# 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.

## 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.

# 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...)!

# On construit l’index sur la clé “titre”

| Annie Hall | Greystoke | Metropolis | Smoke |
|---|---|---|---|
| | | | |

| | | |
|---|---|---|
| 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 | ... |

# On construit un nouvel index sur la clé “année”

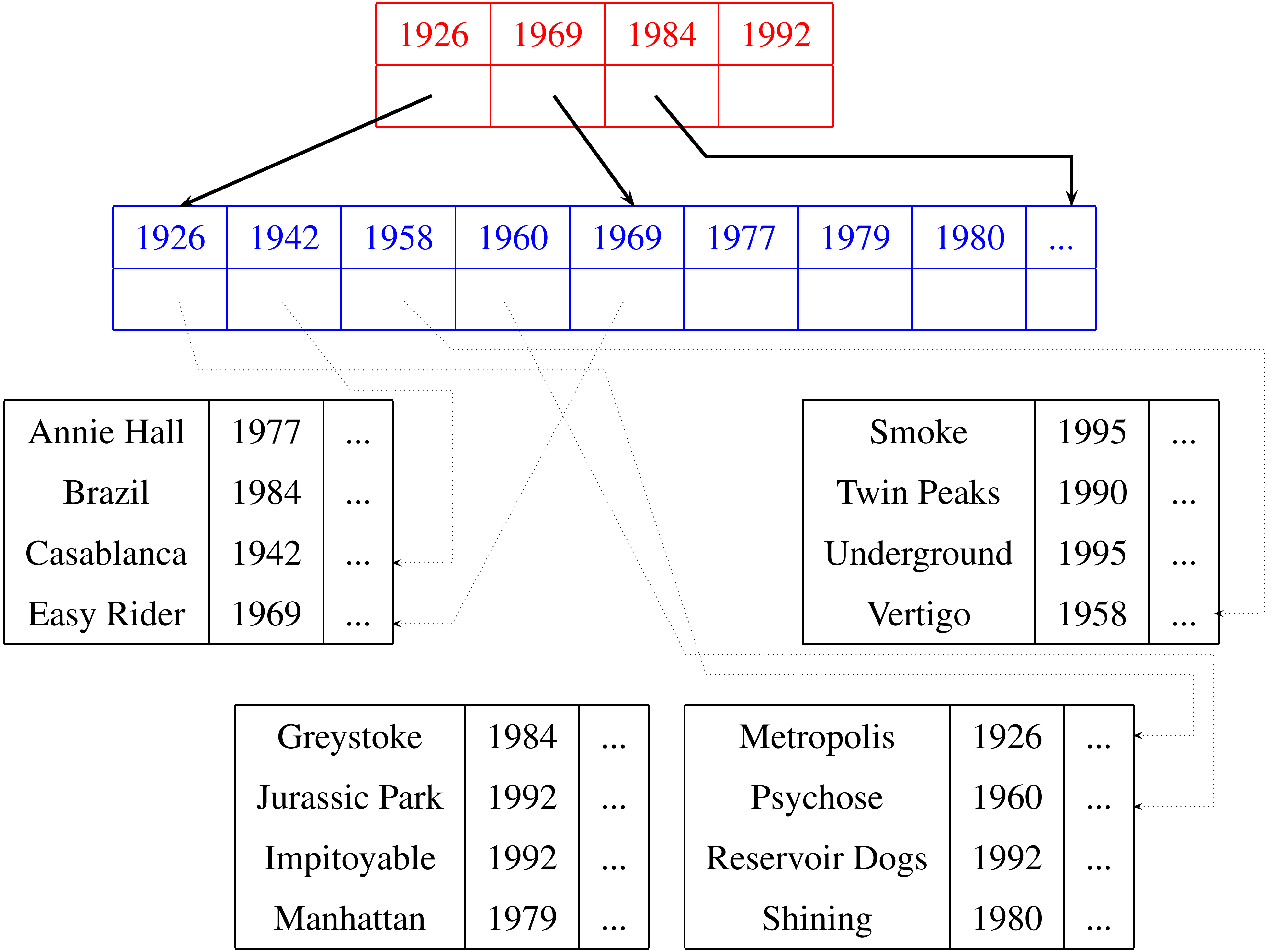
| 1926 | 1942 | 1958 | 1960 | 1969 | 1977 | 1979 | 1980 | ... |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

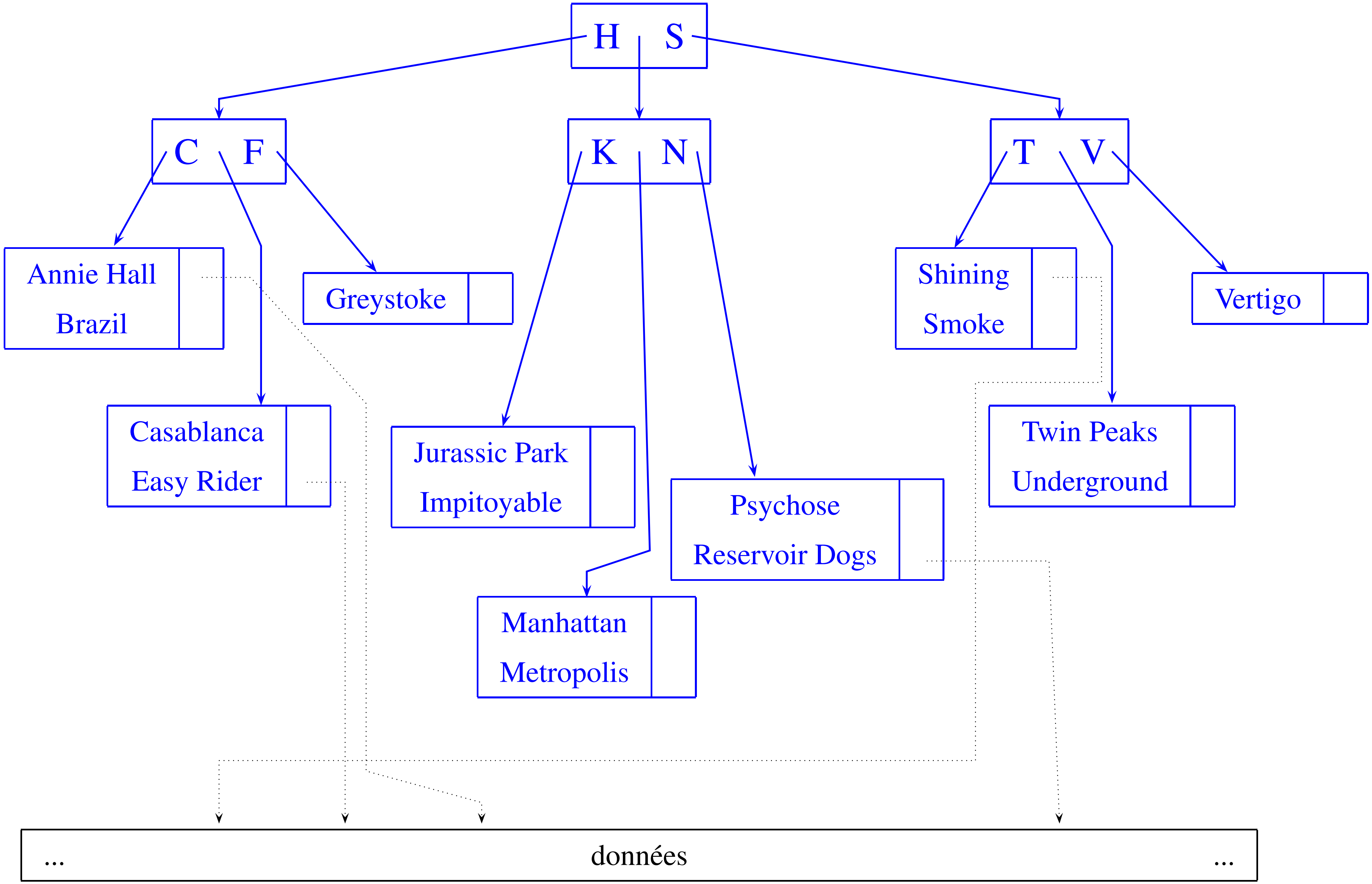
| | | |
|---|---|---|
| Annie Hall | 1977 | ... |
| Brazil | 1984 | ... |
| Casablanca | 1942 | ... |
| Easy Rider | 1969 | ... |

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

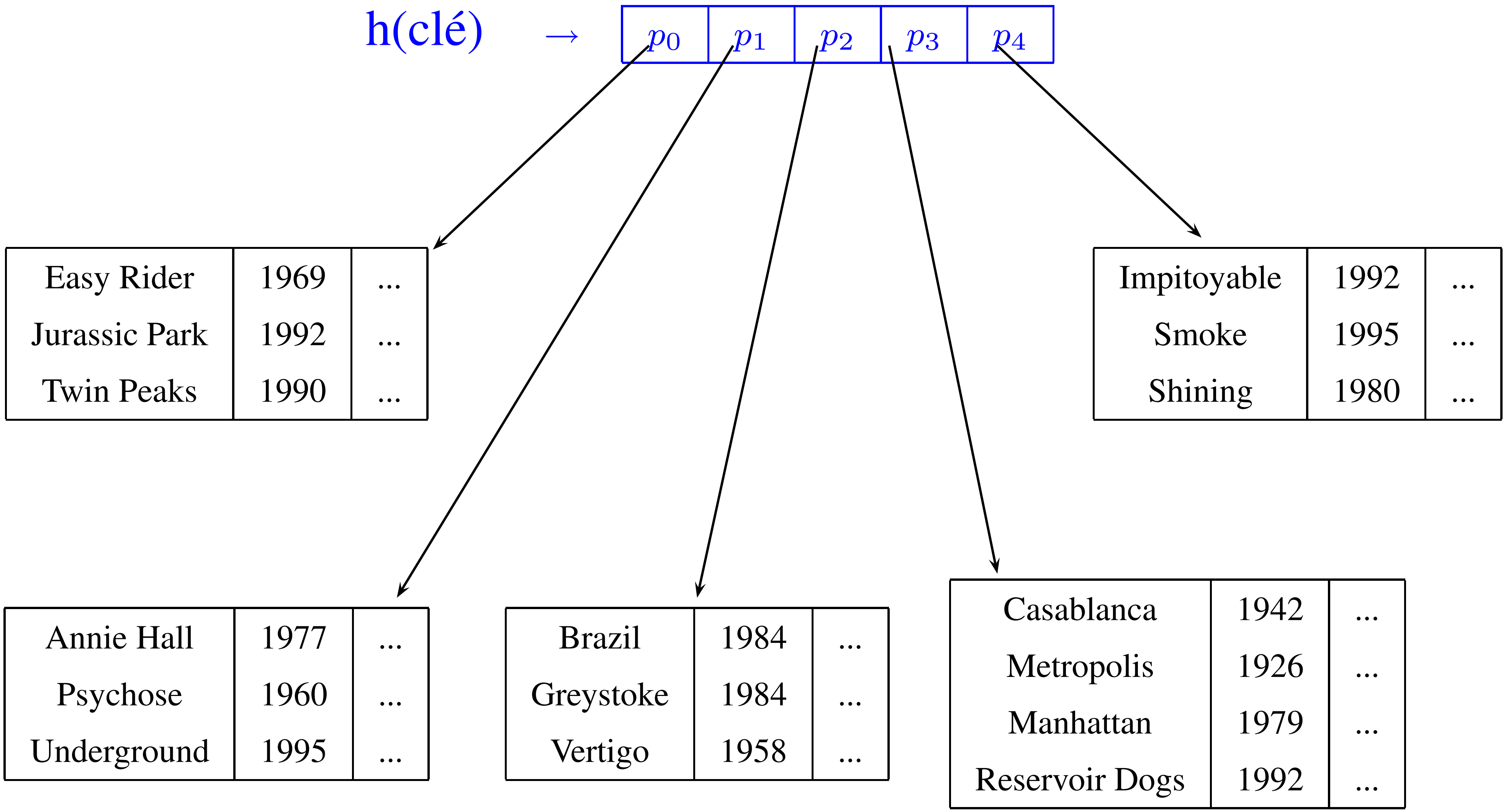
| | | |
|---|---|---|
| Greystoke | 1984 | ... |
| Jurassic Park | 1992 | ... |
| Impitoyable | 1992 | ... |
| Manhattan | 1979 | ... |

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


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 ...


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
...

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



## 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...)!

# Opérateur $\sigma_F$

Deux techniques pour accéder à un enregistrement :

1. **Accès séquentiel ou balayage**. Tous les enregistrements du fichier sont examinés.

2. **Accès par adresse**. On parcours l'index et on accède directement aux blocs.

Si la sélectivité $F$ est très faible, le coût de parcours de l'index deviens supérieur au coût du parcours du fichier $\Rightarrow$ on utilise un index que si $F$ est sélectif.

On utilise des statistiques pour évaluer la sélectivité de $F$.

# Opérateur $\bowtie$

- Jointure sans index
  - Le plus simple : jointure par boucles imbriquées
  - Le plus courant : jointure par tri-fusion
- Jointure avec index
  - Avec un index : jointure par boucles indexées.
  - Avec deux index : on fait comme si on avait un seul index

# Jointure par boucles imbriquées

C’est l’algo bête et méchant.

On essaye quand même de gérer la mémoire au mieux. Par exemple si une des tables est plus petite on essaye de la mettre en mémoire vive.

Permet d’obtenir des résultats **au fur et à mesure**.

## Jointure par tri-fusion

**Plus efficace** que les boucles imbriquées pour de grosses tables.

1. On trie les deux tables sur les attributs de la jointure.
2. On effectue la fusion.

Le tri qui coûte cher (nombreux accès disques pour lire et sauvegarder...).

Important : on ne peut rien obtenir tant que le tri n'est pas fini.

## Jointure par boucles indexées

1. On balaye la table non indexée.
2. Pour chaque ligne, on utilise l'attribut de jointure pour parcourir l'index sur l'autre table.

Avantages : Très efficace (un parcours, plus des recherches par adresse)

Favorise le temps de réponse (on obtient le résultat au fur et à mesure de son calcul) et le temps d'exécution.

## Jointures: quel algo choisir?

Si il y a un index essayer de l'utiliser surtout si l'attribut de jointure est sélectif.

Si il y a deux index, parcourir la petite table et accèder à la grande via l'index.

Si il n'y a pas d'index à voir selon tailles et statistiques.

D'autres algos existent...

Parfois le meilleur sans index: jointure par hachage (on hache la plus petite à la volée)

# Optimisation globale

$\Rightarrow$ On utilise des heuristiques

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

On utilise les règles:

$$E_1 \bowtie E_2 = E_2 \bowtie E_1,$$
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3).$$

$$\sigma_F(E_1 \ \bowtie \ E_2) = \sigma_F(E_1) \ \bowtie \ \sigma_F(E_2)$$

Pour modifier la requête.

On essaye de faire en priorité:

1. les sélections
2. les jointures réduisant le plus la taille des tables (utilisation de statistiques)

## exemple 1

Salle(nom,horaire,titre)     Film(titre,réalisateur,acteur)

REQUÊTE : *Les salles où l'on joue un film avec Stewart*

P1: $\pi_{nom}(\sigma_{acteur=\text{"Stewart"}}(\text{Film} \bowtie \text{Salle}))$

ou

P2: $\pi_{nom}(\sigma_{acteur=\text{"Stewart"}}(\text{Film}) \bowtie \text{Salle})$

1.000.000 films, 10 acteurs/film $\Rightarrow \sharp$ Film =10.000.000

100 salles, 5 films/salle $\Rightarrow \sharp$ Salle = 500

P1: [500 $\bowtie$10.000.000 $\longrightarrow$ 5.000.000.000] + 5.000.000.000

P2: 10.000.000 + [10 $\bowtie$500 $\longrightarrow$ 10] + 10

# exemple 2

Salle(nom,horaire,titre)    Film(titre,réalisateur,acteur)   Eval(acteur,note)

REQUÊTE : *Les salles où l'on joue un film avec un très bon acteur*

P1: $\pi_{\text{nom}}([\sigma_{\text{note="TB"}}(\text{Eval}) \bowtie \text{Film}] \bowtie \text{Salle})$

P2: $\pi_{\text{nom}}(\sigma_{\text{note="TB"}}(\text{Eval}) \bowtie [\text{Film} \bowtie \text{Salle}])$

1.000.000 films, 10 acteurs/film $\Rightarrow \sharp$ Film =10.000.000

100 salles, 5 films/salle $\Rightarrow \sharp$ Salle = 500

100.000 acteurs, 1.000 TB $\Rightarrow \sharp\ \sigma_{\text{note="TB"}}(\text{Eval}) = 1.000$

P1: [1.000 $\bowtie$10.000.000 $\longrightarrow$ 10.000] + [10.000 $\bowtie$500 $\longrightarrow$ 10.000] + 10.000

P2: [10.000.000 $\bowtie$500 $\longrightarrow$ 5.000.000.000] + [1.000 $\bowtie$5.000.000.000 $\longrightarrow$ 10.000] + 10.000

# Optimisation globale

On utilise des heuristiques.

On utilise des statistiques.

On fait les sélections en priorité.

On fait les jointures dont le résultat sera le plus petit en premier.

## Optimization: parcours classique

$q$ requête exprimée dans SQL.

• $q$ est traduite dans ALG $\longrightarrow q'$

• On réécrit $q'$ pour pousser les sélections et ordonner les jointures.

• On choisit pour chaque opérateur l'implantation adaptée en fonction des index existants et de statistiques/heuristiques.

$\longrightarrow$ plan d'évaluation.

• On évalue la requête en suivant le plan d'évaluation.