

Ré-identification sans coordination dans les types de données répliquées sans conflits

Matthieu Nicolas (matthieu.nicolas@loria.fr)

20 décembre 2022

<i>Rapporteurs :</i>	Hanifa Boucheneb Davide Frey	Professeure, Polytechnique Montréal Chargé de recherche, HdR, Inria Rennes Bretagne-Atlantique
<i>Examineurs :</i>	Hala Skaf-Molli Stephan Merz	Maîtresse de conférences, HdR, Nantes Université, LS2N Directeur de Recherche, Inria Nancy - Grand Est
<i>Encadrants :</i>	Olivier Perrin Gérald Oster	Professeur des Universités, Université de Lorraine, LORIA Maître de conférences, Université de Lorraine, LORIA

Conflict-free Replicated Data Types (CRDTs)^[1]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Conçues pour données répliquées

[1]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

Conflict-free Replicated Data Types (CRDTs)^[1]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Conçues pour données répliquées

Propriétés des CRDTs

- Permettent modifications **sans coordination**
- Garantissent la **convergence forte**

[1]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

Conflict-free Replicated Data Types (CRDTs)^[1]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Conçues pour données répliquées

Propriétés des CRDTs

- Permettent modifications **sans coordination**
- Garantissent la **convergence forte**

Convergence forte

Ensemble des noeuds ayant intégrés le même ensemble de modifications obtient des états équivalents, sans nécessiter d'actions ou messages supplémentaires

[1]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

LogootSplit^[3], un CRDT pour le type Séquence

- Assigne **identifiant de position** à chaque élément de la séquence

[2]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

[3]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

LogootSplit^[3], un CRDT pour le type Séquence

- Assigne **identifiant de position** à chaque élément de la séquence

Propriétés des identifiants de position^[2]

1. Unique
2. Immuable
3. Ordonnable par une relation d'ordre strict total $<_{id}$
4. Appartenant à un espace dense

[2]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

[3]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

LogootSplit^[3], un CRDT pour le type Séquence

- Assigne **identifiant de position** à chaque élément de la séquence

Propriétés des identifiants de position^[2]

1. Unique
2. Immuable
3. Ordonnable par une relation d'ordre strict total $<_{id}$
4. Appartenant à un espace dense

- **Ordonne les éléments** entre eux **en utilisant** leurs **identifiants**

[2]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

[3]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

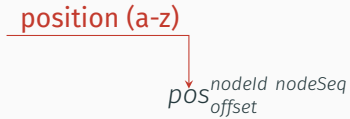
Identifiant

- Composé d'un ou plusieurs tuples suivants

$pos_{offset}^{nodeId \ nodeSeq}$

Identifiant

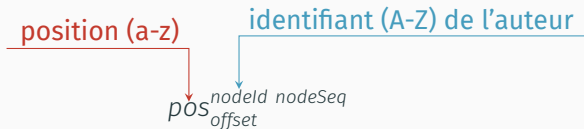
- Composé d'un ou plusieurs tuples suivants



Identifiant LogootSplit

Identifiant

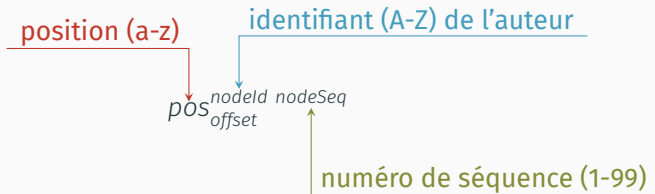
- Composé d'un ou plusieurs tuples suivants



Identifiant LogootSplit

Identifiant

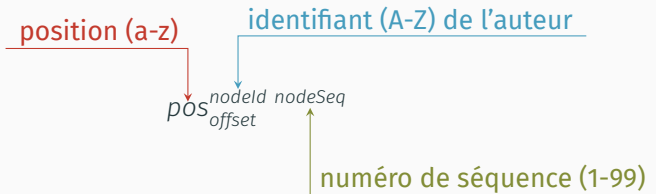
- Composé d'un ou plusieurs tuples suivants



Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



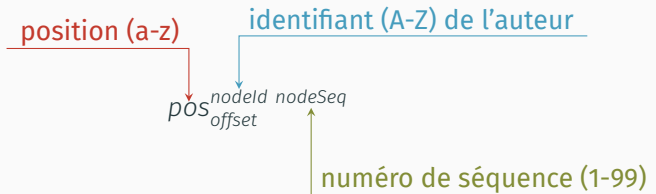
Exemples

d_0^{F5}

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



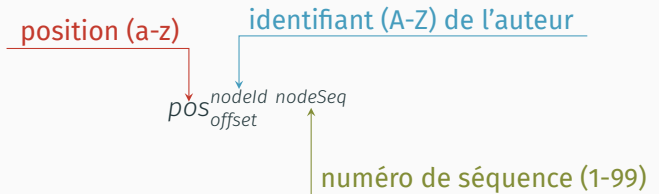
Exemples

$$d_0^{F5} <_{id} m_0^{C1}$$

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



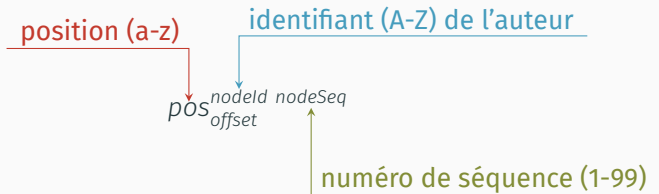
Exemples

$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



Exemples

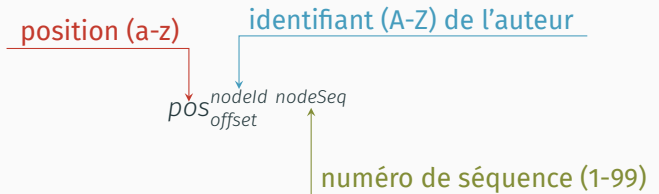
$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

$$i_0^{B1} <_{id} ? <_{id} i_1^{B1}$$

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



Exemples

$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

$$i_0^{B1} <_{id} i_0^{B1} f_0^{A1} <_{id} i_1^{B1}$$

- Coûteux de stocker les identifiants de chaque élément

B	A	N	J	O
m_0^{C1}	m_1^{C1}	m_2^{C1}	m_3^{C1}	m_4^{C1}

Bloc LogootSplit

- Coûteux de stocker les identifiants de chaque élément

B	A	N	J	O
m_0^{C1}	m_1^{C1}	m_2^{C1}	m_3^{C1}	m_4^{C1}

- Aggrège en un **bloc** éléments ayant **identifiants contigus**

Identifiants contigus

Deux identifiants sont contigus si et seulement si les deux identifiants sont identiques à l'exception de leur dernier offset et que leur derniers offsets sont consécutifs.

Bloc LogootSplit

- Coûteux de stocker les identifiants de chaque élément

B	A	N	J	O
m_0^{C1}	m_1^{C1}	m_2^{C1}	m_3^{C1}	m_4^{C1}

- Aggrège en un **bloc** éléments ayant **identifiants contigus**

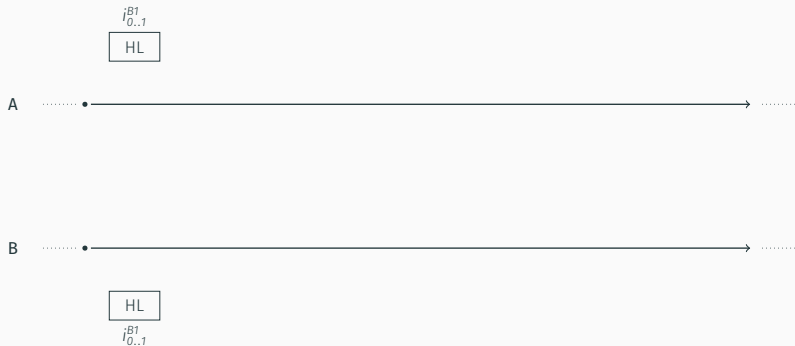
Identifiants contigus

Deux identifiants sont contigus si et seulement si les deux identifiants sont identiques à l'exception de leur dernier offset et que leur derniers offsets sont consécutifs.

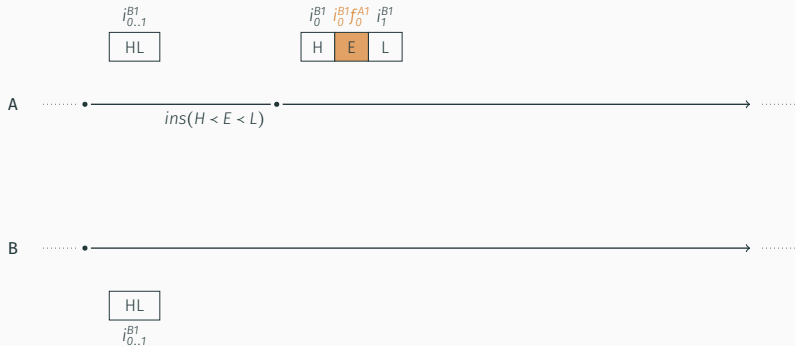
- Note l'intervalle d'identifiants d'un bloc : $pos_{begin..end}^{nodeId nodeSeq}$

BANJO
$m_{0..4}^{C1}$

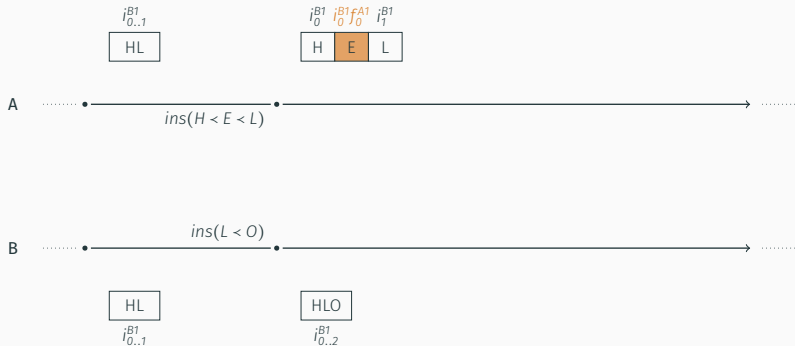
Exemple insertions concurrentes



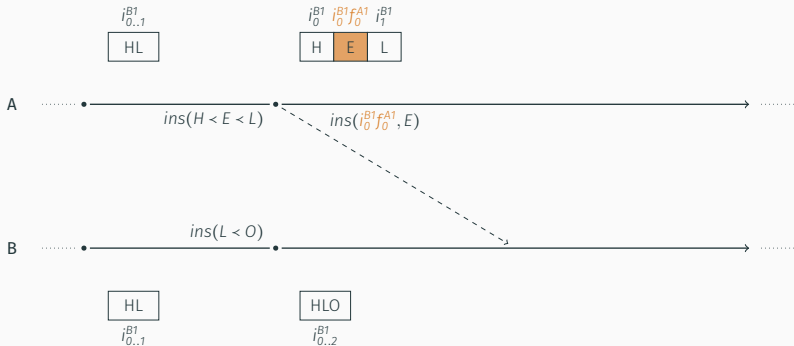
Exemple insertions concurrentes



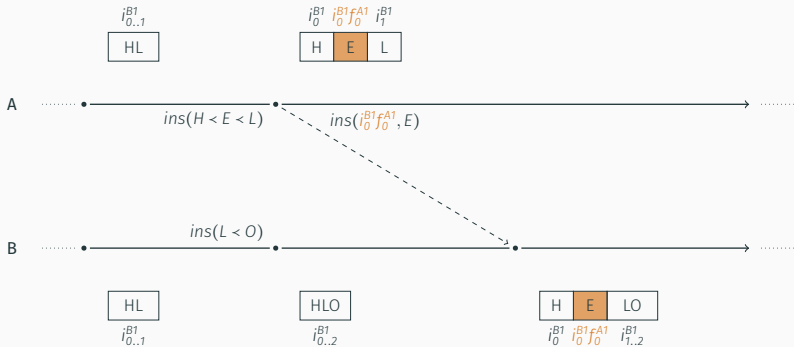
Exemple insertions concurrentes



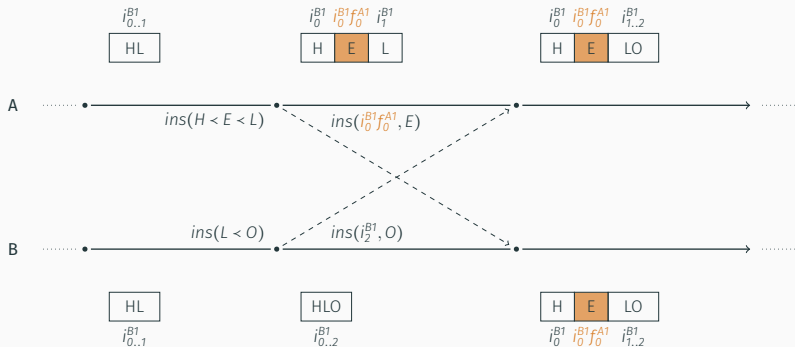
Exemple insertions concurrentes



Exemple insertions concurrentes



Exemple insertions concurrentes



Sources croissance métadonnées

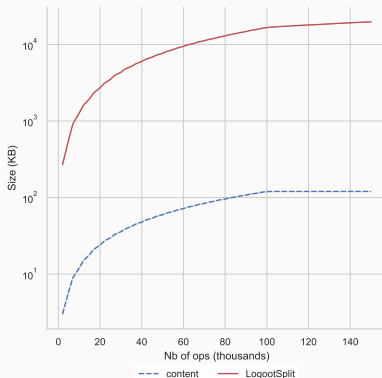
- Croissance non-bornée de la taille des identifiants
- Fragmentation en blocs courts

Limites de LogootSplit

Sources croissance métadonnées

- Croissance non-bornée de la taille des identifiants
- Fragmentation en blocs courts

Taille du contenu comparé à la taille de la séquence LogootSplit



L'approche core-nebula^[4]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Transforme les opérations *insert* et *remove* concurrentes...

[4]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

L'approche core-nebula^[4]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Transforme les opérations *insert* et *remove* concurrentes...
- ...mais ne supportent pas opérations *rename* concurrentes

[4]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

L'approche core-nebula^[4]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Transforme les opérations *insert* et *remove* concurrentes...
- ...mais ne supportent pas opérations *rename* concurrentes

Inadaptée aux applications pair-à-pair

[4]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

Proposition

Mécanisme de renommage supportant les
renommages concurrents

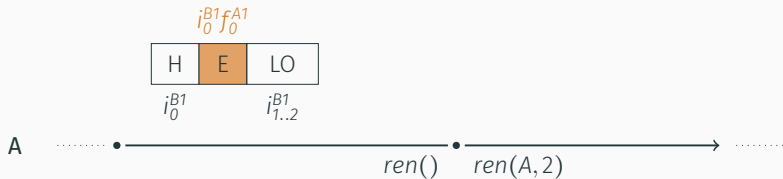
RenamableLogootSplit

- CRDT pour le type Séquence qui incorpore un mécanisme de renommage
- Prend la forme d'une nouvelle opération : *rename*

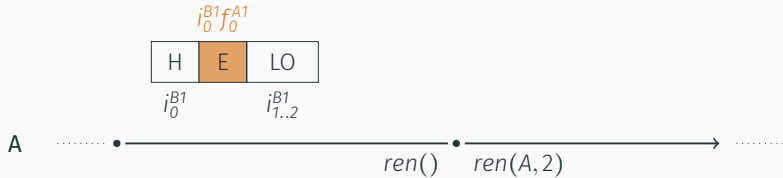
Propriétés de l'opération *rename*

- Est déterministe
- Préserve l'intention des utilisateur-rices
- Préserve la séquence, c.-à-d. unicité et ordre de ses identifiants
- Commute avec les opérations *insert*, *remove* mais aussi *rename* concurrentes

Opération *rename*

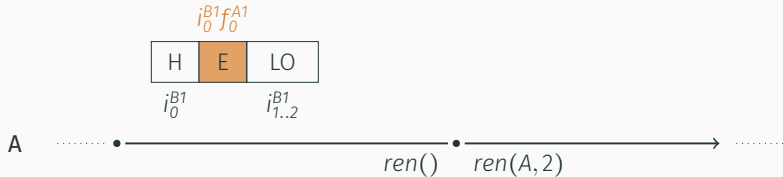


Opération *rename*



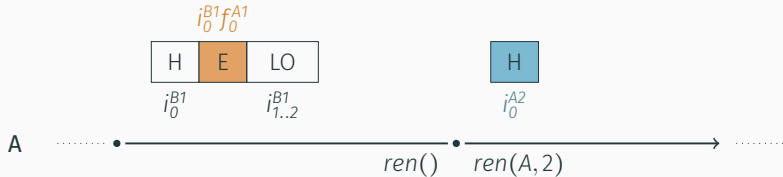
- Génère nouvel identifiant pour le 1er élément :

Opération *rename*



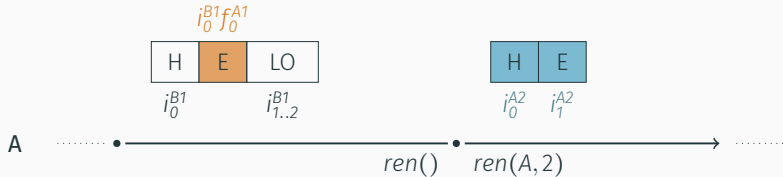
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$

Opération *rename*



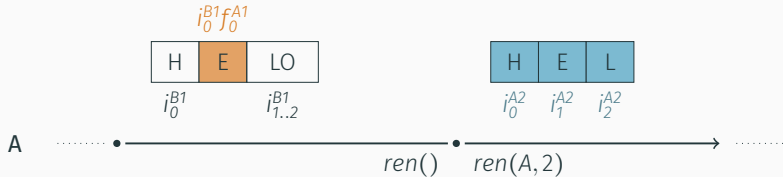
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants :

Opération *rename*



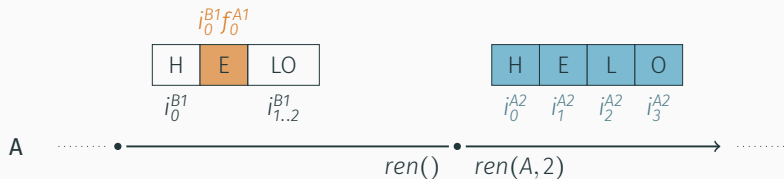
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2}

Opération *rename*



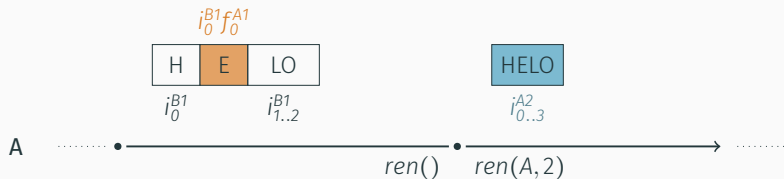
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2}, i_2^{A2}

Opération *rename*



- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : $i_1^{A2}, i_2^{A2}, \dots$

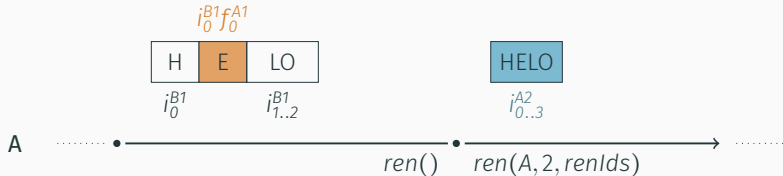
Opération *rename*



- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : $i_1^{A2}, i_2^{A2}, \dots$

Regroupe tous les éléments en 1 unique bloc

Opération *rename*

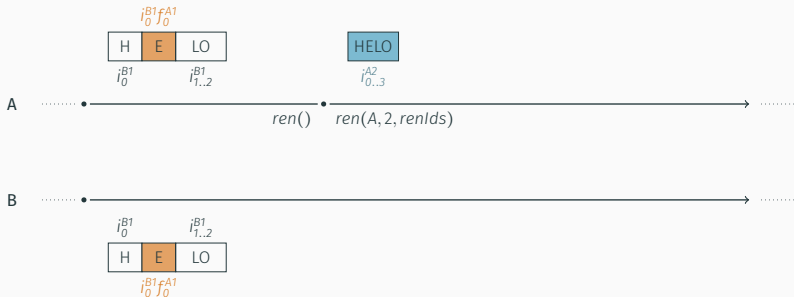


- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : $i_1^{A2}, i_2^{A2}, \dots$

Regroupe tous les éléments en 1 unique bloc

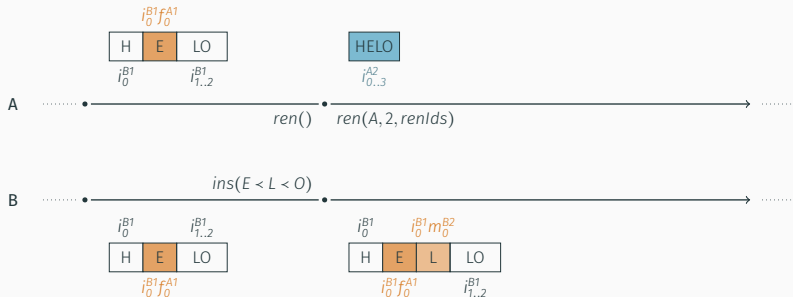
- Stocke identifiants ($[i_0^{B1}, i_0^{B1} f_0^{A1}, \dots]$) de l'état d'origine : *renlds*

Interactions avec opérations *insert* et *remove* concurrentes



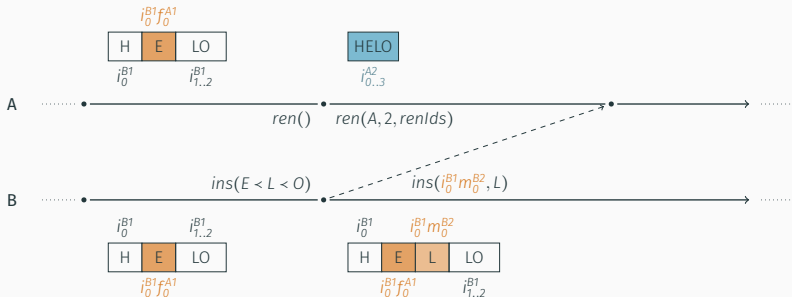
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



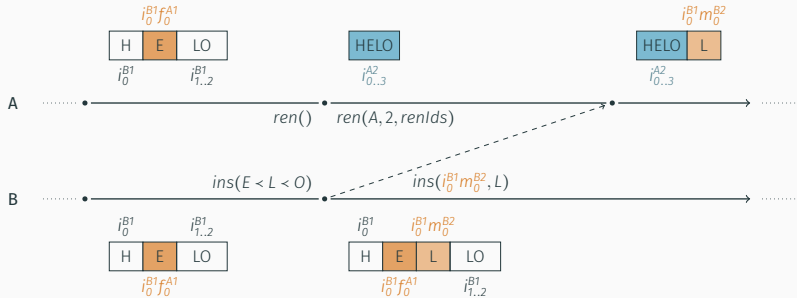
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



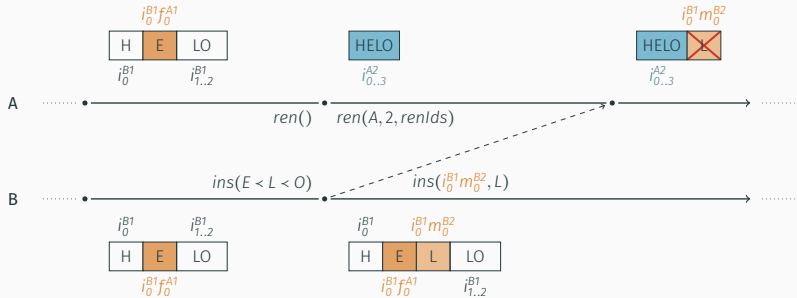
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



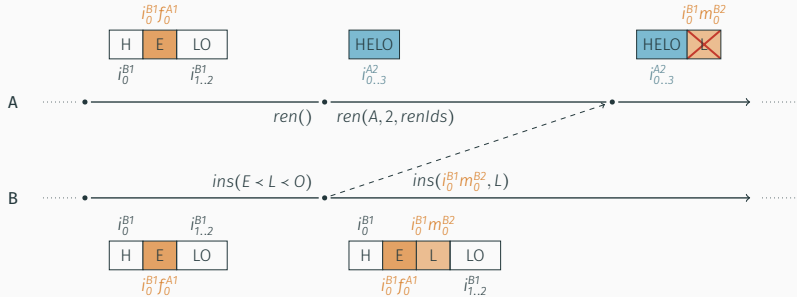
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



- Peuvent générer opérations concurrentes aux opérations *rename*
- Produisent anomalies si intégrées naïvement

Interactions avec opérations *insert* et *remove* concurrentes



- Peuvent générer opérations concurrentes aux opérations *rename*
- Produisent anomalies si intégrées naïvement

Nécessité d'un mécanisme dédié

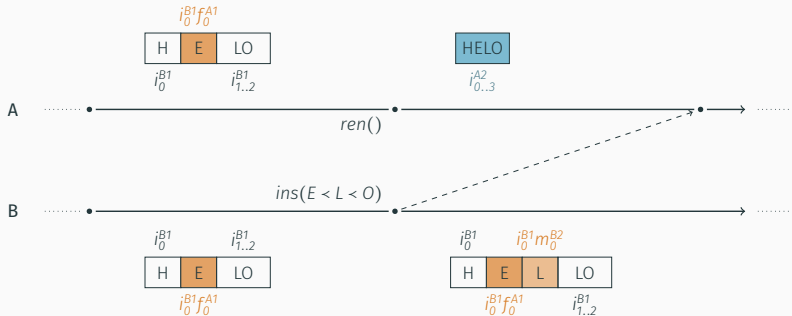
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

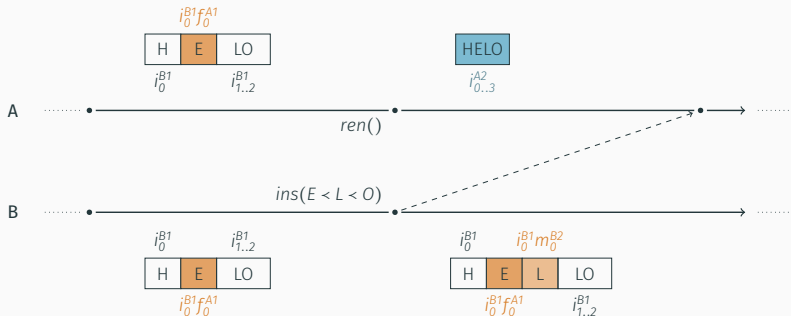
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

Détection des opérations concurrentes à opération *rename*

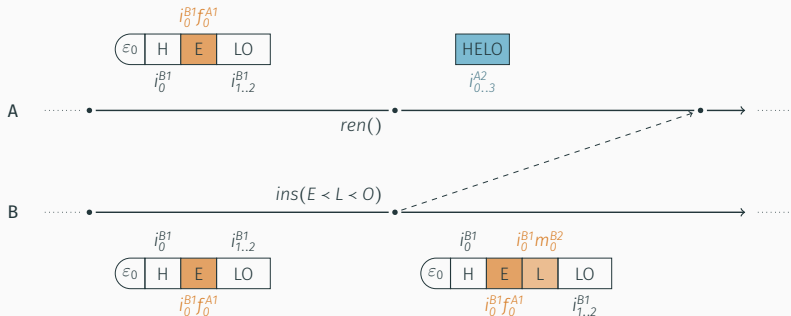


Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

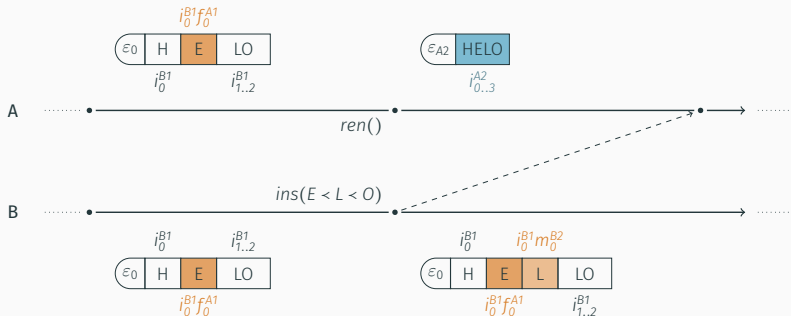
Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ϵ_0

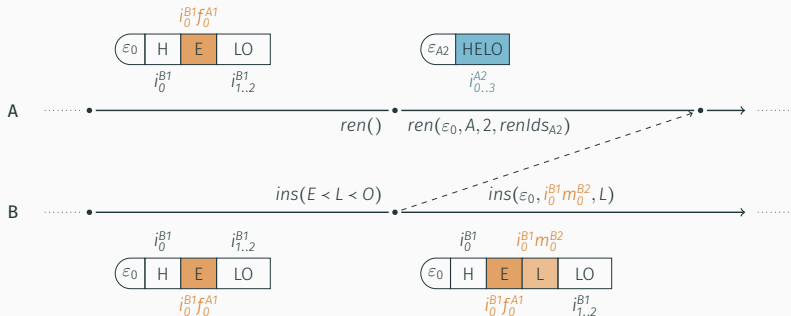
Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ϵ_0
- *rename* font progresser à nouvelle époque, $\epsilon_{nodeId \ nodeSeq}$

Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ϵ_0
- *rename* font progresser à nouvelle époque, $\epsilon_{nodeId} nodeSeq$
- Opérations labellisées avec époque de génération

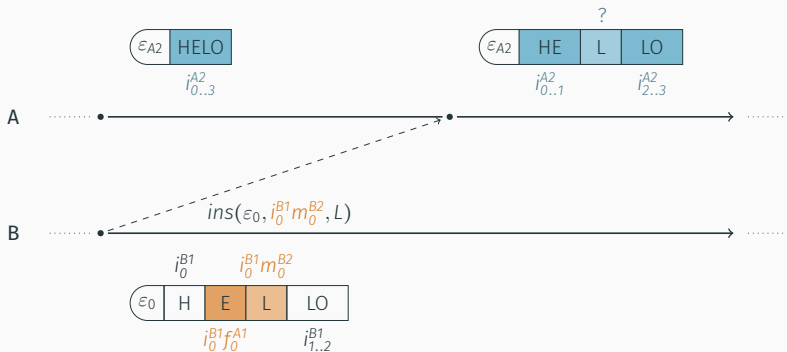
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

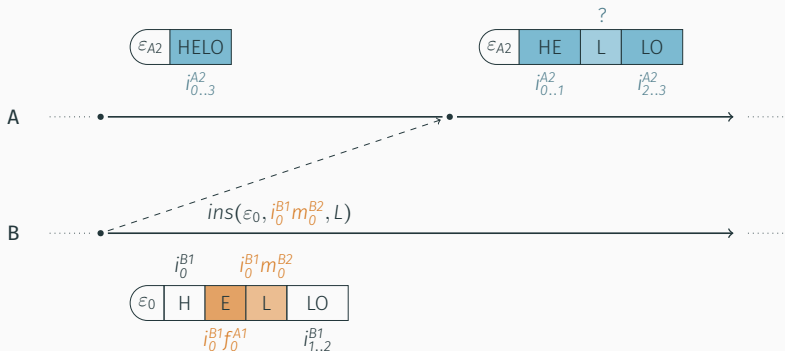
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

Intégration des opérations *insert* et *remove* concurrentes

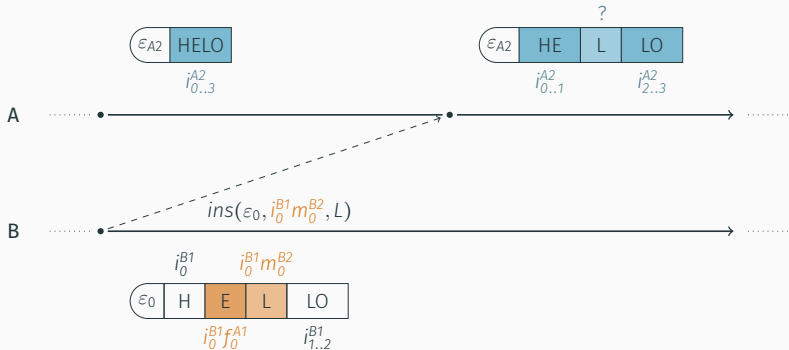


Intégration des opérations *insert* et *remove* concurrentes



Ajout d'un mécanisme de transformation des opérations *insert* et *remove* concurrentes

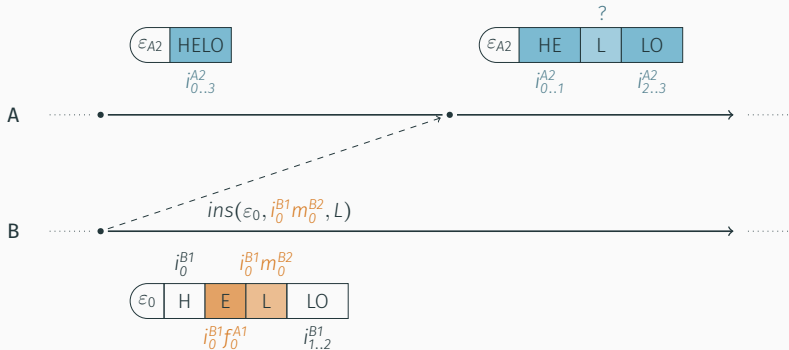
Intégration des opérations *insert* et *remove* concurrentes



Ajout d'un mécanisme de transformation des opérations *insert* et *remove* concurrentes

- Prend la forme de l'algorithme `renameId`

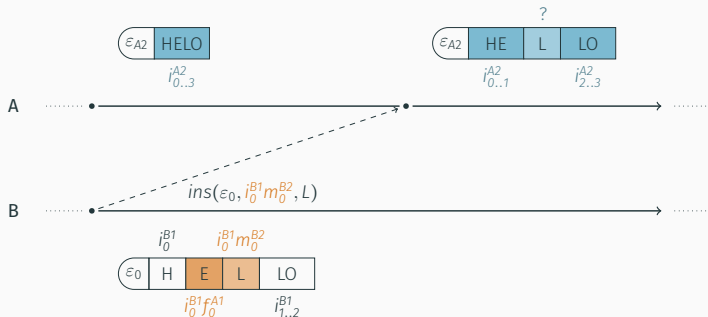
Intégration des opérations *insert* et *remove* concurrentes



Ajout d'un mécanisme de transformation des opérations *insert* et *remove* concurrentes

- Prend la forme de l'algorithme `renameId`
- Inclure l'effet de l'opération *rename* dans l'opération transformée

Exemple de renameId

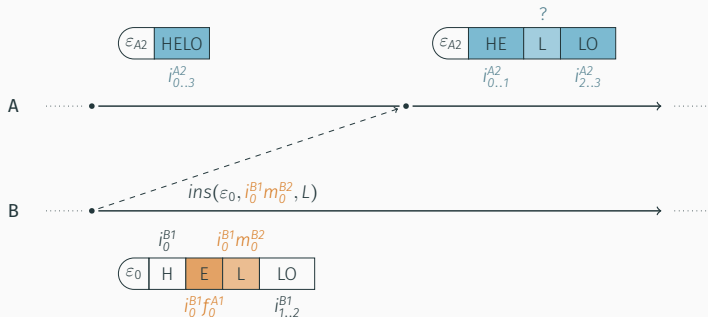


Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

Exemple de renameId



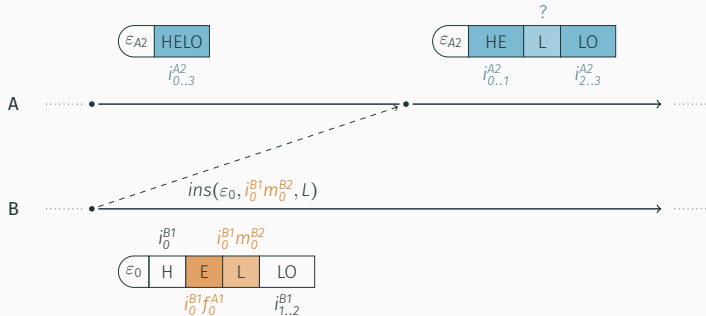
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$

Exemple de renameId



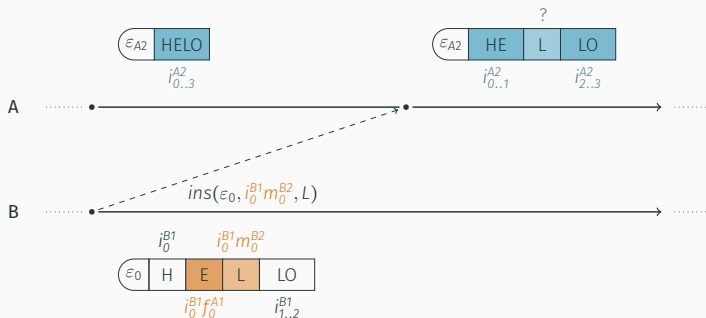
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver son équivalent à l'époque cible ε_{A2} : i_1^{A2}

Exemple de renameId



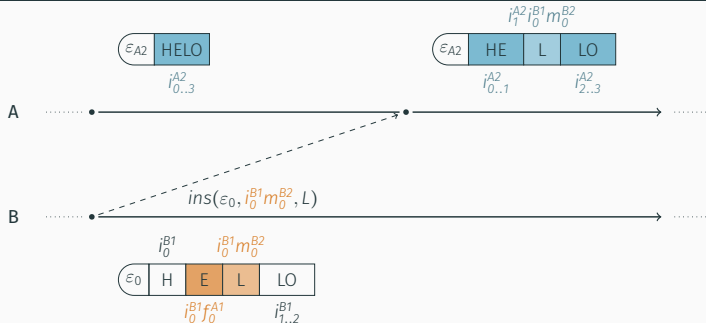
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver son équivalent à l'époque cible ϵ_{A2} : i_1^{A2}
- Préfixer $i_0^{B1} m_0^{B2}$ par ce dernier pour obtenir son équivalent à ϵ_{A2} : $i_1^{A2} i_0^{B1} m_0^{B2}$

Exemple de renameId



Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver son équivalent à l'époque cible ϵ_{A2} : i_1^{A2}
- Préfixer $i_0^{B1} m_0^{B2}$ par ce dernier pour obtenir son équivalent à ϵ_{A2} : $i_1^{A2} i_0^{B1} m_0^{B2}$