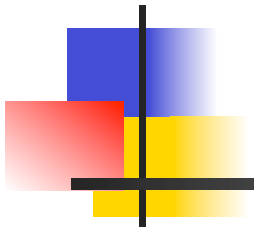


Supercomputers

Serge PETITON

serge.petiton@lifl.fr

17 octobre 2011

Master 2, 2011-2012

Outline

- A brief “history” of supercomputing, an introduction to the high performance computing,
- Flux parallelism, from SIMD to pipelined vector computing; algorithmic, linear algebra examples,
- Data parallelism, stencil and communication “compilers”, toward accelerator programming,
- Large granularity parallelism, from SPMD to GRID and Cloud computing,
- Sparse matrices and unstructured meshes parallel and distributed computing; data structure, algorithms and tools,
- From Petascale to Exascale computing; multi parallel paradigm programming
- High performance linear algebra computing; iterative methods and preconditioning,
- Hybrid Parallel and distributed linear algebra computing, MERAM and others methods
- Application Examples.

Experimentations on the BG IBM (Fortran/MPI at least) : TP + programming personal work (report with performance evaluation, in english) – Guy Bergère

Introduction

Computers are incredibly fasts, accurate, and stupid.
Humans are incredibly slows, inaccurate and brilliants.
Together they are powerful beyond imagination

Albert Einstein

Sommaire 17 octobre

- **Un peu d'histoire en guise d'introduction**
  - Du calcul vectoriel au GRID
  - Le Petaflop et les enjeux nationaux et internationaux
  - Le modèle de programation data parallèle pour accélérateurs matèriels
  - Exemple d'algorithmes data parallèles
  - Cas des structures de données irrégulières

Quelques reférences

- www.irisa.fr/orap
- www.idris.fr
- www.cines.fr
- www.netlib.net
- www.scidac.gov
- www.exascale.org
- www.top500.fr
- *A Data Parallel Scientific Computing Introduction*, Serge Petiton and Nahid Emad, Spriger Verlag, in Lecture Notes in Computer Science, Vol 1132, 1996
- *Vector Models for Data Parallel Computing*, Guy Blelloch, MIT Press, 1990
- *GPU accelerated computing - from hype to mainstream, the rebirth of vector computing*, Satoshi Matsuoka et al., Journal of Physics, Vol 180, 2009

Un peu d'histoire en guise d'introduction

Jadis

Calculs *à la main* (mauvaise précision),

calcul parallèle (Richarson en 1924 veut faire calculer 64000 personnes).

Moitié du siècle dernier, vers le Kilofloat

En route vers le calcul flottant,…Fortran,… IBM 360,….

Dès les années 70-80, l'ère du Mégaflop

Machines vectorielles, CDC 203.Cray 1, superscalaires, RISC, VLIW

Milieu des années 80, l'ère des centaines de Mégaflop

Machines parallèles, data parallèles (CM2 : parallèle avec processeurs vectoriels)

Début des années 90, l'ère des Gigaflop

En plus des machines parallèles, Grappes (clusters) et NOW

Milieu des années 90

Couplage de codes entre centres de calcul intensif, GRID

- Organisation virtuelle, contrôle des logins, identifications
- Co-méthodes et méthodes numériques hybrides

Circa 1997, vers les TéraFloat

Calcul global et à grande échelle

- Grand nombre de participants, calcul global, P2P
- Calculs parallèles et répartis

2008, le PetaFloat, Avril : Road Runner, IBM Cell

2009, GPU dans top500, Tsubame1

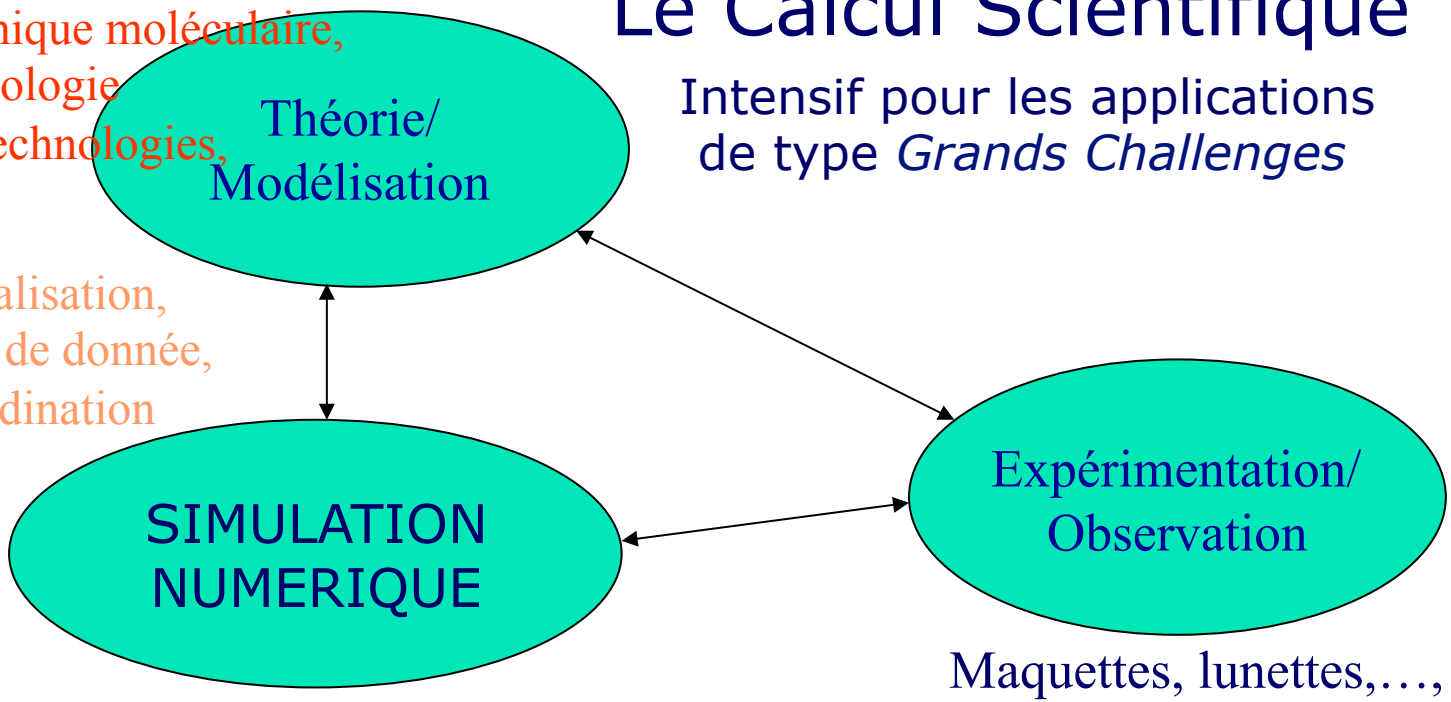
2010, 2.5 Petaflops, multi-core, GPU, Fat tree.

202x, Exaflops : roadmap en cours (IESP)

En attendant le Zettaflops et au-delà!

Le Calcul Scientifique

Intensif pour les applications
de type *Grands Challenges*



Dynamique moléculaire,
Climatologie
Nanotechnologies,
etc.…

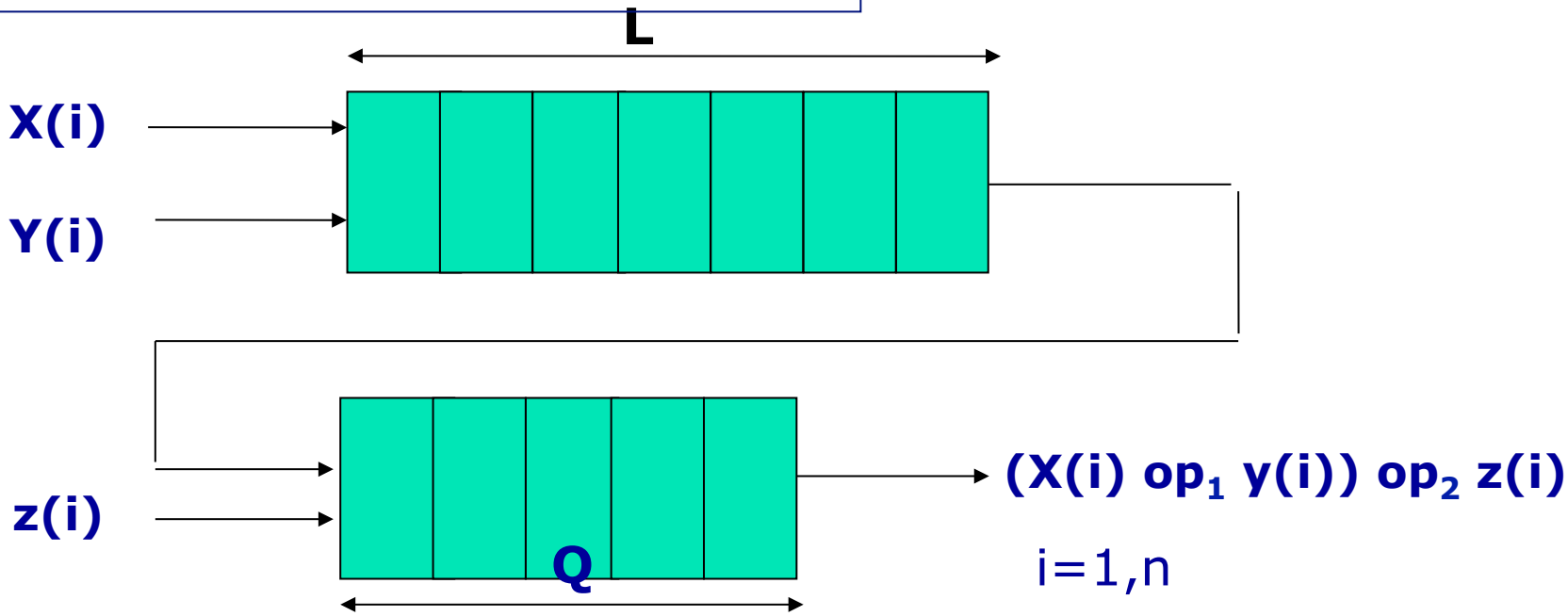
Visualisation,
Fouille de donnée,
coordination

Maquettes, lunettes,…, souffleries,
accélérateurs linéaire, etc.….

La simulation numérique, possible grâce aux puissances de calcul disponibles, change le procédé scientifique

Le calcul scientifique intensif haute performance demande des équipes interdisciplinaires et des moyens très importants

Du calcul vectoriel au GRID



$$t_{seq} = t_{horloge}\ n\ (\ L + Q\ )$$

$$t_{vect} = (L+Q-1)t_{horloge} + n\ t_{horloge}$$

Evaluation des performances

Performances crêtes

Performances soutenues

Performances «TOP500»

Débit asymptotique

N_{max}

$N_{1/2}$

Débit asymptotique =

1/période horloge

D'où la course à la plus petite période horloge.

Limites dues au coût, à la technologie et aux évolutions « politiques »

Article de Satoshi Matsuoka, architecte de Tsubame2

Abstract.

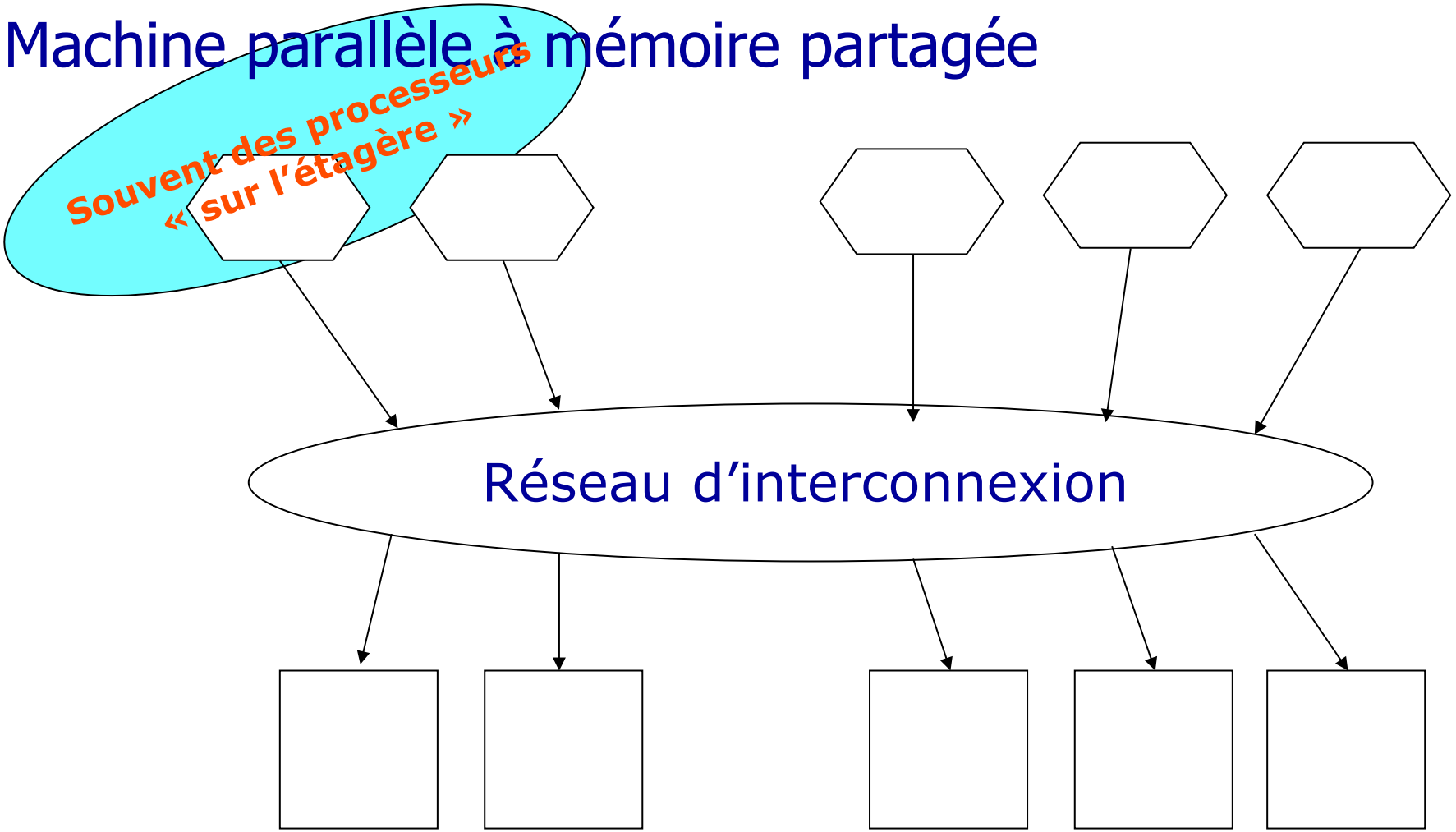
Acceleration technologies, in particular GPUs and Cell, are receiving considerable attention in modern-day HPC. Compared to classic accelerators and traditional CPUs, these devices not only exhibit higher compute density, but also sport significant memory bandwidth and vector-like capabilities to stream data at bandwidth of 100 GB/s or more. The latter qualifies such accelerators as a rebirth of vector computing. With large-scale deployments of GPUs such as Tokyo Tech's TSUBAME 1.2 supercomputer facilitating 680 GPUs in a 100-Teraflops scale supercomputer, we can demonstrate that, even under a massively parallel setting, GPUs can scale both in dense linear algebra codes as well as vector-oriented CFD codes. In both cases, however, careful algorithmic developments, especially latency hiding, are important to maximize their performance.

Et :

In fact, both GPUs and Cell can be regarded as a rebirth of vector computing. Although different from traditional architectures with deep vector pipelining, nonetheless both architectures are quite amenable to where vector computers excelled in the past. In fact, modern algorithms intended to run on cache-oriented standard CPUs often do not work well due to their requirements for shorter memory access latency and locality enabled by extensive caching and associated blocking. Instead, classic algorithms intended for long vector-length "streaming" memory access work quite well.

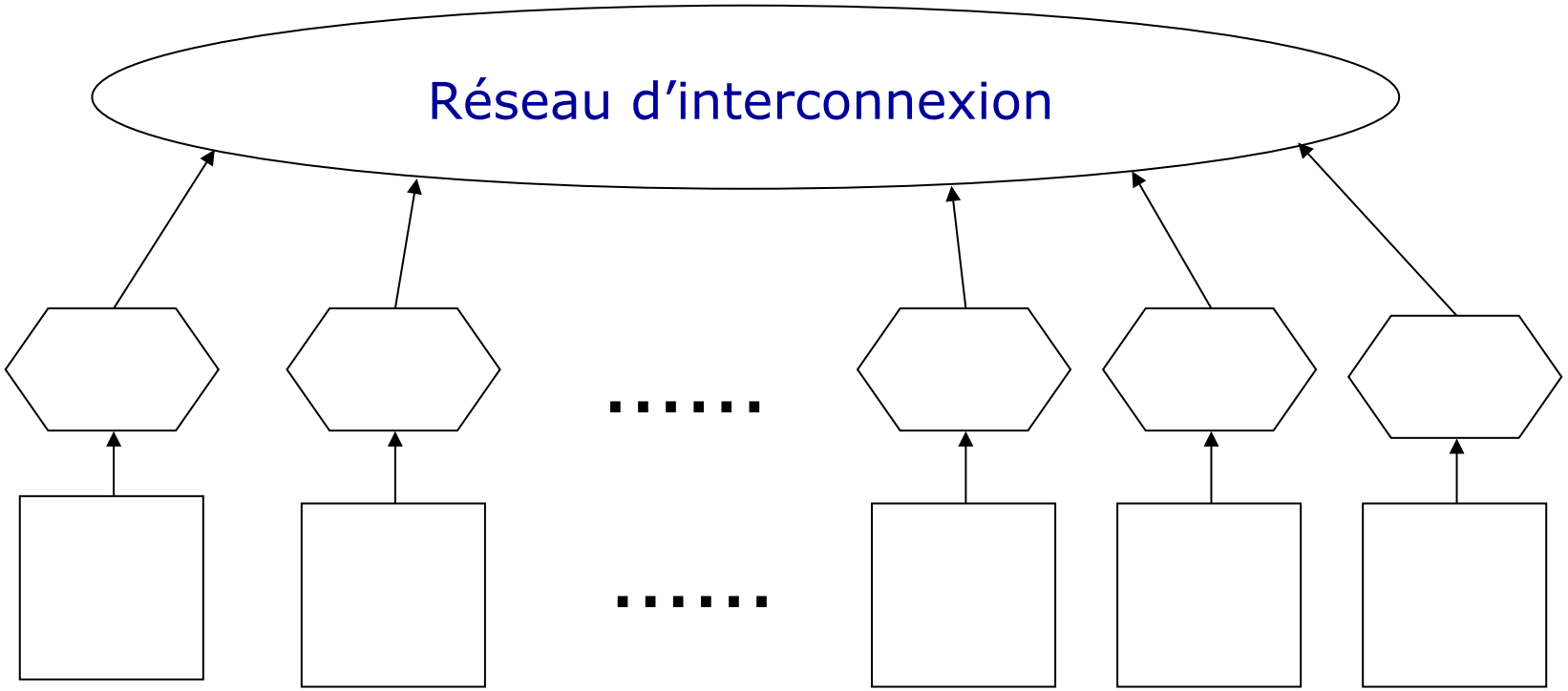
BTW, plutôt « data parallèlisme »

Machine parallèle à mémoire partagée



Uniform Memory Acces (UMA) architectures.
Programmation en MPI ou OpenMP principalement

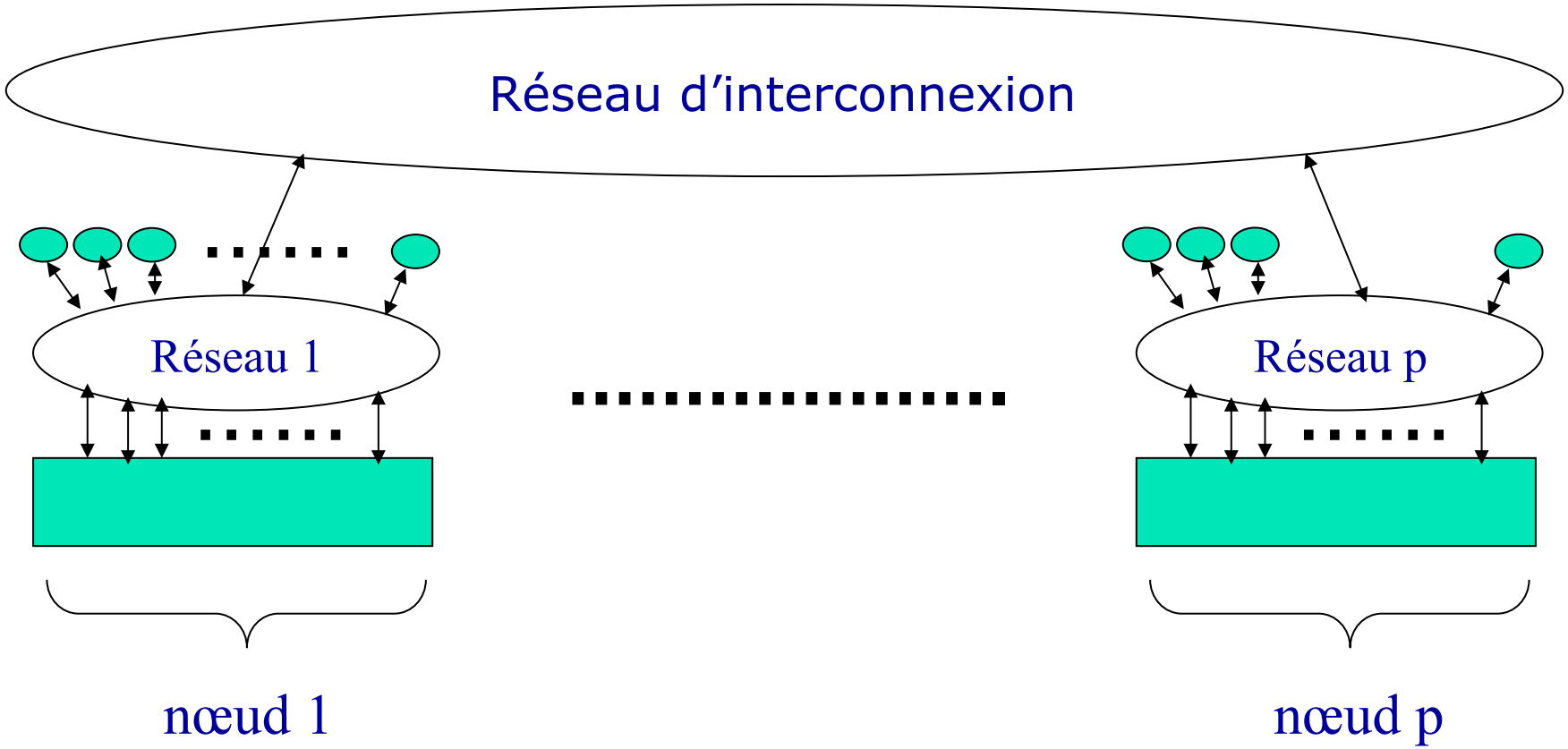
Machine parallèle à mémoires distribuées



Non Uniform Memory Access (NUMA) architectures :

Programmation en MPI principalement, avec langage data parallèle

(Massively Parallel Processors) MPP

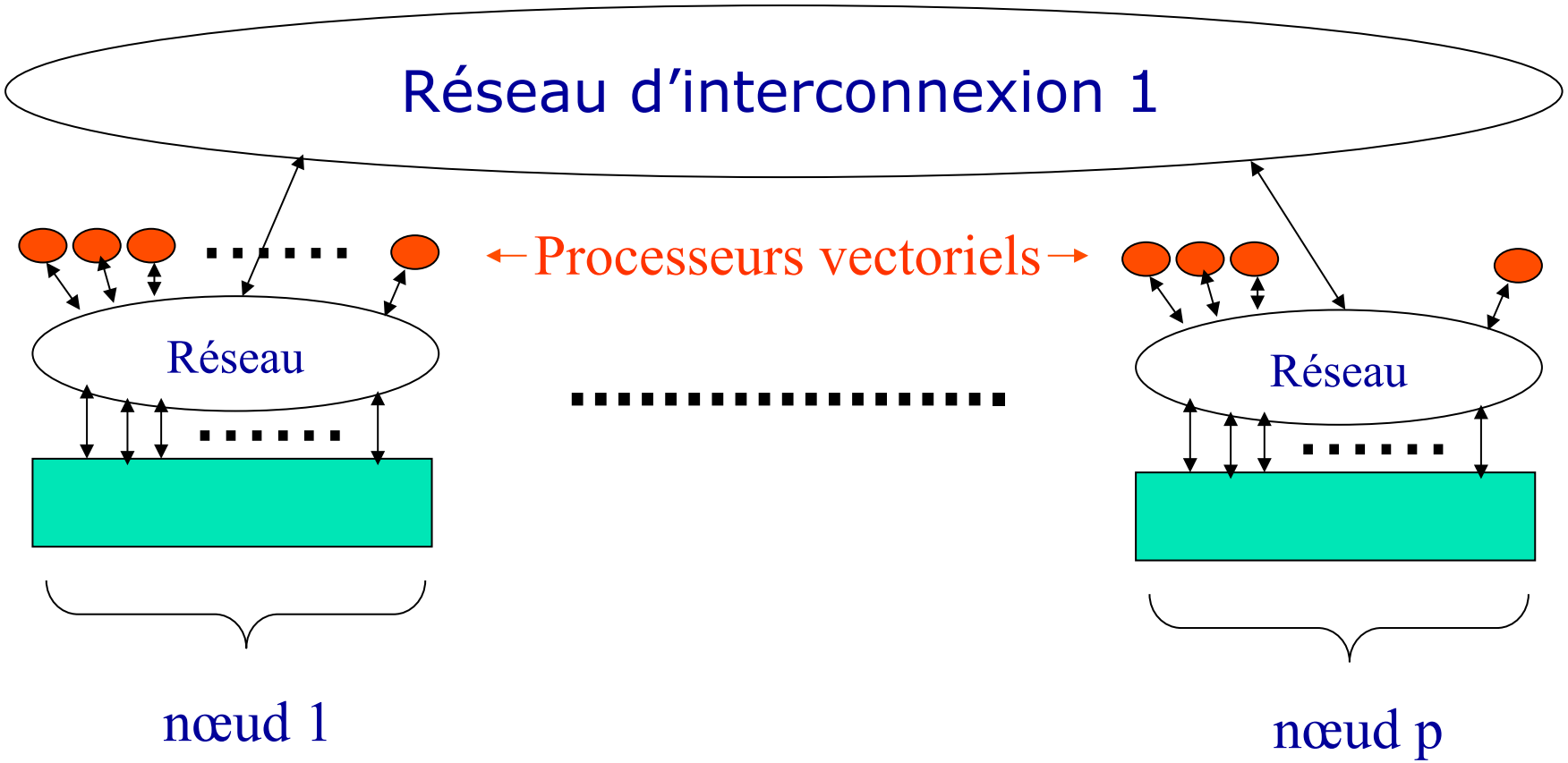


IBM SP4 par exemple, programmation en MPI et/ou OpenMP principalement

MPP/*Earth Simulator*

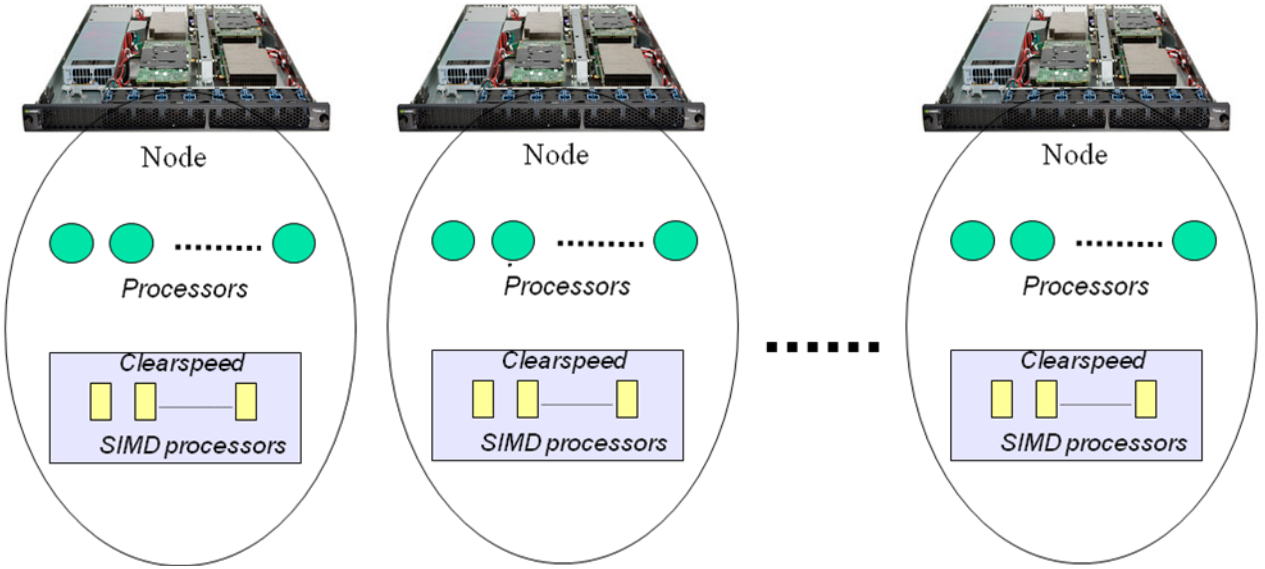
(computenick)

Longtemps Numéro 1 mondial



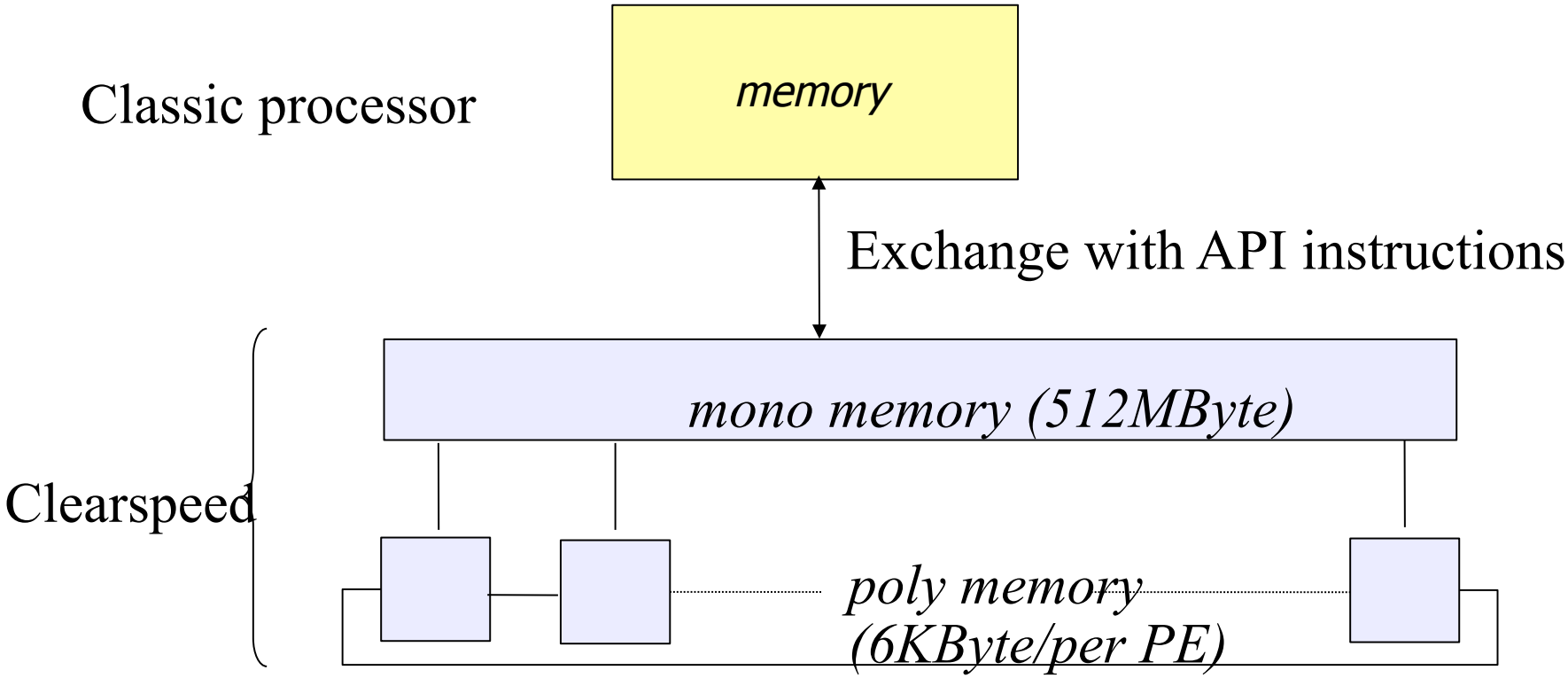
Programmation en MPI et/ou OpenMP ou en HPF

Accélérateurs pour nœuds multi-processeurs, multicores, hybrides

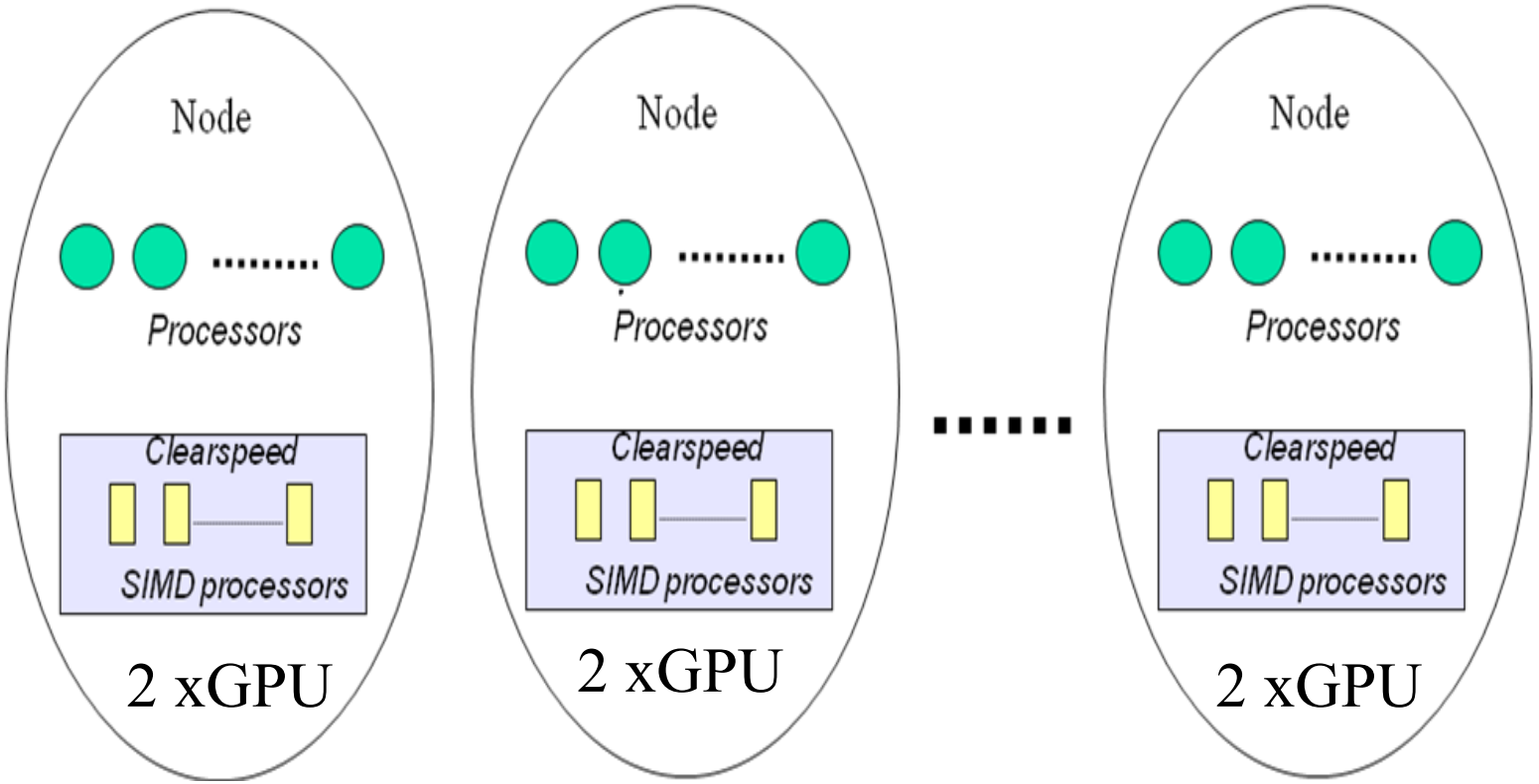


TSUBAME 1.2 ; première machine à entrer dans le top 500 avec un cluster GPU NVIDIA.

Levels of Memory on Clearspeed



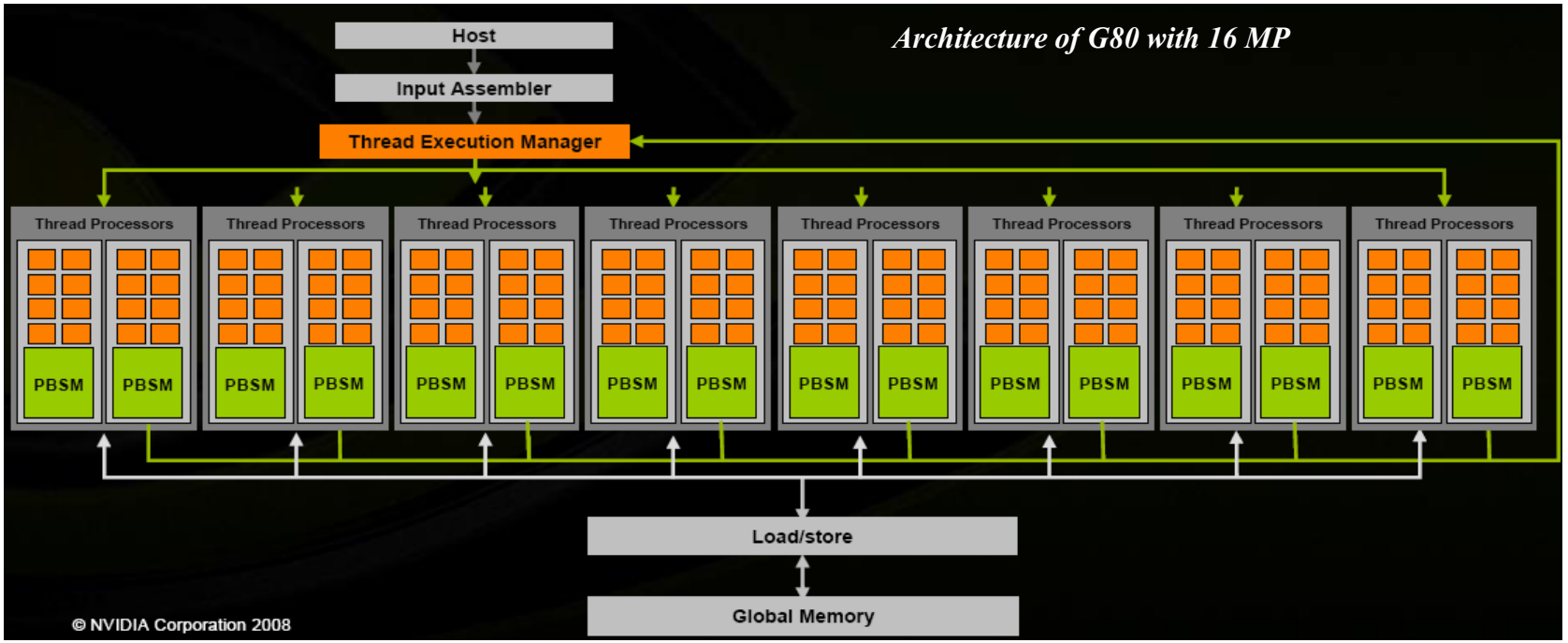
Tsubame 1.2 puis 2, Titech, Japan



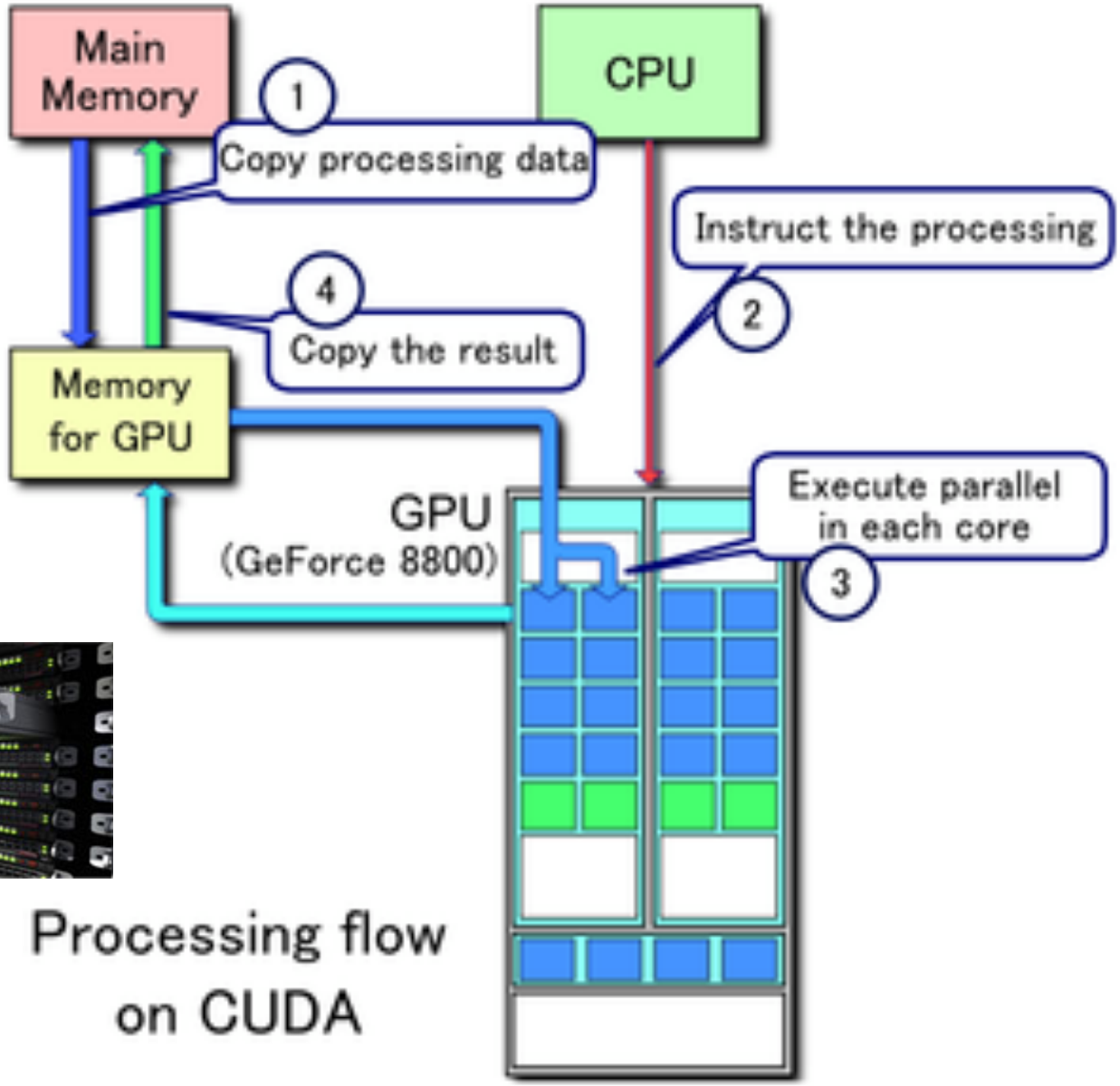
Première machine à entrer au TOP500 avec des GPU il y a 2 ans environ

GPGPU Architecture Overview

- The latest GPU generation may be composed of dozens of multiprocessors
- Each multiprocessor has severa scalar cores
- Each scalar core can execute several (T10 : 32) threads, named warp
- In a single memory instruction : (T10 example):
  - A 32, 64 or 128 bit word is read
  - A half-warp, 16 threads, has a simultaneous memory access

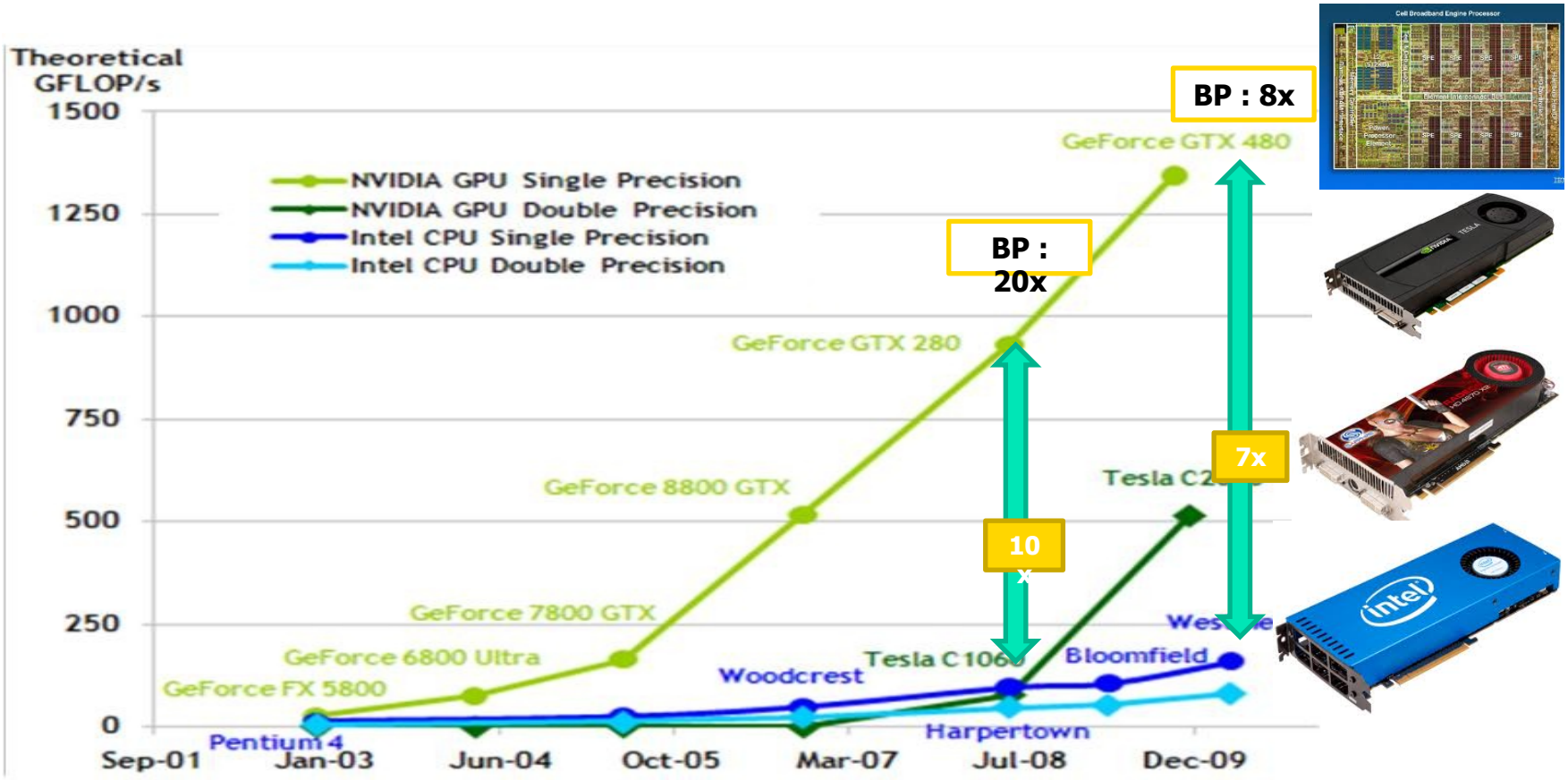


CUDA and GPU



Processing flow on CUDA

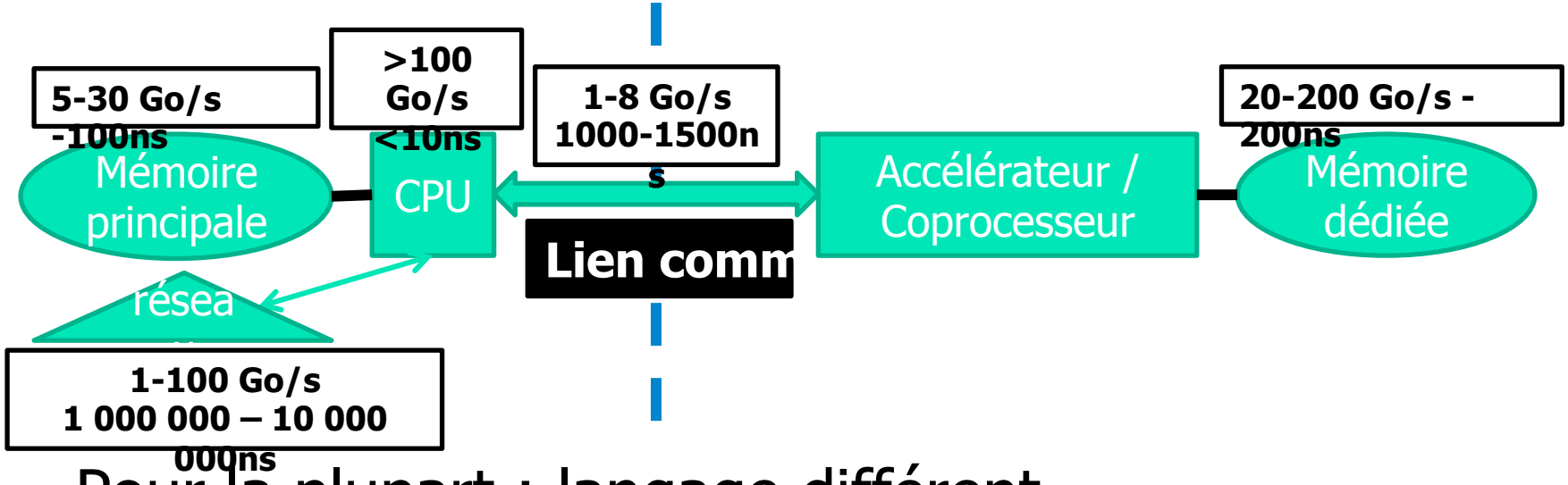
L'écart processeurs/accélérateurs (slide Jerome Dubois)



- **Roadrunner**-2008-IBM Cell / **Thiane-1**-2009-AMD 4870X2 / **Nebulae**-2010-Nvidia C2050 / **Thiane-1A**-2010-Nvidia C2050 / **TSUBAME2**-2011-Nvidia M2050 / **Stampede**-2012-Intel MIC,…
- Et présent dans le grand public : cet ordinateur portable!

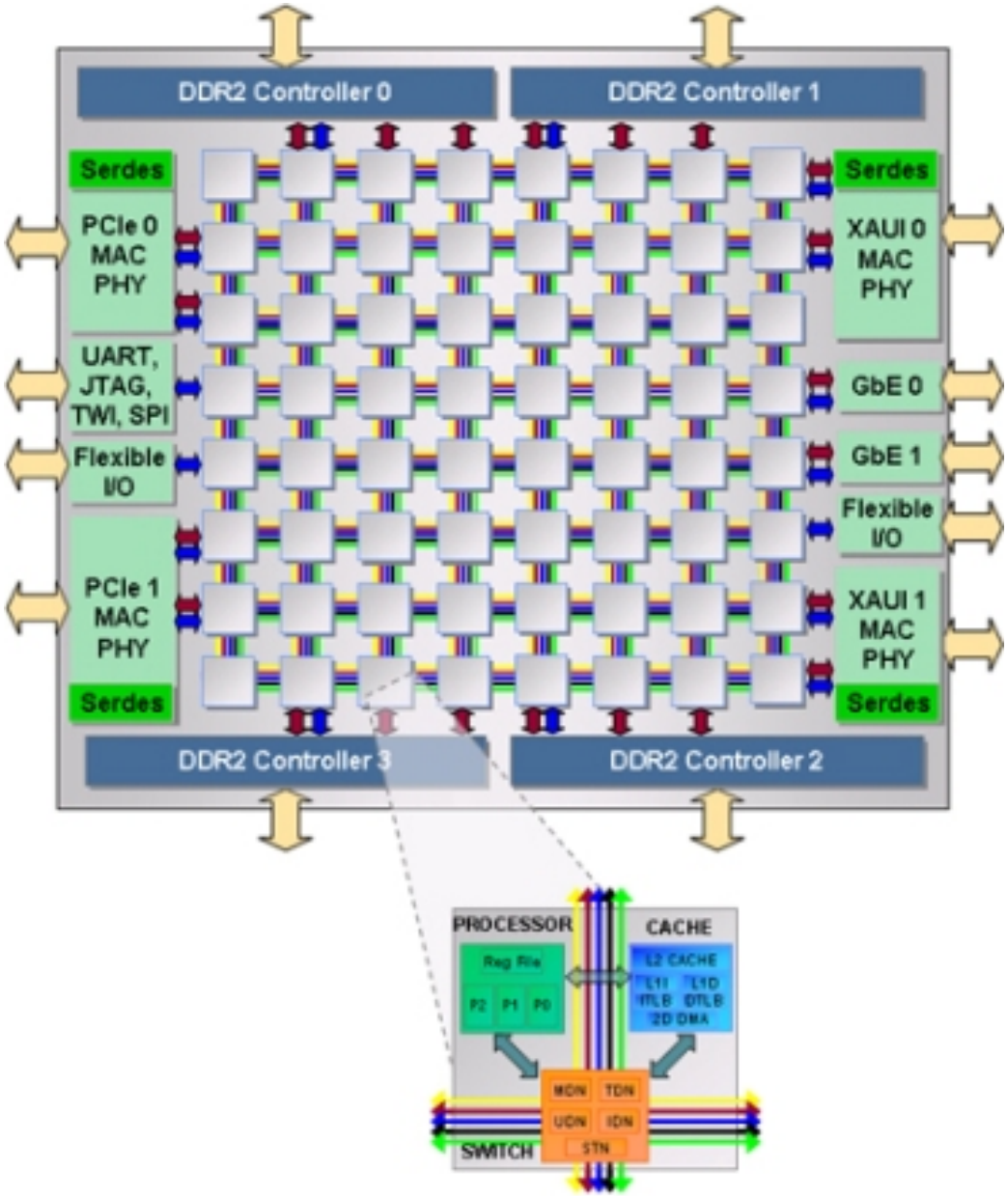
Problématique des accélérateurs (slide Jerome Dubois)

- Pour tous : adressage mémoire différent



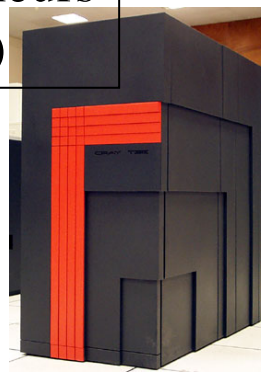
- Pour la plupart : langage différent
  - Nvidia CUDA
  - Extensions C pour le Cell
  - OpenCL pour accélérateurs (AMD, Nvidia, Intel, IBM, …)

Multi-core processors



Du HPC jusqu'aux grilles de calcul

Architecture superscalaire massivement parallèle (> 1000 CPUs, répartis en plusieurs nœuds à mémoire partagée)



Architecture monoprocesseur scalaire (1 CPU)

Architecture monoprocesseur vectorielle

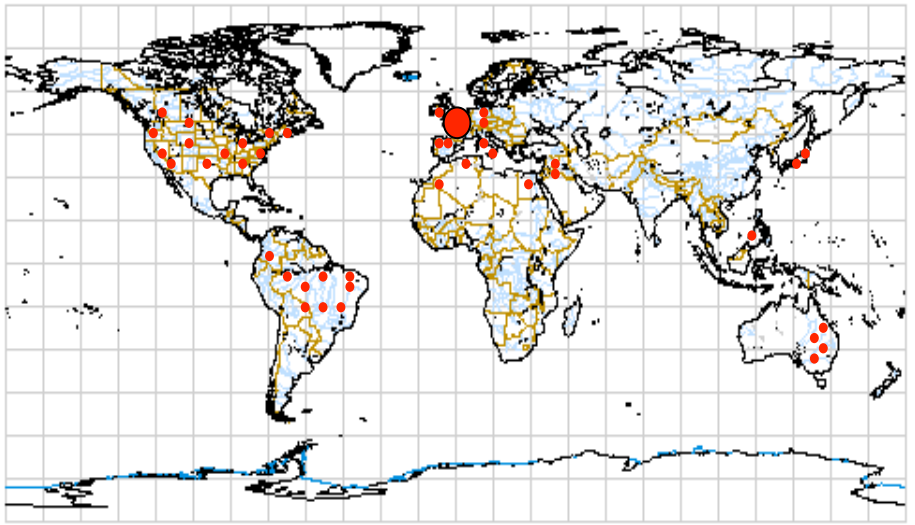
Architecture parallèle parallèle à mémoire partagée (quelques douzaines de CPUs vectoriels,…)

Du HPC jusqu'aux grilles de calcul

Clusters (grappes) de PCs
(NOWs network of workstations)

Clusters de machine parallèles à
l'échelle d'un centre

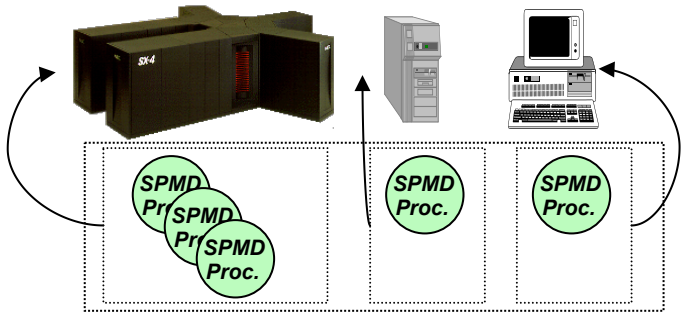
Vol de cycles à l'échelle
d'un réseau local



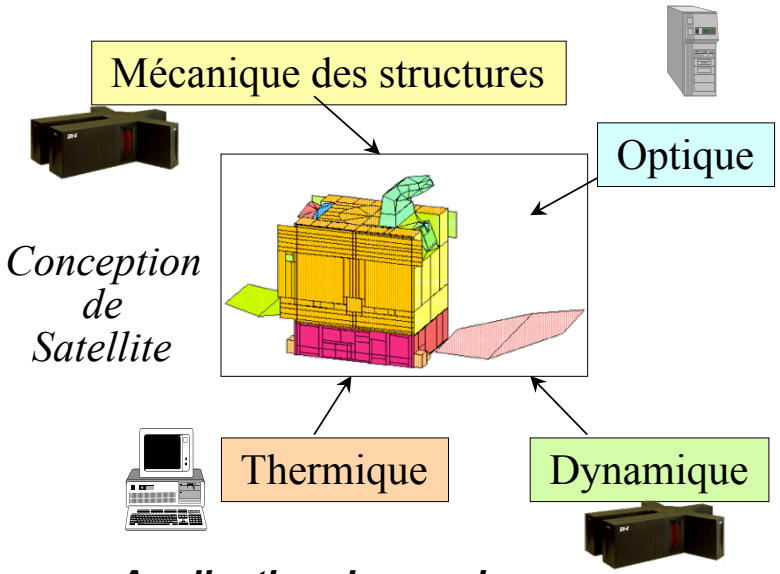
Grilles de calcul et de stockage

Programmer les grilles de calcul

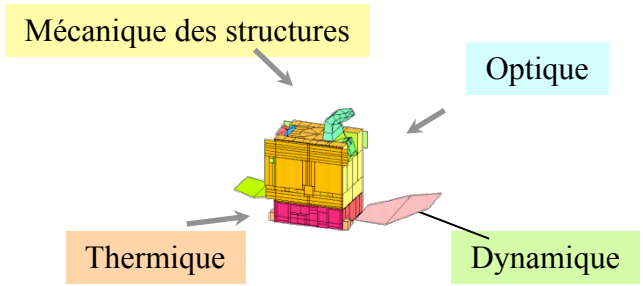
- Un champ applicatif vaste avec des besoins variés…
- Codes parallèles
  - Une grille de calcul est vue comme un calculateur parallèle virtuel (la genèse du Grid)
- Couplages de codes
  - Une application est un assemblage de plusieurs codes de calcul modélisant des physiques différentes



Application parallèle

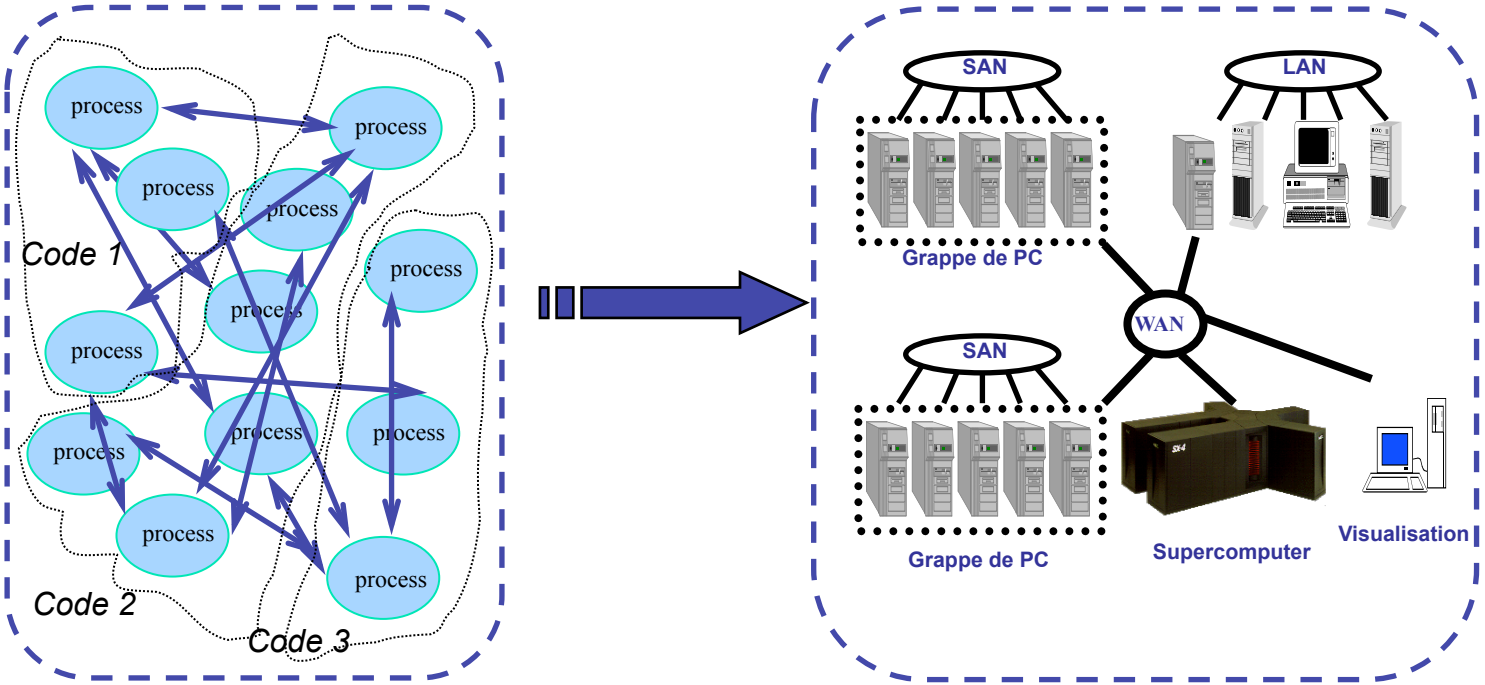


Application de couplage



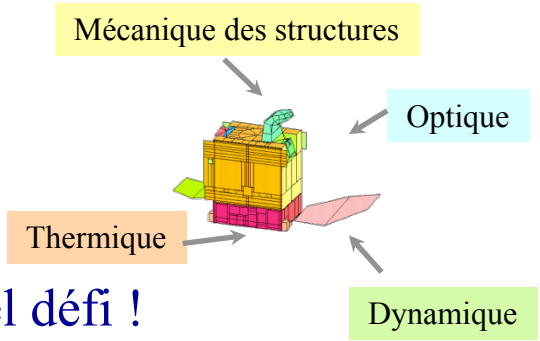
Couplage de codes

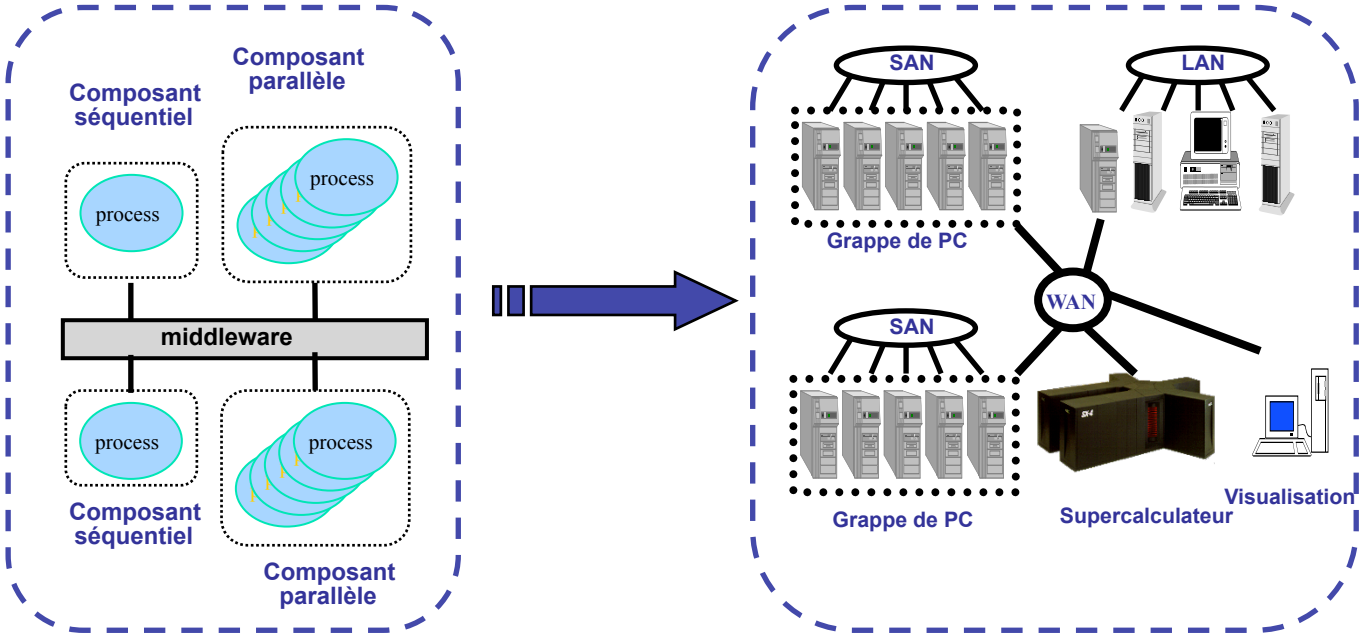
- Utilisation des exécutifs conçus pour la programmation parallèle
  - Une grille de calcul est un calculateur parallèle virtuel, la programmation par échange de messages s'impose, tout repose alors sur le Middleware…



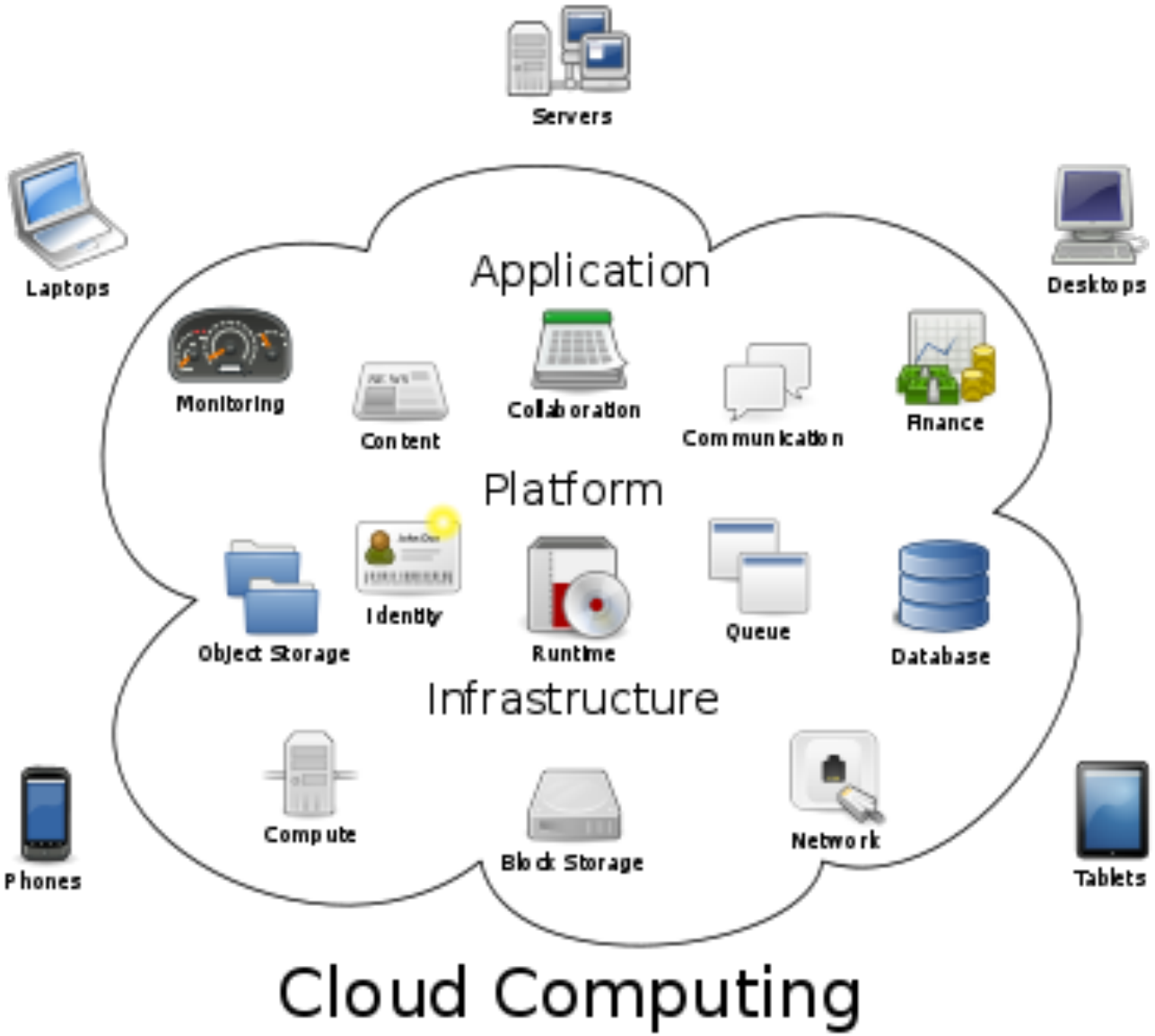
Une approche plus moderne

- Objets distribués/composants
  - Structuration de l'application
  - Encapsulation des codes
- Couplage de codes parallèles
  - Interconnexion des objets/composants ➡ un réel défi !

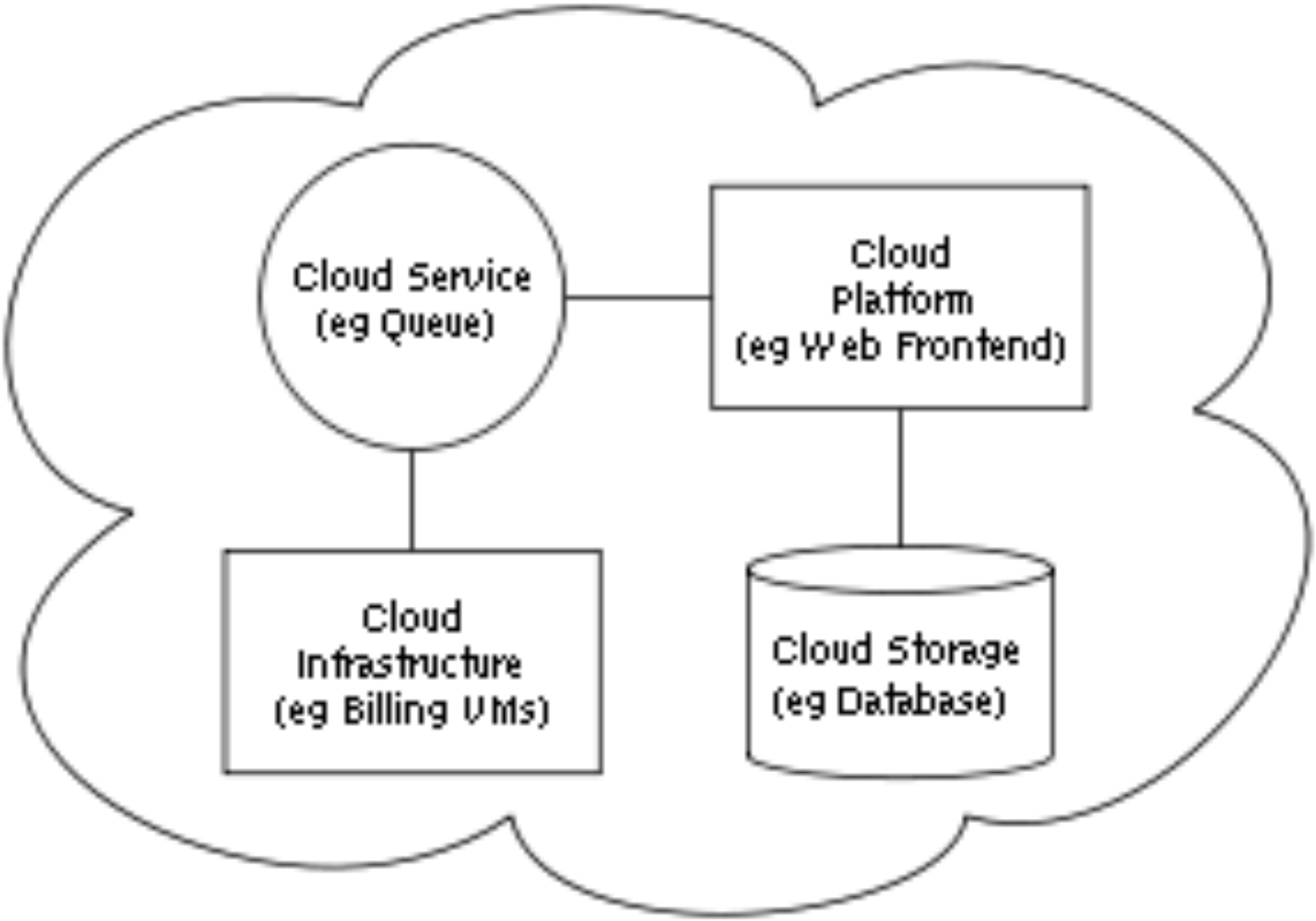


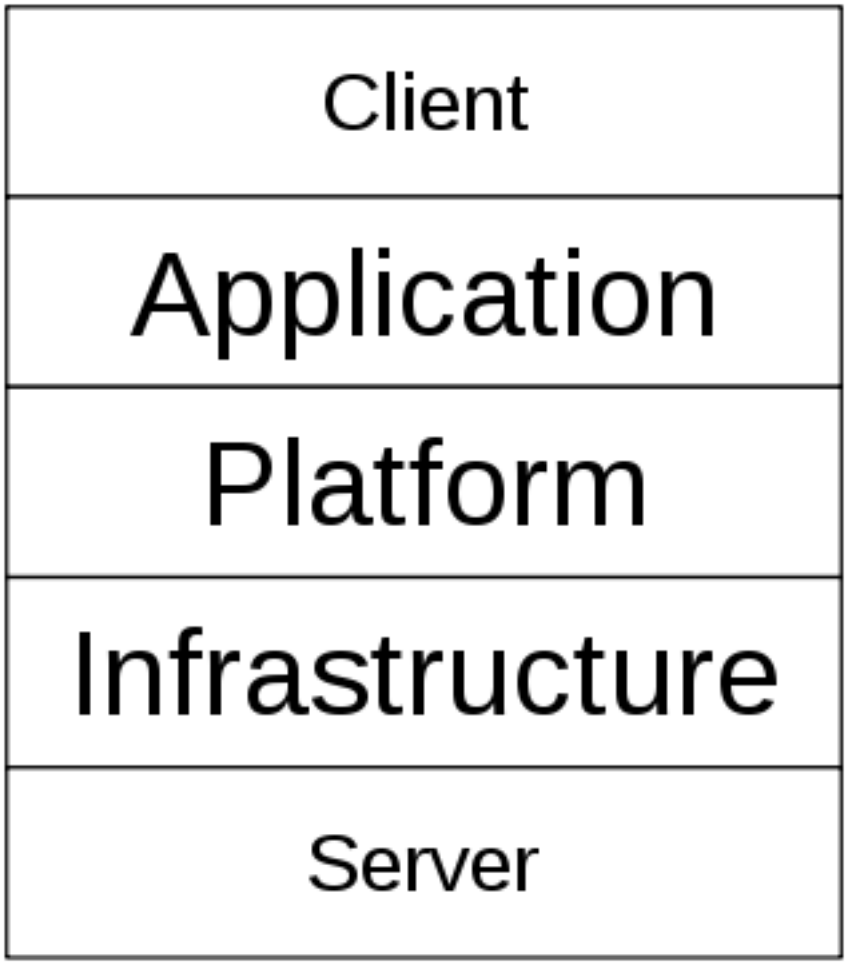


Cloud


Servers
Laptops
Desktops
Application
Monitoring
Content
Collaboration
Communication
Finance
Platform
Identity
Object Storage
Runtime
Queue
Database
Infrastructure
Compute
Block Storage
Network
Phones
Tablets
Cloud Computing

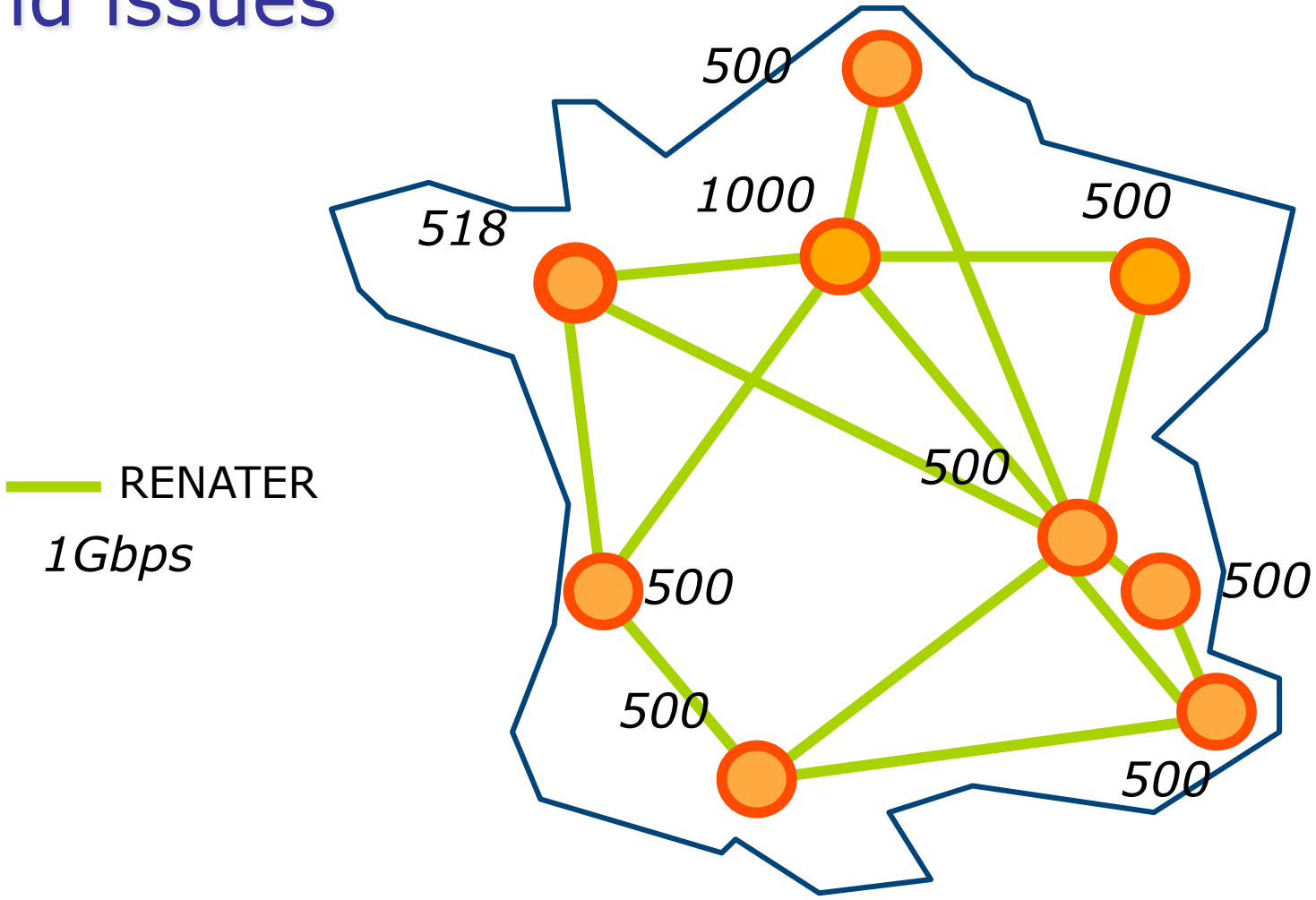
Architecture




Client
Application
Platform
Infrastructure
Server

Grid'5000 map (objective)

A large scale Instrument to study Grid issues

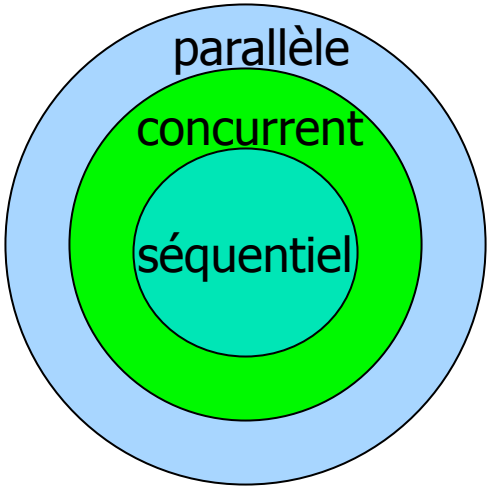


The Grid'5000 Project

1) Building a nation wide experimental platform for Grid & P2P researches (like a particle accelerator for the computer scientists)
   - 9 geographically distributed sites
   - every site hosts a cluster (from 256 CPUs to 1K CPUs)
   - All sites are connected by RENATER (French Res. and Edu. Net.)
   - RENATER hosts probes to trace network load conditions
   - Design and develop a system/middleware environment for safely test and repeat experiments

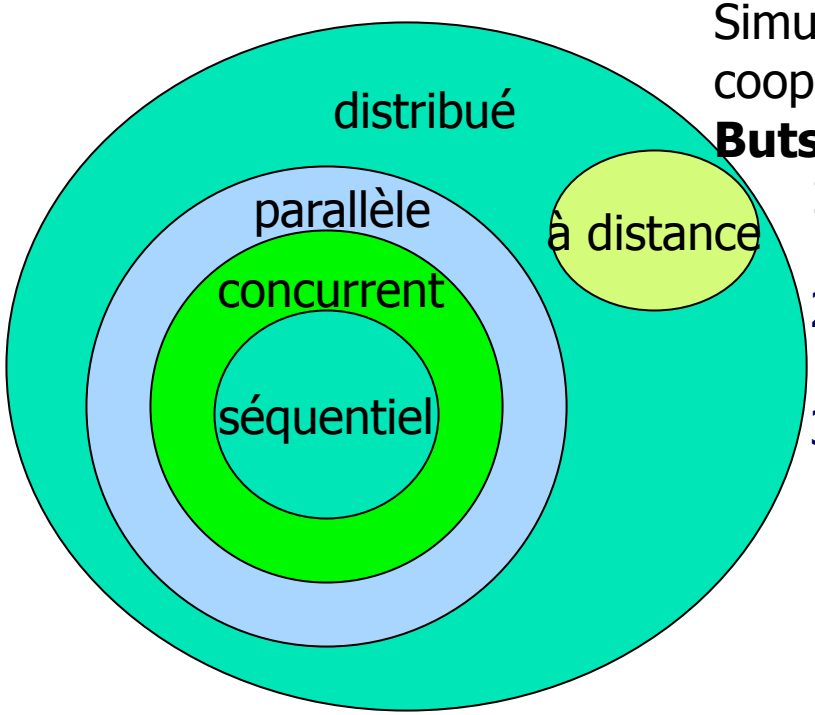
2) Use the platform for Grid experiments in real life conditions
   - Address critical issues of Grid system/middleware:
     - Programming, Scalability, Fault Tolerance, Scheduling
   - Address critical issues of Grid Networking
     - High performance transport protocols, Qos
   - Port and test applications
   - Investigate original mechanisms
     - P2P resources discovery, Desktop Grids

Parallélisme et distribution : panorama



- **Traitement séquentiel**
  Exécution séquentielle d'un flux d'instructions sur un seul processeur
- **Tâches concurrentes**
  Exécution simultanée de plusieurs tâches concurrentes
  But : meilleure utilisation CPU, meilleures performances globales
- **Traitements parallèles**
  Exécution simultanée de plusieurs traitements (sous-tâches coopérant à la réalisation d'un traitement)
  But : accroître la rapidité d'exécution pour
  1) obtenir plus rapidement un résultat,
  2) traiter des problèmes de plus grande taille,
  3) améliorer la qualité du résultat

Parallélisme et distribution : panorama
(suite)



◆**Traitement distribué**

Répartition géographique de traitements Simultanés sur des processeurs distincts et coopérants entre eux

Buts :

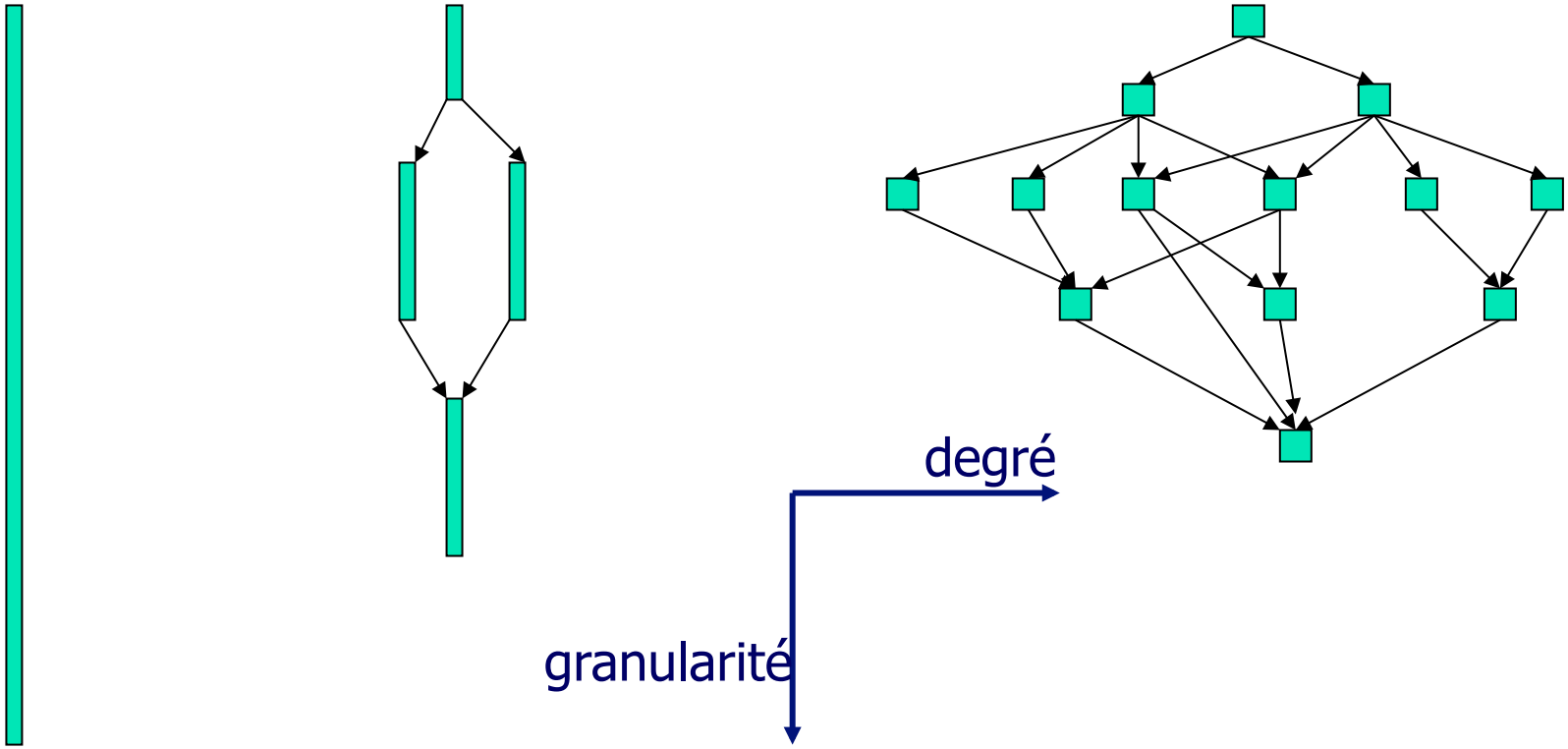
1. Mettre en œuvre le parallélisme pour augmenter l'efficacité de l'exécution,
2. Utiliser les ressources disponibles non utilisées,
3. Exploiter une distribution pré-existante des informations ou des traitements

ii. Degré et granularité du parallélisme

Séquentiel

Gros grain (*coarse grain)*

Fin grain (*fine grain)*



iii. Accélération et lois classiques

- **Accélération (speed-up) $S(n)=t_s / t_p(n)$ (ou $1- t_s / t_p(n)$)**
- **Loi d'Amdhal (n processeurs)**
  - $S(n) = n / (1+(n-1)\, s)$ où s = fraction du temps de l'exécution séquentielle qui ne peut être parallélisée (exécutée en séquentiel)

⇒ Accélération maximale $S(n) \rightarrow 1/s$ quand $n \rightarrow \infty$

Exemple : s=10%, N=10 processeurs => accélération max 5,3 %
s=20%, N=10 processeurs => accélération max 3,6 %
s=5%, N=20 processeurs =>accélération max 10.26%

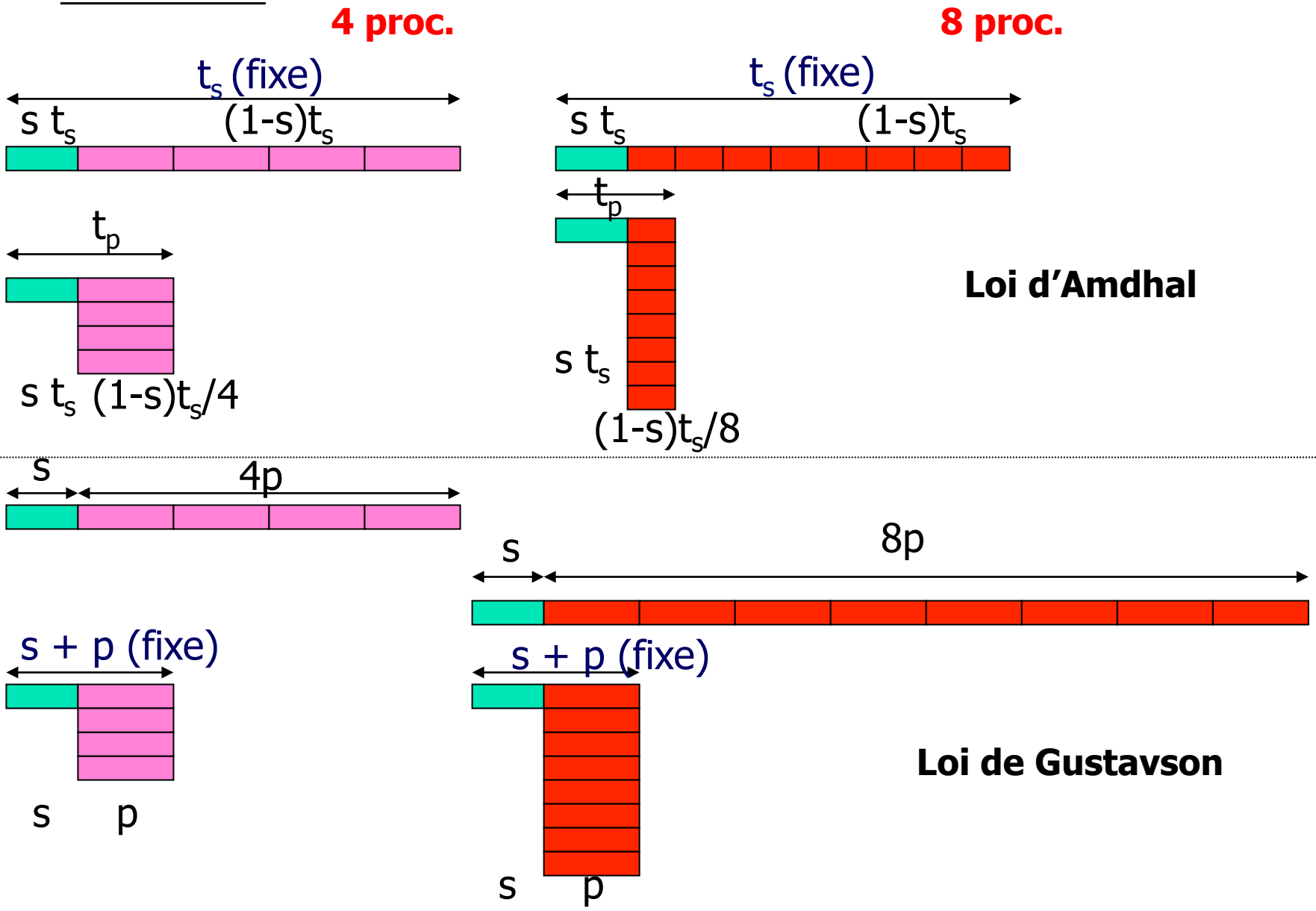
- **Loi de Gustavson**
  - $S(n) = n + (1-n)\, s$

s=5%, N=20 processeurs =>accélération max 19.05%

- **Lois basées sur des hypothèses différentes**

- Loi d'Amdhal : taille du pb constant t_s fixe
- Loi de Gustavson : Tps d'exécution constant (pb de plus grande taille) s+p fixe

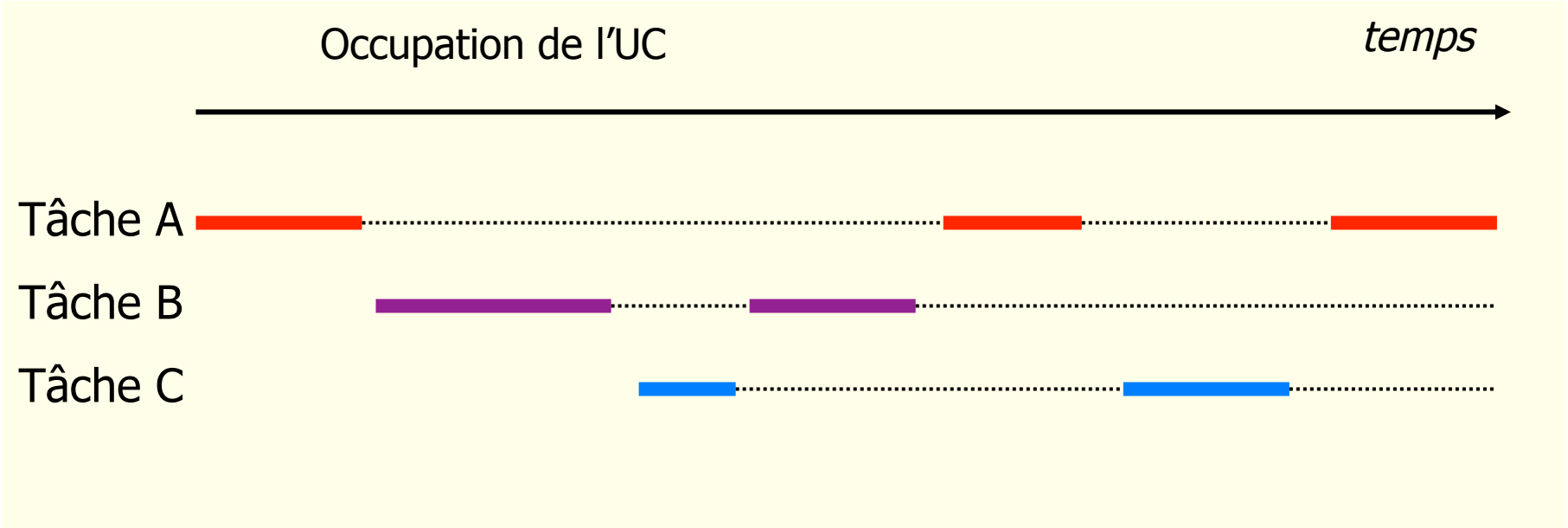
Illustration



Les formes de parallélisme

- **Concurrence**
  - Partage de l'UC
- **Le parallélisme de contrôle**
  - Graphe de tâches
- **Le parallélisme de données (SPMD)**
  - Fragmentation des données
- **Le parallélisme de flux**
  - Fonctionnement pipe-line

Traitements concurrents

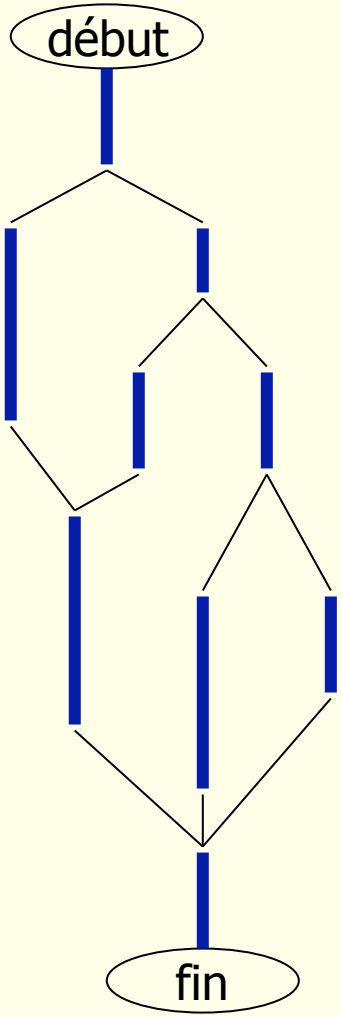


Une seule tâche est allouée à la fois à l'UC par le scheduler, sauf dans le cas de processeur disposant de l'hyperthreading

◆ **Le parallélisme de contrôle** (basé sur le flux d'instructions)

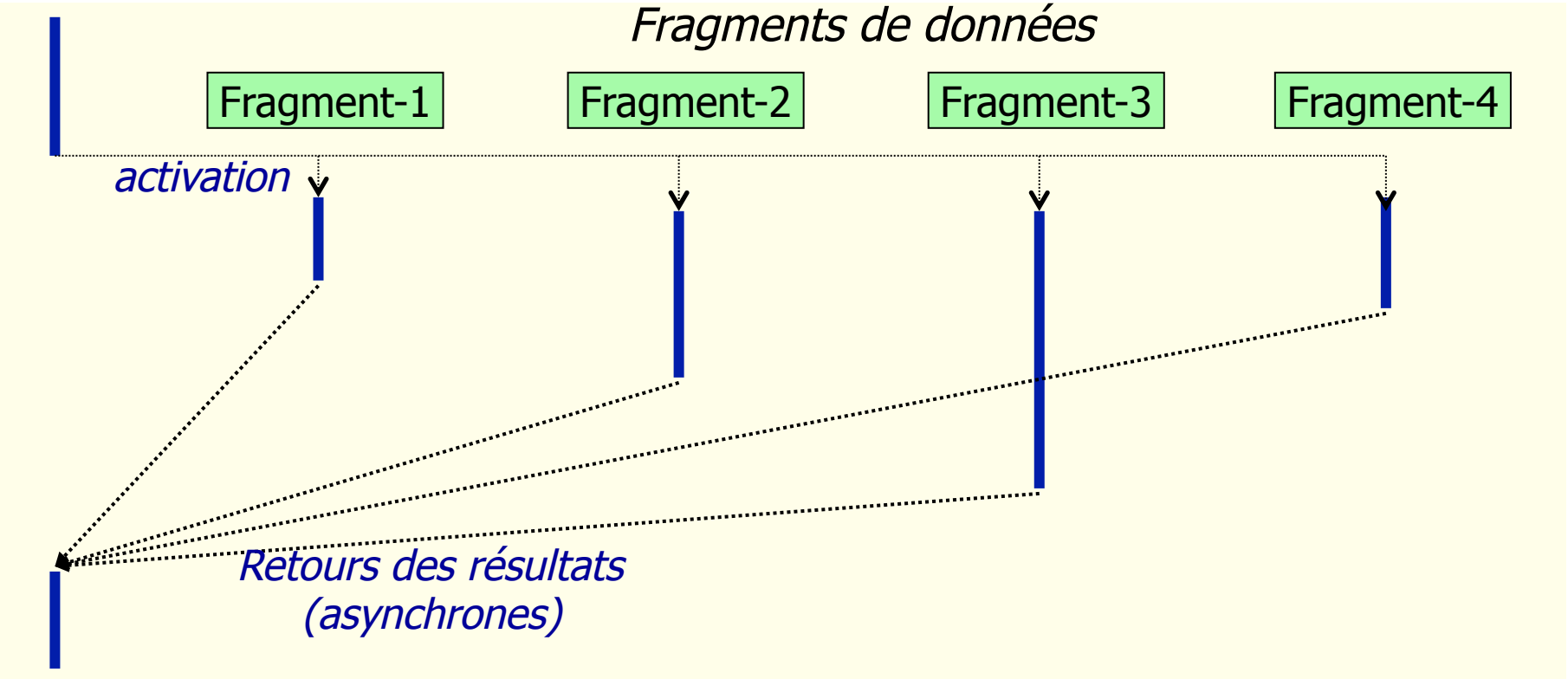
Découpage de la tâche en un **graphe de sous-tâches** à distribuer (statique)

Une bonne distribution suppose une connaissance suffisante des temps d'exécution respectifs des sous-tâches



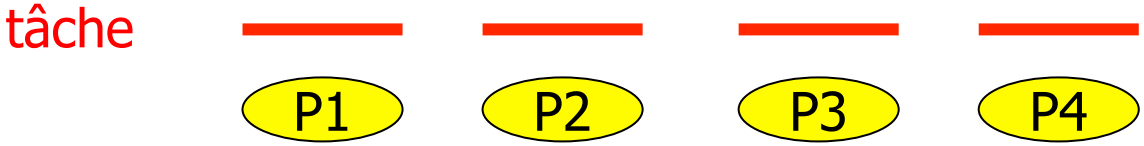
Le parallélisme de données

- mode SPMD : Single Program Multiple Data
- s'appuie sur une fragmentation des données : une tâche s'exécute sur chacun des fragments de données

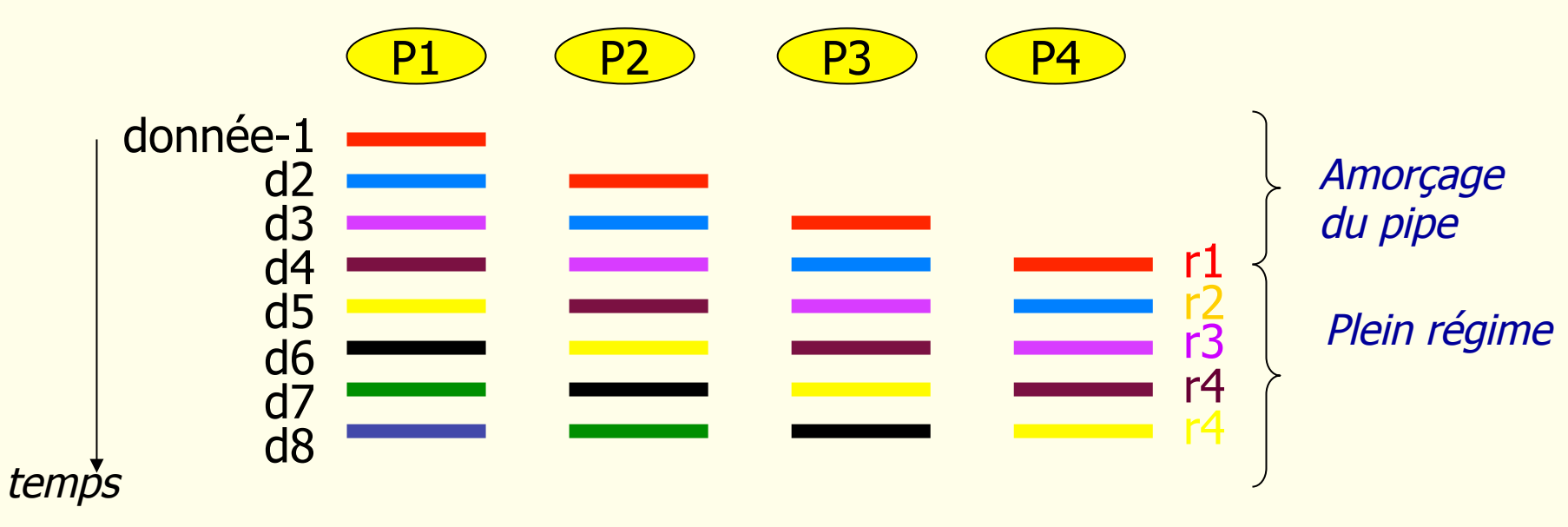


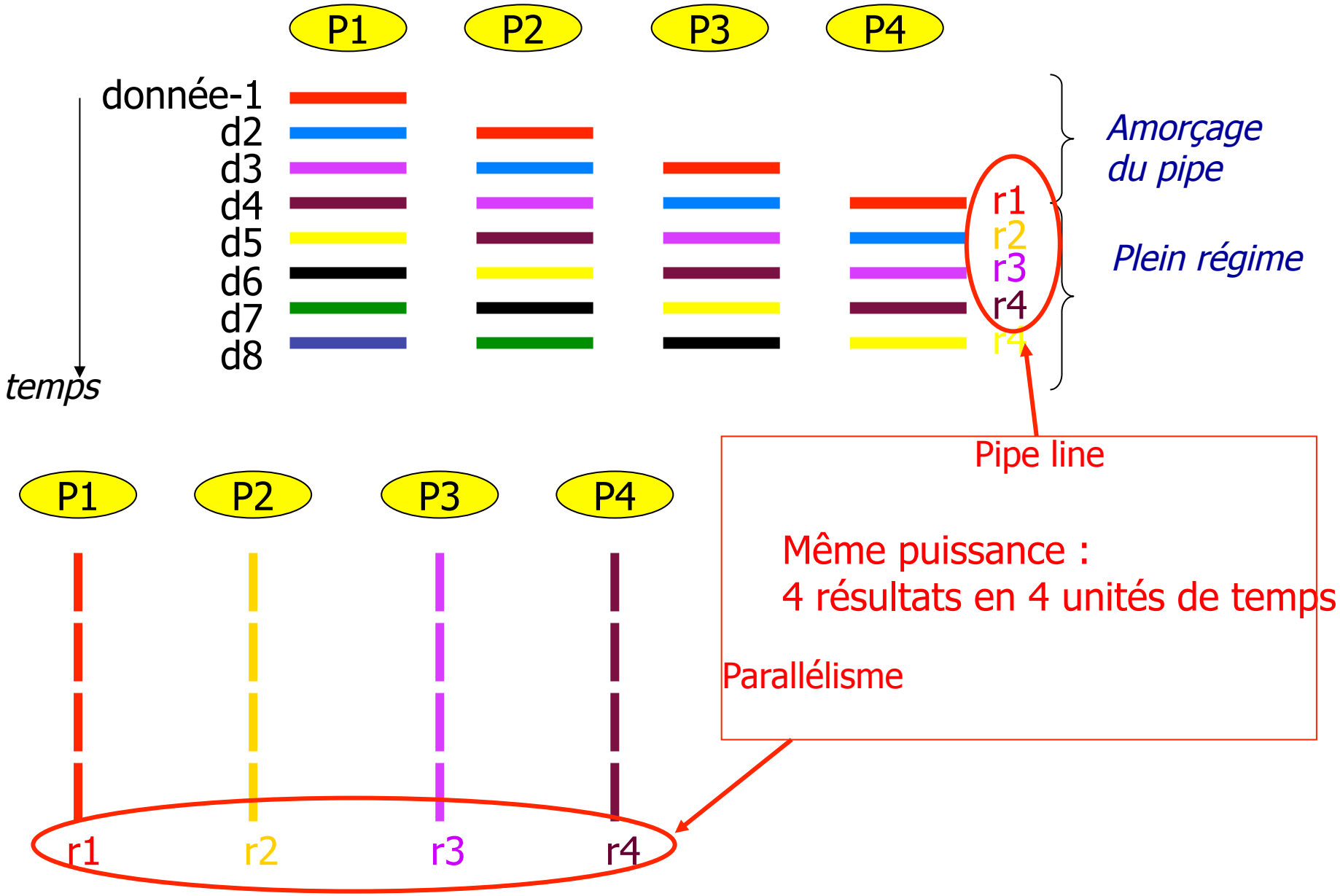
Le parallélisme de flux (mode pipe line)

- une tâche est divisée en une succession de sous-tâches qui sont exécutées sur une suite de processeurs



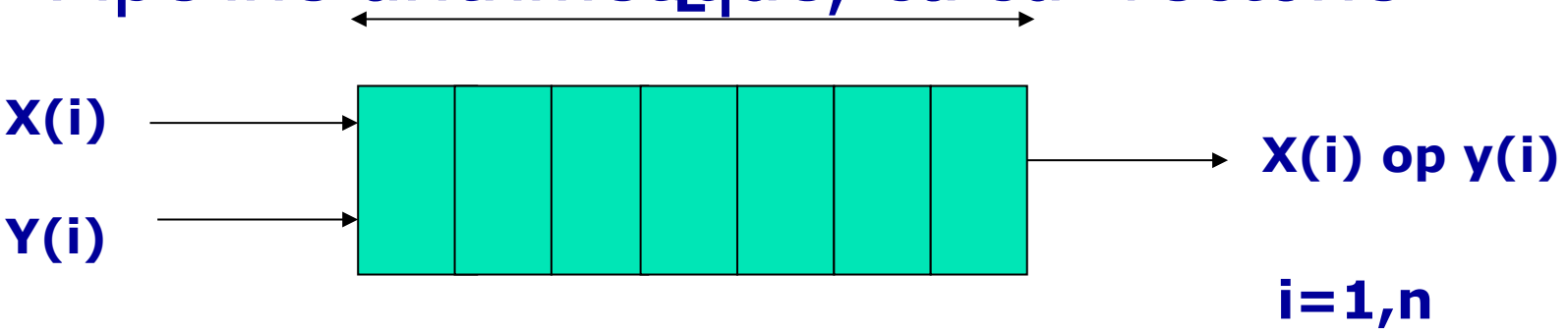
- Les données à traiter passent les unes à la suite des autres dans chacun des processeurs




P1
P2
P3
P4
donnée-1
d2
d3
d4
d5
d6
d7
d8
temps
r1
r2
r3
r4
r4
Amorçage du pipe
Plein régime
Pipe line
P1
P2
P3
P4
Même puissance :
4 résultats en 4 unités de temps
Parallélisme
r1
r2
r3
r4

Calcul séquentiel versus vectoriel

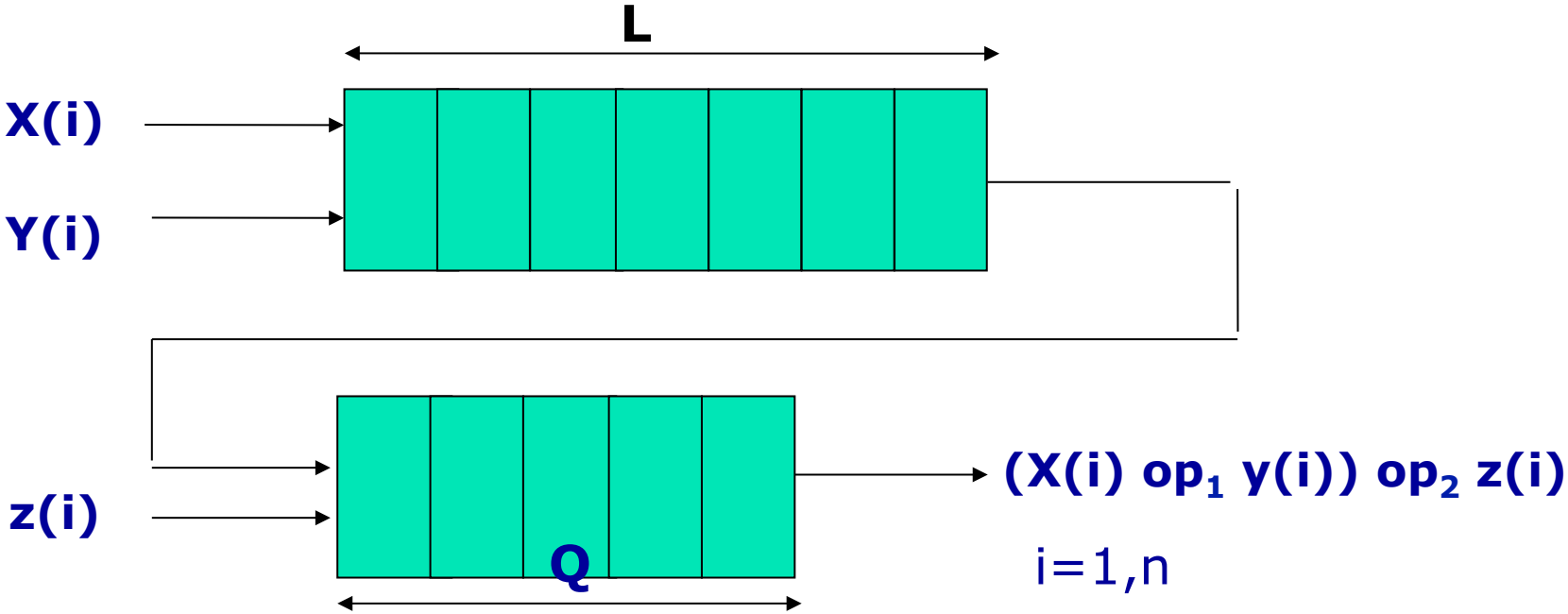
- Calcul VLIW, optimisation des caches
- Pipeline arithmétique, calcul vectoriel



$t_{seq} = t_{horloge}\ n\ L$

$t_{vect} = (L-1)t_{horloge} + n\ t_{horloge}$

Calcul vectoriel, chaînage de pipes



$$t_{seq} = t_{horloge}\ n\ (\ L + Q\ )$$

$$t_{vect} = (L+Q-1)t_{horloge} + n\ t_{horloge}$$

Chaînage de pipes et parallélisme