Processus concurrents et parallélisme

Chapitre 7 - Systèmes multiprocesseurs et répartis

Gabriel Girard

15 mars 2023

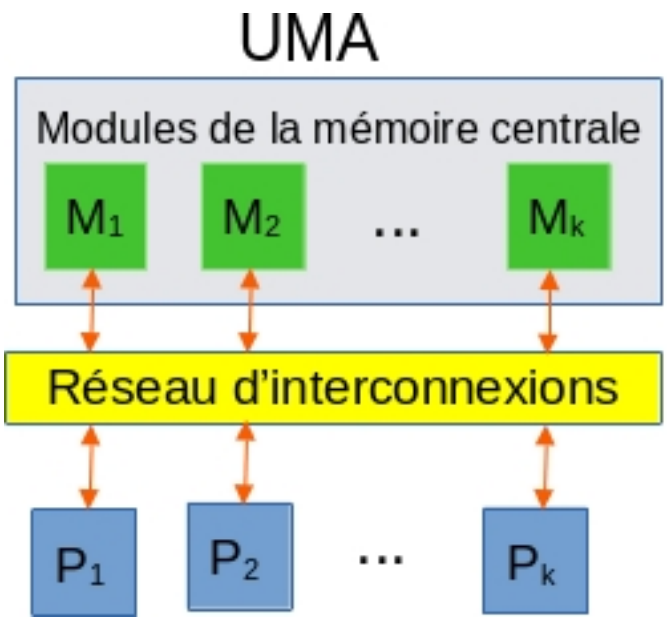
Chapitre 7 - Systèmes multiprocesseurs et répartis

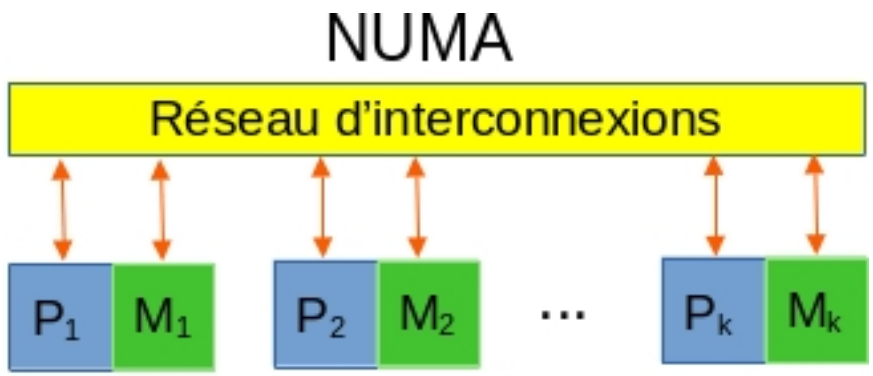
Chapitre 7 - Systèmes multiprocesseurs et répartis

Classifications

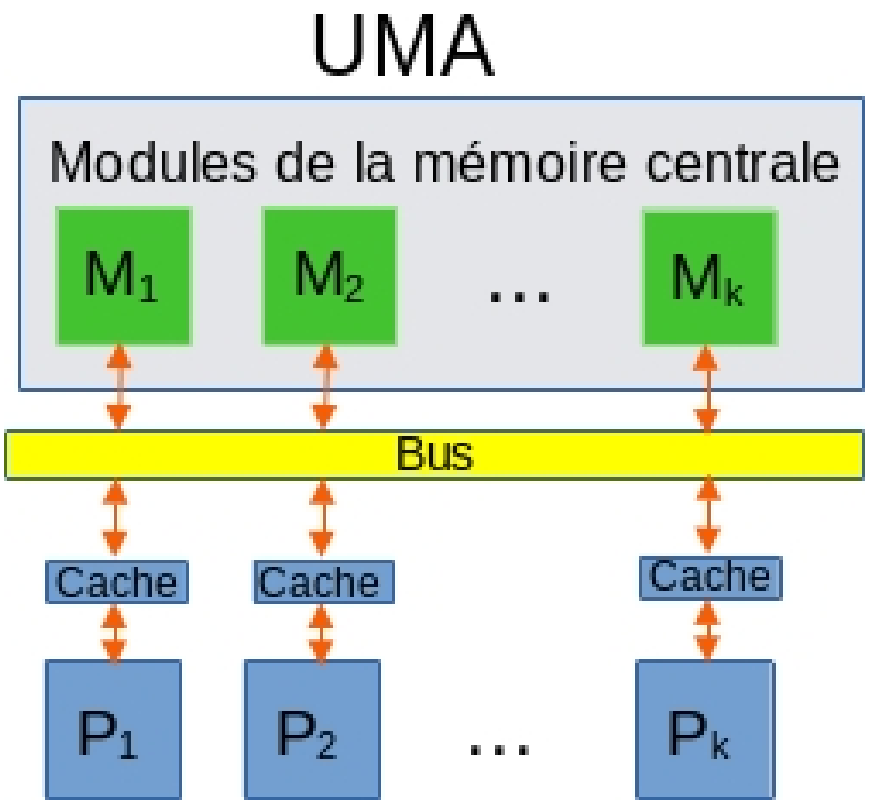
- Systèmes fortement couplés (mémoire commune)
- Systèmes faiblement couplés (réseau)
- Systèmes avec des processeurs fonctionnellement spécialisés

Classifications - UMA vs NUMA

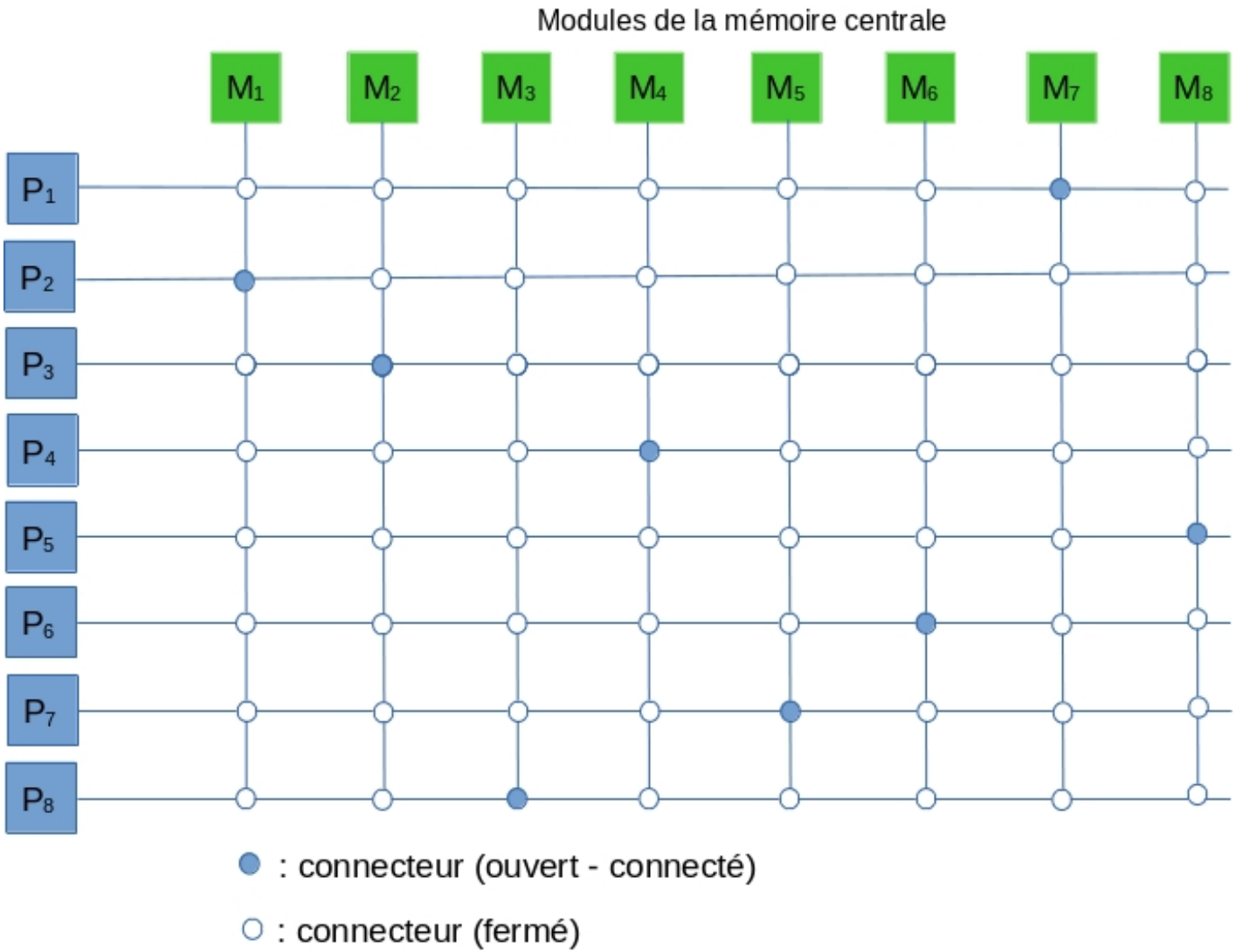




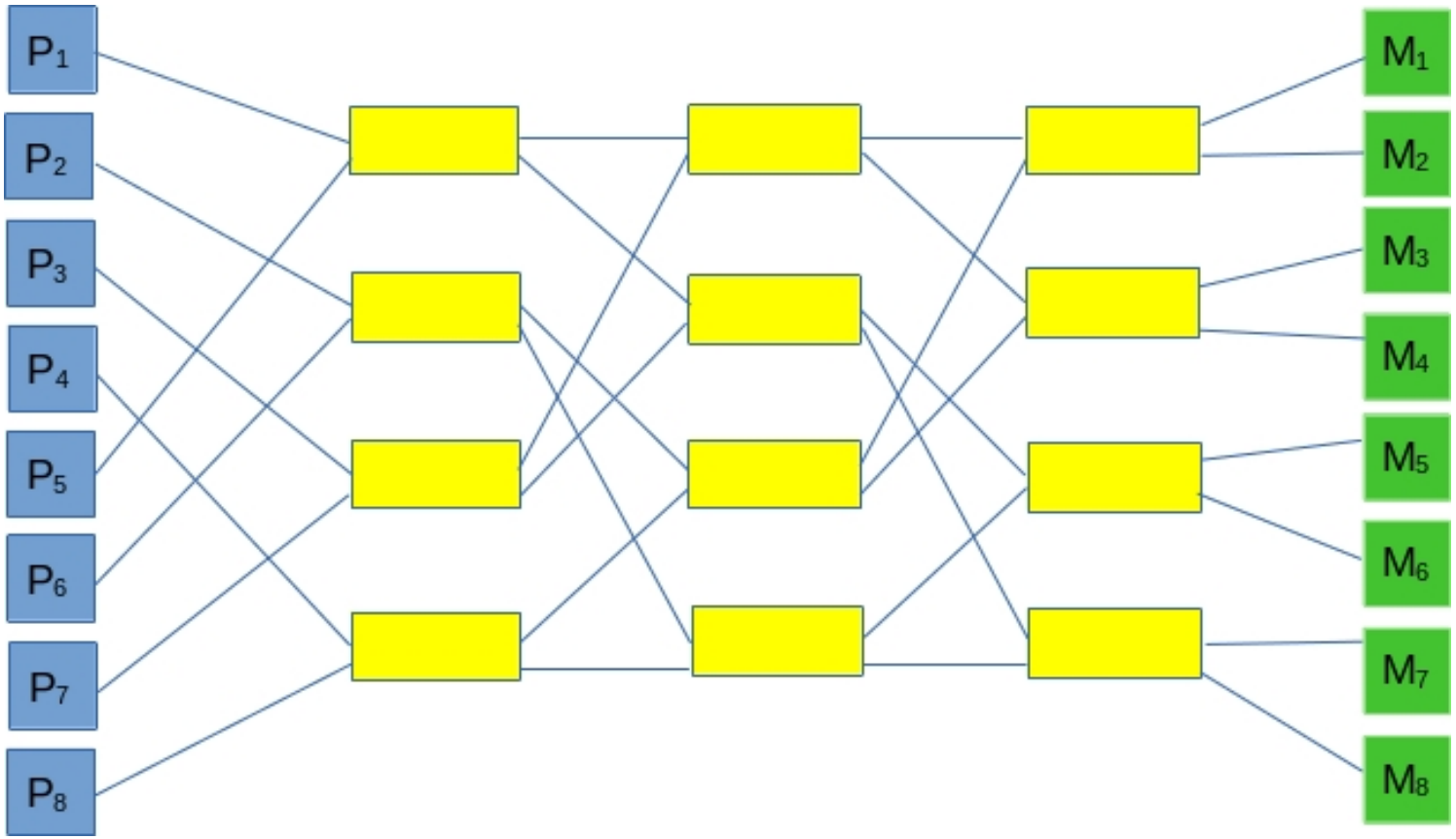
UMA


UMA
Modules de la mémoire centrale
M1
M2
...
Mk
Bus
Cache
Cache
Cache
P1
P2
...
Pk

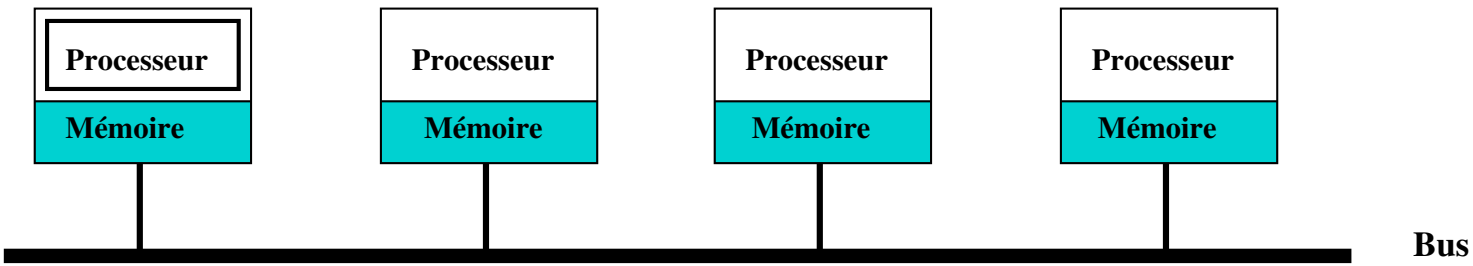
UMA


Modules de la mémoire centrale
M1
M2
M3
M4
M5
M6
M7
M8
P1
P2
P3
P4
P5
P6
P7
P8
: connecteur (ouvert - connecté)
: connecteur (fermé)

UMA



NUMA


Processeur
Mémoire
Processeur
Mémoire
Processeur
Mémoire
Processeur
Mémoire
Bus

Systèmes d'exploitation multi-processeurs

- Trois organisations possibles :
  - systèmes indépendants
  - systèmes asymétriques (administrateur/Travailleurs)
  - systèmes symétriques

Synchronisation de systèmes multiprocesseurs

- Instructions spéciales
- Problèmes
  - verrouillage du bus $\rightarrow$ performance ?
  - Attente active vs changement de contexte

Systèmes d'exploitation multi-ordinateurs

- Deux types de systèmes peuvent opérer sur un réseau :
  - systèmes d'exploitation réseaux
  - systèmes d'exploitation répartis

Systèmes d'exploitation réseau

- Système d'exploitation gérant de façon non transparente (généralement) un réseau.
  - rlogin, telnet, ftp, ssh, sftp, ...
  - NFS, Samba, ...

Systèmes d'exploitation répartis (DOS)

- Système d'exploitation créé pour fonctionner avec les réseaux et qui les gère de façon transparente si cela est requis (localisation, migration, réplication, concurrence)
- Ces systèmes sont souvent homogènes

Systèmes d'exploitation répartis (DOS)

- Problèmes ou attributs reliés au DOS
  - performance
  - capacité d'expansion (scalability)
  - migration
  - synchronisation (processus, horloge, ...)
  - réplication
  - fichiers
  - fiabilité et tolérance aux fautes
  - sécurité
  - transparence (localisation, accès, panne, réplication, persistance, migration, relocalisation et transaction)

Concepts utiles

- Processus
- Fil d'exécution (thread)
- Clients/serveurs + RPC
- Messages
- Logiciels médiateurs (middleware)

Fonctions

- Reprenons les fonctions d'un système d'exploitation centralisé pour voir comment elles peuvent être distribuées :
  - gestion de périphériques
  - gestion de fichiers
  - gestion de processus
  - gestion de la mémoire
  - ...

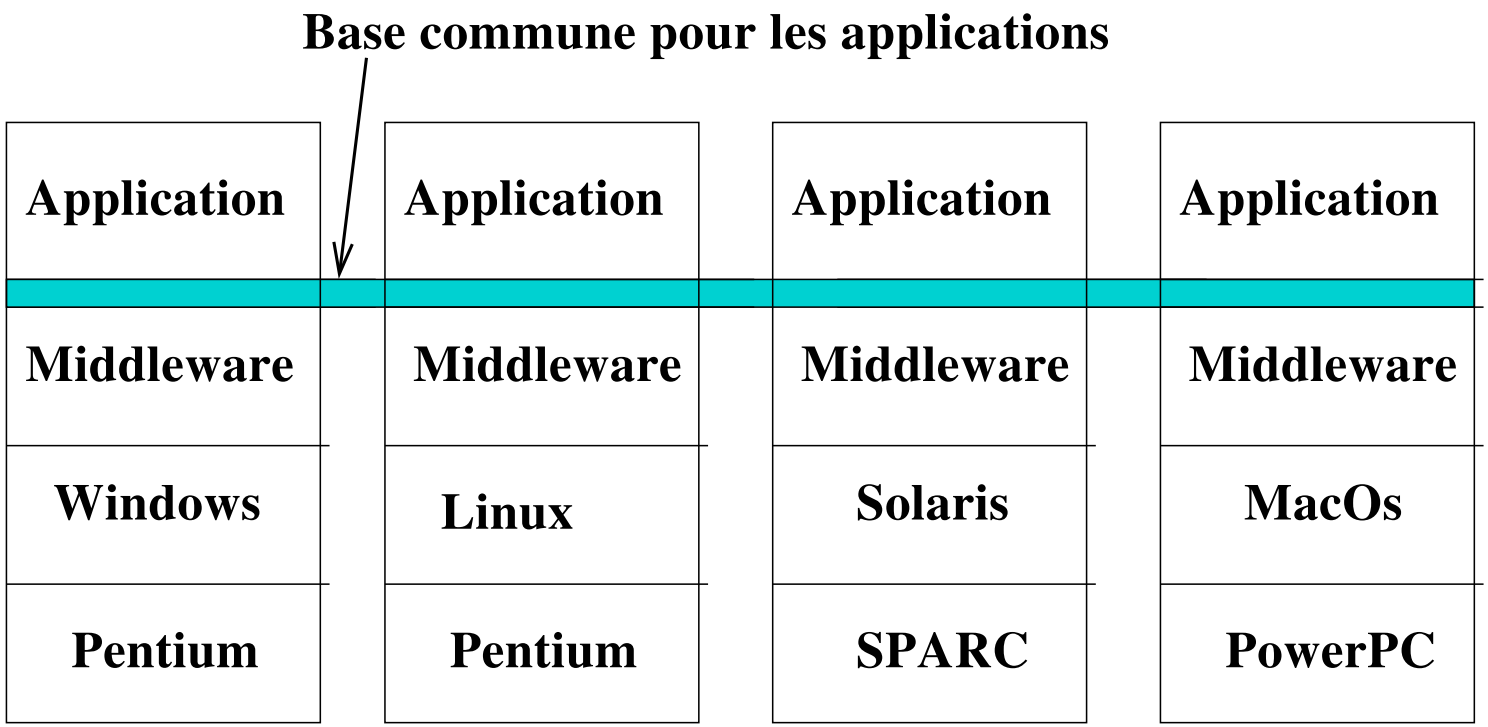
Exemples de systèmes et d'environnements

- NOW
- GRID
- Cluster
- P2P
- Internet
- Informatique en nuage (Cloud)

Middleware

- Il n'existe pas de vrais DOS
- On utilise un compromis
  - un système d'exploitation réseau
  - un Middleware
- Raisons de cette approche ?
- Cette approche offre une combinaison acceptable....

Middleware


Base commune pour les applications
Application
Application
Application
Application
Middleware
Middleware
Middleware
Middleware
Windows
Linux
Solaris
MacOs
Pentium
Pentium
SPARC
PowerPC

Rôle du Middleware

- Le rôle du Middleware est de fournir :
  - portabilité
  - transparence
  - inter-opérabilité
- Exemples de Middleware...

Exemples de Middleware

- WEB (basé sur les documents)
- Corba (basé sur les objets)
- Samba, NFS (basé sur les fichiers)
- Linda, Jini (basé sur la coordination)
- MPI, PVM
- Hadoop, OpenStack

Chapitre 7 - Systèmes multiprocesseurs et répartis

Transferts de fichiers

- La première approche consistait à transférer ≪manuellement≫ les fichiers
- FTP, UUCP, FTAM

Systèmes de fichiers répartis

- Plusieurs aspects ressemblent aux systèmes centralisés
- Définition :
  - Service de fichiers : spécification des services offerts par le serveur au client (interface, API)
  - Serveur de fichiers : processus qui fournit les services (ou une partie)
    On peut avoir plusieurs serveurs. Leur nombre et leur localisation doivent être transparents.
    Chaque serveur peut fournir des services différents.
  - Deux concepts : fichiers et répertoires

Interface

- Qu'est-ce qu'un fichier ?.
- Attributs d'un fichier ?
  - normaux +
  - immuable (Amoeba)
  - append (Google)
- Protection : pouvoirs ou listes d'accès

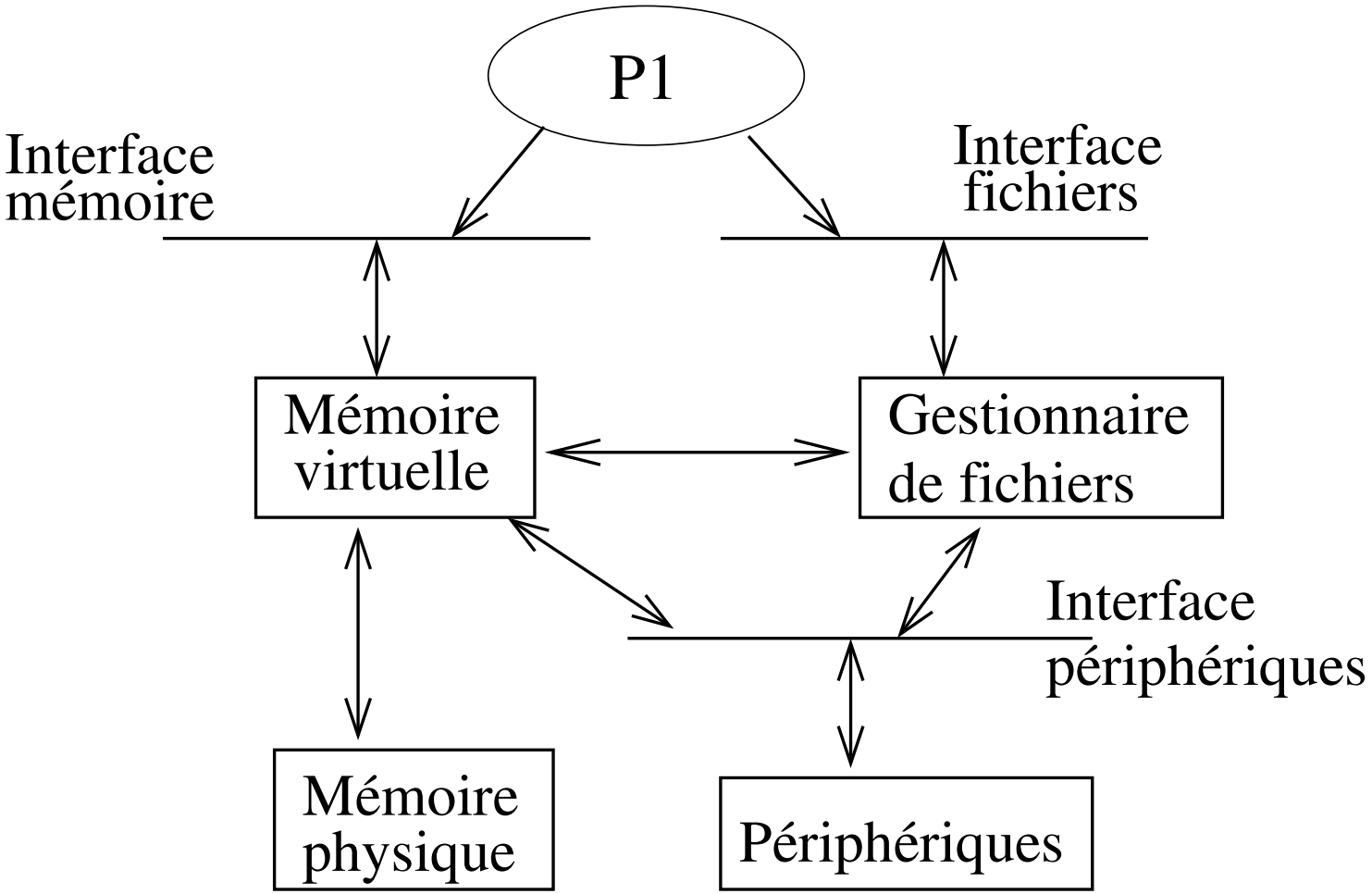
Interface

- Les opérations doivent être les mêmes que sur un système centralisé
- open(), close(), read(), write(), seek(), ...
- Exemple :
  - local : open("fichier1")
  - distant non transparent : open("132.210.40.88 :/usr/local/foo/fichier1")
  - distant transparent : open("/home/public/cours/ift630") ;

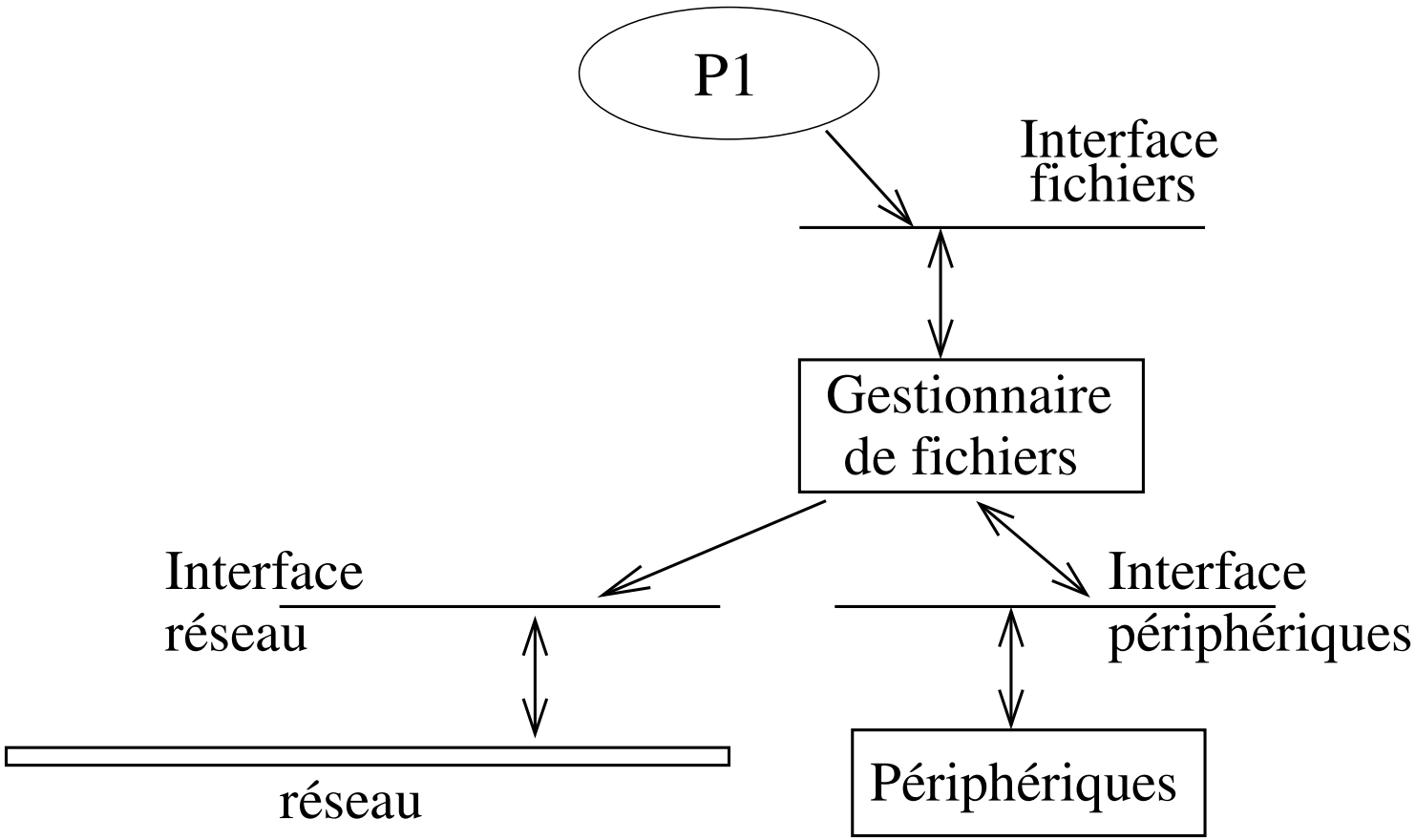
Architecture

- Il y a trois architectures principales pour les DOS :
  - les disques éloignés (remote disk)
    $\longrightarrow$ le système de fichiers est local
  - le serveur de fichiers éloignés (accès éloigné)
  - le serveur de fichiers éloignés (upload/download)

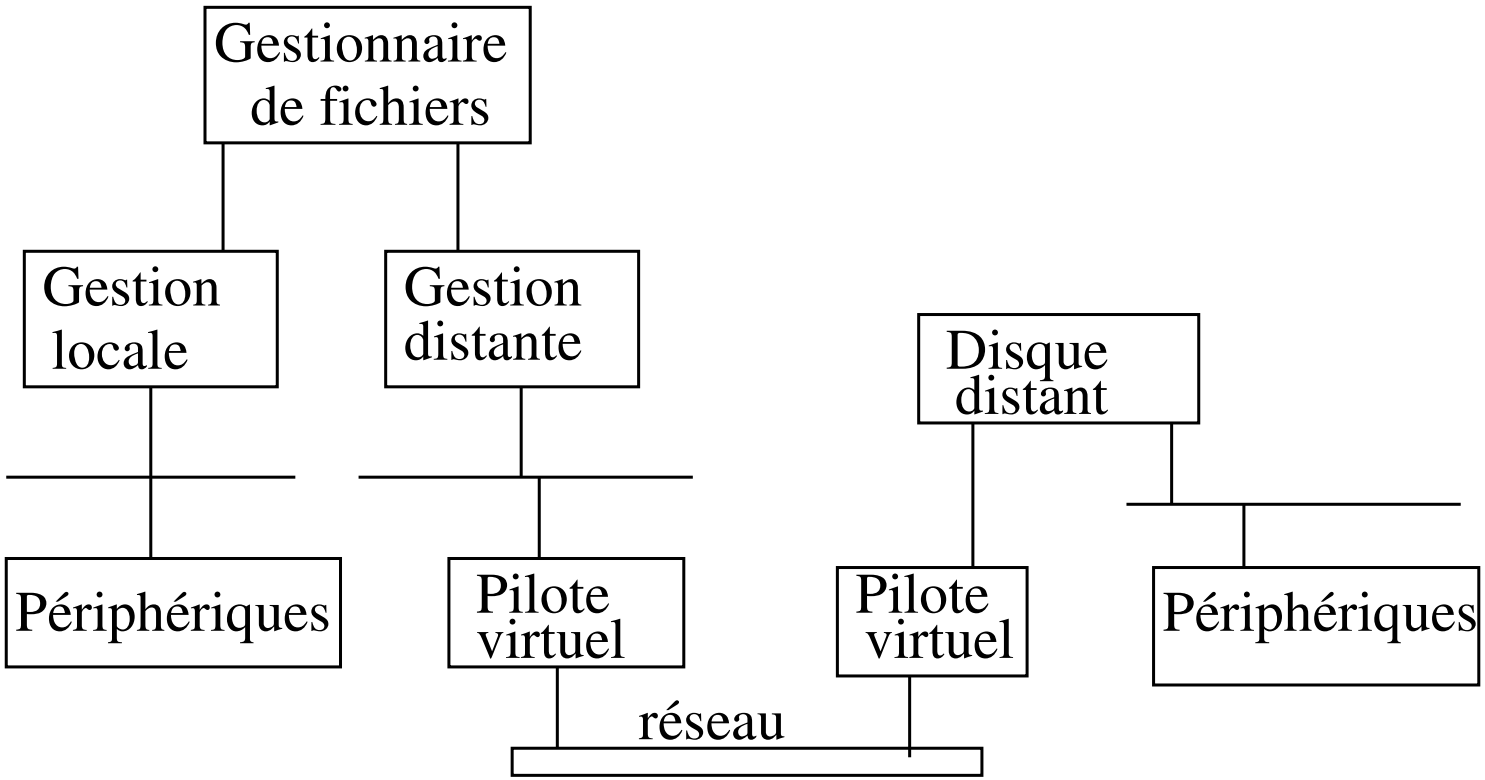
Architecture (Système de fichiers centralisé)



Architecture (Système de fichiers réparti)



Architecture (Disque distant)



Disque distant

- Utilisé principalement avec des stations de travail sans disque (diskless)
- Un serveur fournit l'espace disque pour la station de travail
- N'est plus très courant aujourd'hui

Disque distant

- Le client est responsable de presque tout le système de fichiers
- Le client contient :
  - le système de fichiers et les algorithmes
  - un pilote de disque virtuel qui fournit le même API qu'un pilote local
- S'il n'y a pas de disque local, tous les accès aux fichiers passe par le disque éloigné
- S'il y a une disque local, le système de fichiers est chargé de déterminer si le fichier est emmagasiné sur un disque local ou éloigné

Disque distant

- Le serveur est responsable des accès aux disque distant
- Le serveur contient contient :
  - l'application disque distant
  - le pilote et le disque
- l'application reçoit l'opération sur le disque distant et l'exécute sur le disque local
- Le résultat est retourné par l'application disque distant vers le pilote virtuel

Disque distant

- Le client maintient toutes les informations sur les fichiers ouverts
- Lors de l'exécution d'un programme, celui-ci est chargé du disque distant
- La mémoire virtuelle peut se servir du disque distant pour la pagination et l'échange

Disque distant

- Simple
- Performance ? ? ? (réseau + optimisation)
- Fiabilité ? ? ?
  - Ré-exécution des commandes lors de pannes ou de pertes (lecture, écriture, seek, état, ...)
    Comme les opérations de lecture et d'écriture sont idempotentes, il n'y a pas trop de problèmes
  - Reprise après un panne du disque (ou du serveur)
    Simple car le serveur est «stateless»

Serveur distant

- Une partie du serveur de fichier est locale et une autre distante
- Les fonctions de chacun des parties dépend du modèle (upload/download ou accès à distance)

Upload/download

- Aussi appelé «file level caching»
- Lecture : transfert le fichier vers le client (en mémoire ou sur disque)
- Écriture : transfert le fichier vers le serveur
- Problèmes de cohérence (cohérence de cache ou version)
- Avantages : simple et efficace (du côté du serveur)
- Inconvénients : coût en espace et en transfert

Accès éloigné

- Aussi appelé ≪block caching≫
- Le serveur reçoit et exécute les demandes
- Le serveur doit fournir plus d'opérations (ouverture, fermeture, lecture, écriture, déplacement, ...)
- Avantages : peu d'espace chez le client
- Inconvénients : serveur complexe, accès lent
- Combinaison des deux : cache

Accès éloigné

- Architecture ressemble au disque distant
- Le serveur éloigné implante plus de fonctionnalités
- Le serveur peut conserver des informations sur les fichiers ouverts (réduction du trafic sur le réseau)
- Cela peut améliorer la performance mais le serveur et le client doivent maintenir de l'information sur les fichiers ouverts (duplication et problème de cohérence)

Service de répertoire

- Identification à deux niveaux (chaque fichier possède deux noms)
  - nom symbolique (pour les usagers)
  - nom binaire (pour le système)
- Les répertoires fournissent une correspondance entre les deux

Serveur de répertoire

- Fait la traduction nom symbolique $\rightarrow$ nom binaire
- Ce service fournit :
  - un alphabet et une syntaxe pour les noms symboliques
  - une hiérarchie de répertoire (arbre, graphe)
  - des liens symboliques ou physiques

Répertoire - Organisation des noms symboliques

- A-t-on la même vue de la hiérarchie ?
  - même vue : un chemin est valide sur tous les sites
  - différente vue : comportement peut différer d'un site à l'autre (mount, facile à implanter)
- Existe-t-il une racine globale ?

Répertoire - Gestion des noms symboliques

- Caractéristiques importantes des noms :
  - transparence de localisation
  - indépendance de localisation
- Approches
  - machine + chemin : /machine/path...
  - monter dans la hiérarchie locale
  - un espace de nom unique pour toutes les machines

Répertoire - Noms binaires

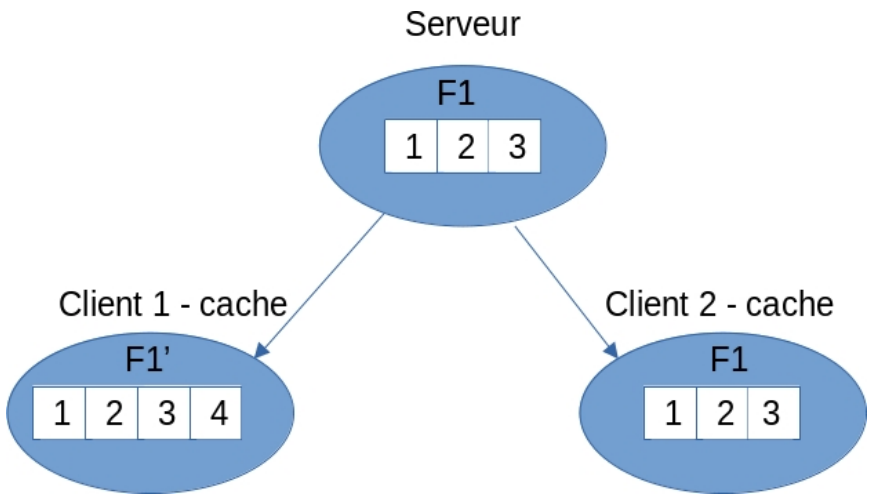
- Lorsqu'un fichier est ouvert, le système recherche le nom binaire
- Le nom binaire varie d'un système à l'autre
  - Local : I-node ou autre type de pointeurs
  - Éloigné : nom du serveur + nom local
  - Un pouvoir peut servir de nom binaire
- Un nom symbolique peut avoir plusieurs noms binaires
- Traduction peut se faire par le serveur local ou par diffusion

Sémantique de partage

- Sur un système central :
  - La lecture retourne la valeur de la dernière écriture s'il y en a plusieurs successives (total ordering ou absolute time ordering)
- Sur un système répartis :
  - possible si un seul serveur ($\neq$ absolute time ordering)
    Un seul serveur n'est pas efficace sur un système réparti.
    Il doit traiter toutes les demandes et devient vite un goulot d'étranglement.

Performance

- Pour obtenir une meilleure performance : le client maintient une cache
- Problème de cohérence entre les caches



Cohérence de cache

- Propagation immédiate (peu efficace)
- Sémantique de session (modifications simultanées et position dans le fichier)
- Changer la sémantique et rendre les fichiers immuables
- Utilisation des transaction atomiques

Implantation

- Remarques importantes :
  - la plupart des fichier sont petits
  - la plupart des fichiers ont une courte durée de vie
  - peu de fichiers sont partagés
  - différents types de fichiers impliquent différents traitements

Structure

- Clients/serveurs différents ou identiques ? ?
  Ex. : NFS et Amoeba
- Comment organiser fichiers et répertoires ?
  un serveur, deux serveurs distincts, ...
- Comment rechercher un nom s'il y a plusieurs serveurs ?
  Usager, serveurs ou diffusion.
- Stateless (NFS) ou statefull (AFS et RFS)

Structure

- Comment organiser la cache ? ?
  - Où la mettre ? Disque ou mémoire, client ou serveur ? Mémoire du client, mémoire du système, du serveur ou de l'usager ?
  - Que faire si elle est pleine ?
  - Comment maintenir la cohérence ?
    rien, écriture immédiate, écriture avec délai, sémantique de session, contrôle centralisé, invalidation de la cache, élimination de la cache lors d'écriture, ...

Structure

- Fait-on de la réplication ?
  - Pourquoi la réplication ?
  - Transparente ou non ?
  - Type de réplication : par l'usager, lazy replication, group communication, ...
  - Mise à jour ? ?
    un message à chaque copie, copie primaire, vote, ...

Fiabilité

- Si la commande distante semble perdue (time out).
- Trois cas :
  - la commande ne s'est pas rendu
  - le résultat s'est perdu
  - la réponse est retardée
- Problème : on peut refaire plusieurs fois la même commande

Fiabilité

- Pour éviter les répétitions les commandes doivent être ≪idempotentes≫
  - la lecture est idempotente (le client doit tenir compte des multiples réponses)
  - l'écriture est idempotente
  - le seek n'est pas idempotent
- On peut utiliser la mémoire stable (stable storage)

Exemples d'implantation

- NFS (Unix, Sun)
- AFS (Andrew File System), OpenAFS (IBM)
- NCP (Novell)
- Google FS
- HDFS (Hadoop)
- Swift (OpenStack)
- S3 (Amazon)
- Azure (Microsoft)
- Samba
- Locus, Coda, ...

Chapitre 7 - Systèmes multiprocesseurs et répartis

Quoi planifier et comment ?

- Processus ou fils ?
- Une vs deux dimensions :
  - une dimension : on prend le suivant.
  - deux dimensions : choix du processeur.

Caractéristiques des algorithmes

- Assignation statique ou dynamique.
- Avec ou sans multiprogrammation.
- Existance ou non de l'algorithme de planififcation court-terme.

Algorithme : Partage de charge

- Liste de processus prêts globales
- FCFS, plus petit nbr de fils en premier, avec ou sans requisition
- Problèmes :
  - verrous $\rightarrow$ planification intelligente
  - cache et TLB $\rightarrow$ planification par affinité

Algorithme : Partage d'espace

- Situation ou un processus possède des fils qui interagissent
- Assignation d'UCTs dédiés
- Cas :
  - chaque pcs a ses UCTs dédiés selon son nombre de fils (FCFS)
  - chaque pcs peut varier ou non son nbr de fils
- Problème d'allocation de processeurs ressemble à celui de l'allocation de la mémoire

Algorithme : Planification de groupe

- Tous les UCT sont planifiées de façon synchrone
- Soit N uct et M processus avec k fils ($k < N$)
- Algorithmes :
  - Chaque pcs reçoit $1/N$ du temps
  - Chaque pcs reçoit une tranche proportionnelle au nbr de fils
  - Exécution de plusieurs pcs en même temps

Algorithme : Planification dynamique

- Allocation d'un certain nombre d'UCT aux processus
- Les pcs planifient leurs fils
- Système lors d'une demande d'UCT
  - Si UCT inactif $\rightarrow$ allocation
  - Si nouveau pcs $\rightarrow$ allocation au moins 1 UCT
  - Si demande non répondu $\rightarrow$ conserve demande
- Système lors de libération
  - allocation 1 UCT à chaque pcs qui n'en a pas
  - allocation du reste en FCFS

Gestion de l'UCT : Systèmes répartis

- On gère encore plusieurs UCT mais sans mémoire commune
- Quelques algorithmes peuvent s'adapter
- Plusieurs questions ?
  - Où va-t-on exécuter le processus ?
  - Migration statique ou dynamique ?
  - Comment identifier une processus de façon unique ?

Gestion de l'UCT

- Rappel sur les processus
  - état + informations
  - opérations
- planification
- synchronisation
- interblocage

Gestion de l'UCT

- Les primitives les plus importantes : création/destruction
- La création exige :
  - allocation d'un espace d'adresses
  - allocation d'un descripteur (PCB)
  - démarrage du processus

Planification répartie ?

- Utilité ? ? ? ?
  - minimiser le temps de calcul
  - meilleure utilisation des ressources
  - balancer la charge de travail

Planification implicite ou explicite ?

- Explicite : exige une interface de programmation...
- Implicite : transparent pour l'usager...

Où exécuter le processus ?

- Le but est de balancer la charge de travail et d'obtenir un temps de virement minimal
- Approche statique
  - le site d'exécution est déterminé à la création
  - s'exécute en permanence sur ce site
- Approche dynamique
  - on transfert le processus pendant son exécution
  - algorithme complexe
  - beaucoup de facteurs à considérer : temps de transfert, surcharge, communication inter-processus, utilisation de ressources, ...

Approche statique

- Il est important de bien choisir le site d'exécution (on ne peut pas changer d'idée..)
- Utilise information suivante :
  - besoin en UCT
  - besoin en mémoire
  - besoin en communication avec les autres processus
- Objectifs de l'algorithme
  - minimiser les pertes de cycle UCT (UCT qui ne fait rien)
  - minimiser les communications
  - assurer l'équité

Algorithme 1

- Algorithme déterministe basé sur la théorie des graphes
- Suppose que l'utilisation de la mémoire et de l'UCT est connue
- Suppose l'existence d'une matrice spécifiant le niveau de communication entre chaque processus
- Si le nombre de processus $>$ que le nombre d'UCTs, alors k processus seront associés à chaque UCT
- Cette association doit minimiser le trafic sur le réseau.

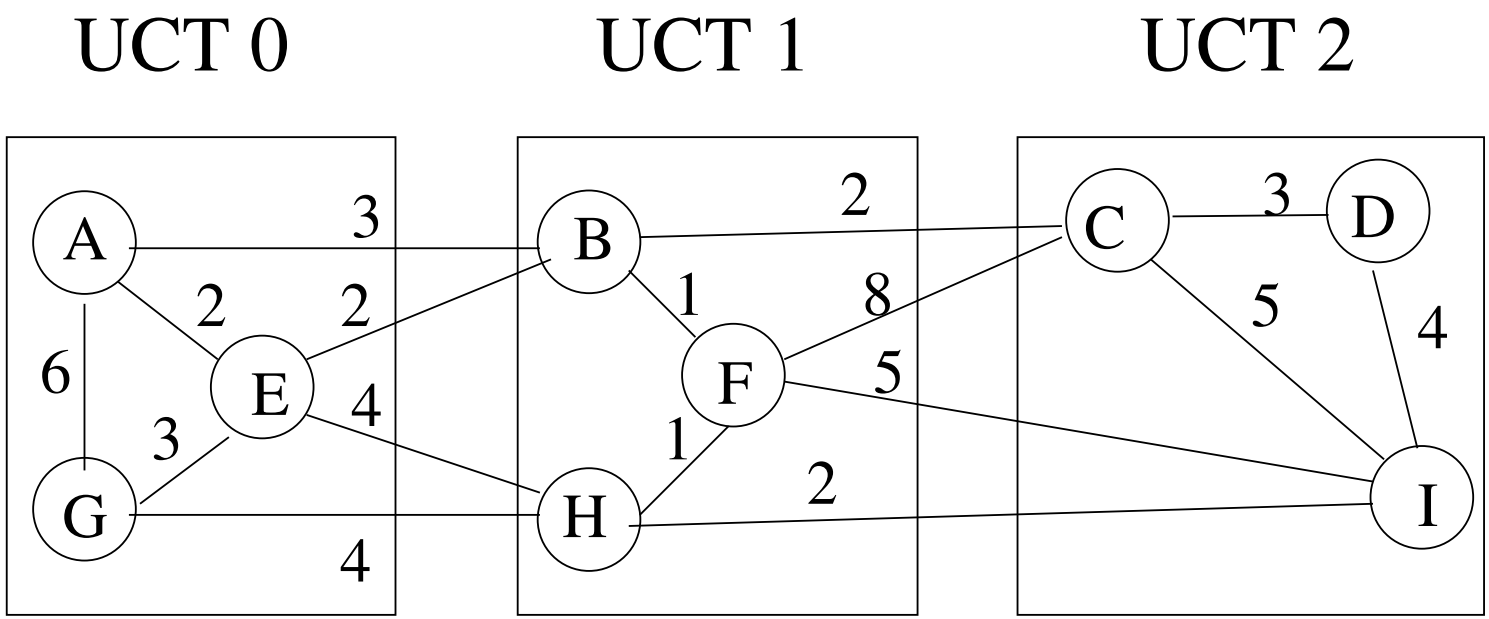
Algorithme 1

- Le système est représenté par un graphe
- Un noeud représente un processus
- Un arc représente le flux de messages entre deux processus
- Le problème se réduit à trouver une façon de partitionner le graphe en K sous-graphes selon certaines contraintes (temps UCT et utilisation mémoire $<$ limite)

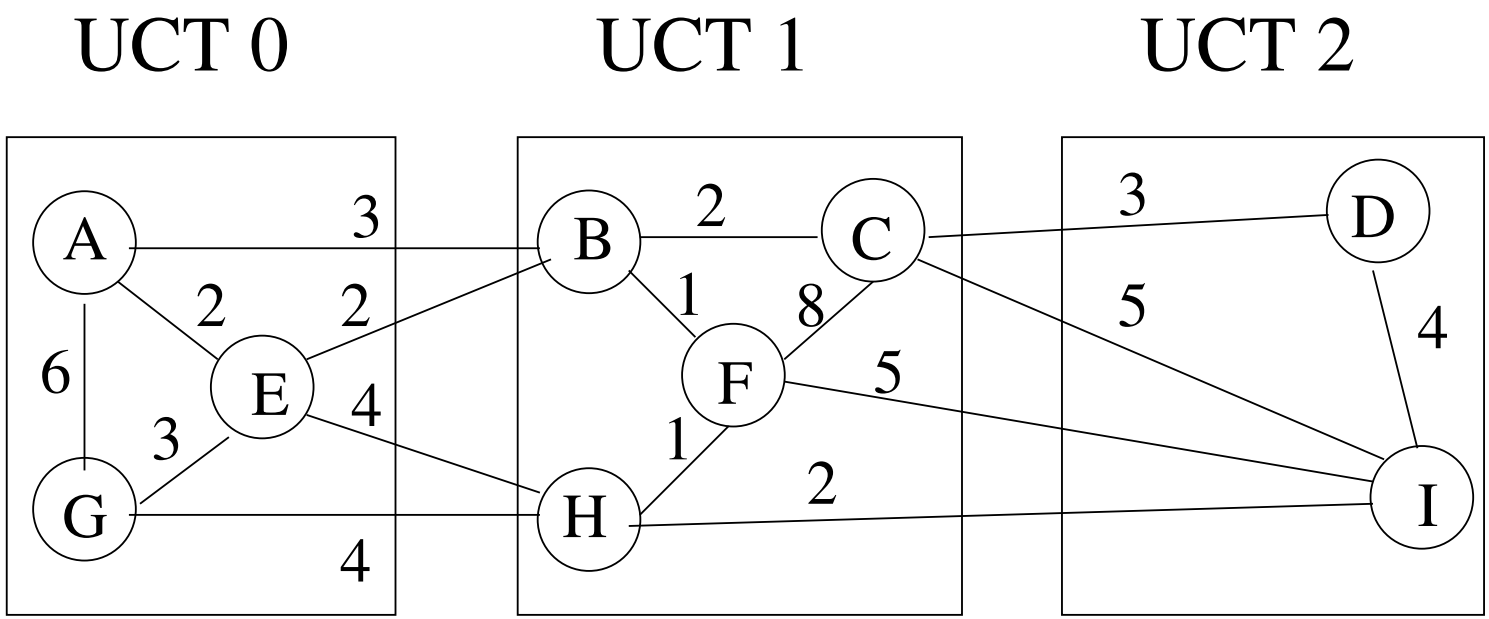
Algorithme 1

- Pour chaque solution respectant les contraintes :
  - les arcs internes à un sous-graphe représentent les communications intra-machine et peuvent être ignorées
  - les arcs allant d'un sous-graphe à l'autre représentent le trafic sur le réseau
- Le but est de trouver le partitionnement qui minimise le trafic tout en rencontrant les contraintes

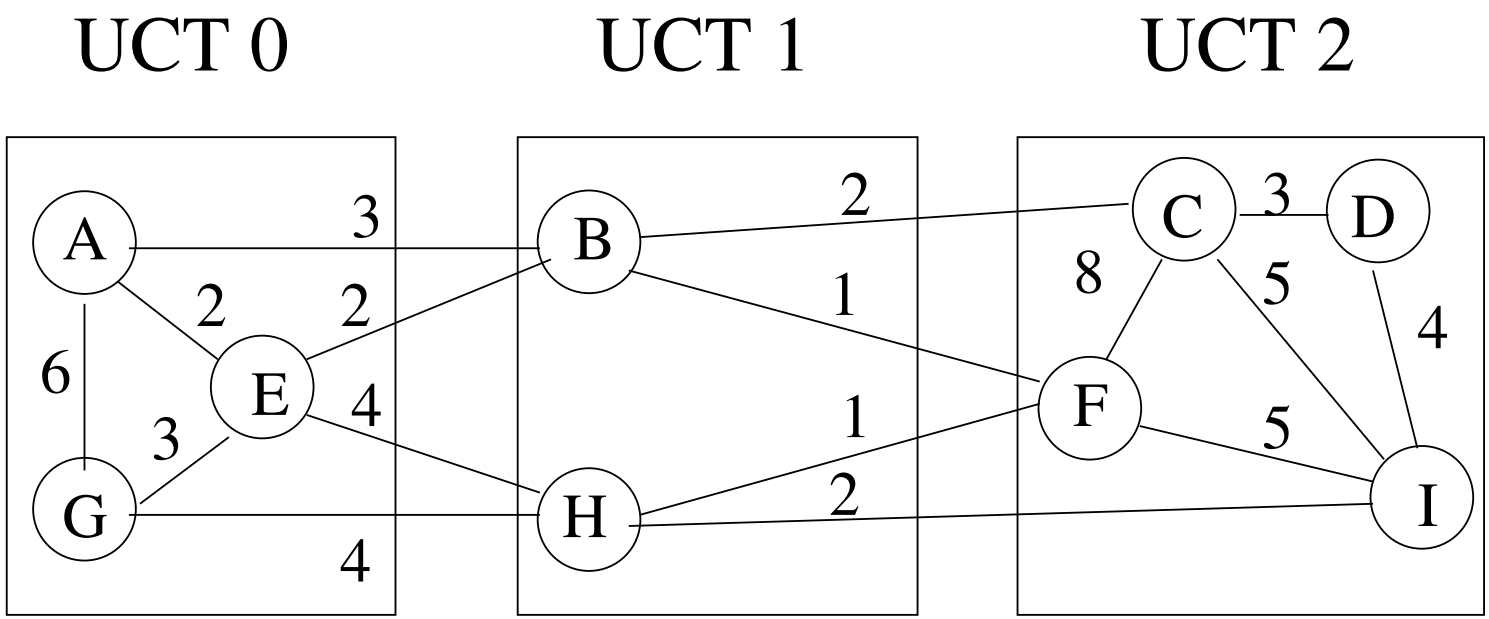
Algorithme 1



Algorithme 1



Algorithme 1



Algorithme 2

- Algorithme heuristique initié par l'expéditeur
- Lors de la création d'un processus, il s'exécute localement sauf si le site de création est surchargé
- Si le site est surchargé, un autre site est choisi au hasard et sa charge est demandée
- Si la charge du nouveau site est inférieure à une certaine limite, le processus est créé sur ce nouveau site
- S'il n'est pas sous la frontière, un second site est choisi au hasard et sa charge est demandée
- Si aucun site n'est choisi après N tentatives, le processus est exécuté localement

Algorithme 2

- Cet algorithme fonctionne bien en général...
- Cause beaucoup de trafic sous une lourde charge...

Algorithme 3

- Algorithme heuristique initié par le récepteur
- Lorsqu'un processus termine, le système vérifie la charge locale
- Si celle-ci est trop faible, il choisit un site au hasard et lui demande du travail
- S'il n'a trouvé aucun travail après N essais, il arrête temporairement
- Recommence lorsqu'un autre processus termine ou après un temps d'attente si inactif
- Avantage : moins de messages sous charge élevée
- Inconvénient : beaucoup de trafic sous charge légère

Algorithme 4

- Combinaison des deux....
- Autres optimisation (se rappeler les charges de certains sites, ...)

Algorithme 5

- Algorithme basé sur le concept d'appels d'offre
- Simule un mini système économique
- Les joueurs sont les processus qui doivent acheter du temps UCT
- Chaque site soumet des prix pour ses services
- Les prix sont des indicateurs sur la valeur du service (vitesse de la machine, quantité de mémoire, charge, temps réponse, autres ressources, ...)

Algorithme 5

- Lors du démarrage d'un processus :
  - recherche les sites fournissant le service requis
  - trouve les sites qu'il peut se payer
  - calcule le meilleur candidat (le moins coûteux)
  - génère une offre et la soumet au site choisi
- La machine choisie reçoit toutes les offres et en choisit une
- Le gagnant et les perdants sont informés
- Les prix sont ensuite mis à jour
- Problèmes : d'où vient l'argent virtuel ? ? ?

Autres approches

- Diffusion des états périodiques...
- Exemples : Slurm, Torque/Maui, Torque/Moab, PBS

Approche dynamique

- Peu d'algorithmes connus
- Les erreurs sont tolérables car un processus peut migrer

Gestion de l'identification

- Comment gérer les descripteurs ? ? ?
  - une seule copie sur le site de départ ?
  - une seule copie sur le site d'exécution ?
  - un copie complète sur chaque site ?
  - une copie partielle sur chaque site ?

Identification du processus ?

- Comment identifier une processus de façon unique ?
  - nom site + nom local
  - nom global

Exemples

- bidding, échange d'états
- Mach
- Amoeba
- PBS, Torque/Maui, Torque/Moab
- Sun Grid
- Condor, Boinc
- Mosix
- Slurm
- ...

Chapitre 7 - Systèmes multiprocesseurs et répartis

Gestion de la mémoire

- Concerne la mémoire centrale + le disque
- Gestion de la mémoire virtuelle avec swapping et pagination
- MMU, TLB, Contexte
- Pagination, segmentation
- Mémoire partagée (code, bibliothèques)
- Stations sans disque

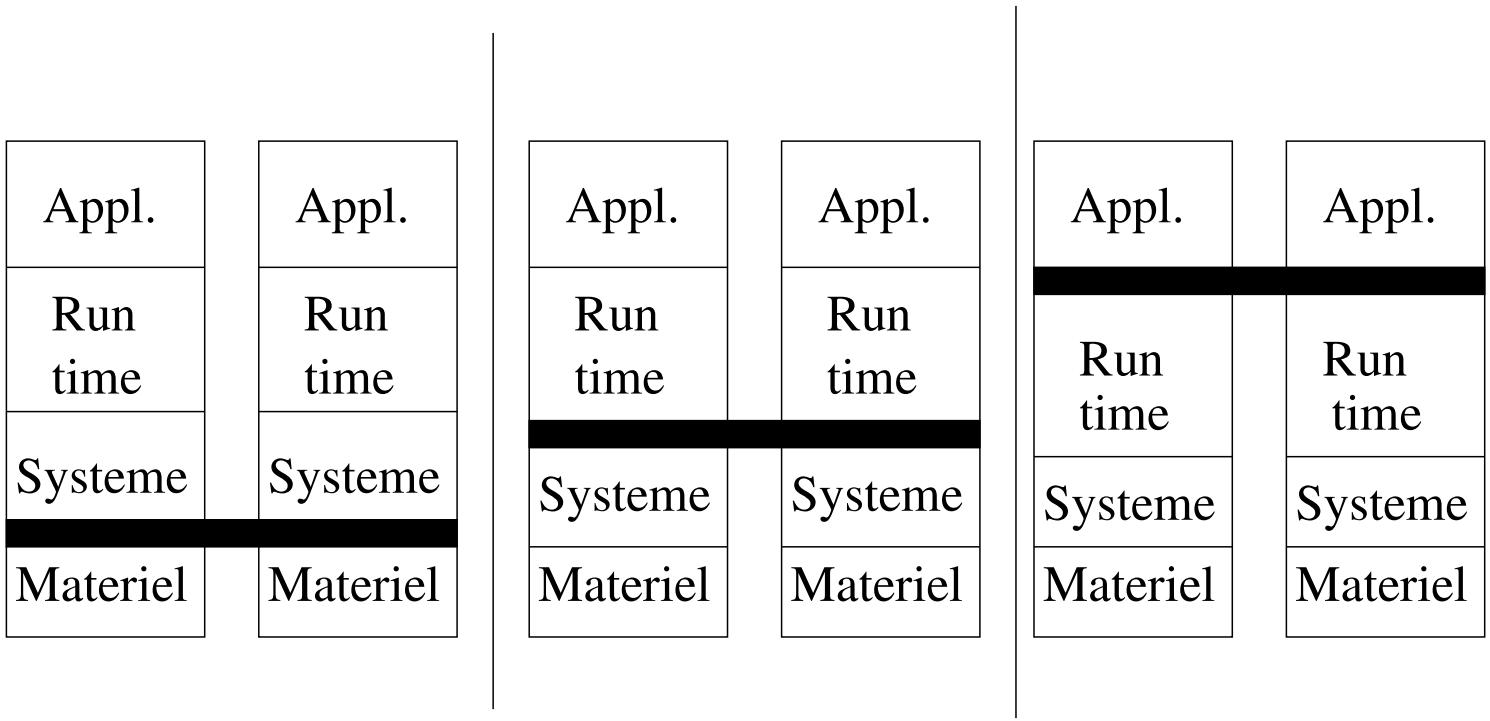
Gestion de la mémoire

- On partage la mémoire sur un multiprocesseur
- Le bus devient le goulot d'étranglement
- Utilise les caches
- UMA vs NUMA (vitesses d'accès différentes)

Gestion de la mémoire

- Sur un réseau, on utilise principalement les messages
- Toutefois certains développeurs préfèrent utiliser le concept de mémoire partagée
- On appelle ce concept DSM (Distributed Shared Memory)
- Il y a plusieurs approches

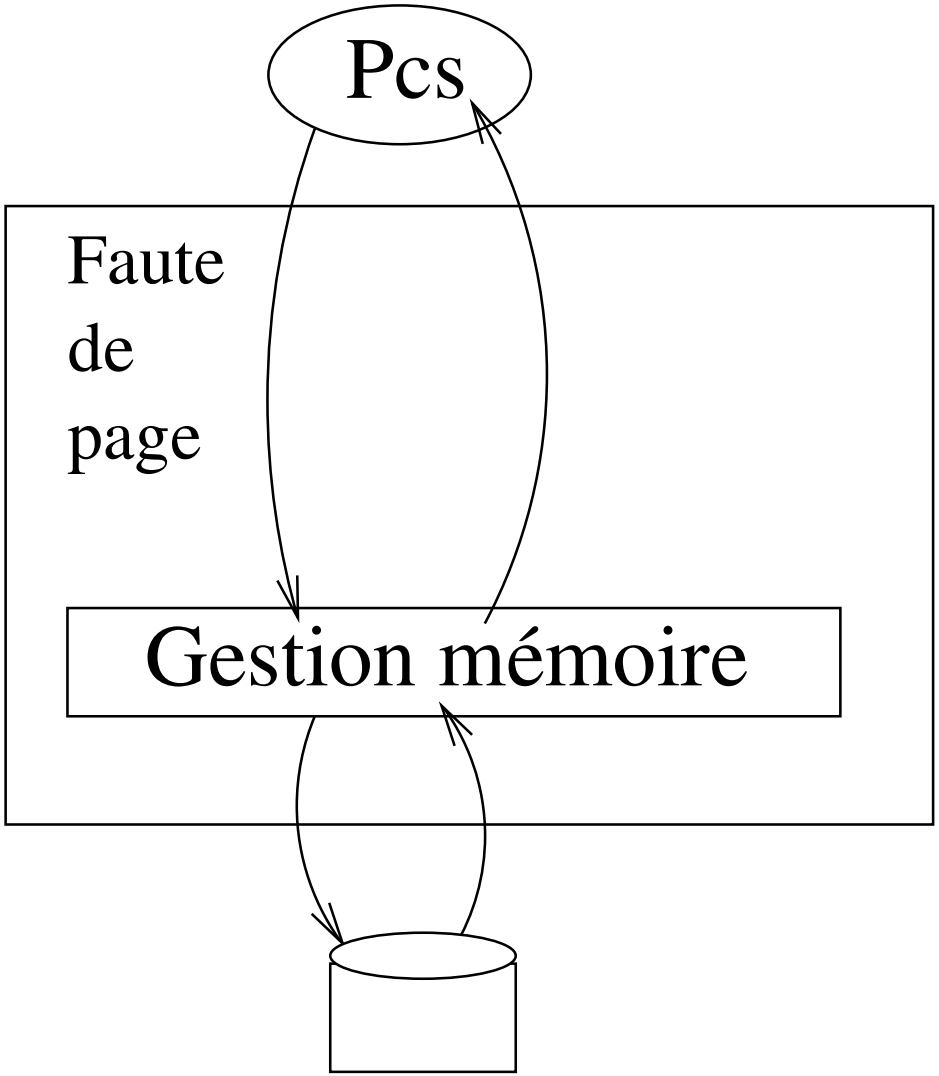
Gestion de la mémoire



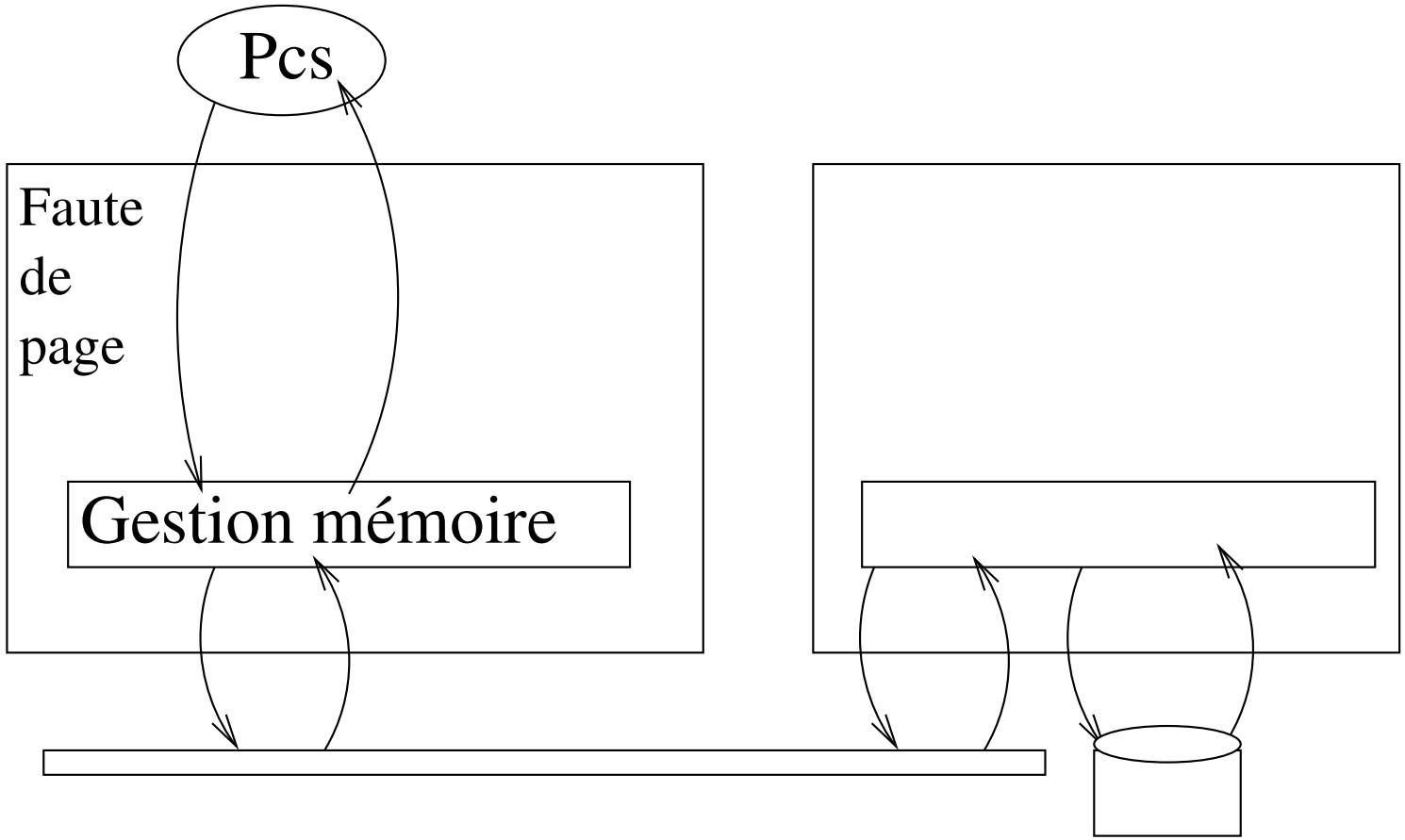
Niveau OS

- La principale approche consiste à partager les pages par l'intermédiaire de la mémoire virtuelle
- DSVM (Distributed Shared Virtual Memory)
- Rappelons que la mémoire virtuelle permet d'exécuter un programme partiellement chargé
- La mémoire virtuelle sépare le programme en pages (1K, 2K, 4K ou 8K)
- Les pages sont chargées au besoin

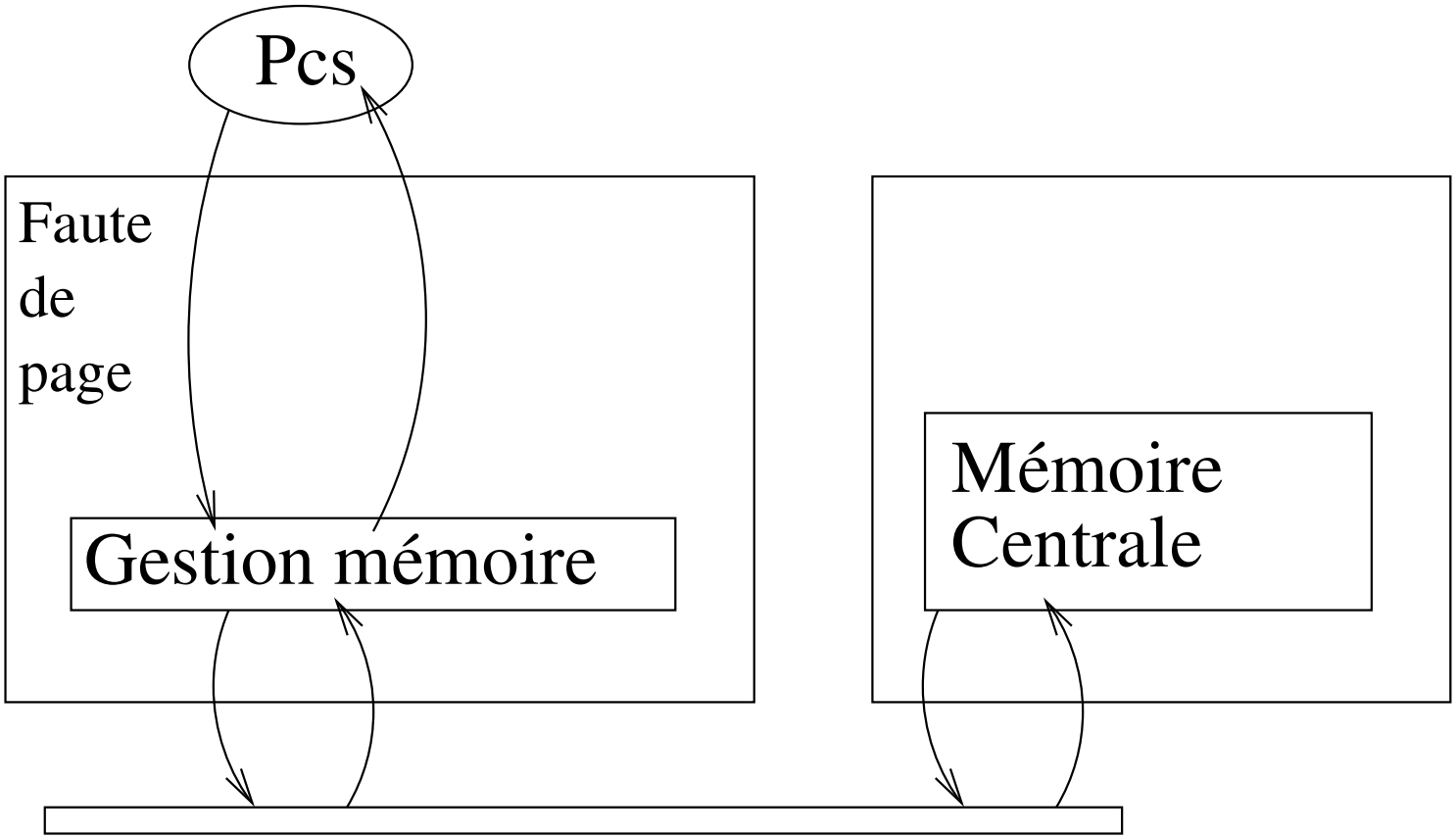
Mémoire virtuelle



Mémoire virtuelle - disque distant



Mémoire virtuelle - mémoire centrale distante



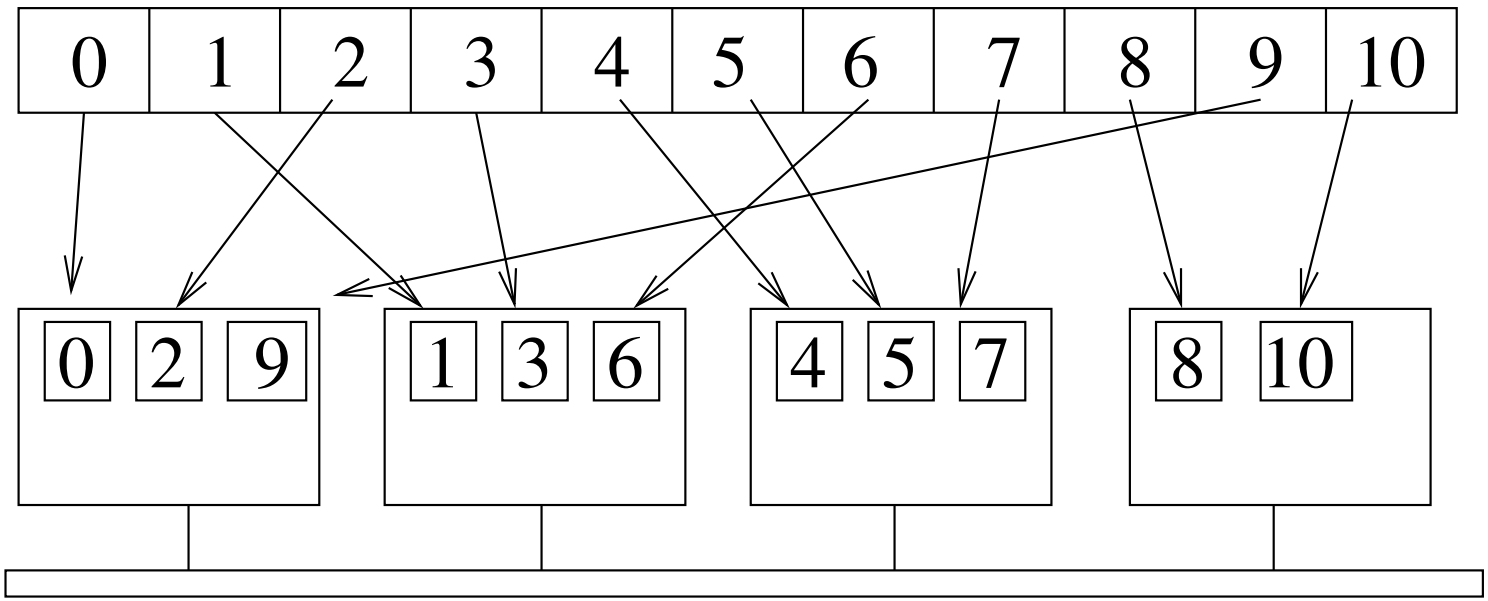
DSVM

- Cette dernière approche a principalement été utilisée pour la migration dynamique de processus
- Il a ensuite été plus loin pour obtenir la DSVM
- Avec la DSVM, on partage une zone de la mémoire virtuelle (zone globale en mémoire centrale)
- Les pages de cette zone sont distribuées sur différents sites

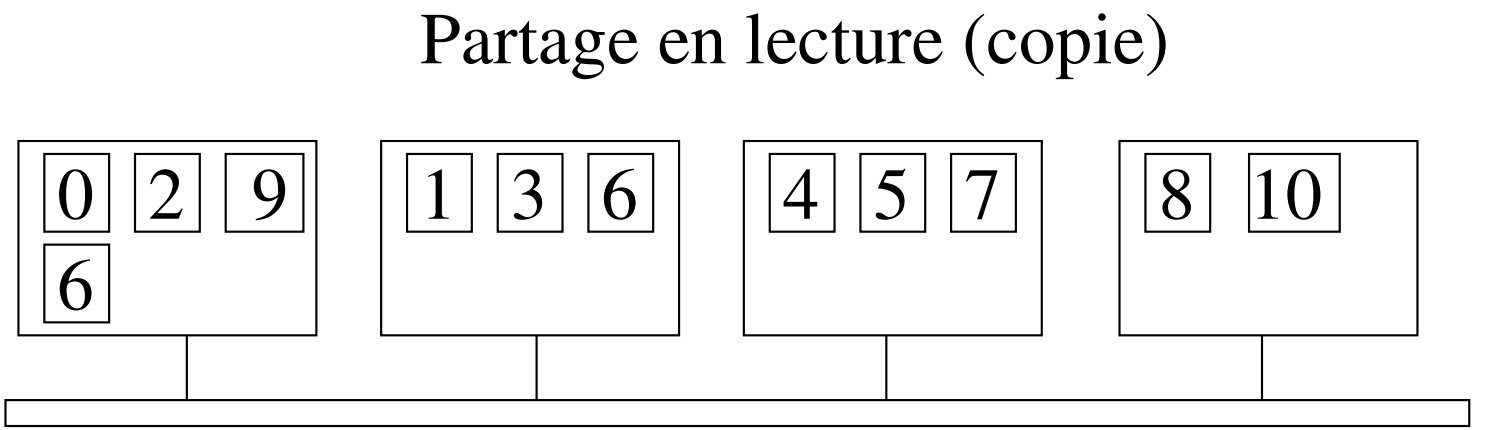
DSVM

- Avec la DSVM, chaque page se trouve en mémoire centrale d'une des machines
- Lorsqu'une machine fait référence à une page qui n'est pas en mémoire :
  - faute de page
  - la page est localisée et une demande de transfert est envoyée
  - la page est transférée (marquée absente sur l'ancien site)

DSVM



DSVM - Page 6 partagée



DSVM

- Qu'arrive-t-il si on tente de répliquer une page en écriture ?
- Qu'arrive-t-il si on tente de transférer une page dans laquelle le site écrit activement ?
  On peut la verrouiller et interdire le transfert (performance ?)
- Problème du faux partage ! !

DSVM

- Si les pages modifiables sont partagées ! !
- Situation semblable au problème des caches
- Solutions :
  - Lors d'une modification on élimine les copies
  - Verrouillage
  - Propagation des modifications seulement (verrous + optimisations)

DSVM

- Lors d'une référence, quelle est la taille du bloc transféré ?
- Plus c'est grand plus c'est efficace mais plus il y a de faux partage

Cohérence

- Le modèle de cohérence définit l'ensemble des valeurs qu'une lecture peut retourner.

Cohérence

- Cohérence stricte ou forte (atomique)
  Une lecture retourne la valeur de la plus récente écriture
- Cohérence séquentielle
  Le résultat de toute exécution est la même que si les lectures et écritures se sont produites dans un ordre quelconque et que les opérations sur un même processeur apparaissent dans la séquence définie par le programme.

Cohérence

- Cohérence de cache (cache consistency)
- Cohérence causale (causal consistency)
- Cohérence PRAM (PRAM)
- Cohérence de processeur (processor consistency)
- Cohérence lente (slow memory)

Cohérence

- Cohérence faible (weak consistency)
- Cohérence de libération (release consistency)
- Cohérence de libération paresseuse
- Cohérence d'entrée (entry consistency)
- Cohérence à terme

Gestion d'espaces d'adresses de 64 bits

- Avec les espaces d'adresses de 64 bits
- Un seul espace d'adresse pour tous les processus sur une même machine
- Un seul espace d'adresses pour tout le réseau
- On assigne une adresse à un objet pour toute sa vie (plus d'édition de liens)

Chapitre 7 - Systèmes multiprocesseurs et répartis

Gestion du temps

- Une des caractéristiques des systèmes répartis est qu'ils n'y a pas de concept de temps global ou unique
- Chaque système possède sa propre horloge physique
- Ces horloges n'enregistrent pas la même heure et évoluent à des rythmes différents
- Le concept d'ordonnancement dans le temps est la base de la synchronisation, alors comment peut-on avoir une gestion de temps unique ou globale ?

Qu'est-ce que le concept de temps

- Le temps sur Terre est établi par convention (il n'y a pas de temps universel)
- Pour fonctionner nous avons besoin d'une notion de temps
- Celle que nous utilisons est basée sur la rotation de la Terre en terme d'années et d'heures
- Les valeurs précises associées à ce temps sont aujourd'hui dérivée d'une horloge atomique
- Ce temps est appelé Universal Coordinated Time (UTC)

Les ordinateurs et le temps

- Les ordinateurs possède une horloge physique basée sur un cristal de quartz
- Elles sont programmables (généralement programmées pour générer des interruptions à toutes les 10ms)
- La précision de ces horloges est d'environ $1/10^6$ (1 seconde sur 11,6 jours)

Les ordinateurs et le temps

- Il est possible d'acheter un ordinateur contenant un périphérique qui peut recevoir un signal UTC soit par ondes radio, par satellite ou par le réseau (modem, ...)
- La précision est de 10ms pour les ondes radio
- La précision est de 0,1ms pour le service satellite GEOS (geostationnary operationnal environment satellite) et de 1ms pour le GPS
- La précision est de 1 millionième de seconde pour une horloge atomique

Les ordinateurs et le temps

- Il est toutefois impossible de connecter tous les ordinateurs à une source UTC
- On équipe seulement quelques ordinateurs qui servent de serveur de temps

Les ordinateurs et le temps

- Le problème demeure ! ! !
- Comment conserver un temps unique ou global même si les horloges dérivent et en présence de délais dans les communications ? ?
- Deux solutions : horloges logiques et synchronisation des horloges physiques

Utilité d'une horloge globale

- Synchronisation
- Réservation de ressources (billets d'avion, ...)
- Systèmes bancaire
- Environnement de programmation
- Jeux en réseau
- Bases de données distribuées (journal, log, ...)
- Contrôle du trafic aérien
- Simulation interactive
- ...

Horloges logiques

- Un système réparti est composé de processus qui collaborent vers l'atteinte d'un objectif commun
- Comme il est difficile d'avoir une horloge physique globale, on utilise une autre méthode pour assurer l'ordonnancement : les horloges logiques
- Les horloges logiques sont des compteurs d'événements dont la mise à jour est basée sur la causalité et le happens-before

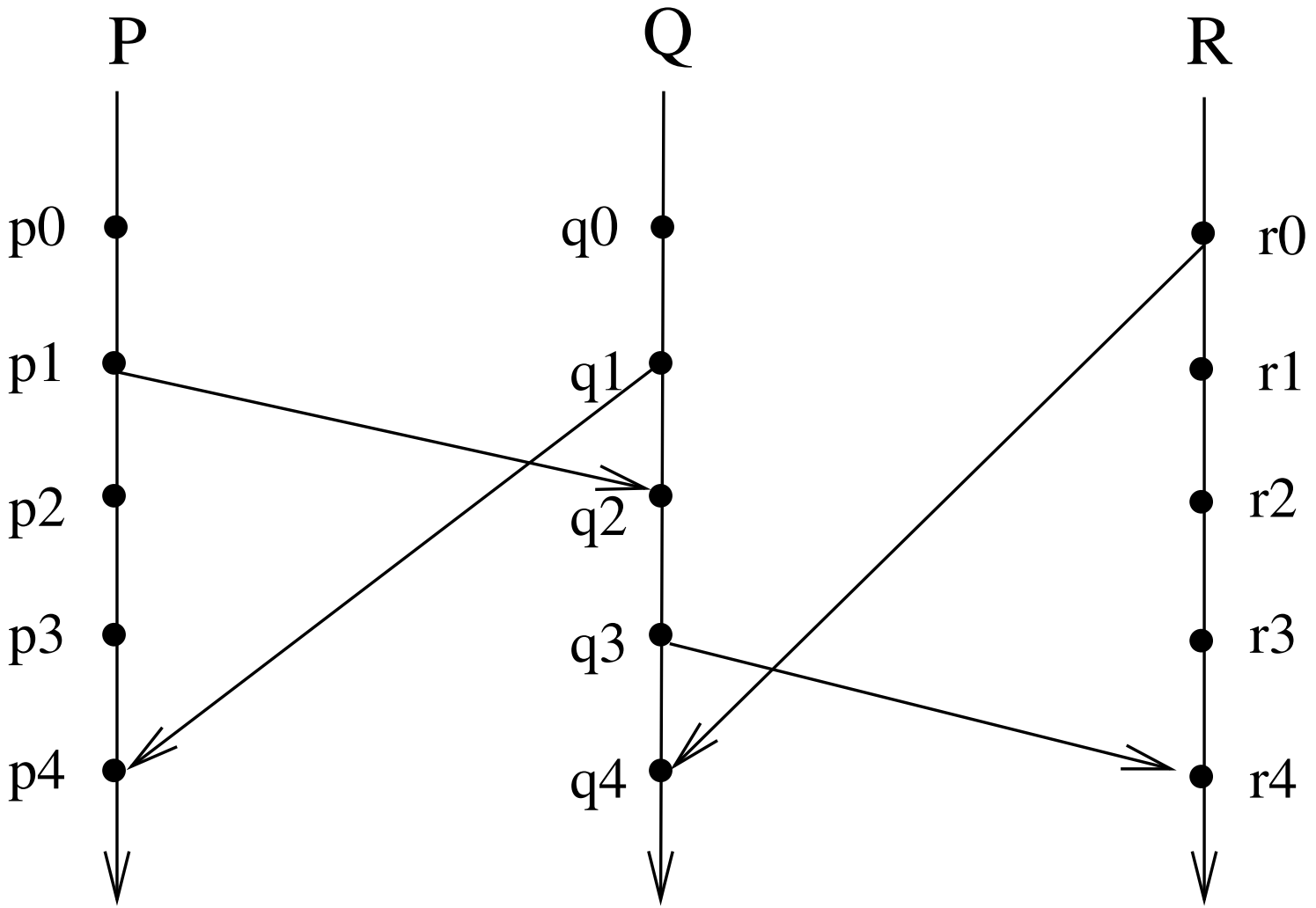
Horloges logiques

- Soit deux événements a et b, la relation «happens-before», est notée $a \rightarrow b$
- C'est une relation de cause à effet
- Ainsi $a \rightarrow b$ si :
  - a et b sont deux événements d'un même processus et a s'exécute avant b
  - a est l'envoi d'un message par un processus P_1 et b est la réception de ce message par un processus P_2
  - il existe un événement c tel que $a \rightarrow c$ et $c \rightarrow b$ (transitivité)

Horloges logiques

- La relation $a \rightarrow b$ crée un ordre partiel irréflexif sur les événements
- Si a et b ne sont pas reliés par la relation $\rightarrow$, ils sont dits concurrents (a ne s'est pas produit avant b et b ne s'est pas produit avant a)
- Si $a \rightarrow b$ alors l'événement a peut affecter l'événement b.

Horloges logiques



Horloges logiques

- $p1 \rightarrow q2$,
  $r0 \rightarrow q4$,
  $q3 \rightarrow r4$,
  $p1 \rightarrow q4 \quad (p1 \rightarrow q2 \rightarrow q4)$
- $q0 \parallel p2$,
  $r0 \parallel q3$,
  $r0 \parallel p3$,
  $q3 \parallel p3$

Horloges logiques

- Dans bien des situations, les événements concurrents n'ont pas besoin d'être ordonnés car ils sont indépendants
- Dans d'autres situations, il faut un ordre total

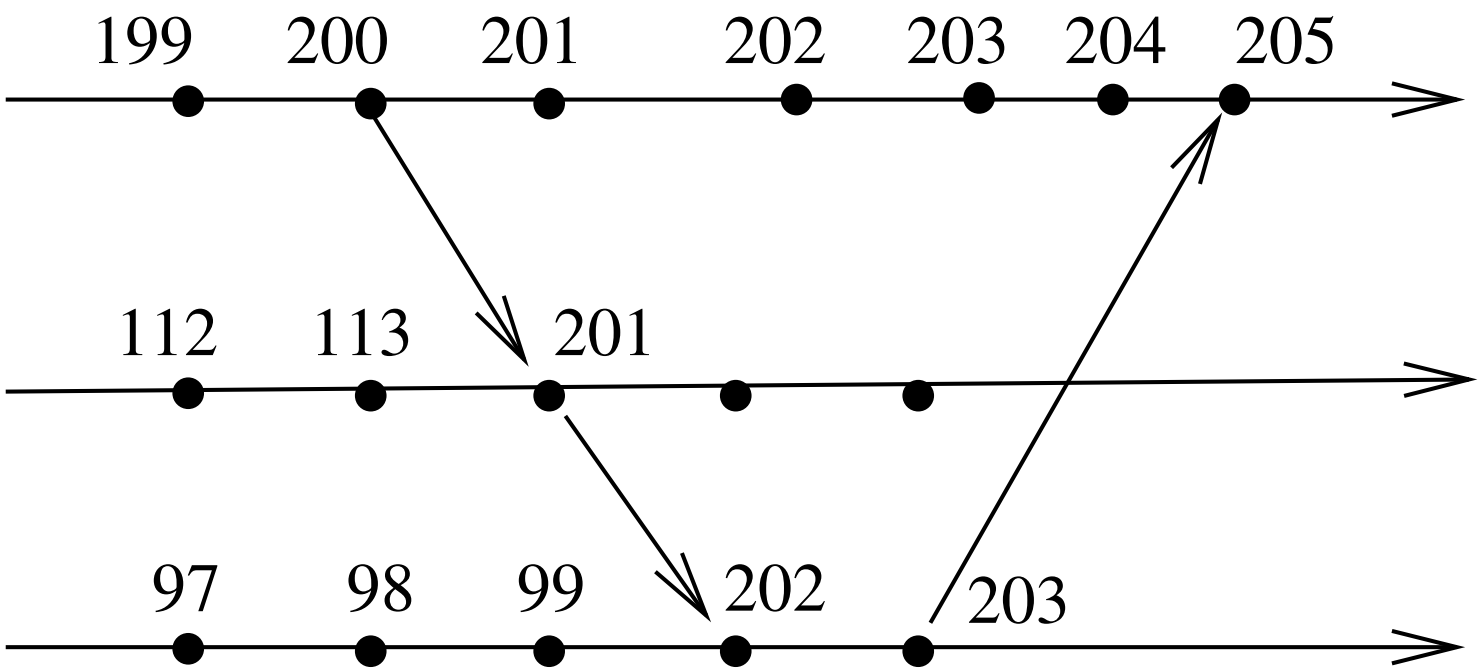
Implantation

- On associe une horloge logique (L_i) à chaque processus (P_i)
- Cet horloge ne mesure pas le temps.
- C'est un compteur qui est incrémenté à chaque événement du processus (compteur d'événements)
- On se sert de cet horloge pour affecter une estampille (S) à chaque événement pour obtenir un ordre total dans un même processus
- Si $a \rightarrow b$ alors $S(a) < S(b)$

Implantation

- Pour étendre notre système à un ensemble de processus, on s'assure que chaque envoi de message possède une estampille inférieure à sa réception : $S(send(m)) < S(receive(m))$
- Pour y parvenir, chaque envoi de message est estampillé avec l'heure de l'horloge logique du processus P_i qui fait l'envoi
- Lorsque le message est reçu par un processus P_j, l'horloge logique de P_j est mise à jour de la façon suivante : $L_j = MAX(L_j + 1, S(send(m) + 1)$
- La réception du message est estampillé avec cette horloge à jour (L_j).

Implantation



Horloges physiques

- Comment synchroniser des horloges physiques ? ?
- Il existe différents types d'algorithmes :
  - ceux qui se synchronisent avec un serveur de temps UTC
  - ceux qui se synchronisent sans serveur de temps UTC
  - ceux qui fonctionnent pour une masse de clients

Horloges physiques - avec serveur UTC

- Soit un serveur de temps
- Le client interroge le serveur périodiquement pour ajuster l'heure (la période dépend de la précision désirée)
- Le serveur retourne l'heure ou un écart
- Le client reçoit le message
  - il peut l'utiliser directement
  - il peut l'ajuster pour compenser les délais de communication en ajoutant un délai minimal connu ou la moitié du temps depuis l'envoi de la demande.

Horloges physiques - avec serveur UTC

- Si la valeur du serveur $>$ l'heure locale
  - ajuste l'heure directement
  - accélère temporairement l'horloge pour le rattrapage
- Si la valeur du serveur $<$ l'heure locale
  - ajuste l'heure directement
  - ralentit l'horloge (en ajustant les incréments)
    Peut par exemple provoquer les interruptions aux 11ms au lieu des 10ms normales et incrémenter l'horloge de 10ms à chaque interruption

Horloges physiques - sans serveur UTC

- Un ordinateur peut servir de coordonnateur
- Il demande le temps de chaque machine, calcule une moyenne et diffuse le résultat
- Les machines ajustent leur heure (comme avec un serveur UTC)
- Des ajustements manuels peuvent se faire

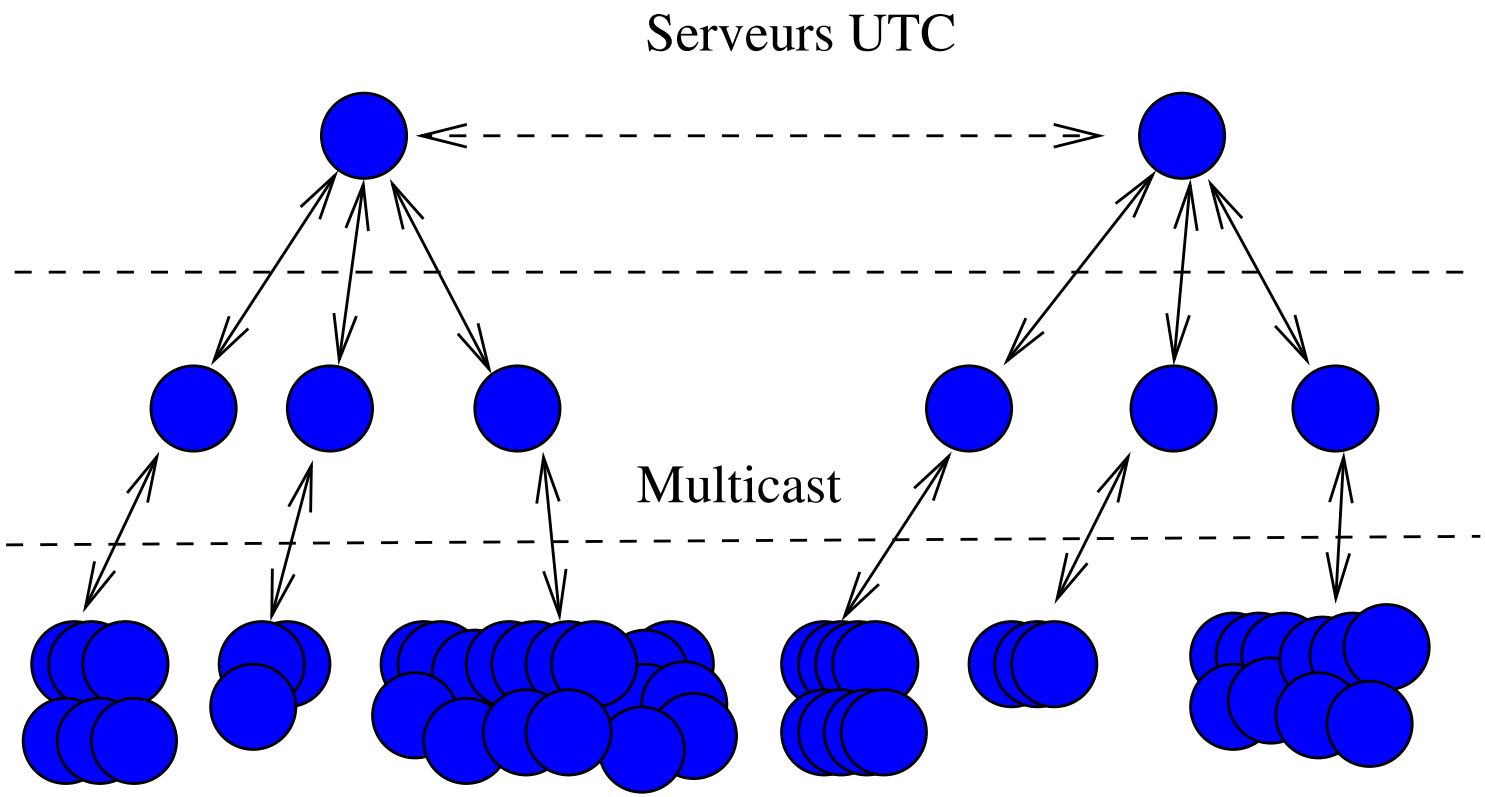
Horloges physiques - grande échelle

- Un seul serveur n'est pas idéal (fiabilité, performance)
- Les approches basées sur un seul serveur ne supporte pas un accroissement significatif du nombre de clients
- Certains protocoles sont prévus pour servir à grande échelle (NTP)
- NTP parvient à synchroniser les horloges à quelques dizaines de millisecondes près

Horloges physiques - grande échelle

- Approche normale : hiérarchie de serveurs
- NTP utilise des hiérarchies à trois niveaux
  - À la racine, il y a les serveurs primaires avec les récepteurs UTC
  - Ceux-ci propage l'heure au niveau inférieur, les serveurs secondaires ou les clients
  - Au second niveau, il y a un groupe de serveurs qui interagissent avec le serveur UTC
  - Chaque second niveau diffuse ensuite l'heure au troisième niveau

NTP



Chapitre 7 - Systèmes multiprocesseurs et répartis

Synchronisation

- Nous avons vu la synchronisation sur les systèmes centralisés
- Comment peut-on l'implanter dans un environnement réparti
- Le système réparti est modélisé comme un ensemble de processus (un par machine) numéroté de 1 à n
- Il y a plusieurs approches à la synchronisation

Approche centralisée

- Un des processus sert de coordonnateur
- Un processus qui veut entrer en exclusion mutuelle envoie un message de type *demande* au coordonnateur
- Lorsque le processus reçoit la *réponse*, il entre en section critique
- Lorsqu'il sort de la section critique, il envoie un message de type *libération* au coordonnateur

Approche centralisée

- Lorsque le coordonnateur reçoit un message de type *demande*, il vérifie si un processus occupe déjà la section critique
- Si la section critique est disponible, il envoie une *réponse*
- Si la section critique n'est pas disponible, la demande est mise en attente (file d'attente)
- Lorsque le coordonnateur reçoit un message de type *libération*, il retire une demande de la file d'attente et envoie une *réponse*.

Approche centralisée

- Cet algorithme assure l'exclusion mutuelle et assure l'équité (dépendamment de la façon dont la file d'attente est gérée)
- Cette approche nécessite trois messages par entrée en section critique
- Qu'arrive-t-il si le coordonnateur tombe en panne ?

Approche centralisée

- Cet algorithme assure l'exclusion mutuelle et assure l'équité (dépendamment de la façon dont la file d'attente est gérée)
- Cette approche nécessite trois messages par entrée en section critique
- Qu'arrive-t-il si le coordonnateur tombe en panne ?
  Il doit y avoir élection et le nouveau coordonnateur interroge les processus pour reconstruire la file d'attente

Approche distribuée

- Lorsqu'un processus P_i veut entrer en section critique, il génère une estampille E en envoie un message $demande(P_i, E)$ à tous les processus du système incluant lui-même
- Lorsqu'une *demande* est reçue, un processus peut répondre immédiatement ou attendre
- Un processus qui a reçu des réponses de tous les processus peut entrer en section critique (met les demandes qu'il reçoit en attente)

Approche distribuée

Un processus répond ou non à une demande selon certains critères :

- Si le processus est en section critique il met la demande en attente
- Si le processus ne veut pas entrer en section critique, il envoie une réponse
- Si le processus veut entrer en section critique mais n'a pas reçu toutes les réponses :
  - il compare l'estampille de sa demande avec celle de la demande reçue
  - Si son estampille est plus grande, il envoie une réponse
  - Si son estampille est plus petite, il met la demande en attente

Approche distribuée

Soit trois processus qui participent à ce protocole :

- P_1 fait une demande au temps $E = 10$;
- P_2 ne fait aucune demande ;
- P_3 fait une demande au temps $E = 4$.

Approche distribuée

P_1	P_2	P_3
$(P_1, 10) \rightarrow$		$(P_3, 4) \rightarrow$
	$\rightarrow (P_1, 10) \rightarrow R$	
		$\rightarrow (P_1, 10)$
$\rightarrow (P_3, 4) \rightarrow R$	$\rightarrow (P_3, 4) \rightarrow R$	
$\rightarrow (P_2, OK)$		$\rightarrow (P_1, OK)$
		$\rightarrow (P_2, OK)$
		Excl. mut
		$(P_1, 10) \rightarrow$ R
$\rightarrow (P_3, OK)$		
Excl. mut		

Approche distribuée

- Cet algorithme assure l'exclusion mutuelle sans interblocage
- Il est équitable
- Le nombre de messages pour une entrée en section critique est $2 \times (n - 1)$
- Il faut connaître les processus impliqués (l'ajout d'un processus est complexe)
- Qu'arrive-t-il si un processus tombe en panne ?

Passage de jeton

- Un jeton est un message spécial qui circule dans le système
- La possession du jeton donne le droit d'entrer en section critique
- Pour faire circuler le jeton convenablement, les processus sont organisés en anneau logique

Passage de jeton

- L'exclusion mutuelle est garantie (car un seul jeton)
- C'est équitable si l'anneau est unidirectionnel
- Le nombre de messages par entrée varie de 1 à une infinité...
- Qu'arrive-t-il si le jeton est perdu ?
- Qu'arrive-t-il si un processus tombe en panne ?

Passage de jeton

- L'exclusion mutuelle est garantie (car un seul jeton)
- C'est équitable si l'anneau est unidirectionnel
- Le nombre de messages par entrée varie de 1 à une infinité...
- Qu'arrive-t-il si le jeton est perdu ? Élection
- Qu'arrive-t-il si un processus tombe en panne ? Reconstruction

Atomicité

- Nous avons déjà mentionné qu'une technique pour gérer les fichiers répartis requiert l'utilisation du concept de transaction atomique
- Une transaction est atomique si toutes les opérations qui lui sont associées sont exécutées entièrement ou aucune n'est exécutée
- Implanter une transaction atomique sur un système réparti est complexe car plusieurs sites peuvent participer à la transaction (mise à jour de copies multiples)
- La panne d'un ou plusieurs sites ne doit pas nuire à l'atomicité de la transaction

Atomicité

- Pour implanter les transactions atomiques, on utilise des coordonnateurs de transactions
- Chaque site possède un coordonnateur de transactions qui est responsable des transactions démarrées localement
- Le coordonnateur doit :
  - démarrer la transaction locale
  - briser la transaction en sous-transactions
  - distribuer les sous-transactions vers les sites appropriés
  - coordonner la fin de la transaction (engagement)

Atomicité

- Le problème est de coordonner la fin de la transaction
- On suppose que chaque site possède un journal et une mémoire stable (le journal est copié en mémoire stable)
- Tous les sites doivent s'entendre sur le résultat final de la transaction (engagement ou annulation)
- Pour assurer cette entente, le coordonnateur exécute un protocole d'engagement
- Le principal algorithme d'engagement est l'engagement à deux phases

Engagement à deux phases

- Soit T une transaction initiée à un site S_i par un coordonnateur C_i
- Lorsque T complète son exécution, C_i débute le protocole d'engagement
- Phase 1 :
  - C_i enregistre $<$ *prepare* $T >$ (stable) dans son journal
  - C_i envoie $<$ *prepare* $T >$ à tous les sites
  - un site qui reçoit ce message décide de s'engager ou non
  - s'il ne s'engage pas, il enregistre (stable) $<$ *non* $T >$ puis envoie $<$ *annulation* $T >$ à C_i
  - s'il s'engage, il enregistre (stable) $<$ *ok* $T >$ et tous les éléments de T puis envoie $<$ *ok* $T >$ à C_i

Engagement à deux phases

- Phase 2 :
  - Lorsque C_i a reçu toutes les réponses (ou un timeout)
  - Si engagement, il enregistre (stable) $<$ *engagement* $T >$ et envoie $<$ *engagement* $T >$ à tous les sites
  - Si annulation, il enregistre (stable) $<$ *annulation* $T >$ et envoie $<$ *annulation* $T >$ à tous les sites
  - Lorsqu'un site reçoit ce message il l'enregistre dans le journal

Engagement à deux phases

- Un site peut annuler T en tout temps avant son engagement
- Un fois l'engagement expédié un site doit compléter la transaction même en présence de pannes
- L'engagement est précédé de l'enregistrement de toutes les informations pour exécuter ou ré-exécuter T en mémoire stable
- Si un seul site refuse de s'engager, elle est entièrement annulée

Engagement à deux phases

- Qu'arrive-t-il s'il y a une panne d'un site ?
- Qu'arrive-t-il s'il y a une panne du coordonnateur ?
- Qu'arrive-t-il s'il y a une panne du réseau ?

Panne d'un site

- Lorsque le site effectue la reprise après sa panne, il examine son journal
- Si le journal contient $< \textit{engagement } T >$ $\rightarrow$ exécute $\textit{redo}(T)$
- Si le journal contient $< \textit{annulation } T >$ $\rightarrow$ exécute $\textit{undo}(T)$
- Si le journal contient $< \textit{ok } T >$ $\rightarrow$
  1. consulte C_i
  2. C_i retourne état $\rightarrow$ $\textit{undo}(T)$ ou $\textit{redo}(T)$
  3. si C_i est en panne, contacte les autres sites
  4. si pas d'information, attend et interrogation périodique
- Si le journal ne contient rien.....$\textit{undo}(T)$

Panne du coordonnateur

- Les sites participants doivent décider
- Si au moins un site contient $< \textit{engagement } T >\rightarrow T$ est engagé
- Si au moins un site contient $< \textit{annulation } T >\rightarrow T$ est annulé
- Si quelques sites ne contiennent pas $< \textit{ok } T >\rightarrow T$ est annulé
- Si aucun des cas précédents
  - tous les sites contiennent $< \textit{ok } T >$
  - impossible de connaître la décision de C_i
  - attend la reprise de C_i
  - ressources verrouillées ! ! !

Panne du réseau

- Panne d'une lien
  Cela apparaît comme une panne d'un site ou du coordonnateur
- Panne de plusieurs liens
  Création de sous-réseaux (partitionnement). Cela apparaît comme une panne d'un site ou du coordonnateur

Transactions concurrentes

- Synchronisation de transactions concurrentes
  - verrouillage (réparti, centralisé, protocole majoritaire, ...)
  - estampilles
    Les opérations en conflits sont résolues selon l'ordre des estampilles.

Autres sujets

- Élections
  - Plusieurs algorithmes utilisent un coordonnateur
  - S'il tombe en panne on doit en trouver un autre...
  - *Bully algorithm* : Si un processus détecte que le coordonnateur est en panne il tente de se faire élire nouveau coordonnateur
  - *Ring algorithm* : le processus avec le plus grand identificateur est élu

Autres sujets

- Accords (agreement)
  - Pour des raisons de fiabilité, il est nécessaire d'avoir un mécanisme qui permet à plusieurs processus de s'entendre sur une valeur commune
  - Raisons d'un tel accord
    - panne impropre du réseau (diffusion de valeurs erronées)
    - panne impropre d'un site (diffusion de valeurs erronées)
  - Byzantine generals problem

Chapitre 7 - Systèmes multiprocesseurs et répartis

Interblocage – Analogie

- Loi passée au Kansas au début du $20^{ième}$ siècle
  When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.
- Catch-22
  Terme utilisé en référence au roman de Joseph Heller pour désigner une situation où un individu ne peut éviter un problème en raison de la contradiction des règles ou des contraintes.

Interblocage

- Tous les algorithmes déjà vus peuvent être adaptés aux DOS
- Prévention et évitement
  - ordonnancement global
  - estampilles + sans réquisition (wait-die)
    P_i attend que P_j libère la ressource ssi son estampille est plus petite (sinon meure)
  - estampilles + réquisition (wound wait)
    P_i attend que P_j libère la ressource ssi son estampille est plus grande. Si l'estampille de P_i est plus petite, P_j est réquisitionné.

Interblocage

- Détection
  - couvre plusieurs sites
  - on construit un wait-for graph sur le système réparti
  - chaque site construit un graphe local.
  - si graphe local contient un cycle, interblocage
  - sinon on effectue l'union des graphes locaux
  - union et détection se font par une approche centralisée ou répartie

Interblocage

- Problèmes de la détection
  - Quand la mise à jour du graphe global se fait-elle ?
    À toutes les modifications locales, périodiquement ou lorsque le coordonnateur exécute son algorithme de détection.
  - Risque de faux cycles..

Chapitre 7 - Systèmes multiprocesseurs et répartis

Autres

- Protection
- Hétérogénéité (XDR)