Système d'Exploitation

Sous-système de Gestion Fichiers

Université de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Licence Sciences et Technologies
Mention : Informatique
2ème Année

Mohamed TAGHELIT
taghelit@univ-tours.fr

Plan

1. Concepts de fichier et Répertoires
2. Méthodes d'accès aux fichiers
3. Méthodes d'allocation des fichiers
4. Gestion de l'espace libre
5. Système de gestion des fichiers Unix

Introduction aux Systèmes de Fichiers

- Le Système de Gestion de Fichiers est la partie la plus visible d'un SE
- Le SGF fournit les mécanismes :
  - Stockage des
  - Accès aux

  } programmes et données
- Le SGF se compose principalement de deux parties distinctes :
  - Une collection de fichiers : chacun stockant des données
  - Une structure de répertoires : qui organise et fournit les informations sur les fichiers du système
  - [Les partitions] : utilisées pour séparer physiquement et logiquement une grande collection de répertoires

Concept de Fichier

- Fichier → unité de stockage logique
  - Collection nommée d'informations, enregistrée sur un stockage secondaire
  - Plus petite allocation de stockage secondaire logique (d'un point de vue utilisateur)
  - Représente généralement des programmes et des données
  - Les informations contenues dans un fichiers peuvent être de plusieurs sortes :
    - Programme source,
    - Programme objet,
    - Programme exécutable,
    - Données numériques,
    - Texte,
    - Images graphiques,
    - Enregistrements sonores
    - …

Attributs d'un Fichier

Peuvent varier d'un SE à un autre :

- **Nom** : nom symbolique du fichier, seule information conservée dans un format compréhensible
- **Identifiant** : généralement un nombre qui identifie le fichier à l'intérieur du système de fichiers
- **Type** : information nécessaire pour les systèmes qui supportent différents types de fichiers
- **Emplacement** : constitue un pointeur sur un périphérique et sur l'emplacement du fichier sur ce périphérique
- **Taille** : la taille actuelle du fichier et éventuellement la taille maximum autorisée
- **Protection** : contrôle d'accès au fichier (lecture, écriture, exécution)
- **Heure**, **Date** : informations concernant la création, la dernière modification et la dernière utilisation du fichier
- **Identification de l'utilisateur** : informations utiles dans la protection, la sécurité et la surveillance des utilisations

Les informations concernant tous les fichiers se retrouvent dans la structure des répertoires.

Opérations sur les Fichiers

Un fichier est un type de donnée abstrait qui nécessite, pour le définir correctement, de tenir compte des opérations qui peuvent lui être appliquées.

Opérations de base :

- **Création d'un fichier** : nécessite deux étapes : l'allocation de l'espace destiné au nouveau fichier et création d'une nouvelle entrée dans le répertoire.
- **Écriture dans un fichier** : nécessite l'indication du nom du fichier et les informations à y enregistrer. Le système doit conserver un *pointeur* d'écriture de l'emplacement dans le fichier de la prochaine écriture.
- **Lecture dans un fichier** : nécessite l'indication du nom du fichier et l'endroit (en mémoire) où le prochain bloc du fichier doit être placé. Le système doit conserver un *pointeur* de lecture de l'emplacement dans le fichier de la prochaine lecture.
- **Repositionnement dans un fichier** : le *pointeur* d'écriture/lecture dans le fichier est initialisée à une valeur donnée.
- **Suppression d'un fichier** : implique la recherche du fichier dans le répertoire et la suppression de son entrée (effacement qui la rend à nouveau utilisable pour un autre fichier).
- **Troncature d'un fichier** : efface le contenu d'un fichier tout en gardant tous ses attributs (excepté la taille) et libération de son espace.

La plupart des opérations de base sur les fichiers impliquent la recherche de l'entrée concernée dans le répertoire → le système conserve des tables des fichiers ouverts

Structure des Fichiers

- La structure logique des enregistrements des fichiers est déterminée par la manière dont ils sont accédés. Pour cela, plusieurs critères sont à prendre en considération :
  - Un accès rapide,
  - Une mise à jour aisée,
  - Économie de l'espace de stockage,
  - Maintenance simple,
  - Fiabilité.

- L'importance donnée à chacun de ces critères dépend des applications qui vont utiliser les fichiers,

- Ces critères peuvent être conflictuels

- Plusieurs organisation sont proposées, parmi lesquelles :
  - Le fichier séquentiel,
  - Le fichier direct,
  - Le fichier indexé.

Méthodes d'Accès aux Fichiers

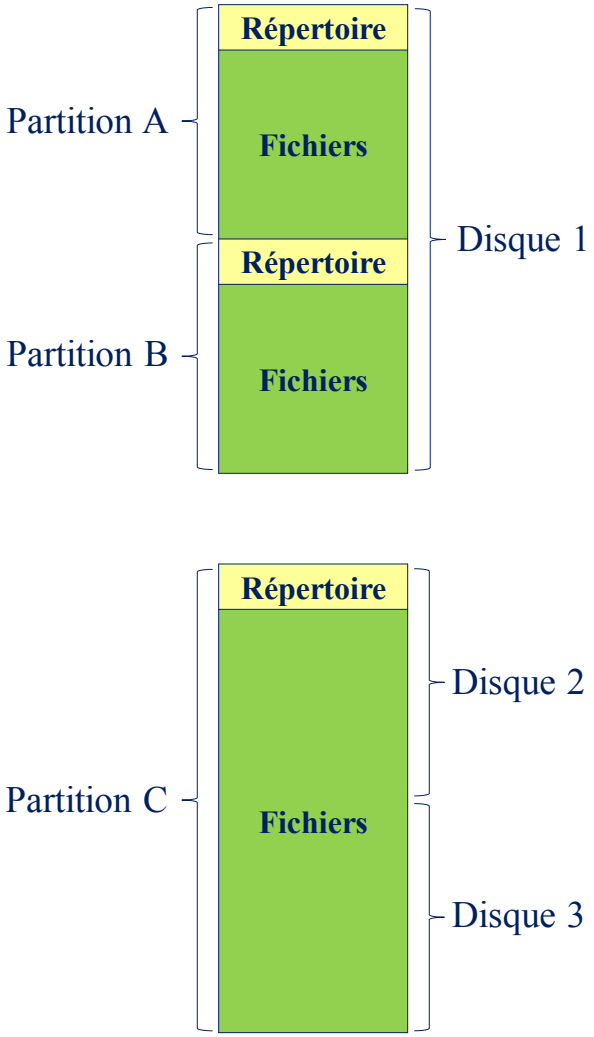
- Les informations dans un fichier peuvent être accédées de plusieurs façons :
  - **Accès séquentiel** :
    - Méthode d'accès la plus simple,
    - Les informations du fichiers sont traitées dans l'ordre, enregistrement par enregistrement,
    - Mode d'accès le plus courant (éditeurs, compilateurs, …).
  - **Accès direct** :
    - Le fichier est constitué d'enregistrements logiques de longueur fixe,
    - Lecture/Écriture rapide d'enregistrements dans un ordre quelconque.
  - **Accès direct indexé** :
    - Implique généralement la construction d'un index du fichier,
    - l'index contient les pointeurs vers les différents blocs.

Organisation des Données

Il est nécessaire d'organiser la masse importante de données :

- Les disques sont organisés en partitions :
  - Une partition est une structure hébergeant les fichiers et les répertoires,
  - Pour certains systèmes, une partition peut être utilisée pour fournir une zone indépendante du disque et/ou pour grouper plusieurs disques.

- Chaque partition contient des informations sur les fichiers :
  - Une partition est considérée comme un périphérique de stockage individuel.
  - Les informations sont conservées dans les entrées d'un répertoire (de périphérique).
  - Le répertoire mémorise des informations (nom, emplacement, type, taille) de tous les fichiers d'une partition.



Contenu des Répertoires

- Pour chaque fichier, les informations suivantes sont contenues dans un répertoire :

Informations de base

- **Nom du fichier** : nom tel que choisi par le créateur (utilisateur ou programme). Doit être unique dans un répertoire donné.
- **Type du fichier** : texte, binaire, module chargeable, …
- **Organisation du fichier** : pour les systèmes supportant différentes organisations.

Informations d'adressage

- **Volume** : indique l'unité sur laquelle le fichier est stocké.
- **Adresse de début** : adresse physique de début sur l'unité de stockage secondaire (par exemple, cylindre, piste, et numéro de block sur le disque.
- **Tailles utilisée, allouée** : les tailles courante du fichier (en octets, mots ou blocks) et maximale.

Informations de contrôle

- **Propriétaire** : utilisateur auquel est donné le contrôle sur ce fichier.
- **Actions autorisées** : contrôle la lecture, l'écriture et l'exécution.

Informations D'utilisation

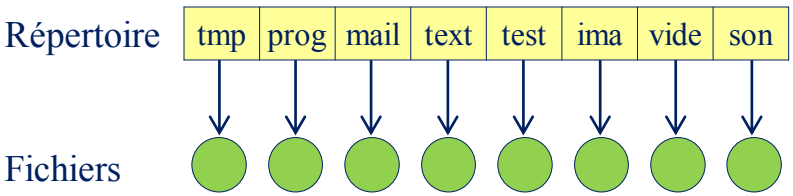
- **Date de création** : indique quand le fichier a été placé pour la première fois dans le répertoire.
- **Identité du créateur** : souvent le propriétaire courant mais pas nécessairement.
- **Dates dernier accès, dernière modification** : date de la dernière lecture, modification, insertion ou suppression.
- **Utilisation courante** : activités courantes sur le fichier (ouvert, verrouillé, modifié en mémoire mais pas encore sur disque).

Opérations sur les Répertoires

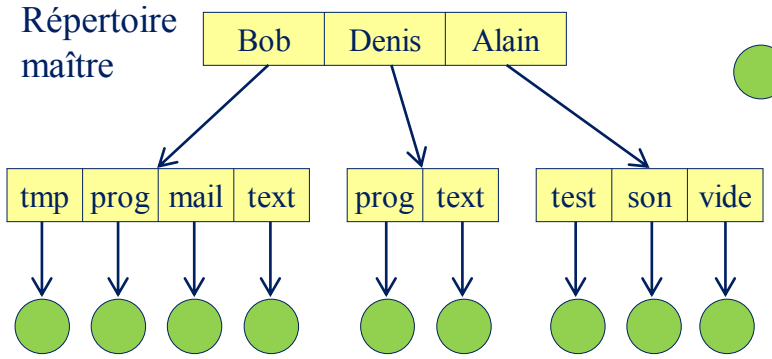
Opérations de base :

- **Recherche d'un fichier** : quand un utilisateur ou un programme référence un fichier, le répertoire doit être parcouru pour trouver l'entrée correspondant à ce fichier.
- **Création d'un fichier** : quand un fichier est créé, une nouvelle entrée doit être ajoutée au répertoire.
- **Suppression d'un fichier** : quand un fichier est supprimé, son entrée doit être supprimée du répertoire.
- **Listage du répertoire** : tout ou une partie du contenu du répertoire peut être requis. Cette requête est généralement faite par un utilisateur et résulte en un listage des fichiers concernés ainsi que leurs attributs (type, informations sur le contrôle d'accès, utilisation courante, …).
- **Mise à jour du répertoire** : certains attributs de fichier étant stockés dans le répertoire, la modification de l'un d'entre eux entraine la modification de l'entrée correspondante dans le répertoire.

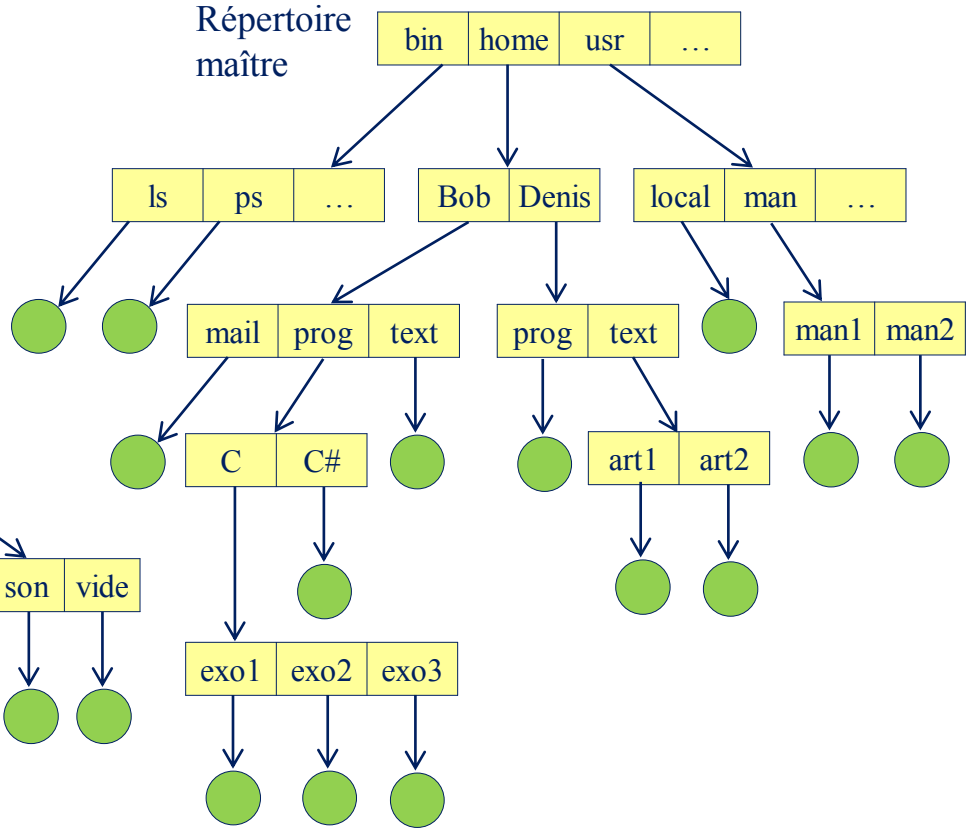
Structure des Répertoires



Répertoire à un niveau

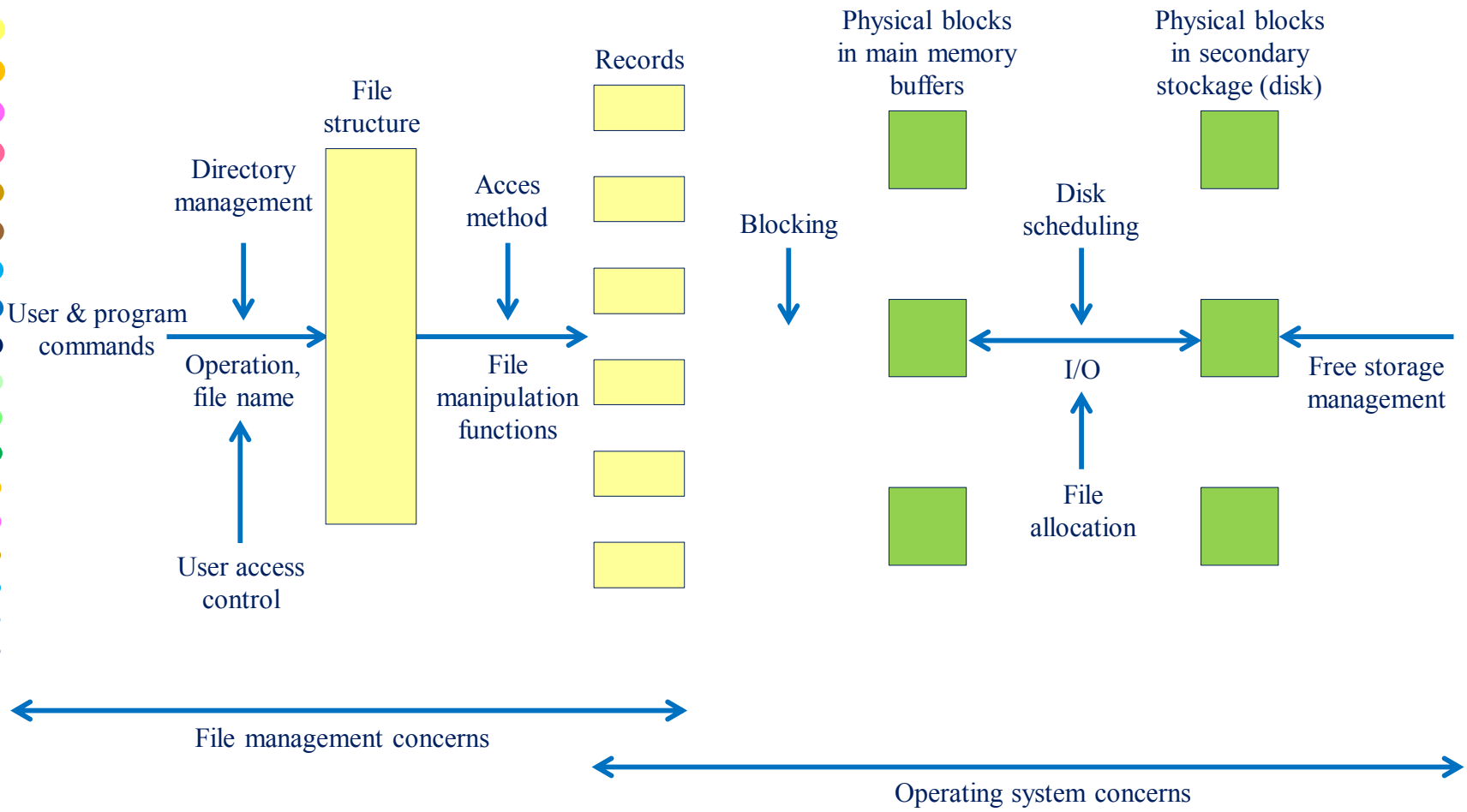


Répertoire à deux niveaux



Arborescence de répertoires

Fonctions d'un SGF



Relations Enregistrement - Bloc

Pour accomplir les E/S, les enregistrements doivent être organisés en blocs. Plusieurs questions doivent être examinées :

- **Blocs de taille fixe ou variable** :

  Dans la plupart des systèmes, les blocs sont de taille fixe. Cela simplifie :
  - Les entrées/sorties
  - L'allocation des tampons en mémoire principale
  - L'organisation des blocs en mémoire secondaire

- **Taille des blocs par rapport à celle des enregistrements** :

  L'idée est : plus est grand le bloc, plus le nombre d'enregistrements transférés en une opération d'E/S est grand.
  - Si le fichier est parcouru séquentiellement :
    - Le nombre d'opérations d'E/S est réduit,
    - La vitesse de traitement est augmentée.
  - Si le fichier est parcouru aléatoirement :
    - Transfert inutile d'enregistrements non utilisés,

  La combinaison de la fréquence d'opérations séquentielles et du principe de localité, fait que le temps de transfert est réduit lorsque des blocs de grande taille sont utilisés.

Méthodes de “Mise en Blocs”

- **Fixed blocking** :

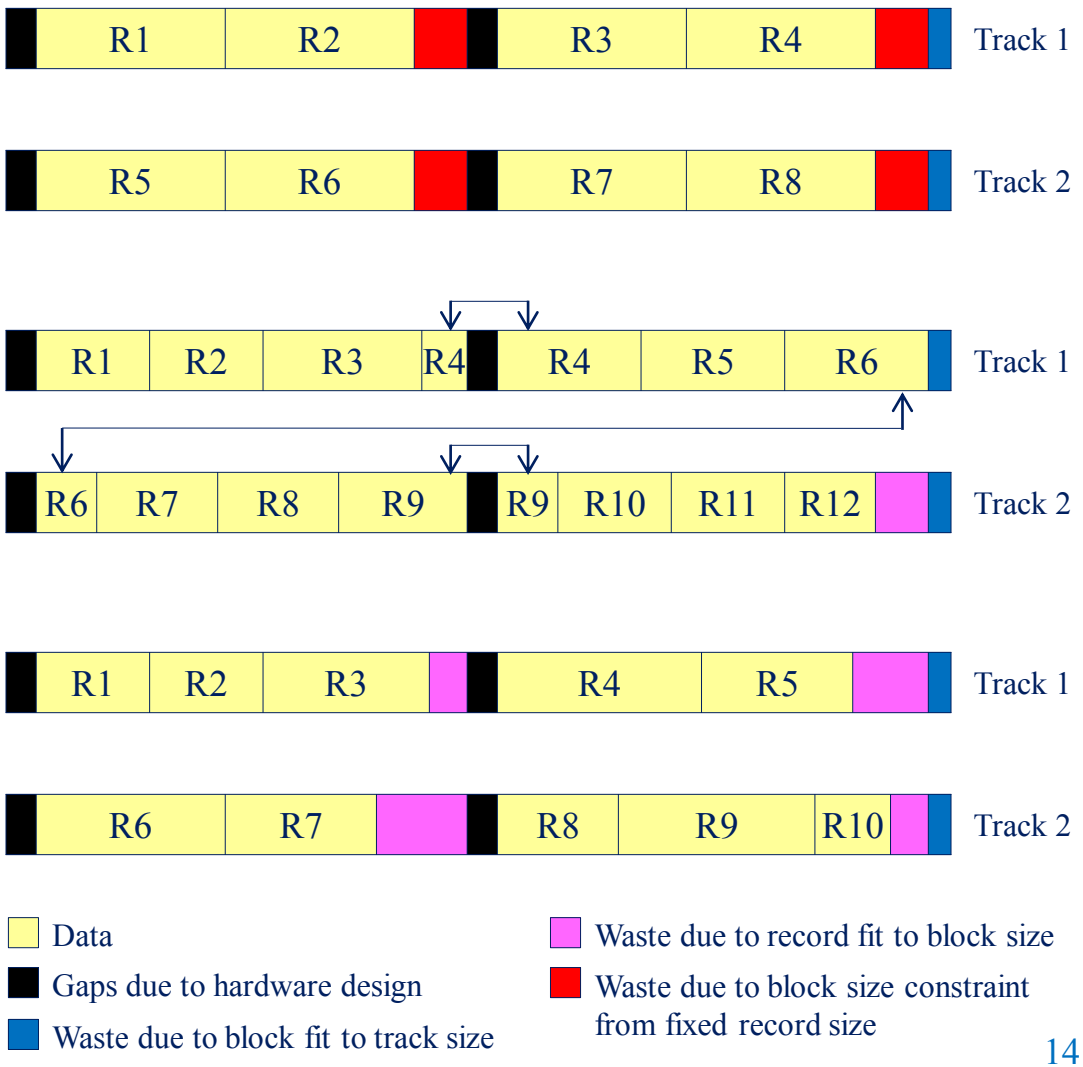
  Des enregistrements de taille fixe sont utilisés, et un nombre entier d’enregistrements sont stockés dans un bloc. Il peut y avoir un espace non utilisé à la fin de chaque bloc (fragmentation interne).

- **Variable-length spanned blocking** :

  Des enregistrements de taille variable sont utilisés et stockés dans les blocs sans espace non utilisé. Ainsi, certains enregistrements peuvent s’étaler sur 2 blocs (la suite indiquée par un pointeur vers le bloc suivant).

- **Variable-length unspanned blocking** :

  Des enregistrements de taille variable sont utilisés mais le chevauchement n’est pas utilisé. Il y a une perte d’espace dans la plupart des blocs due à la non utilisation du reste d’un bloc si le prochain enregistrement est plus grand que ce dernier.



Allocation de fichiers

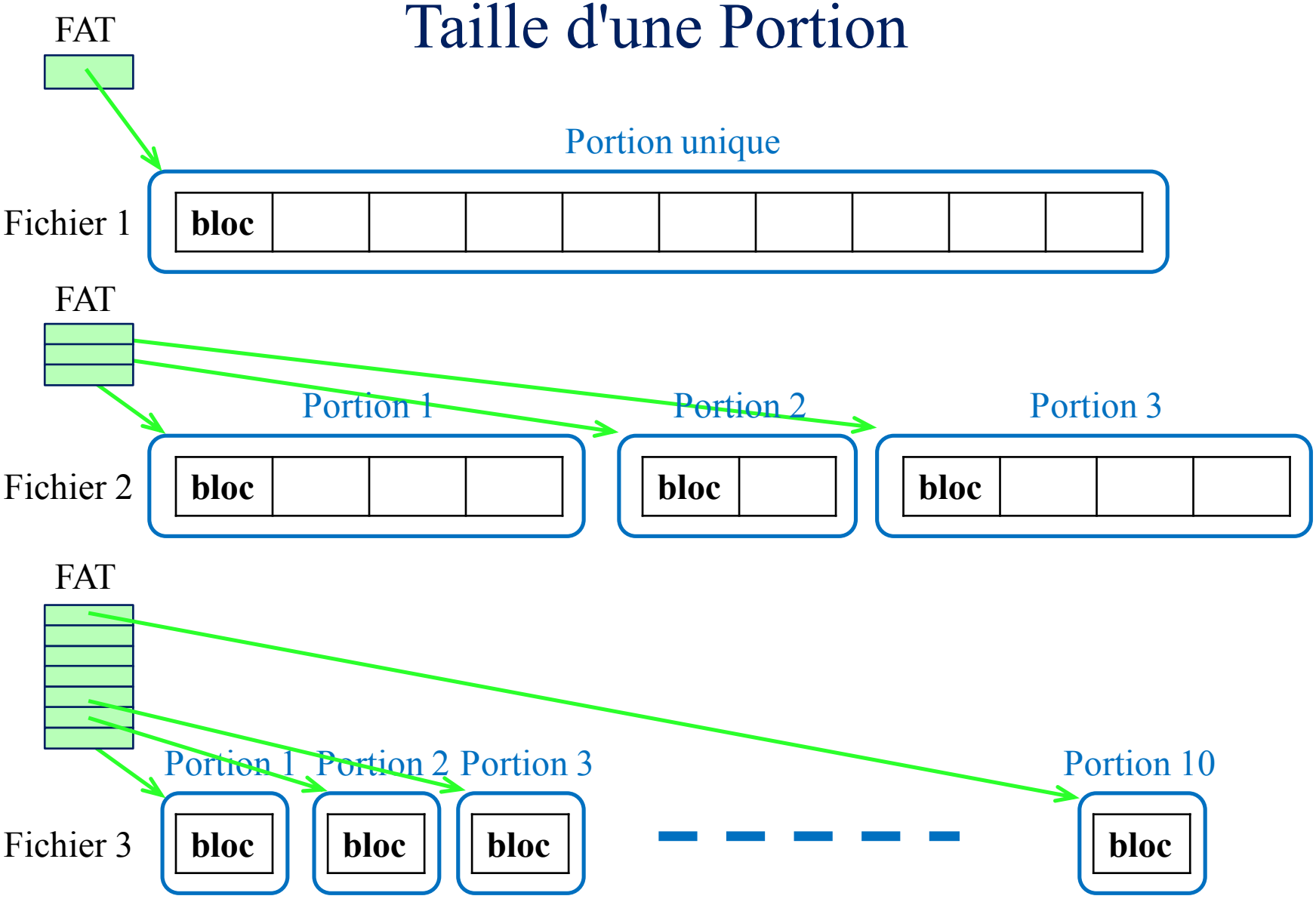
- En mémoire secondaire, un fichier consiste en une collection de blocs. Le SE (ou le SGF) est responsable de l'allocation des blocs aux fichiers :
  - l'espace en mémoire secondaire doit être alloué aux fichiers,
  - il est nécessaire de garder trace de l'espace disponible pour allocation.
- L'allocation de fichiers soulèvent les problèmes suivants :
  - Quand un nouveau fichier est créé, est-ce que le maximum d'espace requis par le fichier est alloué en une seule fois ?
  - L'espace alloué à un fichier est constitué d'une ou plusieurs portions contigües. La taille d'une portion peut varier d'un bloc à la totalité du fichier. Quelle doit être la taille d'une portion pour l'allocation des fichiers ?
  - Quel type de structures de données ou de table est utilisé pour garder trace des portions assignées à un fichier ? Une telle table est généralement référencée comme "file allocation table" (FAT).

Allocation de fichiers

- **Pré-allocation vs Allocation dynamique**
  - Pré Allocation
    - La taille maximale doit être déclarée à la création du fichier (difficile d'estimer la taille, voire impossible, dans la plupart des cas)
    - Souvent, la taille doit être surestimé (pour ne pas en manquer) . Ceci entraîne une perte d'espace en mémoire secondaire.
  - Allocation Dynamique
    - Résolve les problèmes de la pré-allocation. Allocation d'espace à un fichier selon les besoins.

- **Taille d'une portion**
  - Compromis entre : une seule portion assez grande pour contenir la totalité du fichier et l'espace disque alloué bloc par bloc à chaque fois ⇒ compromis entre : l'efficacité d'un point de vue d'un fichier et l'efficacité de la totalité du système.
    - La contiguïté augmente les performances,
    - Un grand nombre de petites portions augmente la taille des tables nécessaires à la gestion de l'espace alloué,
    - Des portions de tailles fixe (par exemple blocs) simplifient la réallocation de l'espace,
    - Une taille variable ou une petite taille fixe minimisent la perte d'espace non utilisée.

Taille d'une Portion



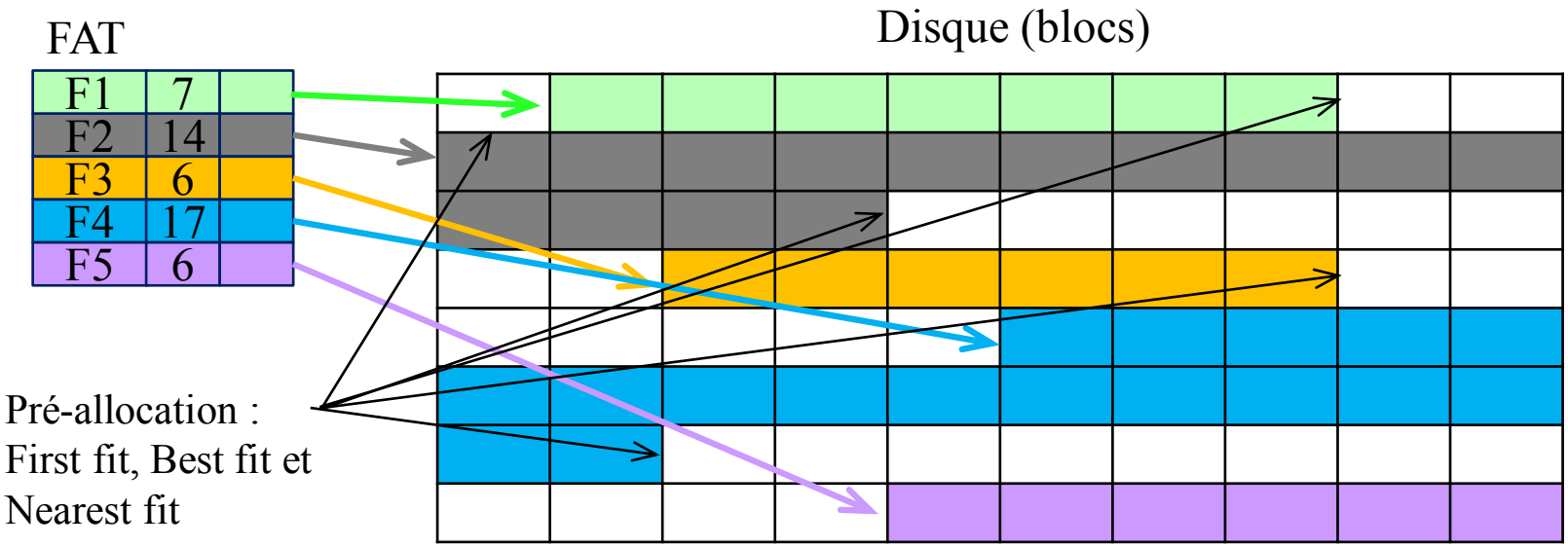
Allocation de fichiers

- **Deux alternatives**
  - Portions contigües de grande taille variable :
    - meilleures performances,
    - la taille variable évite la perte,
    - les tables d'allocation sont petites,
    - mais la réutilisation de l'espace est plus difficile.
  - Blocs :
    - plus grande flexibilité,
    - peut nécessiter de grandes tables ou des structures complexes,
    - et la contigüité est abandonnée.

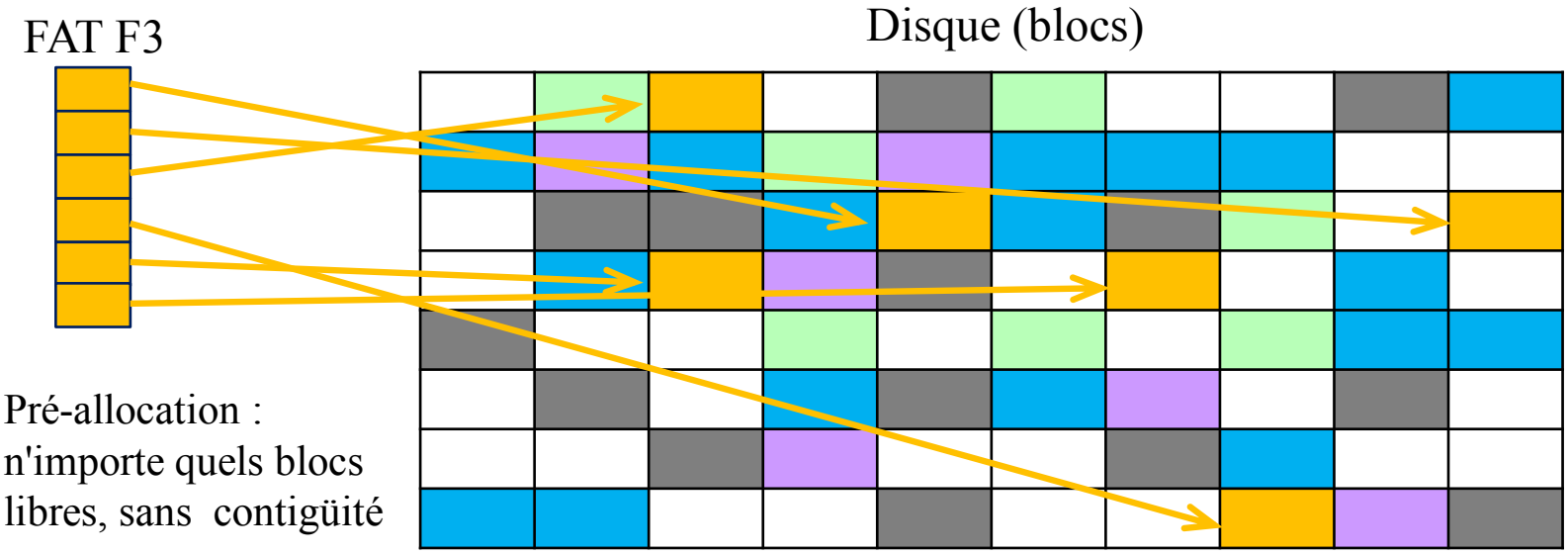
Toutes deux compatibles avec la pré-allocation ou l'allocation dynamique

- À un fichier est pré-alloué un groupe de blocs contigus
  - Élimine la nécessité d'une table d'allocation (pointeur sur le 1er bloc et le nombre de blocs alloué),
- Fragmentation de l'espace libre
  - First fit, Best fit et Nearest fit.
- À un fichier sont pré-alloués, en une seule fois, toutes les portions requises
  - La table d'allocation reste de taille fixe,

Pré-allocation avec portions de grandes tailles



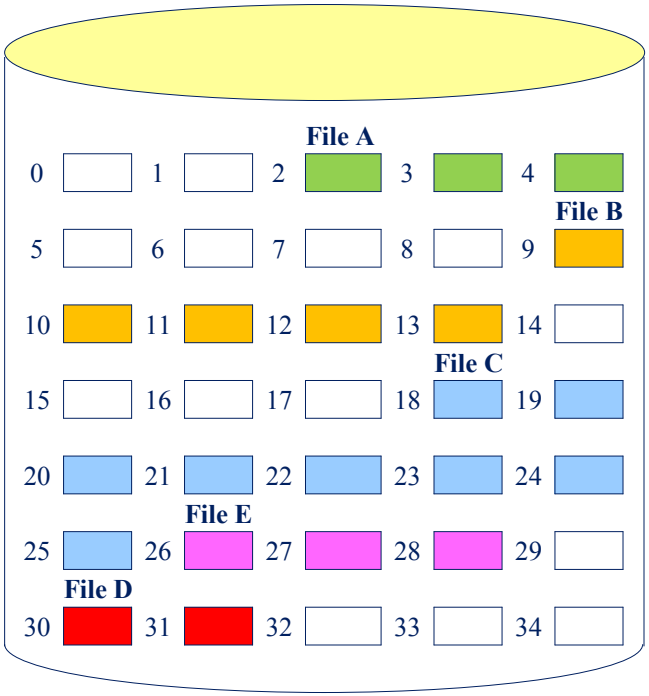
Pré-allocation par bloc



Méthode d'Allocation Contigüe

- **Allocation contigüe**
  - Un seul ensemble de blocs contigus est alloué à la création du fichier,
  - Stratégie de pré-allocation avec portions de taille variable
  - La table d'allocation nécessite une seule entrée pour chaque fichier
    - Le bloc de début (b) et la taille du fichier,
    - Facilité de retrouver un quelconque bloc i (b + i – 1).
  - Meilleurs performance d'un point de vue d'un fichier
    - Plusieurs blocs peuvent être transférés en une seule fois, améliorant ainsi les performances d'E/S des traitements séquentiels
  - Fragmentation externe impliquant une difficulté à trouver des blocs contigus d'un espace de taille suffisante
    - Compactage nécessaire
  - Problème lié à la pré-allocation : estimation de la taille du fichier à sa création

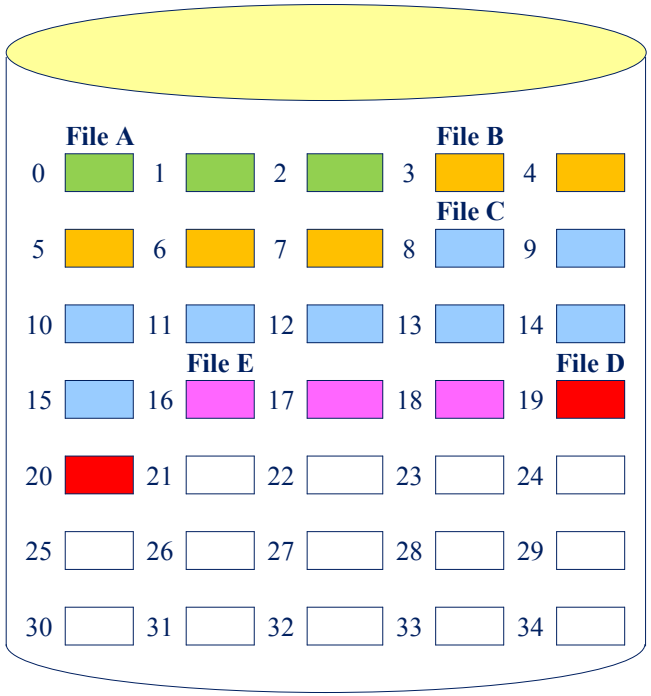
Exemple d'Allocation Contigüe



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Contiguous File Allocation



File Allocation Table

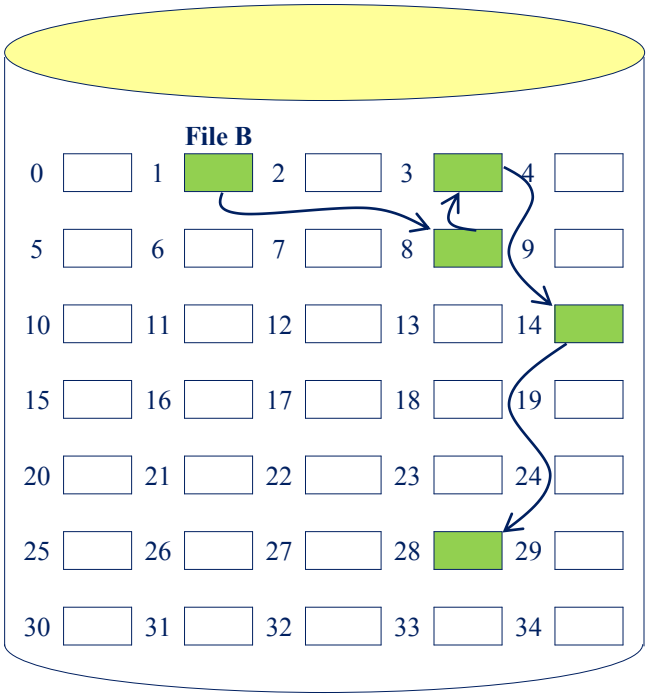
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Contiguous File Allocation (After Compaction)

Méthode d'Allocation Chaînée

- **Allocation chaînée**
  - Allocation sur la base d'un seul bloc à la fois,
  - Chaque bloc contient un pointeur vers le prochain bloc de la chaîne,
  - La table d'allocation nécessite une seule entrée pour chaque fichier,
    - Le bloc de début et la taille du fichier
  - Bien que la pré allocation soit possible, il est commun d'allouer les blocs en fonction des besoins,
  - La sélection des blocs est simplifiée,
    - N'importe quel bloc libre est ajouté à la chaîne
  - Absence de fragmentation externe,
  - Organisation physique adaptée aux fichiers séquentiels devant être traités séquentiellement,
  - La sélection d'un bloc donné du fichier nécessite le parcours de la chaîne jusqu'au bloc désiré,
  - Ne tire par profit du principe de localité,
    - S'il est nécessaire, à un moment, de transférer plusieurs blocs du fichier, alors une série d'accès à différents endroits du disque sont requis
    - Consolidation (périodique) des fichiers pour y remédier

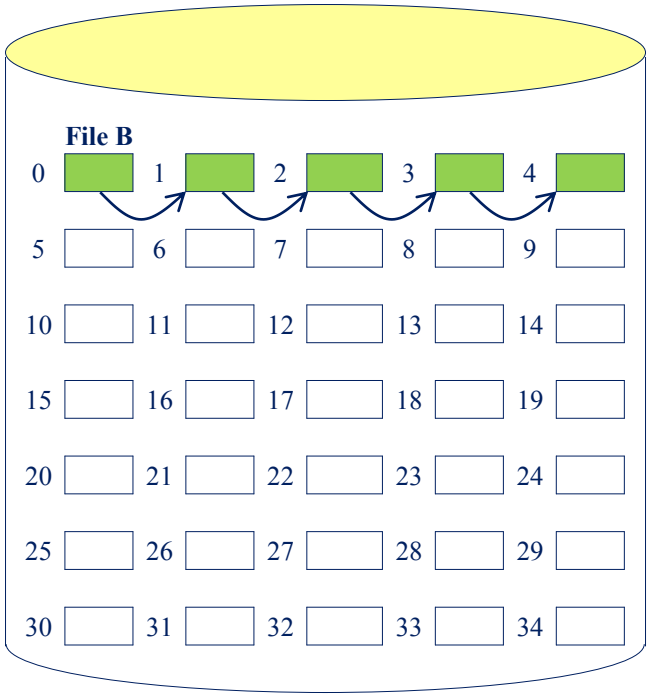
Exemple d'Allocation Chaînée



File Allocation Table

File Name	Start Block	Length
…	…	…
File B	1	5
…	…	…
…	…	…

Chained Allocation



File Allocation Table

File Name	Start Block	Length
…	…	…
File B	0	5
…	…	…
…	…	…

Chained Allocation
(After Consolidation)

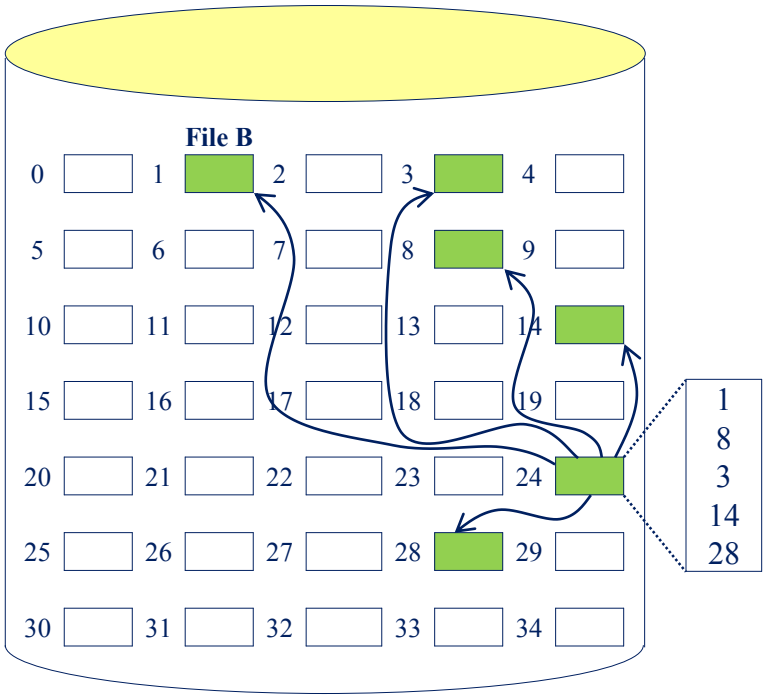
Méthode d'Allocation Indexée

- **Allocation indexée**
  - Résolve plusieurs problèmes liés aux allocations contigüe et chaînée,
  - La table d'allocation contient, pour chaque fichier, un pointeur vers un **bloc index**,
    - Le bloc index possède une entrée pour chaque portion allouée au fichier,
  - L'allocation peut se faire sur la base de :

    Portions de taille variable
    - Améliore la localité
    - Consolidation des fichier (réduit la taille du bloc index)

    Blocs
    - élimine la fragmentation externe
    - Consolidation des fichier
  - Supporte aussi bien l'accès séquentiel que l'accès direct aux fichiers (méthode la plus populaire)

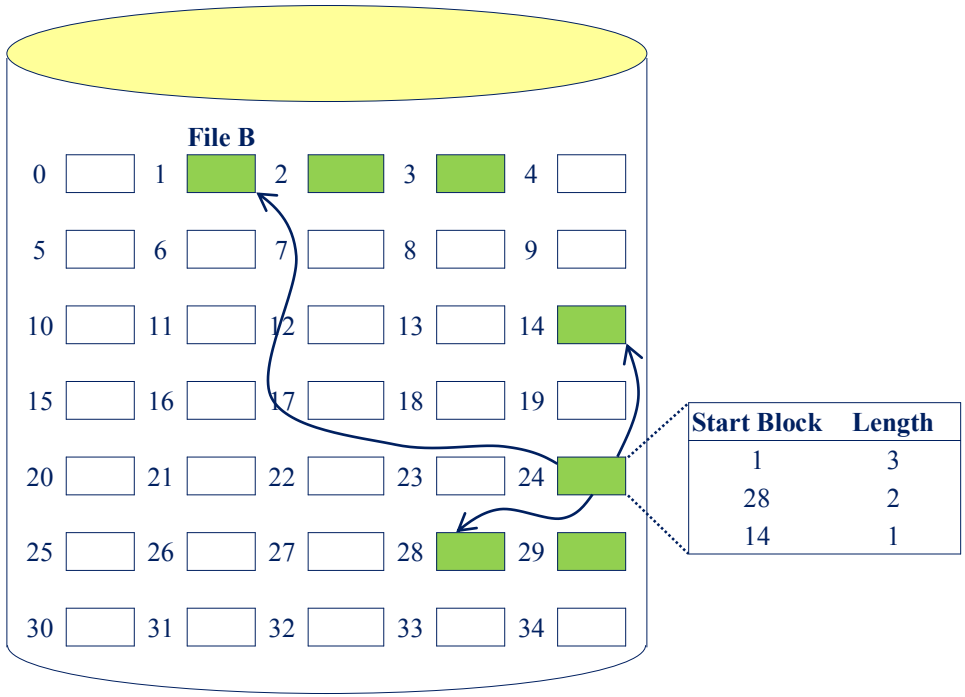
Exemple d'Allocation Indexée



File Allocation Table

File Name	Index Block
…	…
File B	24
…	…
…	…

Indexed Allocation
with Block Portions



File Allocation Table

File Name	Index Block
…	…
File B	24
…	…
…	…

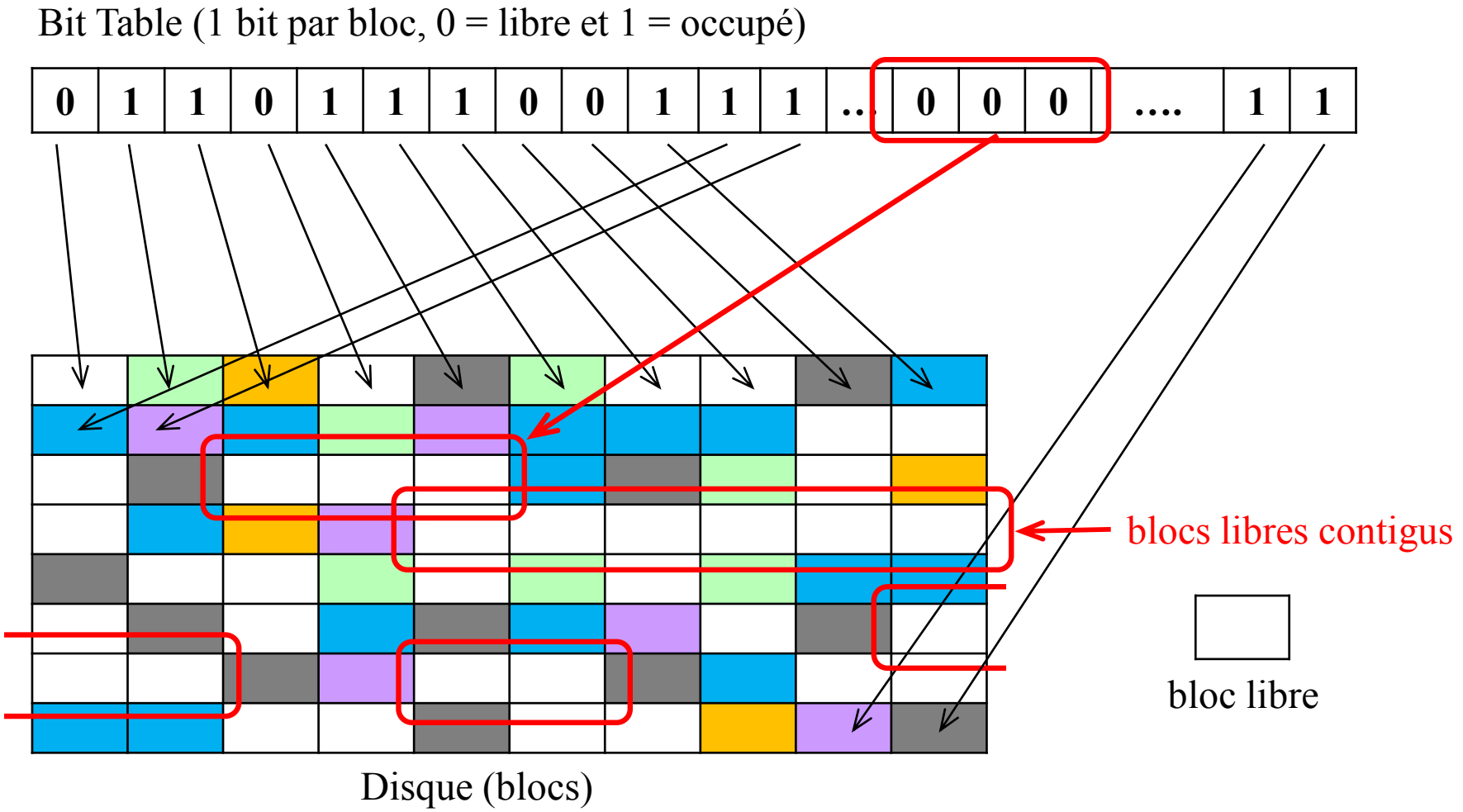
Indexed Allocation
with Variable-Length Portions

Gestion de l'Espace Libre

- De la même façon que l'espace qui est alloué aux fichiers doit être géré, l'espace qui n'est pas actuellement alloué à aucun fichier doit être géré.
- Pour accomplir une quelconque des méthodes d'allocation précédentes, il est nécessaire de connaître quels blocs du disque sont libres.
- Une table d'allocation du disque est nécessaire en plus de la table d'allocation des fichiers.
- Plusieurs techniques sont implémentées :

  - **Bit Tables**
    - Cette technique utilise un vecteur contenant un bit pour chaque bloc sur le disque (0 = libre, 1 = alloué)
    - Facilité de recherche d'un ou plusieurs blocs libres contigus,
    - Applicable pour toute méthode d'allocation de fichiers,
    - De taille aussi petite que possible, mais reste importante (disque = 16Gbyte, bloc = 512 octets $\Rightarrow$ table = 4 Mbytes)

Gestion de l'espace libre par Bit Table



Gestion de l'Espace Libre

- **Portions Libres chaînées**

  - Les portions libres sont chaînées entre elles en utilisant un pointeur et une taille dans chaque portion libre,

  - Surcharge en espace négligeable (la table d'allocation disque n'est pas nécessaire), juste un pointeur vers le début de la chaîne et la longueur de la 1ère portion,

  - Adaptée à toutes les méthodes d'allocation de fichiers,

    Allocation par bloc → choix du 1er bloc en tête de chaîne et mise à jour du pointeur et, éventuellement, de la taille,

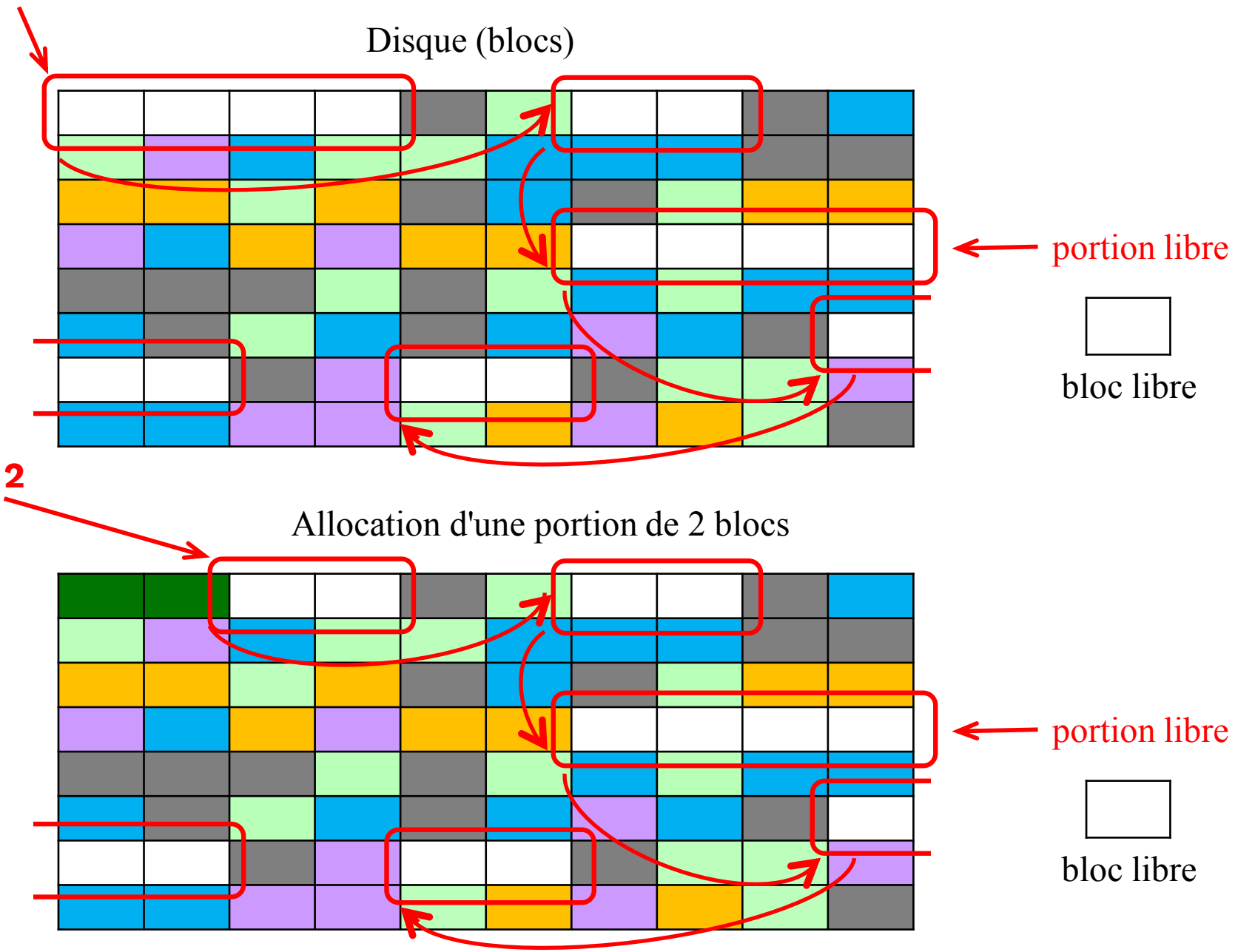
    Portions de taille variable → un algorithme First-fit peut être utilisé : les entêtes des portions sont examinés les uns après les autres pour retrouver dans la chaîne la portion libre suffisante. Pointeur et taille sont mis à jour.

  - Le disque finit par être fragmenté (éventuellement, chaque portion limitée à un bloc)

    Chaque allocation d'un bloc nécessite sa lecture (pour récupérer le pointeur sur le nouveau 1er bloc libre) avant son utilisation (écriture)

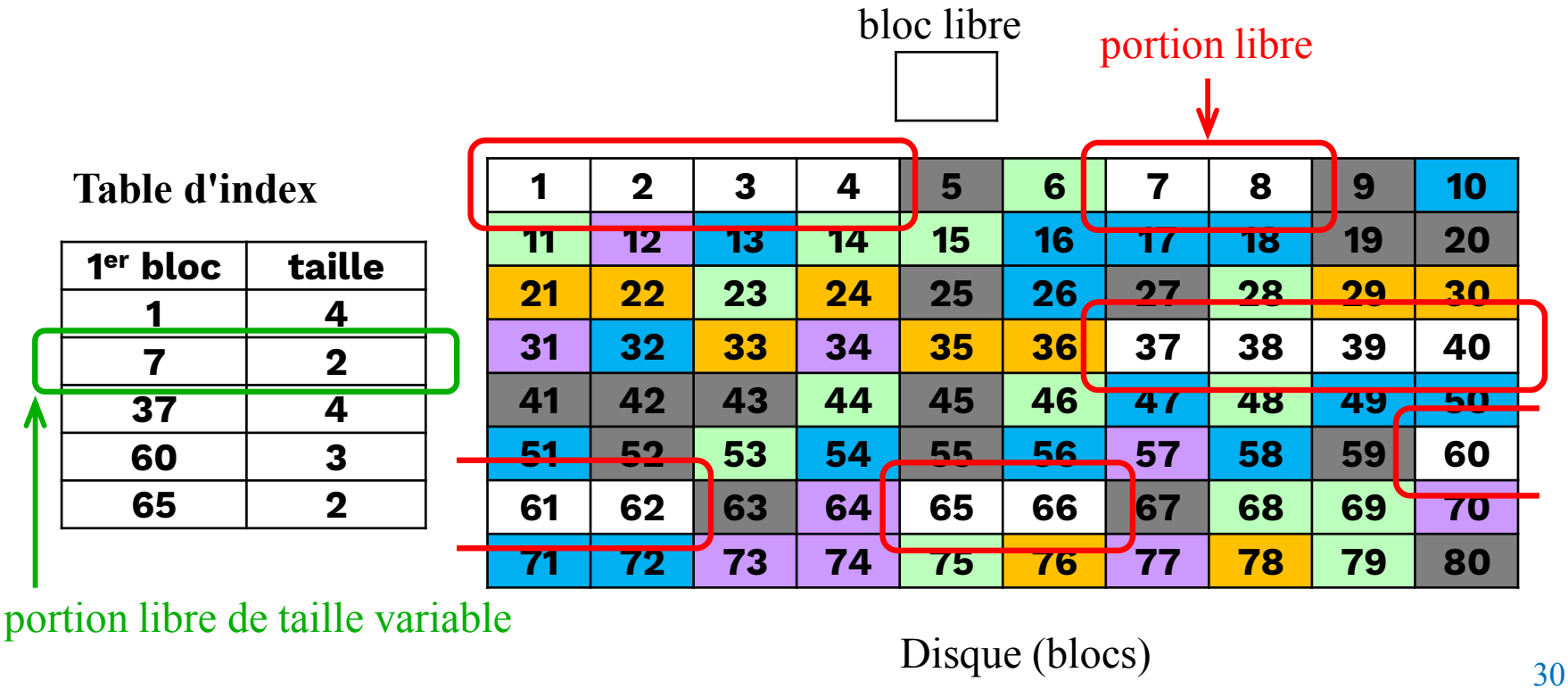
    Si, à un moment, plusieurs blocs individuels doivent être alloués pour une opération fichier, alors la création d'un fichier sera grandement ralentie. La suppression d'un fichier très fragmenté entrainera également beaucoup de temps.

Gestion de l'espace libre par portions libres chaînées



Gestion de l'Espace Libre

- **Indexation**
  - Cette technique traite l'espace libre comme un fichier et utilise une table d'index (tel que pour l'allocation indexée),
  - Par efficacité, l'index est sur la base de portions de taille variable (plutôt que blocs),
  - Une entrée dans la table pour chaque portion libre sur le disque.



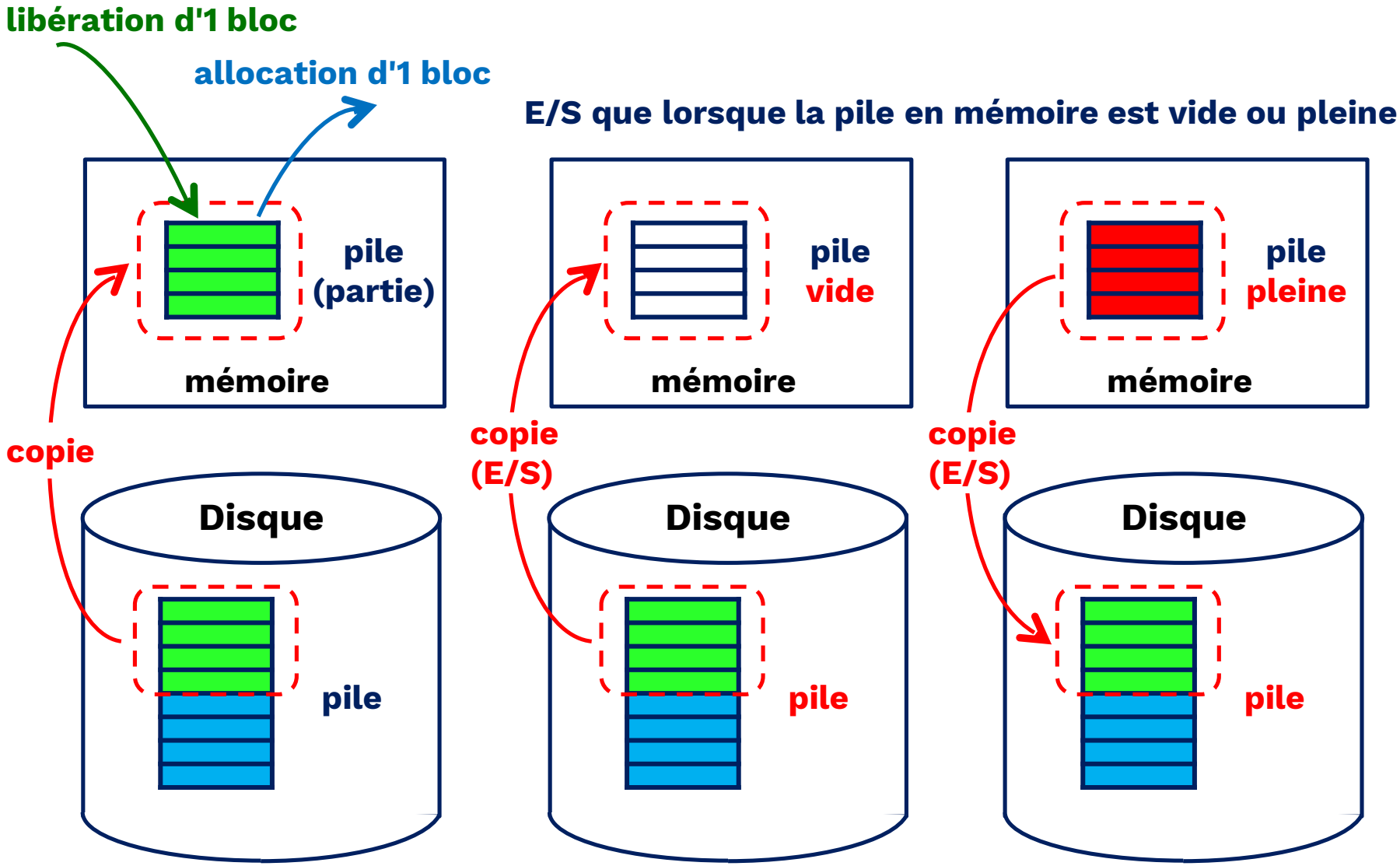
Gestion de l'Espace Libre

- **Liste des Blocs Libres**
  - Dans cette technique, à chaque bloc est assigné un nombre séquentiellement,
  - La liste de nombres de tous les blocs libres est maintenue dans une portion réservée du disque,
  - 24 bits (ou 32, selon la taille du disque) sont nécessaires pour stocker le nombre d'un seul bloc (24 ou 32 fois la taille de la **bit table** correspondante),
  - Technique assez satisfaisante en considérant les points suivants :
    - L'espace dédié à la liste est inférieur à 1% de la totalité de l'espace disque (4 bytes pour chaque bloc de 512 bytes, dans le cas d'un nombre sur 32 bits)
    - Bien que la liste soit trop importante pour être stockée en mémoire principale, il existe deux techniques efficaces pour stocker qu'une petite partie :

Techniques de Gestion de la Liste des Blocs Libres

- Chacune de ces deux techniques ne stocke en mémoire principale qu'une petite partie de la liste :
  - la liste est considérée comme une pile et seules les 1ères centaines des nombres de la liste sont maintenues en mémoire
    - Quand un nouveau bloc est alloué, il est dépiler du sommet de la pile (partie en mémoire principale),
    - Quand un bloc est dé-alloué, il est empilé sur la pile,
    - Il n'y a de transfert entre le disque et la mémoire que lorsque la partie de la pile en mémoire est soit vide soit remplie,
    - Cette technique induit presque aucun temps d'accès la plupart du temps.

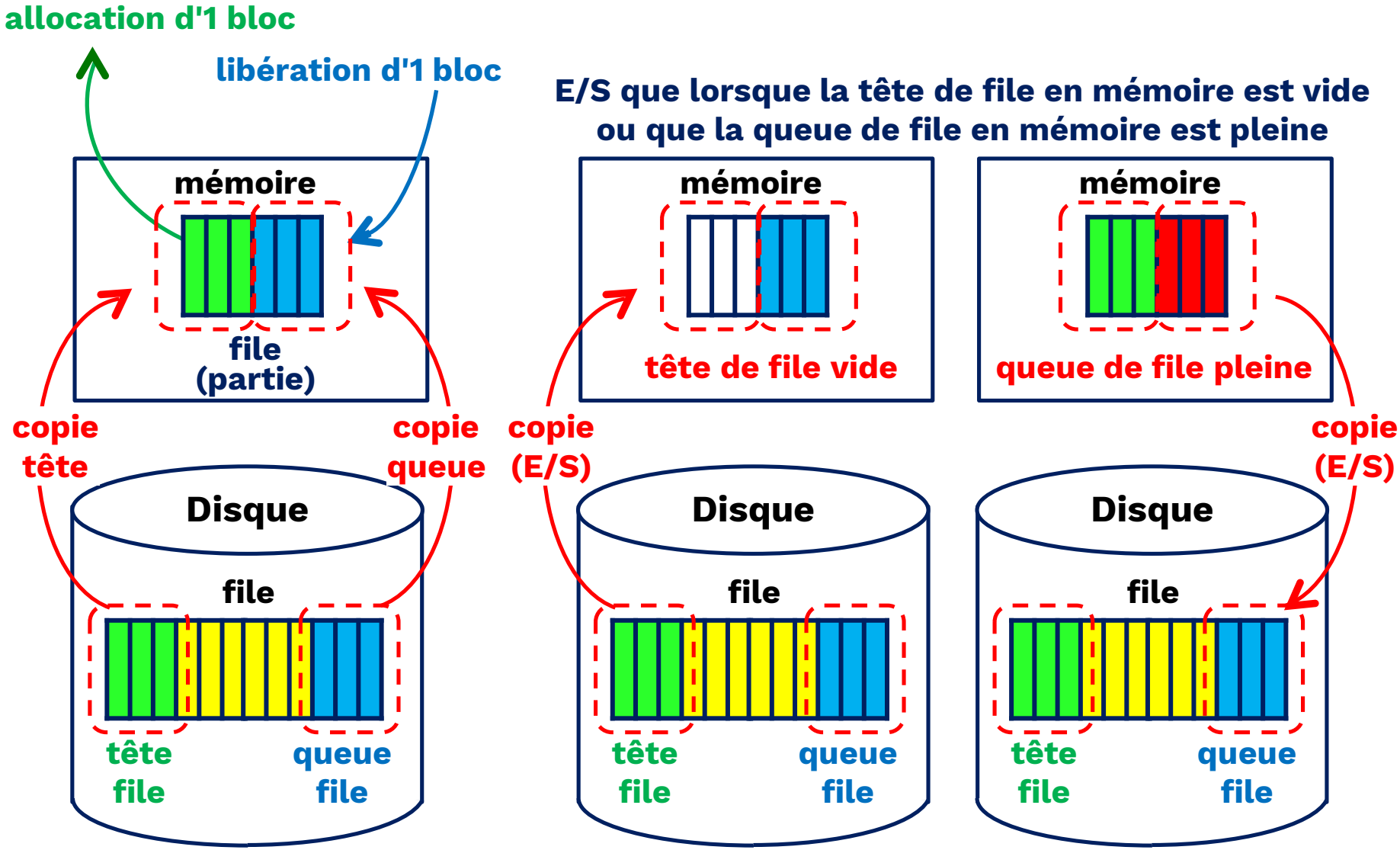
Gestion de l'espace libre par liste des blocs libres (Pile)



Techniques de Gestion de la Liste des Blocs Libres

- Chacune de ces deux techniques ne stocke en mémoire principale qu'une petite partie de la liste :
  - la liste est considérée comme une FIFO, avec quelques centaines d'entrées parmi la tête et la queue de la file en mémoire
    - Un bloc est alloué en prenant la 1ère entrée de la tête de la file (partie en mémoire principale),
    - Un bloc est dés-alloué en le mettant à la fin de la queue de la file (partie en mémoire principale),
    - Il n'y a de transfert entre le disque et la mémoire que lorsque la partie tête de la file en mémoire est vide ou lorsque la partie fin de file en mémoire est pleine.
- Pour chacune de ces deux techniques, un processus en arrière-plan peut trier la liste en mémoire pour favoriser l'allocation contigüe.

Gestion de l'espace libre par liste des blocs libres (File)



Gestion de Fichiers Unix

- Le noyau Unix considère les fichiers comme des flots d'octets,
- Toute structure logique interne est spécifique à l'application (qui l'utilise),
- Seule la structure physique des fichiers est prise en compte par Unix
- Quatre types de fichiers sont distingués par Unix :

  - **Ordinary** : fichiers contenant des informations introduites par un utilisateur, des programmes applicatifs ou des utilitaires système,

  - **Directory** : contient une liste de noms de fichiers et des pointeurs vers les **inodes** (index nodes) associés.
    - Sont organisés hiérarchiquement,
    - Sont considérés comme des fichiers ordinaires mais avec des protections en écriture particulières qui font que seul le SGF peut y écrire alors que l'accès en lecture est disponible pour les programmes utilisateurs.

  - **Special** : utilisés pour accéder des unités périphériques, tels que les terminaux et imprimantes. À toute unité d'E/S est associé un fichier de ce type,

  - **Named** : ce sont les tubes nommés.

Structure *inode*

- Tous les types de fichiers Unix sont gérés par le SE au moyen des inodes,
- Un inode est une structure de contrôle qui contient les informations clés nécessaires au SE,
  - Plusieurs noms de fichiers peuvent être associés au même inode,
  - Un inode est associé à exactement un seul fichier,
  - Un fichier est contrôlé par exactement un seul inode.

File Mode	16-bit flag that stores access and execution permissions associated with the file	**Link Count**	Number of directory references to this inode
	12-14 File type (regular, directory, character or block special, FIFO pipe)	**Owner ID**	Individual Owner of file
	9-11 Execution flags	**Group ID**	Group owner associated with this file
	8 Owner read permission	**File size**	Number of bytes in file
	7 Owner write permission	**File Adresses**	39 bytes of adress information
	6 Owner execute permission	**Last Accessed**	Time of last file access
	5 Group read permission	**Last Modified**	Time of last file modification
	4 Group write permission	**Inode Modified**	Time of last inode modification
	3 Group execute permission		
	2 Other read permission		
	1 Other write permission		
	0 Other execute permission		

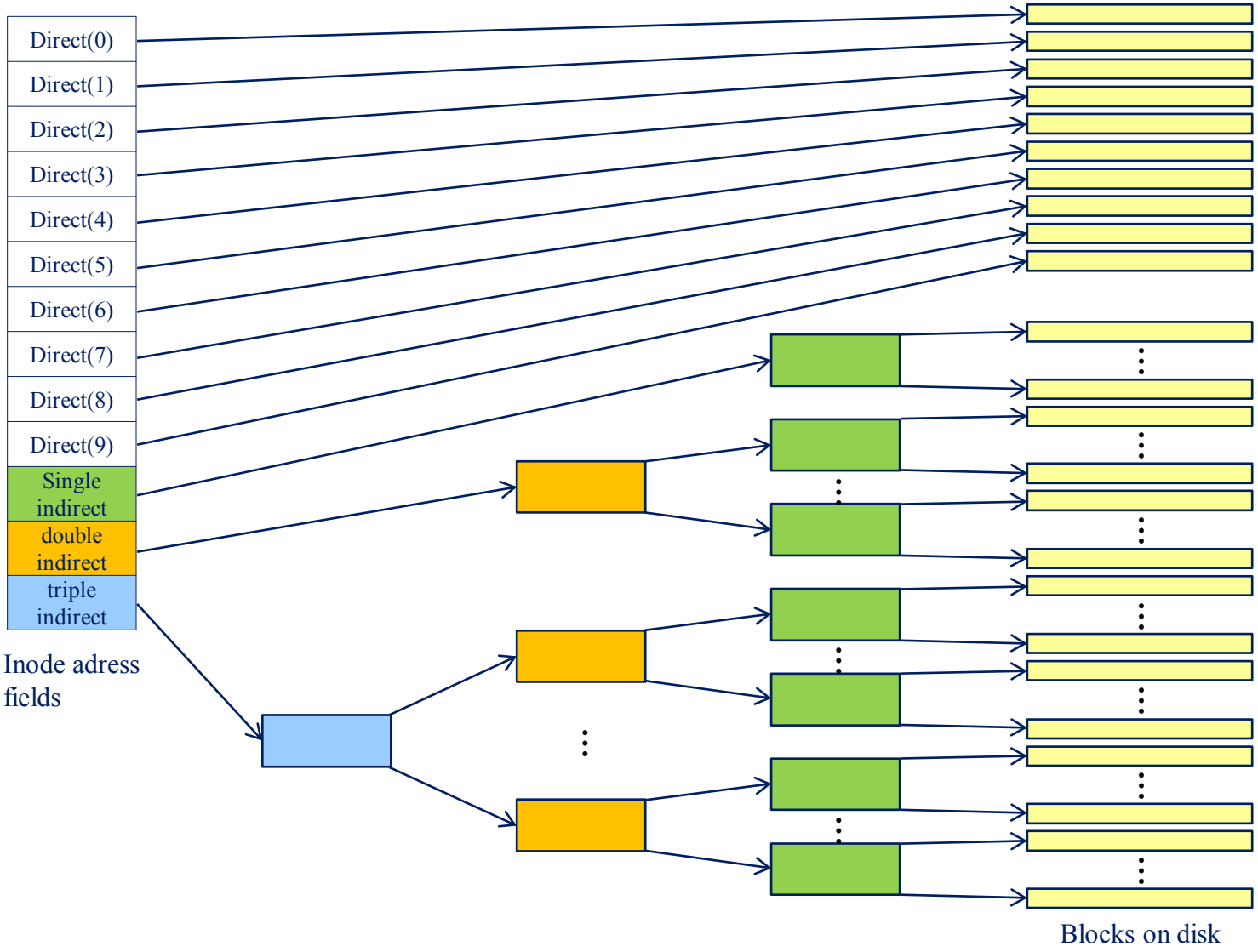
Information in a UNIX Disk-Resident Inode

Méthode d'Allocation des Fichiers

- L'allocation est accomplie sur la base de bloc,
- L'allocation est dynamique :
  - Allocation "à la demande" (pas de pré-allocation),
  - Les blocs ne sont pas nécessairement contigus,

- Une méthode indexée est utilisée pour garder trace de chaque fichier :
  - Une partie de l'index est stockée dans l'inode du fichier,
  - L'inode inclut 39 octets d'information adresses, organisés en 13 adresses (ou pointeurs) de 3 octets chacune : les 10 premières adresses pointent sur les 10 premiers blocs de données du fichier. Si le fichier est plus grand que la taille des 10 blocs, alors un ou plusieurs niveaux d'indirection sont utilisés comme suit :
    - La 11ème adresse dans l'inode pointe vers un bloc sur disque qui contient la partie suivante de l'index. Ce bloc, appelé bloc d'indirection simple, contient les pointeurs vers les blocs de données suivants du fichier,
    - Si le fichier contient plus de blocs, la 12ème adresse dans l'inode pointe vers un bloc d'indirection double. Ce bloc contient une liste d'adresses de blocs d'indirection simple additionnels. Chaque bloc d'indirection simple, à son tour, contient des pointeurs vers des blocs de données du fichier.
    - Si le fichier contient toujours plus de blocs, la 13ème adresse dans l'inode pointe sur un bloc d'indirection triple, qui est le 3ème niveau d'indexation. Ce bloc contient des pointeurs vers des blocs d'indirection double additionnels.

Adressage des Blocs



Capacité d'un Fichier Unix

- Cas du System V :
  - Taille d'un bloc = 1 Kbyte,
  - Chaque bloc peut contenir un total de 256 adresses de blocs,

Level	Number of Blocks	Number of Bytes
Direct	10	10 K
Single Indirect	256	256 K
Double Indirect	256 x 256 = 65 K	65 M
Triple Indirect	256 x 65 K = 16 M	16 G

- Avantages de cette organisation
  - L'inode est de taille relativement petite et fixe, lui permettant d'être maintenu en mémoire de longues périodes,
  - Les petits fichiers peuvent être accédés avec peu ou pas du tout d'indirection, réduisant ainsi les temps de traitement et d'accès disque,
  - La taille maximale d'un fichier est assez grande pour satisfaire pratiquement presque toutes les applications.

Le Système de Fichiers NFS

- **Fonctionnalités de base**
  - Accès transparent aux systèmes de fichiers et répertoires distants autorisés
  - Indépendance par rapport au type de réseau (fonctionnement courant sur réseau local)
  - Indépendance par rapport au protocole de transport (fonctionnement courant sur le protocole UDP)
  - Indépendance par rapport au type de machine et au système d'exploitation (MS-DOS, UNIX/SYSTEM V, UNIX/BSD, VMS, …)
  - Serveur sans état (Stateless server) moins complexe et plus robuste en cas de panne
  - Maintien (partiellement) de la sémantique UNIX sur les systèmes de fichiers
  - Pas de gestion de périphériques
  - Pas de gestion des accès concurrents (service Lock Manager)
  - Adaptation des noms de fichiers selon l'OS (longueur, types de caractères, séparateurs dans les chemins)
  - Fiabilité : liée à la simplicité de relance en cas de panne
  - Sécurité : authentification, droits d'accès aux ressources partagées, sélection de clients, différents types de montages (lecture seule, …)
  - Portabilité : NFS a été porté sur différentes plates-formes matérielles sous des OS différents, ce qui permet l'accès aux ressources d'un environnement hétérogène
  - Les différentes machines du réseau peuvent à la fois être clients et serveurs
  - Administration simplifiée : pas d'administration générale sauf dans le cas du service NIS
  - Extension de fonctionnalités possibles par ajout d'autres services RPC
  - Possibilité d'utilisation de NFS sur des stations sans disque
  - Standard de facto

Le Système de Fichiers NFS

- **Concepts de base**
  - Modèle Client-Serveur

    L'accès aux ressources distantes sous NFS est conforme au modèle Client-Serveur (Serveur : machine quelconque du réseau ayant mis à disposition pour le partage certaines de ses ressources)

    Une machine serveur ne peut pas servir d'intermédiaire (accès direct uniquement)

    Chaque machine du réseau peut être à la fois client et serveur
  - VFS (Virtual File System)

    Interface implémentée dans le noyau UNIX permettant l'accès à différents systèmes de fichiers de façon transparente
  - VNODE (Virtual Node)

    Interface implémentée dans le noyau UNIX permettant de séparer les opérations sur les systèmes de fichiers de leurs particularités d'implémentation
  - RPC (Remote Procedure Call)

    Mécanisme d'appel de procédures distantes assurant à NFS son indépendance par rapport aux protocoles et aux réseaux de transport

    Ensemble de programmes constitués eux-mêmes de procédures : une procédure est identifiée de façon unique dans un programme par un numéro, un programme est lui-même défini par un numéro ainsi qu'un numéro de version.
  - XDR (External Data Representation

    Fournit une description et une représentation communes de données échangées entre les différentes machines du réseau.

    Définit les types de données, leur taille, l'alignement entre ces types et l'ordre des octets.

    Bibliothèque dont les routines sont indépendantes du sens de transmission.
  - Serveur sans état

    Le protocole NFS ne maintient pas la trace des requêtes passées. L'état du client n'est donc pas sauvegardé pour les différentes opérations effectuées d'une transaction à l'autre

Le Protocole NFS

- Procédures RPC constituant le protocole NFS
  - Recherche d'un fichier

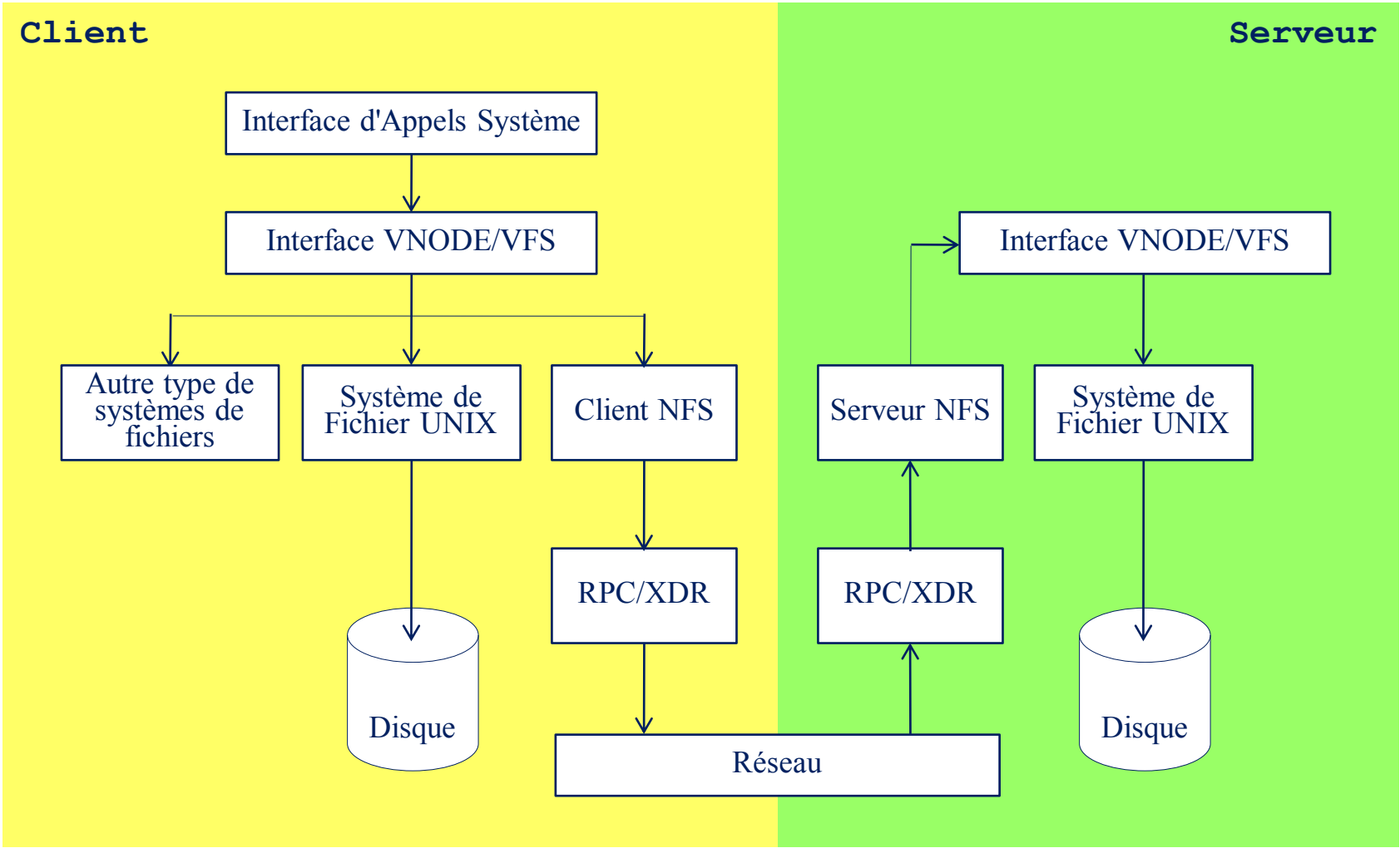
    La recherche se fait étape par étape (composant par composant. Chaque fois qu'un nom est localisé, la procédure retourne le "file handle" correspondant, ainsi de suite jusqu'au nom recherché.

    File handle : objet opaque retourné par le serveur comme élément de référence pour toutes opérations sur un fichier donné.
  - Lecture d'un ensemble d'entrées de répertoire
  - Manipulation des liens et des répertoires

    Lecture d'un lien (readlink, ls), création d'un lien (ln), suppression d'un fichier (unlink), renommage d'un fichier (mv), création et suppression d'un répertoire (mkdir, rmdir)
  - Accès aux attributs de fichiers

    Lecture des attributs (stat, fstat, access, …) positionnement des attributs (chown, chmod, …) d'un fichier.
  - Lecture et écriture de fichiers

Schéma de Communication Client-Serveur sous NFS



Bibliographie

- Operating Systems, Internals and Design Principles,
  Willam Stallings - Printice Hall

- The magic garden explained, The internals of Unix System V Release 4
  Berny Goodheart & James Cox - Prentice Hall

- Principes des Systèmes d'Exploitation,
  A. Silberschatz, P.B. Galvin, G. Gagne – Vuibert

- NFS, Système de fichiers distribués sous Unix,
  Jean-Raphaël Tong-Tong – Eyrolles