Le contrôle de versions pour tous

Une algèbre de modifications

Pierre-Étienne Meunier (Coturnix)

20 avril 2023

Travail commun avec Florent Becker et la communauté Pijul

Plan

Le contrôle de versions

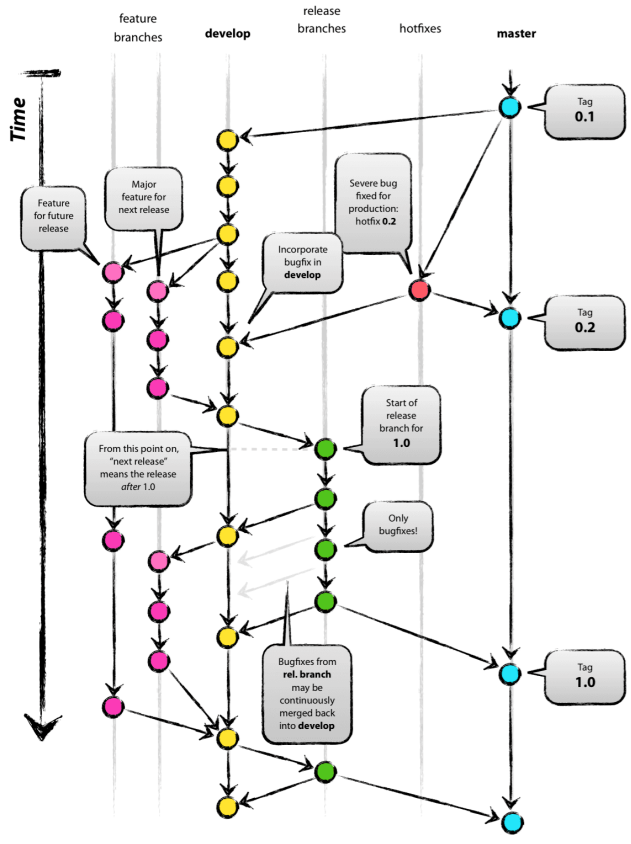
- Un ou plusieurs auteurs collaborent sur un arbre de documents
- Asynchronisme : ils peuvent choisir le moment de synchronisation
- Conflits
- Préservation de l'historique

Un problème réglé ?

Nos outils (Git, SVN…) :

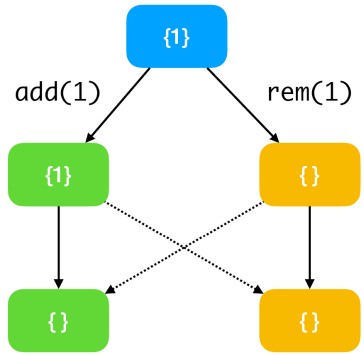
- Ne sont utilisés par aucun non-programmeur
- Sont inutilisables sans serveur central (GitHub)
- Demandent une discipline extrême
- Gaspillent un temps humain conséquent

Les tentatives d'amélioration (Darcs) ne passent pas à l'échelle



MRDTs (ou Git, SVN, CVS...)

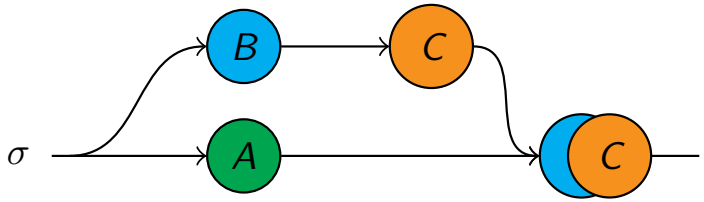
- Exposé de KC Sivaramakrishnan

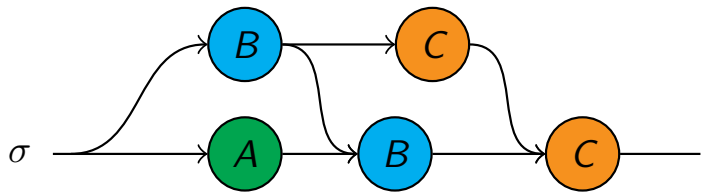


- La fonction de fusion doit satisfaire les bonnes propriétés :
  - **Associativité des changements :** on peut faire des fusions intermédiaires
  - **Commutativité des changements :** tous les changements indépendants commutent
- Complexité algorithmique

MRDTs et Associativité

Possibilité de faire des fusions intermédiaires



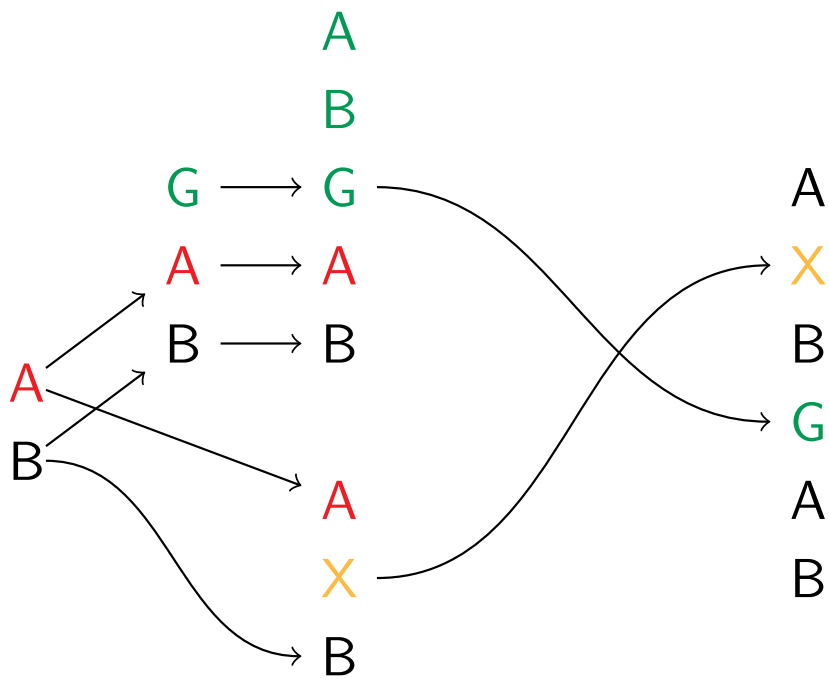


Pour tout x, y d'ancêtre commun σ, soit $\mathsf{m}(\sigma, x, y)$ la fusion entre x et y.

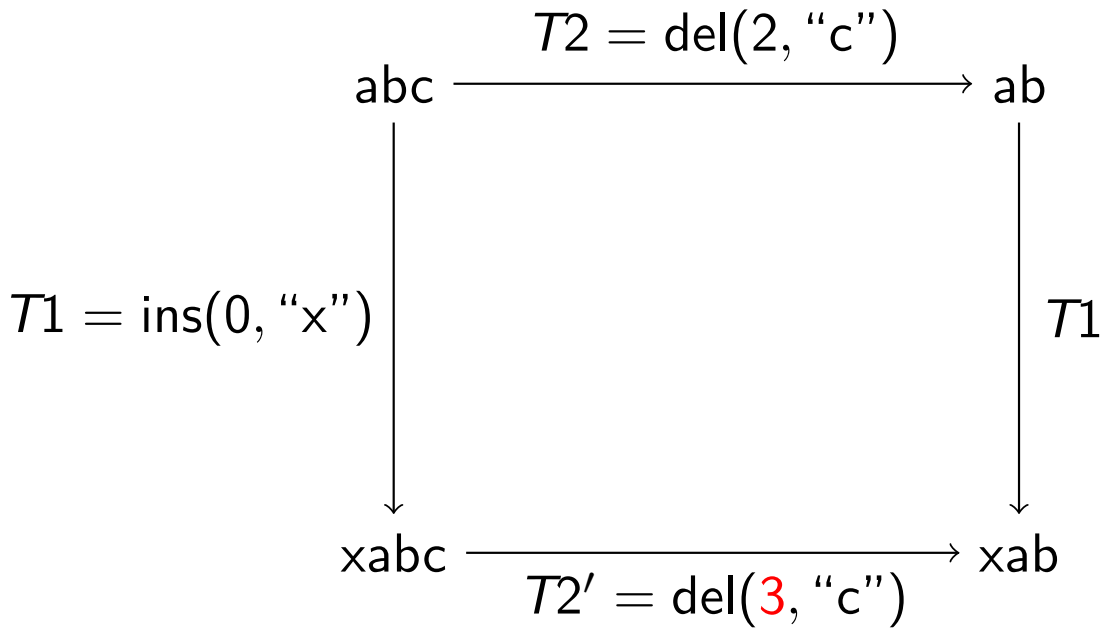
On veut :

$$\mathsf{m}(\sigma, a, c) = \mathsf{m}(b, \mathsf{m}(\sigma, a, b), c)$$

Git, SVN, CVS, Hg... ne sont pas associatifs



Transformées opérationnelles



- Preuves quadratiques en le nombre de types d'opérations.
- Un cauchemar à implémenter

CRDTs

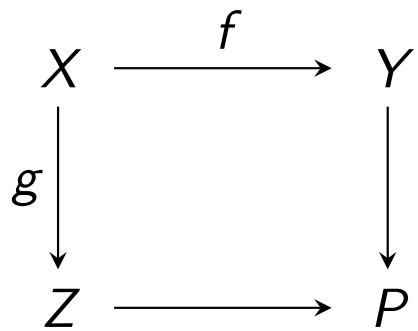
- Principe général : concevoir une structure où toutes les opérations commutent
- Exemples naturels : compteurs (avec incrémentation), ensembles (avec insertion)...
- Plus subtil : tombstones, horloges de Lamport (voir l'exposé de KC Sivaramakrishnan)
- Mauvaise foi : un dépôt Git complet (pas juste `HEAD`)

Les conflits

- Là où un bon outil est le plus nécessaire
- La définition dépend du contexte et de l'outil
- **Exemple :** Alice et Bob écrivent au même endroit d'un fichier
- **Exemple :** Alice renomme un fichier de f à g pendant que Bob renomme f en h
- **Exemple :** Alice renomme une fonction f pendant que Bob ajoute un appel à f

Une solution catégorique

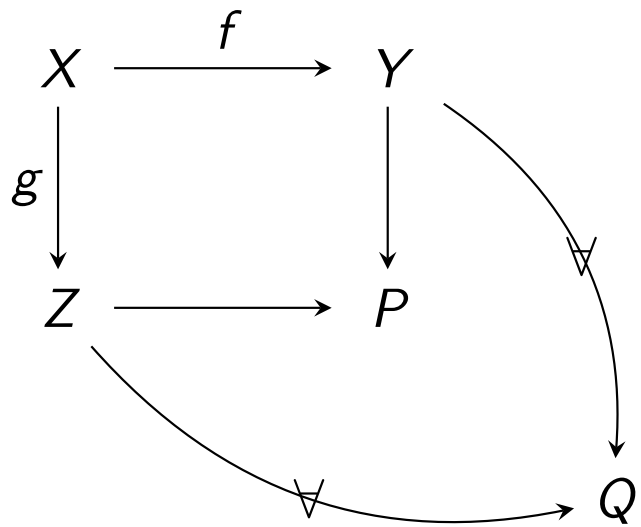
Pour deux patchs f et g, on cherche un unique état P tel que :



Réflexion démarrée par Samuel Mimram et Cinzia Di Giusto

Une solution catégorique

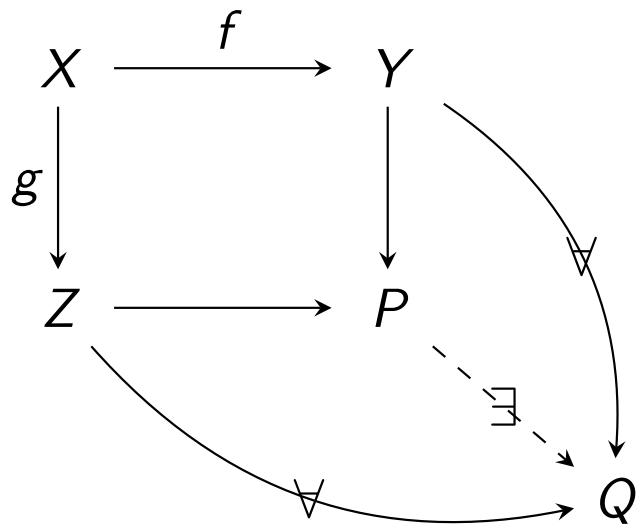
Pour deux patchs f et g, on cherche un unique état P tel que :
Pour tout état Q accessible par Alice et Bob après f et g



Réflexion démarrée par Samuel Mimram et Cinzia Di Giusto

Une solution catégorique

Pour deux patchs f et g, on cherche un unique état P tel que :
Pour tout état Q accessible par Alice et Bob après f et g
Il existe un chemin de P à Q.

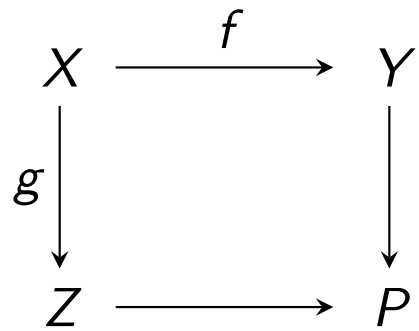


Si P existe (et est unique), on l'appelle le *pushout* de f et g.

Réflexion démarrée par Samuel Mimram et Cinzia Di Giusto

Problème : le pushout n'existe pas toujours

- Équivalent à dire que parfois, il y a des conflits.
- Comment généraliser la représentation des états (X, Y, Z) pour que toutes les paires (f et g) aient un pushout ?



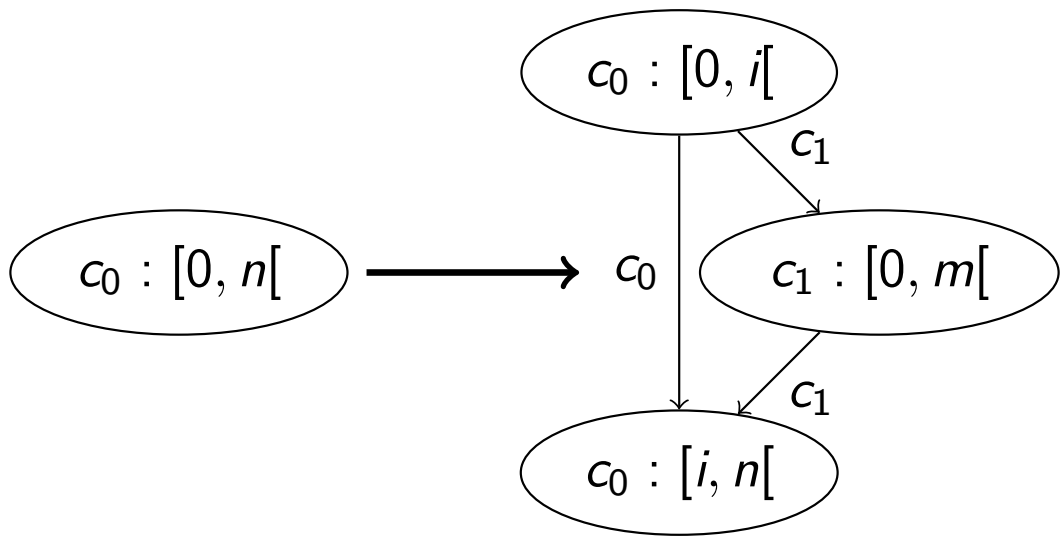
Solution : Les états sont des graphes orientés, où :

- Les sommets sont des octets (ou intervalles d'octets).
- Les arêtes représentent l'union de tous les ordres connus entre les sommets.

Ajouter une ligne

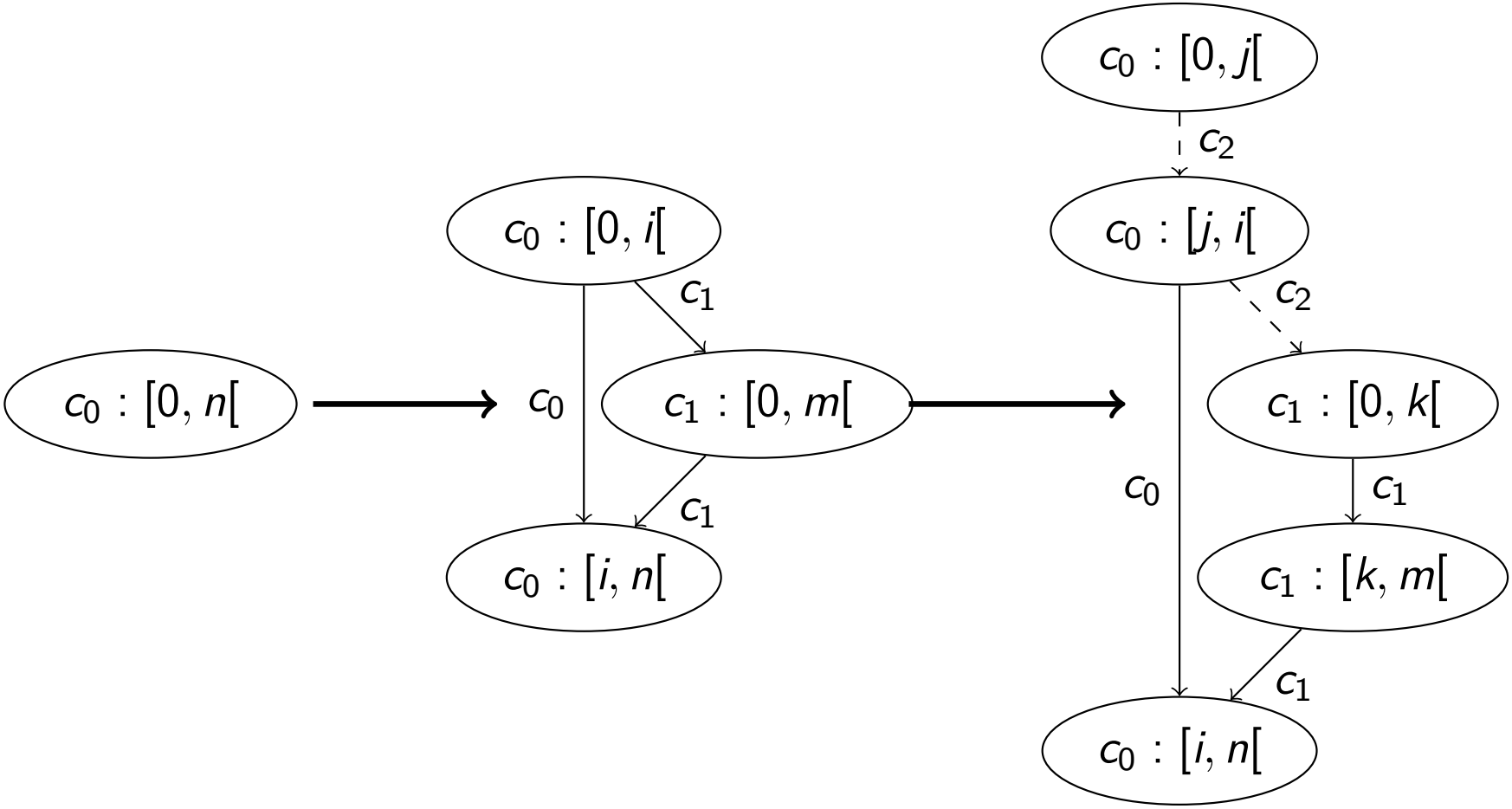
- Les sommets sont étiquetés par un numéro de patch c_0 et un intervalle (par exemple $[0, n[$) dans ce patch.
- Les arêtes sont étiquetées par le patch qui les a introduites.

Ici c_1 ajoute m octets entre les positions $i-1$ et i de c_0 :



Supprimer une ligne

On supprime les octets de j à i dans c_0, et de 0 à k dans c_1 :



Remarque : on doit ajouter un bit dans les étiquettes des arêtes.

C'est (presque) tout !

Deux types de changement :

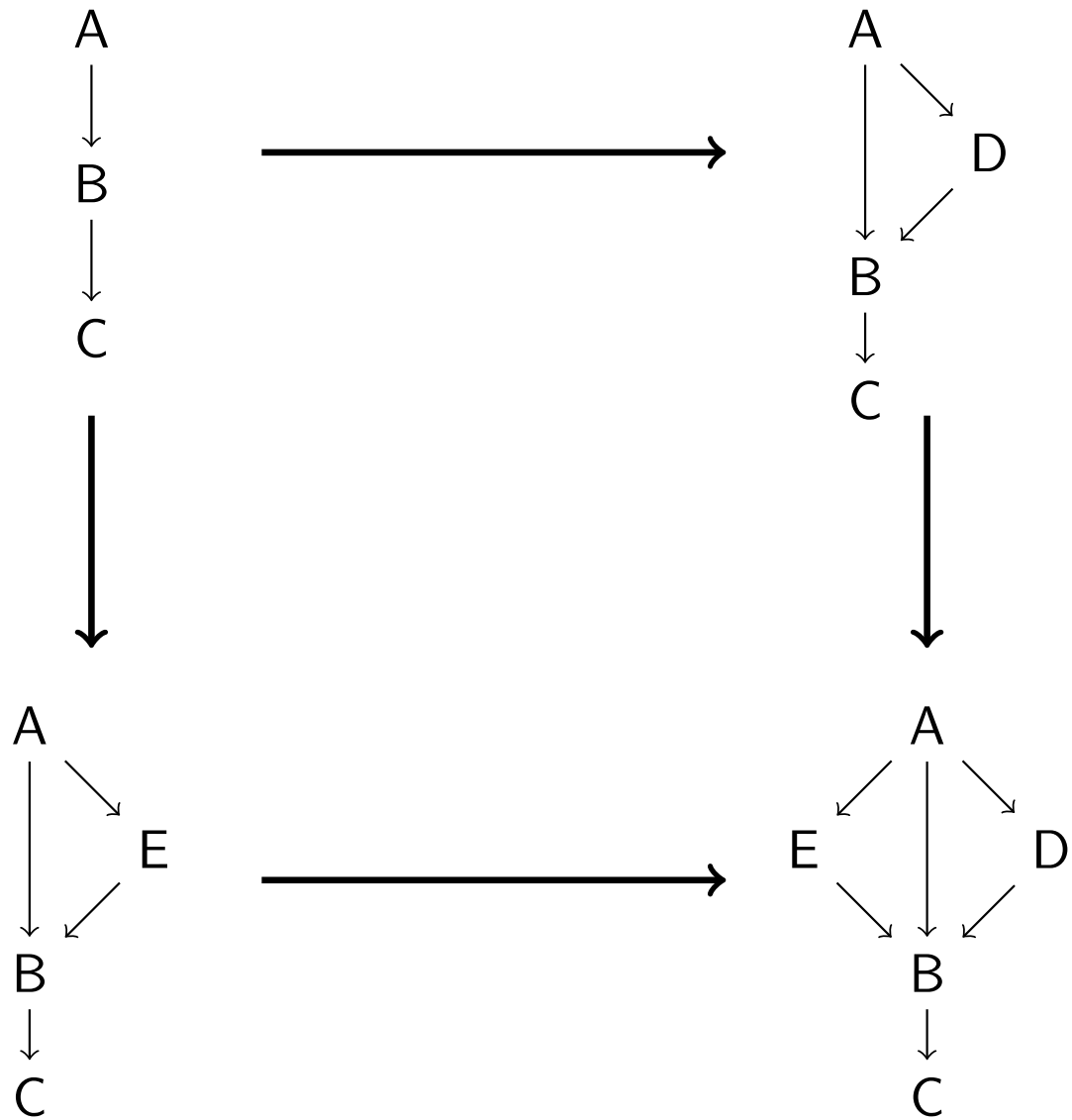
- Ajouter un sommet (un intervalle d'octets), dans un **contexte**
- Changer l'étiquette d'une arête

Les conflits

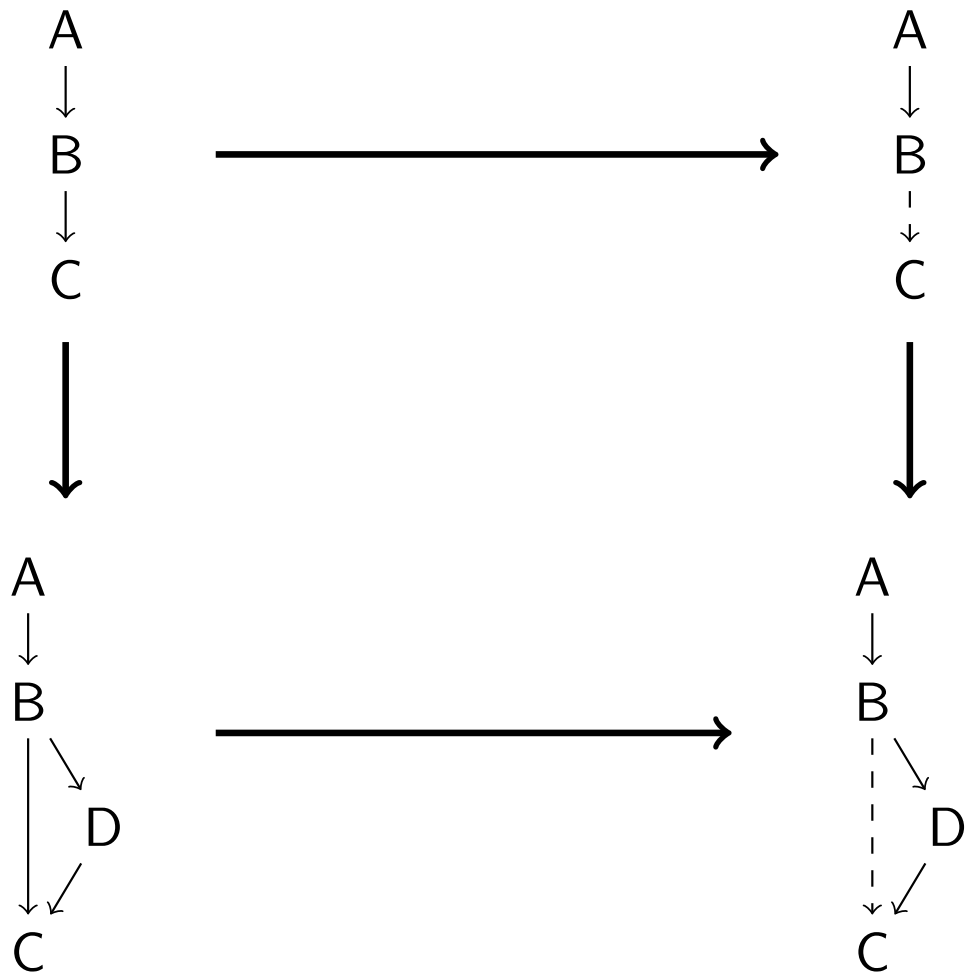
- Les sommets **vivants** sont ceux dont toutes les arêtes entrantes sont vivantes
- Les sommets **morts** sont ceux dont toutes les arêtes entrantes sont mortes
- Les autres sont des **zombies**.

Un graphe est **sans conflit** ssi il n'a aucun zombie
et ses sommets vivants sont totalement ordonnés.

Exemple : un conflit d'ordre



Exemple : suppression de contexte descendant



Plan

Définition du problème

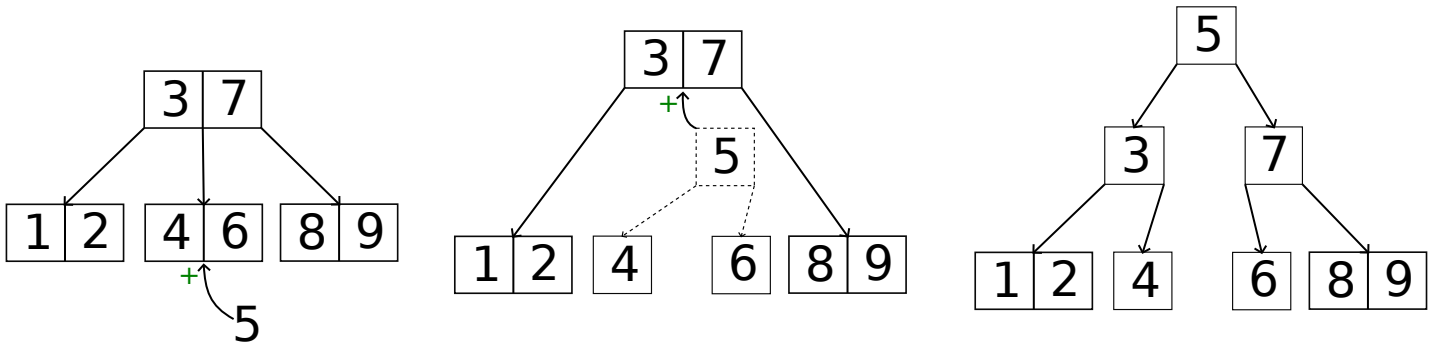
- Comment stocker ce graphe sur le disque ?
- Dictionnaire clef-valeur
- Éviter la corruption : transactions validées atomiquement
  L'écriture d'**un seul secteur (4ko)** valide une transaction entière.
- Clone sans copie

Allouer dans un fichier, naïvement

- Restriction de l'atomicité : on alloue par multiples de 4ko
- Version naïve : à chaque fois qu'on a besoin d'un bloc, on étend le fichier.

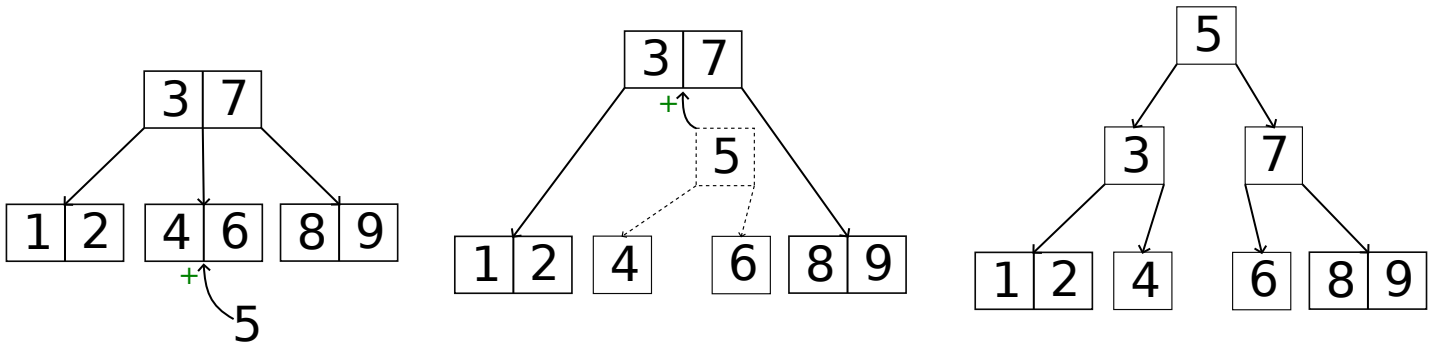
Une structure particulière : les B trees

Les nœuds ont une **capacité** (ici 2), sont toujours $\geq$ moitié remplis
Insertion par les feuilles :

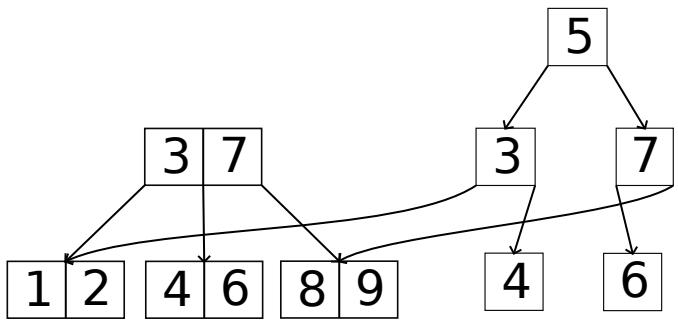


Une structure particulière : les B trees

Les nœuds ont une **capacité** (ici 2), sont toujours $\geq$ moitié remplis
Insertion par les feuilles :

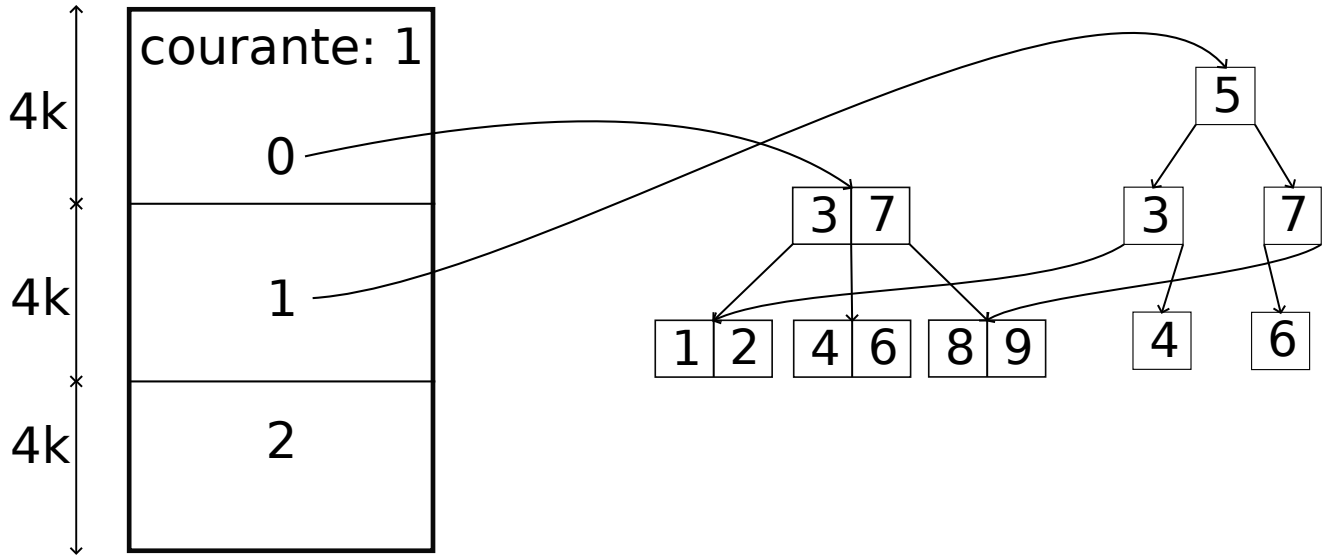


Dans Sanakirja on ne modifie pas l'état provenant d'une transaction précédente :



Mais ça ne coûte pas plus cher, puisqu'on doit déjà copier depuis le disque quand c'est le cas !

Atomicité



- La bascule se fait quand on met à jour le premier octet du fichier.
- Chaque nouvelle transaction remplace la plus ancienne.
- Les transactions en écriture ne bloquent pas la lecture :
  Un lecteur peut survivre à au plus $n - 1$ transactions en écriture.

Un allocateur plus malin

Distinguer deux types de blocs libres :

- Blocs allouées dans la transaction en cours, puis libérées
- Blocs libérées par la transaction en cours

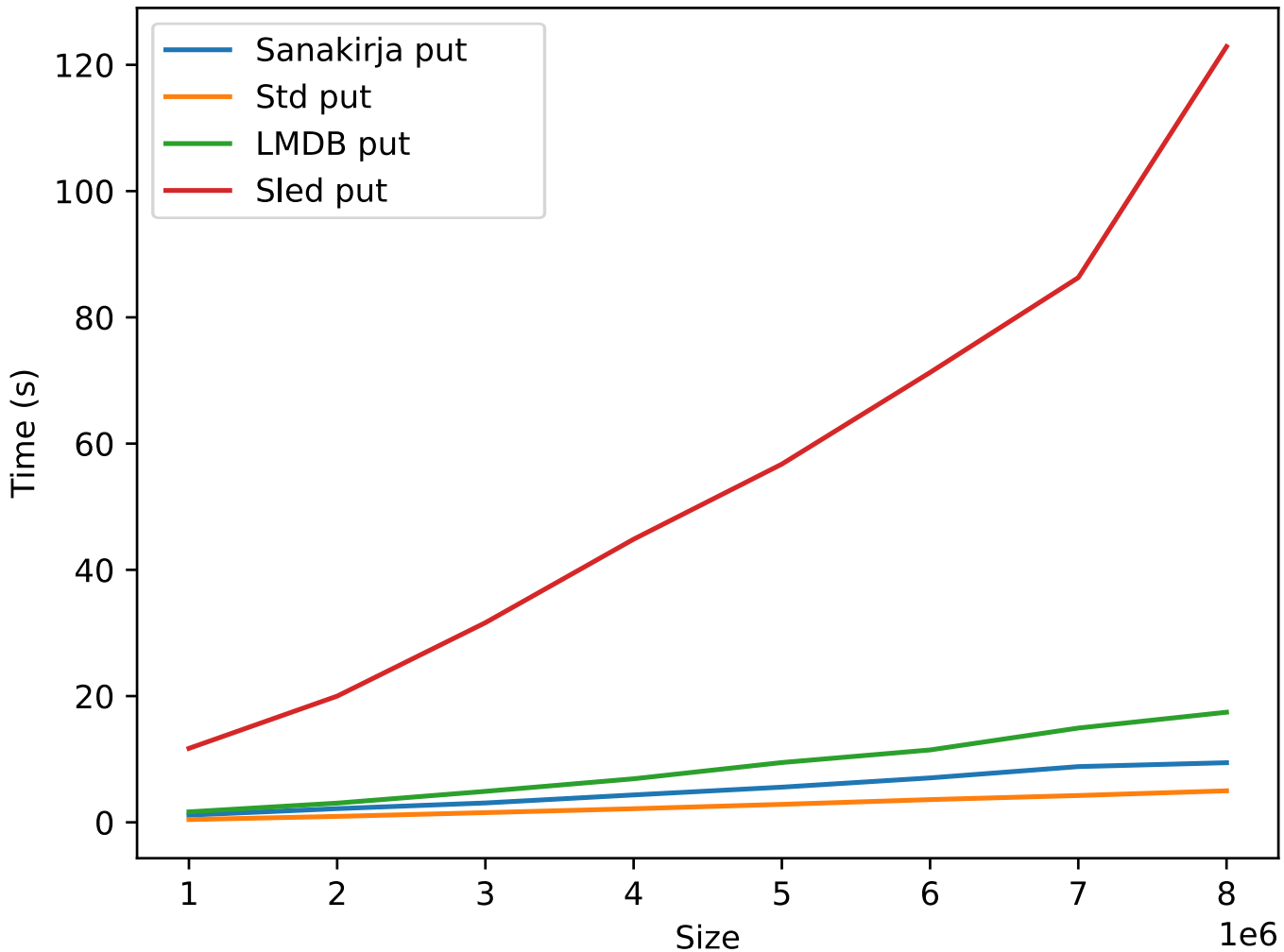
On stocke les blocs libres dans un B tree (qui utilise l'allocateur naïf)

Comptage de références (cloner sans copier)

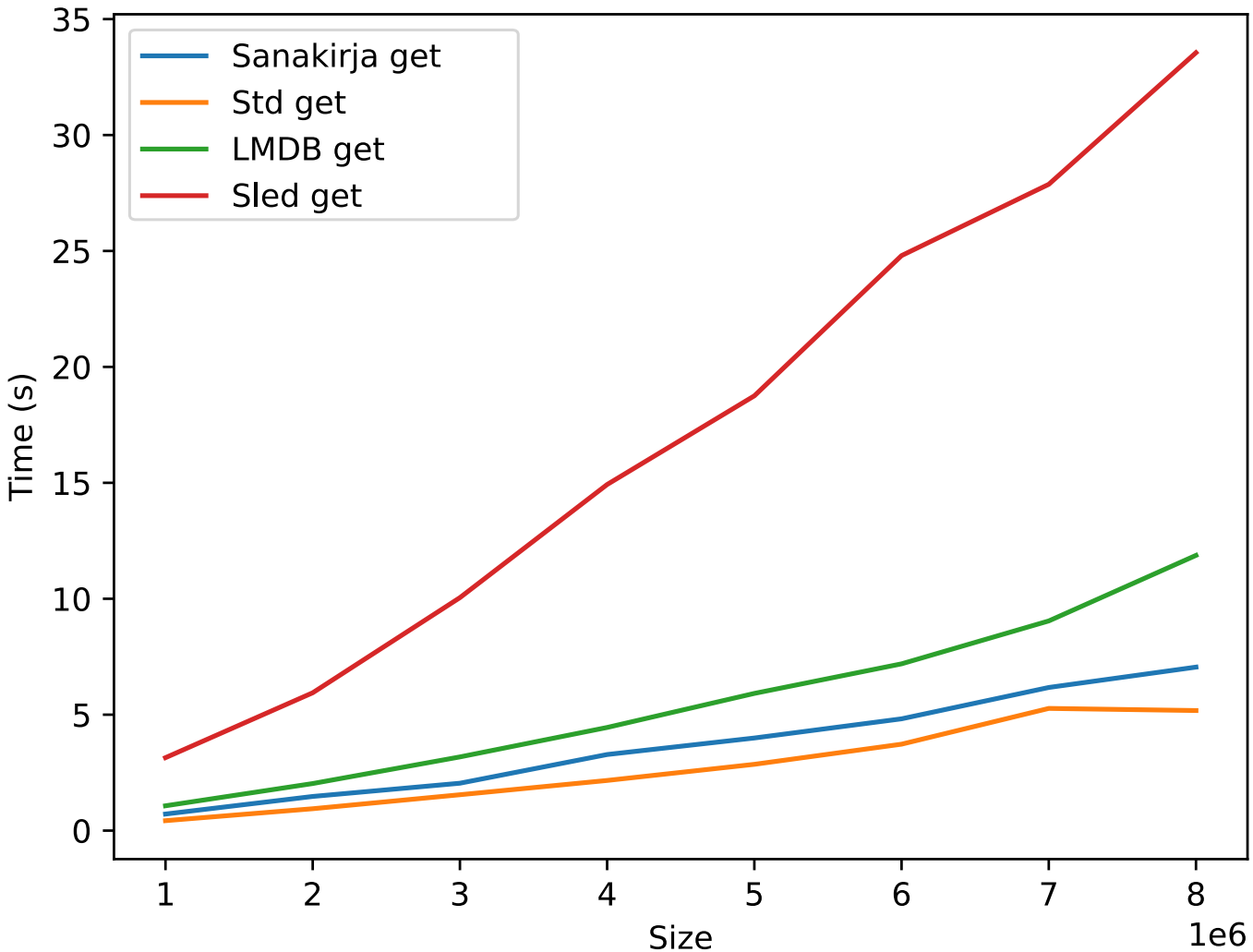
- Complication : le comptage doit être lui aussi atomique, on ne peut donc pas l'écrire sur les blocs comptées
- Un B tree associe l'adresse du bloc à son compte :

  Pour cloner, il suffit de changer cette valeur !
- Propagation récursive des comptes quand on modifie un arbre
- Pour éviter les cycles, seuls les blocs de compte ≥ 2 sont comptées
- Remarque : même si les transactions passées peuvent référencer des blocs encore actifs, mais le comptage ne concerne que les blocs clonés **au sein d'une seule transaction**.

Performance



Performance



Une bibliothèque extrêmement générique

- Généricité des clefs, des valeurs, de la structure interne des blocs
- Autres structures de données (cordes)
- On peut changer l'allocateur de blocs
- Et la façon de garder les blocs :
  - fichier mapés en mémoire
  - tas + `write`, pour contrôler l'utilisation mémoire
  - plateformes sans appels systèmes standards

Plan

Projet en cours : plate-forme d'hébergement

- Prototype en fonctionnement depuis 2016, avec soubresauts
- Incendie d'OVH en mars 2021
- Aujourd'hui dans 3 datacenters (Canada, France, Singapour)
- De très nombreux déboires de réplication de bases de données

Projet en cours : plate-forme d'hébergement

- Prototype en fonctionnement depuis 2016, avec soubresauts
- Incendie d'OVH en mars 2021
- Aujourd'hui dans 3 datacenters (Canada, France, Singapour)
- De très nombreux déboires de réplication de bases de données

Nouvelle version entièrement Serverless (FaaS), open source

Sortie prévue très bientôt

Problèmes ouverts

- Prouver Sanakirja
- Extension à d'autres catégories que des monoïdes
- Diff avec découpage syntaxique
- Archivage des vieux morceaux du graphe
- Contrôle de versions accessible à tous (avocats, artistes, associations, parlementaires...)

Merci pour votre attention