PLP

Memory

31 January 2026

Table des matières

1 Memory management

Les différents langages de programmation utilisent diverses techniques pour gérer la mémoire. Par exemple, en `C`, nous avons un contrôle direct sur l'allocation et la libération de mémoire via `malloc` et `free`. En revanche, des langages comme `Java` ou `Python` utilisent la collecte de déchets (garbage collection) pour automatiser la gestion de la mémoire.

1.1 Memory life cycle

Le cycle de vie de la mémoire comprend plusieurs étapes clés :

1. **Allocation**: La mémoire est allouée pour stocker des données.
2. **Utilisation**: Utilisation de la mémoire allouée pour les opérations de lecture et d'écriture.
3. **Libération**: La mémoire n'est plus nécessaire et doit être libérée pour éviter les fuites de mémoire.

La deuxième étape est explicite dans tous les langages, tandis que l'allocation et la libération peuvent être gérées automatiquement ou manuellement selon le langage.

1.2 Memory manager

Le gestionnaire de mémoire est responsable de l'allocation et de la libération de la mémoire heap. Il maintient des structures de données pour suivre les blocs de mémoire alloués et libres, et optimise l'utilisation de la mémoire pour minimiser la fragmentation. La désallocation mémoire peut-être soit explicite (comme en `C` avec `free`), soit implicite via des techniques de garbage collection.

1.2.1 Explicite

Dans les langages avec gestion explicite de la mémoire, le programmeur est responsable de l'allocation et de la libération de la mémoire. Cela offre un contrôle précis mais peut entraîner des erreurs telles que des fuites de mémoire ou des accès à des zones mémoire libérées. Nous avons souvent les erreurs suivantes:

1. **Dangling pointer**: Un pointeur qui référence une zone mémoire déjà libérée.
2. **Memory leak**: La mémoire allouée n'est jamais libérée, ce qui peut épuiser les ressources mémoire.
3. **Double free**: Tenter de libérer une zone mémoire déjà libérée.

1.2.2 Implicite

Dans les langages avec gestion implicite de la mémoire, le système de runtime gère automatiquement l'allocation et la libération de la mémoire. Cela réduit le risque d'erreurs liées à la mémoire, mais peut introduire des pauses dans l'exécution du programme lors de la collecte de déchets.

On suit la définition suivante:

Si un block mémoire est atteignable, alors il est vivant; sinon, il n'est plus utilisable et peut être récupéré par le garbage collector.

2 Garbage collection

La collecte de déchets (garbage collection) est une technique de gestion automatique de la mémoire qui identifie et libère la mémoire qui n'est plus utilisée par le programme. Cela permet de prévenir les fuites de mémoire et d'améliorer la sécurité de la mémoire.

Il existe 3 techniques principales de garbage collection:

1. **Reference counting**: Chaque objet maintient un compteur de références. Lorsque le compteur atteint zéro, l'objet est libéré.
2. **Marking and sweeping**: Le garbage collector marque les objets accessibles et balaie la mémoire pour libérer les objets non marqués.
3. **Copying garbage collection**: La mémoire est divisée en deux zones. Les objets vivants sont copiés d'une zone à l'autre, et la zone source est libérée en bloc.

2.1 Reachable objects

On parle d'un objet atteignable s'il peut être accédé directement ou indirectement à partir des racines (comme les variables globales, les piles d'exécution, etc.). Si un objet n'est plus atteignable, il est considéré comme inutilisable et peut être récupéré par le garbage collector.

3 GC Data structures

3.1 Free list

Une liste libre est une structure de données utilisée par le gestionnaire de mémoire pour suivre les blocs de mémoire disponibles pour l'allocation. Chaque bloc libre contient un pointeur vers le prochain bloc libre, formant ainsi une liste chaînée.

Pour libérer un bloc de mémoire, il est ajouté à la liste libre. Lorsqu'une allocation est demandée, le gestionnaire de mémoire parcourt la liste libre pour trouver un bloc suffisamment grand.

3.2 Block header

Un en-tête de bloc est une structure de données placée au début de chaque bloc de mémoire. Il contient des informations sur le bloc, telles que sa taille, son état (alloué ou libre), et parfois des pointeurs vers les blocs adjacents.

3.3 BiBOP

BiBOP (Big Bag of Pages) est une technique de gestion de la mémoire qui divise la mémoire en pages, chaque page étant dédiée à un type spécifique d'objets. Cela permet d'optimiser l'allocation et la collecte de déchets en regroupant les objets similaires.

3.4 Fragmentation

La fragmentation de la mémoire se produit lorsque l'espace mémoire disponible est divisé en petits blocs non contigus, rendant difficile l'allocation de grands blocs de mémoire. Il existe deux types de fragmentation:

1. **Fragmentation externe**: Se produit lorsque des blocs libres sont dispersés dans la mémoire.
2. **Fragmentation interne**: Se produit lorsque des blocs alloués sont plus grands que nécessaire, laissant de l'espace inutilisé à l'intérieur des blocs.

4 Reference counting

L'approche de comptage de références consiste à maintenir un compteur pour chaque objet, indiquant le nombre de références pointant vers cet objet. Lorsque le compteur atteint zéro, l'objet est libéré.

Si l'objet est référencé alors on incrémente le compteur, sinon on le décrémente. Cette technique est simple à implémenter mais ne gère pas les cycles de références, où deux objets se référencent mutuellement.

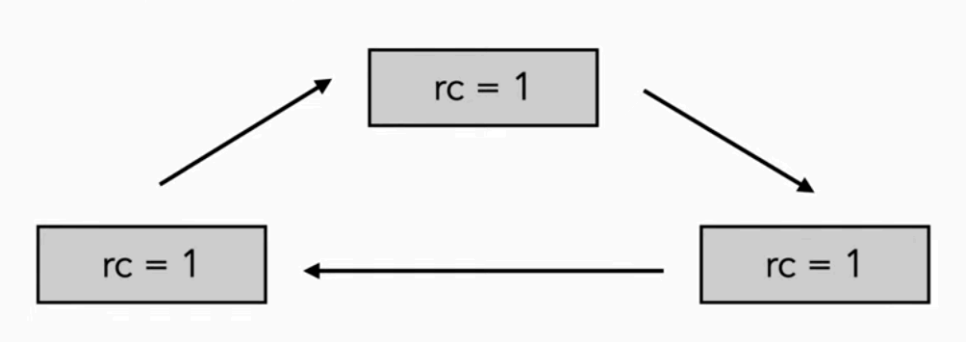

Fig. 2. – Capture des slides du cours – Reference Counting Diagram

Le comptage de référence, en raison de son incapacité à gérer les cycles de références, est souvent combiné avec d'autres techniques de collecte de déchets pour assurer une gestion complète de la mémoire.

4.1 Smart pointers

Les pointeurs intelligents sont des abstractions de pointeurs qui gèrent automatiquement la durée de vie des objets en utilisant des techniques comme le comptage de références. Ils aident à prévenir les fuites de mémoire et les erreurs liées à la gestion manuelle de la mémoire.

5 Copying GC

La technique de copie pour la collecte de déchets divise la mémoire en deux zones. Lors de la collecte, les objets vivants sont copiés d'une zone à l'autre, et la zone source est libérée en bloc. Cela réduit la fragmentation et améliore les performances d'allocation.

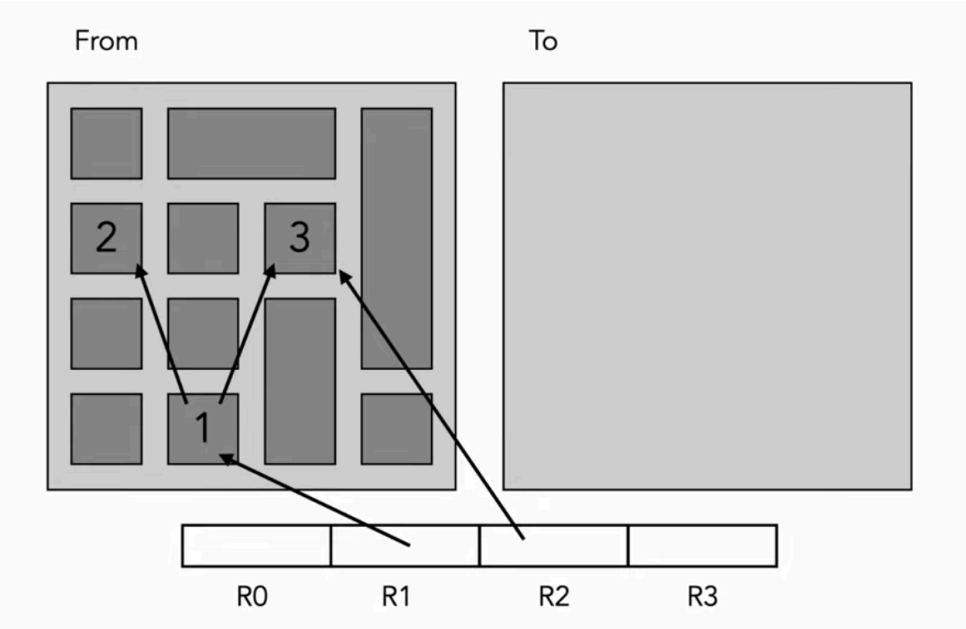

Fig. 3. – Capture des slides du cours – Copying Garbage Collection Diagram

1. On parcourt chaque racine pour trouver les objets atteignables.
2. On copie les objets atteignables dans la zone de destination.
3. On met à jour les références pour pointer vers les nouvelles adresses.

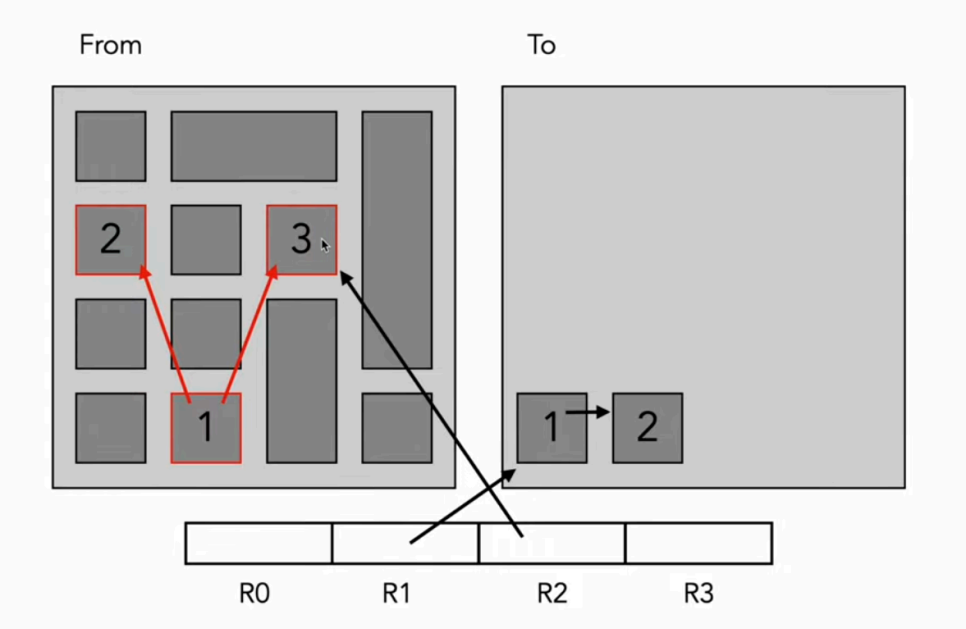

Fig. 4. – Capture des slides du cours – Before and After Copying GC

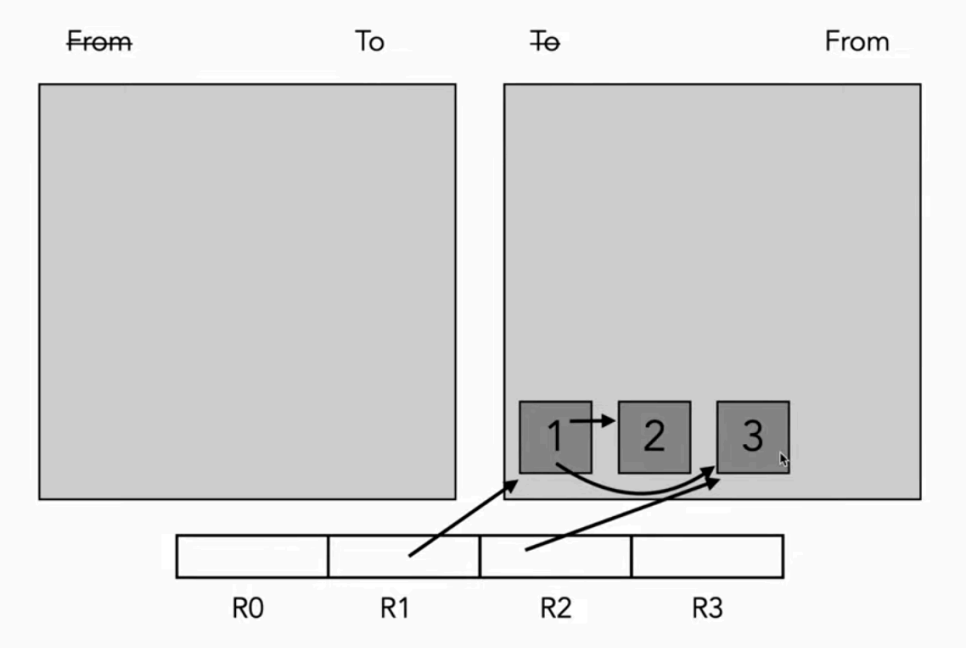

Fig. 5. – Capture des slides du cours – Copying GC Process

Un des gros avantages de cette technique est qu'elle compresse la mémoire en éliminant la fragmentation, ce qui améliore l'efficacité de l'utilisation de la mémoire.

Le gros inconvénient est que la mémoire utilisée est le double de la mémoire réellement nécessaire, car une moitié est toujours inutilisée pendant la collecte.

5.1 Forwarding pointers

Les pointeurs de redirection sont utilisés dans les techniques de collecte de déchets, comme la copie, pour maintenir la référence aux objets déplacés. Lorsqu'un objet est copié vers une nouvelle adresse, un pointeur de redirection est créé à l'ancienne adresse, pointant vers la nouvelle adresse. Cela permet de mettre à jour toutes les références vers l'objet déplacé sans avoir à parcourir toute la mémoire.

5.2 Cheney's copying GC

La variante de la collecte de déchets par copie de Cheney est une approche élégante qui ne nécesessite un passage en largeur sur les objets atteignables. Elle ne nécessite qu'un seul pointeur comme état, contrairement aux autres méthodes qui en nécessitent plusieurs.

6 Mark and sweep GC

La technique de marquage et balayage pour la collecte de déchets fonctionne en deux phases. Dans la phase de marquage, le garbage collector parcourt tous les objets atteignables à partir des racines et les marque comme vivants. Dans la phase de balayage, il parcourt toute la mémoire et libère les objets non marqués.

Cet algorithme est donc en deux phases:

1. **Mark**: Parcourir les objets atteignables et les marquent comme vivants.
2. **Sweep**: Parcourir toute la mémoire et libérer les objets non marqués.

Une troisième phase dite **compactage** peut être ajoutée pour réduire la fragmentation en déplaçant les objets vivants vers une zone contiguë de la mémoire.

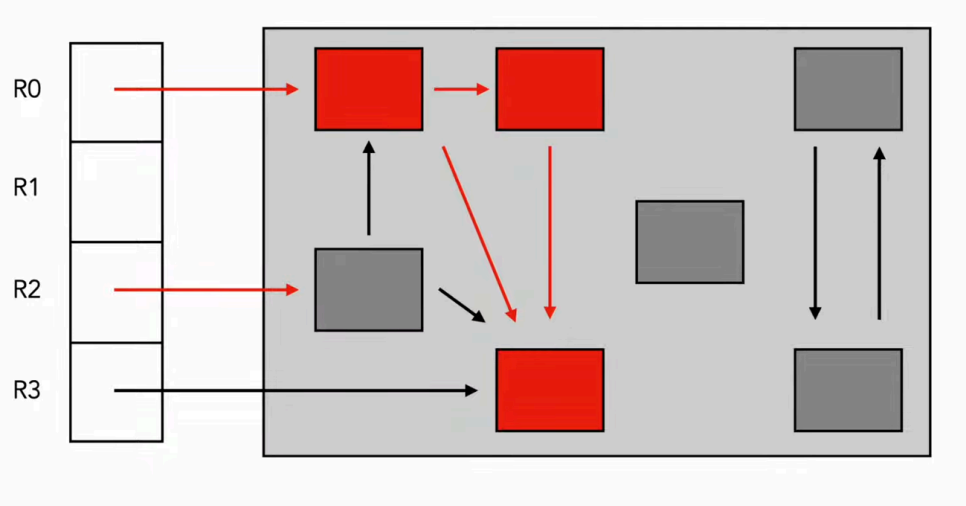

Fig. 6. – Capture des slides du cours – Mark and Sweep GC Diagram

Le principal avantage de cette technique est qu'elle peut gérer les cycles de références, contrairement au comptage de références. Cependant, elle peut introduire des pauses dans l'exécution du programme pendant la collecte de déchets, ce qui peut affecter les performances.

6.1 Marking objects

Etant donné que marquer un objet ne nécessite que d'ajouter un bit supplémentaire. Cependant, si le système garanti que tous les objets sont alignés sur des adresses paires, on peut réutiliser le bit de poids faible de l'adresse pour marquer l'objet sans avoir besoin d'espace supplémentaire.

Il est aussi possible d'utiliser une table de bits séparée pour suivre l'état de marquage des objets, mais cela nécessite un accès supplémentaire à la mémoire lors du marquage et du balayage.

6.1.1 Allocation policy

Lors de l'allocation de mémoire, le gestionnaire de mémoire peut utiliser différentes politiques pour choisir le bloc libre à allouer. Les politiques courantes incluent:

1. **First fit**: Allouer le premier bloc libre suffisamment grand.
2. **Best fit**: Allouer le plus petit bloc libre qui convient.

Si le bloc assigné est plus grand que nécessaire, il peut être divisé en deux blocs: un pour l'allocation et un autre pour le reste libre.

Lors-ce que la mémoire est libérée, le gestionnaire de mémoire peut fusionner les blocs libres adjacents pour réduire la fragmentation.