Garbage Collection

20210463 박연종

park@duck.com

SSL SEMINAR

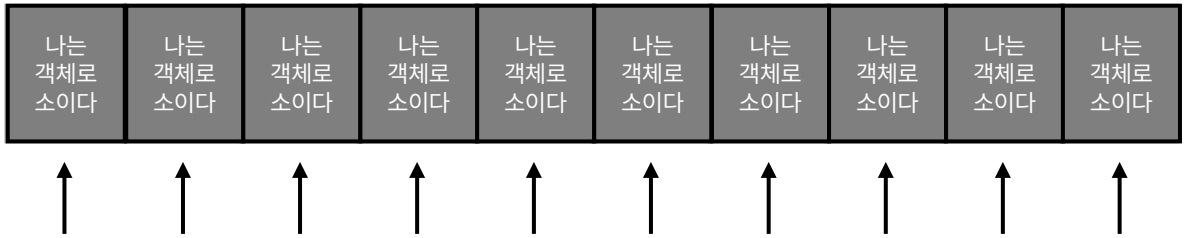
Photo by Pawel Czerwinski

01.
Java의 가비지 컬렉션

메모리 관리의 필요성

프로그램에서 메모리 공간을 운영체제로부터 할당 받은 후 반환하지 않는 작업이 축적돼 사용할 수 있는 메모리 공간이 감소하는 **메모리 누수 현상** 발생

```
Exception in thread "main" java.lang.OutOfMemoryError:
request bytes for Chunk::new. Out of swap space?
```




```java
class Stack {  2 usages
    private Integer[] stackArray;  3 usages
    private int capacity;  2 usages
    private int top;  3 usages

    public Stack() {  1 usage
        this.capacity = Integer.MAX_VALUE / 2;
        this.stackArray = new Integer[this.capacity];
        this.top = -1;
    }

    public void push(Integer e) {  1 usage
        stackArray[++top] = e;
    }

    public Integer pop() {  1 usage
        return stackArray[top--];
    }
}
```

메모리 관리의 필요성

```java
class Stack {  2 usages
    private Integer[] stackArray;  3 usages
    private int capacity;  2 usages
    private int top;  3 usages

    public Stack() {  1 usage
        this.capacity = Integer.MAX_VALUE / 2;
        this.stackArray = new Integer[this.capacity];
        this.top = -1;
    }

    public void push(Integer e) {  1 usage
        stackArray[++top] = e;
    }

    public Integer pop() {  1 usage
        return stackArray[top--];
    }
}
```

가비지 컬렉션

- Java에서는 활용되지 않는 동적 할당된 메모리 공간을 JVM 실행 엔진의 **Garbage Collector**가 정리해 개발자가 메모리에 신경 쓰지 않고 개발에 집중할 수 있도록 도와줌
- C++11부터는 메모리 누수를 방지하기 위해 하나의 포인터만이 객체를 가리킬 수 있도록 하거나, 참조 개수를 기록하는 포인터로 참조 개수가 0이 되면 자동으로 `delete`를 실행해 메모리 공간을 해제하는 **스마트 포인터**를 지원함
- JVM 계열 외에도 Python, Ruby, JavaScript, Golang 등으로 만들어진 실행 파일에는 Garbage Collector가 포함됨

- 메모리 공간이 언제 해제되는지 정확하게 알 수 없어 제어하기 힘듦
- Garbage Collection 처리 동안에는 JVM이 애플리케이션 실행을 멈추어 서비스가 중단될 수 있음 (Stop-The-World)

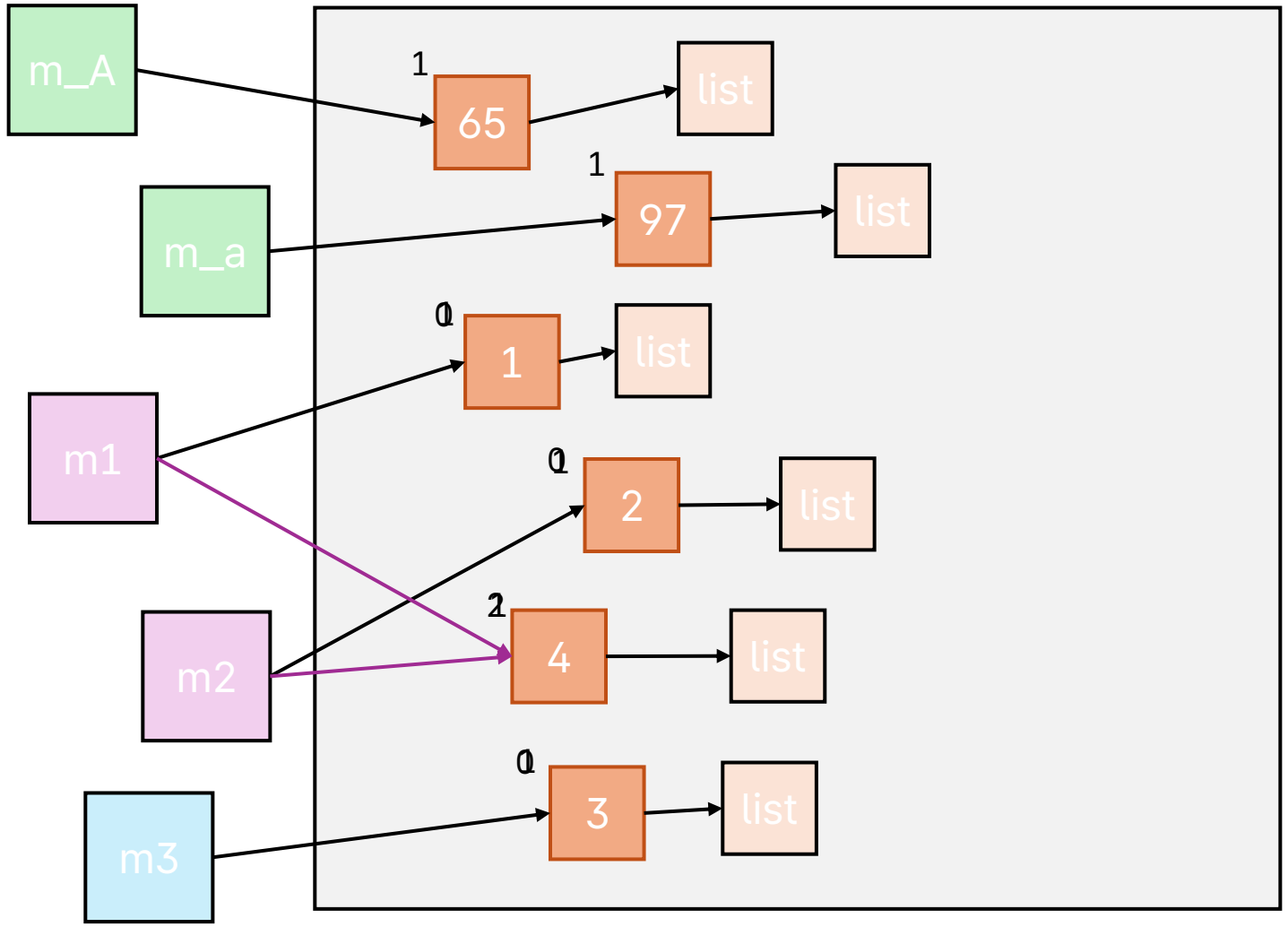
02.
Reference Counting

가비지 컬렉션 알고리즘

Reference Counting


```java
public class Main {
    static Mana mana_A = new Mana( i: 65);  no usages
    static Mana mana_a = new Mana( i: 97);  no usages

    public static void main(String[] args) {
        Mana mana1 = new Mana( i: 1);
        Mana mana2 = new Mana( i: 2);
        createMana3();
        mana1 = new Mana( i: 4);
        mana2 = mana1;
    }

    static void createMana3() {  1 usage
        Mana mana3 = new Mana( i: 3);
    }
}
```

```java
class Mana {  11 usages
    List<Integer> list;  1 usage

    public Mana(int i) {  6 usages
        this.list = new ArrayList<>();
    }
}
```

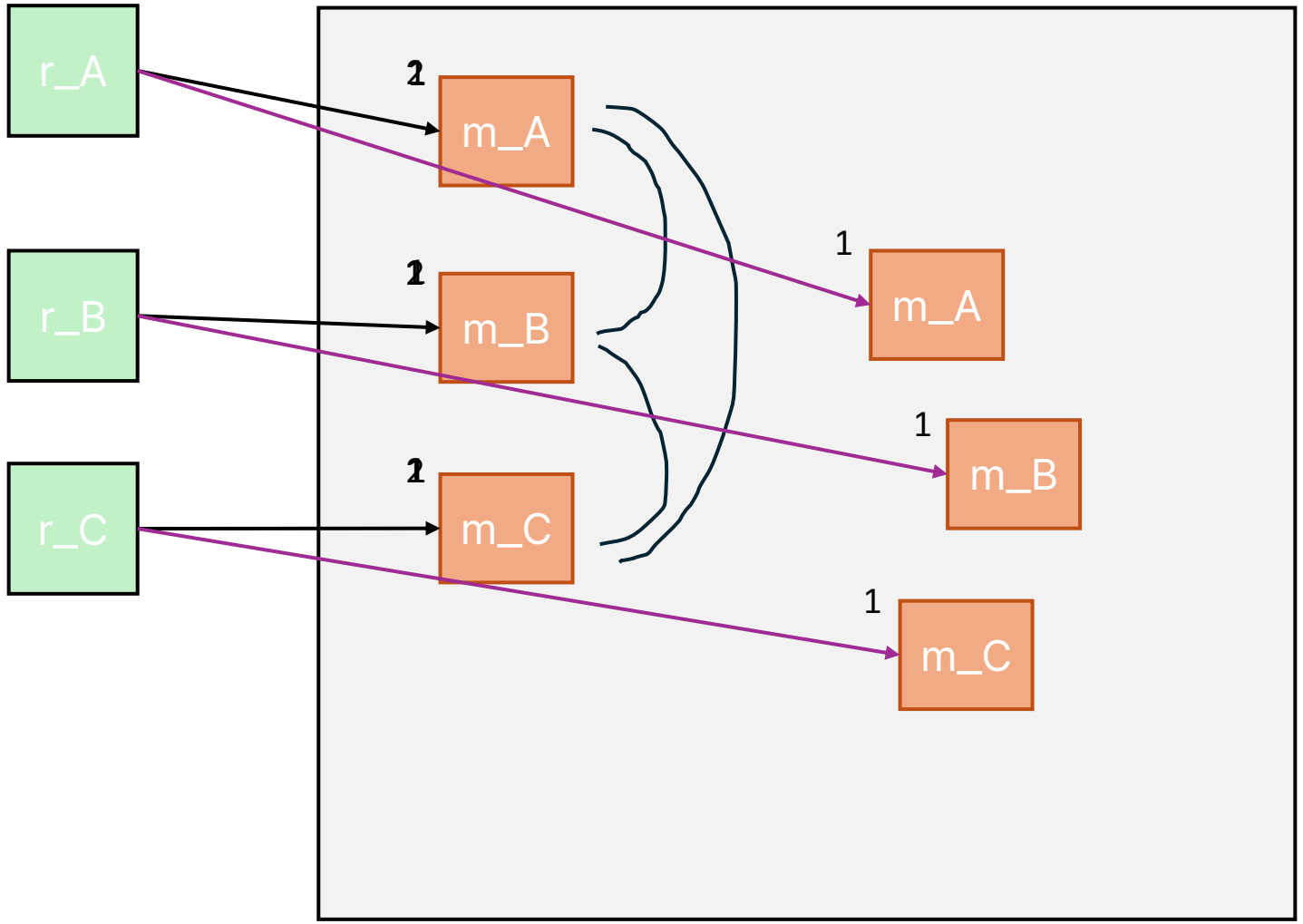


GC Root

Heap Area

Reference Counting 시의 문제점

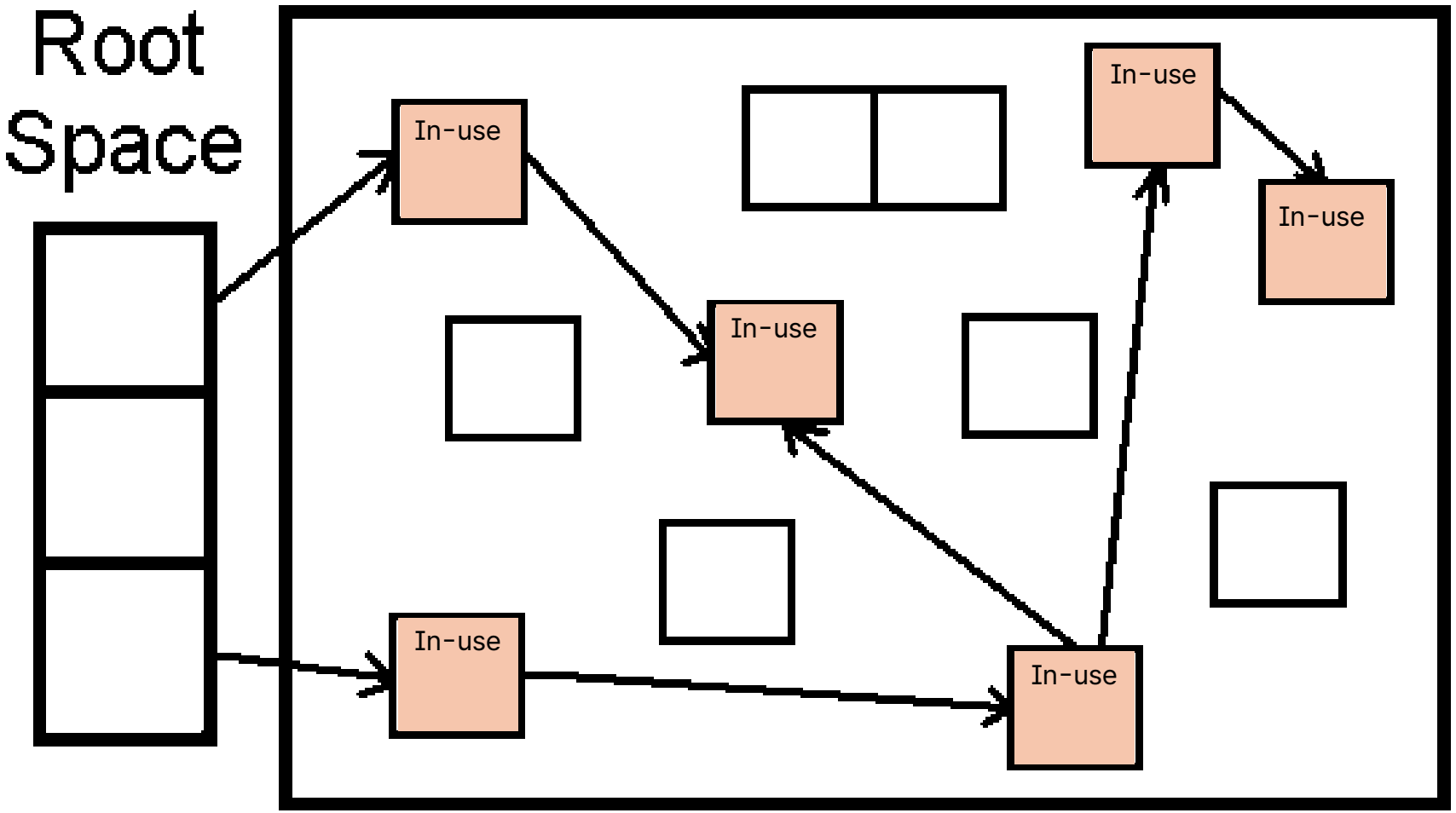

```java
public class Main {
    public static void main(String[] args) {
        Marron marronA = new Marron();
        Marron marronB = new Marron();
        Marron marronC = new Marron();
        marronA.setSibling(marronB);
        marronB.setSibling(marronC);
        marronC.setSibling(marronA);

        marronA = new Marron();
        marronB = new Marron();
        marronC = new Marron();
    }
}
```



03.
Mark and Sweep

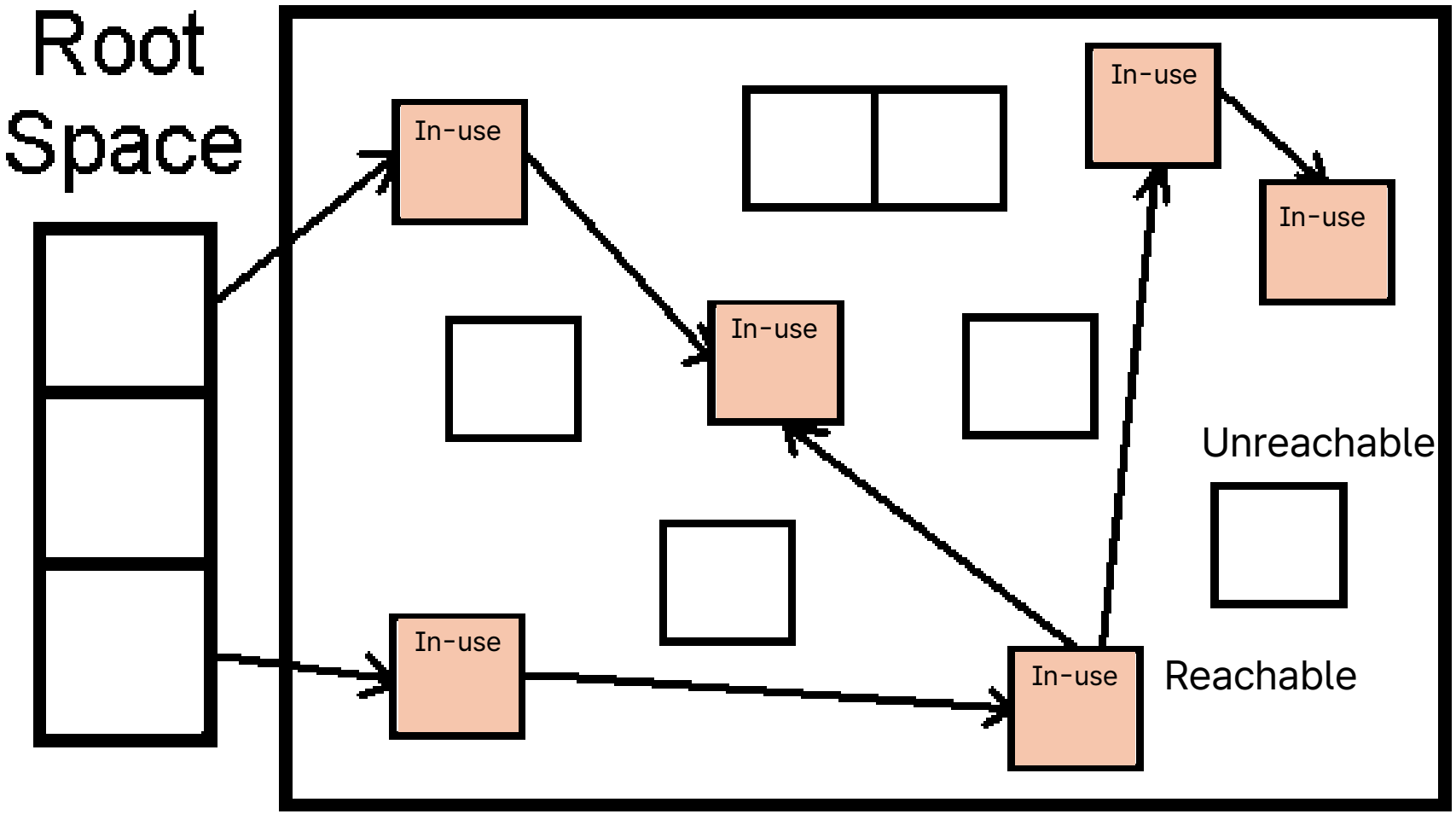
가비지 컬렉션 알고리즘

Mark and Sweep



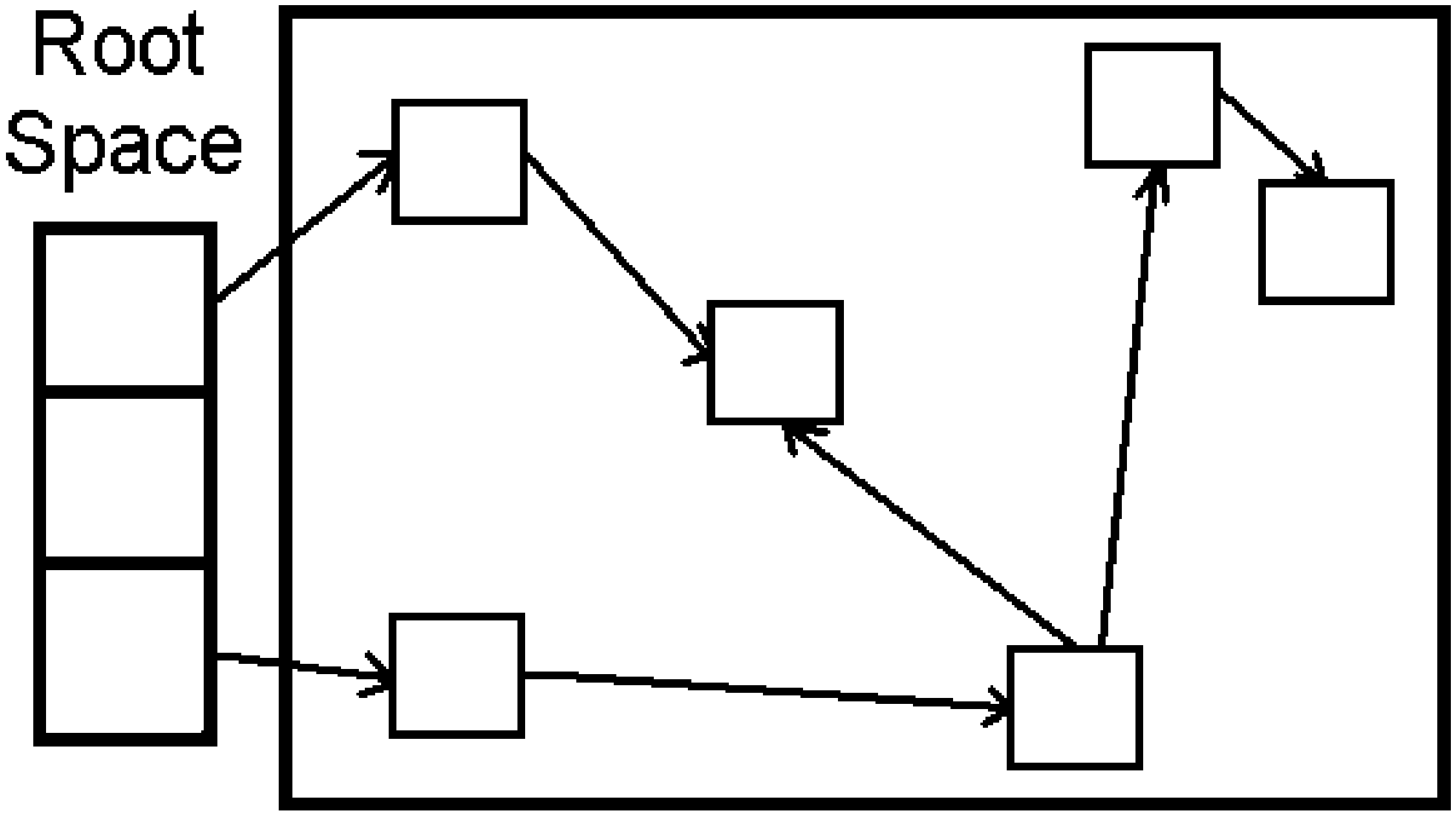
https://rebelsky.cs.grinnell.edu/Courses/CS302/99S/Presentations/GC/

Mark and Sweep



https://rebelsky.cs.grinnell.edu/Courses/CS302/99S/Presentations/GC/

Mark and Sweep

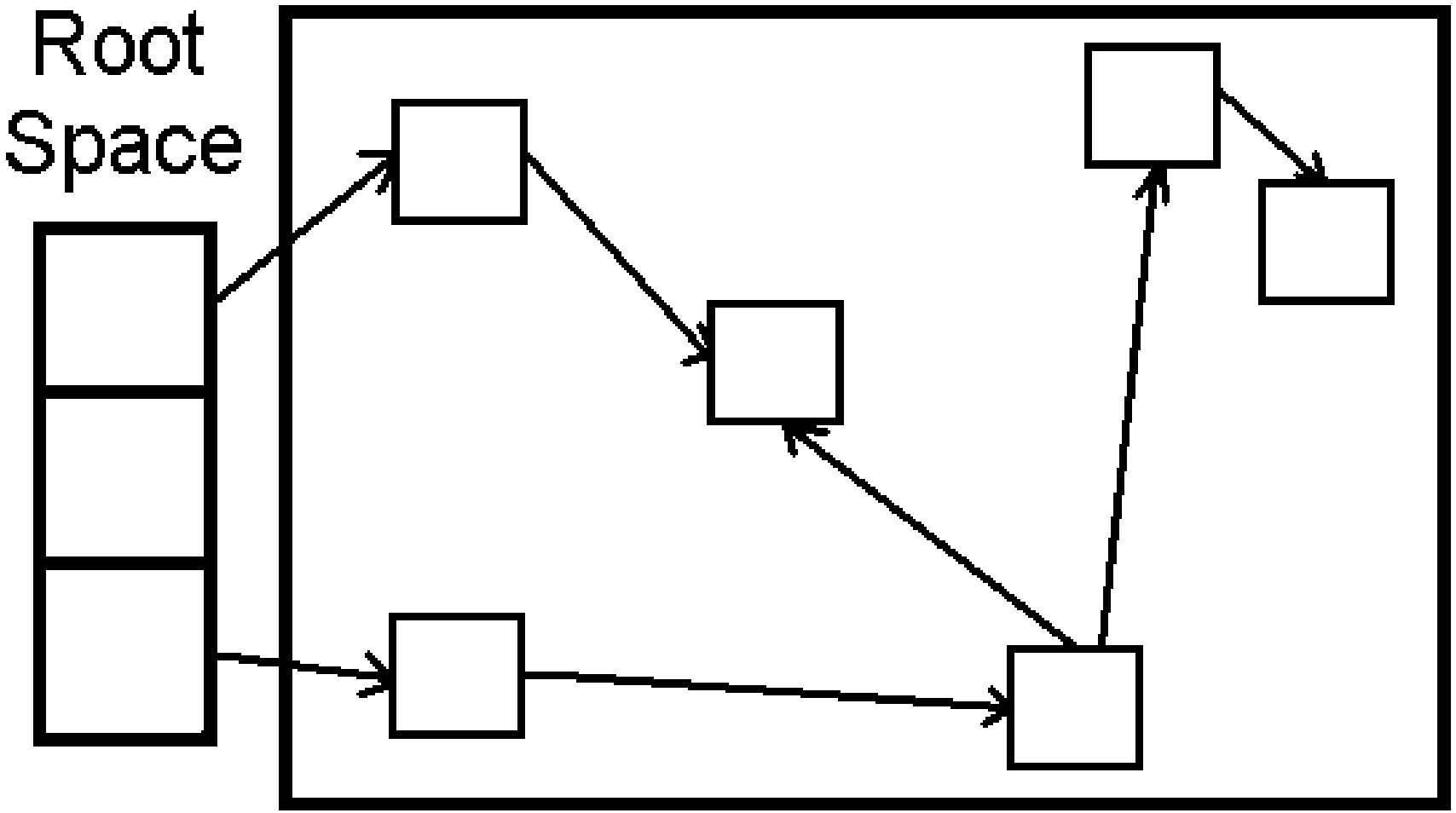


https://rebelsky.cs.grinnell.edu/Courses/CS302/99S/Presentations/GC/

04.
Mark-Sweep-Compact

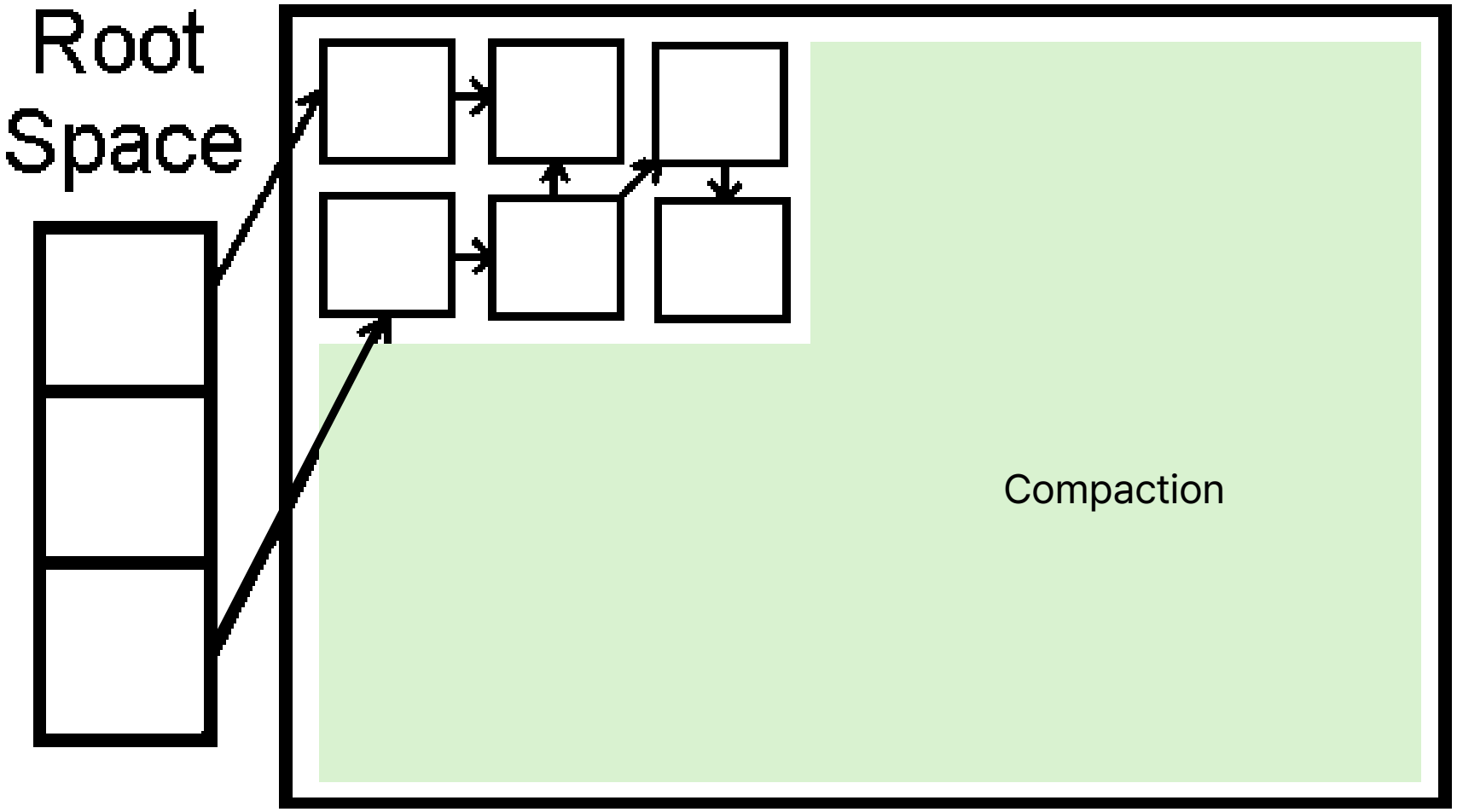
가비지 컬렉션 알고리즘

Mark-Sweep-Compact



https://rebelsky.cs.grinnell.edu/Courses/CS302/99S/Presentations/GC/

Mark-Sweep-Compact

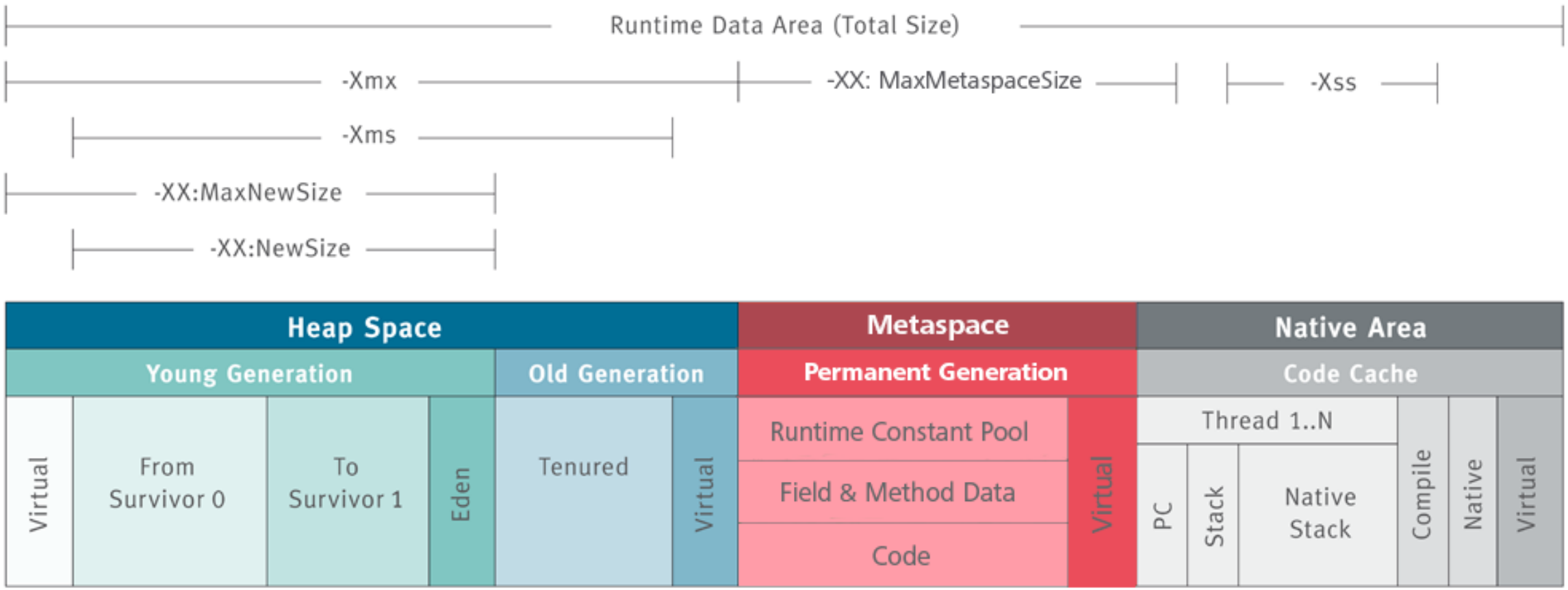


https://rebelsky.cs.grinnell.edu/Courses/CS302/99S/Presentations/GC/

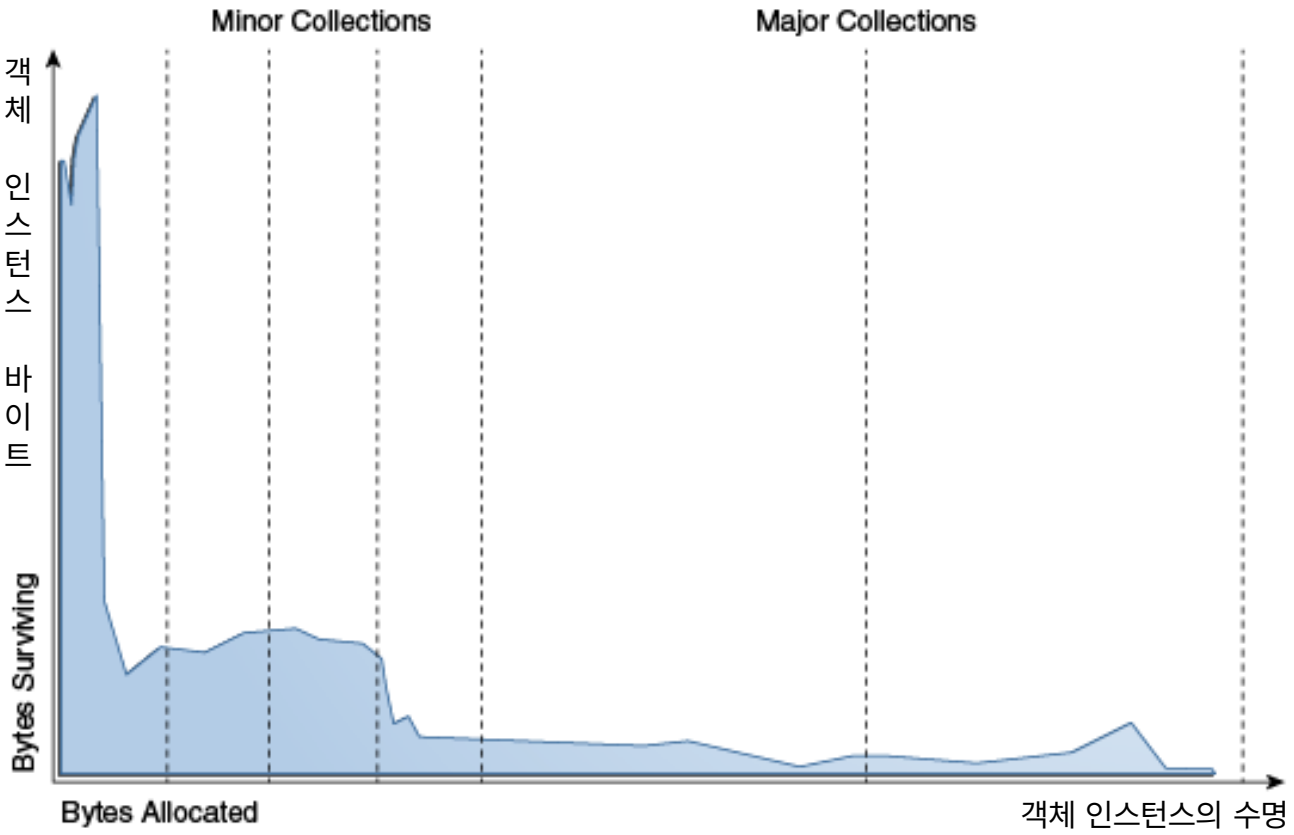
05.
Heap Space

JVM Runtime Data Area의 Heap Space 구조

Java 9+의 Runtime Data Area



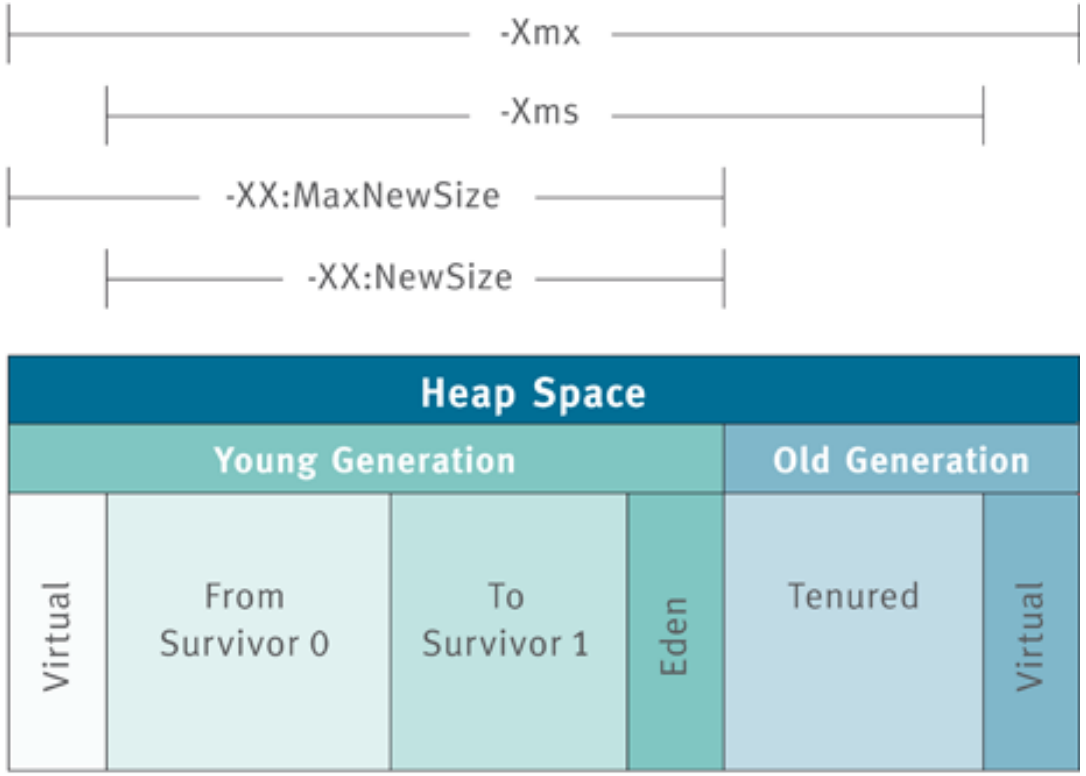
약한 세대 가설 *(Weak Generational Hypothesis)*



- 대부분의 객체는 금방 접근 불가능 상태가 됨
- 오래된 객체에서 젊은 객체로의 참조는 아주 적게 존재

https://docs.oracle.com/en/java/javase/16/gctuning/garbage-collector-implementation.html

Heap Space



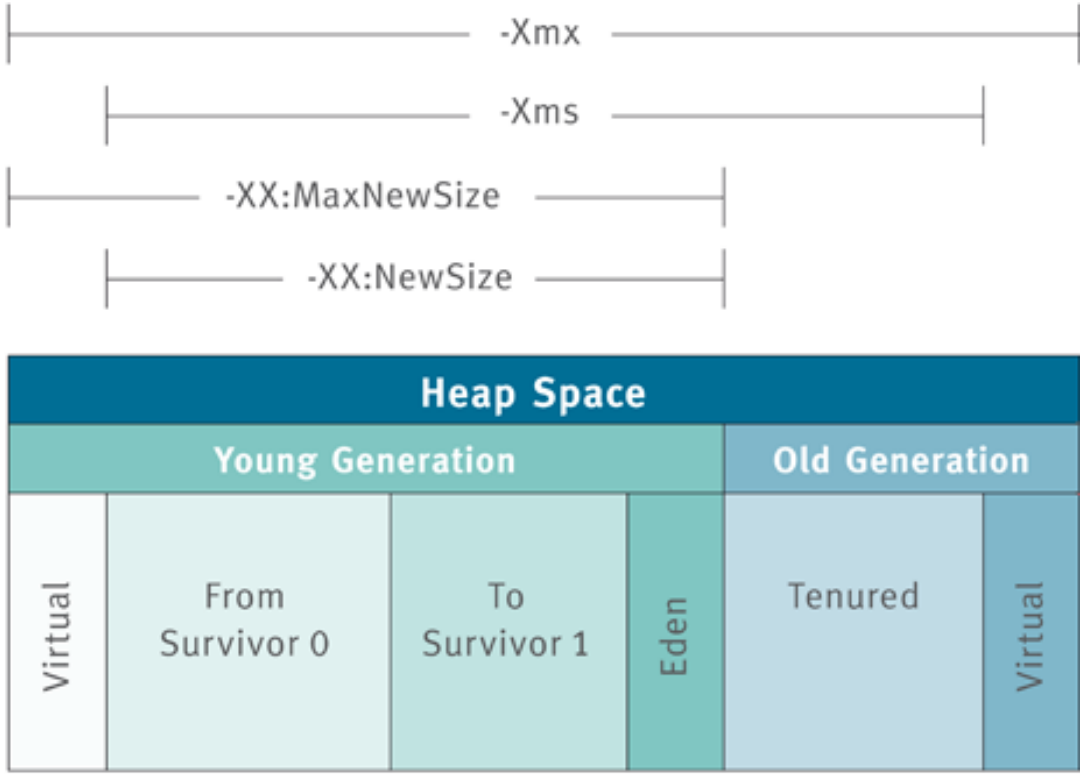
Young Generation

- 새롭게 생성되는 객체 인스턴스를 위한 공간이 할당되는 영역
- 대부분의 객체가 얼마 되지 않아 접근 불가 상태가 되므로 많은 객체 인스턴스가 이곳에서 생성 후 삭제됨

Old Generation

- Young Generation 영역에서 접근 가능 상태를 유지해 살아남은 객체가 복사돼 공간을 차지하는 영역
- 일반적으로 Young Generation 영역보다 공간을 크게 잡음

Heap Space



Eden

- new 키워드를 통해 새로 생성되는 객체 인스턴스가 위치하는 공간

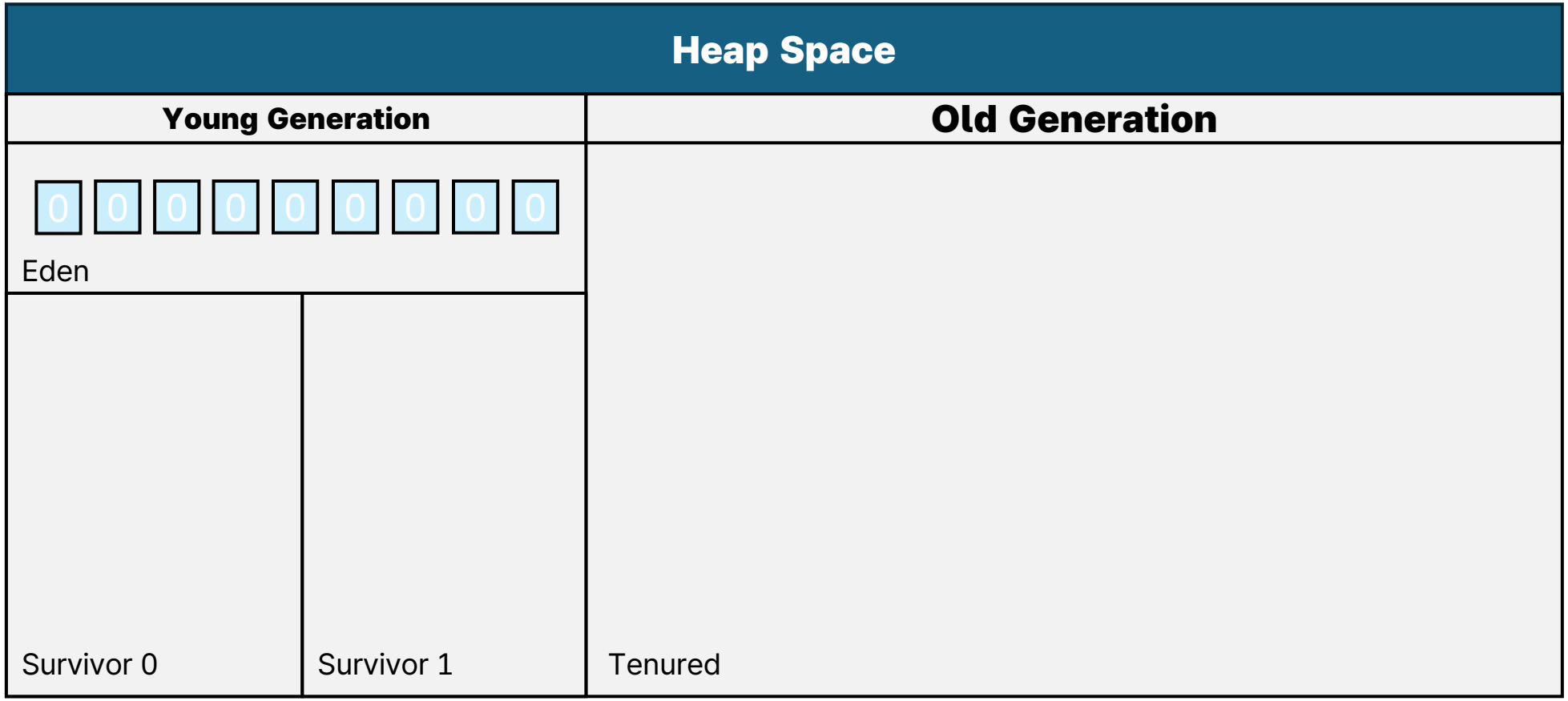
Survivor 0 / Survivor 1

- 최소 한 번 이상의 가비지 컬렉션 이후 존재하는 객체 인스턴스가 위치하는 공간
- Swap을 위해 Survivor 0 또는 Survivor 1 중 하나는 비어 있어야 함

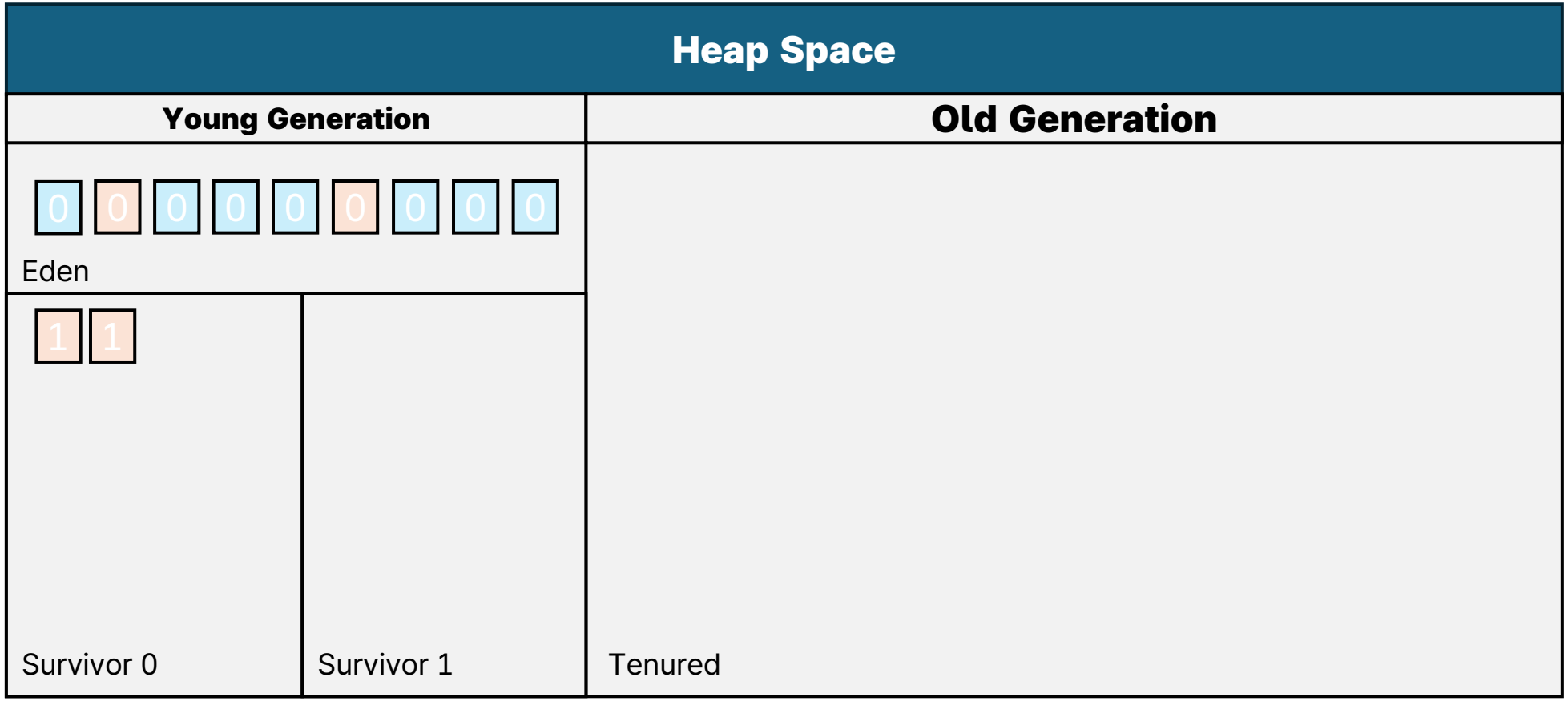
06.
Minor GC

Heap의 Young Generation 영역의 가비지 컬렉션

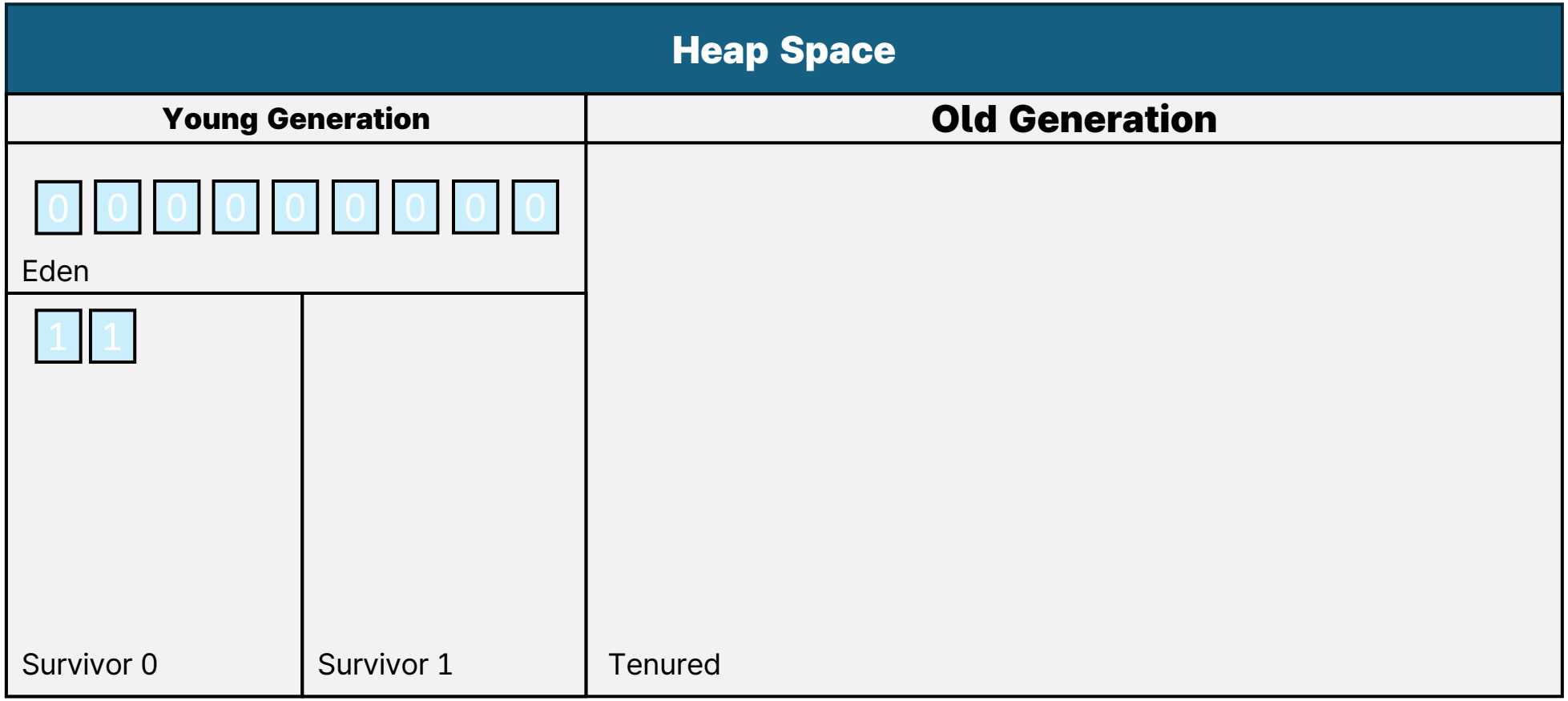
Minor GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0 0
Eden
Survivor 0
Survivor 1
Tenured

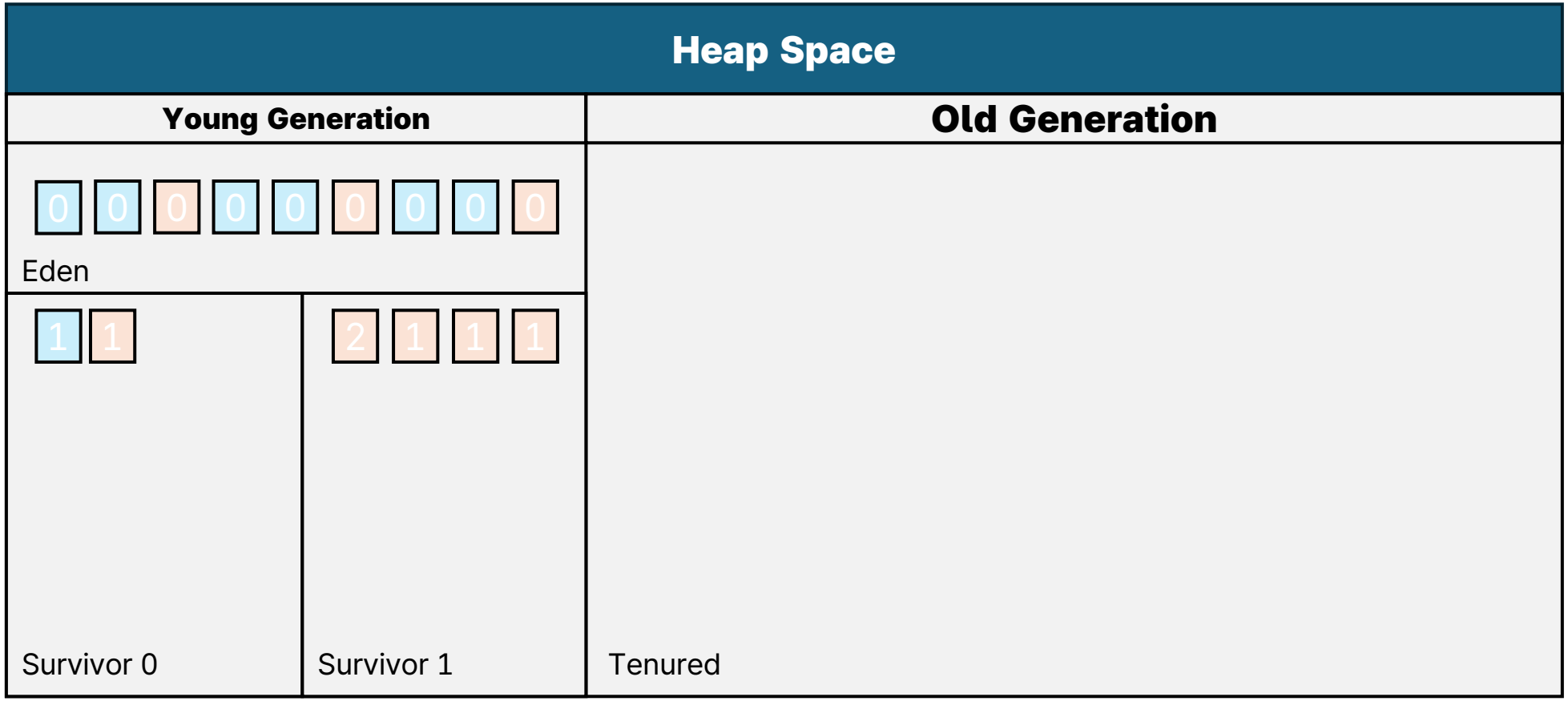
Minor GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0 0
Eden
1 1
Survivor 0
Survivor 1
Tenured

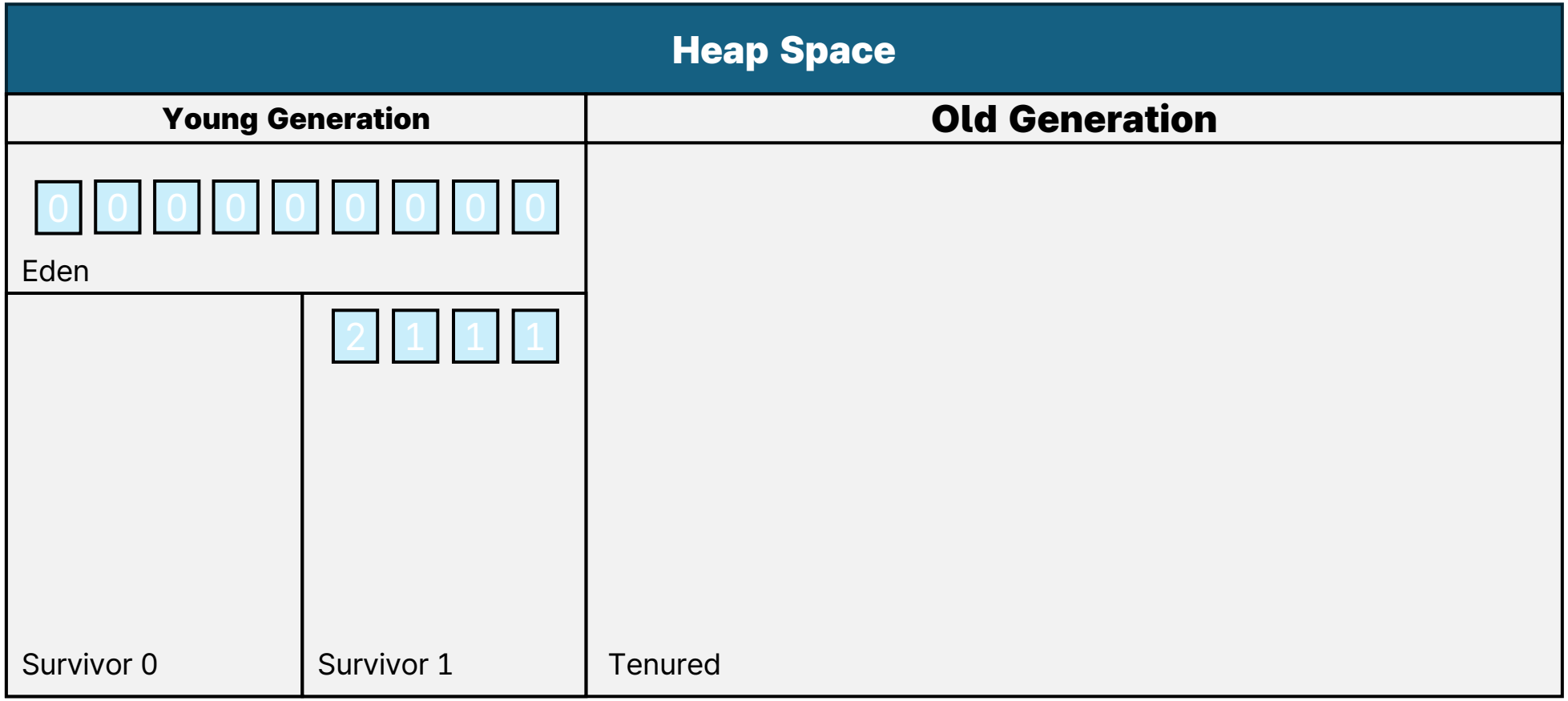
Minor GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0 0
Eden
1 1
Survivor 0
Survivor 1
Tenured

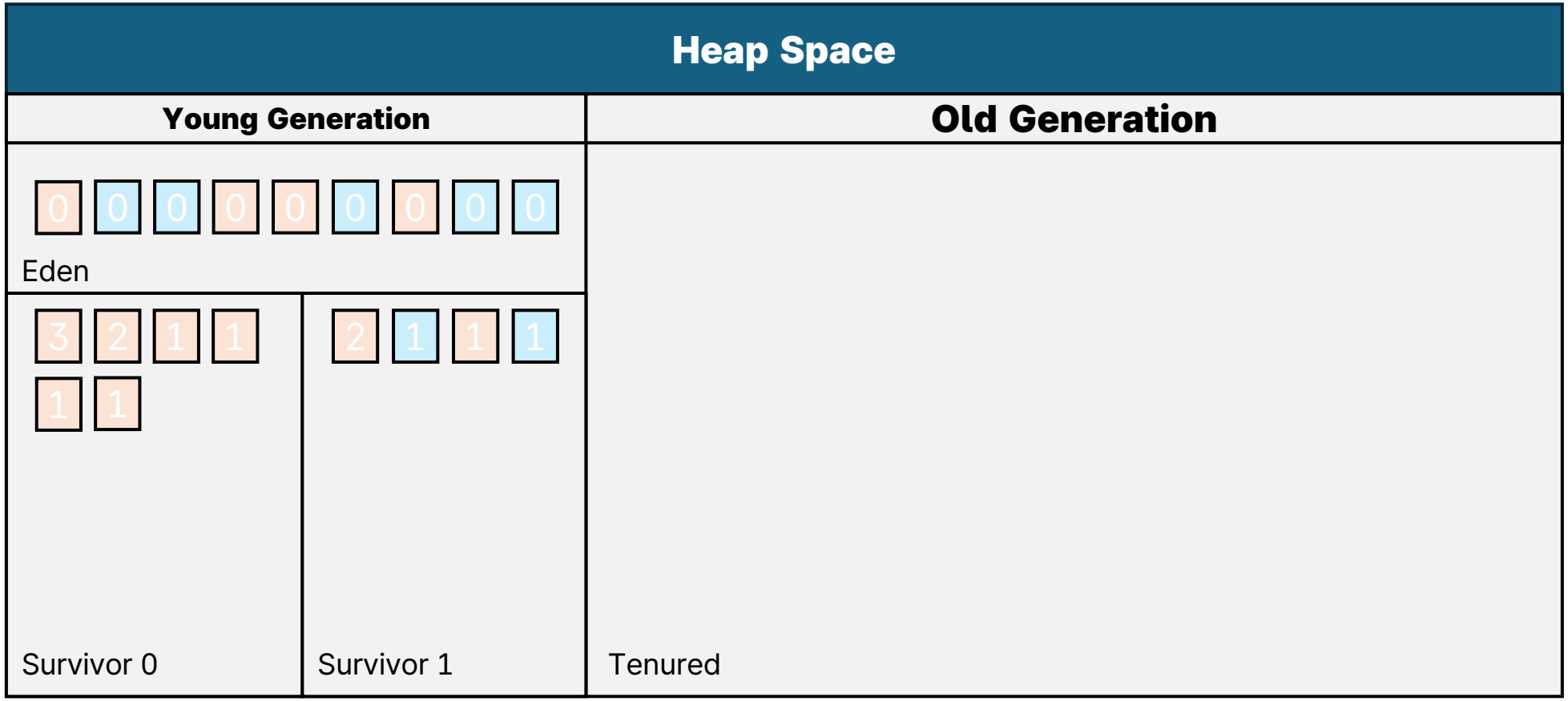
Minor GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0 0
Eden
1 1
Survivor 0
2 1 1 1
Survivor 1
Tenured

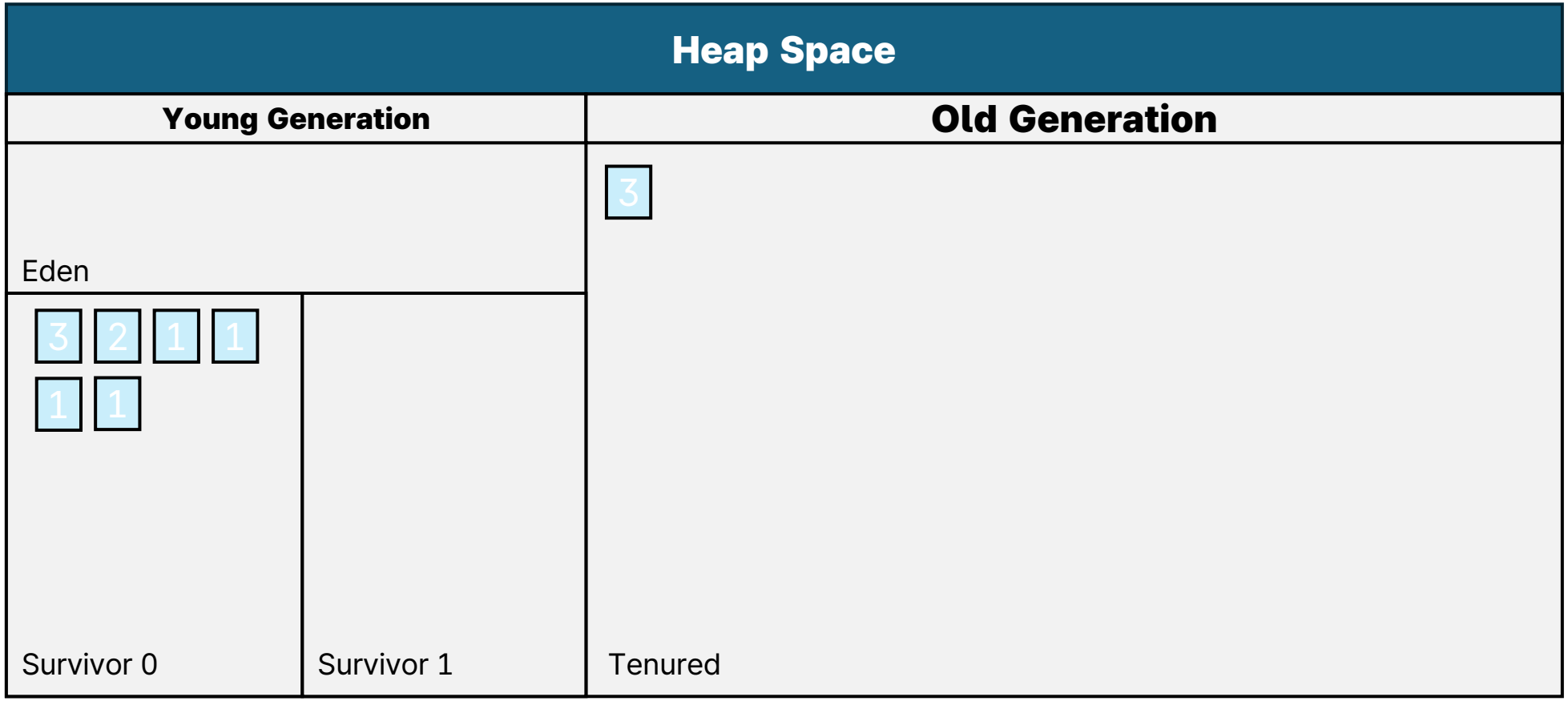
Minor GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0 0
Eden
2 1 1 1
Survivor 0
Survivor 1
Tenured

Minor GC

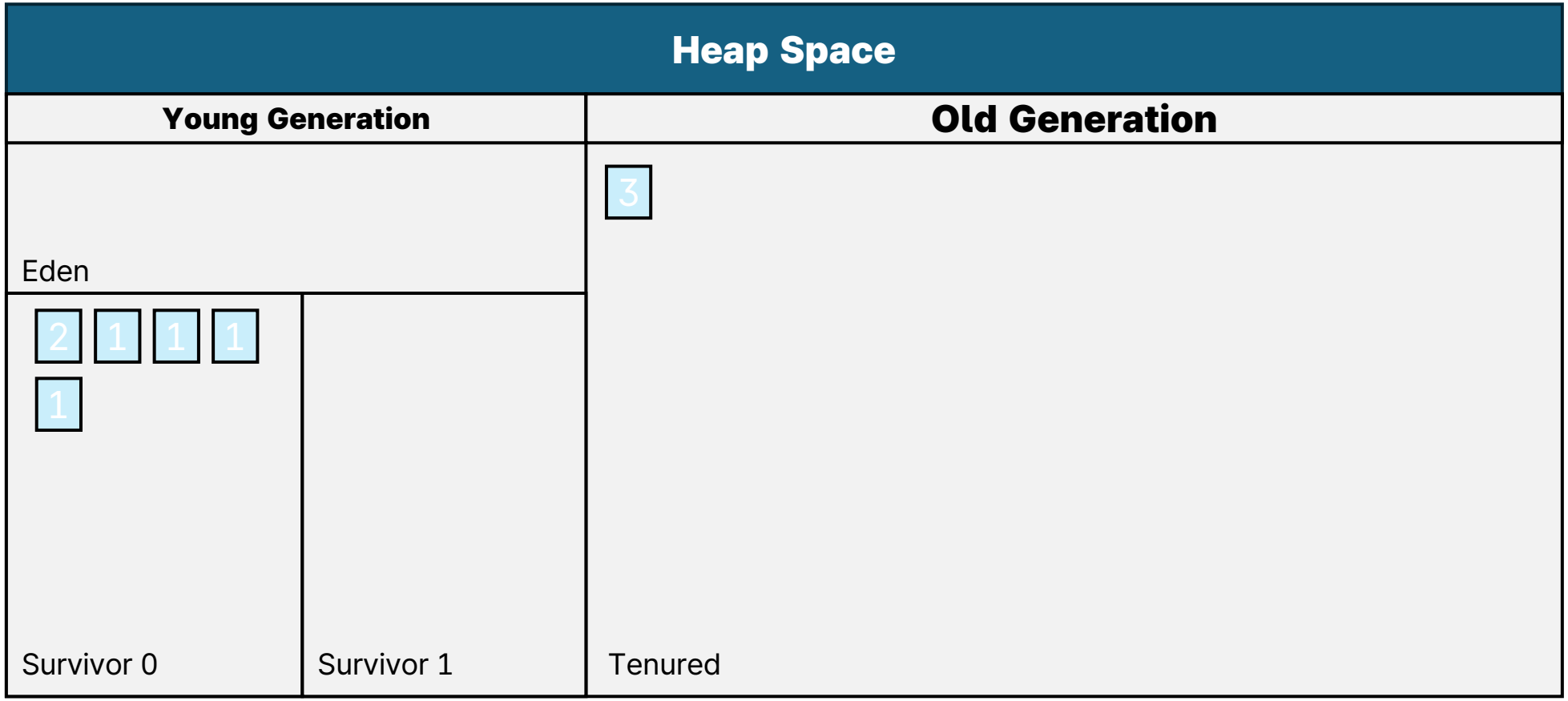

Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0 0
Eden
3 2 1 1 1 1
2 1 1 1
Survivor 0
Survivor 1
Tenured

Minor GC


Heap Space
Young Generation
Old Generation
3
Eden
3
2
1
1
1
1
Survivor 0
Survivor 1
Tenured

Minor GC

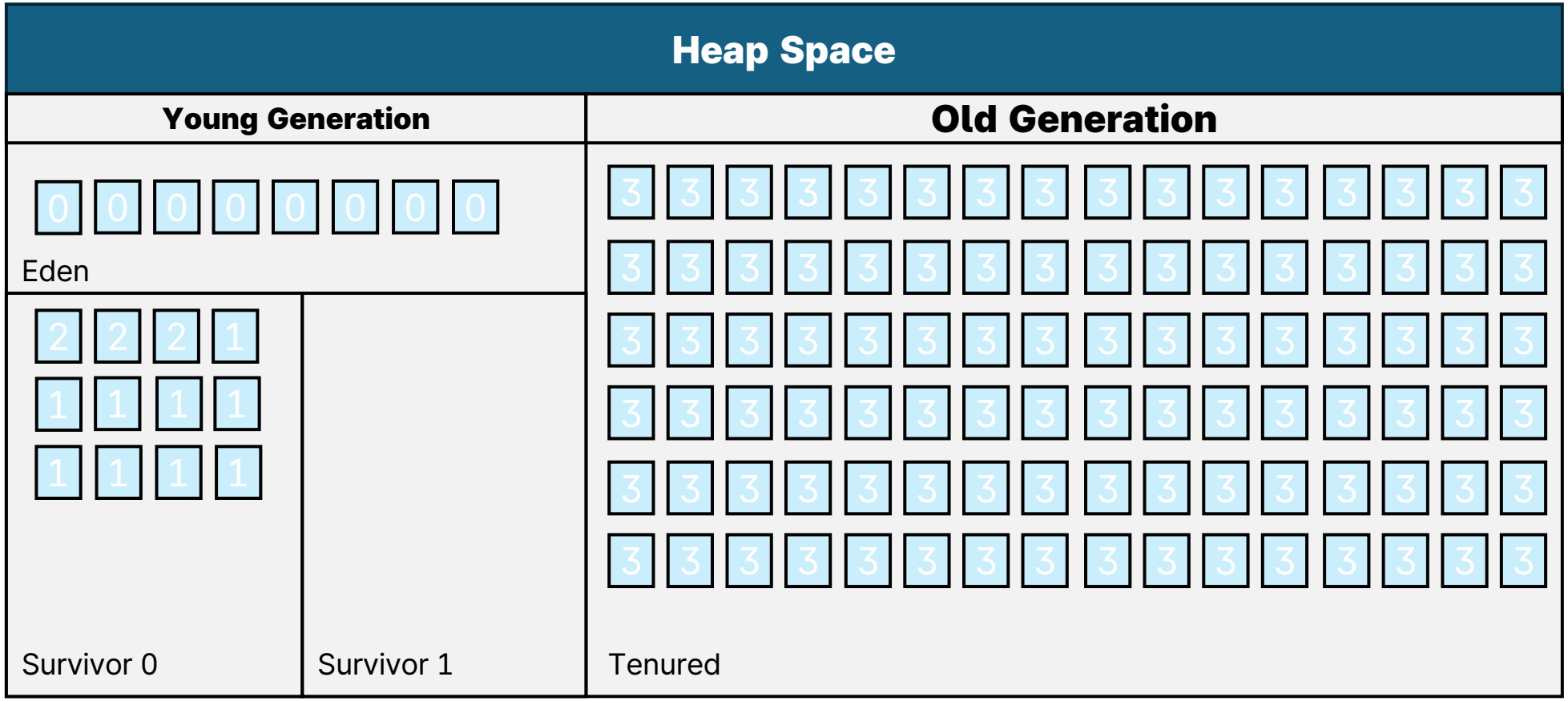
Promotion 임계점(age) 이상이 되면 Old Generation으로 이동 (Java 7+의 기본값은 15)



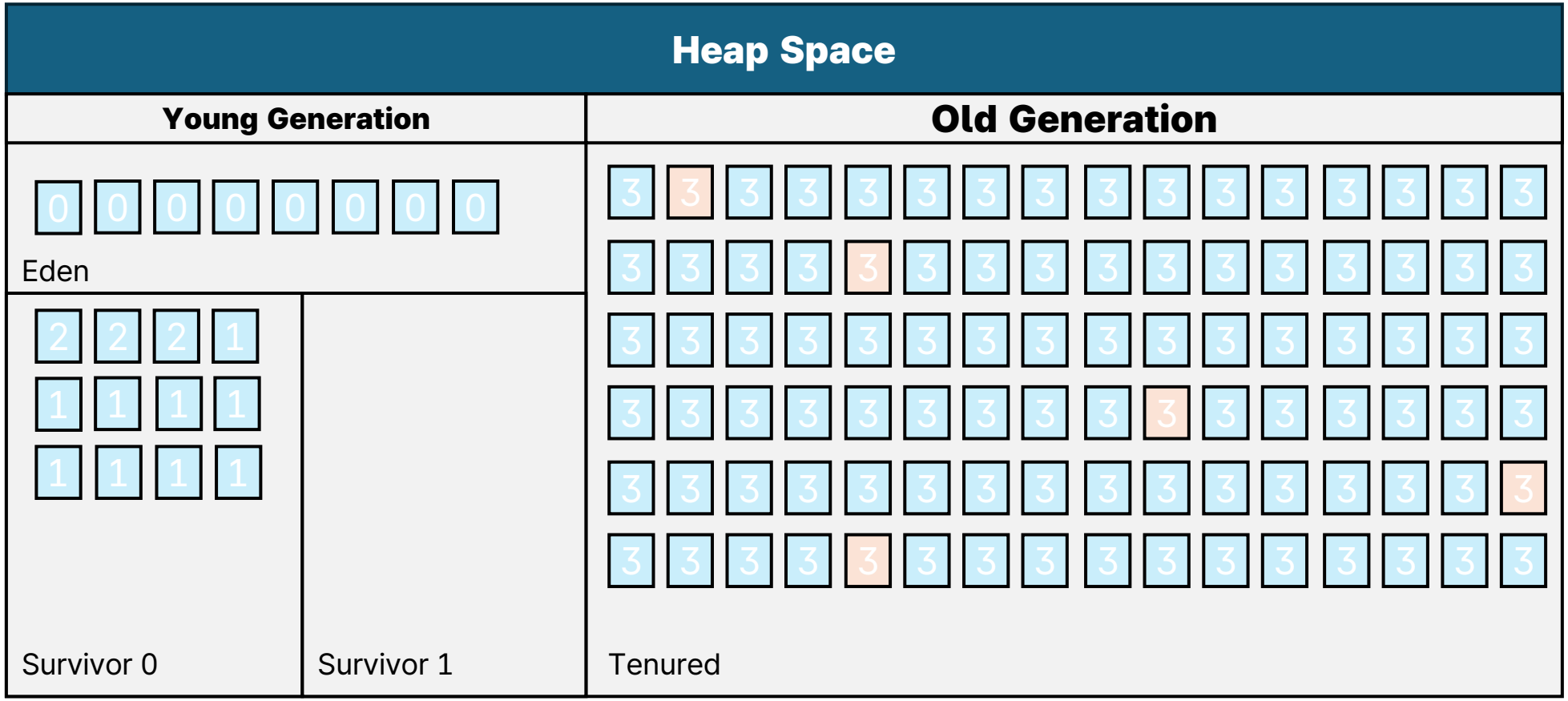
07.
Major GC *(Full GC)*

Heap의 Old Generation 영역의 가비지 컬렉션

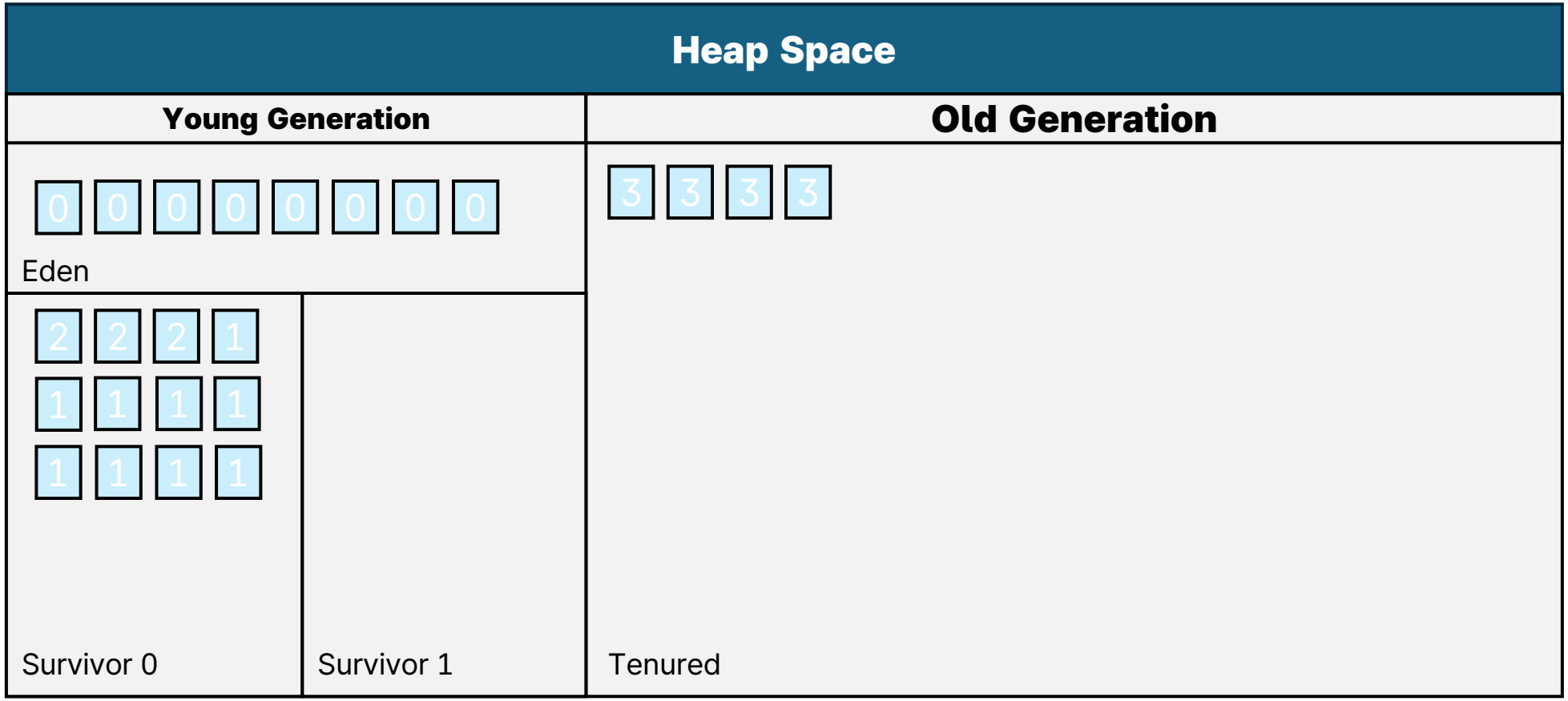
Major GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0
Eden
2 2 2 1
1 1 1 1
1 1 1 1
Survivor 0
Survivor 1
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
Tenured

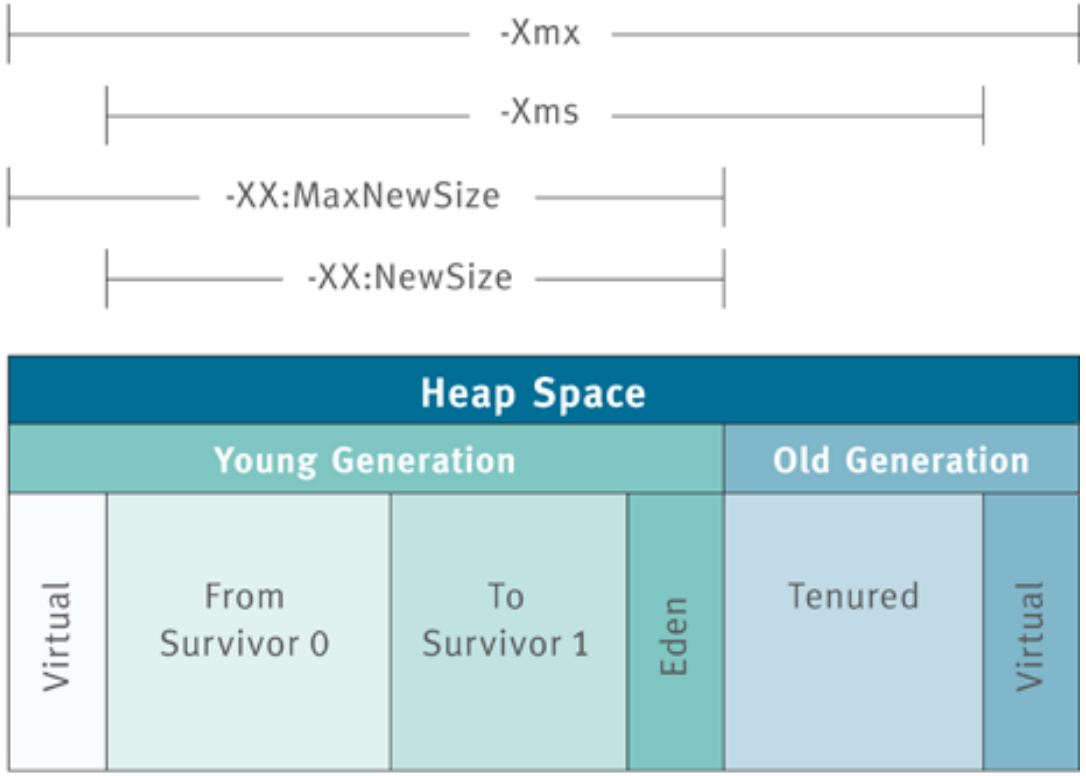
Major GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0
Eden
2 2 2 1
1 1 1 1
1 1 1 1
Survivor 0
Survivor 1
Tenured

Major GC


Heap Space
Young Generation
Old Generation
0 0 0 0 0 0 0 0
Eden
3 3 3 3
2 2 2 1
1 1 1 1
1 1 1 1
Survivor 0
Survivor 1
Tenured

Heap Space



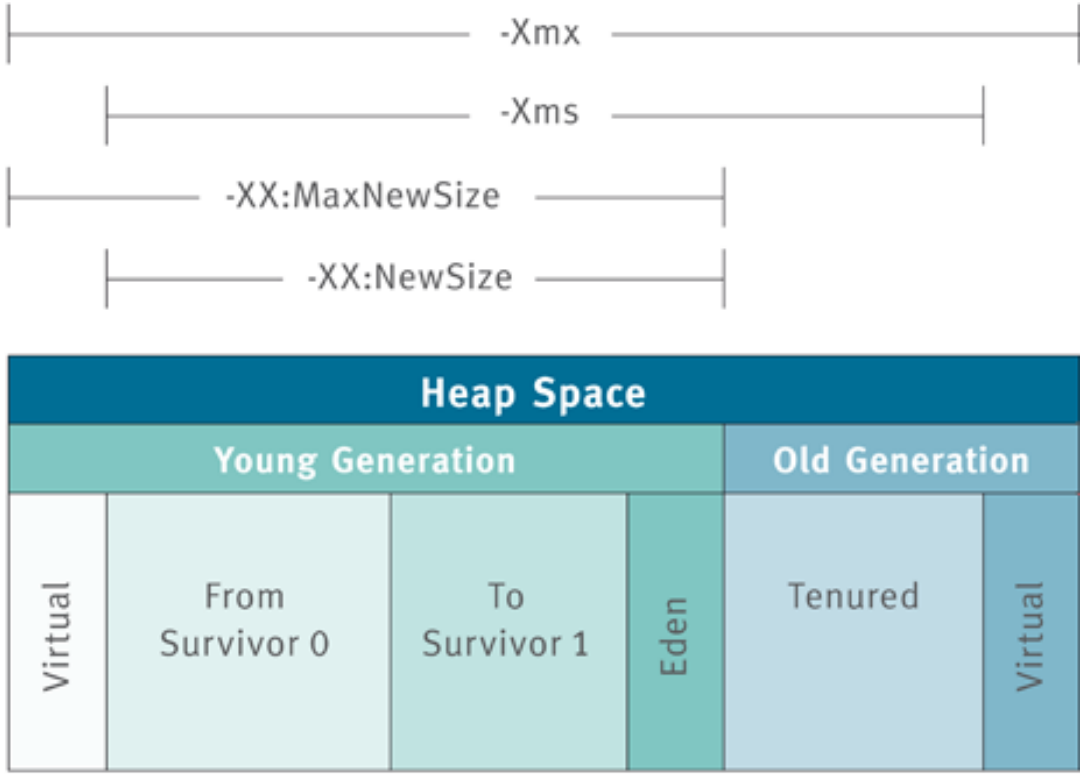
Young Generation

- 새롭게 생성되는 객체 인스턴스를 위한 공간이 할당되는 영역
- 대부분의 객체가 얼마 되지 않아 접근 불가 상태가 되므로 많은 객체 인스턴스가 이곳에서 생성 후 삭제됨

Old Generation

- Young Generation 영역에서 접근 가능 상태를 유지해 살아남은 객체가 복사돼 공간을 차지하는 영역
- 일반적으로 Young Generation 영역보다 공간을 크게 잡음

Heap Space



Young Generation

- 새롭게 생성되는 객체 인스턴스를 위한 공간이 할당되는 영역
- 대부분의 객체가 얼마 되지 않아 접근 불가 상태가 되므로 많은 객체 인스턴스가 이곳에서 생성 후 삭제됨

Old Generation

- Young Generation 영역에서 접근 가능 상태를 유지해 살아남은 객체가 복사돼 공간을 차지하는 영역
- **일반적으로 Young Generation 영역보다 공간을 크게 잡음**

가비지 컬렉션

- Java에서는 활용되지 않는 동적 할당된 메모리 공간을 JVM 실행 엔진의 **Garbage Collector**가 정리해 개발자가 메모리에 신경 쓰지 않고 개발에 집중할 수 있도록 도와줌
- C++11부터는 메모리 누수를 방지하기 위해 하나의 포인터만이 객체를 가리킬 수 있도록 하거나, 참조 개수를 기록하는 포인터로 참조 개수가 0이 되면 자동으로 `delete`를 실행해 메모리 공간을 해제하는 **스마트 포인터**를 지원함
- JVM 계열 외에도 Python, Ruby, JavaScript, Golang 등으로 만들어진 실행 파일에는 Garbage Collector가 포함됨

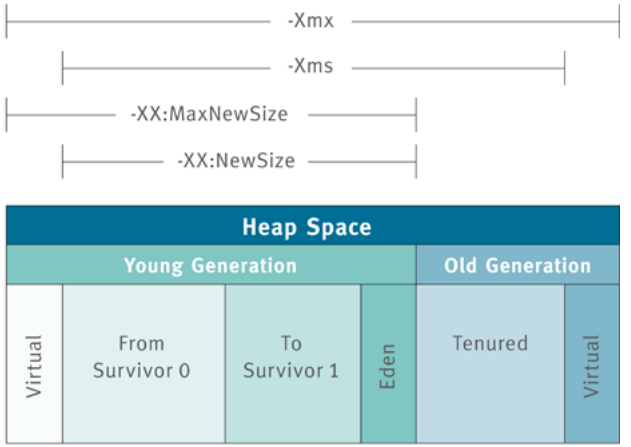
- 메모리 공간이 언제 해제되는지 정확하게 알 수 없어 제어하기 힘듦
- **Garbage Collection 처리 동안에는 JVM이 애플리케이션 실행을 멈추어 서비스가 중단될 수 있음 (Stop-The-World)**

08.
가비지 컬렉션 튜닝

가비지 컬렉션 튜닝의 목표

Major GC (Full GC) 빈도와 실행 시간을 줄이는 것이 목표!

- **객체 인스턴스를 재사용 또는 수명을 연장**
- **Old 영역으로 넘어가는 객체 수를 최소화**
  - Young 영역의 크기를 늘려 해결 가능
  - Young 영역의 크기가 너무 크면 Stop-The-World 시간이 길어짐
- **Major GC 시간을 최소화**
  - Old 영역의 크기를 줄여 해결 가능
  - Old 영역의 크기가 너무 작으면 OutOfMemoryError 또는 Major GC 횟수가 증가



가비지 컬렉션 튜닝의 목표

Major GC (Full GC) 빈도와 실행 시간을 줄이는 것이 목표!

- **다른 가비지 컬렉션 알고리즘을 사용하는 최적의 가비지 컬렉터로 변경**
  - 가비지 컬렉션 스레드가 하나인 Serial GC (Stop-The-World 시간이 가장 길며 성능이 별로인 환경에서 사용)
  - Minor GC를 멀티 스레드, Major GC를 싱글 스레드로 처리하는 Parallel GC (Java 8의 기본 GC)
  - Minor GC와 Major GC 모두 멀티 스레드로 수행하는 Parallel Old GC
  - 애플리케이션 스레드와 GC 스레드를 동시에 실행해 복잡한 GC 과정과 메모리 파편화가 일어나는 Concurrent Mark Sweep GC (Java 9부터 deprecated되어 Java 14부터 사용 중지)
  - 전체 힙 영역을 고정된 Young / Old 영역으로 분할하지 않고 Region이라는 영역으로 나누어 Eden, Survivor, Old 역할을 동적으로 부여하는 Garbage First GC (Java 9+의 기본 GC)
  - 큰 GC를 적은 횟수로 수행하기보다 작은 GC를 여러 번 수행해 일정한 STW 시간의 Shenandoah GC (Java 12에서 출시)
  - G1 GC의 Region은 고정된 크기였으나 Zpage라는 2MiB 배수로 크기가 지정되는 Z Garbage Collector (Java 15에서 출시)

감사합니다

Email / park@duck.com
Insta / @yeonjong.park
GitHub / patulus

Photo by Pawel Czerwinski