Ordonnancement du point de vue architecture et système

Jean-François Méhaut
Université Joseph Fourier, Grenoble 1

Projet Mescal
Laboratoire ID-IMAG UMR 5132 CNRS INPG INRIA UJF
Antenne ENSIMAG Montbonnot
3833O Montbonnot Saint Martin

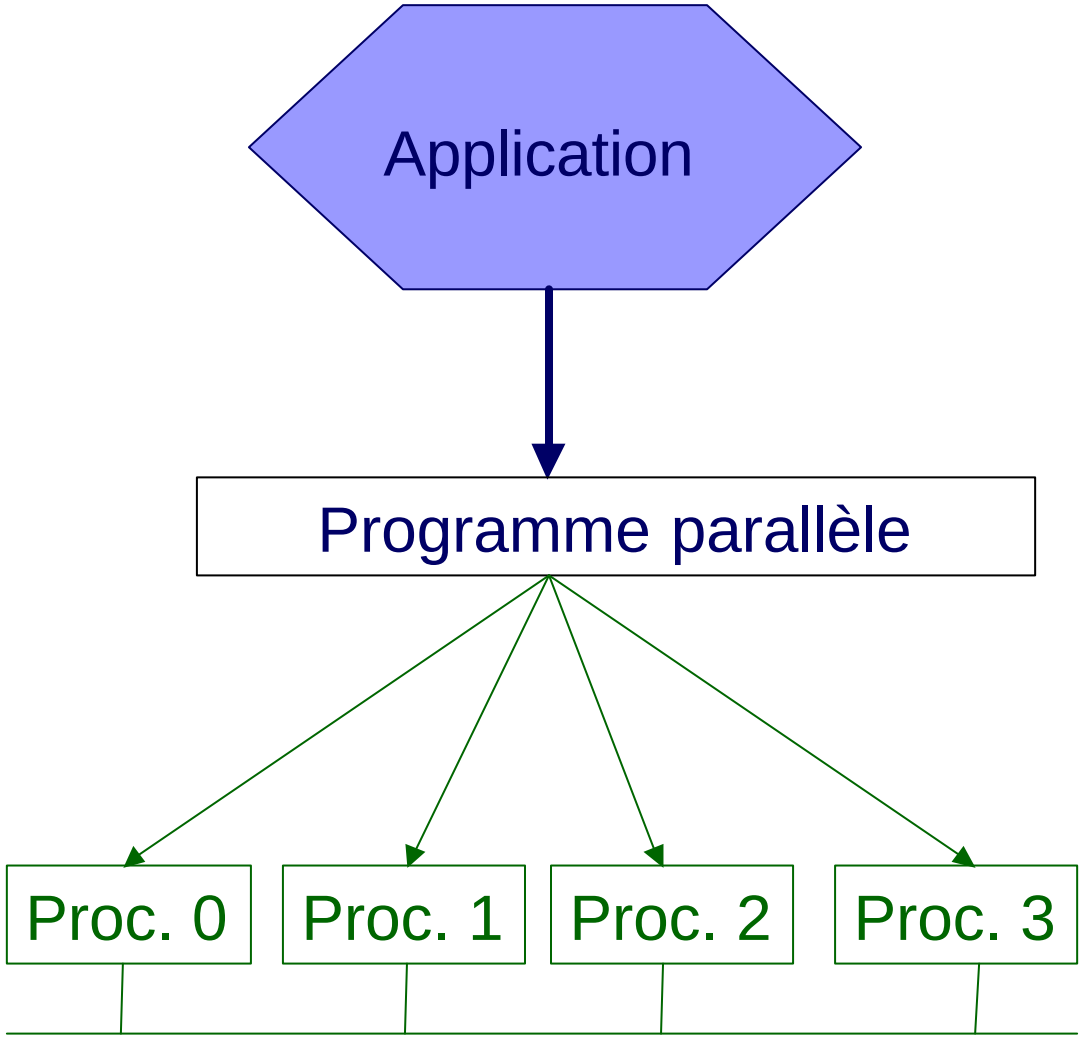
Jean-Francois.Mehaut@imag.fr

Plan

- Introduction
  - Supports d'exécution pour le calcul parallèle, environnements, middlewares, intergiciels, ...
- Multithreading, Hyperthreading, SMT
  - Rappels, exploitation des SMP, ordonnancement de threads...
- Communications dans les grappes
  - Technologies, interfaces, ordonnancement des communications

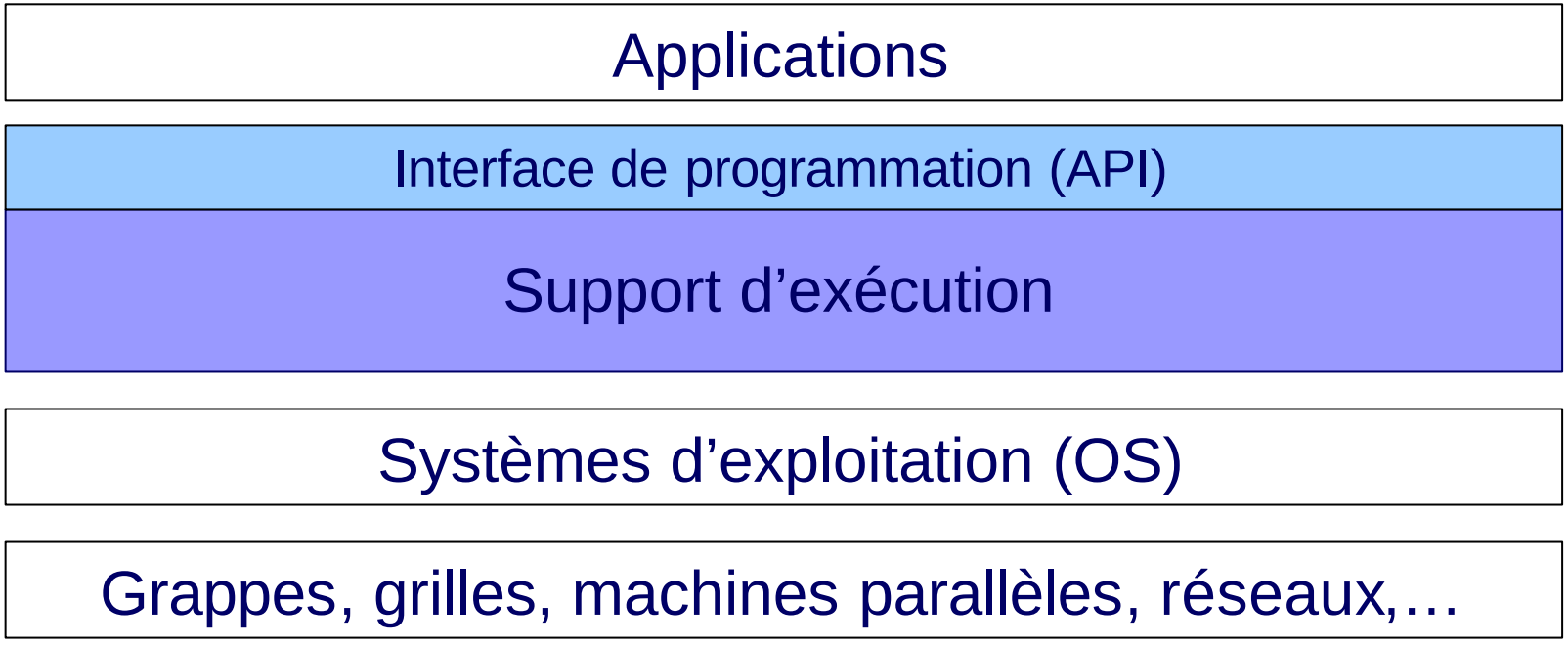
Programmation parallèle et répartie

- Conception
  - Modélisation
  - Algorithmique
  - Langage
  - Compilation
- Exécution
  - Gestion d'activités
  - Ordonnancement
  - Communications
  - Mise au point
  - Régulation de charge
  - Gestion de données
  - Sécurité
  - ...

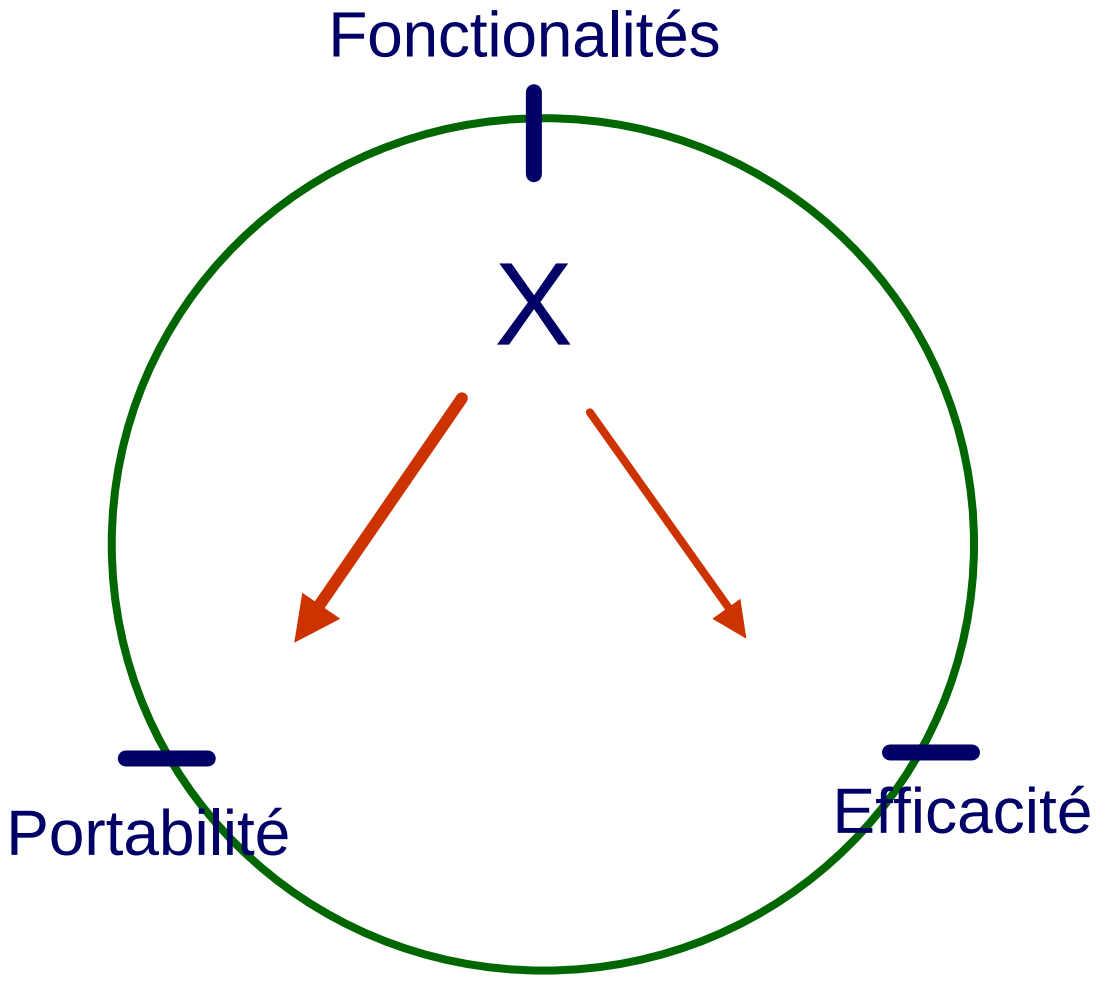


Supports et environnements d'exécution

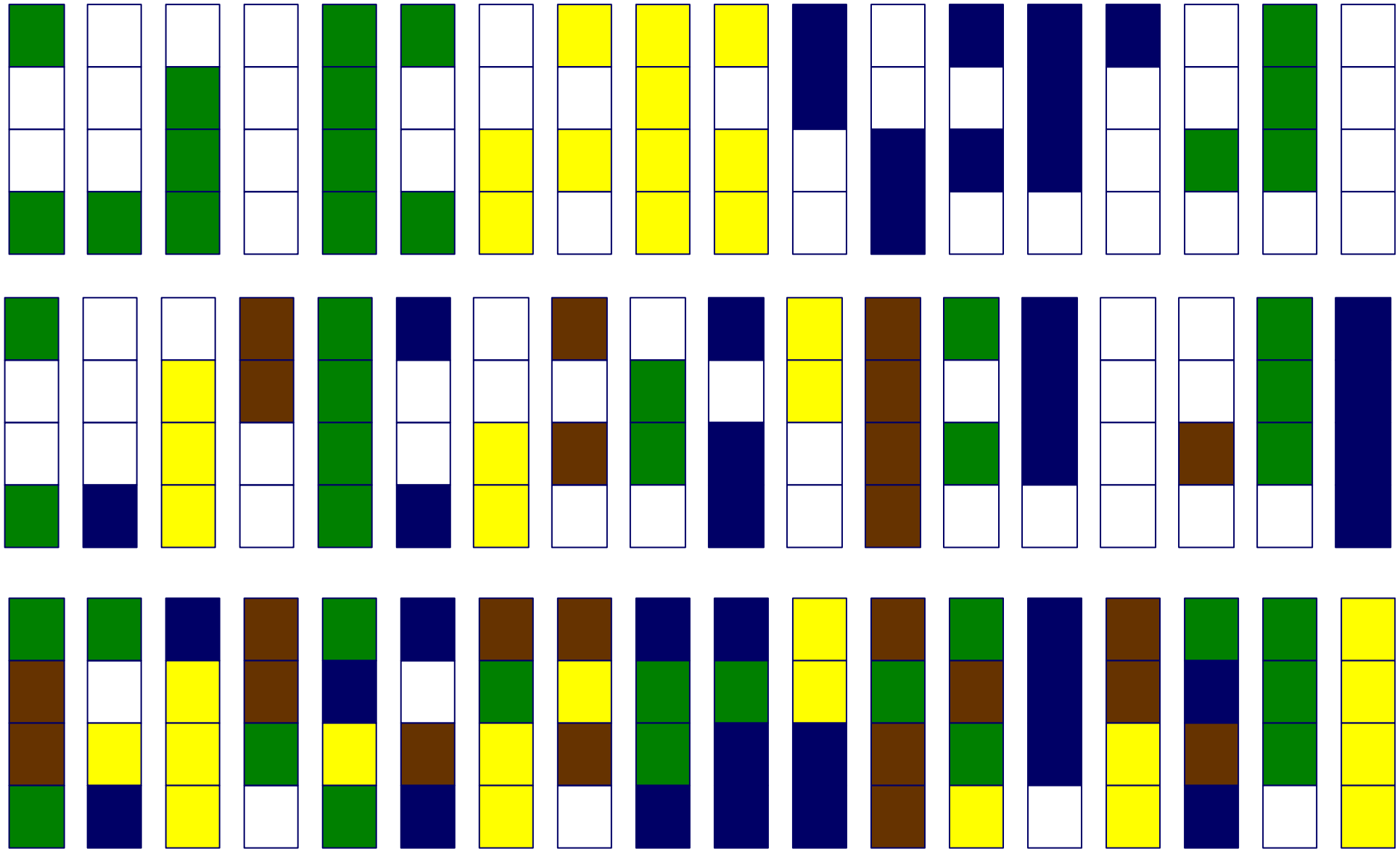
- Pour les utilisateurs et leurs applications
  - Abstractions de haut niveau
  - Portabilité
  - Efficacité !



Compromis à trouver



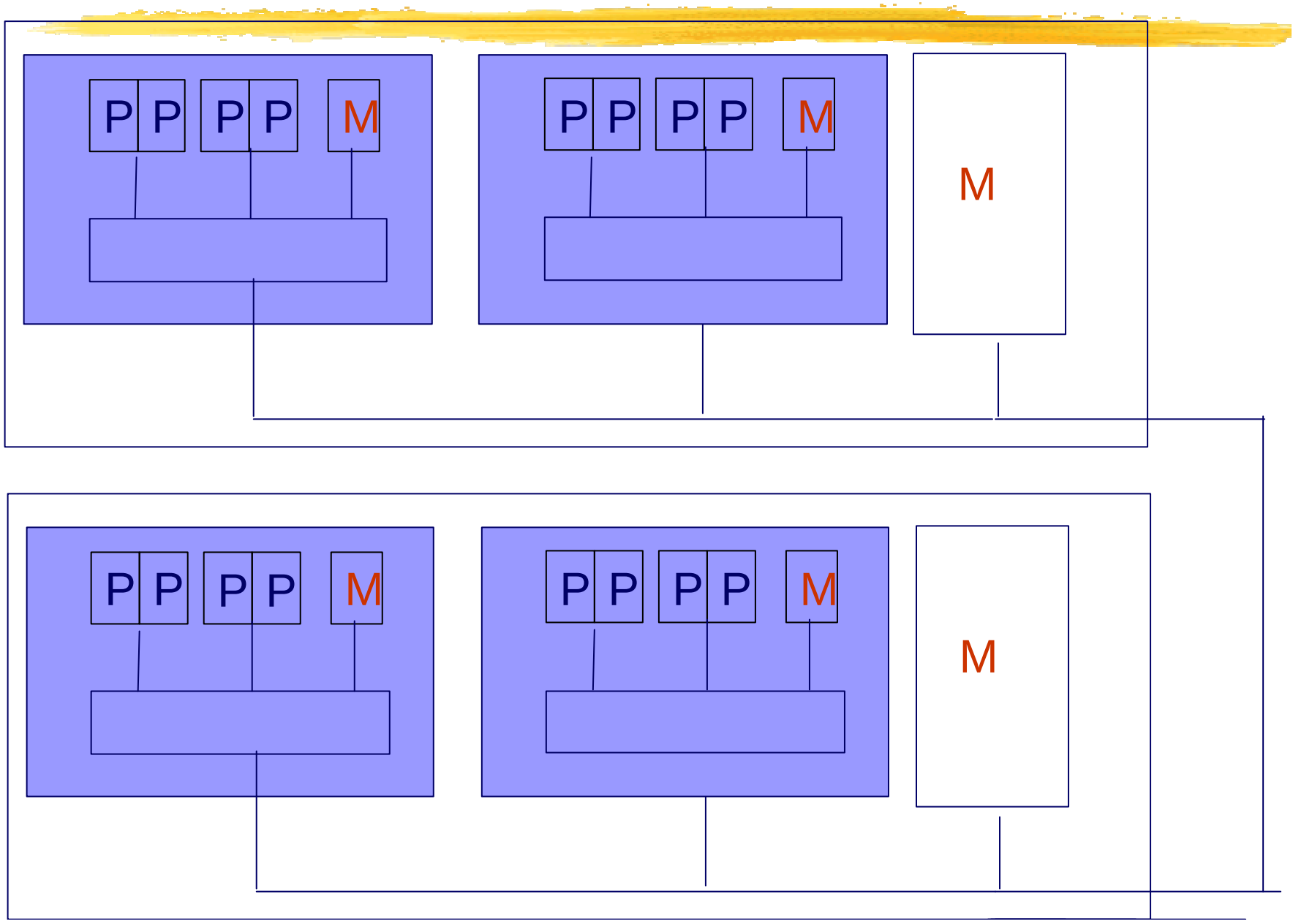
Hyperthreading, SMT, NUMA



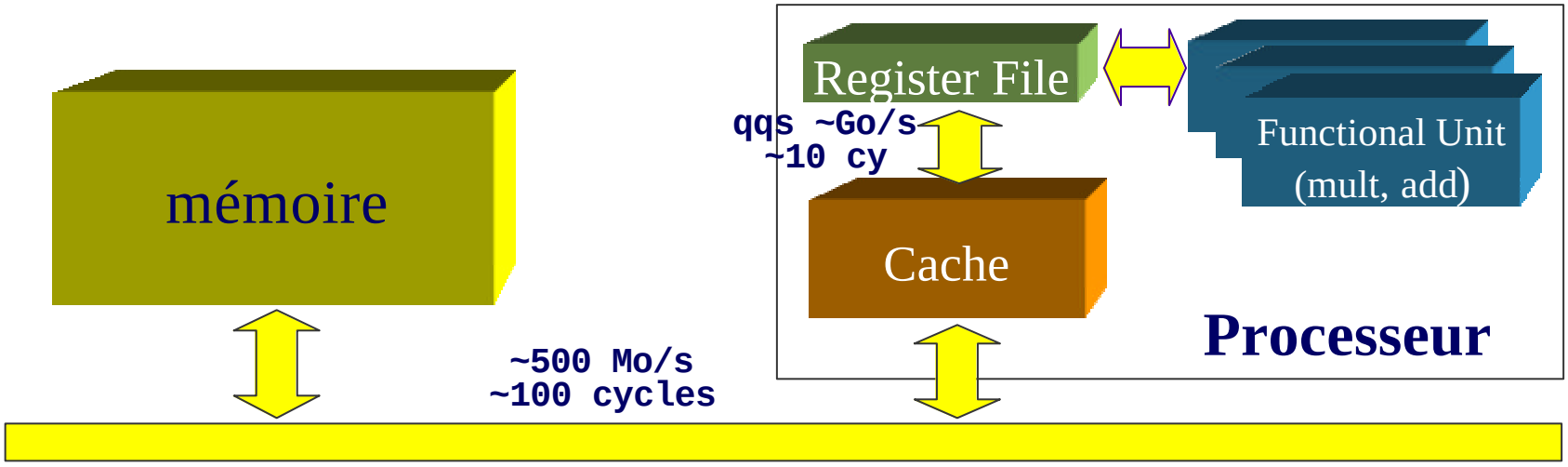
Introduction

- Tendance actuelle
  - NUMA
  - Multi-Core
  - HyperThreading
  - SMT Simultaneous MultiThreading
  - ...
- Solutions commerciales
  - Intel
  - IBM
  - SUN
  - HP

Machines de Demain

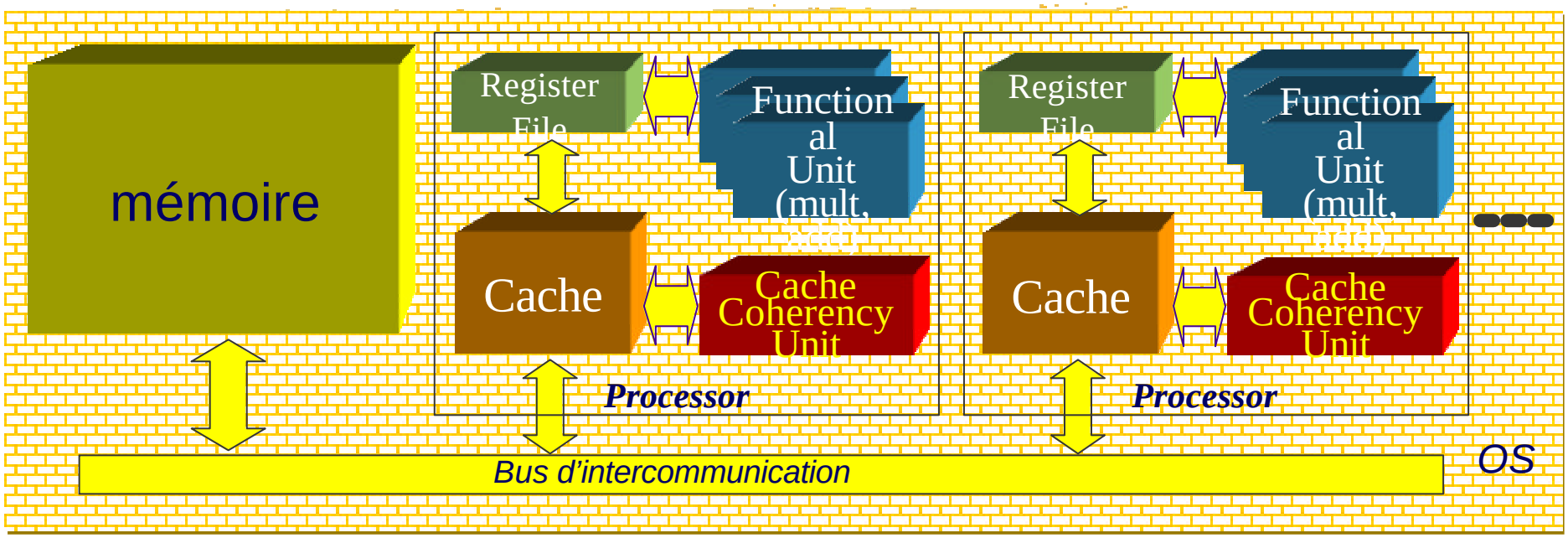


Architecture Scalaire



- Reduced Instruction Set (RISC) Architecture:
  - Les instructions load/store font référence à la mémoire
  - Les unités fonctionnelles travaillent sur des données stockées dans les registres
  - Hiérarchie mémoire dans une architecture scalaire :
    - Les éléments utilisés récemment sont copiés dans le cache,
    - Les accès au cache sont plus rapides que les accès à la mémoire.

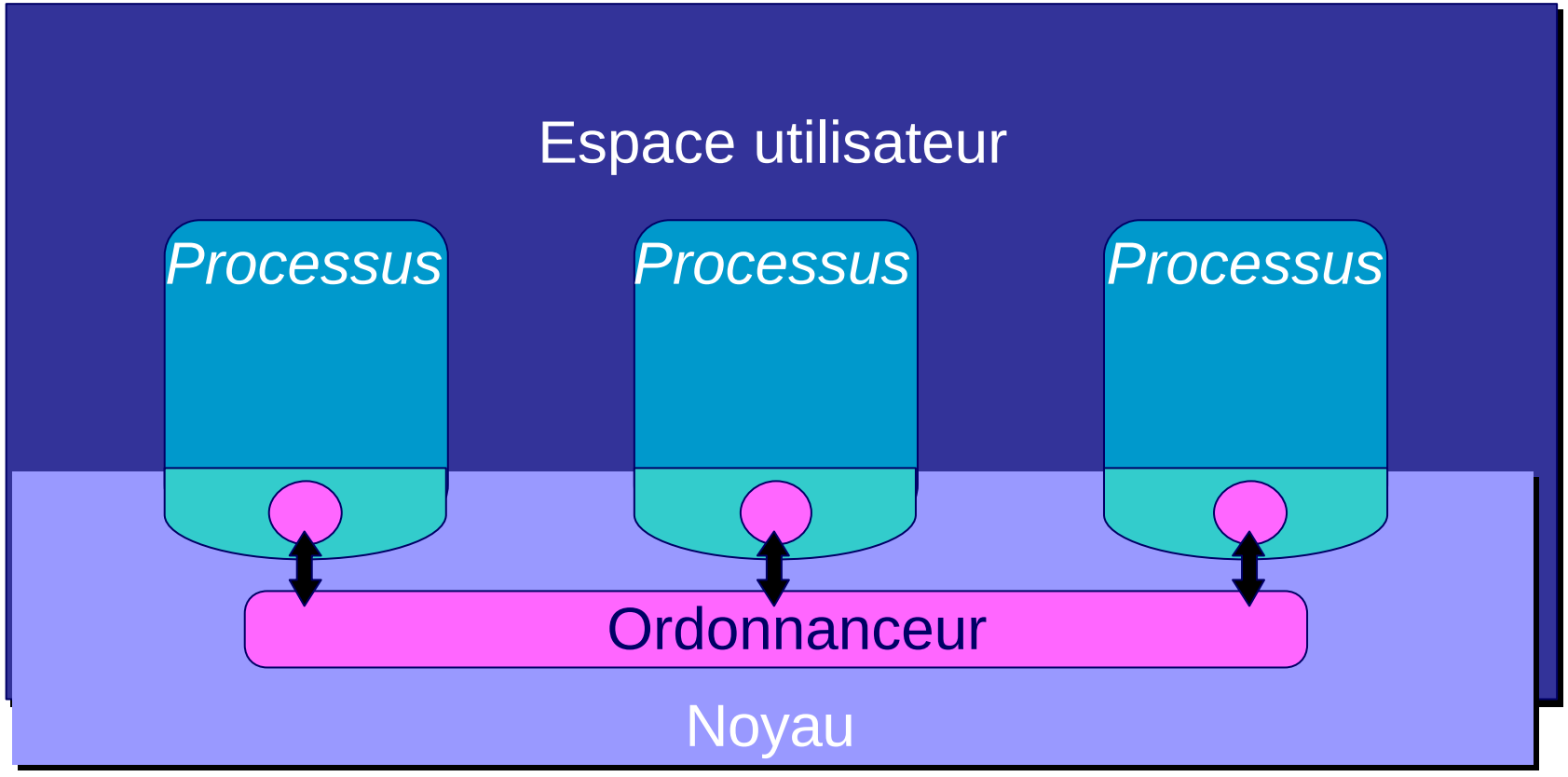
Architecture SMP UMA



- **Mémoire centrale et E/S**
  - **Partagées par tous les processeurs**
- **Modèle de programmation**
  - **Extension du modèle de programmation monoprocesseur**
- **Bus d'interconnexion entre la mémoire et les processeurs**
- **Protocole de cohérence cache**
  - **Plusieurs processeurs modifient des éléments de la même ligne de cache**
- **Faible nombre de processeurs**

“Vieux” Système d’Exploitation SMP

- Ordonnancement de processus



Programmation Mono-Threadée

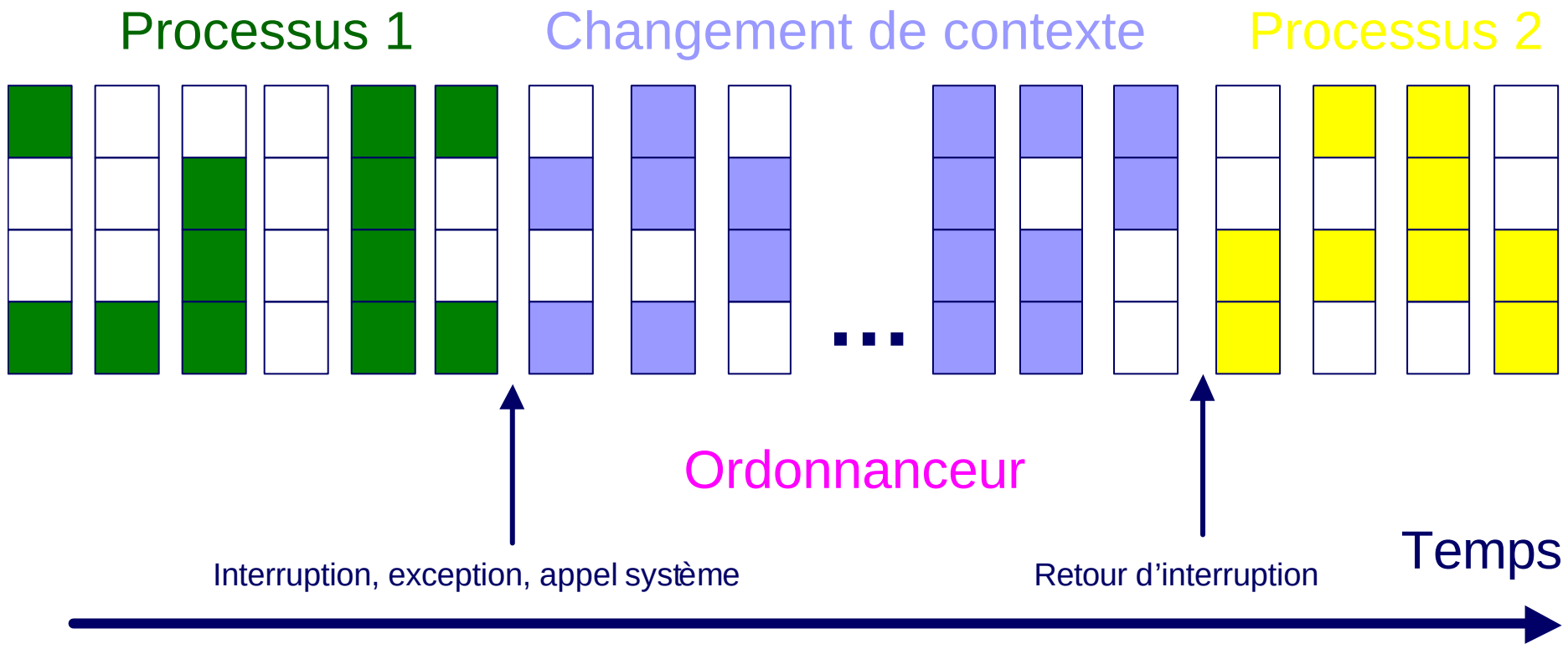
- Exécution séquentielle d'un programme
  - Instruction par instruction
  - Instructions: Calcul, Mémoire, Branchement, appel de procédure,...
- Processus, processus Lourd
  - Structuration des systèmes d'exploitation
  - Multi-programmation, temps-partagé
- Caractéristiques
  - Entité active directement supportée par l'OS
    - Flot d'exécution
    - Pile des contextes de procédure
    - Espace d'adressage privé
    - Ressources systèmes
- Coût de gestion élevé
  - Allocation des ressources (mémoire,...)
  - Appels systèmes (Fork, exec, ...)

Instruction Level Parallelism

- Programme séquentiel
  - N'y aurait-il pas des instructions indépendantes qui pourraient être exécutées en parallèle?
- Comment générer de l'ILP?
  - Pipe-line du processeur
    - Recouvrement d'exécution d'instructions
    - Limité par la divisibilité de l'instruction
  - Superscalaire
    - Plusieurs unités fonctionnelles
    - Limité par le parallélisme intrinsèque du programme seq.
- Comment accroître l'ILP?
  - Prédictions sur les branchements conditionnels
  - Réordonnancer les instructions (Out of Order Execution)
- Recherche
  - Domaines de l'Architecture-compilation

Processeur SuperScalaire

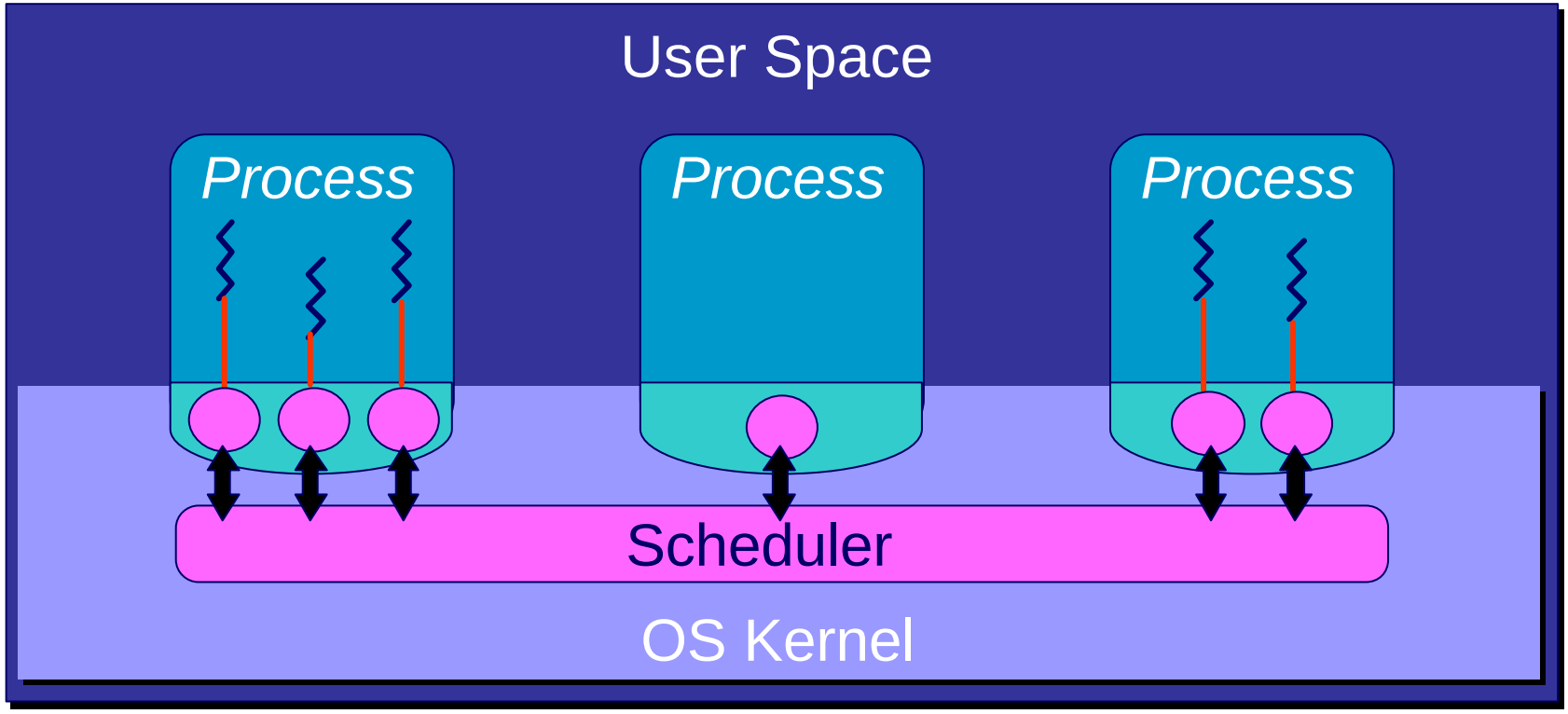
4 Unités fonctionnelles



- Réduire le temps des changements de contexte (cases grisées-bleues)
- Accroître l'utilisation des unités fonctionnelles (cases blanches)

Système d'exploitation SMP plus récent

- Ordonnancement de threads :



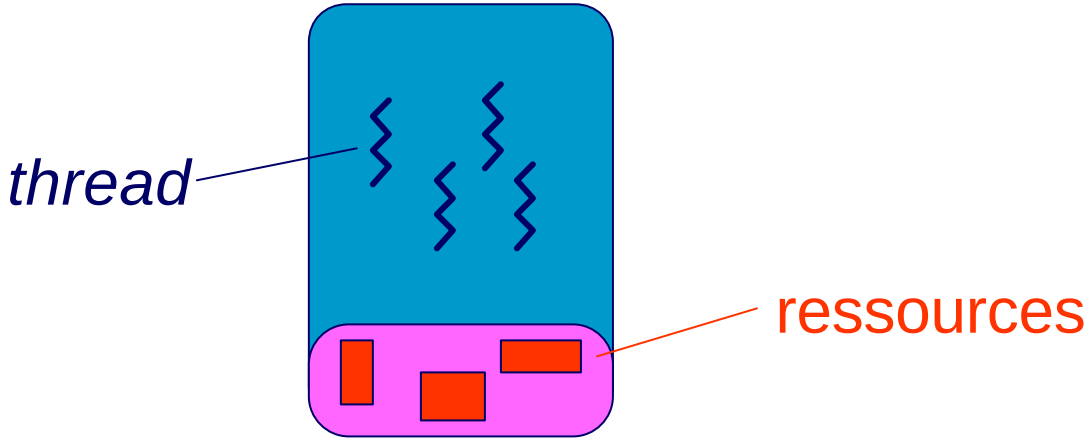
Processus Légers/Threads

- Objectifs
  - Mener plusieurs activités indépendantes au sein d'un processus
  - Exploitation des architectures SMP
  - Améliorer l'utilisation du processeurs (context-switch)
- Exemples
  - Simulations
  - Serveurs de fichiers
  - Systèmes d'exploitation (!)
- Solution sans l'aide du multithreading
  - Automate à états finis implanté « à la main »

    (sauvegardes d'états)

Les processus légers

- Principe
  - Détacher flot d'exécution et ressources



- Introduits dans divers langages & systèmes
  - Programmation concurrente
  - Recouvrement des E/S
  - Exploitation des architectures SMP

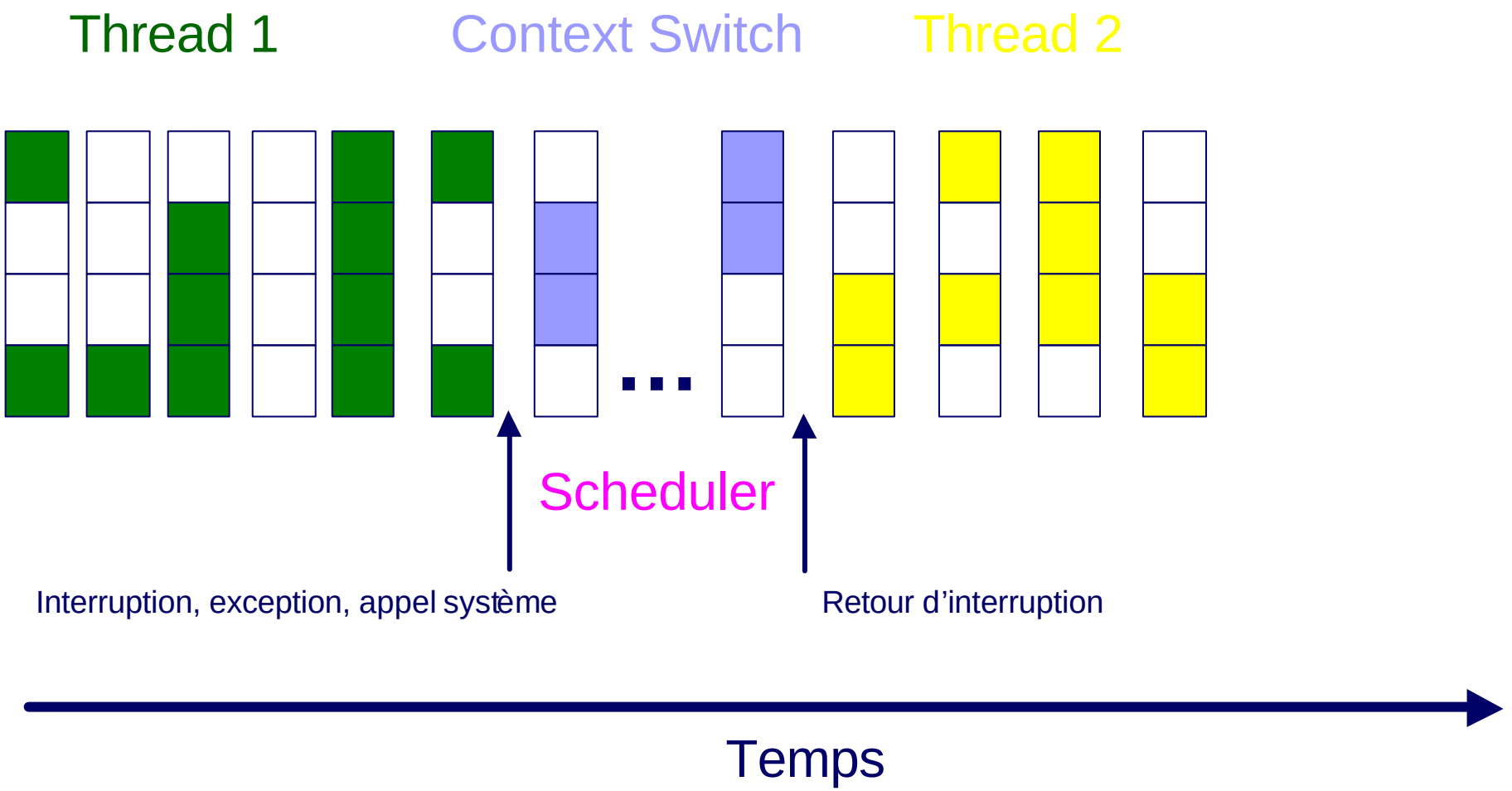
Performance des *threads*

- Opérations critiques
  - Création/destruction (gestion mémoire)
  - Changement de contexte (temps-partagé)
  - Synchronisation (mode utilisateur)
- Programme d'évaluation
  - Création d'une activité (processus, *thread*) + synchronisation (terminaison de l'activité)

OS/Processeur	**Processus**	***Thread* noyau**	***Thread* utilisateur**
Linux 2.2/PII 450	0.540	0.130	-
Solaris 2.7/PII 350	8.622	0.214	0.160

Multithreading et Processeur SS

Processeur SuperScalaire



Hyperthreading

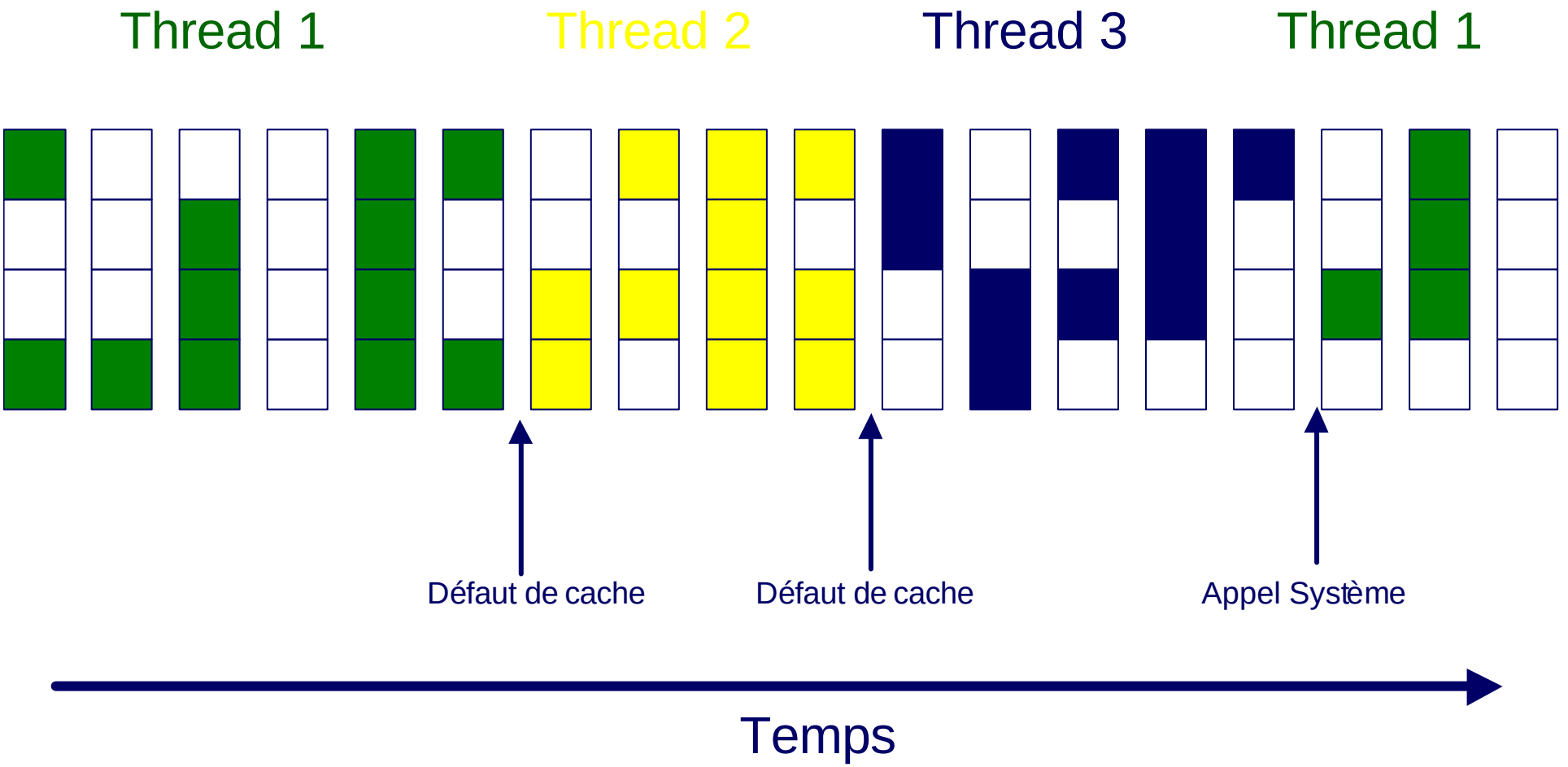
Evolution des architectures de processeurs

Phase 1: Processeurs Multithreadés (1)

- Modification de l'architecture du processeur
  - Incorporer au processeur deux (ou plus) jeux de registres pour les contextes des threads
    - Registres généraux
    - Program Counter (PC), registre d'instruction
    - Process Status Word (PSW), registre d'état
  - A tout instant, un thread et son contexte sont actifs
  - Changement du contexte courant instantané
    - Appel système, IT
    - Défaut de cache
- Processeur IBM PowerPC RS 64
  - Recherche, non commercialisé
- Processeur Intel Xeon Hyperthreading
  - Serveur Bi-processeur Xeon Hyperthreadé
    - Vue de Linux ou Windows, 4 processeurs

Phase 1: Processeurs Multithreadés (2)

Multithread à Gros Grain (Coarse-Grained Multi-threaded)



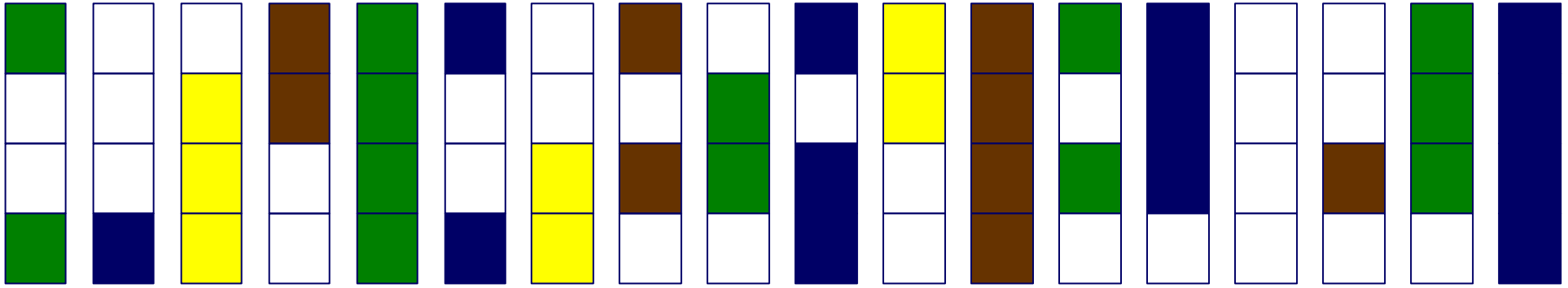
Phase 2: Processeurs Multithreads (1)

- Modification de l'architecture du processeur
  - Incorporer au processeur N jeux de registres pour les contextes des threads
    - Registres généraux
    - Program Counter (PC), registre d'instruction
    - Process Status Word (PSW), registre d'état
  - A tout instant, un thread et son contexte sont actifs
  - Changement de contexte à chaque cycle
    - Chaque thread dispose de son ratio du processeur (1/N)
- Processeur TERA

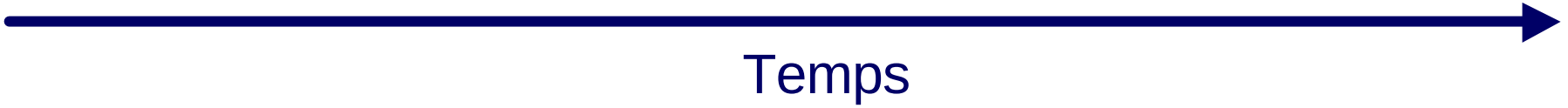
Phase 2: Processeurs Multithreadés (2)

Multithread à Grain Fin (Fine-Grained Multi-threaded)

4 registres de threads : ¼ temps processeur par thread



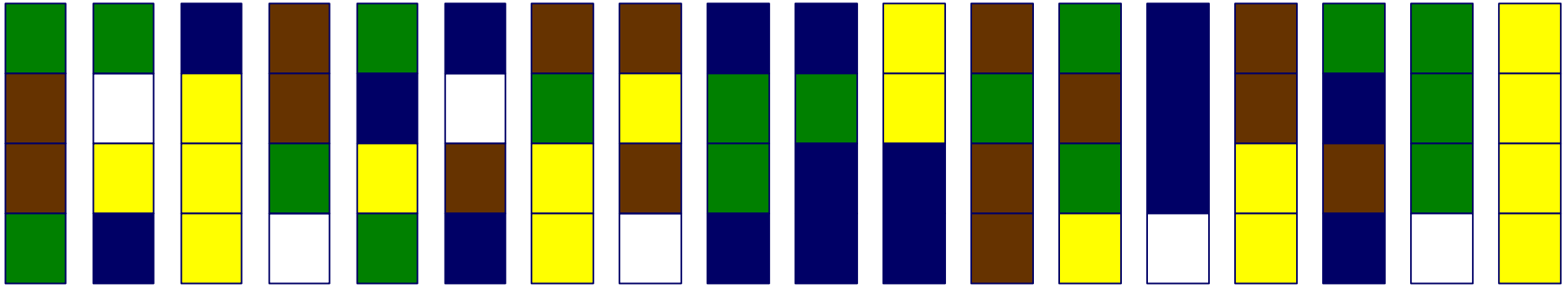
Thread 1 Thread 2 Thread 3 Thread 4



Processeurs HyperThreadés (1)

- Modification de l'architecture du processeur
  - Incorporer au processeur N jeux de registres pour les contextes des threads
    - Registres généraux
    - Program Counter (PC), registre d'instruction
    - Process Status Word (PSW), registre d'état
  - A un instant donné, les unités du processeurs peuvent être partagées entre plusieurs threads
- Eviter de stresser les mêmes ressources
  - Conflit d'accès à certaines unités d'exécution
- Processeur ALPHA EV8

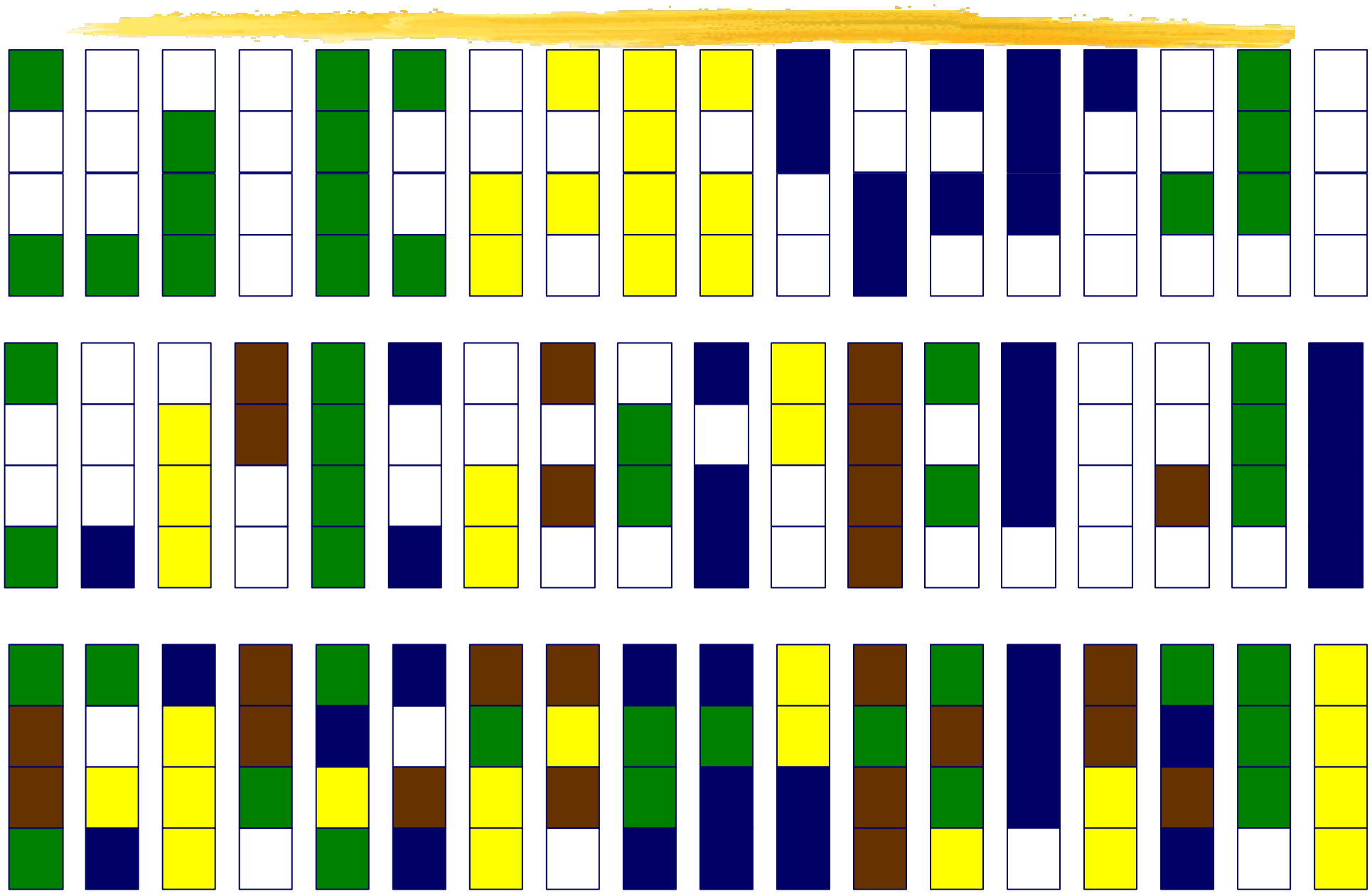
Processeurs Hyper-Threadés (2)



Thread 1 Thread 2 Thread 3 Thread 4

Temps

Résumé



Processeur Itanium2 ® Intel ®

◆Architecture Itanium2 :

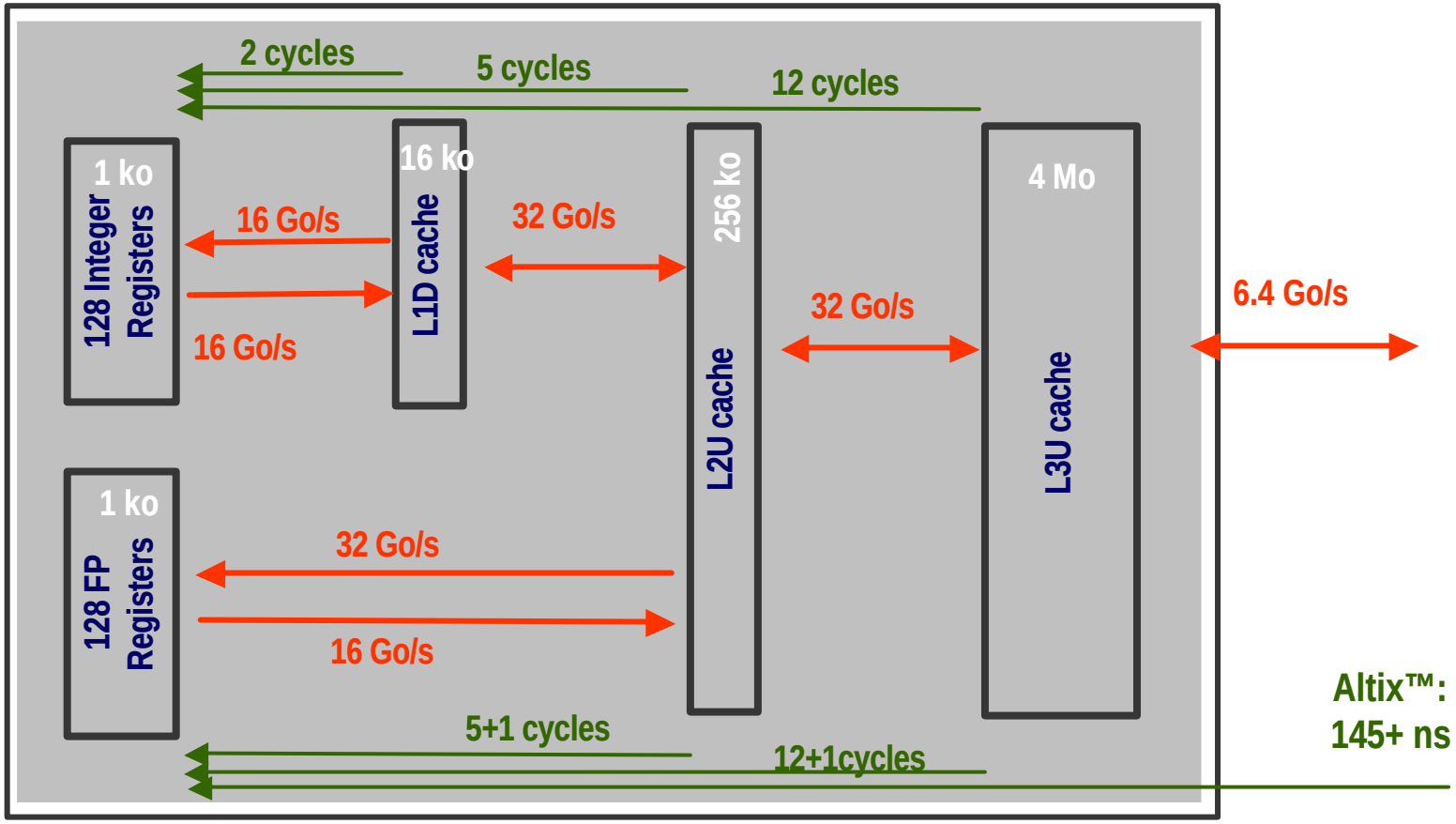
- 4ème génération de processeurs 64 bits Intel : Itanium2 (Madison9M)
- EPIC: Explicit Parallel Instruction Computing
- Fréquence : 1.5 Ghz
- Puissance crête : 6 Gflops/s
  - 1500 MHz * 2 madd/cycle → 6 GFLOPS

◆Intel Itanium2 :

- **L1I** : 16ko; 64o/line ; 4 way
- **L1D** : write through; 16ko; 1/- cycle; 64o/line ; 4 way ; (2ld&2st)/cycle
- **L2U** : write back; 256ko; 5/6cycle; 128o/line; 8 way; (4ldf) | (2ldf[p]&2stf)
- **L3U** : write back; 4Mo; 12/13cycle; 128o/line ; 24 way ; 48Go/s
- **Memory Front Side Bus (FSB)** : 128o/line ; 6.4 Go/s

Processeur Itanium2 ® Intel ®

Débits et latences dans le processeur Itanium



Processeur Itanium2 ® Intel ® : Roadmap

4Gflops Max.

2002

Itanium® 2 (McKinley)

- 900 Mhz, 3.6 Gflops, 1.5 Mo L3 cache
- 1 GHz, 4Gflops, 3 Mo L3 Cache)

>5Gflops Max.

2003

Itanium® 2 (Madison)

- 1.5 GHz, 6Gflops, 6 Mo L3 Cache
- 1.3 Ghz, 5.2 Gflops, 3 Mo L3 cache

Low Power Itanium® 2 (Deerfield)

- 1.0 GHz, 4Gflops, 1.5Mo L3 Cache 62 Watts
- 1.4 Ghz, 4.6 Gflops, 1.5/3 Mo L3 cache

6.4Gflops

Fin 04

Itanium® 2 (Madison 9M)

-1.6 GHz, 6.4Gflops, 6/9 Mo L3 Cache
- 1.5 Ghz, 6 Gflops, 4 Mo L3 cache

Low Power Itanium® 2 Deerfield+ Processor *Follow-on*

>16Gflops

2005

Montecito (Dual Core on a Die) Each Core (>=2 GHz, >=8Gflops, 12Mo L3 Cache)

...

Low Power Montecito Dual Core Processor *Follow-on*

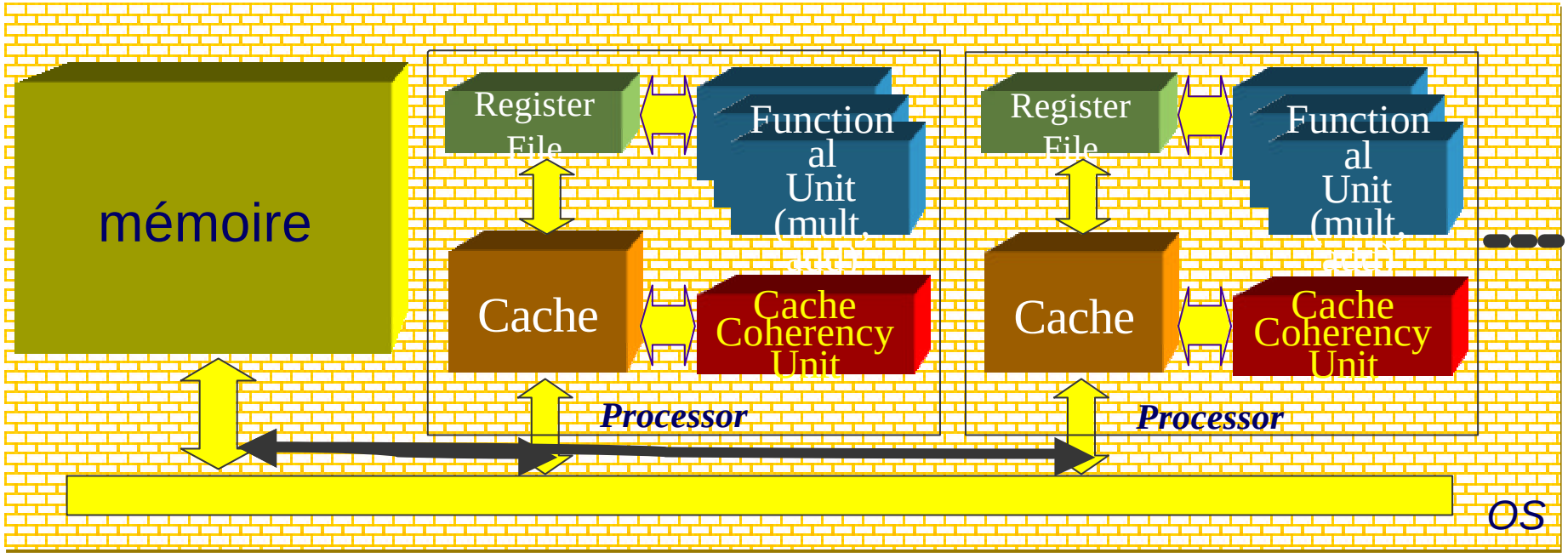
Silicon Process

180 nm	130 nm	90 nm

Problèmes des architectures UMA

Accès à la mémoire:

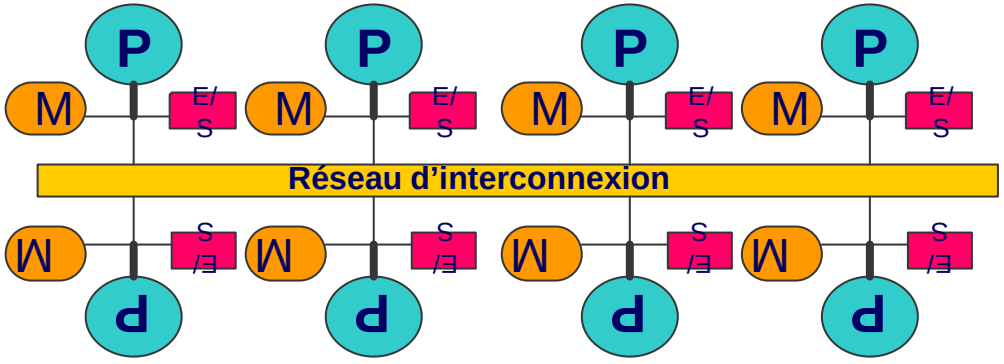
- des niveaux de caches efficaces permettent d'économiser des références à la mémoire
- pour les autres (non cachés), les accès concourants des processeurs à la mémoire partagée
  - un *goulot d'étranglement*



- Cette difficulté peut être levée avec les architectures à *mémoire distribuée*

Mémoire distribuée

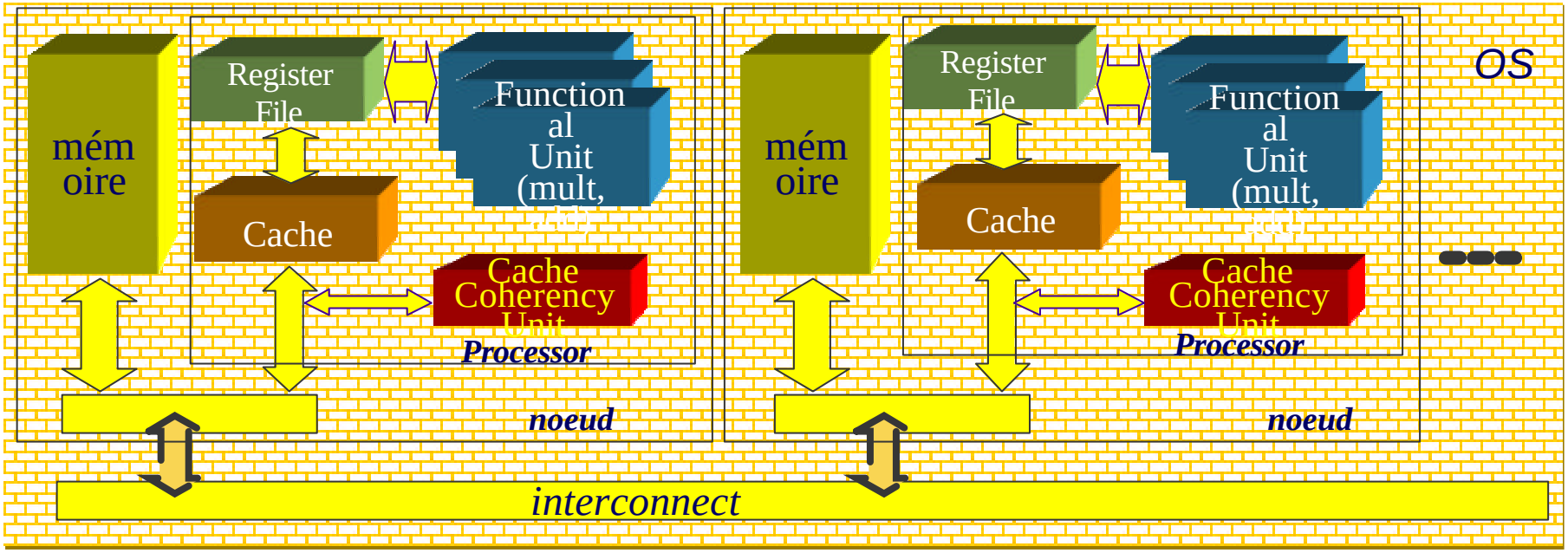
- **Technologie de masse**
- **la bande passante globale mémoire-processeur est proportionnelle au nombre de processeurs**
- **modèles d'exécution : SIMD, MIMD, SPMD**
- **2 paradigmes de communications** :
  - Mémoire partagée : OpenMP (si adressage global), POSIX Threads
  - Mémoire distribuée : MPI, PVM, ...



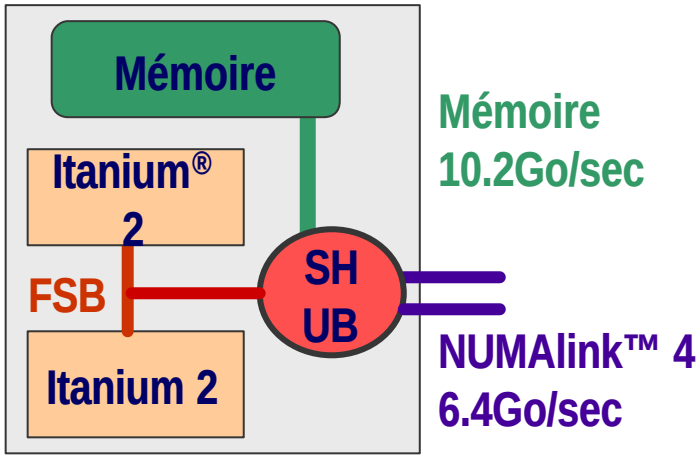
Les noeuds individuels peuvent contenir plusieurs processeurs connectés entre eux par la même technologie que le réseau.

Architecture à mémoire partagée distribuée

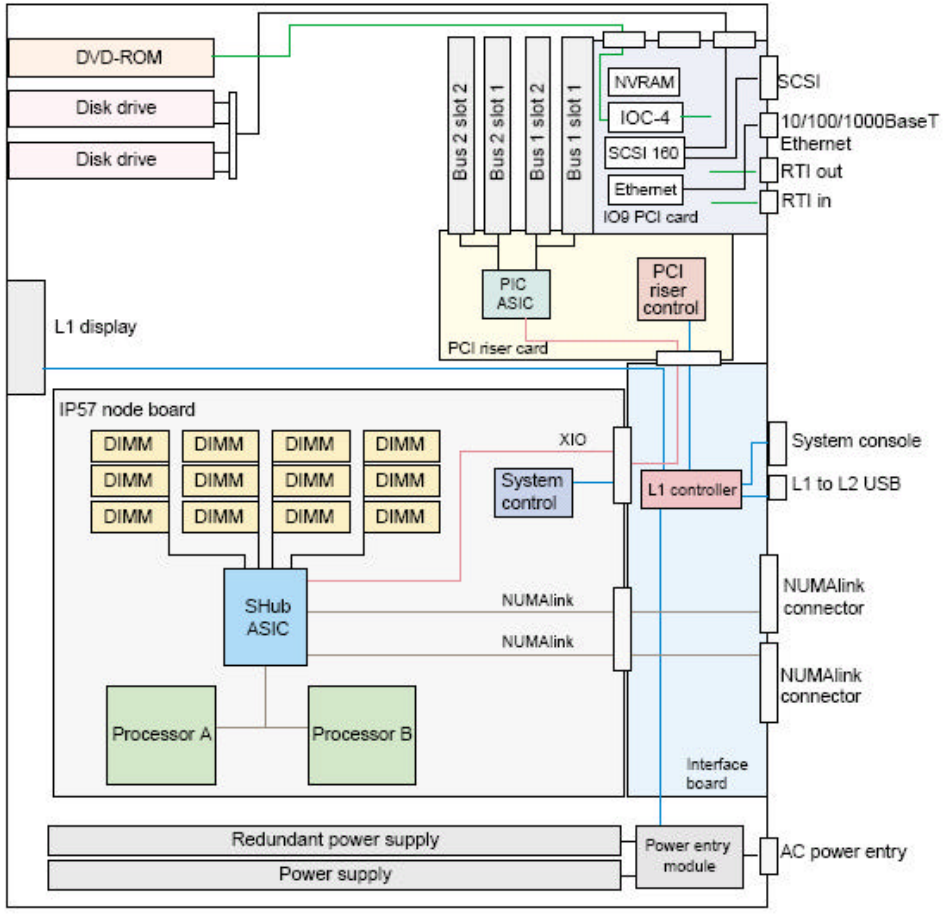
- **Pour chaque processeur,les accès à la mémoire locale sont indépendants**
- **La mémoire totale est globalement adressable (point de vue du programmeur)**
- **Non-uniform memory access (NUMA):**
  - **Les accès locaux sont plus rapides que les accès lointains (peu sensible sur SGI3000/SGIAltix)**
  - **Les modèles de programmation en mémoire partagée sont utilisables**
  - **la distribution des données est conseillée pour améliorer les performances (prise en compte de l 'architecture à mémoire distribuée)**



SGI ALTIX™ 350 : *module de base*



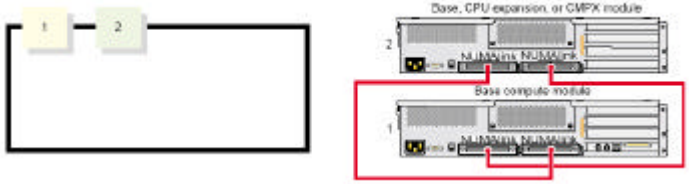
- 2 processeurs Intel® Itanium® 2
- 2 processeurs par frontside bus (6.4Go/sec)
- jusqu 'a 24 Go de mémoire par module
- contrôleur mémoire : SHUB
  8.51–10.2Go/sec bande passante mémoire
- 6.4GB/sec bande passante d 'interconnexion agrégé
- 4.8GB/sec bande passante I/O agrégé



SGI ALTIX™ 350 : Topologies

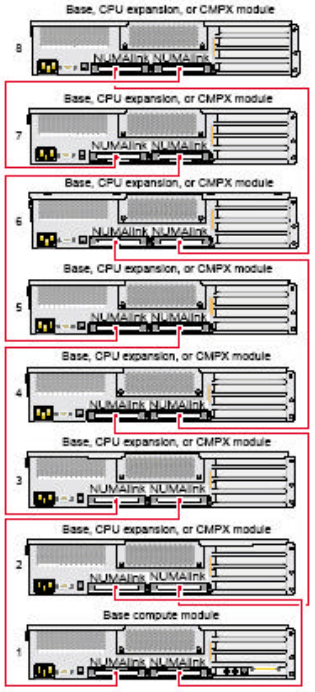
- Une Altix 350 (sans routeur):
  - jusqu 'a 16 cpu en SSI
  - Topologie : anneau

Altix350 4 cpu



Altix350 8 cpu

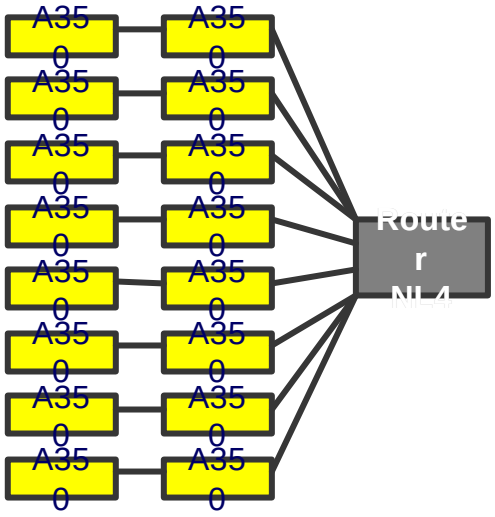
Altix350 16 cpu



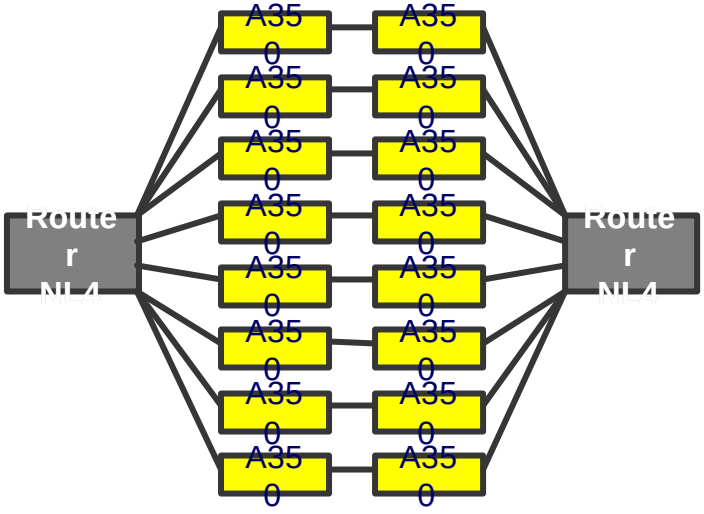
SGI ALTIX™ 350 : Topologies

- Une Altix 350 (avec routeur):
  - Jusqu'à 32 cpu en SSI
  - Topologie : simple/double plan

Configuration simple plan



Configuration double plan



NASA Ames Background

Premier client SGI

« Record extreme computing » :

- Avec un seul OS IRIX :
  - 1er système Origin 2000 128cpu
  - 1er système Origin 2000 256 cpu
  - 1er système Origin 2000/3000 512 cpu
  - 1er système Origin 3000 1024 cpu

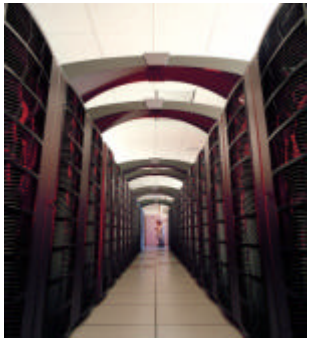
Avec un seul OS Linux :

- 1er système Altix 512 cpu

Origin 2000 256p

Origin 2000 512p

Origin 3000
512p et 1024p



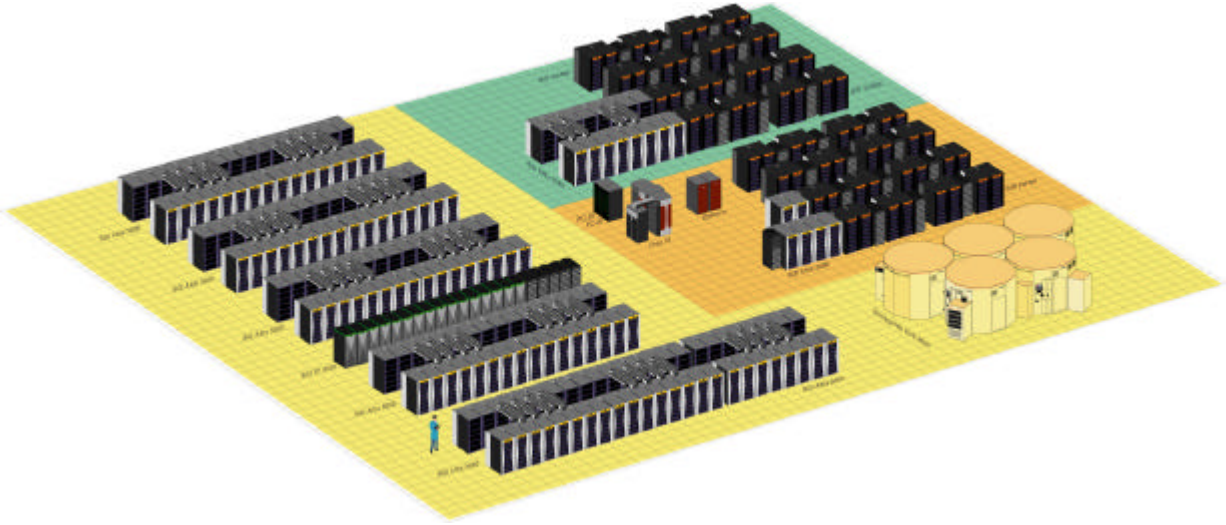
Altix 512p

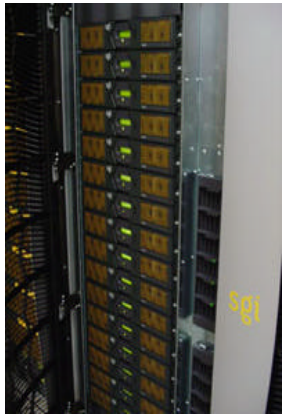
Project Columbia

Projet Columbia

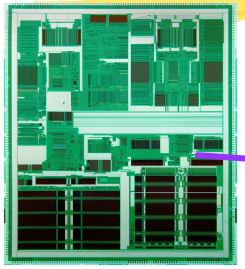
- **Utilisateurs de la NASA et d'autres agences gouvernementales, de la recherche et de l'industrie**
- **10 240 processeurs Intel Itanium® 2**
  - 20 x 512 cpu SGI Altix
  - 2 x 64 cpu SGI Altix ⇒ frontal d'accès
- **20 téraoctets de mémoire totale total mémoire**
  - 1 téraoctets de mémoire par 512 cpu
- Réseau **infiniband**, 1 / 10 gigabit Ethernet
  - 6 Infiniband HCAs / système
- **~500 To stockage gérés par CXFS/DMF**
- 128 x pipes Silicon Graphics Prism
- TOP500 (Novembre 2004, www.top500.org) :
  - Classé numéro 2
  - Rpeak (GFlops):**60960**
  - Rmax (GFlops):**51870**

Configuration Columbia





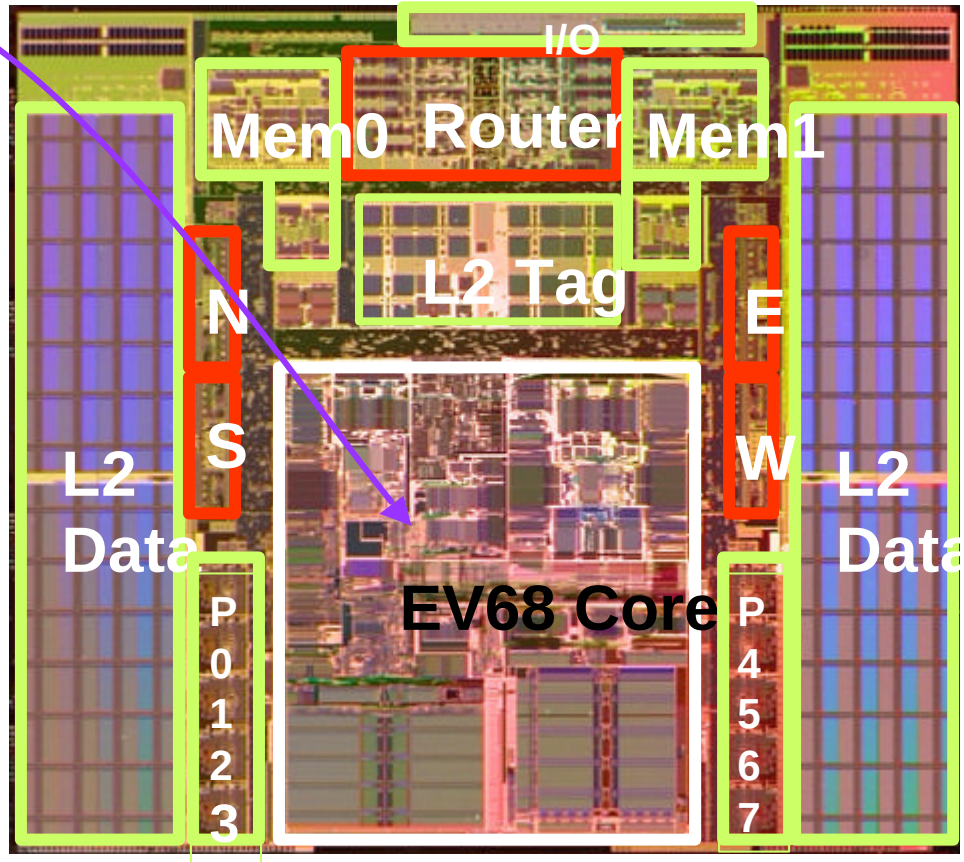
EV7 – The System is the Silicon....



SMP CPU interconnect was external logic..

Now it's on the chip !

- 21264 (EV68) core with enhancements
- Integrated L2 cache
  - 1.75 MB (ECC)
  - 20 GB/s bandwidth
- Integrated memory controllers
  - Direct RAMbus (ECC)
  - 12 GB/s bandwidth
  - Optional RAID in memory
- Integrated network interface
  - Direct processor-processor interconnects
  - 4 links - 25.6 GB/s aggregate bandwidth
  - ECC (single error correct, double error detect)
  - 3.2 GB/s I/O interface per processor



Torus Grid: HP EV7 16P System

EV7 64P Latency

319	283	247	211	247	283	319	355
283	247	211	175	211	247	283	319
247	211	175	140	175	211	247	283
211	175	140	75	140	175	211	247
247	211	175	140	175	211	247	283
283	247	211	175	211	247	283	319
319	283	247	211	247	283	319	355
355	319	283	247	283	319	355	391

250ns average memory latency

Processor Sets

- A processor set is a collection of zero or more processors
- Every thread on the system belongs to a processor set
- A thread may only be scheduled on a processor within its processor set
- At bootup, a default processor set is created and this “default_pset” contains all processors on the system

Processor sets

- Psets can be created or destroyed
  - default processor set is the exception
- processors can be added or removed from processor sets

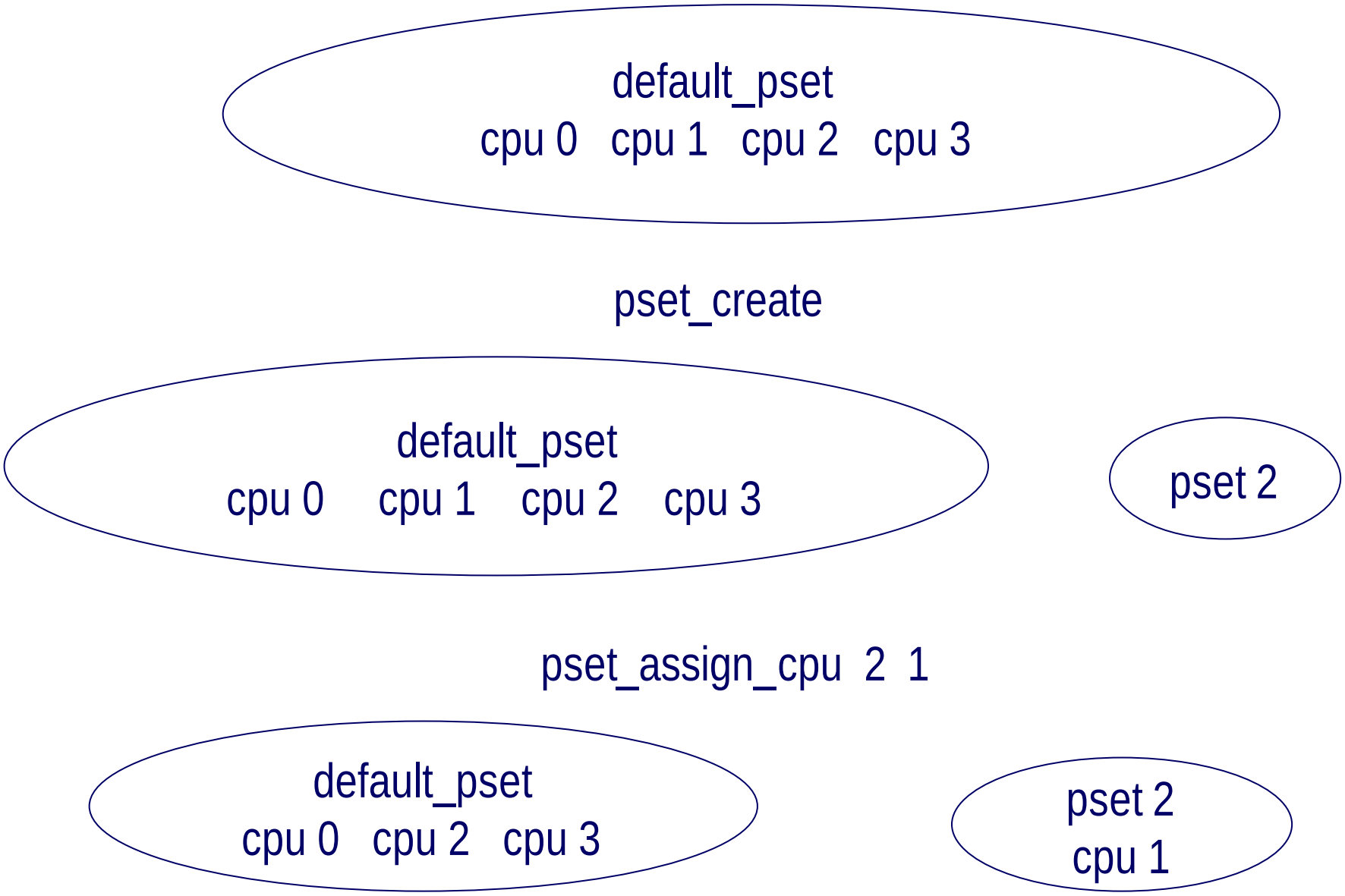
Processor sets

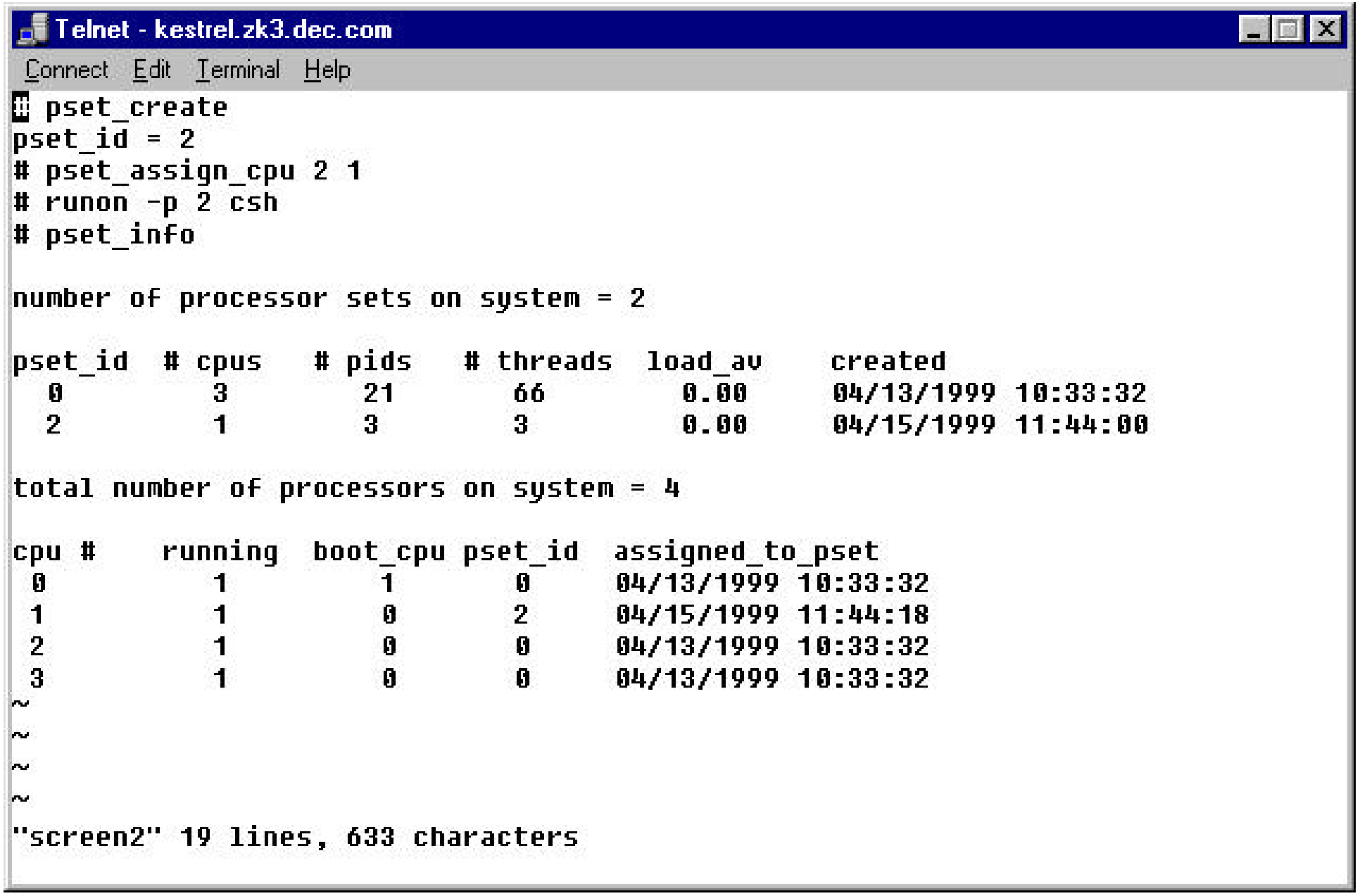
- Processors sets allow you to run jobs on a specific group of processor(s).
  - If there is only one process running on a processor, there can be a performance boost because of cache & scheduling efficiencies.
  - good for real-time applications because a processor can be guaranteed to be instantly available to run a time critical task.
  - Another way to guarantee cpu resource availability

Processor Sets

- By default, all threads are assigned to the `default_pset`
- new pset can be created with the `pset_create` command. This pset will have no processors.
- Processors can be added to a pset with the `pset_assign_cpu` command.
- If a created pset is destroyed (pset_destroy command), all processors from that pset are returned to the default_pset (V4.X)

Processor Sets : example




Telnet - kestrel.zk3.dec.com
Connect Edit Terminal Help
# pset_create
pset_id = 2
# pset_assign_cpu 2 1
# runon -p 2 csh
# pset_info
number of processor sets on system = 2
pset_id # cpus # pids # threads load_av created
0 3 21 66 0.00 04/13/1999 10:33:32
2 1 3 3 0.00 04/15/1999 11:44:00
total number of processors on system = 4
cpu # running boot_cpu pset_id assigned_to_pset
0 1 1 0 04/13/1999 10:33:32
1 1 0 2 04/15/1999 11:44:18
2 1 0 0 04/13/1999 10:33:32
3 1 0 0 04/13/1999 10:33:32
~
~
~
~
"screen2" 19 lines, 633 characters

Using Processor sets

- Runon (1) command
  - generally binds a job to a processor within its processor set
    - runon 1 ls runs the ls command on processor 1
  - runon -p binds a job to another processor set
    - runon -p 2 csh runs a c-shell on processor set 2
    - now all commands issued from that shell will be bound to processor set 2
  - runon -p 2 -x csh gives exclusive use of the processor set

Comment ordonnancer efficacement?

- Performance
  - Affinité entre threads et mémoire prise en compte
- Flexibilité
  - Exécution contrôlée par l'application et le programmeur
- Portabilité
  - Application doit s'adapter à de nouvelles architectures ou topologies

Ordonnancement de threads sur machines hiérarchiques

- Trois approches d'ordonnancement
  - pré-calculé
  - Opportuniste
  - Négocié

Ordonnancement pré-calculé

- Etape préliminaire
  - Placement des données
  - Ordonnancements de threads
- Exécution
  - Threads et données sont placés explicitement par le programmeur, sans l'aide du système
- Example
  - Pastix: Solveur d'algèbre linéaire (LaBRI, Bordeaux)
  - Performances excellentes
  - Difficile sur les problèmes très irréguliers

Ordonnancement opportuniste

- Algorithmes gloutons (self-scheduling)
  - Single task list (UMA)
  - Multiple task list (NUMA)
  - Groups of task lists
- Algorithmes implantés dans les systèmes actuels
  - Linux, FreeBSD, Solaris, Windows,...
- Portabilité
  - Passage à l'échelle
- Performance
  - Aucune information sur l'affinité des processus

Ordonnancement négocié

- Extensions de langages (directives)
  - OpenMP, HPF
  - Le compilateur ajoute du code pour gérer les threads
- Performance
  - Adaptation à l'architecture de la machine
- Flexibilité
  - Manque de généricité de l'interface

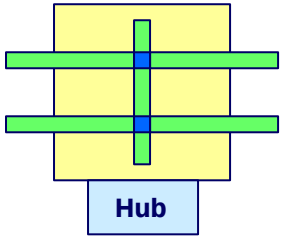
- Support du système d'exploitation
  - Libnuma (Linux), liblgroup (Solaris)
  - Les programmeurs ne sont pas concernés par ce niveau

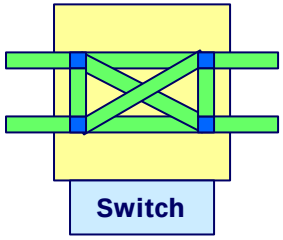
Technologies matérielles

Cartes passives, actives, réseaux à capacité d'adressage

(Fast|Giga)-Ethernet

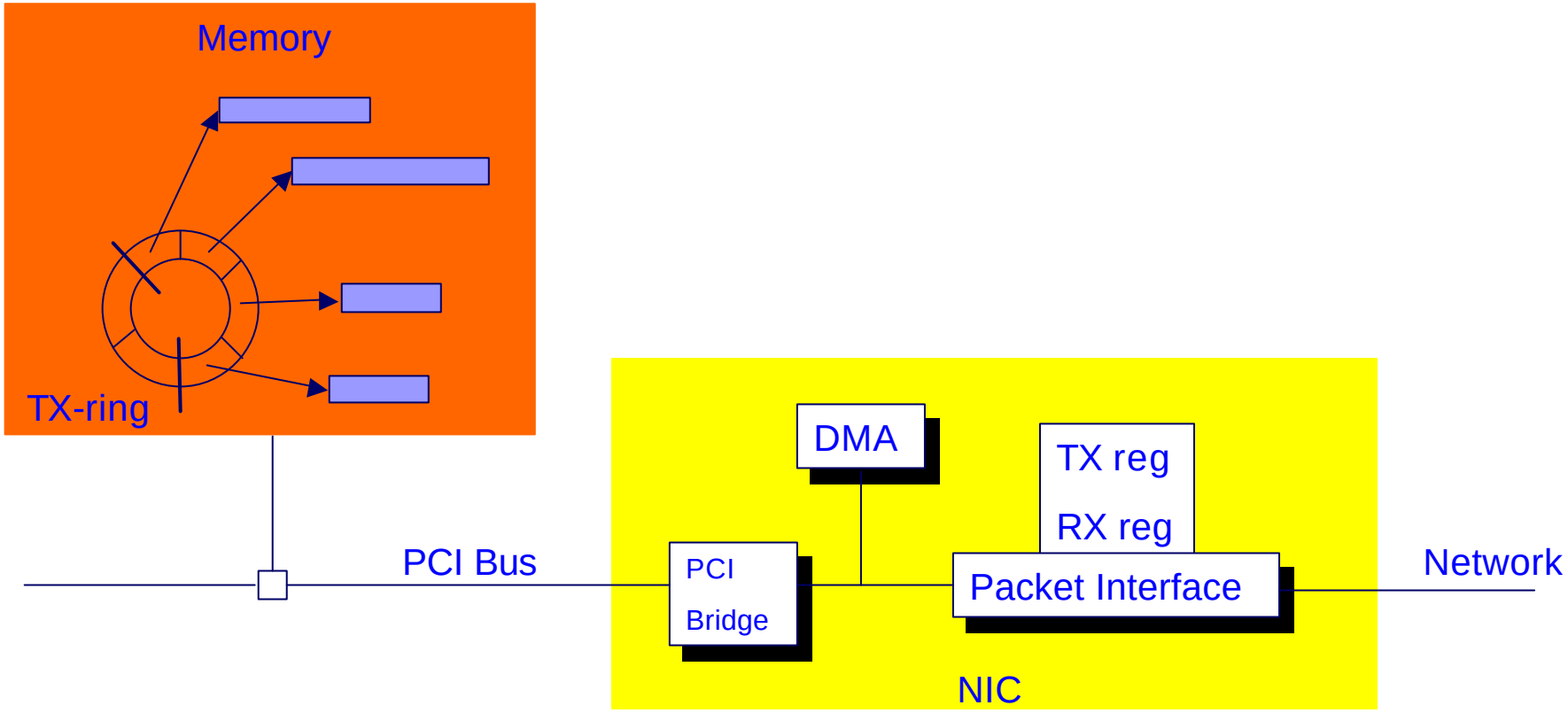
- Interconnexion
  - Hub ou switch
- Câblage
  - Cuivre ou fibre optique
- Latence
  - ~20 µs
- Débit
  - 100 Mb/s et 1Gb/s
- Note
  - Compatibilité avec l'Ethernet classique





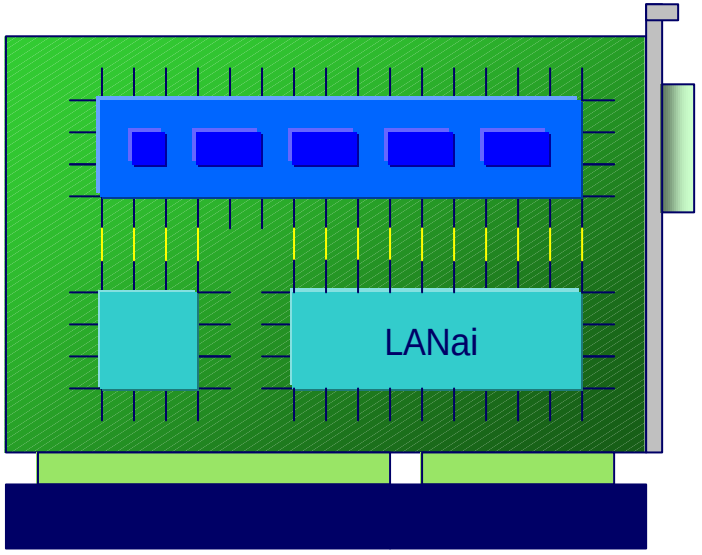
Ethernet

- Cartes passives (sauf Giga-Ethernet)
  - Interfaces : TCP, SBP, GAMMA, VIA, ...



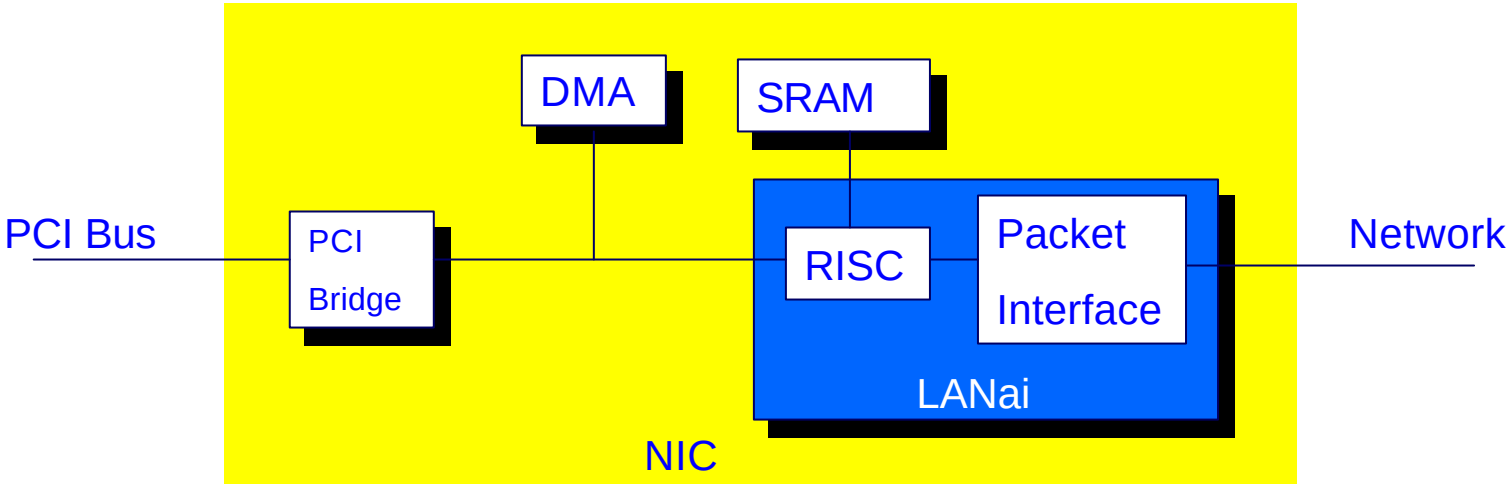
Myrinet

- Société Myricom (C. Seitz)
- Interconnexion
  - Switch
- Câblage
  - Nappes courtes
- Cartes équipées d'un processeur
- Latence
  - 1~2 µs
- Débit
  - 1 Gb/s
- Note
  - Durée de vie limitée des messages (50 ms)



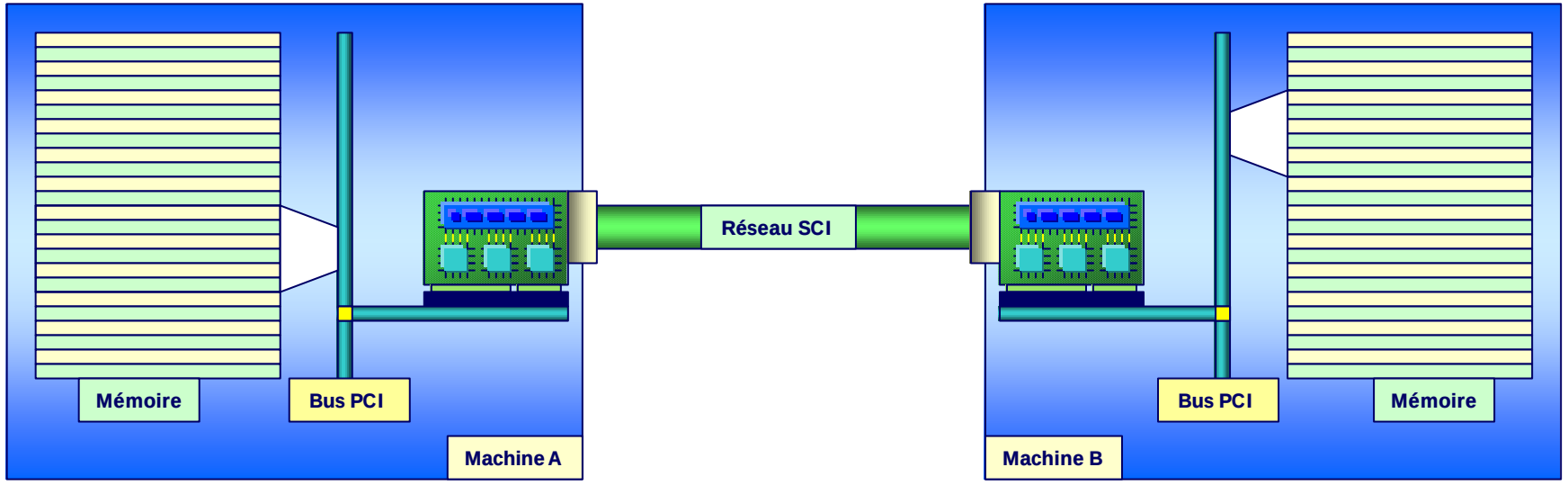
Myrinet

- Routage
  - Réseau commuté, routage *wormhole*
- Carte programmable
  - Protocoles de transmission « intelligents »
    - Stratégie adaptée à la taille des messages
    - Déclenchement d'interruptions

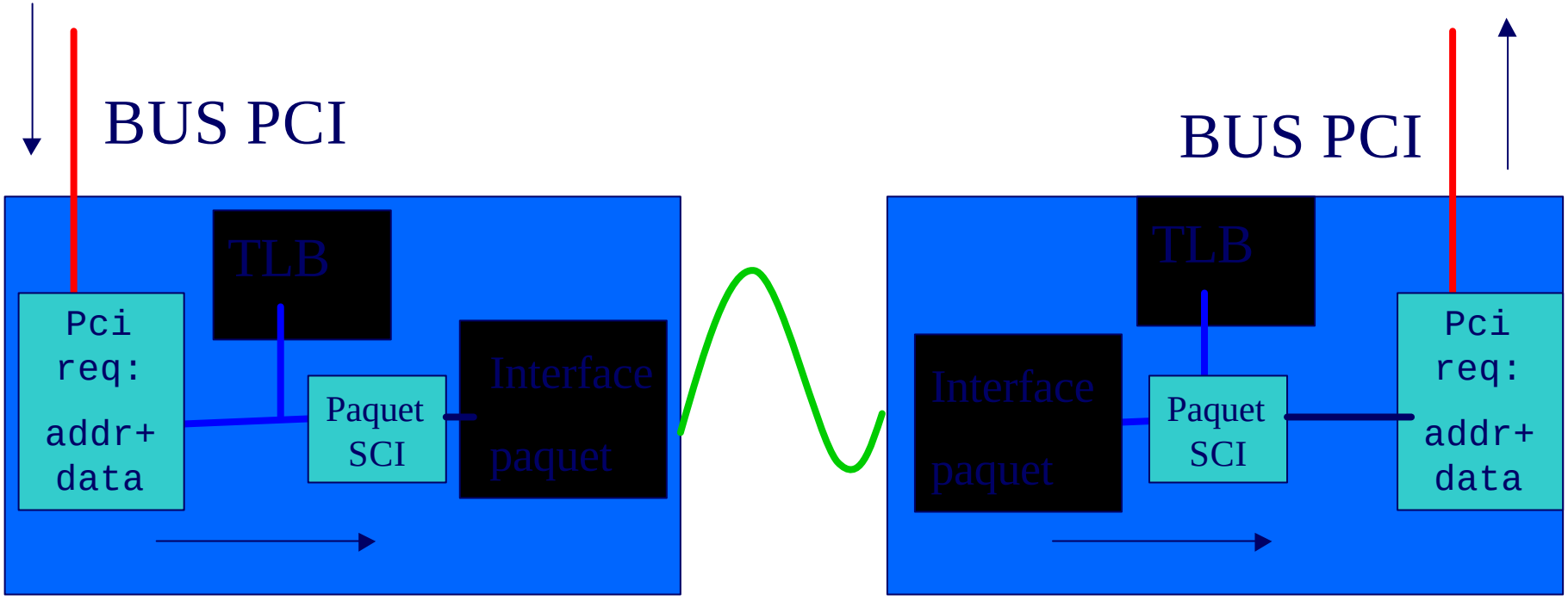


SCI

- Scalable Coherent Interface
  - Norme IEEE (1993)
  - Société Dolphin
- Fonctionnement par accès mémoire distants
  - Projections d'espaces d'adressage



Carte à capacité d ’addressage

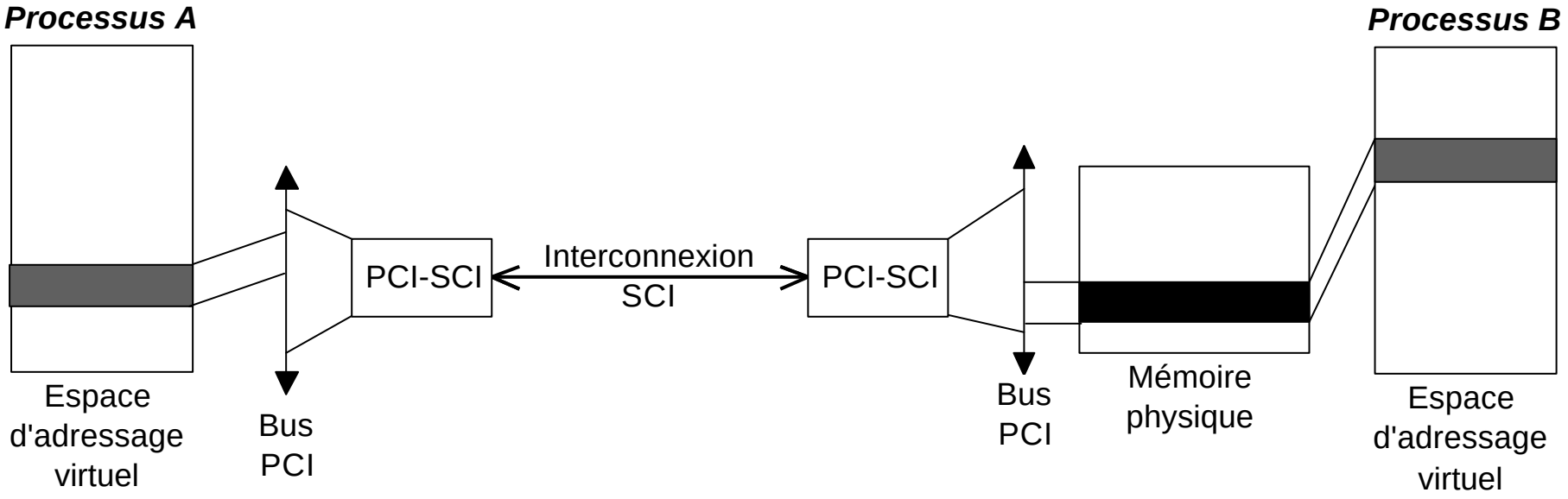


Partie émission

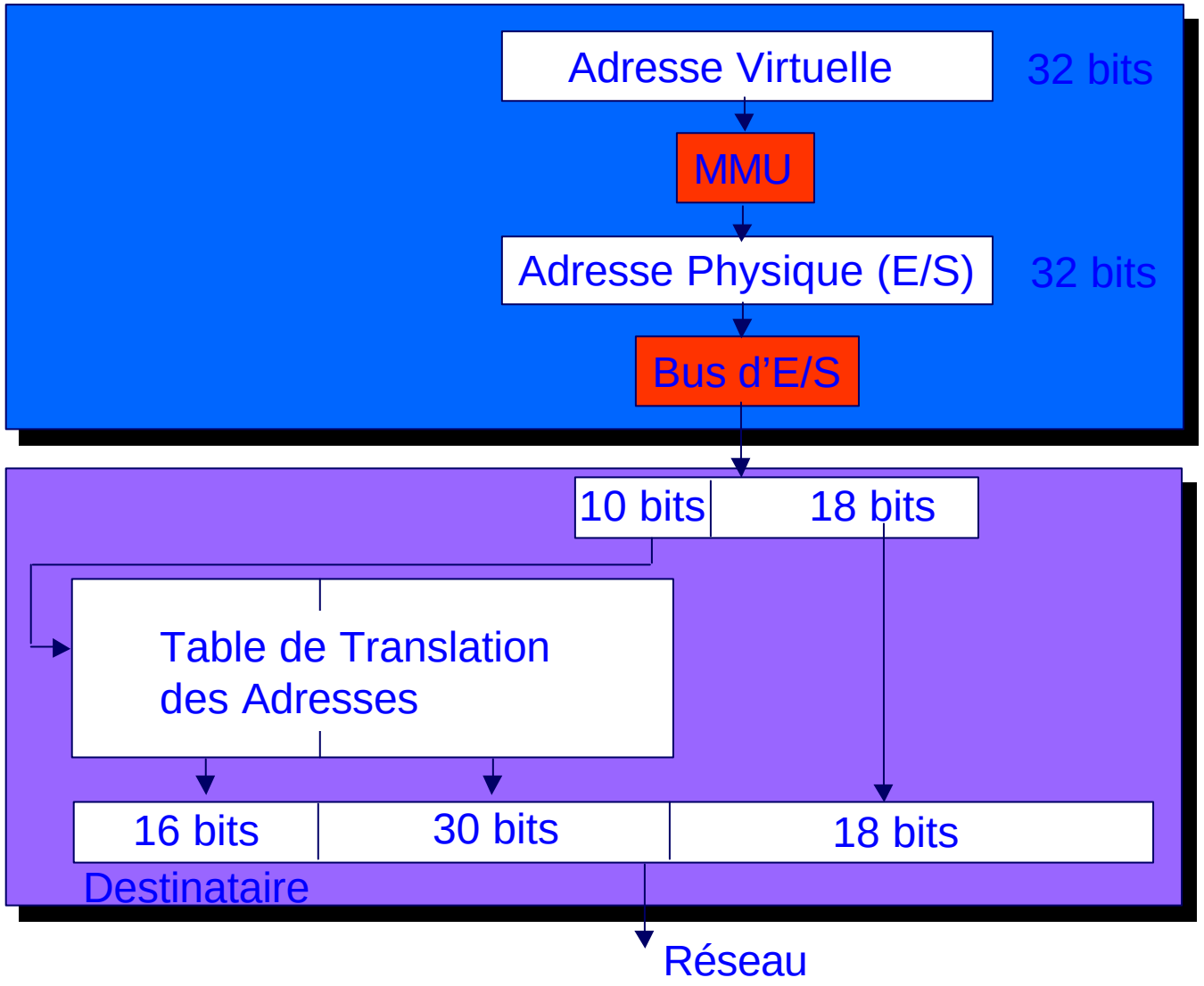
Partie réception

Adressage à distance

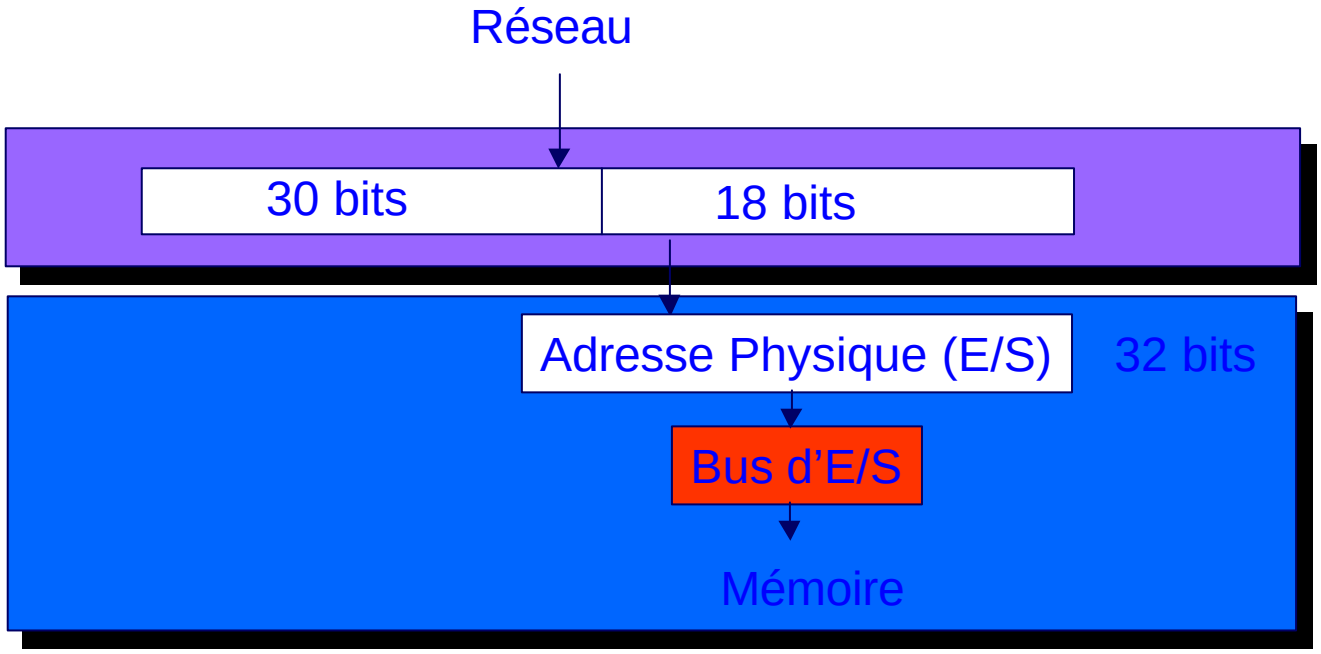
- Projections effectués par le pilote (SISCI)
  - Zones de mémoire physiques souvent spécifiques
- Accès mémoire effectués par le processeur
  - Le processeur distant n'est pas (forcément) interrompu



SCI : mécanisme d'adressage

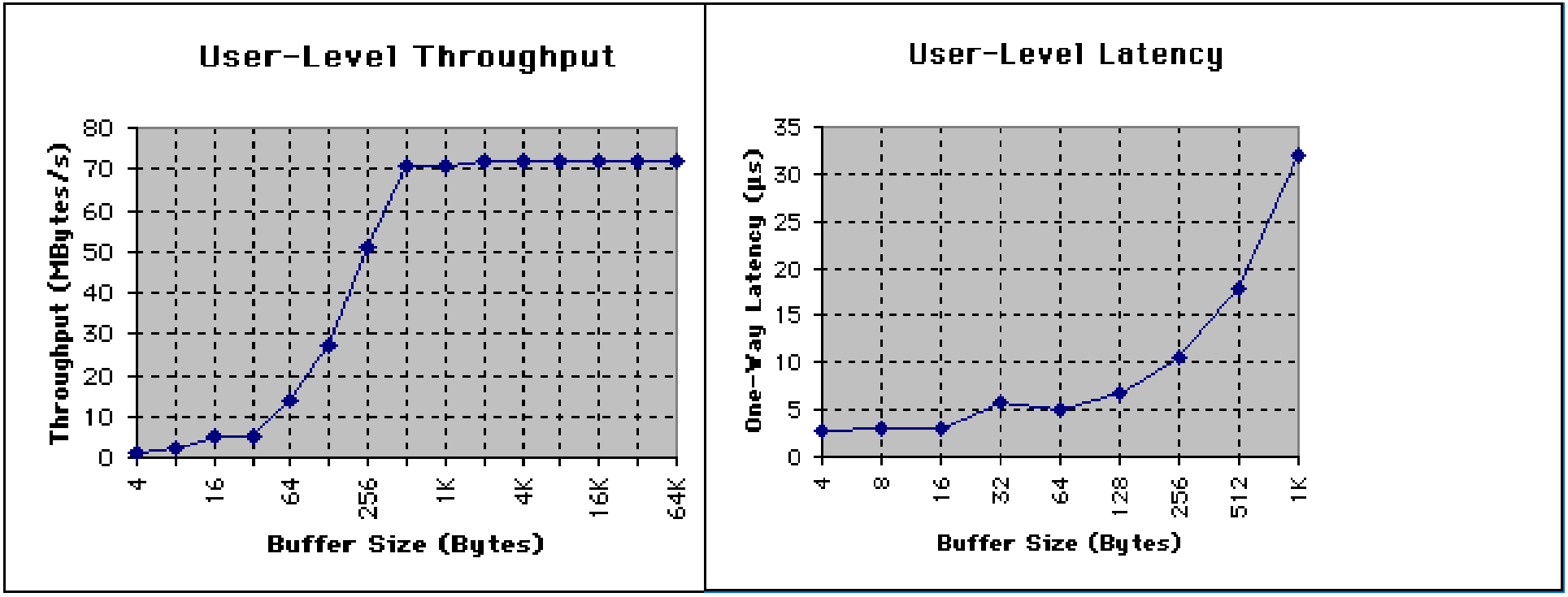


SCI : mécanisme d'adressage



SCI : performances

- Latence : 2.5 µs (écriture de processus à processus)
- Débit : 45 Mo/s
- Verrou : < 10 µs (fetch&add)



Interfaces de bas niveau

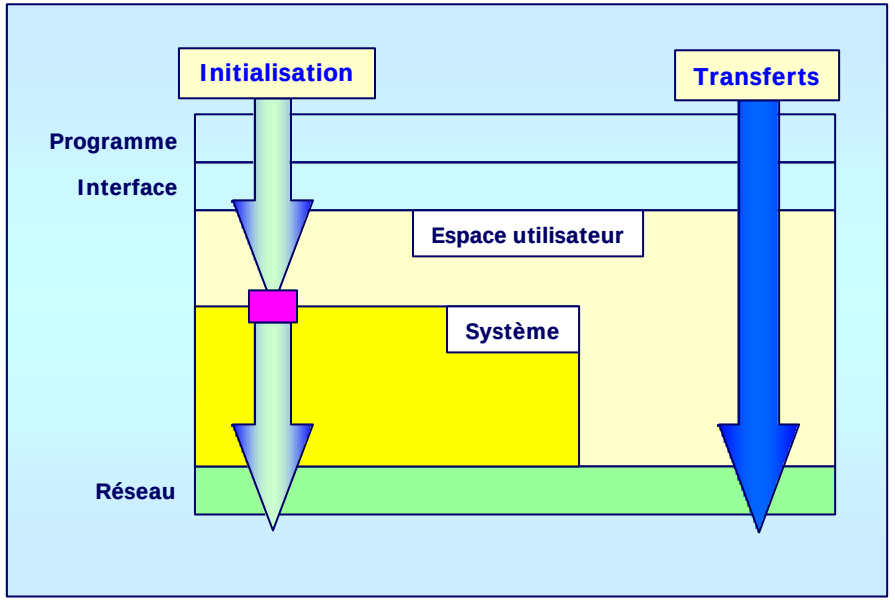
BIP, SISCI, VIA

Communications performantes

- Comment exploiter les réseaux rapides ?
  - Faible latence
    - Quelques microsecondes
  - Bande passante élevée
    - De l'ordre du Gb/s
- Tendance actuelle
  - Interaction directe avec la carte réseau
    - Communication en « mode utilisateur »
  - Transmissions zéro-copie
    - La carte récupère/dépose les données au bon endroit

Interfaces

- Initialisation
  - Réservée au système
  - Uniquement en début de session
- Transferts
  - Directs depuis l'espace utilisateur
  - Pas d'appels systèmes
  - Pas de transitions
  - Transmissions zéro-copie



Streamline Buffer Protocol

- UNH (R. Russell & P. Hatcher)
- Principe
  - Partage de tampons entre noyau/processus
  - Tampons préformatés (trames ethernet)
- Deux jeux de files de tampons : RQ & SQ
- Performances (Linux, P133)
  - Latence : 24 us, débit : ~ 12 Mo/s

Basic Interface for Parallelism: BIP

- L. Prylli & B. Tourancheau
- Principe
  - Envoi de message “classique” (asynchrone)
  - Pas de contrôle de flux
  - Pas de detection d’erreur
- Performances
  - Latence : 4.8us, débit : 126 Mo/s

BIP

- Fonctionnalités réduites au minimum
  - Messages courts : recopiés à l'arrivée
  - Messages longs : mode zéro-copie (RDV)
- Contrôle de flux minimal
  - Matériel (msgs "évaporés" au dela de 50ms)

Interface Dolphin pour SCI

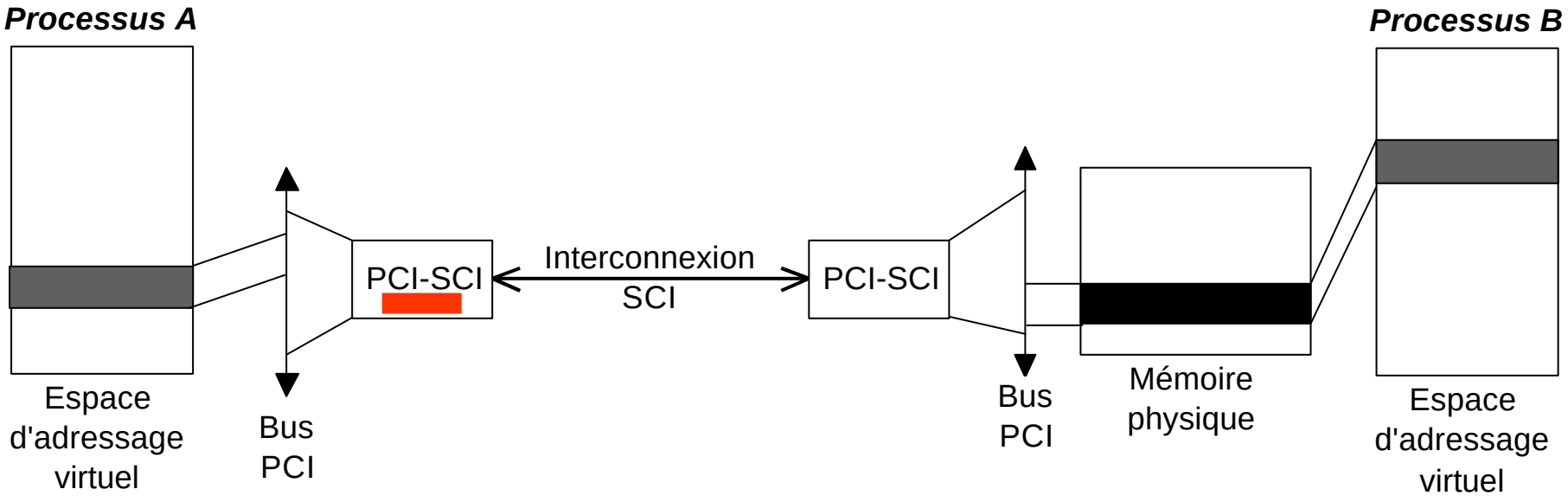
- Deux niveaux :
  - Interface orientée “fichiers projetés”
  - Interface orientée “VIA” (cf + loin)
- Fichiers projetés
  - Utilisation de “mmap”
- Synchronisation
  - Segments spéciaux “fetch & add”

SISCI: principe

- Communications
  - Accés mémoire distants
    - Implicites (après projection)
    - Explicites (*remote DMA*)
  - Support matériel pour une MVP (?)
- Performances
  - Ecriture ≈ 2 us, lecture ≈ 4 us
  - Bande passante ≈ 85 Mo/s (difficilement !)

SCI : optimisations matérielles

- Caches dans la carte :
  - Read Prefetching
  - Write Combining



Conséquences

- Une Grappe SCI est une NC-NUMA
  - Non-Coherent Non-uniform Memory Arch.
  - Cohérence séquentielle non vérifiée
- Plus gênant : ordre des écritures modifié
  - Pb d'atomicité des transactions PCI
  - Assez inhabituel (?)

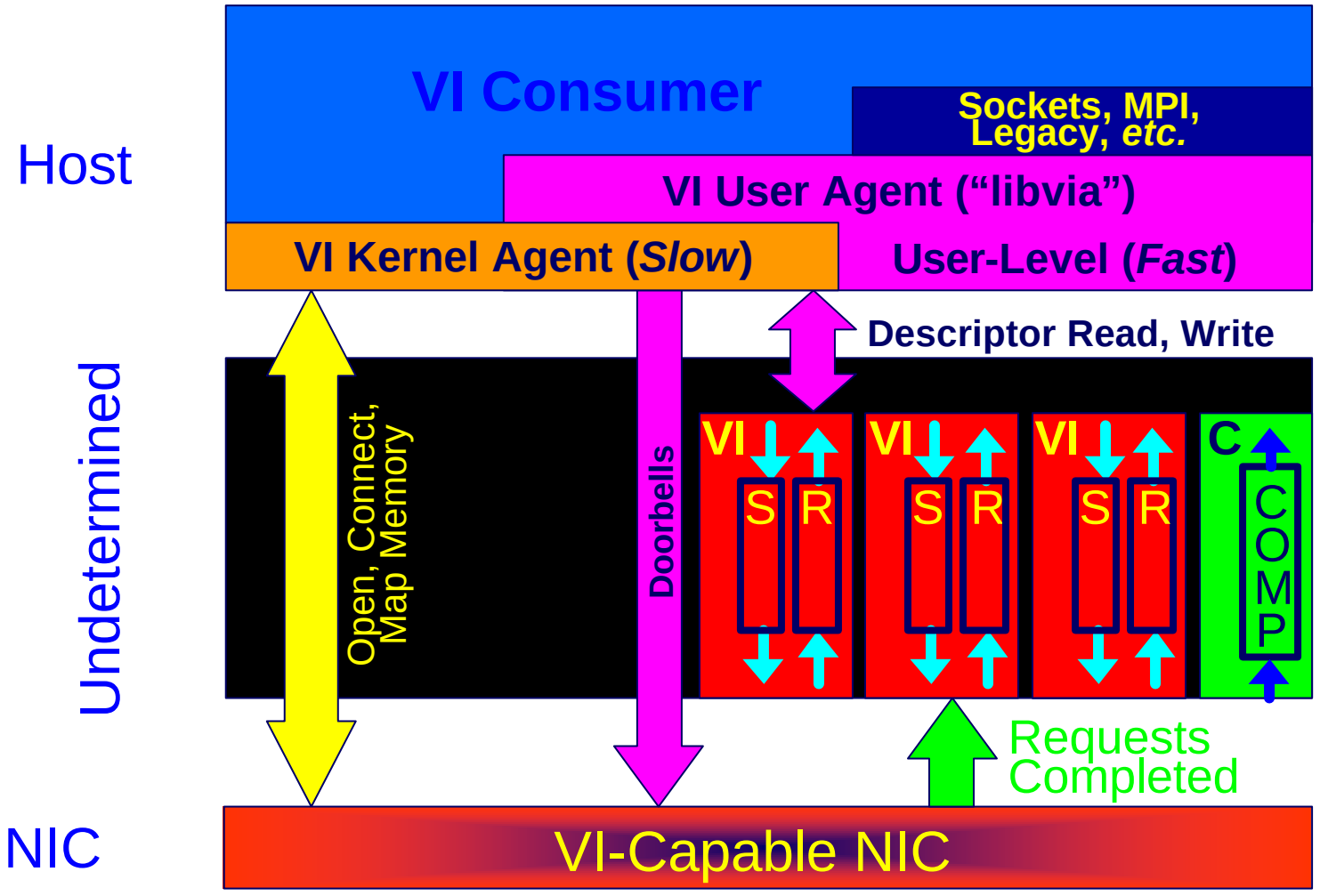
VIA

- Virtual Interface Architecture
- Tentative de standardisation
  - Beaucoup d'industriels impliqués
- Caractéristiques
  - Principe d'interfaces virtuelles
  - Zones de transmission protégées
  - Lectures/Ecritures distantes

VIA: Basic Principles

- Use the Kernel for Set-Up...
  ...and *Get It Out of the Way* for Send/Receive!
- The "Virtual Interface" (VI)
  - Protected Application-Application Channel
  - Memory *Directly Accessible* by User Process
- Target Environment
  - LANs and "SAN"s at Gigabit Speeds
  - No Reliability of Underlying Media Assumed

VI Architecture



VI Kernel Agent

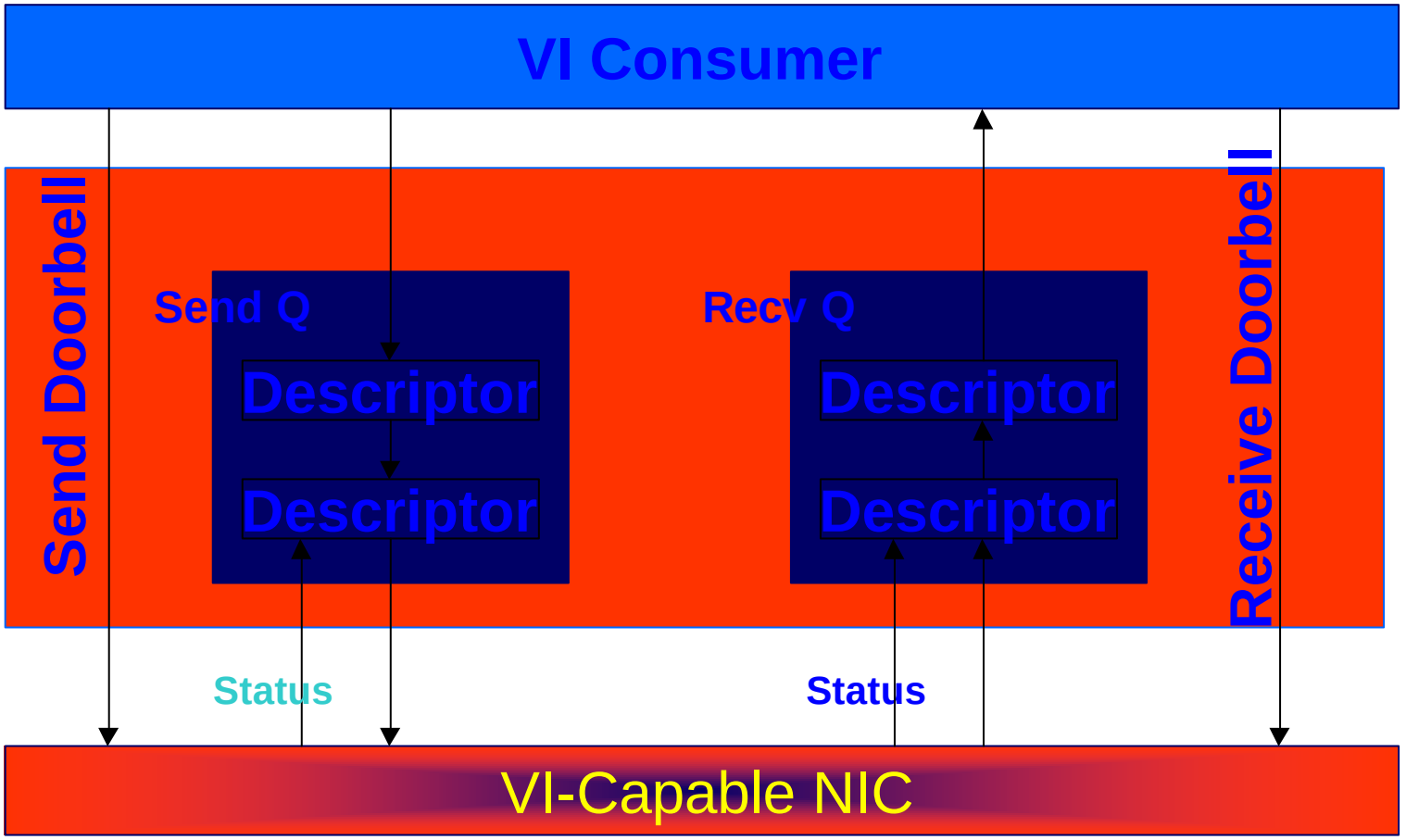
A Privileged Part of Operating System (driver)

- Usually supplied by the VI NIC vendor
- Possibly supplied by research groups (UCB,…)

Setup and Resource Management Functions

- Creation/Destruction of Vis
- VI Connection setup/tear-down
- Management of System Memory used by the VI NIC

A Virtual Interface



Descriptors

- Descriptors Contain:
  - Address and Length of Data Buffer
  - Status Fields
  - Memory Protection Information
  - Multiple Segments to Allow Scatter/Gather
  - *etc., etc., etc.*
- A minimum of 45 bytes long
  - Many messages may only be a few bytes...

Queues and Doorbells

Queues of Descriptors
- Transmit and Receive
- Completions and Errors
- May Reside on Host or NIC (Unspecified)

Doorbells
- “Addresses” of Descriptors, Small and Fast
- Allows NIC to Use Descriptors…
- Future “VIA-NICs” May Have Hardware Support

Memory Registration

- Data buffers and descriptors must reside within a region of "registered memory"
- Call **`VipRegisterMemory`**
- Pins the specified pages into physical memory
- Communicates the addresses to the NIC
  - To allow DMA I/O from the NIC

Ce qu'il faut retenir

- Interfaces de très bas niveau !
  - Fonctionnalités proches du matériel
    - Grande efficacité
    - Paradigmes très différents
    - Approche non généralisable
  - Pas de consensus
    - Tentative de standard : VIA
      - Virtual Interface Architecture (Intel, Microsoft, Compaq)
      - But : dénominateur commun
      - Bas niveau, peu adapté à certaines technologies
  - Portabilité ???

Interfaces de haut niveau

MPI : la solution idéale ?

Bibliothèques

- Paradigme passage de message
  - Les nœuds se synchronisent et communiquent par messages
- Deux instructions de base
  - Send émission d'un message
  - Receive réception d'un message
- Points forts
  - Simple à mettre en oeuvre
  - Permet d'émuler les autres paradigmes

PVM

- Parallel Virtual Machine
  - Laboratoire National d'Oak Ridge (Tennessee)
  - 1989
- Caractéristiques
  - Notion de machine virtuelle
    - Ensemble de machines physiques
    - Construction préalable au lancement de la session
  - Disponibilité très large
  - Réseaux
    - UDP + protocole de réémission
  - Support de l'hétérogénéité
    - XDR

MPI

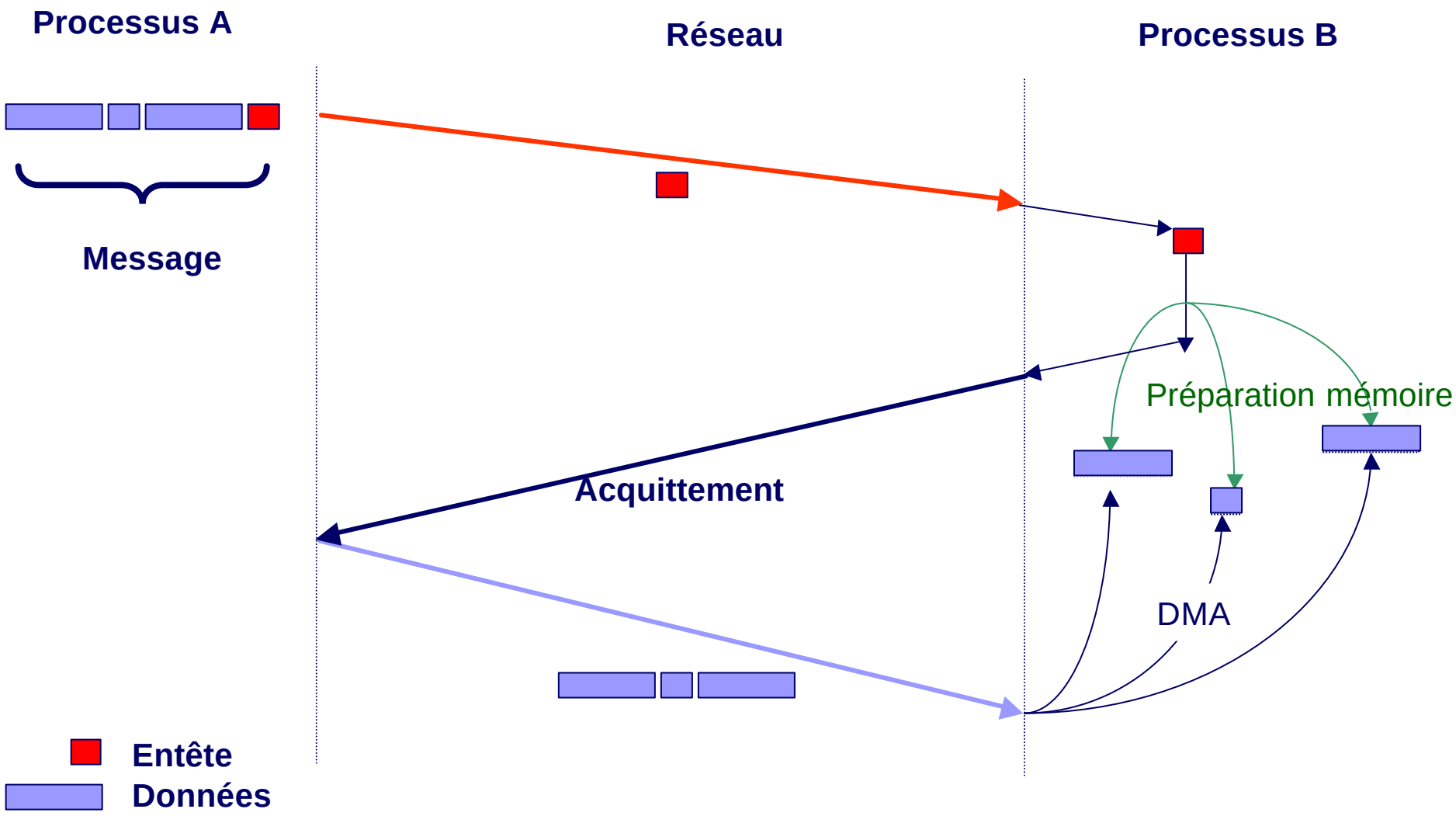
- Message Passing Interface
  - MPI-Forum v1.0 1994 v2.0 1997

- Caractéristiques
  - Un standard, pas une bibliothèque
  - Diverses implémentations
    - MPI-CH
    - LAM-MPI
    - ...
  - Supplante PVM
  - Version 2.0 encore peu implémentée

MPI répond-t-il aux besoins ?

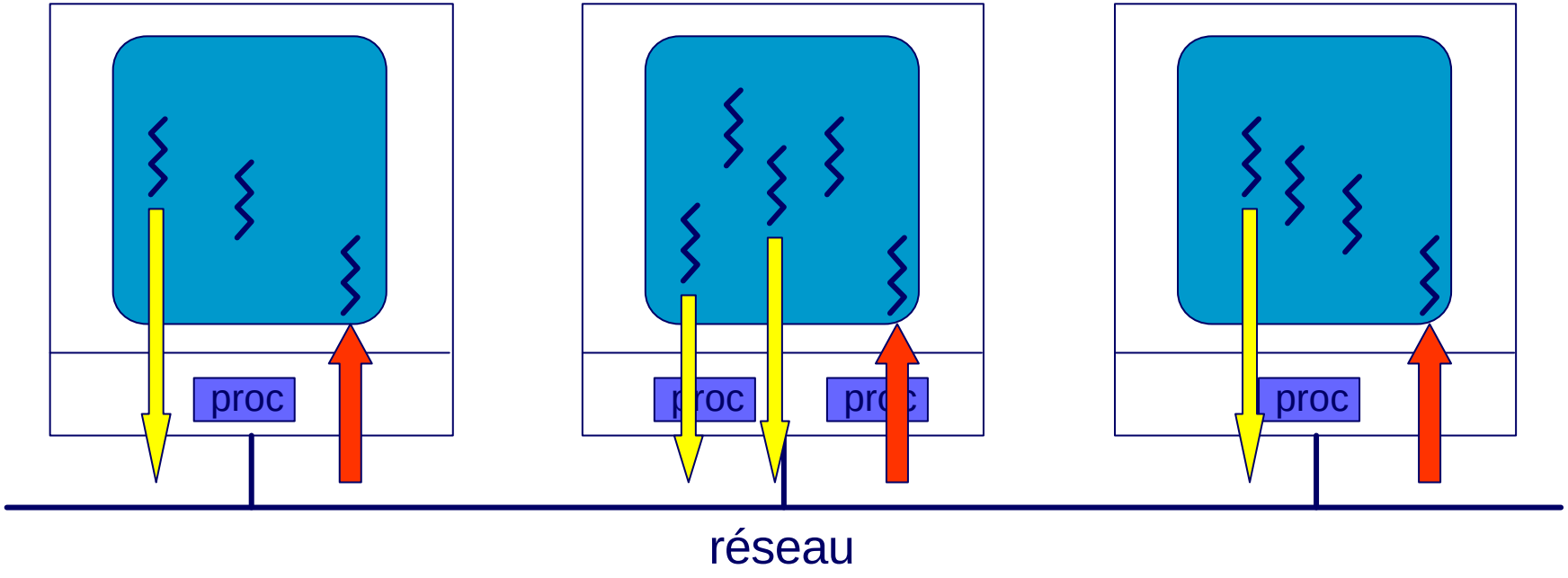
- Implantations efficaces existantes
  - MPICH/BIP, MPICH/SISCI, etc.
- Quid des schémas de communication de la vraie vie ?
  - Messages dont le contenu est inconnu *a priori* par le récepteur
    - Transmissions zéro-copie ?
  - Messages asynchrones
    - Recouvrement des communications ?
  - Accès mémoire distants (PUT/GET)
    - Temps de réponse ?

Transmissions zéro-copie



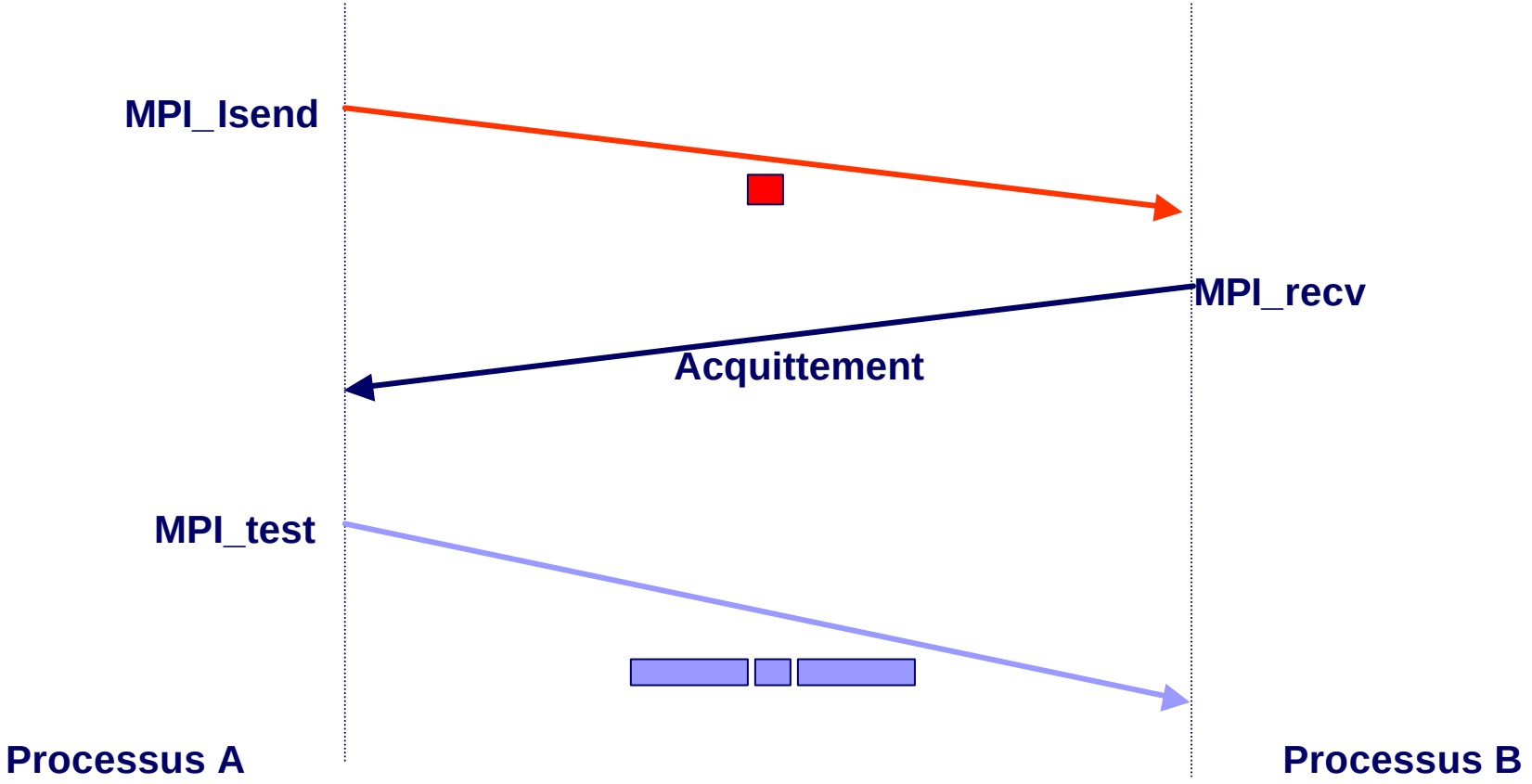
Et la réactivité alors ?

- Problèmes
  - Assurer la progression des communications asynchrones
  - Réagir rapidement aux sollicitations extérieures



Envois asynchrones

- Parvient-on vraiment à assurer du recouvrement ?

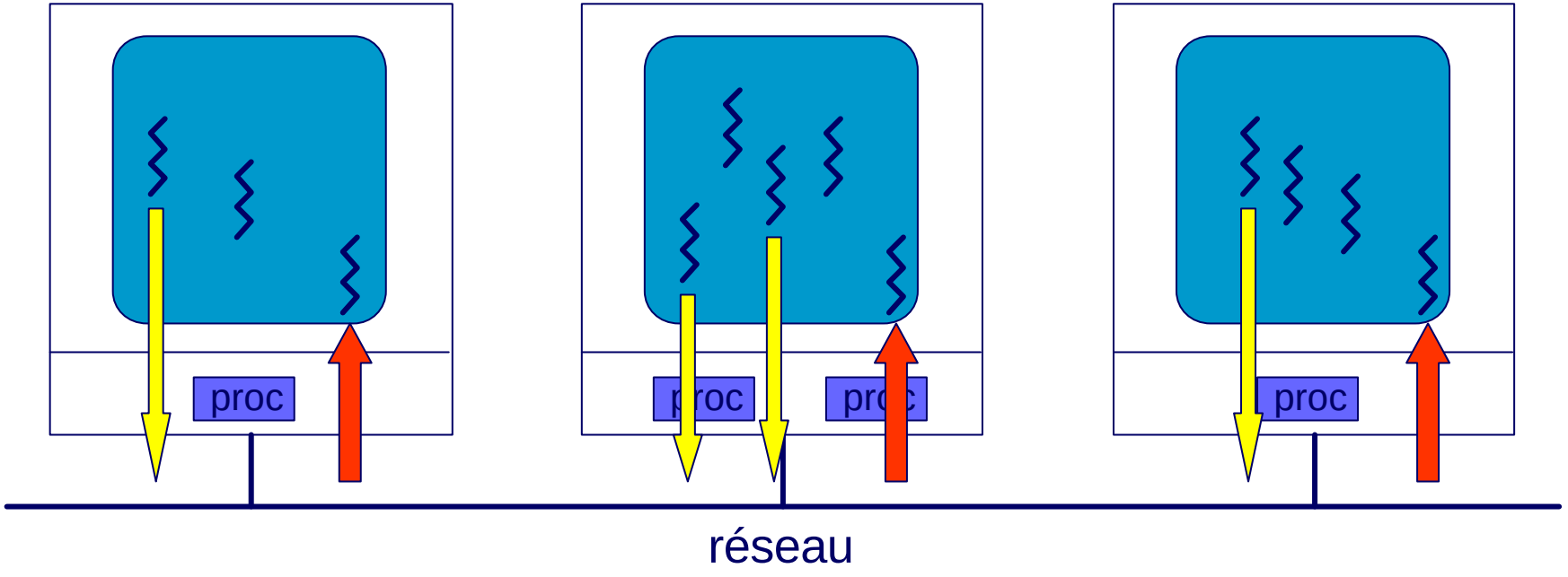


Intégration des threads et des communications

Réactivité des nœuds aux E/S

Progression des communications

- Problème
  - Comment assurer la progression des communications ?



Scrutation et interruptions

- La scrutation est nécessaire
  - API réseau ne fournissant pas d'appels bloquants
  - OS ne fournissant pas "d'activations"
  - Problème
    - Fréquence difficile à assurer
    - Coûteux en présence de multiple "*pollers* "
- Les interruptions sont nécessaires
  - Réactivité
  - Problème
    - Outils de synchronisation "interrupt safe" ?

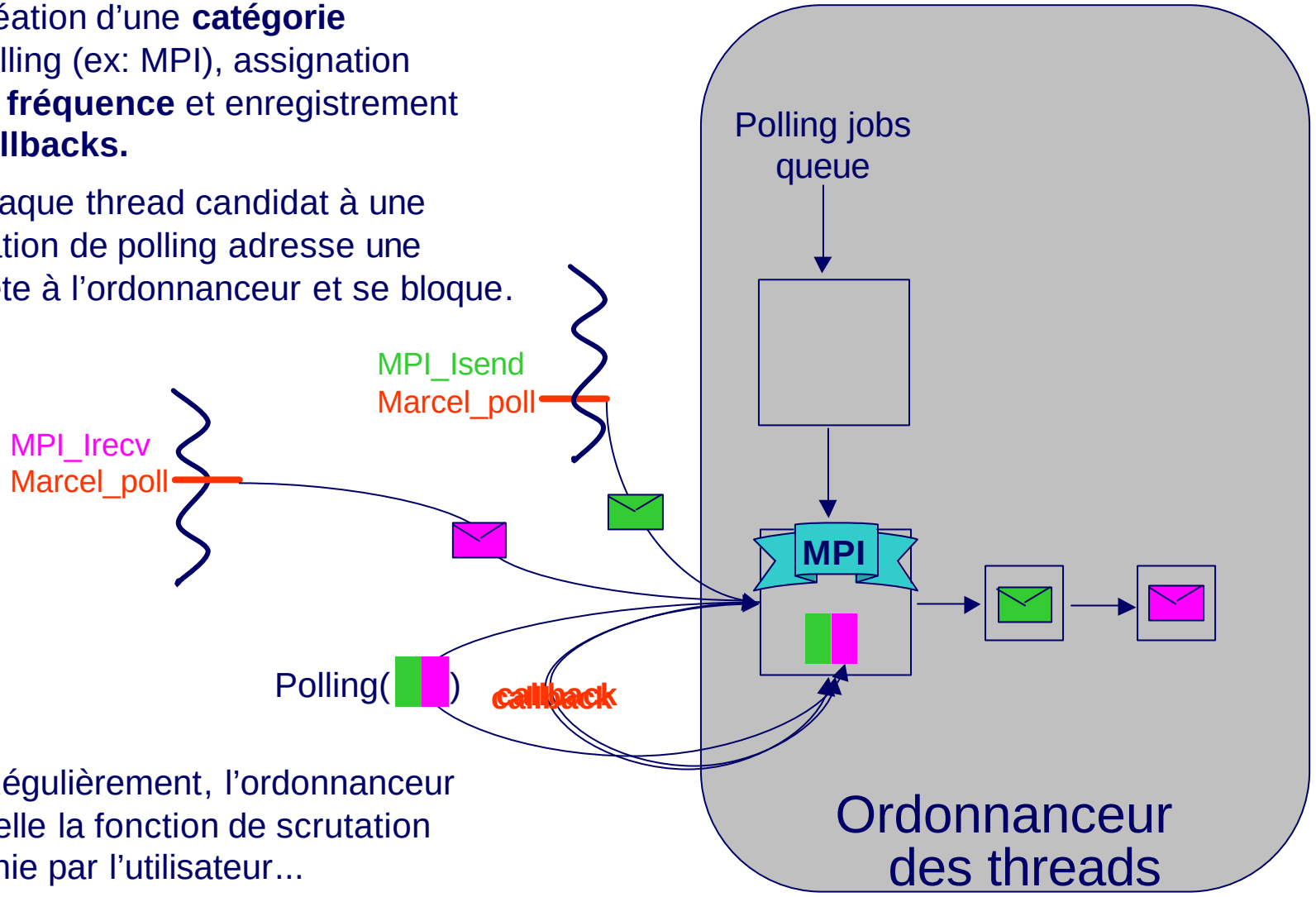
Support de l'ordonnanceur

- Ordonnanceur = serveur de scrutation
  - Choix de la méthode d'accès (scrutation/intr.)
- Support pour la scrutation
  - Fréquence contrôlée
  - Factorisation des scrutations multiples
- Support pour les interruptions
  - Utilisation possible des activations
  - Verrous spécifiques « interrupt-safe »

Scrutation par l'ordonnanceur

❶ Création d'une **catégorie** de polling (ex: MPI), assignation d'une **fréquence** et enregistrement de **callbacks.**

❷ Chaque thread candidat à une opération de polling adresse une requête à l'ordonnanceur et se bloque.



❸ Régulièrement, l'ordonnanceur appelle la fonction de scrutation définie par l'utilisateur…