Memory Allocation in C/C++

1) life & scope of primitive variables & reference variables - (same for C++ & Java)

2) life & scope of Array / String literals / objects.

3) (Dynamic Memory Allocatn) → new / malloc
& delete (dynamic memory Allocatn) → delete / free } in C++/C

4) Memory leaks in C/C++ (consumed memory & free memory)

5) Deadlocks & Starvation in C/C++

6) Constructors & Destructors in C++
(new) (delete)

Garbage Collection in Java

1) Automatic Garbage Collection → Garbage collector

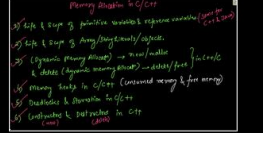
2) no delete keyword, no destructors in Java!

3) Unreferenced objects → garbage collectible/dead or abandoned objects → 3ways

4) JVM → daemon thread → garbage collector
   → main thread (user thread)
   → Thread scheduler (daemon/background method)

5) On the spot OR cummulative ?
   ↳ on memory fragmentation!

6) Garbage Collector Working



finalize() method

(1) Similar to destructor in C++

(2) Runs before object is about to destroy

(3) Present in Object class → can be overided to provide clean up of System resources (eg database, file, I/ostream, Hardware, etc)

(4) Called automatically only once by JVM (Garbage Collector thread)

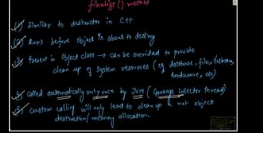
(5) custom calling will only lead to clean up & not object destruction/ memory allocation.

(6) Exceptions in finalize method are ignored/not propogated by garbage collector thread

(7) There is no chaining of finalize() method implicitly like for constructor chaining in Java and destructor chaining in C++.

Forceful execution of finalize() method

- Runtime.getRuntime().
- System.gc()

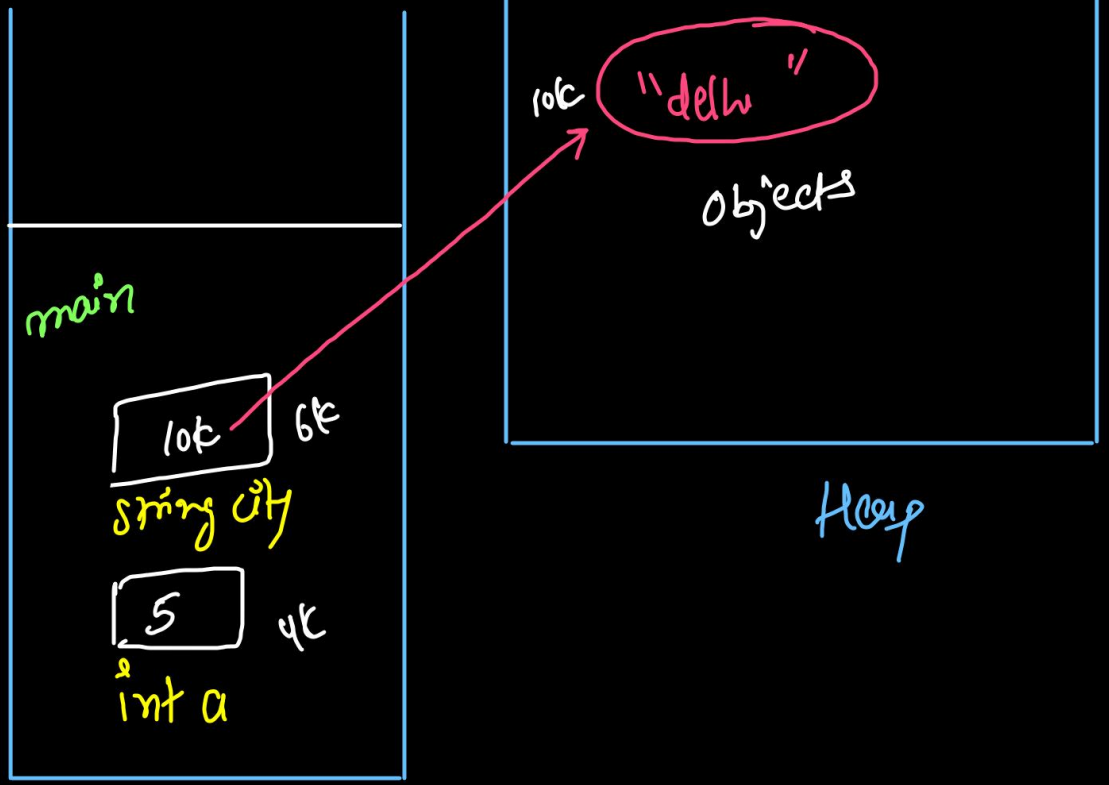


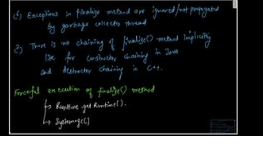
Memory Allocation of Primitive/Reference Variables

* Stack
  - Primitive Variables
  - Reference Variables

Scope :→ Functn / Block scope

Lifetime :→ Function Push in Stack
↳ create/Allocate
Functn Pop in Stack
↳ Deallocate/Release





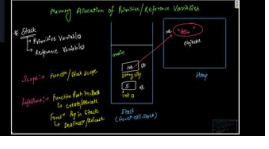
```java
public static void fun() {
    System.out.println(x: "Primitive & Reference Variables Creation / Allocation");
    // Function Scope Local Variables:
    int a = 5;
    String str = "hello";

    for (int idx = 0; idx < 5; idx++) {
        // Index is accessible within for loop only
        System.out.print(idx + " ");
    }

    if (a % 2 == 0) {
        String even = "even";
        System.out.println(even);
    } else {
        String odd = "odd";
        System.out.println(odd);
    }

    System.out.println(x: "Primitive & Reference Variables Deletion / Deallocation");
}
```

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % javac 07.GarbageCollection.java
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Primitive & Reference Variables Creation / Allocation
0 1 2 3 4 odd
Primitive & Reference Variables Deletion / Deallocation
```



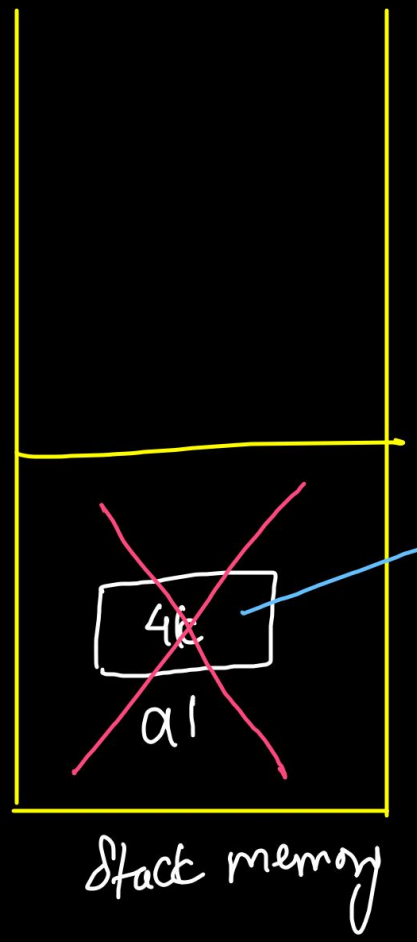
Memory Allocation of Heap Variables

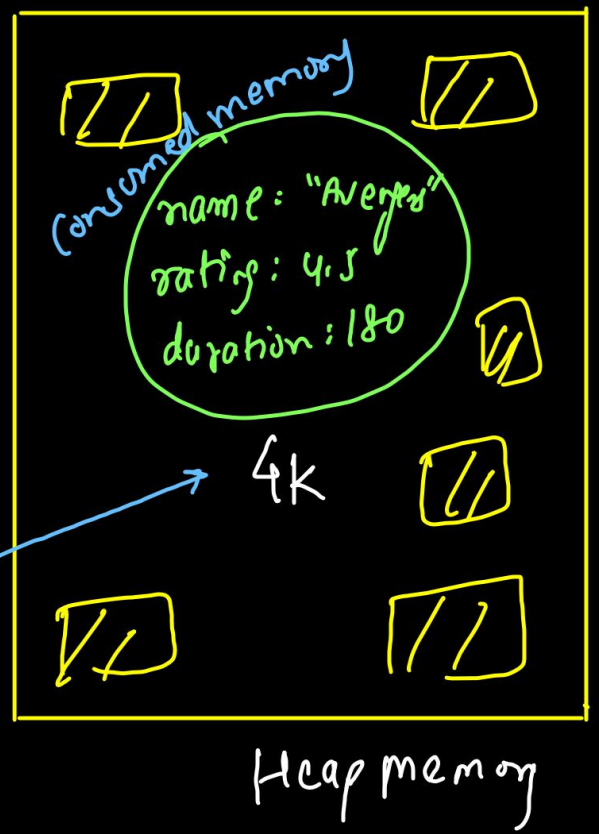
C/C++

Movie a1 = new Movie(); (Constructor)

fragmented memory
↳ memory leak
- on Heap
- Developers mistakes
- Objects were not deallocated from heap

delete a1; → explicit deallocation
↳ destructor





Movie Ticketing App

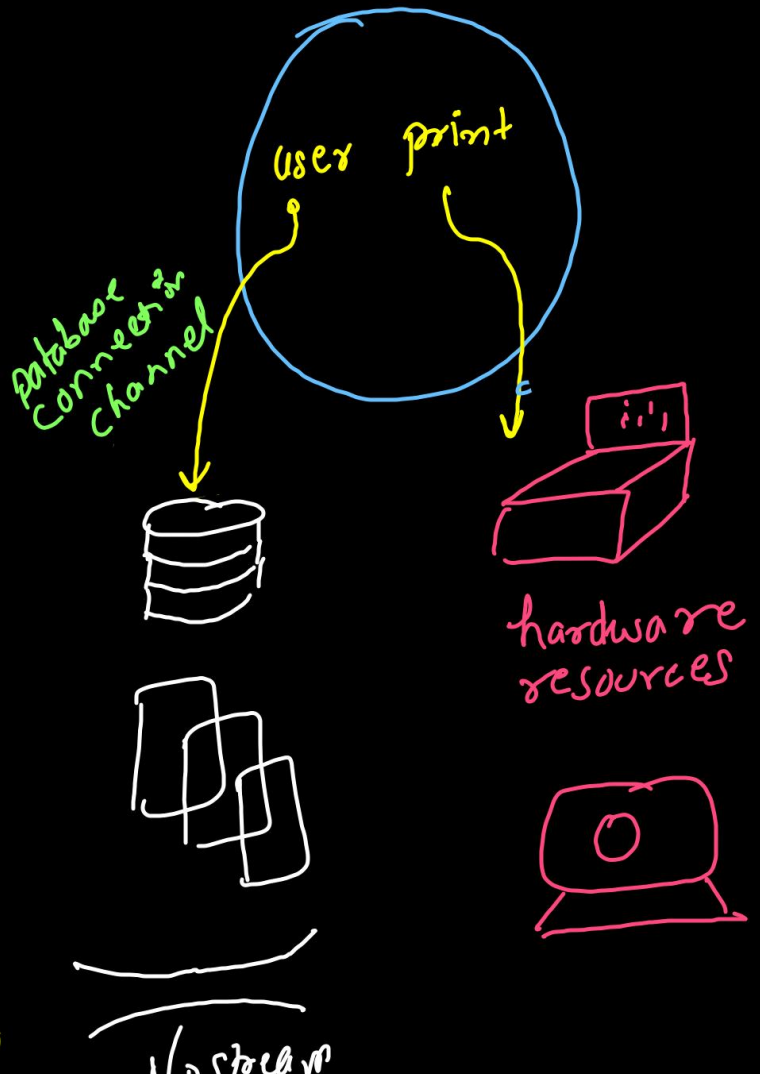
memory Deallocation (&)

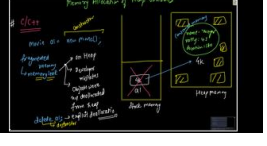
- Hardware Resources
- Database Connection

} Decoupling &

Starvation Deadlock

for other objects/applications!





Automatic Garbage Collection in Java.

Stack

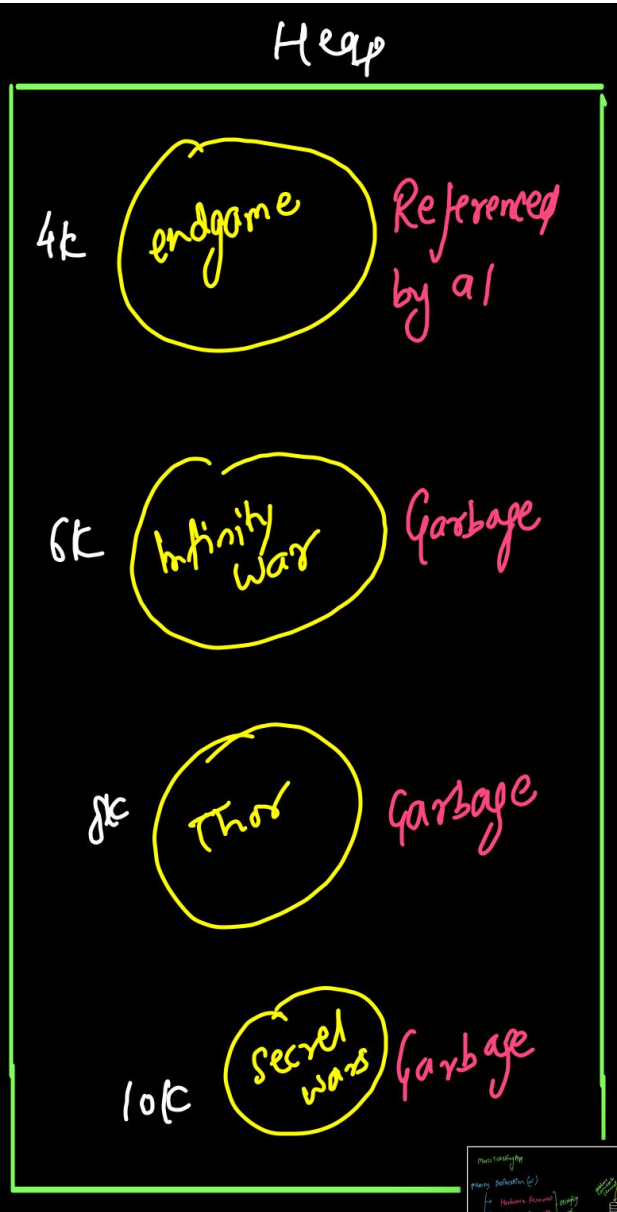
a1 = 4k a2 = ~~6k~~ null a3 = ~~9k~~ 9k

```java
public static void automaticGarbageCollection() {
    Movie a1 = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    // Object cannot be deleted because it is referenced

    Movie a2 = new Movie(duration: 150, name: "Infinity War", rating: 4.2);
    a2 = null; // 1. Nulling the Reference

    Movie a3 = new Movie(duration: 120, name: "Thor", rating: 2.5);
    a3 = a1; // 2. Updating the Reference

    new Movie(duration: 120, name: "Secret Wars", rating: 2.5);
    // 3. Anonymous Object
}
```



```java
class Movie {
    static int countCreation = 0;
    static int countDeletion = 0;
    int duration;
    String name;
    double rating;

    public Movie(int duration, String name, double rating) {
        System.out.println(x: "Memory Allocation - Initialization of Variables");
        countCreation++;
        this.duration = duration;
        this.name = name;
        this.rating = rating;
    }

    @Override
    public void finalize() throws Throwable {
        System.out.println(x: "Memory Deallocation");
        countDeletion++;
    }
}
```

```java
public static void automaticGarbageCollection() {
    Movie a1 = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    // Object cannot be deleted because it is referenced

    Movie a2 = new Movie(duration: 150, name: "Infinity War", rating: 4.2);
    a2 = null; // 1. Nulling the Reference

    Movie a3 = new Movie(duration: 120, name: "Thor", rating: 2.5);
    a3 = a1; // 2. Updating the Reference

    new Movie(duration: 120, name: "Secret Wars", rating: 2.5);
    // 3. Anonymous Object

    // Garbage Collector: Memory Deallocation:
    // Automatic, Cummulative, On Memory Leak Avoidance
    for (int idx = 0; idx < 1000000; idx++) {
        new Movie(duration: 120, name: "Temp", rating: 2.5);
    }

    System.out.println(Movie.countCreation + " " + Movie.countDeletion);
}
```

≈ 10 lakh

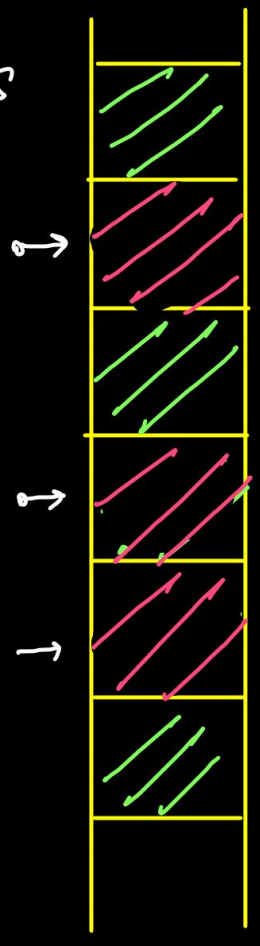
≈ 5.25 lakh

Garbage Collector Working

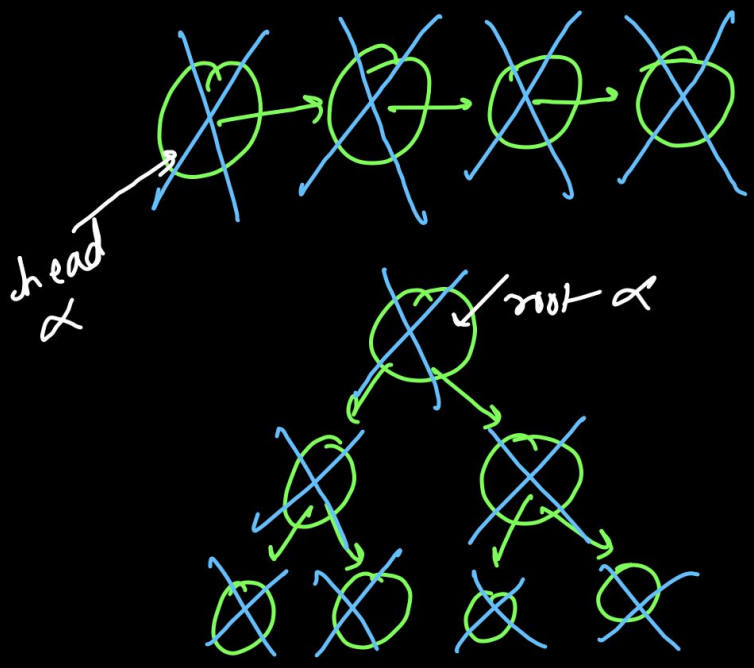
(i) Marking the unreachable objects

(ii) Destroying them : free the memory

(iii) Compaction of memory

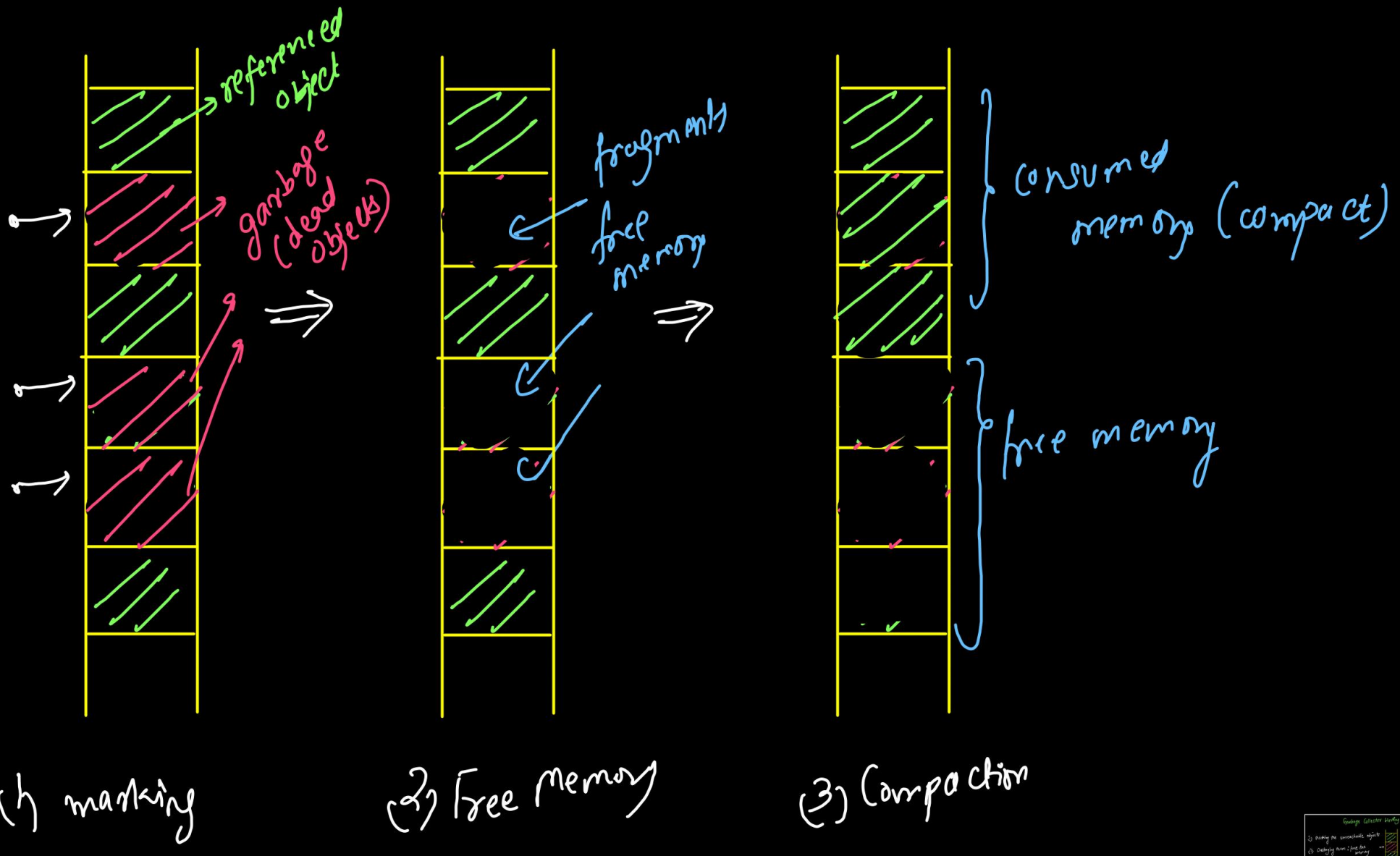


(1) GC roots



≡ Referenced

≡ Un


referenced object
garbage (dead objects)
fragments
free memory
consumed memory (compact)
free memory
(1) marking
(2) Free memory
(3) Compaction

```java
class Movie {
    static int countCreation = 0;
    static int countDeletion = 0;
    int duration;
    String name;
    double rating;
    FileInputStream ticket;

    public Movie(int duration, String name, double rating) throws Exception {
        System.out.println(x: "Memory Allocation - Initialization of Variables");
        countCreation++;

        this.duration = duration;
        this.name = name;
        this.rating = rating;

        // System Resources Couple
        ticket = new FileInputStream(name: "/Users/architaggarwal/Documents/DSA-And-LLD/LLD or OOAD/ticekt.txt");
    }

    @Override
    public void finalize() throws Throwable {
        System.out.println(x: "Memory Deallocation");
        // System Resources Decouple: Clean Up Code
        ticket.close();
        countDeletion++;
    }
}
```

```java
public static void finalizeDemo() throws Exception {
    Movie a = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    a.finalize();
}
```

↗ object Creation

↳ object Deletion (assume)

```java
class Movie {
    static int countCreation = 0;
    static int countDeletion = 0;
    int duration;
    String name;
    double rating;
    Scanner scn;

    public Movie(int duration, String name, double rating) throws Exception {
        System.out.println(x: "Memory Allocation - Initialization of Variables");
        countCreation++;

        this.duration = duration;
        this.name = name;
        this.rating = rating;
        You, 1 second ago • Uncommitted changes
        // System Resources Couple
        scn = new Scanner(System.in);
    }

    @Override
    public void finalize() throws Throwable {
        System.out.println(x: "Memory Deallocation");
        // System Resources Decouple: Clean Up Code
        scn.close();
        countDeletion++;
    }
}
```

Last code before Object Deallocation

↳ clean up code

↳ System resources release

```java
public static void finalizeDemo() throws Exception {
    Movie a = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    try {
        a.finalize();
    } catch (Throwable e) {
    }
}
```

Driver

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % javac 07.GarbageCollection.java
Note: 07.GarbageCollection.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Memory Allocation - Initialization of Variables
Memory Deallocation
```

Output

```java
public static void finalizeDemo() throws Exception {
    Movie a = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    try {
        a.finalize();
    } catch (Throwable e) {
    }

    System.out.println(a); // It will still print the object: Object is still there in memory
    // Object Deallocation due to Developer's finalize Call is not happening      You, 1 secon
}
```

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Memory Allocation - Initialization of Variables
Clean Up Code
Movie@5a39699c
```