Master d'Informatique

SAM - 4I803 – COURS 8

Réplication

2023

La réplication

Plan:

Objectifs
Fonctions
Propagation
Détection des modifications
Etude de cas: Dynamo DB

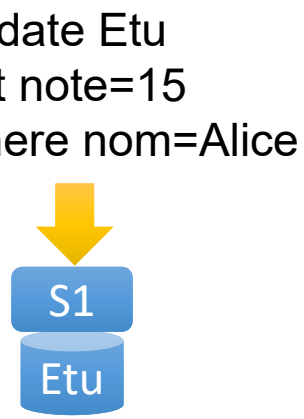
Objectifs de la réplication

+ Accès simplifié, plus performant pour les lectures
+ Résistance aux pannes
+ Parallélisme accru
+ Evite des transferts

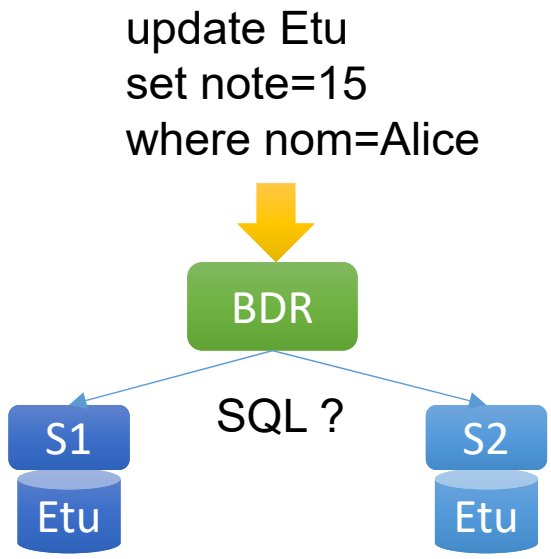
- Surcoût (*overhead*) en mise à jour
- Cohérence des données
- Toujours bien si on privilégie les lectures et/ou si peu de conflit entre màj

Problèmes de la réplication

- Donnée **non** répliquée :
  - stockage sur un site et accès réseau depuis les autres sites
  - problèmes de performances et de disponibilité



- Données répliquées
  - **Etu@S1** = Etu@S2
  - Réplication transparente
    - Accès à **un seul site** ?
  - Géo-réplication
    - Distance entre S1 et S2



Fonctions d'un réplicateur

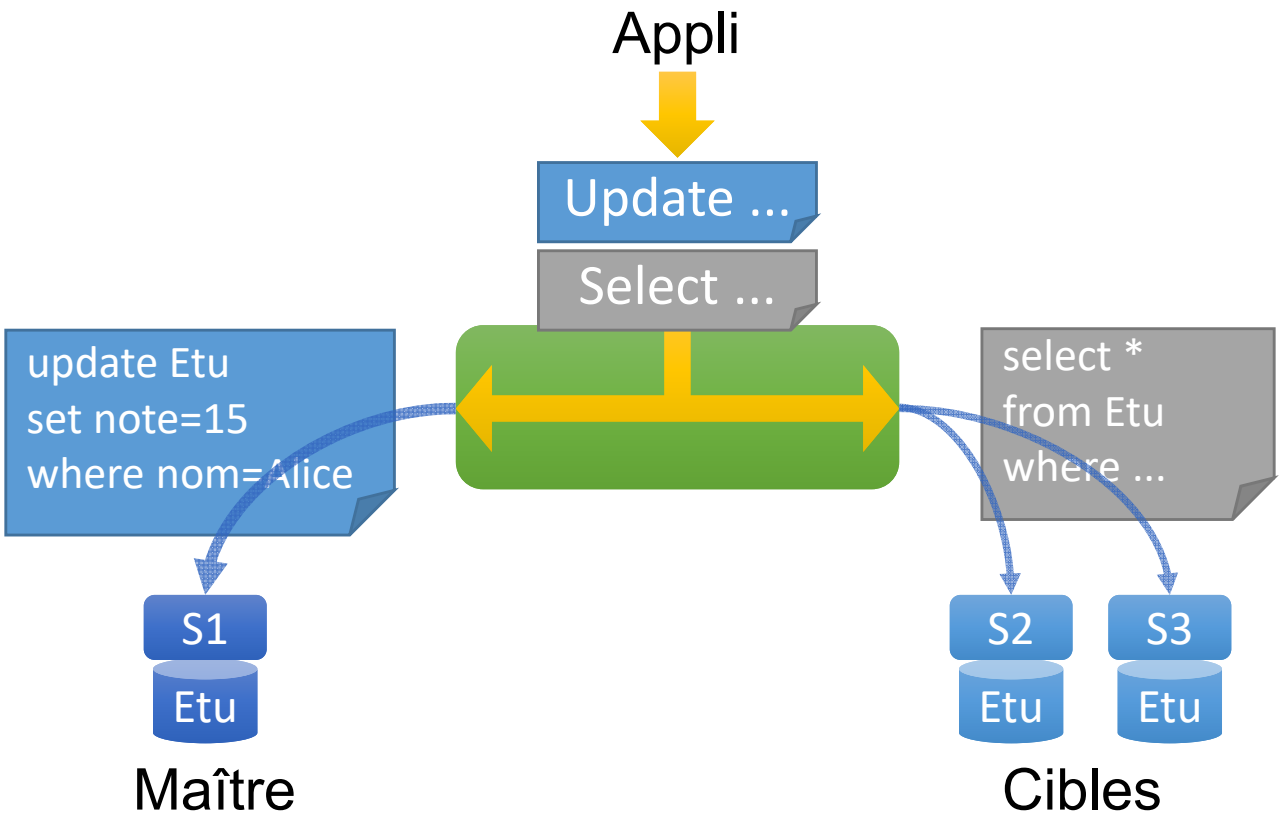
- Définition des objets répliqués
  - table cible = sous-ensemble horizontal et/ou vertical d'une ou plusieurs tables

- Définition de la fréquence de rafraîchissement
  - immédiat (après mise à jour des tables primaires)
  - à intervalles réguliers (heure, jour, etc.)
  - à partir d'un événement produit par l'application

- Rafraîchissement
  - complet ou partiel (propagation des modifications)
  - push (primaire -> cibles) ou pull (cible -> primaire)

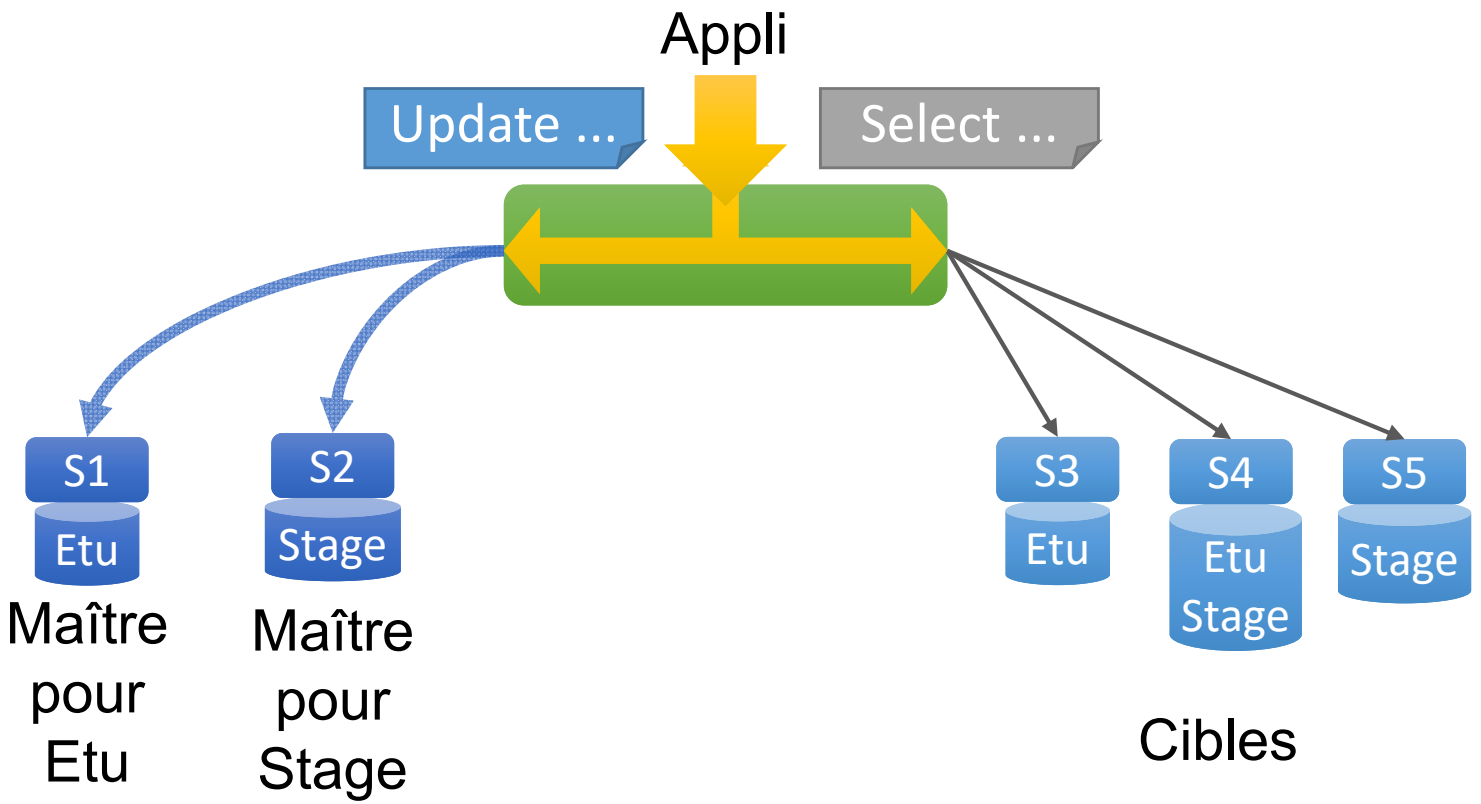
Réplication Monomaître

- Seul le site **primaire** reçoit les transactions des applications
  - insert, update, delete
- les sites **cibles** ne reçoivent que des requêtes en lecture seule
  - select



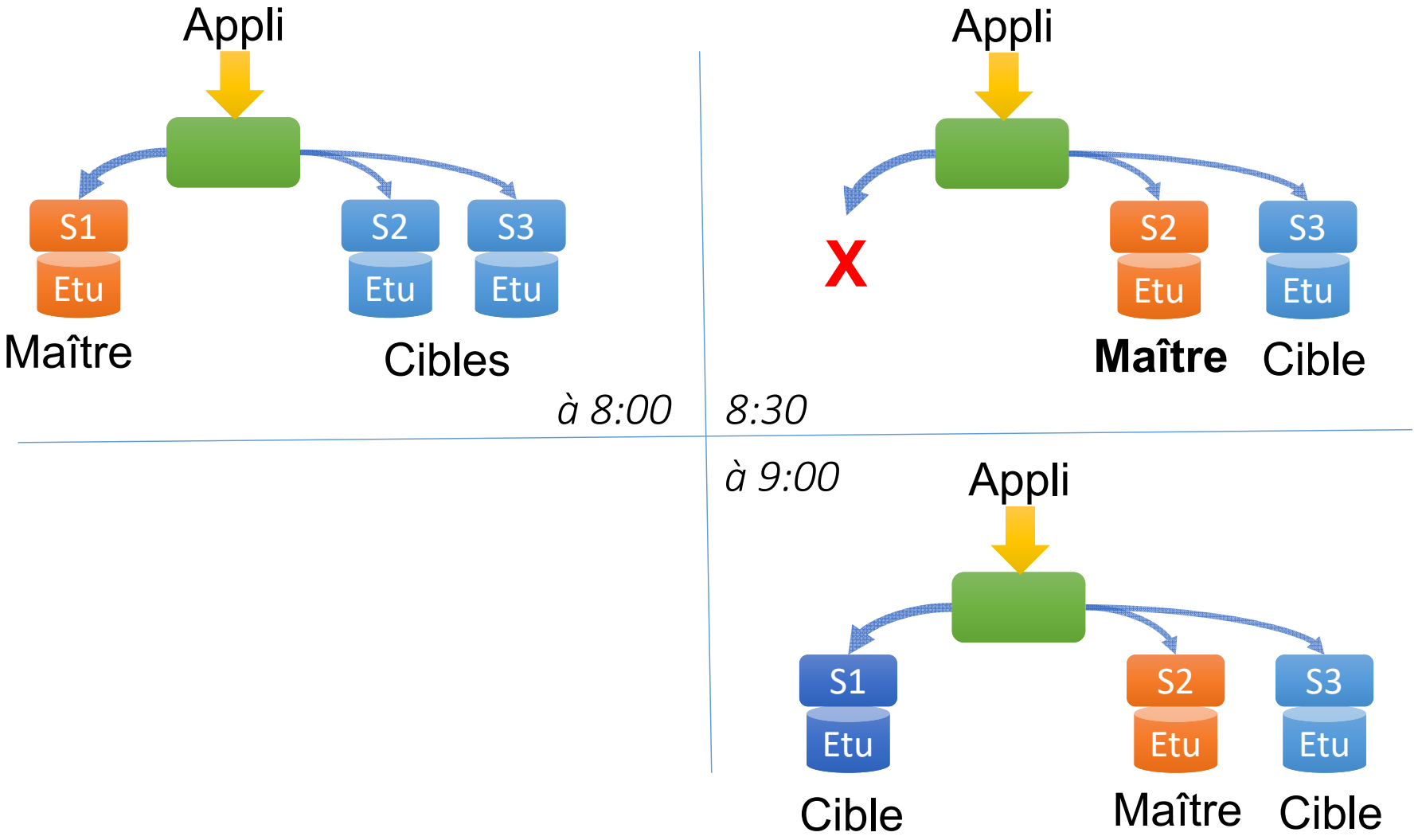
Réplication Monomaître : Consolidation

- Un maitre par table, plusieurs sites maîtres **disjoints**
- Un site cible peut gérer plusieurs tables



Monomaitre avec appartenance dynamique

Le site **primaire** peut être différent au cours du temps, en fonction d'événements: panne d'un site, état de la données, etc.



Classification des solutions

- Synchrone /Asynchrone
- Monomaître / Multimaîtres
- Vocabulaire
  - Primary copy = Monomaître
  - Update Everywhere = Multi-maître
  - Eager Replication = Repl. avec propagation synchrone
  - Lazy Replication = Repl. avec propagation asynchrone

Propagation des mises à jour

Propagation gérée par le SGBD réparti

Synchronisée
ou
Non synchronisée

avec la transaction

Propagation synchrone

- L'application obtient une réponse **APRES** la propagation
  1) transaction locale
  2) Propagation
  3) validation
  4) réponse
- Propagation immédiate après chaque instruction (update, insert, ...)
  - 1 message par instruction : interaction linéaire
  - Difficulté : chaque cible détermine l'ordre des transactions
    - Il faut garantir que l'ordre des instructions soit le même sur le maître et les cibles
  - Validation répartie entre le maître et les cibles (cf 2PC)
- Propagation différée à la fin de la transaction
  - 1 message par transaction : interaction constante
    - Contenu du message : SQL ou Log
  - Traitement sur les cibles selon l'ordre déterminé par le maître

Propagation asynchrone

- L'application obtient une réponse **AVANT** la propagation

1. transaction locale
2. validation locale
3. réponse

4. propagation
5. validation

Etapes (1,2,3) indépendantes de (4,5)

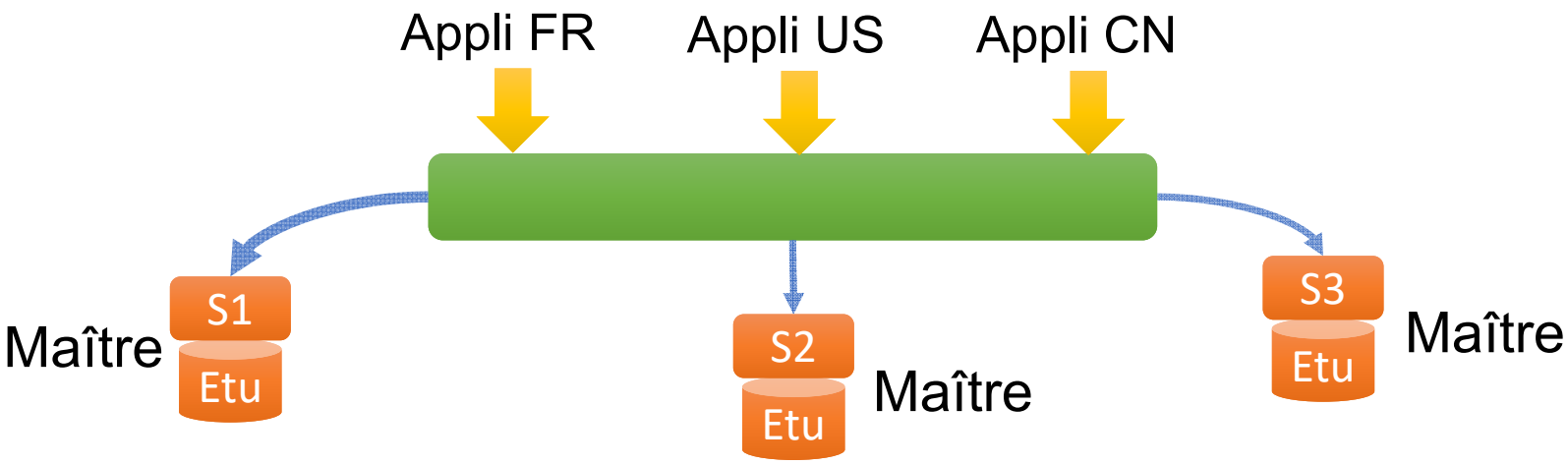
Détection des mises à jour à propager

- Solution 1 : utilisation du journal appelé *log*
  - les transactions qui modifient écrivent une marque spéciale dans le journal
  - détection périodique en lisant le journal, indépendamment de la transaction qui a modifié
  - Inconvénient : modifier la gestion du journal

- Solution 2 : utilisation de triggers
  - la modification d'une donnée répliquée déclenche un trigger
  - mécanisme général et extensible
  - la détection fait partie de la transaction et peut la ralentir.

Réplication Multimaîtres (1/2)

- Plusieurs maîtres pour une donnée
  - Transactions posées sur Etu@S_1 ou Etu@S_2 ou Etu@S_3
- Augmente la disponibilité
  - Intéressant si plusieurs transactions accèdent simultanément au même fragment mais modifient des données disjointes.
- Intéressant pour la géo-réplication : utilisateurs sur plusieurs continents
  - Choix du maitre le plus proche : distance topologique ou géographique



Réplication multi-maîtres (2/2)

Propagation des mises à jour ?

- Les mises à jour à propager viennent de plusieurs maîtres
- Une réplique doit recevoir toutes les mises à jour reçues par ses maîtres

- Optimiste = conflits possibles
  - détecter les conflits et les résoudre
  - Intéressant si conflits rares et résolution automatique
- Préventif = empêcher les conflits
  - Détecter *a priori* les conflits potentiels
  - Nécessite de connaitre les données lues et/ou écrite par une transaction avant de la traiter
  - Déterminer un ordre global des transactions
    - Inconvénient surcoût de communication entre les maitres

Etude de cas : Amazon Dynamo DB

Dynamo : Stockage

- Partitionnement des items par hachage
- Hachage consistent :
  - Combine du partitionnement par hachage et par intervalle
    - Différent du hachage extensible
  - Une seule fonction de hachage h(x) : String → D
    - D est l'ensemble des nombres entiers dans [0, 2^{64}[
    - D est découpé n segments : {d1, d2, ...dn}
      - Segment S1 : [d1, d2[
      - Segment Sn: [dn, 2^{64}[union [0, d1[
      - segmentation "circulaire" facilite l'ajout d'un segment:
        - tout segment a un successeur et un prédécesseur
    - h(key) → nombre dans D → n° de segment → machine gérant la partition

Gestion des partitions

- Table de Routage
  - Table associant un segment à une machine: Routage global
  - Info décentralisée si plusieurs millions de machines (*cf.* P2P)

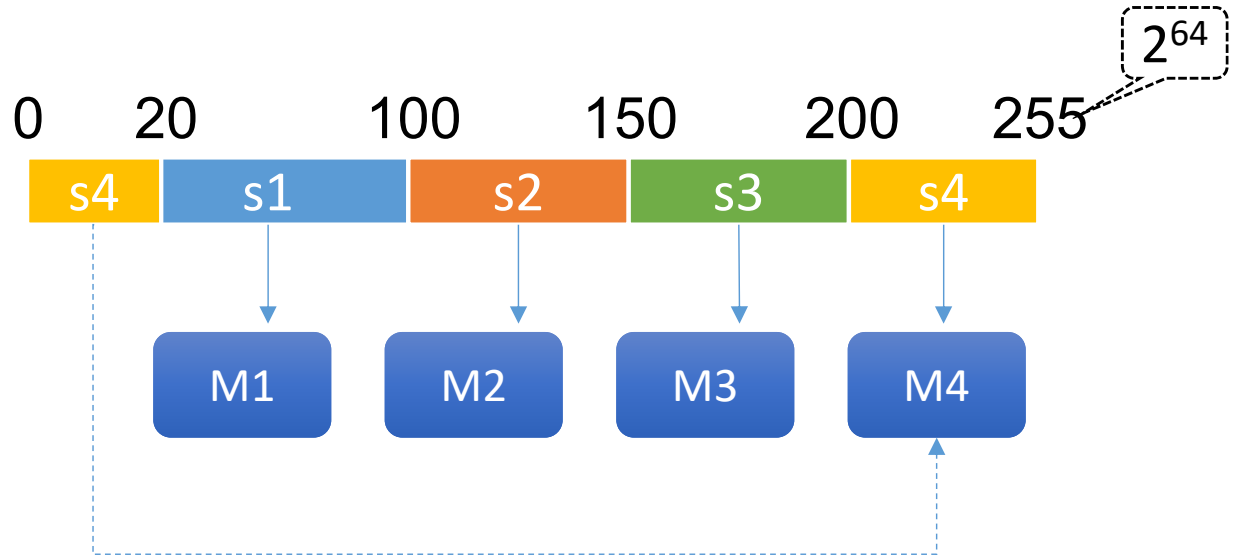


Table de routage

Di	Machine
20	M4
100	M1
150	M2
200	M3

Exemple de routage :
10 → M_4 23 → M_4
248 → M_4

Partitions : extension (1)

- Si une partition devient **trop remplie**
  - Scinder en 2 partitions
  - pas de surcoût sur les autres partitions

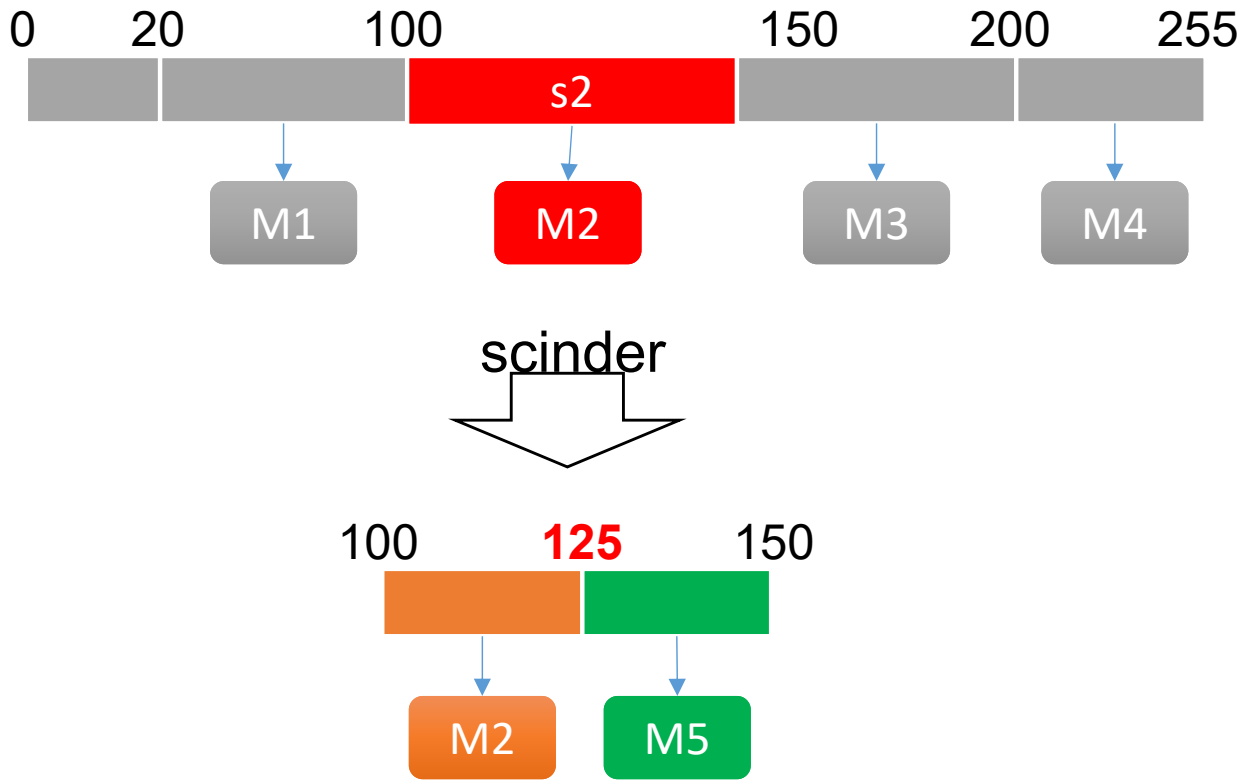


Table de routage

Di	Machine
20	M4
100	M1
125	**M5**
150	M2
200	M3

Partitions : extension (2)

- Si une partition devient **trop remplie**
  - Redistribuer seulement avec les deux voisines
  - Faible surcoût

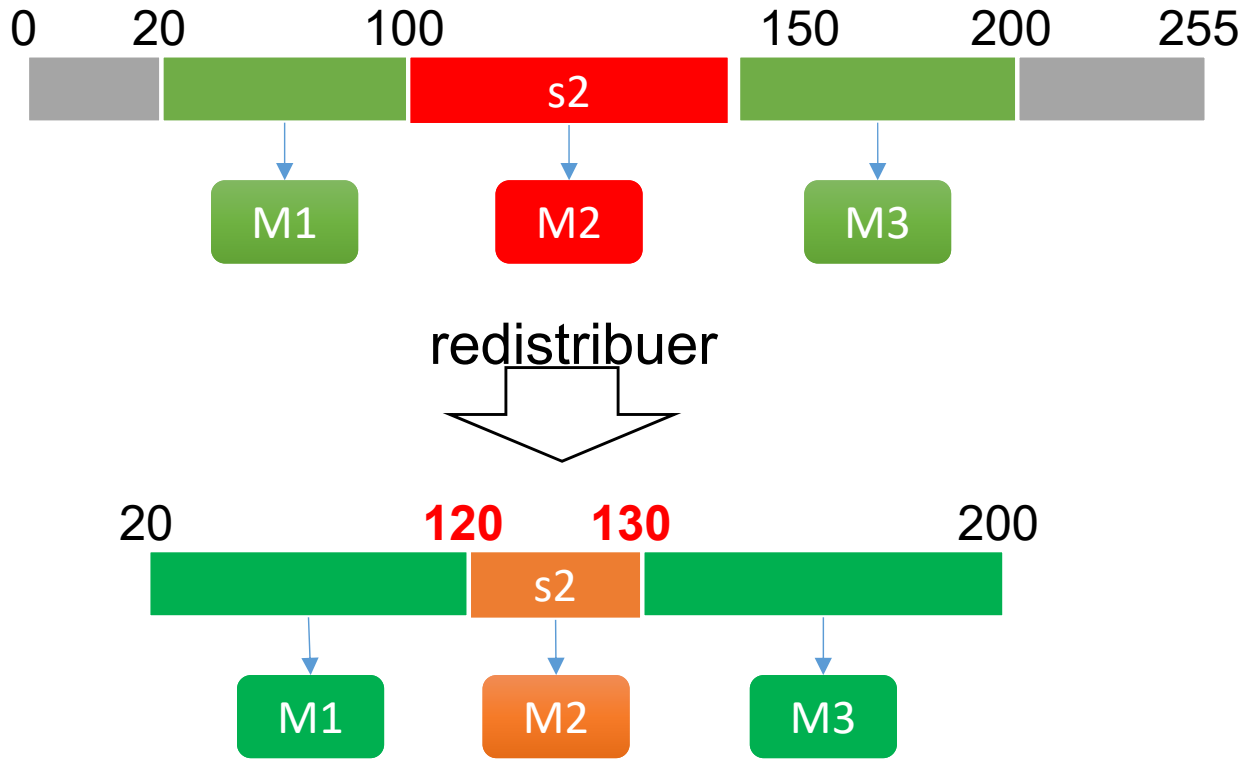
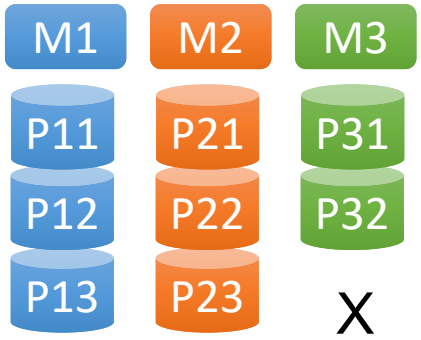


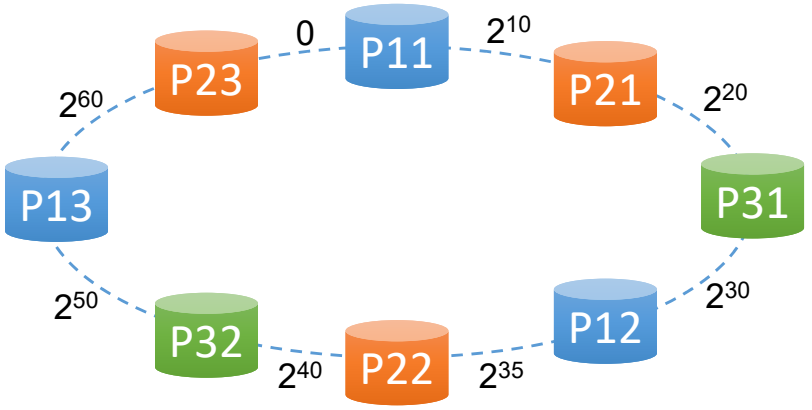
Table de routage

Di	Machine
20	M4
120	M1
130	M2
200	M3

Dynamo : Hétérogénéité

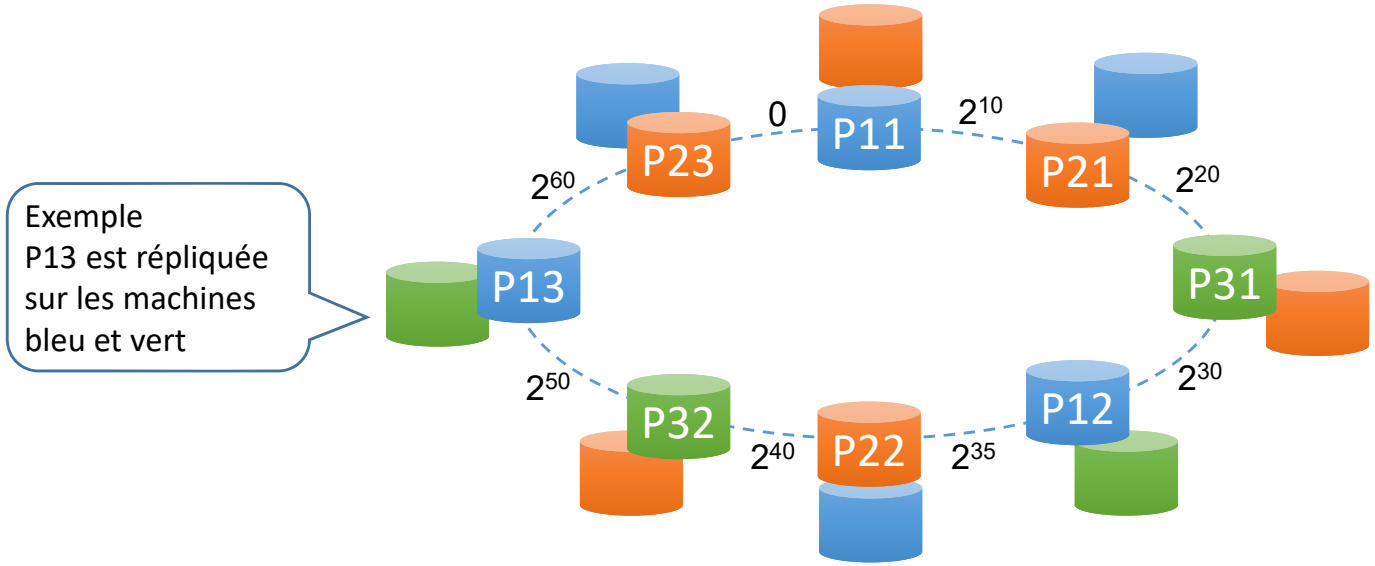
- Plusieurs partitions par machine
  - Partition "virtuelle" : une machine peut gérer plusieurs partitions
- Gérer des machines hétérogènes
  - Machine puissantes gèrent plus de partitions
- Répartir les partitions d'une machine à travers le domaine D
  - Extensibilité préservée si 2 partitions *voisines* sont sur des machines différentes





Dynamo : Tolérance aux pannes

- Répliquer une partition sur plusieurs machines
  - Degré de réplication selon la tolérance aux pannes souhaitée (r = 3)
- Pour une machine ayant N partitions : jusqu'à N * r machines stockant une réplique
  - En cas de panne : surcoût amorti par davantage de machines
  - Restauration plus rapide car transfert des répliques depuis N machines



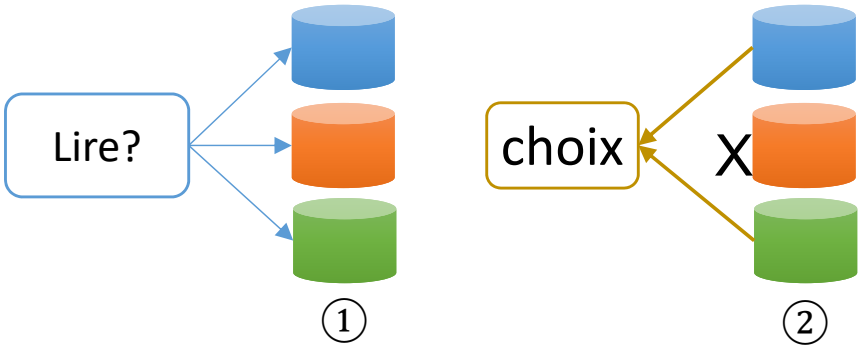
Dynamo : Cohérence des répliques

- Maintien des répliques cohérentes
- **Cohérence en lecture : 2 options possibles**
  - Cohérence à terme (*eventual*)
    - Lire une seule réplique
    - La valeur lue peut être "ancienne" et différer de la valeur courante

  ou
  - Cohérence forte
    - Lire une majorité des répliques

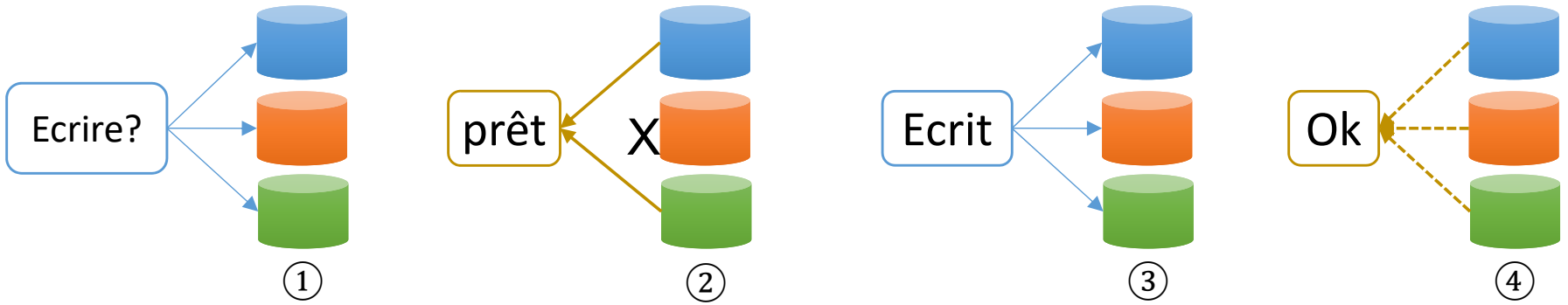
Dynamo : Lecture d'une donnée répliquée

- Donnée : clé →(valeur, version).
  - Exple A→("Alice",1)
- A est répliquée sur (M2, ..., Mk)
- M1 veut **lire** la dernière version de la donnée A
  - M1 envoie aux répliques : "Quelle est la dernière version de A ?"
    - Une réplique répond : "mon A courant est (valeur, version)"
  - M1 reçoit les versions et **choisit** celle retournée par une **majorité** de répliques



Dynamo : Ecriture d'une donnée répliquée (1/3)

- **Ecriture** : protocole décentralisé, inspiré de Paxos
  - Nécessite 2 messages envoyés à toutes les répliques
  - Confirmation d'une majorité (suffisant)



Dynamo : Ecriture d'une donnée répliquée (2/3)

- Données clé →(valeur, version).
- M1 veut écrire une nouvelle version i de A.
  - M1 **propose** aux répliques le n° de version i pour A
    - Une réplique contient la version c de A (c≠i).
    - Si i > c alors réponse "**accepte i**", sinon réponse "décline à cause de c"
  - M1 reçoit les acceptations : si une majorité des répliques accepte i, alors M1 envoie A(v2, i) aux répliques
    - Les répliques accusent réception de l'écriture
- Si plusieurs machines veulent écrire A
  - Chacune doit proposer des nouveaux n° de version différents
  - Risque d'interblocage "actifs" (live lock)
- Sinon M1 a le rôle de leader pour A
  - Nécessite d'élire un leader, faible surcoût

Dynamo : Ecriture d'une donnée répliquée (3/3)

- Support très limité des transactions
  - Limité à la mise à jour **d'une seule** donnée
- Problème : traiter deux opérations en série
  - Cas du "test and set" : une lecture suivie d'une écriture
- Protocole décentralisé
  - Lecture en 1 aller-retour
    - Envoyer "prépare", puis confirmation par une majorité
    - Lire la donnée
  - Ecriture en 2 allers-retours
    - Envoyer "propose", puis confirmation
    - Envoyer "validation" et confirmation

Conclusions et perspectives

Applications
- Décisionnelles, analyse + Transactionnelles

Réplication asynchrone avec délai le plus court possible

Applications à large échelle
- grand nombre de sources
- géo réplication de plus en plus présente

Réplication dans les systèmes NewSQL
- Plus de flexibilité pour choisir le mode de réplication

Une solution de l'état de l'art

- Decongestant: A Breath of Fresh Air for MongoDB Through Freshness-aware Reads,
  - EDBT 2021, https://edbt2021proceedings.github.io/docs/p135.pdf
  - Huang, Cahil, Kekete, Rohm, University of Sydney, Australia

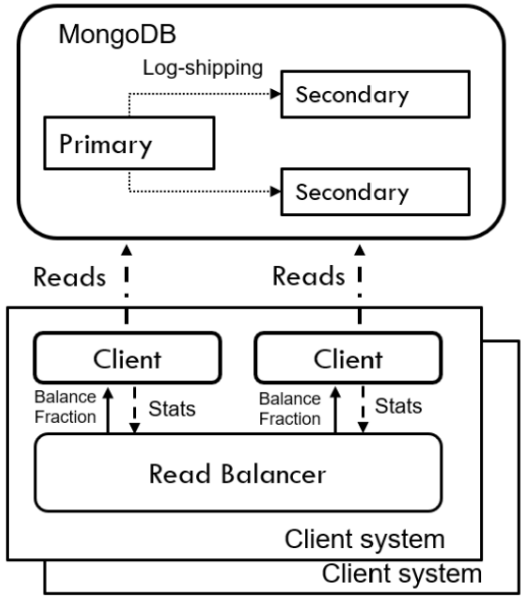


Figure 1: A simplified architecture of Decongestant

Choix automatique du site maitre ou cible sur lequel traiter une requête.
Equilibrage de charge dynamique: tient compte de la charge courante des sites.