Introduction à l'informatique (1)

Définition: *(1965)* «**Science** du traitement [automatique] rationnel de **l'information**, considérée comme le support des connaissances dans les domaines scientifiques, économiques et sociaux, notamment à l'aide de machines automatiques.»

Les termes anglais et français illustrent bien l'évolution dans l'utilisation de ces machines, évolution parallèle à l'émergence de la discipline informatique.

Computer: Historiquement: « Calculateur numérique électronique »
De fait, les premières machines étaient pratiquement exclusivement utilisées pour effectuer des suites d'opérations arithmétiques.
Aujourd'hui, les manipulations arithmétiques ont été généralisées en des manipulations de symboles (**calcul symbolique, traitement de l'information**)

Ordinateur: *(1951)* « Machine électronique de traitement de l'information »
Il est intéressant de remarquer que le sens premier du terme signifie: «ce qui ordonne, met en ordre»

[Petit-Robert, ed. 1996]

Le terme français est mieux adapté à la réalité actuelle, car il s'éloigne de la connotation *exclusivement numérique*.

[Le sens du terme anglais a par ailleurs évolué, et est maintenant strictement équivalent à celui en français]

Utilité

L'ordinateur est une machine à part.

Contrairement à toutes les autres réalisations humaines,
elle n'a pas d'utilité pré-déterminée; son comportement est définit par programme,
et celui-ci peut être modifié à [presque] tout moment

⇨ **automate programmable**.

Dès lors, il est normal que les **domaines**
d'application d'une telle machine soient nombreux et **variés**.

En fait, dans le monde occidental du moins,
l'informatique est présente presque **partout**.

Comme nous allons le voir, nous somme tous, tous les jours et
quasiment en permanence, confrontés aux ordinateurs, visibles ou invisibles.

Trois classes d'applications

S'il est possible d'utiliser un même ordinateur pour des applications très variées,
chaque type d'application possède toutefois des exigences propres,
qui ont conduit à une **spécialisation** des ordinateurs.

On peut mettre en évidence trois grands types d'applications,
desquels on peut dériver trois familles de systèmes informatiques.

- ⇨ le calcul scientifique
- ⇨ la gestion d'informations
- ⇨ la conduite de processus

Calcul scientifique

- **RÔLE:** C'est l'utilisation historique des ordinateurs, héritée de la génération des calculateurs.

- **EXIGENCES:** Puissance de calculs phénoménale, sur des nombres entiers, à virgules flottantes ou les vecteurs.

- **FAMILLES:** La famille des gros calculateurs est extrêmement riche: superordinateurs, super calculateurs, ordinateurs massivement parallèles et ordinateurs vectoriels. (Cray-1, Cray T3D, SV1, ...)

  D'immense bibliothèques de sous-programmes sont utilisées, réalisant un très grand nombre de calculs mathématiques usuels: statistiques, calculs matriciels, transformée de Fourier, calcul intégral et différentiel, ...

- **UTILISATION:** simulation de systèmes complexes (compréhension de fonctionnement, test d'hypothèses, prédiction): climatologie, météorologie, géologie, physique des particules, physique des plasmas, astro-physique, biologie moléculaire, ...

Système d'information

- **RÔLE:** Gestion et traitement des données.

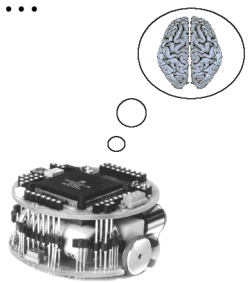
- **EXIGENCES:** importantes capacités de stockage, traitement efficace (rapide et fiable) de gros flux d'informations

- **FAMILLES:** Ordinateurs avec mémoire de masse importante, et fortes capacités en matière de communications (entrées/sorties): ordinateur et mini-ordinateur, serveurs de fichiers, serveurs de données, ... et plus récemment, agenda multifonction de poche (IBM 3090/300, Sun-4, ... Palm, Psion)

- **UTILISATION:** gestions des données utilisées ou produites par les simulations de modèles complexes, gestion de systèmes banquaires ou boursiers, comptabilité d'entreprise, fichiers de polices, serveur vidéo, mais également agenda électronique de poche.

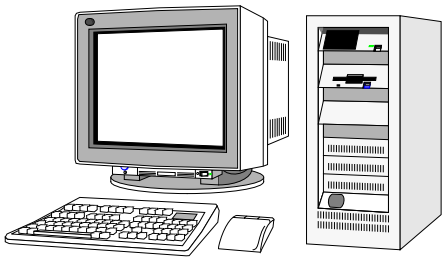
La conduite de processus

- **RÔLE:** L'ordinateur en tant qu'automate de commande (*systèmes embarqués*)
- **EXIGENCES:** Ces équipements sont généralement présents en nombre élevé, d'où la nécessité d'un faible encombrement, d'une consommation réduite, et souvent d'un coût minimum. Par ailleurs, on exige généralement une grande fiabilité (même dans des environnement hostiles) -> tolérances aux pannes, acquisition et traitement des données temps réels, redondance d'équipements critiques transparente....
- **FAMILLES:** Initialement l'ensemble des micro-controleurs, mais on utilise de plus en plus souvent des processeurs, voir des ordinateurs complets (micro-controlleur, mini-ordinateur, composants spécifiques...)
- **UTILISATION:** multitudes d'applications, allant du pilotage/surveillance de processus industriels (chaînes de fabrication, de montage, mais aussi réseaux de distribution d'énergie, centrale atomique) aux fonctionalités de domotique très courante (four micro-ondes, téléphones cellulaires, machines à laver, chronométrage, carburateur de voiture, système de freinage ABS) en passant par l'avionique, la robotique,...



Et l'ordinateur personnel ?

Mais où se trouve l'ordinateur que [la plupart de] nous connaissons bien ?
(l'ordinateur personnel)



On le verra par la suite, avec l'essort de l'informatique domestique,
les différences entres les familles précédemment décrites ont tendance à s'estomper
(mais les architectures restent spécifiques).

L'ordinateur personnel est à la croisée de ces familles,
avec toutefois une prédilection pour le domaine de la gestion des informations
(application bureautique: traitement de texte, tableur, petite base de données,
mais aussi calcul – imagerie – et jeux).

La préhistoire (1)

LA PRÉHISTOIRE:

-500 Les premiers outils de calculs datent de plusieurs milliers d'années: bien avant l'ère Chrétienne,les civilisations méditerranéennes utilisent l'abaque, tandis que le boulier est d'usage en Chine et au Japon.[1]

1614 *Neper* présente sa théorie des logarithmes; les *tables de Neper* permettent de transformer des multiplications compliquées en simple additions.

1620 Invention de la *règle à calcul*, utilisant les tables de Neper. Ce dernier met également au point un système non logarithmique basé sur le déplacement de tiges (connues sous le nom de *Bâtons* ou *Os de Neper*).

1623 *Schickard* utilise le principe de déplacement des tiges pour construire une machine à calculer, constituée vraisemblablement des roues chiffrées. La machine sera malheureusement perdue au cours de la guerre de Trente Ans.

1642 *Pascal* (agé alors de 19 ans) présente la première version de la *Pascaline*, machine permettant d'effectuer des additions et soustractions avec des nombres de 6 chiffres.

1673 *Leibniz* modifie la *Pascaline* en lui ajoutant les multiplications et additions.

1728 *Falcon* construit une commande de métier à tisser à l'aide d'une planchette de bois munies de trous. => Il s'agit de la première machine capable d'exécuter un programme externe.

1805 *Jacquard* perfectionne le système de Falcon, en remplaçant les planches par des cartes perforées.

---

1. Remarquons que les érudits de Babylone utilisaient, en 300 avant J.-C., une numération positionnelle en base 60, incluant le zéro.

Les précurseurs (1)

LES PRÉCURSEURS MODERNES:

1822 Charles Babbage construit un prototype de machine pour le calcul et l'impression de tables numériques nécessaire à la navigation et la balistique. Sa *machine à différences* n'utilise qu'un algorithme (différences finies de polynômes) prédeterminé, mais c'est la restitution des résultats – gravage d'un plateau de cuivre par un timbre en acier – qui est intéressante. Pendant 10 ans, il tentera de construire un modèle utilisable, sans toutefois y parvenir.

1833 Babbage entame une réalisation plus ambitieuse encore, la Machine Analytique, conçue pour effectuer des séquences d'opérations arithmétiques, en fonctions d'instructions données par l'utilisateur. Un autre personnage célèbre participa à l'aventure: Ada Augusta, comtesse de Lovelace et fille de Lord Byron, s'évertua à écrire les programmes nécessaires au fonctionnement de la machine. Malheuseusement, la machine était trop complexe et ambitieuse pour la technologie de l'époque, incapable d'assurer la production des milliers d'engrenages, roues dentées et autres pièces avec une qualité suffisante.

Les précurseurs (2)

1854 Reprennant les spéculations de Leibniz , George Bool publie un essai intitulé Une étude des lois de la pensée, dans lequel il expose ses idées sur la formulation mathématique des propositions logiques.

1936 Publication par Alan Turing de l'essai A propos des nombres calculables, traitant des problèmes théoriquement non solubles (indécidabilité). Préfigurant les caractéristiques de l'ordinateur moderne, il énonce le principe d'une machine universelle, purement imaginaire, appelée depuis Machine de Turing.

1938 Claude Shannon fait le rapprochement entre les nombres binaires, l'algèbre booléenne et les circuits électriques. Dix ans plus tard, il publiera une théorie mathématique de la communication, connue aujourd'hui sous le nom de théorie de l'information. Il prouvera que les nombres binaires conviennent également pour les relations logiques, et que tous les calculs logiques et arithmétiques peuvent être réalisés à l'aide des trois opérations logique de base: ET, OU et NON.

Génération 0

LA GÉNÉRATION ZÉRO: LE RELAIS ÉLECTOMÉCANIQUE *(1930~1945)*

Z1,Z2 Dès 1936, Konrad Zuse fabrique, avec des moyens très modestes, les machines électromécaniques Z1 et Z2, fonctionnant selon le système binaire. Il propose en 1938 la construction d'un calculateur électronique, mais l'Etat allemand juge le projet irréalisable, et refuse le financement. Zuse construit alors un calculateur binaire universel avec 2'600 relais de téléphone, le Z3, achevé en 1941. Les programmes sont introduits au moyen d'un film perforé, et une multiplication dure environ 5 secondes. Le Z3 et son successeur le Z4 seront tout deux utilisés en aéronautique et en balistique.

Mark1 De l'autre côté de l'Atlantique, Howard Aiken réalise pour le compte d'IBM et de l'Université de Harvard, une énorme machine électromécanique, le Mark 1. Réalisé entre 1939 et 1944, ce calculateur était capable de multiplier deux nombres de 23 chiffres décimaux en 6 secondes, et d'effectuer des additions et soustractions en 3 dixièmes de secondes. Avec plusieurs milliers de roulements à billes, et 760'000 pièces électromécaniques, le Mark 1 se révéla obsolète avant même son achèvement.

D'autres chercheurs réalisèrent, pendant cette période, des prototypes de calculateurs; parmis eux, citons encore John Atanasoff (université de l'Iowa) et George Stibitz (Bell Laboratories), qui tout deux adoptèrent le système binaire.

LA 1ÈRE GÉNÉRATION: LE TUBE À VIDE *(1945~1955)*

La guerre qui faisait rage alors précipita l'avènement des ordinateurs:

COLOSSUS Le premier calculateur électronique numérique vit le jour en 1943. Construit dans le plus grand secret par les services spéciaux britanniques, et littéralement porté par Alan Turing, le COLOSSUS fut construit pour permettre le décryptage des messages radios transmis par les forces de l'Axe, et codés au moyen de la fameuse machine ENIGMA.
Placée sous le sceau du secret militaire pendant plus de 30 ans, cette réalisation fut une impasse du point de vue scientifique.

ENIAC Dans l'optique d'établir des tables pour le réglage des tirs d'artillerie, l'armée américaine accepta, en 1943, de financer les traveaux d'Eckert et Mauchly, qui aboutirent, fin 45 début 46, en la réalisation d'une machine pour le moins célèbre: l'eniac.

La guerre étant terminée, Eckert et Mauchly furent autorisés à exposer leur travaux à leurs collègues scientifiques. Il en résultat une multitudes de projets et de machines (JOHNIAC, illiac, maniac, weizac, edvac, l'ias, whirlwind,... et également l'edsac, finalisée en 1949, reprenant les principes énoncés en 45 par John Von Neumann, alors consultant sur le projet eniac, dans lesquels il décrivait ce qui allait désormais être appelé l'architecture de Von Neumann, qui guida la conception des ordinateurs jusqu'à nos jours.[1])

1. L'un des nombreux tours de force de Von Neumann fut d'intégrer, dans la mémoire même de la machine, instructions et données.

Caractéristiques de l'ENIAC

l'ENIAC c'est...

18'000 tubes à vide
1500 relais
6000 commutateurs
30 tonnes
140 KW

Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

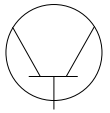
Et en 1957:

7'247 heures de fonctionnement, dont:
3'491 en production
1'061 en «problèmes»
196 d'inactivité
651 d'interventions planifiées
1'848 d'interventions non planifiées
(90% recherche et rempl. de tubes)

19'000 tubes remplacés

Seconde génération

LA 2E GÉNÉRATION: LE TRANSISTOR – *début de l'industrie informatique (1955~1965)*



Le transistor, découvert (inventé) par les laboratoires Bell en 1948, est utilisé dans les ordinateurs, en remplacement des tubes à vide, si encombrants, coûteux et peu fiables.

Les ordinateurs deviennent plus petits, plus performants.

Les mini-ordinateurs apparaissent avec le PDP-1 de DEC; relativement bon marché, ce type d'ordinateurs se vend bien, et ouvre la voie à de nouvelle applications: au MIT, on adjoindra au PDP-1 un écran de visualisation (CRT) de 512x512 points adressables...

... il ne fallut pas longtemps pour que le premier jeu vidéo apparaîsse.

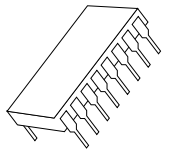
Outre IBM et DEC, de nouvelles firmes se lancent sur le marché: HP, Data Général, ...

Commence alors une course effrénée vers des systèmes toujours plus performants, à des prix de plus en plus compétitifs.[1]

1. Le successeur du PDP-1, le PDP-8, sera vendu 7.5 fois moins chère que son prédécesseur, et à 1000 fois plus d'exemplaires: 50'000 exemplaires à 16'000 $ pour le PDP-8, contre 50 exemplaires à 120'000 $ pour le PDP-1, 5 ans plus tôt.

Troisième génération

LA 3E GÉNÉRATION: LE CIRCUIT INTÉGRÉ *(1965~1980)*



Les microprocesseurs font leur apparition, ce qui accélère encore la chute des prix.

Le taux d'intégration sur les microplaquettes (puces ou chips) passe en peu de temps de quelques composants à **plusieurs milliers**, sur une puce d'une **dizaine de millimètres carrés**.

Les premières familles d'ordinateurs apparaissent (IBM 360) et avec elles le concept de compatibilité descendante (conservation du logiciel).

Autre grande innovation: la multiprogrammation (multi-tâches préemptif).

Les systèmes à avoir le plus marqué cette époque sont:

le PDP-11 de DEC,
le 8080 d'Intel (1er microprocesseur),
le CRAY-1 de Cray (1er superordinateur),
et le VAX de DEC (miniordinateur 32 bits).

Quatrième génération

LA 4E GÉNÉRATION: VLSI – *Essort des ordinateurs personnels (1980~20??)*

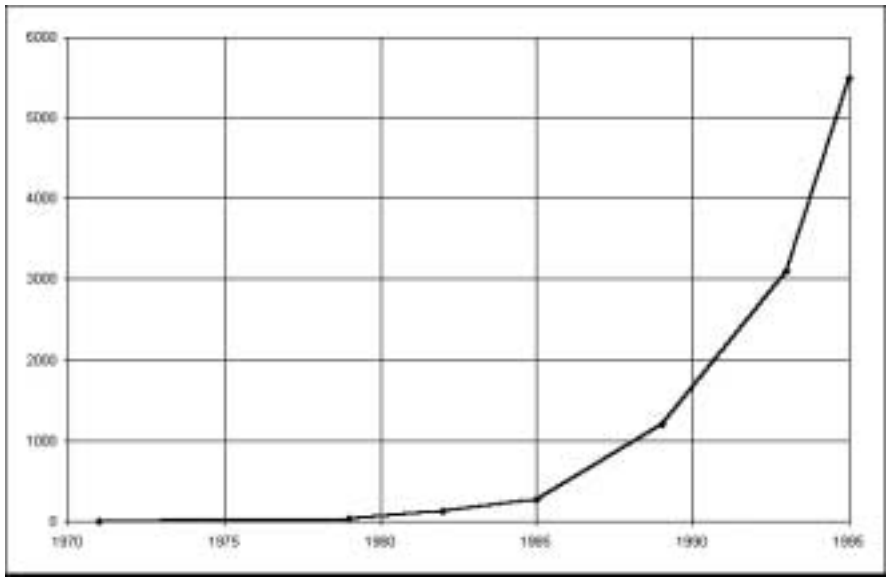
Le Very Large Scale Integration accélère encore le développement,
en rendant possible des intégrations à des taux véritablement phénoménaux,
pour atteindre de nos jours **plusieurs centaines de milliers de composants** sur une même puce.

Cette génération couvre un intervalle énorme de performances et de besoins,
allant des supercomputers massivement parallèles (Paragon, Cray T3D, CM), utilisés
principalement pour le calcul scientifique, aux ordinateurs personnels (PS1, Atari, Mac)
utilisé pour des applications de bureautique, de formation, ... et de jeux.

C'est également cette génération qui verra l'essort d'unité périphériques
en tout genre, et également l'interconnexion des machines en divers architectures réseaux.

Loi de Moore

«Le nombre de transistors intégrés dans une puce doublera tous les 2 ans»[1]



Affirmation faite en 1965 par *Gordon Moore*, cofondateur d'Intel, connue actuellement sous le nom de «*Loi de Moore*», et qui s'est révélée, au cours des 30 dernières années, une méthode de prédiction très précise, la courbe s'étant même infléchie depuis 1992.

*Ce que la loi ne dit pas, c'est que **le volume des investissements** nécessaire pour une telle croissance **suit la même courbe**.*

Cette loi possède de nombreux corollaires, notamment adaptés des lois de Murphy:

«Un microprocesseur est obsolète dès le début de sa fabrication série»

«Le prix d'un ordinateur baisse de 50% le lendemain de son achat»

…

1. Plus exactement, Moore prédit tout d'abord un doublement du nombre de transistors chaque années, puis rectifia en un doublement tous les 18 mois.

Cinquième génération

LA 5E GÉNÉRATION ?

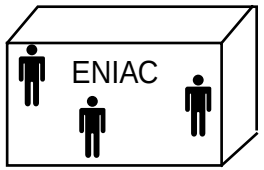
Lancement par les japonais, dans le courant des années 90, d'un grand projet visant à introduire les idées de l'Intelligence Artificielle dans la technologie des ordinateurs.

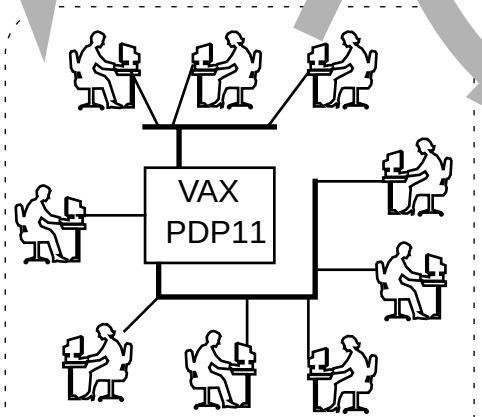
Les résultats n'ont, jusqu'à présents, pas été à la hauteur de leur ambitions, même si des retombées très intéressantes ont vu jour (comme la logique floue).

Américains et Européens ont relevé le défit, essayant de repousser les limites physiques, et d'introduire de nouvelle technologie ou architecture, inspirées par le vivant, dans le but d'accroître capacités et performances, et de rendre l'utilisation des systèmes toujours plus simple et naturelle.

Evolution en images

Années 45~60

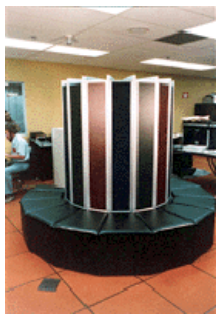




Années 55~80

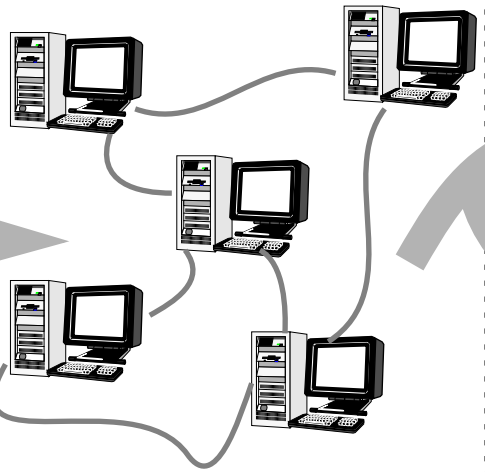
Personal Computers

Années 75~90



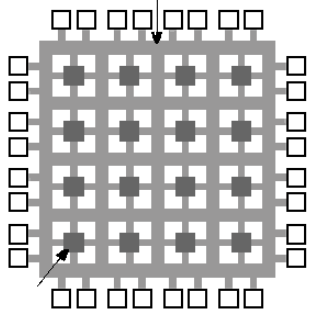
Supers-calculateurs

network computer



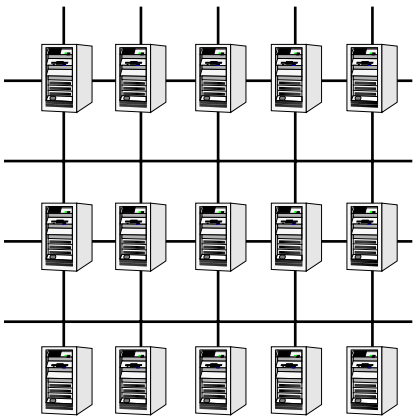
Années 95~00

FPGA: co-proc. configurables
(Hardware programmable)



Années 2000

Cluster de PCs



Conclusion

Bien qu'elle n'existe que depuis un demi-siècle, l'informatique
est une **science** à **part entière**, comportant des aspects tant théoriques que pratiques:

- **THÉORIQUES:**

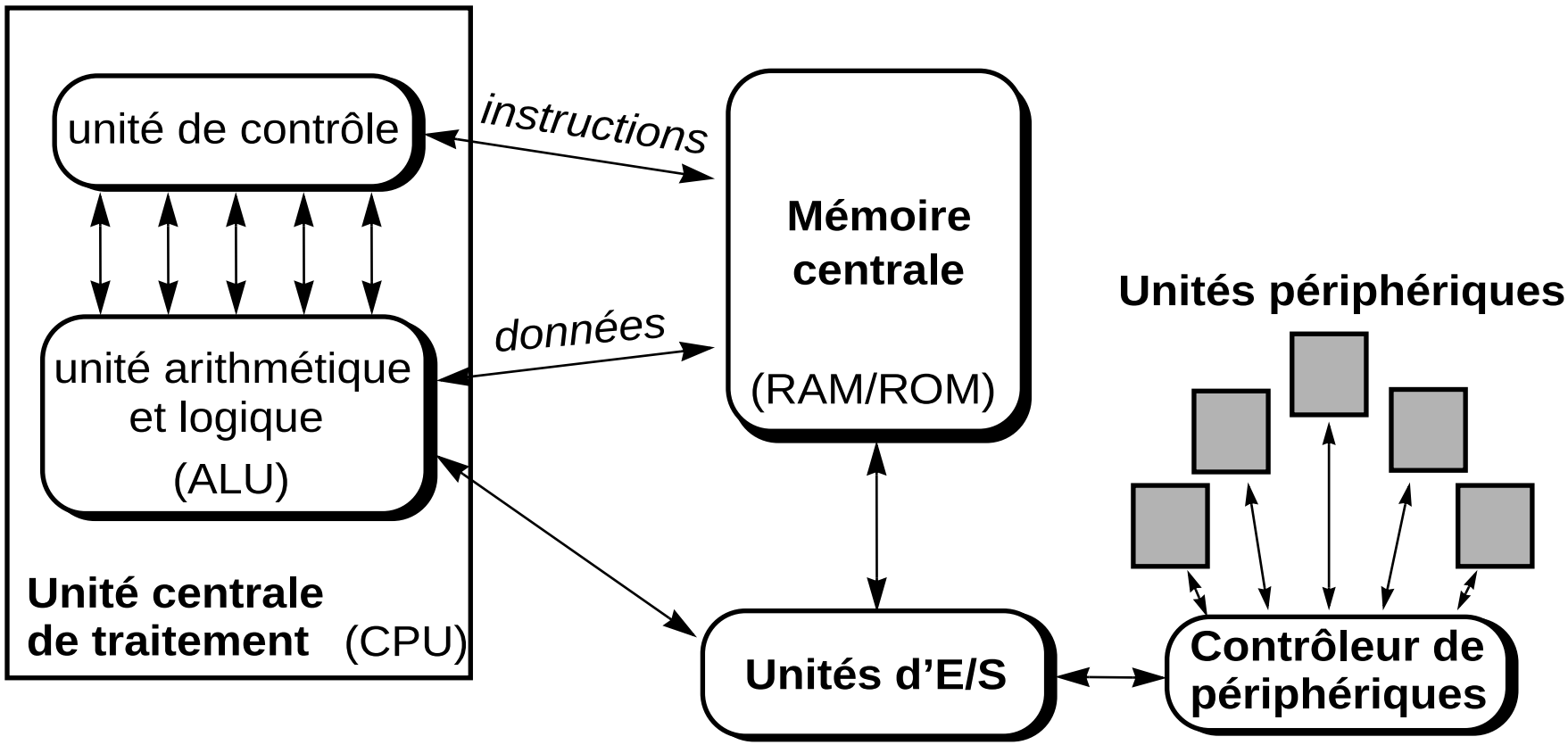
  logique, calculabilité, algorithmique, modélisation, intelligence artificielle,
  théorie de l'information, théorie des automates, sociologie, ...

- **PRATIQUES:**

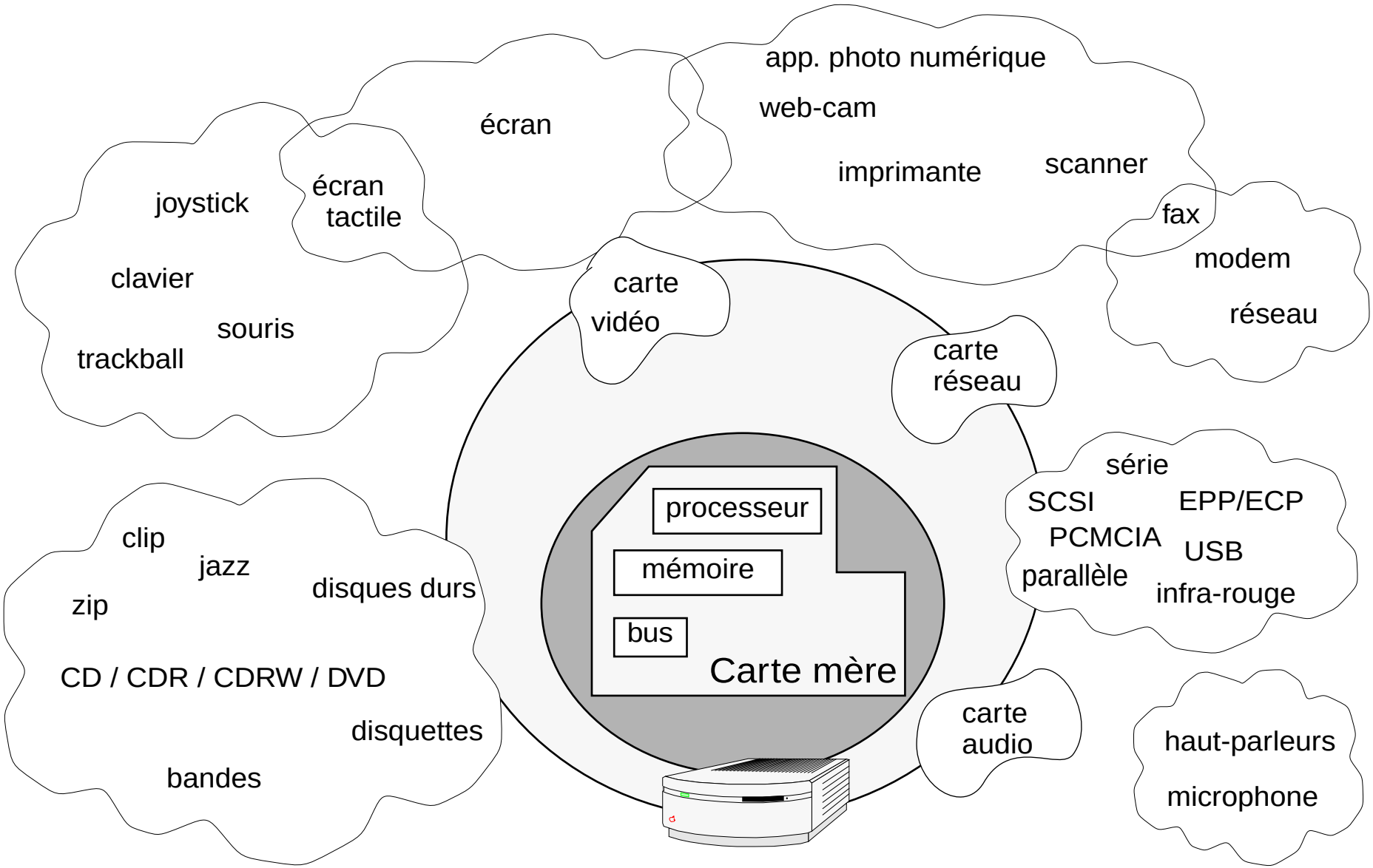
  technologie des composants (électronique, microtechnique, chimie, physqique, ...)
  développement de programmes (algorithmique, ..., mais aussi sociologie)
  gestion de systèmes informatiques (systèmes d'exploitations, réseaux, ...)

L'informatique est donc non-seulement une
mine d'interdisciplinarité, mais également une discipline qui
connaît, aujourd'hui, un **développement** particulièrement **rapide**.

Architecture de Von Neumann



Les périphériques [du PC]



Unité centrale

Processeur: Souvent définit comme étant le «cœur» de l'ordinateur.

→ fabriquant, famille, modèle, packaging
(*Intel/AMD/Cyrix 80X86, Motorola 680X0, AlphaX, slot1, slot7, slotA,...*)

→ fréquence d'horloge interne (*100~1200MHz*)

→ taille de la mémoire cache (*0 ~ 512 Ko, voir plus*)

Mémoire: Assure le stockage à court terme des instructions à exécuter, ainsi que des données.

→ type, packaging (*Simm: EDO/FPM, 30 à 72 pins – Dimm: SDRAM/RDRAM, 168 pins*)

→ capacité (*n x de 8 à 256MB*)

→ temps d'accès (*Simm: 50~90 ns – Dimm: 3~9 ns*)

→ cadence du bus associé (*33~133 MHz*)

Carte mère: Support pour le/s processeur/s, ainsi que la mémoire. C'est elle qui contient les bus.

→ limitations sur le/s processeur/s (*nombre, famille, slot, fréquence, voltage*) et sur la mémoire

→ nature des bus externes et nombre de connecteurs (*AGP ?x, (E)IDE, PCI,...*)

→ taille de la mémoire cache externe (*0 ~256 Ko*)

Boîtier: → format (*desktop, tower, medium-tower, mini-tower*)

Clavier, souris, écran

Clavier: Périphérique de saisie par excellence, tant qu'il est dans la bonne langue.

→ type, nombre de touches (*QWERTY, AZERTY, SuisseRomand, ... 90~115 touches*)

→ connection (*port standard, port PS/2, port USB, clavier sans fil (IR ou radio)*)

Souris: Périphérique permettant le pointage rapide d'éléments.

→ type, nombre de boutons (*Optique, mécanique, trackball, ... de 1 à 4 boutons + 1 roulette)*

→ connection (*port série, port PS/2, port USB, souris sans fil (IR ou radio)*)

Ecran: Périphérique de visualisation

→ technologie (*écran plat, tube trinitron, ...*)

→ surface utilisable, encombrement, poids (*10~22 pouces, 10~40 Kg*)

→ résolution maximale [*640~2400 x 480~1600 pixels*]

→ fréquence de rafraichissement (*60~120 KHz*)

Carte Vidéo: Permet l'interconnexion, en offrant une zone mémoire à accès multiple.

→ taille mémoire (*2~64Mo*) => résolution x couleurs

→ type de connecteur bus (*PCI, AGP x ?*)

→ instructions spécialisée de dessin 2D et/ou 3D

Mémoire de masse

Mémoire périphérique à accès aléatoire:

Disques durs, disquettes, Zip, Jazz, CDs

→ Nature du support et amovibilité (*lecture seule, écriture unique, lecture/écriture*)

→ capacité (*720Ko à 25Go, voir 100Go*)

→ taux de transfert (débit) et temps d'accès (*500Ko/s à 20Mo/s*)

→ durée de vie, fiabilité (*qcq semaines à qcq décennies*)

Mémoire périphérique à accès séquentiel:

Bandes en tout genre

→ Nature du support et amovibilité (*lecture seule, écriture unique, lecture/écriture*)

→ capacité (*80Mo à 4Go, voir qcq To*)

→ taux de transfert (débit) (2*50Ko/s à 5Mo/s*)

→ durée de vie, fiabilité (*qcq années*)

Imprimante, scanner, modem

Imprimante:

→ protocole de communication (*Postscript niveau ? ou language propriétaire*)

→ technologie, couleur ou noir/blanc (*matricielle, à jet/bulles d'encre, à encre solide, laser*)

→ résolution max (*entre 300 et 2400 DotsPerInch*)

→ format/type de papier (*A4, A3, ... enveloppes*)

→ rapidité (pages par minute) (*d'une demi à quelques dizaines*)

Scanner:

Le *scanner* ou *digitaliseur* permet de numériser des documents, sous forme d'images.

→ format (*scanner à main, pleine page, A3, ...*)

→ résolution optique maximale (*entre 300 et 1200 DPI*)

modem:

Modulateur-Démodulateur, le *modem* permet une communication entre ordinateur, via un média destiné au transport d'information audio (ligne téléphonique).

→ technologie (*modem standard analogique, ADSL, numérique*)

→ vitesse d'émission/réception
(*de 9600 bauds à 56 Kb en analogique, 25 à 100 x plus en ADSL, et 64Kb/s en numérique*)

carte audio:

Ouvre les portes à l'exploitation des données audio.

→ nombre et nature des E/S (*audio, midi, mélange de canaux...*)

→ stéréophonie (*totale, sur certains canaux, à certaines fréquences, quadriphonie, ...*)

→ fréquence d'échantillonnage et espace de codage (*de 8KHz 8bits à 44Khz 16 bits*)

Ports d'entrées/sorties

Port série: Très longtemps utilisé, car simple à mettre en oeuvre, et possibilité d'utiliser les câble long, sans que cela ne perturbe la transmission.

→ débit maximum (*de 9600 à 19'200 bauds*)

Port parallèle: Utilisé pour transmettre les données plus rapidement que via une ligne série, il requiert l'utilisation d'un cable spécial, qui ne peut en aucun cas avoir la longueur du port série. Par ailleurs, le port dans sa version originelle est unidirectionel.

→ standard supporté (*port normal, EPP ou ECP port*)

Port SCSI: Le *Small Computer System Interface* est un bus d'entrées/sorties parallèles utilisés comme interface standard entre ordinateur et périphériques (max 8)

→ standard supporté (*SCSI-1: débit de 4 Mo/s, SCSI-2: débit de 10 à 40 Mo/s*)

Port PCMCIA: Le *Personal Computer Memory Card International Association* est une association dont la tâche est la normalisation des liaisons et du format des extensions pour les ordinateurs personnels, principalement les portables. Le format adopté est celui d'une carte de crédit, avec 3 épaisseurs différentes (3.3mm, 5mm et 10.5mm), et 1 connecteur de 68 broches.

Exemples d'offres (du jour)

Processor Intel Celeron 600 FCPGA	AMD Duron K7 3D Now 700 Mhz + Refroidisseur actif
Memory 64 MB	Dimm 128 MB PC-100-133 7ns 168pins SDRAM EEPROM
Hard disk 10.2GB 5400	20Gb IBM 2MB de cache Ultra DMA/66 7200rpm
Floppy 3.5 - 1.44 MB	Lecteur de disquettes 720Ko et 1.44MB
Graphic card 2D/3D ATI Rage Pro 8MB AGP	3D Blaster GeForce 256 Annihilator Pro DDR
CD-ROM 48x	Lecteur DVDROM Pioneer PC-DVD10x CD40x EIDE
Souncard on board	Creative Sound Blaster Live Player 1024
Speaker	HautParleurs Creative PCWorks FourPointSurround 1000
Logitech Mouse Pilot PS/2 & Mousepad	Souris Logitech Pilot 3 Boutons
Keyboard CH 105	Clavier 105 Touches (Suisse-Romand)
Midi Tower ATX	Medium-Tower ATX Case 250W Extreme
Windows 98 2nd ou Garantie Hardware 1 an	
Monitor 17" Typhoon T70S	
LG Goldstar Tube 0.27mm - 70kHz Garantie 3 ans	19 VideoSeven (Mitsubishi Diamondtron NF 0'25)
1690.-	**3005 .-**

Les principaux domaines de l'informatique

... abordés dans le cadre de ce cours:

- La Programmation

- Les Systèmes d'Exploitation
- Les Systèmes d'Information
- La Conception d'Interfaces
- Le Calcul Scientifique
- Les Agents Logiciels

Le système d'exploitation

Au fil des années, une spécialisation progressive des logiciels s'est réalisée:

- logiciels d'application:
  → résolution de problèmes spécifiques (traitement de textes, PAO, tableurs, logiciels de comptabilité, CAO (conception, design et simulation), ...)

- logiciels utilitaires:
  → logiciels qui servent au développement des applications (assembleur, compilateurs, dévermineur, ... , mais aussi gestionnaires de versions, gestionnaire de fenêtres, librairie de dessin, outils de communications...)

- logiciels systèmes (regroupés dans le *système d'exploitation*):
  → présents au cœur de l'ordinateur, ces logiciels sont à la base de toute exploitation, coordonnant les tâches essentielles à la bonne marche du matériel. Nous allons voir plus en détails leurs caractéristiques.

C'est du système d'exploitation que dépend la qualité
de la gestion des ressources (processeur, mémoire, périphériques)
et la convivialité de l'utilisation d'un ordinateur.

Définition d'un système d'exploitation

☞ Le système d'exploitation est l'ensemble des programmes qui se chargent de résoudre les problèmes relatifs à l'exploitation de l'ordinateur.

Plus concrètement, on assigne généralement deux tâches distinctes à un système d'exploitation:

- Gérer les ressources physiques de l'ordinateur

  → assurer l'exploitation **efficace, fiable et économique** des ressources critiques (processeur, mémoire)

- Gérer l'interaction avec les utilisateurs

  → faciliter le travail des utilisateurs en leur présentant une machine plus simple à exploiter que la machine réelle (concept de ***machine virtuelle***)

Apparition des systèmes d'exploitation

Les premières machines étaient dépourvues de système d'exploitation; à cette époque, toute programmation était l'affaire de l'utilisateur.

Dès lors, le «passage en machine» (exécution d'un programme) nécessitait un ensemble d'opérations longues et fastidieuses. Par exemple, lorsque la machine s'arrêtait (suite à une panne), il fallait à nouveau programmer à la main l'amorcage.[1]

Avec les machines de seconde génération, on commenca à automatiser les opérations manuelles, ce qui améliora l'exploitation des différentes unités.

Pour cela, des programmes spécifiques appelés *moniteurs* ou *exécutifs* firent leur apparition, leur rôle étant d'assurer la bonne marche des opérations (séquencement des travaux des utilisateurs).

⇨ le système d'exploitation était né.

1. Entre autre l'accès à la mémoire secondaire (lecteur de cartes ou de bandes), puis le chargement des utilitaires de bases, tels l'assembleur, et finalement le programme lui-même.

Caractéristiques fondamentales

Il existe aujourd'hui un nombre très important de systèmes d'exploitation (plusieurs centaines). La tendance actuelle est toutefois à la standardisation, en se conformant aux systèmes existants plutôt qu'en en développant de nouveaux.

⇨ un avantage essentiel est la **portabilité** du code existant.

Classification fréquente des systèmes d'exploitation:

<table>
<tr><th colspan="2">Mono-tâche</th><th colspan="2">Multi-tâches</th></tr>
<tr><td colspan="2">A tout instant, un seul programme est exécuté; un autre programme ne démarrera, sauf conditions exceptionnelles, que lorsque le premier sera terminé.</td><td colspan="2">Plusieurs processus (i.e. un «programme» en cours d'exécution) peuvent s'exécuter simultanément (systèmes multi-processeurs) ou en quasi-parallélisme (systèmes à temps partagé)</td></tr>
<tr><th colspan="3">mono-session</th><th>multi-sessions</th></tr>
<tr><td colspan="3">Au plus un utilisateur à la fois sur une machine. Les systèmes réseaux permettent de différencier plusieurs utilisateurs, mais chacun d'eux utilise de manière exclusive la machine (multi-utilisateurs, mono-session)</td><td>Plusieurs utilisateurs peuvent travailler simultanément sur la même machine.</td></tr>
<tr><td>Exemple: DOS</td><td>Exemple: Windows (tous)</td><td></td><td>Exemple: VMS, Unix</td></tr>
</table>

Structure en couche d'un SE moderne

Les couches permettent d'identifier clairement les niveaux d'interdépendance entre les fonctions fournies par le S.E.

Le noyau

Les fonctions principales du noyau (d'un SE multi-tâches) sont:

- Gestion du processeur:
  - → reposant sur un allocateur (*dispatcher*) responsable de la répartition du temps processeur entre les différents processus,
  - → et un planificateur (*scheduler*) déterminant les processus à activer, en fonction du contexte.

- Gestion des *interruptions*:
  - → les *interruptions* sont des signaux envoyés par le matériel, à destination du logiciel, pour signaler un évènement.

- Gestion du multi-tâches:
  - → simuler la simultanéité des processus coopératifs (i.e. les processus devant se synchroniser pour échanger des données)
  - → gérer les accès concurrents aux ressources (fichiers, imprimantes, ...)

Du fait de la fréquence élevée des interventions du noyau, il est nécessaire qu'il réside en permanence et en totalité dans la mémoire centrale. Son codage doit donc être particulièrement soigné, pour être à la fois performant et de petite taille.

Système de Fichiers

Le concept de fichiers est une structure adaptée aux mémoires secondaires et auxiliaires permettant de regrouper des données.

Le rôle d'un système d'exploitation est de donner corps au concept de fichiers (les gérer, c'est-à-dire les créer, les détruires, les écrire (modifier) et les lires, en offrant la possibilité de les désigner par des noms symboliques).

Dans le cas de systèmes multi-utilisateurs, il faut de plus assurer la **confidentialité** de ces fichiers, en protégant leur contenu du regard des autres utilisateurs.

Remarquons que cet aspect des systèmes multi-utilisateurs implique l'autentification de l'utilisateur en début de session de travail, généralement au moyen d'un ***login*** (nom d'utilisateur + mot de passe)

Entrées-Sorties

La gestion des entrées-sorties est un domaine délicat
dans la conception d'un système d'exploitation:

il s'agit de permettre le dialogue (échange d'informations)
avec l'extérieur du système.

La tâche est rendue ardue, part la diversité des périphériques
d'entrées-sorties et les multiples méthodes de codage des informations
(différentes représentations des nombres, des lettres, etc.)

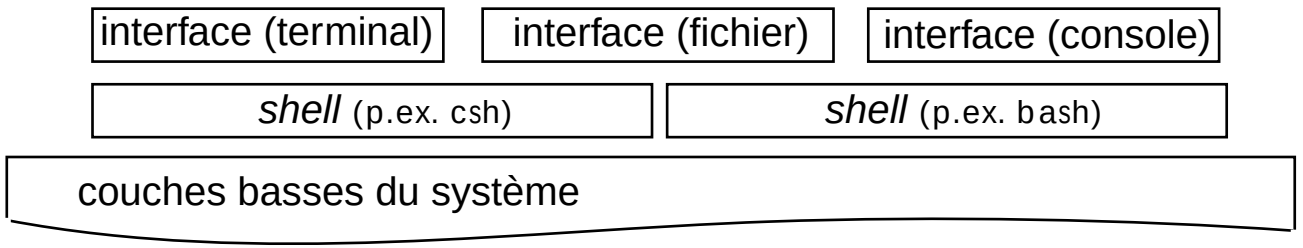
Concrètement, la gestion des E/S implique que le SE mette à disposition
de l'utilisateur des procédures standard pour l'émission et la réception des données,
et qu'il offre des traitements appropriés aux multiples conditions d'erreurs susceptibles
de se produire (plus de papier, erreur de disque, débit trop différent, ...)

Interprêteur de commandes (1)

Pour interagir avec l'utilisateur,
un système informatique doit disposer au minimum
d'un **interprêteur de commandes** (*shell*) couplé à une **interface** rudimentaire.[1]

Le *shell* attend les ordres que l'utilisateur transmet par le biais de l'interface, décode et décompose ces ordres en actions élémentaires, et finalement réalise ces actions en utilisant les services des couches plus profonde du système d'exploitations.

Le *shell* est donc un traducteur de type interprêteur, et le **langage de commande** qu'il prend en compte est un langage de programmation interprêté.



Parmis les *shells Unix* les plus utilisés, citons:
Bourne [Again] shell (sh et bash), *C shell* (csh), *Korn shell* (ksh), *Z shell* (zsh),
et celui présent par défaut sur les comptes du cours, l'*Enhanced C shell* (**tcsh**).

1. Contrairement à l'architecture figée de Dos ou Windows, les systèmes de la famille Unix définissent en tant que composants externes l'interface utilisateur et l'interprêteur de commandes. Ne faisant pas directement partie du système, ils peuvent être changés si nécessaire.

Interprêteur de commandes (2)

Pour faciliter le travail de l'utilisateur,
les interprêteurs de commandes offrent la possibilité de
définir des variables d'environnement, de **renommer les commandes**,
d'en définir de nouvelles, par «paramètrage» et enchaînement des commandes
de base, etc. La plupart offrent également des **facilités d'édition** comme
le rappel des commandes précédentes (historique des commandes),
la complétion, la correction (suggestion de correction) en cas
de commande invalides, et bien d'autres fonctionnalités.

Tous les systèmes d'exploitation permettent par ailleurs,
en plus de l'interaction «directe» (au moyens de terminaux ou
de consoles dans le cas d'*Unix*), le «***traitement par lots***» (***batch***).

Ce mode de traitement non-interactif est obtenu en regroupant
les comandes dans un fichier alors appelé ***script***.

Mémoire virtuelle (1)

La mémoire centrale a toujours été une ressource critique: initialement très coûteuse et peu performante (torres magnétiques), elle était de très faible capacité.

Avec l'apparition des mémoires électroniques, la situation s'est améliorée, mais comme parallèlement la taille des programmes considérablement augmenté, la mémoire demeure, même de nos jours, **une ressource critique**, et donc à économiser.

Pour pallier le manque de mémoire centrale, l'idée est venue d'utiliser des **mémoires secondaires** [de type disque dur], **plus lentes**, mais de beaucoup **plus grandes capacités**.

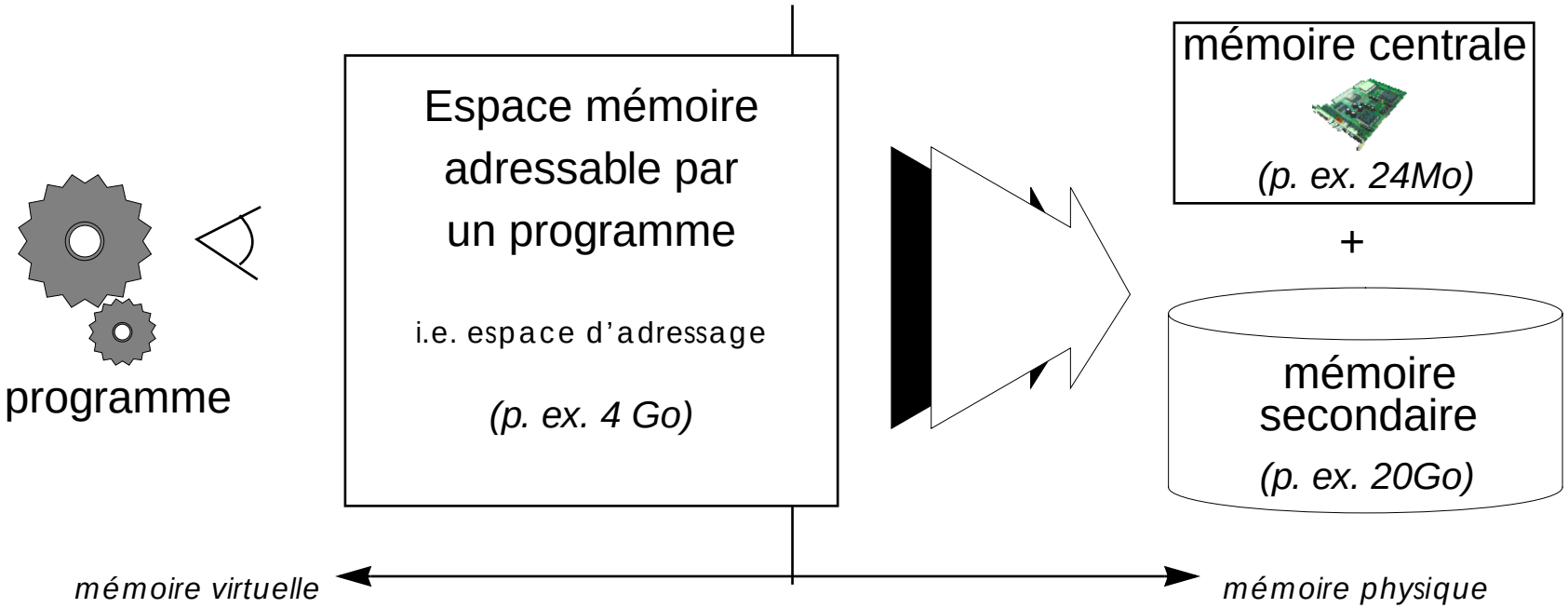
Pour cela, un concept à la fois simple et élégant fut établit:

la **mémoire virtuelle**.[1]

1. Le concept de mémoire virtuelle est présenté dans le cadre des systèmes d'exploitation, car historiquement c'était le S.E. qui en était chargé. Pour des raison de performances, une grande partie du travail est maintenant réalisée par microprogrammation, directement dans les processeurs.

Mémoire virtuelle (2)

La mémoire virtuelle repose sur une **décorellation** entre la **mémoire physique** (centrale ou secondaire), présente sur la machine, et **l'espace mémoire mis à disposition des programmes** par le système d'exploitation (la **mémoire virtuelle**, ou **logique**).

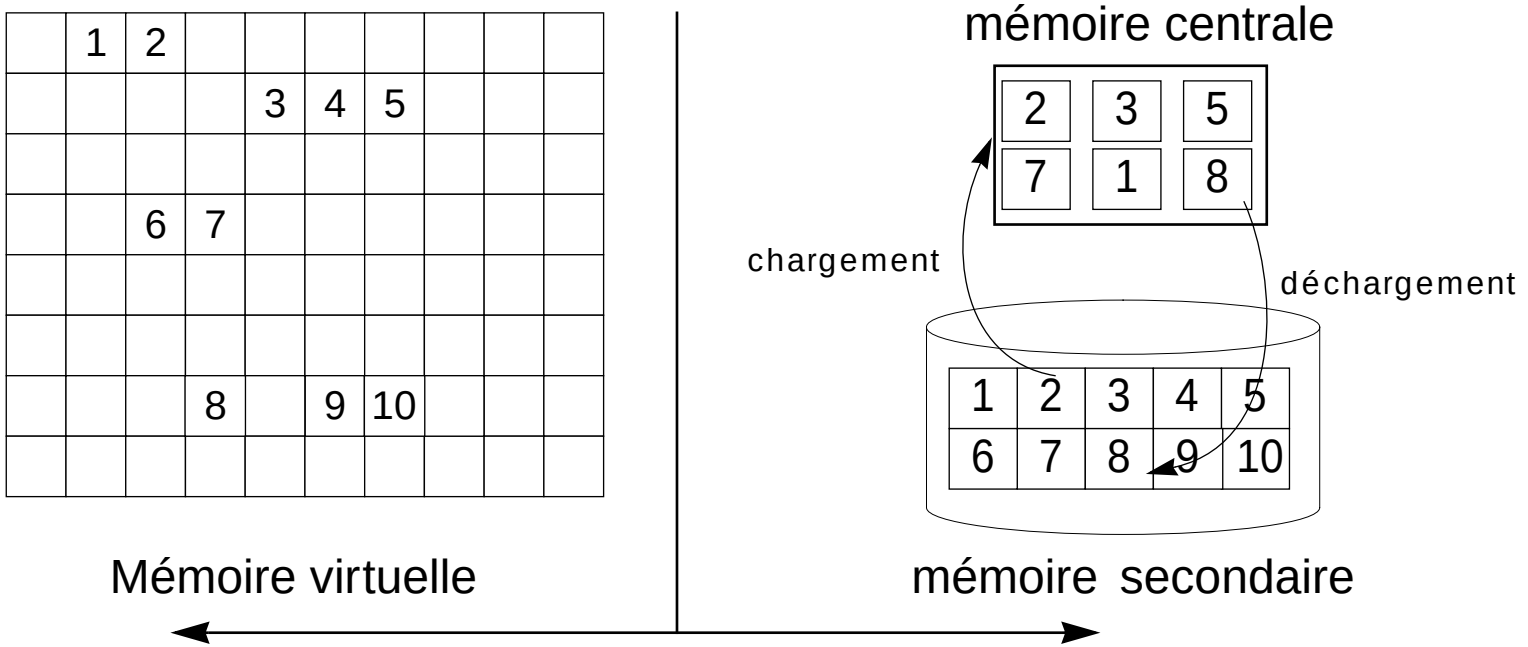


C'est le système d'exploitation qui prend en charge la **mise en correspondance** entre les adresses utilisées dans les programmes (*adresses virtuelles*) et les adresses correspondant effectivement à de la mémoire physique (*adresses réelles*).

Mémoire virtuelle (3)

Une technique fréquemment utilisée pour réaliser
cette mise en correspondance est la **pagination**.

La mémoire virtuelle utilisable par un programme (aussi grande que le permet la taille du codage des adresses) est **segmentée** en zones de tailles identiques, les **pages**. A tout instant, **seul un nombre limité** de ces pages est effectivement (i.e. physiquement) présent dans la mémoire centrale, le reste [des pages allouées, i.e. utilisées] étant conservé en mémoire secondaire, et **chargé** en mémoire centrale en cas de besoin.



Mémoire virtuelle (4)

☞ La mémoire centrale ne pouvant contenir qu'un nombre limité de pages, il faut donc parfois effectuer des **remplacements** (*swapping*).[1]

Quelle stratégie employer pour réaliser ces remplacements ?

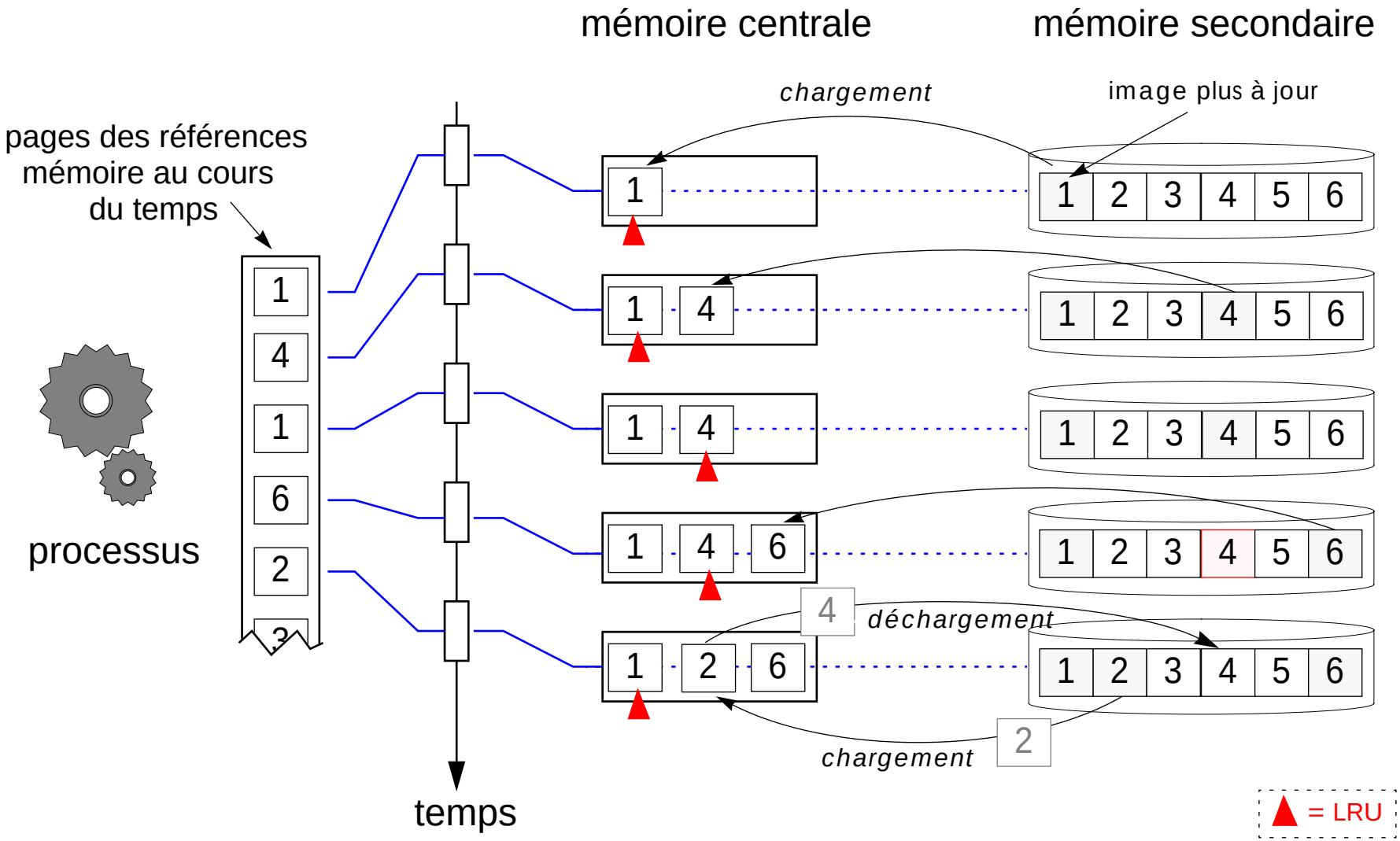
Dans le cas idéal, il faudrait remplacer les pages qui ne seront plus utilisées, du moins celle qui le seront le plus tard possible... Malheureusement, il n'est en général pas possible pour le système de prévoir le déroulement des processus.

Une stratégie largement répandue est celle du **LRU** (*Least Recently Used*), et consiste à **remplacer la page la moins récemment utilisée**.[2]
Cette heuristique est basée sur l'observation suivante:[3]

«les références mémoires d'un processus restent groupées, et évoluent lentement dans l'espace d'adressage»

1. On parle de *défaut de page* lorsque survient un référencement à une page non présente en mémoire centrale.
2. Parmis les autres algorithmes courants, citons FIFO, qui consiste à éliminer de la mémoire la plus ancienne page, LFU (*Least Frequently Used*) la page la moins fréquemment référencée, et ... RANDOM, qui choisit la page de manière aléatoire (algorithme sous-optimal de référence) !
3. Egalement appelé «principe de localisation des références mémoire». Cette observation entre dans le cadre de la *règle des 80-20* (aussi désignée sous le nom de *règle des 90-10*); nous aurons l'occasion de revoir cette règle en détails dans la suite du cours, mais l'on peut d'ores et déjà donner son interprétation dans le contexte de l'utilisation mémoire: 80% (respectivement 90%) des références mémoires d'un programme sont faite dans les mêmes 20% (respect. 10%) de l'espace d'adressage total.

Mémoire virtuelle (5)



Mémoire virtuelle (6)

Naturellement, l'utilisation d'un mécanisme de mémoire virtuelle a un **coût**:

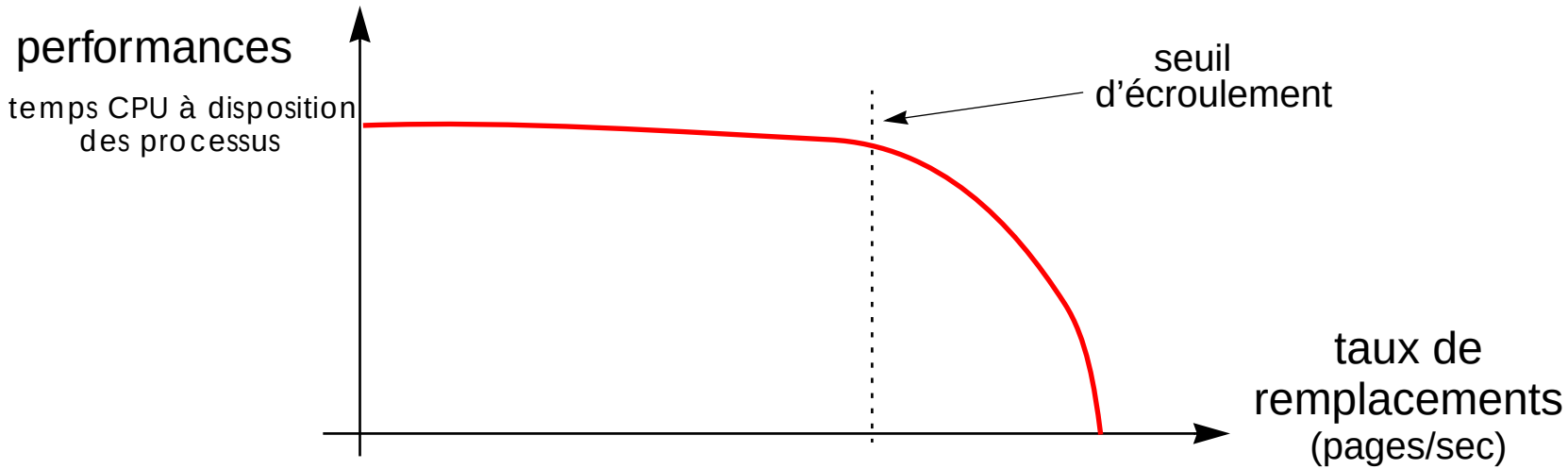
- **En espace mémoire**:
  la gestion des pages implique l'utilisation de tables de descripteurs (informations sur la localisation des pages, validité de l'image disque, droits d'accès en fonction des processus, pile du *LRU*, ...), ce qui consomme de l'espace mémoire, et les pages ainsi gérées sont sujettes à la *fragmentation interne* (les pages ne sont souvent que partiellement utilisées, mais la totalité de l'espace mémoire correspondant est réservé).

- **En temps de traitement**:
  consommé par la gestion, le chargement et le déchargement des pages.

Mémoire virtuelle (7)

La **fréquence des remplacements de pages** est directement liée au rapport entre le nombre de pages pouvant être logée en mémoire et le nombre totale de pages allouables par un programme.

Lorsqu'un programme provoque des remplacements trop fréquents, on dit qu'il *s'écroule*.
Dans ce cas, il peut arriver que le système monopolise l'essentiel du temps pour les transferts de pages entre disque et mémoire, et ne permette plus aux programmes de progresser dans leur exécution, **ce qu'il faut à tout prix éviter**.[1]



1. La plupart des systèmes d'explopitation permettent de verrouiller des pages en mémoire, afin d'interdire leur remplacement. Cette possibilité permet de limiter le risque d'écroulement, typiquement dans le cas de processus temps-réel, en assurant la présence permanente dans la mémoire des pages «critiques» (i.e. périodiquement accédées)

Processus (1)

☞ Les périphériques réalisent souvent des opérations **lentes** à l'échelle du processeur.

Considérons par exemple un programme qui effectue une requête de lecture de données sur un CD-ROM.

⇨ La première opération à faire, au niveau du système, sera de positionner la tête de lecture du CD-ROM sur le premier bloc à lire.

Admettons qu'un tel positionnement se fasse en moyenne en 50 [ms], et que le système soit doté d'un processeur de type RISC, cadensé à 500 [MHz].

Le temps de positionnement de la tête de lecture représente l'exécution de:

$$(50\times10^{-3})[\mathrm{s}]\times(500\times10^{6})[\mathrm{op/s}] \;=\; 25\times10^{6}[\mathrm{op}]$$

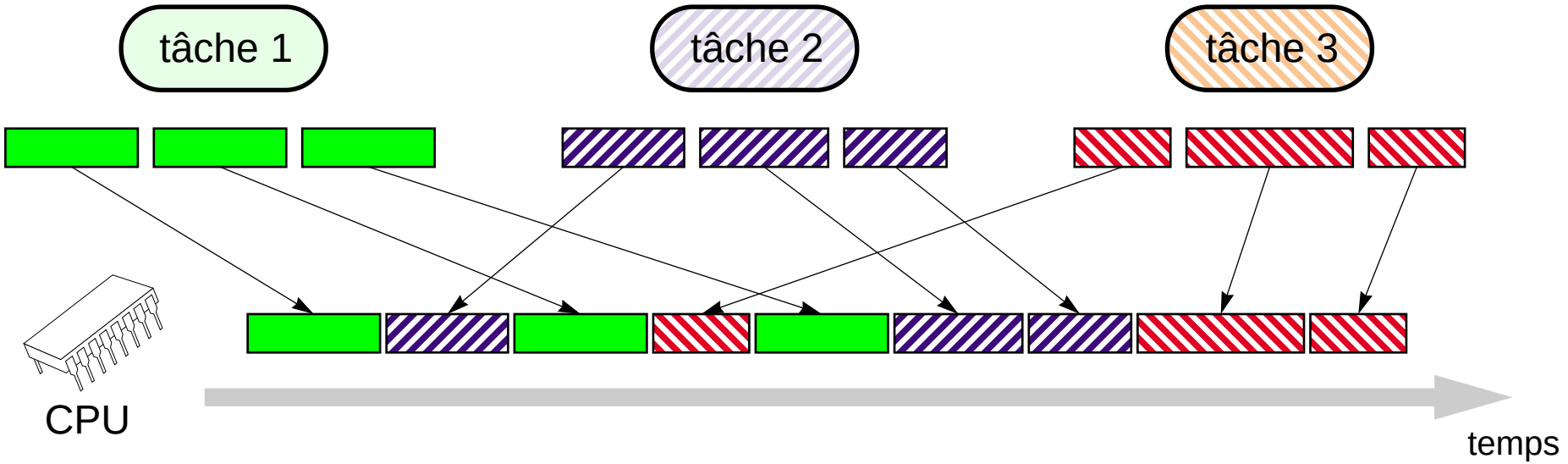
soit **25 millions d'opérations**

qui pourraient être exécutées en attendant la fin de l'opération d'entrée-sortie.

Processus (2)

L'idée est donc tout naturellement venue d'utiliser ce temps d'oisiveté au profit d'autres tâches se déroulant **en parallèle** des opérations d'entrée-sortie.

Cette idée a rapidement été étendue à une forme plus générale de parallélisme basé sur l'**entrelacement de l'exécution des tâches** (***parallélisme logique***).



☞ On parle alors de ***systèmes à temps partagé*** [*time sharing*].

Processus (3)

L'un des rôles importants d'un système d'exploitation
est de permettre la **mise en œuvre effective du *parallélisme logique***.

Unix est ainsi un système qui permet de **lancer simltanément plusieurs programmes** sur une même machine (mono ou multi-processeur):

☞ On parle dans ce cas de *système* [ou *environnement*] ***multi-tâches***.

☞ Dans la suite, nous appelerons:

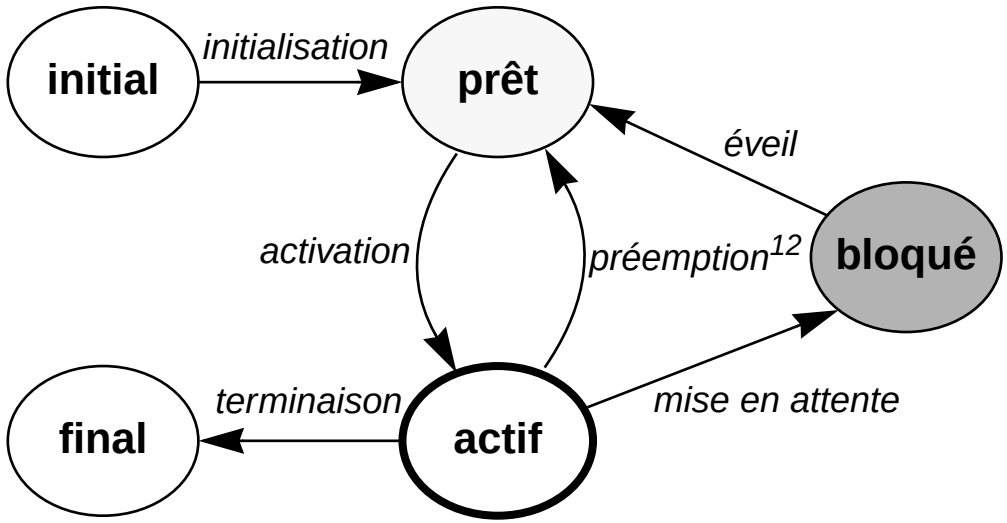
«***processus***» (*process*) ou «***tâche***» (*task*)

tout programme exécuté dans un environnement

multi-tâches.

Processus (4)

De plus, on définit, pour tout processus, les états suivants: (*cycle de vie*)



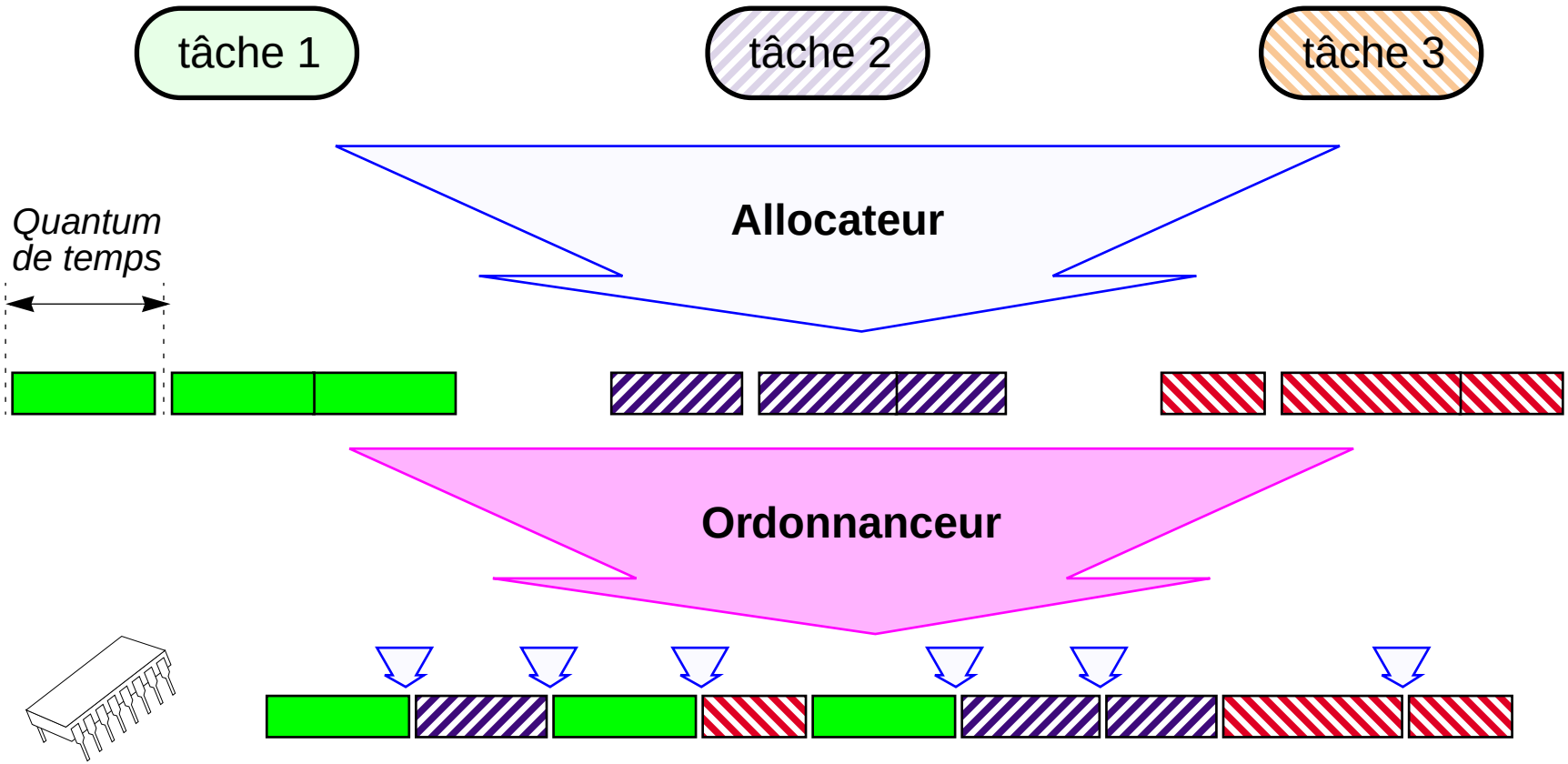
- Les processus ***prêts*** sont les processus potentiellement exécutables;
- Les processus ***actifs*** sont ceux en cours d'exécution (par le processeur)
- Les processus ***bloqués*** sont ceux en attente de synchronisation (fin d'une opération d'entrée-sortie, ou plus généralement en attente d'un autre processus).

Sur un système monoprocesseur, il ne peut (évidemment) y avoir à tout instant qu'un et un seul processus actif.

1. La préemption correspond à l'interruption (temporaire) d'un processus, à la fin du quantum de temps qui lui était alloué.

Processus (5)

L'***ordonnanceur*** (*scheduler*) est responsable du **choix du processus à activer**, parmis l'ensemble des processus prêts. L'***allocateur*** (*dispatcher*) définit le temps maximum pendant lequel un processus pourra rester actif (le *quantum de temps* alloué au processus); il **fixe les échéances des préemptions** (i.e. des mise en veille temporaire des processus).

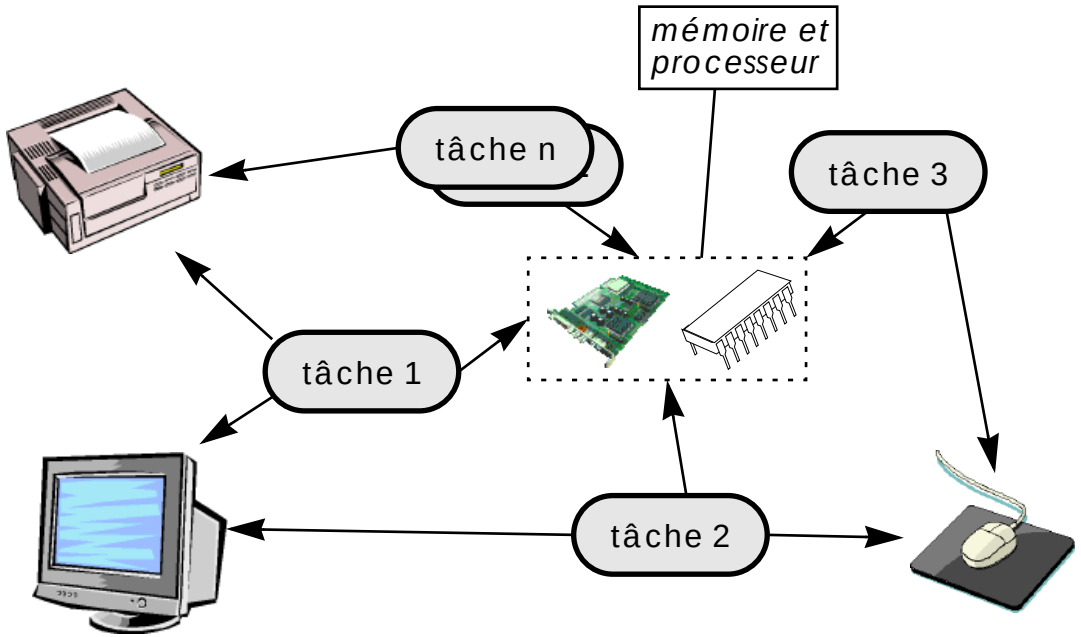


Processus (6)

La mise en œuvre d'un système multi-tâches implique que les processus puissent:

• s'exécuter comme s'ils étaient seuls sur la machine (monopole)
⇨ **concurrence d'accès aux ressources**

• coopérer avec d'autre processus s'exécutant simultanément
⇨ **communication**
⇨ **échange de données**
⇨ nécessité de **synchronisation**.



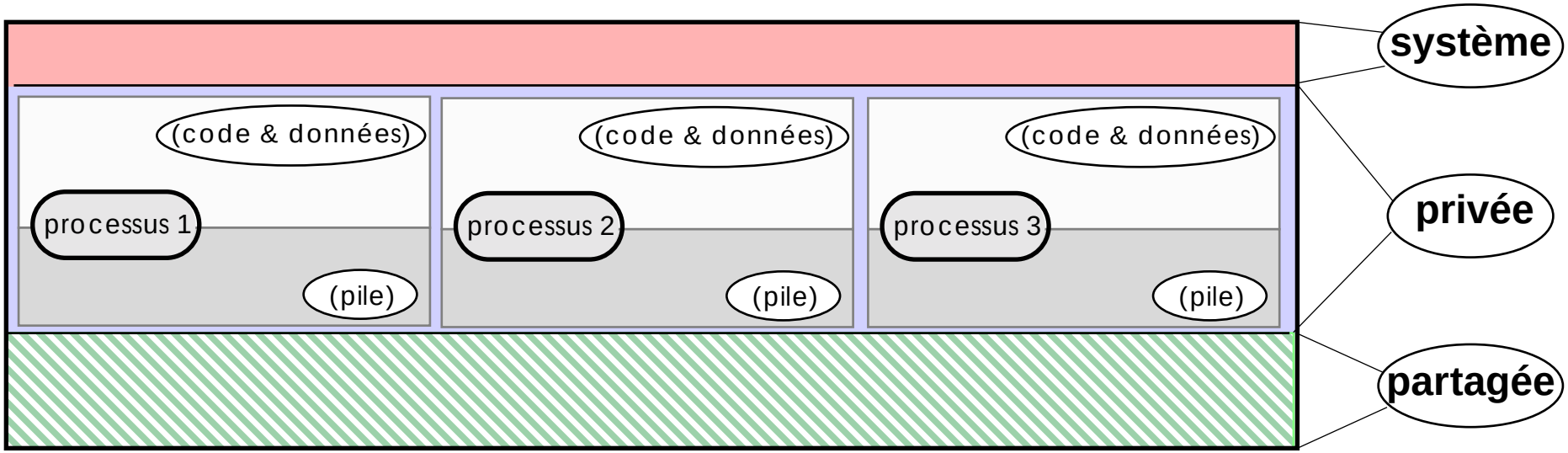
Les systèmes multi-tâches sont ainsi des exemples de

systèmes concurrents,

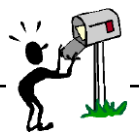
les tâches étant **en compétition** les unes par rapport aux autres pour l'obtention des ressources physiques

Concurrence d'utilisation mémoire

Pour régler les problèmes de concurrence d'utilisation mémoire, tout en permettant la coopération des processus, l'espace d'adressage est décomposée en zones: *(par exemple)*



- La **zone «système»** contient le noyau du système d'exploitation et les services de base.
- Une **zone privée** propre à chaque processus, dans laquelle le code et les données sont logées; il est assez aisé pour le gestionnaire de mémoire virtuelle de dupliquer cet espace pour chaque processus.
- Finalement, on trouve généralement un **espace commun** à tous les processus (accessible en écriture), permettant une communication appelée **communication par mémoire partagée**, ou ***par boîtes aux lettres***).



Concurrence d'accès aux périphériques (1)

Une possibilité pour résoudre les concurrences d'accès est
la **réservation** des périphériques par les processus devant en faire usage:
le périphérique est alloué au processus, jusqu'à la terminaison de l'opération.
Mais les périphériques travaillent beaucoup plus lentement que le processeur,

☞ Se pose alors le problème de **savoir à quel moment cette opération est terminée** ?

Une première possibilité, et dans bien des cas la seule,
est d'utiliser le principe du **drapeau** (*flag* ou *status*):

en début d'opération, une variable (drapeau) est positionnée à une certaine valeur;
lorsque l'opération se termine, le périphérique modifie la valeur de ce drapeau.

Parallélement, le processus teste à intervalles réguliers la valeur du drapeau, jusqu'à ce que le périphérique signale la fin de l'opération.

On appelle ce mode de travail la ***scrutation*** (*polling*), et
la mise en attente du processus sur une boucle de test est appellée: ***attente active***.

Concurrence d'accès aux périphériques (2)

Afin de réduire le gaspillage de temps induit par les scrutations infructueuses, une autre solution à été envisagée, au niveau du matériel:

Au lieu de laisser le contrôle à la charge du processus invoquant l'opération, on va donner la possibilité au périphérique de **signaler de manière active** la fin de l'opération (en fait tout événement digne d'être signalé), en **interrompant le programme** en cours de traitement (*rupture de séquence*) au profit d'une routine spéciale (routine de service), destinée à effectuer le minimum d'opérations nécessaires à la prise en compte de l'événement. Lorsque cette routine se termine, l'exécution du programme initial reprend.

L'occurrence d'un tel signal émis par le périphérique est appellée ***interruption***.

La différence fondamentale entre la syncronisation par scrutation et celle par interruption réside au niveau du **passage de contrôle**:

- dans le premier cas, le programme *garde le contrôle*, et le «prête» explicitement au périphérique par le biais de la boucle d'attente;
- dans le second cas, le contrôle *est pris* par le périphérique, de manière implicite vis à vis du programme.

Concurrence d'accès au terminal

Le fait que plusieurs programmes soient exécutés «simultanément» a une incidence sur les **interactions de ces programmes avec l'utilisateur**, en particulier pour ceux qui utilisent la console ou le **terminal** (i.e. les programmes qui n'ont pas d'interface de dialogue privée – *ce sera le cas de la plupart de ceux écrits dans le cadre de ce cours*).

Cependant, si le programme dispose de sa propre fenêtre d'interaction (c'est le cas de la plupart des applications – Netscape, Emacs – utilisées dans le cadre de ce cours), l'interaction dans le terminal se réduit à la commande de lancement du programme. Dans ce cas, l'utilisateur peut lancer ce programme en **mode «*détaché*»** (on parle également de lancement en «***arrière-plan***»), par exemple en faisant suivre la commande du signe « &»[1].

Le processus sera alors exécuté par le système, **indépendamment du terminal** qui aura servi à le lancer.

1. `CTRL+Z` permet de suspendre le processus actif relié au clavier (il devient bloqué), tandis que `CTRL+C` l'interrompt définitivement. «`bg`», acronyme de *background*, permet de passer le processus en arrière-plan (son exécution reprendra, mais détachée du shell qui l'a lancé) tandis que «`fg`», acronyme de *foreground*, permet de ré-attacher (repasser au premier plan) le dernier processus détaché.

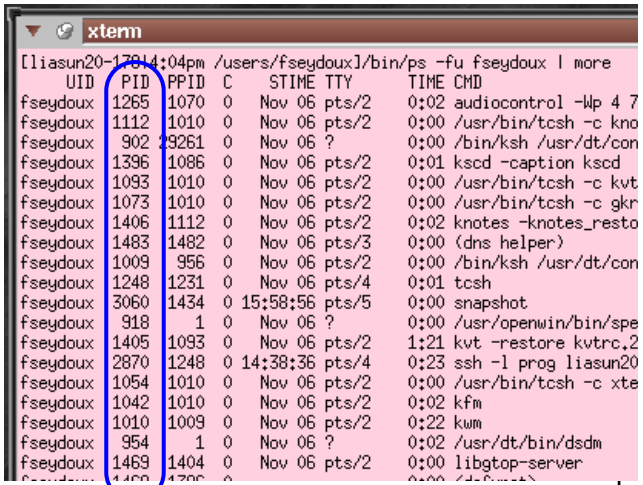
Processus, commandes pratiques

Exemples pratiques:

- Pour obtenir la liste des processus d'un utilisateur (user):
  `$>ps -fu <user> | more`

- Pour obtenir de manière actualisée la liste des principaux processus, triés selon leur consomation CPU:
  `$>top`

- Pour détruire un processus:
  `$>kill <PID>` : on demande au processus son auto-destruction
  `$>kill -9 <PID>` : destruction du processus commandée au niveau système
  `$>xkill [-frame]` : permet de choisir à la souris l'application graphique à tuer.



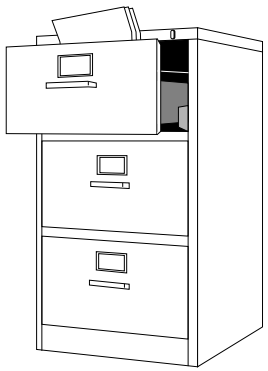
Fichiers

La capacité des *mémoires auxiliaires* – i.e. les *mémoires secondaires* (disques durs) et les *mémoires de masse* (CDs, bandes) – étant très grande, on peut y loger un grand nombre de données.

Il est donc indispensable de fournir un moyen pour **organiser ces données** (i.e. les regrouper au sein d'entités logiques de plus haut niveau). A cette fin, les systèmes d'exploitation utilisent le concept de «***fichier***»:

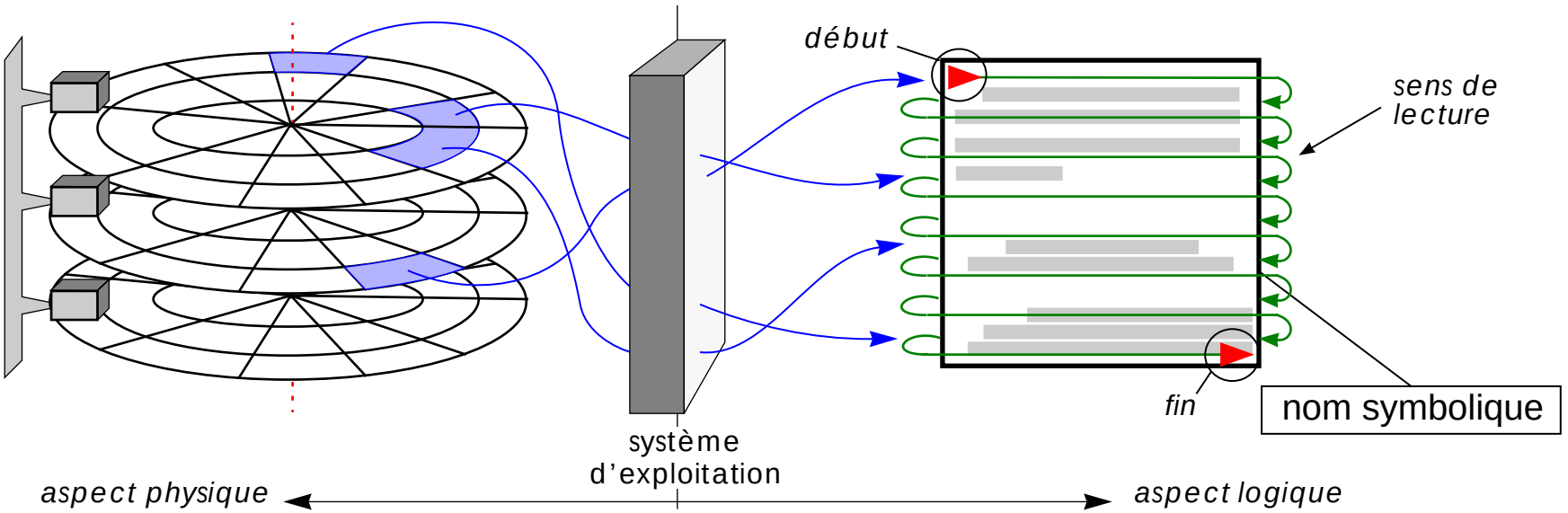
Un *fichier* est un **regroupement logique de données** présentes sur une mémoire secondaire, avec un début, une fin, une taille d'enregistrement de base, un ordre de lecture et un certain nombres d'attributs.

C'est donc une **collection ordonnée de données**, **représentant une entité** pour l'utilisateur.



Organisation de la mémoire secondaire

L'un des rôles du système d'exploitation est de permettre la mise en œuvre du concept de fichier, **indépendamment du support physique utilisé**.



Le système d'exploitation doit donc permettre la **gestion** (création, destruction, modification et lecture) des fichiers, en offrant en particulier à l'utilisateur la possibilité de les désigner par des **noms symboliques**, sans avoir à se préocuper de leur organisation physique dans la mémoire auxiliaire..

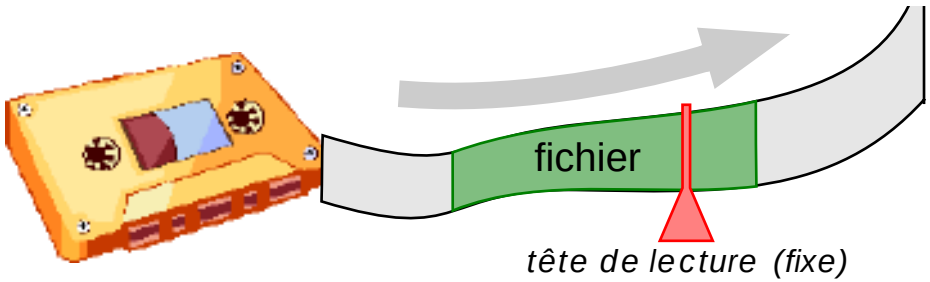
Dans le cas de systèmes multi-utilisateurs, il faut de plus assurer la **confidentialité** des données, en protégant le contenu des fichiers du regard des autres utilisateurs.

Méthodes d'accès aux données

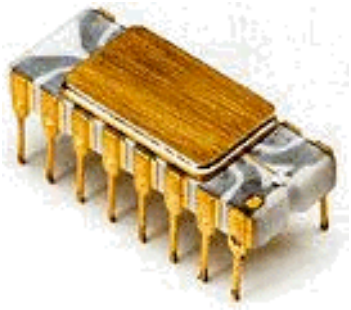
L'accès aux données stockées dans un fichier est généralement contraint par la mémoire auxiliaire utilisée.

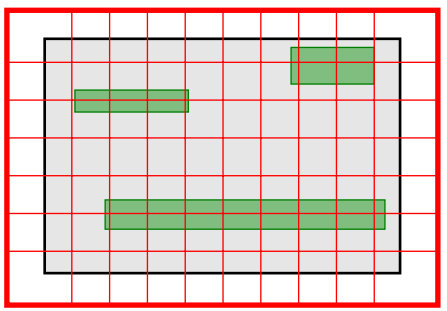
On distingue habituellement deux grands types de fichiers,[1] suivant la nature des accès qu'ils permettent:

- les *fichiers à **accès séquentiel***:
  suite d'enregistrements, accessibles les uns après les autres. Pour lire une portion du fichier, il faut le parcourir depuis le début, jusqu'à atteindre la portion en question.



- les *fichiers à **accès direct***:
  (aussi appelés *fichiers à accès aléatoire*) chaque enregistrement du fichier est accessible directement. Pour lire une portion du fichier, il suffit de connaître le *déplacement* (rang de l'enregistrement concerné × la taille des enregistrements).



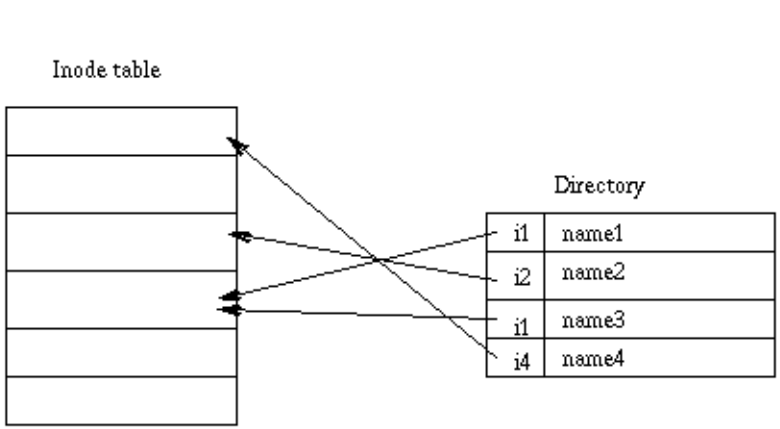


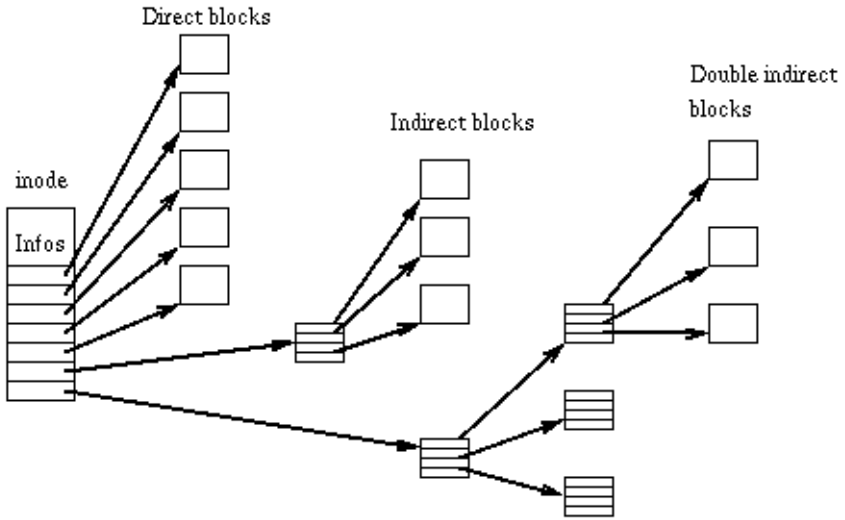
1. Remarquons qu'il existe beaucoup d'autre types de fichiers (par exemple les fichiers relatifs, indexés, indexés à clefs multiples, ISAM, VSAM, etc). Toutefois, ces types «exotiques», apanage des «gros» systèmes (VMS, IBM) des années 80 ont tendance à disparaîtres avec eux.

Système de fichiers

Pour assurer la gestion des fichiers, un système d'exploitation utilise un [voire plusieurs] ***système[s] de fichiers*** (*file system*).

C'est le *système de fichiers* qui détermine les **structures internes** utilisées pour organiser les fichiers.





Parmis les très nombreux systèmes de fichiers, citons:

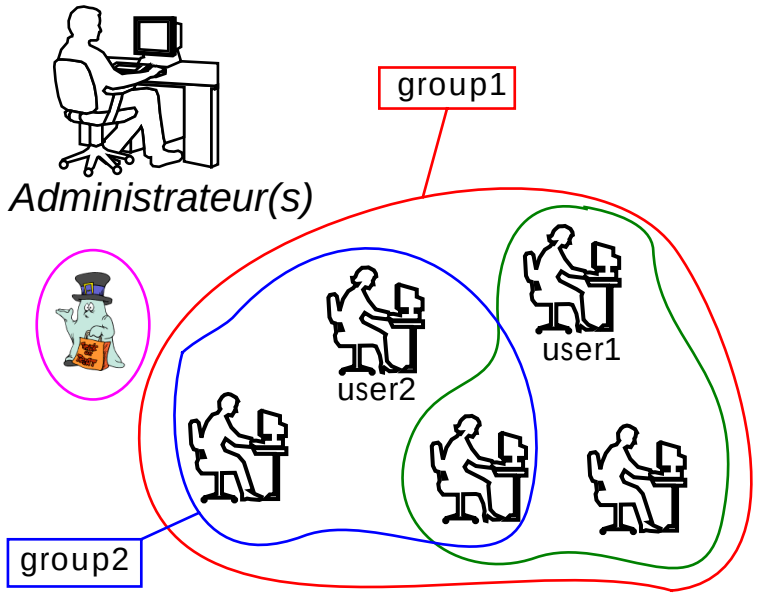
- **FAT**, VFat, HPFS, **NTFS** (*Dos* & *Windows*),
- ISO9600, UDF, UFS, Joliet, RockRidge (CD),
- Ext, Ext2, Ext3, (*Linux*)
- SystemV, VxFS, Spiralog (*Solaris*, *VMS*)
- **NFS** (*Network File System*)

Utilisateurs et groupes

Dans le cas d'architectures multi-utilisateurs, il est primordial d'identifier les personnes désireuses de travailler avec le système, afin de pouvoir assurer la confidentialité de leur données, et parfois de leur facturer les ressources utilisées.

☞ A cette fin, les systèmes d'exploitation utilisent un modèle à 2 entités:
- ⇨ les **utilisateurs**, et
- ⇨ les **groupes** d'utilisateurs

Ceci permet de définir une politique des droits (ressources utilisables et dans quelle quantité – ***privilèges***) commune à un ensemble d'utilisateurs.



☞ Chaque utilisateur appartient à au moins 1 groupe

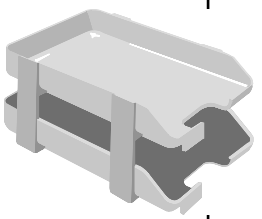
☞ Il existe des utilisateurs et des groupes d'utilisateurs particuliers:

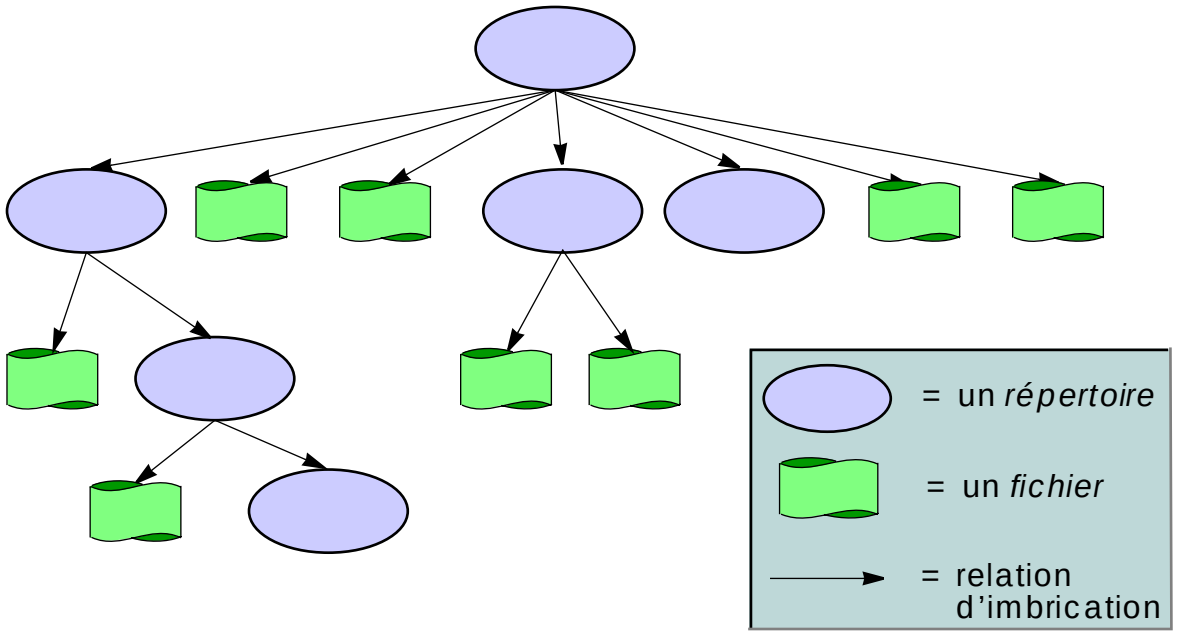
- les administrateurs (utilisateurs en charge du système, avec tous les privilèges)
- les utilisateurs «fantômes», utilisés par des programmes particuliers (par exemple les programmes réalisant les sauvegardes, les démons, ...

Structuration d'un système de fichiers (1)

La capacité des mémoires auxiliaires étant très grande, on peut y loger un **grand nombre de fichiers**. Par exemple, un compte d'utilisateur peut facilement contenir plusieurs milliers (voire dizaines de milliers) de fichiers, un système de fichiers complet pouvant contenir quant à lui plusieurs millions de fichiers.

Il est par conséquent indispensable de fournir un moyen pour **organiser ces fichiers**. A cette fin, pratiquement tous les systèmes d'exploitation utilisent le concept de **répertoire** (aussi appelé *catalogue* ou *directory*).





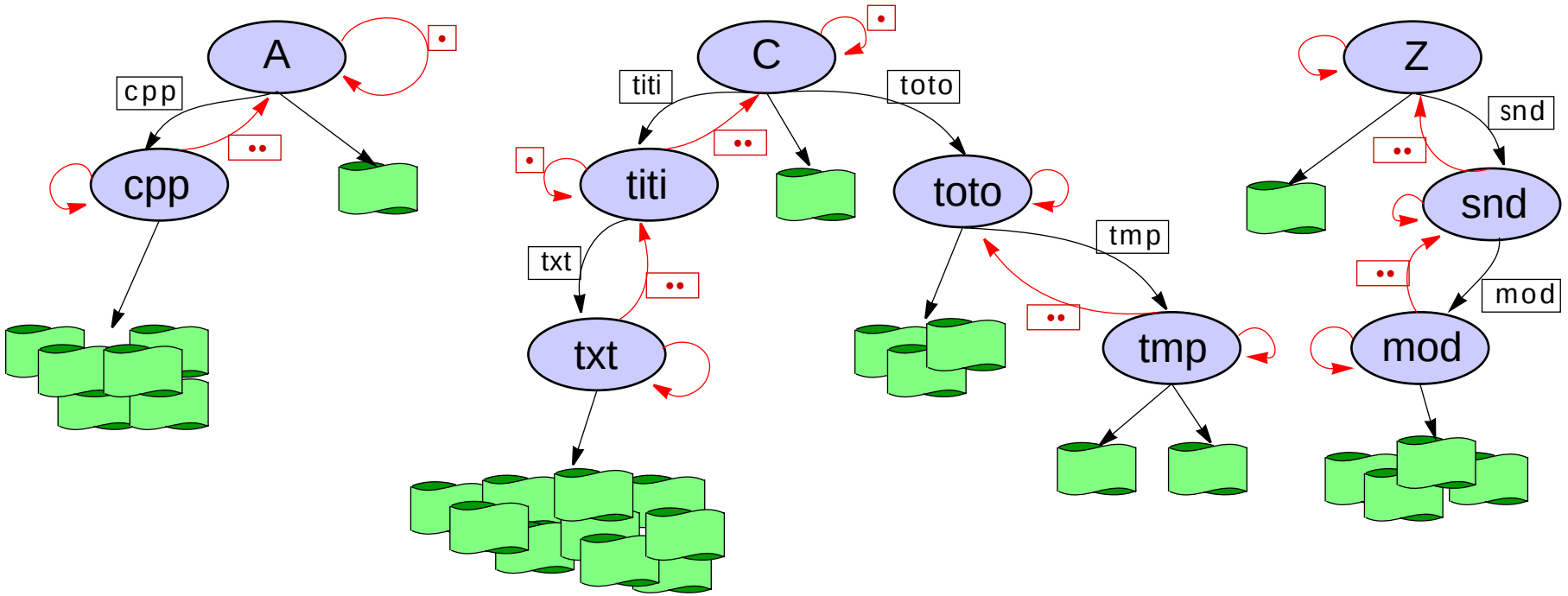
Un *répertoire* est une **collection** (généralement non ordonnée) **de fichiers ou de répertoires** (alors appelés *sous-répertoires*).

Ils permettent d'organiser l'ensemble des fichiers dans une **structure arborescente**:

Structuration d'un système de fichiers (2)

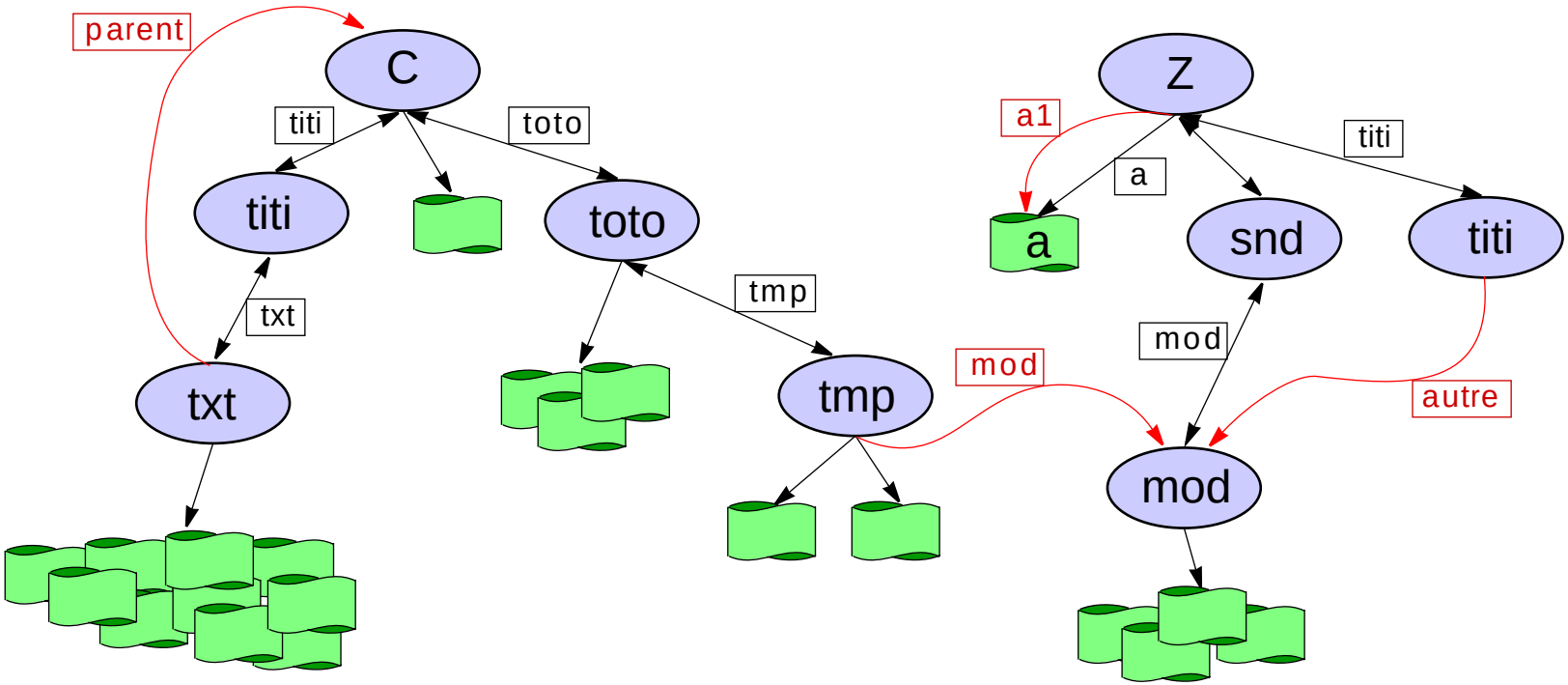
De nombreux systèmes (p.ex. *Windows*) utilisent en fait une **forêt**, c'est à dire qu'il maintiennent plusieurs structures arborescentes, en associant les racines de ces structures à des noms de lecteurs. Ce n'est pas le cas de Unix, qui ne possède qu'une arborescence.

Une arborescence de fichiers peut être parcourue **dans les deux sens**.
Cette possibilité de double parcours est assurée **par le système d'exploitation**, qui conserve, pour chaque répertoire, une référence vers le répertoire *parent*.



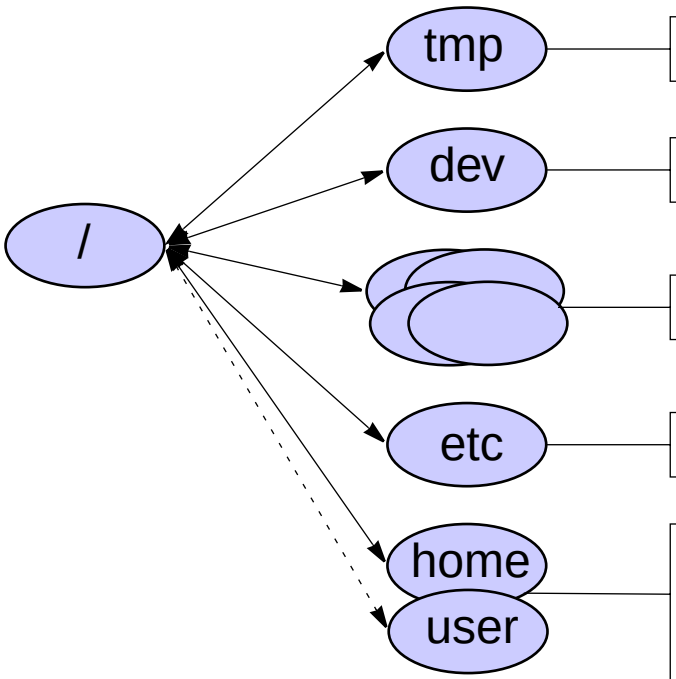
Structuration d'un système de fichiers (3)

En plus de la notion de répertoire, la plupart des systèmes permettent également de définir des **liens symboliques** vers des fichiers ou des répertoires (*soft/hard links* avec *Unix*, *raccourcis* (~) avec *Windows*), qui permettent d'une part de définir des **alias**, et d'autre part d'**assouplir la structure d'arbre**.



Arborescence UNIX et *Home Directory*

Un système *Unix* contient, en plus de la racine,
un certain nombre de répertoires plus ou moins standards
(c'est-à-dire présents, avec quelques variations, sur tous les systèmes):[1]



/tmp: répertoire des fichiers temporaires (local à chaque machine)

/dev: répertoire des périphériques systèmes

/bin, /sbin, /urs/bin. /usr/sbin: répertoires des commandes et utilitaires

/etc: administration système

☞ Chaque utilisateur possède un ***répertoire personnel*** (*home directory*, ou *répertoire d'utilisateur*).

C'est en fait la racine d'un sous-arbre administré par l'utilisateur, et dans lequel il peut placer ses fichiers personels.

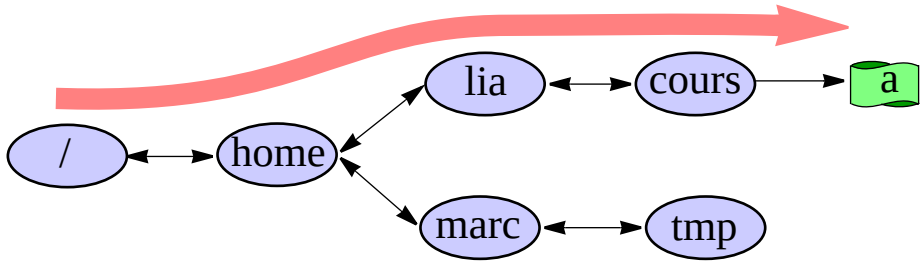
1. Certains de ces répertoires ne sont que logiques (pas physiquement présents sur les disques). C'est notamment le cas de /proc, qui contient une images des processus en cours d'exécutions.

Nommage des fichiers: absolu et relatif

On appelle «***chemin***» la succession des répertoires conduisant à un fichier, à partir d'un endroit donné dans l'arborescence. Par extension, le *chemin* désigne la **succession des répertoires et le nom du fichier lui-même**.

Pour désigner un fichier, il est possible de procéder de deux manières:

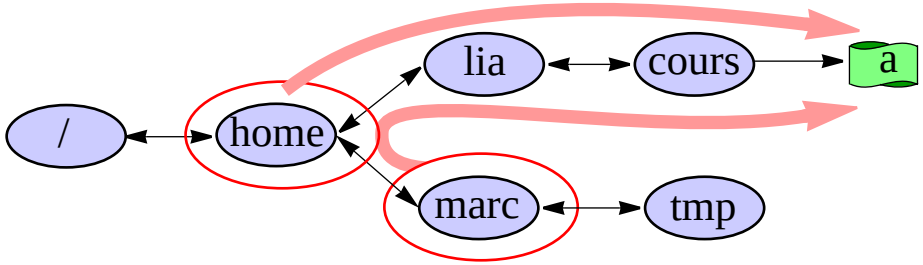
- à l'aide d'un **chemin absolu**:
  on prend comme convention un parcours de l'arbre partant de la racine (dans le cas d'une forêt, le nom du lecteur (i.e. la désignation de la racine de l'arbre) doit être tout d'abord spécifié.
  *(**Analogie**: «Av. des Morgines 18, CH-1213 Petit-Lancy»)*



/ + home + lia + cours + a

- à l'aide d'un **chemin relatif**:
  c'est la succession des répertoires à traverser, à partir d'un répertoire donné de l'arborescence (autre que la racine)
  *(**Analogie**:*
  «tout droit, puis à gauche au carrefour, 2e immeuble, 3e allée»)

lia + cours + a



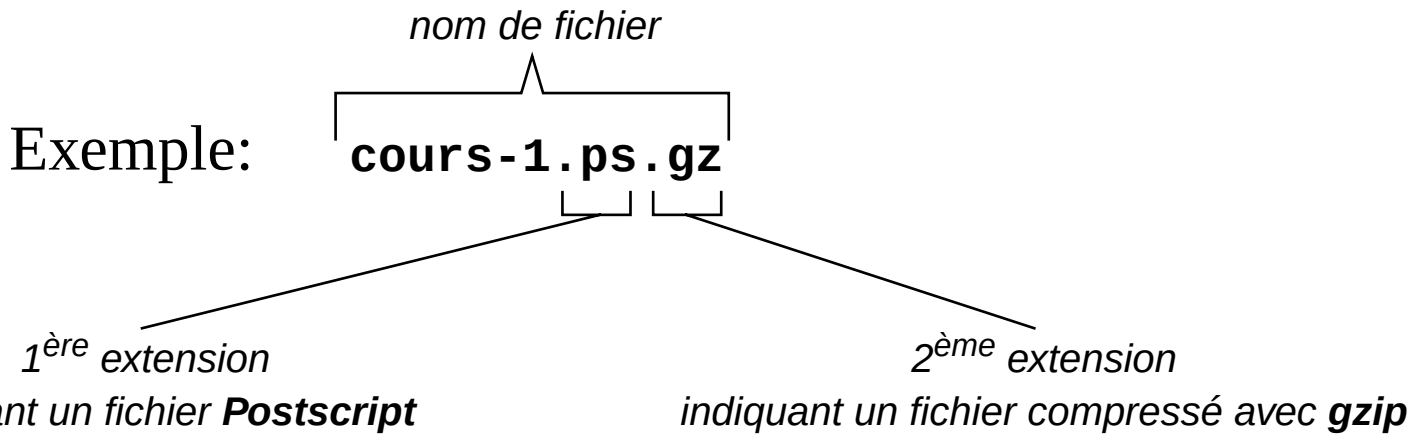
.. + lia + cours + a

Système de fichiers UNIX (1)

Sous *Unix*, le nom d'un fichier est composé d'au plus 255 caractères.

Les fichiers sont généralement (**mais il n'y a aucune obligation**)
munis d'une extension, délimitée par un point («.»).
Cette extension peut être utilisée pour indiquer la nature du fichier,
c'est-à-dire **l'application à laquelle il est associé**.

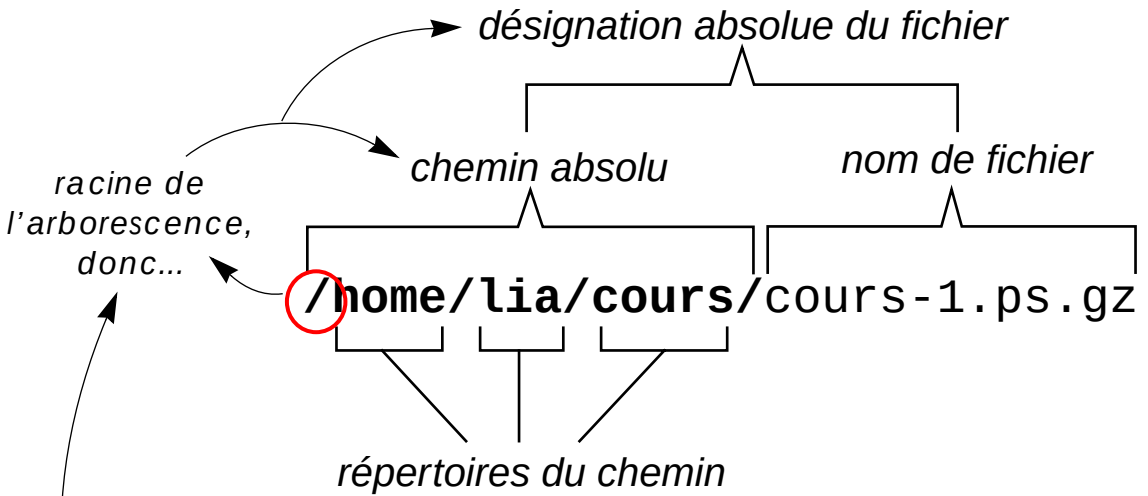
Contrairement à *Dos*, sous *Unix* les fichiers peuvent avoir
0, 1 ou plusieurs «extension».



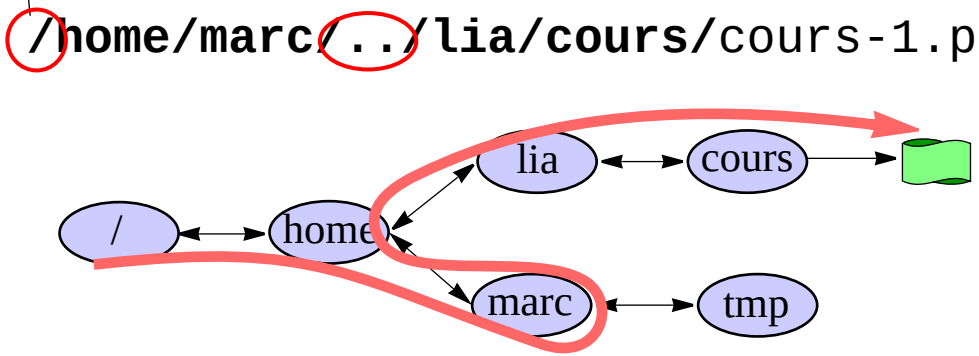
Système de fichiers UNIX (2)

Dans les *chemins*, le délimiteur entre nom de répertoire et nom de fichier est la barre oblique *slash* «/»

- Le slash est également le nom **de la racine** de l'arborescence Unix:



- Le répertoire parent d'un sous-répertoire est désigné par deux points («..»), tandis que le répertoire lui-même par un point («.»)

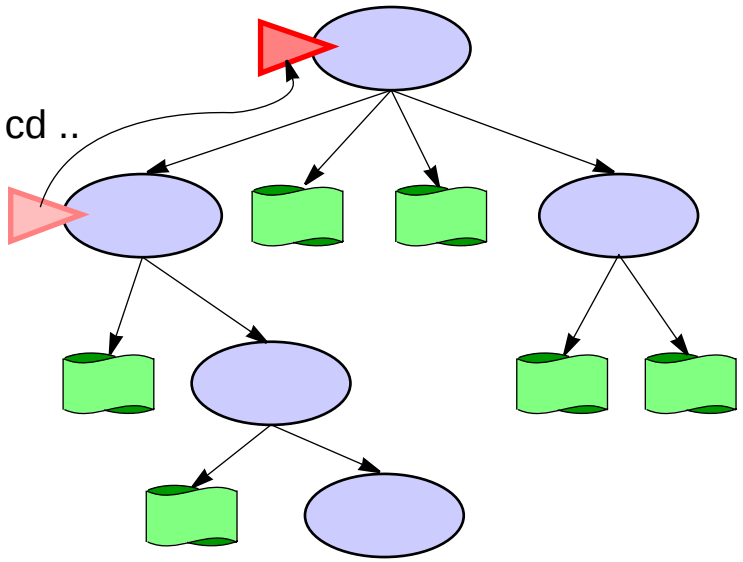


Fichiers et shell (1)

Un certain nombre des fonctions du *shell* sont relatives au système de fichiers.

Le *shell* a ainsi la responsabilité de:

- **Permettre la navigation dans la structure des fichiers**: il définit pour cela la notion de *répertoire* ***courant***, une commande de modification de ce répertoire (`cd` = *change directory*), et un ensemble d'autres permettant de lister le contenu d'un répertoire (`ls`), de copier des fichiers (`cp`), de les déplacer (`mv`), les effacer (`rm`), faire des liens (`ln`), etc. Toutes les commandes soumises au *shell* sont interprétées relativement à ce répertoire courant.



- **Permettre l'utilisation de jockers**: lors du nommages des fichiers et répertoires, le point d'interrogation «?» peut être utilisé comme substitut de **1** caractère, tandis que l'étoile sert à substituer une chaîne de caractères (**n** éléments).

`cours-?.ps.gz`

```
cours-1.ps.gz
cours-2.ps.gz
     ...
cours-Z.ps.gz
```

`cours-1*`

```
cours-1.ps.gz
cours-1.pdf
     ...
cours-1-temp.xyz
```

Attributs de fichiers

Les attributs typiques d'un fichier sont:

- son nom,
- sa taille,
- la date et heure de création
- modifiable, exécutable, caché,...
- système, possédant des alias,...
- le propriétaire (créateur)
- les droits d'accès des autres utilisateurs
- ...
- la date et l'auteur de la denière révision,
- no de version (système VMS),...

Dans le cas du système Unix:

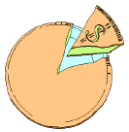
- On distingue les fichiers cachés au moyen d'une convention de nommage: ils sont préfixés par un «.»
- A chaque fichier est associé un utilisateur propriétaire (le créateur du fichier) et un groupe propriétaire (l'un des groupes auxquels appartient l'utilisateur)
- Les droits d'accès définissent 3 attributs, et sont paramétrables pour 3 classes d'utilisateurs:

Attributs
Visibilité (lecture)
Modification (écriture, effacement)
Exécution

Classes d'utilisateurs
Propriétaire (owner ou user)
Groupe (group)
Autres (others)

Quotas

Dès lors que plusieurs personnes travaillent ensemble, et utilisent une ressource commune, il faut mettre en œuvre des mécanismes assurant qu'un utilisateur **ne monopolise pas cette ressource au détriment des autres**.



C'est également le cas avec les disques durs.

☞ Le «***quota disque***» représente l'espace qu'un utilisateur peut occuper sur le disque contenant son répertoire personnel (*home directory*).

C'est en fait l'espace mémoire maximum autorisé pour la somme des tailles des fichiers stockés dans le répertoire personnel de l'utilisateur.[1]

Lorsqu'un utilisateur a «atteint son quota»[2], le système refusera toute tentative conduisant à une augmentation de l'espace utilisé. Par ailleurs, l'utilisateur ne pourra plus démarrer de sessions en tant qu'utilisateur local.

Pour vérifier votre utilisation d'espace disque,
utiliser la commande: **`quota -v`**

1. En effet, les élements hors de ce sous-arbre (lecteur de disquettes, /tmp, ...) ne sont pas pris en compte dans le calcul de l'espace occupé. Par contre, leur usage peut également parfois être restreints par le biais de quotas.
2. En réalité, le quota disque est dépassable, dans une faible proportion, pour un court laps de temps (typiquement, +10% pendant 7 jours maximum)

Les principaux domaines de l'informatique

... abordés dans le cadre de ce cours:

- La Programmation
- Les Systèmes d'Exploitation

- Les Systèmes d'Information
- La Conception d'Interfaces
- Le Calcul Scientifique
- Les Agents Logiciels

Systèmes d'information: SGBD (1)

Lorsque les données à gérer informatiquement
par une entité (entreprise, université, association, ...)
sont de nature diverses et possèdent de nombreux liens entre elles,
les fonctionnalités fournies par un système de fichiers ne sont plus suffisantes; il
convient de faire alors appel à des fonctions de
gestion d'information plus sophistiquées; fournies par
un ***système d'information***.

L'exemple le plus courant de système d'information sont les
Systèmes de Gestion des Bases de Données (SGBD).

Sytèmes d'information: SGBD (2)

Un SGBD permet de gérer l'ensemble des informations nécessaires à la réalisation d'un **objectif** (tâche) **commun** au sein d'une entreprise ou de toute autre collectivité d'individus travaillant en coordination.

Exemples de tâches:

gestion des étudiants d'une université,

gestion des réservations des places d'avions,

gestion de comptes bancaires, ...

Système d'information: SGBD (3)

Exemple 1:

Catalogue de produits vendus par une entreprise:

Base de données relativement simple

⇒ peut être gérée directement (données stockées dans un simple fichier, une feuille de tableur, ...)

Exemple 2:

Gestion des cours et étudiants d'une université:

Données beaucoup plus complexes, car faisant intervenir des informations diverses, **liées entre-elles**:

- informations de type académique, sur les étudiants (matricule, date d'inscription, section, notes, ...)
- informations de type personnelles, sur les étudiants (nom, prénom, adresse, no AVS, ...)
- informations sur les cours dispensés (titre, prérequis, matière, langue, enseignant, horaire, salle, ...)
- informations sur les enseignants (nom, prénom, bureau, téléphone, statut, no AVS, ...)
- informations sur les cours dispensés (titre, matière, langue, enseignant, horaire, salle, ...)
- ...

⇒ Ensemble de données trop complexe pour être géré «manuellement»: il faut faire appel à un système d'information.

Cycle de vie d'une base de données

Le **cycle de vie** d'une base de donnée (BD) se décompose en trois phases

- La ***conception***: définition des fonctionnalités,
- L'***implantation***: réalisation effective de la base,
- L'***exploitation***: utilisation et maintenance de la base.

Cycle de vie d'une BD: la conception

La **phase de conception** est une phase d'analyse,
qui aboutit à **déterminer le futur contenu** de la BD.

L'ensemble des concepteurs et des utilisateurs potentiels doivent se mettre d'accord sur la nature et les caractéristiques des informations qui devront être manipulées.

La description obtenue (qui ne fait généralement référence à aucun système de SGBD particulier) utilise un **langage formel** basé sur des concepts bien établis, comme les *objets*, les *liens* et les *propriétés*. Cette description est appelée:

schéma conceptuel *(des besoins*).

L'ensemble des concepts utilisés par le langage formel de description choisi est appelé le ***modèle conceptuel***.

Cycle de vie d'une BD: schéma conceptuel

Un **schéma conceptuel** se décompose généralement en deux parties:

- Une partie ***statique*** décrivant la structure des données;
- Une partie ***dynamique*** décrivant les opérations sur les données

De plus, il faudra pouvoir décrire des ***contraintes*** (règles) pesant sur les données (comme par exemple: «il ne doit pas y avoir plus de 20% d'écart entre les salaires des employés d'une même service et d'une même catégorie»).

Ces contraintes, appelées ***contraintes d'intégrité***, seront décrites dans un langage permettant d'exprimer des règles compatibles avec le modèle conceptuel choisi.

Le modèle conceptuel illustré dans le cours est le modèle ***entité-association.***

Cycle de vie d'une BD: implantation

La **phase d'implantation** est une phase qui consiste:

- à définir le modèle choisi pour la base de données (à l'aide d'un langage symbolique de description des données –LDD– spécifique au SGBD choisi), et

- à entrer les premières données, i.e. contruire une première version de la BD.

Cycle de vie d'une BD: schéma logique

La phase d'implantation nécessite
la **traduction du *schéma conceptuel*** dans un schéma utilisant
les concepts du modèle employé par le SGBD (appelé ***modèle logique***).

Le nouveau schéma obtenu est appelé

le **schéma logique**

(ou aussi quelquefois [malheureusement] «schéma conceptuel»).

Le modèle logique illustré dans le cours est le ***modèle relationnel.***

Cycle de vie d'une BD: schéma interne

Pour l'implémentation effective des données,
il faut encore effectuer les choix relatifs à leur stockage et leur structuration sur les mémoires secondaires, sous la forme d'un **ensemble de fichiers**.

Ces choix sont consignés dans ce qu'on appelle le ***schéma interne*** de la base de données, qui repose sur le ***modèle interne***, dont les concepts sont ceux du **système de fichiers** utilisé.

Cycle de vie d'une BD: exploitation

En phase d'exploitation, l'utilisation de la BD se fait au moyen d'un ***langage de manipulation de données*** –LMD– qui permet d'exprimer aussi bien des *requêtes d'interrogation* (pour obtenir des informations contenues dans la base) que des *requêtes de mise à jour* (pour modifier le contenu de la base).

Le langage de manipulation de données illustré dans le cours est:

SQL (*Structured Query Langage*) , version 92.

Cycle de vie d'une BD: schémas externes

Lors de son interaction avec la BD,
chaque utilisateur (ou groupe d'utilisateurs) n'est généralement intéressé
que par **une partie des données stockées** dans la base.

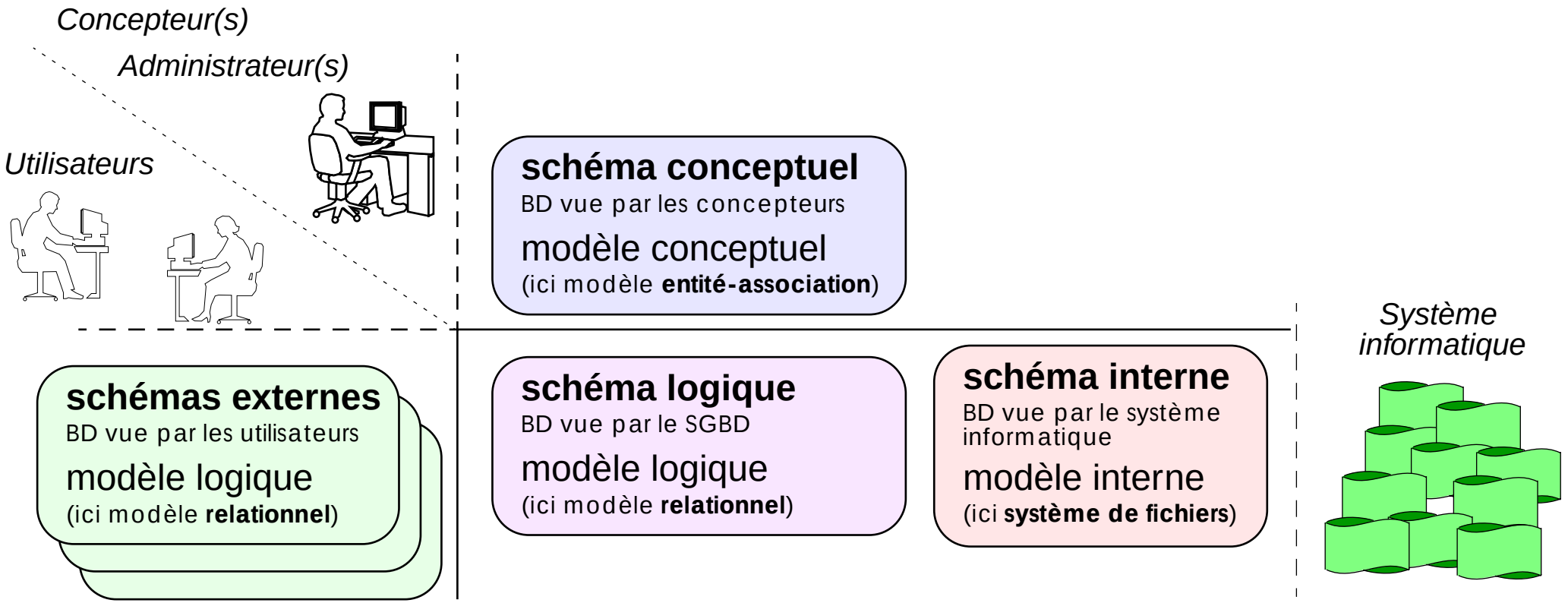
On lui associe donc un ***schéma externe*** (aussi appelé ***vue***)
décrivant le sous-ensemble de la base auquel il a accès,
structuré de façon à répondre à ses besoins spécifiques.

Dans les SGBD actuels, le modèle utilisé pour décrire les schémas externes
est le même que celui du schéma logique.

Les schémas d'une base de données

La description complète d'une base de données est donc réalisée à l'aide de 4 types de schémas, dont 3 sont directement utilisés par le SGBD.

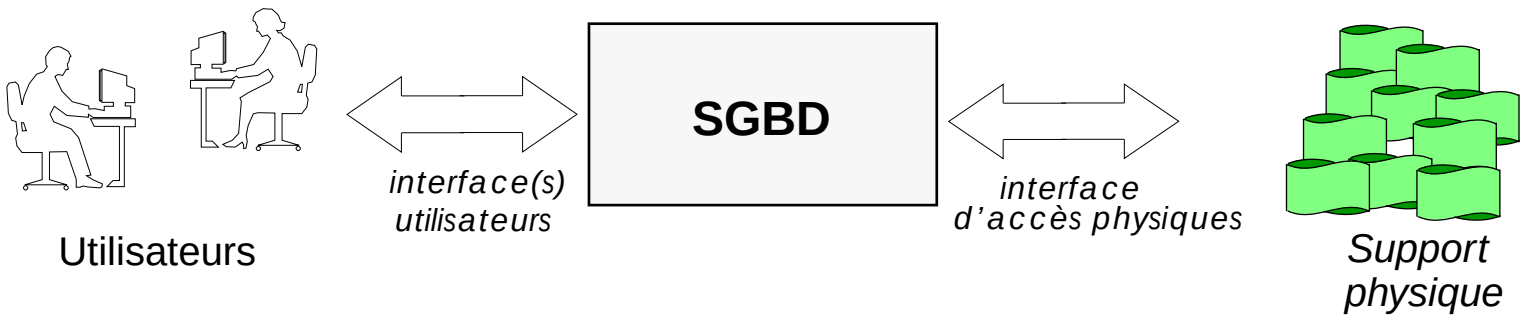
Il sont organisés de la façon suivante:



Principe de fonctionnement d'un SGBD (1)

L'objectif fondamental de l'organisation d'un SGBD est d'assurer **l'indépendance programmes/données**:

- D'une part, un utilisateur doit pouvoir modifier sa vue de la base, sans avoir à se soucier des choix opérés au niveau interne en matière de fichiers;
- d'autre part, un administrateur de système doit avoir la possibilité de modifer ces choix sans que cela ait un impact sur les utilisateurs



De plus un SGBD étant utilisé simultanément par plusieurs utilisateurs, il doit pouvoir résoudre les problèmes internes de coordination des actions, de cohérence (intégrité) des données, et contrôler le bon déroulement continu de ses activités.

Principe de fonctionnement d'un SGBD (2)

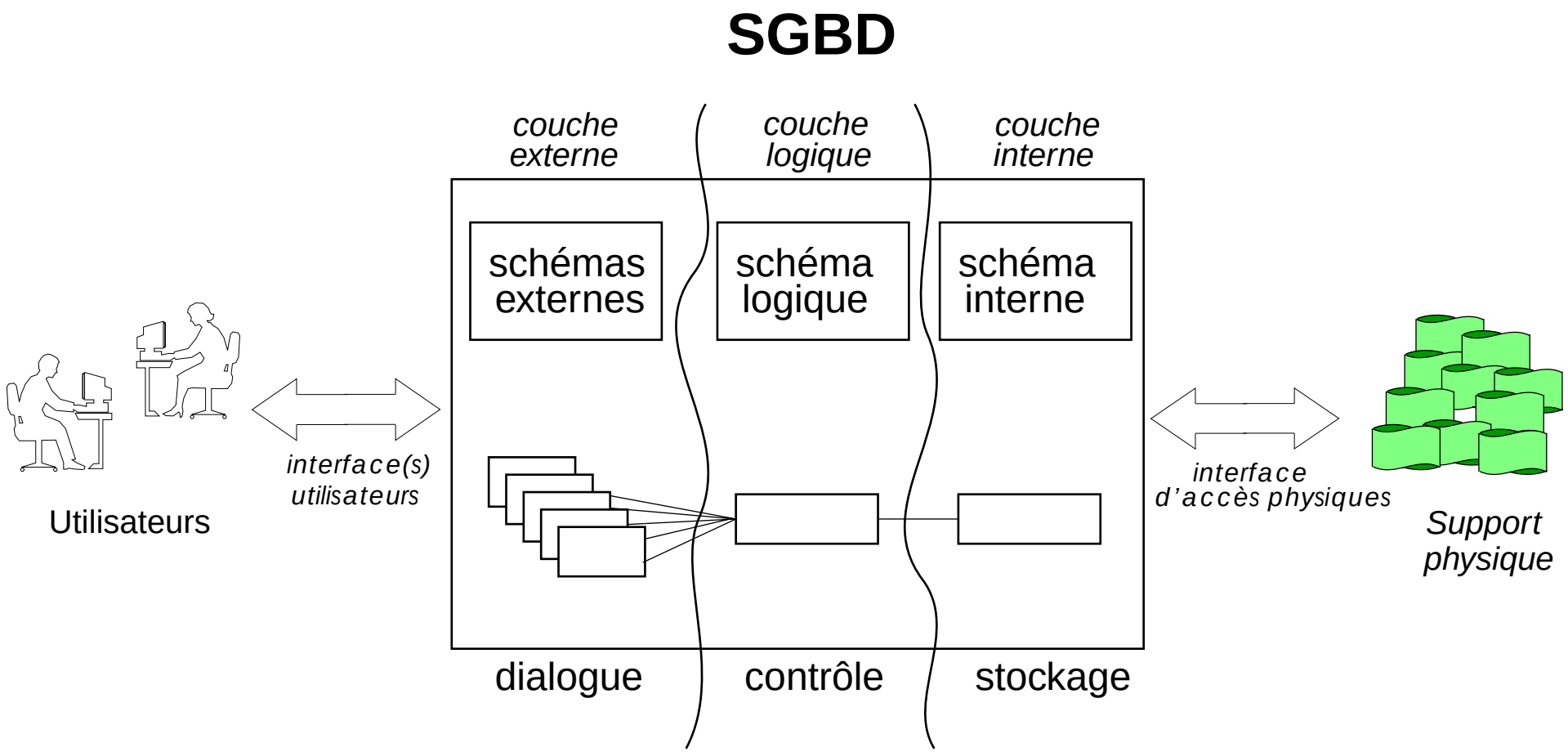
En conséquence, un SGBD est habituellement organisé en trois couches :

- La ***couche externe***:
  qui prend en charge l'interface avec les utilisateurs
  (analyse des requêtes –interrogation, modification de la BD–,
  contrôle des droits d'accès, présentations des résultats, ...)

- La ***couche intermédiaire*** (ou ***logique***):
  qui assure les fonctions de contrôle global
  (optimisation des requêtes, gestion des conflits d'accès, contrôle de la cohérence globale de la base, garantie du bon déroulement des actions en cas de panne, ...)

- La ***couche interne***:
  qui s'occupe du stockage des données sur les supports physiques et de la gestion des fichiers et des accès (index, clés, ...).

Principe de fonctionnement d'un SGBD (3)



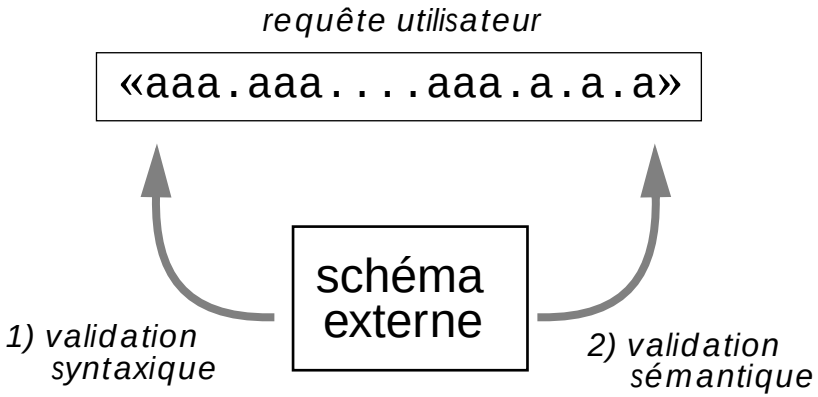
Principe de fonctionnement d'un SGBD (4)

Avec la structuration qui vient d'être définie,
le principe de fonctionnement d'un SGBD est le suivant:

1. une requête, exprimée par l'utilisateur dans le LMD accepté par le SGBD est d'abord

   validée du point de vue **syntaxique** (conformité à la grammaire du langage)

2. puis validée du point de vue **sémantique** (les objets cités doivent être connus dans le schéma externe de l'utilisateur).



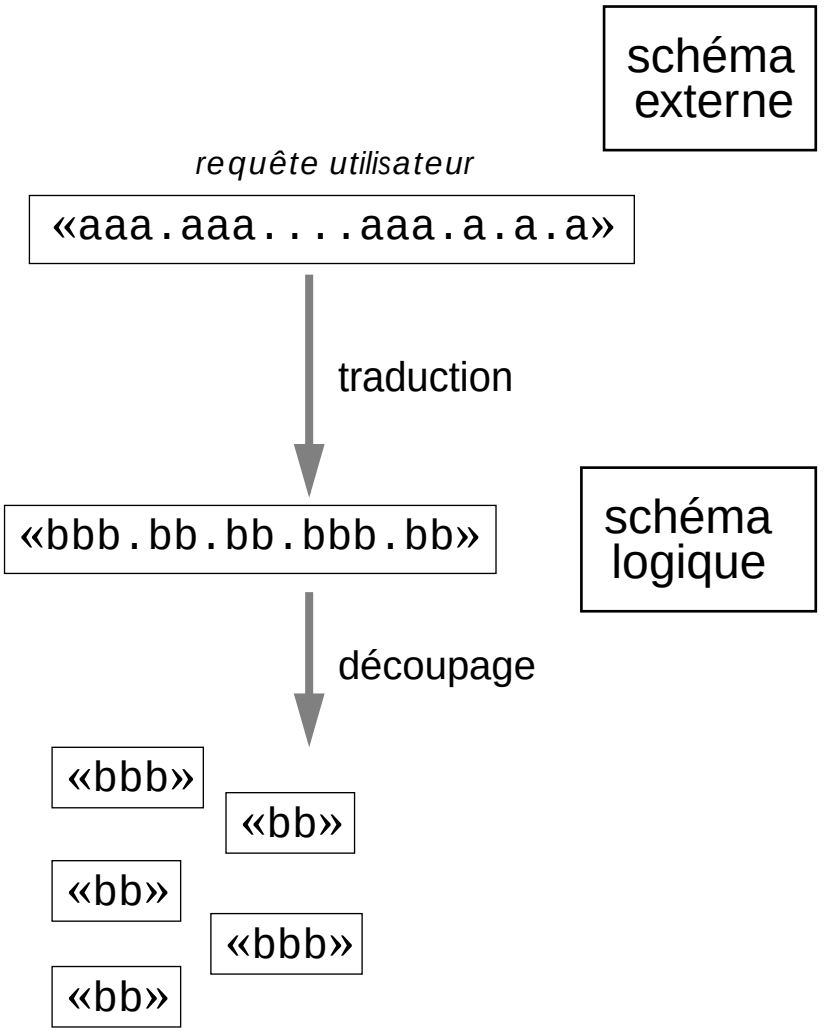
Principe de fonctionnement d'un SGBD (5)

3. Après la validation faite dans la couche externe, on utilise les règles de correspondance entre schéma externe et schéma logique (établies au moment de la définition du schéma externe) pour:

   traduire la requête dans le modèle logique.

   Cette traduction se fait dans la couche logique, et est accompagnée des contrôles sur la confidentialité, la concurrence d'accès, ...

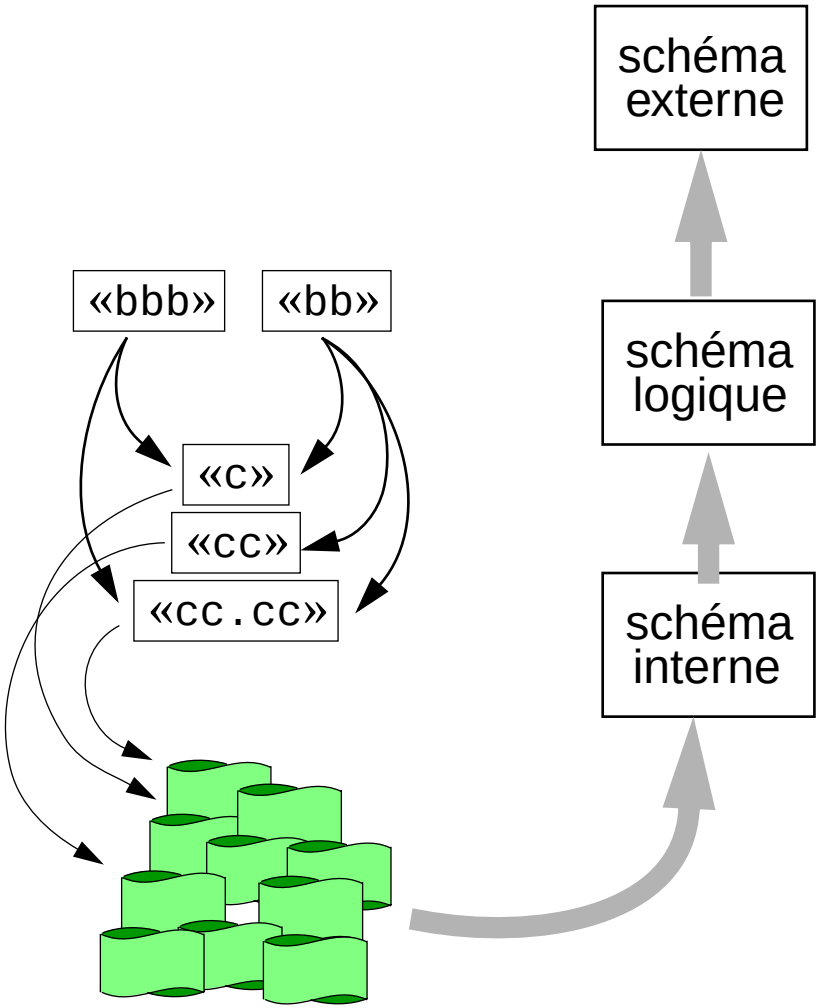
4. Si la requête est acceptée, elle est optimisée puis **découpée en sous-requêtes plus élémentaires**, qui sont transmises à la couche interne;
   sinon elle peut être mise en attente ou refusée.



Principe de fonctionnement d'un SGBD (6)

5. Au niveau interne, chaque sous-requête est **traduite en une ou plusiers requêtes physiques**, en fonction des informations contenues dans le schéma interne;
   le SGBD réalise ensuite l'accès physique aux données (extraction ou modification).

6. Les éventuelles données extraites sont passées à la couche logique, puis à la couche externe, où elles sont ré-organisées en fonction du schéma externe de l'utilisateur.



Modèle «entité-association» (1)

Le **modèle *entité-association***
(aussi appelé **modèle *entité-relation***)
est le modèle conceptuel de description statique
utilisé dans la plupart des méthodes et outils d'aide à la
conception de base de données (MERISE, IDA, ...)

Les concepts de base de ce modèle sont

- Les ***entités***
- Les ***associations***
- Les ***attributs***

Modèle «entité-association» (2)

Les ***entités*** permettent de représenter les objets (concrets ou abstraits) du monde réel à propos desquels on veut enregistrer des informations.

Les entités sont structurées par le biais de **types** qui regroupent des ensembles d'entités perçues comme similaires et ayant les mêmes caractéristiques.

Les ***associations*** permettent de représenter les liens entre entités, liens au sein desquels les entités peuvent jouer des ***rôles*** spécifiques (agent, objet, producteur, ...).

De même que les entités, les associations peuvent être typées, i.e. regroupées en ensembles homogènes d'associations (associations liant des entités de même type, avec les mêmes rôles et possèdant les mêmes propriétés).

Les ***attributs*** permettent de représenter des propriétés associées à un type d'entité (TE), un type d'association (TA) ou participant à la définition d'un attribut complexe.

L'ensemble des attributs d'un TE ou TA représente l'ensemble des informations inhérentes que l'on souhaite conserver sur les entités ou les associations.

Modèle «entité-association» (3)

Exemples d'entités:

- un étudiant
- un enseignant
- un cours

Exemples d'associations:

- suivre (un *étudiant* suit un *cours*)
- dispenser (un *enseignant* dispense un *cours*)

Exemples d'attributs:

- nom, prénom, adresse,
- no de téléphone, no bancaire,
- cycle, matricule, nombre d'heures
- date de naissance, horaire,
- statut, état civile,...

Identifiants

Un ***identifiant*** de TE (respectivement de TA) est un **ensemble minimum d'attributs** tel qu'il n'existe pas deux occurrences de TE (respectivement de TA) partageant **simultanément** les mêmes valeurs pour cet ensemble d'attributs.

Un identifiant permet donc de référencer de façon **non ambiguë** une occurrence de TE ou de TA.

Par exemple, le **matricule** peut servir d'identifiant pour les *étudiants*

Attributs

Un ***attribut*** est décrit par les spécifications suivantes:

- Le nom (unique) de l'attribut
- Définition sous forme de texte libre
- Ses ***cardinalités*** **minimale** et **maximale**, i.e. les nombres minimum et maximum de valeurs autorisées pour cet attribut dans une occurrence de TE ou de TA, ou dans l'attribut dont il est un composant.
- Si l'attribut est ***composé***, la description de ses attributs composants; sinon (i.e. l'attribut est ***simple***) son ***domaine de valeurs*** (l'ensemble des valeurs autorisées)

Exemple:

Attribut «Date de naissance»	
nom	date de naissance
définition	Indique les jour, mois et année de naissance d'une personne
cardinalités	min=1, max=1 *(toute personne à exactement une date de naissance)*
type:	jour *(fait l'objet d'une définition séparée)*
	mois *(définition séparée)*
	année *(définition séparée)*

Interprétation des cardinalités

min = 0	l'attribut est **facultatif** *(exemple: un numéro de téléphone privé)*	***max = 1***	l'attribut est **monovalué** *(exemple: une date de naissance)*
min = 1	l'attribut est **obligatoire** *(exemple: une date de naissance)*	***max = n***	l'attribut est **multivalué** *(exemple: les prénoms d'une personne)*

Récapitulatif des types d'attributs:

- *attribut simple:* (type = ***simple***)
  un attribut qui n'est pas composé (et est donc associé à un domaine de valeurs).
  Exemple: salaire (encore que...), couleur, ...
- *attribut complexe:* (type = ***composé***)
  un attribut composé
  Exemple: date (jour+mois+année), ...
- *attribut monovalué:* (max = 1)
  un attribut qui ne peut prendre qu'une seule valeur par occurrence
  Exemple: date de naissance, nom, ...
- *attribut multivalué:* (max = n)
  un attribut qui peut prendre plusieurs valeurs par occurrence
  Exemple: prénom, n° de téléphone, ...
- *attribut facultatif:* (min = 0)
  un attribut qui peut ne pas prendre de valeur dans une occurrence
  Exemple: salaire, n° de téléphone, ...
- *attribut obligatoire:* (min = 1)
  un attribut qui doit prendre au moins une valeur par occurrence
  Exemple: nom, couleur, ...

Entités

Un ***type d'entité*** (TE) est décrit par les spécifications suivantes:

- Le nom (unique) du TE
- Une définition sous la forme d'un texte libre
- La description des éventuels attributs du TE
- La composition des éventuels identifiants du TE.

Exemple:

Type Entité «cours»	
nom	cours
définition	regroupe les informations gérées par le SAC sur les cours dispensés à l'EPFL
attributs:	attribut-1: nom_cours
	attribut-2: cycle_section
identifiant	attribut-1+attribut-2

Associations

Un ***type d'association*** (TA) est décrit par les spécifications suivantes:

- Le nom (unique) du TA
- Une définition sous la forme d'un texte libre
- Les noms des TE participant au TA, avec le nom du rôle les associant au TA
- Pour chaque rôle, sa cardinalité minimale et maximale, i.e. le nombre min et max d'occurrences du TE qui peuvent, à un instant donné, être liés par ce rôle à une occurrence du TA
- La description des éventuels attributs du TA
- La composition des éventuels identifiants du TA.

Exemple:

Type association «suit»	
nom	suit
définition	définit les cours suivis par un étudiant sous la forme: *un étudiant suit un cours*
TE participants	<étudiant (1:n),>, <cours (0:n>
attributs:	attribut-1: identifiant de étudiant
	attribut-2: identifiant de cours
identifiant	attribut-1+attribut-2

Interprétation des cardinalités

min = 0	le rôle est **facultatif** *(exemple: un cours peut n'être suivi par aucun étudiant)*	***max = 1***	le rôle est **exclusif** *(un cours ne peut être donné que dans 1 seule salle)*
min = 1	le rôle est **obligatoire** *(un cours doit être donné par au moins 1 professeur)*	***max = n***	le rôle est **duplicable** *(un cours peut être suivi par plusieurs étudiants)*

Occurrence et population

On appelle ***occurrence*** d'un TE (respectivement d'un TA), toute entité (respectivement association) appartenant à l'ensemble décrit par le TE (respectivement le TA).

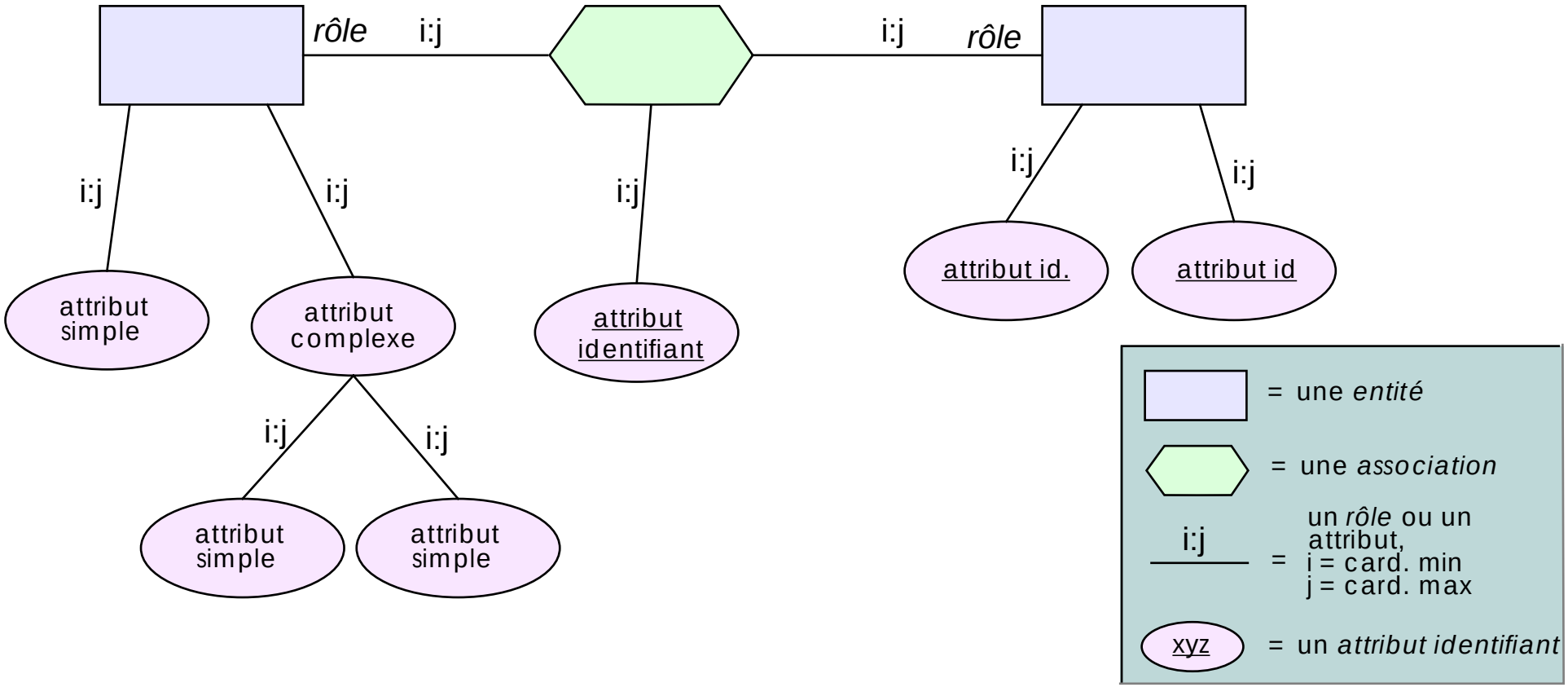
On appelle ***population*** du TE (respectivement du TA), l'ensemble des occurrences du TE (respectivement du TA).

La **base de données** décrite par un schéma entité-association est donc **l'ensemble des populations des TE et des TA** apparaissant dans le schéma.

Les définitions des TE et des TA peuvent, de plus, être structurées au sein d'une **hiérarchie de généralisations**.

Diagramme entité-association

Le modèle *entité-association* permet un **représentation graphique** [plus lisible] – appelée *diagramme* – du schéma d'une base de données.



Contraintes d'intégrité

Les concepts d'entités, d'associations et d'attributs
ne suffisent généralement pas pour décrire tout ce qui caractérise
les données associée à un schéma.

On peut être amené à exprimer des règles addtionnelles pour restreindre la
combinatoire des occurrences autorisées par la description statique.
De telles règles, souvent exprimées dans un formalisme inspiré
de la logique du 1er ordre sont appelées:
contraintes d'intégrité (CI)

Exemple de CI

Un cours c_1 ne peut pas être prérequis pour un cours c_2 s'il appartient à un cycle postérieur à celui de c_2:

$$\forall c_1 \in \text{Cours}, \forall c_2 \in \text{Cours}$$

$$(\text{Prérequis}\ (c_1, c_2) \Rightarrow c_2 \bullet \text{cycle} \geq c_1 \bullet \text{cycle})$$

Les arguments c1 et c2 de l'association «Prérequis(c1,c2)» ont ici respectivement pour rôle: «*est-un*» et «*pour*»
«Prérequis(c1,c2)» signifie donc: *c1 est-un Prérequis pour c2*

Les contraintes d'intégrité les plus fréquentes limitent les valeurs possibles d'un attribut
à certaines valeurs du domaine sous-jacent.

Modèle logique: modèle relationnel

Le **modèle relationnel** [inventé en 1960],
est actuellement le modèle logique le plus répandus
parmis les SGBD du marché.

Son principal avantage est sa **grande simplicité**
(ce qui lui permet d'être bien étudié sur le plan théorique, et facilement implantable) ...

... mais c'est également son principal défaut,
car sa simplicité en fait un **outil sémantiquement** trop **pauvre**
pour pouvoir correctement modéliser la complexité du monde réel
(pour cela, d'autres modèles plus sophistiqués ont été développés,
tels que les modèles orientés objets)

Constituants d'un modèle relationnel

Dans le modèle relationnel, les *types d'entités* et les *types d'associations* sont représentés par un concept unique: ***la relation***.

La relation représentant un TE ou un TA est un **tableau à deux dimensions**, usuellement appelé ***table***, dont les colonnes sont associées aux **attributs** du TE ou du TA, et les lignes correspondent aux **occurrences** (également appelées ***tuples***) du TE ou du TA.

Relation «Etudiant»

Matricule	***Nom***	***Prénom***	***Age***
136	Dupont	Jean	19
141	Dupond	Annie	20
245	Duval	Annie	19
341	Dumond	Marc	19
...	...	...	...

Usuellement, le ou les attributs identifiant de la relation sont soulignés.

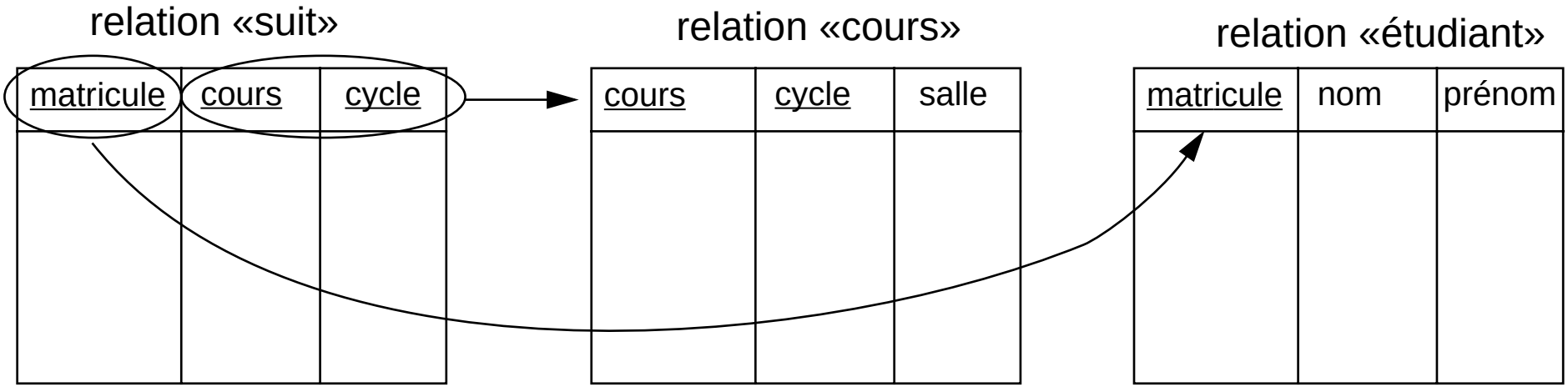
Identifiants et identifiants externes

Comme dans le cas du modèle entité-association,

un ***identifiant*** de relation est un **ensemble minimal d'attributs**,
tel qu'il n'existe pas deux tuples de la relation ayant des mêmes valeurs pour ces attributs.

Certains ensembles d'attributs d'une relation peuvent correspondre aux identifiants d'une autre relation.

Ces ensembles d'attributs sont alors appelés ***identifiants externes***.



Définition d'une relation

Une ***relation*** est décrite par les spécifications suivantes:

- le nom (unique) de la relation
- une définition, sous la forme d'un texte libre
- une liste d'attributs, chacun associé à un **domaine**
- le(s) **identifiant**(s)
- le(s) éventuel(s) **identifiant**(s) **externe**(s)

Exemple de domaines:

`Dnom`: *chaîne (caractères) de lng. maximal 30*

`Dnum`: *entier compris entre 0 et 99999*

`Dcol`: *{bleu, vert, rouge}*

Exemple:

Relation «étudiant»			
nom	Etudiant		
définition	ensemble des informations gérées par le SAC concernant les étudiants de l'EPFL		
attributs	matricule	***Domaine***	Dnum
	nom	***Domaine***	Dnom
identifiants	matricule		
id. externes	∅		

Population, schéma et contraintes

La ***population*** d'une relation est l'ensemble de ses tuples.

Le **schéma** (logique) d'une base de données relationnelle est l'ensemble des définitions de ses relations.

Contraintes imposées par le modèle relationnel:

- Les attributs sont tous simples et monovalué (les notions d'attributs complexes, multivalués ou facultatifs n'existent pas dans le modèle relationnel).
- Toute relation à nécessairement au moins un identifiant.

Règles de modélisation (1)

La **prise en compte des attributs complexes** du modèle entité-association peut se faire de trois façons:

- Les ***valeurs composites*** d'un attribut complexe sont considérées comme des **valeurs *atomiques*** (p.ex. de type chaîne de caractères); l'attribut est conservé, mais les valeurs de ses constituants ne sont plus individuellement accessibles
  (perte d'information structurelle)

date de naissance
"21 mars 1975"
"18 décembre 1961"
"21 mars 1975"

- L'attribut complexe est représenté par **plusieurs attributs simples** (i.e. plusieurs colonnes dans la table): les valeurs des constituants sont individuellement accessibles, mais l'attribut complexe n'existe plus en tant que tel
  (perte d'information structurelle)

Jour	Mois	Année
21	3	1975
18	12	1961
21	3	1975

- L'attribut complexe est représenté par une **relation** (aggrégation des attributs élémentaires). La table représentant l'attribut complexe possède un identifiant (propre), et l'ensemble des identifiants externes des tables matérialisant les attributs élémentaires): cela revient à expliciter une relation de type: *est-constituant-de*.
  (pas de perte d'information, mais complexification notable du schéma)

date de naissance
d1
d2
d1

<u>date</u>	Jour	Mois	Année
d1	21	3	1975
d2	18	12	1961

Règles de modélisation (2)

La **prise en compte des attributs multivalués** du modèle entité-association se fait par la **création d'une relation supplémentaire**, associant les valeurs d'un identifiant de l'entité ou de l'association décrite par l'attribut multivalué aux valeurs multiples de cet attribut.

Matricule	Nom	Prénoms
231	Dupont	Pierre, Claude
428	Durand	Claude, Jean

⇨

Matricule	Nom
231	Dupont
428	Durand

+

Matricule	Prénom
231	Pierre
231	Claude
428	Claude
428	Jean

Création du schéma relationnel (1)

La démarche générale est:

1. **Traduction du schéma *entité-association* en un schéma *relationnel***,
   en utilisant un algorithme de traduction, et les règles de modélisation précédentes.

2. ***Amélioration* éventuelle du schéma relationnel** ainsi obtenu,
   par une décomposition en relations en *troisième forme normale*
   (sans perte d'information ou de dépendances fonctionnelles).

Création du schéma relationnel (2)

L'algorithme de traduction est le suivant[1]:

(a) Pour chaque TE, créer une relation:

- dont le **nom** est le nom du TE
- dont les **attributs** sont les attributs monovalués du TE avec, comme nom, la concatenation du nom du TE et du nom de l'attribut du TE (par exemple, l'attribut `matricule` du TE `Etudiant` sera nommé `Etudiant.matricule`)
- dont **l'identifiant** est constitué des attributs identifiants du TE (si tous les identifiants sont multivalués, alors il faut créer un attribut identifiant spécifique supplémentaire pour la relation).

Remarque:

la relation de nom **`R`** décrite par les attributs `R.X1, ..., R.Xn` pourra être notée plus simplement **`R`**`(X1,...,Xn)`.

1. Pour simplifier, on ne considère ici que des attributs simples. les éventuels attributs complexes sont à modéliser en respectant les règles énoncées précédemment.

Création du schéma relationnel (3)

(b) Pour chaque TA, créer une relation:

- dont le **nom** est le nom du TA
- dont les **attributs** sont:
  - les attributs monovalués du TA avec, comme nom, la concatenation du nom du TA et du nom de l'attribut (par exemple, l'attribut `salle` de la relation `Suivre` sera nommé `Suivre.salle`)
  - les attributs identifiants des TE liés au TA avec, comme nom, la concatenation du nom du TE, du rôle du TE et du nom de l'attribut
- dont **l'identifiant** est constitué des attributs identifiants du TA.
- dont les **identifiants externes** sont les identifiants de TE liés (qui référencent les relations décrivant ces TE).

Création du schéma relationnel (4)

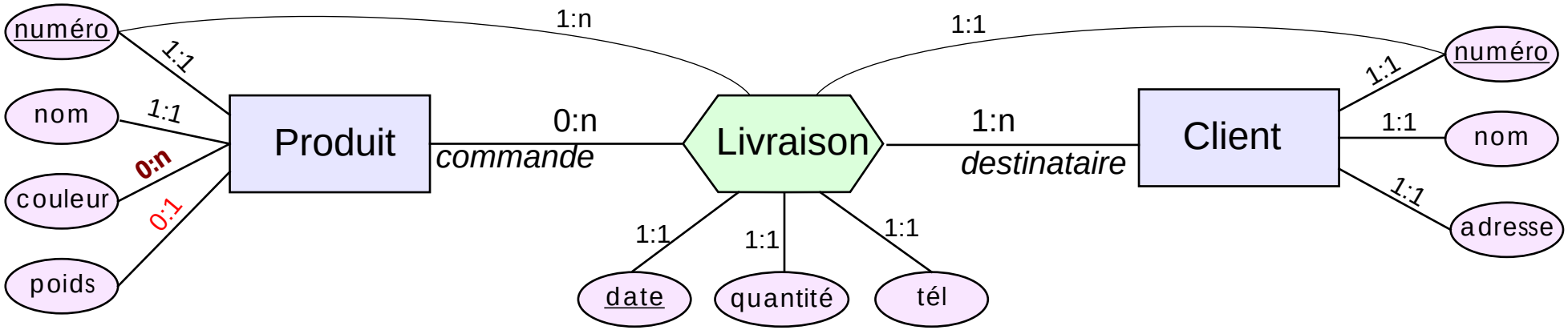
(c) Pour chaque attribut multivalué d'un objet O (TE ou TA), créer une relation

- dont le **nom** est le nom de l'objet O concaténé à celui de l'attribut
- dont les **attributs** sont l'attrbut lui-meme et les attributs identifiants de l'objet O
- dont **l'identifiant** est constitués de l'attribut et des attributs identifiants de l'objet O
- dont les **identifiants externes** sont les identifiants de de l'objet O (qui référencent la relations décrivant cet objet).

Les domaines de valeurs des attributs des relations sont les domaines de valeurs des attributs de TE ou de TA correspondants (plus, sauf pour les attributs des relation correspondant à des attributs multivalués, une ***valeur nulle*** **«*»** si l'attribut d'origine est **facultatif**)

Création du schéma relationnel (5)

Soit le schéma relationnel suivant:



En appliquant l'algorithme de traduction précédent, on obtient:[2]

```
(a)   Produit(numéro, nom, poids(*))
(c)   Produit.Couleur(couleur, Produit.numéro)
(a)   Client(numéro, nom, adresse)
(b)   Livraison(date, quantité, tél,
                Produit.commande.numéro,
                Client.destinataire.numéro)
```

«(*)» indique que l'attribut peut avoir une valeur nulle.

2. Afin de ne pas trop alourdir les définitions, les préfixes des noms d'attributs concaténés ne sont indiqué que si le préfixe diffère du nom de la relation.

Amélioration des schémas relationnels (1)

Considérons le schéma relationnel suivant,
décrivant les produits, les clients et les livraisons d'une entreprise :

```
Produit1(numéro, nom, couleur)
Client(numéro, nom, adresse)
Livraison(date, quantité, Client.destinataire.tél,
      Produit.commande.numéro,
      Client.destinataire.numéro)
Produit2(numéro, poids)
```

Ce schéma pose plusieurs problèmes:

- s'il n'y a plus de livraisons pour un client, son numéro de téléphone est perdu;
- il faut ressaisir le numéro de téléphone du client à chaque livraison et, de plus, vérifier s'il est cohérent avec l'information déjà saisie pour les autres livraisons;
- l'information concernant les produits et les clients est éparpilée dans plusieurs relations.

Le schéma n'est donc pas optimal et doit donc être **amélioré**.

Amélioration des schémas relationnels (2)

Une possibilité est:

```
Produit(numéro, nom, couleur, poids)
Client(numéro, nom, adresse, tel)
Livraison2(date, quantité,
      Produit.commande.numéro,
      Client.destinataire.numéro)
```

Ce schéma ne pose plus les problèmes mentionnés.
Il est donc meilleur que le précédent.

D'une façon générale,
les relations qui ne posent pas de problèmes
lors de l'insertion/modification/suppression des tuples
sont appelées des ***relations normalisées*** et le processus général
d'amélioration d'une relation ou de tout schéma relationnel
est appelé ***normalisation***.

Normalisation de schémas relationnels

La normalisation des relations peut être faite
en **découpant les relations posant problème** (i.e. *non normalisées*)
en **plusieurs relations *mieux formées*** (i.e. *normalisées*)
et décrivant la même information.

La normalisation d'un schéma correspondra alors
à une décompositions des relations non normalisées en relations normalisées,
suivie d'une recomposition des relations normalisées ainsi obtenues,
permettant un meilleur regroupement des informations reliées.

Bien sûr, toutes ces transformations devront se faire **sans perte d'information**.

Il existe plusieurs méthodes pour normaliser des schémas relationnels[3]
et pour décrire un exemple de telle méthode, nous allons avoir besoin des
concepts de ***dépendance fonctionnelle*** et de ***graphe fonctionnel***.

3. Notez cependant qu'aucune de ces méthodes, si elle est appliquée de façon purement automatique, n'est totalement satisfaisante, car on ne peut garantir que les relations normalisées produites seront sémantiquement signifiantes.

Dépendance fonctionnelle (1)

Etant donnée une relation **R** et *X* et *Y* deux attributs (ou ensembles d'attributs) de **R**, on dit qu'il existe une ***dépendance fonctionnelle*** (ou DF) de *X* vers *Y* si la propriété suivante est vérifiée :

> Si deux tuples quelconques de **R** ont les mêmes valeurs pour *X*, alors ils ont aussi nécessairement les mêmes valeurs pour *Y*

La dépendance fonctionnelle de *X* vers *Y* est notée: X→Y .
X (repectivement *Y*) est appelé la ***source*** (resp. la ***cible***) de la DF.

Exemple

Soit une description de produits manufacturé en terme de:
(1) type d'alimentation, (2) voltage, (3) norme de securité et (4) couleur.

Dépendances fonctionnelles *sémantiques*

`alim→volts`
`alim→norme`
`alim→volts,norme`

alim	volts	norme	couleur
non élec.	0	CE-010	bleu
pile	9	CE-125	bleu
non élec.	0	CE-010	rouge
secteur	230	CE-130	rouge
batterie	9	CE-125	bleu

Dépendances fonctionnelles *issues des données*

`volts,norme→alim`

La dépendance disparaît si on ajoute cette donnée

Dépendance fonctionnelle (2)

Si Y est réduit à un attribut unique et X est un ensemble minimal d'attributs pour **R** (i.e. $X = x_1, \ldots, x_k$ et il n'existe pas de sous-ensemble strict X' des x_i tel que $X' \rightarrow Y$), la dépendance fonctionnelle est dite **élémentaire**.

```
(1) NoProduit → CouleurProduit
(2) NoProduit → CouleurProduit, PoidsProduit
(3) NoProduit,CouleurProduit → PoidsProduit
```

Dans l'exemple ci-contre, (1) est **élémentaire**, tandis que (2) et (3) ne le sont pas.

Notion de graphe minimal des DF

Dépendance fonctionnelle déduite

Si, dans une relation **R**, on a les DF $X \rightarrow Y$ et $Y \rightarrow Z$, alors on a aussi la DF $X \rightarrow Z$ qui est dite DF ***déduite*** (des deux autres).

Une méthode pour déterminer si une DF $X \rightarrow Y$ d'une relation **R** est déduite est de vérifier si, après avoir supprimé la DF $X \rightarrow Y$ de l'ensemble des DF de **R**, Y peut encore être déduite de X.

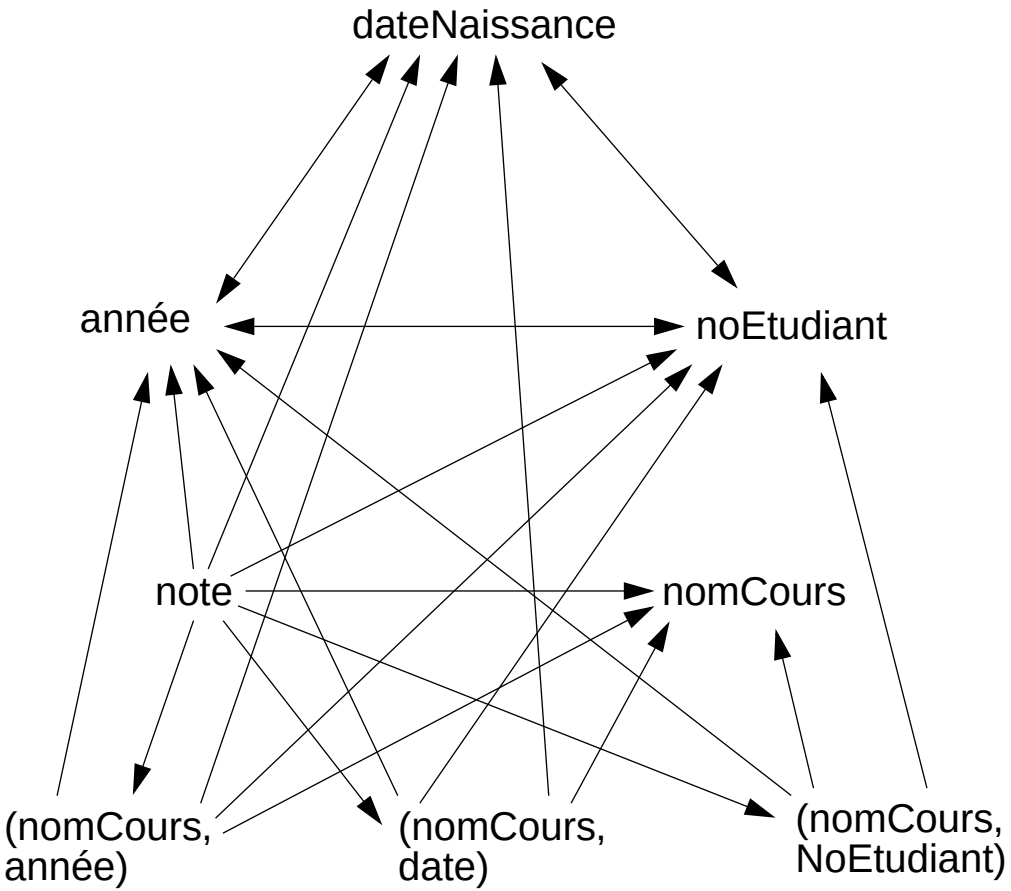
Graphe minimal des dépendances fonctionnelles

On appelle ***graphe minimal*** (des dépendances fonctionnelles) d'une relation **R** tout ensemble de DF élémentaires

- non déduites,
- dont toute DF élémentaire de **R** peut être déduite.

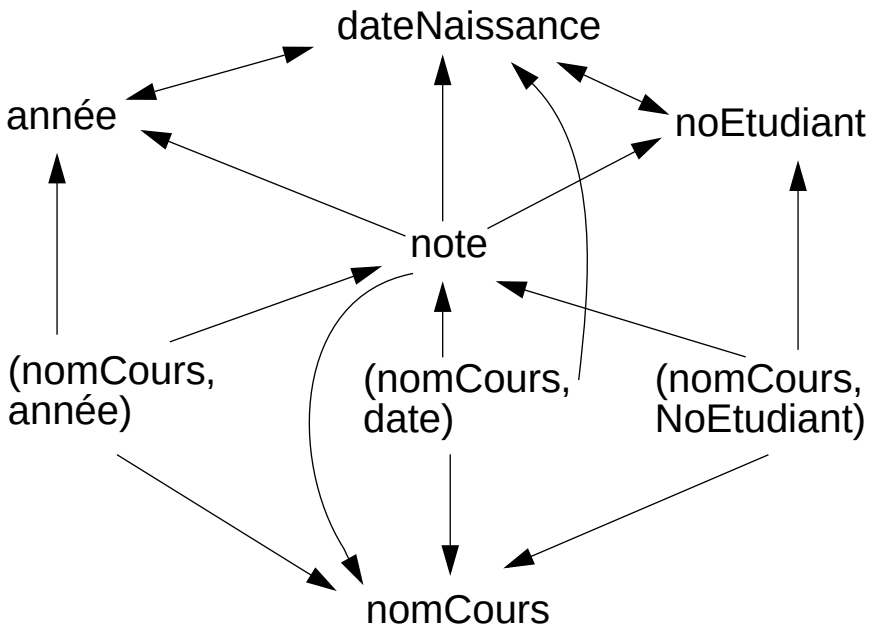
Exemple de graphe minimal des DF

Graphe complet



dateNaissance	noEtudiant	nomCours	note	année
3/5/59	22	algo	12	1988
3/5/59	22	C	13	1988
2/2/75	41	algo	10	1997

Graphe minimal



Méthode de synthèse

Exemple de méthode de normalisation d'un schéma relationnel (méthode dite ***de synthèse***):

A partir du schéma relationnel à normaliser :

(1) ⇨ **créer le graphe minimal des DF**

(2) ⇨ **créer**, pour chaque source de DF, **une relation** comprenant comme attributs la source et toutes les cibles de la source.

Cette méthode est intéressante, car elle est **sans perte ni d'information ni de DF**, et mène à une décomposition des relations en **relations normalisées** (en «*troisième forme normale*»).

Méthode de synthèse: conséquences

Les relations produites par cette méthode sont sous une forme particulière appelée ***relations normalisées*** en ***troisième forme normale***.

L'intérêt des relations sous cette forme est qu'elles ne posent pas de problèmes (intégrité des données) lors d'opérations d'insertion, modification ou suppression de tuples.

Exemple

`Livraison(Produit.commande.numéro, Client.destinataire.numéro, date, quantité, tél)`

La relation `Livraison` n'est pas en *troisième forme normale*, car:

- `(Produit.commande.numéro, Client.destinataire.numéro, date)` est *identifiant*,
- et l'on à la dépendance fonctionnelle: `Client.destinataire.numéro` → `tél`

Lorsque le client change de n° de téléphone, on est obligé, avec une telle représentation, de répercuter ce changement sur l'ensemble des tuples qui représentent une livraison pour ce client, ce qui n'est pas le cas avec la relation `Livraison2`, qui est elle *bien normalisée* (en ***troisième forme normale***)

`Livraison2(Produit.commande.numéro, Client.destinataire.numéro, date, quantité)`

(L'information sur le *téléphone* étant déplacée dans la relation concernant les *clients*.)

Méthode de synthèse: illustration

Soit la relation **R** suivante (**Livraison**)

```
R(date, quantité, tél,
   Produit.commande.numéro,
   Client.destinataire.numéro)
```

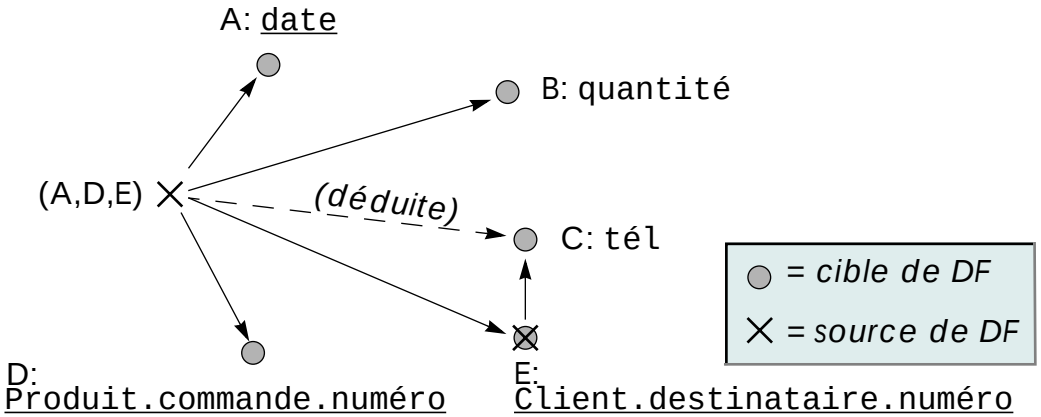
On a, par définition:

```
date,Produit.commande.numéro,
Client.destinataire.numéro → quantité,tél
```

Si on suppose de plus:
(sémantique de l'application ou analyse de données)

```
Client.destinataire.numéro → tél
```

(1) *Créer le graph minimal de **R**:*



(2) *Décomposition de la relation **R**:*

```
R =
   R1(date, quantité,
      Produit.commande.numéro,
      Client.destinataire.numéro)
+
   R2(Client.destinataire.numéro,
      tél)
```

Méthode de synthèse: justification (1)

Nous avons indiqué que la méthode de traduction/normalisation présentée est intéressante, car elle permet un **traduction** du *schéma conceptuel* en un *schéma relationnel* dont la normalisation (i.e. décomposition en relations normalisées) se fait **sans perte** d'information ni de dépendance fonctionnelle.

En effet:

(1) Par construction, toutes les relations produites vérifient les propriétés suivantes :

- tous leurs attributs sont simples et monovalués (conséquence du mécanisme de traduction);
- tous leurs attributs qui ne font pas partie d'un identifiant dépendent uniquement des identifiants entiers (conséquence de l'utilisation d'un graphe minimal de DF).

(2) la décomposition d'un schéma relationnel quelconque en relations en troisième forme normale, selon la méthode indiquée, se fait sans perte d'information ni de DF.

- sans perte d'information: conséquence du théorème de Heath;
- sans perte de DF: car on travaille sur le graphe minimal des DF.

Méthode de synthèse: justification (2)

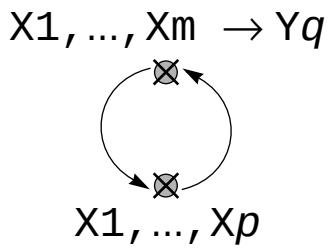
Justification de (1), par l'absurde:

Soit la relation **R**(X1,X2,...,Xm, Y1,...,Yn).

On a, par définition: (X1,...,Xm) → Y*i*, pour 1 ≤ *i* ≤ n

Supposons qu'il existe:

- *p* tel que *p*<m,
- et *q* tel que (X1,X2,...,X*p*) → Y*q* existe et soit dans le graphe minimal des DF.



Comme, par construction, (X1,...,Xm) → (X1,...,X*p*), on a :

Le graphe n'étant pas minimal, l'hypothèse d'existence de *p* et *q* n'est pas vérifiée.

Pour justifier (2) – la normalisation sans perte d'information – nous avons besoin de définir des opérations de **décomposition** et de **recomposition** de relations.
Plus précisement, on aura besoin d'opérations permettant:

- ☞ de décomposer une relation en sous-relations: la ***projection***
- ☞ de recomposer une relation à partir de sous-relations: la ***jointure naturelle***

Projection d'une relation

Etant donnée une relation **R** décrite par les attributs (X1,X2,...,Xm), on appelle ***projection*** de **R sur un sous-ensemble d'attributs** (Xi_1,...,Xi_n) de (X1,X2,...,Xm),

la relation décrite par les attributs (Xi1,...,Xi_n), et dont la population est l'ensemble des tuples de **R** tronqués à leurs valeurs pour (Xi1,...,Xi_n).

La projection de **R** sur (Xi1,...,Xi_n) sera notée: **R**(Xi1,...,Xi_n).
☞ Notons que **R** elle-même peut alors être notée **R**(X1,X2,...,Xn).

Exemples:

Produit(NoProduit, NomProduit)
Produit(NoProduit, NomProduit, CouleurProduit)

sont des projections de

Produit(NoProduit, NomProduit, CouleurProduit, PoidsProduit)

Jointure naturelle de deux relations

Etant données deux relations **R1**(X1,...,Xm,Y1,...,Yn) et **R2**(X1,...,Xm,Z1,...,Zo), partageant les attributs X1,...,Xm. On appelle ***jointure natuelle de R1 et R2*** la nouvelle relation **R**(X1,...,Xm,Y1...,Yn,Z1,...,Zo), définie comme suit:

Pour tout tuple t de **R**, il existe un tuple t_1 de **R1** et un tuple t_2 de **R2** tels que:

$\forall i \in [1,m]$, ☞ $t.X_i = t_1.X_i = t_2.X_i$
$\forall j \in [1,n]$, ☞ $t.Y_j = t_1.Y_j$
$\forall k \in [1,o]$ ☞ $t.Z_k = t_2.Z_k$

La *jointure naturelle* de **R1**(X1,...,Xm,Y1,...,Yn) et de **R2**(X1,...,Xm,Z1,...,Zo) sera notée **R1**(X1,...,Xm,Y1,...,Yn)×**R2**(X1,...,Xm,Z1,...,Zo)[4].

Exemple:

Produit(<u>NoProduit</u>, NomProduit)×Produit(<u>NoProduit</u>, CouleurProduit)

est la relation:

Produit(<u>NoProduit</u>, NomProduit, CouleurProduit)

4. Par définition, les tuples de **R1** (respectivement de **R2**) dont les valeurs pour les attributs partagés X1,...,Xm n'ont pas de correspondant pour **R2** (respectivement **R1**) n'apparaissent pas dans le résultat de la jointure.

Décomposition sans perte d'information

D'une façon plus générale, la normalisation d'un schéma relationnel, constitué des relations **R1**, ..., **Rm**, en l'ensemble de relations normalisées **R1'**, ..., **Rm'**, est dite «**sans perte d'information**» si on a:

$$\mathbf{R1} \times \ldots \times \mathbf{Rm} \quad = \quad \mathbf{R1'} \times \ldots \times \mathbf{Rn'}$$

On peut montrer, par le *théorème de* ***Heath***, que la méthode de normalisation par synthèse présentée dans ce cours vérifie cette propriété:

Etant donnée une relation **R**(X1,...,Xm,Y1,...,Yn,Z1,...,Zo), si (X1,...,Xm) → (Y1,...,Xn), alors **R** est décomposable *sans perte d'information* en **R1**(X1,...,Xm,Y1,...,Yn) et **R2**(X1,...,Xm,Z1,...,Zo), soit:

R(X1,...,Xm,Y1,...,Yn,Z1,...,Zo) = **R1**(X1,...,Xm,Y1,...,Yn) × **R2**(X1,...,Xm,Z1,...,Zo)

Exemple:

Hypothèse: Client.numéro → Client.tél

Livraison(Produit.numéro, Client.numéro, Date, Quantité, Client.tel)
est décomposable, **sans perte d'information**, en les 2 projections suivantes:

- Livraison1(Client.numéro, Client.tel)
- Livraison2(Produit.numéro, Client.numéro, Date, Quantité)

Structured Query Language

Les *langages de manipulation de données* : **SQL**

SQL est le *langage de manipulation des données* relationnelles le plus utilisé aujourd'hui.

C'est un **standard de fait** pour les systèmes de gestion de base de données relationnels.

Format de base d'une requête

La manipulation des données se fait
par le biais de ***requêtes***, soumise au SGBD, et
écrite dans le langage de manipulation des données.

C'est donc une forme de programmation,
avec un langage **interpêté**.

La requête utilisée le plus fréquemment est celle d'accès
aux données; sa syntaxe est:

```
(SELECT «Liste des noms d'attributs du résultat»
 FROM   «Nom d'une relation»
[ WHERE    «Condition logique qui définit les tuples du résultat» ]
[ ORDER BY «ordres de tri pour les tuples du résultat» )
```

Exemple

Considérons la table (relation) **R** associant
des numéros de fournsseurs (NP), des numéros de produits (NP),
des quantités livrées (Qte) et des dates de livraison (Date) :

R:

NF	NP	Qte	Date
1	1	2	11/10/1999
1	3	1	11/10/1999
1	3	1	10/10/1999
2	2	1	13/7/2000
2	2	5	1/10/1999
3	1	2	13/7/2000

☞ La requête :

```
(  SELECT NP, Qte
   FROM R
   WHERE Date = "13/7/2000"
   ORDER BY NP)
```

donne comme résultat la relation:

NP	Qte
1	2
2	1

☞ La requête :

```
(  SELECT NP, Qte, Date
   FROM R
   WHERE NF = 1
   ORDER BY NP, Date)
```

donne comme résultat la relation:

NP	Qte	Date
1	2	11/10/1999
3	1	10/10/1999
3	1	11/10/1999

Select et jocker «*»

La valeur *jocker* «*» dans le champ **SELECT** indique que la sélection porte sur **l'ensemble** des attributs de la relation..

Exemple:

La requête :

```
(   SELECT *
    FROM R
    WHERE Date = "13/7/2000"
    ORDER BY NP)
```

donne comme résultat la relation:

NF	NP	Qte	Date
2	2	1	13/7/2000
3	1	2	13/7/2000

Suppression des doublons

Pour les SGBD qui ne réalisent pas automatiquement la **suppression des *doublons*** dans le résultat, on peut utiliser le mot clé réservé «**DISTINCT**» dans le champ **SELECT**, pour forcer cette suppression.

Exemple:

La requête:

```
(   SELECT NP
    FROM R
    WHERE NF = 1)
```

peut produire comme résultat:

NP
1
3
3

Alors que la requête:

```
(   SELECT DISTINCT NP
    FROM R
    WHERE NF = 1)
```

produit nécessairement le résultat:

NP
1
3

Conditions dans le champ WHERE

Une ***condition de sélection*** apparaissant dans le champ WHERE de l'instruction SELECT peut être:

(1) Soit une condition élémentaire, de la forme:

- «*nom attribut*» «*opérateur de comparaison*» «*valeur attribut*»
  les opérateurs de comparaison définis étant:
  « = », « != » (ou « <> »), « < », « <= », « > », et « >= »
  Exemples: « `Qte < 3` », « `NP = 2` »

- «*nom attribut*» «*opérateur d'appartenance*» «*ensemble*»
  les opérateurs d'appartenance des ensembles sont:
  « `IN` », « `NOT IN` »
  Exemples: « `NP IN (1,2)` »

- «*nom attribut*» «*opérateur de comparaison ensembliste*» «*ensemble*»
  les opérateurs de comparaison ensemblistes sont:
  « `CONTAINS` », « `NOT CONTAINS` »

(2) Soit une condition complexe, composée à l'aide de conditions élémentaires et des connecteurs logiques «AND», «OR» et «NOT».
Exemple: « `(Qte > 2) AND (NP NOT IN (2,3))` »

Requêtes avec des blocs emboîtés

Lorsqu'une requête intègre elle-même **une autre requête** comme condition dans le champ SELECT, on parle de *requête avec **bloc emboîté***.

Dans le cas d'une requête avec bloc emboîté, le SGBD exécute d'abord la requête correspondant au bloc emboîté, mémorise temporairement le résultat, puis exécute la requête externe.

Dans ce cas, il peut également être nécessaire de **restreindre** la portée des noms des attributs utilisés dans la requête associée au bloc emboîté. Ceci se fait **en concatenant** la nom de la relation avec le nom de l'attribut sous la forme:

```
relation.attribut
```

Exemple:

```
R.NP
```

Utilisation de la restriction de portée

Pour obtenir le numéro des fournisseurs **ne livrant pas** le produit 2, il faut tout d'abord déterminer, pour un fournisseur donné, disons NF0, l'ensemble des produits qu'il livre:

Ceci peut être fait par la requête:

```
(  SELECT   NP
   FROM     R
   WHERE    NF = NF0)
```

Si l'on note cet ensemble *E(NF0)*, il faut ensuite determiner les fournisseurs NF pour lesquels *E(NF)* ne contient pas 2.

```
(  SELECT   NF
   FROM     R
   WHERE    E(NF) NOT CONTAINS (2))
```

Ce qui conduit à la requête emboîtée suivante:

```
(  SELECT   NF
   FROM     R
   WHERE    (  SELECT   NP
               FROM     R
               WHERE    NF = NF)
            NOT CONTAINS (2))
```

Où la condition «NF = NF» n'est pas celle que l'on voudrait (car le 1er «NF» est celui correspondant au SELECT externe alors que le 2e correspond au SELECT emboîté).

Renommage temporaire des relations

La solution est de **renommer la relation** servant de support à la requête emboîtée, en précisant une dénomination temporaire dans le champs `FROM`, sous la forme:

`FROM` «*relation*» «*alias*»

et d'utiliser la possibilité de restriction de portée mentionnée précedemment.

Exemple

La requête correcte pourrait donc s'écrire:

```
(  SELECT   NF
   FROM     R
   WHERE    (  SELECT   NP
               FROM     R R2
               WHERE    R.NF = R2.NF)
            NOT CONTAINS (2))
```

Requêtes multi-relationnelles

Le format générale des requêtes portant **simultanément** sur **plusieurs relations** est:

```
(  SELECT   Attribut1, Attribut2, ...
   FROM     Relation1, Relation2, ...
   WHERE    «Condition de jointure entre les relations»
            AND
            «Conditions de la requête»
```

où les conditions de ***jointure*** entre les relations sont des conditions portant simultanément sur des attribut de plusieurs des relations.

Exemple de requêtes multi-relationnelles

Si l'on considère en plus de la relation **R**, la relation **R2** suivante définissant les couleur et les poids (en grammes) des produits:

R2:

NP	Couleur	Poids
1	bleu	200
2	bleu	100
3	blanc	200

La requête permettant d'obtenir les produits bleus livrés par le fournisseur 1 sera:

```
(   SELECT   NP
    FROM     R,R2
    WHERE    R.NP = R2.NP
             AND
             R2.Couleur = "bleu" AND R.NF = 1)
```

SGBD «Bataille Navale» – TP

Descriptif:

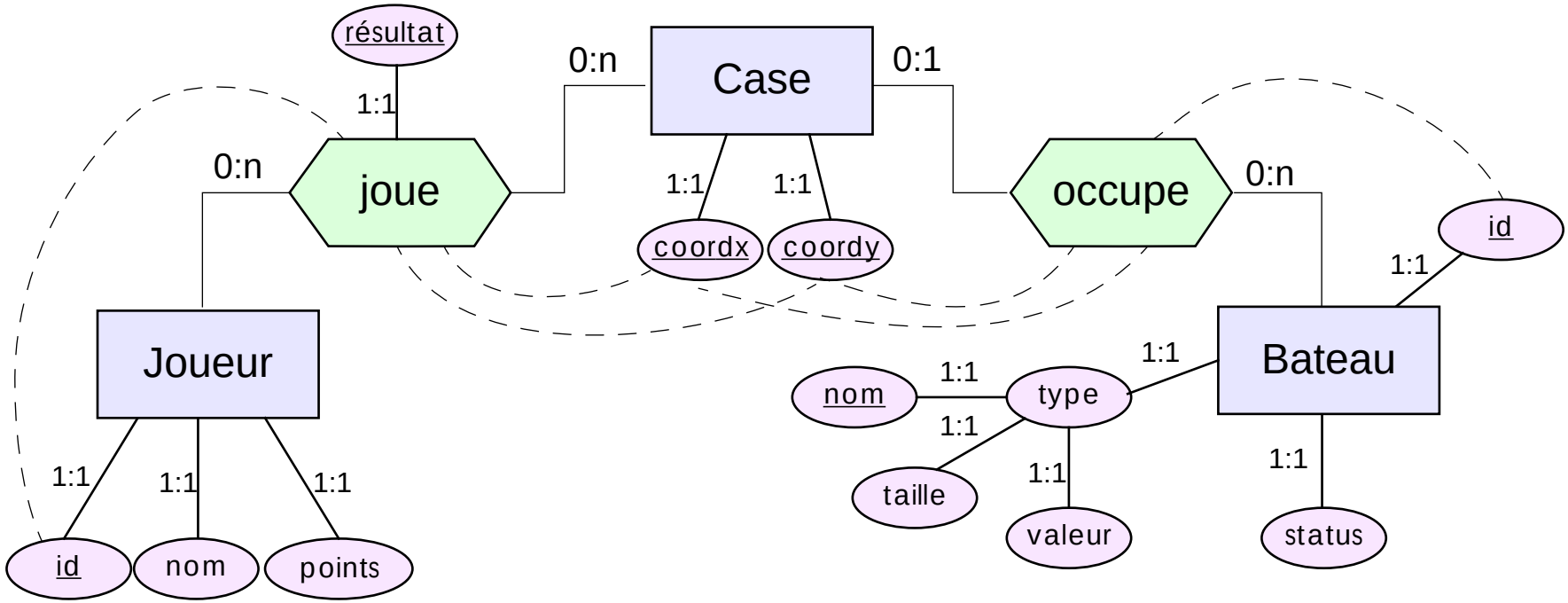
L'objectif est de réaliser les modèles conceptuels et relationnels d'un SGBD permettant de jouer à une version quelque peu modifiée de la bataille navale; le projet à réaliser en fin de semestre sera échafaudé à partir de la base à construire.

Une partie de bataille navale fait intervenir plusieurs **joueurs**, chacun d'eux étant caractérisés par un **identificateur** (unique), et possédant un **nom**. A chaque joueur, on associera en outre un **total de points**, afin de déterminer le vainqueur de la partie.

L'objectif du jeu est de trouver (et couler) des **bateaux**, situés dans un espace rectangulaire (de dimensions quelconques), segmentés en **cases** de même taille, servant à définir les coordonnées des bateaux et des **coups** joués.
Les **bateaux** sont de différents **types**: porte-avion, croiseur, ... Chaque bateau est caractérisé par un **identificateur** unique, un **nom** qui représente le type de bateau, et associé à ce type, une **taille** (en nombre de case) et une **valeur** (qui représente les points attribués à un joueur qui coule le bateau, c'est-à-dire qui touche la dernière case encore intacte de ce bateau).
Tout les **coups** joués par un joueur doivent être mémorisés, avec leur **résultat** (touché si la case jouée est occupée par un bateau, coulé si la case touchée est la dernière encore intacte du bateau, ou dans l'eau si aucun bateau n'occupe la case, ou que cette case à déjà été touchée)

De plus, on doit être capable à tout moment de déterminer la liste des bateaux du jeux, avec leur **status** (état): intacte, touché ou coulé.

Bataille navale – modèle conceptuel



Contraintes d'intégrité:

① Chaque case doit au moins participer à une des relations *joue* ou *occupe*

② `Bateau.status` = coulé ssi le bateau ne participe pas à la relation *occupe*

③ Pour toutes les participations d'une case donnée à la relation *joue*, il ne peut y avoir qu'un seul `joue.résultat` valant touché ou coulé

Schéma relationnel «brut»

A partir du schéma entité-association précédent,
et en appliquant brutalement les règles de traduction,
on obtient un schéma relationnel non nécessairement normalisé (optimal):

Entités:

```
Bateaux(id, status, nom, taille, valeur)
Joueur(id, nom, points)
Case(coordx, coordy)
```

Associations:

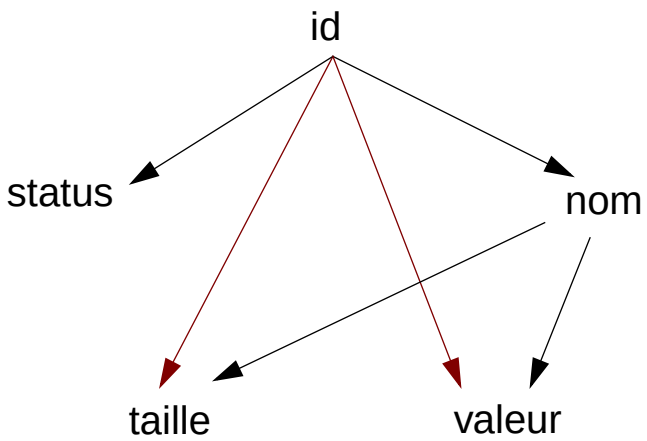
```
Joue(résultat, Joueur.id, Case.coordx, Case.coordy)
Occupe(Bateau.id, Case.coordx, Case.coordy)
```

Il reste maintenant à établir les graphes minimaux de dépendances fonctionnelles,
afin de normaliser notre base.

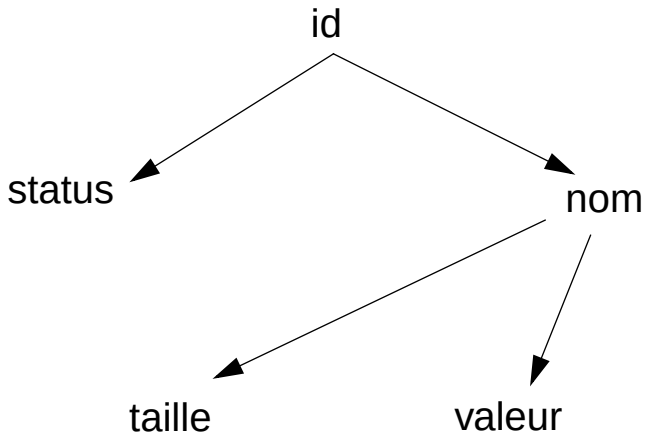
Graphe de dépendances fonctionnelles

Seul la relation «Bateau» est examinée ici:

`Bateaux(id, status, nom, taille, valeur)`



Graphe complet



Graphe minimal

En appliquant la méthode de synthèse, on décompose la relation `Bateau` en:

```
Type_Bateau(nom, taille, valeur)
Bateau(id, Type_Bateau.nom, status)
```

Schéma relationnel normalisé

En appliquant successivement le processus précédent aux autres relations, et en constatant que la relation Case, qui ne se réduit qu'à ses identifiants, et dont tout les tuples sont forcément contenus dans les relations Occupe et Joue, en vertu de la contrainte d'intégrité 1, peut donc être supprimée du schéma, on obtient finalement:

Entités:

```
Type_Bateau(nom, taille, valeur)
Bateau(id, Type_Bateau.nom, status)
Joueur(id, nom, points)
```

Associations:

```
Joue(résultat, Joueur.id, Case.coordx, Case.coordy)
Occupe(Bateau.id, Case.coordx, Case.coordy)
```

Les principaux domaines de l'informatique

... abordés dans le cadre de ce cours:

- La Programmation
- Les Systèmes d'Exploitation
- Les Systèmes d'Information

- La Conception d'Interfaces
- Le Calcul Scientifique
- Les Agents Logiciels

Interfaces Humains-Machine (IHM)

Pourquoi faut-il s'intéresser à la conception
et au développement des ***Interfaces Humains-Machine*** (IHM) ?

Une mauvaise interface peut provoquer le rejet
de la part des utilisateurs, leur frustration, voire leur anxiété,
face au système qu'ils ont à utiliser.

Inversement, une bonne IHM amplifie les sensations
positves de succès et de contrôle.

Mais qu'est-ce qu'une bonne IHM ?

D'une façon générale, une *bonne* IHM
est une interface **que l'utilisateur ne remarque plus** !

Objectifs de la conception des IHM (1)

Concrètement, les ***objectifs généraux*** pour la conception d'une IHM sont:

(US Military Standards for Human Engineering Design Criteria)

- permettre la ***réalisation*** des tâches prévues,
- ***minimiser l'investissement*** nécessaire pour pouvoir utiliser l'interface
- garantir des ***interactions fiables***
- favoriser la ***standardisation***

Objectifs de la conception des IHM (2)

- **Permettre la réalisation des tâches prévues:**

  La **richesse fonctionnelle** doit être **adaptée**.
  Un ensemble insuffisant de fonctions rend un système inutilisable, quelque soit la qualité de sont interface....
  mais une richesse fonctionnelle trop importante
  rend un système difficile à maîtriser.

  ⇨ Une analyse fonctionnelle doit être réalisée pour recenser l'ensemble des tâches et sous-tâches véritablement nécessaires, ainsi que leur fréquence d'utilisation.

- **Minimiser l'investissement de l'utilisateur:**

  Minimiser la **durée d'apprentissage** et le **niveau de compétences** requis, ce qui peut s'obtenir, entre autre, par un mimétisme plus ou moins marqué avec d'autres IHM auxquelles l'utilisateur à déjà été confronté.

Objectifs de la conception des IHM (3)

- **Garantir des interactions fiables:**

  Garantir un bon degré de *fiabilité* lors des interactions.

  La confiance que place l'utilisateur dans le système est souvent fragile! Les interactions offertes par une bonne IHM doivent donc contribuer à **augmenter la confiance de l'utilisateur**: fonctionnement sans erreurs, organisation fonctionnelle claire et cohérente, stabilité dans le temps, ...

- **Favoriser la standardisation:**

  La standardisation permet de **réduire les temps d'apprentissage**, augmente la confiance et les performances des utilisateurs (moins d'erreurs) et **améliore** la **portabilité des systèmes**.

Domaines connexes

Le développement d'IHM est une activité ***multidisciplinaire***.

Il faut des specialistes:

- en ***pychologie*** et ***facteurs humains***
  pour prendre en compte les concepts issus
  des *théories de la perception et de la cognition*;

- en ***conception logicielle***
  pour utiliser au mieux les techniques informatiques disponibles;

- en ***développement de matériel***
  pour mettre à profit les progrès dans le domaine de la conception
  de *nouveau périphériques*, et offrir un accès au système au plus
  grand nombre de personnes (handicapés compris)

- en ***conception graphique***
  pour la fabrication des «layouts» qui seront utilisés dans le cas
  d'interfaces visuelles

- ..., en traitement de la parole, en ergonomie, ...

Principales étapes de la conception

Les principales ***étapes de la conception*** d'une IHM sont:

1) **déterminer l'ensemble des tâches** que l'IHM devra permettre de réaliser:
une bonne IHM est une IHM dont les objectifs fonctionnels sont clairement identifiés;

2) **déterminer les caractéristiques principales des utilisateurs** qui seront amenés à utiliser l'IHM (leur *profil*): la qualité d'une IHM est directement dépendante de sont adéquation avec la population d'utilisateurs pour laquelle elle est prévue;

3) **proposer plusieurs prototypes** d'interface qui seront discutés et évalués par les concepteurs et les **utilisateurs potentiels**: une bonne IHM naît le plus souvent de la diversité... et plusieurs pistes doivent donc être explorées;

4) **produire une spécification explicite** de l'IHM, décrivant à la fois les *contraintes fonctionnelles* et les *contraintes de layout*; un **manuel d'utilisation** et une **référence technique** pourront également être produits durant cette phase;

5) **réaliser l'IHM** proprement dite (phase d'implémentation effective);

6) **évaluer l'IHM** produite sur la base *d'indicateurs reconnus* (voir ci-après).

Critères d'évaluation

Dans le but de ***quantifier*** (mesurer) **la qualité d'une IHM**, quelques ***indicateurs*** sont fréquemment employés:

1. la **durée d'apprentissage**:
   la durée moyenne nécessaire pour qu'un utilisateur typique maîtrise les fonctions pour lesquelles l'IHM a été développée;
2. la **rapidité d'exécution**:
   la durée moyenne de réalisation d'un ensemble test de tâches par un groupe d'utilisateurs de référence;
3. le **taux d'erreur**:
   le nombre **et la nature** des erreurs faites par le groupe d'utilisateurs de référence lors de la réalisation de l'ensemble test de tâches;
4. la **mémorisation dans le temps**:
   c'est-à-dire l'évolution dans le temps des critères précédents (qu'en est-il après une heure d'utilisation, un jour, une semaine, ... ?);
5. la **satisfaction subjective**:
   la satisfaction subjective des utilisateurs pourra être mesurée par le biais de questionnaires ou d'interviews face-à-face.

Il n'est souvent **pas possible d'optimiser l'ensemble des critères** mentionnés ci-dessus et des **compromis** devront être faits selon les domaine d'application des IHM.

Domaines d'application (1)

Les principaux domaines d'application pour
les IHM se répartissent dans trois grandes catégories:

- ⇨ les ***systèmes critiques*** (généralement temps réel)
- ⇨ les ***systèmes commerciaux et industriels***
- ⇨ les ***systèmes personnels et de loisirs***

Domaines d'application (2)

- **Les systèmes critiques:**

(contrôle aérien, pilotage d'avions ou de
centrales nucléaires, appareillage médical, ...)

Dans ce domaine, la ***fiabilité*** (faible taux d'erreur) et la ***performance*** (temps de réponse très court) sont centrales, y compris et en particulier dans des **conditions de stress** pour les utilisateurs.

Elles seront souvent obtenues au prix de **durées d'apprentissage plus longues**, et la mémorisation est garantie par des **entrainements fréquents** (*drills*)

Domaines d'application (3)

- **Les systèmes commerciaux et industriels:**

(applications de type transactionnels, dans le domaine
des banques, assurances, gestion des stocks, comptabilité,
réservations d'avions, de trains ou d'hôtels, de la vente, ...)

Dans ce domaine, le facteur déterminant est le ***coût***.

La **formation des utilisateurs** coûte cher; la durée d'apprentissage
devra donc être réduite... tout en garantissant cependant des **taux d'erreurs acceptables** (car les erreurs représentent également un coût, souvent mesurable).

La **rapidité d'exécution** est également importante,
car elle conditionne le nombre d'opérations (ou de transactions) réalisées.

Ainsi, en 1988, une étude américaine a montré que, pour l'IHM
du service de renseignement téléphonique d'un opérateur (équivalent du 111 en Suisse),
une réduction de 0.8 sec de la durée moyenne des appels (15 sec) représentait
une économie annuelle de l'ordre de 40 millions de dollars!

Domaines d'application (4)

- **Les systèmes personnels et de loisirs:**

(suites bureautiques, jeux, applications éducatives, systèmes d'exploitation, ...)

Dans ce domaine, la ***satisfaction subjective*** **des utilisateurs**
est l'élément central car c'est souvent le critère déterminant pour
le choix d'un système... et la concurrence est féroce.

La **rapidité de l'apprentissage** et de **faibles taux d'erreurs**
seront également déterminants.

Une difficulté supplémentaire dans ce domaine
est **l'extrême variété des utilisateurs** (du néophyte à l'expert confirmé),
qui va souvent nécessiter une **structure à plusieurs niveaux**
pour les interfaces (*multilayered interfaces*).

Conception d'une IHM (1)

La conception d'une IHM peut
être guidée, à des niveaux variés de généralité, par:

1. des ***modèles conceptuels*** **de haut niveau**
2. des ***principes généraux*** **de conception**
3. des ***recommandations pratiques*** **spécifiques**

Conception d'une IHM (2)

Un exemple de modèle conceptuel:

le modèle *syntaxique-sémantique* «***objets-actions***»

☞ L'idée de ce modèle est de postuler
qu'une bonne description d'IHM doit distinguer entre:

⇨ **le *niveau sémantique***,
dont l'objectif est de **décrire les concepts** (objets et actions)
nécessaires à la réalisation des tâches pour lesquelles l'IHM est conçue;

⇨ **le *niveau syntaxique***,
dont l'objectif est de **décrire les séquences de commandes** ou
d'actions spécifiques qui devront être effectuées pour réaliser les tâches,
avec un système donné.

Seul le ***niveau syntaxique*** est donc **dépendant**
des **détails d'implémentation liés à un système particulier**.

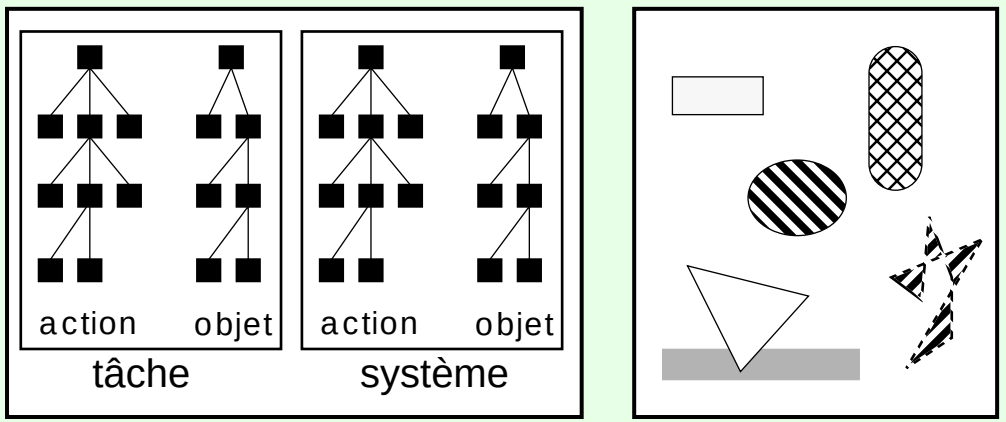
Conception d'une IHM (3)

Le ***niveau sémantique*** est également décomposable en deux parties:

- ⇨ la **description des *concepts liés aux tâches***;
- ⇨ la **description des *concepts liés à la réalisation*** de ces tâches par le biais d'un **système informatique**.

Il est important de bien noter la différence entre ces **deux aspects**, qui peuvent être **très différemment maîtrisés par les utilisateurs** un utilisateur peut en effet être un expert de la tâche et ne rien connaître à l'informatique ou inversement.

Modèle conceptuel



Exemple de description de tâche (1)

Tâche: ***formatter un document***

1. ***Description conceptuelle*** de la tâche dans le modèle sématique (ici sa **décomposition en sous-tâches**):

   Formatter un document:

   1. centrer le titre
   2. mettre le titre en gras
   3. justifier le corps du document

2. **Description des objets et actions identifiés**, sous la forme de **concepts informatiques**:

   document = fichier manipulé par le traitement de texte utilisé.

   Dans le traitement de texte utilisé:

   - un *document* est décomposé en *paragraphes*, pouvant chacun être associé à des actions de formattage spécifiques;
   - un paragraphe est constitué de *caractères*, pouvant chacun être également associé à des actions de formattage spécifiques;

Exemple de description de tâche (2)

De (1) et (2), on peut obtenir la ***traduction informatique*** de la *description conceptuelle*:

1. appliquer l'action de formattage «*centrer*» au paragraphe contenant le titre;
2. appliquer l'action de formattage «*mettre en gras*» à l'ensemble des caractères du titre;
3. appliquer l'action de formattage «*justifier*» à l'ensemble des paragraphes du corps du document;

2. ***Description syntaxique*** **des actions identifiés**, dans le cadre d'un système et d'une IHM (typiquement graphique) donné:

   - Pour **appliquer une action de formattage**, il faut **sélectionner** les objects concernés, puis effectuer l'action.

   - Pour **selectionner les objets** (paragraphes ou caractères), il faut pointer le début de la zone d'écran concernéee à l'aide du curseur de la souris, appuyer et maintenir le bouton gauche [de la souris] tout en déplaçant le curseur vers la fin de la zone, et relâcher le bouton de la souris.

   - Pour **effectuer une action sur une zone**, il faut la sélectionner au sein des menus hiérarchiques (contextuels) apparaîssant après un *clic*-droit de la souris.

Caractéristiques du niveau syntaxique

Le ***niveau syntaxique*** est celui qui
pose le plus de problèmes aux utilisateurs car:

⇨ il est **fortement dépendant du système** utilisé,
il faut donc recommencer par une phase d'apprentissage
chaque fois que l'on change de système;

⇨ il est **difficile à décrire de façon structurée ou modulaire**,
et il est donc difficile pour l'utilisateur de s'en faire un modèle mental;

⇨ il **s'appuie** en outre souvent **sur des conventions arbitaires**,
qui doivent être mémorisées telles quelles.

Principes généraux (1)

Quelques **principes généraux** de conception d'IHM

1. Connaître les utilisateurs

Une interface ne peut être satisfaisante pour tous les types d'utilisateurs.

On distingue habituellement entre:

- les ***utilisateurs novices***:
  - pas de connaissances syntaxiques,
  - peu de connaissances sur les concepts sémantiques et informatiques

    La complexité (**richesse fonctionnelle**) de l'IHM doit être **limitée** à un sous-ensemble de tâches simples et fréquentes; l'aide en ligne doit être fortement développée et adaptée; les retours du système (confirmation d'action, messages d'erreur) doivent être fréquent et compréhensibles.

- les ***utilisateurs-experts débutants***:
  - pas de connaissances syntaxiques,
  - bonne connaissance de la sémantique des tâches,
  - peu de connaissances sur les concepts informatiques

    La richesse fonctionnelle peut être augmentée mais par le biais d'une approche syntaxique fortement structurée (par exemple des menus hiérarchiques)

Principes généraux (2)

- les ***utilisateurs-experts intermittents***:
  - pas de connaissances syntaxiques,
  - bonnes connaissances sur les concepts sémantiques et informatiques

    Le problème est ici de pallier une mémorisation incomplète des éléments de syntaxe. **La richesse fonctionnelle peut être maximale**, mais une aide en ligne exhaustive, munie d'un système de recherche efficace, une aide contextuelle développée ainsi que des techniques de complétion automatique de commandes peuvent être utiles.

- les ***utilisateurs-experts intensifs*** (*power user*):
  - bonnes connaissances syntaxiques,
  - bonnes connaissances sur les concepts sémantiques et informatiques

    Pour ces utilisateurs, l'élément primordial est la **rapidité de l'interaction**. Des courts-circuits (commandes abrégées) et des macros doivent être fournis, et les retours du système réduits au strict nécessaire.

Il est important de bien définir pour quelle classe d'utilisateurs une IHM est prévue. Si l'on veut une IHM utilisable par plusieurs classes d'utilisateurs, il faudra prévoir une **IHM à plusieurs niveaux**.

Principes généraux (3)

2. Définir les tâches

Il est non seulement important de bien **définir les tâches** pour lesquelles une IHM est conçue, mais aussi de **choisir correctement le niveau de granularité** de l'ensemble des fonctions qui seront accessible par le biais de l'IHM pour réaliser les tâches choisies.

Si le niveau de granularité est trop fin,
l'utilisateur devra enchaîner un nombre trop important
de fonctions pour réaliser une tâche. Inversement, si le niveau de granularité
est trop grossier, l'interface devra proposer un nombre trop important
de fonctions, ou l'utilisateur n'aura pas la possibilité de réaliser
ses objectifs dans toutes leurs variétés.

Par ailleurs, la **fréquence moyenne d'utilisation** d'une fonction doit être **prise en compte** lors de son intégration dans l'IHM:

les fonctions fréquemment utilisées devront être
également les plus faciles à mettre en œuvre, quitte à rendre
des fonctions moins fréquentes un peu plus complexes.

Principes généraux (4)

3. Choisir le type d'interaction

Plusieurs types d'interaction sont envisageables pour la conception d'IHM. Parmi les plus courants, on peut citer:

- ⇨ **interaction à base de *menus*** (éventuellement hiérarchiques) ;
- ⇨ **interaction à base de *formulaires*** ;
- ⇨ **interaction à l'aide d'un *langage de commandes*** (éventuellement à l'aide du langage naturel);
- ⇨ **manipulation directe** (avions, voitures, réalité virtuelle, ...)

Les techniques à base de menus ou de formulaires permettent une bonne structuration de l'interaction et fournissent un bon support pour les utilisateurs novices ou débutants. Ils mènent cependant souvent à des IHM relativement **lentes**, et sont peu adaptées à des tâches complexes et des utilisateurs expérimentés.

Les techniques à base de langage de commandes sont plus complexes à mettre en œuvre (à la fois pour l'implémentation et pour l'utilisation) mais permettent une grande richesse fonctionnelle, et sont de ce fait bien adaptées aux utilisateurs-experts confirmés. Les durées d'apprentissage sont cependant plus élevées et leur mémorisation est plus délicate.

Règles d'or (1)

Les huit ***règles d'or*** de la conception d'IHM:

1. **Assurer la cohérence**

   C'est la règle la plus souvent violée... alors qu'elle est normalement facile à respecter. Il s'agit par exemple d'assurer la **cohérence *lexicale/terminologique*** (i.e. les mêmes concepts sont systématiquement identifiés par les mêmes termes), et la **cohérence *structurelle*** (des actions composites similaires doivent pouvoir être réalisées par des séquences d'actions élémentaires similaires)

2. **Raccourcis dans l'interaction**

   Proposer des courts-circuits (commandes abrégés, macros, ...)
   pour les utilisateurs confirmés.

3. **Retours système pertinents**

   Des retours système (*system feedback*) sont en particulier nécessaires pour confirmer le résultat de la réalisation des actions (ces retours doivent en règle générale être **modestes pour des actions fréquentes** et **plus consistant pour des actions complexes** ou rares). Des retours système (par exemple par le biais de changements dans la visualisation) sont également nécessaires pour signaler les modifications intervenues dans le système.

Règles d'or (2)

4. Identifier clairement la terminaison des actions

Ceci est particulièrement nécessaire dans le cas d'actions complexes, correspondant à un enchaînement de plusieurs actions atomiques (indivisibles), ou dans le cas d'actions critiques (sauvegardes, logout, ...).

5. Offrir des mécanismes ergonomiques de correction des erreurs

Permettant en particulier à l'utilisateur de ne pas avoir à resaisir l'ensemble des paramètres d'une commande si l'un d'entre eux est erroné.

6. Garantir la réversibilité des actions

Ce qui permet de réduire le *stress* de l'utilisateur lors de son interaction avec le système.

Règles d'or (3)

7. Augmenter le contrôle de la part de l'utilisateur

Ceci signifie en particulier d'éviter toute action «surprenante» de la part du système, la règle étant de préserver autant de possible la **causalité des actions**.

8. Réduire la surcharge de la mémoire à court terme des utilisateurs

La règle heuristique est que les utilisateurs stockent environ 7 éléments d'information dans leur mémoire à court terme. Des mécanismes doivent donc être disponibles pour réduire toute surcharge inutile: existence de pagination contrôlée, historique des commandes, ...

Exemple de recommandations pratiques (1)

Pour la présentation des données (*display*)

- ⇨ cohérence pour les labels et les conventions graphiques
- ⇨ standardisation des abréviations
- ⇨ cohérence du *layout* (structure des écrans produits)
- ⇨ identification claire des écrans produits, permettant d'en déterminer la fonction, et de les référencer aisément
- ⇨ ne présenter que les données pertinentes pour la tâche en cours
- ⇨ utiliser des formats de représentation intuitif (par exemple graphiques) réduisant le besoin d'interprétation des données visualisées
- ⇨ ne présenter des valeurs numériques que lorsque leur connaissance est effectivement nécessaire à l'utilisateur
- ⇨ commencer par une conception monochrome pour le *display*, et n'ajouter les couleurs que progressivement, si elles apportent une aide effective à l'utilisateur
- ⇨ ...
- ⇨ intégrer les utilisateurs dans le processus de conception des *displays* !

Exemple de recommandations pratiques (2)

Pour **attirer l'attention** de l'utilisateur:

- ⇨ n'utilisez que 2 niveaux d'intensité
- ⇨ n'utiliser que 4 niveaux de taille
- ⇨ n'utiliser que 3 familles de caractères (fontes)
- ⇨ utilisez la vidéo inverse et le clignotement
- ⇨ n'utilisez que 4 couleurs de base, en réservant les autres couleurs pour des usages occasionnels
- ⇨ en cas de clignotement d'éléments colorés, changez la couleur lors du clignotement
- ⇨ ...
- ⇨ pour l'utilisation des sons, préférez les sons agréables pour les *feedbacks* positifs, et les sons désagréables pour les alarmes.