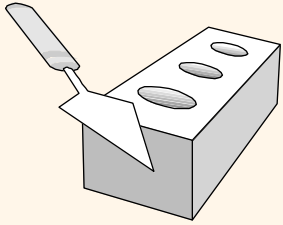


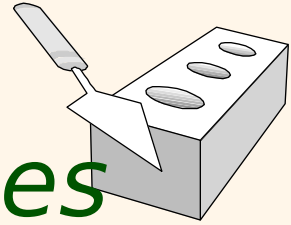
Bases de Données Avancées

Ioana Ileana

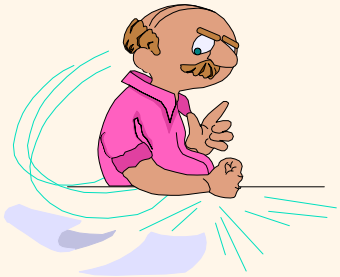
Université Paris Descartes

Cours basé sur le livre (et les diapositives de):
Database Management Systems 3ed, R. Ramakrishnan et J. Gehrke

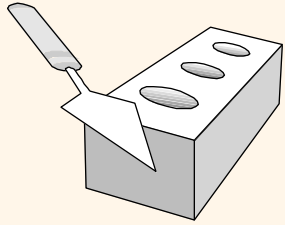
LIPADE
Laboratoire d'Informatique PAris DEscartes



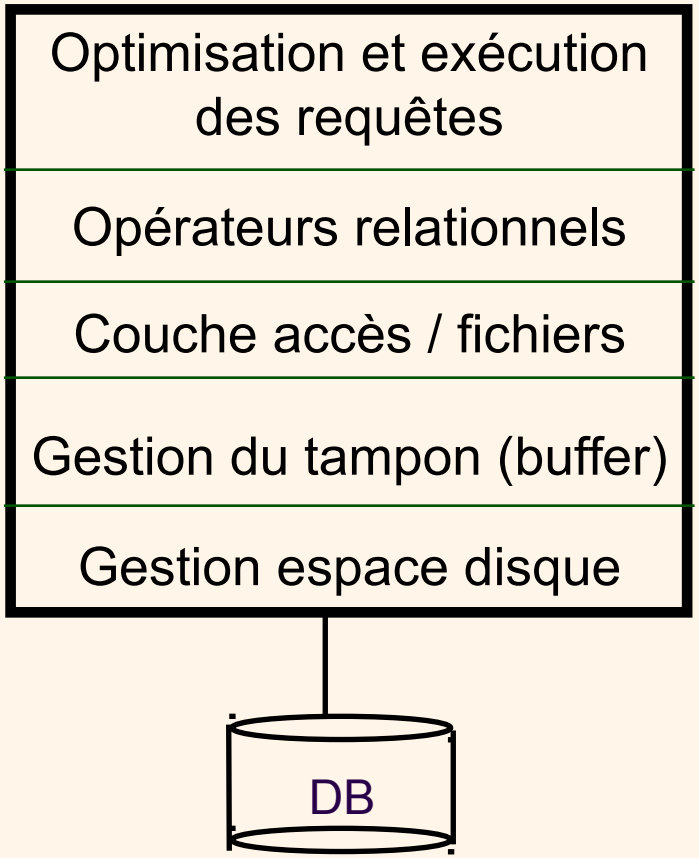
Cours 2: Stockage des données (1ère partie)



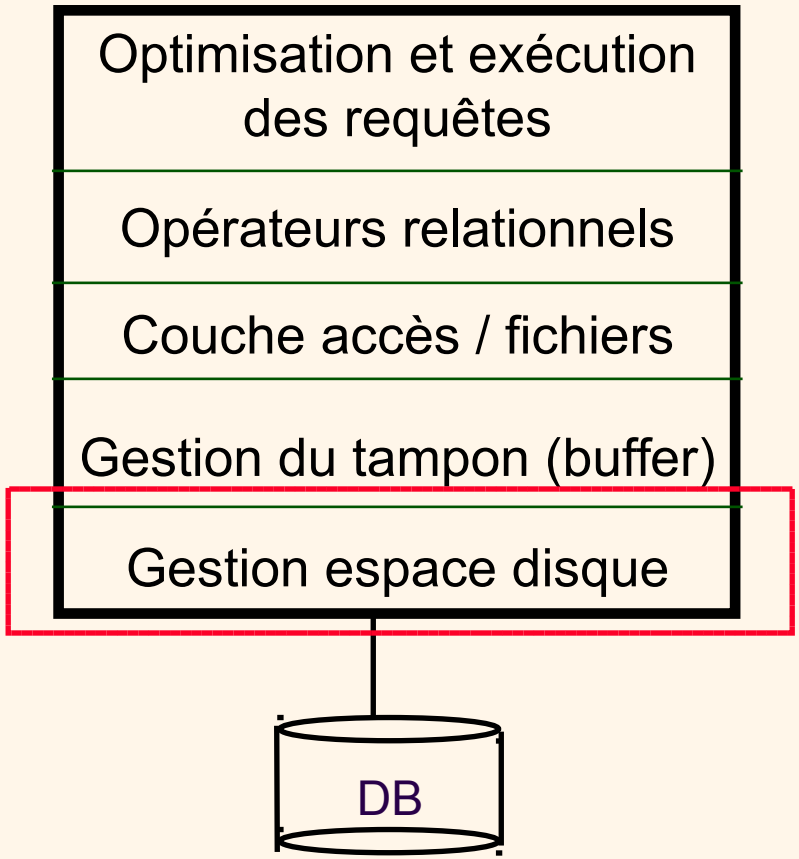
- Diapos traduites et adaptées du matériel fourni en complément du livre Database Management Systems 3ed, par Ramakrishnan et Gehrke ; un grand merci aux auteurs pour la réalisation et la disponibilité de ce matériel !
- Les diapos originales (en anglais) sont disponibles ici : http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- Plus particulièrement, ce cours touche aux éléments dans le Chapitre 9 du livre ci-dessus
- Également, merci à Themis Palpanas pour ses adaptations de diapos en partie reprises dans ce cours !



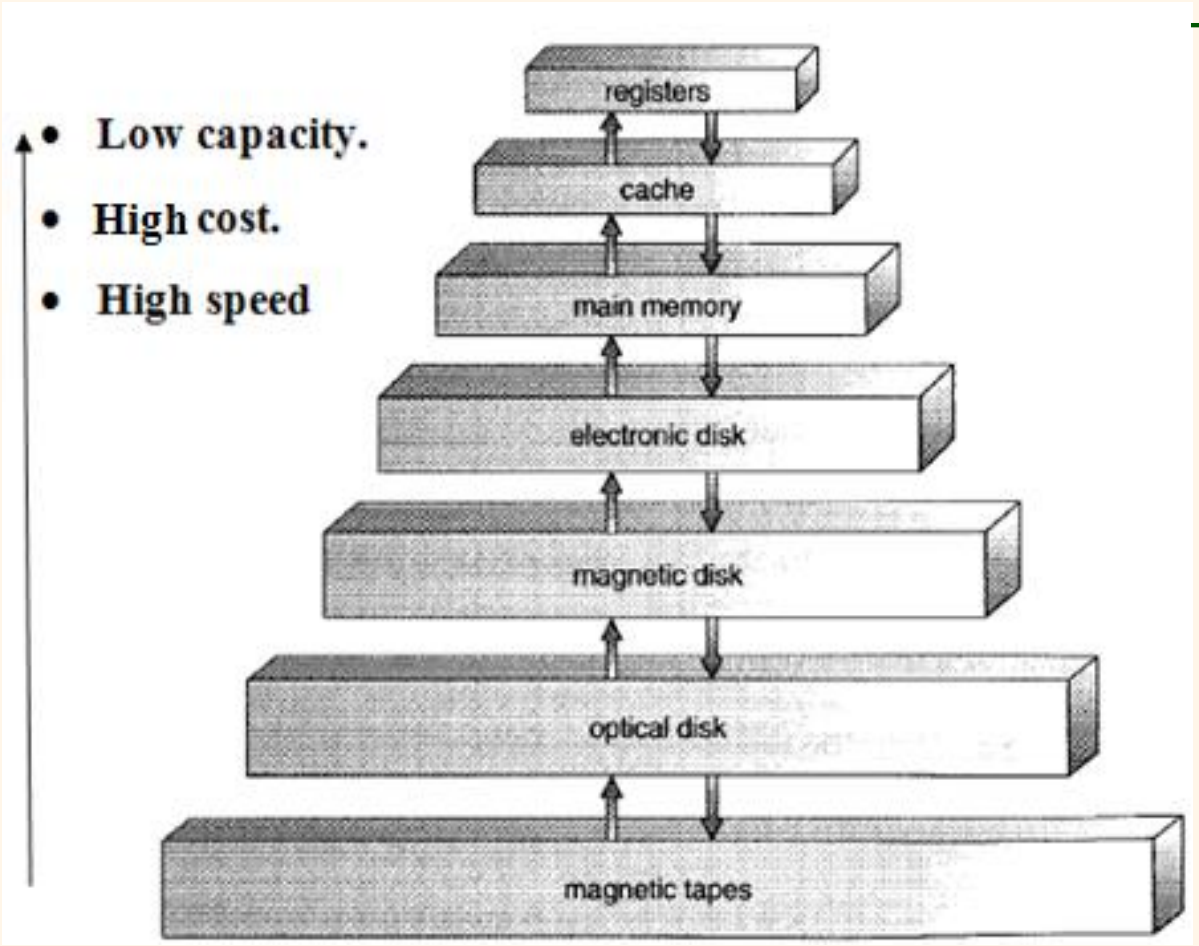
Rappel: structure d'un DBMS



Le gestionnaire de l'espace disque



Hierarchie des types de stockage

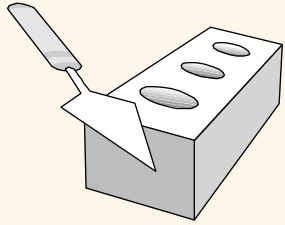


+Petit, +Rapide, +Cher

+Grand, +Lent

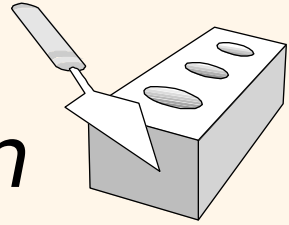
Source: https://www.researchgate.net/figure/283778784_fig2_Figure-3-Storage-device-hierarchy

Cours basé sur le livre (et les diapositives de):
Database Management Systems 3ed, R. Ramakrishnan et J. Gehrke



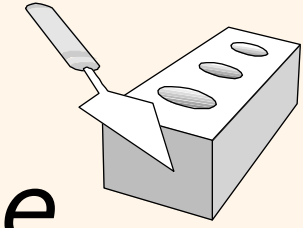
Le stockage disque

- Les données sont généralement stockées sur le(s) disque(s) et non pas en mémoire vive (RAM)
- Ceci a des implications majeures sur le design des SGBDs!
  - Lecture (READ): données transférées / déplacées depuis le disque vers la mémoire vive
  - Ecriture (WRITE): données transférées / déplacées depuis la RAM vers le disque.
  - Les deux sont des opérations coûteuses, par rapport aux opérations impliquant uniquement la mémoire vive; elles doivent dont être planifiées de manière efficace / optimisée!



Pourquoi ne pas stocker le tout en RAM?

- *Trop coûteux.*
- *La RAM est volatile*. Nous voudrions (bien évidemment) que les données soient persistantes entre deux opérations SGBD...
- Hiérarchie typique de stockage des données:
  - Mémoire vive (RAM, main memory) pour les données couramment utilisées.
  - Disque (stockage secondaire – secondary storage) pour les bases de données.
  - Bandes magnétiques (oui, oui...) / disques optiques (stockage tertiaire – tertiary storage) pour archiver (backup) des versions plus anciennes des données.

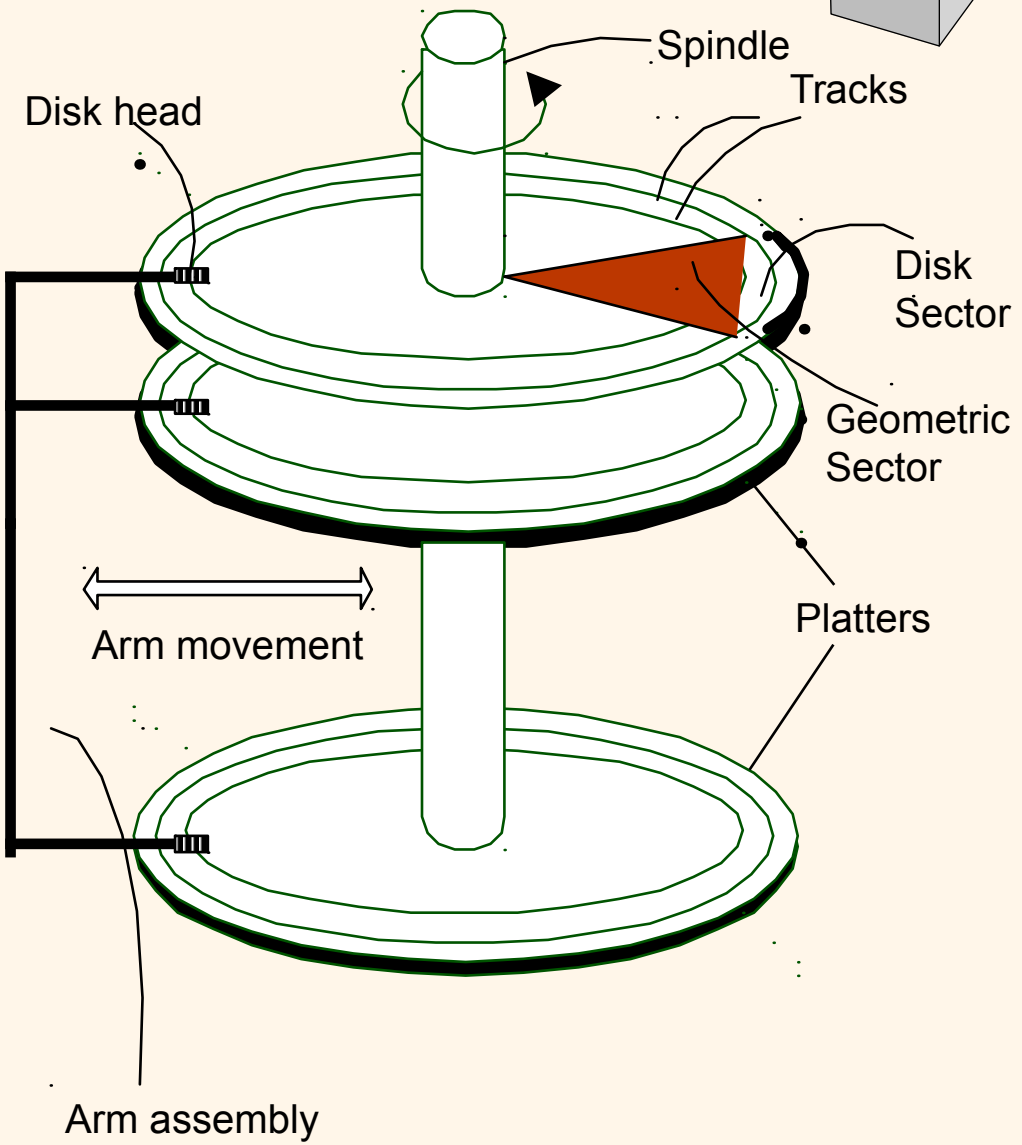


Les disques et les blocs disque

- Les disques = les dispositifs de stockage secondaire les plus connus / utilisés.
- Avantage principal par rapport aux bandes magnétiques: *accès aléatoire (random access)* vs. *accès séquentiel (sequential access)*.
- Les données sont stockées et récupérées par unités appelées *blocs disque (disk blocks)*
  - Le bloc est une « unité de lecture / écriture », aussi appelée « unité entrée / sortie » (en : I/O, Input/Output)
  - Un accès lecture = lecture d'un bloc en entier (de même pour l'écriture)
- En général, si deux « bouts d'information » sont « fréquemment utilisés ensemble », ils est utile de les placer sur le même bloc !
  - Le deuxième sera gratuit ... ;)
- De plus, dans le cas des disques magnétiques, cette notion de « proximité qui baisse le coût » s'étend à plusieurs blocs*, car l'ordre d'accès aux blocs est important !*

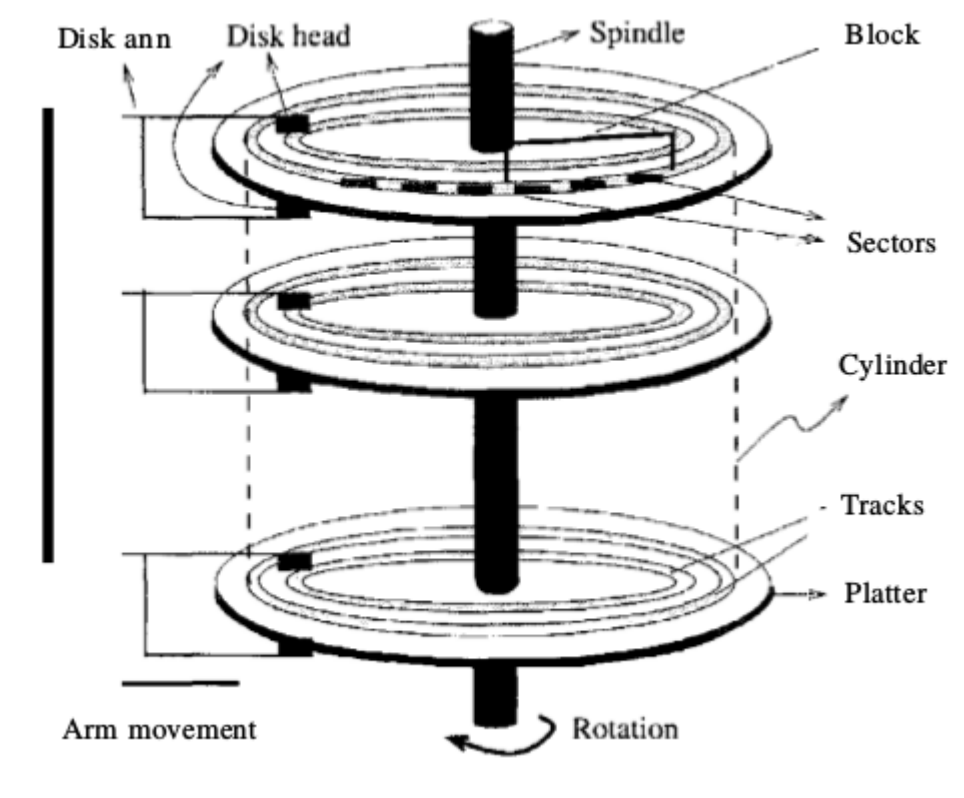
Les disques magnétiques (HDD)

- Les plateaux (platters) tournent (ex 90rps)
- L'assemblage des bras (arms) est déplacé pour positionner une des têtes (heads) sur la piste (track) désirée. Les pistes sous les têtes forment un *cylindre* (imaginaire!).
- Une seule parmi les têtes lit / écrit à un moment donné

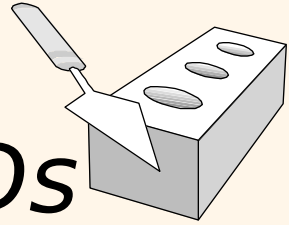


Les disques magnétiques (HDD)

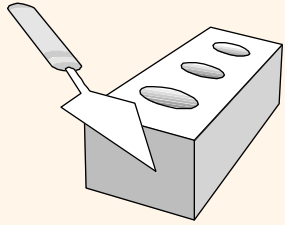
- En général, un bloc comprend plusieurs secteurs consécutifs sur la même piste
- → *La taille (au sens nombre d'octets / bits) d'un bloc (block size)* est un multiple de la taille d'un secteur disque (fixe)



L'accès aux données sur les HDDs

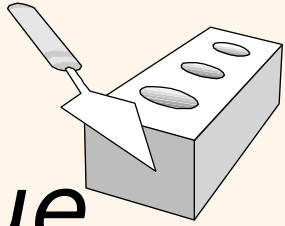


- Le temps pour accéder (lecture/écriture) à un bloc d'un disque magnétique comprend:
  - *Le temps de recherche (seek time)* : déplacer les bras pour positionner une tête
  - *Le temps de latence rotationnelle (rotational delay)* : attendre que (par rotation) le bloc se positionne sous la tête
  - *Le temps de transfert (transfer time)* : le temps passé à déplacer les données depuis / vers la surface du disque
- Le seek time notamment, mais aussi le rotational delay sont dominants → pour réduire le temps I/O : réduire les délais de seek / rotation!



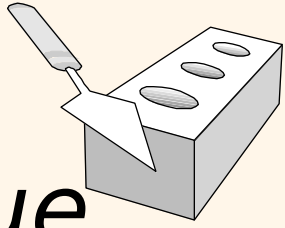
Disposition des blocs

- Concept de “*prochain” (“next”)* bloc:
  - blocs sur la même piste, suivis par
  - blocs sur le même cylindre, suivis par
  - blocs sur un cylindre adjacent
- Les blocs dans un fichier (ou en général une zone de données qu’on souhaite lire “en contigu”) devraient être disposés de manière séquentielle (conformément à la notion de “next”), pour minimiser les temps de recherche et rotation.
- Pour un scan (lecture) séquentiel (sequential scan), le *pre-fetch(“pre-lecture”)* de plusieurs blocs s’avère très utile!



Le gestionnaire de l'espace disque

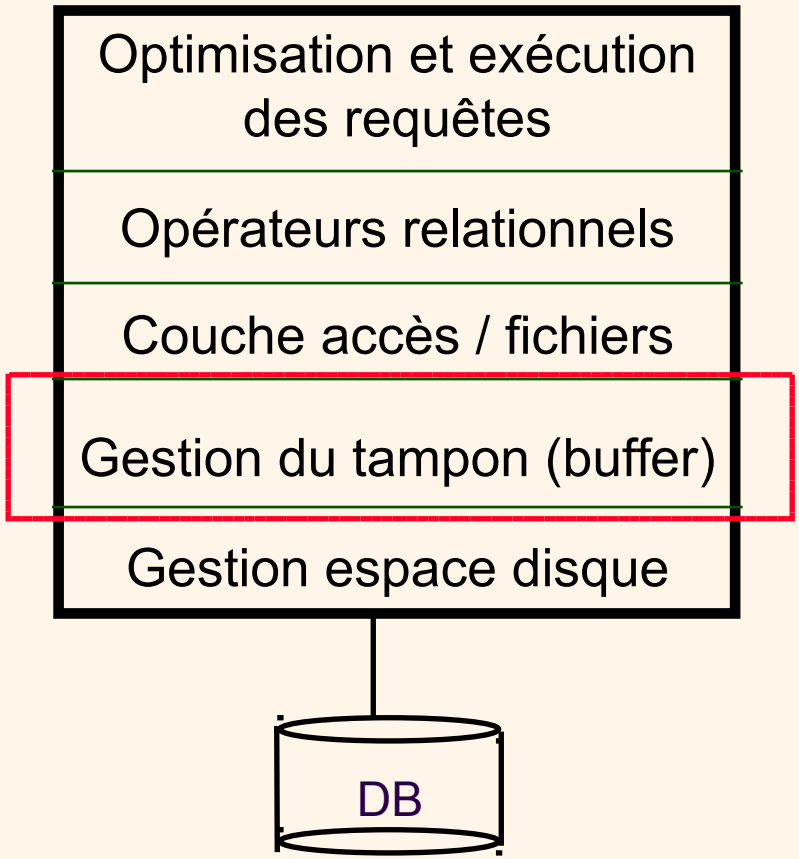
- La couche "de plus bas niveau" du SGBD s'occupe de la gestion de l'espace disque
- Unité de lecture / écriture = une page, *qui correspond à un bloc disque* !
  - Chaque page est stockée sur un bloc
  - Taille page = taille bloc ; ainsi une page est lue / écrite en un seul accès disque !
- Le gestionnaire de l'espace disque « cache » les détails hardware (et souvent OS) aux couches plus hautes, leur offrant la vue des / l'accès aux données en tant qu'un ensemble de pages !



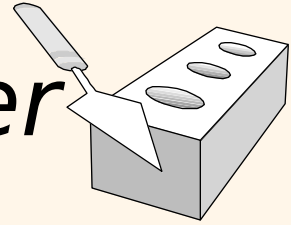
Le gestionnaire de l'espace disque

- Les couches plus hautes font des appels à ce gestionnaire pour:
  - allouer/désallouer une page
  - lire/écrire une page
- Les demandes d'allocation pour une *séquence* de pages doivent être satisfaites en allouant les pages de manière séquentielle sur le disque!
  - Les niveaux au-dessus n'ont pas besoin de connaître les détails de comment cela est fait, ni comment l'espace libre est géré!
- Le gestionnaire de l'espace disque doit en revanche «garder le compte » des blocs disponibles (liste chaînées, bitmap...)

Le gestionnaire du tampon (buffer manager)

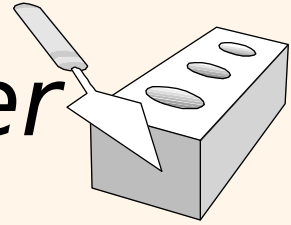


Le gestionnaire du tampon (buffer manager)



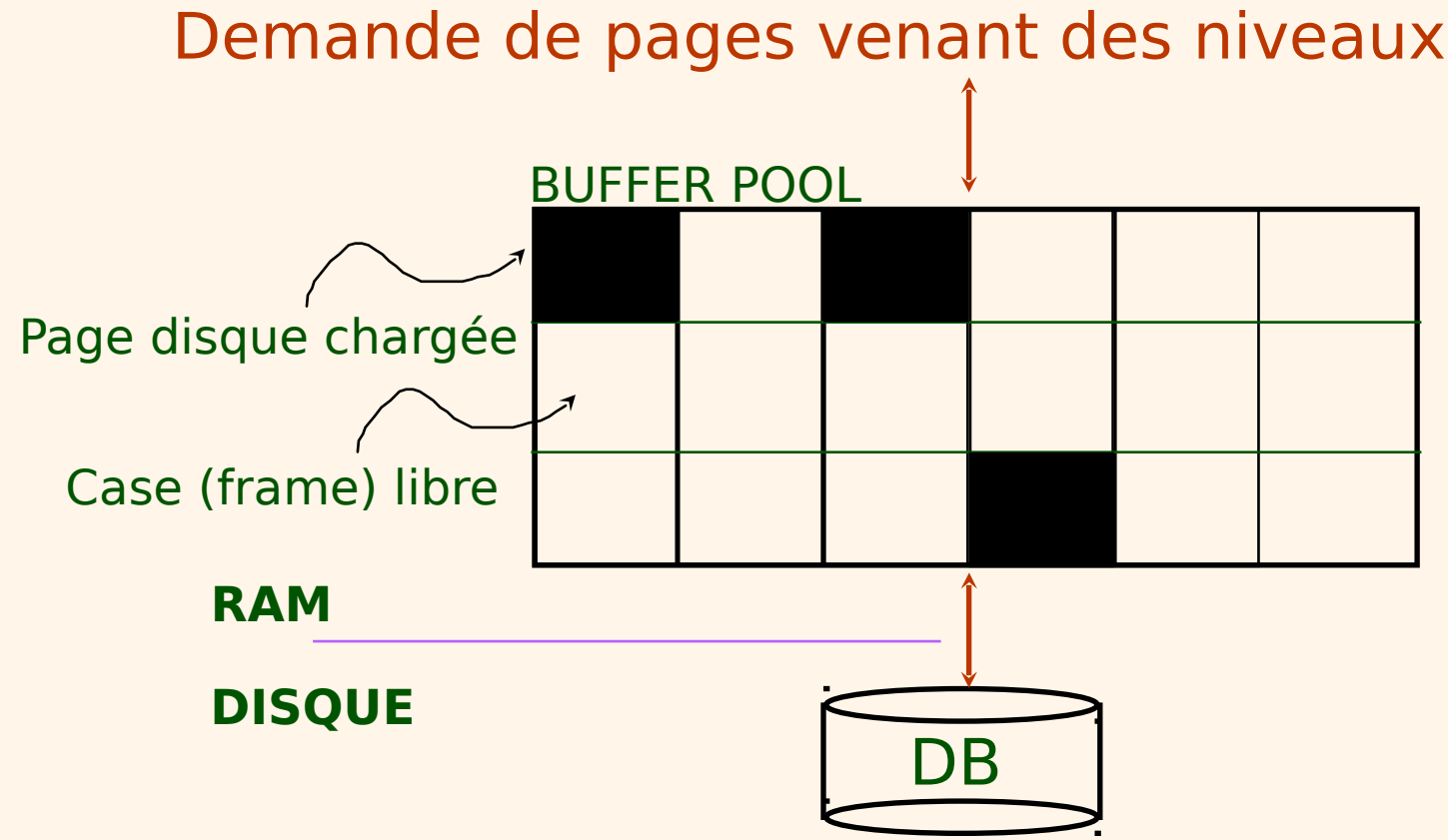
- Les données sont persistées sur le disque *mais doivent être dans la RAM pour que le SGBD puisse agir dessus*
- Le gestionnaire du tampon (buffer manager) = *responsable du transfert des pages entre le disque et la RAM*
  - Suivant les besoins de lecture / écriture des couches plus hautes
  - Ces couches pourront lire et écrire depuis / vers la RAM directement, « oubliant » que les pages sont stockées sur le disque !
- Intérêt essentiel du buffer manager : éviter les « échanges » RAM-disque non nécessaires !

Le gestionnaire du tampon (buffer manager)

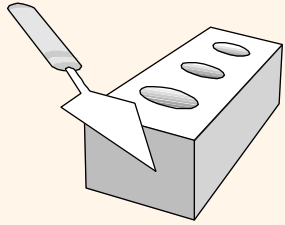


- Le buffer manager est responsable du transfert des pages entre le disque et la RAM (dans un sens et dans l'autre!)
- Pour cela, il partitionne la mémoire disponible en un ensemble de cases (ou cadres ; en : frames) qui seront « remplies » avec des contenus de pages disque
  - En fonction des demandes d'accès à des pages émanant des couches plus hautes
  - Une frame = un buffer ; l'ensemble des frames = le « buffer pool »

Le gestionnaire du tampon (buffer manager)

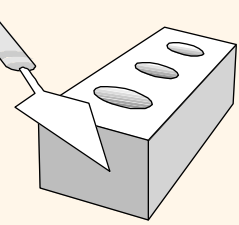


Buffer manager : pages « déjà chargées » dans les frames



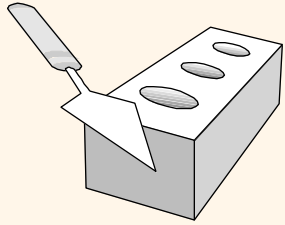
- Le buffer manager doit évidemment se souvenir de « quelle page est dans quelle frame » !
  - *Si une page donnée a déjà été chargée dans une frame, il ne faudrait surtout pas refaire un accès disque pour la re-charger lors d'une deuxième demande !*
  - Il convient donc de stocker pour chaque frame « occupée » l'identifiant de la page (PageId) déjà chargée dans la frame !
  - Lors d'une demande de page venant des niveaux plus hauts, avant de faire un chargement depuis le disque, le buffer manager vérifiera d'abord si la page existe déjà dans une frame.

Buffer manager : modifications et flag « dirty »



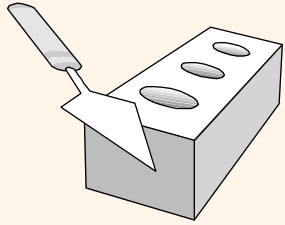
- La modification des pages en RAM, en passant par le buffer manager : très bien MAIS les données doivent être persistées sur disque !
  - Que se passe-t-il quand on arrête le logiciel ? Les modifications courantes en RAM doivent être écrites sur disque, autrement elles sont perdues !
- On pourrait penser à écrire sur disque toutes les pages couramment chargées, mais cela pourrait faire beaucoup plus d'accès disque que nécessaire
  - Car il peut y avoir de nombreuses pages qui ont été juste *lues, et non pas modifiées !*
- Solution : maintenir pour chaque case un flag (booléen, bit 0 ou 1…) « *dirty* » qui nous dit si la page dans la case a été modifiée ou pas
  - Lorsqu'une page doit être « enlevée de la RAM » on sait qu'on doit l'écrire sur disque ssi son flag dirty = 1 !

Buffer manager : pin_count et remplacements



- Que faire si *toutes les frames sont occupées* et que le buffer manager reçoit une demande de page qui ne figure dans aucune frame (une page qu'il faut donc charger depuis le disque)?
  - → Besoin de *remplacer* le contenu d'une frame !
  - Q : Peut-on remplacer le contenu de n'importe quelle frame ??
- Pin_count = compteur d'utilisations d'une page ; contient le nombre d' « utilisateurs en cours» pour une page donnée (chargée dans une frame)
  - Les clients du buffer manager « demandent » et « libèrent » explicitement les pages, ce qui incrémente, respectivement décrémente le pin_count !
- Une page « en cours d'utilisation » ne peut pas être remplacée !!!
  - → Une page peut être remplacée ssi son pin_count = 0 !!

Buffer manager : synthèse

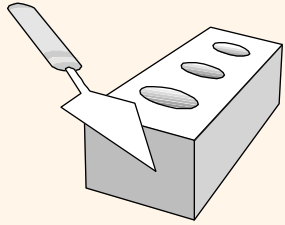


- Le buffer manager doit garder une table d'informations qui associe à chaque frame les trois informations suivantes :

  <PageId, pin_count, dirty>

- Toutes les frames ont initialement leur pin_count et dirty à 0
- Quand une page est demandée:
  - Si la page demandée n'est pas dans une des frames :
    - S'il n'existe plus de frame libre :
      - Choisir une frame parmi celles dont le contenu n'est pas utilisé couramment (pin_count=0) pour *remplacer* son contenu ;
      - Si la frame est marquée comme "dirty", écrire d'abord son contenu sur le disque puis remettre son dirty à 0
    - Sinon, choisir une frame libre
    - Charger (lire) le contenu de la page demandée dans la frame choisie
  - Incrémenter le pin_count (compteur d'utilisations) associé à la frame de la page et retourner la page.
- Le demandeur de la page devra la "libérer" en fin d'utilisation (ce qui permettra de décrémenter le pin_count), et indiquer si la page a été modifiée (cela ajustera le flag *"dirty"*)

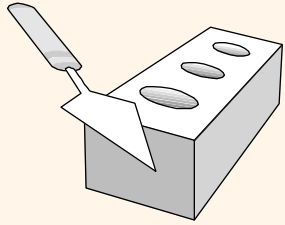
  ☛ *Si les demandes de pages peuvent être anticipées (ex: scan / parcours) séquentiel, plusieurs pages peuvent être pre-fetched en même temps!*



Politiques de remplacement (replacement policies)

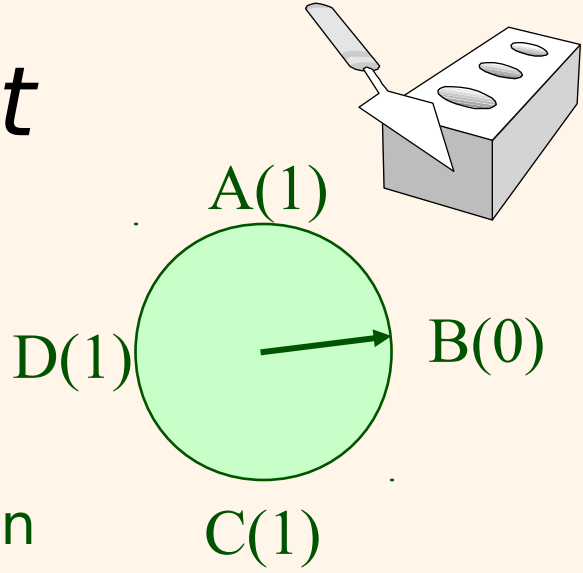
- Quand toutes les cases sont pleines et une demande de “nouvelle” page arrive, le gestionnaire du buffer doit *choisir une case (parmi celles à pin_count = 0) pour remplacer son contenu*
- Ce choix est fait suivant une *politique de remplacement (replacement policy):*
  - Exemples: le moins récemment utilisé (Least-recently-used: LRU), MRU, Clock, etc.
- Cette politique peut avoir un impact important sur le nombre d’opérations I/O; en effet, suivant le type d’accès aux pages (access pattern), certaines politiques peuvent s’avérer sensiblement plus performantes que d’autres !

Politique de remplacement LRU (le moins récemment utilisé)



- *Least Recently Used (LRU)*
  - Pour chaque case/page dans le buffer pool, noter la dernière fois quand son pin_count est passé à 0 (elle a été *unpinned*)
  - Remplacer le contenu de la case qui a été « unpinned » le moins récemment (« le plus ancien temps de unpin »)
  - Politique très répandue : intuitive et simple
    - Marche bien pour des accès répétés à des pages “populaires”
- *Problème: “flooding séquentiel »*: LRU + répétition d’un scan séquentiel.
  - N cadres et N+1 pages lues en séquentiel; dès la deuxième lecture chaque demande de page cause un remplacement (donc une opération I/O).
  - D’autres politiques plus intéressantes ?

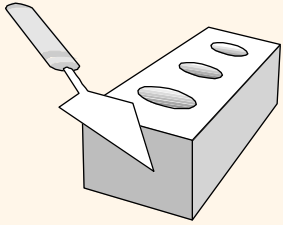
Politique de remplacement « horloge » (« clock »)



- Une approximation de LRU
- Cadres arrangés dans un cycle, chacun a un *reference bit*
  - Peut être vu comme le bit de la “deuxième chance” :) (2nd chance bit)
- Quand le pin_count descend à 0, activer (=mettre à 1) le ref bit
- Quand il y a besoin de remplacement:

```
do for each page in cycle {
        if (pin_count == 0 && ref bit == 1)
            ref bit = 0;
        else if (pin_count == 0 && ref bit == 0)
            choose this page for replacement;
} until a page is chosen;
```

Gestionnaires disque / tampon vs l'OS (mémoire virtuelle, système de fichiers) ?



Pourquoi ne pas laisser la gestion des buffers et de l'espace disque à la charge entière des fonctionalités déjà fournies par l'OS ?

- Différences dans le support en fonction des OS : problème de portabilité
- Certaines limitations, ex: un fichier ne peut pas être "à cheval" sur deux disques, ou limitations sur la taille max d'un fichier…
  - Mais le gestionnaire de l'espace disque peut tout de même choisir d'utiliser des fichiers «classiques» pour stocker les pages de données !
- La gestion des buffers dans un SGBD nécessite de pouvoir faire des actions spécifiques:
  - "Pin" une page (augmenter son pin_count) dans le buffer pool, forcer une page vers le disque (important pour implémenter le contrôle de la concurrence et la récupération en cas de crash)
  - Ajuster la *politique de remplacement,* et pouvoir pre-fetch les pages en fonction de patterns d'accès correspondant à des opérations typiques BD (qui ne sont pas connues par l'OS).