CCIN2P3

cnrs

Centre de Calcul

de l'Institut National de Physique Nucléaire
et de Physique des Particules

ANF Ceph 2022
Bases de stockage distribué

Loïc Tortay, tortay@cc.in2p3.fr

Pourquoi ?

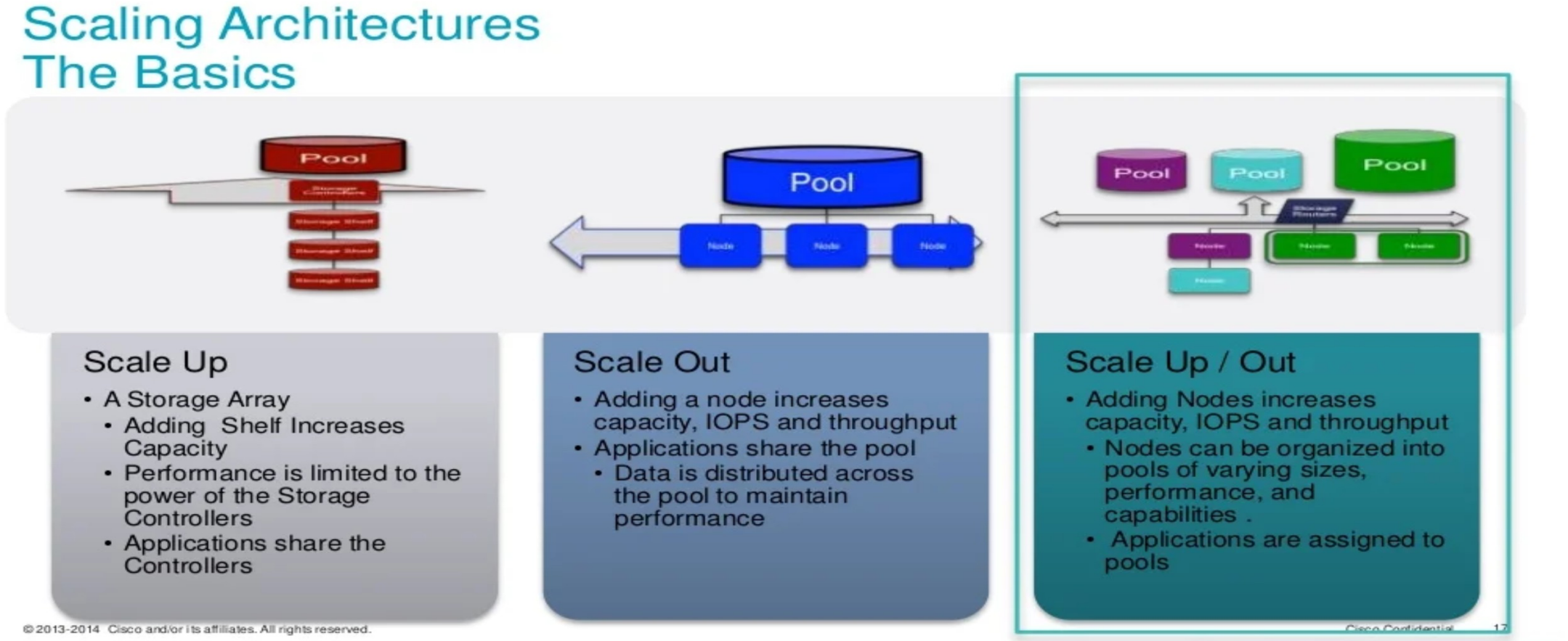
- Stockage distribué pour pallier aux limites d'un serveur unique :
  - capacité
  - performance
  - disponibilité des données
- Limites de la redondance « classique » (RAID « standard ») :
  - tailles des périphériques individuels & volumes RAID
  - souplesse de configuration
  - temps de reconstructions
- Coût (& complexité perçue) des réseaux de stockage spécialisés (SAN)

Infrastructures de stockage distribué

- Avec partage (de ressources de stockage) :
  - périphériques partagés par 2 serveurs ou plus (redondance des accès aux données)
  - disponibilité des données indépendante de la disponibilité d'un serveur précis (application de stockage idoine nécessaire)
  - topologies SAN éventuellement complexes (routage)
- Sans partage (*shared nothing*) :
  - modèle dominant actuellement (*hyperscalers*, Ceph, ...)
  - disponibilité des données liée à la disponibilité du serveur qui les contient (éventuellement redondance des données sur d'autres serveurs)

Scale-out & brique de base

- *Scale-out* : infrastructure qui croît par ajout de « briques de base »



Cisco

Infrastructures de stockage distribué ?

- Avec partage (de ressources de stockage) :
  - coût à l'espace utile souvent plus faible
  - croissance de type *scale-out* avec des briques de base
  - souvent préféré pour les systèmes de fichiers parallèles (HPC)
- Sans partage (*shared nothing*) :
  - coût au serveur en général plus faible
  - croissance de type *scale-out* avec des briques de base
  - souvent préféré pour les systèmes de stockage « modernes » (dont Ceph)
  - redondance des données souvent par réplication (coût en capacité)

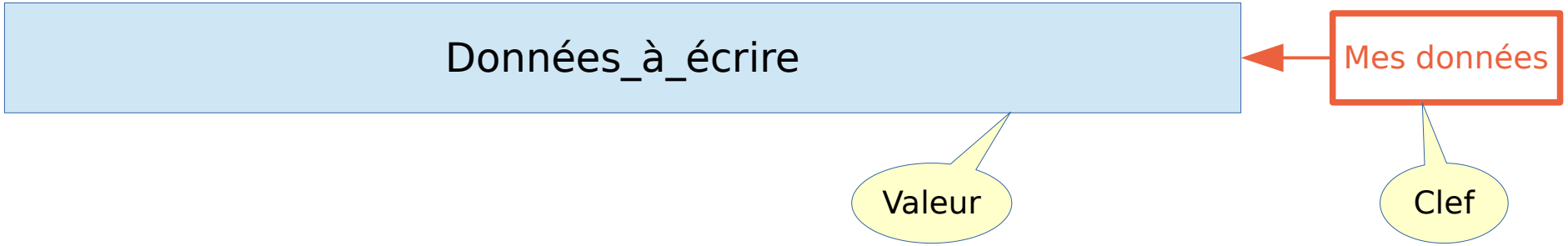
Distribution des données

- Données distribuées sur les différents serveurs
  - sans relation directe avec la vue utilisateur (espace de nommage ou autre)
  - avec une granularité qui peut être variable
- Souvent, distribution dépendant (entre autres) du résultat d'une fonction de hachage :
  - parfois des données elles-mêmes (*Content-Addressed-Storage*)
  - plus souvent des identifiants des données dans l'application (nom, position, numéro d'ordre, ...)
  - pour ne pas dépendre d'un unique serveur qui calcule les condensats (*hash*) ⇨ DHT (*Distributed Hash Table*)

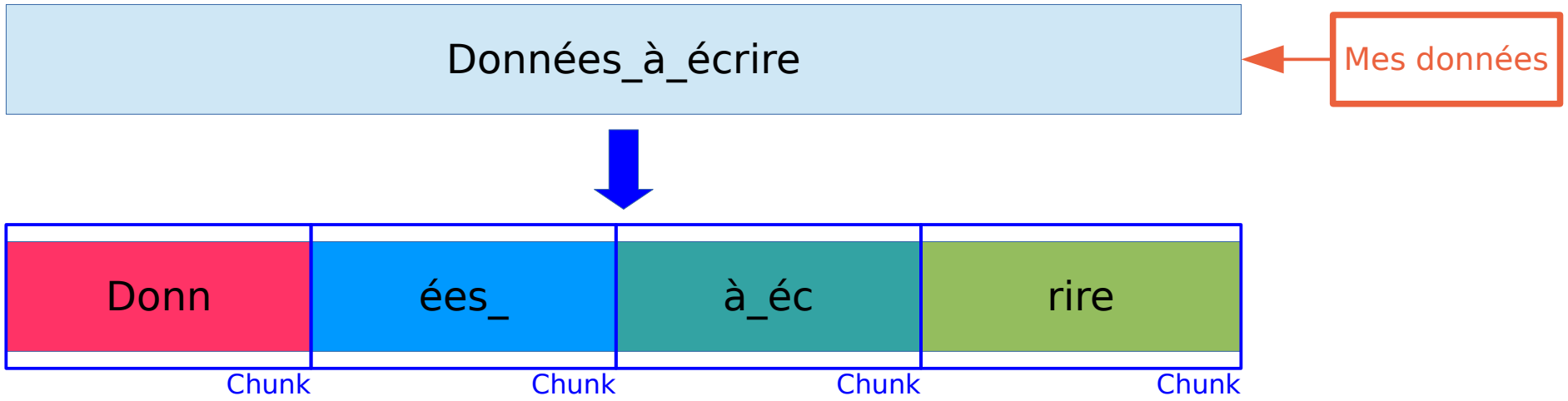
Distribution des données : principe

☞ *Exemple simplifié (voire simpliste) de système clef/valeur **non** basé sur un système particulier*

Distribution des données : principe

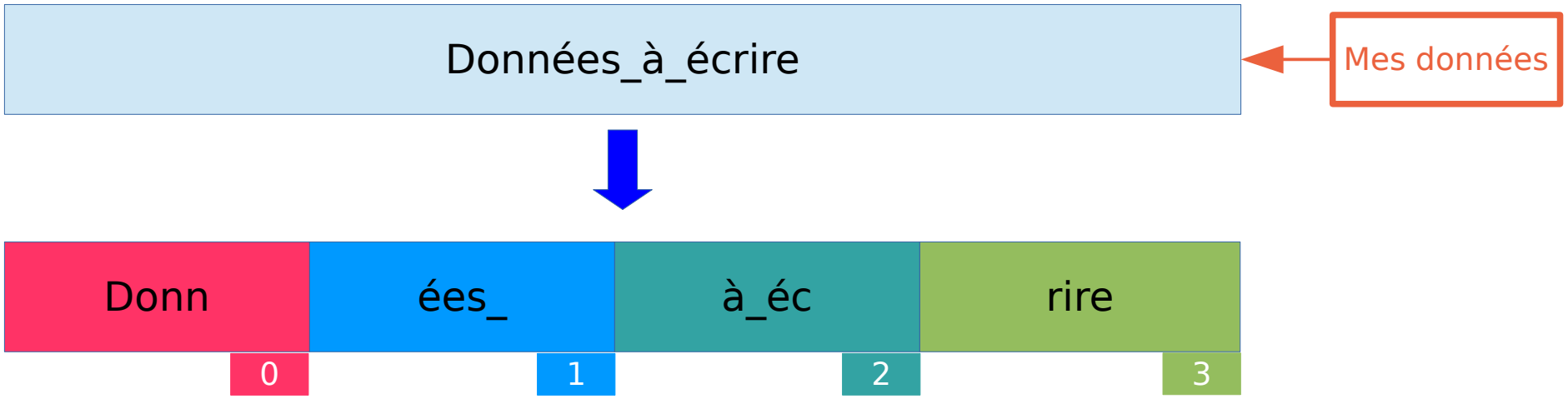


Distribution des données : principe



*Les données sont découpées en morceaux, appelés **ici** « chunks »*

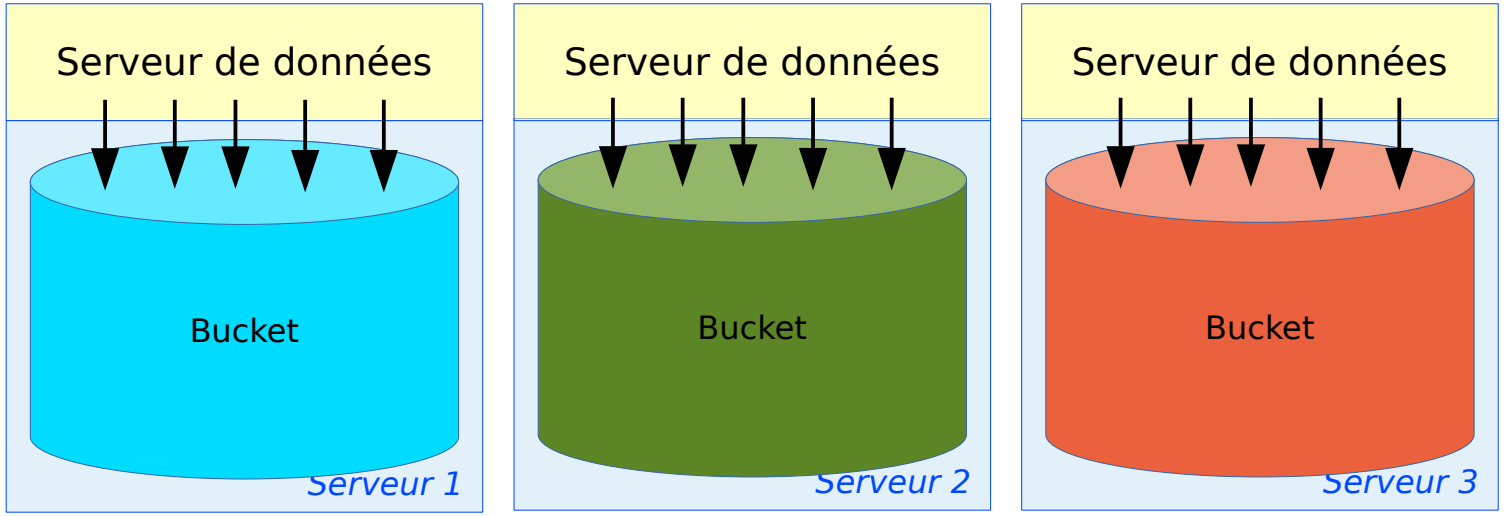
Distribution des données : principe



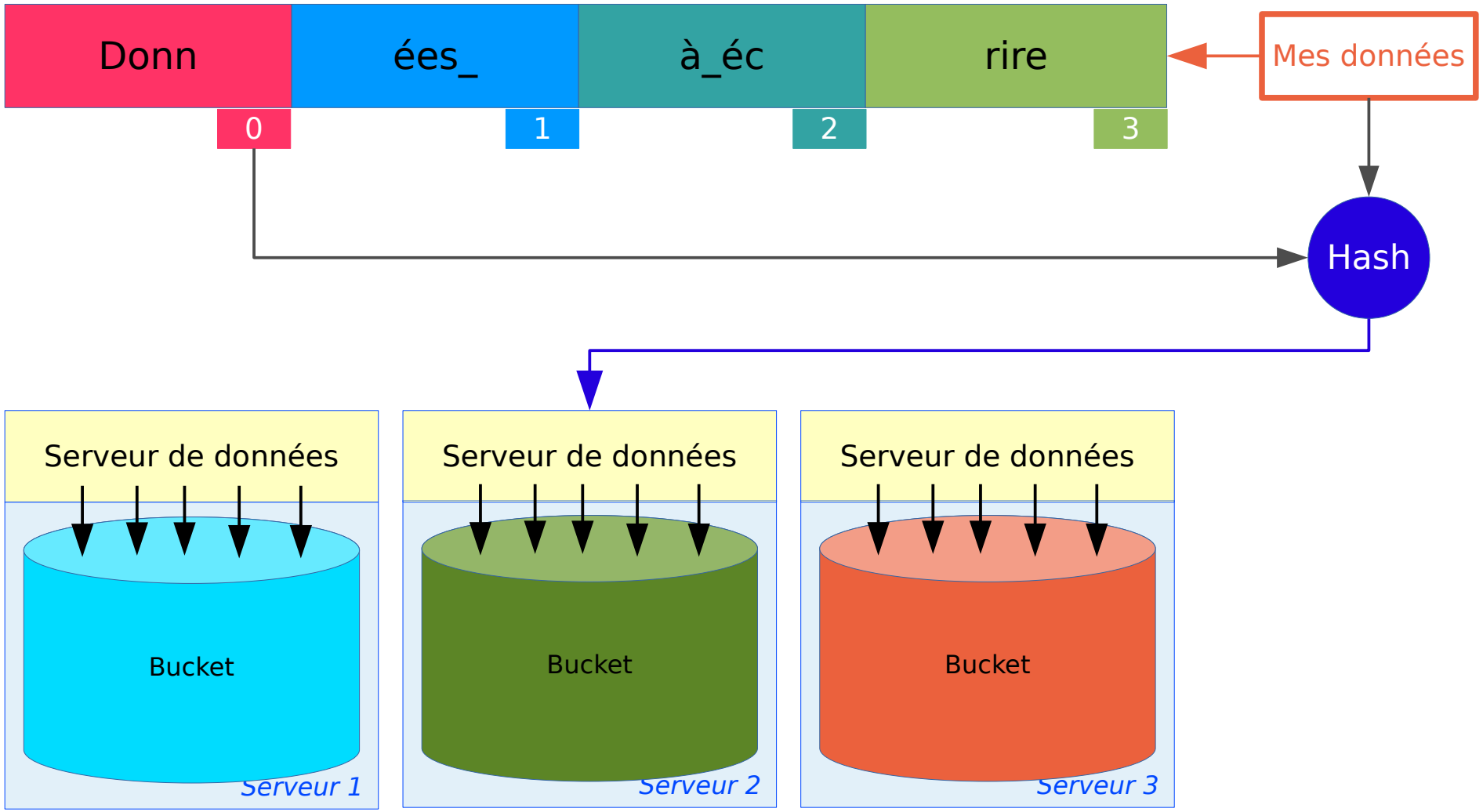
On associe un numéro d'ordre à chaque « chunk »

Distribution des données : principe

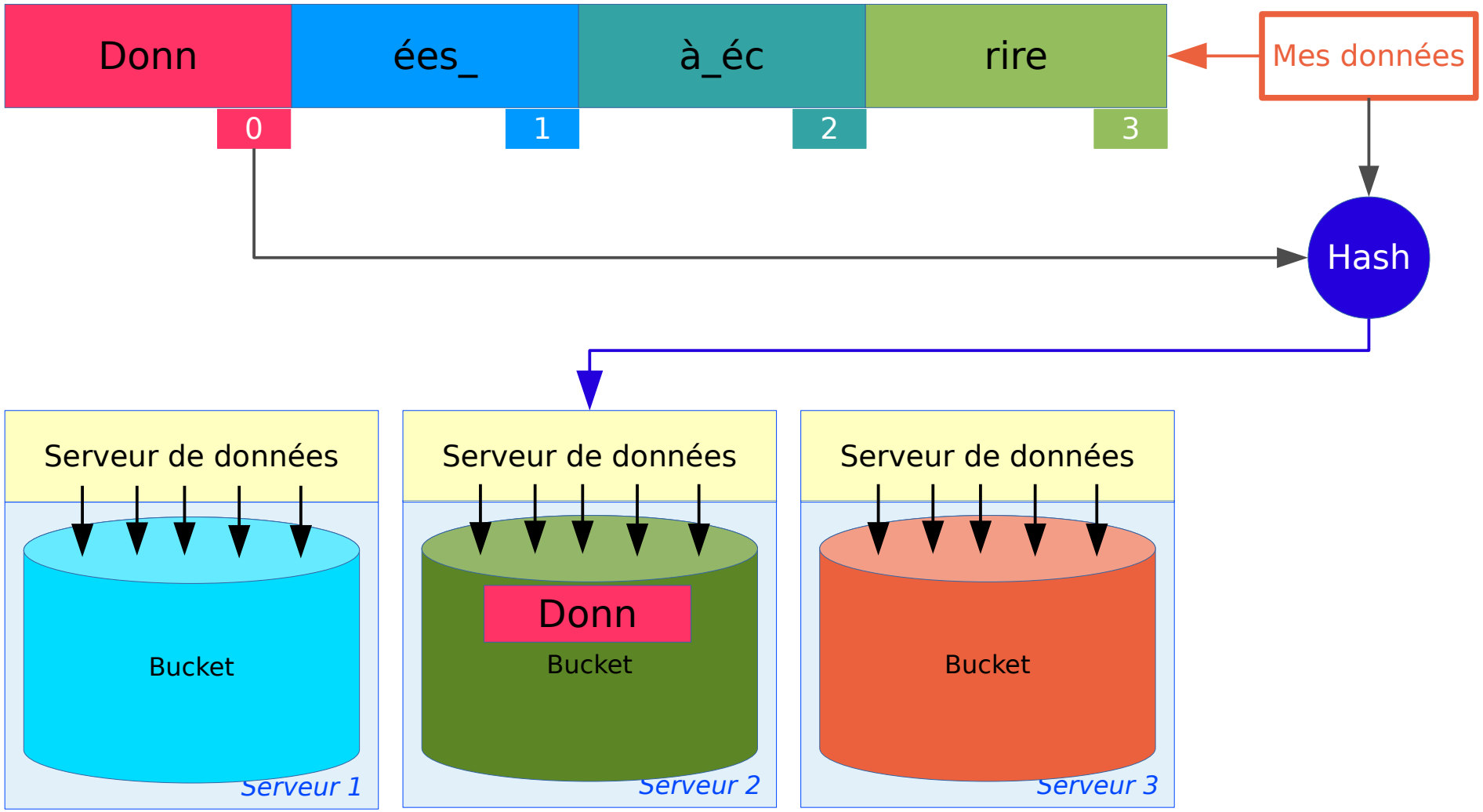
*Les serveurs fournissent chacun une « portion » de stockage appelée **ici** « bucket », dans certains systèmes un serveur a plusieurs « portions » (bucket, shard, OST, OSD, NSD, slice, ...)*



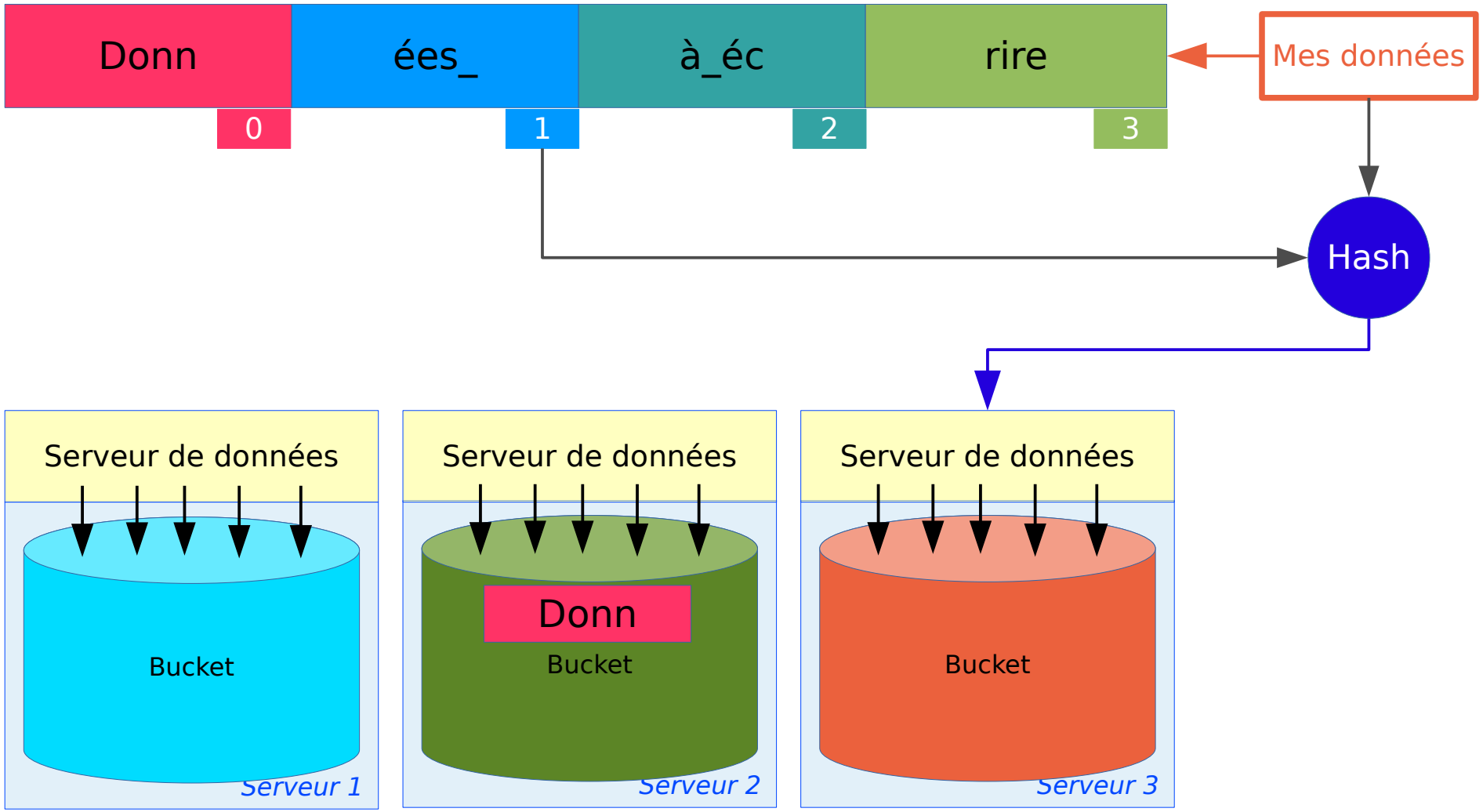
Distribution des données : principe



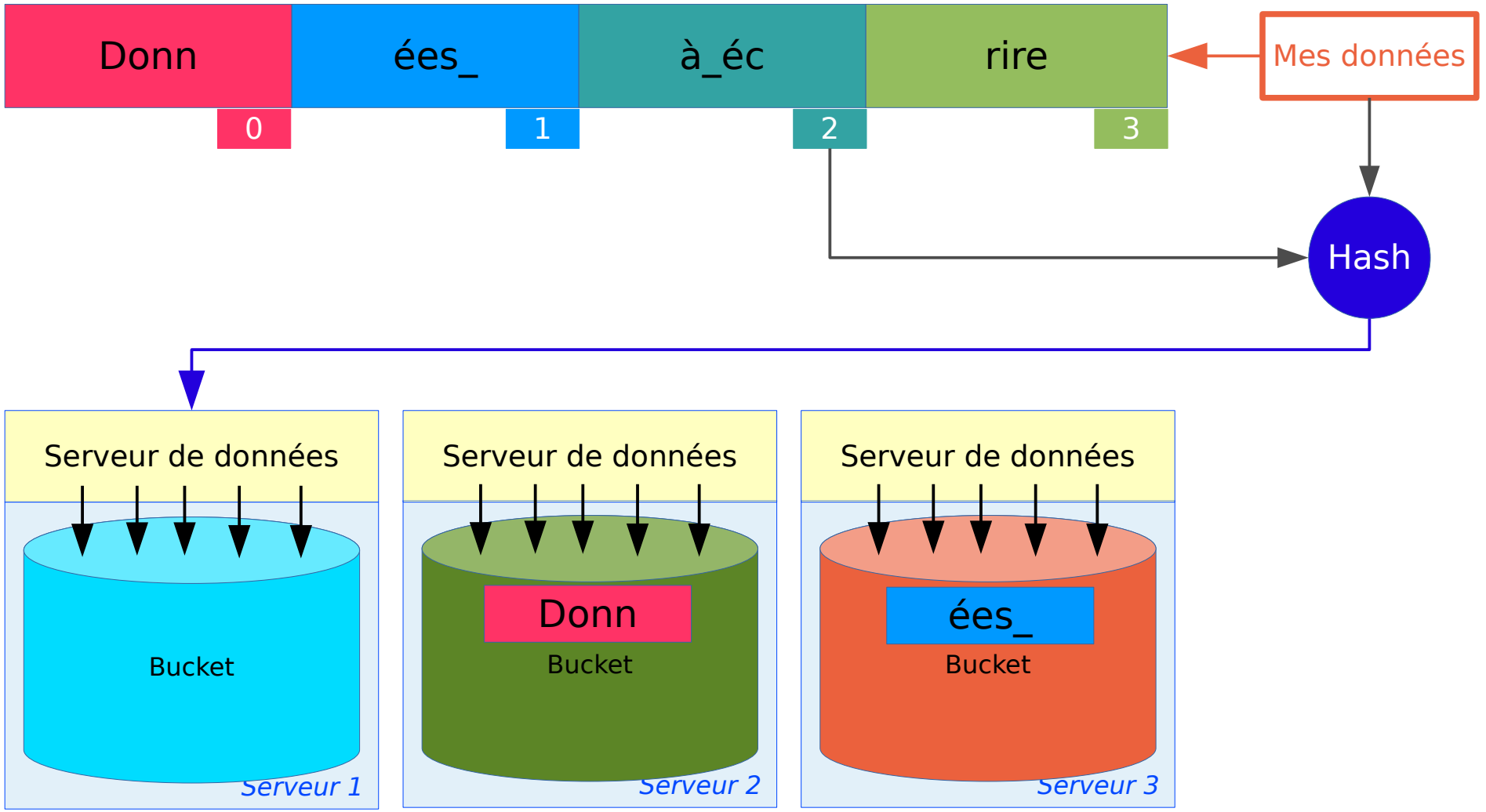
Distribution des données : principe



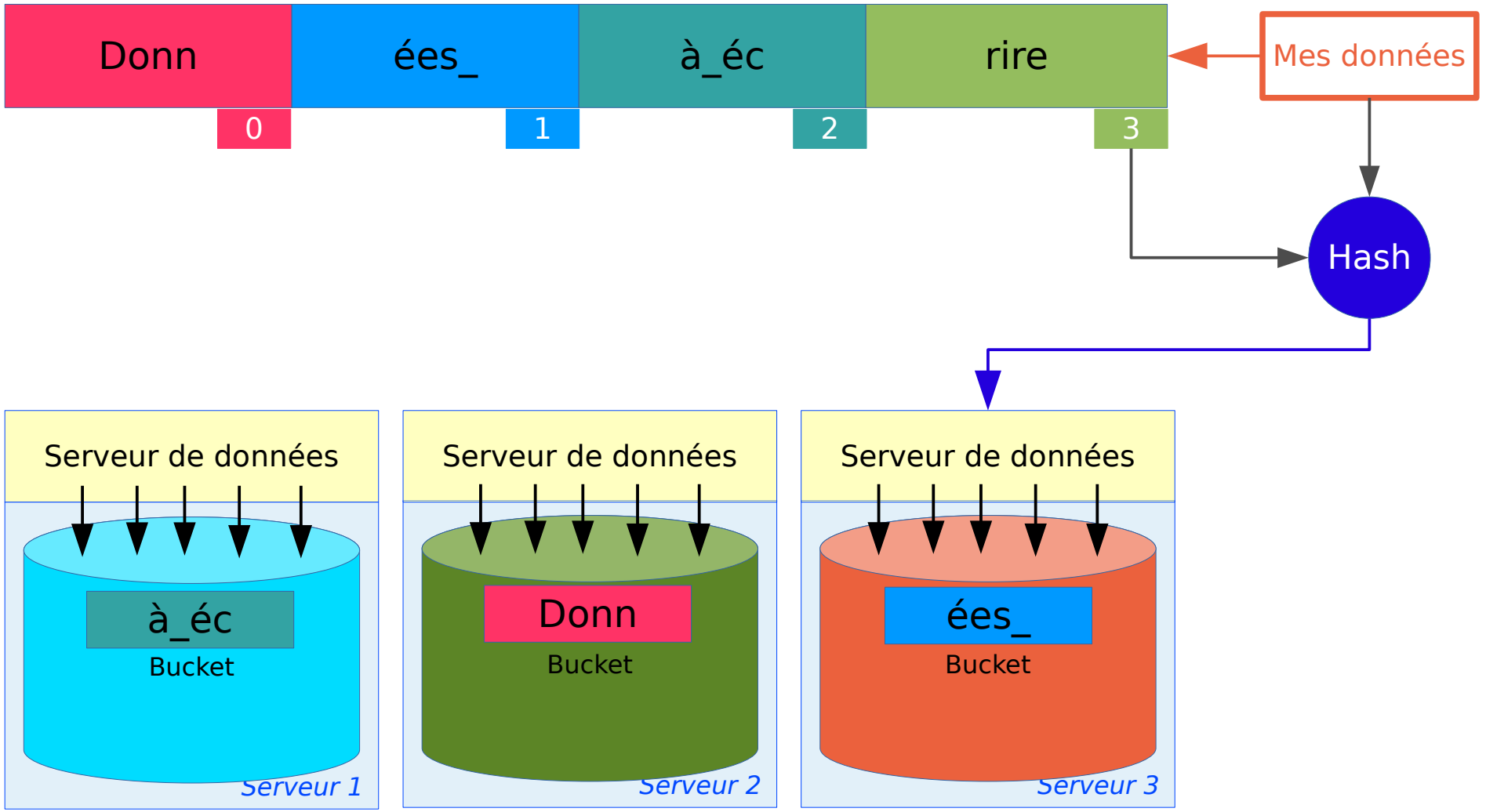
Distribution des données : principe



Distribution des données : principe

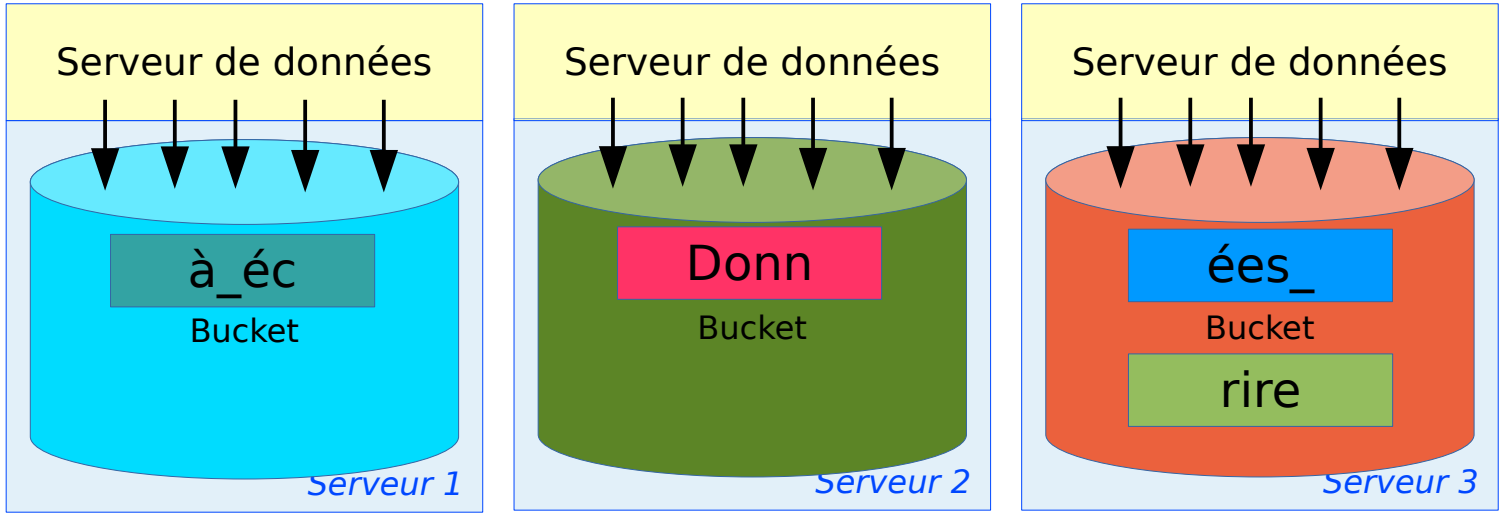


Distribution des données : principe

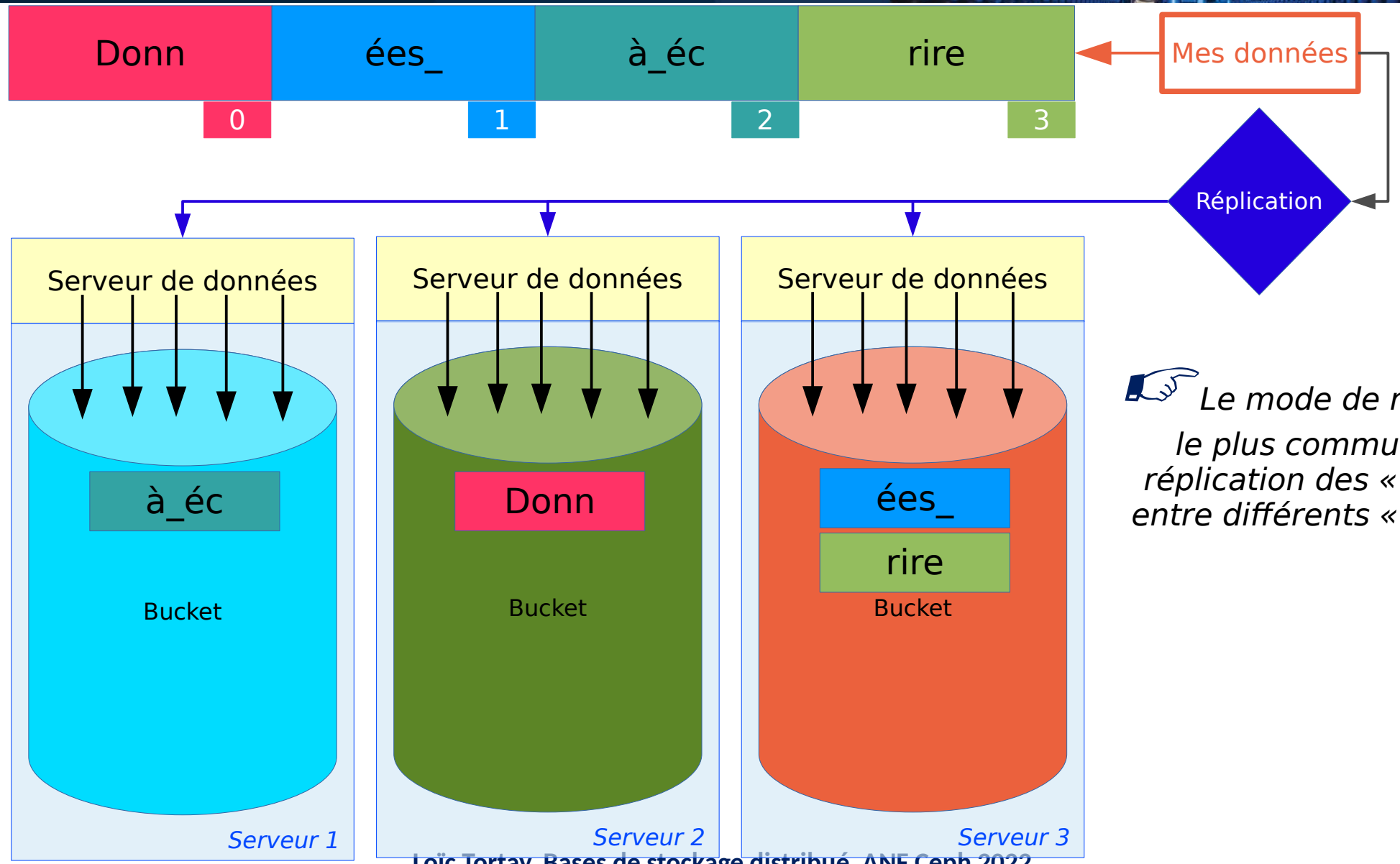


Distribution des données : principe

☞ *Sans redondance sur les serveurs de « buckets », on ne fait que de la dispersion des données*

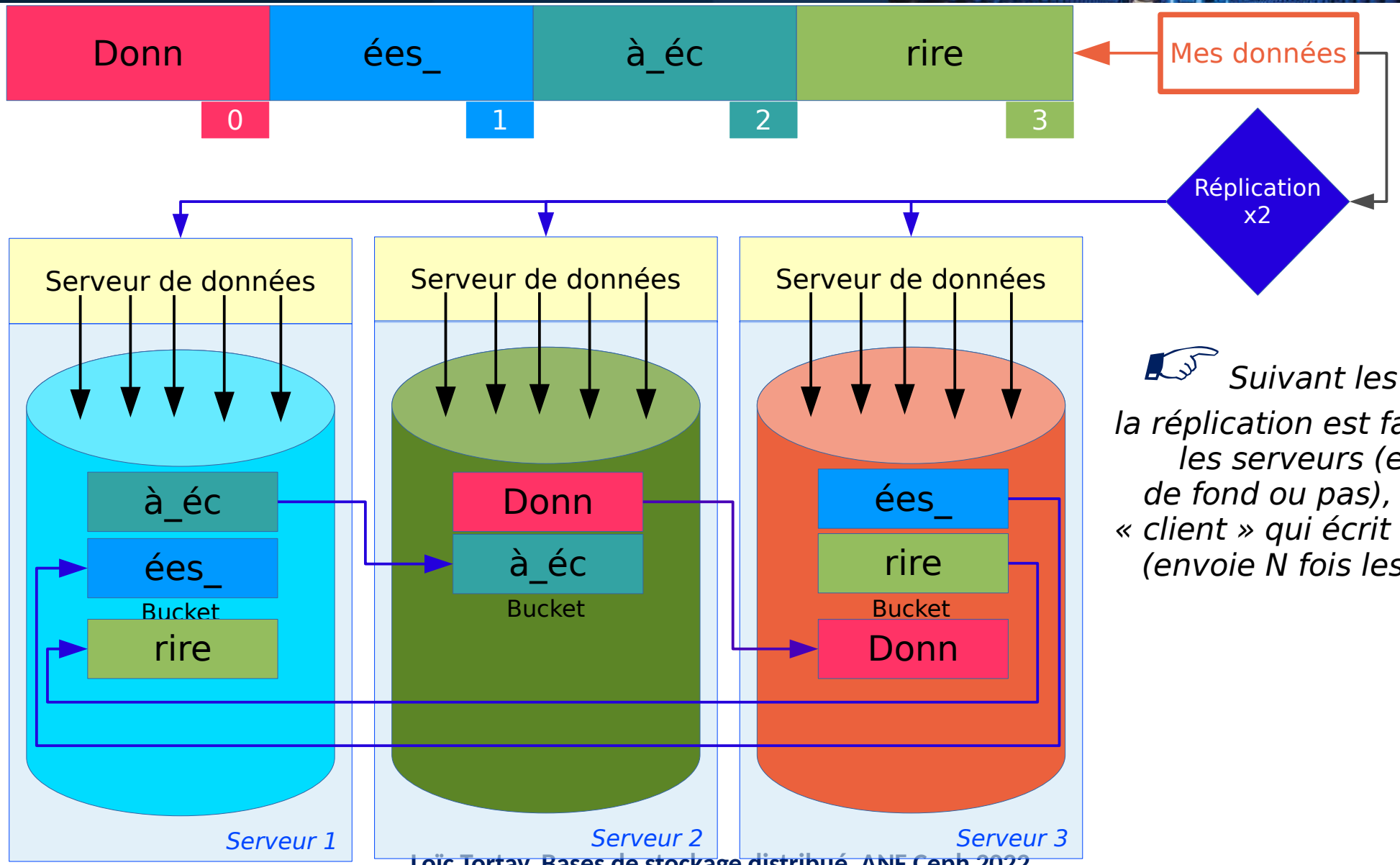


Distribution des données : principe



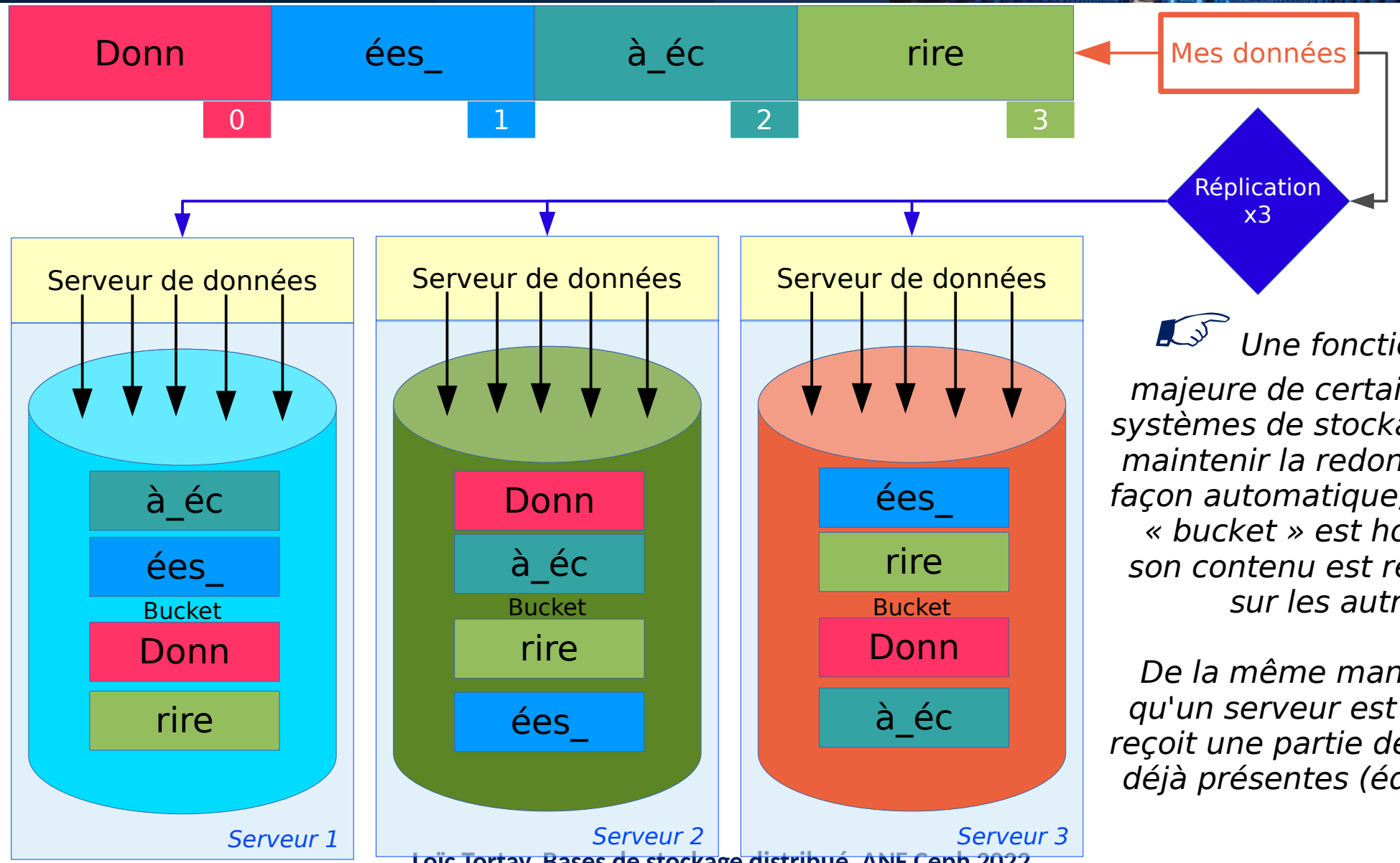
Le mode de redondance le plus commun est la réplication des « chunks » entre différents « buckets »

Distribution des données : principe



Suivant les systèmes, la réplication est faite soit par les serveurs (en tâche de fond ou pas), soit par le « client » qui écrit les données (envoie N fois les données)

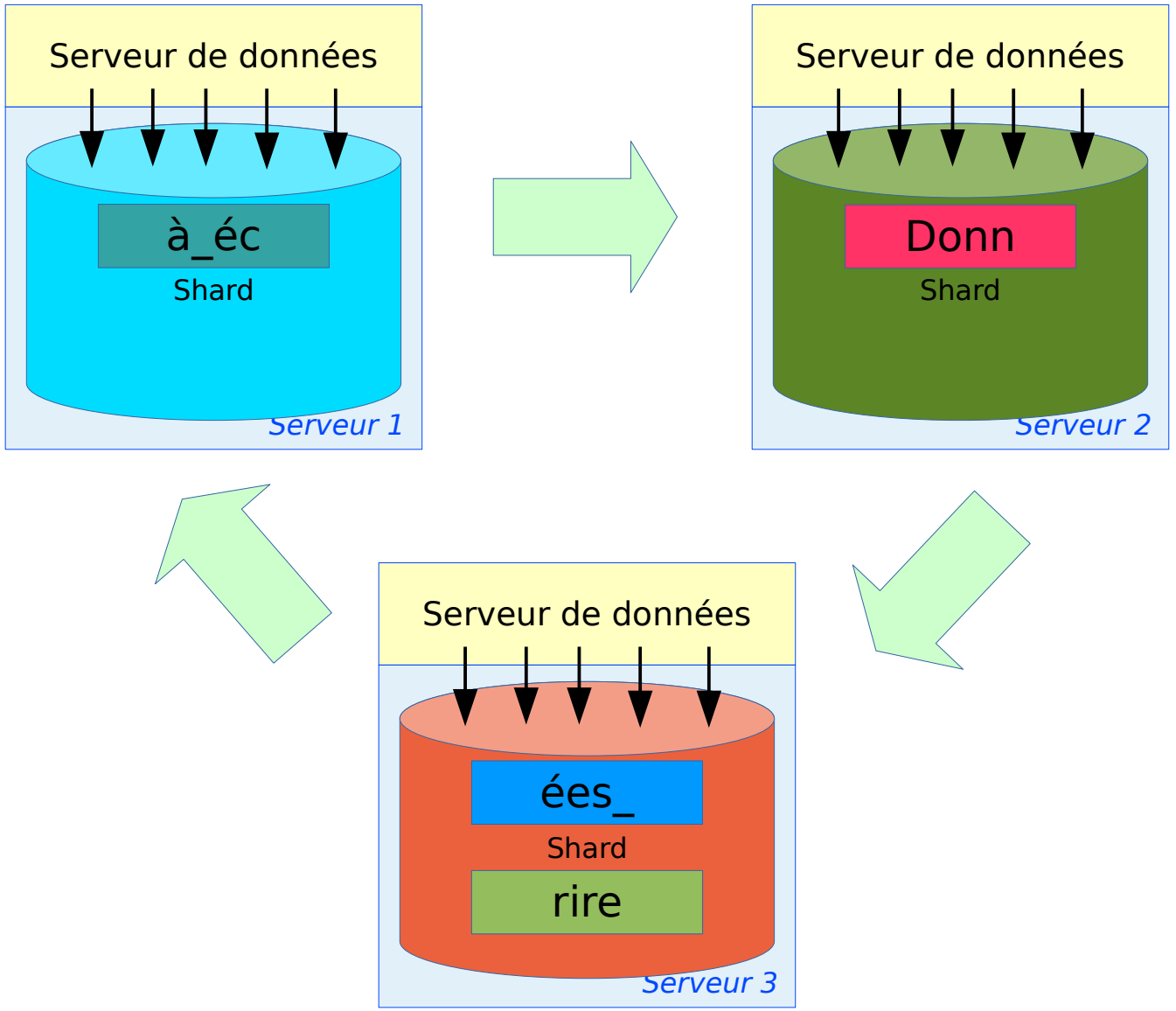
Distribution des données : principe



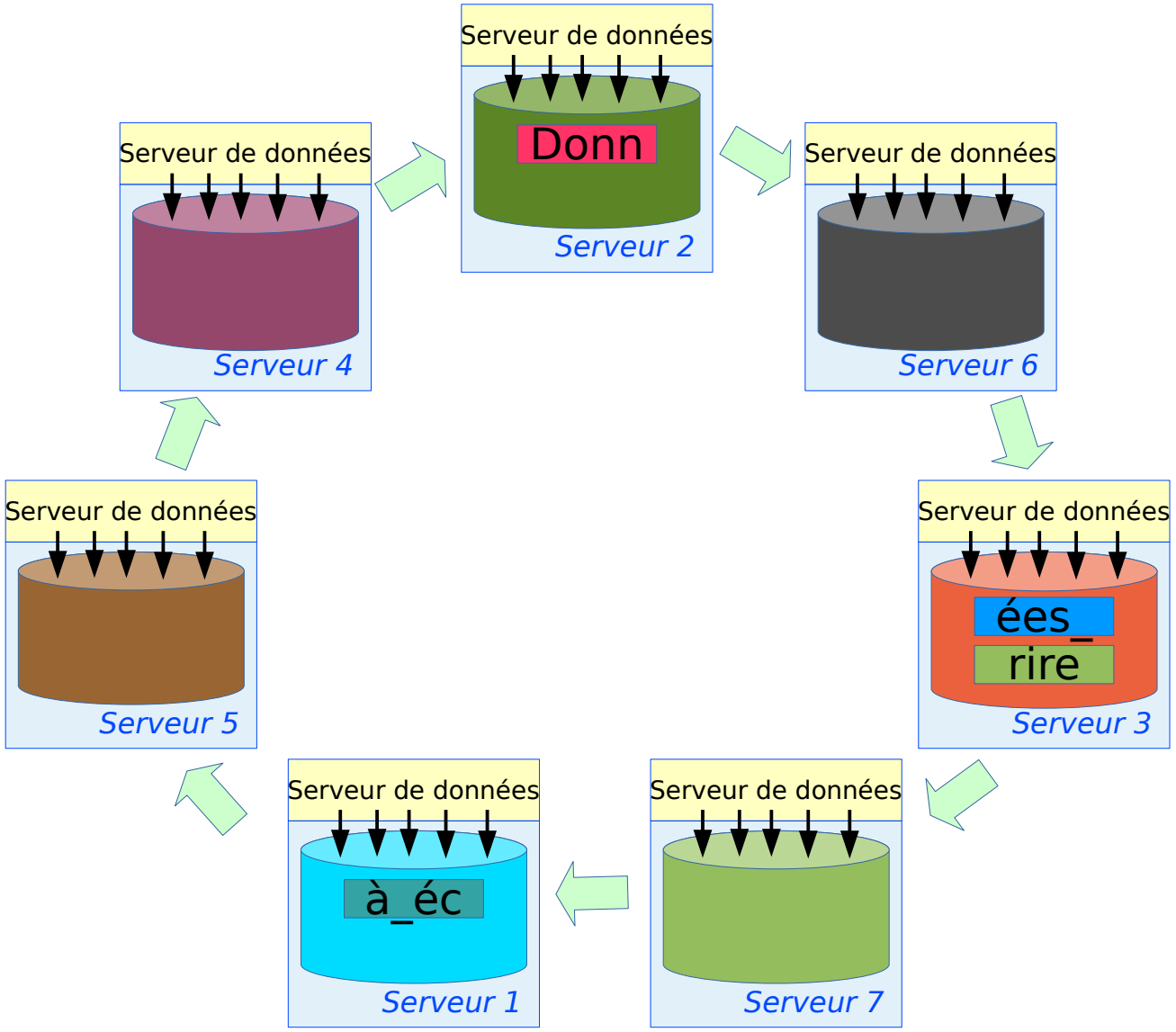
Une fonctionnalité majeure de certains de ces systèmes de stockage est de maintenir la redondance de façon automatique, dès qu'un « bucket » est hors-ligne, son contenu est reconstruit sur les autres.

De la même manière, dès qu'un serveur est ajouté, il reçoit une partie des données déjà présentes (équilibrage)

Distribution des données : anneaux [1/4]



Distribution des données : anneaux [2/4]

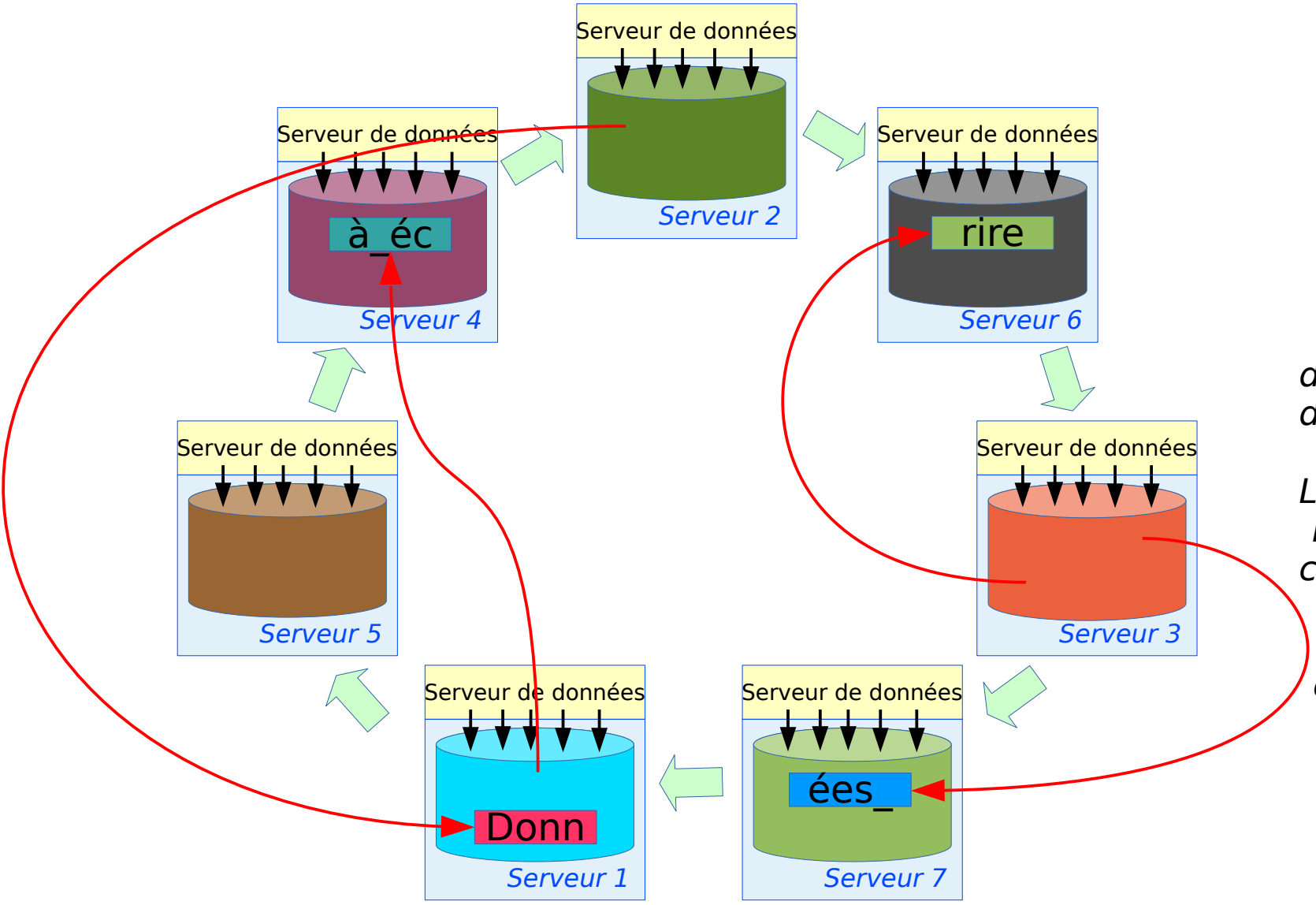


Espace des valeurs de la DHT

*(produites **ici** à partir des identifiants des données -- clef -- & numéro d'ordre) découpé/partagé entre les membres de l'anneau.*

Les nouveaux membres s'insèrent entre les membres existants et prennent en charge une partie des valeurs de la DHT

Distribution des données : anneaux [3/4]



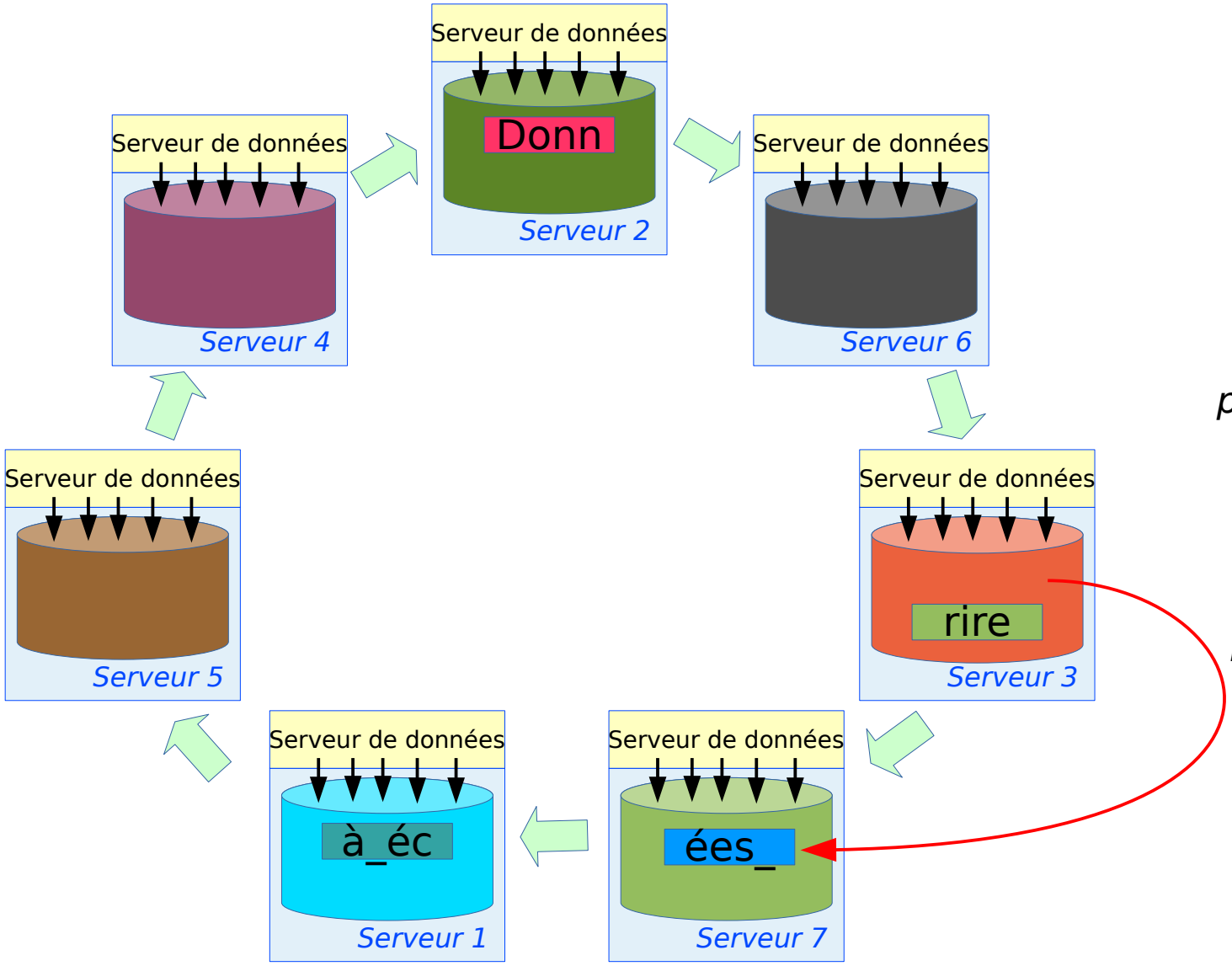
Espace des valeurs de la DHT

*(produites **ici** à partir des identifiants des données -- clef -- & numéro d'ordre) découpé/partagé entre les membres de l'anneau.*

Les nouveaux membres s'insèrent entre les membres existants et prennent en charge une partie des valeurs de la DHT

Redistribution des données lors des changements de taille (ajout ou retrait de serveur)

Distribution des données : anneaux [4/4]



Les fonctions de hachage « stables » (consistent hashing), permettent de limiter la quantité de données à redistribuer lors des changements de taille de la table

Par rapport au cas naïf précédent, on redistribue environ 1/n de l'ensemble des données au lieu de la totalité des données (ajout d'1 serveur à n)

Distribution des données : ~~anneaux~~ arbres

- Ceph n'utilise pas un anneau mais un arbre de distribution
- L'algorithme de distribution de Ceph s'appelle CRUSH (*Controlled Replication Under Scalable Hashing*) qui est une forme de hachage stable et pondéré
- Le résultat de la fonction de hachage donne l'identifiant du *PG (placement group),* le PGID et la topologie courante du système déterminent les disques (serveurs) qui vont contenir les données
- Pour plus de détails :
  - https://www.slideshare.net/sageweil1/a-crash-course-in-crush
  - https://en.wikipedia.org/wiki/Rendezvous_hashing#Weighted_variations
  - ...

Redondance des données (distribuées)

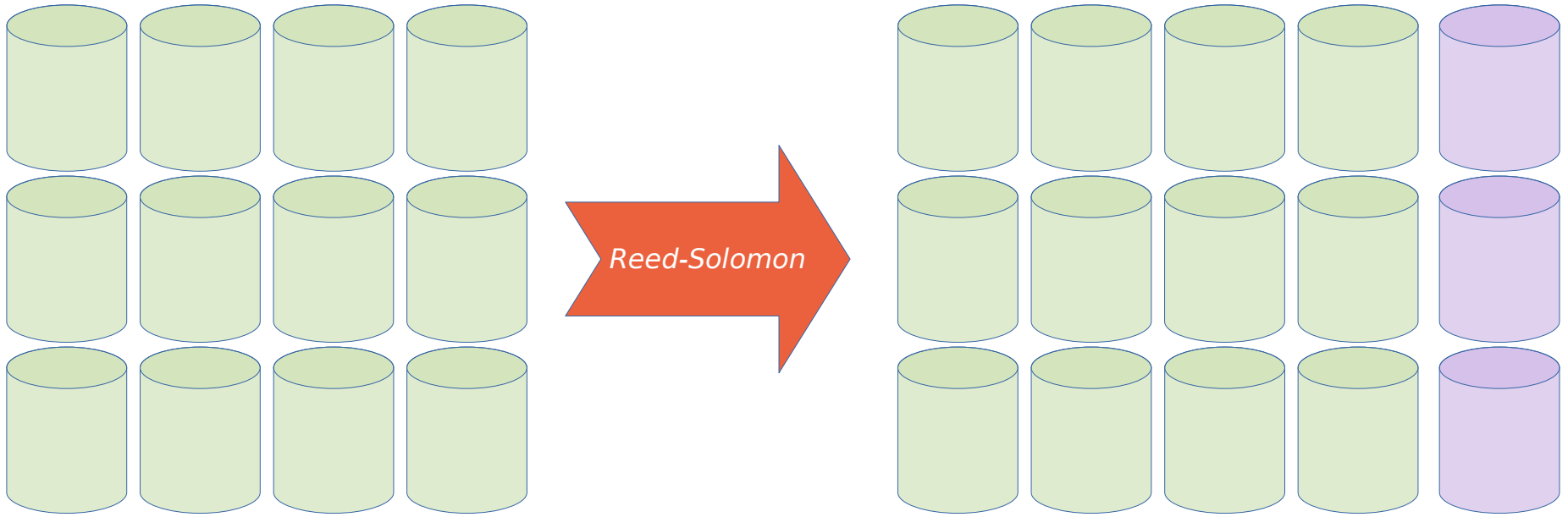
- Réplication :
  - entre composants d'un serveur ou entre serveurs
  - « facile » mais coûteuse en espace
  - difficulté majeure : cohérence des réplicats (consensus)
- Codes correcteurs à effacement (*erasure coding)* :
  - au sein d'un serveur (RAID ou équivalent) ou entre serveurs
  - « difficile » mais économique en espace
  - difficulté majeure : performance des reconstructions

Codes correcteurs (à effacement)

- « à effacement » : reconstruire des données partiellement manquantes (*erasure coding*)
- Stocker (ou transférer) les données avec une certaine redondance sans avoir plusieurs copies complètes
- Pour le stockage, les codes correcteurs de type Reed-Solomon sont très communs (par exemple RAID-{3,4,5,6}, CD/DVD, etc.)
- Données découpées en morceaux de taille fixe (blocs), redondance soit :
  - combinée avec les données (codes non-systématiques)
  - stockée « à côté » des données (codes systématiques) dans des blocs dédiés

Reed-Solomon facile

- Notation courante : RS(n, k)



k blocs de données

n blocs = k blocs de données
*+ **m blocs de codage***

Reed-Solomon trivial

- On veut stocker/transférer 3 blocs : `12 34 56`
- On peut calculer une « somme de contrôle » (*checksum*), souvent appelée « parité » :
  - `C1 = 12 + 34 + 56 = 102`
- On transfère/lit les 3 blocs de données et la parité : `12` **`32`** `56 102`
  - `C1' = 12 + 32 + 56 = 100 ≠ 102`
  - ⇨ les données lues/transférées **sont** corrompues
- Avec un bloc de contrôle, on peut détecter la présence d'une erreur mais pas sa position exacte (sans informations corollaires)

Reed-Solomon trivial

- On veut stocker/transférer 4 blocs : `87 65 43 21`
- On peut calculer deux blocs de contrôle :
  - `C1 = 87 + 65 + 43 + 21 = 216`
  - `C2 = 87*1 + 65*2 + 43*3 + 21*4 = 430`
- On lit les 4 blocs de données et les deux blocs de redondance : `87 65` **`51`** `21 216 430`
  - `C1' = 87 + 65 + 51 + 21 = 224 ≠ 216`
  - `C2' = 87*1 + 65*2 + 51*3 + 21*4 = 454 ≠ 430`
  - ⇨ les données transférées **sont** corrompues
- Mais **on peut** trouver la position de l'erreur

Reed-Solomon trivial

- Amplitude de l'erreur : `E = C1 - C1' = 216 - 224 = -8`
- Position de l'erreur (bloc endommagé) :
  - `P = (C2 - C2') / E = (430 - 454) / -8 = 3`
- Le troisième bloc a été corrompu, on peut le reconstruire :
  - $Bloc3_{corrigé} = Bloc3_{reçu} + E$
- Deux blocs de contrôle permettent de :
  - localiser puis corriger une erreur (panne non franche ou passagère)
  - corriger deux erreurs si on connait leur position (disque en panne franche, ...)

Reed-Solomon moins trivial

- Dans les « vrais »codes correcteurs Reed-Solomon (y compris RAID) :
  - arithmétique modulaire dans les Corps de Galois (en général GF(2^8)) et pas arithmétique dans $\mathbb{Z}$
  - grandes tailles de blocs (centaines de milliers ou millions de bits par bloc)
  - pas de systèmes d'équations résolus au cas par cas, mais des matrices initialisées en fonction des paramètres du code utilisé (performance des reconstructions)
- Caractéristique MDS (*Maximum Distance Separable*) : pouvoir reconstruire jusqu'à **m** blocs parmi les **n**, quels que soient ces blocs
- https://en.wikipedia.org/wiki/Reed–Solomon_error_correction

Reed-Solomon en pratique (pour Ceph)

- De façon générale, $k < 10$ recommandé
- RAID-6 = RS(n, n-2) avec n = 4 ou 8 recommandé
- RAID-5 = RS(n, n-1)
- Utilisation avec Ceph :
  - RS(10, 8) : 8 + 2 (80% d'espace utile)
  - RS(11, 8) : 8 + 3 (72% d'espace utile)
  - RS(12, 9) : 9 + 3 (75% d'espace utile)
  - RS(12, 8) : 8 + 4 (66% d'espace utile)
  - Souvent un « bloc » par serveur (`crush-failure-domain=host`) ⇨ prévoir assez de serveurs

Autres codes correcteurs

- De nombreux autres types de codes à effacement que Reed-Solomon existent
- Souvent basés sur d'autres outils mathématiques
- Par exemple, en stockage :
  - Mojette : code non-systématique efficace (basé sur la transformée de Radon) qui « garantit » l'intégrité des données lues et minimise les écritures lors des modifications (base de RozoFS)
- Beaucoup de recherche sur la vitesse/l'impact de reconstruction :
  - en particulier LRC (*Local Reconstruction Codes*), disponible dans certaines versions de Ceph

Codes correcteurs ?

- Plus de souplesse de configuration avec des codes correcteurs au niveau de l'application de stockage :
  - par rapport au RAID-6, 3 ou 4 blocs de redondance possibles (et assez communs)
- Certains systèmes de stockage utilisent des codes correcteurs pour les données surtout lues (ou exclusivement lues : *sealed blocks*)
- Plus économique en espace disque que la réplication
- Au final choix basé sur le coût et la relation entre les capacités diponible, utile, de reconstruction/correction
- Temps de reconstruction

Stockage objet [1/3]

- Dualité/ambiguïté :
  - infrastructure de stockage objet (Ceph, Scality, ...)
  - interface d'accès objet (API S3, CDMI, ...)
- Les objets peuvent être vus comme une généralisation des fichiers
- Objets indépendants les uns des autres, pas nécessairement de relation entre objets
- Pas d'espace de nommage (hiérarchique) ⇨ meilleure capacité de croissance
- Standards T10/SCSI OSDv1/v2 pour les infrastructures, peu utilisés
- Pour beaucoup : « stockage objet » ⇨ « interface d'accès objet » type API HTTP REST S3)

Stockage objet [2/3]

- Applications d'infrastucture (Ceph RADOS, etc.) :
  - identifiants des objets ± indépendants des noms logiques
  - pas de répertoire ou équivalent
- Applications pour l'utilisateur final (Dropbox, iCloud, ...)
  - parfois simulation des répertoires pour fournir une vue utilisateur « confortable/familière »
- Du point de vue logique (utilisateur), données organisées en *bucket* (seau) avec généralement :
    - un utilisateur par *bucket*
    - pas de *buckets* imbriqués
    - communément plusieurs *buckets* par utilisateur

Stockage objet [3/3]

- API S3 (Ceph RGW -- RADOS Gateway, etc.) :
  - API de type HTTP(S) REST
  - PUT (écriture) d'objets complets (en général/historiquement)
  - GET (lecture) d'objets complets ou pas
  - DELETE
  - identifiants logiques des objets appelés clefs (*key*)
  - URLs du type :
    - `https://moncephs3.domaine.fr/bucket/key`
    - *bucket* et *key* sont les identifiants respectifs
  - ACL par objet ou *bucket*

Disponibilité des données distribuées [1/2]

- Architectures sans partage de ressources de stockage
- Mais souvent d'autres ressources sont partagées :
  - équipements réseau(x), blocs de distribution électrique, armoires, salle machine, ...
- Certains systèmes de stockage utilisent des informations sur ces ressources partagées pour améliorer ou garantir la disponibilité des données
- En pratique :
  - on décrit la topologie du matériel en groupant les équipements, qui partagent des ressources, dans des groupes de disponibilité (*failure group, failure domain, protection domain, availability zone*, ...)
  - le système répartit les données entre ces groupes de disponibilité

Disponibilité des données distribuées [2/2]

- Dans Ceph, ce sont les informations de topologies utilisées par l'algorithme CRUSH (p. 25)
- On indique :
  - dans quelles armoires sont les serveurs
  - dans quelles allées/salles machines sont les armoires
  - etc.
- Ceph répartit les données de façon à maintenir l'accès aux données en cas de problème sur des groupes de composants
- Les usages communs (serveurs dans une seule armoire, quelques armoires, deux salles machines, etc.) sont gérés facilement

Cohérence globale [1/2]

- Difficulté : s'assurer que les différentes machines impliquées dans un système (de stockage) distribué ont la même vision :
  - des données
  - de l'état des autres machines
- Réseaux très (mais pas complètement) fiables :
  - gérer les problèmes de communication (partition réseau)
  - dans les cas extrêmes les conflits entre *vues* divergentes (*split brain*)
- Cohérence globale « relachée » ou « forte »

Cohérence globale [2/2]

- Cohérence globale :
  - « relachée », obtenue au bout d'un certain temps (*eventual consistency*) ⇨ nombreuses bases de données distribuées NoSQL (données non critiques, jeux en lignes, réseaux sociaux, etc.)
  - « forte » avec des protocoles de consensus (Paxos, Raft, ...) ⇨ Ceph, GPFS, ExtreemFS/Quobytes, Zookeeper, MongoDB, ...
- Protocole de consensus pour :
  - déterminer l'état des autres machines, éventuellement exclure celles qui ne répondent pas correctement
  - voter lorsque c'est nécessaire pour obtenir la majorité (*quorum*)

Cohérence globale : CAP [1/2]

- Théorême CAP (*Consistency, Availability, Partition-Tolerance*) ⇨ choix nécessaire entre :
  - cohérence globale « forte » ou pas
  - disponibilité : « toujours » répondre aux requêtes clients ou pas
  - résistance aux pannes partielles, en particulier du réseau (partition ici avec le sens problème de communication)
- Le théorême indique que seules deux des trois caractéristiques peuvent être satisfaites :
  - résistance aux pannes réseaux indispensable
  - choix limité à une des autres caractéristiques (en cas de problème réseau)

Cohérence globale : CAP [2/2]

- En cas de partition, choisir entre :
  - disponibilité ⇨ répondre aux demandes, mais peut-être avec des données pas à jour
  - cohérence ⇨ ne répondre aux demandes **que** si les données sont à jour
- PACELC (« CAP dans la vraie vie ») : le réseau marche en général bien donc choix entre :
  - latence (réponse rapide peut-être pas à jour)
  - cohérence (réponse la plus à jour, mais peut-être lente)

C'est tout ?

- ANF Stockage distribué 2016 (Benoît Parrein) :
  - https://indico.mathrice.fr/event/5/contributions/717/attachments/742/995/04_-_Stockage_distribue_-_Benoit_PARREIN.pdf
- ANF Ceph 2017 (version précédente, avec un focus un peu différent, de ces transparents) :
  - https://atrium.in2p3.fr/nuxeo/nxfile/default/4b3c95f4-ea12-4476-a632-40b7af5baa79/files:files/0/file/bases-de-stockage-distribue--ecole-informatique-in2p3-cnrs-2017--loic-tortay.pdf
- https://docs.ceph.com/
- ...