2ème Ingéniorat en Informatique

Structure de Fichiers & Structure de Données

Chapitre III
Organisations des fichiers

2023/2024

Dr. HANNOUSSE Abdelhakim

Contenu de chapitre

- Méthodes d'allocation
- Méthodes d'indexation
- Méthodes de hachage
- Choix d'une organisation

Méthodes d'allocation

Contiguë, Chainée, Chainée Indexée, Par allocation des nœuds

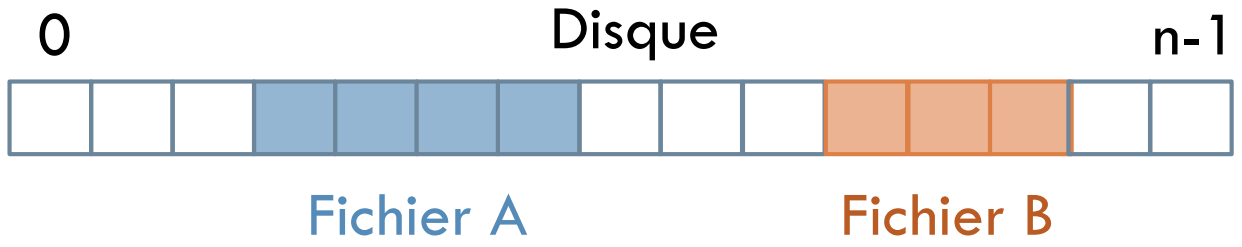
Allocation de fichiers

- L'allocation de fichiers fait référence à la manière dont l'espace de stockage sur un support de stockage (comme un disque dur) est attribué aux fichiers.
- Il existe plusieurs méthodes d'allocation de fichiers, chacune ayant ses avantages et ses inconvénients en fonction de:
  - **Besoins de stockage:** archivage de documents, stockage de fichiers multimédias, stockage de données générées par des capteurs IoT.
  - **Gestion des données:** bases de données, gestion de contenu.
  - **Performance:** traitement en temps réel, traitement de données de simulation, jeux en ligne.

Allocation contiguë (1/2)

- Chaque fichier est stocké sous forme de blocs contigus (adjacents) sur le support de stockage.
- Un fichier peut être vue comme un tableau.
- Les enregistrements de fichiers pas forcement être ordonnées.
- Si les enregistrements ne sont pas ordonnées, les ajoutes des nouveaux enregistrements ce fait à la fin du fichier → accès séquentiel.
- Si les enregistrements sont ordonnées, les ajoutes des nouveaux enregistrement nécessite un décalage → accès séquentiel ou dichotomique.

Fichier	Block 0	Nbre. Blocks
A	3	4
B	10	3



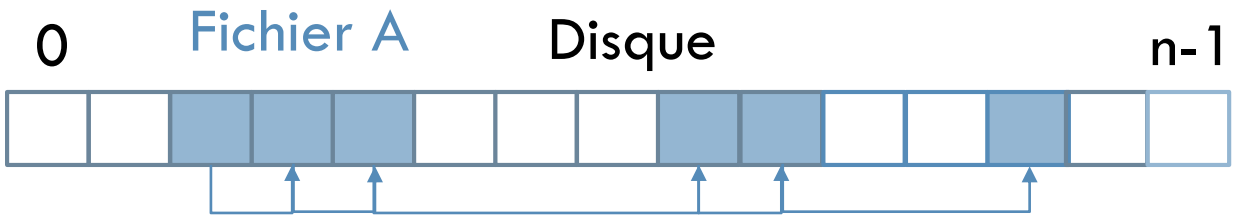
Allocation contiguë (2/2)

- **Avantages:**
  - Simple à implémenter.
  - Accès directe aux blocs en temps constant.
  - Particulièrement utile pour les petits fichiers.
  - Solution pratique pour les supports de stockage ROM comme les CDROM
    - Un fichier vidéo de 2 Go est stocké de manière contiguë sur un disque dur. Cela permet une lecture fluide de la vidéo.
- **Inconvénients:**
  - Entraîne un gaspillage d'espace disque, car il peut être difficile de trouver un emplacement continu suffisamment grand pour certains fichiers.
  - Difficile à gérer dans un environnement où les fichiers sont créés et supprimés fréquemment.
  - La fragmentation peut se produire si l'espace libre est fragmenté. Les fichiers de grande taille peuvent être difficiles à stocker de manière contiguë.

Allocation chaînée (1/2)

- Chaque fichier est associé à une liste chaînée de blocs de données.
- Chaque bloc contient une partie du fichier et un pointeur vers le bloc suivant.
- Les enregistrements de fichiers pas forcement être ordonnées.
- Si les enregistrements ne sont pas ordonnées, les ajoutes des nouveaux enregistrements ce fait à la fin du fichier.
- Si les enregistrements sont ordonnées, les ajoutes des nouveaux enregistrement causent des insertions de nouveaux blocs (allocation dynamique).
- L'accès est séquentiel dans les deux cas.

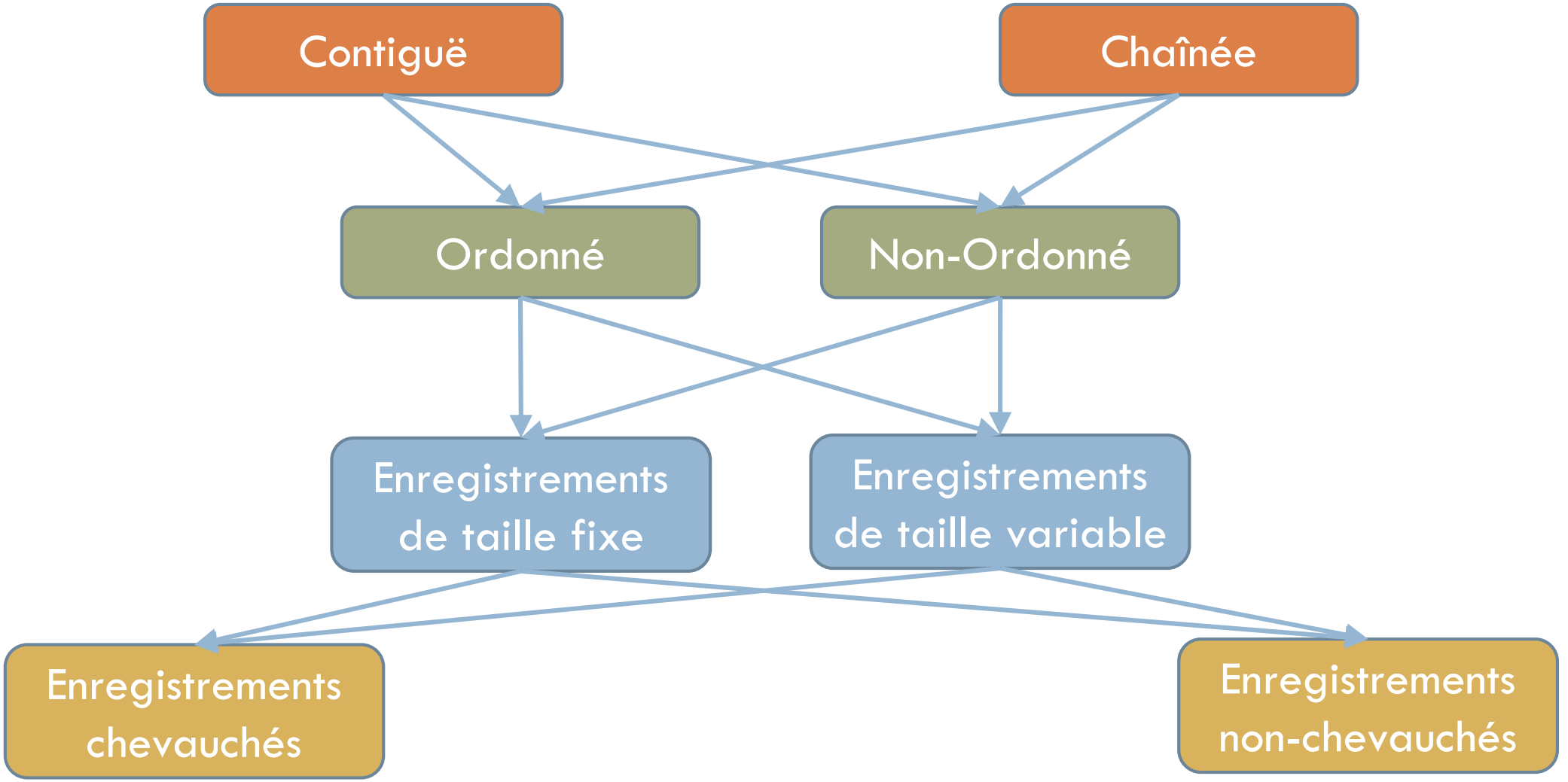
Fichier	Block 0
A	2



Allocation chaînée (2/2)

- **Avantages:**
  - Pas de problème d'extension.
  - Meilleur exploitation d'espace de stockage = Tout bloc libre peut être utilisé pour satisfaire une requête d'allocation.
  - Particulièrement utile pour les fichiers volumineux.
- **Inconvénients:**
  - L'accès aux données est plus lent car l'accès à un bloc quelconque nécessite l'accès à tous les blocs qui le précèdent.
  - Nécessite plus d'espace de stockage puisque les pointeurs doivent aussi être stocké.
  - Bien que l'allocation chaînée élimine la fragmentation externe, elle peut souffrir de fragmentation interne. Cela se produit lorsque chaque bloc de données contient un petit espace inutilisé, ce qui peut gaspiller de l'espace disque.

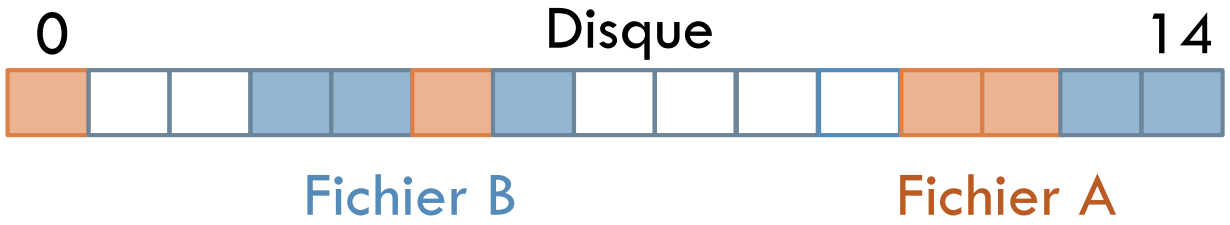
Différentes organisations internes possibles



Allocation chaînée indexée (1/2)

Fichier	Block 0
A	0
B	3

- L'idée consiste à séparer les pointeurs et les données.
- Une table d'allocation de fichier (FAT: File Allocation Table) est utilisé pour stocker les pointeurs.
- Pour protéger la FAT principale, une copie de sauvegarde de la FAT, appelée FAT d'ombre est crée.
- Chaque bloc est associée une entrée dans la FAT qui contient le numéro du bloc suivant.
- Cette méthode est utilisé par MS-DOS.



0	5
1	
2	
3	4
4	6
5	11
6	13
7	
8	
9	
10	
11	12
12	-1
13	14
14	-1

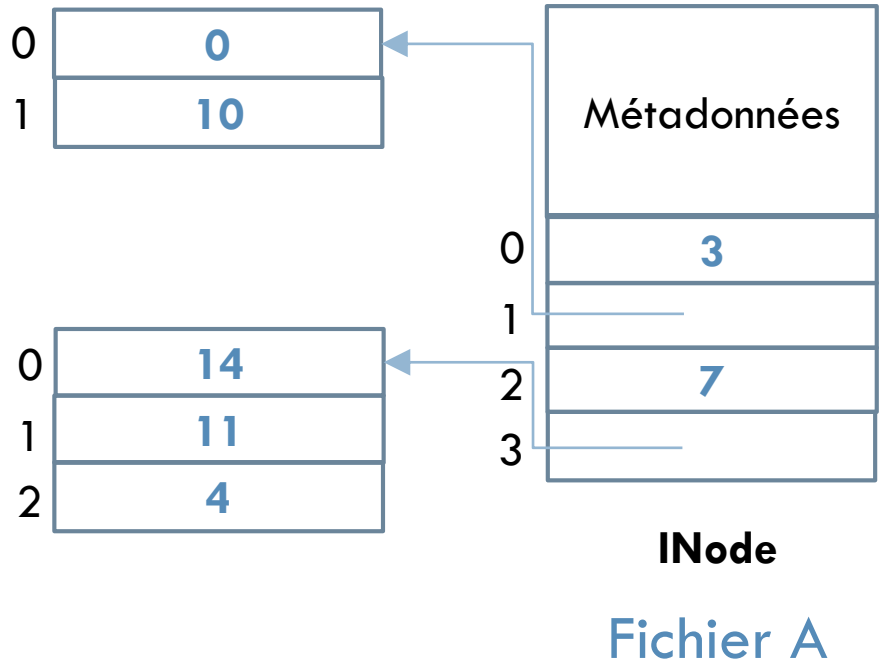
FAT

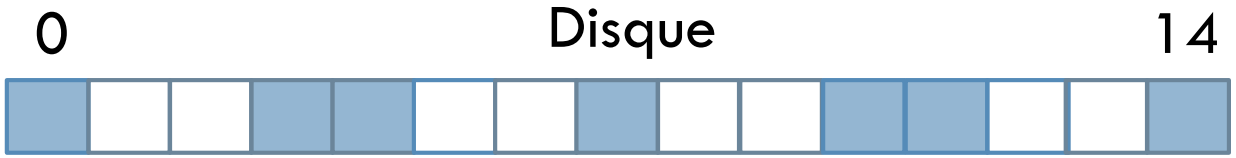
Allocation chaînée indexée (2/2)

- **Avantages:**
  - Pas de problème d'extension de fichiers.
  - Les blocs contiennent uniquement des données.
  - Accès direct, facile et rapide aux différentes blocs.
- **Inconvénients:**
  - Nécessite plus d'espace de stockage pour sauvegarder la table FAT en mémoire.
  - Problème de supports de stockage volumineux: un disque de 1Go nécessite 4Mo pour sauvegarder la table FAT32 + 4Mo pour la FAT32 ombre.

Allocation par nœud d'information (1/2)

- Pour optimiser la gestion des fichiers, la table FAT est subdivisée en de plusieurs petites tables, appelées Nœuds d'Information **INodes**.
- Chaque fichier présent est associé à un **INode** spécifique qui agit comme une carte d'identité du fichier, stockant des métadonnées et des informations cruciales pour la gestion du fichier.
- À l'intérieur de chaque **INode**, on trouve aussi les adresses des blocs sur le disque qui contiennent le contenu du fichier.
- Utilisé pour les systèmes Unix (UFS) **avec différents niveaux de hiérarchies.**





Allocation par nœud d'information (2/2)

- **Avantages:**
  - Exploitation efficaces des données: Seuls les nœuds d'information des fichiers ouverts sont chargés en mémoire centrale
  - Accès direct facile et rapide: nécessite un nombre minimal d'accès au disque.
  - Particulièrement utile pour les disques volumineux.
- **Inconvénients:**
  - La nécessité de gérer les INodes peut rendre la gestion des fichiers plus complexe.
  - L'allocation par INodes est inefficace pour les petits fichiers.
  - Les systèmes de fichiers basés sur les INodes utilise un nombre prédéfini d'INodes.

Méthodes d'index

Primaire, Secondaire

Méthodes d'Indexation

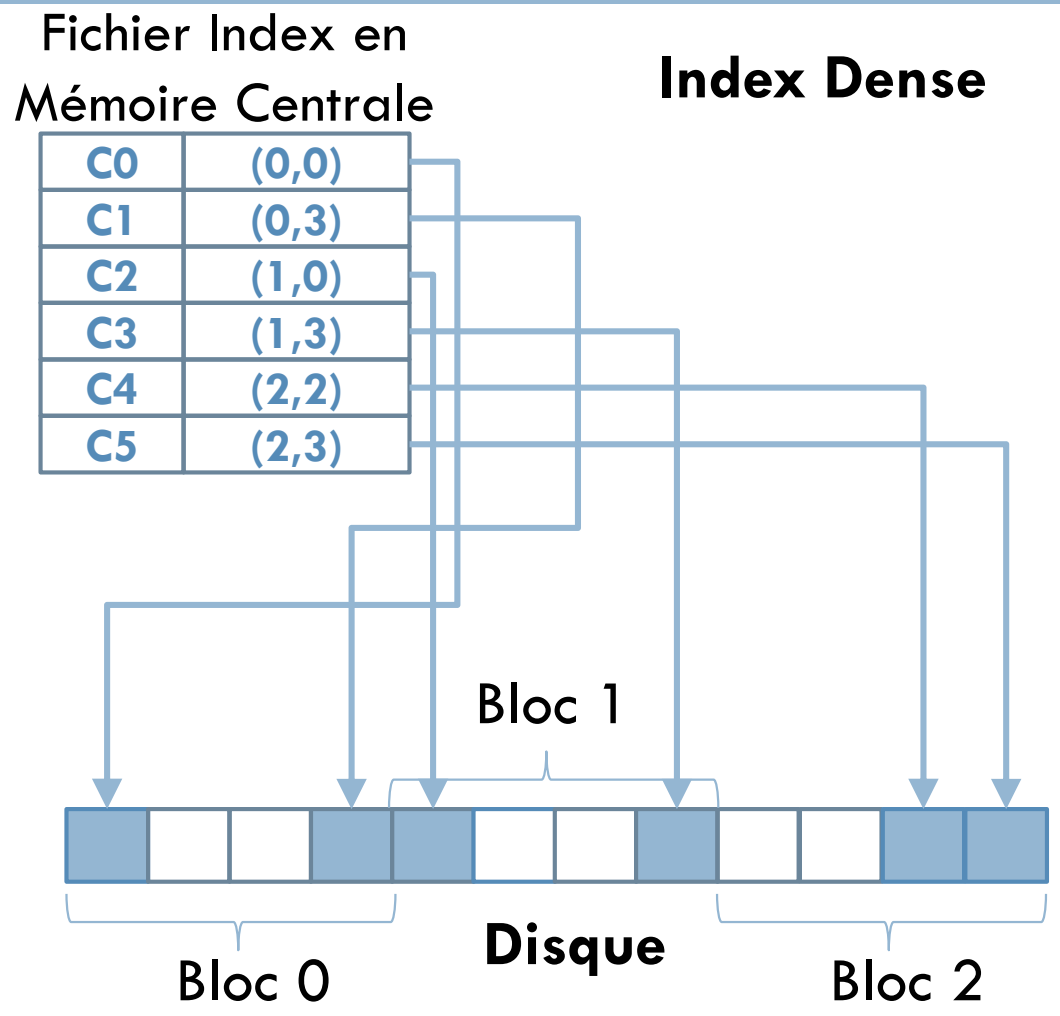
- Quand le fichier de données devient volumineux, les opérations d'accès (recherche, insertion, etc.) deviennent très inefficaces.
- Les méthodes d'index permettent d'améliorer les performances en gérant une structure auxiliaire (table d'index) pour accélérer la recherche.
- Deux fichiers sont donc utilisés : *fichier d'index* + *fichier données*.
- Le fichier de données n'est pas forcement trié. Il peut être un tableau ou une liste linéaire chaînée de blocs sur le disque.
- Le fichier d'index est supposé en mémoire et trié selon les clés des enregistrements.
- Permettent la recherche dichotomique même pour les articles de longueur variable.
- Faire la recherche dichotomique sur le fichier d'index est beaucoup plus rapide que sur le fichier de données lui-même.
- La suppression des enregistrements peut être fait par des **flag** sans accès aux disques.

Index Dense vs. Index Non-dense

- **Index Dense :** il existe une correspondance directe entre les emplacements de l'index et les données → Chaque enregistrement a une position dans le fichier index.
  - Permet un accès rapide aux données, car il n'est généralement pas nécessaire de parcourir l'ensemble de l'index pour accéder à un enregistrement spécifique.
  - Pratique pour les fichiers non-triés mais utile aussi pour les fichiers triés.
- **Index Non-dense :** les emplacements de l'index ne correspondent pas directement aux données. Il peut y avoir des références indirectes aux données.
  - L'accès aux données peut nécessiter un parcours de l'index ou une résolution d'adresses supplémentaire, ce qui peut rendre les opérations de lecture plus lentes que dans le cas d'un index dense.
  - Il est possible de construire des indexes hiérarchisés ce qui permet de gérer efficacement de gros fichiers.
  - Pratique uniquement pour les fichiers triés.

Méthode d'Index primaire (Index à 1 niveau)

- On maintient en mémoire centrale une table ordonnée, contenant toutes les clés du fichier de données (**clé, adresse**)
- A chaque clé est alors associée l'adresse de l'enregistrement dans le fichier de données. L'adresse est un couple: **(numéro de bloc, déplacement).**
- Les fichiers de données et d'index peuvent être de n'importe quelle structure (blocs contigus, blocs chaînés). De même que les enregistrements peuvent être à format fixe ou variable (avec ou sans chevauchement).

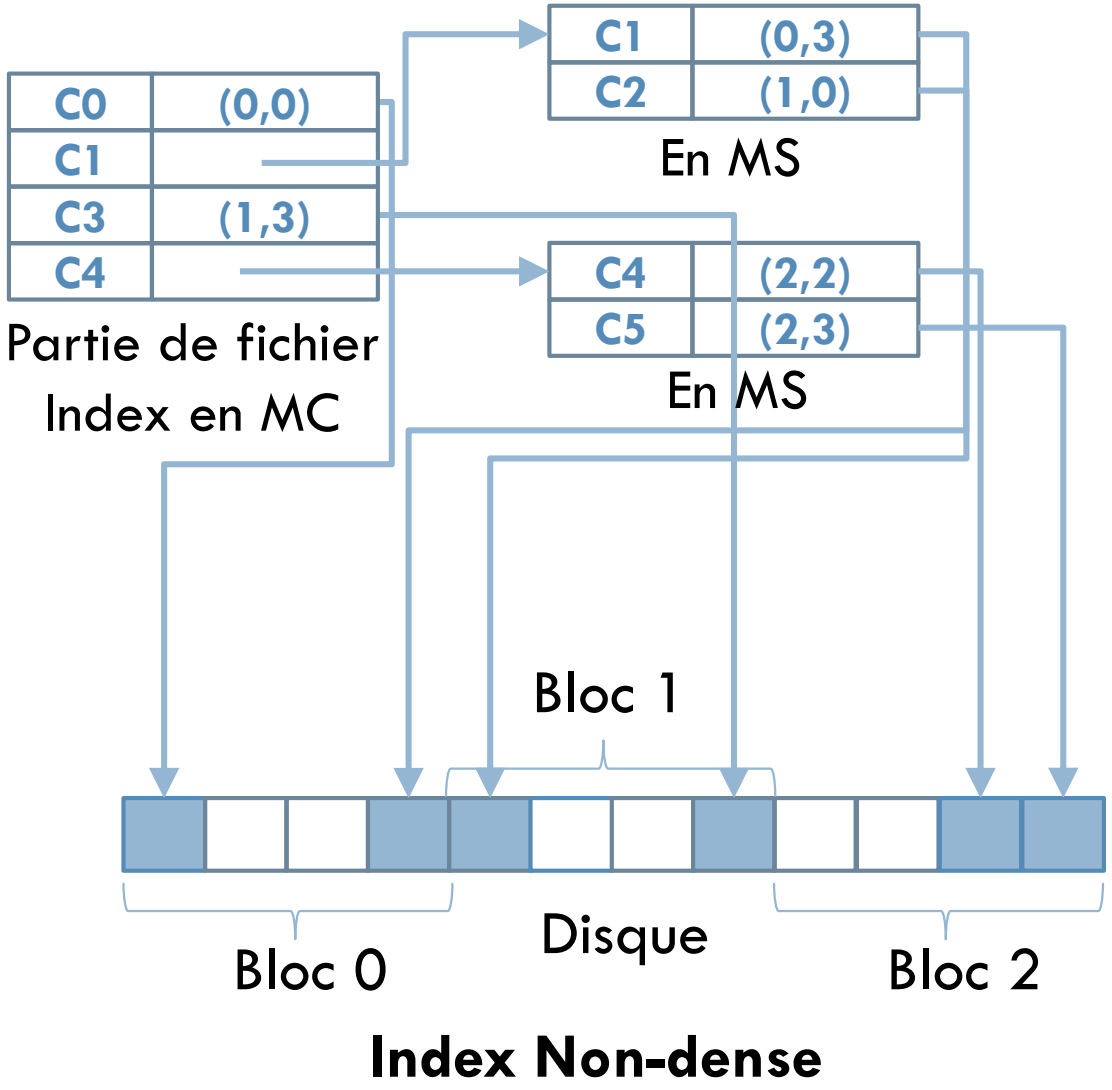


Méthode d'Index primaire (Index à 1 niveau)

- **Recherche:** On utilise la recherche dichotomique de la clé de l'enregistrement dans la table d'index:
  - Si la clé est trouvée, le bloc contenant l'article est ramené en mémoire centrale afin de récupérer l'article.
  - Si ce dernier est à cheval sur deux bloc, le deuxième bloc est ramené pour récupérer le reste de l'article.
- **Insertion:** L'insertion d'un nouvel enregistrement se fait en fin de fichier de données. Sa clé est insérée dans la table d'index avec décalages pour garder l'ordre des clés → coût = 2 accès disques.
- **Suppression:** La suppression nécessite d'éliminer physiquement les enregistrements supprimés, un nouveau fichier peut être créer pour accélérer l'opération.

Méthode d'Index primaire (Index à 2 niveaux)

- Lorsque la taille du fichier index dépasse la capacité de la mémoire centrale, une approche consiste à créer un deuxième niveaux d'index.
- Une seule clé est sélectionnée pour chaque bloc du fichier index (ce que l'on appelle un index non dense) pour élaborer le deuxième index.

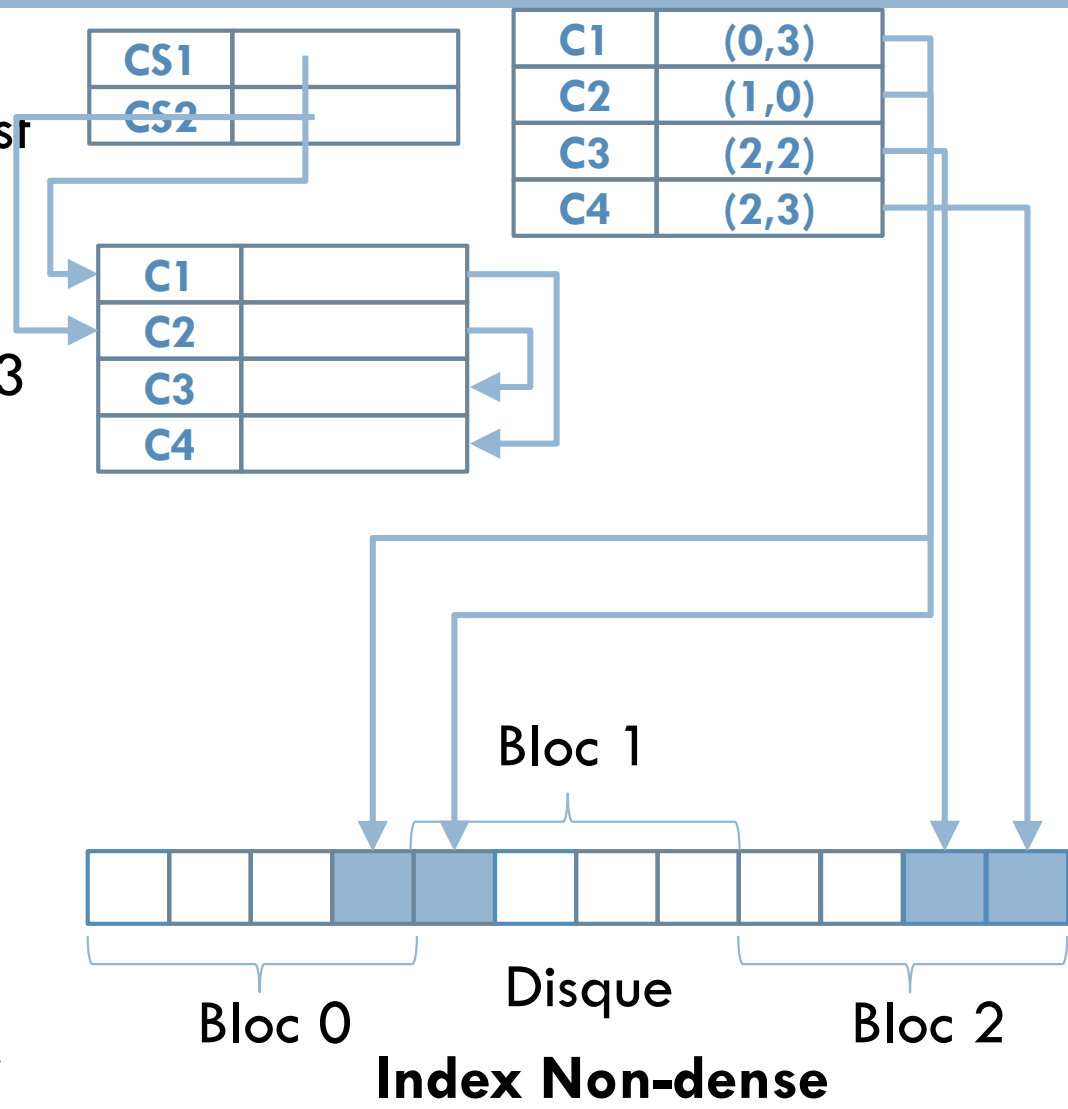


Méthode d'Index primaire (Index à 2 niveaux)

- **Recherche:** On utilise la recherche dichotomique de la clé de l'enregistrement dans la table d'index.
  - Rechercher sa clé dans l'index du niveau 1, un intervalle est alors sélectionné.
  - La recherche continue dans l'index de niveau 2 uniquement dans cet intervalle.
  - Si la clé est trouvée, on procède de la même façon que dans le cas avec un seul index.
- Les recherches dichotomiques sur les deux petits vecteurs (index de niveau 1 et une partie de l'index de niveau 2) sont beaucoup plus rapides qu'une seule recherche dichotomique sur un seul grand vecteur (index de niveau 2).

Méthode d'Index secondaire

- Pour rechercher des enregistrements en fonction d'autres zones que la clé de l'enregistrement, il est utile d'utiliser une méthode d'index secondaire.
  - **Exemples:** retrouver tous les livres parus en 1973 ou retrouver tous les livres parus en 1973 de l'auteur X donné.
- La méthode d'index secondaire consiste à créer des indexes pour chaque attribut.
- Une entrée d'un index secondaire est le couple (clé-secondaire, clé-primaire)
- Quand il peut exister plusieurs enregistrements pour une clé secondaire. Des listes des clés primaires sont associées à chaque clé secondaire.



Méthode d'Index secondaire

- **Recherche:** recherche dichotomique de la clé secondaire, puis la clé primaire de l'enregistrement avant d'accéder à l'enregistrement lui-même.

Indexation

Méthode d'arbre B

Méthode d'arbre B

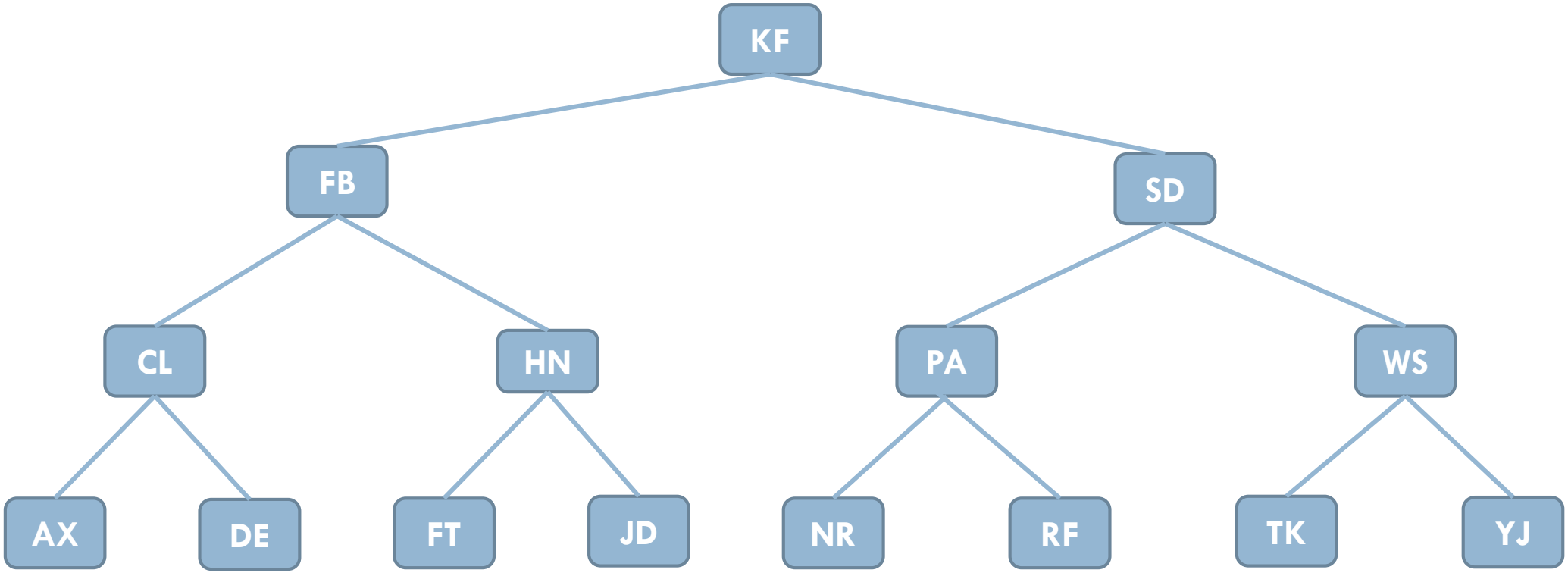
- Si le fichier est très volumineux, on ne peut garder la totalité de fichier index en MC.
- De plus, garder l'index en mémoire secondaire coûte cher :
  1. la recherche dichotomique exige beaucoup d'accès disque.
  2. il faut maintenir l'index ordonné (pour pouvoir faire la recherche dichotomique).
- **Solution =** Méthode d'arbre binaire de recherche.
- Avec un arbre B, un fichier est organisé sous forme hiérarchique:
  - **Racine :** enregistrement principal qui représente le point de départ de l'arbre.
  - **Sous arbres gauche et droit :** chaque nœud peut avoir deux sous-arbres qui contiennent des enregistrements liés à l'enregistrement parent. Le sous-arbre gauche contient généralement des enregistrements avec des valeurs de clés inférieures à celles de l'enregistrement parent, tandis que le sous-arbre droit contient des enregistrements avec des valeurs des clés supérieures.
  - **Feuilles:** les nœuds de l'arbre qui n'ont pas des sous-arbres gauche ni droit.

Méthode d'arbre B: Exemple

- Soit une liste de données triée:

  AX, CL, DE, FB, FT, HN, JD, KF, NR, PA, RF, SD, TK, WS, YJ

  - Voici sa représentation avec un arbre B:



Méthode d'arbre B – Représentation par tableau

- **Exemple:** Soit une liste de données triée:

  AX, CL, DE, FB, FT, HN, JD, KF, NR, PA, RF, SD, TK, WS, YJ

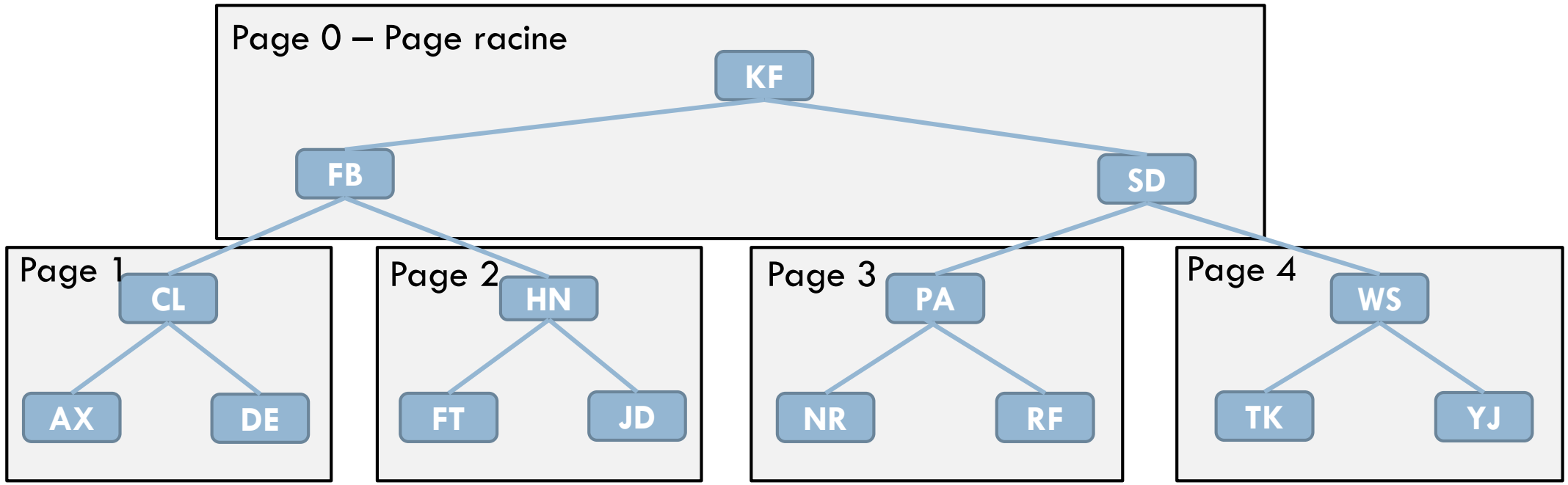
  - Voici la représentation arbre B sous forme de tableau:

Droit	8			13			2		1	3	12			5	
Index	FB	JD	RF	SD	AX	YJ	PA	FT	HN	**KF**	CL	NR	DE	WS	TK
Gauche	10			6			11		7	0	4			14	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

  - Pas de besoins de trier l'index pour pratiquer la recherche dichotomique.
  - Perte d'espace puisque la moitié des liens ne sont pas utiles (cas des nœuds feuilles).

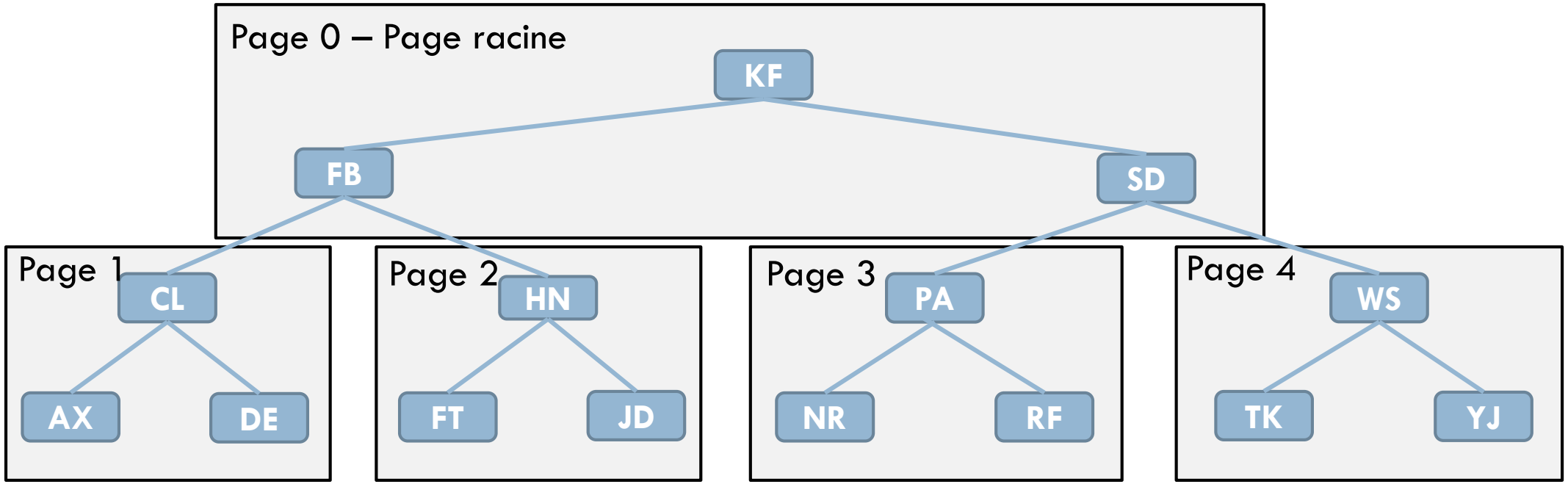
Méthode d'arbre B: Pagination de l'arbre (1/2)

- La pagination de l'arbre offre une solution au **problème de chargement en mémoire.**
- Organiser l'arbre en pages où chaque page contient une portion de l'arbre B.
- Les pages sont rangées dans des blocs sur le disque et chargé à la demande.
- L'ajout d'un enregistrement nécessite un décalage dans une page en mémoire.



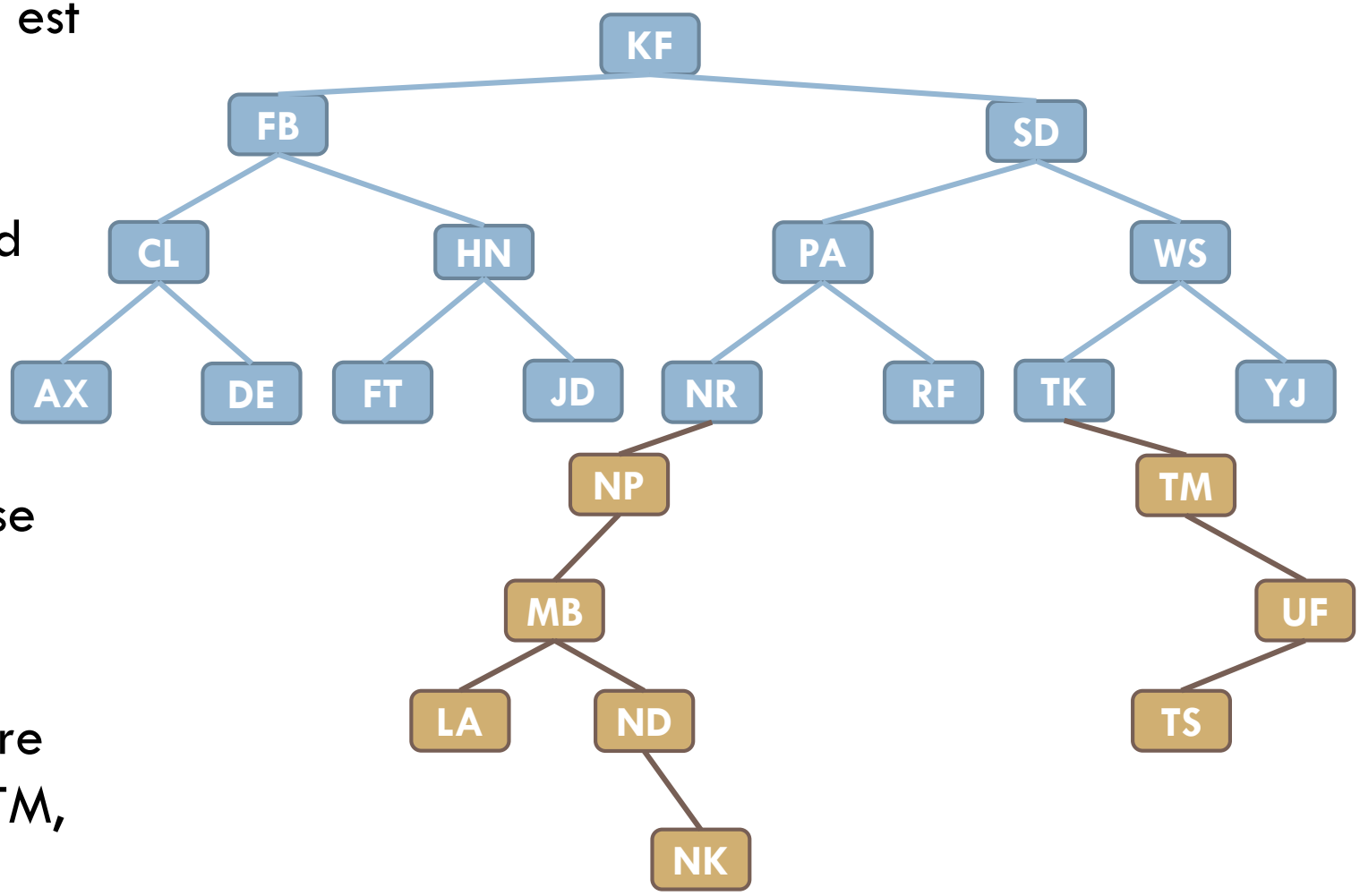
Méthode d'arbre B: Pagination de l'arbre (2/2)

- Complexité de la recherche d'un enregistrement en mémoire:
  - **Cas d'un arbre B non paginé:** $O(\log_2(N+1))$ où N est le nombre total d'enregistrement.
  - **Cas d'un arbre B paginé:** $O(\log_{k+1}(N+1))$ où k est le nombre d'enregistrements par page.



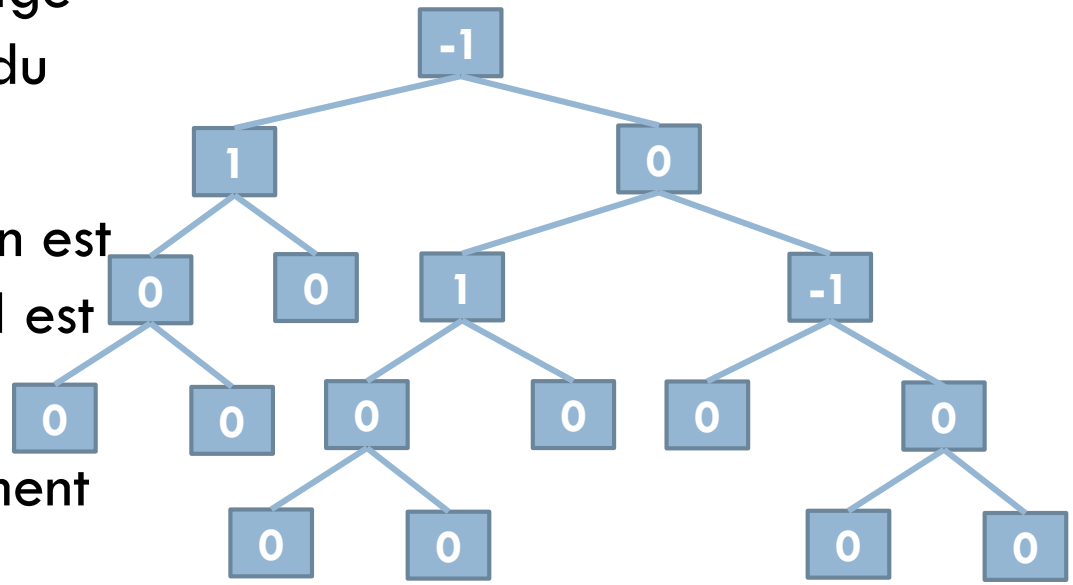
Méthode d'arbre B: Problème de déséquilibrage

- La performance de recherche est très mauvaise si l'arbre est déséquilibré.
- La structure de l'arbre dépend de l'ordre d'ajout des enregistrements. Quand les enregistrements arrivent de façon aléatoire, l'arbre peut se déséquilibrer et donc les performances se détériorent.
- **Exemple:** Si on ajoute à l'arbre précédent les clés : NP , MB, TM, LA, UF, ND, TS, NK.



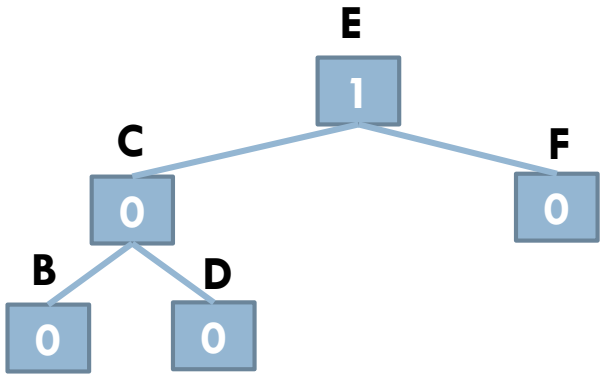
Méthode d’arbre B: Arbre AVL (1/3)

- Un arbre AVL est un arbre équilibré = les profondeurs des deux sous arbres de chaque nœud ne diffèrent pas plus d'un.
- A chaque nœud est associé un facteur d'équilibrage qui est égal à la différence entre la profondeur du sous arbre gauche et celle du sous arbre droit.
- Lorsqu'une opération d'insertion ou de suppression est effectuée, le facteur d'équilibre de chaque nœud est mis à jour, et l'arbre est rééquilibré si nécessaire.
- Un arbre AVL garantit que l'arbre reste relativement équilibré, ce qui, à son tour, garantit des performances de recherche et d'insertion efficaces → solution au **problème de déséquilibrage**.

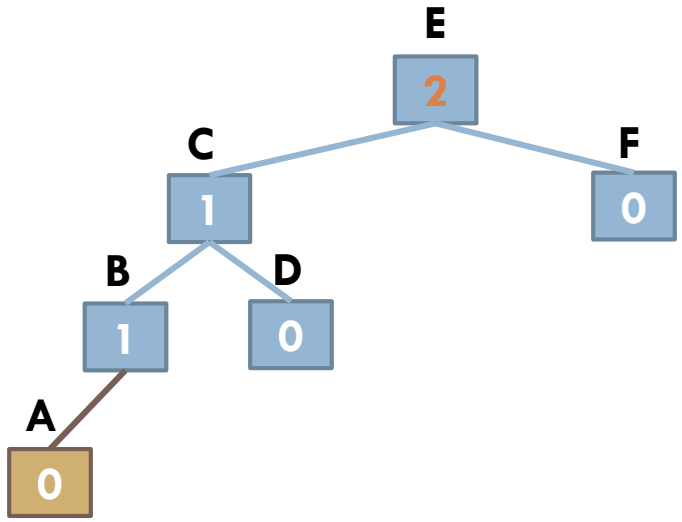


Méthode d'arbre B: Arbre AVL (2/3)

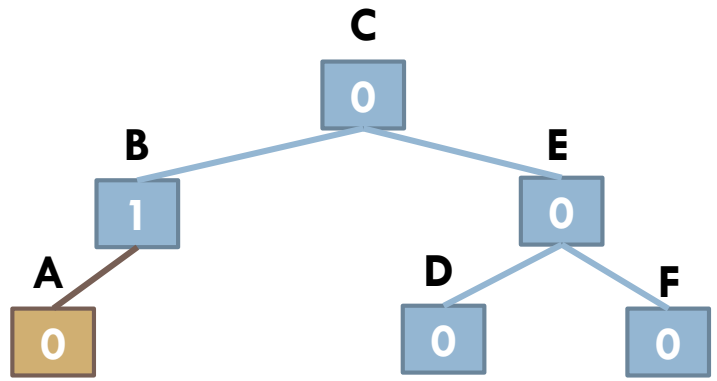
- Les rotations d'arbre sont l'une des principales techniques utilisées pour rééquilibrer l'arbre AVL tout en maintenant son ordre de recherche.
- La gestion de l'équilibrage de l'arbre peut rendre les opérations d'insertion et de suppression légèrement plus complexes que dans un ABR standard. Cependant, cette complexité est compensée par les performances globales améliorées de l'arbre AVL.



Cas initial



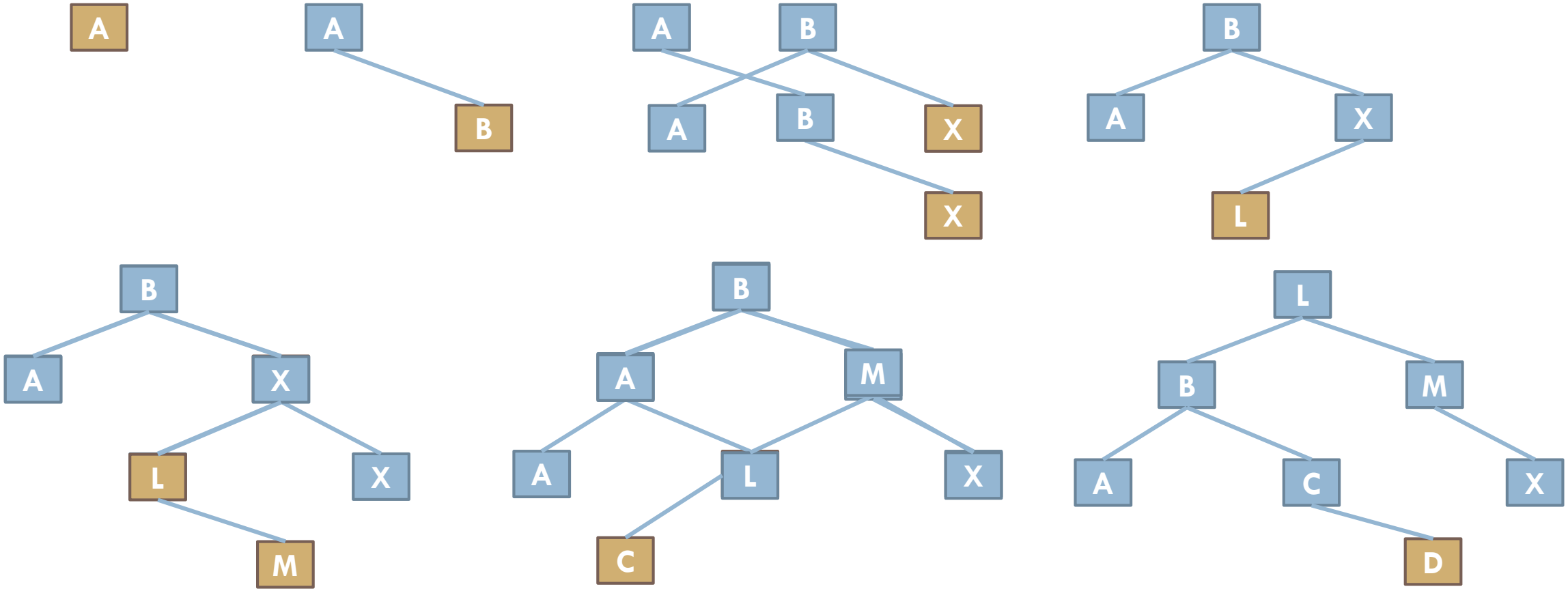
Insertion d'un nouveau enregistrement



Arbre rééquilibré

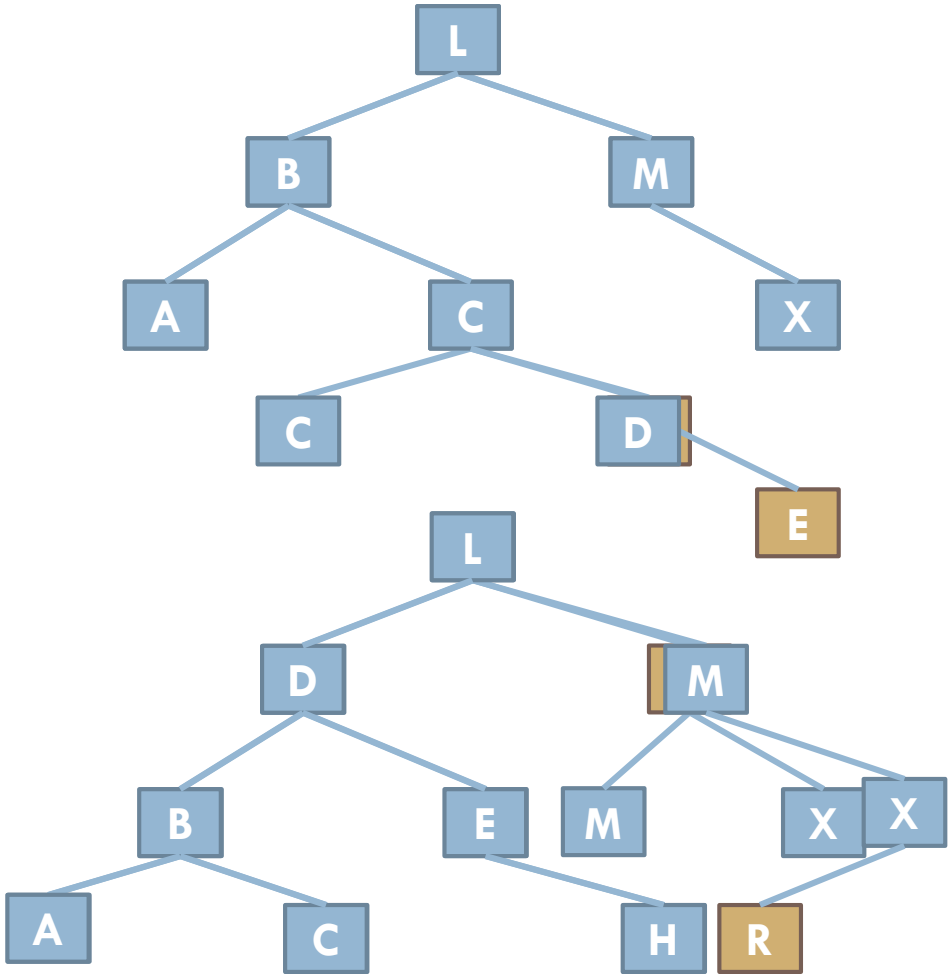
Méthode d'arbre B: Arbre AVL (3/3) - Exemple

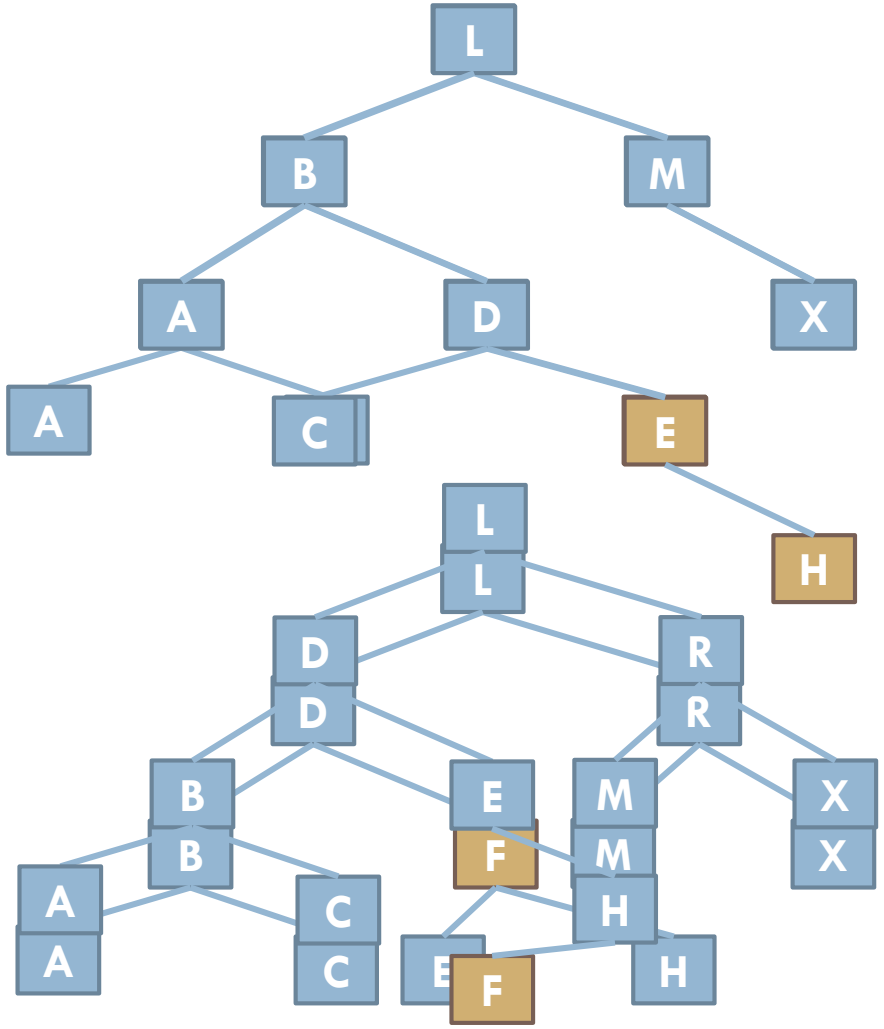
- Insertion de la séquence : A, B, X, L, M, C, D, E, H, R, S, F dans un arbre AVL:



Méthode d'arbre B: Arbre AVL (3/3) - Exemple

- Insertion de la séquence : A, B, X, L, M, C, D, E, H, R, F dans un arbre AVL:





Les méthodes de hachage

Méthodes de Hachage

- Le hachage est une technique utilisée pour indexer et accéder efficacement aux fichiers et ses enregistrements stockés dans un disque.
- Le principe de base du hachage repose sur la transformation des noms des fichiers et des clés d'enregistrements en des valeurs numériques (hachage) à l'aide d'une fonction de hachage. Ces valeurs sont utilisées pour accéder rapidement aux emplacements physiques des fichiers ou des enregistrements.
- Une table de hachage est une structure de données qui associe les valeurs de hachage aux emplacements réels des données (fichiers, enregistrements) dans le système de fichiers. Chaque emplacement dans la table de hachage est appelé un **seau** (ou **slot**).

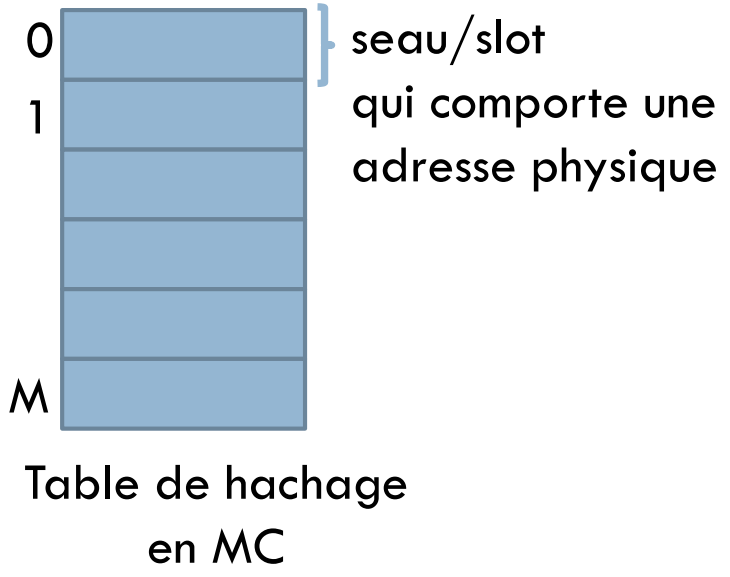


Table de hachage
en MC

Méthodes de Hachage

- Lorsqu'un fichier est créé, ou un enregistrement est ajouté a un fichier, son emplacement est calculé à l'aide de la **fonction de hachage**. La valeur de hachage du nom du fichier ou de l'enregistrement ajouté est utilisée pour déterminer dans quel seau de la table de hachage son adresse physique doit être stocké.
- Lorsqu'un utilisateur souhaite accéder à un fichier/enregistrement, le système de fichiers utilise la même fonction de hachage pour calculer la valeur de hachage du nom du fichier/clé de l'enregistrement. Ensuite, il accède a cet seau dans la table de hachage pour obtenir l'emplacement du fichier.
- Une collision apparaît quand plus de **fb** (facteur de blocage) enregistrements ont la même adresse de bloc.
- Il existe plusieurs méthodes de résolution de collisions:
  - Chaînage avec des listes séparées.
  - Chaînage ouvert.

Hachage statique

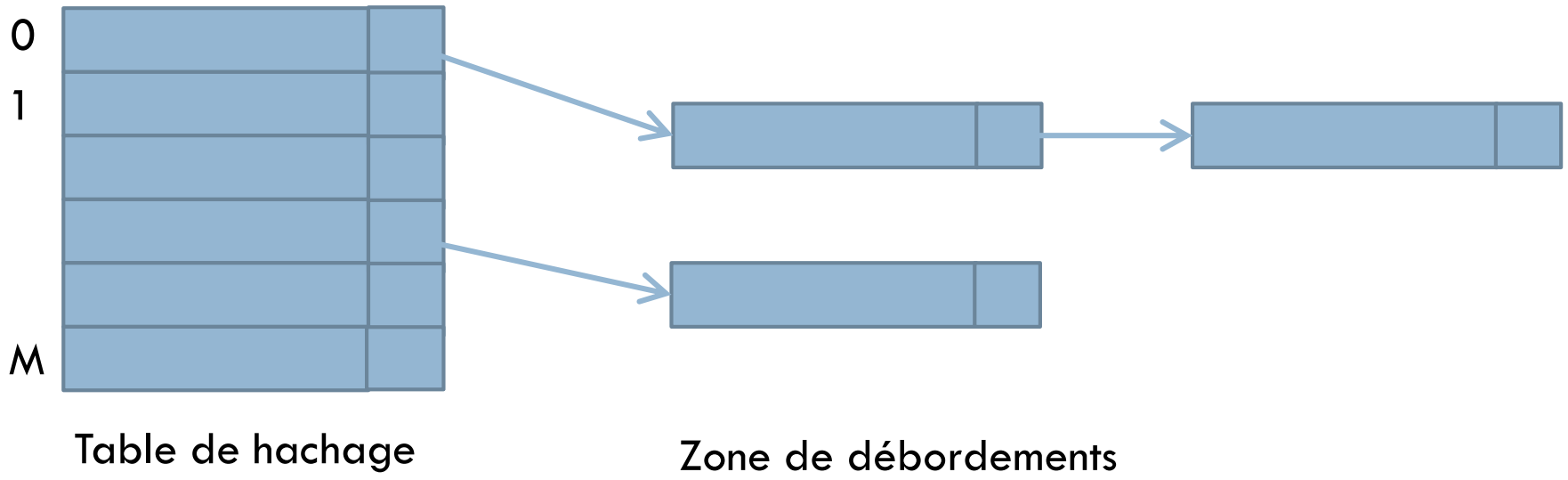
- **Hachage Statique :**
  - La taille de la table de hachage est fixe et déterminée à l'avance.
  - Lorsqu'une collision se produit, on utilise une technique de résolution de collision pour gérer les éléments qui se retrouvent à la même position.
- **Avantages :**
  - La structure de la table reste constante.
  - Plus simple à mettre en œuvre.
- **Inconvénients :**
  - Un dimensionnement initial incorrect peut entraîner un grand nombre de collisions ou un gaspillage d'espace.

Hachage dynamique

- **Hachage Dynamique :**
  - La taille de la table de hachage peut être ajustée dynamiquement en fonction du nombre d'éléments présents dans la table.
  - L'idée est d'augmenter la taille de la table lorsque celle-ci devient trop pleine (trop de collisions) et de la réduire si elle est trop peu remplie.
- **Avantages :**
  - Évite le gaspillage d'espace → la table est ajusté en fonction du nombre d'éléments.
- **Inconvénients :**
  - Plus complexe à mettre en œuvre.
  - Les opérations de redimensionnement peuvent être coûteuses en termes de performances.

Méthodes de résolutions de collisions: Chaînage avec des listes séparées

- Cette technique permet de gérer les éléments en collision en les stockant dans une structure de données distincte, généralement une liste, à l'intérieur de chaque seau (ou emplacement de la table de hachage).
- Si plus de **b** articles arrivent sur le même bloc, un lien à un article de débordement est inséré à la fin du premier bloc.



Méthodes de résolutions de collisions: Adressage ouvert

- Le chaînage ouvert consiste à stocker les éléments directement dans la table de hachage elle-même. Lorsqu'une collision se produit, le chaînage ouvert détermine un nouvel emplacement au sein de la table pour stocker le nouvel élément.
- Plusieurs techniques de peuvent être utilisées pour calculer le prochain emplacement disponible.
  - Hachage linéaire
  - Double hachage

Méthodes de résolutions de collisions: Adressage ouvert - Hachage linéaire

- Lorsqu'une clé est insérée dans la table de hachage, la fonction de hachage est utilisée pour calculer un emplacement initial pour cette clé.
- En cas de collision, le hachage linéaire explorer les emplacements suivants par un décalage fixe, généralement 1.
- Lorsqu'un emplacement vide est trouvé, l'élément en collision est stocké à cet emplacement.
- **Exemple:** Considérer la fonction de hachage h(X) = X **mod** 7. Nous voulons ajouter:

  50, 700, 76, 85, 92, 73, 101.

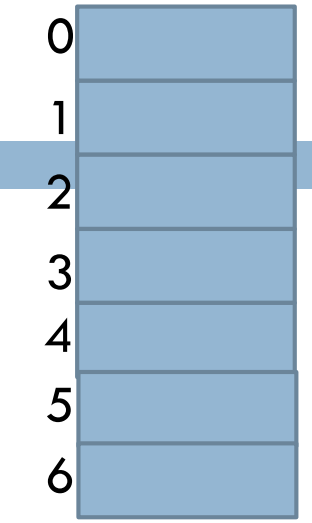
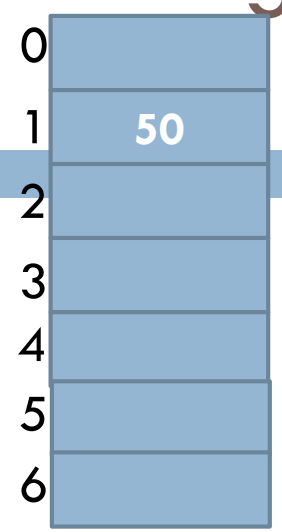
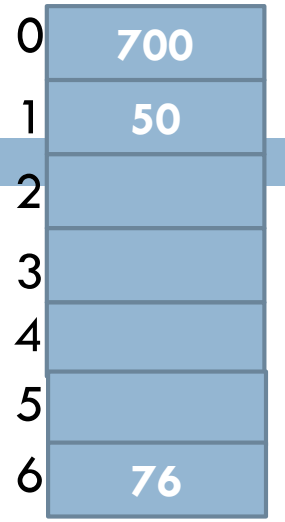


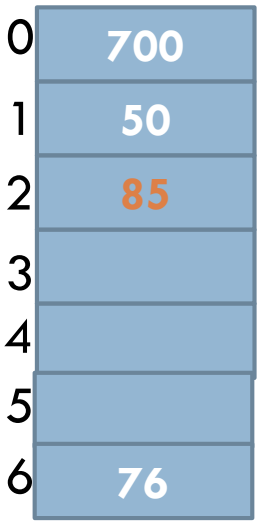
Table de hachage initial



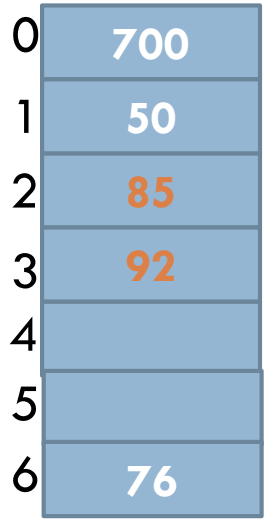
h(50) = 1



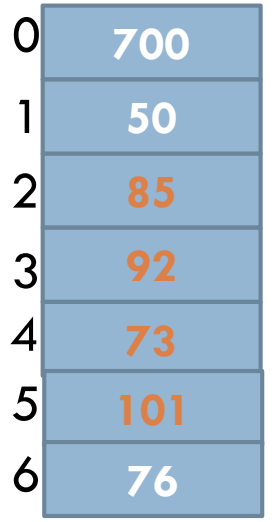
h(700) = 0,
h(76)= 6



h(85) = 1
Collision



h(92) = 1
Collision



h(73) = 3,
h(101) = 3

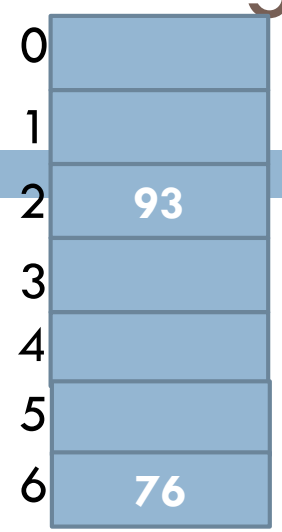
Méthodes de résolutions de collisions: Adressage ouvert - Double hachage

- Deux fonctions de hachage sont utilisées: **h1, h2**
- Lorsqu'une clé est insérée dans la table de hachage, la fonction **h1** est utilisée pour calculer un emplacement initial pour cette clé.
- En cas de collision, la fonction **h2** est utilisée pour calculer le pas en fonction de la clé.
- Lorsqu'un emplacement vide est trouvé, l'élément en collision est stocké à cet emplacement.
- **Exemple:** Considérer la fonction de hachage h1(X) = X **mod** 7, h2(X) = 5 – (X **mod** 5)

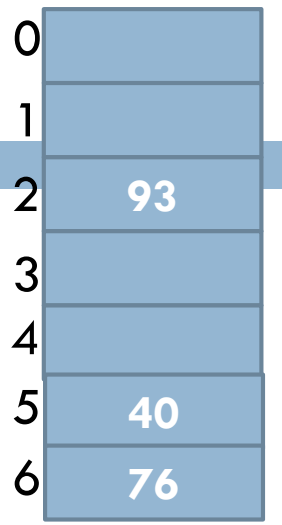
  Nous voulons ajouter:

  76, 93, 40, 47, 10, 55.

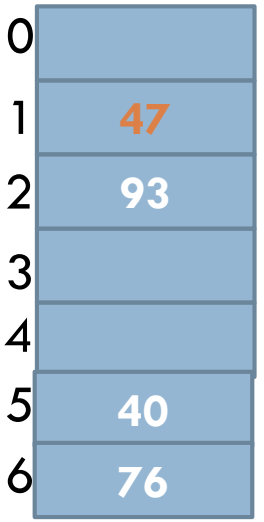
h1(76) = 6



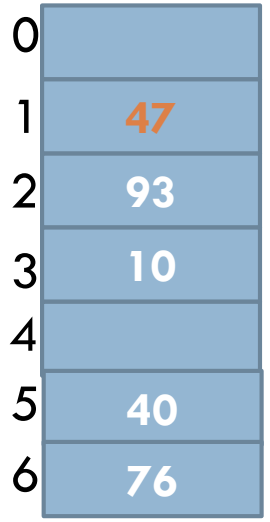
h1(93) = 2



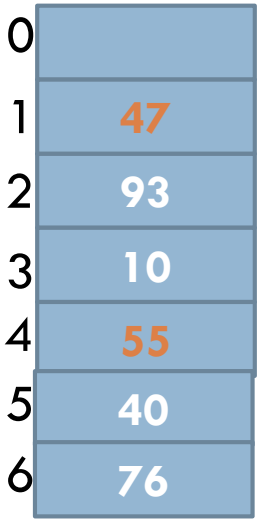
h1(40) = 5



h1(47) = 5
h2(47) = 3



h1(10) = 3



h1(55) = 6,
h2(55) = 5

Choix d'organisation

Choix d’organisation (1/3)

- Le choix de l'organisation des fichiers est une décision critique lors de la conception d'un système de gestion de fichiers. Il détermine la manière dont les fichiers seront stockées, accédés et gérées.
- Le choix est étroitement lié aux méthodes d'allocation, d'indexation et de hachage.
  - **Méthode d'Allocation :** consiste à décider comment les enregistrements des fichiers seront physiquement stockées sur le support de stockage.
  - **Méthode d'Indexation :** consiste à créer des structures d'index pour accélérer la recherche et l'accès aux fichiers et ses différents enregistrements.
  - **Méthode de Hachage :** utilisée pour accélérer la recherche de fichiers et d’enregistrements en associant une clé (comme un nom de fichier ou identifiant d’un enregistrement) à une adresse physique sur le disque.

Choix d'organisation (2/3)

- Le choix de la méthode d'allocation dépend des besoins de l'application, de la taille des fichiers et de la performance souhaitée.
- Les systèmes de fichiers modernes utilisent souvent une combinaison de ces méthodes pour optimiser l'utilisation de l'espace de stockage et assurer des performances efficaces lors de la lecture et de l'écriture de fichiers.
  - L'**allocation contiguë** est utile pour gérer les petits fichiers (fichiers de config.).
  - L'**allocation contiguë** se révèle particulièrement idéale pour le stockage de fichiers multimédias, surtout lorsqu'il s'agit de disques optiques.
  - L'**allocation chaînée** est utilisée pour des fichiers dont la taille peut évoluer au fil du temps ou comporte des enregistrements de tailles variables. Un système de messagerie utilise une allocation chaînée pour les e-mails.
  - L'**allocation indexée** se révèle efficace pour gérer un grand nombre de fichiers, en particulier lorsque leurs tailles sont variables.

Choix d’organisation (3/3)

- Utilisez un index primaire basé sur la clé d'indexation la plus pertinente.
  - Le système d’exploitation utilise le nom du fichier comme l'index primaire pour permettre une recherche rapide par nom.
- Si des recherches fréquentes sont effectuées en fonction d'autres attributs que la clé principale, des index secondaires deviennent plus utiles.
  - Pour les fichiers multimédias il est utile d’utiliser un index secondaire basé sur le type de média (photos, vidéos, documents) peut permettre une recherche plus ciblée.
- Un arbre B peut être utilisé pour gérer un grand nombre de fichiers dans un système de fichiers hiérarchique.
- Il faut choisir la méthode d'indexation en fonction de la nature des données stockées.
  - Pour des recherches a base des données diverses, un B-Tree peut être optimal. Si la recherche est principalement basée sur une clé unique, une table de hachage pourrait être plus efficace.