Développement en JAVA

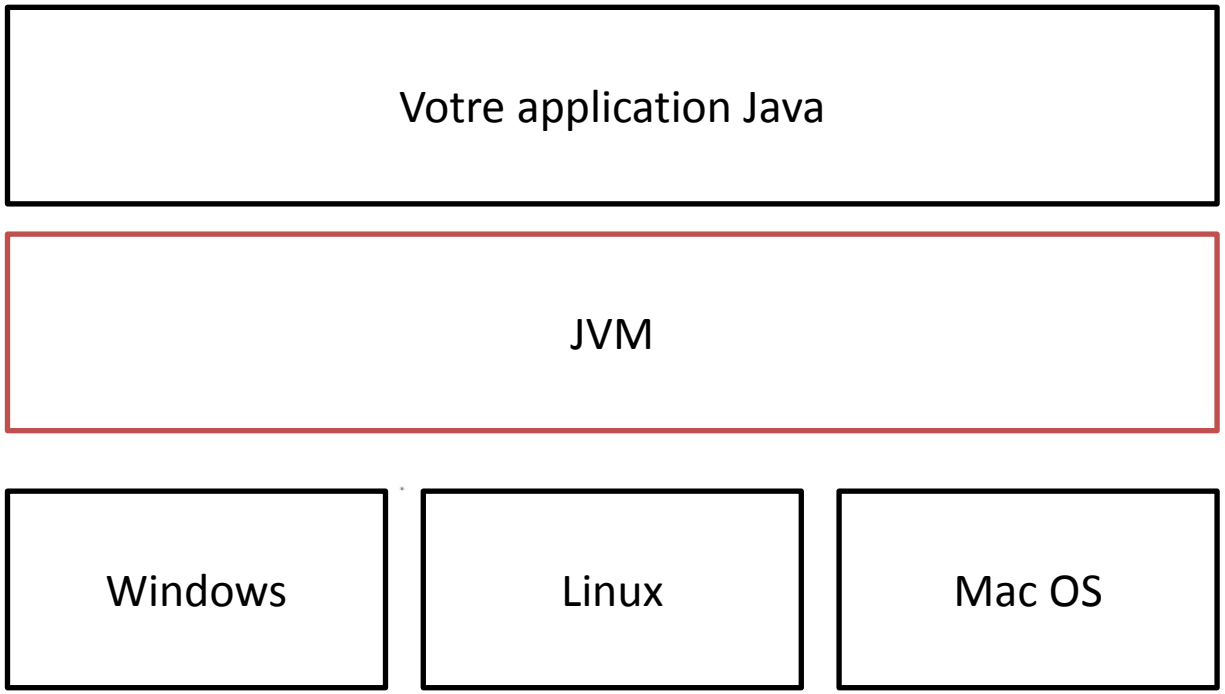
JVM et compilation

Agenda

- JVM
- Bytecode
- Gestion de la mémoire,
- Heap, stack
- Garbage collection
- Langage compilé vs interprété

La JVM

- **J**ava **V**irtual **M**achine

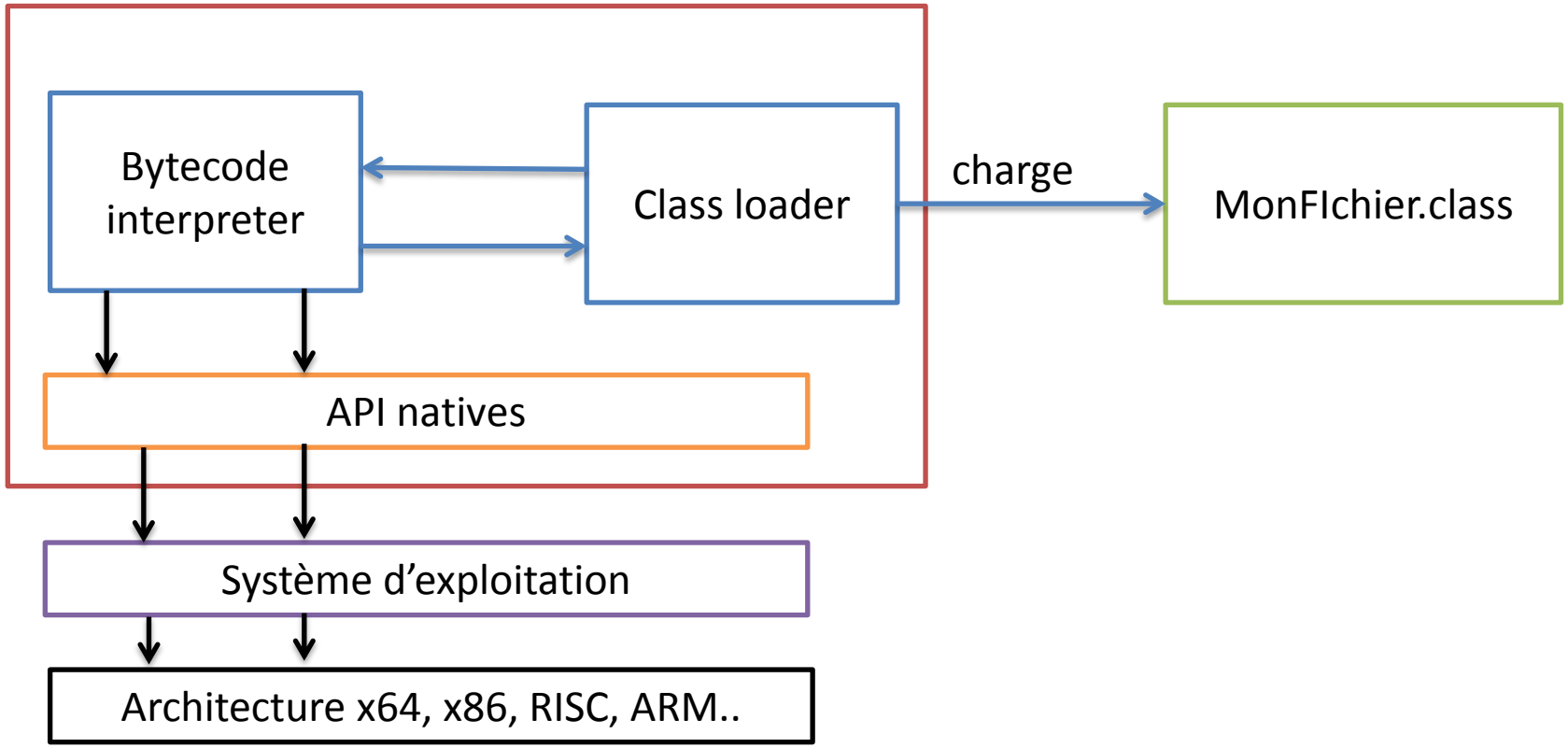


La JVM

- La JVM est environnement d'exécution
- Elle offre une abstraction des APIs système
- Elle sécurise les applications en limitant les possibilité d'accès à l'OS: notion de « sandbox »
- Elle gère la mémoire
- Est écrite en C++

La JVM

Architecture simplifiée de la JVM



La JVM

- La JVM interprète du bytecode
- Le bytecode est un langage intermédiaire possédant un jeu d'instruction similaire à de l'assembleur
- On parle donc de machine virtuelle car l'interpréteur de bytecode de la JVM se comporte comme un processeur

Just in time compilation

- Le Java est un langage qui compile en bytecode, comme le C est un langage qui compile en assembleur
- Le bytecode est portable d'une architecture à une autre
- L'interpréteur de la JVM traduit le bytecode du programme en instruction pouvant être exécuté par la machine physique

  => On parle de **Just In Time compilation**

Just in time compilation

- Le bytecode peut être optimisé à la volée par la JVM afin de le rendre plus rapide

Compilation statique

Java n'est pas le seul langage compilant en bytecode et pouvant être exécuté par la JVM:

- Clojure
- Scala
- Groovy
- Ceylon
- JRuby…

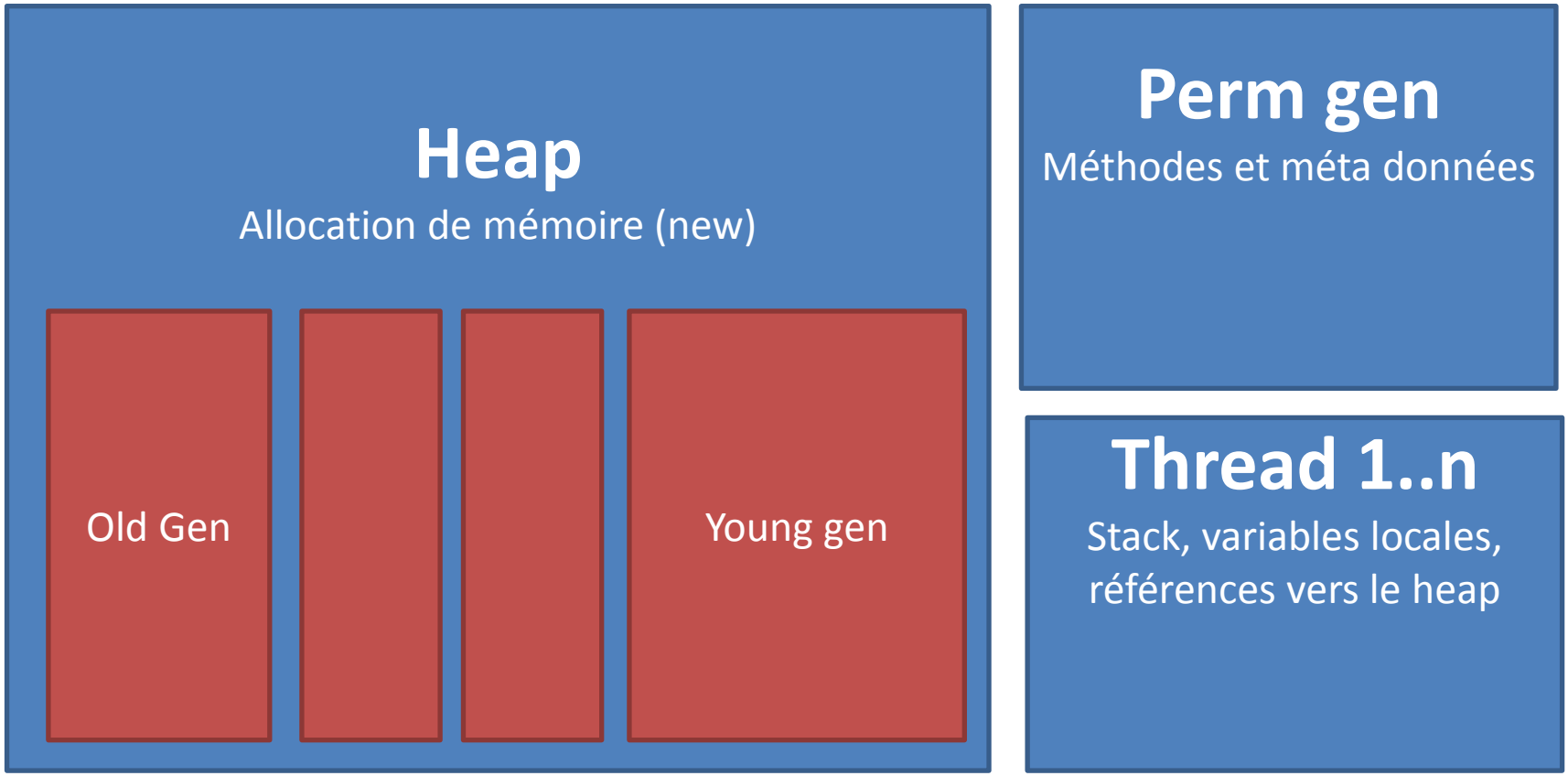
Compilation « classique »

- Avant de transformer le code Java en bytecode, le compilateur effectue des vérifications:
- Lexicale
  - Ce mot clé appartient-il bien au langage ?
    - Exemple: Faute de typo
- Syntaxique
  - Cette instruction est-elle bien formatée ?
    - Exemple: point virgule manquant
- Sémantique
  - Ce code a-t-il vraiment du sens ?
    - Exemple: utilisation d'une variable non initialisé

La gestion de la mémoire

- Contrairement au C et au C++, la mémoire n'est pas gérée directement par le développeur
- La JVM se charge d'allouer et de désallouer la mémoire en fonction des besoins du programme

Structure de la mémoire

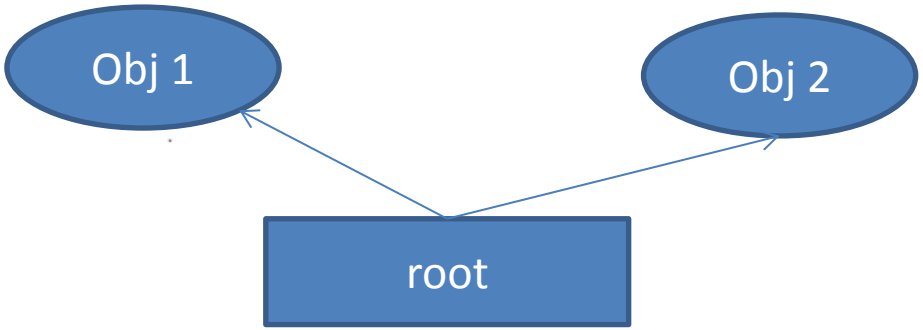


Le Garbage Collector

- Le garbage collector est un service de la JVM en charge de l'entretien de la mémoire
- Il parcourt le graph de dépendances et élimine les objets qui ne peuvent plus être utilisés
- Lors de son exécution **le garbage collector stop l'exécution du programme**

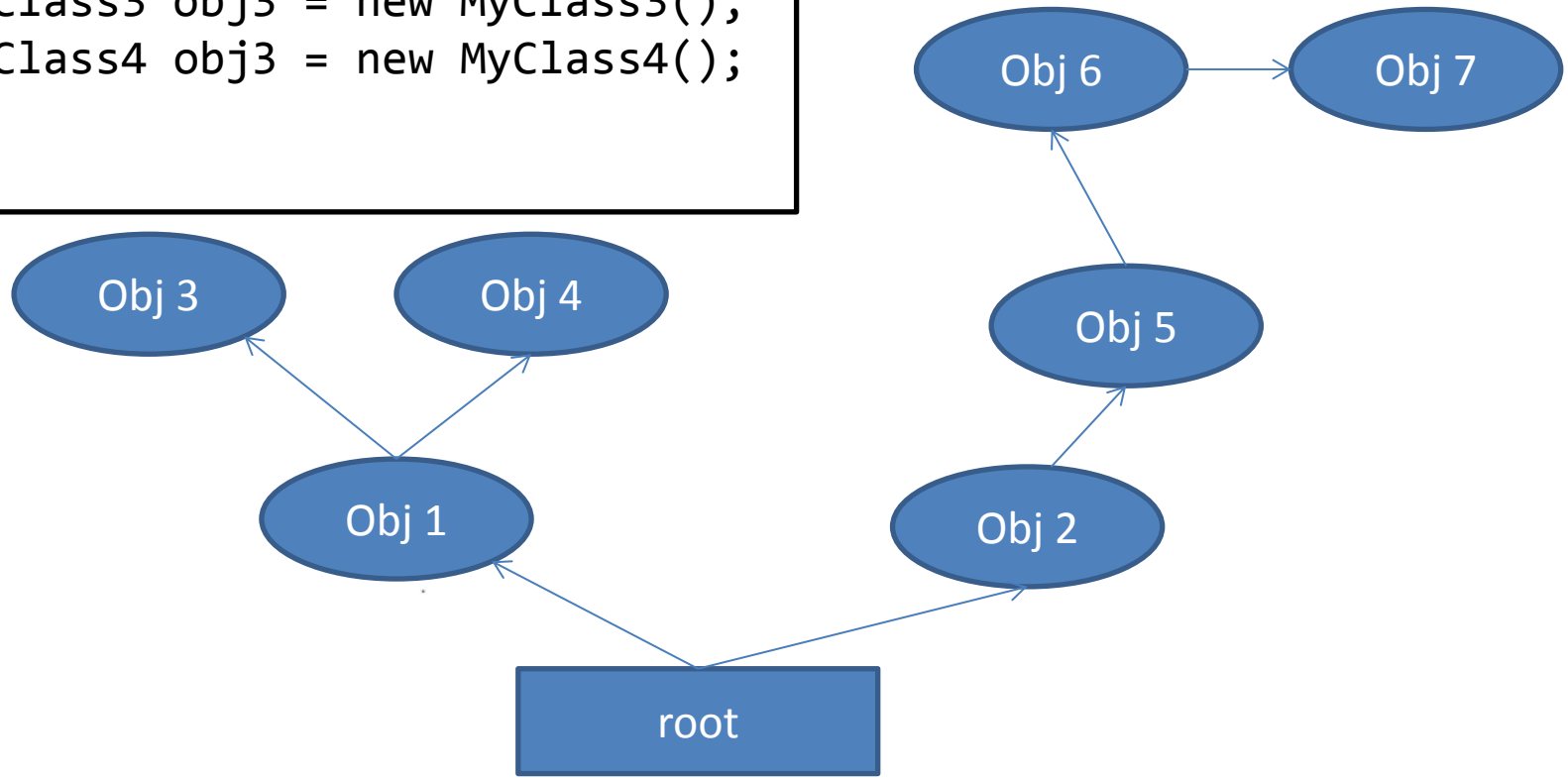
Le Garbage Collector

```
Object obj1 = new MyClass1();
Object obj2 = new MyClass2();
```



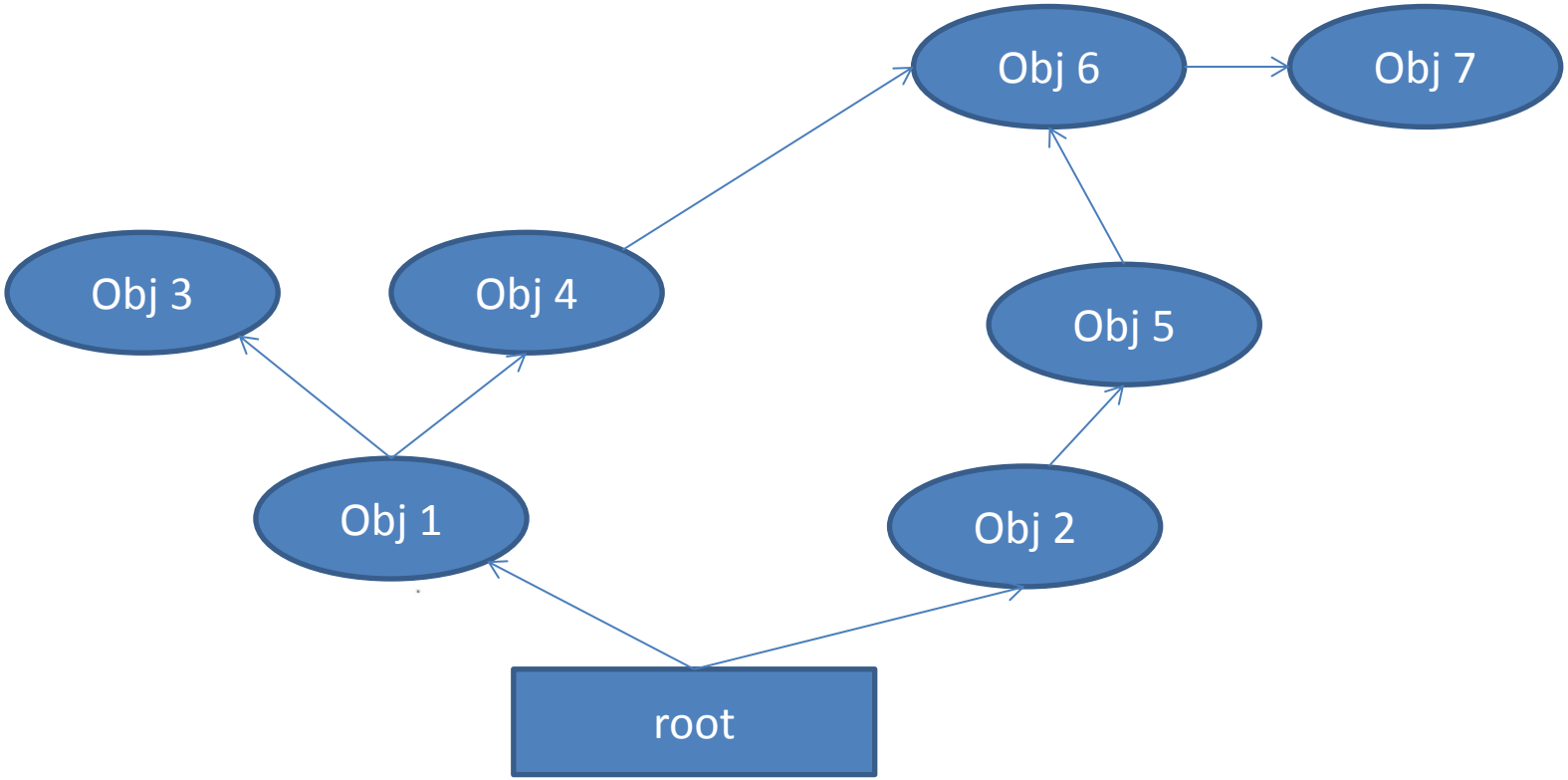
Le Garbage Collector

```
Public MyClass1{
  MyClass3 obj3 = new MyClass3();
  MyClass4 obj3 = new MyClass4();
}
…
```



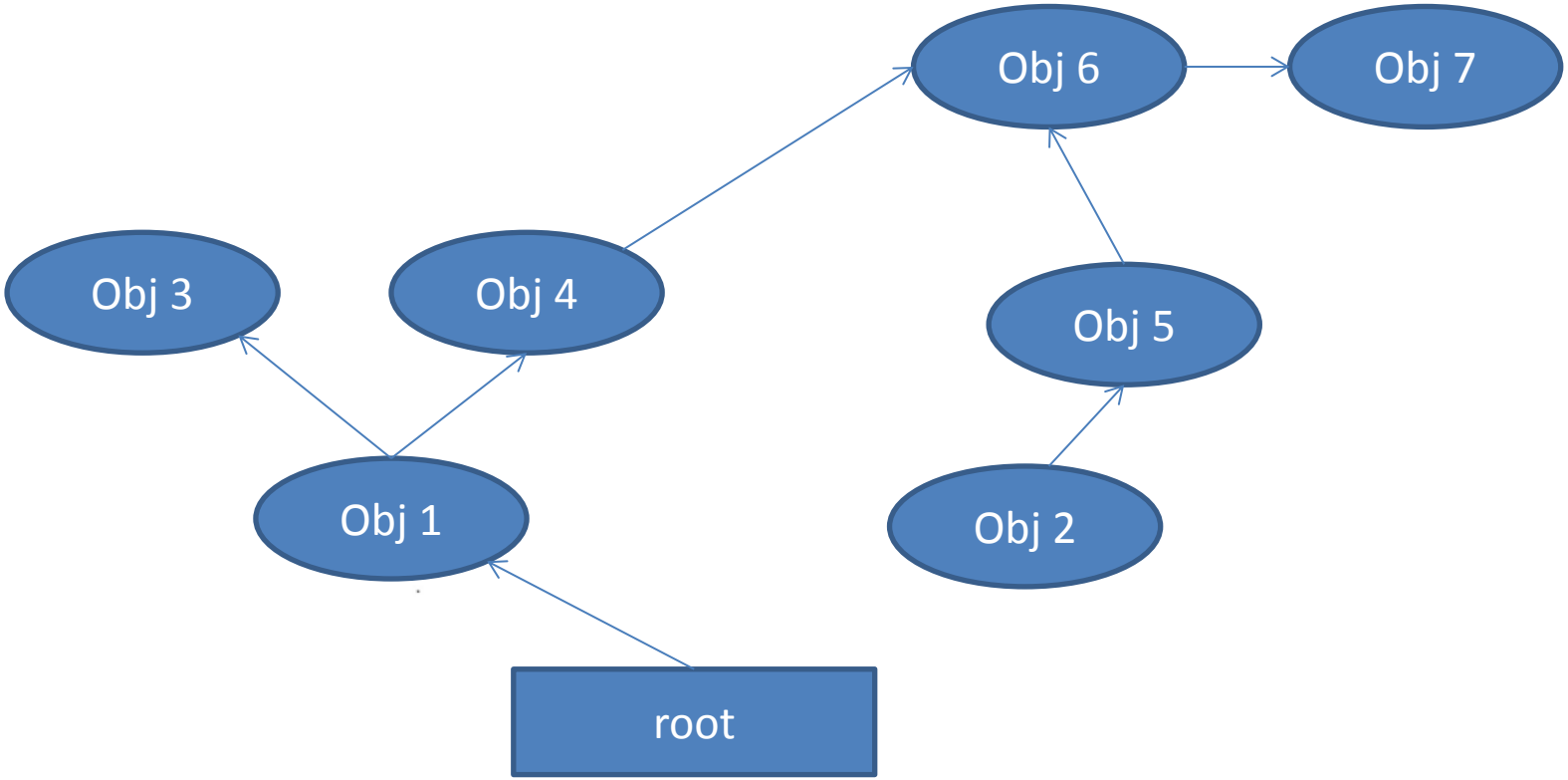
Le Garbage Collector

```
obj4.set(obj6);
```



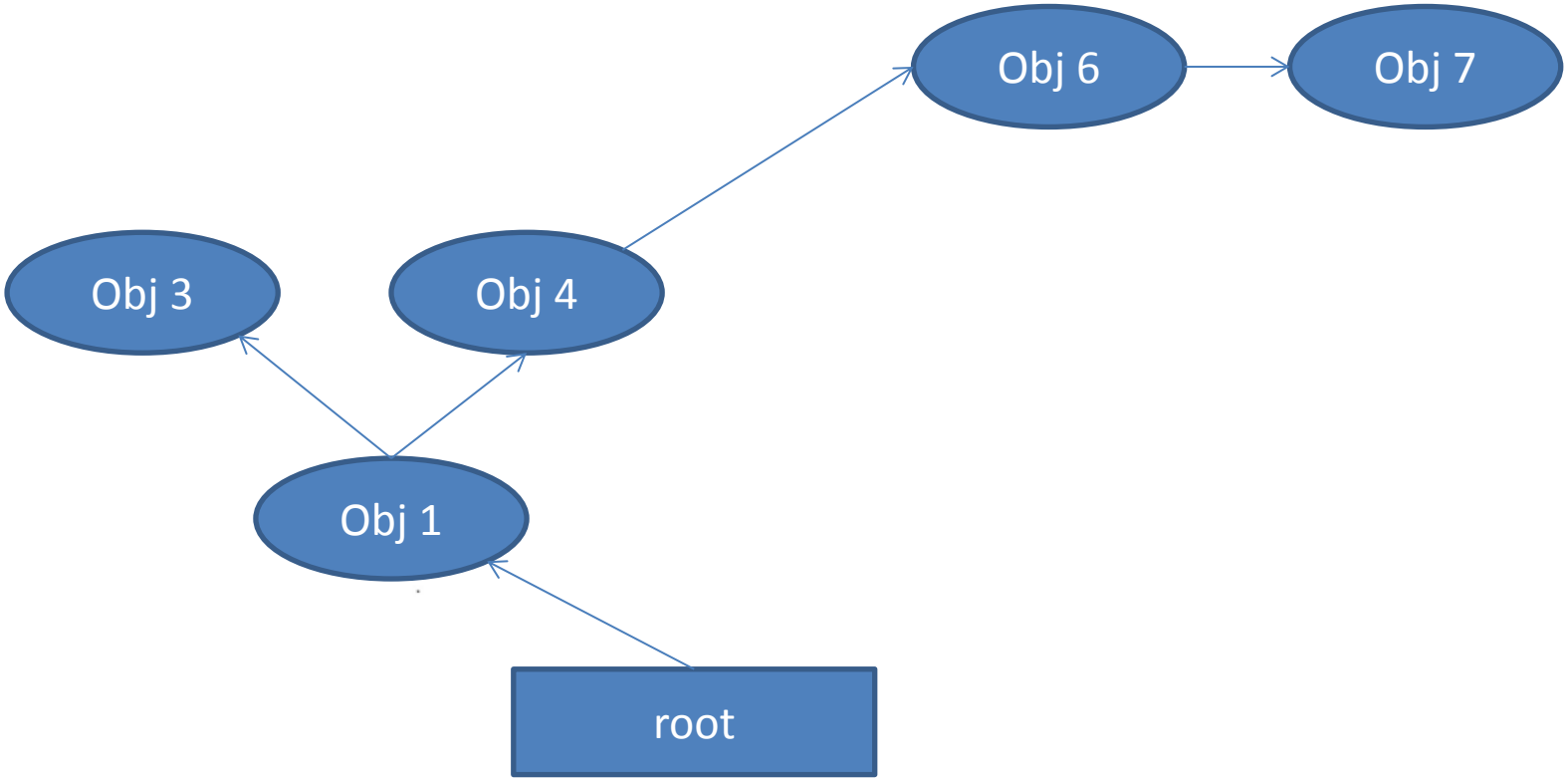
Le Garbage Collector

L'objet 2 sort du scope



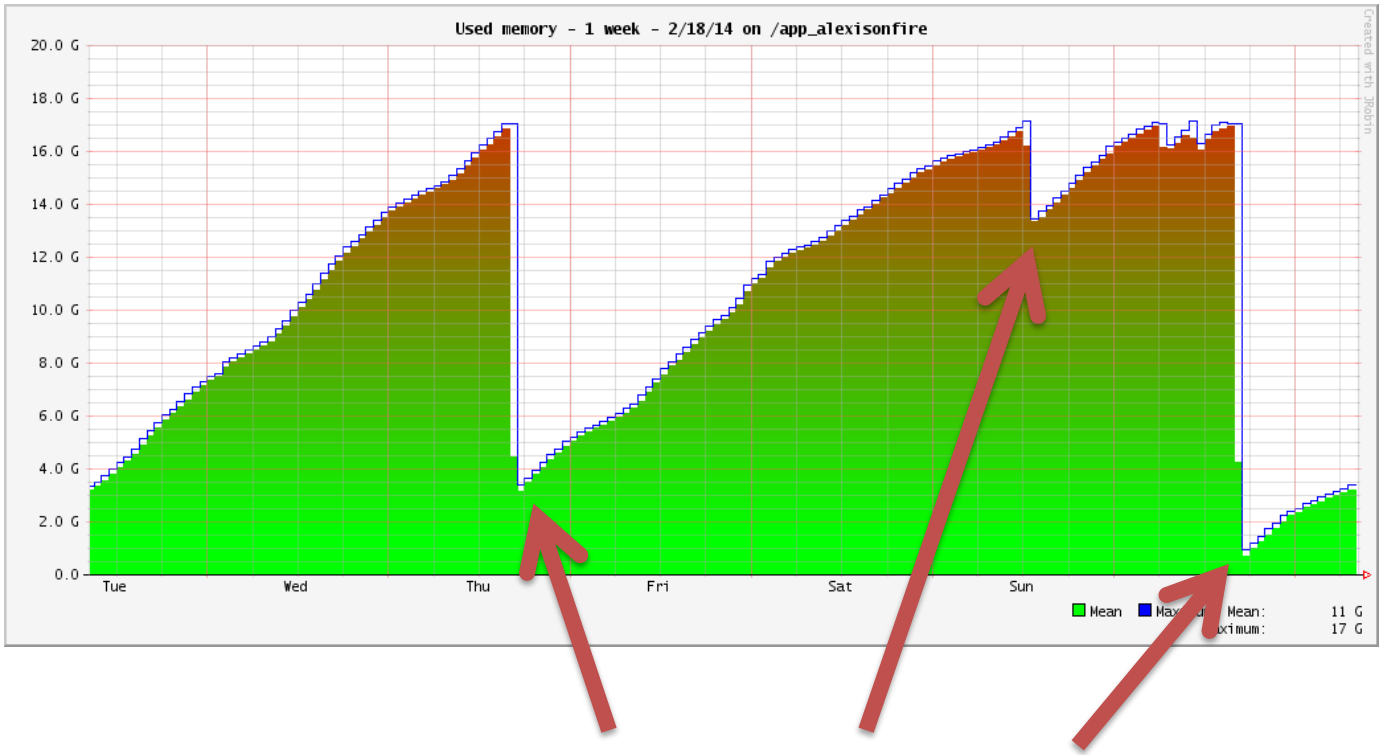
Le Garbage Collector

Passage du GC



Profiling de la mémoire

Exemple de comportement mémoire d'une application Java



Garbage collection

Où est la fuite mémoire ?

```
public class Stack {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack(){
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(Object e){
        ensureCapacity();
        elements[size++] = e;
    }

    public Object pop(){
        if(size == 0)
                throw new EmptyStackException();
        return elements[--size];
    }

    private void ensureCapacity(){
        if(elements.length == size)
            elements =  Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Les références ne sont pas éliminées

Solution

```
public Object pop(){
    if(size == 0)
            throw new EmptyStackException();
    Object result = elements[--size];
    elements[size] = null;
    return result;
}
```