Interconnect et MPI

Réseau d'interconnection

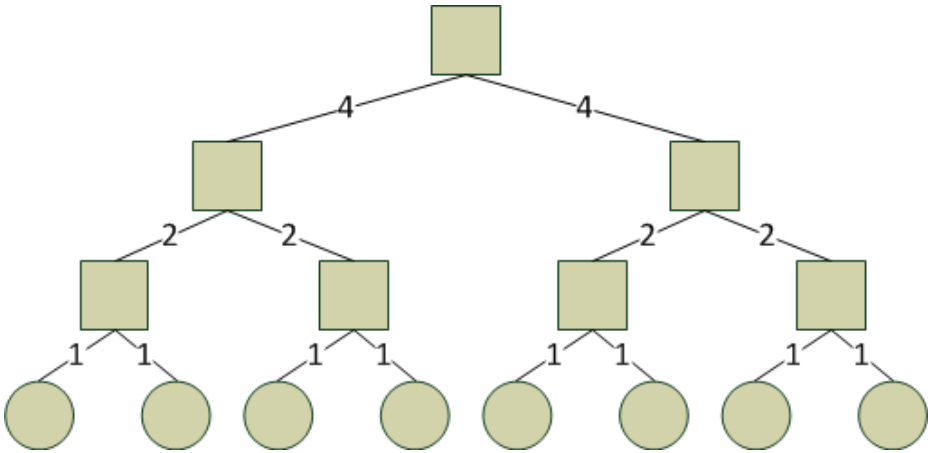
- Hiérarchie de switches
  - Chaque passage dans un switch (hop) induit un coup (latence)
  - L'agencement des nœuds et des switches suit une topologie
    - Réseau de clos (par exemple « fat-tree »), hypercube, …
    - Ilotage
  - Propriété d'un réseau : diamètre, bande passante de bissection
- Noeuds de calcul équipe de HCA
  - degré (ou *sortance*) des nœuds,
  - Bas niveau : latence, bande-passante, taux de messages (« issue rate »)
- La connectique !
  - Cables (types et longueurs)
  - « transceivers »
- Algorithme de routage
  - Statique, dynamique
  - Nombre de routes par lien ...
- Software
  - Switches/cartes : firmware
  - OS : drivers, bibliothèques de communication bas niveau
  - Applicatifs : MPI, Système de fichier (stockage)

Paramètres important d'un réseau

- **Diamètre** : le maximum des plus petits chemins entre deux noeuds (sur l'ensemble des paires possibles).
- **Largeur de bissection** : nombre de liens que l'on « coupe » en divisant un réseau en deux sous-parties égales.
- **Bande passante totale de bissection** : bande passante minimale soutenue disponible entre chacune des sous-parties.
- **Topologies courantes** : tore, hypercube, fat-tree, fat-tree « prunée » (réseau de Clos au sens général).

Ci-contre : fat-tree binaire

		N=8
Diamètre	$2\log_2(N)$	6
Largeur	N	8
Bande passante de bissection	NxB	8B



Performances bas niveau

Les plus communes :

- **Latence**
  - Ex. de performances bas niveau (perf. Infiniband QDR, PCIe gen 2)

		MPI measurements				lowlevel
		IMB benchmark		OSU benchmark		ib_rdma_lat
		latency	bandwidth	latency	bandwidth	latency
Configuration		usec	MiB/sec	usec	MB/sec	usec
internode	intersocket	1.54	2802	1.25	3281	1.09
	intrachassis	1.44	3137	1.40	3409	1.23
	Extrachassis	1.69	3132	1.66	3409	1.48

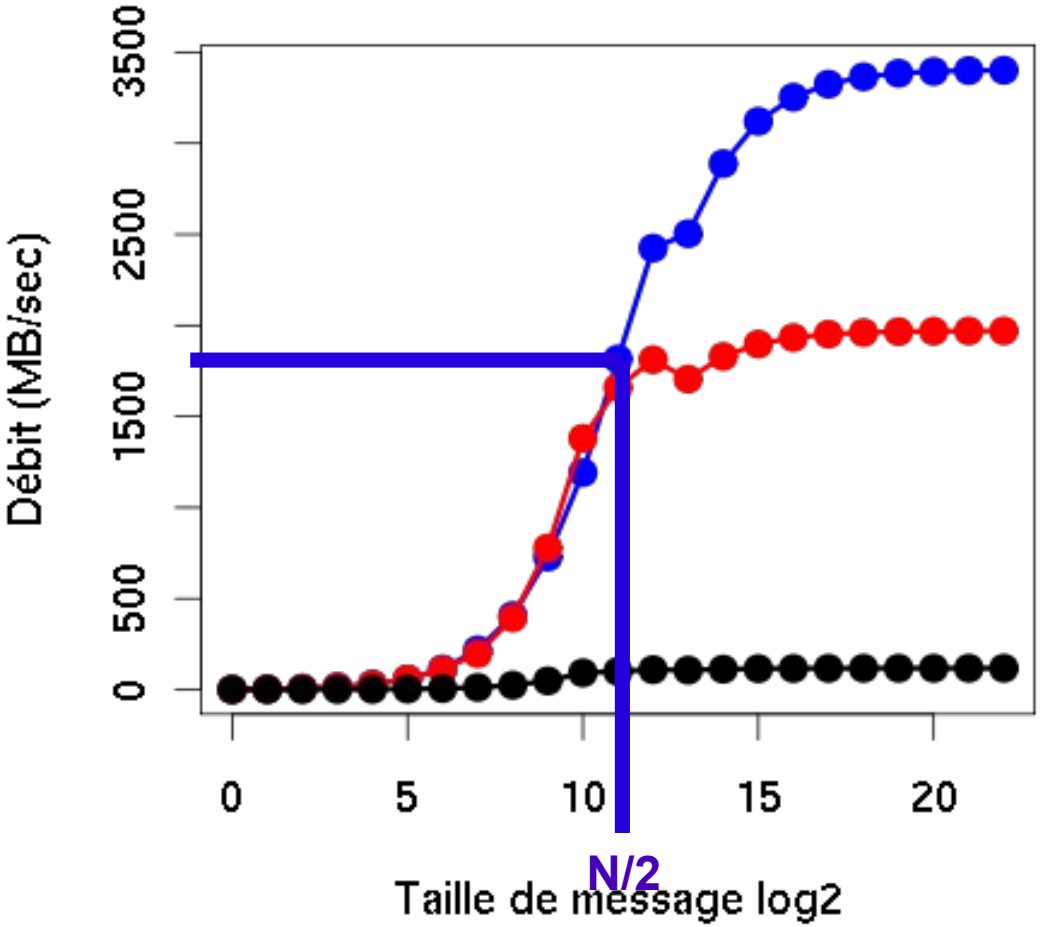
Latence de la carte : 1.1 µsec, latence d'un switch : 100-150 nsec, câble 100-150 nsec

- **Bande passante**
  - Exprimée en GB ($8*10^3$ bits) ou GiB ($8*2^{20}$ bits)
  - La bande passante effective est généralement plus petite que la bande passante physique « vraie » du lien et ce à cause de l' « overhead » du protocole (protocole généralement 8b/10b).
    - QDR : 40 Gbits – bande passante effective 32 Gbits
    - D'autre sorte de contention peuvent exister (ex. PCIe)

MPI

Performances point-à-point

- Indicateurs :
  - Latence
    - Petits messages
  - bande passante
    - Gros messages
  - N/2
  - taux de messages
- Permet de comparer les technos entre elles.

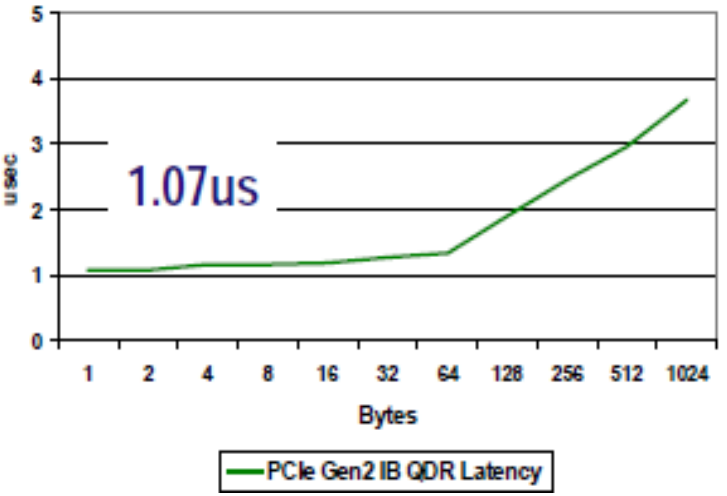


La bande passante ...

- D'un point de vue utilisateur applicatif : bande passante MPI
  - Les perf. bas niveaux sont intéressantes pour le stockage (SAN)
- Benchmarks élémentaires : PingPong, PingPing, SendRecv
  - Plusieurs implem. Disponibles : IMB (ex. pallas), OMB (OSU MPI Benchmarks) et les vôtres !
  - Bande passante d'un lien point-à-point (mono- et bidirectionnelle).
    - → Ne détecte pas de possible congestions : « pruning », routage ...
- Opérations point-à-point simultanées entre paires multiples
  - Montrent les contentions
- Attention : sur les SMP (!) un test sur l'ensemble des noeuds donne un résultat « moyenné » entre internoeud (interconnect) et intranoeud (« shared memory »).
  - Encore une fois, tout dépend ce que l'on souhaite mesurer
    - → 1 processus par coeur : c'est ce que va faire (en général) une appli. Utilisateur
    - → 1 processus par noeud : test l'interconnect eau travers de MPI.

… Mais pas que la bande passante

- La latence est une quantité primordiale pour les petits messages
  - Elle permet aussi de définir le taux de messages (0 ou 2 bytes)
- Les tests sur les opérations collectives sont intéressantes pour l'applicatif
  - Sensible à l'engorgement
  - MPI_Alltoall(v)
    - Code à base de FFT, par ex. les codes de chimie (cpmd)
- Le taux de messages est une quantité intéressante : il fournit une indication sur la capacité de traitement de la carte.



IMB

- http://software.intel.com/en-us/articles/intel-mpi-benchmarks/

- Intel MPI Benchmarks (ex-PALLAS)
- Portage
  - C
  - Immédiat
- Exécution
  - Simple

OMB

- http://mvapich.cse.ohio-state.edu/benchmarks/

- OSU MPI Benchmarks
- Bien que livré par défaut avec les implémentations MVAPICH, il fonctionne avec toutes les souches
  - Complémentaire d'IMB
    - Certains test intéressant : message_rating ...
- Portage
  - C
  - Immédiat
- Exécution
  - Simple

Contenu de la boîte à outil « interconnect-MPI »

- Des tests interconnect bas niveau
  - Stack soft de l'interconnect
    - Débit, latence, trace_route et autres outils de diag.S
- Des tests MPI
  - OMB
  - IMB
  - Tests
    - Point-à-point
    - Opérations Collectives et « message-rating »
    - Test fonctionnel de certaines « features » MPI
  - **Le benchmarks MPI sont un domaine de prédilection pour les benchmarks maison !**

Les temps de restitution ...

Les temps de restitution

- Tout le monde est pressé et veut des résultats tout de suite
- Un code est-il performant parce qu'il tourne vite ?
- Code parallèle : la parallélisation sert-elle uniquement à faire à ce que le code donne un résultat plus rapidement ?

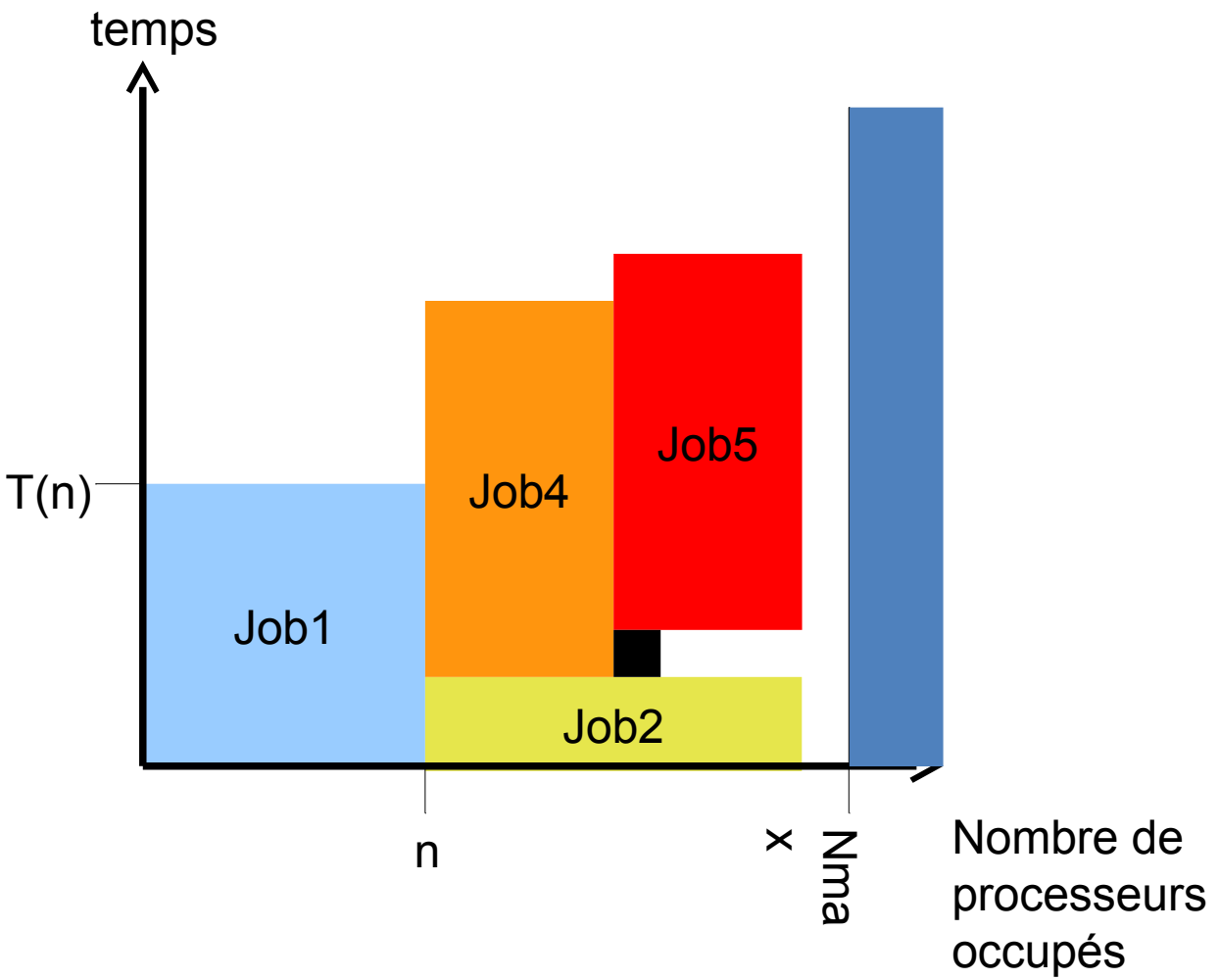
En conclusion

- La charge détériore les performances : stress de la bande passante
  - Mémoire (de 1 coeur à socket/machine pleine)
  - Réseau/Interconnect (au delà d'une machine).
- Il faut faire un compromis entre performance individuelle des codes et performance globale de la machine.
- Il faut toujours voir une machine et son utilisation sous ces deux dimensions :

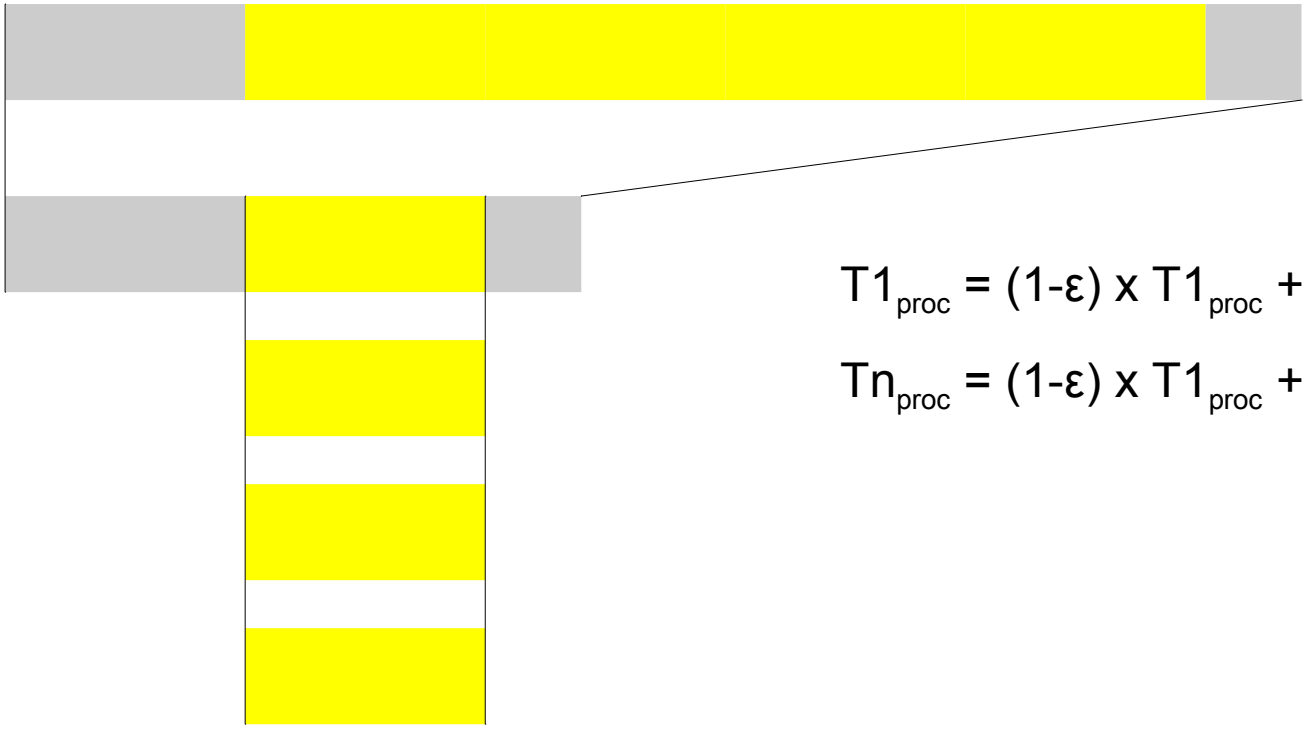
  - Le nombre de cœur
  - Le temps de restitution

  = queue d'occupation d'un « gestionnaire de batch »).
  - Le nombre maximum de processeur (axe des X) est fini : il est contraint par des aspects financiers, de place, de puissance électrique, de puissance dissiper. Les benchs aide au sizing de cette quantité.
  - Les temps de restitution des jobs ne sont que des « boites » qui remplissent cet espace.

Vision à 2D


temps
T(n)
Job1
Job4
Job5
Job2
n
x
Nmax
Nombre de processeurs occupés

Loi d'Amdhal



$$T1_{proc} = (1-\varepsilon) \times T1_{proc} + \varepsilon \times T1_{proc}$$

$$Tn_{proc} = (1-\varepsilon) \times T1_{proc} + \varepsilon \times T1_{proc} / n$$

- Loi d'Amdhal :
  - Déduire la part séquentielle, la part parallélisable
  - A très grand nombre de processeurs, le temps global de restitution temps vers le temps séquentiel ...
    - L'efficacité d'un code parallèle tient dans sa capacité à pouvoir réduire sa part séquentielle ...

Critique de la loi d'Amdhal

- Il y a plusieurs problèmes à cette loi :
  - Son domaine d'application est limité
    - On néglige tous les terme d'ordre supérieur à 1 dont les communications
      - → Négliger les communication n'est pas acceptable pour un code parallèle (en général!)
      - → A grand n sont sont ces termes qui domine vis-à-vis de 1/n ... la loi est dure mais c'est la loi ...
  - Les parties séquentielles (IO, initialisation) ne sont pas forcément elles-mêmes indépendante de n !
  - Il implique de l'utilisateur se focalise à **ressources constantes** sur la diminution sans condition de son temps de restitution, mais est-ce bien raisonnable ???
    - Argument de John L. Gustafson : « *it may be most realistic to assume that run time, not problem size, is constant* »
    - Maillage de 1000x1000x10000, en double précision représente 74 GiB de mémoire !
      Sur des machine de 8 coeurs avec 2 GiB/coeurs : au minimum 5 machines !!!

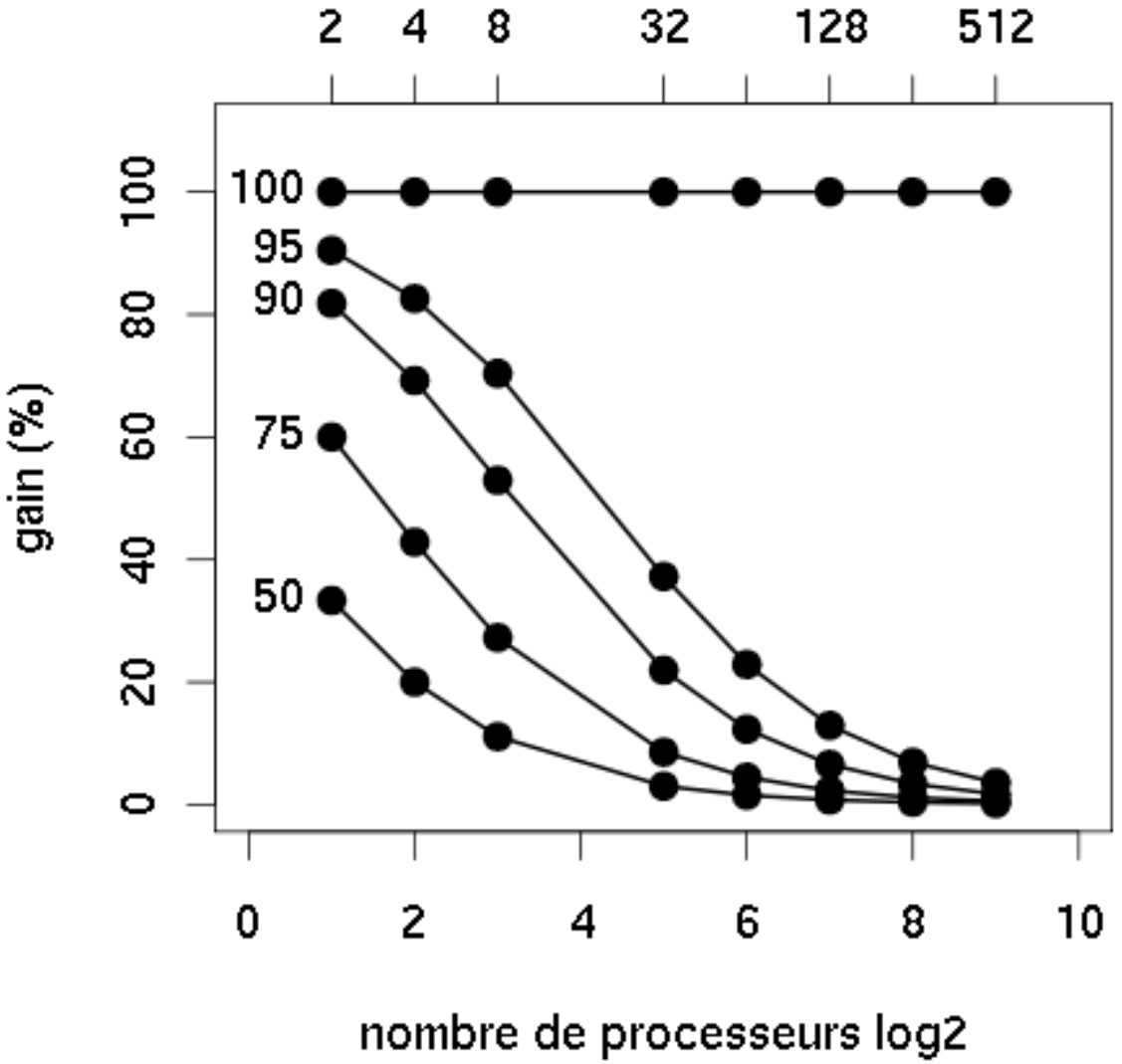
Passage à l'échelle

- Un paramètre important à déterminer dans le cas des applications parallèle est la « scalibilité »

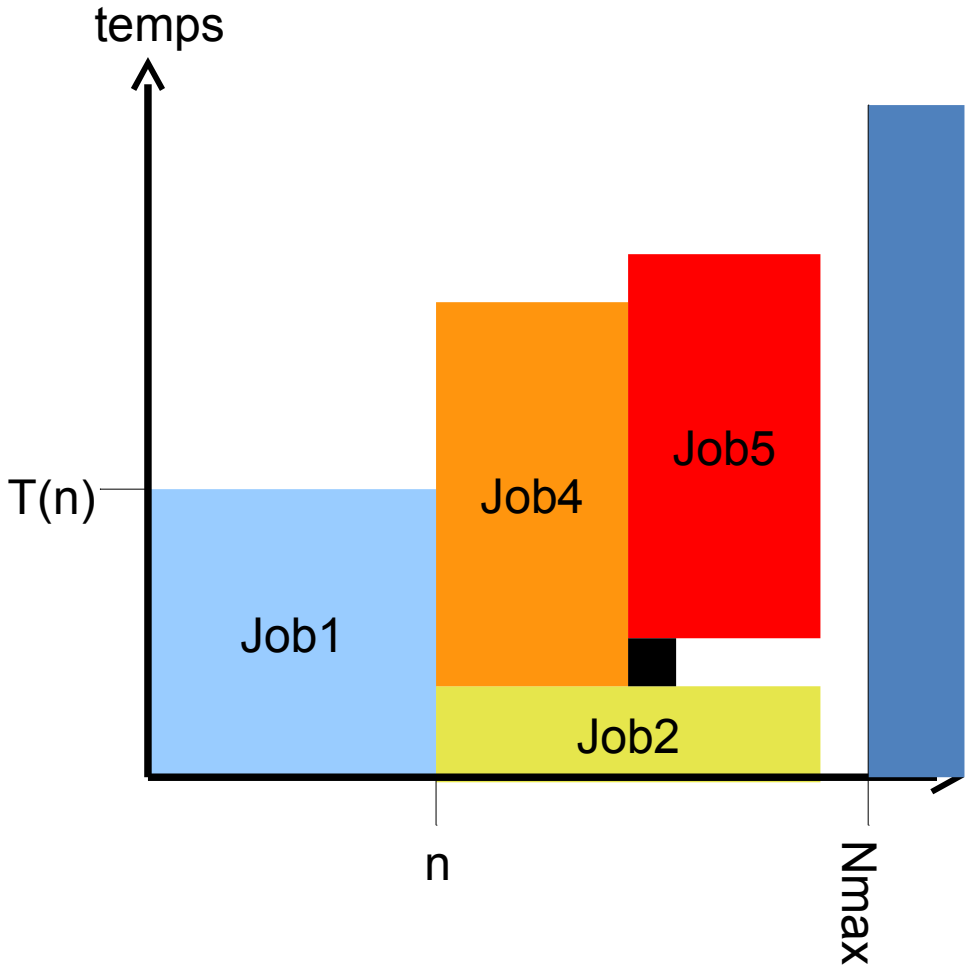
  $S(n) = T1_{proc}/T_{nproc}$

- Si l'on suit la loi d'Amdhal, elle temps vers 1/(1-ε) pour les n grands.

Scalabilité

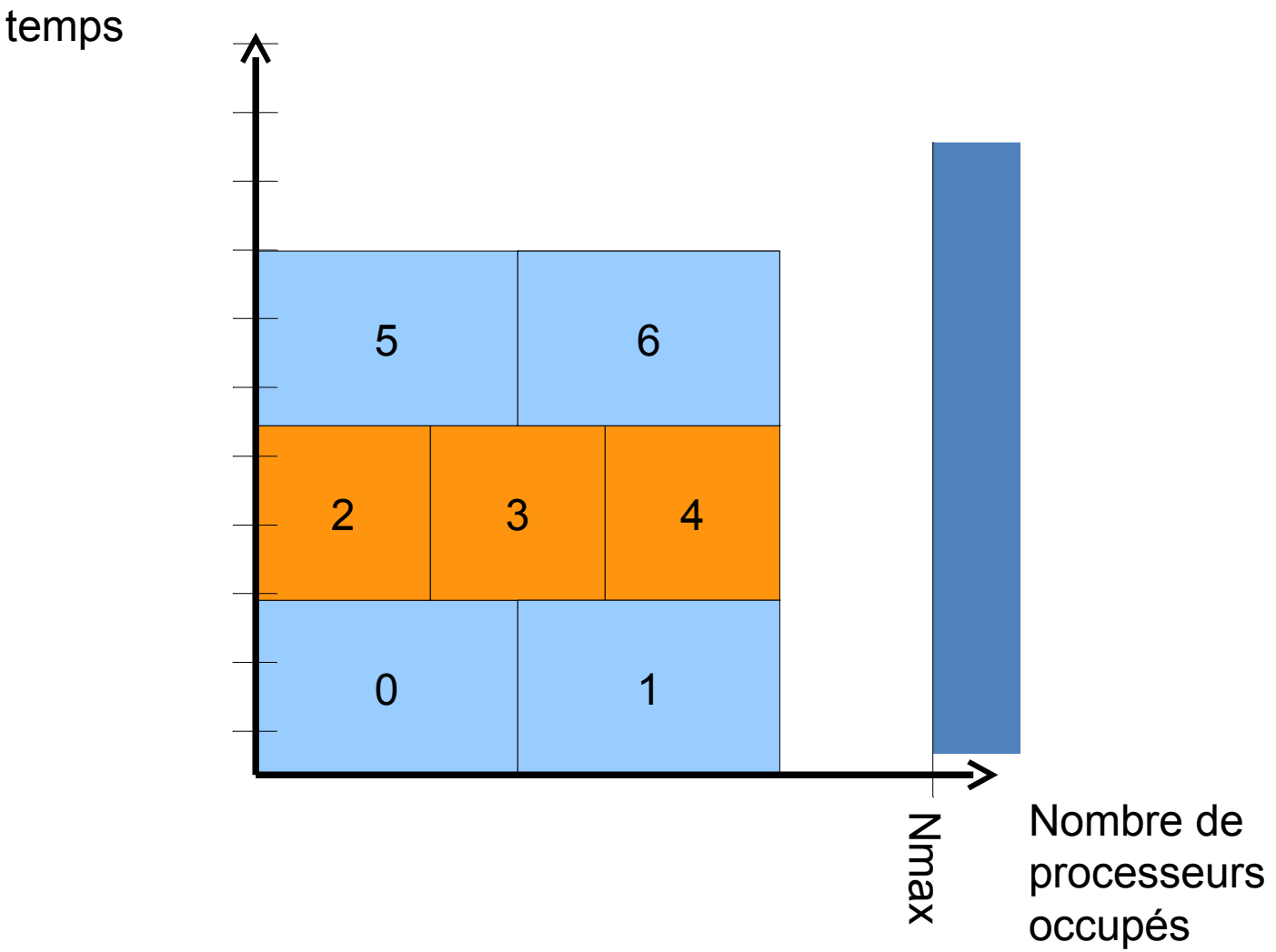


N

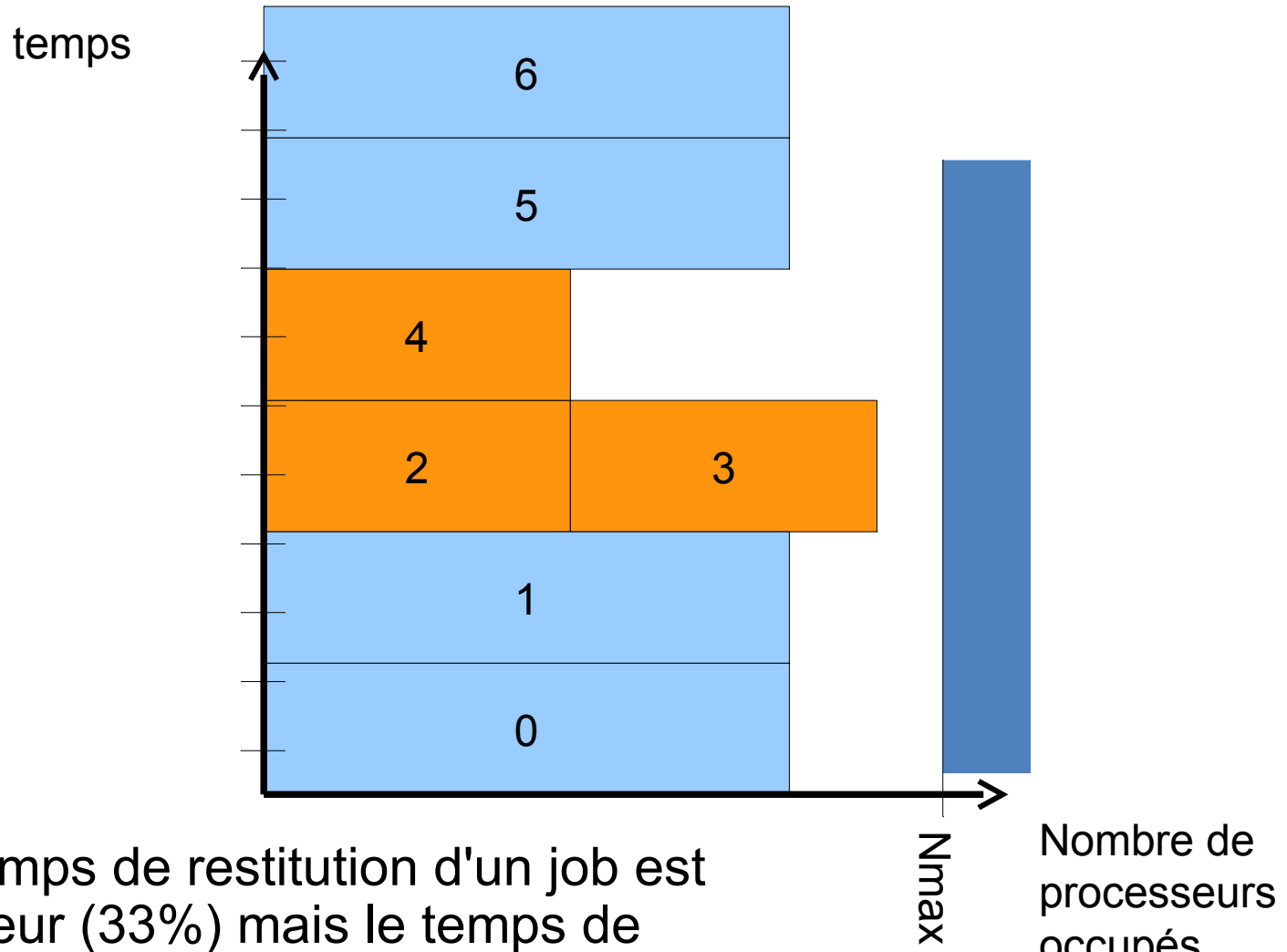


- Surface d'une boîte :
  $S(n) = n\,T(n) = n(1-\varepsilon)T1 + \varepsilon T1$
- L'idéal c'est S(n) soit constant avec n
  - Vrai si ε=1
- Problème :
  - L'utilisateur est tenté de réduire ces temps de restitution
    - Augmenter n !

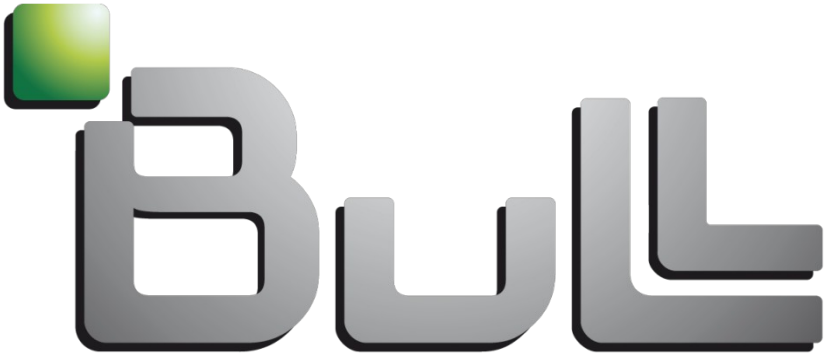
N



N



- Le temps de restitution d'un job est meilleur (33%) mais le temps de restitution de n jobs c'est fortement dégradé ...
- Il faut un compromis



Architect of an Open World™