## Partitionnement des données

- Permet la scalabilité de l'application
- *Sharding* = partitionnement
- *Shard* = fragment

**Définition**

Une partition d'une collection $C = \{(i, o)\}$ est un ensemble de fragments $\{F_1, F_2, \ldots, F_n\}$ tel que :
- $\cup_{i=1}^{n} F_i = C$
- $F_i \cap F_j = \emptyset$ pour tout i, j avec $i \neq j$

Partitionnement s'effectue en fonction d'un critère
- **clé**
- ou un ou plusieurs attributs



Bon partitionnement :
- répartit les objets en fragments de taille comparable
- doit être **dynamique**
  - nombre de fragments doit pouvoir évoluer en fonction de la taille de la collection et du nombre de serveurs

**Structure de routage *(hash directory)***

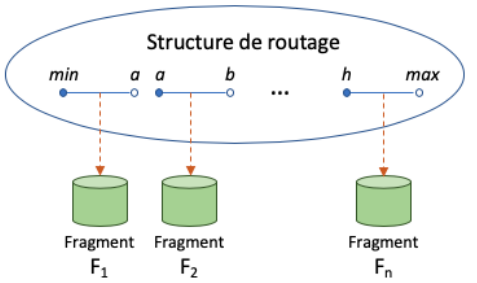
établit la correspondance entre le critère de partitionnement (clé) et le fragment qui lui est associé

2 grandes approches pour le partitionnement en fonction d'une clé :
- par intervalle
- par hachage



## Partitionnement par intervalle

Domaine de valeur de la clé divisé en $n$ intervalles
- chaque intervalle définit un fragment




**Remarque**

Si nombre de fragments est très grand, la recherche du bon fragment peut être long ($O(N)$)
$\Rightarrow$ on peut utiliser un B-arbre (idem index BD relationnelle)

*Search Trees*

- initialement, 1 seul fragment couvrant la totalité du domaine de la clé
- quand fragment plein
  - éclatement du fragment en 2 parties égales correspondant à 2 intervalles distincts

*Exemple :* si 8 documents max par fragments


La collection est totalement ordonnée :

- au niveau de chaque fragment
- ordre des fragments dans la structure de hachage



## Partitionnement par hachage

Idée de base

- disposer d'une table de correspondance entre des entiers (clés) et des fragments
  - utilisation d'une fonction de hachage
    - si $n$ fragments, $h(cle) \rightarrow [0,\ n-1]$

**Problème**

Pas dynamique car ajout d'un fragment
$\Rightarrow$ modification de la fonction de hachage

Solution : **Hachage cohérent**



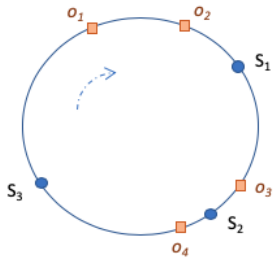
## Hachage cohérent

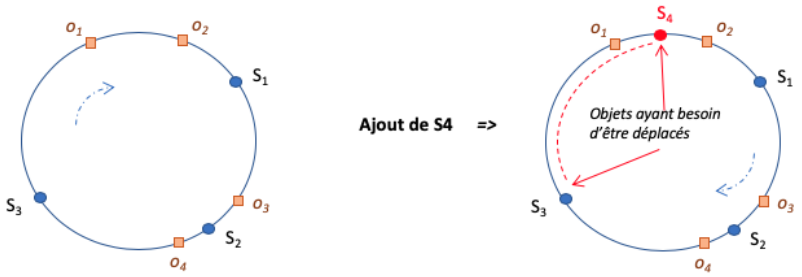
1. Utilisation d'une fonction de hachage non mutable $h(cle) \rightarrow [0,\ n-1]$
   - $[0,\ n-1]$ : espace d'adressage
   - $n$ très grand : en général $2^{32}$ ou $2^{64}$
   - s'applique sur
     - l'adresse IP des serveurs
     - la clé des objets
2. Espace d'adressage
   - organiser en anneau
   - parcouru dans le sens des aiguilles d'une montre

**Consistent Hashing mapping rule**

Si $S$ et $S'$ sont 2 serveurs adjacents sur l'anneau, toutes les clés de l'intervalle $]h(S), h(S')]$ sont stockées sur $S'$.







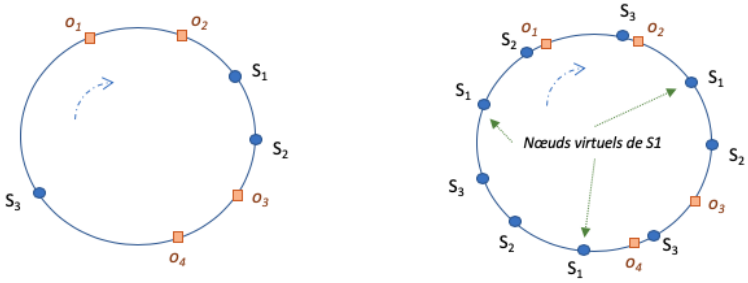

Ajout d'un nouveau nœud

- le positionner sur l'anneau en fonction de sa valeur de hachage
- une partie des objets stockés dans son successeur doivent être déplacés
  - réorganisation locale car les autres nœuds ne sont pas affectés



**Problème**

Répartition de la charge des serveurs inégales car dépend de la fonction de hachage
⇒ utilisation de nœuds virtuels






- Localisation d'un serveur sur l'anneau correspond à une machine virtuelle
- Plusieurs machines virtuelles peuvent être hébergées sur le même serveur physique

Quand un serveur physique est ajouté ou retiré de l'anneau, le déplacement des objets est partagé par tous les successeurs des nœuds virtuels, ce qui évite de surcharger un seul serveur.

Cette "virtualisation" permet de régler des problèmes d'hétérogénéité des serveurs en assignant à chaque serveur un nombre de nœuds virtuels en fonction de sa capacité.