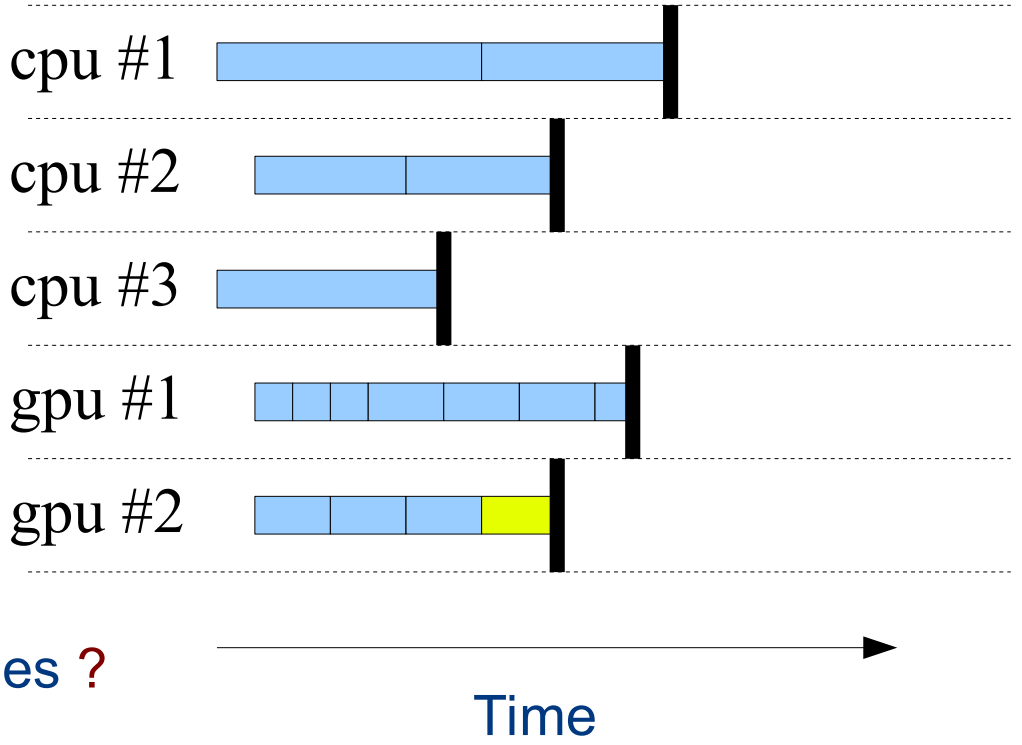
Attention à profiter des GPUs

✔

Attention au chemin critique

✔

Attention à la localité des données ?



# Ordonnancement de graphe de tâches

## Localité des données

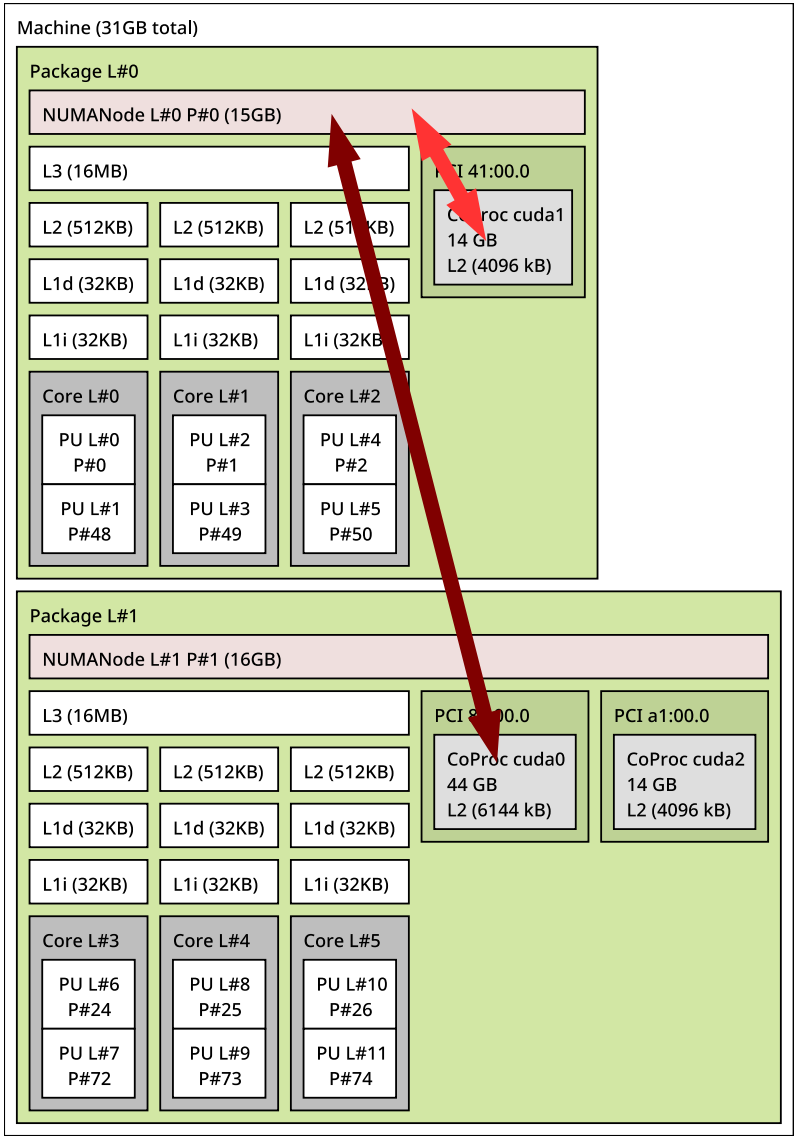
- Mémoire CPU vs mémoire GPU
- Bus PCI et hôte *lents*

## Localité vs équilibre de charge

- Ne peut pas éviter tous les transferts
- Les minimiser

## Utiliser une prédiction du temps de transfert

- Calibration



# Ordonnancement de graphe de tâches

Suppose que la durée des tâches peut être prévue ✔

Ordonnancement DMDA (Augonnet'10)

- Deque Modeling Data-Aware

Attention à profiter des GPUs

✔

Attention au chemin critique

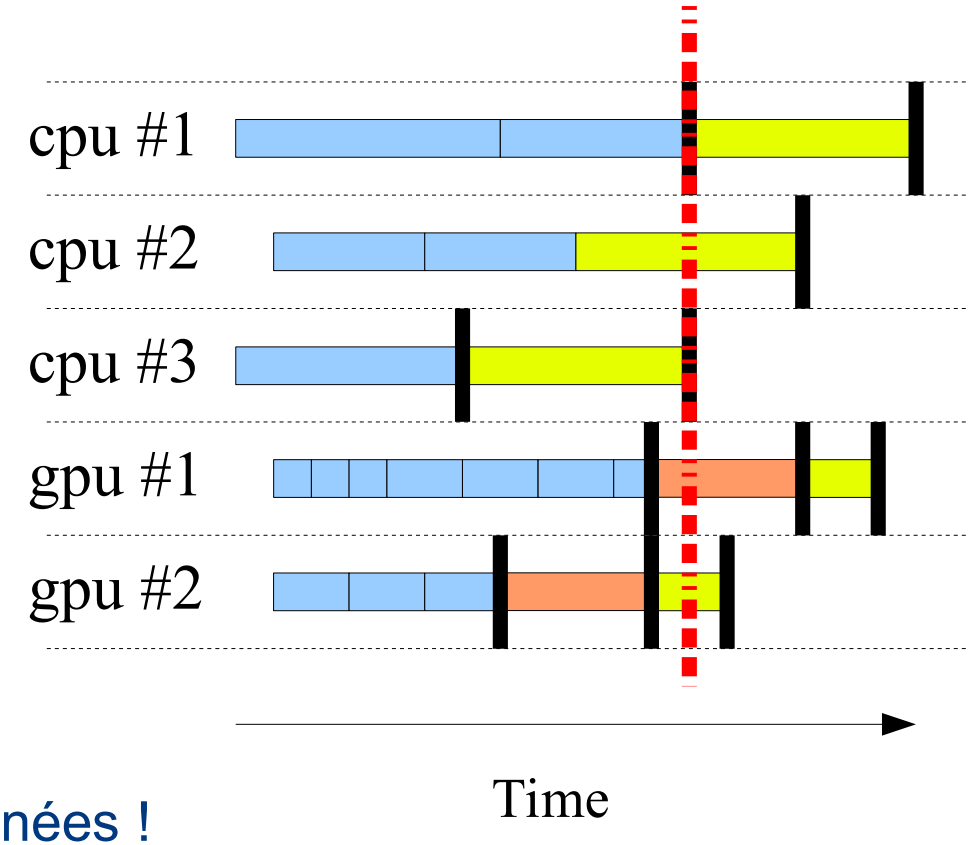
✔

Attention à la localité des données

✔

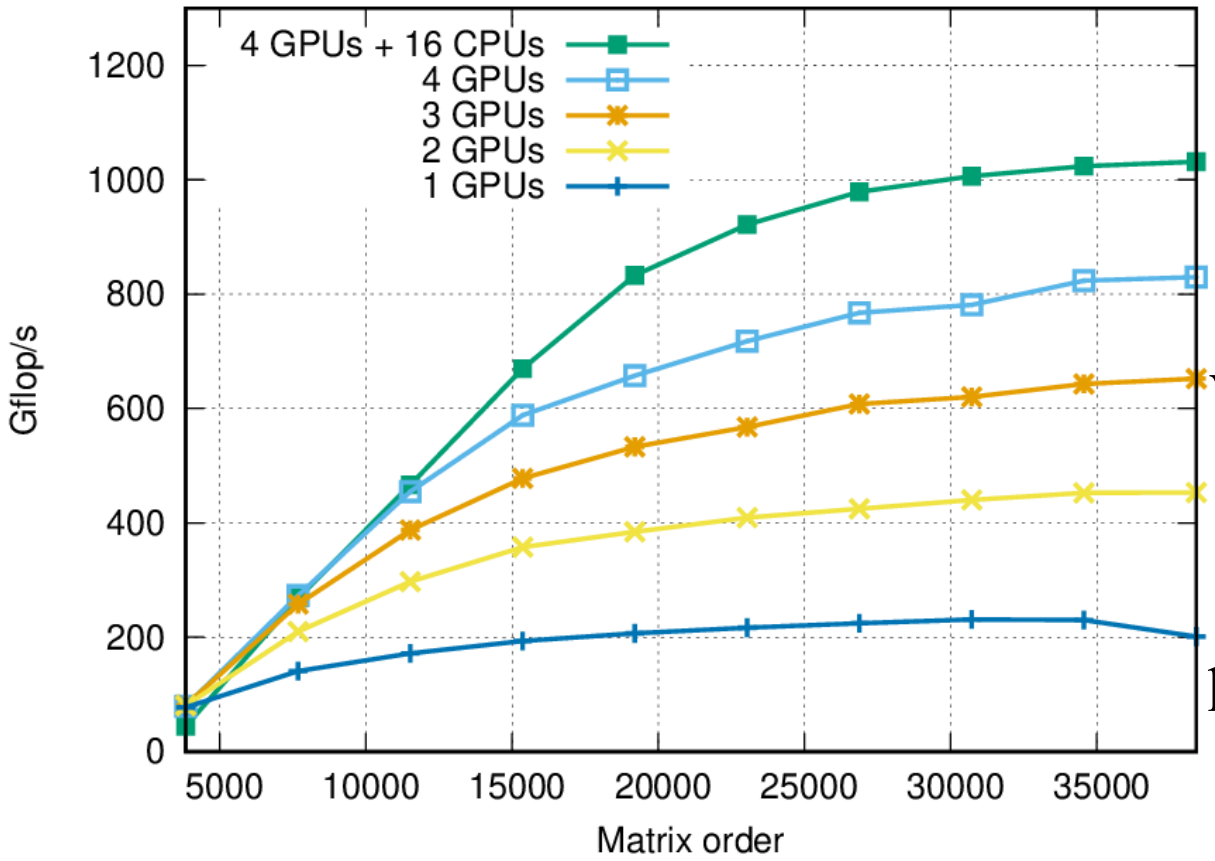
Peut anticiper les transferts de données !

- Prefetch



# Ordonnancement de graphe de tâches

- Décomposition QR
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)




+12 CPUs
~200GFlops

vs ~150Gflops mesuré!

**Grâce** à l'heterogeneité

# Ordonnancement de graphe de tâches

- Efficacité « Super-Linéaire » de QR?
  - Efficacité des kernels hétérogène
    - sgeqrt
      - CPU: 9 Gflops GPU: 30 Gflops (Speedup : ~3)
    - stsqrt
      - CPU: 12Gflops GPU: 37 Gflops (Speedup: ~3)
    - somqr
      - CPU: 8.5 Gflops GPU: 227 Gflops (Speedup: ~27)
    - Sssmqr
      - CPU: 10Gflops GPU: 285Gflops (Speedup: ~28)
  - Distribution des tâches observée
    - sgeqrt: 20% des tâches sur GPUs
    - Sssmqr: 92.5% des tâches sur GPUs
  - Tirer parti de l'hétérogénéité !
    - Faire seulement ce pour quoi on est bon
    - Ne pas faire ce pour quoi on n'est pas bon

# Et l’éviction des données ?

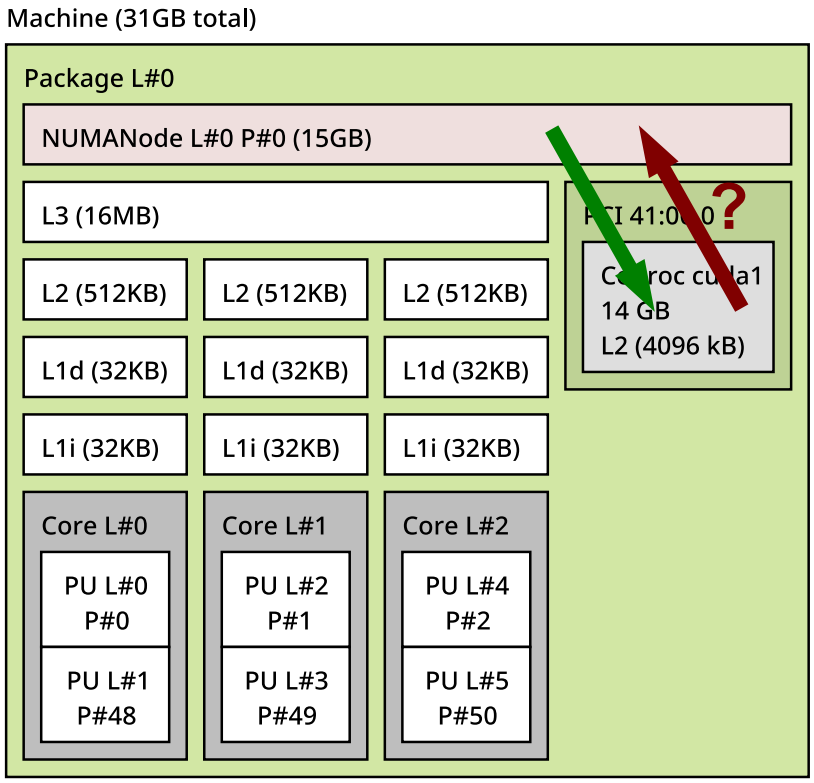
# Éviction des données

GPUs rapides

- Exécuter la plupart des calculs dessus
- Charger les données correspondantes

Quid lorsque sa mémoire est pleine ?

- Problème classique des OS
  - Disque/mémoire



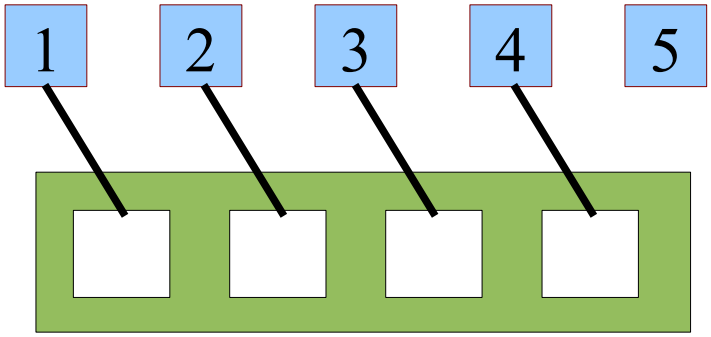
# Éviction des données

LRU

- Least-Recently Used
- Heuristique classique en OS

Mais aussi cas pathologique classique 😨

- Accès à D1, D2, D3, D4, D5, D1, D2, D3, D4, D5, ...
- Mais place pour 4 données seulement sur le GPU



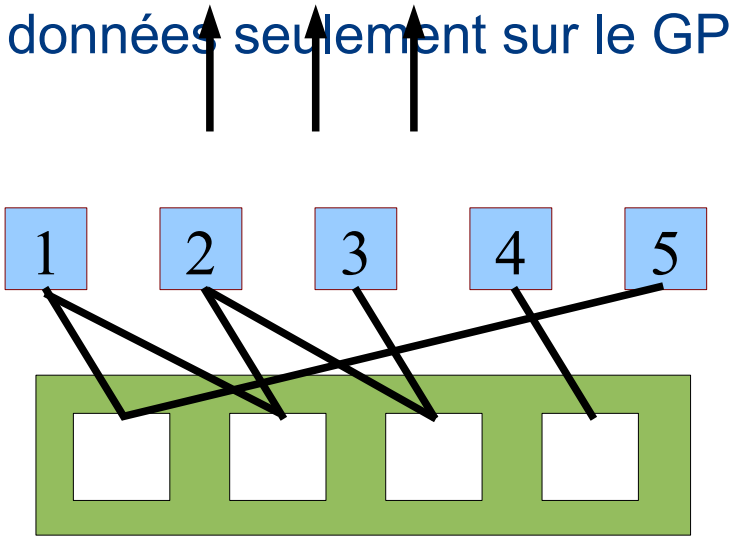
# Éviction des données

LRU

- Least-Recently Used
- Heuristique classique en OS

Mais aussi cas pathologique classique 😨

- Accès à D1, D2, D3, D4, D5, D1, D2, D3, D4, D5, ...
- Mais place pour 4 données seulement sur le GPU

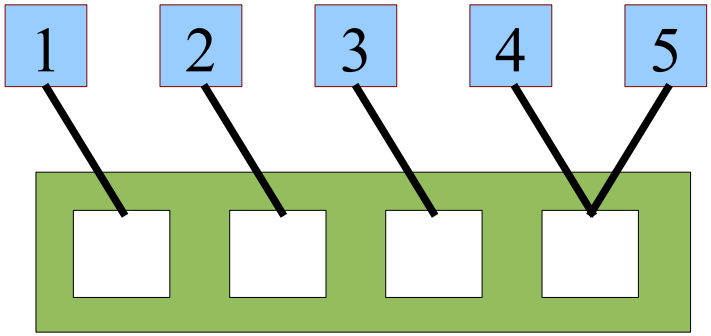


# Éviction des données

LUF

- Least-Used in the Future (~= madvise(COLD))

- Accès à D1, D2, D3, D4, D5, D1, D2, D3, D4, D5, D1, D2, D3, ...



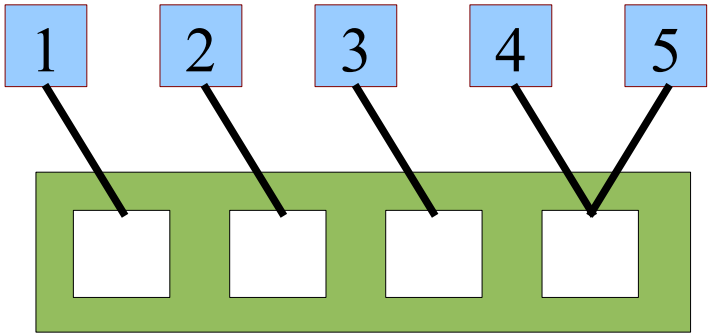
# Éviction des données

LUF

- Least-Used in the Future (~= madvise(COLD))

- Accès à D1, D2, D3, D4, D5, D1, D2, D3, D4, D5, D1, D2, D3, ...

- Besoin de connaître le futur
  - On le connaît ! C'est l'ordonnancement prévu



# Éviction des données

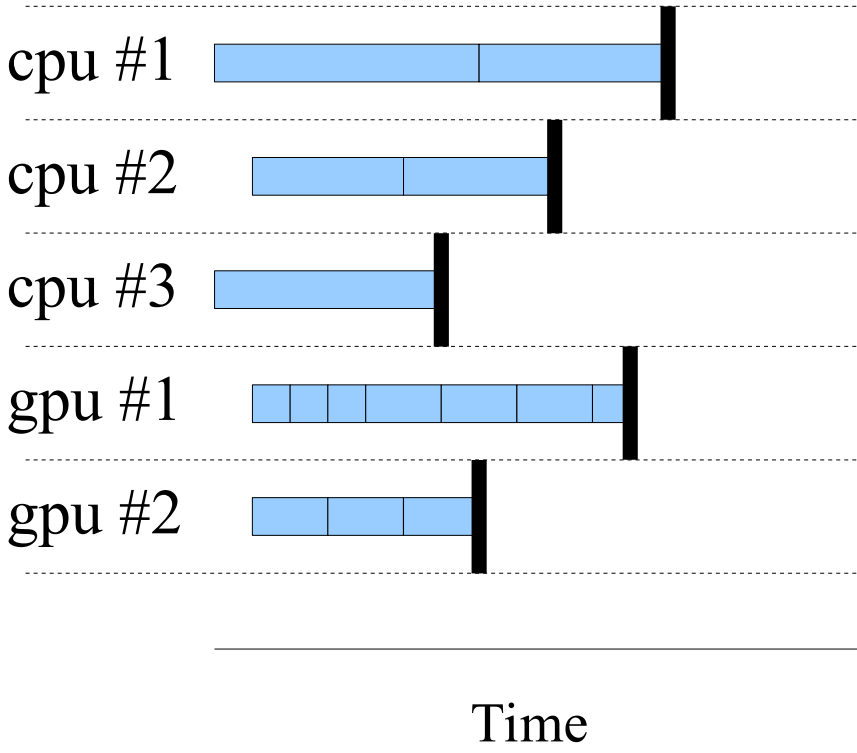
Ordonnancement : eviter les évictions

- Favoriser la réutilisation des données

~~Choisir la prochaîne tâche~~

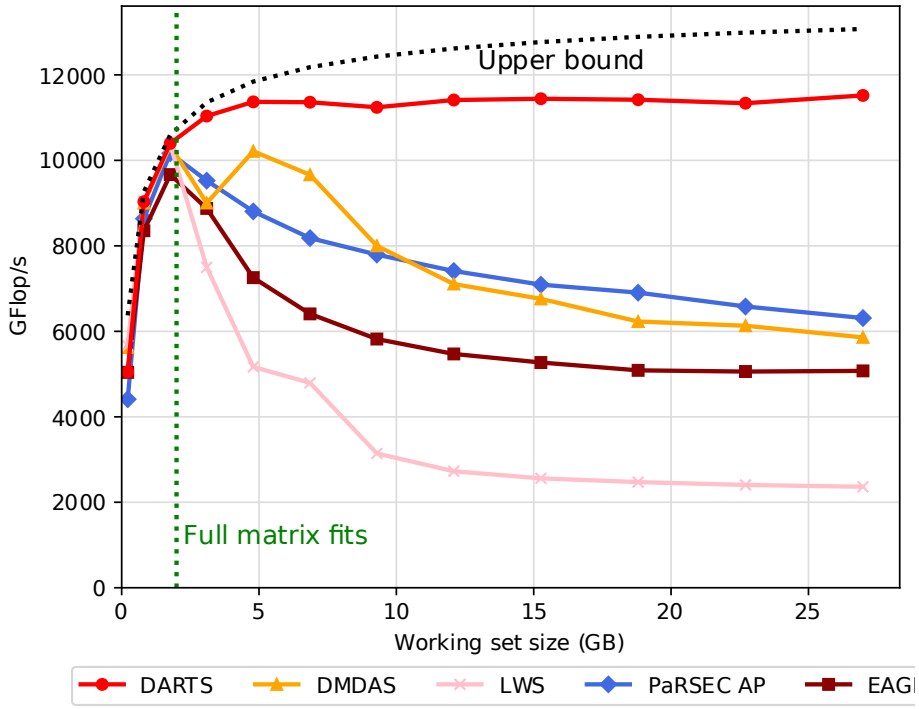
Choisir les données à charger

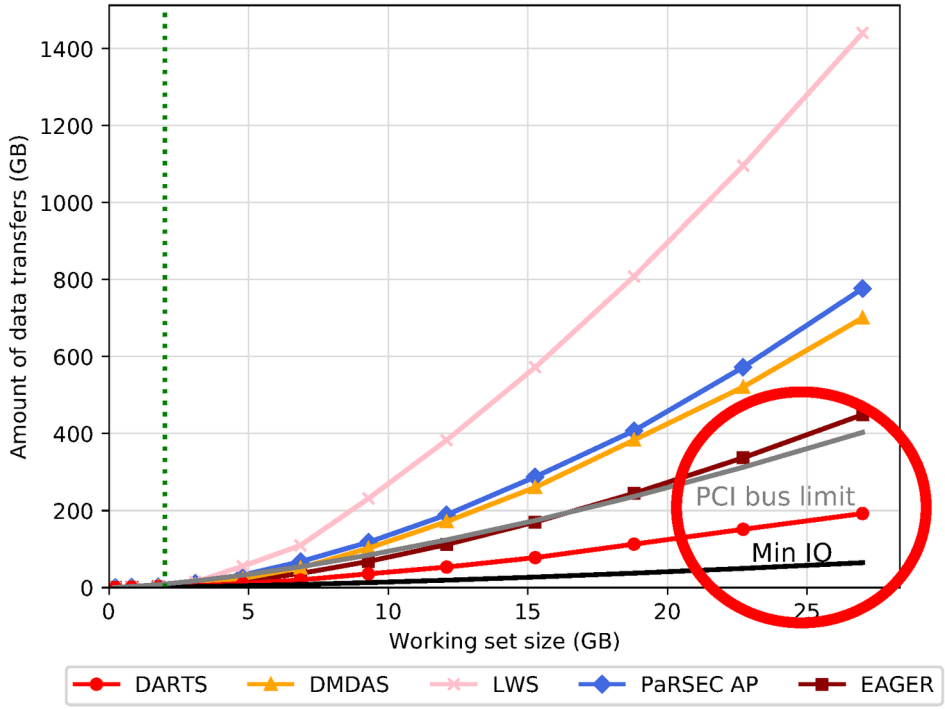
- Celles utilisées par le plus de tâches prêtes
- DARTS (Gonthier'23)
  - Dynamic Data-Aware Reactive Task Scheduling



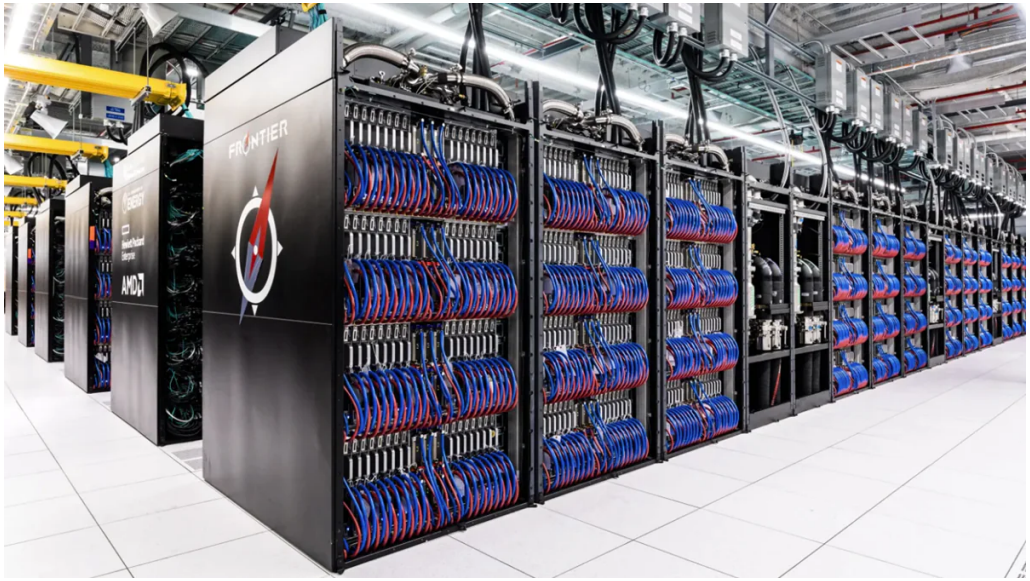
# Éviction des données

## DARTS



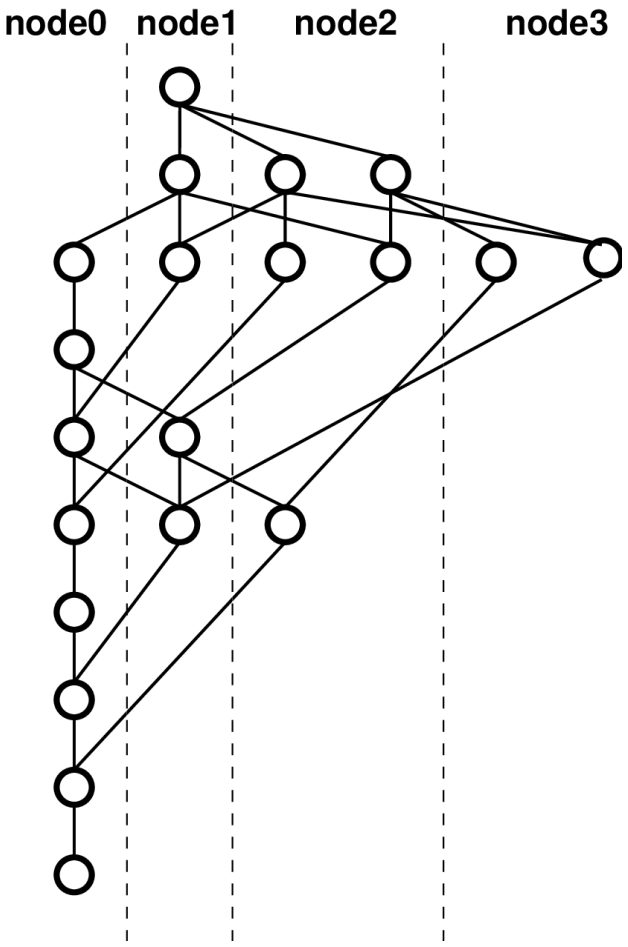
# Calcul distribué ?



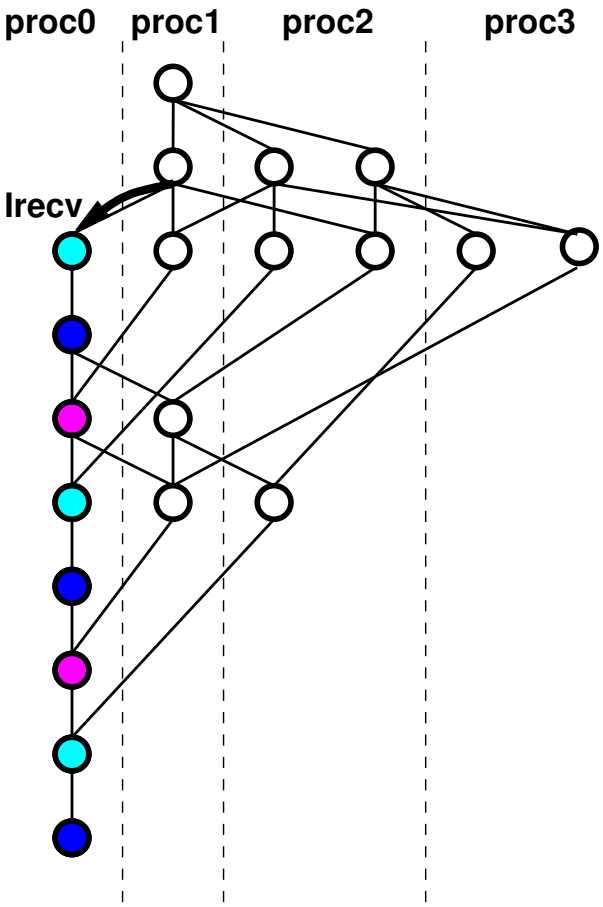
# Calcul distribué

Distribution du graphe de tâches sur les nœuds

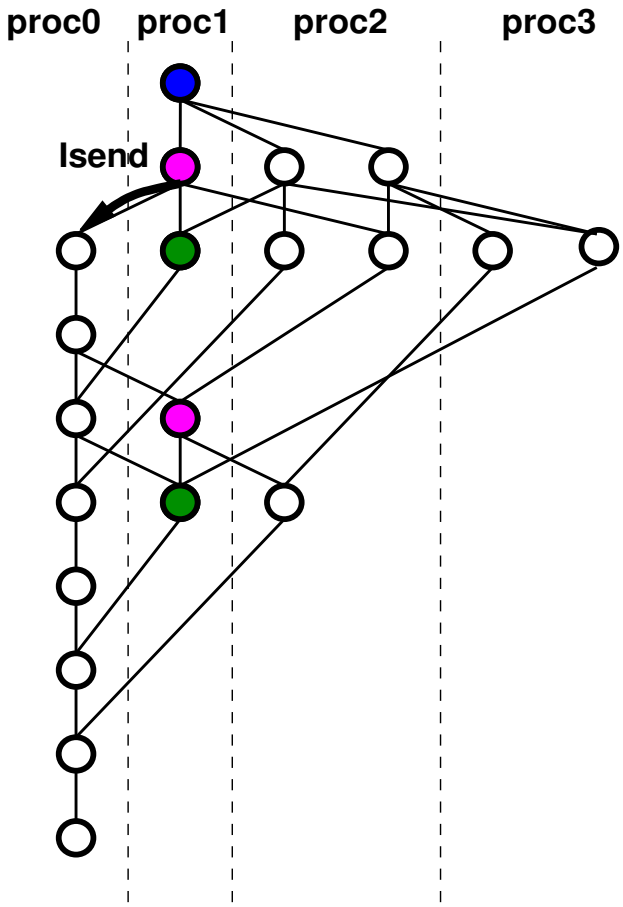
- À partir du placement des données
- De manière implicite
- De manière cohérente
- Sans coordination explicite



# Calcul distribué



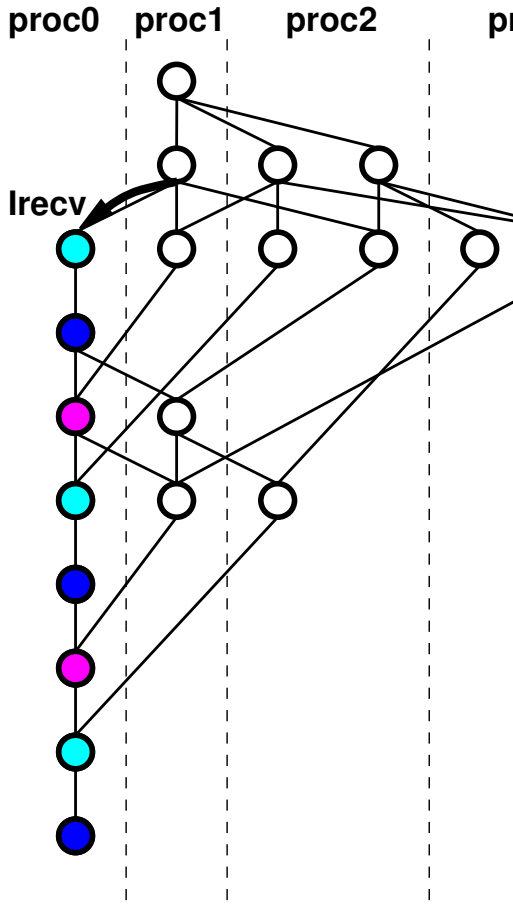

Node 0 execution




Node 1 execution

# Calcul distribué

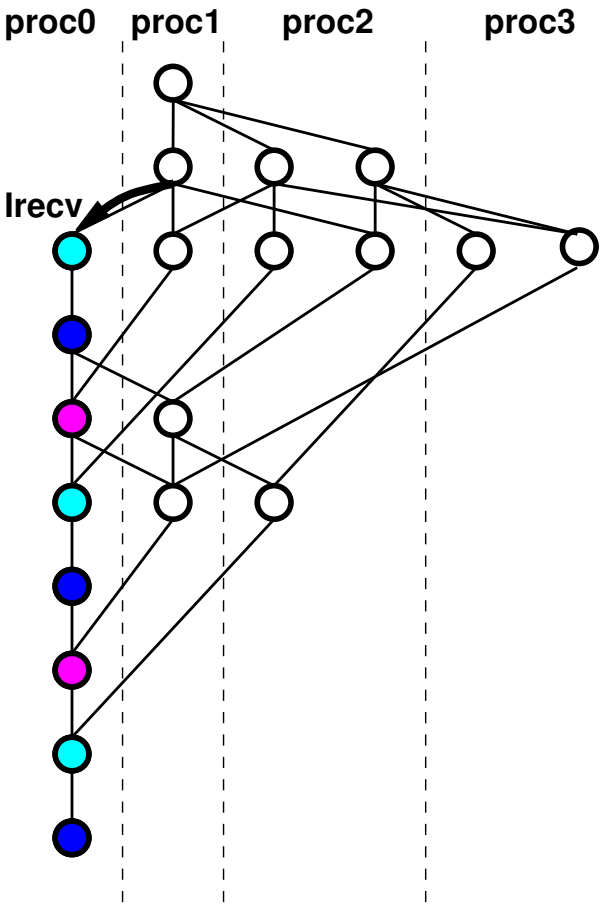
- Chacun sait à quoi s'attendre
  - Pas de *unexpected*
- Chacun peut se préparer à l'avance pour recevoir
  - Allouer la place sur le GPU
  - Envoyer les données directement du réseau au GPU



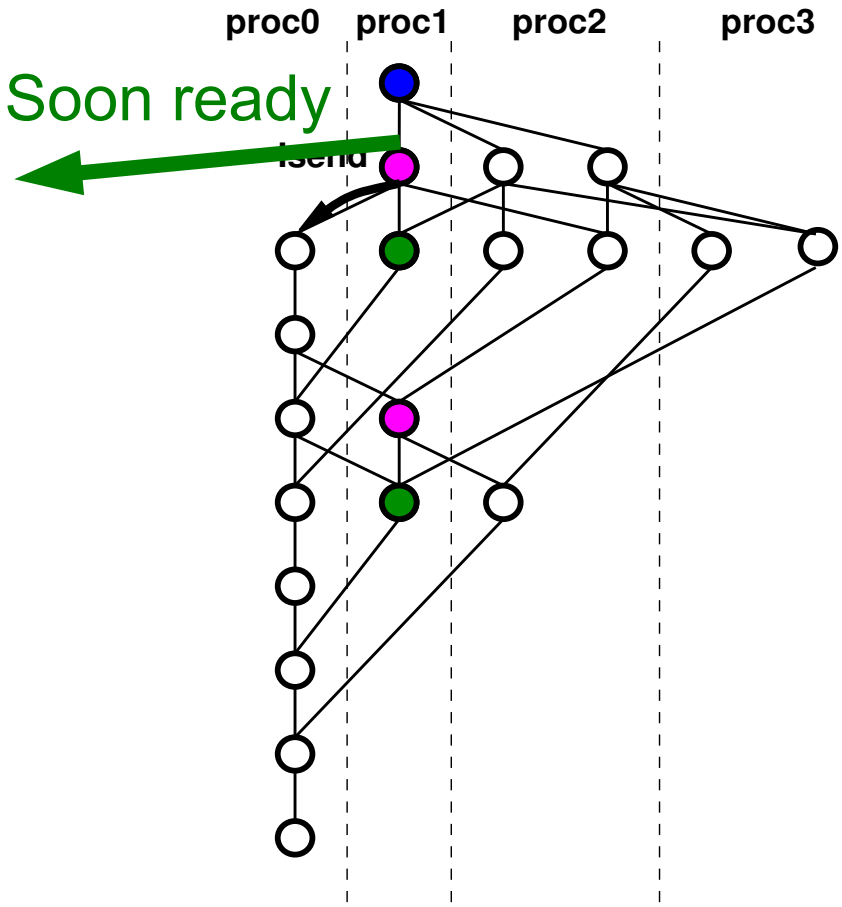

Faire mieux ?

- Allouer *pas trop* à l'avance
  - Éviter d'allouer tout !
- Anticiper les communications prioritaires
  - Ne pas démarrer une communication non prioritaire

# Calcul distribué




Node 0 execution

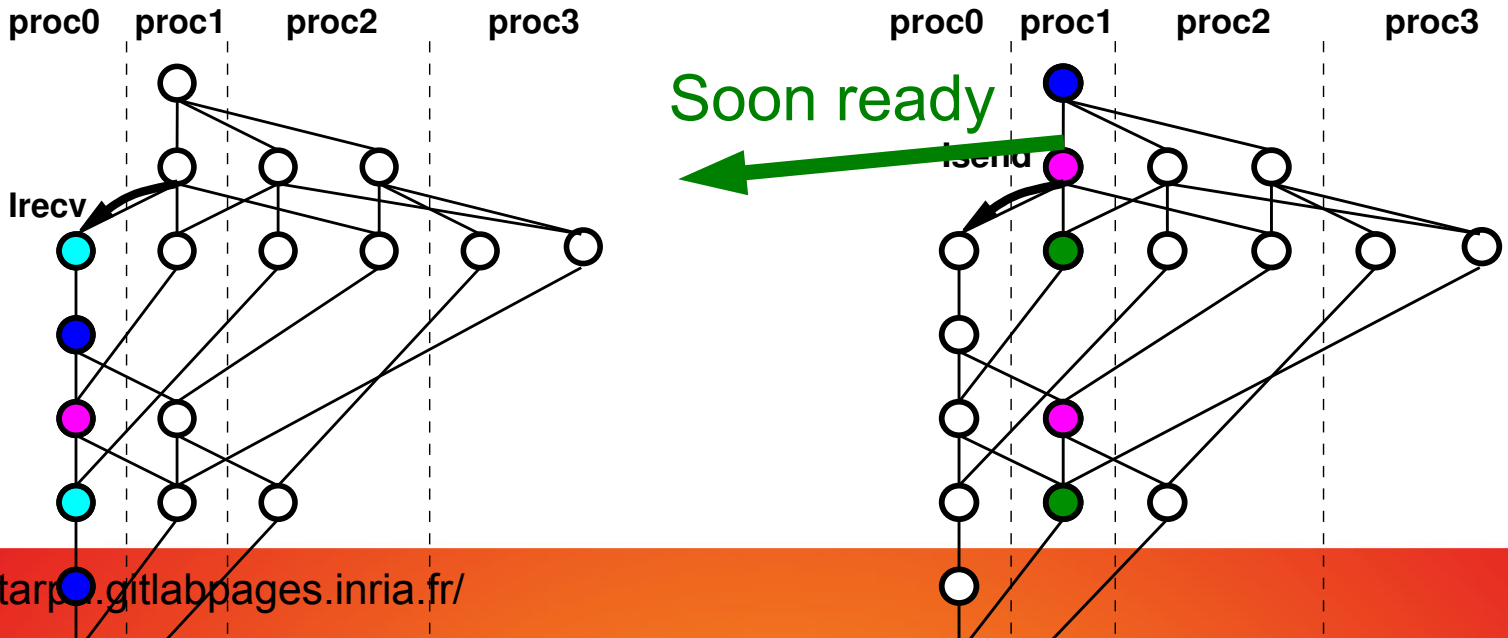



Node 1 execution

# Calcul distribué

## Soon ready

- L'émetteur anticipe le départ des données
  - Temps de terminaison de la tâche, quelques ms
  - Peut retarder des communications moins prioritaires
- Le récepteur peut anticiper la réception
  - A quelques ms pour allouer, verrouiller, ...
  - Charger d'autres données utiles pour les tâches débloquées



# Anticiper encore plus ?

Parcourir le graphe de tâches ?

- Cher
  - Des millions de tâches
- Gain modique

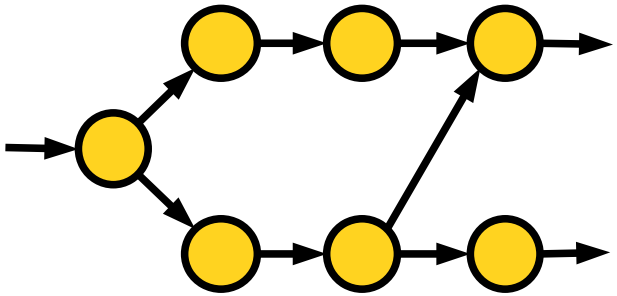
Plutôt une vision limitée

- Tâches *bientôt prêtes*
- Soumission progressive du graphe de tâches

# Conclusion

Graphes de tâches

- Une vision du futur
- Prédire le futur
- Choisir le futur
- Optimiser !



*Optimiser le futur en fonction de*
*la connaissance du futur prédit ou choisi*