Java™

ORACLE®

JVM Troubleshooting MOOC:

Troubleshooting Memory Issues in Java Applications

Poonam Parhar
JVM Sustaining Engineer
Oracle

Java
Your
Next
(Cloud)

Lesson 1

HotSpot JVM Memory Management

Poonam Parhar
JVM Sustaining Engineer
Oracle

Java
Your
Next
(Cloud)

Agenda

1. Automatic Memory Management
2. Generational Garbage Collection and Memory Spaces in the HotSpot JVM
3. Garbage Collectors in the HotSpot JVM
4. Garbage Collection Changes Between Java 7, 8 and 9

Lesson 1-1

Automatic Memory Management

Poonam Parhar
JVM Sustaining Engineer
Oracle

Java
Your
Next
(Cloud)

Automatic Memory Management

- Memory Management ? process of allocating, determining when objects are not referenced, and then de-allocating them
- Explicit memory management in some programming languages
- HotSpot JVM employs automatic memory management
- Managed by a sub-system called the garbage collector
- The garbage collector in the HotSpot JVM automatically manages memory freeing the programmer from worrying about the object de-allocations

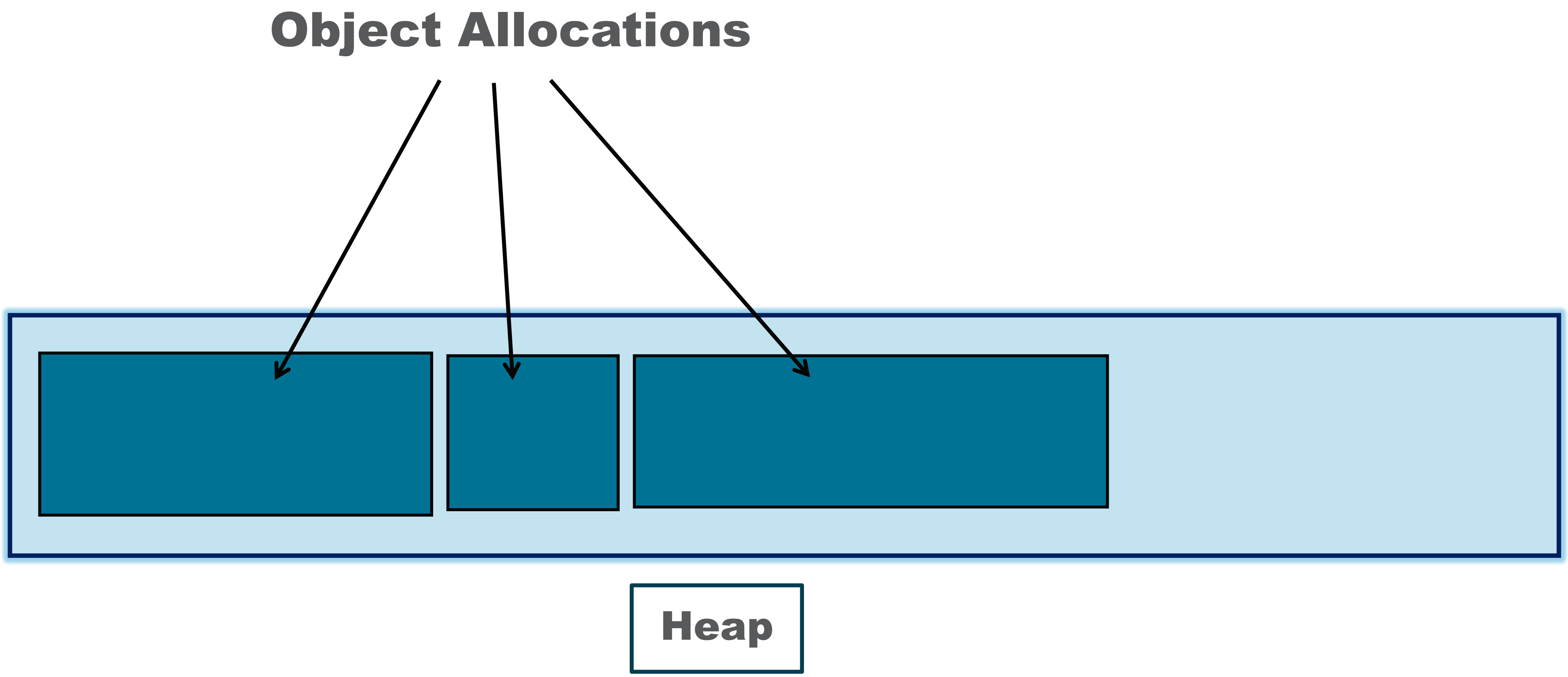
Memory allocation examples

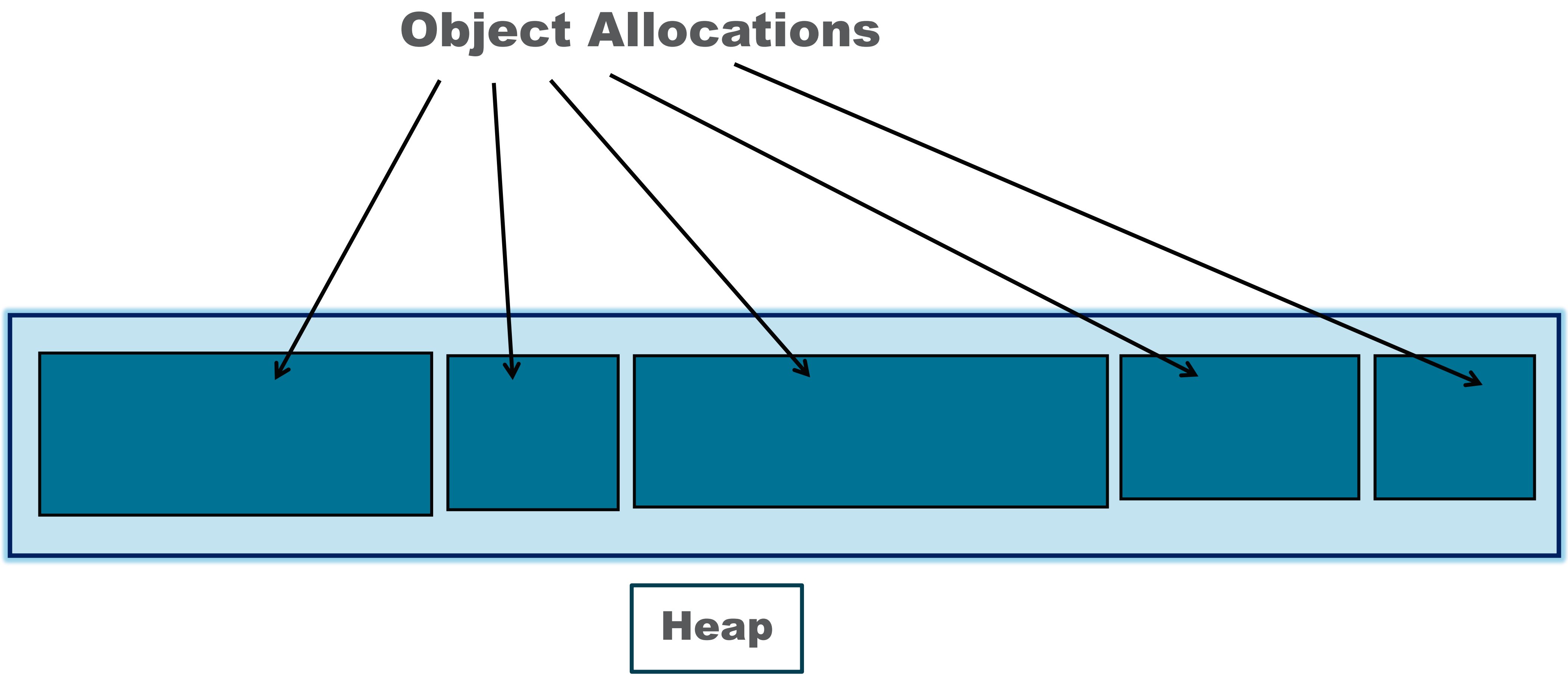
Example in C language:

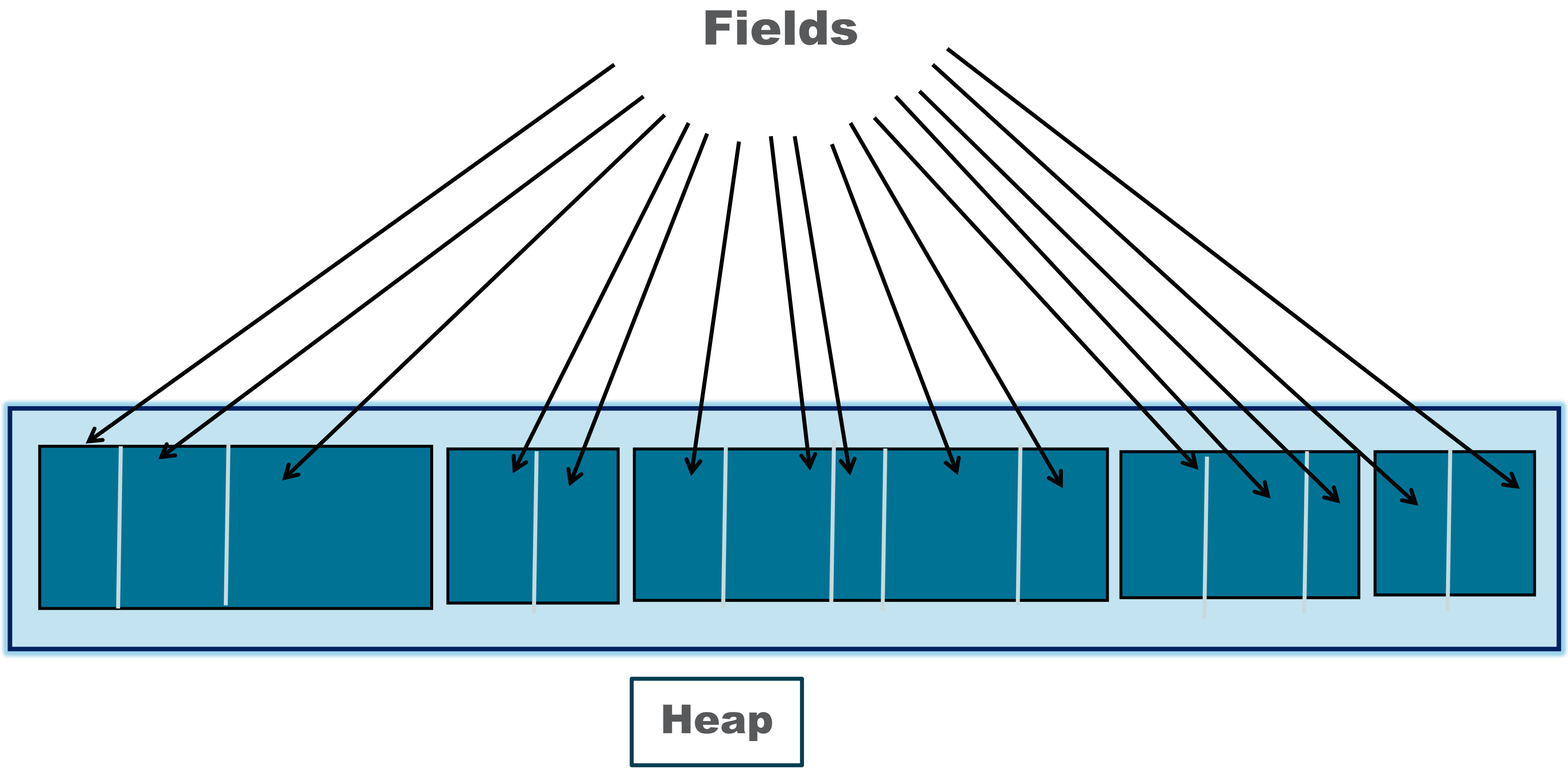
```
//allocate memory
int* array = (int*)malloc(sizeof(int)*20);

//explicitly deallocate memory
free(array);
```

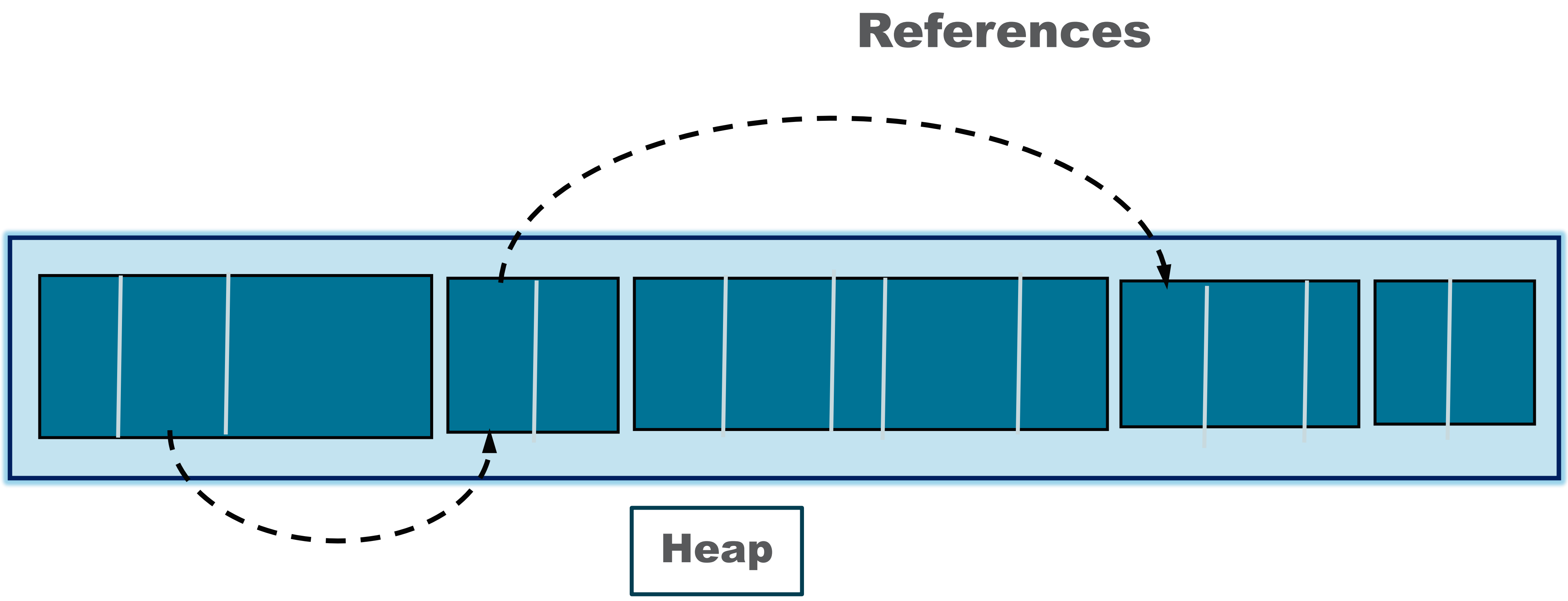
Example in Java that employs automatic memory management:

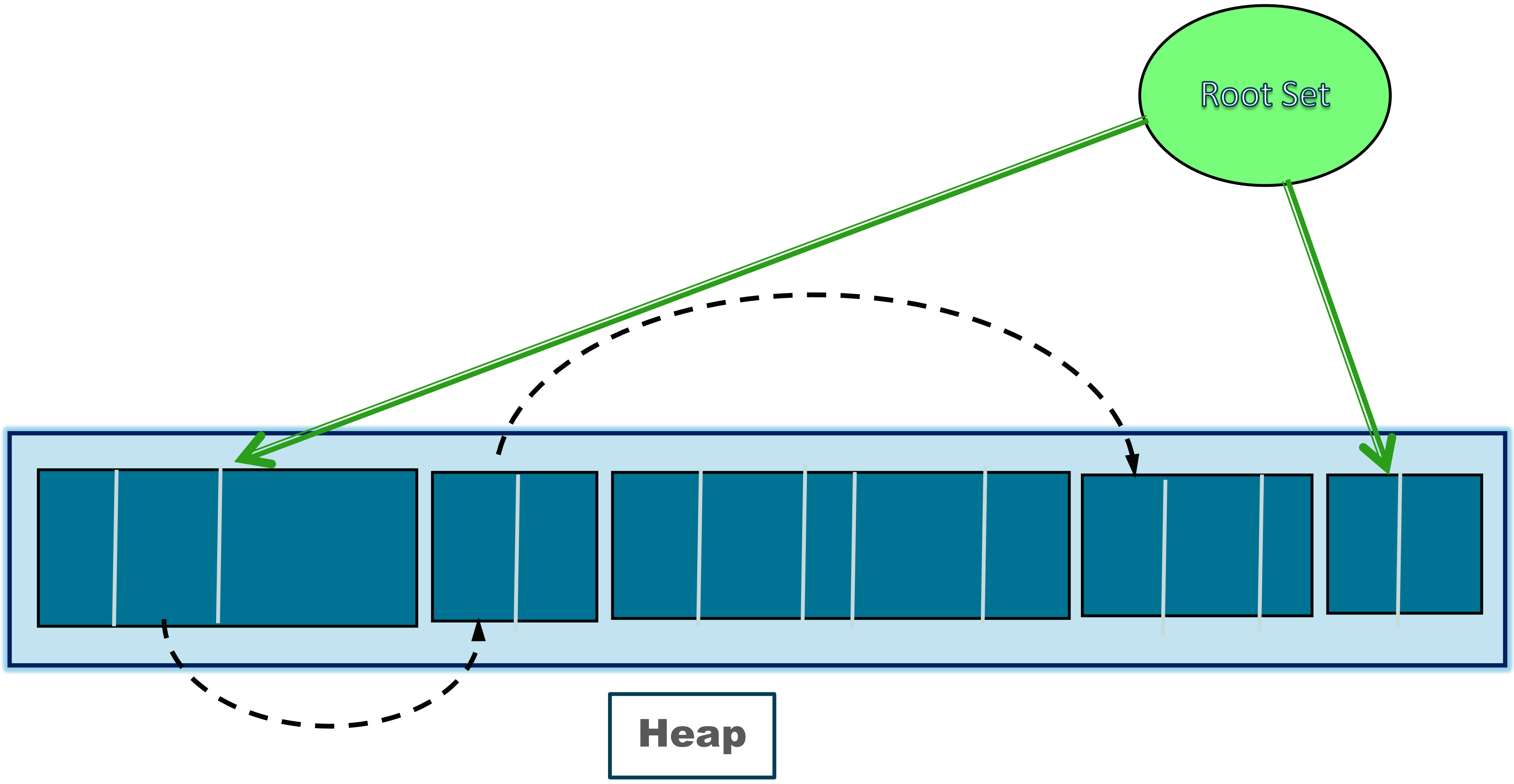
```
//allocate memory for String object
String s = new String("Hello World");

//no need to explicitly free memory occupied by 's'. It would be released by the
  garbage collector when 's' goes out of scope.
```

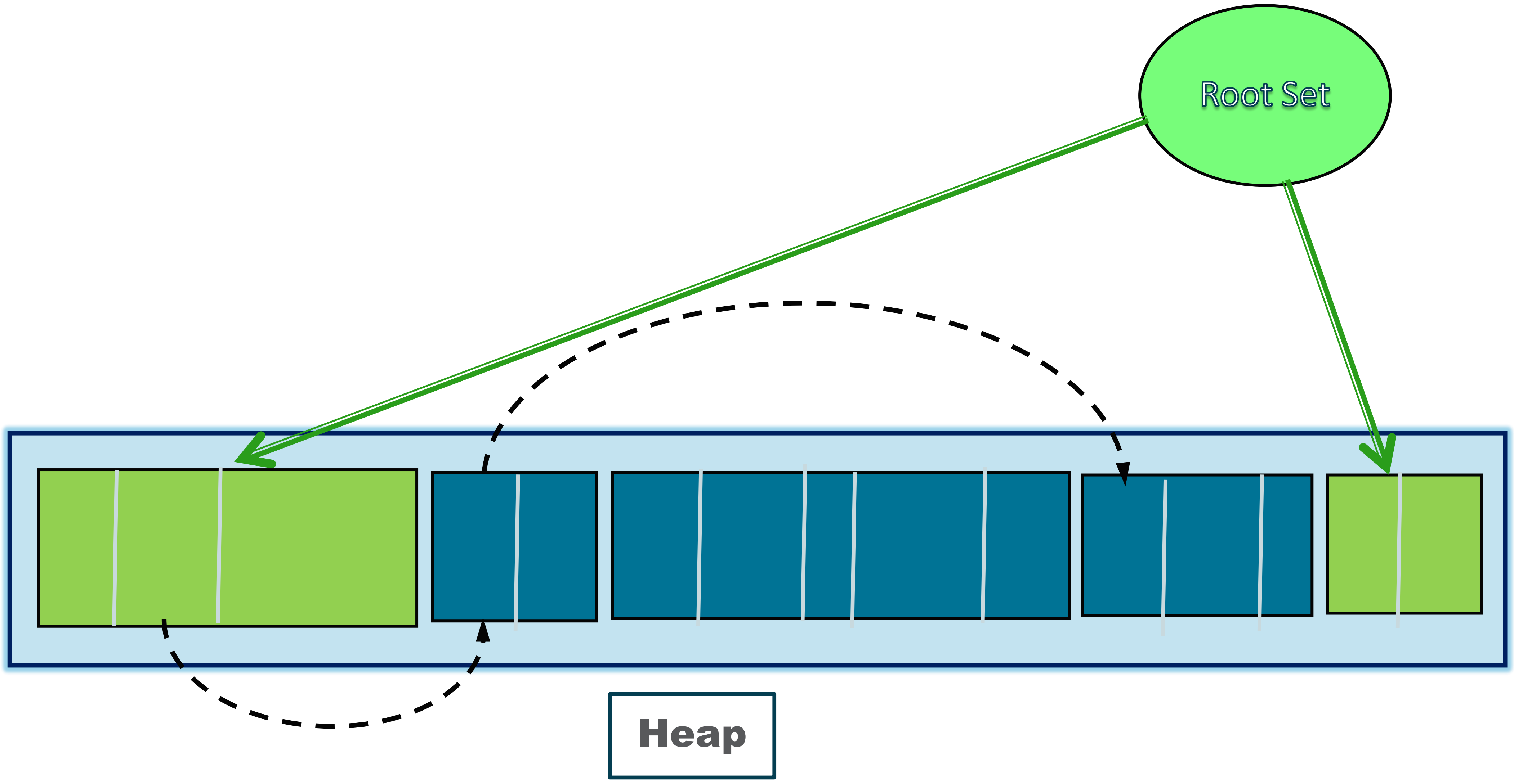

Object Allocations
Heap

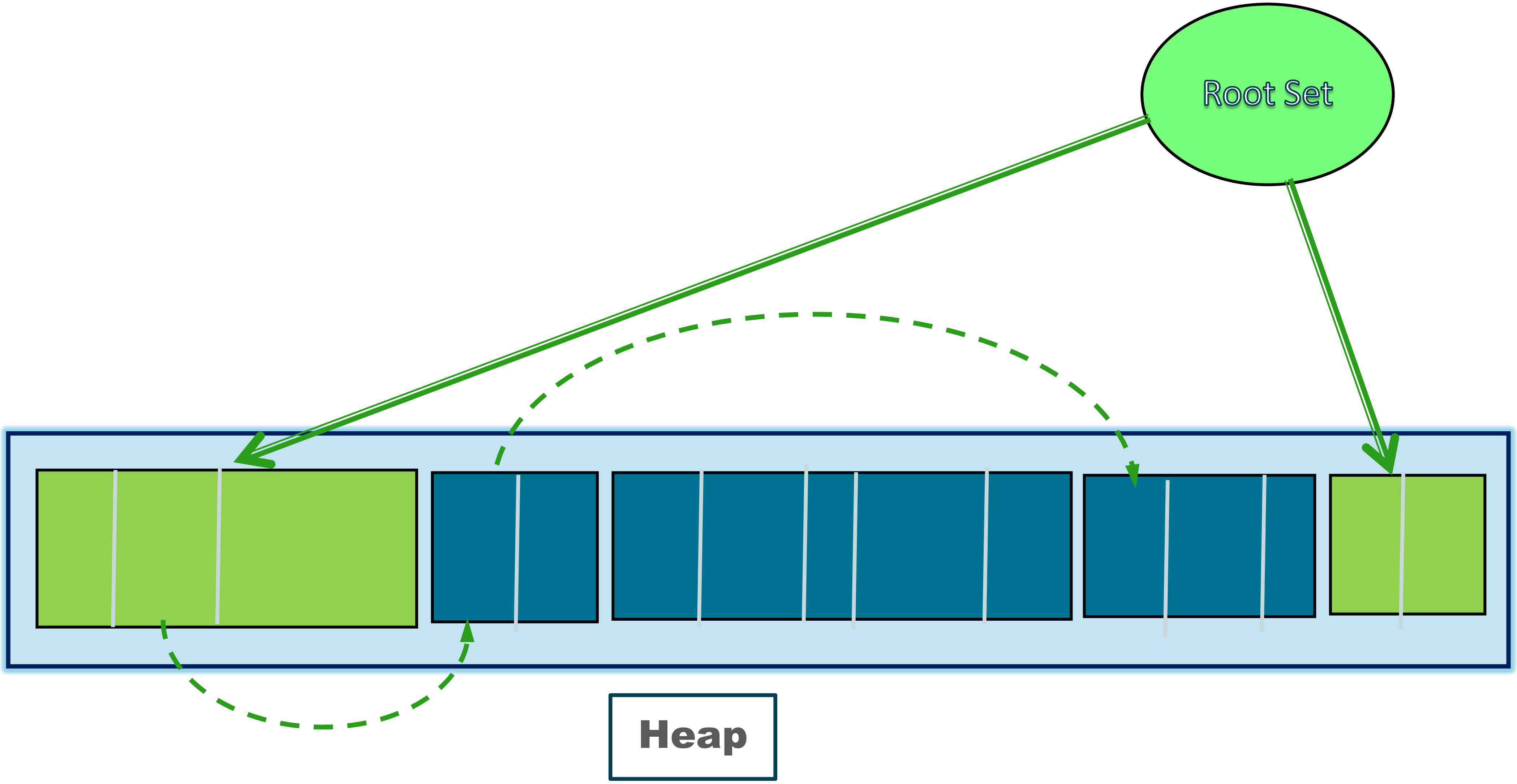

Object Allocations
Heap

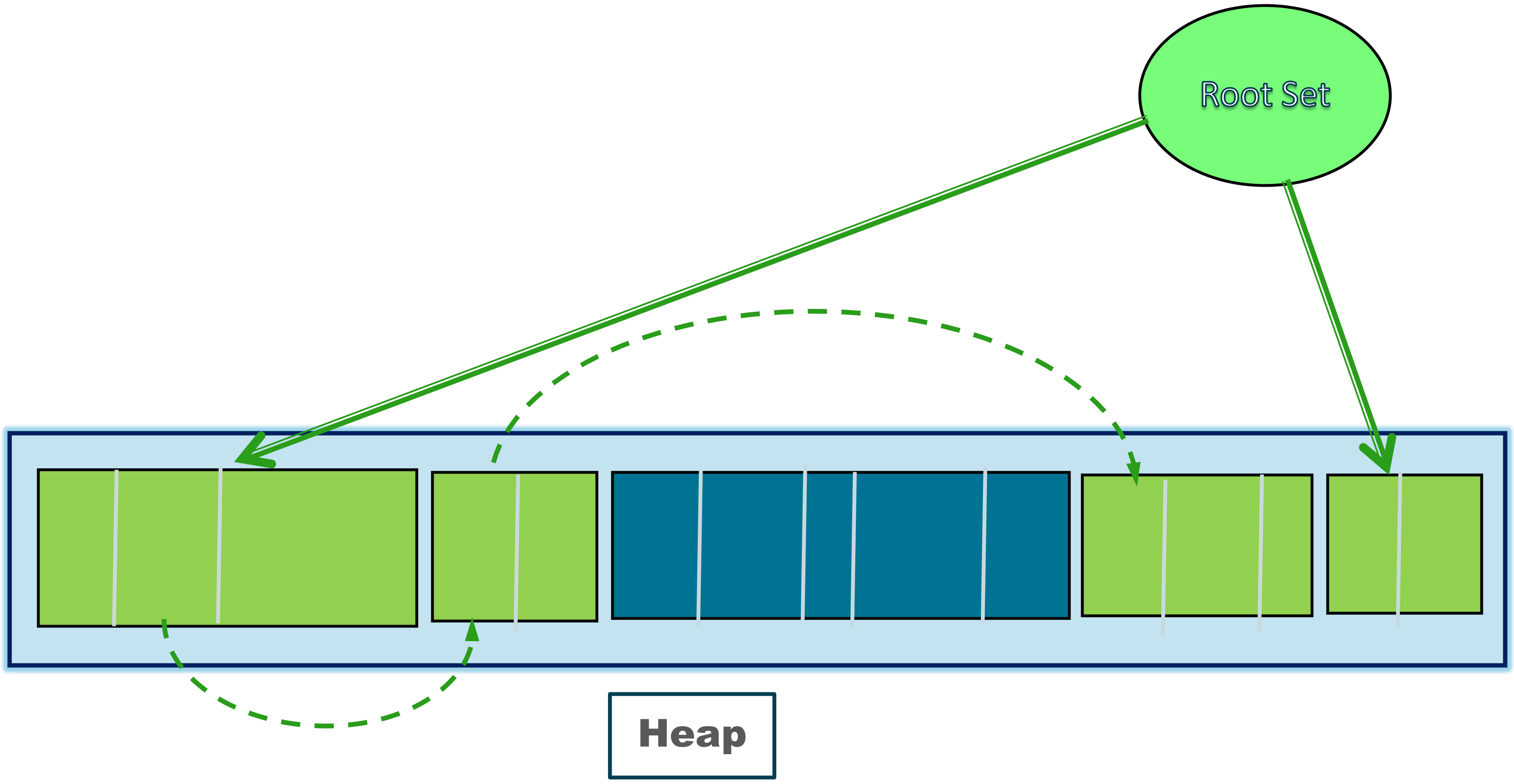

Fields
Heap


References
Heap

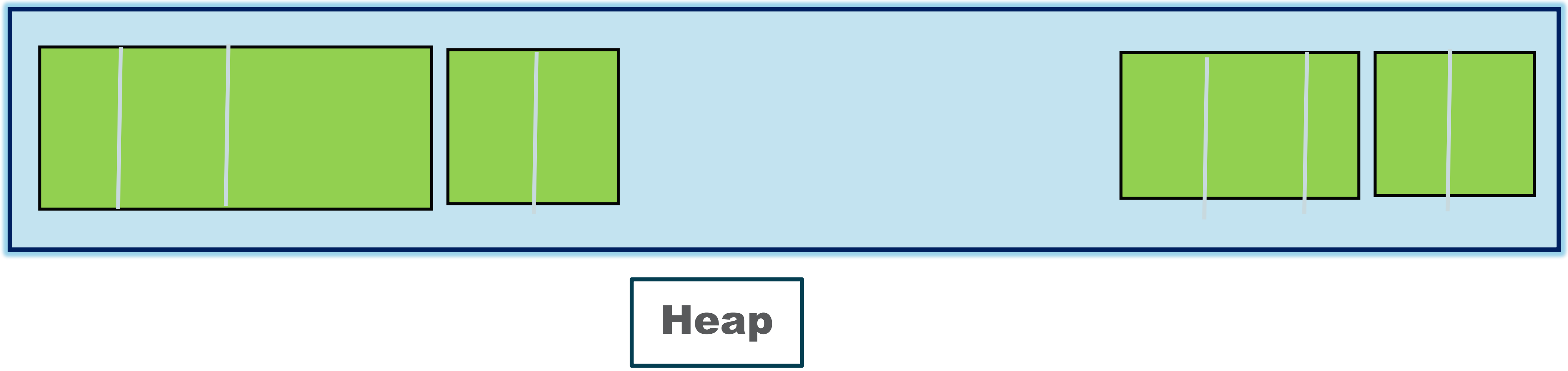

Root Set
Heap


Root Set
Heap

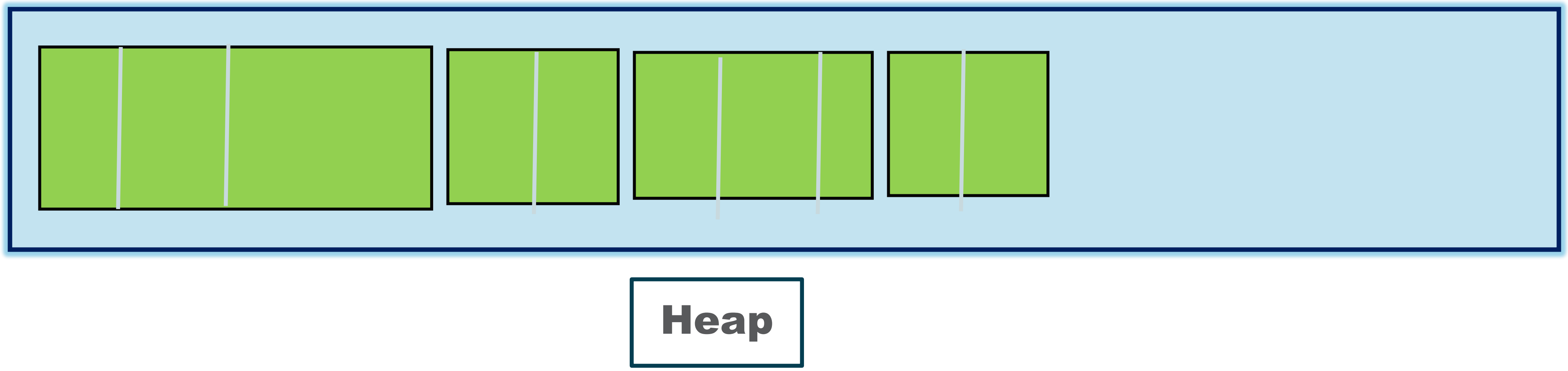

Root Set
Heap


Root Set
Heap

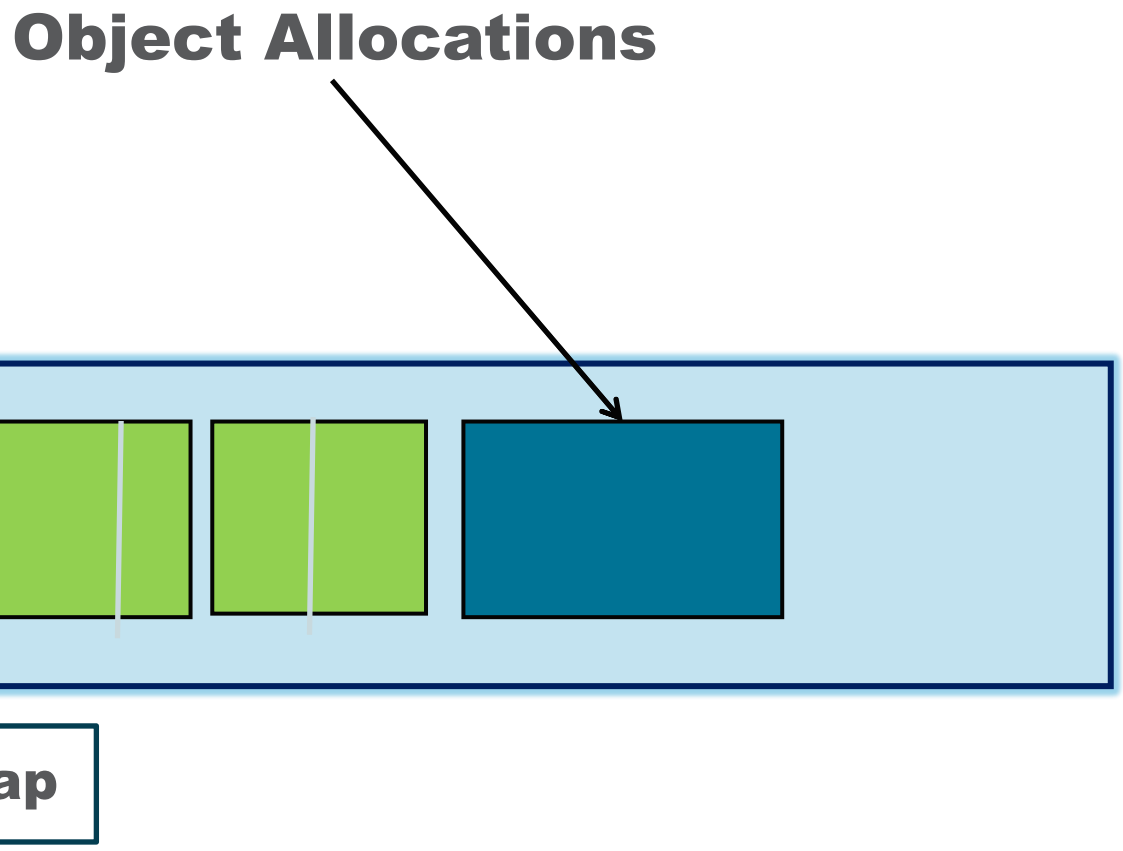
Collect Garbage



Compact



Allocations Continue



Garbage Collector

- Garbage collector is responsible for
  - Memory allocation
  - Keeping the referenced objects in memory
  - Reclaiming the space used by unreachable objects
- Unreachable objects are called garbage
- This whole process of reclaiming memory is garbage collection

Summary: Section 1

- HotSpot JVM uses automatic memory management
- Memory is automatically managed by a sub-system called the garbage collector
- Garbage collector is responsible for
  - Allocations
  - Keeping the alive objects in memory
  - Reclaiming the space used by unreachable objects

Lesson 1-2

Generational Garbage Collection and Memory Spaces in the HotSpot JVM

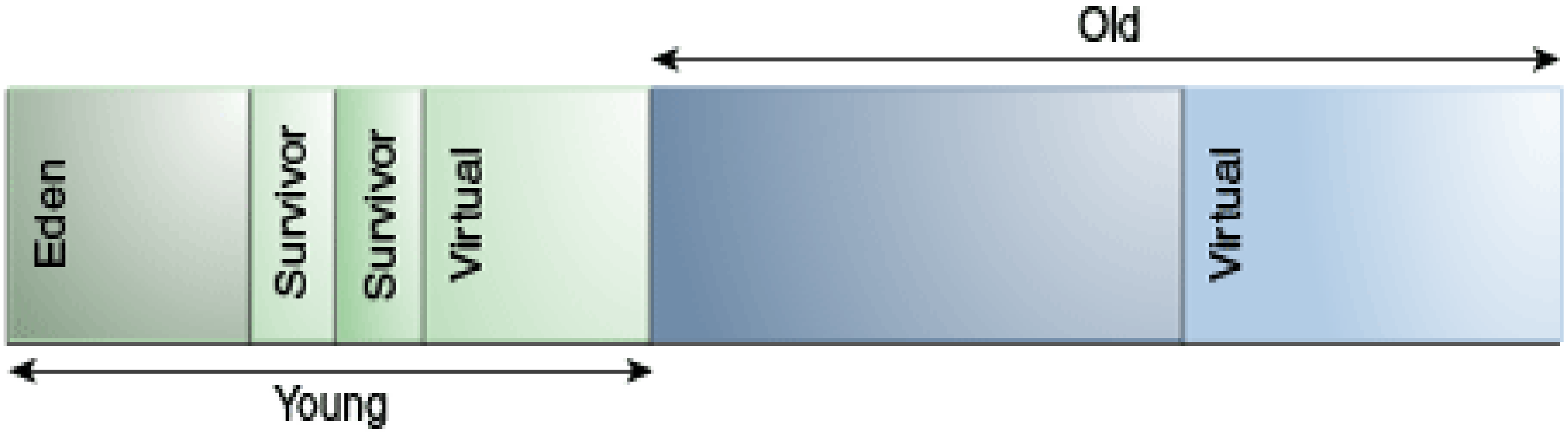
Poonam Parhar
JVM Sustaining Engineer
Oracle

Java
Your
Next
(Cloud)

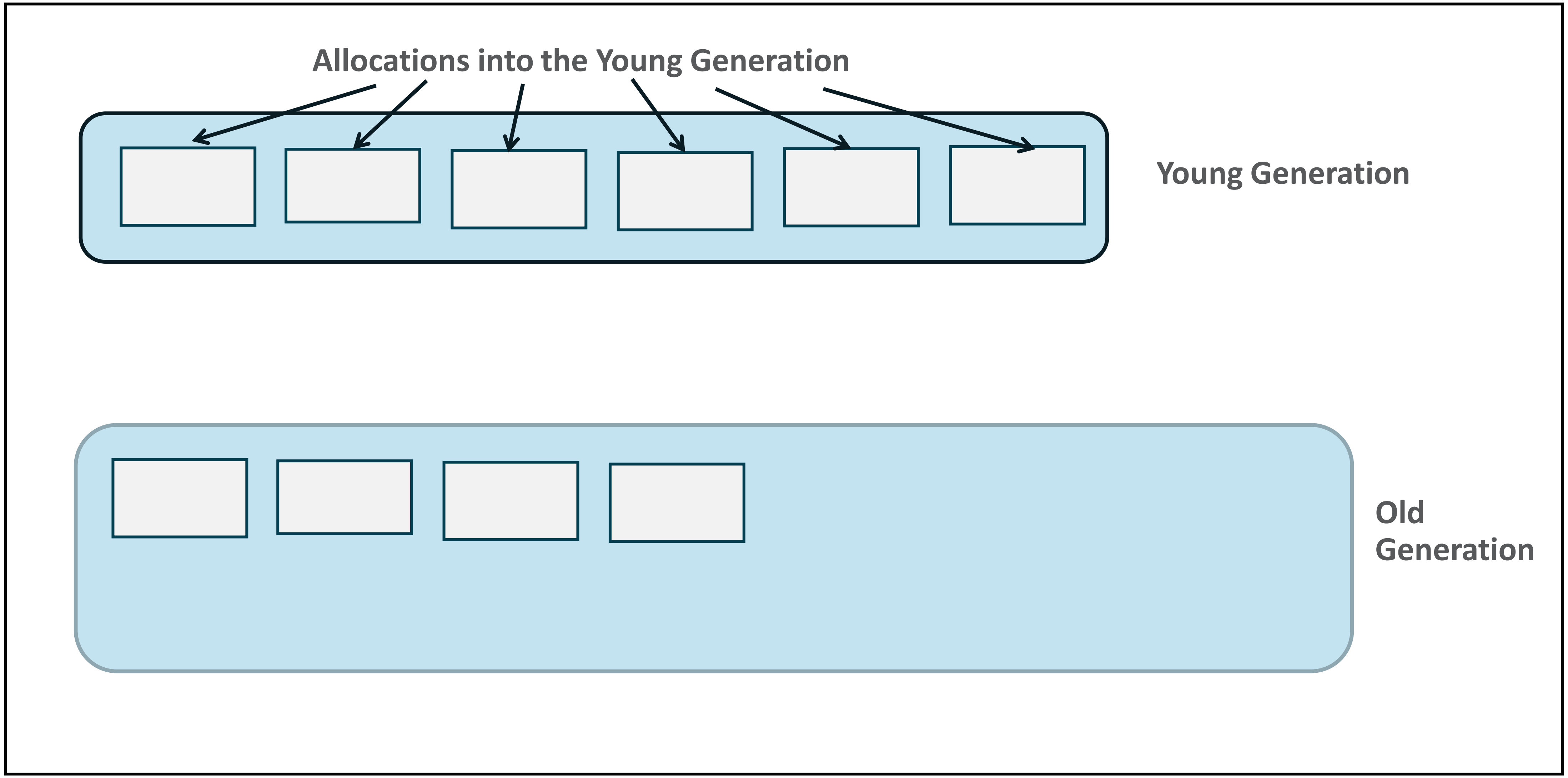
Generational Garbage Collection

- Memory space is divided into generations
- Separate pools holding objects of different age ranges
- Based on hypothesis:
  - Most allocated objects die young
  - Few references from older to younger objects exist
- To take advantage of this hypothesis, heap is divided into two generations
  - Young: small and collected frequently
  - Old : larger and occupancy grows slowly
- Minor(young) and Major(Full) collections

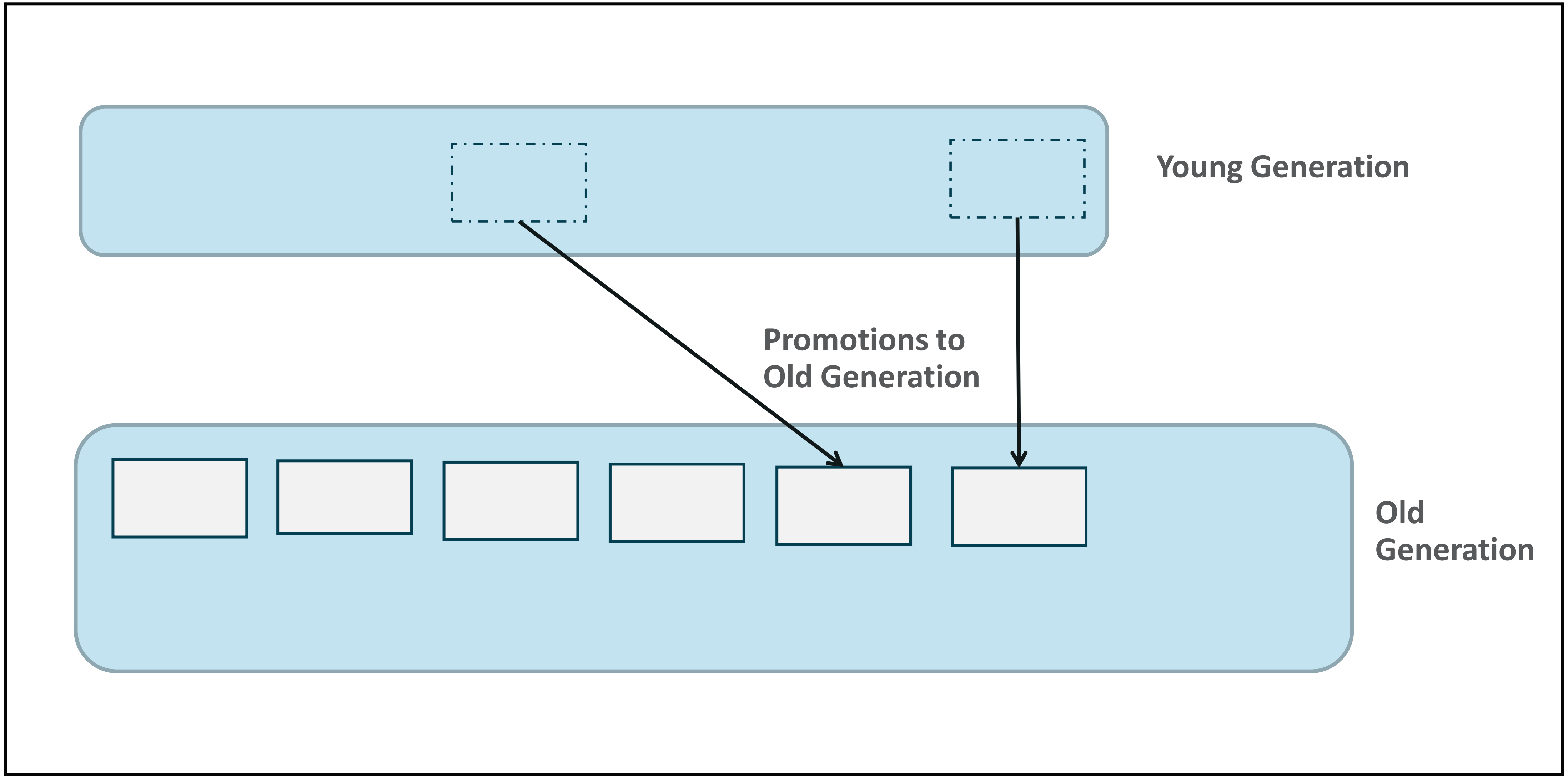
Young and Old Generation



Before Young Collection



After Young Collection



Permanent Generation

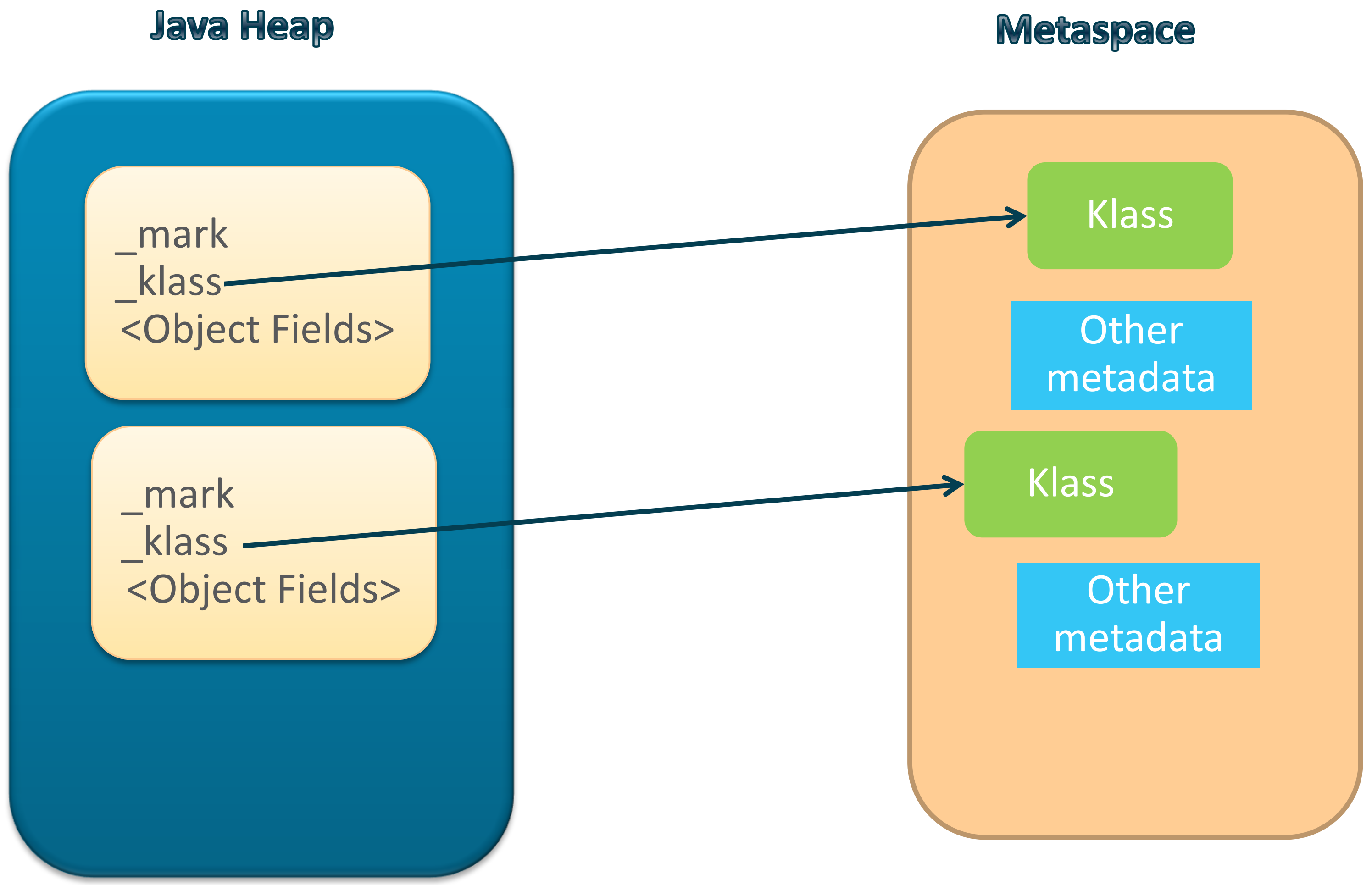
- HotSpot JVM prior to JDK 8 had a third generation called Permanent Generation
- Used for:
  - JVM internal representation of classes and their metadata
  - Class statics
  - Interned strings
- Contiguous with the Java Heap

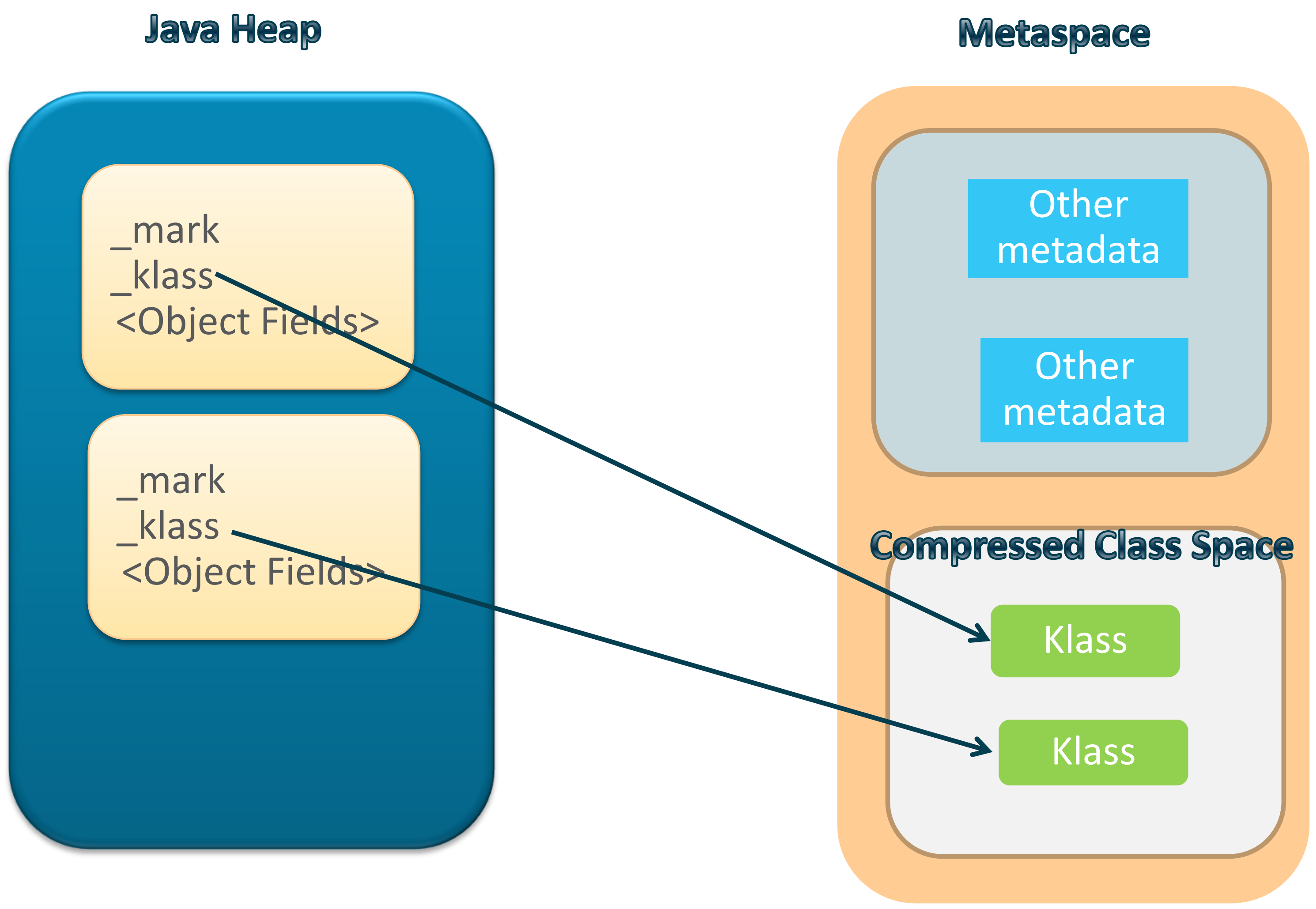
Metaspace

- JDK 8 does not have Permanent Generation
- Class metadata is stored in a new space called Metaspace
- Not contiguous with the Java Heap
- Metaspace is allocated out of native memory
- Maximum space available to the Metaspace is the available system memory
- This can though be limited by **MaxMetaspaceSize** JVM option

Compressed Class Space

- If UseCompressedClassesPointers is enabled then two separate areas of memory are used for the classes and its metadata
  - Metaspace
  - Compressed class space
- 64-bit class pointers are represented with 32-bit offsets
- Class metadata referenced by the 32-bit offsets is stored in the Compressed Class Space
- By default compressed class space is sized at 1GB
- **MaxMetaspaceSize** sets an upper limit on the committed size of both of these spaces.


Java Heap
Metaspace
_mark
_klass
<Object Fields>
_mark
_klass
<Object Fields>
Klass
Other metadata
Klass
Other metadata

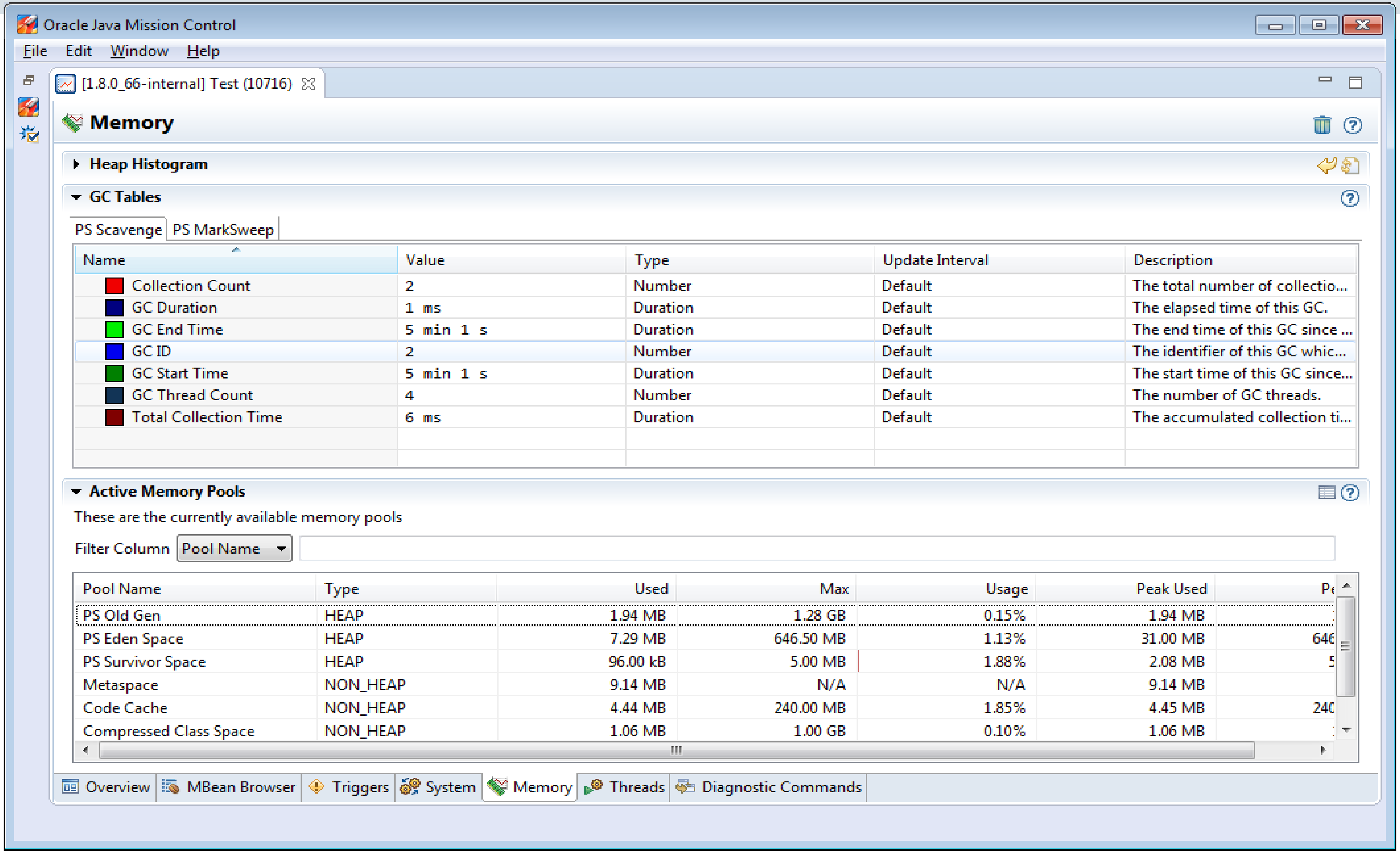

Java Heap
_mark
_klass
<Object Fields>
_mark
_klass
<Object Fields>
Metaspace
Other metadata
Other metadata
Compressed Class Space
Klass
Klass

Code Cache

- Code Cache is used to store the compiled code generated by the Just-in-time compilers
- It is allocated out of native memory
- Managed by the Code Cache Sweeper

Native Memory

- Available system memory
- Not managed by the JVM memory management



Memory

Heap Histogram

GC Tables

Name	Value	Type	Update Interval	Description
🟥 Collection Count	2	Number	Default	The total number of collectio...
🟦 GC Duration	1 ms	Duration	Default	The elapsed time of this GC.
🟩 GC End Time	5 min 1 s	Duration	Default	The end time of this GC since ...
🟦 GC ID	2	Number	Default	The identifier of this GC whic...
🟩 GC Start Time	5 min 1 s	Duration	Default	The start time of this GC since...
🟦 GC Thread Count	4	Number	Default	The number of GC threads.
🟥 Total Collection Time	6 ms	Duration	Default	The accumulated collection ti...

Active Memory Pools

These are the currently available memory pools

Filter Column: Pool Name

Pool Name	Type	Used	Max	Usage	Peak Used	Pe
PS Old Gen	HEAP	1.94 MB	1.28 GB	0.15%	1.94 MB	
PS Eden Space	HEAP	7.29 MB	646.50 MB	1.13%	31.00 MB	646
PS Survivor Space	HEAP	96.00 kB	5.00 MB	1.88%	2.08 MB	5
Metaspace	NON_HEAP	9.14 MB	N/A	N/A	9.14 MB	
Code Cache	NON_HEAP	4.44 MB	240.00 MB	1.85%	4.45 MB	240
Compressed Class Space	NON_HEAP	1.06 MB	1.00 GB	0.10%	1.06 MB	

Summary: Section 2

- Memory space is divided into memory pools
- Java Heap
  - Young generation
  - Old generation
- Classes and metadata space
  - Permanent Generation (before JDK 8)
  - Metaspace (metaspace + compressed class space) (JDK 8 onwards)
- CodeCache
- Native Memory

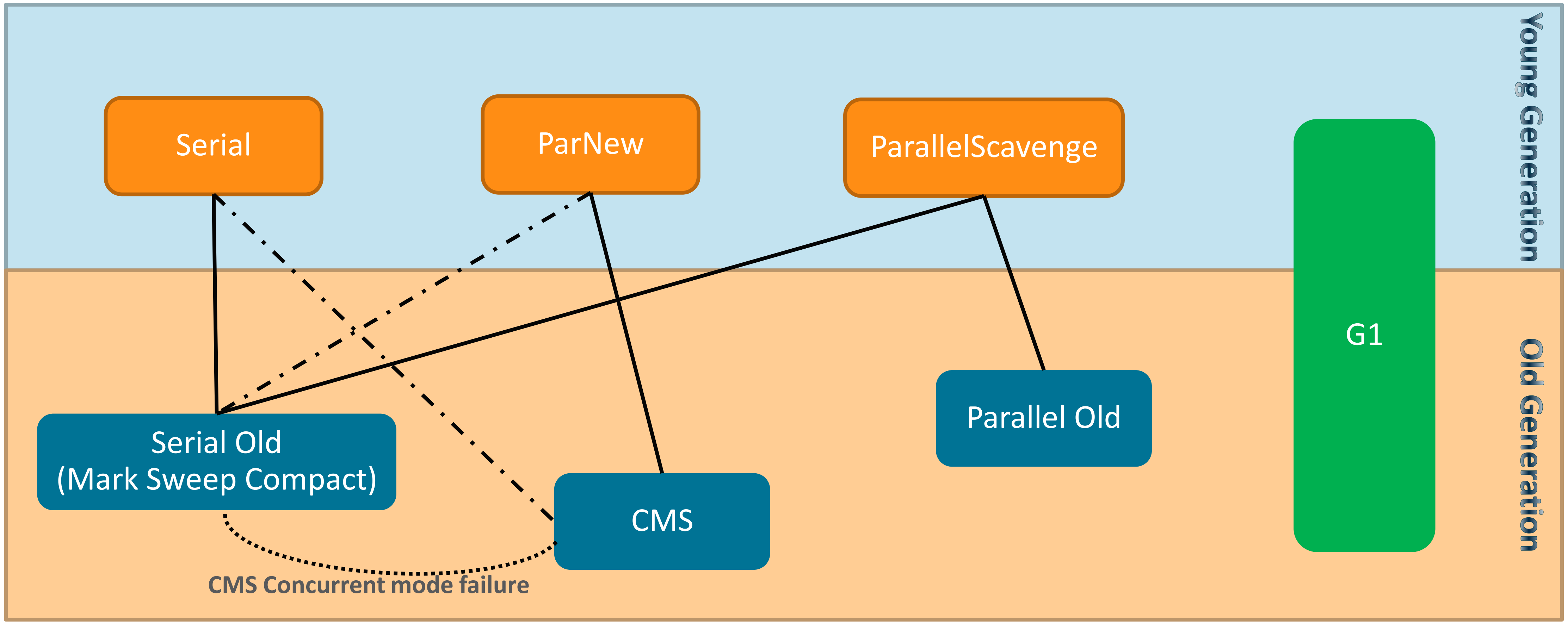
Lesson 1-3

Garbage Collectors in the HotSpot JVM

Poonam Parhar
JVM Sustaining Engineer
Oracle

Java
Your
Next
(Cloud)

HotSpot Garbage Collectors



- · - · - Combination not available in 9

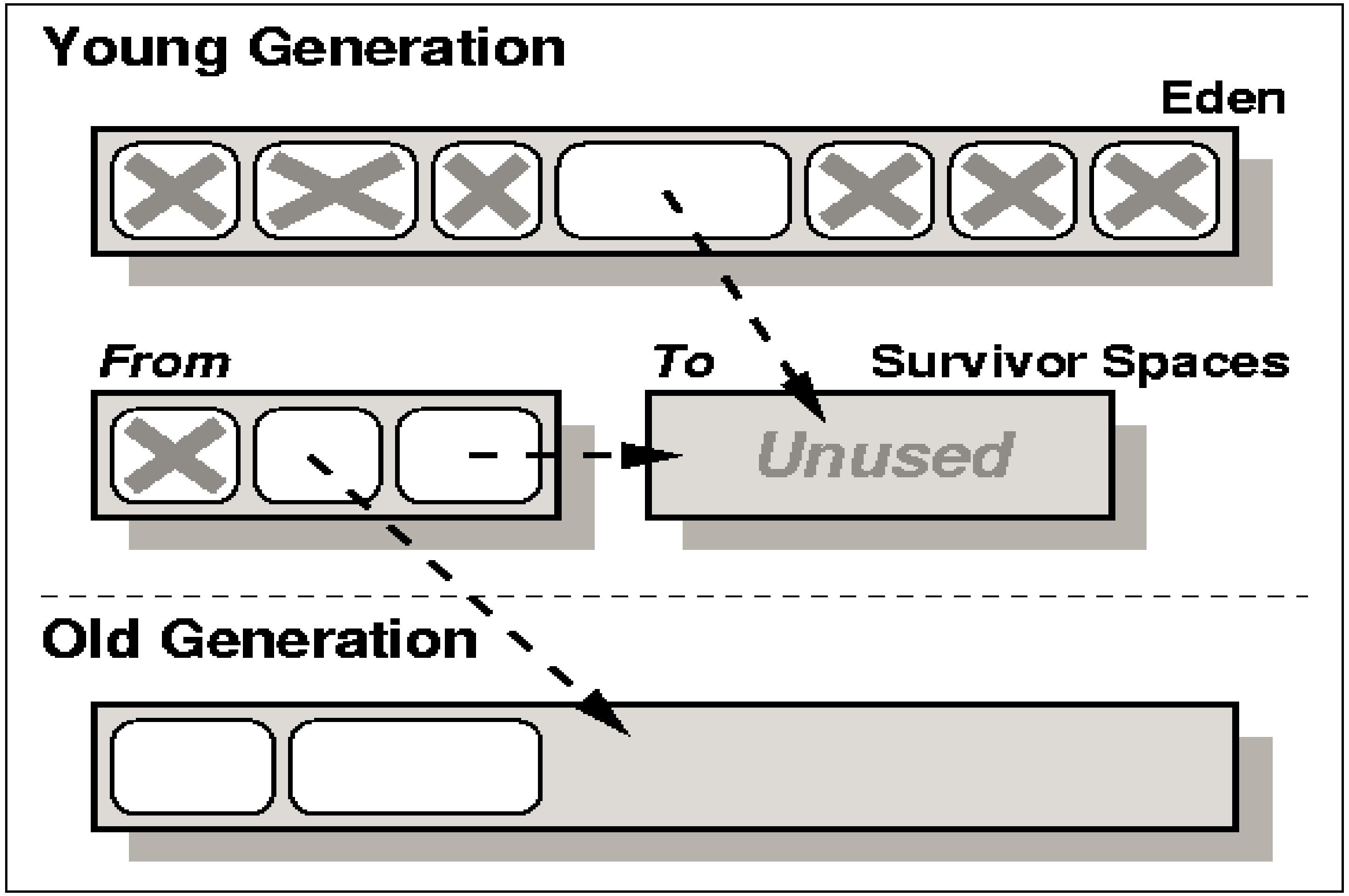
HotSpot Garbage Collection Types

- **Young Generation Collection**
  - **Serial** is a stop-the-world, copying collector that uses a single GC thread
  - **ParNew** is a stop-the-world, copying collector that uses multiple GC threads
  - **Parallel Scavenge** is a stop-the-world, copying collector that uses multiple GC threads
- **Old Generation Collection**
  - **Serial Old** is a stop-the-world, mark-sweep-compact collector that uses a single GC thread
  - **CMS** is a mostly concurrent, low-pause collector
  - **Parallel Old** is a compacting collector that uses multiple GC threads
- **G1** is the Garbage First collector for large heaps and provides reliable short GC pauses
  - Has generations but uses different memory layout

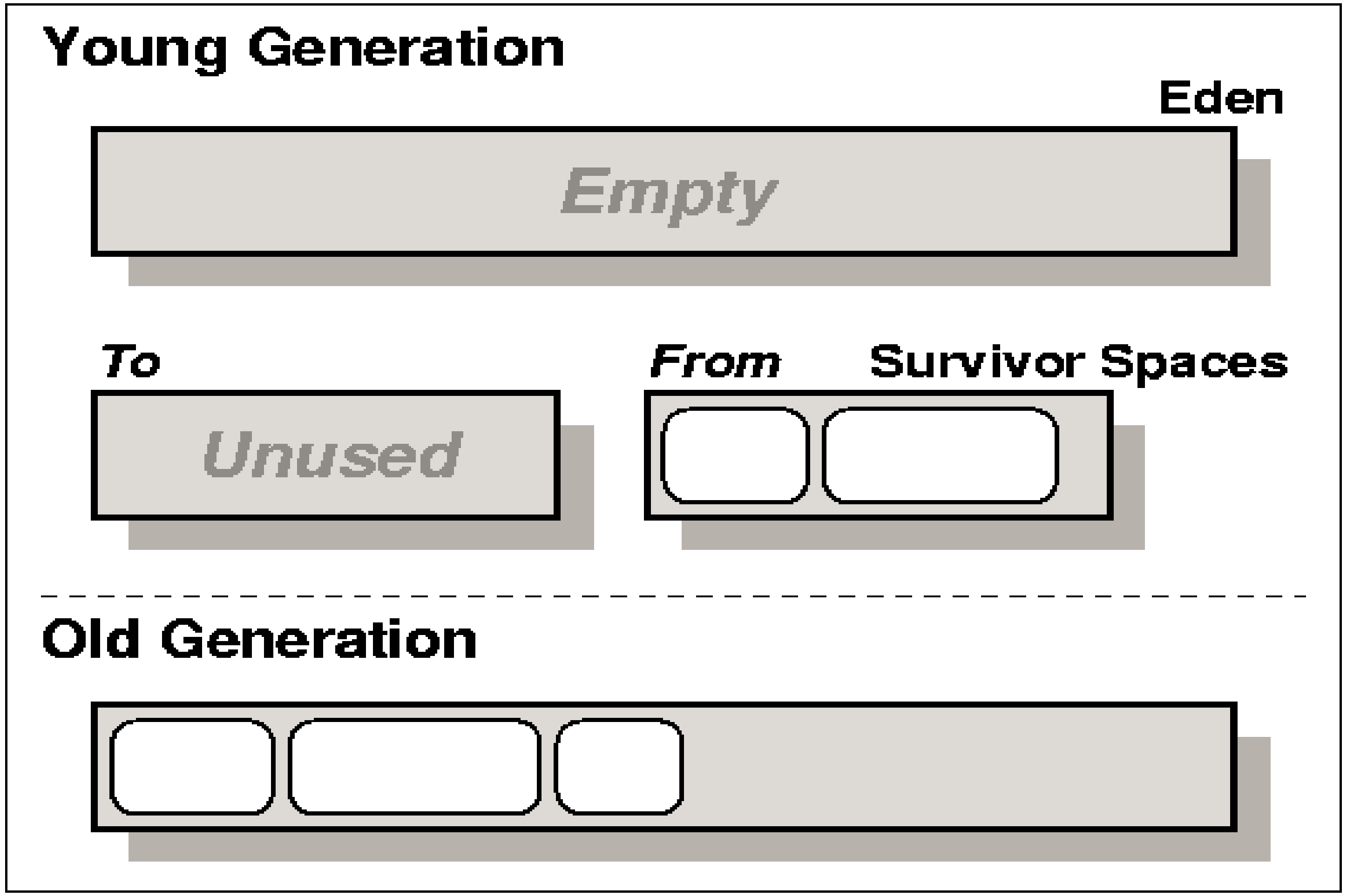
How to enable the Collectors

- UseSerialGC : Serial + Serial Old
- UseParNewGC : ParNew + Serial Old
- UseConcMarkSweepGC : ParNew + CMS + Serial Old. CMS is used most of the time to collect the old generation. Serial Old is used when a concurrent mode failure occurs.
- UseParallelGC : Parallel Scavenge + Parallel Old
- UseG1GC : G1 GC for both generations

Young Collections



Young Collections



Mark-Sweep-Compact Collector

- -XX:+UseSerialGC or –XX:+UseParNewGC selects Serial Old collector for old generation
- Stop-the-world old generation collector.
- Old is collected using mark-sweep-compact.
- Mark phase: marks all live objects.
- Sweep phase: sweeps over the heap identifying garbage.
- Slide phase: the GC performs a sliding compaction, sliding the live objects towards the start of the Heap.

Parallel Collector

- *-XX:+UseParallelGC*
  - Young Generation collected with Parallel Scavenge
  - Old Generation collected with Parallel collector
- It is also called the throughput collector
- Stop-the-world collections
- Default on Server type machines up until JDK 9
- Collection is performed in parallel on multiple cores

Concurrent Mark Sweep Collector

- *-XX:+UseConcMarkSweepGC*
  - Young gen – ParNew collector
  - Old gen – CMS collector
- Low-latency collector, Mostly concurrent
- No heap compaction – fragmentation
- Free lists link unallocated regions
- Allocations expensive as compared to bump-the-pointer allocations
- Additional overhead on young collections
- Larger heap size requirement and floating garbage
- Deprecated in Java 9

G1 Collector

- Server-style garbage collector, targeted for multi-processor machines with large memories
- Meets garbage collection (GC) pause time goals with a high probability, while achieving high throughput.
- Better GC ergonomics
- Low pauses without fragmentation
- Parallelism and concurrency in collections
- G1 is a compacting collector.
- Fully supported in Oracle JDK 7 update 4 and later releases
- Default collector in JDK 9

Summary: Section 3

- Serial Collector
- Parallel Collector – throughput collector
- Low Pause Collectors
  - G1 Garbage Collector – default in Java 9
  - CMS – deprecated in Java 9

References

- GC Tuning Guide
  - https://docs.oracle.com/javase/9/gctuning/toc.htm
- Books
  - Garbage Collection: Algorithms for Automatic Dynamic Memory Management. Wiley, Chichester, July 1996. With a chapter on Distributed Garabge Collection by R. Lins. Richard Jones, Antony Hosking, and Elliot Moss.
  - The Garbage Collection Handbook: The Art of Automatic Memory Management. CRC Applied Algorithms and Data Structures. Chapman & Hall, January 2012

Lesson 1-4

Garbage Collection Changes Between Java 7, 8 and 9

Poonam Parhar
JVM Sustaining Engineer
Oracle

Java Your Next (Cloud)

JDK 7 Changes

Permanent Generation Partially Removed

- In JDK 7, some things were moved out of PermGen
  - Symbols were moved to native memory
  - Interned strings were moved to the Java Heap
  - Class statics were moved to the Java Heap
- Might contribute to slight increase in the young collections pause times
- Can revert this change with –XX:+JavaObjectsInPerm

JDK 8 Changes

Permanent Generation Removed

- Permanent Generation has been removed in JDK 8
- PermGen tuning options are now obsolete in JDK 8

MetaSpace in JDK 8

- Metaspace added in JDK 8
- Classes and metadata stored in the Metaspace
- It replaces PermGen
- New Metaspace tuning JVM options added
  - MetaspaceSize
  - MaxMetaspaceSize

Class Unloading Related Changes

- Until JDK 8u40, G1 unloads classes only at Full GCs
  - Classes don’t get unloaded during the young or mixed collections and a Full GC is required to unload classes
- G1 supports concurrent unloading of classes from JDK 8

JDK 9 Changes

G1 Garbage Collector

- JDK 9 makes the G1 Garbage collector the default garbage collector
- It is surely possible to explicitly pick the garbage collector of your choosing
- For example:

  -XX:+UseParallelGC would select Parallel GC as the garbage collector

CMS Collector

- JDK 9 deprecates the CMS garbage collector
- Warning message is issued when -XX:+UseConcMarkSweepGC is used
- Planned for removal in a future major release

GC Combinations Removed in Java 9

- DefNew + CMS : -XX:-UseParNewGC -XX:+UseConcMarkSweepGC
- ParNew + SerialOld : -XX:+UseParNewGC
- ParNew + iCMS : -Xincgc
- ParNew + iCMS : -XX:+CMSIncrementalMode -XX:+UseConcMarkSweepGC
- DefNew + iCMS : -XX:+CMSIncrementalMode -XX:+UseConcMarkSweepGC -XX:-UseParNewGC
- CMS foreground : -XX:+UseCMSCompactAtFullCollection
- CMS foreground : -XX:+CMSFullGCsBeforeCompaction
- CMS foreground : -XX:+UseCMSCollectionPassing

Java™

ORACLE®