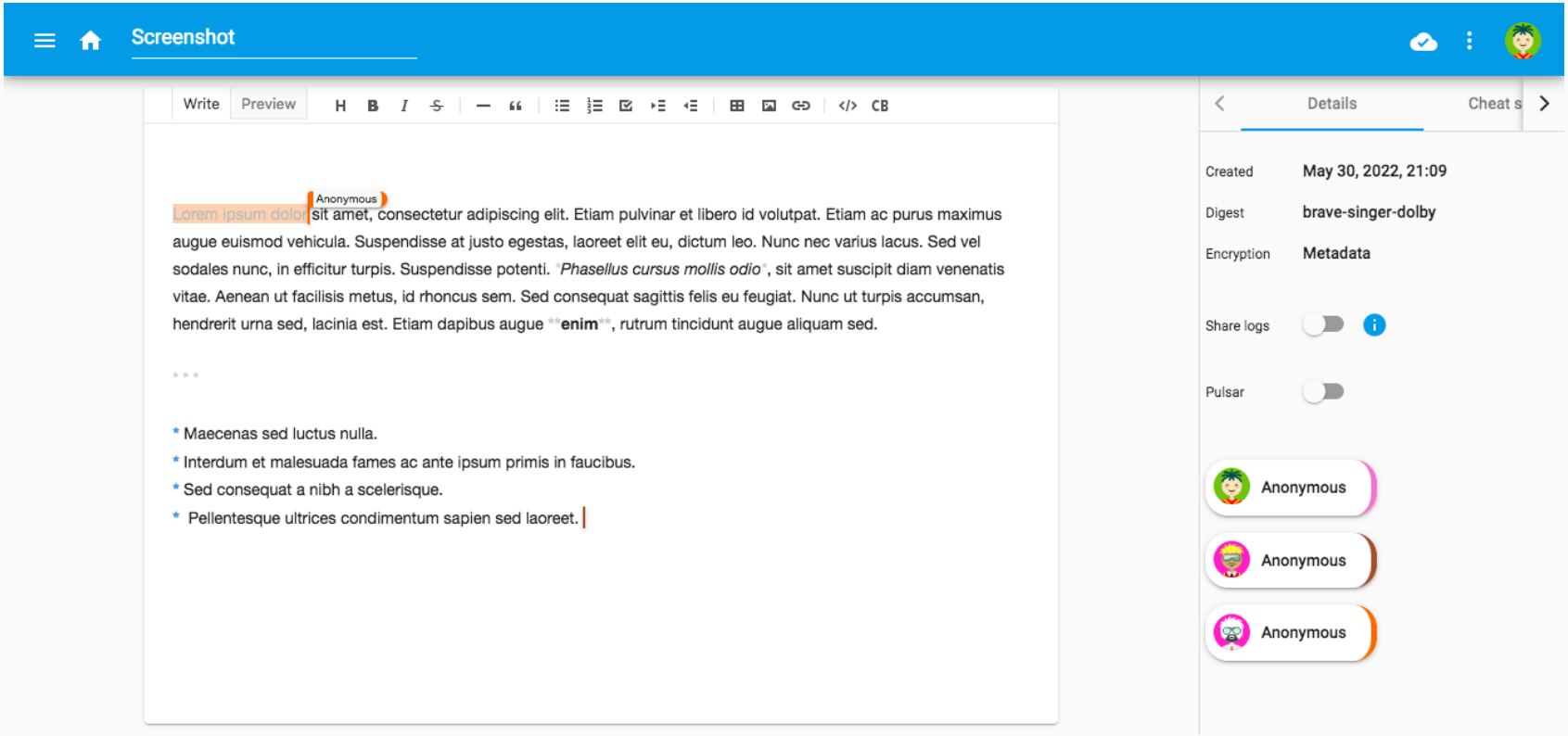
Ré-identification sans coordination dans les types de données répliquées sans conflits

Matthieu Nicolas (`matthieu.nicolas@loria.fr`)

20 décembre 2022

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

MUTE *, un exemple de Local-First Software (LFS) [1]

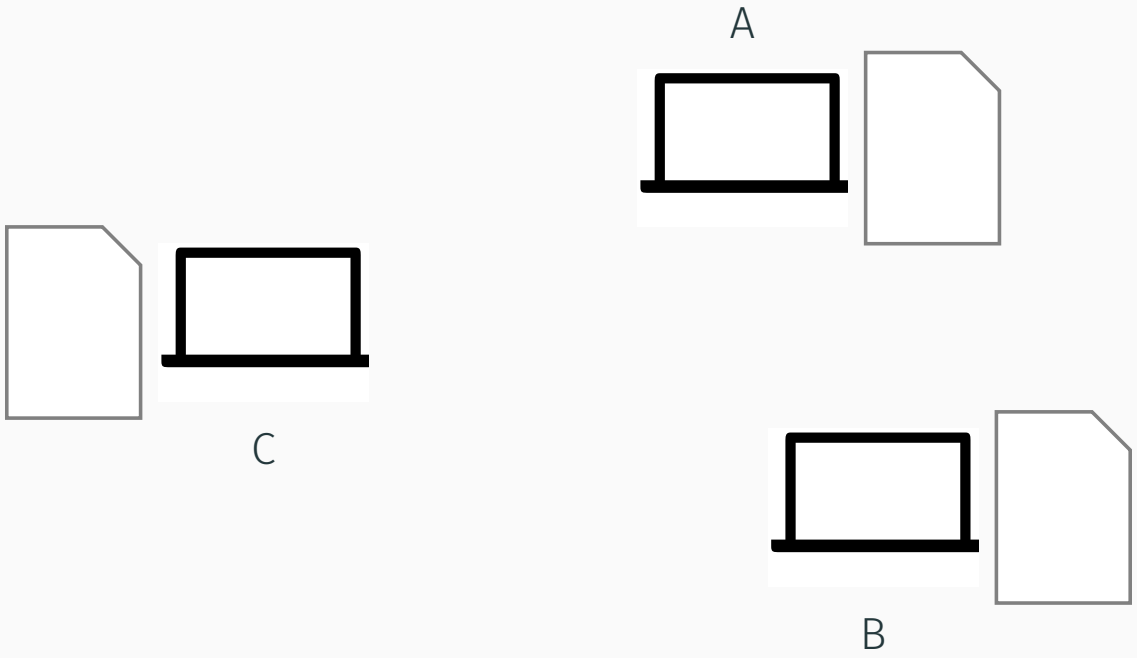


- Application pair-à-pair
- Permet de rédiger collaborativement des documents texte
- Garantit la confidentialité & souveraineté des données

*. Disponible à : `https://mutehost.loria.fr`

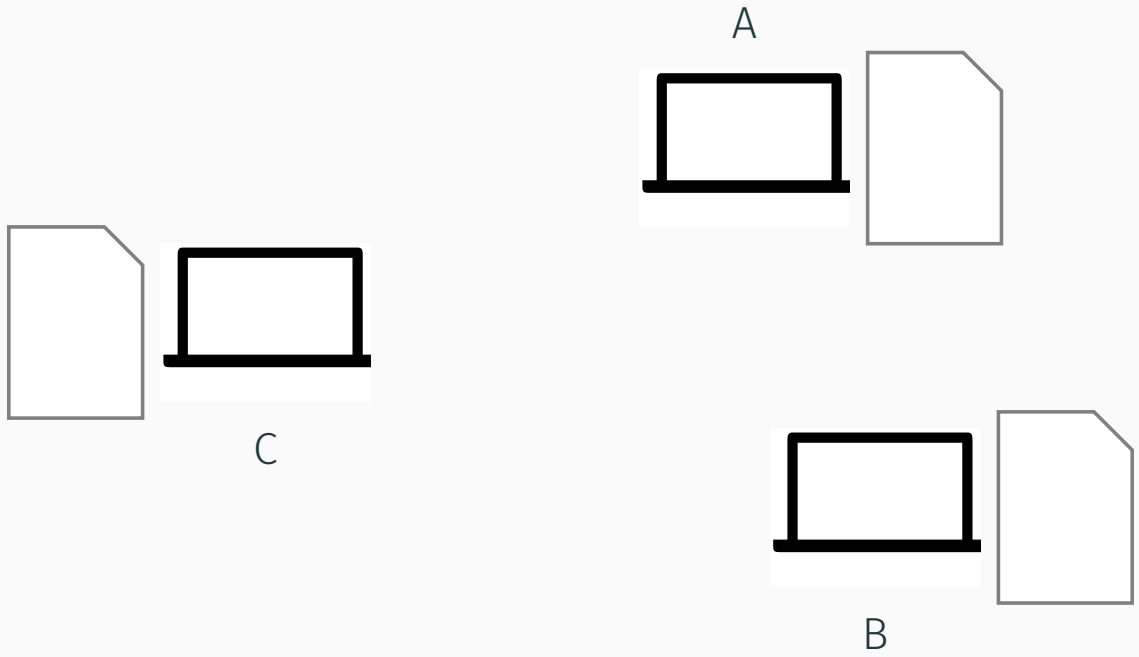
[1]. KLEPPMANN et al., « Local-First Software : You Own Your Data, in Spite of the Cloud ».

Réplication dans applications collaboratives pair-à-pair



[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

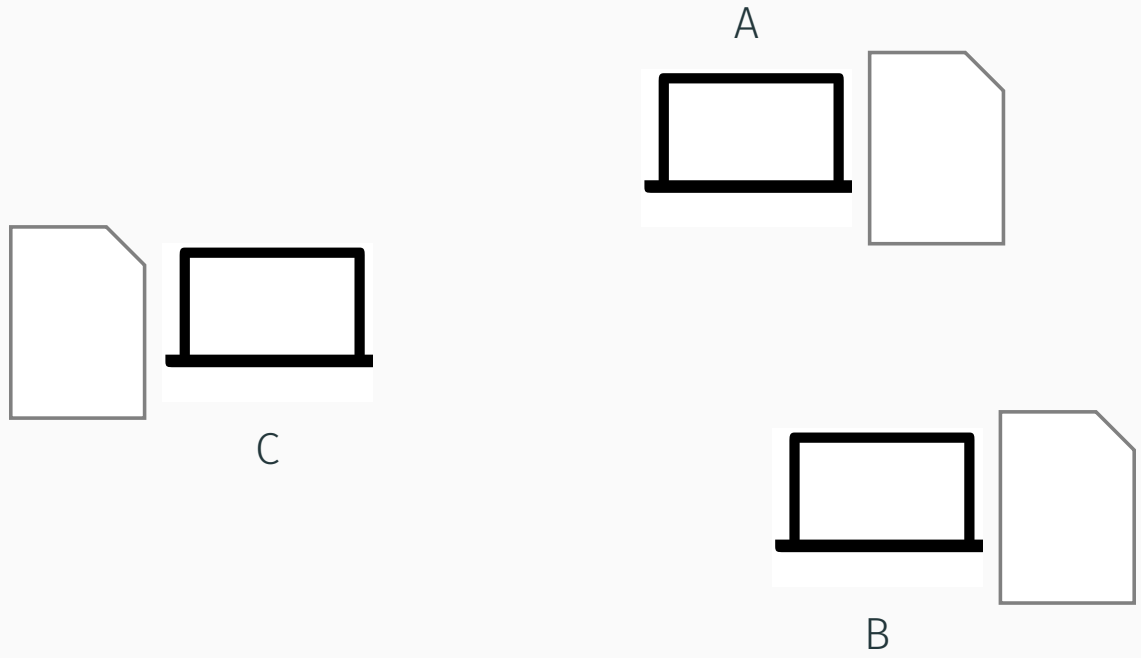
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés

[1]. TERRY et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

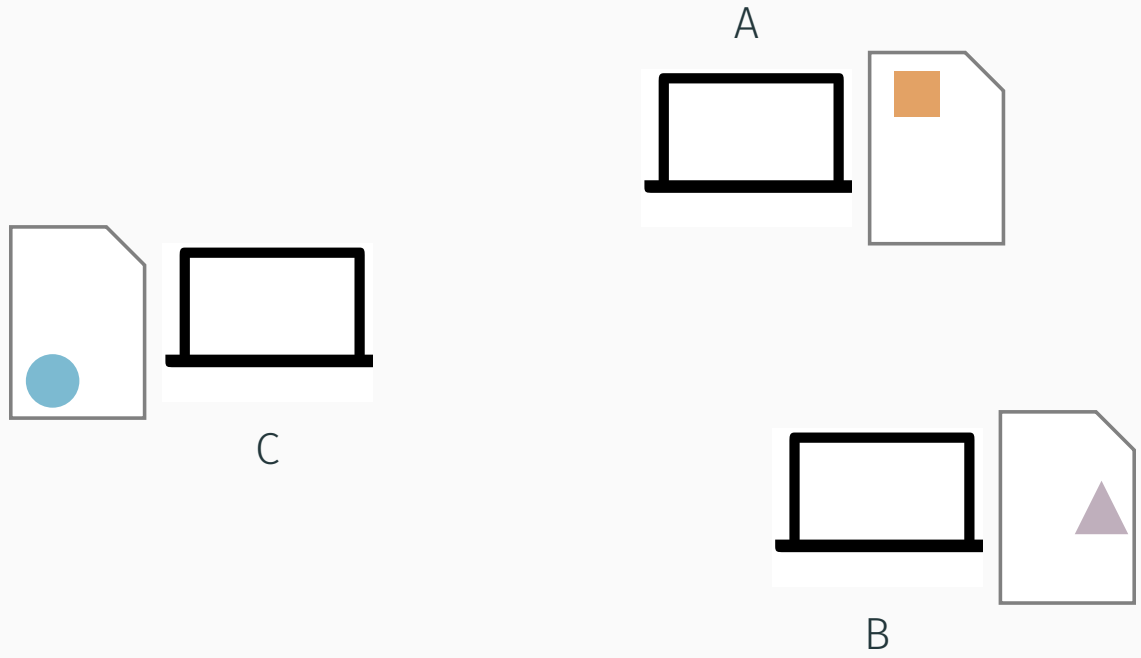
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

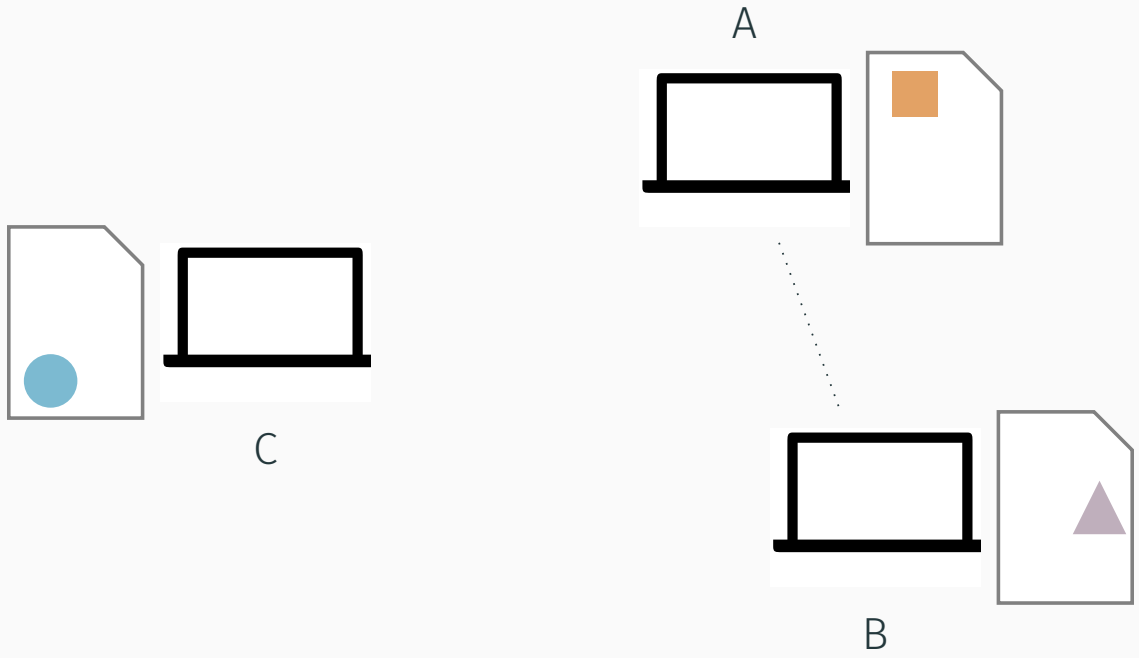
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

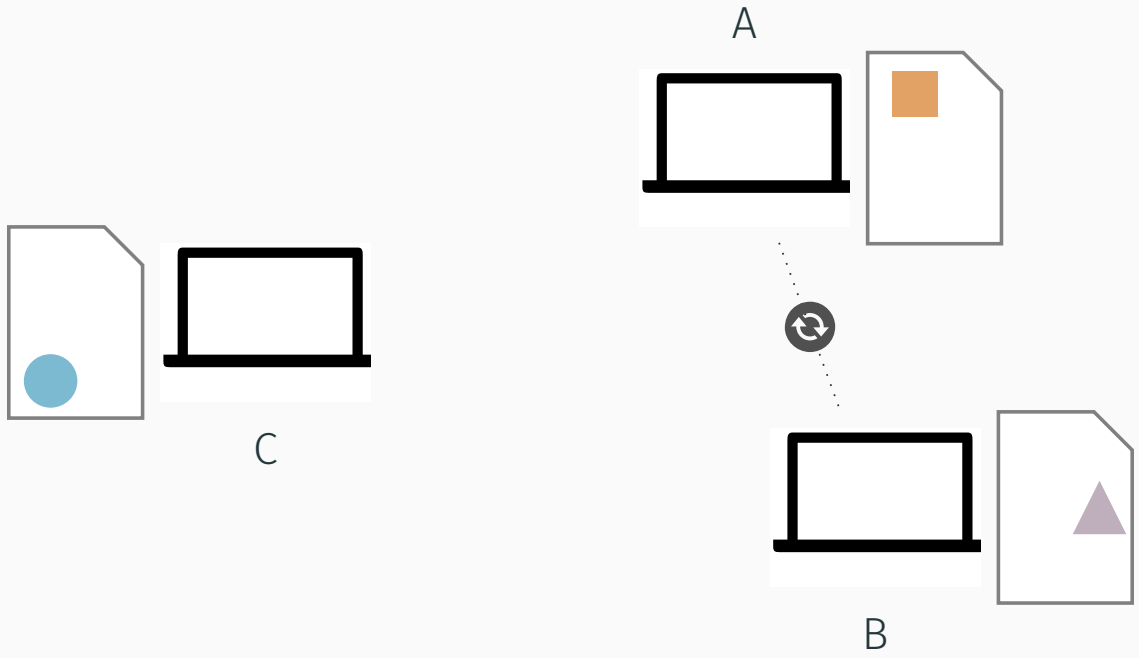
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

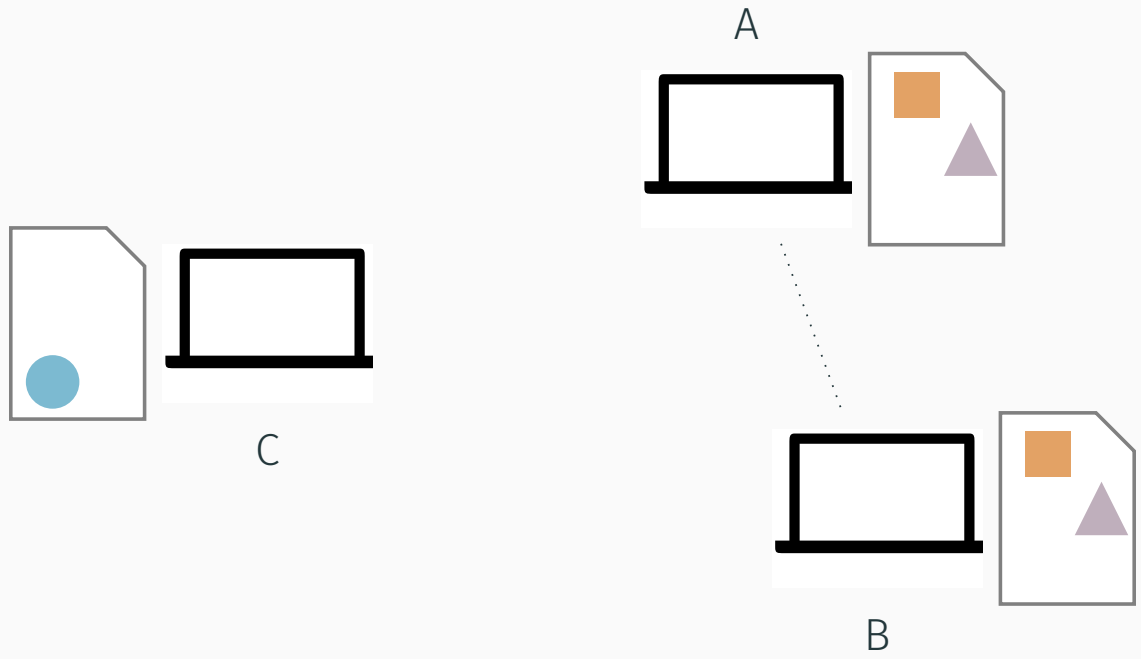
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

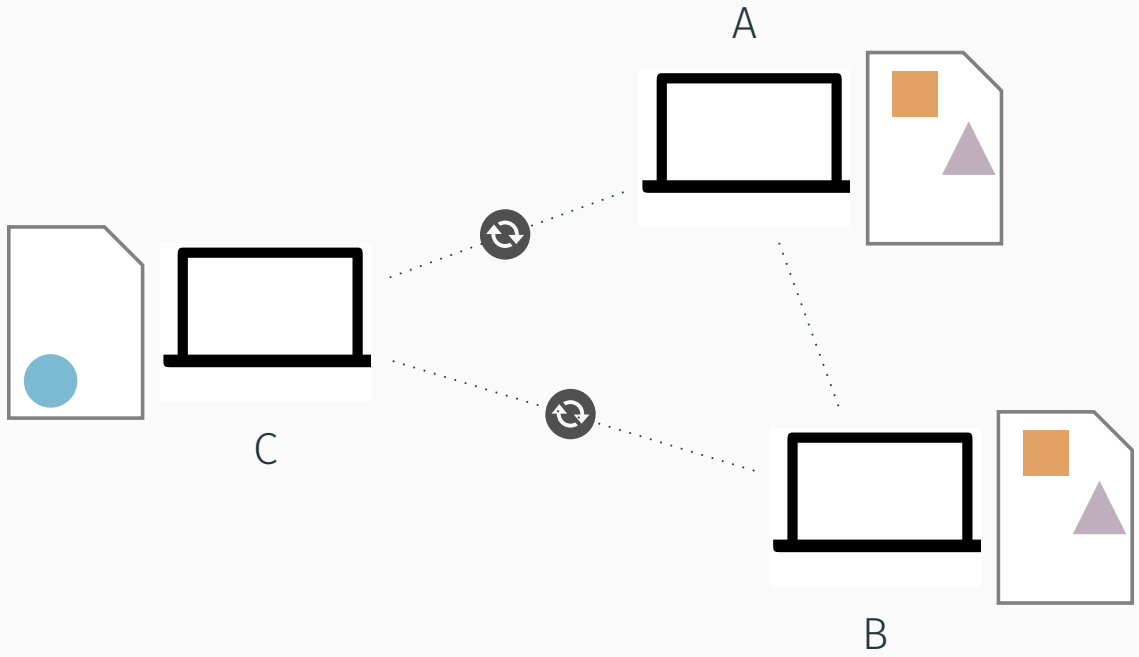
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

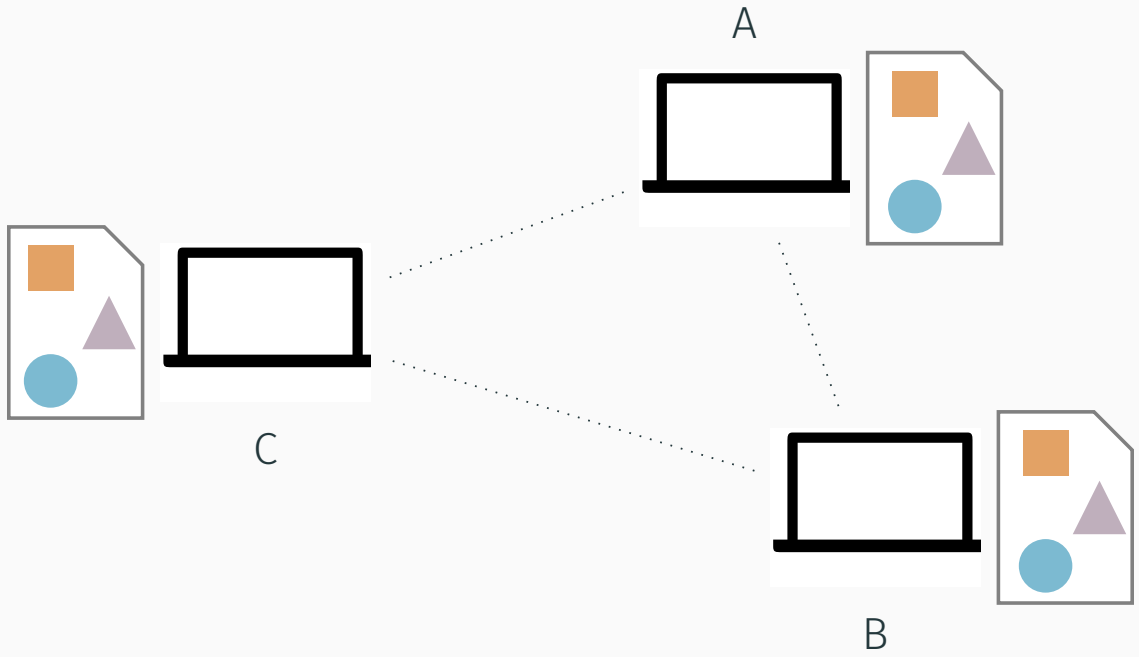
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

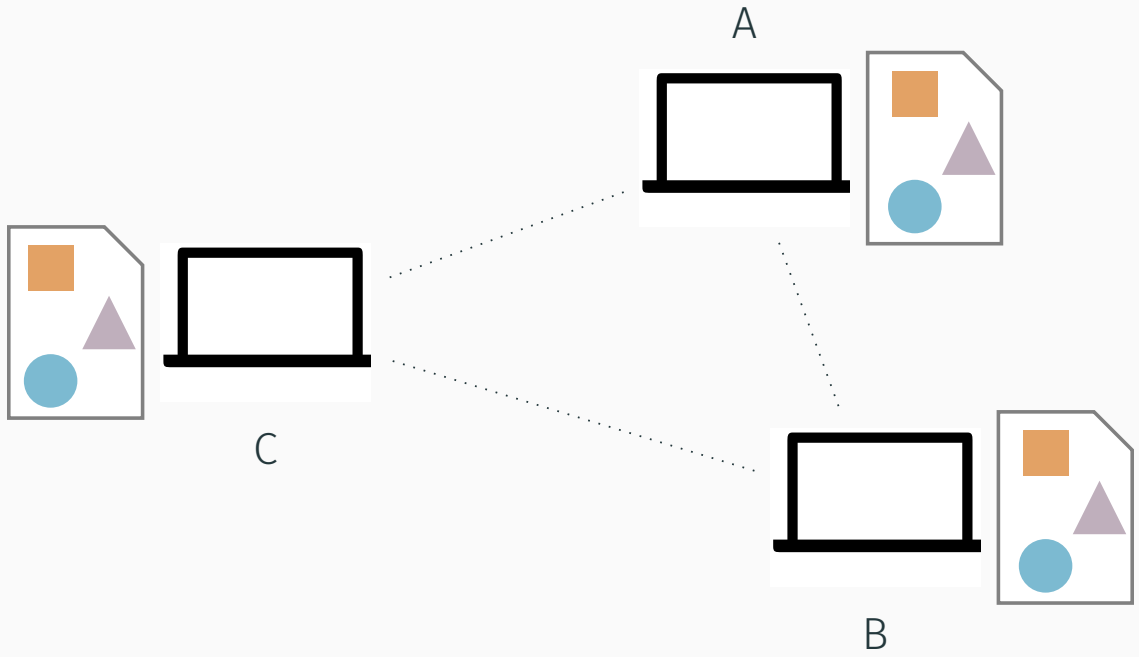
Réplication dans applications collaboratives pair-à-pair



- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)
- Doit garantir convergence à terme [1]...
- ...malgré ordres différents d'intégration des modifications

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

Réplication dans applications collaboratives pair-à-pair



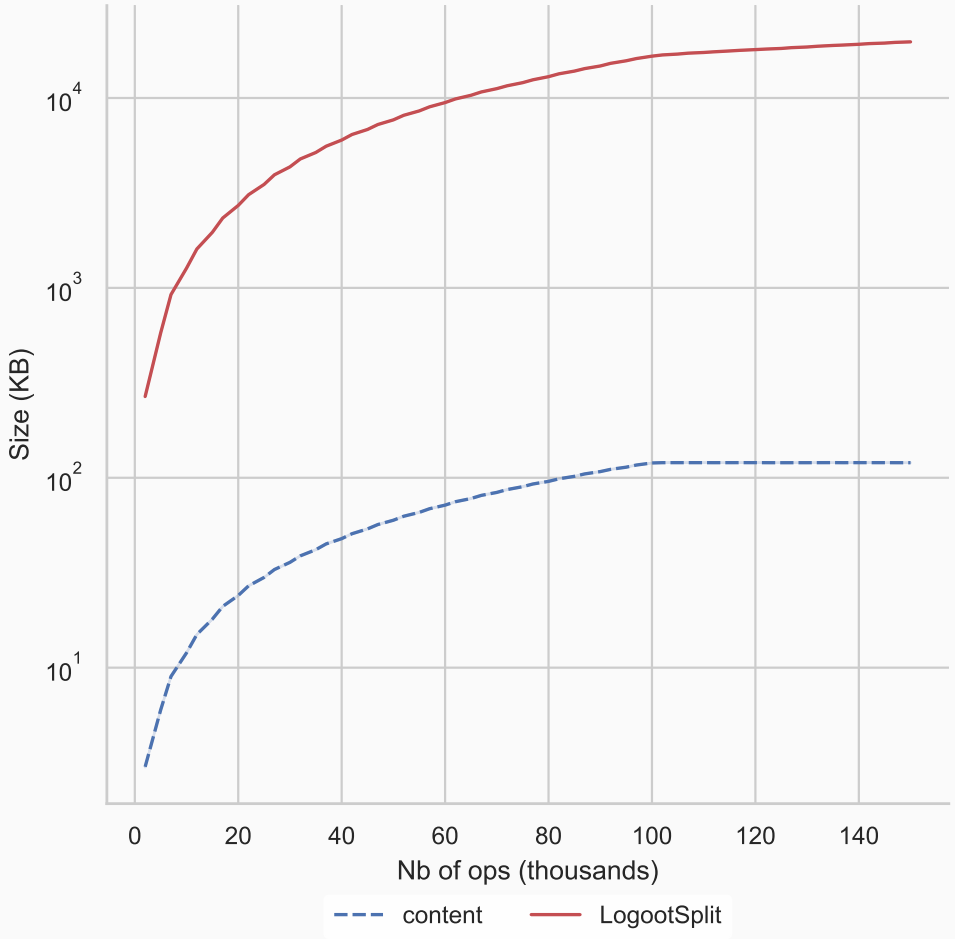
- Noeuds peuvent être déconnectés
- Doivent pouvoir travailler sans coordination synchrone préalable (par ex. consensus)
- Doit garantir convergence à terme [1]...
- ...malgré ordres différents d'intégration des modifications

Nécessite des *mécanismes de résolution de conflits*

[1]. Terry et al., « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System »

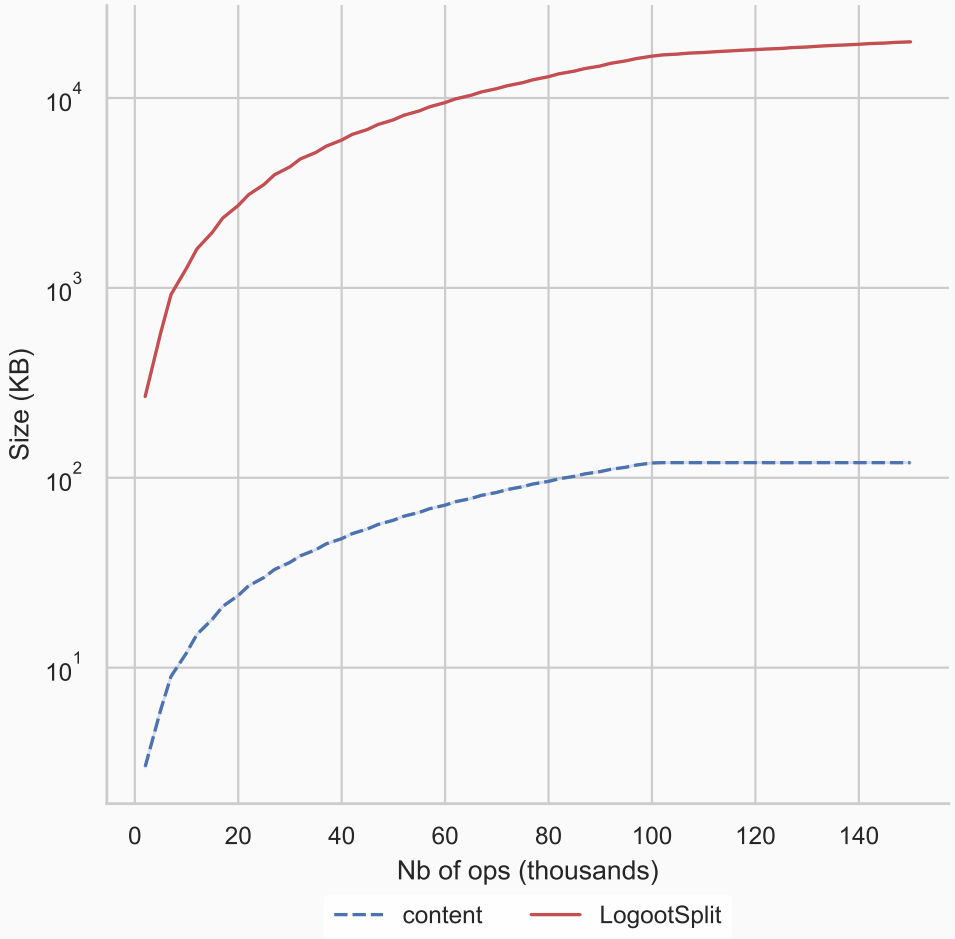
Évaluation de MUTE

Taille du texte comparée à taille de la séquence répliquée



Évaluation de MUTE

Taille du texte comparée à taille de la séquence répliquée

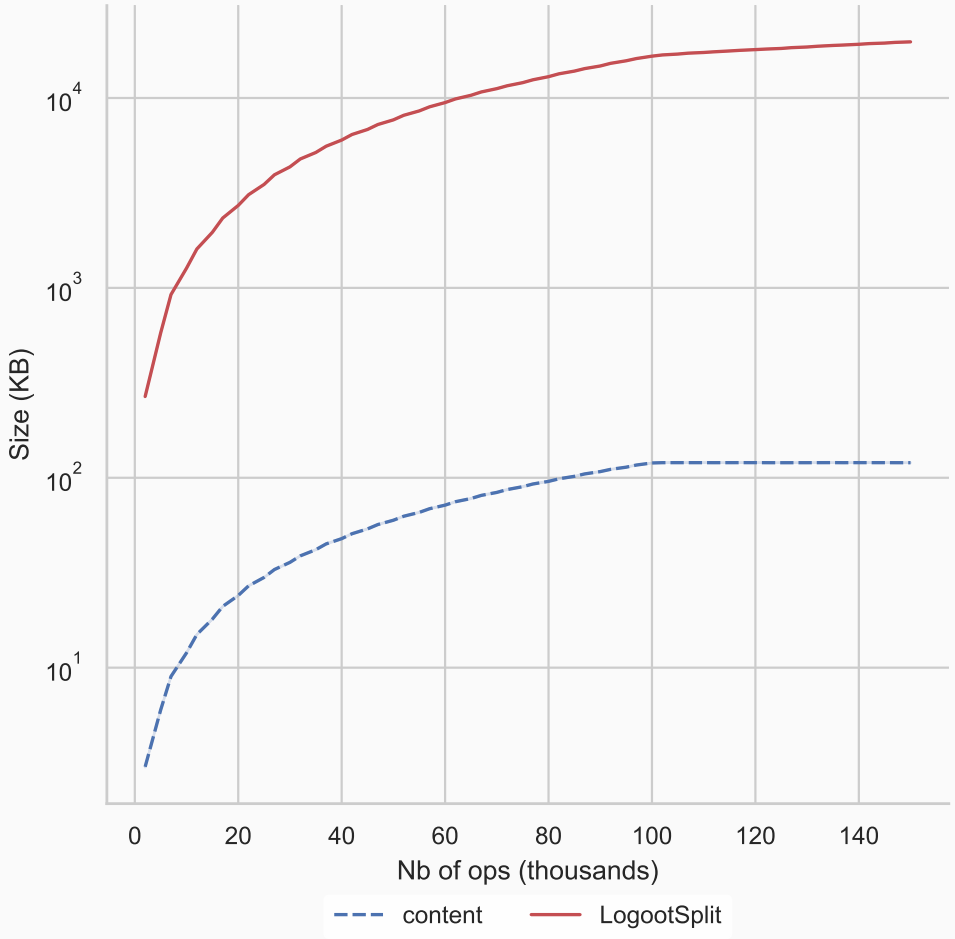


Constat

- 1% contenu...
- ...99% métadonnées

Évaluation de MUTE

Taille du texte comparée à taille de la séquence répliquée



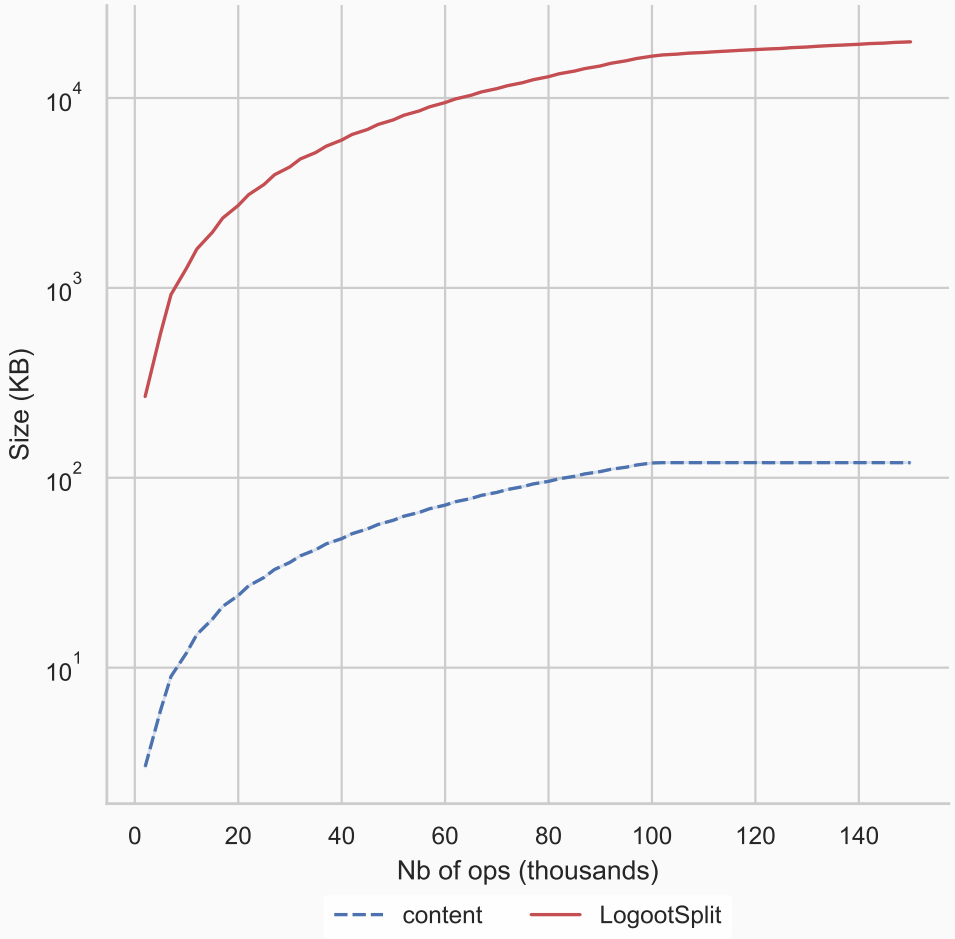
Constat

- 1% contenu...
- ...99% métadonnées

Et ça augmente !

Évaluation de MUTE

Taille du texte comparée à taille de la séquence répliquée



Constat

- 1% contenu...
- ...99% métadonnées

Et ça augmente !

Impact

- Surcoût mémoire...
- ...mais aussi surcoût en calculs et en bande-passante

Comment peut-on réduire le surcoût des mécanismes de résolution de conflits dans les applications pair-à-pair ?

Plan de la présentation

Plan

- L'origine de la croissance du surcoût des mécanismes de résolution de conflits pour le type Séquence

Plan de la présentation

Plan

- L'origine de la croissance du surcoût des mécanismes de résolution de conflits pour le type Séquence
- **Contribution** : Un mécanisme pair-à-pair de réduction du surcoût des mécanismes de résolution de conflits

Plan de la présentation

Plan

- L'origine de la croissance du surcoût des mécanismes de résolution de conflits pour le type Séquence
- **Contribution** : Un mécanisme pair-à-pair de réduction du surcoût des mécanismes de résolution de conflits
- Conclusion générale & perspectives

Conflict-free Replicated Data Types (CRDTs)[2]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Incorpore nativement mécanisme de résolution de conflits

[2]. Shapiro et al., « Conflict-Free Replicated Data Types ».

Conflict-free Replicated Data Types (CRDTs) [2]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Incorpore nativement mécanisme de résolution de conflits

Propriétés des CRDTs

- Permettent modifications sans coordination
- Garantissent la convergence forte

[2]. Shapiro et al., « Conflict-Free Replicated Data Types ».

Conflict-free Replicated Data Types (CRDTs) [2]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Incorpore nativement mécanisme de résolution de conflits

Propriétés des CRDTs

- Permettent modifications sans coordination
- Garantissent la convergence forte

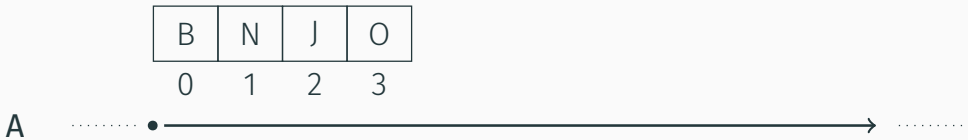
Convergence forte

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

[2]. Shapiro et al., « Conflict-Free Replicated Data Types ».

CRDTs pour le type Séquence

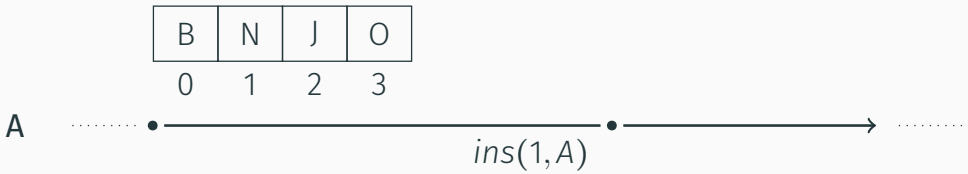
Type Séquence usuel



[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

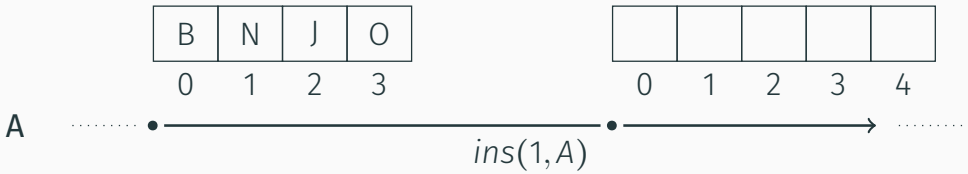
Type Séquence usuel



[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

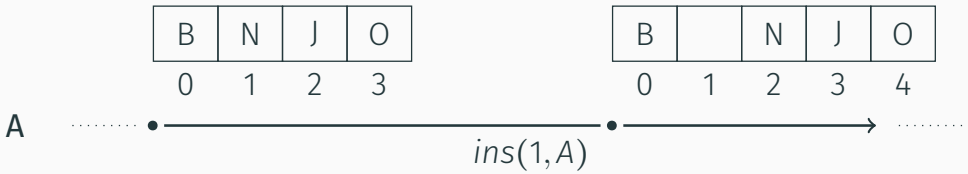
Type Séquence usuel



[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

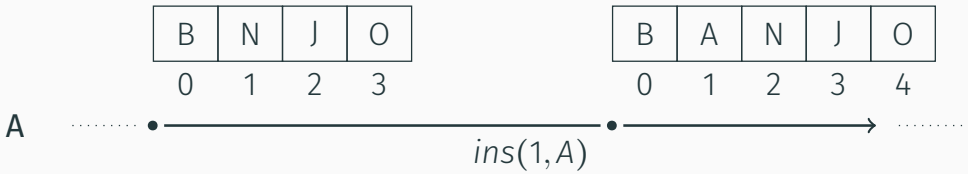
Type Séquence usuel



[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

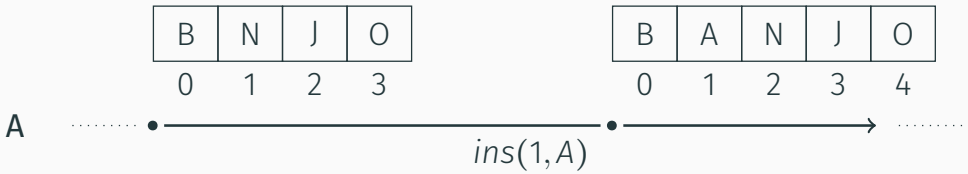
Type Séquence usuel



[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel

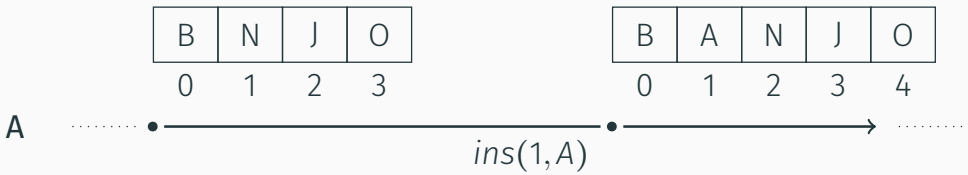


- Changements des indices est source de conflits

[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel



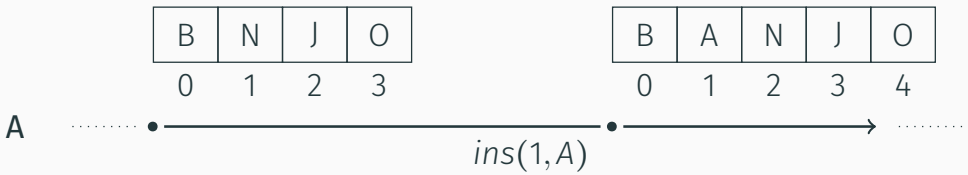
CRDTs pour Séquence

- Changements des indices est source de conflits
- Assignent des identifiants de position [3] à chaque élément
- Permettent d'ordonner les élements

[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel



CRDTs pour Séquence

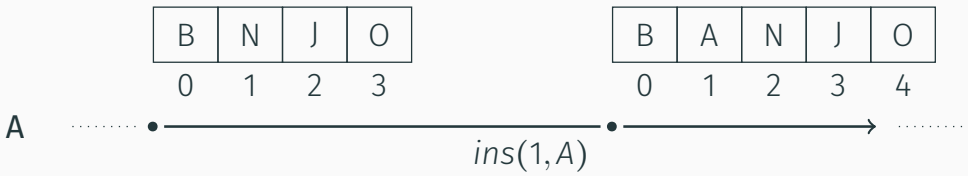
- Changements des indices est source de conflits
- Assignent des identifiants de position [3] à chaque élément
- Permettent d'ordonner les élements

$$id_0 <_{id} id_1 <_{id} id_2 <_{id} id_3$$

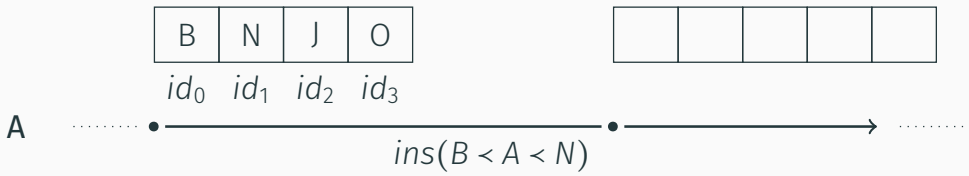
[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel



CRDTs pour Séquence



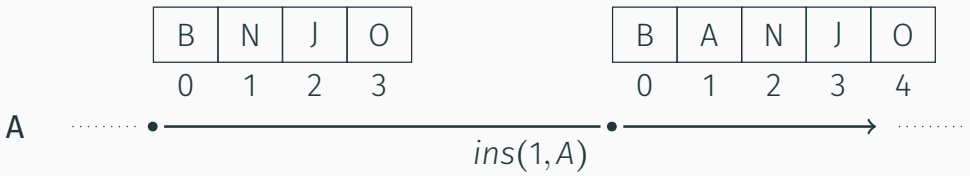
- Changements des indices est source de conflits
- Assignent des identifiants de position [3] à chaque élément
- Permettent d'ordonner les élements

$$id_0 <_{id} id_1 <_{id} id_2 <_{id} id_3$$

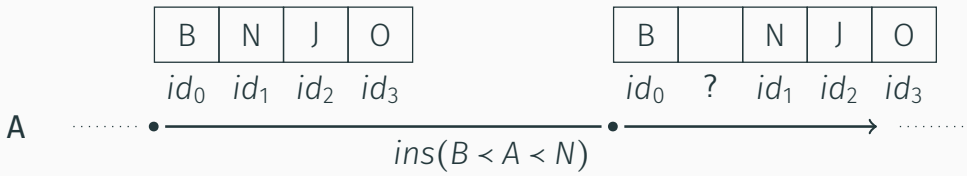
[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel



CRDTs pour Séquence



- Changements des indices est source de conflits
- Assignent des identifiants de position [3] à chaque élément
- Permettent d'ordonner les élements

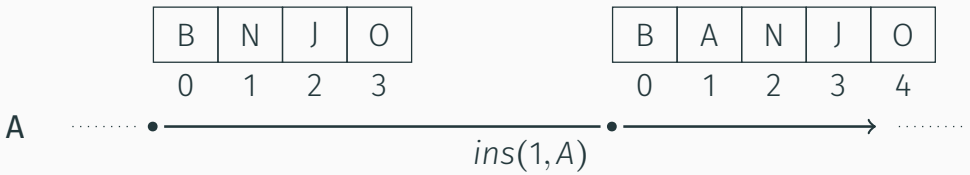
$$id_0 <_{id} id_1 <_{id} id_2 <_{id} id_3$$

- Appartiennent à un espace dense

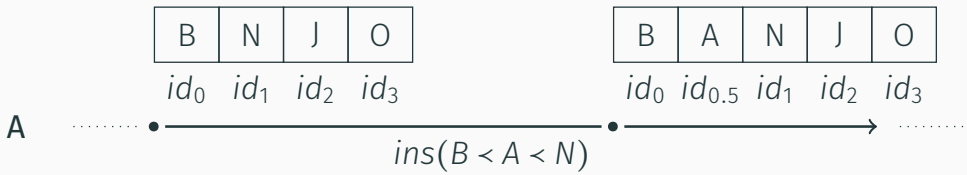
[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel



CRDTs pour Séquence



- Changements des indices est source de conflits
- Assignent des identifiants de position [3] à chaque élément
- Permettent d'ordonner les élements

$$id_0 <_{id} id_1 <_{id} id_2 <_{id} id_3$$

- Appartiennent à un espace dense

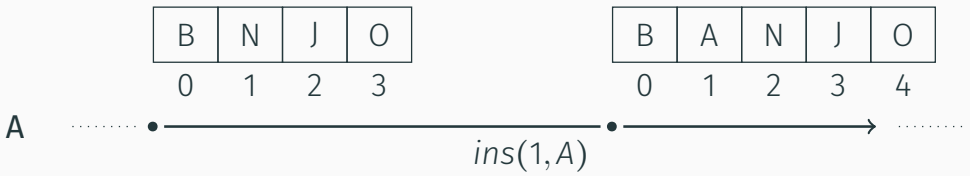
$$id_0 <_{id} id_{0.5} <_{id} id_1$$

---

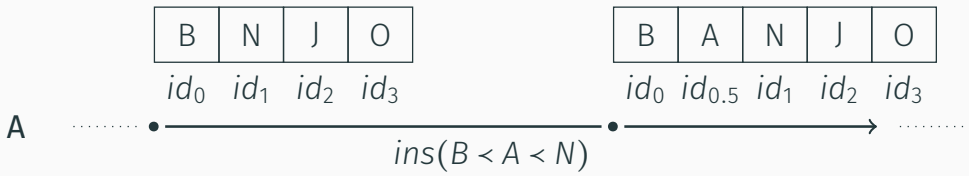
[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

CRDTs pour le type Séquence

Type Séquence usuel



CRDTs pour Séquence



- Changements des indices est source de conflits
- Assignent des identifiants de position [3] à chaque élément
- Permettent d'ordonner les élements

$$id_0 <_{id} id_1 <_{id} id_2 <_{id} id_3$$

- Appartiennent à un espace dense

$$id_0 <_{id} id_{0.5} <_{id} id_1$$

Utilise LogootSplit [4] comme base

[3]. Preguica et al., « A Commutative Replicated Data Type for Cooperative Editing ».
[4]. André et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

Identifiant LogootSplit

Identifiant

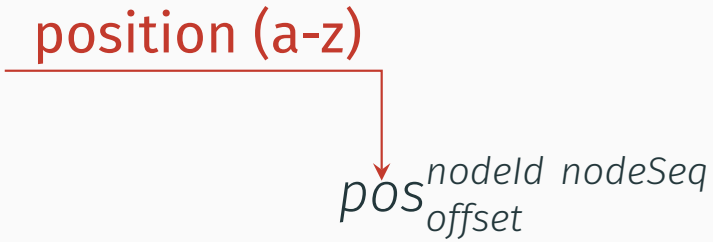
- Composé d'un ou plusieurs tuples de la forme

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

Identifiant LogootSplit

Identifiant

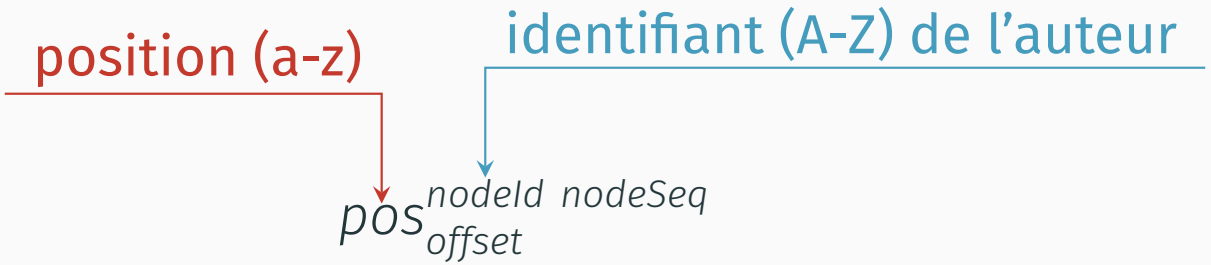
- Composé d'un ou plusieurs tuples de la forme



Identifiant LogootSplit

Identifiant

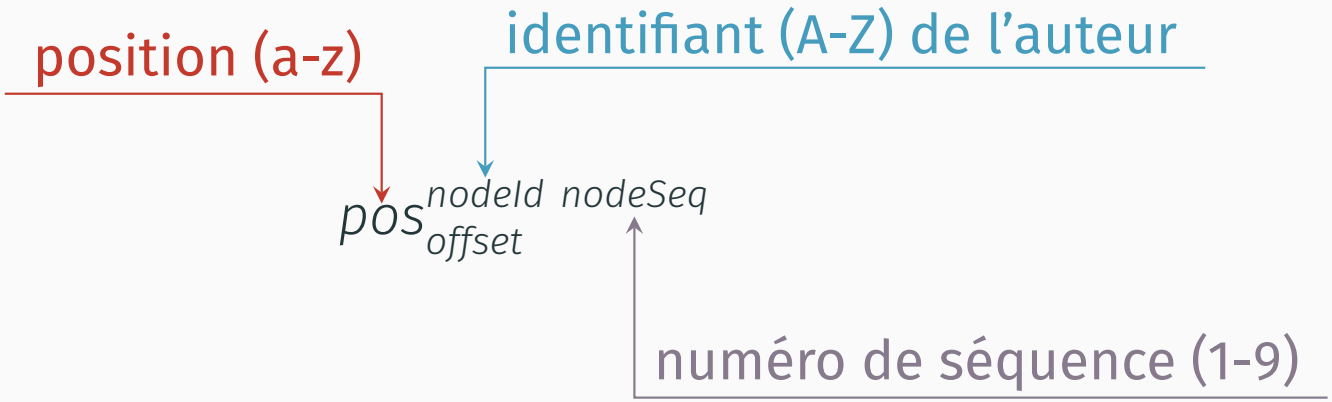
- Composé d'un ou plusieurs tuples de la forme



Identifiant LogootSplit

Identifiant

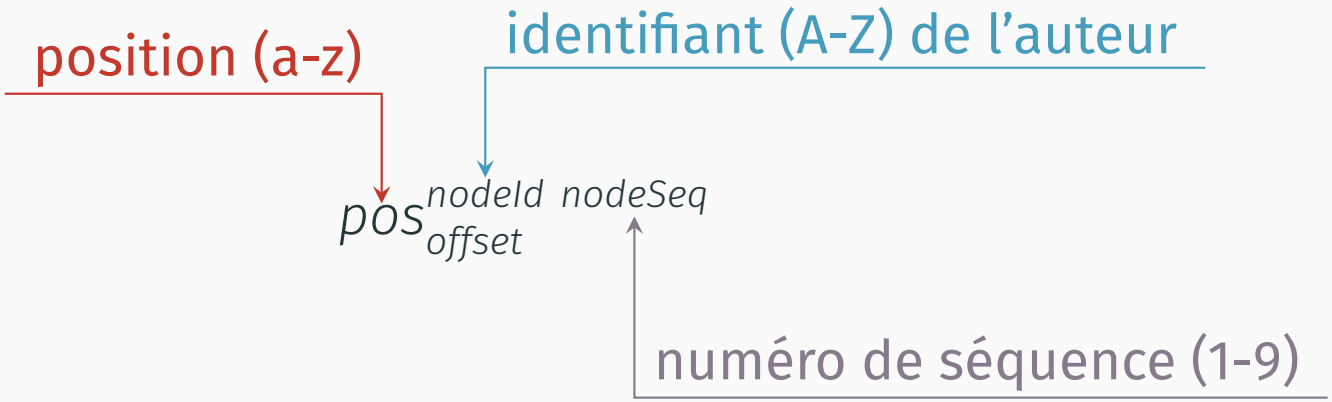
- Composé d'un ou plusieurs tuples de la forme



Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples de la forme



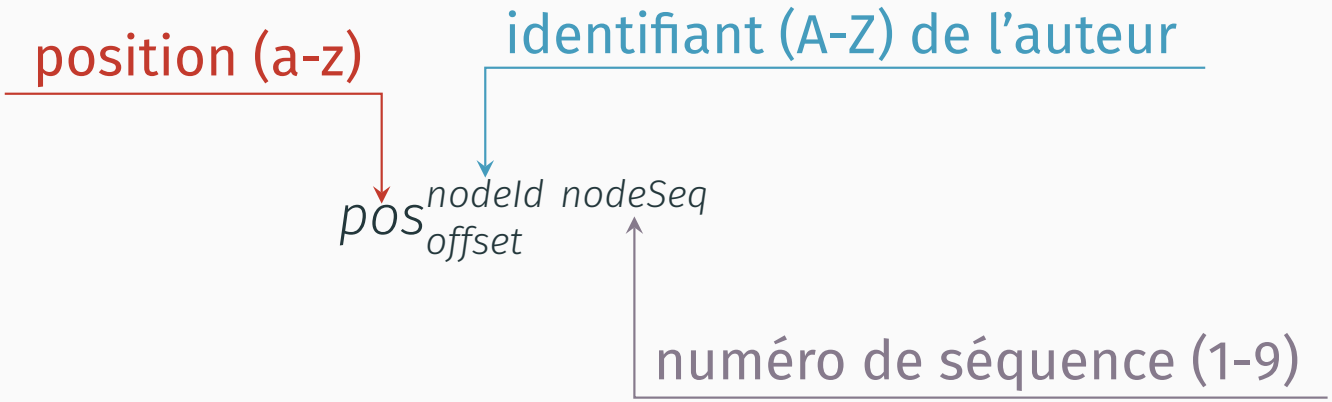
Exemples

d_0^{F5}

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples de la forme



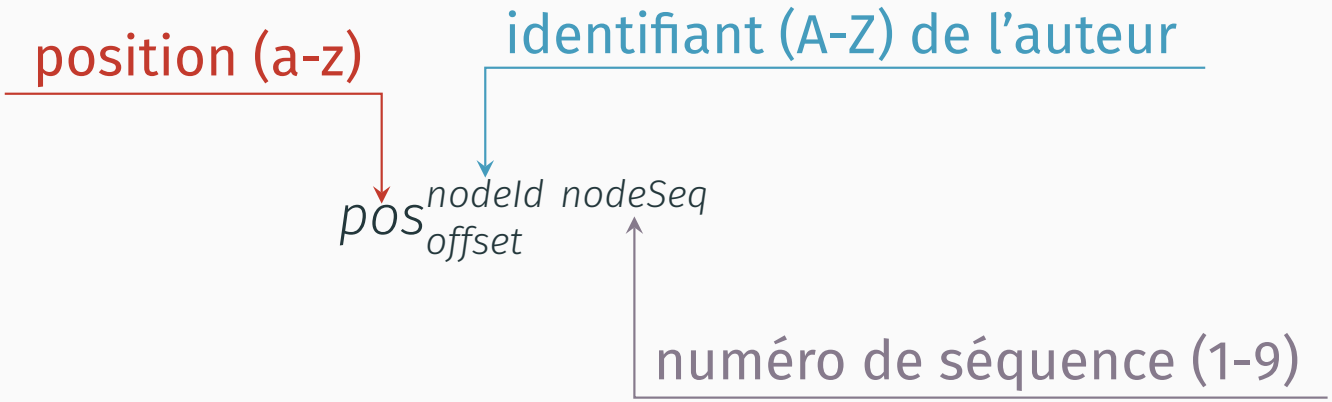
Exemples

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

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples de la forme



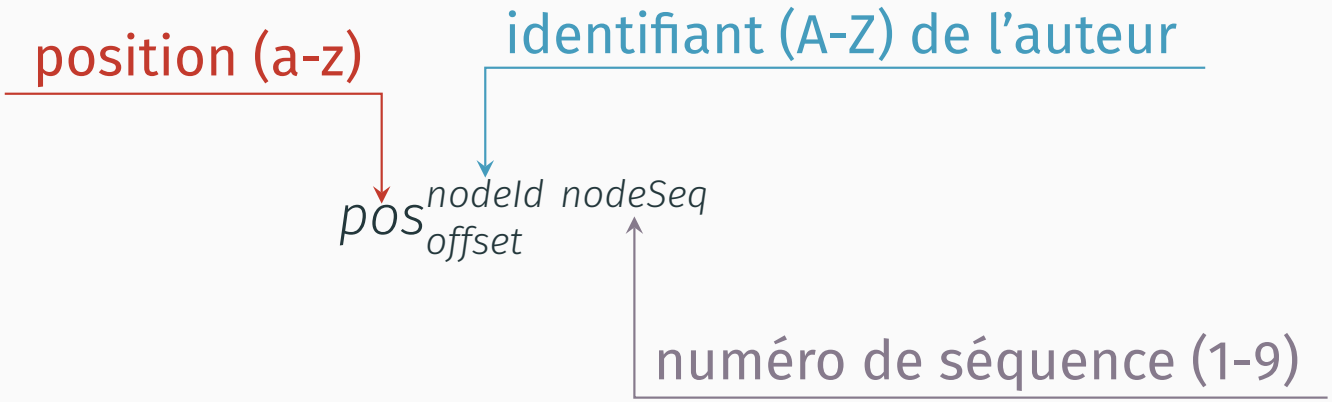
Exemples

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

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples de la forme



Exemples

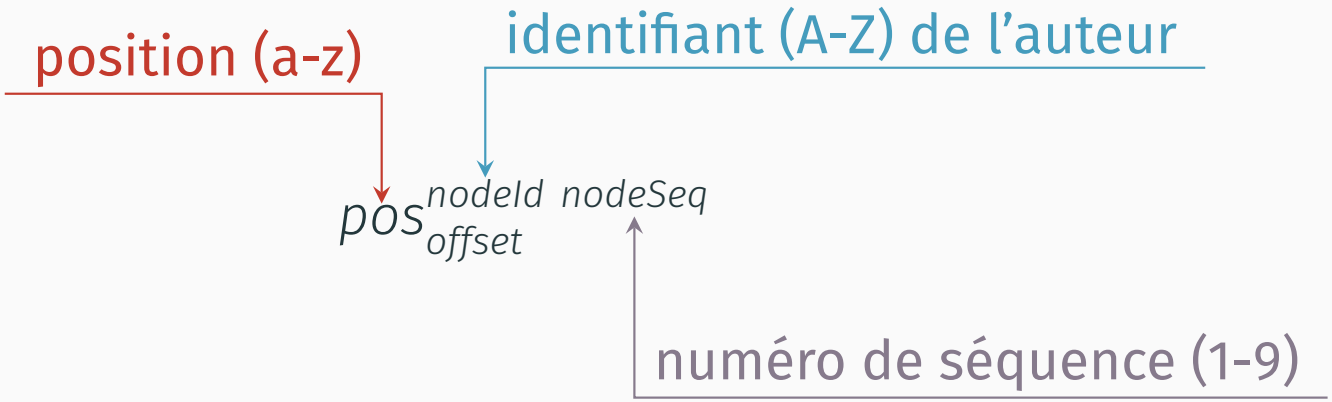
$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

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

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples de la forme



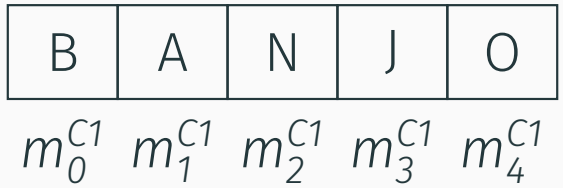
Exemples

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

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

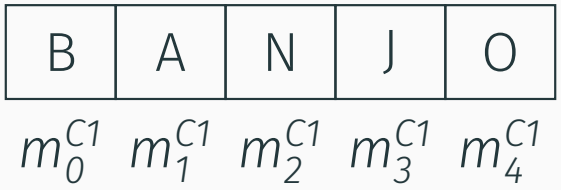
Bloc LogootSplit

- Coûteux de stocker les identifiants de chaque élément



Bloc LogootSplit

- Coûteux de stocker les identifiants de chaque élément



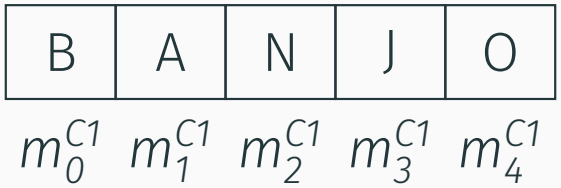
- Aggrège en un bloc éléments ayant identifiants contigus

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



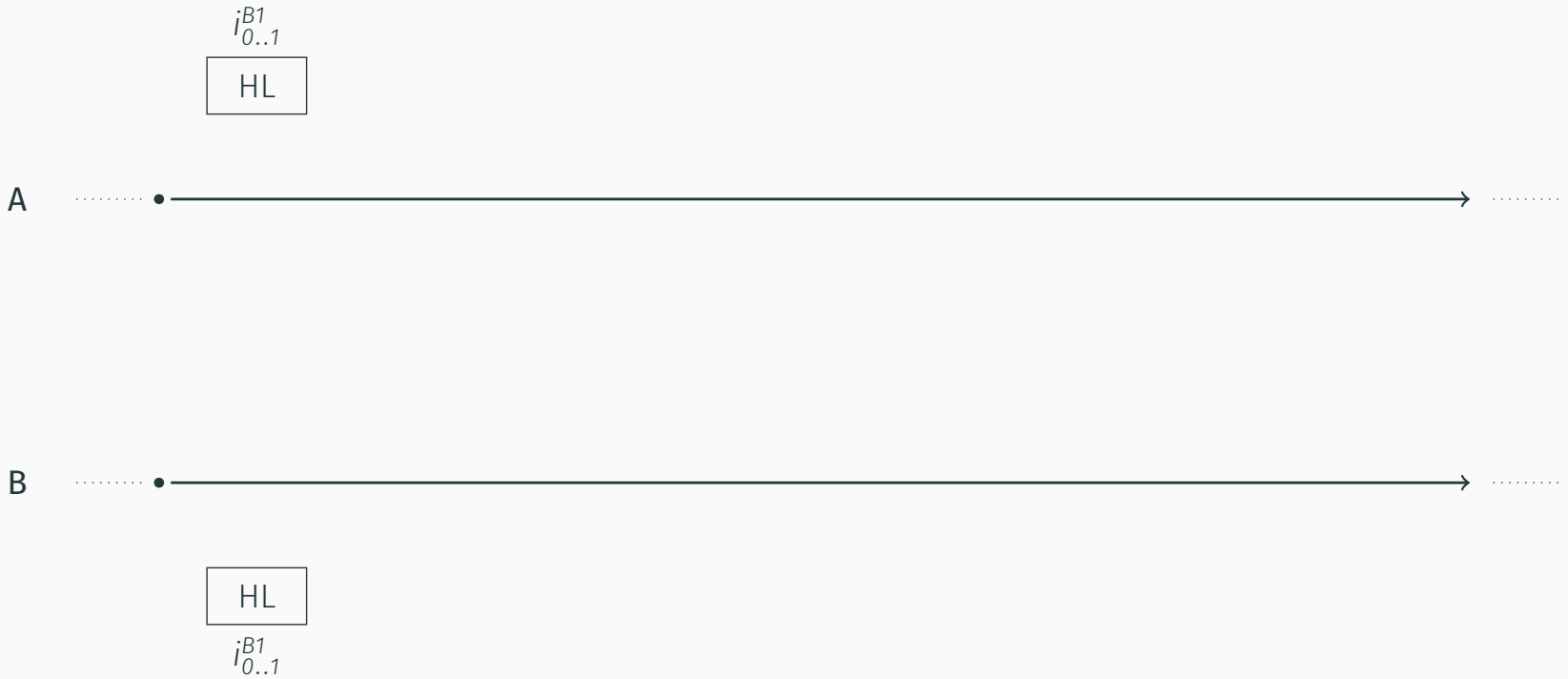
- Aggrège en un bloc éléments ayant identifiants contigus

Identifiants contigus

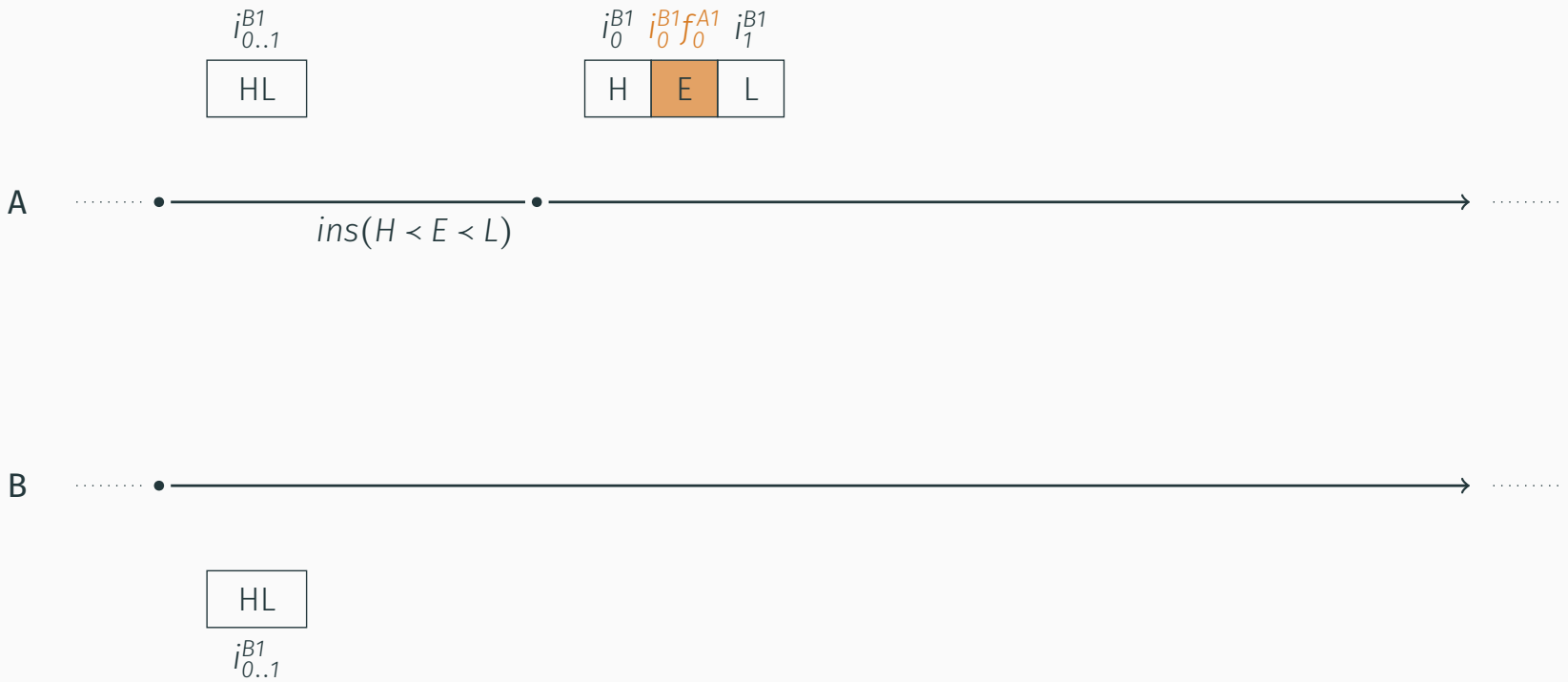
Deux identifiants sont contigus si et seulement si les deux identifiants sont identiques à l'exception de leur dernier offset et que leur derniers offsets sont consécutifs.

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

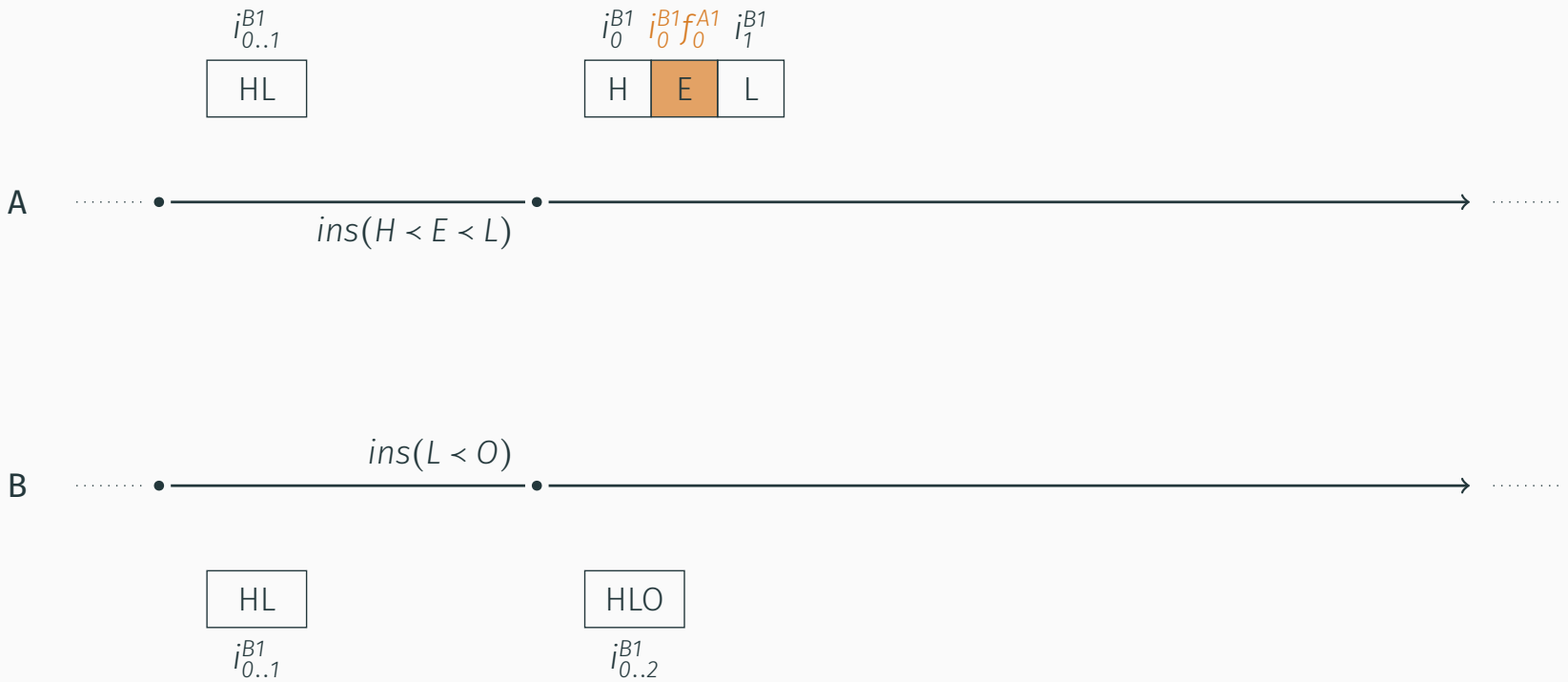
Exemple insertions concurrentes



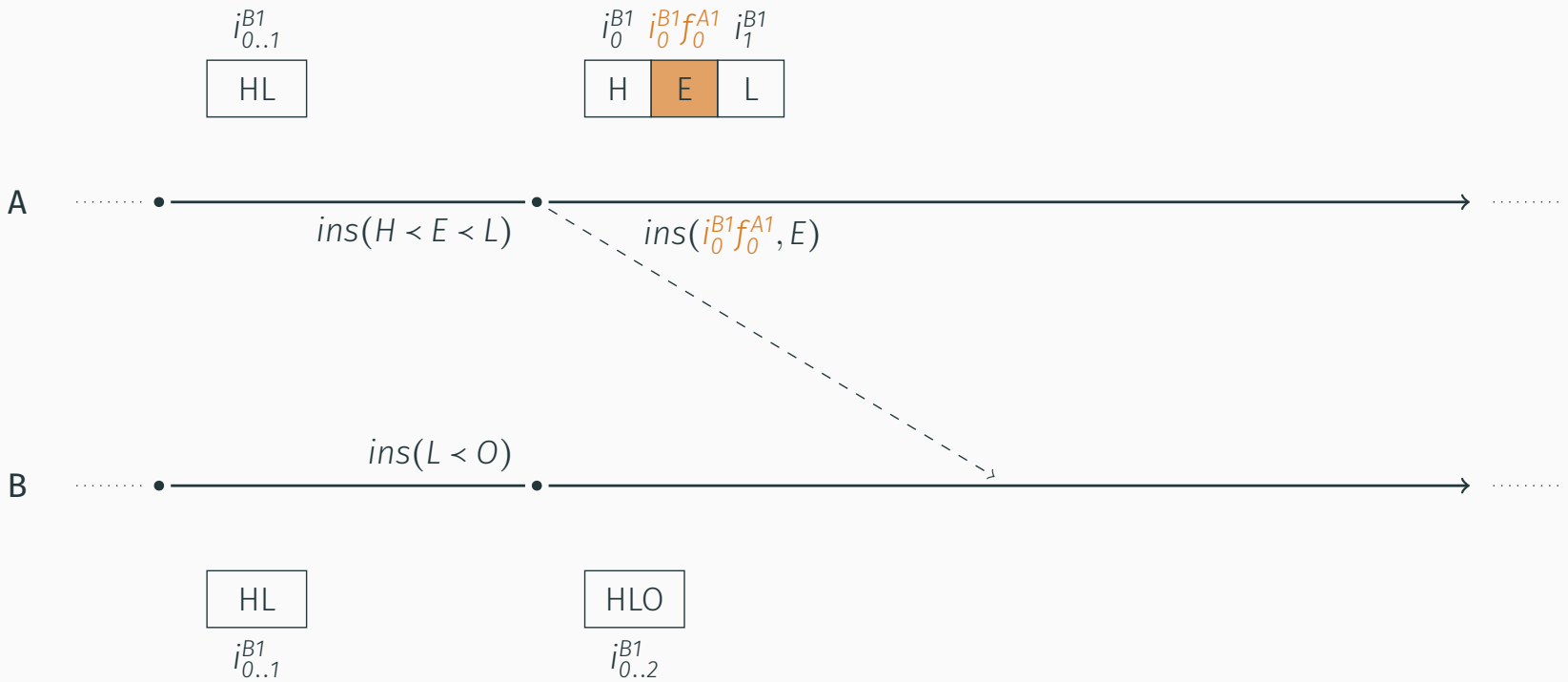
Exemple insertions concurrentes



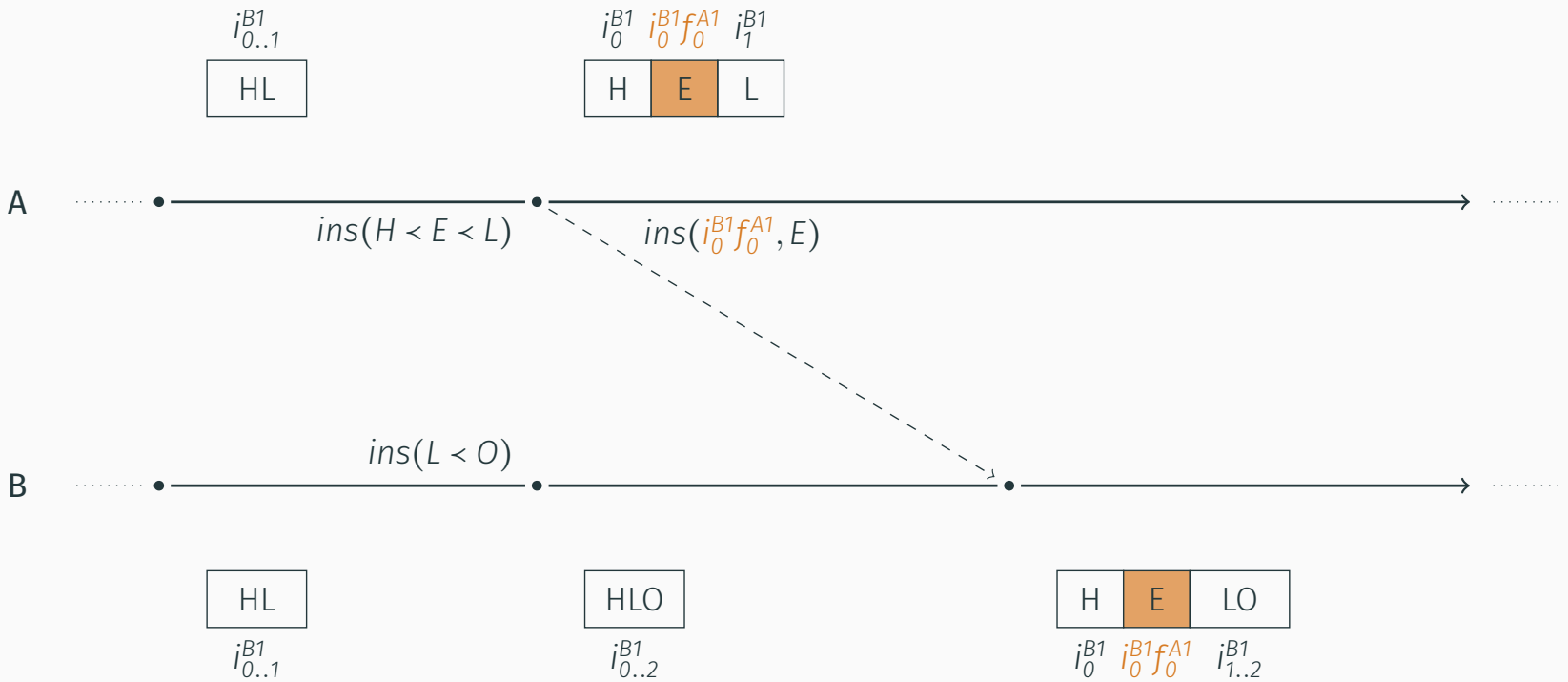
Exemple insertions concurrentes



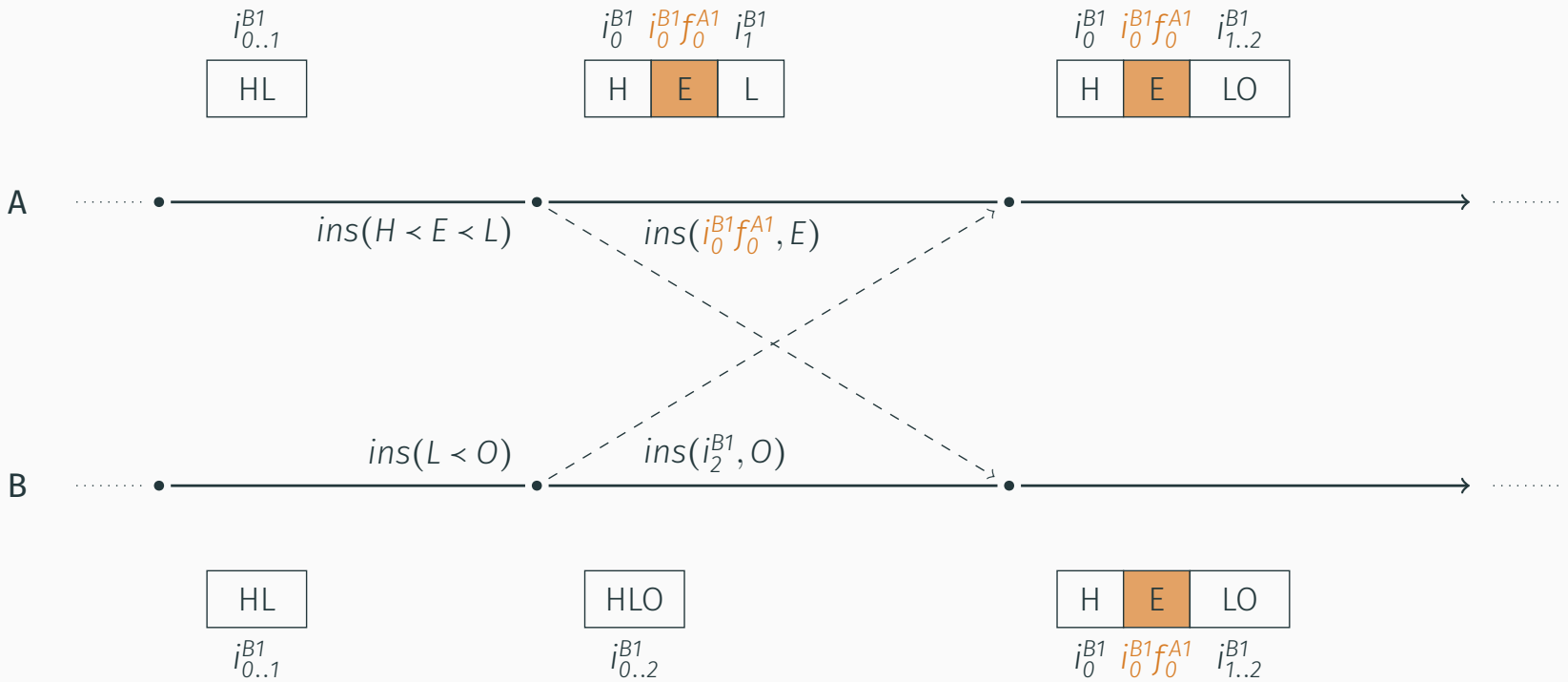
Exemple insertions concurrentes



Exemple insertions concurrentes



Exemple insertions concurrentes



Limites de LogootSplit

Sources de la croissance des métadonnées

- Augmentation non-bornée de la taille des identifiants
- Fragmentation de la séquence en un nombre croissant de blocs

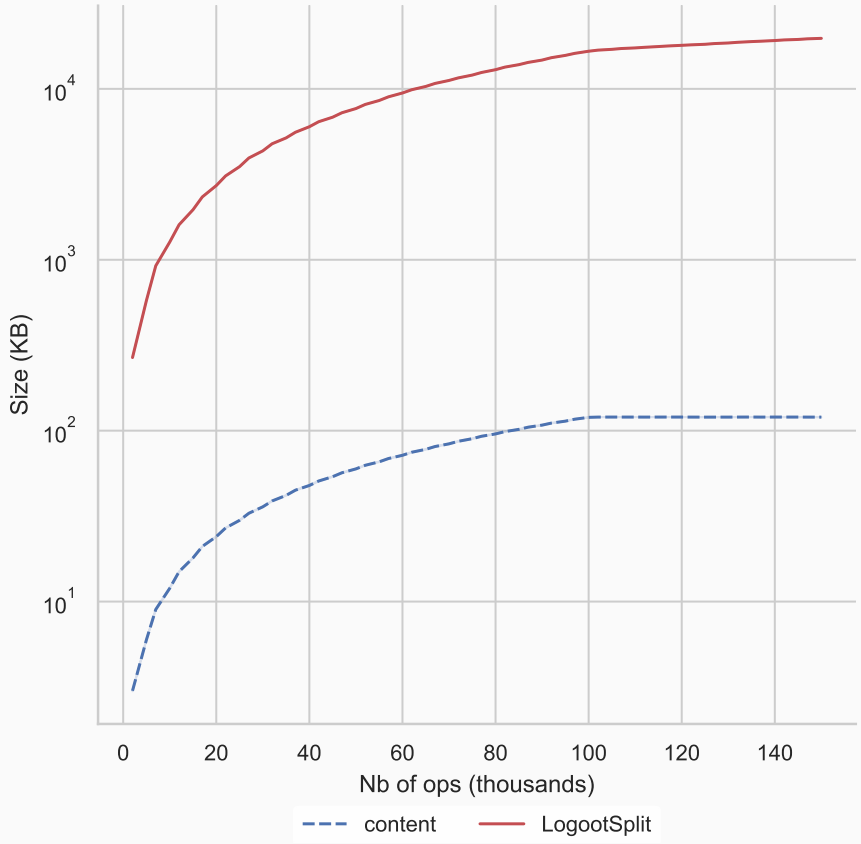
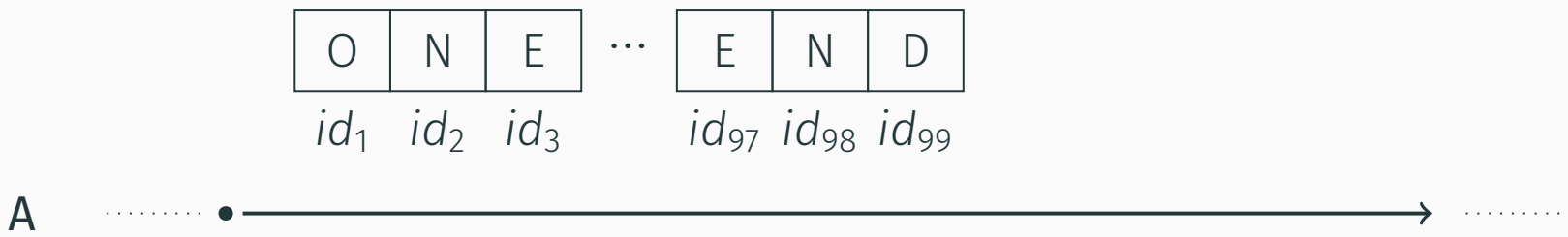


Figure 1 – Taille du contenu comparée à la taille de la séquence LogootSplit

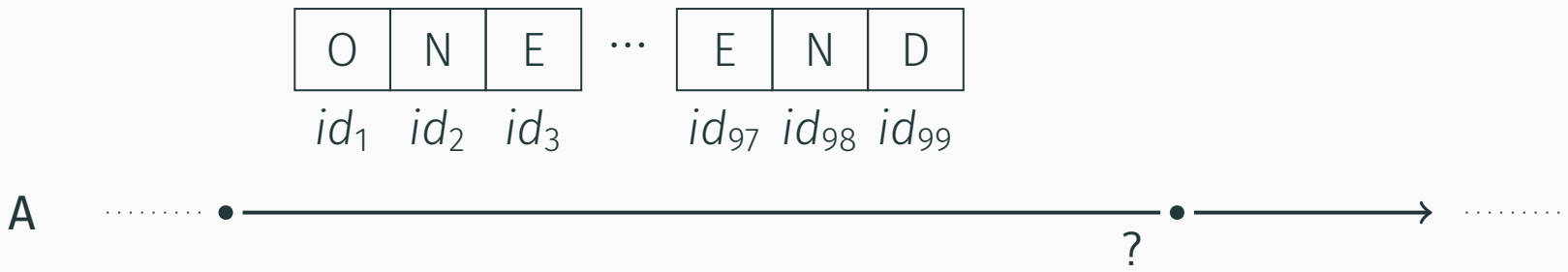
Diminution des performances du point de vue mémoire, calculs et bande-passante

Solution naïve



- Convertir l'état inefficient...

Solution naïve



- Convertir l'état inefficient...
- ...à l'aide d'une nouvelle opération ...

Solution naïve

- Convertir l'état inefficient...
- ...à l'aide d'une nouvelle opération ...
- ...en un état optimisé (identifiants de taille minimale, moins de blocs)

Mitigation du surcoût des CRDTs pour le type Séquence

L'approche core-nebula [5], pour Treedoc

- Ré-assigne des identifiants plus courts aux éléments
- Transforme les opérations *insert* et *remove* concurrentes…

[5]. Zawirski et al., « Asynchronous rebalancing of a replicated tree ».

Mitigation du surcoût des CRDTs pour le type Séquence

L'approche core-nebula [5], pour Treedoc

- Ré-assigne des identifiants plus courts aux éléments
- Transforme les opérations *insert* et *remove* concurrentes…
- …mais ne supporte pas opérations *rename* concurrentes
- Repose sur un algorithme de consensus pour décider du renommage

[5]. Zawirski et al., « Asynchronous rebalancing of a replicated tree ».

Mitigation du surcoût des CRDTs pour le type Séquence

L'approche core-nebula [5], pour Treedoc

- Ré-assigne des identifiants plus courts aux éléments
- Transforme les opérations *insert* et *remove* concurrentes…
- …mais ne supporte pas opérations *rename* concurrentes
- Repose sur un algorithme de consensus pour décider du renommage

Inadaptée aux applications pair-à-pair

[5]. Zawirski et al., « Asynchronous rebalancing of a replicated tree ».

Proposition

Mécanisme de renommage supportant les renommages concurrents

RenamableLogootSplit

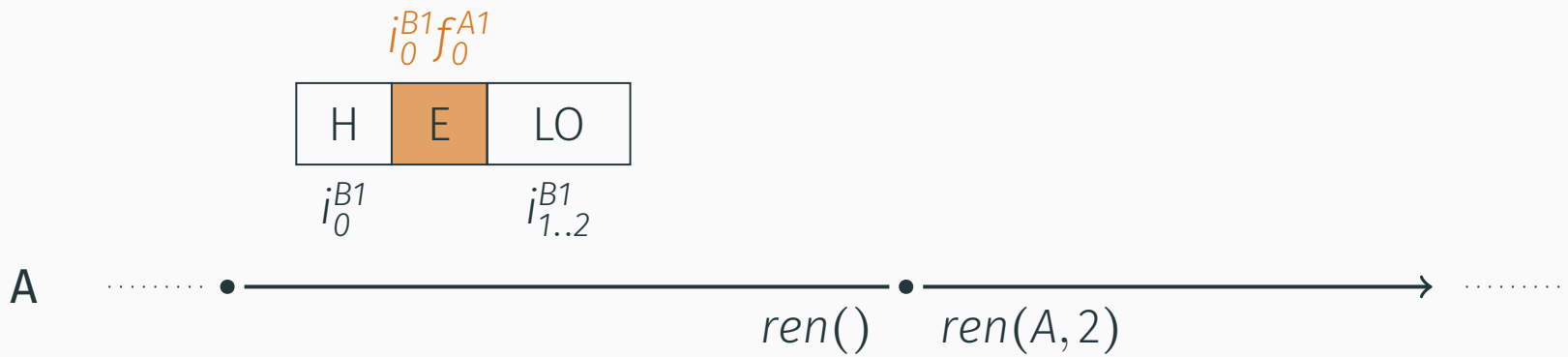
Contribution : RenamableLogootSplit

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

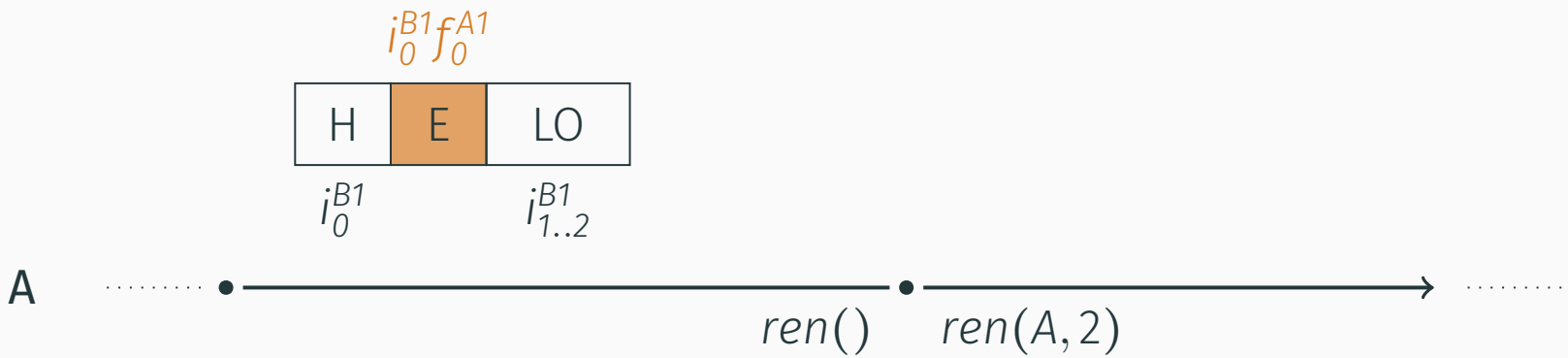
Propriétés de l'opération *rename*

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

Opération *rename*

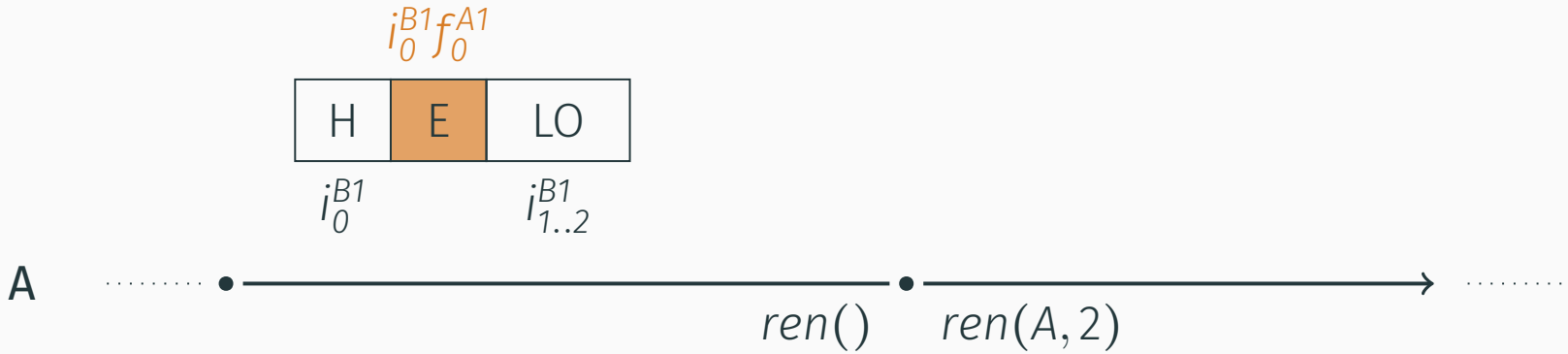


Opération *rename*



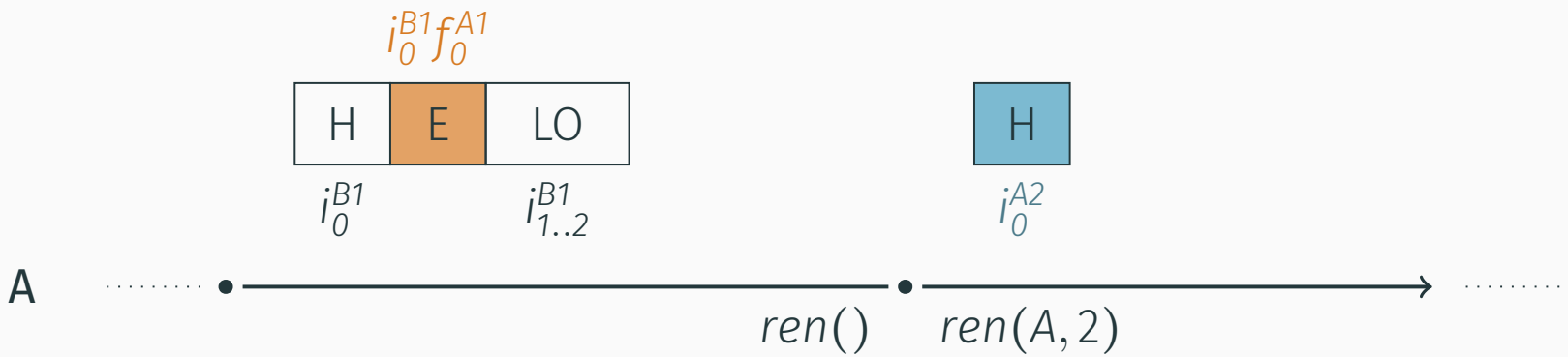
- Génère nouvel identifiant pour le 1er élément :

Opération *rename*



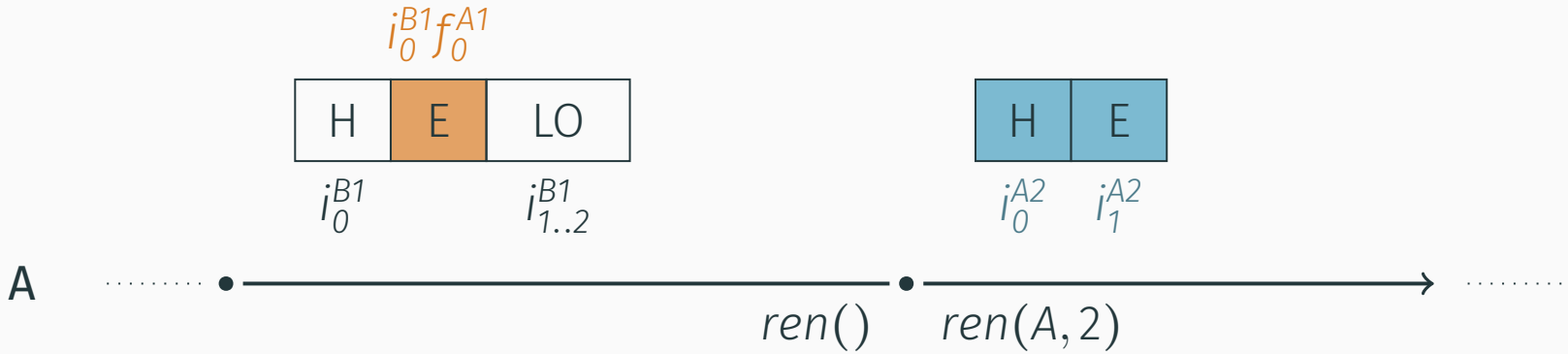
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$

Opération *rename*



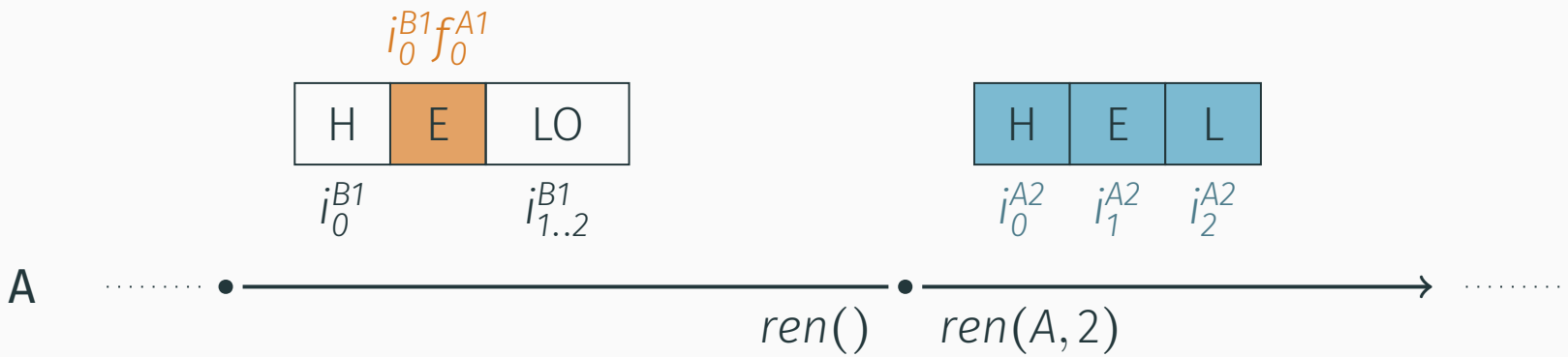
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants :

Opération *rename*



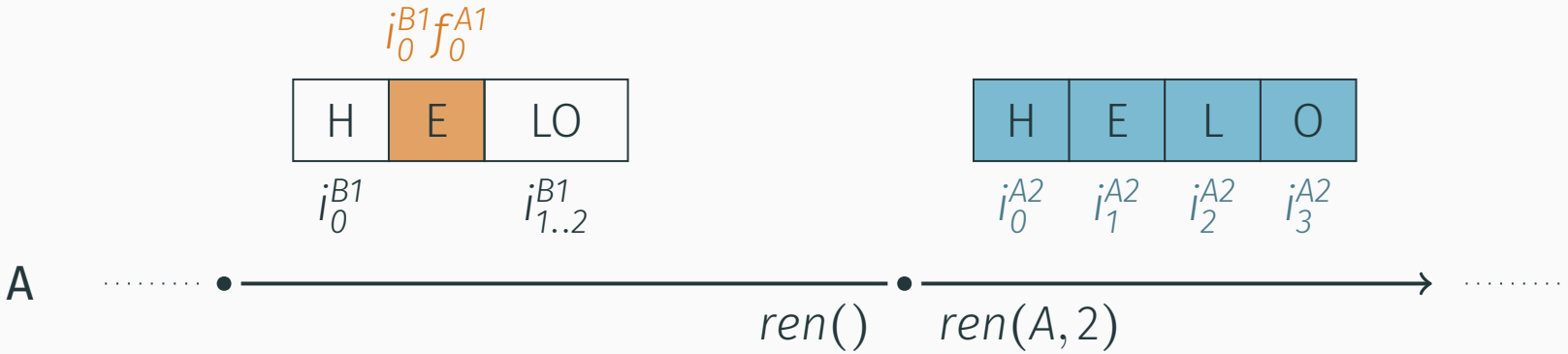
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2}

Opération *rename*



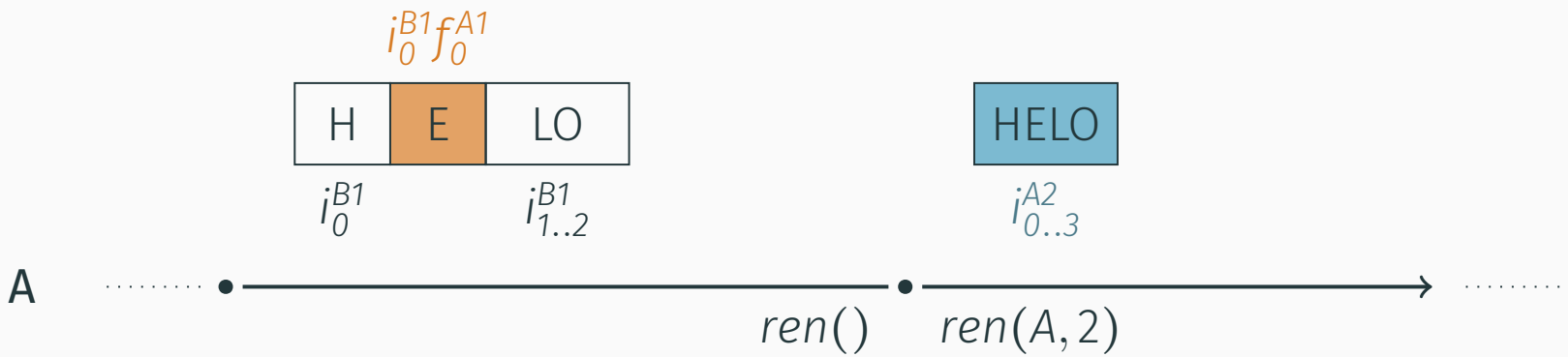
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2}, i_2^{A2}

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}, ...

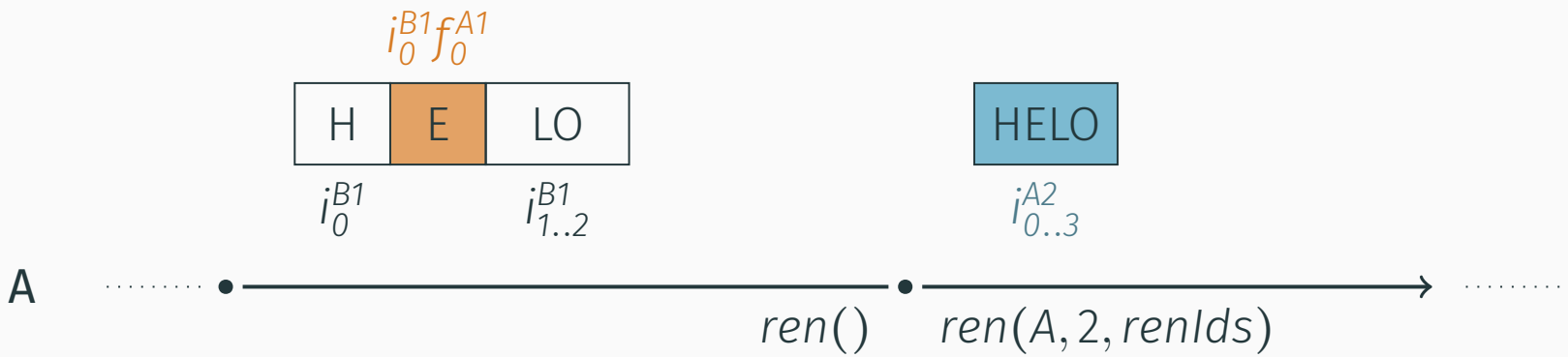
Opération *rename*



- 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}, ...

Regroupe tous les éléments en 1 unique bloc

Opération *rename*



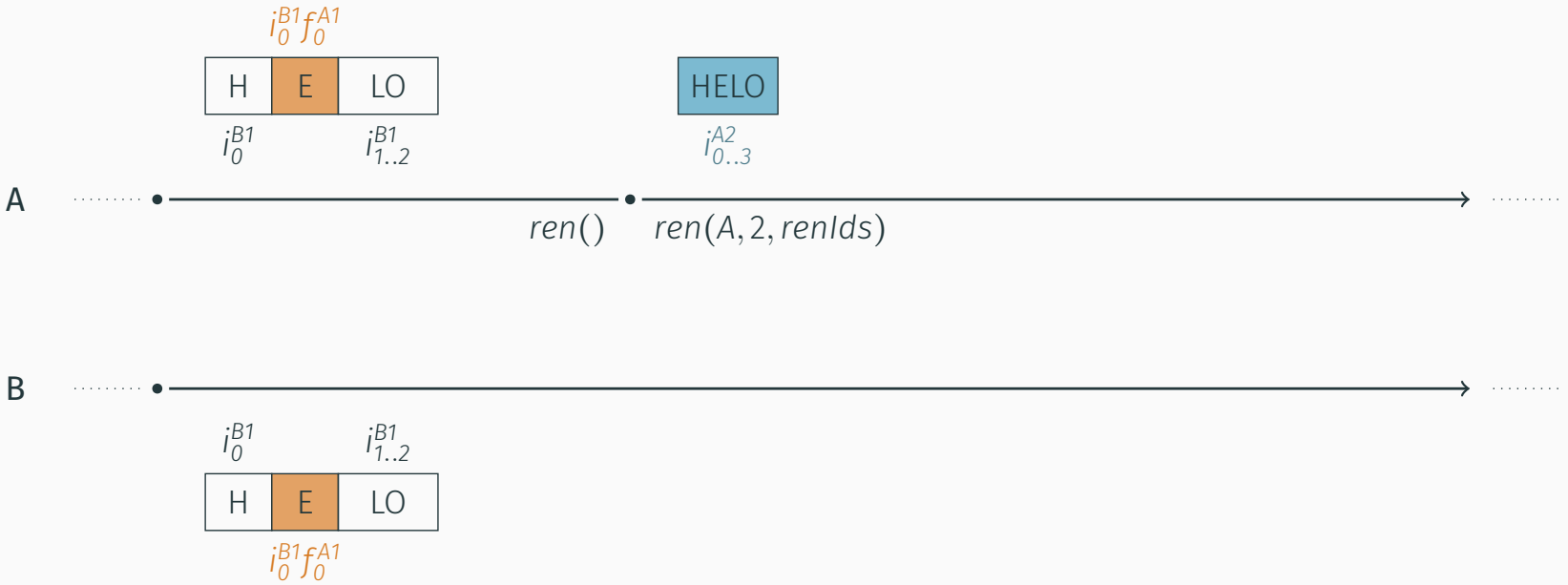
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2}, i_2^{A2}, ...

Regroupe tous les éléments en 1 unique bloc

Pour plus tard :

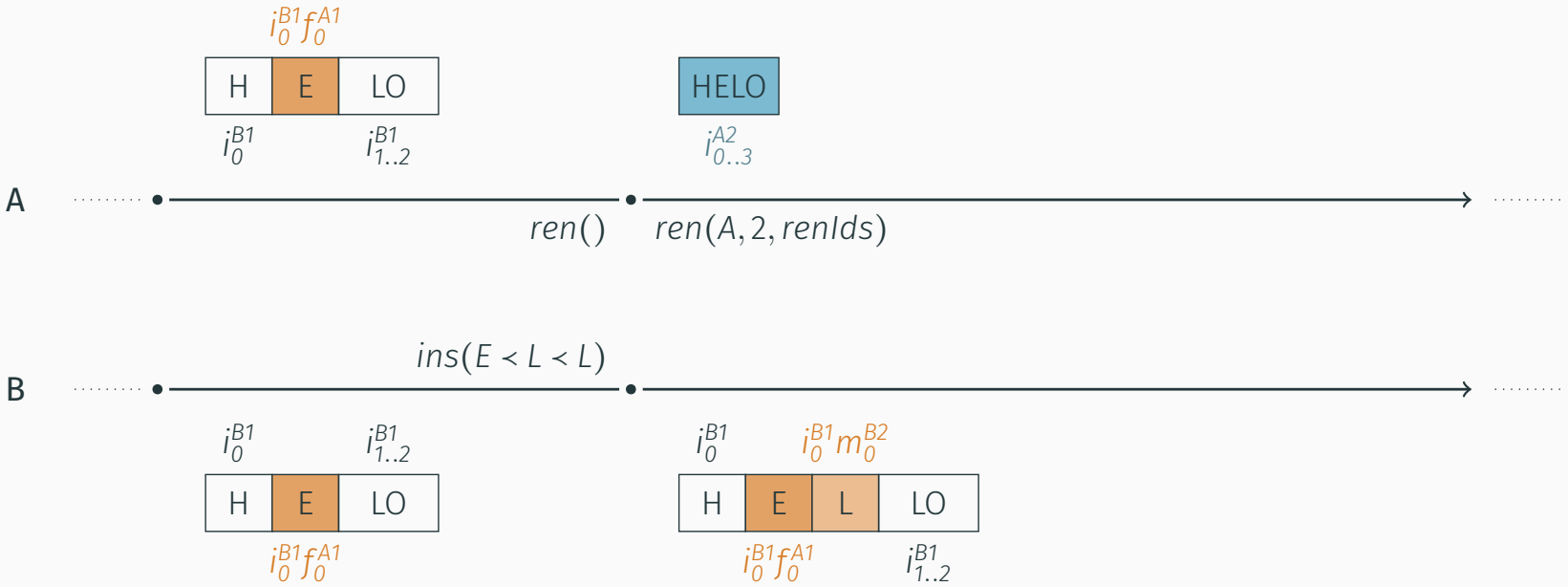
- Stocke identifiants ($[i_0^{B1}, i_0^{B1} f_0^{A1}, \ldots]$) de l'état d'origine : *renIds*

Interactions avec opérations *insert* et *remove* concurrentes



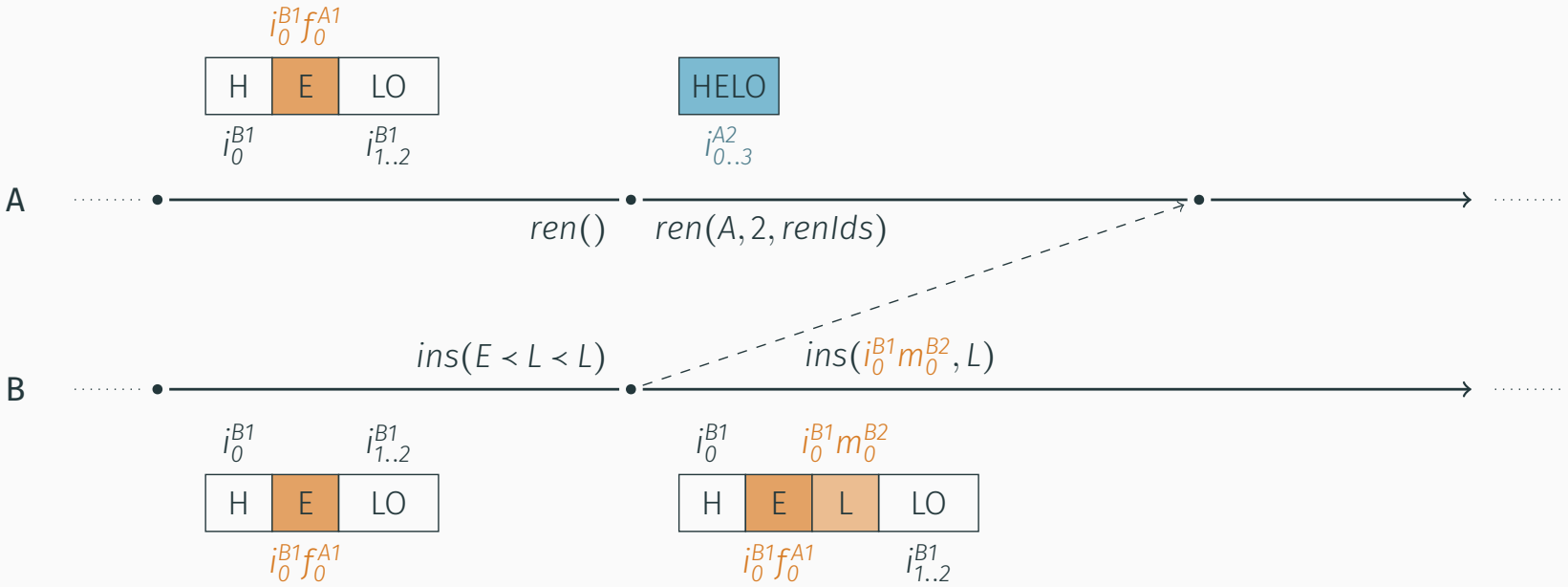
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



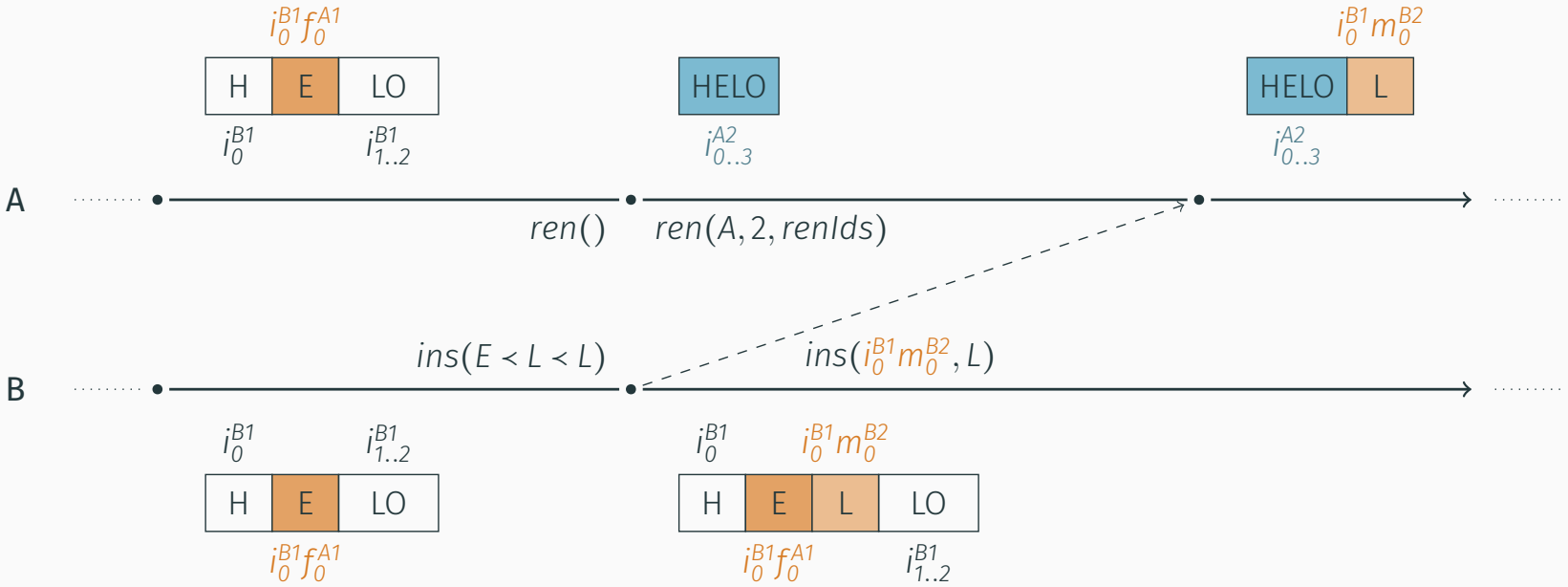
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



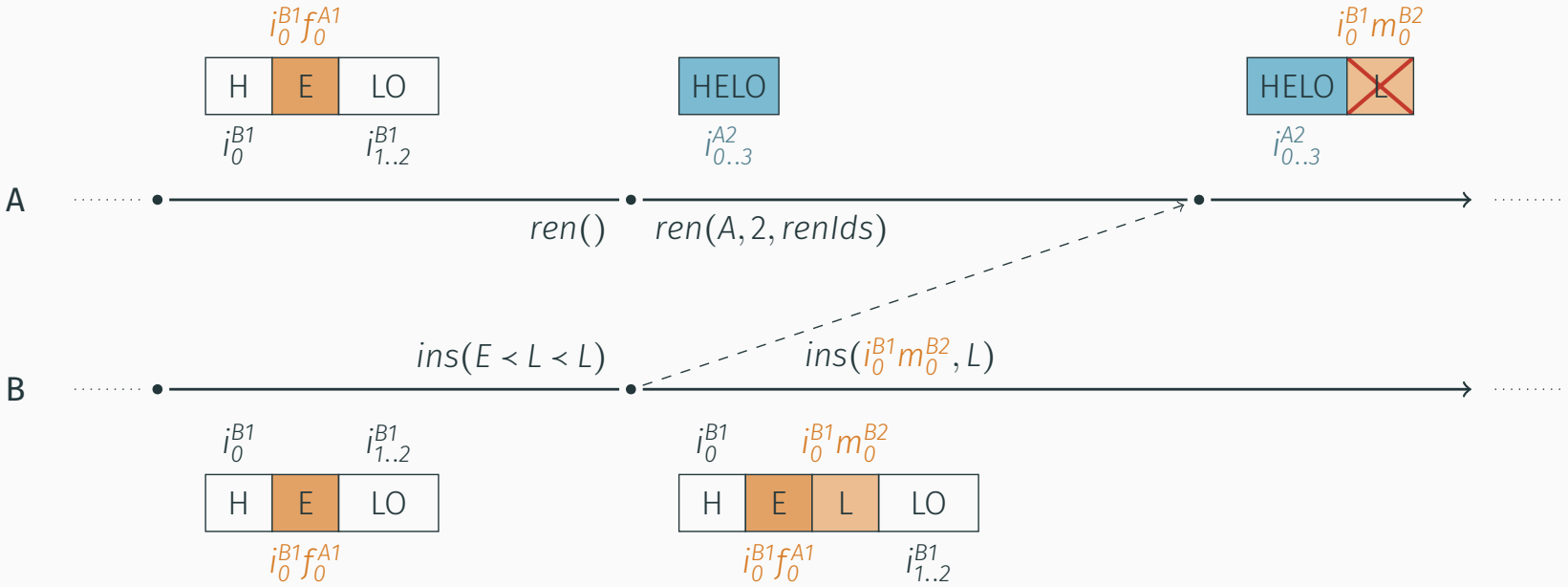
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



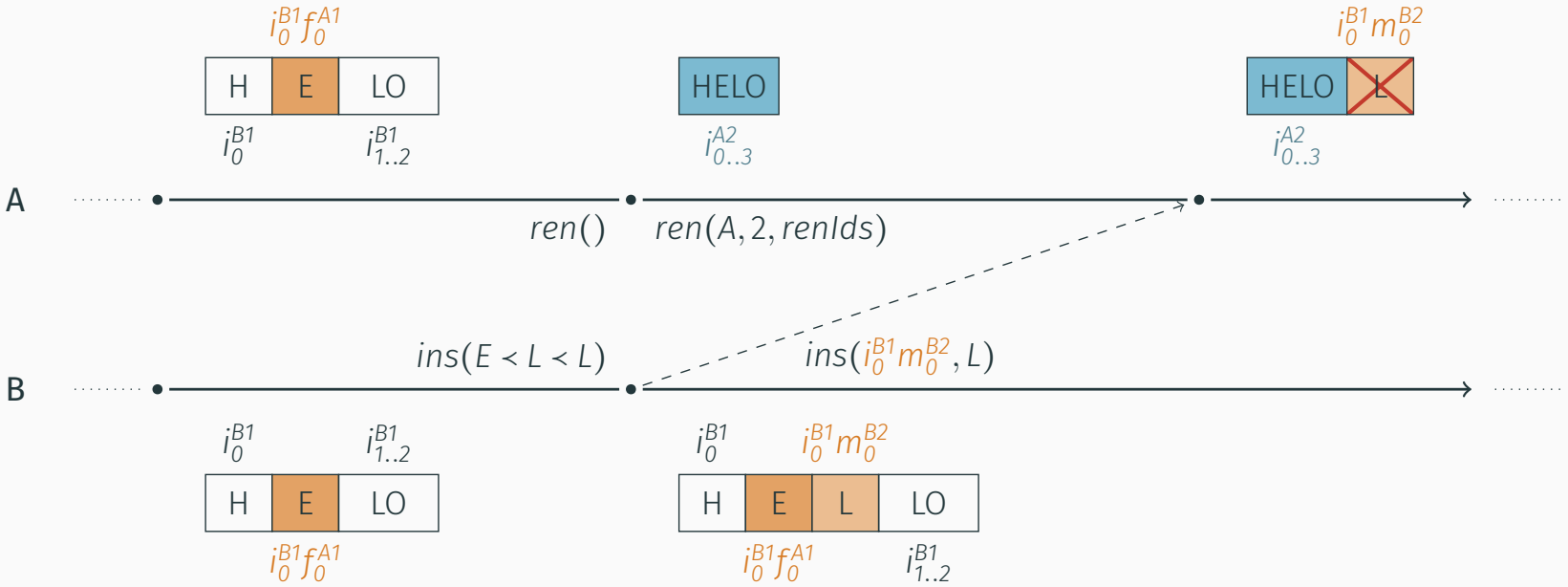
- Peuvent générer opérations concurrentes aux opérations *rename*

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

Mécanisme de résolution de conflits

Besoins

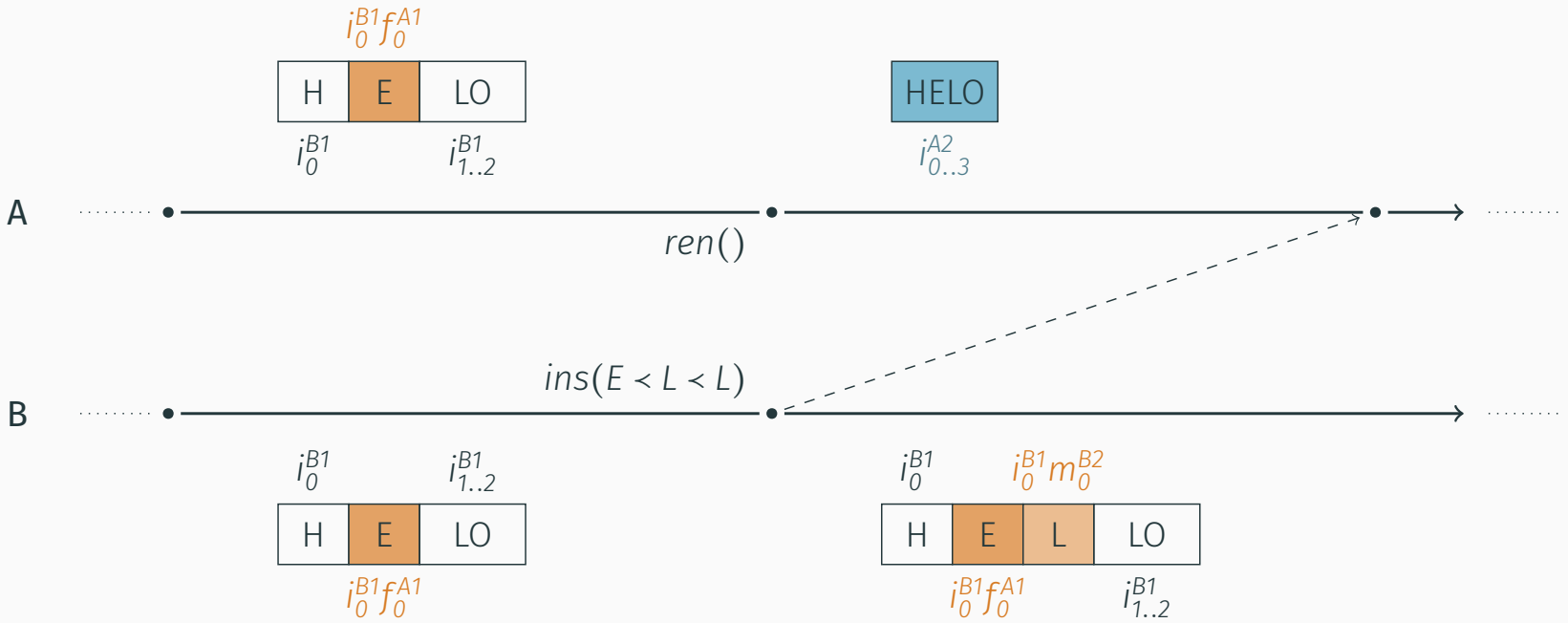
1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

Mécanisme de résolution de conflits

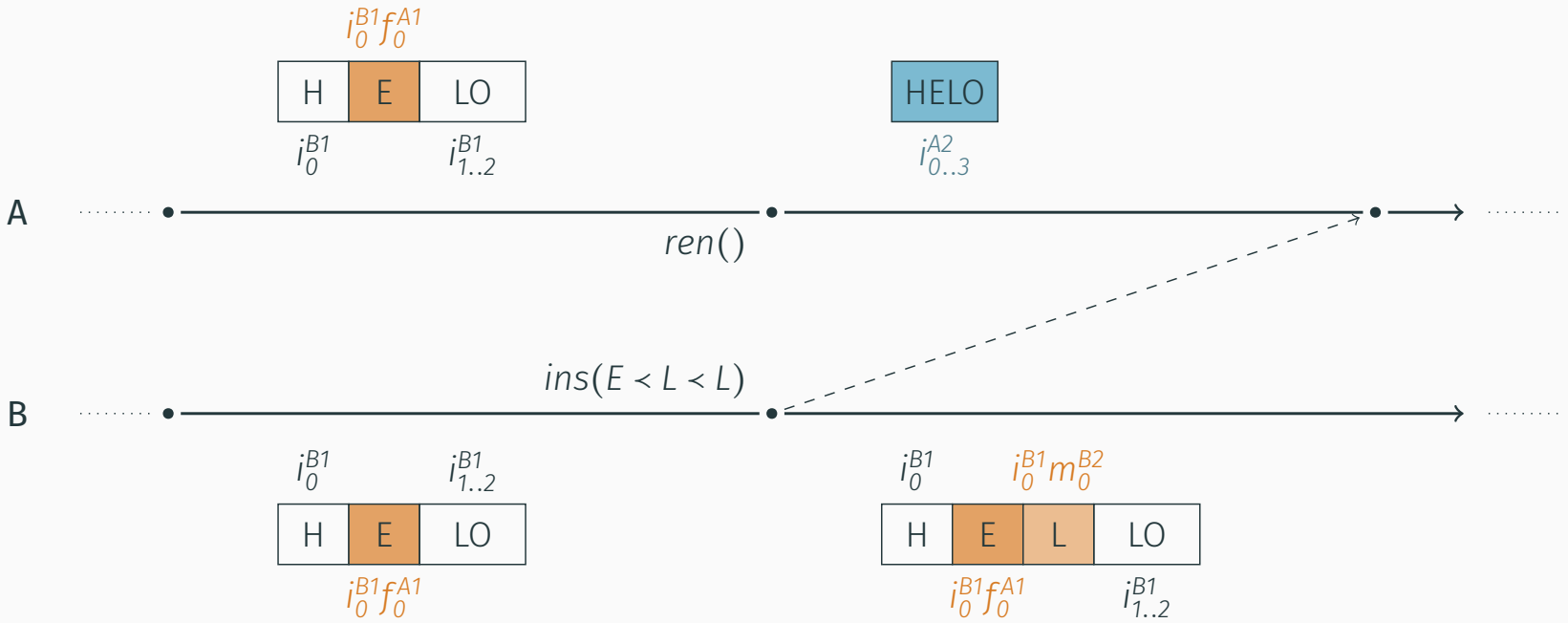
Besoins

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*

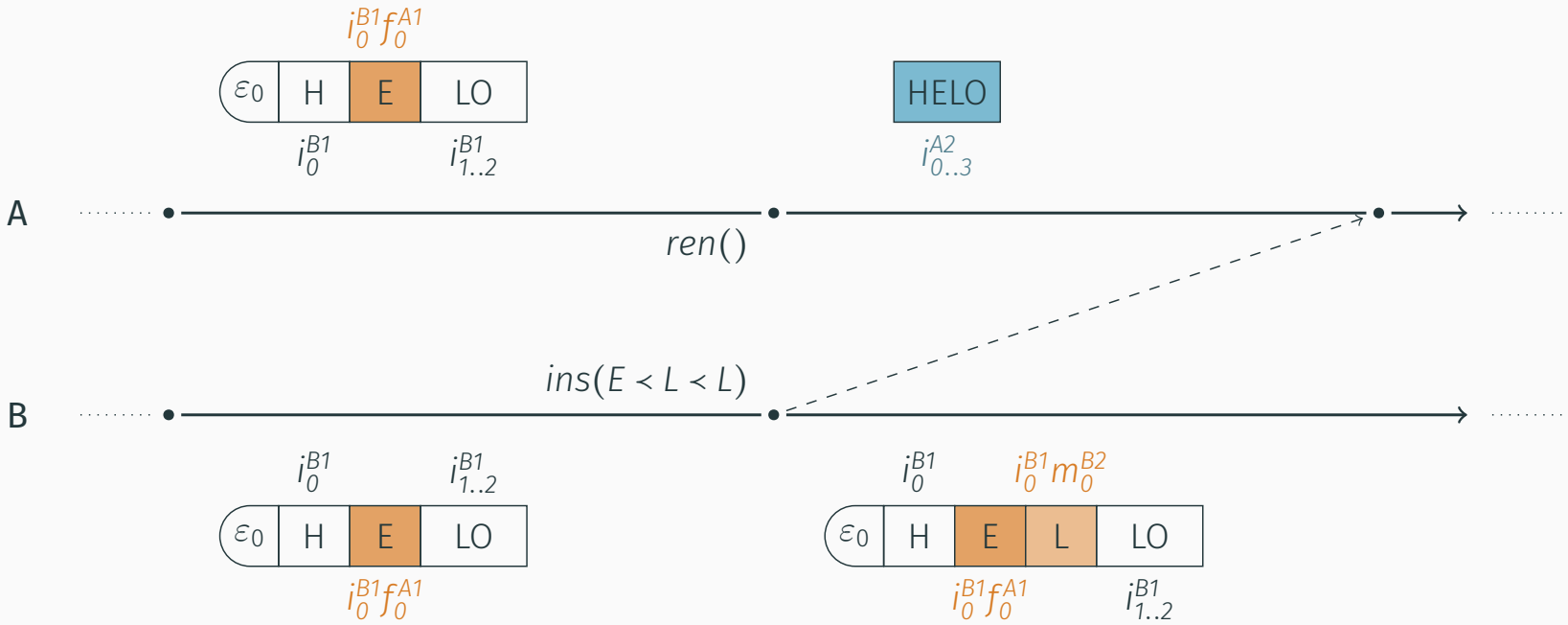


Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

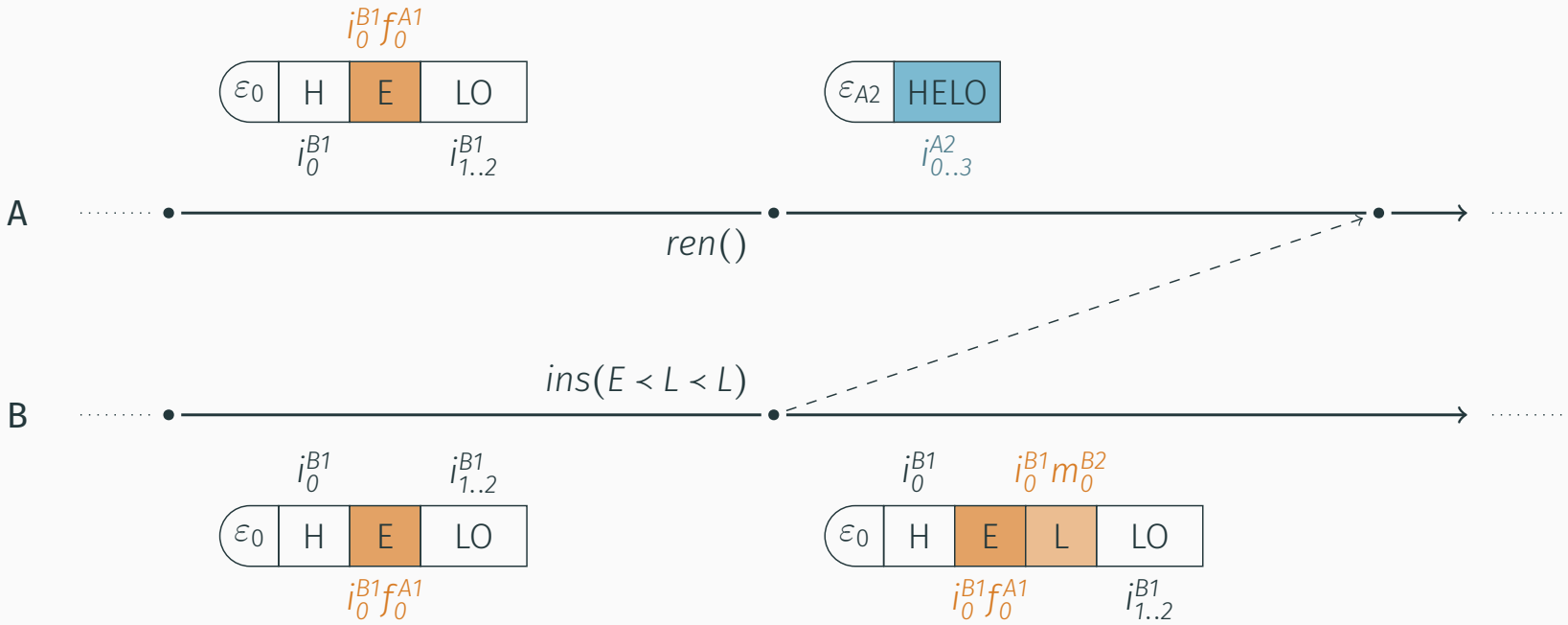
Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ε_0

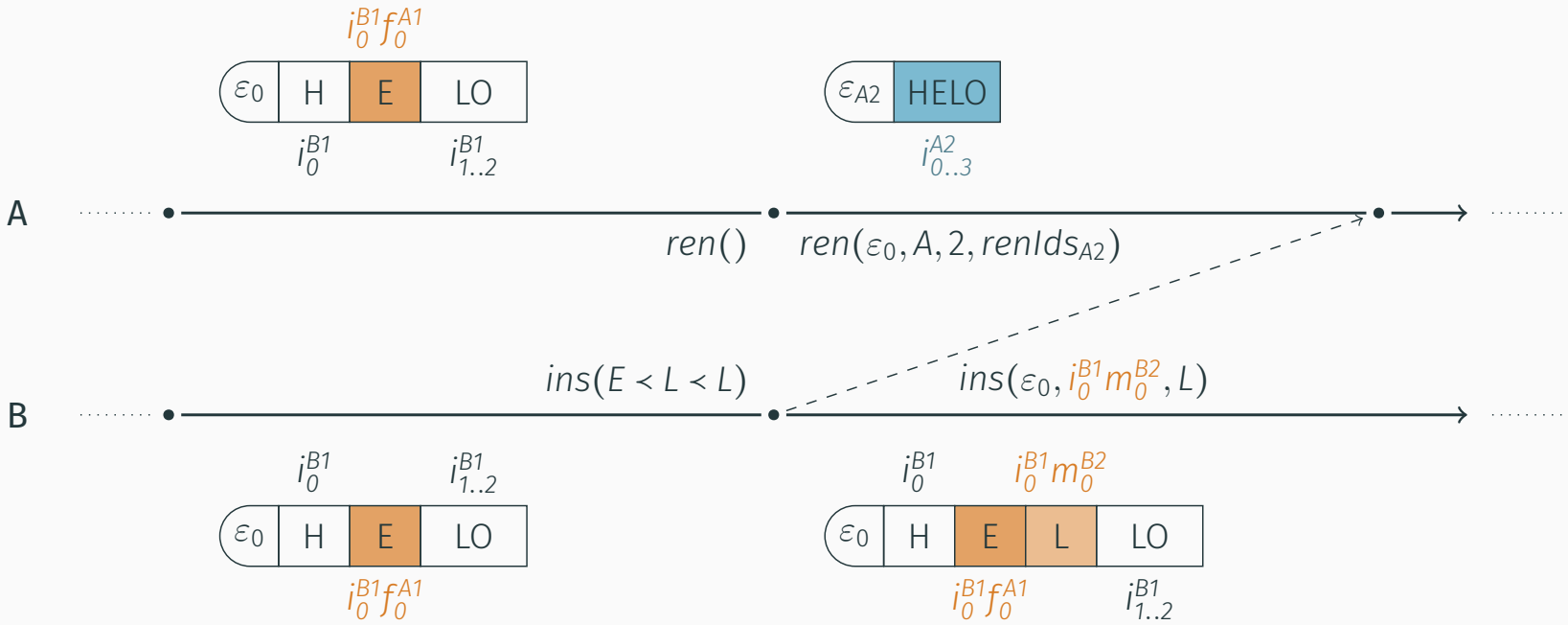
Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

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

Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

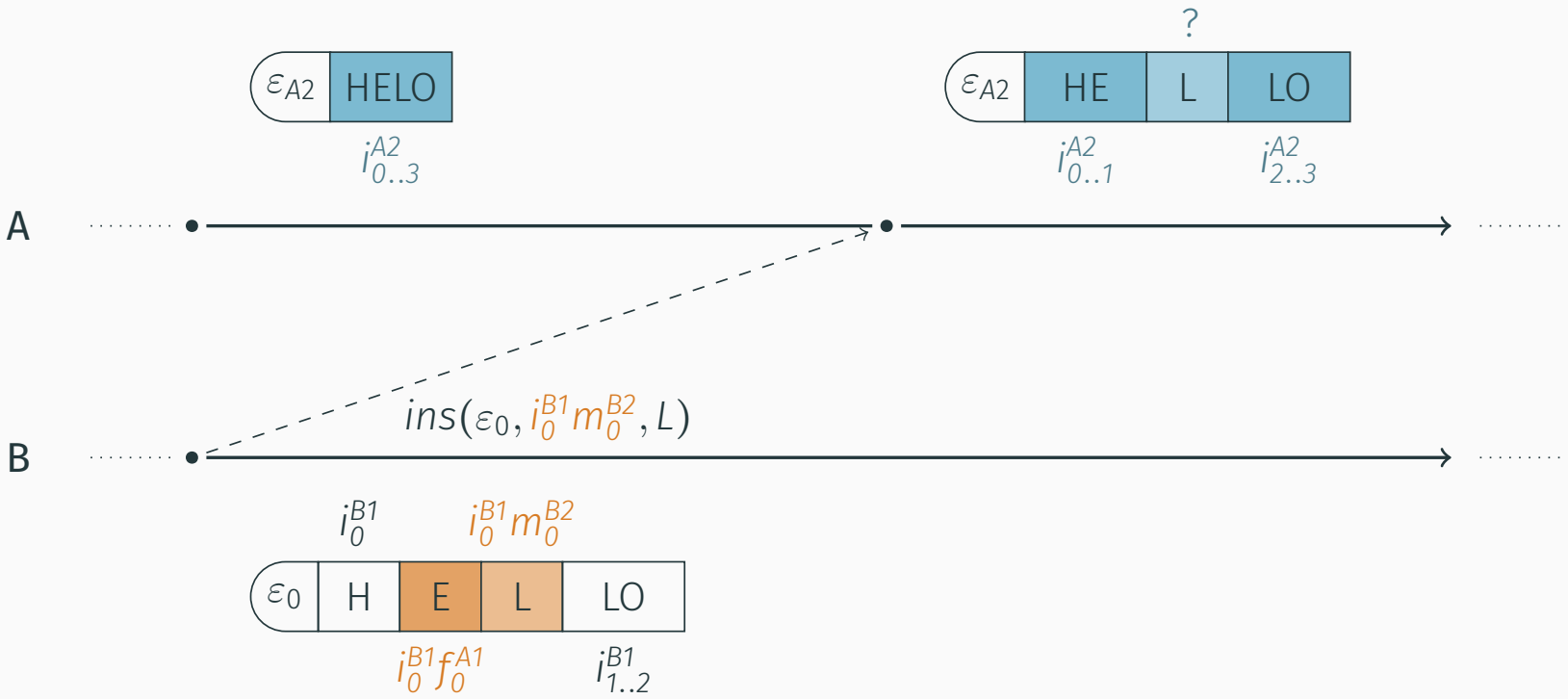
- Séquence commence à époque d'origine, notée ε_0
- *rename* font progresser à nouvelle époque, $\varepsilon_{nodeId\ nodeSeq}$
- Opérations labellisées avec époque de génération

Mécanisme de résolution de conflits

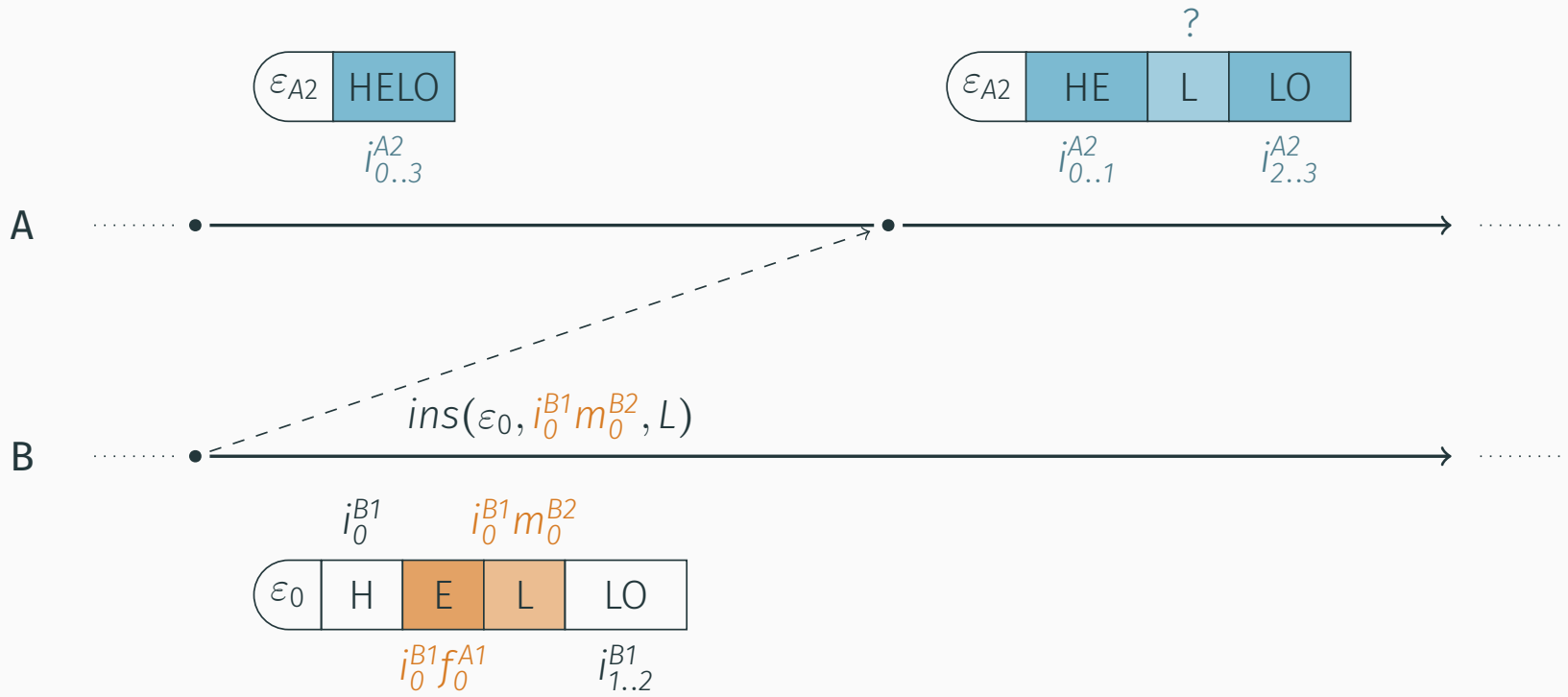
Besoins

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

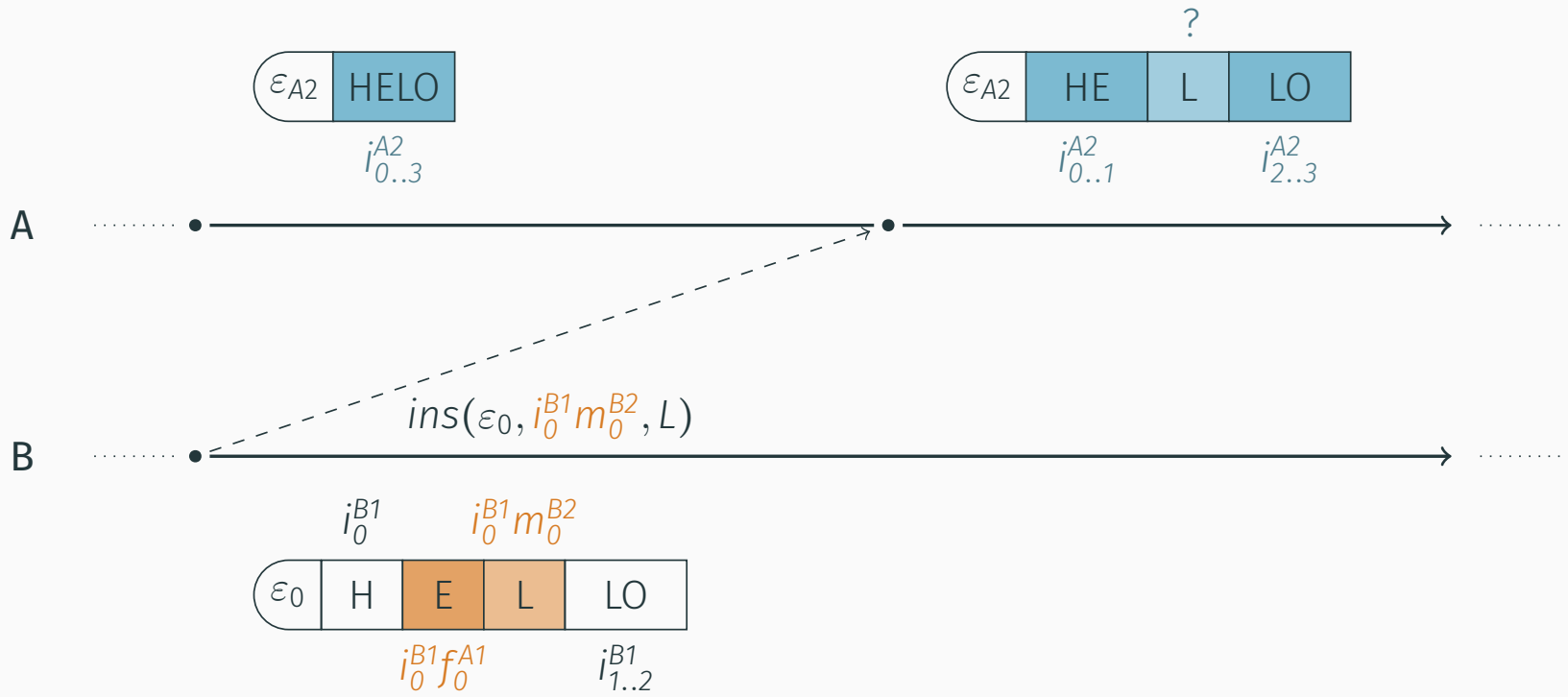


Intégration des opérations *insert* et *remove* concurrentes



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

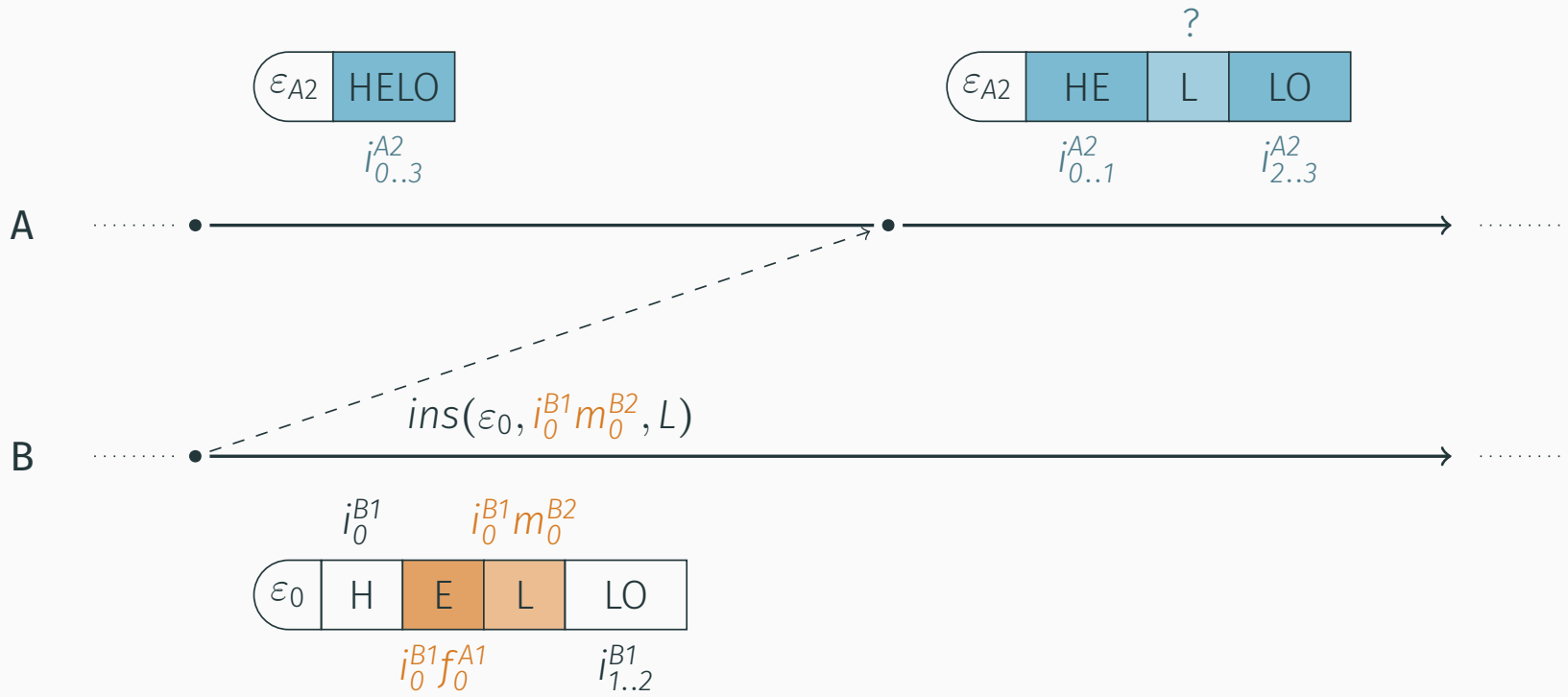
Intégration des opérations *insert* et *remove* concurrentes



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

- Prend la forme de l'algorithme `renameId`

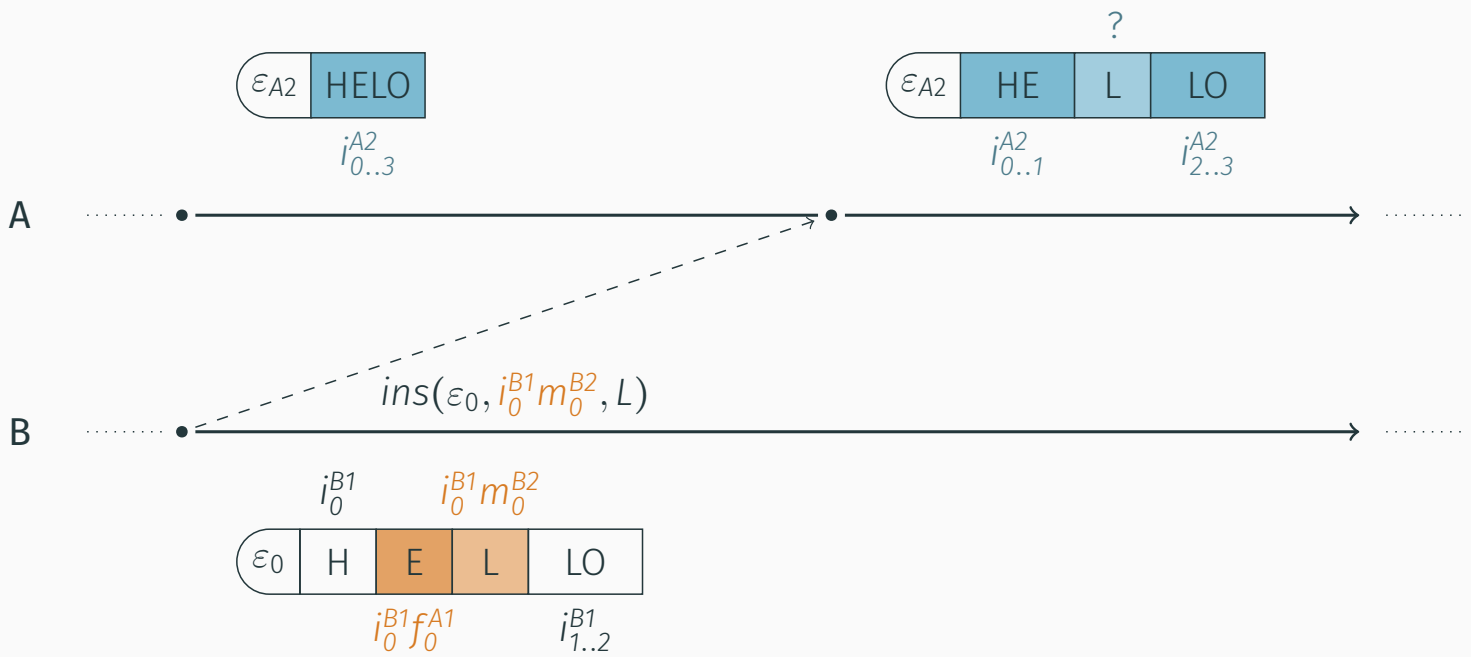
Intégration des opérations *insert* et *remove* concurrentes



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`

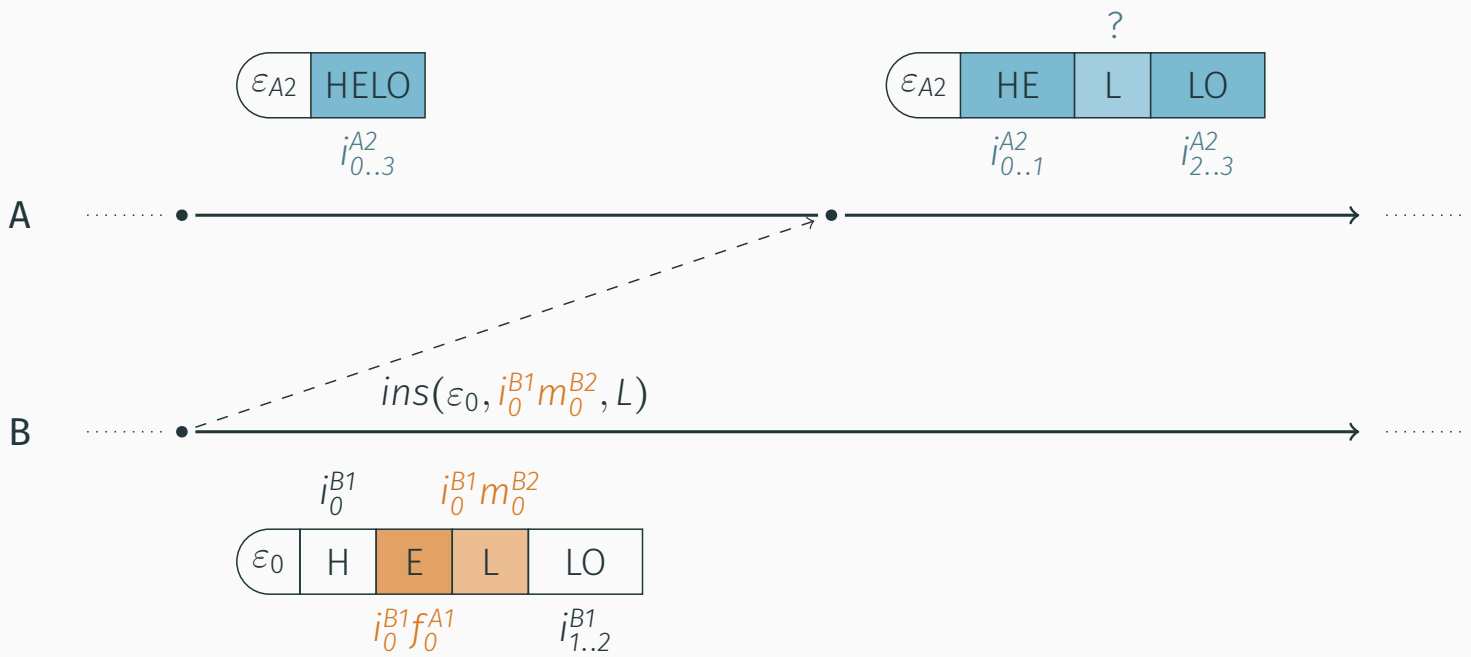


Rappel :

$$renIds_{A2} = [i^{B1}_0, i^{B1}_0 f^{A1}_0, i^{B1}_1, i^{B1}_2]$$

Exemple avec $i^{B1}_0 m^{B2}_0$

Exemple de `renameId`



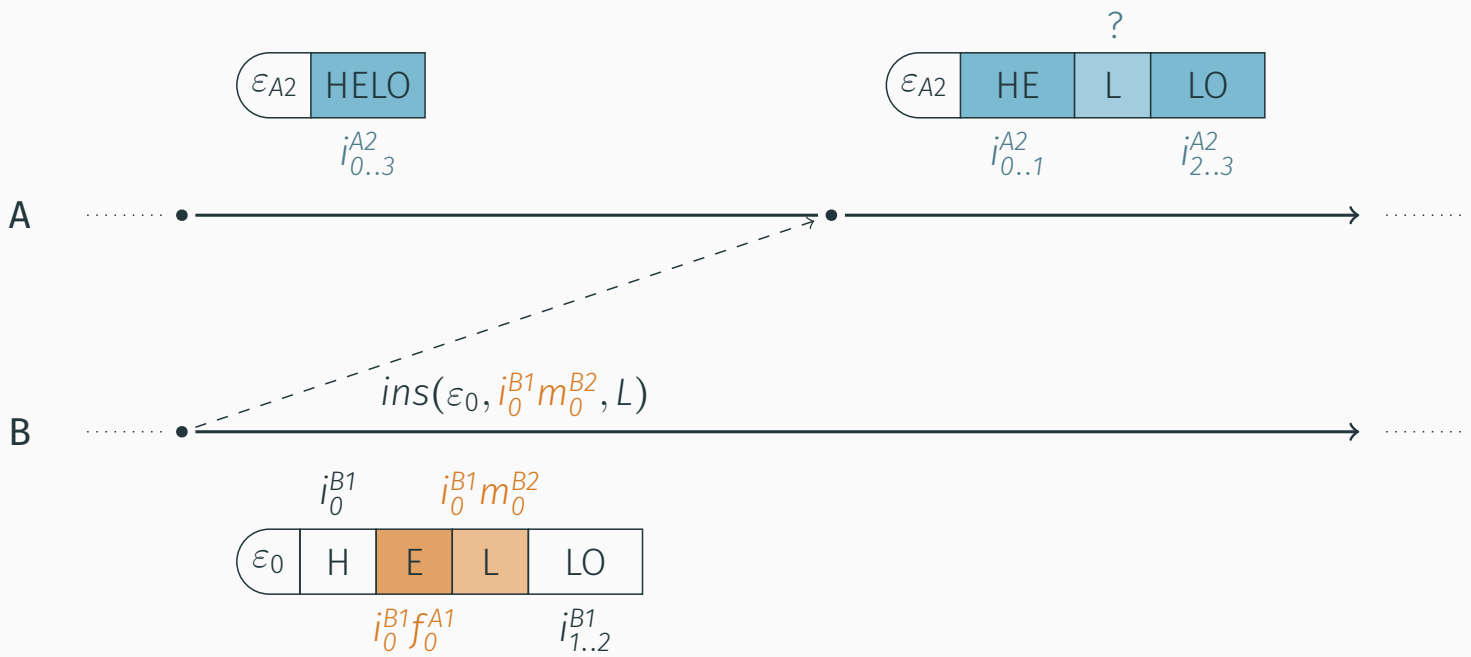
Rappel :

$$renIds_{A2} = [i^{B1}_0, i^{B1}_0 f^{A1}_0, i^{B1}_1, i^{B1}_2]$$

Exemple avec $i^{B1}_0 m^{B2}_0$

- Rechercher son prédecesseur dans $renIds_{A2}$: $i^{B1}_0 f^{A1}_0$

Exemple de `renameId`



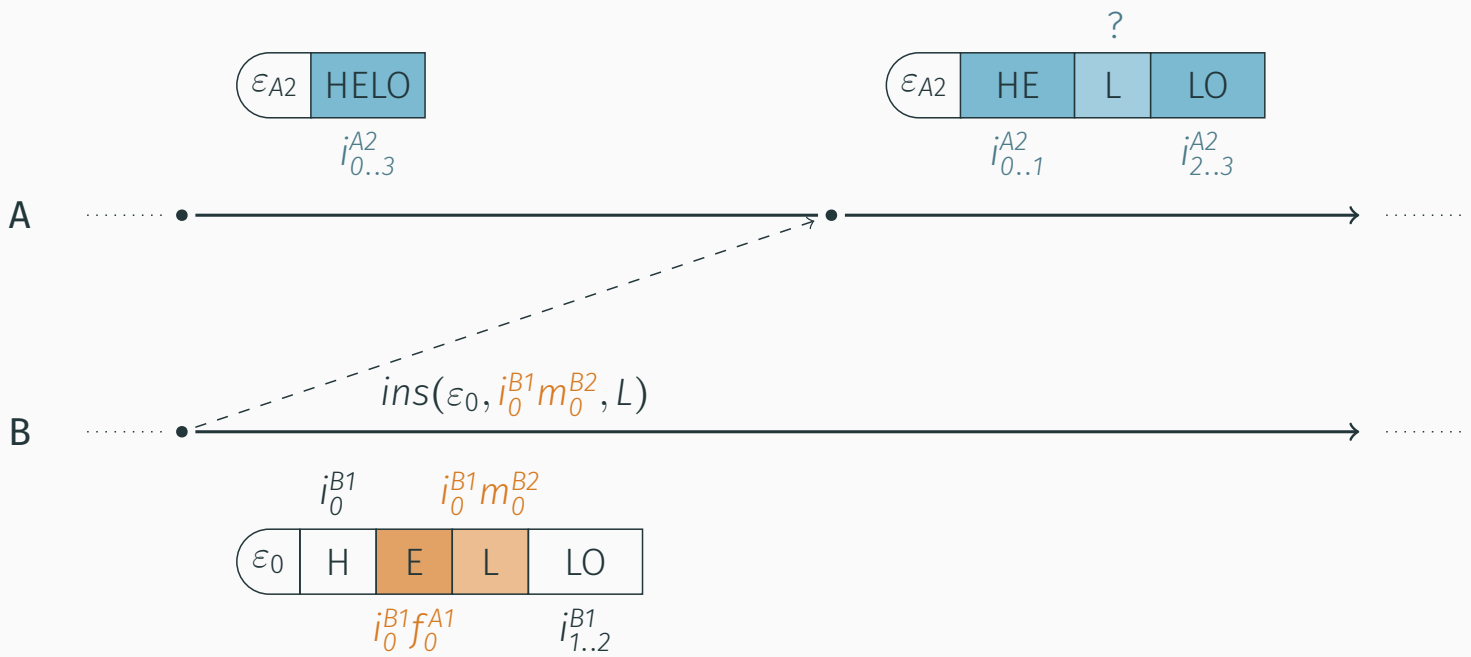
Rappel :

$$renIds_{A2} = [i^{B1}_0, i^{B1}_0 f^{A1}_0, i^{B1}_1, i^{B1}_2]$$

Exemple avec $i^{B1}_0 m^{B2}_0$

- Rechercher son prédecesseur dans $renIds_{A2}$: $i^{B1}_0 f^{A1}_0$
- Utiliser son index (1) pour calculer équivalent à époque ε_{A2} : i^{A2}_1

Exemple de `renameId`



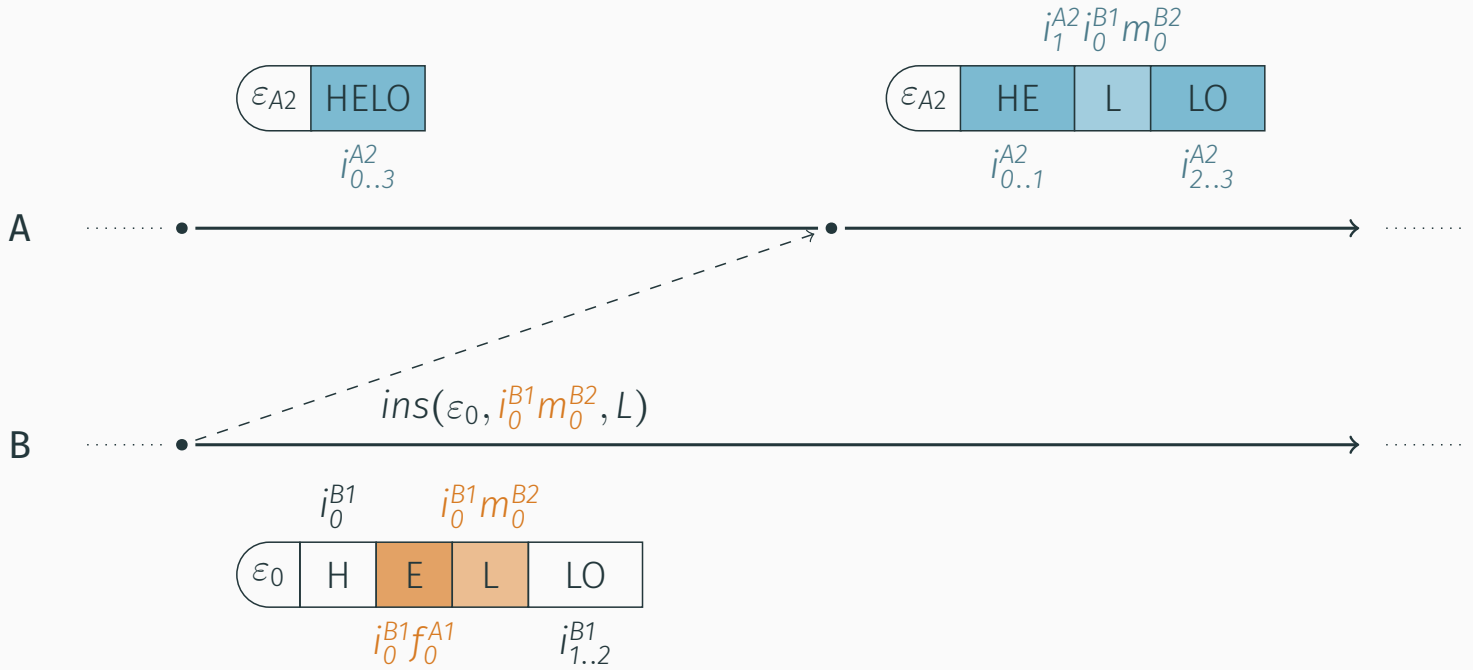
Rappel :

$$renIds_{A2} = [i^{B1}_0, i^{B1}_0 f^{A1}_0, i^{B1}_1, i^{B1}_2]$$

Exemple avec $i^{B1}_0 m^{B2}_0$

- Rechercher son prédecesseur dans $renIds_{A2}$: $i^{B1}_0 f^{A1}_0$
- Utiliser son index (1) pour calculer équivalent à époque ε_{A2} : i^{A2}_1
- Préfixer $i^{B1}_0 m^{B2}_0$ par ce dernier : $i^{A2}_1 i^{B1}_0 m^{B2}_0$

Exemple de `renameId`



Rappel :

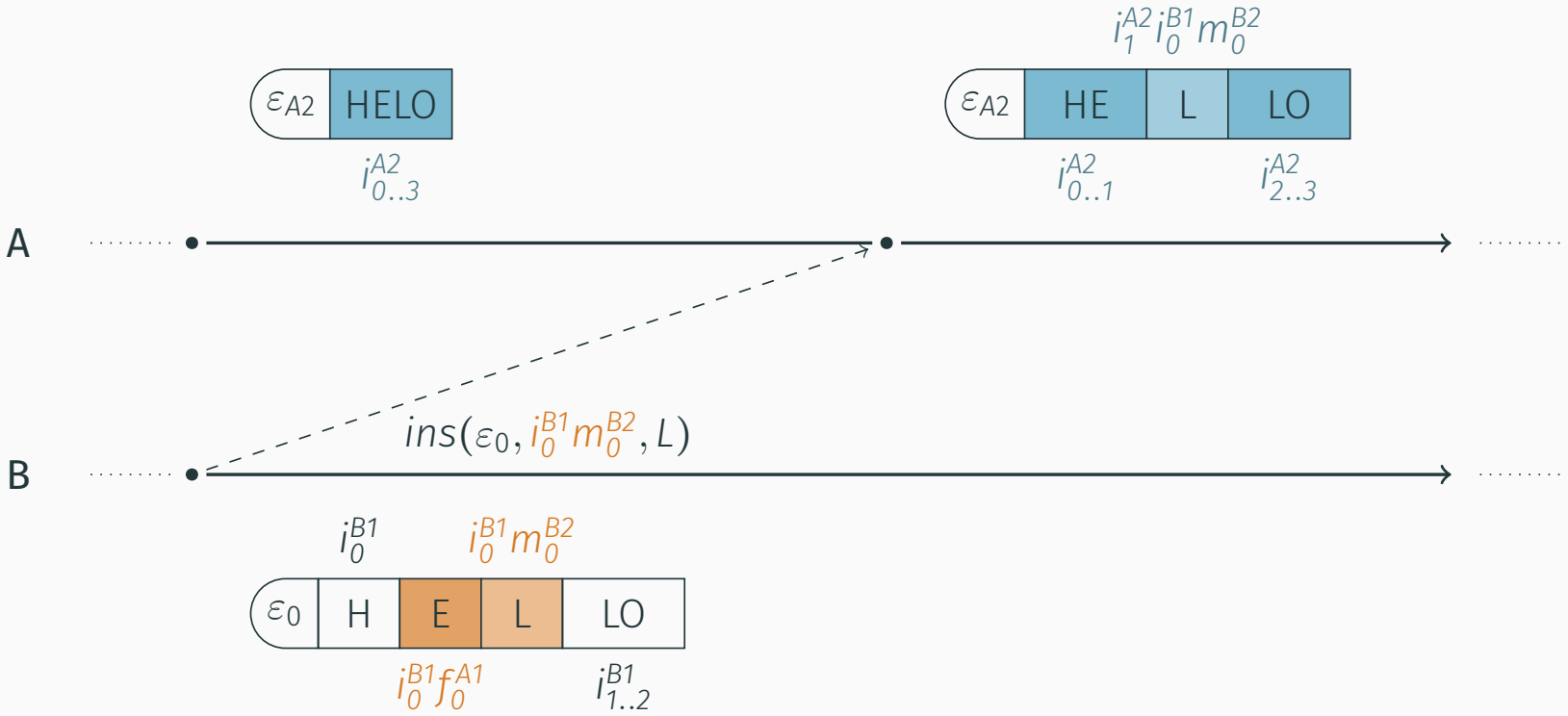
$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

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

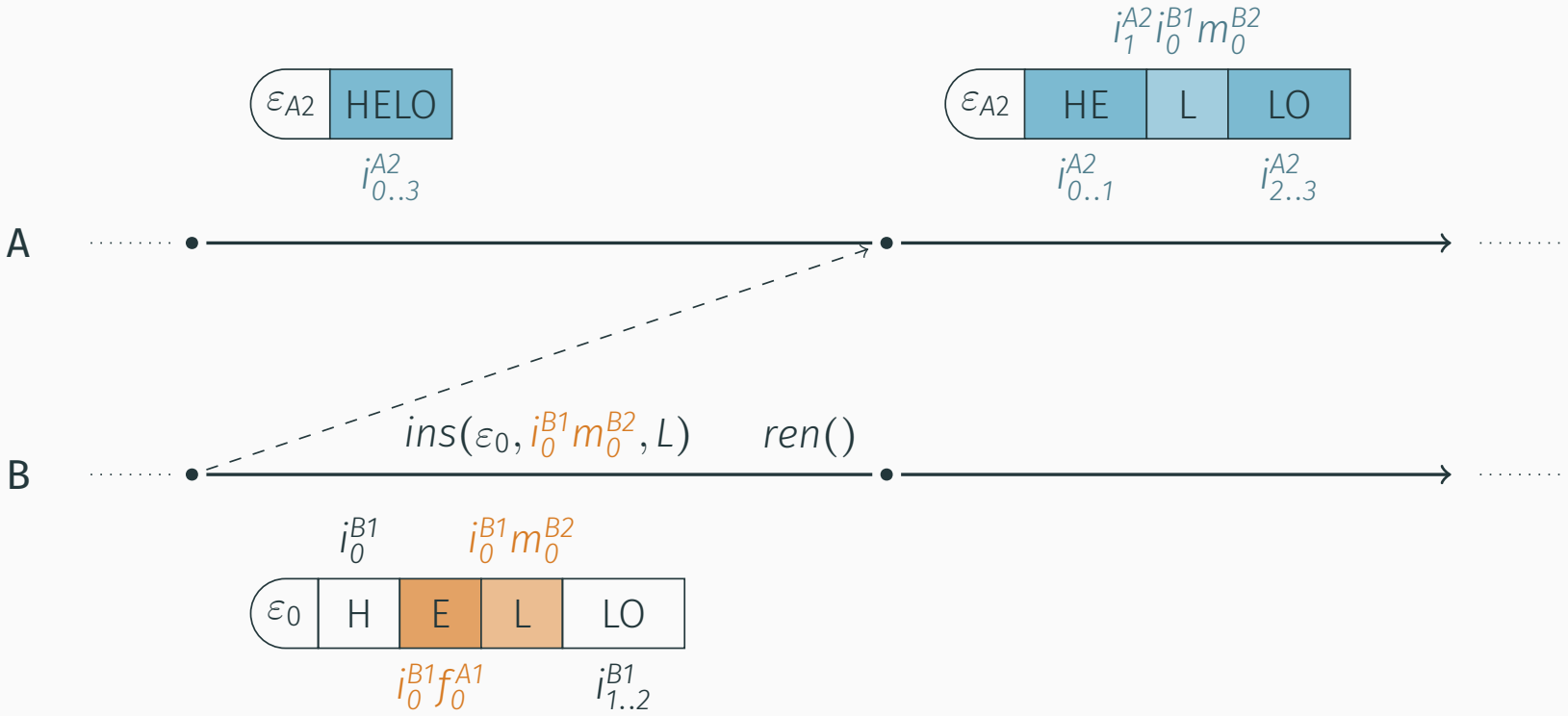
- Rechercher son prédecesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Utiliser son index (1) pour calculer équivalent à époque ε_{A2} : i_1^{A2}
- Préfixer $i_0^{B1} m_0^{B2}$ par ce dernier : $i_1^{A2} i_0^{B1} m_0^{B2}$

Et en cas d’opérations *rename* concurrentes ?

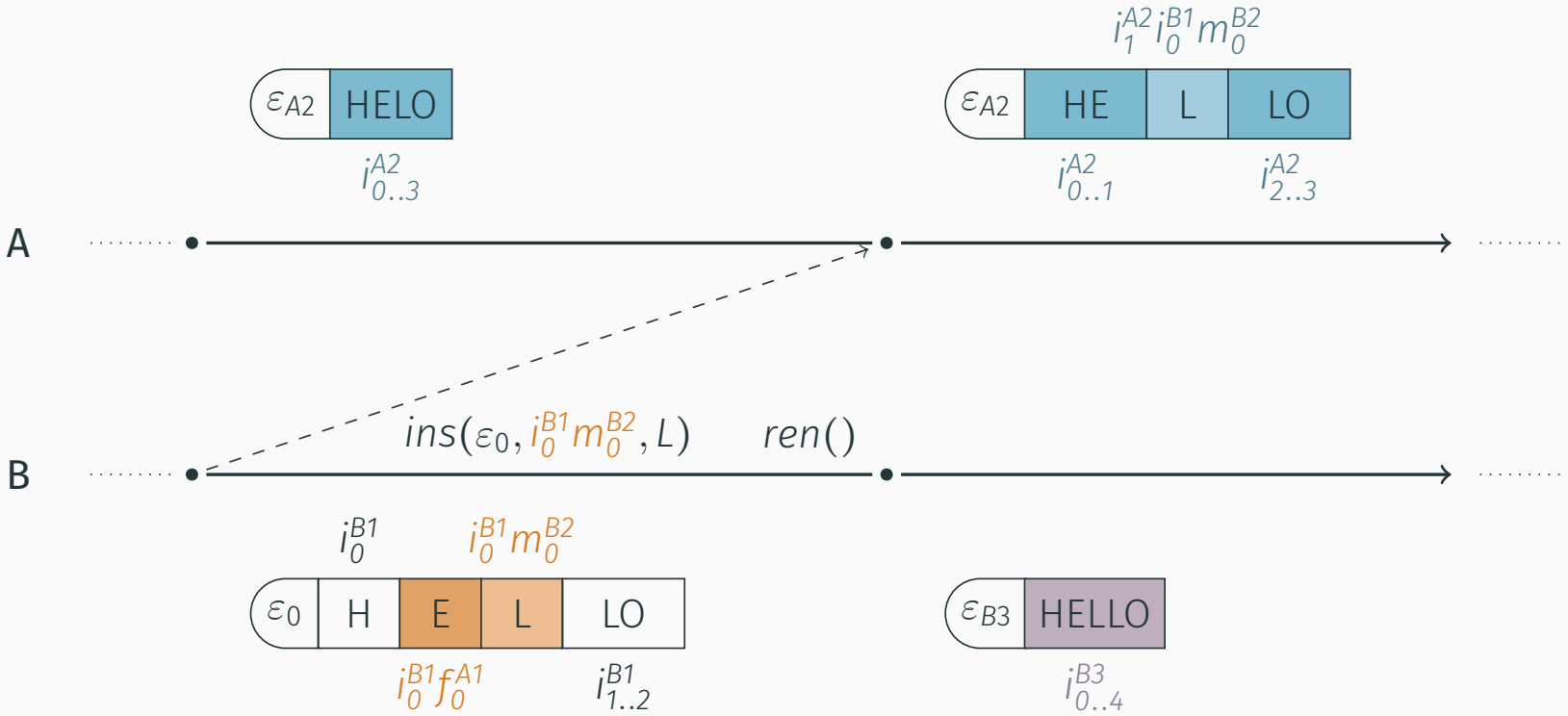
Opérations *rename* concurrentes



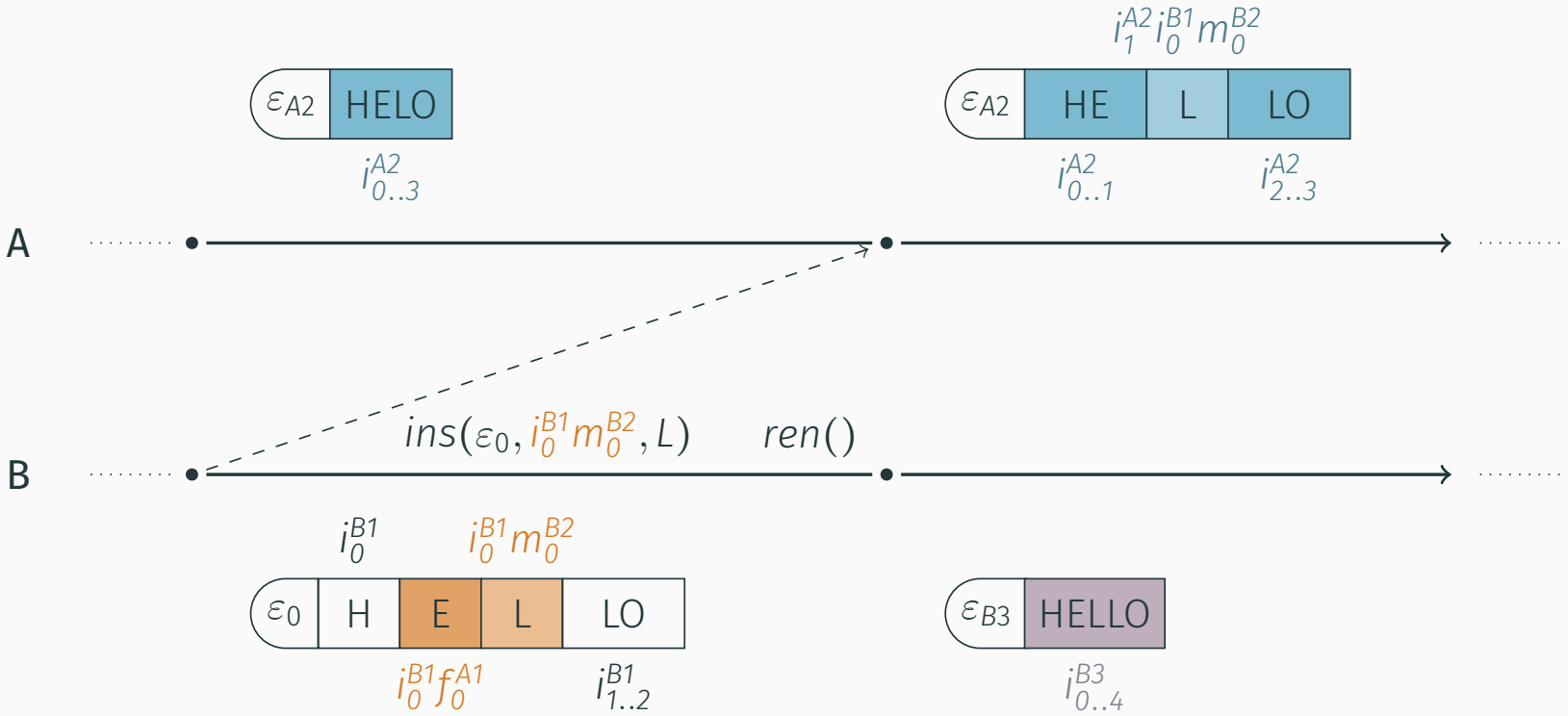
Opérations *rename* concurrentes



Opérations *rename* concurrentes

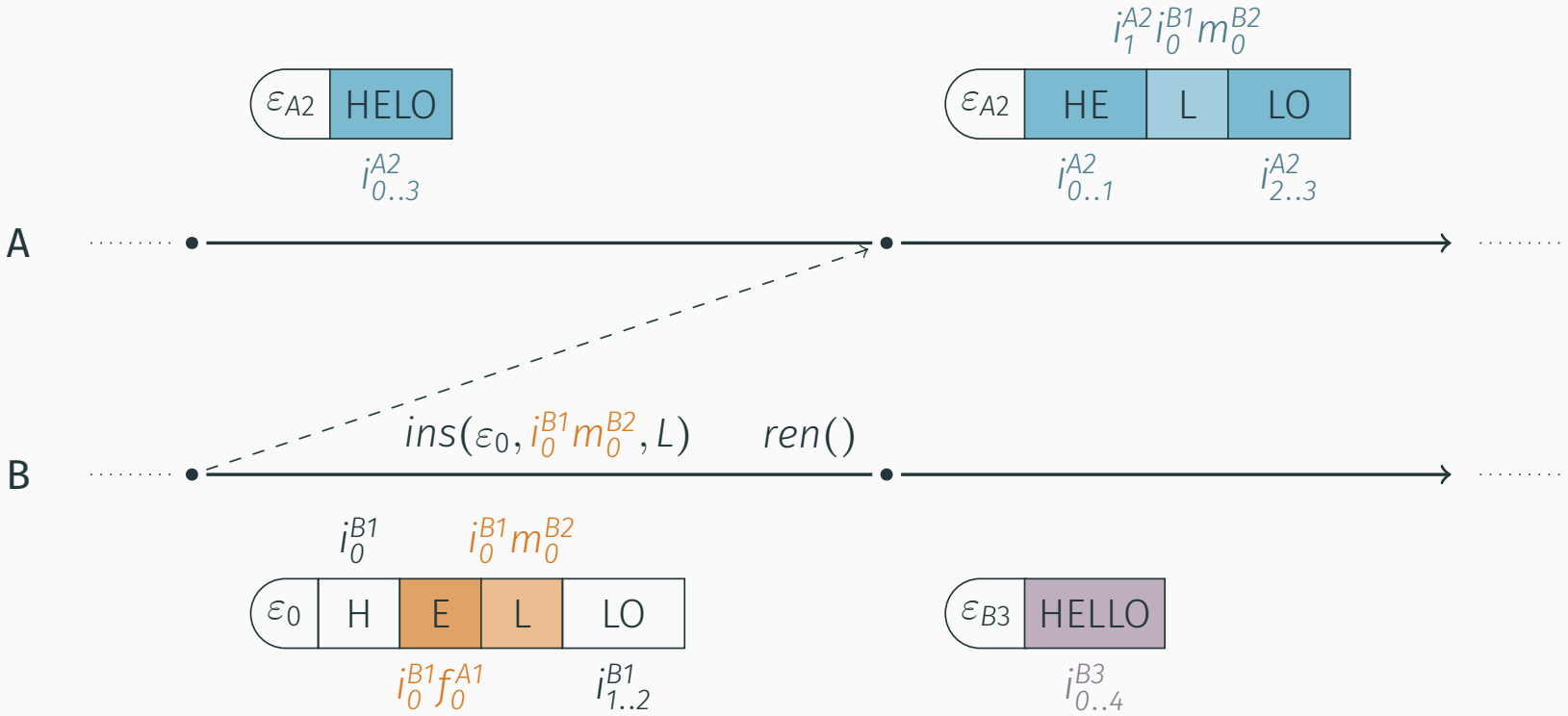


Opérations *rename* concurrentes



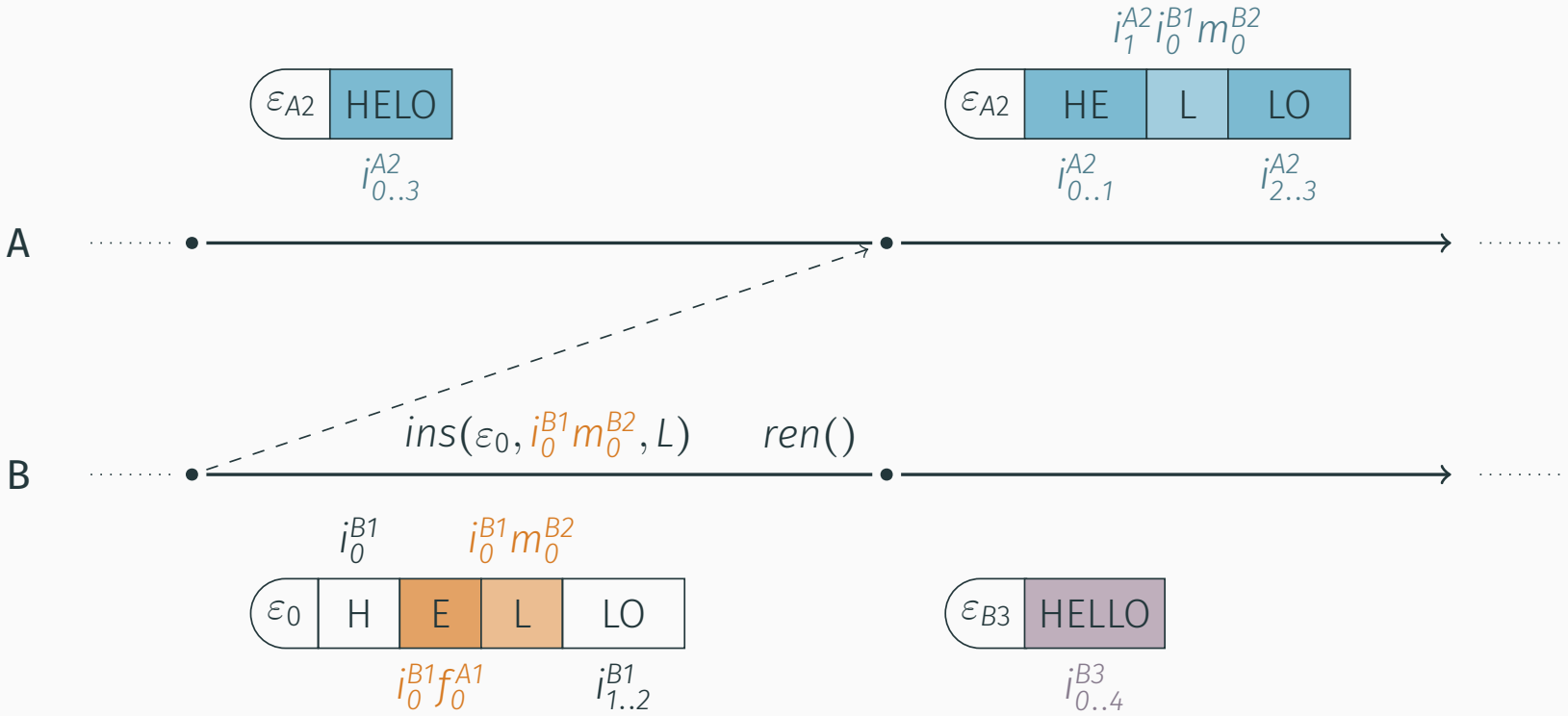
- Noeuds possèdent des contenus identiques...

Opérations *rename* concurrentes



- Noeuds possèdent des contenus identiques...
- ...mais des états différents (identifiants, époques)

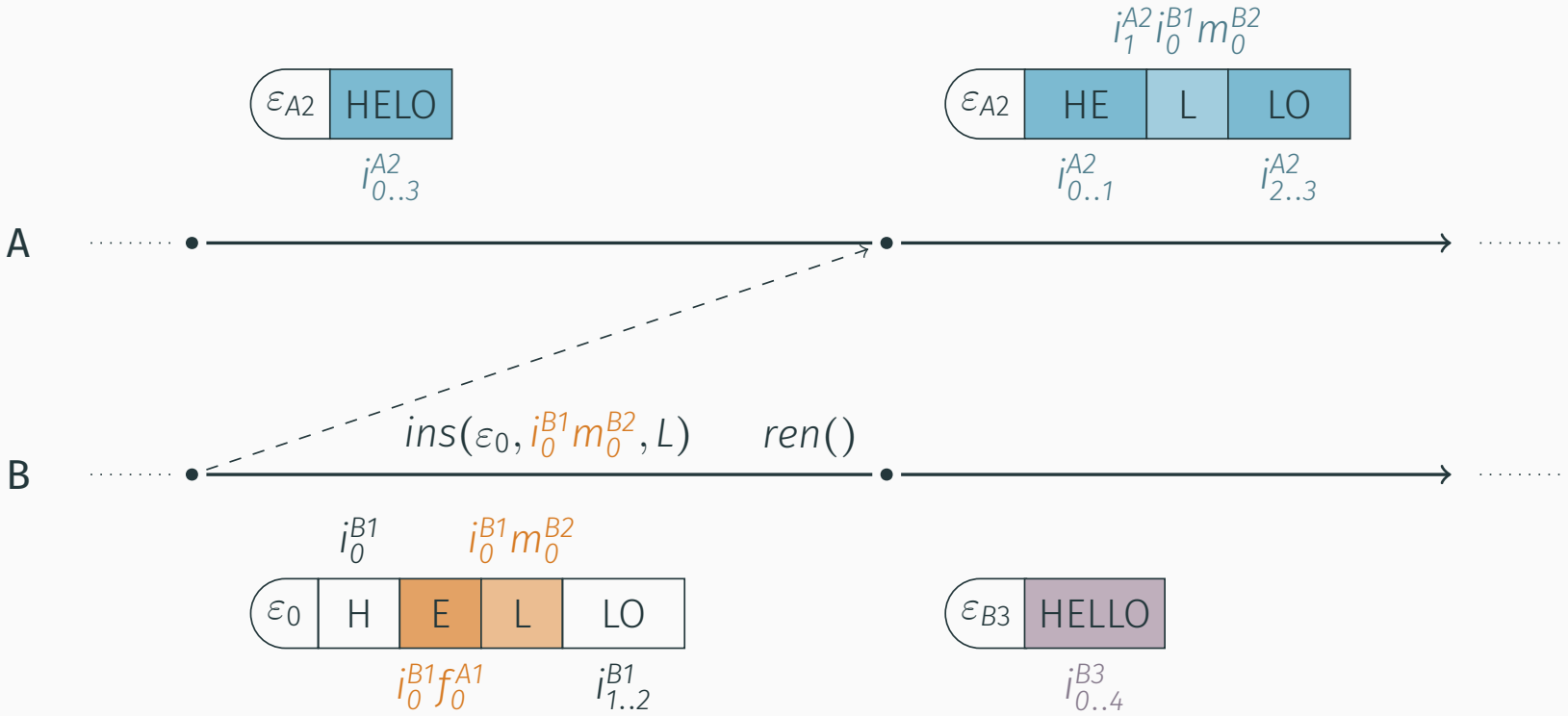
Opérations *rename* concurrentes



- Noeuds possèdent des contenus identiques…
- …mais des états différents (identifiants, époques)

Ne parlent plus la même langue, comment les faire converger ?

Opérations *rename* concurrentes



- Noeuds possèdent des contenus identiques…
- …mais des états différents (identifiants, époques)

Ne parlent plus la même langue, comment les faire converger ?

Besoin d'un mécanisme additionnel de résolution de conflits

Résolution de conflits entre opérations *rename* concurrentes

Observation

- Opérations *rename* sont des opérations systèmes...
- ...pas des opérations utilisateur-rices

Résolution de conflits entre opérations *rename* concurrentes

Observation

- Opérations *rename* sont des opérations systèmes...
- ...pas des opérations utilisateur-rices

Proposition

- Considérer une opération *rename* comme prioritaire...
- ...et ignorer les opérations *rename* en conflit avec elle

Algorithme d'intégration d'une opération *rename*

Intuition

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
   - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
   - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
   - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
   - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
   - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
   - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

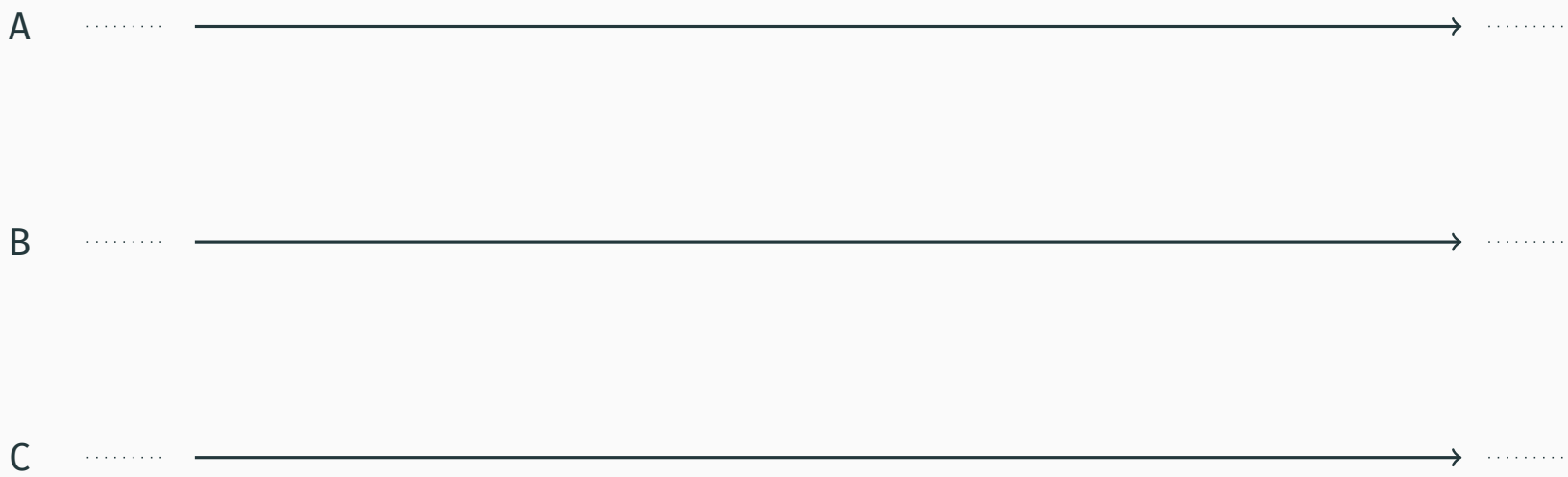
1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
   - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
   - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
   - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

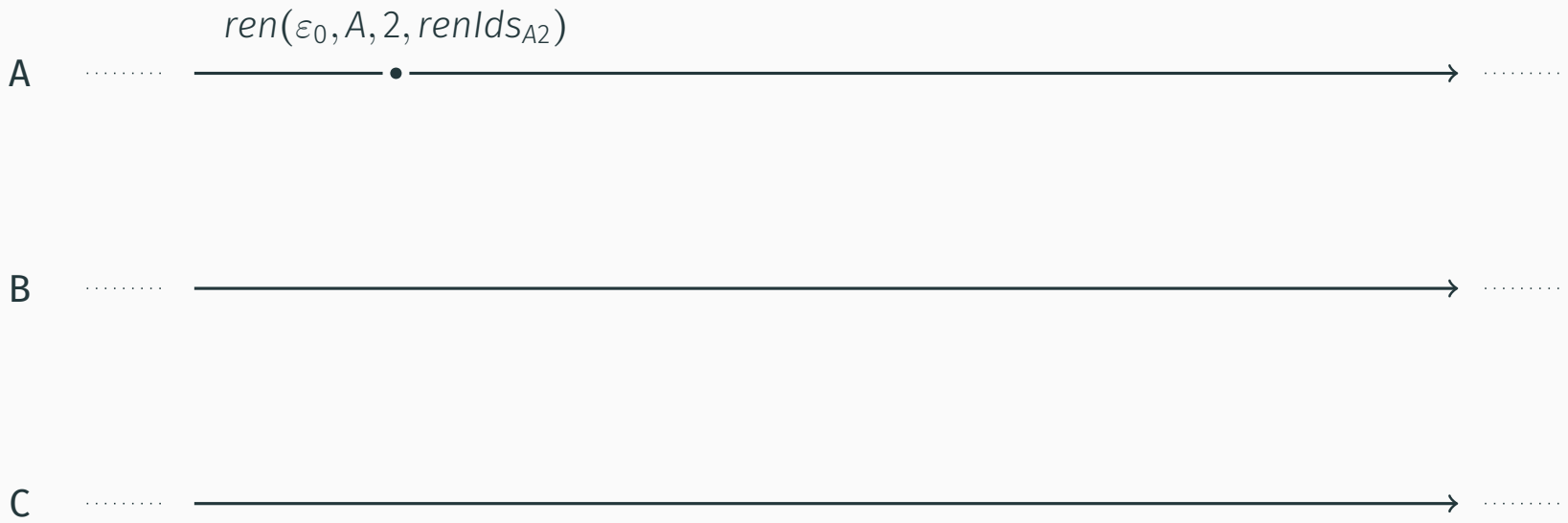
1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
   - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
   - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
   - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Choisir une époque comme époque cible

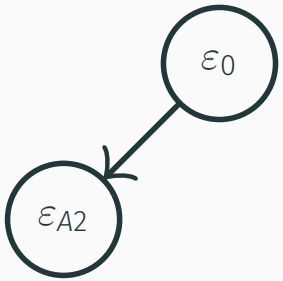


Arbre des époques

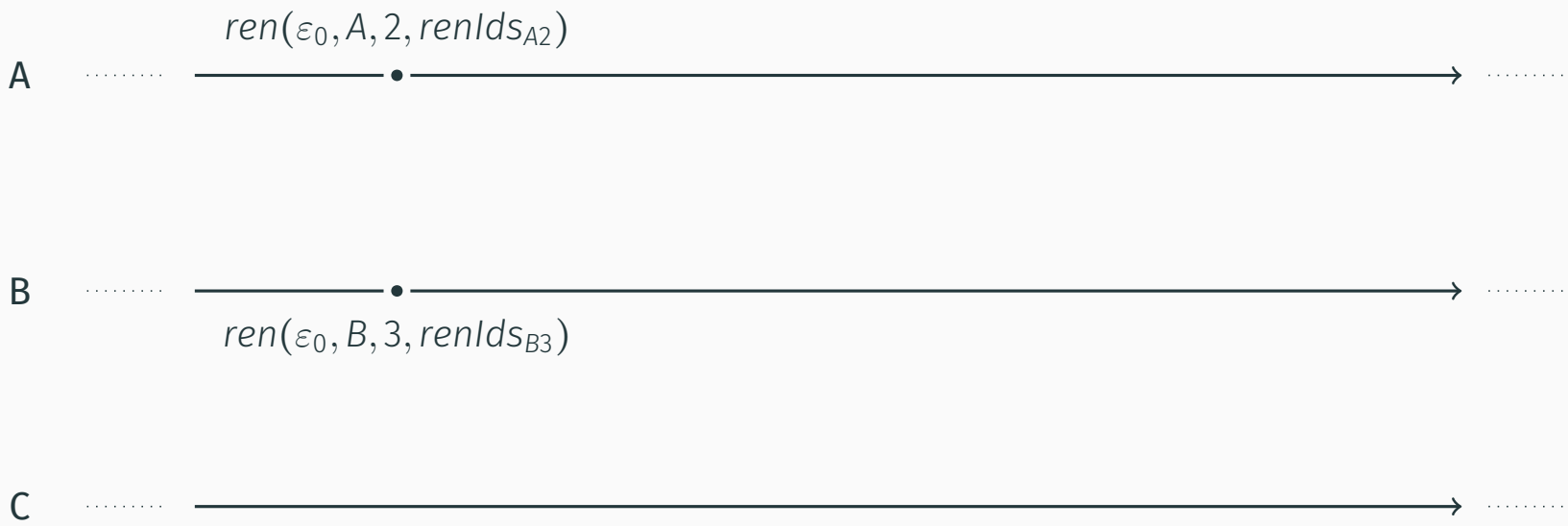
Choisir une époque comme époque cible



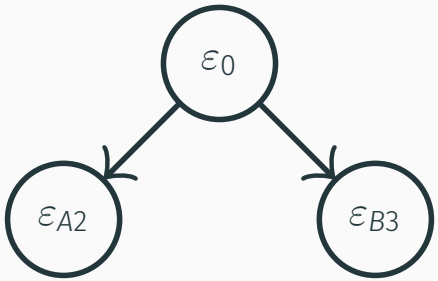
Arbre des époques



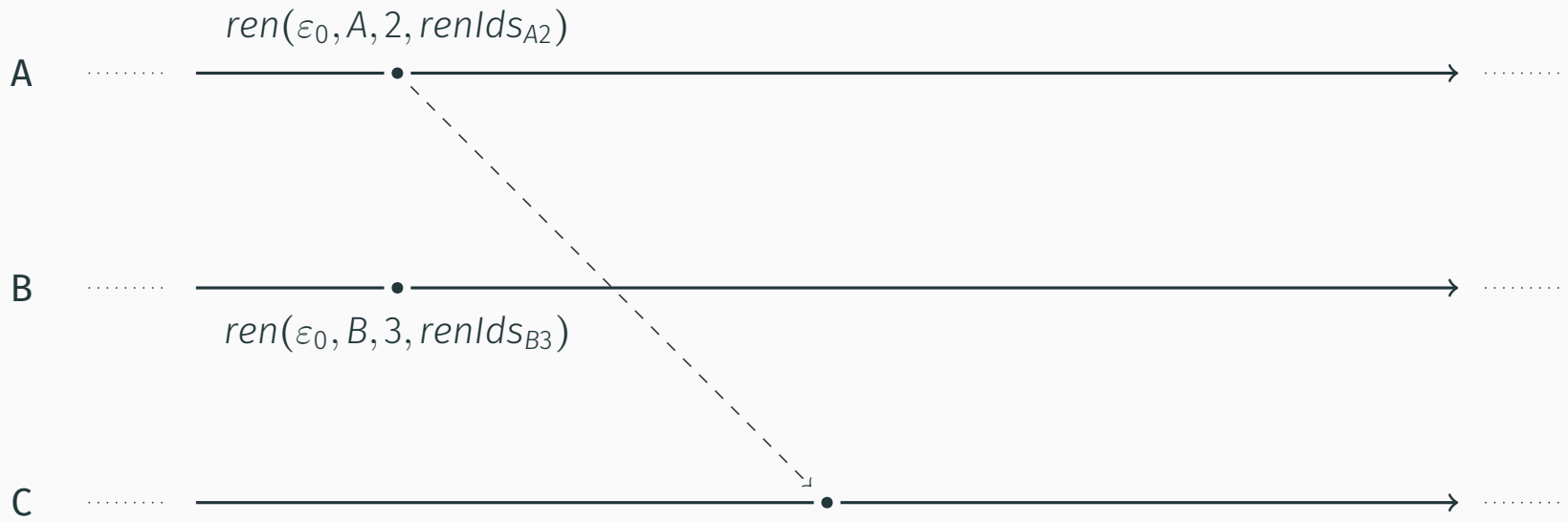
Choisir une époque comme époque cible



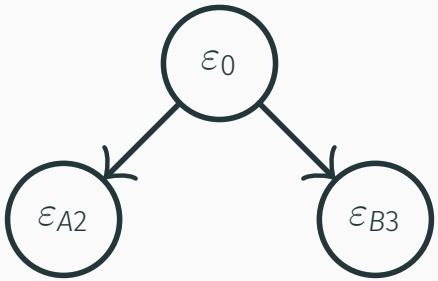
Arbre des époques



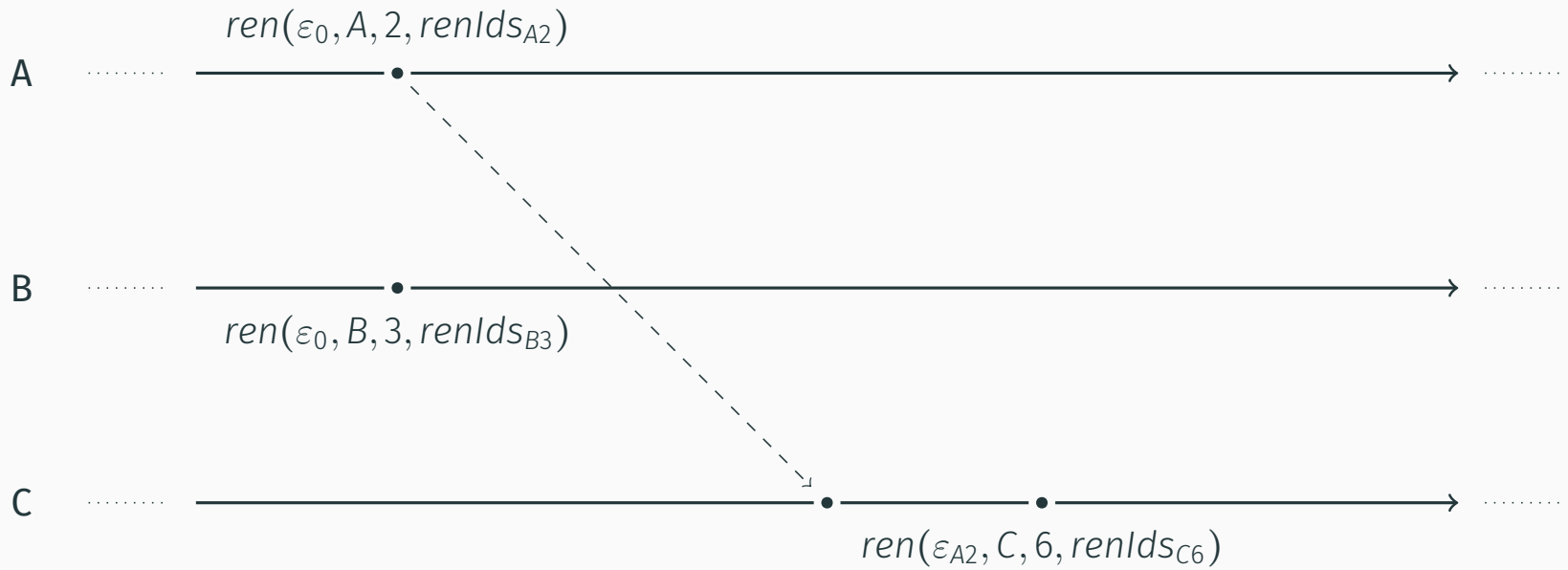
Choisir une époque comme époque cible



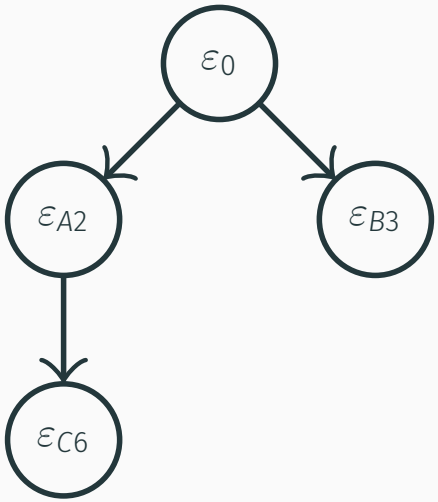
Arbre des époques



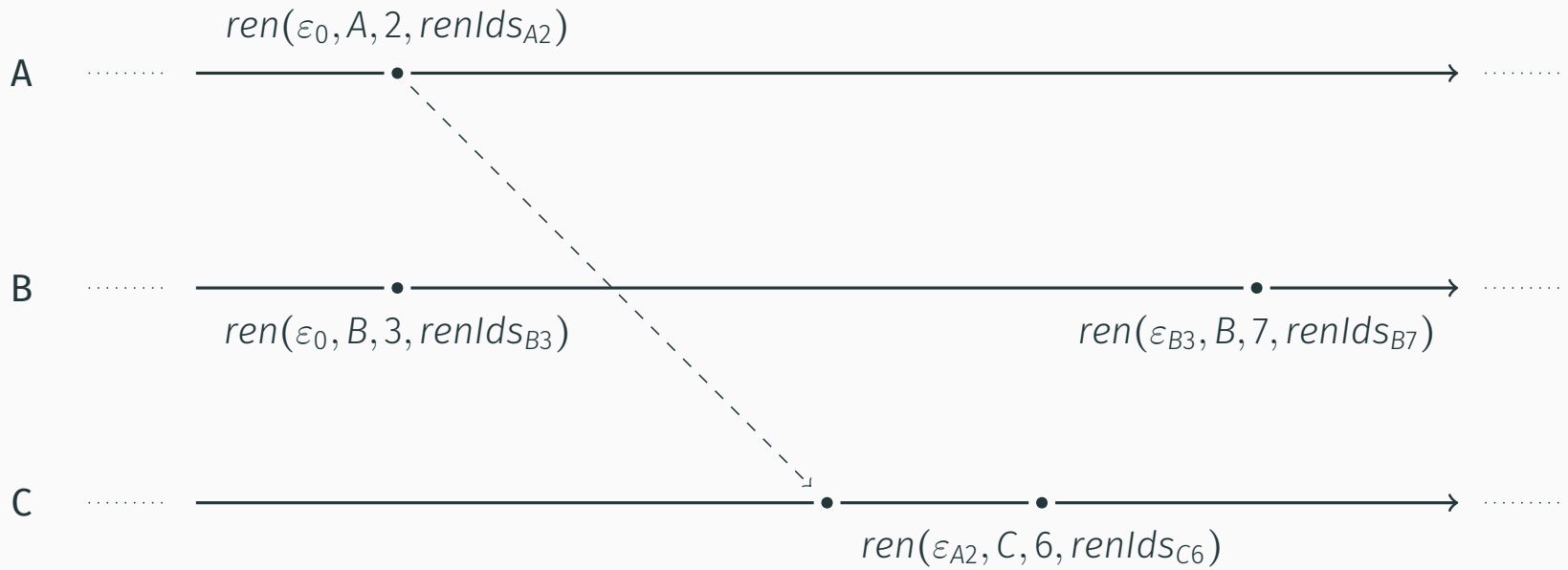
Choisir une époque comme époque cible



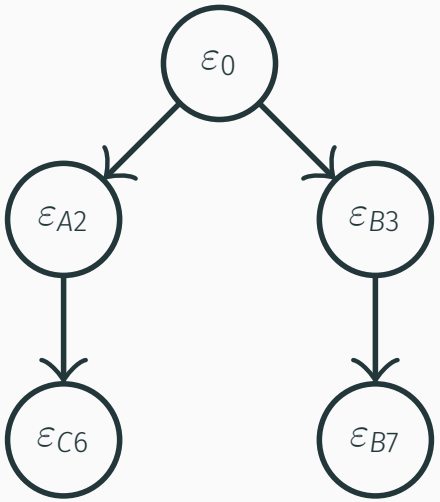
Arbre des époques



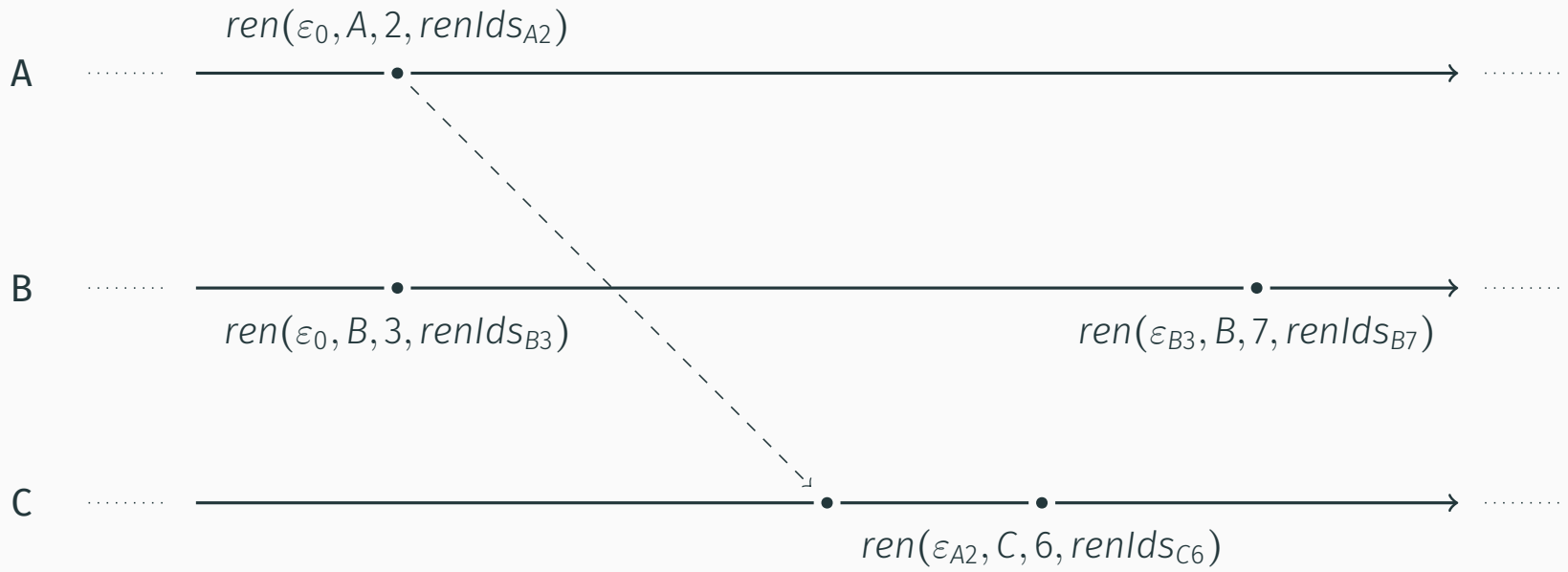
Choisir une époque comme époque cible



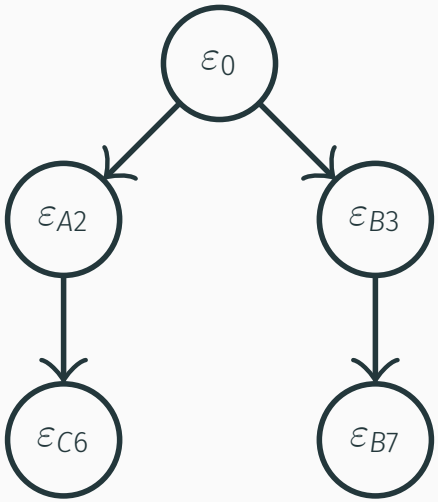
Arbre des époques



Choisir une époque comme époque cible

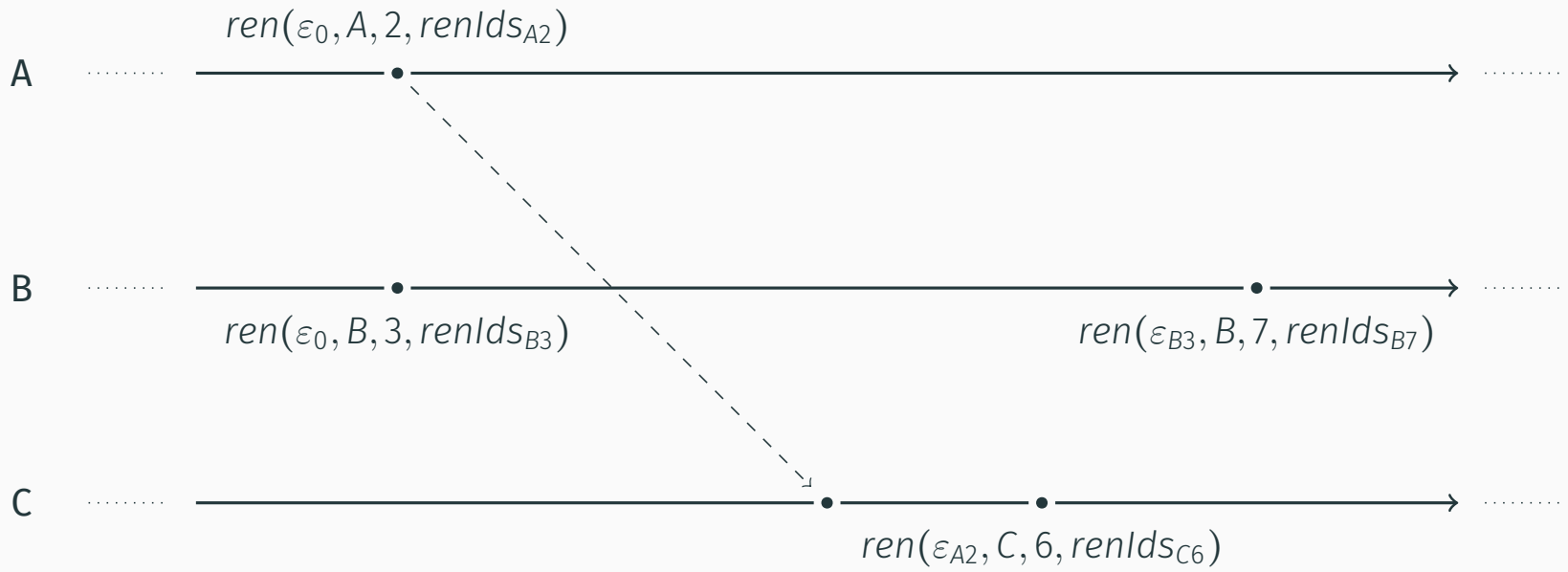


Arbre des époques

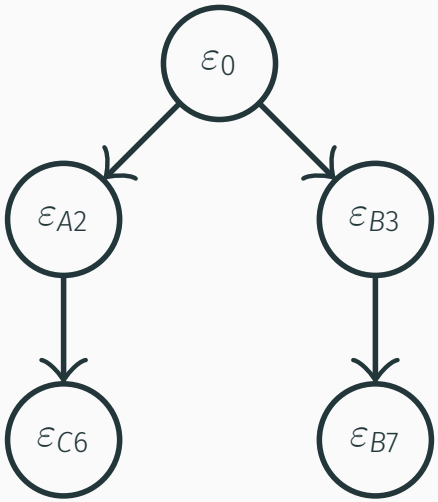


Comment choisir ?

Choisir une époque comme époque cible



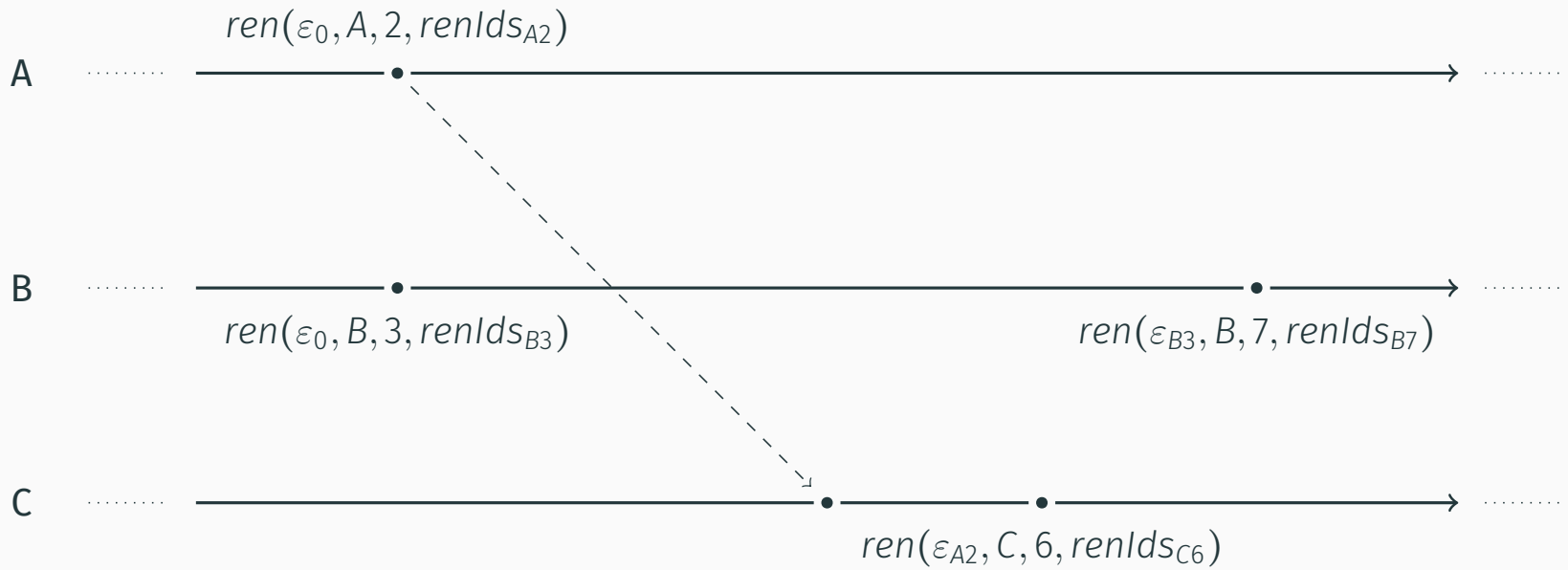
Arbre des époques



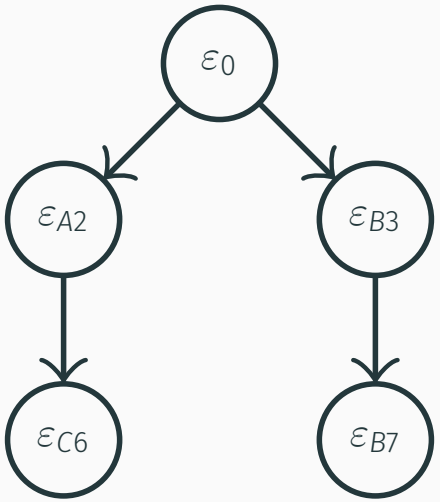
Comment choisir ?

- Définit relation *priority*, notée $<_\varepsilon$, ordre strict total sur les époques

Choisir une époque comme époque cible



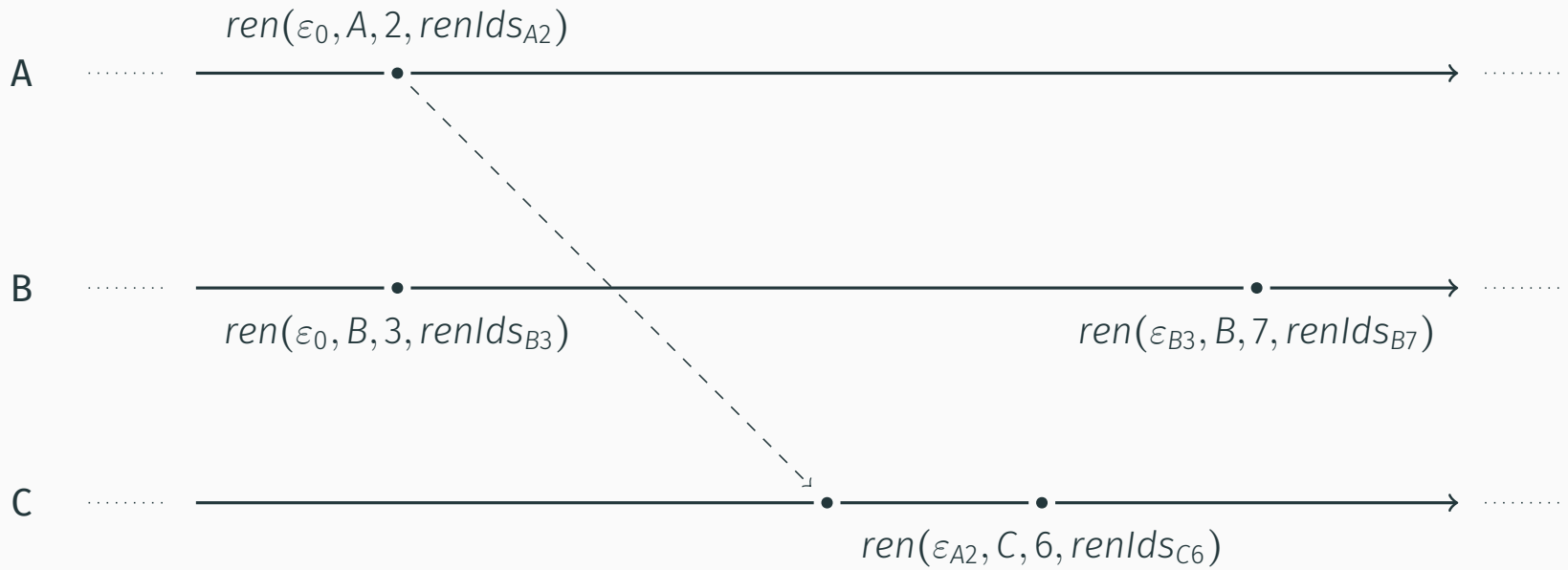
Arbre des époques



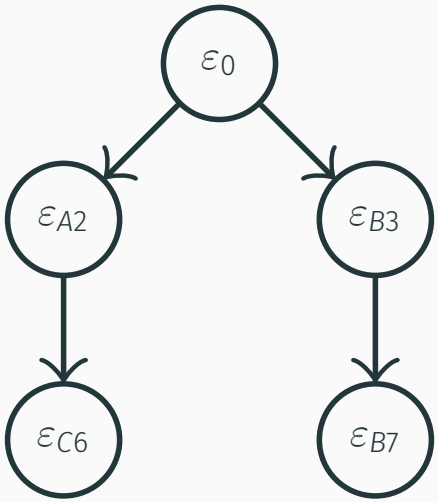
Comment choisir ?

- Définit relation *priority*, notée $<_\varepsilon$, ordre strict total sur les époques
- Utilise ordre lexicographique sur chemins des époques dans l'arbre

Choisir une époque comme époque cible



Arbre des époques



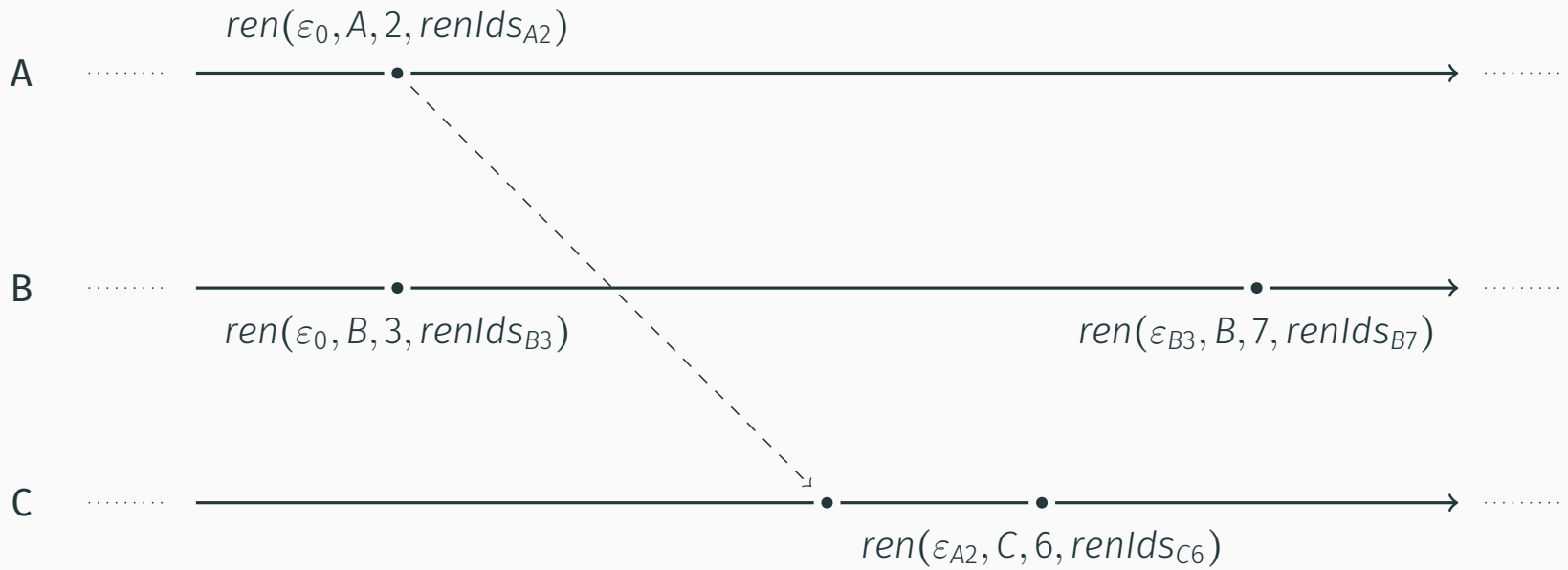
Comment choisir ?

- Définit relation *priority*, notée $<_\varepsilon$, ordre strict total sur les époques
- Utilise ordre lexicographique sur chemins des époques dans l'arbre

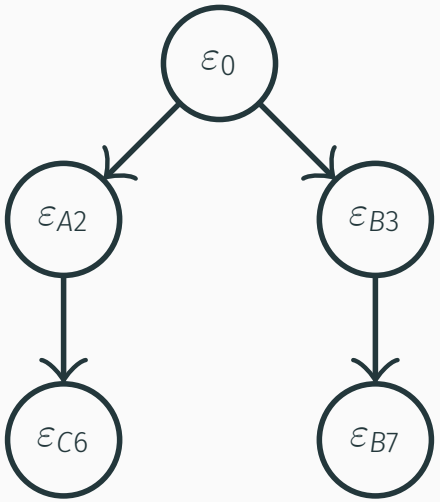
Exemple

$$\varepsilon_0 < \varepsilon_0\varepsilon_{A2}$$

Choisir une époque comme époque cible



Arbre des époques



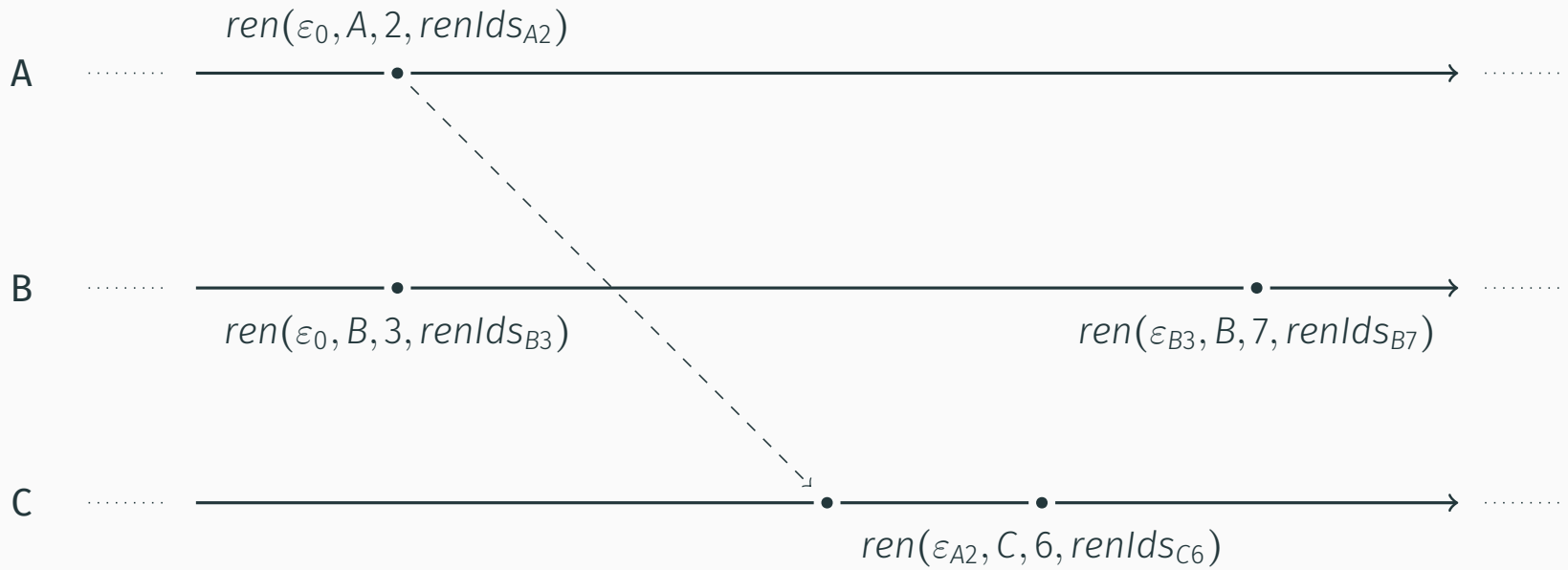
Comment choisir ?

- Définit relation *priority*, notée $<_\varepsilon$, ordre strict total sur les époques
- Utilise ordre lexicographique sur chemins des époques dans l'arbre

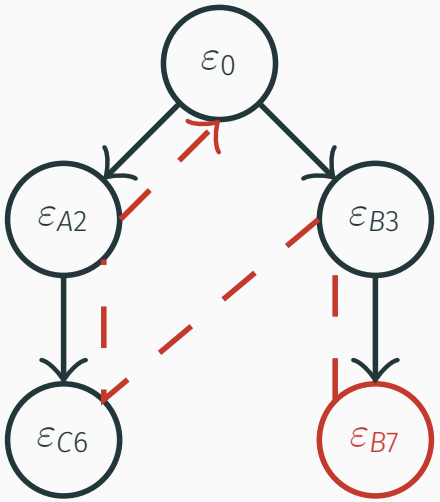
Exemple

$$\varepsilon_0 < \varepsilon_0\varepsilon_{A2} < \varepsilon_0\varepsilon_{A2}\varepsilon_{C6}$$

Choisir une époque comme époque cible



Arbre des époques



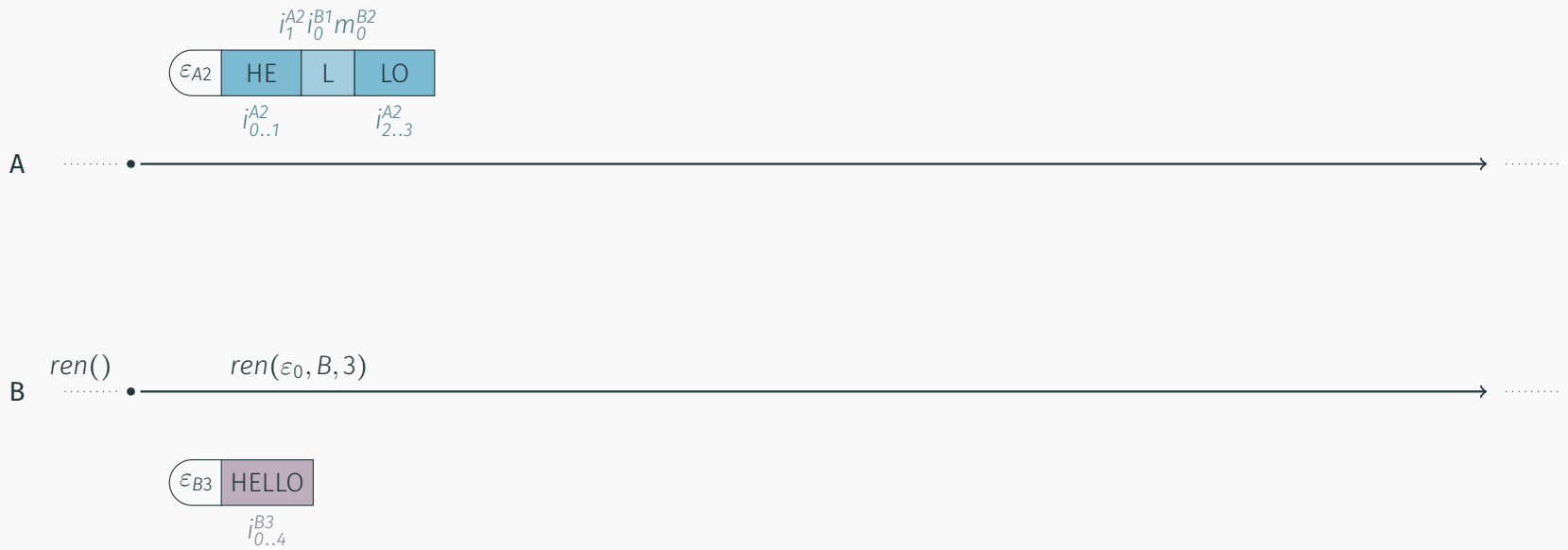
Comment choisir ?

- Définit relation *priority*, notée $<_\varepsilon$, ordre strict total sur les époques
- Utilise ordre lexicographique sur chemins des époques dans l'arbre

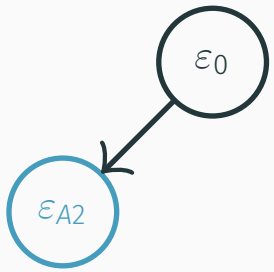
Exemple

$$\varepsilon_0 < \varepsilon_0\varepsilon_{A2} < \varepsilon_0\varepsilon_{A2}\varepsilon_{C6} < \varepsilon_0\varepsilon_{B3}\varepsilon_{B7}$$

Exemple - Calculs des transformations à effectuer



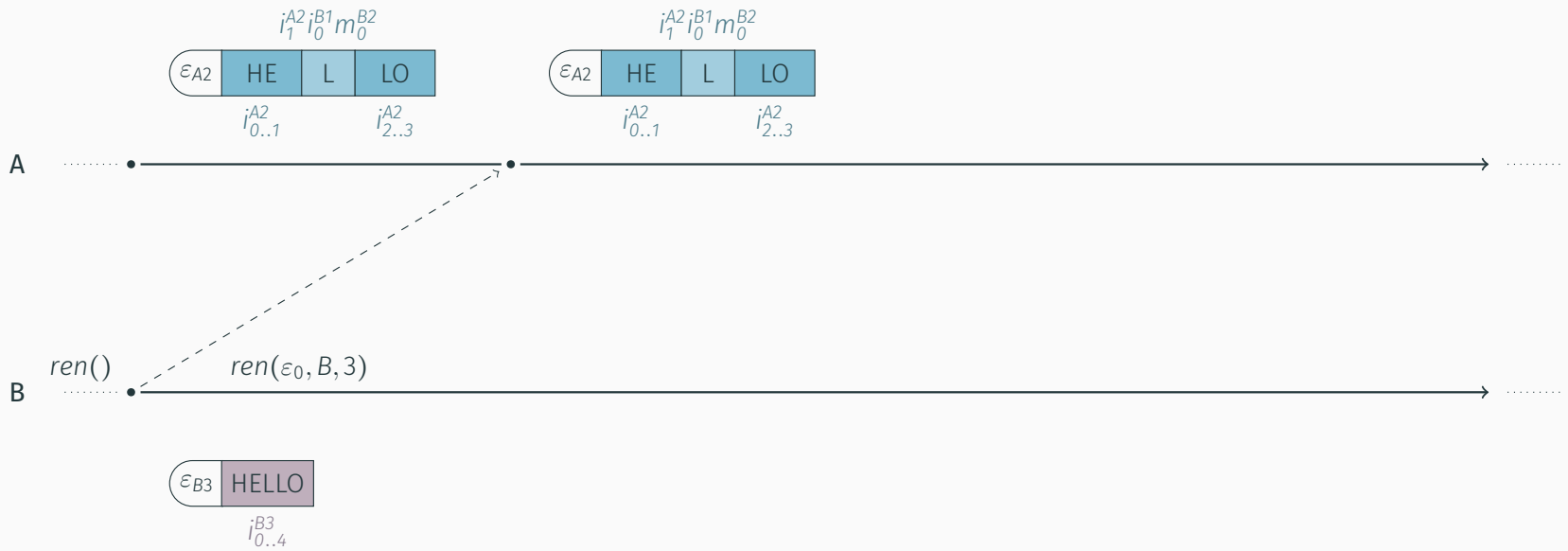
Arbre des époques de A



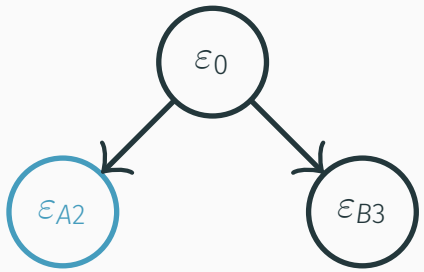
Étapes

- Époque courante : ε_{A2}

Exemple - Calculs des transformations à effectuer



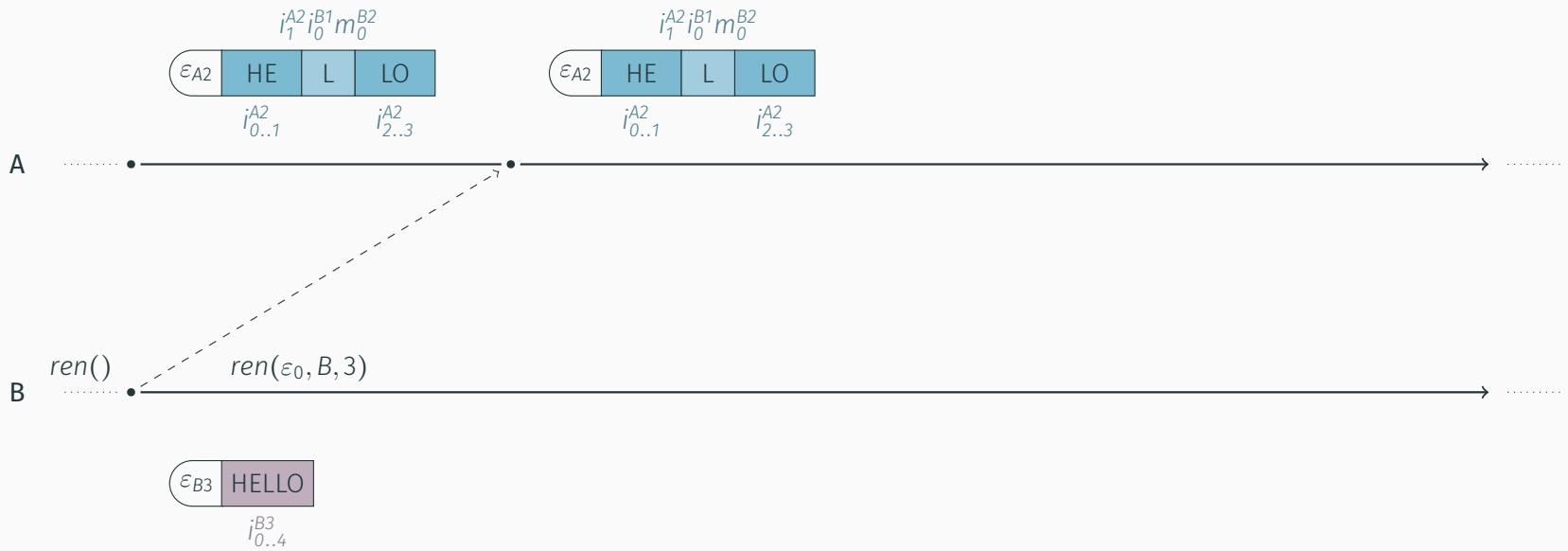
Arbre des époques de A



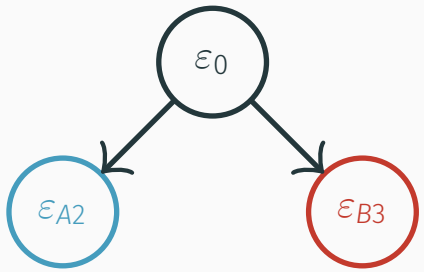
Étapes

- Époque courante : ε_{A2}

Exemple - Calculs des transformations à effectuer



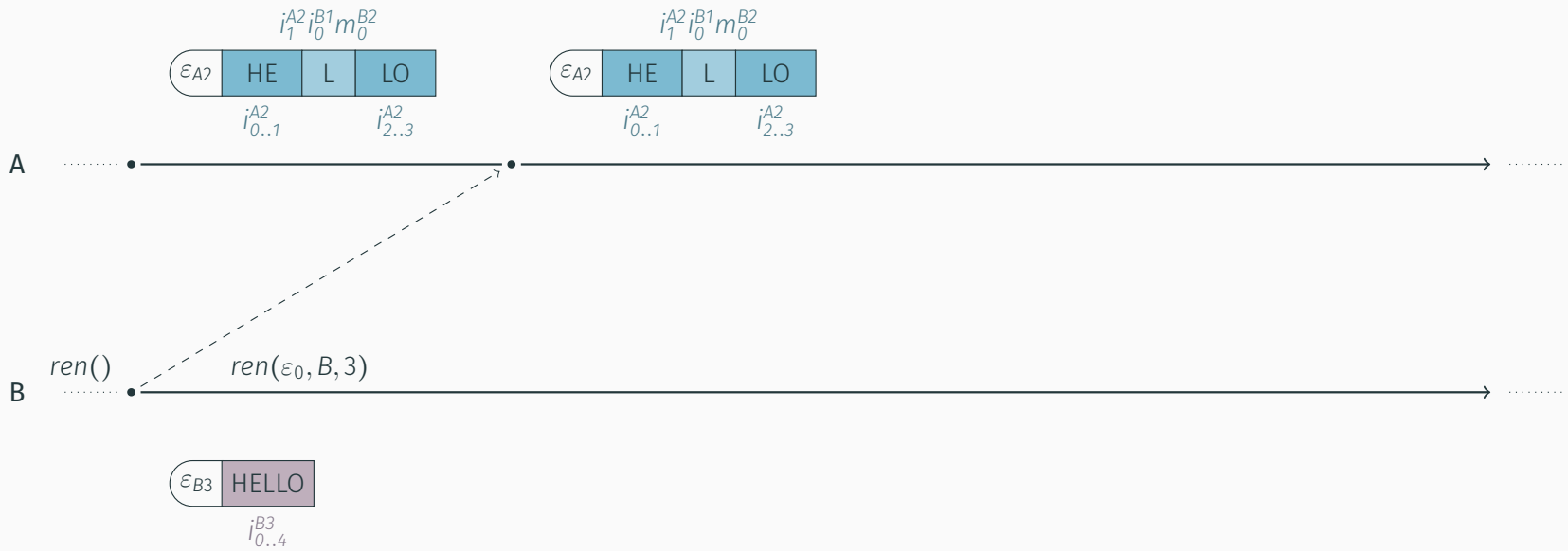
Arbre des époques de A



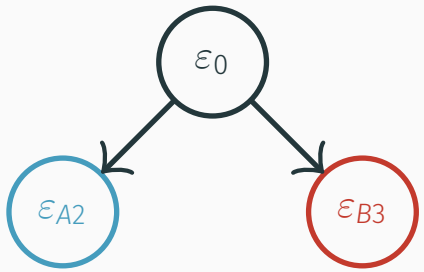
Étapes

- Époque courante : ε_{A2}
- Époque cible : ε_{B3}

Exemple - Calculs des transformations à effectuer



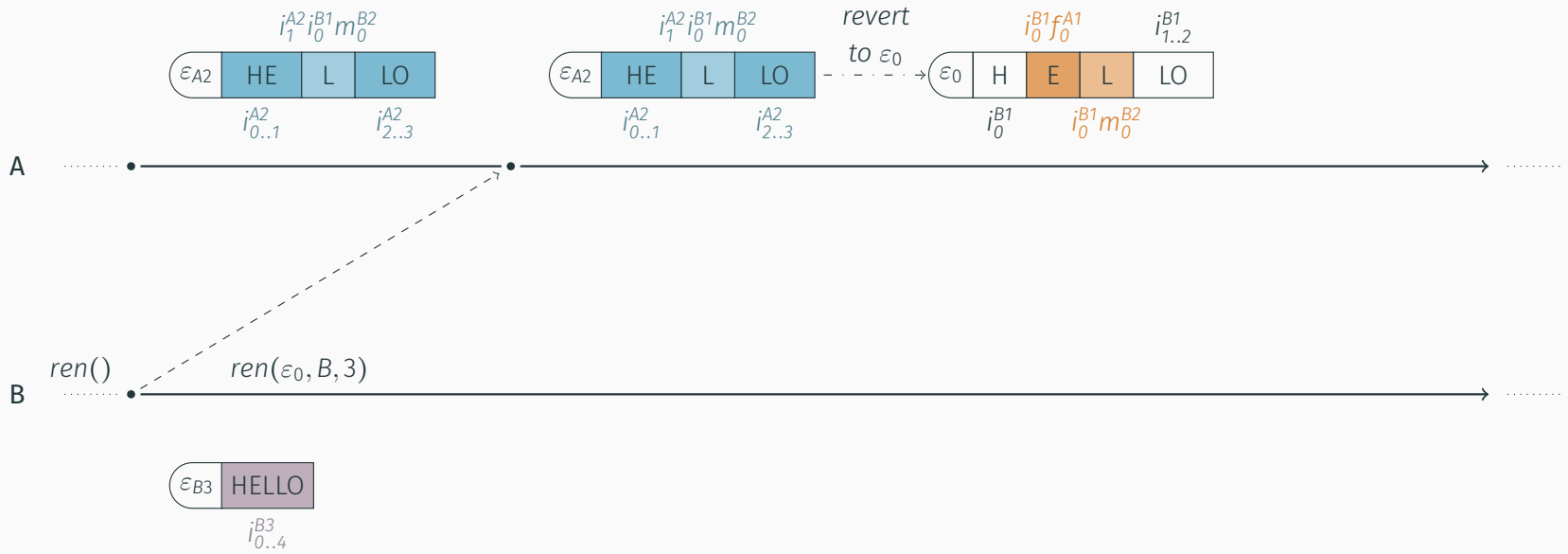
Arbre des époques de A



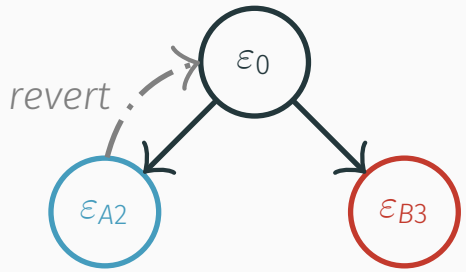
Étapes

- Époque courante : ε_{A2}
- Époque cible : ε_{B3}
- Plus Proche Ancêtre Commun : ε_0

Exemple - Calculs des transformations à effectuer



Arbre des époques de A

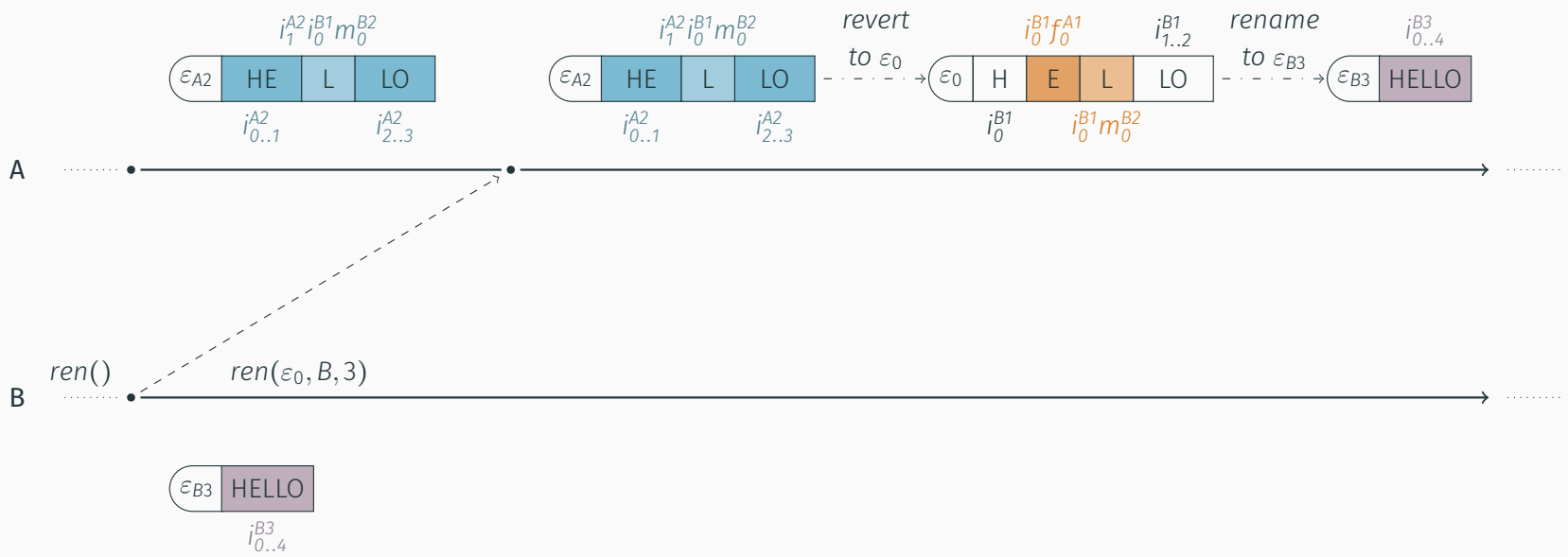


Étapes

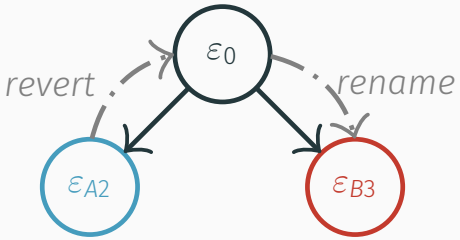
- Époque courante : ε_{A2}
- Époque cible : ε_{B3}
- Plus Proche Ancêtre Commun : ε_0

Doit annuler ε_{A2}

Exemple - Calculs des transformations à effectuer



Arbre des époques de A



Étapes

- Époque courante : ε_{A2}
- Époque cible : ε_{B3}
- Plus Proche Ancêtre Commun : ε_0

Doit annuler ε_{A2} puis appliquer ε_{B3}

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
   - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
   - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
   - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Annuler l’effet d’une opération *rename*

Ajout d’un nouveau mécanisme de transformation

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*
- Distingue plusieurs cas par filtrage par motif

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*
- Distingue plusieurs cas par filtrage par motif

Intuition

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*
- Distingue plusieurs cas par filtrage par motif

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*
- Distingue plusieurs cas par filtrage par motif

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a été inséré après le renommage : doit retourner une valeur qui préserve l'ordre

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*
- Distingue plusieurs cas par filtrage par motif

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a été inséré après le renommage : doit retourner une valeur qui préserve l'ordre
3. *id* a été inséré après ou en concurrence : doit restaurer son ancienne valeur ou générer nouvelle valeur

Annuler l'effet d'une opération *rename*

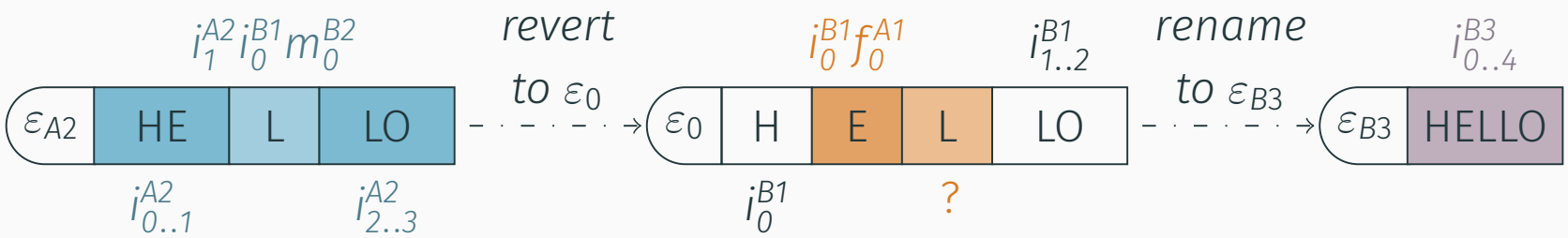
Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*
- Distingue plusieurs cas par filtrage par motif

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a été inséré après le renommage : doit retourner une valeur qui préserve l'ordre
3. *id* a été inséré après ou en concurrence : doit restaurer son ancienne valeur ou générer nouvelle valeur

Exemple de `revertRenameId`

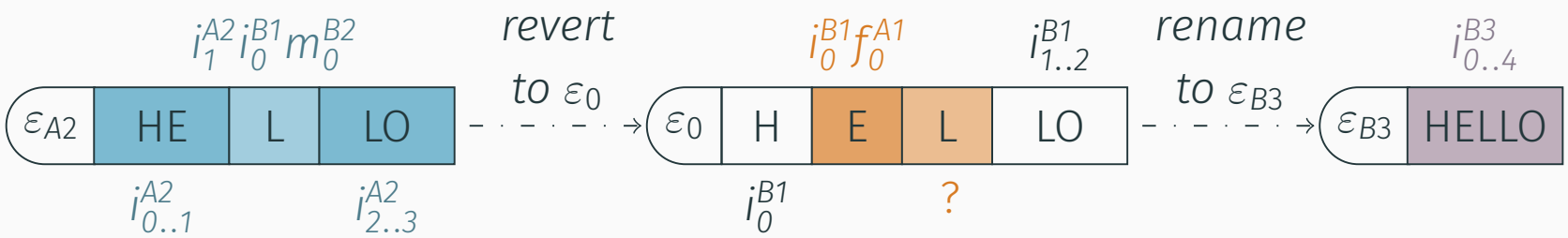


Rappel :

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

Exemple avec $i_1^{A2}\, i_0^{B1}\, m_0^{B2}$

Exemple de `revertRenameId`



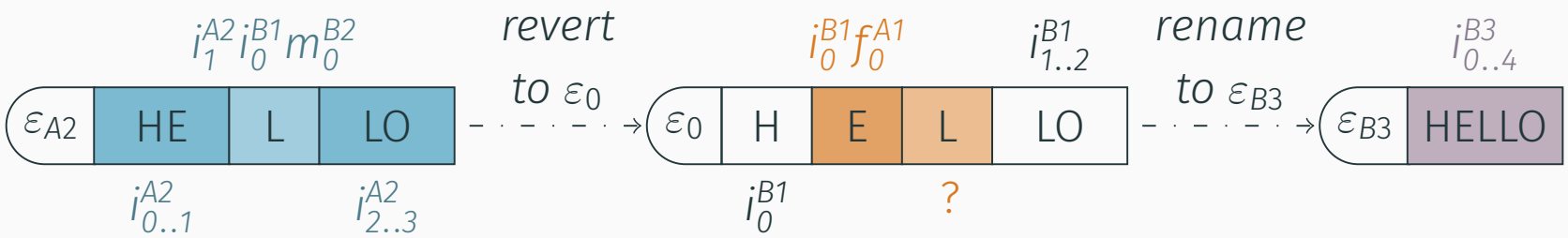
Rappel :

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

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme i_1^{A2} concaténé à $i_0^{B1} m_0^{B2}$: cas 2 ou 3

Exemple de `revertRenameId`



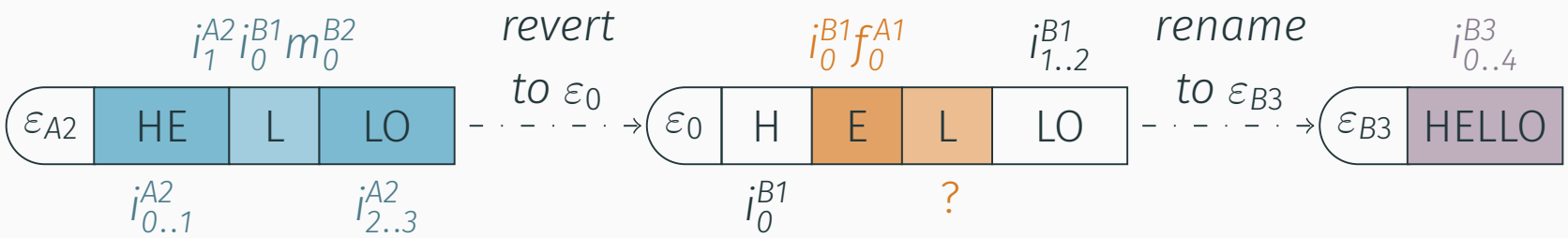
Rappel :

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

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme i_1^{A2} concaténé à $i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Rechercher l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$

Exemple de `revertRenameId`



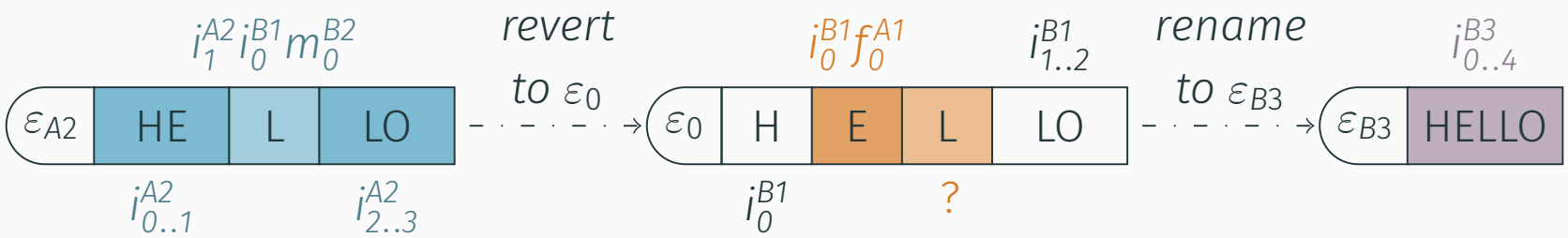
Rappel :

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

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme i_1^{A2} concaténé à $i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Rechercher l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Rechercher l'équivalent de i_2^{A2} dans $renIds_{A2}$: i_1^{B1}

Exemple de `revertRenameId`



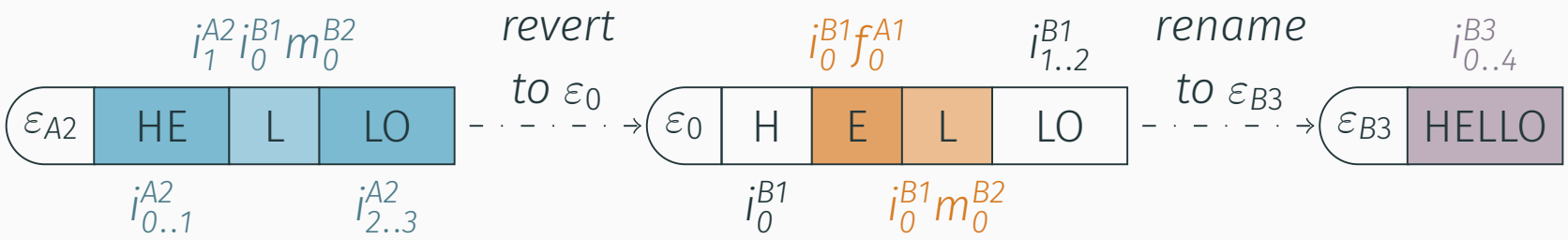
Rappel :

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

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme i_1^{A2} concaténé à $i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Rechercher l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Rechercher l'équivalent de i_2^{A2} dans $renIds_{A2}$: i_1^{B1}
- Comparer $i_0^{B1} m_0^{B2}$ avec ces derniers : $i_0^{B1} f_0^{A1} <_{id} i_0^{B1} m_0^{B2} <_{id} i_1^{B1}$

Exemple de `revertRenameId`



Rappel :

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

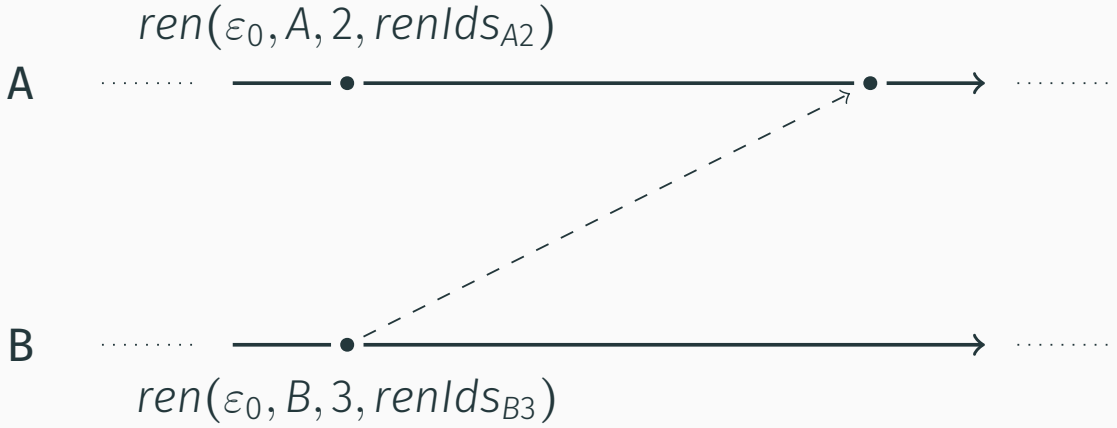
Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

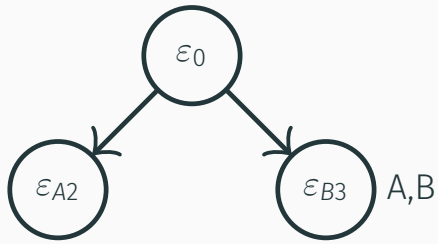
- Est de la forme i_1^{A2} concaténé à $i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Rechercher l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Rechercher l'équivalent de i_2^{A2} dans $renIds_{A2}$: i_1^{B1}
- Comparer $i_0^{B1} m_0^{B2}$ avec ces derniers : $i_0^{B1} f_0^{A1} <_{id} i_0^{B1} m_0^{B2} <_{id} i_1^{B1}$
- Retourner $i_0^{B1} m_0^{B2}$

Mais ça sert à quoi de renommer ?

Puisqu'on doit conserver les *renIds* pour gérer les opérations concurrentes...

Suppression des *renIds* obsolètes (Garbage Collection)

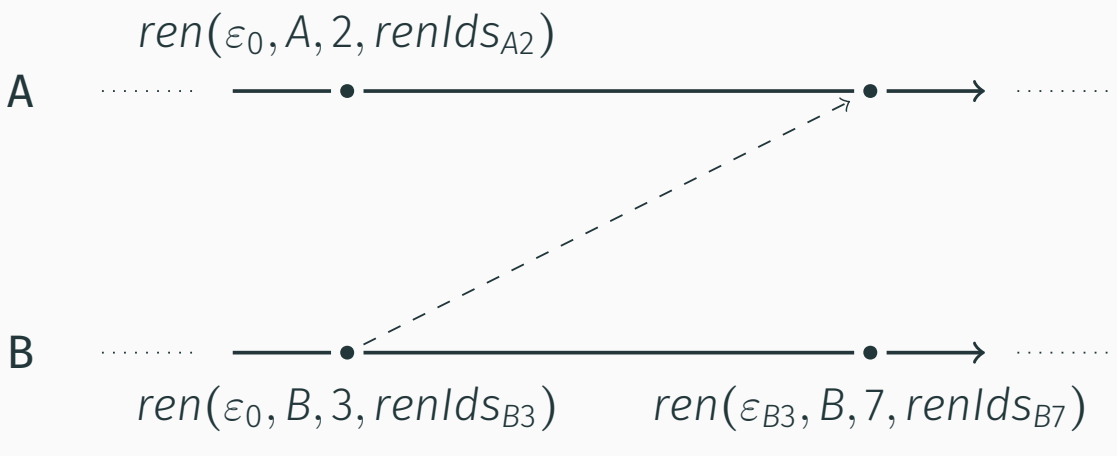


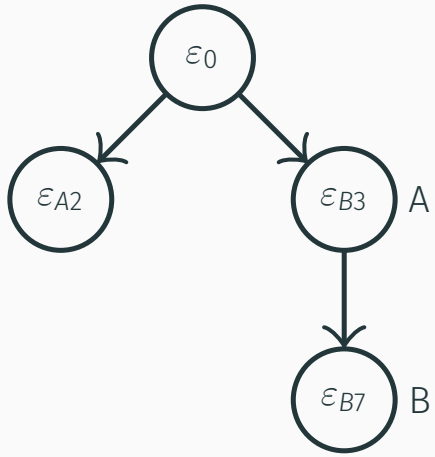


- Besoin de garder *renIds* pour transformer les opérations

[6]. Baquero et al., « Making Operation-Based CRDTs Operation-Based ».

Suppression des *renIds* obsolètes (Garbage Collection)

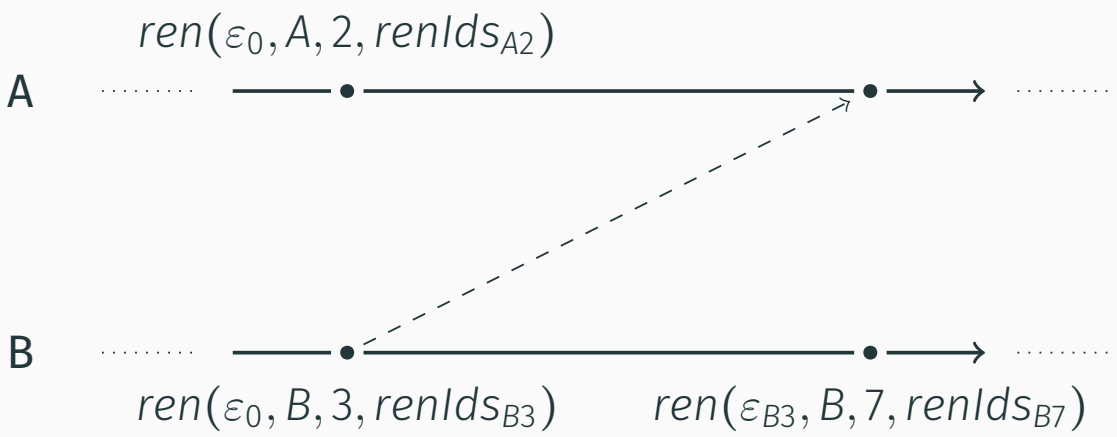


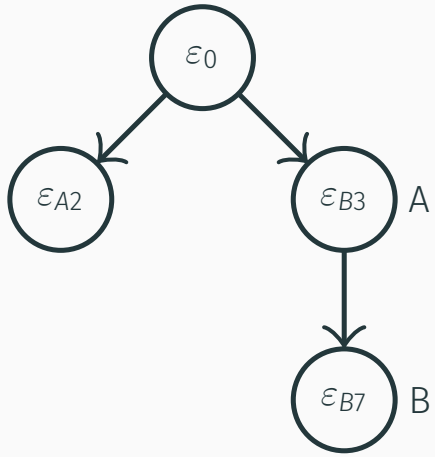


- Besoin de garder *renIds* pour transformer les opérations

[6]. Baquero et al., « Making Operation-Based CRDTs Operation-Based ».

Suppression des *renIds* obsolètes (Garbage Collection)

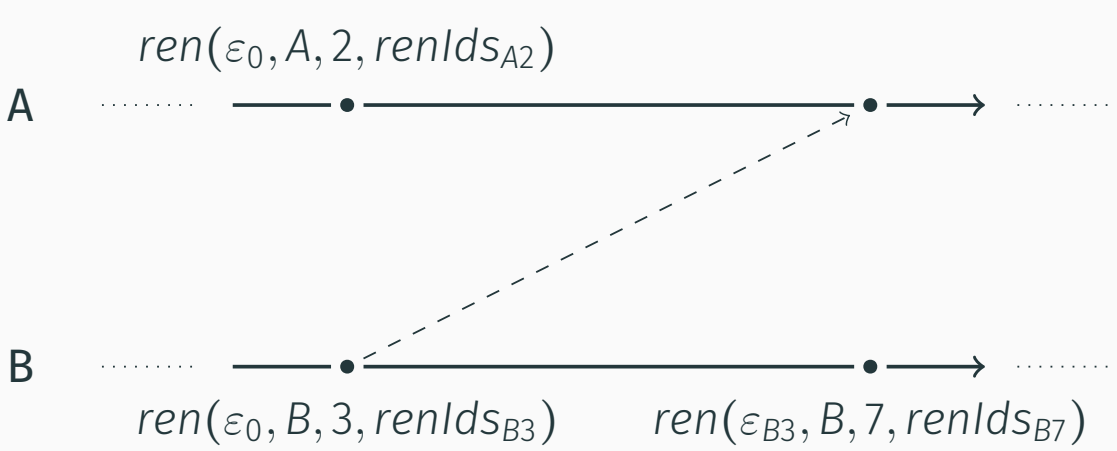


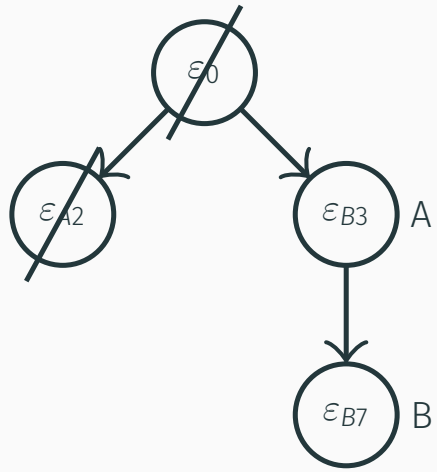


- Besoin de garder *renIds* pour transformer les opérations
- Si plus d'opérations nécessitant des transformations vers une époque donnée…

[6]. Baquero et al., « Making Operation-Based CRDTs Operation-Based ».

Suppression des *renIds* obsolètes (Garbage Collection)

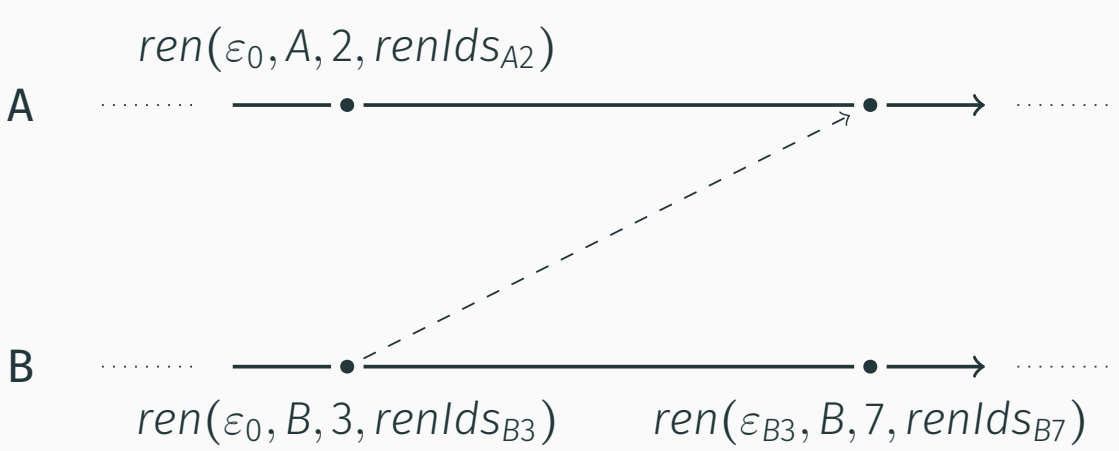


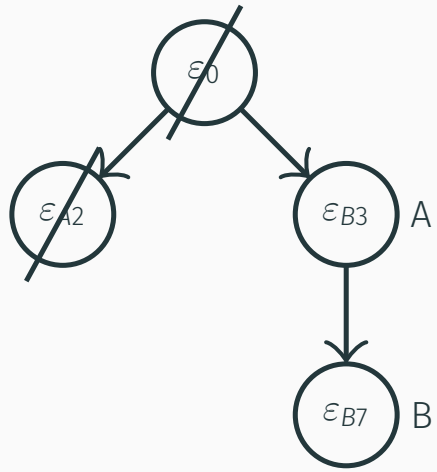


- Besoin de garder *renIds* pour transformer les opérations
- Si plus d'opérations nécessitant des transformations vers une époque donnée…
- …alors époque et *renIds* correspondant obsolètes

[6]. Baquero et al., « Making Operation-Based CRDTs Operation-Based ».

Suppression des *renIds* obsolètes (Garbage Collection)





- Besoin de garder *renIds* pour transformer les opérations
- Si plus d'opérations nécessitant des transformations vers une époque donnée…
- …alors époque et *renIds* correspondant obsolètes

Besoins

- Détecter stabilité causale [6] des opérations *rename*
- Connaître noeuds appartenant au groupe

[6]. Baquero et al., « Making Operation-Based CRDTs Operation-Based ».

Synthèse

Adaptation du mécanisme de renommage pour LogootSplit

Synthèse

Adaptation du mécanisme de renommage pour LogootSplit

- Opération *rename* permettant de minimiser le surcoût de l'état
- Mécanisme de détection des opérations concurrentes
- Algorithme pour intégrer l'effet d'une opération *rename* dans une opération *insert* ou *remove* concurrente

Synthèse

Adaptation du mécanisme de renommage pour LogootSplit

- Opération *rename* permettant de minimiser le surcoût de l'état
- Mécanisme de détection des opérations concurrentes
- Algorithme pour intégrer l'effet d'une opération *rename* dans une opération *insert* ou *remove* concurrente

Conception d'un mécanisme de résolution de conflits pour opérations *rename* concurrentes

Synthèse

Adaptation du mécanisme de renommage pour LogootSplit

- Opération *rename* permettant de minimiser le surcoût de l'état
- Mécanisme de détection des opérations concurrentes
- Algorithme pour intégrer l'effet d'une opération *rename* dans une opération *insert* ou *remove* concurrente

Conception d'un mécanisme de résolution de conflits pour opérations *rename* concurrentes

- Mécanisme pour désigner une époque comme l'époque cible, sans coordination
- Algorithme pour annuler l'effet d'une opération *rename*

Synthèse

Adaptation du mécanisme de renommage pour LogootSplit

- Opération *rename* permettant de minimiser le surcoût de l'état
- Mécanisme de détection des opérations concurrentes
- Algorithme pour intégrer l'effet d'une opération *rename* dans une opération *insert* ou *remove* concurrente

Conception d'un mécanisme de résolution de conflits pour opérations *rename* concurrentes

- Mécanisme pour désigner une époque comme l'époque cible, sans coordination
- Algorithme pour annuler l'effet d'une opération *rename*

Conception d'un mécanisme de suppression des époques obsolètes

RenamableLogootSplit

Validation

Objectifs

- Montrer convergence des noeuds
- Montrer que mécanisme de renommage améliore performances de la séquence répliquée (mémoire, calculs, bande-passante)

Objectifs

- Montrer convergence des noeuds
- Montrer que mécanisme de renommage améliore performances de la séquence répliquée (mémoire, calculs, bande-passante)

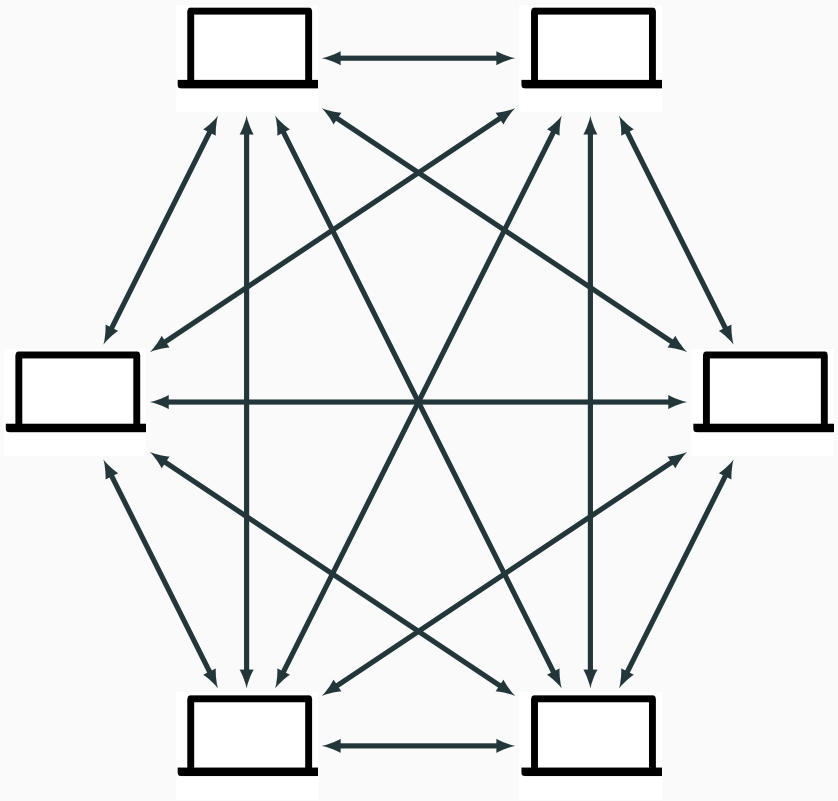
Conduite d'une évaluation expérimentale

Absence d’un jeu de données de sessions d’édition collaborative

Absence d’un jeu de données de sessions d’édition collaborative

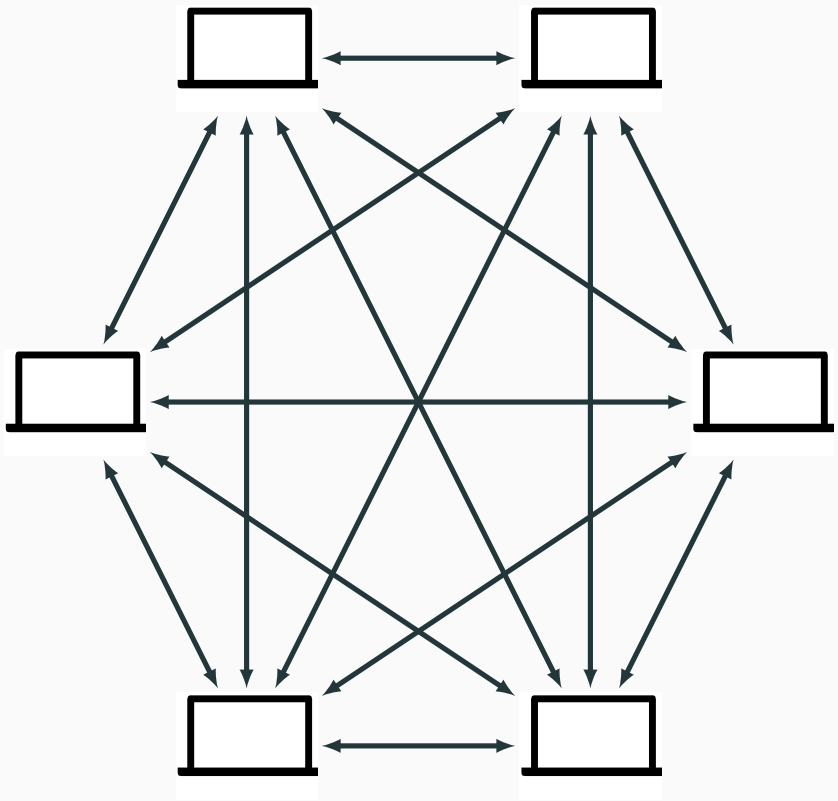
Mise en place de simulations pour générer un jeu de données

Simulations - Architecture



- 10 noeuds éditent collaborativement un document
- Utilisent soit LogootSplit (LS), soit RenamableLogootSplit (RLS)

Simulations - Architecture



- 10 noeuds éditent collaborativement un document
- Utilisent soit LogootSplit (LS), soit RenamableLogootSplit (RLS)
- Topologie réseau entièrement maillée
- Ne considère pas pannes ou pertes de message

Simulations - Modifications

- Phase 1 (génération du contenu) : Beaucoup d'insertions, quelques suppressions (80/20%)
- Phase 2 (édition) : Équilibre insertions/suppressions (50/50%)
- Noeuds passent à la phase 2 quand document atteint taille donnée (15 pages - 60k caractères)

Simulations - Modifications

- Phase 1 (génération du contenu) : Beaucoup d'insertions, quelques suppressions (80/20%)
- Phase 2 (édition) : Équilibre insertions/suppressions (50/50%)
- Noeuds passent à la phase 2 quand document atteint taille donnée (15 pages - 60k caractères)

- Noeuds terminent quand ensemble des noeuds a effectué nombre donné de modifications (15k)...
- ...et intégré celles des autres (150k au total)

Simulations - Mécanisme de renommage

- Noeuds désignés comme *noeuds de renommage* (1 à 4)
- Noeuds de renommage effectue un renommage à toutes les 30k opérations qu'ils intègrent (5 opérations *rename* par noeud de renommage)
- Opérations *rename* générées à un point donné sont concurrentes

Simulations - Sorties

- Instantané de l'état de chaque noeud à différents points de la simulation (10k opérations et état final)
- Journal des opérations de chaque noeud

*. Code des simulations et benchmarks :
`https://github.com/coast-team/mute-bot-random`

Simulations - Sorties

- Instantané de l'état de chaque noeud à différents points de la simulation (10k opérations et état final)
- Journal des opérations de chaque noeud

Permet de conduire évaluations sur ces données *

*. Code des simulations et benchmarks :
`https://github.com/coast-team/mute-bot-random`

Convergence

Intuition

Comparer l'état final des différents noeuds d'une session pour confirmer l'absence de divergence

Convergence

Intuition

Comparer l'état final des différents noeuds d'une session pour confirmer l'absence de divergence

- Ensemble des noeuds convergent

Convergence

Intuition

Comparer l'état final des différents noeuds d'une session pour confirmer l'absence de divergence

- Ensemble des noeuds convergent
- Un résultat empirique, pas une preuve...
- ...mais un premier pas vers la validation de RLS

Surcoût en métadonnées - 1 noeud de renommage

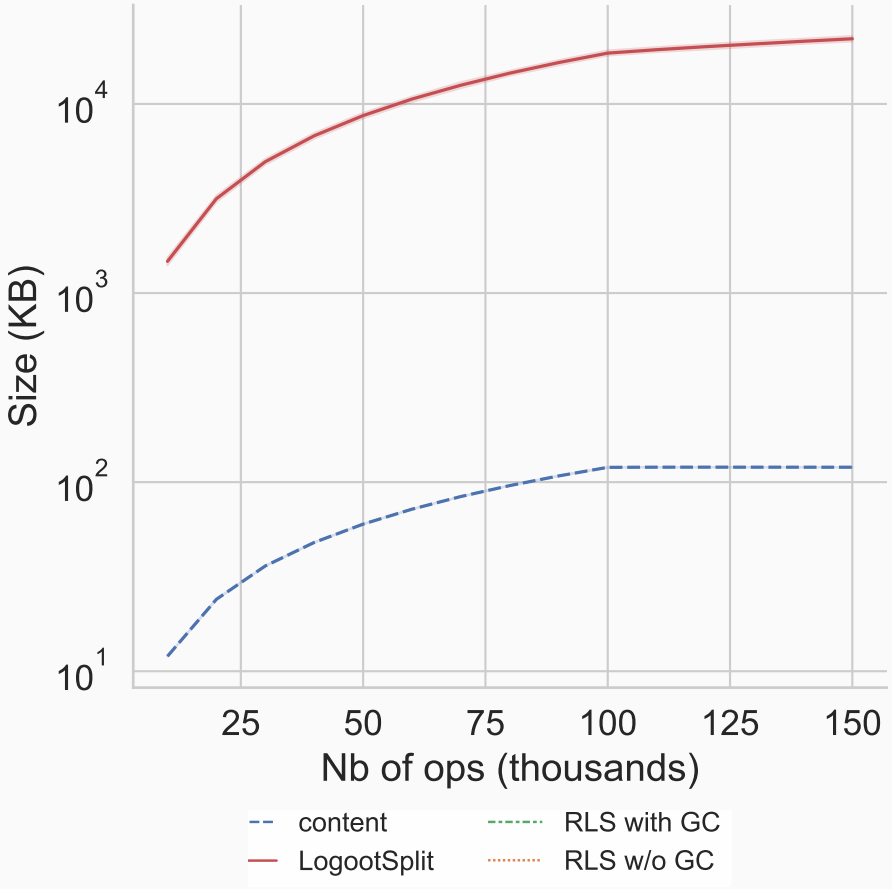
Intuition

Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage

Surcoût en métadonnées - 1 noeud de renommage

Intuition

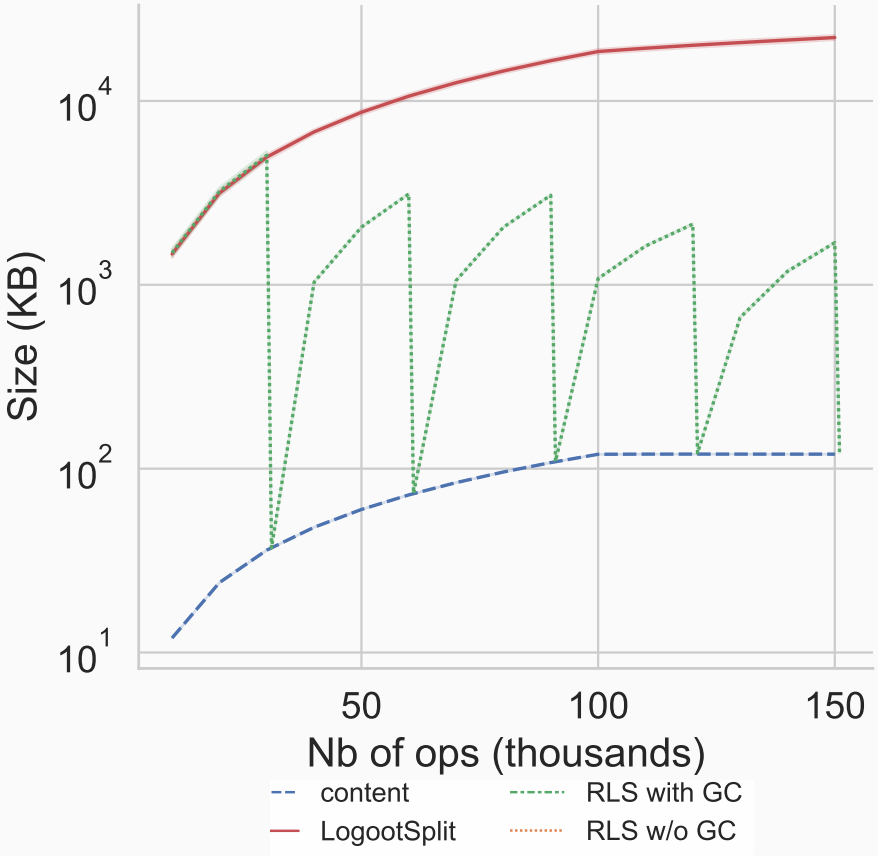
Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage



Surcoût en métadonnées - 1 noeud de renommage

Intuition

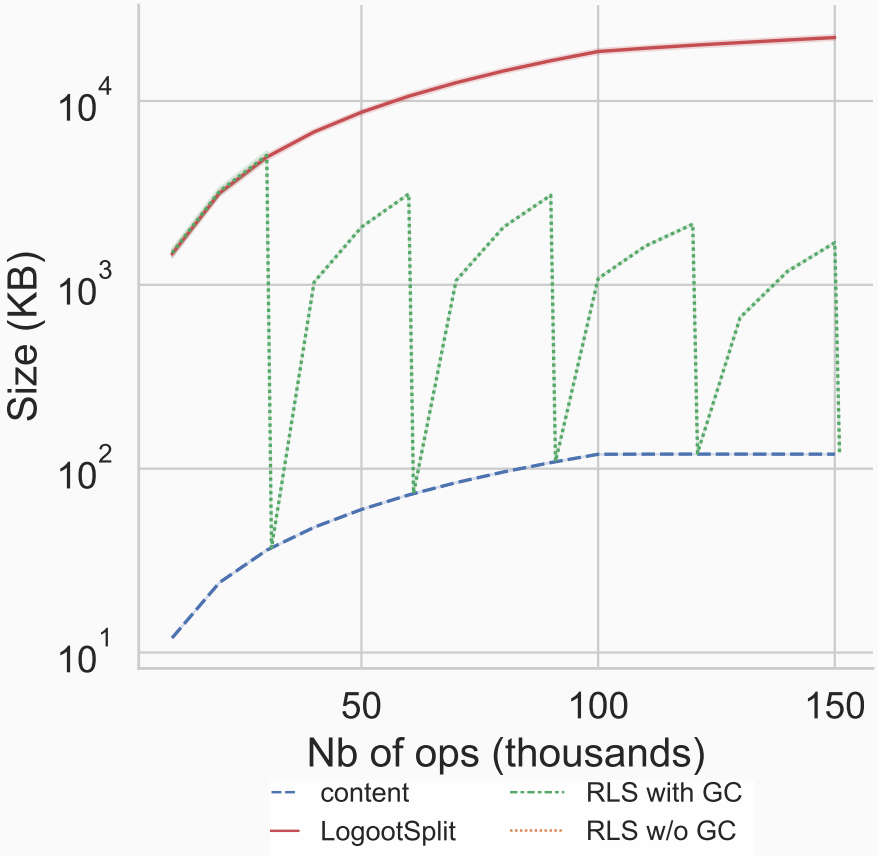
Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage



Surcoût en métadonnées - 1 noeud de renommage

Intuition

Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage



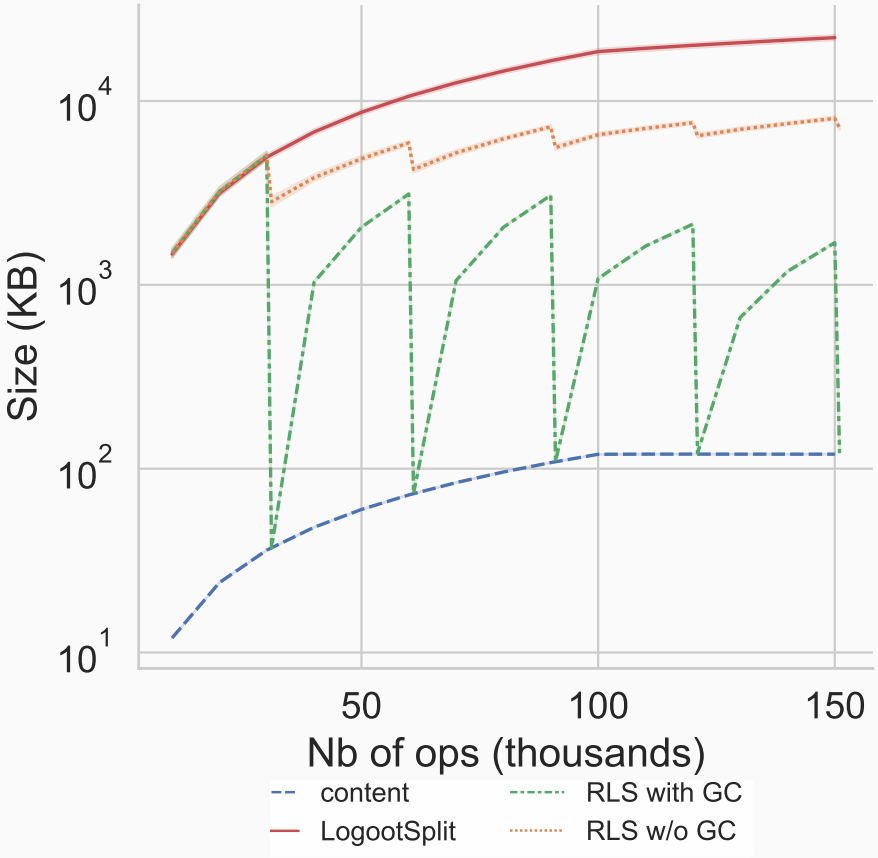
Observations

- Opération *rename* réinitialise surcoût du CRDT, si GC possible

Surcoût en métadonnées - 1 noeud de renommage

Intuition

Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage



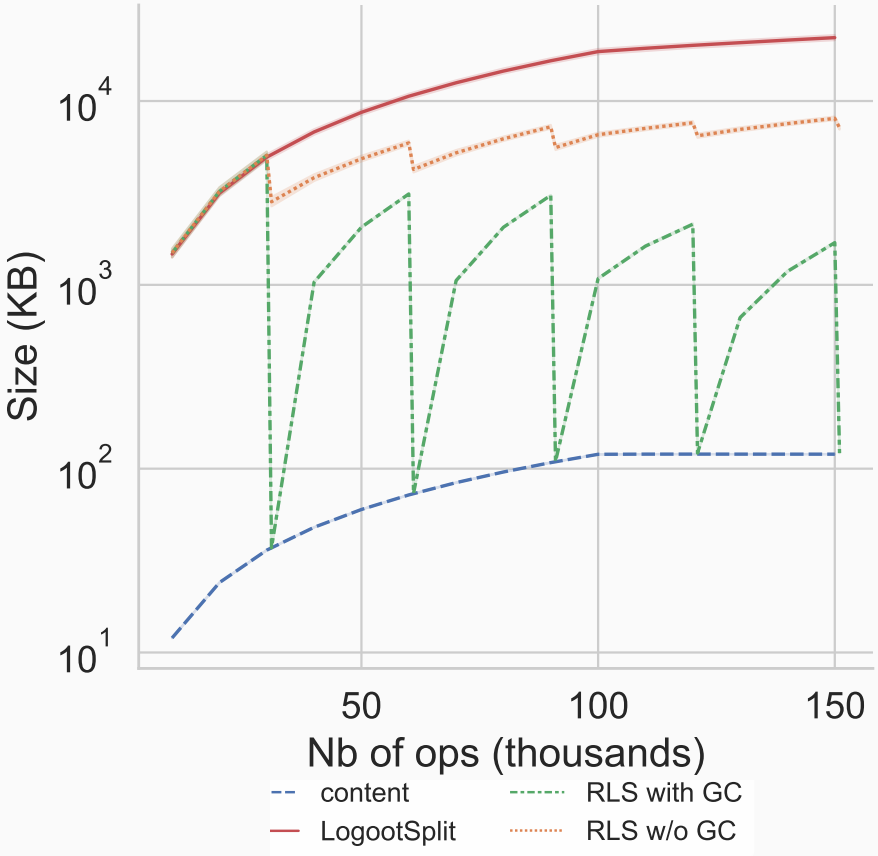
Observations

- Opération *rename* réinitialise surcoût du CRDT, si GC possible

Surcoût en métadonnées - 1 noeud de renommage

Intuition

Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage



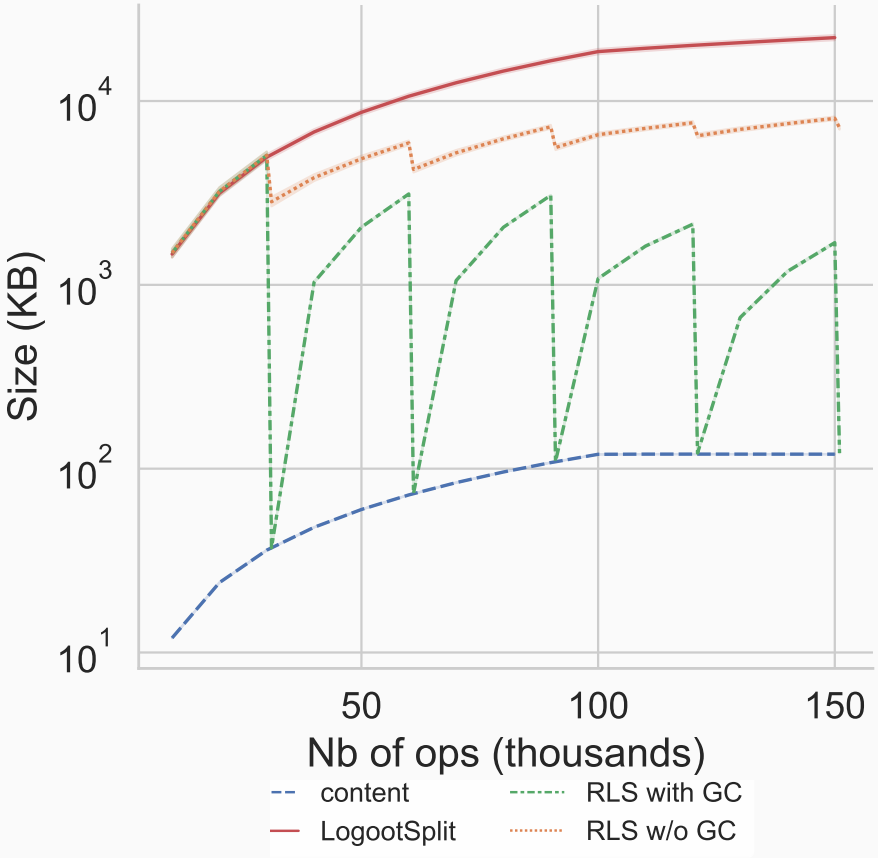
Observations

- Opération *rename* réinitialise surcoût du CRDT, si GC possible
- Opération *rename* réduit de 66% surcoût du CRDT, si GC impossible

Surcoût en métadonnées - 1 noeud de renommage

Intuition

Mesurer évolution de la taille de la structure de données à partir des instantanés des sessions avec 1 seul noeud de renommage



Observations

- Opération *rename* réinitialise surcoût du CRDT, si GC possible
- Opération *rename* réduit de 66% surcoût du CRDT, si GC impossible

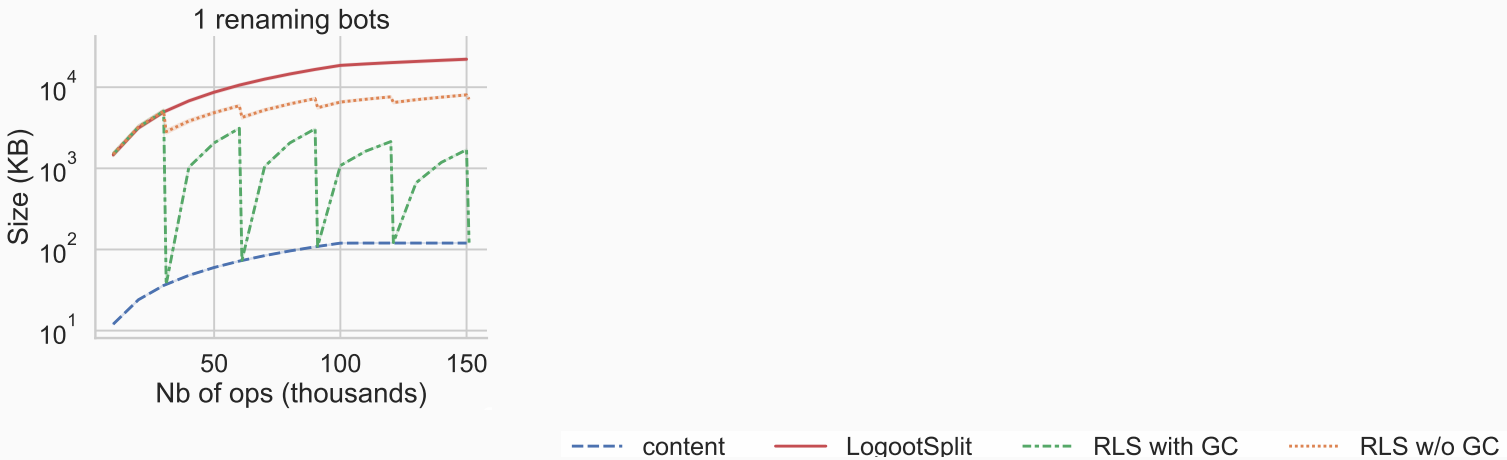
Explications

- Un seul bloc après une opération *rename*
- Réduction du nombre de blocs réduit les métadonnées utilisée par l'implémentation de la Séquence (arbre AVL)

Surcoût en métadonnées - 1 à 4 noeuds de renommage

Intuition

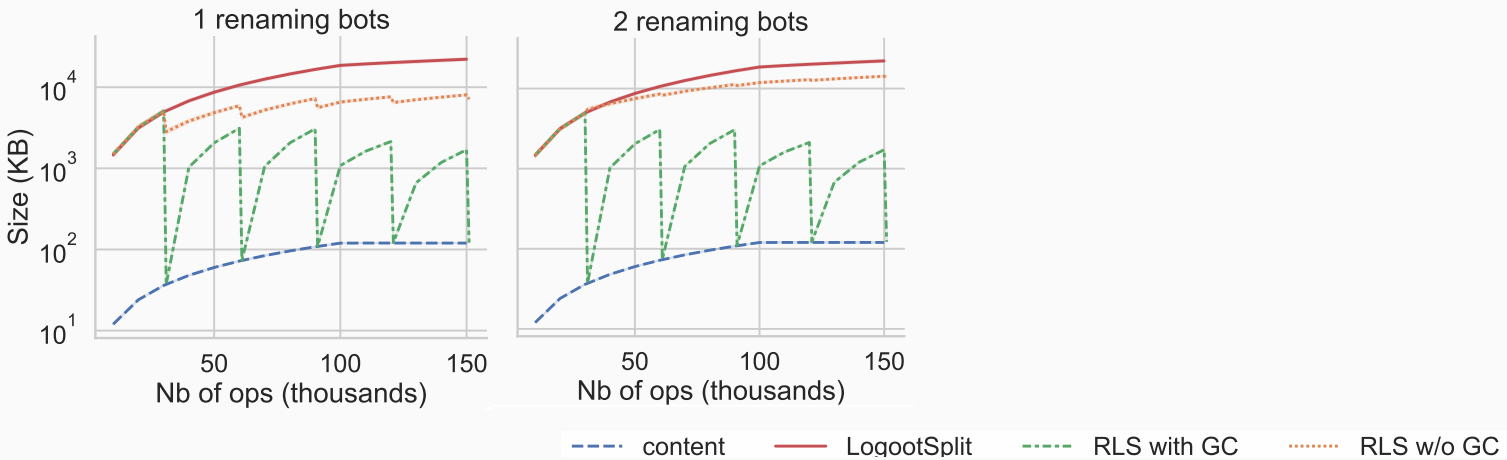
Mesurer évolution de la taille de la structure de données en fonction du nombre de noeuds de renommage



Surcoût en métadonnées - 1 à 4 noeuds de renommage

Intuition

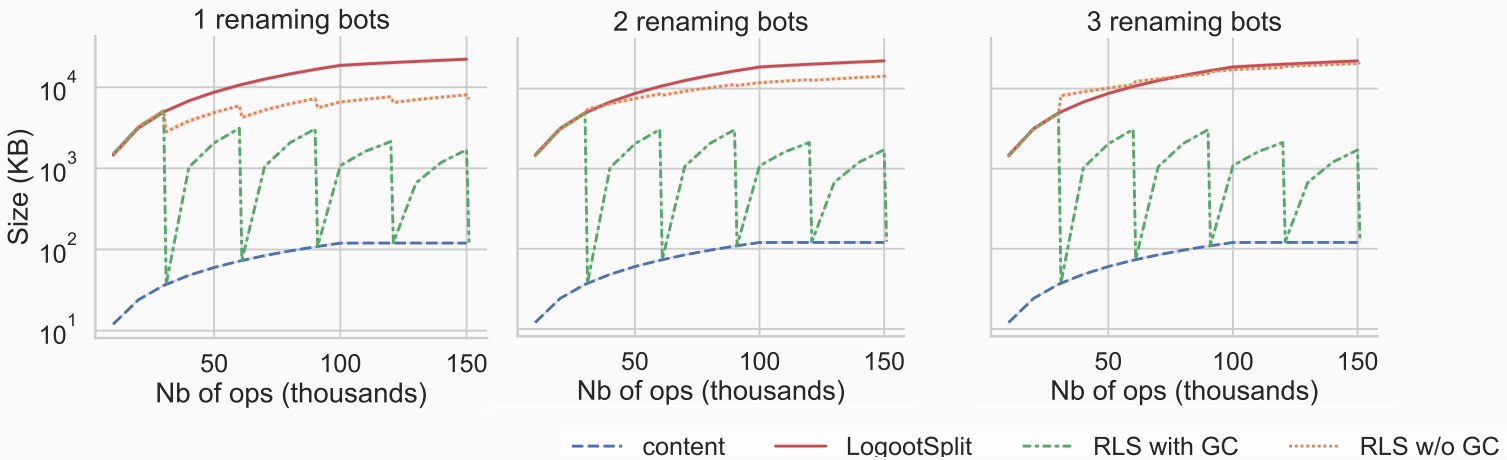
Mesurer évolution de la taille de la structure de données en fonction du nombre de noeuds de renommage



Surcoût en métadonnées - 1 à 4 noeuds de renommage

Intuition

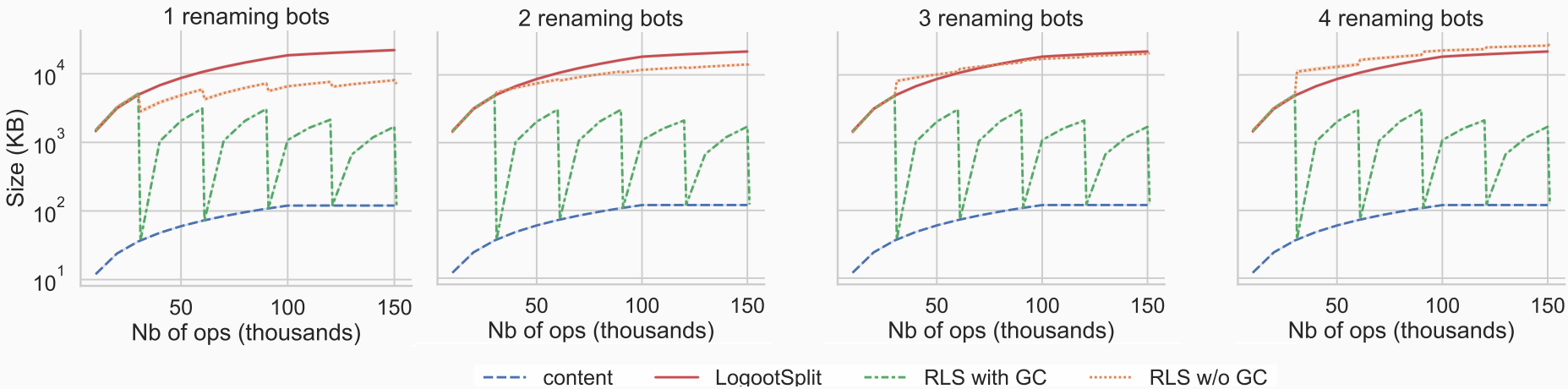
Mesurer évolution de la taille de la structure de données en fonction du nombre de noeuds de renommage



Surcoût en métadonnées - 1 à 4 noeuds de renommage

Intuition

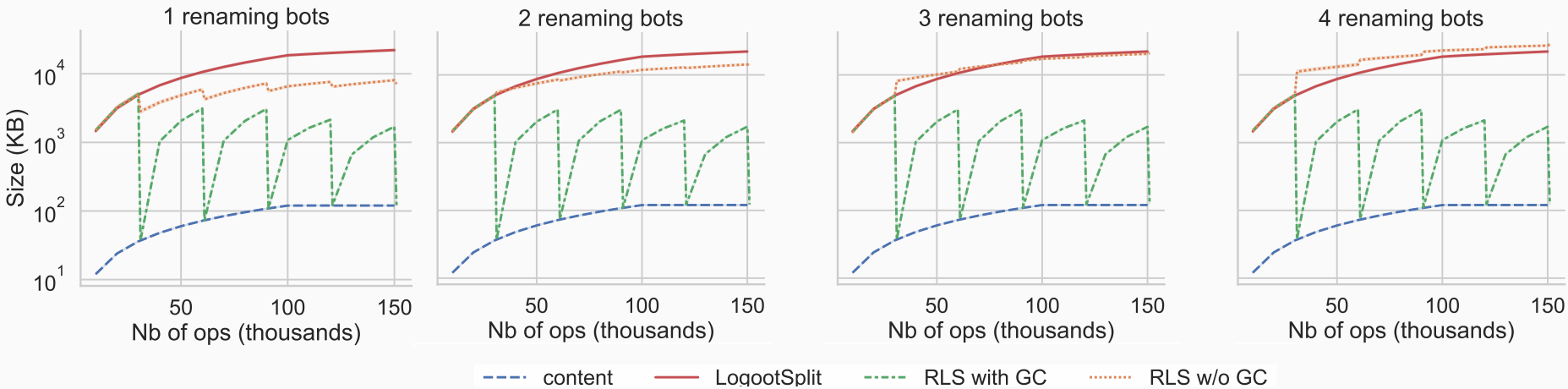
Mesurer évolution de la taille de la structure de données en fonction du nombre de noeuds de renommage



Surcoût en métadonnées - 1 à 4 noeuds de renommage

Intuition

Mesurer évolution de la taille de la structure de données en fonction du nombre de noeuds de renommage



- Aucun impact si GC possible
- Métadonnées (époque et *renIds*) de chaque opération *rename* s'additionne si GC impossible

Surcoût en calculs - Opérations *insert* et *remove*

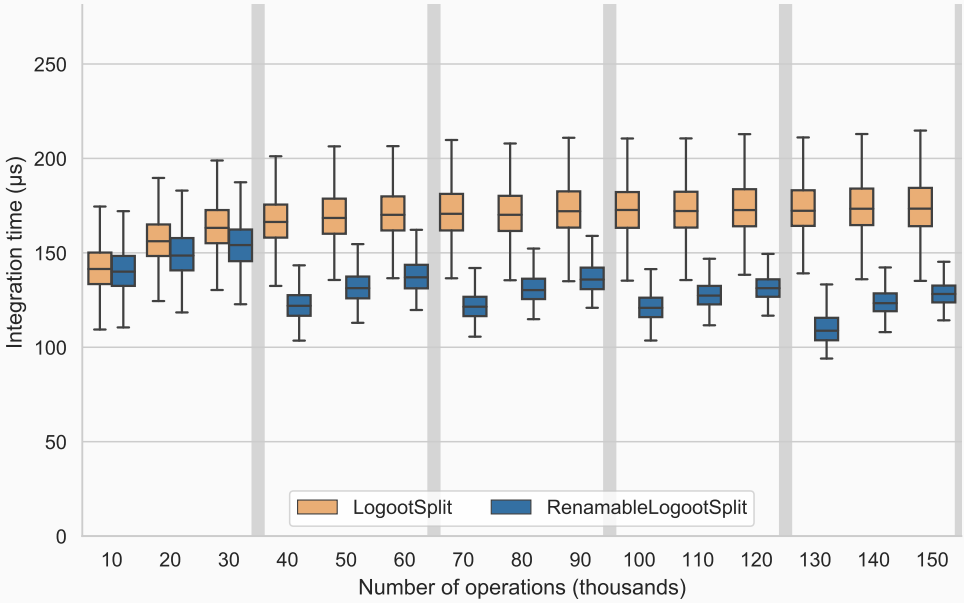
Intuition

Mesurer temps d'intégration local et distant d'opérations *insert* à différents stades de la collaboration

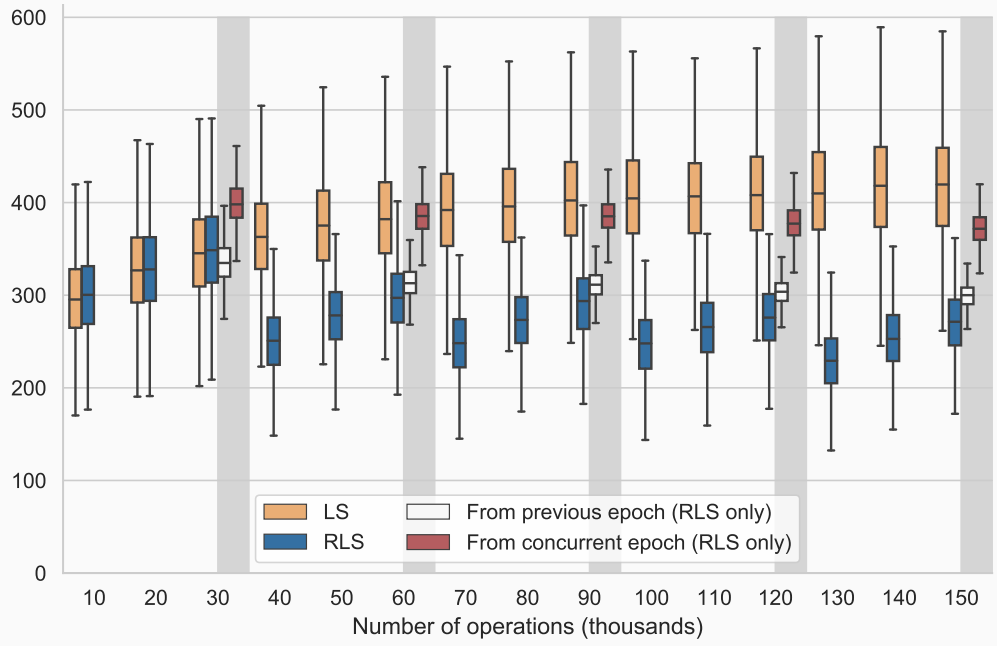
Surcoût en calculs - Opérations *insert* et *remove*

Intuition

Mesurer temps d'intégration local et distant d'opérations *insert* à différents stades de la collaboration



(a) Temps intégration modifs locales

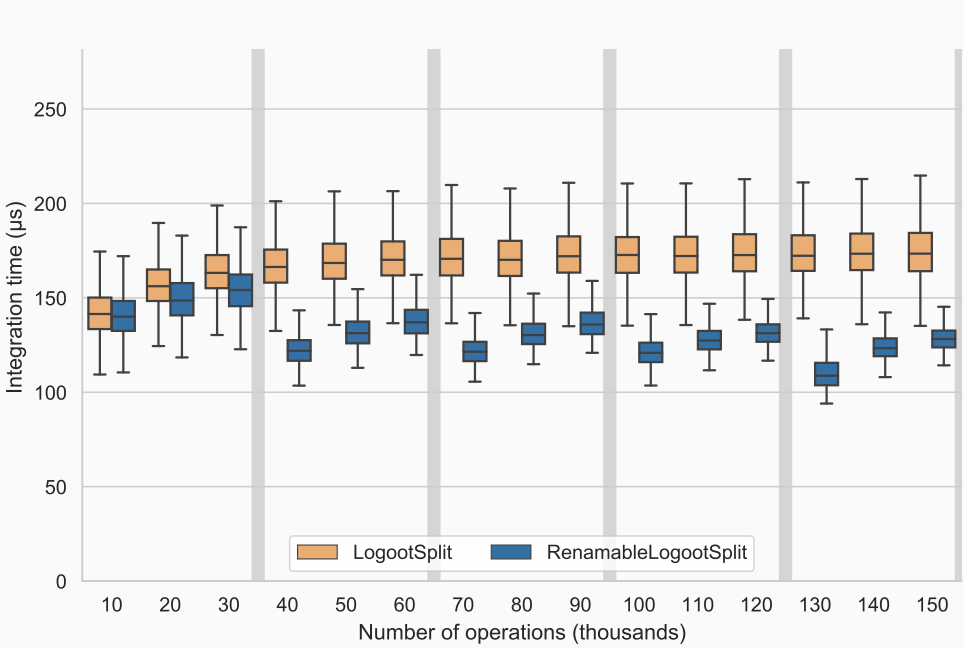


(b) Temps intégration modifs distantes

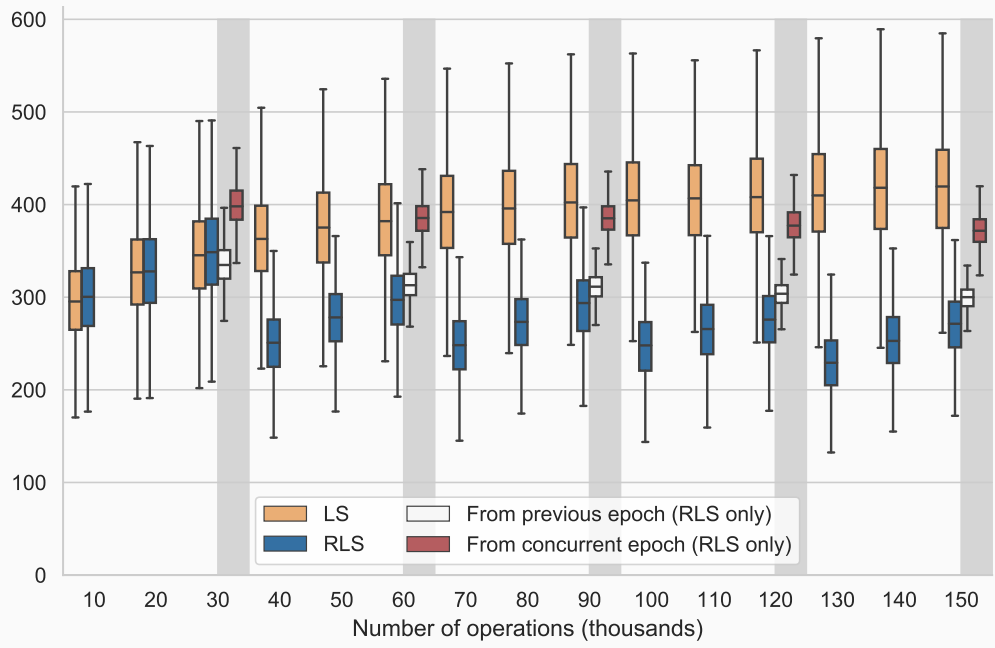
Surcoût en calculs - Opérations *insert* et *remove*

Intuition

Mesurer temps d'intégration local et distant d'opérations *insert* à différents stades de la collaboration



(a) Temps intégration modifs locales



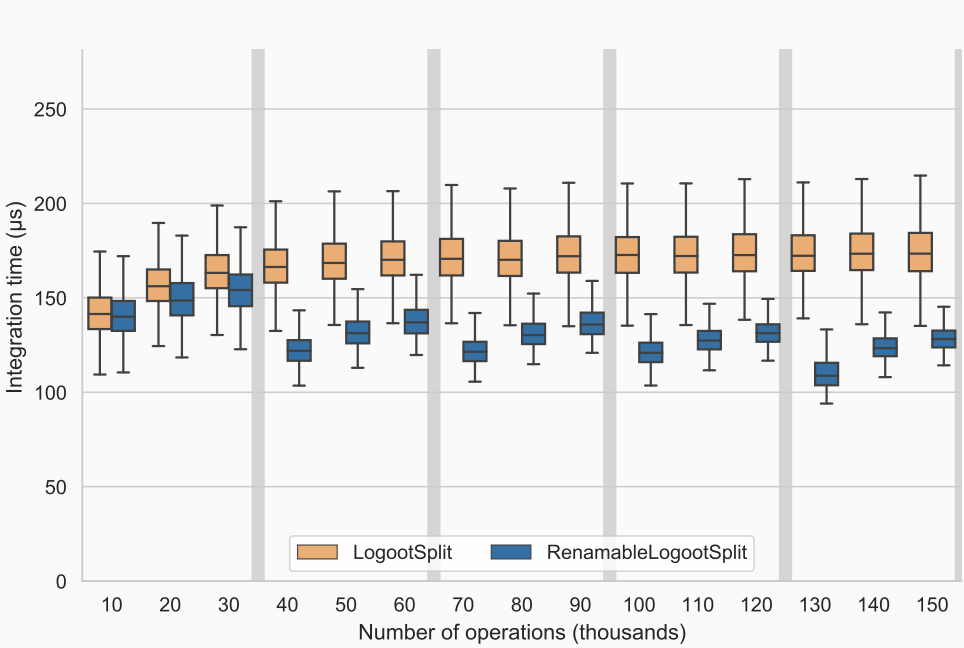
(b) Temps intégration modifs distantes

- Opération *rename* réduit les temps d'intégration

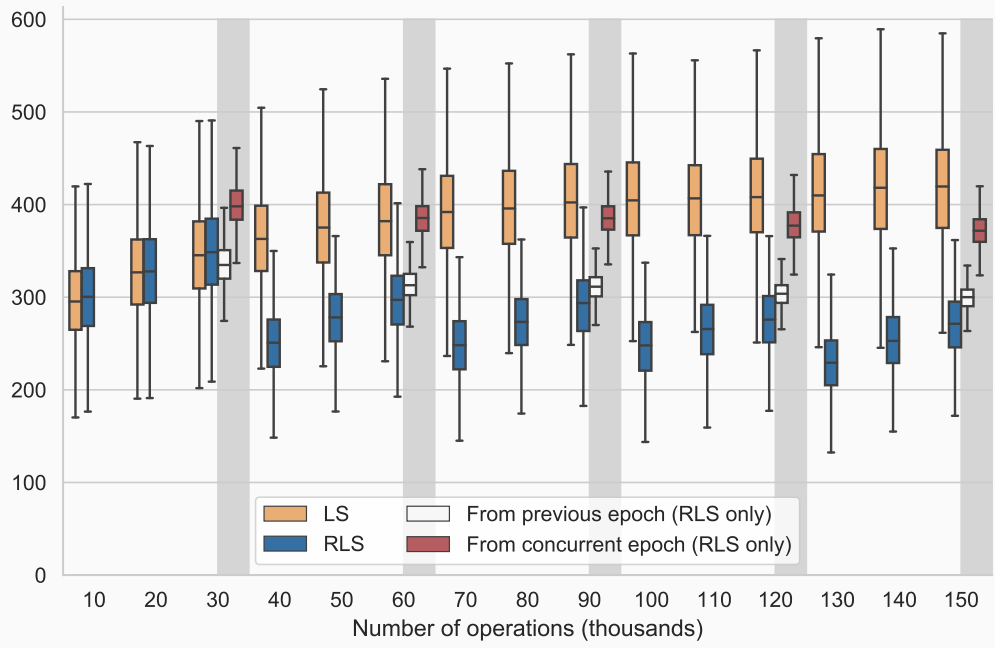
Surcoût en calculs - Opérations *insert* et *remove*

Intuition

Mesurer temps d'intégration local et distant d'opérations *insert* à différents stades de la collaboration



(a) Temps intégration modifs locales



(b) Temps intégration modifs distantes

- Opération *rename* réduit les temps d'intégration
- Réduction du nombre de blocs contrebalance le surcoût des transformations

Conclusion générale & Perspectives

Conclusion

Contributions

- Conception d'un mécanisme de renommage pour CRDTs pour le type Séquence à identifiants densément ordonnés
  - Implémentation et instrumentation de RenamableLogootSplit et de ses dépendances (protocole d'appartenance au réseau, couche de livraison)

Conclusion

Contributions

- Conception d'un mécanisme de renommage pour CRDTs pour le type Séquence à identifiants densément ordonnés
  - Implémentation et instrumentation de RenamableLogootSplit et de ses dépendances (protocole d'appartenance au réseau, couche de livraison)
- Comparaison des différents modèles de synchronisation pour CRDTs...
- ...et des différentes approches pour CRDTs pour le type Séquence

Limites

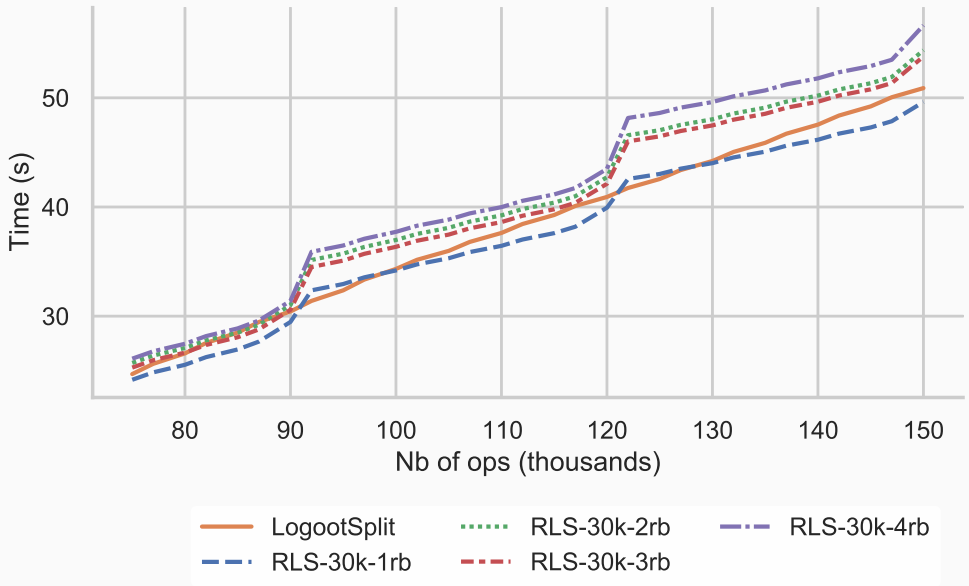
Limites de RenamableLogootSplit

- Surcoût de l'opération *rename*

Limites

Limites de RenamableLogootSplit

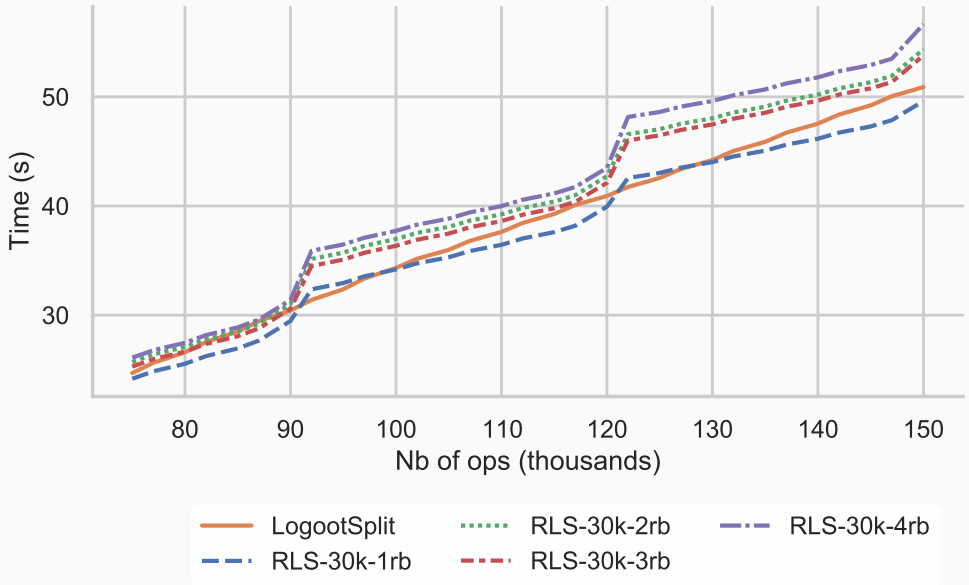
- Surcoût de l'opération *rename*



Limites

Limites de RenamableLogootSplit

- Surcoût de l'opération *rename*

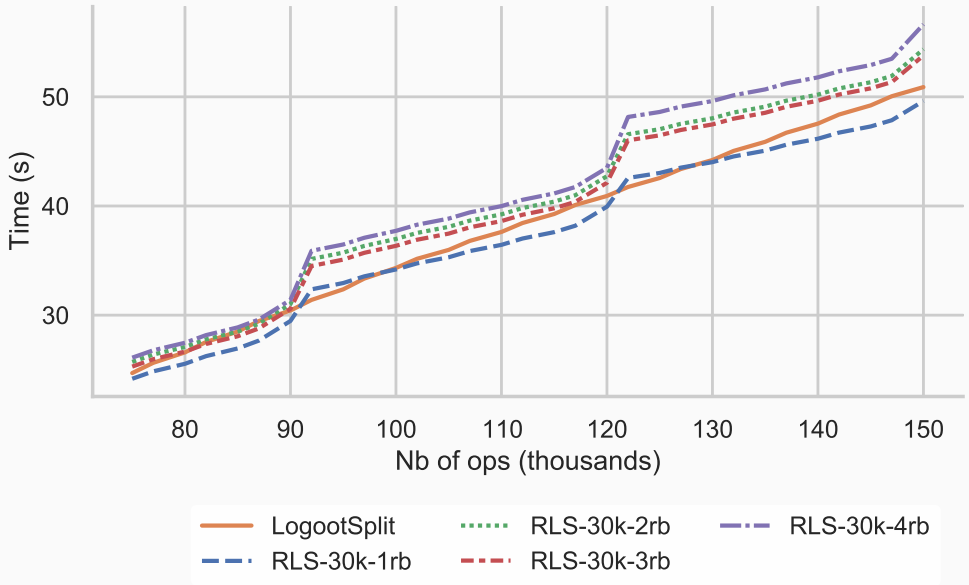


- Évaluations montrent que le temps d'intégration de l'opération *rename* peut atteindre 2s
- A privilégié la correction…
- …doit améliorer les performances (algorithme et implémentation)

Limites

Limites de RenamableLogootSplit

- Surcoût de l'opération *rename*



- Évaluations montrent que le temps d'intégration de l'opération *rename* peut atteindre 2s
- A privilégié la correction…
- …doit améliorer les performances (algorithme et implémentation)

- Stabilité causale requise pour supprimer les métadonnées

Perspectives

Perspectives autour de RenamableLogootSplit

- Comment définir des relations *priority* $<_\varepsilon$ qui minimisent les renommages vains ?
- Peut-on prouver formellement la correction RenamableLogootSplit ?

Perspectives

Perspectives autour de RenamableLogootSplit

- Comment définir des relations *priority* $<_\varepsilon$ qui minimisent les renommages vains ?
- Peut-on prouver formellement la correction RenamableLogootSplit ?

Perspectives autour des CRDTs

- Doit-on encore concevoir CRDTs synchronisés par états ou opérations ?
- Peut-on proposer un framework pour conception de CRDTs synchronisés par opérations ?

Merci de votre attention, avez-vous des questions ?

Publications

- **Article de position** à Middleware 2018 - 19th ACM/IFIP International Middleware Conference (Doctoral Symposium), Dec 2018, Rennes, France.
- **Article d'atelier** avec Gérald Oster et Olivier Perrin à PaPoC 2020 - 7th Workshop on Principles and Practice of Consistency for Distributed Data, Apr 2020, Heraklion / Virtual, Greece.
- **Article de revue** avec Gérald Oster et Olivier Perrin dans IEEE Transactions on Parallel and Distributed Systems, Institute of Electrical and Electronics Engineers, 2022, 33 (12), pp.3870-3885.

Benchmarks

- Node.js, version 13.1.0, avec option `jitless`
- Machiné équipée d'un Intel Xeon CPU E5-1620 (10MB Cache, 3.50 GHz), de 16GB de RAM et utilisant Fedora 31
- Taille des documents obtenus en utilisant notre fork de *object-sizeof*[*]
- Mesures de temps avec `process.hrtime.bigint()`

*. `https://www.npmjs.com/package/object-sizeof`

Doit-on encore concevoir CRDTs synchronisés par états ou opérations ?

	Sync. par états	Sync. par opérations	Sync. par diff. d'états
Forme un sup-demi-treillis	✓	✓	✓
Intègre modifications par fusion d'états	✓	✗	✓
Intègre modifications par élts irréductibles	✗	✓	✓
Résiste nativ. aux défaillances réseau	✓	✗	✓
Adapté pour systèmes temps réel	✗	✓	✓
Offre nativ. modèle de cohérence causale	✓	✗	✗

- Synchronisation par différences offre meilleur des mondes…
- …y a-t-il encore un intérêt aux autres modèles, e.g. pour composition ou sécurité ?