Machines Virtuelles
et bazard autour

Rémi Forax

Avant propos

Quelle est la complexité du code ci-dessous ?

Avec un processeur à 1Ghz, combien de temps le calcul prendra t'il ?

```
public static void main(String[] args) {
  int sum = 0;
  for(int i=0; i<Integer.MAX_VALUE; i++) {
    for(int j=0; j<Integer.MAX_VALUE; j++) {
      sum++;
    }
  }
  System.out.println(sum);
}
```

Plan du cours

- Langage & Runtime
- Anatomie d'une machine virtuelle
- Front-end: AST, typechecker, bytecode
- Verifier
- Linker
- Compiler
- GC
- JSR 292

Java != C

Et si le code est le suivant ?

```
public static void main(String[] args) {
  int sum = 0;
  for(int i=0; i<Integer.MAX_VALUE; i++) {
    sum = m(sum);
  }
  System.out.println(sum);
}

private static int m(int sum) {
  for(int j=0; j<Integer.MAX_VALUE; j++) {
      sum++;
  }
  return sum;
}
```

Java != C

```
public static void main(String[] args) {
  int sum = 0;
  for(int i=0; i<Integer.MAX_VALUE; i++) {
    sum = m(sum);
  }
  System.out.println(sum);
}

private static int m(int sum) {
  for(int j=0; j<Integer.MAX_VALUE; j++) {
      sum++;
  }
  return sum;
}
```

```
java -server -XX:+PrintCompilation ...
     72    1 %           Test1::m @ 5 (19 bytes)
   1470    1             Test1::m (19 bytes)
   2836    2 %           Test1::main @ 7 (22 bytes)
```

Langage & Plateforme

- Un language informatique
  - Une description souvent textuelle
  - Fonctionne sur une plateforme

    x86/Linux, x86/Windows, Java, CLR

- Le language utilise des fonctionnalités
  - Une fonctionnalité est soit fournie par la plateforme soit par le runtime du langage

    par ex: les exceptions
    - Java language => Java la plateforme
    - C++ => insère du code + gotos

Typage ?

Les instructions des CPUs sont typés

- Unités de calcul entier / unités calcul flottant

Langages typés statiquement

- L'utilisateur ou le compilateur fourni les types
- Le code utilisé par le runtime est typé

Langages typés dynamiquement

- Pas de type déclaré
- Le runtime vérifie les classes (type dynamique) avant d'effectuer l'opération ou même choisie l'opérateur en fonction des classes

Langage dynamique ?

Dynamique ?

- Type inexistant, on utilise la/les classes à l'exécution
- On peut modifier les structures
  - Ajouter/retirer des champs
  - Ajouter/retirer des méthodes
  - Changer le schéma de délégation
  - Changer le code exécuter (eval)
  - Changer des valeurs de la pile d'appel

  etc.

Un poil d'histoire

- PDP-7/B: tous est un int
- PDP-11/C: B + type primitif + struct
- SmallTalk: tout est class, pas de type primitif, pas typé
- C++: type primitif + struct + classe, typé
- Java: type primitif + classe, typé
- OCaml: fonctionel + mixin, typage par le compilo
- Python: fonctionel + classe (open), non typé
- Ruby: fonctionel + classe (open) + mixin, non typé
- Javascript: fonctionnel + hash + delegation, non typé
- Scala: fonctionnel + classe + trait + ..., typé
- Dart: fonctionel + classe, type optionel (pas présent à l'exécution)

Langage, compilateur & runtime

VM Java

Method handle
Primitive types + ops
Object model
Memory model
Dynamic Linking
Access Control
GC
Unicode

Java le langage

Checked Exceptions
Generics & enums
Overloading
Constructor Chaining
Program Analysis
Lambda Java 8 (MH)
Primitive types + ops
Object model
Memory model
Dynamic Linking
Access Control
GC
Unicode

Ruby le langage

Open classes
Dynamic typing
eval
Mixin
Regular expressions
Closure, block (MH)
~~**Primitive types + ops**~~
Object model
Memory model
Dynamic Linking
~~**Access Control**~~
GC
~~**Unicode**~~

Machine virtuelle ?

- 1 environment d'exécution
  - abstrait la plateforme sous-jacente
    - Fourni le même environement abstrait sur différente plateforme
  - interagit avec le code exécuté
    - Profile exécution du programme
    - Adapte le code exécuté dynamiquement
      - boucle de rétroaction ?

C++ runtime

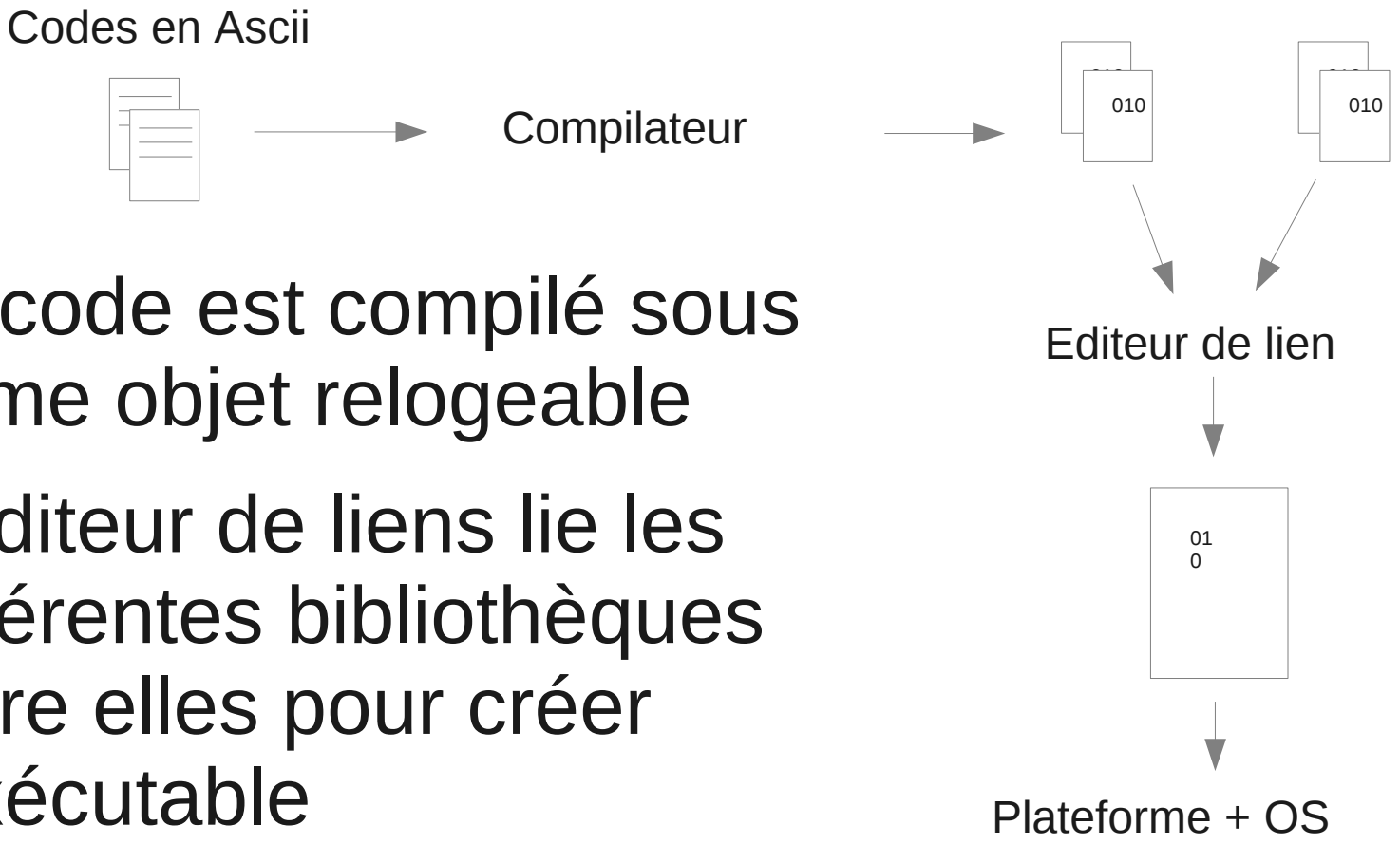
Le C++ possède un runtime mais pas de machine virtuelle

Runtime du C++

- Une bibilothèque standard
- Une ABI standard
- Code builtin inséré

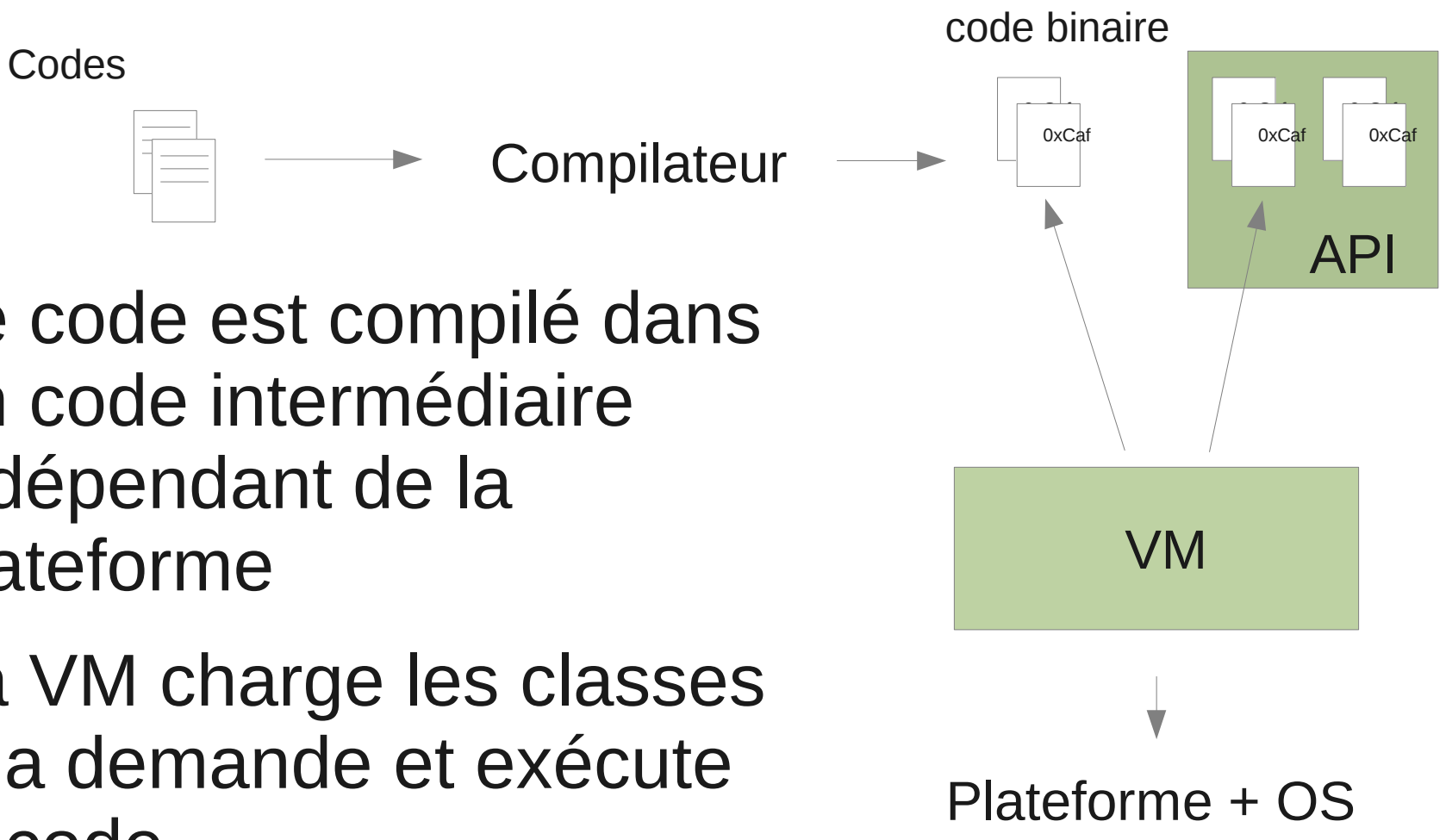
donc pas une machine virtuelle

Architecture en C/C++



- Le code est compilé sous forme objet relogeable
- L'éditeur de liens lie les différentes bibliothèques entre elles pour créer l'exécutable

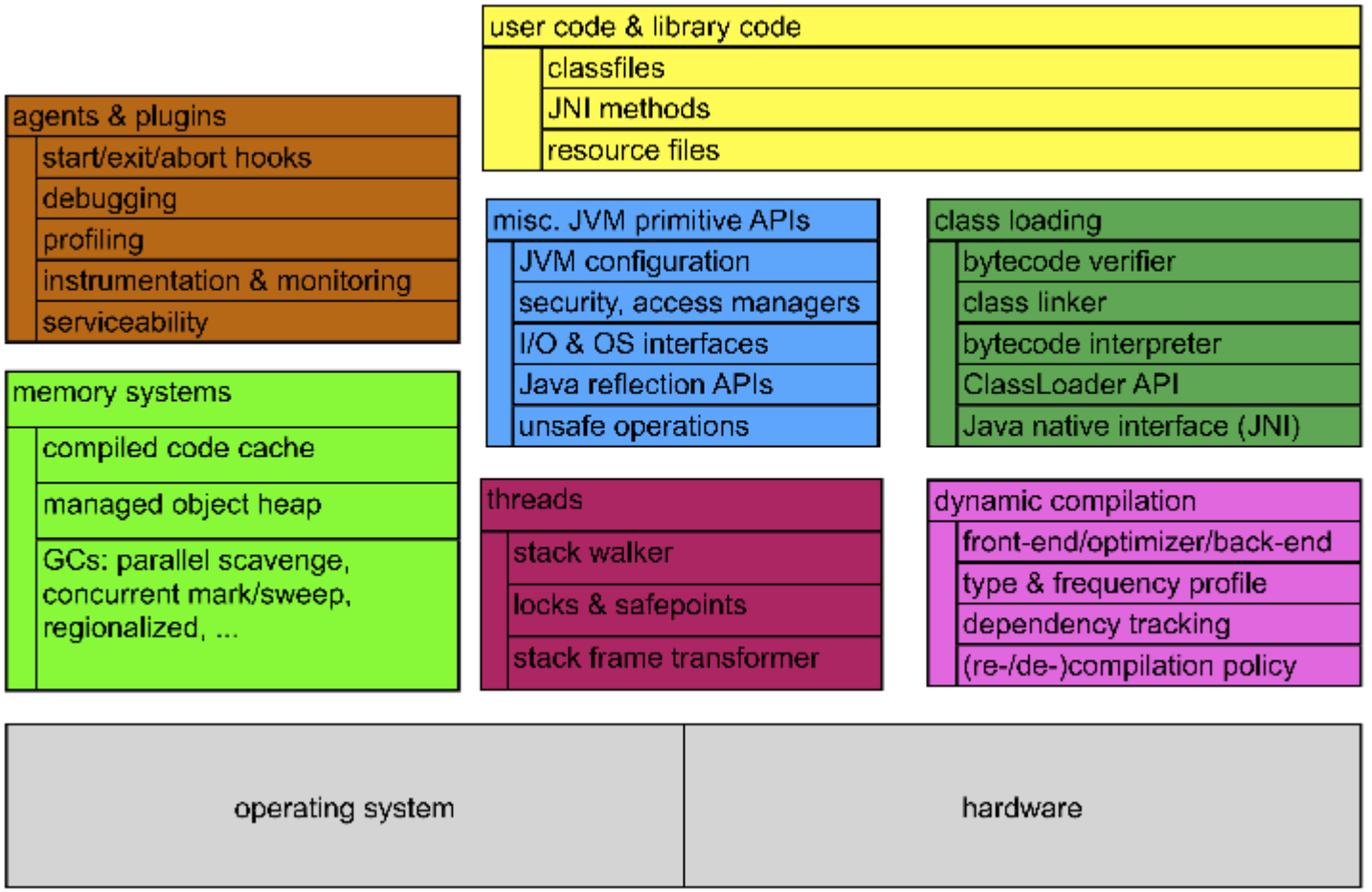
Architecture avec une VM



- Le code est compilé dans un code intermédiaire indépendant de la plateforme
- La VM charge les classes à la demande et exécute le code

Anatomie d'une VM

Exemple, la JVM



Toward the universal VM (c) Oracle

VM & Format d'entrée

- code binaire à pile

  Microsoft CLR, JVM, Smalltalk Squeak

- code binaire à registre

  Dalvik (DEX), Parrot (Perl?)

- code source

  Google V8 (Javascript)

Verification du code

- Certaines VMs vérifient que le code est valide avant exécution
  - Les VMs Java vérifie le code des applis pas le code du JDK

    Peut prendre bcp de temps

    surtout si l'algo est stupidement exponentiel !

  - Dalvik vérifie le code à l'installation et calcul un hachage cryptographique dessus pour vérifier si le code à changer à postériori.

Interpreteur ?

- Certaines VMs ne possèdent qu'un interpreteur
  - Perl 5
    - Perl6 (rakudo) est basé sur Parrot
  - Python
    - Interpreter avec un bytecode
  - Ruby
    - CRuby 1.8, AST-walker
    - CRuby 1.9, utilise un bytecode
- Les autres génèrent du code assembleur à la volée (JIT)
  - Java (IBM J9, Oracle Hotspot, Oracle JRockit) / Javascript (Google V8 / Mozilla *Monkey) / C# / Lua

AST interpreter

- Le language est représenté par un arbre abstrait
- On parcourt l'arbre en faisant des appels récursifs à la fonction d'évaluation sur chaque noeud de l'arbre
- Bref on utilise la pile d'appel comme pile d'évaluation
- Facile à implanter mais peut être très lent en pratique

Bytecode interpreter

Une boucle + 1 switch

Le switch indique ce qui faut faire pour chaque instruction

Pas rapide mais

- On peut utiliser les goto labels de GCC
- On peut spécifier chaque case (ou au moins les plus importants) du switch en assembleur

  Template interpreter ou Patch interpreter

  - Hotspot, Dalvik

  peut être suffisamment rapide par rapport à code compilé car le code de l'interpreteur réside dans le cache L1 d'instructions du processeur

  note: Il n'y a pas de d'instruction de prefetch pour les instructions

JITs

- Plusieurs stratégies
  - Just in Time

    On transforme en assembleur lors du premier appel
  - Tiered

    Souvent deux JITs, 1 simple rapide + 1 second optimisé (plus lent/code + rapide) si la méthode est chaud.
  - Mixed-mode

    Interpreter + JITs (1 ou plusieurs)

    Oracle Hotspot (1 interpreter + 2 JITs)
  - Ahead of time

    Le code est JITé à l'installation

    C# native image generator (ngen)

Pourquoi en 2 étapes

Plusieurs raisons

L'ensemble de l'application n'est pas exécuté de la même façon

- Une partie du code n'est pas exécuté souvent voir 1 seul fois

Optimizé correctment du code prend du temps

Donc

Cela sert à rien de générer un code optimizé pour tout le code

- Interpreteur ou JIT rapide mais avec peu d'optimizations
- Si le code est exécuté souvent, on peut toujours générer un meilleur code (+ rapide) plus tard
  - Dans ce cas, ou pourra Jité en parallèle de l'exécution du code

De plus, on peut profiler le code et utiliser les informations dynamiques

Garbage Collector

- Algorithme utilisé pour réclamer la mémoire de plusieurs objets lorsque ceux-ci ne sont plus accessibles
- Il se déclenche
  - Lors d'un new si il n'y a plus assez de mémoire
  - Périodiquement
    - en minutes sur un desktop, en milisecondes sur un server
- Limitation
  - Le GC doit connaitre la mémoire totale qui lui sera allouée.

Mark !

Contrairement à une idée assez répandu, les GCs modernes ne comptent pas les références sur les objets

- Les GCs de Perl (avant v6), Python ou CRuby ne sont modernes !

On utilise un algo de marquage qui parcours tous les objets atteignables, ceux qui ne sont pas atteignables sont considérés comme mort !

Garbage Collector

Il existe deux algos différents

- Mark & Sweep
- Mark & Compact

L'algo de marquage

A partir des objets racines (roots):

- Des références stockées dans les piles des threads
- Les variables statiques

puis récursivement marque tous les objets atteignables

Ce marquage peut se faire en parallèle de l'exécution du programme

-

Mark & sweep

- Phase 1: mark

  À partir des piles de chaque threads et des variable statiques, le GC marque tous les objets atteignablent

- Phase 2: sweep

  Le GC libère la mémoire (comme free) des objets non-marqué

- Inconvénient

  on doit gérer une liste des blocks libres, etc

Mark & compact

- Phase 1: mark

  À partir des piles de chaque threads et des variable statiques, le GC marque tous les objets atteignablent
- Phase 2: compact
  - Le GC recopie tout les objets marqués dans une autres zones
- Inconvénient
  - Cela prend le double de place en mémoire

GC générationel

Hypothèses:

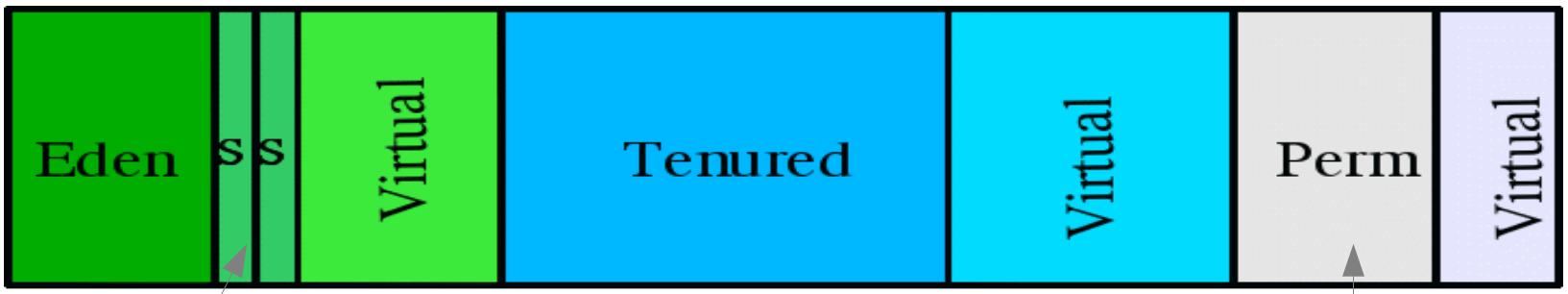
- les objets allouées en même temps ont la même durée de vie
- beaucoup d'objets meurent rapidement

Il n'est nécessaire de collecter l'ensemble de la mémoire, les zones recevant des objets vieux peuvent être collecter moins fréquemment (major collection) que les zones recevant des objets jeunes (minor collection)

Generationnel mark & compact/sweep

Utilise les 2 techniques et les 2 hypothèses générationnels

- mark & compact eden -> survivor, survivor -> survivor, survivor -> tenured
- mark & sweep tenured



survivor space

Class + JITed code