Université des Sciences et de la Technologie
Houari Boumediene
USTHB

Système de gestion de fichiers

Dr BOUYAKOUB F. M
bouyakoub.f.m@gmail.com

Décembre 2012

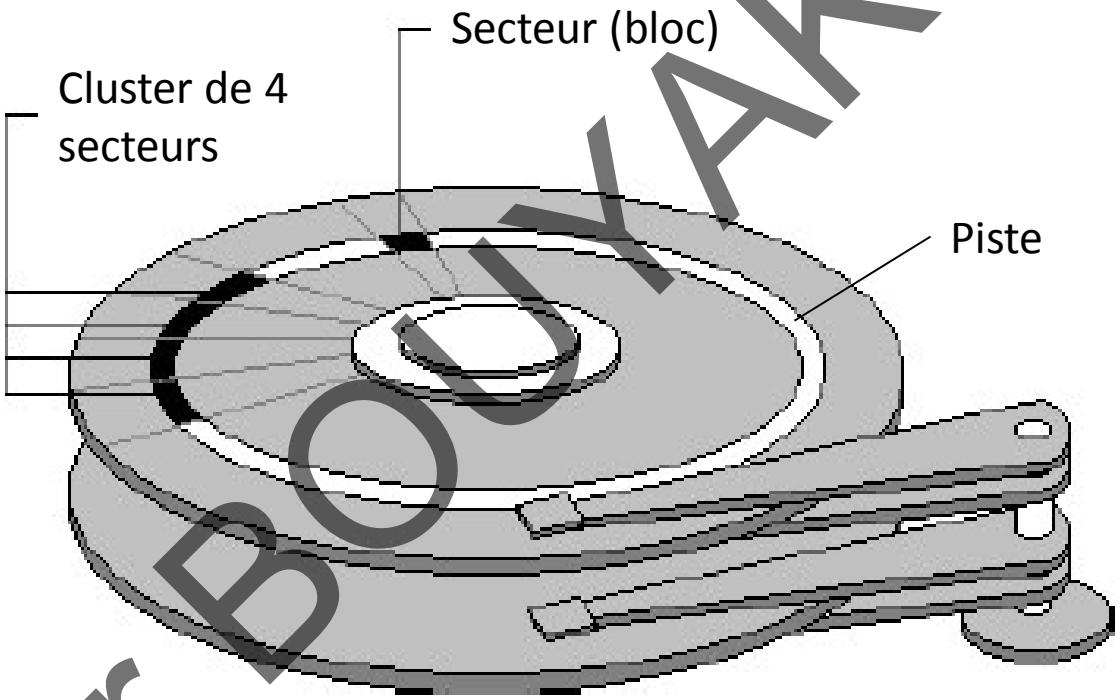
Plan du cours

- Notion de système de gestion de fichier
- Politiques d'allocations de blocs
- Gestion de l'espace libre
- Fichier et processus

Notion de système gestion de fichiers

bouyakoub.f.m@gmail.com

Structure d'un disque



Un **disque** est un support de stockage de données non volatile

Composants du disque

- L'unité d'allocation d'un disque s'appelle un **secteur**.
- Un disque est organisé en **secteurs**, **cylindres** et **plateaux**.
- Le cylindre est constitué par toutes les pistes superposées verticalement qui se présentent simultanément sous les têtes de lecture/écriture.
- Chaque piste est découpée en secteurs dont la taille est généralement de 512 octets.
- On ne manipule (lectures / écritures) pas les octets individuellement mais plutôt des **blocs physiques (clusters)** composé de plusieurs octets (typiquement 512 ou 1024).
- Une **partition** est une portion du disque.
- Une partition comporte un **système de gestion de fichiers**.

Notion de cluster

- C'est la plus petite entité, en terme d'unité de stockage, que peut gérer le SE.
- Un cluster est composé d'un ou plusieurs secteurs (1 à 64) de 512 octets ➔ cluster peut avoir une taille comprise entre 512 Ø et 32 K Ø
- Un fichier est stocké sur un nombre entier de clusters ➔ aussi petit soit-il, le fichier occupe un cluster
- Plus les clusters sont grands, plus il y a perte de place sur disque

Représentation de données et notion de fichier

- Le SE doit pouvoir stocker à long terme de grandes quantités d'informations:
  - ses propres modules,
  - les applications,
  - les bibliothèques de fonctions,
  - Les données des utilisateurs.

- Toute information stockée sur une unité de stockage est rangée sous forme de fichier

  ➔ la gestion de fichier est donc à la base de toute organisation de données

Contraintes de gestion des données

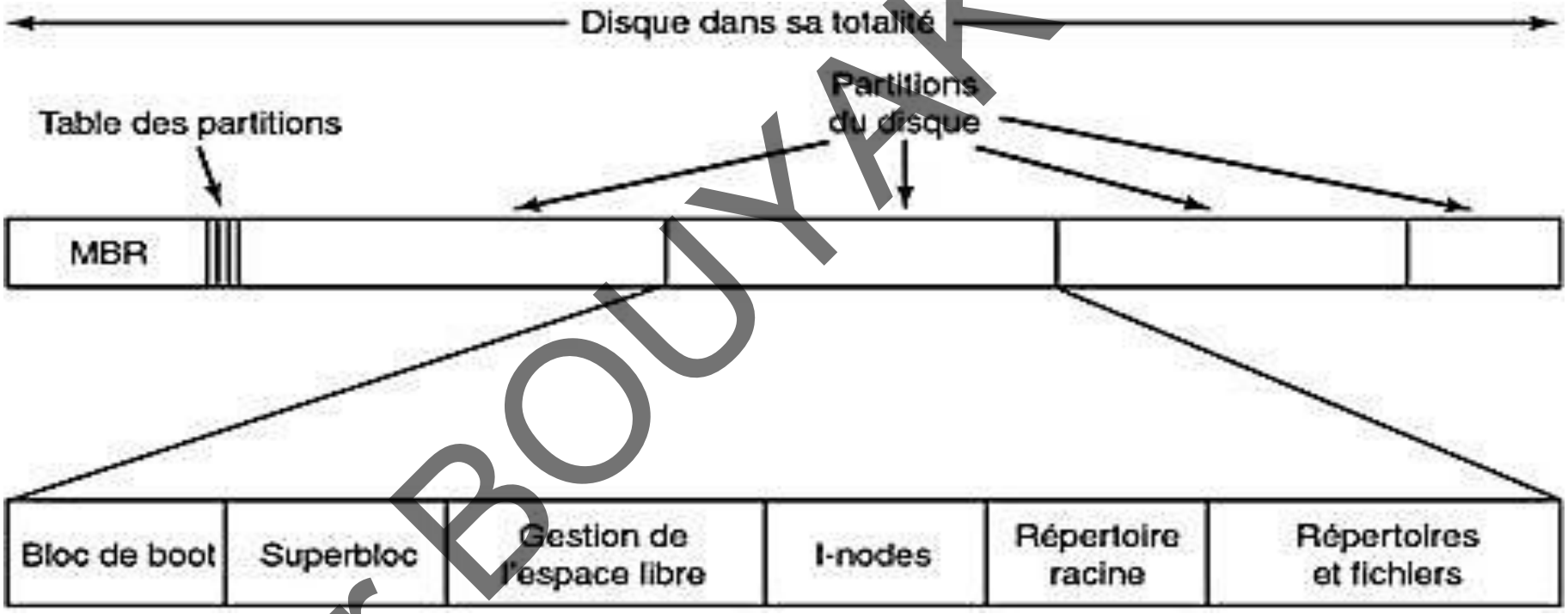
- Le support physique (mémoire secondaire) doit être de grande capacité et à un faible coût. Par contre il possède des temps d'accès beaucoup plus long que les supports volatiles (mémoire centrale).
- Le système d'exploitation doit assurer la correspondance entre la représentation physique des informations et la représentation logique qu'en a l'utilisateur.

➔ le module du SE responsable est le SGF

Définition et objectifs des SGF

- Le **système de gestion de fichiers SGF** est la partie du système d'exploitation qui se charge de gérer les fichiers:
  - Création (identification, allocation d'espace sur disque), suppression, les accès en lecture et en écriture, le partage de fichiers et leur protection.
  - Organiser l'implantation des données sur un disque de manière optimale
  - Permettre au SE d'utiliser l'espace disque pour stocker et utiliser des fichiers
  - Définir une politique d'accès aux fichiers

Organisation possible d'un système de fichiers 1/2



Organisation possible d'un système de fichiers 2/2

- Le ***Master Boot Record* (MBR)**: Le secteur zéro du disque est appelé le MBR et est utilisé pour le démarrage (boot) de la machine.
- La **table des partitions**: donne la partition active et les adresses de début et de fin de chaque partition.
- Le ***bloc de boot*** occupe la première partie du système de fichiers et contient le code qui est lu et mis dans la machine pour amorcer le système (chargement du SE en mémoire).
- Le ***super bloc*** décrit l'état du système de fichiers, le nombre de fichiers qu'il peut stocker, les blocs libres, la taille des blocs…
- La ***liste des i-nœuds***: contient la définition (caractéristiques) des fichiers.
- Les ***blocs de données*** contiennent les données des fichiers
- Enfin une partie ***swap*** pour mémoire virtuelle

Notion de fichier

- Pour le SE, un fichier est une suite d'octets.
- Les utilisateurs peuvent donner des significations différentes au contenu d'un fichier (suites d'octets, suite d'enregistrements, etc.).
- Chaque fichier est identifié par un **nom** auquel on associe un **emplacement** sur le disque et un ensemble de **propriétés**.

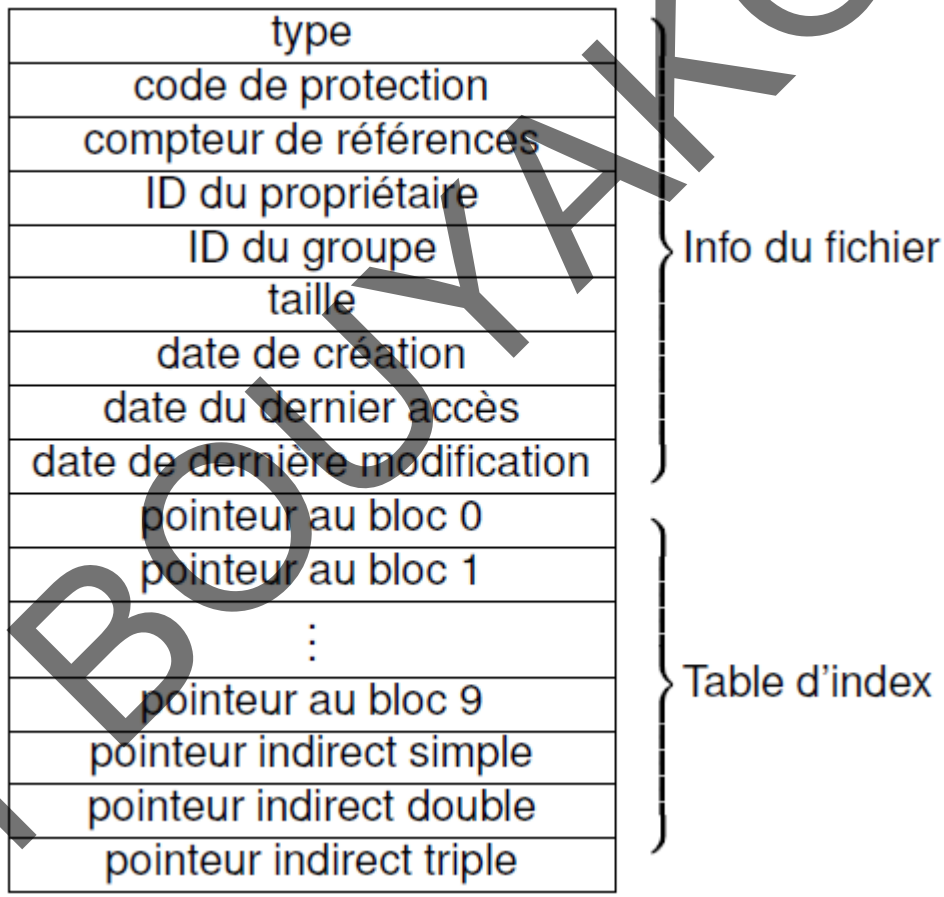
Types de fichiers

- Les **fichiers ordinaires** contiennent les informations des utilisateurs.
- Les **répertoires** sont des fichiers système qui maintiennent la structure du système de fichiers.
- Les **fichiers spéciaux caractère** sont liés aux E/S et permettent de modéliser les périphériques d'E/S.
- Les **fichiers spéciaux bloc** modélisent les disques.
- Dans le système Unix le:
  - "-" désigne les fichiers ordinaires,
  - "d" les répertoires,
  - "c" les périphériques à caractères,
  - "b" les périphériques à blocs,
  - "p" les tubes nommés.

Notion d'i-node (index node) ou i-noeud

- Dans le système Unix toutes les informations des fichiers sont rassemblées dans une structure associée au fichier, appelée **nœud d'information**, **i-nœud** ou **i-node**.
- L'i-node contient les informations suivantes:
  - le type du fichier (ordinaire, répertoire, caractère, bloc ou tube);
  - le code de protection sur 9 bits;
  - l'identificateur et groupe du propriétaire;
  - les dates de création, du dernier accès et de la dernière modification;
  - la taille;
  - la table d'index.

L'i-node



Les répertoires

- Les systèmes d'exploitation modernes adoptent une **structure arborescente** pour représenter le système de fichiers

  ➔Les nœuds de l'arbre sont des répertoires et les feuilles sont des fichiers.
- Un répertoire est composé de fichiers et de sous répertoires.
  ➔ Pour accéder à un fichier, il suffit de spécifier les répertoires du chemin qui mène de la racine de l'arborescence au fichier(chemin d'accès).
- Dans le système Unix, chaque répertoire contient aussi sa propre référence "." et celle du répertoire immédiatement supérieur ".."

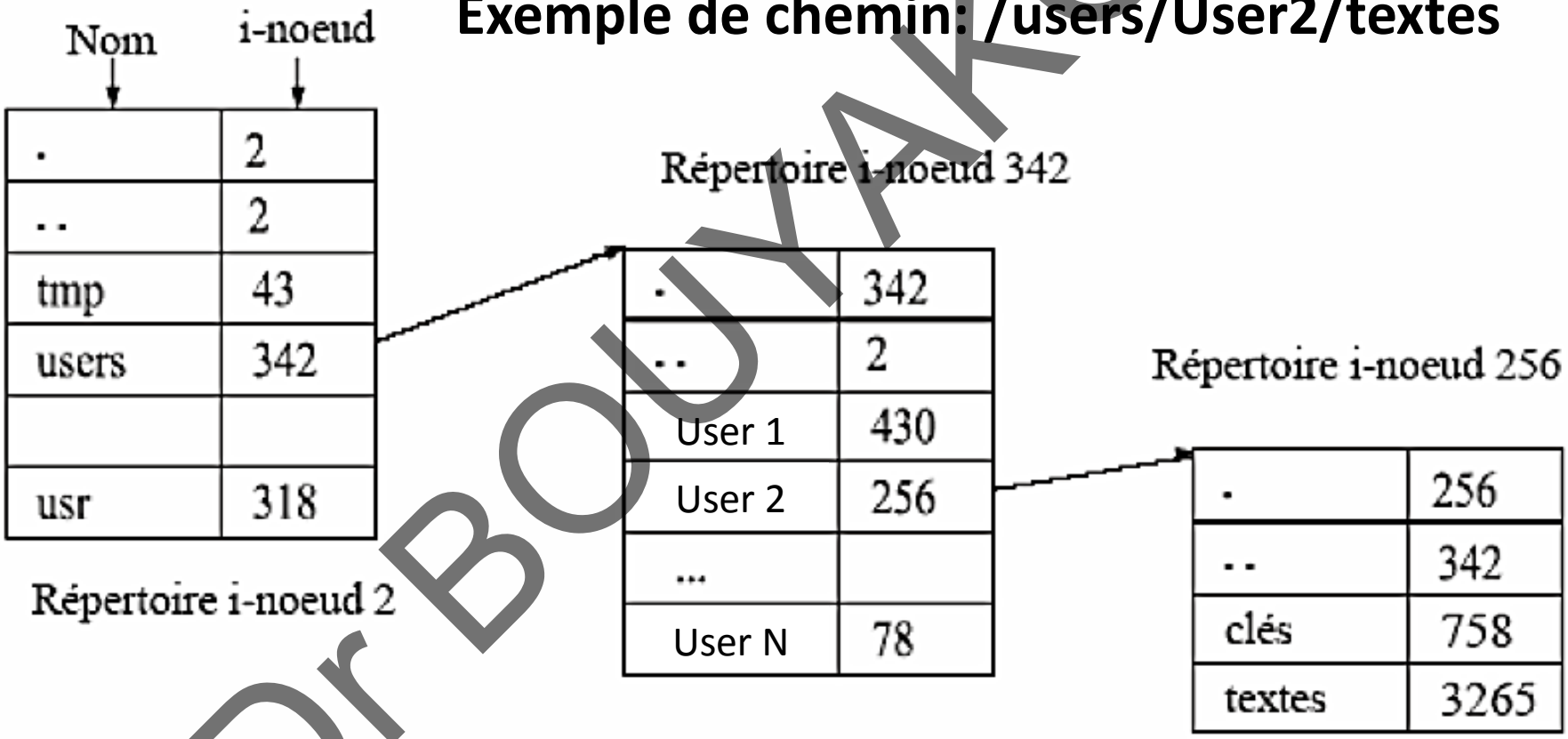
L'i-node des répertoires

- Un répertoire est un fichier qui dispose d'une structure logique: il est considéré comme un tableau qui contient une entrée par fichier. L'entrée du répertoire permet d'associer au nom du fichier (nom externe au SGF) les informations stockées en interne par le SGF.

i-nœud	Nom du fichier
20	.
3	..
100	chap1
2378	chap2
125	scheduler
⋮	⋮

Arborescence des répertoires et i-node

Exemple de chemin: /users/User2/textes



Stockage de fichiers

Politiques d'allocations de blocs

- Les fichiers de données sur les disques se répartissent dans des blocs de taille fixe

  ➔ La lecture ou l'écriture d'un élément d'un fichier impliquera donc le transfert du bloc entier qui contient cet élément.

  Pour un accès rapide, on aura donc intérêt à prendre des blocs de grandes tailles

  mais

  les fichiers, y compris les fichiers de 1 octet, ont une taille minimale de 1 bloc ➔ Si un disque comprend beaucoup de fichiers de petites tailles et si les blocs sont de grandes dimensions, l'espace gaspillé sera alors considérable.

- Il faut donc implanter des techniques adéquates de stockage de fichiers.
  - **Allocation contiguë**,
  - **liste chaînée de blocs**
  - **l'indexation par i-nodes**.

L'allocation contiguë

- Pour chaque fichier à enregistrer, le système recherche une zone suffisamment grande pour accueillir le fichier.

  ➔Le fichier sera alors constitué de plusieurs blocs contigus.
- La table d'allocation de fichiers contient seulement une entrée par fichier, avec le bloc de début et la taille du fichier.

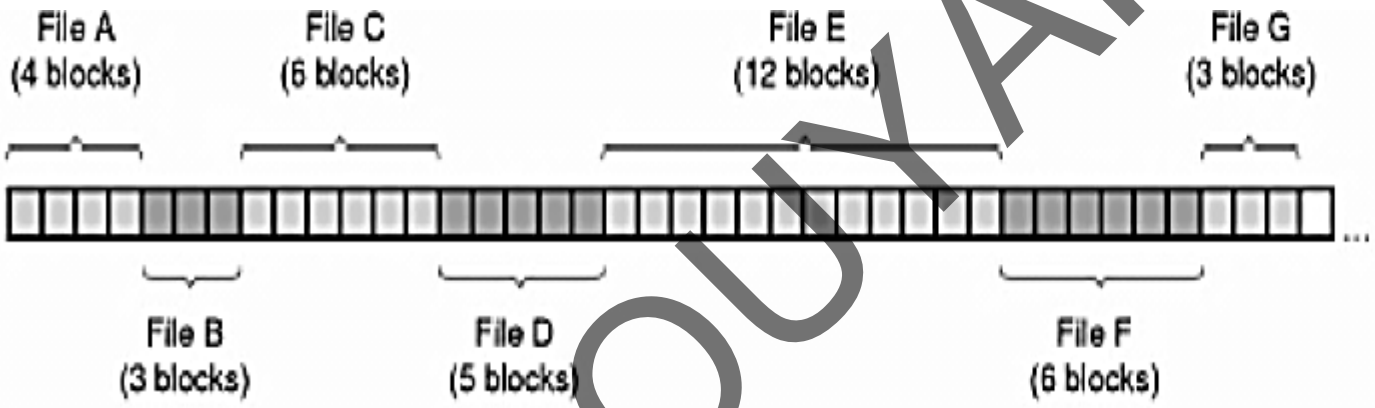
- Cette méthode présente l'avantage de la simplicité, et la rapidité de l'accès ➔ les blocs étant contigus, on limite les déplacements de la tête le lecture/écriture, coûteux en temps.

Les inconvénients de l'allocation contiguë

- Le dernier bloc est généralement sous-utilisé ➔ **gaspille d'espace de stockage**.
- Il est difficile de prévoir la taille qu'il faut réserver au fichier: un fichier est amené à augmenter de taille ➔ il faut prévoir de l'espace libre après le dernier secteur alloué.
- Si le fichier est agrandi, il faudra le **déplacer** pour trouver un nouvel ensemble de blocs consécutifs de taille suffisante.
- La perte d'espace sur le disque: si on prévoit trop d'espace libre, le fichier risque de ne pas l'utiliser en entier. En revanche, si on prévoit trop peu d'espace libre, le fichier risque de ne pas pouvoir être étendu.
- Problème de **fragmentation externe**: c'est l'espace perdu en dehors des fichiers. On peut effacer des données ou supprimer des fichiers ce qui libère des blocs sur le disque.

  ➔Création d'un grand nombre de petites zones dont le total correspond a un espace assez volumineux.

Représentation graphique de l'allocation contiguë



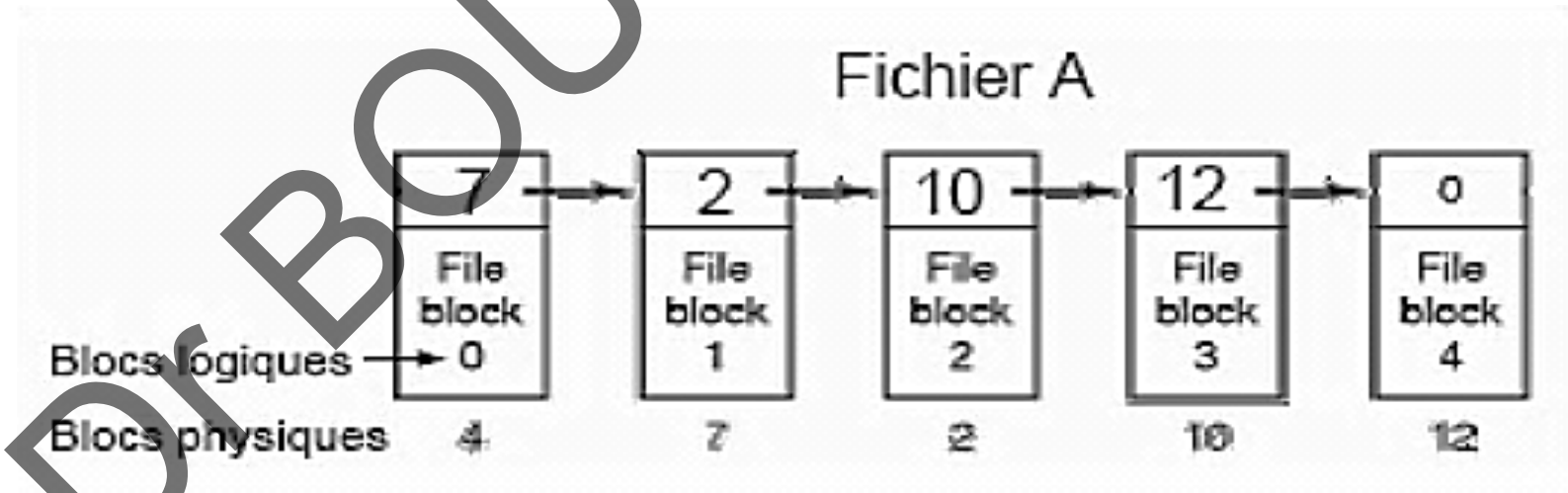
Fichier	Début	Fin
A	0	3
B	4	6
C	7	12
D	13	17
E	18	29
F	30	35
G	36	38

L'allocation par liste chaînée

- Les blocs d'un même fichier sont liés sous forme de **liste chaînée**.
- Chaque bloc contiendra des données ainsi que l'adresse du bloc suivant.
- Le fichier doit mémoriser le numéro du 1er bloc.
- Un fichier peut désormais être éparpillé sur le disque puisque chaque bloc permet de retrouver le bloc suivant.
- Lorsque le fichier change de taille, la gestion des blocs occupés est simple. Il n'y a donc aucune limitation de taille, si ce n'est l'espace disque lui-même.

Exemple d'une allocation chainée

- Si un bloc comporte 1024 octets et si le numéro d'un bloc se code sur 2 octets, 1022 octets seront réservés aux données et 2 octets au chaînage du bloc suivant.



Avantages et inconvénient de l'approche

- Avantages
  - Cette méthode présente l'avantage de l'élimination du problème de fragmentation externe.
- Les inconvénients :
  - L'accès au fichier est totalement séquentiel, on doit toujours commencer le parcours du fichier à partir du début➔ Pour atteindre un élément sur le bloc *n* d'un fichier, le système devra parcourir les *n*-1 blocs précédents.
  - La perte d'un chaînage entraîne la perte de tout le reste du fichier.

Allocation non contiguë indexée

- Tous les inconvénients de l'allocation chaînée peuvent être résolus d'une manière simple: il suffit de retirer les pointeurs des blocs et de les placer dans une structure de données gardée en mémoire centrale, ainsi, les informations sur les numéros de blocs peuvent être obtenue à tout moment.

Cas du système Unix

- Le système Unix associe à chaque fichier un numéro unique d'identification.
- A chaque numéro est associé un ensemble d'informations appelé **nœud d'information** ou **i-nœud**. Parmi les champs de l'i-nœud, la **table d'index** qui indique l'emplacement physique du fichier.
- Elle est composée de 13 entrées.

Structure de la table d'index

- Les 10 premières entrées contiennent les numéros des 10 premiers blocs composant le fichier.

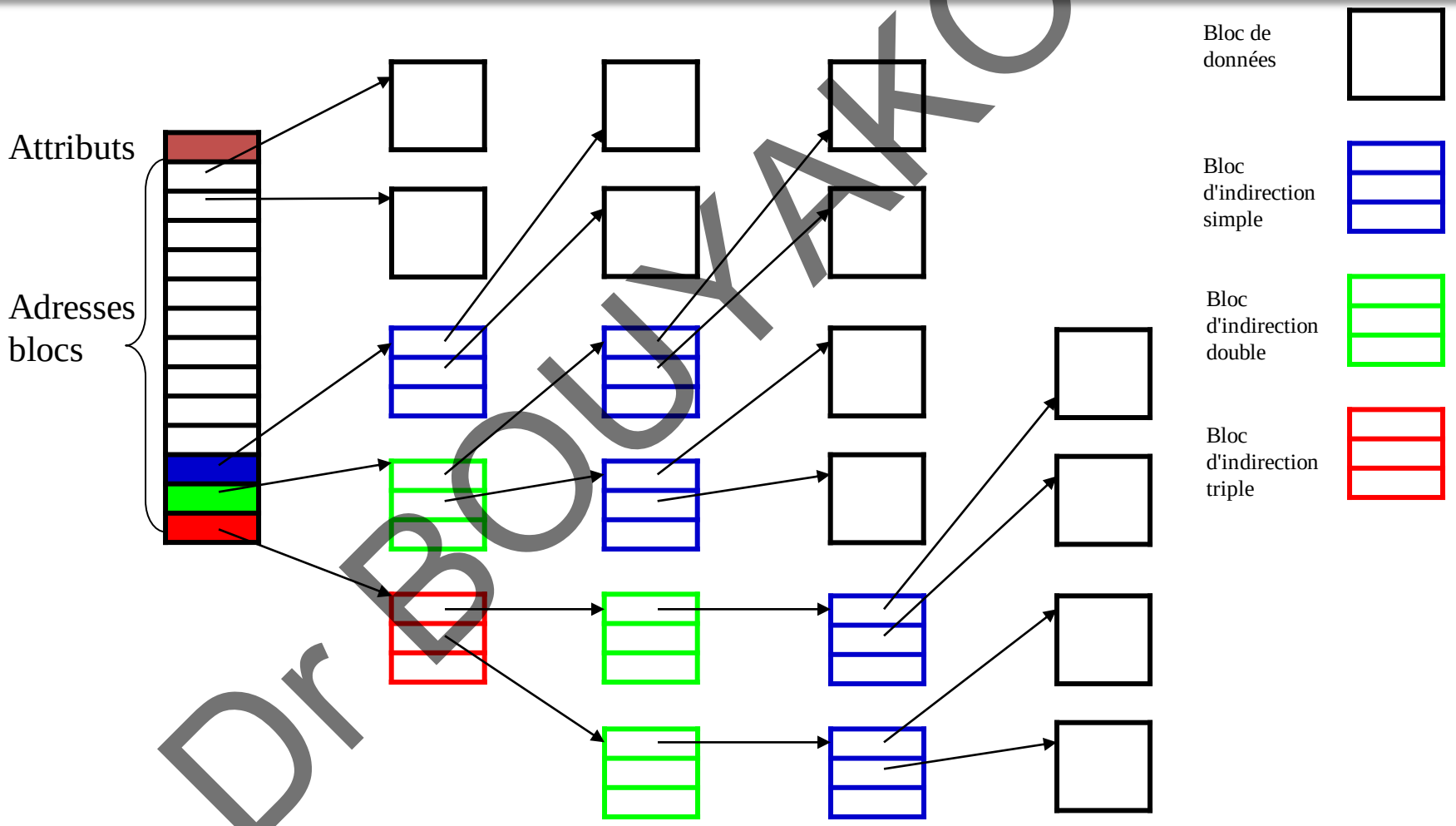
  ➔Pour les fichiers de plus de 10 blocs, on a recours à des indirections.

- L'entrée N° 11 (**simple indirection**) contient le numéro d'un bloc composé lui-même d'adresses de blocs de données. Si les blocs ont une taille de 1024 ø et s'ils sont numérotés sur 4 ø, l'entrée 11 pourra désigner jusqu'à 256 blocs. Au total, le fichier utilisant la simple indirection aura alors une taille maximale de 266 blocs.

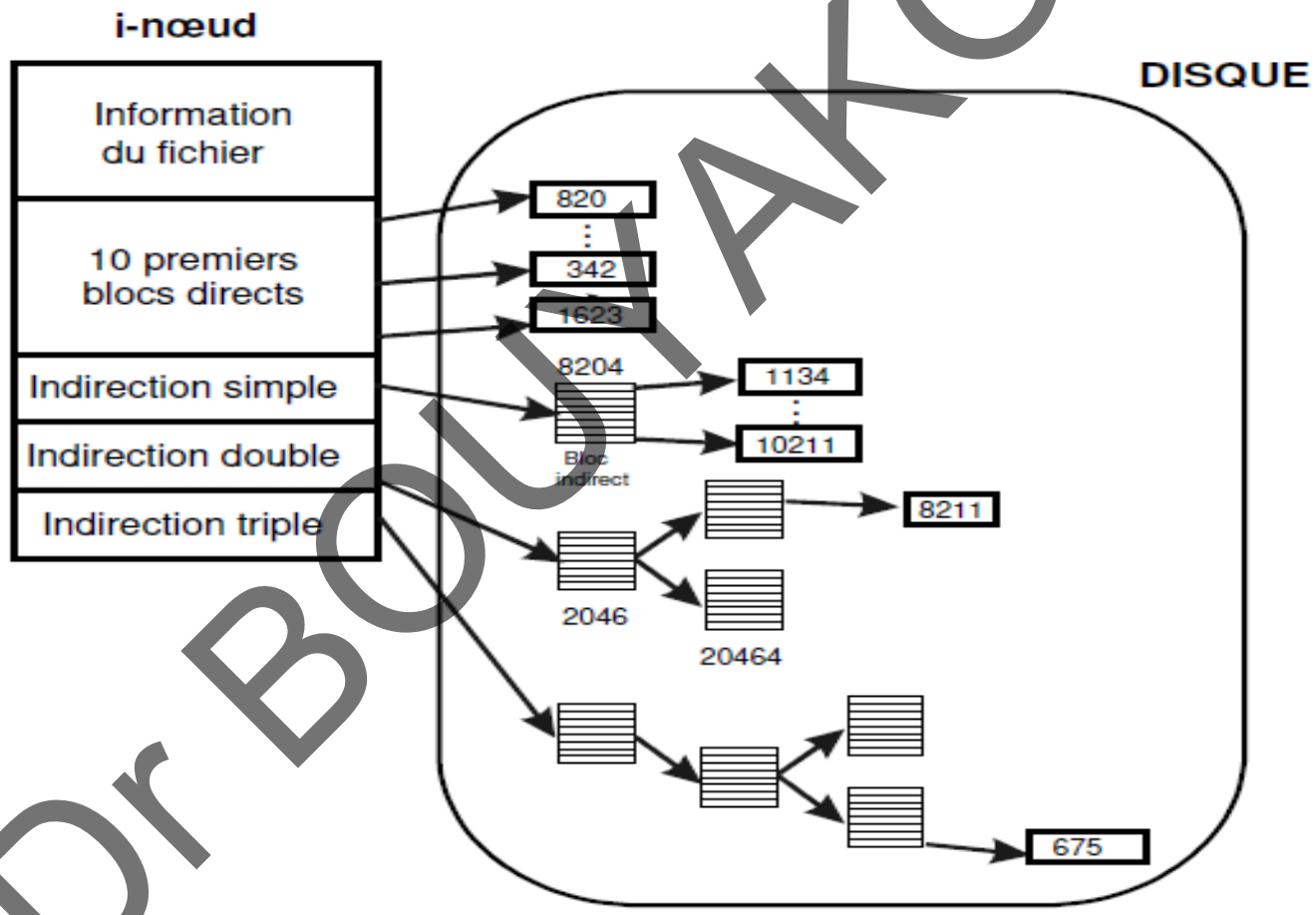
Structure de la table d'index

- L'entrée N° 12 pointe sur un bloc d'index qui contient 256 (en supposant un bloc de 1024 Ø et 4 Ø pour l'@) pointeurs sur bloc d'index dont chacun contient 256 pointeurs sur bloc de données (**double indirection**)
- L'entrée N° 13 pointe sur un bloc d'index qui contient 256 pointeurs sur bloc d'index dont chacun contient 256 pointeurs sur bloc d'index dont chacun contient 256 pointeurs sur bloc de données (**triple indirection**)

Représentation graphique



Structure de la table d'index



La gestion de l'espace libre

Politique de gestion de l'espace libre

- Les systèmes d'exploitation utilisent essentiellement deux approches pour mémoriser l'espace libre:
- Une approche statique ➔Bitmap
- Une approche dynamique ➔liste chaînée

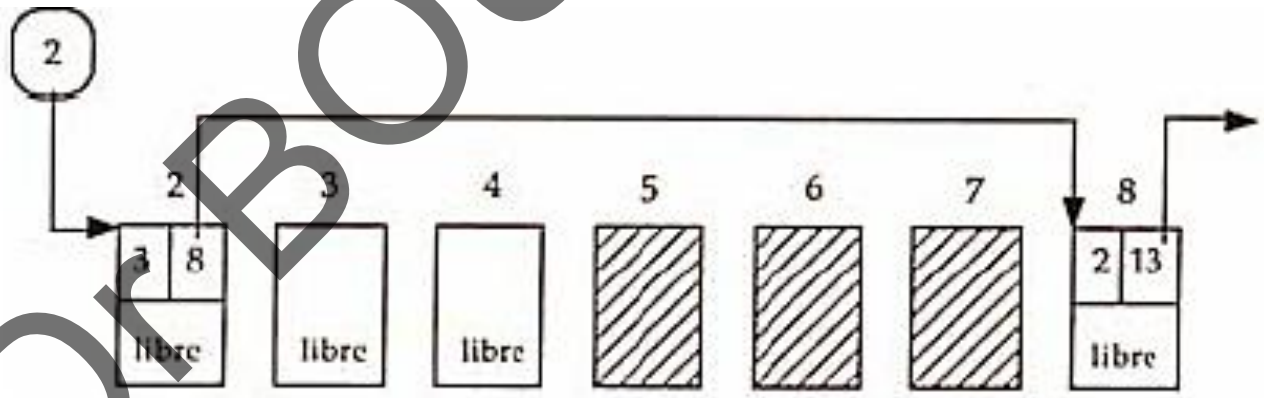
La technique du bitmap

- Utilise une *table de bits* comportant autant de bits que de blocs sur le disque. A chaque bloc du disque, correspond un bit dans la table, positionné à 1 si le bloc est occupé, à 0 si le bloc est libre (ou vice versa).
- Exemple:

Si les blocs 3, 4, 5, 9, 10, 15, 16 sont libres: 1110001110011110 0…

La liste chaînée

- Utilise une liste chaînée constituée d'éléments, chacun mémorisant des numéros de blocs libres. Tous les blocs libres sont liés ensemble par des pointeurs.



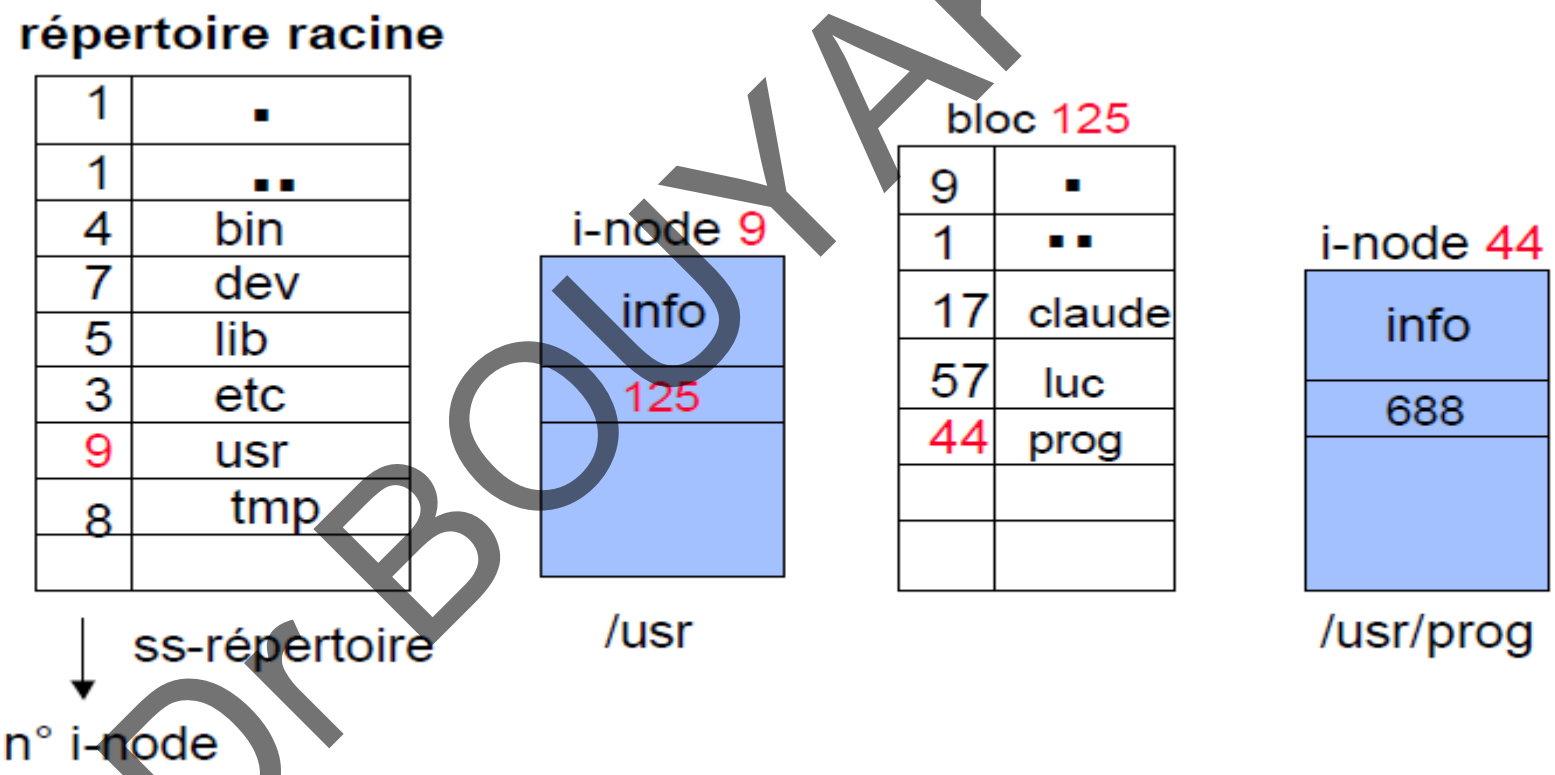
Fichiers et processus

Accès à un fichier par un processus

- Quand un processus se réfère à un fichier par un nom, le noyau analyse le nom du fichier, une composante à la fois, vérifie que le processus a la permission d'inspecter les répertoires du chemin, et finalement extrait l'i-nœud du fichier.

Exemple d'accès à un répertoire

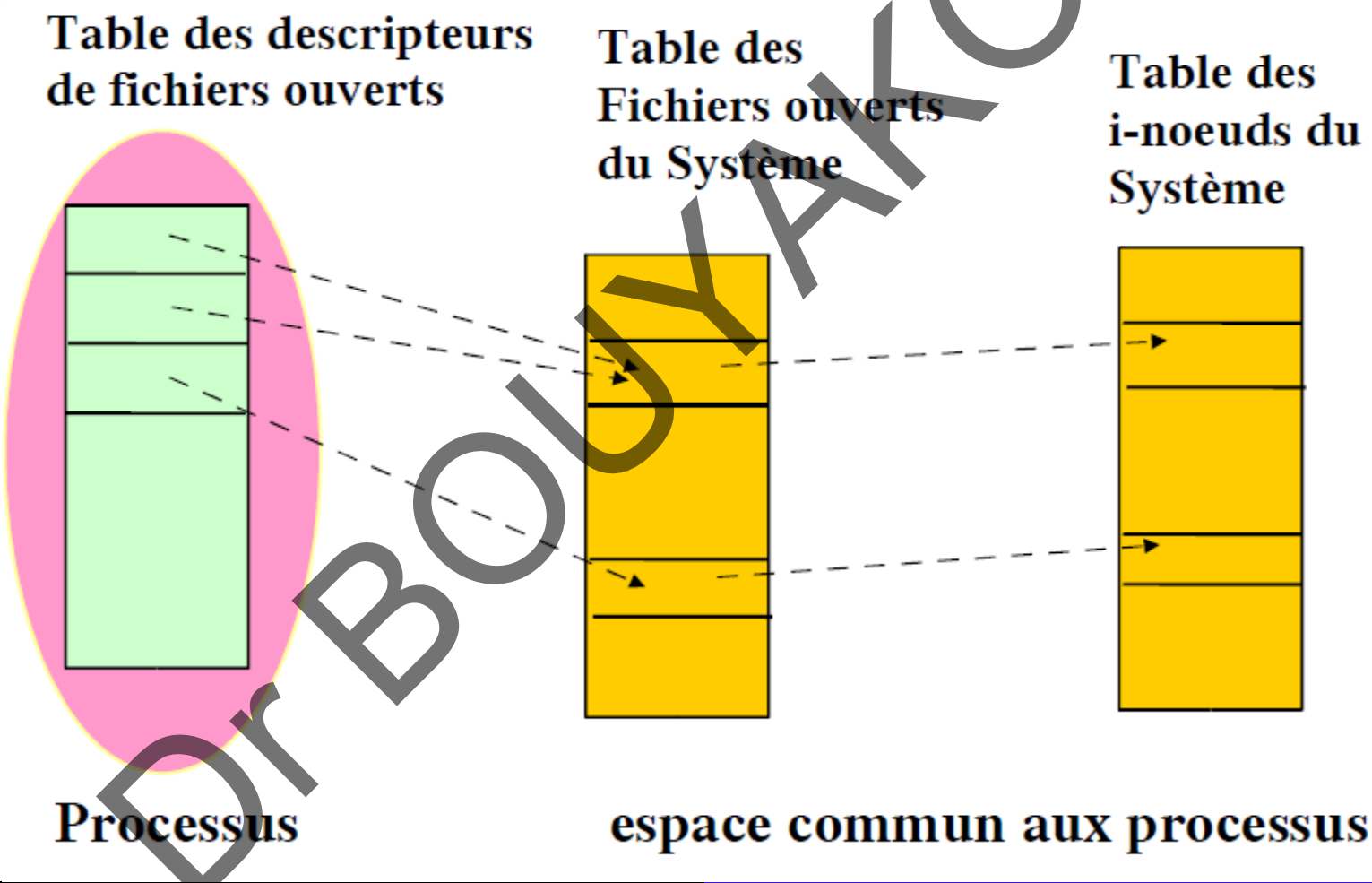
- **/usr/prog**



Création de fichier par un processus

- Quand un processus crée un nouveau fichier, le noyau lui assigne un i-nœud inutilisé.
- Les i-nœuds sont rangés dans le système de fichiers, mais le noyau les lit et les mémorise dans une table des i-nœuds lorsqu'il manipule les fichiers.
- Le noyau contient deux autres structures de données, *la table des fichiers ouverts* et la *table des descripteurs de fichier utilisateur*.

Les différentes tables utilisées pour l'accès au fichier



De la création et l'ouverture à la modification

- Le noyau retourne un descripteur de fichier sur les appels système *open()* et *create()*, qui est un indice dans la table des descripteurs de fichier utilisateur.
- En exécutant les appels système *read()* et *write()*, le noyau utilise le descripteur de fichier pour accéder à la table des i-nœuds, et de l'i-nœud, retrouve les données du fichier.

Questions ?