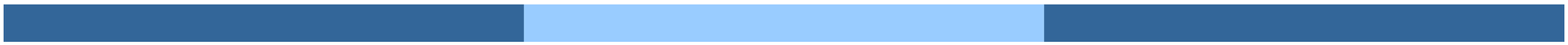
Implémentation du système de fichier



Menu

- Structure du systèmes de fichiers
- Implémentations de répertoire de fichiers
- Méthodes d'allocations d'espace
- Gestion de l'espace libre
- Fiabilité et performance
- NFS

Menu

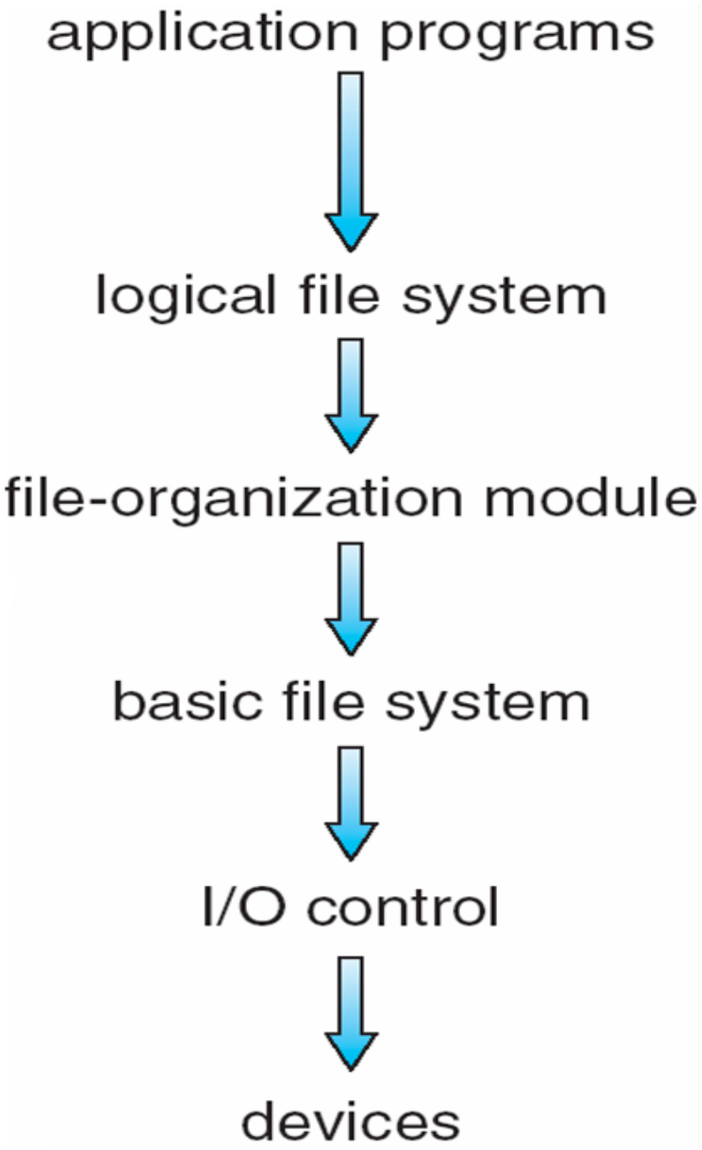
- **Structure du systèmes de fichiers**
- Implémentations de répertoire de fichiers
- Méthodes d'allocations d'espace
- Gestion de l'espace libre
- Fiabilité et performance
- NFS

Systèmes de fichiers

- Le système de fichiers réside sur le stockage secondaire (disques)
  - Traduit les adresses de logique à physique
  - Fournit un accès efficace et pratique au disque en permettant le stockage et la récupération des données facilement
- Le disque fournit
  - réécriture sur place (lire, modifier, réécrire)
  - accès aléatoire
- **File control block** (Unix = **inode**) – structure de stockage consistant en des informations sur un fichier

Niveaux de système de fichiers

- **I/O control**
  - Étant donné des commandes telles que "lire le lecteur 1, le cylindre 72, le secteur 10, dans l'emplacement mémoire 1060", il envoie des commandes spécifiques au matériel de bas niveau au contrôleur matériel.
- **Basic file system**
  - Reçoit une commande comme "récupérer le bloc 123" est traduit pour le niveau "I/O control"
  - Gère également les mémoires tampon et les caches (allocation, libération, remplacement)
- **File organization module**
  - comprend les fichiers, l'adresse logique et les blocs physiques
  - Traduit l'adresse de bloc logique en adresse de bloc physique
  - Gère l'espace libre, l'allocation de disque
- **Logical file system**
  - gère les informations de métadonnées
  - Traduit le nom de fichier en numéro de fichier, identificateur de fichier, emplacement en conservant les blocs de contrôle de fichier (inodes sous UNIX)
  - Gère les répertoires



Structures de système de fichiers sur disques

- Sur le disque, un système de fichiers peut contenir:
  - **Boot control block** contient les informations nécessaires au système pour démarrer le système d'exploitation à partir de ce volume
  - **Volume control block (superblock, master file table)** contient des détails sur le volume / la partition
    - Nombre total de blocs, nombre de blocs libres, taille de blocs, pointeurs de blocs libres ou tableau
  - La structure du répertoire organise les fichiers (un par système de fichiers)
  - **File Control Block (FCB)** contient de nombreux détails sur le fichier

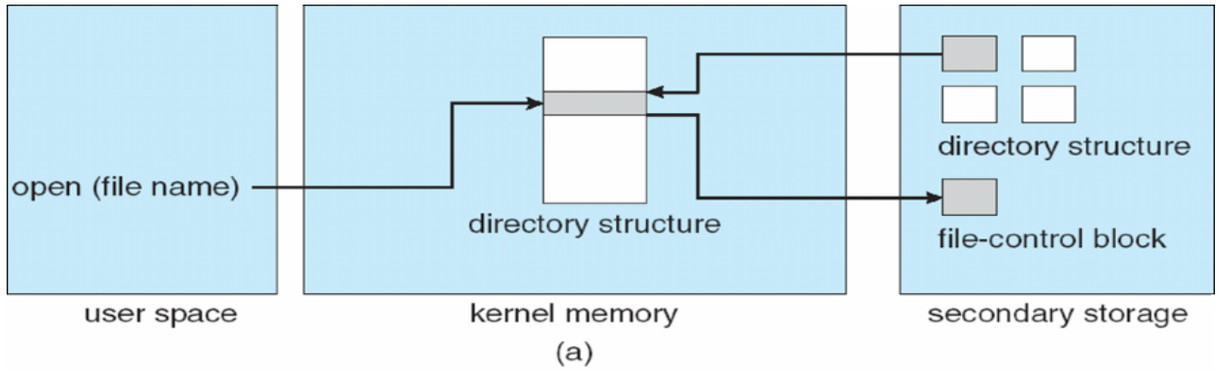
Typical FCB

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Étapes pour la création de fichier:
1. allouer FCB
2. lire le répertoire dans la mémoire
3. mettre à jour le répertoire avec le nouveau nom de fichier
4. écrire des données sur le disque

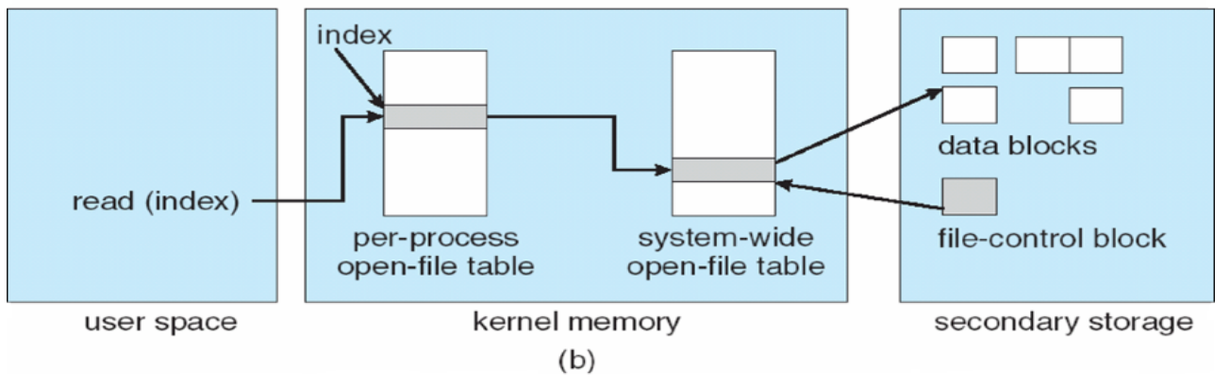
Structures de système de fichiers en mémoire

- **En mémoire,** un système de fichiers utilise ces informations pour la gestion et la performance (mise en cache):
  - **Mount table** stockage des montages de système de fichiers, des points de montage, des types de systèmes de fichiers
  - **System-wide open-file table** contient un FCB pour chaque fichier ouvert.
  - **Per-process open-file table** a un pointeur sur l'entrée dans la table de fichiers ouverts à l'échelle du système.



(a) Open retourne un descripteur / descripteur de fichier pour utilisation ultérieure

```
open():
1. if no entry in SWOF
     search dir and create an entry
2. create entry in PPOF
3. add pointer to SWOF
4. increment file used counter in SWOF
close():
1. reverse order of operations
2. if file counter is 0
     remove entry in SWOF and write metadata to disk
```



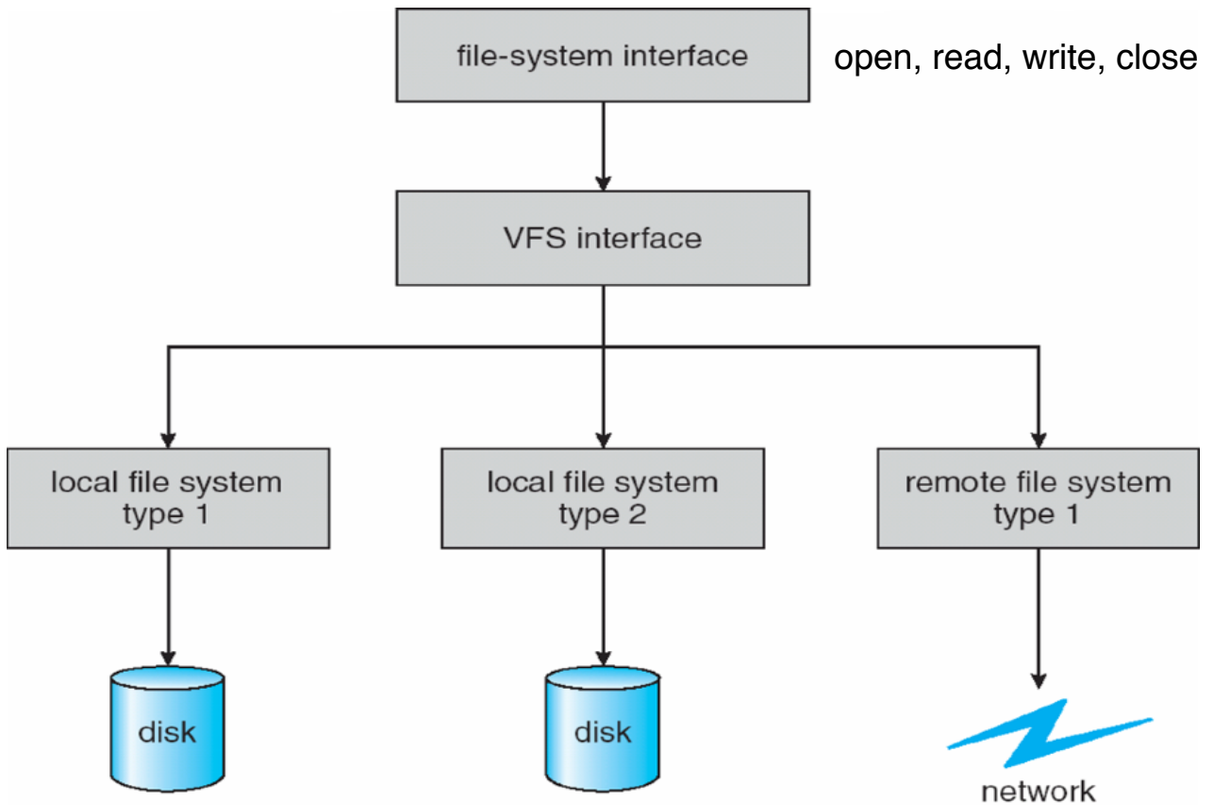
(b) Les données de “read” sont copiées dans l'adresse mémoire de processus utilisateur spécifiée

Partitions et mounting

- Le disque peut être divisé en plusieurs partitions
- Partition peut être:
  - un volume contenant un système de fichiers
  - **brut** – juste une séquence de blocs sans système de fichiers (e.g., swap, database)
- Boot block peut pointer vers boot volume ou boot loader
  - Ensemble de blocs contenant suffisamment de code pour savoir comment charger le noyau à partir du système de fichiers
  - Ou un programme “boot management” pour multi-OS booting (e.g., GRUB)
- **Root partition** contient le SE, d'autres partitions peuvent contenir d'autres SE, d'autres systèmes de fichiers ou être crues
  - Mounted à temps de boot
  - D'autres partitions peuvent être montées automatiquement ou manuellement
- À temps de “mount”, la cohérence du système de fichiers est vérifiée
  - Toutes les métadonnées sont-elles correctes?
    - NON: réparez-le, réessayez
    - OUI: mets-le dans le mount table, donne l’accès

Virtual File Systems (VFS)

- Besoin d'accéder à des systèmes de fichiers différents sur le même disque, ou sur un réseau, montés localement
- Virtual File Systems (VFS) sur UNIX fournir un moyen orienté objet de mettre en œuvre des systèmes de fichiers
- VFS permet d'utiliser la même interface d'appel système (API) pour différents types de systèmes de fichiers
  - Sépare les opérations génériques du système de fichiers des détails d'implémentation
    - Implémente les **vnodes** qui contiennent des **inodes** ou des détails de fichiers réseau
  - Puis envoie l'opération aux routines appropriées d'implémentation du système de fichiers
- L'API est à l'interface VFS, plutôt que tout type de système de fichiers spécifique



Virtual File System

- Par exemple, Linux a quatre types d'objets:
  - inode: fichier individuel
  - fichier: fichier ouvert
  - Superblock: système de fichiers entier
  - dentry: entrée de répertoire individuelle

- VFS définit un ensemble d'opérations sur les objets à implémenter
  - Chaque objet a un pointeur vers une table de fonction
    - La table de fonction a des adresses de routines pour implémenter cette fonction sur cet objet
    - Ils comprennent open (), close (), read (), write (), mmap (), etc.

Menu

- Structure du systèmes de fichiers
- **Implémentations de répertoire de fichiers**
- Méthodes d'allocations d'espace
- Gestion de l'espace libre
- Fiabilité et performance
- NFS

Implémentation de répertoires

- **Linear List** des noms de fichiers avec des pointeurs vers les blocs de données
  - Simple à programmer
  - Il faut beaucoup de temps pour exécuter

- **Hash Table** – liste linéaire avec structure de données de hachage
  - Diminue le temps de recherche dans l'annuaire
  - **Collisions** – situations dans lesquelles deux noms de fichier hachent au même endroit
  - Seulement bon si les entrées sont de taille fixe, ou alors utilisez la méthode chained-overflow (l'entrée hash peut être une liste chaînée au lieu d'une valeur individuelle)

Menu

- Structure du systèmes de fichiers
- Implémentations de répertoire de fichiers
- **Méthodes d'allocations d'espace**
- Gestion de l'espace libre
- Fiabilité et performance
- NFS

Méthodes d'allocation

- Comment allouer de l'espace aux fichiers afin que l'espace disque soit utilisé efficacement?

- Trois méthodes:
  - Contiguë
    - ✓ “Extents”
  - Chaîné
  - Index

- Considérations:
  - Généralement on veux allouer les blocs de manière contiguë
  - Le taille du fichier n’est pas connue à l’avance
  - Accès séquentielle vs directe
  - Fragmentation

Fragmentation des données

- Nous avons déjà discuté de deux types de fragmentation:
  - interne
  - externe

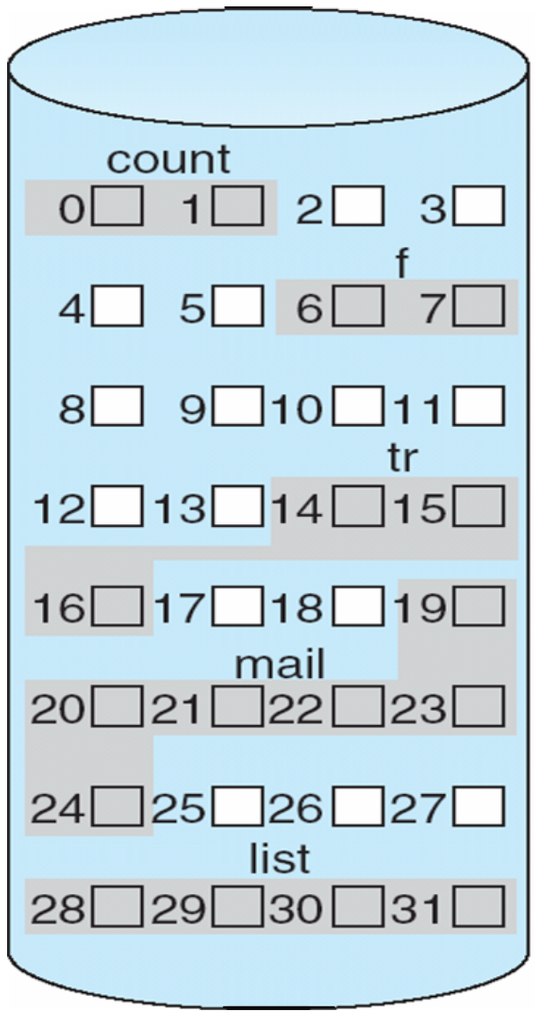
- Rappel:
  - allocation par bloc pas de fragmentation externe
  - si le fichier a une taille égale à N blocs - pas de fragmentation interne

- Maintenant un troisième type de fragmentation - la fragmentation des données
  - Si les blocs d'un fichier sont distribués sur le disque, il sera moins efficace d'y accéder - le seek time et latence sera plus élevé pour accéder à chaque bloc dans le fichier

Méthodes d’allocations - contiguë

- Chaque fichier occupe un ensemble de blocs contiguës
- Meilleure performance (IOPS) dans la plupart des cas
  - La tête de disque ne saute pas de secteur en secteur, mais se déplace le long d'une piste, et se déplace d'une piste du cylindre au cylindre suivant
- Simple
  - Seul l'emplacement de départ (adresse de bloc) et la longueur (nombre de blocs) sont requis
  - L'accès direct est également soutenu par un calcul facile pour trouver un bloc
- Les problèmes incluent:
  - trouver de l'espace pour un nouveau fichier
  - connaître la taille du fichier
    - ✓ facile lorsque le fichier est copié, mais un fichier peut encore évoluer
    - ✓ algorithme de meilleur ajustement peut être appliqué pour déplacer avec le nouveau fichier de taille, mais beaucoup de temps
  - fragmentation externe
    - ✓ besoin de compactage hors ligne (temps d'arrêt) ou en ligne (mais pénalité de performance)

Méthodes d'allocations - contiguë



directory

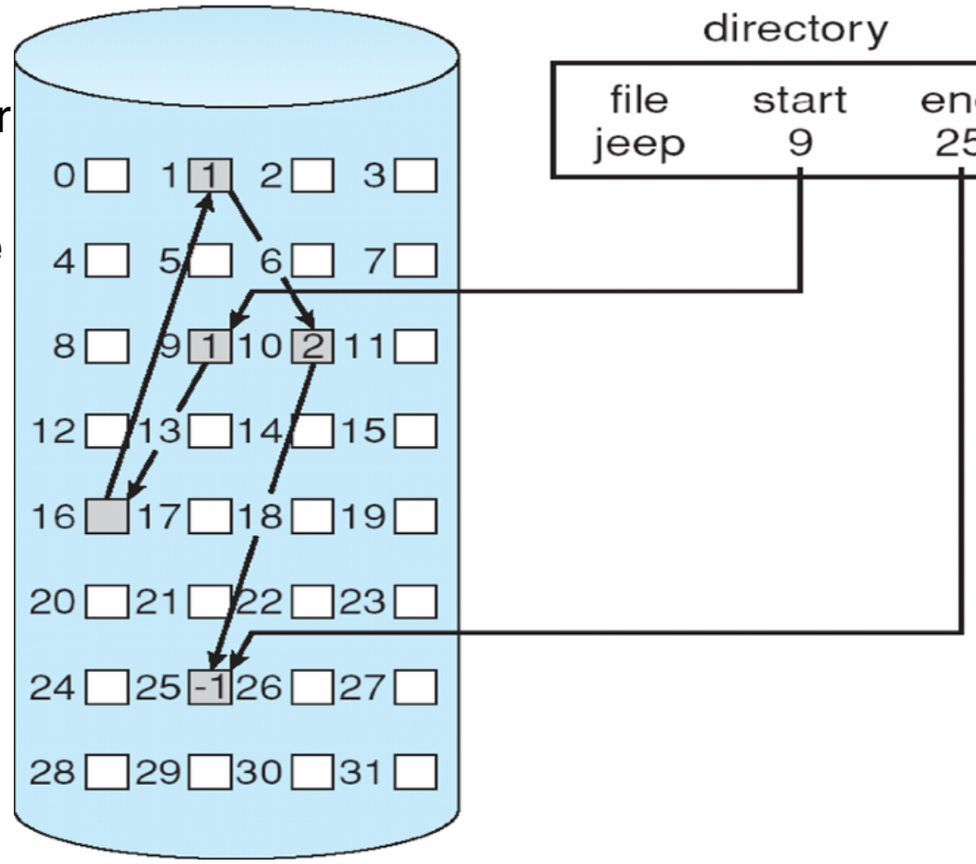
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation par extents

- Extent: ensemble de blocs contigus
  - peut être fixé taille (pas de fragmentation externe) ou taille variable (pas de fragmentation interne)
- Inode contient liste ou index des extents d'un ficher
  - Fichiers contigu: comme allocation contiguë
  - Fichier fragmenté: pire qu'allocation indexées
- Fragmentation de fichiers est un problème de toute façon
  - Profite des efforts de défragmentation
- En générale, allocation par extents est plus compact qu'un index classique
  - plus compact = plus efficace (rapide)

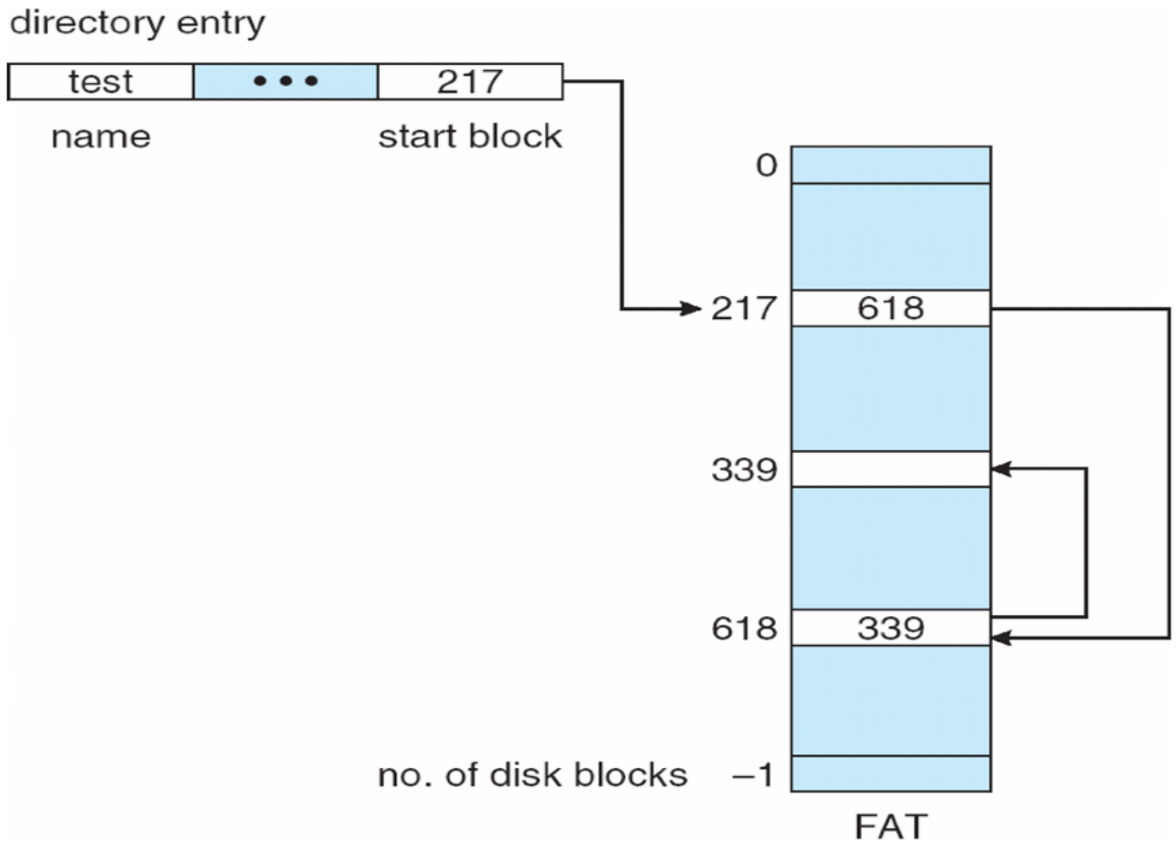
Méthodes d'allocation - chaîné

- Chaque fichier est une liste chaînée de blocs
- Le répertoire a un pointeur vers le premier et le dernier bloc
- Chaque bloc contient un pointeur sur le bloc suivant, le dernier bloc a un pointeur nul
- Pas de fragmentation externe, pas de compactage nécessaire
- Le fichier peut se développer tant qu'il y a des blocs libres
- Système de gestion de l'espace libre appelé lorsqu'un nouveau bloc est nécessaire
- Besoin d'allouer de l'espace pour les pointeurs de blocs
  - Améliorer l'efficacité en regroupant les blocs en groupes, mais cela augmente la fragmentation interne
- L'accès direct peut être inefficace
  - La localisation d'un bloc peut nécessiter de nombreuses E / S et recherches de disques (il faut suivre les pointeurs)
- La fiabilité peut être un problème
  - Le pointeur endommagé perd le reste du fichier
  - Partiellement résolu avec une liste doublement chaînée ou une liste de pointeurs, mais plus de frais généraux



Méthodes d'allocation - chaîné

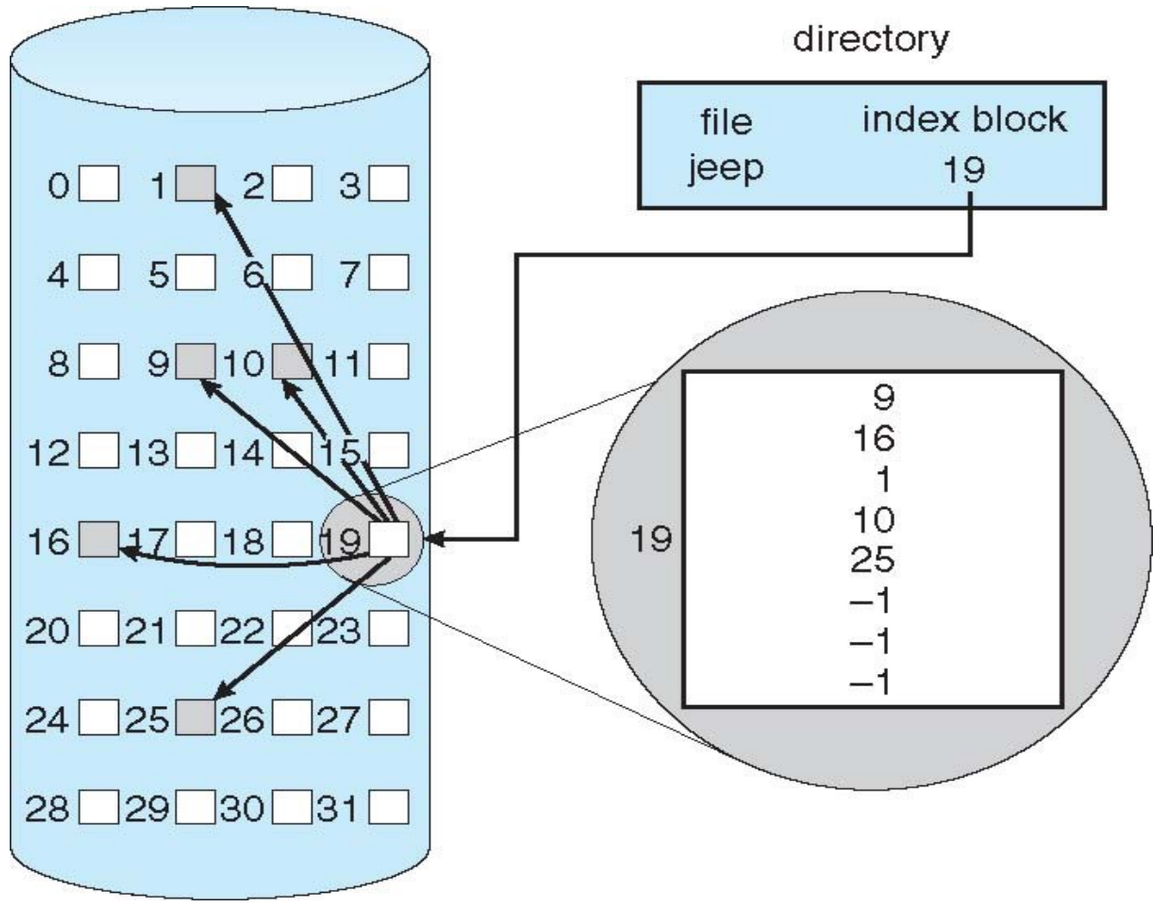
- FAT (File Allocation Table)
  - Début du volume a table, indexé par numéro de bloc
  - L'entrée de table i contient l'index de bloc suivant
  - Tout comme une liste liée, mais plus rapide sur le disque et cacheable (garder FAT en mémoire)
  - Nouvelle attribution de bloc simple



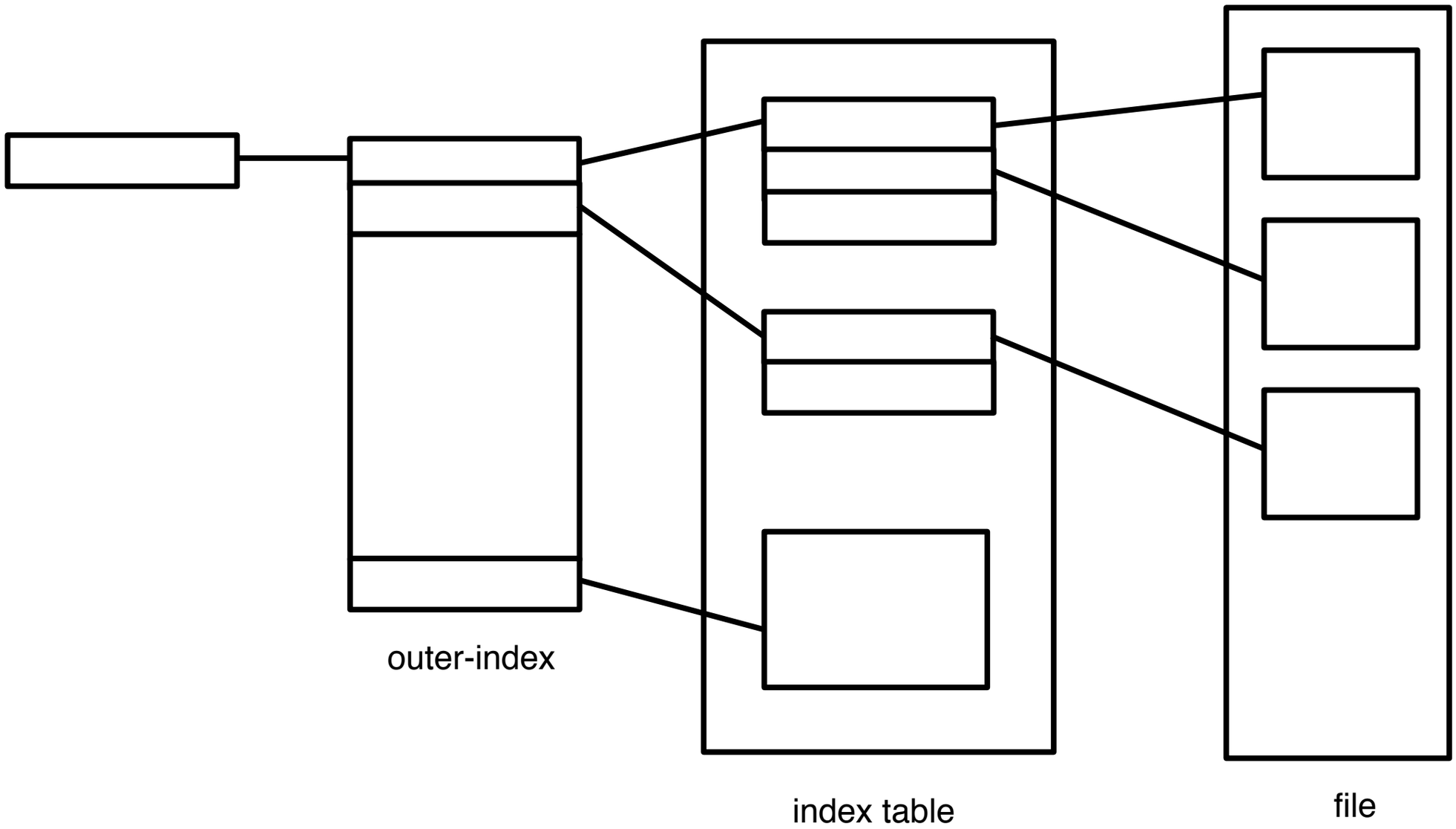
Méthodes d'allocations - index

- Chaque fichier a son propre bloc d'index de pointeurs vers ses blocs de données
- Permet accès aléatoire efficace
- Pas de fragmentations
- Pas besoin de compacter

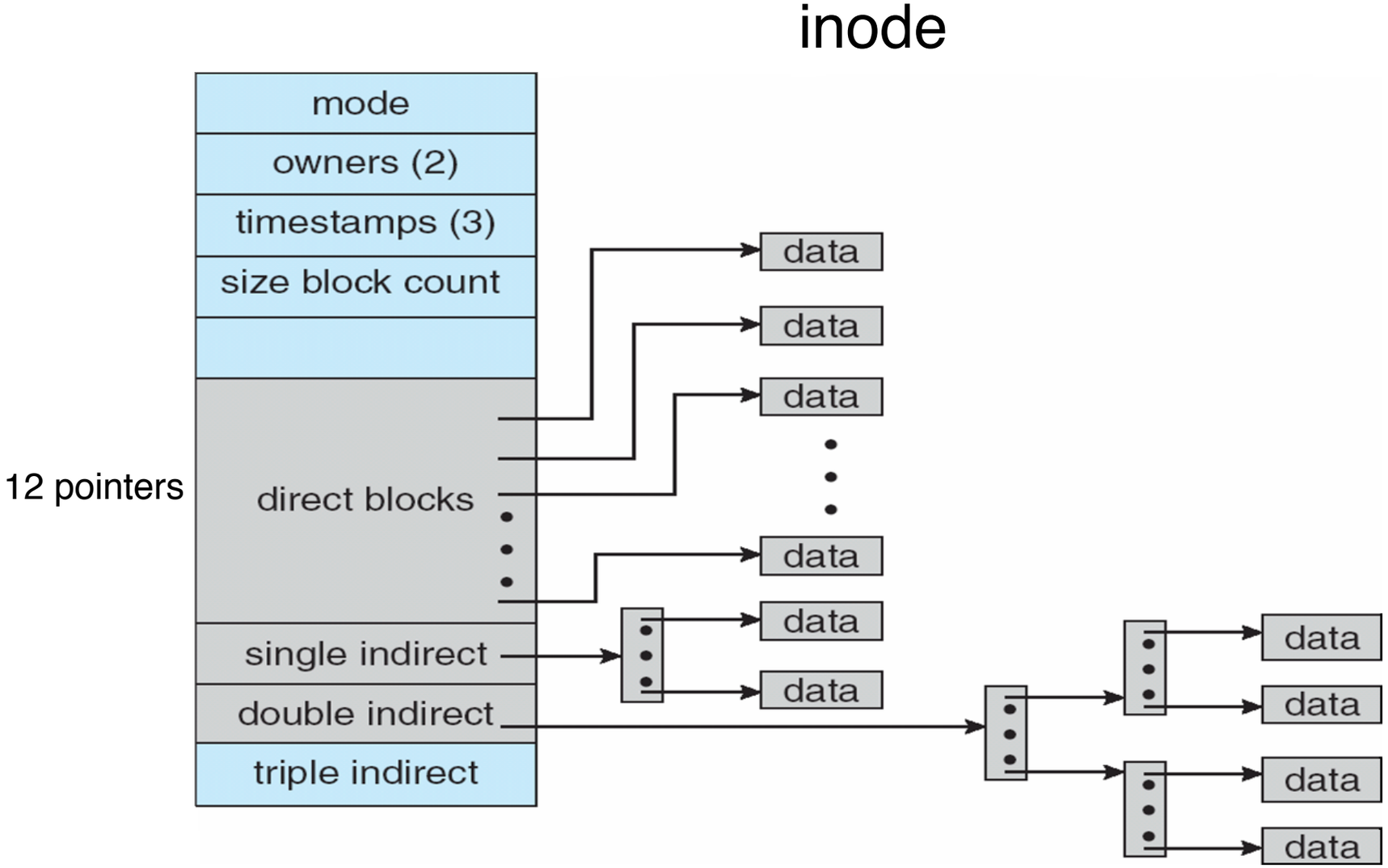
- overhead pour la lecture du bloc d'index
- L'index peut occuper plusieurs blocs
  - index contigu? chaîné? indexé?



Multi-level index



Une combinaison: UNIX File System (UFS)

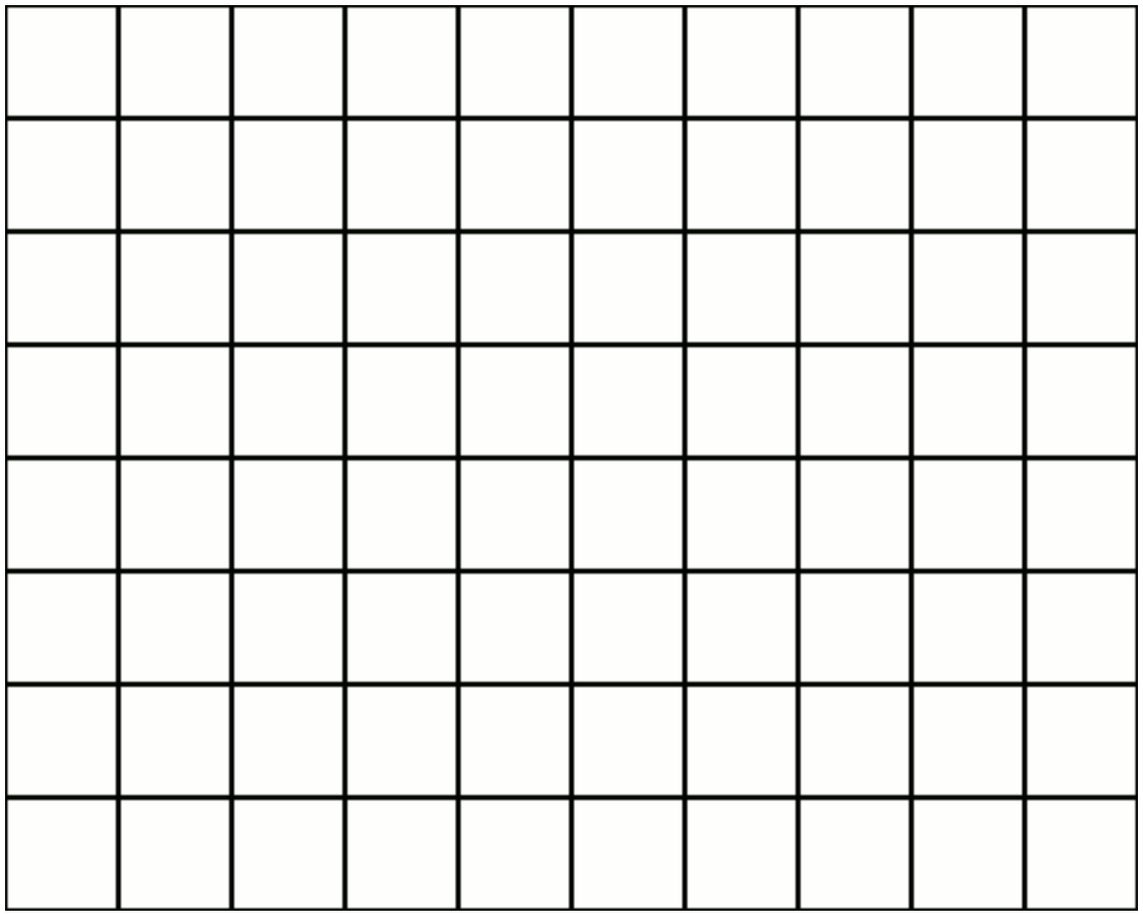


Performance

- La meilleure méthode d'allocation dépend du type d'accès au fichier
- Contigu:
  - idéal pour les accès séquentiels et aléatoires
- Chaîné:
  - bon pour séquentiel (conserver l'adresse du bloc suivant en mémoire)
  - mauvais pour aléatoire (i-ème bloc a besoin de i lectures)
- Certains systèmes prennent en charge les schémas d'allocation contigus et liés
  - Déclarez le type d'accès à la création, sélectionnez soit contigu ou lié
  - Peut se convertir en recréant / copiant le fichier
- Indexé: plus complexe pour évaluer sa performance
  - L'accès à un seul bloc peut nécessiter une ou plusieurs lectures de bloc d'index, puis un bloc de données lu
  - La mise en cluster peut aider à améliorer le débit, à réduire les frais généraux du processeur

Défragmentation

- Éviter la fragmentation:
  - Comprendre la source de fragmentation
  - Placer un nouveau fichier là où il reste de la place
  - Éviter de bloquer un fichier encore ouvert avec un nouveau fichier

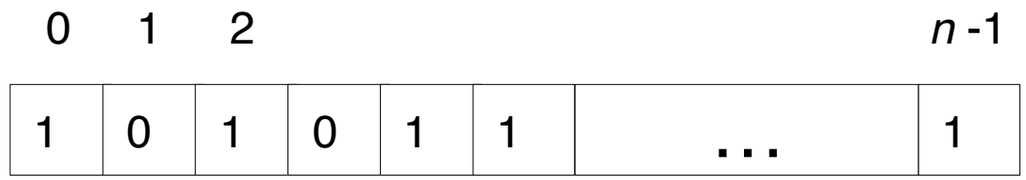


Menu

- Structure du systèmes de fichiers
- Implémentations de répertoire de fichiers
- Méthodes d'allocations d'espace
- **Gestion de l'espace libre**
- Fiabilité et performance
- NFS

Gestion de l'espace libre

- Le système de fichiers maintient une **free-space list**
- **Bit vector** or **bit map** (*n* blocks)



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ vide} \\ 0 \Rightarrow \text{block}[i] \text{ occupé} \end{cases}$$

- Facile et efficace pour obtenir des fichiers contigus
- **Calcul de numéro de bloc vide : (nombre de bits par mot) x (nombre de mots de valeur 0) + décalage du premier bit "1"**
- Bit map nécessite plus d'espace et les disques sont plus grands

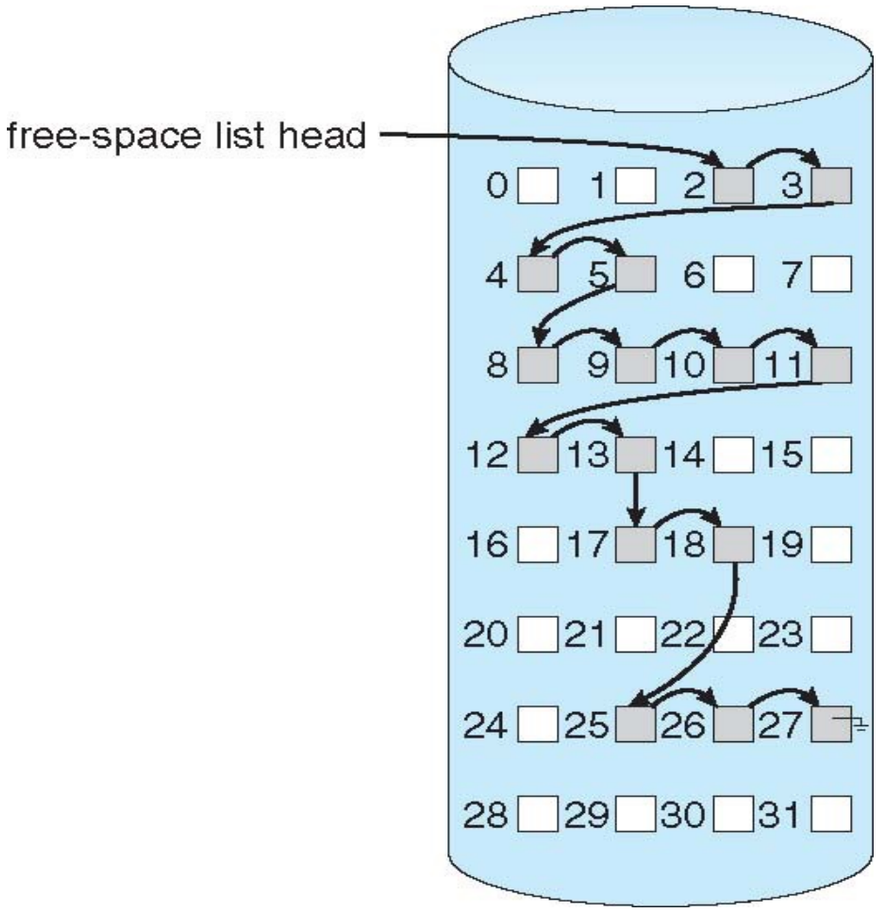
  block size = 4KB = 2^{12} bytes

  disk size = 2^{40} bytes (1 terabyte)

  $n = 2^{40}/2^{12} = 2^{28}$ bits (or 256MB);

Autres techniques de gestion de l'espace libre

- Liste chaîné (free list) avec un pointeur de tête de liste spécial
  - Pas de gaspillage d'espace
  - Impossible d'obtenir facilement de l'espace contigu
  - Traverser la liste est lent
  - Peut utiliser quelque chose comme un FAT
- Grouping
  - Liste chaîné avec *n - 1* pointeur vers les blocs vides (similaire à l'indexation)
  - La dernière adresse dans le bloc est un pointeur vers le bloc suivant qui contient des pointeurs de blocs libres
- Counting
  - L'espace est fréquemment utilisé et libéré de manière contiguë, avec une allocation contiguë, des étendues ou un regroupement
  - Keep address of first free block and a count of following (contiguous) free blocks
    - (numéro de bloc, nombres de blocs vides)



L’efficacité du disque

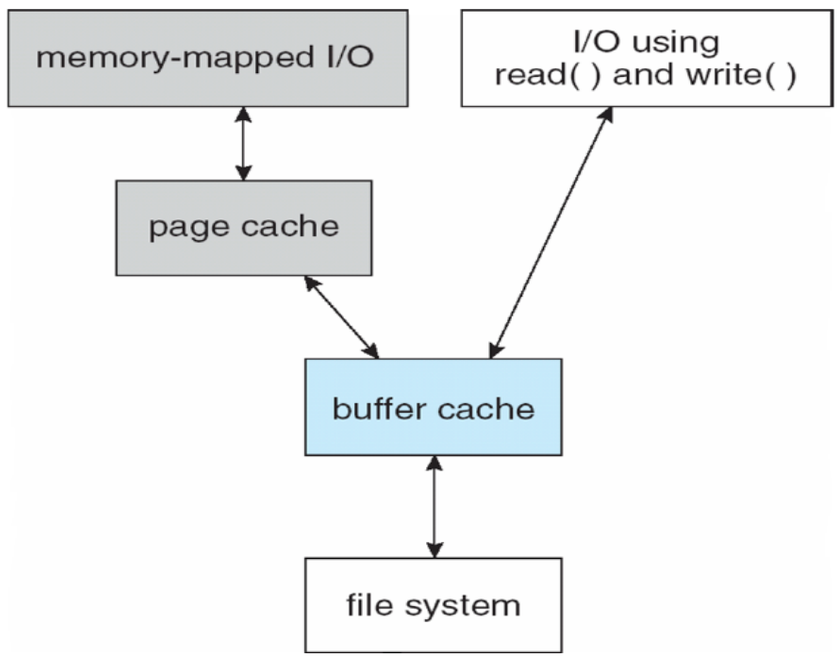
- Efficacité dépendante de:
  - Allocation de disque et algorithmes de répertoires
    - ✓ Par exemple, pré-allouer des inodes étalés sur le disque de telle sorte que les données seront plus tard près de son bloc inode
  - Types de données conservés dans l'entrée du répertoire du fichier
  - Par exemple, l'heure de la dernière lecture d'un fichier nécessite d'écrire des données dans l'inode du fichier (coûteux pour lire un fichier)
  - Attribution préalable ou allocation de structures de métadonnées selon les besoins
  - Structures de données de taille fixe ou variable
    - ✓ Par exemple, la taille du pointeur dans divers tableaux, l'évolution de la technologie et des besoins

Performances du disque

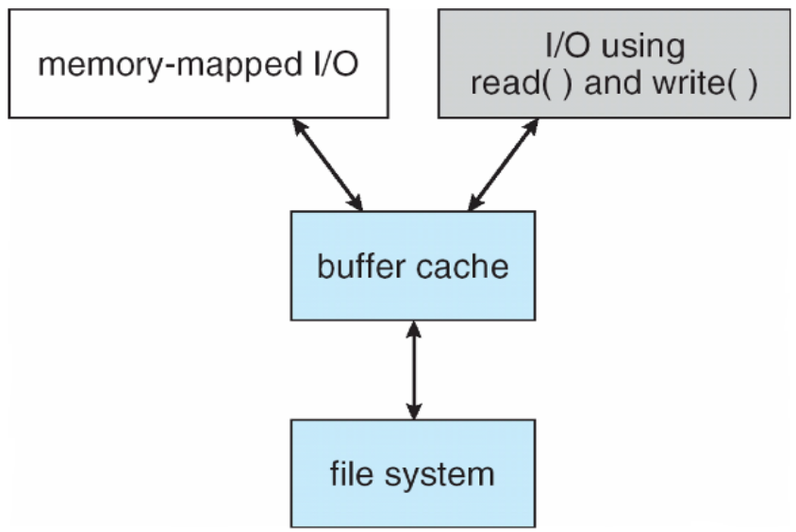
- Garder les données et les métadonnées proches les unes des autres
- **Buffer cache** - section séparée de la mémoire principale pour les blocs fréquemment lues du disque
- Synchronous vs. asynchronous écrit sur le disque
  - Écriture **synchrone** parfois demandée par les applications ou requise par le système d'exploitation, par exemple, pour les métadonnées
    - Aucune mise en mémoire tampon / mise en cache autorisée - les écritures doivent être enregistrées sur le disque avant l'accusé de réception
  - Les écritures **asynchrones** sont plus courantes, plus tamponnées, plus rapides
- Les techniques **free-behind** et **read-ahead** pour optimiser l'accès séquentiel (contrairement à LRU) car nous savons que la page précédente ne sera plus utilisée, et les pages suivantes seront bientôt nécessaires
- Contre-intuitif: lit fréquemment plus lent que les écritures, car les écritures sont asynchrones et optimisées par le contrôleur de disque, et les lectures peuvent inclure des lectures anticipées.

Unified Buffer Cache

- Un **cache de pages** met en cache des pages plutôt que des blocs de disque à l'aide de techniques et d'adresses de mémoire virtuelle
- Les E / S mappées en mémoire (memory-mapped I/O) utilisent un cache de pages
- Les E / S de routine via le système de fichiers utilisent le cache (disque)
- Un **unified buffer cache** utilise le même cache de pages pour mettre en cache les pages mappées en mémoire et les E / S ordinaires du système de fichiers afin d'éviter la “**double caching**”
  - Àttention à la cohérence des cache



Pas de unified buffer cache



avec unified buffer cache

Menu

- Structure du systèmes de fichiers
- Implémentations de répertoire de fichiers
- Méthodes d'allocations d'espace
- Gestion de l'espace libre
- **Fiabilité et performance**
- NFS

Cohérence en cas de panne

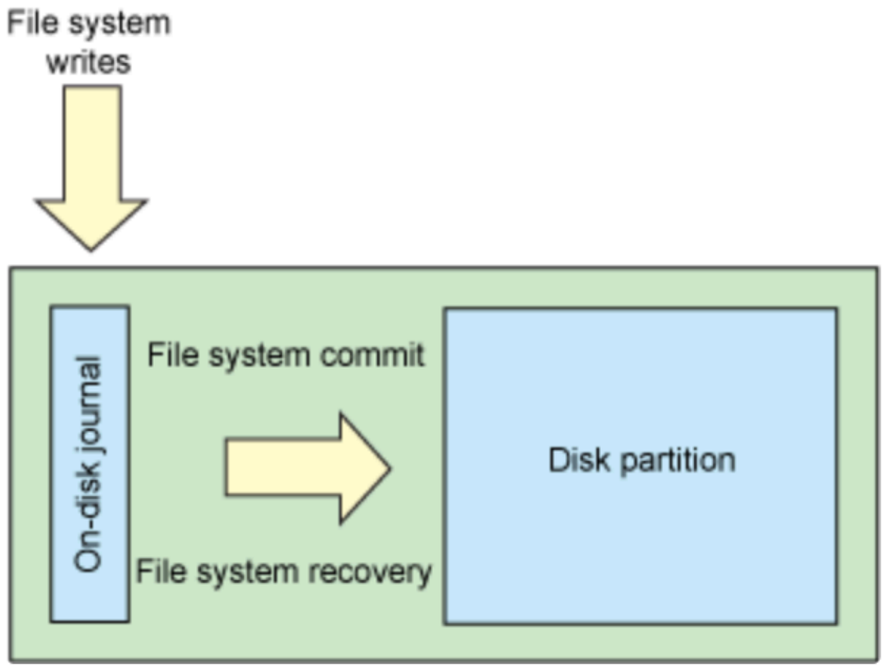
- Ajouter un bloc à un fichier requiert les modifications suivantes sur le disque:
  - Écrire le bloc
  - Mettre à jour l'inode (FCB)
  - Mettre à jour la liste des blocs libres
- Que se passe-t-il si le système s'arrête à mi-course?
- Créer ou détruire un fichier est encore plus compliquer
- Différente formes d'incohérence:
  - Espace disque libre mais inutilisables
  - Modifications perdues
  - Fichier en cours de modification corrompus
  - Victimes innocents

Vérification de la cohérence

- Comparez les données dans la structure du répertoire avec les blocs de données sur le disque et essayez de corriger les incohérences
  - e.g. fsck sur Linux
  - Ne peut pas faire de miracles
  - Couteux en temps, retarde le redémarrage après une panne
  - Il faut ordonnancer aves soin les écritures
    - UNIX écrit de manière synchrone résulte de l'allocation d'espace ou des modifications de métadonnées (non sûrs aux pannes)

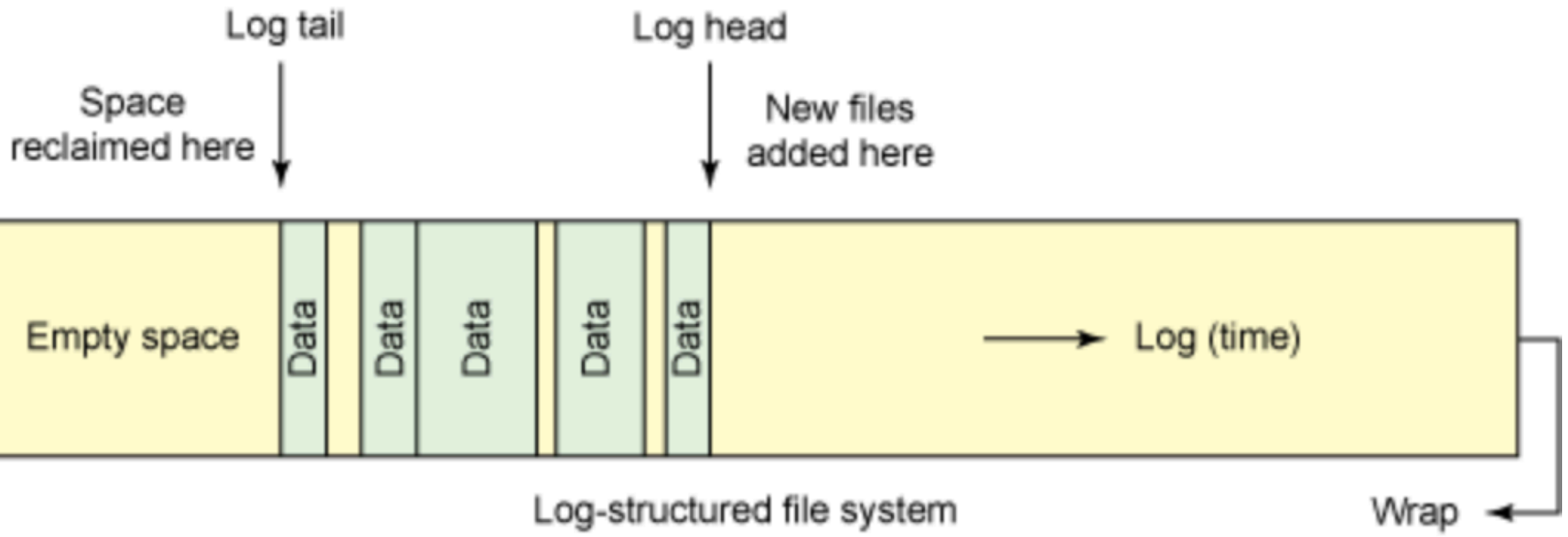
Journaling file systems

- Avant toute modification, écrire une description de la modification
- Le journal (ou log) garde la description des modifications en cours
- Modifications complétées peuvent être retirées du log
- Il faut quand même ordonnancer les écritures avec soin:
  - Écrire le log sur le disque avant d'effectuer les autres écritures
- Peut doubler le nombre d'écritures
- Une description peut être: data+metadata (physical journaling) ou juste metadata (logical journaling)
- Logical journaling
  - Écriture de block de la mémoire sur le disque
  - Mettre à jour l'inode dans le journal
  - Copiez l'inode du journal sur le disque
- Physical journaling
  - Copiez toutes les données dans le journal
  - Marquer comme engagé
  - Copiez toutes les données sur le disque



Log-structured file systems

- Comme journaling mais on ne garde que le journal (le journal **est** le système fichier)
  - Fonctionne sur le principe de la copy-on-write - les blocs de données ne sont pas écrasés mais réécrits
- Avantages:
  - Écritures par gros segments séquentiels: rapides!
  - Les anciennes données ne sont pas écrasées immédiatement (snapshots)
- Inconvénients
  - Les inodes ne sont plus fixes: il faut un inode-map
  - Il faut récupérer les blocs obsolete (garbage collector)
  - La lecture peut être plus lente



Menu

- Structure du systèmes de fichiers
- Implémentations de répertoire de fichiers
- Méthodes d'allocations d'espace
- Gestion de l'espace libre
- Fiabilité et performance
- **NFS**

Système de fichiers par réseau

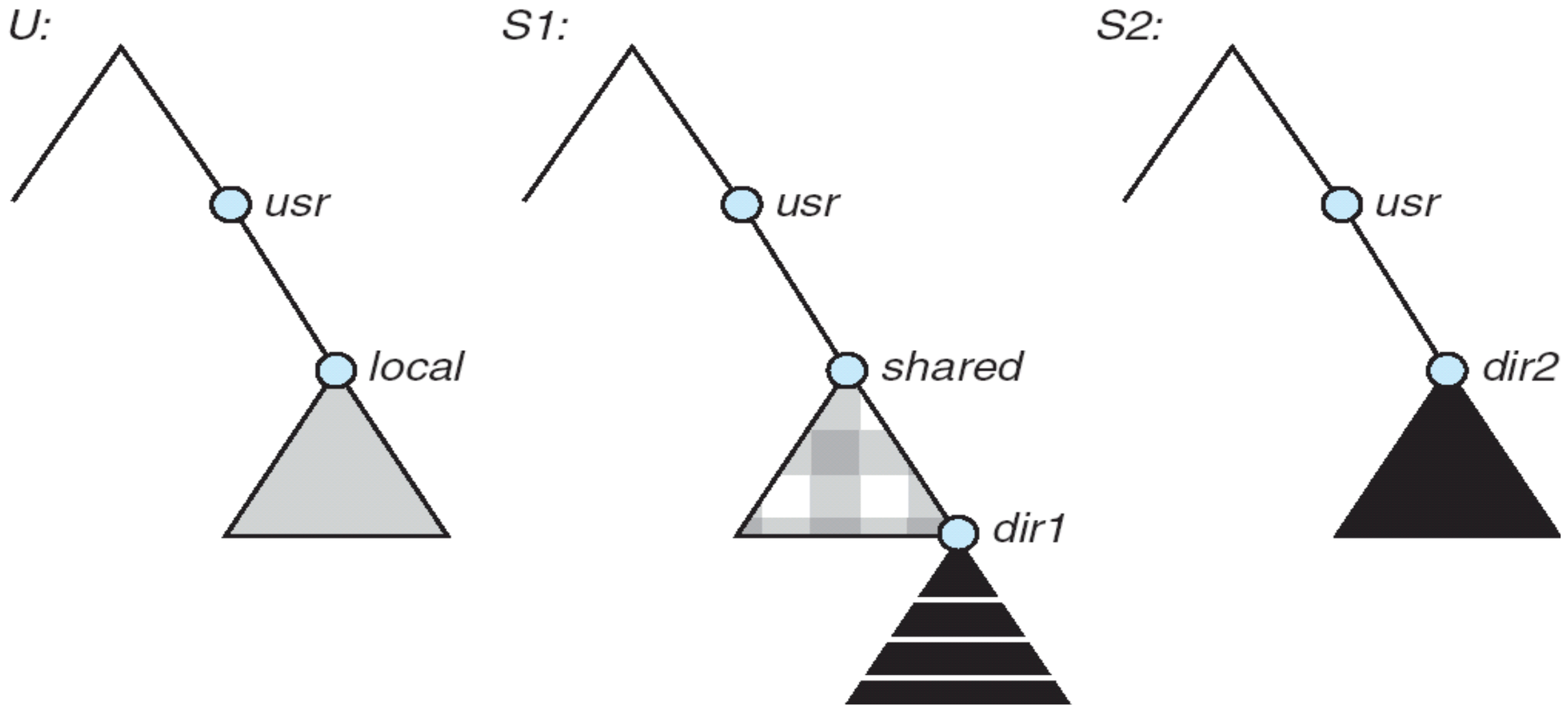
- Accéder “localement” aux fichiers d’une autre machine
- Idée: transférer les appels VFS par RPC

- Problèmes:
  - Fichiers modifiés hors de notre contrôle
  - Gestion du cache
  - Gestion des pannes
  - Authentication
  - Désaccords entre VFS et le protocole utilisé

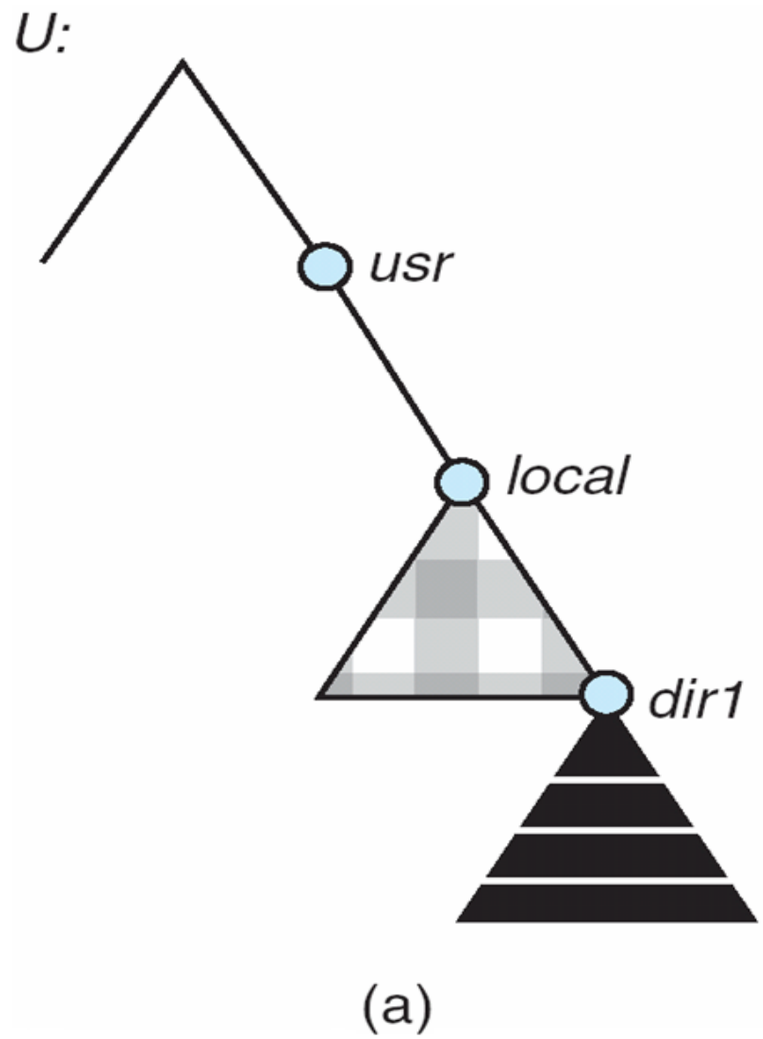
Network File System (NFS)

- Une implémentation et une spécification d'un système logiciel permettant d'accéder à des fichiers distants sur des réseaux locaux (ou WAN)
- Noeds interconnectés considérés comme un ensemble de machines indépendantes avec des systèmes de fichiers indépendants, ce qui permet le partage entre ces systèmes de fichiers de manière transparente
  - Un répertoire distant est monté (mount) sur un répertoire de système de fichiers local
    - Le répertoire monté ressemble à un sous-arbre intégral du système de fichiers local, remplaçant le sous-arbre descendant du répertoire local
  - La spécification du répertoire distant pour l'opération de montage est non transparente; le nom d'hôte du répertoire distant doit être fourni
    - Les fichiers du répertoire distant peuvent ensuite être consultés de manière transparente
  - Soumis à l'accréditation de droits d'accès, potentiellement n'importe quel système de fichiers (ou répertoire dans un système de fichiers), peut être monté à distance sur n'importe quel répertoire local

Trois systèmes de fichiers indépendants

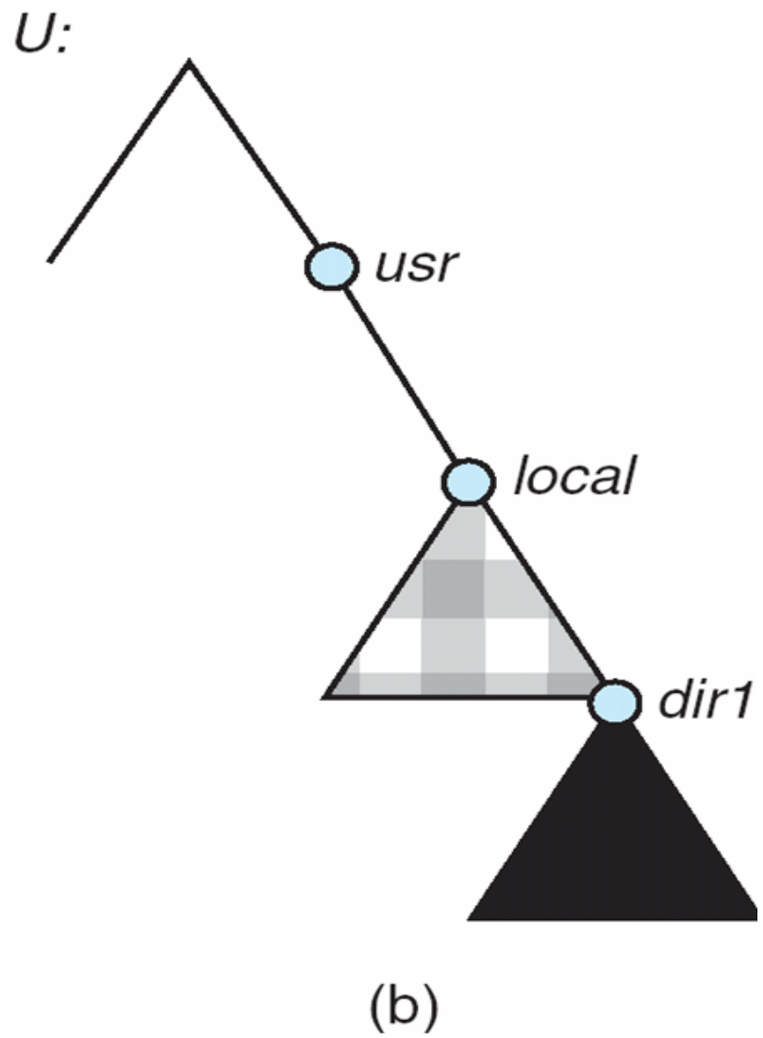


Cascading mounts



(a)

Mounts



(b)

“Cascading mounts”

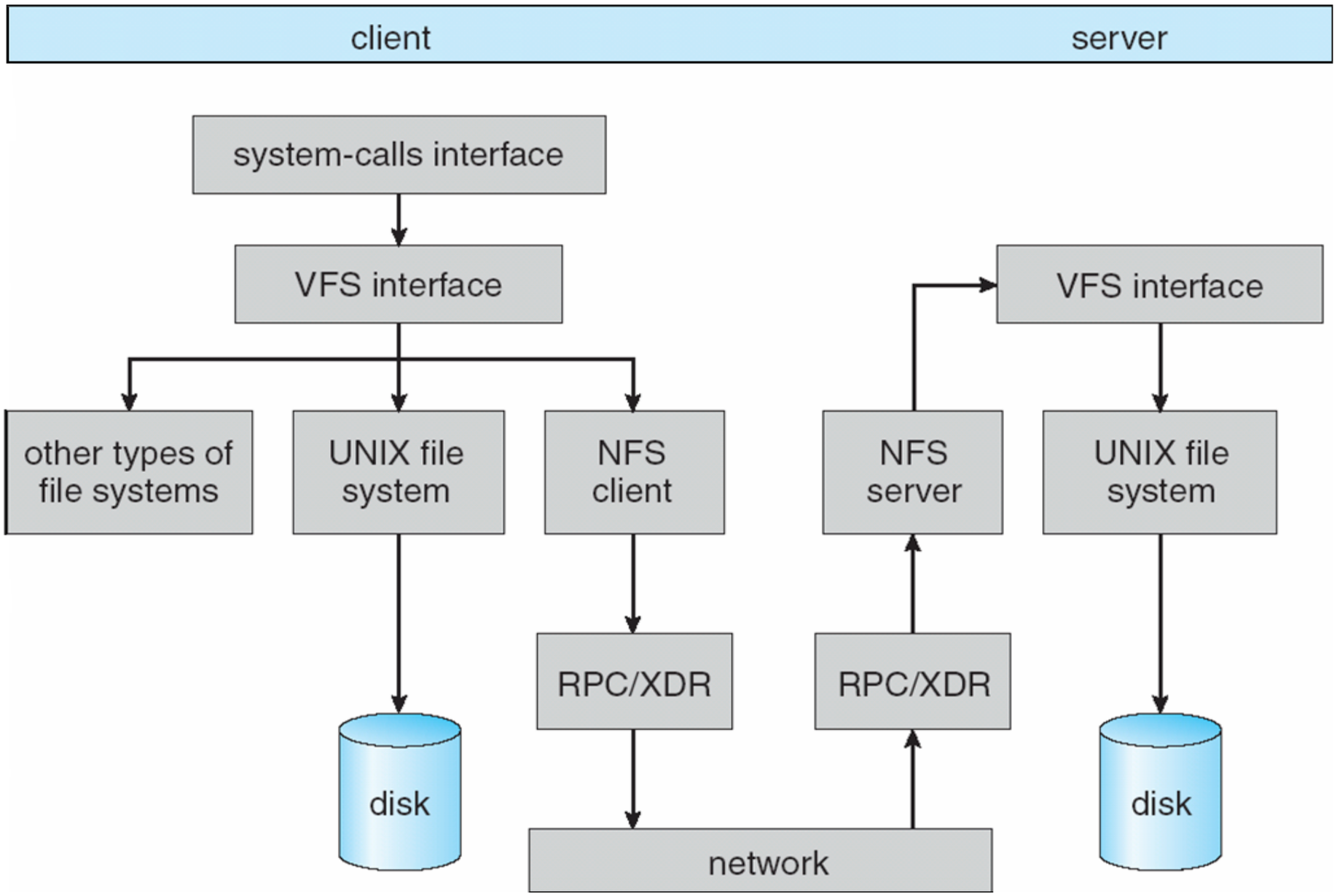
NFS Mount

- Établit la connexion logique initiale entre le serveur et le client
- L'opération de montage inclut le nom du répertoire distant à monter et le nom de la machine serveur le stockant
  - La requête de montage est mappée au RPC correspondant et transmise au serveur de montage en cours d'exécution sur la machine du serveur
  - Liste d'exportation - spécifie les systèmes de fichiers locaux que le serveur exporte pour le montage, ainsi que les noms des machines autorisées à les monter
- Suite à une demande de montage conforme à sa liste d'exportation, le serveur renvoie un handle de fichier, une clé pour d'autres accès.
- Fichier handle - un identificateur de système de fichiers et un numéro d'inode pour identifier le répertoire monté dans le système de fichiers exporté
- L'opération de montage modifie uniquement la vue de l'utilisateur et n'affecte pas le côté serveur

NFS Protocol

- Fournit un ensemble d'appels RPC pour les opérations de fichiers à distance. Les procédures prennent en charge les opérations suivantes:
  - rechercher un fichier dans un répertoire
  - lire un ensemble d'entrées de répertoire
  - manipuler des liens et des répertoires
  - accéder aux attributs de fichier
  - lire et écrire des fichiers
- Les serveurs NFS sont sans état; chaque requête doit fournir un ensemble complet d'arguments
- Les données modifiées doivent être validées sur le disque du serveur avant que les résultats ne soient retournés au client (perdre les avantages de la mise en cache)
- Le protocole NFS ne fournit pas de mécanismes de contrôle de concurrence

Schematic View of NFS Architecture



Sommaire

- Les systèmes de fichiers sont composés de couches et nécessitent plusieurs structures de données sur disque et en mémoire pour fonctionner
- Une considération clé est comment le disque est alloué aux fichiers
- De plus, l'espace libre doit être géré de manière efficace
- Certains mécanismes et systèmes de fichiers sont capables de gérer les défaillances, par ex. systèmes avec “journalling”
- L'interface du système de fichiers réseau permet d'accéder aux systèmes de fichiers distants via une interface client-serveur