Tolérance aux fautes - 2

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~/krakowia

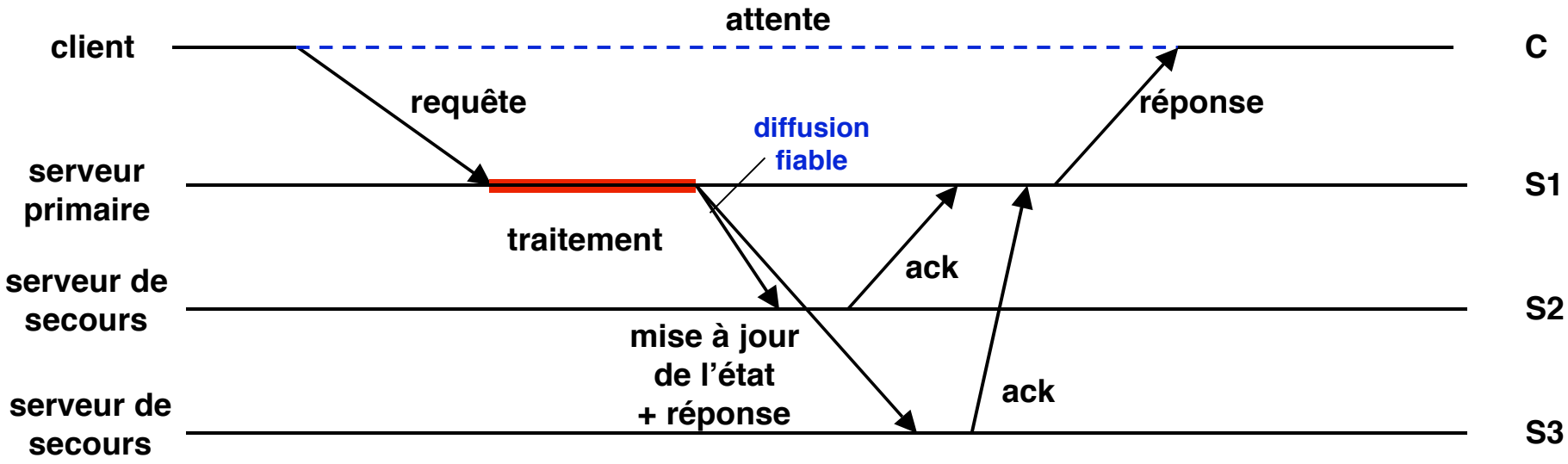
Serveurs tolérants aux fautes : principes

- **La tolérance aux fautes est obtenue par redondance**
  - On utilise N serveurs pour résister à la panne de N—1 serveurs
  - Hypothèse : panne franche (*fail stop*)
- **La redondance impose des contraintes de cohérence**
  - Vu du client, l'ensemble des N serveurs doit se comporter comme un serveur "abstrait" unique (les spécifications d'interface du serveur doivent être respectées)
- **Conditions suffisantes de cohérence**
  - Si une requête est traitée par un serveur, elle doit être traitée par tous les serveurs (en état de marche)
  - Deux requêtes différentes doivent être traitées dans le même ordre par tous les serveurs

Construction de serveurs tolérants aux fautes

- **Deux techniques de base (nombreuses variantes)**
  - Rappel : N serveurs pour résister à N—1 pannes (franches)
- **Serveur primaire - serveur de secours (poursuite)**
  - Un serveur (primaire) joue un rôle privilégié
  - La panne du primaire est visible par les clients (changement de serveur)
  - La panne d'un serveur de secours est invisible aux clients
- **Redondance active (compensation)**
  - Les N serveurs jouent un rôle symétrique et exécutent tous les mêmes requêtes
  - Les pannes sont invisibles aux client tant qu'il reste un serveur en marche

Serveur primaire - serveur de secours



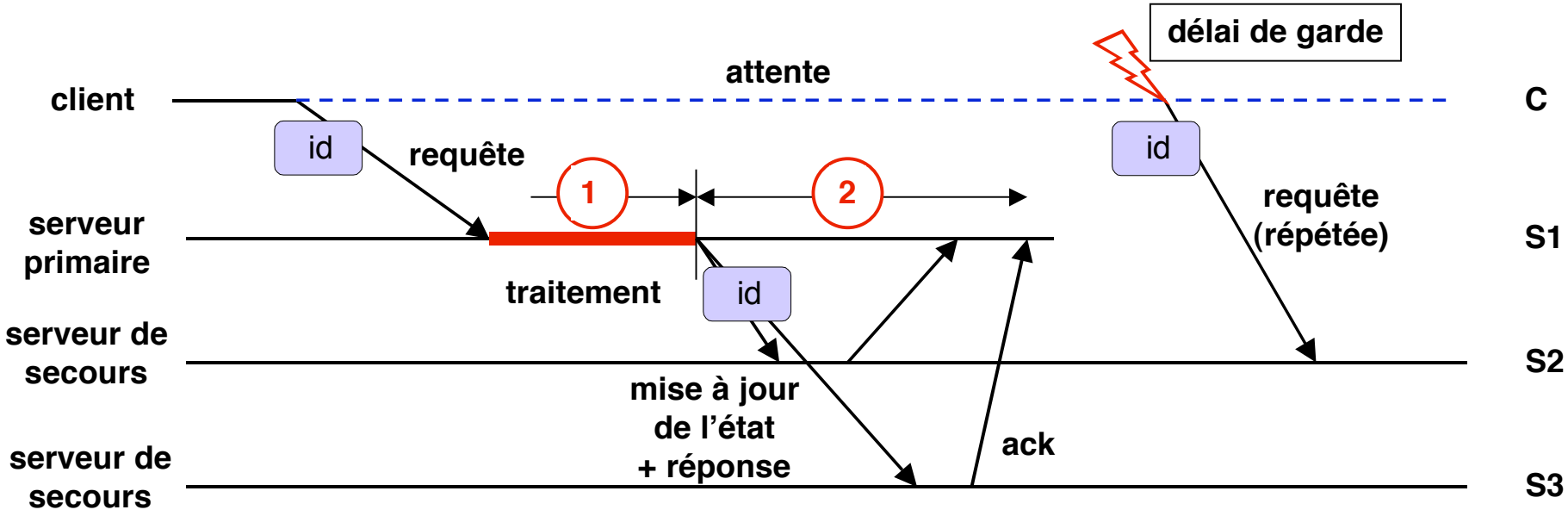
Diffusion fiable : le message parvient à tous ses destinataires non en panne, ou à aucun (pas de diffusion partielle)

Lorsque le primaire renvoie la réponse, tous les serveurs non en panne sont donc dans le même état

Tout serveur de secours peut remplacer le primaire en cas de panne (l'ordre de succession est défini à l'avance, par exemple S2 - S3)

Serveur primaire - serveur de secours

Traitement des défaillances



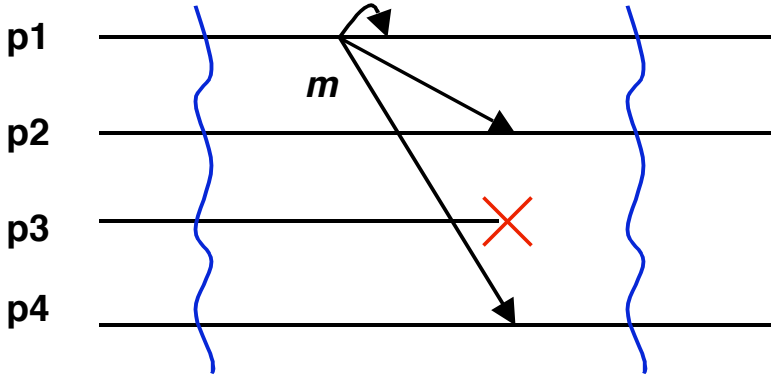
Défaillance du primaire : détectée par le client (délai de garde) et par les serveurs de secours (mécanisme de groupe, défini plus loin)

Le client contacte le serveur de secours qui a pris la place du primaire (il connaît son adresse)
Grâce à la diffusion fiable, tous les serveurs de secours sont à jour, ou aucun.

- **Si la panne est arrivée en 1 (aucun serveur de secours n'est à jour) : le nouveau primaire réexécute la requête et met à jour les serveurs de secours**
- **Si la panne est arrivée en 2 (tous les serveurs de secours sont à jour) : le nouveau primaire renvoie simplement la réponse**
- **Les requêtes ont une identification unique, ce qui permet de distinguer entre 1 et 2**

Serveur primaire - serveur de secours
Mécanisme de base

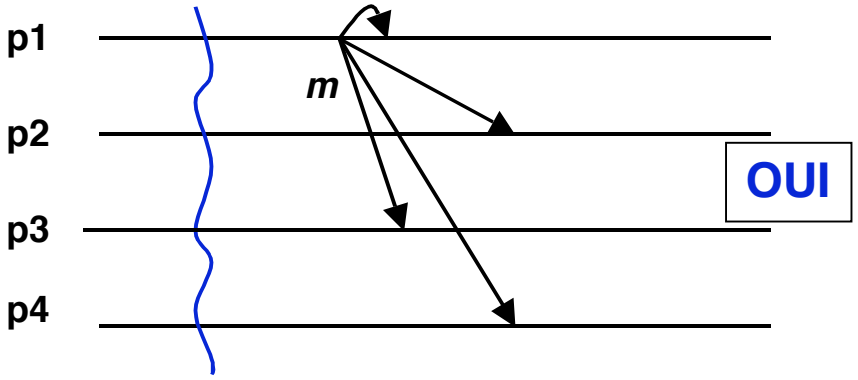
- **Gestion de groupe**
  - Mécanisme permettant de connaître à tout instant la composition d'un groupe de processus (ou serveurs), c-à-d. l'ensemble des processus valides (non en panne)
  - Outil de base : la "vue" : message contenant la liste des processus valides
  - Propriété de cohérence : vues synchrones (cohérence entre diffusion des vues et diffusion des messages)



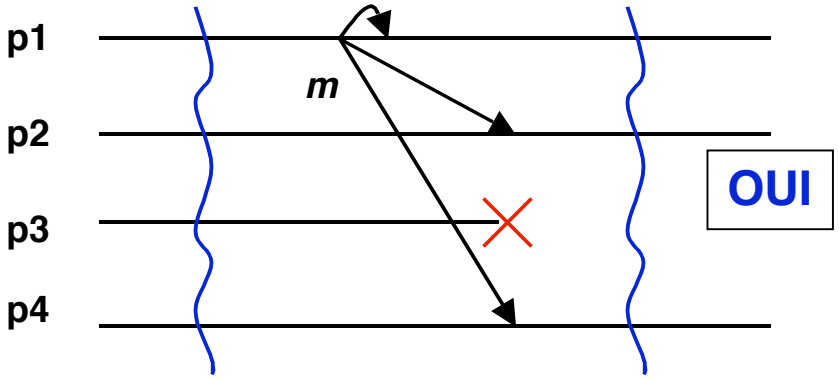
v_i=(p1, p2, p3, p4) v_{i+1}=(p1, p2, p4)

Soit *m* un message diffusé dans une vue v_i
Soit P = { processus communs à v_i et v_{i+1}}
Alors :
- ou bien *m* parvient à **tous** les membres de P avant v_{i+1}
- ou bien *m* ne parvient à **aucun** membre de P

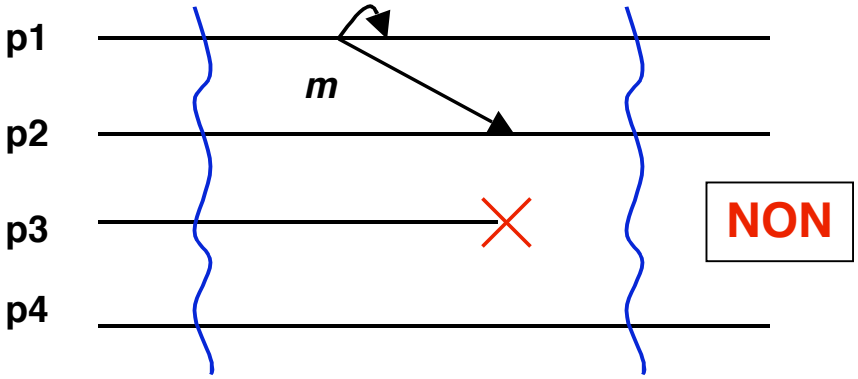
Vues synchrones



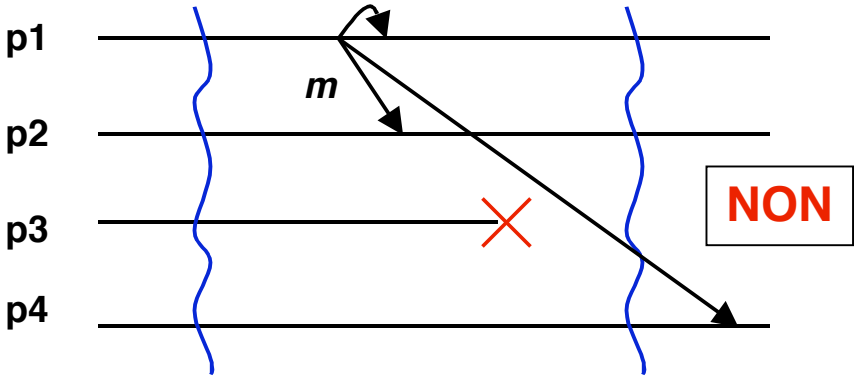
v_i=(p1, p2, p3, p4)



v_i=(p1, p2, p3, p4) v_{i+1}=(p1, p2, p4)



v_i=(p1, p2, p3, p4) v_{i+1}=(p1, p2, p4)



v_i=(p1, p2, p3, p4) v_{i+1}=(p1, p2, p4)

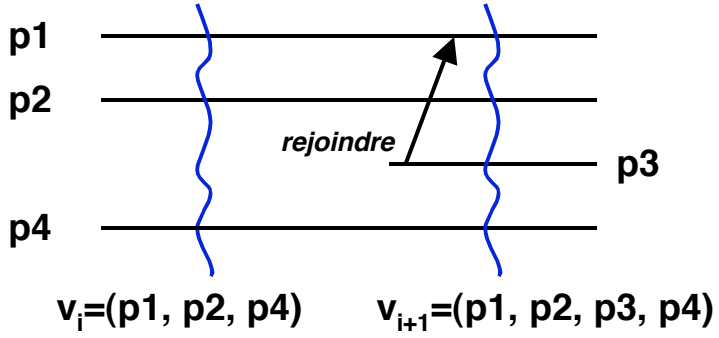
Serveur primaire - serveur de secours

Réinsertion après panne

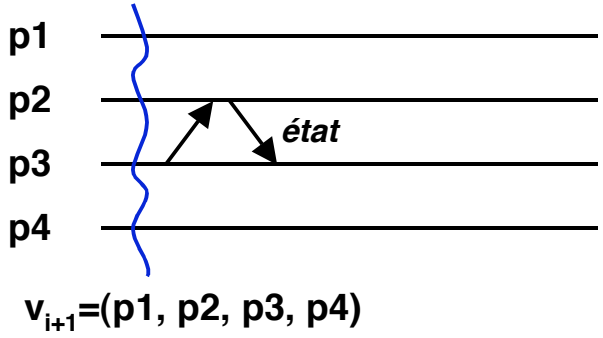
Après réparation, le serveur défaillant doit se réinsérer et restaurer son état

Réalisé grâce au mécanisme de groupe

1) le serveur qui se réinsère envoie un message *rejoindre* qui provoque un changement de vue



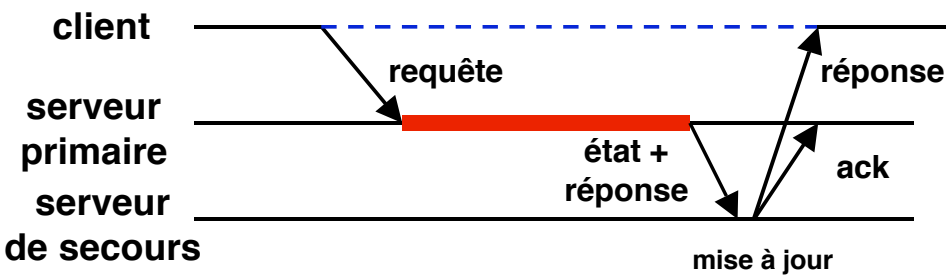
2) le serveur réinsèré demande une copie de l'état à un des autres serveurs



Le serveur réinsèré peut fonctionner (répondre aux requêtes)

Serveur primaire - serveur de secours

Cas particulier de deux serveurs (schéma de Alsberg & Day)



On gagne un peu de temps par rapport au schéma classique (réponse par le serveur de secours)
On n'a pas besoin d'un mécanisme de vues
Le serveur de secours surveille le primaire

En cas de panne du serveur primaire

Détection par le client (délai de garde)
Le serveur de secours devient primaire
Réinsertion du primaire (comme serveur de secours) après réparation, avec copie de l'état

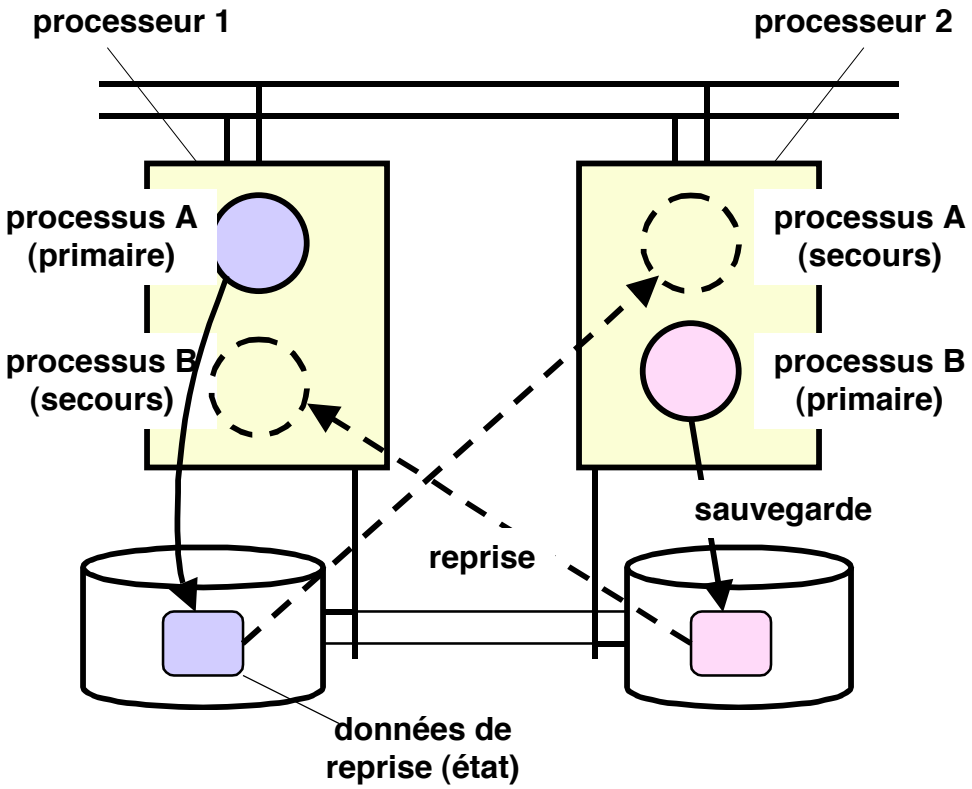
En cas de panne du serveur de secours

Rien à faire par le client
Réinsertion après réparation (copie de l'état)

Le schéma tolère la défaillance d'un serveur

Serveur primaire - serveur de secours

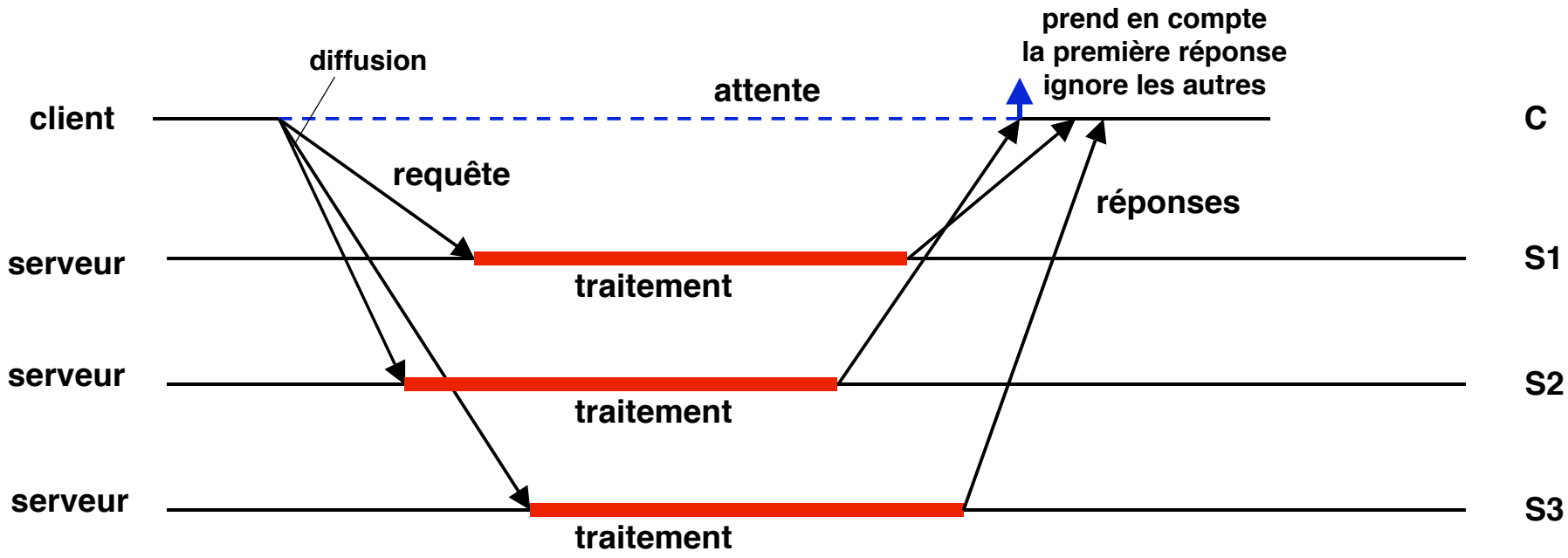
Application à la reprise dans Tandem



Principe de la reprise

- Sauvegarde périodique sur disque de l'état des processus. Surveillance mutuelle de chaque système par l'autre (messages "je suis vivant").
- En cas de dépassement du délai de garde, le processeur survivant reprend les processus défaillants à partir du dernier point de reprise enregistré.

Redondance active



Tous les serveurs sont équivalents et exécutent le même traitement

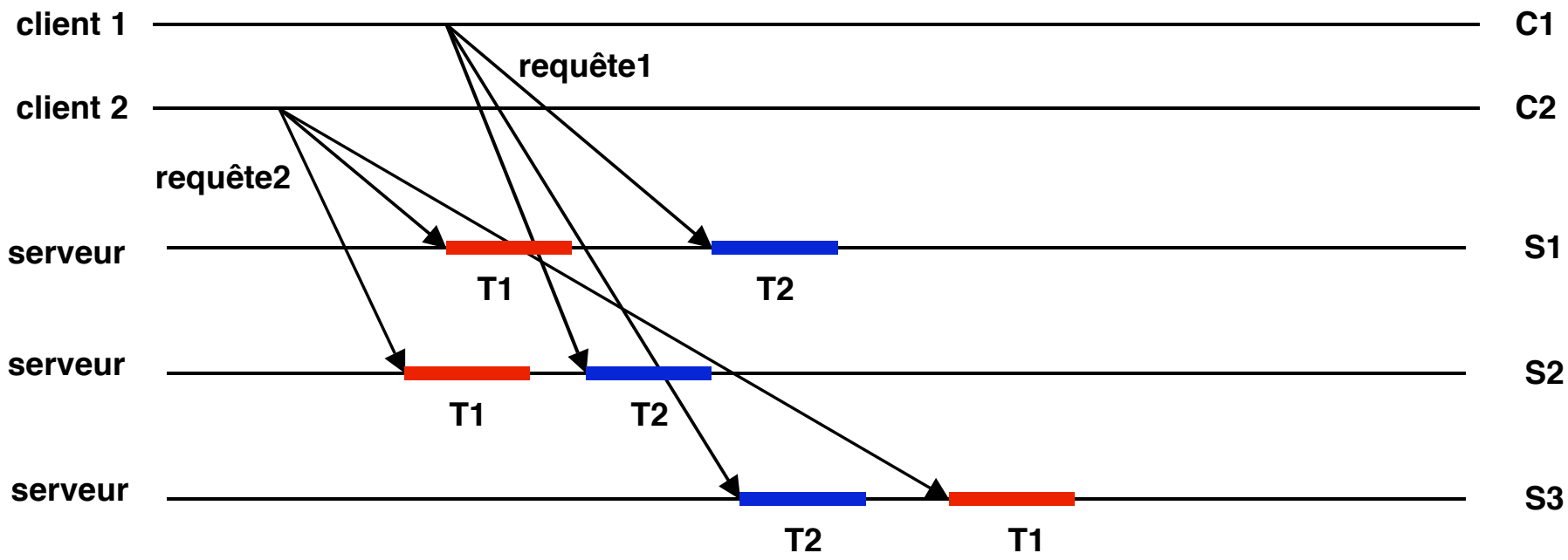
La diffusion doit avoir les propriétés suivantes (pour satisfaire la cohérence)

- Un message diffusé est reçu par tous les destinataires (non en panne), ou par aucun
- Deux messages différents sont reçus dans le même ordre par leurs destinataires

Ces propriétés définissent la diffusion atomique, ou totalement ordonnée

Redondance active

Nécessité de la diffusion atomique



Dans l'exemple ci-dessus, la diffusion n'est pas atomique

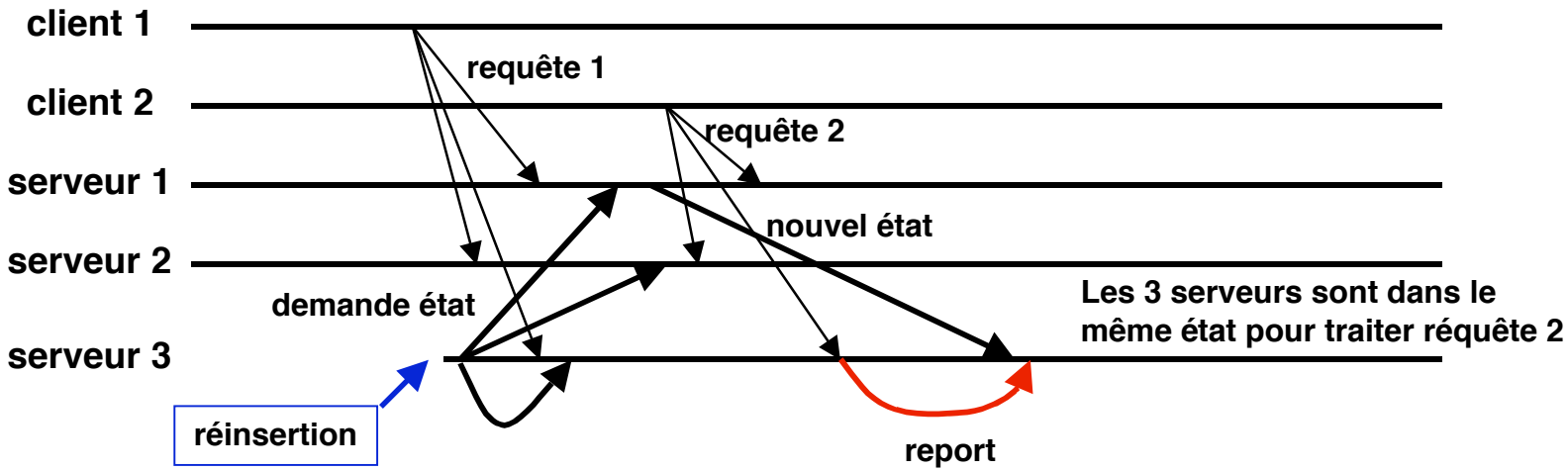
Serveur 1 : T1 ; T2
Serveur 2 : T1 ; T2
Serveur 3 : T2 ; T1

Si les traitements modifient l'état des serveurs, le système est incohérent

Redondance active
Réinsertion après panne

La panne d'un serveur est invisible aux clients

Pour se réinsérer après une panne, un serveur diffuse (atomiquement) une demande de reconstitution d'état



Comportement du nouveau serveur

- **Un message tel que requête 1 (parvenant avant sa propre demande d'état) est ignoré**
- **Le traitement d'un message tel que requête 2 (parvenant avant la réponse à la demande d'état) est retardé jusqu'après l'arrivée du nouvel état**

Motivation : préserver la cohérence, grâce aux propriétés de la diffusion atomique

Comparaison entre serveur primaire et redondance active

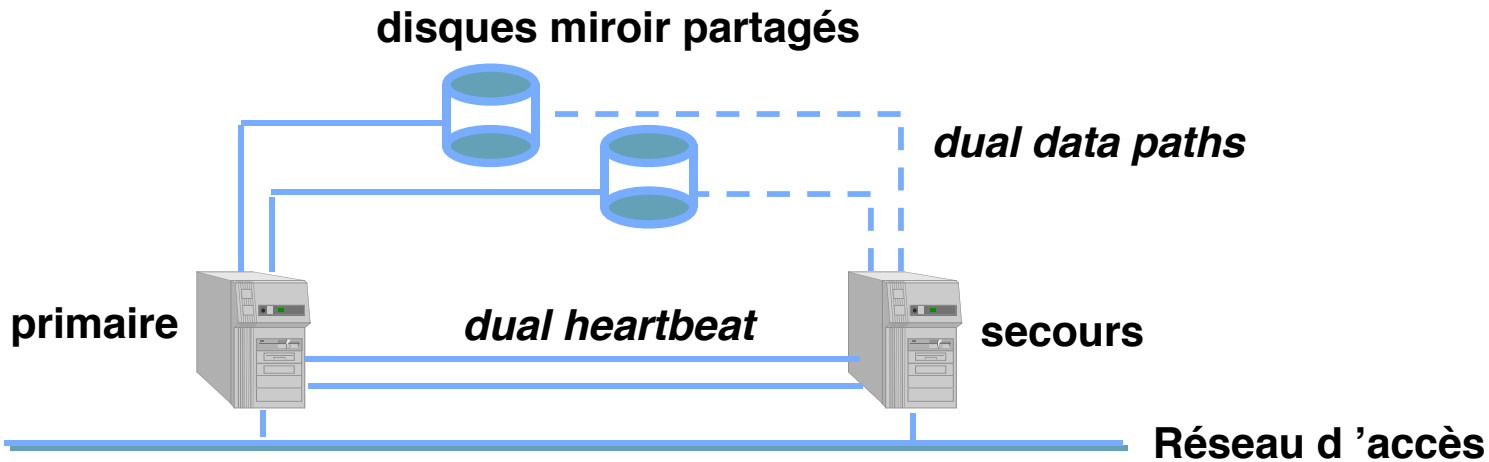
- **Mécanismes nécessaires**
  - Serveur primaire : technique de groupe dynamique (vue synchrones, sauf si 1 seul serveur de secours, cas très fréquent)
  - Redondance active : diffusion atomique
  - Les deux mécanismes sont équivalents (en complexité)
- **Usage des ressources**
  - Serveur primaire : les serveurs de secours peuvent exécuter une tâche de fond, non prioritaire ; reprise non immédiate
  - Redondance active : tous les serveurs sont mobilisés, reprise immédiate
- **Travail pour les clients**
  - Serveur primaire : le client doit détecter la panne du primaire
  - Redondance active : le client n'a rien à faire
- **Conclusion**
  - Serveur primaire : le plus utilisé dans les applications courantes
  - Redondance active : applications critiques, temps réel

Serveurs à haute disponibilité

Serveur d'information redondant

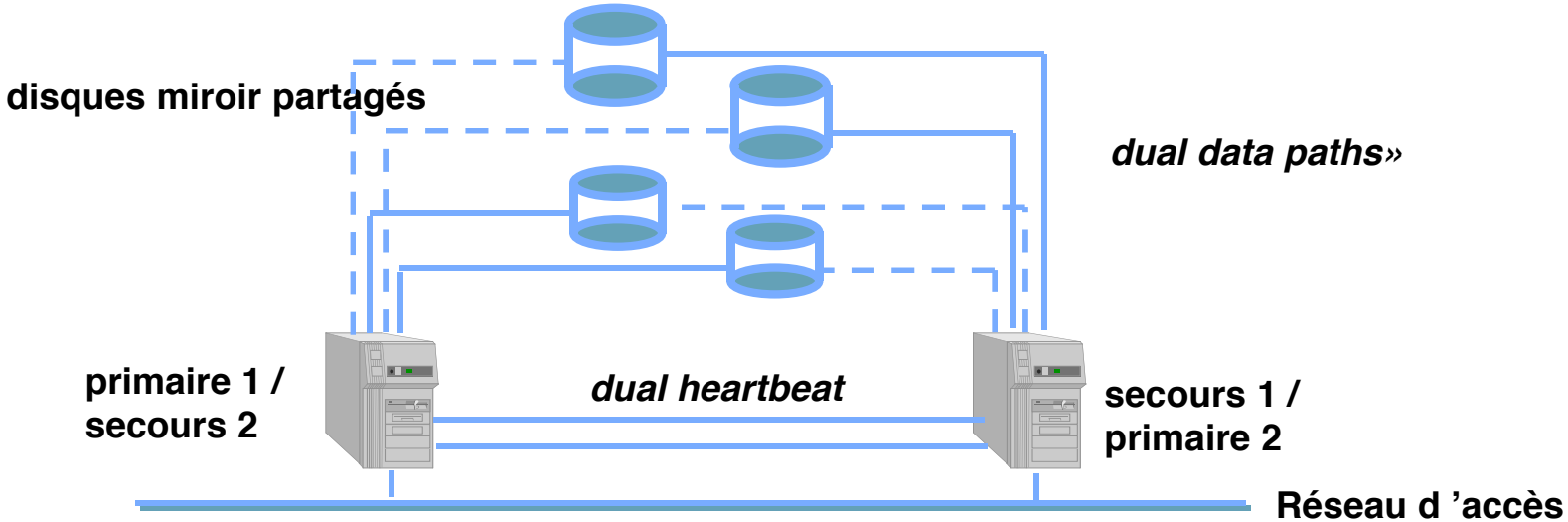
- Duplication passive des serveurs : primaire et secours
- Réseaux : réseau d'accès, réseau d'administration, réseaux *heartbeat*
- Disques :
  - Disques privés (par serveur)
  - Disques partagés (redondants -> RAID miroir ou parité)
- Pas de point de défaillance unique (*single point of failure*)
  - Alimentation électrique
  - Événements externes

Serveurs : configurations (1)



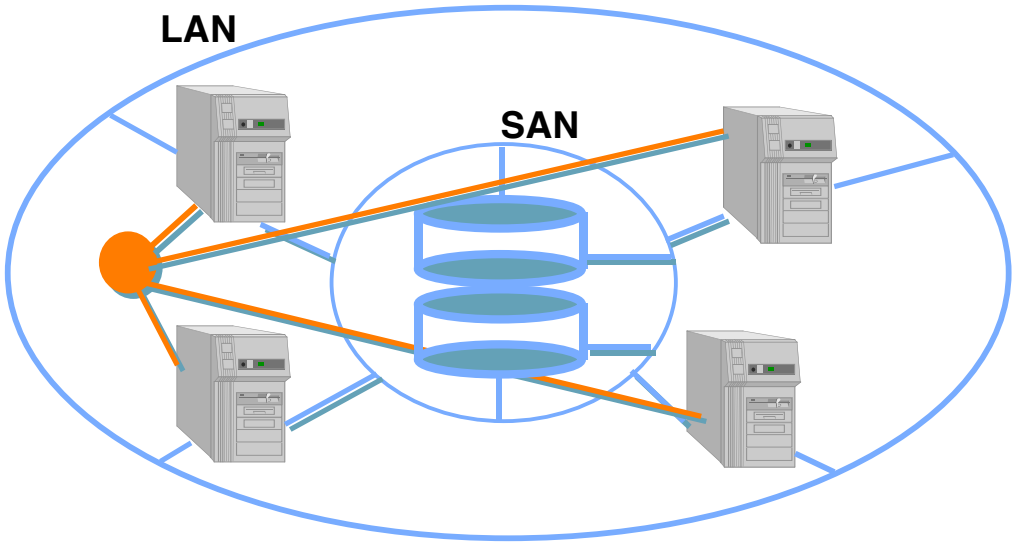
- Configuration 1-1 asymétrique
- Réseaux *heartbeat* dédiés
- Les disques partagés contiennent l 'état du système
- Bascule (*failover*) entre primaire et secours en cas de panne du primaire (accès aux disques partagés bascule également)

Serveurs : configurations (2)



- Configuration symétrique
- Les serveurs restent indépendants l'un de l'autre
- Moindre coût
- Impact sur la performance en cas de bascule
- Peut être généralisée à plusieurs applications (*service level failover*)

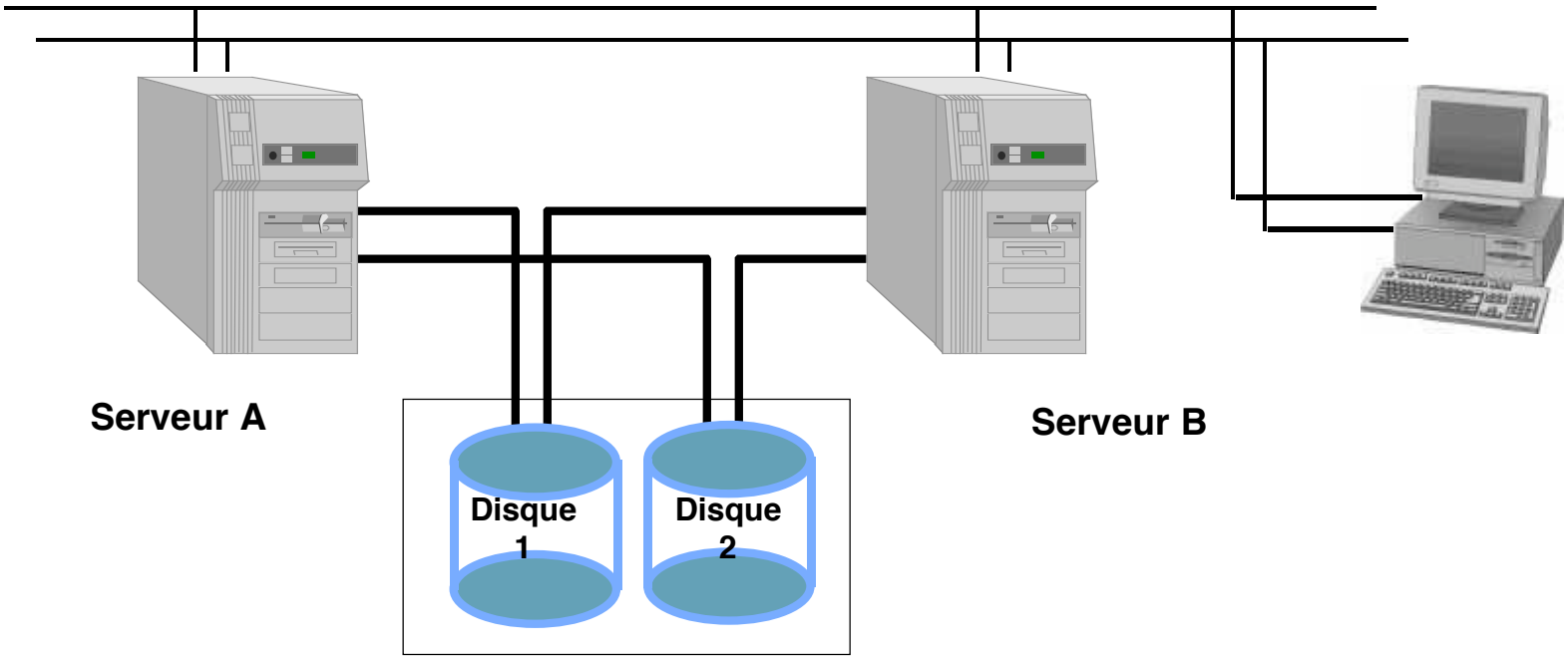
Serveurs : configurations (3)



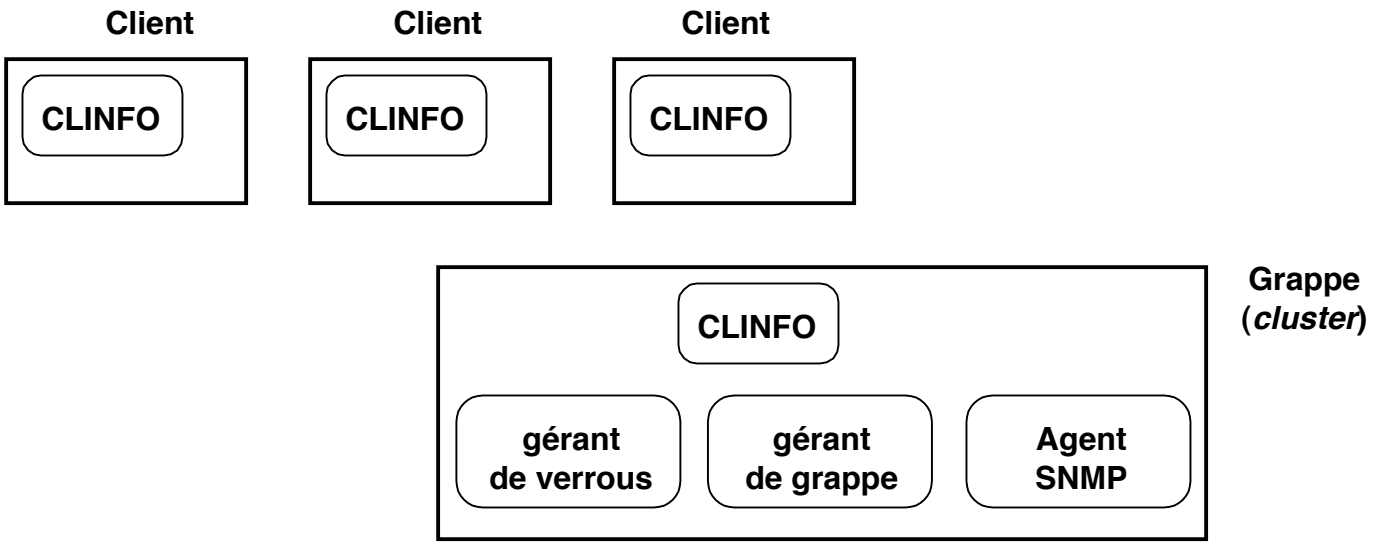
- Accès aux disques miroir via un SAN
- Réseaux *heartbeat* séparés
- Flexibilité accrue
- Utilisation pour serveurs en grappes (*server farms*)

Exemple de système à haute disponibilité : IBM HA-CMP

- **HA-CMP : *High Availability Cluster MultiProcessing***
  - **Principe : disque partagé (accès multiple)**



IBM HA-CMP : architecture logicielle (1)



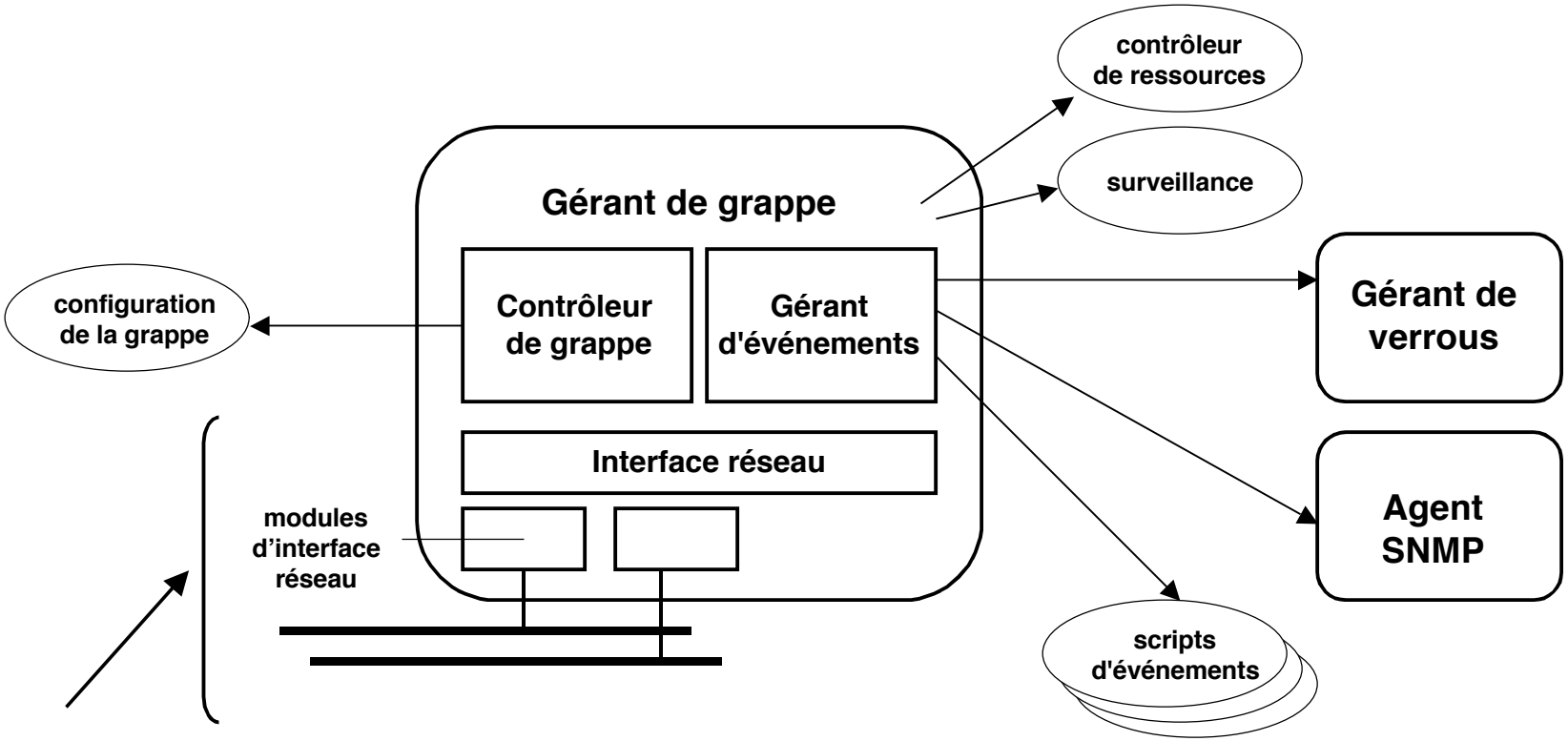
CLINFO : service d'information sur la grappe (informe les clients sur l'état de la grappe)

Agent SNMP : reçoit information du gérant de grappe et la rend accessible via SNMP (*Simple Network Management Protocol*, protocole d'administration des systèmes Unix)

Gérant de verrous : synchronisation distribuée pour la gestion des verrous des transactions

Gérant de grappe : gestion globale de ressources (voir suite)

IBM HA-CMP : architecture logicielle (2)



IBM HA-CMP : principe de la reprise

Plusieurs variantes

- **Recouvrement simple : l'application fonctionne sur l'un des systèmes (primaire). L'autre système est inactif ou exécute des tâches de fond, non critiques.**
- **Recouvrement mutuel : les applications sont exécutées sur les deux systèmes, mais sans partage d'information (fichiers disjoints). En cas de défaillance d'un système, ses applications sont reprises par l'autre.**
- **Partage de charge : les applications sont exécutées sur les deux systèmes et peuvent partager des données. Mais ce partage passe nécessairement par un gérant distribué de verrous pour assurer des propriétés transactionnelles**

Dans tous les cas, la communication (pour reprise) se fait toujours par le disque partagé.

Groupes et diffusion : un outil pour la haute disponibilité

Un **groupe** de processus est un ensemble spécifié de processus, pour lequel on définit des fonctions liées à

- l'appartenance (*group membership*) : changement de composition du groupe, connaissance à tout instant de sa composition courante
- la diffusion (*broadcast* ou *multicast*) : communication d'information à un ensemble de processus (avec des propriétés spécifiées)

La composition d'un groupe peut changer soit par entrée ou sortie volontaire soit par suite de défaillances ou réinsertions ; certaines spécifications restreignent les changements possibles

Motivations

- ensemble de processus se comportant comme un processus unique, avec tolérance aux fautes : groupe de serveurs, gestion de données dupliquées, systèmes sur grappes de machines, etc.
- travail coopératif, partage d'information

Diffusion : définitions

La diffusion est un mode de communication dans lequel un processus émetteur envoie un message à un ensemble de processus destinataires.

Dans la diffusion générale (*broadcast*), les destinataires sont tous les processus d'un seul ensemble défini implicitement (mais qui peut ou non être rigoureusement identifié, cf plus loin). L'émetteur est également destinataire. Exemples : les membres d'un groupe unique, vus de l'intérieur du groupe ; "tous" les processus du système

Dans la diffusion de groupe (*multicast*) , les destinataires sont les membres d'un ou plusieurs groupe(s) spécifié(s), les groupes pouvant ne pas être disjoints. L'émetteur peut ne pas appartenir au(x) groupe(s) destinataires(s)

Les primitives sont notées broadcast (p, m) et multicast (p, m, {g})

p : émetteur
m : message
{g} : ensemble des groupes destinataires

Diffusion : propriétés (1)

Propriétés indépendantes de l'ordre d'émission (concernent uniquement les récepteurs).

Diffusion fiable : un message est délivré à tous ses destinataires ou à aucun

Diffusion totalement ordonnée (ou atomique) : la diffusion est fiable et les messages sont délivrés dans le même ordre à tous leurs destinataires

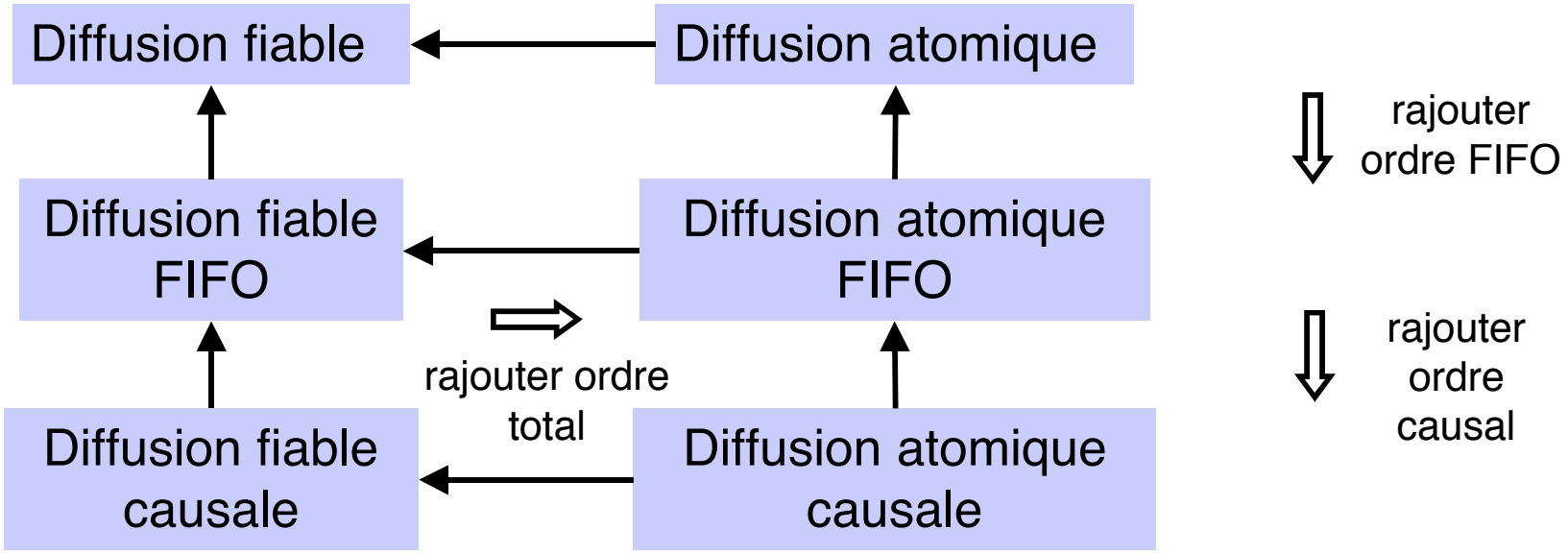
Propriétés liées à l'ordre d'émission

Diffusion FIFO : deux messages issus du même émetteur sont délivrés à tout récepteur dans leur ordre d'émission

Diffusion causale : pour tout récepteur, l'ordre de réception de deux messages respecte leur ordre causal d'émission (implique FIFO)

Les deux classes de propriétés sont indépendantes : 6 combinaisons possibles : fiable sans ordre, fiable FIFO, … , atomique causale

Diffusion : propriétés (2)



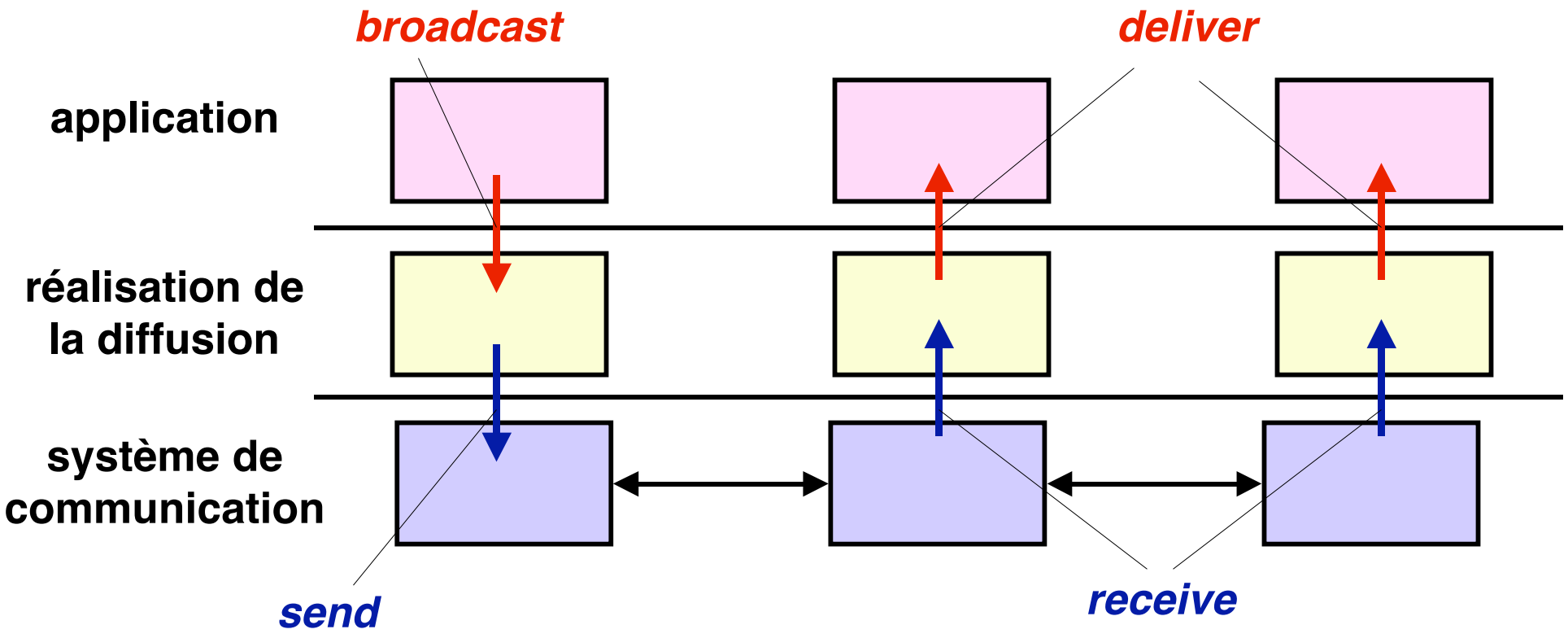
La flèche simple indique une implication (atomique implique fiable, etc)
On peut en outre ajouter les propriétés de temporisation et/ou d'uniformité

Temporisation : diffusion dans un délai fixé
Uniformité : les propriétés spécifiées s'appliquent à tous les processus, corrects ou défaillants

Réalisation de la diffusion (1)

La diffusion est réalisée au-dessus d'un **système de communication** permettant l'envoi et la réception de messages (primitives ***send*** et ***receive***).

Les primitives de la diffusion sont ***broadcast*** et ***deliver***



Réalisation de la diffusion (2)

On suppose que le système de communication est fiable (tout message finit par arriver, intact, s'il existe un lien physique entre émetteur et récepteur). Deux cas :

- **communication synchrone** : on connaît une borne supérieure pour le délai de transmission d'un message
- **communication asynchrone** : on ne connaît pas de borne supérieure pour le délai de transmission d'un message (exemple : l'Internet)

Dans le cas asynchrone, on ne sait pas déterminer à coup sûr si un processeur qui ne répond pas est en panne, ou si la communication avec lui est retardée.

Résultats

La diffusion fiable est réalisable simplement, même dans le cas asynchrone

La diffusion atomique (totalement ordonnée) n'est pas réalisable en asynchrone (cette propriété peut être démontrée).

Tous les algorithmes pratiques de réalisation de la diffusion atomique font donc une hypothèse de synchronisme (et utilisent un délai de garde)

Diffusion fiable (uniforme)

Tout processus p exécute le programme suivant :

pour exécuter **broadcast**(p, m) :

estampiller m avec
sender(m) (processus émetteur) et seq(m)
send(m) à tous les voisins de p, et à p

deliver(p, m) se produit comme suit :

sur exécution de **receive**(m)
par processus p
if p n'a pas précédemment exécuté
deliver(m) **then**
if sender(m) ≠ p **then send**(m)
à tous les voisins de p
deliver(m)

Le protocole ci-dessus réalise la diffusion fiable uniforme. Il tolère les pannes franches et les pertes de messages en réception, tant que la communication reste possible entre deux processus corrects

La propriété qui assure l'uniformité est que tout processus qui délivre un message l'a au préalable envoyé à ses voisins. Si ce n'était pas le cas, un processus pourrait défaillir après avoir délivré un message, sans que celui-ci soit délivré à un autre processus

Diffusion totalement ordonnée

Plusieurs méthodes sont utilisées (l'ordre peut être imposé soit à l'émission soit à la réception)

Méthode la plus courante

Utiliser un **séquenceur**. Le séquenceur est un site particulier. Pour diffuser un message, on l'**envoie** au séquenceur. Celui-ci lui attribue un numéro, dans une suite croissante (1, 2, 3, etc.), et l'**envoie** à tous les destinataires. Les destinataires **délivrent** les messages diffusés par un site dans l'ordre croissant des numéros (bien faire la différence entre envoyer (*send*) et délivrer (*deliver*).

Problème : résister à la panne du séquenceur (sera vu en TD)

Utilisé dans le protocole JavaGroups

Diffusion totalement ordonnée (2)

Autre méthode

Ordre utilisant des estampilles en réception. Chaque message m est estampillé provisoirement à l'arrivée par l'heure logique de réception (heure universelle compatible avec la causalité). Les différents récepteurs se communiquent leurs estampilles, et quand toutes sont connues, on attribue définitivement à m la plus grande. Donc tout message a la même estampille (définitive) sur tous les sites récepteurs. Les messages sont délivrés dans l'ordre de ces estampilles.

N.B. La stabilité est garantie par la règle du max : quand un message estampillé par *e* est délivré, un autre message non encore délivré ne peut recevoir une estampille inférieure à *e*.
Principe utilisé dans la version initiale d'ABCAST (Isis). Problème : le traitement des défaillances est complexe

Rappel sur les estampilles et horloges logiques

Une horloge logique (ou horloge de Lamport), sur un site, est un compteur, initialement 0, soit HL(i) sur le site i.

Règles d'incrémentation sur un site i:

- À chaque événement local à i, HL(i) := HL(i) + 1
- À chaque émission de message m, on crée une estampille E = HL(i) (heure logique d'émission), et on envoie (m, E)
- À chaque réception d'un message (m, E), on incrémente H(i) ainsi :

$$H(i) := \max(E, H(i)) + 1$$

La dernière règle assure la cohérence causale : l'heure logique de réception d'un message est toujours supérieure à son heure logique d'émission

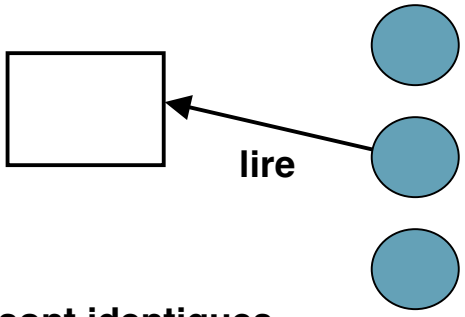
Duplication de données

- **Objectif : assurer la disponibilité des données**
  - **... en présence de défaillances (matérielles ou logicielles)**
  - **Le principe est toujours la redondance**
- **Techniques logicielles**
  - **Bases : déjà vues pour les serveurs (l'accès aux données est réalisé par un serveur particulier)**
  - **Des techniques spécifiques pour divers types de cohérence (plus ou moins stricte suivant besoins)**
- **Techniques matérielles**
  - **Disques RAID**
  - **SAN (*Storage Area Network*) avec duplication**

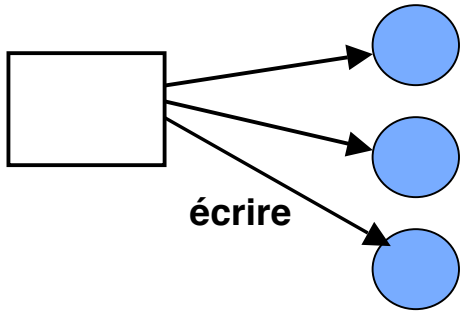
Duplication de données
Techniques logicielles

Solutions

- Un serveur par jeu de données, avec utilisation de l'une des techniques de disponibilité pour serveurs (primaire-secours ou duplication active)
- Ou bien : technique propre aux données dupliquées
  - deux opérations : lecture et écriture
  - technique élémentaire : *n* copies, lire un, écrire tous
  - inconvénient : écriture impossible si 1 copie en panne



Hypothèse :
toutes les copies sont identiques



Pour assurer que toutes les copies restent identiques : la diffusion des écritures doit être totalement ordonnée si les mises à jour ne commutent pas

Duplication de données
Vote majoritaire

- **Principe**
  - Autoriser une opération (écriture ou lecture) seulement si une majorité de copies est disponible
  - chaque copie a un numéro de version (incrémenté à l'écriture)

Exemple : 5 copies

pour écrire, il faut au moins 3 copies disponibles

pour lire, on cherche 3 copies ayant le même numéro de version (c'est nécessairement le plus récent).

Duplication de données
Vote pondéré (1)

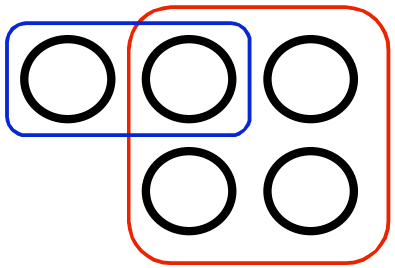
Principe

- Autoriser une opération (écriture ou lecture) seulement si un quorum est atteint (nombre de copies disponibles)
- Fixer les quorums (différents en lecture et en écriture selon les propriétés souhaitées

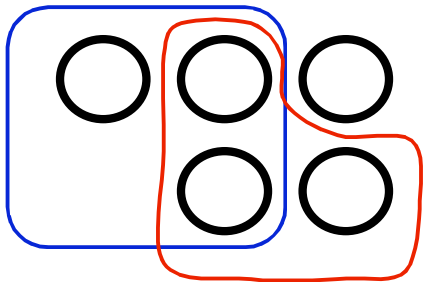
N = nombre de copies
nr = quorum de lecture, nw = quorum d'écriture, nr + nw > N

Exemple

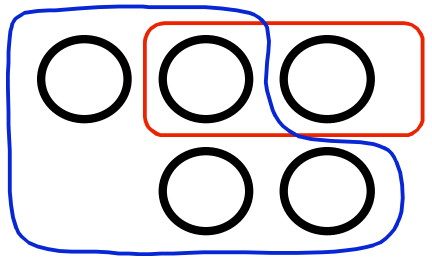
N=5, nr = 2, nw = 4



N=5, nr = 3, nw = 3



N=5, nr = 4, nw =2

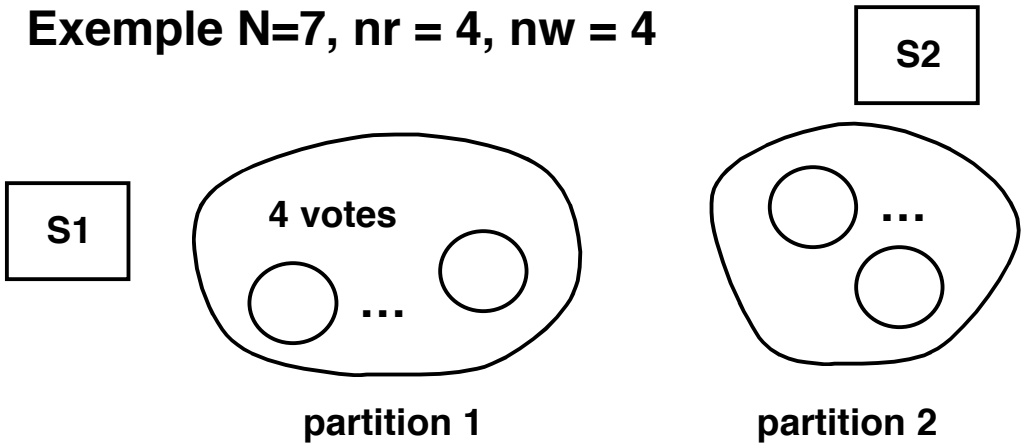


Duplication de données
Vote pondéré (2)

- **Principe**
  - Donner des poids différents aux copies selon leur importance présumée (raisons d'accessibilité, administration, etc.)
  - Utilité : partition de réseau (copies inaccessibles)

N = nombre de votes disponibles (et non plus nombre de copies)
nr = quorum de lecture, nw = quorum d'écriture
nr + nw > N et 2w > N

Exemple N=7, nr = 4, nw = 4

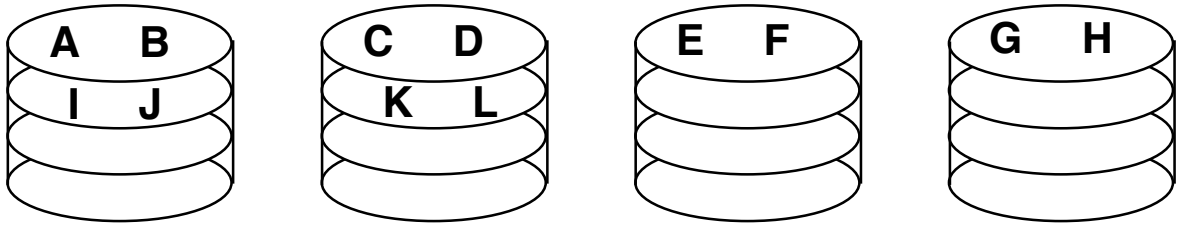


Si S1 écrit dans la la partition 1, on est sûr que S2 ne peut pas écrire dans la partition 2 (car elle ne peut pas avoir plus de 3 votes
Il est également impossible à S2 de lire pendant que S1 écrit (lectures et écritures sont sérialisées)

Disques RAID (Redundant Array of Inexpensive [ou Independent] Disks)

- **Motivations**
  - Augmenter les performances (bande passante) par l'accès parallèle
  - Augmenter la disponibilité par la redondance
- **Historique**
  - Origine : Berkeley, 1987
  - Nombreux produits commerciaux (HP AutoRaid, IBM, StorageTek, ...)
- **Classification**
  - "Striping" : RAID 0
  - Miroir : RAID 1
  - Accès parallèles synchronisés : RAID 2 et 3
  - Accès indépendants : RAID 4 et 5
  - Double redondance : RAID 6
  - Extensions diverses (ex : allocation dynamique)

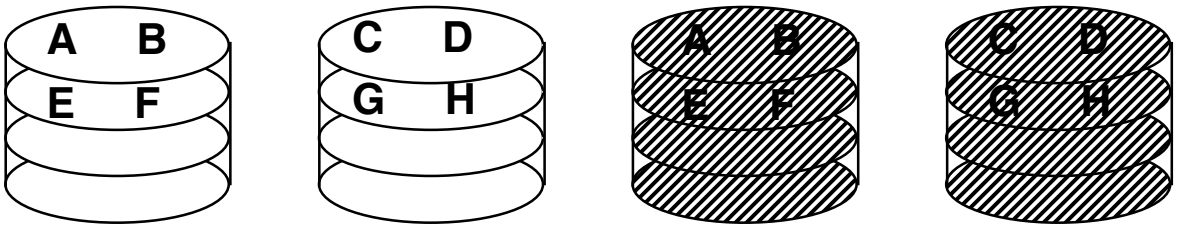
Disques RAID : RAID 0



- "Striping" séquentiel sur l'ensemble des disques
- Accès parallèle possible à des enregistrements indépendants
- Pas de protection des données contre les pannes
  - la défaillance d'un disque entraîne une perte de données

- Usage : calcul scientifique (performances d'accès = facteur dominant)

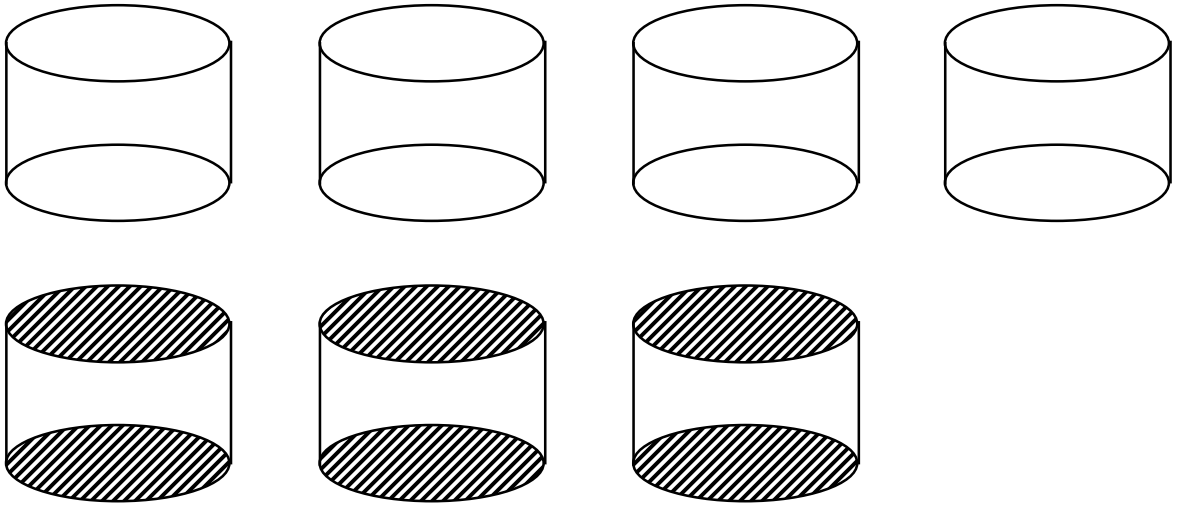
Disques RAID : RAID 1



disques miroir

- **“Striping” séquentiel sur l’ensemble des disques**
- **Accès parallèle possible à des enregistrements indépendants**
- **Redondance par duplication intégrale (disques miroir)**
  - **coût de la redondance = 100% de la capacité**
- **Usage : SGBD avec exigence forte de disponibilité et peu de contraintes de capacité**

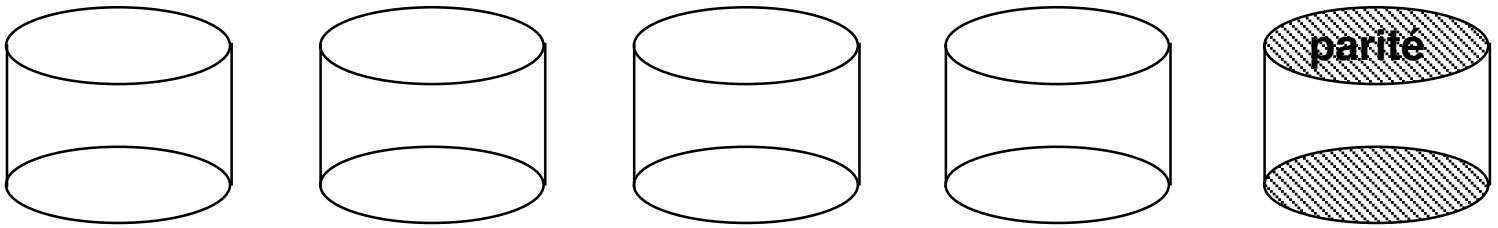
Disques RAID : RAID 2



- “Striping” séquentiel à grain fin sur l’ensemble des disques
- Accès synchrone à l’ensemble des disques
- Redondance par code correcteur d’erreur (Hamming)
  - capacité additionnelle croit comme log (nb. disques)
  - exemple : 4 (+ 3), 10 (+ 4)

- Usage : applications scientifiques (accès séquentiel, grand volume)

Disques RAID : RAID 3

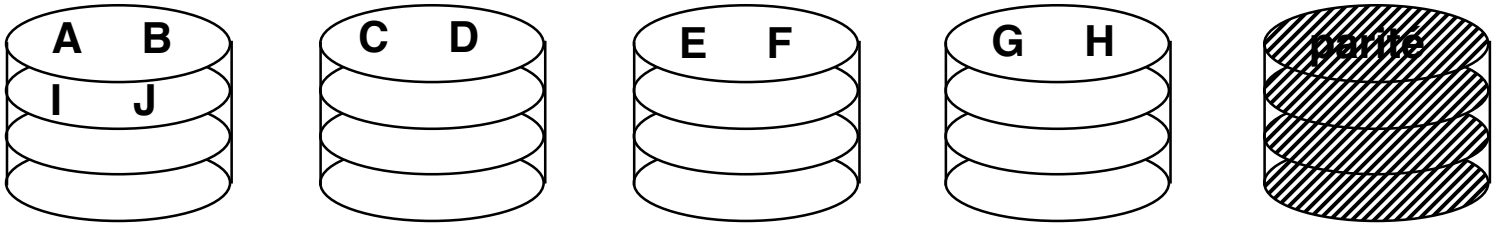


- L'information de parité permet de reconstruire le contenu d'un disque défaillant, mais nécessite l'accès à l'ensemble des disques valides.

- "Striping" séquentiel à grain fin sur l'ensemble des disques
- Accès synchronisés à l'ensemble des disques
- Redondance : informations de parité sur disque unique
- Tolère la défaillance d'un disque

Usage : transfert séquentiel de grands volumes de données

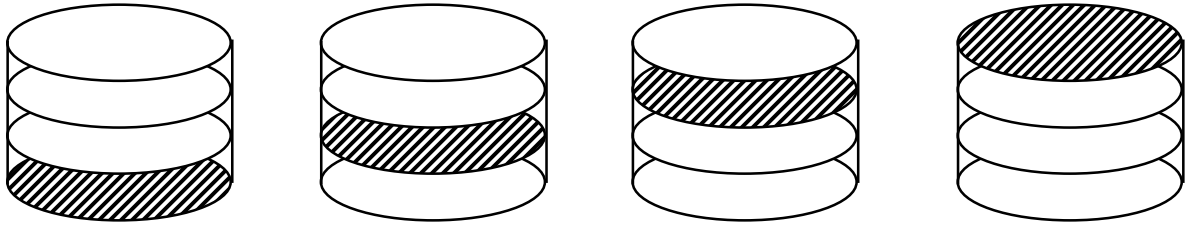
Disques RAID : RAID 4



- “Striping” séquentiel sur l’ensemble des disques, par blocs
- Accès non synchronisés (accès indépendants possibles en parallèle)
- Redondance : informations de parité sur disque unique

- Limitation : coût du maintien de la parité si mises à jour fréquentes - surcharge du disque de parité

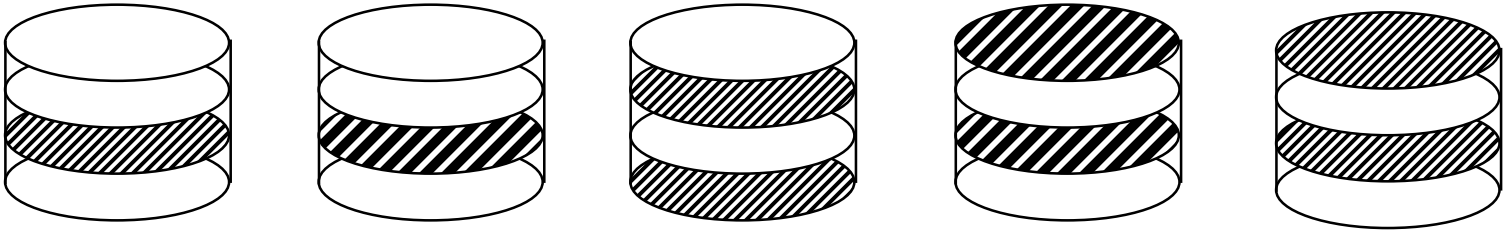
Disques RAID : RAID 5



- “Striping” séquentiel sur l’ensemble des disques, par blocs
- Accès non synchronisés (accès parallèle possible)
- Redondance : informations de parité réparties sur tous les disques (élimine la surcharge du disque unique de parité)

- Efficace pour lectures (toutes tailles) et grandes écritures
  moins efficaces pour petites écritures

Disques RAID : RAID 6



- Amélioration de RAID 5, tolère la défaillance de deux disques
- Caractéristiques : comme RAID 5, mais
  - inefficacité accrue en écriture (mise à jour parité)

Conclusions sur RAIDs

- La plupart des produits utilisent RAID 3 ou RAID 5
- Le réglage d'un RAID est un processus délicat
  - nombreux paramètres de réglage (taille des tranches, ...)
  - sensibilité aux variations des paramètres de réglage
  - sensibilité aux variations de charge
  - coût de la reconfiguration en cas de changement
- Problèmes de performances dans certaines conditions
  - pour les "petites écritures"
  - en mode dégradé -> redondance supplémentaire "manuelle"
- Solution encore relativement coûteuse
  - circuits spécialisés pour calcul de parité
- Quelques tendances
  - ajouter "de l'intelligence" dans les contrôleurs
  - gestion à plusieurs niveaux
  - gestion adaptative