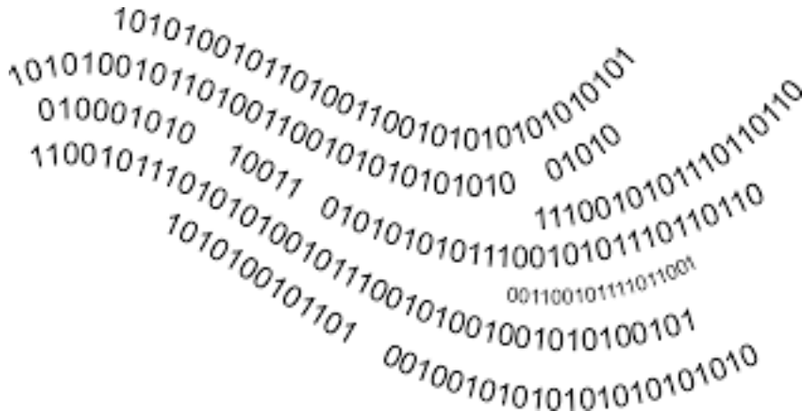
Comment trouver une aiguille dans une botte de foin

Amélie Gheerbrant
amelie@irif.fr
IRIF, Université Paris Diderot
Semaine d'accueil des L1
10 septembre 2018

(Figuré) *(Familier)* Se dit en parlant d'une chose que l'on cherche, mais qui est très difficile à trouver, à cause de sa petitesse.

- *Essayer de trouver quelqu'un dans une foule immense, c'est* ***chercher une aiguille dans une botte de foin****.*
- *Je ne peux tout de même pas tous les soupçonner, cela revient à* ***chercher une aiguille dans une botte de foin****.* — (Marie Liandier, *Opération Puy Mary*, 2006)



Big Data

- Le Big Data c'est quoi ?
- D'abord, des 0 et des 1 (bits)
- Unité de base : l'octet (suite de 8 bits)
- Ordres de grandeur : ko, Mo, Go, etc

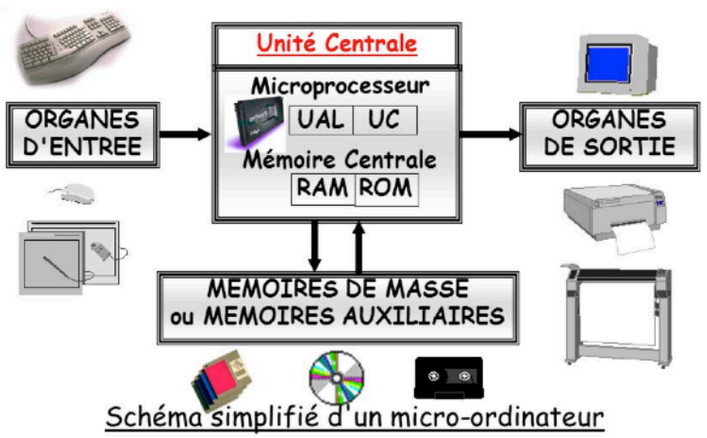
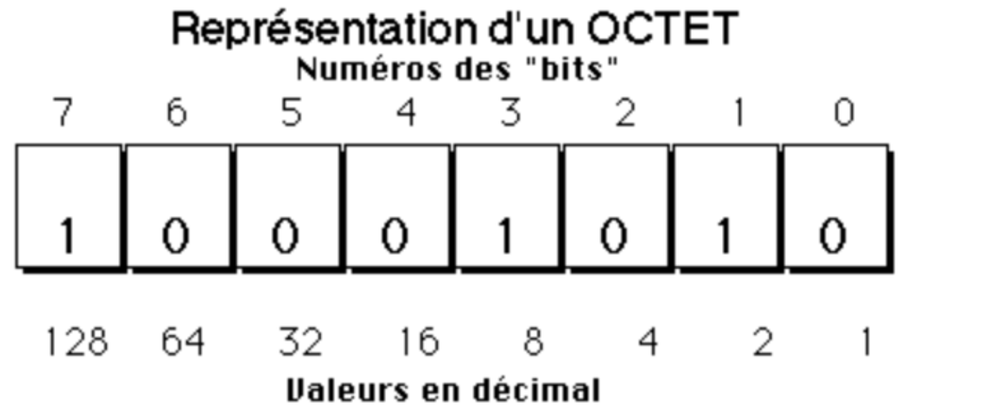


Schéma simplifié d'un micro-ordinateur

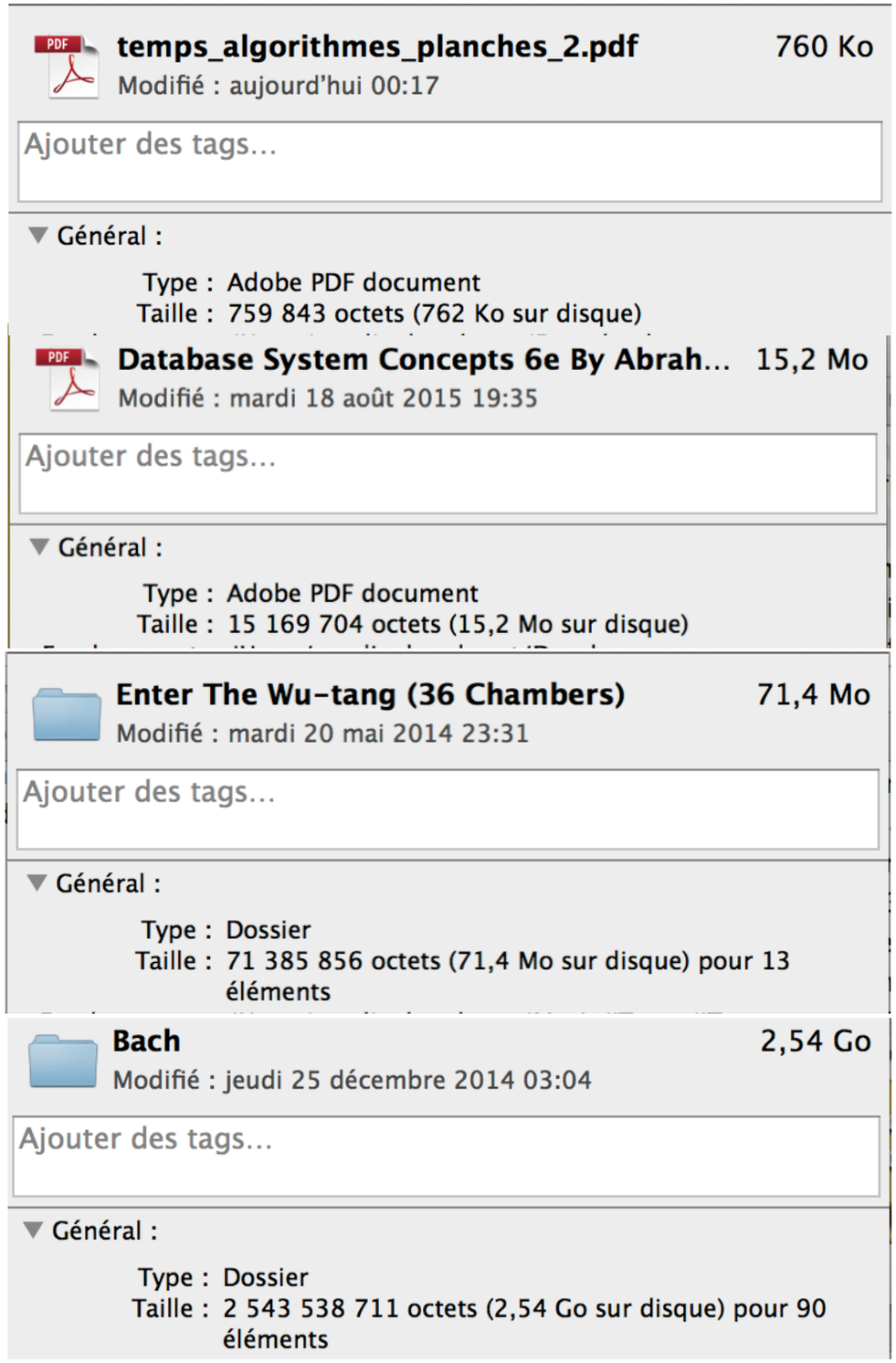


138 (décimal)

10001010 (binaire)

Ordres de grandeur

Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
exaoctet	Eo	10^{18}
zettaoctet	Zo	10^{21}
yottaoctet	Yo	10^{24}



Ordres de grandeur

Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
exaoctet	Eo	10^{18}
zettaoctet	Zo	10^{21}
yottaoctet	Yo	10^{24}

Tous les livres jamais écrits : quelques centaines de téraoctets (en texte brut)

Ordres de grandeur

Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
exaoctet	Eo	10^{18}
zettaoctet	Zo	10^{21}
yottaoctet	Yo	10^{24}

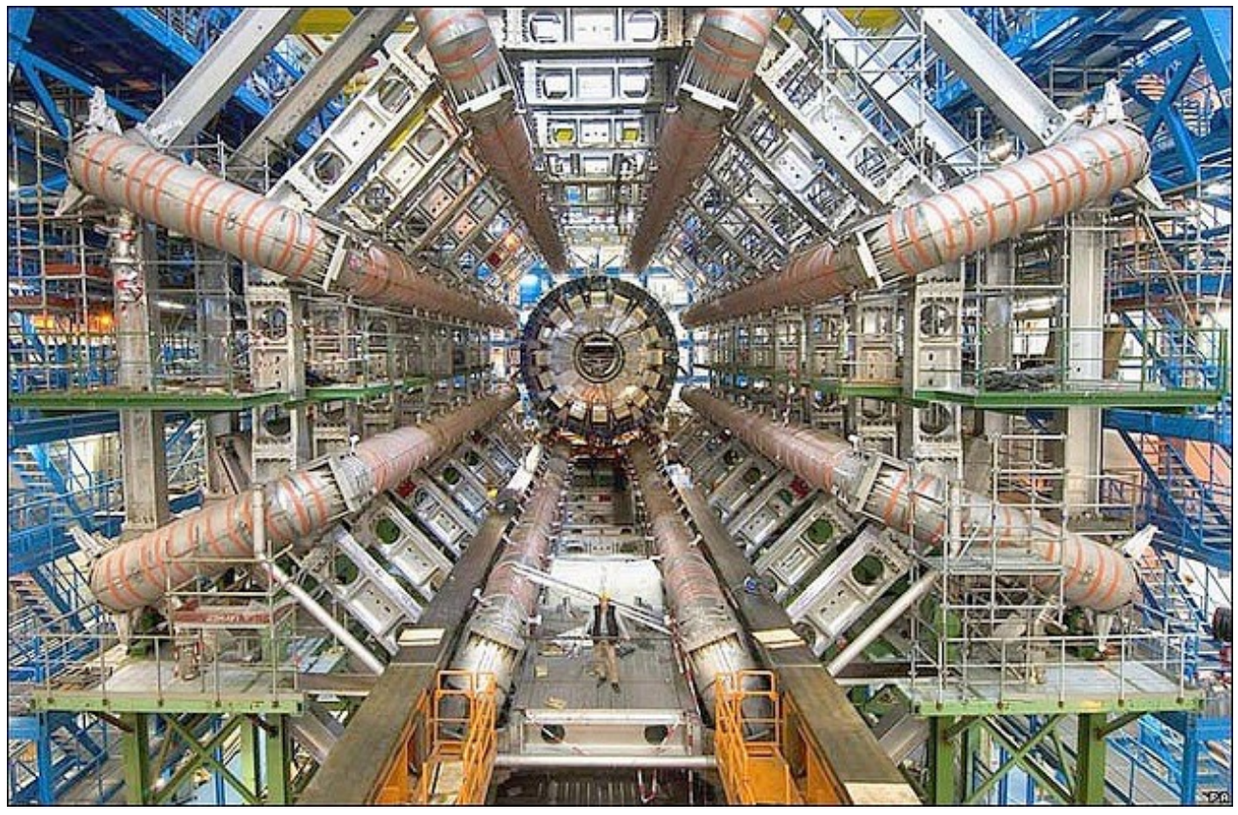
100 téraoctet :
volume quotidien des contenus mis en ligne sur Facebook en 2013

460 téraoctets :
base de données clients de Wal-Mart, chaîne américaine de supermarché

Ordres de grandeur

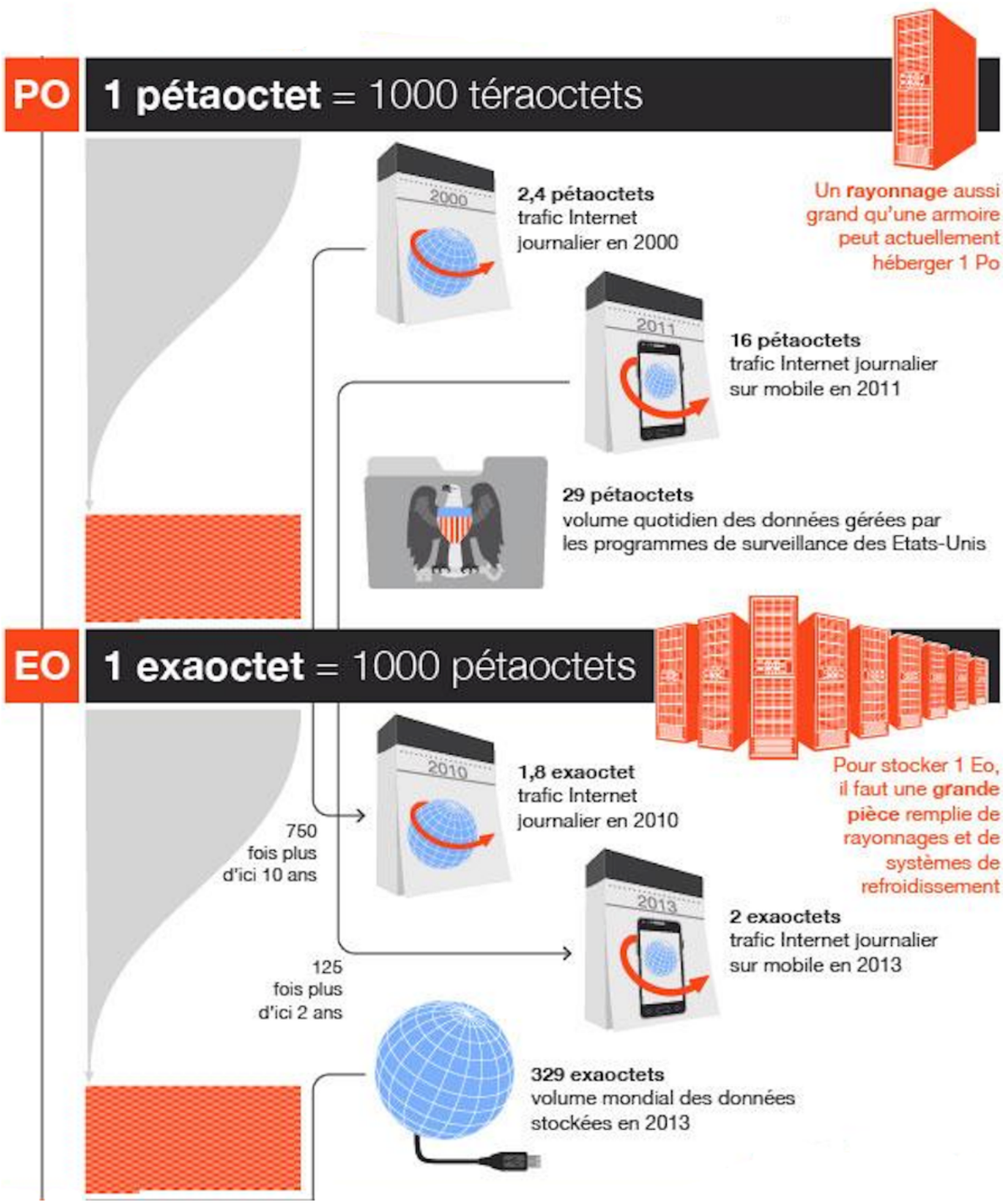
Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
exaoctet	Eo	10^{18}
zettaoctet	Zo	10^{21}
yottaoctet	Yo	10^{24}

La quantité de données produite par l'accélérateur de particules du CERN en un jour : une centaine de pétaoctets



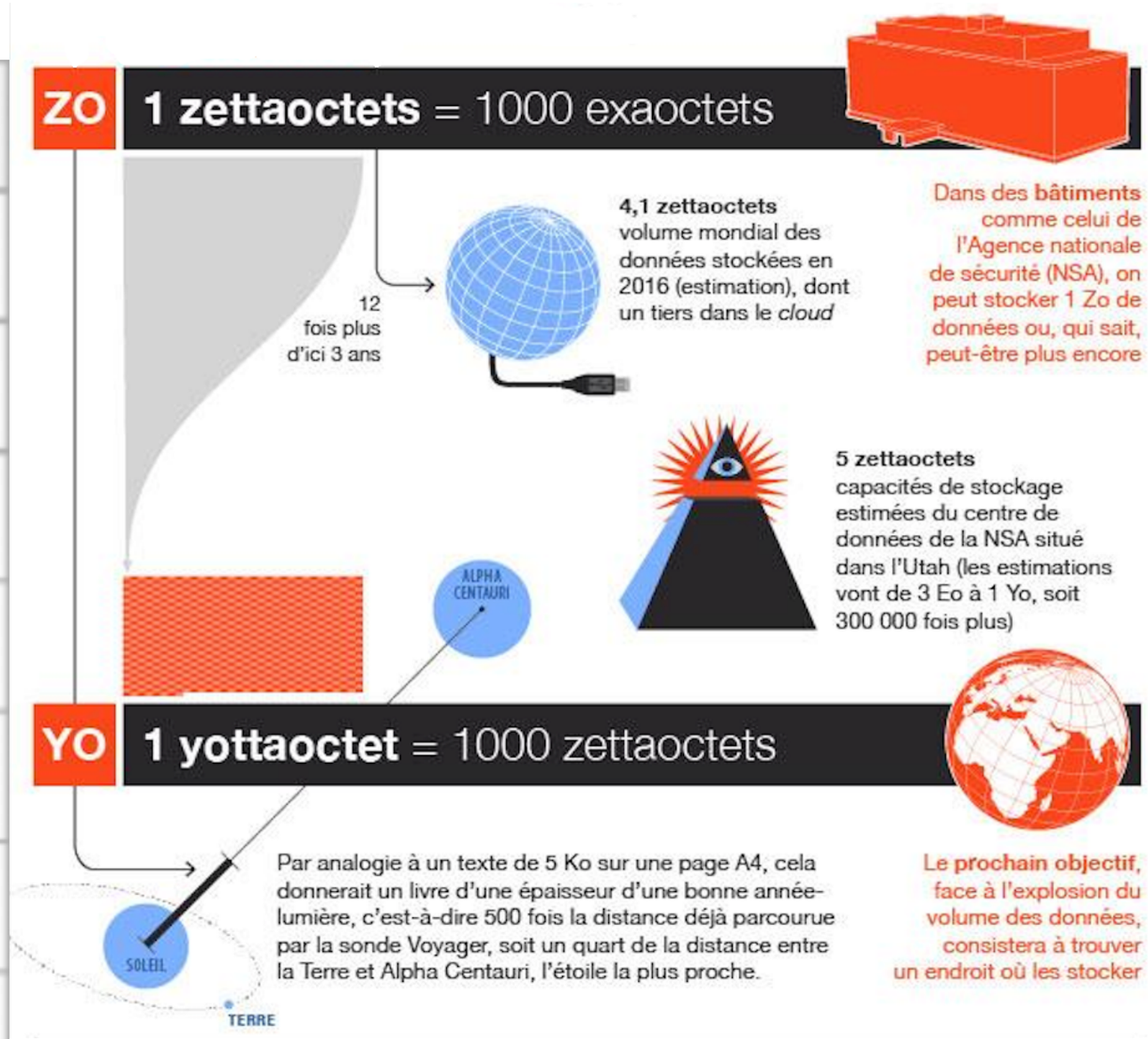
Ordres de grandeur

Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
exaoctet	Eo	10^{18}
zettaoctet	Zo	10^{21}
yottaoctet	Yo	10^{24}



Ordres de grandeur

Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
exaoctet	Eo	10^{18}
zettaoctet	Zo	10^{21}
yottaoctet	Yo	10^{24}



Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard

Marcelo Arenas
Department of Computer Science
PUC Chile
marenas@ing.puc.cl

Sebastián Conca
Department of Computer Science
PUC Chile
saconca@puc.cl

Jorge Pérez
Department of Computer Science
Universidad de Chile
jperez@dcc.uchile.cl

Bases de données

- Dans Big Data il y a « data » : données
- Organisation sous forme de bases de données : Facebook, Amazon, Twitter, Instagram, Discogs, etc
- Gestion via un SGBD (Système de Gestion de Bases de Données) : système très puissant, capable de trouver une aiguille dans une botte de foin en une fraction de seconde

ORACLE DATABASE

IBM DB2

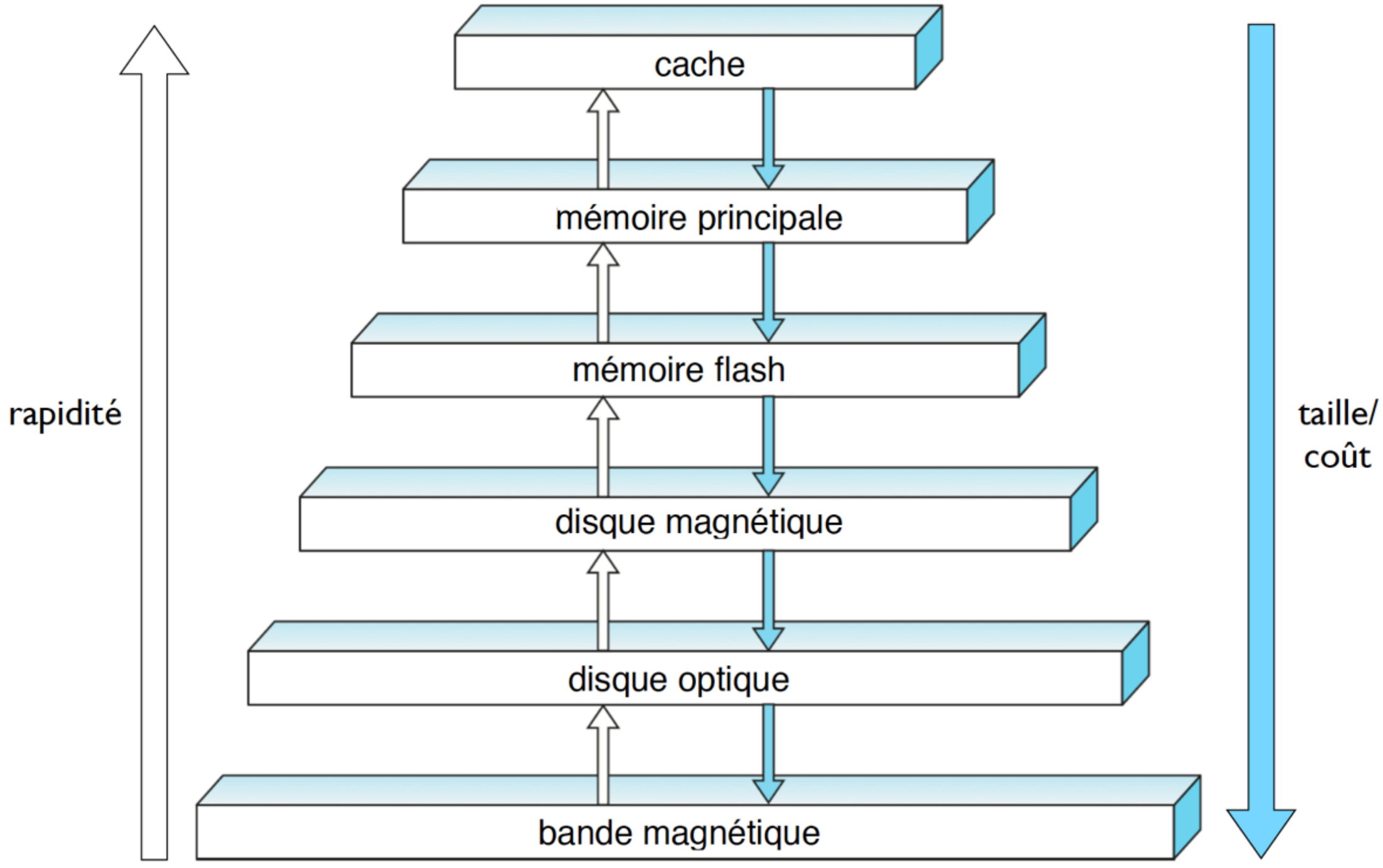
MySQL

PostgreSQL

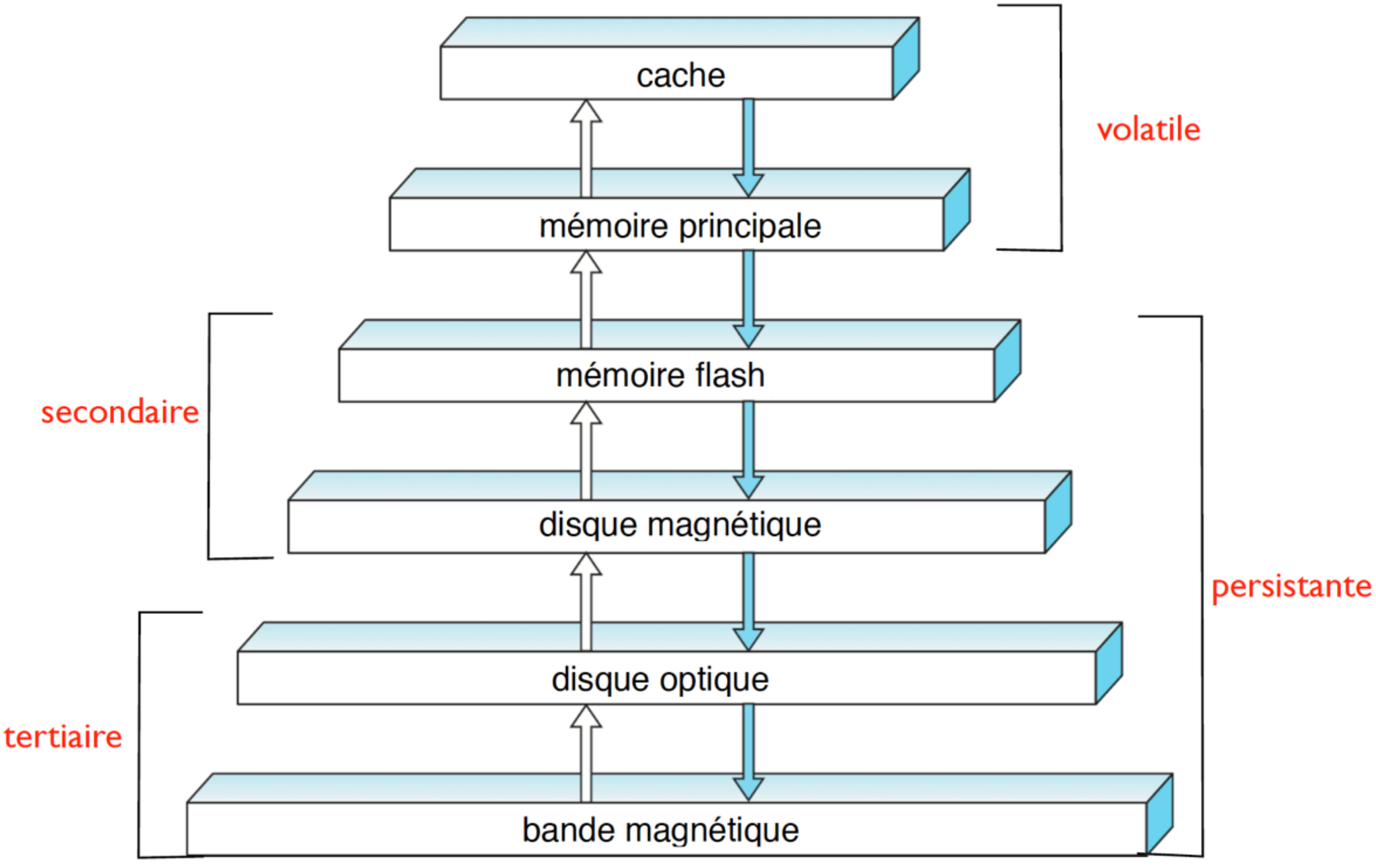
Stockage des données

- Le SGBD stocke les données dans une mémoire **persistante**
  - mémoire persistante: les contenus sont préservés même en l'absence d'alimentation.
  - mémoire volatile : perd sont contenu quand l'alimentation s'arrête
- Le SGBD interagit avec plusieurs type de mémoires pour gérer/traiter les données

Hiérarchie de mémoire



Hiérarchie de mémoire



Hiérarchie de mémoire

- **Cache** – la plus rapide et la plus coûteuse; volatile
  Maintient les infos les plus récemment utilisées
- **Mémoire principale** (RAM):
  - volatile
  - accès rapide (10s à 100s de nanosecondes)

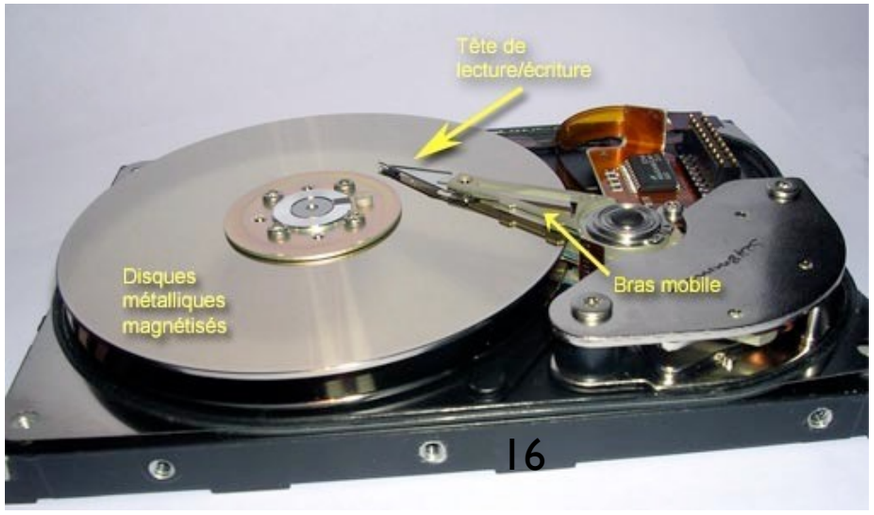
  - en générale trop petite (ou trop coûteuse) pour stocker la BD entière
    - en train de changer - on commence à parler de mémoires principales suffisamment grandes…
    - la BD doit de toute façon être également stockée en mémoire persistante
  - taille jusqu'à quelques Gigaoctets aujourd'hui
    - la taille augmente (et le coût diminue) d'un facteur 2 tous les deux/trois ans à peu près

Hiérarchie de mémoire

- **Mémoire *flash*** *(Electrically erasable programmable read-only memory)*
  - persistante
  - lectures presque aussi rapides qu'en mémoire principale
  - écritures plus lentes (quelques microsecondes)
  - largement utilisée dans les systèmes embarqués tels que cameras numériques, téléphones, clefs USB
  - remplace parfois aussi le disque

Hiérarchie de mémoire

- **Disque magnétique**
  - persistant, disque à écriture/ lecture magnétique
    - des pannes de disque peuvent détruire les données mais c'est rare
  - support principal de stockage long-terme : stocke typiquement la BD entière
  - accès beaucoup plus lent que la mémoire principale (millisecondes) : 10 000 à 100 000 fois plus lent que la RAM !!
  - <u>accès direct</u> – il est possible de lire/écrire du disque dans n'importe quel ordre (à la différence de la bande magnétique)
  - taille typiquement jusqu'à quelques teraoctets
    - augmente d'un facteur de 2 à 3 tous les deux ans environ

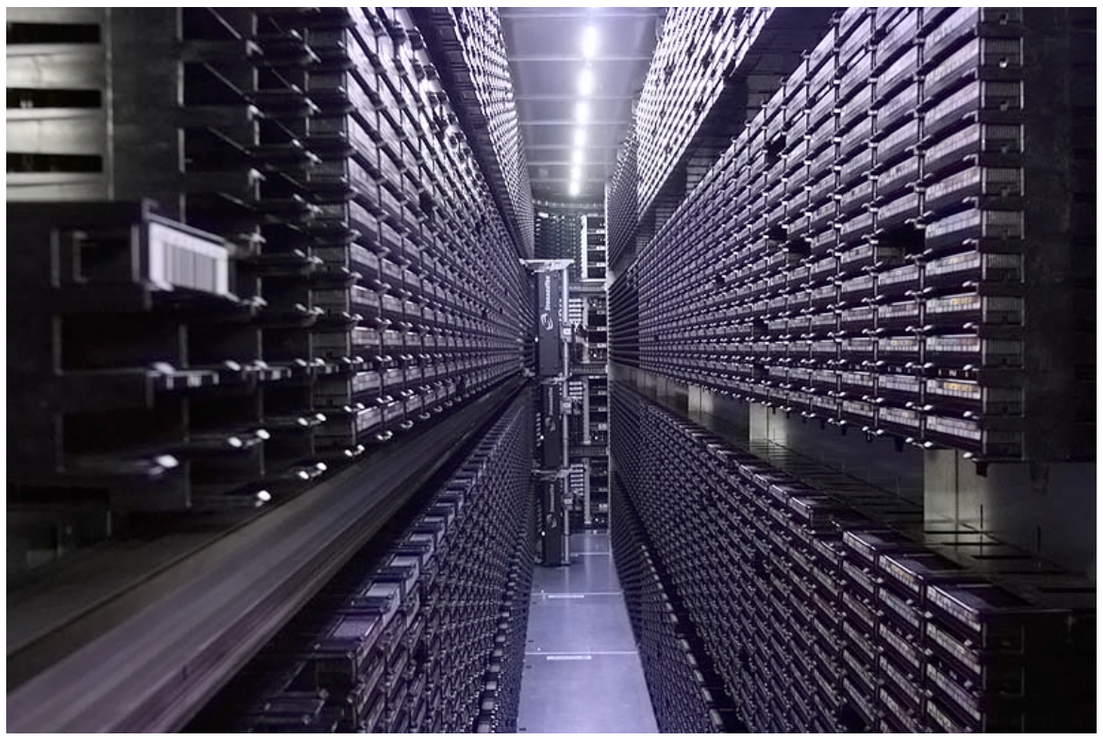


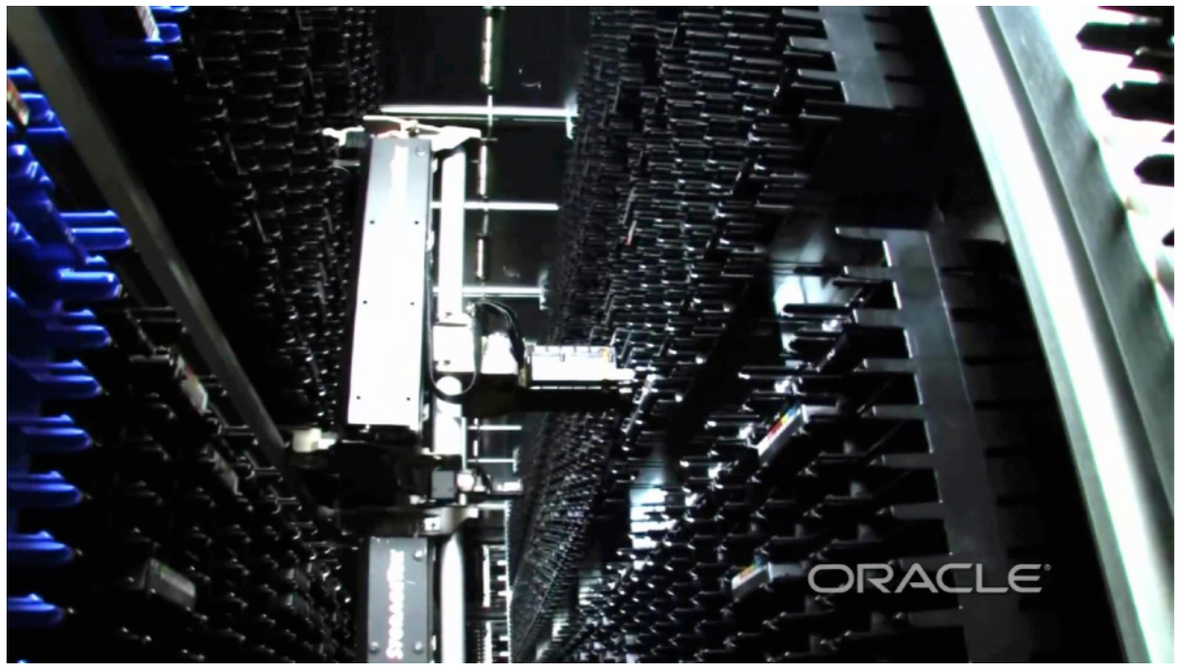
Hiérarchie de mémoire

- **Disque optique**
  - persistent, lectures/écritures optiques au laser
  - CD-ROM (640 MB) et DVD (4.7 to 17 GB) les plus populaires
  - disques Blu-ray : 27 GB à 54 GB
  - les types *Write-one, read-many* (WORM) sont utilisés pour archivage (CD-R, DVD-R, DVD+R)
  - Des versions à plusieurs écritures sont aussi disponibles (CD-RW, DVD-RW, DVD+RW, et DVD-RAM)
  - lectures et écritures plus lente qu'avec le disque magnétique
  - systèmes "Juke-box" pour le stockage de grande quantités de données

Hiérarchie de mémoire

- **Bande magnétique**
  - persistant, utilisé principalement pour *backup* et pour archivage
  - accès séquentiel – beaucoup plus lent que le disque
  - très grande taille par unité (bande de 40 à 300 GB disponibles)
  - beaucoup moins chère que le disque
  - "Juke-boxes" de bandes pour stocker des quantités massives de données
    - de centaines de terabytes (1 terabyte = 10^{12} bytes)
      à plusieurs petabytes (1 petabyte = 10^{15} bytes)





Stockages des données et types de mémoires

- Une BD est stockée en général en **mémoire secondaire** (disques magnétiques, flash)
  - raisons :
    - persistance, stockage de grande quantité à coût raisonnable
    - accès direct et en temps réel (par rapport à la mémoire tertiaire)
  - aujourd'hui BD très volumineuse ⇒ un seul disque ne suffit pas

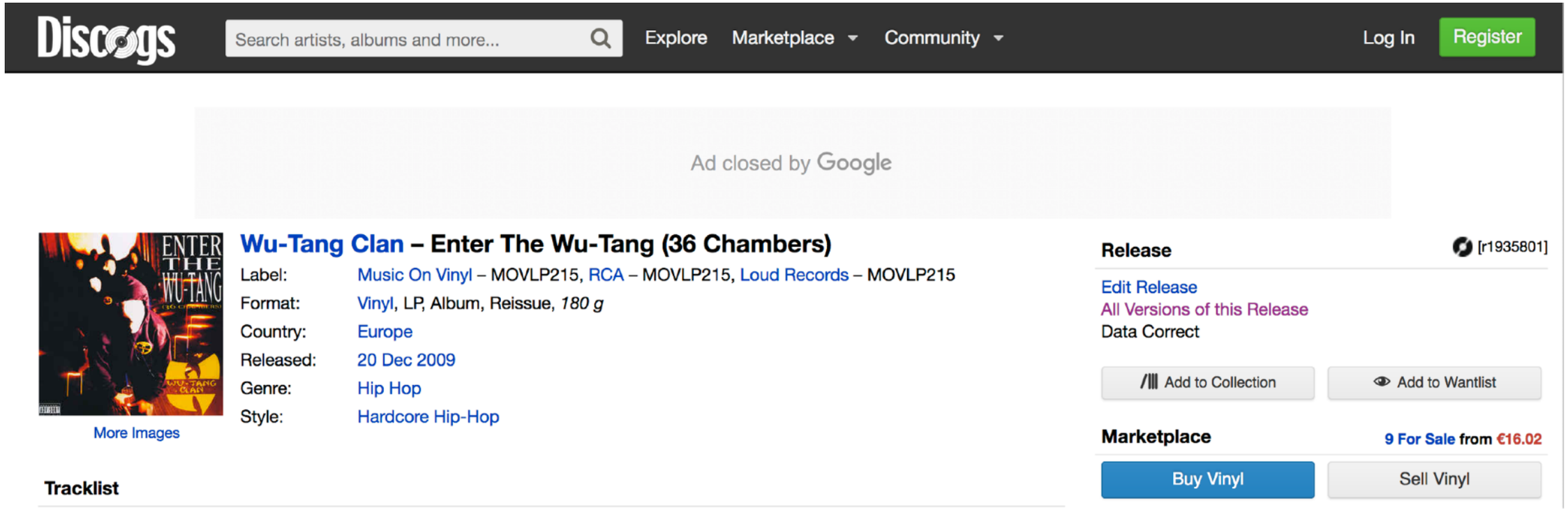
    de plus : besoin de fiabilité des données
    - RAID: *Redundant Arrays of Independent Disks*
    - à plus grande échelle : BD distribuée sur le réseau (*Cloud*)
- Les données sont déplacées du disque en **mémoire principale** pour l'accès, puis réécrites sur disque pour le stockage
- Les données plus modifiées et rarement lues sont transférées en **mémoire tertiaire** (disques optiques, bandes magnétiques) pour l'archivage

Stockages des données et types de mémoires

- Une BD est stockée en général en **mémoire secondaire** (disques magnétiques, flash)
  - raisons :
    - persistance, stockage de grande quantité à coût raisonnable
    - accès direct et en temps réel (par rapport à la mémoire tertiaire)
  - aujourd'hui BD très volumineuse ⇒ un seul disque ne suffit pas

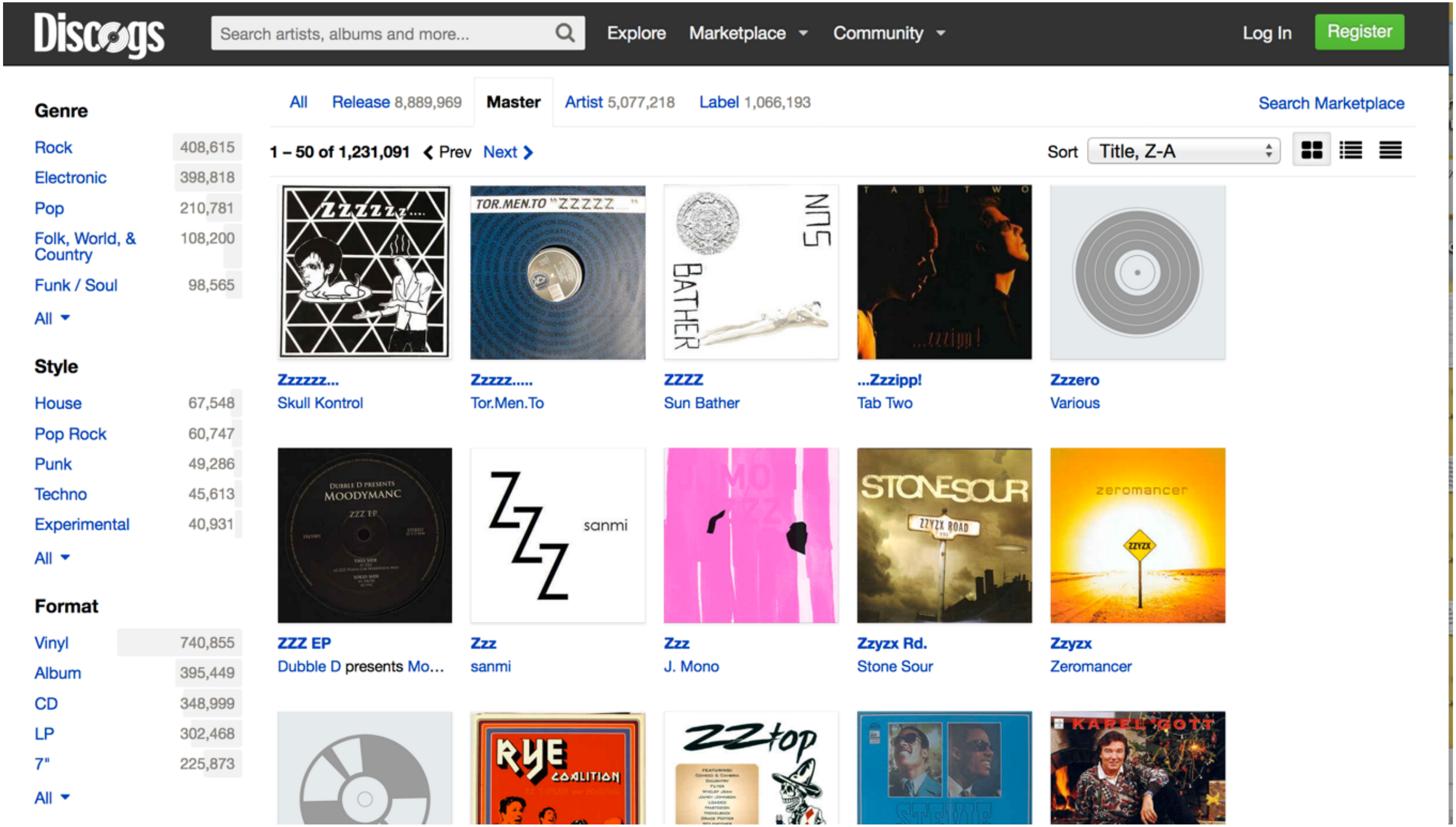
    de plus : besoin de fiabilité des données
    - RAID: *Redundant Arrays of Independent Disks*
    - à plus grande échelle : BD distribuée sur le réseau (*Cloud*)
- Les données sont déplacées du disque en **mémoire principale** pour l'accès, puis réécrites sur disque pour le stockage
- Les données plus modifiées et rarement lues sont transférées en **mémoire tertiaire** (disques optiques, bandes magnétiques) pour l'archivage

Discogs : une petite BD



- Une petite base de données ~ 80Go (0,08To)
- Un catalogue de plus de 8 800 000 références, 5 000 000 d'artistes et 1 000 000 de labels
- Plus de 369 000 contributeurs

Discogs


Discogs
Search artists, albums and more...
Explore
Marketplace
Community
Log In
Register
Genre
Rock 408,615
Electronic 398,818
Pop 210,781
Folk, World, & Country 108,200
Funk / Soul 98,565
All
Style
House 67,548
Pop Rock 60,747
Punk 49,286
Techno 45,613
Experimental 40,931
All
Format
Vinyl 740,855
Album 395,449
CD 348,999
LP 302,468
7" 225,873
All
All
Release 8,889,969
Master
Artist 5,077,218
Label 1,066,193
Search Marketplace
1 – 50 of 1,231,091
Prev
Next
Sort
Title, Z-A
Zzzzzz...
Skull Kontrol
Zzzzz.....
Tor.Men.To
ZZZZ
Sun Bather
...Zzzipp!
Tab Two
Zzzero
Various
ZZZ EP
Dubble D presents Mo...
Zzz
sanmi
Zzz
J. Mono
Zzyzx Rd.
Stone Sour
Zzyzx
Zeromancer

Blocs et enregistrements

Une BD contient des tableaux dont on appelle chaque ligne «enregistrement», par exemple :

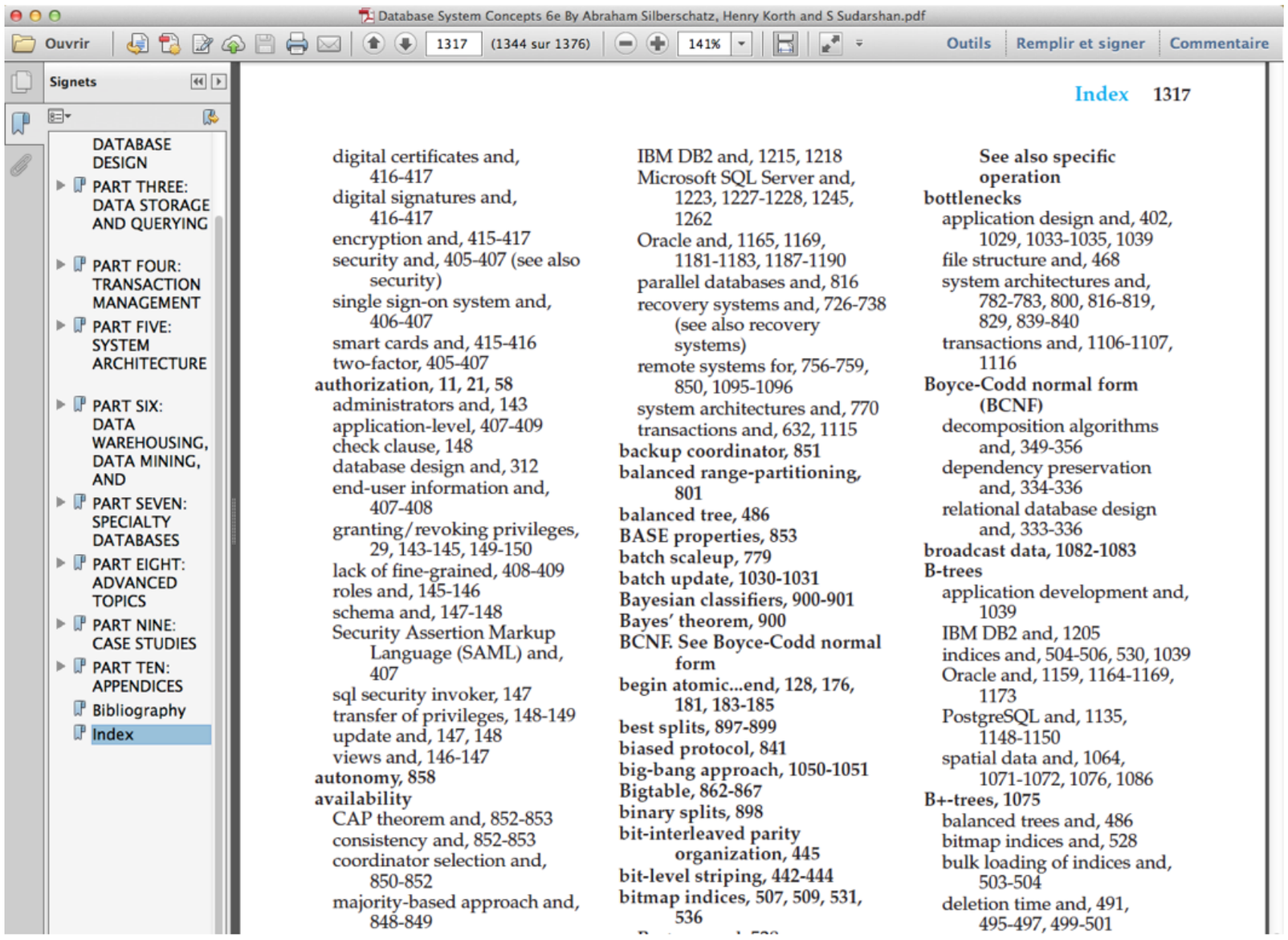
ALBUM	Num_Album	Nom_Artiste	Nom_Album	Genre	Année
	42	Wu-Tang Clan	Enter The Wu-Tang (36 Chambers)	Hip Hop	1993
	69	Mykki Blanco	Cosmic Angel : The Illuminati Prince/ss		2012
	555	Five Finger Posse	Hood Goth	Hip Hop	2016
	3	Mason vs. Princess Superstar	Perfect (Exceeder)	Electro	2006

- Ces enregistrements sont rangés dans des «pages» stockées dans des «blocs» du disque
- Chaque enregistrement a une adresse physique en mémoire
- Le bloc est l'unité de transfert avec la mémoire principale

Indexes

- Afin de déplacer un enregistrement du disque à la mémoire principale, il nous faut son adresse physique.

- Les SGBD rangent ces adresses physiques grâce à des «indexes»



Efficacité de la recherche

- Objectif : être capable de trouver efficacement les enregistrements d'un tableau avec une valeur particulière pour un certain champ
- Modèle de coût pour évaluer l'efficacité :
  - Mesure du coût : nombre de lectures/écritures de blocs du disque
  - Justification :
    - coût d'un accès au disque (lecture/ écriture d'un bloc) : millisecondes
    - coût d'un accès en mémoire centrale : nanosecondes (milliardièmes de seconde)

    ⇒ toutes les opérations de traitement d'un bloc en mémoire centrale ont un coût négligeable

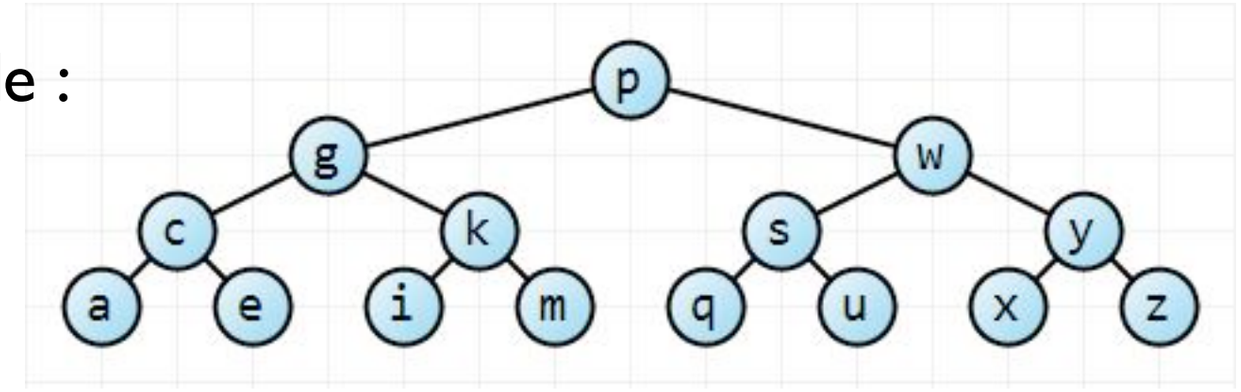
Recherche séquentielle

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»
- Approche naïve : parcourir tous les blocs de Album triés par ordre alphabétique
  - Coût : environ $\lceil n\ r\ /B \rceil$
    - n : nombre d'enregistrements dans Album,
    - r : taille d'un enregistrement
    - B : taille d'un bloc
- Si n est très grand ça peut être prohibitif
- Exemple : n = 8 000 000, r = 100, B = 8000
  - Coût : environ 100 000 * 10 millisecondes = 1000 secondes = 16,66 mins !!!!
    - (8 000 000 * 100) / 8000 = 100 000 et 1seconde = 0,0001millisecondes
    - 10 millisecondes : temps typique d'accès au disque

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :

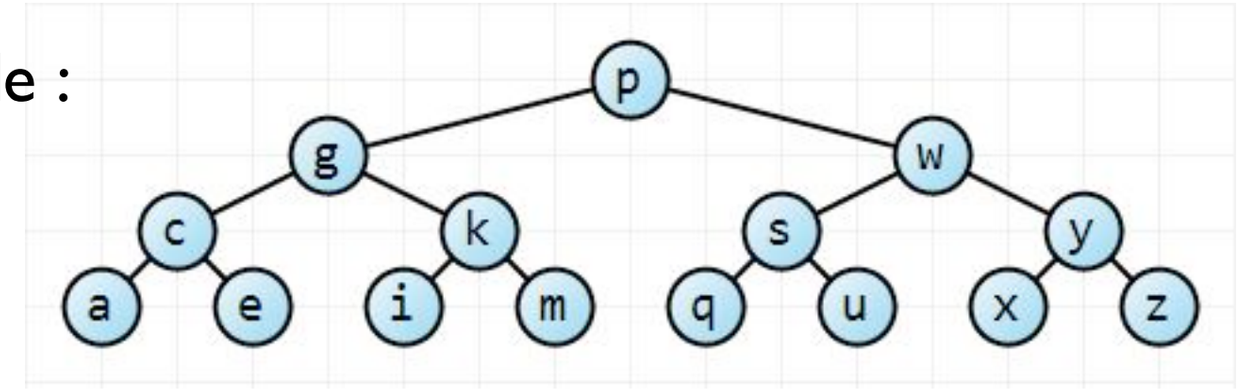


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :

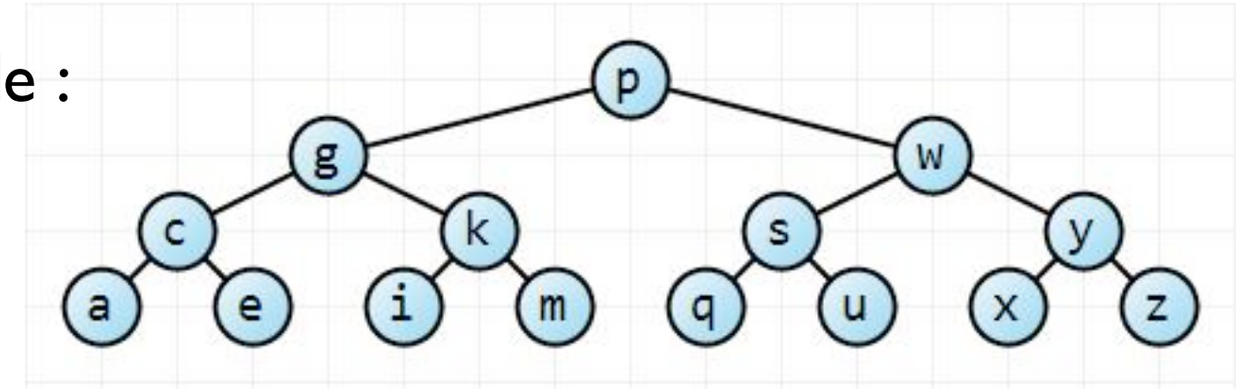


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :

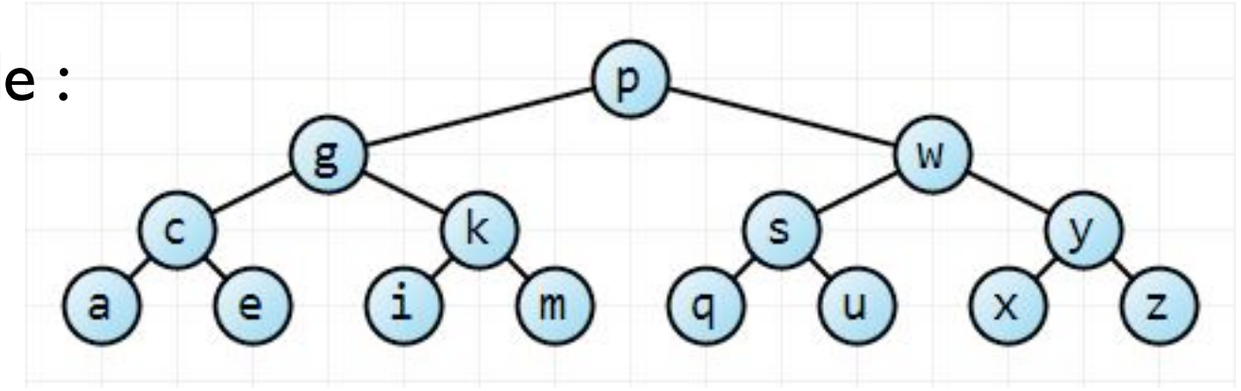


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p
- Est-ce que x est égal à p, avant p, ou après p dans l'ordre alphabétique ? après

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :

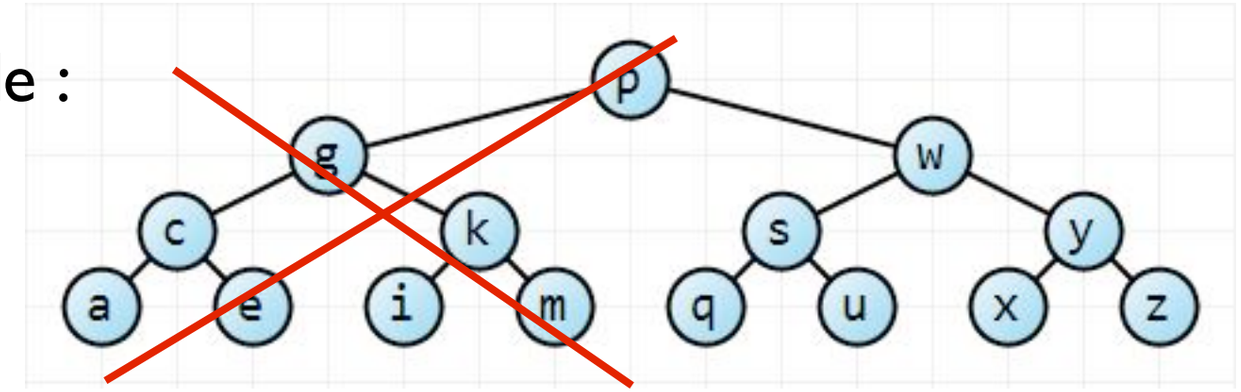


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p
- Est-ce que x est égal à p, avant p, ou après p dans l'ordre alphabétique ? après
- Toute la partie allant jusqu'à p ne nous intéresse donc plus

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :

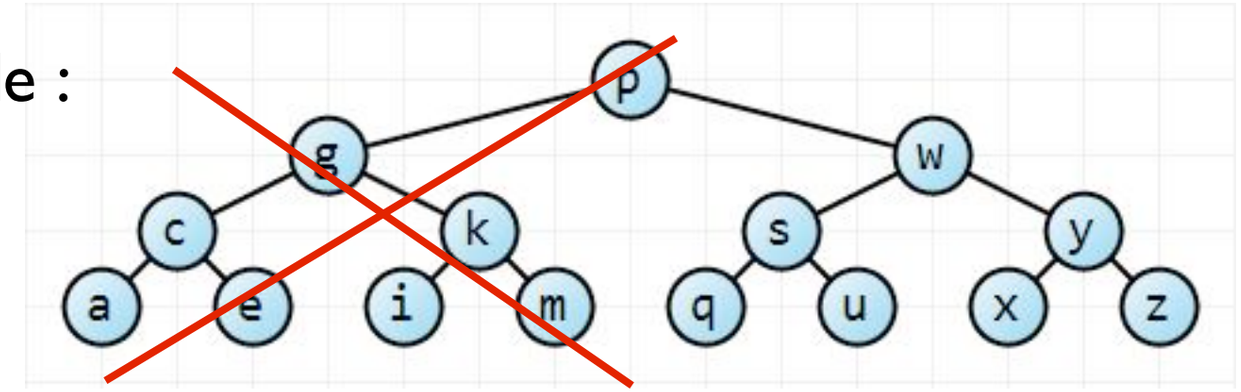


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p
- Est-ce que x est égal à p, avant p, ou après p dans l'ordre alphabétique ? après
- Toute la partie allant jusqu'à p ne nous intéresse donc plus

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :

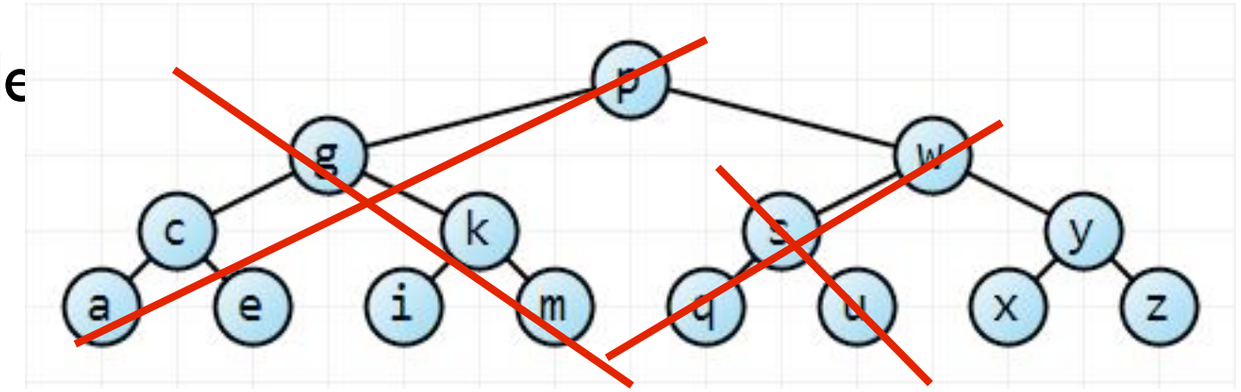


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p
- Est-ce que x est égal à p, avant p, ou après p dans l'ordre alphabétique ? après
- Toute la partie allant jusqu'à p ne nous intéresse donc plus
- On répète alors le même raisonnement pour le fragment de l'alphabet allant de q à z.

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

- Approche naïve 2 : diviser pour régner
- Un exemple plus simple

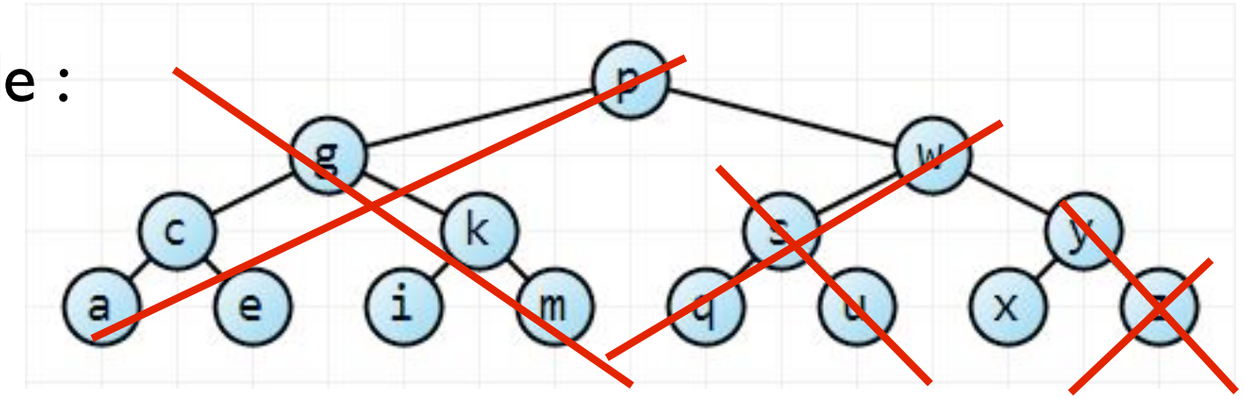


- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p
- Est-ce que x est égal à p, avant p, ou après p dans l'ordre alphabétique ? après
- Toute la partie allant jusqu'à p ne nous intéresse donc plus
- On répète alors le même raisonnement pour le fragment de l'alphabet allant de q à z. Etc.

Recherche dichotomique

- Objectif : trouver l'enregistrement correspondant à «Enter the Wu-Tang (36 Chambers)»

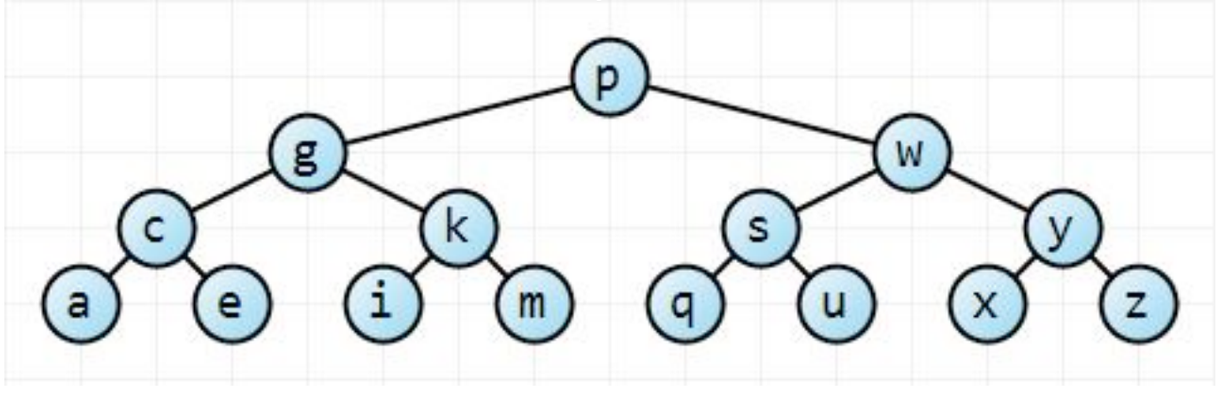
- Approche naïve 2 : diviser pour régner
- Un exemple plus simple :



- On cherche la lettre x parmi 15 lettres ordonnées de manière alphabétique.
- Quelle lettre se trouve au milieu dans l'ordre alphabétique ? p
- Est-ce que x est égal à p, avant p, ou après p dans l'ordre alphabétique ? après
- Toute la partie allant jusqu'à p ne nous intéresse donc plus
- On répète alors le même raisonnement pour le fragment de l'alphabet allant de q à z. Etc.

Recherche dichotomique

- Approche naïve 2 : diviser pour régner
- (Oublions provisoirement les blocs.)



  - Coût : pas plus de $\lceil \log_2 15 \rceil = 4$ étapes
    - 15 : nombre de lettres,
    - $\log_2$: le logarithme en base 2 de 15 est la puissance à laquelle il faut élever 2 pour arriver à 15
    - $\lceil \log 15 \rceil$ la partie entière par excès (ou «plafond») de $\log_2 15$
    - $\log_2 15 \sim 3{,}9$ ($2*2*2 = 8$ et $2*2*2*2=16$, or $8 < 15 < 16$)
- Beaucoup mieux que l'approche 1, qui aurait pris 14 étapes !

Recherche dichotomique

- Remarque: ce coût peut être élevé pour un grand nombre d'enregistrements
  - Exemple: 8 000 000 d'enregistrements
    - supposer par exemple 80 enregistrements par bloc
    - ⇒ 100 000 blocs
    - ⇒ un recherche demande $\lceil \log_2 100\,000 \rceil = 17$ accès aux blocs du disque (en effet 65 536 < 100 000 < 131 072)
    - temps typique d'accès à un bloc : 10 ms ⇒ 170 ms = 0,17 s par recherche
    - Beaucoup beaucoup beaucoup mieux que l'approche 1 !!! (16,66 mins)
    - Mais quand même.... seulement 5 recherches par seconde...

Indexation

- Approche 3 : Utiliser une structure de données auxiliaire
- On garde si possible cette structure en mémoire principale
- Permet d'accéder facilement à l'enregistrement cherché
    - Donne le bloc où se trouve l'enregistrement cherché
    - Donne l'adresse dans le bloc
- - Lecture uniquement du (ou des) bloc(s) utile(s)

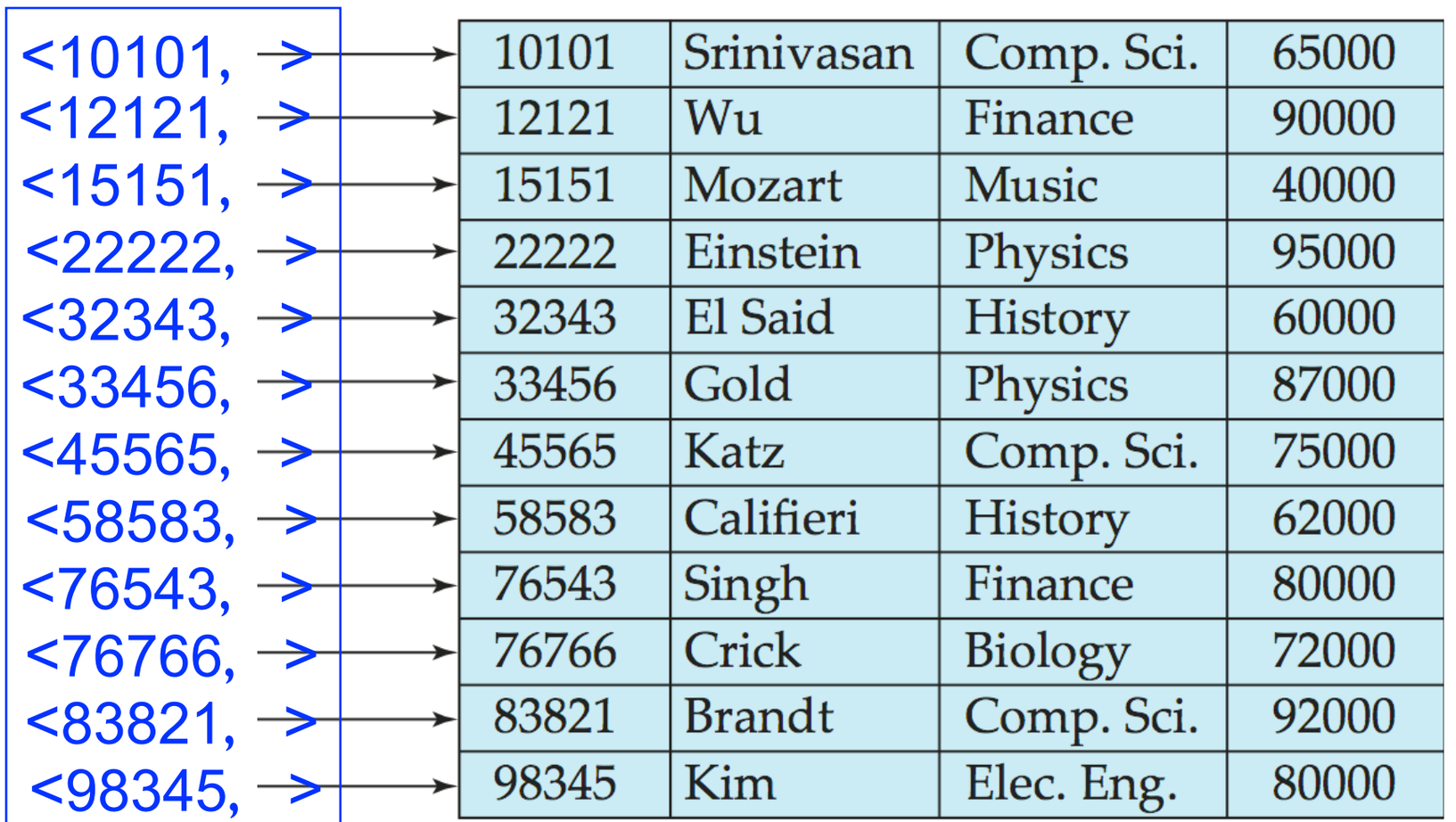
Index : une structure physique auxiliaire construite sur une clef de recherche qui permet d'accéder "presque directement" aux enregistrements ayant une valeur particulière de la clef de recherche

Index

- Un index sur une clef de recherche est une collection de couples

< c, • >

pointeur à un enregistrement ayant valeur *c* de la clef de recherche



Indexation : principe

- Rechercher des enregistrements ayant une valeur c de la clef de recherche :
  - parcourir l'index
  - si un couple < c, · > est trouvé suivre le pointeur
    - charger en mémoire principale le bloc contenant l'enregistrement

- Nombres d'entrées dans l'index : au plus le nombre n d'enregistrements dans le fichier de données
- Mais un couple <c, · > occupe beaucoup moins de place qu'un enregistrement!

Indexation : principe

- Coût de la recherche : dépend de l'implémentation de l'index mais au pire

environ $\lceil n\ i\ /B \rceil + 1$ (i : taille d'une entrée de l'index)

(n : nombre d'enregistrements)

B : nombre de blocs occupés par l'index

un accès au fichier de données

$\lceil n\ i\ /B \rceil << \lceil n\ r\ /B \rceil$ puisque i << r

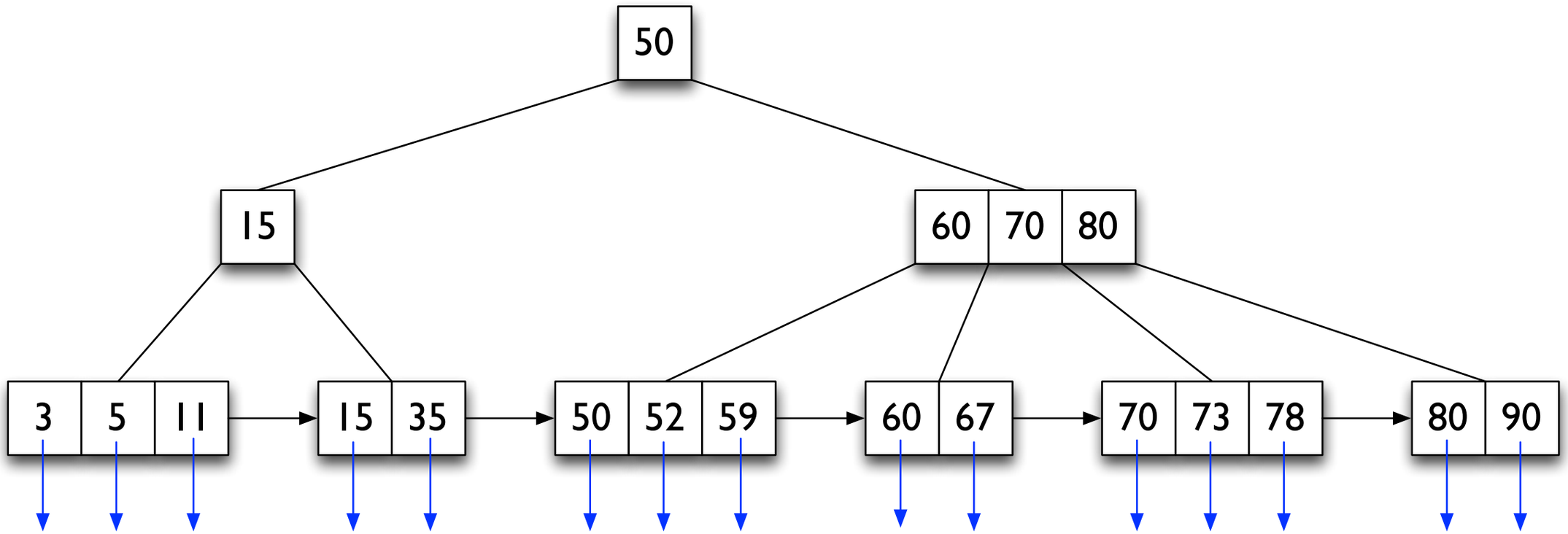
(r : taille d'un enregistrements)

- De plus il existe des implémentations efficaces des indexes :
  le coût de la recherche d'un couple $< c, \cdot >$ dans l'index peut passer de $\lceil n\ i\ /\ B \rceil$ à "presque" constant !

Implementation des indexes

- Différentes structures de données sont adaptées à l'implémentation d'indexes
- Les plus fréquentes :
  - arbres B+, arbres B
  - Hash
  - Bitmap
- Nous allons briévement présenter les arbres B+.
- Vous verrez toutes ces structures en détail dans le cours de Bases de Données avancées de M1 :-)

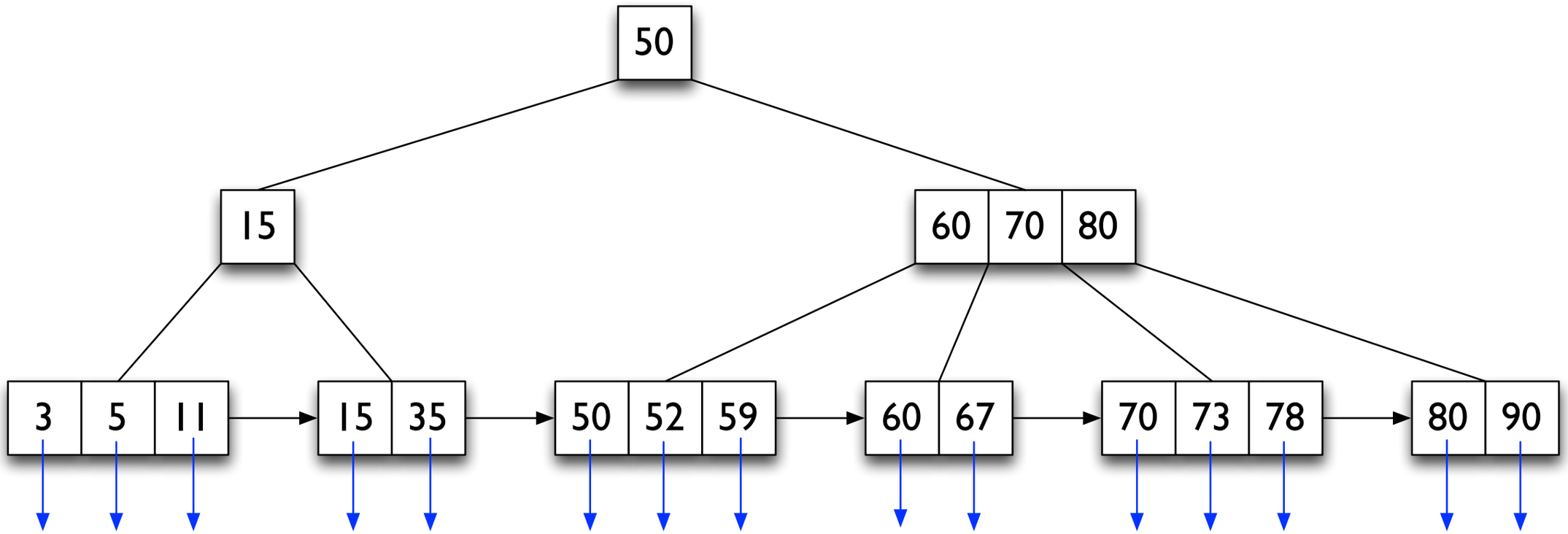
Arbres B+ : introduction



pointeurs aux enregistrements

- Structure arborescente (index multi-niveau)
  - un noeud de l'arbre = un bloc du disque
  - les noeuds feuille contiennent les couples <*clef, pointeur*> de l'index
  - les noeuds internes contiennent des doublons des clefs (appelés *balises*) pour orienter la recherche
  - tous les noeuds feuille sont liés en liste chainée, de gauche à droite (raison : permettre un parcours linéaire de la table par clef de recherche)

Arbres B+ : introduction



- Si un noeud (interne ou feuille) contient k clefs, il contient k+1 pointeurs
  - dans un noeud interne : pointeurs aux sous-arbres
  - dans un noeud feuille : pointeurs aux enregistrements et à la prochaine feuille

Soit n l'entier maximal tel que un bloc du disque contienne n clefs et n+1 pointeurs

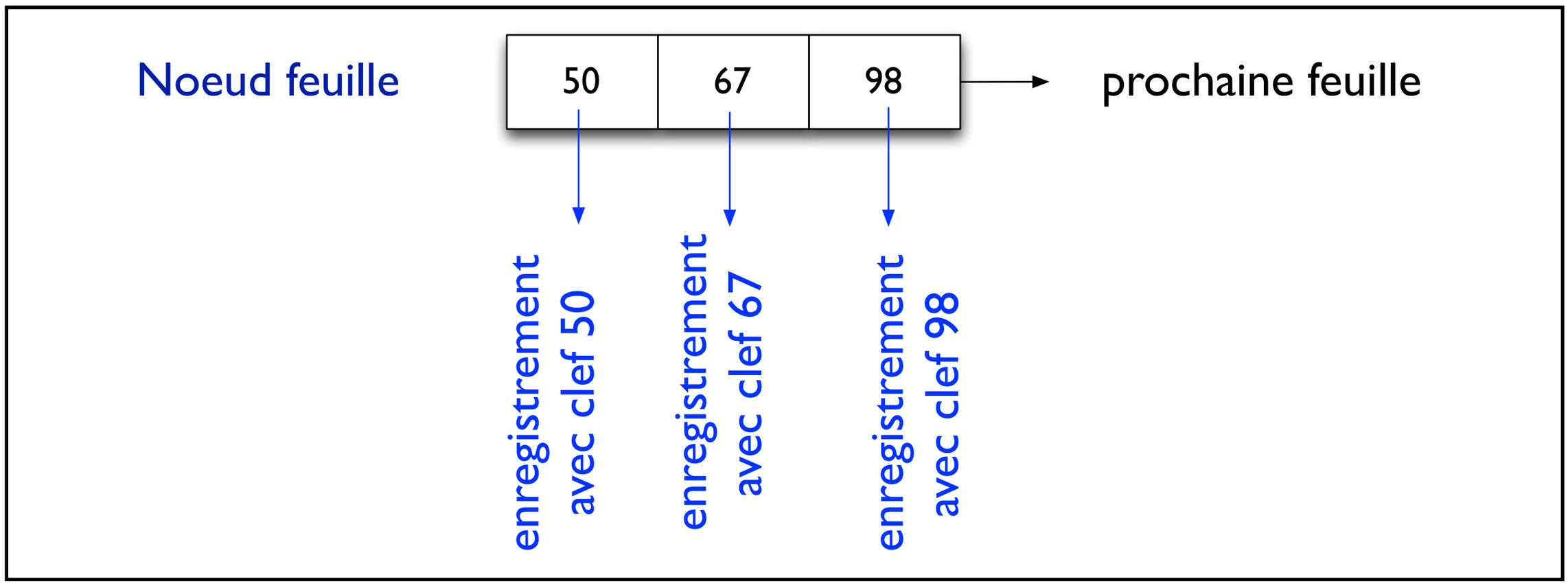
⇒ n est le nombre maximal de clefs dans un noeud de l'arbre

Arbres B+ : définition

Un arbre B+ possède les propriétés suivantes :

1) les feuilles

- ont toutes la même profondeur,
- contiennent des couples <*clef, pointeur*>, triés par clef, et un pointeur à la prochaine feuille



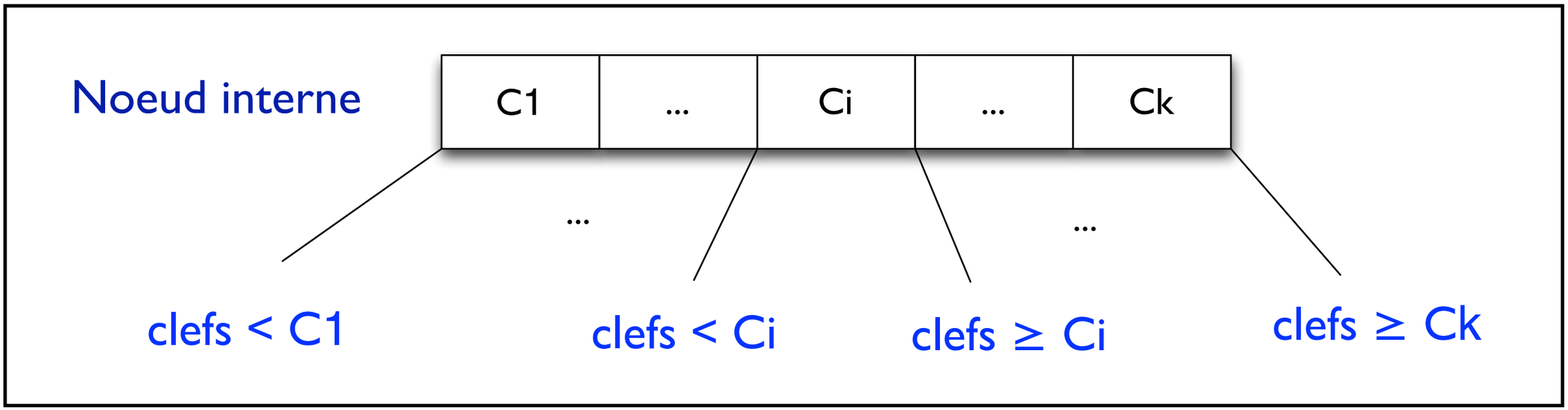
- contiennent entre ⌈ n/2 ⌉ et n clefs (n défini comme au slide précédent)

Arbres B+ : définition

Un arbre B+ possède les propriétés suivantes :

2) les noeuds internes contiennent

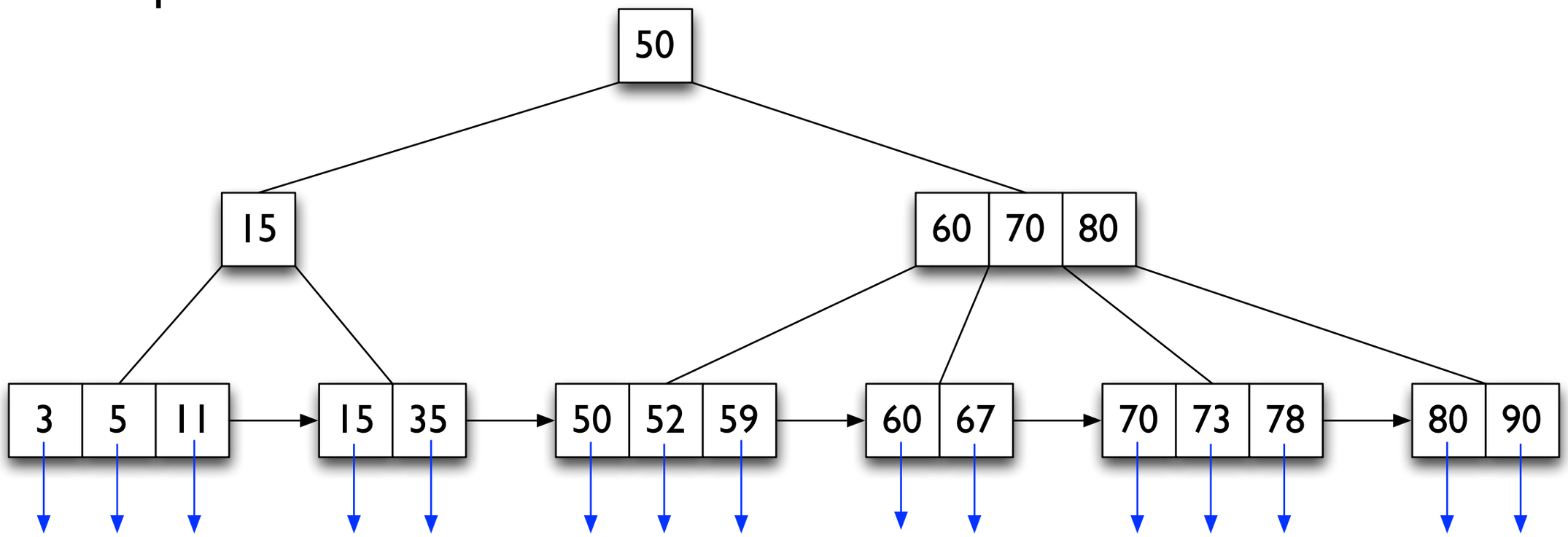
- une suite de clefs triées par ordre croissant
- un nombre de pointeurs à sous-arbres = nombre de clefs + 1, avec la propriété suivante pour chaque clef Ci:



- tous noeuds internes sauf la racine : entre $\lfloor n/2 \rfloor$ et n clefs
- la racine : entre 1 et n clefs

Arbres B+ : exemple

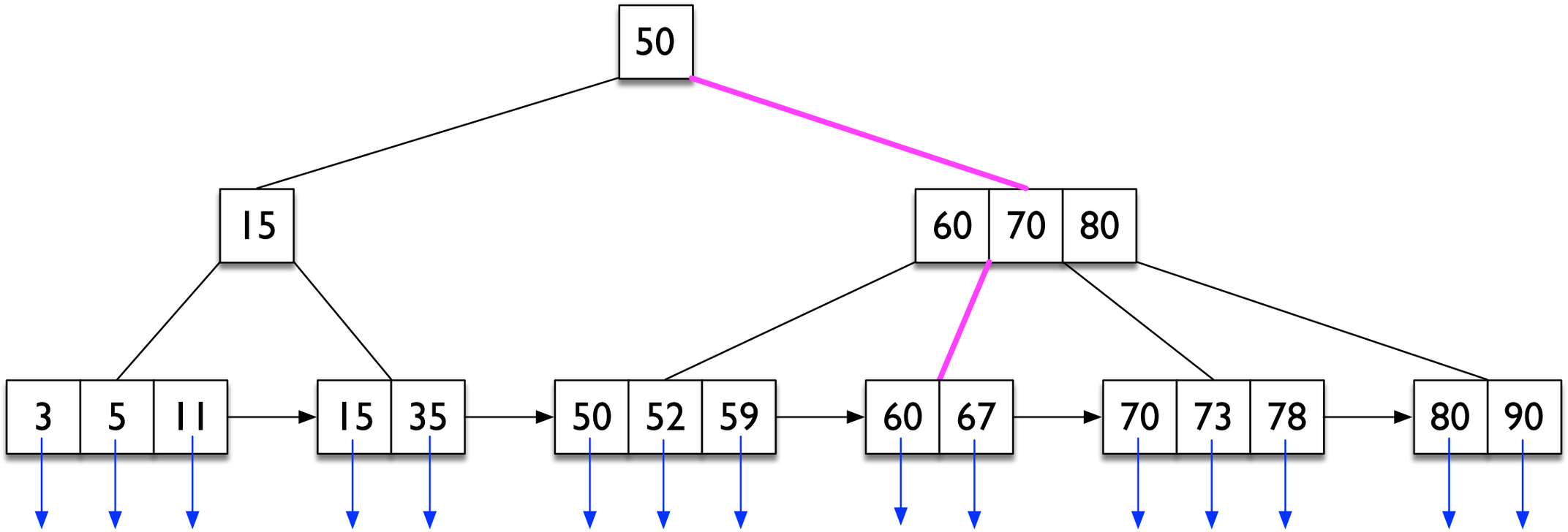
- Exemple avec n= 3



- noeuds feuilles : entre 2 et 3 clefs
- noeuds internes sauf la racine : entre 1 et 3 clefs
- la racine : entre 1 et 3 clefs

Arbres B+ : recherche

- Exemple c = 67



- Ici le nombre de noeud maximal depuis la racine jusqu'à une feuille est 3 : on dit que cet arbre est de hauteur 3.
- Ici le nombre de noeuds depuis la racine jusqu'à une feuille est toujours le même : on dit que l'arbre est équilibré.
- Le nombre de lectures de noeuds pour arriver à un enregistrement est égal à la longueur d'une branche, c'est à dire à la hauteur de l'arbre.

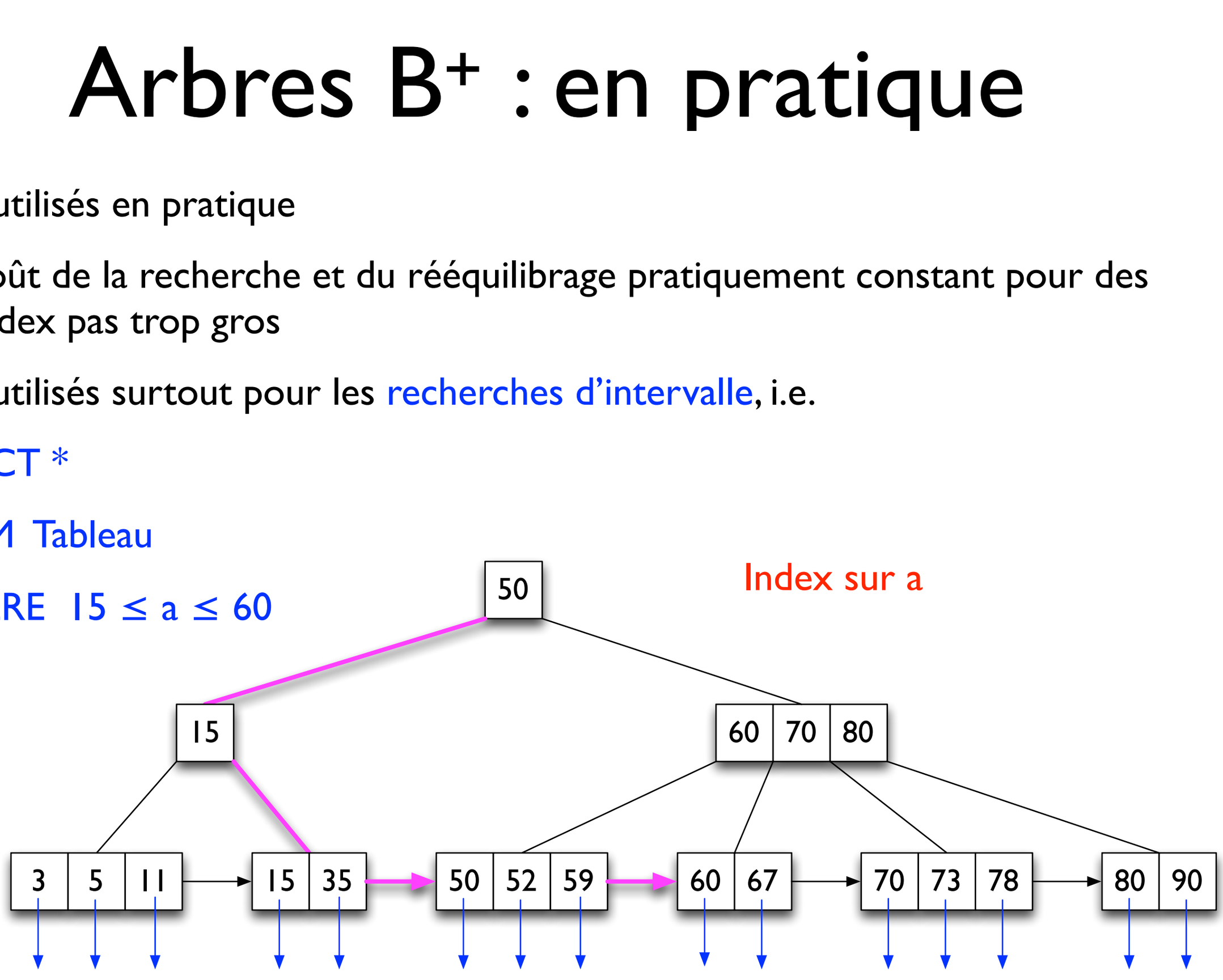
Arbres B^+ : en pratique

- Très utilisés en pratique
  - coût de la recherche et du rééquilibrage pratiquement constant pour des index pas trop gros
- Très utilisés surtout pour les recherches d'intervalle, i.e.

SELECT *

FROM Tableau

WHERE 15 ≤ a ≤ 60



Arbres B+ : bornes

- But des bornes sur le nombre de clefs:
  - borne supérieure : pour pouvoir stocker un noeud dans un bloc
  - borne inférieure :
    - pour la procédure de rééquilibrage de l'arbre après insertion / suppression
    - pour éviter des blocs trop vides (le blocs sont au moins à moitié pleins)

Arbres B+ : coût de la recherche

- Exemple : 8 000 000 enregistrements, 80 enregistrements par noeud, 41 = nombre minimal d'enregistrement par noeud interne
- Hauteur de l'arbre : $\lceil \log_{41}(8\,000\,000/2 + 1) \rceil = 5$
- 41*41 = 1681
- 1681*41 = 68921
- 68921*41 = 2 825 761
- 2 825 761 * 41 = 115 856 201
- La recherche dans l'abre B demande donc 5 accès disques seulement.
  - temps typique d'accès à un bloc : 10 ms ⇒ 50 ms = 0,05 s par recherche
- La recherche dichotomique demandait $\lceil \log_2 100\,000 \rceil = 17$ accès disques.
  - temps typique d'accès à un bloc : 10 ms ⇒ 170 ms = 0,17 s par recherche
- Gain de performance très important ! (5 versus 1200 requêtes par seconde)

Différence fondamentale :
on est passé de $\log_2$ à $\log_b$ avec b beaucoup plus grand que 2

Conclusions