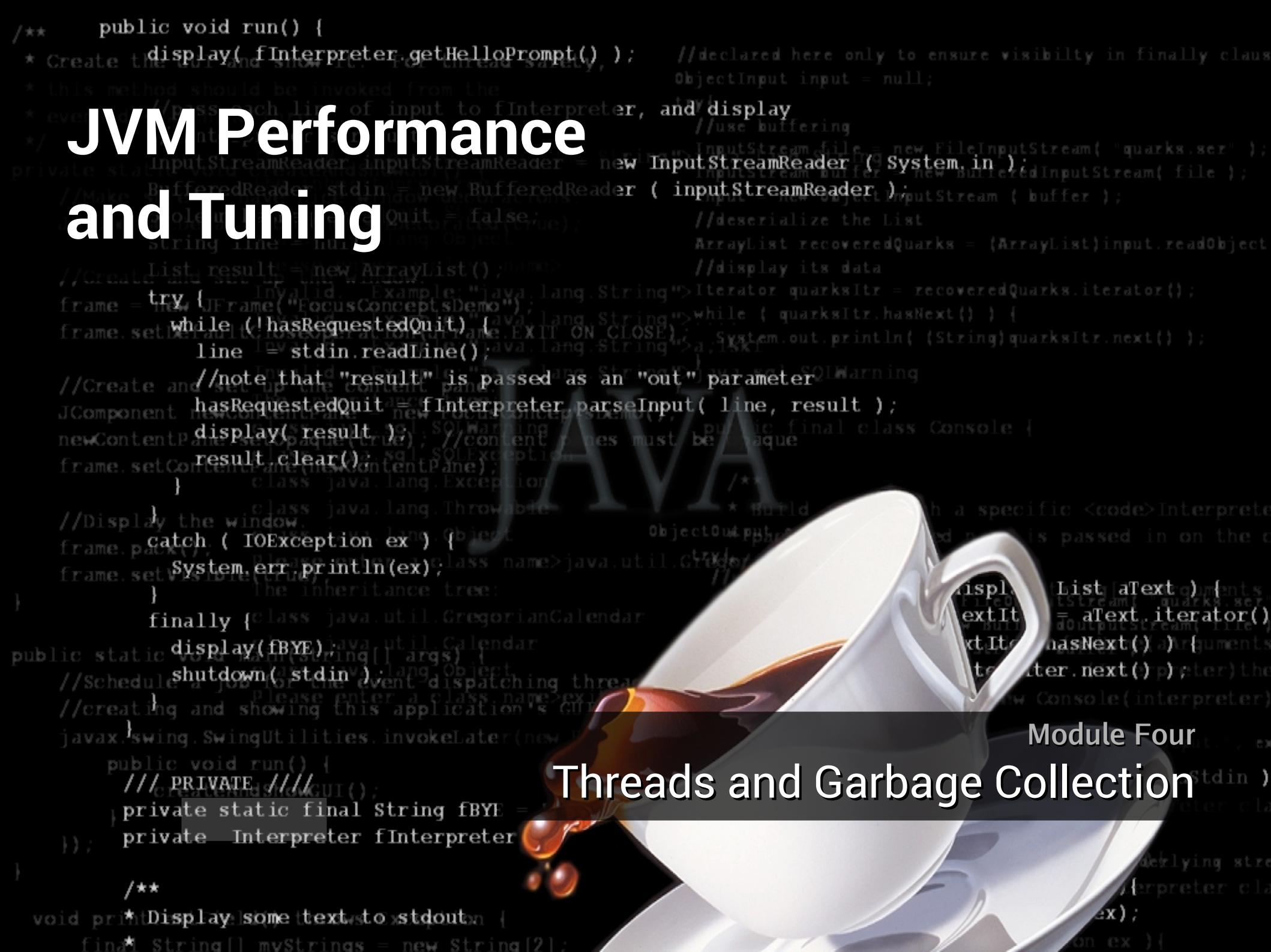
JVM Performance and Tuning



Module Four

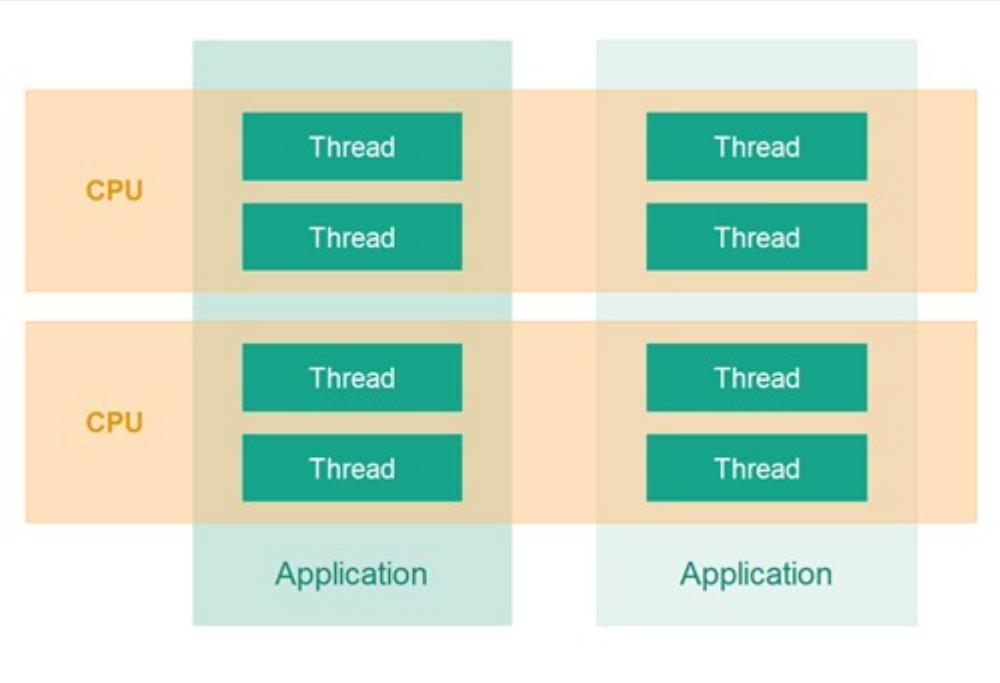
Threads and Garbage Collection

Module Topics

1. Threads
2. Garbage Collection

Threads

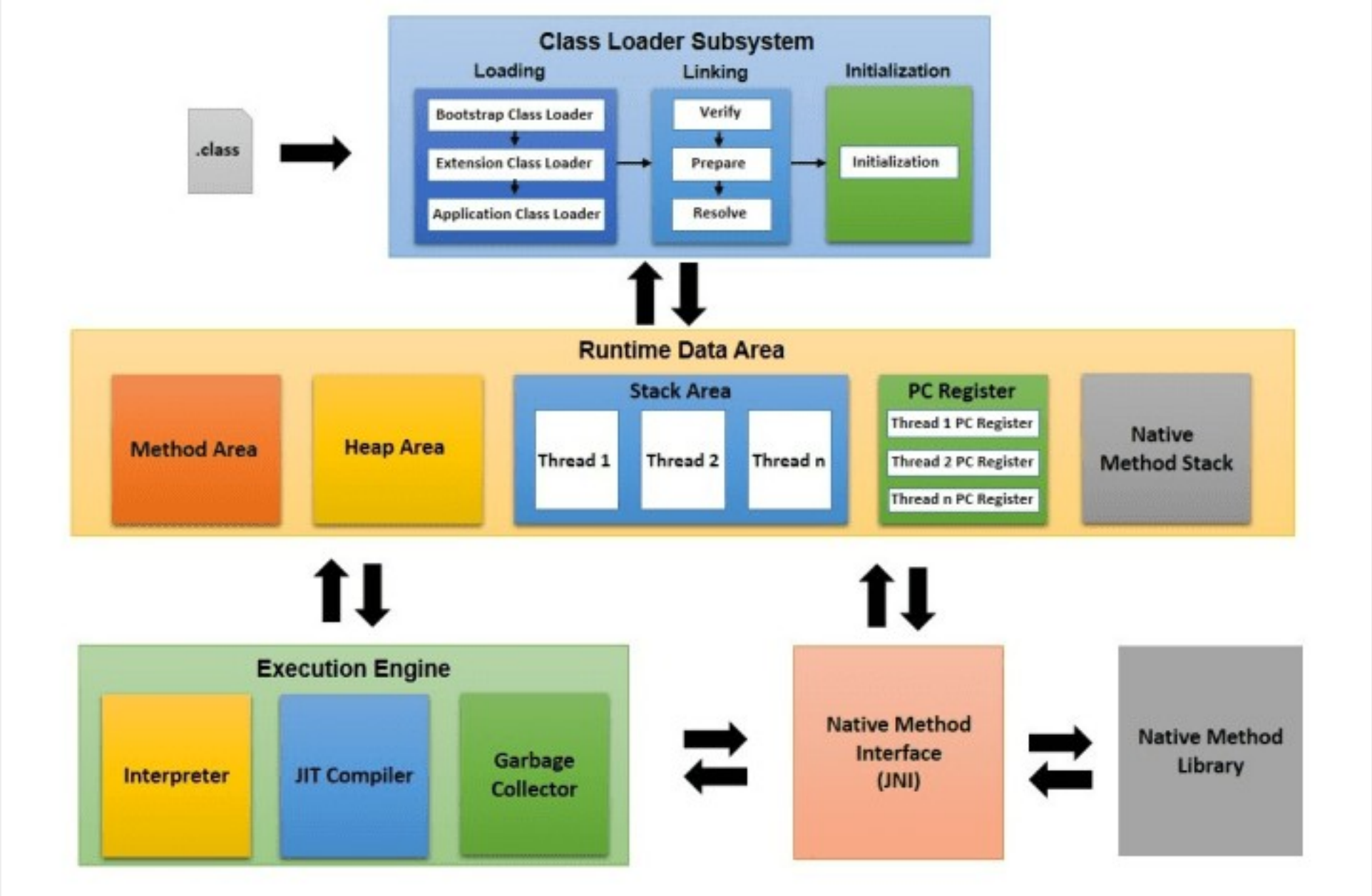
- A CPU may have multiple threads of execution
  - Each thread has exclusive access to the CPU for a time slice
  - In a multi-core environment, a thread may use different cores
-



Why Use Threads

- Generally applications use different resources
  - Threads allow for more effective utilization of resources
  - Allows for more effective use of multiple processors
- MultiThreading introduces a layer of complexity
  - How thread are managed can impact performance
  - Thread have their own stack areas but share the heap

JVM Architecture



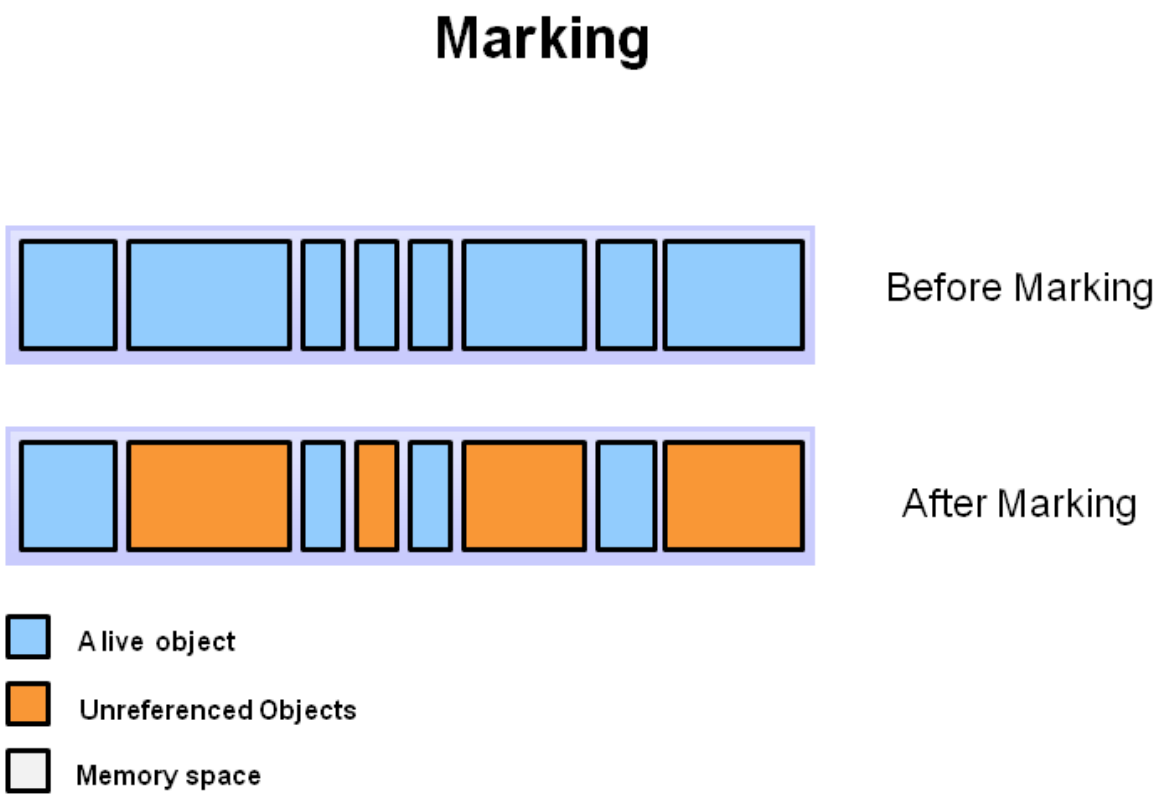
Non–Daemon vs Daemon Threads

- Non-daemon threads
  - Also known as “user” threads or “non-native” threads
  - The main() statement starts a use thread
  - The JVM exits when the last non-daemon thread ends
- Daemon threads
  - Ending a daemon thread does not cause the JVM to exit
  - Manage a lot of the internal JVM tasks
    - *Eg. Garbage collection*

Module Topics

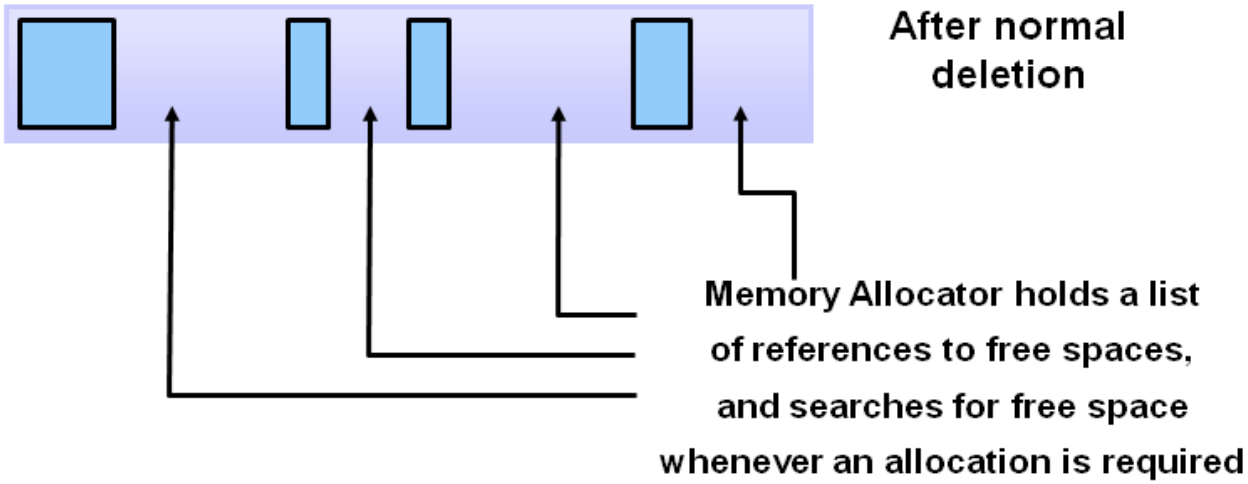
1. Threads
2. Garbage Collection

Step 1: Marking



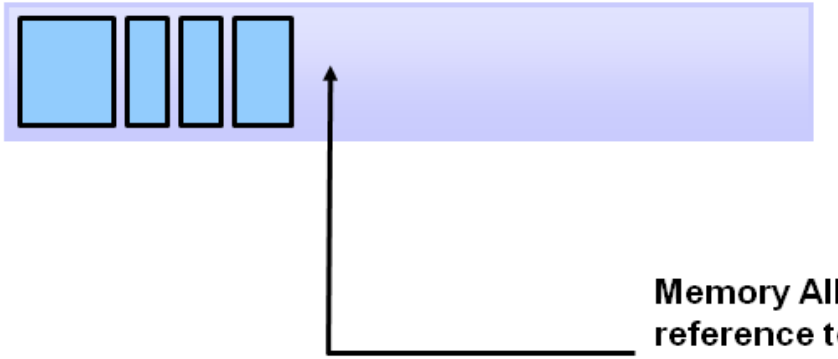
Step 2: Normal Deletion

Normal Deletion



Step 2a: Deletion with Compacting

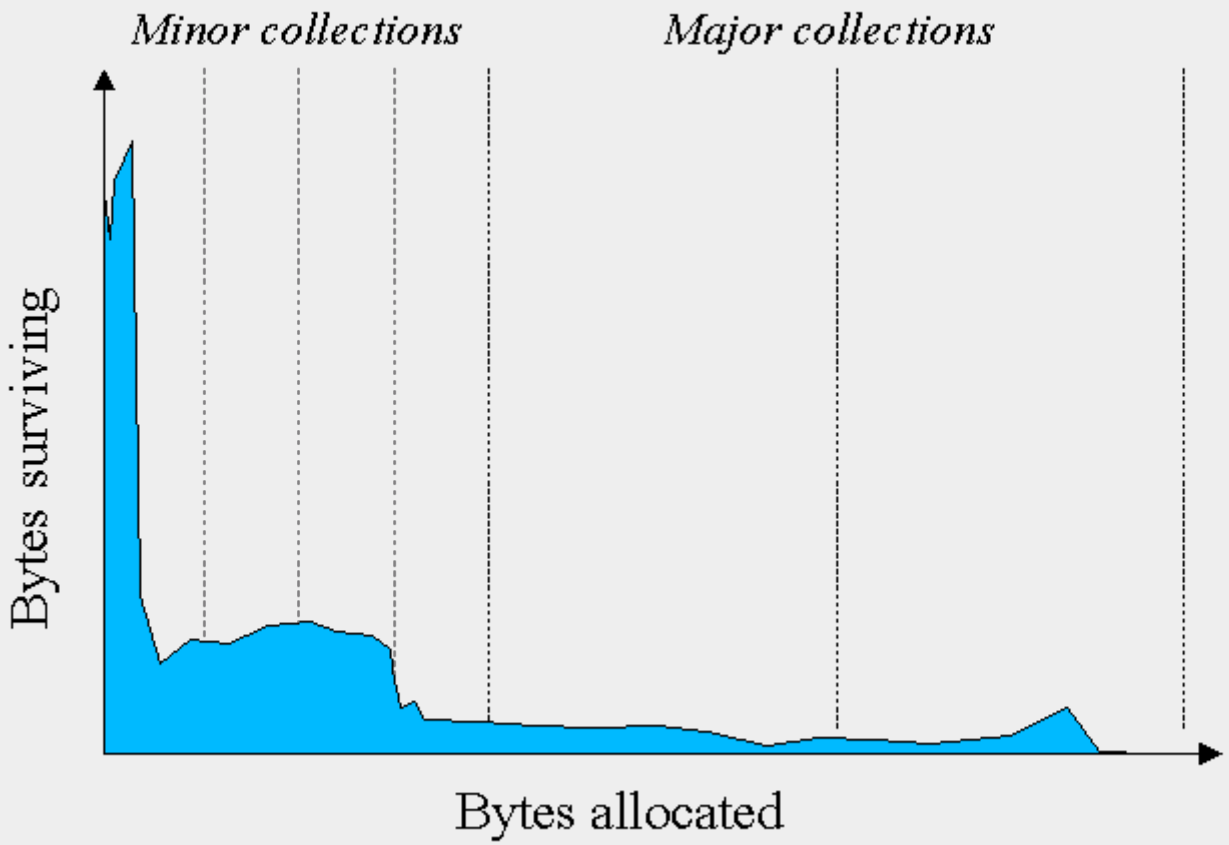
Deletion with **Compacting**



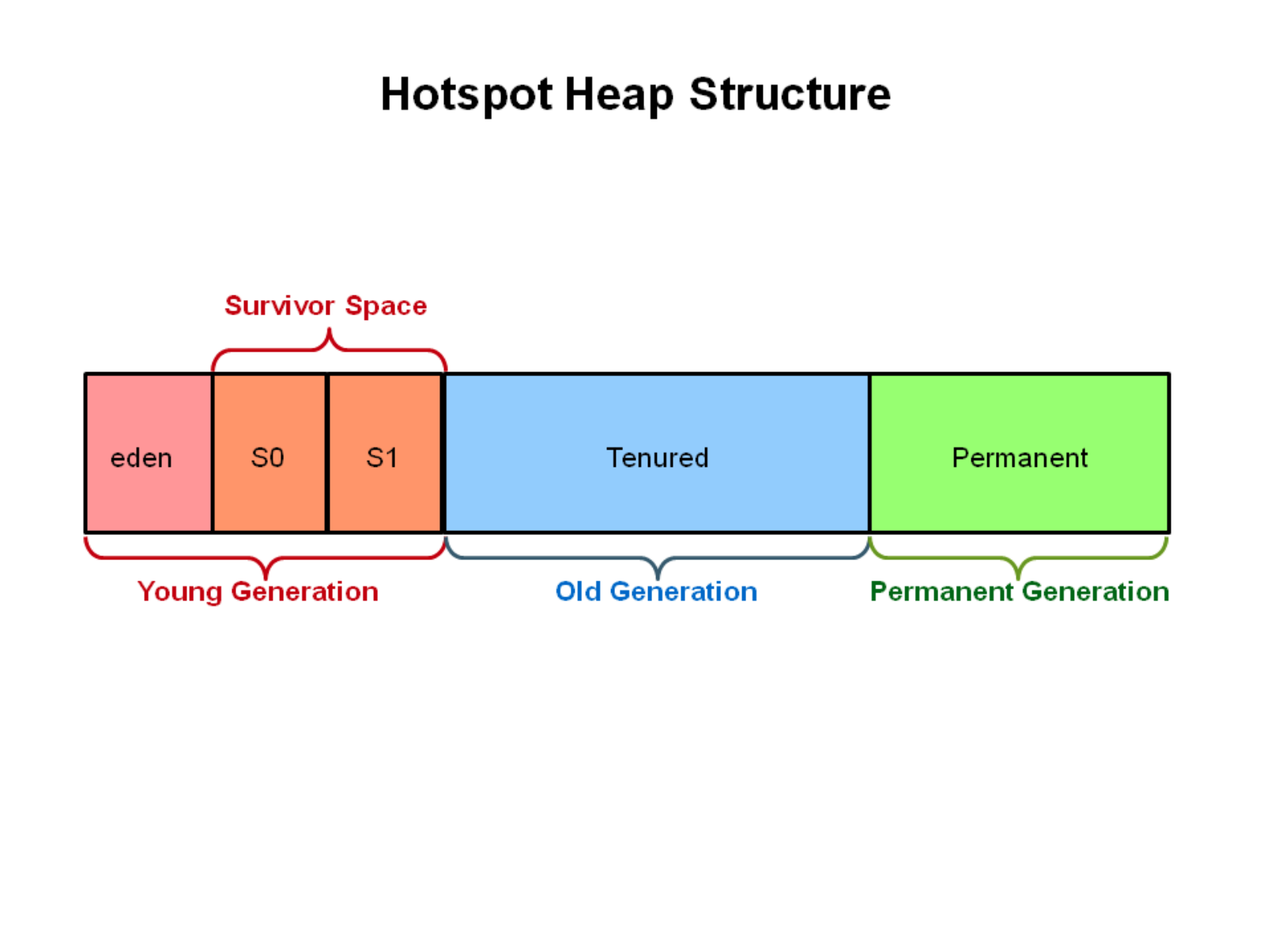
After normal Deletion with compacting

Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

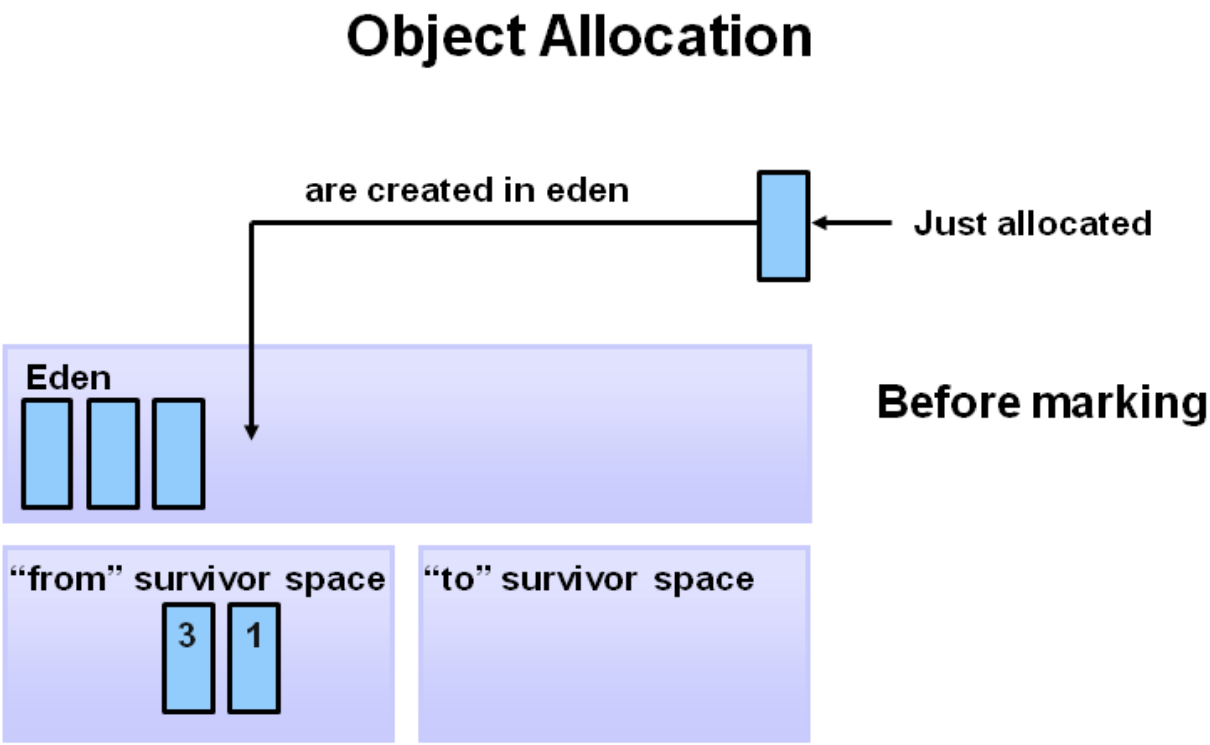
Why Generational Garbage Collection?



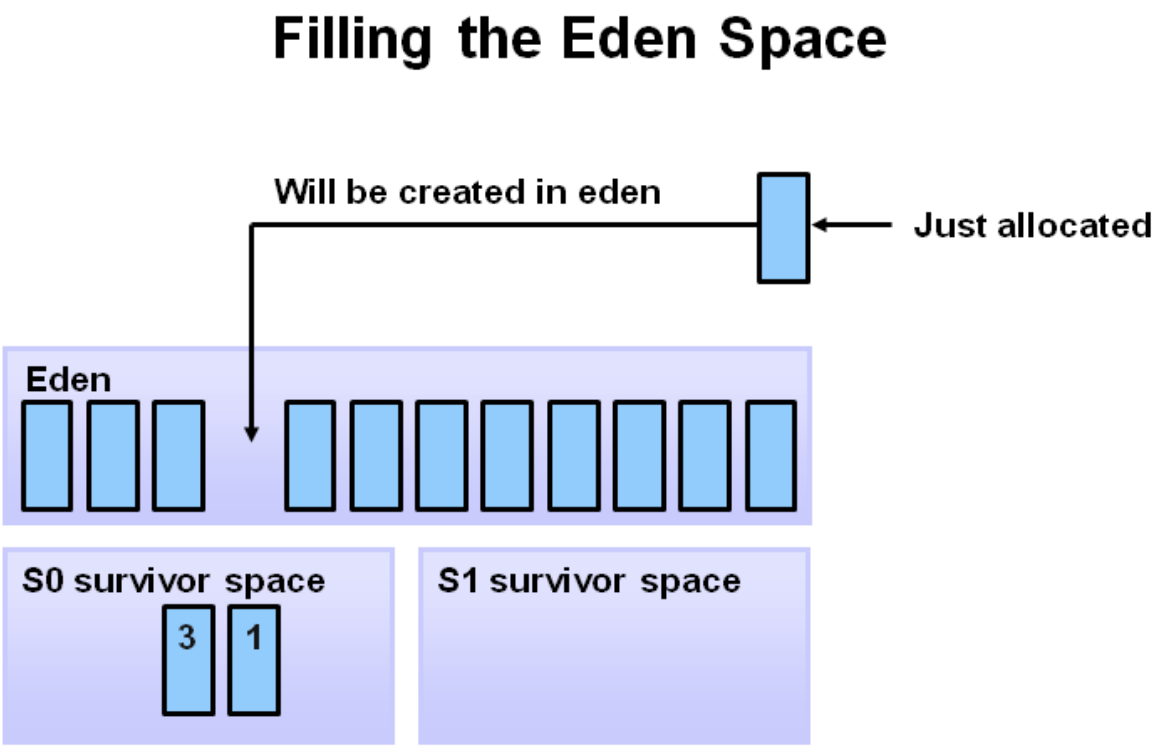
JVM Generations



Object Allocation

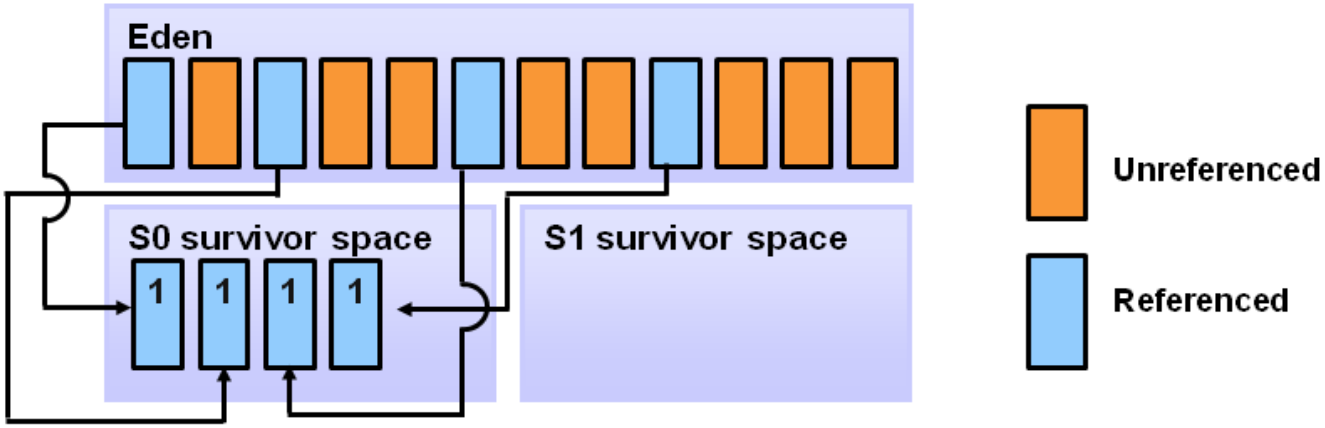


Triggering Minor GC

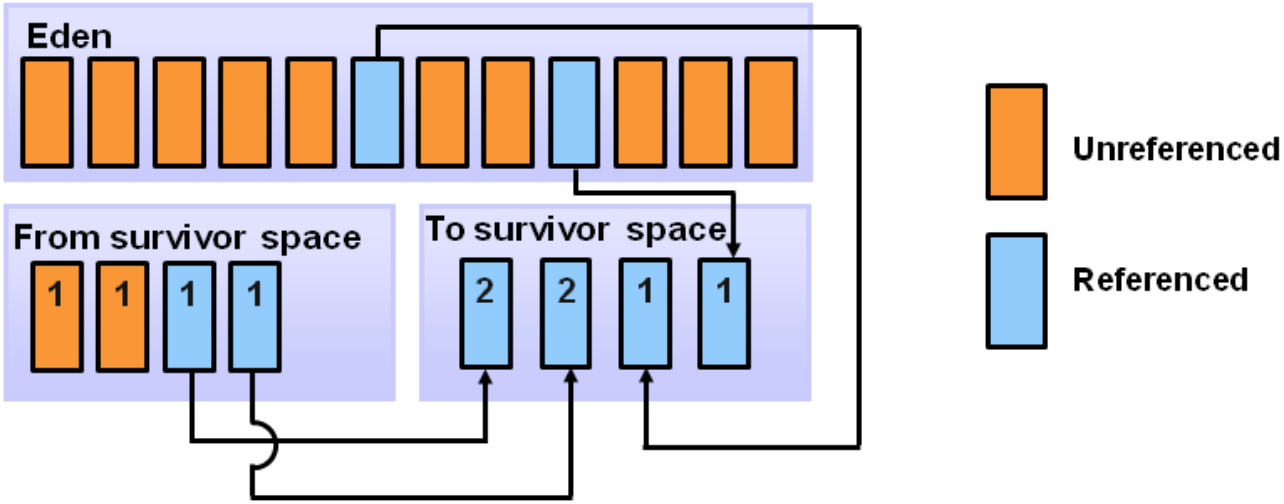


Executing Minor GC

Copying Referenced Objects

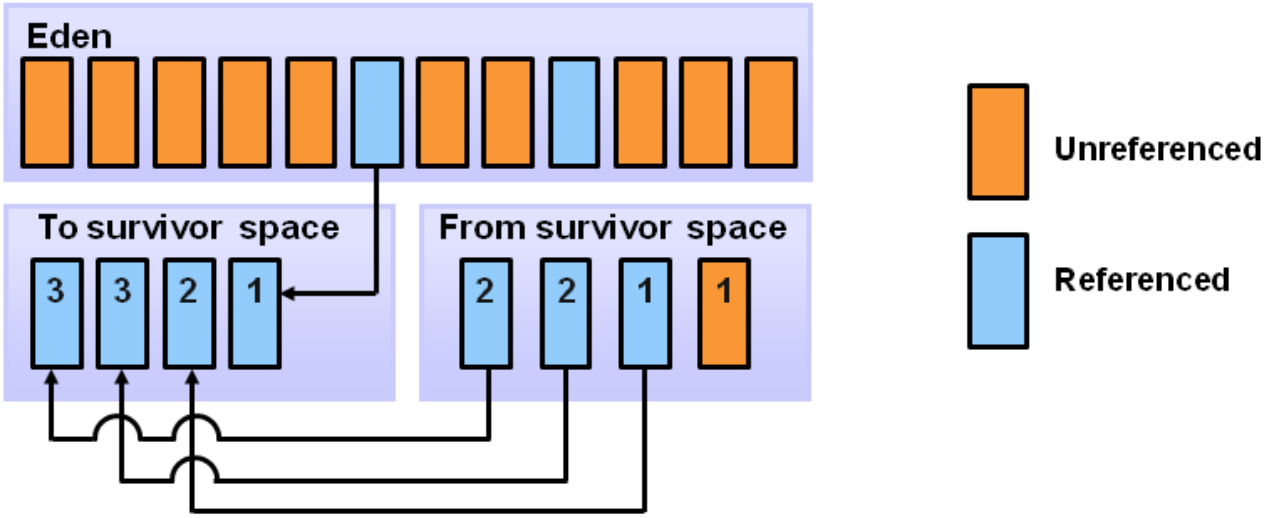


Next Minor GC

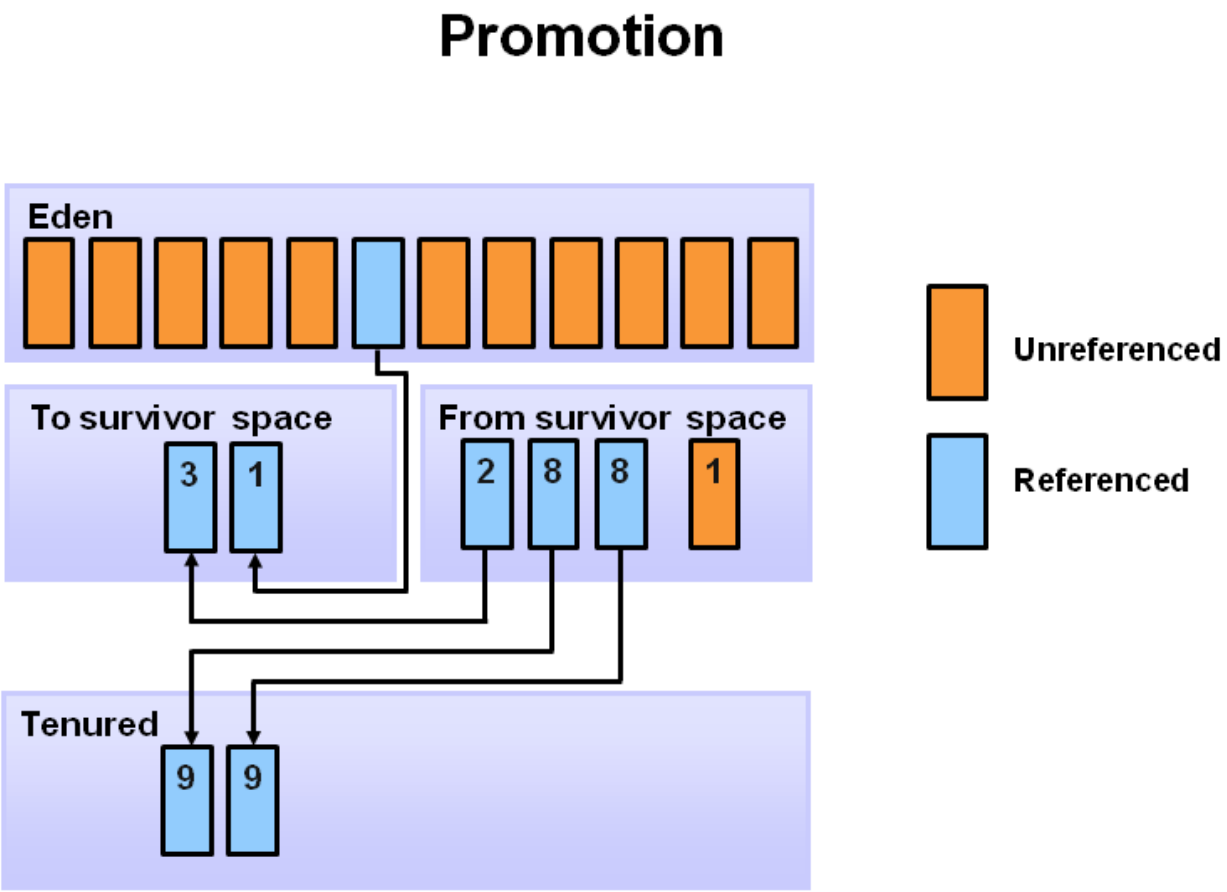


Next Minor GC

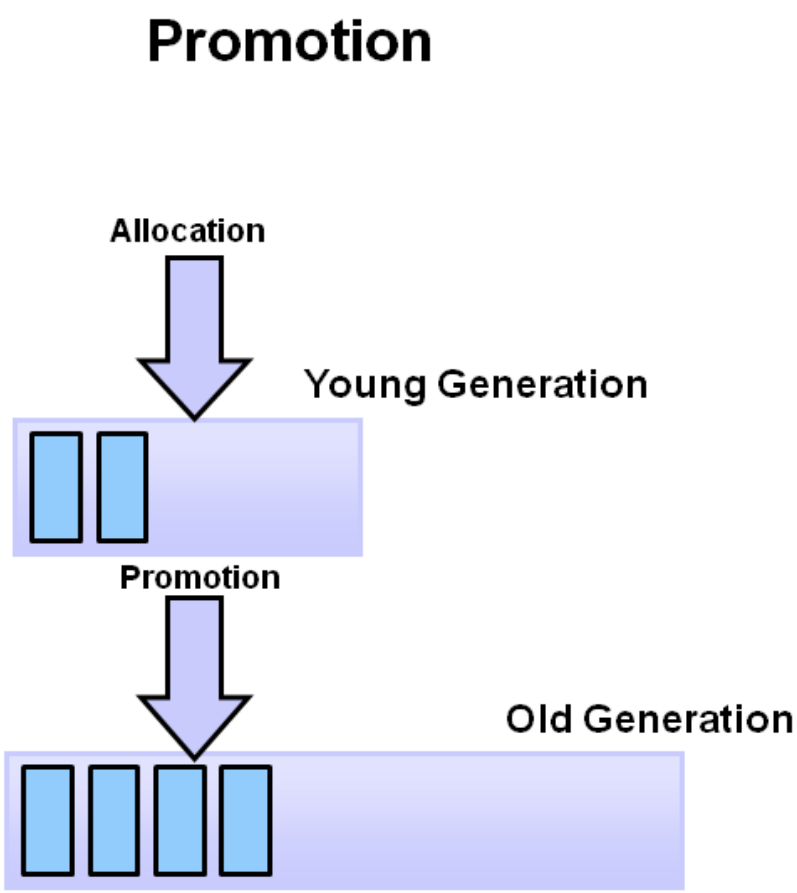
Additional Aging



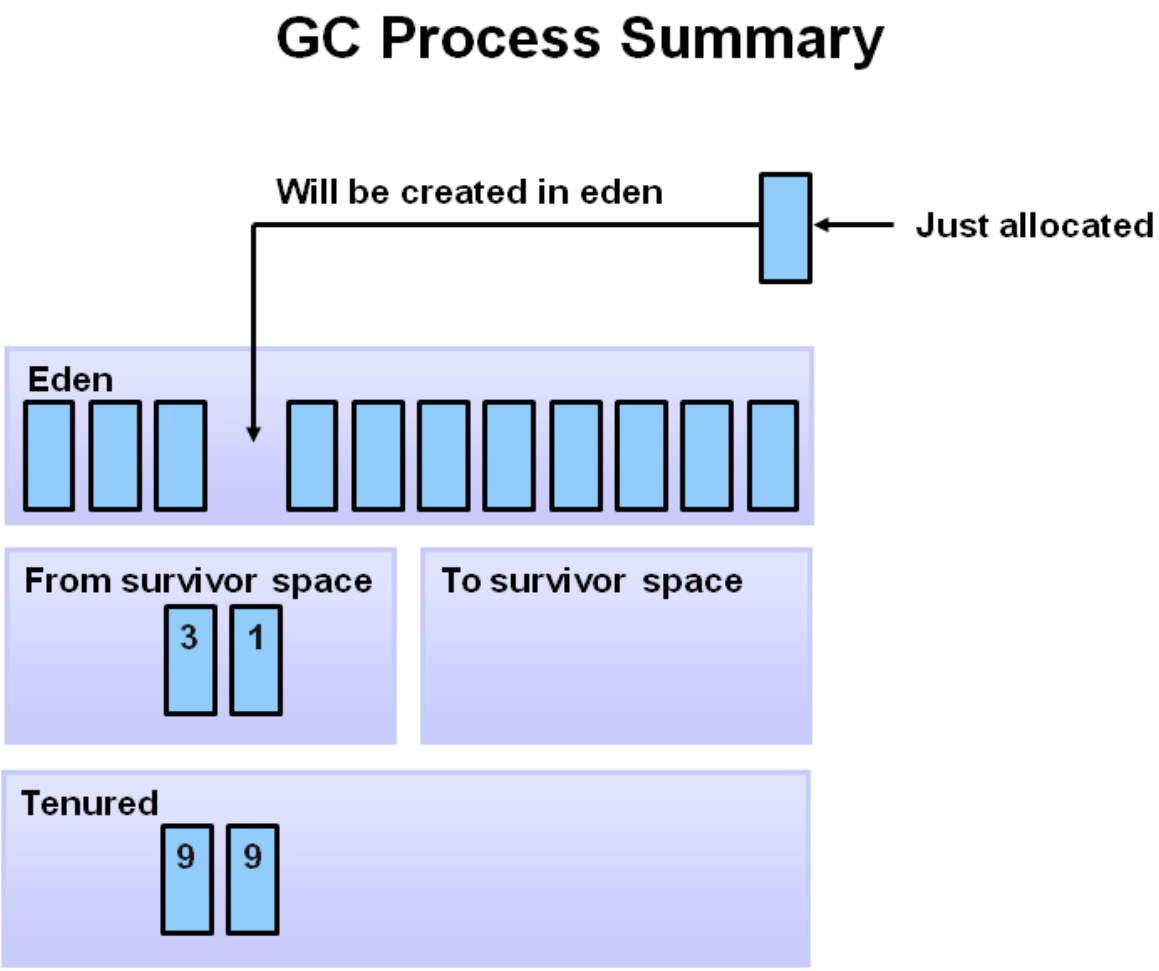
Move to Tenured Storage



Continued Promotion



Major GC


GC Process Summary
Will be created in eden
Just allocated
Eden
From survivor space
3
1
To survivor space
Tenured
9
9

Module End