Rappels

Plusieurs façons d'exprimer la même requête dans SQL.

Si q est dans SQL, chaque q' de ALG équivalent à q peut-être vu comme un plan d'évaluation de q.

Plusieurs façons d'exprimer la même requête dans ALG. Donc plusieurs plans d'évaluation possibles.

$\Rightarrow$ comment trouver le meilleurs?

Optimisation

Que veut on optimiser?

- Le temps global d'évaluation
- Le temps pour obtenir la première réponse
- Le débit réseau
- etc...

On s'intéresse ici au premier point uniquement.

Rappels

Comparer tous les algorithmes possibles est insurmontable. Même pour des requêtes conjonctives on passerait plus de temps à optimiser qu'à évaluer.

$\Rightarrow$ On utilise des heuristiques

1. "Pousser les sélections"
2. Ordre sur les jointures

Deux axes principaux

Avoir des implantations des opérateurs de l'algèbre relationnelle les plus performantes possibles.

L'efficacité dépend du contexte: on aura donc plusieurs implantations de chaque opérateur et le système choisira la plus appropriée selon le contexte.

Choisir un plan global d'évaluation le plus performants possible.

Cela veut dire par exemple choisir l'ordre dans lequel on va faire les jointures et décider d'une implantation pour chaque opérateur physique.

Aujourd'hui

Implantation des opérateurs de l'algèbre relationnelle.

Implantation des opérateurs physiques

On considère ici les opérateurs σ et $\bowtie$

C'est à dire on cherche à faire des parcours de fichiers avec un critère de sélection.

Quelques notions sur les mémoires:

Mémoire cache: très rapide (10 nanosec.), très chère, < 1M

Mémoire principale: rapide (10 à 100 nanosec.), chère, < 1G

Disque Dur: lent (10 millisec.), bon marché, centaine de G
données regroupées en blocs de qq K.

Bandes: très lent (1 sec.), peu chère, plusieurs Téras.

Plus une mémoire est rapide plus elle est chère et (donc) moins il y en a.

Un accès en mémoire vive est environ 1 million de fois plus rapide qu'un accès sur disque dur!

Importance pour les bases de données:

Un SGBD doit ranger sur disque les données (parce qu'elles sont trop volumineuses et qu'elles doivent persister à long terme.) il doit les amener en mémoire vive pour les traiter.

Si possible, les données utiles devraient résider le plus possible en mémoire vive.

La performance d'un SGBD dépend de sa capacité à gérer efficacement les transferts disque/mémoire.

Les solutions pour améliorer les performances

1. Techniques de cache du disque dur en mémoire vive.
2. Regroupement des blocs sur le disque dur pour limiter les mouvements de la tête de lecture.
3. Limitation des Entrées/Sorties lors des sélections/jointures (ordre sur les enregistrements, Arbre B, hachage etc...)

1. Techniques de caches: classiques, non présentées dans ce cours.

2. Regroupement des blocs: regroupement physiquement sur le disque dur des enregistrements souvent lus en même temps $\Rightarrow$ limite les mouvements de la tête de lecture $\Rightarrow$ améliore sensiblement les performances. Se fait lors de la création de la base sur la base d'heuristiques ou plus tard sur la base de statistiques.

3. Les Index

Indispensable pour les opérateurs σ et $\bowtie$.

Sans index, seule solution : parcours séquentiel du fichier (complexité linéaire)

si le fichier est trié on peut faire un parcours par dichotomie (complexité logarithmique)

Avec index: parcours de l'index, puis accès direct à l'enregistrement.

Mais attention : mises à jour plus coûteuses !!

Clé de recherche = liste d'un ou plusieurs attributs sur lesquels portent les critères de sélection.

Types de recherche :

- Recherche par clé : on associe une valeur à chaque attribut de la clé

  Rechercher les acteurs du film "Vertigo"

- Recherche par intervalle : on associe un intervalle de valeurs à chaque attribut de la clé

  Rechercher les films parus entre 1960 et 1975

- Recherche par préfixe : on associe une valeur à un préfixe de la clé

  Rechercher les films dont le titre commence par V

Le fichier est trié

L'index est un nouveau fichier dont les enregistrements sont de la forme [valeur, Addr]. valeur est une valeur de la clé et Addr est la position du premier bloc du disque dur contenant un enregistrement avec cette valeur.

Le fichier d'index est trié sur valeur.

Un seul enregistrement par bloc du fichier de données est représenté dans le fichier d'index.

Exemple:

titre	année	...	titre	année	...
Vertigo	1958	...	Annie Hall	1977	...
Brazil	1984	...	Jurasic Park	1992	...
Twin Peaks	1990	...	Metropolis	1926	...
Underground	1995	...	Manhattan	1979	...
Easy Rider	1969	...	Reservoir Dogs	1992	...
Psychose	1960	...	Impitoyable	1992	...
Greystoke	1984	...	Casablanca	1942	...
Shining	1980	...	Smoke	1995	...

Exemple:

avec :

- 1 000 000 (1 million) de films
- 120 octets par enregistrement
- 34 enregistrements par bloc de 4K.
- environ 30 000 blocs, 120 Mo

On trie le fichier:

titre	année	...	titre	année	...
Annie Hall	1977	...	Metropolis	1926	...
Brazil	1984	...	Psychose	1960	...
Casablanca	1942	...	Reservoir Dogs	1992	...
Easy Rider	1969	...	Shining	1980	...
Greystoke	1984	...	Smoke	1995	...
Jurasic Park	1992	...	Twin Peaks	1990	...
Impitoyable	1992	...	Underground	1995	...
Manhattan	1979	...	Vertigo	1958	...

On construit l'index sur la clé "titre"

Annie Hall	Greystoke	Metropolis	Smoke

Annie Hall	1977	...
Brazil	1984	...
Casablanca	1942	...
Easy Rider	1969	...

Greystoke	1984	...
Jurassic Park	1992	...
Impitoyable	1992	...
Manhattan	1979	...

Metropolis	1926	...
Psychose	1960	...
Reservoir Dogs	1992	...
Shining	1980	...

Smoke	1995	...
Twin Peaks	1990	...
Underground	1995	...
Vertigo	1958	...

Sur notre fichier de 120 Mo

En supposant qu'un titre occupe 20 octets, une adresse 8 octets

Taille de l'index : $30000*(20+8) \simeq 800K$

Beaucoup plus petit que le fichier!

Peut éventuellement tenir en mémoire vive.

Index sur fichier trié

Permet une recherche:

Par clé : dichotomie sur l'index

Par intervalle : recherche de la borne inférieure, accès au fichier, puis parcours séquentiel

Par préfixe : cas particulier de la recherche par intervalle

Les plus: petite taille et simplicité

Les moins: mises à jour difficiles (il faut retrier le fichier ET l'index)

ne peut porter que sur une seule clé.

Index sur fichiers non triés

L'index est encore un nouveau fichier

Il est aussi trié sur la clé.

Tous les enregistrements sont représentés!

On construit un nouvel index sur la clé “année”

1926	1942	1958	1960	1969	1977	1979	1980	...

Annie Hall	1977	...
Brazil	1984	...
Casablanca	1942	...
Easy Rider	1969	...

Smoke	1995	...
Twin Peaks	1990	...
Underground	1995	...
Vertigo	1958	...

Greystoke	1984	...
Jurassic Park	1992	...
Impitoyable	1992	...
Manhattan	1979	...

Metropolis	1926	...
Psychose	1960	...
Reservoir Dogs	1992	...
Shining	1980	...

Sur notre fichier de 120 Mo

En supposant qu'une année occupe 4 octets, une adresse 8 octets

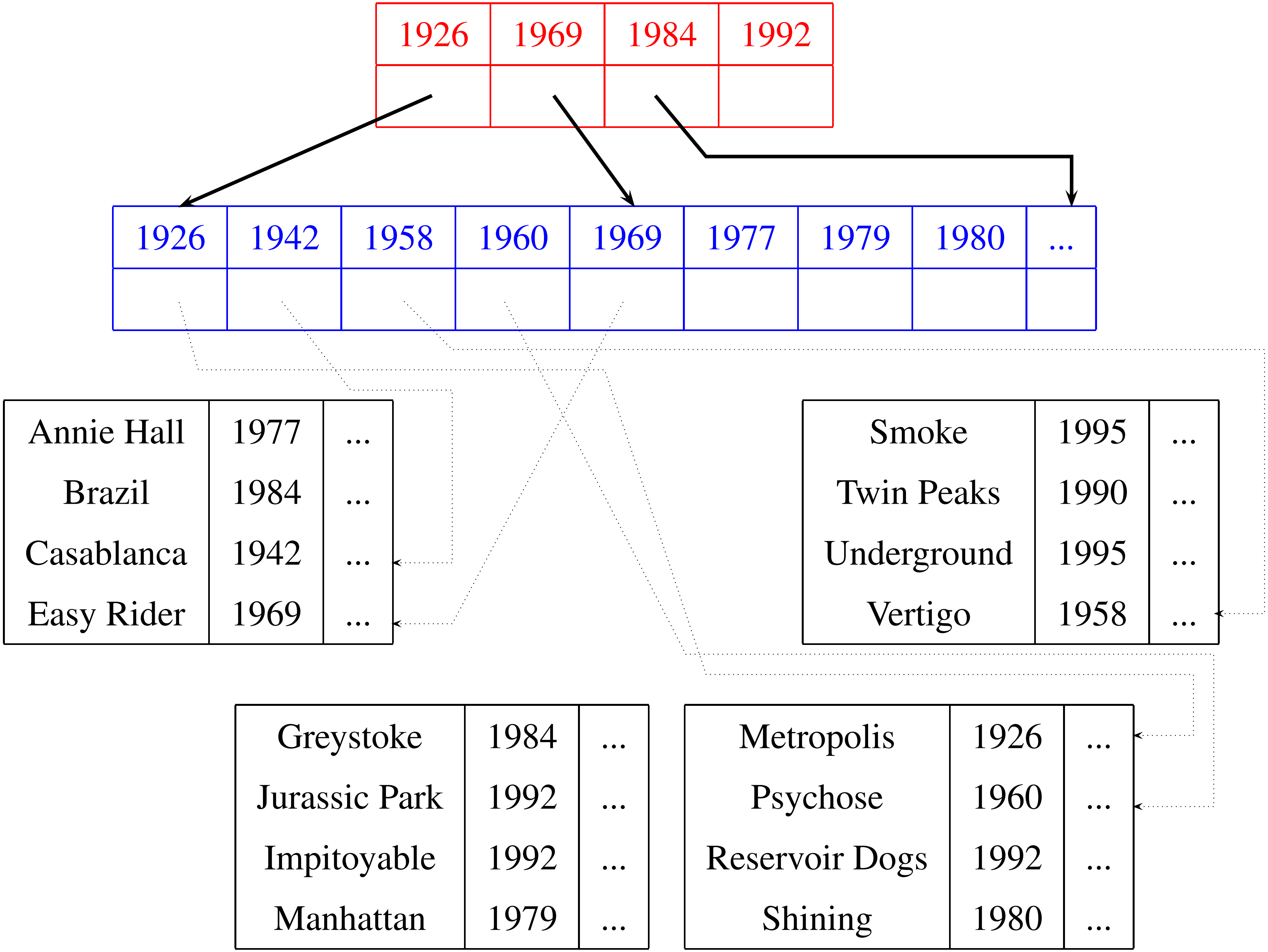
Taille de l'index : $1000000 * (4 + 8) \simeq 12M$

Seulement dix fois plus petit que le fichier : la taille de l'index ne peut plus être négligée!

Si le fichier est trop gros:

L'index est un fichier trié que l'on peut indexer à son tour par la méthode précédente!

On obtient une structure séquentielle indexée.


1926 1969 1984 1992
1926 1942 1958 1960 1969 1977 1979 1980 ...
Annie Hall 1977 ...
Brazil 1984 ...
Casablanca 1942 ...
Easy Rider 1969 ...
Smoke 1995 ...
Twin Peaks 1990 ...
Underground 1995 ...
Vertigo 1958 ...
Greystoke 1984 ...
Jurassic Park 1992 ...
Impitoyable 1992 ...
Manhattan 1979 ...
Metropolis 1926 ...
Psychose 1960 ...
Reservoir Dogs 1992 ...
Shining 1980 ...

Index sur un fichier non trié

Permet une recherche:

Par clé : dichotomie sur l'index comme avant

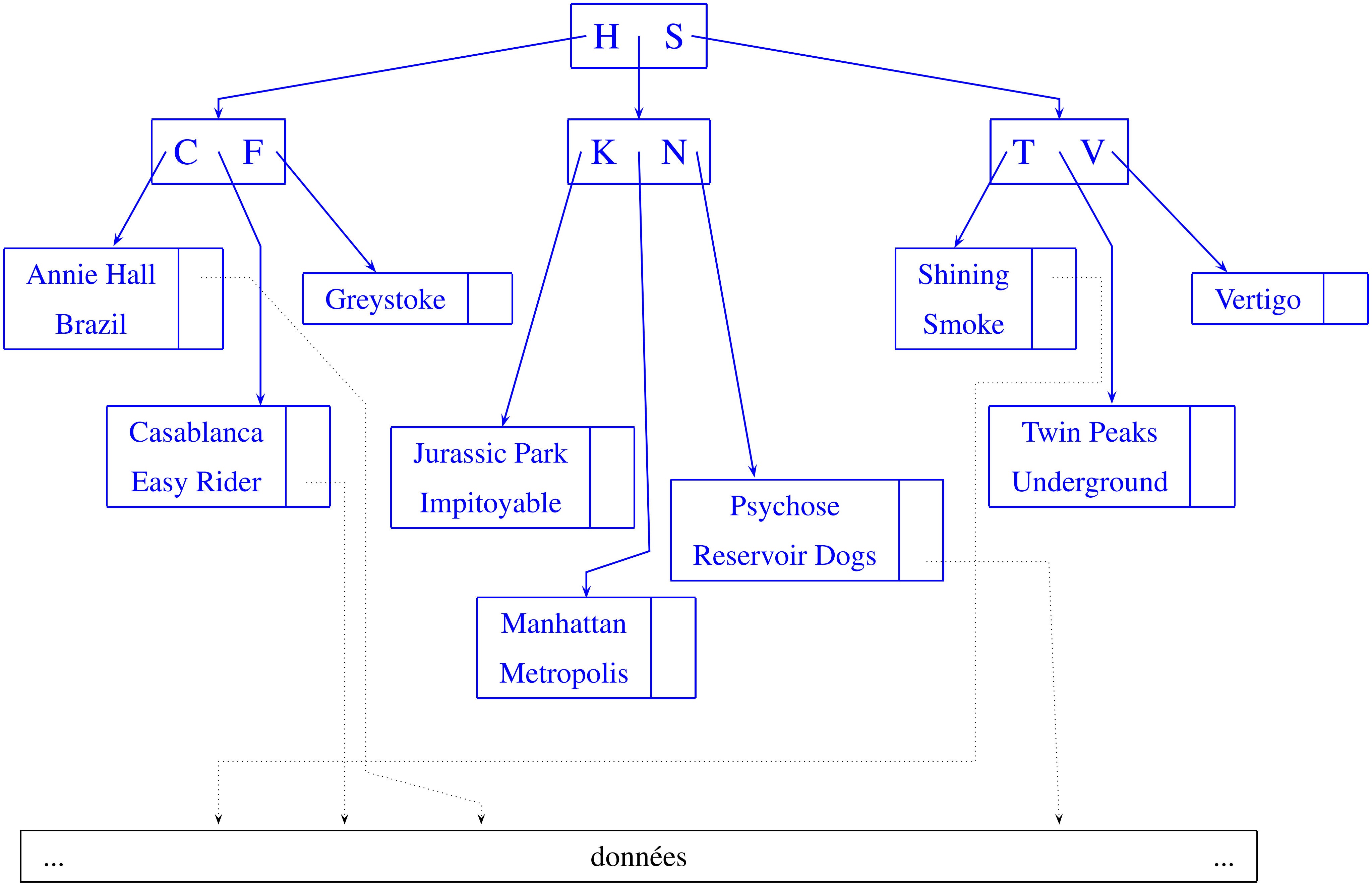
Par intervalle : recherche dans l'index de la borne inférieure parcours séquentiel de l'index et pour chaque valeur accès au fichier de données $\Rightarrow$ coût beaucoup plus élevé.

Autre possibilité : Arbre B+

Aboutissement des structures d'index basées sur l'ordre des données.

- C'est un arbre équilibré,
- chaque noeud est un **bloc** de [valeur,addr],
- il se réorganise dynamiquement.

Utilisé universellement !


H S
C F
K N
T V
Annie Hall
Brazil
Greystoke
Casablanca
Easy Rider
Jurassic Park
Impitoyable
Manhattan
Metropolis
Psychose
Reservoir Dogs
Shining
Smoke
Vertigo
Twin Peaks
Underground
... données ...

Sur notre fichier de 120 Mo

En supposant qu'un titre occupe 20 octets, une adresse 8 octets et un bloc 4K.

Nombre d'entrées par bloc: $4096/28 \simeq 146$

Nombre de feuilles dans l'index: $1.000.000/146 \simeq 6850$

Nombre de noeuds: $6850/146 \simeq 47$

La racine contient un bloc avec 47 entrées.

Taille totale de l'index: 6900 blocs $\simeq$ 28M

Taille non négligeable!

Recherche par clé: on fait un parcours de l'index puis un accès direct à l'enregistrement sur le disque.

Un accès disque par niveau dans l'index + 1

Recherche par intervalle: on fait un parcours de l'index pour trouver la plus petite valeur pour on parcours les feuilles de l'index jusqu'à la plus grande valeur. À chaque fois on va chercher l'enregistrement sur le disque.

Attention aux accès disque!

Permet une insertion dynamique: on rajoute l'enregistrement dans le bloc correspondant. Si celui-ci deviens trop gros, on le divise en deux en rajoutant une nouvelle entrée dans le noeud supérieur de l'arbre et ainsi de suite.

Linéaire dans la profondeur de l'index.

On a rarement besoin de beaucoup de niveaux:

Titre: avec 4 niveaux on peut indexer 146^4=450 millions de films.

Année: avec 3 niveau on peut indexer $(4096/12)^3$=39 millions de films

Plus la taille de la clé est petite plus l'index sera petit et donc efficace.

Bilan de l'Arbre B+

- On peut en avoir autant que l'on veut.
- On a très rarement besoin de plus de 4 niveaux.
- Le coût d'une recherche par clé est donc faible.
- Supporte des recherches par clé, par intervalle ou par préfixe.
- Il est dynamique.

Par contre sa taille est importante

Tables de hachage

Un concurrent de l'arbre- B+,

meilleur pour les recherches par clé

n'occupe aucune place

mais

se réorganise difficilement,

ne supporte pas les recherches par intervalle

Principe du hachage:

On **calcule** la position d'un enregistrement d'après la clé.

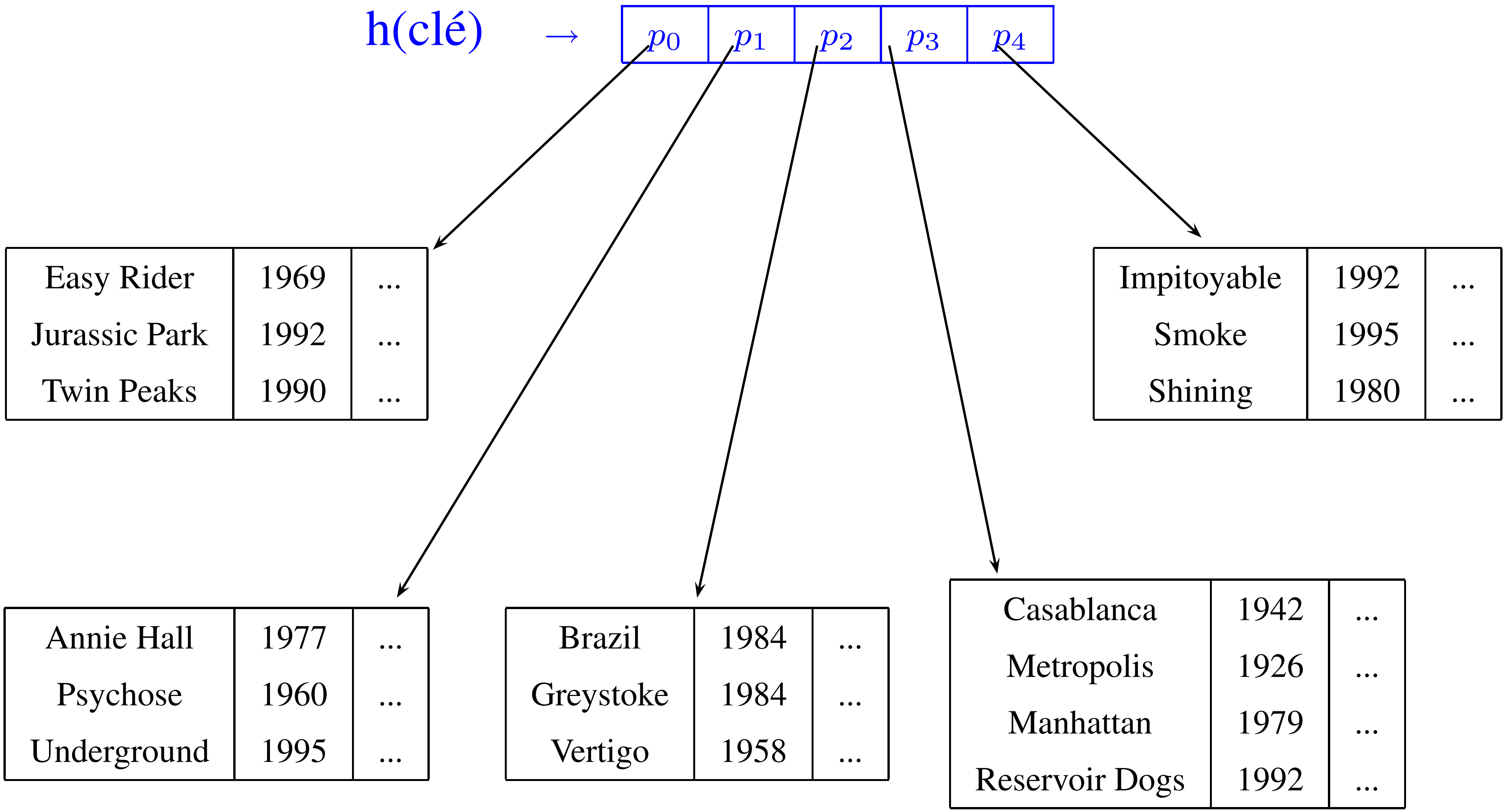
Une fonction de hachage h associe des valeurs de clé à des adresses de bloc.

h doit répartir **uniformément** les enregistrements dans les n blocs alloués à la structure

Recherche d'un enregistrement $\Rightarrow$ calcul de l'endroit où il se trouve.

Simple et efficace!

Table de hachage: h(titre)=rang(titre[0]) modulo 5



Bilan sur le hachage

La structure simple décrite précédemment n'est pas dynamique:

on ne peut pas changer un enregistrement de place.

⇒ chaînage des blocs ou hachage dynamique. Baisse des performances.

Permet une recherche par clé mais pas de recherche par intervalle.

Taille nulle.

Comparaison: Hachage vs. Arbre B+

La hachage, intéressant quand :

- Le jeux de données est **figé**
- Les recherches se font **par clé**

Ce sont des situations relativement courantes : le hachage n'occupe alors pas de place.

Sinon l'arbre B+ est meilleur.

Bilan sur les index

- **Tri**: pas dynamique, une seule clé possible.
- **Arbre B+**: dynamique, efficace, grosse taille, toutes sortes de recherches possibles.
- **Hachage**: simple, efficace, petite taille, peu dynamique, pas de recherche par intervalle.

Il en existe d'autres (bitmap etc...)!