SYSTEMES de FICHIERS REPARTIS

Un Exemple : NFS

Plan

1.

1. Introduction

2. NFS - Network File System

3. Concepts Généraux

INTRODUCTION

Objectifs d'un système de gestion de fichiers (SGF)

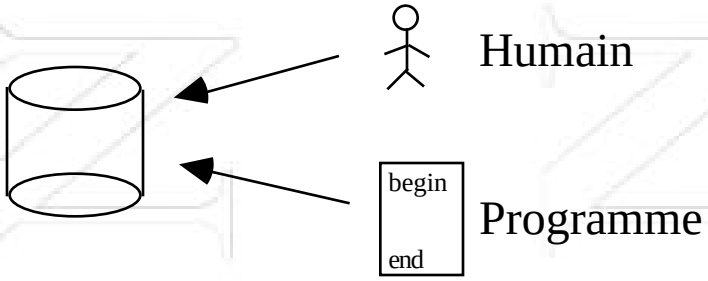
* stockage permanent des informations sous forme de fichiers, **on parle de persistance**

Archive sur : disque, bande (dérouleur, cassette, hexabyte, DAT), disque amovible (optique, disquette) ... **mémoire stable**

Nature du contenu :

. données structurées
. données non structurées
. de natures différentes : exécutable, texte, image, son, vidéo, alphanumériques
. importance du volume des informations

* Utilisateurs :



. Contrôle d'accès,
. Partage de fichiers entre utilisateurs

Modèles d'architectures

La simplicité : micro-ordinateur de type PC/MS-DOS
Un utilisateur -> un seul programme à la fois

Le SGF résoud les 4 problèmes suivants :
- **désignation des fichiers** (par une arborescence souvent)
- **interface d'accès** pour les programmes
- **correspondance nom symbolique-adresse physique**
- **intégrité des données par rapport aux pannes** : alimentation, carte processeur, support disque (grace aux sauvegardes), logiciel

Un peu plus : OS/2 ou Macintosh Système 7
Un utilisateur -> plusieurs programmes (processus) à la fois

Les données peuvent être accédées de façon concurrente, le SGF prend en charge le **contrôle de concurrence**.

Encore un peu plus : Unix
Système temps partagé -> Multi-utilisateurs et Multi-processus

importance de la sécurité et de la protection

Supports d'archivages disque

Machine serveur :
- disques **Redundant Arrays of Inexpensive Disks** (RAID): on utilise une suite de disques dont certains peuvent être le mirroir d'un autre, ou contenir des informations de contrôle pour la détection et la correction d'erreurs

RAID 0* : pas de redondance et pas de contole d'erreur -> rapide mais pas tolérant les pannes
RAID 1* : autant de mirroir que de disque de données -> le plus rapide en lecture, tolère une panne, mais le plus couteux
RAID 2 : utilise plusieurs disques pour stocker le code correcteur et détecteur d'erreurs (Code de Hamming : corrige 1 erreur et en détecte 2)
RAID 3* : les données d'un fichier sont réparties sur l'ensemble des disques de données à l'échelle du bit, la détection d'erreur utilise la parité, un disque seulement consacré au stockage du contrôle d'erreur
RAID 4 : idem RAID3 répartition et parité porte sur le bloc -> les solutions RAID 3 et RAID 4 sont vulnérables si le disque qui contient les informations de parité tombe en panne
RAID 5* : RAID 4 mais avec répartition de l'information de parité sur tous les disques
RAID 6* : RAID 5 avec informations de contrôle d'erreur répartie utilisant une technique P+Q plus efficace

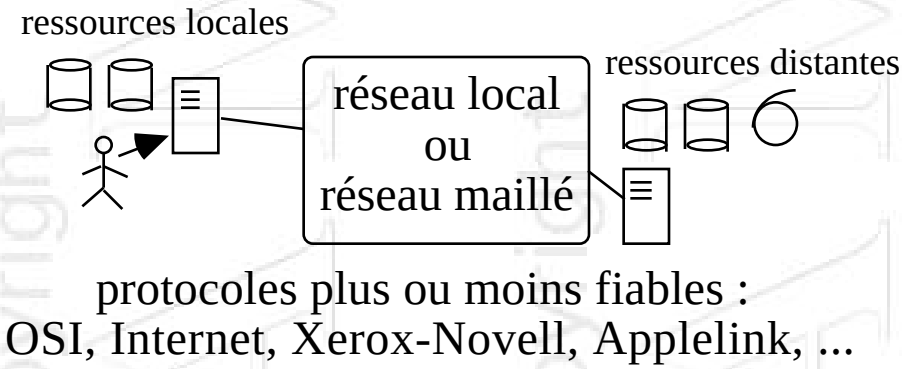
- disques **Write Once Read Many** (WORM)

Gestion des accès :

- Unix File System -UFS (System V) : une partition est répartie sur une suite contigue de secteur
- Fast File System -FFS (BSD) : les informations de gestion d'une partition et des fichiers sont réparties sur plusieurs faces
- Linux File System
- Log Structured File Systems : une partition est organisée comme un journal de log
- Compression : les fichiers avant d'être recopiés sur disques sont compressés

Objectifs d'un système de gestion de fichiers répartis (SGFR)

Extension des propriétés précédentes à un ensemble de machines reliées par un réseau de communication (réseau local, liaisons hertziennes, lignes spécialisées, liaisons modem, NUMERIS).



Les utilisateurs se répartissent sur les machines et ne travaillent pas toujours au même endroit (mobilité de son environnement)

Postes de travail de natures différentes :

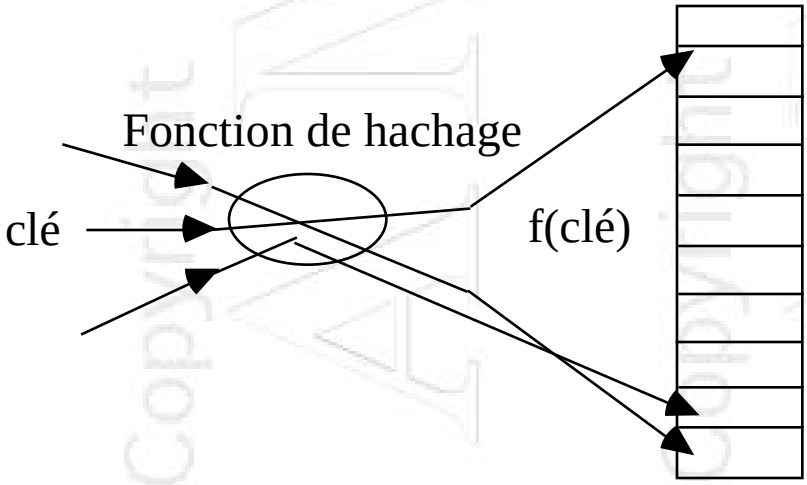
. micro-ordinateur, type PC ou Macintosh
. station de travail
. terminal X (serveur graphique) et calculateur

Rappels : Modèles de fichiers (1)

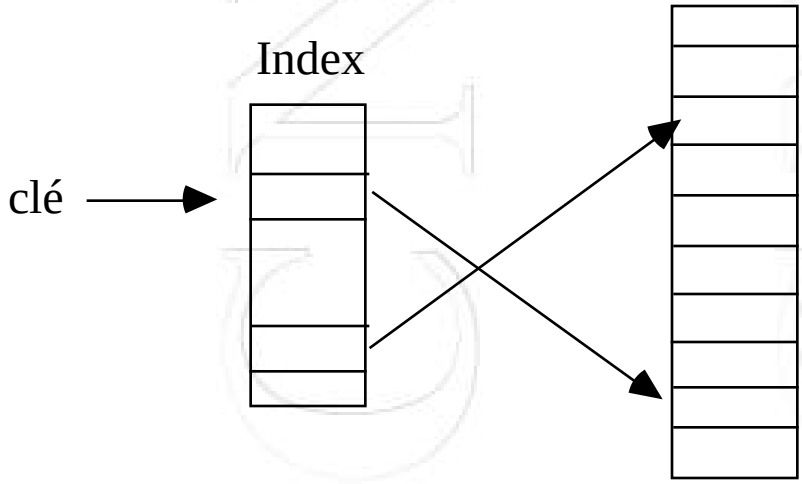
Ce qui distingue les fichiers, c'est leur mode d'accès :

Accès séquentiel : depuis le début

Accès direct par adressage dispersé fonction d'une clé



Accès direct par index sur une clé



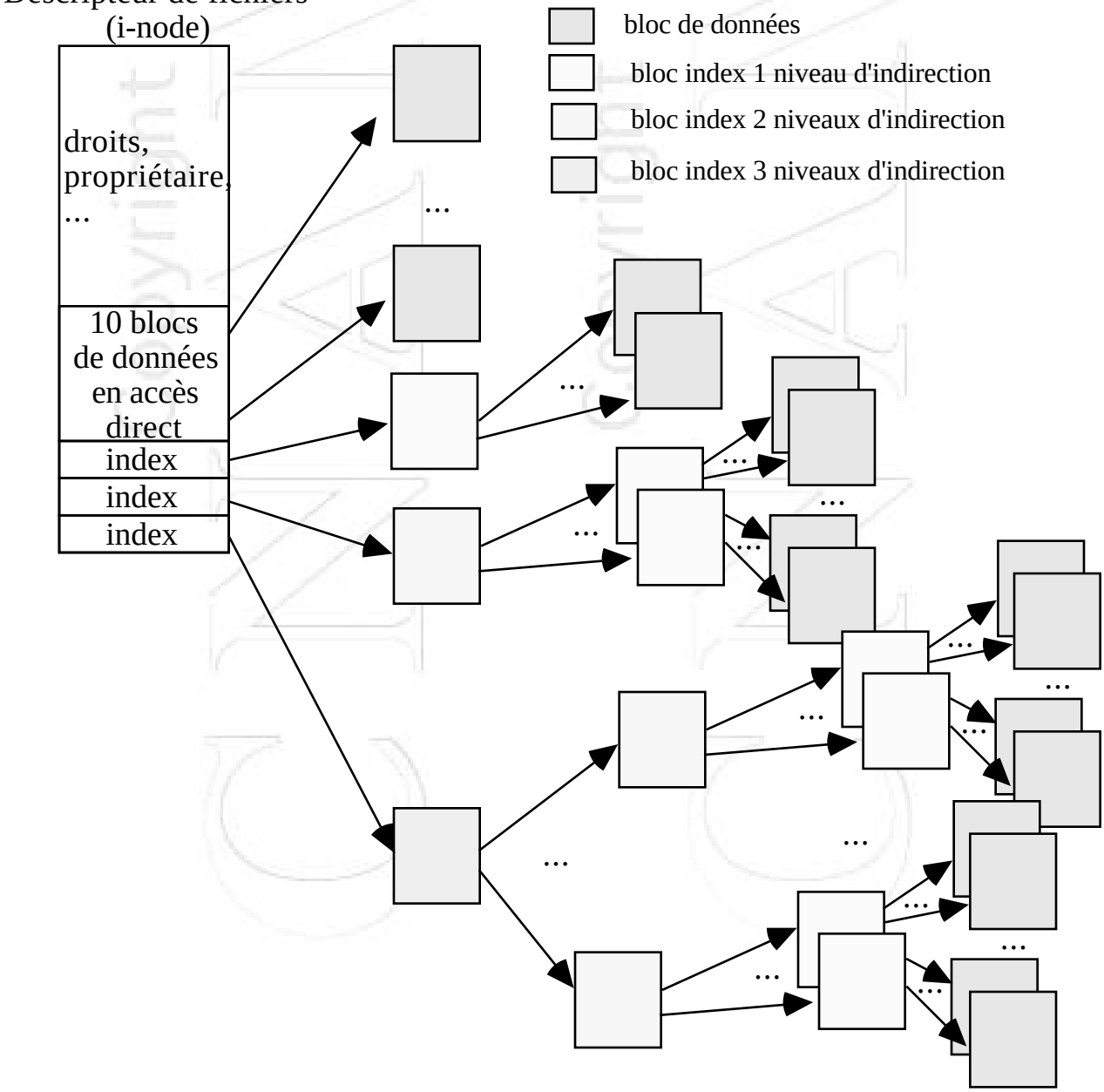
possibilité d'existence de plusieurs index : index primaire et des index secondaires

Rappels : Modèles de fichiers (2)

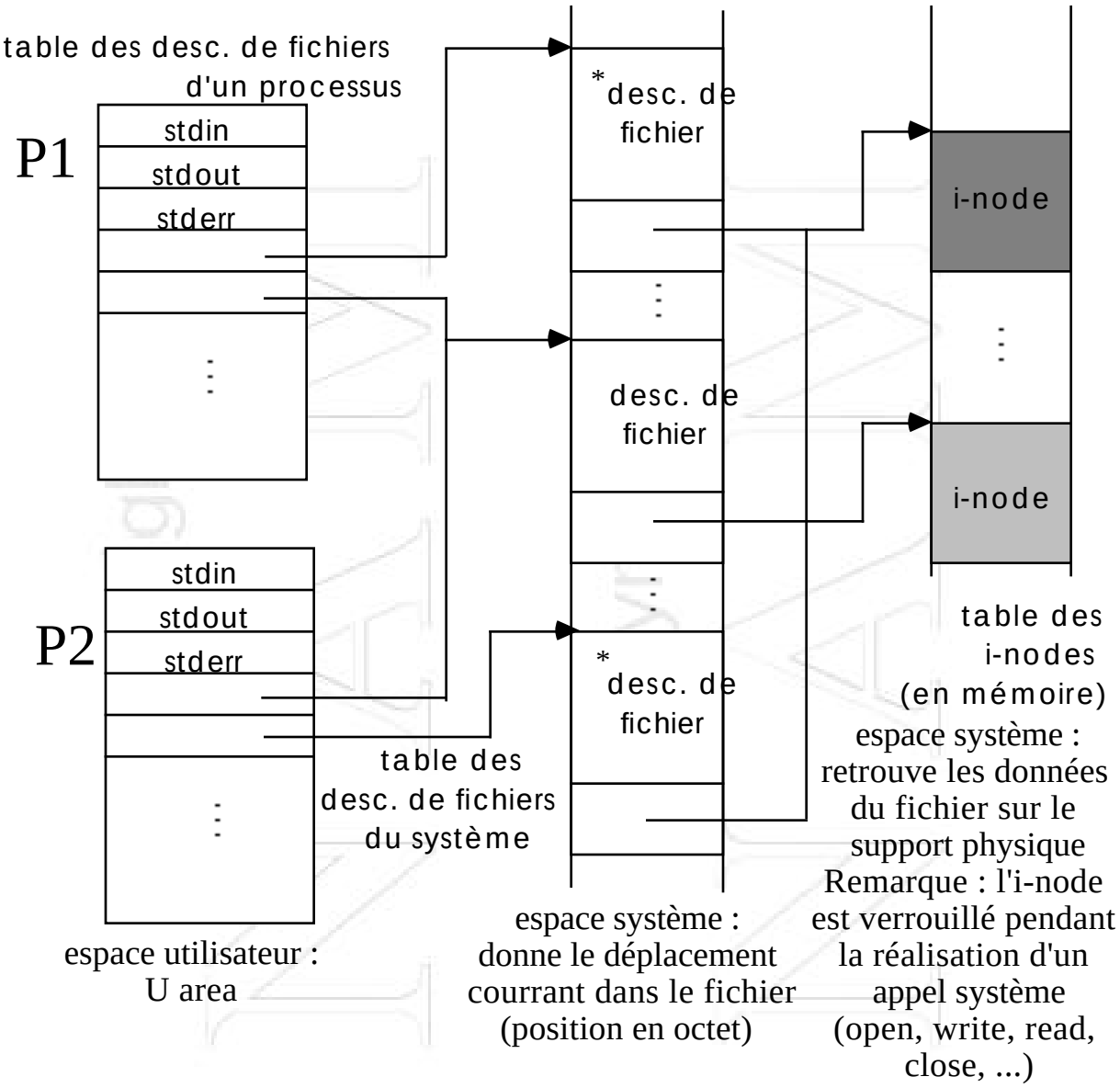
Exemple : Fichiers non structurés, de type Unix ou MS-DOS, les plus répandus

Vue utilisateur :

Vue Système :



Vision des fichiers dans l'environnement d'un processus Unix :



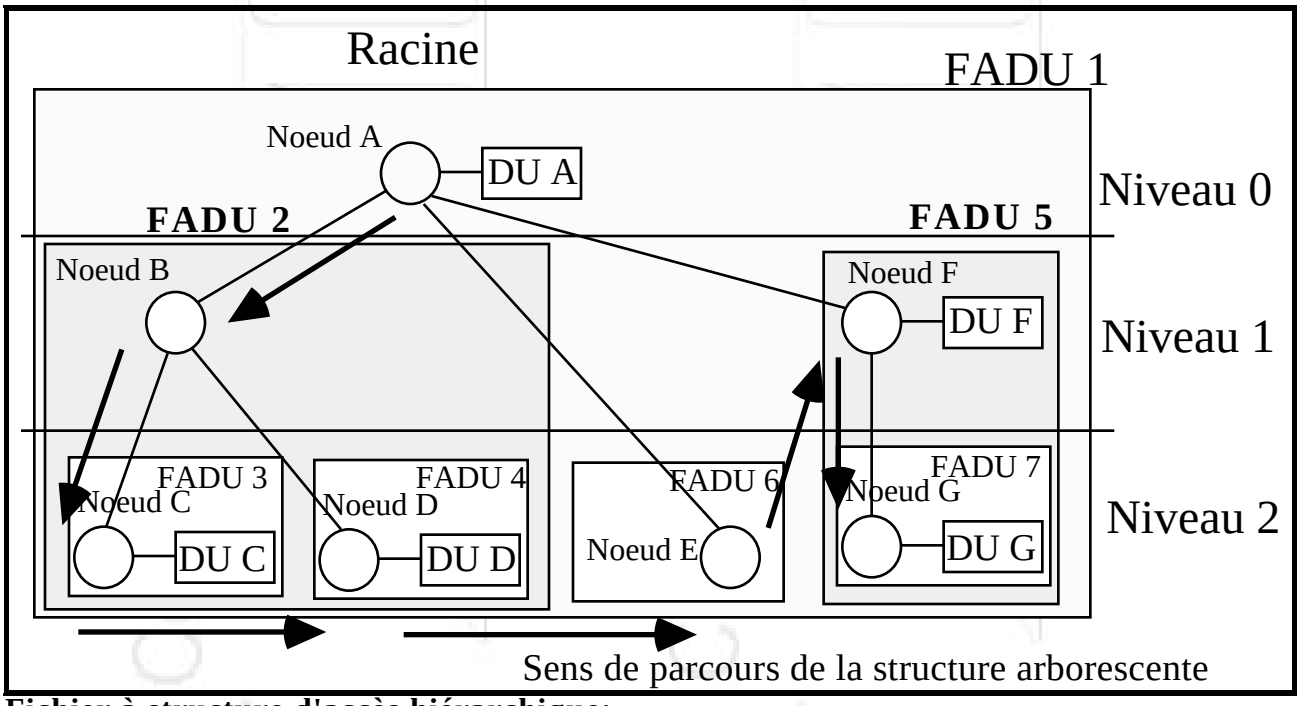
* le même fichier qui possède deux noms, l'un peut être un lien direct (obtenu par la commande ln)

- Tout lecture voit les effets de toutes les écritures précédentes.
- Les processus concurrent qui accèdent partagent le pointeur de déplacement à l'intérieur d'un fichier.

Copyright CNAM

Modèles de fichiers considérés par FTAM[1] pour le transfert de fichiers

Modèle Général d'un fichier virtuel :



Fichier à structure d'accès hiérarchique:

- un fichier est considéré comme une structure arborescente

- c'est un ensemble de noeuds et d'unités de données (DU) regroupés en unités d'accès au fichier (FADU)

- une FADU est un sous arbre de l'arbre complet[2]

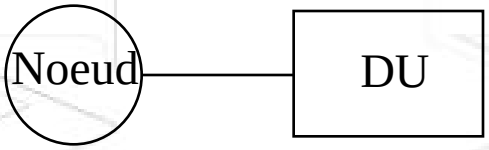
---

[1] FTAM : File Transfert Access Management

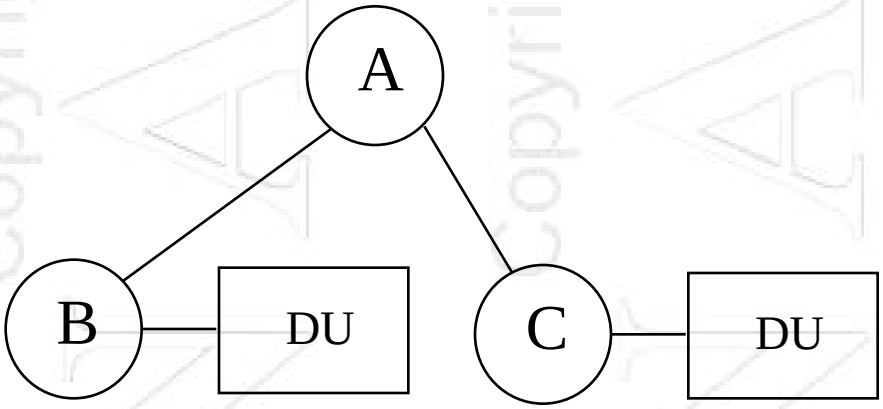
[2] les FADU sont transférées d'un site FTAM à l'autre par envoie des données et des

Exemples :

Fichier virtuel non structuré (type Unix) :



Fichier virtuel séquentiel plat :

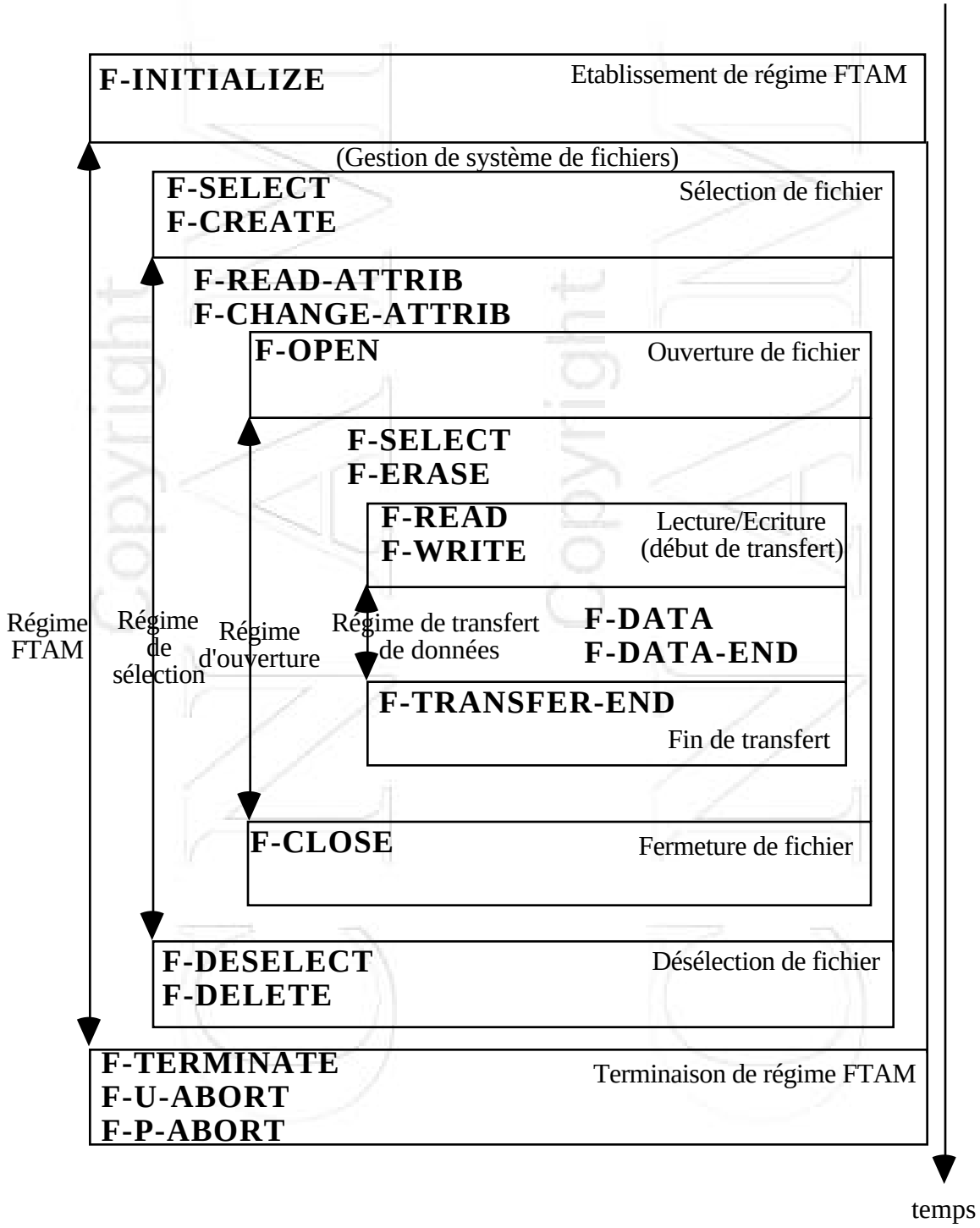


Approches de la répartition de fichiers (1)

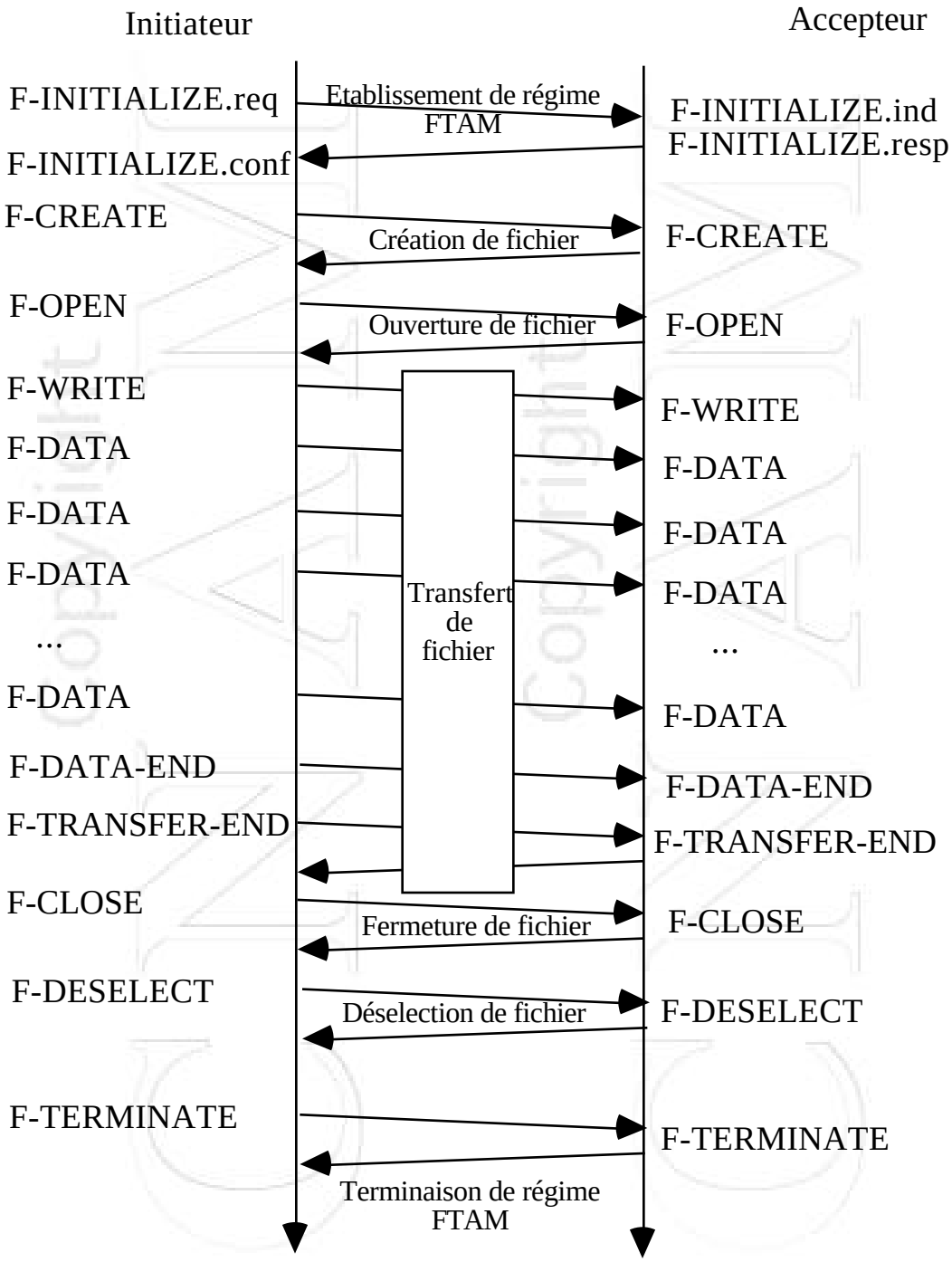
1. **Transfert de fichiers :** Kermit, UUCP, ftp-Internet, ...
-> FTAM en est la version normalisée, un fichier est accessible par programme ou par utilisateur (commande)
- permet la gestion de l'hétérogénéité : de machines, de systèmes d'exploitation, de structures de fichiers, de systèmes de fichiers
- ensemble de primitives de service dont l'enchainement est structuré par des régimes, chaque régime correspond à des opérations particulières.

Les opérations de lecture et d'écriture d'un fichier impliquent tout ou partie du contenu de celui-ci. Elles font référence à la FADU concernée par l'opération et indique en cas d'écriture la spécification de l'action demandée: insertion, remplacement,extension
Le transfert proprement dit est marqué par une suite de primitives F-DATA, qui marquent à chaque fois le transfert d'un segment de données, un F-DATA-END indique la fin du transfert de données.
Il est possible de regrouper certains services en un méta-service, dans ce cas les primitives sont encadrées par F-BEGIN-GROUP et F-END-GROUP qui pourraient être comparées à une "(" et à une ")".

Régimes de service de fichiers FTAM



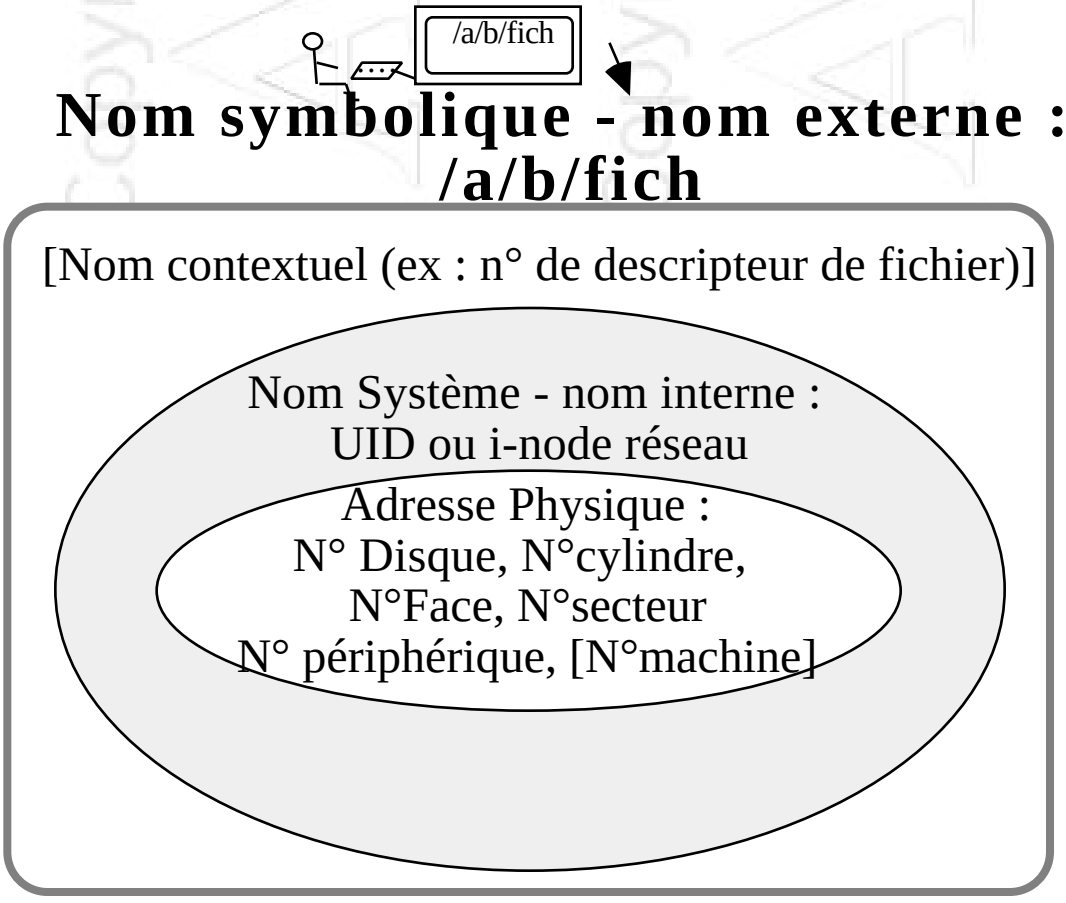
Exemple : Un transfert de fichier



Approches de la répartition de fichiers (2)

2. **Extension du modèle Unix à un réseau de machines :** NFS (Sun)
Apporte : **Performance**, hétérogénéité, modèle Internet, **"transparence"**

3. **Système Réparti :** Chorus (Chorus systèmes), OSF1-Mach (OSF), Amoeba (Tanenbaum)
possède des noms internes uniques et globaux au système (UID)



Apporte : **intégration du système d'exploitation et du système de fichiers**

Approches de la répartition de fichiers (3)

4. **Systèmes à objets répartis :** Commandos (projet Esprit), Guide puis SIRAC (IMAG-Grenoble), SPRING (Sun)

La gestion de la persistance des objets masque l'utilisation d'un système de gestion de fichiers

Apporte : une meilleure adéquation entre la conception et l'implantation d'applications distribuées

5. **Bases de Données Réparties :** R* (IBM, produit jamais commercialisé)

l'accès aux données se fait par des prédicats qui portent sur des données réparties à travers la planète

Apporte : un modèle de données (relations, ...) adapté aux besoins des entreprises de gestion

Critères d'analyse d'un SGFR (1)

TRANSPARENCE :

- Transparence d'accès à travers le réseau :

Mêmes opérations d'accès aux fichiers en local qu'en distant, le transfert explicite des données est complètement invisible à l'utilisateur (humain ou programme)

- Transparence à la localisation des données :

Pas de contraintes dans le choix de la machine de travail, les données sont visibles de partout,

Pas besoin d'indiquer le nom de la machine où elles résident

PERFORMANCE :

Mêmes délais en accès fichier local qu'en accès fichier distant. Pour cet aspect, il faut évaluer :

la surcharge induite sur le réseau par le SGFR
le comportement du SGFR coté demandeur d'accès
le comportement du SGFR coté serveur de fichiers

MAINTENABILITE :

Amène une souplesse dans l'administration du SGFR

EXTENSIBILITE :

Evolution facile :

- Ajouts d'utilisateurs
- Ajouts de sites serveurs/ressources
- Ajouts de sites clients
- Interconnexion de réseaux

Critères d'analyse d'un SGFR (2)

TOLERANCE AUX PANNES :
Fonctionnement du SGFR malgré les pannes avec éventuellement une baisse de performances ou une baisse de fonctionnalité proportionnelle au type de panne subie :

- panne processeur (franche "fail-stop", omission, temporelle, byzantine)
- panne réseau (partitionnement, coupure, ...)
- panne du support (totale, corruption des données, ...)

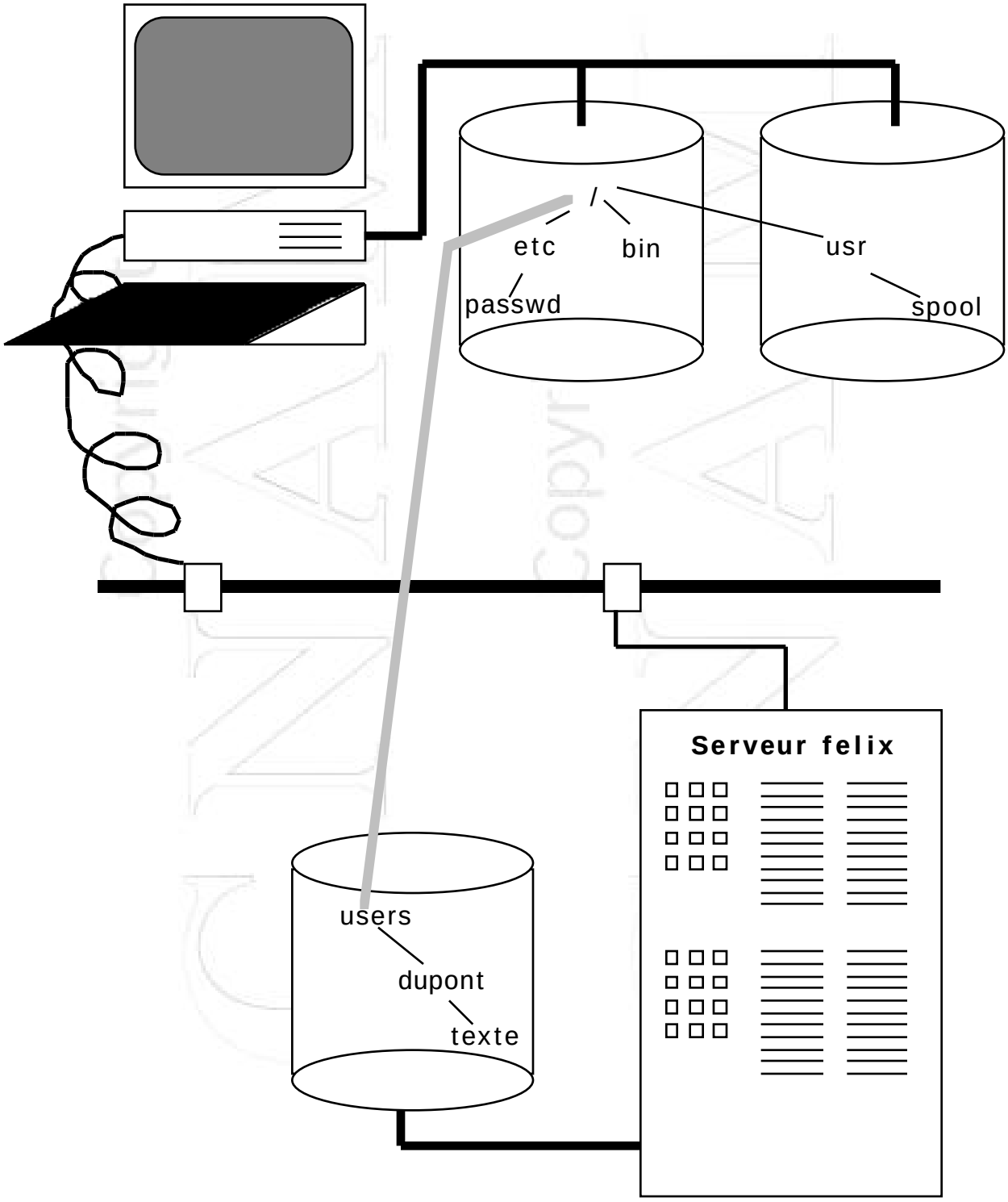
ADAPTABILITE :
Résistance à l'accroissement de la charge due aux utilisateurs et aux programmes qui augmentent.
Evaluer les facteurs d'échelles qui induisent la saturation.

ADAPTABILITE et TOLERANCE AUX PANNES sont liés : une mauvaise réaction à la charge peut amener le processeur d'un serveur à avoir un comportement fautif (panne d'omission, panne temporelle)

NFS - Network File System

Objectifs d'NFS

Partage de fichiers :



Propriétés recherchées

- Partage de fichiers à travers un réseau de machines et de systèmes hétérogènes (VAX-VMS, PC-MSDOS, et de nombreux UNIX-NFS).

- Accès aux fichiers distants masqué aux utilisateurs et aux programmes

- Performances des accès voisines de celles d’un disque local

- Résistance aux pannes : serveur, client et réseau

- Extensibilité du système de fichiers simple

- Facilité d'administration (NIS, Network Information Service, ex YP)

Implantation d’NFS

Point de vue Réseau :

- Le protocole de Transport en **mode datagramme** UDP est utilisé

- Conçu sur le modèle Client/Serveur, à travers un **RPC de sémantique d'exécution au moins une fois**

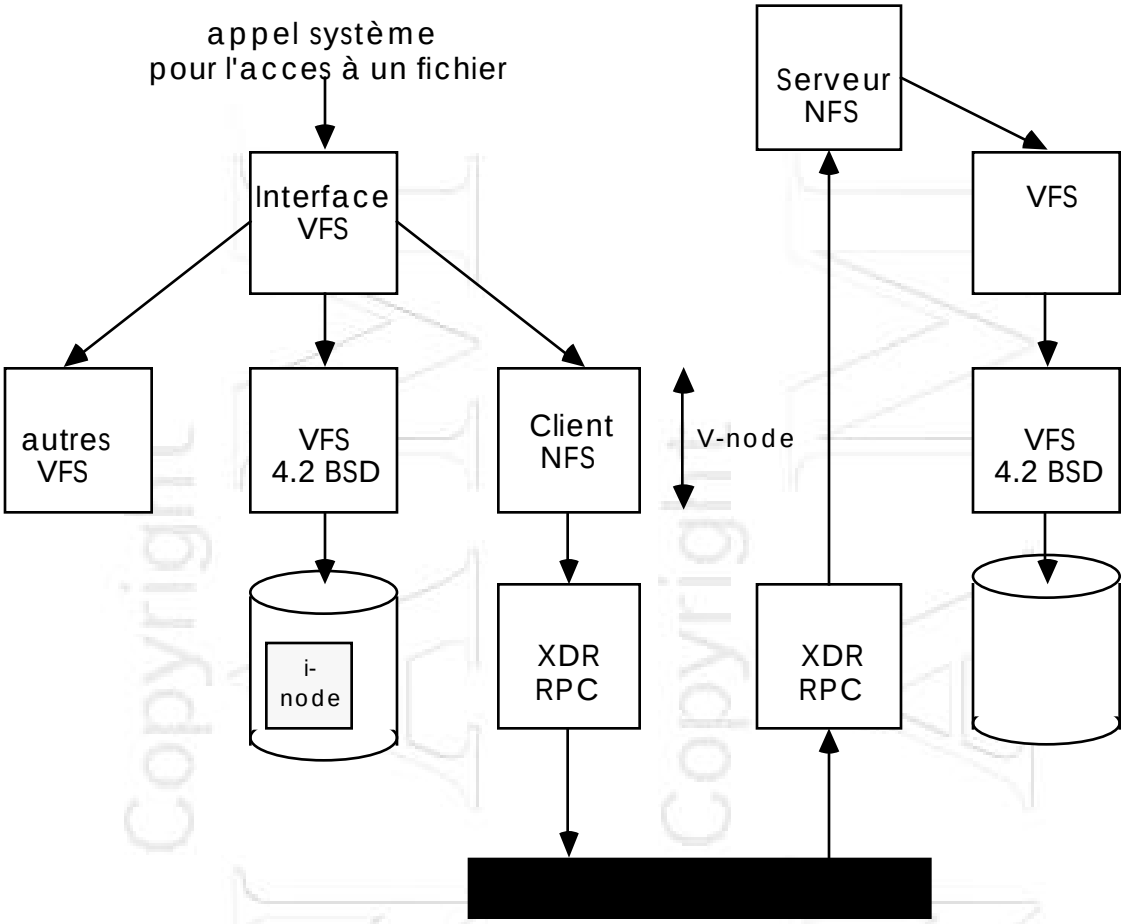
- Gestion de l'hétérogénéité par XDR

Point de vue Système :

- **Réécriture de l’interface d’E/S avec le noyau Unix** : Concept de **"Virtual File System VFS"**, et de **"virtual node V-node"**

- Protocole **NFS** et Serveur sans état

- Protocoles périphériques et Serveur avec état : **Mount**, Montage d'arborescences
  NIS, Gestion des paramètres
  Lock Manager, Gestion des verrous
  Network Status, Surveillance du réseau


appel système
pour l'acces à un fichier
Interface
VFS
autres
VFS
VFS
4.2 BSD
i-
node
Client
NFS
XDR
RPC
V-node
Serveur
NFS
XDR
RPC
VFS
VFS
4.2 BSD

VFS

VFS : Couche logique, interface qui masque l'accès à un sous-arbre local ou distant de l'arborescence des fichiers ou partition ("File System" au sens Unix)

- il est construit sur le concept de V-node
- il gère le montage et le démontage de partitions
- il permet l'accès aux fichiers lors de traversées de points d'attachement ou points de montage ("mount point")

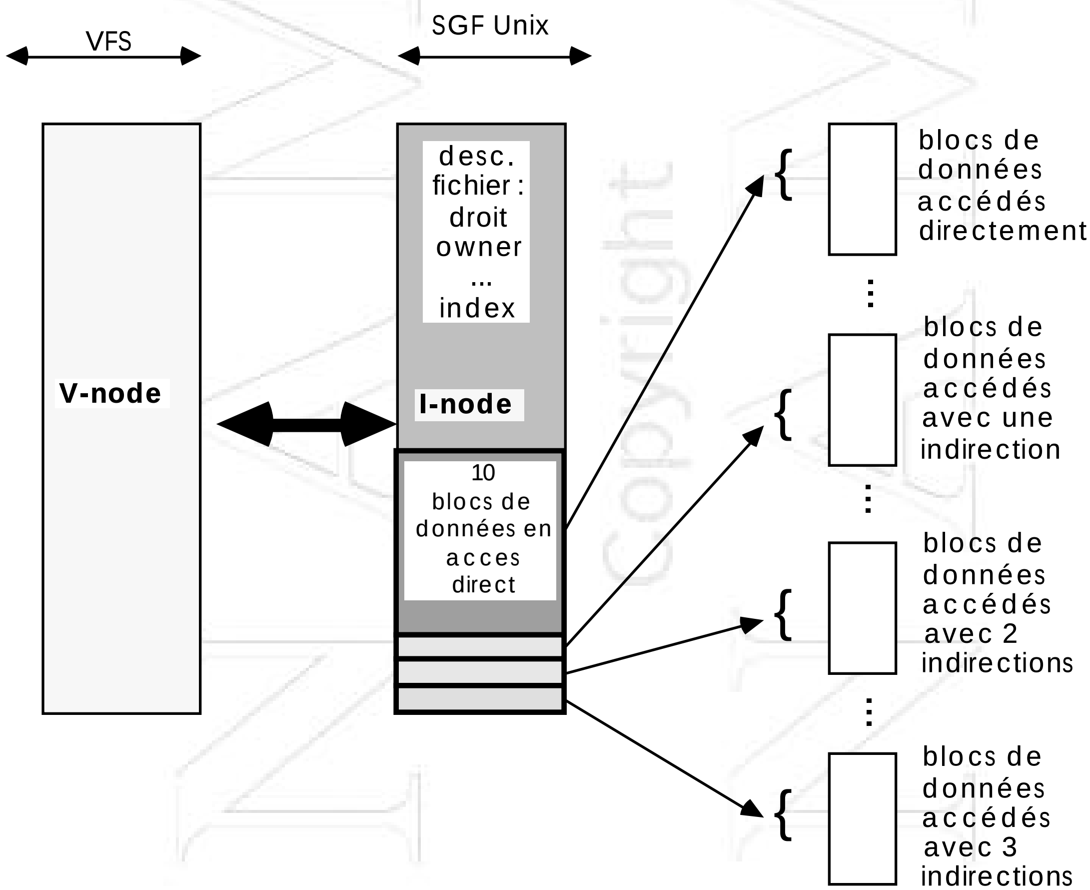
Notion de partition virtuelle

```
struct vfs {
        struct vfs      *vfs_next;              /* next vfs in vfs list */
        struct vfsops   *vfs_op;                /* operations on vfs */
        struct vnode    *vfs_vnodecovered;      /* vnode we mounted on */
        int             vfs_flag;               /* flags */
        int             vfs_bsize;              /* native block size */
        fsid_t          vfs_fsid;               /* file system id */
        u_short         vfs_exroot;             /* exported fs uid 0 mapping */
        short           vfs_exflags;            /* exported fs flags */
        caddr_t         vfs_data;               /* private data */
};

/*¨vfs flags */

#define VFS_RDONLY      0x01            /* read only vfs */
#define VFS_MLOCK       0x02            /* lock vfs so that subtree is stable */
#define VFS_MWAIT       0x04            /* someone is waiting for lock */
#define VFS_NOSUID      0x08            /* someone is waiting for lock */
#define VFS_EXPORTED 0x10               /* file system is exported (NFS) */

struct vfsops {
        int     (*vfs_mount)();                 /* mount file system */
        int     (*vfs_unmount)();               /* unmount file system */
        int     (*vfs_root)();                  /* get root vnode */
        int     (*vfs_statfs)();                /* get fs statistics */
        int     (*vfs_sync)();          /* flush fs buffers */
        int     (*vfs_vget)();                  /* get vnode from fid */
};
```

v-node[3]

v-node : Couche logique, interface qui généralise la notion de descripteur de fichier (i-node Unix), et qui sépare les opérations d'accès à un fichier de leur implantation sur disque.

Notion de descripteur virtuel de fichier



[3] [illegible]

```
struct vnode {
        u_short                 v_flag;         /* vnode flags (see below)*/
        u_short                 v_count;                /* reference count */
        u_short                 v_shlockc;              /* count of shared locks */
        u_short                 v_exlockc;              /* count of exclusive locks */
        struct vfs              *v_vfsmountedhere;  /* ptr to vfs mounted here */
        struct vnodeops         *v_op;          /* vnode operations */
        union {
                struct socket   *v_Socket;              /* unix ipc */
                struct stdata   *v_Stream;              /* stream */
        } v_s;
        struct vfs              *v_vfsp;                /* ptr to vfs we are in */
        enum vtype              v_type;                 /* vnode type */
        dev_t                   v_rdev;                 /* device (VCHR, VBLK) */
        caddr_t                 v_data;         /* private data for fs */
};

/* vnode operations */

struct vnodeops {
        int (*vn_open)();       int (*vn_close)();      int (*vn_rdwr)();
        int (*vn_ioctl)();      int (*vn_select)();     int (*vn_getattr)();
        int (*vn_setattr)();    int (*vn_access)();     int (*vn_lookup)();
        int (*vn_create)();     int (*vn_remove)();     int (*vn_link)();
        int (*vn_rename)();     int (*vn_mkdir)();      int (*vn_rmdir)();
        in (*vn_readdir)();     int (*vn_symlink)();    int (*vn_readlink)();
        in (*vn_fsync)();       int (*vn_inactive)();   int (*vn_bmap)();
        in (*vn_strategy)();    int (*vn_bread)();      int (*vn_brelse)();
        in (*vn_lockctl)();     int (*vn_fid)();
};

/ * Vnode attributes.  A field value of -1 represents a field whose value is unavailable (getattr) or
which is not to be changed (setattr) */

struct vattr {
        enum vtype      va_type;        /* vnode type (for create) */
        u_short         va_mode;        /* files access mode and type */
        short           va_uid;         /* owner user id */
        short           va_gid;         /* owner group id */
        long            va_fsid;        /* file system id (dev for now) */
        long            va_nodeid;      /* node id */
        short           va_nlink;       /* number of references to file */
        u_longva_size;          /* file size in bytes (quad?) */
        long            va_blocksize;  /* blocksize preferred for i/o */
        struct timeval  va_atime;       /* time of last access */
        struct timeval  va_mtime;       /* time of last modification */
        struct timeval  va_ctime;       /* time file ``created */
        dev_t           va_rdev;        /* device the file represents */
        long            va_blocks;      /* kbytes of disk space held by file */
};
```

Le Montage de fichiers distants (1)

Rappel du Montage d'un fichier sur un disque local

Notion de file system :

- Vue "administration système" :

Partition logique du disque physique qui contient une sous-arborescence des fichiers du système

La partition système est privilégiée parmi les autres. Elle contient les fichiers vitaux du système d'exploitation (souvent la première partition d'un disque, par exemple sd0a)

Une autre partition a un rôle important, la partition qui est liée à la pagination et au swapping (souvent la deuxième partition sur plusieurs disques , par exemple sd0b et sd1b)

- Vue implantation :

Un file system est désigné par un n° de périphérique logique et contient :

- Boot bloc
- Des informations de gestion, super-bloc
- Une suite d' i-node[4], descripteurs de fichiers ou de répertoires
- Une suite de blocs de données

Attachement d'une partition :

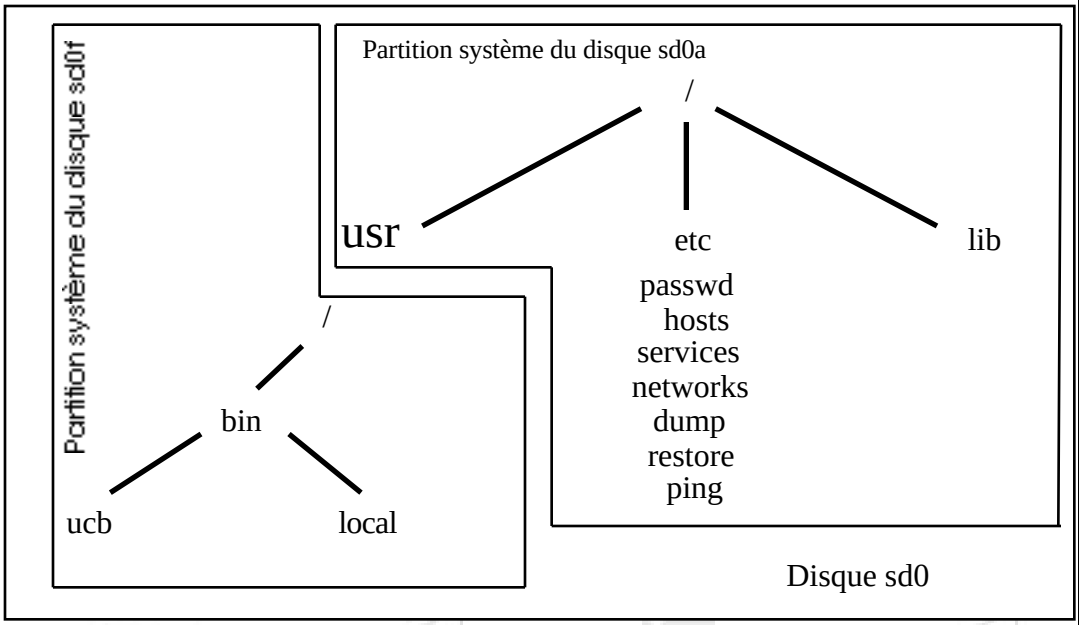
Association d'une partition logique d'un disque à un noeud de l'arborescence des fichiers d'une machine

Association d'un i-node (qui correspond au noeud d'attachement) de la *table des i-node résidente en mémoire* avec un super-bloc, cet i-node est marqué comme point de montage. Cette association ne se fait qu'en mémoire centrale, jamais sur disque

---

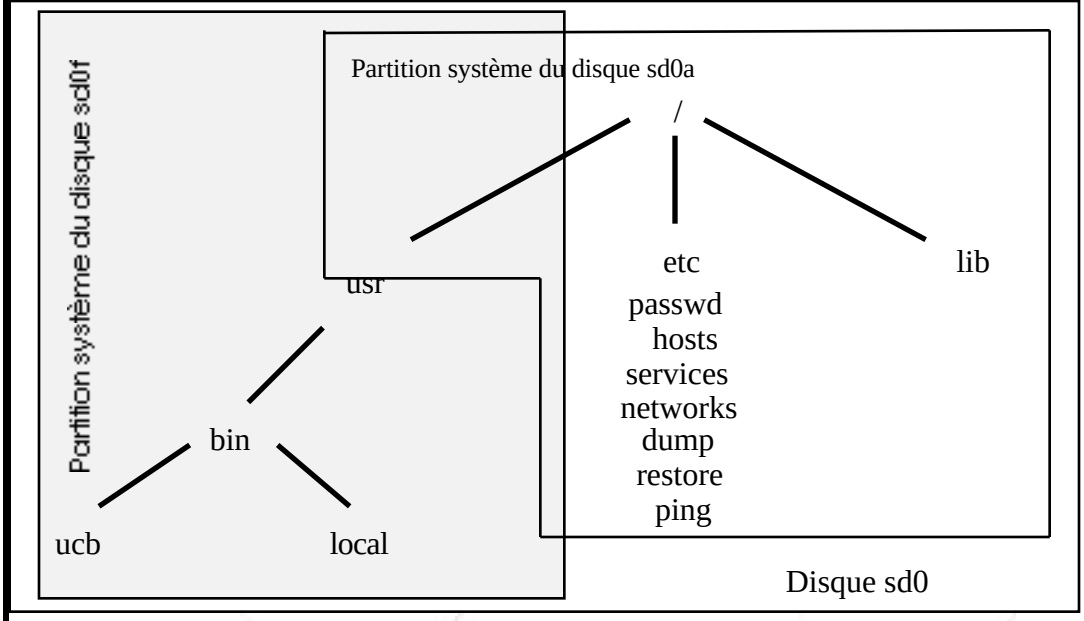
[4] Un i-node a un numéro unique dans un file system

L'inode # 2 correspond à la racine de la partie d'arborescence contenue dans la partition



Avant montage de la partition sd0f :

% cd /usr ; ls -l
%



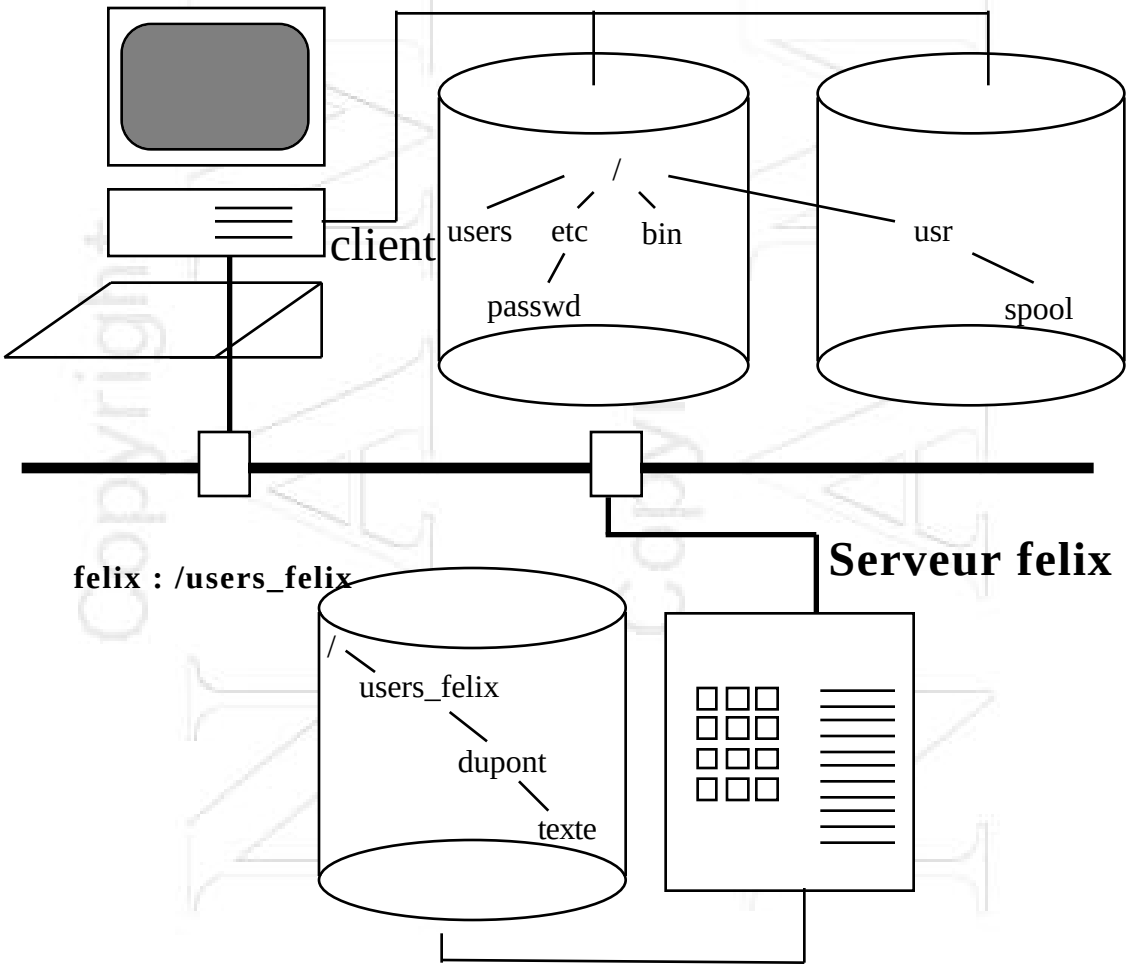
Après montage de la partition sd0f :

%mount /dev/rsd0f /usr
% cd /usr ; ls -l
drwxr-xr-x .... bin .
%

Le Montage de fichiers distants (2)

Extension du mécanisme de montage à travers le réseau :

Vue de l'arborescence des fichiers avant le montage d'un répertoire distant sur le client :

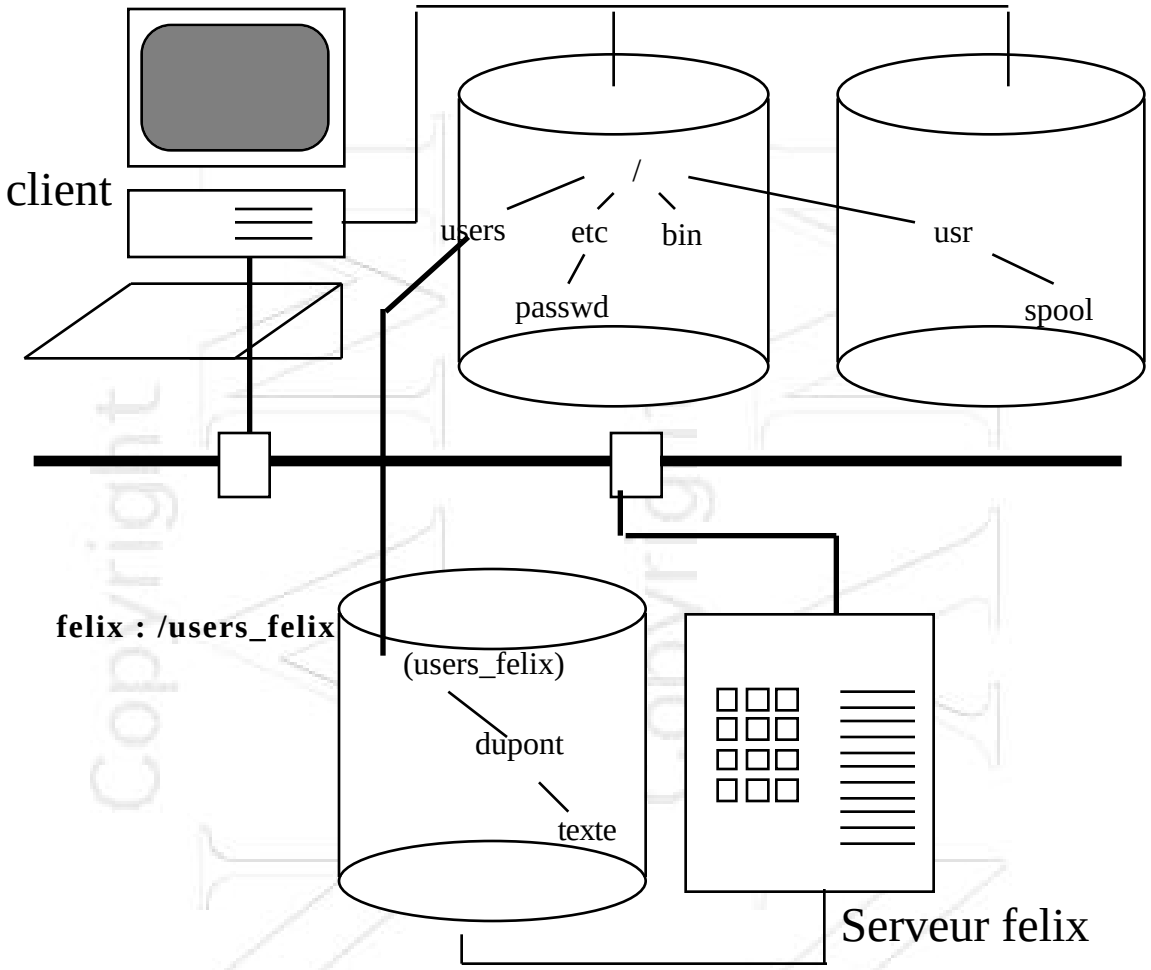


La liste des répertoires attachables exportés par le serveur est décrit dans le fichier /etc/exports : < nom de répertoire, liste d'accès des utilisateurs et machines autorisées >

```
/usr            -access=jo:dan:athena
/home/server    -access=iris
/users_felix    -access=client
```

Remarque : Le nom d'un répertoire à attacher ne correspond pas nécessairement à une sous-arborescence complète et donc à une partition locale du serveur.

Vue de l'arborescence des fichiers après l'attachement d'un réperoire distant sur le client : **commande mount felix:/users_felix /users**



Problème de désignation des fichiers :

Avec ce système d'attachement, les chemins d'accès aux fichiers peuvent différer d'un client à l'autre, les clients n'ont pas la même vue logique de l'arborescence complète, ceci amène à rechercher des noms uniques et globaux

->

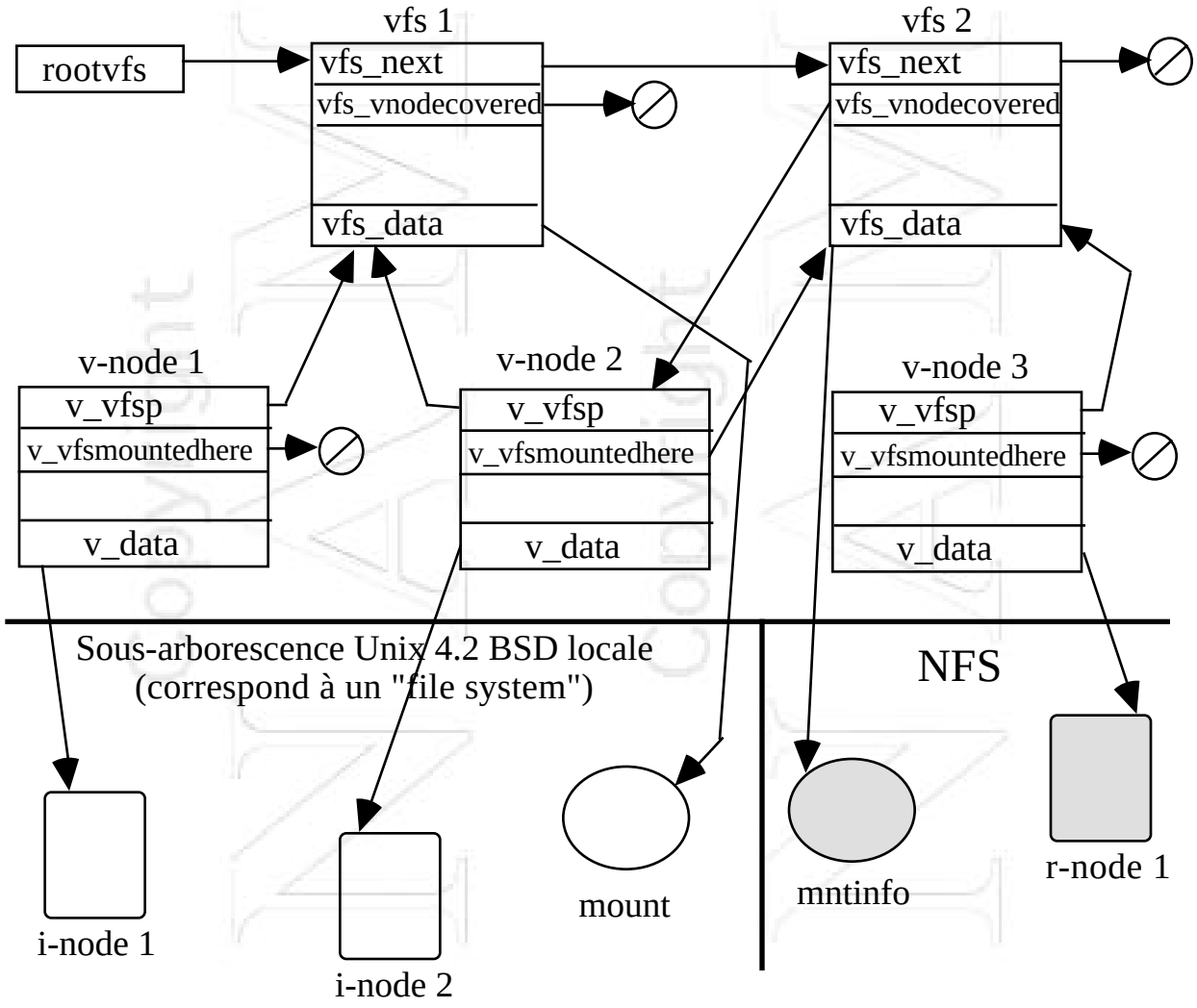
Les arborescences montées sont préfixées par un nom qui masque le serveur dans un répertoire dédié au montage de systèmes de fichiers distants.

=> **Noms administrativement globaux** dans NFS[5]

---

[5] Les noms administrativement globaux sont masqués à l'utilisateur par des **liens symboliques** qui rendent les noms des fichiers indépendants de leur localisation sur

Relation VFS-VNODE- SGF local



Poignée - File Handle

Identificateur de fichier sur le serveur:
- pour le client : 32 octets qu'il n'interprète pas (opaque XDR)
- pour le serveur : 32 octets dans lesquels il met les informations nécessaires à retrouver un fichier qu'il gère, ou le descripteur de celui-ci

Dans les premières versions de NFS, il était d'une longueur de 32 octets. Maintenant, il est de taille variable : <longueur,tableau>, dans NFS V3.

Pour un serveur Unix, le file handle est :

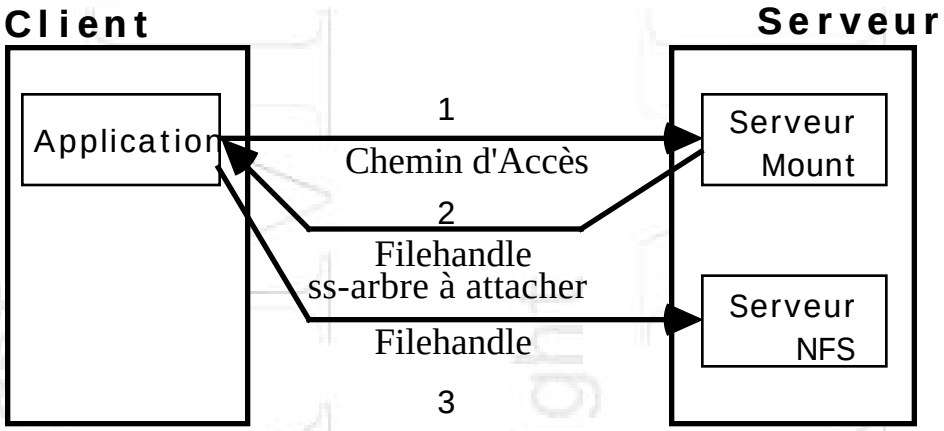
<n° de filesystem (8o), n° d'i-node (4o), n° de génération (4o), remplissage (18o)>

struct svcfh dans nfs.h

Le numéro de génération fonctionne comme un numéro d'époque. Si le fichier correspondant à un i-node a été détruit, puis qu'il a été réattribué lors de la création d'un fichier, le client risque de présenter un n° d'i-node obsolete et de recevoir des données du nouveau fichier. Le serveur détecte cette situation grace à ce numéro.

Protocole Mount (1)

Le Protocole MOUNT, c’est un peu plus :



Le protocole "mount" résoud la correspondance entre un nom de répertoire : chemin d'accès sous forme de chaine de caractère qui représente un point d'attachement, et sa localisation sur le serveur cible.
=> rend NFS indépendant des conventions de désignation d’un répertoire (point d'attachement) sur un serveur

1. Le client envoie le nom du répertoire à attacher au serveur MOUNT

2. Le serveur "mount" lui retourne la **"poignée"** pour pouvoir entrer dans le répertoire d'attachement après avoir vérifié les droits d’accès. C'est une première poignée, et le client la conserve soigneusement sinon, il ne pourra plus accéder au sous-arbre correspondant.

3. Le client, lors d'une opération d'accès à un fichier, envoie au le serveur NFS le file handle qui lui a été retourné pour parcourir le chemin d’accès au fichier qui passe par le point d'attachement.

Protocole Mount (2)

Le protocole mount est réalisé par un ensemble de procédures RPC Sun :

0	NULL	ne fait rien
1	MOUNT[6]	retourne le file handle qui correspond au point d'attachement d'un sous-arbre
2	READMOUNT	retourne la liste des sous-arbres attachés
3	UNMOUNT	enlève un fichier de la liste des sous arbres attachés
4	UNMOUNTALL	vide la liste des sous-arbres attachés
5	READEXPORT	retourne la liste des sous-arbres exportés

[6] Read le numéro de file handle d'un répertoire à monter

Comment se passe le montage de serveurs en cascade sur NFS :

cas 1 :

client :	mount serveur1:/usr/local	/usr/local
serveur1 :	mount serveur2:/usr/local/work	/usr/local/work

cas 2 :

client :	mount serveur1:/usr/local	/usr/local
client :	mount serveur2:/usr/local/work	/usr/local/work

Donnez le résultat.

Le protocole NFS (1)

1. Le **serveur NFS est sans état**. Il ne maintient aucune information sur les fichiers qu'il gère pour le compte d'un client. C'est le client qui conserve toutes les informations qui permettent au serveur de retrouver le fichier, elles sont dans le **filehandle**.

L'open() sur un fichier ne laisse donc aucune trace sur le serveur!
Il ne va même pas jusqu'au serveur, seul le parcours du chemin d'accès au fichier demandé provoque des échanges (procédure lookup)

2. Résistance aux pannes : Quand un serveur tombe en panne, les clients restent bloqués jusqu'à la remise en route du serveur (mécanisme de RPC avec temporisation 1100s entre deux tentatives et 4 tentatives avant un message d'erreur).

"Server not responding. Still trying."

Puis tout reprend comme si rien ne s'était passé ...

Le protocole NFS (2)

3. idempotence du service : La sémantique du RPC[7] choisie par Sun (au moins une fois) implique l'idempotence de toutes les opérations d'accès aux fichiers, par exemple l'écriture séquentielle doit être transformée en écriture en accès direct ...

Toutes les opérations ne sont pas idempotentes[8] !!!

4. Cumul possible du rôle de serveur et du rôle de client

[7] Les appels de procédures RPC sont tous synchrones. C'est à dire bloquants pour ceux qui l'effectue

[8] Opérations de destruction par exemple

Quelques procédures du protocole NFS

Le protocole NFS est lui aussi réalisé à l'aide de procédure de type RPC Sun :

1 GETATTR (fhandle,attr)
rend les attributs d’un fichier

4 LOOKUP(dir,name,file,attr)
rend un fhandle file et les attributs attr du fichier name dans le répertoire de filehandle dir

6 READFILE(file,offset,count,attr,data)
rend count octets de données du fichier de filehandle file dans les données opaques data, à partir de la position offset dans ce fichier, les attributs attr du fichier sont retournés

8 WRITE(file,offset,data)
écrit dans le fichier de file handle file à partir de la position offset, les données opaques data transmises, l’opération write est atomique

9 CREATE(dir,name,satt,file,attr)
crée un fichier de nom name dans le répertoire de file handle dir avec les attributs satt, le file handle file, et les attributs du fichier créé sont retournés

parmi les paramètres, on peut trouver un "magic cookie" : indicateur sur de qui a déjà été lu dans un répertoire par le client, seul le serveur est capable d’interpréter cette information.

Précautions :

Les identificateurs d’utilisateurs et de groupes d’utilisateurs entre machines clientes et serveurs doivent être identiques sinon, il risque d’y avoir des problèmes de propriétés et de droits d’accès !!! L’utilisation de NIS est recommandée pour gérer cette homogénéité !!!

Les accès concurrents ne sont pas supportés par NFS, c’est le dernier qui écrit qui a gagné !!!

Synchronization des horloges sur les différentes machines pour que la commande make fonctionne correctement.

Mecanisme de cache - biod (1)

Les clients NFS peuvent utiliser une technique de cache pour améliorer les performances

-> Le **cache réside en mémoire centrale du client**, le disque local s'il existe n'est pas impliqué dans la gestion de cache

-> Les informations mises dans le cache sont :
- page contenu de fichier
- page contenu de répertoire
- descripteur de fichier

Mecanisme de cache - biod (2)

Problème de validation du cache en mémoire centrale :

Les pages d'un fichier sont estampillées avec leur date de dernière modification. Il y a validation par comparaison de cette date avec celle qui réside avec le fichier sur le serveur, s'il y a une différence, la page doit être rechargée.

Quand cette comparaison peut-elle s'effectuer ?

A l'initiative du client

Pour un fichier :
. A chaque ouverture de fichier
. A chaque défaut de page dans le cache
Pour un répertoire :
. A chaque ouverture de fichier, il y a contrôle de validité pour le répertoire qui le contient
. Les répertoires sont conservés dans le cache en lecture seulement, les modifications sont opérées sur le serveur directement

Validation garantie périodiquement sinon :

toutes les 3s pour un fichier
toutes les 30s pour un répertoire

Mecanisme de cache - biod (3)

Mise à jour du serveur :

Une page modifiée est marquée "sale" et devra être transférée vers le serveur.

Cette activité est réalisée de façon asynchrone par le noyau ... quand il a le temps.

Garantie :
Toutes les pages modifées seront recopiées sur le serveur au plus tard avant la fermeture du fichier (technique "write-on-close")

La gestion de la concurrence est sous la responsabilité de l'utilisateur ... à défaut, c'est le dernier qui écrit qui a gagné ... attention aux risques d'incohérence.

Transfert serveur-client :
. par blocs de 8 Ko
. tout le fichier pour les petits fichiers moins grands que 8 Ko

Réplication - "Automounter"

Pour un client, ce mécanisme permet d'avoir plusieurs serveurs pour réaliser un attachement de sous-arbre. **C'est une forme de réplication de serveurs**.

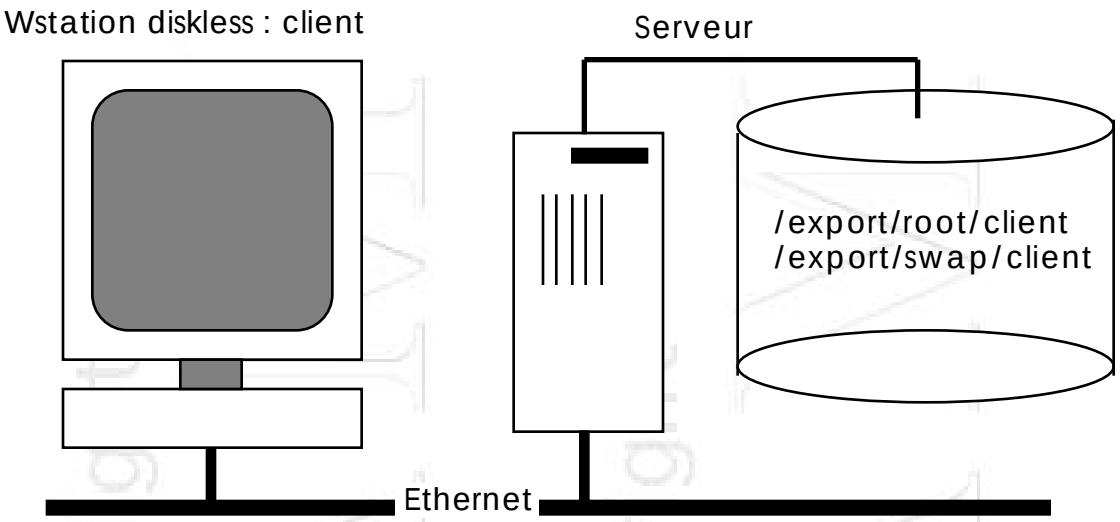
C'est le premier serveur qui répond au client demandeur qui est utilisé.

Pratique pour les fichiers exécutables et les fichiers de données accédés en lecture => TOLÉRANCE AUX PANNES pour ces fichiers.

Pour les fichiers modifiés, la propagation des écritures d'un serveur à un autre doit être faite "à la main".

Copyright CNAM

Les Stations sans disque



Les stations sans disque utilisent NFS pour leur partition système (/vmunix, les fichiers de configuration, et les programmes du système fondamentaux), pour l'accès aux fichiers généraux (bibliothèques, compilateurs, commandes, espace utilisateur), et aussi pour leur partition de swap.

Rien ne s'oppose à ce que l'espace swap soit vu comme un large fichier en accès direct sur un disque distant.

Les Yellow Pages - NIS (1)

Principes :

- Les “Yellow Pages” forment une base de données dupliquée qui fait correspondre une “map” à un fichier[9] :

Fichier des utilisateurs	/etc/passwd
Fichier des sites connectés	/etc/hosts
Fichier des services	/etc/services
.....	

Elles permettent d’avoir une définition unique de certains paramètres de configuration du réseau de machines. Un réseau de machines qui utilisent la même copie de la base YP, définit un domaine d’administration YP.

- Deux types de machines :
Les serveurs qui possèdent une copie de la base de données (ypserv) :

. Le serveur maître sur lesquelles sont effectuées les mises à jour et qui effectue la propagation de ces mises à jour

[9] Il est possible, pour certaines “map”, de consulter le fichier local avant de faire une requête à un serveur YP. C’est le cas, pour le fichier des utilisateurs.

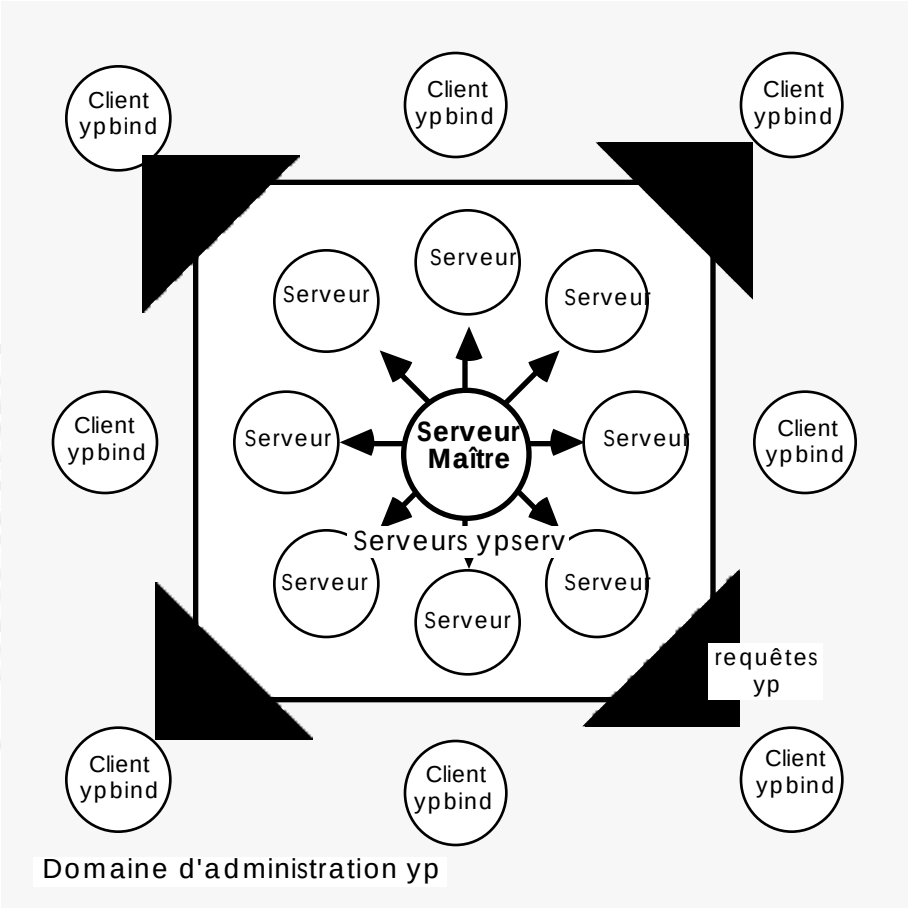
. Les serveurs esclaves => résistance aux pannes, récupération des mises à jours

Les clients qui interrogent un serveur pour accéder aux informations de la base de données (ypbind), pour connaître le serveur associé, utiliser la commande ypwhich.

- Les YP sont construites au-dessus de RPC/XDR.

Copyright CNAM

Les Yellow Pages - NIS (2)



Les Yellow Pages - NIS (3)

- Cohérence faible des données entre serveur maître et serveurs esclaves, la mise à jour d'un mot de passe par exemple n'est prise en compte sur tous les serveurs qu'au moment de la prochaine mise à jour des "map" sur les serveurs esclaves.

- Lors de la panne d'un serveur , un client joint un nouveau serveur par une requête de diffusion, le premier qui a répondu devient son nouveau serveur.

Quand le serveur maître tombe en panne, plus aucune mise à jour n'est possible....

- Fonctionne bien entre machines homogènes, mal entre machines hétérogènes :

"tables" différentes, protocoles différents ???

Reste à voir en périphérie d'NFS:
. le Lock Manager => accès concurrent aux fichiers
. le Network Status Monitor => gestion de l'état des différentes machines du réseau[10]

[10] Serveur avec état qui complète tout ce que ne peut pas maintenir NFS pour rester

Concepts Généraux

. Désignation et Transparence

. Sémantique du partage de fichiers

. Méthode d'accès à distance

. Tolérance aux pannes

. Extensibilité

Désignation

Comment passe-t-on d'un **nom symbolique** qui repère un fichier dans une **organisation logique** à l'**objet physique** qui le contient ?

La désignation gère la correspondance :

Nom Symbolique

↓

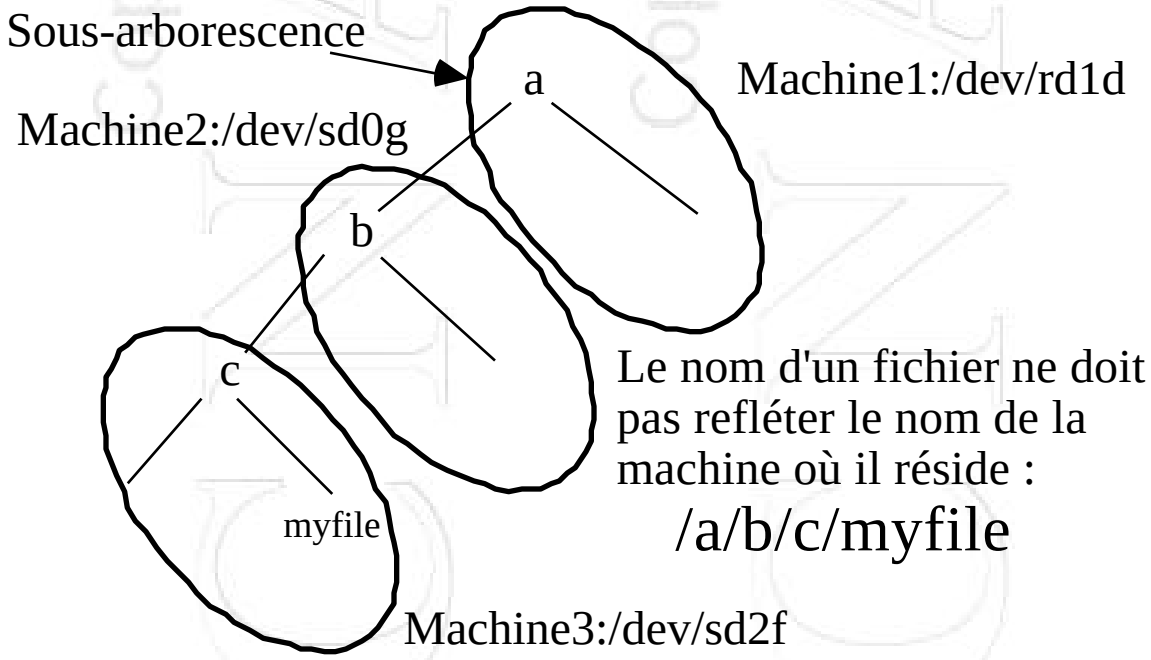
Nom interne

↓

Nom Physique

Modèle Unix est souvent retenu comme modèle de désignation symbolique:

ensemble de sous-arborescences connectées entre elles



Le nom d'un fichier ne doit pas refléter le nom de la machine où il réside :

/a/b/c/myfile

Localisation - transparence

La transparence, c'est pouvoir masquer l'endroit (la machine) où est archivé un fichier et le moyen d'y accéder (protocole d'accès).

Les fichiers distants sont désignés comme s'ils étaient locaux.

Cette propriété est particulièrement importante quand le SGFR s'appuie sur des copies multiples pour la TOLERANCE AUX PANNES.

Pour réaliser la transparence, on conserve dans une table une correspondance entre une sous-arborescence et la machine où elle réside. Cette correspondance est invisible aux utilisateurs humains comme programmes.

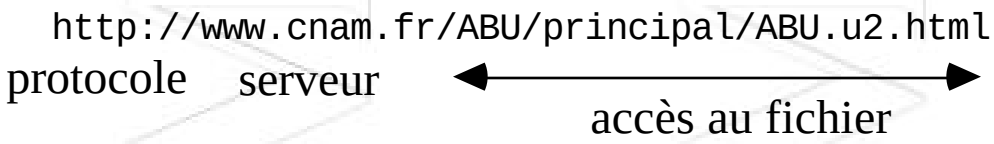
Tout le monde fournit cette propriété sauf Newcastle Connexion qui introduit une syntaxe d'exception :
- super-racine ou racine réseau : //
- le nom de serveur préfixe toute arborescence locale d'une machine

Problème : pas transparent, perte de portabilité
changement de serveur => changement de nom dans les programmes

Cette propriété souvent recherchée semble abandonnée dans certaines circonstances (World Wide Applications).

Le Web est un exemple de cette démarche à contre courant. C'est un environnement qui permet d'accéder à des informations maintenues sur les sites Web de l'Internet.

Les noms des fichiers accessibles par ce moyen sont appelés URL[11] :



Les facteurs qui ont présidés ce choix semblent être :

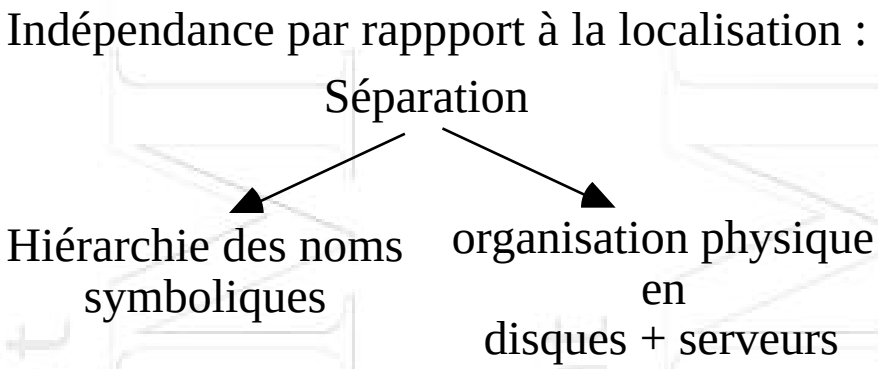
- le très grand nombre de serveurs implique une désignation uniforme même au prix d'une syntaxe d'exception qui provoque la perte de la transparence

- la nécessité d'une uniformisation de l'accès aux données malgré la diversité des protocoles de communication pour retirer ces données

[11] Uniform Resource Locator

Localisation - indépendance

L'indépendance est une propriété beaucoup plus forte que la transparence.



Elle permet la migration des fichiers car leur nom n'a pas besoin de changer.

Propriété utile pour faire de la répartition de charge en faisant migrer une sous-arborescence vers un serveur moins chargé.

Propriétés visées :

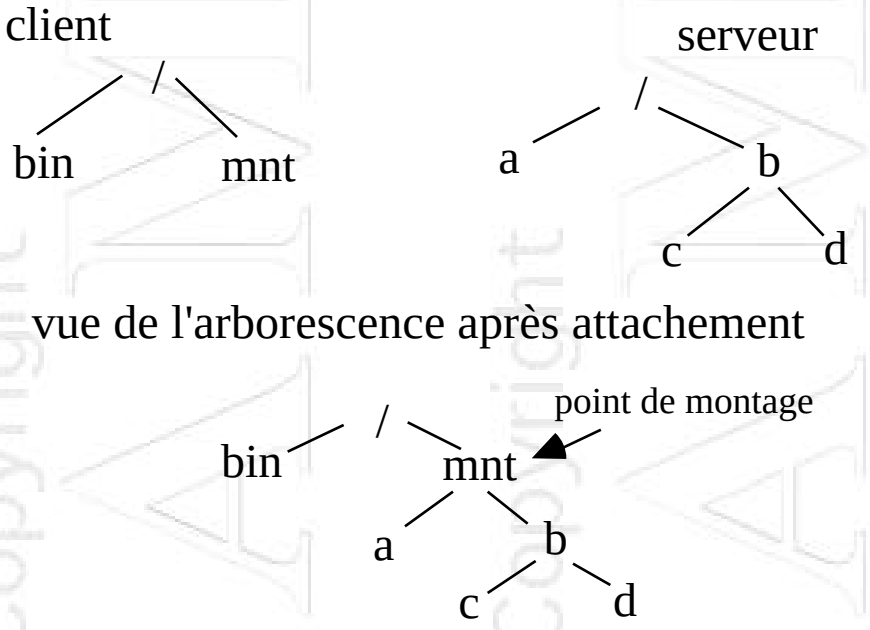
ADAPTABILITE et RECONFIGURABILITE

On trouve cette possibilité plutôt dans les systèmes répartis.

Techniques de Désignation (1)

1. **Mise en commun d'Arborescence** (Newcastle Connexion) ne fournit pas la transparence

2. **Attachement d'arborescences**



Transparence sur chaque site, mais il y a autant de vues de l'arborescence des fichiers qu'il y a de machines.

Les noms ne sont pas globaux.

En respectant un certain nombres de règles pour les opérations d'attachement (opérations effectuées par un administrateur d'habitude) on obtient des **noms administrativement globaux**.

Techniques de Désignation (2)

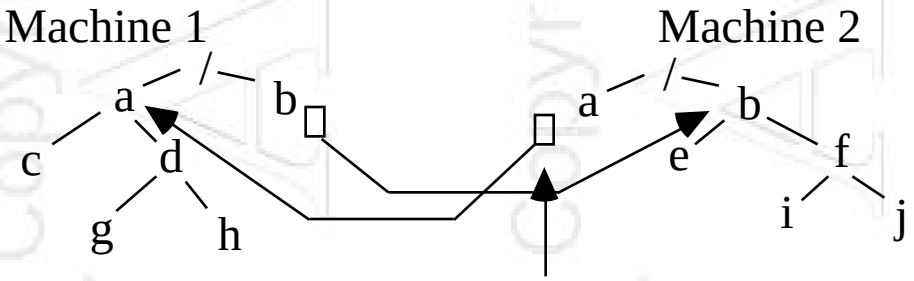
3.**Arborescence virtuellement centralisée** ou **arborescence de désignation unique**

La vue logique des fichiers est la même sur toutes les machines. Les noms sont globaux. Les fichiers sont physiquement répartis sur les serveurs.

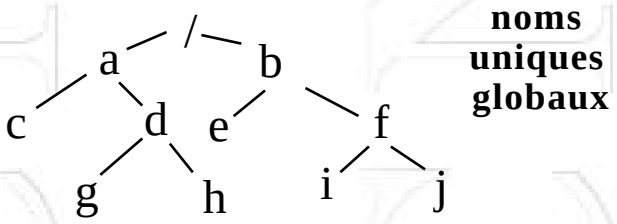
Cette solution s'appuie sur des noms internes uniques dans le temps et dans l'espace (UID).

UID Apollo :

site de création (20b)	date de création (36b)	type (8b)



Vue de l'utilisateur

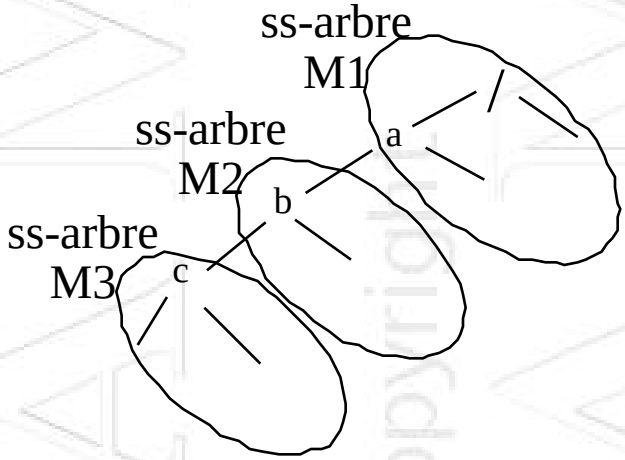


Solution qui n'est pas toujours commode pour tout ce qui est privé à une machine comme par exemples les executables.

Techniques de Résolutions des noms (1)

1. **Transformation du chemin d'accès**

Principe : Chaque machine résoud le morceau de chemin qui la traverse.



Un client veut accéder au fichier /a/b/c :

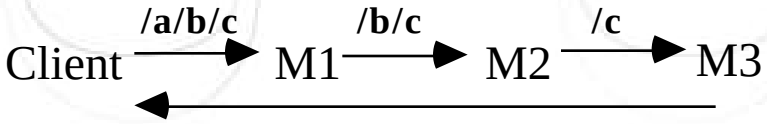
a. Client -> M1 qui reçoit tout le chemin

b. sur M1 :

- **/ évalué sur M1** : donne le **nom interne "racine"**, on va sur le disque
- **dans le répertoire racine** on trouve le **nom interne de "a"**, on va sur le disque
- **dans le répertoire "a"**, on trouve une indication telle **"b dans ss-arbre sur M2"**, on passe **/b/c à M2**

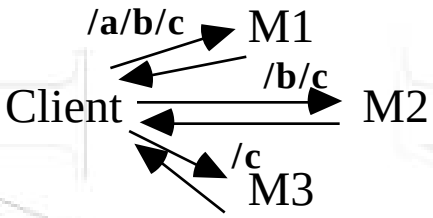
c. sur M2 : le reste du chemin **/b/c/myfile est évalué** de la même façon, **/c est passé à M3**

d. sur M3 la **fin du nom est résolu** et retour du résultat au client



Autre variante de la transformation de chemin d'accès :

. Résolution par chaque serveur et retour au client après chaque étape (image du polling).



L'effort de résolution est supporté surtout par le client.

Solution adoptée par NFS.

Techniques de Résolutions des noms (2)

2. **Méthode des identificateurs structurés**

Pour réaliser la correspondance entre le nom d'un fichier et sa localisation on utilise un identificateur structuré, celui-ci identifie une sous-arborescence qui contient le fichier cible.

identificateur structuré (is):

< identif de ss-arbre ; identif de fichier dans le ss-arbre>

Une table contient sur chaque site la correspondance :

nom symbolique <-> id ss-arb

La table peut être gérée de différentes façons :

-> remplissage suivant le principe de résolution par transformation de chemin d'accès

-> gérée sur chaque site à l'image d'un cache ou accédée sur un serveur de noms

Exemple :

/a/b/c	<ss-arbre 3; 11>

ss-arbre 3	M3

Le nom est indépendant de la localisation, quand le fichier migre il suffit de mettre à jour la table.

Techniques de Résolutions des noms (3)

3. **Méthode des caches de suggestion ("hints")**

. intervient pour la localisation
. améliore la performance si le cache contient une information non périmée (valide)
. si l'information est invalide, on peut utiliser les techniques précédentes pour la recharger.

Solutions utilisées lors d'une résolution de chemin d'accès :

- accès cache de noms côté client sinon requête au serveur de fichiers (ANDREW)
- accès cache de noms (partie préfixe du chemin) sinon diffusion de la demande (SPRITE)
- accès cache de noms sinon emploi d'une heuristique (APOLLO-DOMAIN)
- accès cache de noms sinon interrogation d'un serveur de noms (GRAPEVINE)

Techniques de Résolutions des noms (4)

4. **Points d'attachement**

Un point d'attachement est l'association d'une sous-arborescence à une feuille d'une autre sous arborescence.

Une table mémorise l'association

<feuille ; sous-arbre>

On parcours la table à chaque fois qu'un chemin d'accès traverse un point d'attachement.

Sémantique du partage de fichiers

PB : Comment gère-t-on l'accès simultané à un fichier ?

Problème plus large que le contrôle d'accès concurrents :
- . respect de la cohérence des données
- . respect de prédicat sur les données
- . sérialisabilité des accès

-> règles explicites de type verrou ou transaction

Session d'accès à un fichier :

ouvrir {accès lecture et écriture} **fermer**

Entrelacement d'opérations par +sieurs processus

Quelles règles implicites sont adoptées par le SGFR ?

Sémantiques

- **Sémantique "UNIX" :** reproduit le comportement d'Unix mais pour des accès distants
- **Sémantique "SESSION" :**
  - les écritures sont immédiatement visibles par les processus sur le même site mais invisibles pour des processus distants qui ont aussi ouvert le fichier
  - les changements sont visibles à tous les sites dès la fermeture

  => copies obsolettes, et pb quand écrivains multiples pour fusionner les copies différentes
- **Sémantique "FICHIER PARTAGE INVARIABLE" :** dès qu'un fichier est déclaré partagé, son nom et son contenu ne peuvent plus changer

  => partage en lecture, une modification oblige a créer un nouveau fichier
- **Sémantique "TRANSACTION" :** l'accès a un fichier équivaut à une transaction, il est verrouillé entièrement

  open() = début transaction

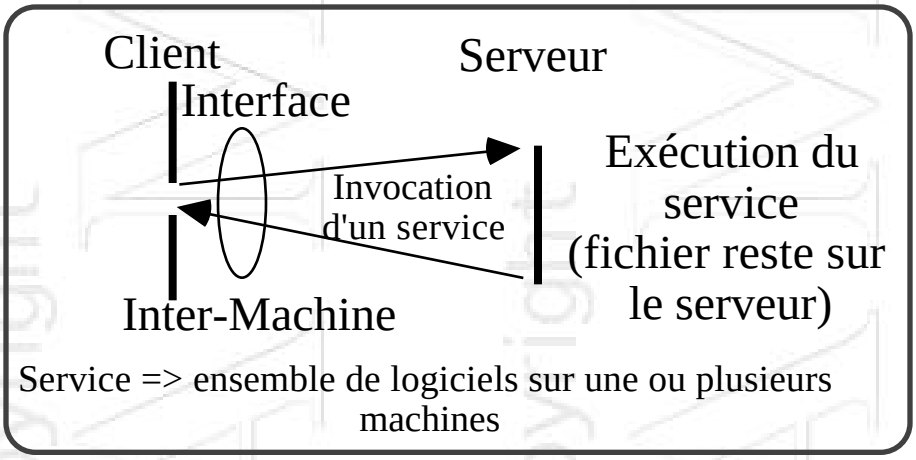
  close() = fin transaction

  => Sérialisation des accès, pas de concurrence

Méthode d'accès aux fichiers distants (1)

Modèle Client-Serveur +
Interface qui masque l'implantation

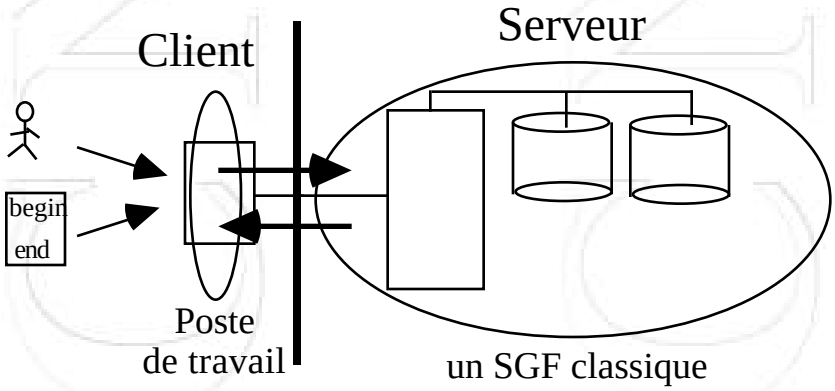
Accès à distance (approche client-serveur) :



-> charge du serveur
-> charge du réseau

Importance de la sémantique d'exécution des demandes de service

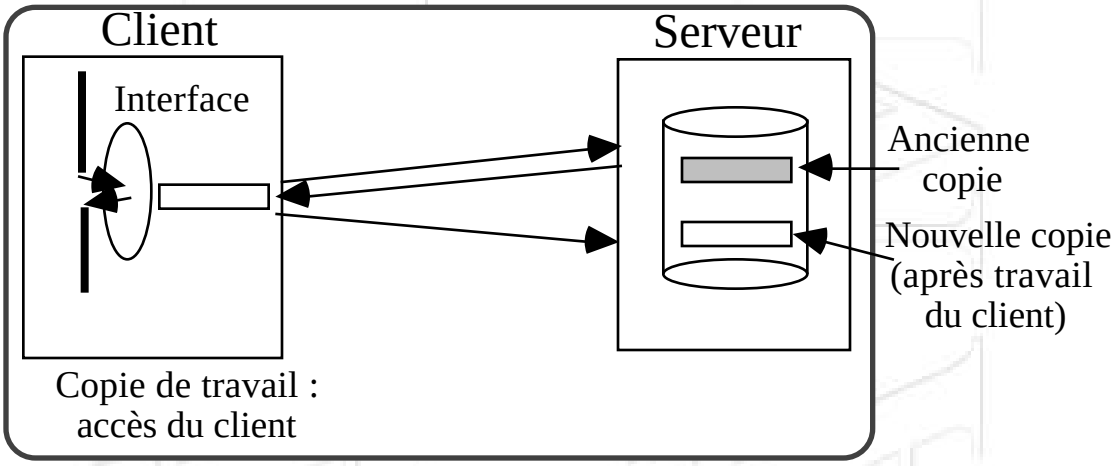
Application aux SGFR :



Interface :ensemble d'opérations sur les fichiers distants
créer, détruire, lire, écrire, changer les droits, ...

Méthode d'accès aux fichiers distants (2)

Téléchargement :

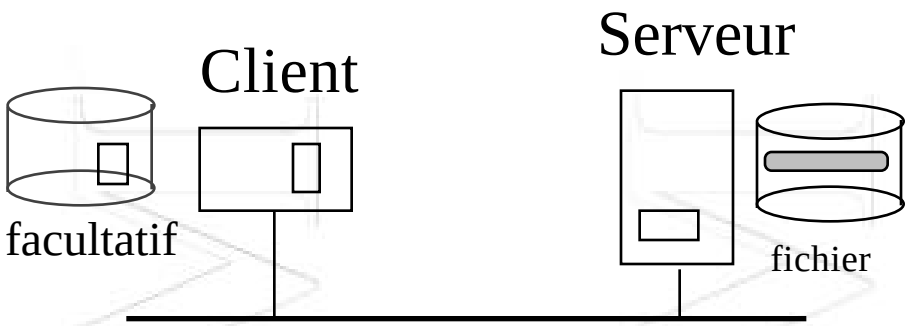


Importance de la politique de rafraichissement du serveur.

Le téléchargement d'un fichier peut être partiel, dans ce cas, c'est une **technique de cache** :
-> cache qui conserve une copie des données
-> transfert d'informations d'un fichier par groupe de blocs de données (repose sur le principe de localité des références - technique du "read ahead")
-> vidage du cache : LRU (adapté), FIFO (facile à implanter)
-> fixer la copie de référence, en général celle du serveur, il faut gérer la cohérence (toutes les copies sont identiques)

La sémantique du partage de fichier a une importance sur la méthode d'accès.

Conception d'un cache de fichiers



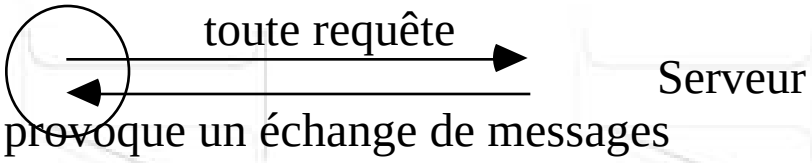
Un cache en mémoire principal du serveur améliore la performance du serveur.

Côté client, le cache peut être en mémoire, ou sur disque. L'unité de cache peut être un morceau du fichier, ou, l'ensemble du fichier. La granularité a son importance pour maximiser le "hit ratio" tout en évitant le faux partage en évitant des temps de transfert trop longs et une surcharge du logiciel de communication, sans rendre trop difficile le maintien de la cohérence.

Surcharge du logiciel de communication : gros grain -> 1 seul message chez le serveur et 1 acquit chez le client mais plus de fragmentation couche basse (trame Ethernet = 1,5Ko), ne pas oublier qu'un segment TCP = 64Ko.

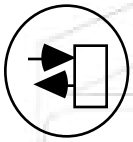
Localisation du cache

Pas de cache :

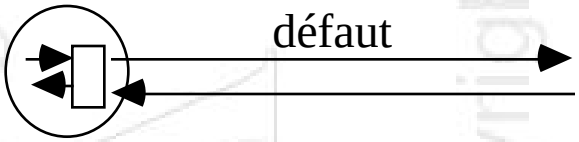


provoque un échange de messages

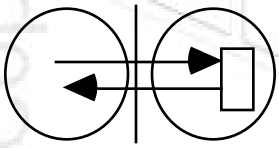
Cache dans les processus :



pas de défaut

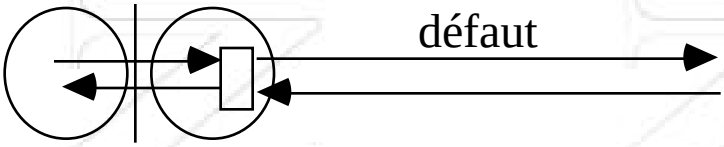


Cache dans le noyau :

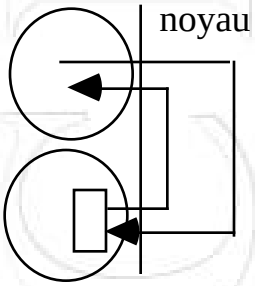


pas de défaut

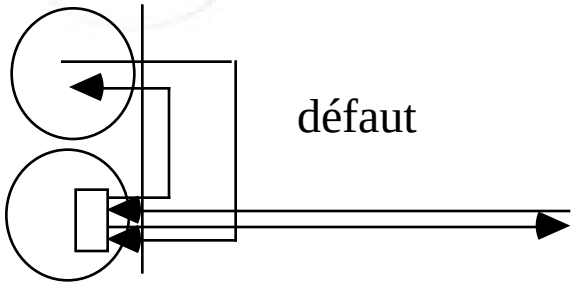
noyau



Cache dans un serveur spécialisé :



pas de défaut



Politique de Mise à jour du serveur

Problème de la mise à jour du serveur :

. A chaque modification : meilleur pour les lectures, toujours coûteux pour les écritures, et cohérence insuffisante entre clients (idem write-throught avec problème mémoire répartie)

. Quand le bloc est vider pour faire de la place (write-back)

. A la fermeture (write-on-close idem write-back) : meilleur dans le cas de fichiers temporaires, mais problème de cohérences entre clients

. Périodiquement : compromis entre les deux stratégies précédentes (30s dans SPRITE)

La gestion de la cohérence des caches peut être vue comme un problème de mémoire répartie partagée.

Validation du contenu du cache client

Un client qui ne modifie pas sa copie doit vérifier si elle est toujours valable.

. **A la charge du client :** Le client interroge le serveur à propos de la validité du contenu de son cache. Sa copie est-elle identique à celle du serveur ?
Vérification par comparaison d'une date associée aux données quand elles ont été chargées ? A quelle fréquence : à chaque accès, périodiquement, lors d'une nouvelle ouverture ?

. **A la charge du serveur :** Le serveur maintient une liste des clients qui ont une copie du fichier ... lors d'une modification, il prévient le clients ... très lourd ! La copie sur le serveur est la copie de référence.
On a transgressé le schéma de relation client/serveur, puisque le serveur a une rétroaction sur les clients.

Tolérance aux pannes

- **Serveur avec Etat :** le serveur maintient des informations sur les clients qui utilisent ses fichiers

Avantages :
- identification des clients en permanence
- transfert en mode connecté (plus fiable)

Inconvénients :
- récupérer les ressources mobilisées à la fermeture
- reconstruire l'état du serveur en cas de panne
- détecter et éliminer les orphelins en cas de panne des clients

solution pour les fichiers distribués sur réseaux WAN

- **Serveur sans Etat :** aucune inforamtion conservée par le serveur sur ses clients

Avantages :

Inconvénients : Requêtes idempotentes
- READ/WRITE en mode "append" transformées en opération à une position dans le fichier
- destructions idempotentes ?
- sécurité plus difficile à maintenir ... il faut authentifier le demander à chaque transaction

Disponibilité

Propriétés des fichiers :
- robustesse : survivre à la panne du support (sauvegardes)
- reprenable : possibilité de revenir à un état antérieur si une opération sur le fichier échoue ou est annulée en cours d'accès
- disponible : l'accès au fichier est toujours possible malgré les pannes de processeur, de disque ou de réseau

attention : robuste ≠ reprenable et robuste ≠ disponible

Moyens : limiter le nombre de machines qui interagissent
- pendant la phase de localisation : pas de pb, si les dépendances se situent entre un client et un serveur, mais pb quand un chemin d'accès traverse plusieurs machines qui peuvent être en panne
  -> réplication des répertoires
  -> cache des informations de localisation dans les répertoires d'attachement
  -> cache de suggestions

- pendant la phase d'accès : serveurs redondants

Réplication de fichiers

Améliore la disponibilité et la performance

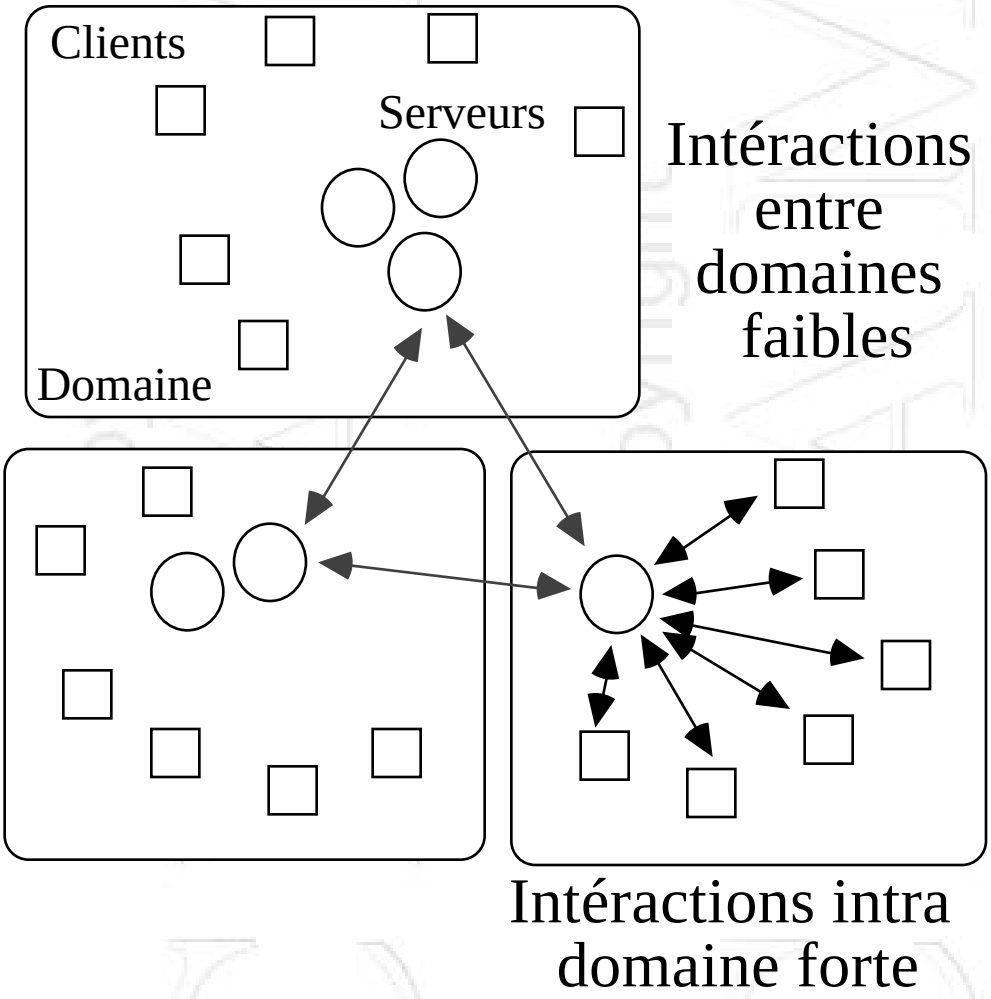
Objectifs :
- masquer l'existence des copies multiples à l'utilisateur en particulier dans le nom du fichier
- indépendance des copies : si une panne se produit sur une copie, les autres restent accessibles
- distinguer les copies par des noms internes différents
- copies cohérentes lors de mises à jour

Solutions :
- les fichiers sont répliqués en lecture
- lors d'une panne du réseau (partitionnement par ex) aucune cohérence n'est maintenue
- les copies d'un fichier sont organisées en une copie maître et des copies secondaires : mise à jour sur le maître puis propagation sur les autres copies

Adaptabilité (1)

Influence de l'architecture du réseau :

Arriver à borner la demande quelque soit le nombre de machines



Influence de la conception des serveurs :

utilisation du "multi-threading" -> un thread par requête client

Exemples

- ANDREW (parent du DFS de DCE)
- SPRITE (l'an prochain)

Copyright CNAM

Andrew File System

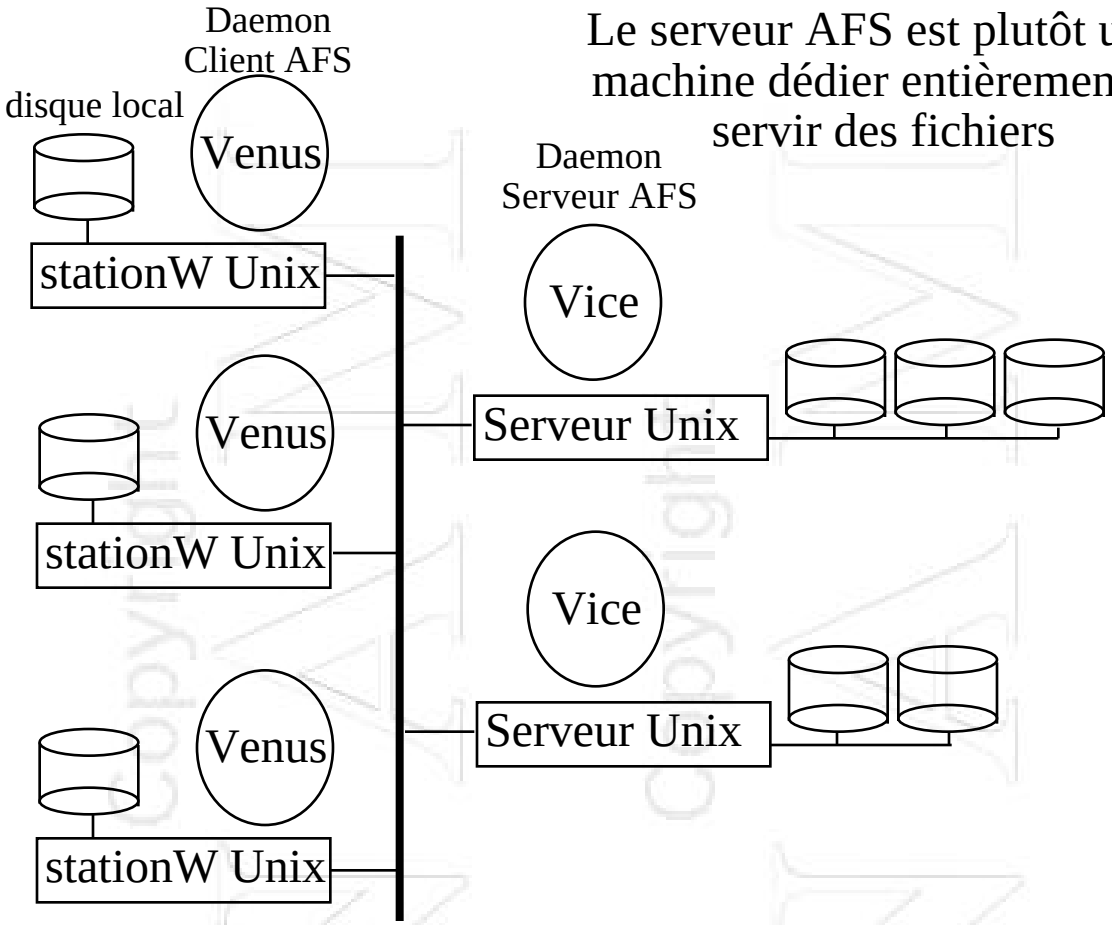
Pas les mêmes objectifs que NFS :
NFS a été conçu pour un groupe de clients de 10 à 15 tandis que AFS vise plusieurs dizaines de clients. AFS vise **l'adaptabilité** et **l'extensibilité**.

Caractéristiques d'AFS :
- Les clients cachent des fichiers entiers, le cache d'un client se fait sur disque et peut avoir une taille de quelques centaines de Mo
- Les serveurs transfèrent tout un fichier quand un client fait un accès sur celui-ci

Hypothèses de conception :
- Les fichiers sont petits < 10 Ko
- Il y a plus de lectures que d'écritures (6 fois)
- L'accès est souvent séquentiel rarement aléatoire
- Peu de partage entre utilisateurs, quand il y a partage, c'est souvent un seul utilisateur qui le modifie
- Il existe un principe de localité spatiale et temporelle

AFS fonctionne bien pour les fichiers qui sont manipulés par un seul utilisateur et qui ne sont pas souvent modifiés. Il n'est pas recommandé pour supporter des Bases de Données.

Architecture d'AFS

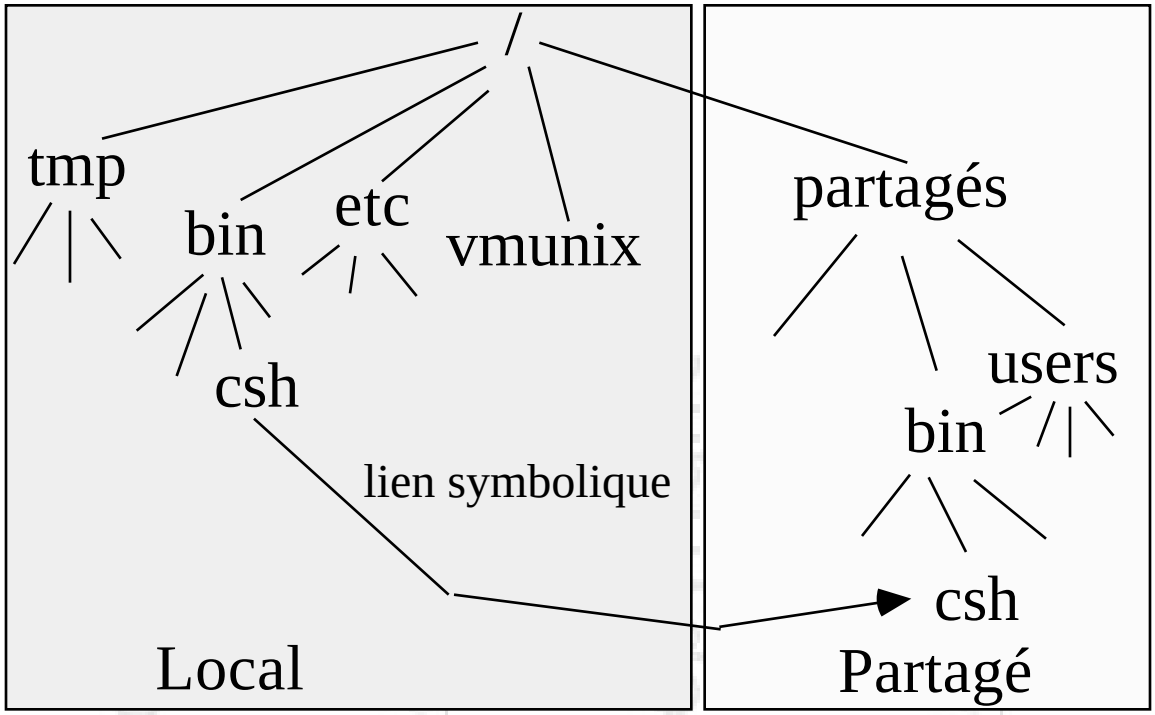


Le serveur AFS est plutôt une machine dédier entièrement à servir des fichiers

Comme pour NFS, un appel système concernant un fichier est examiné par le noyau. Si le fichier n'est pas local, le serveur correspondant est contacté.

La localisation des serveurs contenant un fichier demandé est stockées dans un annuaire. Cet annuaire est répliqué sur tous les clients, un protocole gère la cohérence des réplicats qui peuvent parfois ne pas être tout à fait à jour.

Espace des noms de fichiers d'un Client



Espace local : fichiers temporaires, paramètres de configuration système, tous les fichiers privés à la machine client

Espace des fichiers partagés : binaires, utilisateurs

Aspects généraux

- Les fichiers respectent le modèle Unix : suite d'octets "plate"

- Les fichiers sont regroupés par **Volumes**, le volume est l'unité de localisation et de migration. Une partition = +sieurs volumes.

L'arborescence des fichiers est consituée de volume reliés par des points de montage.

- Les répertoires et les fichiers dans l'espace partagé sont identifiés de façon unique par un fids de 96 bits :

32 bits	32 bits	32 bits
N° Volume	File Handle	champ pour unicité

Il est indépendant de la localisation.

La résolution des noms est à la charge des clients (Venus) et est faite morceau par morceau comme dans NFS.

Protocole de gestion de cohérence des caches

AFS cache des fichiers et non des blocs de fichiers.
La cohérence des caches est fondée sur un mécanisme de "promesse de rappel" :

Quand un client demande un fichier, le serveur lui fournit une copie et conserve une "promesse de rappel" avec le nom du client. Le client à la réception de son fichier garde un jeton qui a l'état valide.
Lorsqu'un client vide son cache, le fichier est recopié sur le serveur. Si d'autres clients ont copie du fichier dans leur cache, ils sont rappelés par le serveur. Quand un client est rappelé, il met le jeton concerné à invalide. Si un programme utilisateur sur le client veut accéder à ce fichier, il doit demander une copie fraiche au serveur.
En cas de panne d'un client, ses jetons ne sont peut-être plus à jour ? Le client effectue une requête de validation auprès du serveur avec la date de modification du fichier. Si rien n'a changé, le jeton redevient valide sinon il devient invalide.
Le vidage du cache se fait suivant la stratégie LRU. Un cache fait une taille de l'ordre de 100Mo. Les transferts entre le client et le serveur s'effectuent par messages de 64Ko.

Bibliographie

Tanenbaum 1995. "Distributed Operating Systems". Prentice Hall.

G. Colouris, J. Dollimore, T. Kindberg. 1994. "Distributed systems, Concept and Design". 2nd Edition. Addison-Wesley.

Stern 1992. "Managing NFS and NIS". O'Reilly& Associate.

Levy, Silberschatz 1990. "Distributed File Systems : Concepts and Exemples". ACM computing Surveys. V22. N4.

Nussbaumer 1991. "Téléinformatique IV : Messagerie électronique X400, Messagerie Industrielle MMS, Transfert de fichiers FTAM". Presses Polytechniques Romandes.