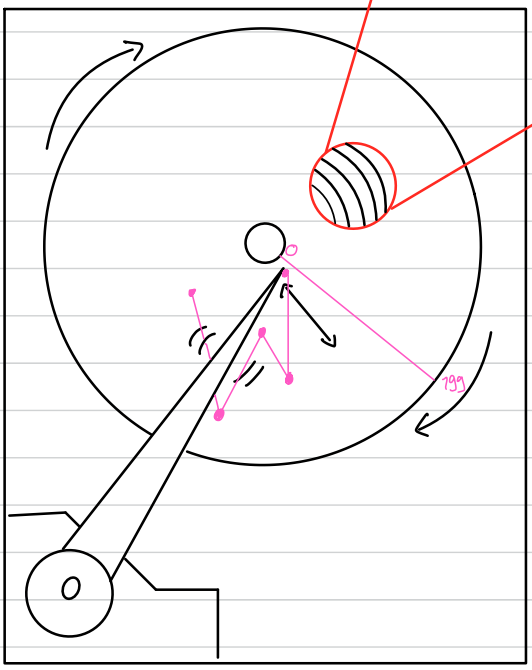
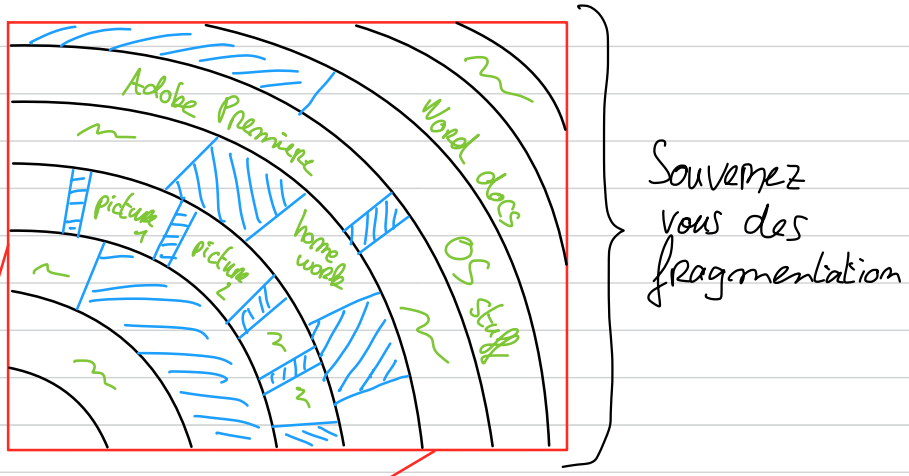
Mémoire en masse

- Bien visualiser





IOPS = throughput / block_size
$T_{io} = T_a + T_l + T_t = 1/IOPS$
T_t = block_size / R_s
throughput = $R_s * T_t / T_{io}$
cas optimal: $T_r = T_{io}$

- Performances disques → formules sur fiche !
  ↳ que veulent dire les variables !

- Connaissez les algos
  - FCFS
  - Shortest-seek-time-first (famine)
  - Scan (scanning mouvement donc touche bord
  - Look (cherche spécifiquement)
  - C-Scan } même mais 1 seul sense ! tête retourne
  - C-Look }

- Connaissez + comprennez vos RAIDs !!

- Bits de parité :

that might be starting to fail or has already failed.

Memory systems have long detected certain errors by using parity bits. In this scenario, each byte in a memory system has a parity bit associated with it that records whether the number of bits in the byte set to 1 is even (parity = 0) or odd (parity = 1). If one of the bits in the byte is damaged (either a 1 becomes a 0, or a 0 becomes a 1), the parity of the byte changes and thus does not match the stored parity. Similarly, if the stored parity bit is damaged, it does not match the computed parity. Thus, all single-bit errors are detected by the memory system. A double-bit-error might go undetected, however. Note that parity is easily calculated by performing an XOR (for "eXclusive OR") of the bits. Also note that for every byte of memory, we now need an extra bit of memory to store the parity.

Parity is one form of **checksums**, which use modular arithmetic to

- Hamming code

Entraînement !

- Bon les exos sur les algos des disques ✓ sont pretty straight foward mais vous pouvez demander si vous avez un doute (no shame !)

(c) (2 points) Avec un système avec codage de Hamming à parité paire calculée de gauche à droite, vous recevez le code de Hamming 12 bits suivant:

0 0 1 1 1 0 0 1 0 1 1 1

$\bar{P}_1$ $\bar{P}_2$ $\bar{P}_4$ $\bar{P}_8$

Ce code a une erreur de 1 bit, quel bit ?

On nous a envoyé un code de Hamming, donc les bits de parités sont déjà dans la chaîne :

P_1 : 0 0 1 1 1 0 0 1 0 1 1

P_1 est ici

(En commençant au bit de parité)

P_1 c'est, on prend 1, skip 1, repeat

On a donc P_1 : 011001 = 1 car n°1 est impair

Rappel (voir plus haut) : on compte les 1

impair → 1
pair → 0

P_2 : 0 0 1 1 1 0 0 1 0 1 1

P_2 est ici

Prend 2, skip 2, repeat

On a donc P_2 : 010011 = 1

Tip ⚠ : Ici, ou bien dans le Hamming coding, si vous prenez un autre bit de parité dans vos "prendre x, skip x, repeat", alors vous avez mess up vos "take + skip".

Donc marqué tout de suite $P_1, P_2, P_4 \ldots$ sur la chaîne.

P_4 : Prend 4, skip 4, repeat (you get the deal)

0 0 1 1 1 0 0 1 0 1 1 1

P_4

P_4 : 11001 $= 1$

P_8 : Prend 8, skip 8

0 0 1 1 1 0 0 1 0 1 1 1

P_8

P_8 : 10111 $= 0$

Finalement

P_1 : 011001	$= 1$	1
P_2 : 010011	$= 1$	$+2$
P_4 : 11001	$= 1$	$+4$
P_8 : 10111	$= 0$	$+0$ car pas 1
		$=$
		$\neq$

Si on veut fix la chaîne ? On switch juste le bit 7 ☺

0 0 1 1 1 0 0 1 0 1 1 1

0 0 1 1 1 0 1 1 0 1 1 1

et pour l'octet correct on retire les bits de parité souligné

Fichiers

Question 4. *Système de fichiers (4 points au total)*

(a) (3 points) Un système de fichiers **ext2** utilise un adressage **32 -bit** et une taille de bloc de **2 KB**.

i. (1 point) Quelle est la taille maximale qu'un fichier peut avoir pour qu'il n'ait pas besoin d'utiliser le pointeur simple (*single*) indirection dans l'inode (en B, peut être une expression) ?

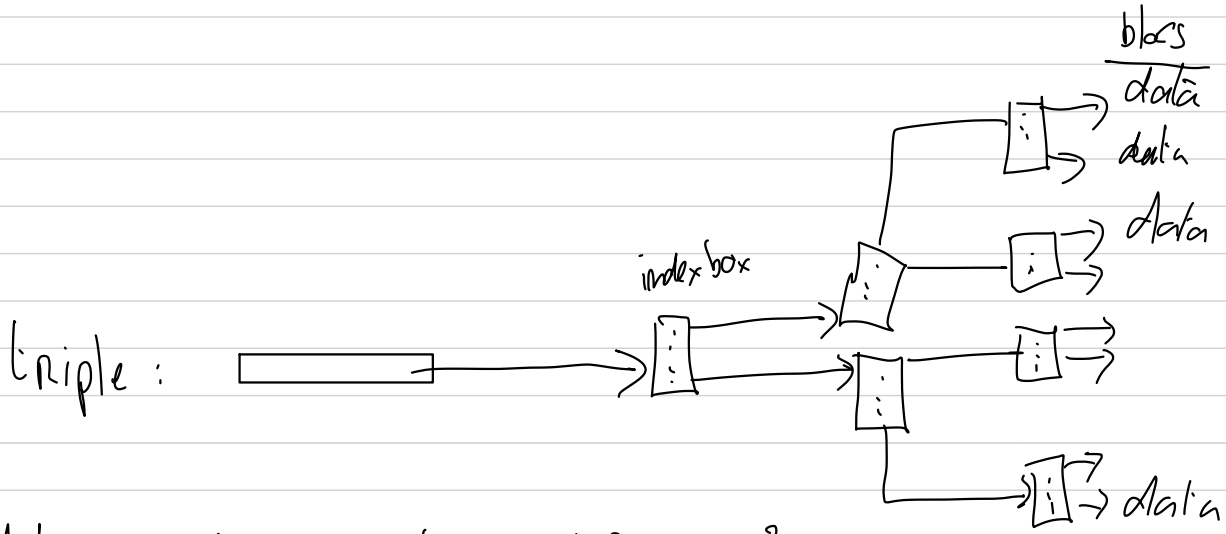
ii. (1 point) Quelle est la taille maximale qu'un fichier peut avoir pour qu'il n'ait pas besoin d'utiliser le pointeur double indirection dans l'inode (en B peut être une expression) ?

iii. (1 point) Quelle est la taille maximale qu'un fichier peut avoir pour qu'il n'ait pas besoin d'utiliser le pointeur triple indirection dans l'inode (en B, peut être une expression) ?

(b) (1 point) Cela décrit le mieux quel type de système de fichiers ? " Similaire à un système de fichiers qui est en chaîne mais avec un meilleur accès aléatoire "

Rappel ext utilise 12 pointeurs pour direct
3 autres pointeurs pour single, double et triple

2 KB = ? $2 = 2^1$ $KB = 2^{10}$ donc 2^{11} B



Adresse de 32 bits = 4B = 2^2 B

donc nombre de rangés possibles dans une index box avec des adresses de 32b ? $\Rightarrow \frac{2^{11}}{2^2} = 2^9$

(b) (2 points) Compte tenu des informations suivantes:
— Taille d'un bloc est **32 KB** block_size
— Le disque tourne a une vitesse de **7200 révolutions par minute**
— Le taux de transfert est **5 Mb/s** (N.B. b≠B) R_s
— Le seek time moyen est **9 ms** T_a

i. (0.5 points) Quel est le temps de transfert pour un bloc (en ms) ?
ii. (0.5 points) Quel est la latence rotationelle moyen (en ms) ?
iii. (1 point) Quel est le temps **total moyen** pour transférer un bloc (en ms) ?

Question exam (b)

Convert bloc size to bits

$1B = 8b$

$1KB = 2^{10}B = 1024B$

Donc $32KB = (32 \times 1024) \times 8 = 262144b$

Taux de transfert : $5Mb/s = 5 \times 2^{20} b$

⚠ $b \neq B$

$$T_t = \frac{262144}{5 \times 2^{20}} = 0.05s$$
$$= 50ms$$

ii) $1min = 60s$ donc $\frac{60}{7200}$ minutes per Rotation

Latence Rotationel moyenne $= \frac{60}{7200} \times \frac{1}{2}$

↳ en mins

60 000 ms dans une minute

Donc

Latence Rotationnel moyenne $= \frac{60}{7200} \times \frac{1}{2} \times 60\,000$

$= 4.17$ ms

iii)

temps total moyen = Avg seek time
+ Avg Rot latency
+ transfer time

$50 + 4.17 + 9 = 63.17$