# Mac OS X 10.6 Léopard des Neiges

## La revue de Ars Technica

Traduit par Philippe Lecarpentier du site Mac Puissance dix qui a obtenu l'autorisation de John Soracusa, l'auteur de cet article :

«Pour ceux qui peinent avec la langue de Shakespeare, je me suis résolu à proposer quelques traductions d'articles que je considère comme fondamentaux et dont la fréquentation peut être utile. Gros travail, que la traduction intégrale d'articles qui peuvent être parfois assez longs (puisque justement, je les considère comme importants). C'est une œuvre de longue haleine, et ne vous étonnez pas que tout ce qui est prévu ou annoncé ne soit pas immédiatement disponible.

Pour l'instant (septembre 2009), j'ai paré au plus pressé en vous proposant l'article de John Siracusa paru le 30 Août 2009 sur Léopard des neiges. Siracusa a proposé un article détaillé à la sortie de chacune des versions de Mac OS X ; leur étude, et leur comparaison sont pleines d'enseignements. La suite viendra peut être un jour.

Bonne lecture...» http://www.macpuissancedix.com/100.php

Mac OS X 10.6, autrement dit Léopard des neiges, est arrivé. Pour cette fois, Apple fait dans la discrétion, pour ce qui est un énorme travail sous le capot. John Siracusa plonge profondément dans ce que propose le nouvel OS pour voir ce qui est nouveau, ce qui reste le même, et si la mise à jour vaut le coup.

Un article de John Siracusa, traduit par Philippe Lecarpentier, d'après la dernière mise à jour datée du 31 Août 2009, 10h00 PM CT


*Tiger 10.4 : 150 nouveautés*

En Juin 2004, pendant la conférence de présentation de la WWDC, Steve Jobs révélait Mac OS X 10.4 Tiger pour la première fois aux développeurs et au public. Quand le produit fini est arrivé en Avril 2005, Tiger était la livraison la plus grosse, la plus importante, la plus riche en nouvelles possibilités de toute l'histoire de Mac OS X, et de loin. La campagne de marketing d'Apple avait orchestré cela en vantant "plus de 150 nouveautés".

Toutes ces nouvelles caractéristiques avaient pris du temps. Depuis son introduction en 2001, il y avait eu au moins une mise à jour majeure de Mac OS X chaque année. Tiger a mis plus d'un an et demi à voir le jour. A l'époque, cela valait incontestablement le coup d'attendre. Tiger fit impression parmi les utilisateurs et les développeurs. Apple en a retenu la leçon pour provoquer un coup de cœur et l'attente envers une nouvelle mise à jour de Mac OS X, Léopard. Par plusieurs canaux de communication, Apple manifesta son intention de changer le cycle des mises à jour de Mac OS X de 12 mois à 18 mois. Léopard fut officiellement programmé pour "le printemps 2007".

A mesure que la date approchait, la machine de marketing d'Apple, emprunta une voie prévisible.


*Steve Jobs à la WWDC de 2007 : 300 nouveautés dans Léopard*

Apple alla assez loin pour fournir une liste de 300 nouvelles caractéristiques sur son site web. Avec la tournure des choses, le "printemps"" se révéla un peu optimiste. Leopard ne fut en fait envoyé qu'à la fin d'Octobre 2007, presque deux ans et demi après Tiger. Ce qui est sûr, c'est que Léopard contenait une solide moisson de caractéristiques et de technologies, et que nous en considérons maintenant beaucoup comme dues. (Par exemple, avez-vous discuté avec un utilisateur potentiel de Mac depuis la sortie de Leopard *sans* mentionner Time Machine ? Je m'en suis bien gardé).

Mac OS X semblait devenir mature. La progression était claire : des cycles plus longs, plus de possibilités. A quoi allait ressembler Mac OS X 10.6 ? Est-ce qu'il arriverait trois ou trois ans et demi après Léopard ? Est-ce qu'il allait rassembler 500 nouvelles possibilités? Ou un millier ?

A la WWDC de 2008, Bertand Serlet annonça un changement qu'il qualifia de "sans précédent" dans l'industrie du PC.


*Mac OS X 10.6 : lisez sur les lèvres de Bertrand Serlet : aucune nouveauté*

C'était vrai, la nouvelle mise à jour majeure de Mac OS X n'aurait *aucune* nouvelle possibilité. Le nom du produit reflétait cela : "Léopard des neiges". Mac OS X 10.6 ne serait simplement qu'une variante de Léopard. Meilleure, plus rapide, plus raffinée, euh... neigeuse.

C'était une stratégie risquée pour Apple. Après le feu nourri des mises à jour de 10.1, 10.2, et 10.3, suivi de la bataille des nouvelles possibilités et des API dans 10.4 et 10.5, Apple pouvait-il vraiment se permettre un "hors jeu" ? J'imagine que Bertrand transpirait beaucoup en faisant cette annonce sur scène à la WWDC devant une assemblée attentive de développeurs Mac. Et leur réaction ? Des *applaudissements* spontanés. Il y eu même quelques hululements et sifflets.

Beaucoup de ces même développeurs avaient applaudi aux "150 Nouvelles possibilités" de Tiger, et aux "300 nouvelles possibilités" de Léopard aux précédentes WWDCs. Maintenant, ils applaudissaient au *zéro* nouvelle possibilité de Léopard des neiges. Qu'est-ce qui peut expliquer cela ?

Cela va sans doute aider de savoir que la diapo "0 nouvelle possibilité" venait à la fin d'une présentation d'une heure, qui avait détaillé les principales APIs nouvelles et les technologies de Léopard des neiges. Elle fut aussi rapidement suivie en rétro-pédalage (en fait, il y a *une* nouvelle possibilité) d'une diapo qui montrait l'addition du support de Microsoft Exchange. Isolé de son contexte, "0 nouvelle possibilité"

peut sembler représenter la stagnation. Mais dans le contexte, c'était un message amical délivré aux développeurs.

Le message d'Apple à destination des développeurs était du genre : "Nous ajoutons des tonnes de choses nouvelles à Mac OS X pour vous aider à écrire de meilleures applications, en rendre votre code existant plus rapide, et nous allons nous assurer que tout cela est solide comme du roc, et aussi dépourvu de bogues que possible. On ne va pas s'épuiser à rajouter une barge de possibilités pour satisfaire le client et le marketing. A la place, nous allons nous concentrer 100 % sur les choses qui vous intéressent, vous, développeurs".

Mais si Léopard des neiges est une lettre d'amour aux développeurs, est-ce que c'est une lettre du genre Mon cher Jean aux utilisateurs ? Vous savez, ces gens que le service de marketing appelle si crûment des "clients". Qu'en est il pour eux ? Croyez-le ou non, la difficulté de persuader les acheteurs est en fait tout à fait semblable. Aussi épuisante qu'a été l'obligation pour les développeurs de se maintenir à flot dans un flux incessant de nouvelles APIs, aussi pénalisant ce peut être pour les clients de se maintenir au top niveau des nouvelles possibilités de Mac OS X. Exposé, un nouveau Finder, Spotlight, un nouveau Dock, Time Machine, un nouveau Finder encore, un nouveau iLife, et un nouvel iWork presque tous les ans, et ainsi de suite. Et, autant les développeurs détestent les bogues dans les nouvelles APIs d'Apple, autant les utilisateurs qui expérimentent ces bogues dans les applications ont les mêmes raisons d'être irrités.

Et puis, entrée de Léopard des neiges : la mise à jour qui nous permet à tous une pause dans la cage d'écureuil -nouvelles possibilités/nouvelles bogues- du développement de Mac OS X. Voilà la persuasion.

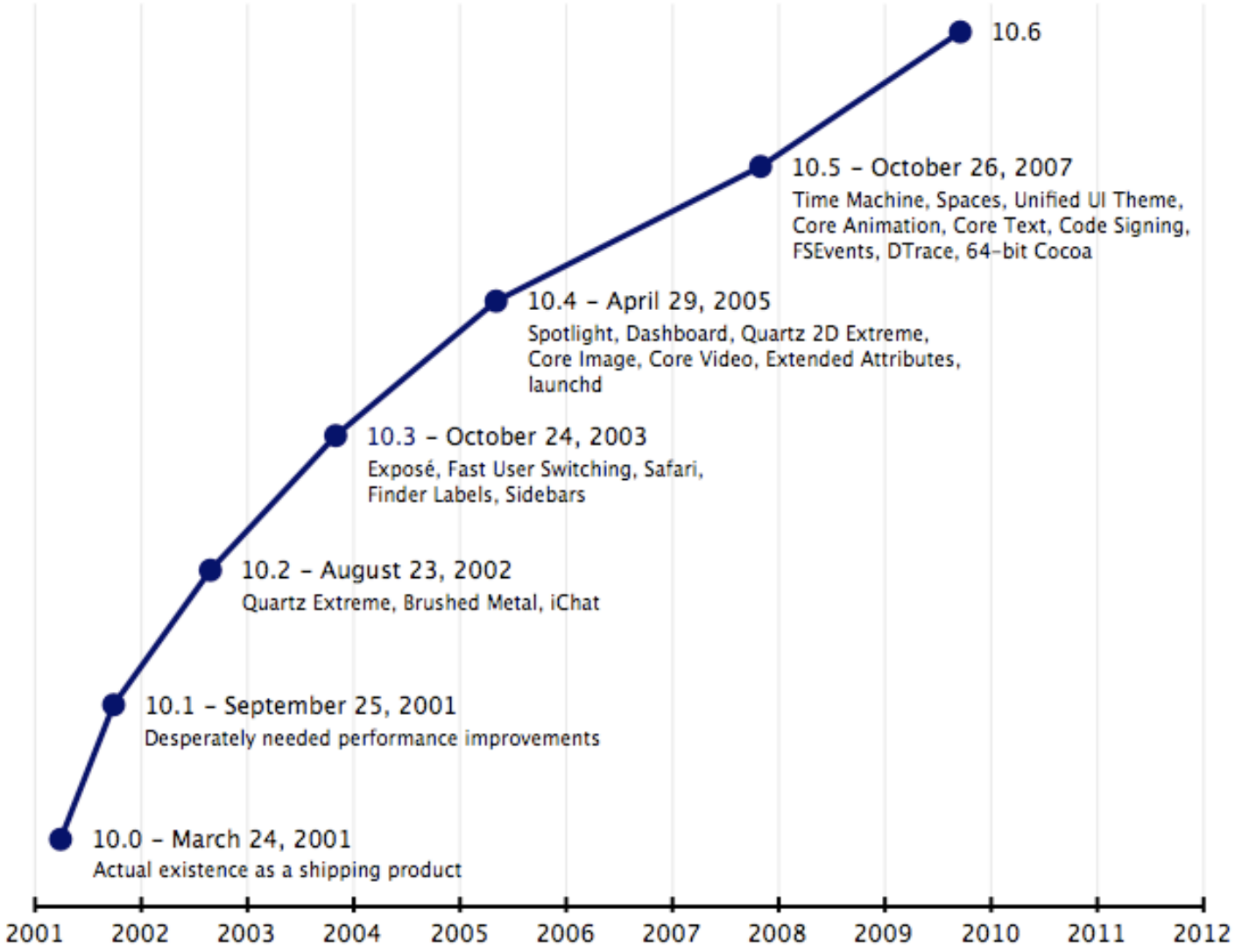
**Réalités inconfortables**

Mais attendez un peu, n'ai-je pas mentionné "une présentation d'une heure" avec Léopard des neiges proposant de nouvelles APIs et de nouvelles technologies ? Quand il parle aux développeurs, le message d'Apple "pas de nouvelles possibilités" est une autre façon de dire "pas de nouvelles bogues". Léopard des neiges est supposé supprimer les vieilles bogues sans en introduire de nouvelles. Mais rien n'annonce mieux que de nouvelles APIs majeures de nouvelles bogues inévitables.

De la même façon, pour les utilisateurs, "pas de nouvelles possibilités" sous-entend stabilité et fiabilité. Mais si Léopard des neiges contient assez de changements dans le cœur de l'OS pour meubler une longue heure d'introduction à la WWDC, plus d'un an avant sa sortie, Apple peut-il réellement tenir sa promesse ? Ou bien les

utilisateurs vont-ils cumuler les désavantages d'une mise à jour pleine de nouveautés comme Tiger ou Léopard, - les inévitables bogues 10.x.0- et de nouvelles fonctionnalités non familières et non testées, sans avoir l'avantage de nouvelles possibilités ?

Oui, c'est assez pour porter un regard cynique sur les motivations réelles d'Apple. Et pour mettre un peu plus d'huile sur le feu, jetez un coup d'œil sur la courbe temporelle des mise à jour d'Apple ci-dessous. Pour chaque mise à jour, j'y ai fourni une liste de ses améliorations les plus significatives.




*L'échelonnement des livraisons de Mac OS X*

Cette courbe prend une allure indéniablement pendante, comme si elle était alourdie par un nombre toujours plus important de nouvelles possibilités. (Ces mises à jour sont réparties uniformément sur l'axe des Y.) Peut-être pensez-vous qu'il serait raisonnable d'étaler le temps entre les révisions du fait que chacun apporte un lot d'améliorations plus important que la précédente, mais gardons à l'esprit la conséquence logique d'une telle courbe sur un long~~horn~~ effort.

Ah oui, il y a une petite poussée vers le haut à la fin, pour 10.6, mais elle est supposée être une livraison "sans nouvelles possibilités". La version 10.1 se voulait aussi sans fioritures, mais a pris sacrément moins de temps à venir.

En voyant ce graphique, il est difficile de ne pas se demander s'il n'y a pas quelque chose qui siphonne les ressources dans l'effort de développement de Mac OS X. Disons, peut-être un projet qui en est à sa première, deuxième ou troisième étape de la vie, encore dans la partie abrupte de la courbe en ce qui concerne son propre développement. Oui, je veux parler de l'iPhone, et particulièrement de iPhone OS. Le business de l'iPhone a explosé, sur les bilans comptables d'Apple comme aucun autre produit ne l'a fait avant, même le iPod. Il attire aussi l'appétit des développeurs à un taux alarmant.

Il n'est pas difficile d'imaginer que de nombreux artistes et développeurs, qui ont planché sur les caractéristiques visibles par l'utilisateur ont été réorientés (temporairement ou non) sur l'OS de l'iPhone. Après tout, Mac OS X et iPhone OS partagent le même cœur de système d'exploitation, le même langage pour le développement de l'Interface Utilisateur Graphique, et beaucoup des mêmes APIs. La migration d'une certaine force de travail semble avoir été inévitable.

Et n'oublions pas que les technologies de "Mac OS X" que nous avons apprises récemment ont été développées pour l'iPhone, et qu'il s'est trouvé qu'elles ont été annoncées pour le Mac d'abord (parce que l'iPhone était encore un secret), comme Core Animation et la signature de code. Une telle conspiration du silence n'est certainement pas rehaussée par le dédain et le mépris manifesté à la présentation d'introduction de la WWDC envers Mac OS X et le Mac en général depuis que l'iPhone est arrivé sur le devant de la scène. Si bien que, par dessus tout, Léopard des neiges a pour objet de redonner quelque lustre à Mac OS X.

Vous avez compris ? A peu près deux ans de cycle de développement, mais pas de nouvelles fonctionnalités. De nouveaux frameworks majeurs pour les développeurs, mais peu de nouvelles bogues. Des changements significatifs dans le cœur de l'OS, mais une plus grande fiabilité. Et une cure de jouvence pour agir, avec peu de changements visibles pour l'utilisateur.

Ça suffit pour rendre blanc le Léopard.

# Le ticket d'entrée

L'ouverture en musique sur Léopard des neiges commence par son prix : 29 $ (29 Euros) pour une mise à jour à partir de Léopard. La première livraison de Mac OS X, 10.0, et les quatre autres livraisons majeures qui ont suivi ont été proposées à 129 $, sans prix spécial pour les mises à jour. Après 8 ans de ce genre de pratique financière, les utilisateurs de Léopard pourraient bien être tentés d'arrêter de lire la suite, et d'acheter leur copie tout de suite. Le prix de mise à jour de Léopard des neiges est bien en dessous du seuil reflex d'achat pour beaucoup de gens. Vingt neuf dollars (Euros), plus un minimum de confiance en Apple, pour sa capacité à améliorer l'OS à chaque version, et boum, j'achète tout de suite.

Vous êtes encore là ? Bien, parce qu'il y a encore quelque chose que vous avez besoin de savoir à propos de Léopard des neiges. C'est une ouverture d'un genre particulier, plutôt en forme de bâton que de carotte. Léopard des neiges ne fonctionne que sur les Macs à CPU Intel. Désolé (encore une fois), amoureux du PPC, mais c'est la fin de la route pour vous. La transition vers Intel a été annoncée il y a quatre ans, et le tout dernier Mac à Power PC a été livré en Octobre 2005. C'est fini.

Mais si Léopard des neiges a pour objet de pousser ceux qui font de la résistance avec leur Power PC dans le nouvel âge Intel, la déclaration "pas de nouvelles fonctionnalités" (et le manque d'attrait visuel supplémentaire) vont à l'encontre. Pour ceux qui ont Léopard sur un Mac à PPC, il n'y a pratiquement rien dans Léopard des neiges qui puisse les inciter à faire l'achat d'un nouveau Mac à quatre chiffres. Pour les possesseurs de Macs à PPC, le seuil pour l'achat d'un nouveau Mac reste inchangé. Quand leur vieux Mac tombera en panne ou semblera vraiment trop lent, ils vont en acheter un nouveau, et il viendra avec Léopard des neiges pré-installé.

Si Léopard des neiges finit par pousser des possesseurs de Macs PPC vers de nouveaux Macs, ce sera plutôt par résignation que par envie. Léopard des neiges sur Intel seulement, est plus significatif pour ce qu'il n'est pas : une extension supplémentaire de la durée de vie des PPCs sur la plateforme Mac.

Un dernier groupe intéressant est celui des possesseurs de Macs Intel qui tournent encore sous Tiger, Mac OS X 10.4. Apple a livré des Macs Intel avec Tiger installé

dessus pendant un peu plus d'un an et neuf mois. Les possesseurs de ces machines, qui n'ont jamais fait la mise à jour vers Léopard n'ont pas le droit à la mise à jour pour 29 $ de Léopard des neiges. Apparemment, ils n'ont pas le droit non plus d'acheter Léopard des neiges pour le prix traditionnel de 129 $. Voici ce que Apple a à dire à propos du prix de Léopard des neiges.

> La version Mac OS X 10.0 Léopard des neiges sera disponible comme mise à jour de la version 10.5, Léopard, de Mac OS X en Septembre 2009 [...]. La licence mono-utilisateur de Léopard des neiges sera proposée au prix de détail suggéré de 29 $ (US) et le pack familial de Léopard des neiges, une licence pour 5 utilisateurs vivant dans le même foyer, au prix de détail suggéré de 49 $ (US). Pour les utilisateurs de Tiger® avec un Mac Intel, l'ensemble Mac Box inclut Léopard des neiges, iLife®'09 et iWork®'09, et sera disponible au prix suggéré de 169 $ (US), et un pack familial au prix suggéré de 229 $(US).

Si on fait abstraction pour le moment des packs familiaux, cela veut dire que Léopard des neiges sera gratuit sur votre nouveau Mac, à 29 € si vous avez déjà Léopard, ou à 169 € si vous avez un Mac Intel sous Tiger. Les gens qui font la mise à jour à partir de Tiger auront en prime (si on peut dire) la dernière version de iLife et de iWork, qu'ils le veuillent ou non. Il semble bien qu'il y aurait place pour une mise à jour vers Léopard des neiges à 129 €. Mais en fait, tout dépend peut être de la façon dont Apple entend faire respecter sa mise à jour de Léopard des neiges à 29 €.

(En guise de remarque pour les non utilisateurs de Macs, notez que la version non-serveur de Mac OS X est dépourvue d'un numéro de série pour chaque utilisateur, qu'il n'y a aucune sorte de processus d'activation, et qu'il n'y an a jamais eu. L'enregistrement pratiqué par Apple pendant l'installation est totalement optionnel, et ne sert qu'à collecter une information démographique. L'absence d'enregistrement, (ou la fourniture d'une information complètement erronée) n'a aucun effet sur votre possibilité à utiliser l'OS. Ceci est considéré comme un avantage authentique de Mac OS X, mais cela signifie aussi qu'Apple ne dispose d'aucun enregistrement fiable de qui, exactement, est un utilisateur "légitime" de Léopard.)

Une possibilité aurait été que le DVD de mise à jour de Léopard des neiges à 29 € ne s'installe que sur une installation existante de Leopard. Apple a fait ce genre de chose auparavant, et cela court-circuite tous les embêtements de la preuve d'achat. Mais, cela aurait introduit un nouveau problème : dans l'éventualité d'une panne de disque dur, ou sur une simple décision de refaire une installation propre, les possesseurs d'une mise à jour à 29 € de Léopard des neiges auraient été forcés de

réinstaller d'abord Léopard, puis par dessus, Léopard des neiges, ce qui aurait au moins doublé le temps d'installation, et multiplié par cinq les ennuis.

Compte tenu des antécédents d'Apple dans ce domaine, on peut être surpris de constater qu'Apple a choisi l'option la plus simple : le DVD de mise à jour de 29 € de Léopard des neiges va s'installer en fait sur tout Mac qui le permet, que Léopard soit installé ou non dessus. Il va même s'installer sur un disque complètement vierge.

Pour être clair, l'installation de la mise à jour à 29 € de Léopard des neiges sur un système qui ne contient pas une copie légitime de Léopard est une violation de l'agrément de licence de l'utilisateur final qui vient avec le produit. Mais la décision d'Apple est un changement tonique : remercier les gens honnêtes avec un produit dépourvu de désagréments, plutôt que de tenter de punir les gens malhonnêtes en traitant tout le monde comme délinquant. Cette pratique de mise à jour basée sur un code d'honneur explique en partie le saut énorme à 169 € de l'ensemble Max Box, qui fournit un cadre honnête pour obtenir iLife et iWork à leur prix normal, et Léopard des neiges pour 11 € de plus.

Et puisqu'on parle d'installation, venons y finalement.

# L'installation

Apple fait savoir que le processus d'installation de Léopard des neiges est "jusqu'à 40 % plus rapide". Les durées d'installation varient considérablement en fonction de la vitesse, du contenu, et de la fragmentation du disque, de la vitesse du lecteur de disque optique, et d'autres choses. Mais l'installation n'intervient qu'une fois, et ce n'est pas quelque chose de très intéressant, à moins que quelque chose n'aille très mal. Toutefois, puisque Apple fait cette promesse, autant la tester.

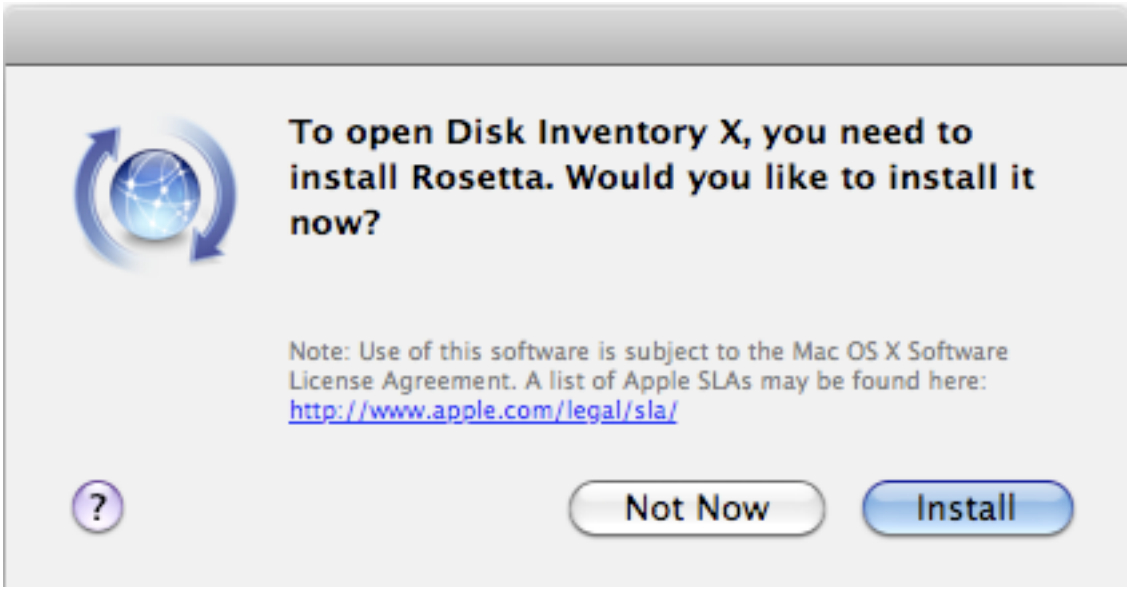
Pour éliminer autant de variables que possible, j'ai installé à la fois Léopard et Léopard des neiges d'un disque dur sur un autre qui était vide. Il faut noter que cette modification empêche quelques unes des plus importantes optimisations d'installation de Léopard des neiges, qui se concentrent sur la réduction de l'accès aléatoire aux données à partir du disque optique.

Même avec ce désavantage, l'installation de Léopard des neiges a pris environ 20 % de moins de temps que celle de Léopard. C'est loin de la déclaration d'Apple "jusqu'à 45 %", mais voyez ce qui précède (et n'oubliez pas la précaution d'Apple : "jusqu'à"). Les deux versions se sont installées en moins de 30 minutes.

Ce qui est frappant, avec l'installation de Léopard des neiges, c'est avec quelle rapidité le processus d'indexation de Spotlight s'est fait. Là, Léopard des neiges a été 174 % plus rapide, selon mon test. Bien sûr, les temps sont courts 5mn49 contre 3mn30, et encore une fois, une nouvelle installation sur un disque vide n'est pas la norme. Mais l'attente plus courte pour l'indexage de Spotlight vaut d'être notée parce que c'est la première indication que les utilisateurs vont avoir que Léopard des neiges signifie efficacité en raison de sa performance.

Une autre chose notable dans l'installation est **ce qui n'est pas** installé par défaut : Rosetta, l'outil qui permet aux binaires PowerPC de tourner sur des Macs Intel. Bon, Apple, on a compris. Le Power PC est contraignant et dépourvu de vie. Il repose en paix. Il est passé derrière le rideau, et a rejoint le chœur invisible. Pour Apple, le Power PC est un ex-ISA (un jeu d'instructions révolu).

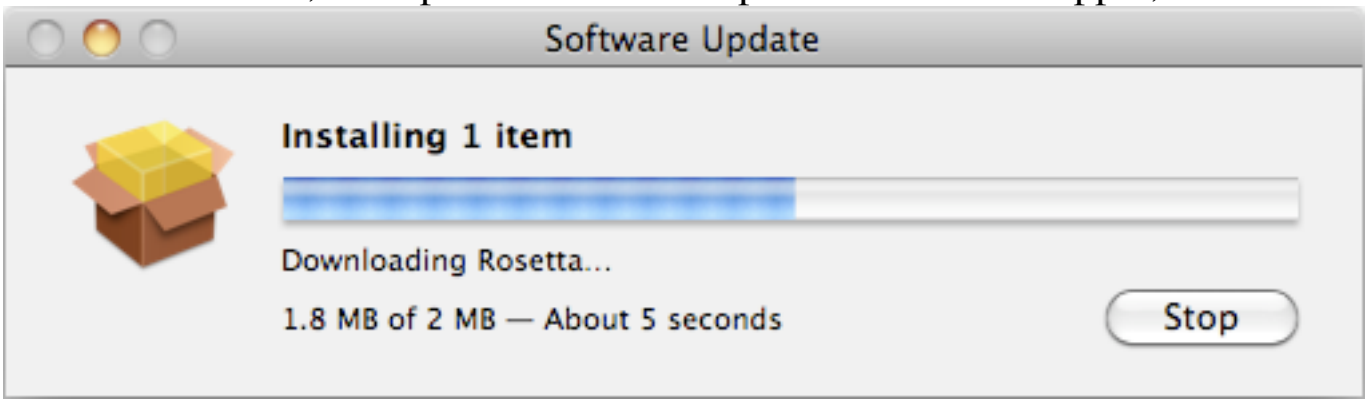
Mais, ne pas installer Rosetta par défaut ? Cela semble un peu rude, et même grossier. Que va-t-il arriver à tous ceux qui auront fait la mise à jour à Léopard des neiges, et qui vont double-cliquer sur une application dont ils ont oublié depuis longtemps qu'elle était faite pour un Power PC ? D'une façon surprenante, peut-être, voici ce qui arrive :




*Rosette : installation automatique pour votre confort*

C'est ce que j'ai vu apparaître quand j'ai cherché à lancer Disk Inventory X sur Léopard des neiges, une application dont j'avais oublié depuis longtemps qu'elle

était pour PPC seulement. Après avoir cliqué sur le bouton Install, je m'attendais à être pressé de mettre le DVD d'installation. Mais à la place, Léopard des neiges est allé sur Internet, a rapatrié Rosetta depuis le serveur Apple, et l'a installé.




Il n'y a pas eu besoin de redémarrer, et Disk Inventory X s'est chargé sans problème après l'installation de Rosetta. Mac OS X n'a pas fait grand usage de l'approche "installer à la demande" des composants logiciels du système, mais le moyen utilisé pour installer Rosetta semble tout à fait convenable. Quand on clique "Install", une property list XML contenant un vaste catalogue de packages disponibles pour Mac OS X a été téléchargée. Léopard des neiges utilise le même procédé pour installer les pilotes d'imprimantes à la demande, et évite un nouvel accès au DVD d'installation. J'espère que cette technique va être encore plus utilisée dans le futur.

## L'empreinte de l'installation

Rosetta mis à part, Léopard des neiges dépose simplement moins de bits sur votre disque. Apple prétend que cela prend "jusqu'à moins de la moitié d'espace disque que la version précédente". Une installation propre par défaut, (incluant les index Spotlight entiers) fait 16,8 Go pour Léopard, et 5,9 Go pour Léopard des neiges. (Incidemment, ces nombre sont tous deux puissances de deux ; voir l'aparté).

---

**Un Gigabyte ou un autre**

Léopard des neiges a un autre tour dans son sac quand il s'agit de l'usage du disque. Le Finder de Léopard des neiges considère que 1 Go est égal à $10^9$ (1 000 000 000 octets), alors que le Finder de Léopard - et notons-le de toutes les versions antérieures- assimile 1 Go à $2^{30}$ (1 073 741 824 octets). Cela a pour conséquence de faire apparaître votre disque soudain plus grand après l'installation de Léopard des neiges. Par exemple, mon disque de 1 To apparaît dans le Finder de Léopard avec 931,19 Go. Sous Léopard des neiges, il fait 999,86 Go. Comme vous l'avez sans doute deviné, les fabricants de disques durs utilisent le système en puissances de dix. C'est tout à fait désolant, en fait. Bien que je me tienne fermement du côté des défenseurs des puissances de deux, je ne peux pas trop blâmer Apple de vouloir se

mettre au niveau du standard de mesure des vendeurs de disques, établi de longue date (mais stupide, pensez-y).

---

Léopard des neiges a plusieurs secrets pour perdre du poids. Le premier est évident : l'absence de support du Power PC signifie qu'il n'y a pas de code Power PC dans les exécutables. Rappelez-vous le code binaire maximum dans un exécutable Léopard : 32 bits PowerPC, 64 bits PowerPC, x86, et x86_64. Maintenant, barrez la moitié de ces architectures de la liste. Je vous l'accorde, très peu d'applications dans Léopard utilisent un code 64 bits, mais c'est quand même une réduction de 50 % des exécutables, quelle que soit la façon dont vous comptez.

Bien sûr, tous les fichiers d'un système d'exploitation ne sont pas des exécutables. Il y a les fichiers de données, les images, les fichiers audio, et même un peu de fichiers vidéo. Mais la plupart de ces fichiers non exécutables ont une chose en commun : ils sont normalement stockés dans des formats de fichiers compressés. Les images sont en PNG ou en JPEG, l'audio est en AAC, la vidéo en MPEG-4, et les fichiers de préférences eux-mêmes, et autres property-lists (listes d'attributs) sont maintenant par défaut dans un format binaire compact plutôt qu'en XML.

Dans Léopard des neiges, d'autres formes de fichiers montent dans le wagon de la compression. Pour ne donner qu'un exemple, 97 % des fichiers exécutables de Léopard des neiges sont compressés. Compressés comment ? Regardons un peu :

```
% cd Applications/Mail.app/Contents/MacOS
% ls -l Mail
-rwxr-xr-x@ 1 root  wheel  0 Jun 18 19:35 Mail
```

Hé, c'est euh plutôt petit, non ? Est-ce que c'est vraiment un exécutable, ou quoi ? Essayons de tester nos affirmations.

```
% file Applications/Mail.app/Contents/MacOS/Mail
Applications/Mail.app/Contents/MacOS/Mail: empty
```

Ben quoi ? Qu'est-ce qui se passe ? Eh bien, ce que je ne vous ai pas dit, c'est que les commandes présentées ci-dessus ont été exécutées sur un système Léopard, inspectant un disque Léopard des neiges. En fait, tous les fichiers compressés de Léopard des neiges apparaissent avec 0 octets quand ils sont vus depuis une version de Mac OS X antérieure à Léopard des neiges. (Bien sûr, ils apparaissent et agissent de façon parfaitement normale sous Léopard des neiges).

Alors, où sont les données ? Le petit "@"à la fin de la chaîne des permissions dans le résultat de la commande ls ci-dessus (une caractéristique introduite dans Léopard) fournit un indice. Bien que l'exécutable Mail ait une taille de fichier de zéro, il a en fait quelques attributs étendus :

```
% xattr -l Applications/Mail.app/Contents/MacOS/Mail
com.apple.ResourceFork:
0000     00 00 01 00 00 2C F5 F2 00 2C F4 F2 00 00 00 32    .....,...,.....2
0010     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
         (184,159 lines snipped)
2CF610   63 6D 70 66 00 00 00 0A 00 01 FF FF 00 00 00 00    cmpf............
2CF620   00 00 00 00                                        ....

com.apple.decmpfs:
0000   66 70 6D 63 04 00 00 00 A0 82 72 00 00 00 00 00    fpmc......r.....
```

Ah, voilà toutes les données. Mais attendez, c'est dans la ResourceFork (fourche ressources)? N'ont-elles pas été abandonnées depuis environ 8 ans ? Bien sûr, elles l'ont été. Ce dont vous êtes témoins ici, est encore une autre addition au dada favori d'Apple en matière de système de fichiers, HFS +.

A l'aube de Mac OS X, Apple a ajouté la journalisation, les liens symboliques et les liens matériels. Dans Tiger, les attributs étendus et les listes de contre d'accès (ACLs) ont été ajoutés. Dans Léopard, HFS+ a gagné l'adjonction des liens matériels sur les répertoires. Dans Léopard des neiges, HFS + apprend un nouveau tour : la compression par fichier.

La présence de l'attribut com.apple.decmpfs est le premier indice que ce fichier est compressé. Cet attribut est en fait caché de la commande xattr quand on démarre sous Léopard des neiges. Mais vu d'un système Léopard qui n'a aucune connaissance de sa signification particulière, il est révélé en plein jour.

Plus d'information encore est fournie grâce au programme hfsdebug dans le livre Mac OS X Internals du gouru Amit Singh.

```
% hfsdebug /Applications/Mail.app/Contents/MacOS/Mail
  ...
  compression magic    = cmpf
  compression type     = 4 (resource fork has compressed data)
  uncompressed size    = 7500336 bytes
```

Et là c'est sûr, comme nous l'avons vu, la fourche ressource contient des données compressées. Mais encore, pourquoi la fourche ressource ? Cela fait partie de

l'habile gymnastique usuelle d'Apple pour assurer une compatibilité rétrograde. Un exemple récent dans cette voie est la façon dont les liens matériels sur des répertoires apparaissent - et fonctionnent- comme des alias vus depuis une version pré-Léopard de Mac OS X.

Dans le cas de la compression dans HFS +, Apple a été incapable (c'est compréhensible) de faire que les anciens systèmes, pré-Léopard des neiges puissent lire et interpréter des donnés compressées, qui sont stockées d'une façon qui n'existait pas quand ces systèmes plus précoces ont été écrits. Mais plutôt que de laisser les applications (et les utilisateurs) qui fonctionnent sur des systèmes antérieurs à 10.6 se battre avec -ou pire, corrompre après modification- le contenu de fichiers qu'on de sait pas identifier comme compressés, Apple a choisi de cacher les données compressées.

Et où le contenu d'un fichier qui peut être gros peut-il être caché de façon que les systèmes pré-Léopard puissent encore copier le fichier sans perdre de données ? Dans la fourche ressources, bien sûr. Le Finder a toujours préservé correctement les métadata spécifiques au Mac, et à la fois les fourches ressources et data, quand il déplace ou duplique des fichiers. Dans Léopard, même les commandes de bas niveau cp et rsync le font. Si bien que, quelque fantomatique que ce soit de voir tous ces fichiers "vides" à 0 Ko quand on regarde un disque Léopard des neiges depuis un système Léopard, le risque de perdre des données est faible même si vous déplacez ou copiez un de ces fichiers.

La fourche ressources n'est pas le seul endroit où Apple a décidé de passer ses données en contrebande. Pour de très petits fichiers hfsdebug montre ceci :

```
% hfsdebug /etc/asl.conf
  ...
  compression magic    = cmpf
  compression type     = 3 (xattr has compressed data)
  uncompressed size    = 860 bytes
```

Là, les données sont assez petites pour tenir entièrement dans l'attribut étendu bien que sous forme comprimée. Et puis, l'étape ultime :

```
% hfsdebug /Volumes/Snow Time/Applications/Mail.app/Contents/PkgInfo
  ...
  compression magic    = cmpf
  compression type     = 3 (xattr has inline data)
  uncompressed size    = 8 bytes
```

C'est vrai, voici le contenu d'un fichier entier stocké de façon non comprimée dans un attribut étendu. Dans le cas d'un fichier PkgInfo standard comme celui-ci, ces

contenus sont les codes classiques dans Mac OS, en quatre octets, du type et du créateur.

```
% xattr -l Applications/Mail.app/Contents/PkgInfo
com.apple.decmpfs:
0000   66 70 6D 63 03 00 00 00 08 00 00 00 00 00 00 00    fpmc............
0010   FF 41 50 50 4C 65 6D 61 6C                        .APPLemal
```

Il y a encore le même préambule "fpmc..." que nous avons vu dans tous les exemples précédents de l'attribut com.apple.decmpfs, mais à la fin, les données attendues apparaissent au grand jour : le code du type "APPL" (application) et le code du créateur "emal" (pour l'application Mail, pertinent comme dans la tradition de Mac OS.)

Vous vous demandez peut-être, puisqu'on est dans la compression de données, comment le fait de stocker 8 octets non compressés, plus un préambule de 17 bytes dans un attribut étendu sauve de l'espace disque ? La réponse tient dans la manière dont HFS + alloue l'espace disque. Quand il stocke de l'information dans une fourche data ou ressource, HFS + alloue l'espace en multiples de la taille du bloc d'allocation du système de fichiers (4 Ko par défaut). Si bien que ces 8 bits prendront un espace *minimum* de 4096 octets dans la façon traditionnelle de stocker. Quand on alloue de l'espace disque pour des attributs étendus, cependant, la taille du bloc d'allocation n'est pas en cause ; les données sont compactées de façon plus serrée. Au bout du compte, l'espace effectivement gagné en stockant ces 25 octets de données dans un attribut étendu dépasse 4000 octets.

Mais la compression ne sert pas seulement à sauver de l'espace disque. C'est aussi un exemple classique d'utilisation des cycles des CPUs pour diminuer le temps d'attente des entrées/sorties, et améliorer la bande passante. Pendant les dernières décades, la performance des CPUs s'est améliorée (et les ressources de calcul sont devenues plus abondantes -plus là-dessus plus tard), à une cadence beaucoup plus grande que la performance des disques ne s'est accrue. Le temps de positionnement des disques, et les délais de rotation se mesurent encore en millisecondes. En une milliseconde, un CPU à 2 GHz passe par deux millions de cycles. Et puis, il faut prendre en compte le temps effectif de transfert des données.

Je l'accorde, plusieurs niveaux de cache dans le système d'exploitation et le matériel contribuent efficacement à masquer ces délais. Mais tous ces bits doivent bien sortir du disque à un moment ou un autre pour nourrir ces caches. La compression implique que moins de bits doivent être transférés. Etant donnée l'abondance presque comique des ressources CPU sur un Mac moderne multicœurs en utilisation normale, le temps total nécessaire pour transférer une charge compressée depuis le disque et utiliser le CPU pour décompresser son contenu en mémoire sera

encore habituellement beaucoup plus faible que le temps que cela prendrait pour transférer des données non compressées.

Cela explique les bénéfices potentiels de performance obtenus en transférant moins de données, mais l'utilisation des attributs étendus pour stocker le contenu des fichiers peut effectivement rendre les choses plus rapides aussi. Tout cela a à voir avec le positionnement des données.

S'il y a une chose qui ralentit un disque dur plus que le transfert de grandes quantités de données, c'est le déplacement de ses têtes d'une partie à une autre du disque. Chaque déplacement implique du temps pour que la tête commence à se déplacer, puis s'arrête, puis qu'elle s'assure qu'elle s'est positionnée à la bonne place, puis qu'elle attende que le disque en rotation passe au bon endroit pour y déposer les bits voulus. Physiquement, ce sont des parties mobiles bien réelles, et il est étonnant qu'elles puissent effectuer leur danse aussi vite et aussi efficacement, mais la physique a ses limites. Ces mouvements réduisent réellement les performances pour les systèmes de stockage à rotation que sont les disques durs.

Le format des volumes HFS + stocke toutes ses informations sur les fichiers -les métadonnées- à deux endroits principaux sur un disque : le Fichier Catalogue, qui stocke les dates de fichier, les permissions, l'appartenance, et beaucoup d'autres choses, et le Fichier des Attributs, qui stocke les "fourches nommées".

Les attributs étendus dans HFS + sont implémentées comme fourches nommées dans le fichier des attributs. Mais à la différence des fourches de ressources, qui peuvent être très grosses (jusqu'à la taille maximum de fichier supportée pas le système de fichiers), les attributs étendus de HFS + sont stockés "en ligne" dans le fichier d'attributs. En pratique, cela signifie une limite des 128 octets par attribut. Mais cela signifie aussi que la tête de lecture n'a pas besoin d'entreprendre une migration vers une autre partie du disque pour accéder aux données réelles.

Comme vous pouvez l'imaginer, les blocs du disque qui constituent les fichiers Catalogue et Attributs subissent des accès très fréquents, et pour cette raison, ont plus de chances de se retrouver en cache quelque part. Tout cela concourt à donner au stockage complet d'un fichier, y compris ses méta-données et ses données au sein des fichiers catalogue et Attributs structurés en B-Tree, un gain de performances d'ensemble. Même une charge de 8 octets qui se gonfle à 25 octets n'a pas d'importance, tant qu'elle peut tenir dans un nœud B-tree dans le fichier d'attributs, que de toute façon, le système d'exploitation doit lire entièrement.

Il y a d'autres contributions significatives de Léopard des neiges à la réduction de l'empreinte sur le disque (notamment la suppression des localisations superflues et

les fichiers nib adaptables), mais la compression HFS + est de loin la technique la plus intéressante.

## L'intelligence de l'installeur

Apple promet deux choses intéressantes à propos du processus d'installation :

> Léopard des neiges teste vos applications pour s'assurer qu'elles sont compatibles, et met à part tout programme connu pour être incompatible. Au cas où une rupture de courant interromprait votre installation, il repartira sans perdre aucune donnée.

La mise à l'écart des applications "connues comme incompatibles" est indubitablement une réponse aux problèmes d'écran bleu rencontrés lors du passage de Tiger à Léopard il y a deux ans qui était provoquée par la présence d'extensions système tierces incompatible -certains diraient "illicites"-. J'ai une vision catégoriquement pragmatique de ce genre de logiciels, et suis heureux de constater qu'Apple adopte une approche pratique semblable pour minimiser ses conséquences sur les utilisateurs.

Bien sûr, on ne peut pas s'attendre à ce qu'Apple détecte et invalide tout logiciel potentiellement incompatible. Je soupçonne que seulement les logiciels les plus populaires ou ceux à plus hauts risques sont identifiés. Si vous êtes un développeur, cette caractéristique de l'installeur peut être une bonne façon de savoir si vous êtes sur la liste noire d'Apple.

Quant à la poursuite de l'installation après une panne de courant, je n'ai pas eu le toupet de tester cette possibilité (et j'ai un onduleur). Pour des opérations qui durent longtemps comme une installation, ce genre de sécurité est bienvenu, notamment pour les ordinateurs avec une batterie comme les portables.

Je mentionne ces deux détails du processus d'installation, principalement parce qu'ils illustrent le genre de choses qui est possible quand chez Apple, on donne le temps aux développeurs de polir leurs composants respectifs du système d'exploitation. Vous pourriez penser que l'équipe d'installation a été pressurée pour venir à bout de suffisamment de choses pendant un cycle de développement d'environ deux ans. Visiblement, ce n'est pas le cas, et ce sont les consommateurs qui en moissonnent les bénéfices.

# Le nouvel aspect de Léopard des neiges

J'ai depuis longtemps regretté qu'Apple ne se démarque pas nettement, au moins visuellement, du passé Aqua de Mac OS X. Hélas, je vais devoir attendre un peu plus longtemps, parce que Léopard des neiges n'introduit aucune révolution de ce genre. Et encore, je suis là dans une section familière qui semble démontrer le contraire. La vérité est que Léopard des neiges change effectivement l'apparence de presque chaque pixel sur votre écran, mais pas de la façon dont vous pourriez l'imaginer.

Depuis l'origine de la couleur sur le Macintosh, le système d'exploitation a utilisé par défaut une valeur de correction de gamma de 1,8. Entre temps, Windows - autrement dit le reste du monde- a utilisé une valeur de 2,2. Bien que cela puisse paraître sans importance à tout le monde, sauf les graphistes professionnels, la différence est d'ordinaire apparente, même pour un observateur occasionnel, quand on voit la même image sur des écrans côte à côte, avec les deux méthodes d'affichage.

Bien que les adeptes du Mac préféreront instinctivement l'image avec un gamma de 1,8 à laquelle ils sont habitués, Apple a décidé que cette différence historique représentait plus de tracas que cela ne vaut. La correction de gamma par défaut sous Léopard des neiges est maintenant 2,2. Point !

S'ils arrivent à le voir, les utilisateurs vont probablement ressentir ce changement comme la sensation que l'interface de Léopard des neiges est un peu plus contrastée que celle de Léopard. Cela est renforcé par le nouveau fond d'écran du bureau par défaut, une version re-dessinée, et plus saturée que le fond d'écran par défaut de Léopard. (Notez que ces deux images entièrement différentes ne sont pas une tentative pour démontrer les effets des corrections de gamma différentes.)

*Léopard (à gauche) et Léopard des neiges (à droite)*


*L'effet d'éclairage du Dock*

Mais même au delà de la correction de couleur, à laquelle on peut s'attendre, Apple n'a pas pu résister à l'envie de rajouter quelques ajustements graphiques à l'interface de Léopard des neiges. Les modifications les plus apparentes sont associées au Dock. D'abord, ce nouvel aspect éclairé pris par l'icône dune application dans le dock quand on click dessus et maintien la pression. (Ceci lance Exposé, mais seulement pour les fenêtres qui appartiennent à l'application sur laquelle on a cliqué ; plus de détails plus tard).

Qui plus est, tous les menus Pop up du Dock sans exception, (et seulement du Dock), ont un aspect particulier dans Léopard des neiges avec un rendu spécifique de la sélection (qui, pour un changement, s'assortit assez bien au rendu des sélections dans l'ensemble du système).


*Le nouvel aspect des menus du Dock... autoritaire.*

Pour les utilisateurs de Macs d'un certain âge, ces menus rappellent le thème d'apparence Hi-Tech du mauvais épisode de Copland. Ils sont en réalité beaucoup plus subtiles, cependant. Notez les bordures translucides, qui accentuent les coins arrondis. Le gradient sur la sélection est aussi remarquablement maîtrisé.

Néanmoins, ceci est un rendu complètement nouveau pour une seule application (communément utilisée, il est vrai), et il n'y a pas un bit qui détonne avec l'apparence d'étagère inclinée et réfléchissante du Dock par défaut. Mais j'ai déjà dit ailleurs ce que j'en pensais, et bien plus. Si le serment concernant l'apparence de Léopard des neiges était "d'abord, pas de dégâts", je serais alors enclin à décerner une mention passable -enfin, presque.

Si j'avais à caractériser ce qui ne va pas avec les additions visuelles de Léopard des neiges en deux mots seulement, ce serait ceci : tout s'affadit. Dans Léopard des neiges, Apple a saupoudré de la même façon chaque application, de la poussière magique de Core Animation. Si une partie ou une autre de l'interface apparaît, disparaît ou change d'une façon significative, c'est accompagné par une animation, et un ou plusieurs changements d'intensité.

A dose modérée, ce genre d'effets est bien. Mais en plusieurs circonstances, Léopard des neiges dépasse les bornes. Ou du moins, il dépasse *mes* bornes, qui se trouvent, il faut le noter, très loin à l'intérieur du pays Candy. D'autres, avec une moindre tolérance pour les animations, et qui sont déjà irrités par les colifichets de Léopard et des précédentes versions ne vont pas trouver grand chose à aimer dans les modifications visuelles de Léopard des neiges.

Celle qui m'a littéralement poussé à bout, c'est la petite danse futile du nom de fichier qui intervient dans le Finder (oh surprise !) quand on renomme un fichier sur le bureau. Il y a là tant de variations d'intensité, de modifications de couleurs, et de déplacements de texte qui interviennent si rapidement, et sont concentrés sur une si petite surface, que cela me fait hurler. Et que j'attende ou non que ces animations soient finies pour pouvoir utiliser ma machine, j'en ai en tout cas le sentiment désagréable.

Cependant, je peux sans enthousiasme prédire que la plupart des gens normaux (c'est à dire ceux qui ne vont pas lire tout cet article), ou bien trouveront réjouissantes ces nouvelles touches visuelles, ou bien (plus vraisemblablement), ne les remarqueront même pas.

## Affaire de marque

Animation mise à part, l'uniformité visuelle de Léopard des neiges présente à Apple un certain défi pour le marketing. Même au delà de problème évident qu'il y a à promouvoir auprès des consommateurs une mise à jour de système d'exploitation avec "aucune nouvelle possibilité", il y a la difficulté d'obtenir que les gens remarquent au moins que ce nouveau produit existe.

Dans la période qui a précédé la livraison de Léopard des neiges, Apple s'en est tenu à une version modifiée du thème de l'espace utilisé dans Léopard. Elle était dans les diapos de la keynote, sur les banderoles de la WWDC, sur les DVDs de la livraison aux développeurs, et partout sur le site d'Apple à la section consacrée à Mac OS X. L'image d'entête de la page web d'Apple sur Mac OS X jusqu'à une semaine avant la sortie de Léopard des neiges apparaît ci-dessous. Elle est plutôt froide et sèche : l'espace, les étoiles, de riches nébuleuses violettes, un reflet de lentille.


*Léopard des neiges, la dernière frontière*

Puis est venu le golden master de Léopard des neiges, qui dans un changement agréable par rapport aux précédentes livraisons, a été distribué aux développeurs quelques semaines avant que Léopard des neiges soit livré. Le disque d'installation présente un aspect entièrement différent, qui a finalement été retenu pour le conditionnement de détail. Pour apprécier le changement, alignons les disques plutôt que le conditionnement (qui s'est rapidement rétréci au point de n'inclure que

le disque, de toute façon). Voici, de gauche à droite, et du haut vers le bas, les disques de Mac OS 10.0 à 10.6 (les disques 10.0 et 10.1 d'aspect identique ont été réunis).


Eh oui, c'est un léopard des neiges. Avec de la neige dessus. Un peu trop sur le nez, à mon goût, mais pas sans un certain charme. Et il y a une chose importante qui va avec : il est immédiatement reconnaissable comme quelque chose de nouveau, et de différent. On ne peut pas le rater ; c'est comme cela que je peux résumer le conditionnement. Huit ans d'un X géant, centré, embelli de façons variées, et boum, un chat. Il y a peu de chances que quelqu'un qui a vu Léopard sur l'étagère d'un distributeur Apple pendant deux ans ne s'aperçoive pas qu'il s'agit là d'un nouveau produit.

Si vous voulez avoir votre propre image de Snowy, le léopard des neiges (c'est moi qui l'ai nommé ainsi), Apple a eu la gentillesse de l'inclure dans les les images de fond d'écran avec le système. Les utilisateurs impénitents de Windows peuvent le télécharger directement.

# Détails internes

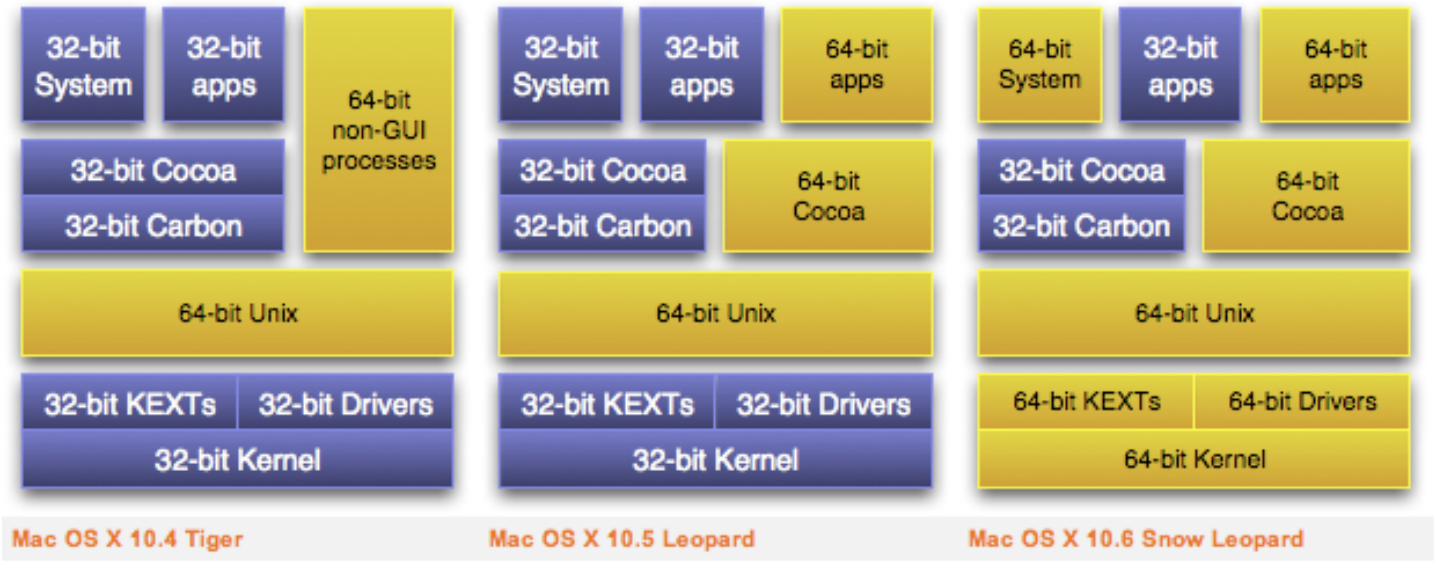
## Attention, détails techniques en vue...

Nous voilà maintenant au début de la section habituelle sur les détails techniques internes. Léopard des neiges est plein de changements internes, et ça se reflète dans le contenu de ce compte-rendu. Si vous êtes seulement intéressé(e) par les modifications que l'utilisateur peut voir, vous pouvez passer plus loin, mais vous allez manquer la viande du repas, et le cœur du nouvel OS d'Apple.

## 64 bits : la route conduit toujours plus loin

Mac OS X a commencé sa conversion à 64 bits en 2003 avec la livraison de Panther, qui comprenait le support minimal le plus élémentaire du nouveau processeur Power PC G5 64 bits. En 2005, Tiger a apporté la possibilité de créer de vrais processus 64 bits - tant qu'ils ne se liaient pas aux bibliothèques GUI (celles de l'interface Utilisateur Graphique). Finalement, Léopard en 2007 a incorporé le support 64 bits pour les applications GUI. Mais il y avait encore des limitations : le support 64 bits concernait les applications Cocoa seulement. Dans les faits, c'était la fin de la route pour Carbon.

En dépit des promesses apparemment impressionnantes de Léopard, il faut encore quelques étapes supplémentaires pour que Mac OS X puisse atteindre le nirvana 64 bits. Le diagramme ci-dessous le montre :



Comme nous le verrons, tout ce qui est jaune dans le diagramme de Léopard des neiges représente les possibilités, pas nécessairement le mode de fonctionnement par défaut.

## K64

Léopard des neiges est la première version de Mac OS X à délivrer un noyau 64 bits ("K64" dans la terminologie Apple), mais il n'est pas validé par défaut sur la plupart des systèmes. La raison en est simple. Rappelez-vous qu'il n'y a pas de mode mixte sous Mac OS X. A l'exécution, un processus est soit 32 bits, soit 64 bits, et ne peut charger d'autre code -bibliothèques, greffons- que du même type.

Une catégorie importante de greffons chargés par le noyau est constituée pas les pilotes de périphériques. Si Léopard des neiges avait par défaut un noyau 64 bits, seuls les pilotes 64 bits pourraient se charger. Et quand on se représente que Léopard des neiges est la première version de Mac OS X à inclure un noyau 64 bits, il y aurait eu très peu de ces pilotes précieux sur les systèmes des utilisateurs le jour du lancement.

| Product | Model name | K64 status |
|---|---|---|
| Early 2008 Mac Pro | MacPro3,1 | Capable |
| Early 2008 Xserve | Xserve2,1 | Default |
| MacBook Pro 15"/17" | MacBookPro4,1 | Capable |
| iMac | iMac8,1 | Capable |
| UniBody MacBook Pro 15" | MacBookPro5,1 | Capable |
| UniBody MacBook Pro 17" | MacBookPro5,2 | Capable |
| Mac Pro | MacPro4,1 | Capable |
| iMac | iMac9,1 | Capable |
| Early 2009 Xserve | Xserve3,1 | Default |

Si bien que, par défaut, Léopard des neiges ne démarre avec un noyau 64 bits que sur les XServe de 2008 ou plus récents. Je suppose que le raisonnement est que tous les périphériques couramment rattachés à un Xserve seront fournis par Apple avec des pilotes 64 bits pour Léopard des neiges.

D'une façon peut-être surprenante, tous les Macs avec des processeurs 64 bits ne sont même pas aptes à démarrer dans un noyau 64 bits. Bien que cela puisse

changer dans les prochaines versions intermédiaires de Léopard des neiges, le tableau ci-contre donne une liste de tous les Macs qui sont capables de démarrer K64, ou démarrent ainsi par défaut. (Pour trouver le nom de modèle de votre Mac, choisissez "A propos de ce mac" dans le menu Apple, puis cliquez "Plus d'infos...", et lisez l'"Identifiant du modèle" à la deuxième ligne de la rubrique Matériel).

Pour tous les Macs capables K64, démarrez en maintenant simultanément les touches 6 et 4, pour sélectionner le noyau 64 bits. Pour une solution plus permanente, utilisez la commande nvram et ajoutez arch=x86_64 à la chaîne des boot-args, ou bien éditez le fichier /Library/Preferences/SystemConfiguration/com.apple.Boot.plist, et ajoutez arch=x86_64 à la chaîne des drapeaux de noyau :

```
...
        <key>Kernel</key>
        <string>mach_kernel</string>
        <key>Kernel Flags</key>
        <string>arch=x86_64</string>
...
```

Pour revenir au noyau 32 bits, maintenez les touches 3 et 2 pendant le démarrage, ou utilisez une des techniques ci-dessus pour remplacer "x86_64" par "i386".

Nous avons déjà dit pourquoi, au moins initialement, vous n'aurez sans doute pas envie de démarrer en K64. Mais à mesure que l'adoption de Léopard des neiges va monter, et que des mises à jour à 64 bits des extensions existantes du noyau vont apparaître, pourquoi pourriez-vous avoir envie d'utiliser effectivement le noyau 64 bits ?

La première raison concerne la RAM, et pas de la façon dont vous vous pourriez l'imaginer. Bien que Léopard utilise un noyau 32 bits, les Macs qui utilisent Léopard peuvent contenir et utiliser beaucoup plus de RAM que la limite de 4 Go que le qualificatif de 32 bits semble impliquer. Mais, à mesure que les tailles de RAM augmentent, il y a une autre raison : le manque d'espace d'adressage, non pour les applications, mais pour le noyau lui-même.

En tant que processus 32 bits, le noyau est limité à un espace d'adressage de 32 bits (soit 4 Go). Cela ne semble pas être un problème ; après tout, le noyau a-t-il réellement besoin de plus de 4 Go pour faire son travail ? Mais rappelez-vous qu'une partie du travail du noyau est de chercher et de gérer la mémoire du système. Il utilise une structure de 64 octets pour contrôler l'état de chaque page de 4 Ko de Ram utilisée par le système.

C'est 64 *octets*, pas des kilo-octets. Cela n'est pas beaucoup. Mais considérez maintenant un Mac qui, dans un futur pas très lointain, contiendra 96 Go de RAM. (Si cela vous semble ridicule, imaginez à quel point ridicules les 8 Go du Mac sur lequel je tape ce texte maintenant auraient paru il y a cinq ans). Contrôler 96 Go de RAM nécessite 1,5 Go d'espace d'adressage pour le noyau. Utiliser plus d'un tiers de l'espace d'adressage du noyau juste pour le contrôle de la mémoire est une situation plutôt inconfortable.

A l'inverse, un noyau 64 bits a un espace d'adressage à peu près illimité (16 exaOctets). K64 est une nécessité inévitable, étant donnée l'augmentation rapide de la taille de la mémoire système. Bien que vous n'en ayez pas nécessairement besoin maintenant sur votre ordinateur de bureau, c'est déjà courant de voir installé sur les serveurs un espace de RAM à deux chiffres.

Une autre chose que K64 a en sa faveur, est la vitesse. L'architecture de l'ensemble d'instructions du x86, a eu une histoire un peu tortueuse. Quand il a conçu x86-64, l'extension à 64 bits du x86, AMD a saisi l'occasion pour laisser de côté un peu des laideurs du passé, et incorporer plus de possibilités modernes : des registres plus nombreux, des nouveaux modes d'adressage, des possibilités de virgule flottante ne reposant pas sur une pile, etc... K64 profite de ces bénéfices. Apple déclare les performances suivantes :
• 250 % plus rapide pour les points d'entrée des appels système
• 70 % plus rapide pour la copie en mémoire (utilisateur/noyau)

Des tests ciblés confirmeraient cela, j'en suis sûr. Mais dans la pratique quotidienne, vous avez peu de chances de pouvoir attribuer au noyau un quelconque gain de performance. Pensez plutôt à K64 comme moyen de supprimer les goulots d'étranglement dans un petit nombre d'applications (généralement basées sur des serveurs), qui exploitent le noyau effectivement et intensément.

Si vous vous sentez mieux de savoir que votre noyau est plus efficace, et que si vous aviez 96 Go de RAM installés, le noyau ne manquerait pas d'espace d'adressage, et si vous n'avez aucun pilote 32 bits indispensable, alors, en tout état de cause, démarrez en mode 64 bits.

Pour tous les autres, je pense qu'il faut se réjouir que K64 soit prêt, et attende qu'on en ait réellement besoin, et de grâce, faites savoir à tous les vendeurs qui écrivent des extensions du noyau que vous comptez installer le support 64 bits aussi vite que possible.

Finalement, il est bon de le répéter : gardez à l'esprit que vous n'avez pas besoin d'un noyau 64 bits pour faire tourner des applications 64 bits ou pour installer plus

de 4 Go de RAM dans votre Mac. Les applications tournent très bien en mode 64 bits sur un noyau 32 bits, et même dans des versions précédentes de Mac OS X, on peut tirer parti de beaucoup plus que 4 Go de RAM.

## Les applications 64 bits

Bien que Léopard ait apporté le support pour des applications GUI 64 bits, il n'en incorporait réellement que très peu. En fait, selon mes comptes, seules deux applications GUI ont été livrées avec Léopard : Xcode (dont l'installation est optionnelle), et Chess. Et bien que Léopard ait rendu possible pour les développeurs extérieurs à Apple la production d'applications graphiques (GUI) 64 bits, très peu en ont profité, parfois pour des circonstances malheureuses, mais le plus souvent parce qu'il n'y avait pas de bonnes raisons de le faire, et d'abandonner dans la course les utilisateurs de Mac OS X 10.4 ou de versions plus anciennes.

Apple pousse maintenant la transition vers 64 bits plus vigoureusement. Et commence en montrant l'exemple. Léopard des neiges est fourni avec 4 applications GUI pour l'utilisateur final qui *ne sont pas* 64 bits : iTunes, Grapher, Front Row, et DVD Player. *Toutes les autres* sont en 64 bits. Le Finder, le Dock, Mail, TextEdit, Safari, iChat, Address Book, Dashboard, Help Viewer, Installer, Terminal, Calculator, quoi que vous en pensiez, c'est du 64 bits.

La seconde grosse carotte, (ou bâton, selon la manière dont vous voyez les choses), est le manque de support 32 bits pour les nouvelles APIs et technologies. Léopard a lancé la tendance délaissant les APIs démodées, en ne portant que les nouvelles sous 64 bits. L'exécutif amélioré Objective C 2.0 introduit avec Léopard était aussi en 64 bits seulement.

Léopard des neiges continue dans la foulée. Dans l'exécutif Objective-C 2.1, les variables d'instance non-fragile, le modèle unifié d'exceptions avec C++, et la répartition vtable rapide ne restent disponibles que pour les applications 64 bits. Mais la nouvelle API uniquement 64 bits la plus significative est QuickTime X - assez significative pour être traitée séparément, alors, restez branché.

## 64 bits ou la faillite

Tout cela fait partie de la façon pas très subtile qu'a Apple de dire aux développeurs qu'il est temps de passer maintenant au 64 bits, et que ce doit être le mode par défaut pour toutes les applications, que le développeur estime que c'est un besoin ou non. Dans la plupart des cas, ces nouvelles APIs n'ont pas de rapport intrinsèque

avec le 64 bits. Apple a simplement choisi de les utiliser comme une forme supplémentaire de persuasion.

En dépit de tout ce qui précède, je qualifierais encore Léopard des neiges comme l'avant-dernière étape dans la course au 64 bits, du début à la fin. J'espère vraiment que Mac OS X 10.7 démarrera par défaut avec le noyau 64 bits, qu'il sera livré avec des versions 64 bits de toutes les applications, greffons (plug-ins), et extensions du noyau, et laissera encore plus d'APIs dépassées et démodées se dissoudre dans l'espace 32 bits.

# QuickTime X

Apple a fait quelque chose d'un peu bizarre, dans Léopard, quand il a négligé de porter l'API QuickTime basée sur C en 64 bits. A l'époque, cela ne semblait pas être un enjeu important. La transition de Mac OS X vers le 64 bits s'était déjà étalée sur plusieurs années et plusieurs versions majeures. On pouvait penser que ce n'était pas encore l'heure pour QuickTime de passer en 64 bits.

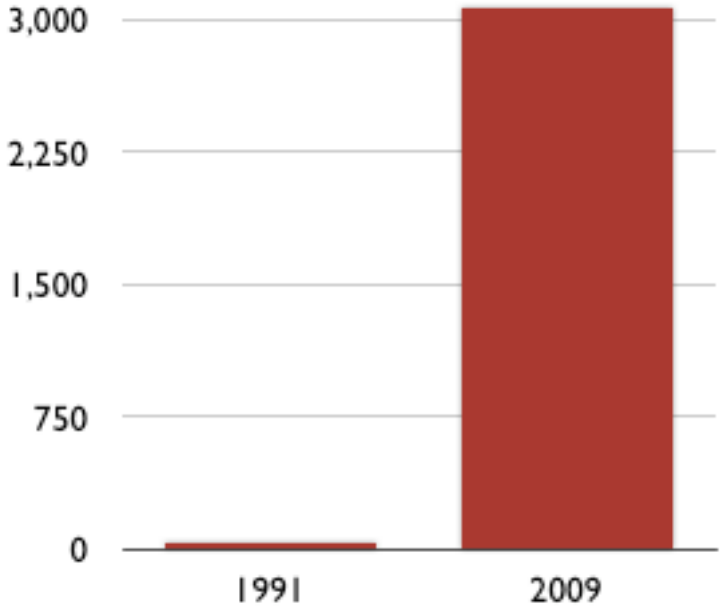
Tel que cela se présente, mon évaluation de la situation, abrupte, mais pessimiste à l'époque était juste : QuickTime a reçu le "traitement Carbon". Comme Carbon, la vénérable API QuickTime, que nous connaissons et aimons bien, ne sera pas -jamais- de la transition vers le 64 bits.

Pour être clair, QuickTime, la *technologie* et QuickTime, la *marque* vont à coup sûr passer en 64 bits. Ce qu'on abandonne avec la version 32 bits, c'est l'API basée sur C, introduite en 1991, et utilisée pendant 18 ans. Son remplaçant dans le monde 64 bits a été opportunément appelé QuickTime X.

Le X de QuickTime X, comme celui de de Mac OS X se prononce "dix". Ce n'est pas le premier de ces étranges parallèles. Comme Mac OS X avant lui, QuickTime X :
• entend se départir clairement de son prédécesseur
• est basé sur une technologie développées sur une autre palteforme
• inclut une compatibilité transparente avec son incarnation antérieure

• promet de meilleurs performances et une architecture plus moderne
• manque des possibilités importantes qu'avait la précédente version.

Prenons les choses l'une après l'autre. D'abord, pourquoi une rupture nette était-elle nécessaire ? Pour le dire simplement, QuickTime est vieux, réellement vieux. L'image vidéo épouvantablement pointilliste, de la taille d'un timbre-poste de la version initiale en 1991 était considérée comme un tour de force technologique.




*La vitesse CPU maximum disponible (en MHz)*

A cette époque, le Macintosh le plus rapide qu'on pouvait acheter, avait un processeur à 25 MHz. Le graphique ridicule à droite enfonce le clou. Une conception prévoyante ne peut vous mener que jusqu'à un certain point. L'état du monde dans lequel une technologie est née dicte inévitablement son destin. Et cela est particulièrement vrai pour des APIs qui ont vécu longtemps comme QuickTime, avec une forte inclinaison pour une compatibilité rétrograde.

Comme première implantation réussie de la vidéo sur un ordinateur personnel, il est vraiment étonnant que l'API QuickTime ait vécu aussi longtemps. Mais le monde a bougé. Tout comme Mac OS se retrouva englué dans le ghetto du multitâche coopératif et la mémoire non protégée, QuickTime s'est traîné jusqu'en 2009 avec des notions dépassées de concurrence et un sous-système entravé par sa conception.

Quand le temps fut venu d'écrire le code de gestion de la vidéo pour l'iPhone, la dernière version de QuickTime, QuickTime 7 n'était tout simplement pas à la hauteur. Il s'était inefficacement enflé pendant son existence sur les ordinateurs de bureau, il manquait d'un bon support pour un rendu Vidéo accéléré par GPU, indispensable pour utiliser des codecs vidéo modernes sur un portable (même avec un CPU 16 fois plus rapide que celui des Macs disponibles quand QuickTime 1.0 a été livré). Si bien qu'Apple a créé un moteur de rendu vidéo solide, moderne, à même d'utiliser les GPU, qui puisse s'accommoder confortablement des contraintes de mémoire et de CPU de l'iPhone.

Hum... Une API vidéo pour PC vieillissante, qui avait besoin d'être remplacée. Une toute nouvelle bibliothèque vidéo avec de bonnes performances, même sur un matériel (comparativement) anémique. Apple a fait la relation. Mais le tour est toujours en train d'être joué. Heureusement, c'est le fort d'Apple. QuickTime avait déjà vécu lui-même sur trois architectures CPU différentes, et trois systèmes d'exploitation complètement différents.

Le passage à 64 bits est encore un autre point d'inflexion (quoi que moins dramatique), et c'est ce qu'Apple a choisi pour marquer la différence entre le vieux QuickTime 7 et le nouveau QuickTime X. Pour cela, dans Léopard des neiges, Apple a limité toute utilisation de QuickTime par des applications 64 bits au framework Objective C QTKit.

## Le nouvel ordre du monde de QTKit

QTKit n'est pas nouveau ; il a commencé en 2005 comme une interface plus intuitive à QuickTime 7 pour les applications Cocoa. Ce niveau supplémentaire d'abstraction est la clé de la transition vers QuickTime X. QTKit cache maintenant derrière les murs de l'architecture orientée objet aussi bien QuickTime 7 que QuickTime X. Les applications utilisent le QTKit comme auparavant, et derrière le rideau, QTKit choisit d'utiliser soit QuickTime 7, soit QuickTime X pour réaliser la requête.

Si QuickTime X est si supérieur, pourquoi QTKit ne l'utilise-t-il pas pour tout ? La réponse est que QuickTime X, comme son pendant Mac OS X, a des possibilités très limitées dans sa version initiale. QuickTime X sait lire, capturer, et exporter, mais il manque d'un moyen d'édition vidéo à usage général. Il reconnaît les formats vidéo "modernes" (tout ce qu'on peut jouer sur un iPod, un iPhone, ou l'Apple TV), mais pour les autres codecs vidéo, hé bien, vous pouvez les oublier, ainsi que les greffons, parce que QuickTime X ne les connaît pas non plus.

Pour chacun des cas où QuickTime X n'est pas capable de faire le travail, QuickTime 7 va le faire. Couper, copier, coller des portions d'une vidéo ? QuickTime 7. Extraire une piste d'un film ? QuickTime 7. Jouer un film non reconnu par les mobiles existants d'Apple ? QuickTime 7. Enrichir le jeu de codecs de QuickTime au moyen d'un greffon de quelque sorte que ce soit ? QuickTime 7.

Mais, attendez un peu. Si QTKit est la seule façon pour une application 64 bits d'utiliser QuickTime, si QTKit choisit entre QuickTime 7 et QuickTime X derrière le rideau, que QuickTime 7 est seulement en 32 bits, et que Mac OS X ne supporte pas les processus en mode mixte capables d'exécuter à la fois du code 32 bits et du code 64 bits, alors, comment diable un processus 64 bits peut-il faire quelque chose qui requiert le concourt de QuickTime 7 ?

Pour comprendre, lancez la nouvelle application 64 bits QuickTime Player (que je vais aborder séparément plus tard), et ouvrez un fichier qui a besoin de QuickTime 7. Disons par exemple, un qui utilise le codec vidéo Sorensen. (Vous vous rappelez ? Le bon temps). A coup sûr, ça marche. Mais cherchez "QuickTime" dans le moniteur d'activité, et vous allez voir ceci :


*Plutôt furtif, le processus 32 bits QTKit*

Voilà donc la réponse. Quand une application 64 bits qui utilise QTKit a besoin du service de rendu en 32 bits de QuickTime 7, QTKit fait éclore un processus QTKit server 32 bits séparé pour faire le travail, puis communique le résultat au processus 64 bit qui en est à l'origine. Si vous laissez ouvert Activity Monitor en utilisant l'application QuickTime Player, vous pouvez observer les processus QTKit server aller et venir selon les besoins. Tout ceci est géré de façon transparente par le framework QTKit ; l'application elle même n'a pas besoin d'être au courant de ces opérations.
Oui, cela va être long, très long, avant que QuickTime 7 ne disparaisse complètement de Mac OS X (au moins, Apple a été assez élégant pour ne pas l'appeler QuickTime Classic), mais la voie est claire. Avec chaque nouvelle livraison de Mac OS X, attendez-vous à ce que les possibilités de QuickTime X

s'étendent, et que le nombre de choses qui requièrent QuickTime 7 diminue. Dans Mac OS X 10.7, par exemple, j'imagine que QuickTime X va recevoir le support des greffons (plug-ins). Et sans doute, pour Mac OS X 10.8, QuickTime X pourra bénéficier d'un support d'édition complet. Tout cela va se dérouler derrière la façade unificatrice de QTKit, jusqu'à ce que les services de QuickTime 7 ne soient plus du tout nécessaires.

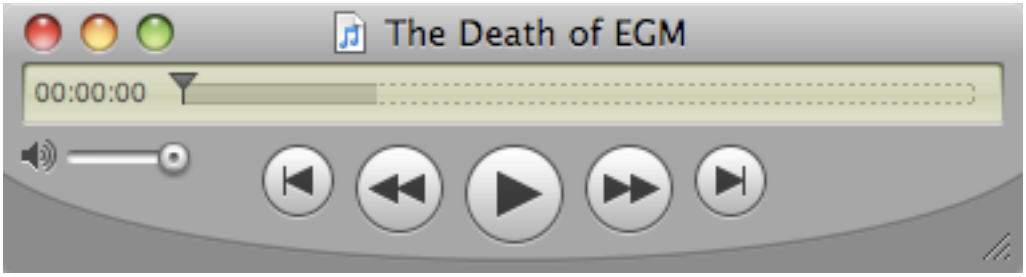
## Dites ce que vous voulez faire

En même temps, d'une façon surprenante peut être, beaucoup des limitations actuelles de QuickTime X soulignent en fait ses avantages exclusifs, et renseignent sur l'évolution de l'API QTKit. Bien qu'il n'y ait aucun moyen direct pour un développeur de demander que QTKit utilise le rendu QuickTime X, il y a plusieurs moyen indirects d'influencer la décision. La clé est l'API QTKit, qui repose intensément sur le concept d'*intention*.

Les versions 1 à 7 de QuickTime utilisent une seule représentation interne pour toutes les ressources média : un objet Movie. Cette représentation inclut une information sur les pistes individuelles qui constituent le film, les tables d'échantillonnage pour chaque piste, et ainsi de suite - tout ce dont QuickTime a besoin pour comprendre et manipuler le média.

Ça paraît bien, jusqu'à ce que vous réalisiez que pour faire quoi que ce soit avec une ressource média, QuickTime nécessite la construction complète de cet objet Movie. Par exemple, considérons la lecture d'un fichier MP3 par QuickTime. QuickTime doit créer sa représentation interne de l'objet Movie du fichier MP3 avant de pouvoir le restituer. Malheureusement, le format MP3 contient rarement une information complète sur la structure des données audio. C'est seulement un flux de paquets. QuickTime doit laborieusement passer en revue et analyser le flux audio dans sa totalité pour créer l'objet Movie.

QuickTime 7 et les versions antérieures rendent le processus moins pénible en procédant à la lecture et à l'analyse en tâche de fond, de façon incrémentale. Vous pouvez constater cela dans de nombreuses application qui s'appuient sur QuickTime Player, sous la forme d'une barre de progression superposée au contrôleur de visualisation. L'image ci-dessous montre le chargement d'un podcast MP3 de 63 Mo dans la version Léopard de QuickTime Player. La portion grisée de la barre remplit lentement la zone pointillée de gauche à droite.




*QuickTime 7 en fait plus que nécessaire.*

Bien que la lecture puisse commencer presque immédiatement, (à condition de commencer au début), cela vaut le coup de revenir un peu en arrière, et de considérer ce qui se passe. QuickTime est en train de créer un objet Movie adapté à n'importe quelle opération que QuickTime peut entreprendre ; l'extraction ou l'addition de pistes, l'exportation, et tout ce que vous voulez. Mais si vous voulez seulement jouer le fichier ?
Le problème, c'est que l'API QuickTime 7 manque d'un moyen d'exprimer ce genre d'intention. Il n'y a aucun moyen de dire à QuickTime 7 "Ouvre seulement ce fichier aussi vite que possible pour que je puisse l'entendre ou le visualiser. Ne t'embarrasse pas à lire chaque octet, à l'interpréter et à en définir la structure, pour le cas où je voudrais en éditer ou exporter le contenu. S'il te plaît, ouvre le, seulement pour le relire".
L'API QTKit fournit exactement cette possibilité. En fait, la seule façon d'obtenir un rendu de QuickTime X est d'exprimer explicitement votre intention de ne pas faire ce que QuickTime X ne sait pas gérer. Qui plus est, toute tentative d'effectuer une opération dont vous n'avez pas préalablement exprimé l'intention provoquera une exception de la part de QuickTime X.
Le mécanisme d'intention est aussi la manière dont les nouvelles possibilités de QuickTime X sont rendues visibles, comme la capacité à charger de façon asynchrone des fichiers de films énormes ou distants (autrement dit sur lien internet lent), sans bloquer l'interface utilisateur qui utilise le fil (thread) principal de l'application.
De plus, il y a beaucoup de raisons de faire ce qu'il faut pour pouvoir monter à bord du train QuickTime X. Pour les formats de média qu'il supporte, QuickTime X est moins pénalisant pour le processeur pendant la lecture que QuickTime 7. (C'est indépendant du fait que QuickTime X ne gâche pas son temps à se préparer une représentation interne du film pour éditer ou exporter, quand on ne veut que le relire). QuickTime X dispose aussi de la possibilité de lecture accélérée par les GPU du format H 264, dans sa version de base, uniquement pour les Macs équipés s'un GPU NVIDIA 9400 M (c'est à dire quelques iMacs de 2009, et plusieurs modèles de Mac Books de 2008 et 2009). Et pour finir, QuickTime X inclut aussi un support Color Sync étendu pour la vidéo, qu'on attendait depuis longtemps.

### Le facteur X

C'est seulement le début d'un long périple pour QuickTime X, qui n'y est apparemment pas très bien adapté. Un moteur QuickTime sans possibilité d'édition ? Pas de greffons ? Cela semble ridicule de l'avoir proposé, tout simplement. Mais c'est la manière de faire d'Apple depuis quelques années : un progrès régulier, délibérément. Apple affectionne de ne pas proposer des possibilités avant que leur temps ne soit venu.

Aussi impatients que les développeurs puissent être pour un moteur 64 bits complet capable de succéder à QuickTime 7, Apple est lui-même assis au sommet d'une des plus importantes base de code de l'industrie, basée sur QuickTime (et rajoutée au départ à Carbon) : Final Cut Studio. De ce fait, il reste condamné au 32 bits. Dire qu'Apple est "hautement motivé" à étendre les possibilités de QuickTime X serait sous-estimer les choses.

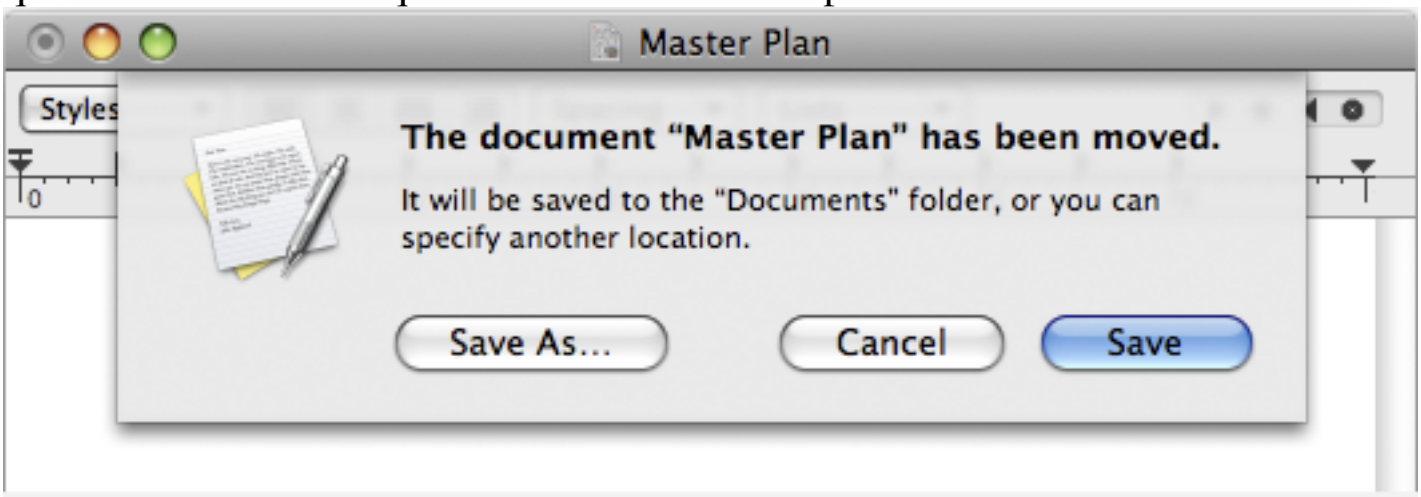
Néanmoins ne vous attendez pas à ce qu'Apple se rue de l'avant inconsidérément. Reproduire les fonctionnalités d'une API vieille de 18 ans, et développée de façon continue ne va pas se faire du jour au lendemain, et cela va demander encore plus longtemps pour que chaque application importante de Mac OS X soit mise à jour pour utiliser exclusivement QTKit. Les transitions, il vous faut les aimer que diable !

# Unification de l'API du système de fichiers

Mac OS X a dans le passé autorisé de nombreuses façons différentes pour se référer aux fichiers sur un disque à partir des applications. Les chemins classiques (du genre /Users/john/Documents/monFichier) sont supportés aux niveaux les plus élémentaires du système d'exploitation. Ils sont simples, prévisibles, mais ne constituent peut-être pas une idée géniale à utiliser comme seul moyen pour une application de suivre les fichiers. Voyez ce qui arrive, si une application ouvre un fichier basé sur son chemin, et que l'utilisateur déplace le fichier ailleurs pendant qu'il est encore en cours d'édition. Quand on demande à l'application de sauver le

fichier, si elle n'a que le chemin pour le faire, elle va finir par créer un nouveau fichier à l'ancien emplacement, ce qui n'est certainement pas ce que voulait l'utilisateur.
Mac OS Classic avait une représentation interne des fichiers plus élaborée, qui lui permettait de retrouver des fichiers indépendamment de leur position réelle sur le disque. Cela se faisait avec l'aide d'identificateurs uniques de fichiers (ids), supportés par HFS/HFS +. L'incarnation Mac OS X de ce concept est le type de donnée FSRef.
Finalement, dans nos temps modernes, les URLs sont devenus le mode de représentation de facto pour des fichiers qui peuvent être localisé quelque part ailleurs que sur l'ordinateur local. Les URLs peuvent aussi se référer à des fichiers locaux, mais dans ce cas, ils ont tous les mêmes désavantages que les chemins de fichiers.
La diversité des types de données se reflète dans les APIs du système de fichiers de Mac OS X. Certaines fonctions prennent les chemins comme arguments, d'autres utilisent des références opaques aux fichiers, et d'autres encore ne fonctionnent qu'avec des URLs. Les programmes qui utilisent ces APIs passent souvent une bonne partie de leur temps à convertir des références de fichiers d'une représentations à une autre.
La situation est similaire quand il s'agit de récupérer les informations au sujet des fichiers. Il y a une quantité énorme de fonctions de récupération des méta-données du système de fichiers, et pas une seule d'entre elles ne permet de tout faire. La récupération de toutes les informations disponibles sur un ficher nécessite le recours à plusieurs appels, et chacun d'eux peut attendre comme argument un type différent de référence au fichier.
Voici un exemple fourni par Apple à la WWDC. L'ouverture d'un seul fichier dans la version Léopard de l'application Aperçu provoque :
• 4 conversions de FSRef en chemin de fichier
• 10 conversions de chemin de ficher en FSRef
• 25 appels à getattrlist()
• 8 appels à stat()/lstat()
• 4 appels à open()/close()
Dans Léopard des neiges, Apple a créé un nouvel ensemble d'APIs de système de fichier, unifié et complet construit à partir d'un seul type de données : les URLs. Mais ce sont des "objets" URLs, c'est à dire les types opaques NSURL et CFURL avec des passerelles entre eux, qui ont été inspirés par tous les attributs désirables de FSRef.
Apple s'est appuyé sur ces types de données parce que leur nature opaque permet ce genre d'amélioration, et par ce qu'il y a un grand nombre d'APIs qui les utilisent. Les URLs sont aussi la meilleure assurance pour le futur parmi tous les choix possibles, avec la partie scheme qui fournit une flexibilité pratiquement illimitée

pour de nouveaux types de données et ne nouveaux mécanismes d'accès. Les APIs du nouveau système de fichiers construites autour de ces types d'URL opaques supportent les caches, et la recherche anticipée des méta-données pour améliorer encore les performances.
Il y a aussi une nouvelle représentation des fichiers sur le disque appelée les Bookmarks (à ne pas avec ceux des browsers) qui remplacent les alias de Mac OS Classic d'une manière plus adaptée au réseau. Les Bookmarks sont la façon la plus robuste pour créer une référence à un fichier à partir d'un autre fichier. Il est aussi possible de rattacher des méta-données arbitraires à chaque Bookmark. Par exemple, si une application veut garder une liste persistante de fichiers "favoris", plus quelque information spécifique à l'application, et qu'elle veut que ce soit résistant à tout mouvement des fichiers derrière son dos, les Bookmarks sont le meilleur outil pour le faire.
Je mentionne tout cela, non pas parce que je m'attends à ce que les APIs du système de fichier aient un gros intérêt pour des gens sans fascination particulière envers cette partie du système d'exploitation, mais parce que comme Core Text avant elles, c'est une indication de la réelle jeunesse de Mac OS X comme plateforme. Même après 7 révisions majeures, Mac OS X lutte encore pour se démarquer de l'ombre de ses trois ancêtres, NextSTP, Mac OS Classic, et BSD Unix. Ou peut-être, cela montre-t-il comment l'équipe Core OS d'Apple est amenée à remplacer sans aucun ménagement des APIs vieilles et encroûtées, par des versions plus modernes.
Il va s'écouler longtemps avant que les bénéfices de ces changements se diffusent (vers le bas ou vers le haut?) jusqu'aux utilisateurs sous la forme d'applications pour le Mac, écrites ou modifiées pour utiliser ces nouvelles APIs. La plupart des applications Mac bien écrites disposent déjà de ce comportement désirable. Par exemple, l'application TextEdit de Léopard est capable de détecter correctement quand un fichier sur lequel elle travaille a été déplacé.




*TextEdit, un bon citoyen Mac OS X*

Bien sûr, la condition essentielle est "bien écrites". La simplification des APIs du système de fichiers signifie que plus de développeurs seront désireux d'étendre leur

effort, -maintenant grandement réduit- pour proposer ces comportements favorables aux utilisateurs. L'amélioration en performance qui l'accompagne n'est que la cerise sur le gâteau, mais une raison de plus pour que les développeurs modifient leurs applications existantes et fonctionnelles pour utiliser ces nouvelles APIs.

# Faire plus avec plus

La loi de Moore est largement citée dans les milieux technologiques, et est aussi très communément mal comprise. Sa formulation la plus souvent utilisée est que "les ordinateurs doublent de vitesse tous les ans, ou presque", mais ce n'est pas du tout ce que Gordon Moore a écrit. Son article de 1965 dans la revue "Electronics" abordait de nombreux sujets sur l'industrie des semi-conducteurs, mais s'il fallait le résumer par une seule "loi", ce serait que, grosso-modo, le nombre de transistors qu'on peut loger dans un pouce carré de silicium double tous les 12 mois.

Plus tard, Moore a modifié cela à deux ans, mais la durée n'est pas le point où les gens se trompent. Le problème, c'est la confusion entre le doublement de la densité des transistors, et le doublement de la "vitesse de l'ordinateur". (Il est encore plus problématique de définir une "loi" sur la base d'un seul article de 1965, mais, bon, mettons ça de côté pour l'instant . Pour une discussion plus complète de la loi de Moore, lisez s'il vous plaît l'article classique de Jon Stokes).

Pendant des décennies, chaque augmentation de la densité des transistors fut en fait accompagnée d'une augmentation comparable de la vitesse de calcul, grâce à des vitesses d'horloge toujours plus élevées, et à l'apparition de l'exécution superscalaire. Cela a bien marché -le code s'exécutait plus rapidement sur chaque nouveau CPU- jusqu'à ce que la dure réalite de le densité de puissance mette fin à la fête.

La loi de Moore continue , du moins pour l'instant, mais notre capacité à permettre au code de tourner plus vite à chaque nouvel accroissement de la densité des transistors s'est considérablement ralentie. Le repas gratuit, c'est terminé. Les vitesses d'horloge des CPU qui ont stagné depuis des années sont en fait en train de diminuer le plus souvent. (Le dernier Mac Pro 2009 de haut de gamme contient un CPU à 2,93 GHz, alors que le modèle de 2008 pouvait être équipé d'un CPU à 3,2

GHz). L'ajout d'unités d'exécution à un CPU a aussi atteint un point d'inflexion depuis longtemps, étant données les limites du parallélisme des instructions dans le code des applications courantes.

Et nous avons encore toutes ces transitions qui pleuvent sur nous, de plus en plus chaque année. Le défi est de trouver de nouvelles façons de les utiliser efficacement pour rendre les ordinateurs plus rapides.

Au point où nous en sommes, la réponse de l'industrie des semi-conducteurs a été de nous donner plus que ce que nous avons déjà. Alors qu'autrefois, un CPU contenait une seule unité logique de calcul, maintenant, les CPUs, même dans les ordinateurs de bas de gamme, contiennent deux cœurs, et les modèles de haut de gamme affichent deux processeurs avec 8 cœurs logiques chacun. Je l'accorde, les cœurs eux-mêmes deviennent plus rapides en général en en faisant plus que leurs prédécesseurs pour une même vitesse d'horloge, mais cela n'intervient pas jusqu'à approcher le taux auquel les cœurs se multiplient.

Malheureusement, d'une façon générale, un CPU à deux cœurs ne va pas faire tourner votre application deux fois plus vite qu'un simple CPU. En fait, votre application n'ira sans doute pas plus vite du tout, à moins qu'elle n'ait été écrite pour tirer avantage de plus d'un seul cœur. Confrontés à une surabondance de transistors, les fabricants de processeurs se sont retournés, et fournissent plus de ressources de calcul que les programmeurs ne sont capables d'en utiliser, transférant ainsi l'essentiel de la responsabilité d'ordinateurs plus rapides aux gens chargés du logiciel.

## Nous sommes dans le système d'exploitation, et nous sommes là pour vous aider

C'est dans cet environnement que Léopard des neiges est né. S'il y a une responsabilité (la sécurité mise à part) que les vendeurs d'un système d'exploitation devraient ressentir, en cette année 2009, c'est de trouver un moyen pour les applications (et l'OS lui-même) d'utiliser la richesse toujours plus grande des ressources de calcul à leur disposition. Si j'avais à choisir un seul "thème" technologique pour Léopard des neiges, ce serait : aider les développeurs à utiliser tout ce silicium disponible; les aider à faire plus, avec plus.

A cette fin, Léopard des neiges inclut deux nouvelles APIs significatives, secondées par plusieurs améliorations de l'infrastructure, plus petites, mais aussi importantes. Nous allons commencer en bas, et croyez-le ou non, par le compilateur.

# LLVM et Clang

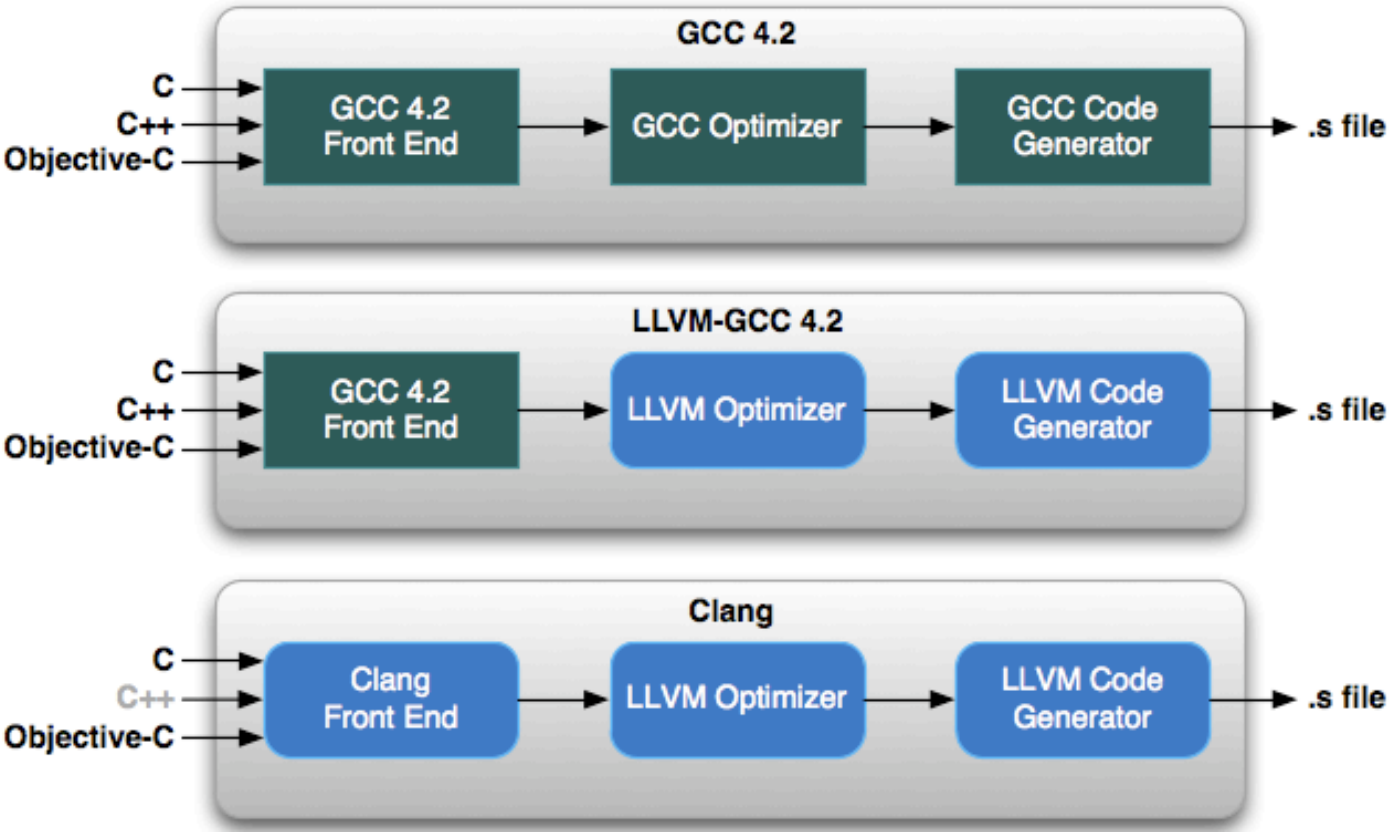
Apple a fait un investissement stratégique dans le projet open source LLVM il y a plusieurs années déjà. J'ai abordé les fondamentaux de LLVM dans mon compte-rendu de Léopard. (Si vous n'êtes pas au courant, mettez vous à la page avant de continuer). J'y ai montré comment Léopard utilisait LLVM pour fournir une implémentation logicielle compilée à la volée (JIT), des fonctions OpenCL en les rendant considérablement plus efficaces.

Ne soyez pas abusé(e) par son usage limité sous Léopard ; Apple a des plans ambitieux pour LLVM. Jusqu'où ? Pourquoi ne pas remplacer les boyaux du compilateur gcc que Mac OS X utilise actuellement, par ceux équivalents de LLVM ? Ce projet est bien lancé. Pas assez ambitieux ? Pourquoi ne pas s'affranchir entièrement de gcc et le remplacer par un nouveau compilateur, basé sur LLVM (mais compatible avec gcc) ? Ce projet s'appelle Clang et il déjà fourni quelques résultats de performance impressionnante.

Avec l'introduction de Léopard des neiges, c'est officiel : Clang et LLVM représentent la stratégie adoptée par Apple. LLVM a un nouveau logo stylé, un hommage appuyé à un manuel bien connu de conception des compilateurs.

*LLVM! Clang!*

Apple fournit maintenant pas moins de 4 compilateurs pour Mac OS X : GCC 4.0, GCC 4.2, LLVM-GCC 4.2 (GCC 4.2 en accès direct, et LLVM dans les coulisses), et Clang, pour améliorer la migration vers LLVM. En voici un croquis :




*Les compilateurs Mac OS X*

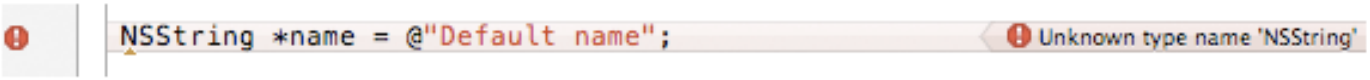
Tous ces compilateurs ont un binaire compatible sous Mac OS X, si bien que vous pouvez par exemple, construire une bibliothèque avec un compilateur, et la lier à un exécutable construit avec un autre compilateur. Ils sont aussi -en théorie au moins- tous compatibles pour les sources et les commandes. Clang ne supporte pas encore quelques caractéristiques ésotériques de GCC, et il ne supporte aussi que C, Objective C, un peu de C++ (Clang(age) ; vous avez compris ?), alors que GCC en supporte beaucoup plus. Apple se consacre au support complet de C++ pour Clang, et espère éliminer les incompatibilités qui subsistent avec GCC pendant la durée de vie de Léopard des neiges.

Clang apporte deux attributs importants que vous êtes en droit d'attendre d'un nouveau compilateur au goût du jour : des délais de compilation plus courts, et des exécutables plus rapides. Selon les tests d'Apple avec ses propres applications, comme iCal, Address Book ou même XCode, et des applications tierces comme Adium ou Growl, Clang compile à peu près trois fois plus vite que GCC 4.2. Quant à la vitesse du produit fini, LLVM en arrière plan, que ce soit dans Clang ou dans LLVM_GCC, produit des exécutables qui sont 5 à 25 % plus rapides que ceux créés par GCC 4.2.

Clang est aussi plus facile pour les développeurs que ses prédécesseurs GCC. J'admet que cela n'a pas grand chose à voir avec l'avantage d'utiliser des CPU à plusieurs cœurs, ou des choses de ce genre, mais il est sûr que c'est une des premières choses qu'un développeur remarque effectivement. Permettez-moi de m'en réjouir.

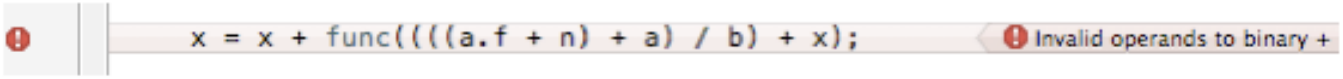
Pour les débutants, Clang est encapsulable, si bien que XCode peut utiliser la même infrastructure de compilateur pour les possibilités de l'IDE (recherche par symbole, complétion de code) que celle qu'il utilise pour compiler le code final exécutable. Clang crée et conserve aussi des méta-data beaucoup plus étendues pendant la compilation, ce qui permet un bien meilleur rapport d'erreur. Par exemple, quand GCC vous dit ceci :


Ce n'est pas très clair de savoir où est le problème, particulièrement si vous débutez en C. Bien sûr, vous les experts, vous savez déjà où il est (et notamment si vous avez déjà vu ces exemple à la WWDC) mais je crois que tout le monde sera d'accord pour admettre que ce message d'erreur généré par Clang et bien plus utile :

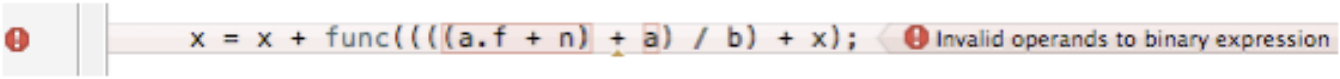



Peut-être qu'un novice ne saura pas encore quoi faire, mais au moins on sait clairement où est le problème. Imaginer *pourquoi* le compilateur ne connaît pas NSString est une tâche bien plus ciblée que ce que le message d'erreur obscur de GCC permet de déduire.

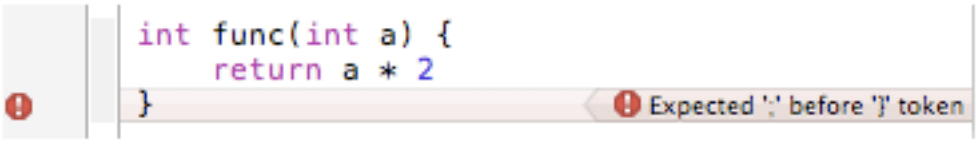
Même quand le message est clair, le contexte peut ne pas l'être. Voyez cette erreur de GCC :



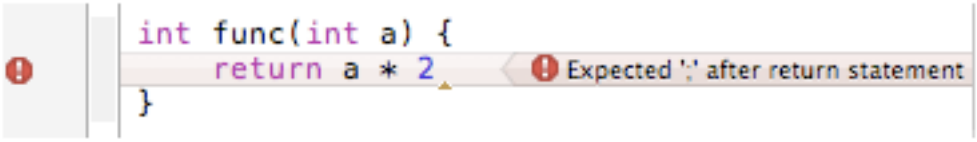

Il y a bien quatre opérateurs "+" sur cette ligne. Lequel a des opérandes qui posent problème ? Grâce à ses méta-data plus détaillées, Clang peut mieux le localiser :



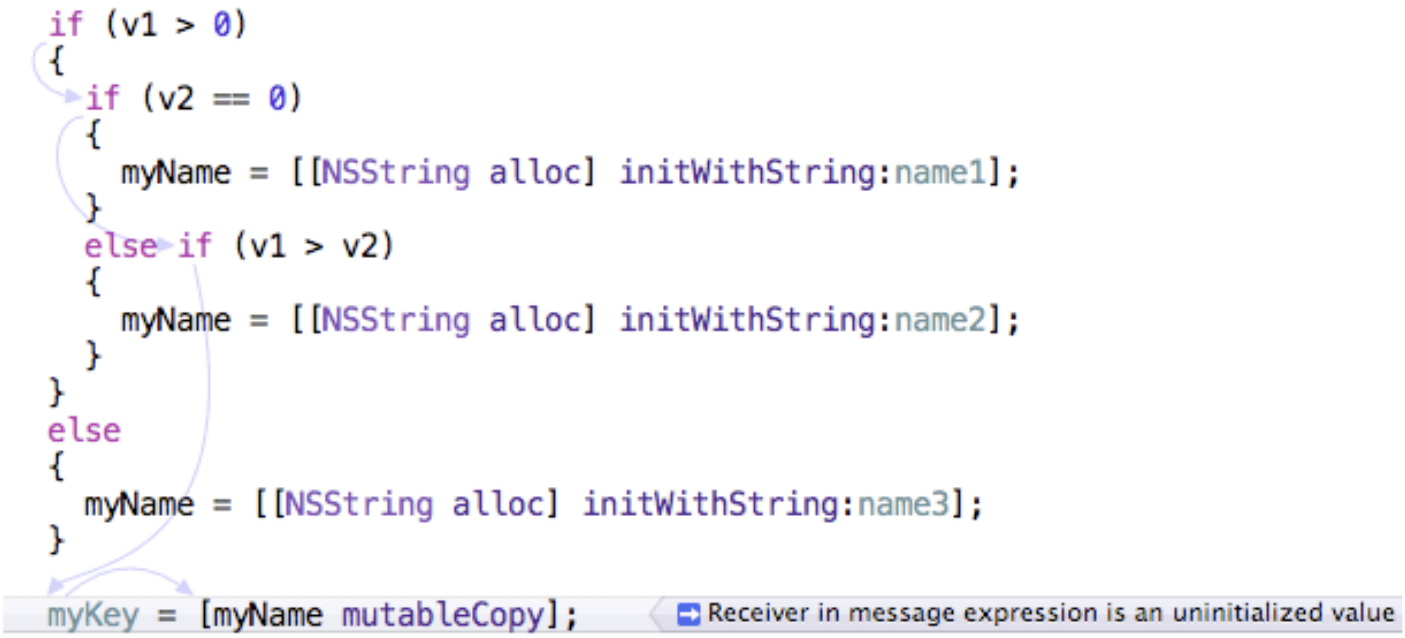

Parfois, l'erreur est parfaitement claire, mais elle semble seulement un peu décalée, comme dans cette situation où GCC positionne le message d'erreur à la ligne en-dessous celle où vous devez rajouter le ; manquant.




Ces petites chose là comptent, vous savez ? Clang fait mieux :




Que vous le croyiez ou non, ces petites choses ont beaucoup d'importance pour les développeurs. Et puis, il y a ces choses pas si petites, qui comptent encore plus, comme l'analyseur statique fourni par LLVM. L'image qui suit montre comment il affiche sa découverte comme une bogue possible.




Au delà de la fantaisie des petites flèches (qui, admettez-le, sont adorables) la bogue qu'elles soulignent est quelque chose que tout programmeur peut arriver à faire (pour peu qu'il écrive un peu vite). L'analyseur statique est arrivé à la conclusion qu'il y a au moins un chemin par lequel on ne passe pas dans ces conditions imbriquées, et qui laisse la variable myName non initialisée, ce qui rend potentiellement dangereuse la tentative de lui envoyer le message mutableCopy, sur la dernière ligne.

Je suis sûr qu'Apple doit être très friand de l'analyseur statique pour toutes ses applications, et le système d'exploitation lui-même. La recherche d'une méthode automatique pour découvrir les bogues qui peuvent exister depuis des années dans les entrailles d'une base de code gigantesque est presque un sujet pornographique

pour les développeurs - en particulier ceux qui proposent des systèmes d'exploitation. Dans la mesure où Mac OS X 10.6 a moins de bogues que les versions précédentes (10.x.0), LLVM y a sûrement une part significative.

### Le maître du logis

En s'engageant dans la voie du couple puissant Clang-LLVM, Apple a finalement pris le contrôle complet de son système de développement. L'expérience de Code Warrior a finalement convaincu Apple qu'il n'était pas avisé de s'appuyer sur un tiers pour les outils de sa plate-forme de développement. Bien que cela ait pris de nombreuses années, je pense que le plus inconditionnel des fans de Metrowerks devrait admettre que XCode, dans Léopard des neiges est un IDE sacrément bon.

Après des années de bataille à cause de la différence entre les objectifs du projet GCC et ses propres besoins en compilateur, Apple a finalement coupé le cordon. Bien sûr, c'est vrai, GCC 4.2 *est encore* le compilateur par défaut dans Léopard des neiges, Mais c'est une phase de transition. Clang est le compilateur *recommandé*, et le point de mire de tous les efforts futurs d'Apple.

Je sais ce que vous pensez. C'est de l'enflure, et tout... Comment ces compilateurs peuvent-ils aider les développeurs à démultiplier la puissance des myriades de transistors mise à leur disposition ? Comme vous allez le voir dans les sections qui suivent, la tête écailleuse et métallique de LLVM se pointe à quelques endroits clé.

# Les blocs

Dans Léopard des neiges, Apple a introduit une extension de langage C appelée les "blocs". Les blocs ajoutent des fermetures (closures) et des fonctions anonymes (anonymous functions) au C et à ses langages dérivés (C++, Objective C et Objective C++).

Ces caractéristiques sont disponibles dans les langages de programmation dynamique comme Lisp, Smalltalk, Perl, Python, Ruby, et même dans le modeste

Javascript depuis longtemps (des décades, dans le cas de Lisp- un fait volontairement mis en avant par ses adeptes). Alors que les programmeurs de langages dynamiques considèrent des fermetures et les fonctions anonymes comme normales, ceux qui travaillent dans des langages compilés statiques comme C et ses dérivés les trouvent plutôt exotiques. Quant aux non programmeurs, il est vraisemblable qu'ils n'éprouvent aucun intérêt pour le sujet. Mais je vais quand même tenter de m'expliquer, car les blocs forment le fondement de quelques technologies intéressantes qui vont être abordées plus tard.

Peut-être que la façon la plus simple d'expliquer les blocs, c'est de dire qu'ils font des fonctions une autre forme de données. Les langages hérités de C ont déjà des pointeurs sur les fonctions, qui peuvent être passés comme des données, mais ceux-ci ne peuvent pointer que sur des fonctions qui ont été créées au moment de la compilation. La seule façon d'influencer le comportement d'une telle fonction, c'est de passer des arguments différents à la fonction, ou de définir des variables globales auxquelles on accède à partir de la fonction. Ces deux approches ont de gros désavantages.

Le passage d'arguments devient compliqué quand leur nombre et leur complexité s'accroît. Et puis, il se peut que vous ayez un contrôle limité sur les arguments qui vont être passés à votre fonction, comme dans le cas des fonctions de rappel (callbacks). Pour compenser, vous pouvez avoir à empaqueter tout ce dont vous avez besoin dans un objet contexte d'une forme ou d'une autre. Mais quand, comment, par qui les données du contexte vont-elle être utilisées peut être difficile à définir précisément. Souvent, une seconde fonction de rappel est nécessaire pour cela. C'est très pénible.

Quant à l'utilisation des variables globales, en plus du fait bien connu qu'elles sont anti-pattern, elles sont aussi non sécurisées vis à vis des threads. Les sécuriser exige des verrous ou une autre forme d'exclusion mutuelle pour empêcher de multiples invocations de la même fonction de se marcher sur les pieds. Et s'il y a quelque chose de pire que de naviguer dans la mer des APIs basées sur des fonctions de rappel, c'est de se colleter manuellement à des problèmes de sécurité des processus légers (threads).

Les blocs évitent tous ces problèmes en permettant à des amas de code fonctionnel -les blocs- d'être définis à l'exécution. C'est plus facile à comprendre avec un exemple. Je vais commencer en utilisant du Javascript, qui a une syntaxe un peu plus familière, mais les concepts sont les mêmes.

```
b = get_number_from_user();

multiplier = function(a) { return a * b };
```

J'ai créé une fonction appelée multiplier qui prend un seul argument, a, et le multiplie par une seconde valeur, b, qui est founie par l'utilisateur à l'exécution. Si l'utilisateur fournit le nombre 2, alors, un appel à multiplier(5) renvoie la valeur 10.

```
b = get_number_from_user(); // assume it's 2

multiplier = function(a) { return a * b };

r = multiplier(5); // 5 * 2 = 10
```

Voici l'exemple ci-dessus réalisé avec des blocs en C :

```
b = get_number_from_user(); // assume it's 2

multiplier = ^ int (int a) { return a * b; };

r = multiplier(5); // 5 * 2 = 10
```

Par comparaison avec le code Javascript, j'espère que vous pouvez vous rendre compte comment la version C fonctionne. Dans l'exemple en C, l'accent circonflexe est la clé de la syntaxe des blocs. C'est un peu vilain, mais c'est très proche du C, dans la mesure où on adopte la syntaxe C existante pour les pointeurs sur une fonction, avec un ^ qui remplace *, comme le montre l'exemple.

```
/* A function that takes a single integer argument and returns
   a pointer to a function that takes two integer arguments and
   returns a floating-point number. */
float (*func2(int a))(int, int);

/* A function that takes a single integer argument and returns
   a block that takes two integer arguments and returns a
   floating-point number. */
float (^func1(int a))(int, int);
```

Vous pouvez me croire, quand je vous dis que cette syntaxe est tout à fait cohérente pour des programmeurs C aguerris.

Et maintenant, est-ce que ça veut dire que C est soudain devenu dynamique, un langage de haut niveau comme JavaScript ou Lisp ? Pas vraiment. La distinction

entre la pile et le tas, les règles qui régissent les variables automatiques et statiques, et autres choses sont encore là, avec leurs pleins effets. Et en plus, il y a maintenant un *nouvel ensemble* de règles pour définir comment les blocs interagissent avec. Il y a même un attribut new_block de type storage pour contrôler la visibilité et la durée de vie des valeurs utilisées dans les blocs.

Tout cela dit, les blocs représentent encore un gain énorme dans C. Grâce aux blocs, les APIs sympathiques dont on a depuis longtemps profité dans les langages dynamiques sont maintenant disponibles pour les langages dérivés du C. Par exemple, supposons que vous vouliez appliquer une certaine opération à chaque ligne d'un fichier. Pour le faire, un langage de bas niveau comme le C a besoin du code classique pour ouvrir et lire le fichier, gérer les erreurs, lire chaque ligne dans une mémoire tampon, et tout nettoyer à la fin.

```
FILE *fp = fopen(filename, "r");

if (fp == NULL) {
  perror("Unable to open file");
}
else {
  char line[MAX_LINE];

  while (fgets(line, MAX_LINE, fp)) {
    work;
    work;
    work;
  }

  fclose(fp);
}
```

La partie en rouge est une représentation simplifiée de ce que vous voulez faire pour chaque ligne. Le reste est le code standard nécessaire. Si vous avez à faire des opérations variées sur les lignes de nombreux fichiers, ce code standard devient lassant.

Ce que vous aimeriez faire, c'est l'incorporer dans une fonction que vous pourriez appeler. Mais alors, vous êtes confronté(e) à la difficulté d'exprimer les opérations que vous voudriez faire sur chaque ligne des fichiers. Au milieu de chaque bloc standard, beaucoup de lignes de code, peut-être pour exprimer les opérations à faire. Ce code peut être amené à utiliser ou modifier des variables locales qui sont modifiées par programme, si bien que des pointeurs à des fonctions ne marchent pas. Que faire ?

Grâce aux blocs, vous pouvez définir une fonction qui prend un nom de fichier et un bloc comme arguments. Cela vous cache tout le code sans intérêt.

```
foreach_line(filename, ^ (char *line) {
  work;
  work;
  work;
});
```

Ce qui reste est une expression beaucoup plus claire de ce que vous voulez faire, avec moins de bruit autour. L'argument après le nom de fichier est un bloc littéral, qui prend comme argument une ligne de texte.

Même quand le volume du code standard est faible, le gain en simplicité et en clarté sont encore appréciables. Voyez cette boucle, la plus simple possible, qui s'exécute un nombre de fois déterminé. En C, même cette construction de base offre un grand nombre de risques de bogues. Faisons dix fois do_something() :

```
for (int i = 0; i <= 10; i++) {
  do_something();
}
```

Mince, j'ai fait une petite faute, là non ? Ça arrive aux meilleurs d'entre nous. Mais pourquoi ce code serait-il plus compliqué que la phrase qui sert à le décrire ? Répéter quelque chose 10 fois ! Je n'ai pas voulu le faire 11 fois ! Les blocs peuvent aider, si on fait un peu l'effort de définir une fonction plus claire :

```
typedef void (^work_t)(void);

void repeat(int n, work_t block) {
  for (int i = 0; i < n; ++i)
    block();
}
```

On peut alors supprimer la bogue pour de bon :

```
repeat(10, ^{ do_something() });
repeat(20, ^{ do_other_thing() });
```

Et souvenez-vous, l'argument du bloc pour repeat() peut contenir exactement le même type de code, copié et collé, littéralement, que celui qui serait apparu dans une boucle for...

Toutes ces possibilités, et plus, ont été utilisées par les langages dynamiques : map, reduce collect, etc... Bienvenue, programmeurs en C, dans un ordre plus haut.

Apple a appris la leçon par cœur, en rajoutant plus de 100 nouvelles APIs qui utilisent les blocs dans Léopard des neiges. Beaucoup de ces APIs ne seraient pas possibles sans les blocs, et toutes sont plus élégantes et plus concises qu'elles ne l'auraient été autrement.

Apple a l'intention de proposer les blocs comme extension à un ou plusieurs des langages basés sur le C, bien qu'il ne soit pas bien facile de savoir, quel organisme de standard peut être réceptif à ce genre de proposition. Pour le moment, les blocs sont supportés par les quatre compilateurs d'Apple dans Mac OS X.

# La concurrence dans un monde nouveau : prélude

La lutte pour faire un usage efficace d'un grand nombre d'ordinateurs indépendants n'est pas nouvelle. Pendant des décades, le champ de l'informatique à haute performance a abordé ce problème. Les défis rencontrés par les gens qui écrivent des programmes pour les super-ordinateurs sont maintenant descendus jusqu'aux ordinateurs de bureau, et même jusqu'au mobiles.

Dans l'industrie informatique, certains ont vu les choses venir plus tôt que d'autres. Il y a presque 20 ans, Be Inc fut constituée autour de l'idée visant à créer une plate-forme d'ordinateur personnel débarrassée des limitations du passé, et prête pour exploiter l'abondance qui s'annonçait de plusieurs unités de calcul indépendantes. Pour cela, Be créa la BeBox, un ordinateur personnel à deux CPUs, et BeOS, un tout nouveau système d'exploitation.

L'expression préférée pour caractériser BeOS fut le "le multithread ubiquitaire" (pervasive mutithreading) . La Be Box, et les autres machines qui faisaient tourner Be OS tiraient le meilleur parti des ressources de calcul limitées (selon les standards actuels) à leur disposition. Les démos furent impressionnantes. Une machine à deux processeurs de 66 MHz (ne m'obligez pas à faire un autre croquis) pouvait afficher plusieurs vidéos simultanément, tout en jouant aussi

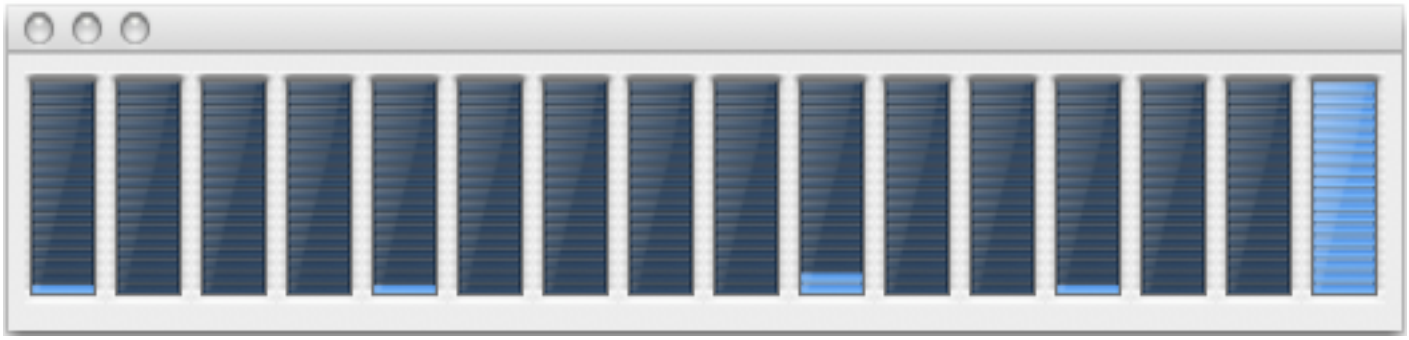
plusieurs pistes audio à partir d'un CD, -quelques unes en tâche de fond- et en même temps, l'interface utilisateur restait capable de répondre parfaitement.

Laissez-moi vous dire, car j'ai vécu cette période moi-même, que l'expérience était à vous couper le souffle à l'époque. Be OS fit instantanément des convertis parmi des centaines de mordus de technologie, dont beaucoup considèrent que l'expérience d'un ordinateur de bureau actuel n'atteint pas la vitesse de réaction de Be OS. C'est sans doute vrai émotionellement, si ce n'est littéralement.

Après avoir été sur le point d'acheter Be à la fin des années 90, Apple a finalement acquis NeXT, et le reste, c'est de l'histoire. Mais si Apple avait adopté le plan Be, les développeurs Mac auraient eu une route très difficile devant eux. Alors que ce multithread envahissant permettait des démos d'une technologie impressionnante, et une expérience inoubliable pour l'utilisateur, il pouvait être très exigeant pour le programmeur. Be OS était tout en threads, allant jusqu'à utiliser un thread séparé pour chaque fenêtre. Que vous le vouliez ou non, votre programme Be OS devait être multithread.

La programmation parallèle est notoirement difficile, et la gestion manuelle des threads de style POSIX représente la partie la plus profonde de la piscine. Les meilleurs programmeurs dans le monde sont sollicités pour créer de gros programmes multithread dans des langages de bas niveau comme C ou C++, sans se retrouver empalés sur les pieux de l'interblocage (deadlock), des situations de compétition (race conditions), et d'autres périls inhérents à l'exécution de très nombreux threads simultanés qui partagent le même espace de mémoire. Il faut faire une utilisation très prudente des primitives de verrouillage pour éviter des situations de concurrence dans des données partagées, qui accaparent les performances, et les bogues, oh les bogues ! Le terme "Heisenbug" (Heisenbug) aurait bien pu être inventé pour la programmation multithread.

Dix neuf ans après que Be se soit engagé dans la voie consistant à mettre plus de silicium dans les PCs, le défi n'a fait que s'amplifier. Les transistors sont là, plus que jamais. Les programmes à un seul thread sur les Macs de haut de gamme d'aujourd'hui même quand ils utilisent "100 % de CPU" n'illuminent qu'une seule tour parmi les 16 qu'affiche la fenêtre du moniteur de CPU.



*Plein de transistors inutilisés*

*La roulette de la mort (The Iconfactory)*

Et pitié pour l'utilisateur si ce cœur encombré du CPU fait tourner le thread principal d'une application avec interface graphique sous Mac OS X. Un thread principal sur un CPU saturé signifie que aucune entrée utilisateur n'est poussée dans la queue d'événements par l'application. Quelques secondes de ce régime, et une vieille amie fait son apparition : la roulette multicolore de la mort.

Voilà l'ennemi : un matériel avec plus de ressources de calcul que les programmeurs ne savent en utiliser, la plus grande partie inactive, et pendant ce temps, l'utilisateur est complètement bloqué dans ses tentatives d'utiliser l'application courante. Quelle est la réponse de Léopard des neiges ? Lisez la suite...

# Grand Central Dispatch


*La marque Apple de GDC (le service du rail)*

La réponse de Léopard des neiges aux cafouillages de la simultanéité s'appelle Grand Central Dispatch (GCD). Comme pour QuickTime X, le nom est très bien choisi, quoi qu'il ne soit pas entièrement compréhensible avant que vous ayez compris la technologie.

La première chose à savoir au sujet de GCD est que ce n'est pas un nouveau framework Cocoa ou un ornement spécifique de ce genre. C'est une bibliothèque C complète, enchâssée dans les couches les plus élémentaires de Mac OS X. (Il est dans libSystem, qui contient aussi libc, et d'autre code, qui se place tout à fait à la

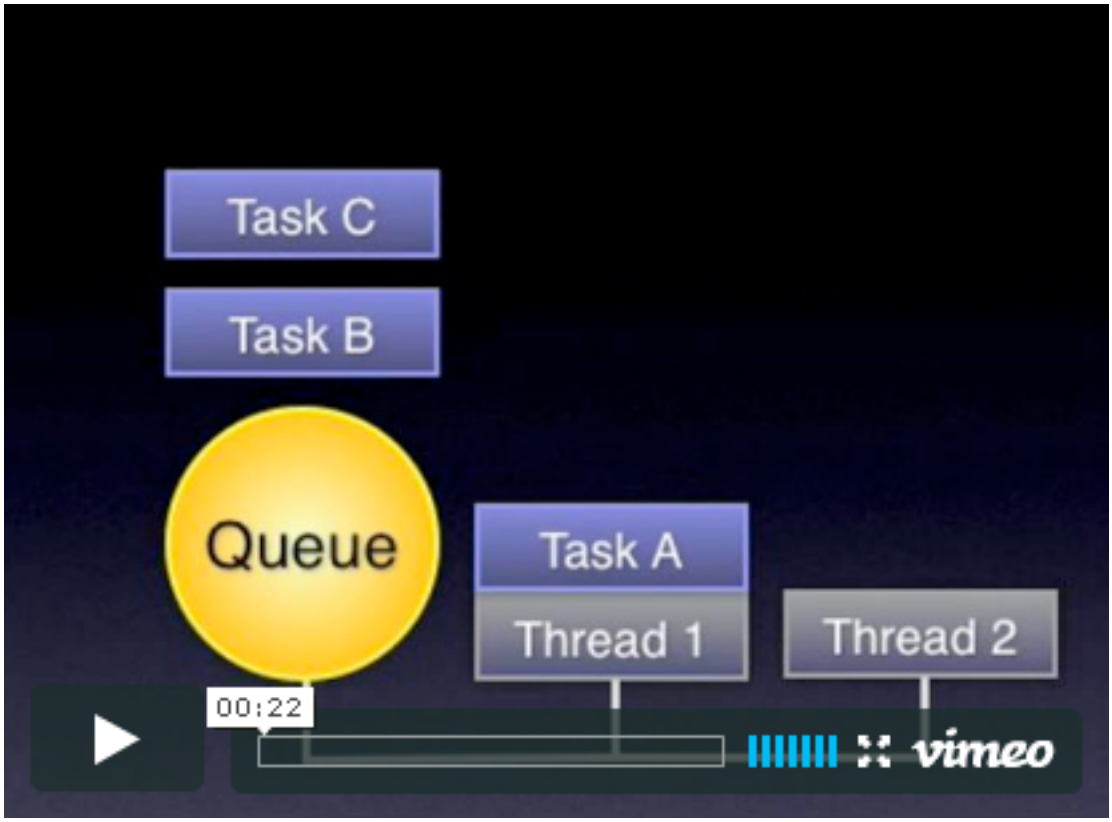
base de l'espace utilisateur).

Il n'est pas nécessaire de lier une nouvelle bibliothèque quand vous utilisez GCD dans votre programme. insérez seulement <dispatch/dispatch.h> et vous voilà dans la course. Le fait que GCD est une bibliothèque C signifie qu'on peut l'utiliser avec tous les langages dérivés du C supportés par Mac OS X : objective C, C++ et Objective C++.

## Les queues et les threads

GCD est basé sur quelques entités simples. Commençons par les queues. Une queue dans GCD est juste ce que cela suggère. Les tâches sont insérées dans la queue, et retirées de la queue à l'aide d'une procédure FIFO. (Cela veut dire "premier entré, premier sorti", comme la queue à la caisse d'un supermarché, pour ceux qui ne voudraient pas suivre le lien précédent). Retirer une tâche de la queue consiste à la passer à un thread qui va provoquer son exécution et faire le travail qu'elle doit faire.

Bien que les queues de GCD transfèrent les tâches à des threads dans un ordre FIFO, plusieurs tâches d'une même queue peuvent être exécutées en parallèle à un moment donné comme le montre cette animation.




*Lancez l'animation pour voir la queue de GCD en action*

Remarquez que la tâche B se termine avant la tâche A. Bien que la sortie de queue soit en FIFO, l'exécution des tâches ne l'est pas. Et notez que, bien qu'il y ait eu trois tâches dans la queue, deux threads seulement ont été utilisés. C'est une caractéristique importante de GCD que nous allons voir bientôt.

Mais d'abord, considérons une autre forme de queue. Une queue sérielle fonctionne comme une queue normale, mais n'exécute qu'une tâche à la fois. Cela veut dire que, dans une queue sérielle, l'exécution des tâches est aussi FIFO. Des queues sérielles peuvent être créées explicitement, comme des queues normales, mais chaque application a aussi sa propre "queue principale", implicite, qui est une queue sérielle, et qui s'exécute sur le thread principal.

L'animation montre que des threads sont créés quand il y a du travail à faire, et disparaissent quand on n'en a plus besoin. D'où viennent ces threads, et où vont-ils quand ils ont fini ? GDC maintient un paquet global de threads qu'il attribue aux queues quand elles en ont besoin. Quand une queue n'a plus de tâches en attente à fournir au thread, le thread retourne dans le paquet.

Ceci est un aspect très important de la conception de GCD. D'une façon étonnante, l'une des choses les plus difficiles pour obtenir le maximum de performances à partir d'un ensemble de threads gérés manuellement est de se représenter exactement combien il faut créer de threads. Trop peu, et vous laissez votre matériel inoccupé. Trop de threads, et vous passez une partie importante du temps à les répartir entre les cœurs disponibles.

Supposons qu'un programme a un problème qui peut être scindé en 8 unités de travail séparées, indépendantes. Si ce programme crée quatre threads sur une machine à 8 cœurs, est-ce que c'est trop, ou trop peu de threads ? Question piège ! La réponse est que cela dépend de ce qui se passe en plus sur le système.

Si six des huit cœurs sont complètement saturés quand il font un autre travail, la création de quatre threads obligera l'OS à gaspiller du temps à faire alterner ces quatre threads sur les deux cœurs disponibles. Mais si le processus qui saturait les six cœurs se termine ? Il y a alors huit cœurs disponibles, mais seulement quatre threads, ce qui laisse la moitié des cœurs inoccupés.

A l'exception des programmes qui peuvent raisonnablement s'attendre à utiliser la totalité de la machine pour leur exécution, il n'y a aucun moyen pour le programmeur de prévoir à l'avance combien de threads il doit créer. Parmi les cœurs d'une machine, combien sont utilisés ? Si des cœurs deviennent disponibles, comment mon programme peut-il le savoir ?

La conclusion, c'est que ce nombre optimum de threads à mettre en jeu à un moment donné peut être déterminé par une entité unique, qui est à même de connaître ce qui se passe. Sous Léopard des neiges, cette entité s'appelle GCD. Elle va maintenir zéro threads dans son paquet s'il n'y a aucune queue avec des tâches à exécuter. A mesure que les tâches sont retirées de la queue, GCD va créer et distribuer les threads de façon à optimiser le matériel disponible. GCD connaît le nombre de cœurs du système, et il sait combien de threads sont en train d'exécuter des tâches à un moment donné. Quand une queue n'a plus besoin d'un thread, il est remis dans le paquet, où GCD peut le réutiliser pour une autre queue qui a une tâche en attente.

Il y a d'autres optimisations inhérentes à ce schéma. Dans Mac OS X, les threads sont relativement chargés. Chaque thread maintient son propre ensemble de valeurs de registre (register), son pointeur de pile, et son compteur ordinal, plus des structures de données du noyau qui vérifient ses certifications de sécurité, la priorité d'exécution, un ensemble de signaux en cours d'utilisation et de masques de signaux, etc... Tout cela rajoute jusqu'à 512 Ko de mémoire supplémentaire (overhead, ou à côtés) par thread. Créez un millier de threads, et vous avez avez brûlé environ un demi Go de mémoire et de ressources du noyau seulement en à côtés, en plus des données effectives de chaque thread.

Comparez un thread avec un bagage de 512 Ko avec les queues de GCD qui ont simplement 256 octets d'overhead. Les queues sont très légères, et les développeurs sont encouragés à en créer autant qu'ils en ont besoin, même des milliers. Dans l'animation, quand la queue reçoit deux threads pour exécuter ses trois tâches, elle exécute deux tâches dans l'un des threads. Non seulement les threads sont lourds en terme de mémoire utilisée en overhead, mais ils sont aussi coûteux à créer. Créer un nouveau thread pour chaque tâche serait le scénario le plus mauvais possible. A chaque fois que GCD utilise un thread pour exécuter plus d'une tâche, on y gagne en efficacité totale du système.

Rappelez-vous le problème du programmeur cherchant à se représenter combien il doit créer de threads. S'il utilise GCD, il n'a pas besoin de s'embêter avec tout ça. Il peut à la place se concentrer à optimiser dans l'abstrait la simultanéité de son algorithme. Si le scénario du meilleur cas pour ce problème devrait utiliser 500 tâches concurrentes, il peut y aller et créer 500 queues GCD, et distribuer son travail entre elles. GCD va s'arranger pour définir combien de threads il doit créer pour faire le travail. Et en plus, il va ajuster le nombre de threads de façon dynamique à mesure que les conditions changent dans le système.

Mais ce qui est encore plus important, à mesure que de nouveaux matériels sont disponibles avec de plus en plus de cœurs de CPUs, le programmeur n'a pas besoin

de changer son application. GCD, va s'adapter de façon transparente à toutes les ressources de calcul disponibles, jusqu'à un optimum de simultanéité (mais pas au delà) tel qu'il a été défini par le programmeur quand il a choisi le nombre de queues à créer.

Mais attendez, il y a plus encore ! les queues GCD peuvent être organisées en graphes orientés acycliques de complexité arbitraire. (En fait, ils peuvent être cycliques aussi, mais leur comportement n'est pas défini ; ne faites pas ça). Des hiérarchies de queues peuvent être utilisées pour canaliser les tâches à partir de sous-systèmes disparates, en un ensemble plus étroit de queues contrôlées de façon coordonnée, et pour forcer un ensemble de queues normales à déléguer leur travail à une queue sérielle, ce qui indirectement, les sérialise toutes.

Il y a aussi plusieurs niveaux de priorité pour les queues, qui définissent à quel rythme et avec quelle priorité les threads leur sont attribués à partir du paquet. Les queues peuvent être suspendues, reprises, et annulées. Les queues peuvent aussi être regroupées ce qui permet à toutes les tâches fournies au groupe d'être suivies et de compter pour une unité.

Finalement, l'utilisation des queues et des threads dans GCD représente une architecture simple, élégante, mais aussi très pragmatique.

# L'asynchronicité

D'accord, GCD est une manière élégante d'utiliser avec efficacité le matériel disponible. Mais, est-ce que c'est vraiment mieux que l'approche par le multi-threading utilisée par Be OS ? Nous avons déjà vu quelques façons dont GCD évite les pièges de Be OS (notamment la réutilisation des threads et la gestion d'un paquet global de threads de taille adaptée au matériel utilisé). Mais que dire de la surcharge subie par le programmeur associée à l'utilisation de threads à des endroits où ils compliquent, plutôt qu'ils améliorent l'application ?

GCD représente une philosophie qui est à l'opposé du concept de multi-thread ubiquitaire de Be OS. Plutôt que d'obtenir la réactivité en permettant à chaque composant possible d'une application de s'exécuter en simultanéité dans son propre

thread (et en payant le prix fort en terme de complexité pour le partage des données et les opérations de verrouillage), GCD encourage une approche beaucoup plus limitée et hiérarchique : un thread d'application principale, où tous les évènements utilisateurs sont gérés, et l'interface mise à jour, et des threads spécifiques au programme, qui font le travail voulu.

Autrement dit, GCD n'*exige* pas des développeurs qu'ils imaginent la meilleure façon de scinder le travail de leur application en une multitudes de threads concurrents (bien que, s'ils sont prêts à le faire, GCD va collaborer et être capable d'aider). A son niveau le plus élémentaire, GCD ambitionne à encourager les développeurs à ne plus penser de façon synchrone, mais asynchrone. Quelque chose du genre : "écrivez votre application comme vous en avez l'habitude, mais s'il y a une partie quelconque du travail dont on peut raisonnablement penser qu'il prendra plus de quelques secondes, alors, pour l'amour de Zarzycki, sortez le du thread principal !"

C'est comme ça ; pas plus, pas moins. Le bannissement de la roulette multicolore est la pierre angulaire de la réactivité de l'interface utilisateur. D'une certaine façon, tout le reste, c'est de la sauce. Mais la plupart des développeurs le savent intuitivement, alors, pourquoi voyons-nous encore cette roulette dans les applications Mac OS X ? Pourquoi toutes les applications n'exécutent-t-elles pas déjà les tâches susceptibles d'être longues dans des threads en tâche de fond ?

Quelques raisons ont déjà été mentionnées (c'est à dire la difficulté de connaître le nombre de threads à créer) mais la raison principale est beaucoup plus réaliste. Isoler un thread, et récupérer son résultat a toujours été un peu difficile. Ce n'est pas tant que c'est techniquement difficile, c'est seulement que cela constitue une rupture, le passage de l'écriture du travail fait par l'application au codage détaillé de la gestion des tâches. Si bien que, notamment dans les cas limites, comme une opération qui peut prendre 3 à 5 secondes, les développeurs la réalisent de façon synchrone, et passent à la suite.

Malheureusement, il y a un nombre surprenant de choses très communes qu'une application peut faire pour s'exécuter rapidement la plupart du temps, mais qui peuvent aussi durer beaucoup plus longtemps que quelques secondes quand quelque chose va mal. Tout ce qui concerne le système de fichiers peut caler aux niveaux les plus bas de l'OS (c'est à dire avec des appels read() et write() qui se bloquent), et être sujet à une suspension de très longue durée non prise en compte par le développeur de l'application. La même chose intervient pour les recherches de noms (par exemple DNS ou LDAP), qui presque toujours s'exécutent instantanément, mais laissent beaucoup d'applications pratiquement sans défense quand elles décident de prendre tout leur temps pour renvoyer un résultat. Si bien

que les applications les plus méticuleusement construites peuvent finir par vous renvoyer la roulette multicolore en pleine figure de temps en temps.

Avec GCD, Apple dit que cela n'a pas à se produire. Par exemple, supposez qu'une application basée sur un document a un bouton qui, quand on clique dessus, analyse le document en question, et affiche des statistiques intéressantes à son propos. Dans les cas courants, cette analyse doit se faire en moins d'une seconde, si bien qu'on utilise le code suivant pour associer le bouton à l'action :

```
- (IBAction)analyzeDocument:(NSButton *)sender
{
  NSDictionary *stats = [myDoc analyze];
  [myModel setDict:stats];
  [myStatsView setNeedsDisplay:YES];
  [stats release];
}
```

La première ligne du corps de la fonction analyse le document, la seconde met à jour l'état interne de l'application, et la troisième dit à l'application que la vue statistiques doit être mise à jour pour être conforme au nouvel état. Tout cela suit un patron très courant, et ça marche bien aussi longtemps qu'aucune de ces étapes, qui tournent toutes dans le même thread principal, ne prend pas trop longtemps. Parce que, après que l'utilisateur a pressé le bouton, l'application a besoin de prendre en compte cette entrée aussi vite que possible, pour revenir à la boucle d'évènements principale, et gérer l'action suivante de l'utilisateur.

Le code en question marche bien jusqu'à ce que l'utilisateur ouvre un document très gros, ou très complexe. Soudain, l'étape d'analyse ne prend plus une ou deux secondes mais 15 ou 30. Salut, la roulette multicolore. Et alors, le developpeur risque de tergiverser : "c'est vraiment une situation exceptionnelle. La plupart de mes utilisateurs n'ouvriront jamais un fichier aussi gros. Et de toute façon, je n'ai pas l'intention de commencer à lire la documentation, et à rajouter du code supplémentaire à cette fonction simple de quatre lignes. Ce rajout amoindrirait le code qui fait du bon travail".

Bien, et si je vous disais que vous pouvez déplacer l'analyse du document à l'arrière plan en ajoutant seulement 2 lignes de code, (oui, et aussi 2 lignes d'accolades de fermeture), toutes incluses dans la fonction existante ? Pas d'objets globaux, pas de gestion des threads, pas de fonction de rappel, pas d'arrangement des arguments, pas d'objets contextuels, pas même de variables additionnelles. Admirez ! C'est Grand Central Dispatch.

```
- (IBAction)analyzeDocument:(NSButton *)sender
{
  dispatch_async(dispatch_get_global_queue(0, 0), ^{
    NSDictionary *stats = [myDoc analyze];
    dispatch_async(dispatch_get_main_queue(), ^{
      [myModel setDict:stats];
      [myStatsView setNeedsDisplay:YES];
      [stats release];
    });
  });
}
```

Il y a une foutue accumulation de puissance dans ces deux lignes de code. Toutes les fonctions de GCD commencent par dispatch_, et vous pouvez voir quatre appels de ce type dans les lignes en bleu du code ci-dessus. La clé à l'intrusion minimale de ce code est révélées dans le second argument des deux appels à dispatch_async (). Jusqu'à présent, j'ai abordé les "unités de travail" sans préciser exactement comment GCD les organise. La réponse est là, et devrait sembler évidente avec du recul : les blocs ! La capacité des blocs à s'accaparer le contexte environnant est ce qui permet à ces appels GCD d'être lâchés en plein dans du code existant, sans avoir besoin de dispositions supplémentaires, ni d'une réorganisation, ni d'autres contorsions au service de l'API.

Mais la meilleure partie de ce code est la façon dont il détecte quand la tâche en arrière plan a terminé, pour montrer le résultat. Dans le code synchrone, l'appel à la méthode *analyse*, et le code pour mettre à jour l'affichage de l'application figurent simplement dans l'ordre voulu à l'intérieur de la fonction. Dans le mode asynchrone, miraculeusement, c'est aussi le cas. Voilà comment ça marche.

L'appel externe à dispatch_async() dépose une tâche sur la queue globale simultanée. Cette tâche, représentée par le bloc passé comme second argument, contient l'appel à la méthode *analyse* potentiellement consommatrice de temps, plus *un autre* appel à dispatch_async qui met la tâche sur la queue principale -une queue sérielle qui exécute le thread principal- pour mettre à jour l'interface utilisateur de l'application.

Les mises à jour de l'interface utilisateur doivent toutes être faites dans le thread principal pour une application Cocoa, si bien que le code dans le bloc intérieur ne pourrait pas être exécuté ailleurs. Mais plutôt que d'obliger le thread en arrière plan à renvoyer une forme spécifique de notification au thread principal quand l'appel à la méthode *analyse* est terminé (donc à ajouter du code à l'application pour détecter et gérer cette notification), le travail qui doit être accompli sur le thread principal

pour mettre à jour l'affichage est encapsulé dans un autre bloc à l'intérieur du bloc le plus gros. Quand l'appel à *analyse* est terminé, le bloc interne est déposé dans la queue principale, où il va (le cas échéant) s'exécuter dans le thread principal, et faire son travail consistant à mettre à jour l'affichage.

Simple, élégant, et efficace. Et pour les développeurs, plus d'excuses...

Croyez-le ou non, c'est tout aussi simple de prendre une implémentation sérielle de tout un ensemble d'opérations indépendantes, et de les parallèliser. Le code ci-dessous fonctionne sur *count* éléments de *data*, et récapitule les résultats quand tous les éléments ont été pris en compte.

```
for (i = 0; i < count; i++) {
    results[i] = do_work(data, i);
}

total = summarize(results, count);
```

Et voici maintenant la version parallèle qui dépose une tâche séparée sur la queue globale simultanée. (Encore une fois, c'est à GCD de décider combien de threads il va effectivement utiliser pour exécuter ces tâches).

```
dispatch_apply(count, dispatch_get_global_queue(0, 0), ^(size_t i) {
    results[i] = do_work(data, i);
});

total = summarize(results, count);
```

Et là, vous l'avez : une boucle *for* remplacée par un équivalent de calcul simultané dans une ligne de code. Aucune préparation, pas de variables additionnelles, pas de décisions impossibles à prendre à propos du nombre optimal de threads, pas de travail supplémentaire requis pour attendre que tous ces tests indépendants soient terminés. (L'appel à *dispatch_apply()* ne va pas se terminer avant que toutes les tâches qu'il a distribuées ne se soient finies). Ahurissant.

## L'excellence de Grand central

De toutes les APIs rajoutées à Léopard des neiges, Grand Central Dispatch a les implications les plus profondes sur le futur de Mac OS X. Jamais auparavant, il

n'avait été aussi facile de travailler de façon asynchrone, et de répartir la charge de travail entre de nombruex CPUs.

Quand j'ai entendu parler de Grand central Dispatch pour la première fois, j'étais très sceptique. Les cerveaux les plus éminents de l'informatique ont travaillé pendant des décennies sur le problème de la meilleure façon de parallèliser les charges de calcul. Et maintenant, Apple promettait apparemment de résoudre ce problème ? Ridicule !

Mais Grand Central Dispatch ne résout pas ce problème *du tout*. Il ne fournit aucune aide dans la façon de décider *comment* partager votre travail en tâches indépendamment exécutables, autrement dit choisir quels morceaux peuvent être exécutés de façon asynchrone ou parallèle. C'est encore entièrement le rôle du développeur (et c'est encore très difficile). Ce qu'apporte GCD à la place, est beaucoup plus pragmatique. Quand un développeur a identifié quelque chose qui peut être séparé en tâches simultanées, GCD rend la chose aussi facile et non intrusive que possible pour le réaliser *effectivement*.

L'utilisation des queues FIFO, et particulièrement l'existence des queues sérialisées , semblent aller à l'encontre de l'esprit d'une simultanéité ubiquitaire. Mais nous avons vu où l'idéal platonique du multithread conduit, et ce n'est pas un endroit plaisant pour les développeurs.

L'un des slogans utilisés par Apple pour Grand Central Dispatch est "des îles de sérialisation dans un océan de simultanéité". Cela correspond bien à la réalité pratique qui consiste à rajouter plus de simultanéité aux applications de bureau qui tournent actuellement. Ces îles sont ce qui protège les développeurs des problème épineux de l'accès simultané aux données, du verrouillage, et autres pièges du multi-threading. Les développeurs sont encouragés à identifier les fonctions de leurs applications qui seront mieux exécutées à l'écart du thread principal, même si elles consistent en plusieurs tâches séquentielles, ou en tâches indépendantes. GCD facilite la partition d'une unité de travail tout en maintenant l'ordre existant et les dépendances entre les sous-tâches.

Ceux qui ont une certaine expérience dans la programmation multithread peuvent ne pas être impressionnés par GCD. Ainsi, Apple a fait un paquet de threads. Quel défi ! Ça a toujours été le cas. Mais les anges sont dans les détails. Oui, l'implémentation des queues et des threads a une élégante simplicité, et l'incorporer dans les couches les plus basses de l'OS contribue à abaisser la hauteur de la barrière d'entrée, mais c'est l'API construite autour des blocs qui rend Grand Central Dispatch aussi attractif pour les développeurs. Tout comme Time Machine fut le "premier dispositif de sauvegarde que les gens utilisent réellement", Grand Central

Dispatch est prêt à répandre la conception asynchrone des applications, jusqu'à présent un art obscur, auprès de tous les développeurs de Mac OS X. Je ne peux pas attendre.

# OpenCL

*OpenCL est associé au Core*

Jusqu'à présent, nous avons vu quelques exemples consistant à faire plus avec plus : une infrastructure de compilateur plus moderne, qui supporte une nouvelle possibilité importante du langage et une API pour la simultanéité, puissante et pragmatique. Tout cela représente un gros effort pour aider les développeurs, et l'OS lui-même fait un maximum pour utiliser le matériel disponible.

Mais les CPUs ne sont pas les seuls composants qui fournissent une surabondance de transistors. Quand on parle de prolifération de moteurs de calcul indépendants, un autre morceau de silicium est un tenant du titre incontestable : le GPU.

Les chiffres racontent l'histoire. Alors que les CPUs des Macs contiennent quatre cœurs (qui peuvent monter à huit cœurs logiques grâce au multithreading symétrique) les GPUs de haut de gamme peuvent contenir bien plus de 200 cœurs. Alors que les CPUs dépassent tout juste les 100 GFLOPS, les meilleurs GPUs peuvent dépasser 1000 GFLOPS. C'est mille milliards d'opérations en virgule flottante par seconde. Et comme les CPUs, il y a maintenant plusieurs GPUs sur une carte.

## Ecrire pour un GPU

Malheureusement, les cœurs des GPUs ne sont pas des processeurs à usage général (du moins, pas encore). Ce sont des engins de calcul plus simples, qui ont évolué, à partir des fonctions implantées dans le silicium de leurs ancêtres, qu'on ne pouvait pas du tout programmer. Ils ne fournissent pas le riche ensemble d'instructions des CPUs, la taille maximum des programmes qu'ils peuvent faire tourner est souvent limitée et très petite, et ils ne supportent pas toutes les possibilités du standard de calcul en virgule flottante IEEE.

Aujourd'hui, les GPUs peuvent être programmés, mais les possibilités les plus communes de programmation s'inscrivent dans le domaine graphique : calcul de vertex, de primitives géométriques, des attributs de pixels. La plupart des langages utilisés pour programmer les GPUs sont aussi orientés graphique : HLSL, GLSL, Cg.

Il y a cependant des tâches de calcul en dehors du graphisme qui peuvent tirer profit des GPUs en tant que matériel. Ce serait bien s'il y avait un langage non orienté graphique pour écrire des programmes pour elles. Mais créer une chose de ce genre est un vrai défi. Le matériel des GPUs varie de toutes les façons imaginables : nombre et type des unités d'exécution, jeu d'instructions, architecture de mémoire, et tout ce que vous pouvez imaginer. Les programmeurs ne veulent pas être confrontés à ces différences, mais il est difficile de contourner une absence complète d'une caractéristique, ou l'indisponibilité d'un type particulier de données.

Le vendeur de GPUs NVIDIA a cependant fait une tentative avec CUDA : un sous-ensemble du langage C avec des données de type vecteur, des identificateurs de stockage des données, qui reflètent la hiérarchie de mémoire typique des GPUs, et plusieurs bibliothèques de calcul associées. CUDA n'est qu'un entrant dans le champ bourgeonnant des GPGPU (General Purpose computing on Graphics Processing Units). Mais en tant que fabriquant de CPUs, il doit affronter une bataille difficile avec les développeurs qui veulent en fait une solution indépendante d'un vendeur.

Dans le domaine de la programmation 3G, OpenGL remplit ce rôle. Et comme vous l'avez sûrement deviné à présent, OpenCL a pour ambition de jouer le même rôle pour les calculs d'usage général. En fait, Open CL est encouragé par le même consortium que Open GL, le groupe Khronos, au nom inquiétant. Mais ne vous y trompez pas, OpenCL est le bébé d'Apple.

Apple a compris que la meilleure chance de succès d'OpenCL était de devenir un standard de l'industrie, pas seulement une technologie Apple. Et pour que ça arrive,

Apple avait besoin de la collaboration des principaux vendeurs de GPUs, et en plus, d'un agrément avec un organisme de standardisation bien établi et largement reconnu. Cela a mis du temps, mais maintenant, tout est en place.

OpenCL est beaucoup comme CUDA. Il utilise un langage proche du C, avec les extensions de vecteurs, il a un modèle de hiérarchie de mémoire similaire, et ainsi de suite. Ce n'est pas une surprise, quand on considère qu'Apple a étroitement travaillé avec NVIDIA pendant le développement d'OpenCL. Il n'y a aussi aucune raison pour qu'un vendeur de GPUs bien établi modifie son matériel de façon radicale, jusqu'à adopter un standard qui n'est pas encore établi, si bien qu'OpenCL devait fonctionner parfaitement avec les GPUs conçus pour supporter CUDA, GLSL, et d'autre langages de programmation existants pour les GPUs.

# La différence OpenCL

Tout ça c'est bien beau, mais pour avoir un impact sur la vie de tous les jours des utilisateurs de Macs, les développeurs doivent effectivement *utiliser* OpenCL dans leurs applications. Historiquement, les langages de programmation GPGPU n'ont pas été beaucoup utilisés dans les applications traditionnelles sur les PCs. Il y a à cela plusieurs raisons.

Tout d'abord, écrire des programmes pour un GPU obligeait souvent à utiliser des langages en assembleur spécifiques au vendeur qui étaient très éloignés de l'expérience d'écriture d'une application typique pour PC, utilisant une API GUI (Graphic User Interface) contemporaine. La plupart des langages de type C qui sont venus après restaient très centrés sur le graphisme, ou propres à un vendeur, ou les deux. A moins que faire tourner du code sur un GPU n'accélère une composante essentielle d'une application d'un ordre de grandeur, la plupart des développeurs ne pouvaient pas s'embêter à naviguer dans ce genre de monde étranger.

Et même si le GPU apportait une énorme amélioration de vitesse, s'appuyer sur un matériel graphique pour un calcul à usage général était de nature à rétrécir la portée de l'application. Beaucoup des plus anciens GPUs, notamment ceux des ordinateurs portables, ne peuvent pas du tout utiliser un langage comme CUDA.

La décision majeure d'Apple dans la conception de OpenCL a été de permettre aux programmes OpenCL de tourner non pas seulement sur des GPUs, mais aussi sur des *CPUs*. Un programme OpenCL peut interroger le matériel sur lequel il tourne et identifier tous les composants éligibles à OpenCL, que ce soit des CPUs, des GPUs, ou des accélérateurs pour OpenCL (le serveur Cell Blade d'IBM, oui, ce processeur cellulaire en fait apparemment partie). Le programme peut alors distribuer ses tâches OpenCL à tout composant disponible. On peut aussi créer un seul composant logique, qui consiste en n'importe quelle combinaison de ressources éligibles : deux GPUs, un autre GPU, et deux CPUs, etc...

Les avantages d'être capable de faire tourner des programmes OpenCL sur des GPUs et des CPUs sont évidents. Tout Mac qui tourne sous Léopard des neiges, pas seulement ceux équipés des plus récents modèles de GPUs peut exécuter des programmes qui incorporent du code OpenCL. Mais il y a plus.

Certaines formes d'algorithmes tournent effectivement plus vite sur des CPUs multi-cœurs de haut de gamme que sur des GPUs, fussent-ils les plus rapides disponibles. A la WWDC 2009, un ingénieur de Electronic Arts a fait la démonstration d'une version OpenCL d'un moteur d'habillage d'un de ses jeux qui tournait plus de quatre fois plus vite sur un mac Pro à quatre cœurs que sur une carte NVIDIA GeForce GT285. La réorganisation de l'algorithme, et beaucoup d'autres modifications pour s'adapter aux limitations (et aux avantages) des GPUs, ont réorienté vers les CPU dans une large mesure, mais parfois vous désirez seulement que le système dont vous disposez fonctionne bien tel qu'il est. Etre capable de définir le CPU comme cible est très utile dans ces cas là.

En plus, l'écriture de code vectoriel pour des CPUs Intel "à la manière ancienne", peut être une vraie difficulté. Il faut prendre en compte MMX, SSE, SSE2, SSE3 et SSE4, tous avec des possibilités légèrement différentes, et chacune oblige le programmeur à écrire du code du genre :

```
r1 = _mm_mul_ps(m1, _mm_add_ps(x1, x2));
```

Le support natif d'OpenCL pour les types vecteur démêle le code de façon considérable.

```
r1 = m1 * (x1 + x2);
```

Semblablement, le support d'OpenCL pour un parallélisme implicite fait qu'il est beaucoup plus facile de tirer avantage de nombreux cœurs de CPU. Plutôt que d'écrire tout la logique nécessaire pour scinder les données en morceaux et les

distribuer au matériel disponible pour le parallélisme, OpenCL vous permet de n'écrire que le code pour un seul morceau de données, puis de l'envoyer avec son bloc complet de données, et le niveau désiré de parallélisme au composant de calcul voulu.

Cette disposition est considérée comme normale dans les programmes graphiques traditionnels où le code fonctionne implicitement sur tous les pixels d'une texture ou sur tous les sommets d'un polygone ; le programmeur n'a besoin que d'écrire le code qui doit être dans la boucle intérieure, pour ainsi dire. Une API avec ce genre de parallélisme, qui fonctionne sur des CPUs aussi bien que sur des GPUs rempli un vide important.

Ecrire pour OpenCL est aussi une assurance sur le futur pour le code concernant les tâches ou les données en parallèle. Tout comme un même code OpenGL sera de plus en plus rapide à mesure que des GPUs plus puissants sont disponibles, le code OpenCL est appelé à se comporter mieux quand les CPUs et les GPUs iront plus vite. Le niveau supplémentaire d'abstraction que fournit OpenCL rend cela possible. Par exemple, bien qu'un code vectoriel écrit il y a plusieurs années en utilisant MMX aille plus vite à mesure que les vitesses d'horloge du CPU augmentent, une amélioration de performance plus significative requiert probablement le portage du code vers l'un des nouveaux jeux d'instructions SSE.

A mesure que de nouveaux jeux d'instructions vectoriels et des matériels de calcul parallèles deviennent disponibles, Apple va mettre à jour les implémentations OpenCL pour en tirer avantage, tout comme les fabricants de cartes vidéo ou les vendeurs d'OS mettront à jour leurs pilotes OpenGL, pour profiter de GPUs plus rapides. Pendant ce temps, le code de l'application demeure inchangé pour les développeurs. Il n'est même pas besoin de recompiler.

## Qu'il y ait ici des dragons (et des trains)

Vous vous demandez peut-être comment le même code compilé peut en arriver à utiliser SS2 sur une machine, et SS4 sur une autre, ou un GPU NVIDIA sur une machine, et un GPU ATI sur une autre ? Pour le faire, vous devez traduire le code OpenCL indépendant du composant dans le jeu d'instructions du composant cible, *à l'exécution*. Quand il tourne sur un GPU, OpenCL doit aussi envoyer les données et le nouveau code résultant de la traduction à la carte vidéo, et récupérer les résultats à la fin. S'il tourne sur un CPU, OpenCL doit s'organiser pour un niveau requis de parallélisme en créant et en répartissant les threads convenablement aux cœurs disponibles.

Et bien, le croiriez-vous ? Apple dispose justement de deux technologies qui résolvent exactement ces problèmes.

Vous voulez compiler du code "à la volée" et l'envoyer au composant de calcul ? C'est ce pour quoi LLVM a été fait, et bien sûr ce qu'Apple a fait avec, dans Léopard, bien que à une échelle plus limitée. OpenCL est une extension naturelle de ce travail. LLVM permet à Apple d'écrire un seul générateur de code pour chaque jeu d'instructions cible, et concentre tous ses efforts sur un seul optimiseur de code indépendant du composant. Il n'y a plus aucun besoin de dupliquer ces tâches, en utilisant un compilateur pour créer une application statique exécutable, et en improviser un autre pour la compilation à la volée.

Oh, et par la même occasion, vous vous rappelez Core Image ? Voilà une autre API qui a besoin de se compiler à la volée, et être envoyée pour l'exécution sur les matériels en parallèle comme des GPUs et des CPUs multi-cœurs. Dans Léopard des neiges Core Image a été réécrit en utilisant OpenCL, et cela fournit une confortable amélioration de performances de 25 % au total.

Pour gérer le parallélisme des tâches et fournir des threads, OpenCL est construit au dessus de Grand Central Dispatch. C'est un chose si normale, qu'il semble un peu surprenant que l'API OpenCL n'utilise pas les blocs. Je pense qu'Apple a décidé qu'il ne fallait pas forcer la chance quand il s'agit de voir ses technologies maison adoptées pas d'autres vendeurs. Cette décision semble se révéler payante, puisque AMD a sa propre implantation de OpenCL en développement.

## Le sommet de la pyramide

Bien que les technologies sous-jacentes, Clang, les blocs, et Grand Central Dispatch seront sans aucun doute plus largement utilisées par les développeurs, OpenCL représente le point culminant de cet ensemble technologique dans Léopard des neiges. C'est une médaille d'or dans la conception du logiciel : créer une nouvelle API publique, en la construisant au sommet d'autres APIs publiques de bas niveau, tout aussi bien conçues et implémentées.

Une abstraction unifiée pour une accumulation hétérogène et toujours plus importante de silicium destiné au calcul parallèle dans les PCs était une impérieuse nécessité. On a observé une population croissante de cœurs de CPUs puissants, mais en nombre, ils sont plusieurs ordres de grandeur plus faibles que les centaines d'unités de calcul des GPUs modernes. D'un autre côté, les GPUs ont encore du travail à faire pour rivaliser avec la puissance et la flexibilité des cœurs de CPU à maturité. Mais même compte tenu de ces différences, écrire du code exclusivement pour l'un ou l'autre de ces mondes équivaut à laisser de l'argent sur la table.

Avec OpenCL sous la main, il n'est plus besoin de mettre tous ses œufs dans le même panier de silicium. Et avec l'arrivée des hybrides CPU/GPUs comme le Larabee d'Intel qui utilise des moteurs de calcul du calibre des CPUs, mais en nombre beaucoup plus élevé, OpenCL peut se révéler encore plus important dans les années qui viennent.

### La moisson de transistors

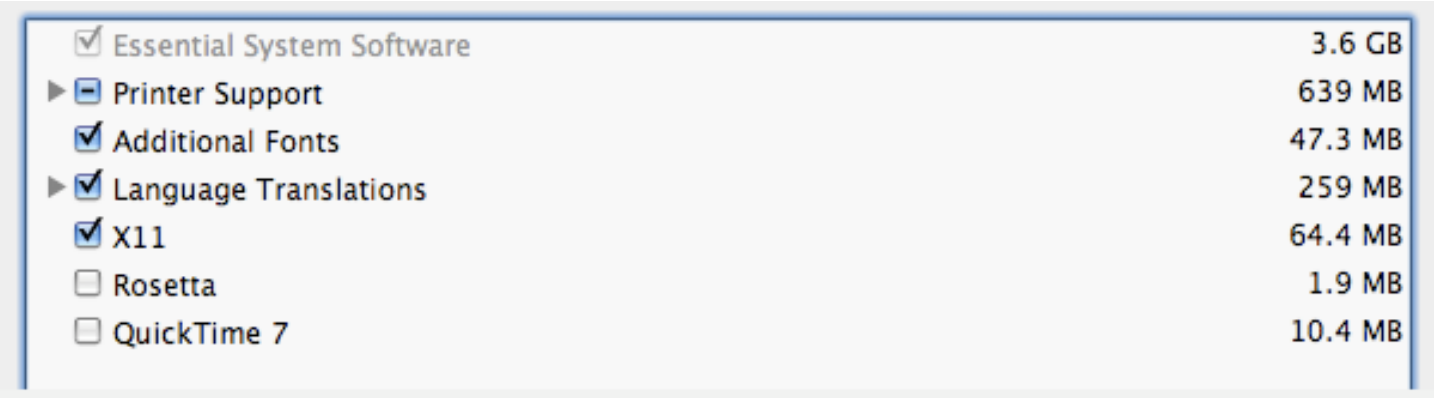
Toutes ensemble, les techniques de simultanéité introduites dans Léopard des neiges représentent le plus important progrès dans le développement du logiciel asynchrone et parallèle de toutes les livraisons de Mac OS X, et peut-être de tous les systèmes d'exploitation jamais délivrés. Il est sans doute difficile, pour les utilisateurs finaux, d'être enthousiasmé par des technologies fondamentales comme Grand Central Dispatch, OpenCL, sans parler des compilateurs, et des propriétés des langages de programmation, Mais c'est sur ces fondations que les développeurs vont pouvoir créer des ensembles logiciels toujours plus impressionnants. Et si ces applications surpassent leurs prédécesseurs synchrones et sériels, ce sera parce qu'elles s'appuient sur les épaules de géants.

## QuickTime Player

*Nouvelle icône de QuickTime Player*

Il y a eu une certaine confusion autour de QuickTime dans Léopard des neiges. La page sur QuickTime X explique ce que vous avez besoin de savoir sur le présent et le futur de QuickTime en tant que technologie et API. Mais quelques unes des décisions d'Apple, et le terme très surchargé de "QuickTime" dans l'esprit des consommateurs ont quelque peu brouillé l'image.

La première occasion de se gratter la tête intervient au cours de l'installation. S'il vous arrive de cliquer sur le bouton "Personnaliser", vous voyez les options suivantes :




*QuickTime 7 est-il une installation optionnelle ?*

Nous avons déjà abordé la question de l'installation optionnelle de Rosetta mais QuickTime 7 l'est-il aussi ? QuickTime n'est-il pas sévèrement handicapé sans QuickTime 7 ? Pourquoi diable cela devrait-il être une installation optionnelle ?

Bon, pas besoin de paniquer. L'item que vous voyez dans l'installeur aurait dû être écrit "QuickTime Player 7". QuickTime 7, le framework vieux mais très doué dont nous avons parlé précédemment *est* installé par défaut dans Léopard des neiges -en fait, c'est obligatoire-. Mais l'application QuickTime Player, celle avec une icône bleue en forme de Q, celle que beaucoup d'utilisateurs occasionnels considèrent comme *étant* QuickTime, celle-là a été remplacée par une version adaptée à QuickTime X, avec une nouvelle icône (en haut à droite) un peu enrobée.

Le nouveau player est une grosse différence avec le précédent. Clairement, il met à profit QuickTime X pour une visualisation vidéo plus efficace, mais l'interface utilisateur est aussi entièrement nouvelle. La bordure grise est partie, et les contrôles à base de boutons du vieux QuickTime Player ont aussi été remplacés par une fenêtre sans cadre, avec une barre de titre noire, et un ensemble de contrôles à glissière.

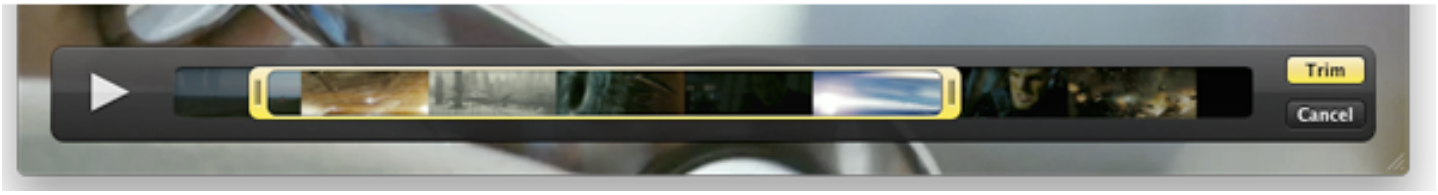

*Le nouveau QuickTime Player... courageusement sur les traces de Nice Player*

C'est comme une combinaison de la fenêtre de l'excellente application Nice Player et des contrôles en plein écran du vieux QuickTime Player. Je suis un peu embarrassé par deux choses. La première, les coins très légèrement arrondis me semblent une mauvaise idée. Je suis donc obligé de renoncer à cette douzaine de pixels en moins ? Nice Player fait mieux avec ses coins rectangulaires.

La seconde, les contrôles flottants cachent l'image. Et si je veux scruter l'image là où se trouve le contrôle ? Oui, vous pouvez déplacer les contrôles, mais comment faire si je cherche quelque chose à un endroit inconnu dans l'image ? Et puis, la barre de titre obscurcit une ligne entière en haut de l'image, et elle ne peut pas être enlevée. J'apprécie la compacité de cette approche, mais ce serait bien que la superposition de la barre de titre puisse être éliminée, et que les contrôles puissent être déplacés entièrement hors de l'image, et placés en bas, ou autrement.

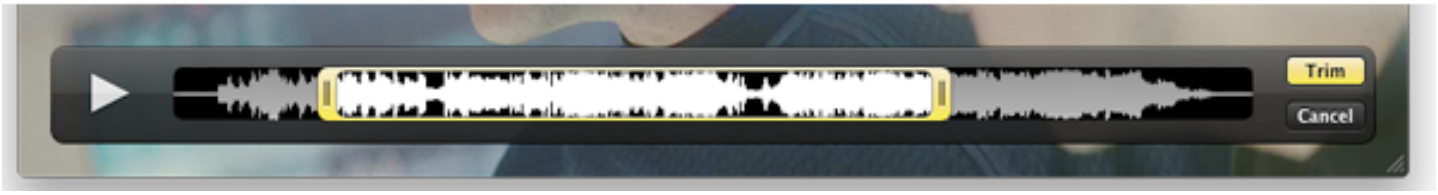
(Une grâce pour les gens qui partagent ma tendance obsessionnelle au désordre : si vous déplacez les contrôles flottants, ils ne se souviennent pas de leur position la prochaine fois que vous ouvrez la séquence. Pourquoi est-ce une grâce ? Parce que, si ça fonctionnait autrement, nous passerions tous beaucoup trop de temps à grommeler sur notre incapacité à rétablir le contrôleur exactement en position centrée. Regrettable, mais vrai).

Le nouveau QuickTime Player présente une interface pour faire des coupures de séquences qui ressemble beaucoup à iMovie (à moins que ce ne soit à l'iPhone ?). Des extraits immobiles de séquence sont disposés côte à côte pour constituer une base de temps, avec des arrêts ajustables à chaque extrémité.




*Les coupures dans le nouveau QuikTimePlayer*

Si on appuie sur la touche option, la base de temps est remplacée par le spectre audio.




*Coupure avec le spectre audio*

Dans les deux cas (vidéo et audio), j'admire, mais je me demande quelle peut être l'utilité de ces barres fantaisies. Le spectre audio est tout petit et compressé, et l'espace horizontal limité de la fenêtre d'affichage fait qu'on ne peut voir que quelques images de la séquence vidéo. D'autre part, s'il y a une quelconque possibilité à opérer des ajustements fins avec autre chose que des mouvements extrêmement précis de la souris, je n'ai pas pu la trouver. Rien à voir avec Final Cut Pro.

QuickTime Player a appris un autre tour : l'enregistrement de l'écran. Les contrôles sont limités, si bien que des utilisateurs plus exigeants ont encore besoin d'un enregistreur plus élaboré. Mais QuickTime player peut faire le travail.


*L'enregistrement d'écran avec QuickTimePlayer*

Il y a aussi une option audio, avec une collection de réglages aussi simplifiée.


*Enregistrement audio*

Finalement, le nouveau QuickTime Player a la possibilité de télécharger une séquence vidéo directement depuis YouTube ou MobileMe, en envoyer une par courriel, ou l'ajouter à votre bibliothèque iTunes. Les options d'exportation sont aussi très simplifiées, avec des options prédéfinies pour l'iPhone/iPod, l'Apple TV, et HD480p et 720p.

Malheureusement, la liste des choses que vous ne pouvez pas faire avec QuickTime Player est plutôt longue. Vous ne pouvez pas Couper/Copier/Coller une portion donnée de la séquence (les coupures n'affectent que les extrémités) ; vous ne pouvez pas extraire ou effacer des pistes individuellement, ou superposer une piste à une autre (et optionnellement, mettre à l'échelle) ; vous ne pouvez pas exporter une séquence en choisissant parmi l'ensemble des codecs audio ou vidéo disponibles dans QuickTime. Tout cela était possible avec l'ancien QuickTime Player -à condition de payer la licence QuickTime Pro de $30. Dans le passé, j'ai qualifié cette option comme "criminellement stupide", mais les possibilités qu'elle fournissait à QuickTime Player étaient vraiment utiles.

Il est tentant d'attribuer leur absence dans le nouveau QuickTime Player aux limitations précédemment évoquées de QuickTime X. Mais le nouveau QuickTime Player est basé sur QTKit, qui sert de façade à la fois pour QuickTime X et QuickTime 7. Et il présente quelques possibilités d'édition limitées (comme les coupures), et quelques possibilités auparavant réservées à QuickTime Pro, comme la lecture en plein écran. De plus, le nouveau QuickTime Player peut jouer des séquences de greffons de tierces parties, une caractéristique clairement associée à QuickTime Pro.

Si bien que Léopard des neiges vous réserve une très agréable surprise si vous installez l'option QuickTime 7. En le faisant, j'ai obtenu le vieux QuickTime Player, installé de façon un peu infamante dans le dossier "Utilitaires", avec toutes ses possibilités "Pro" définitivement déverrouillées. Eh oui, la tyrannie de QuickTime Pro semble terminée...


*QuickTime Pro : gratuit pour tout le monde ?*

... mais peut-être le mot essentiel est-il "semble", parce que QuickTime Player 7 n'a pas toutes les caractéristiques "pro" déverrouillées pour tout le monde. J'ai installé Léopard des neiges sur un disque vide, et QuickTime 7 n'a *pas* été installé automatiquement (c'est fait quand l'installeur détecte une licence de QuickTimePro sur le disque cible). Après avoir démarré sur mon nouveau volume Léopard des neiges, j'ai installé manuellement le composant optionnel "QuickTime 7", en utilisant le disque d'installation de Léopard des neiges.

Pour moi, le résultat a été l'application QuickTime Player 7, avec toutes les caractéristiques pro, mais pas d'information d'enregistrement QuickTime Pro visible. Mais j'avais une licence de QuickTime Pro sur l'un des disques montés. Apparemment, l'installeur l'a détecté, et m'a installé une application QuickTime Player 7 déverrouillée, bien que le volume de démarrage n'ai jamais contenu une licence de QuickTime Pro.

# Le Dock

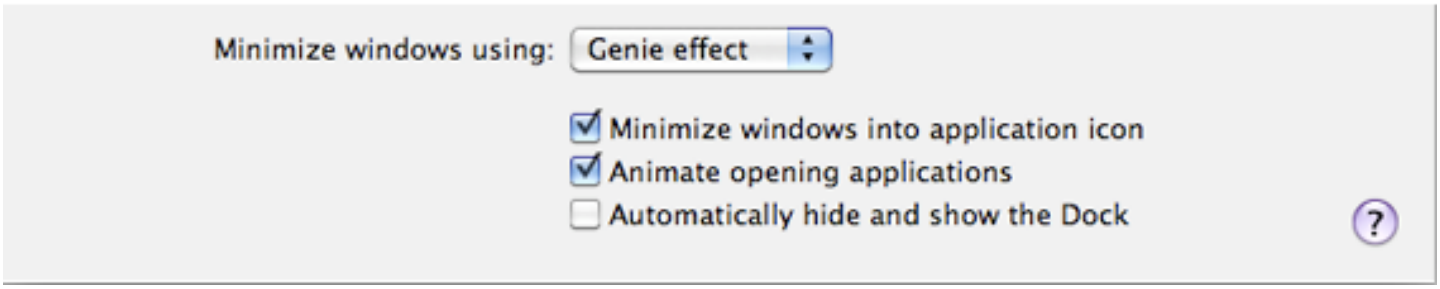
La nouvelle présentation de quelques aspects du Dock s'accompagne aussi de quelques nouvelles possibilités. En maintenant un clic sur l'icône dans le Dock d'une application qui fonctionne, on valide Exposé, mais pour les seules fenêtres qui appartiennent à l'application. Si on déplace un fichier sur l'icône d'une

application dans le Dock, et qu'on maintient un peu, on a le même résultat. Vous pouvez alors continuer à déplacer le fichier vers l'une des vignettes affichées par Exposé, maintenir un peu pour ramener la fenêtre sur le devant, et lâcher votre fichier dedans. C'est une technique très pratique quand vous en avez pris l'habitude.

L'affichage d'Exposé a changé lui aussi. Maintenant, les fenêtres minimisées sont affichées en petite taille en dessous d'une ligne fine en bas de l'écran


*Représentation du Dock avec la nouvelle position des fenêtres minimisées*

Dans l'image de l'écran, ci-dessus, vous pouvez remarquer qu'aucune des fenêtres minimisées n'apparaît dans le Dock, grâce à une nouvelle addition : la possibilité de minimiser les fenêtre "dans" l'icône de l'application. Ce réglage est localisé dans les préférences du Dock.




*Nouvelles préférences du Dock : minimisation des fenêtres dans l'icône de l'application*

*Les fenêtres minimisées sont marquées par un carreau*

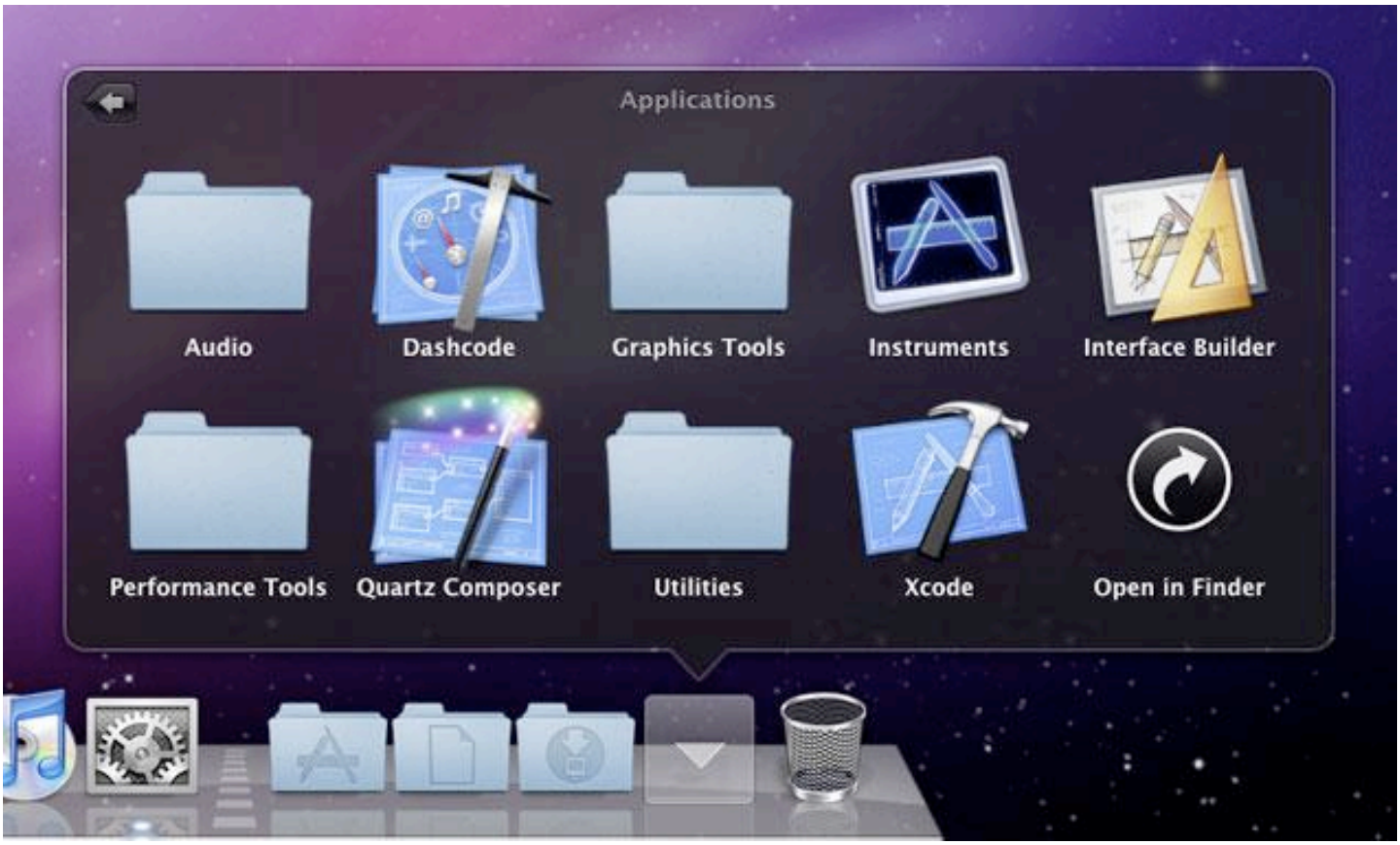
Quand cette option est sélectionnée, les fenêtres se glissent derrière l'icône de leur application parente, puis disparaissent. Pour les récupérer, on peut faire un clic droit sur l'icône de l'application (figure à droite), ou activer Exposé. La vue matricielle des dossiers du Dock contient maintenant une barre de défilement quand il y a trop d'items à afficher convenablement. Cliquer sur l'icône d'un dossier dans la grille matricielle affiche le contenu de ce dossier dans la grille, et permet de descendre dans la hiérarchie des dossiers pour trouver un item profondément enfoui. Un petit bouton de navigation en arrière apparaît dans ce cas.

Ce sont des comportements qui sont tous utiles, et apportent un plus, si on considère le discours "0 nouvelle possibilité" associé à Léopard des neiges. Mais la nature même du dock est inchangée. Les utilisateurs qui voudraient un affichage minimisé plus souple ou plus puissant pour lancer/afficher ou organiser les fenêtres d'une application doivent encore, ou bien sacrifier certaines possibilités (les icônes du Dock ou la notification rebondissante), ou continuer à utiliser le Dock avec une application tierce.

L'option consistant à conserver des fenêtres minimisées pour éviter de désorganiser le Dock, était attendue depuis longtemps. Mais mon enthousiasme est tempéré par la frustration associée à l'impossibilité de cliquer sur un fichier dans le Dock, et d'obtenir son ouverture dans le Finder, tout en conservant la possibilité de déposer des items dans le dossier. C'était le comportement par défaut des dossiers dans le

Dock pendant les six premières années de Mac OS X, mais cela a changé avec Léopard. Léopard des neiges n'améliore pas les choses.

Déposer un alias dans le Dock permet d'ouvrir à l'aide d'un simple clic, mais des items ne peuvent pas être déposés dans l'alias d'un dossier présent dans le Dock pour une raison inexplicable. (Radar5775786, a fermé en Mars 2008 avec cette explication laconique, "pas actuellement supporté"). Pire, déplacer un item vers l'alias d'un dossier dans le Dock réagit comme si ça marchait (l'icône s'illumine), mais quand on relâche, l'item qu'on a déplacé revient à sa position initiale. J'espérais vraiment que cela serait corrigé dans Léopard des neiges. Pas de chance.




*Dans la vue en grille du Dock, navigation avec le bouton Back*

# Le Finder

L'une des premières fuites de copies d'écran de Léopard des neiges contenait une fenêtre d'aspect anodin "Get Info..." pour le Finder, vraisemblablement pour montrer que le numéro de version avait été mis à jour à 10.6. Le morceau

d'information le plus intéressant qu'elle révélait était que le Finder de Léopard des neiges était une application 64 bits.

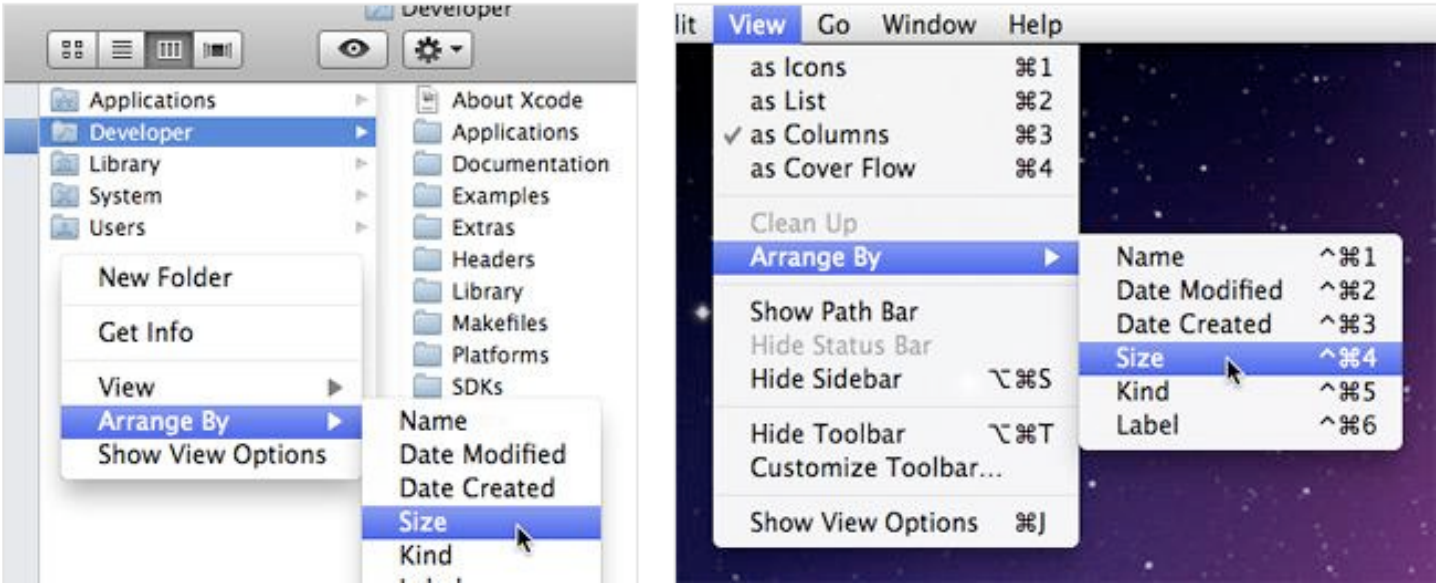
Le Finder de Mac OS X a commencé son existence comme une application maison pour l'API Carbon permettant une compatibilité rétrograde pour Mac OS X. Avec le temps, le Finder a été une cible fréquente de désagrément et de mépris. Ces mauvais sentiments se sont souvent répandus dans un débat parallèle sur la supériorité des API : Carbon contre Cocoa.

"Le Finder est mauvais parce que c'est une application Carbon. Ce dont nous avons besoin, c'est d'un Finder Cocoa. Cela va sûrement résoudre tous nos ennuis." Et bien Léopard des neiges contient un Finder 64 bits, et comme nous le savons tous, Carbon n'a pas été porté en 64 bits. Et voilà ! Un Finder Cocao dans Léopard des neiges. (Un peu plus sur les ennuis dans un instant)

La conversion à Cocoa a suivi la formule de Léopard des neiges : pas de nouvelles possibilités... à l'exception d'une ou deux peut-être. Si bien que le "nouveau" Finder Cocoa ressemble à, et fonctionne presque exactement comme, le vieux Finder sous Carbon. L'indicateur le plus net de sa filiation Cocoa est l'utilisation étendue de transitions avec Core Animation. Par exemple, quand une fenêtre du Finder accomplit sa transformation schizophrène depuis une fenêtre du browser décorée d'une barre latérale en une forme de décor minimum, cela ne se fait plus en un coup d'œil. Au lieu de cela, la barre latérale se retire et s'estompe, la barre d'outil se rétrécit, et tout se ratatine pour prendre sa nouvelle forme.

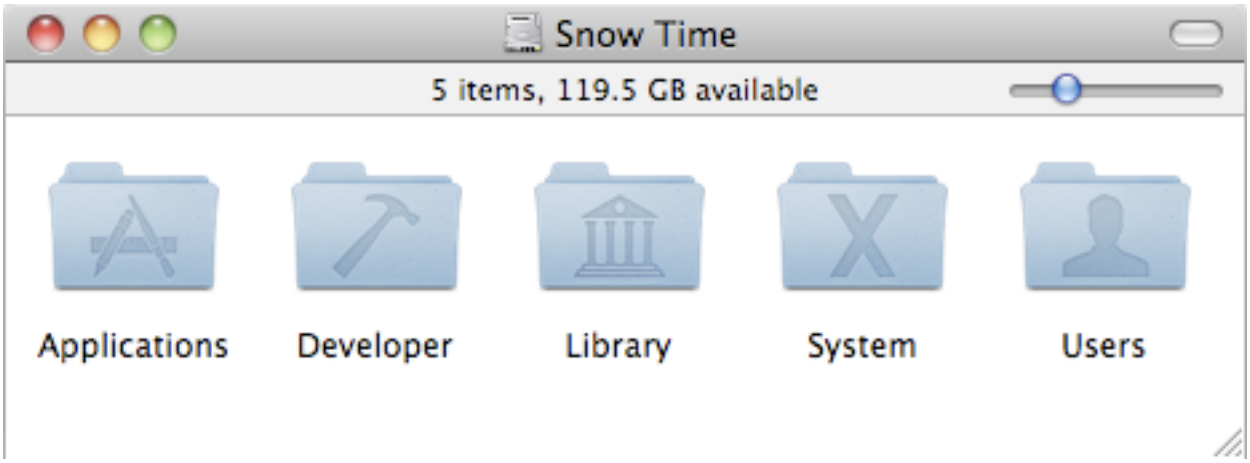
Bien qu'elles passent les bornes dans quelques cas, les transitions de Core Animation font que l'application semble plus finie, eh oui, "plus Cocoa". Et vraisemblablement, l'utilisation de Cocoa a rendu si facile l'addition de nouvelles caractéristiques que les développeurs n'ont pas pu résister, et en ont jeté quelques unes dans la corbeille.

La demande numéro 1 venue des utilisateurs intensifs de la présentation en colonnes a finalement été implémentée : des colonnes qu'on peut trier. L'ordre du tri s'applique à toutes les colonnes à la fois, ce qui n'est pas aussi bien qu'un tri par colonne, mais c'est bien mieux que rien du tout. L'ordre du tri peut être défini à l'aide d'une commande du menu (chacune a un raccourci clavier) ou par un click droit sur un espace inutilisé de la colonne qui permet de choisir dans le menu contextuel qui s'en suit.




*Tri de la présentation en colonne par le menu contextuel (à gauche)*
*Menu de tri de la présentation en colonne (à droite)*

Même la présentation par icônes, plus insignifiante, a été améliorée dans Léopard des neiges. Chaque fenêtre d'icônes inclut une petite glissière pour contrôler la taille des icônes.

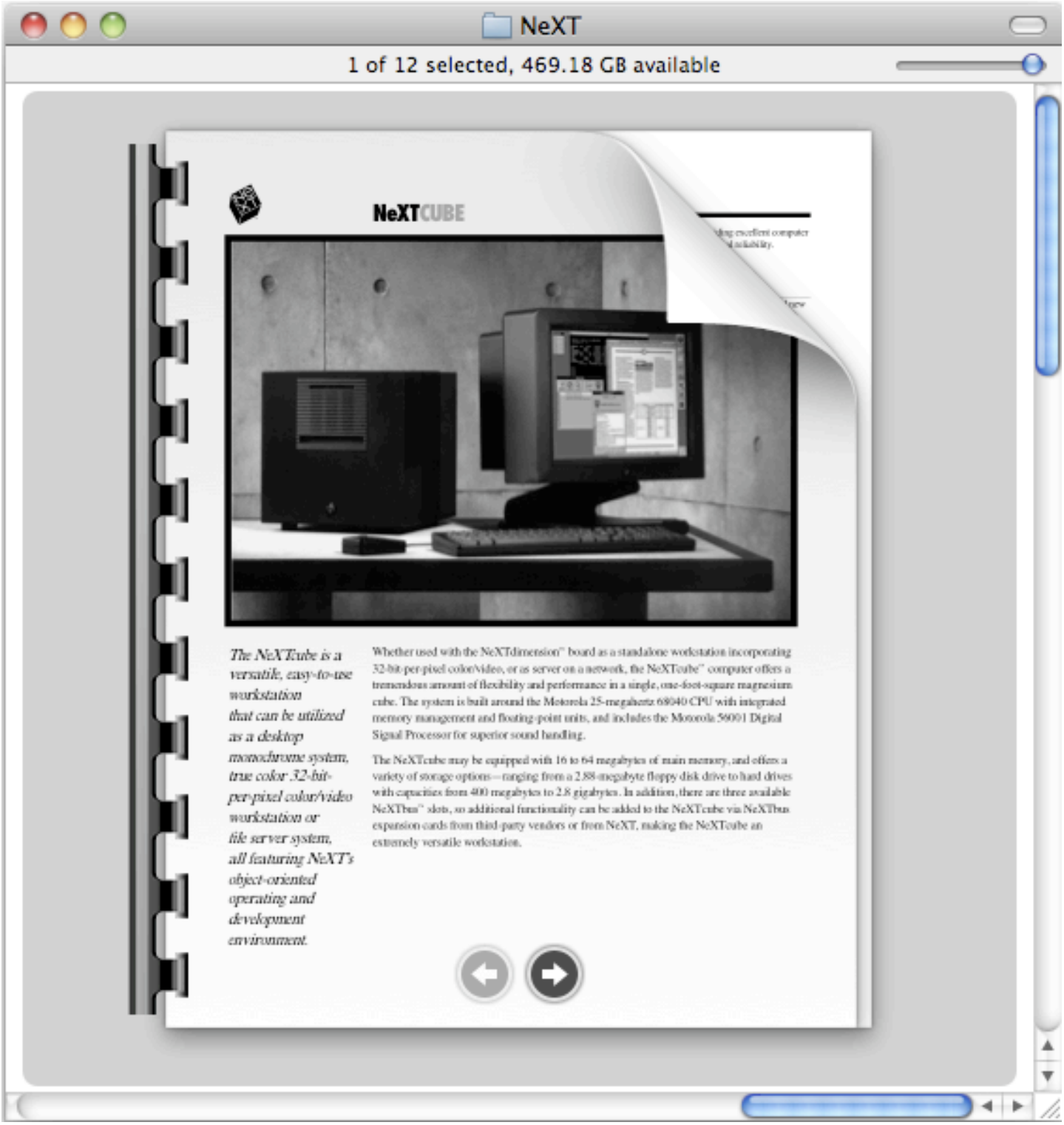



*La vue par icônes du Finder, avec une glissière*

Ça peut sembler un peu bizarre -à quelle fréquence les gens changent-ils la taille des icônes ?- mais cela a beaucoup plus d'intérêt dans le contexte de la pré-visualisation d'images dans le Finder. Ce mode d'utilisation est rendu encore plus pertinent avec la récente extension à un maximum de 512 x 512 pixels de la taille des icônes.

La présentation des icônes a été améliorée pour mieux se rapprocher des possibilités disponibles dans QuickLook. Mettez tout cela ensemble, et vous pouvez sans à coups zoomer sur une petite icône PDF, par exemple, jusqu'à une vue haute fidélité impressionnante montrée ci-dessous, avec la possibilité de tourner la

page. Une pression sur la barre d'espace, et vous obtenez une vue QuickLook encore plus grande et plus adaptable. C'est une expérience réellement douce.




*Des icônes de taille : 512 x 512 pixel pour la visualisation PDF multi-page*

Les aperçus QuickTime ont été améliorés de la même façon. Si vous zoomez sur l'icône, il se transforme en un minuscule movie player (joueur de séquences), agrémenté d'un indicateur circulaire de progression . En supposant que les utilisateurs veuillent bien affronter les bizarreries des réglages d'affichage du

Finder, jusqu'à réussir à obtenir une vue d'icône qui colle à une fenêtre, ce qui est le plus utile, je pense que cette étrange petite glissière peut se révéler bien utile.


*Prévisualisation de QuickTime dasn le Finder*

La vue en liste a aussi quelques améliorations - accidentelles, incidentes, ou autre. La zone qui permet de saisir chaque item d'une vue liste s'étend maintenant à toute la ligne. Dans Léopard, bien que la ligne ait été entièrement illuminée, seul le nom de fichier ou l'icône permettait de la saisir. Essayer de déplacer ailleurs ne faisait

qu'étendre la sélection à d'autres items de la vue liste à mesure que le curseur se déplaçait. Je ne sais pas si cette modification de comportement est intentionnelle ou seulement une conséquence inattendue du dispositif de contrôle sous-jacent utilisé pour la vue en liste dans le nouveau Finder sous Cocoa. Mais de toute façon, chapeau.

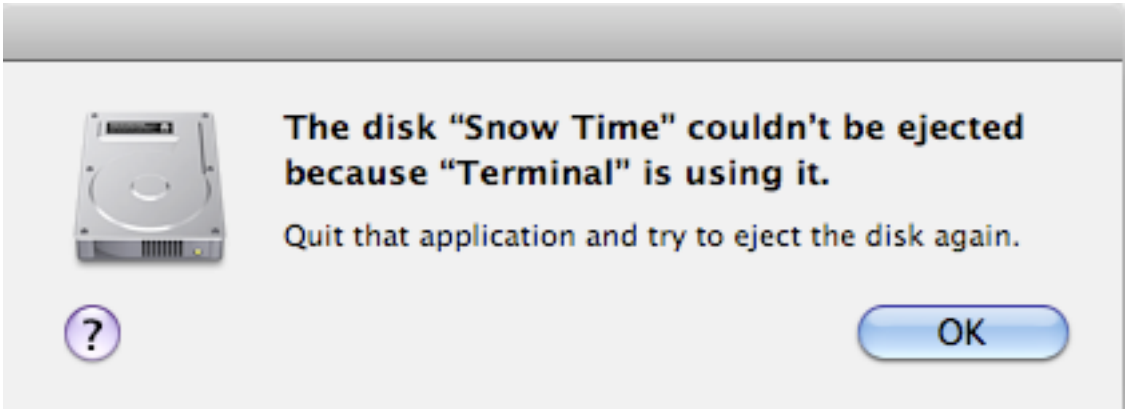
Double-cliquer sur la ligne de séparation entre les entêtes de colonnes dans la vue Liste va ajuster la largeur de colonne. Pour la plupart des colonnes, cela correspond à élargir ou rétrécir la colonne pour qu'elle s'ajuste à la taille de l'item le plus long qu'elle contient. Les dates se rétrécissent jusqu'à afficher des formats moins verbeux. Cela était sensé fonctionner de façon intermittente sous Léopard. Mais, que Cocoa permette d'obtenir cette facilité pour la première fois, ou qu'il la fasse maintenant fonctionner correctement, c'est un changement bénéfique.

Les recherches utilisant la vue en colonnes du Finder ont été grandement facilitées par l'implémentation d'une de ces petites choses que beaucoup d'utilisateurs ont réclamé depuis des années. Il y a maintenant un réglage de préférences pour définir le champ par défaut de la recherche dans la barre d'outils de la fenêtre du Finder. Ça vous va ?


*Enfin, l'espace de recherche est configurable dans le Finder (avec les options possibles)*

Dans le même ordre d'idées, il y a des améliorations attendues depuis longtemps, qui vont efficacement contribuer à faire apparaître l'environnement de bureau plus robuste. Un bon exemple est la meilleure gestion de la fenêtre d'erreur tant redoutée "impossible d'éjecter, le disque est utilisé". La question évidente que se pose

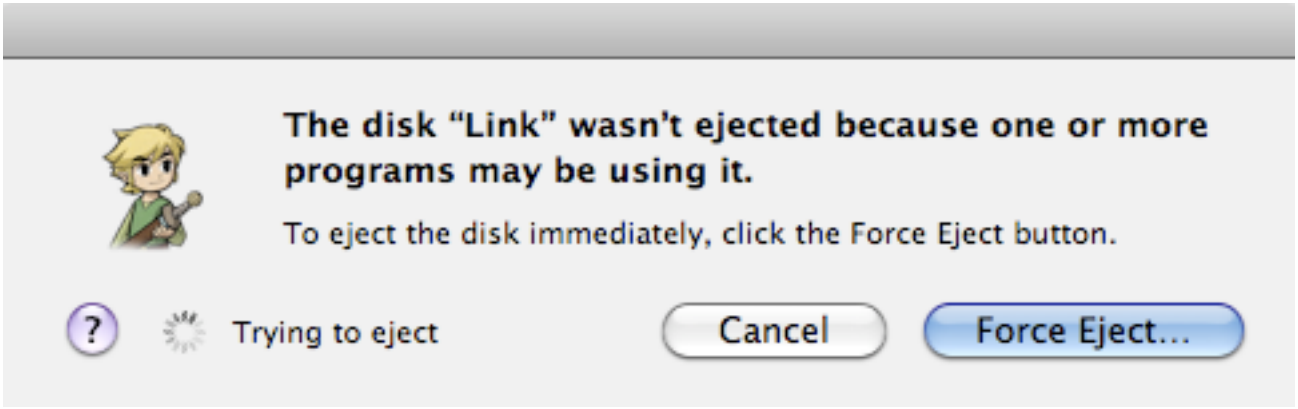
l'utilisateur, c'est "d'accord, mais qui l'utilise ?" Léopard des neiges fournit maintenant cette information.




*Plus besoin de deviner...*

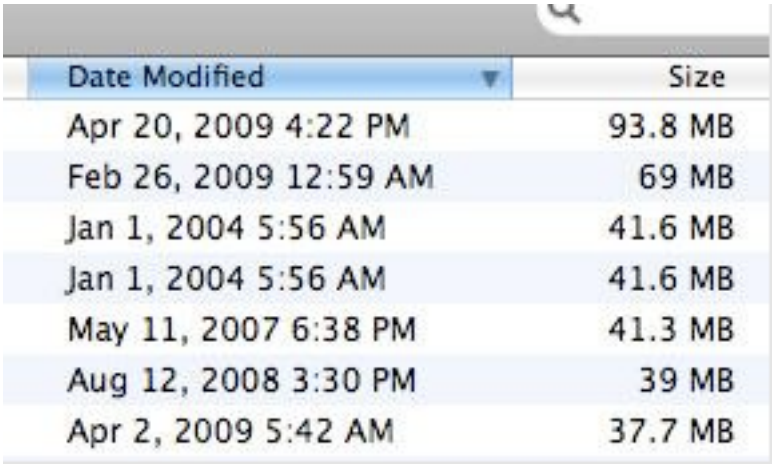
(Et puis, Mac OS X refuse d'éjecter un disque si le répertoire courant dans une ligne de commande est situé sur ce disque. Une bonne chose normalement, mais qui peut être ennuyeuse).

Une autre réponse possible de l'utilisateur à l'erreur du disque en cours d'utilisation est "je m'en fous, je suis pressé, force l'éjection !". C'est maintenant une option.




*Il est possible de forcer l'éjection du disque*

Oui, mais pourquoi l'application gênante apparaît-elle dans une fenêtre de dialogue, et l'option pour forces l'éjection dans une autre, mais qu'aucune ne présente les deux choix possibles ? C'est un mystère pour moi, mais je suppose que c'est associé à l'information dont dispose le Finder sur le comportement du disque. (Comme cela a toujours été le cas, la commande *lsof* est disponible si vous voulez le savoir à la manière ancienne)




*Hum...*

Alors, le nouveau Finder Cocoa a-t-il finalement banni toutes ces bogues gênantes du bon vieux temps de Carbon ? Pas tout à fait. C'est la version 1.0 du Finder sous Cocoa, et elle récupère les bogues associés à toute version 1.0. En voilà une, découverte par Glen Aspeslagh (image à droite)

L'avez-vous vue ? Sinon, regardez bien l'ordre des dates dans la colonne "Date Modified". Eh oui, les mystères du vieux Finder n'ont pas encore été complètement éliminés.

Il y a aussi des bizarreries dans le fonctionnent de la matrice d'icônes. Dans une vue où l'affichage en icônes est actif (ou est permis temporairement par un glisser accompagné de la touche Commande), les icônes semblent terrifiés par les autres, au point de laisser une énorme distance entre elles et leurs voisines quand elles décident de se positionner sur la grille. C'est comme si le Finder vivait dans la peur fatale d'avoir à afficher un nom de fichier de 200 caractères qui pourrait se superposer à celui de son voisin.

La pire incarnation de ce comportement se produit sur la bordure droite de l'écran quand des volumes montés sont présents sur le bureau (Remarquons incidemment que ce n'est pas le comportement par défaut ; si vous voulez voir ces disques sur le bureau, vous devez autoriser cette préférence dans le Finder). Quand je monte un nouveau disque sur le bureau, je suis souvent surpris de l'endroit où il apparaît. S'il y a d'autres icônes proches du bord droit, le disque refuse de s'y placer. Là encore, le Finder ne cherche pas à éviter une superposition des noms ou des icônes. Il semble seulement éviter *la possibilité* d'une superposition, qui pourrait intervenir dans le futur. Idiot.

### Conclusion sur le Finder

Au total, le Finder de Léopard des neiges a quelques avancées significatives : le support 64bits/Cocoa, quelques possibilités nouvelles et utiles, plus de poli, et seulement quelques pas en arrière avec l'utilisation un peu abusive de l'animation, et la présence permanente de quelques bogues gênantes. Si on considère le temps qu'il a fallu au Finder sous Carbon pour rassembler l'ensemble des caractéristiques et le niveau de finition obtenus avant Léopard des neiges, c'est un exploit pour le Finder Cocoa de se placer à la hauteur, ou de surpasser son prédécesseur dès la toute première version. Je suis sûr que les combattants Carbon contre Cocoa se seraient affrontés sur cette opinion, quand Carbon a été mis sur la touche par Léopard. Mais ce fut ainsi, et le vainqueur récupère le butin.

# Exchange

Le titre-slogan de Léopard des neiges "une nouvelle possibilité" est le support de Microsoft Exchange. Cela semble, au moins partiellement, être un autre emprunt à l'iPhone, qui a disposé du support d'Exchange dès sa version 2.0, et l'a étendue dans sa version 3.0. Le support d'Exchange pour Léopard des neiges est interconnecté en particulier avec le lot attendu des applications de Mac OS X : iCal, Mail et Address Book.

La principale limitation est qu'il fonctionne seulement avec un serveur utilisant Exchange 2007 (Service Pack 1, mise à jour 4) ou plus récent. Je suis sûr que Microsoft applaudit à toute amélioration de revenu que cette décision entraîne, mais cela signifie que pour les utilisateurs qui ont encore des vieilles versions d'Exchange sur leur lieu de travail, le support d'Exchange par Léopard des neiges pourrait tout aussi bien ne pas exister.

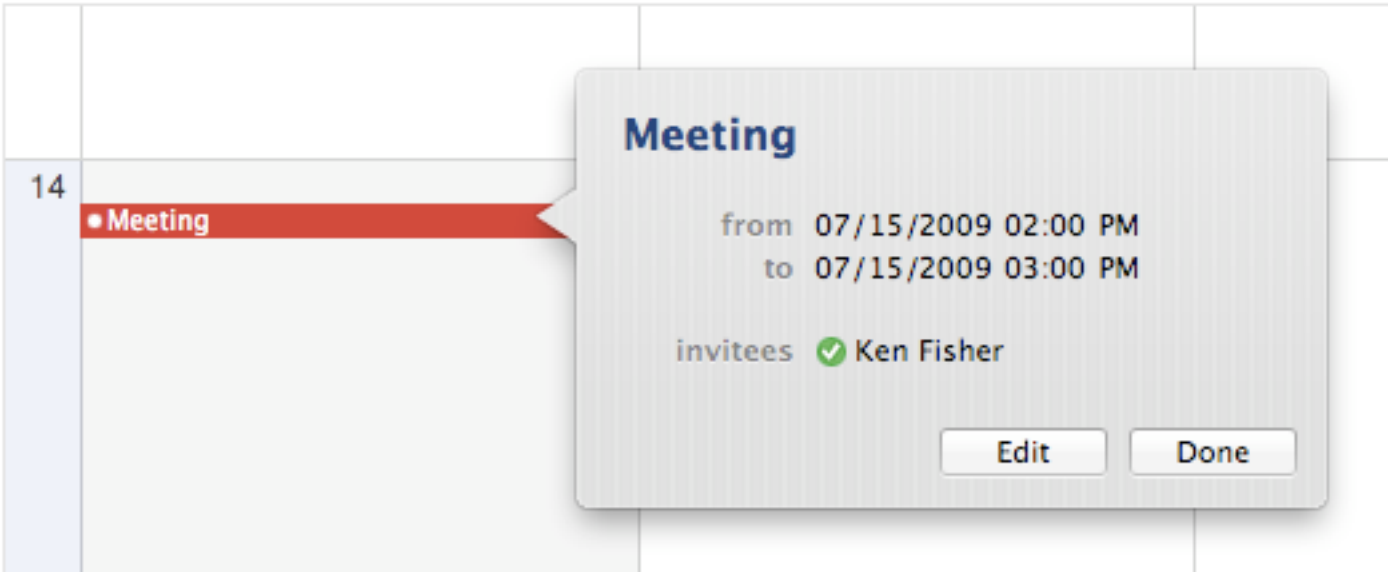
Ces utilisateurs utilisent probablement déjà le seul autre client d'Exchange disponible pour Mac OS X, Microsoft Entourage, si bien qu'ils vont rester assis tranquillement en attendant que leur direction informatique fasse la mise à jour. En mêmem temps, Microsoft donne des gages à ces utilisateurs avec la promesse - enfin- d'une version convenable de Outlook pour Mac OS X.

Au cours de mes tests que j'admets brefs, le support d'Exchange par Léopard des neiges semble marcher comme on peut s'y attendre. Il m'a fallu recourir à un spécialiste Microsoft pour mettre en route un serveur Exchange dans Ars Orbiting HQ uniquement pour les besoins de ce compte-rendu. Peu importe la façon dont il était configuré, tout ce que j'ai eu à rentrer dans mon application fut mon nom, l'adresse eMail, et le mot de passe. Et il a automatiquement trouvé tous les réglages nécessaires, et configuré iCal et Address Book à ma place.


*Les réglages d'Exchange : étonnamment faciles*

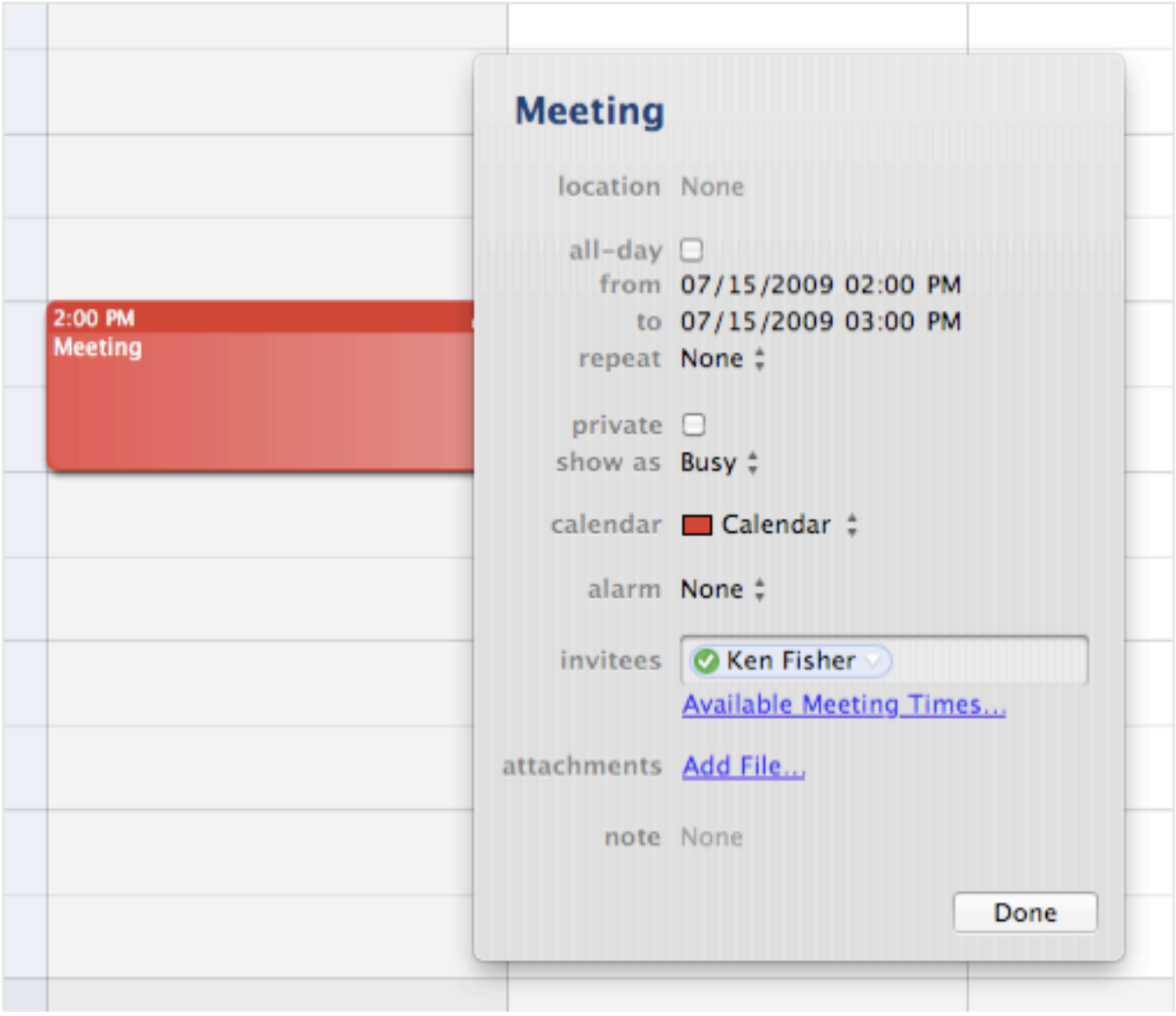
Les utilisateurs de Windows sont sans doute habitués à ce genre d'intégration avec Exchange, mais c'est la première fois que je la vois sur une plate-forme Mac, et cela inclut les nombreuses années où j'ai utilisé Entourage.

L'accès aux caractéristiques d'Exchange est à coup sûr facilité, en le limitant aux interfaces existantes pour Mail, iCal et Address Book. Si vous vous attendez à la myriade de panneaux, et de boutons d'outils rencontrés dans Outlook, vous allez être étonné(e). Par exemple, voilà la vue de détail d'une rencontre dans iCal :



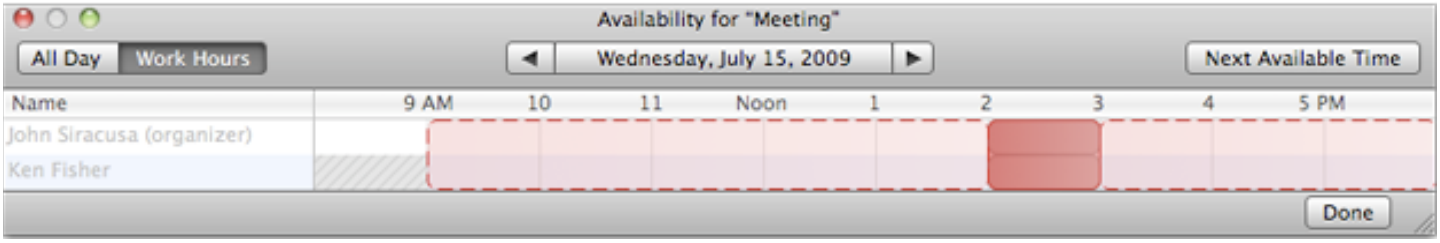

*Détail d'un événement iCal*

En cliquant le bouton "Edit" on n'en obtient pas beaucoup plus :




*L'éditeur d'événements : très simple*

La fenêtre "disponibilité" (Available Meeting Times) contient aussi le strict minimum de contrôles et d'affichage pour permettre de faire le travail.




*Test de disponibilité des réunions*

L'intégration dans Mail et Address Book est encore plus subtile, presque totalement transparente. Cela doit être interprété comme une propriété, je suppose. Mais quoi que je ne connaisse pas Exchange assez bien pour en être sûr, je ne peux pas m'affranchir de l'impression qu'il y a des ressources de Microsoft Exchange qui restent inaccessibles aux clients Mac OS X. Par exemple, comment est-ce que je réserve une "ressource" dans une réunion ? S'il y a une manière de le faire, je n'ai pas réussi à la trouver.

Néanmoins, une intégration toute prête, même élémentaire d'Exchange, fait beaucoup pour rendre Mac OS X mieux venu dans des environnements d'entreprises. Il reste à voir comment des gestionnaires des technologies de l'information (IT) pourront être convaincus de la "réalité" de l'intégration d'Exchange à Léopard des neiges. Mais j'ai tendance à penser qu'être capable d'envoyer et de recevoir du courrier, créer des invitations de rencontre et y répondre, et utiliser le carnet d'adresse général de l'entreprise suffit à tout utilisateur de Mac pour se sentir à peu près bien dans un environnement centré sur Exchange.

# Les performances

Le fait est qu'il n'y a pas beaucoup de choses à dire à propos des performances de Léopard des neiges. Des douzaines de graphiques de benchmaks amènent à la même conclusion : Léopard des neiges est plus rapide que Léopard. Pas beaucoup, au moins dans l'ensemble, mais il est plus rapide. Si on isole un sous-système particulier avec un micro-benchmark cela peut révéler des valeurs qui forcent

l'admiration, mais c'est la manière avec laquelle de petites modifications se combinent pour améliorer le contact réel avec le système qui fait vraiment la différence.
Un exemple que Apple a donné à la WWDC fut de créer une sauvegarde initiale avec Time Machine sur un réseau avec une Time Capsule. L'approche d'Apple pour optimiser cette opération fut de s'intéresser à chacun des sous-systèmes mis en jeu.
Time Machine elle-même a été dotée de la possibilité d'entrées/sorties qui peuvent se chevaucher. L'indexage Spotlight, qui intervient aussi sur les volumes de Time Machine a été identifié comme une tâche consommatrice de temps pendant la sauvegarde, et sa performance a été améliorée. Le code pour le réseau a été étendu pour tirer parti de checksums accélérées par le matériel quand c'était possible, et le code de checksum en logiciel a été manuellement retouché pour une performance maximum. Le rendement de HFS + journalisation, qui accompagne chaque mise à jour des méta-data du système de fichiers a aussi été amélioré. Pour les sauvegardes de Time Machine qui écrivent dans des images-disque plutôt que sur le système de fichiers HFS + natif, Apple a rajouté la possibilité d'un accès simultané aux images disque. La quantité de trafic réseau créée par AFP pendant les sauvegardes a aussi été réduite.
Tout cela se cumule pour un gain de vitesse d'ensemble respectable de 55 % comparé à la sauvegarde initiale avec Time Machine. Et bien sûr, les améliorations de performance des sous systèmes individuels profitent à toutes les applications qui y ont recours, pas seulement à Time Machine.
Cette approche holistique de l'amélioration de la performance n'est pas de nature à provoquer l'enthousiasme, mais à chaque fois que vous passez par une de ces fonctionnalités de Léopard des neiges qui profite de façon disproportionnée de ces sous-systèmes optimisés, c'est un vrai plaisir.
Par exemple, Léopard des neiges s'arrête et redémarre beaucoup plus vite que Léopard. Je ne parle pas du temps de démarrage, mais du temps écoulé entre la sélection de la commande "Redémarrer" ou "Eteindre" et le moment où le système s'arrête ou commence son nouveau cycle de démarrage. Léopard ne prend pas longtemps, seulement une douzaine de secondes quand il n'y a aucune application ouverte. Mais avec Léopard des neiges, c'est si rapide que j'ai souvent pensé que le système s'était écrasé au lieu de s'arrêter gentiment. (en fait, ce n'est pas trop loin de la vérité).
Les améliorations de performances fournies par les précédentes versions majeurs de Mac OS X font paraître naines celles de Léopard des neiges, mais c'est essentiellement parce que Mac OS X, dans les premières années, était énormément nonchalant et empoté. C'est facile d'obtenir une amélioration de performance énorme quand vous partez d'un niveau abyssalement bas. Le fait que Léopard des neiges obtient des améliorations mesurables et consistantes par rapport à Léopard, qui était déjà rapide, est d'autant plus remarquable.

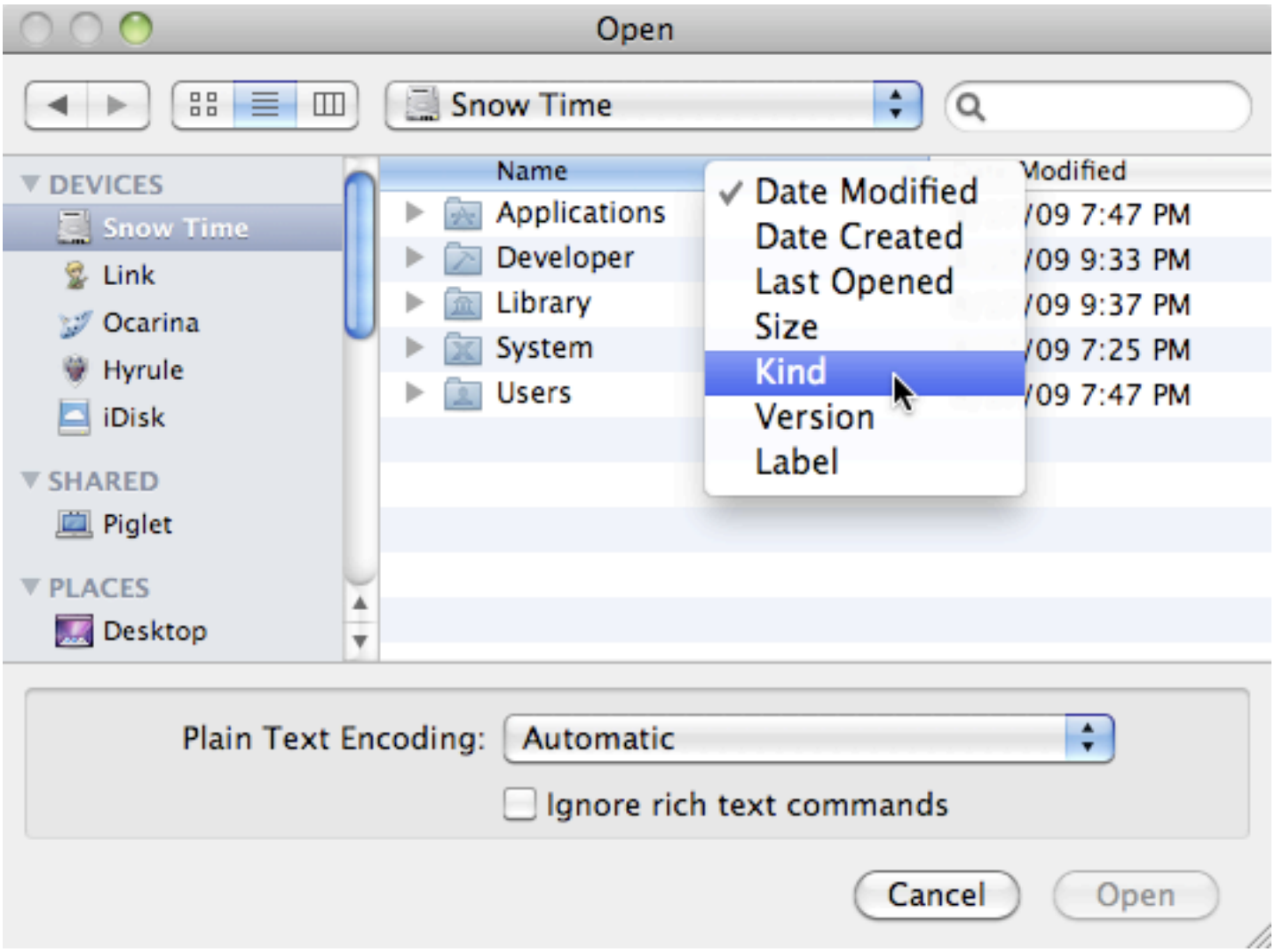
Et oui, pour la septième fois consécutive, la nouvelle version de Mac OS X est plus rapide sur un même matériel que celle qui précède. (Et pour la première fois, elle prend aussi moins de place). Que pouvez-vous demander de mieux, vraiment ? Même ce vieil épouvantail de la performance, le re-dimensionnement des fenêtres a été complètement défait. Attrapez le coin d'une fenêtre iCal pleinement remplie -le pire scénario aux temps anciens pour le re-dimensionnement de fenêtre- et agitez le aussi vite que vous voulez. Votre curseur ne sera jamais plus loin que quelques millimètres au delà de la poignée de la fenêtre ; il suit votre mouvement sauvage à la perfection. Sur la plupart des Macs, c'est aussi vrai pour Léopard. Cela ne fait que démontrer jusqu'où Mac OS X est allé sur le front des performances. Par les temps qui courent, on considère cela comme normal, ce qui est exactement ce qu'il faut faire.

# Pot pourri

Dans la section "Pot pourri" j'examine habituellement des caractéristiques mineures, largement sans lien entre elles, qui ne méritent pas un chapitre par elles-mêmes. Mais quand on en vient aux caractéristiques visibles par l'utilisateur, Léopard des neiges est du genre "tout dans le pot", si vous voyez ce que je veux dire. Apple en a même sa propre version avec une page géante de "raffinements". Je vais sans doute faire double emploi avec quelques uns, mais il y en a aussi des nouveaux ici.

## De nouvelles colonnes dans les dialogue ouvrir/sauver

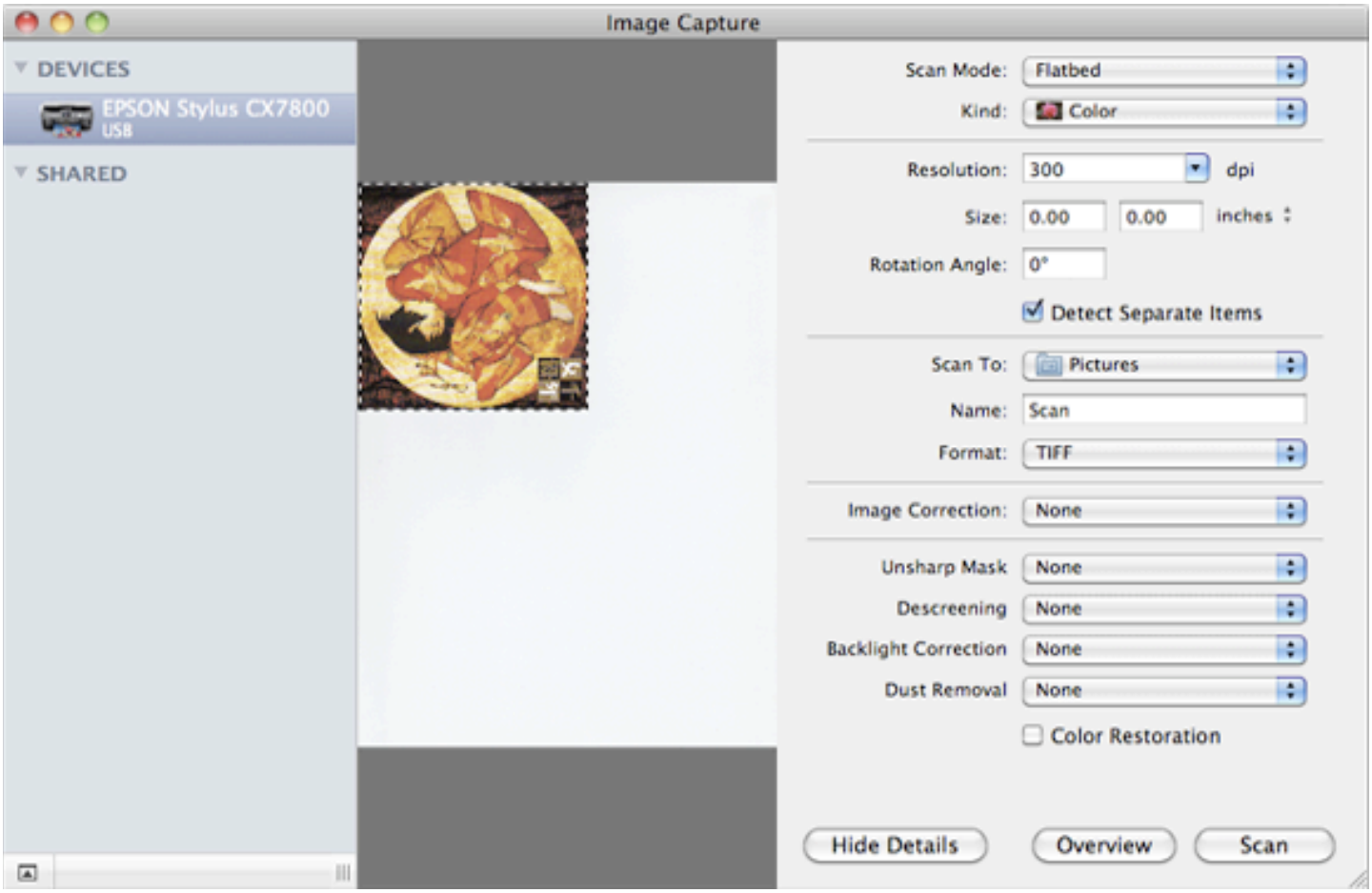
La vue en liste de la fenêtre de dialogue ouvrir/sauver propose maintenant plus que les colonnes "Nom" et "Date de modification". Un clic droit sur n'importe quelle colonne propose un choix d'autres colonnes à afficher. Il y a longtemps que j'avais besoin de cette possibilité, et je suis satisfait que quelqu'un ait finalement eu le temps de l'implanter.




*Des colonnes configurables dans les dialogues Ouvrir/Sauver*

## Support de scanner amélioré

L'application Image Capture fournie avec Mac OS X a maintenant la capacité d'utiliser un large éventail de scanners. J'ai branché mon Epson Stylus CX7800, un périphérique qui nécessitait auparavant un logiciel de tierce partie pour pouvoir scanner, et Image capture l'a détecté immédiatement.

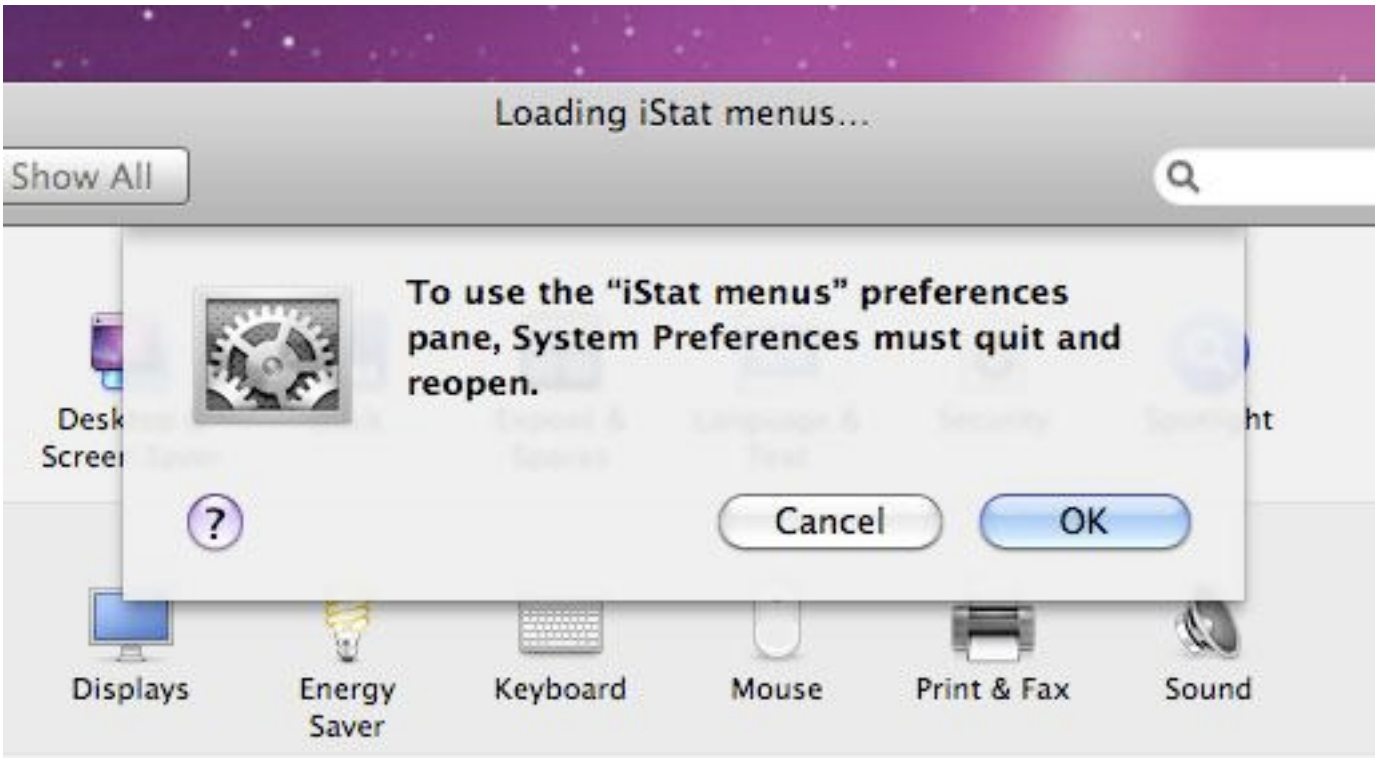



*Scanner Epson et Transfert d'image*

Image Capture est aussi une petite application de scanning pas mauvaise. Il permet une assez bonne détection automatique d'objets, y compris d'objets multiples, évitant le besoin de les récupérer manuellement. Compte tenu de la qualité parfois médiocre des pilotes de scanners et d'imprimantes pour Mac OS X, la possibilité d'utiliser une version fournie est bienvenue.

## La guerre des bits dans Préférence Système

Préférences Système, comme à peu près toutes les autres applications dans Léopard des neiges, et en 64 bits. Mais comme les applications 64 bits ne peuvent pas charger des greffons 32 bits, cela pose un problème pour la moisson existante de préférences en tierce partie. Préférences Système gère cette situation avec une certaine grâce. Au démarrage, il affichera toutes les icônes des fenêtres de préférences, 64 bits ou 32 bits. Mais si vous cliquez sur une préférence en 32 bits, vous obtenez une notification comme celle-ci :



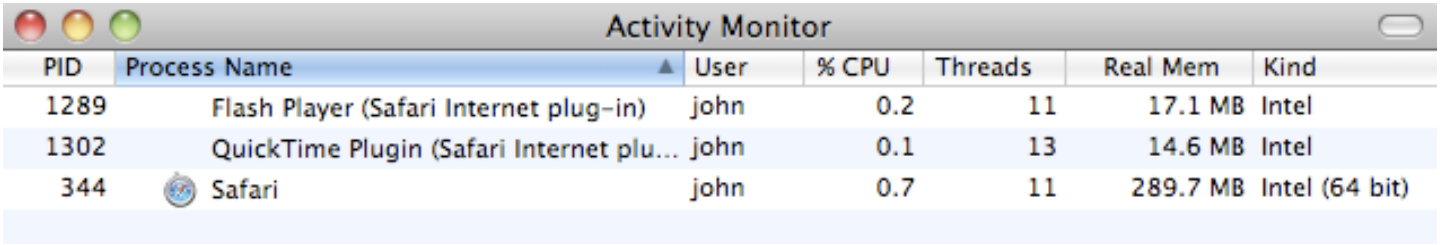

*Conflit entre Applications 64 bits et greffons 32 bits.*

Cliquez sur "OK", et Préférences système va se relancer en mode 32 bits, qui est clairement indiqué dans la barre de titre. Comme toutes les fenêtres de préférences d'Apple sont compilées à la fois pour 64 et 32 bits, les préférences système n'ont pas besoin de se relancer pendant la durée d'utilisation. Cela amène à de poser la question : pourquoi ne pas lancer tout le temps Préférences Système en 32 bits ? Je soupçonne que c'est une autre façon pour Apple d'"encourager" les développeurs à créer des binaires compatibles 64 bits.

## Les greffons de Safari

L'impossibilité des applications 64 bits à charger des greffons 32 bits est aussi un problème pour Safari. Les greffons sont si importants pour travailler sur le Web, que relancer le mode 32 bits n'est pas réellement une option. Vous seriez sans doute obligé de relancer en mode 32 bits dès que vous visitez votre première page web. Mais Apple tient à ce que Safari fonctionne en mode 64 bits, à cause d'améliorations importantes des performances dans le moteur Javascript, et d'autres parties de l'application qui ne sont pas disponibles en mode 32 bits.

La solution retenue est similaire à ce qu'Apple a fait avec QuickTime X et les greffons 32 bits de QuickTime 7. Safari fait tourner les greffons 32 bits dans des processus 32 bits séparés si nécessaire.




*Des processus séparés pour les greffons 32 bits Safari*

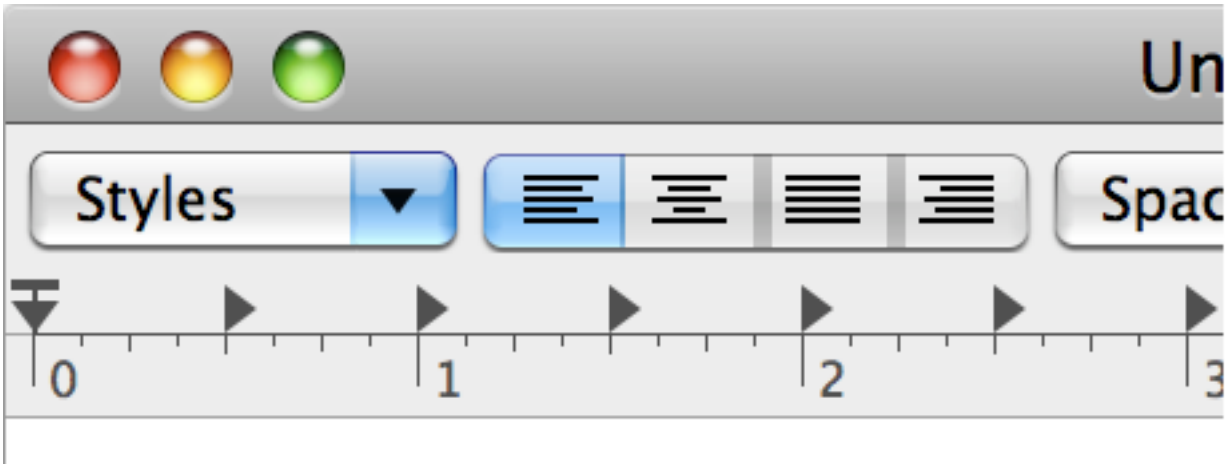
Ceci a l'avantage supplémentaire, très important, d'isoler les greffons potentiellement bogués. En se basant sur les rapports de crash automatisés, Apple a identifié que la cause principale des crash est les greffons du navigateur Web. Faites attention, ce n'est pas la cause numéro 1 des crash dans Safari, c'est la cause numéro 1 quand on considère les crash de *toutes* applications dans Mac OS X. Et bien que le nom n'ait pas été donné, je pense que nous connaissons tous le responsable principal.

Comme vous le voyez ci-dessus, le greffon QuickTime a le même traitement que Flash et les autres greffons 32 bits de parties tierces pour Safari. Tout ceci signifie que quand un greffon se crash, Safari dans Léopard des neiges résiste. La fenêtre ou l'onglet contenant le greffon en cause ne se ferme même pas. Vous pouvez simplement cliquer sur le bouton "recharger", et donner au greffon à problème une nouvelle chance de fonctionner convenablement.

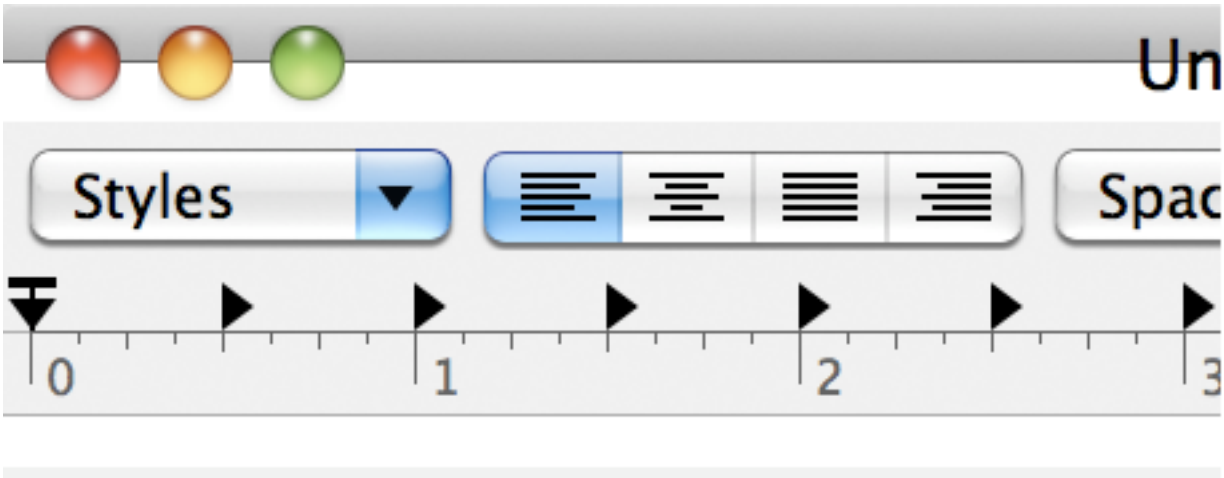
Bien que cela soit encore loin de l'approche beaucoup plus robuste employée par Google Chrome où chaque onglet existe dans son propre processus indépendant, si on en croit les statistiques d'Apple, l'isolation des greffons apporte l'essentiel des bénéfices de processus séparés, avec un changement beaucoup moins radical de l'application Safari elle-même.

## Indépendance vis à vis de la résolution

Quand nous avons laissé pour la dernière fois Mac OS X dans son interminable quête d'une interface utilisateur réellement capable de changements d'échelle, c'était *presque* prêt pour la première fois. Je suis au regret de dire que l'indépendance vis à vis de la résolution n'a pas été, à l'évidence une priorité de Léopard des neiges, parce que ce n'est pas mieux, et que ce serait même plutôt une régression. Voilà comment TextEdit ressemble à un facteur d'échelle 2.0 dans Léopard et dans Léopard des neiges




*TextEdit à l'échelle 2 dans Léopard*

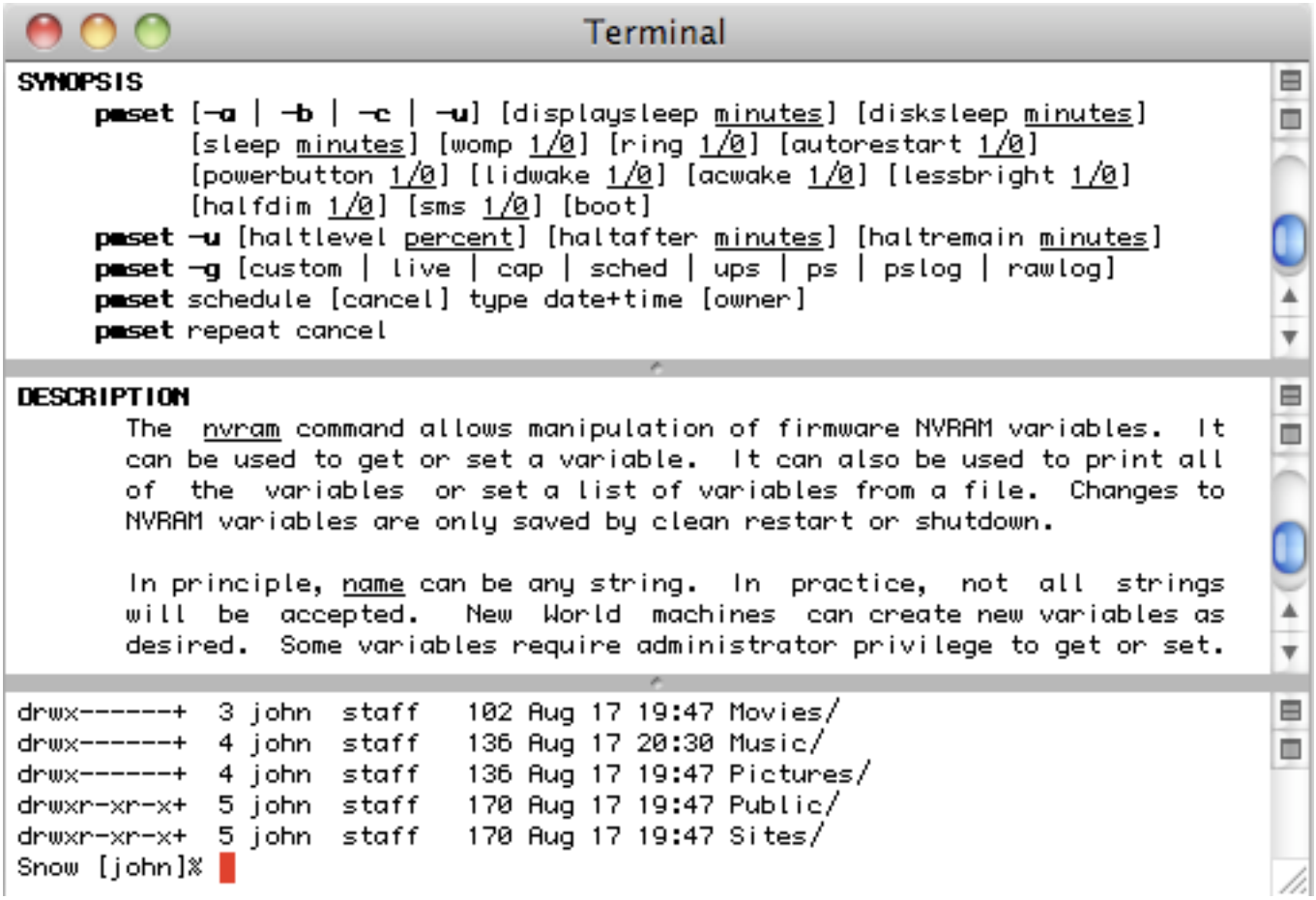



*TextEdit à l'échelle 2 dans Léopard des neiges*

Ouais, c'est désolant. Je me rappelle encore Apple encourageant les développeurs à avoir leurs applications prêtes pour l'indépendance vis à vis de la résolution pour 2008. C'est une des rares dates que Apple de l'ère Jobs II n'a pas été capable de respecter, et c'est de plus en plus en retard. D'un autre côté, on ne voit pas non plus les moniteurs à 200 DPI pleuvoir du ciel. Mais j'aimerais vraiment voir Apple persévérer là dedans. Cela va sans aucun doute prendre beaucoup de temps pour que tout apparaisse et fonctionne correctement, alors prenons le départ.

## Les séparations dans Terminal

L'application Terminal dans Tiger et les précédentes versions de Mac OXS X permettait que chaque fenêtre soit séparée horizontalement en deux vitres distinctes. C'était indispensable pour se référer à une partie antérieure du texte en défilement, tout en tapant des commande à l'invite. De façon regrettable, la séparation avait disparu dans Léopard. Dans Léopard des neiges, la voilà qui revient, et se venge.




*Des séparations comme on veut*.

(Maintenant, si mon éditeur de texte favori montait aussi dans le train pour Séparation-City... )

Dans Léopard des neiges, le Terminal utilise aussi par défaut la nouvelle fonte Menio. Mais contrairement à des affirmations passées, la seule vraie fonte monospace, Monaco, est aussi incluse à coup sûr dans Léopard des neiges (voyez l'image au dessus), et elle fonctionne très bien.

## Les errances de Préférences Système

Le réarrangement apparemment obligatoire des icônes dans l'application Préférences système qui accompagnent chaque version de Mac OS X continue avec Léopard des neiges.


*Les Préférences Système ont encore été rebattues.*


*Préférences Système dans le menu Dock.*

Cette fois, la vitre "Clavier et Souris" a été séparée en deux : "Clavier" , et "Souris". "International" devient "Langage et Texte" et la section "Internet & Réseau""

devient "Internet & Sans Fil" et récupère la vitre des préférences Bluetooth.

Peut-être, dans un futur lointain, Apple arrivera-t-il finalement à l'arrangement "ultime" des préférences, et nous pouvons tous supporter plus de deux ans sans que notre mémoire motrice soit perturbée.

Avant de passer à la suite, les préférences systèmes ont un truc. Vous pouvez lancer directement une vitre spécifique des Préférences en faisant un clic droit sur l'icône des Préférences Système dans le Dock. Cela marche même quand les préférences système ne tournent pas. Un peu inhabituel mais utile.

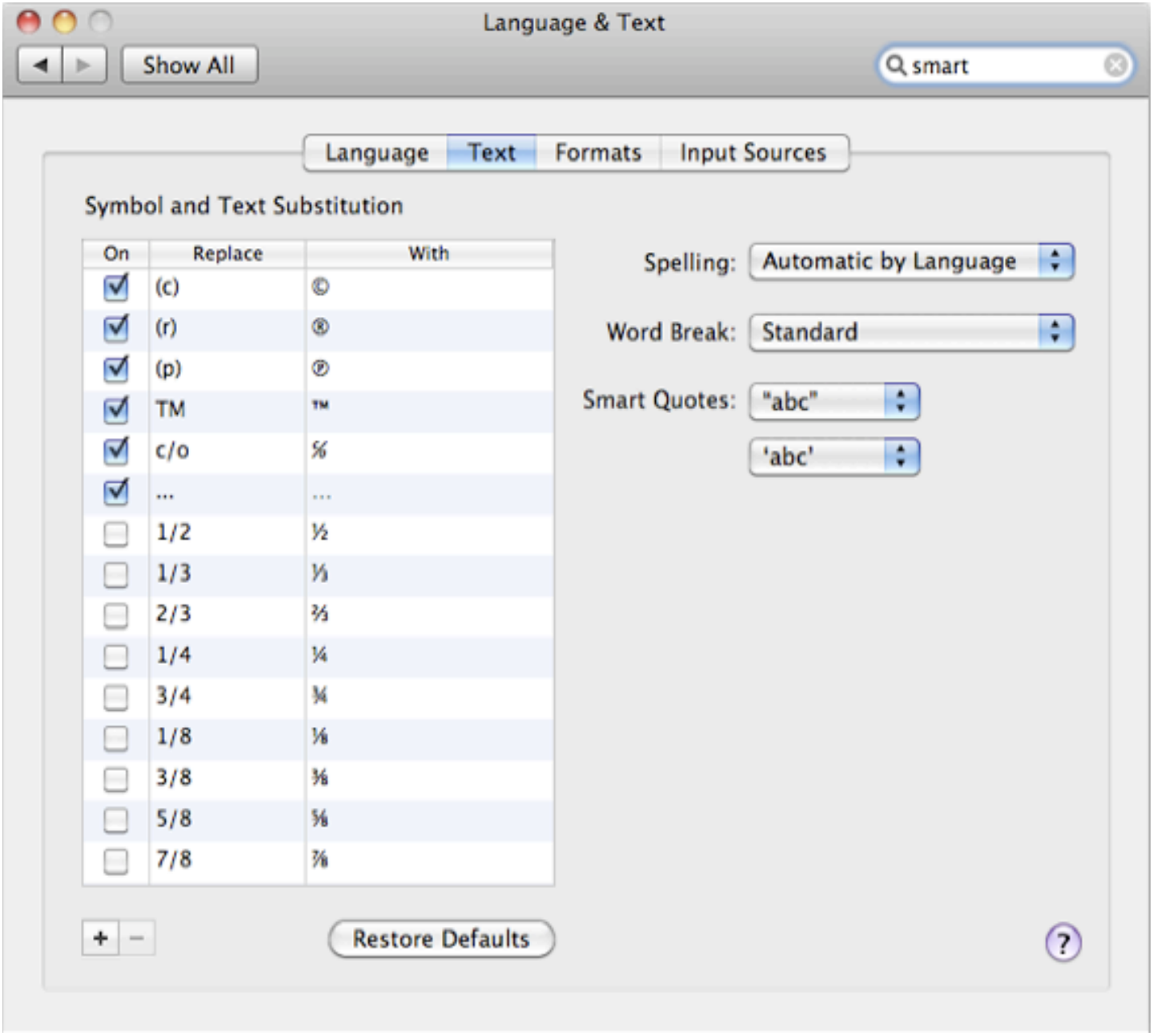
## Core Location

Un autre cadeau venu de l'iPhone, Core Location, permet de savoir où vous êtes, sur terre. La vitre de préférences "Date et heure" vous donne votre zone horaire automatiquement, en utilisant cette nouvelle possibilité.


*Réglez votre zone horaire en fonction de votre position sur le globe, grâce à Core Location*

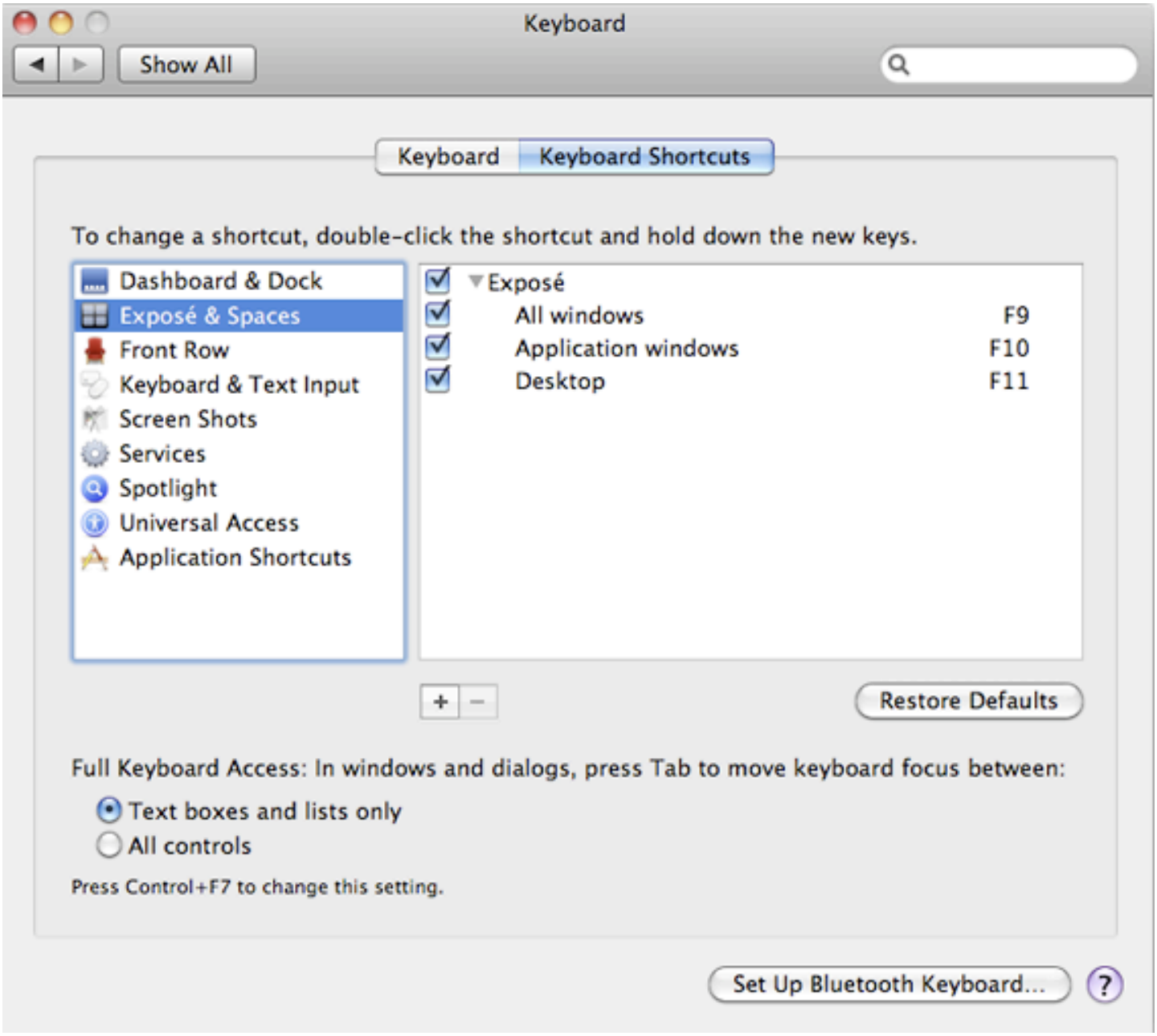
## La magie du clavier

Léopard des neiges inclut un outil système simple pour l'auto-correction et l'expansion de texte disponible dans la vitre de préférences "Langage & Texte". Il n'est pas tout à fait prêt pour remplacer une application tierce payante, mais, hé, il est gratuit.




*Expansion du texte et auto-correction*

La vitre des préférences raccourcis clavier a été réorganisée. Maintenant, au lieu d'une seule longue liste de raccourcis valables pour tout le système, ils sont groupés en catégories. Cela réduit le désordre, mais rend aussi plus difficile l'accès au raccourci qui vous intéresse.




*Les raccourcis clavier, maintenant regroupés en catégories*.

# Pot pourri (suite)

## Le dilemme du Mac endormi

Je n'aime pas laisser mon Mac Pro tourner 24 heures sur 24, particulièrement en été dans ma maison sans air conditionné. Mais je veux avoir accès aux fichiers sur mon

Mac quand je suis ailleurs, au travail, sur la route, etc... Il possible de réveiller à distance un Mac endormi, mais il faut pour cela être sur le même réseau local.

Ma solution a été de laisser un portable, plus petit, moins gourmand en énergie ouvert tout le temps sur le même réseau que mon Mac Pro. Pour réveiller mon Mac Pro à distance, j'envoie un ssh sur le portable, et j'envoie le paquet magique du réveil au Mac Pro. (Pour que ça marche, la case "Réactiver lors d'un accès administrateur via le réseau Ethernet" doit être cochée dans les Options de la vitre de Préférences "Economiseur d'énergie").

Léopard des neiges fournit une solution pour cela sans laisser aucun de mes ordinateurs en fonctionnement toute la journée. Quand un Mac sous Léopard des neiges se met en veille, il cherche à récupérer son adresse IP après du routeur. (Cela ne fonctionne qu'avec une base Airport Extrême de 2007 ou plus récente, ou une Time Capsule de 2008 ou plus récente, avec le dernier firmware -7.4.2- installé). Le routeur surveille alors tout essai pour se connecter à l'adresse IP. Quand il s'en produit un, il réveille le propriétaire, renvoie l'adresse IP, et transmet le trafic comme il se doit.

Vous pouvez même réveiller quelques Macs de modèle récent par Wi Fi. Combiné avec la fonction de reconnaissance "Back to my Mac" du DNS dynamique de MobileMe, cela signifie que je peux laisser tous mes Macs endormis, et avoir encore accès à leur contenu, n'importe où, n'importe quand.

## Retour sur le hack

Comme cela est devenu traditionnel, cette nouvelle version de Mac OS X rend la vie un peu plus difficile pour les développeurs dont le logiciel fonctionne en bricolant la représentation en mémoire d'autres applications actives, ou du système d'exploitation lui-même. Cela inclut les gestionnaires d'entrée, les greffons SIMBL, et bien sûr, les redoutés Haxies

Les gestionnaires d'entrée récupèrent la plus mauvaise part. Actuellement, ils ne sont pas acceptés, et ne fonctionnent pas dans les applications 64 bits depuis Léopard. Ce n'était pas une grosse affaire, quand Mac OS X était livré avec seulement deux applications 64 bits. Mais maintenant, avec presque toutes les applications de Léopard des neiges en 64 bits, cela devient très important.

Grâce à l'absence dans Safari d'un mécanisme d'extension officiellement approuvé, les développeurs cherchant à en augmenter les possibilités se sont le plus souvent reportés sur l'utilisation de gestionnaires d'entrée, et sur SIMBL (qui est un framework de gestionnaire d'entrée). Safari en 64 bits réduit considérablement ce

marché. Bien qu'il soit possible de configurer manuellement Safari pour qu'il se lance en 32 bits - Get Info sur l'application, dans le Finder, et cocher une case- , ce n'est pas dans l'idéal ce que les développeurs veulent forcer les utilisateurs à faire.

Heureusement, au moins une des améliorations communément utilisées de Safari a le bon goût de s'installer par dessus l'API de greffons à usage du navigateur, utilisée par Flash, QuickTime, etc., et officiellement acceptée. Mais ce n'est pas une approche facile pour les extensions Safari qui améliorent les fonctionnalités d'une façon qui n'est pas directement associée à l'affichage de types de contenus particuliers dans une page Web.

Bien que j'envisage d'utiliser Safari dans son mode par défaut en 64 bits, je vais beaucoup regretter Saft, une extension Safari que j'utilise pour la restauration d'une session (oui, je sais que Safari a cette possibilité, mais elle est activée *manuellement*, une horreur) et pour les raccourcis de la barre d'adresse (par exemple "w noodles" pour aller voir "noodles" dans Wikipédia). J'espère que des développeurs avisés vont trouver un moyen de dépasser ce nouveau défi. Ils semblent toujours y arriver, en fin de compte. (Ou bien, Apple pourrait rajouter sa propre extension système à Safari, évidemment. Mais je ne retiens pas mon souffle).

Quant aux Haxies, ils ne fonctionnent habituellement plus après chaque mise à jour du système, bien sûr. Et à chaque fois, les gars déterminés chez Unsanity, en dépit de toute probabilité, réussissent à maintenir leur logiciel en marche. Je les salue pour leur effort. J'ai retardé ma mise à jour à Léopard pendant longtemps uniquement à cause de l'absence de l'adoré WindowShade X. J'espère que je n'aurai pas à attendre trop longtemps une version compatible avec Léopard des neiges.

La tendance générale dans Mac OS X est toujours depuis une forme quelconque de partage involontaire de l'espace mémoire, vers des greffons "externes" qui vivent leur propre vie, des processus séparés. Même les greffons de menu contextuel dans le Finder ont été invalidés, et remplacés par une API Services, améliorée, mais toujours moins efficace. A nouveau, je fais confiance aux développeurs pour s'adapter. Mais l'attente est la chose la plus difficile.

## L'échec sur ZFS

Il semble que nous allons tous attendre un peu plus longtemps pour qu'un système de fichiers en armure dorée remplace le vénérable HFS+ (onze ans d'âge) comme système de fichiers par défaut dans Mac OS X. En dépit des rumeurs, de déclarations claires, de beaucoup de code disponible en pre-release, le support de l'impressionnant système de fichiers ZFS n'est pas présent dans Léopard des neiges.

C'est une honte, parce que Time Machine aurait véritablement besoin de la puissance de ZFS. Qui plus est, Apple semble d'accord, comme le montre la réponse d'un employé d'Apple à la liste sur ZFS l'an dernier . Interrogé sur une implémentation utilisant ZFS, la réponse était encourageante : " Cette chose là est importante, et viendra sans doute un jour, mais pas pour SL" (SL mis pour Snow Leopard).

Il y a beaucoup des raisons pour lesquelles ZFS (ou un système de fichiers similaire) s'adapte parfaitement à Time Machine, mais la plus importante est sa capacité à envoyer seulement les modifications au niveau d'un bloc pour chaque sauvegarde. Tel que Time Machine est actuellement fait, si vous faites une petite modification dans un fichier géant, le fichier géant tout entier est copié sur le volume de Time Machine au cours de la sauvegarde suivante. Cela est très dispendieux et prend beaucoup de temps, en particulier pour les gros fichiers qui sont constamment modifiés au cours d'une journée (par exemple la base de courriel d'Entourage). Time Machine tournant sur ZFS ne pourrait transmettre que les blocs du disque qui ont changé (128 Ko au maximum sous ZFS, et généralement beaucoup moins).

ZFS apporterait aussi une robustesse accrue pour les données et les méta-données, un modèle de stockage mutualisé (pool), des clichés et des clones à heure constante, et un poney. Les gens demandent parfois qu'est-ce qui ne va pas avec HFS+. En plus du manque évident des possibilités listées plus haut, HFS + est limité de nombreuses façons par sa conception ancienne, basée sur HFS, un système de fichiers qui a maintenant 25 ans.

Pour ne donner qu'un exemple, le fichier Catalogue, au centre du système, qui doit être modifié à chaque modification de la structure du système de fichiers est une inévitable et fréquente source d'ennuis. Les systèmes de fichiers modernes dispersent leurs méta-données, pour plus de robustesse (des copies multiples sont souvent conservées à des endroits différents sur le disque), et permettent une meilleure simultanéité.

En pratique, pensez à ces moments où vous utilisez Utilitaire de Disques sur un volume HFS + et où il trouve (et répare si possible) une foule d'erreurs. C'est mauvais, n'est-ce pas ? C'est quelque chose qui ne devrait pas arriver avec un système de fichiers moderne, soigneusement vérifié, et toujours consistant sur le disque, à moins qu'il y ait des problèmes matériels (et un pool de stockage ZFS peut tout à fait gérer cela aussi). Et pour le moment, cela arrive tout le temps avec les disques HFS + de Mac OS X, quand certains bits de méta-données se trouvent corrompus, ou sont périmés.

Apple laisse passer les années, en raccommodant de nouvelles fonctionnalités sur HFS+ avec du ruban adhésif et des prières, mais à un certain moment, il faut bien qu'il y ait un successeur, que ce soit ZFS, un système de fichiers concocté chez Apple, ou quelque chose de complètement différent. Je croise les doigts pour Mac OS 10.7.

# Le futur, bientôt

La création d'un système d'exploitation est une pratique sociale autant qu'une prouesse technique. C'est encore plus vrai pour la création d'une plate-forme. Toutes les réussites techniques considérables de Léopard des neiges ne sont pas seulement conçues pour bénéficier aux utilisateurs, elles ont aussi pour but d'aiguillonner, de convaincre, et aussi de conduire les développeurs dans la direction qu'Apple considère comme la plus bénéfique pour l'avenir de la plate-forme.

Pour que ça marche, Léopard des neiges doit passer entre les mains des consommateurs. Le prix aide beaucoup pour cela. Mais même si Léopard des neiges était gratuit, il y aurait encore un coût pour les consommateurs, -en temps, ennuis, mises à jour, etc.- à faire une mise à jour majeure du système d'exploitation. La même chose vaut pour les développeurs, qui doivent, au minimum, certifier que leurs applications existantes fonctionnent correctement sur le nouvel OS.

La meilleure façon de dépasser cette hésitation à faire la mise à jour a été de mettre des nouveautés avec les nouvelles livraisons. Les nouveautés se vendent, et plus il y a de copies du nouveau système utilisées, plus les développeurs sont motivés pour mettre à jour leurs applications, pas seulement pour marcher sur le nouvel OS, mais aussi pour tirer parti de ses nouvelle possibilités.

Une mise à jour nouvelle avec "aucune nouvelle fonctionnalité" doit appliquer des règles différentes. Chaque partie impliquée s'attend à une contre-partie pour l'absence de nouvelles caractéristiques. Dans Léopard des neiges, ce sont les développeurs qui vont pouvoir récupérer les plus gros bénéfices, grâce à un ensemble de nouvelles technologies impressionnant, dont beaucoup concernent des secteurs auparavant jamais abordés. Apple ressent clairement que le futur de la plate-forme dépend d'une bien meilleure utilisation des ressources de calcul, et il fait tout pour permettre facilement aux développeurs d'aller dans cette direction.

Bien qu'il soit évident que Léopard des neiges contienne beaucoup moins de fonctionnalités externes que son prédécesseur, je prendrais le risque d'affirmer qu'il contient au moins autant, sinon plus, de modifications internes que Léopard. Cela signifie, je le crains, que la version initiale de Léopard des neiges risque de souffrir des bogues typiques à une version 10.x.0. Il y a déjà eu des rapports sur de nouvelles bogues, introduites dans les APIs existantes de Léopard des neiges. Ceci va à l'encontre de la promesse implicite à l'égard des utilisateurs et des développeurs, selon laquelle Léopard des neiges allait se concentrer sur les fonctionnalités existantes, et les rendre plus robustes sans en introduire de nouvelles, avec leur cortège ne nouvelles bogues. Voilà le côté pile.

Du côté face, j'imagine que toutes les équipes d'Apple ont travaillé sur Léopard des neiges absolument ravies par l'opportunité de polir leur propre sous-système, sans être importunées par les exigences marketing sur la caractéristique du mois. Dans un produit à long cycle de vie, il faut ce genre de soupape au bout de quelques années pour éviter que la base tout entière de code ne s'échappe dans la nature.

Il y a eu aussi une autre livraison de Mac OS X "sans nouvelles caractéristiques". Mac OS X 10.1, sortie à peine six mois après la version 1.0 fut proposée gratuitement par Apple à la conférence des publications Seybold de 2001, et plus tard, dans les magasins Apple. Elle était aussi disponible en ligne pour 19,95 dollars (avec une copie de Mac OS 9.2.1 pour l'environnement classique). C'était une époque différente pour Mac OS X : les versions 10.0 et 10.1 étaient lentes, incomplètes, très largement immatures. La transition depuis Mac OS Classic était loin d'être achevée.

Considérée comme une incarnation moderne de la livraison 10.1, Léopard des neiges apparaît comme vachement bonne. Le prix est semblable, et les bénéfices, pour les développeurs *et* les utilisateurs sont plus grands. Le risque est à l'avenant. Mais cette fois aussi, cela ressemble à l'état horrible dans lequel était Mac OS X 10.0. Choisir de ne pas faire la mise à jour en 10.1 était impensable. Attendre un peu pour se mettre à Léopard des neiges est raisonnable si vous voulez vous assurer que le logiciel dont vous avez besoin est compatible. Mais n'attendez pas trop longtemps, car à cause de la mise à jour à 29 euros, je pense que l'adoption de Léopard des neiges sera très rapide. Des logiciels qui ne fonctionneront plus que sur Léopard des neiges risquent d'être disponibles avant que vous le sachiez.

Devriez-vous acheter Léopard des neiges ? Si vous êtes sous Léopard, la réponse résonne en écho : oui, oui, oui. Si vous êtes toujours sous Tiger, eh bien, c'est probablement le moment d'avoir un nouveau Mac, de toute façon. Quand vous allez l'acheter, il viendra avec Léopard des neiges.

Et pour le futur, il est tentant de considérer Léopard des neiges comme comme un "tic" dans la nouvelle stratégie de style Intel de livraison de Mac OS X en "tic-tac" : de nouvelles caractéristiques fondamentales dans la version 10.7, suivies par des raffinements du style de Léopard des neiges dans la version 10.8, et ainsi de suite, une alternance entre les "caractéristiques", et les "raffinements". Apple n'a pas donné la moindre indication qu'elle est engagée dans ce genre de planification, mais je pense qu'il y a beaucoup de raisons de la recommander.

Léopard des neiges et une livraison unique et belle, à la différence de ce qui a précédé, du point de vue de la cible aussi bien que de l'intention. A un moment, Mac OS X devra sûrement revenir dans la course aux nouvelles caractéristiques bien ciblées. Mais à présent, je suis ravi de Léopard des neiges. C'est le Mac OS X que je connais et que j'aime, mais avec beaucoup de choses qui le rendent faible et étrange, à l'avant-garde.

## Retour sur le passé

Ceci est le dixième compte-rendu d'une livraison de Mac OS X, complète, ou beta publique, ou à destination des développeurs qui paraît sur le site de Ars, depuis la DP2 Mac OS X datant de Décembre 1999. Si vous voulez monter dans la machine à remonter le temps, et voir jusqu'où Apple est arrivé avec Léopard des neiges ( ou si vous voulez seulement passer en revue tous ces noms de gros chats), nous sommes allés à la pêche aux archives, et avons redécouvert quelques uns des plus vieux articles sur Mac OS X. Bonne lecture !

- Cinq années de Mac OS X, 24 Mars 2006
- Mac Os X 10.5 Leopard, 28 Octobre 2007
- Mac OS X 10.4 Tiger, 28 Avril 2005
- Mac OS X 10.3 Panther, 9 Novembre 2003
- Mac OS X 10.2 Jaguar, 5 Septembre 2002
- Mac >OS X 10.1 (Puma), 15 Octobre 2001
- Mac OS X 10.0 (Cheeta), 2 Avril 2001
- Mac OS X public Beta, 3 Octobre 2000
- Mac OS X Q&A, 20 Juin 2000
- Mac OS X DP4, 24 Mai 2000
- Mac OS X DP3 : Trial by Water, 23 Février 2000
- Mac OS X Update : Quartz & Aqua17 janvier 2000
- Mac OS X DP2, 14 Décembre 1999