Entrées sorties

Introduction

Un système sans entrée ou sortie n'est pas très utile. Car s'il n'y a pas d'entrée-sortie, ça veut dire, pas de réseau, pas d'écran, pas de clavier, etc.

Il est donc très important pour le système d'exploitation de contrôler les périphériques. Par exemple, le SE doit pouvoir gérer les interruptions (évènements envoyés par les périphériques) ainsi que traiter les erreurs ou autres évènements qui pourraient survenir (exemple, on débranche la clé USB pendant une action de lecture).

Le SE doit aussi fournir une interface vers ces périphériques de la manière la plus uniforme possible pour rendre les périphériques faciles d'accès et universel.

Côté matériel

Au niveau matériel, on distingue plusieurs éléments :

- Le **périphérique d'entrée-sortie**, par exemple clavier, souris, écran, etc. On pourra le caractériser par sa nature ainsi que sa vitesse de transmission. Ces périphériques vont faire de la **signalisation** (ou **interruption**) pour signaler au système que sa tâche est terminée.
- Le **contrôleur** qui est l'interface entre le périphérique et le système d'exploitation. Le contrôleur permet au SE de contrôler le périphérique.

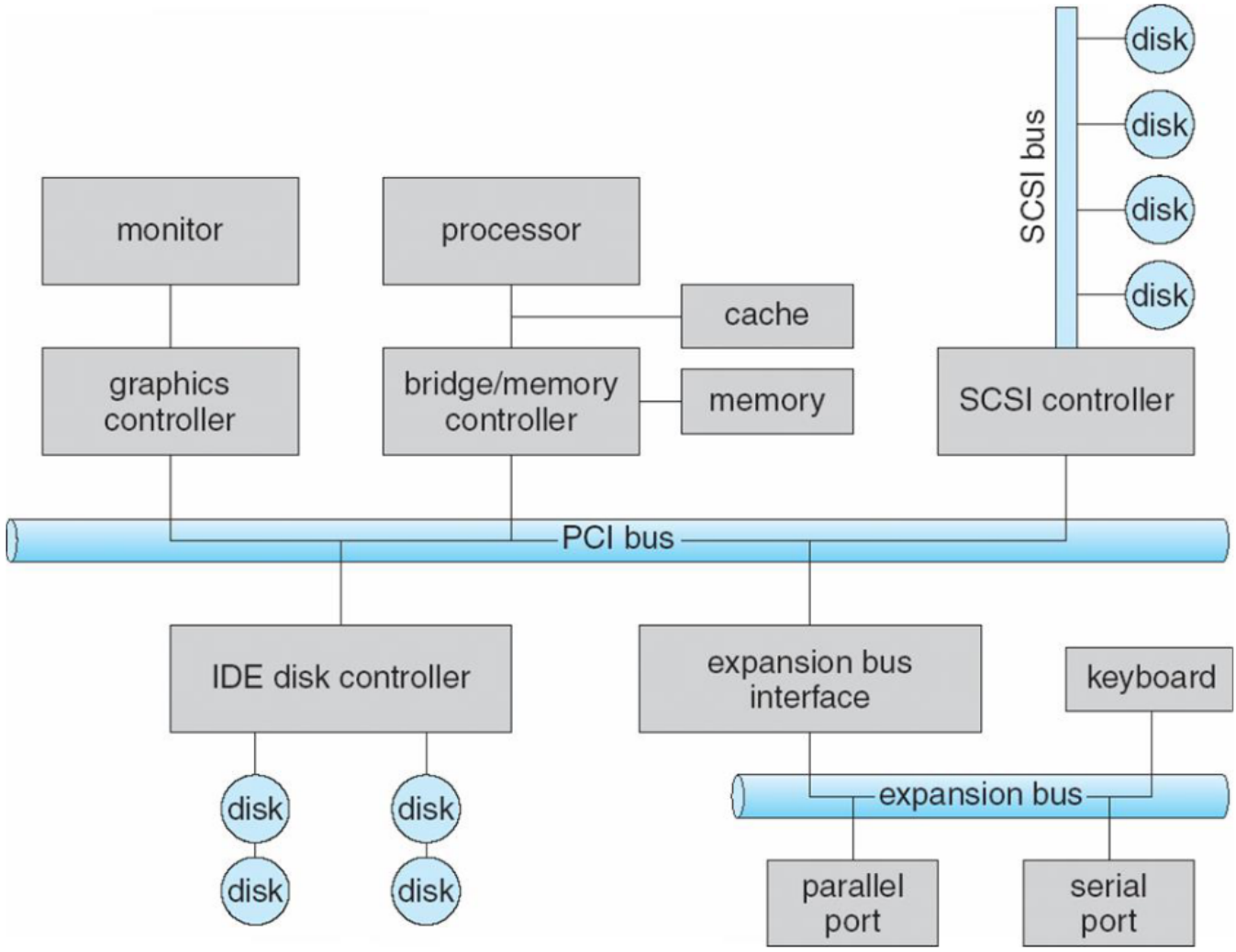
Périphériques E/S

Il existe plusieurs types de périphériques d'E/S, ici, on va parler de deux types principaux.

Il y a les périphériques **blocs** où l'information est stockée dans des blocs de taille fixes et adressables. Un exemple de tels périphériques est un disque dur.

Il y a aussi les périphériques **caractères** où l'information est un flux de caractère sans tenir compte d'une structure quelconque, elle est donc non adressable. Par exemple, la carte réseau, clavier, souris, etc.

Controlleur de périphérique



Le contrôleur est la partie électronique qui contrôle le matériel. Souvent, le périphérique dispose d'une partie mécanique et d'une partie électronique (exemple, un disque dur). Cependant, ce n'est pas toujours le cas, par exemple la carte graphique n'est pas attachée à l'écran.

Certaines cartes d'extensions sont même dissociées de la carte mère, c'est par exemple le cas avec les périphériques USB.

Contrôle/commandes

Le système d'exploitation **commande** le périphérique au moyen du contrôleur, il n'interagit jamais directement avec le périphérique lui-même.

Le contrôleur dispose fréquemment d'un microprocesseur hautement adapté au périphérique, de registres qui permettent de commander le contrôleur et de mémoire pour faire les échanges (par exemple pour faire des buffers).

Interruptions

Le matériel envoie des interruptions pour signaler que l'opération en cours est terminée. Le gestionnaire d'interruption du système d'exploitation va gérer l'interruption.

S'il n'y a qu'une seule interruption, le système gère l'interruption et choisit ensuite le processus suivant à démarrer.

S'il y a plusieurs interruptions, le système traite d'abord les interruptions urgentes en ignorant temporairement les autres.

Fonctionnement

Le CPU sauvegarde le contexte du processus en cours et lance une routine (une fonction indépendante du reste du système) adaptée à l'interruption reçue. La routine à lancer est renseignée dans le **vecteur des interruptions** où pour chaque interruption un pointeur vers la routine à exécuter est renseignée.

Dès que l'interruption est traitée, l'état du contrôleur du périphérique est modifié. Le gestionnaire des interruptions est alors prêt à traiter les interruptions suivantes.

Une fois toutes les interruptions traitées, le système d'exploitation choisit le processus suivant à lancer.

Dialogue via port d'E/S

Ce type de dialogue entre le système et le contrôleur consiste à échanger des données en écrivant ou lisant des données dans les registres du contrôleur.

Cette méthode n'est plus trop utilisée aujourd'hui.

Dialogue via E/S mappé en mémoire

Dans cette méthode, on va faire correspondre une zone mémoire (de la mémoire centrale) aux registres du contrôleur. Ainsi, lorsque l'on lit dans la zone mémoire, on lit et/ou écrit dans les registres du contrôleur.

L'avantage de cette méthode est que l'on a juste à utiliser des instructions habituelles d'accès à la mémoire pour contrôler les périphériques.

Grâce à cette méthode, il n'y a donc pas besoin d'instructions assembleur particulières et il n'y a pas non plus besoin de mettre en place des mécanismes de protection particuliers, car il suffit d'interdire ou d'autoriser l'accès à la zone mémoire.

Amélioration des performances via DMA

Si on utilise simplement le contrôleur d'un disque dur, lorsque l'on veut lire sur ce dernier, le contrôleur va lire un bloc à partir du périphérique, et le placer dans sa mémoire, ensuite, il va vérifier l'information, puis faire une interruption au système d'exploitation. Enfin, ce dernier va exécuter la routine adaptée et copier l'information dans sa mémoire centrale.

Le problème avec ce système est que le CPU est fort sollicité pour faire le transfert et ne peut donc rien faire d'autre. Une solution à cela est d'utiliser un contrôleur **DMA** (Direct Memory Access). C'est un microprocesseur spécialisé dans le transfert d'informations entre un contrôleur quelconque et la mémoire centrale.

Ce contrôleur DMA peut aussi bien se trouver sur certains périphériques que sur la carte mère.

Le contrôleur DMA possède, comme tout contrôleur, plusieurs registres. Les siens sont, l'adresse (pour indiquer où les informations doivent être écrites en mémoire), le compteur d'octets (pour indiquer quelle quantité doit y être écrite), le registre de contrôle (qui spécifie de quel périphérique il s'agit, si c'est une lecture ou une écriture et de l'unité du transfert (octet, mot, etc)).

Ainsi, en utilisant un contrôleur DMA, lorsque l'on veut lire sur un disque dur, le système d'exploitation va indiquer au DMA les données dont il a besoin, puis va demander au disque une lecture. Ensuite le contrôleur disque va lire et vérifier les blocs et c'est le DMA qui va se charger de transférer directement les informations dans la mémoire. Une fois le transfert terminé, le DMA fait une interruption au système d'exploitation.

De cette manière, le CPU peut faire d'autres choses durant le transfert

Vol de cycle

Attention toutefois, lorsque le DMA travaille, il se peut qu'il entre en conflit avec le processus, car les mémoires ne peuvent faire qu'un accès par cycle.

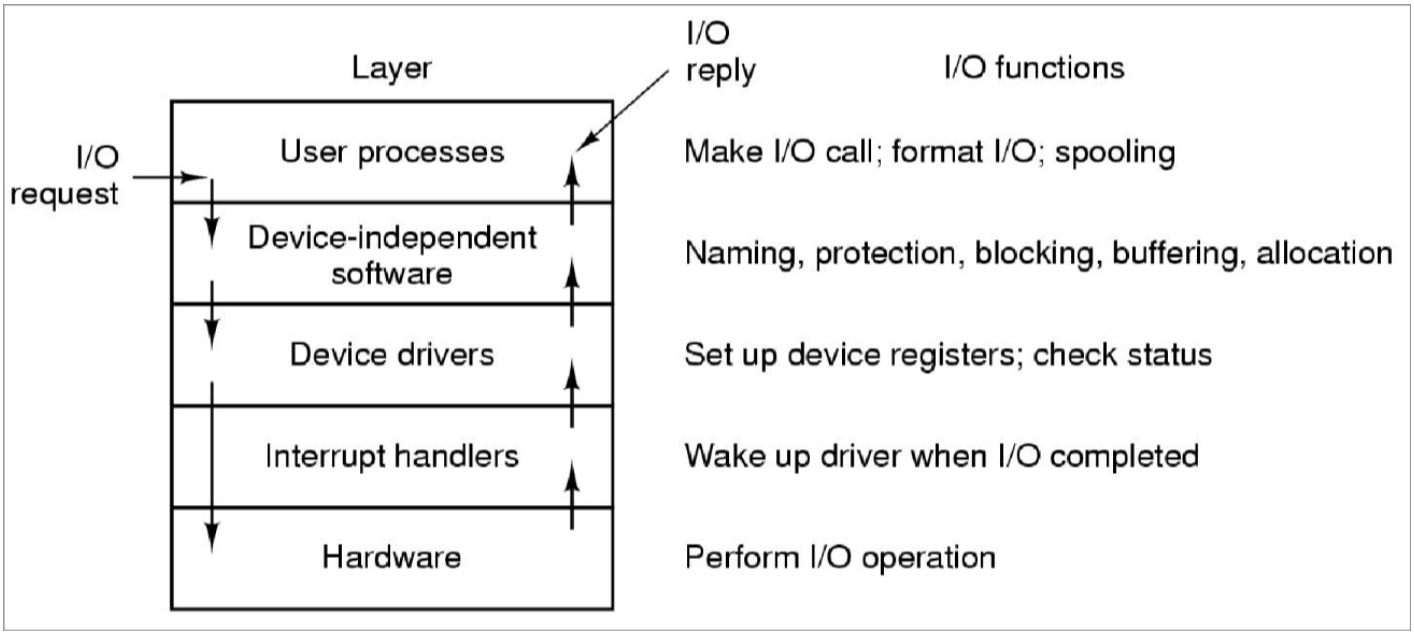
Le DMA ne pouvant pas attendre aussi longtemps que le processeur, parce qu'il risque de perdre des informations, il a donc priorité d'accès à la mémoire. On dit alors qu'il y a un **vol de cycle** du processeur.

Côté logiciel

La partie logicielle de l'entrée-sortie est une partie du système d'exploitation qui a pour but de fournir une interface vers les périphériques tout en assurant une indépendance par rapport au type de périphérique.

Par exemple, l'accès à une clé USB doit être, du point de vue des programmes, identique à l'accès sur un disque dur.

Cette partie du système d'exploitation est divisée en quatre couches, que l'on va adresser en partant du plus proche du matériel vers le plus haut niveau.



Le gestionnaire d'interruption

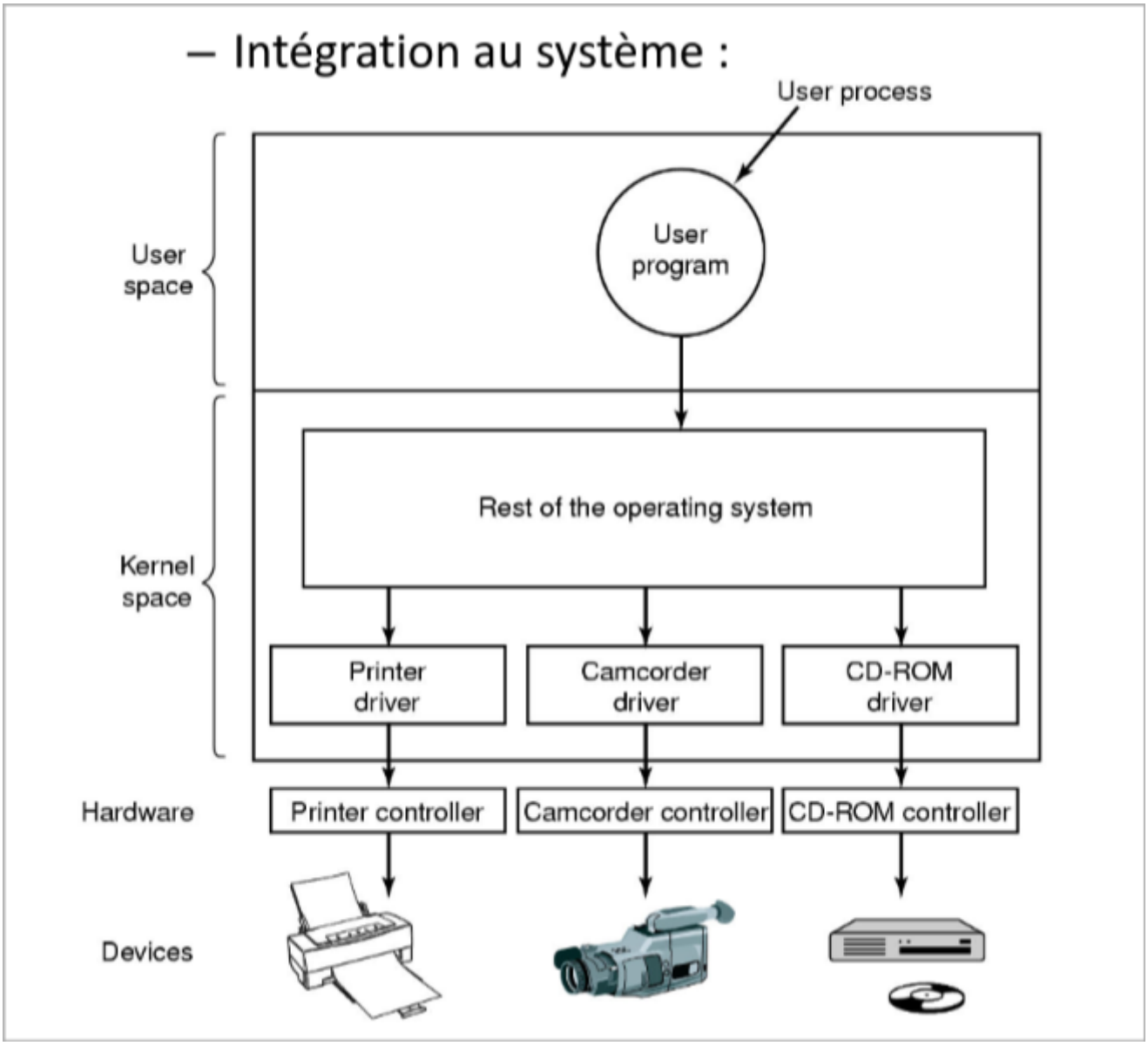
Le gestionnaire de périphérique (le pilote) qui lance l'opération d'entrée-sortie, va attendre une interruption.

Lorsque l'interruption survient, **gestionnaire d'interruptions** va avertir le gestionnaire de périphérique et celui-ci va terminer l'opération voulue.

Lorsqu'une interruption survient, le gestionnaire d'interruptions va sauvegarder le contexte du processus en cours, créer le contexte pour le traitement de l'interruption, se place en état "disponible" pour traiter les interruptions suivantes, exécuter la procédure de traitement (la routine dont on a parlé dans la section précédente).

Une fois les interruptions gérées, le scheduler choisit le processus à démarrer.

Gestionnaire de périphérique (pilote ou driver)



Le **gestionnaire de périphérique**, aussi appelé **pilote** ou **driver**, dépends de la nature du périphérique, ainsi un gestionnaire de disque fonctionne très différemment d'un gestionnaire de souris.

Le gestionnaire de périphérique, afin d'être utilisable, doit être chargé au cœur du système, dans le noyau, en mode protégé. C'est pour cela qu'un mauvais driver pour conduire à des plantages du système d'exploitation.

Mais en même temps, ces gestionnaires de périphériques, écrits par les fabricants, doivent pouvoir être inséré facilement dans le noyau (kernel) pour être utilisé.

Fonctionnement

Le système appelle des fonctions internes (open, read, write, initialize, etc) des pilotes afin de commander le périphérique. Les pilotes vont ensuite vérifier si les paramètres reçus sont corrects

et valide et vont traduire certains paramètres en données physiques.

Par exemple, un pilote de disque dur va traduire un numéro de bloc en cylindre, piste, secteur et tête.

Ensuite, le pilote vérifie si le périphérique est disponible (sinon il attend), il vérifie également l'état du périphérique et le commande.

Pendant le travail du périphérique, le pilote s'endort, car il ne peut plus rien faire. Une fois qu'il est réveillé par le gestionnaire d'interruption, il vérifie que tout s'est bien passé et transmet les informations.

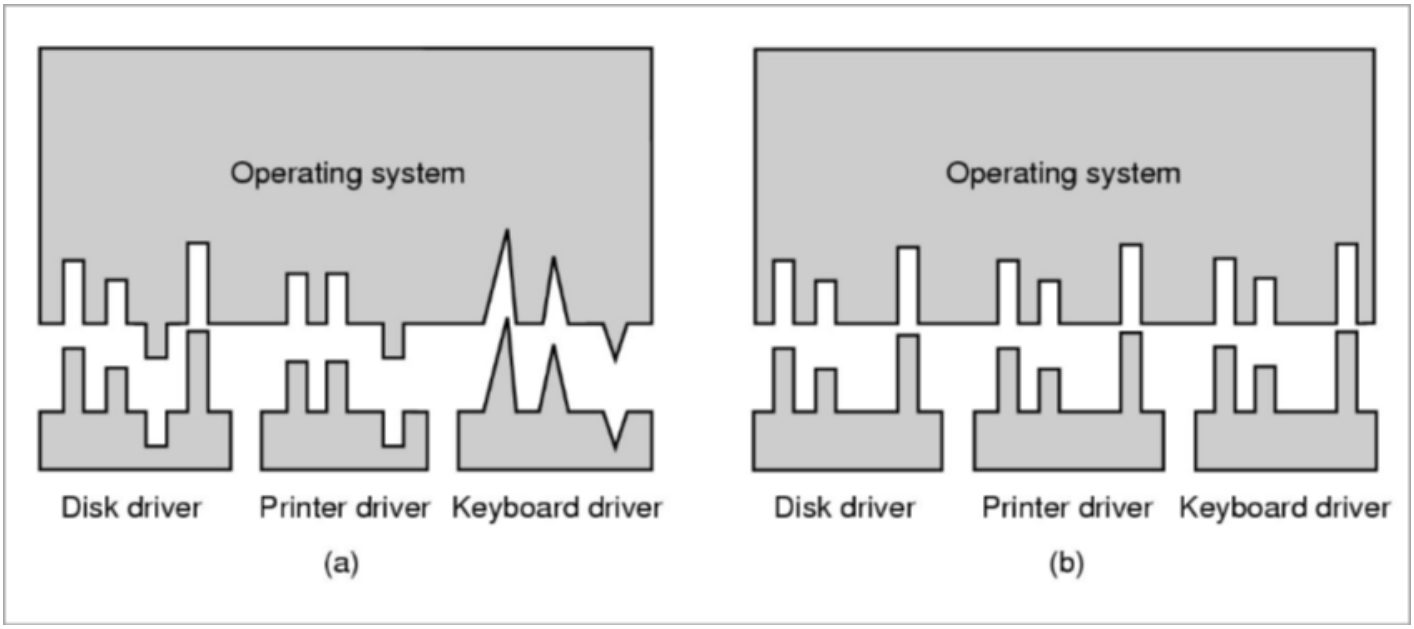
La couche logicielle indépendante

La couche de **logicielle indépendante** (aussi appelée **interface standard**)

Cette couche fournit une interface uniforme pour les drivers (la fameuse API dont on a parlé plus tôt), ainsi que du buffering, de la gestion d'erreur, de l'allocation et libération des périphériques.

Interface standard

Grâce à cette couche, on s'assure que les drivers sont appelés de manière uniforme.



A gauche, un système sans interface standard, à droite un système avec une interface standard

Uniformisation des noms

La désignation d'un périphérique doit être non ambiguë, sous Linux le répertoire `/dev` contient des entrées pour chaque périphérique. Un périphérique est donc traité au même titre qu'un fichier, il ne peut donc pas en avoir deux avec le même nom.

Par exemple, un disque dur sera appelé `/dev/sda`, si on met un autre disque dur, celui-ci sera appelé `/dev/sdb`.

Buffering

Afin d'améliorer les performances du système et d'éviter de faire beaucoup d'accès inutilement, on garde des buffers (tampons) dans le système d'exploitation et dans les contrôleurs. L'objectif étant de placer en mémoire les informations qui sont souvent accédées.

Gestion d'erreurs

C'est aussi à cette couche que revient la gestion des erreurs. L'utilisation de périphériques induit un certain nombre d'erreurs (temporaire ou permanentes).

Il y a deux types d'erreurs, les **erreurs de programmation** qui surviennent lorsque l'opération demandée est impossible (exemple, écriture sur un périphérique d'entrée), dans quel cas on rapporte l'erreur et on s'arrête.

Et les **erreurs d'entrées sorties** qui surviennent lorsqu'une opération se termine de façon anormale (par exemple, on déconnecte le disque, ou le périphérique est défectueux). Dans ce cas, le driver décide de soit tenter à nouveau l'opération ou alors de rapporter l'erreur.

Allocations et libérations des périphériques

Certains périphériques ne sont pas partageables, par exemple avec une imprimante, il est impossible d'imprimer plusieurs choses en même temps.

Il revient alors au système d'exploitation de vérifier si le périphérique est libre et s'il peut être alloué au processus.

La fonction `open` permet à un processus d'informer le système d'exploitation qu'il souhaite utiliser un périphérique, le système vérifie alors sa disponibilité.

Couche d'entrée-sortie applicative

Cette partie de la gestion de l'entrée sortie est faite au niveau de l'application, par exemple via les libraires systèmes liées aux programmes.

C'est là que l'on va par exemple trouver les appels systèmes de `stdio` tel qu'open, printf, read, write, etc.

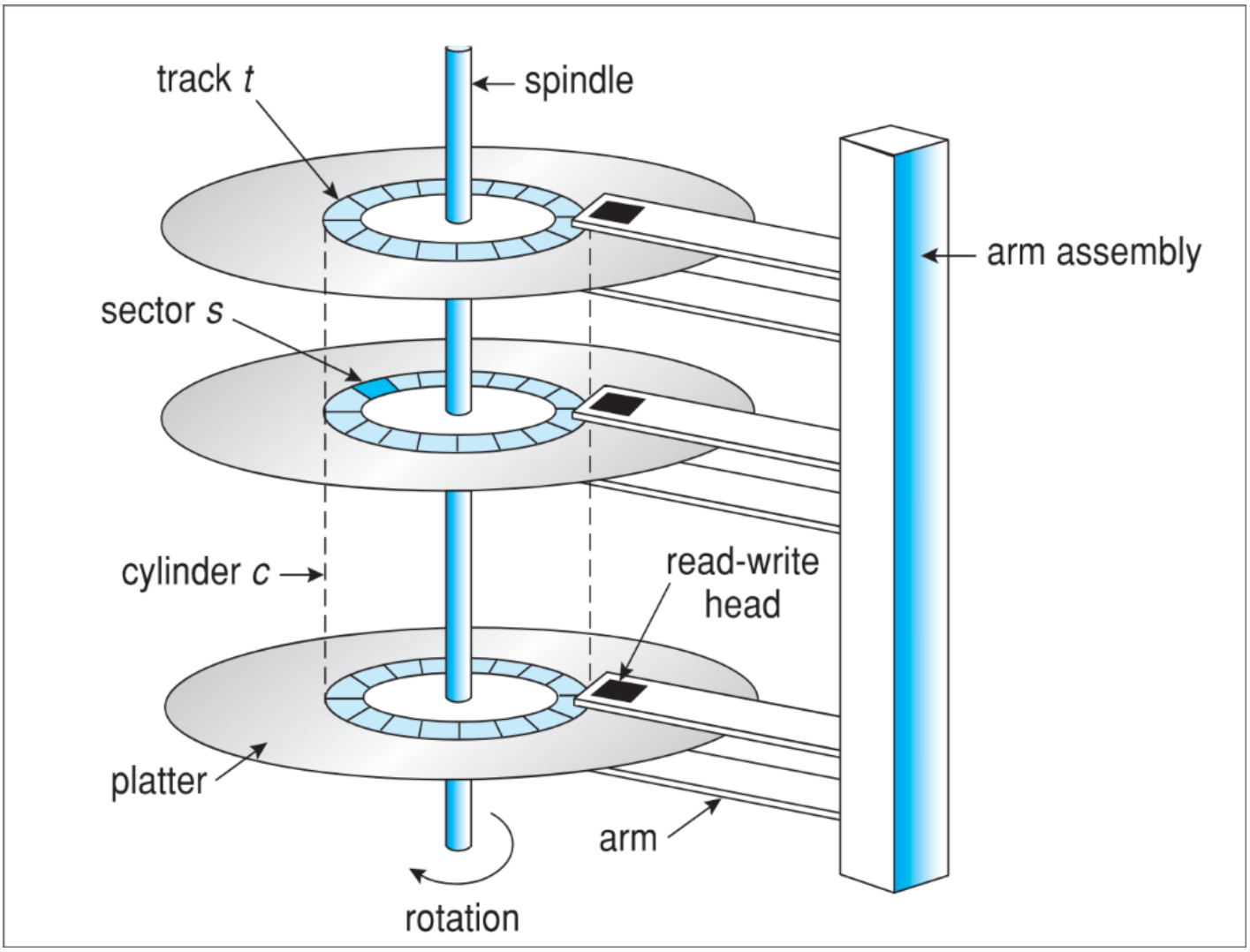
Cette couche va également fournir un système de **spooling** qui permet de créer une fille d'attente pour l'accès aux périphériques non partageables. Par exemple via une liste de jobs d'impression pour une imprimante.

Disque

Matériel

Un disque magnétique est composé de plusieurs disques physiques appelés les **plateaux**.

Le disque est découpé en cylindres, pistes et secteurs. Il y a autant de pistes que de positions différentes de la tête de lecture.



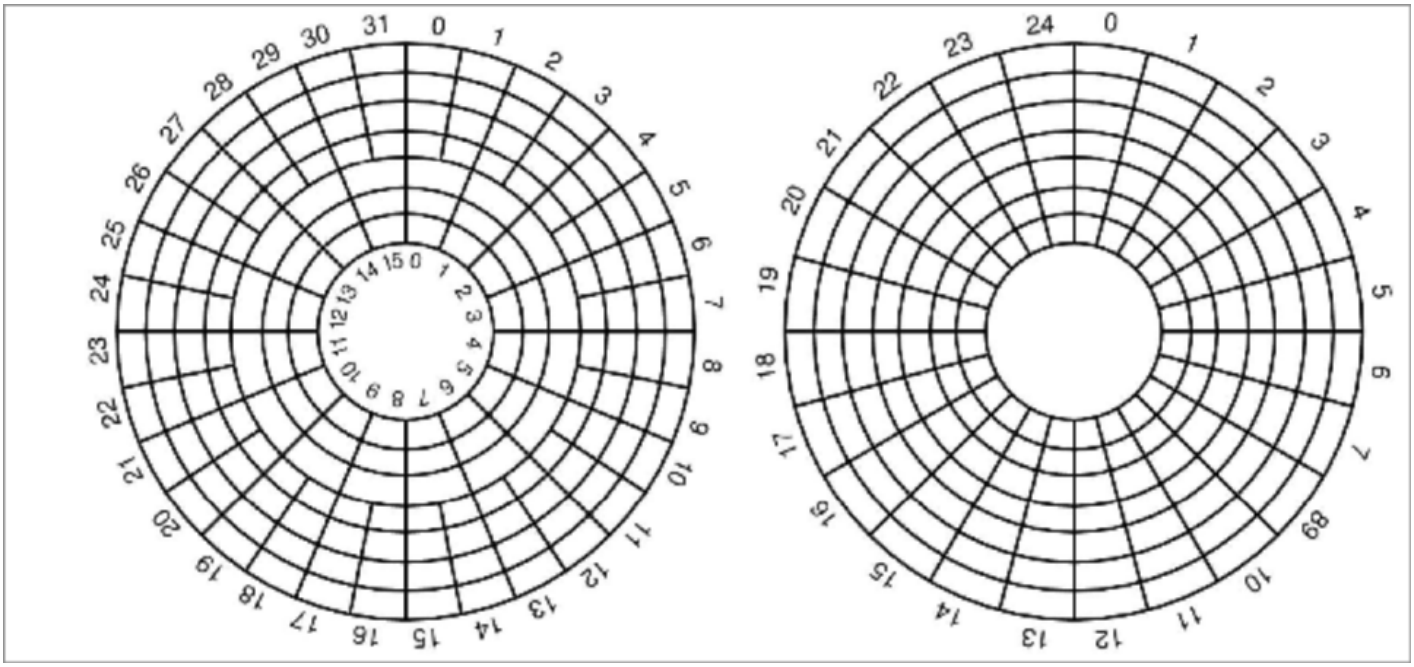
Un **cylindre**, c'est l'ensemble de pistes à une position donnée de la tête de lecture.

Une **piste** correspond à un cercle sur un plateau à une certaine position de la tête de lecture. Chaque piste est elle-même découpée en secteur.

Les **secteurs** sont des blocs de tailles fixes qui compose chaque piste.

> Pour en savoir plus sur le fonctionnement des disques dur magnétiques, vous pouvez consulter cette page Wikibooks.

Les disques dur ont une interface qui permet au contrôleur d'utiliser des commandes de haut niveau. La géométrie du disque dur n'est pas nécessairement celle qui est annoncée, car le nombre de secteurs par piste n'est pas constant.



A gauche, véritable géométrie du disque. À droite, géométrie du disque simplifie pour rendre les accès plus simple

RAID

RAID est une technologie qui consiste à combiner plusieurs disques afin d'améliorer les performances (si on a plusieurs disques, on peut par exemple écrire sur plusieurs disques en même temps) et/ou la fiabilité (si on a plusieurs disques, on peut par exemple dupliquer les informations sur tous les disques, comme ça si un disque tombe en panne, on peut restaurer les données).

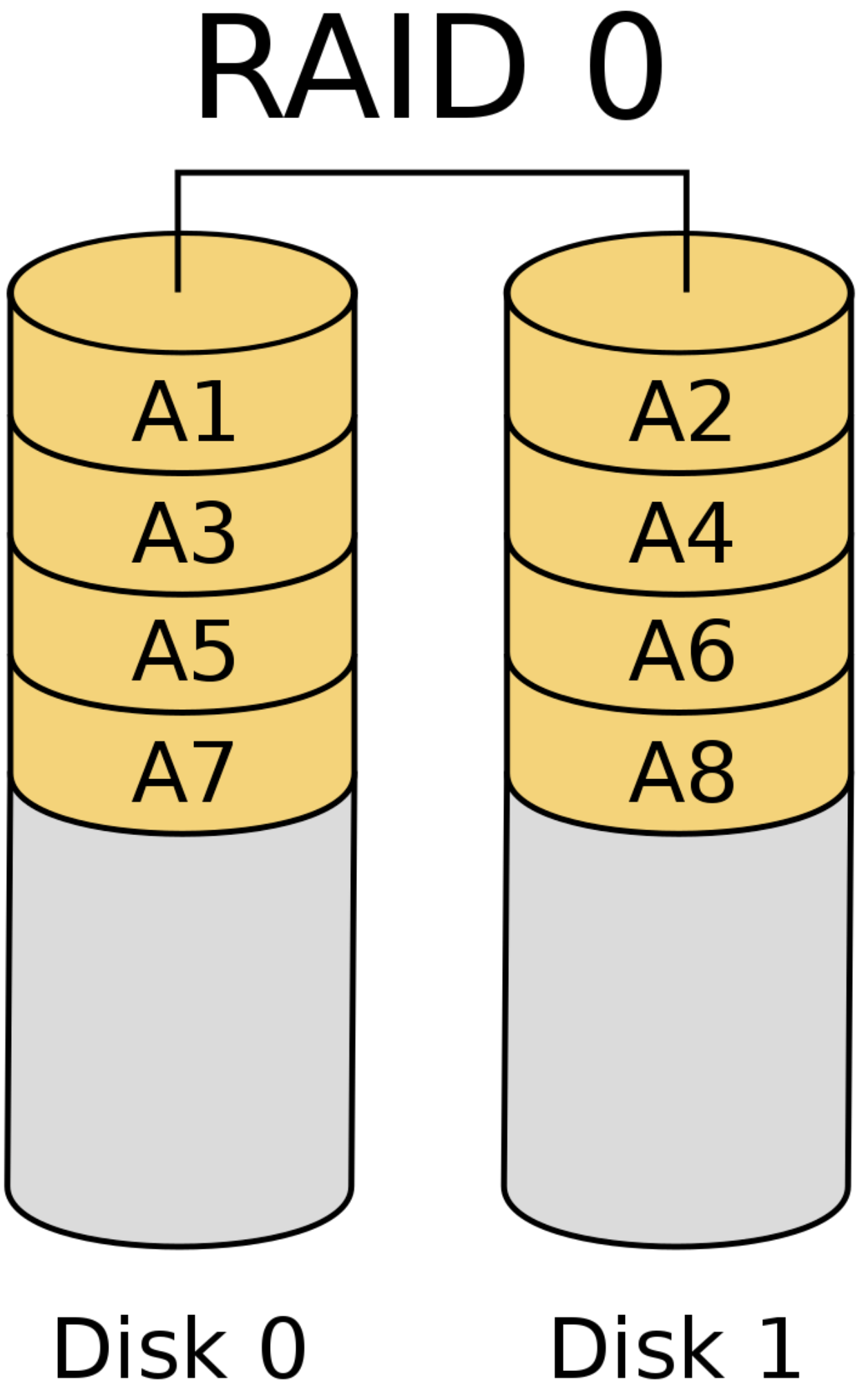
Le **mirroring** consiste à dupliquer toutes les informations sur un second. Ainsi, lors d'un accès en écriture, le contrôleur effectue la modification sur les deux disques simultanément et indique qu'il a terminé lorsque l'opération est achevée sur les deux disques.

Le **stripping** consiste à répartir l'information sur plusieurs disques. De cette manière, si on a huit disques, on peut distribuer les informations sur chaque disque de façon à écrire huit fois plus rapidement qu'avec un seul.

Il existe plusieurs niveaux de RAID, certains utilisent du mirroring, d'autre du stripping et d'autres les deux de manière à favoriser les performances et la fiabilité.

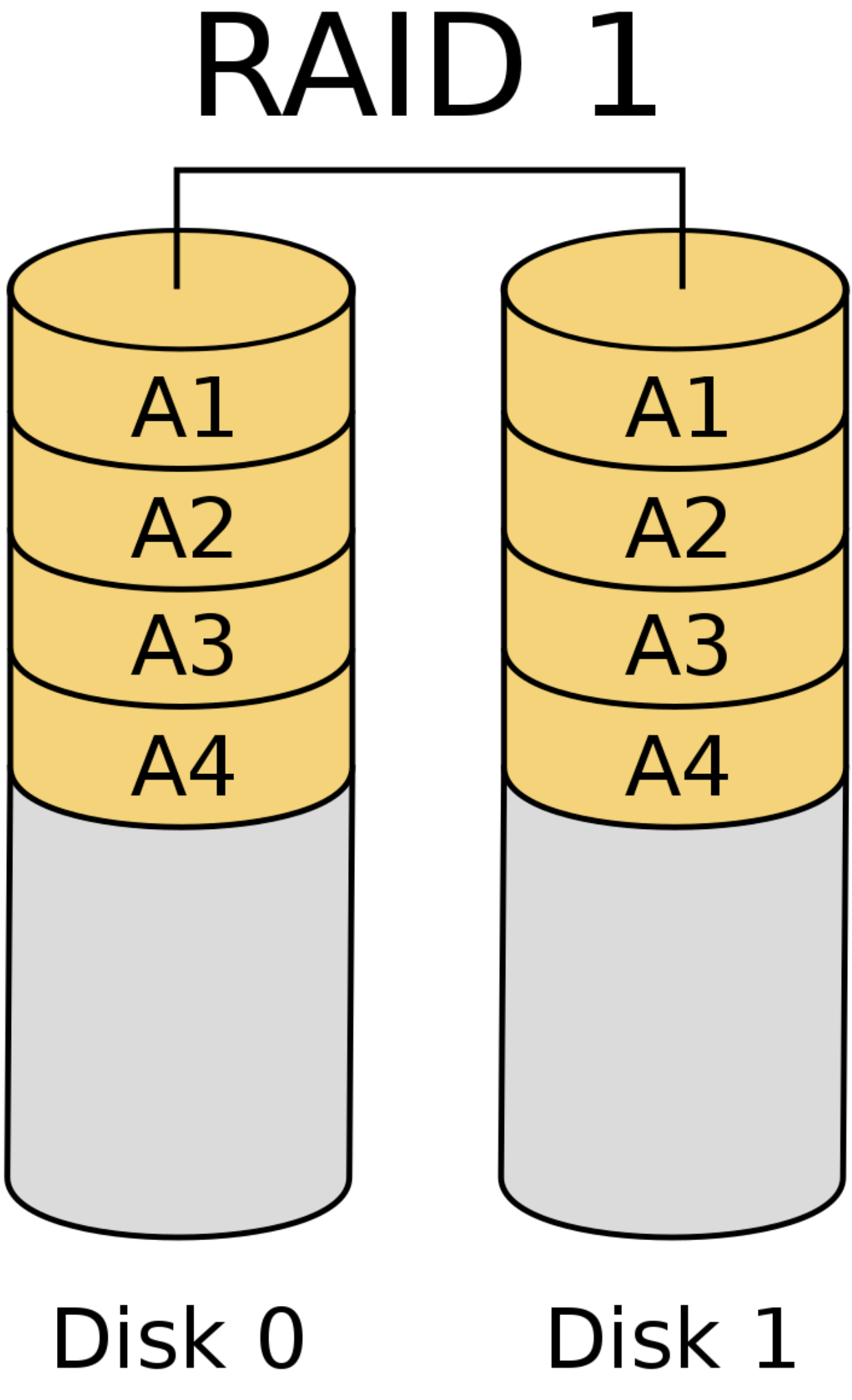
> Si vous souhaitez en savoir plus sur les niveaux de RAID, vous pouvez consulter cet article Wikipédia et celui-ci.

RAID 0


RAID 0
A1
A3
A5
A7
A2
A4
A6
A8
Disk 0
Disk 1

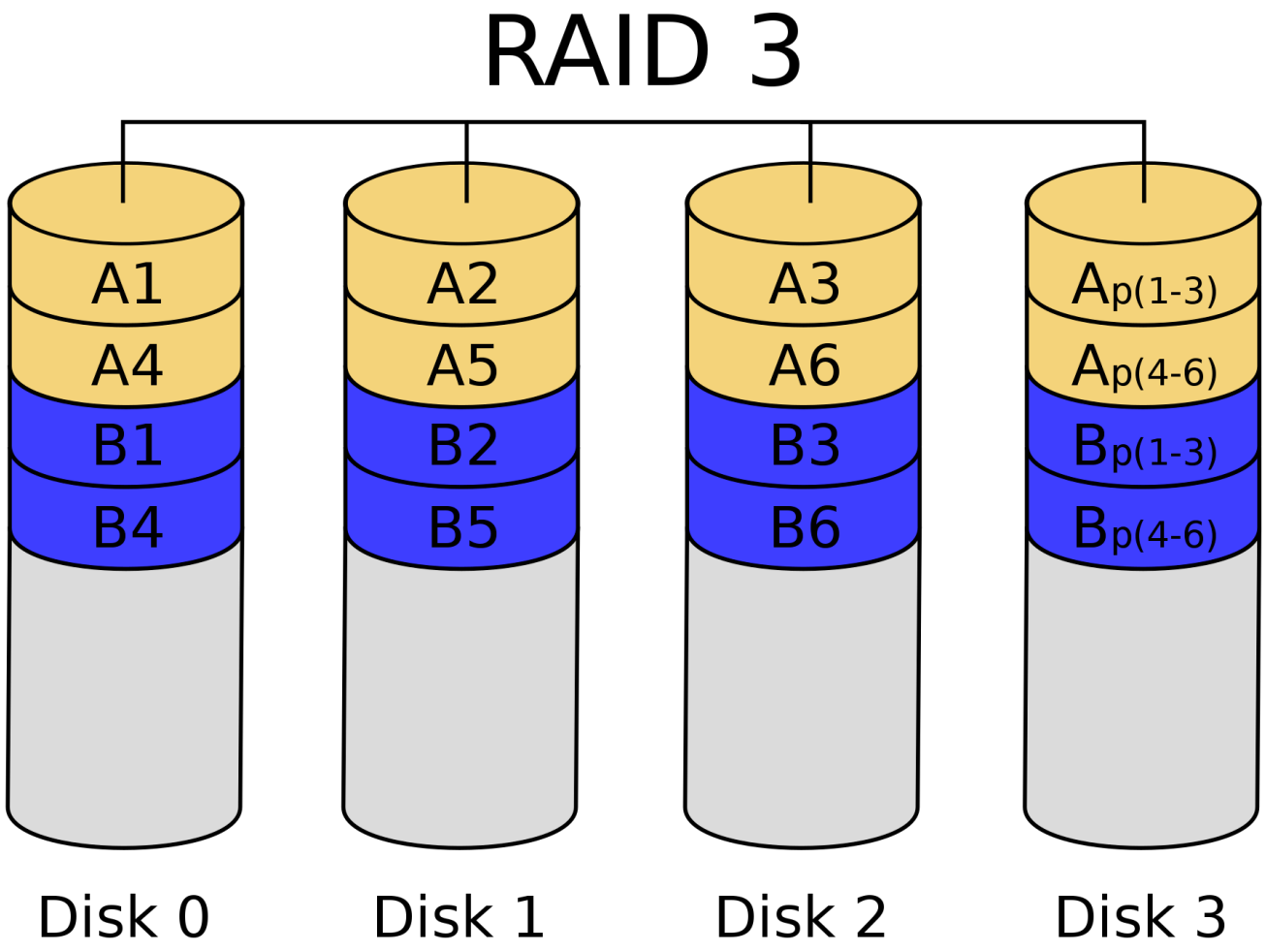
Le RAID 0 fait uniquement du stripping. Il va donc construire un grand espace disque en combinant les disques. Ainsi, les données sont réparties entre des différents disques de manière à améliorer les performances.

RAID 1


RAID 1
A1
A2
A3
A4
A1
A2
A3
A4
Disk 0
Disk 1

Le RAID 1 fait exclusivement du mirroring, il faut donc doubler le nombre de disques. Cela améliore beaucoup la fiabilité de l'information, mais il est assez couteux.

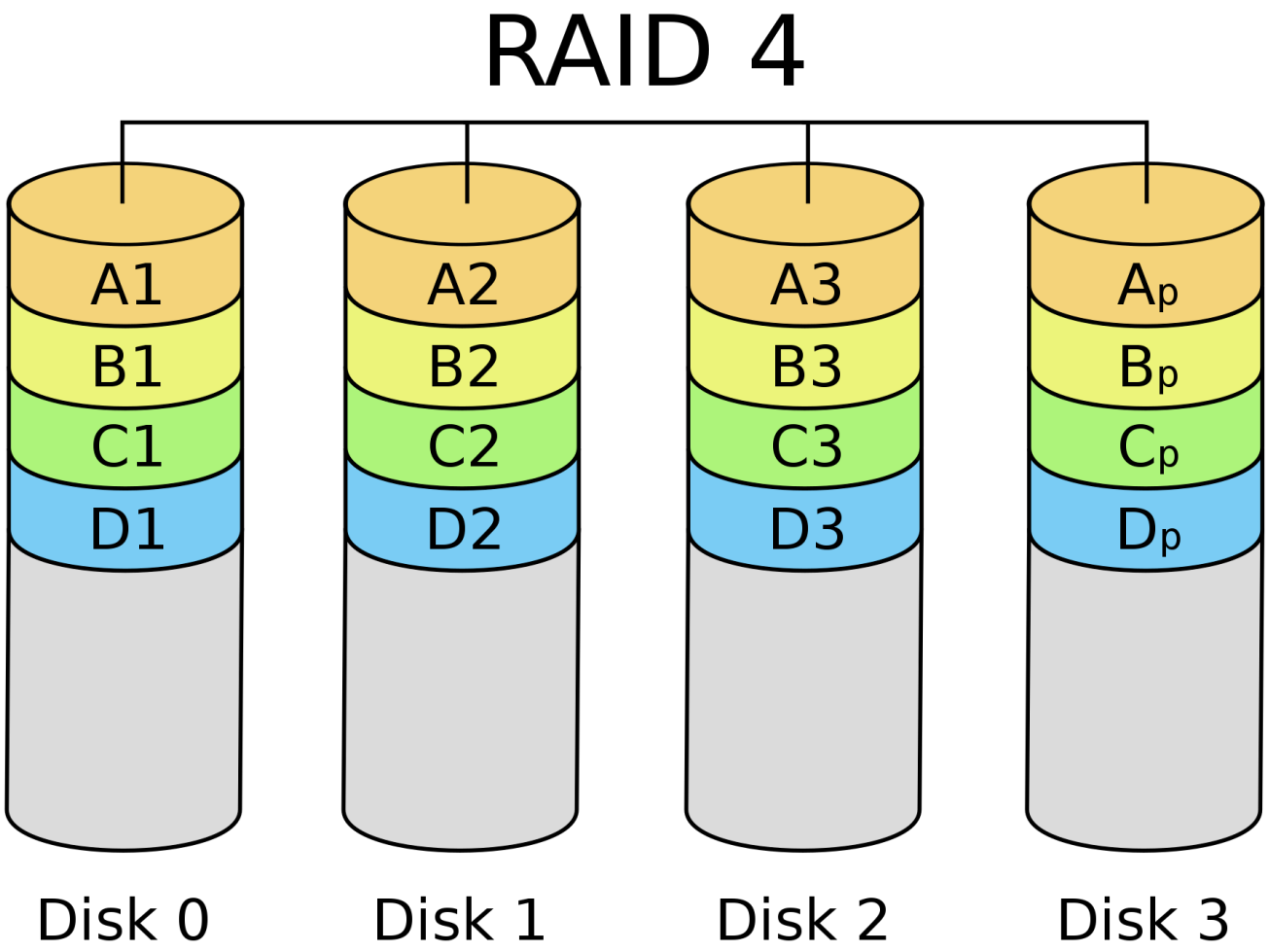
RAID 3



Le RAID 3 fait du stripping au niveau des octets. Un bit de parité est gardé sur un disque séparé et les octets vont être réparti sur les différents disques.

L'avantage du RAID 3 est que si on perd un disque, on peut reconstruire l'information à la volée en utilisant le disque de parité.

RAID 4

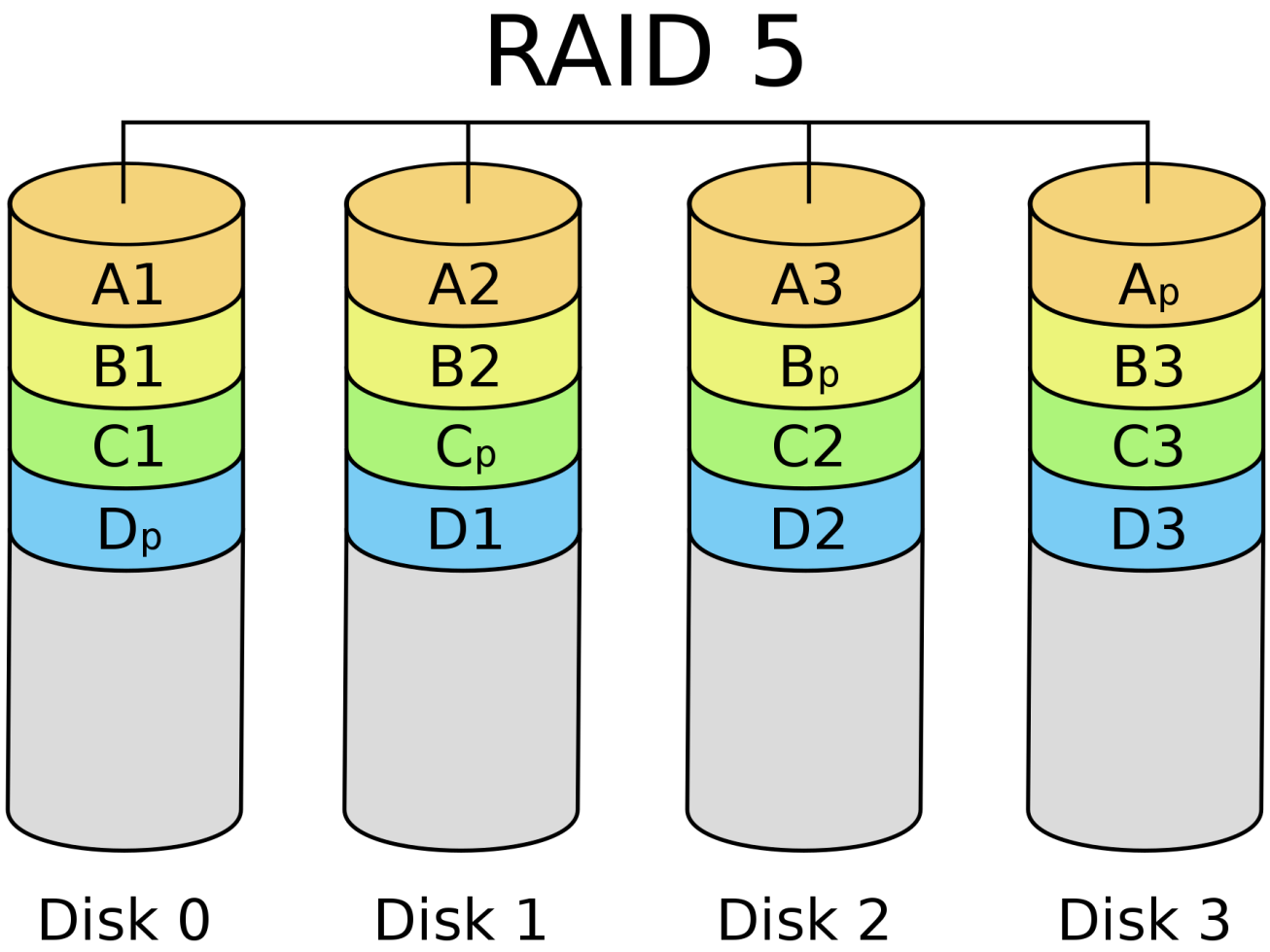


Le RAID 4 fait du stripping au niveau des blocs. Les blocs de parités sont gardés sur un disque séparé.

Un avantage du RAID 4 est que la lecture d'un bloc ne nécessite l'accès qu'à un seul disque (contrairement au RAID 3), on peut donc également satisfaire la lecture de plusieurs blocs simultanément si ceux-ci ne sont pas localisés sur le même disque.

Il y a cependant un problème à l'écriture, étant donné que tous les bits de parité sont sur le disque de parité, on ne peut pas écrire des blocs en parallèle, car on ne peut avoir qu'un seul accès au disque de parité à la fois.

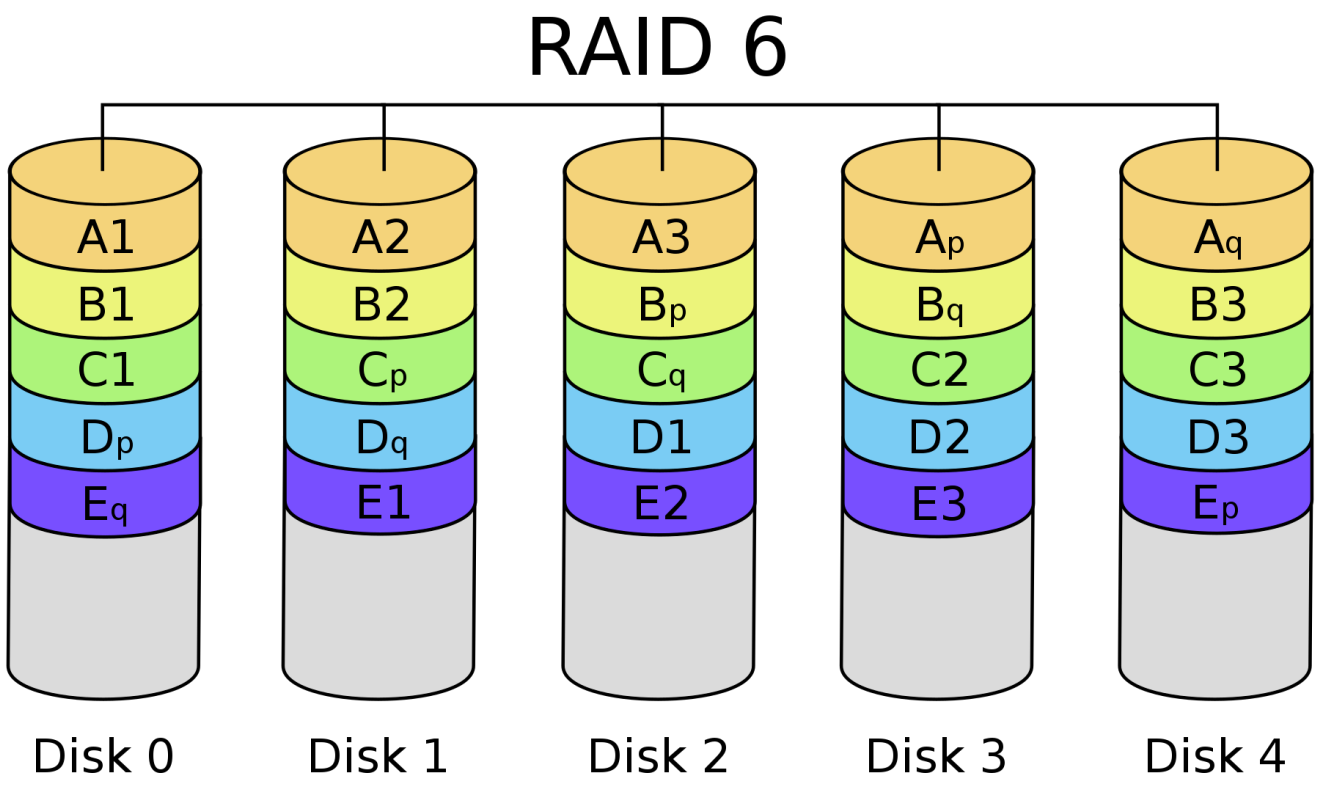
RAID 5



Le RAID 5 est une amélioration du RAID 4 qui va distribuer les informations sur la parité, de cette manière chaque disque contient des blocs de données et de parité. À chaque bloc de donnée correspond un bloc de parité stocké sur un disque.

Lors d'une écriture, il y a alors moins de problème, car plusieurs requêtes d'écriture peuvent être faites en même temps parce que tout dépend de la position de la donnée à écrire et de la localisation de l'information de parité.

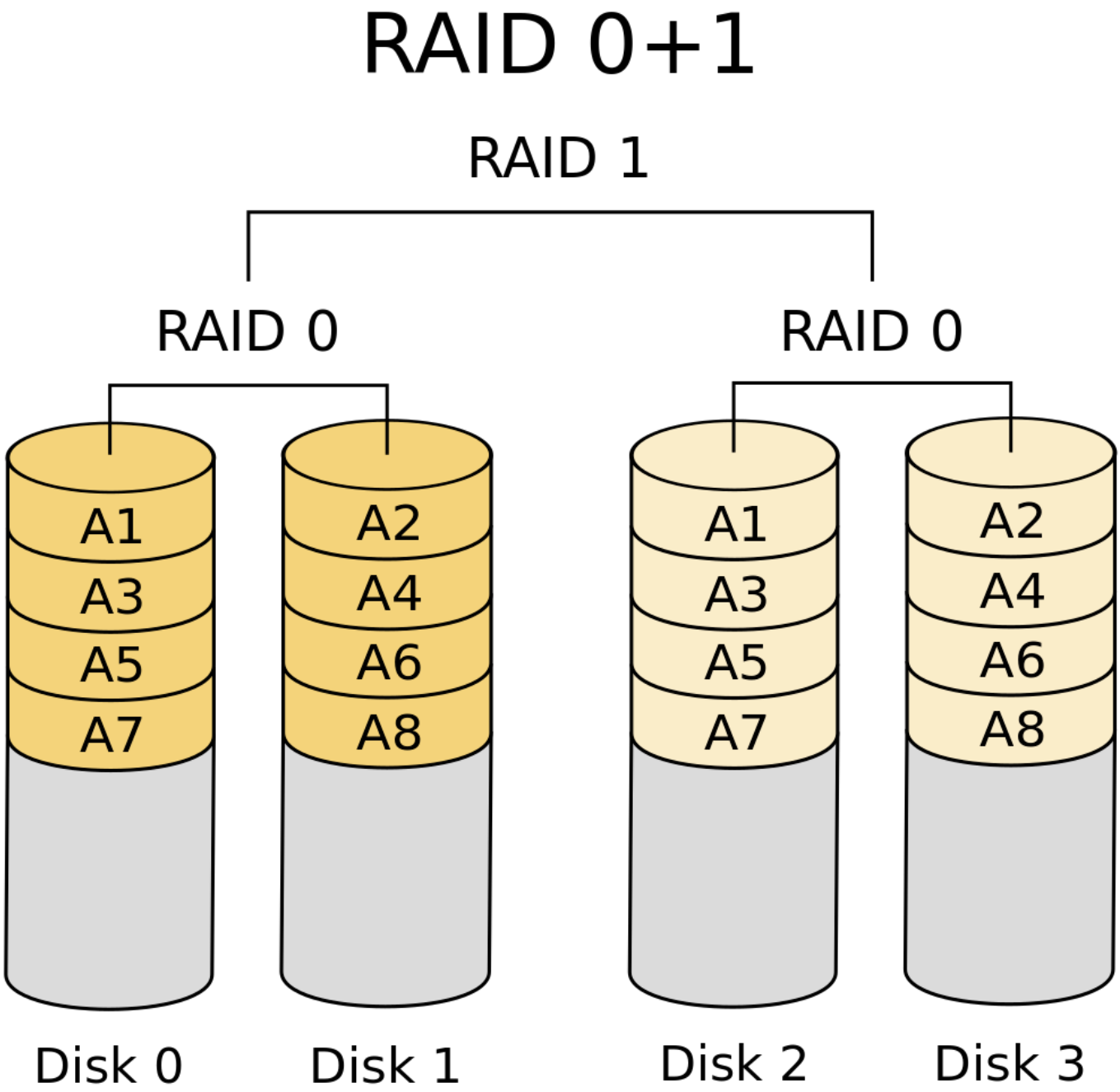
RAID 6



Le RAID 6 permet de protéger de la perte de deux disques dur, elle utilise un algorithme de parité plus complexe et nécessite un CPU plus important pour le contrôleur.

Plus il y a de données, plus le RAID 6 est préférable au RAID 5.

RAID 0+1



Ce niveau utilise RAID 0 pour le stripping et RAID 1 pour le mirroring de tous les disques en stripping. Il est très utile si l'efficacité et la protection sont importantes, cependant il est assez cher.

Lequel choisir ?

Généralement, c'est le RAID 5 qui est choisi s'il y a plusieurs disques ou alors le RAID 1 s'il n'y a que deux disques. Sauf dans des cas où il y a des demandes plus particulières au niveau de la performance et/ou de la fiabilité du système.

Options particulières

Il y a des options particulières sur les différents systèmes, notamment la possibilité de faire du **hot-swap** ou d'avoir des disques **spare**.

Le hot-swap consiste à pouvoir retirer des disques pendant que ceux-ci sont connectés (du moment qu'il n'est pas actuellement utilisé).

Le spare consiste à avoir un disque présent inutilisé, mais prêt à l'emploi. Ainsi, si un disque dur tombe en panne, le disque spare peut alors être utilisé.

Autres considérations

Il est important de bien considérer que le système ait un débit important pour avoir les meilleures performances.

Mais surtout, il est important d'avoir une certaine diversité entre les disques dur. Il ne faut donc pas prendre tous les disques dur de la même marque au même moment, de la même génération. Car alors, il y a de grandes chances que les disques se fatiguent de la même manièrent et tombe en panne plus ou moins en même temps.

Formatage

Avant qu'un disque ne puisse être utilisé, il doit être formaté (**formatage de bas niveau**), cette opération consiste à écrire la géométrie du disque (secteur, cylindre, piste, etc).

Cette opération doit être réalisée par le fabricant. De cette manière, chaque secteur contient un préambule (informations décrivant le secteur tel que le numéro ou le cylindre), les données et un code ECC pour la gestion des erreurs.

Une fois l'espace créé, il est possible de mettre en place des partitions, c'est la première structure logique du disque.

Le disque est séparé en plusieurs partitions, et chaque partition est vue par le système comme un espace séparé. Par exemple, un système Linux va voir les partitions `/dev/sda1` et `/dev/sda2` comme deux espaces complètements différents.

Le secteur zéro du disque contient la partition control block et le boot control block qui mentionnent comment le disque et découpé et comment le système peut démarrer.

Le **formatage de bas-niveau** est une opération de formatage réalisée par le système d'exploitation, elle consiste à donner une structure à la partition (un format de système de fichier).

Scheduling

Le disk arm scheduling est une opération faite par le système d'exploitation pour planifier les requêtes d'entrées-sorties.

En effet, plusieurs requêtes d'E/S peuvent arriver depuis plusieurs processus pendant que le disque est toujours en train de traiter une requête, donc les autres requêtes doivent attendre.

L'importance du scheduling (avec les HDD) est de minimiser le **seek-time**, c'est-à-dire minimiser les mouvements de la tête de lecture afin de pouvoir lire les informations plus vite.

Nous allons donc, voire plusieurs algorithmes de scheduling différents,

- Le **FCFS** (First Come First Served), soit les requêtes sont servies dans l'ordre dans lesquelles elles arrivent. Cet algorithme a l'avantage d'être équitable, mais a de mauvaises performances, car il ne fait rien pour améliorer le temps de réponse du disque.
- Le **SSTF** (Shortest Seek-Time First), autrement dit de servir toujours la requête la plus proche de la position courante. Cet algorithme a l'avantage de faire diminuer le temps de réponse du disque. Mais il a le désavantage de devoir calculer les positions et risque également de causer de la famine parmi les requêtes les plus éloignées.
- Le **SCAN** (ou algorithme de l'ascenseur), où le bras de lecture va toujours dans la même direction jusqu'à arriver à la fin du disque avant de retourner dans le sens inverse, aussi, il n'y a plus de famine, car quoi qu'il arrive, on avance.
- Le **C-SCAN** (SCAN circulaire), fonctionne comme le SCAN sauf qu'au lieu de partir dans l'autre sens lorsque la fin du disque est atteinte, elle retourne à l'autre extrémité du disque.
- Le **LOOK** est une variante du SCAN qui a la place de continuer jusqu'aux extrémités du disque, va s'arrêter lorsqu'il n'y a plus de requête dans la direction.
- Le **C-LOOK** (LOOK circuliare), comme le LOOK, excepté qu'à la place d'aller dans la direction inverse, il va directement à la première requête.

> Vous pouvez découvrir d'autres algorithmes ou avoir plus d'explication et d'illustrations sur les algorithmes décrit ici en allant lire cette page.

Choix de l'algorithme

Le SSTF est courant et fournit un bon remplacement à FCFS.

Si le disque est très chargé, C-SCAN et C-LOOK sont intéressants.

Le plus souvent on va trouver du SSTF ou une variante de LOOK.

Aujourd'hui, le scheduling est réalisé à plusieurs niveaux, par exemple au niveau du système d'exploitation, mais également au niveau des contrôleurs.

Gestion des erreurs

Il peut y avoir beaucoup d'erreurs différentes sur un disque.

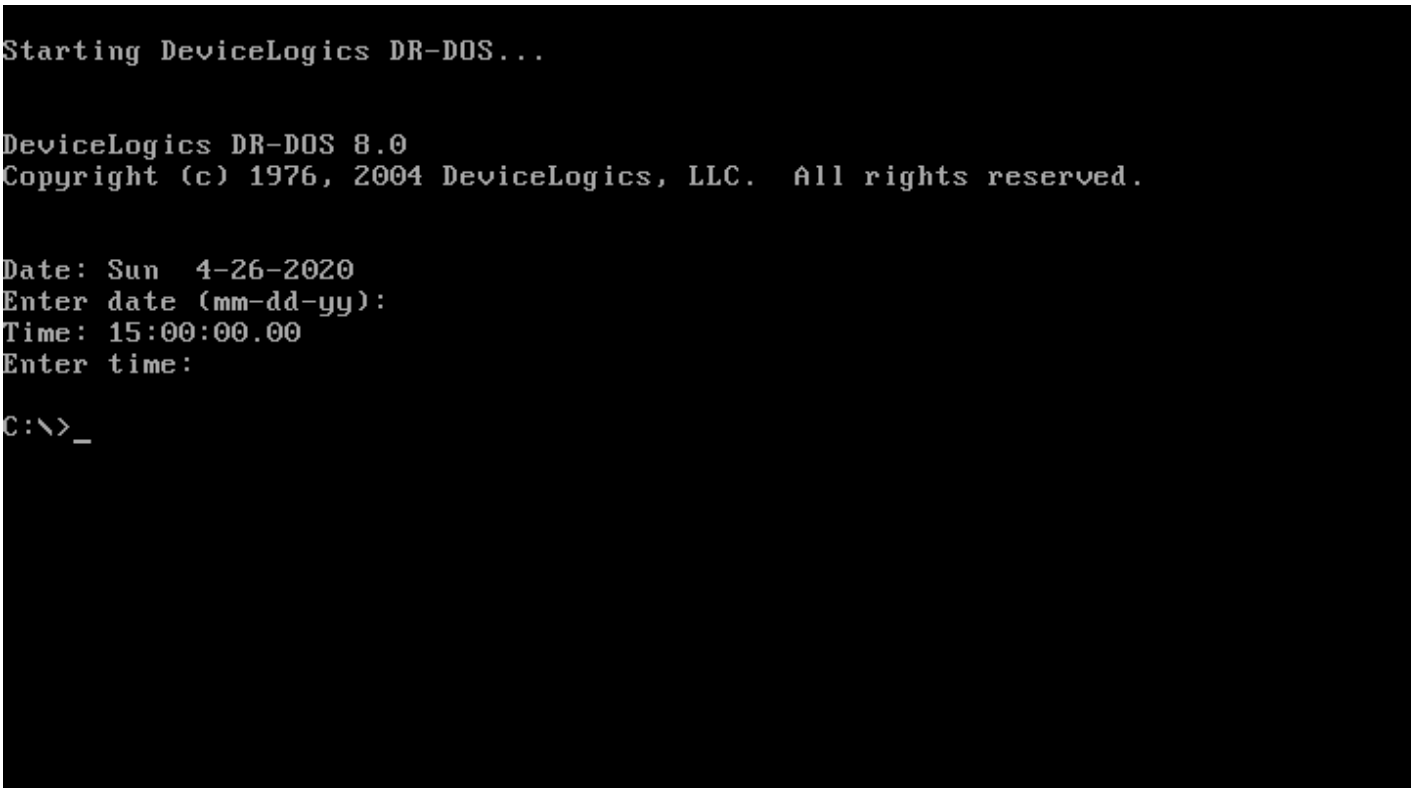
Par exemple, des secteurs défectueux peuvent survenir lors de la construction du disque dur, notamment lors du formatage de bas niveau, ou même survenir durant l'utilisation du disque.

C'est pourquoi le système d'exploitation doit pouvoir se souvenir des blocs défectueux afin de ne plus les utiliser dans le futur.

Horloge

L'horloge est un périphérique spécial et essentiel. Il sert simplement à déterminer la date et l'heure.

C'est une fonction essentielle, car c'est nécessaire pour maintenir des statistiques, calculer le quantum de temps des processus, etc.



> Fun fact : la Raspberry Pi n'a pas d'horloge, la date et l'heure est synchronisés via le réseau au démarrage. Et s'il n'y a pas de connexion, l'horloge va commencer à compter à partir de l'heure à laquelle il s'est arrêté.
>
> Les systèmes MS-DOS n'avaient pas non plus d'horloge, il fallait donc indiquer la date et l'heure manuellement au démarrage.

L'horloge fonctionne avec un quartz, un compteur et un registre. Le quartz, lorsqu'il est sous tension, génère un signal périodique très précis, ce signal peut ensuite être utilisé pour la synchronisation des composants. Le signal est ensuite fourni au compteur qui va décompter jusqu'à arriver à zéro.

Une fois arrivé à zéro, une interruption est émise vers le CPU et le gestionnaire d'horloge prend la main.

Ensuite, ici, il y a deux modes de fonctionnement. Il y a le mode **one-shot**, le système envoie l'interruption et attends une réaction. Sinon, il y a le mode **square wave** qui, une fois que le compteur arrive à 0, il recommence. On parle ici de **ticks d'horloge**.

Le système d'exploitation va ensuite maintenir l'heure en comptant le nombre de secondes depuis une date et heure de référence (par exemple, sous Linux/UNIX, on compte le nombre de secondes depuis le 1ier janvier 1970 00:00 UTC).

Pour synchroniser l'heure, il suffit donc de savoir le nombre de secondes depuis un point de référence.

Le logiciel d'horloge a ensuite pour but de maintenir l'heure, vérifier qu'un processus ne dépasse pas son quantum, compter l'utilisation du CPU, gérer l'appel système alarm, maintenir différentes statistiques, etc.

Par exemple, pour vérifier qu'un processus ne dépasse pas son quantum, le compteur d'horloge est initialisé par le scheduler en fonction du quantum. À chaque interruption, il est diminué et une fois arrivé à zéro le processus est remplacé.

Pour pouvoir gérer plusieurs évènements temporels en même temps, le système met en place une **file d'évènements**, qui agit comme une sorte d'agenda pour planifier des évènements.