

UNIVERSITE MOHAMMED PREMIER
ECOLE SUPERIEURE DE TECHNOLOGIE
Département Génie Informatique
OUJDA

SYSTÈME D'EXPLOITATION LINUX

Omar MOUSSAOUI

Omar.moussaoui78@gmail.com

2017 — 2018

Objectifs du cours

- **Comprendre la notion de système d'exploitation**
 - ▣ **Son utilité**
 - ▣ **Ses fonctionnalités**
- **Être capable d'utiliser les fonctionnalités de base d'un système Linux**
 - ▣ **Commandes principales**
 - ▣ **Langage de scripts**

Plan

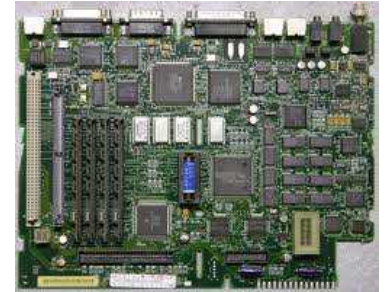
- **Introduction**
- **Notion de système de gestion de fichier**
- **Interpréteur de commandes**
- **Implantation des systèmes de gestion de fichiers**
- **Notion de processus**
- **Systèmes de gestion de la mémoire**
- **Les scripts shell**

Introduction

□ Ordinateur = ensemble de dispositifs physiques

- ▣ **Processeur**
- ▣ **Mémoire**
- ▣ **Carte mère**
- ▣ **Périphériques**

■ Écrans, Disques durs, Imprimantes, ...



□ Problèmes

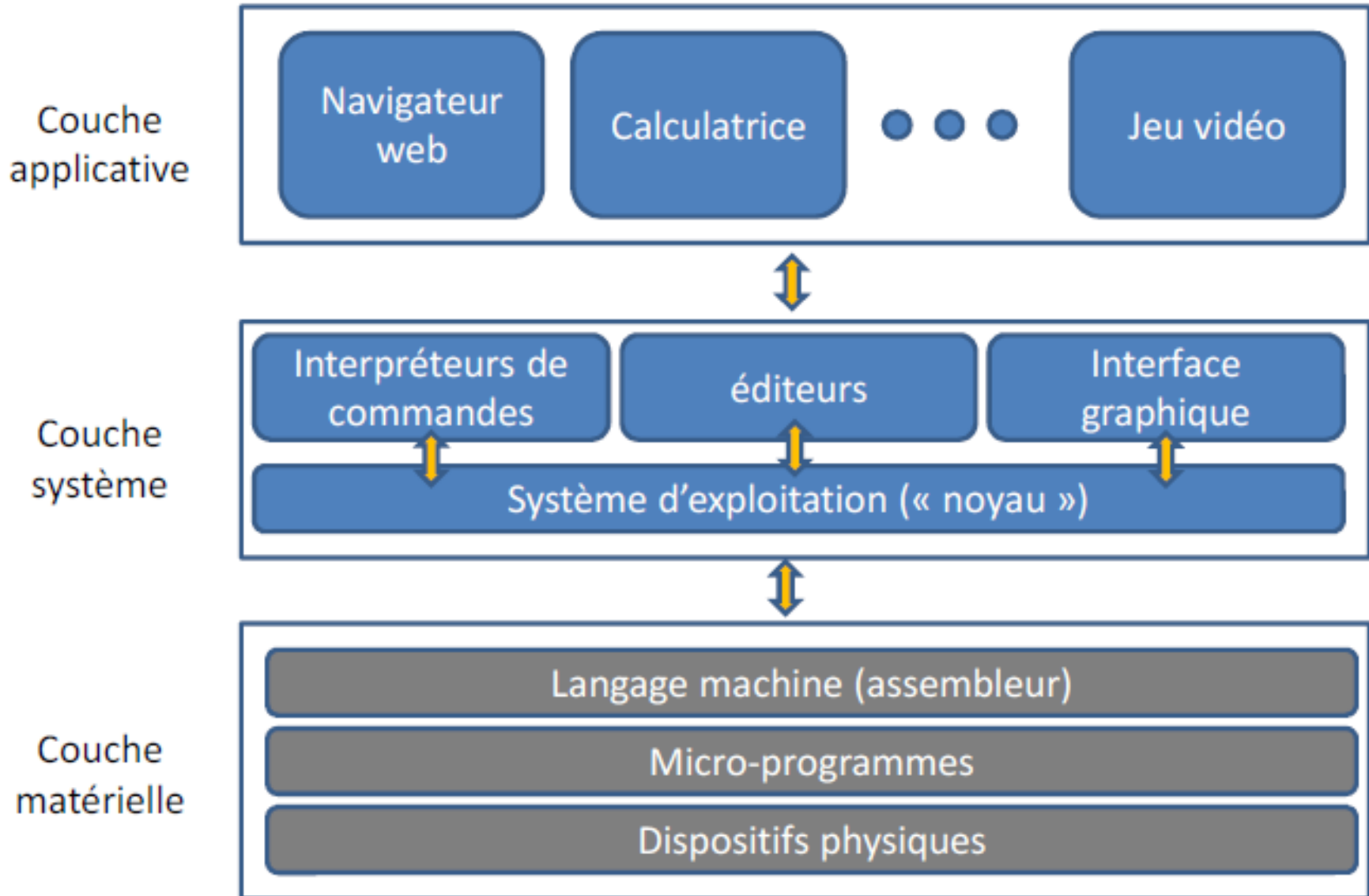
- ▣ **Comment faire fonctionner (correctement) tous ces dispositifs physiques ?**
- ▣ **Comment rendre leur utilisation simple à l'utilisateur ?**

□ Solution → utiliser un Système d'Exploitation (SE)

- ▣ **C'est un ensemble des programmes qui se chargent de tous les problèmes relatifs à l'exploitation des fonctionnalités d'un ordinateur**
- ▣ Interface entre la machine et l'utilisateur ou les programmes d'application
- ▣ Notion de machine virtuelle
 - Le système cache toutes les difficultés d'utilisation du matériel
 - L'utilisateur ne voit qu'une vue simplifiée de la machine

Introduction

□ Schématisation



Introduction

□ Exemples de système d'exploitation

- **MS-DOS/Windows**
- **Unix/Linux**
- **Mac/OS**
- **Android**
- ...



□ Où trouve-t-on les S.E. ?

- **Ordinateurs**
- **Téléphones portables**
- **Smartphone**
- **Cartes à puce**
- ...



Introduction

□ Définitions

▣ Noyau (kernel)

- Partie résidente du SE
- **Chargé en MEMOIRE CENTRALE (RAM) au démarrage de l'ordinateur**
- Fonctionne en mode superviseur, dans lequel les programmes peuvent exécuter toutes les instructions du processeur

▣ Mode utilisateur

- Les applications s'exécutent en mode utilisateur, dans lequel **les programmes ont un accès restreint aux fonctionnalités du processeur**. Ainsi, pas d'accès direct au matériel (périphériques, horloges, disques, mémoire...). L'accès à toutes ces fonctionnalités se fait en utilisant les **appels système (fonctions système)**

▣ Multiprogrammation

- Plusieurs programmes sont exécutés en même temps sur la machine
- **Objectif : optimiser l'utilisation du processeur**
 - Les entrées-sorties (lecture/écriture en mémoire, sur disque, etc.) sont beaucoup plus longues à exécuter que les instructions classiques du processeur

➔ **Implique un partage des ressources qui doit être géré par le système**

▣ Temps partagé

- Plusieurs utilisateurs utilisent simultanément la machine
- Connexion depuis leur propre terminal

➔ **Implique un contrôle des ressources en fonction des utilisateurs**

Introduction

□ Les classes de S.E.

▣ Mono-utilisateur

- Un seul utilisateur
- Généralement pas de notion d'utilisateur ni de mécanisme de protection des données
- Exemple : MS-DOS, Windows 3.1

▣ Multiutilisateurs

- Plusieurs utilisateurs
- Multitâches
- Implémentent des mécanismes de protection
- Exemples : Linux, Mac-OS

▣ Systèmes temps-réel

- Contrôle de processus complexes en milieu industriels

▣ Systèmes distribués

- Gèrent un réseau de machines distantes
- L'utilisateur ne voit qu'une seule machine

Plan

- Introduction
- **Notion de SGF (Système de Gestion de Fichier)**
- Interpréteur de commandes
- Implantation des systèmes de gestion de fichiers
- Notion de processus
- Systèmes de gestion de la mémoire
- Les scripts shell

Notion de SGF

□ Tâche la plus visible du SE

- ▣ Organiser les données de l'utilisateur
- ▣ Organiser les données du système
- ▣ Être indépendant du périphérique de stockage

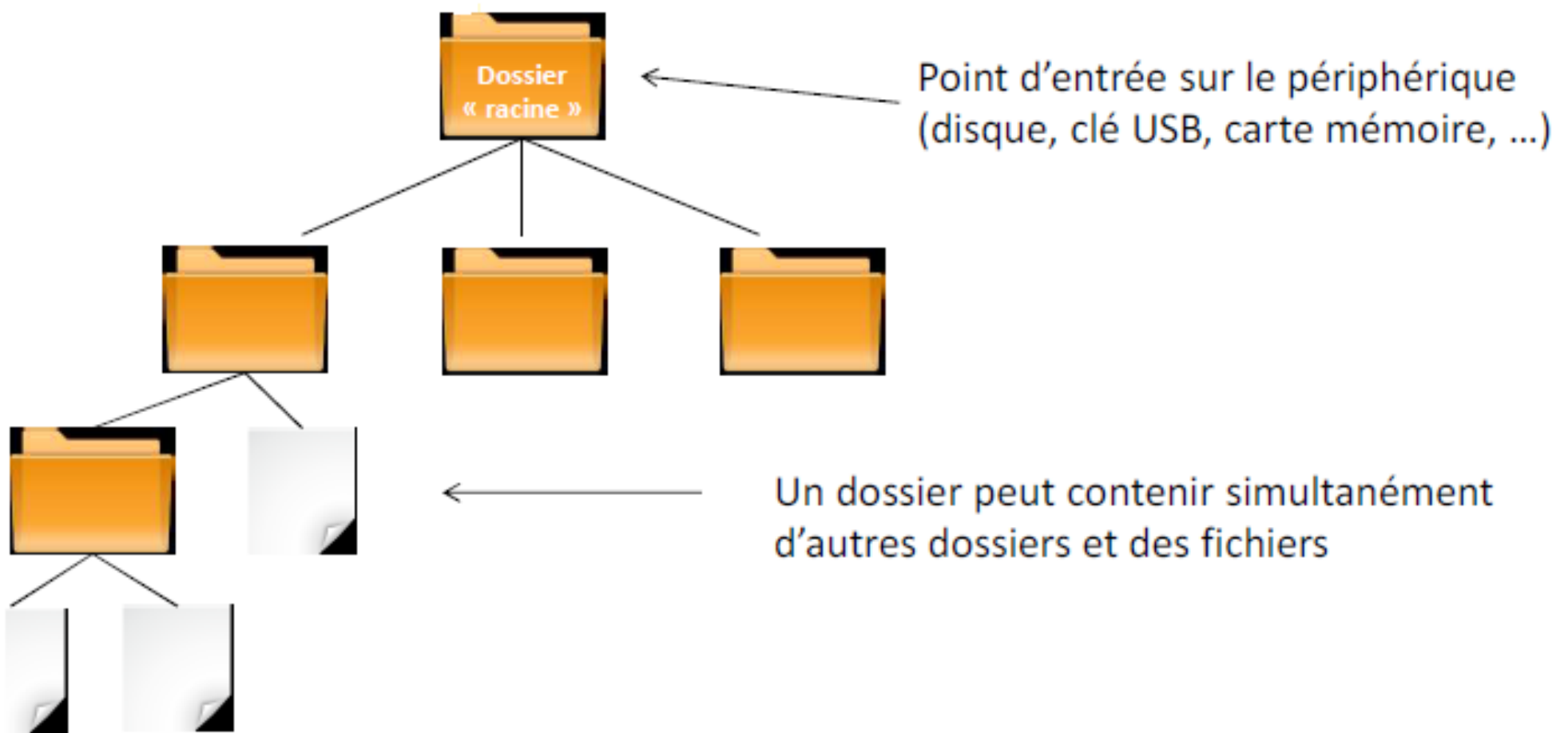
□ Définitions

- ▣ **Fichier** : ensemble de données ayant un lien logique, mémorisées sous un nom unique
 - **Exemple** : les pixels d'une image, les caractères constituant le texte d'un rapport, etc.
- ▣ **Dossier (répertoire)** : ensemble de fichiers ayant un lien logique entre eux et mémorisés au même endroit
 - Les fichiers sont organisés en répertoires (directory) et sous-répertoires (subdirectory), formant une **structure arborescente**
 - Le répertoire qui est à l'origine de cette arborescence est appelé répertoire racine (root directory). Il est désigné par un nom standardisé : /
 - **Objectif** : faciliter la recherche des données par l'utilisateur
 - Le répertoire courant peut être désigné par un point : .
 - Le répertoire parent du répertoire courant peut être désigné par deux points : ..

Notion de SGF

□ Organisation du SGF

- ▣ **Arborescence** → la représentation la plus commune



Notion de SGF

□ Arborescence Linux

Dossier	Contenus
/	la racine, elle contient les répertoires principaux
/bin	les exécutables essentiels au système, employés par tous les utilisateurs
/boot	les fichiers de chargement du noyau, dont le chargeur d'amorçage
/dev	les points d'entrée des périphériques
/etc	fichiers de configuration nécessaires à l'administration du système (fichiers <i>passwd</i> , <i>group</i> , <i>inittab</i> , <i>shadow</i> , <i>resolv.conf</i> , ...)
/home	les répertoires personnels des utilisateurs
/lib	les bibliothèques standards partagées entre les différentes application du système
/mnt	les points de montage des partitions temporaires (cd-rom, flashdisk, nfs ...)
/proc	ensemble de fichiers virtuels permettant d'obtenir des informations sur le système et les processus en cours d'exécution
/root	répertoire personnel de l'administrateur root
/sbin	les exécutables système essentiels
/tmp	les fichiers temporaires ➔ effacés au redémarrage du système
/usr	les fichiers binaires, les commandes utilisateurs, les bibliothèques, les codes sources, ...
/var	les fichiers de bases de données (ftp, www), les fichiers journaux (logs)

Notion de SGF

□ Format des noms de fichier

▣ Dépendent du système d'exploitation

■ Format, Caractères autorisés

MS-DOS	UNIX
8 caractères . 3 caractères (ex: fichier1.exe)	Longueur d'un nom de fichier quelconque
Pas d'espace, ni de caractères non alphanumériques	Tous les caractères sont autorisés, sauf le /
Majuscules et minuscules non différenciées	Différenciation des majuscules et des minuscules
Extension significative - Applications : .bin, .com, .exe - fichiers word : .doc, .docx	Extension non significative
	Les fichiers commençant par . sont des fichiers cachés (par défaut)

▣ Conseil sous linux (lisibilité)

- Éviter les espaces et les caractères spéciaux
- Noms de fichiers en minuscule
- Noms des dossiers personnels commençant par une majuscule
- Conserver l'extension pour les formats standards (portabilité)

Notion de SGF

□ Notion de droits

- **Principe** : différencier les utilisateurs en fonction de leurs prérogatives

- **Objectifs**

- Sécuriser les données des utilisateurs
- Sécuriser les données systèmes
- Fondamental dans le cas d'un système multiutilisateurs

- **Sous Windows il y'a deux types d'utilisateurs**

- **Administrateur** : possède le droit de modifier le système (installation de logiciels par exemple)
- **Non administrateur** : ne peut modifier que ses propre données

- **Sous Linux il y'a 3 types d'utilisateurs**

- **User** : le propriétaire d'un fichier ou d'un dossier
- **Group** : chaque utilisateur appartient à un groupe, créé par l'administrateur du système
- **Other** : tous les autres utilisateurs du système
- **NB** : existence d'un super utilisateur « **root** » qui a tous les droits ➔ **administrateur du système**

Notion de SGF

□ Gestion des droits sous Linux

□ 3 types de droits

■ Lecture (r)

- Le contenu du fichier peut être lu ou copié
- Le contenu du répertoire peut être visualisé

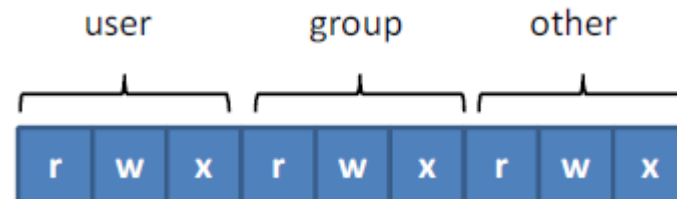
■ Écriture (w)

- Modification autorisée pour le fichier
- Création ou suppression de fichiers possible dans un dossier

■ Exécution (x)

- Le fichier peut être exécuté (c'est une application ou un script)
- Le dossier peut être accédé (on peut s'y déplacer)

□ Chaque fichier/dossier dispose donc de 9 droits distincts



□ Exemple



- Fichier non exécutable, éditable par son propriétaire et consultable uniquement par les membres du groupe

Notion de SGF

□ Point de vue utilisateur

□ Manipulation du SGF via

- une interface graphique
- un ensemble de **commandes** qui dépendent du système d'exploitation
 - L'utilisation de commandes nécessite de pouvoir préciser l'emplacement d'un fichier/dossier dans l'arborescence → **Syntaxe spécifique au système**
 - L'utilisation de commandes nécessite de disposer d'un programme permettant d'interpréter et d'exécuter ces commandes → **Interpréteur de commandes**

□ Chemins d'accès (linux)

□ Deux types de chemins

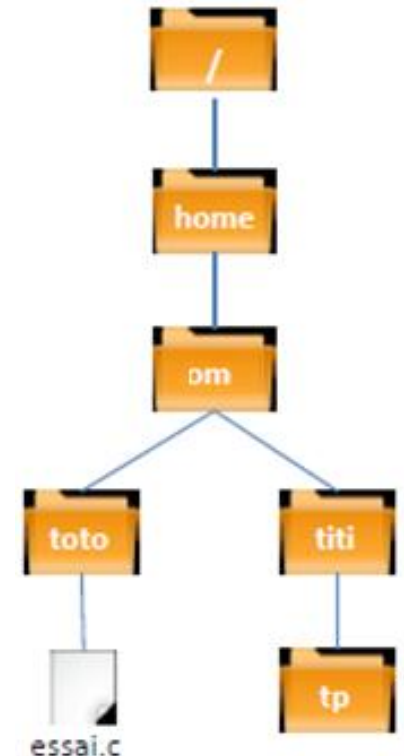
- **Chemins absolus** : on part à chaque fois de la racine du SGF (/)
- **Chemins relatifs** : on part de l'endroit où l'on se trouve

□ Exemple : accès à **essai.c** depuis **home**

- Chemin absolu : **/home/om/toto/essai.c**
- Chemin relatif : **./om/toto/essai.c** (caractère **.** = partir du dossier courant)

□ Exemple : accès à **essai.c** depuis **tp**

- Chemin absolu : **/home/om/toto/essai.c**
- Chemin relatif : **../../toto/essai.c** (caractères **..** = remonter d'un dossier)



Plan

- Introduction
- Notion de SGF (Système de Gestion de Fichier)
- **Interpréteur de commandes**
- Implantation des systèmes de gestion de fichiers
- Notion de processus
- Systèmes de gestion de la mémoire
- Les scripts shell

Interpréteur de commande

□ Programme permettant d'envoyer des instructions au système

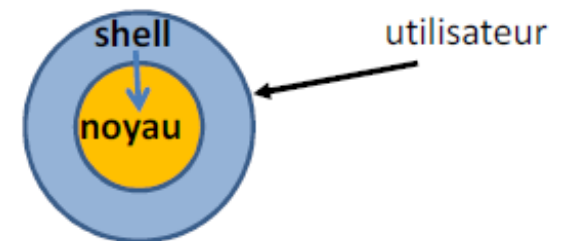
- Utilisation de commandes spécialisées, qui dépendent du système d'exploitation
- Exemple : afficher le contenu d'un dossier
 - Windows : **dir** *nom_dossier* Linux : **ls** *nom_dossier*

□ Intérêts

- Historiquement : pas d'interface graphique
- Commandes complexes : interfaces graphiques peu adaptées
- Efficacité : **souvent plus rapide d'utiliser une commande que l'interface graphique**
- Gestion à distance

□ Plusieurs versions d'interpréteurs «shell» existe sous Unix/Linux

Shell unix les plus courants	
<u>Nom / commande</u>	<u>signification</u>
sh	Bourne shell
csh	C shell
bash	Bourne again shell
tcsh	Tenex c shell



- **Chaque utilisateur dispose d'un interpréteur de commande fixée par l'administrateur système**

Interpréteur de commande

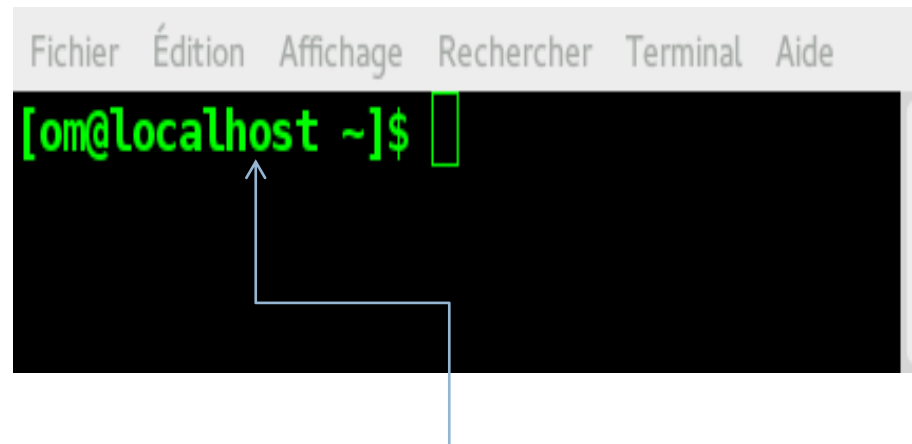
□ Lancement du Shell

▣ Sans interface graphique

- À la connexion, l'utilisateur se trouve directement sur son interpréteur de commandes

▣ Avec interface graphique

- Lancer un terminal



« Prompt »

[Nom_utilisateur@nom_machine:dossier_courant]\$

- ▣ **Remarque :** ~ correspond au dossier racine de l'utilisateur

Interpréteur de commande

□ Commandes de base

- ▣ Syntaxe générale : **Nom_commande** [options] [paramètres]
- ▣ Exemple : afficher le contenu du dossier courant (dossier dans lequel on se trouve)

- Commande **ls**

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost SE]$ ls
lancement.png  ls_l.png  ls.png  test.pdf  TP1
[om@localhost SE]$
```

- Commande **ls** avec l'option **-l**

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost SE]$ ls -l
total 40
-rw-rw-r--. 1 om admin 4919 4 févr. 18:45 lancement.png
-rw-rw-r--. 1 om om 12503 4 févr. 00:51 ls_l.png
-rw-rw-r--. 1 om om 9296 4 févr. 18:47 ls.png
-rw-rw-r--. 1 om admin 0 4 févr. 00:19 test.pdf
drwxrwxr-x. 2 om admin 4096 4 févr. 00:30 TP1
```

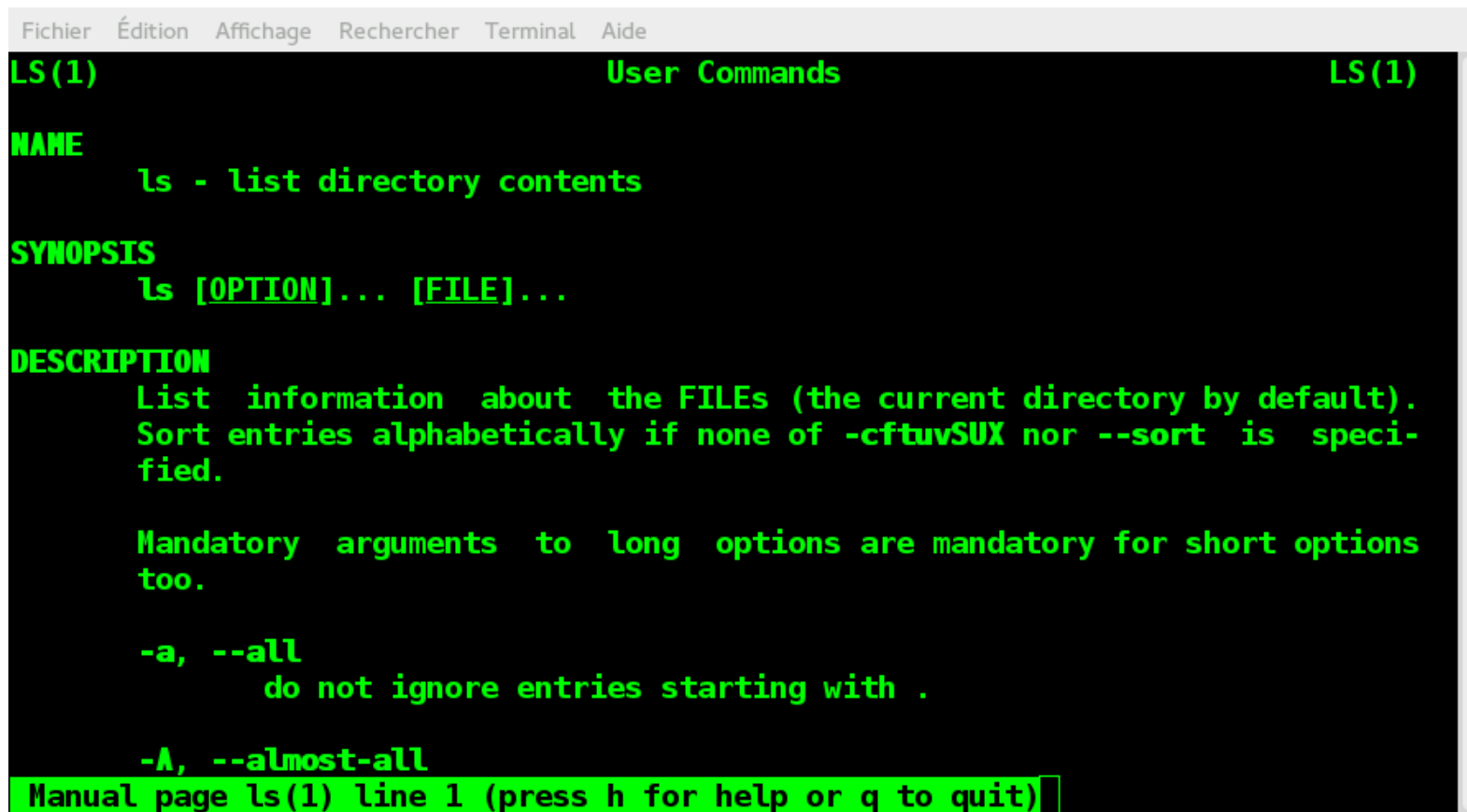
fichier → -rw-rw-r--.
dossier → drwxrwxr-x.

droits propriétaire groupe taille en octets date et heure de dernière modification nom

Interpréteur de commande

□ Les pages de manuel

- ▣ **Sous linux, chaque commande est documentée dans une page de manuel**
- ▣ La commande «**man**» permet d'afficher la page correspondant à la commande recherchée
- ▣ Syntaxe : **man** *nom_commande*
 - Exemple : **man** *ls*



```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
LS(1)                                     User Commands                                     LS(1)

NAME
  ls - list directory contents

SYNOPSIS
  ls [OPTION]... [FILE]...

DESCRIPTION
  List information about the FILES (the current directory by default).
  Sort entries alphabetically if none of -cftuvSUX nor --sort is speci-
  fied.

  Mandatory arguments to long options are mandatory for short options
  too.

  -a, --all
        do not ignore entries starting with .

  -A, --almost-all
        do not ignore entries starting with .

Manual page ls(1) line 1 (press h for help or q to quit)
```

Interpréteur de commande

□ Les méta-caractères

- ▣ Le «shell» interprète certains caractères comme ayant une définition spéciale

Caractère	Signification
*	une chaîne de caractères quelconques, de longueur quelconque (y compris nulle)
?	un caractère quelconque
[x-x']	tout caractère compris entre le caractère x et le caractère x' <ul style="list-style-type: none">• [0-9] : un chiffre• [a-z] : une lettre minuscule• [a-z,A-Z] : une lettre minuscule ou majuscule

▣ Exemples

- **ls f*** : lister tous les fichiers dont le nom commence par un **f**
- **ls *.jpg** : lister tous les fichiers se terminant par **.jpg**
- **ls *d*** : lister tous les fichiers dont le nom contient la lettre **d**

▣ Exercices

- Afficher tous les fichiers d'extension **.zip**
- Afficher tous les fichiers dont l'extension comporte **3 caractères**
- Afficher toutes les images jpeg (extension **.jpg**) qui comportent le mot **photo** ou **Photo** dans le nom de fichier

Interpréteur de commande

□ Manipulation des fichiers

Commande	Description
cp	copie d'un fichier dans un autre (cp <i>fichier nouveau_fichier</i>)
mv	<ul style="list-style-type: none">• Re-nommage d'un fichier (mv <i>nom_fichier nouveau_nom_fichier</i>)• Déplacement d'un fichier dans un autre dossier (mv <i>nom_fichier nom_dossier</i>)
cat	affichage du contenu du fichier dans le terminal (cat <i>nom_fichier</i>)
more (less)	idem, mais page par page (more <i>nom_fichier</i>)
head	afficher le début d'un fichier texte
tail	afficher la fin d'un fichier texte
touch	Change l'horodatage du fichier. Si le fichier n'existe pas, un fichier vide est créé
find	Recherche des fichiers dans des arborescences de répertoires
locate	plus rapide que find , utilise sa propre base de données de l'arborescence des fichiers. updatedb permet de mettre à jour la base de données de locate
chmod	changer les permissions d'un fichier (répertoire)

Interpréteur de commande

□ Manipulation des dossiers

Commande	Description
mkdir	création d'un dossier vide (mkdir <i>nom_dossier</i>)
rmdir	suppression d'un dossier <ul style="list-style-type: none">• rmdir <i>nom_dossier</i> (le dossier doit être vide)• rmdir -R <i>nom_dossier</i> (effacement récursif)
cd	changement de dossier courant <ul style="list-style-type: none">• cd <i>nom_dossier</i> (le dossier doit exister)• cd (retour dans le dossier racine de l'utilisateur)
cp	copie d'un dossier dans un autre <ul style="list-style-type: none">• cp <i>nom_dossier1 nom_dossier2</i> (1er niveau)• cp -R <i>nom_dossier1 nom_dossier2</i> (copie récursive)
mv	renommage d'un dossier (mv <i>nom_dossier1 nom_dossier2</i>)
pwd	affiche le nom (chemin) du répertoire courant (répertoire de travail)
ls	affiche le contenu d'un répertoire

Interpréteur de commande

□ Gestion des utilisateurs

- Pour accéder aux ressources du système informatique fonctionnant sous le système Unix il faut disposer d'un **compte**, c'est à dire être un **utilisateur (user)** du système
- Un utilisateur est reconnu par un nom unique (**login**) et un numéro unique (**UID - User Identifier**)
 - Ces données avec d'autres informations concernant le compte (mot de passe, répertoire personnel et autres) sont stockées dans les fichiers **/etc/passwd** et **/etc/shadow**
- Le **super utilisateur** (super user) du système : login: **root**, **UID : 0**
- Un utilisateur UNIX appartient à un ou plusieurs **groupes** (**groupe primaire, groupes secondaires**)
- Chaque groupe est identifiée par un *nom unique* et un *numéro unique* (**GID – Group Identifier**)
- Chaque fichier (normal, répertoire ... etc) appartient à un **utilisateur et à un groupe**
 - Cette appartenance peut être à tout moment modifiée par le *super utilisateur* (commandes **chown** – *change owner* et **chgrp** – *change group*)

Commande	Description
useradd	créer un nouvel utilisateur (useradd login)
passwd	permet de changer le mot de passe (passwd login)
chown/chgrp	changer le propriétaire (groupe) d'un fichier (<i>chown</i> : root seulement) Ex. chown om *.jpg ; om devient propriétaire de tous les fichiers dont le nom termine par .jpg

Interpréteur de commande

□ Gestion des droits

- ▣ La commande **chmod** permet de modifier les droits des fichiers et dossiers (qui appartiennent à l'utilisateur qui lance la commande)
- ▣ Syntaxe : **chmod** [*options*] [*mode*] *fichier(s)*
 - **L'option principale -R** agit récursivement sur les dossiers présents dans la liste des fichiers fournis en paramètre
 - **Mode**
 - Précise le mode qui doit être appliqué au(x) fichier(s)
 - Utilise une notation symbolique :

ugo

Les droits à affecter s'appliquent sur le champ utilisateur (**u**), sur le champ groupe (**g**), sur le champ autre (**o**) ou sur tous les champs (**a**).

+ -=

L'opérateur précise si on ajoute les droits qui suivent (**+**), si on retire les droits qui suivent (**-**) ou si on donne explicitement la valeur des différents droits (**=**)

rwX

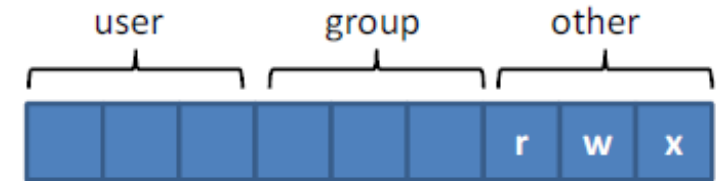
La liste contient les droits qui doivent être modifiés

Interpréteur de commande

□ Gestion des droits

▣ Exemples

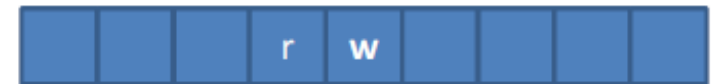
▪ `chmod o=rwx nom_fichier`



▪ `chmod o=r nom_fichier`



▪ `chmod g+rw nom_fichier`

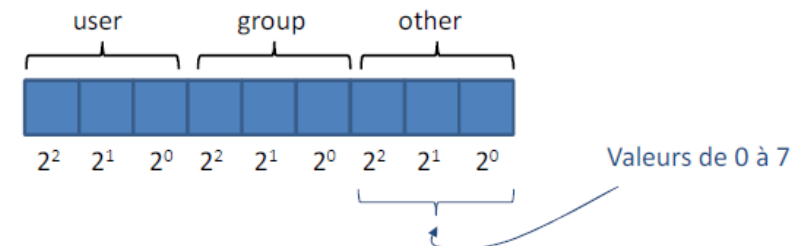


▪ `chmod a-w nom_fichier`

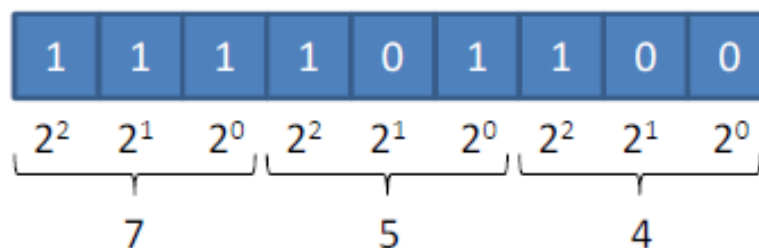


▣ Notation octale : chaque triplet *rwX* se voit attribuer une puissance de 2

- Mise à **1** de la position : **le droit est activé**
- Mise à **0** de la position : **le droit n'est pas activé**
- On fait la somme des poids binaires pour chaque triplet



▣ Exemple



`chmod 754 nom_fichier`

Interpréteur de commande

□ Mécanismes de redirection

- ▣ **Sous Unix/linux, toutes les entrées/sorties sont vues comme des fichiers**
 - Fichiers, dossiers
 - Périphériques (écrans, clavier, imprimantes, disques, etc.)
- ▣ **3 E/S standard** existent au démarrage
 - **Entrée standard** : par défaut, **le clavier**
 - **Sortie standard** : par défaut **l'écran**
 - **Sortie standard d'erreur** : par défaut **l'écran**
- ▣ Possibilité de **rediriger** ces E/S via quelques opérateurs
 - Redirection vers un fichier : **>**
 - Concaténation vers un fichier : **>>**
 - Lecture dans un fichier : **<**
 - Redirection vers une autre commande : **|**

Interpréteur de commande

□ Mécanismes de redirection

▣ Redirection vers un fichier

- Syntaxe : **commande** **>** *nom_fichier*
- Les résultats affichés par la commande sont écrits dans le fichier dont le nom est donné
- Si le fichier existait avant l'exécution de la cmd, son ancien contenu est effacé ; sinon le fichier est créé
- Exemple : **ls** **>** *resls.txt*

▣ Redirection avec concaténation vers un fichier

- Syntaxe : **commande** **>>** *nom_fichier*
- Les résultats affichés par la commande sont écrits à la fin du fichier dont le nom est donné
- Si le fichier n'existait pas avant le lancement de la commande, il est créé
- Exemple : **ls** **>>** *resls.txt*

▣ Lecture des entrées dans un fichier

- Syntaxe : **commande** **<** *nom_fichier*
- La commande récupère les données dont elle a besoin dans le fichier donné
- Ne fonctionne que si la commande est susceptible de demander des données à l'utilisateur par l'intermédiaire du clavier ...

▣ Redirection vers une autre commande ➔ les Tubes (*pipes*)

- Syntaxe : **commande1** **|** **commande2**
- Toutes les données affichées en sortie par la commande 1 sont envoyées à la commande 2 qui les prend comme entrée

Interpréteur de commande

□ Notion de liens

- ▣ **Objectif** : partager des données sans les dupliquer

- ▣ **2 types de liens**

- **Liens symboliques**

- Création d'un fichier/dossier qui ne contient que le chemin d'accès à la donnée (fichier/dossier) partagée
 - c'est un raccourci pointant vers un fichier ou un répertoire
- **On peut établir des liens symboliques entre différents systèmes de fichiers**
- **Inconvénient : le système ne sait pas que les données sont partagées**
 - si le propriétaire des données change leur emplacement ou les efface, le lien devient invalide sans que l'utilisateur du lien le sache

- **Liens physiques**

- Création d'un fichier/dossier qui fait référence aux données physiquement présentes sur le périphérique de stockage
- Ajout d'un compteur de liens qui permet au système de connaître le nombre de fichiers/dossiers partageant les données
- **Les liens durs peuvent être établis seulement entre fichiers appartenant au même système de fichiers (partition)**
- Le fichier existe tant qu'il existe au moins un lien dur vers ce fichier
 - Le nombre de liens durs est inscrit dans l'inode et quand il tombe à zéro l'espace disque occupé par le fichier et son inode sont libérés. (C'est à dire le fichier est effacé)
 - **rm** efface un lien du répertoire ! et décrémente le nombre de liens dans l'inode

Interpréteur de commande

□ Notion de liens

▣ Syntaxe : **ln** [option] nom_du_fichier_cible nom_du_lien

Création d'un lien symbolique
(option **-s**)

Création d'un lien physique
(par défaut)

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost SE]$ ln -s test.pdf lien_symbolique
[om@localhost SE]$ ln test.pdf lien_physique
[om@localhost SE]$ ls -l
total 64
drwxr-xr-x. 2 om admin 4096 5 févr. 20:27 captures
-rw-r--r--. 2 om admin 25559 5 févr. 20:33 lien_physique
lrwxrwxrwx. 1 om admin 8 5 févr. 20:36 lien_symbolique -> test.pdf
-rw-r--r--. 2 om admin 25559 5 févr. 20:33 test.pdf
drwxrwxr-x. 2 om admin 4096 4 févr. 00:30 TP1
[om@localhost SE]$
```

Lien
symbolique

Nombre de références
vers le fichier physique

Lien matériel vers le
fichier physique

Lien d'accès vers le
fichier physique

Interpréteur de commande

□ La commande find

▣ Objectif : **trouver un fichier ou un groupe de fichiers dans l'arborescence**

- Les fichiers trouvés sont affichés dans la console
- Possibilité de leur appliquer une action

▣ Syntaxe simplifiée

Emplacement à partir duquel effectuer la recherche

- Recherche récursive dans les sous-dossiers
- Par défaut, recherche à partir du dossier courant

Critères de sélection
des fichiers

find **[chemin]** **[options]** **[tests]** **[actions]**

Paramétrage de la manière dont la
recherche va s'effectuer
Ex. : profondeur maximale de recherche

Actions à appliquer
sur les fichiers trouvés
(**-print**, **-delete**, ...)

- Les options sont nombreuses et les critères de recherche (*tests*) très variés

- recherche par nom **-name**, temps de dernière modification **-mtime** ou accès **-atime**, propriétaire **-user**, permissions **-perm** et bien d'autres (**voir man**)

Interpréteur de commande

□ La commande *find* – Principaux tests

Test	Signification
-name <i>motif</i>	Recherche par nom de fichier
-type <i>c</i>	Recherche par type de fichier f – fichier normal, d – répertoire, l – lien symbolique, s – socket (voir <i>man</i>)
-user <i>nom_user</i>	Recherche par propriétaire (<i>Non</i> ou <i>UID</i>)
-group <i>nom_group</i>	Recherche par appartenance à un groupe
-size <i>n[ckMG]</i>	Recherche par taille de fichier . Par défaut <i>n</i> blocs de 512 octets, sinon avec le suffixe c – octets, k – kilooctets, M – mégaoctets, G – gigaoctets
-atime <i>n</i>	Recherche par temps de dernier accès (dernier accès il y a <i>n*24 heures</i>)
-mtime <i>n</i>	Recherche par temps de dernière modification (dernière modification il y a <i>n*24 heures</i>)
-perm <i>permissions</i>	Recherche par permissions d'accès (<i>permissions</i> : en octal ou noms symboliques)
-links <i>n</i>	Recherche par nombre de liens au fichier

□ **N.B.** : dans les tests le paramètre numérique *n* a la signification suivante

- **+n** : plus grand que *n*; **-n** : plus petit que *n*; **n** : exactement *n*;
- Le caractère **!** avant le test inverse la condition (ex. **!-type d** veut dire "tous les fichiers qui ne sont pas des répertoires")

Interpréteur de commande

□ La commande *find* – Exemples

▣ **find . -type d -name "*s" -print**

- affiche (action **-print**) tous les répertoires (option **-type d**) contenu dans l'arborescence du répertoire courant (.) dont le nom se termine par 's'

▣ **find /tmp -name "core" -type f -print**

- recherche dans **/tmp** et sous-rép. les fichiers réguliers (**-type f**) au nom **"core"** et affiche leurs noms complets

▣ **find \$HOME -mtime 0**

- recherche à partir de votre rép. perso et affiche tout les noms de fichiers modifiés il y a moins de 24 heures

▣ **find . -perm 654**

- recherche à partir du rép. courant les fichiers avec les permissions **rw-rx-r-**

▣ **find ~ -regex ".*boucle\.c\$"**

- recherche de **boucle.c** en utilisant une expression régulière

▣ **find . ! -user root -print**

- affiche tous les fichiers qui n'appartiennent pas à **root**

▣ **find ~ -size +10M -delete**

- supprime tous les fichiers dont la taille est supérieure à 10 mégaoctets

Interpréteur de commande

□ La commande grep

- ▣ **Objectif** : rechercher un «**motif**» dans un ensemble de fichiers
 - Les lignes contenant le motif sont affichées, précédées par le nom du fichier concerné
- ▣ **Syntaxe**

Paramètre la manière dont la recherche va s'effectuer

Liste des fichiers dans lesquels rechercher le motif

grep

[**options**]

motif

[**fichiers**]

Une expression régulière

- **NB**: si aucun fichier est précisé, **grep** lu de l'entrée standard (ex. clavier, tubes...)
- ▣ **Exemples d'options**
 - **-i** : permet de ne pas différencier majuscules et minuscules
 - **-E** : interpréter le MOTIF comme une expression rationnelle étendue
 - **-v** : inverser le sens de la correspondance pour sélectionner les lignes qui ne correspondent pas
 - **-color** : affichage en couleur des correspondances trouvées
 - **-o** : afficher seulement les parties correspondantes des lignes trouvées

Interpréteur de commande

□ Expressions régulières

- **motif** décrivant un ensemble de chaînes de caractères
- utilisée dans les compilateurs, interpréteurs, utilitaires de recherche de motif (**grep, awk, ...**)
- utilise une syntaxe très précise
 - **Un caractère «classique»** : représente ce caractère tel quel (*c* représente la lettre *c*)
 - **Existences de caractères spéciaux (méta-caractères)**

Méta-caractères	Correspondances
.	un caractère unique quelconque
[]	Une liste de caractères, encadrée par [et] peut être mise en correspondance avec n'importe quel caractère appartenant à la liste (un seul caractère !) ex. [0-9A-Fa-f] : correspond à un chiffre hexadécimal → à l'intérieure d'une liste, la plupart des méta-caractères perdent leur signification spéciale
^	début de ligne (ex. ^begin : ligne qui commence par la chaîne <i>begin</i> , mais aussi <i>beginning ...</i>)
\$	fin de ligne (ex. end\$: ligne qui termine par la chaîne <i>end</i> , mais aussi <i>weekend ...</i> ; ^\$: ligne vide)
\<	Le symbole \< correspond à une chaîne vide en début de mot
\>	Le symbole \> correspond à une chaîne vide en fin de mot
	Alternance : deux expressions rationnelles peuvent être reliées par l'opérateur

Interpréteur de commande

□ Expressions régulières

▣ Répétitions d'un caractère unique ou d'une sous-expression (...)

Méta-caractères	Correspondances
?	ce qui précède peut être trouvé zéro ou une fois
+	ce qui précède doit être trouvé au moins une fois
*	ce qui précède peut être trouvé un nombre quelconque de fois
{n}	ce qui précède doit être trouvé <i>n</i> fois
{n,m}	ce qui précède doit être trouvé entre <i>n</i> et <i>m</i> fois
{n,}	ce qui précède doit être trouvé au moins <i>n</i> fois

- ▣ **NB** : pour utiliser les méta-caractères (. + * ^ \$ [\ | { ()) tel quel dans le MOTIF il faut les précéder du **\ (back-slash)** ex. \\$, \\, \., \{, ...

Interpréteur de commande

□ Expressions régulières

▣ Exemples

Motif	Correspondances
abc	la chaîne abc
[abc]	l'un des caractères a, b ou c
[a-e]	l'un des caractères a, b, c, d ou e
X[a-c]Y	les chaînes XaY, XbY ou XcY
Xa+Y	les chaînes XaY, XaaY, XaaaY, XaaaaY, ...
Xa*Y	les chaînes XY, XaY, XaaY, XaaaY, XaaaaY, ...
Xa{2,5}Y	les chaînes XaaY, XaaaY, XaaaaY ou XaaaaaY
[^f]ac	les chaînes de trois lettres qui se terminent par "ac" et ne commencent pas par 'f'
(alfa beta)X	les chaînes alfaX, betaX
^Info\$	que le mot "Info" seul dans une ligne
\<[[:alpha:]]{3}\>	tous les mots de 3 lettres
[+-][[:digit:]]{1,6}	nombres entiers avec signe, maximum 6 chiffres

▣ Exercices : donnez les correspondances des motifs suivants

- **[Aa]*llo**
- **[Aa]{1,2}L{2}o**

Interpréteur de commande

□ Application à grep

- Le motif à rechercher suit la syntaxe des expressions régulières

□ Exemples

- **grep [Hh]ello *.txt**

- ➔ Recherche les mots *Hello* et *hello* **n'importe où** dans les fichiers d'extension .txt

- **grep ^[Hh]ello *.txt**

- ➔ Recherche les mots *Hello* et *hello* **en début de ligne** dans les fichiers d'extension .txt

- **grep [Bb]ye\$ *.txt**

- ➔ Recherche les mots *Bye* et *bye* en fin de ligne dans les fichiers d'extension .txt

□ Exercices

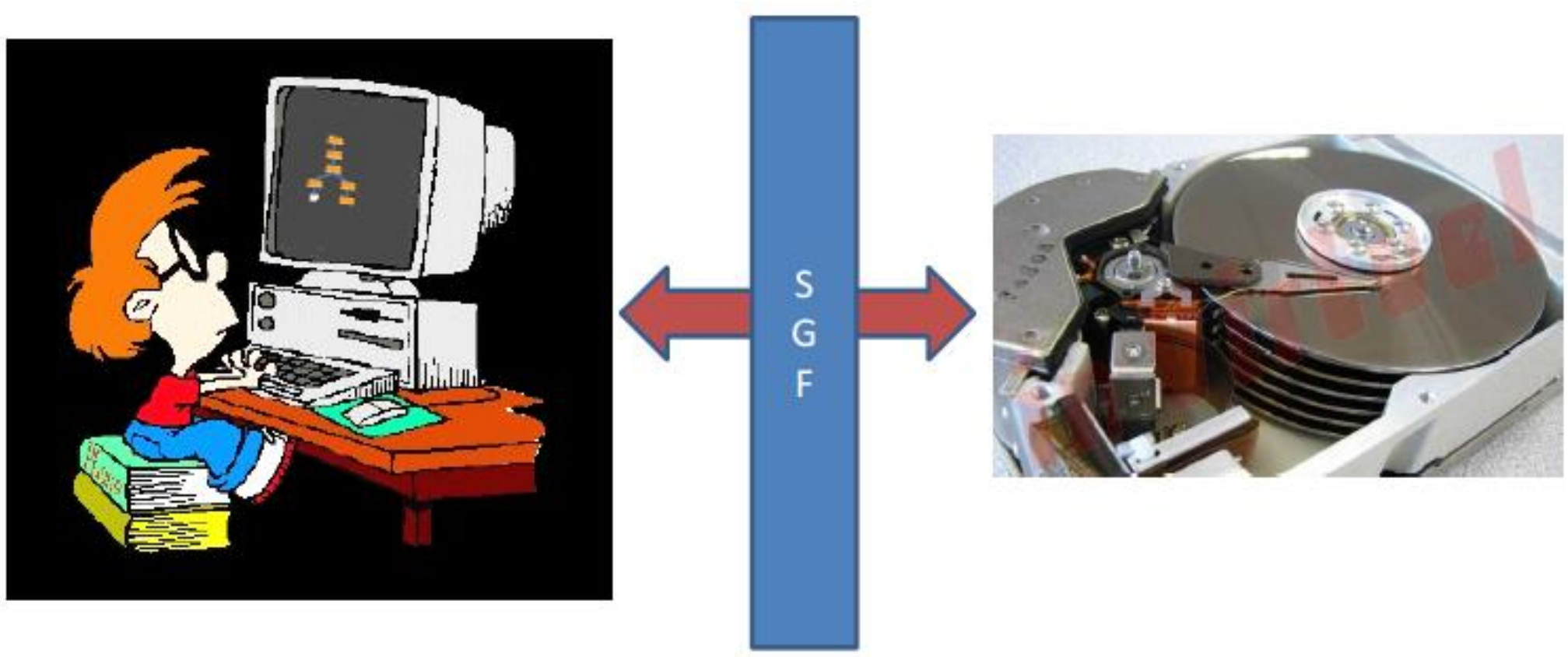
- Donner la syntaxe de la commande **grep** avec une expression régulière qui permet de vérifier le format
 - d'une adresse IP
 - d'une adresse email

Plan

- Introduction
- Notion de SGF (Système de Gestion de Fichier)
- Interpréteur de commandes
- **Implantation des systèmes de gestion de fichiers**
- Notion de processus
- Systèmes de gestion de la mémoire
- Les scripts shell

Implantation des SGF

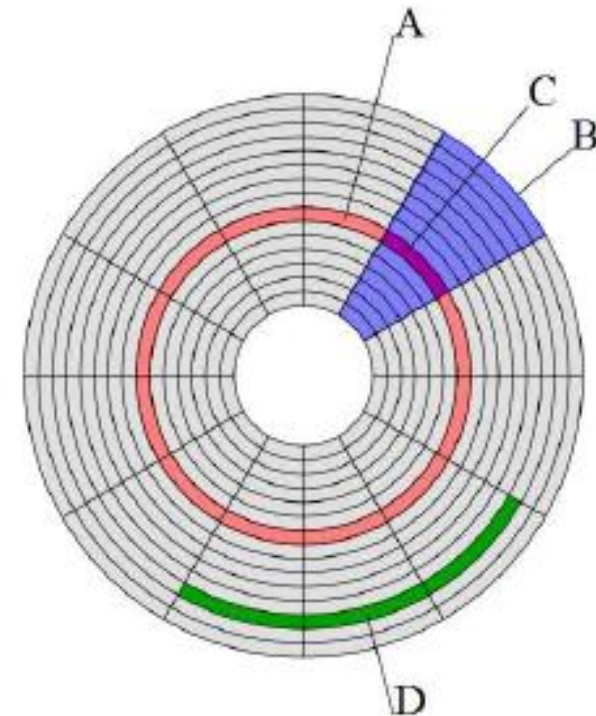
- **SGF (Systèmes de Gestion de Fichiers)**
 - ▣ **Interface entre l'utilisateur et le matériel**



Implantation des SGF

□ **Formatage d'un disque dur**

- ▣ **Plusieurs plateaux (disques) superposés**
- ▣ Chaque plateau est découpé en **pistes (A)** et en **secteurs géométriques (B)**
- ▣ Un secteur de données (C) est
 - Une intersection d'une piste et d'un secteur géométrique
 - La plus petite unité physique de stockage sur un support de donnée
 - capacité au minimum de 512 octets sur un disque dur
- ▣ Un Cluster (D) est
 - Un regroupement de plusieurs secteurs
 - Référencés dans le système de gestion de fichier
 - Chaque cluster possède un identifiant dans le SGF et une localisation sur le support de données
 - La plus petite unité de stockage logique
 - Un fichier est représenté par un nombre entier de cluster
 - La partie d'un cluster non totalement utilisée par les données du fichier ne peut pas être utilisée par un autre fichier
 - La taille d'un cluster doit être choisie consciencieusement au formatage du disque, en fonction du SGF utilisé



Découpage en secteurs = formatage de bas niveau
Découpage en clusters = formatage logique

Implantation des SGF

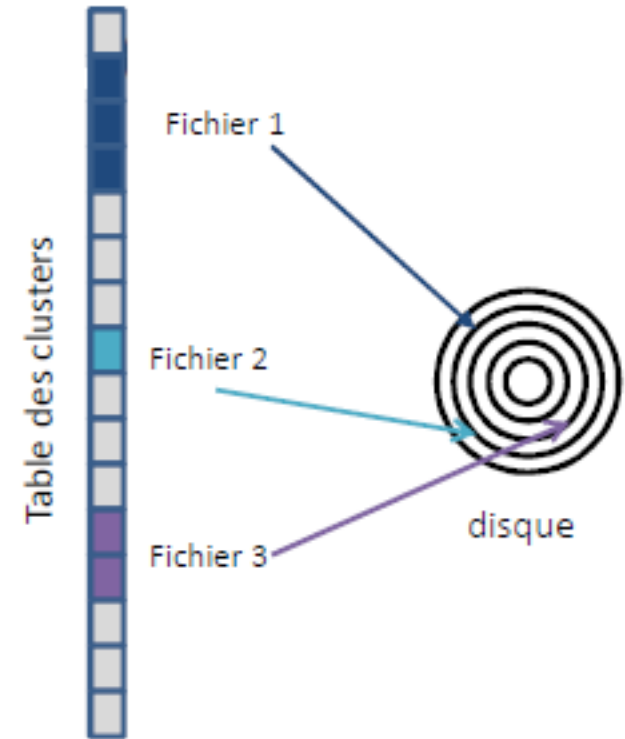
□ Formatage logique

▣ Objectif : **créer le SGF**

- Créer une table de clusters
 - Taille des clusters
 - Nombre de clusters
- Initialiser la table de clusters
 - Mémoriser leur emplacement physique
 - Initialiser leur état à « libre »

▣ Remarques

- Nombreux petits fichiers
 - Préférer une petite taille de cluster
 - ➔ **moins de perte d'espace disque**
- Nombreux gros fichiers
 - Privilégier une taille plus importante
 - ➔ **favorise la proximité des données sur le disque**
- Généralement on utilise des clusters de 4 Ko



Ex. fichier de 750 octets

Clusters 1ko



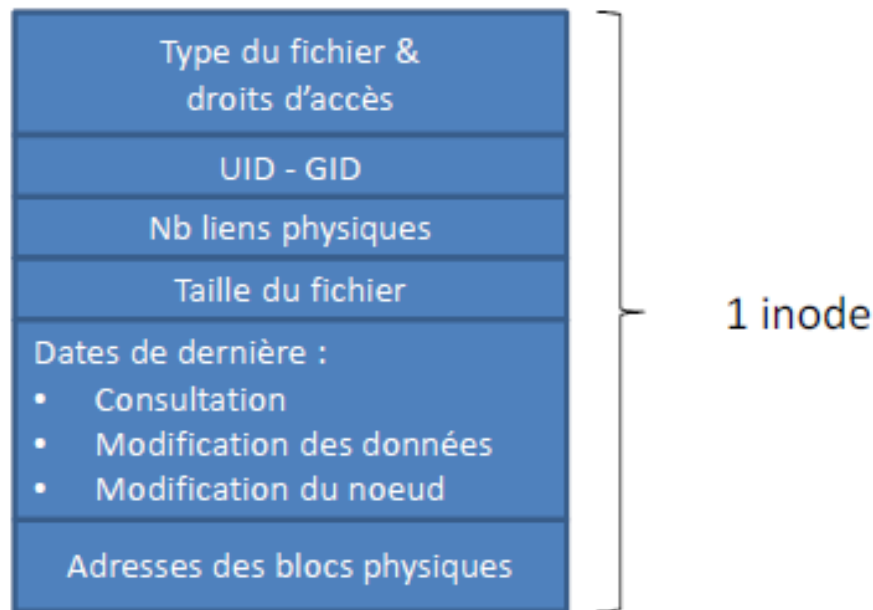
Clusters 4 ko



Implantation des SGF

□ Systèmes de gestion de fichiers UNIX

- Organisation autour de la notion des **i-nodes** (nœud d'index)
- **Les i-nodes sont des structures de données contenant des informations concernant les fichiers stockés dans le système de fichiers**
- Les informations stockées dans un i-node sont
 - Id du volume (périphérique, partition) contenant le fichier
 - Type de fichier; droit d'accès; propriétaire de fichier (*UID, GID*)
 - Nombre de liens physiques sur cet i-node; Taille du fichier
 - Dates de consultation, modification
 - Adresses des blocs (ou clusters) de données sur disque (emplacements physiques du fichier)



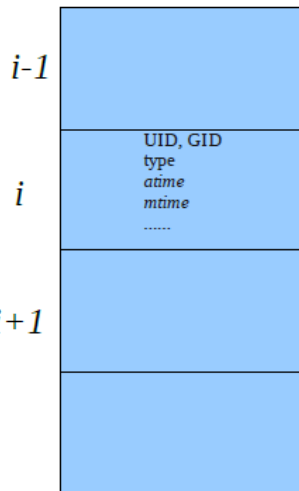
Implantation des SGF

□ Les i-nodes

- A chaque fichier correspond un numéro d'i-node (**i-number**) unique au volume sur lequel il est situé
 - Ce **i-number** est un indice dans la table d'i-nodes du volume
- **Représentation des répertoires**
 - Même principe que pour les fichiers → **un i-node est associé à chaque dossier**
 - C'est le type du i-node qui permet de différencier un dossier d'un autre objet du système
 - Contenu du dossier : **ensemble de couples (numéro de i-node, nom de fichier)**
 - **Un i-node ne contient pas le nom de fichier / dossier !!!**

Table d'inodes

...



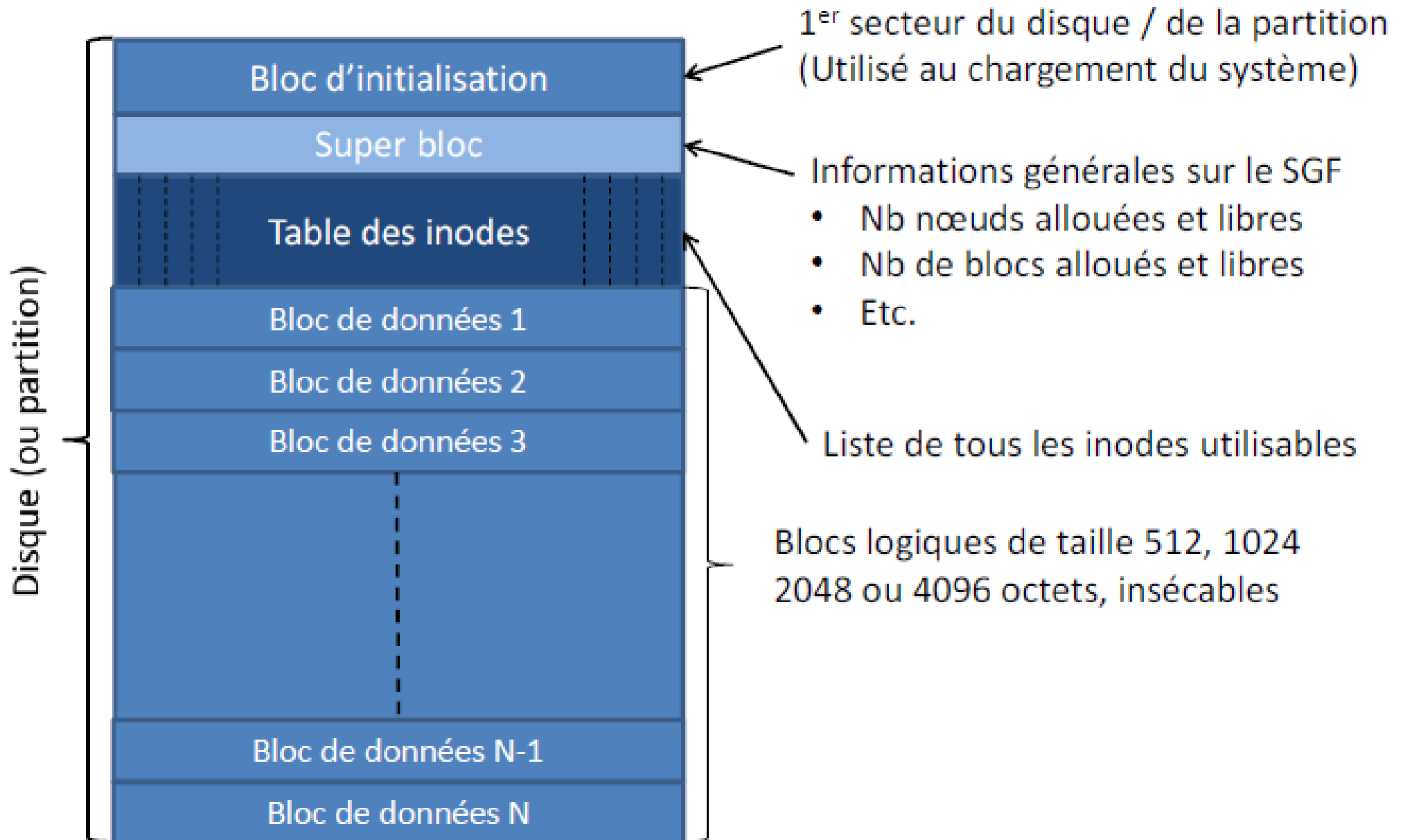
...

□ N.B.

- Le **i-number** est unique seulement sur la partition donnée
 - Les mêmes numéros sont utilisés sur des partitions différentes
- Environ 1% de l'espace disponible est affecté aux i-nodes
 - le nombre d'i-nodes correspond au nombre maximum de fichiers possible sur le volume
- Pour afficher le contenu d'un répertoire on utilise la commande **ls**
 - Pour afficher les **i-numbers** des fichiers on utilise la commande **ls** avec l'option **-i**
 - Pour afficher les informations contenues dans les **i-nodes** (détails sur les fichiers) on utilise la commande **ls** avec l'option **-l** (**Fedora** → **alias ll**)
- **attention** : Pour afficher les informations sur les fichiers **.** et **..** il faut utiliser l'option **-a**
 - Les fichiers dont le nom commence par un **.** sont des fichiers cachés (hidden files)

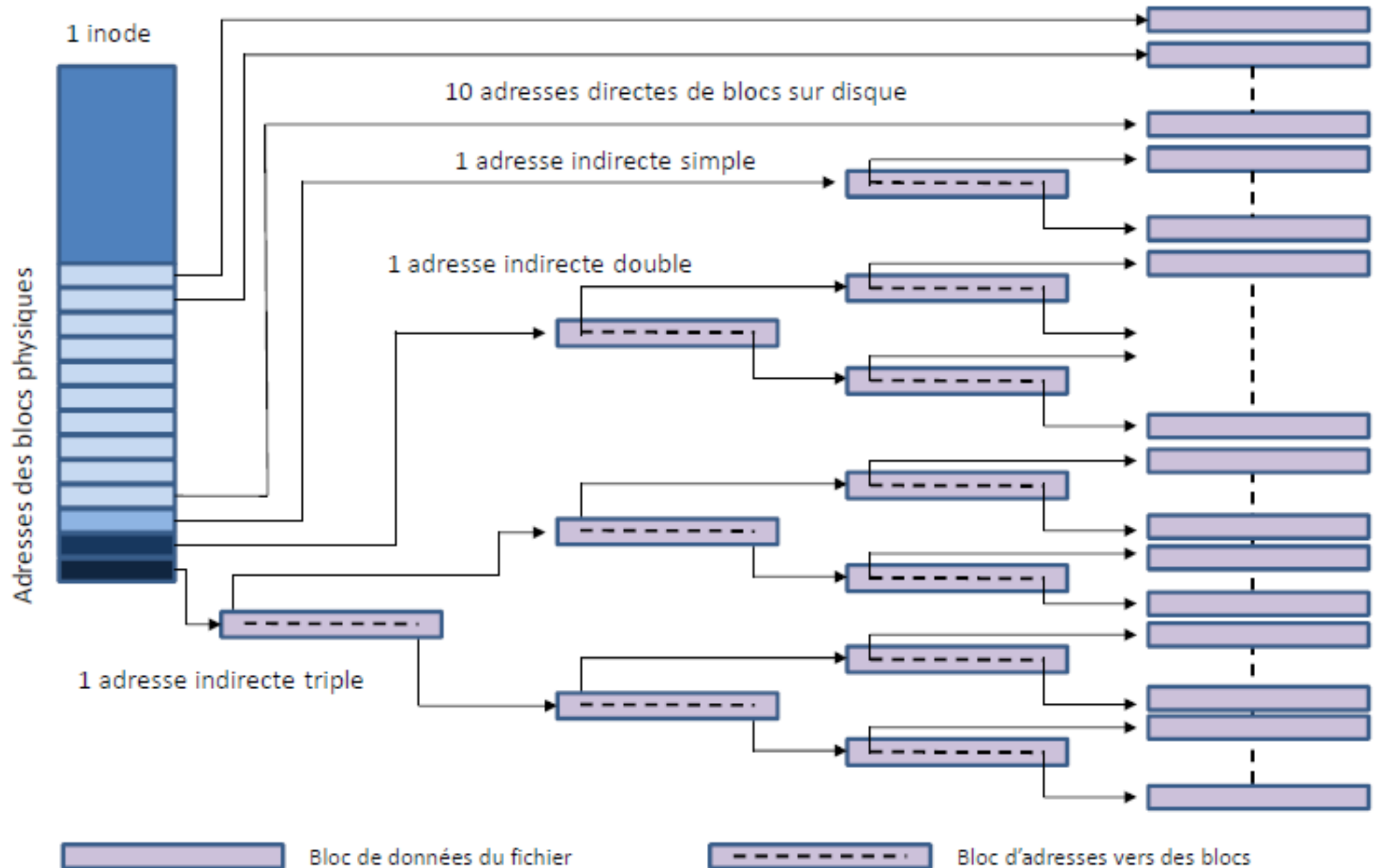
Implantation des SGF

□ Organisation du disque



Implantation des SGF

□ Système d'adressage des blocs



Implantation des SGF

□ Système d'adressage des blocs (exemple *EXT2*)

- **Ext2** est un système de fichiers courant sous Linux, bien que maintenant souvent remplacé par **Ext4** (en fait, Ext3 est un Ext2 avec un journal en plus)
- Chaque i-node contient environ **64 champs**, dont **13** d'entre eux contiennent des blocs pouvant être de deux types
 - Des blocs de données, qui contiennent les données du fichier
 - **Adresses directes** ➔ référencement aisé et rapide des « petits » fichiers
 - Des blocs d'adresses, qui contiennent des pointeurs vers d'autres blocs
 - **Adresses indirectes** ➔ Prise en compte de la taille croissante des fichiers
- Les 10 premiers champs (sur les 13) contiennent les adresses des 10 premiers blocs de données du fichier (à raison d'une adresse par bloc)
 - **Si les blocs sur lesquels pointent les 10 premiers champs sont suffisants pour contenir le fichier, les champs 11, 12 et 13 ne sont pas utilisés**
- Dans le cas contraire, en plus des 10 premiers blocs, les blocs 11, 12 et 13 sont utilisés
 - Ces blocs fonctionnent selon un système d'indirection
 - Il existe trois niveaux d'indirection
 - **La simple indirection, utilisée par le champ 11**
 - **La double indirection, utilisée par le champ 12**
 - **La triple indirection, utilisée par le champ 13**
 - **NB:** plus le niveau d'indirection est élevé, plus le nombre final de blocs de données sur lequel pointe le champ (11, 12 ou 13) sera élevé. Ce système permet donc aux fichiers d'avoir une taille considérable.

Implantation des SGF

□ Système d'adressage des blocs

▣ Avantages

- Accès rapide à n'importe quel bloc d'un fichier
 - Au plus 3 lectures de bloc avant d'obtenir l'adresse du bloc contenant la donnée souhaitée
 - Comparer au système FAT (chaînage des blocs)
- Taille potentiellement très importante des fichiers
- Taille réduite des **i-nodes**

▣ Exercice

- En supposant que les blocs ont comme taille **1024 octets (1 Kio)**, et que chaque adresse (dans le cas d'un bloc d'adresses) est stockée sur **32 bits (4 octets)**. Chaque bloc d'adresses en contiendra **256 blocs**.
- Avec ces informations en main, calculez la taille maximale d'un fichier dans le système *ext2*

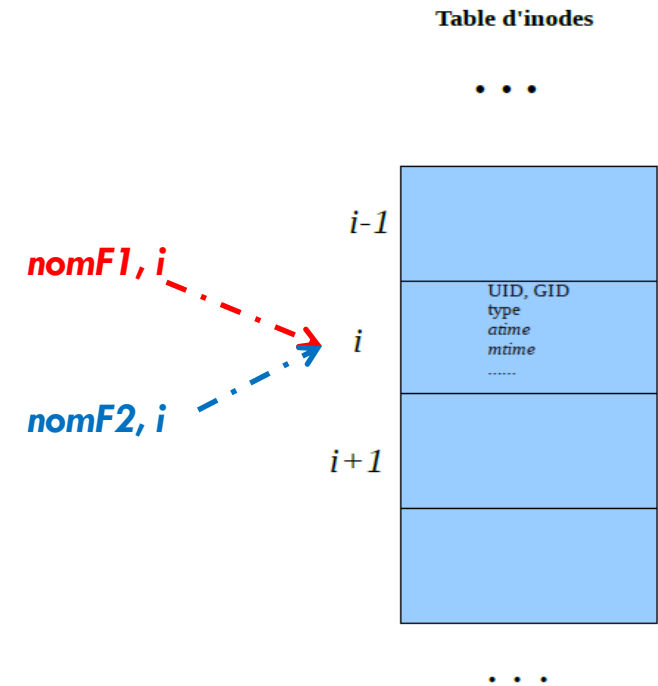
Implantation des SGF

□ Les i-nodes et les liens

- **Lien dur (lien physique, hard link)** : deux ou plusieurs entrées de répertoires ont le même *i-number*, c'est à dire pointent le même *i-node*, donc le même fichier physique. Cela permet de donner plusieurs noms au même fichier

□ NB:

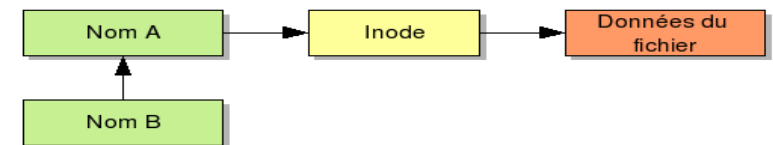
- Pas de lien durs entre les répertoires (seule exception : **.** et **..**)
- Les lien durs peuvent être établis seulement entre fichiers appartenant au même système de fichier (partition) [Pourquoi ?!]
- Le fichier existe tant qu'existe au moins un lien dur ver ce fichier. Le nombre de liens durs est inscrit dans l'i-node et quand il tombe a zéro l'espace disque occupé par le fichier et son i-node sont libérés. (C'est a dire le fichier est effacé)
 - La commande **rm** efface *un lien* du répertoire et **décrémente le nombre de liens dans l'i-node**



□ Lien symbolique (symbolic link, symlink)

- C'est un fichier (**Nom B**) contenant le nom de chemin du fichier (ou répertoire) **Nom A**. Autrement dit – c'est un raccourcis pointant ver un fichier ou un répertoire.

- **rm A ? rm B ?**



- On peut établir des liens symboliques entre différent systèmes de fichier.
- Pour créer un lien symbolique : commande **ln** avec l'option **-s**

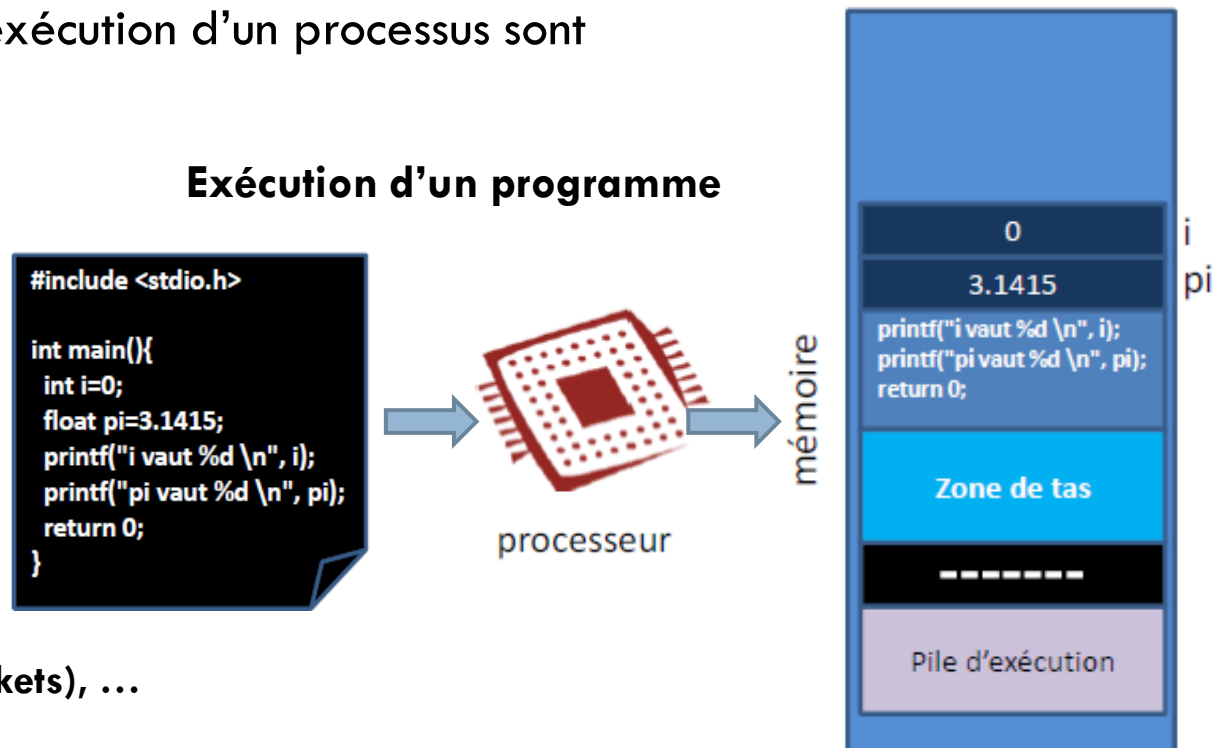
Plan

- Introduction
- Notion de SGF (Système de Gestion de Fichier)
- Interpréteur de commandes
- Implantation des systèmes de gestion de fichiers
- **Notion de processus**
 - ▣ **Processus lourds**
 - ▣ **Commandes Unix & processus**
 - ▣ **Les THREADS (processus légers)**
- Systèmes de gestion de la mémoire
- Les scripts shell

Processus lourds

□ Définitions

- ▣ **Un programme** : suite d'instructions chargées en mémoire au lancement
- ▣ **Un processus** (appelé aussi processus lourd) : programme en exécution avec tout son environnement (contexte d'exécution)
- ▣ Les ressources nécessaires à l'exécution d'un processus sont
 - **Zones mémoires** contenant
 - **données du programme**
 - **code du programme**
 - **tas (heap)**
 - **pile d'exécution (call stack)**
 - **Registres**
 - registres généraux
 - PSW (flags-drapeaux)
 - compteur ordinal
 - SP,...
 - **fichiers ouvert, ports réseau (sockets), ...**



Zone de tas

• **Allocation dynamique explicite** (demandée par le programmeur)

- Fonction **malloc** du langage C
- Opérateur **new** des langages C++ et Java

Pile d'exécution

- **Allocation dynamique implicite** (cachée du programmeur)
- Stockage des paramètres d'une fonction
- Stockage de l'adresse de retour d'une fonction
- Gérée comme une pile

Processus lourds

□ Systèmes multiprocessus

- Un ordinateur équipé d'un système d'exploitation **multitâches (à temps partagé)** est capable d'exécuter plusieurs processus de façon « quasi-simultanée »
- Le système d'exploitation est chargé d'allouer les ressources (mémoires, temps processeur, entrées/sorties) nécessaires aux processus et d'assurer que le fonctionnement d'un processus n'interfère pas avec celui des autres (**isolation**)
- **Plusieurs processus actifs simultanément → doivent se partager les ressources**
 - Accès au processeur à tour de rôle
 - Partage de la mémoire
 - Présence simultanée en différents endroits de la mémoire
 - Accès concurrents aux périphériques, ...
- Exemple de partage du processeur



Notion d'ordonnanceur de tâches (scheduler)

Processus lourds

□ Ordonnancement des processus sous UNIX

□ Trois états possibles pour chaque processus

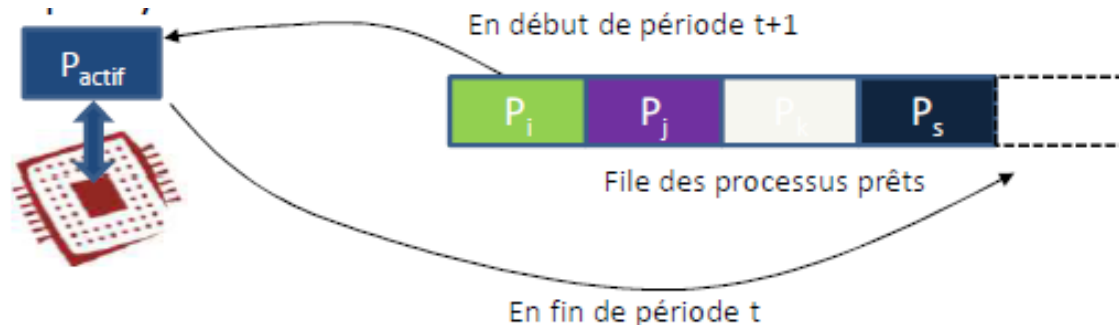
- **élu** (actif : en cours d'exécution)
- **prêt** (suspendu, pour permettre l'exécution d'un autre processus)
- **bloqué** (attente d'un évènement extérieur : entrée/sortie, signal...)

□ Deux files d'attentes principales dans le noyau du système

- **file des processus prêts**
- **file des processus bloqués**

□ L'**ordonnanceur (Scheduler)** : c'est le composant du noyau qui choisit les processus prêts qui vont être exécutés par le (les) processeur(s) d'un ordinateur

- **sans préemption** : FCFS (First Come First Served), SJF (Shortest Job First), à priorité
- **avec préemption** : SJF, Round Robin (tourniquet), les algorithmes à queues multiples ...
- Exemple : **tourniquet (gestion circulaire)** → politique classique d'ordonnancement



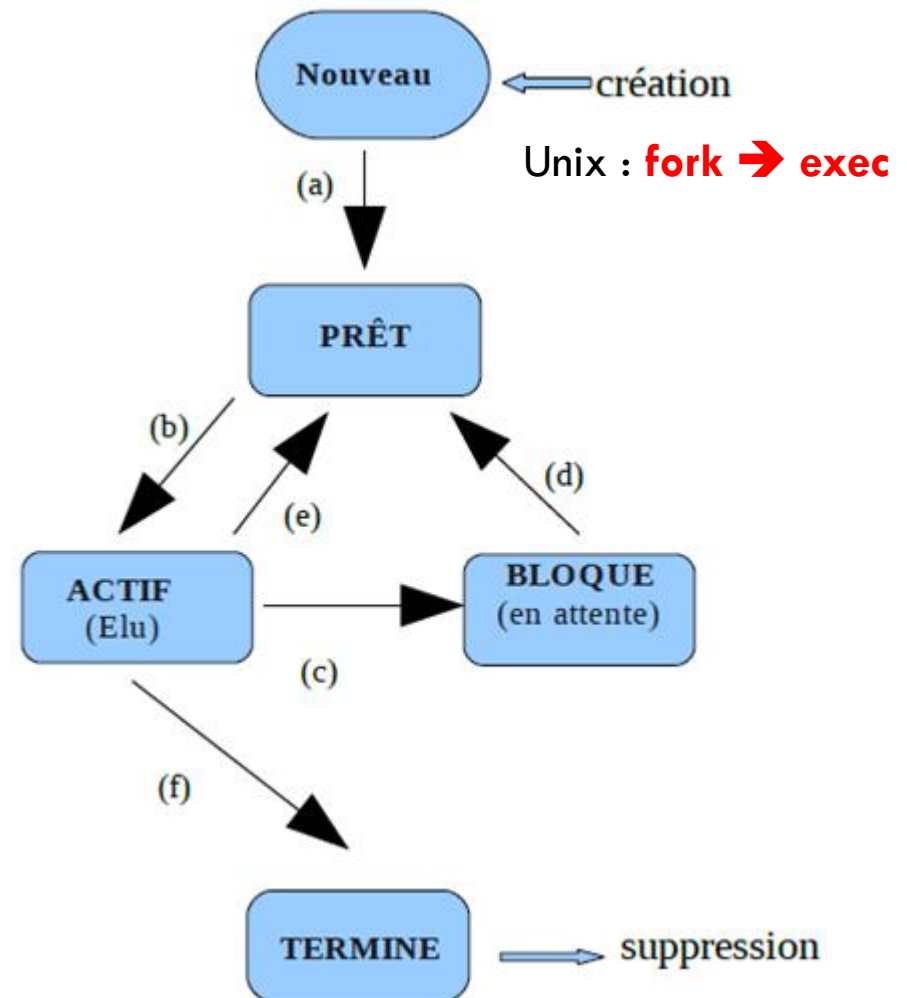
La **préemption** est la possibilité qu'a le système de reprendre une ressource (p.ex. le processeur) à un processus sans que celui-ci ait libéré cette ressource.

Processus lourds

□ Ordonnancement avec préemption (système mono processeur)

▣ Automate d'états d'un processus

- (a) – acceptation
- (b) – nouvelle période d'accès
 - Le processus a été sélectionné par l'ordonnanceur pour avoir accès au processeur
- (c) – blocage par manque de données d'entrée
 - Le processus doit attendre les données requises
- (d) – déblocage du processus
 - Les données en attente sont disponibles
 - Le processus est remis dans la file d'attente des processus à exécuter
- (e) – période d'accès terminée
 - Le processus est remis dans la file d'attente des processus à exécuter
 - l'ordonnanceur choisit un autre processus (**préemption !**)
- (f) – le programme se termine (**exit**) ou est terminé par le processus père (**abort**)



Processus lourds

□ Contexte d'exécution

▣ Ensemble d'informations permettant de connaître l'état courant du processus

- état du processus
- priorité du processus
- valeurs des registres du processeur
- informations sur les zones mémoires (code, donnée, pile, tas)
- informations sur les fichiers ouverts

▣ Commutation de contexte

- Pendant la commutation du processus actif (remplacement du processus en exécution par un autre, choisit parmi les processus PRÊTS) il faut enregistrer tout le contexte de l'ancien processus et charger le contexte du suivant
- Toutes les informations nécessaires à la gestion d'un processus sont stockées dans une structure, appelée **Process Control Block (PCB)**
- Chaque fois qu'un processus devient ACTIF, son PCB permet de reproduire l'état du traitement. Quand il quitte l'état ACTIF son contexte est stocké dans le PCB

(c) ou (e) : passage d'élus à prêt ou bloqué

➔ **Sauvegarde du contexte d'exécution dans le PCB**

(b) : passage de prêt à élu

➔ **Rechargement du contexte d'exécution à partir du PCB**

Reprise du processus dans les conditions exactes dont il disposait au moment de son interruption

Processus lourds

□ Notion de priorité

▣ Possibilité d'affecter une priorité aux processus

- Priorité haute : accès prioritaire au processeur
- Priorité basse : peu d'accès au processeur

▣ Ordonnement avec priorité statique

- Chaque processus a une priorité et on ne lance que les processus de priorité la plus élevée
- En cas de processus de même priorité, on applique la politique du tourniquet
- **Problème : risque de « famine » pour les processus de basse priorité**

▣ Ordonnement avec priorité dynamique

- le scheduler recalcule périodiquement les priorités
 - augmente les priorités des processus pour qu'ils accèdent tous au CPU
- calculées à partir de la priorité de base et du temps processeur consommé réellement
- appliquée dans les systèmes UNIX et Windows

Processus lourds

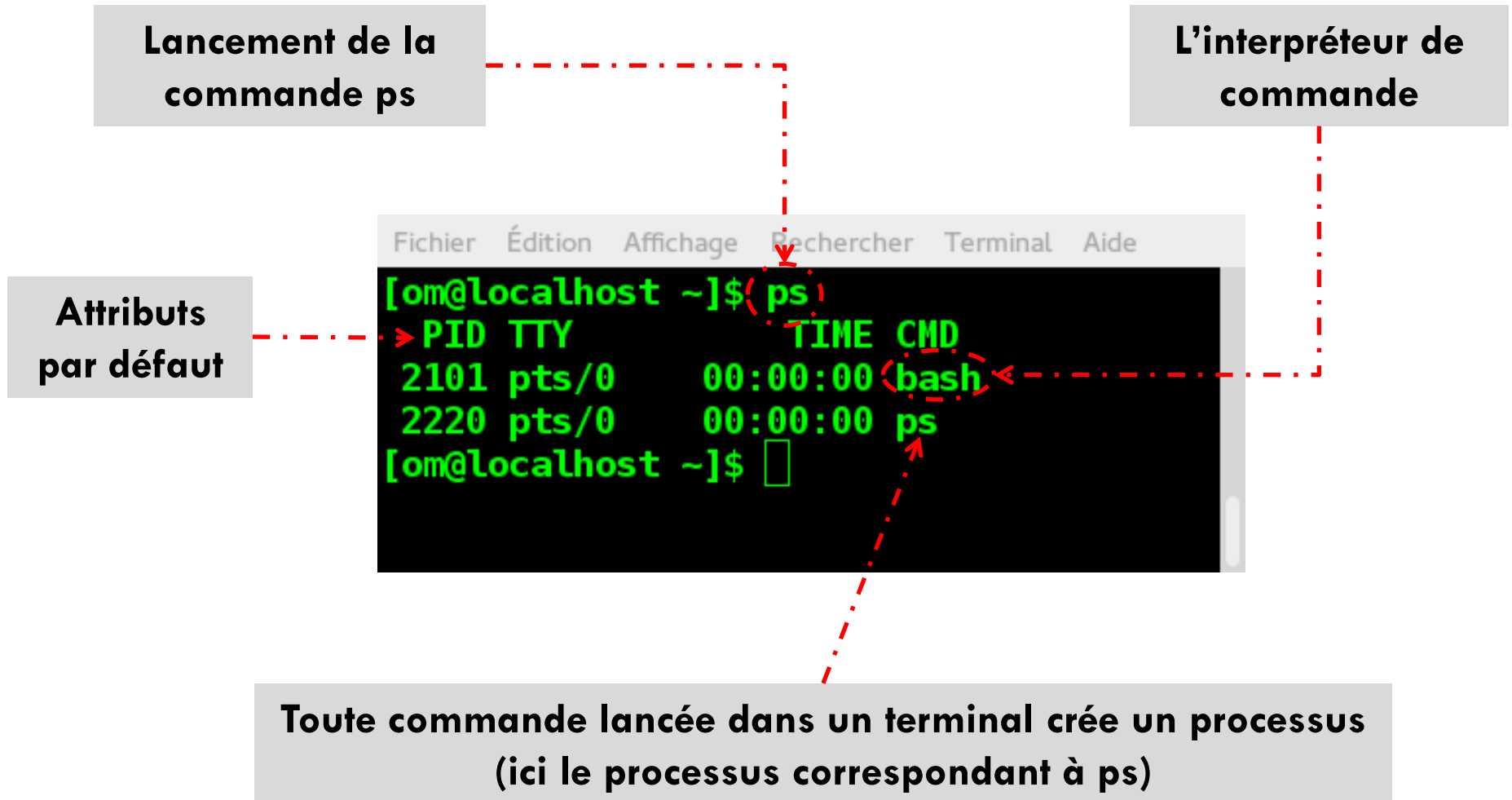
□ Attributs d'un processus

- Dans Unix (et Linux) chaque processus est identifié par un nombre entier unique, le **PID(Process Identifier)**
- Dans un système il peut fonctionner des dizaines, voir des centaines de processus système et utilisateurs
- Le seul moyen de créer un nouveau processus et l'utilisation de la primitive **fork**
- Le processus ainsi créé est appelé **processus fils**, et le processus qui l'a créé –**processus père**
- Au démarrage du système, on démarre le premier processus « **init** »
 - **le parent de tous les autres processus** (arborescence de processus)

Attribut	Signification
PID	Nombre identifiant un processus
PPID	Identité (nombre) du processus père (créateur)
UID	Nom de l'utilisateur qui a lancé le processus
GID	Groupe de l'utilisateur qui a lancé le processus
TTY	Terminal d'attachement
STIME	Date de lancement du processus
TIME	Durée d'utilisation de l'unité centrale
SIZE	Taille de la mémoire allouée
CMD	Nom du processus
C	Priorité du processus (entre -20 et +20)

Commandes Unix & processus

- Quelques commandes utiles pour la gestion des processus sous Unix
 - ▣ **ps** : permet d'afficher la liste des processus lancés depuis le terminal



Commandes Unix & processus

Le lancement d'une commande bloque le terminal jusqu'à la fin de celle-ci

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2220 pts/0        00:00:00 ps
[om@localhost ~]$ gedit
```

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2390 pts/0        00:00:00 ps
[om@localhost ~]$ gedit &
[1] 2395
[om@localhost ~]$
```

Lancement d'une commande en fond de tâche (*gedit &*) → le terminal est débloqué

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2390 pts/0        00:00:00 ps
[om@localhost ~]$ gedit &
[1] 2395
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2395 pts/0        00:00:02 gedit
 2408 pts/0        00:00:00 ps
[om@localhost ~]$
```

Affichage du PID
(*gedit &* : 2395)

Commandes Unix & processus

Arrêt du processus depuis
le terminal : **Ctrl-C**

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2431 pts/0        00:00:00 ps
[om@localhost ~]$ gedit
^C
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2442 pts/0        00:00:00 ps
[om@localhost ~]$
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2454 pts/0        00:00:00 ps
[om@localhost ~]$ gedit
^Z
[I]+  Stoppé
[om@localhost ~]$ bg
[1]+  gedit &
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2458 pts/0        00:00:01 gedit
 2468 pts/0        00:00:00 ps
[om@localhost ~]$
```

Ctrl-Z : le processus est stoppé

bg : le processus stoppé est
relancé en fond de tâche

Commandes Unix & processus

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost ~]$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:03 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 rcu_sched
    8 ?        00:00:00 rcuos/0
```

Option -e : affichage de tous les processus en cours d'exécution

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
2228 ?        00:00:00 kworker/u8:0
2259 ?        00:00:00 systemd
2266 ?        00:00:00 (sd-pam)
2298 ?        00:00:00 kworker/0:1
2430 ?        00:00:00 kworker/2:0
2458 pts/0      00:00:02 gedit
2488 pts/0      00:00:00 ps
[om@localhost ~]$
```

Option -f : affichage d'attributs étendu

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[om@localhost ~]$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
om           2101   2097  0 00:00 pts/0      00:00:00 bash
om           2458   2101  0 00:05 pts/0      00:00:02 gedit
om           2518   2101  0 00:10 pts/0      00:00:00 ps -f
[om@localhost ~]$
```

Commandes Unix & processus

```
Fichier  Édition  Affichage  Recherche  Terminal  Aide
[om@localhost ~]$ gedit &
[1] 2547
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2547 pts/0        00:00:01 gedit
 2554 pts/0        00:00:00 ps
[om@localhost ~]$
[om@localhost ~]$ kill -9 2547
[1]+  Processus arrêté          gedit
[om@localhost ~]$ ps
  PID TTY          TIME CMD
 2101 pts/0        00:00:00 bash
 2566 pts/0        00:00:00 ps
[om@localhost ~]$
```

Commande **kill** : « tuer » un processus

□ Autres commandes utiles

- **nice** : permet de préciser la priorité à accorder au processus
- **time** : permet de récupérer, à la fin du processus, son temps total d'utilisation du processeur
- **top** : permet d'afficher en continu le classement des processus, en fonction de leur pourcentage d'utilisation du processeur

Les THREADS (processus légers)

□ Inconvénients du processus classique

□ Changement de contexte long

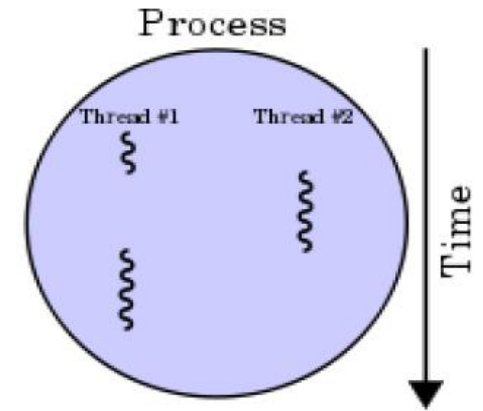
- 90% de ce temps est consacré à la gestion de la mémoire

□ Pas de partage de mémoire

- communications inter processus (IPC) lentes

□ Manque d'outils de synchronisation

□ Interface rudimentaire (fork, exec, exit, wait)



Un processus avec deux Threads

□ On introduit une nouvelle forme de processus ➔ **thread (processus léger)**

□ Les threads partagent la mémoire

- ➔ ainsi on résout le problème du changement de contexte « long »

□ Chaque thread est créé dans le contexte d'un processus

- On peut dire que le processus classique est un processus à un thread

□ **Tous les threads du même processus partagent** la même mémoire (variables globales), les fichiers ouverts, les sockets, ...

- leur contexte "privé" étant limité aux registres (compteur ordinal, PSW, SP, reg. généraux...) et à leur propre pile pour stocker les variables locales

□ **Attention** : le changement de contexte court et le partage de mémoire ne sont vrais qu'entre threads **créés dans le même processus** !

Les THREADS (processus légers)

□ Synchronisation des processus (multitâches)

▣ Exemple d'erreur d'accès sur une variable partagée (V)

Thread A	Thread B
1A: Lire variable V	1B: Lire variable V
2B: Add 1 à la variable V	2B: Add 1 à la variable V
3A: Écrire la variable V	3A: Écrire la variable V

- Dans cette illustration, **le résultat n'est pas prévisible**, du fait qu'on ne peut pas savoir quand les Threads A et B s'exécutent
- **Exécution parallèle** → différents entrelacements possibles
 - Si l'instruction 1B est exécutée entre 1A et 3A, ou si l'instruction 1A est exécutée entre 1B et 3B, **le programme produira des données incorrectes**
 - → **Besoin de synchronisation pour éviter ce genre de problèmes...**

□ Définitions

- ▣ **Section critique** : partie de programme dont **le résultat est imprévisible** si les variables utilisées peuvent être modifiées par des processus parallèles
- ▣ **Exclusion mutuelle** : méthode pour qu'il n'y ait qu'un seul processus (*thread*) dans une section critique donnée (différents algorithmes et solutions matérielles)
 - Une des méthodes – **utilisation d'un sémaphore binaire**

Les THREADS (processus légers)

□ Sémaphore (SEM)

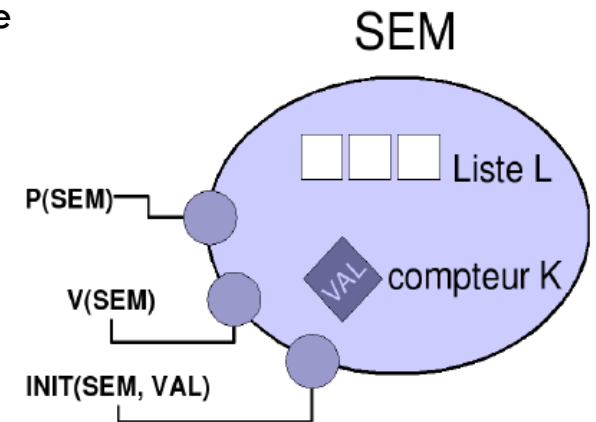
- Un sémaphore est une variable (structure) protégée et constitue une des méthodes pour restreindre l'accès à des ressources partagées (par exemple un espace de stockage, imprimantes, ...) dans un environnement de programmation concurrente
 - Les sémaphores nécessitent une implémentation matérielle (au niveau du microprocesseur), permettant de tester et modifier la variable protégée au cours d'un cycle insécable

- Un sémaphore est constitué de

- **un compteur K**
 - nombre d'unités de ressource disponibles (libres)
- **une file (ou liste) d'attente L**
 - file des processus en attente de la ressource, (ex. **FIFO**)

- Les trois opérations définies pour un sémaphore sont

- **INIT(SEM, VAL) → $K := VAL$** , exécutée une seule fois au début pour initialiser le sémaphore (nombre d'unités initialement disponibles)
 - **P(SEM)** : l'opération **P** est en attente jusqu'à ce qu'une ressource soit disponible, ressource qui sera immédiatement allouée au processus courant
 - **V(SEM)** : **V** est l'opération inverse; elle rend simplement une ressource disponible à nouveau après que le processus a terminé de l'utiliser
- ➔ **Les opérations P et V sont INDIVISIBLES !**



Les THREADS (processus légers)

□ Sémantique des opérations $P(SEM)$ et $V(SEM)$

$P(SEM)$ (<i>Proberen</i> - tester, <i>Puis-je ? ...</i>)	$V(SEM)$ (<i>Verhoren</i> – incrémenter, <i>Vas-y ! ...</i>)
<ul style="list-style-type: none">• $K := K - 1$;• si $K < 0$ alors<ul style="list-style-type: none">• $\text{état}(r) := \text{bloqué}$; /* $P(SEM)$ exécutée par le processus r */• mettre le processus r dans la file d'attente L /* ré-ordonnancement */• fin si	<ul style="list-style-type: none">• $K := K + 1$;• si $K \leq 0$ alors<ul style="list-style-type: none">• sortir un processus de la liste L ; /* soit q son nom */• $\text{état}(q) := \text{prêt}$; /*ré-ordonnancement */• fin si

$K \geq 0$ veut dire : "il y a K unités de la ressource disponibles"
 $K \leq 0$ veut dire : "il y a $\text{abs}(K)$ processus dans la file d'attente L "

□ Sémaphore binaire (*mutex*)

□ Définition : un sémaphore *binaire* est un sémaphore avec $VAL = 1$

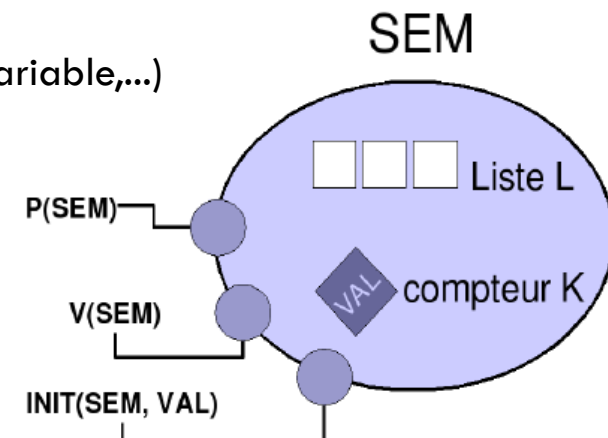
■ $\text{INIT}(SEM, 1) \rightarrow$ **une seule ressource disponible** (une imprimante, une variable,...)

□ Solution du problème de l'exclusion mutuelle \rightarrow **utiliser un mutex**

■ $P(\text{mutex})$;

■ section critique;

■ $V(\text{mutex})$;



Les THREADS (processus légers)

□ Principales fonctions de manipulation de threads

Nom de la fonction (POSIX)	Rôle de la fonction
pthread_create (...)	Création d'un thread
pthread_exit (...)	Terminaison d'un thread
pthread_self (...)	Récupération de l'identité d'un thread
pthread_join (...)	Attente de la fin d'un thread
pthread_mutex_init(...)	Initialisation d'un <i>mutex</i>
pthread_mutex_lock(...)	Verrouillage d'un <i>mutex</i>
pthread_mutex_unlock(...)	Déverrouillage d'un <i>mutex</i>

Les THREADS (processus légers)

- **Exemple** (pour compiler : **gcc -lpthread -o test test.c**)

```
test.c

#include <stdio.h>
#include <pthread.h>

int cpt = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

Void *incr(void *arg) {
    int i;
    printf("[%d]: adr de i = %p, adr de cpt = %p\n", (int)pthread_self(), &i, &cpt);
    for(i=0; i < 1000000; i++) {
        if ( i % 10000 == 0) { printf("[%d]: i = %d\n", (int)pthread_self(), i); }
        pthread_mutex_lock(&mutex);
        cpt++;
        pthread_mutex_unlock(&mutex);
    }
    printf("[%d]: FIN\n", (int)pthread_self()); return NULL;
}

int main() {
    pthread_t tid[2];
    pthread_create(&tid[0], NULL, incr, NULL);
    pthread_create(&tid[1], NULL, incr, NULL);
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    printf("cpt = %d", cpt);
    return 0;
}
```

Plan

- Introduction
- Notion de SGF (Système de Gestion de Fichier)
- Interpréteur de commandes
- Implantation des systèmes de gestion de fichiers
- Notion de processus
- **Systèmes de gestion de la mémoire**
 - ▣ **Mémoire virtuelle et segmentation**
 - ▣ **Mémoire virtuelle et pagination**
- Les scripts shell

Gestion de la mémoire

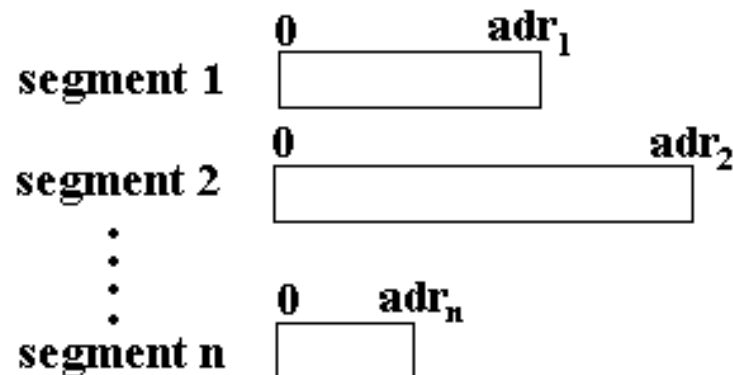
□ Gestion de la mémoire par un OS de multiprogrammation

- Dans un tel OS, **plusieurs tâches sont présentes en mémoire centrale**, à un instant donné
 - il faut donc que chacune dispose d'un espace mémoire qui lui est propre et qui soit protégé de toute interaction avec une autre tâche.
 - Il est donc nécessaire de partitionner la mémoire centrale MC en plusieurs sous-ensembles indépendants.
- **Plusieurs tâches s'exécutant en mémoire centrale utilisent généralement plus d'espace mémoire que n'en contient physiquement la mémoire centrale MC**
 - il est alors indispensable de mettre en place un mécanisme qui allouera le maximum d'espace mémoire physique utile à une tâche et qui libérera cet espace dès que la tâche sera suspendue.
- **Problème : comment allouer/dés-allouer la mémoire physique dans la MC ?**
 - → 2 solutions possibles : **segmentation de la mémoire, pagination de la mémoire**
- On désigne par **mémoire virtuelle**, une méthode de gestion de la mémoire physique permettant de faire exécuter une tâche dans un espace mémoire plus grand que celui de la mémoire centrale (MC)
- Exemple (Windows/Linux)
 - Un processus fixé se voit alloué un espace mémoire de 4 Go, si la mémoire centrale physique possède une taille de 512 Mo, le mécanisme de **mémoire virtuelle** permet de ne mettre à un instant donné dans les 512 Mo de la MC, que les éléments strictement nécessaires à l'exécution du processus, les autres éléments restant stockés sur le disque dur, prêts à être ramenés en MC à la demande.

Mémoire virtuelle et segmentation

□ Segmentation de la mémoire

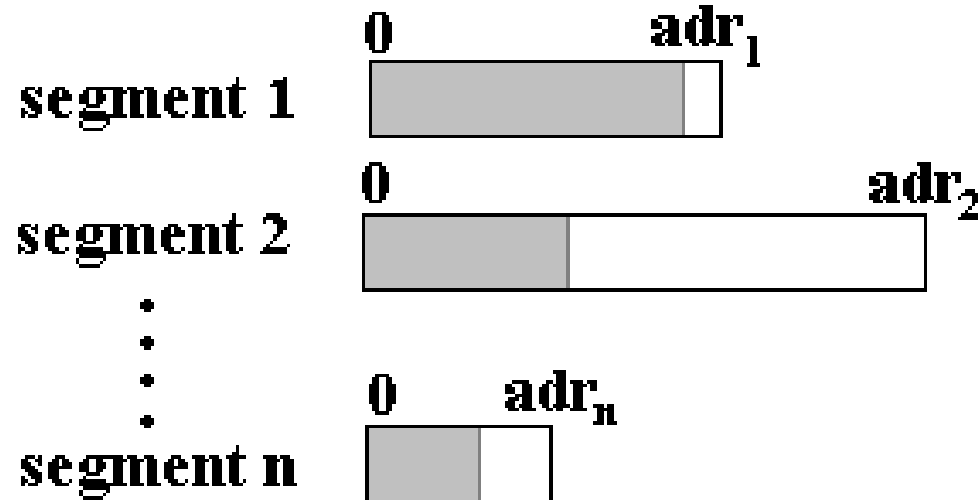
- ▣ **Un segment de mémoire est un ensemble de cellules (ou blocs) mémoires contiguës**
- ▣ Le nombre de cellules d'un segment est appelé la taille du segment
 - ce nombre n'est pas nécessairement le même pour chaque segment
 - chaque segment ne doit pas dépasser une taille maximale fixée
- ▣ La première cellule d'un segment a pour adresse 0
- ▣ la dernière cellule d'un segment adr_k est bornée par la taille maximale autorisée pour un segment
- ▣ Un segment contient généralement des informations de même type (du code, une pile, une liste, une table, ...)
- ▣ La taille d'un segment peut varier au cours de l'exécution (dans la limite de la taille maximale)
 - par exemple une liste de données contenues dans un segment peut augmenter ou diminuer au cours de l'exécution.



Mémoire virtuelle et segmentation

□ Segmentation de la mémoire

- Généralement, les cellules d'un segment ne sont pas toutes entièrement utilisées

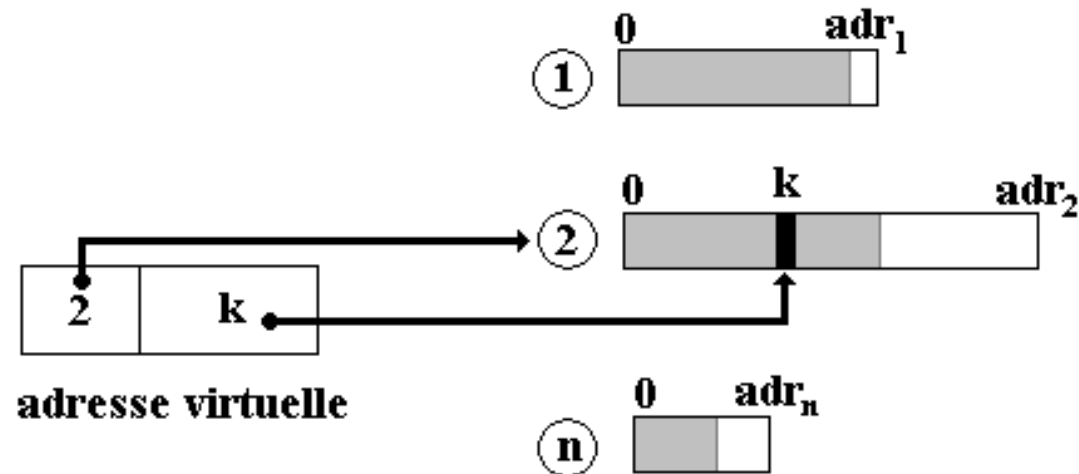


- L'adresse d'une cellule à l'intérieur d'un segment s'appelle **adresse relative** (au segment) ou **déplacement**
 - On utilise plus habituellement la notion d'**adresse logique** permettant d'accéder à une donnée dans un segment, par opposition à l'**adresse physique** qui représente une adresse effective en mémoire centrale.
- Le système de gestion de la mémoire utilise un ensemble de plusieurs segments pour allouer de la place mémoire aux divers processus.
- NB: chaque processus est **segmenté** en un nombre de segments qui dépend du processus lui-même.

Mémoire virtuelle et segmentation

□ Adresse logique ou virtuelle

- Une **adresse logique** aussi nommée **adresse virtuelle** comporte deux parties
 - le **numéro du segment** auquel elle se réfère
 - l'**adresse relative de la cellule mémoire à l'intérieur du segment lui-même**



□ Remarques

- Le nombre de segments présents en MC n'est pas fixe
- La taille effective d'un segment peut varier pendant l'exécution
- Pendant l'exécution de plusieurs processus, la MC est divisée en deux catégories de blocs
 - les blocs de **mémoire libre** (libéré par la suppression d'un segment devenu inutile)
 - les blocs **de mémoire occupée** (par les segments actifs)

Mémoire virtuelle et segmentation

□ Fragmentation mémoire

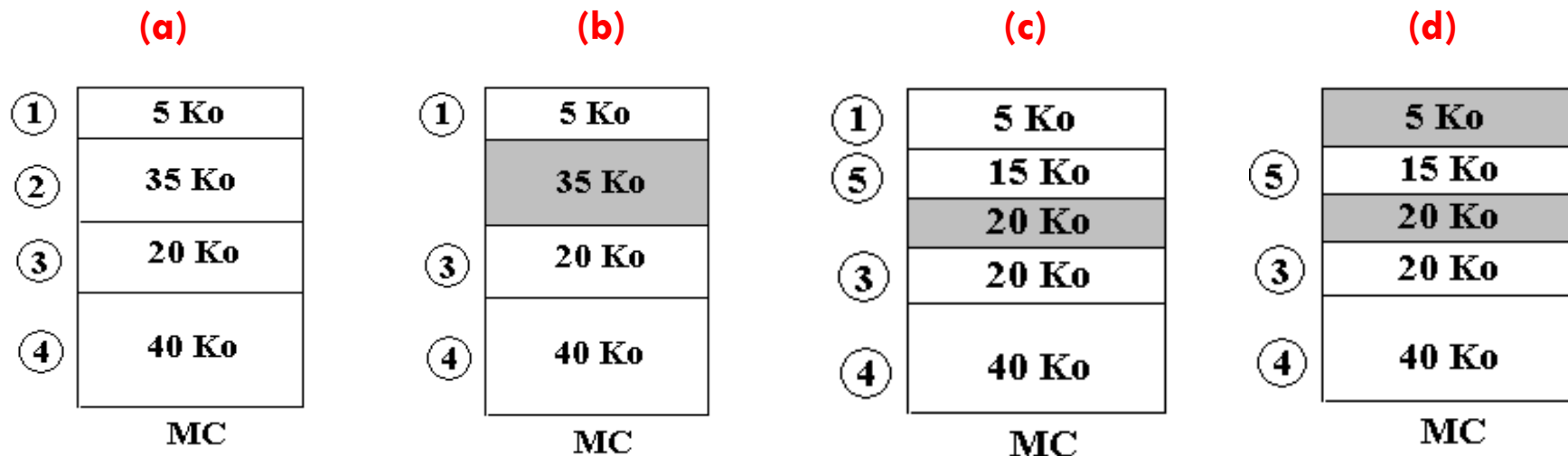
- C'est le partitionnement de la MC entre blocs libres et blocs alloués
 - au bout d'un certain temps, la mémoire contient une multitude de blocs libres qui deviendront de plus en plus petits jusqu'à ce que le système ne puisse plus allouer assez de mémoire contiguë à un processus.
- **Exemple**
 - Soit une MC fictive de 100 Ko segmenté en segments de taille maximale 40 Ko
 - soit un processus P segmenté par le système en 6 segments

Numéro du segment	Taille du segment
1	5 Ko
2	35 Ko
3	20 Ko
4	40 Ko
5	15 Ko
6	23 Ko

Mémoire virtuelle et segmentation

□ Fragmentation mémoire

- (a) Supposons qu'au départ, les segments 1 à 4 sont chargés dans la MC
- (b) Supposons que le segment n°2 devenu inutile soit dés-alloué
- (c) Puis chargeons en MC le segment n°5 de taille 15 Ko dans l'espace libre qui passe de 35 Ko à 20 Ko → **La taille du bloc d'espace libre diminue**
- (d) Continuons l'exécution du processus P en supposant que ce soit maintenant le segment n°1 qui devienne inutile



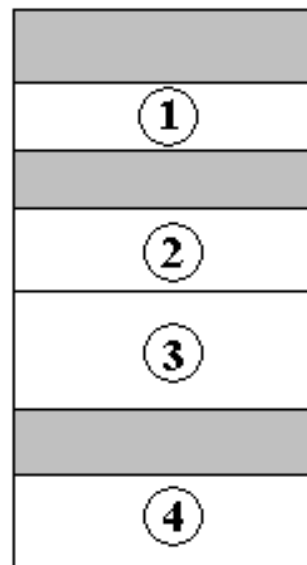
- Il y a maintenant séparation de l'espace libre (fragmentation) en deux blocs, l'un de 5 Ko de mémoire contiguë, l'autre de 20 Ko de mémoire contiguë, soit un total de 25 Ko de mémoire libre.
- Il est toutefois impossible au système de charger le segment n°6 qui occupe 23 Ko de mémoire, car il lui faut 23 Ko de mémoire contiguë.
- → **Le système doit alors procéder à une réorganisation de la mémoire libre afin d'utiliser "au mieux" ces 25 Ko de mémoire libre**

Mémoire virtuelle et segmentation

□ Compactage

- Dans le cas de la gestion de la MC par segmentation pure, un algorithme de compactage est lancé dès que cela s'avère nécessaire afin de ramasser ces fragments de mémoire libre éparpillés et de les regrouper dans un grand bloc de mémoire libre
- On dénomme aussi cette opération de compactage par **ramasse miettes** (ou **garbage collector**)
- Inconvénient : **performances systèmes affaiblies**

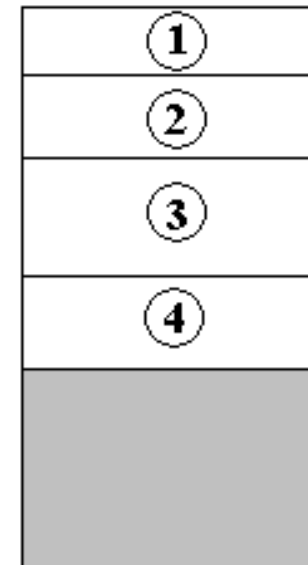
Mémoire fragmentée



MC

Algorithme
de
→
Compactage

Mémoire compactée



MC

Mémoire virtuelle et segmentation

□ Adresse virtuelle - adresse physique

▣ **Problème** : comment le système de gestion de mémoire segmentée retrouve-t-il l'adresse physique associée à une adresse logique d'une donnée ?

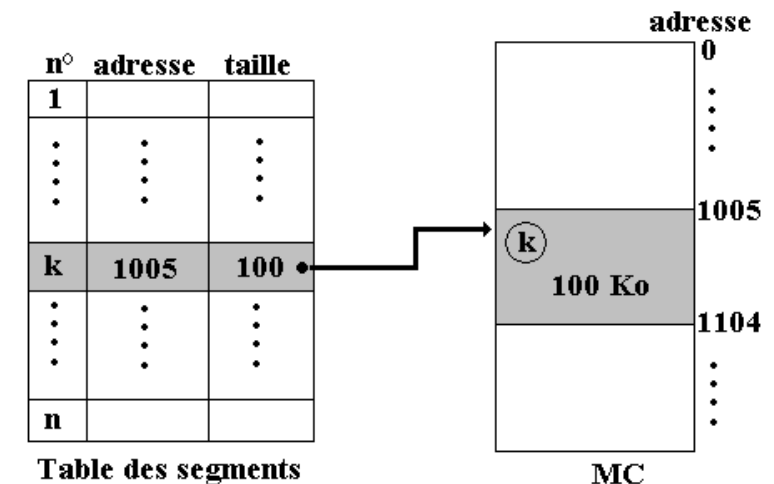
▣ **Solution** : l'OS dispose pour cela d'une **table des segments**, décrivant la carte mémoire de la MC

- la table des segments contient une entrée par segment actif et présent dans la MC
- chaque entrée de la table des segments comporte le **numéro du segment**, l'**adresse physique du segment dans la MC** et la **taille du segment**

n° segment	adresse segment	taille segment
n° segment	adresse segment	taille segment
n° segment	adresse segment	taille segment
⋮		
n° segment	adresse segment	taille segment

▣ **Liaison entre table des segments et le segment lui-même en MC**

- Lorsque le système de gestion mémoire rencontre une **adresse virtuelle de cellule (n° segment, déplacement)**, il cherche dans la table l'entrée associée au numéro de segment, récupère dans cette entrée l'adresse de départ en MC du segment et y ajoute le déplacement de l'adresse virtuelle et obtient ainsi l'adresse physique de la cellule.

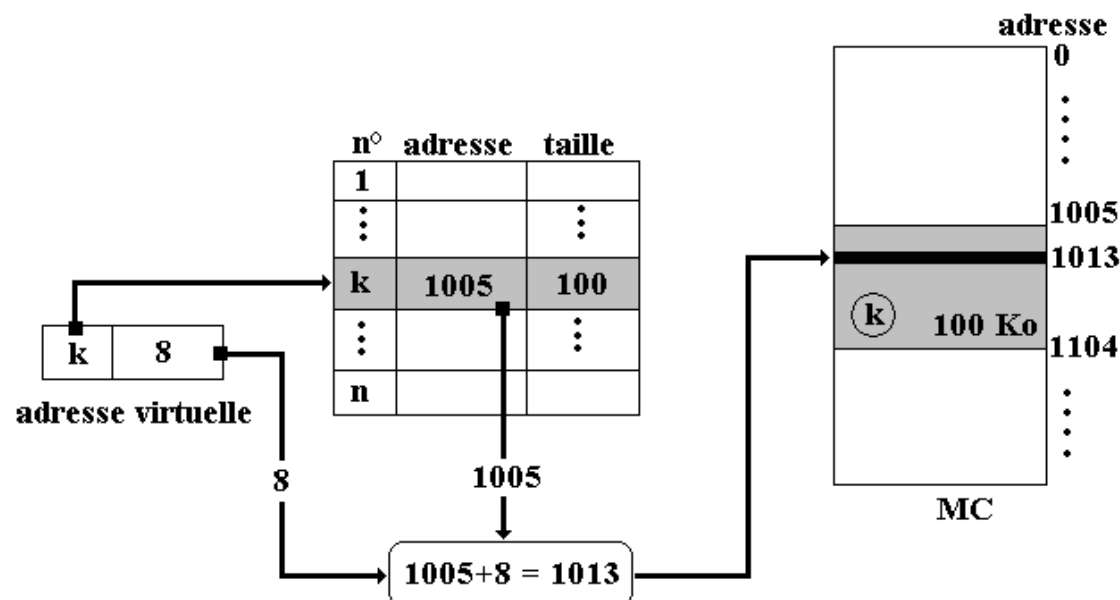


Mémoire virtuelle et segmentation

□ Adresse virtuelle - adresse physique

□ Exemple

- Supposons que nous présentons l'adresse virtuelle (k , 8). Il s'agit de référencer la cellule d'adresse 8 à l'intérieur du segment numéro k.
- Comme le segment n°k est physiquement implanté en MC à partir de l'adresse 1005, la cellule cherchée dans le segment se trouve donc à l'adresse physique $1005+8 = 1013$.
- La figure ci-après illustre **le mécanisme du passage d'une adresse virtuelle vers l' adresse physique à travers la table des segments** sur l'exemple (k , 8).



- La segmentation n'est pas la seule méthode utilisée pour gérer la mémoire virtuelle
 - La **pagination mémoire** est une autre technique (très employée) de gestion de la mémoire virtuelle
 - Les OS actuels emploient un mélange de ces deux techniques

Mémoire virtuelle et pagination

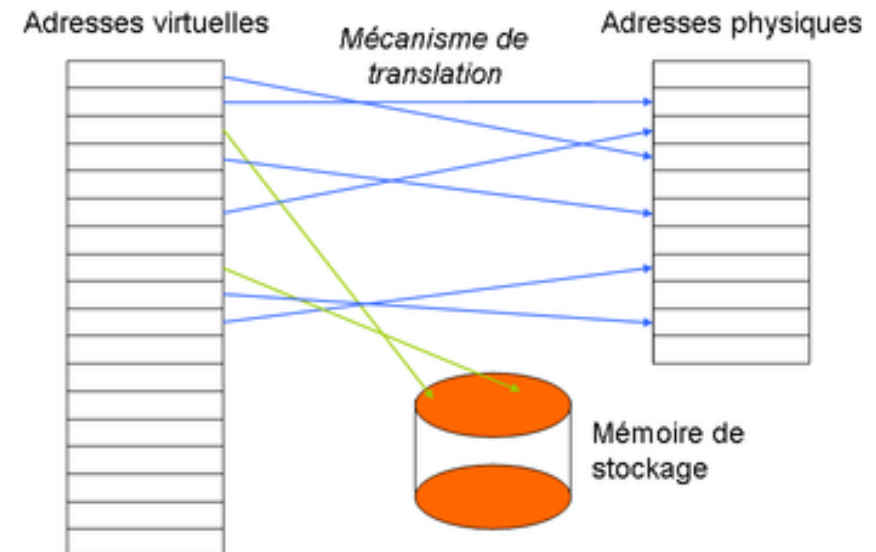
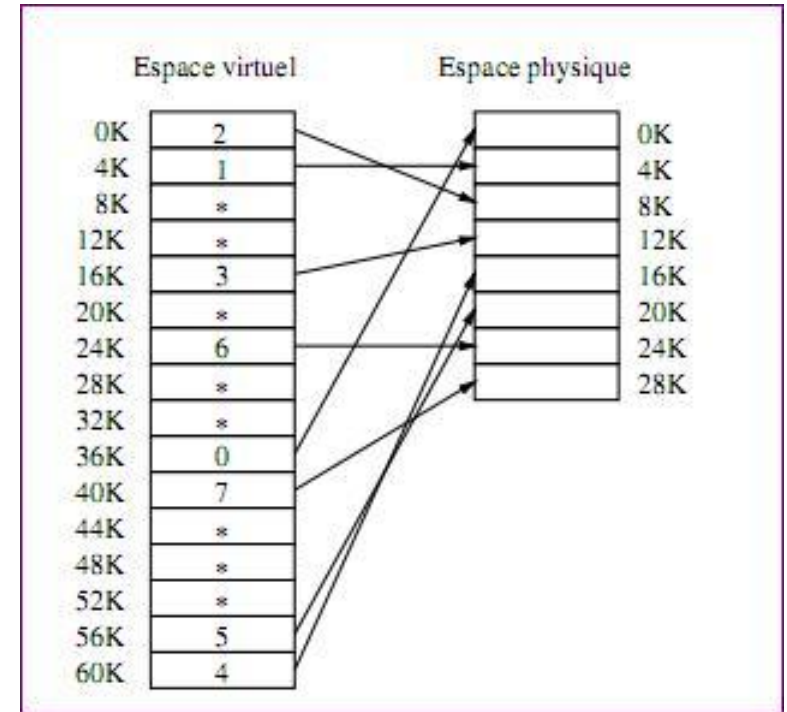
□ Principe de pagination

- Comme dans la segmentation mémoire, la **pagination** est une technique visant à partitionner la mémoire centrale en blocs (nommés ici **cadres de pages**) de taille fixée contrairement aux segments de taille variable.
- **Mémoire virtuelle paginée**
 - plusieurs processus en exécution sont découpés chacun en plusieurs pages nommées **pages virtuelles**
 - **le nombre total de mémoire utilisée par les pages virtuelles de tous les processus, excède généralement le nombre de cadres de pages disponibles dans la MC**
- Le système de gestion de la mémoire virtuelle paginée est chargé de gérer l'allocation et la dés-allocation des pages dans les cadres de pages
 - La MC est divisée en un nombre de cadres de pages fixé par le système
 - généralement la taille d'un cadre de page est une puissance de $2 \leq 64$ Ko
 - La taille d'une page virtuelle est exactement la même que celle d'un cadre de page
 - Le nombre de pages virtuelles est plus grand que le nombre de cadres de pages → **l'espace d'adressage virtuel est plus grand que l'espace d'adressage physique**
 - Seul un certain nombre de pages virtuelles sont présentes en MC à un instant fixé
- **A l'instar de la segmentation, l'adresse virtuelle (logique) d'une donnée dans une page virtuelle, est composée par le numéro d'une page virtuelle et le déplacement dans cette page**
- L'adresse virtuelle est transformée en une adresse physique réelle en MC, par une entité se nommant la **MMU (Memory Management Unit)**

Mémoire virtuelle et pagination

□ Table des pages virtuelles

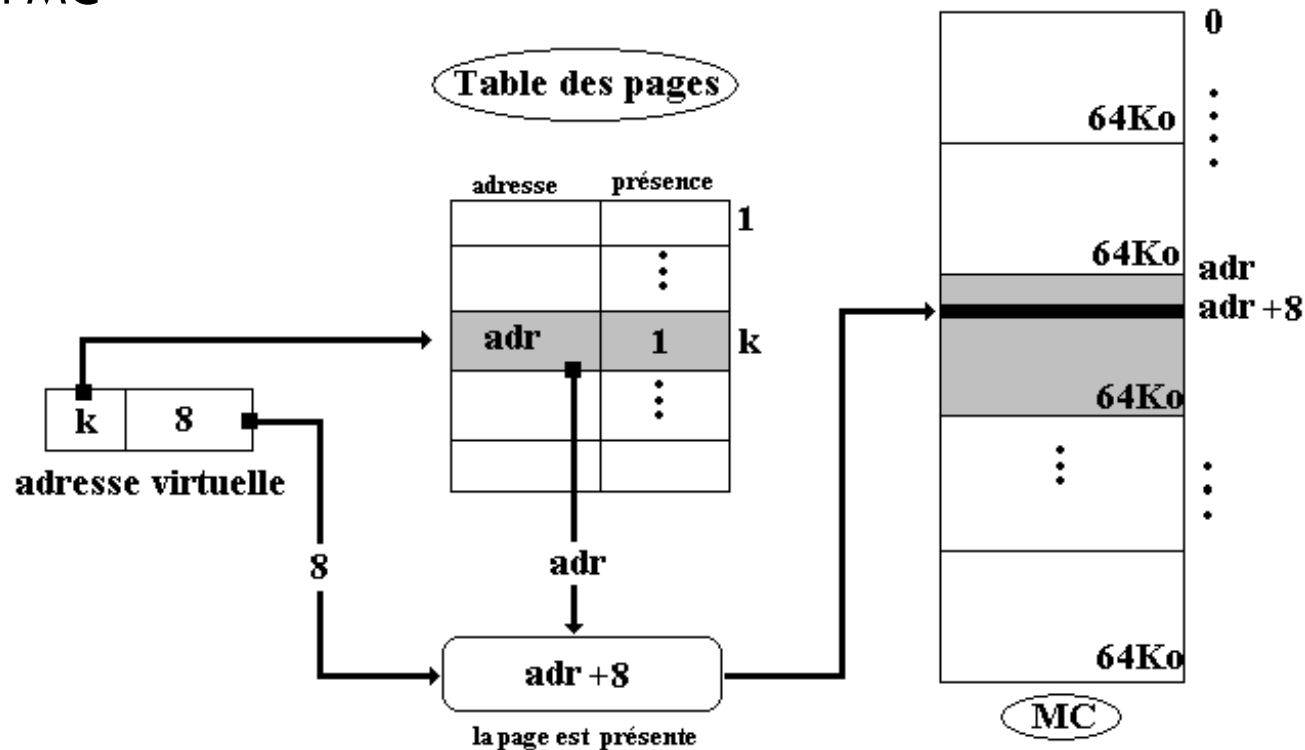
- La table des segments est une liste **ne contenant que les segments présent en MC**
- La table des pages virtuelles quant à elle, est un vrai tableau indicé sur les numéros de pages
 - Le numéro d'une page est l'indice dans la table des pages, d'une cellule contenant les informations permettant d'effectuer la conversion d'une adresse virtuelle en une adresse physique.
- La table des pages doit référencer toutes les pages virtuelles, mais seulement quelques unes d'entre elles sont physiquement présentes en MC
- Chaque page virtuelle se voit attribuer un drapeau de présence
 - **Le bit 0 indique que la page est actuellement absente**
 - **Le bit 1 indique que la page est actuellement présente en MC**



Mémoire virtuelle et pagination

□ Table des pages virtuelles

- Schéma simplifié d'une gestion de MC paginée (page d'une taille de 64Ko) illustrant le même exemple que pour la segmentation, soit accès à une donnée d'adresse 8 dans la page de rang k, le cadre de page en MC ayant pour adresse 1005, la page étant présente en MC



- Lorsque la même demande d'accès à une donnée d'une page a lieu sur une page qui n'est pas présente en MC, **la MMU se doit de la charger en MC pour poursuivre les opérations.**

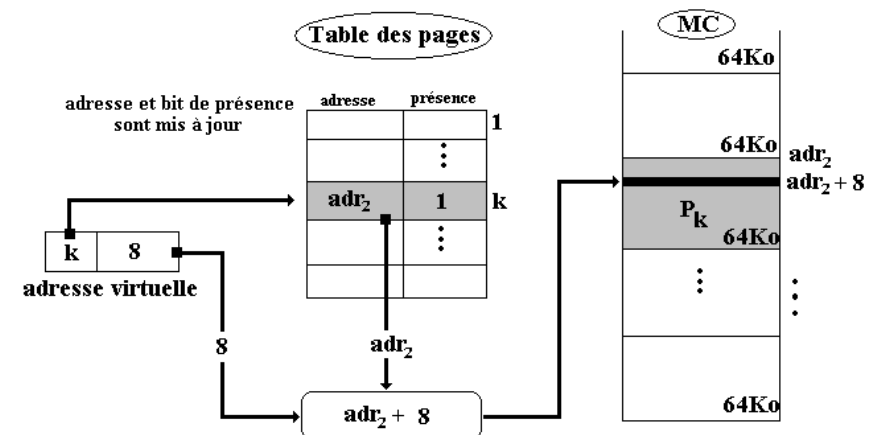
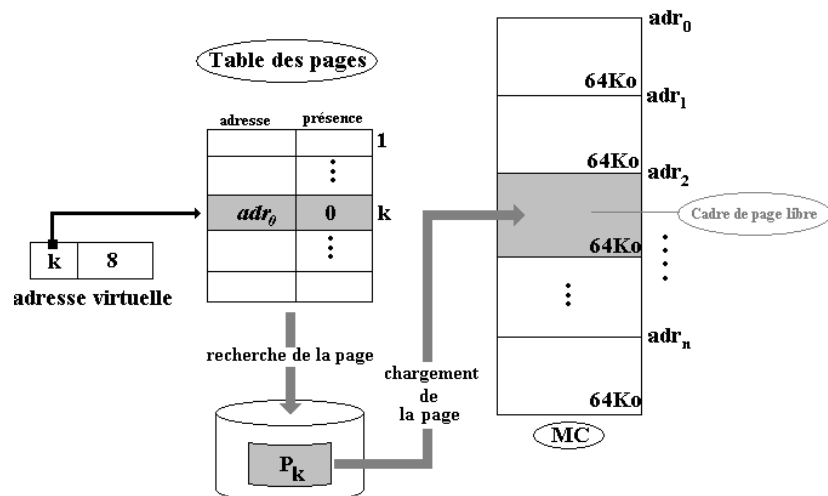
Mémoire virtuelle et pagination

□ Défaut de page

- Nous dirons qu'il y a défaut de page lorsque le processeur envoie une adresse virtuelle localisée dans une page virtuelle dont le bit de présence indique que cette page est absente de la mémoire centrale.
- Dans cette éventualité, le système doit
 - interrompre le processus en cours d'exécution
 - lancer une opération d'entrée-sortie dont l'objectif est de rechercher et trouver un cadre de page libre disponible dans la MC dans lequel il pourra mettre la page virtuelle qui était absente
 - mettre à jour dans la table des pages le bit de présence de cette page et l'adresse de son cadre de page

□ Exemple

- La première figure illustre un défaut de page d'une page P_k qui avait été anciennement chargée dans le cadre d'adresse adr_0 , mais qui est actuellement absente.
- La MMU recherche cette page par exemple sur le disque, recherche un cadre de page libre (ici le bloc d'adresse adr_2 est libre) puis charge la page dans le cadre de page et l'on se retrouve ramené au cas d'une page présente en MC



Mémoire virtuelle et pagination

□ Défaut de page

- En fait, lorsqu'un défaut de page se produit et tous les cadres de pages contiennent des pages qui sont marquées présentes en MC, il faut donc en sacrifier une pour pouvoir caser la nouvelle page demandée.
 - Il est tout à fait possible de choisir aléatoirement un cadre de page, de le sauvegarder sur disque et de l'écraser en MC par le contenu de la nouvelle page.
 - Cette attitude qui consiste à faire systématiquement avant tout chargement d'une nouvelle page une sauvegarde de la page que l'on va écraser, n'est pas optimisée car si la page que l'on sauvegarde est souvent utilisée elle pénalisera plus les performances de l'OS (car il faudra que le système recharge souvent) qu'une page qui est très peu utilisée (qu'on ne rechargera pas souvent).
- Cette recherche d'un "bon" bloc à libérer en MC lors d'un défaut de page est effectuée selon plusieurs algorithmes appelés algorithmes de remplacement. Ces algorithmes diffèrent par la méthode qu'ils emploient pour choisir la page de remplacement (bloc libre) selon sa fréquence d'utilisation ou bien selon le temps écoulé depuis sa dernière utilisation :
 - **NRU** (Not Recently Use)
 - **LRU** (Last Recently Use)
 - **LFU** (Last Frequently Use)
 - **MFU** (Most Frequently Use)
 - **NFU** (Not Frequently Use)
 - **FIFO** (First In First Out)
- L'algorithme **LRU** semble être le plus performant dans le maximum de cas et il est celui qui est le plus utilisé.

Plan

- Introduction
- Notion de SGF (Système de Gestion de Fichier)
- Interpréteur de commandes
- Implantation des systèmes de gestion de fichiers
- Notion de processus
- Systèmes de gestion de la mémoire
- **Les scripts shell**

Les scripts shell

□ Deux modes de fonctionnement pour les shell

- Exécution immédiate des commandes entrées par l'utilisateur
- Lecture et exécution des commandes rédigées dans un fichier texte

□ Notion de script → interprété et non pas compilé

- On oppose l'interprétation à la compilation, dans laquelle le programme **est traduit une fois pour toutes** en langage machine, quel que soit le nombre de ses exécutions (langage C); tandis que le programme interprété **est traduit à la volée** pour chacune de ses exécutions.
- Un script contient
 - Des commandes classiques Unix
 - Des structures de contrôle
 - Des définitions et manipulation de variables, ...→ Équivalent à un langage de programmation

□ Nombreux shells différents (sh, csh, bash, tcsh, ...)

→ Syntaxe et opérateurs souvent spécifiques

□ Scripts bash : shell par défaut sous linux

- est un fichier texte qui doit commencer par la ligne **#!/bin/bash**
 - permet de spécifier **l'interpréteur de commandes** à utiliser
- Le script peut contenir des commentaires, **introduits par #**
- Le fichier contenant le script doit être rendu exécutable : **chmod u+x monscript.bash**

monscript.bash

```
#!/bin/bash
# début du script
```

Les scripts shell : bash

□ Les variables

- Possibilité de définir des variables locales au script
 - **Un nom**
 - **Pas de type** (typage au moment de l'utilisation)
- Définition et/ou affectation d'une valeur : **nom=valeur**
 - Si *nom* existe, sa valeur est modifiée
 - Si *nom* n'existe pas, la variable est créée et initialisée à « valeur »
- Accès à la valeur : **\$nom** ou **\${nom}**
 - Permet d'accéder au contenu de la variable *nom*

```
v="toto"
x=6
echo $v
echo $x
```

toto
6

```
echo $v $x
echo -n $v
echo $x
```

toto6

□ Les variables entières

- Possibilité de définir des variables entières explicitement
 - Syntaxe : **declare -i** *nomVariable1*[=exp arithm] *nomVariable2*[=exp arithm]

```
#!/bin/bash
declare -i x=1 y=32
echo $x $y
```

132

- Les variables peuvent être déclarées comme constantes
 - Syntaxe : **declare -ir** *nomVariable1*[=exp arithm] *nomVariable2*[=exp arithm]

Les scripts shell : bash

□ Les variables d'environnement

▣ Les shells définissent certaines variables communes

- entre plusieurs versions du même shell
- souvent communes entre différentes shells

▣ Objectifs

- Définir les propriétés de l'environnement de l'utilisateur
- Quelques exemples
 - **HOME** : le chemin d'accès au dossier d'accueil de l'utilisateur

`echo $HOME` → */home/om*

- **USERNAME** : le nom de l'utilisateur

`echo $USERNAME` → *om*

- **PATH** : les chemins où rechercher les commandes lancées par l'utilisateur

`echo $PATH` → */usr/bin:/usr/local/bin:.*

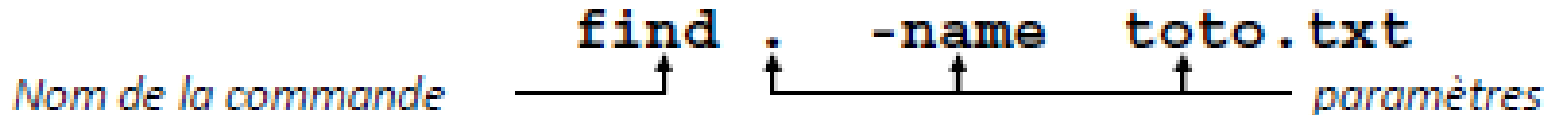
➔ commande **printenv** : affichage de toutes les variables d'environnement

Les scripts shell : bash

□ Les paramètres de la ligne de commande

▣ Principe

Nom de la commande **find** **.** **-name** **toto.txt** *paramètres*



▣ Extension aux scripts

- Un script = une commande
- Possibilité de récupérer les paramètres dans le script

▣ Notations spéciales

- **\$#** : nb de paramètres
- **\$n** : valeur du paramètre n
- **\$*** : liste des paramètres (suite des valeurs des paramètres)
- **\$0** : nom de la commande

▣ Exemple

test.bash

```
#!/bin/bash
echo "Nombre de paramètres : "  $#
echo $*
echo $0
echo $1
echo $4
```



Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost scripts]$ ./test.bash f1.txt 2 7 xyz abc
Nombre de paramètres : 5
f1.txt 2 7 xyz abc
./test.bash
f1.txt
xyz
[om@localhost scripts]$
```

Les scripts shell : bash

□ Écriture

- affichage : la commande **echo**
 - Affiche ce qui suit sur la ligne à l'écran, puis passe à la ligne
 - **Option -n** : reste sur la même ligne à la fin de l'affichage
- protection des écritures
 - Le caractère `\` banalise le caractère suivant
 - Les caractères `" "` banalisent leur contenu, sauf les caractères `\`, `$` et ```
 - Les caractères `' '` banalisent tout le contenu

test.bash

```
#!/bin/bash
i="essai"
echo *
echo \*
echo $i
echo "--$i--"
echo `--$i--`
```

□ Lecture

- Possibilité de saisir des données entrées par l'utilisateur
- **read** : permet de récupérer la prochaine **ligne** de caractères saisie par l'utilisateur

test.bash

```
#!/bin/bash
echo -n "entrez votre nom "
read nom
echo "bonjour " $nom
```



Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost scripts]$ ./test3.bash
entrez votre nom : Moussaoui
bonjour Moussaoui
[om@localhost scripts]$
```

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost scripts]$ ./test2.bash *
test1.bash test2.bash tr1.png
*
essai
--essai--
--$i--
[om@localhost scripts]$
```

Les scripts shell : bash

□ Les opérations arithmétiques

- ▣ Les opérateurs arithmétiques classiques sont disponibles

- `() , + , - , * , /`
- `%`
- `++ , --`

```
#!/bin/bash
(( x = 6 ))
(( y = $x + 2 ))

echo "y=" $y
```

- ▣ Assigner une valeur à une variable numérique : `((var = expression arithmétique))`

□ Les opérateurs booléens

Opérateurs	Opération
<code>==</code>	comparaison
<code>!=</code>	différent de
<code>!</code>	négation
<code>> , < , >= , <=</code>	comparaisons
<code>&&</code>	ET logique
<code> </code>	OU logique

Les scripts shell : bash

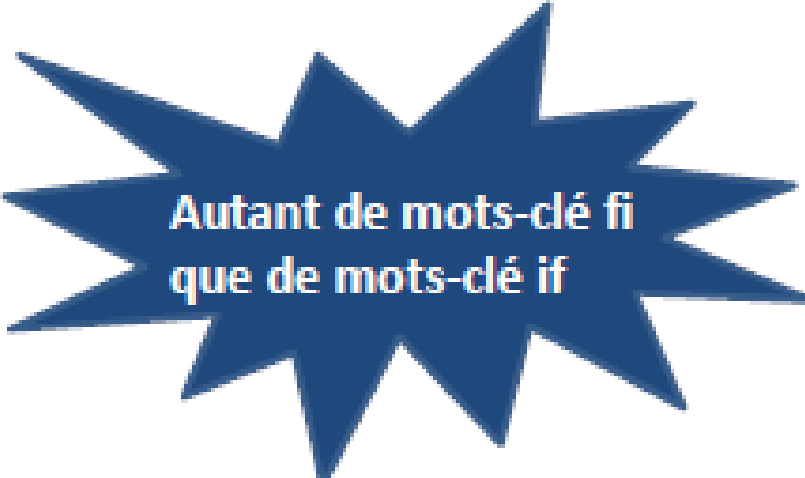
□ L'alternative

▣ Syntaxes

```
if [ exp ]  
then  
    liste_commandes  
fi
```

```
if [ exp ]  
then  
    liste_commandes  
elif [ exp ]  
then  
    liste_commandes  
else  
    ...  
fi
```

```
if [ exp ]  
then  
    liste_commandes  
else  
    liste_commandes  
fi
```



Autant de mots-clé fi
que de mots-clé if

Les scripts shell : bash

□ Les tests

- ▣ Permettent de vérifier des conditions, qui portent
 - Sur des fichiers
 - Sur des chaînes de caractères
 - Sur des expressions arithmétiques
- ▣ Syntaxe : [**test**]
 - **Attention aux espaces de chaque côté du « test » !!!**

```
#!/bin/bash

declare -i i=0
if [ $i == 2 ]
then
    ...
fi
```

□ Les tests sur des fichiers

Test	Description
-e fichier	vrai si le fichier existe
-r fichier	vrai si le fichier existe et est accessible en lecture (R)
-w fichier	vrai si le fichier existe et est accessible en écriture (W)
-x fichier	vrai si le fichier existe et est exécutable (X)
-f fichier	vrai si le fichier existe et est un fichier régulier
-d fichier	vrai si le fichier existe et est un répertoire
-s fichier	vrai si le fichier existe et n'est pas vide

```
#!/bin/bash

if [ -r "/etc/passwd" ]
then
    echo "accès autorisé"
fi
```

Les scripts shell : bash

□ Les tests sur des chaînes

Test	Description
-z chaîne	vrai si la chaîne est vide
-n chaîne	vrai si la chaîne n'est pas vide
=	les chaînes comparées sont identiques
!=	les chaînes comparées sont différentes

```
#!/bin/bash

if [ $1 = $2 ]
then
    echo "chaînes égales"
else
    echo "chaînes différentes"
fi
```



```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide

[om@localhost scripts]$ ./test4.bash abc defgh
chaînes différentes
[om@localhost scripts]$ ./test4.bash abc abc
chaînes égales
[om@localhost scripts]$
```

□ Les tests sur des nombres

Test	Description
-eq	vrai si les deux nombres sont égaux
-ne	vrai si les deux nombres sont différents
-lt	vrai si nombre1 < nombre2
-gt	vrai si nombre1 > nombre2
-le	vrai si nombre1 <= nombre2
-ge	vrai si nombre1 >= nombre2

```
#!/bin/bash

declare -i i=0, j=3
if [ $i -eq $j ]
then
    ...
fi
```

```
#!/bin/bash

declare -i i=0, j=3

if [ ( $i -eq 0 ) -a ($j -ne 0) ]
then
    ...
fi
```

□ Les tests multiples

- **-a** : (and) et logique; **-o** : (or) ou logique; **!** : (not) négation

Les scripts shell : bash

□ La boucle While

▣ Syntaxe

```
while [ exp ]  
do  
    liste_commandes  
done
```

```
#!/bin/bash  
i=0  
while [ $i -lt 4 ]  
do  
    echo $i  
    (( i = $i + 1 ))  
done
```

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost scripts]$ ./test5.bash  
0  
1  
2  
3  
[om@localhost scripts]$
```

□ La boucle for

▣ Syntaxe

```
for index_var in liste  
do  
    liste_commandes  
done
```

```
#!/bin/bash  
for i in un deux trois  
do  
    echo $i  
done
```

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost scripts]$ ./test6.bash  
un  
deux  
trois  
[om@localhost scripts]$
```

▣ NB : la liste peut être spécifiée en utilisant les méta-caractères

```
#!/bin/bash  
for i in *.txt  
do  
    echo $i  
done
```

Fichier Édition Affichage Rechercher Terminal Aide

```
[om@localhost scripts]$ ./test7.bash  
test.txt  
traces.txt  
[om@localhost scripts]$
```

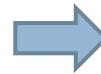
Les scripts shell : bash

□ Interruption de boucles

- ▣ **break** : arrête la boucle
- ▣ **continue** : arrête l'itération courante de la boucle et commence la prochaine itération
- ▣ **exit** : stoppe le script en cours d'exécution
- ▣ **Exemple**

```
#!/bin/bash
somme=0
for i in $#
do
    if [ ($i % 2) == 0 ]
    then
        echo $i " est pair"
        continue
    else
        echo $i " est impair"
    fi
    (( somme = $somme + $i ))
done

echo "somme des arguments impairs = " $somme
```



□ Les choix multiples

```
case valeur in
    expr1) commandes ;;
    expr2) commandes ;;
    ...
esac
```

```
#!/bin/bash
...
echo "oui ou non ?"
read choix
case $choix in
    oui) echo "super !!!" ;;
    non) echo "dommage" ;;
esac
```