Sixième partie

Mémoire virtuelle
Systèmes de fichiers

Plan

Origine

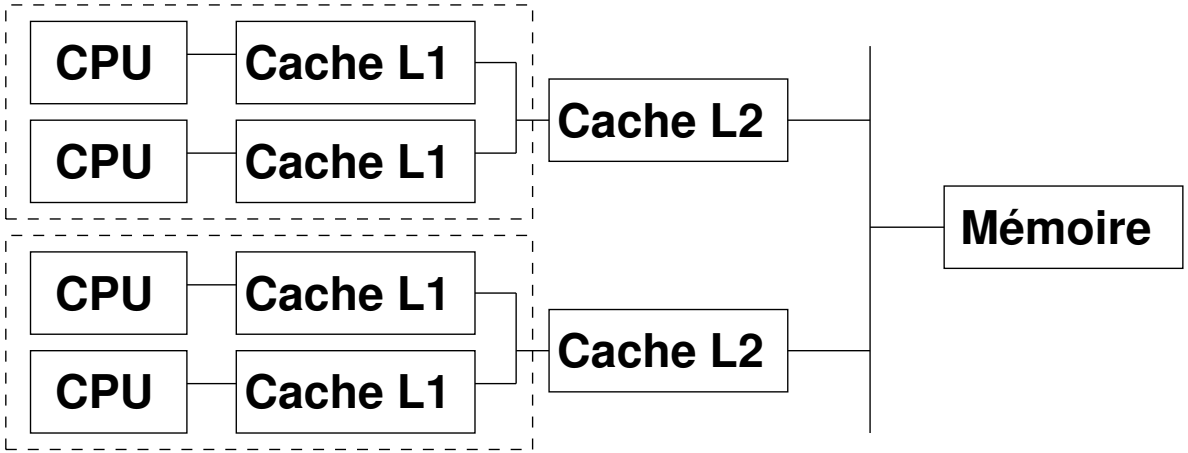
Réplication

Placement de plusieurs exemplaires d'une même donnée sur différents sites.

Exemples de partage :

- partage de la mémoire commune dans un SMP (symmetric multi-processor)
- partage d'un système de fichiers (NFS)
- partage de mémoire virtuelle

Mémoire partagée en multi-processeurs multi-cœurs



Cohérence des écritures en présence de caches ?

Write-Through diffusion sur le bus à chaque valeur écrite

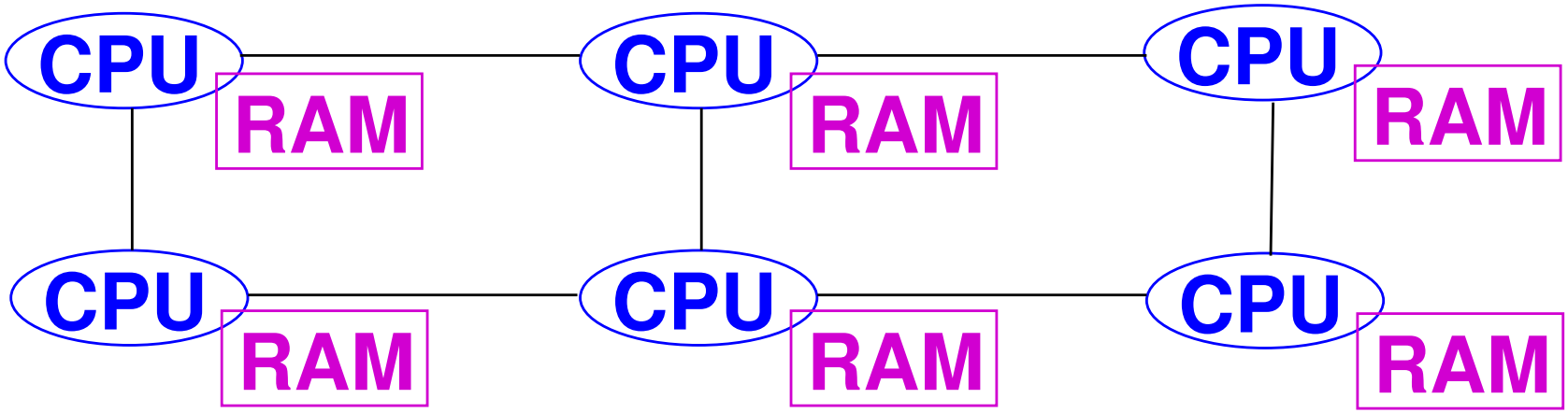
- \+ visible par les autres processeurs
- \+ la mémoire et le cache sont cohérents
- – trafic inutile : écritures répétées, écritures privées

Write-Back diffusion uniquement à l'éviction de la ligne

- \+ trafic minimal
- – cohérence cache - mémoire - autres caches ?

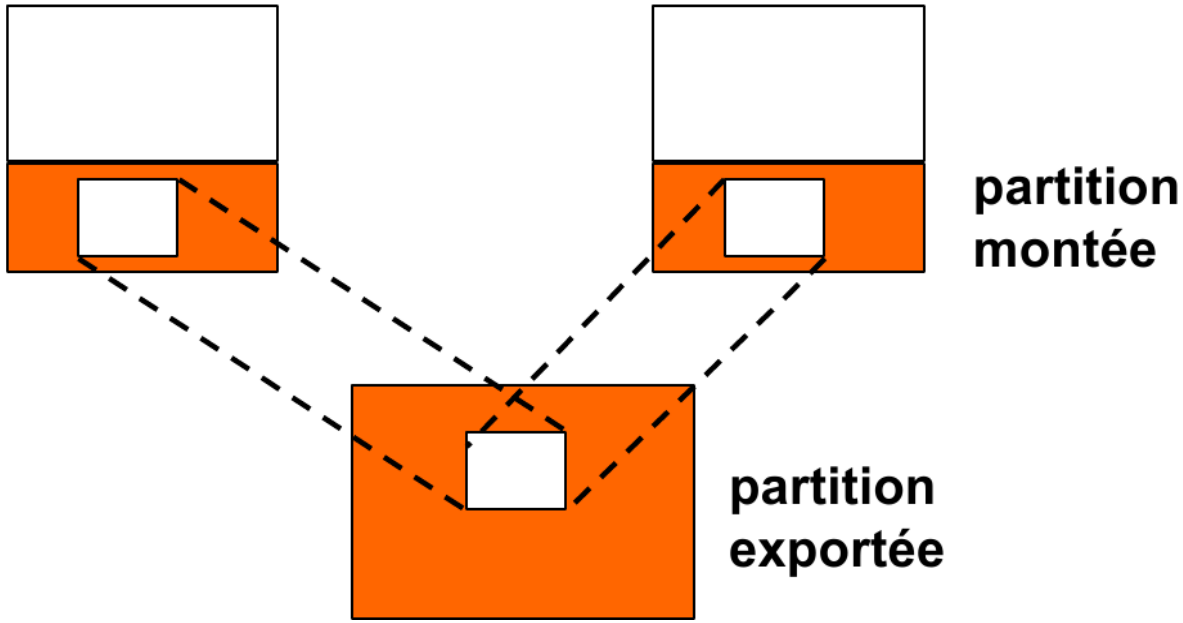
Mémoire partagée en NUMA

Non-Uniform Memory Access



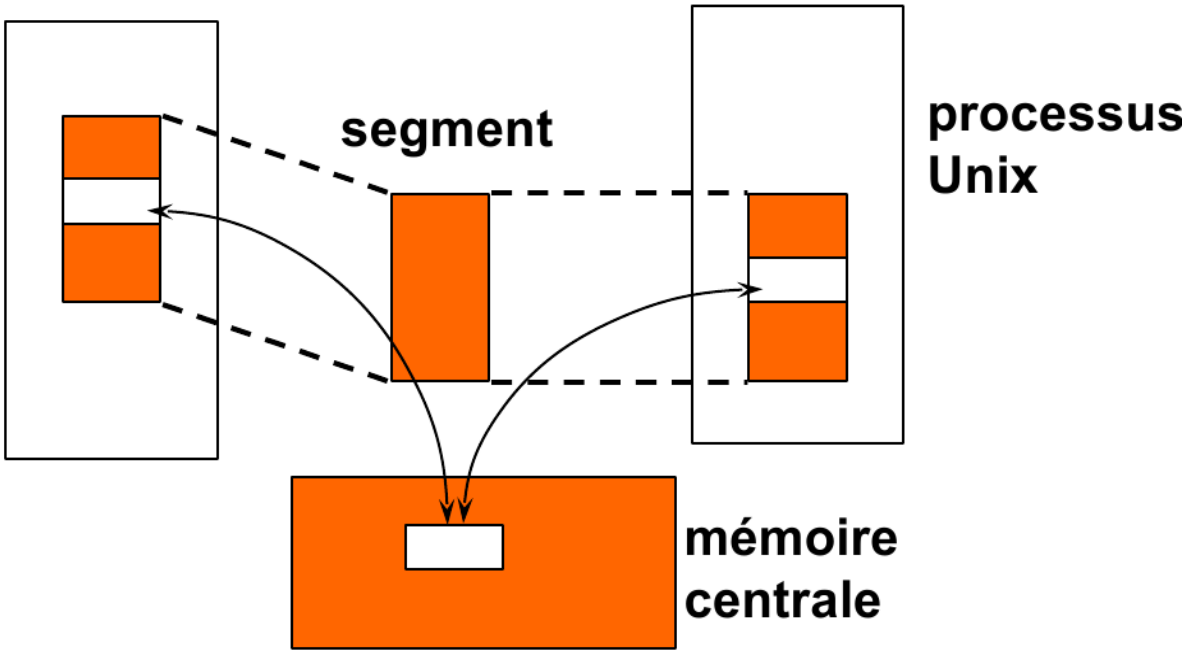
Système de fichiers partagé/réparti

Partage d'un système de fichier



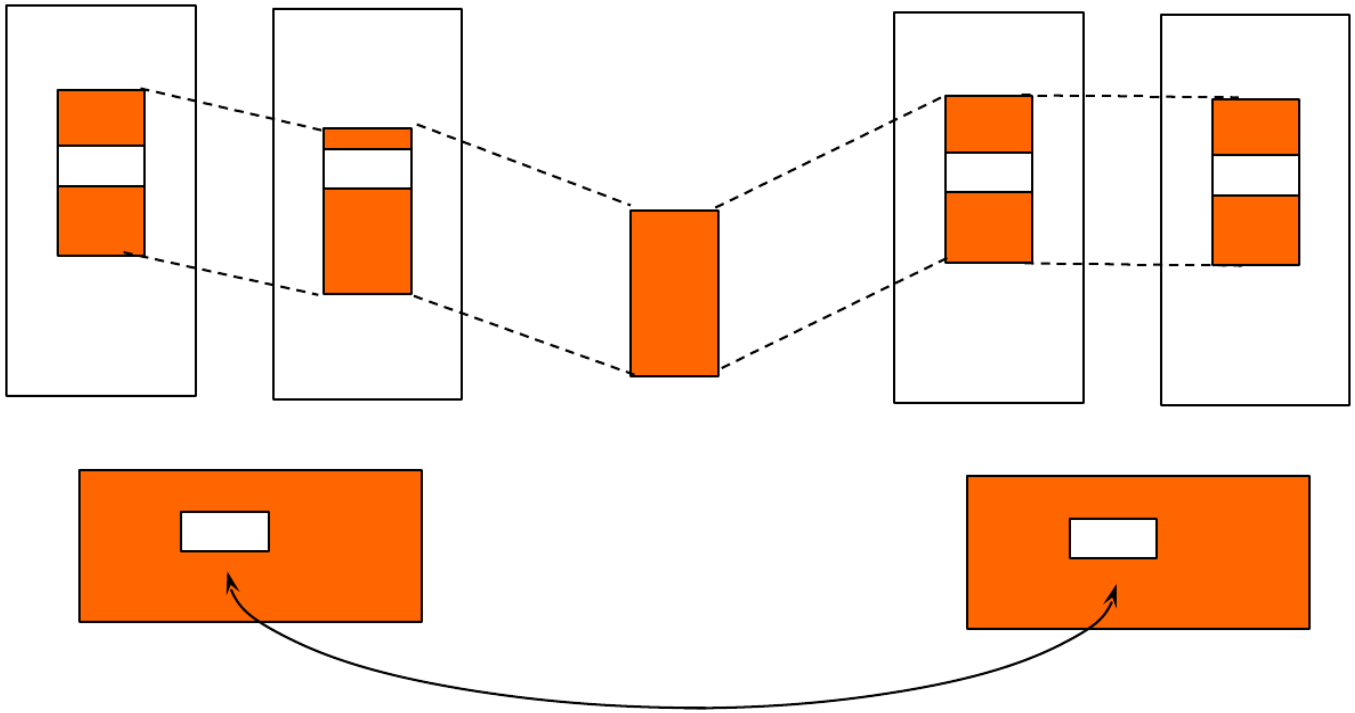
- cache NFS sur les sites clients
- invalidations périodiques

Mémoire virtuelle partagée (centralisée)



- partage de segments (suite de pages)
- partage de la mémoire centrale par couplage

Mémoire virtuelle partagée (distribuée)



Principe général

- Principe : fournir des objets partagés par couplage dans les espaces d'adressage de structures d'exécution (couplage virtuel) réparties
- Partage par copie locale (efficacité)
- Programmation simple (accès local)
- Le système charge éventuellement les données à la demande
- Le système assure la cohérence des données partagées

Paramètres d'implantation

Niveau

- Matériel (multiprocesseurs)
- Système d'exploitation (pagination)
- Langages ou librairies utilisateurs

Unité de partage

Octets / Pages / Objets

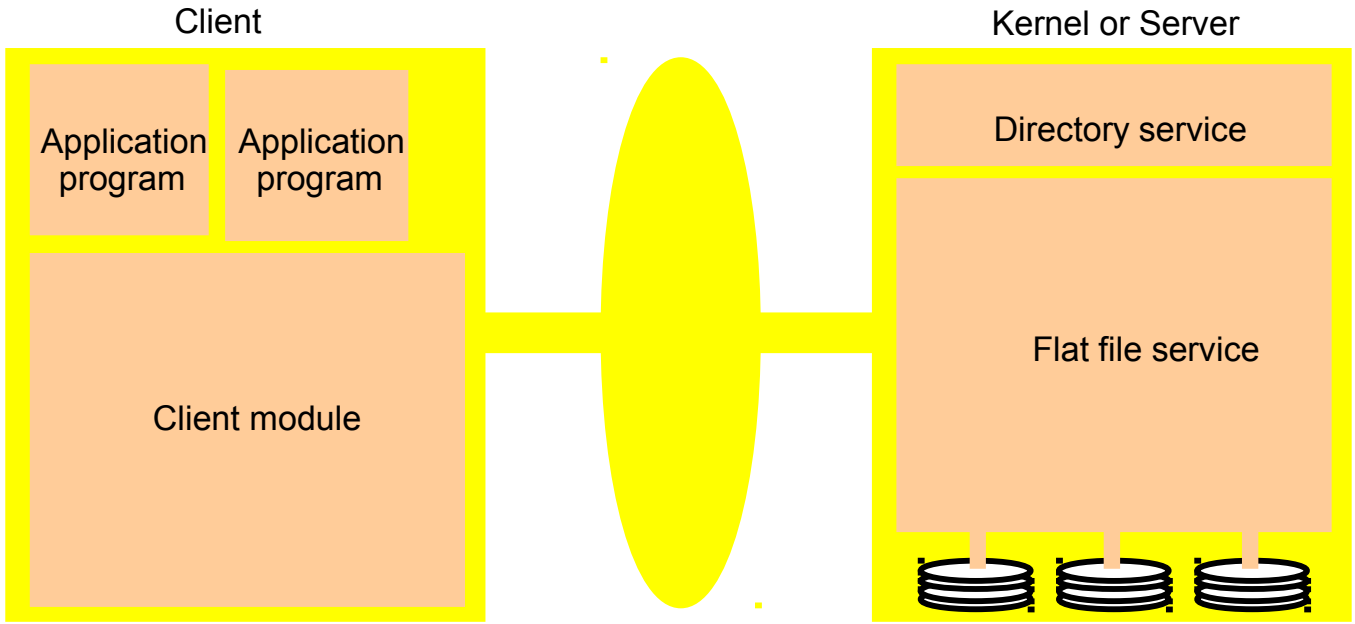
Cohérence

- Propagation des mises à jour
- Différents degrés de cohérence acceptables

Plan

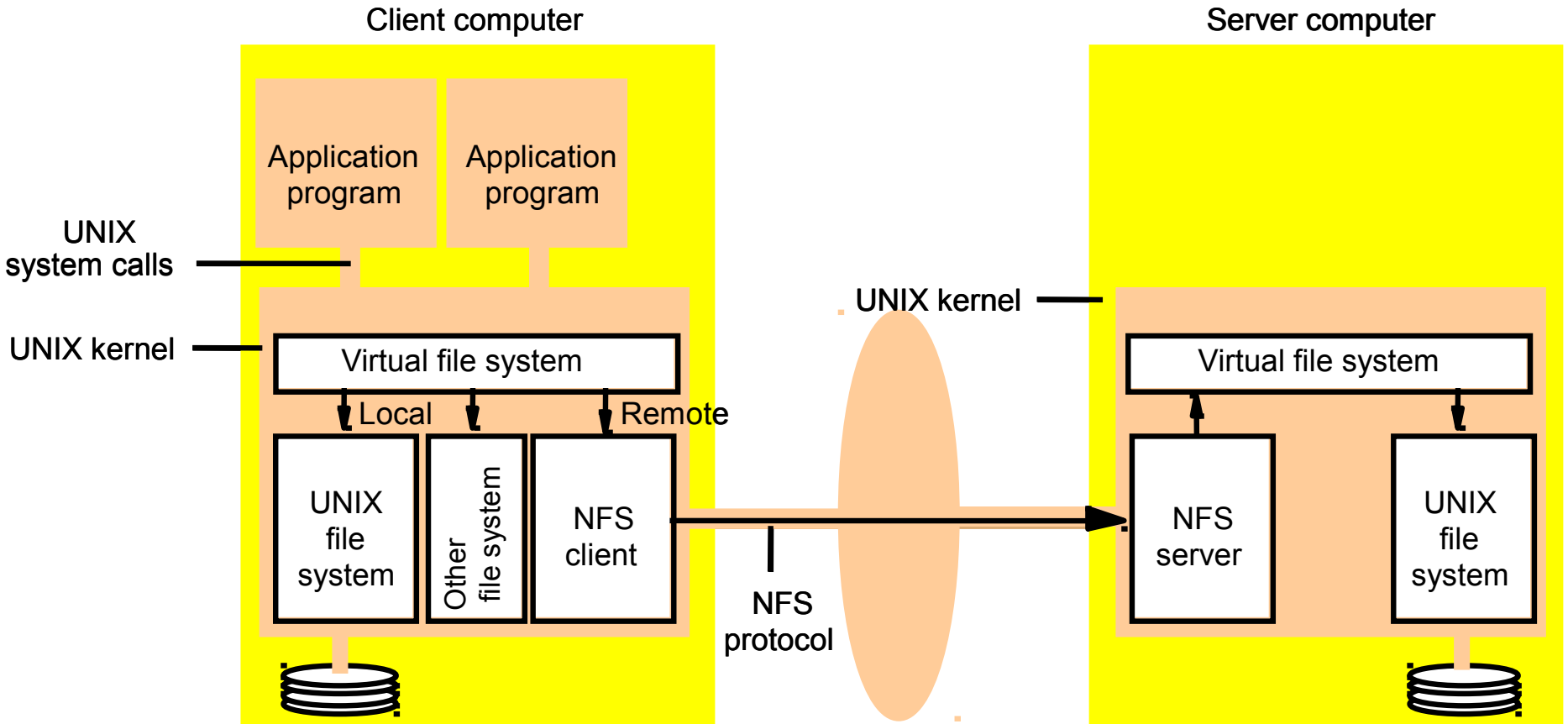
Architecture d'un système de fichiers

- Service de fichiers à plat (flat file system)
- Service de nommage (répertoires)
- Module client : API simple et unique, indépendante de l'implantation



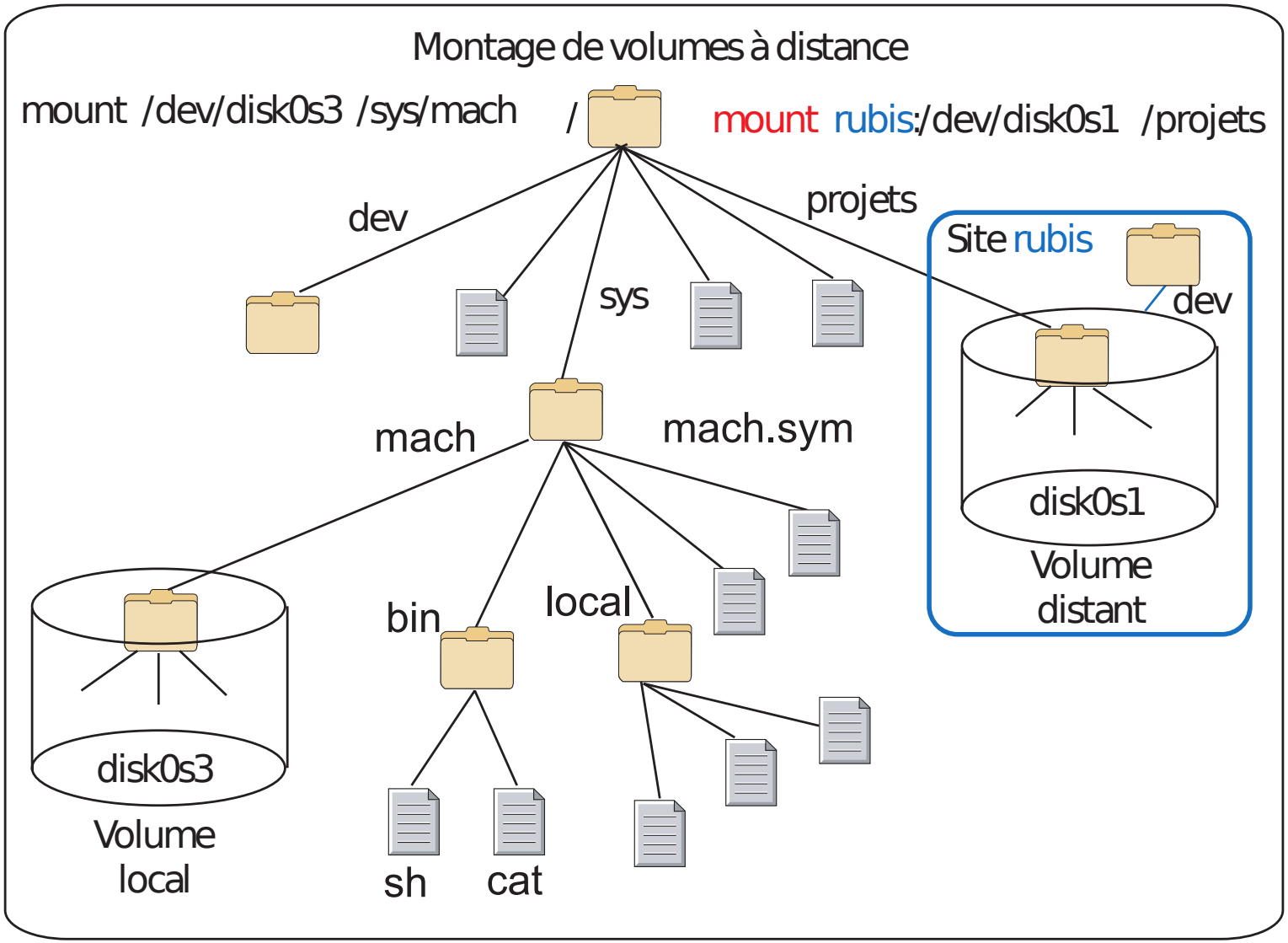
(source : Coulouris – Dollimore)

Architecture de NFS



(source : Coulouris – Dollimore)

Idée de base de NFS : le montage à distance de volumes



Exemple NFS

Transparence de localisation et hétérogénéité

La notion de *virtual file system* (VFS)

- Étend la notion de système de fichiers (file system) aux volumes à distance et à des systèmes de fichiers hétérogènes
- VFS $\rightarrow$ un volume support d'un système de fichiers particulier

La notion de *virtual node* (vnode)

- Extension de la notion de *inode*
- Pointe un descripteur local (*inode*) ou distant (*rnode*)

Notion de file handle ou UFID

Un nom global, engendré par le serveur et donné au client

Identification du FS	n^o inode	n^o de génération

SGF centralisé

API Unix

canal open(nom,mode)	connexion du fichier à un canal
canal creat(nom, mode)	connexion avec création
close(canal)	déconnexion
int read(canal,tampon,n)	lecture de *n* octets au plus
int write(canal,tampon,n)	écriture de *n* octets au plus
pos = lseek(canal, depl, orig)	positionnement du curseur
stat(nom, attributs)	Lecture des attributs du fichier
unlink(nom)	suppression du nom dans le rép.
link(nom_orig,synonyme)	Nouvelle référence

SGF réparti : Niveau répertoire

Fonction

- Noms symboliques $\leftrightarrow$ noms internes (Unique File IDentifiers UFID)
- Protection : contrôle d'accès

Génération de noms internes (UFID) :

- Engendrés par le serveur pour les clients
- Unicité, non réutilisation, protection

API RPC Service Répertoire

UFID Lookup(UFID rép, String nom)	résolution du nom
AddName(UFID rép, String nom, UFID uid)	insérer le nom
UnName(UFID rép, String nom)	supprimer le nom
String[] GetNames(UFID rép, String motif)	recherche par motif

SGF réparti : Niveau fichier

Fonction

- Passage d'un nom interne global (UFID à un descripteur)
- Accès au fichier (attributs)

API RPC Service Fichier

byte[] Read(UFID uid, int pos,int n)	lire *n* octets au + en *pos*
Write(UFID uid,int pos,byte[] z,int n)	écrire *n* octets en *pos*
UFID Create()	créer un fichier
Delete(UFID uid)	supprimer l'UID du fichier
GetAttributes(UFID uid, Attributs a)	lire les attributs du fichier
Attributs SetAttributes(UFID uid)	mettre à jour les attributs

- Opérations idempotentes (sauf create) : RPC at-least-once
- Serveur sans état : redémarrage sans reconstruction de l'état précédent

Gestion de caches clients et serveur

Objectif irréaliste

Garantir la sémantique centralisée :

la version lue est la dernière version écrite

Approximation...

- Sur ouverture, le client mémorise :

  (date de dernière modification, date présente)

- Interrogation du serveur si lecture après plus de p secondes ($p = 3$ sec. pour fichier ordinaire, $p = 10$ sec. si répertoire)

Sémantique de session

Stratégie orientée session

- Un serveur unique maintient la version courante
- Lors de l'ouverture à distance par un client (début de session), une copie locale au client est créée
- Les lectures/écritures se font sur la copie locale
- Seul le client rédacteur perçoit ses écritures de façon immédiate (sémantique centralisée)
- Lors de la fermeture (fin de session), la nouvelle version du fichier est recopiée sur le serveur :
  $\Rightarrow$ La « session » (d'écritures du client) est rendue visible aux autres clients

L'approche session (suite)

Problème

Plusieurs sessions de rédacteurs en parallèle...

- L'ordre des versions est l'ordre de traitement des fermetures
- La version gagnante est indéfinie

Autres solutions

- Invalidation les copies clientes par le serveur lors de chaque création de version
- Le contrôle du parallélisme : garantir un seul rédacteur
- Une autre idée : Fichiers à version unique (immuable)

Évolutions

Limitations

- Introduction de la mobilité : partitionnement réseau, mode déconnecté, pannes, etc
- Réplication (résolution des conflits d'écriture ?)
- Unité de travail = document = ensemble de fichiers

$$\Rightarrow \text{Constant Data Availability}$$

Exemples d'évolutions

- Coda (CMU) : réplication optimiste en mode déconnecté, résolution de conflit manuelle
- Peer-to-peer (voir Locus, extension de NFS) : Ficus (UCLA)
- Travail collaboratif : Bayou (Xerox PARC)
- Systèmes décentralisés de gestion de versions (git)
- Sites d'hébergement / partage de fichiers (Dropbox, Google Drive...)

Plan

Principe général

- Principe : fournir des objets partagés par couplage dans les espaces d'adressage de structures d'exécution (couplage virtuel) réparties
- Partage par copie locale (efficacité)
- Programmation simple (accès local)
- Le système charge éventuellement les données à la demande
- Le système assure la cohérence des données partagées

Cohérence locale entre plusieurs objets

Init : `x = 0` $\wedge$ `y = 0`

Processus P1	Processus P2
(1) `x` $\leftarrow$ `x + 1`	`if (x >= y) printf("OK");`
(2) `y` $\leftarrow$ `y + 1`	

Intuitivement, P2 affiche OK. En pratique, pas nécessairement.

Cohérence répartie

Init : `x = 0` $\wedge$ `y = 0`

Processus P1	Processus P2	Processus P3 et P4
`x` $\leftarrow$ `x + 1`	`y` $\leftarrow$ `y + 1`	`if (x > y) print("1 0");` `if (y > x) print("0 1");`

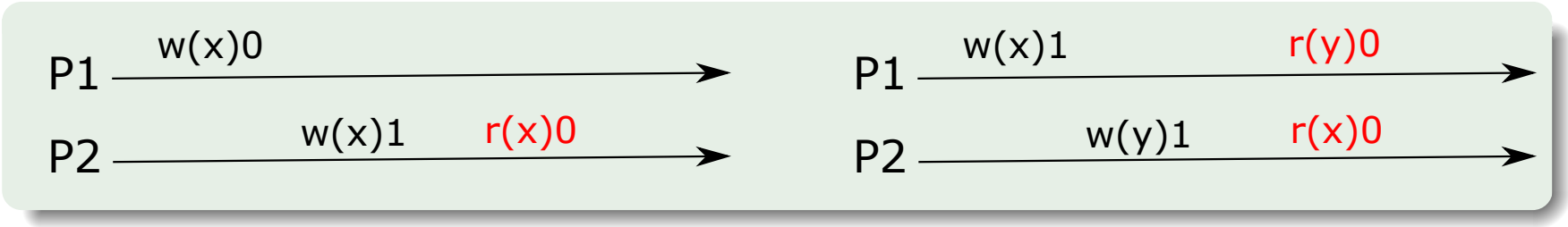
Les processus 3 et 4 peuvent-ils afficher des résultats différents ?

Réplication et répartition des données

Deux sujets :

- Une même donnée, répliquée sur plusieurs sites ⇒ mêmes valeurs ?
- Plusieurs données, sur des sites différents ⇒ l'ensemble est-il cohérent ?

Mais en fait, c'est le même problème !



$w_i(x)b$ = écriture, sur le site i, de la variable x avec la valeur b

Cohérence

Définition

Cohérence : relation que gardent entre elles les différentes copies des données

- Cohérence stricte, linéarisabilité, cohérence séquentielle
- Cohérences faibles (weak consistency)
  - cohérence à la sortie (release consistency)
  - cohérence à l'entrée (entry consistency)
- Cohérences non séquentielles
  - cohérence causale
  - cohérence à terme (eventual consistency)

Réplication optimiste / pessimiste

Réplication optimiste

- Autoriser l'accès à une copie sans synchronisation a priori avec les autres copies
- Modifications propagées en arrière plan
- Conflits supposés peu nombreux, et résolus quand ils sont détectés

Réplication pessimiste

- Garantit l'absence de conflits
- Mécanisme bloquant de prévention

Where a pessimistic algorithm waits, an optimistic one speculates.

Problème 1 : placement des copies

Copies permanentes

Ensemble de copies définies a priori. Configuration statique

⇒ Architecture statique de tolérance aux fautes

Copies à la demande

Création dynamique de copies selon les besoins

⇒ Adaptation à la charge

Problème 2 : propagation des mises à jour

Write-update

- Lecture : copie locale (sans communication)
- Écriture : locale et nouvelle valeur envoyée en multicast
- Cohérence : propriétés sur l'ordre des messages
- Multicast peut être cher (à chaque écriture)

Write-invalidate

- Invalidation diffusée en cas d'écriture
- Lecture $\Rightarrow$ chargement si nécessaire
- Cache $\Rightarrow$ plusieurs lectures/écritures locales sans communication
- Nouvelles valeurs transmises que si nécessaire mais lecture pas nécessairement instantanée

Problème 3 : protocoles de cohérence

Copie primaire fixe

Serveur fixe. Toute demande de modification passe par ce serveur ($\Rightarrow$ accès distant).

Copie primaire locale

Le contrôle est donné au site qui fait une mise à jour $\Rightarrow$ accès local.
Intéressant avec les cohérences liées à la synchronisation.

Duplication active

Accord de l'ensemble des serveurs :

- Diffusion fiable totalement ordonnée (atomique)
- Votes et quorums
- Algorithmes épidémiques

Duplication active – Diffusion

Principe

- Écriture : diffusion atomique totalement ordonnée à toutes les copies
  $\Rightarrow$ toutes les copies mettent à jour dans le même ordre
- Lecture : utilisation d'une copie quelconque :
  - immédiatement (valeur passée possible)
  - causalement (cohérence causale)
  - totalement ordonnée avec les écritures (cohérence séquentielle)

Duplication active – Votes et quorums

Principe

- Pour effectuer une lecture, avoir l'accord de nl copies
- Pour effectuer une écriture, avoir l'accord de ne copies
- Condition : $nl + ne > N$ (nb de copies)
- Garantit la lecture de la plus récente écriture

Votes pondérés

Donner des poids différents aux copies, selon leur importance (p.e. accessibilité)

- N nombre total de votes ($\geq$ nb de copies)
- nl = quorum de lecture
- ne = quorum d'écriture
- Conditions : $nl + ne > N$ et $2 * ne > N$

Conclusion

Intérêt de la réplication

- Favorise les accès locaux : performance
- Tolérance aux pannes (copies multiples)
- Répartition de charge
- Mode déconnecté envisageable en cohérence à terme

Limites

- Diverses formes de cohérence, aucune simple ET performante
- Cohérences faibles $\Rightarrow$ comportements surprenants
- Diverses implantations, toutes complexes
- Difficiles de faire cohabiter plusieurs formes de cohérence selon la nature des données