Bases de stockage distribué

Pour les administrateurs systèmes (et en 3 heures)

École informatique IN2P3/CNRS stockage distribué

Loïc Tortay, 12 juin 2017

Plan

- Matériel pour le stockage (distribué)
- Redondance :
  - RAID
  - Code correcteurs à effacement *(erasure coding)*
  - Réplication
- Infrastructures de stockage distribué
- Distribution des données
- Stockage objet
- Disponibilité & cohérence

Matériel pour le stockage (distribué) : médias (bloc)

- Disques durs toujours dominants :
  - SAS-NL (SATA + interface SAS) : capacitifs, presque uniquement format 3.5", typiquement 5.4krpm ou 7.2krpm, capacité en To (2-12 To)
  - SAS : disques « performants », souvent 10krpm ou 15krpm, très souvent format 2.5", capacité en Go (300-1200 voire 1800 Go)
- NVM *(Non Volatile Memory)*
  - SSD : majoritairement Flash NAND au format « disque », interface SAS ou SATA, 2.5" dominant, SSDs *write-intensive* : capacité & fonctions supplémentaires pour pallier aux limites de la Flash NAND (granularité *pages* vs *blocs*, *write amplification*, *garbage collection*)

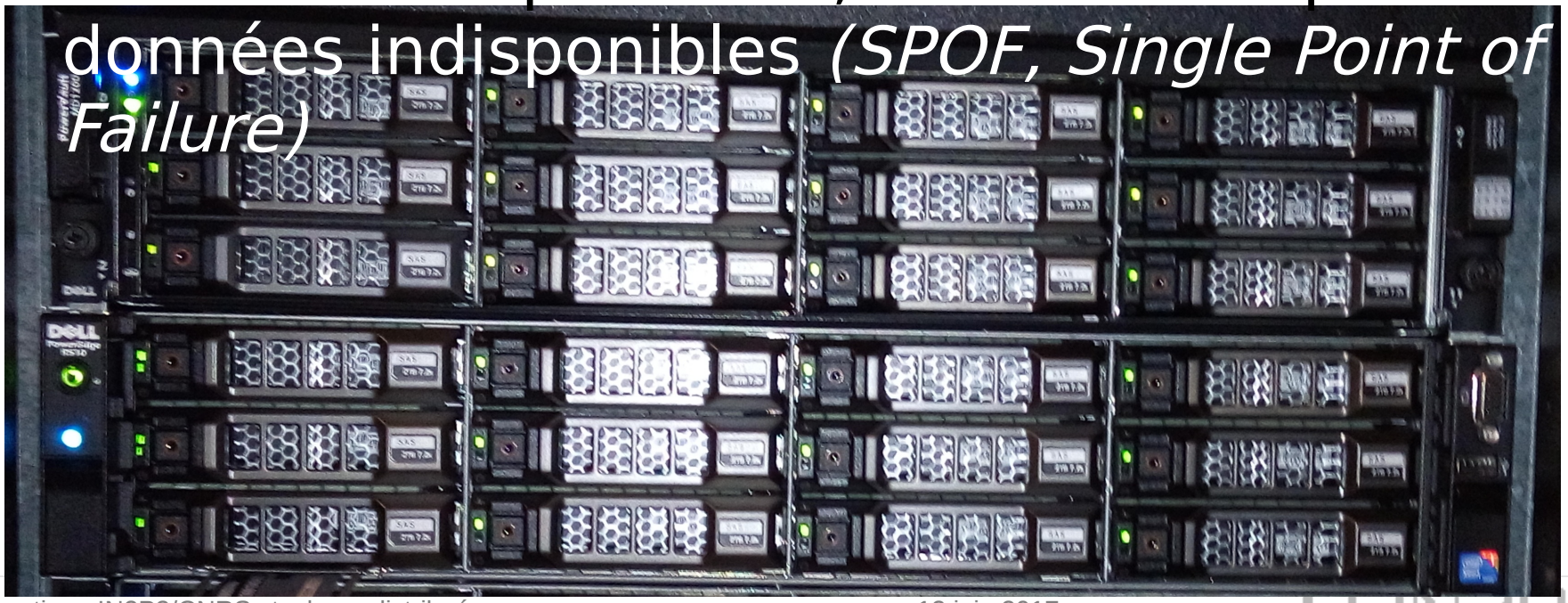
Matériel pour le stockage (distribué) : médias

- NVMe : Flash NAND ou autre NVM, sous forme de carte PCIe ou M.2
- « Flash as RAM » : barrettes DIMM

- « Bientôt » NVRAM (pas juste NVM) :
  - PCRAM, MRAM, RRAM/ReRAM, STTRAM, ...
  - Intel/Micron 3D-Xpoint (Optane), pour l'instant vendue comme NVM
- Disque « Objet » :
  - OSD (*Object Storage Device*, standard SCSI)
  - Seagate Kinetic (interface Ethernet)
- Performances globales :
  - RAM ≥ NVRAM > NVMe > SSD > SAS > SAS-NL

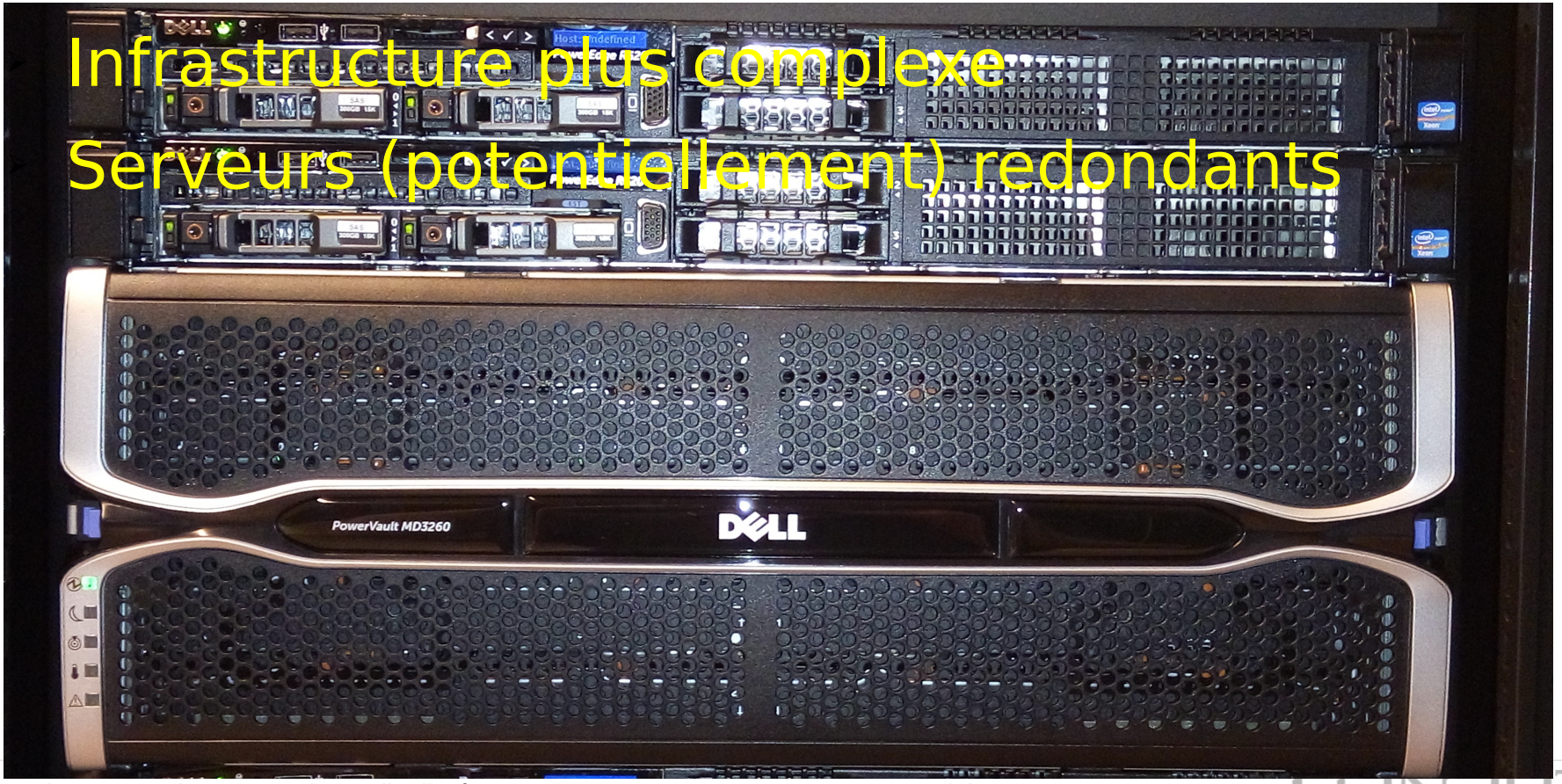
Matériel pour le stockage (distribué) : serveurs DAS

- Serveur avec des disques « attachés » *(DAS : Direct Attached Storage*), disques liés à un serveur (internes et/ou externes) :
  - Coût & simplicité
  - Très commun/dominant
  - « Intelligence » dans le serveur
  - Serveurs indépendants, serveur indisponible ⇒ données indisponibles *(SPOF, Single Point of Failure)*

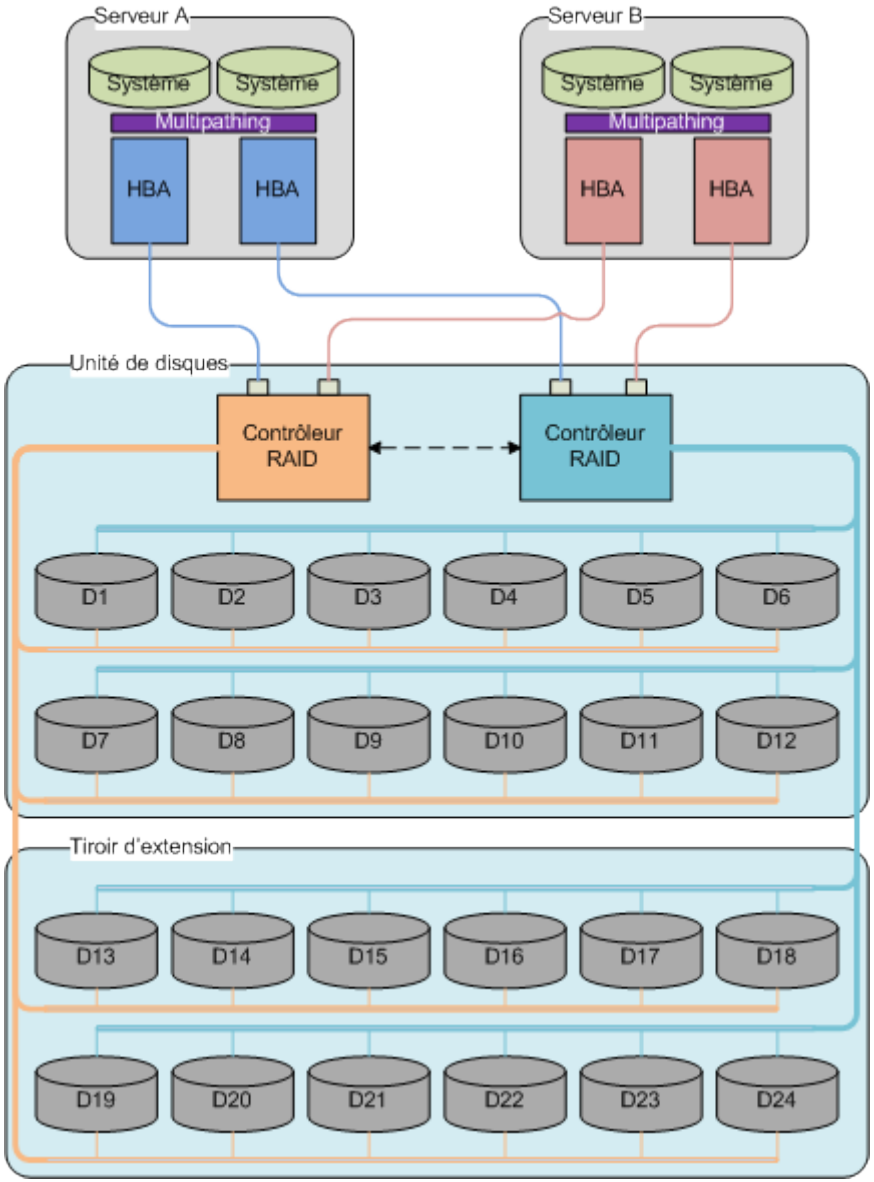


Matériel pour le stockage (distribué) : SAN

- SAN *(Storage Area Network)*, disques accessibles à plusieurs serveurs par l'intermédiaire d'un « réseau de stockage »
  - Unité de disques (ou « baie » de disques)
  - Coût
  - Infrastructure plus complexe
  - Serveurs (potentiellement) redondants



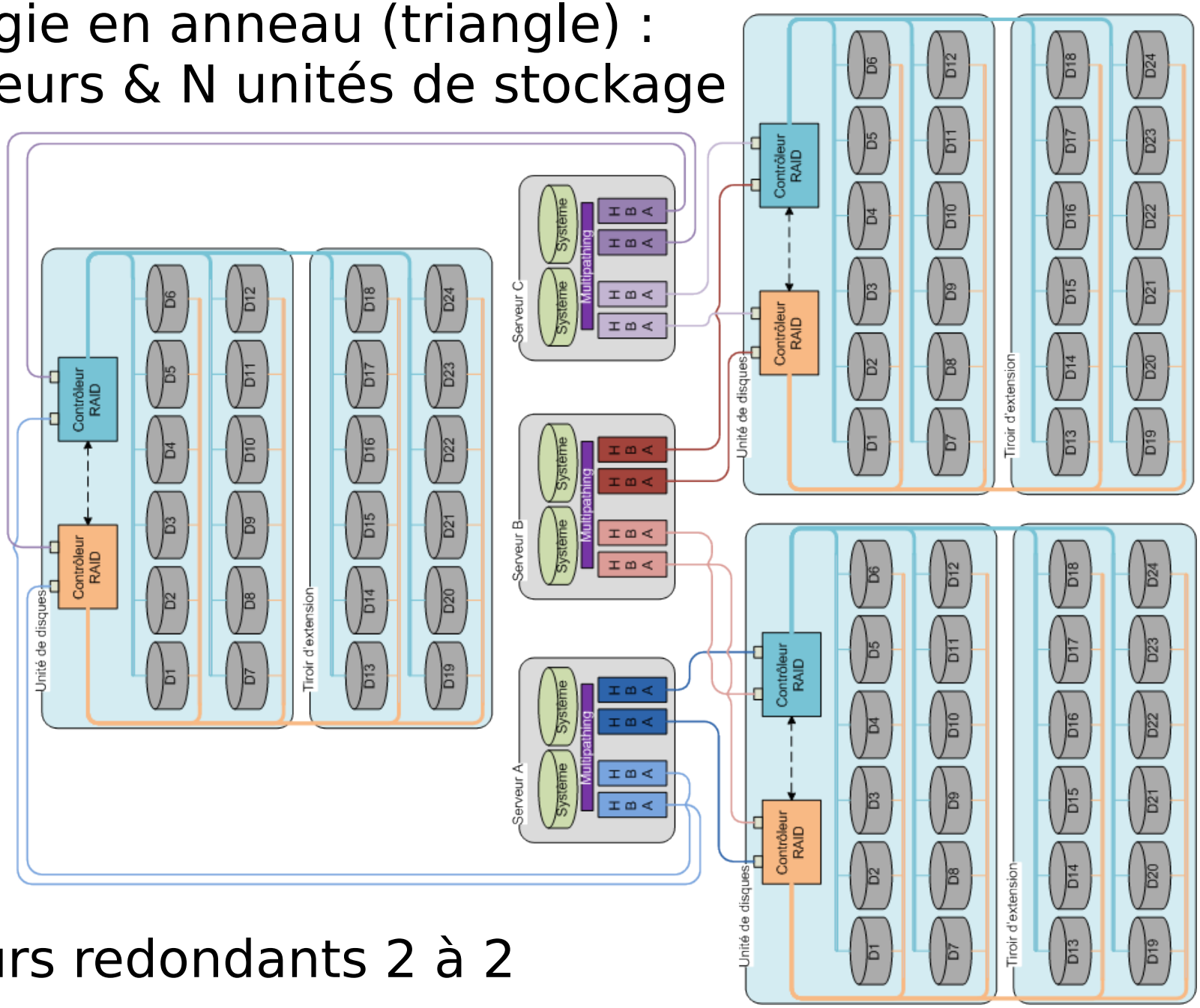
Matériel pour le stockage (distribué) : unité de disques



- Typiquement 4 à 8 ports (pour les données)
- Alimentation redondantes
- Contrôleurs RAID en *cluster* pour la redondance des données
- Gestion des accès des serveurs (y compris redondance des accès)
- Mémoire cache partagée (ou en miroir) entre les 2 contrôleurs
- Souvent mémoire cache avec batterie ou NVRAM

Matériel pour le stockage (distribué) : mini-SAN

- Topologie en anneau (triangle) :
  N serveurs & N unités de stockage

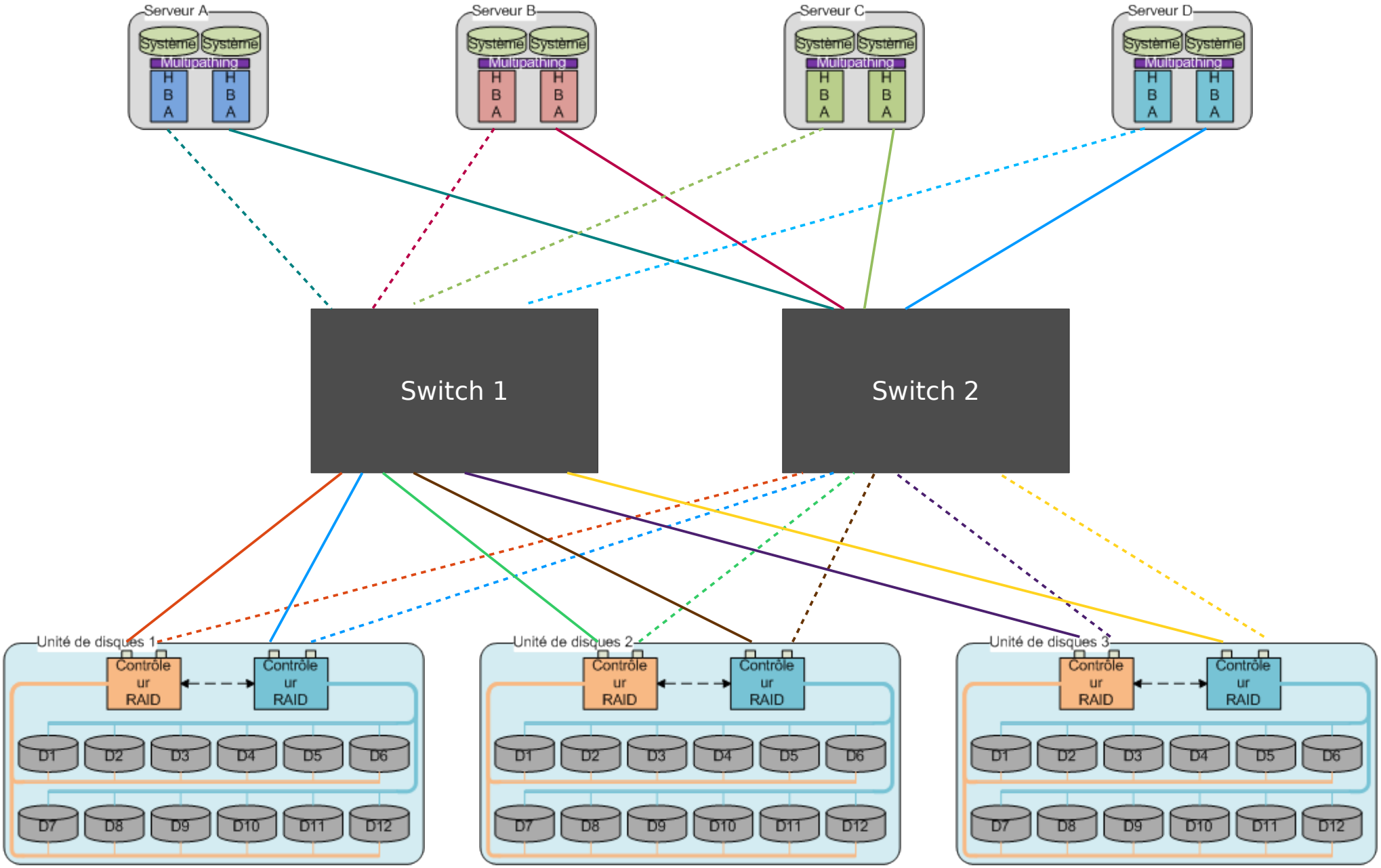


- Serveurs redondants 2 à 2

Matériel pour le stockage (distribué) : SAN (pour de vrai)

- SAN *(Storage Area Network)* : réseau dédié pour connecter des serveurs et des unités de stockage (disques, bandes, ...)
- Interconnexions (par coût croissant) :
  - Ethernet : principalement iSCSI, mutualisation & convergence (RoCE)
  - SAS : commun, faibles distances (10 m)
  - Fibre-Channel : historique, grandes distances (50 km), optique et cuivre (rare), topologies complexes possibles (switches & routage)
  - InfiniBand : besoin de performances (IB EDR 4x : 100 Gbits), réseau HPC mutualisé et/ou fonctionnalités spécifiques (RDMA), topologies complexes possibles (switches & routage)

Matériel pour le stockage (distribué) : réseau de stockage



RAID : principe

- RAID : *Redundant Array of Independent (Inexpensive) Drives (Disks)*
- Mécanismes de redondance : disque virtuel composé de plusieurs disques sur lesquels les données sont distribuées ⇨ interface de type bloc
- Différentes configurations (appelées niveaux), dont :
  - RAID-0 : pas de redondance, distribution des données
  - RAID-1 : réplication (miroir), très souvent à 2 voies (2 exemplaires des blocs de données)
  - RAID-5 & 6 : redondance (simple & double) par codage des blocs de données, redondance(s) distribuée(s)

RAID : niveaux composés & autre

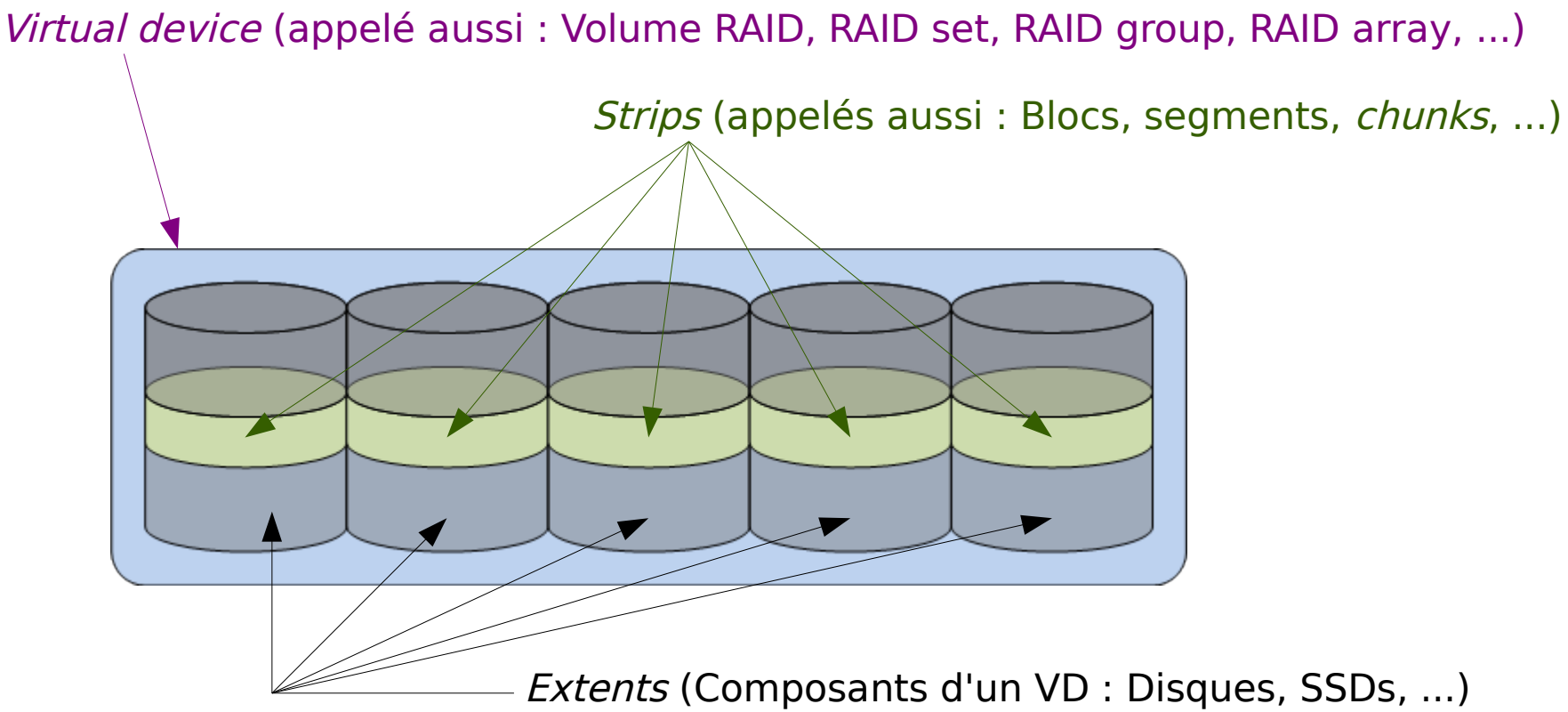
- Niveaux composés courants, en général niveau avec redondance combiné avec RAID-0 :
  - RAID-10 (miroir et distribution) :
    - RAID-1+0 : RAID-0(RAID-1) ☺
    - RAID-0+1 : RAID-1(RAID-0) ☹ (> 4 disques)
  - RAID-50 (redondance distribuée simple & distribution) : RAID-0(RAID-5)
  - RAID-60 (redondance distribuée double & distribution) : RAID-0(RAID-6)
- Avec RAID-6, possibilité de détection et correction d'erreurs sans informations corollaires (erreurs non détectées/signalées par les disques)

RAID : vocabulaire « standard » (SNIA)

- SNIA : *Storage Network Industry Association*
- Vocabulaire et niveaux définis de manière exhaustive :
  - *Virtual device* (volume RAID) : périphérique virtuel (de type bloc) avec des composants *(extents)*, eux aussi de type bloc et de taille **utile** identique
  - Données sur le volume RAID accédées en blocs de taille fixe (bande du volume RAID)
  - Bande du volume RAID découpée en *strips* (de taille fixe), chacun placé sur un *extent*
  - Pour les niveaux de RAID avec redondance, les informations de redondance sont aussi placées dans des *strips*

RAID : vocabulaire « standard » (SNIA)

- *Stripe* (bande) : groupe de N *strips* consécutifs avec N : nombre d'*extents*
- Largeur de bande *(stripe size)* : *N * strip size*



http://www.snia.org/tech_activities/standards/curr_standards/ddf

RAID : problèmes et limites (RAID-5 & RAID-6)

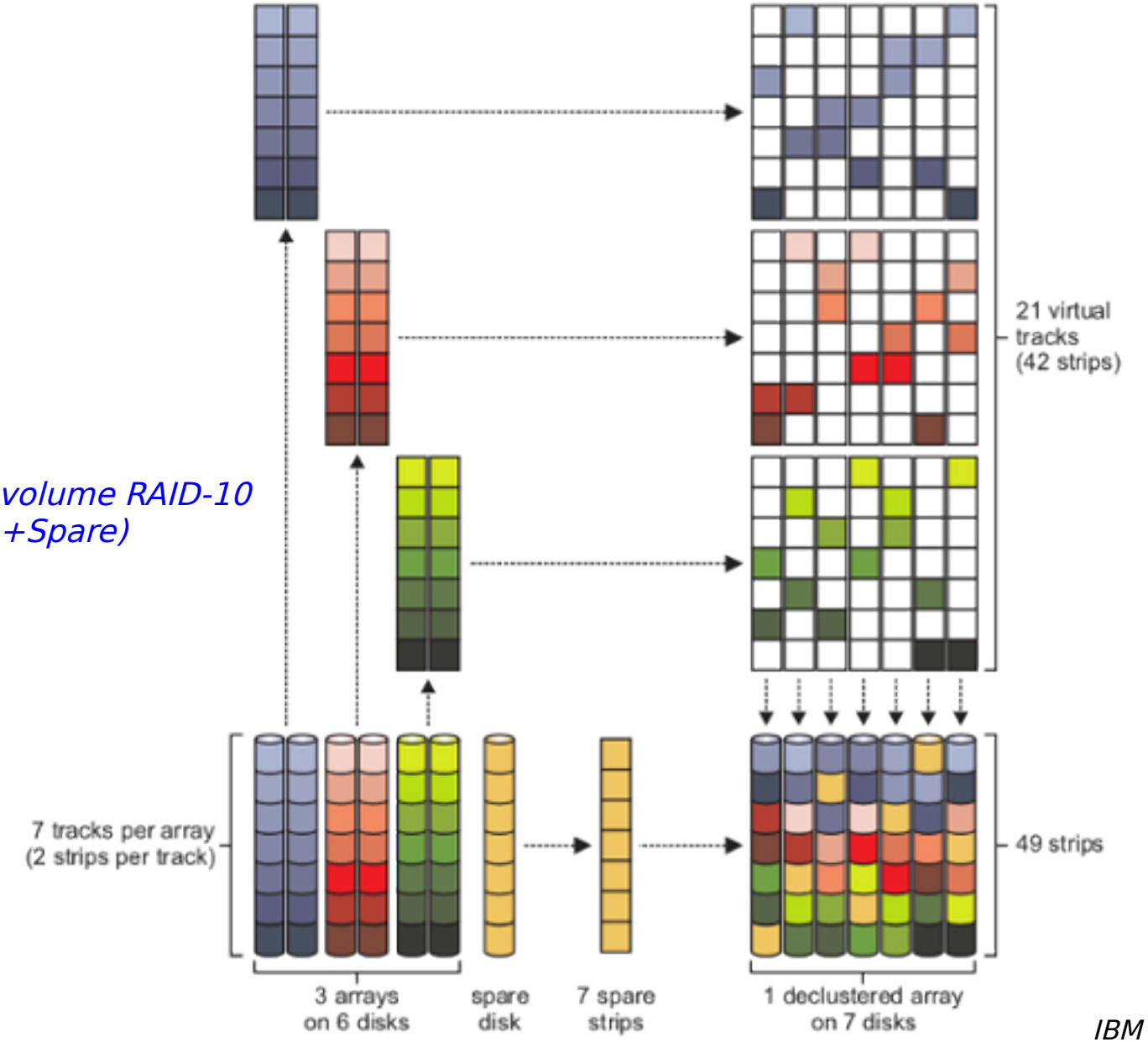
- Taille de bande fixe : écritures < taille de bande inefficaces (cycles *Lecture-Modification-Ecriture*) & *Write-hole/missed-write*
- Tailles de volumes RAID = f(tailles disques)
- Temps de reconstruction, fonction de :
  - Taille du volume RAID
  - Nombre de disques impliqués (de 1 à N-1 disques en lecture, 1 en écriture)
- Disque de secours *(hot-spare)* souvent nécessaire
- Reconstruction (ou « formatage ») systématique : tous les strips du « disque » en panne sont (re)construits même s'ils sont « vides »

RAID déstructuré : principe

- Formes non-standards de RAID « évolué »
- Exemples : DDP de LSI/NetApp ou GNR/SSNR d'IBM, ...
- Objectifs principaux :
  - Maximiser l'utilisation des disques (fiabilité, espace utile & performances)
  - Réduire les temps de reconstruction en cas de panne (tous les disques sont sollicités)
  - Seules les parties effectivement utilisées sont reconstruites
- Espace libre nécessaire à tout moment pour gérer une panne, configuré et exprimé en « équivalent disques de secours »

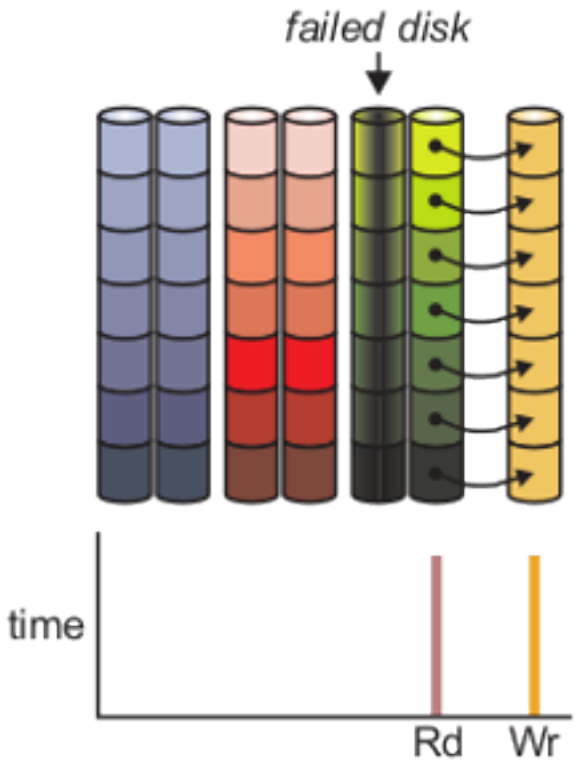
RAID déstructuré : exemple (1/2)

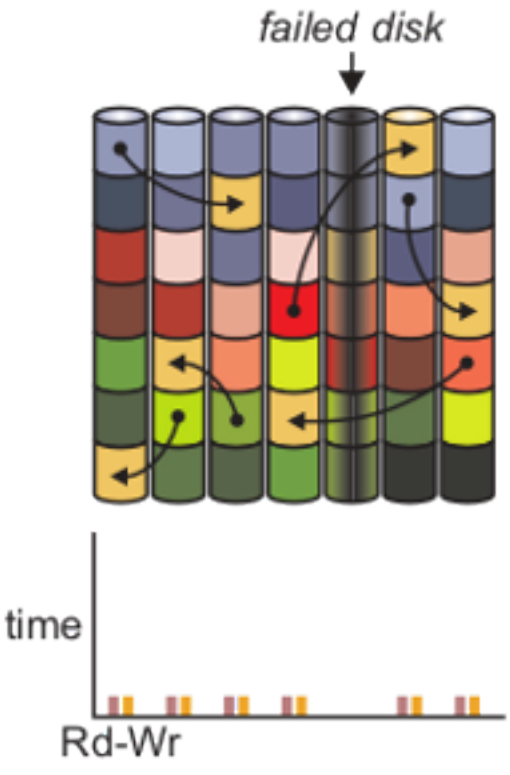
Exemple avec un volume RAID-10 (3xRAID-1+Spare)



IBM

RAID déstructuré : exemple (2/2)



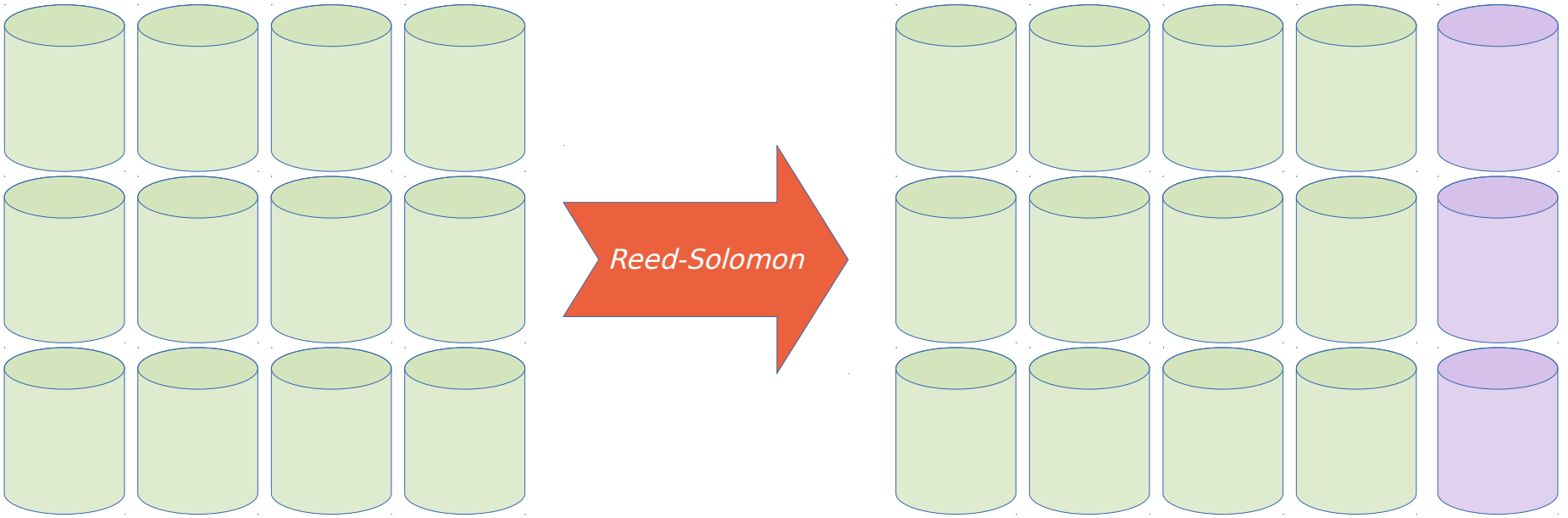


IBM

Codes correcteurs à effacement *(Erasure Coding)* : principe

- Objectif : stocker ou transférer les données de manière redondante en limitant la quantité totale de données à stocker ou à transférer
- En stockage de données, les codes correcteurs de type Reed-Solomon sont très largement utilisés
- *Spoiler* : les niveaux RAID-{3,4,5,6} sont des cas « simples » de codage Reed-Solomon
- Données découpées en morceau de taille fixe, informations de redondance stockées soit :
  - En **combinant** les données et les informations de redondance (codes *non-systématiques*)
  - Dans des **blocs dédiés** de redondance (codes dits *systématiques*)

- On désigne en général un code Reed-Solomon particulier sous la forme RS(n, k)



k blocs de données

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

Codes correcteurs à effacement : Reed-Solomon (exemple trivial 1/3)

- On veut transférer (ou stocker) 3 blocs : 12 34 56
- On peut calculer un bloc de contrôle (« parité ») :
  - C1 = 12 + 34 + 56 = 102
- On transfère les 4 blocs, on reçoit : 12 **32** 56 102
  - C1' = 12 + 32 + 56 = 100 ≠ 102

    ⇨ **Données transférées corrompues**
- Avec un seul bloc de contrôle :
  - Lors d'un transfert, on peut seulement détecter qu'une erreur a eu lieu (on ne sait pas où est l'erreur)
  - Pour du stockage, on peut avoir des informations corollaires qui localisent l'erreur

Codes correcteurs à effacement : Reed-Solomon (exemple trivial 2/3)

- On veut stocker les 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
- Plus tard, on relit tous les blocs (de données & de contrôle), on obtient : 87 65 **51** 21 216 430
  - C1' = 87 + 65 + 51 + 21 = 224 ≠ 216
  - C2' = 87*1 + 65*2 + 51*3 + 21*4 = 454 ≠ 430
- Erreur : C1 - C1' = -8
- Position de l'erreur :
  - (C2 - C2') / Erreur = (430 - 454) / (-8) = 3

Codes correcteurs à effacement : Reed-Solomon (exemple trivial 3/3)

- Le bloc endommagé est le bloc 3 (43 → 51), qu'on peut retrouver avec :
  - $Bloc3_{Corrigé} = Bloc3_{Transféré} + Erreur$
- Deux blocs de contrôle permettent de trouver la position de l'erreur puis de la corriger

- Les vrais codes correcteurs (y compris RAID) :
  - Reposent sur l'arithmétique modulaire dans les corps de Galois (en général $GF(2^8)$) et pas sur l'arithmétique dans $\mathbb{Z}$ (tailles des blocs)
  - Utilisent souvent des matrices initialisées en fonction des paramètres spécifiques du code utilisé

Codes correcteurs à effacement : Reed-Solomon

- Une caractéristique importante voire essentielle des « bons » codes Reed-Solomon est appelée MDS *(Maximum-Distance-Separable)* : n'importe quel bloc défectueux ou manquant peut être recréé, dans la limite des *m* blocs
- En stockage de données : RS(12, 8) et RS(12, 9) sont courants
- En pratique, du codage multiple (entrelacement et code pyramidaux) est souvent utilisé, à la fois pour augmenter la redondance et réduire les temps de reconstruction

Codes correcteurs à effacement : il n'y a pas que Reed-Solomon

- De nombreux autres types de codes à effacement que Reed-Solomon existent
- Souvent ces codes sont basés sur d'autres outils mathématiques que les corps de Galois (transformées de Fourier ou de Radon, ...)
- Certains codes sont très utilisés pour les télécommunications, par exemple : Turbo-Codes & RAPTOR
- Certains codes ont des caractéristiques spécifiques intéressantes, par exemple : Mojette, code non systématique efficace qui « garantit » l'intégrité des données lues et minimise les écritures lors des modifications (base de RozoFS)

Codes correcteurs à effacement : en bref

- Les codes correcteurs sont souvent utilisés dans les systèmes de stockage « modernes » pour assurer la redondance des données peu ou pas modifiées *(sealed blocks)*
- Par rapport au RAID :
  - Les paramètres du code utilisé peuvent être décidés par l'application (au lieu du système) : variables en fonction de l'importance des données (exemples : méta-données ou index répliqués N fois, données surtout **lues** encodées avec RS) ou en fonction du besoin (tailles de blocs non fixes)
  - Davantage de données manquantes ou endommagées peuvent être corrigées

Codes correcteurs à effacement : en conclusion

- Par rapport à la réplication à 2 ou 3 voies :
  - Le stockage est beaucoup plus économique en espace disque nécessaire
- Choix du mode de redondance basé sur la relation entre :
  - Capacité disponible (brute)
  - Capacité utile (redondance prise en compte)
  - Capacité de reconstruction (nombre de pannes)
  - Temps de reconstruction en cas de panne
  - Temps de mise à jour en cas de modification
  - ⇨ **Coût**

Réplication : principe (1/2)

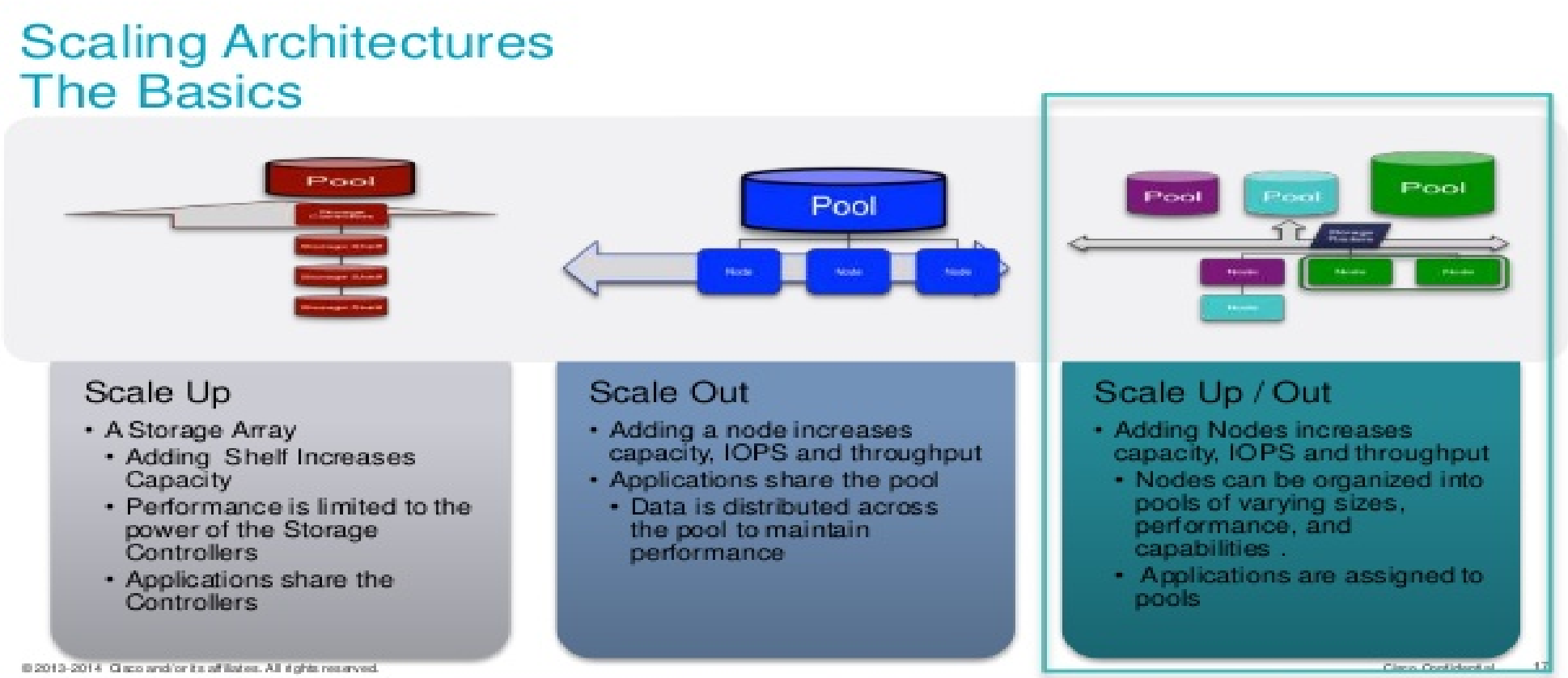
- Mode de redondance en apparence le plus simple
- Difficulté majeure, cohérence des réplicats :
  - Reprise après erreur
  - Cohérence à l'instant t
- Différentes méthodes répandues, dont :
  - Autorité de référence : journal centralisé ou unique, horodatage *(timestamp)*, numéro de version/génération
  - Majorité : les réplicats comparent leur points de référence, la version valide est celle qui est la plus souvent présente (pas nécessairement la dernière)

Réplication : principe (2/2)

- Minorité : les réplicats comparent leur points de référence, la version valide est la dernière (pas nécessairement la plus présente)
- Hybride : un des réplicats (majoritaires) est désigné ou choisi au hasard pour être l'autorité de référence

- Certains systèmes de stockage distribué basés sur la réplication, utilisent un protocole « à majorité » pour chaque lecture et/ou pour valider/acquiter les écritures
- Suivant les systèmes de stockage distribué, la réplication est assurée par les serveurs (souvent en tâche de fond) ou par les clients (envois multiples)

- Brique de base : approche courante pour le stockage (distribué) : standardiser une granularité type de matériel et réutiliser cette configuration pour croître à la fois en performance & capacité

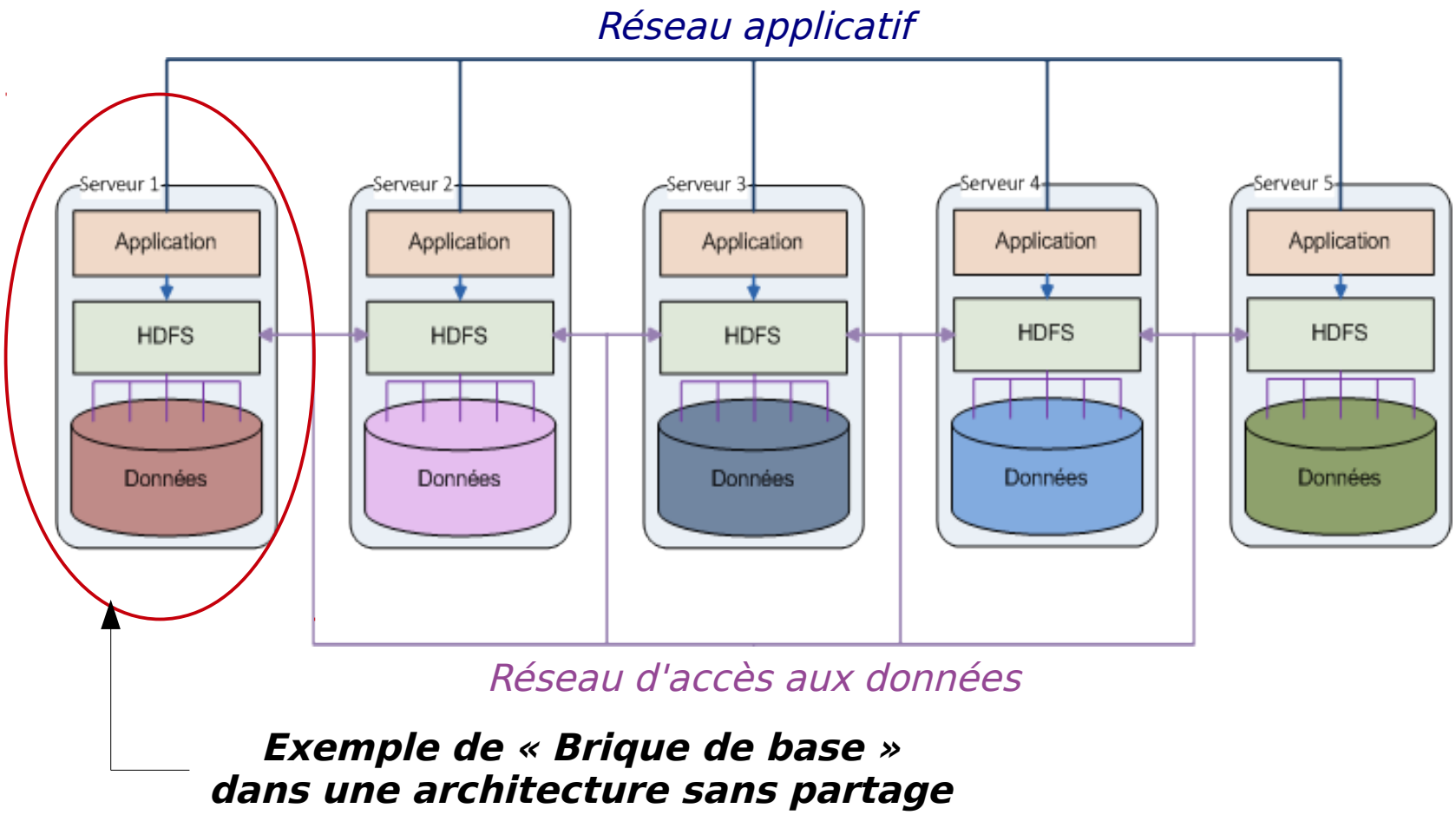


Cisco

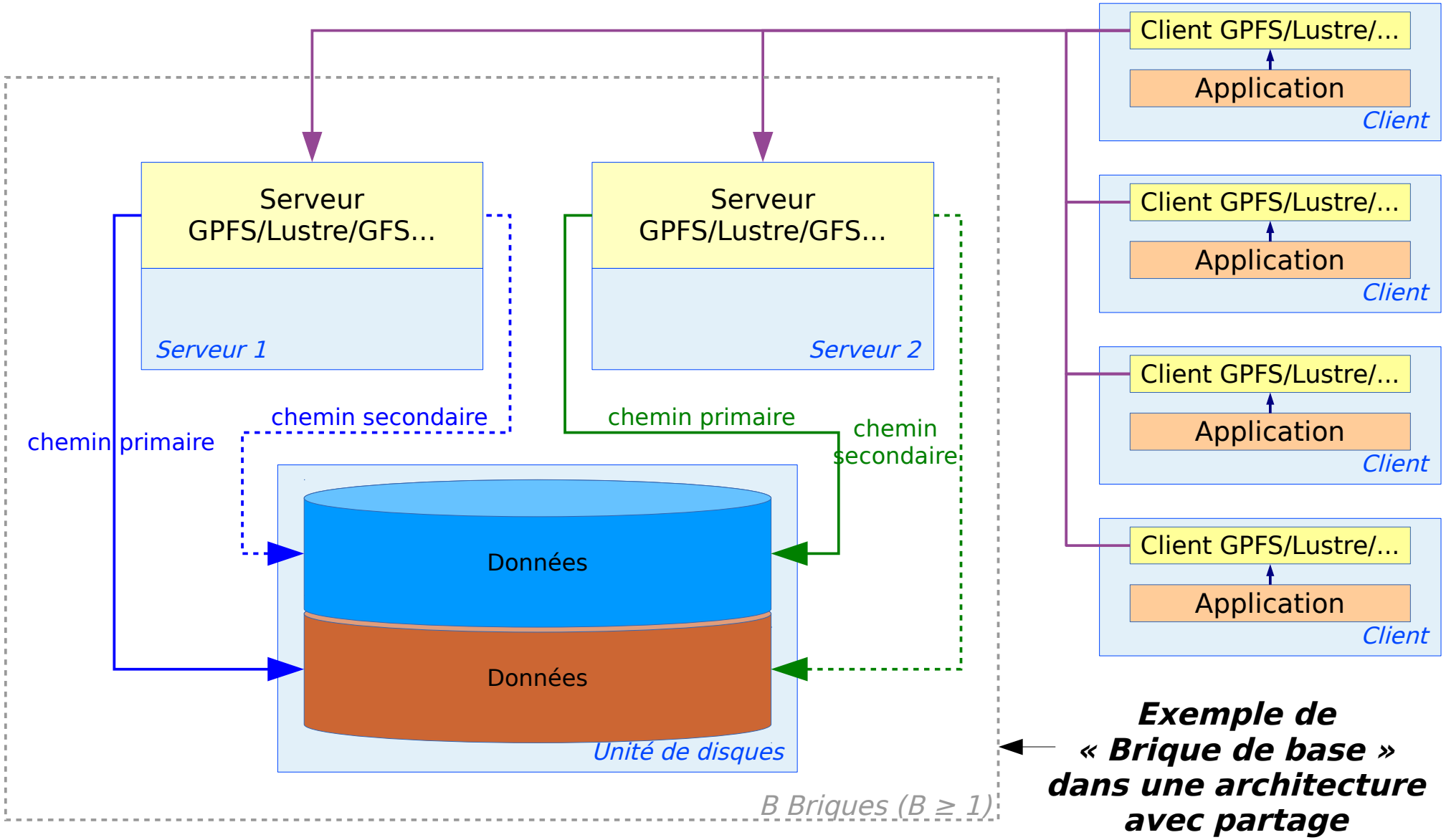
- Deux principales « classes » d'architectures pour le stockage distribué :
  - Sans partage *(Shared Nothing)* : serveurs indépendants sans ressources matérielles (de stockage) partagées ⇨ modèle de base du stockage « moderne », architectures *scale-out*
  - Avec partage : ressources matérielles de stockage partagées par tout ou partie des machines impliquées ⇨ SAN ou mini-SAN(s)

    En pratique, les ressources partagées sont souvent limitées aux serveurs de données (& méta-données) à proprement parler, les clients utilisent alors le réseau généraliste

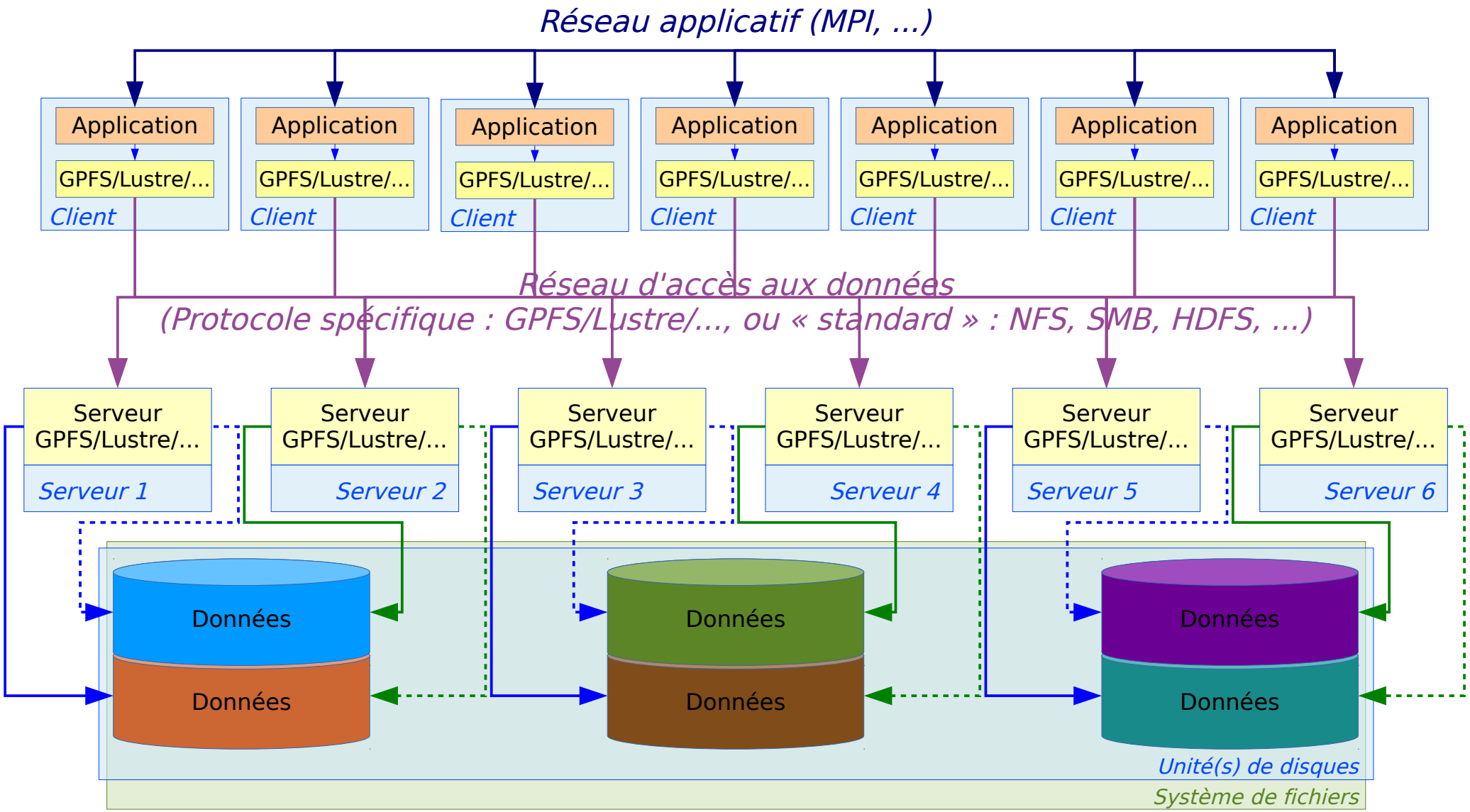
- Exemple de « convergence » (ici avec HDFS) : stockage et applications mutualisés (sur les mêmes serveurs), sans disques partagés



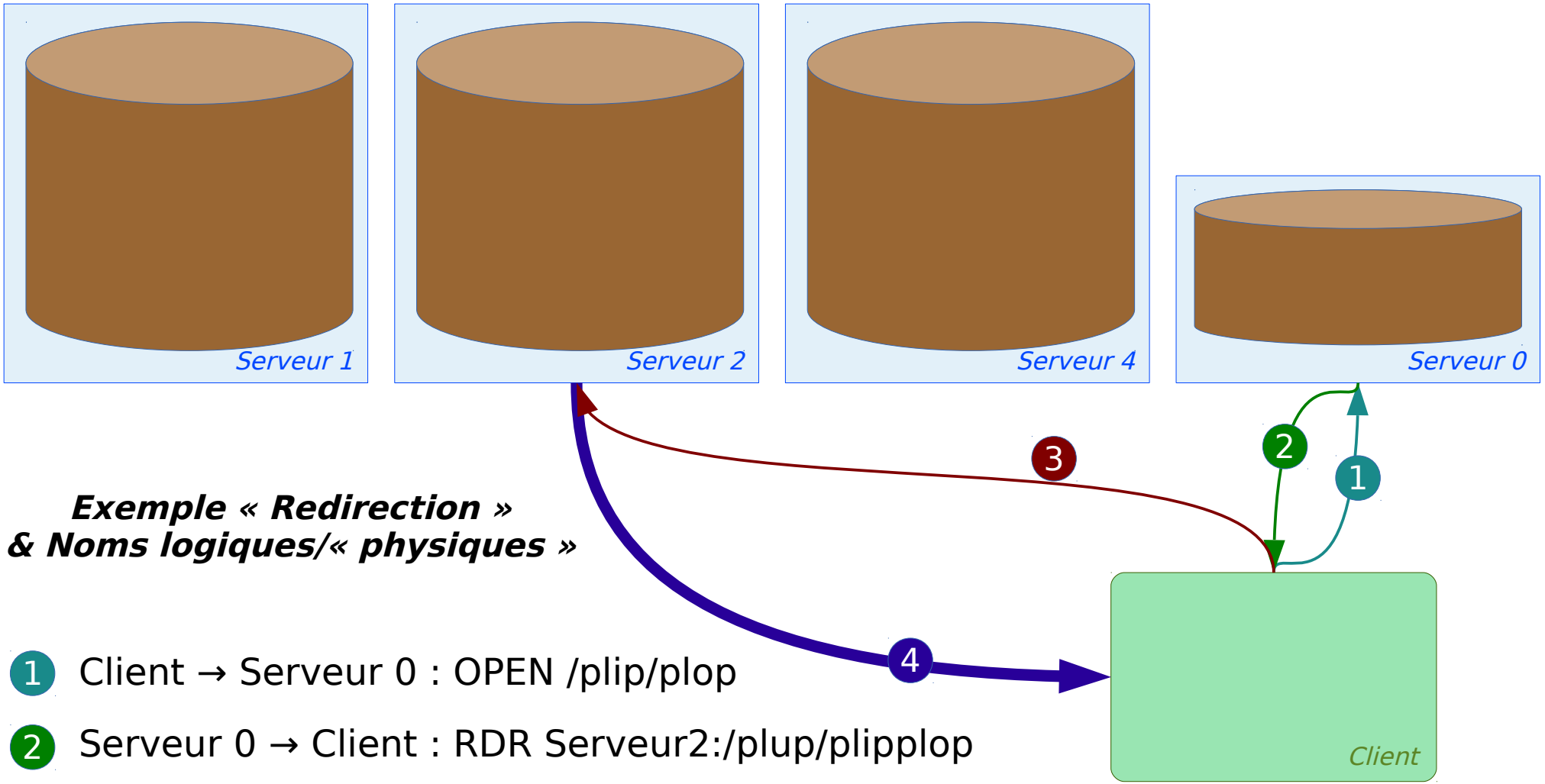
- Exemple avec disques partagés et réseaux distincts application, données (stockage)



- Exemple avec disques partagés et réseaux distincts application, données (stockage)



Infrastructures de stockage distribué : redirections & noms logiques



Exemple « Redirection »
& Noms logiques/« physiques »

1. Client → Serveur 0 : OPEN /plip/plop
2. Serveur 0 → Client : RDR Serveur2:/plup/plipplop
3. Client → Serveur 2 : READ /plup/plipplop
4. Serveur 2 ⇨ Client : DATA /plup/plipplop

"/plip/plop" : Nom logique
"/plup/plipplop" : Nom « physique »
Noms physiques & logiques pas nécessairement identiques ou similaires

- Les « briques de base » fonctionnent aussi bien pour les architectures avec partage (mini-SAN) que pour les architectures sans partage (DAS)
- Ces deux type d'architectures ont chacune des avantages et des inconvénients
- Sans partage :
  - Coût (au serveur) en général plus faible
  - Croissance de type *scale-out*
  - Type d'architecture préféré par beaucoup de systèmes de stockage « modernes »
  - Redondance, le plus souvent, par réplication (coût pour des grandes capacités et serveurs redondants dans les limites de la réplication)

- Avec partage :
  - Coût (à l'espace utile) souvent plus faible
  - Croissance de type scale-out avec briques de base
  - Type d'architecture souvent préféré pour les systèmes de fichiers parallèles
  - Infrastructure plus complexe/coûteuse sans briques de base
- Identités des utilisateurs
- Espace de nommage

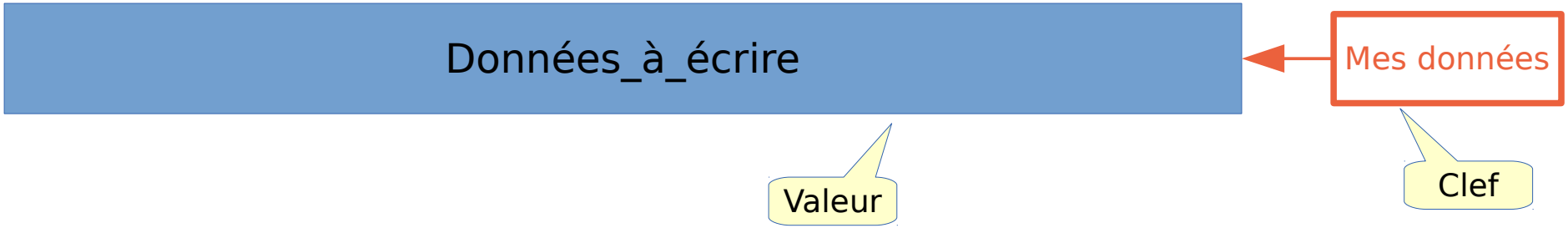
Données distribuées : bases

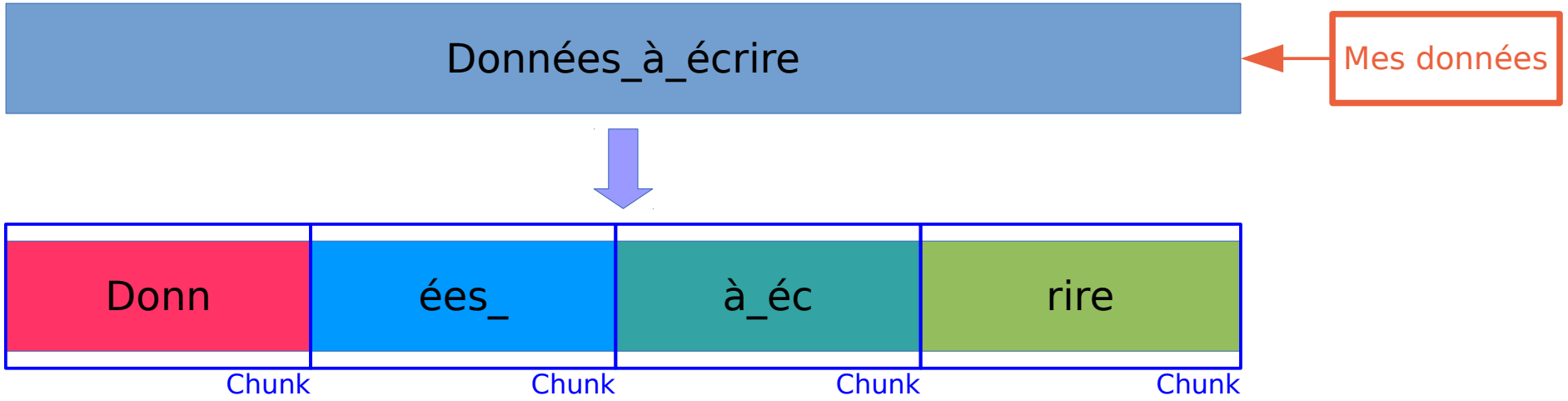
- Nombre d'architectures de stockage distribué « modernes » fonctionnent avec des « briques de base » ⇨ données distribuées sur les serveurs de stockage avec une granularité variable (dont les nom & type dépendent du système de stockage : *chunk*, *shard*, bloc, *metablock*, objet, ...)
- Une solution commune est de distribuer les données en fonction de leur valeur/contenu (ou position) à l'aide d'une fonction de hachage pour éviter un ou quelques « points chauds »
- Ce type de solution repose souvent sur des tables de hachage distribuées *(DHT : Distributed-Hash-Table)*

Données distribuées : *DHT & Network RAID/RAIN*

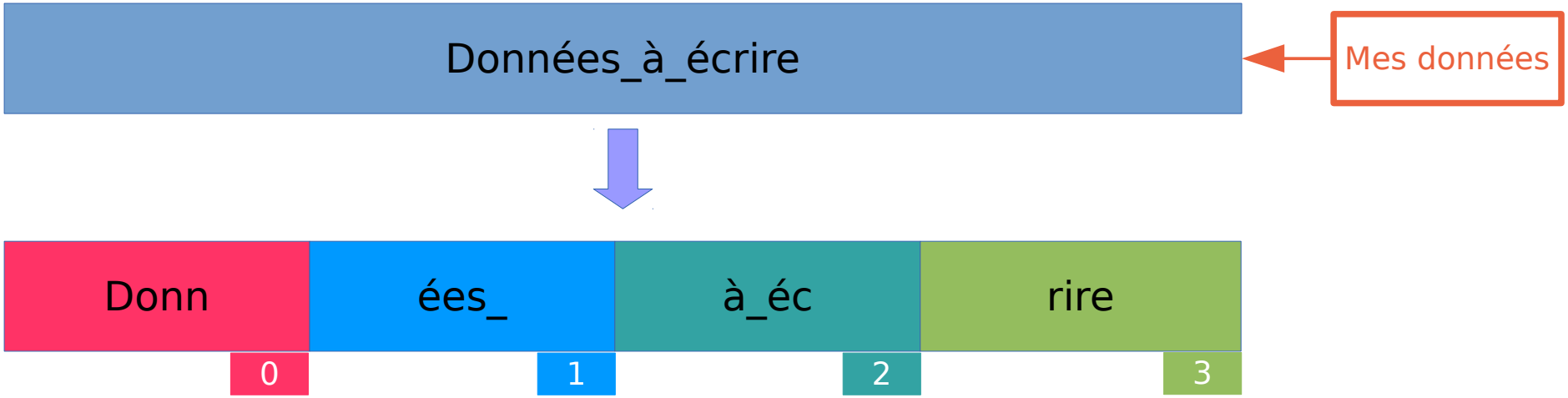
- Les DHT sont à la base des réseaux *peer-to-peer,* des anneaux de stockage et très souvent utilisées dans les systèmes de fichiers distribués
- L'exemple qui suit utilise la réplication, on peut aussi utiliser un code à effacement (équivalent au RAID ou avec plus de redondance) pour assurer la disponibilité des données entre les serveurs, on parle alors de *Network RAID* ou de *RAIN (Redundant Array of Independent Nodes)*

*Exemple simplifié de système clef/valeur **non** basé sur un système particulier*

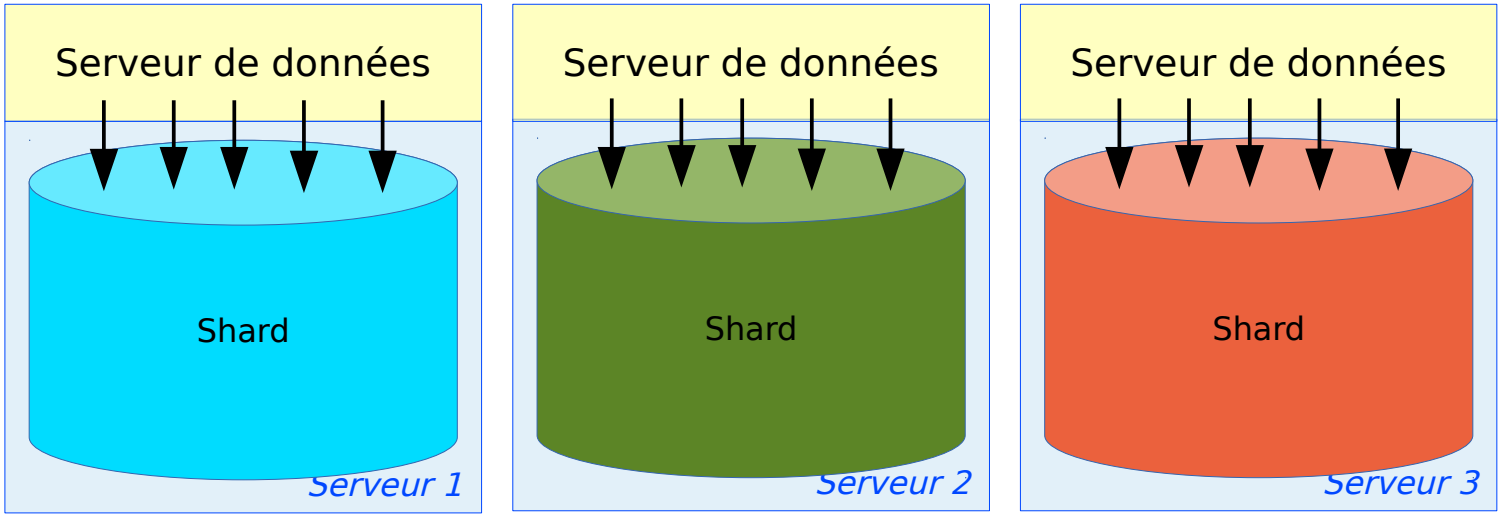

Données_à_écrire
Mes données
Valeur
Clef



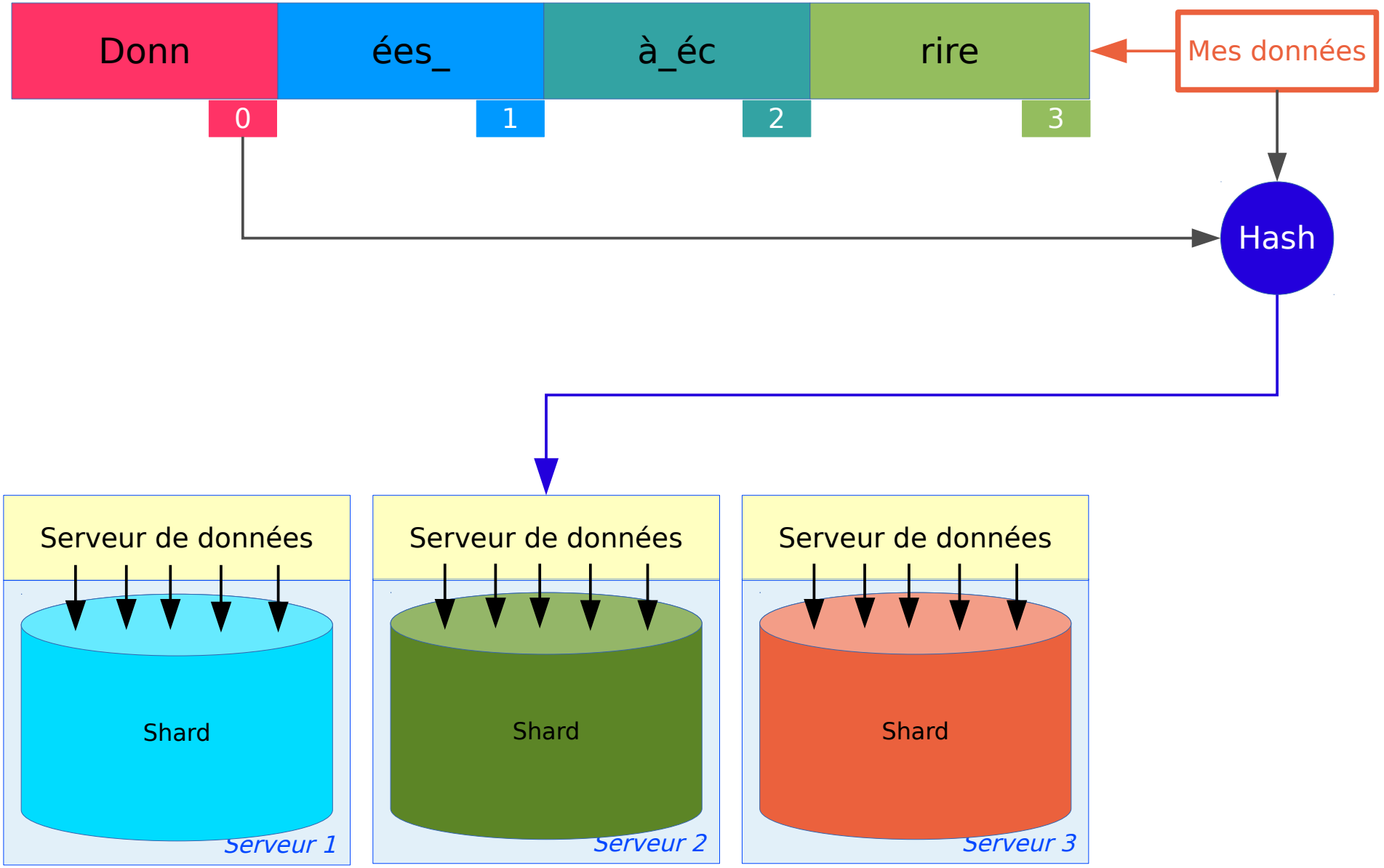
Les données sont découpées en morceaux, ici appelés « chunks »

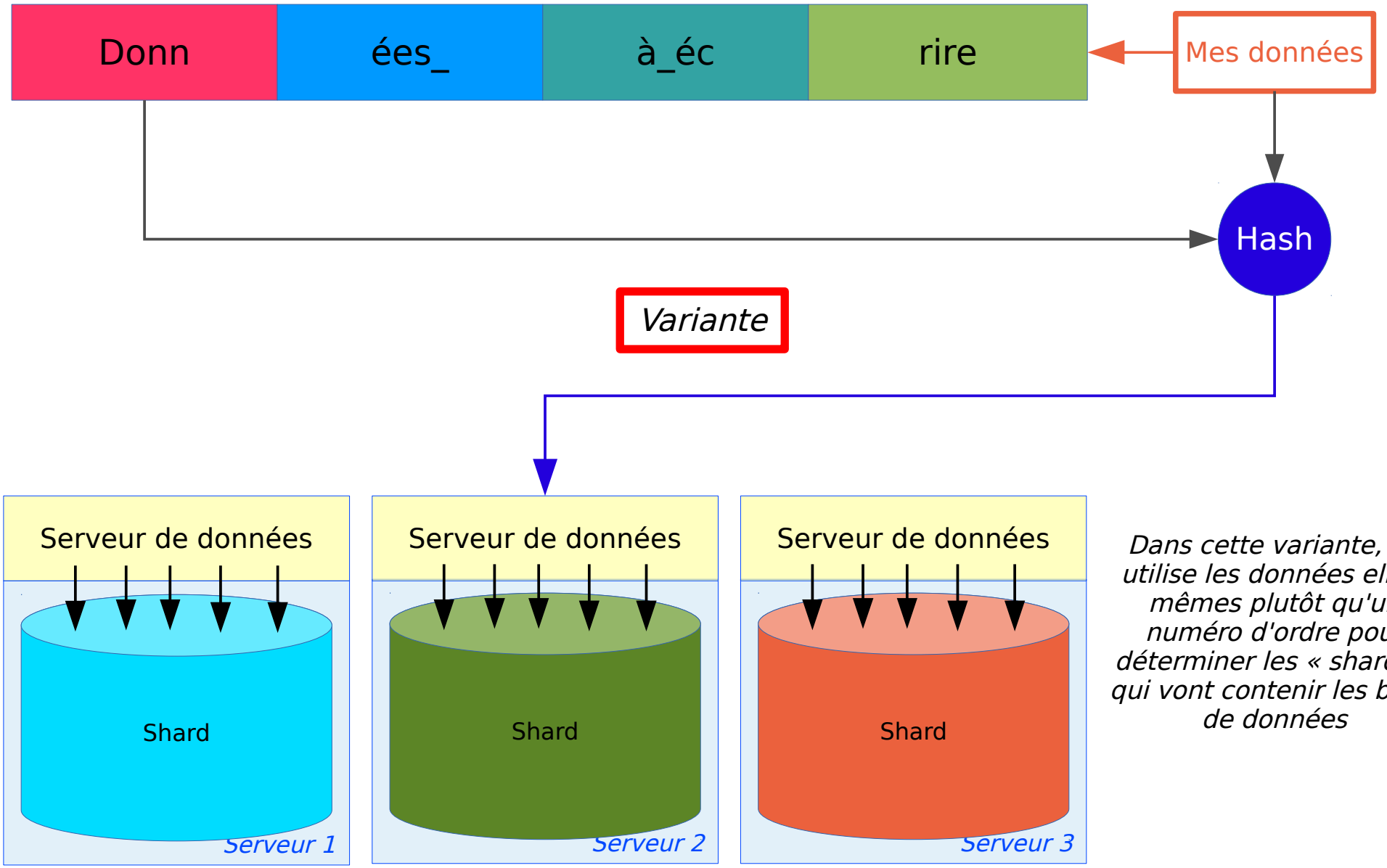


Dans cet exemple, on associe un numéro d'ordre à chaque morceau de données (« chunk »)

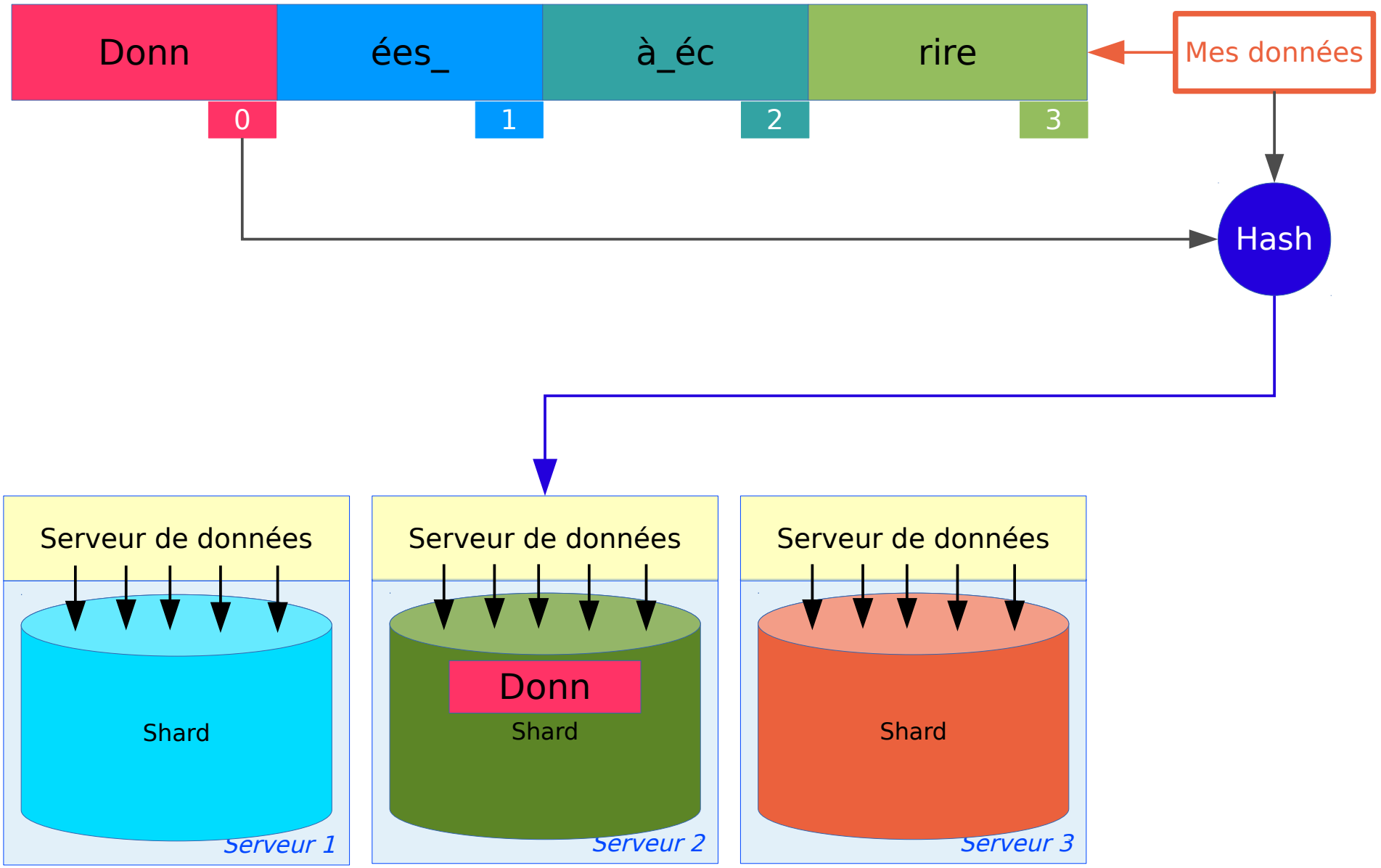


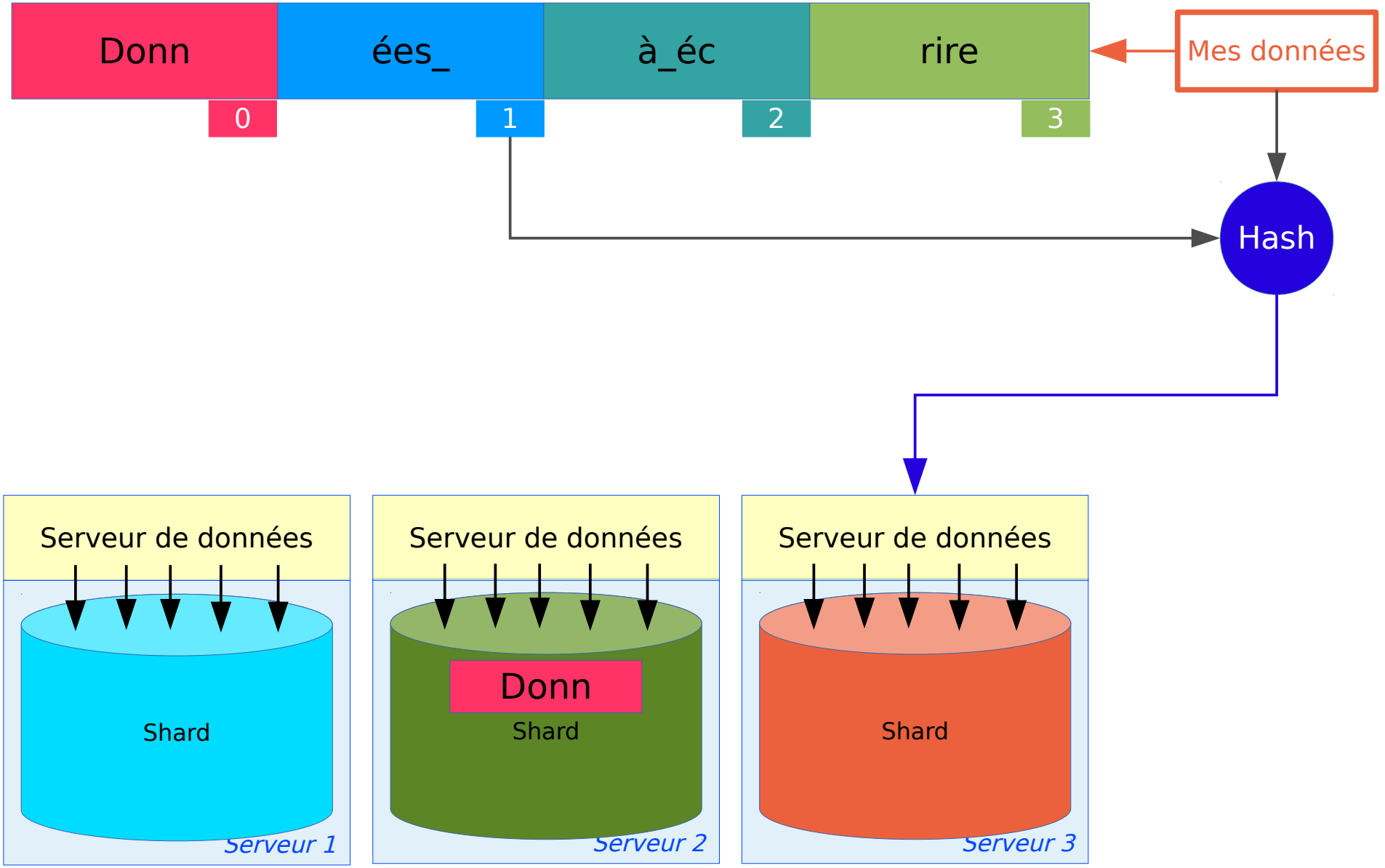
Dans cet exemple, les serveurs fournissent chacun une « portion » de stockage ici appelée « shard », dans certains systèmes on peut avoir plusieurs « portions » (bucket, shard, OST, OSD, NSD, slice, ...) par serveur

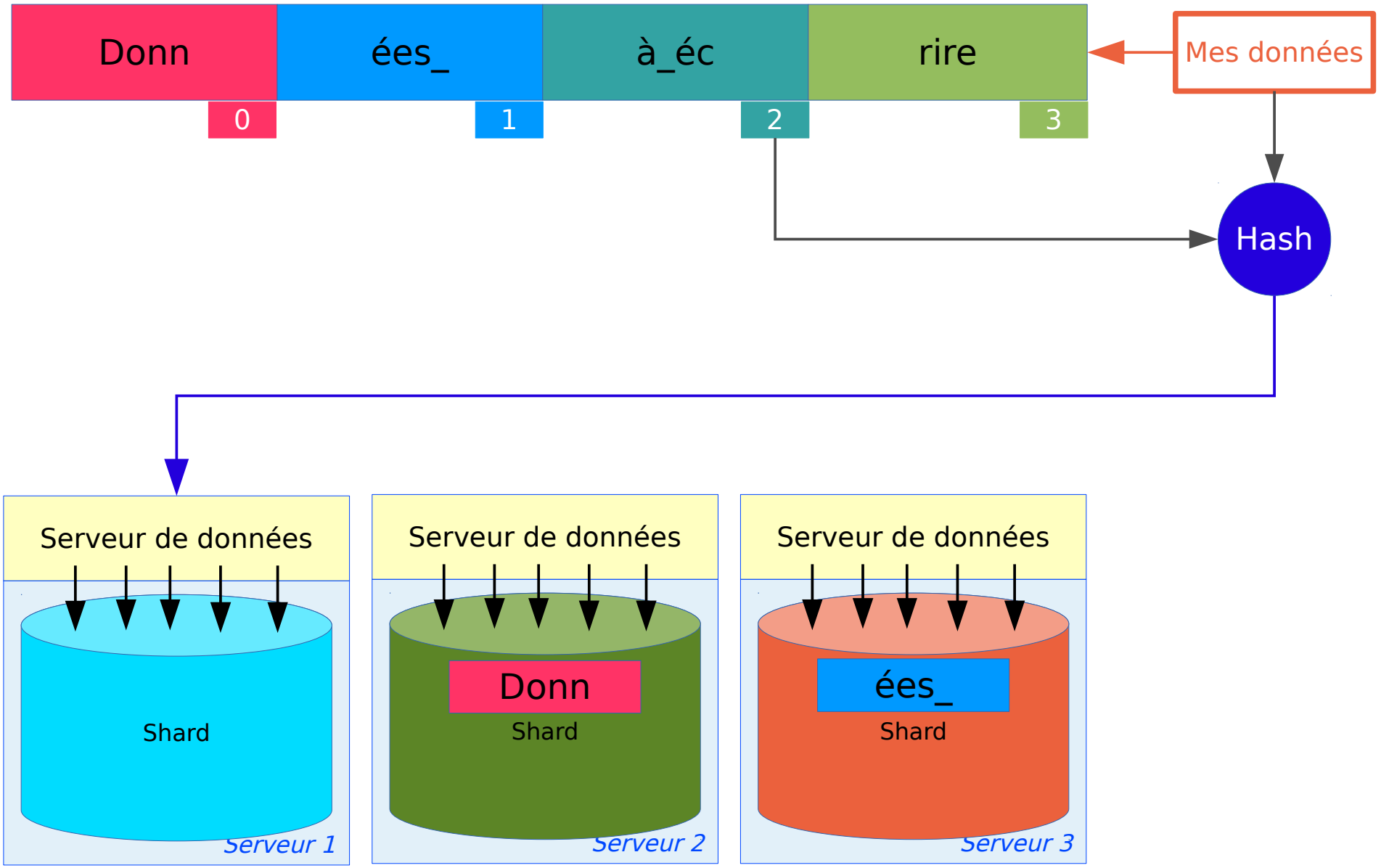

Donn
ées_
à_éc
rire
0
1
2
3
Mes données
Hash
Serveur de données
Shard
Serveur 1
Serveur de données
Shard
Serveur 2
Serveur de données
Shard
Serveur 3

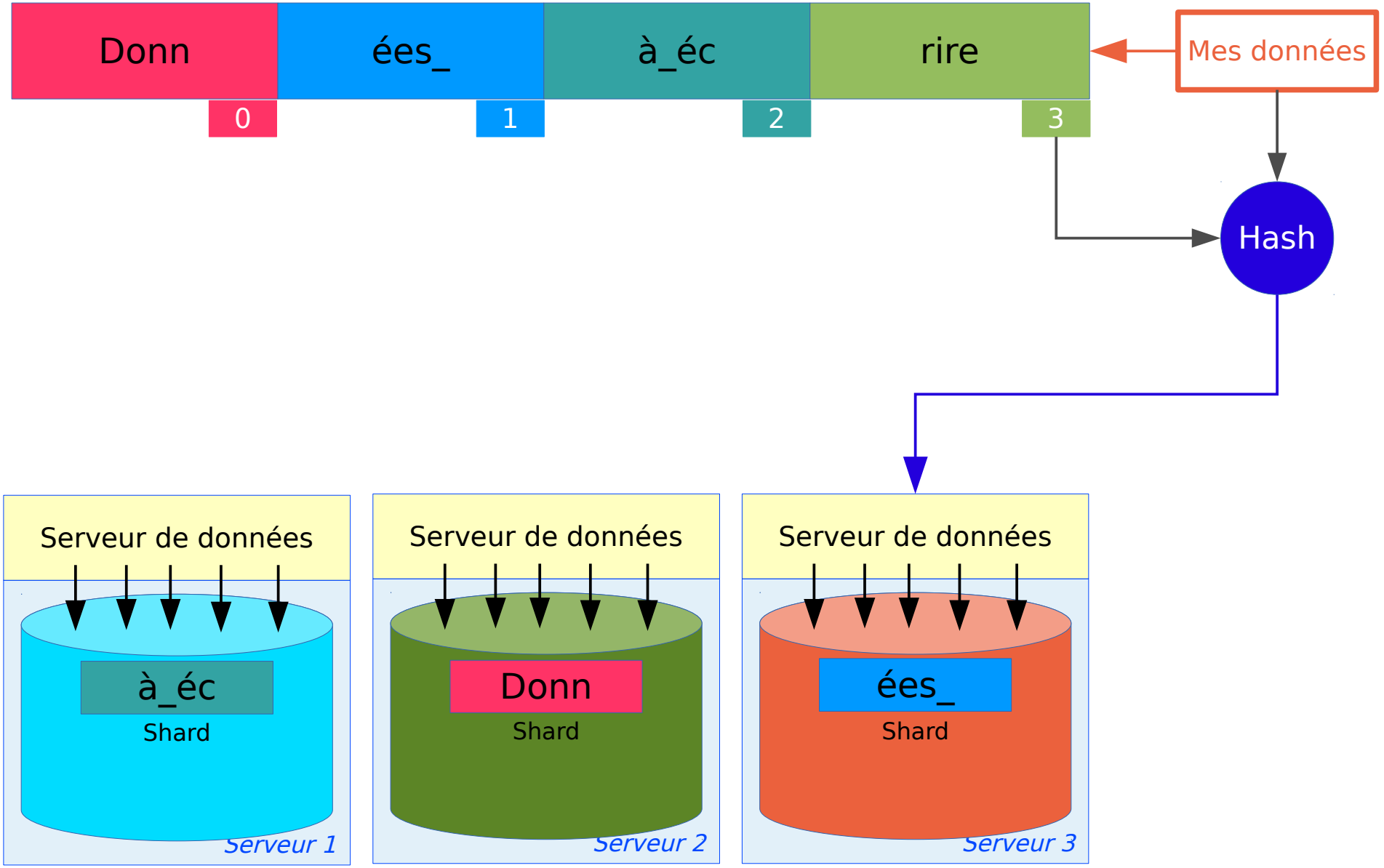


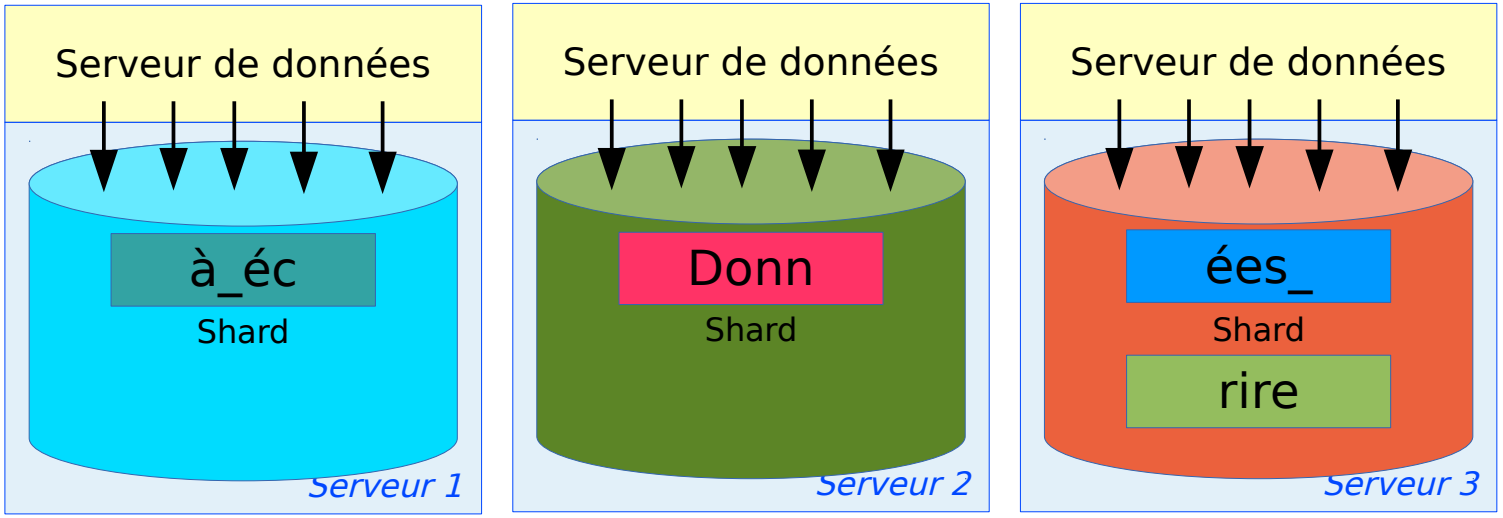
Dans cette variante, on utilise les données elles-mêmes plutôt qu'un numéro d'ordre pour déterminer les « shards » qui vont contenir les blocs de données


Donn
ées_
à_éc
rire
0
1
2
3
Mes données
Hash
Serveur de données
Shard
Serveur 1
Serveur de données
Donn
Shard
Serveur 2
Serveur de données
Shard
Serveur 3

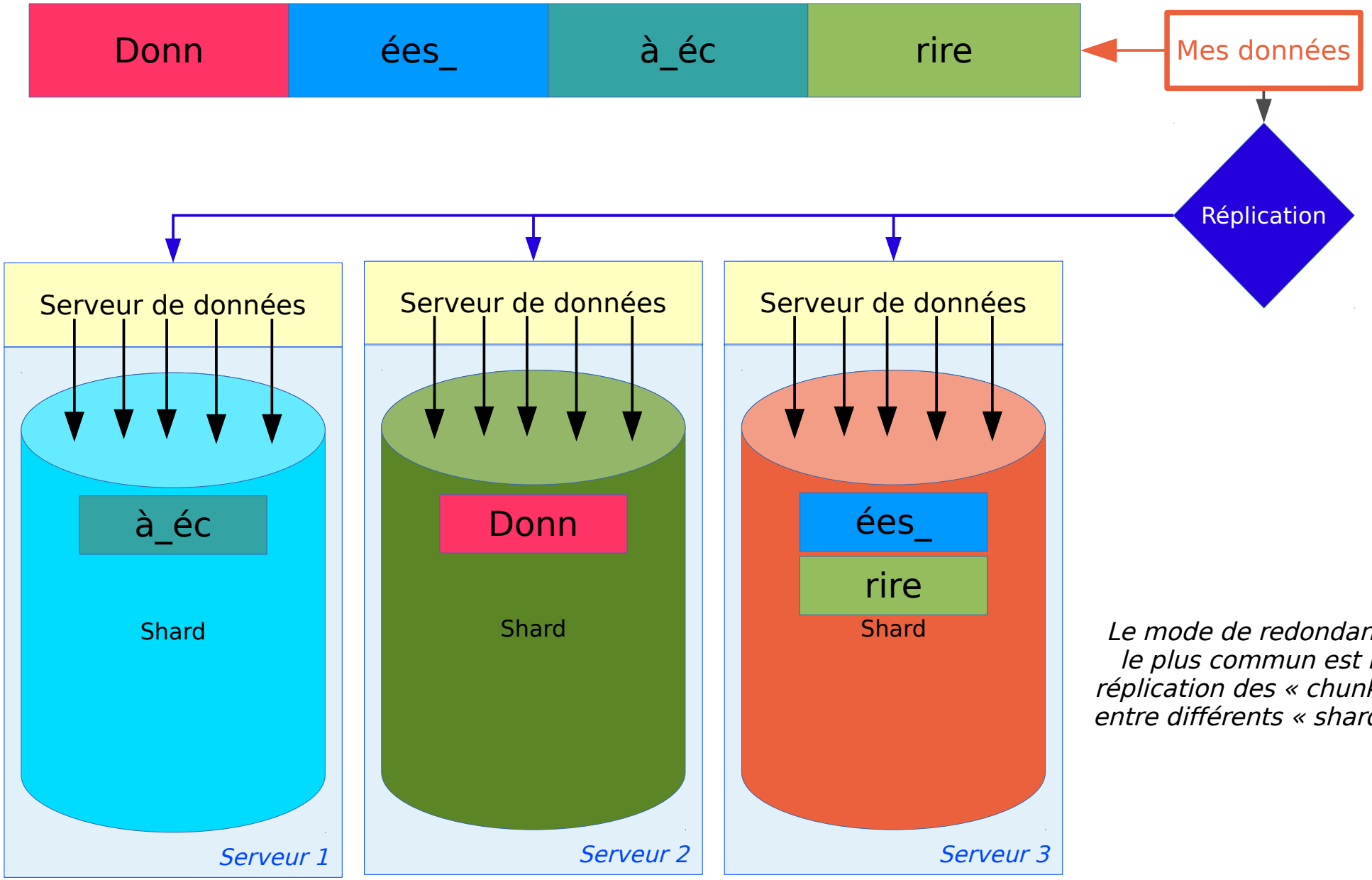

Donn
ées_
à_éc
rire
0
1
2
3
Mes données
Hash
Serveur de données
Shard
Serveur 1
Serveur de données
Donn
Shard
Serveur 2
Serveur de données
Shard
Serveur 3


Donn
ées_
à_éc
rire
0
1
2
3
Mes données
Hash
Serveur de données
Shard
Serveur 1
Serveur de données
Donn
Shard
Serveur 2
Serveur de données
ées_
Shard
Serveur 3

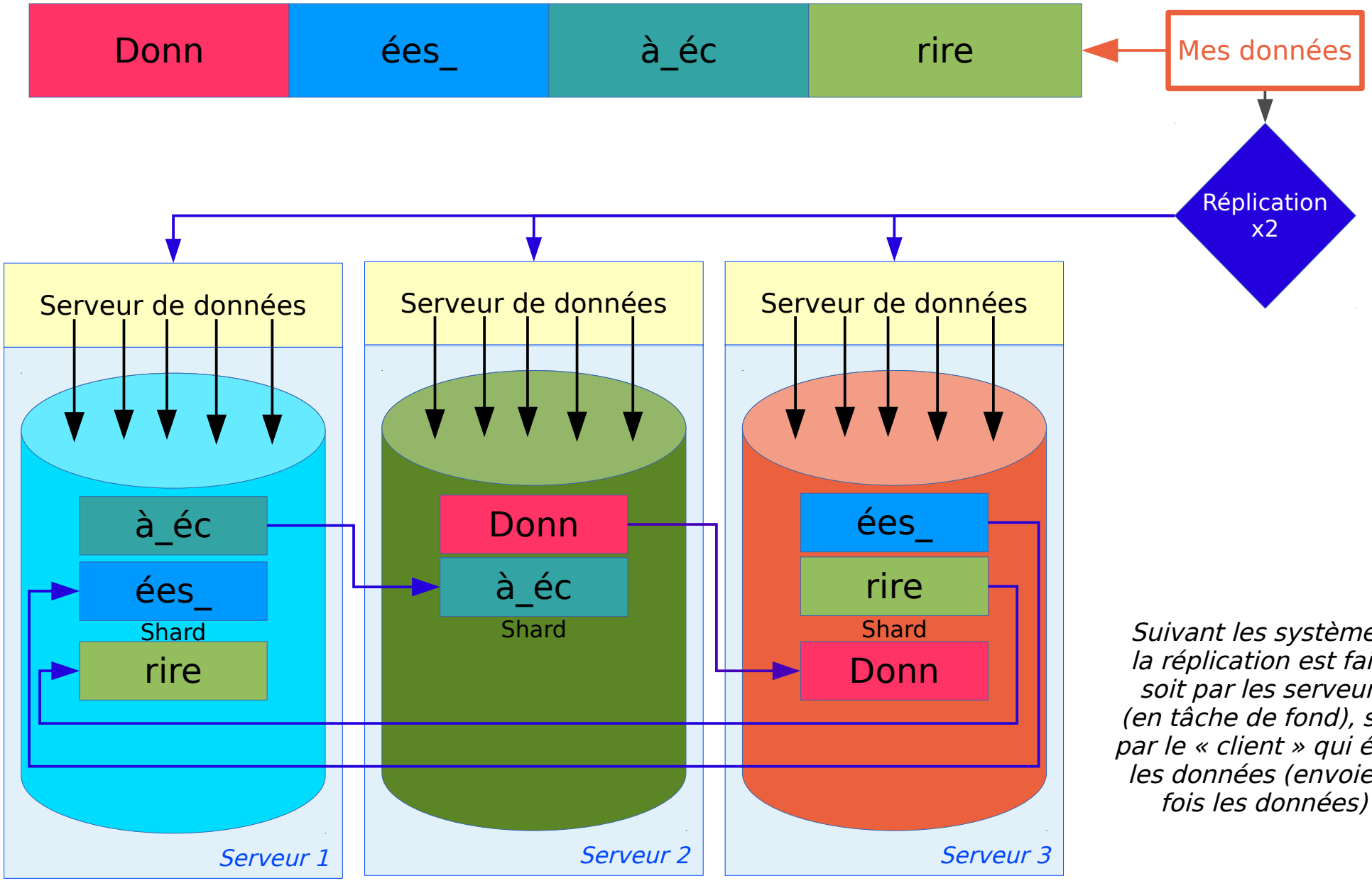

Donn
ées_
à_éc
rire
0
1
2
3
Mes données
Hash
Serveur de données
à_éc
Shard
Serveur 1
Serveur de données
Donn
Shard
Serveur 2
Serveur de données
ées_
Shard
Serveur 3



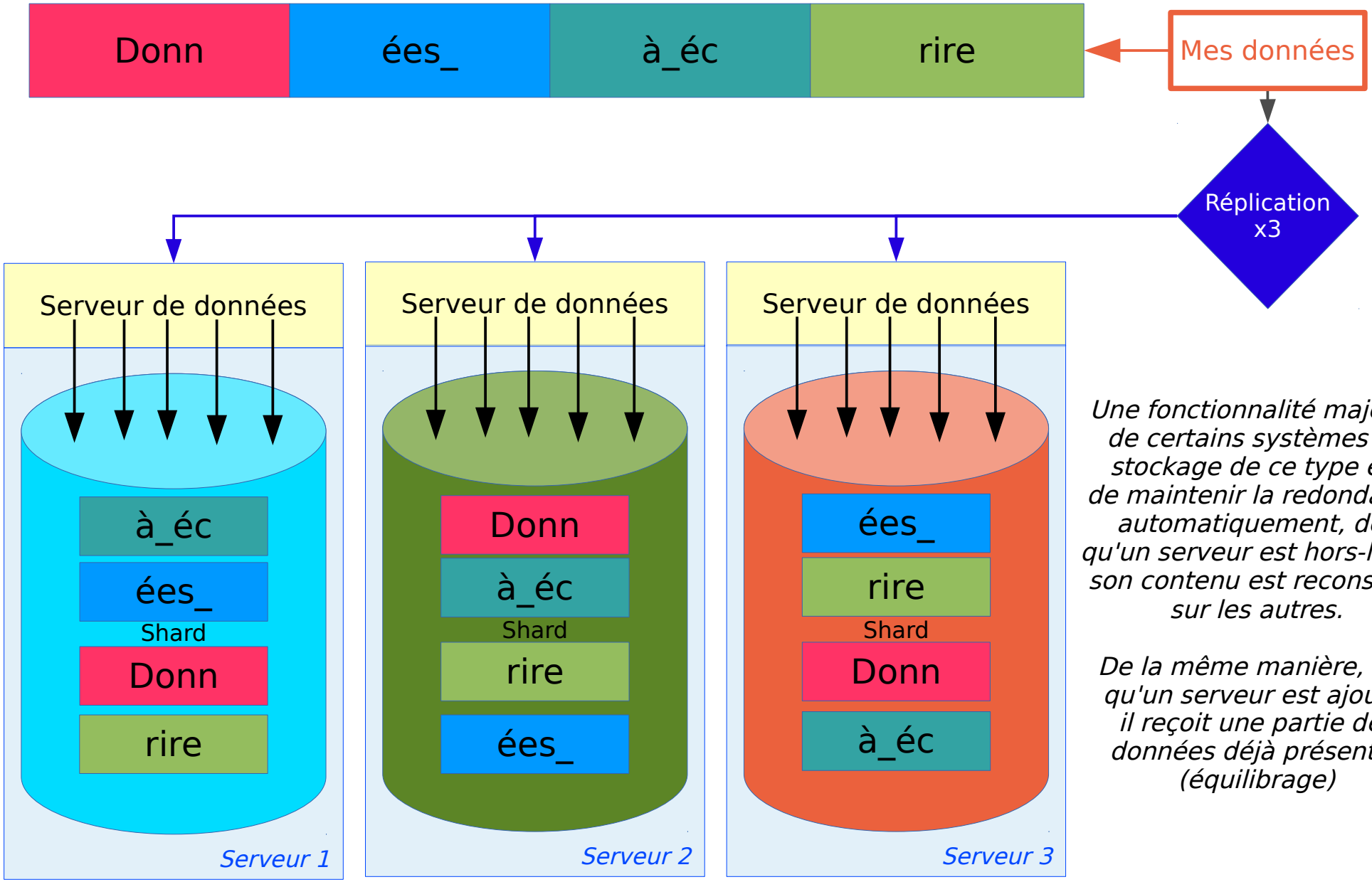
Sans redondance sur les serveurs de « shard », on ne fait que de la dispersion des données



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

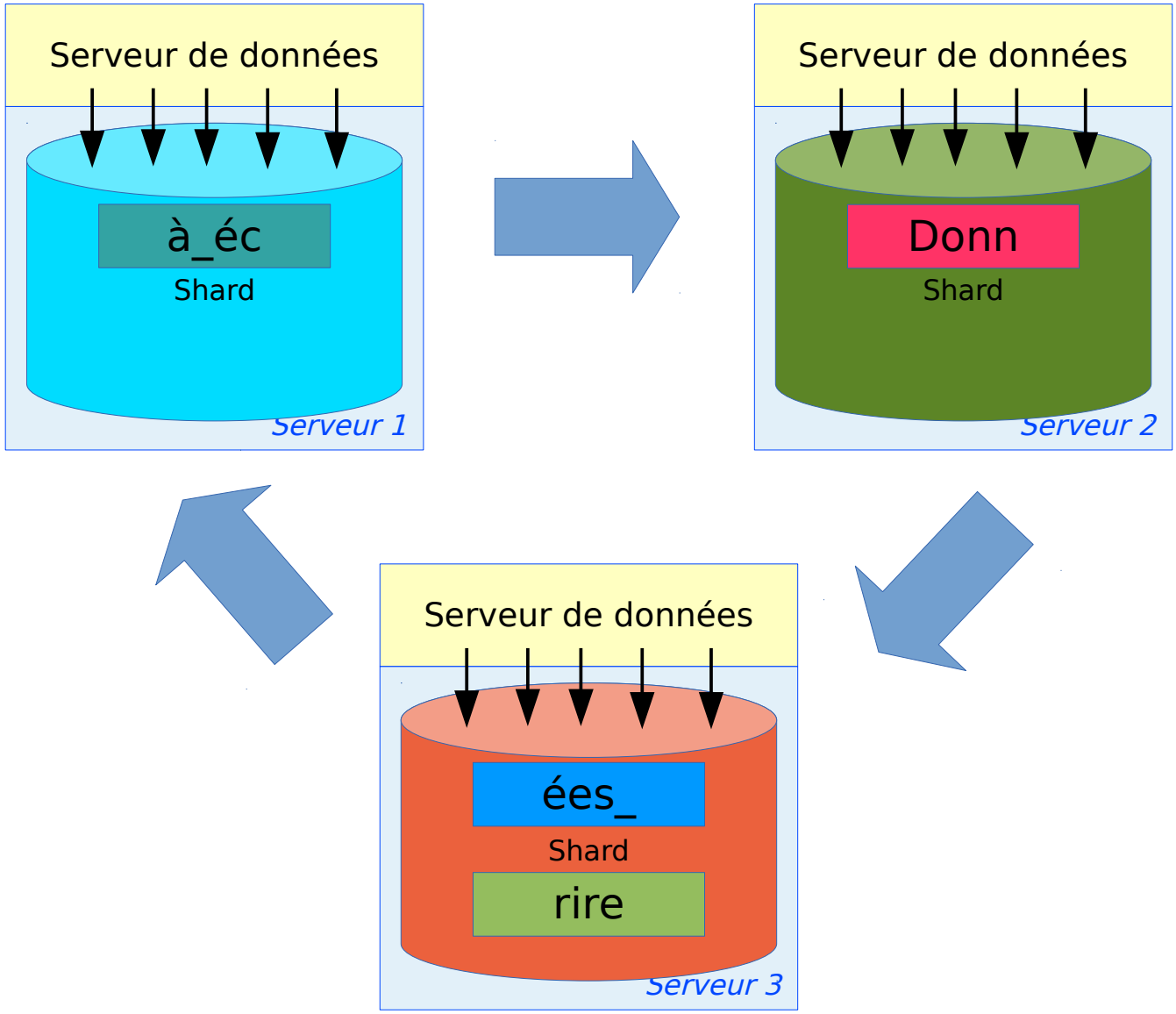


Suivant les systèmes, la réplication est faite soit par les serveurs (en tâche de fond), soit par le « client » qui écrit les données (envoie N fois les données)

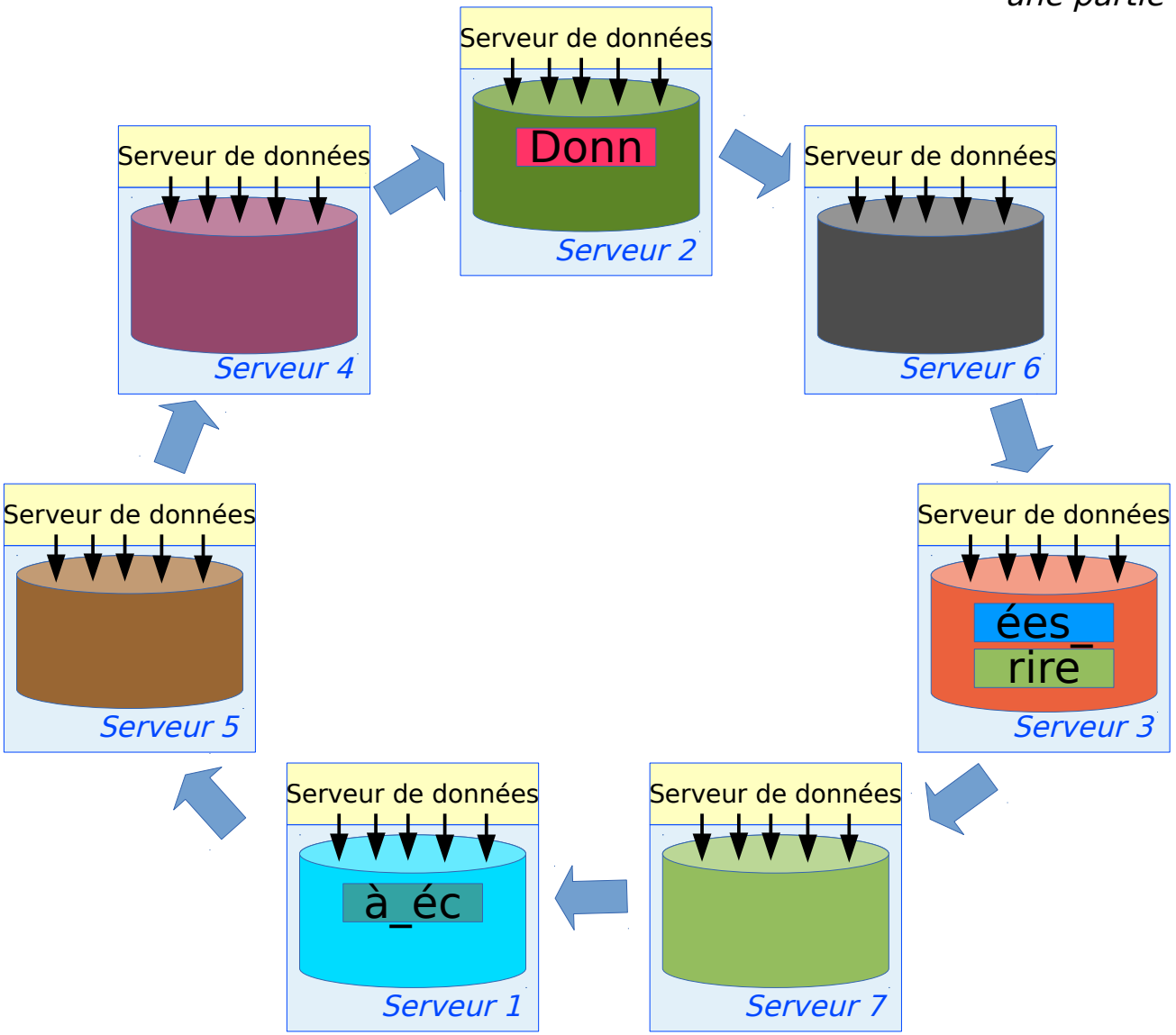


Une fonctionnalité majeure de certains systèmes de stockage de ce type est de maintenir la redondance automatiquement, dès qu'un serveur est hors-ligne, son contenu est reconstruit sur les autres.

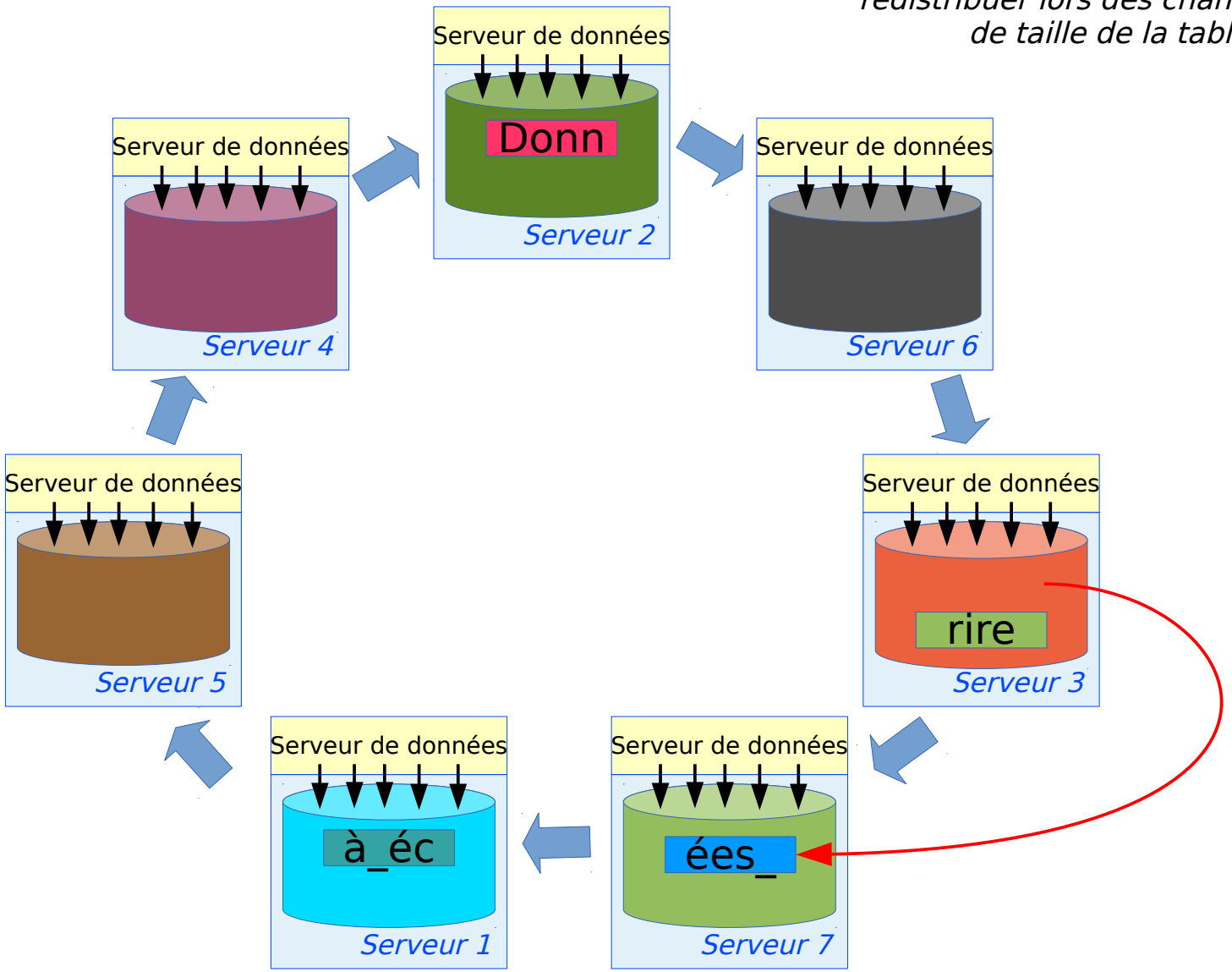
De la même manière, dès qu'un serveur est ajouté, il reçoit une partie des données déjà présentes (équilibrage)


Serveur de données
à_éc
Shard
Serveur 1
Serveur de données
Donn
Shard
Serveur 2
Serveur de données
ées_
Shard
rire
Serveur 3

Espace des clefs de la DHT découpé/partagé entre les membres de l'anneau.
Les nouveaux membres s'insèrent entre les membres existants et prennent en charge une partie de la DHT



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



- Données stockées sous forme d'objets de tailles variables contenus dans des LUNs SCSI, idéalement sans se préoccuper de serveurs avec des OSDs directement connectés au réseau
- Objectif de *simplification* par rapport aux systèmes existants, idéalement plus besoin de RAID, de gestion de volumes (LVM) ou de système de fichiers
- Les standards SCSI OSD ne s'occupent pas de la redondance des données
- Forte séparation entre les méta-données et les données :
  - Accès théoriquement possible à un objet sans utiliser de nom *logique*

- Possibilité pour les utilisateurs de définir des méta-données ou attributs par objet *(tags)* pour aider à la gestion des données
- Dans le standard, les objets sont identifiés par deux nombres (de 64 bits) *:*
  - *Partition ID* : *Périphérique* de stockage au sens le plus large (« disque », serveur, cluster)
  - *Object ID* : Permet au système de trouver la position (& longueur) de l'objet dans la *partition*
- Le *partition ID* n'a de sens qu'au sein de l'application ou du système de stockage
- L'*object ID* n'a de sens qu'au sein du *périphérique* final

- Si on veut des noms logiques, il faut un moyen de gérer ces méta-données ⇨ serveur ou service de méta-données indépendant des OSD eux-mêmes
- Dans le standard OSD-2 (2011) :
  - Notions de groupes d'objets *(collections)*
  - Instantanés *(snapshots)* et *clones* : *snapshots* accessibles en écriture
- En pratique, les standards SCSI OSD sont peu utilisés, la majorité du stockage objet utilise des protocoles spécifiques et non-standards

- Système de stockage qui gére les données non plus par bloc ou fichier mais par « objet »
- Chaque objet « existe » indépendamment des autres, pas nécessairement de relation entre les objets (pas d'espace de nommage hiérarchique ⇨ meilleure capacité de croissance)
- Implémentations « matérielles » (Seagate, DDN WOS, ...) et logicielles (nombreuses)
- Différent niveaux (éventuellement combinés) :
  - Périphérique/Unité de stockage (Seagate Kinetic, etc.)
  - Système de fichiers (Lustre, Panasas, Ceph, ...)

- Application de stockage (AWS S3, MS Azure, ...)
- Application utilisateur final (Dropbox, Facebook, Ubuntu ONE, ...)

- Au niveau application de stockage, l'API dominante est S3 (AWS) qui est très simple : API de type REST *(stateless)* et PUT/GET/DELETE d'objets **entiers**
- S3 permet de définir des ACLs pour l'accès aux objets et des attributs afin de créer des méta-données qui ont un sens pour l'application ou l'utilisateur final
- Chaque utilisateur de S3 dispose d'un conteneur appelé *bucket* (seau) qui est utilisé pour donner une vue logique de **ses** objets

- Chaque objet a un identifiant logique choisi par l'utilisateur appelé clef *(key)*, l'accès à un objet est possible avec une URL du type (exemple avec S3 à proprement parler, le principe est le même avec Swift, Azure, etc.) :

  `http://s3.amazonaws.com/`*`bucket/key`*

- En pratique, *sauf exception* (Seagate, DDN, ± Scality, ± Panasas, ...) les objets sont stockés dans des systèmes de fichiers (sous forme de fichiers)

Stockage objet : CDMI *(Cloud Data Management Interface)*

- Standard ouvert pour le stockage objet, beaucoup plus évolué et complexe que S3 : assez peu répandu en pratique (Scality, EMC ViPR, ...), principalement en raison de sa complexité
- Une fonctionnalité intéressante de CDMI est l'intégration du stockage objet et des accès de type systèmes de fichiers, ce qui réduit le nombre de copies nécessaires pour utiliser du stockage objet avec des applications « non-objet »

Failure domain : principe

- Avec des ressources de type « sans partage », il y a malgré tout souvent des éléments partagés : par exemple, l'infrastructure réseau *(switches)*, les blocs de distributions électriques, les armoires, voire la salle machine, etc.
- Certains systèmes de stockage de type *shared-nothing* (et certains avec des ressources partagées) peuvent utiliser ces informations pour augmenter ou maintenir la disponibilité des données afin de se prémunir contre les pannes d'une partie des ressources matérielles
- Un exemple commun est un fournisseur de stockage « cloud » avec plusieurs salles machines et qui réplique les données entre les salles machines

Failure domain : en pratique

- On décrit la topologie du matériel en groupant les matériels (serveurs, *rack*, *datacenter*, ...) dans des « groupes de disponibilité » appelés suivant les applications : *failure group, failure domain, protection domain*, *availability zone*, ... et le système de stockage s'assure de distribuer les données entre les groupes de disponibilité
- Dans certains systèmes de stockage, on peut définir des règles de placement des données pour définir explicitement comment les données vont être distribuées

- Dans un système de stockage distribué, on veut s'assurer que les machines impliquées «*voient* » les mêmes données
- Toutefois, les réseaux ne sont pas nécessairement fiables et il faut prévoir qu'il y aura des problèmes de communication (partition réseau) et dans les cas extrêmes la possibilité de conflits entre les différents serveurs impliqués *(split-brain)*
- La cohérence des données peut-être « relachée » et n'être « garantie » qu'après un certain laps de temps *(eventual consistency)*, cas de nombreuses bases de données NoSQL (données non critiques, réseaux sociaux, jeux en lignes, etc.)

- La cohérence peut être « forte » et souvent obtenue avec des protocoles de consensus (Paxos, Raft, ...) :
  - Paxos est le protocole de référence, utilisé par Ceph, ExtreemFS/Quobytes, GPFS, ...
  - Ces protocoles servent à déterminer quelles machines sont dans un état correct et à attribuer un rôle de gestion (et décision) à une ou plusieurs de ces machines (systèmes de fichiers distribués, Ceph, etc.)
  - Les machines qui ont un rôle de gestion et décision *(manager, monitor, ...)* sont souvent *élues* parmi un petit nombre de machines choisies par l'administrateur

- Les machines qui ont le droit de voter sont assez souvent appelées machines *quorum* (par abus de langage)

- Le « théorême » CAP *(Consistency, Availability, Partition-Tolerance)* indique que les concepteurs de systèmes de stockage distribué doivent faire des choix entre :
  - Cohérence globale (« garantie/forte » ou pas)
  - Disponibilité (requêtes « clientes » toujours honorées)
  - Résistance à des pannes partielles, y compris du réseau (partition : communication impossible entre certaines machines)
- D'après CAP, dans un système donné, seules deux des trois caractéristiques peuvent être satisfaites :

- La résistance aux problèmes réseaux est une fonctionnalité fondamentable/indispensable dans un système de stockage distribué
- Donc, en cas de partition, il faut donc choisir entre :
  - Disponibilité (répondre aux demandes, éventuellement des données pas à jour)
  - Cohérence (ne répondre que si on est sûr d'avoir les données à jour)

Cohérence globale : CAP & ACID

- Une version « moderne » de CAP, appelée PACELC, inclue le cas courant « pas de problème réseau » et dans ce cas le choix est entre :
  - Latence (réponse rapide mais peut-être pas la plus à jour)
  - Cohérence (réponse la plus à jour mais peut-être plus lente à fournir)
- Les bases de données ont un principe similaire à CAP mais toutefois **différent** : ACID *(Atomicity, Consistency, Isolation, Durability)*, objectifs/sens de la cohérence différents de CAP