Principes des systèmes informatiques

Olivier Ridoux

Principes des systèmes informatiques

Olivier Ridoux – 2020

Introduction

Nous sommes utilisateurs de nombreux systèmes informatiques pour lesquels nous n'avons pas d'autres explications que celles données par le fabricant ou par le vendeur. Nous sommes emportés dans un flux **d'innovations** qui se substituent les unes aux autres. Nous sommes abreuvés d'un discours qui énonce que c'est la nouvelle marche du monde, que les objets et métiers de demain sont inconnus aujourd'hui ; on appelle parfois cela **l'économie numérique**. Un autre discours vient parachever l'édifice en insistant sur la **virtualisation** et la **dématérialisation** qui seraient intrinsèques à ces systèmes. Quand le tout est désigné par des mots comme le **code** ou le **numérique**, on voit que c'est **l'ésotérisme** et la **pensée magique** qui domine notre compréhension de ces systèmes.

Nous proposons une autre lecture de ces systèmes qui oblige à quitter l'habit **d'utilisateur** et à prendre celui de **scientifique**. On voit alors que le fonctionnement des systèmes informatiques obéit à de **grands principes** qui sont extrêmement **stables** et valent donc la peine d'être expliqués (⑥ au verso). On voit aussi que la virtualisation prend un sens précis qui n'est plus mystérieux, mais explique beaucoup de choses. On voit aussi que la dématérialisation n'est qu'une autre **matérialisation**. Mais pour voir tout cela il faut accepter de changer d'habits.

Il est vrai que les systèmes informatiques sont extrêmement complexes quand on les considère comme un assemblage de leurs composants les plus élémentaires, ex. le programme d'un système informatique peut comporter plusieurs milliards de lignes. Nous proposons donc de comprendre ces systèmes par leur désassemblage en sous-systèmes fonctionnellement plus simples dans une approche qu'on qualifie parfois de ***top-down***.

UNIVERSITÉ DE RENNES 1 · IRISA · istic

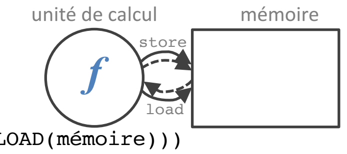
L'entité informatique et ses relations

Un système informatique est bien sûr un **système artificiel** (③ au verso). En isoler des sous-systèmes est aussi une **vue de l'esprit**, tellement les interactions entre sous-systèmes sont denses. C'est pourtant ce qu'on fait tout le temps.

L'entité informatique de base **calcule** et **stocke** des **symboles**, et les **communique** à d'autres entités avec qui elle est en relation. Dans ce modèle, l'entité informatique est composée d'une **mémoire** et d'une **unité de calcul (UC)** qui exécute cycliquement le programme

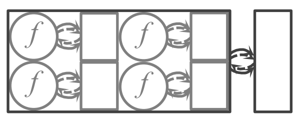
`STORE(mémoire, f(LOAD(mémoire)))`

où f est la **fonction caractéristique** de l'UC, ex. la spécification d'un processeur.

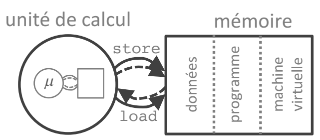


On préférera penser en termes de **symboles** plutôt que de bits ou de nombres. Un symbole est un élément distinguable des autres éléments d'un ensemble fini. Des chiffres sont évidemment des symboles, mais pourquoi se limiter aux chiffres binaires ? Des lettres sont aussi des symboles, mais pourquoi se limiter à l'alphabet romain ? Et pourquoi pas les symboles $, €, ¥ et £ ? Ou ❀, ❀, ❀ et ❀ ? On appelle **vocabulaire**, souvent noté **V**, l'ensemble des symboles utilisés.

Ce modèle peut être décliné de nombreuses façons, ex. en multipliant les UC pour former des **processeurs multi-cœurs**. Cependant, sa déclinaison la plus importante est de mettre en œuvre le **principe de Von Neumann** (1945), lui-même inspiré de la **machine de Turing** (1936), selon lequel la fonction f de l'UC est d'interpréter des symboles stockés en mémoire qu'on appelle des **instructions**. C'est le principe du **programme stocké en mémoire**.



Avant cette idée, on déterminait la fonction réalisée par un calculateur en changeant le câblage de son UC ; depuis on la détermine en déposant un programme en mémoire. Ce principe se décline à son tour en multipliant les **couches d'interprétation**, que ce soit en mémoire ou dans l'UC. C'est la base de la **virtualisation**, qui permet à une entité informatique d'en simuler d'autres, comme par exemple une **machine Java**.



Le système de gestion de fichiers (SGF) sur disque (HD)

Le principe **modèle-vue-contrôle** est une façon raisonnée de décrire un système :

- Le **modèle** décrit le schéma d'organisation d'un système. Ex. les habitants d'un SGF sont des **fichiers** et des **répertoires**, auxquels l'utilisateur donne des **noms arbitraires**. Les fichiers contiennent des **données** de **taille quelconque**. Les répertoires sont des **dictionnaires** qui associent à des habitants du SGF leur nom arbitraire. On dit qu'un répertoire **contient** un habitant du SGF si il connaît son nom.
- La **vue** décrit ce que le système dévoile de son contenu. Pour un SGF, deux styles de vue sont utilisées : un style **graphique** qui montre les répertoires sous la forme d'un arbre (ce qu'ils ne sont pas en général), ou un style **commande** qui montre le contenu d'un répertoire particulier, le **répertoire de travail**, via des commandes comme `ls`, et `cd` pour changer de répertoire de travail.
- Le contrôle décrit la façon d'interagir avec le système. Dans le style commande, le contrôle du SGF est exprimé par des commandes comme `mkdir`, `touch` et `cp` pour créer des habitants, `rmdir` et `rm` pour les supprimer, et `mv` pour changer le nom d'un habitant ou le changer de répertoire.

Comparé à un nommage **plat** (où un répertoire contient tous les fichiers) le **nommage hiérarchique** (où des répertoires en contiennent d'autres) permet des accès beaucoup plus rapides pour peu de stockage en plus. Ex. soit des répertoires de taille **T** (ex. $T=10$), contenant chacun **T** répertoires, le tout sur **n** niveaux (ex. $n=6$), le dernier niveau ne contenant que des fichiers. Il y aura donc $N=T^n$ fichiers ($=10^6$). Le temps d'accès moyen sera de l'ordre de $T\times\log_T N$ ($=60$) et le volume total des répertoires de $N+N/T$ ($=1{,}1\times10^6$), alors qu'avec le nommage plat le temps d'accès et le volume seraient de **N** ($=10^6$). Ce principe est utilisé partout, du SGF au WWW.

Une problématique d'un SGF est de réaliser ces fonctions sur un disque dur magnétique (④ au verso) où (a) les noms sont **conventionnels** (piste×secteur) et pas **arbitraires**, (b) les tailles de fichiers sont **quelconques** et indépendantes de celle, **fixe**, des secteurs, et (c) l'accès à un secteur est environ **10^6 plus lent** que l'accès à une mémoire. Les réponses à ces questions sont de (a) traiter les répertoires comme des fichiers dont le dictionnaire est le contenu, (b) répartir les contenus des fichiers et des répertoires sur plusieurs secteurs, et (c) utiliser une partie de la mémoire pour y représenter la partie du HD la plus utile à un instant donné ; c'est le **cache**.

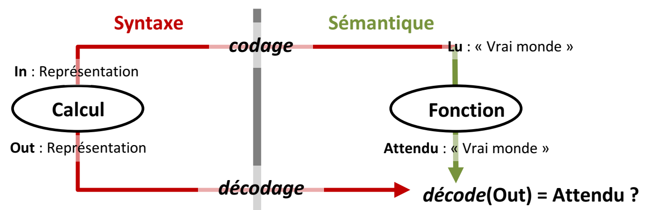
L'usage d'un cache repose sur le **principe de localité**. Ce principe permet de prévoir le futur proche en observant le passé récent selon l'hypothèse que **ce qui s'est passé récemment a une grande probabilité de se reproduire prochainement**. Pour le SGF, cela permet d'estimer la partie utile du HD à un instant donné, de la stocker dans le cache et d'accéder au cache plutôt qu'au disque, c-à-d. 10^6 fois plus vite.

Le bit de mémoire

Le **bit de mémoire** (**b**) représente la capacité de se souvenir d'un choix parmi deux options. Deux bits permettent de se souvenir d'un choix parmi quatre, et plus généralement, **n** bits donnent la capacité de se souvenir d'un choix parmi 2^n. Inversement, pour se souvenir d'un choix parmi **p**, il faudra $\lceil\log_2 p\rceil$ bits (① au verso).

Un **octet** (**o**) est un groupe de 8 bits. Bit et octet forment la base d'une famille d'unités construites comme celles du **système international** (**SI**) : ex. $b\times s^{-1}$ (vitesse de transfert), $€\times o^{-1}\times s^{-1}$ (coût du stockage entretenu) ou **Go** (giga-octet). Les préfixes **kilo**, **méga**, **giga**, etc., qui valent des puissances de 10^3 dans le système SI, sont parfois utilisés comme des puissances de 2^{10} pour mesurer les systèmes informatiques. Si pour **kilo** (10^3 ou 2^{10}) l'erreur n'est que de 2,4 %, pour **téra** (10^{12} ou 2^{40}) elle est de presque 10 %. On peut donc préférer les préfixes de la **Commission électronique internationale** : **kibi**, **mébi**, **gibi**, etc., qui sont bien des puissances de 2^{10}.

Les bits sont utilisés pour **représenter** des données sous des codifications conventionnelles. De telles codifications concernent les **caractères**, ex. **ASCII** ou **UTF**, les **nombres**, ex. les codages **binaires**, en **complément à deux** ou en **virgule flottante**, mais aussi les images, la vidéo, le son, etc. Dans tous les cas, il est important de bien comprendre la différence entre le « vrai monde » et sa représentation. **Les systèmes informatiques ne voient que des représentations** et il faut toujours s'assurer que ce qui résulte des calculs sur les représentations a du sens dans le vrai monde (⓪ au verso). Des catastrophe extrêmement graves ou coûteuses se sont produites faute d'avoir retenu ce grand principe.



Une représentation est souvent imparfaite par construction : arrondi, domaine borné, discrétisation, compromis précision-coût. Attention aux abus de représentations, ex. utiliser **virgule flottante** parce qu'il y a des virgules ou **complément à deux** parce qu'il y a un signe, alors que des représentations plus sobres seraient plus adaptées, ex. en changeant d'unité pour éviter les virgules (ex. **€ ⟶ millième**) ou en changeant de zéro pour éviter les signes (ex. **°C ⟶ °K**).

Le bit d'information

Intuitivement, on a le sentiment que deux messages de même taille peuvent ne pas être aussi informatif l'un que l'autre. Ex. pour 32 caractères tous les deux **Il a gelé en janvier à Rennes** est une banalité alors que **Il a gelé en juillet à Rennes** est un événement. On a aussi le sentiment que le même contenu informatif peut être exprimé de façon plus ou moins compacte, ex. **Autrefois les trains proposaient une première classe, une seconde classe et une troisième classe** ou **Autrefois les trains proposaient trois classes**. On constate aussi que parfois on fait exprès de ne pas choisir la forme la plus compacte : ex. chez le notaire, **L'appartement comporte 3 (trois) pièces**. On introduit des **redondances** exprès.

La **théorie de l'information** de Claude Shannon formalise ces observations, permet de les mesurer, et leur donne un sens. Dans cette théorie, un **bit d'information** est ce qui **divise par deux l'incertitude sur le message reçu**. Ex. si les symboles **A** et **B** sont équiprobables, chacun des symboles du message **ABA** divise par deux l'ensemble des huit messages possibles, mais en français chaque symbole de l'alphabet divise l'incertitude par une quantité qui lui est propre. Par exemple, dans le message **Que sais-je ?** on est presque certain de trouver le **u** après le **Q**. On note $h(s)=-\log_2 p(s)$ l'information portée par un symbole **s** de probabilité **p(s)**. **L'entropie H** d'un **vocabulaire V** est la moyenne pondérée des informations portées par ses symboles, $H(V)=\sum_{s\in V} p(s)\times h(s)$.

Même si il faut distinguer quantité de mémoire et quantité d'information, les deux sont liées. Shannon montre que la taille d'un message (mesurée en bit de mémoire) est toujours supérieure à celle de son contenu informatif (mesuré en bit d'information), mais qu'il est possible de **compresser** le message pour lui donner une taille très proche de celle de son contenu informatif. Le taux de compression est limité par l'entropie.

Inversement, introduire des **redondances** augmente la taille d'un message mais permet de retrouver son contenu initial, même quand le message a été altéré. Shannon montre dans quelle mesure il est possible d'introduire des redondances qui permettent de **détecter** les altérations, de les **localiser** et de les **corriger**.

Les systèmes informatiques sont composés d'éléments très nombreux à toutes les échelles, ex. plusieurs milliards d'abonnés internet, mais aussi plusieurs milliards de transistors dans un circuit intégré, ou plusieurs milliers de milliards de cellules magnétiques dans un disque dur. Ces deux résultats de Shannon sont alors largement utilisés pour permettre de résister aux pannes inévitables de l'un ou l'autre élément en **introduisant des redondances**, et pour optimiser leur fonctionnement en **compressant les données**.

Le port USB

Le port **USB** (***Universal Serial Bus***) permet à des entités informatiques de communiquer dans une relation qu'on appelle **maître-esclave**. Une des entités joue le rôle de **maître**, ex. un PC, et toutes les autres jouent celui **d'esclave**, ex. les périphériques USB. Le propre de cette organisation est que le maître a toujours l'initiative. Le **protocole** USB de base est une séquence de trois **trames** :

trame 1 = **maître ⟶ esclave : un ordre**
trame 2 = **maître ⟶ esclave : complément de l'ordre**
ou = **esclave ⟶ maître : réponse de l'esclave**
trame 3 = **esclave ⟶ maître : bien/mal reçu, recommencez**
ou = **maître ⟶ esclave : bien/mal reçu, recommencez**

La réponse **bien/mal reçu** est possible parce que des **redondances** détectent les anomalies. Le maître adresse ses ordres tour à tour à chacun des esclaves à une fréquence très élevée, ce qui donne l'impression que tous échangent avec le maître en même temps.

Le port USB permet au maître de fournir de l'électricité à ses esclaves, mais pas plus de 5 V et 200 mA (= 1 W) par port. On distingue des objets auto-alimentés, ex. une imprimante, et d'autres qui comptent sur l'alimentation USB.

Il existe de nombreux objets USB avec des comportements très différents qui sont des variations/combinaisons de quatre grands modes de fonctionnement :

1. **commande** : qui implémente le protocole de base.
2. **IHM** (**interface homme-machine**) : **débit faible** et **mode interruptif**, comme pour une souris, un clavier ou un *joystick*. Le maître **scrute** le périphérique à une fréquence suffisante pour donner l'impression que le périphérique commande.
3. **échange de données** : débit au mieux (***best effort***), **garantie de préservation** des données, comme pour une clé USB.
4. **échange isochrone** : **garantie de débit stable et normé**, préservation des données au mieux, comme pour un baladeur USB en mode lecture.

Un même objet peut fonctionner sous plusieurs modes, ex. un baladeur fait aussi de l'échange de données quand on y dépose un fichier.

Chaque objet doit donc **négocier** avec le maître les ressources dont il a besoin pour fonctionner, et le maître peut refuser la connexion si la demande est excessive. Mais l'esclave doit utiliser le port USB pour négocier le droit d'utiliser le port USB ! Pour résoudre ce problème d'œuf et de poule, tout objet USB doit implémenter le mode commande avec une consommation électrique marginale au moins pendant le temps de la négociation. C'est un exemple du principe du **bootstrap** qui permet à un système de mettre en place sa phase stable.

Vers de très grands systèmes

Le système de fichiers et le port USB sont de tout petits systèmes informatiques composés de très peu d'éléments. Qu'en est-il de systèmes plus grands ?

Les grands systèmes utilisent les mêmes principes que les petits, plus de nouveaux principes liés à leur taille. Un de ces nouveaux principes est que lorsqu'il y a de nombreux composants il y en a toujours qui ne marchent pas ; ce qui était **improbable individuellement devient certain collectivement**. On doit affronter le problème de **concevoir un système fiable avec des composants qui ne le sont pas**.

Parmi les plus petits des grands systèmes, on peut s'intéresser aux systèmes **RAID** (***redundant array of inexpensive disks***). L'idée est d'assembler plusieurs **disques durs** de façon à faire **émerger** des propriétés globales de capacité, de vitesse et de robustesse, qu'aucun n'a individuellement. On y retrouve à l'œuvre des principes généraux comme celui de la **redondance**, mais ici au lieu d'avoir des bits ou des mots de parité, on a des **secteurs** ou **des disques de parité**.

Plus grand, on trouve les datacentres (***datacenter*** en anglais). Ce sont des usines (parfois plutôt un atelier) qui rassemblent de nombreuses entités informatiques, parfois des dizaines de milliers, pour offrir des services sécurisés de **calcul**, de **stockage/archivage** ou de **communication**. Ce sont donc aussi des **entités informatiques**. Ex. l'archivage utilisera des bandes magnétiques (de très grande capacité, mais très lentes). Les données migreront des bandes à des disques durs, à des disques SSD, puis à de la mémoire pour être utilisables par les calculs. Elles emprunteront progressivement le chemin inverse quand elles ne serviront plus. Le tout sera gouverné par le **principe de localité**.

Les moteurs de recherches utilisent un type particulier de data-centre. Répondre à la question **Trouver des pages web qui contiennent les mots X et Y** demande de parcourir tout le web à la recherche des pages désirées. On ne peut pas faire cette recherche en réaction à chaque question, car cela prendrait trop de temps. En plus, **tout serait à refaire pour répondre à une autre question**. Au lieu de cela, les moteurs de recherches explorent systématiquement le web en mettant à jour un dictionnaire **mot ⟶ pages** sans attendre qu'une question soit posée. Le moteur de recherche enregistre ainsi les réponses à toutes les questions simples qui pourraient être posées. La réponse à une question précise s'obtiendra en combinant les réponses du dictionnaire sans avoir à explorer le web. Cette façon de réaliser un calcul avant qu'il ne soit demandé s'appelle le **calcul spéculatif**. C'est aussi un grand principe, très largement utilisé, complémentaire du **principe de localité**.

Conclusions

Même si les systèmes décrits ici peuvent devenir obsolètes, les principes qui les gouvernent sont destinés à durer. Ces principes gouvernent les systèmes informatiques depuis très longtemps à des échelles et dans des contextes très divers. Ils sont des **grands principes** qui aident à **mieux comprendre** les systèmes informatiques et à en **parler scientifiquement**.

La sélection des principes évoqués ici concerne le **calcul**, avec Turing (1936) et Von Neumann (1945), la **mémorisation** et la **communication** des symboles, avec Shannon (1948). Certains concernent plusieurs facettes, comme le **principe de localité** (Denning, 1972) ou le **calcul spéculatif**, et plus fondamentalement le concept **d'entité informatique** qui justement possède ces trois facettes. Toutes pourraient être approfondies séparément, et le sont dans des enseignements spécialisés.

D'autres principes, pourtant bien connus dans d'autres sciences, ne gouvernent toujours pas la conception des systèmes informatiques, ex. les principes de la **thermodynamique** ou de la **conservation de l'énergie**. Pourtant, la prise en compte de la **durabilité** va certainement en faire de futurs grands principes.

Bibliographie

- « ***Great Principles of Computing*** », Peter Denning (conférence Purdue University, 2012, et livre MIT Press, 2015). Pour la défense du principe d'en revenir aux grands principes.
- « ***On Computable Numbers with an Application to the Entscheidungsproblem*** », Alan Turing (1936), à lire dans « ***The Annotated Turing*** », Charles Petzold (Wiley, 2008), et « ***First Draft of a Report on the EDVAC*** », John Von Neumann (1945). Pour des principes des systèmes informatiques qui ont plus de 80 et 70 ans.
- « ***The new Turing Omnibus*** », Alexander Dewdney (Owl Books, 2001). Tous les informaticiens devraient l'avoir lu.
- « ***Computer Science, an overview*** », J. Gleen Brookshear (Addison Wesley, 2003) et « ***Computer Science Illuminated*** », Nell Dale et John Lewis (Jones & Bartlett, 2004), multi-réédités. Deux ouvrages d'introduction à la science informatique : très bien conçus et très utilisés dans les 1er cycles universitaires nord-américains.
- « **La physique par les objets quotidiens** », Cédric Ray et Jean-Claude Poizat (Belin, 2007). Une autre façon de changer d'habits ; quelques systèmes informatiques y sont traités, mais comme des objets physiques éclairés par des principes physiques.
- « ***Code: The Hidden Language of Computer Hardware and Software*** », Charles Petzold (Microsoft Press, 2000). Une autre introduction aux systèmes informatiques, mais délibérément *bottom-up*.

Mathématiques et éléments de contexte : mathématiques et systèmes informatiques — lire, apprendre, comprendre.
Pliage : recto visible (autre face), traits gris rentrants et traits **rouges** saillants. Découper selon le trait rouge entre les deux ●, puis achever le pliage.

Prérequis

Tous les modules qui précèdent celui-ci (notamment CODAGE et INF1), ainsi que leurs prérequis, sont des prérequis de celui-ci.

① Puissances, logarithmes et exponentielles

Soit un entier n positif, la $n^{ème}$ puissance d'un nombre réel positif x, notée x^n, est le produit de n copies de x. On appelle n l'exposant de la puissance. Cette opération est inversible, et son inverse est la racine $n^{ème}$ d'un nombre x, notée $\sqrt[n]{x}$. On montre que $x^{a+b} = x^a \times x^b$ et $x^{a\times b} = (x^a)^b = (x^b)^a$, et on convient par extension que $x^{-n} = 1/x^n$ et $x^{1/n} = \sqrt[n]{x}$, et que $x^{a/b} = \sqrt[b]{(x^a)} = (\sqrt[b]{x})^a$. Noter que $(x^a)^b$ n'est généralement pas égal à $x^{(a^b)}$.

Une fonction logarithme f est définie par la propriété que $f(x\times y) = f(x) + f(y)$. On en déduit que $f(1) = 0$. Si on impose que f est dérivable, alors on doit avoir $f'(x) = k/x$. La fonction logarithme naturel (ou népérien), ln, est la solution de ce système quand $k = 1$. On montre que ln est définie de 0 à $+\infty$, et que son image croît strictement de $-\infty$ à $+\infty$, que $\ln x^a = a \times \ln x$ et que $\ln (x/y) = (\ln x) - (\ln y)$. Noter que pour x suffisamment grand on a $\ln x \ll x$, et que pour x voisin de 1 on a $\ln x \approx x - 1$.

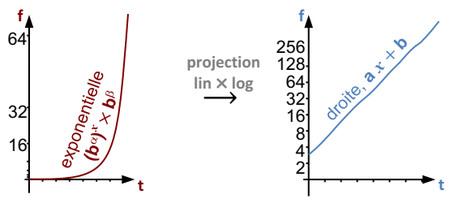
On observe que ln est inversible, et on appelle exponentielle, e(x), son inverse. On montre que $e'(x) = e(x)$. On appelle nombre d'Euler la valeur $e(1) = 2{,}718\ldots$, et on la note e. On convient que $e^x = e(x)$. On montre que $e^{x+y} = e^x \times e^y$ et $e^{x\times y} = (e^x)^y = (e^y)^x$, et que $e^{-x} = 1/e^x$. On convient que $a^x = e^{x\times \ln a}$, ce qui généralise l'opération puissance à des exposants quelconques. Noter que $(a^x)^y$ est en général différent de $a^{(x^y)}$.

Dans le cas général, on note $b = e(1/k)$, et on appelle la solution du système logarithme en base b, noté $\log_b$. On montre que $\log_b b^x = b^{\log_b x} = x$, que $(\log_b b') \times (\log_{b'} b) = 1$ et $\log_b x = (\log_{b'} x) \,/\, (\log_{b'} b)$. On voit que $\ln x = \log_e x$ et $\log_b x = \ln x \,/\, \ln b$.

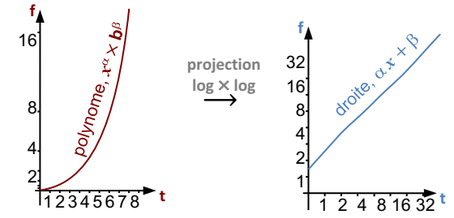
On montre enfin que pour tout $\alpha > 0$, $b > 1$, on a $\lim_{x\to\infty} x^\alpha/\log_b x = \infty$ et $\lim_{x\to\infty} b^x/x^\alpha = \infty$; une puissance, aussi faible soit-elle, finit toujours par dépasser un logarithme, et une exponentielle finit toujours par dépasser une puissance. Ce sont 3 ordres de grandeurs très différents.

Le logarithme permet des changements de variable intéressants. Quand la mesure d'un phénomène croît d'un facteur constant à chaque pas de temps, on dit que sa croissance est exponentielle, car $f(t) = k^{t-t_0}\times f(t_0)$. Ces phénomènes sont difficiles à représenter par une courbe car le rapport entre les plus petites valeurs et les plus grandes est énorme : ex. **environ 8 milliards sur l'histoire des capacités des disques durs depuis leur invention en 1956**, alors qu'une feuille ou un écran ne font que **1000 pixels de haut**. Utiliser le logarithme dans la présentation de ce genre de données permet de mieux en rendre compte graphiquement.

- Le changement de variable $(t, f(t)) \longrightarrow (t, \log_b f(t))$, c-à-d. afficher $\log_b f(x)$ plutôt que $f(x)$ à l'abscisse x, rend le graphe de f plus lisible et l'hypothèse de croissance exponentielle facile à vérifier si les points s'alignent sur une droite ; on appelle cela la projection lin × log.
- Le changement de variable $(t, f(t)) \longrightarrow (\log_b t, \log_b f(t))$, c-à-d. afficher $\log_b f(x)$ à l'abscisse $\log_b x$ plutôt que $f(x)$ à l'abscisse x forme la projection log × log. Son avantage est que le graphe des phénomènes qui ont une croissance polynomiale y tend vers une droite dont la pente est le degré du polynôme. C'est un moyen très simple de vérifier si un phénomène a une croissance polynomiale ou non, et si oui quel est son degré.



projection lin × log ⟶



projection log × log ⟶

② Quelques sommes et produits de suites finies

La somme et le produit étant des opérations associatives, commutatives, ayant chacune un élément neutre, on en dérive des opérations de sommes et produits itérés : $\sum_{i\in[bi\,bs[} t_i$ et $\prod_{i\in[bi\,bs[} f_i$ où les t_i et f_i sont des termes et facteurs dépendants de i. Si l'intervalle des indices est vide, la valeur conventionnelle de l'expression est l'élément neutre de l'opération, ex. 0 pour la somme et 1 pour le produit, mais aussi Faux pour la disjonction (∨), qui est une sorte de somme, et Vrai pour la conjonction (∧), qui est une sorte de produit.

Certaines sommations doivent être connues. Il est même bon de savoir les retrouver :
- $\sum_{i\in[1\,n[} i$: la somme des n 1^{ers} entiers est égale $n\times(n+1)\,/\,2$. Ex. $\sum_{i\in[1\,17[} i = 17\times18\,/\,2 = 153$.
- $\sum_{i\in[0\,n[} x^i$: la somme des n $1^{ères}$ puissances de x est égale à $(x^n-1)\,/\,(x-1)$. Ex. $\sum_{i\in[0\,n[} 2^i = 2^n - 1$.

Il est bon aussi de se rappeler quelques transformations élémentaires :
- $\sum_{i\in[0\,n[} (k\times t_i) = k\times\sum_{i\in[0\,n[} t_i$ et $\sum_{i\in[0\,n[} k = k\times n$, si k ne dépend pas de i.
- $\sum_{i\in[0\,n[} (s_i + t_i) = \sum_{i\in[0\,n[} s_i + \sum_{i\in[0\,n[} t_i$. Attention, rien de tel pour $\sum_{i\in[0\,n[} (s_i \times t_i)$!

③ Éléments d'une théorie des systèmes

Si on veut que système judiciaire, système nerveux, système solaire et système de gestion de fichiers partagent une notion commune de système on ne peut que la définir d'une façon très vague. On dira qu'un système est caractérisé par des entités et par les relations qu'elles entretiennent. Ex. les entités du système solaire sont le Soleil, les planètes et tous les corps qui orbitent autour du Soleil et des planètes, tous entretenant des relations d'attraction gravitationnelle. De même, les entités d'un système de fichiers sont des fichiers et des répertoires qui entretiennent une relation de contenant-contenu : **contient ⊆ répertoires × {fichiers ∪ répertoires}**, car un répertoire peut contenir des fichiers et des répertoires.

Un système peut être naturel, ex. le système solaire, ou artificiel, ex. le système judiciaire. Qu'il soit naturel ou non, un système peut être une vue de l'esprit, un modèle, construit pour simplifier l'analyse, ex. le système Terre-Lune, ou le système judiciaire quand il fait abstraction des questions sociales ou de santé mentale. Un système a en général une notion de dedans et de dehors qui permet de déterminer quelles sont les entités du système. Cette frontière peut être conventionnelle, ex. les limites du système solaire, ou être concrète, ex. la membrane plasmique des cellules biologiques. De même, les entités peuvent être concrètes, ex. la Lune, ou bien abstraite, ex. une société anonyme. Même concrètement délimité, un système peut être ouvert ou fermé. Dans le premier cas, il entretient des relations avec son extérieur, dans le second cas, non. En pratique, il n'existe pas de système fermé, même si on essaye souvent d'en construire en laboratoire dans des situations d'expérimentation, ex. en thermodynamique.

Le plus souvent on décrit un système dans une phase stable, c'est-à-dire obéissant à des règles qui semblent pouvoir durer indéfiniment, et qui semblent durer depuis toujours, ex. le système solaire, la vie cellulaire, la nationalité française, le cycle du carbone en climatologie. Ce sont des vues de l'esprit commodes, mais ça ne dit rien de la création de ces systèmes dont on sait bien qu'ils n'ont pas toujours existé ainsi et qu'ils n'existeront pas toujours. On parlera de phase transitoire pour la période où de nouvelles règles se mettent en place.

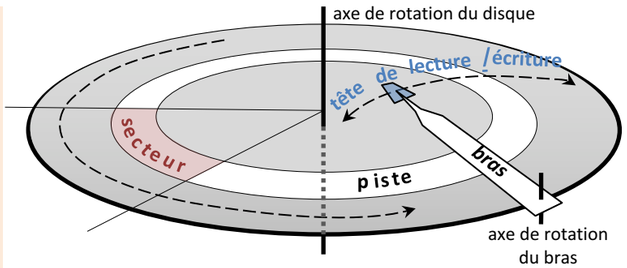
Les entités d'un système peuvent elles-mêmes être analysées comme des systèmes ; on les appellera des sous-systèmes. Souvent, on attribue à des systèmes des propriétés qu'on n'attribue pas à leurs sous-systèmes. Ex. la vie, propriété qu'on attribue aux cellules et aux systèmes biologiques dont elles sont des sous-systèmes, mais qu'on n'attribue pas aux organites et biomolécules qu'elles contiennent. Il en est de même pour les propriétés macroscopiques des matériaux, ex. couleur, brillance ou dureté, qu'on ne peut pas attribuer aux atomes dont ils sont composés. On appelle ces propriétés des propriétés émergentes. Ce concept est parfois résumé dans l'adage Le tout est plus que la somme des parties, mais aussi parfois contesté en affirmant qu'il n'y a pas émergence mais simplement ignorance des mécanismes profonds de ces systèmes.

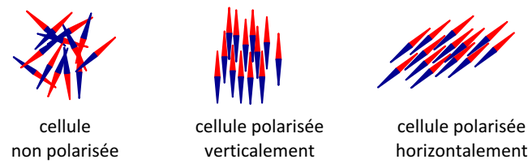
④ Un sous-système nommé disque dur magnétique

Un support de stockage de masse persistant permet de mémoriser une grande quantité de données sans autre apport d'énergie que pour lire et écrire. Le disque dur magnétique (*hard drive*, HD) est un support de stockage persistant qui coexiste dans les systèmes informatiques avec la bande magnétique et le disques SSD.

En 1956, IBM commercialisait le premier disque HD. Son principe est celui d'un disque fait d'un matériaux léger et rigide, ex. verre ou aluminium, revêtu d'une couche ferromagnétique et tournant à vitesse constante autour d'un axe normal à sa surface (aujourd'hui, environ 5000–7000 tr/min, soit env. 1 tour en 10 ms). La surface est divisée en anneaux concentriques qu'on appelle des pistes.

Soumis à un champ magnétique localisé, des groupes de particules ferromagnétiques (formant chacun une cellule) s'orientent (se polarisent) de façon cohérente alors que les groupes voisins gardent leur position initiale. Inversement, une cellule polarisée induit une variation localisée de champ magnétique. On convient donc que les deux valeurs possibles d'un bit de mémoire correspondent à deux polarisations opposées d'une cellule. Le principe de la lecture/écriture est de déplacer au-dessus du disque une tête de lecture/écriture capable d'induire ou de détecter une variation de champ magnétique. À cette fin, la tête de lecture est placée au bout d'un bras qui peut osciller dans le plan du disque de tel sorte que la tête puisse survoler à la demande n'importe quelle piste du disque.





cellule non polarisée — cellule polarisée verticalement — cellule polarisée horizontalement

La surface du disque est aussi divisée en secteurs géométriques, et l'intersection d'une piste et d'un secteur géométrique est simplement appelée secteur. Pour fixer les idées on peut dire qu'un disque comporte quelques dizaines de secteurs géométriques et quelques dizaines de milliers de pistes. La capacité d'un secteur est le nombre de cellules magnétiques qu'on peut y distinguer : ex. quelques millions de millions. Les nombres de cellules par secteur, nombres de secteurs par pistes et nombres de pistes sont les facteurs qui déterminent la capacité totale du disque. Depuis 1956, ces nombres n'ont fait qu'augmenter, permettant à la fois la diminution de la taille des disques et l'augmentation exponentielle de leur capacité (environ un doublement tous les deux ans).

L'unité d'échange avec un disque dur magnétique est le secteur. Le secteur désiré est identifié par ses numéros de piste et de secteur géométrique. Que se soit pour lire ou écrire un secteur il faut d'abord positionner la tête de lecture au-dessus de sa piste, puis attendre que la rotation du disque amène le secteur désiré sous la tête de lecture/écriture, puis faire l'opération demandée (lire ou écrire). Le positionnement du bras et la rotation du disque (1/2 tour en moyenne, soit environ 5 ms) font qu'un accès disque peut prendre quelques dizaines de millisecondes, alors qu'un accès mémoire ne prend que quelques dizaines de nanosecondes. C'est un million de fois plus lent.

Le disque dur à semi-conducteurs, (*solid-state drive*, SSD) est une autre solution de stockage de masse persistant qui remplace progressivement le disque dur magnétique. Utilisant une toute autre technologie que le disque dur magnétique, le transistor à grille flottante, il partage cependant avec lui une unité d'échange assez volumineuse (env. 4000 octets), et un temps de traitement meilleur mais encore assez long, environ 0,1 ms, soit 10 mille fois plus lent que la mémoire.

⑤ Quelques dates

1936 : publication par Alan Turing de son modèle de calcul et preuve de l'existence de machines de Turing universelles où un programme est traité comme une donnée.
1937 : démonstration par Claude Shannon de ce que l'algèbre de Boole est un bon modèle du calcul électronique.
1945 : publication par John Von Neumann de son modèle d'architecture qui réalise l'idée de programme en mémoire traité comme une donnée.
1948 : publication par Shannon de sa théorie de l'information. Cette théorie gouverne une grande partie des systèmes de communication, mais aussi des systèmes de calcul comme ceux de *deep learning*.
1961 : publication par Rolf Landauer de son principe qui relie calcul et énergie. Principe largement ignoré des concepteur de systèmes informatiques.
1972 : publication du principe de localité par Peter Denning. Ce principe gouverne un très grand nombre de dispositifs d'accélération des calculs et des communications. Remarquons du même auteur ce beau principe :

Solidarity, not software, generates collaboration.

⑥ Quelques chiffres

Les chiffres qui suivent sont mondiaux (population > 7,7 milliards). Ils sont de 2018-2019. L'unité de temps est l'année, sauf quand une autre unité est spécifiée.

- Plus de 4 milliards d'internautes ; la moitié en Asie (un milliard égale 10^6 ou environ 2^{30}).
- Environ 28 milliards d'objets connectés.
- Plus de 1,9 milliard de sites web utilisant plus de 333 millions de noms de domaine.
- Plus de 2000 milliards de recherches internet (65 000 par seconde) ; 92 % via Google.
- Environ 260 millions de PC vendus (~ 8 par seconde), 160 millions de tablettes et 1400 millions de smartphones.
- Environ 150 000 péta-octets de trafic internet (un péta vaut 10^{15} ou environ 2^{50}).
- Un système logiciel complexe (ex. Linux, Windows ou Android) compte plusieurs millions de lignes de code.
- Le dépôt des logiciels de Google compte plusieurs milliards de lignes de code.
- Un processeur intégré compte plusieurs milliards de transistors.

⓪ N'oubliez jamais !

Un modèle est toujours imparfait, mais peut être utile, et c'est tout ce qu'on peut lui demander (d'après George Box 1978, voir aussi les cartes à l'échelle 1/1 de Jorge Luis Borges 1946 et celles de Lewis Carroll 1893 pour des exemples de perfection inutilisable).

L'informatique ne travaille que sur la représentation des choses ; il lui faut des capteurs et des actionneurs pour toucher les choses. Par contre, travailler sur des représentations de représentations de … est courant. Par exemple, 440 est une représentation d'un entier qui est la mesure en hertz d'une fréquence, 0x1B8 est une autre représentation du même entier, et $1a_3$ est une autre représentation de la même fréquence. La fréquence double, l'octave, pourrait être représentée par 880, 0x370 ou $1a_4$. Il faut toujours se demander ce qui a du sens par rapport à ce qui est modélisé et ce qu'on va en faire.