# RelaxDHT: a churn-resilient replication strategy for peer-to-peer distributed hash-tables

Sergey Legtchenko[1] Sébastien Monnet[1] Pierre Sens[1] Gilles Muller[2]

LIP6/University of Paris VI/CNRS/INRIA[1]

EMN/INRIA[2]

# Distributed data storage

Context :

- Large-scale data storage
- Data replication (availability)

Problem :

- **Churn** : frequent connections and disconnecitons
- => Replicated data may be lost

Goal :

- Enhance churn resilience to enhance data availability

# Outline

# Peer-to-peer data storage

Storing data-blocks among a very large number of nodes

Constraint :

- Nodes may leave the P2P overlay at any time

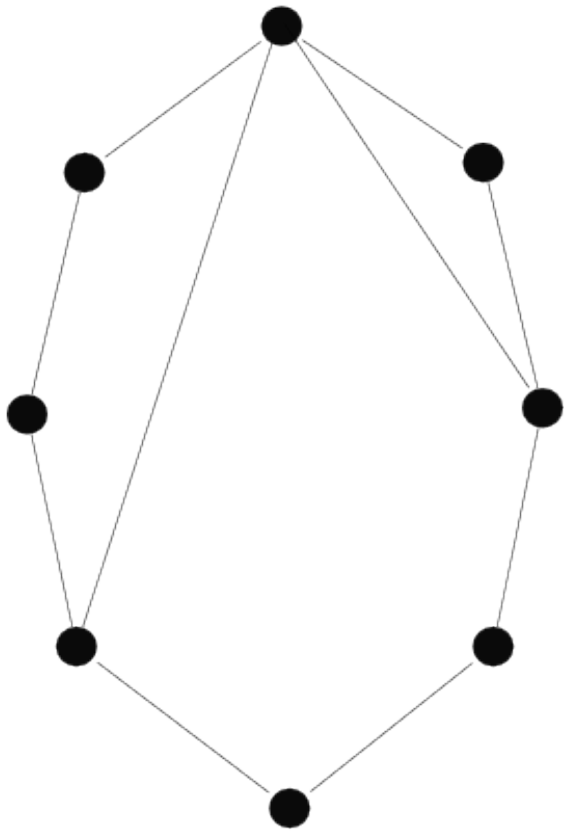
Solution :

- Data blocks are replicated on several host nodes

# Distributed hash tables (DHT)

**Logical ring**

**PAST**, DHASH...

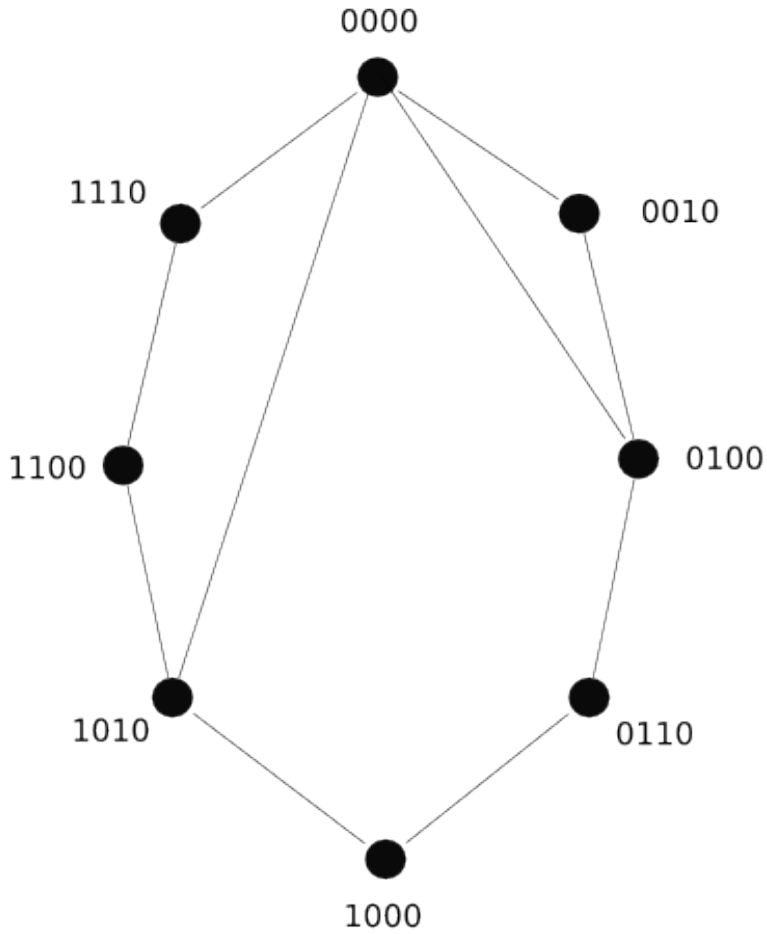


# Distributed hash tables (DHT)

## Logical ring

**PAST**, DHASH...

- Each node as an identifier

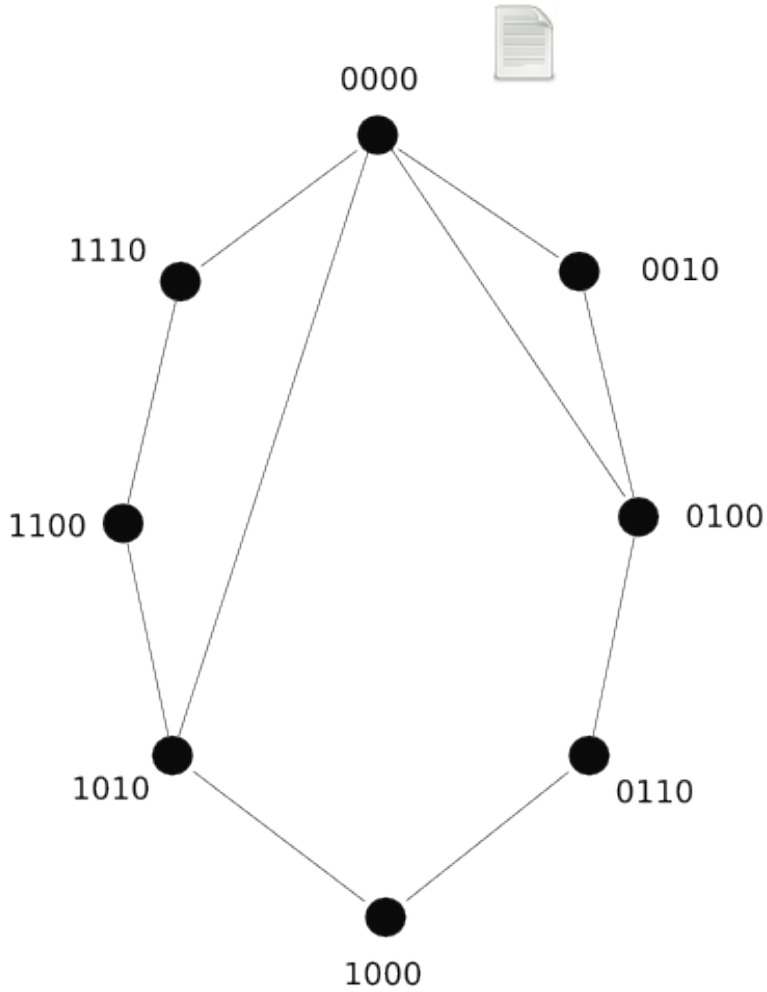


# Distributed hash tables (DHT)

## Logical ring

**PAST**, DHASH...

- Each node as an identifier

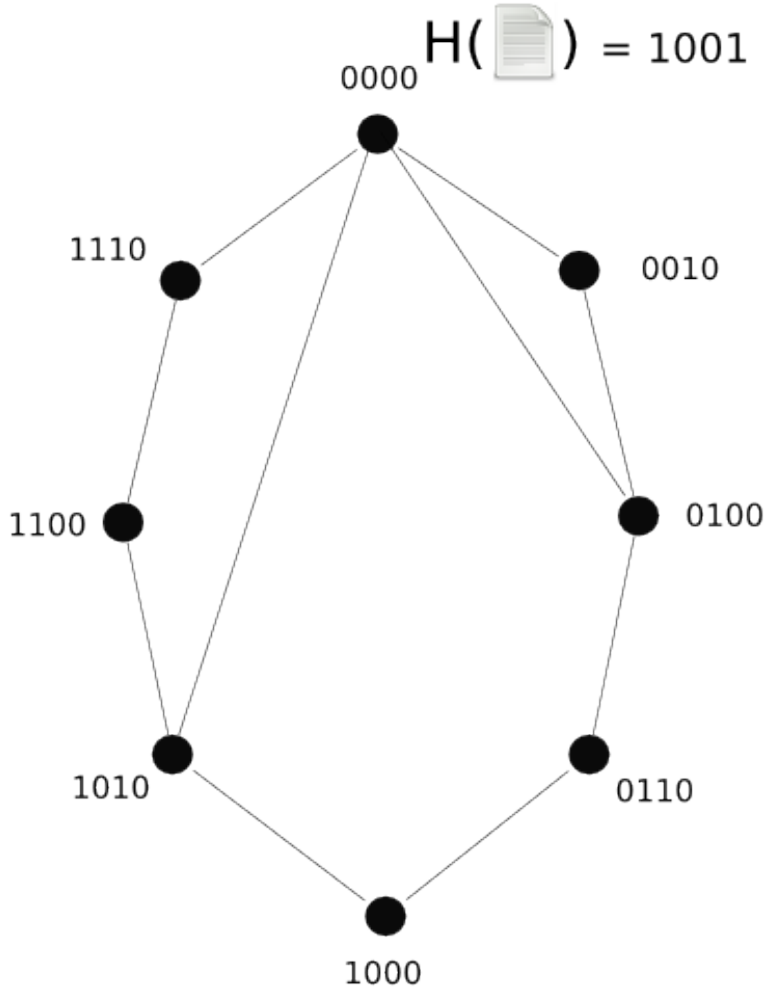


# Distributed hash tables (DHT)

## Logical ring

**PAST**, DHASH...

- Each node as an identifier
- Each data as an identifier (usualy using a hash function)

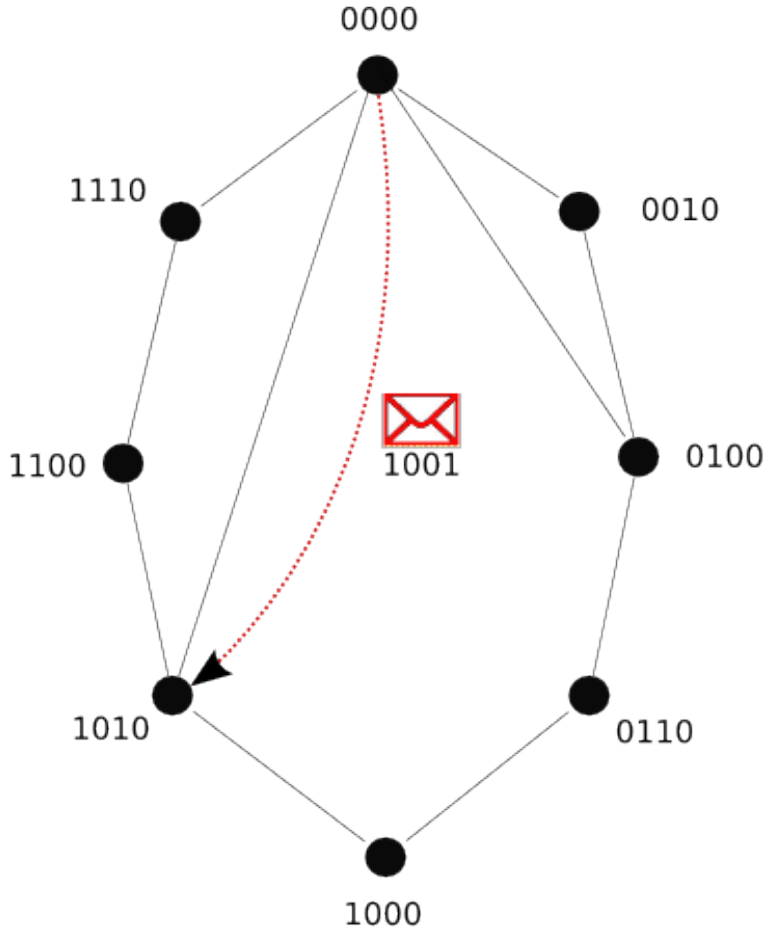


# Distributed hash tables (DHT)

## Logical ring

**PAST**, DHASH...

- Each node as an identifier
- Each data as an identifier (usualy using a hash function)
- The node which identifier is the closest is called the "root peer"

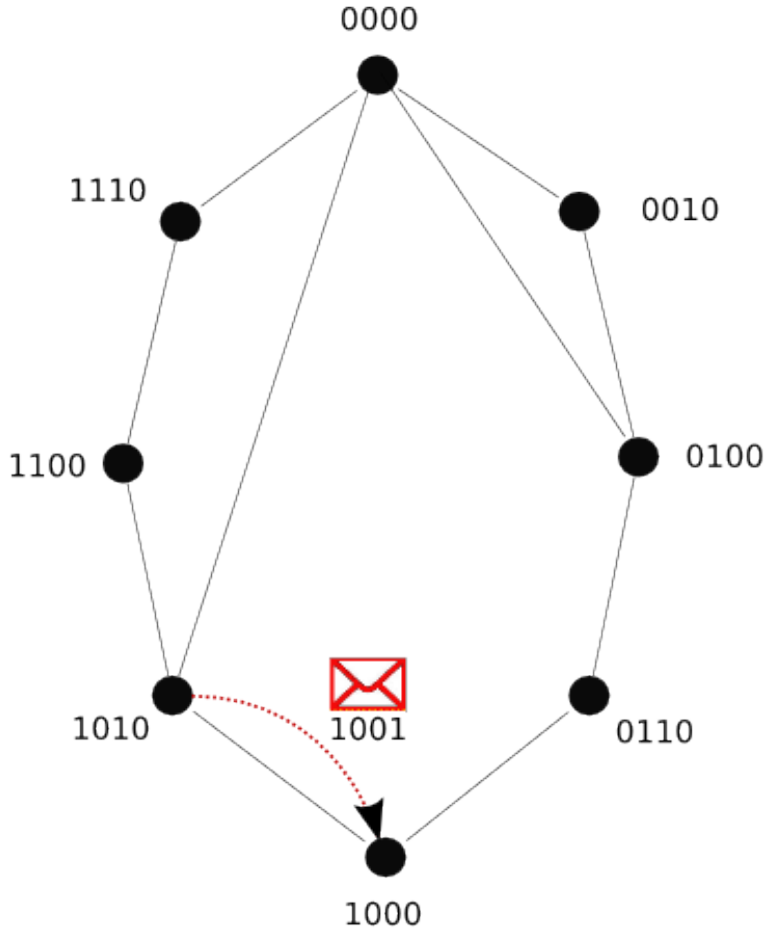


# Distributed hash tables (DHT)

## Logical ring

**PAST**, DHASH...

- Each node as an identifier
- Each data as an identifier (usualy using a hash function)
- The node which identifier is the closest is called the "root peer"

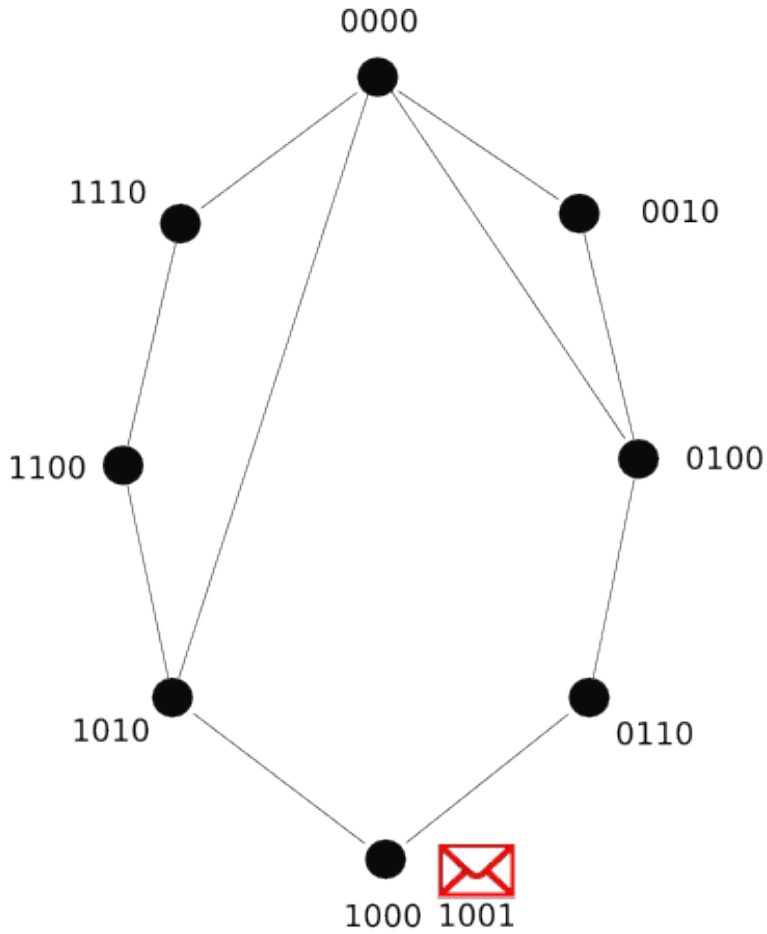


# Distributed hash tables (DHT)

## Logical ring

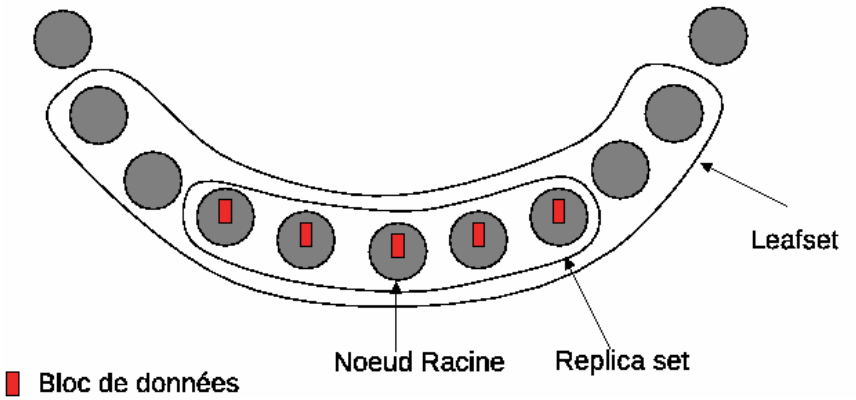
**PAST**, DHASH...

- Each node as an identifier
- Each data as an identifier (usualy using a hash function)
- The node which identifier is the closest is called the "root peer"



# Leafset-based replication strategies

Leafset : logical neighborhood in the ring
$k$ copies are stored contiguously on the root's direct neighbors

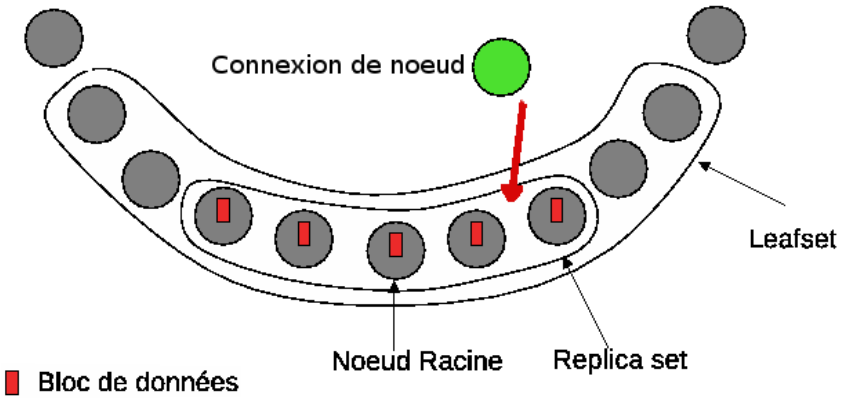



**Drawbacks :**

- Replica placement constraints are strong
- Churn => many data movements

# Leafset-based replication strategies

Leafset : logical neighborhood in the ring
$k$ copies are stored contiguously on the root's direct neighbors

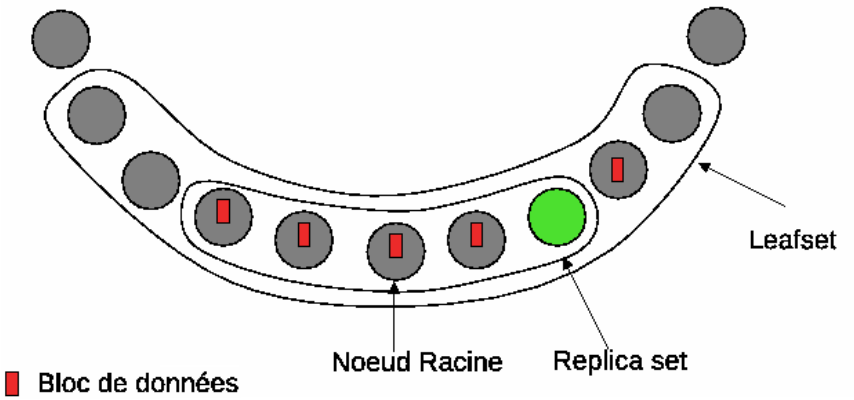



**Drawbacks :**

- Replica placement constraints are strong
- Churn => many data movements

# Leafset-based replication strategies

Leafset : logical neighborhood in the ring
$k$ copies are stored contiguously on the root's direct neighbors

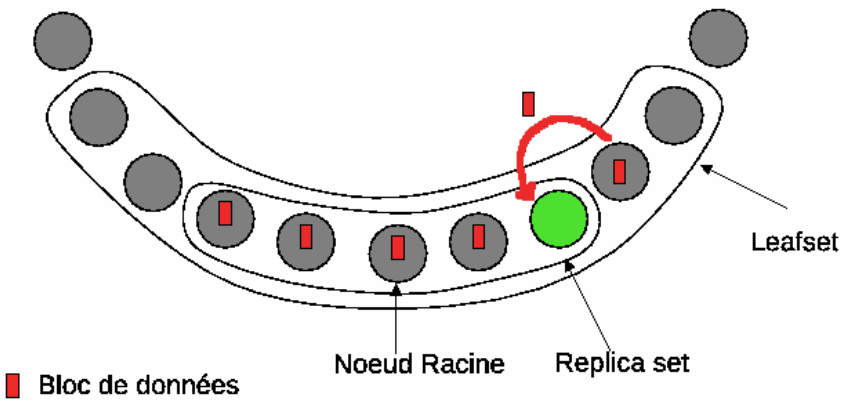



**Drawbacks :**

- Replica placement constraints are strong
- Churn => many data movements

# Leafset-based replication strategies

Leafset : logical neighborhood in the ring
$k$ copies are stored contiguously on the root's direct neighbors

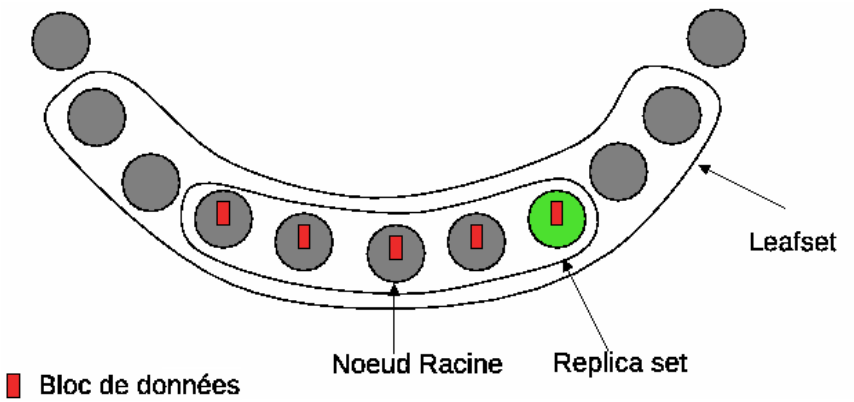



**Drawbacks :**

- Replica placement constraints are strong
- Churn => many data movements

# Leafset-based replication strategies

Leafset : logical neighborhood in the ring
$k$ copies are stored contiguously on the root's direct neighbors

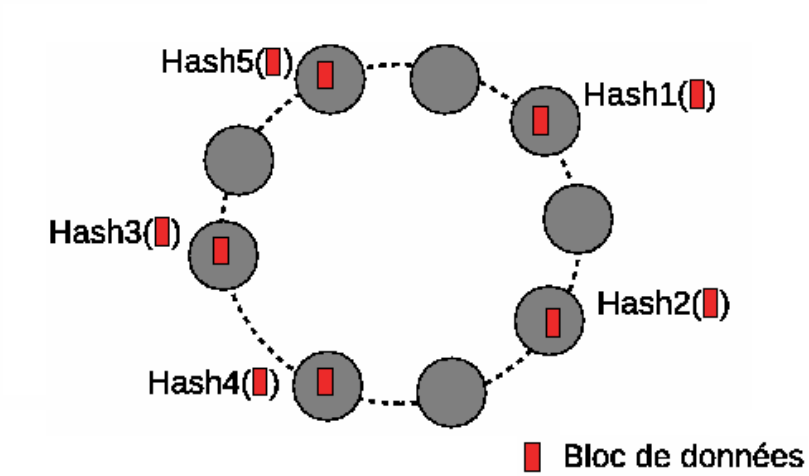



**Drawbacks :**

- Replica placement constraints are strong
- Churn => many data movements

# Multiple key-based replacation strategies

Each data block is assigned multiple identifiers
=> Multiple root peers




**Défauts :**

- Maintenance cost is directly linked to the number of data-blocks
- => does not scale in terms of blocks...

# Plan

# Relaxing placement constraints

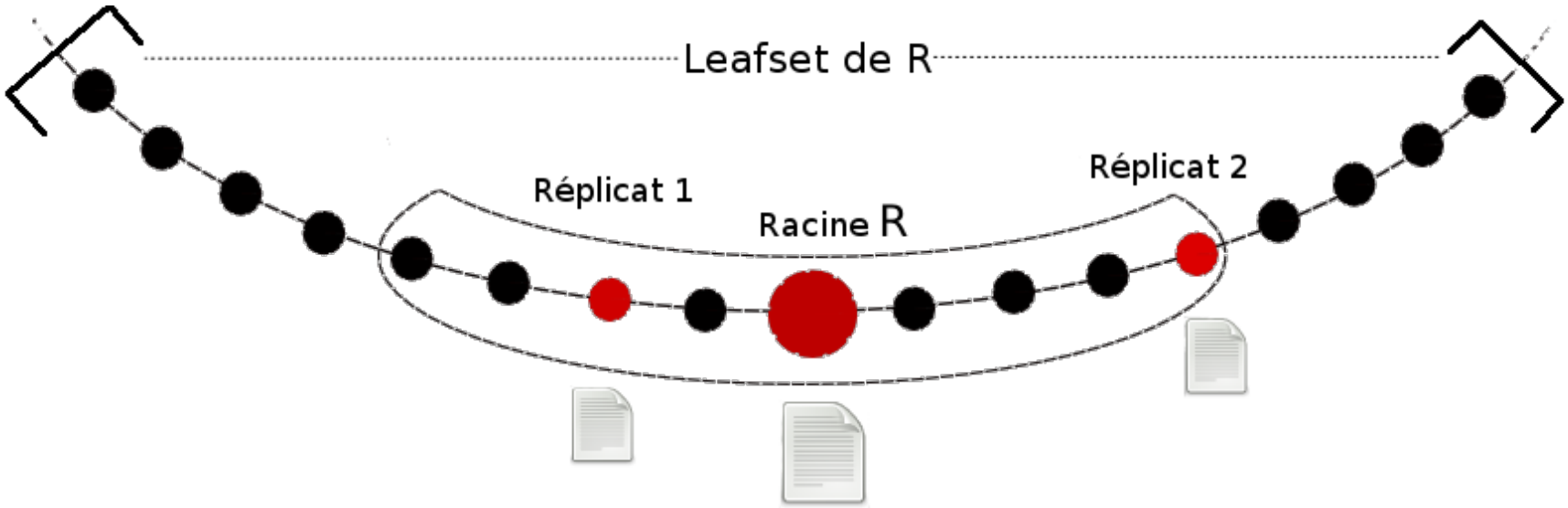



FIGURE: *RelaxDHT principle : leafset-based replication with relaxed placement constraints.*

# Data-block insertion

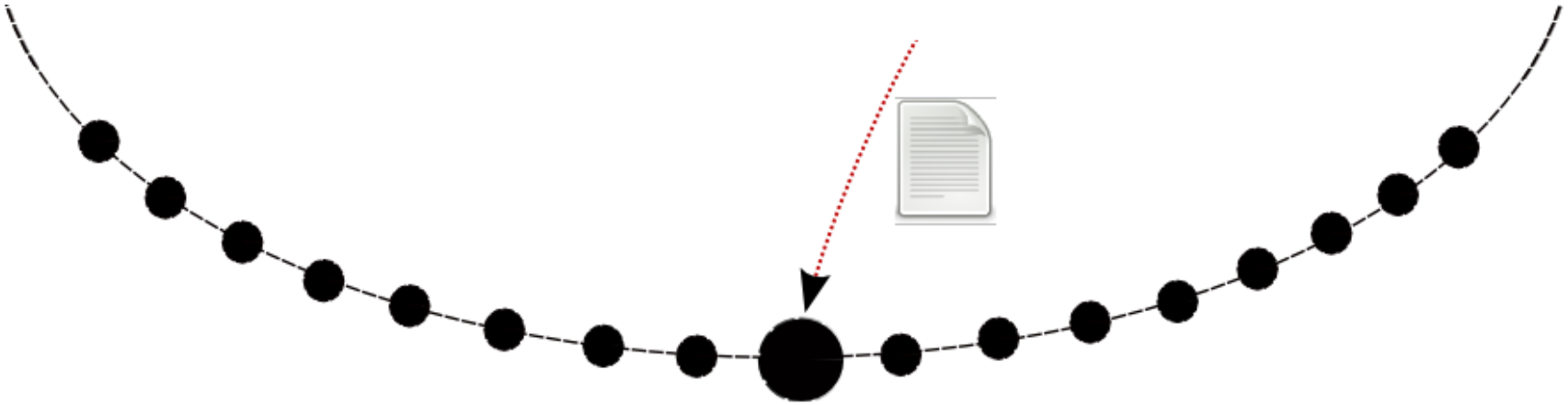


FIGURE: *Choix des réplicats par la racine lors de l'insertion d'un nouveau bloc.*

# Data-block insertion

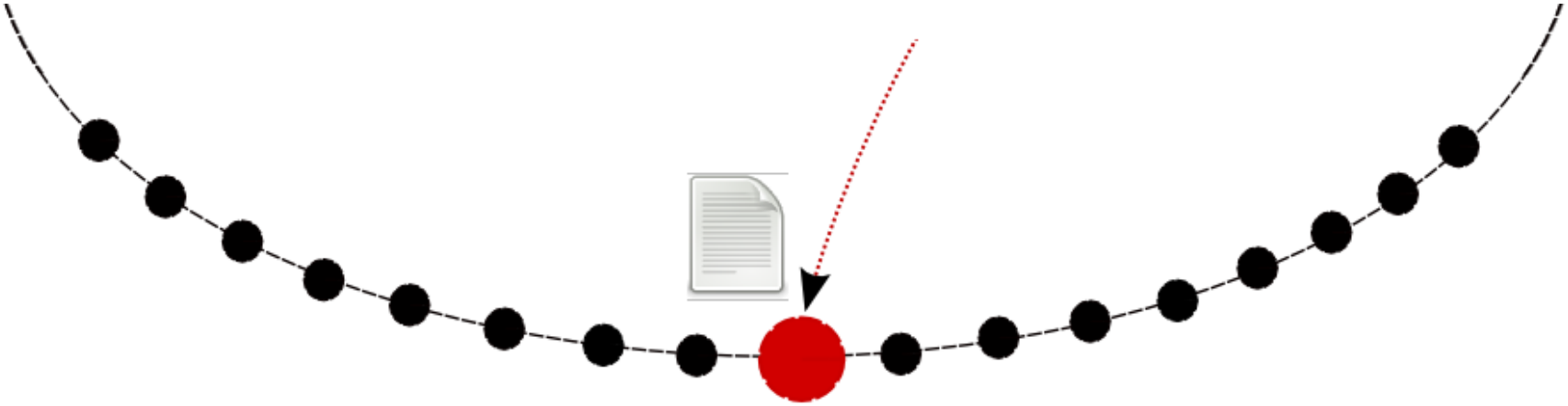


FIGURE: *Choix des réplicats par la racine lors de l'insertion d'un nouveau bloc.*

# Data-block insertion

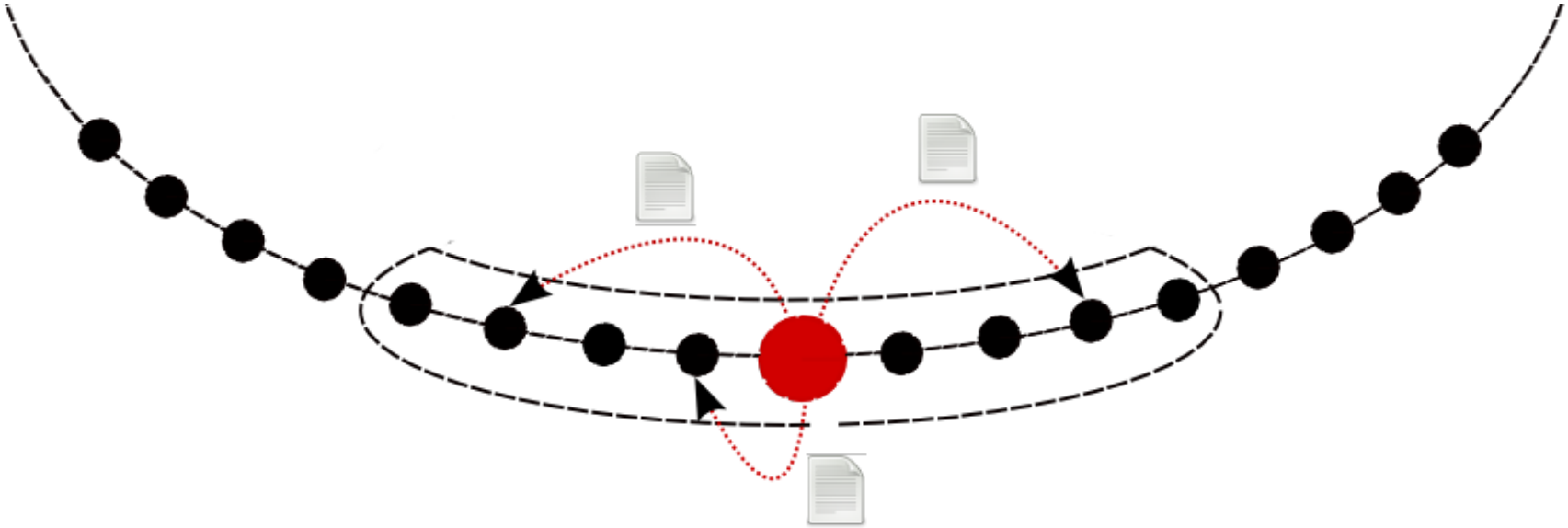


FIGURE: *Choix des réplicats par la racine lors de l'insertion d'un nouveau bloc.*

# Data-block insertion

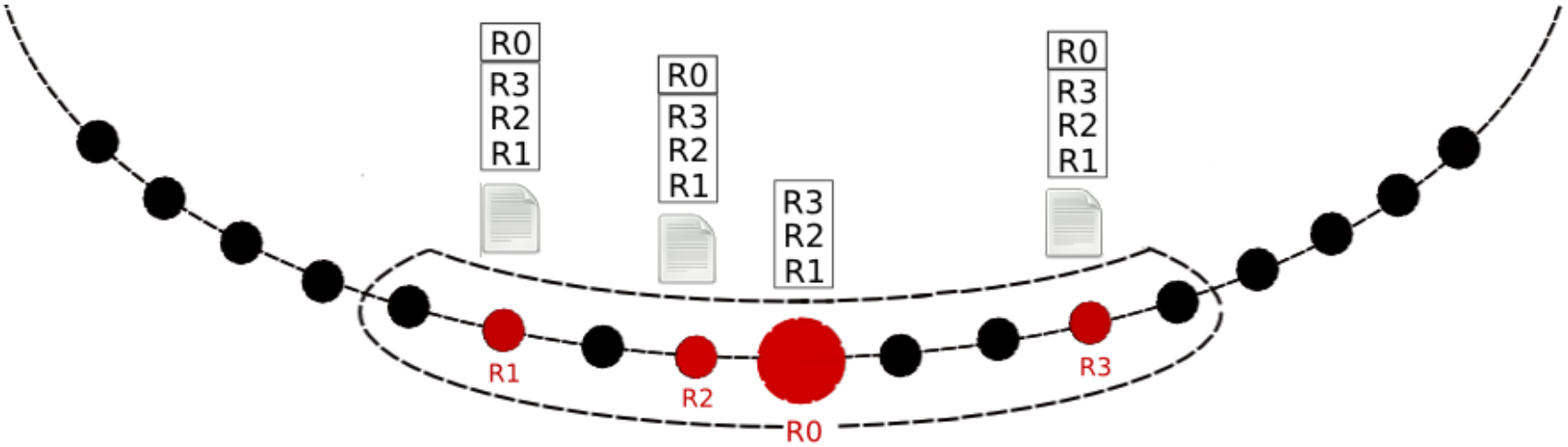



FIGURE: *Choix des réplicats par la racine lors de l'insertion d'un nouveau bloc.*

# Protocole de maintenance

- Un réplicat de bloc se voit associer un bail sur le nœud-réplicat.
- À chaque periode de maintenance, la racine envoie un message de renouvellement de bail.

Si le bail expire, le réplicat est supprimé.

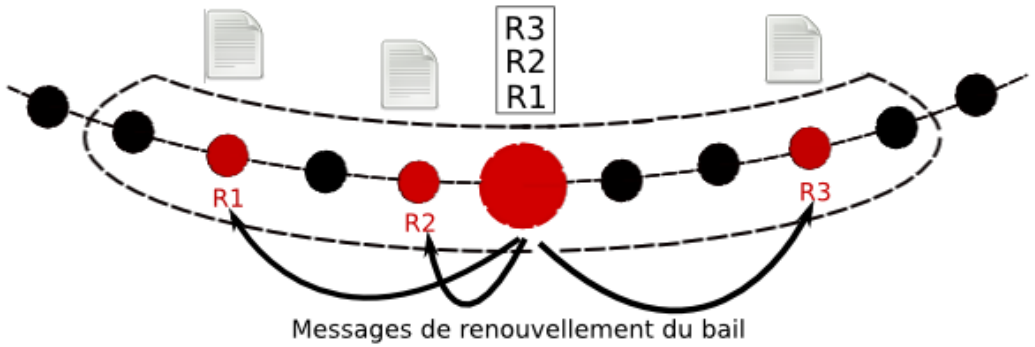
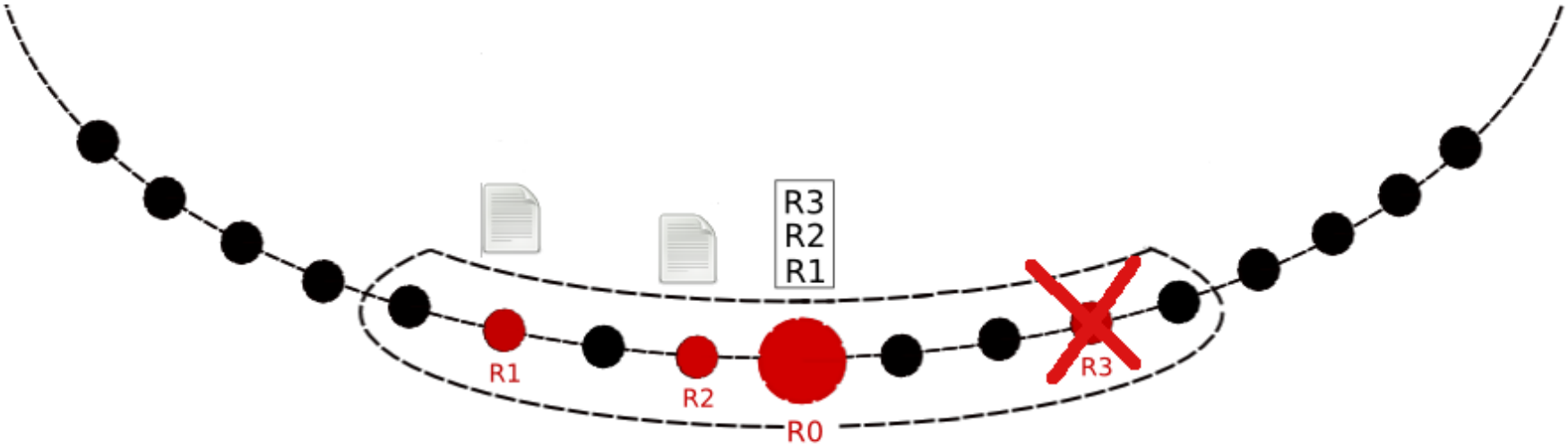



FIGURE: *Envoi de messages de maintenance aux réplicats*

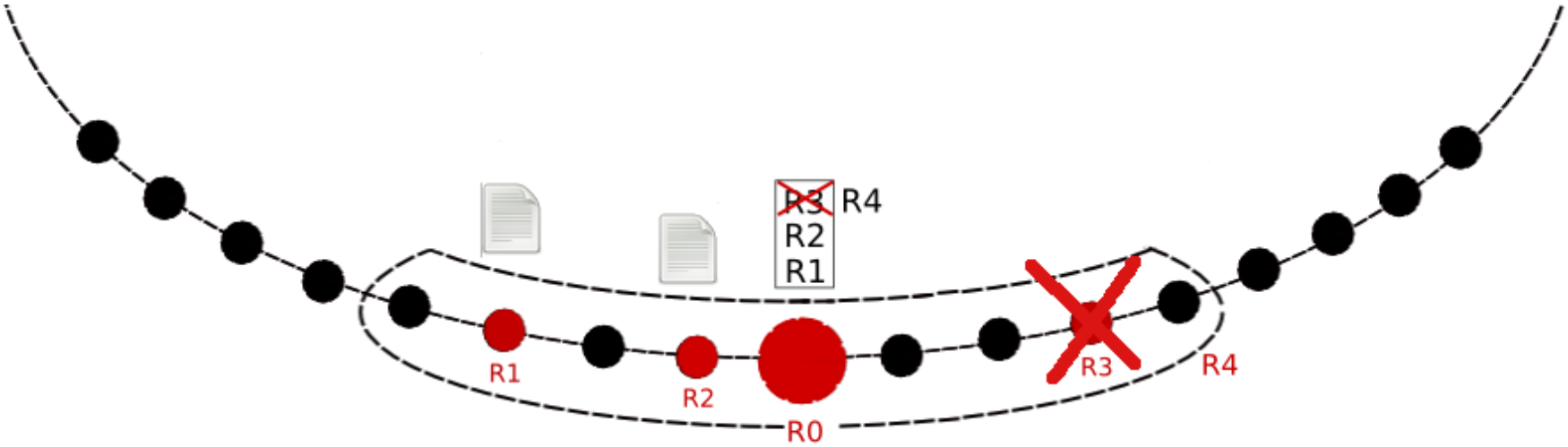
# Choix d'un nouveau nœud réplicat par la racine

En cas de perte ou d'éloignement excessif d'un nœud-réplicat :



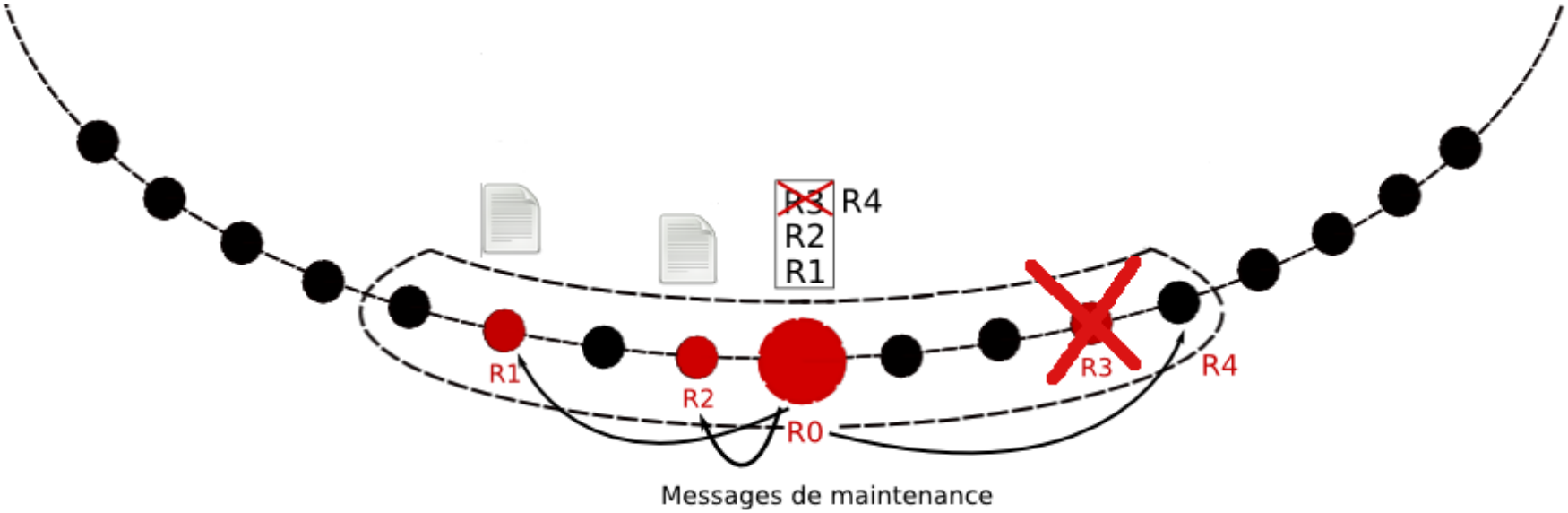
# Choix d'un nouveau nœud réplicat par la racine

En cas de perte ou d'éloignement excessif d'un nœud-réplicat :



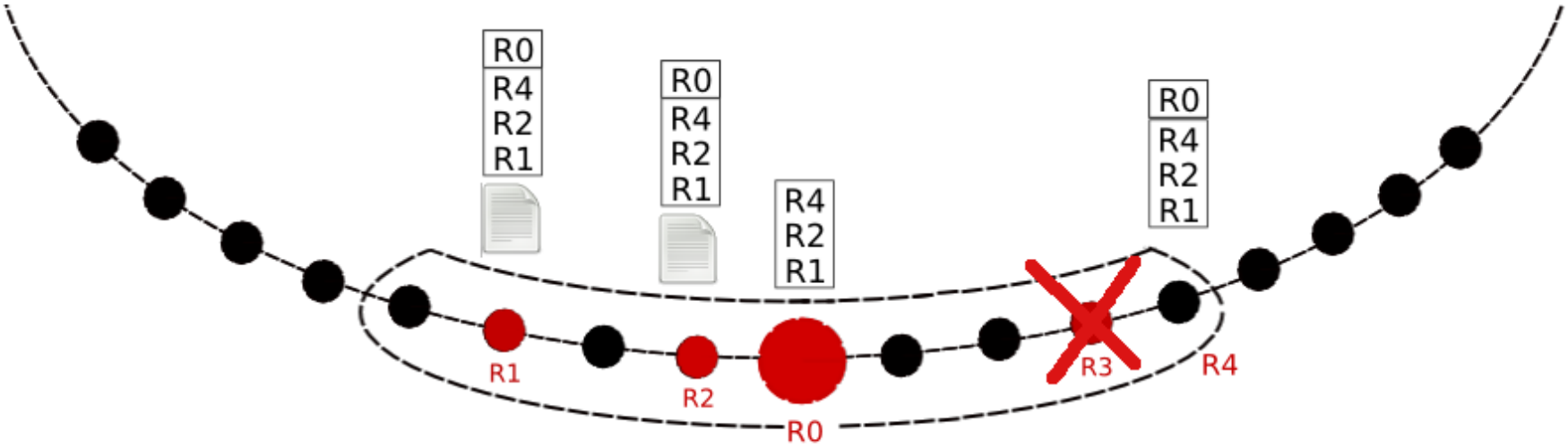
# Choix d'un nouveau nœud réplicat par la racine

En cas de perte ou d'éloignement excessif d'un nœud-réplicat :



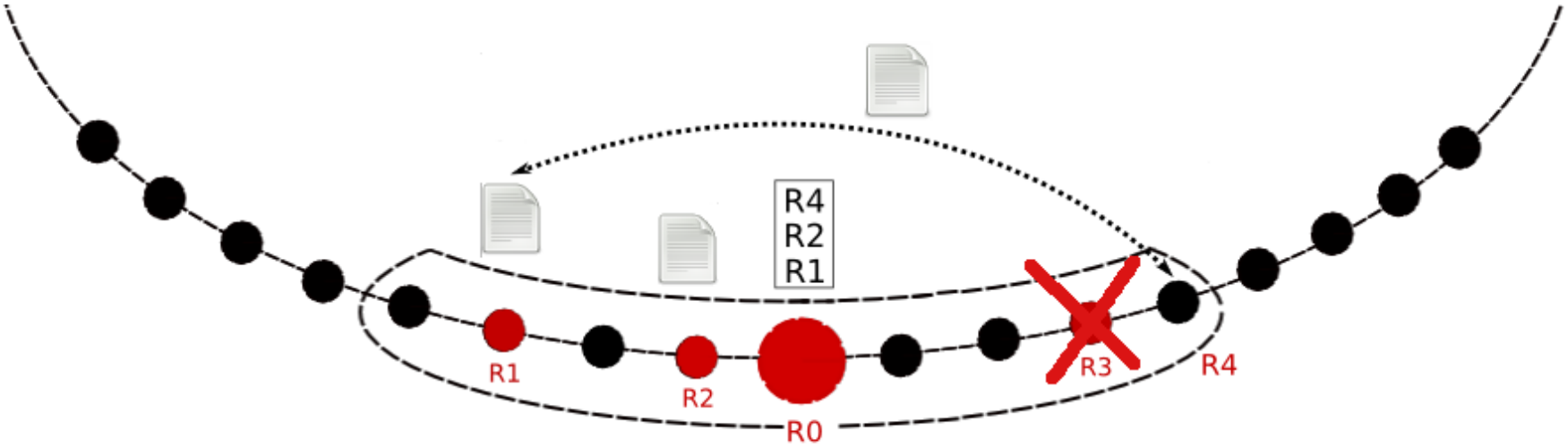
# Choix d'un nouveau nœud réplicat par la racine

En cas de perte ou d'éloignement excessif d'un nœud-réplicat :



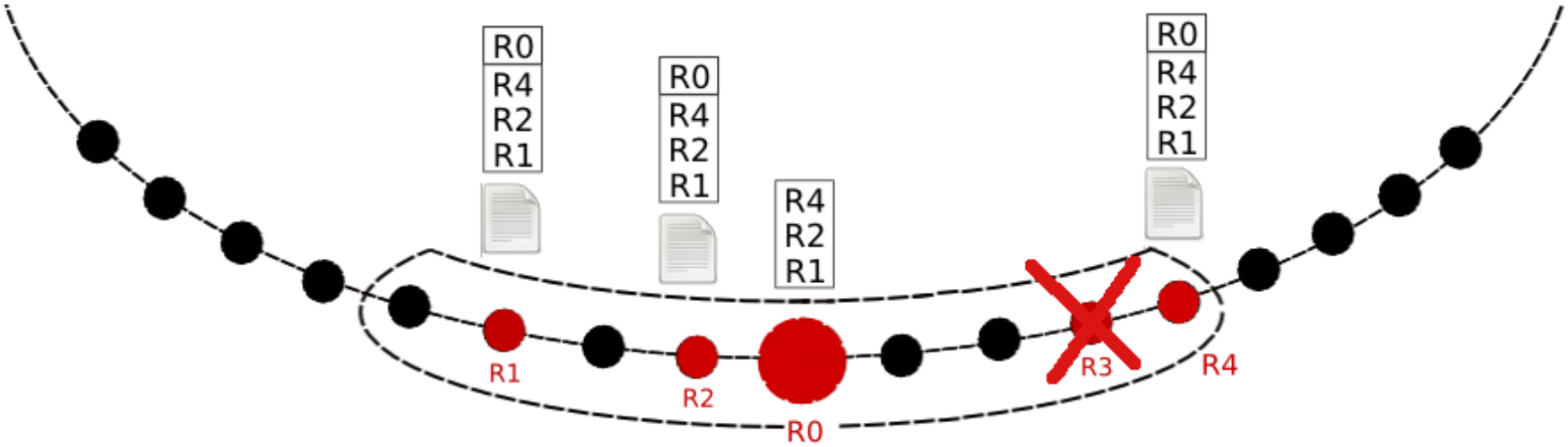
# Choix d'un nouveau nœud réplicat par la racine

En cas de perte ou d'éloignement excessif d'un nœud-réplicat :



# Choix d'un nouveau nœud réplicat par la racine

En cas de perte ou d'éloignement excessif d'un nœud-réplicat :



# En cas de perte de la racine

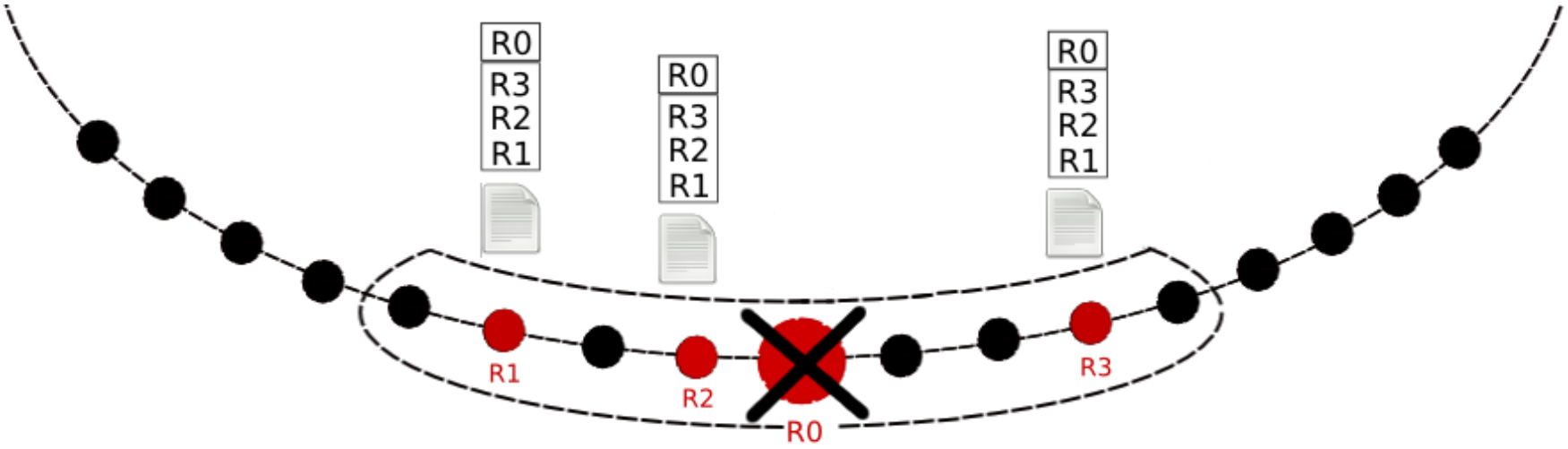



FIGURE: *Lors de la maintenance, une nouvelle racine est désignée par un ou plusieurs nœud réplicats.*

# En cas de perte de la racine

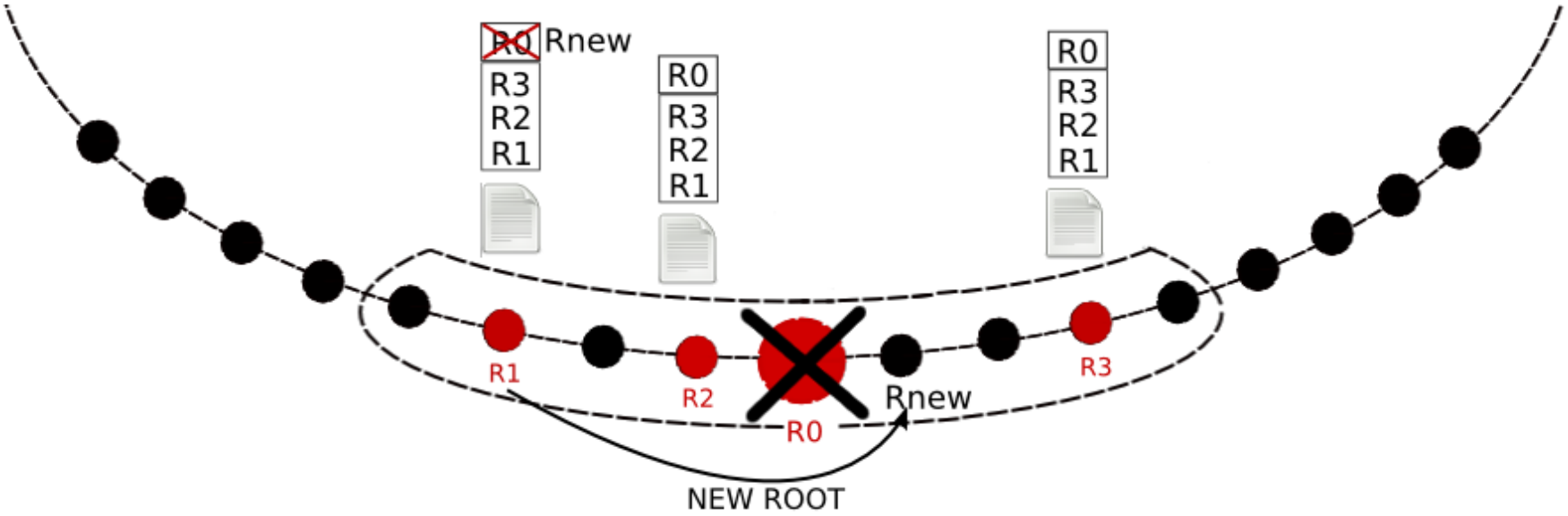



FIGURE: *Lors de la maintenance, une nouvelle racine est désignée par un ou plusieurs nœud réplicats.*

# En cas de perte de la racine

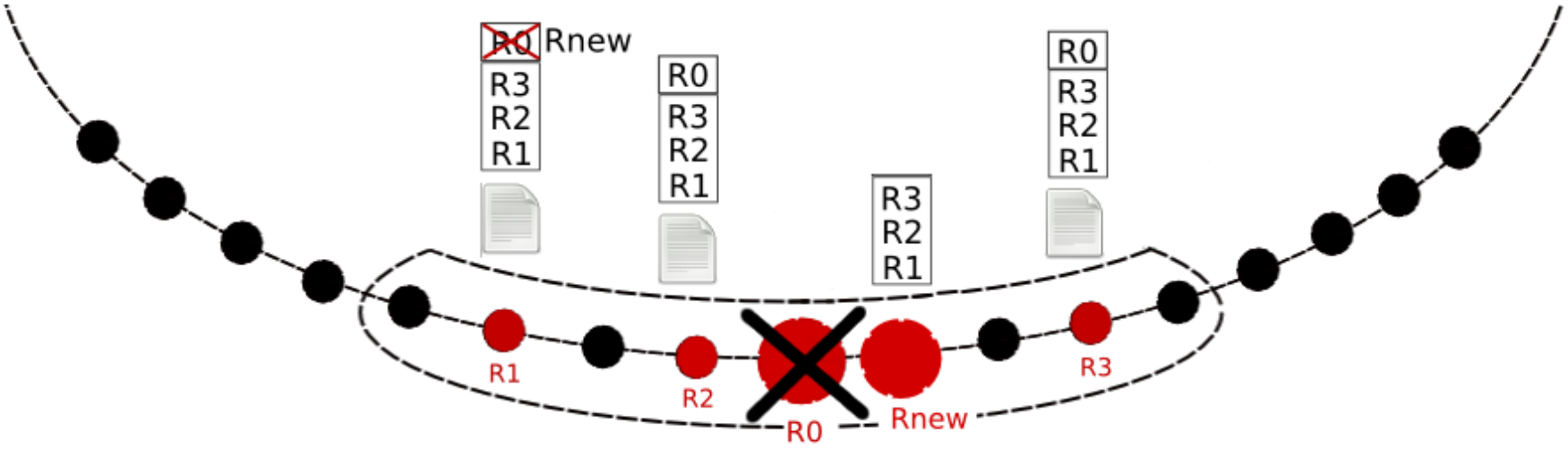



FIGURE: *Lors de la maintenance, une nouvelle racine est désignée par un ou plusieurs nœud réplicats.*

# En cas de perte de la racine

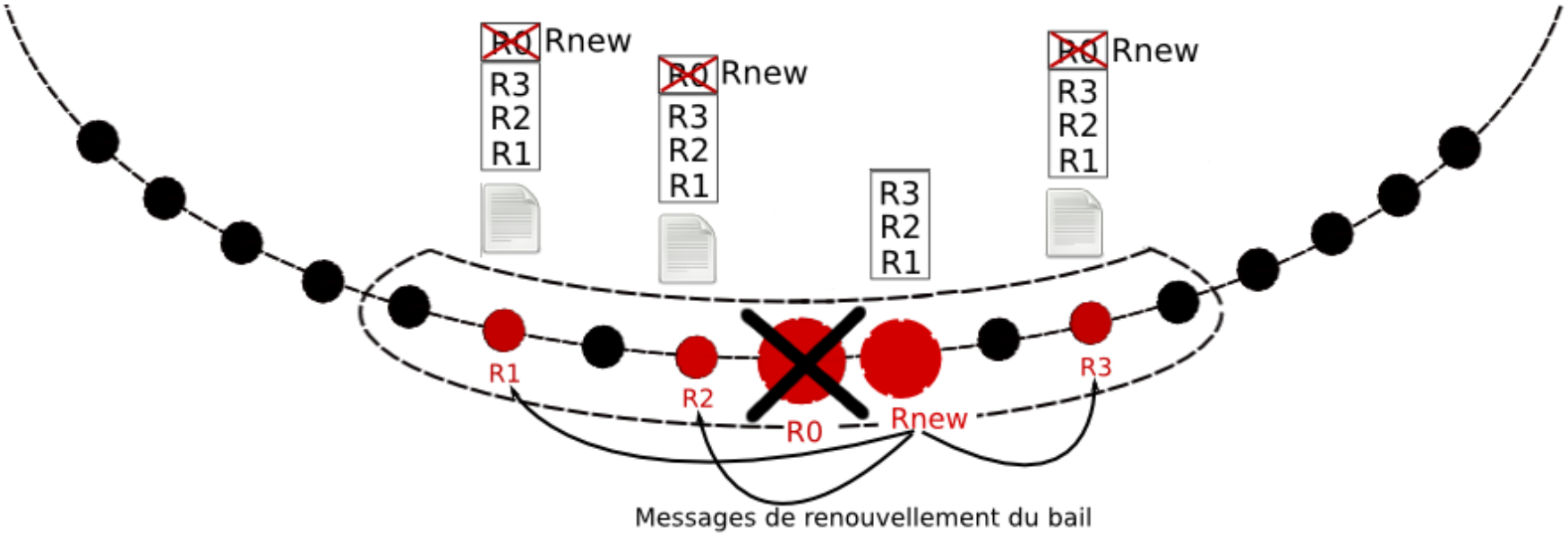



FIGURE: *Lors de la maintenance, une nouvelle racine est désignée par un ou plusieurs nœud réplicats.*

# Gain : augmentation du nombre de sources pour la régénération d'une copie.

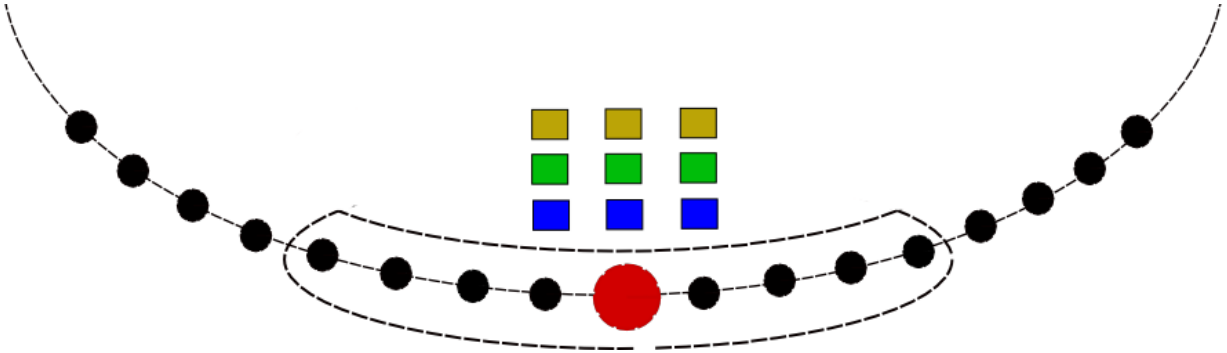


FIGURE: *Stratégie classique à base de leafset.*

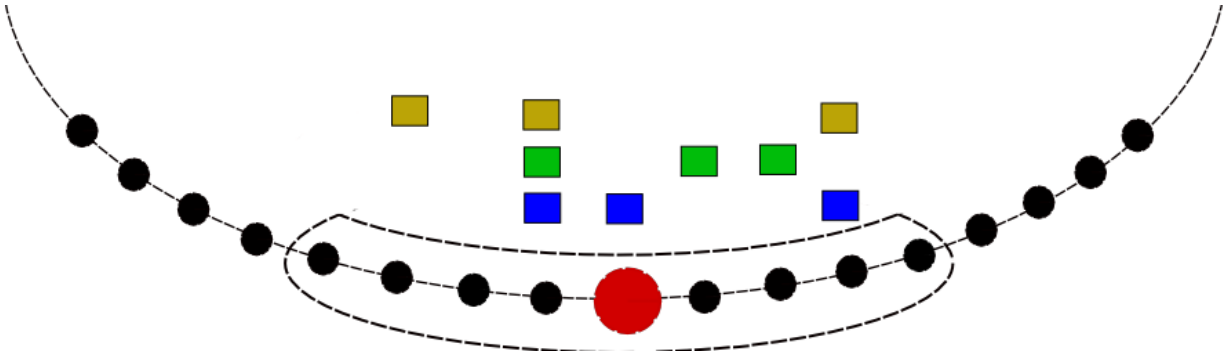


FIGURE: *Stratégie RelaxDHT.*

# Gain : augmentation du nombre de sources pour la régénération d'une copie.

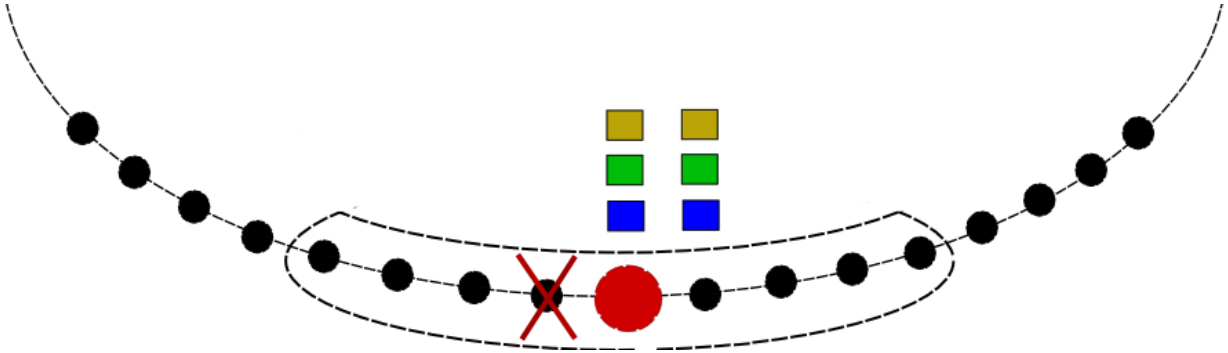


FIGURE: *Stratégie classique à base de leafset.*

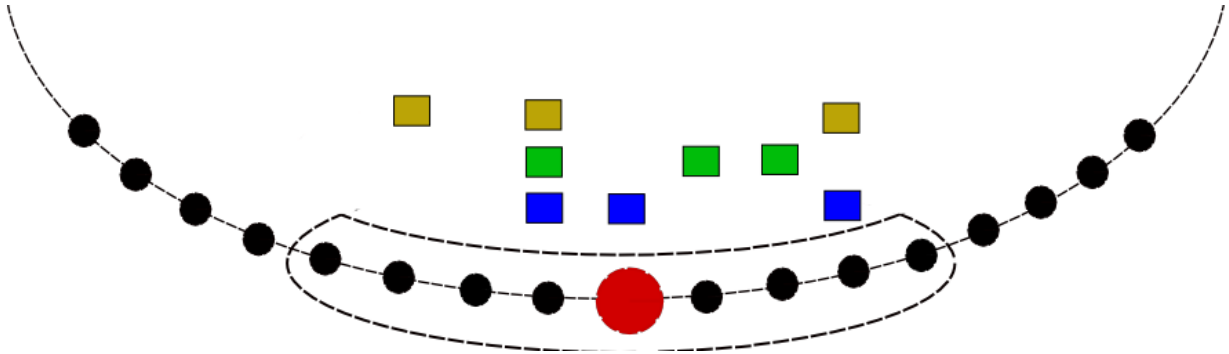


FIGURE: *Stratégie RelaxDHT.*

# Gain : augmentation du nombre de sources pour la régénération d'une copie.

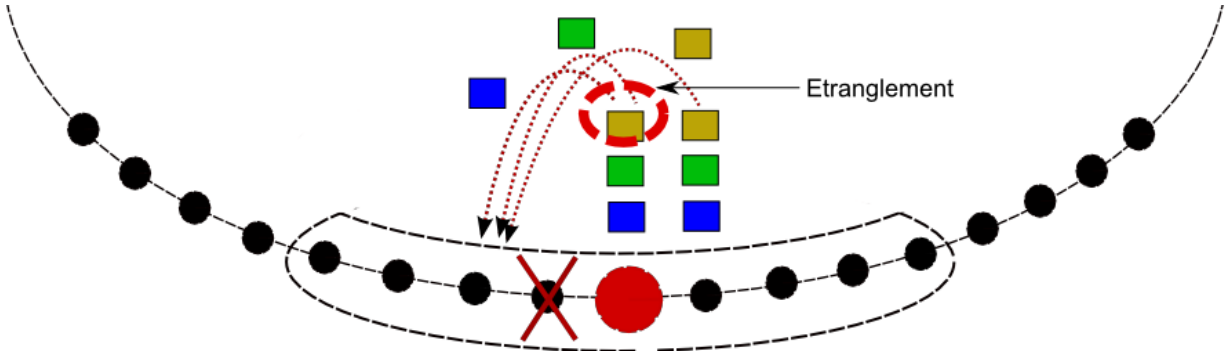



FIGURE: *Stratégie classique à base de leafset.*

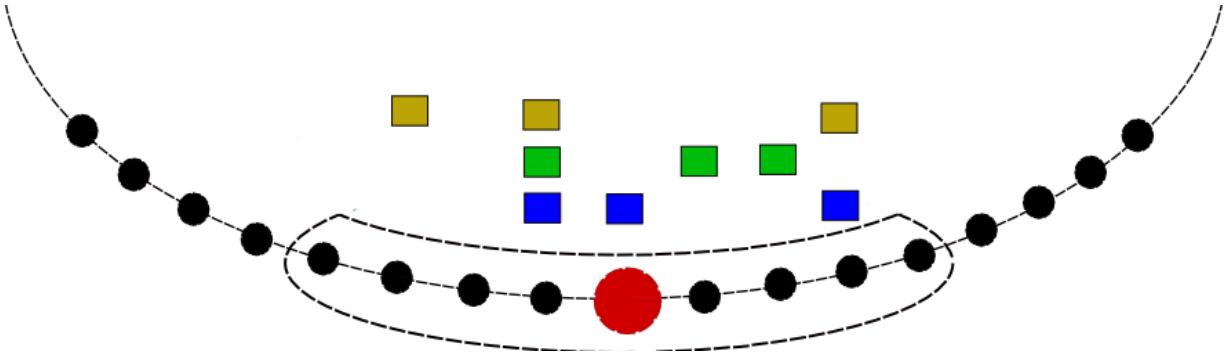


FIGURE: *Stratégie RelaxDHT.*

# Gain : augmentation du nombre de sources pour la régénération d'une copie.

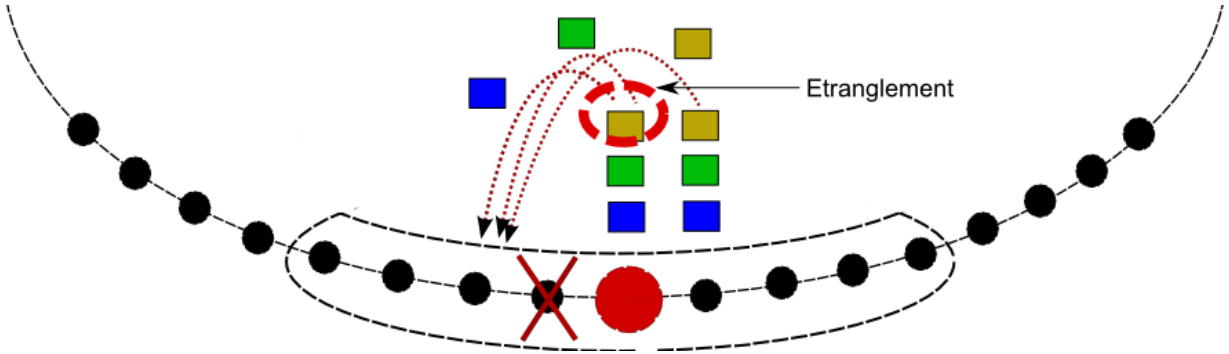



FIGURE: *Stratégie classique à base de leafset.*

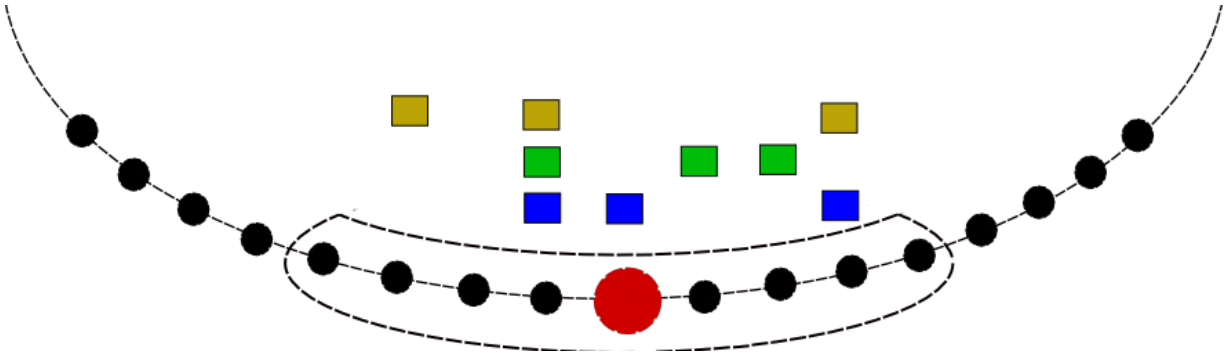


FIGURE: *Stratégie RelaxDHT.*

# Gain : augmentation du nombre de sources pour la régénération d'une copie.

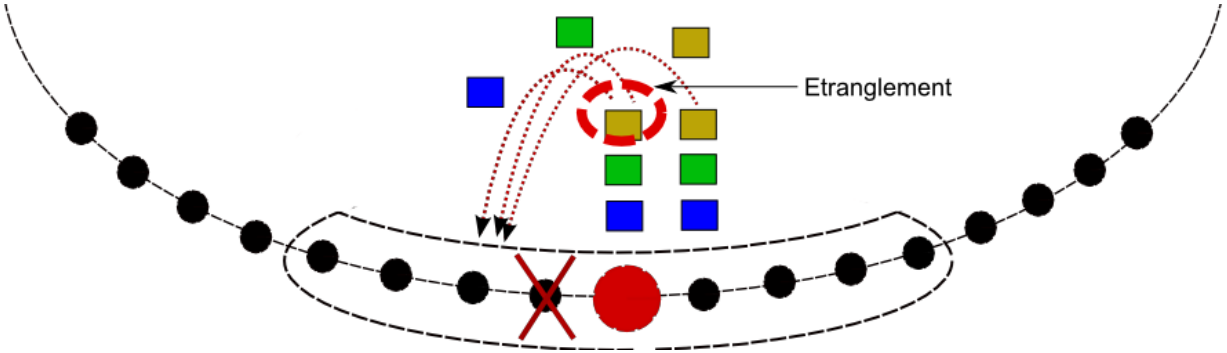



FIGURE: *Stratégie classique à base de leafset.*

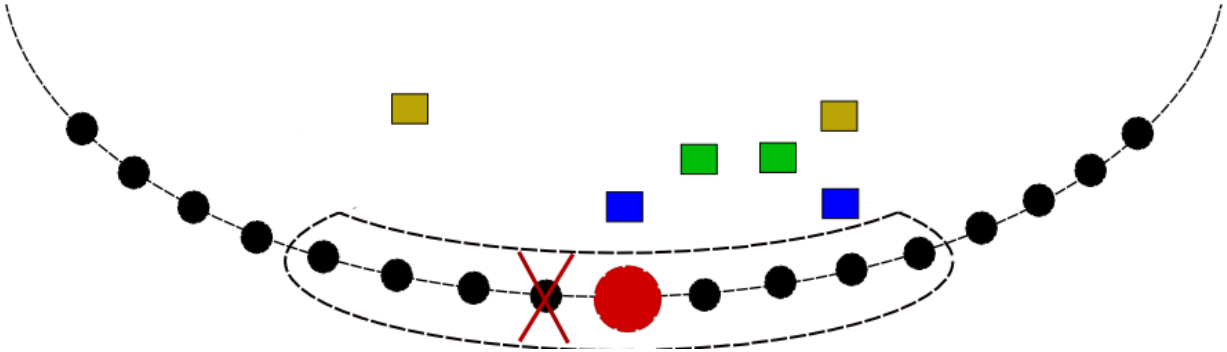


FIGURE: *Stratégie RelaxDHT.*

# Gain : augmentation du nombre de sources pour la régénération d'une copie.

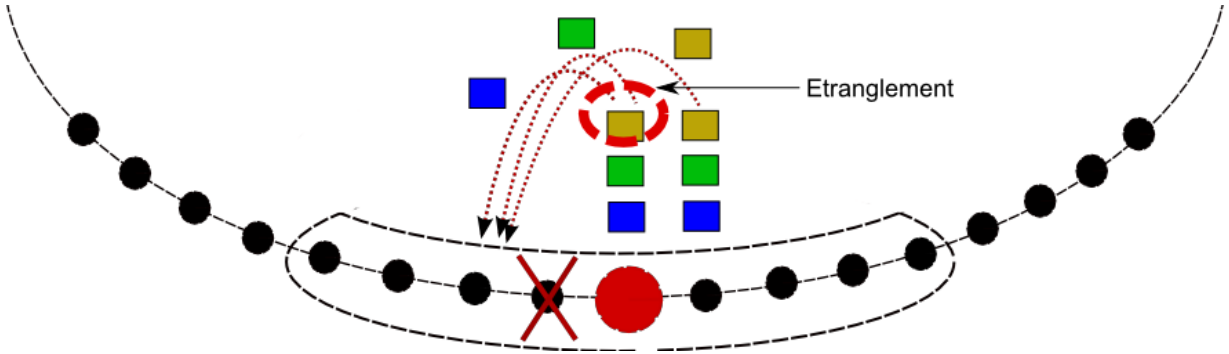



FIGURE: *Stratégie classique à base de leafset.*

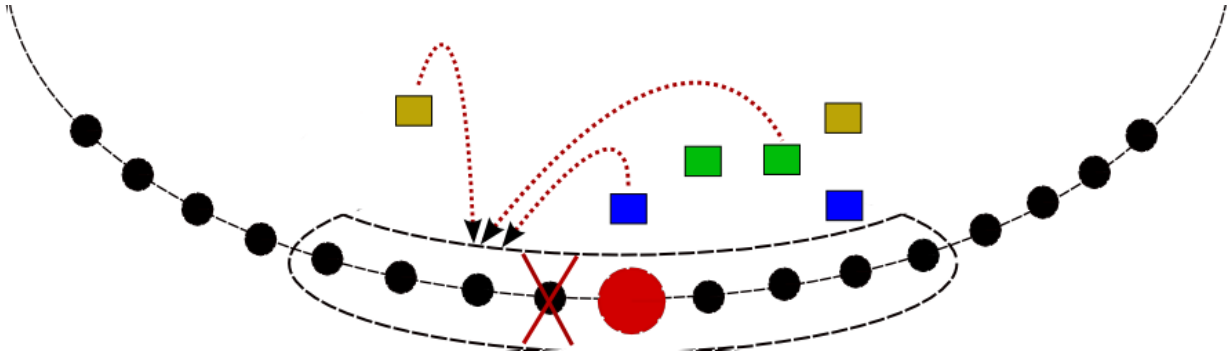


FIGURE: *Stratégie RelaxDHT.*

# Analyse de RelaxDHT

## Avantages :

- **Tolérance aux pannes accrue :**
  - Diminution du nombre de blocs à transferer en cas de churn grâce au relâchement des contraintes de placement.
  - Plus grand nombre de sources disponibles pour la récupération d'un ensemble de blocs.
- **Meilleur passage à l'échelle :**
  - Le nombre de messages est constant (dépend de la taille du leafset).
  - Possibilité de bien compresser les messages.

## Limitations :

En cas de perte de la racine d'un bloc, augmentation *temporaire* du routage d'une requête lookup sur ce bloc.

# Plan

# Conditions d'évaluation

**Implémentation de RelaxDHT dans le simulateur Peersim.**

- DHT de référence : **PAST** (stratégie classique à base de leafset)

**Paramètres :**

- 100 nœuds (+ quelques essais avec 1000 nœuds).
- 10000 blocs de 10 Mo répliqués 3 fois.
- Modèle de churn aléatoire uniforme.
- Taille du leafset : 24
- 1 Mbit/s en flux montant et 10 Mbit/s en flux descendant.

# Évaluation selon deux scénarios différents

1. **Période de churn d'une heure :**

   Une heure (temps simulateur) de perturbations intenses. La période sans churn qui suit permet de mesurer les caractéristiques de convergence du système vers son état normal.

2. **Churn continu :**

   Permet d'évaluer le comportement du système soumis en continu aux connexions/déconnexions de nœuds.

# Métriques employées

**Critères d'évaluation importants :**

- Nombre de blocs perdus (ie. tous réplicats perdus).
- Nombre de blocs échangés.
- Temps de stabilisation post-churn (*Scénario 1*).

On fait varier l'intervalle entre deux perturbations (connexion ou deconnexion d'un nœud aléatoirement choisi dans la DHT).

# Nombre de blocs perdus après la période de perturbation

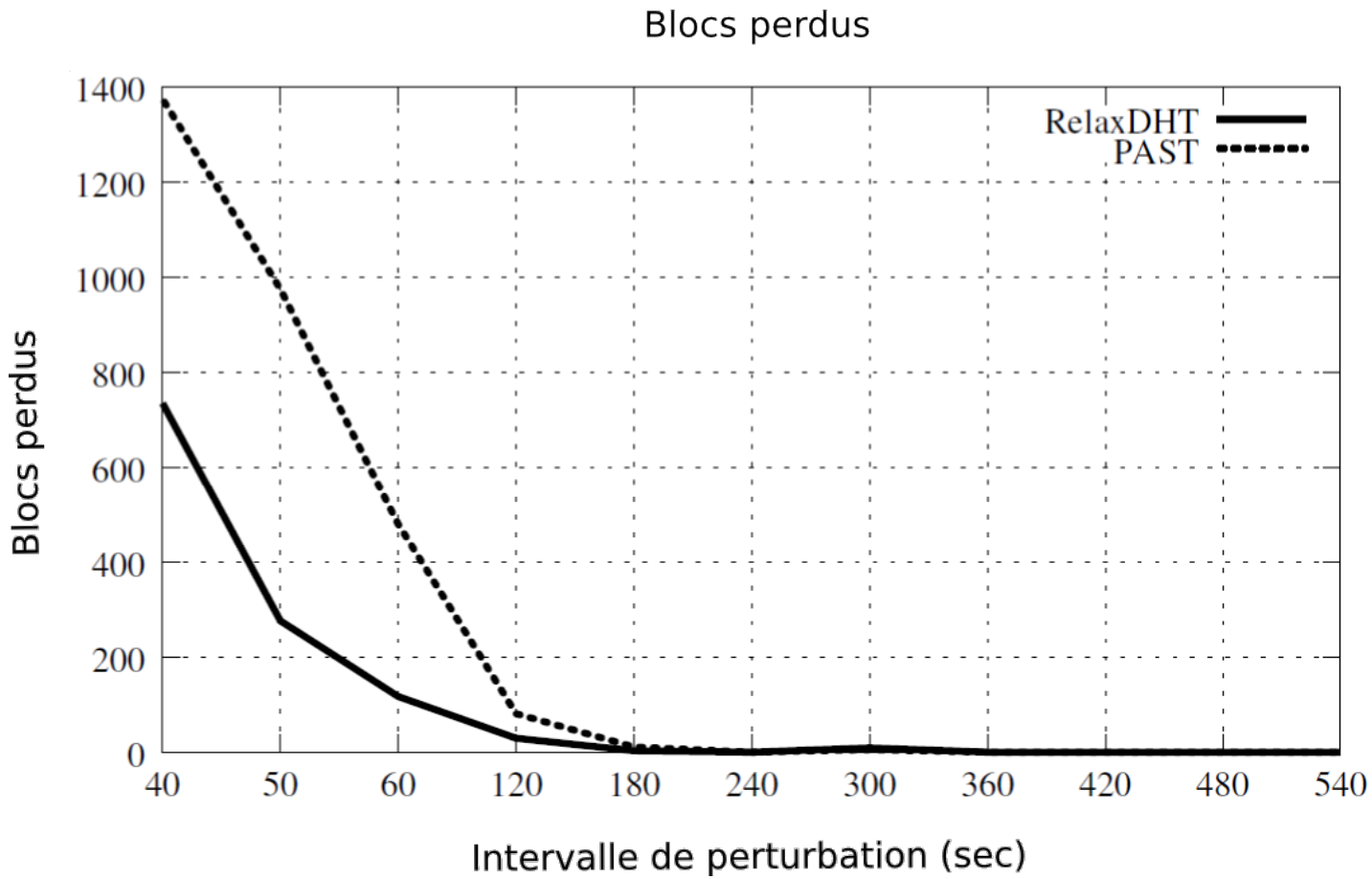



FIGURE: *Évaluation du premier scénario (1).*

# Convergence du système à un état stable après perturbation

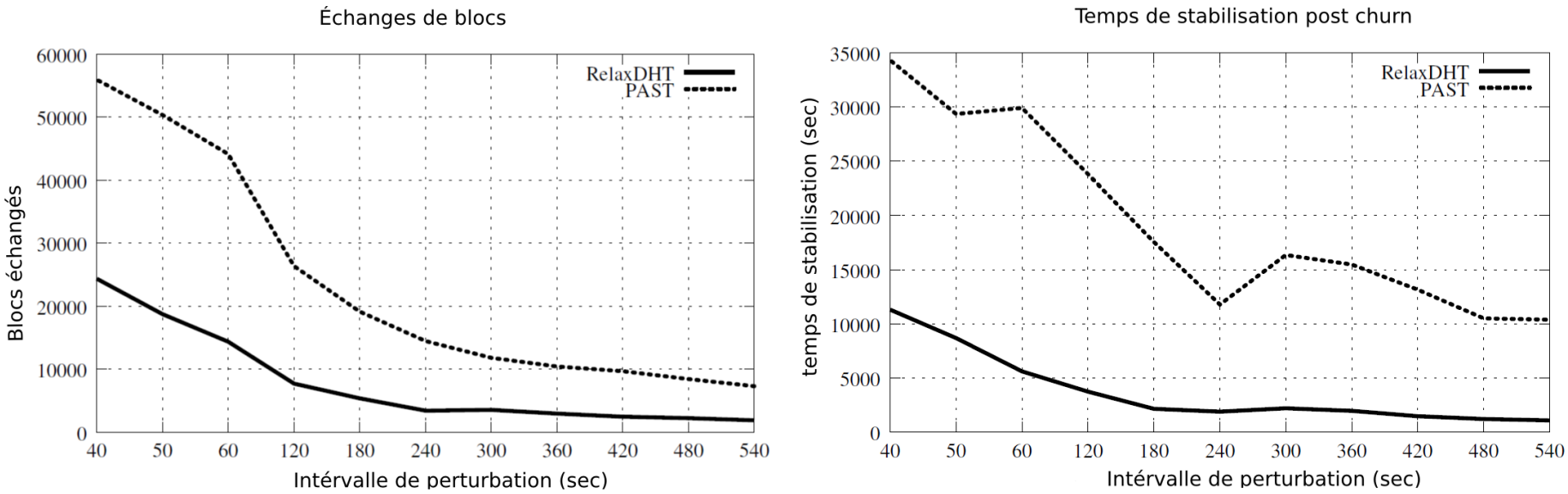
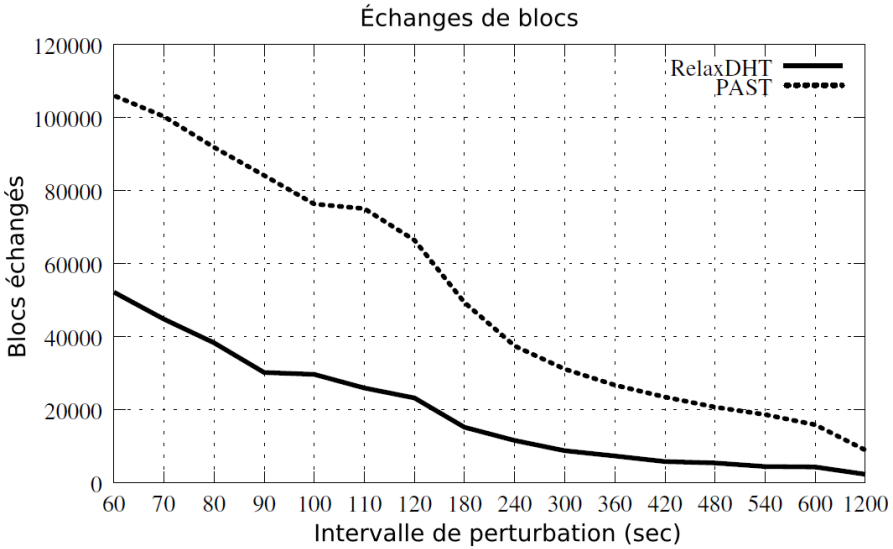



FIGURE: *Évaluation du premier scénario (2) : Convergence du système à un état stable (ie. tous les blocs restants sont à nouveau répliqués 3 fois) :* **quantité de blocs** *échangés et* **temps de convergence**.

# Propriétés du système soumis à un churn continu



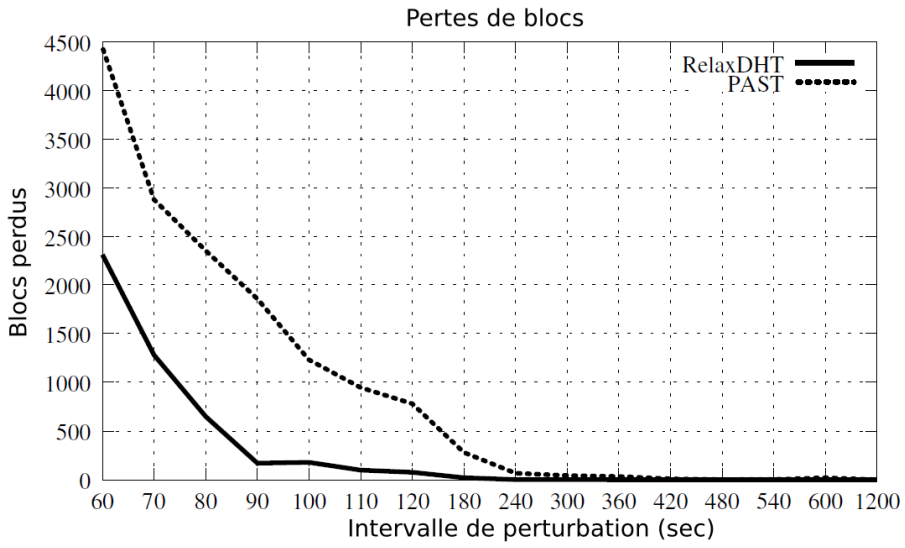




FIGURE: *Évaluation du deuxième scénario :* **nombre total de blocs perdus** *et* **nombre total de blocs échangés**.

# Conclusion

## Gains de 50% en moyenne :

- **Temps de convergence plus court :** Meilleure répartition des sources lors de la récuperation de blocs.
- **Charge réseau plus faible :** moins de transferts de données liés au churn.

## Possibles pertes (à évaluer) :

- les requêtes de lookup.
- la sécurité du système.

## Travaux en cours :

- Expérimentations plus poussées : à plus large échelle, avec un modèle de churn plus évolué, quantification sur le lookup.
- Utilisation de codes correcteurs.

# Merci pour votre attention !

1. Replication strategies
2. RelaxDHT overview
3. Évaluation

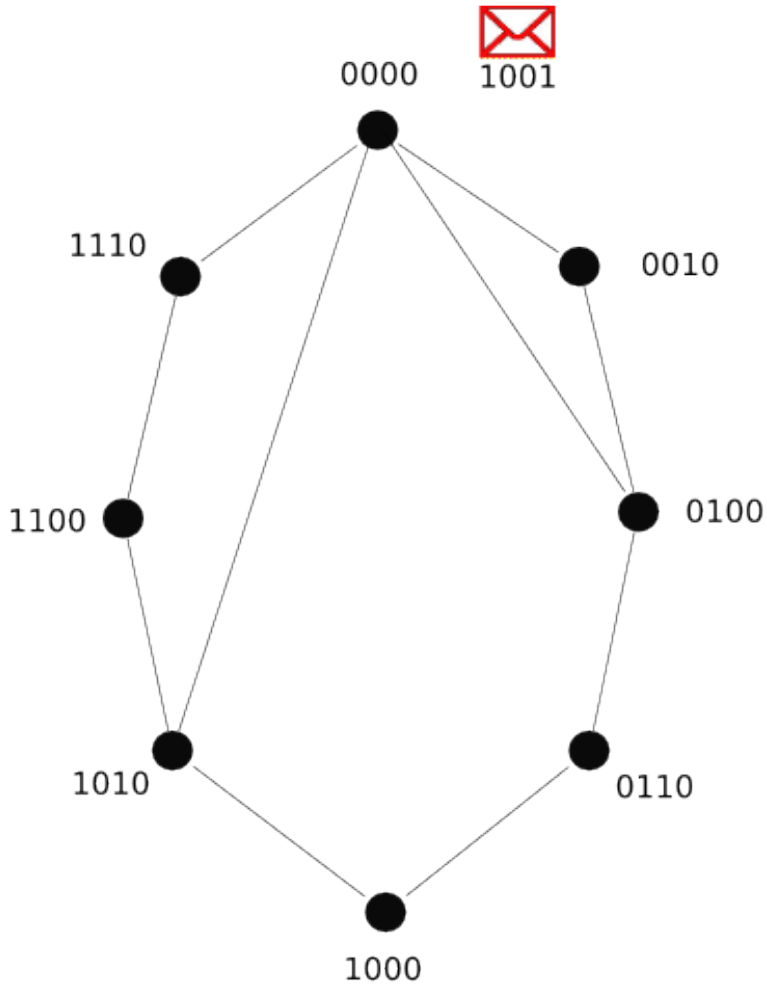

**Questions.**

# Le routage par clés

Couche KBR (*Key-Based Routing*)

## Fonctionnement :

- Valeurs identifiant chaque nœud de la topologie.
- Chaque message est routé selon une clé.
- Le message arrive sur le nœud dont la clé est la plus proche de la sienne.

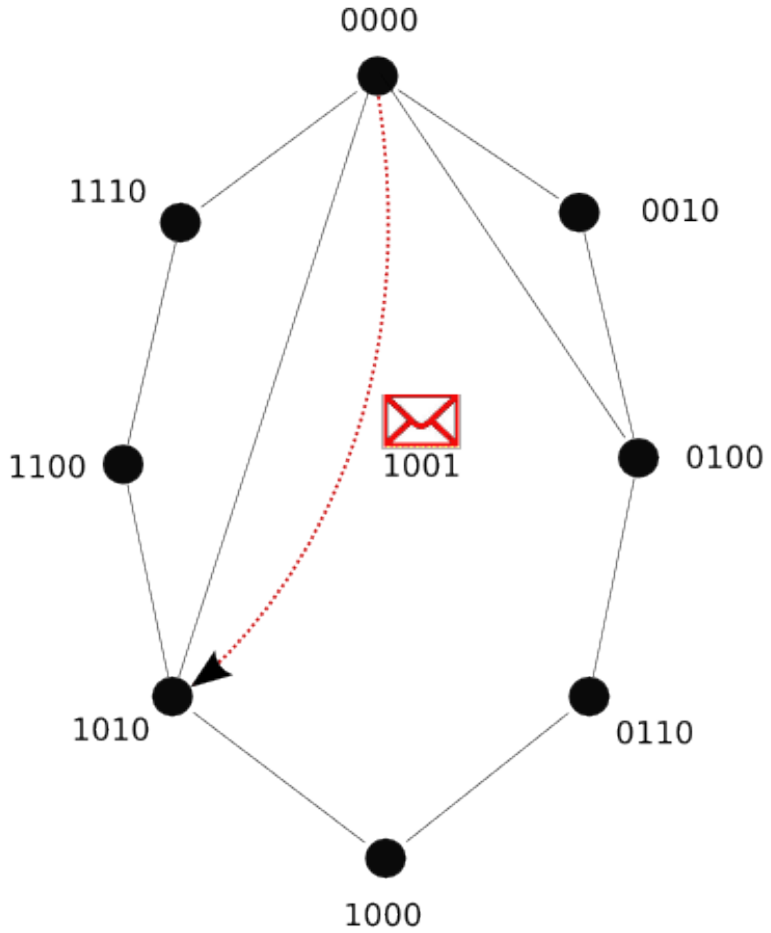


# Le routage par clés

Couche KBR (*Key-Based Routing*)

## Fonctionnement :

- Valeurs identifiant chaque nœud de la topologie.
- Chaque message est routé selon une clé.
- Le message arrive sur le nœud dont la clé est la plus proche de la sienne.

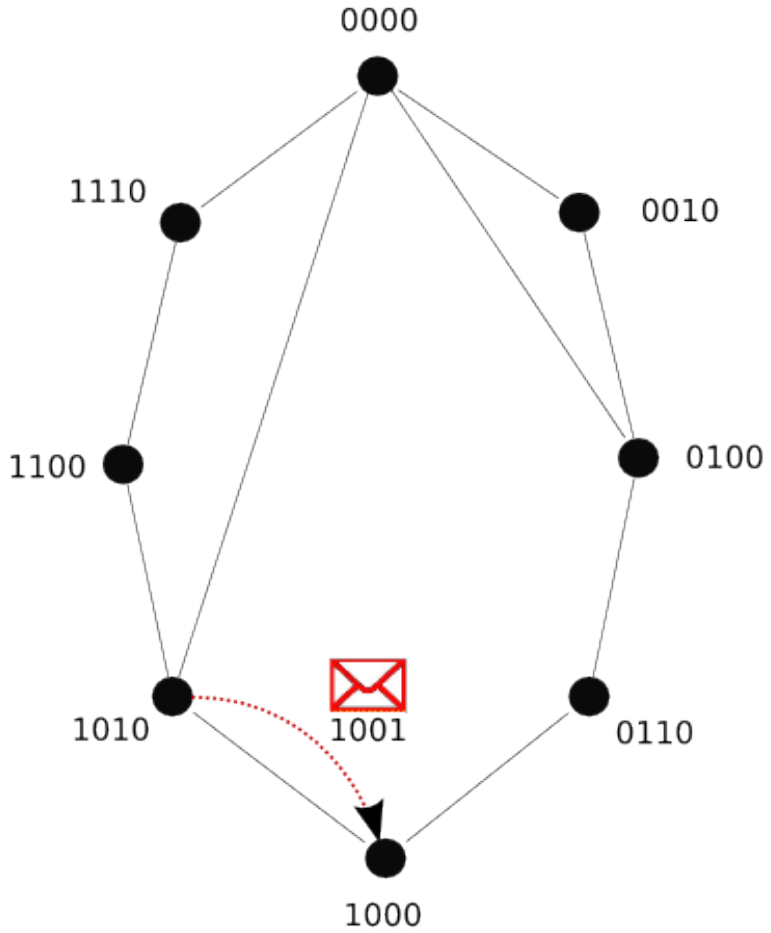


# Le routage par clés

Couche KBR (*Key-Based Routing*)

## Fonctionnement :

- Valeurs identifiant chaque nœud de la topologie.
- Chaque message est routé selon une clé.
- Le message arrive sur le nœud dont la clé est la plus proche de la sienne.

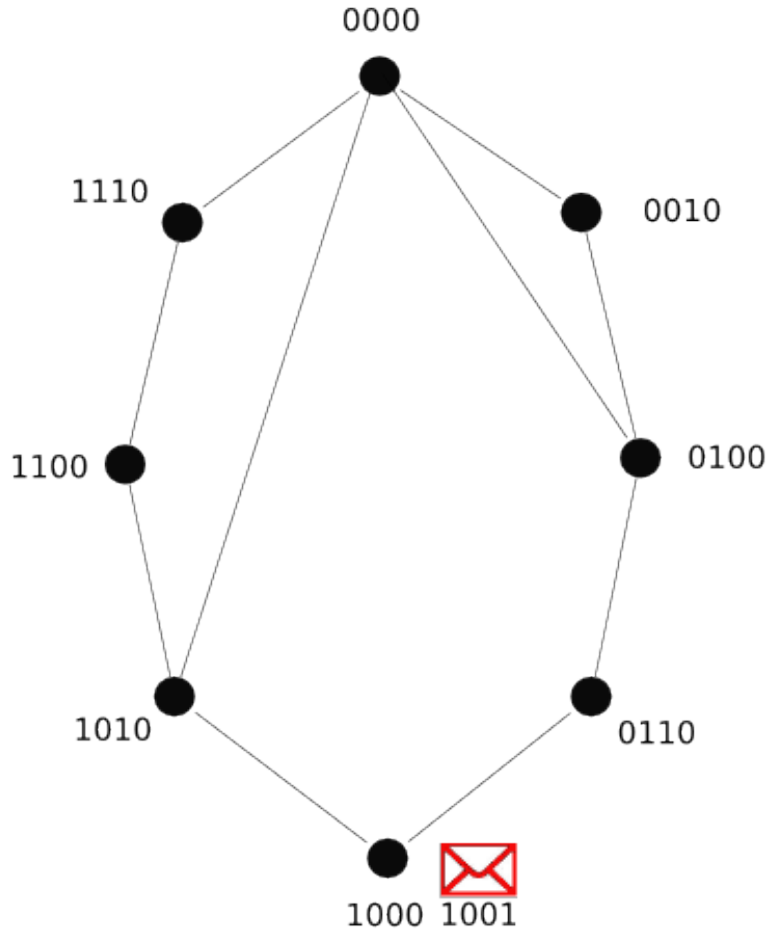


# Le routage par clés

Couche KBR (*Key-Based Routing*)

## Fonctionnement :

- Valeurs identifiant chaque nœud de la topologie.
- Chaque message est routé selon une clé.
- Le message arrive sur le nœud dont la clé est la plus proche de la sienne.



# Une organisation en couches

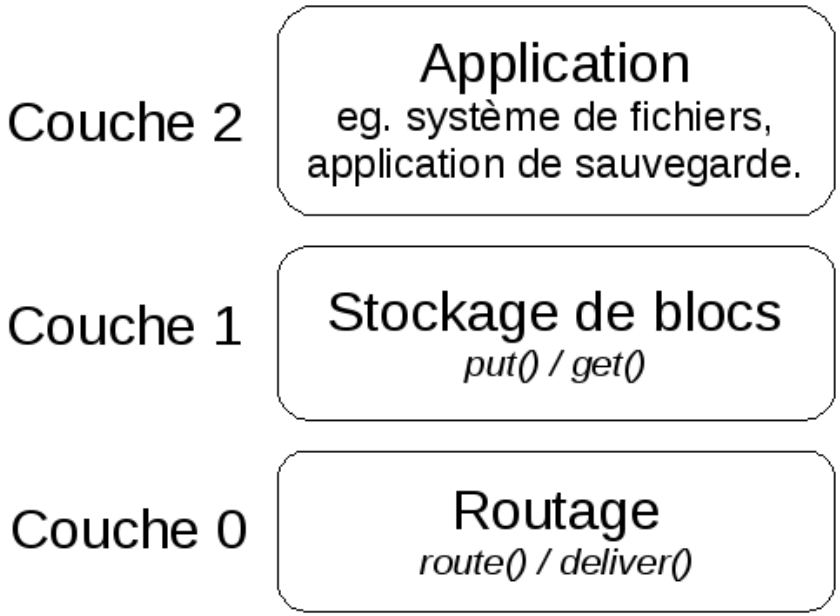



FIGURE: *Spécialisation des couches du système de stockage pair à pair. Les algorithmes de réplication se situent dans la couche de stockage.*