

Java Enterprise Edition

Chapitre 2: Machine Virtuelle Java – JVM

Sébastien Chèvre, 2019

Plan du module – chapitres Java Entreprise Edition - JEE

- Chapitre 01 : Introduction et motivations
- **Chapitre 02 : Machine Virtuelle Java**
- Chapitre 03 : Java EE
- Chapitre 04 : *Spring Boot & Spring MVC*
- Chapitre 05 : *ORM & DAO avec Spring Data & JPA*
- Chapitre 06 : Sécurité avec *Spring Security*
- Chapitre 07 : Tests avec *Spring Testing*
- Chapitre 08 : SOA & Micro-services avec *Spring*
- Chapitre 09 : *JMS (Java Messaging Service)*

- Projet

Objetifs du chapitre

Chapitre 02 : Machine Virtuelle Java - JVM

A la fin de ce chapitre, les étudiants peuvent:

- Comprendre les concepts de base de la JVM
- Mettre en pratique quelques concepts JVM (Garbage Collector, Heap)
- Connaitre les concepts de base d'un serveur d'application JEE
- Suivre et comprendre le comportement d'une application Java (mémoire, performances, garbage collector)

Plan du chapitre 2

Chapitre 02 : Machine Virtuelle Java - JVM

- **Machine Virtuelle Java**
 - Rappel – typologie de langages
 - Concepts de bases d'un JVM
 - Garbage Collector
 - Serveur d'applications JEE

JEE

Sémantique des diapositives



À savoir théoriquement (TE)



Sensibilisation, illustration de concepts, exemples



Aspects pratiques

Rappel

RAPPEL

Langages compilés vs interprétés

Langages interprétés



■ Langage interprétés

- La traduction se fait en **temps réel, lors de l'exécution**
- l'exécution du programme (script) nécessite la présence d'un **interpréteur**
- Un des avantages est **qu'un même script peut être exécuté sur plusieurs plateformes différentes**
- En revanche la traduction (interprétation) du code à chaque exécution a **un impact sur les performances**



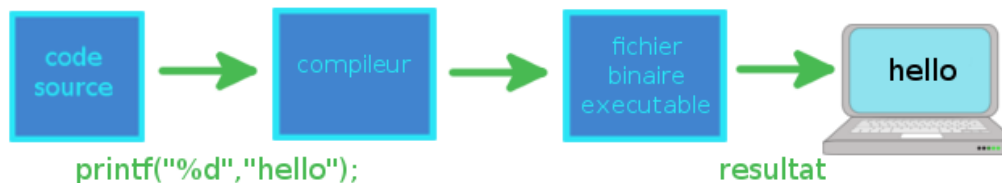
Langages compilés vs interprétés

Langages compilés



■ Langage compilés

- Le code source est tout d'abord **compilé**
- Par un logiciel qu'on appelle **compilateur**,
- En un **code binaire** qu'un humain ne peut pas lire mais qui est très facile à lire pour un ordinateur.
- Le système d'exploitation **va utiliser le code binaire et les données d'entrée pour calculer les données de sortie**



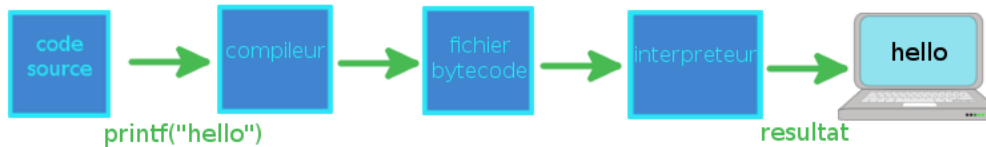
Langages compilés vs interprétés

Langages hybrides



■ Langage hybride ou semi interprété

- Traité par un *interpréteur*
- Le code est préalablement *traduit dans un langage intermédiaire (bytecode)* proche du langage machine, permettant ainsi de préserver *de bonnes performances*



■ La JVM est l'interpréteur du langage Java

Langages compilés vs interprétés

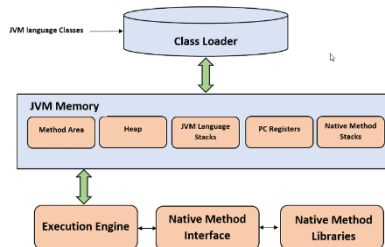
Langages JVM



- La JVM ne supporte pas uniquement Java



Concepts de base d'une JVM



Concepts de base d'une JVM

Qu'est-ce qu'une JVM

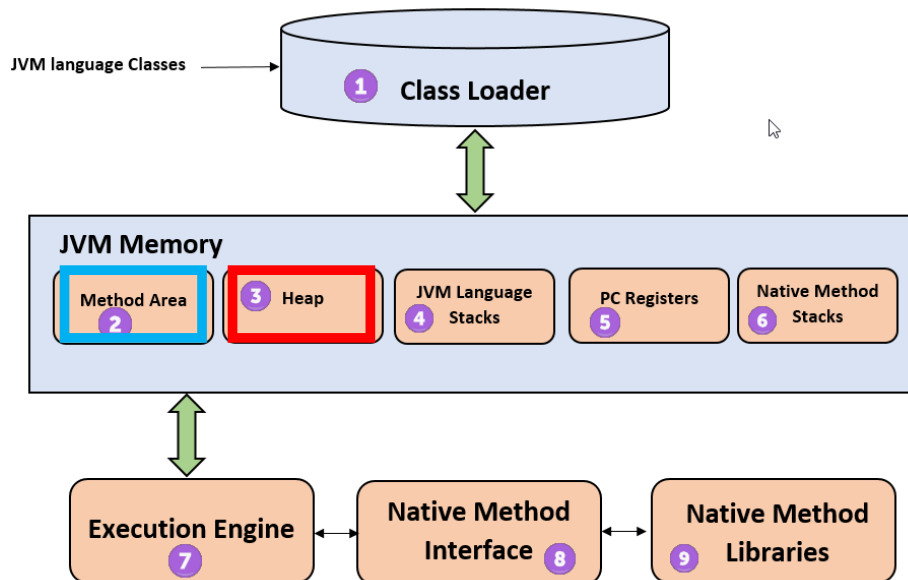


- **JVM : Machine virtuelle destinée à exécuter des programmes Java**
- **Une machine virtuelle (VM) est une architecture abstraite**
- **Elle fait abstraction du système d'exploitation**
- **Permet d'écrire des programmes Java portables sur différents systèmes d'exploitation**

- **Spécifications élaborées par Sun**
- **Différentes implémentations de JVM existent**
 - Sun/Oracle Hotspot
 - J9
 - Jrocket
 - IBM

Concepts de base d'une JVM

Architecture de la JVM



1. Classloader

Chargement des fichiers .clas

2. Method Area

Stockage de la structure des classes, méthodes

3. Heap

Objets, instances et tableaux. Zone commune et partagée

4. JVM Language Stacks

Une par thread. Variable locales et résultats partiels

5. PC Registers

Registre de l'exécution courante

6. Native Method Stacks

Code natif autre que Java

7. Execution Engine

Moteur d'exécution

8. Native Method Interface

Permet d'appeler des librairies et méthodes natives

9. Native Method Libraries

Collection de librairies natives requises par le moteur d'exécution



Concepts de base d'une JVM

Structure de la mémoire



- La mémoire de la JVM est divisée en plusieurs parties:
 - Mémoire **Heap**
 - Mémoire **Non-heap**
 - **Autre** mémoire



Concepts de base d'une JVM

Mémoire HEAP



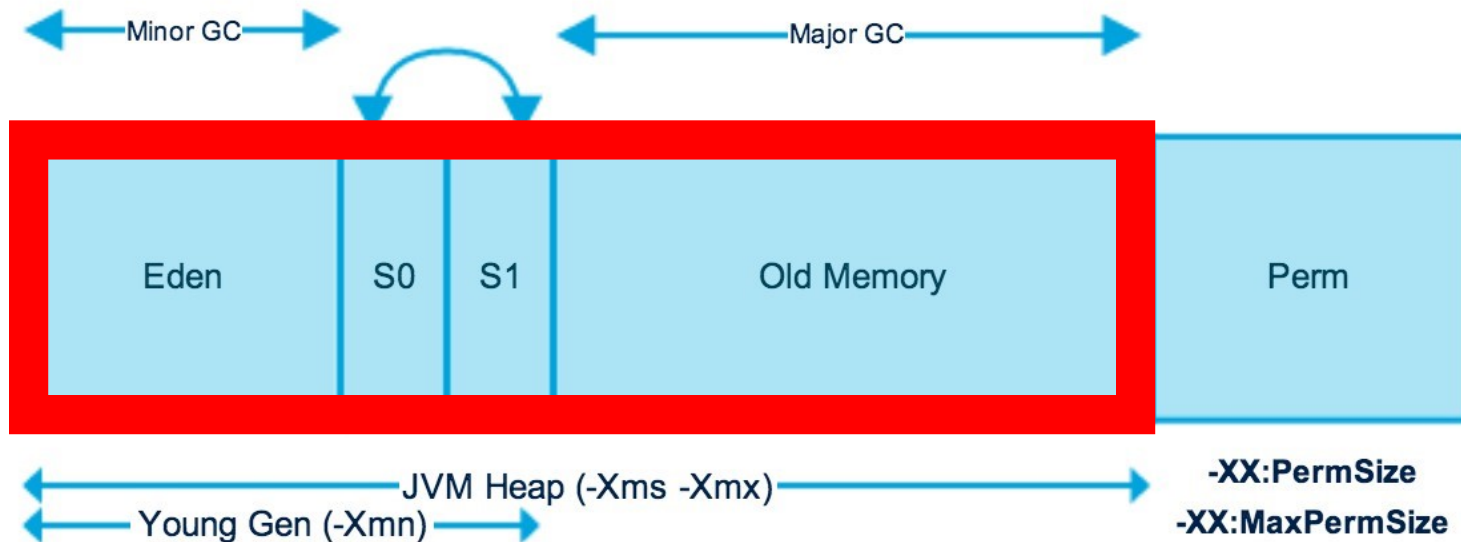
- La mémoire HEAP (littéralement le tas), gère principalement:
 - Les instances de classe
 - *Les tableaux*
- Elle est créée au démarrage de la JVM
- Sa taille par défaut est de 64MO
- Elle est continuellement assaini par un mécanisme automatique appelé le **garbage collector** (GC)
- *Sa taille peut être fixe ou peut varier en fonction de la stratégie du GC*
- *La taille de la HEAP peut être configuré via les option JVM suivante:*
 - **-Xmx<size>** : définis la taille maximum, ex: -Xmx512
 - **-Xms<size>** : définit la taille initiale, ex -Xms256

Concepts de base d'une JVM

Mémoire HEAP



- La mémoire HEAP (littéralement le tas), gère principalement:
 - Les instances de classe
 - *Les tableaux*



Concepts de base d'une JVM

Détail mémoire HEAP



- **Young generation**

- **Eden**

- Tous les objets créés dans les méthodes et les classes*

- **Survivor (0 et 1)**

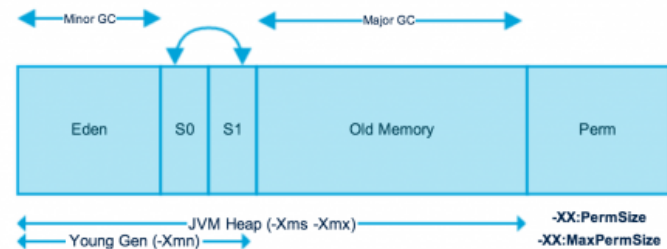
- Objets en utilisation ayant survécu à un GC*

- SWAP entre zone pleine et vide*

- **Old generation**

- Objets référencés depuis longtemps*

- Très gros objets (directement créé dans la zone)*



Concepts de base d'une JVM

Mémoire NON-HEAP



- La mémoire NON-HEAP, gère :
 - Structure des classes
 - Données de champs et méthodes
 - Code des méthodes et constructeurs
 - Appelé **PermGen** avant Java8 et maintenant **MetaSpace**
- Peu d'informations fournies par la JVM (*monitoring*)

Concepts de base d'une JVM

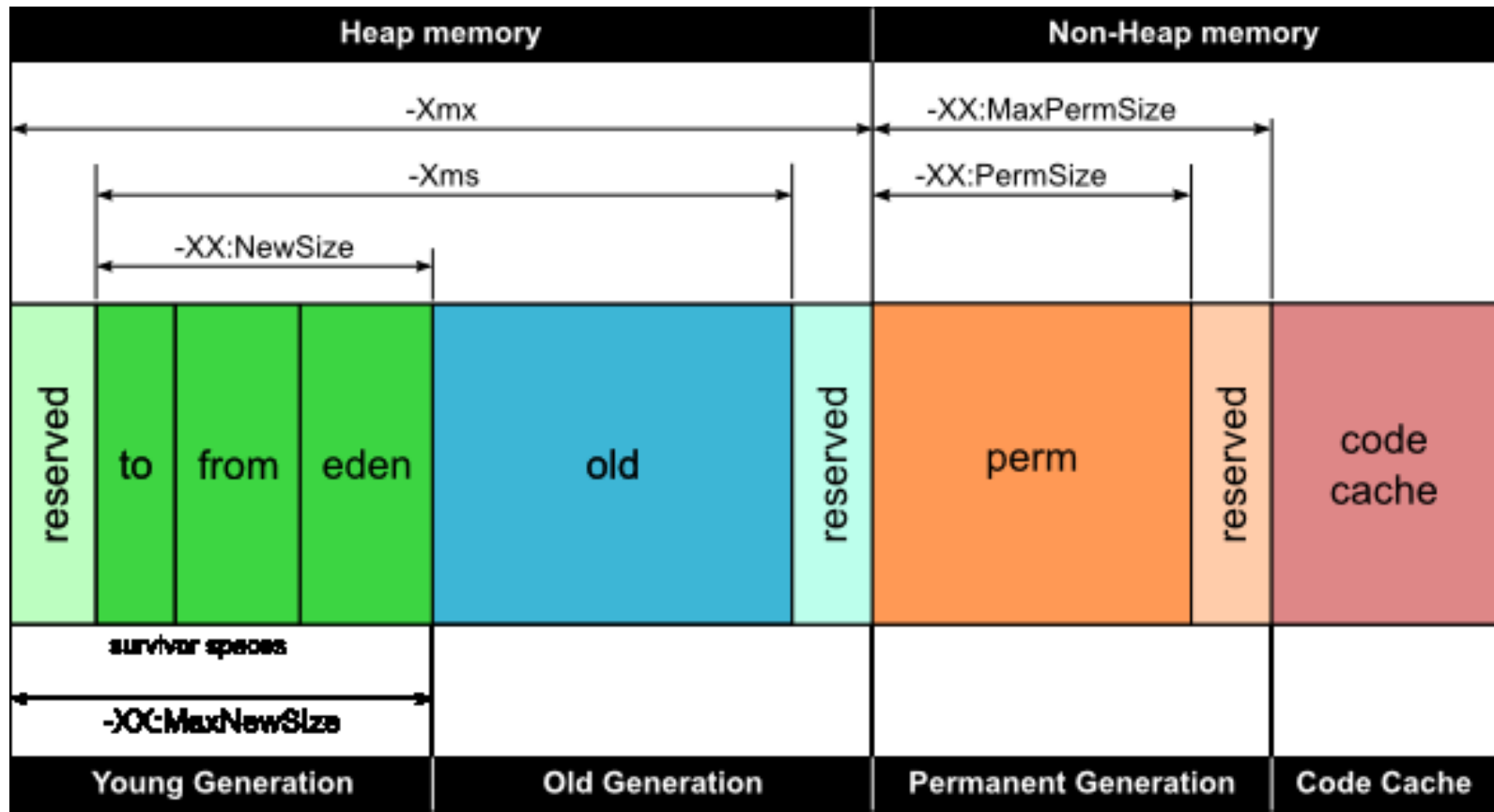
Mémoire Autre



- Le code de la JVM elle-même
- Les structures internes de la JVM
- Les différents agents de profiling

Concepts de base d'une JVM

Mémoire, *the Big Pictures*



Garbage Collector (GC)



Garbage Collector (GC)

Principes



- **Garbage Collector = Ramasse-miettes**
- C'est une fonctionnalité de la JVM qui gère la mémoire notamment en libérant celle des objets qui ne sont plus utilisés
- La mise en œuvre d'un GC possède plusieurs avantages :
 - Il **améliore la productivité du développeur** qui est déchargé de la libération explicite de la mémoire
 - Il participe activement à **la bonne intégrité de la machine virtuelle** : une instruction ne peut jamais utiliser un objet qui n'existe plus en mémoire
- Le GC a plusieurs rôles :
 - s'assurer que tout objet **dont il existe encore une référence** n'est pas supprimé
 - récupérer la mémoire **des objets inutilisés** (dont il n'existe plus aucune référence)
 - éventuellement **défragmenter** (compacter) la mémoire de la JVM selon l'algorithme utilisé
 - intervenir **dans l'allocation de la mémoire** pour les nouveaux objets à cause du point précédent

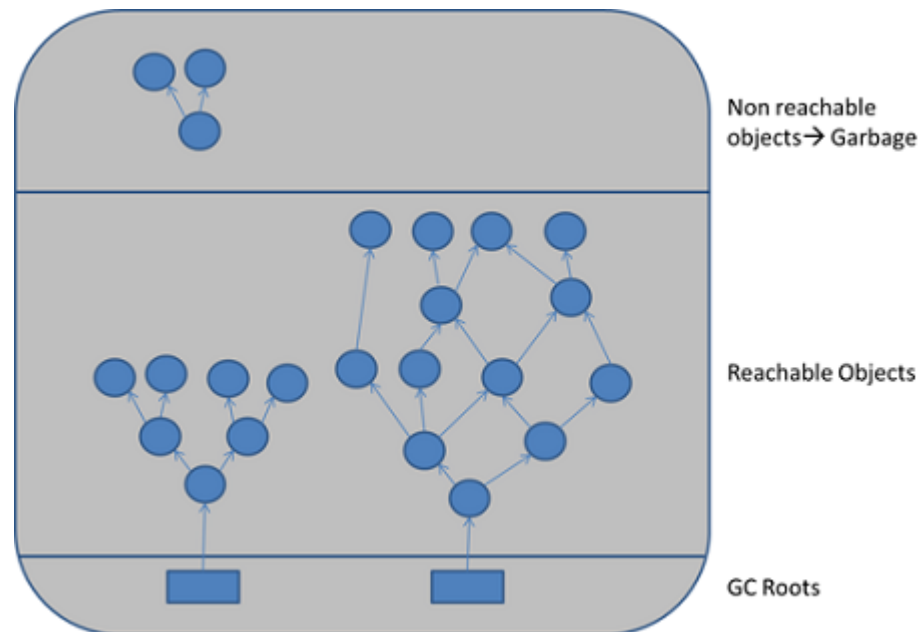
Garbage Collector (GC)

Objets atteignable (références existantes)



■ Quatre types de racines (roots)

1. Variables locales
2. Threads java actifs
3. Variables statiques
4. Références JNI

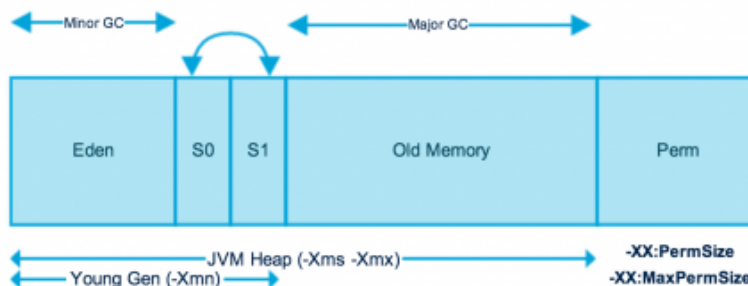


Garbage Collector (GC)

Fonctionnement



- Les **nouveaux objets** sont créés dans la zone **Eden**
- Quand l'**Eden** est plein, un cycle GC (minor GC) collecte les objets atteignables et les déplace dans la zone **survivor** (S0, S1)
- Le minor GC parcourt également les objets de la zone **survivor** et les déplace. A un instant donné il y a toujours un espace **survivor** qui est vide
- Les objets ayant survécu à plusieurs de ces cycles sont déplacés dans la zone **Old Generation** (Old Memory)



Garbage Collector (GC)

Algorithmes de collections



- **Mark & Sweep**
- **Stop & Copy**

Garbage Collector (GC)

Mark & Sweep



- A chaque cycle GC...
- Phase Mark
 - Instanciation d'un objet → marqué à **false**
 - Parcours des instances et marquage des éléments atteignable → **true**
- Phase Sweep
 - Parcours des éléments du *Heap*
 - Suppression de tout ce qui est marqué à **false**
 - Les éléments restants sont marquées → **false**

Garbage Collector (GC)

Mark & Sweep



```
mark_sweep(root)
```

```
  For rootObjet in root
    mark(rootObjet)
  End For
  sweep()
```

```
mark(root)
```

```
  If markedBit(root) = false then
    markedBit(root) = true
  For each child referenced by root
    mark(v)
```

```
sweep()
```

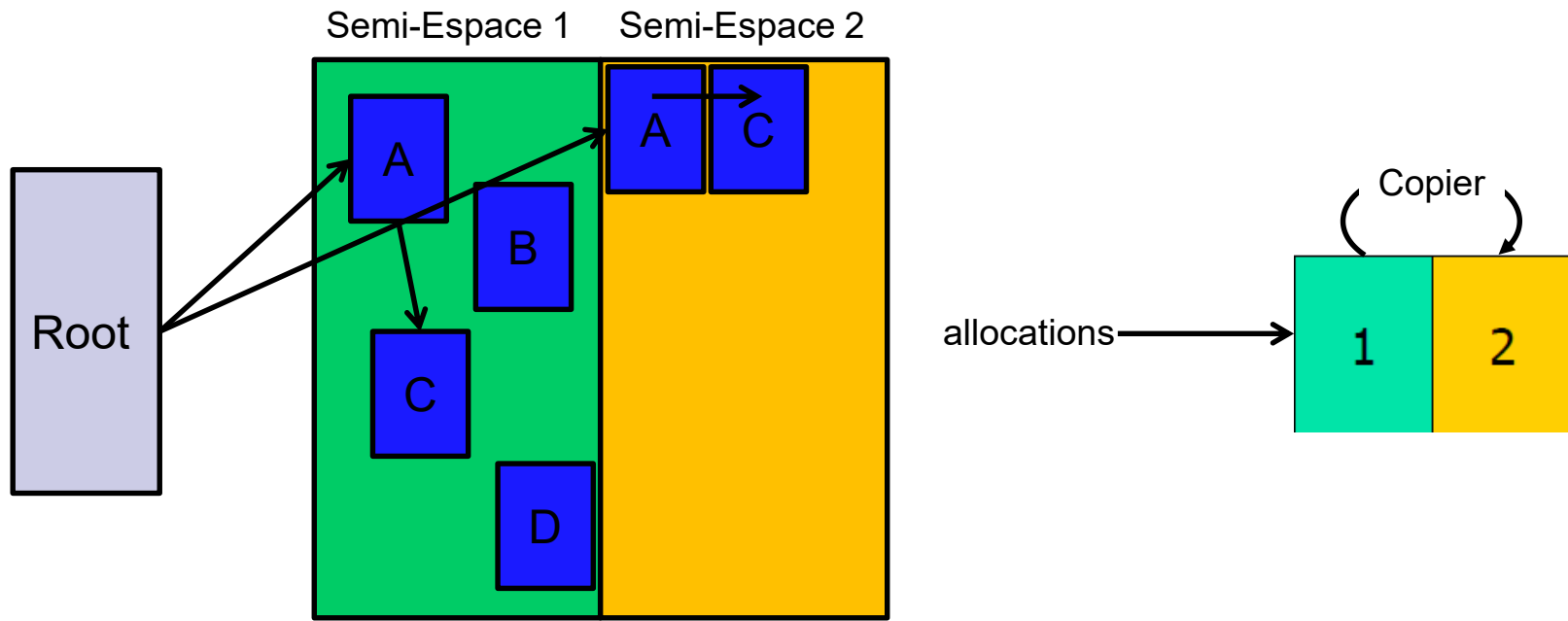
```
  For each object o in heap
    If markedBit(o) = true then
      markedBit(o) = false
    else
      heap.release(o)
```

Garbage Collector (GC)

Stop & Copy



- Espace heap est divisé en 2 semi-espaces
- Les objets sont alloués dans le semi-espace 1
- Lors du GC, les objets référencés sont copiés dans le sous-espace 2

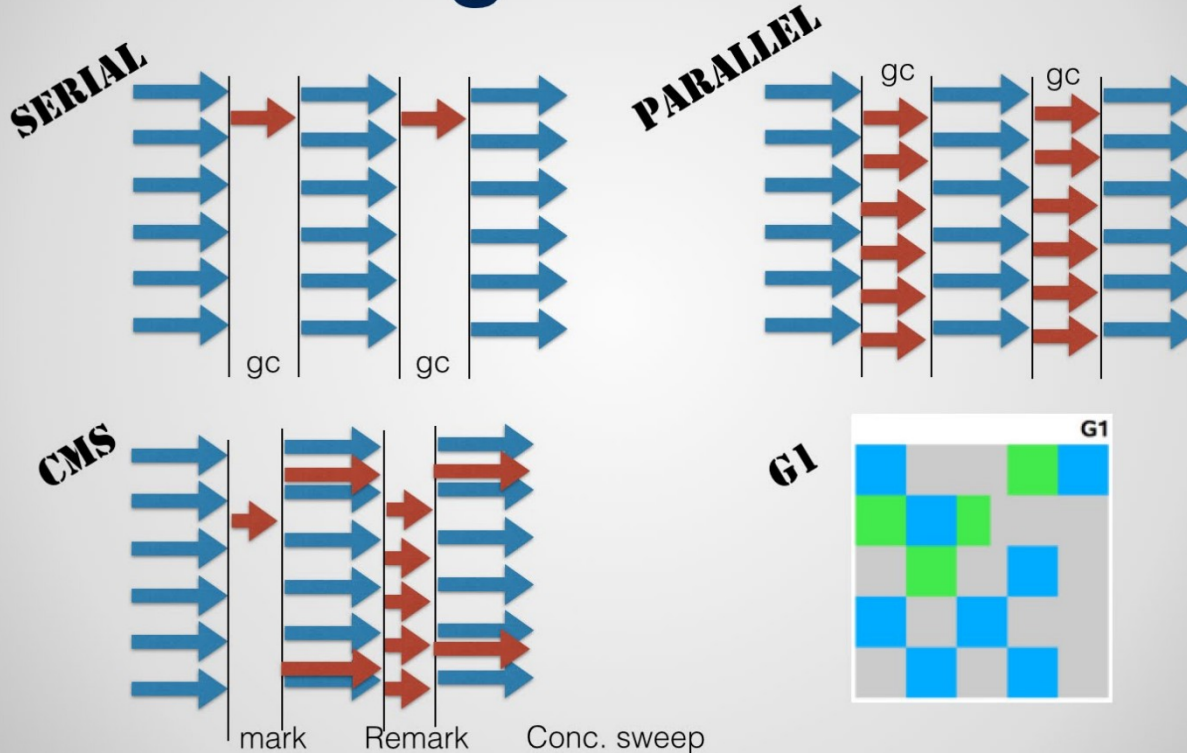


Garbage Collector (GC)

Stratégie de collections



Garbage Collectors



Garbage Collector (GC)

Stratégie de collections



- **Serial Garbage Collector (-XX:+UseSerialGC)**
 - *Uniquement utilisé pas les petites applications (1 CPU)*
 - *Single Thread*
 - *GC Bloquant*
- **Parallel Garbage Collector (-XX:+UseParallelGC)**
 - *Collecteur par défaut pour la JVM*
 - *Bloquant, comme Serial, mais multi-threader*
- **CMS Garbage Collector (-XX:+UseConcMarkSweepGC)**
 - *Multiple thread en même temps pour GC*
 - *Rarement bloquant (cas de figure définis)*
- **G1 Garbage Collector (-XX:+UseG1GC)**
 - *Depuis java 7*
 - *Optimiser pour les grosses heap*
 - *Compactage de la heap libre*

The diagram illustrates a Java EE application architecture. On the left, there are three client types: 'Client Java' (JMS, JPA), 'Client http' (HTTP, HTTPS), and 'Client SOAP' (SOAP, JMS, WS, WSP, CAA). These clients interact with a 'Serveur d'application J2EE' on the right. The server architecture is layered:

- Web Container (Couche Web)**: Contains 'Servlets JSP', 'Servlets HTTP/HTTPS', 'Servlets SOAP', and 'Servlets RESTful'. It connects to the 'Client SOAP' via 'SOAP XML' and to the 'Client http' via 'HTTP/HTTPS'.
- Spring ou EJB Container (Couche Moteur)**: Contains 'Compensateur pour J2EE (Transaction)', 'Entity', and 'Entity'. It connects to the 'Web Container' via 'JMS, JPA' and to the 'Data Base' via 'JMS, JPA'.
- Data Base**: A 'SGBD' (Database Management System) that stores data.

 Arrows indicate the flow of data and interactions between these components.

Serveur d'application JEE

Principes



- Une application **JEE** fonctionne dans un container appelé **serveur d'application JEE**
- Un serveur d'application **JEE** doit implémenter la **norme JEE**
- Les **services JEE** (bibliothèques, implémentations) sont fournies **par le serveur d'application (environnement d'exécution)**

Serveur d'application JEE

Implémentations



- IBM WebSphere
- BEA WebLogic
- Oracle 9i AS
- Jboss
- Geronimo
- TomEE

Tomcat **n'est pas** un serveur *d'application JEE*. Il implémente juste un **container de servlet**

Serveur d'application JEE

Architecture de référence

