JAVA Virtual Machine

Contents

Contents (Real)

01.

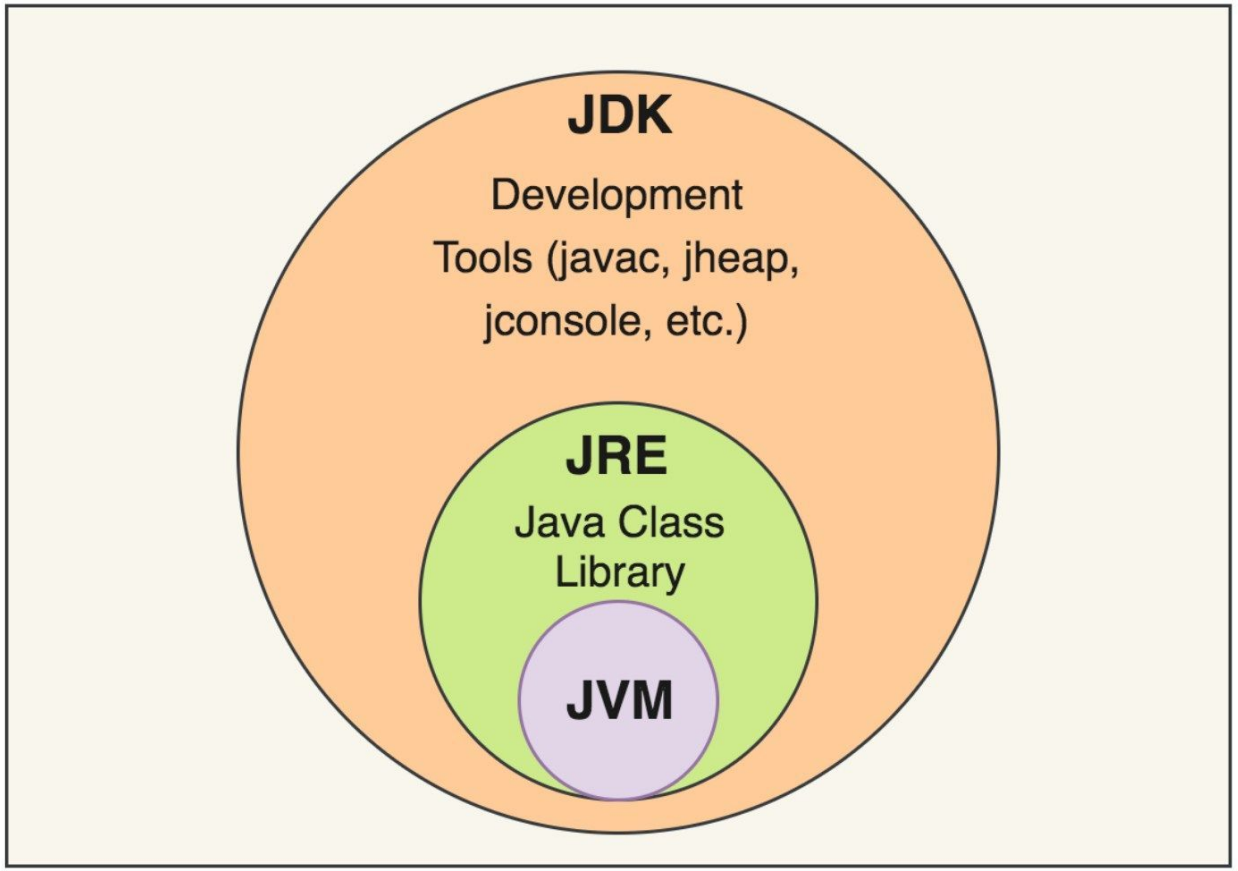
Structure & Terminology

Key Feature of JAVA

1. Platform Independent.
2. Write Once Run Anywhere
3. Backward Compatibility
4. Concurrency Support for Multi-threading

✔ JAVA는 강력한 호환성과 이식성을 겸비한 견고한 언어

JDK, JRE, JVM


JDK
Development
Tools (javac, jheap,
jconsole, etc.)
JRE
Java Class
Library
JVM

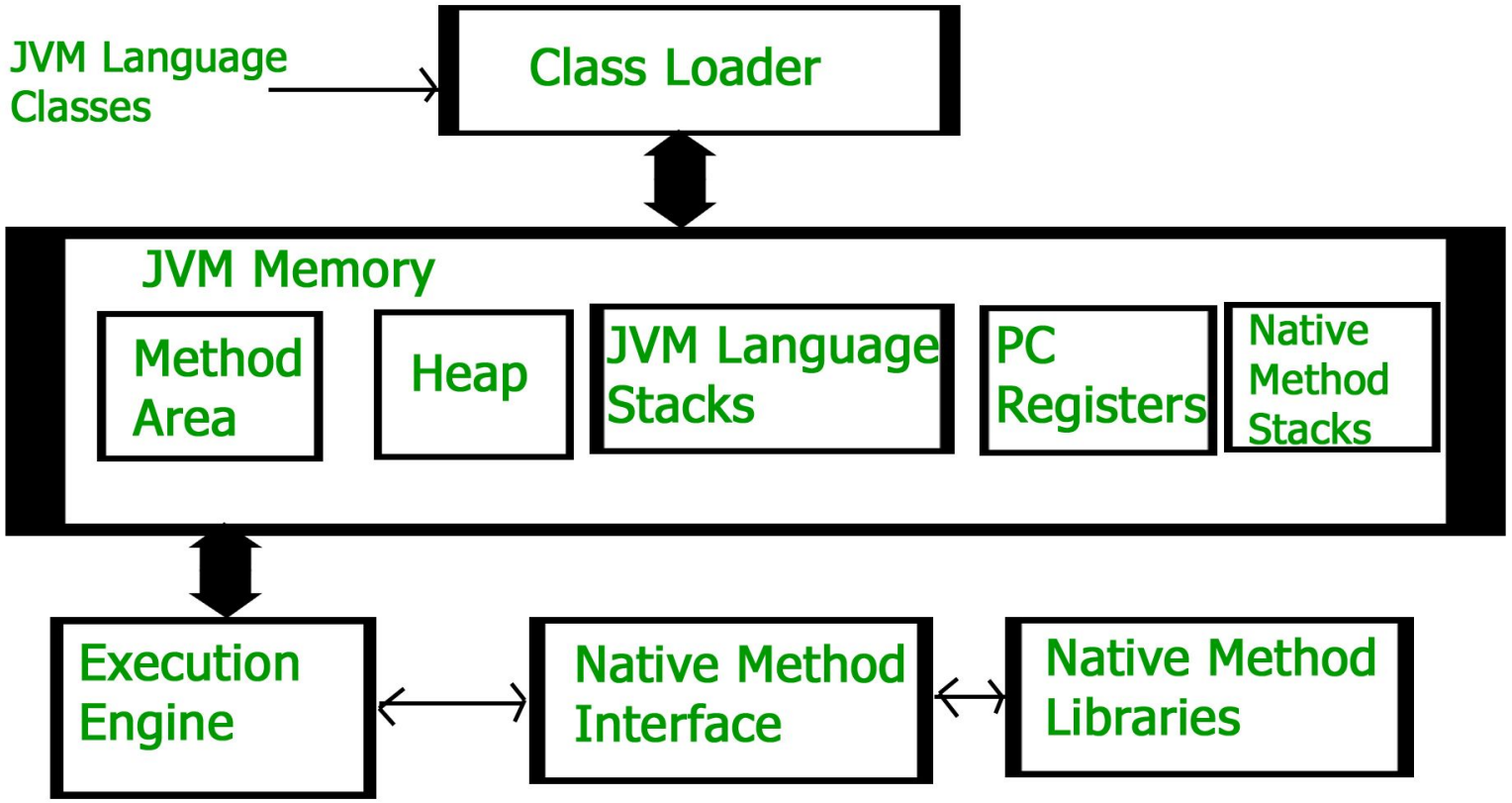
Terminology

Question : JDK? JRE? JVM?

Answer :

- JDK는 Java 프로그램의 개발을 위한 소프트웨어 개발 키트(SDK)
- JRE는 Java 프로그램의 실행하기 위한 환경을 제공하는 소프트웨어 패키지
- JVM은 Java 프로그램의 실행을 위한 가상 머신 (Virtual Machine)

Basic Structure - JVM



Terminology

Question : JAVA Virtual Machine?

Answer :

- **Java Bytecode**로 컴파일 되어 있는 프로그램을 실행하기 위한 가상 머신
- **Java** 뿐만 아니라 **Kotlin, Groovy, Scala, Clojure** 와 같이 **Java Bytecode**로 컴파일 된 프로그램을 실행 가능

Terminology

Question : Virtual Machine?

Answer :

- 컴퓨팅 환경을 구현한 소프트웨어 기반의 Emulator.
- JVM은 Java Bytecode 실행에 특화된 Virtual Machine
- OS와 Java 프로그램 사이에 추상화 된 계층 (Abstracted Layer)을 제공한다.

Terminology

Question : Native Methods?

Answer :

- **Java가 아닌 다른 언어로 작성된 Method들**
- **JVM 자체는 low level language인 C 와 C++로 구현 되어 있다.**

Terminology

Question : PC Registers?

Answer :

- **PC는 Program Counter의 약자**
- **실행할 명령어를 담고있는 Register**
- **전통적으로 Register는 CPU내의 메모리 공간을 뜻함.**

Basic Structure

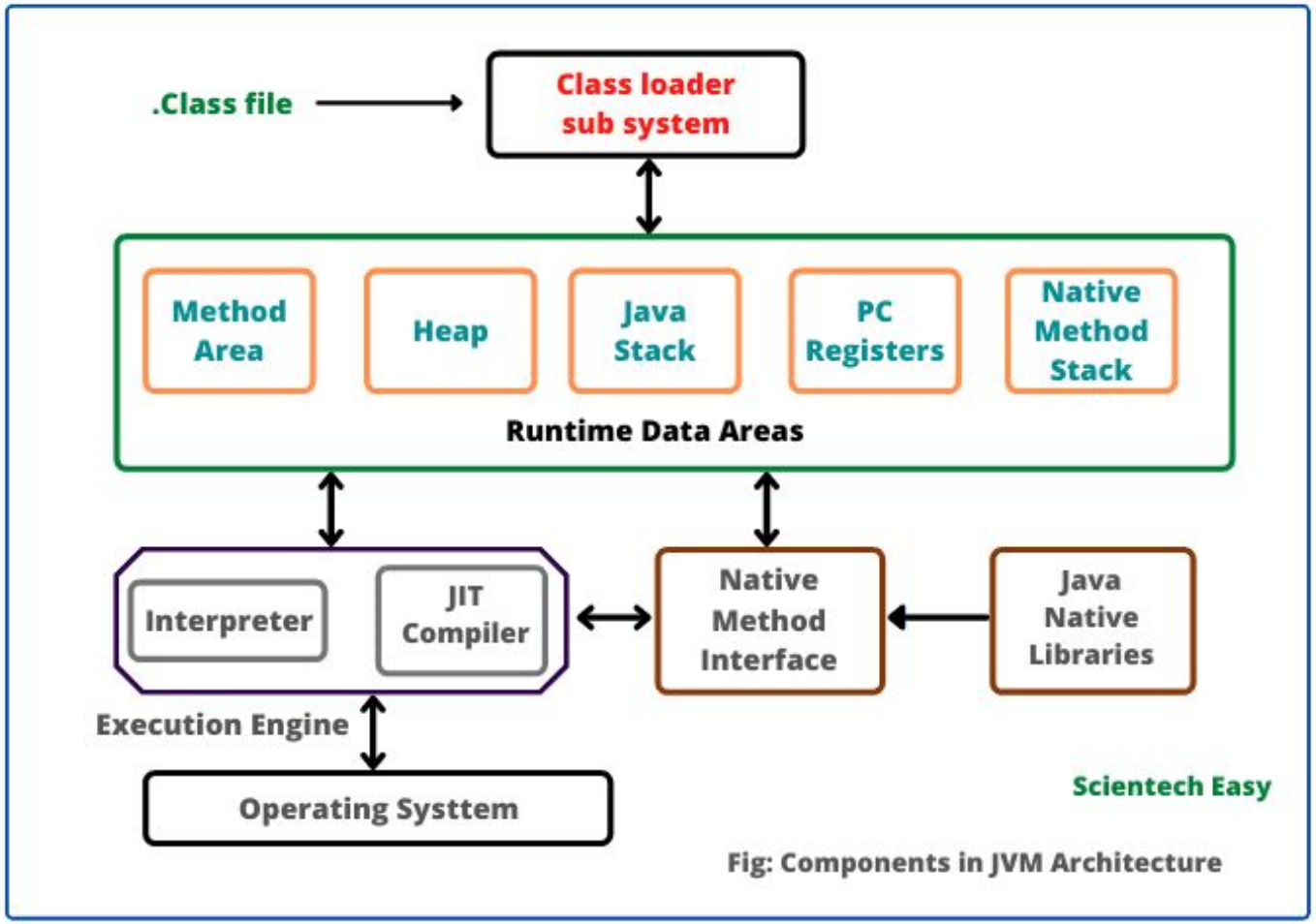
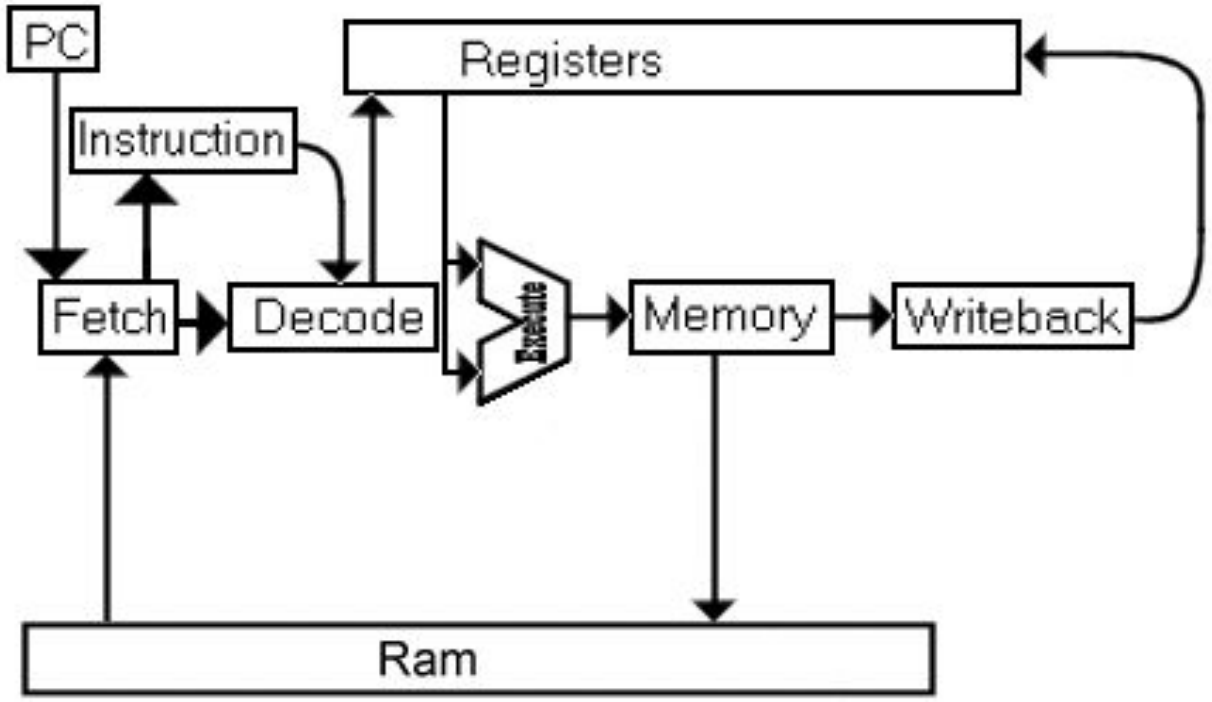


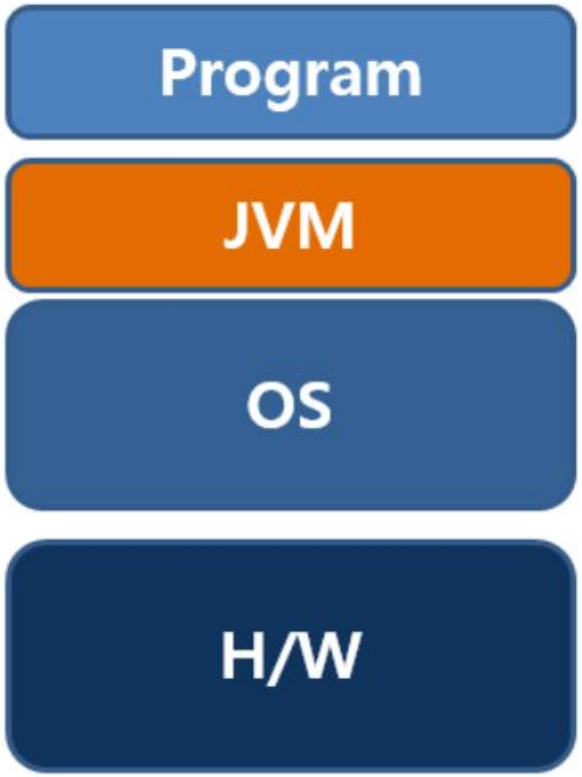
Fig: Components in JVM Architecture

How CPU Process Instruction



Basic Structure

< 일반적인 프로그램 >



< JAVA 프로그램 >

Structure

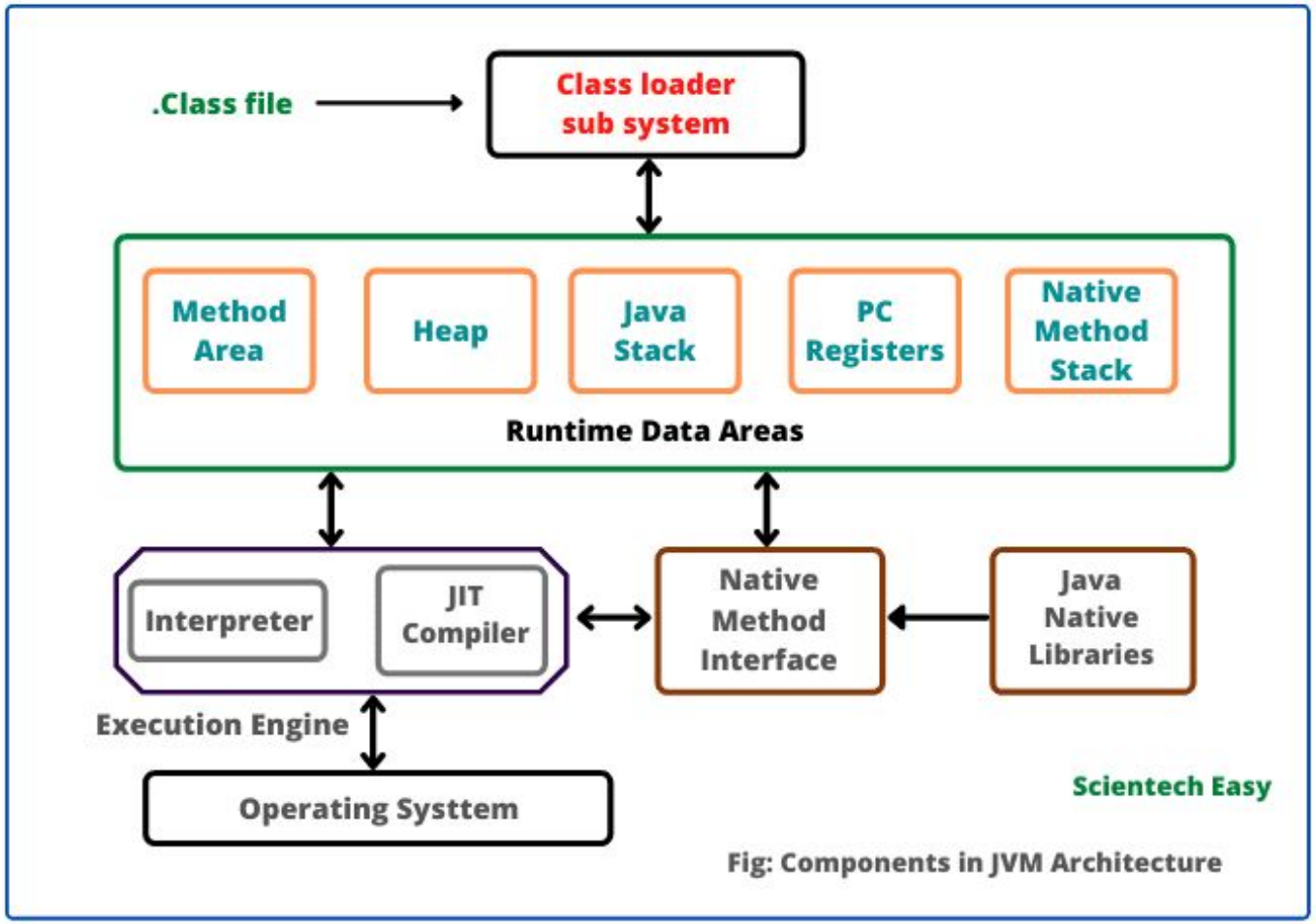


Fig: Components in JVM Architecture

Detail

1. **Loading**
2. **Linking & Verification**
3. **Memory Allocation**
4. **Execution**
5. **Garbage Collection**

6. Thread Management
7. Exception Handling
8. Security Management
9. Native Interface(JNI)
10. Monitoring Management
11. Termination

02.

Class Loader

Class Loading

- 실행에 필요한 Class file들을 시스템, 네트워크 등의 Source로부터 불러오는 과정

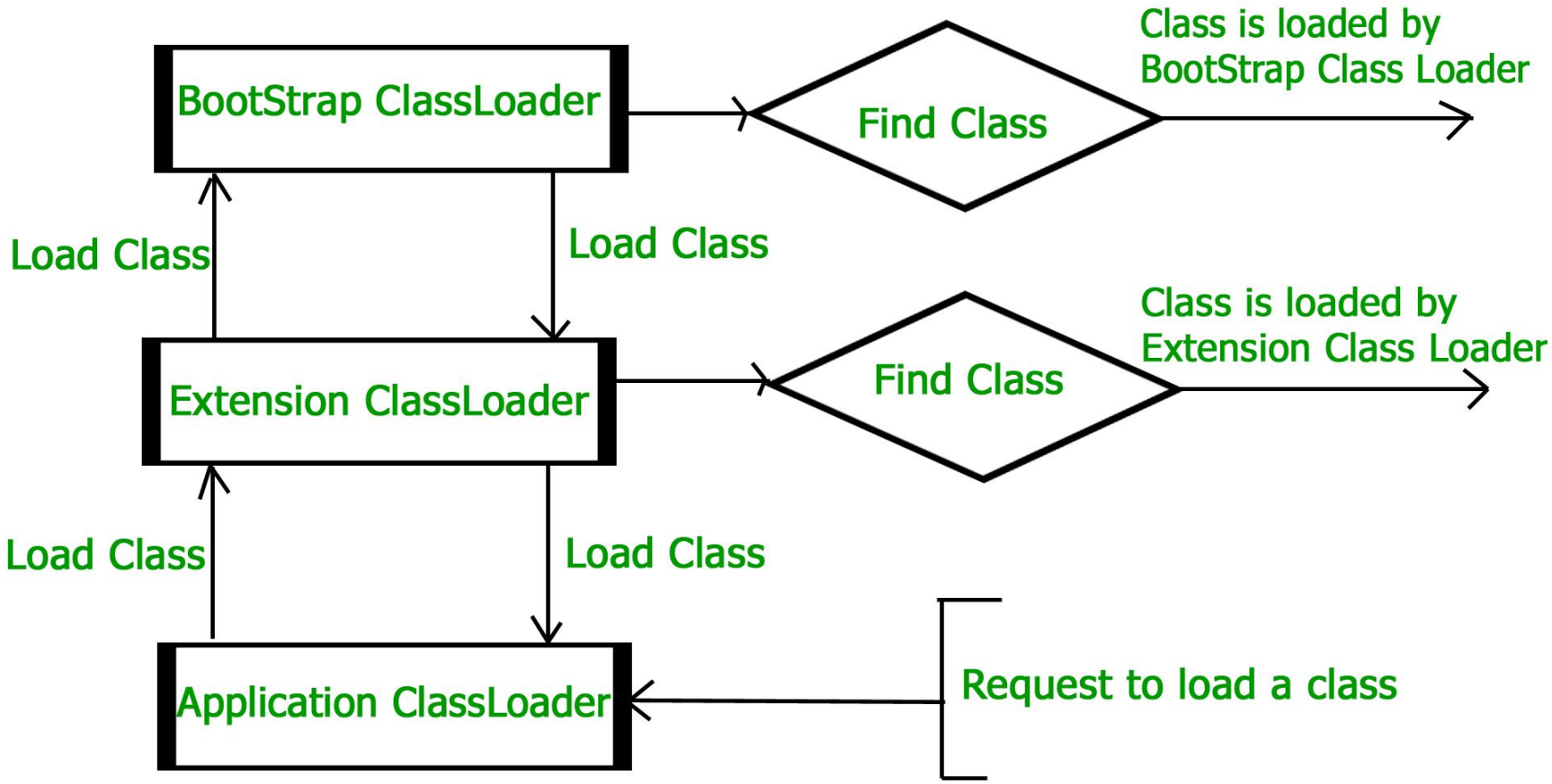
- Class Loader에 의해 실행되며 3 종류의 Class Loader가 존재한다.

  ✔ Bootstrap Class Loader

  ✔ Extension Class Loader

  ✔ Application Class Loader (System Class Loader)

Workflow – Class Loader



Bootstrap Class Loader

- 모든 Class Loader Instance의 부모
- Native 코드로 작성되어있음
- JRE 내부의 Core Class를 Load 할 책임이 있음
  - java.lang.*
  - java.io.*
  - Java.awt.*
  - etc.

Extension Class Loader

- Class가 Java Core Class의 Extension 여부는 Extension Class Loader에서의 load 결정지음 (lib/ext)

- JDK Extension Library 내부의 Java Core Class의 Extension인 클래스를 load할 책임을 가짐

  - JDBC (Java Data Base Connectivity)
  - JCE (Java Cryptography Extension) – 암호화
  - etc.

Application Class Loader (System Class Loader)

- 프로그램의 Classpath내의 Class를 load할 책임을 가짐
  - Windows 환경에서의 환경변수 CLASSPATH
  - IntelliJ 에서의 Content Root
  - Eclipse 에서의 Library, User library
  - etc.

Class Loaders - example

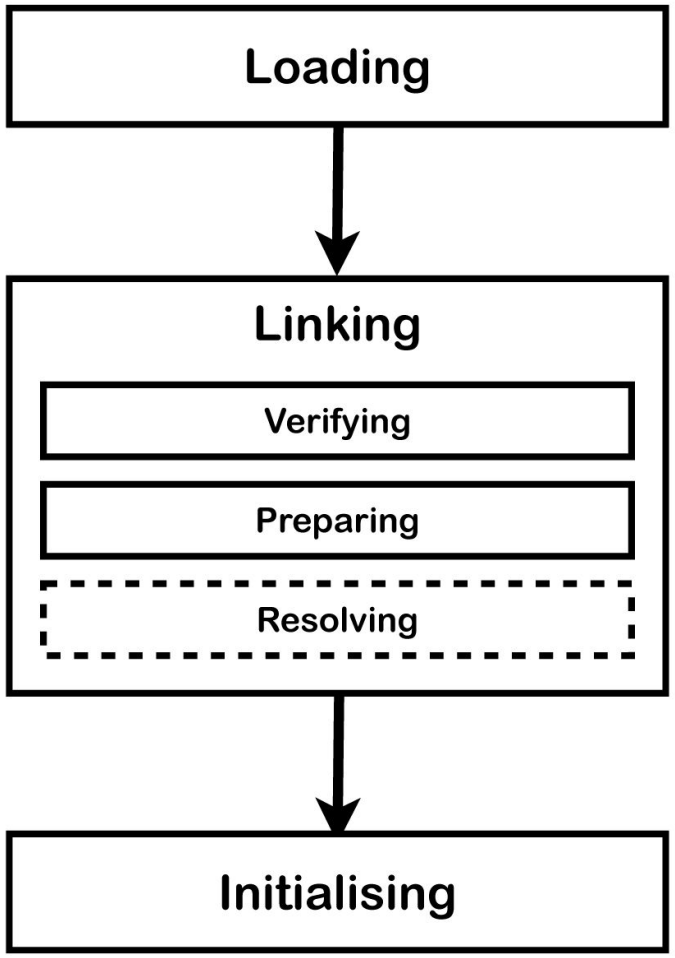
```
public void printClassLoaders() throws ClassNotFoundException {

    System.out.println("Classloader of this class:"
        + PrintClassLoader.class.getClassLoader());

    System.out.println("Classloader of DriverManager:"
        + DriverManager.class.getClassLoader());

    System.out.println("Classloader of ArrayList:"
        + ArrayList.class.getClassLoader());
}
```

```
Classloader of this class:jdk.internal.loader.ClassLoaders$AppClassLoader@73d16e93
Classloader of DriverManager:jdk.internal.loader.ClassLoaders$PlatformClassLoader@1fb700ee
Classloader of ArrayList:null
```

03.

Linking
Verification
Execution

Linking – workflow



Linking – workflow

Linking은 세가지 과정으로 분류된다.

- Verification (검증)
- Preparation (준비)
- Resolution (해결)

Verification

- 클래스 파일의 구조와 내용을 검사하여 유효성을 확인.
- 클래스 파일이 Java 언어 사양에 맞게 올바르게 작성되었는지 검증.
- 메서드 호출이나 필드 접근이 유효한지, 스택 사용이 안전한지 등을 확인합니다.
- 검증이 실패하면 예외가 발생하여 프로그램이 종료될 수 있습니다.

Preparation

- 클래스에 대한 정적 변수(static variable)들을 위한 메모리 공간을 할당하고 초기화.

- 이 단계에서는 정적 변수들에 할당된 메모리 공간만을 준비하며, 실제 초기화는 이후에 진행.

Resolution

- 클래스나 인터페이스에 대한 다른 참조들을 실제 메모리 상의 참조로 교체.

- 이 과정에서는 클래스나 인터페이스의 이름 실제 메모리 상의 주소로 매핑.

Execution

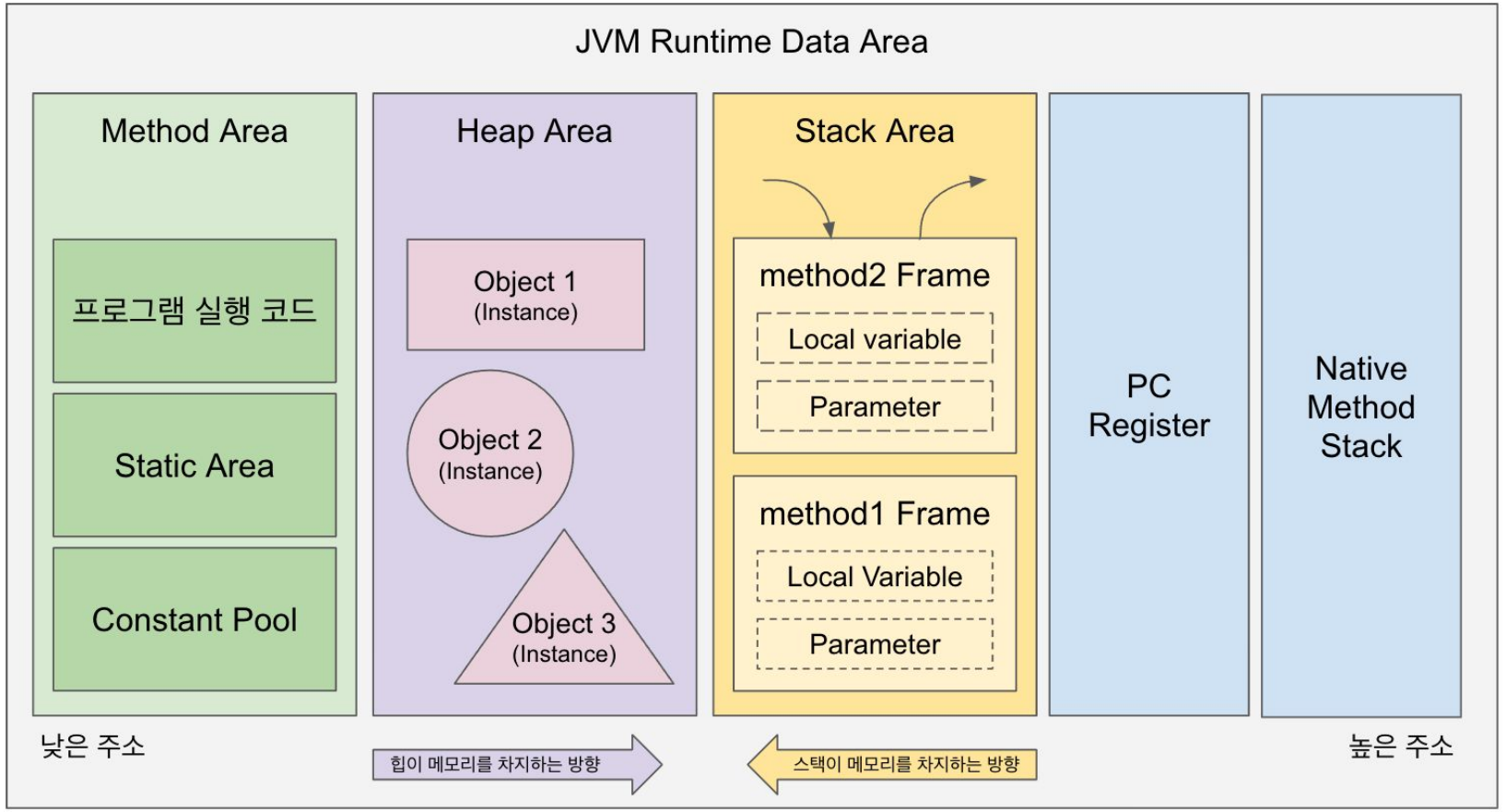
- JVM내의 Interpreter는 각 Bytecode를 하나씩 해석하여 실행함

- 반면 JIT (Just In Time) 컴파일러는 반복되는 코드를 기계어 (Machine Language)로 변환

04.

JVM Memory Structure

Basic Memory Structure



Basic Memory Structure

- Method Area
- Heap Area
- Stack Area
- PC Register
- Native Method Stack

Method Area

- Class의 metadata와 static 변수들이 저장되는 곳
- 크게 다음과 같은 Data가 저장된다
  1. 상수(Constant)들을 상수 풀 (Constant Pool)
  2. 멤버 변수 및 생성자(Constructor) 와 Method
  3. 클래스 변수(Static)

Stack Area

Method 호출 시의 Method의 Frame이 저장되는 영역

- **Method 호출 시 Stack frame을 형성하며 다음과 같은 내용을 담는다.**

  ✔ **Method 호출과 관계되는 지역변수 및 매개변수**

- **LIFO 구조를 가지며 Method의 호출 / 종료에 따라 각각의 Stack Frame이 push / pop 하여 동작한다.**

PC Register – Program Counter Register

- JVM은 Virtual Machine이므로 현재 실행중인 Instruction에 대한 정보를 Register에 담는다.
- 실제 CPU 상의 Physical Register가 아닌 논리적인 Register
- 실제 각 Instruction에 대한 Execution은 CPU에서 진행된다.

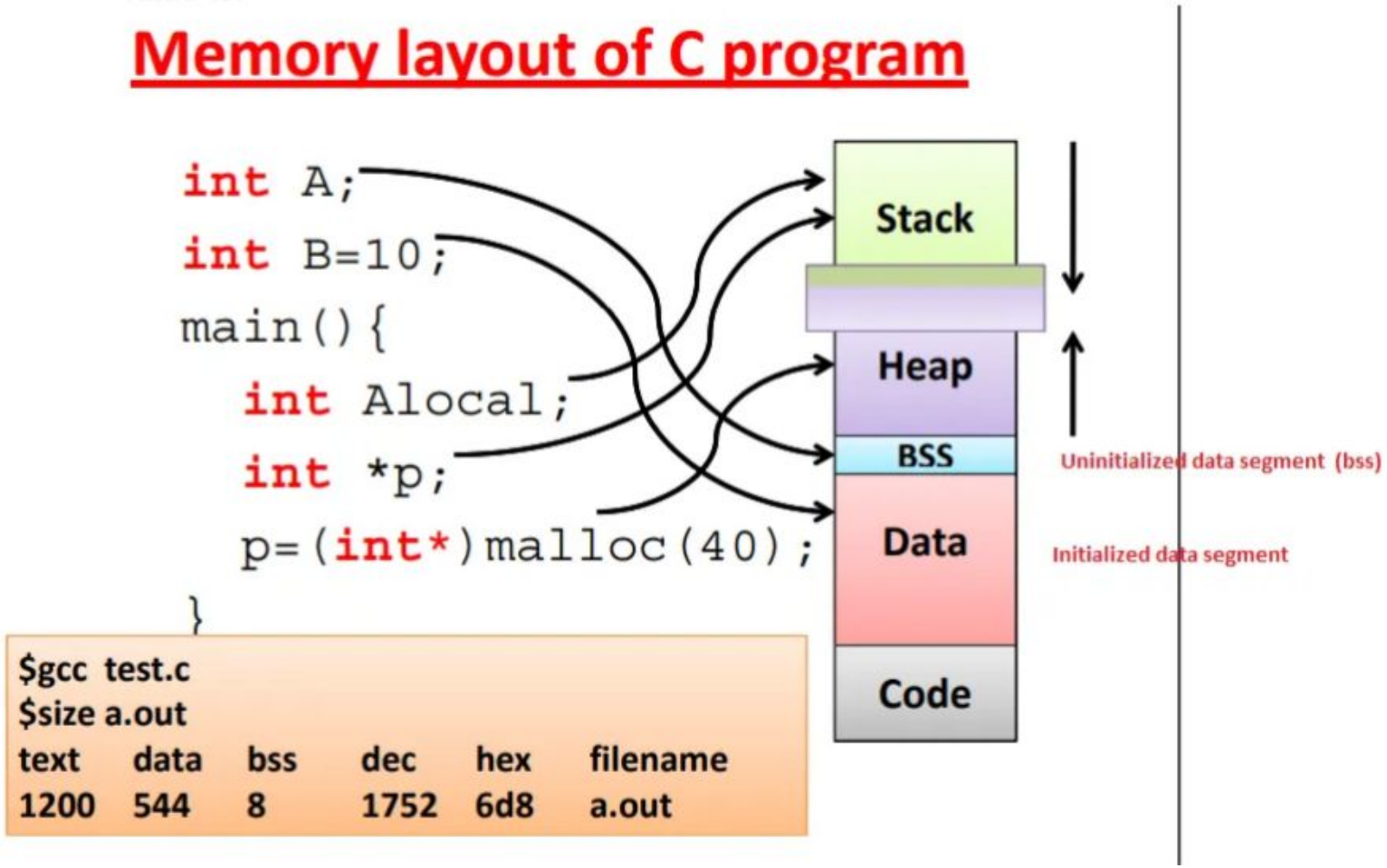
Native Method Stack

- JAVA 이외의 언어로 작성된 Native Code를 위한 Stack
- Java Native Interface를 통해 호출된다.
- C, C++ 코드를 실행하기 위한 스택
- Native Method의 Parameter, Local Variable등을 Bytecode로 저장

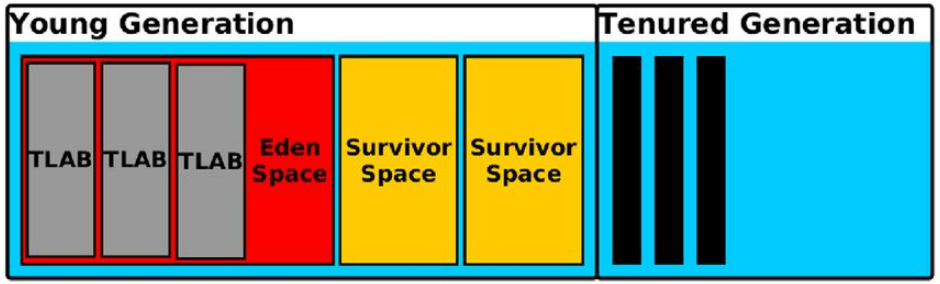
HEAP

- 전통적인 의미로 동적으로 메모리가 할당된 객체가 저장되는 영역
- **Java에서는 Runtime에 할당되는 모든 Class Instance Variable이 저장되는 영역**
- **new()를 통해 생성되는 모든 객체가 해당되며 JAVA Memory Management의 핵심적인 대상**

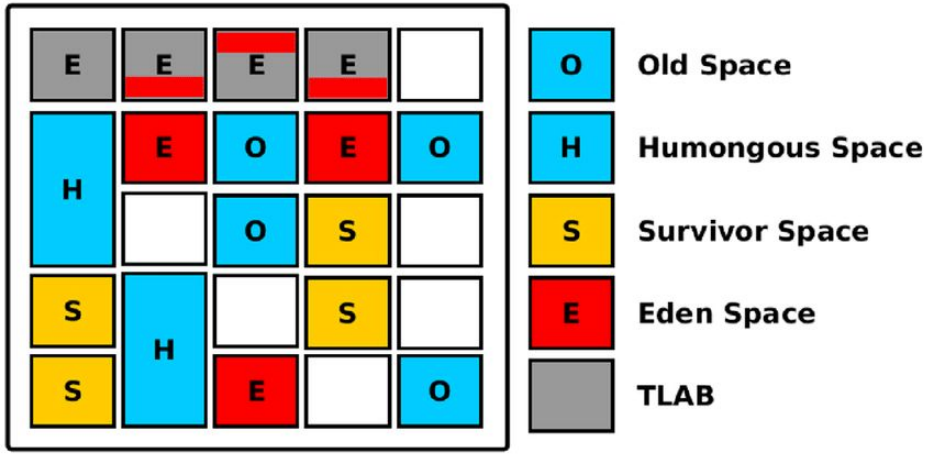
Memory Allocation in C Language



HEAP



(a) Typical SerialGC Generational Heap



(b) Typical G1GC Generational Heap

05.

Garbage Collection

Goal of Garbage Collection

- **Heap Memory에서 Runtime에 동적으로 할당되는 메모리의 관리**
- **더 이상 참조되지 않는 객체에 소모되는 메모리 자원을 회수**
- **메모리 누수(Memory Leak) 및 단편화(Fragmentation) 방지**

Heap Area

- **Young 영역 (Minor GC)**
  - 상당수의 객체가 생성이후 곧 참조되지 않는다는 점을 이용
  - 새로 생성되거나 생성 되고 시간이 얼마 지나지 않은 객체들의 영역
  - **Eden, Survived** 등의 영역이 존재
- **Old 영역 (Major GC)**
  - **Young** 영역에서 일정 횟수 이상 살아남은 객체들이 이동되는 영역

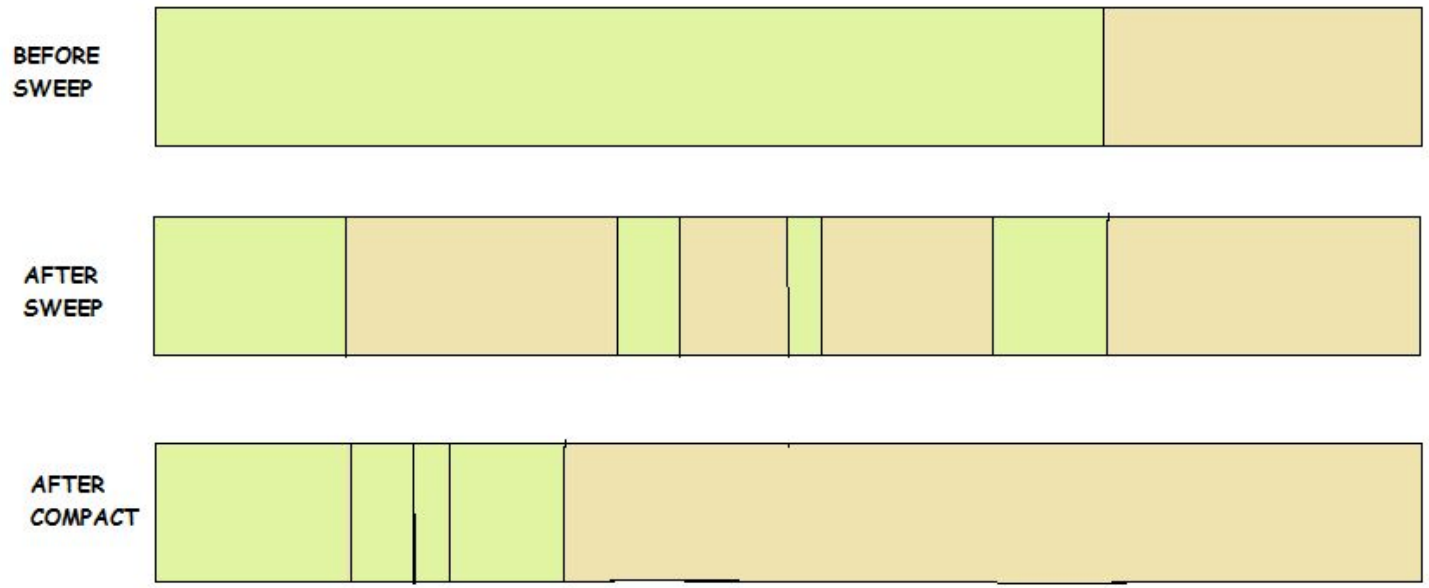
Common Process

- **Mark and Sweep**
  - **Memory 영역을 순회하며 참조 가능한 객체를 표기 (Mark)**
  - **Mark 단계에서 표기되지 않은 참조 불가능한 객체를 제거**

- **Compact**
  - 메모리 단편화를 줄이기 위해 살아남은 객체의 메모리상 위치를 재조정하는 과정

- **Stop the world**
  - 전체 **Thread**의 동작을 멈추고 메모리를 회수하고 정리하는 과정

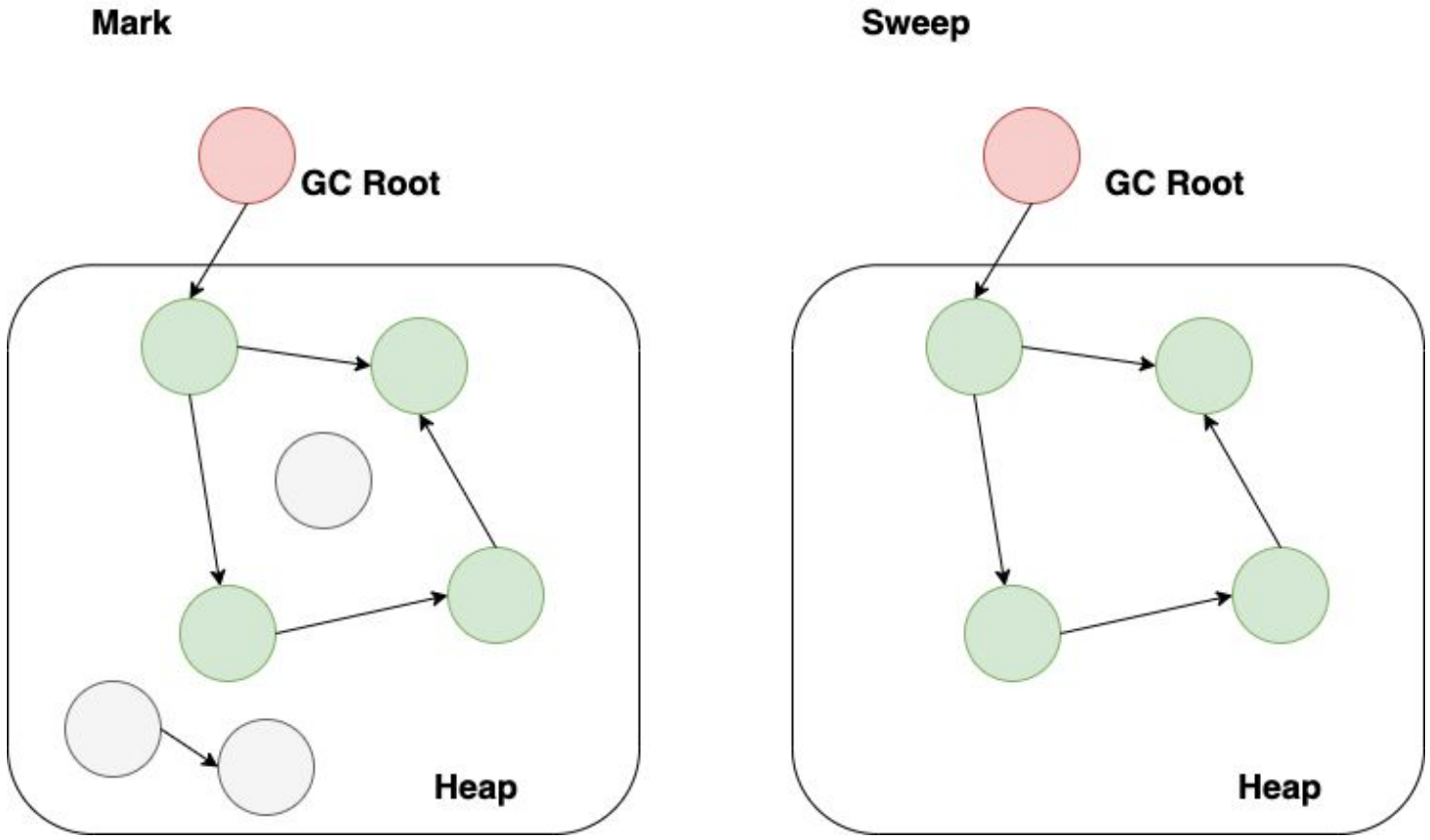
Mark and Sweep & Compact



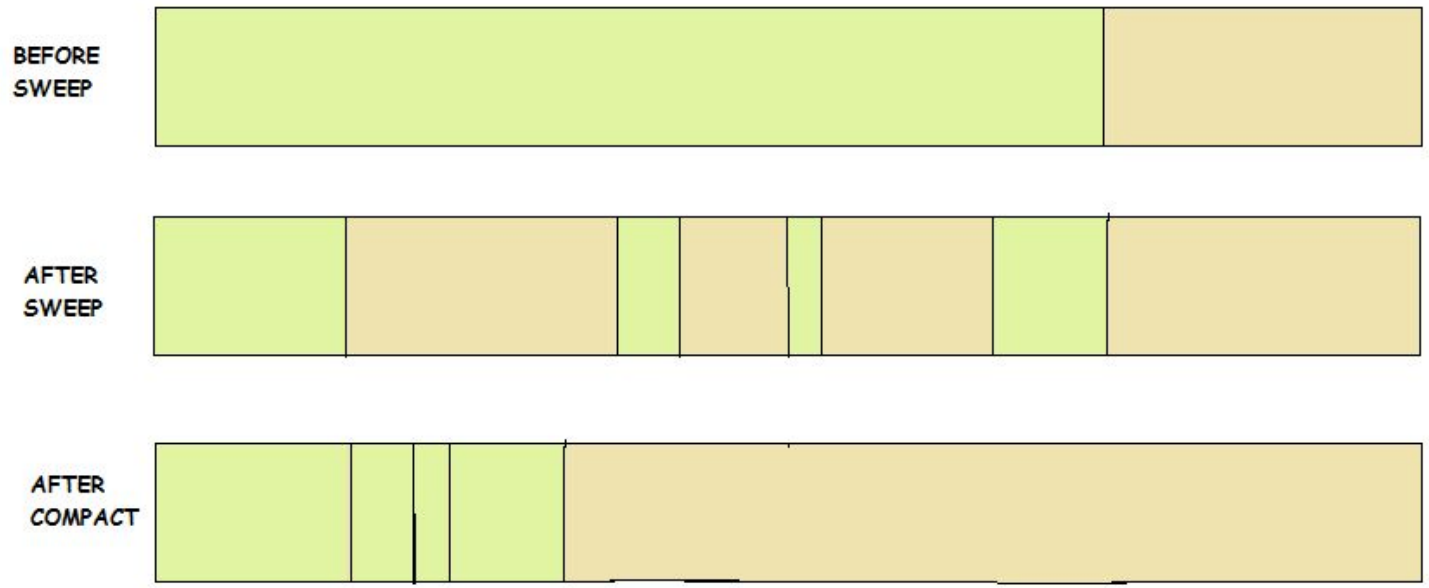
MARK - SWEEP - COMPACT PHASES MEMORY REPRESENTATION

OPENGENUS

Mark and Sweep & Compact



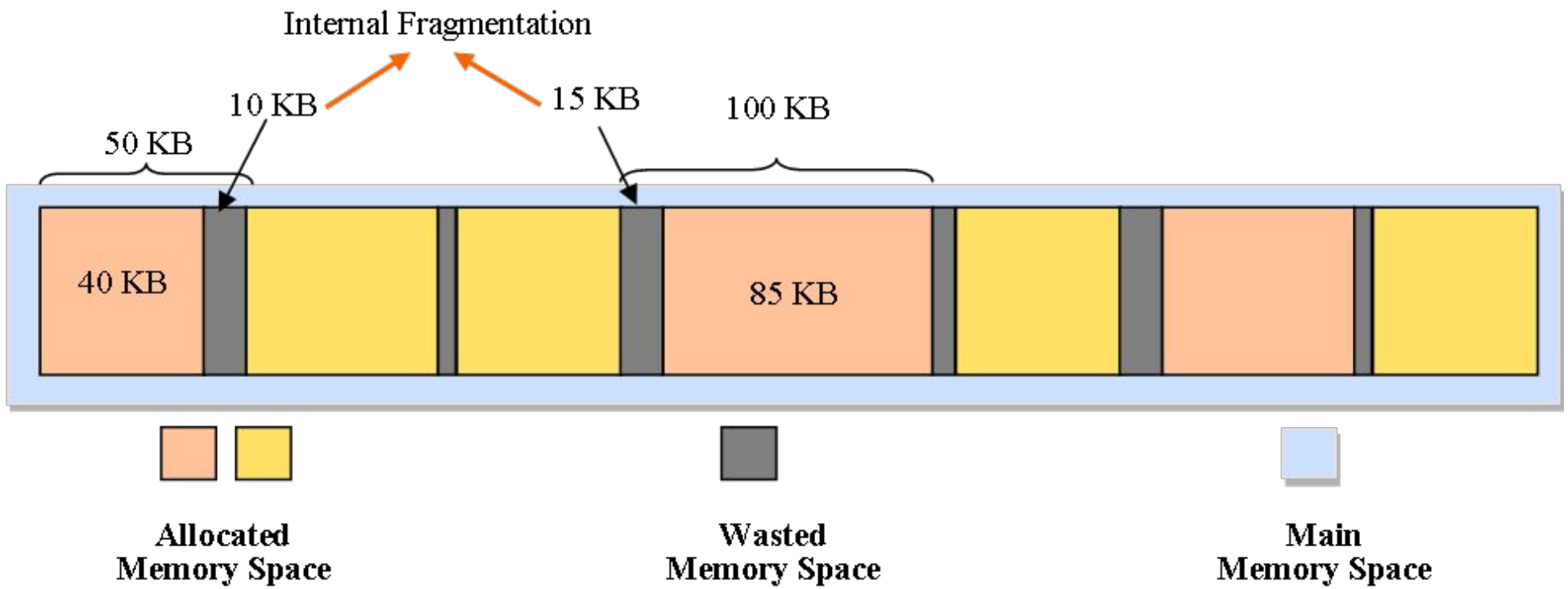
Mark and Sweep & Compact



MARK - SWEEP - COMPACT PHASES MEMORY REPRESENTATION

OPENGENUS

Fragmentation

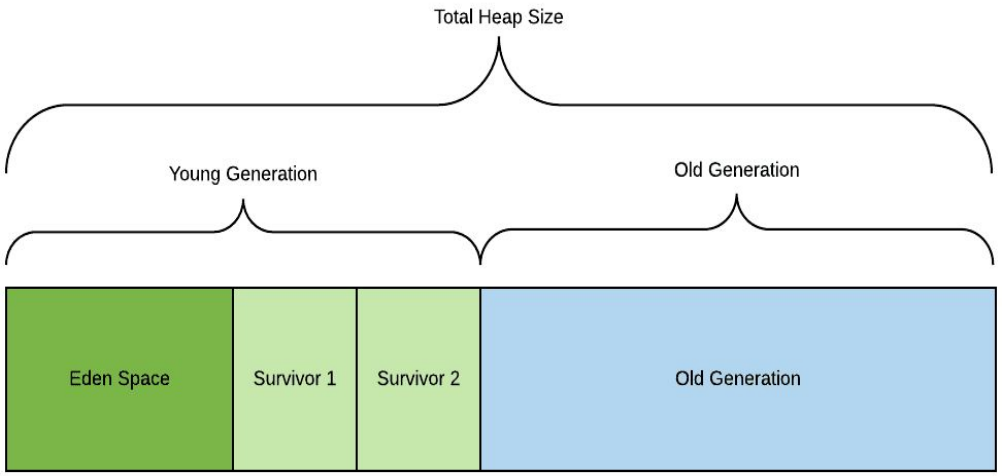


Minor GC & Major GC

- Minor GC
  - Young 영역이 가득 찼을 경우에 발생
  - Young 영역에 대해서 Garbage Collection 시행
  - STW(Stop the World)가 매우 짧게 발생한다.

- Major GC
  - Old 영역이 가득 찼을 경우에 발생
  - 전체 Heap을 대상으로 Garbage Collection이 시행
  - STW가 상대적으로 길게 발생하여 성능을 저하시킴

Old Garbage Collection



- GC 발생시 해당 영역 전체에 적용되어 비 효율적
- 특정 Size의 메모리 단위가 아닌 구역 단위의 GC
- 긴 STW를 발생시켜 JAVA 의 성능 저하에 일조

G1GC (From JDK 8)

- Region이라는 특성 크기의 Memory Chunk 단위로 관리
- 구역 단위(Young, Old 등)가 아닌 특정 Size의 Region 단위로 관리
- 효과적인 STW 관리 및 병렬처리 가능

Mixed GC

- Mixed GC
  - ✔ Young / Survivor 영역과 Old Region의 일부 GC
  - ✔ -XX:G1MixedGCCountTarget 등의 변수 값을 통해 설정

G1GC – Initial Mark

- Root Region을 마킹하는 단계
  - Root란 Heap 외부에서 참조가 가능한 객체를 의미한다.
  - 현재 접근할 수 있는 객체를 통해 접근할 수 없는 객체는 죽은 객체이다.
  - 전체 Heap을 순회하지 않고 빠르게 mark 하기 위함

Extra.

Further Readings

Further Readings

- JVM Execution Engines
- Thread Management
- Legacy GC (Serial GC, CMS GC 등)
- ZGC, Shenandoah GC
- GC Tuning

- CPU, Memory Structure

Fin.

Any Question?