

# INF2

Principes des  
Systèmes  
Informatiques

**le système de gestion de fichiers**

Olivier Ridoux

# INF2 Principes des Systèmes Informatiques

## Plan

- Description
- Challenge
- Réponses

# INF2 Principes des Systèmes Informatiques

## Description

# Système [de gestion] de fichiers

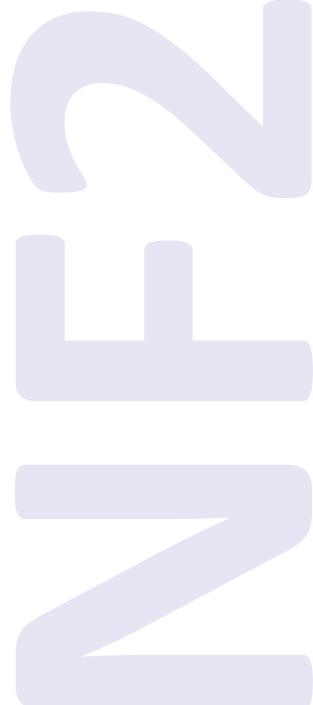
- SGF ou *file system*
- Permet la manipulation d'un ensemble de **fichiers**
  - création / modification / suppression
  - rangement / classement
  - recherche / consultation

# Modèle, vue et contrôle

- Une façon de décrire un système informatique
- Modèle
  - l'**organisation** des données contenues dans le système
- Vue
  - comment sont **présentées** les données
- Contrôle
  - comment on **manipule** les données

# INF2 Principes des Systèmes Informatiques

## Modèle



## Le modèle

- Des **fichiers** organisés en **répertoires** et **sous-répertoires**
  - *folders, directories, dossiers*
  - organisés comment ?
- Version 1
  - organisés en **arbre**
- Version 2
  - organisés en **graphe**

# Rappel ? Puissances (1)

- Définitions

si  $b \in \mathbb{N}$ ,  $a^b = \underbrace{a \times a \times \dots \times a}_{b \text{ fois}}$

$$a^{b+c} = a^b \times a^c$$

$$a^{b-c} = a^b / a^c$$

$$a^{b \times c} = (a^b)^c = (a^c)^b$$

$$a^{b/c} = x \text{ tel que } x^c = a^b$$

- Si  $b > 1$ , comportement « explosif »
  - ex. loi de Moore : la puissance des circuits intégrés double tous les 18 mois



# Rappel ? Logarithmes

- Définitions

- si  $a > 0$ ,  $\log_b a = x$  tel que  $b^x = a$

$$\log_b a = \log_c a / \log_c b$$

$$\log_b a \times a' = \log_b a + \log_b a'$$

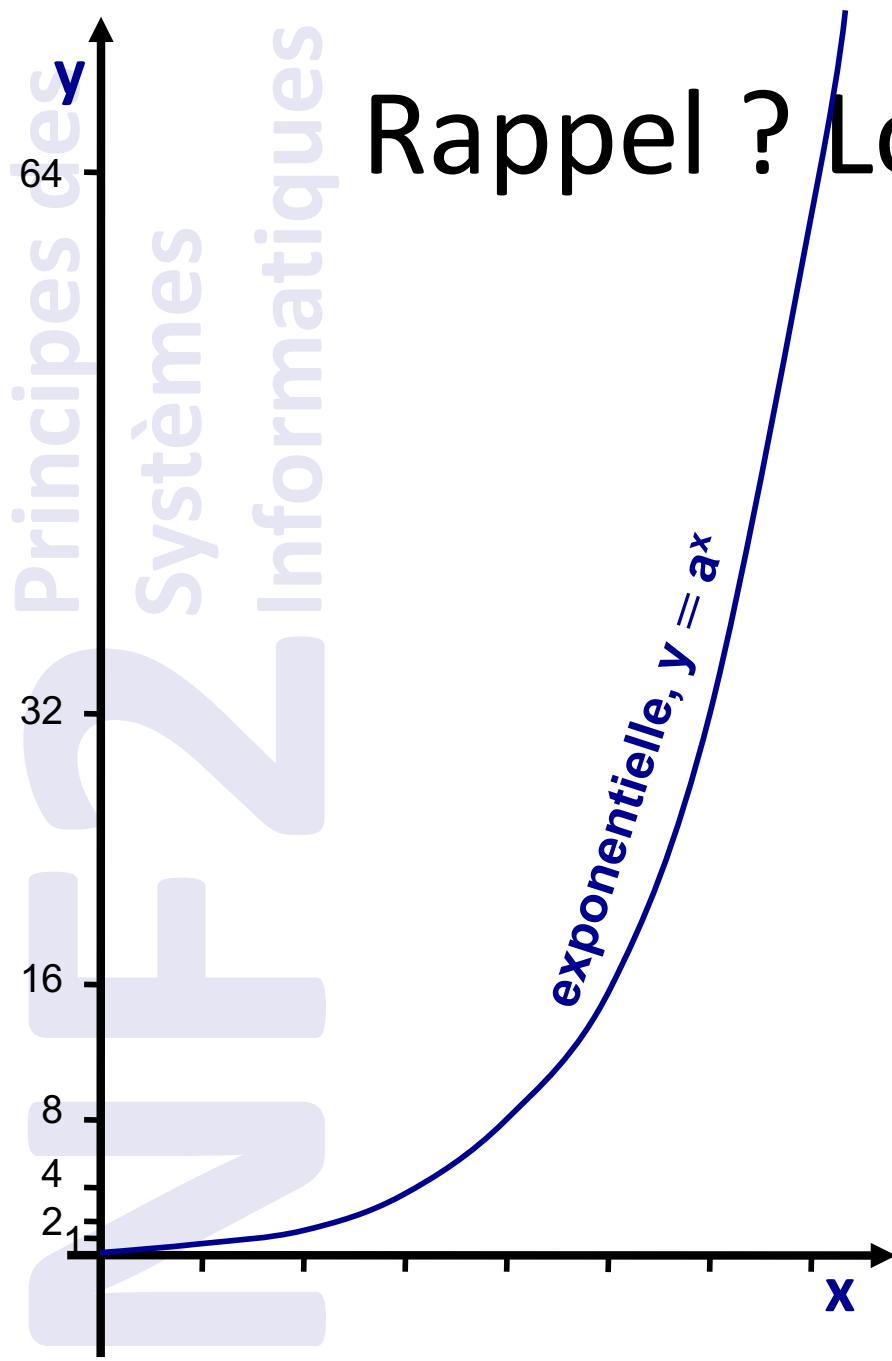
$$\log_b 1/a = -\log_b a$$

$$\log_b b = 1 \text{ et } \log_b 1 = 0$$

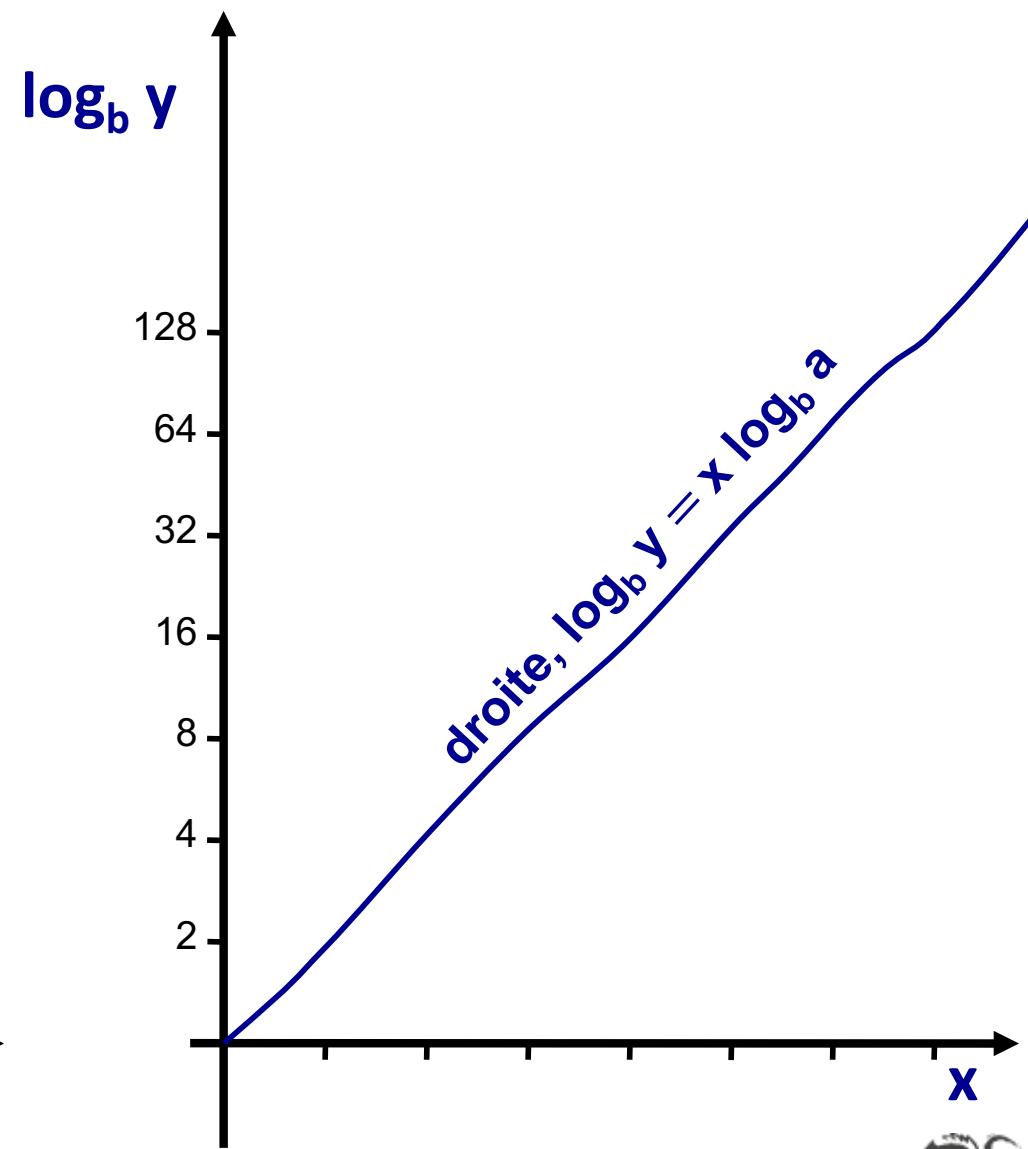
$$x^{\log_b y} = y^{\log_b x}$$

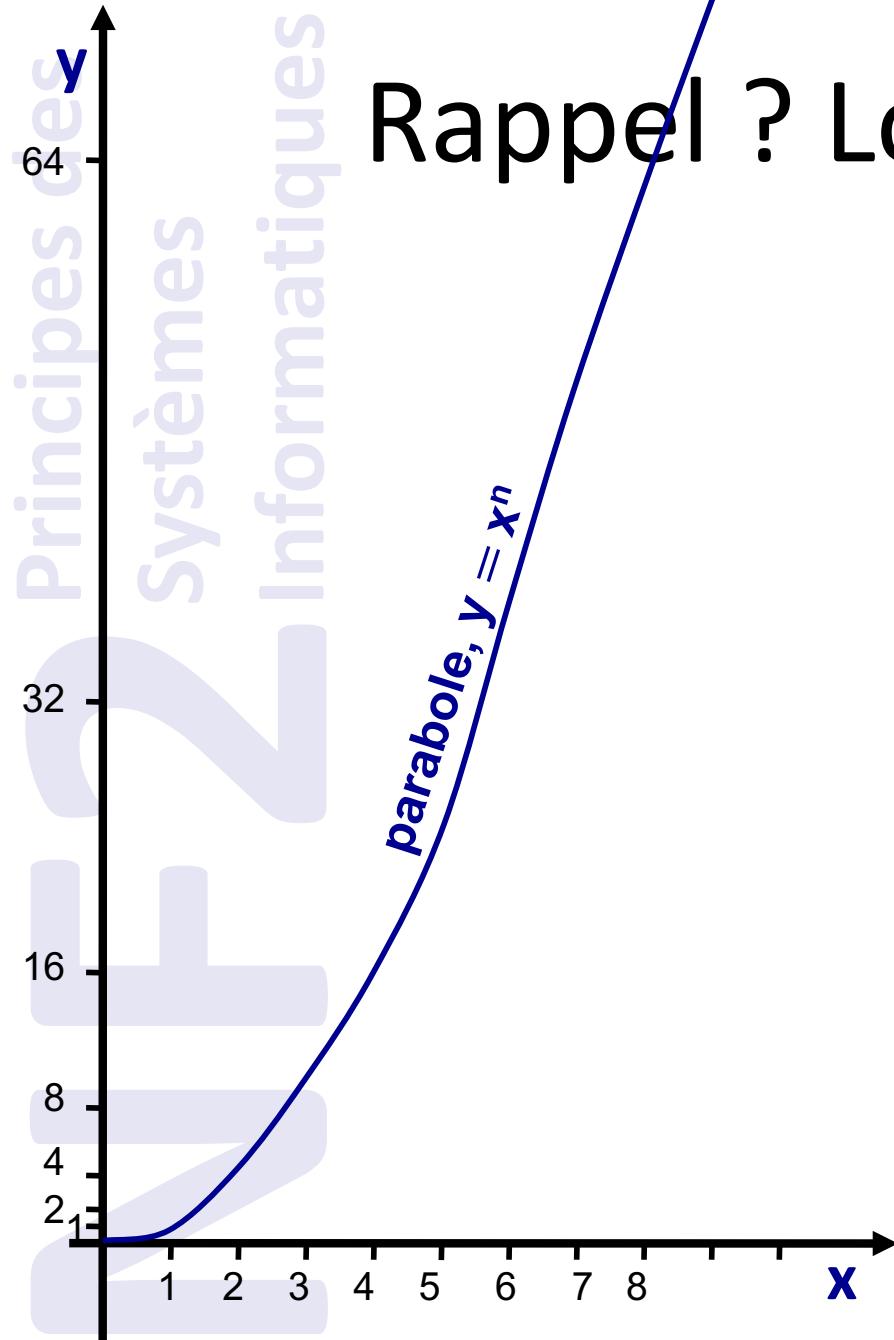
- Si  $b > 1$ , comportement « écrasant »

$$\log_b a \ll a$$

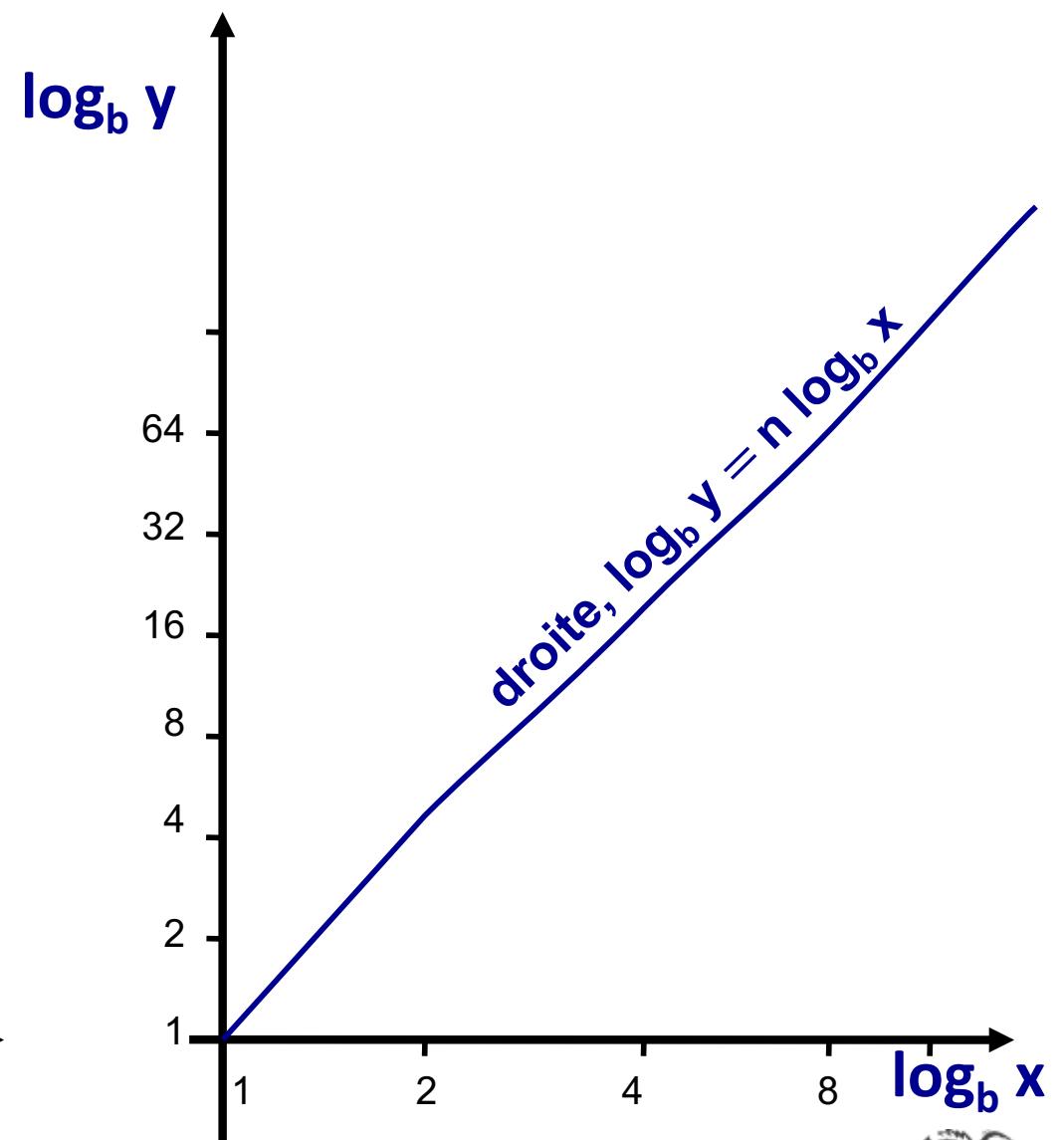


# Rappel ? Logarithmes (2)





# Rappel ? Logarithmes (3)



# Fichier

- Ensemble de **données** à qui l'utilisateur donne un **nom arbitraire**
- Ensemble de données de **taille quelconque**
- Données cohérentes pour une **application**
  - préserver leur intégrité
  - noter l'application : **pdf, java, doc, ...**
- Nom **arbitraire**
  - pas de fonction simple **nom → fichier**
  - utiliser un **dictionnaire**

# Dictionnaire naïf (1)

- Un dictionnaire **nom → fichier**

( (file "nomfile" @système<sub>1</sub>)

...

(file "nomfile" @système<sub>i</sub>)

...

(file "nomfile" @système<sub>N</sub> )

## Dictionnaire naïf (2)

- Coût d'emploi d'un dictionnaire
- On suppose N fichiers
  - N entrées dans le dictionnaire
  - coût de l'insertion d'un fichier
    - vérifier que le nom du fichier est nouveau
    - **coût = N**
  - coût de la recherche d'un fichier
    - recherche séquentielle dans le dictionnaire
    - **coût = N**

**Catastrophique si N est grand !**

# Dictionnaire hiérarchique (1)

- Fragmenter le dictionnaire en sous-dictionnaires
- Un dictionnaire  
**nom → fichier U répertoire**

( (file "nomfile" @système<sub>1</sub>)

...

(dir "nomdir" @système<sub>i</sub>)

...

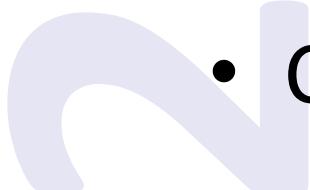
(file "nomfile" @système<sub>N</sub>) )

# Dictionnaire hiérarchique (2)

- Hyp.: sous-dictionnaires de tailles égales = T

**Surcoût très faible**

nb sous-dir.	nb fich.	vol. sous-dir.	surcoût (%)
1	$N = T$	$T = N$	0
$1+T$	$N = T^2$	$T+T^2 = N+N^{1/2}$	$N^{-1/2}$
$1+T+T^2$	$N = T^3$	$T+T^2+T^3 = N+N^{1/3}+N^{2/3}$	$1/N^{1/3} + 1/N^{2/3}$ $\approx 1/N^{1/3} = 1/T$
...			
$1+T+\dots+T^{n-1} [= (T^n-1)/(T-1)]$	$N = T^n$	$T \times (T^n-1)/(T-1)$	$\approx 1/T$



## Aparté : combien ? (1)

- Combien de fichiers sur votre ordinateur personnel ? Quel volume ?  
**≈ 200000 fichiers, 72 Go → 360 Ko/fichier**
- Combien de répertoires ?  
**≈ 50000 → T moyen = 4 → 9 niveaux**
- Combien de fichiers sur votre compte ?  
**≈ 115000, 70 Go → 600 Ko/fichier**
- Combien de répertoires ?  
**≈ 23000 → T moyen = 5 → 9 niveaux**



## Aparté : combien ? (2)

- Combien de fichiers sur votre ordinateur personnel ? Quel volume ?
- Combien de répertoires ?
- Combien de fichiers sur votre compte ?
- Combien de répertoires ?

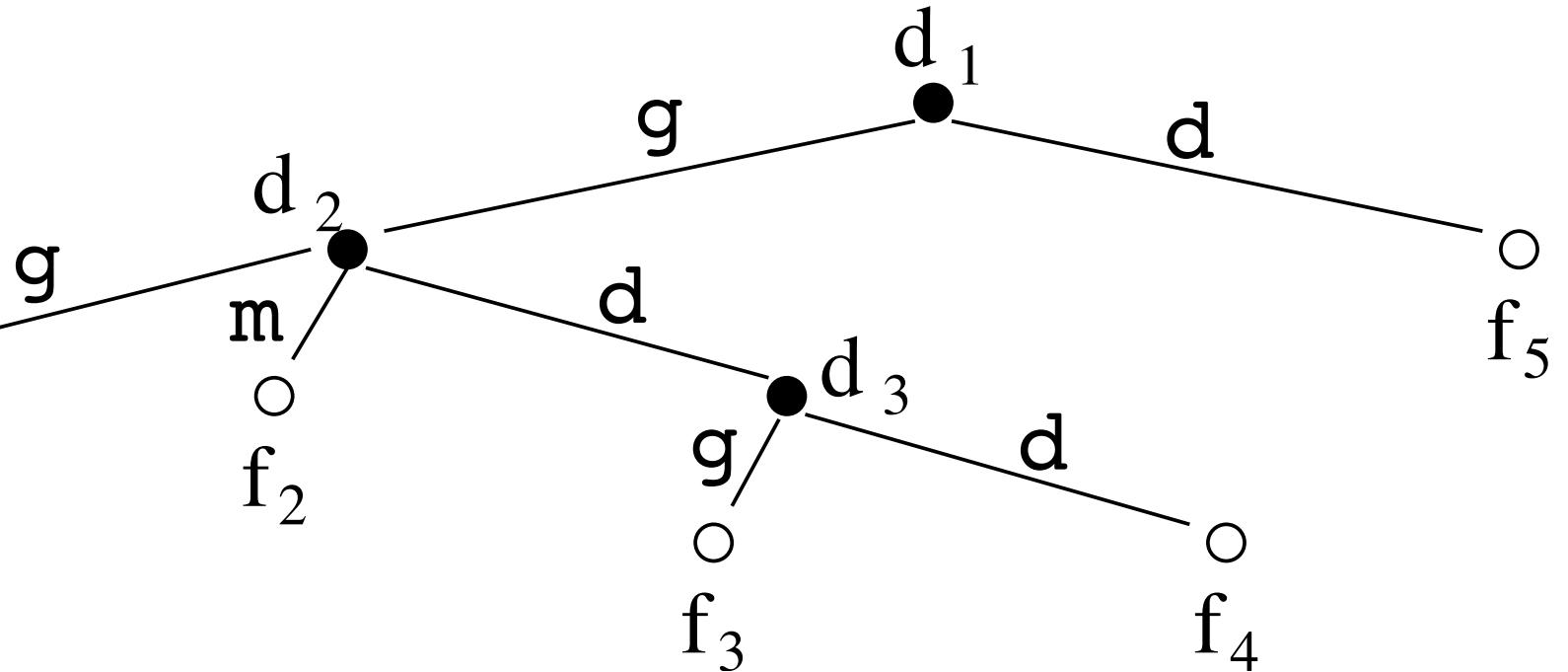
**À faire sur votre ordinateur personnel  
et sur votre compte**

# Dictionnaire hiérarchique (3)

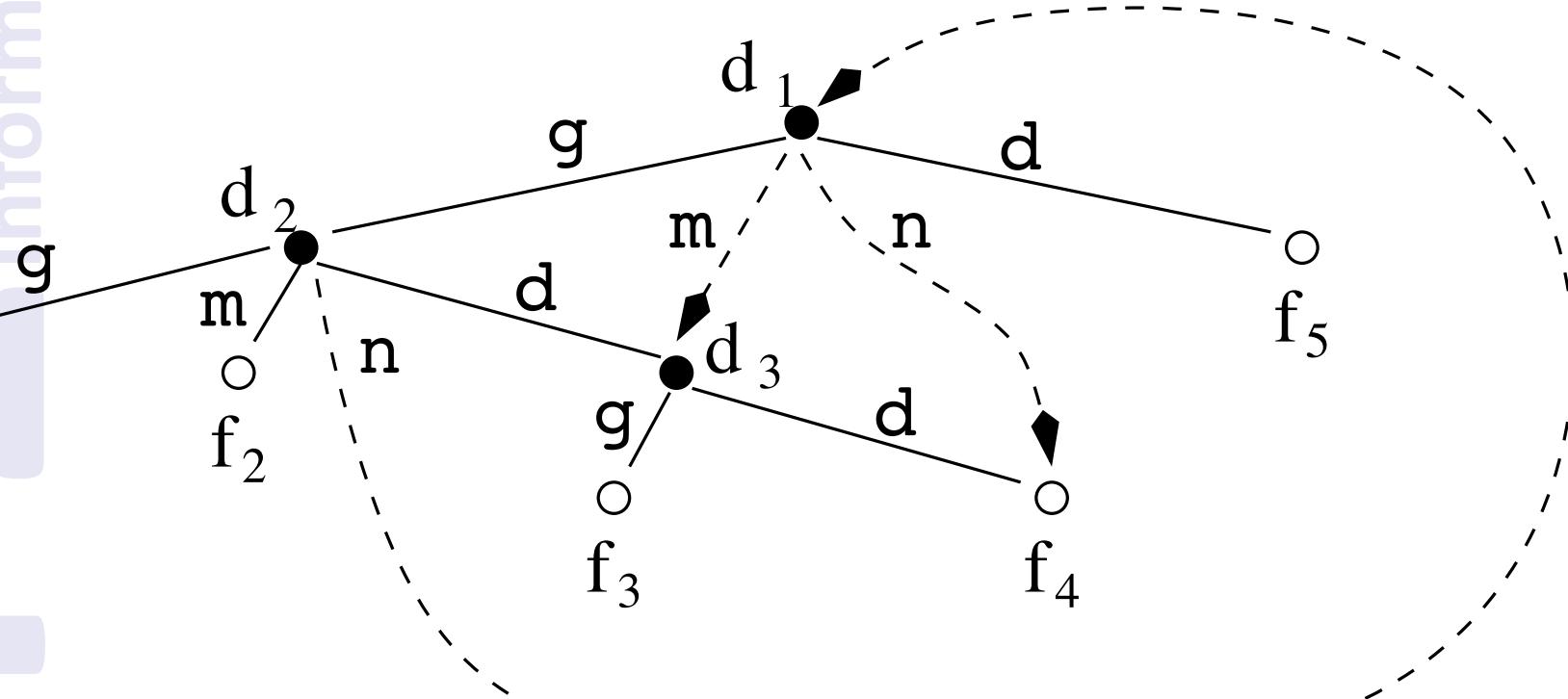
- Coût de l'insertion = T
- Coût de la recherche = nb niv.  $\times$  T =  $\log_T N \times T$

nb fichiers = N	taille sous-dir. = T	coût rech.
100	10	20
100	100	100
10000	10	40
10000	100	200
1000000	10	60
1000000	100	300

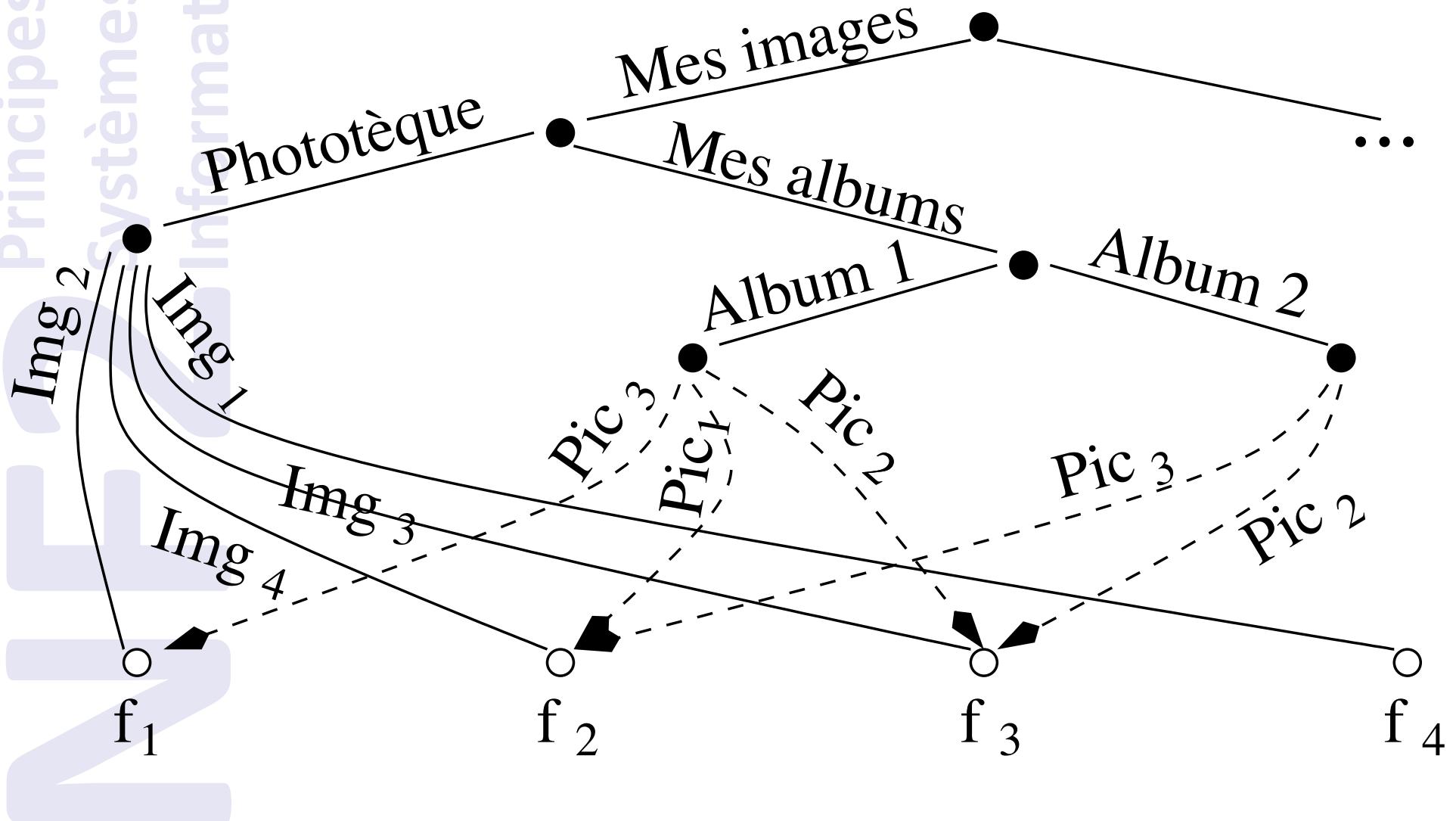
# Le modèle arbre



# Le modèle graphe (1)



# Le modèle graphe (2)



# Un arbre-graphe



Figuier des Banians. (Dessin fait d'après nature, sur les bords du Gange, par M. de Bérard.)

La Nature, 1877

# Nommage hiérarchique

## Nommage **hiérarchique**

- $d^1 / d^2 / \dots d^n / f$  ( $\equiv d^1 \setminus d^2 \setminus \dots d^n \setminus f$ )
- $d^i$  = **nom de répertoire**
- $f$  = **nom de fichier**
- $d^1 / d^2 / \dots d^n / f$  = **chemin d'accès (path)**
- [**racine**] /  $d^1 / d^2 / \dots d^n / f$  = chemin **absolu**
- $d^1 / d^2 / \dots d^n / f$  = chemin **relatif**

# Conclusion modèle (1)



- Des **fichiers**
- Des **répertoires** servant de **fractions** de **dictionnaires**
- Un fichier contient des **données**
- Un répertoire contient des **triplets** (**type "nom" adresse**)
  - certains noms sont des noms de répertoires...
  - ...d'autres des noms de fichiers

# Conclusion modèle (2)

• Organisation approximativement en **arbre**...

...plus précisément en **graphe**

...attention aux approximations populaires !

# Conclusion modèle (3)



## Principe du **nommage hiérarchique**

- internet, web, ...
- applications multimédia, ...
- menus, options, configurations, ...

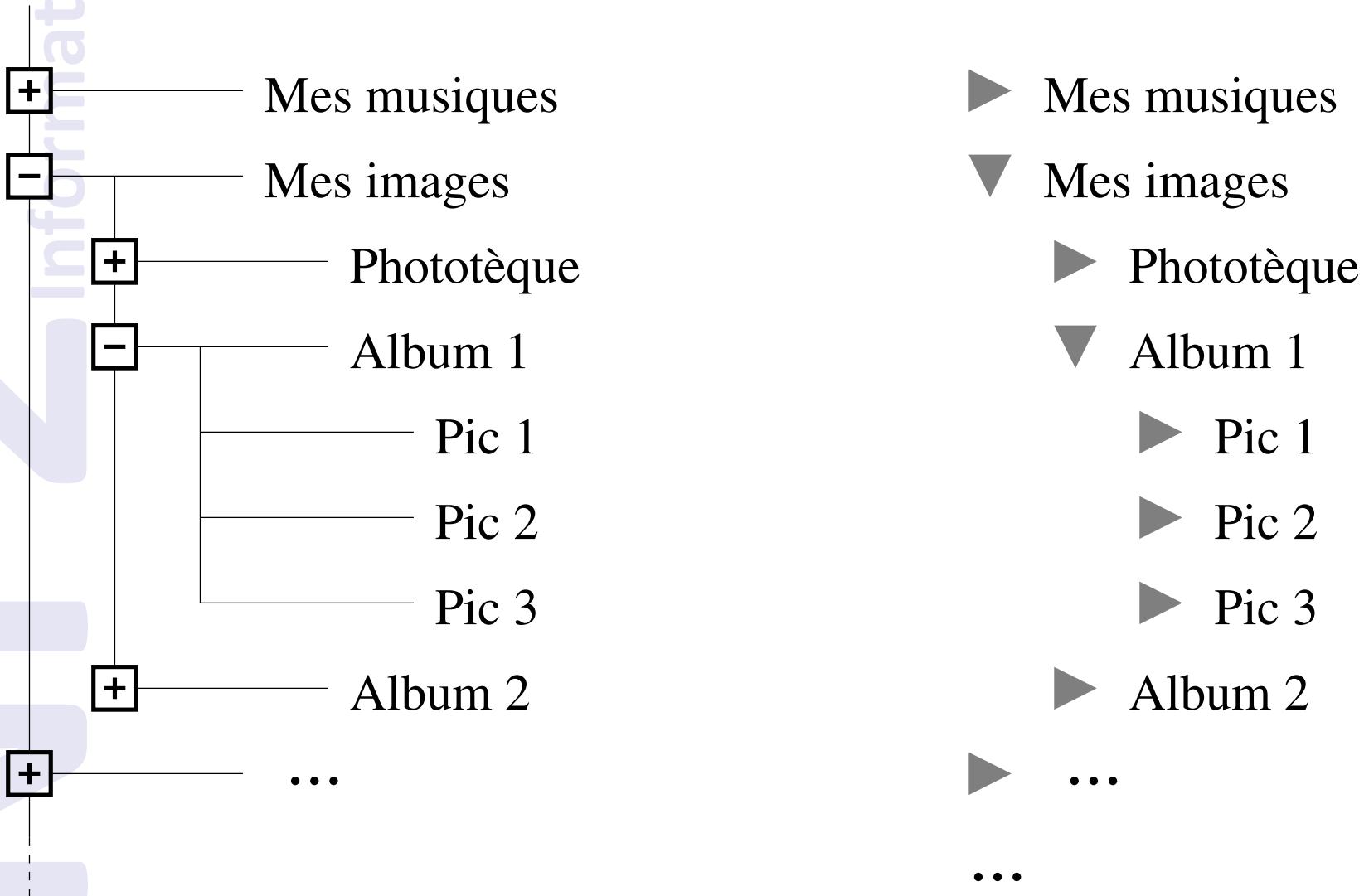
# INF2 Principes des Systèmes Informatiques

## Vue

## Vue

- Ce qu'un système informatique montre des fichiers et répertoires
  - vue **graphique**
    - navigateur de fichier
    - Windows, MacOS, ..., mais aussi Linux
  - vue **commande**
    - langages de script
    - UNIX, Linux, MS-DOS, ...

# Vue graphique (1)



## Vue graphique (2)

- **Très largement utilisée**  
dans les interfaces graphiques  
là où le nommage hiérarchique est utilisé
- **Rend mal compte du modèle graphe** 
- **Peu commode pour des manipulations sophistiquées**

# Vue commande

- Approche linguistique
  - des **noms** et des **verbes**
  - **commande nom<sub>1</sub> [nom<sub>2</sub> ...]**

**les liens entre informatique  
et linguistique sont nombreux**

# Le modèle de la vue commande

- Le répertoire **courant** (., *working directory*)
  - toujours implicite
- Le répertoire **racine** (/, *root*)
  - aussi implicite
- Le répertoire **d'accueil** (~, *home directory*)
  - lui aussi implicite
- Le répertoire **parent** (.., *parent directory*)
  - encore implicite

# Commandes de visualisation (1)

- **ls**
  - liste le contenu du **répertoire courant**
- **ls *chemin***
  - indique si il existe quelque chose à l'adresse ***chemin***
- **chemin = absolu, relatif, ..., ~**

# Commandes de visualisation (2)

- **cd *chemin***
  - vérifie que ***chemin*** mène à un répertoire
  - désigne ***chemin*** comme nouveau répertoire courant
- **cd**
  - désigne le répertoire d'accueil comme nouveau répertoire courant
- **chemin = absolu, relatif, .., ~**

# Commandes de visualisation (3)

- **pwd**
  - affiche un chemin absolu du répertoire courant
- **cat *chemin***
  - ou **application *chemin***, ou ...
  - affiche le contenu du fichier placé à l'adresse ***chemin***
- **chemin = absolu, relatif, .., ~**

# Contrôle

# Contrôle (1)

- Des commandes pour modifier le contenu du système de fichiers
- `mkdir nom`
  - crée un répertoire de nom *nom* dans le répertoire courant

## Contrôle (2)

- touch ***nom***
  - vérifier qu'il n'existe pas de fichier de nom ***nom***
  - crée un fichier de nom ***nom*** dans le répertoire courant

## Contrôle (3)

• **rm *nom***

- supprime l'entrée ***nom*** du répertoire courant

• **rmdir *nom***

- vérifie que ***nom*** est celui d'un répertoire vide
- supprime l'entrée ***nom*** du répertoire courant

## Contrôle (4)

- `mv nom1 nom2`

- renomme l'entrée **nom<sub>1</sub>** du répertoire courant en **nom<sub>2</sub>**

- rm (remove) n'efface pas le fichier sur le disque...

- ...il le « détache »

- mv (move) ne déplace pas le fichier sur disque...

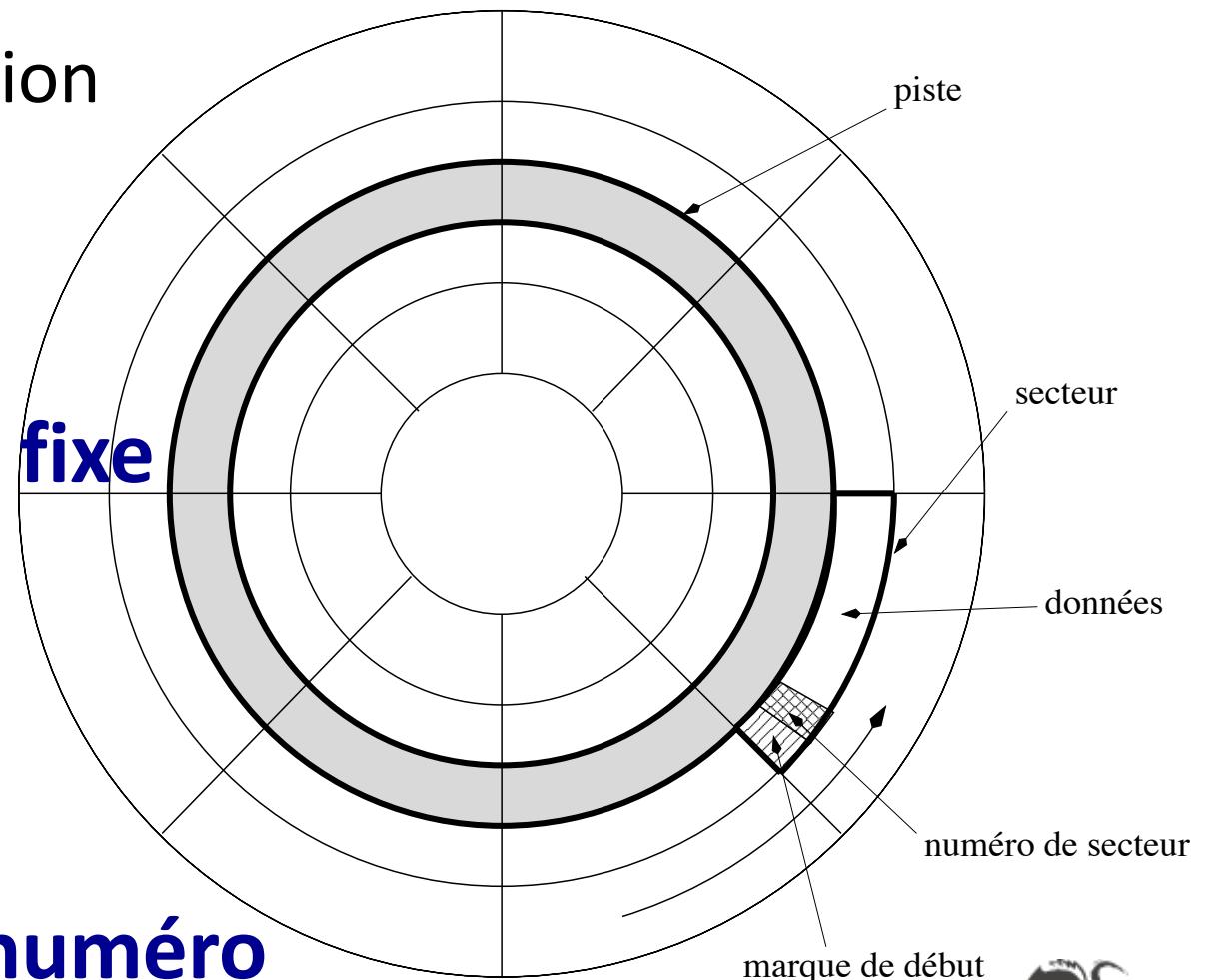
- ...il le « rattache ailleurs »

# INF2 Principes des Systèmes Informatiques

## Challenge

## Structure d'un disque dur

- Ensemble « plat » de **secteurs**
  - pas de structuration
- Secteurs de **taille fixe**
  - ex. 2048 octets
- Désignés par un **numéro**



# Rappel fonctionnement disque dur

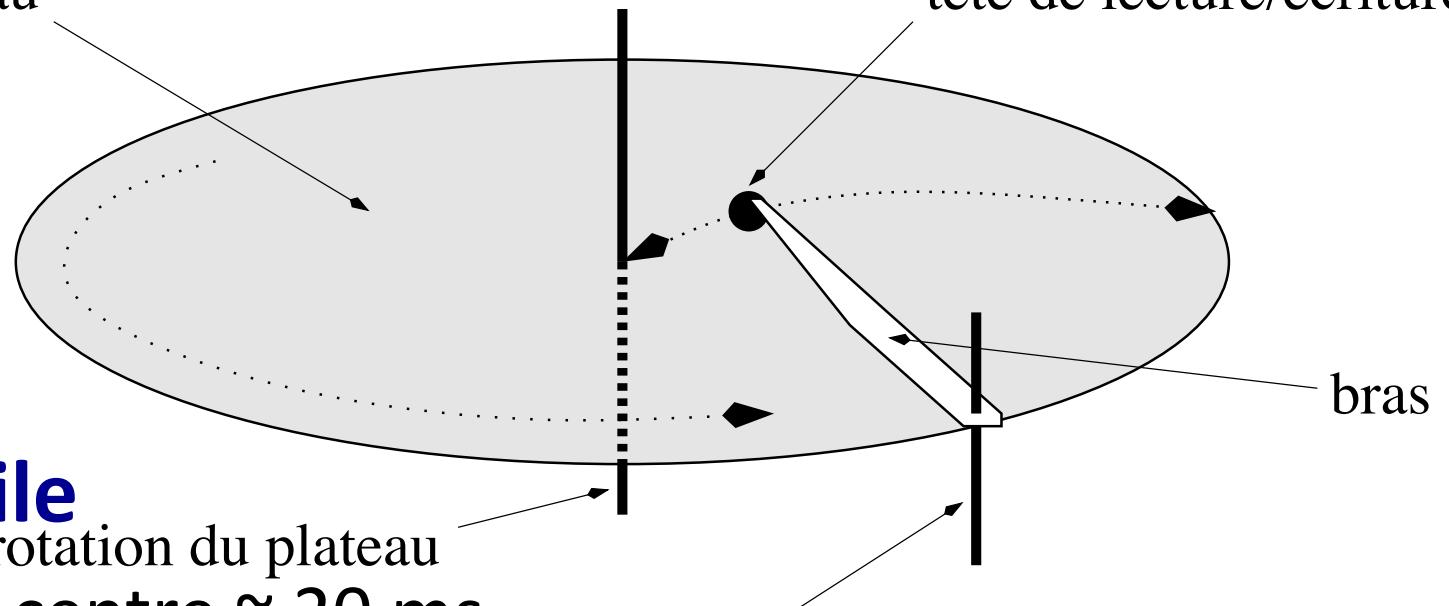
- Plateau **rotatif**

- 3000 à 15000 tr/min

- temps de recherche et temps d'opération

plateau

tête de lecture/écriture



- Bras **mobile**

- axe de rotation du plateau

- AR bord-centre  $\sim 20$  ms

axe de rotation du bras

- temps de latence

# Le challenge

- Concilier les deux ensembles de contraintes

Système de fichiers	Disque dur magnétique
taille arbitraire : N octets	taille fixe : 512 octets
nom arbitraire : MesVacances	nom conventionnel : numéro
structure complexe de répertoires	structure plate
unité d'accès fine : l'octet	unité d'accès grossière : le secteur
accès rapide : $\sim vs$	accès lent : $\sim ms$

# INF2 Principes des Systèmes Informatiques

## Réponses

# Approche progressive

- Challenge du **nommage**
  - désignation des fichiers
- Challenge de la **taille**
  - description des fichiers
- Challenge du **temps d'accès**
  - principe de localité

# Désignation des fichiers (1)

- Hypothèses
  - fichiers de taille fixe : un secteur
  - répertoire de taille fixe : un secteur
- Noter les secteurs libres
  - **liste des secteurs libres**
  - un **bitmap** rangé dans un **secteur conventionnel**, ex. secteur n° 0

# Désignation des fichiers (2)

- Ranger des triplets **(type nom n° secteur)** dans les répertoires
  - (dir "nomdir" n° secteur)
  - (file "nomfich" n° secteur)
- Représenter le répertoire **racine** dans un **secteur conventionnel**, ex. le secteur n° 1

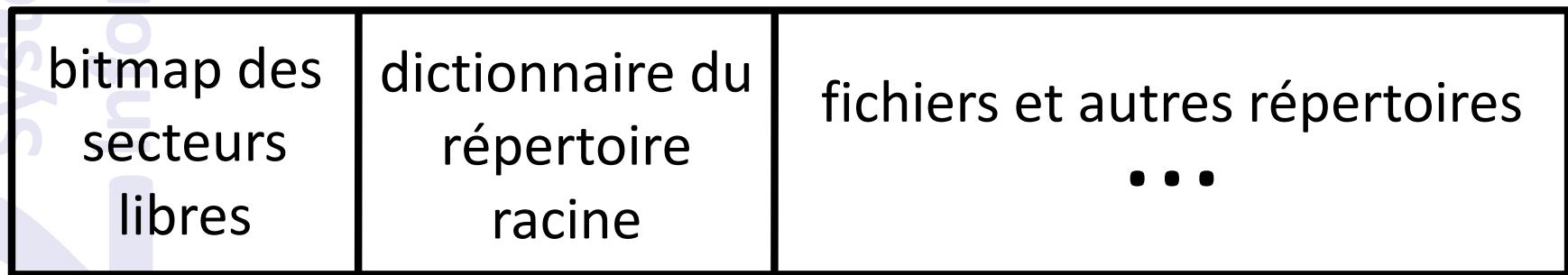
# Désignation des fichiers (3)

- Organisation naïve du disque

secteurs n° 0

n° 1

...



- Exemple introductif

n° 0

n° 1

n° 2

n° 3

n° 4

n° 5

n° 6

n° 7

n° 8

bitmap des secteurs libres	((dir "g" 2) (file "d" 8) (file "n" 7))	((file "g" 4) (file "m" 5) (dir "n" 1) (dir "d" 3))	((file "g" 6) (file "d" 7))	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	...
----------------------------	---	--	--------------------------------	-------	-------	-------	-------	-------	-----

# Désignation des fichiers (4)

- Le disque contient donc des **données**...  
...contenu des fichiers
- ...et des **méta-données**...  
...données sur l'organisation des données
  - répertoires
  - bitmap des secteurs libres

# Description des fichiers (1)

- Hypothèses
  - fichiers de **taille quelconque** :  
un ou **plusieurs** secteurs
  - répertoire de **taille quelconque** :  
un ou **plusieurs** secteurs
- Noter quels secteurs sont occupés par chaque fichier

# Description des fichiers (2)

- Secteurs contigus ou dispersés ?
- Secteurs **contigus**
  - facile à décrire...
  - (**n° premier secteur    nombre de secteurs**)
  - ...mais trop **rigide**
    - si le fichier grandit ?
    - **fragmentation** ?
- Secteurs **dispersés**
  - plus difficile à décrire...
  - ...mais **flexible**

# Fragmentation

- Les fichiers sont **créés puis détruits sans ordre**
  - cela « mite » le disque dur
  - les secteurs libres sont finement mélangés avec les secteurs occupés
- On peut avoir  $\gg N$  secteurs libres  
**mais aucun trou de taille  $> N$** 
  - l'allocation de secteurs contigus ne marche pas !

# Description des fichiers (3)



Ranger des **descripteurs d'implantation de fichiers** dans des secteurs conventionnels

- Dans les répertoires noter les **numéros de descripteurs** au lieu du numéro de secteur

(dir "nomdir" n° descripteur)

(file "nomfich" n° descripteur)

# Description des fichiers (4)

- Les descripteurs d'implantation de fichiers doivent être **tous de même taille...**



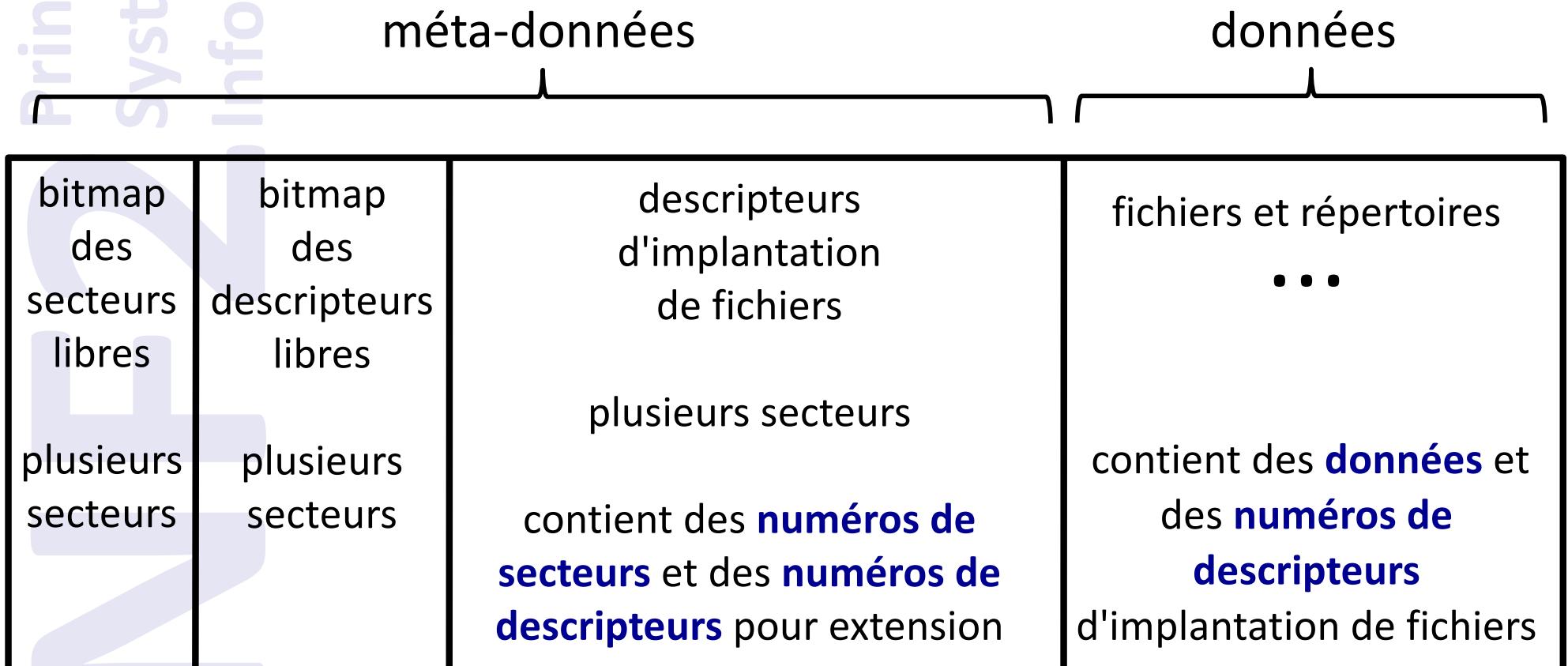
...chaque secteur en contient plusieurs

– un fichier peut occuper plus de secteurs  
qu'un secteur peut en désigner...

...prévoir qu'un descripteur puisse **s'étendre  
dans un autre descripteur**, etc

# Description des fichiers (5)

- Organisation du disque



# Description des fichiers (6)

- Toujours plus de **méta-données**
  - bitmap des secteurs libres
  - bitmaps des descripteurs libres
  - descripteurs d'implantation des fichiers

# Temps d'accès (1)

- Dans le système complet, lire un fichier dans un répertoire supposé lu exige de
  - lire un **descripteur d'implantation**...
  - ...et peut-être son **extension**, etc
  - ...et les **secteurs de données** occupés par le fichier
- Ces opérations sont **coûteuses**,  
**peut-on les éviter ?**

## Temps d'accès (2)

- Idée : **amortir** les opérations coûteuses
  - faire que le résultat d'une opération puisse **resservir** afin de ne pas la refaire
- Utiliser une **mémoire cache**...  
...basée sur le **principe de localité**

# Principes de localité

- **Prévoir le futur en examinant le passé**

- Deux variantes



- principe de **localité temporelle**...

- ...concerne la probabilité de **répéter** une opération

- principe de **localité spatiale**...

- ...concerne la probabilité d'exécuter une opération **voisine** d'une autre

# Principe de localité temporelle

- Un calcul qui a été fait **récemment** a une **forte probabilité** d'être redemandé prochainement
- Une donnée qui a été **accédée récemment** a une **forte probabilité** d'être **ré-accédée prochainement**
- La **mémoire cache** stocke les **Résultats des calculs passés** au cas où ils **resserviraient**

# Principe de localité spatiale

- Une donnée **proche d'une donnée** qui a été **accédée récemment** a une **forte probabilité** d'être **accédée prochainement**



- La lecture **spéculative**

- lire toute une piste plutôt que un secteur seul...  
...car les secteurs voisins risquent d'être lus à leur tour
  - lire un secteur plutôt que un mot...  
...car les mots voisins risquent d'être lus à leur tour



# Mémoire cache (1)

- Mémoire destinée à stocker le contenu de secteurs lus sur le disque
  - Structurée en pages
    - **taille de page = taille de secteur**
    - **nb de pages << nb de secteurs**
    - seuls les secteurs **les plus récemment lus** peuvent résider en cache

## Mémoire cache (2)

```
LireSecteur(s)
{ si s ∈ Cache
  alors LireCache(s)
  sinon ÉcrireCache(s, LireDisque(s))
}
```

- Et les **écritures** sur disque ?
- Et quand le cache est **plein** ?

# Mémoire cache (3)

- Et les **écritures** sur disque ?
  - Écrire dans le cache est  **$10^6$  fois plus rapide** que sur le disque...
    - ...mais alors le contenu du cache **se désynchronise** de celui du disque
      - Cache(s)  $\neq$  Disque(s)
      - ...et c'est le cache le plus à jour
      - ...le disque est en retard
- ...il faudra le resynchroniser un jour !**

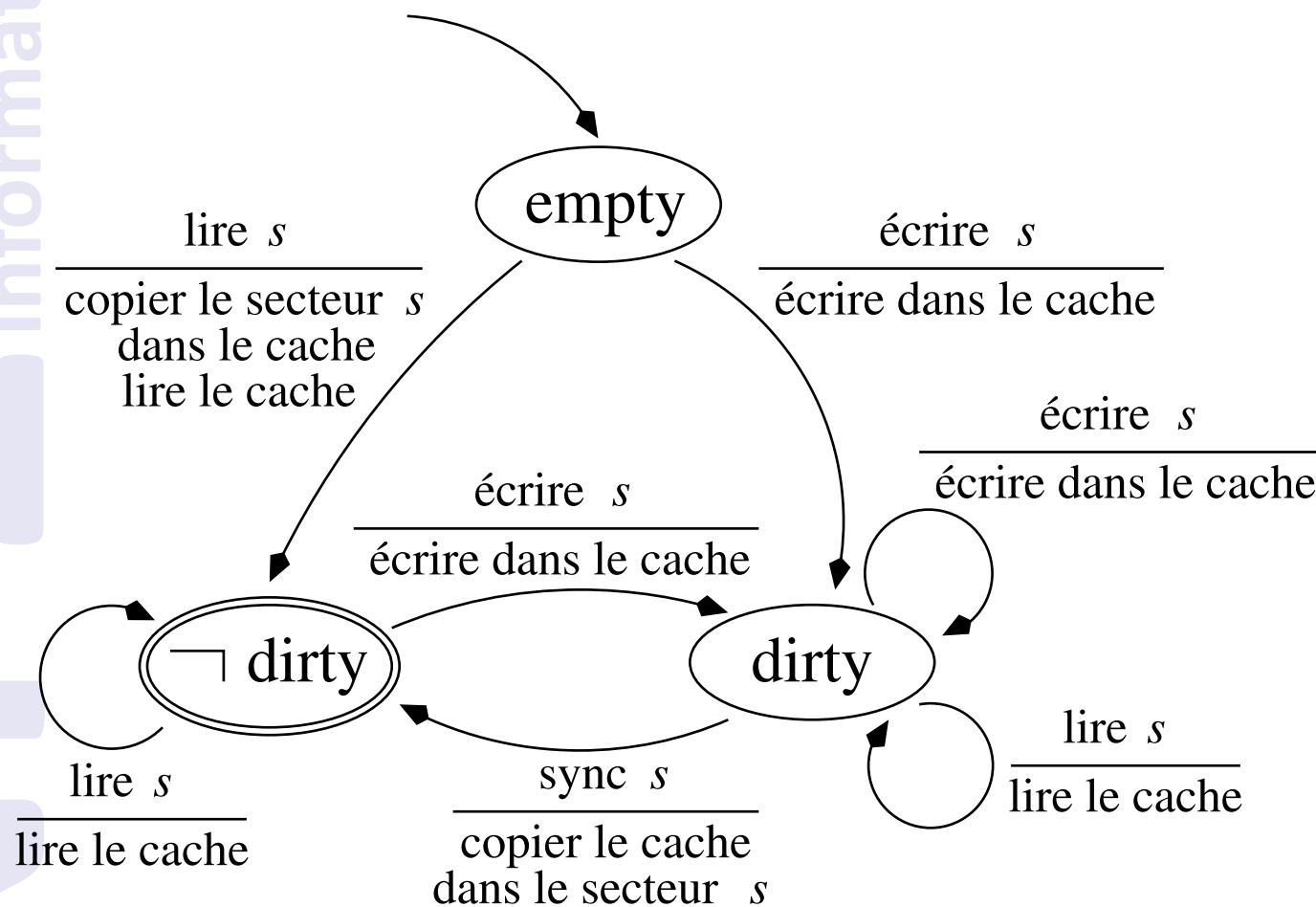
## Mémoire cache (3)

- On ajoute un bit par page du cache pour noter si son contenu est **désynchronisé**
  - bit ***dirty***
  - on dit qu'une page désynchronisée est ***sale***

# Mémoire cache (4)

```
ÉcrireSecteur(s, d)
{ ÉcrireCache(s, d) ;
  Dirty(s) = vrai
}
```

```
ResynchroniserSecteur(s)
{ si Dirty(s)
  alors ÉcrireDisque(s, LireCache(s))
}
```



## Mémoire cache (4)

- Et quand le cache est **plein** ?
  - Il faut sortir du cache un secteur moins récemment accédé
    - sacrifier** un secteur
      - il faut garder **les secteurs les plus récemment utilisés** dans le cache...
      - ...mais c'est **trop compliqué de noter l'heure** de chaque lecture / écriture

# Mémoire cache (5)

- On ajoute un bit par page du cache pour noter si son contenu est récemment accédé
  - bit **used**
  - on dit qu'une page récemment accédée est **utilisée**
  - le bit used est remis à faux régulièrement...  
...aucune page **s** avec **Used(s) = faux**  
n'a été utilisée depuis la dernière remise à faux...  
...elle peut être **sacrifiée**

# Mémoire cache (6)

LireSecteur(s) { ... ; Used(s) = vrai }

ÉcrireSecteur(s,d) { ... ; Used(s) = vrai }

RéinitialiserUsed()

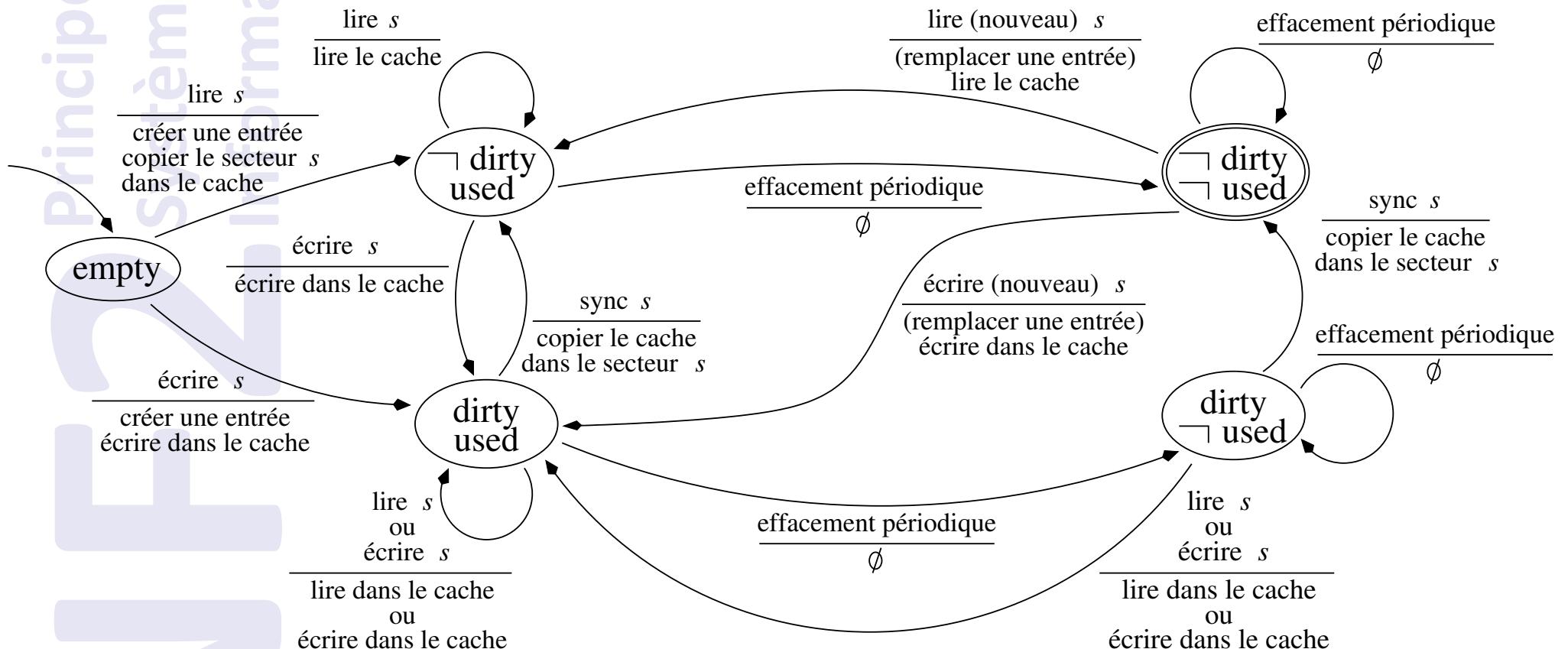
{ pour tout s faire Used(s) = faux }



## Mémoire cache (7)

- Recherche d'un secteur à sacrifier
  - **ne pas sacrifier** un secteur **s** tel que  
**Dirty(s) = vrai**  
exigerait de **resynchroniser**
  - **préférer** un secteur **s** tel que  
**Used(s) = faux  $\wedge$  Dirty(s) = faux**  
pas accédé récemment ni sale
  - sinon un secteur **s** tel que  
**Used(s) = vrai  $\wedge$  Dirty(s) = faux**

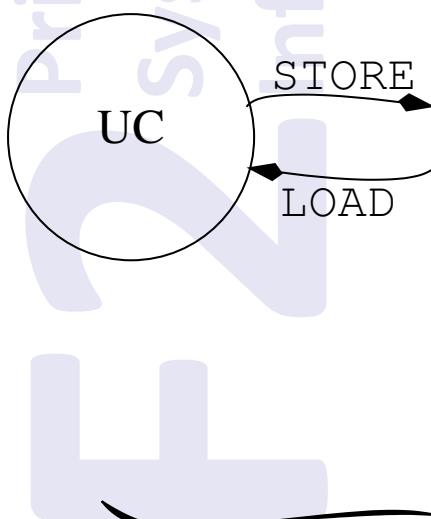
# L'automate des dirty bits et used bits



# Mémoire cache (8)

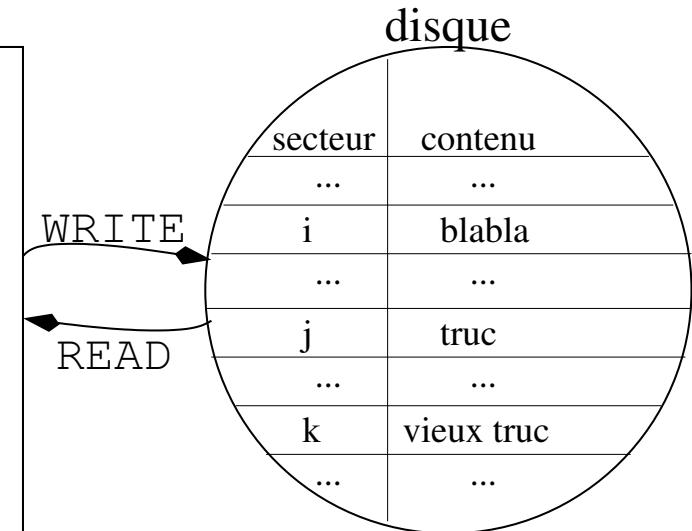
- **Quand resynchroniser ?**
- Cycliquement, par prudence
  - si panne de disque désynchronisé alors **écriture perdue**
- À la demande
  - commande **sync**
- En éjectant proprement un disque amovible (ex. clé USB)

## Mémoire cache (9)



mémoire

cache			
secteur	dirty	used	contenu
...	...	...	...
i	vrai	vrai	blablaBLA
...	...	...	...
j	faux	vrai	truc
...	...	...	...
k	faux	faux	vieux truc
...	...	...	...



copies de secteurs

originaux des secteurs

accès rapide à des copies de secteurs

accès lent aux originaux si besoin

# Mémoire cache (9)

- Bilan
  - **désynchronisation**  
entre accès système de fichiers  
et accès disque
  - moins d'accès disque que d'accès fichier
  - pas dans le même ordre

**sujet du 1<sup>er</sup> TP**

# Conclusion (1)

- Un système complexe...
  - description complexe...  
**modèle – vue – contrôle**
  - réalisation complexe...  
**méta-données**  
**principe de localité**  
**désynchronisation**
- ...présent dans la plupart des systèmes informatisés



## Conclusion (2)

- Modèle – vue – contrôle
- Commencer la description d'un système complexe par ses invariants...

le **modèle**

...ce qu'on en voit

la **vue**

...ce qu'on en fait

le **contrôle**

## Conclusion (3)

- Des principes très puissants
  - **principe de localité**
  - **méta-données**
  - **désynchronisation**