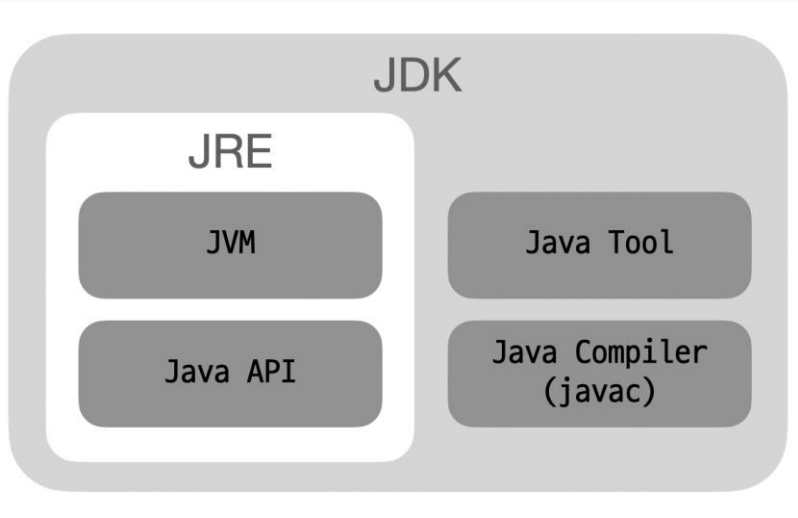
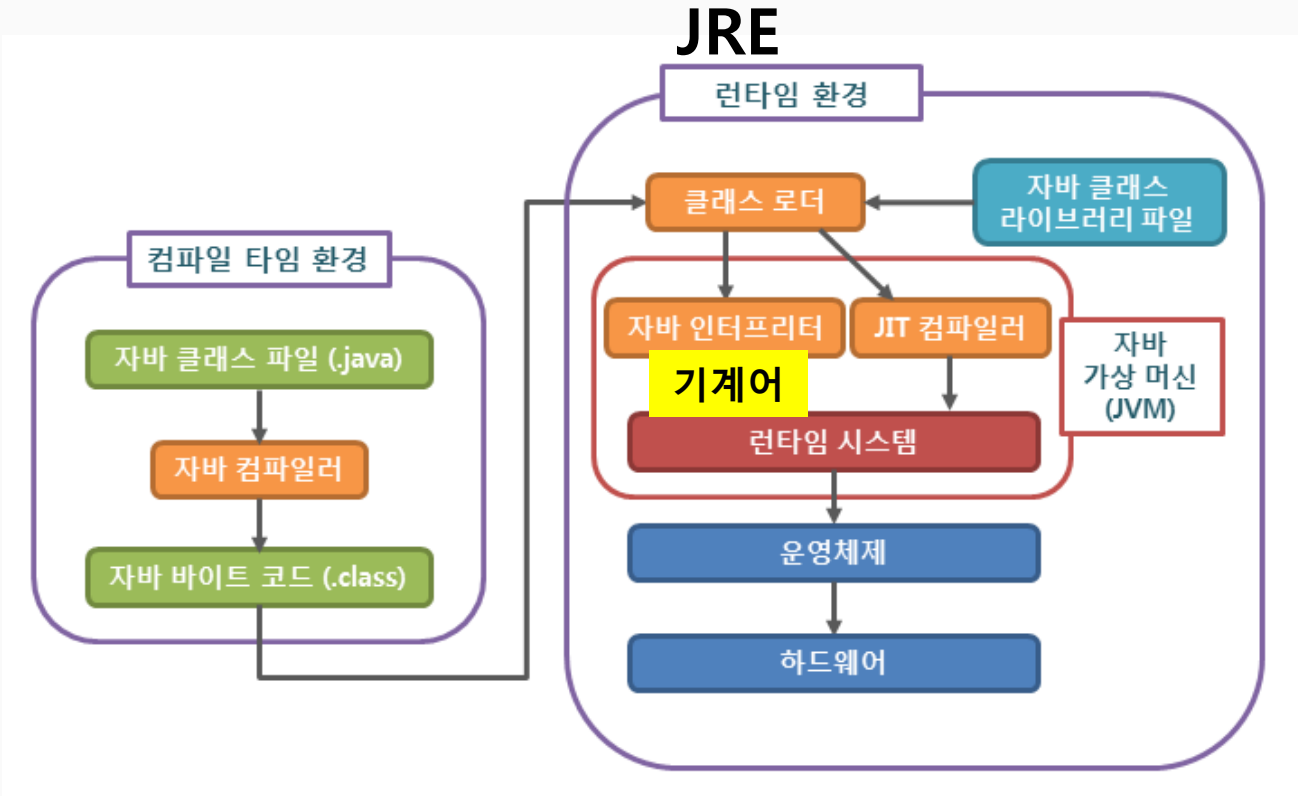
CS study 6주차
JVM 메모리 구조와 가비지 컬렉터😎

자바 바이트코드는 JRE위에서 동작한다.

JRE의 구성요소에는 자바 API와 JVM이 있다.

여기서 **가장 중요한 요소는 JVM**

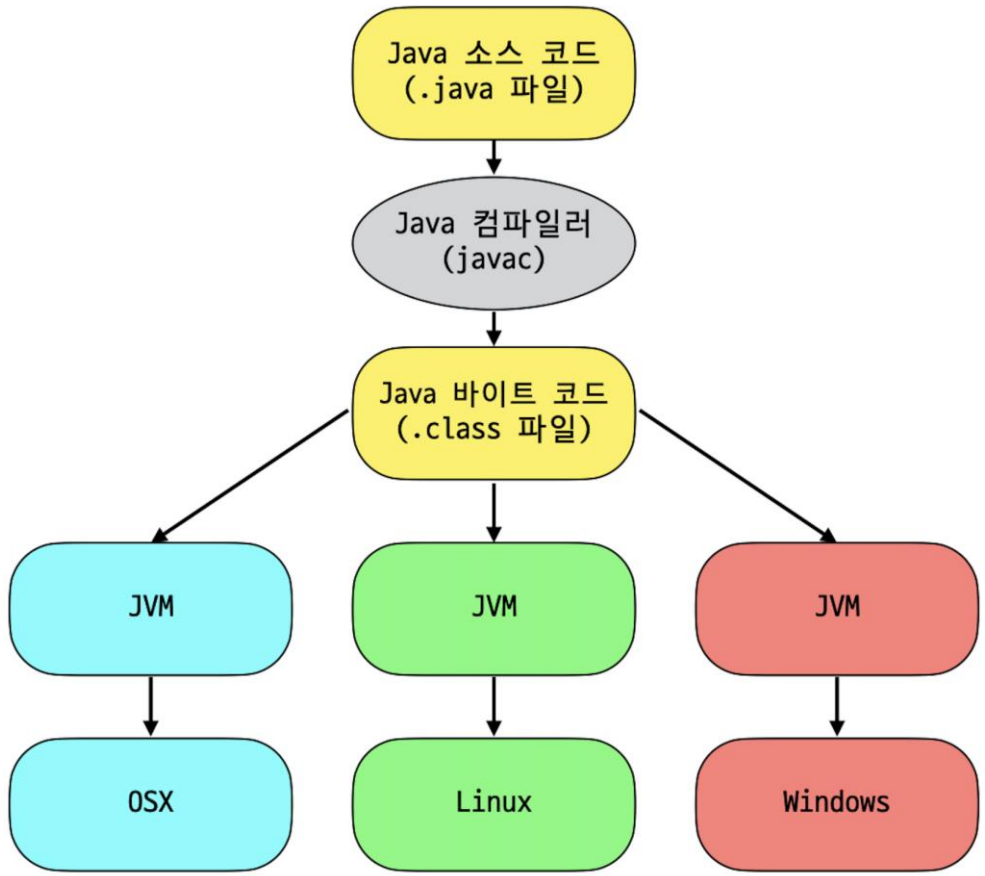




JVM이란? Java Virtual Machine

자바 바이트 코드를 해석하고 실행하는 가상 머신
시스템 메모리를 관리하며 자바 기반 어플리케이션을
위해 이식 가능한 실행 환경을 제공함.

자바 바이트 코드를 실행하고자하는 모든 하드웨어에 JVM을 동작시킴으로서 자바 실행 코드를 변경하지 않고도 모든 종류의 하드웨어에서 동작되게 함

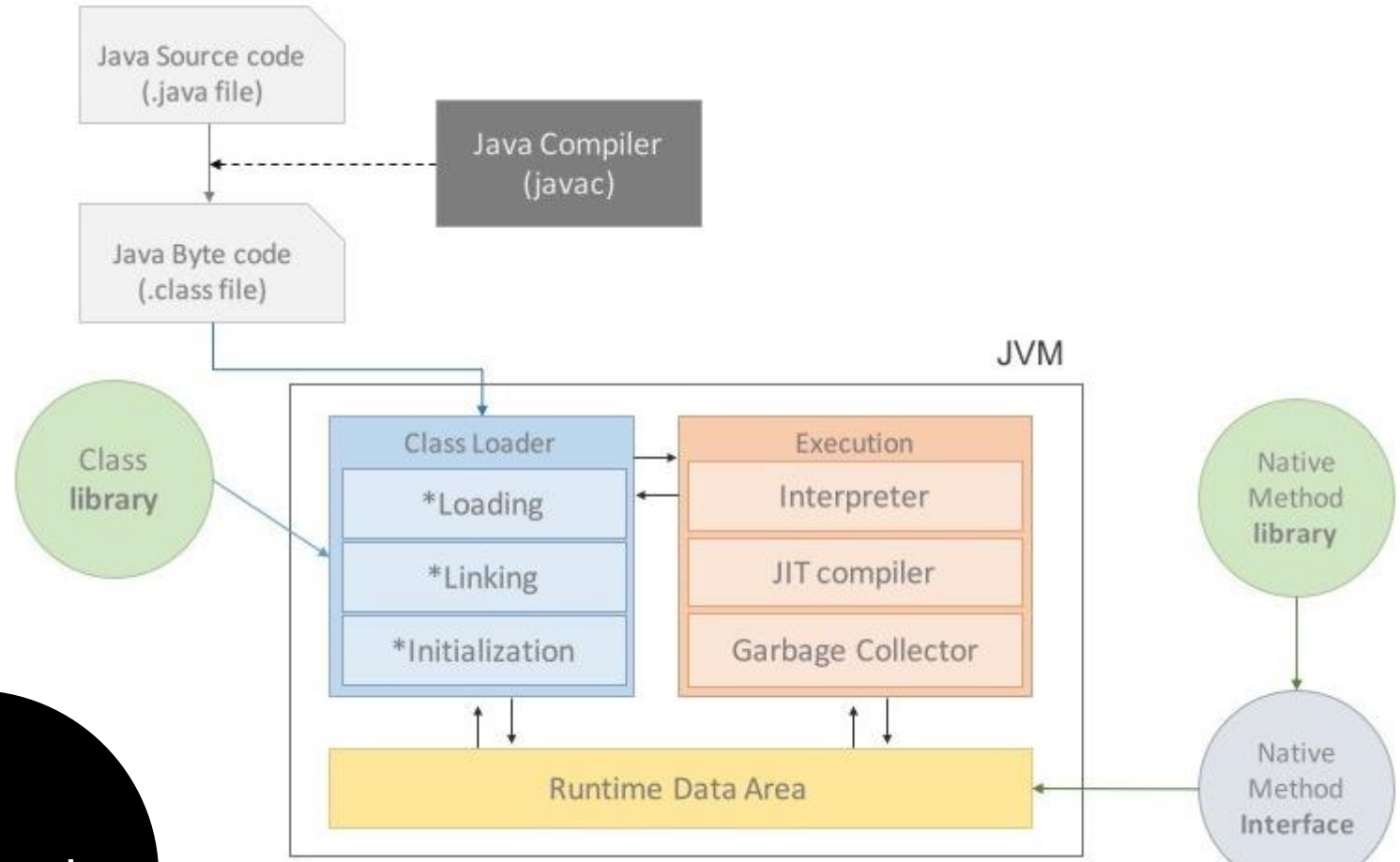


JVM의 구성

Runtime Data Area

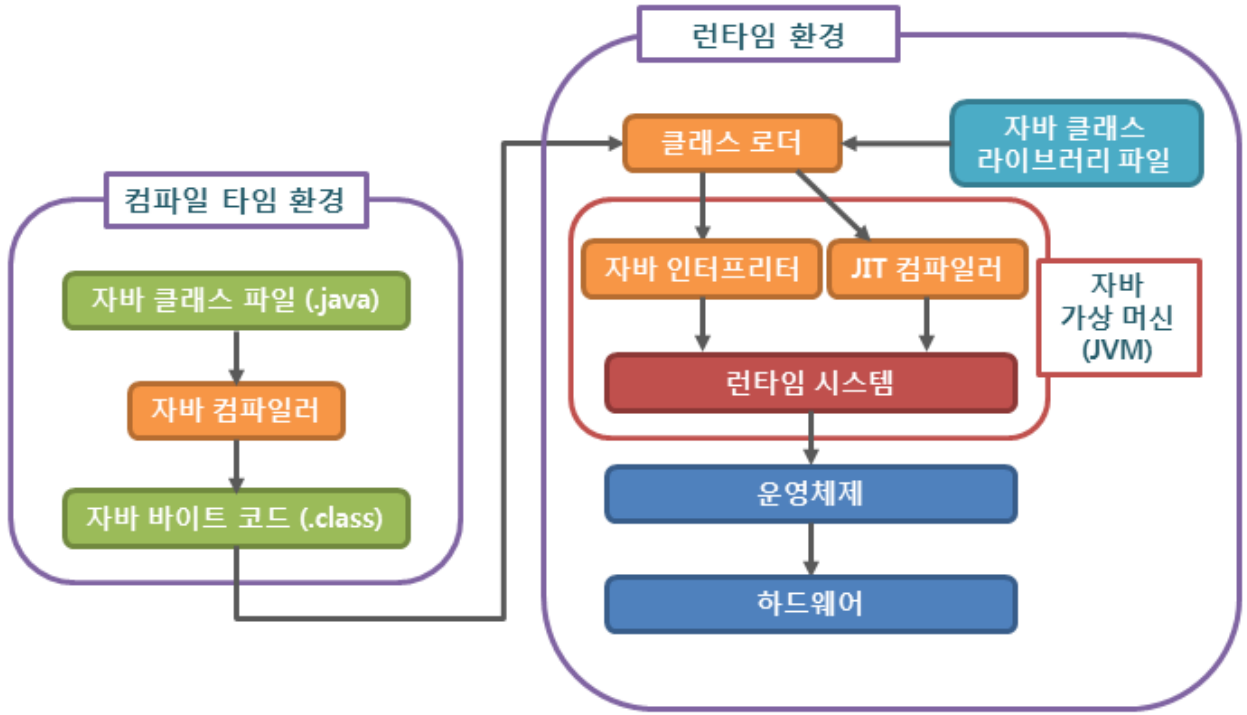
Execution Engine

클래스 로더



클래스 로더

자바 .class 파일들을 JVM으로 로딩해주는 역할



Execution Engine

자바 인터프리터

자바 컴파일러에 의해 변환된 자바 바이트 코드를 읽고 해석하는 역할

JIT 컴파일러

프로그램을 실제 실행하는 시점에 기계어로 변환해 주는 컴파일러
프로그램의 실행 속도를 향상시키기 위해 개발

즉, 자바 컴파일러가 생성한 자바 바이트 코드를 런타임에 바로 기계어로 변환해줌

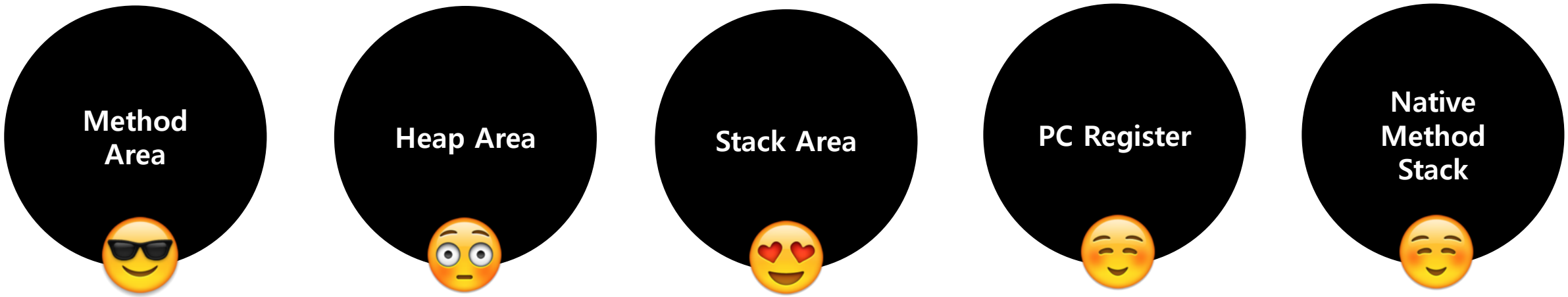
가비지 컬렉터 (Garbage Collector)

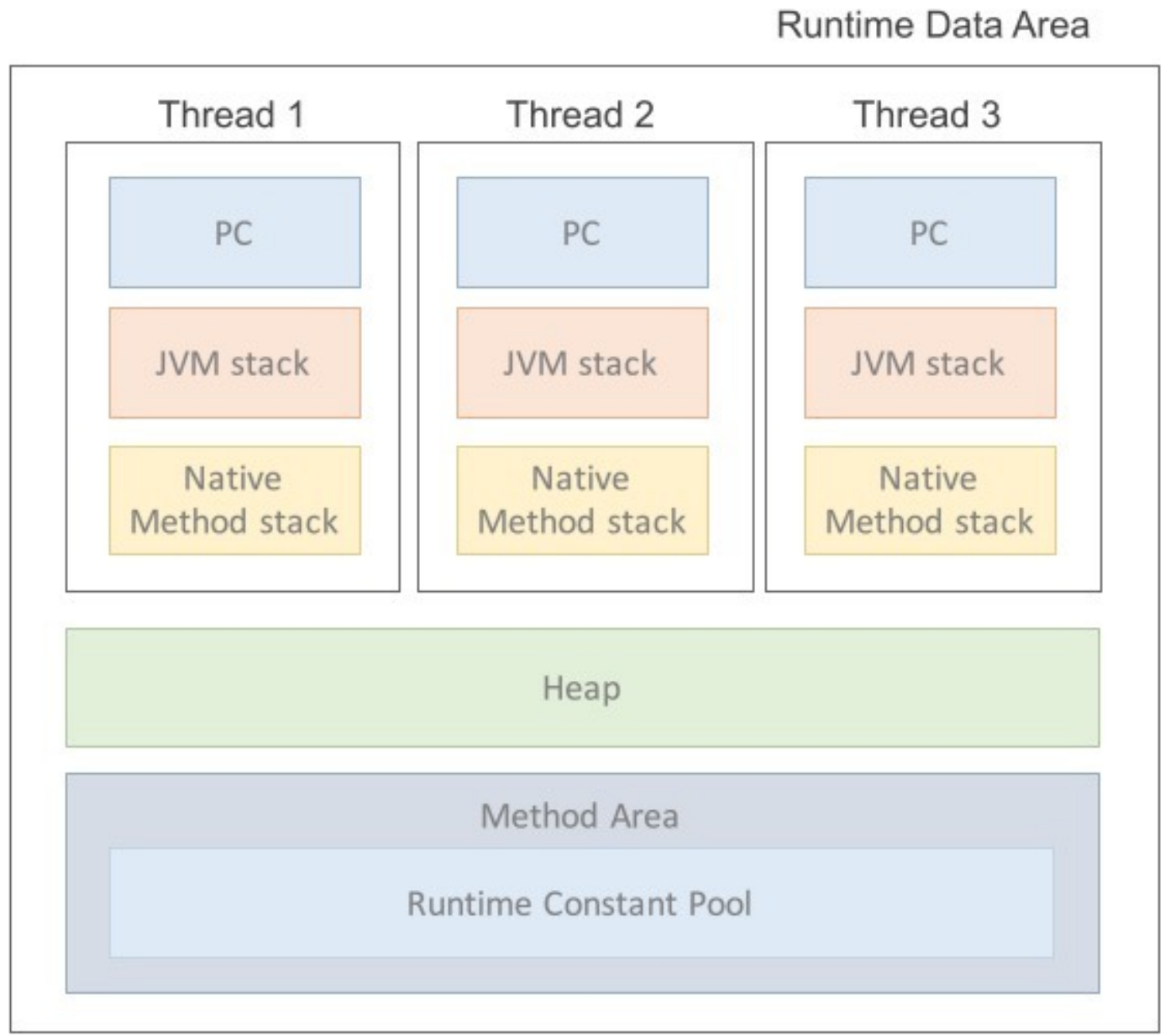
더는 사용하지 않는 메모리를 자동으로 회수해주는 역할
개발자가 따로 메모리를 관리하지 않아도 됨

Runtime Data Area

JVM의 메모리 영역으로 자바 애플리케이션을 실행할 때 사용되는 데이터를 적재하는 영역
해석된 자바 바이트 코드가 이곳에 배치됨

[JVM의 메모리 구조]




Runtime Data Area
Thread 1
Thread 2
Thread 3
PC
PC
PC
JVM stack
JVM stack
JVM stack
Native Method stack
Native Method stack
Native Method stack
Heap
Method Area
Runtime Constant Pool

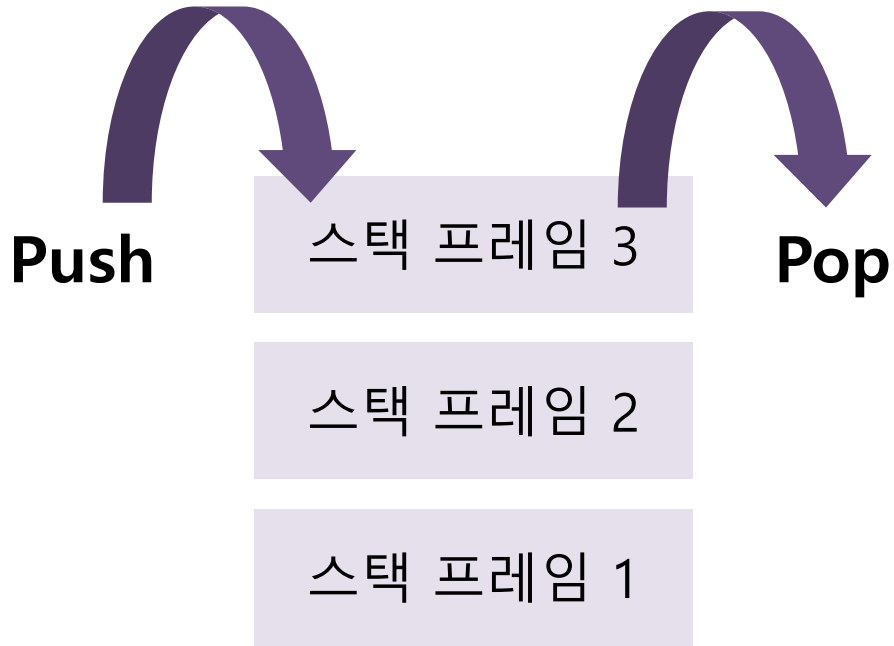
JVM의 메모리 구조 – Method Area = Static Area = Class Area

모든 스레드가 공유하는 메모리 영역
클래스 정보를 처음 메모리 공간에 올릴 때 초기화되는 대상을 저장하기 위한 메모리 공간
클래스, 인터페이스, 메서드, 필드, Static 변수 등의 바이트 코드를 보관함

* Runtime Constant Pool 이라는 별도의 관리 영역이 존재
  상수 자료형을 저장하여 참조하고 중복을 막는 역할을 수행함

JVM의 메모리 구조 – Stack Area

메서드 호출 시마다 각각의 스택 프레임이 생성됨
메서드 안에서 사용되는 값들을 저장하고, 호출된 메서드의 매개변수, 지역변수, 리턴 값 및 연산 시 일어나는 값을 임시로 저장함
메서드 수행이 끝나면 프레임별로 삭제



JVM의 메모리 구조 – Heap Area

객체를 저장하는 가상 메모리 공간

모든 스레드가 공유하며 new 키워드로 생성된 객체와 배열을 저장하는 영역

메서드 영역에 로드된 클래스만 생성이 가능

GC가 참조되지 않는 메모리를 확인하고 제거함

JVM의 메모리 구조 – PC Register

Thread가 시작될 때 생성되며 그때마다 생성되는 공간으로 Thread마다 각각 하나씩 존재함

Thread가 어떤 부분을 무슨 명령으로 실행해야 할 지에 대한 기록을 하는 부분

현재 수행중인 JVM 명령의 주소를 가짐

JVM의 메모리 구조 – Native method stack

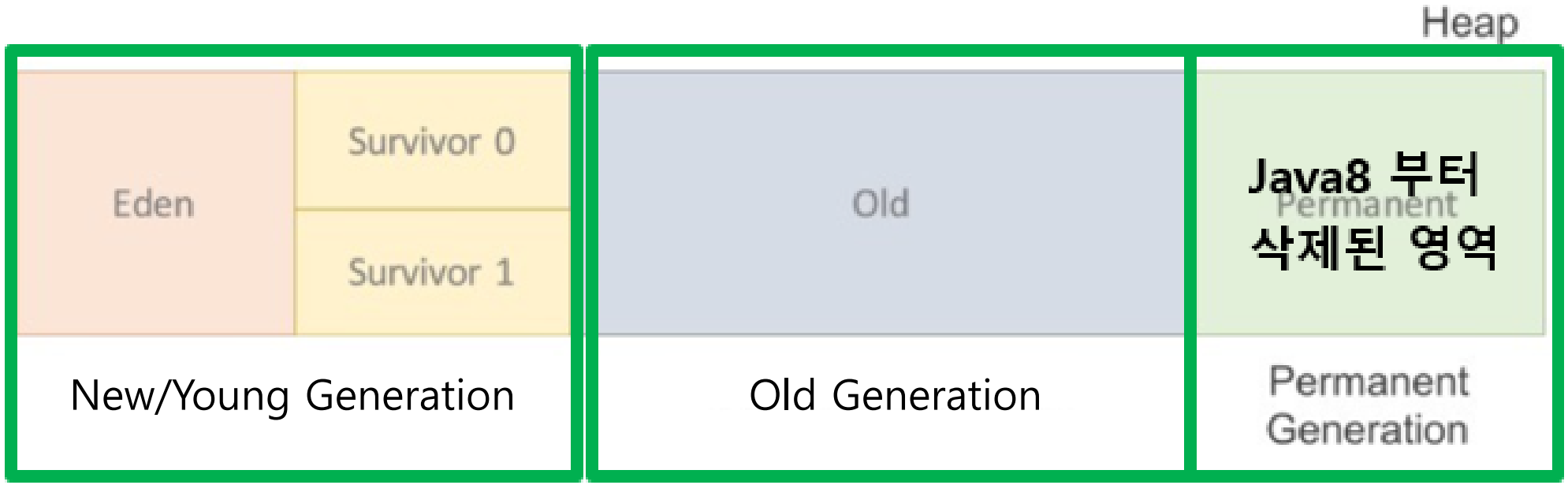
자바 외 언어로 작성된 네이티브 코드를 위한 메모리 영역
이 부분을 통해 C 코드를 실행시켜 Kernel에 접근할 수 있다.

JVM의 메모리 구조 중
Heap 영역에 대해 더 알아보자

Heap Area

객체를 저장하는 가상 메모리 공간

'객체는 대부분 일회성이며, 메모리에 오랫동안 남아있는 경우는 드물다' 라는 전제로 객체의 생존 기간에 따라 Heap의 영역을 나눔

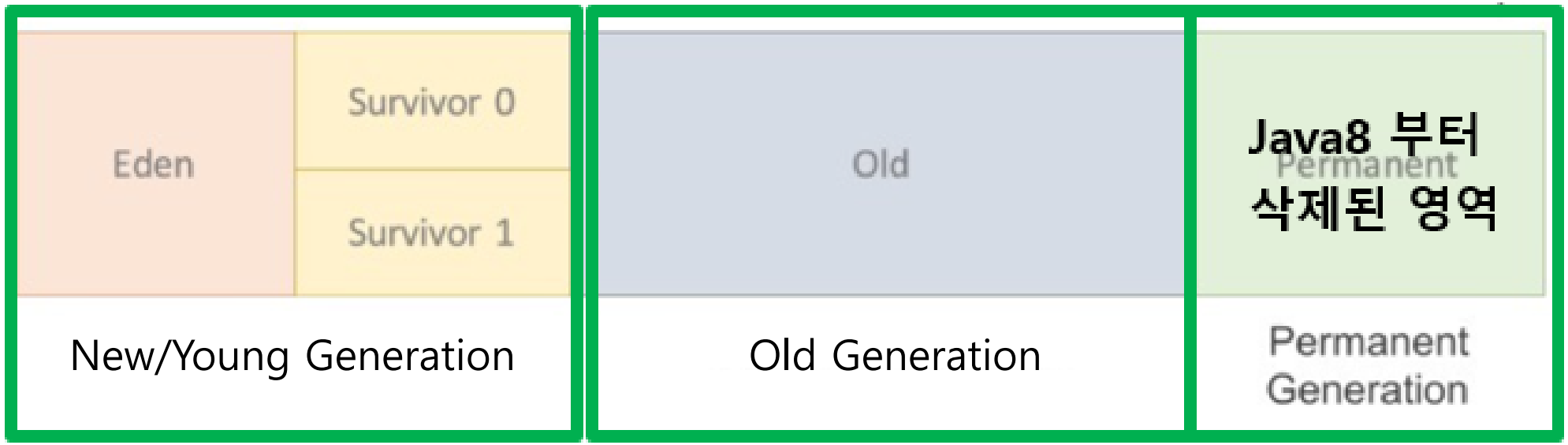


Heap Area - Permanent Generation

생성된 객체들의 정보의 주소값이 저장된 공간
Class Loader에 의해 로드되는 클래스, 메서드에 대한 Meta정보가 저장됨

**** Java8부터는 삭제되고 Metaspace로 교체됨**

Heap Area – New/Young Generation



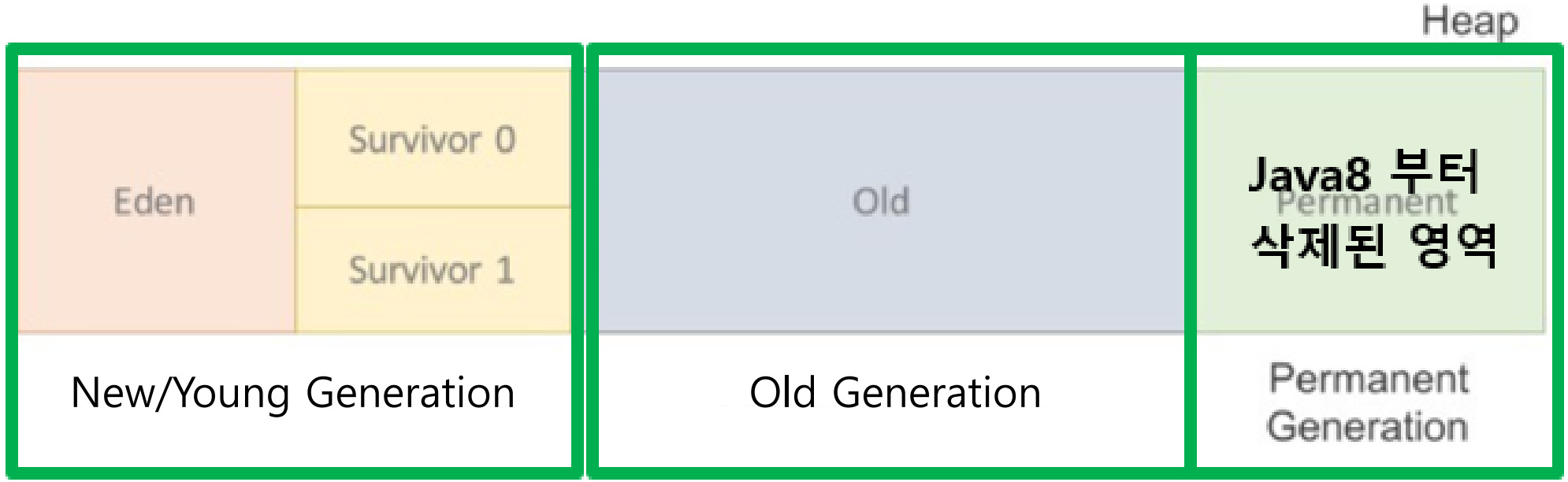
새롭게 생성된 객체가 할당되는 영역

Eden : 객체들이 최초로 생성되는 공간

Survivor 0,1 : Eden에서 참조되는 객체들이 저장되는 공간, 최소 1번의 GC 이상 살아남은 객체가 존재하는 영역

New/Young 부분에 대한 GC를 Minor GC라 함

Heap Area – Old Generation



일정시간 이상 참조되고 있는, 살아남은 객체들이 저장되는 공간

Old 부분에 대한 GC를 Major 또는 Full GC라 함

가비지 컬렉션 (GC)

가비지 컬렉션

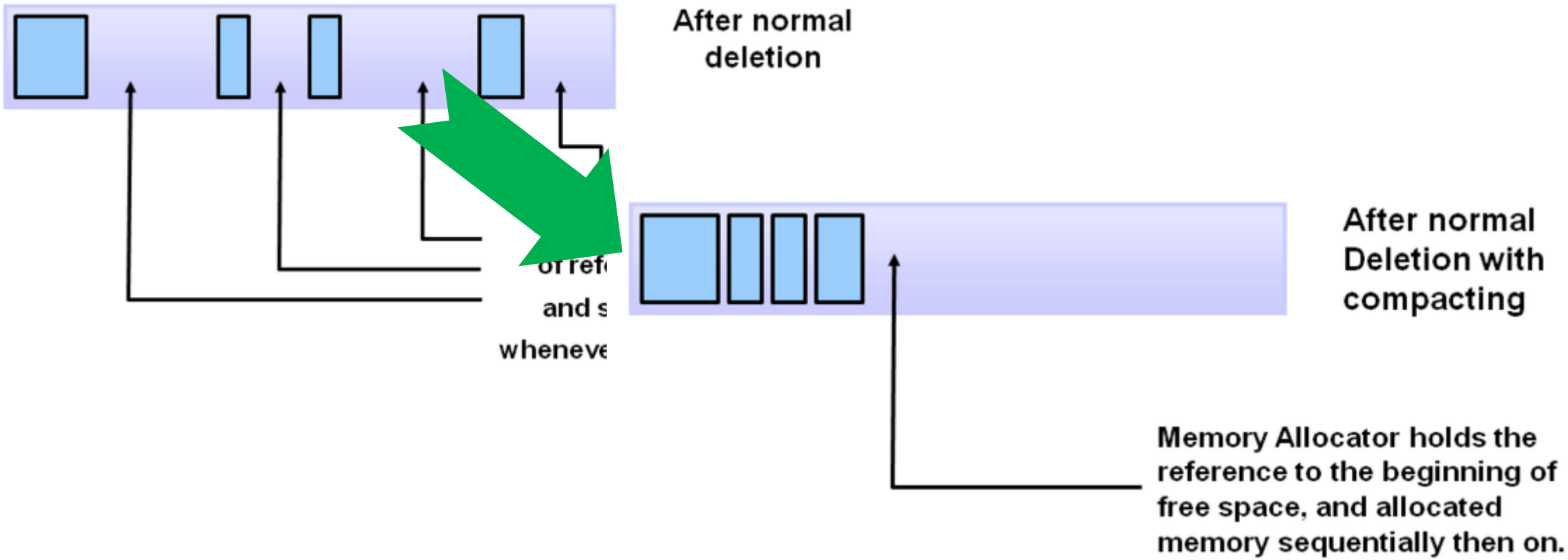
참조되지 않는 객체들을 탐색 후 삭제

삭제된 객체의 메모리를 반환 -> Heap 메모리의 재사용

가비지 컬렉션의 과정

1. **Heap내의 객체 중 가비지를 찾아냄** (Marking)
    1. 모든 오브젝트를 탐색하기 때문에 시간 오래걸림
2. **가비지를 제거하고, 메모리를 반환**
    1. 반환되어 비어진 메모리의 참조 위치를 저장해 두었다가, 새로운 객체가 선언되면 이곳에 할당되도록 함
3. **남은 참조되는 객체를 묶음**
    1. 새로운 메모리 할당 시 더 쉽고 빠르게 할당 진행 가능

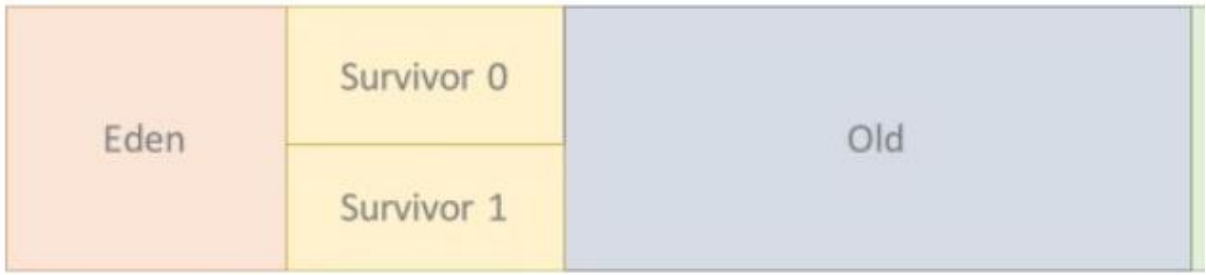
가비지 컬렉션의 과정


After normal deletion
and s
whenev
After normal Deletion with compacting
Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

가비지 컬렉션 – Minor GC

1. **Eden 영역에 객체가 가득 차게 되면 첫번째 GC가 발생**
    1. (Eden 영역에 있는 값들을 Survivor 0 영역에 복사하고, 이 영역을 제외한 나머지 영역의 객체를 삭제)
2. **Eden 영역과 Survivor 0영역의 메모리가 기준치 이상일 경우, Eden 영역의 객체와 Survivor0 영역에 참조되고 있는 객체가 있는지 검사**
    1. 참조되는 객체를 Survivor 1 영역에 복사, 제외한 나머지 영역의 객체를 삭제

이를 반복해 일정시간 이상 참조되고 있는 객체들을 Old 영역으로 이동



가비지 컬렉션 – Major GC

1. **Old 영역이 꽉 찼을 경우 Old 영역에 있는 모든 객체들을 검사**
    1. 참조되지 않는 객체들을 한꺼번에 삭제
2. **가비지 객체 삭제**

Minor GC에 비해 시간이 오래 걸리고 실행 중 프로세스가 정지됨
이때, 프로세스를 정지되는 것을 'Stop-the-world' 라고 하는데 Major GC가 발생하며 GC를 실행하는 스레드를 제외한 나머지 스레드는 모두 작업을 멈춤

가비지 컬렉션의 원리

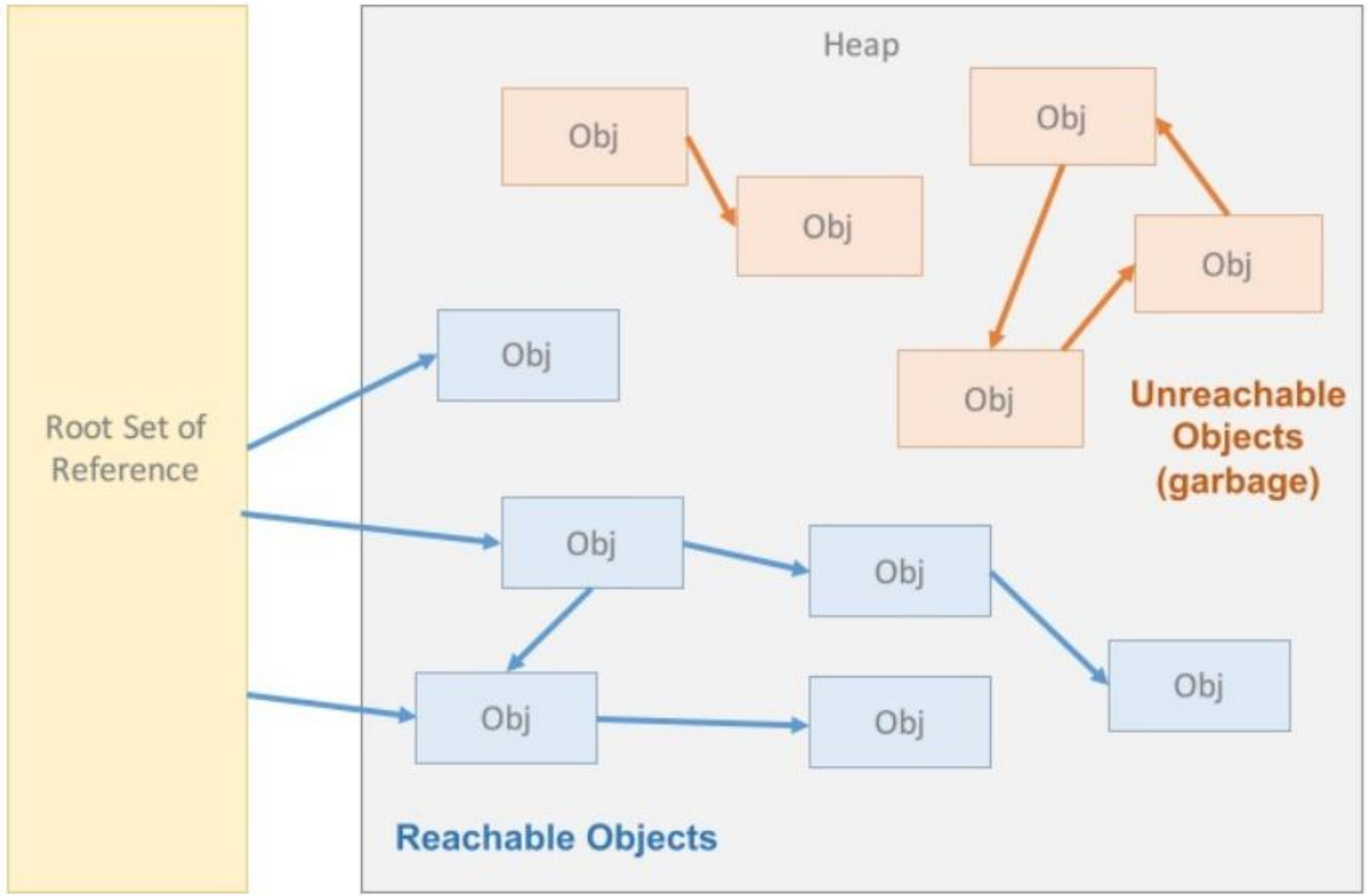
Garbage Collector는 Heap내의 객체 중 *가비지를 찾아내고, 찾아낸 가비지를 처리해 Heap의 메모리를 회수한다. -> 이러한 행위를 Mark and Sweep 이라 함
이때, 객체가 가비지인지 아닌지 판단하기 위해 reachability라는 개념을 사용

어떤 Heap 영역에 할당된 객체가 유효한 참조가 있으면 reachability,
없다면 unreachability로 판단함

*** 가비지**

참조되고 있지 않은 객체

가비지 컬렉션의 원리

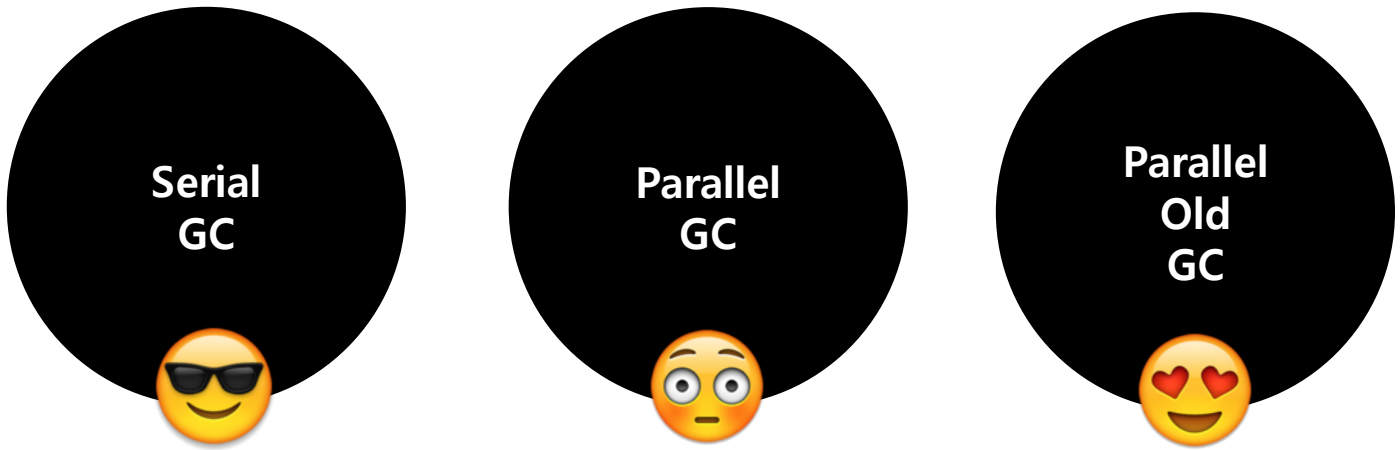

Heap
Obj
Obj
Obj
Obj
Obj
Obj
Obj
Root Set of Reference
Unreachable Objects (garbage)
Obj
Obj
Obj
Obj
Obj
Reachable Objects

가비지 컬렉션의 원리

객체가 가비지 컬렉션의 대상이 되었다고 해서 바로 소멸 되는 것은 아님!

빈번한 가비지 컬렉션의 실행은 시스템에 부담이 될 수 있음

따라서 성능에 영향을 미치지 않도록 가비지 컬렉션 실행 타이밍은 별도의 알고리즘을 기반으로 계산됨



JVM?

JVM의 메모리 구조?

가비지 컬렉션?

GC가 관리하는 메모리 영역?

가비지 컬렉션의 과정

Minor GC와 Major GC의 차이점